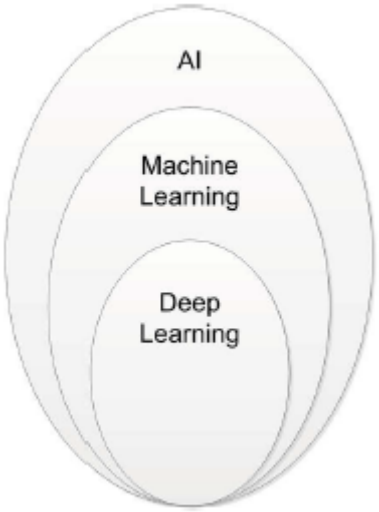
Przedstawiamy głębokie uczenie się

Prawdopodobnie wiele słyszałeś o głębokim uczeniu się. Termin pojawia się wszędzie i wydaje się mieć zastosowanie do wszystkiego. W rzeczywistości uczenie głębokie jest podzbiorem uczenia maszynowego, które z kolei jest podzbiorem sztucznej inteligencji (AI). Pierwszym celem jest pomoc w zrozumieniu, na czym naprawdę polega głębokie uczenie się i jak ma ono zastosowanie w dzisiejszym świecie. Możesz być zaskoczony, gdy dowiesz się, że głębokie uczenie się nie jest jedyną grą w mieście; istnieją inne metody analizy danych. W rzeczywistości głębokie uczenie się spełnia określony zestaw potrzeb, jeśli chodzi o analizę danych, więc możesz używać innych metod i nawet o tym nie wiedzieć. Uczenie głębokie to tylko podzbiór sztucznej inteligencji, ale jest to ważny podzbiór. Widzisz techniki głębokiego uczenia się używane do wielu zadań, ale niezbyt zadań. W rzeczywistości niektórzy ludzie kojarzą głębokie uczenie się z zadaniami, których nie są w stanie wykonać. Następnym krokiem w odkrywaniu uczenia głębokiego jest zrozumienie, co może, a czego nie może dla Ciebie zrobić. W ramach pracy z głębokim uczeniem napiszesz aplikacje, które wykorzystują głębokie uczenie do przetwarzania danych, a następnie generują pożądane wyniki. Oczywiście musisz trochę wiedzieć o środowisku programistycznym, zanim będziesz mógł wiele zrobić. Tak, uczenie głębokie może wykonywać niesamowite zadania, gdy jest odpowiednio używane, ale może również powodować poważne problemy w przypadku problemów, których nie obsługuje dobrze. Czasami trzeba spojrzeć na inne technologie, aby wykonać dane zadanie, lub dowiedzieć się, których technologii użyć z głębokim uczeniem, aby zapewnić bardziej wydajne i eleganckie rozwiązanie określonych problemów.

Zdefiniowanie, co oznacza głębokie uczenie się

Zrozumienie głębokiego uczenia się zaczyna się od precyzyjnej definicji terminów. W przeciwnym razie trudno będzie oddzielić szum medialny od realiów tego, co faktycznie może zapewnić głębokie uczenie się. Uczenie głębokie jest częścią zarówno sztucznej inteligencji, jak i uczenia maszynowego, jak pokazano na rysunku .



Aby zrozumieć głębokie uczenie się, musisz zacząć od zewnątrz - to znaczy zacząć od sztucznej inteligencji, a następnie przejść przez uczenie maszynowe, a na końcu zdefiniować głębokie uczenie się. Poniższe sekcje pomogą Ci przejść przez ten proces.

Począwszy od sztucznej inteligencji

Stwierdzenie, że sztuczna inteligencja jest sztuczną inteligencją, tak naprawdę nie mówi nic znaczącego, dlatego tak wiele dyskusji i nieporozumień pojawia się w związku z tym terminem. Tak, możesz argumentować, że to, co się dzieje, jest sztuczne i nie pochodzi z naturalnego źródła. Jednak część dotycząca inteligencji jest w najlepszym przypadku niejednoznaczna. Ludzie definiują inteligencję na wiele różnych sposobów. Możesz jednak powiedzieć, że inteligencja obejmuje pewne ćwiczenia umysłowe składające się z następujących czynności:

Uczenie się: posiadanie umiejętności pozyskiwania i przetwarzania nowych informacji.

Rozumowanie: umiejętność manipulowania informacjami na różne sposoby.

Zrozumienie: rozważenie wyniku manipulacji informacją.

Chwytanie prawd: ustalanie ważności zmanipulowanych informacji.

Dostrzeganie relacji: rozróżnianie interakcji zweryfikowanych danych z innymi danymi.

Rozważanie znaczeń: Stosowanie prawd do określonych sytuacji w sposób zgodny z ich relacją.

Oddzielenie faktów od przekonań: ustalenie, czy dane są odpowiednio poparte udowodnionymi źródłami, w przypadku których można wykazać, że są konsekwentnie prawidłowe.

Lista mogłaby być dość długa, ale nawet ta lista jest stosunkowo podatna na interpretację przez każdego, kto zaakceptuje ją jako wykonalną. Jak widać z listy, inteligencja często jednak podąża za procesem, który system komputerowy może naśladować w ramach symulacji:

1. Wyznacz cel w oparciu o potrzeby lub pragnienia.

2. Oceń wartość wszelkich obecnie znanych informacji wspierających cel.

3. Zbierz dodatkowe informacje, które mogą wesprzeć cel.

4. Manipuluj danymi tak, aby uzyskały formę zgodną z istniejącymi informacjami.

5. Zdefiniuj relacje i wartości prawdy między istniejącymi a nowymi Informacjami.

6. Określ, czy cel został osiągnięty.

7. Zmodyfikuj cel w świetle nowych danych i jego wpływu na prawdopodobieństwo sukcesu.

8. W razie potrzeby powtarzaj kroki od 2 do 7, aż cel zostanie osiągnięty (uznany za prawdziwy) lub wyczerpią się możliwości jego osiągnięcia (uznany za fałszywy).

Mimo że można tworzyć algorytmy i zapewniać dostęp do danych wspierających ten proces w komputerze, zdolność komputera do uzyskiwania inteligencji jest poważnie ograniczona. Na przykład komputer nie jest w stanie niczego zrozumieć, ponieważ opiera się na procesach maszynowych do manipulowania danymi przy użyciu czystej matematyki w sposób ściśle mechaniczny. Podobnie komputery nie mogą łatwo oddzielić prawdy od nieprawdy. W rzeczywistości żaden komputer nie jest w stanie w pełni zrealizować żadnej z czynności umysłowych opisanych na liście opisującej inteligencję. Myśląc o sztucznej inteligencji, musisz wziąć pod uwagę cele ludzi, którzy ją rozwijają. Celem jest naśladowanie ludzkiej inteligencji, a nie jej powielanie. Komputer tak naprawdę nie myśli, ale sprawia wrażenie, jakby myślał. Jednak komputer faktycznie zapewnia ten wygląd tylko w logicznej / matematycznej formie inteligencji. Komputer z umiarkowanym powodzeniem naśladuje wzrokowo-przestrzenną inteligencję i kinestetyczną inteligencję ciała. Komputer ma niską, zadowalającą zdolność

inteligencji interpersonalnej i językowej. Jednak w przeciwieństwie do ludzi komputer nie ma możliwości naśladowania inteligencji intrapersonalnej lub twórczej.

Biorąc pod uwagę rolę AI

Jak opisano w poprzedniej sekcji, pierwszą koncepcją, którą należy zrozumieć, jest to, że sztuczna inteligencja nie ma nic wspólnego z ludzką inteligencją. Tak, część sztucznej inteligencji jest modelowana tak, aby symulować ludzką inteligencję, ale tym właśnie jest: symulacja. Myśląc o sztucznej inteligencji, zwróć uwagę, że istnieje zależność między poszukiwaniem celu, przetwarzaniem danych wykorzystywanym do osiągnięcia tego celu oraz pozyskiwaniem danych wykorzystywanym do lepszego zrozumienia celu. Sztuczna inteligencja polega na algorytmach, aby osiągnąć wynik, który może, ale nie musi mieć nic wspólnego z ludzkimi celami lub metodami ich osiągania. Mając to na uwadze, możesz sklasyfikować sztuczną inteligencję na cztery sposoby:

* Ludzkie działanie: gdy komputer zachowuje się jak człowiek, najlepiej odzwierciedla to test Turinga, w którym komputer odnosi sukces, gdy nie jest możliwe rozróżnienie między komputerem a człowiekiem. Ta kategoria odzwierciedla również to, co media sugerują, że myślisz o sztucznej inteligencji. Widzisz, że jest stosowany w technologiach, takich jak przetwarzanie języka naturalnego, reprezentacja wiedzy, automatyczne rozumowanie i uczenie maszynowe (z których wszystkie cztery muszą być obecne, aby zdać test). Oryginalny test Turinga nie obejmował żadnego kontaktu fizycznego. Nowszy, Total Turing Test obejmuje kontakt fizyczny w postaci przesłuchania zdolności percepcyjnych, co oznacza, że komputer musi również wykorzystywać widzenie komputerowe i robotykę, aby odnieść sukces. Nowoczesne techniki obejmują ideę osiągnięcia celu, a nie całkowite naśladowanie ludzi. Na przykład braciom Wright nie udało się stworzyć samolotu poprzez dokładne odwzorowanie lotów ptaków; raczej ptaki dostarczyły pomysłów, które doprowadziły do aerodynamiki, która z kolei ostatecznie doprowadziła do walki ludzi. Celem jest latanie. Zarówno ptaki, jak i ludzie osiągają ten cel, ale stosują różne podejścia.

* Myślenie po ludzku: Kiedy komputer myśli jak człowiek, wykonuje zadania, które wymagają od człowieka inteligencji (w przeciwieństwie do rutynowych procedur), aby odnieść sukces, na przykład prowadzenie samochodu. Aby określić, czy program myśli jak człowiek, musisz mieć jakąś metodę określania sposobu myślenia ludzi, którą definiuje podejście do modelowania poznawczego. Ten model opiera się na trzech technikach:

Introspekcja: wykrywanie i dokumentowanie technik używanych do osiągnięcia celów poprzez monitorowanie własnych procesów myślowych.

Testy psychologiczne: obserwowanie zachowania osoby i dodawanie go do bazy danych podobnych zachowań innych osób ze względu na podobny zestaw okoliczności, celów, zasobów i warunków środowiskowych (między innymi).

Obrazowanie mózgu: monitorowanie aktywności mózgu bezpośrednio za pomocą różnych środków mechanicznych, takich jak komputerowa tomografia osiowa (CAT), pozytonowa tomografia emisyjna (PET), obrazowanie metodą rezonansu magnetycznego (MRI) i magnetoencefalografia (MEG).

Po utworzeniu modelu możesz napisać program symulujący model. Biorąc pod uwagę różnorodność procesów myślowych ludzi i trudność w dokładnym przedstawieniu tych procesów myślowych jako części programu, wyniki są w najlepszym przypadku eksperymentalne. Ta kategoria ludzkiego myślenia jest często używana w psychologii i innych dziedzinach, w których niezbędne jest modelowanie procesu myślenia człowieka w celu stworzenia realistycznych symulacji.

* Myślenie racjonalne: badanie sposobu myślenia ludzi przy użyciu pewnych standardów umożliwia stworzenie wytycznych opisujących typowe ludzkie zachowania. Osoba jest uważana za racjonalną, gdy postępuje zgodnie z tymi zachowaniami w ramach pewnych poziomów odchylenia. Komputer, który myśli racjonalnie, opiera się na zarejestrowanych zachowaniach, aby stworzyć przewodnik dotyczący interakcji ze środowiskiem w oparciu o dostępne dane. Celem tego podejścia jest logiczne rozwiązywanie problemów, gdy jest to możliwe. W wielu przypadkach takie podejście umożliwiłoby stworzenie podstawowej techniki rozwiązania problemu, która byłaby następnie modyfikowana w celu rzeczywistego rozwiązania problemu. Innymi słowy, rozwiązanie problemu w zasadzie często różni się od rozwiązania go w praktyce, ale nadal potrzebujesz punktu wyjścia.

* Racjonalne działanie: badanie, jak ludzie zachowują się w określonych sytuacjach, przy określonych ograniczeniach, pozwala określić, które techniki są zarówno wydajne, jak i skuteczne. Komputer, który działa racjonalnie, polega na zarejestrowanych działaniach w celu interakcji ze środowiskiem w oparciu o warunki, czynniki środowiskowe i istniejące dane. Podobnie jak w przypadku racjonalnego myślenia, czyny racjonalne zależą od rozwiązania w zasadzie, które w praktyce może nie okazać się przydatne. Jednak racjonalne działania stanowią podstawę, na której komputer może rozpocząć negocjacje dotyczące pomyślnego zakończenia celu.

CZŁOWIEK A PROCES RACJONALNY

Procesy ludzkie różnią się wynikami od procesów racjonalnych. Proces jest racjonalny, jeśli zawsze działa właściwie na podstawie aktualnych informacji, mając idealną miarę wydajności. Krótko mówiąc, racjonalne procesy przebiegają według książki i zakładają, że „książka" jest rzeczywiście poprawna. Ludzkie procesy obejmują instynkt, intuicję i inne zmienne, które niekoniecznie odzwierciedlają książkę i mogą nawet nie uwzględniać istniejących danych. Na przykład racjonalnym sposobem prowadzenia samochodu jest zawsze przestrzeganie prawa. Jednak ruch nie jest racjonalny. Jeśli ściśle przestrzegasz przepisów, utkniesz gdzieś, ponieważ inni kierowcy nie przestrzegają ich dokładnie. Aby odnieść sukces, samochód jeżdżący samodzielnie musi więc działać bardziej po ludzku niż racjonalnie. Obecnie sztuczna inteligencja jest używana w wielu aplikacjach. Jedynym problemem jest to, że technologia działa tak dobrze, że nawet nie wiesz, że istnieje. W rzeczywistości możesz być zaskoczony, że wiele urządzeń w Twoim domu już korzysta z tej technologii. Zastosowania sztucznej inteligencji liczą się w milionach - wszystkie są bezpiecznie poza zasięgiem wzroku, nawet jeśli mają dość dramatyczny charakter. Oto tylko kilka sposobów, w jakie możesz zobaczyć użycie sztucznej inteligencji:

* Wykrywanie oszustw: Otrzymujesz telefon od wystawcy karty kredytowej z pytaniem, czy dokonałeś konkretnego zakupu. Firma obsługująca karty kredytowe nie jest wścibska; po prostu ostrzega Cię o tym, że ktoś inny może dokonywać zakupu przy użyciu Twojej karty. Sztuczna inteligencja osadzona w kodzie wystawcy karty kredytowej wykryła nieznany wzorzec wydatków i zaalarmowała o tym kogoś.

* Planowanie zasobów: wiele organizacji musi efektywnie planować wykorzystanie zasobów. Na przykład szpital może być zmuszony do określenia, gdzie umieścić pacjenta, na podstawie potrzeb pacjenta, dostępności wykwalifikowanych ekspertów oraz czasu, przez jaki lekarz spodziewa się, że pacjent będzie w szpitalu.

* Złożona analiza: ludzie często potrzebują pomocy przy złożonej analizie, ponieważ istnieje dosłownie zbyt wiele czynników do rozważenia. Na przykład ten sam zestaw objawów może wskazywać na więcej niż jeden problem. Lekarz lub inny ekspert może potrzebować pomocy w szybkim postawieniu diagnozy, aby uratować życie pacjenta.

* Automatyzacja: każda forma automatyzacji może skorzystać na dodaniu sztucznej inteligencji do obsługi nieoczekiwanych zmian lub zdarzeń. Problem z niektórymi typami automatyzacji polega

obecnie na tym, że nieoczekiwane zdarzenie, takie jak obiekt w niewłaściwym miejscu, może w rzeczywistości spowodować zatrzymanie automatyzacji. Dodanie sztucznej inteligencji do automatyzacji może pozwolić automatyzacji na obsługę nieoczekiwanych zdarzeń i kontynuowanie, jakby nic się nie wydarzyło.

* Obsługa klienta: linia obsługi klienta, do której dzwonisz dzisiaj, może nawet nie mieć za sobą człowieka. Automatyzacja jest wystarczająco dobra, aby śledzić skrypty i korzystać z różnych zasobów, aby odpowiedzieć na większość pytań. Przy dobrej infekcji głosowej (również zapewnianej przez sztuczną inteligencję) możesz nawet nie być w stanie stwierdzić, że rozmawiasz z komputerem.

* Systemy bezpieczeństwa: Wiele systemów bezpieczeństwa występujących obecnie w różnego rodzaju maszynach opiera się na sztucznej inteligencji, która przejmuje kontrolę nad pojazdem w czasie kryzysu. Na przykład wiele automatycznych układów hamulcowych polega na sztucznej inteligencji, która zatrzymuje samochód na podstawie wszystkich sygnałów wejściowych, które może dostarczyć pojazd, takich jak kierunek poślizgu.

* Wydajność maszyn: AI może pomóc w sterowaniu maszyną w taki sposób, aby uzyskać maksymalną wydajność. Sztuczna inteligencja kontroluje wykorzystanie zasobów, aby system nie przekraczał szybkości ani innych celów. Każda uncja energii jest wykorzystywana dokładnie tak, jak potrzeba, aby zapewnić pożądane usługi.

Koncentrując się na uczeniu maszynowym

Uczenie maszynowe to jeden z wielu podzbiorów sztucznej inteligencji i jedyny jaki omówimy. W uczeniu maszynowym celem jest stworzenie symulacji ludzkiego uczenia się, aby aplikacja mogła dostosować się do niepewnych lub nieoczekiwanych warunków. Aby wykonać to zadanie, uczenie maszynowe opiera się na algorytmach analizujących ogromne zbiory danych. Obecnie uczenie maszynowe nie zapewnia takiego rodzaju sztucznej inteligencji, jaki prezentują filmy (maszyna nie może się intuicyjnie uczyć, tak jak człowiek); może symulować tylko określone rodzaje uczenia się i to tylko w wąskim zakresie. Nawet najlepsze algorytmy nie potrafią myśleć, czuć, prezentować żadnej formy samoświadomości ani wykonywać wolnej woli. Cechy, które są podstawowe dla ludzi, są frustrująco trudne do uchwycenia przez maszyny z powodu tych ograniczeń percepcji. Maszyny nie są samoświadome. Uczenie maszynowe może wykonywać analizy predykcyjne znacznie szybciej niż jakikolwiek człowiek. W rezultacie uczenie maszynowe może pomóc ludziom pracować wydajniej. Obecny stan sztucznej inteligencji polega zatem na przeprowadzaniu analiz, ale ludzie nadal muszą brać pod uwagę konsekwencje tej analizy: podejmować wymagane decyzje moralne i etyczne. Istota sprawy polega na tym, że uczenie maszynowe zapewnia tylko uczącą się część sztucznej inteligencji, a ta część nie jest jeszcze w przybliżeniu gotowa do stworzenia sztucznej inteligencji, jaką można zobaczyć w filmach.

Głównym punktem pomyłki między uczeniem się a inteligencją jest to, że ludzie zakładają, że tylko dlatego, że maszyna staje się lepsza w swojej pracy (może się uczyć), jest również świadoma (ma inteligencję). Nic nie potwierdza tego poglądu na uczenie maszynowe. To samo zjawisko występuje, gdy ludzie zakładają, że komputer celowo powoduje u nich problemy. Komputer nie może przypisywać emocji i dlatego działa tylko na podstawie dostarczonych danych wejściowych i instrukcji zawartych w aplikacji, aby przetworzyć te dane. Prawdziwa sztuczna inteligencja w końcu pojawi się, gdy komputery będą mogły wreszcie naśladować sprytną kombinację używaną przez naturę:

* Genetyka: powolna nauka z pokolenia na pokolenie

* Nauczanie: Szybka nauka ze zorganizowanych źródeł

* Eksploracja: spontaniczne uczenie się poprzez media i interakcje z innymi

Aby koncepcje uczenia maszynowego były zgodne z możliwościami maszyny, należy rozważyć konkretne zastosowania systemów uczących się. Warto przyjrzeć się zastosowaniom uczenia maszynowego poza normalną sferą tego, co wielu uważa za domenę sztucznej inteligencji. Oto kilka zastosowań uczenia maszynowego, których możesz nie kojarzyć z AI:

* Kontrola dostępu: W wielu przypadkach kontrola dostępu jest propozycją tak lub nie. Karta inteligentna pracownika zapewnia dostęp do zasobu w taki sam sposób, jak ludzie używali kluczy od wieków. Niektóre blokady oferują możliwość ustawiania czasu i dat, w których dostęp jest dozwolony, ale taka zgrubna kontrola nie spełnia wszystkich potrzeb. Korzystając z uczenia maszynowego, możesz określić, czy pracownik powinien uzyskać dostęp do zasobu na podstawie roli i potrzeb. Na przykład pracownik może uzyskać dostęp do sali szkoleniowej, gdy szkolenie odzwierciedla rolę pracownika.

* Ochrona zwierząt: ocean może wydawać się wystarczająco duży, aby umożliwić zwierzętom i statkom wspólne życie bez problemu. Niestety co roku wiele zwierząt zostaje potrąconych przez statki. Algorytm uczenia maszynowego mógłby pozwolić statkom na unikanie zwierząt dzięki poznaniu dźwięków i cech zarówno zwierzęcia, jak i statku. (Statek polegałby na podwodnym sprzęcie nasłuchowym, który śledziłby zwierzęta za pomocą ich dźwięków, które można usłyszeć z dużej odległości od statku).

* Przewidywanie czasu oczekiwania: większość ludzi nie lubi czekać, gdy nie mają pojęcia, jak długo będzie czekać. Uczenie maszynowe umożliwia aplikacji określenie czasu oczekiwania na podstawie poziomów zatrudnienia, obciążenia personelu, złożoności problemów, które personel próbuje rozwiązać, dostępności zasobów i tak dalej.

Przejście od uczenia maszynowego do uczenia głębokiego

Jak wspomniano wcześniej, uczenie głębokie to podzbiór uczenia maszynowego. W obu przypadkach wydaje się, że algorytmy uczą się, analizując ogromne ilości danych (jednak w niektórych przypadkach uczenie się może nastąpić nawet przy niewielkich zbiorach danych). Jednak głębokie uczenie się różni się głębokością analizy i rodzajem automatyzacji, które zapewnia. Możesz podsumować różnice między nimi w następujący sposób:

* Zupełnie inny paradygmat: uczenie maszynowe to zbiór wielu różnych technik, które umożliwiają komputerowi uczenie się na podstawie danych i wykorzystywanie tego, czego się uczy, do udzielania odpowiedzi, często w formie przewidywań. Uczenie maszynowe opiera się na różnych paradygmatach, takich jak analiza statystyczna, znajdowanie analogii w danych, używanie logiki i praca z symbolami. Porównaj niezliczone techniki stosowane w uczeniu maszynowym z jedną techniką stosowaną w uczeniu głębokim, która naśladuje funkcje ludzkiego mózgu. Przetwarza dane za pomocą jednostek obliczeniowych, zwanych neuronami, ułożonych w uporządkowane sekcje, zwane warstwami. Techniką leżącą u podstaw głębokiego uczenia się jest sieć neuronowa.

* Elastyczne architektury: rozwiązania uczenia maszynowego oferują wiele gałek (korekty) zwane hiperparametrami, które dostosowujesz w celu optymalizacji uczenia algorytmu na podstawie danych. Rozwiązania do uczenia głębokiego również używają hiperparametrów, ale używają też wielu warstw konfigurowanych przez użytkownika (użytkownik określa liczbę i typ). W rzeczywistości, w zależności od powstałej sieci neuronowej, liczba warstw może być dość duża i tworzyć unikalne sieci neuronowe zdolne do wyspecjalizowanego uczenia się: niektóre mogą nauczyć się rozpoznawać obrazy, podczas gdy inne mogą wykrywać i analizować polecenia głosowe. Chodzi o to, że termin „głęboki” jest

właściwy; odnosi się do dużej liczby warstw potencjalnie używanych do analizy. Architektura składa się z zespołu różnych neuronów i ich ułożenia w warstwach w rozwiązaniu do głębokiego uczenia się.

* Definicja funkcji autonomicznych: rozwiązania uczenia maszynowego wymagają interwencji człowieka, aby odnieść sukces. Aby poprawnie przetwarzać dane, analitycy i naukowcy wykorzystują dużą część własnej wiedzy do opracowywania działających algorytmów. Na przykład w rozwiązaniu uczenia maszynowego, które określa wartość domu, opierając się na danych zawierających wymiary ścian różnych pomieszczeń, algorytm uczenia maszynowego nie będzie w stanie obliczyć powierzchni domu, chyba że analityk określi sposób obliczania to wcześniej. Tworzenie odpowiednich informacji dla algorytmu uczenia maszynowego nazywa się tworzeniem funkcji, co jest czynnością czasochłonną. Uczenie głębokie nie wymaga od ludzi wykonywania żadnych czynności związanych z tworzeniem funkcji, ponieważ dzięki wielu warstwom definiuje własne najlepsze funkcje. Dlatego też uczenie głębokie przewyższa uczenie maszynowe w bardzo trudnych zadaniach, takich jak rozpoznawanie głosu i obrazów, rozumienie tekstu lub pokonanie ludzkiego mistrza w grze Go (cyfrowa forma gry planszowej, w której zdobywasz terytorium przeciwnika).

Musisz zrozumieć wiele problemów związanych z rozwiązaniami uczenia głębokiego, z których najważniejszym jest to, że komputer nadal niczego nie rozumie i nie jest świadomy rozwiązania, które zapewnił. Po prostu zapewnia formę pętli sprzężenia zwrotnego i automatyzacji połączonych w celu wytworzenia pożądanych wyników w krótszym czasie, niż człowiek mógłby ręcznie uzyskać dokładnie ten sam wynik, manipulując rozwiązaniem uczącym się.

Druga kwestia polega na tym, że niektórzy ludzie w ciemności nalegali, aby warstwy głębokiego uczenia się były ukryte i niedostępne do analizy. Tak nie jest. Wszystko, co może zbudować komputer, jest ostatecznie śledzone przez człowieka. W rzeczywistości ogólne rozporządzenie o ochronie danych (RODO) wymaga, aby ludzie przeprowadzali taką analizę. Wymóg przeprowadzenia tej analizy jest kontrowersyjny, ale obowiązujące prawo mówi, że ktoś musi to zrobić. Trzecią kwestią jest to, że samoregulacja idzie tylko do tej pory. Uczenie głębokie nie zawsze zapewnia rzetelny lub poprawny wynik. W rzeczywistości rozwiązania do głębokiego uczenia mogą pójść strasznie źle. Nawet jeśli kod aplikacji nie idzie źle, urządzenia używane do obsługi głębokiego uczenia mogą. Mimo to, mając na uwadze te problemy, możesz zobaczyć głębokie uczenie używane w wielu niezwykle popularnych aplikacjach.

Korzystanie z Deep Learning w prawdziwym świecie

Nie popełnij błędu: ludzie używają uczenia głębokiego w prawdziwym świecie do wykonywania szerokiego zakresu zadań. Na przykład wiele samochodów używa obecnie interfejsu głosowego. Interfejs głosowy może wykonywać podstawowe zadania, nawet od samego początku. Jednak im więcej z nim rozmawiasz, tym lepiej działa interfejs głosowy. Interfejs uczy się, gdy z nim rozmawiasz - nie tylko sposobu, w jaki mówisz, ale także osobistych preferencji. Poniższe sekcje zawierają trochę informacji o tym, jak głębokie uczenie działa w prawdziwym świecie.

Zrozumienie pojęcia uczenia się

Kiedy ludzie się uczą, polegają na czymś więcej niż tylko danych. Ludzie mają intuicję oraz niesamowitą wiedzę o tym, co zadziała, a co nie. Częścią tej wrodzonej wiedzy jest instynkt, który jest przekazywany z pokolenia na pokolenie poprzez DNA. Sposób, w jaki ludzie wchodzą w interakcje z danymi wejściowymi, również różni się od tego, co robi komputer. W przypadku komputera uczenie się polega na zbudowaniu bazy danych składającej się z sieci neuronowej, która ma wbudowane wagi i odchylenia, aby zapewnić prawidłowe przetwarzanie danych. Sieć neuronowa przetwarza następnie dane, ale nie w sposób, który jest nawet zdalnie taki sam, jak będzie to robił człowiek.

Wykonywanie zadań uczenia głębokiego

Ludzie i komputery są najlepsi w różnych zadaniach. Ludzie są najlepsi w rozumowaniu, myśleniu o etycznych rozwiązaniach i emocjonalności. Komputer ma przetwarzać dane - dużo danych - naprawdę szybko. Często używasz uczenia głębokiego do rozwiązywania problemów, które wymagają wyszukiwania wzorców w ogromnych ilościach danych - problemów, których rozwiązanie jest nieintuicyjne i niezauważalne od razu. Niemal w każdym przypadku można podsumować problem i jego rozwiązanie jako szybkie przetwarzanie ogromnych ilości danych, szukanie wzorców, a następnie poleganie na tych wzorcach, aby odkryć coś nowego lub stworzyć konkretny rodzaj wyjścia.

Wykorzystanie uczenia głębokiego w aplikacjach

Uczenie głębokie może być samodzielnym rozwiązaniem, ale często jest używane jako część znacznie większego rozwiązania i jest mieszane z innymi technologiami. Na przykład łączenie uczenia głębokiego z systemami eksperckimi nie jest rzadkością, jednak rzeczywiste aplikacje to coś więcej niż tylko liczby generowane z jakiegoś mglistego źródła. Pracując w świecie rzeczywistym, należy również wziąć pod uwagę różne rodzaje źródeł danych i zrozumieć, jak one działają. Kamera może wymagać innego rodzaju rozwiązania do głębokiego uczenia, aby uzyskać z niej informacje, podczas gdy termometr lub detektor zbliżeniowy może generować proste liczby (lub dane analogowe, które wymagają pewnego rodzaju przetwarzania). Rzeczywiste rozwiązania są niechlujne, więc musisz być przygotowany na więcej niż jedno rozwiązanie problemów w swoim zestawie narzędzi.

Biorąc pod uwagę środowisko programowania Deep Learning

Możesz automatycznie założyć, że musisz przeskoczyć przez okropny zestaw kółek i nauczyć się ezoterycznych umiejętności programowania, aby zagłębić się w głębokie uczenie się. To prawda, że zyskujesz elastyczność, pisząc aplikacje przy użyciu jednego z języków programowania, które dobrze sprawdzają się w przypadku potrzeb uczenia głębokiego. Jednak Deep Learning Studio i inne podobne produkty umożliwiają ludziom tworzenie rozwiązań do głębokiego uczenia bez programowania. Zasadniczo, takie rozwiązania obejmują opisanie tego, co chcesz jako wynik, poprzez graficzne zdefiniowanie modelu. Tego rodzaju rozwiązania sprawdzają się dobrze w przypadku prostych problemów, które inni musieli już rozwiązać, ale brakuje im elastyczności, aby coś zrobić zupełnie inaczej - zadanie, które wymaga czegoś więcej niż zwykłej analizy. Rozwiązania do uczenia głębokiego w chmurze, takie jak te dostarczane przez Amazon Web Services (AWS), mogą zapewnić dodatkową elastyczność. Te środowiska również zwykle upraszczają środowisko programistyczne, zapewniając tyle lub mało wsparcia, ile chcesz. W rzeczywistości AWS zapewnia obsługę różnego rodzaju bezserwerowego computingu, w którym nie musisz martwić się o żadną infrastrukturę. Jednak te rozwiązania mogą stać się dość drogie. Mimo że zapewniają większą elastyczność niż korzystanie z gotowego rozwiązania, nadal nie są tak elastyczne, jak w przypadku rzeczywistego środowiska programistycznego.

Masz też do rozważenia inne rozwiązania nieprogramistyczne. Na przykład, jeśli potrzebujesz mocy i elastyczności, ale nie chcesz programować, aby to uzyskać, możesz polegać na produkcie takim jak MATLAB, który zapewnia zestaw narzędzi do głębokiego uczenia się. MATLAB i niektóre inne środowiska koncentrują się bardziej na algorytmach, których chcesz używać, ale aby uzyskać z nich pełną funkcjonalność, musisz pisać skrypty jako minimum, co oznacza, że zanurzasz się w programowaniu niektórych stopni. Problem z tymi środowiskami polega na tym, że może im brakować również w dziale zasilania, więc niektóre rozwiązania mogą zająć więcej czasu, niż się spodziewasz. W pewnym momencie, bez względu na to, ile innych rozwiązań spróbujesz, poważne problemy z głębokim uczeniem będą wymagały programowania. Przeglądając wybory online, często widzisz sztuczną inteligencję, uczenie maszynowe i uczenie głębokie, wszystkie razem. Jednak podobnie jak te trzy

technologie działają na różnych poziomach, tak samo działają języki programowania , których potrzebujesz. Dobre rozwiązanie do głębokiego uczenia będzie wymagało zastosowania przetwarzania wieloprocesowego, najlepiej z wykorzystaniem jednostki przetwarzania grafiki (GPU) z wieloma rdzeniami. Twój wybrany język musi również obsługiwać GPU za pośrednictwem kompatybilnej biblioteki lub pakietu. Zatem sam wybór języka zwykle nie wystarczy; musisz dokładniej zbadać, czy język rzeczywiście spełnia Twoje potrzeby. Mając to na uwadze, oto najpopularniejsze języki (w kolejności popularności w chwili pisania tego tekstu) do zastosowania w uczeniu głębokim:

*Pyton

* R

* MATLAB (język skryptowy, a nie produkt)

*Oktave

Jedynym problemem z tą listą jest to, że inni programiści mają inne opinie. Python i R zwykle pojawiają się na górze wszystkich list, ale potem możesz znaleźć różnego rodzaju opinie. Wybierając język, zazwyczaj musisz wziąć pod uwagę następujące kwestie:

* Krzywa uczenia się: Twoje doświadczenia mają wiele do powiedzenia na temat tego, czego możesz się najłatwiej nauczyć. Python jest prawdopodobnie najlepszym wyborem dla kogoś, kto programuje od wielu lat, ale R może być lepszym wyborem dla kogoś, kto ma już doświadczenie w programowaniu funkcjonalnym. MATLAB lub Octave mogą działać najlepiej dla matematyka.

* Szybkość: każde rozwiązanie do głębokiego uczenia będzie wymagało dużej mocy obliczeniowej. Wiele osób twierdzi, że ponieważ R jest językiem statystycznym, oferuje więcej wsparcia statystycznego i zwykle zapewnia szybsze wyniki. W rzeczywistości obsługa Pythona dla doskonałego programowania równoległego prawdopodobnie niweluje tę przewagę, gdy masz wymagany sprzęt.

* Wsparcie społeczności: istnieje wiele form wsparcia społeczności, ale dwa najważniejsze dla głębokiego uczenia się to pomoc w zdefiniowaniu rozwiązania i dostęp do wielu gotowych pomocy programowych. Po czwarte, Octave prawdopodobnie zapewnia najmniej wsparcia społeczności; Python zapewnia najwięcej.

* Koszt: ile kosztuje język, zależy od wybranego rozwiązania i miejsca jego uruchomienia. Na przykład MATLAB to zastrzeżony produkt, który wymaga zakupu, więc masz coś zainwestowanego natychmiast, gdy używasz MATLAB. Jednak mimo że inne języki są na początku bezpłatne, możesz znaleźć ukryte koszty, takie jak uruchomienie kodu w chmurze, aby uzyskać dostęp do obsługi GPU. Obsługa struktur DNN: Framework może znacznie ułatwić pracę z Twoim językiem. Jednak musisz mieć framework która dobrze współgra ze wszystkimi innymi częściami rozwiązania. Dwie najpopularniejsze frameworki to TensorFlow i PyTorch. Co dziwne, Python jest jedynym językiem, który obsługuje oba, więc oferuje największą elastyczność. Używasz Case z MATLAB i TensorFlow z R.

*Gotowy do produkcji: język musi obsługiwać taki rodzaj wyjścia, jaki jest potrzebny w Twoim projekcie. Pod tym względem Python wyróżnia się, ponieważ jest językiem ogólnego przeznaczenia. Możesz z nim stworzyć dowolną aplikację. Jednak bardziej szczegółowe środowiska zapewniane przez inne języki mogą być niezwykle pomocne w przypadku niektórych projektów, dlatego należy wziąć pod uwagę wszystkie z nich.

Pokonywanie szumu związanego z głębokim uczeniem się

W poprzednich częściach omówiono niektóre kwestie związane z postrzeganiem uczenia głębokiego, takie jak przekonanie niektórych ludzi, że pojawia się ono wszędzie i robi wszystko. Problem z głębokim uczeniem się polega na tym, że padł on ofiarą własnej kampanii medialnej. Uczenie głębokie rozwiązuje specyficzne problemy. Poniższe sekcje pomogą Ci uniknąć szumu związanego z głębokim uczeniem się.

Odkrywanie ekosystemu start-upów

Korzystanie z rozwiązania do uczenia głębokiego znacznie różni się od tworzenia własnego rozwiązania do uczenia głębokiego. Samo spełnienie wymagań edukacyjnych może zająć trochę czasu. Jednak po samodzielnym przepracowaniu kilku projektów zaczynasz zdawać sobie sprawę, że szum wokół głębokiego uczenia się rozciąga się aż do początku konfiguracji. Głębokie uczenie nie jest dojrzałą technologią, więc próba jej wykorzystania przypomina budowanie wioski na Księżycu lub głębokie nurkowanie w Rowie Mariańskm. Napotkasz problemy, a technologia będzie się nieustannie zmieniać. Niektóre metody stosowane do tworzenia rozwiązań głębokiego uczenia również wymagają pracy. Koncepcja, że komputer faktycznie uczy się czegokolwiek, jest fałszywa, podobnie jak idea, że komputery mają w ogóle jakąkolwiek formę świadomości. Powodem, dla którego Microsoft, Amazon i inni dostawcy mają problemy z głębokim uczeniem się, jest to, że nawet ich inżynierowie mogą mieć nierealistyczne oczekiwania. Głębokie uczenie się sprowadza się do matematyki i dopasowywania wzorców - z pewnością naprawdę wymyślnej matematyki i dopasowywania wzorców, ale myśl, że to cokolwiek innego, jest po prostu błędna.

Wiedzieć, kiedy nie używać głębokiego uczenia

Głębokie uczenie to tylko jeden sposób przeprowadzania analizy i nie zawsze jest to najlepszy sposób. Na przykład, mimo że systemy eksperckiee są uważane za starą technologię, tak naprawdę nie można stworzyć autonomicznego samochodu bez niego . Rozwiązanie do głębokiego uczenia się okazuje się zbyt wolne dla tej konkretnej potrzeby. Twój samochód prawdopodobnie będzie zawierał ogólnie uczącą się sztuczną inteligencję, a uczenie głębokie w szczególności może trafić na nagłówki gazet, gdy technologia nie spełni oczekiwań. Błędem jest myślenie, że głębokie uczenie się może w jakiś sposób podejmować decyzje etyczne lub że wybierze właściwy sposób działania w oparciu o uczucia (których nie ma żadna maszyna). Antropomorfizacja korzystania z głębokiego uczenia się zawsze będzie błędem. Niektóre zadania wymagają po prostu człowieka. Szybkość i zdolność do myślenia jak człowiek to najważniejsze kwestie związane z głębokim uczeniem się, ale jest ich o wiele więcej. Na przykład nie możesz korzystać z uczenia głębokiego, jeśli nie masz wystarczających danych, aby go wytrenować.

Przedstawiamy zasady uczenia maszynowego

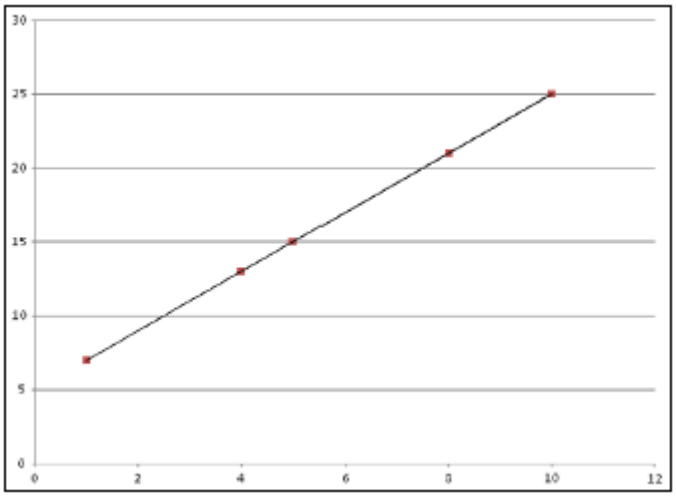
Jak omówiono w Części 1, koncepcja uczenia się przez komputer różni się od koncepcji uczenia się przez ludzi. Jednak Część 1 tak naprawdę nie opisuje uczenia maszynowego, rodzaju uczenia się, z którego korzysta komputer, w żaden sposób. W końcu to, na co naprawdę patrzysz, to zupełnie inny rodzaj uczenia się, który niektórzy ludzie postrzegają jako połączenie matematyki, dopasowywania wzorców i przechowywania danych. Ta część rozpoczyna się od wskazania drogi do głębszego zrozumienia działania uczenia maszynowego. Jednak wyjaśnienie uczenia maszynowego nie pomaga całkowicie zrozumieć, co się dzieje, gdy z nim pracujesz. Istotne jest również to, jak działa uczenie maszynowe, o czym mowa w kolejnej części. W tej sekcji odkryjesz, że nie istnieją doskonałe metody przeprowadzania analiz. Być może będziesz musiał poeksperymentować z analizą, aby uzyskać oczekiwany wynik. Ponadto dostępne są różne podejścia do uczenia maszynowego, a każde z nich ma zalety i wady. Trzecia część obejmuje to, co odkryłeś w poprzednich dwóch częściach, i pomaga to zastosować. Bez względu na to, jak kształtujesz dane i przeprowadzasz na nich analizę, uczenie maszynowe jest w niektórych przypadkach niewłaściwym podejściem i nigdy nie zapewni użytecznych wyników. Znajomość właściwych zastosowań uczenia maszynowego jest niezbędna, jeśli chcesz otrzymywać spójne wyniki, które pomagają w wykonywaniu interesujących zadań. Celem uczenia maszynowego jest nauczenie się czegoś interesującego z danych, a następnie zrobienie z nimi czegoś interesującego.

Definiowanie uczenia maszynowego

Oto krótka definicja uczenia maszynowego: jest to aplikacja sztucznej inteligencji, która może automatycznie uczyć się i ulepszać na podstawie doświadczenia, bez konieczności bezpośredniego zaprogramowania. Uczenie się następuje w wyniku analizowania coraz większej ilości danych, więc podstawowe algorytmy się nie zmieniają, ale wewnętrzne wagi i odchylenia kodu użyte do wybrania konkretnej odpowiedzi tak. Oczywiście nic nie jest takie proste. W kolejnych sekcjach omówiono więcej na temat tego, czym jest uczenie maszynowe, abyś mógł zrozumieć jego miejsce w świecie sztucznej inteligencji i co czerpie z uczenia głębokiego. Naukowcy zajmujący się danymi często nazywają technologię używaną do implementacji uczenia maszynowego algorytmami. Algorytm to seria operacji wykonywanych krok po kroku, zwykle obliczeń, które mogą rozwiązać określony problem w skończonej liczbie kroków. W uczeniu maszynowym algorytmy wykorzystują serię skończonych kroków, aby rozwiązać problem, ucząc się na podstawie danych.

Zrozumienie, jak działa uczenie maszynowe

Algorytmy uczenia maszynowego uczą się, ale często trudno jest znaleźć dokładne znaczenie terminu uczenie się, ponieważ istnieją różne sposoby wydobywania informacji z danych, w zależności od tego, jak zbudowany jest algorytm uczenia maszynowego. Ogólnie proces uczenia się wymaga ogromnych ilości danych, które zapewniają oczekiwaną odpowiedź przy określonych danych wejściowych. Każda para wejście / odpowiedź stanowi przykład, a więcej przykładów ułatwia algorytmowi naukę. Dzieje się tak, ponieważ każda para wejście / odpowiedź pasuje do linii, klastra lub innej reprezentacji statystycznej, która definiuje dziedzinę problemu. Uczenie się to czynność polegająca na optymalizacji modelu, który jest matematyczną, podsumowaną reprezentacją samych danych, tak aby mógł przewidzieć lub w inny sposób określić odpowiednią reakcję, nawet jeśli otrzyma dane wejściowe, których wcześniej nie widział. Im dokładniej model może podać prawidłowe odpowiedzi, tym lepiej model nauczył się na podstawie dostarczonych danych wejściowych. Algorytm dopasowuje model do danych, a ten proces dopasowywania jest uczeniem. Rysunek przedstawia niezwykle prosty wykres, który symuluje to, co dzieje się w uczeniu maszynowym.



W tym przypadku, zaczynając od wartości wejściowych 1, 4, 5, 8 i 10 i łącząc je z odpowiadającymi im wyjściami 7, 13, 15, 21 i 25, algorytm uczenia maszynowego ustala, że najlepszym sposobem reprezentacji Relacja między wejściem a wyjściem to formuła 2x + 5. Ta formuła definiuje model używany do przetwarzania danych wejściowych - nawet nowych, niewidocznych danych - do obliczania odpowiadającej im wartości wyjściowej. Linia trendu (model) przedstawia wzorzec utworzony przez ten algorytm, tak że nowe dane wejściowe 3 dadzą przewidywany wynik równy 11. Mimo że większość scenariuszy uczenia maszynowego jest znacznie bardziej skomplikowana (a algorytm nie może utworzyć reguły, które dokładnie odwzorowują każde wejście na precyzyjne wyjście), przykład daje podstawowe pojęcie o tym, co się dzieje. Zamiast indywidualnie programować odpowiedź dla danych wejściowych 3, model może obliczyć poprawną odpowiedź na podstawie par wejście / odpowiedź, których się nauczył.

Zrozumieć, że to czysta matematyka

Główną ideą uczenia maszynowego jest to, że można przedstawić rzeczywistość za pomocą funkcji matematycznej, której algorytm nie zna z góry, ale której może odgadnąć po obejrzeniu pewnych danych (zawsze w postaci sparowanych danych wejściowych i wyjściowych). Możesz wyrazić rzeczywistość i całą jej trudną złożoność za pomocą nieznanych funkcji matematycznych, które algorytmy uczenia maszynowego znajdują i udostępniają jako modyfikację ich wewnętrznej funkcji matematycznej. Oznacza to, że każdy algorytm uczenia maszynowego jest zbudowany wokół modyfikowalnej funkcji matematycznej. Funkcję można modyfikować, ponieważ ma do tego celu wewnętrzne parametry lub wagi. W rezultacie algorytm może dostosować funkcję do określonych informacji zaczerpniętych z danych. Ta koncepcja jest głównym pomysłem dla wszystkich rodzajów algorytmów uczenia maszynowego. Uczenie się w uczeniu maszynowym jest czysto matematyczne i kończy się powiązaniem pewnych danych wejściowych z określonymi wynikami. Nie ma to nic wspólnego ze zrozumieniem, czego nauczył się algorytm. (Kiedy ludzie analizują dane, w pewnym stopniu budujemy zrozumienie danych). Proces uczenia się jest często opisywany jako szkolenie, ponieważ algorytm jest uczony, aby dopasować poprawną odpowiedź (dane wyjściowe) do każdego oferowanego pytania (dane wejściowe. Uczenie maszynowe, pomimo braku świadomego zrozumienia i bycia procesem matematycznym, może okazać się przydatne w wielu zadaniach. Zapewnia wielu aplikacjom sztucznej inteligencji moc naśladowania racjonalnego myślenia w określonym kontekście, gdy uczenie się odbywa się przy użyciu odpowiednich danych.

Uczenie się za pomocą różnych strategii

Uczenie maszynowe oferuje wiele różnych sposobów uczenia się na podstawie danych. W zależności od oczekiwanych wyników i typu wprowadzonych danych wejściowych, algorytmy można kategoryzować według stylu uczenia się. Wybrany styl zależy od rodzaju posiadanych danych i oczekiwanego wyniku. Cztery style uczenia się używane do tworzenia algorytmów to

* Nadzorowane

* Bez nadzoru

* Samonadzorowany

* Wzmocnienie

W poniższych sekcjach omówiono style uczenia się.

Nadzorowane

Podczas pracy z nadzorowanymi algorytmami dane wejściowe są oznaczone i mają określony oczekiwany wynik. Trening służy do tworzenia modelu dopasowanego do danych przez algorytm. W miarę postępu szkolenia przewidywania lub klasyfikacje stają się dokładniejsze. Oto kilka przykładów algorytmów uczenia nadzorowanego:

* Regresja liniowa lub logistyczna

* Obsługa maszyn wektorowych (SVM)

* Naïve Bayes

* K-Nearest Neighbors (KNN)

Musisz rozróżnić między problemami regresji, których celem jest wartość liczbowa, a problemami klasyfikacyjnymi, których celem jest zmienna jakościowa, taka jak klasa lub znacznik. Zadanie regresji mogłoby określić średnie ceny domów w rejonie Bostonu, podczas gdy przykładem zadania klasyfikacyjnego jest rozróżnienie rodzajów kwiatów tęczówki na podstawie ich miary działkowej i płatkowej. Oto kilka przykładów nadzorowanego uczenia się:

Wejście danych (X): Wyjście danych (y): Aplikacja w świecie rzeczywistym

Historia zakupów klientów: lista produktów, których klienci nigdy nie kupili: system rekomendacji

Obrazy: lista pól oznaczonych nazwą obiektu: Wykrywanie i rozpoznawanie obrazu

Tekst angielski w formie pytań: Tekst angielski w formie odpowiedzi: Chatbot, aplikacja umożliwiająca konwersację

Tekst w języku angielskim: Tekst w języku niemieckim: Tłumaczenie maszynowe

Audio: Zapis tekstowy: Rozpoznawanie mowy

Obraz, dane z czujnika: Kierowanie, hamowanie lub przyspieszanie: planowanie behawioralne dla autonomicznej jazdy

Bez nadzoru

Podczas pracy z nienadzorowanymi algorytmami dane wejściowe nie są etykietowane, a wyniki nie są znane. W takim przypadku analiza struktur w danych tworzy wymagany model. Analiza strukturalna

może mieć kilka celów, takich jak zmniejszenie nadmiarowości lub grupowanie podobnych danych. Przykłady uczenia się bez nadzoru to

* Clustering

*Wykrywanie anomalii

*Sieci neuronowe

Samonadzorowany

W Internecie można znaleźć wszelkiego rodzaju rodzaje uczenia się, ale samokontrola jest odrębną kategorią. Niektórzy ludzie opisują to jako autonomiczne nadzorowane uczenie się, które daje korzyści z nadzorowanego uczenia się, ale bez całej pracy wymaganej do oznaczania danych. Teoretycznie samokontrola może rozwiązać problemy z innymi rodzajami uczenia się, z których obecnie możesz korzystać. Poniższa lista porównuje samodzielne uczenie się z innymi rodzajami uczenia się, z których korzystają ludzie.

* Uczenie się nadzorowane: Najbliższą formą uczenia się związaną z uczeniem się bez nadzoru jest uczenie się nadzorowane, ponieważ oba rodzaje uczenia się opierają się na parach wejść i oznaczonych wynikach. Ponadto obie formy uczenia się są związane z regresją i klasyfikacją. Różnica polega jednak na tym, że samodzielne uczenie się nie wymaga od osoby etykietowania wyniku. Zamiast tego opiera się na korelacjach, osadzonych metadanych lub wiedzy dziedzinowej osadzonej w danych wejściowych w celu kontekstowego odkrycia etykiety wyjściowej.

* Uczenie się nienadzorowane: Podobnie jak uczenie się nienadzorowane, uczenie się samodzielnie nadzorowane nie wymaga etykietowania danych. Jednak uczenie się bez nadzoru koncentruje się na strukturze danych, czyli wzorcach w danych. Dlatego nie używasz samokontroli uczenia się do zadań takich jak grupowanie, grupowanie, redukcja wymiarowości, silniki rekomendacji itp.

* Częściowo nadzorowane uczenie: częściowo nadzorowane rozwiązanie do nauki działa jak nienadzorowane rozwiązanie do nauki, ponieważ szuka wzorców danych. Jednak częściowo nadzorowane uczenie się opiera się na połączeniu danych oznaczonych i nieoznaczonych, aby wykonywać swoje zadania szybciej niż jest to możliwe przy użyciu ściśle nieoznaczonych danych. Samodzielne uczenie się nigdy nie wymaga etykiet i używa kontekstu do wykonania swojego zadania, więc faktycznie zignorowałoby je, gdy zostały dostarczone.

Wzmocnienie

Możesz postrzegać uczenie się ze wzmocnieniem jako rozszerzenie samokontroli uczenia się, ponieważ obie formy wykorzystują to samo podejście do uczenia się z nieoznaczonymi danymi, aby osiągnąć podobne cele. Jednak uczenie się przez wzmacnianie dodaje pętlę sprzężenia zwrotnego. Kiedy rozwiązanie uczenia się ze wzmocnieniem wykonuje poprawnie zadanie, otrzymuje pozytywną informację zwrotną, która wzmacnia model w łączeniu docelowych danych wejściowych i wyjściowych. Podobnie może otrzymać negatywną informację zwrotną za nieprawidłowe rozwiązania. Pod pewnymi względami system działa tak samo, jak praca z psem w oparciu o system nagród. Szkolenie, walidacja i testowanie danych . Uczenie maszynowe jest procesem, tak jak wszystko jest procesem w świecie komputerów. Aby zbudować skuteczne rozwiązanie do uczenia maszynowego, wykonuj te zadania w razie potrzeby i tak często, jak potrzeba:

* Szkolenie: uczenie maszynowe rozpoczyna się, gdy trenujesz model przy użyciu określonego algorytmu w odniesieniu do określonych danych. Dane szkoleniowe są oddzielne od innych danych, ale muszą być również reprezentatywne. Jeśli dane uczące nie reprezentują naprawdę domeny problemu,

wynikowy model nie może dostarczyć użytecznych wyników. Podczas procesu uczenia widzisz, jak model reaguje na dane uczące i w razie potrzeby wprowadzasz zmiany w używanych algorytmach i sposobie, w jaki masujesz dane przed wprowadzeniem do algorytmu

* Weryfikacja: wiele zbiorów danych jest wystarczająco dużych, aby podzielić je na część szkoleniową i testową. Najpierw trenujesz model przy użyciu danych uczących, a następnie weryfikujesz go przy użyciu danych testowych. Oczywiście dane testowe muszą ponownie dokładnie reprezentować dziedzinę problemu. Musi być również statystycznie zgodny z danymi uczącymi. W przeciwnym razie nie zobaczysz wyników odzwierciedlających rzeczywiste działanie modelu.

* Testowanie: po przeszkoleniu i walidacji modelu nadal musisz przetestować go przy użyciu rzeczywistych danych. Ten krok jest ważny, ponieważ musisz sprawdzić, czy model rzeczywiście będzie działał na większym zbiorze danych, którego nie używałeś ani do szkolenia, ani do testowania. Podobnie jak w przypadku etapów szkolenia i walidacji, wszelkie dane używane na tym etapie muszą odzwierciedlać domenę problemową, z którą chcesz współdziałać przy użyciu modelu uczenia maszynowego.

Szkolenie zapewnia algorytm uczenia maszynowego z różnego rodzaju przykładami pożądanych danych wejściowych i wyjściowych, których oczekuje się od tych danych wejściowych. Algorytm uczenia maszynowego wykorzystuje następnie te dane wejściowe do tworzenia funkcji matematycznej. Innymi słowy, uczenie to proces, w którym algorytm sprawdza, jak dostosować funkcję do danych. Wynik takiej funkcji jest zwykle prawdopodobieństwem określonego wyniku lub po prostu wartością liczbową jako wyjściem. Aby dać wyobrażenie o tym, co dzieje się w procesie szkolenia, wyobraź sobie dziecko uczące się odróżniać drzewa od przedmiotów, zwierząt i ludzi. Zanim dziecko będzie mogło to zrobić w sposób niezależny, nauczyciel przedstawia dziecku pewną liczbę obrazów drzew, wraz ze wszystkimi faktami, które sprawiają, że drzewo można odróżnić od innych obiektów świata. Takimi faktami mogą być cechy, takie jak materiał drzewa (drewno), jego części (pień, gałęzie, liście lub igły, korzenie) oraz lokalizacja (posadzona w glebie). Dziecko buduje zrozumienie tego, jak wygląda drzewo, kontrastując wyświetlanie cech drzewa z obrazami innych, różnych przykładów, takich jak meble wykonane z drewna, które nie mają innych cech z drzewem. Klasyfikator systemów uczących się działa tak samo. Algorytm klasyfikatora zapewnia klasę jako wynik. Na przykład może ci powiedzieć, że zdjęcie, które podajesz jako dane wejściowe, pasuje do klasy drzewa (a nie do zwierzęcia lub osoby). Aby to zrobić, buduje swoje zdolności poznawcze, tworząc formułę matematyczną, która obejmuje wszystkie dane wejściowe w sposób, który tworzy funkcję, która może odróżnić jedną klasę od drugiej.

Szukam uogólnienia

Aby model uczenia maszynowego był przydatny, musi przedstawiać ogólny widok dostarczonych danych. Jeśli model nie śledzi wystarczająco dokładnie danych, jest niedopasowany - to znaczy niedostatecznie dopasowany z powodu braku szkolenia. Z drugiej strony, jeśli model zbyt dokładnie podąża za danymi, jest zbyt dopasowany, podążając za punktami danych jak rękawiczka z powodu zbyt intensywnego treningu. Niedopasowanie i nadmierne dopasowanie powodują problemy, ponieważ model nie jest wystarczająco uogólniony, aby uzyskać użyteczne wyniki. Biorąc pod uwagę nieznane dane wejściowe, wynikające z nich prognozy lub klasyfikacje będą zawierać duże wartości błędów. Tylko wtedy, gdy model jest prawidłowo dopasowany do danych, zapewni wyniki w rozsądnym zakresie błędu. Cała ta kwestia uogólniania jest również ważna przy podejmowaniu decyzji, kiedy zastosować uczenie maszynowe. Rozwiązanie uczenia maszynowego zawsze uogólnia od konkretnych przykładów do ogólnych przykładów tego samego rodzaju. Sposób wykonania tego zadania zależy od orientacji rozwiązania uczenia maszynowego i algorytmów użytych do jego działania. Problem

naukowców zajmujących się danymi i innych osób korzystających z technik uczenia maszynowego i głębokiego uczenia się polega na tym, że komputer nie wyświetla znaku informującego o poprawnym dopasowaniu modelu do danych. Często kwestią ludzkiej intuicji jest podjęcie decyzji, kiedy model jest wystarczająco wyszkolony, aby zapewnić dobry ogólny wynik. Ponadto twórca rozwiązania musi wybrać odpowiedni algorytm z tysięcy istniejących. Bez odpowiedniego algorytmu dopasowującego model do danych wyniki będą rozczarowujące. Aby proces selekcji zadziałał, analityk danych musi posiadać

* Dobra znajomość dostępnych algorytmów

* Doświadczenie w pracy z takimi danymi

* Zrozumienie pożądanego wyniku

* Chęć eksperymentowania z różnymi algorytmami

Ostatni wymóg jest najważniejszy, ponieważ nie ma sztywnych reguł, które mówią, że określony algorytm będzie działał z każdym rodzajem danych w każdej możliwej sytuacji. Gdyby tak było, tak wiele algorytmów nie byłoby dostępnych. Aby znaleźć najlepszy algorytm, analityk danych często ucieka się do eksperymentowania z wieloma algorytmami i porównywania wyników.

Poznanie granic uprzedzeń

Twój komputer nie ma uprzedzeń. Nie ma celu dominacji nad światem ani utrudniania ci życia. W rzeczywistości komputery nie mają żadnych celów. Jedyne, co komputer może dostarczyć, to dane wyjściowe oparte na danych wejściowych i technice przetwarzania. Jednak stronniczość nadal dostaje się do komputera i na wiele sposobów wpływa na wyniki, które zapewnia:

* Dane: same dane mogą zawierać nieprawdy lub po prostu nieprawdziwe informacje. Na przykład, jeśli dana wartość pojawia się w danych dwa razy częściej niż w świecie rzeczywistym, dane wyjściowe rozwiązania uczenia maszynowego są skażone, nawet jeśli same dane są poprawne.

* Algorytm: użycie niewłaściwego algorytmu spowoduje, że rozwiązanie uczące się nieprawidłowo dopasuje model do danych.

* Trening: zbyt dużo lub zbyt mało treningu zmienia sposób dopasowania modelu do danych, a tym samym do wyniku.

* Interpretacja ludzka: nawet jeśli rozwiązanie uczenia maszynowego zwraca prawidłowy wynik, człowiek korzystający z tego wyniku może go błędnie zinterpretować. Wyniki są równie złe, a być może gorsze niż wtedy, gdy rozwiązanie uczenia maszynowego nie działa zgodnie z oczekiwaniami.

Musisz wziąć pod uwagę skutki uprzedzeń bez względu na rodzaj tworzonego rozwiązania do uczenia maszynowego. Ważne jest, aby wiedzieć, jakie rodzaje ograniczeń nakładają te uprzedzenia na Twoje rozwiązanie i czy jest ono wystarczająco niezawodne, aby zapewnić użyteczne wyniki.

Pamiętając o złożoności modelu

Prostsze jest zawsze lepsze, jeśli chodzi o uczenie maszynowe. Wiele różnych algorytmów może dostarczyć użytecznych danych wyjściowych z rozwiązania uczenia maszynowego, ale najlepszym algorytmem jest ten, który jest najłatwiejszy do zrozumienia i zapewnia najprostsze wyniki. Occam’s Razor (http://math.ucr.edu/home/baez/physics/General/occam.html) jest ogólnie uznawany za najlepszą strategię do naśladowania. Zasadniczo Brzytwa Occam'a mówi ci, aby użyć najprostszego

rozwiązania, które rozwiąże określony problem. Wraz ze wzrostem złożoności rośnie też możliwość wystąpienia błędów.

Biorąc pod uwagę wiele różnych dróg do nauki

Ucząca się część uczenia maszynowego sprawia, że jest ona dynamiczna - to znaczy może się zmieniać, gdy otrzyma dodatkowe dane. Zdolność uczenia się sprawia, że uczenie maszynowe różni się od innych rodzajów sztucznej inteligencji, takich jak wykresy wiedzy i systemy eksperkie. Nie sprawia, że uczenie maszynowe jest lepsze niż inne sztucznej inteligencji, ale jest po prostu przydatne w przypadku określonego zestawu problemów. Oczywiście problem z kwantyfikacją tego, co pociąga za sobą uczenie się, polega na tym, że ludzie i komputery inaczej postrzegają uczenie się. Ponadto komputery używają różnych technik uczenia się niż ludzie, a niektórzy ludzie mogą nie widzieć części uczenia się jako uczenie maszynowe w ogóle jako uczenie się. W poniższych sekcjach omówiono metody używane przez algorytmy uczenia maszynowego do uczenia się, aby można było lepiej zrozumieć, że uczenie maszynowe i uczenie się człowieka są z natury różne.

Zrozumienie, że nie ma darmowego lunchu

Być może słyszałeś powszechny mit, że można mieć wszystko na drodze do wyjścia komputera bez wkładania dużego wysiłku w znalezienie rozwiązania. Niestety, nie ma absolutnego rozwiązania żadnego problemu, a lepsze odpowiedzi są często dość kosztowne. Podczas pracy z algorytmami szybko odkrywasz, że niektóre algorytmy radzą sobie lepiej niż inne w rozwiązywaniu pewnych problemów, ale nie ma też jednego algorytmu, który najlepiej sprawdza się w przypadku każdego problemu. Dzieje się tak z powodu matematyki stojącej za algorytmami. Pewne funkcje matematyczne dobrze radzą sobie z przedstawieniem niektórych problemów, ale mogą uderzyć w ścianę w przypadku innych problemów. Każdy algorytm ma swoją specjalność.

Odkrywanie pięciu głównych podejść

Algorytmy mają różne formy i wykonują różne zadania. Jednym ze sposobów kategoryzacji algorytmów jest szkoła myślenia - metoda, która według grupy podobnie myślących myślicieli rozwiązałaby określony rodzaj problemu. Oczywiście istnieją inne sposoby kategoryzacji algorytmów, ale takie podejście ma tę zaletę, że pomaga lepiej zrozumieć zastosowania i orientacje algorytmów. Poniższe sekcje zawierają przegląd pięciu głównych technik algorytmicznych.

Symboliczne rozumowanie

Grupa zwana symbolistami polega na algorytmach, które wykorzystują rozumowanie symboliczne, aby znaleźć rozwiązanie problemów. Termin odwrotna dedukcja często pojawia się jako indukcja. W rozumowaniu symbolicznym dedukcja poszerza sferę ludzkiej wiedzy, podczas gdy indukcja podnosi poziom wiedzy ludzkiej. Indukcja zwykle otwiera nowe pola eksploracji, a dedukcja bada te pola. Jednak najważniejszą kwestią jest to, że indukcja jest naukową częścią tego typu rozumowania, podczas gdy dedukcja jest inżynierią. Obie strategie działają ręka w rękę, aby rozwiązać problemy, najpierw otwierając pole potencjalnych eksploracji w celu rozwiązania problemu, a następnie badając to pole, aby określić, czy w rzeczywistości go rozwiązuje. Jako przykład tej strategii, dedukcja mówi, że jeśli drzewo jest zielone, a zielone drzewa żyją, to drzewo musi być żywe. Myśląc o indukcji, powiedziałbyś, że drzewo jest zielone i że drzewo również żyje; dlatego zielone drzewa żyją. Indukcja dostarcza odpowiedzi na pytanie, jakiej wiedzy brakuje, biorąc pod uwagę znane dane wejściowe i wyjściowe.

Sieci neuronowe

Sieci neuronowe są dziełem grupy zwanej koneksjonistami. Ta grupa algorytmów stara się odtworzyć funkcje mózgu za pomocą krzemu zamiast neuronów. Zasadniczo każdy z neuronów (stworzony jako algorytm modelujący odpowiednik w świecie rzeczywistym) rozwiązuje niewielką część problemu, a użycie wielu neuronów równolegle rozwiązuje problem jako całość. Sieć neuronowa może zapewnić metodę korekty błędnych danych, a najpopularniejszą z nich jest propagacja wsteczna. Propagacja błędów, ma na celu określenie warunków, w których błędy są usuwane z sieci zbudowanych tak, aby przypominały ludzkie neurony, poprzez zmianę wag (ile danych wejściowych wpływa na wynik) i uprzedzeń (które cechy są wybierane) sieci. Celem jest kontynuacja zmiany wag i odchyleń do czasu, gdy rzeczywiste dane wyjściowe będą zgodne z docelowymi. W tym momencie sztuczny neuron odpala i przekazuje swoje rozwiązanie do następnego w kolejce neuronu. Rozwiązanie stworzone przez każdy pojedynczy neuron jest tylko częścią całego rozwiązania. Każdy neuron kontynuuje przekazywanie informacji do następnego neuronu w kolejce, aż grupa neuronów utworzy ostateczny wynik.

Algorytmy ewolucyjne

Grupa zwana ewolucjonistami w rozwiązywaniu problemów opiera się na zasadach ewolucji. Ta strategia opiera się na przetrwaniu najlepiej przystosowanych, usuwając wszelkie rozwiązania, które nie pasują do pożądanych wyników. Funkcja przystosowania określa zdolność każdej funkcji do rozwiązania problemu. Korzystając ze struktury drzewiastej, metoda rozwiązania wyszukuje najlepsze rozwiązanie w oparciu o dane wyjściowe funkcji. Zwycięzca każdego poziomu ewolucji może zbudować kolejny poziom funkcji. Następny poziom przybliży się do rozwiązania problemu, ale może go nie rozwiązać całkowicie, co oznacza, że potrzebny jest kolejny poziom. Ten konkretny typ algorytmiczny w dużym stopniu opiera się na rekurencji i językach które silnie wspierają rekurencję w celu rozwiązywania problemów. Ciekawym wynikiem tej strategii są algorytmy, które ewoluują same: jedna generacja algorytmów faktycznie tworzy następną generację.

Wnioskowanie bayesowskie

Bayesiści używają różnych metod statystycznych do rozwiązywania problemów. Biorąc pod uwagę, że metody statystyczne mogą stworzyć więcej niż jedno pozornie poprawne rozwiązanie, wybór funkcji staje się jednym z określania, która funkcja ma największe prawdopodobieństwo sukcesu. Na przykład, korzystając z tych technik, możesz zaakceptować zestaw symptomów jako dane wejściowe. Algorytm obliczy prawdopodobieństwo, że dana choroba będzie wynikiem objawów jako wynik. Biorąc pod uwagę, że wiele chorób ma te same objawy, prawdopodobieństwo jest ważne, ponieważ użytkownik zobaczy sytuacje, w których wynik o niższym prawdopodobieństwie jest w rzeczywistości prawidłowym wynikiem dla danych okoliczności. Ostatecznie algorytmy bayesowskie opierają się na założeniu, że nigdy nie można całkowicie ufać żadnej hipotezie (wynikowi, który ktoś ci dał) bez zapoznania się z dowodami użytymi do ich sformułowania (danymi wejściowymi, które druga osoba wykorzystała do sformułowania hipotezy). Analiza dowodów dowodzi lub obala hipotezę, którą potwierdzają. W związku z tym nie możesz określić, na jaką chorobę choruje ktoś, dopóki nie przetestujesz wszystkich objawów. Jednym z najbardziej rozpoznawalnych wyników z tej grupy algorytmów jest filtr spamu.

Systemy uczące się przez analogię

Analogizatory używają maszyn jądra do rozpoznawania wzorców w danych. Rozpoznając wzorzec jednego zestawu danych wejściowych i porównując go ze wzorcem znanego wyniku, można stworzyć rozwiązanie problemu. Celem jest wykorzystanie podobieństwa do określenia najlepszego rozwiązania problemu. Jest to rodzaj rozumowania, który określa, że użycie określonego rozwiązania zadziałało w danych okolicznościach w jakimś wcześniejszym czasie; dlatego też użycie tego rozwiązania w podobnych okolicznościach powinno również działać. Jednym z najbardziej rozpoznawalnych wyników

z tej grupy algorytmów są systemy rekomendujące. Na przykład, kiedy wchodzisz na Amazon i kupujesz produkt, system rekomendacji wymyśla inne powiązane produkty, które możesz również chcieć kupić.

Zagłębianie się w różne podejścia

Dobrze jest mieć kilka poglądów na algorytmy, aby zrozumieć, co robią i dlaczego to robią. W poprzedniej sekcji omówiono algorytmy oparte na grupach, które je utworzyły. Jednak masz inne podejścia, których możesz użyć do kategoryzacji algorytmów. Poniższa lista kategoryzuje niektóre popularne algorytmy według podobieństwa:

* Sztuczna sieć neuronowa: modeluje strukturę lub funkcję biologicznych sieci neuronowych (lub czasami robi jedno i drugie). Celem jest dopasowanie wzorców dla problemów związanych z regresją i klasyfikacją. Jednak technika ta naśladuje podejście stosowane przez organizmy biologiczne, zamiast ściśle polegać na podejściu opartym na prawdziwej matematyce. Oto przykłady algorytmów sztucznych sieci neuronowych:

• Perceptron

• Sieć neuronowa typu feed-forward

• Sieć Hopfield

• Radial Basis Function Network (RBFN)

• Samoorganizująca się mapa (SOM)

* Reguła asocjacji: wyodrębnia reguły, które pomagają wyjaśnić relacje między zmiennymi w danych. Możesz użyć tych reguł, aby odkryć przydatne skojarzenia w ogromnych zbiorach danych, które zwykle są łatwe do przeoczenia. Oto bardziej popularne algorytmy reguł asocjacyjnych:

• Algorytm Apriori

• Algorytm Eclat

* Bayesian: stosuje twierdzenie Bayesa do problemów z prawdopodobieństwem. Ta forma algorytmu ma zastosowanie w problemach klasyfikacji i regresji. Oto przykłady algorytmów Bayesa:

• Naïve Bayes

• Bayes naiwny Gaussa

• Wielomianowy Naïve Bayes

• Bayesian Belief Network (BBN)

• Sieć Bayesa (BN)

* Klastrowanie: opisuje model organizowania danych według klasy lub innych kryteriów. Wyniki często mają charakter centroidowy lub hierarchiczny. To, co widzisz, to relacje danych w sposób, który pomaga zrozumieć dane - to znaczy, jak wartości wpływają na siebie nawzajem. Poniższa lista zawiera przykłady algorytmów klastrowania:

• K-środki

• K-mediany

• Maksymalizacja oczekiwań (EM)

• Klastrowanie hierarchiczne

* Drzewo decyzyjne: konstruuje model decyzji w oparciu o rzeczywiste wartości znalezione w danych. Wynikowa struktura drzewa umożliwia bardzo szybkie wykonywanie porównań między nowymi danymi a istniejącymi danymi. Ta forma algorytmu jest często wykorzystywana do rozwiązywania problemów związanych z klasyfikacją i regresją. Poniższa lista przedstawia niektóre typowe algorytmy drzew decyzyjnych:

• Drzewo klasyfikacji i regresji (CART)

• Iteracyjny dychotomiser 3 (ID3)

• C4.5 i C5.0 (różne wersje zaawansowanego podejścia)

• Automatyczne wykrywanie interakcji chi-kwadrat (CHAID)

* Głębokie uczenie: zapewnia aktualizację sztucznych sieci neuronowych, które opierają się na wielu warstwach, aby wykorzystywać jeszcze większe zbiory danych i budować złożone sieci neuronowe. Ta szczególna grupa algorytmów sprawdza się dobrze w przypadku częściowo nadzorowanych problemów uczenia się, w których ilość oznaczonych danych jest minimalna. Oto kilka przykładów algorytmów uczenia głębokiego:

• Głęboka maszyna Boltzmanna (DBM)

• Sieci głębokich przekonań (DBN)

• Konwolucyjna sieć neuronowa (CNN)

• Powtarzalna sieć neuronowa (RNN)

• Stacked Auto-Encoders

* Redukcja wymiarowości: wyszukuje i wykorzystuje podobieństwa w strukturze danych w sposób podobny do algorytmów grupowania, ale z wykorzystaniem bez nadzoru metody. Celem jest podsumowanie lub opisanie danych przy użyciu mniejszej ilości informacji dzięki czemu zbiór danych staje się mniejszy i łatwiejszy w zarządzaniu. W niektórych przypadkach, ludzie używają tych algorytmów do rozwiązywania problemów związanych z klasyfikacją lub regresją. Tutaj jest lista typowych algorytmów redukcji wymiarowości:

• Analiza głównych komponentów (PCA)

• Analiza czynnikowa (FA)

• Wielowymiarowe skalowanie (MDS)

• t-Distributed Stochastic Neighbor Embedding (t-SNE)

* Zbiór: Łączy grupę wielu słabszych modeli w spójną całość, której indywidualne prognozy są w jakiś sposób łączone w celu zdefiniowania ogólnej prognozy. Korzystanie z zespołu może rozwiązać pewne problemy szybciej, wydajniej lub przy mniejszej liczbie błędów. Oto kilka typowych algorytmów zbiorczych:

• Wzmocnienie

• Agregacja Bootstrapped (pakowanie)

• AdaBoost

• Losowy las

• Maszyny do zwiększania gradientu (GBM)

* Oparte na instancjach: definiuje model problemów decyzyjnych, w którym dane szkoleniowe składają się z przykładów, które są później używane do celów porównawczych. Miara podobieństwa pomaga określić, kiedy nowe przykłady wypadają korzystnie w porównaniu z przykładami istniejącymi w bazie danych. Niektórzy nazywają te algorytmy uczeniem się opartym na pamięci lub zwycięzca bierze wszystko ze względu na sposób, w jaki działają. Poniższa lista zawiera niektóre typowe powiązane algorytmy z tą kategorią:

• K-Nearest Neighbors (KNN)

• Nauka kwantyzacji wektorów (LVQ)

* Regresja: Modeluje relacje między zmiennymi. Ta relacja jest iteracyjnie udoskonalana za pomocą miary błędu. Ta kategoria jest intensywnie wykorzystywana w statystycznym uczeniu maszynowym. Poniższa lista przedstawia algorytmy zwykle kojarzone z tego rodzaju algorytmem:

• Zwykła regresja metodą najmniejszych kwadratów (OLSR)

• Regresja logistyczna

* Regularyzacja: reguluje inne algorytmy, penalizując złożone rozwiązania i faworyzując prostsze. Ten rodzaj algorytmu jest często używany z metodami regresji. Celem jest zapewnienie, że rozwiązanie nie zgubi się w swojej własnej złożoności i dostarcza rozwiązania w określonym czasie przy użyciu jak najmniejszej liczby zasobów. Oto przykłady algorytmów regularyzacji:

• Regresja grzbietu

• Najmniej bezwzględny operator skurczu i wyboru (LASSO)

• Elastyczna siatka

• Regresja pod najmniejszym kątem (LARS)

* Maszyny wektorów pomocniczych (SVM): nadzorowane algorytmy uczenia się, które rozwiązują problemy klasyfikacji i regresji poprzez oddzielenie tylko kilku przykładów danych (zwanych podpórkami, stąd nazwa algorytmu) od reszty danych za pomocą funkcji. Po oddzieleniu tych podpór przewidywanie staje się łatwiejsze. Forma analizy zależy od typu funkcji (zwanego jądrem): podstawa liniowa, wielomianowa lub radialna. Oto przykłady algorytmów SVM.

• Maszyny wektorów nośnych liniowych

• Maszyny wektorowe wspomagające radialne funkcje bazowe

• Jednoklasowe maszyny wektorowe wspierające (do uczenia się bez nadzoru)

* Inne: masz do wyboru wiele innych algorytmów. Ta lista zawiera główne kategorie algorytmów. Niektóre kategorie, których nie ma na tej liście, należą do kategorii używanych do wyboru funkcji, dokładności algorytmów, miar wydajności i wyspecjalizowanych podpól uczenia maszynowego. Na przykład całe kategorie algorytmów są poświęcone tematowi widzenia komputerowego (CV) i przetwarzania języka naturalnego (NLP). Znajdziesz wiele innych kategorii algorytmów i możesz zacząć się zastanawiać, w jaki sposób analityk danych może dokonać dowolnego wyboru, a tym bardziej właściwego.

W oczekiwaniu na kolejny przełom

Przełomy wymagają cierpliwości, ponieważ komputery są z natury oparte na matematyce. Możesz nie widzieć ich jako takich podczas pracy z językiem wyższego poziomu, takim jak Python, ale wszystko, co dzieje się pod maską, wymaga ekstremalnego zrozumienia matematyki i danych, którymi manipuluje. W związku z tym można spodziewać się w przyszłości nowych zastosowań uczenia maszynowego i uczenia głębokiego, ponieważ naukowcy będą nadal znajdować nowe sposoby przetwarzania danych, tworzenia algorytmów i używania tych algorytmów do definiowania modeli danych. Niestety, praca z tym, co jest dostępne dzisiaj, nie wystarczy do tworzenia aplikacji jutra (pomimo tego, w co mogą wierzyć filmy). W przyszłości można spodziewać się postępu w sprzęcie, który sprawi, że aplikacje, które obecnie nie są wykonalne, będą całkiem wykonalne. Nie jest to tylko kwestia dodatkowej mocy obliczeniowej lub większej pamięci. Komputer jutra będzie miał dostęp do czujników, które nie są obecnie dostępne; procesory, które robią rzeczy, których dzisiejsze procesory nie potrafią; oraz metody sprawdzania, jak myślą komputery, których jeszcze nie przewidziano. Świat najbardziej potrzebuje teraz doświadczenia, a jego nagromadzenie zawsze wymaga czasu.

Zastanawianie się nad prawdziwymi zastosowaniami uczenia maszynowego

Fakt, że masz wiele opcji do wyboru, jeśli chodzi o sztuczną inteligencję, oznacza, że uczenie maszynowe nie jest jedyną technologią, którą powinieneś rozważyć, aby rozwiązać dany problem. Uczenie maszynowe doskonale pomaga w rozwiązywaniu określonych kategorii problemów. Aby określić, gdzie uczenie maszynowe działa najlepiej, musisz zacząć od rozważenia, w jaki sposób algorytm się uczy, a następnie zastosować tę wiedzę do zajęć z problemami, które musisz rozwiązać. Pamiętaj, że uczenie maszynowe polega na uogólnianiu, więc nie działa szczególnie dobrze w tych scenariuszach:

* Wynik musi zawierać dokładną odpowiedź, taką jak obliczenie podróży na Marsa.

* Możesz rozwiązać problem za pomocą uogólnień, ale inne techniki są prostsze, takie jak tworzenie oprogramowania do obliczania silni liczby.

* Nie masz dobrego uogólnienia problemu, ponieważ problem jest źle zrozumiany, nie ma określonego związku między danymi wejściowymi a wynikami lub domena problemu jest zbyt złożona.

W poniższych sekcjach omówiono rzeczywiste zastosowania uczenia maszynowego z perspektywy tego, jak się uczy, a następnie definiuje korzyści płynące z uczenia maszynowego w określonych domenach problemowych.

Zrozumienie zalet uczenia maszynowego

To, jak możesz skorzystać z uczenia maszynowego, zależy częściowo od środowiska, a częściowo od tego, czego od niego oczekujesz. Na przykład, jeśli spędzasz czas na zakupach produktów Amazon, możesz w pewnym momencie oczekiwać, że systemy uczące się wydadzą przydatne zalecenia na podstawie wcześniejszych zakupów. Te zalecenia dotyczą produktów, o których inaczej byś nie wiedział. Rekomendowanie produktów, których już używasz lub których już nie potrzebujesz, nie jest szczególnie przydatne, dlatego w grę wchodzi uczenie maszynowe. W miarę jak Amazon gromadzi więcej danych na temat twoich nawyków zakupowych, zalecenia powinny stać się bardziej przydatne, chociaż nawet najlepszy algorytm uczenia maszynowego nigdy nie odgadnie poprawnie twoich potrzeb za każdym razem. Oczywiście uczenie maszynowe przynosi wiele innych korzyści. Deweloper może wykorzystać uczenie maszynowe, aby dodać możliwość NLP do aplikacji. Naukowiec mógłby go użyć, aby znaleźć następne lekarstwo na raka. Możesz już używać go do filtrowania spamu w swoich wiadomościach e-mail lub polegać na nim, wsiadając do samochodu, jako część interfejsu głosowego.

Mając to na uwadze, następujące korzyści prawdopodobnie bardziej pasują do biznesowej perspektywy efektywnego korzystania z uczenia maszynowego, ale należy pamiętać, że istnieje również wiele innych sposobów:

* Uprość marketing produktów: Jednym z problemów, z którymi boryka się każda organizacja, jest określenie, co i kiedy sprzedawać, na podstawie preferencji klientów. Kampanie sprzedażowe są drogie, więc jedna porażka zwykle nie wchodzi w grę. Ponadto organizacja może znaleźć dziwne fragmenty informacji: klienci mogą lubić produkty w kolorze czerwonym, ale nie w kolorze zielonym. Wiedza o tym, czego chce klient, jest niesamowicie trudna, chyba że możesz przeanalizować ogromne ilości danych dotyczących zakupów, co jest czymś, co dobrze radzi sobie z uczeniem maszynowym.

* Dokładnie przewiduj przyszłą sprzedaż: działalność biznesowa może przypominać hazard, ponieważ nie możesz być pewien, że Twoje zakłady się opłacą. Uczenie maszynowe może śledzić sprzedaż minuta po minucie i śledzić trendy, zanim staną się oczywiste. Możliwość wykonywania tego rodzaju śledzenia oznacza, że możesz dokładniej dostroić kanały sprzedaży, aby zapewnić optymalne wyniki i upewnić się, że sklepy mają wystarczającą ilość odpowiednich produktów do sprzedaży. Nie przypomina to dokładnie patrzenia w kryształową kulę, ale jest blisko.

* Prognozowanie przestojów pracowników i innych pracowników: Co dziwne, niektóre organizacje mają problemy, ponieważ pracownicy wybierają najgorsze możliwe okresy nieobecności w pracy. W niektórych przypadkach te nieobecności wydają się nieprzewidywalne, na przykład potrzeby medyczne, podczas gdy w innych można je przewidzieć, na przykład nagła potrzeba osobistego czasu. Śledząc różne trendy z łatwo dostępnych źródeł danych, możesz śledzić nieobecności zarówno medyczne, jak i osobiste w całej branży, lokalizacji jako całości, a w szczególności w Twojej organizacji, aby upewnić się, że masz wystarczającą liczbę osób do podjęcia pracy. wykonane w dowolnym momencie.

* Zmniejsz liczbę błędów przy wprowadzaniu danych: Niektóre rodzaje błędów podczas wprowadzania danych można stosunkowo łatwo uniknąć, używając poprawnie funkcji formularza lub włączając moduł sprawdzania pisowni do aplikacji. Ponadto dodanie pewnych rodzajów dopasowania do wzorca może pomóc w zmniejszeniu liczby błędów związanych z pisaniem wielkich liter lub nieprawidłowych numerów telefonów. Uczenie maszynowe może przenieść redukcję błędów na inny poziom, prawidłowo identyfikując złożone wzorce, które zostaną pominięte przez inne techniki. Na przykład zamówienie klienta może wymagać jednej części A i dwóch części B, aby utworzyć całość. Dopasowanie wzorców dla tego rodzaju sprzedaży może być nieuchwytne, ale uczenie maszynowe może to umożliwić, zmniejszając liczbę błędów, które są szczególnie różne do znalezienia i wyeliminowania.

* Popraw reguły finansowe i precyzję modelowania: Utrzymanie finansów w dobrej kondycji może okazać się trudne w organizacji dowolnej wielkości. Uczenie maszynowe umożliwia wykonywanie zadań, takich jak zarządzanie portfelem, handel algorytmiczny, gwarantowanie pożyczek i wykrywanie oszustw z większą precyzją. Nie można wyeliminować udziału człowieka w takich przypadkach, ale współpraca człowieka i maszyny może stać się niezwykle wydajną kombinacją, która nie pozwoli, aby wiele błędów mogło przejść niezauważenie.

* Przewidywanie potrzeb konserwacyjnych: każdy system, który składa się z elementów fizycznych, prawdopodobnie wymaga różnego rodzaju konserwacji. Na przykład uczenie maszynowe może pomóc przewidzieć, kiedy system będzie wymagał czyszczenia na podstawie wcześniejszej wydajności i monitorowania środowiska. Możesz również zaplanować wymianę lub naprawę określonego sprzętu na podstawie wcześniejszych napraw i statystyk sprzętu. Rozwiązanie wykorzystujące systemy uczące się umożliwia nawet określenie, czy lepszym rozwiązaniem jest wymiana czy naprawa.

* Rozszerz interakcję z klientem i popraw satysfakcję: Klienci lubią czuć się wyjątkowo; w rzeczywistości każdy to robi. Jednak próba ręcznego stworzenia niestandardowego planu dla każdego klienta okazałaby się niemożliwa. W źródłach internetowych można znaleźć wiele informacji o klientach, w tym wszystko, od ostatnich zakupów po spójne nawyki zakupowe. Łącząc wszystkie te dane z dobrym rozwiązaniem uczenia maszynowego i personelem obsługi klienta, który ma wymagające oczy, można wydawać się, że osobiście stworzyłeś specjalne rozwiązanie dla każdego klienta, mimo że czas potrzebny na to jest minimalny.

Odkrywanie ograniczeń uczenia maszynowego

Granice technologii są często trudne do całkowitego oszacowania, ponieważ są one często wynikiem braku wyobraźni twórcy lub konsumenta tej technologii. Jednak uczenie maszynowe ma pewne wyraźne ograniczenia, które należy wziąć pod uwagę przed użyciem tej technologii do wykonania dowolnego zadania. Poniższa lista nie jest kompletna. W rzeczywistości możesz nawet nie do końca się z tym zgadzać, ale jest to dobry punkt wyjścia.

* Potrzeba ogromnych ilości danych szkoleniowych: w przeciwieństwie do wcześniejszych rozwiązań programowanych, uczenie maszynowe wymaga do trenowania ogromnych ilości danych. Wraz ze wzrostem złożoności problemu rośnie liczba punktów danych wymaganych do modelowania konkretnego problemu, co sprawia, że potrzeba jeszcze więcej danych. Chociaż ludzie generują coraz większe ilości danych w określonych domenach problemowych, a moc obliczeniowa potrzebna do przetwarzania tych danych również rośnie z każdym dniem, w niektórych domenach problemowych po prostu brakuje wystarczającej ilości danych lub wystarczającej mocy obliczeniowej, aby uczenie maszynowe było skuteczne.

* Etykietowanie danych jest żmudne i podatne na błędy: podczas korzystania z techniki nadzorowanego uczenia ktoś musi oznaczyć dane, aby zapewnić wartość wyjściową. Proces etykietowania ogromnych ilości danych jest zarówno żmudny, jak i czasochłonny, co czasami utrudnia uczenie maszynowe. Problem polega na tym, że człowiek może spojrzeć na dowolną liczbę przykładów czegoś takiego jak znak stopu i wiedzieć, że wszystkie są znakami stopu, ale komputer musi mieć każdy znak stopu indywidualnie oznaczony.

* Maszyny nie potrafią się wyjaśnić: w miarę jak systemy uczące się stają się bardziej elastyczne i wydajne; ilość ukrytych funkcji również się zwiększa. W rzeczywistości, gdy mamy do czynienia z rozwiązaniami uczenia głębokiego, okazuje się, że rozwiązanie zawiera jedną lub zwykle więcej ukrytych warstw, które tworzy rozwiązanie, ale ludzie nie poświęcili czasu na zbadanie. W związku z tym zarówno uczenie maszynowe (w pewnym stopniu), jak i uczenie głębokie (w większym stopniu) napotykają problemy, dla których przejrzystość jest ceniona i sprzeczna z niektórymi przepisami, takimi jak ogólne rozporządzenie o ochronie danych lub RODO (https://eugdpr.org /). Ponieważ proces staje się nieprzejrzysty, człowiek musi teraz przeanalizować proces, który ma być automatyczny. Potencjalnym rozwiązaniem tego problemu mogą być nowe strategie, takie jak Local Interpretable Model-Agnostic Explanations (LIME)

* Odchylenie sprawia, że wyniki są mniej użyteczne: algorytm nie jest w stanie stwierdzić, kiedy dane zawierają różne nieprawdy. W związku z tym uważa wszystkie dane za bezstronne i całkowicie zgodne z prawdą. W rezultacie każda analiza wykonana przez algorytm wyszkolony przy użyciu tych danych jest podejrzana. Problem staje się jeszcze większy, gdy sam algorytm jest stronniczy. W Internecie można znaleźć niezliczone przykłady algorytmów błędnie identyfikujących typowe obiekty, takie jak znaki stop, z powodu kombinacji danych zawierających nieprawdę i tendencyjne algorytmy.

* Systemy uczące się nie mogą współpracować: jedną z najważniejszych zalet bycia człowiekiem jest umiejętność współpracy z innymi. Potencjał wiedzy rośnie wykładniczo, ponieważ każda strona potencjalnego rozwiązania przekazuje swoją wiedzę, aby stworzyć całość, która jest znacznie większa niż suma jej części. Pojedyncze rozwiązanie do uczenia maszynowego pozostaje pojedynczym rozwiązaniem do uczenia maszynowego, ponieważ nie może uogólniać wiedzy, a tym samym przyczyniać się do kompleksowego rozwiązania z wieloma współpracującymi stronami.

Pobieranie i używanie Pythona

Głębokie uczenie się wymaga użycia kodu i masz do wyboru wiele języków. Jednak my opieramy się na Pythonie, ponieważ działa na wielu różnych platformach i cieszy się znacznym wsparciem w społeczności programistów. W rzeczywistości, według Tiobe Index , Python jest czwartym językiem na świecie i tym, który najlepiej sprawdzi się w przypadku głębokich uczenia się, według wielu źródeł .

Praca z Pythonem

Środowisko Pythona nieustannie się zmienia. W miarę jak społeczność Pythona wciąż ulepsza Pythona, język doświadcza przełomowych zmian - tych, które tworzą nowe zachowania przy jednoczesnym zmniejszeniu kompatybilności wstecznej. Te zmiany mogą nie być poważne, ale rozpraszają, które zmniejszą Twoją zdolność do odkrywania technik programowania głębokiego uczenia się. Oczywiście chcesz odkrywać głębokie uczenie się z jak najmniejszą ilością rozpraszaczy, więc posiadanie odpowiedniego środowiska jest niezbędne. Oto, czego potrzebujesz, aby używać Pythona u nas:

- Notatnik Jupyter wersja 5.5.0
- Środowisko Anaconda 3 w wersji 5.2.0
- Python w wersji 3.6.7

Jeśli nie masz tej konfiguracji, może się okazać, że przykłady nie działają zgodnie z przeznaczeniem. Zrzuty ekranu najprawdopodobniej będą się różnić, a procedury mogą nie działać zgodnie z planem. Podczas przeglądania musisz zainstalować różne pakiety Pythona, aby kod działał. Podobnie jak środowisko Python, które skonfigurujesz, pakiety te mają określone numery wersji. Jeśli używasz innej wersji pakietu, przykłady mogą w ogóle nie zostać wykonane. Ponadto możesz się sfrustrować, próbując przepracować komunikaty o błędach, które nie mają nic wspólnego z kodem, ale wynikają z użycia niewłaściwego numeru wersji. Zachowaj ostrożność podczas instalowania Anacondy, Jupyter Notebook, Pythona i wszystkich pakietów potrzebnych, aby Twoje doświadczenie głębokiego uczenia się było tak płynne, jak to tylko możliwe.

Uzyskiwanie kopii Anacondy

Zanim ruszysz dalej, musisz zdobyć i zainstalować kopię Anacondy. Tak, możesz pobrać i zainstalować Jupyter Notebook osobno, ale brakuje Ci też różnych innych aplikacji, które są dostarczane z Anacondą, takich jak Anaconda Prompt, która pojawia się w różnych częściach. Najlepszym pomysłem jest zainstalowanie Anacondy zgodnie z instrukcjami, które pojawiają się w poniższych sekcjach dla konkretnej platformy (Linux, MacOS lub Windows).

Uzyskiwanie Anakonda Continuum Analytics

Podstawowy pakiet Anaconda można pobrać bezpłatnie ze strony : https://repo.anaconda.com/archive/, aby uzyskać wersję 5.2.0. Po prostu kliknij jeden z linków Python 3.6 Version, aby uzyskać dostęp do darmowego produktu. Żądana nazwa pliku zaczyna się od Anaconda3-5.2.0-, po której następuje platforma i 32-bitowa lub 64-bitowa wersja, taka jak Anaconda3-5.2.0-Windows-x86_64.exe dla 64-bitowej wersji systemu Windows. Anaconda obsługuje następujące platformy:

»»Windows 32-bitowy i 64-bitowy (Instalator może zaoferować tylko wersję 64-bitową lub 32-bitową, w zależności od wykrytej wersji Windows.)

- Linux 32-bitowy i 64-bitowy
- Mac OS X 64-bitowy

Darmowy produkt to wszystko, czego potrzebujesz. Jednak, gdy spojrzysz na stronę, zobaczysz, że dostępnych jest wiele innych produktów dodatkowych. Te produkty mogą pomóc w tworzeniu solidnych aplikacji. Na przykład, dodając Accelerate do miksu, uzyskujesz możliwość wykonywania operacji wielordzeniowych i obsługujących GPU. Korzystanie z tych produktów dodatkowych wykracza poza zakres tej książki, ale witryna Anaconda zawiera szczegółowe informacje na temat ich używania.

Instalowanie Anacondy w systemie Linux

Używasz wiersza poleceń, aby zainstalować Anacondę w systemie Linux; nie istnieje graficzna opcja instalacji. Zanim będziesz mógł przeprowadzić instalację, musisz pobrać kopię oprogramowania Linux z witryny Continuum Analytics. Wymagane informacje o pobieraniu można znaleźć w sekcji „Pobieranie Anacondy Continuum Analytics" tego rozdziału. Poniższa procedura powinna działać poprawnie w każdym systemie Linux, niezależnie od tego, czy używasz 32-bitowej czy 64-bitowej wersji Anacondy:

1. Otwórz kopię Terminala.

Pojawi się okno Terminala.

2. Zmień katalogi na pobraną kopię Anacondy w swoim systemie.

Nazwa tego pliku jest różna, ale zwykle pojawia się jako Anaconda3-5.2.0-Linux-x86.sh dla systemów 32-bitowych i Anaconda3-5.2.0-Linux-x86_64.sh dla systemów 64-bitowych. Numer wersji jest osadzony jako część nazwy pliku. W tym przypadku nazwa pliku odnosi się do wersji 5.2.0, która jest wersją używaną w tej książce. W przypadku korzystania z innej wersji mogą wystąpić problemy z kodem źródłowym i konieczne jest wprowadzenie poprawek podczas pracy z nim.

3. Wpisz bash Anaconda3-5.2.0-Linux-x86 (dla wersji 32-bitowej) lub Anaconda 3-5.2.0-Linux-x86_64.sh (dla wersji 64-bitowej) i naciśnij Enter. Zostanie uruchomiony kreator instalacji z prośbą o zaakceptowanie warunków licencyjnych dotyczących korzystania z Anacondy.

4. Przeczytaj umowę licencyjną i zaakceptuj warunki, korzystając z metody wymaganej dla Twojej wersji systemu Linux. Kreator poprosi o podanie miejsca instalacji Anacondy. Książka zakłada, że używasz domyślnej lokalizacji ~/anaconda. Jeśli wybierzesz inną lokalizację, być może będziesz musiał zmodyfikować niektóre procedury w dalszej części , aby działały z twoją konfiguracją.

5. Podaj lokalizację instalacji (jeśli to konieczne) i naciśnij Enter (lub kliknij Dalej). Zobaczysz, że zaczyna się proces wyodrębniania aplikacji. Po zakończeniu wyodrębniania zobaczysz komunikat o zakończeniu.

6. Dodaj ścieżkę instalacji do instrukcji PATH, korzystając z metody wymaganej dla Twojej wersji systemu Linux.

Możesz zacząć używać Anacondy.

Instalowanie Anacondy na MacOS

Instalacja systemu Mac OS X jest dostępna tylko w jednej formie: 64-bitowej. Zanim będziesz mógł przeprowadzić instalację, musisz pobrać kopię oprogramowania Mac z witryny Continuum Analytics. Poniższe kroki pomogą Ci zainstalować 64-bitową Anacondę w systemie Mac:

1. Znajdź pobraną kopię Anacondy w swoim systemie. Nazwa tego pliku jest różna, ale zwykle pojawia się jako Anaconda3-5.2.0-MacOSX-x86_64.pkg. Numer wersji jest osadzony jako część nazwy pliku. W tym przypadku nazwa pliku odnosi się do wersji 5.2.0, która jest wersją używaną w tej książce. W

przypadku korzystania z innej wersji mogą wystąpić problemy z kodem źródłowym i konieczne jest wprowadzenie poprawek podczas pracy z nim.

2. Kliknij dwukrotnie plik instalacyjny.

Zobaczysz okno dialogowe wprowadzenia.

3. Kliknij Kontynuuj.

Kreator zapyta, czy chcesz przejrzeć materiały Read Me. Możesz przeczytać te materiały później. Na razie możesz spokojnie pominąć informacje.

4. Kliknij Kontynuuj.

Kreator wyświetla umowę licencyjną. Koniecznie przeczytaj umowę licencyjną, aby poznać warunki użytkowania.

5. Kliknij Zgadzam się, jeśli zgadzasz się z umową licencyjną.

Kreator poprosi o podanie miejsca docelowego instalacji. Miejsce docelowe kontroluje, czy instalacja dotyczy pojedynczego użytkownika, czy grupy. Możesz zobaczyć komunikat o błędzie informujący, że nie możesz zainstalować Anacondy w systemie. Komunikat o błędzie pojawia się z powodu błędu w instalatorze i nie ma nic wspólnego z twoim systemem. Aby pozbyć się komunikatu o błędzie, wybierz opcję Zainstaluj tylko dla mnie. Nie możesz zainstalować Anacondy dla grupy użytkowników w systemie Mac.

6. Kliknij Kontynuuj.

Instalator wyświetla okno dialogowe zawierające opcje zmiany typu instalacji. Kliknij Zmień lokalizację instalacji, jeśli chcesz zmienić lokalizację instalacji Anacondy w systemie. (Zakładamy, że używasz domyślnej ścieżki ~/anaconda.) Kliknij Dostosuj, jeśli chcesz zmodyfikować działanie instalatora. Na przykład możesz zdecydować, że nie chcesz dodawać Anacondy do swojej instrukcji PATH. Książka zakłada jednak, że wybrałeś domyślne opcje instalacji i nie masz powodu, aby je zmieniać, chyba że masz zainstalowaną inną kopię Pythona w innym miejscu.

7. Kliknij Zainstaluj.

Zobaczysz rozpoczęcie instalacji. Pasek postępu informuje o postępie procesu instalacji. Po zakończeniu instalacji pojawi się okno dialogowe zakończenia.

8. Kliknij Kontynuuj.

Możesz zacząć używać Anacondy.

Instalowanie Anacondy w systemie Windows

Anaconda jest dostarczana z graficzną aplikacją instalacyjną dla systemu Windows, więc uzyskanie dobrej instalacji oznacza użycie kreatora, podobnie jak w przypadku każdej innej instalacji. Oczywiście przed rozpoczęciem potrzebna jest kopia pliku instalacyjnego, a wymagane informacje o pobieraniu można znaleźć w sekcji „Pobieranie Anacondy Continuum Analytics” we wcześniejszej części tego rozdziału. Poniższa procedura powinna działać poprawnie w każdym systemie Windows, niezależnie od tego, czy używasz 32-bitowej czy 64-bitowej wersji Anacondy:

1. Znajdź pobraną kopię Anacondy w swoim systemie. Nazwa tego pliku jest różna, ale zwykle pojawia się jako Anaconda3-5.2.0-

Windows-x86.exe dla systemów 32-bitowych i Anaconda3-5.2.0-Windowsx86_64.exe dla systemów 64-bitowych. Numer wersji jest osadzony jako część nazwy pliku. W tym przypadku nazwa pliku odnosi się do wersji 5.2.0, która jest wersją używaną w tej książce. W przypadku korzystania z innej wersji mogą wystąpić problemy z kodem źródłowym i konieczne jest wprowadzenie poprawek podczas pracy z nim.

2. Kliknij dwukrotnie plik instalacyjny.

(Może pojawić się okno dialogowe Otwórz plik — Ostrzeżenie o zabezpieczeniach z pytaniem, czy chcesz uruchomić ten plik. Kliknij Uruchom, jeśli pojawi się to okno dialogowe.) Pojawi się okno dialogowe Konfiguracja Anacondy 5.2.0.

Dokładne okno dialogowe, które zobaczysz, zależy od wersji pobranego programu instalacyjnego Anaconda. Jeśli masz 64-bitowy system operacyjny, zawsze najlepiej jest używać 64-bitowej wersji Anacondy, aby uzyskać najlepszą możliwą wydajność. To pierwsze okno dialogowe informuje, kiedy masz 64-bitową wersję produktu.

3. Kliknij Dalej.

Kreator wyświetla umowę licencyjną. Koniecznie przeczytaj umowę licencyjną, aby poznać warunki użytkowania.

4. Kliknij Zgadzam się, jeśli zgadzasz się z umową licencyjną.

Zostaniesz zapytany, jaki rodzaj instalacji wykonać.

W większości przypadków chcesz zainstalować produkt tylko dla siebie. Wyjątkiem jest sytuacja, gdy z systemu korzysta wiele osób i wszyscy potrzebują dostępu do Anacondy. Wybór Just Me lub All Users wpłynie na folder docelowy instalacji w następnym kroku.

5. Wybierz jeden z typów instalacji, a następnie kliknij Dalej.

Kreator zapyta, gdzie zainstalować Anacondę na dysku, jak pokazano na rysunku 3-3. Książka zakłada, że używasz lokalizacji domyślnej, która zazwyczaj instaluje produkt w folderze C:\Users\<User Name>\Anaconda3. Jeśli wybierzesz inną lokalizację, być może będziesz musiał zmodyfikować niektóre procedury w dalszej części książki, aby działały z twoją konfiguracją. Możesz zostać zapytany, czy chcesz utworzyć folder docelowy. Jeśli tak, po prostu zezwól na tworzenie folderów.

6. Wybierz lokalizację instalacji (jeśli to konieczne), a następnie kliknij Dalej.

Zobaczysz Zaawansowane opcje instalacji. Te opcje są wybierane domyślnie i w większości przypadków nie ma powodu, aby je zmieniać. Być może będziesz musiał je zmienić, jeśli Anaconda nie zapewni domyślnej konfiguracji Pythona 3.6. Jednak zakładamy, że Anaconda została skonfigurowana przy użyciu domyślnych opcji.

Opcja Dodaj anakondę do mojej zmiennej środowiskowej PATH jest domyślnie wyłączona i należy ją pozostawić niezaznaczoną. Dodanie go do zmiennej środowiskowej PATH daje możliwość zlokalizowania plików Anacondy podczas korzystania ze standardowego wiersza polecenia, ale jeśli masz zainstalowanych wiele wersji Anacondy, dostępna jest tylko pierwsza zainstalowana wersja. Otwarcie zamiast tego komunikatu Anacondy jest znacznie lepsze, dzięki czemu uzyskasz dostęp do oczekiwanej wersji.

7. Zmień zaawansowane opcje instalacji (jeśli to konieczne), a następnie kliknij Zainstaluj. Zobaczysz okno dialogowe Instalowanie z paskiem postępu. Proces instalacji może potrwać kilka minut, więc weź

sobie filiżankę kawy i chwilę poczytaj komiks. Po zakończeniu procesu instalacji zobaczysz włączony przycisk Dalej.

8. Kliknij Dalej.

Kreator poinformuje Cię, że instalacja została zakończona

9. Kliknij Dalej.

Anaconda oferuje możliwość zintegrowania obsługi kodu Visual Studio. Nie potrzebujesz tego wsparcia dla tej książki, a dodanie go może zmienić sposób działania narzędzi Anaconda. Jeśli nie potrzebujesz absolutnie wsparcia Visual Studio, chcesz zachować czyste środowisko Anaconda.

10. Kliknij Pomiń.

Zobaczysz ekran zakończenia. Ten ekran zawiera opcje, aby dowiedzieć się więcej o Anaconda Cloud i uzyskać informacje o rozpoczęciu swojego pierwszego projektu Anaconda. Wybór tych opcji (lub odznaczenie ich) zależy od tego, co chcesz zrobić dalej, a opcje nie mają wpływu na konfigurację Anacondy.

11. Wybierz żądane opcje. Kliknij Zakończ.

Możesz zacząć używać Anacondy.

Uruchamianie notatnika Jupytera

Większość platform udostępnia ikonę umożliwiającą dostęp do notesu Jupyter. Wystarczy otworzyć tę ikonę, aby uzyskać dostęp do notatnika Jupyter. Na przykład w systemie Windows wybierasz Start ➪ Wszystkie programy ➪ Anaconda3 ➪ Jupyter Notebook. Dokładny wygląd w Twoim systemie zależy od używanej przeglądarki i rodzaju zainstalowanej platformy. Jeśli korzystasz z platformy, która nie oferuje łatwego dostępu za pomocą ikony, możesz wykonać następujące czynności, aby uzyskać dostęp do notesu Jupyter:

1. Otwórz okno wiersza polecenia Anacondy, wiersza polecenia lub terminala w systemie. Otworzy się okno, w którym możesz wpisywać polecenia.

2. Zmień katalogi na katalog \Anaconda3\Scripts na twojej maszynie. Większość systemów umożliwia użycie do tego zadania polecenia CD.

3. Wpisz ..\python Jupyter-script.py notebook i naciśnij Enter. W przeglądarce otworzy się strona Jupyter Notebook.

RÓŻNICA MIĘDZY NOTATNIKIEM A IDEA

Notatnik różni się od edytora tekstu tym, że koncentruje się na technice zaawansowanej przez informatyka Stanforda Donalda Knutha, zwanej programowaniem piśmiennym, która służy do tworzenia prezentacji kodu, notatek, równań matematycznych i grafiki. Krótko mówiąc, kończysz z notatnikiem naukowca pełnym wszystkiego, co jest potrzebne do pełnego zrozumienia kodu. Często spotykasz piśmienne techniki programowania używane w drogich pakietach, takich jak Mathematica i MATLAB. Rozwój notebooków wyróżnia się w

• Demonstracji

• Współpracy

• Badania

• Celu nauczania

• Prezentacji

Wykorzystujemy kolekcję narzędzi Anaconda, ponieważ nie tylko zapewnia wspaniałe wrażenia z kodowania w Pythonie, ale także pomaga odkryć ogromny potencjał technik programowania umiejącego czytać i pisać. Jeśli spędzasz dużo czasu wykonując zadania naukowe, Anaconda i podobne produkty są niezbędne. Ponadto Anaconda jest bezpłatna, dzięki czemu zyskujesz korzyści płynące z piśmiennego stylu programowania bez ponoszenia kosztów innych pakietów.

Zatrzymywanie serwera Jupyter Notebook

Bez względu na to, jak uruchomisz Notatnik Jupyter (lub po prostu Notatnik), system zazwyczaj otwiera wiersz polecenia lub okno terminala, aby hostować Notatnik. To okno zawiera serwer, dzięki któremu aplikacja działa. Po zamknięciu okna przeglądarki po zakończeniu sesji wybierz okno serwera i naciśnij Ctrl+C lub Ctrl+Break, aby zatrzymać serwer.

Definiowanie repozytorium kodu Kod, który utworzysz i użyjesz w tej książce, będzie przechowywany w repozytorium na dysku twardym. Pomyśl o repozytorium jako swego rodzaju szafie na dokumenty, w której umieszczasz swój kod. Notatnik otwiera szufladę, wyjmuje folder i pokazuje kod. Możesz go modyfikować, uruchamiać poszczególne przykłady w folderze, dodawać nowe przykłady i po prostu wchodzić w interakcję z kodem w naturalny sposób. Poniższe sekcje umożliwiają rozpoczęcie pracy z Notatnikiem, dzięki czemu możesz zobaczyć, jak działa cała koncepcja repozytorium.

Definiowanie folderu

Używasz folderów do przechowywania plików kodu dla konkretnego projektu. a przykład DL4D . Poniższe kroki pomogą Ci utworzyć nowy folder dla tej książki:

1. Wybierz Nowy ➪ Folder.

Notatnik utworzy dla Ciebie nowy folder. Nazwa folderu może się różnić, ale dla użytkowników systemu Windows jest po prostu wymieniona jako folder bez tytułu. Może być konieczne przewinięcie listy dostępnych folderów, aby znaleźć dany folder.

2. Zaznacz pole obok Folder bez tytułu.

3. Kliknij Zmień nazwę u góry strony.

Zobaczysz okno dialogowe Zmień nazwę katalogu, pokazane na rysunku 3-6.

4. Wpisz DL4D i naciśnij Enter.

Notatnik zmienia nazwę folderu za Ciebie.

Tworzenie nowego notatnika

Każdy nowy notatnik jest jak folder plików. Pojedyncze przykłady można umieścić w folderze plików, tak jak w przypadku kartek papieru w fizycznym folderze plików. Każdy przykład pojawia się w komórce. Możesz również umieścić inne rzeczy w folderze plików, ale zobaczysz, jak te rzeczy działają w miarę postępów w książce. Wykonaj poniższe czynności, aby utworzyć nowy notatnik:

1. Kliknij folder DL4D na stronie głównej.

Zobaczysz zawartość folderu projektu dla tej książki, który będzie pusty, jeśli wykonujesz to ćwiczenie od zera.

2. Wybierz Nowy ➪ Python 3.

W przeglądarce zostanie otwarta nowa karta z nowym notatnikiem, jak pokazano na rysunku 3-7. Zauważ, że Notatnik zawiera komórkę i że Notatnik wyróżnił tę komórkę, dzięki czemu można rozpocząć w niej wpisywanie kodu. Tytuł notatnika to teraz Bez tytułu. Nie jest to szczególnie pomocny tytuł, więc musisz go zmienić.

3. Kliknij Bez tytułu na stronie.

Notatnik pyta, jakiej nazwy chcesz użyć jako nowej nazwy, jak pokazano na rysunku 3-8.

4. Wpisz DL4D_Sample i naciśnij Enter.

Nowa nazwa mówi, że jest to plik, Sample.ipynb. Użycie tej konwencji nazewnictwa ułatwi odróżnienie tych plików od innych plików w repozytorium.

Eksportowanie notatnika

Tworzenie notatników i trzymanie ich wszystkich dla siebie nie jest zbyt zabawne. W pewnym momencie chcesz podzielić się nimi z innymi ludźmi. Aby wykonać to zadanie, musisz wyeksportować notatnik z repozytorium do pliku. Następnie możesz wysłać plik do innej osoby, która zaimportuje go do swojego repozytorium. W poprzedniej sekcji pokazano, jak utworzyć notatnik o nazwie DL4D_Sample. Możesz otworzyć ten notatnik, klikając jego wpis na liście repozytoriów. Plik zostanie ponownie otwarty, dzięki czemu możesz ponownie zobaczyć swój kod. Aby wyeksportować ten kod, wybierz Plik ➪ Pobierz jako ➪ Notatnik (.ipynb). To, co zobaczysz dalej, zależy od przeglądarki, ale zazwyczaj widzisz jakieś okno dialogowe do zapisania notesu jako pliku. Użyj tej samej metody do zapisywania pliku Jupyter Notebook, jak w przypadku każdego innego pliku zapisanego za pomocą przeglądarki.

Zapisywanie notatnika

W końcu chcesz zapisać swój notes, aby móc później przejrzeć kod i zaimponować znajomym, uruchamiając go po upewnieniu się, że nie zawiera żadnych błędów. Notatnik okresowo automatycznie zapisuje Twój notatnik. Aby jednak zapisać go ręcznie, wybierz Plik ➪ Zapisz i punkt kontrolny.

Zamykanie notatnika

Zdecydowanie nie powinieneś zamykać okna przeglądarki po zakończeniu pracy z notatnikiem. Spowoduje to prawdopodobnie utratę danych. Musisz wykonać uporządkowane zamknięcie pliku, co obejmuje zatrzymanie jądra używanego do uruchamiania kodu w tle. Po zapisaniu notesu możesz go zamknąć, wybierając kolejno opcje Plik ➪ Zamknij i zatrzymaj. Zobaczysz swój notatnik wprowadzony na listę notatników dla twojego folderu projektu.

Usuwanie notatnika

Czasami notatniki stają się nieaktualne lub po prostu nie musisz już z nimi pracować. Zamiast pozwalać na zapychanie się repozytorium plikami, których nie potrzebujesz, możesz usunąć te niechciane notatniki z listy. Wykonaj poniższe czynności, aby usunąć plik:

1. Zaznacz pole wyboru obok wpisu DL4D_Sample.ipynb.

2. Kliknij ikonę Usuń (kosz).

Zostanie wyświetlony komunikat ostrzegawczy dotyczący usuwania notebooka.

3. Kliknij Usuń.

Notatnik usuwa plik notatnika z listy.

Importowanie notatnika

Aby użyć kodu źródłowego, musisz zaimportować pobrane pliki do swojego repozytorium. Kod źródłowy znajduje się w pliku archiwum, który wyodrębniasz do lokalizacji na dysku twardym. Archiwum zawiera listę plików .ipynb (notatnik IPython) zawierających kod źródłowy tej książki (szczegółowe informacje na temat pobierania kodu źródłowego można znaleźć we wstępie). Poniższe kroki pokazują, jak zaimportować te pliki do swojego repozytorium:

1. Kliknij Prześlij na stronie Notebook DL4D.

To, co widzisz, zależy od Twojej przeglądarki. W większości przypadków zobaczysz jakiś rodzaj okna dialogowego Przesyłanie plików, które zapewnia dostęp do plików na dysku twardym.

2. Przejdź do katalogu zawierającego pliki, które chcesz zaimportować do Notatnika.

3. Zaznacz jeden lub więcej plików do zaimportowania, a następnie kliknij przycisk Otwórz (lub inny, podobny), aby rozpocząć proces przesyłania. Zobaczysz plik dodany do listy przesyłania. Plik nie jest jeszcze częścią repozytorium - po prostu wybrałeś go do przesłania.

4. Kliknij Prześlij.

Notatnik umieszcza plik w repozytorium, dzięki czemu możesz zacząć z niego korzystać.

Pobieranie i używanie zbiorów danych

Używamy wielu zestawów danych, z których niektóre można pobrać bezpośrednio z sieci, a inne pojawiają się w pakietach Pythona, takich jak biblioteka Scikit-learn. Te zestawy danych przedstawiają różne sposoby interakcji z danymi i są używane w przykładach do wykonywania różnych zadań. Poniższa lista zawiera szybki przegląd funkcji używanych do importowania zestawów danych ze Scikitlearn do kodu Pythona:

- load_boston(): Analiza regresji z zestawem danych cen domów w Bostonie
- load_iris(): Klasyfikacja za pomocą zestawu danych Iris
- load_digits([n_class]): Klasyfikacja za pomocą zestawu danych cyfr
- fetch_20newsgroups(subset='train'): Dane z 20 grup dyskusyjnych

Technika ładowania każdego z tych zestawów danych jest taka sama we wszystkich przykładach. Poniższy przykład pokazuje, jak załadować zestaw danych cen domów w Bostonie.

ze sklearn.datasets importuj load_boston

Boston = load_boston()

print(Boston.data.shape)

Aby zobaczyć, jak działa kod, kliknij Uruchom komórkę. Dane wyjściowe z wywołania drukowania to (506, 13). Możesz zobaczyć wyjście pokazane na rysunku 3-12. (Bądź cierpliwy; ładowanie zestawu danych może potrwać kilka sekund).

Tworzenie aplikacji

Sekcja „Tworzenie nowego notatnika” pokazuje, jak utworzyć pusty notatnik, co jest miłe, ale nie pomocne. Chcesz używać notatnika do przechowywania aplikacji, której możesz użyć do odkrywania wewnętrznych mechanizmów głębokiego uczenia się. W poniższych sekcjach pokazano, jak pracować z notatnikiem w sposób, który umożliwia tworzenie prostej aplikacji do dowolnego celu. Jednak zanim zaczniesz, upewnij się, że masz otwarty plik DL4D_Sample.ipynb, ponieważ jest on potrzebny do eksploracji Notatnika.

Zrozumienie komórek

Gdyby Notebook był standardowym IDE, nie miałbyś komórek. To, co masz, to dokument zawierający jedną, ciągłą serię stwierdzeń. Aby oddzielić różne elementy kodu, potrzebujesz oddzielnych plików. Komórki są różne, ponieważ każda komórka jest oddzielna. Tak, wyniki rzeczy, które robisz w poprzednich komórkach, mają znaczenie, ale jeśli komórka ma działać samodzielnie, możesz po prostu przejść do tej komórki i ją uruchomić. Aby zobaczyć, jak to działa dla siebie, wpisz następujący kod w pierwszej komórce przykładowego pliku DL4D_Sample_:

myVar = 3 + 4

drukuj(myVar)

Teraz kliknij Uruchom (strzałka skierowana w prawo). Kod zostanie wykonany i zobaczysz wynik. Wynik wynosi 7, zgodnie z oczekiwaniami. Zwróć jednak uwagę na wpis In [1]:. Ten wpis informuje, że jest to pierwsza wykonywana komórka. Teraz umieść kursor w drugiej komórce - tej, która jest obecnie pusta - i wpisz print("To jest myVar: ", myVar). Kliknij Uruchom. Dane wyjściowe pokazują, że komórki zostały wykonane pojedynczo (ponieważ wpis In [2]: pokazuje oddzielne wykonanie), ale myVar jest globalna dla notebooka. To, co robisz w innych komórkach z danymi, wpływa na każdą inną komórkę, bez względu na kolejność wykonania.

Dodawanie komórek dokumentacji

Komórki występują w wielu różnych formach. My nie wykorzystujemy ich wszystkich. Jednak wiedza o tym, jak korzystać z komórek dokumentacji, może się przydać. Wybierz pierwszą komórkę (obecnie oznaczoną 1). Wybierz Wstaw ➪ Wstaw komórkę powyżej. Zobaczysz nową komórkę dodaną do notatnika. Zwróć uwagę na listę rozwijaną, która aktualnie zawiera kod. Ta lista pozwala wybrać rodzaj tworzonej komórki. Wybierz Markdown z listy i wpisz # Tworzenie aplikacji (aby utworzyć nagłówek poziomu 1). Kliknij Uruchom (co może wydawać się bardzo dziwne, ale spróbuj). Widzisz, że nagłówek zamienia się w rzeczywisty nagłówek z ciemniejszym, większym tekstem. Być może myślisz teraz, że te specjalne komórki działają jak strony HTML i masz rację. Wybierz Wstaw ➪ Wstaw komórkę poniżej, wybierz Markdown z listy rozwijanej, a następnie wpisz ## Rozumienie komórek (aby utworzyć nagłówek poziomu 2). Kliknij Uruchom. Lliczba znaków haszujących (#) dodanych do tekstu wpływa na poziom nagłówka, ale znaki haszujące nie pojawiają się w rzeczywistym nagłówku.

Korzystanie z innych typów komórek

Ta część nie przedstawia wszystkich rodzajów zawartości komórek, które można zobaczyć za pomocą Notatnika. Możesz jednak dodawać również inne elementy, takie jak grafikę, do swoich notatników. Kiedy nadejdzie czas, możesz wydrukować (wydrukować) swój notatnik jako raport i wykorzystać go w różnego rodzaju prezentacjach. Technika programowania umiejącego czytać i pisać różni się od tej, której być może używałeś w przeszłości, ale ma zdecydowane zalety, jak zobaczysz w kolejnych częściach.

Zrozumienie użycia wcięcia

Podczas pracy z przykładami zauważysz, że niektóre wiersze są wcięte. W rzeczywistości przykłady zawierają również sporo odstępów (takich jak dodatkowe wiersze między wierszami kodu). Python ignoruje wszelkie wcięcia w Twojej aplikacji. Głównym powodem dodania wcięcia jest zapewnienie wizualnych wskazówek dotyczących kodu. W ten sam sposób, w jaki używane jest wcięcie w konspektach książek, wcięcie w kodzie pokazuje relacje między różnymi elementami kodu. Różne zastosowania wcięć staną się bardziej znajome, gdy będziesz przerabiać przykłady. Powinieneś jednak wiedzieć na początku, dlaczego używane jest wcięcie i jak jest ono wprowadzane. W tym celu nadszedł czas na kolejny przykład. Poniższe kroki pomogą Ci stworzyć nowy przykład, który używa wcięć, aby związek między elementami aplikacji był dużo bardziej widoczny i

łatwiej rozgryźć później.

1. Wybierz Nowy ➪ Python3.

Jupyter Notebook tworzy dla Ciebie nowy notatnik. Źródło do pobrania używa nazwy pliku DL4D_Indentation.ipynb, ale możesz użyć dowolnej nazwy.

2. Wpisz print(„To jest naprawdę długa linia tekstu, która będzie ” +. Widzisz tekst wyświetlany normalnie na ekranie, tak jak oczekiwałeś. Znak plus (+) mówi Pythonowi, że jest dodatkowy tekst do wyświetlenia. Dodawanie tekstu z wielu wierszy razem w jeden długi fragment tekstu nazywa się konkatenacją. Więcej informacji o korzystaniu z tej funkcji znajdziesz w dalszej części książki, więc nie musisz się tym teraz martwić.

3. Naciśnij Enter.

Punkt wstawiania nie wraca na początek wiersza, jak można się spodziewać. Zamiast tego kończy się bezpośrednio pod pierwszym podwójnym cudzysłowem. Ta funkcja, zwana automatycznym wcięciem, jest jedną z cech odróżniających zwykły edytor tekstu od edytora przeznaczonego do pisania kodu.

4. Wpisz „występują w wielu wierszach w pliku kodu źródłowego.”) i naciśnij Enter. Zauważ, że punkt wstawiania wraca na początek wiersza. Gdy program Notebook wykryje, że doszedłeś do końca kodu, automatycznie przesunie tekst do oryginalnej pozycji.

5. Kliknij Uruchom.

Zobaczysz wynik pokazany na rysunku 3-16. Mimo że tekst pojawia się w wielu wierszach w pliku kodu źródłowego, pojawia się tylko w jednym pliku wyjściowym. Linia pęka ze względu na rozmiar okna, ale w rzeczywistości jest to tylko jedna linia.

Dodawanie komentarzy

Ludzie cały czas tworzą dla siebie notatki. Kiedy musisz kupić artykuły spożywcze, przeglądasz swoje szafki, określasz, czego potrzebujesz, i zapisujesz to na liście lub mówisz do aplikacji w telefonie. Po dotarciu do sklepu przeglądasz swoją listę, aby zapamiętać, czego potrzebujesz. Korzystanie z notatek przydaje się do wszelkiego rodzaju potrzeb, takich jak śledzenie przebiegu rozmowy między partnerami biznesowymi lub zapamiętywanie istotnych punktów wykładu. Ludzie potrzebują notatek, aby pobudzić pamięć. Komentarze w kodzie źródłowym to tylko inna forma notatki. Dodajesz je do kodu, aby móc później zapamiętać, jakie zadanie kod wykonuje. Poniższe sekcje opisują komentarze bardziej szczegółowo.

NAGŁÓWKI A KOMENTARZE

Nagłówki i komentarze na początku mogą być nieco mylące. Nagłówki pojawiają się w oddzielnych komórkach; komentarze pojawiają się wraz z kodem źródłowym. Służą różnym celom. Nagłówki

informują o całej grupie kodu, a poszczególne komentarze informują o poszczególnych krokach kodu, a nawet wierszach kodu. Nawet jeśli używasz obu do dokumentacji, każdy z nich służy unikalnemu celowi. Komentarze są zazwyczaj bardziej szczegółowe niż nagłówki.

Zrozumienie komentarzy

Komputery potrzebują specjalnego sposobu, aby ustalić, czy tekst, który piszesz, jest komentarzem, a nie kodem do wykonania. Python udostępnia dwie metody definiowania tekstu jako komentarza, a nie jako kodu. Pierwsza metoda to komentarz jednowierszowy. Używa znaku liczby (#), w ten sposób:

```
# To jest komentarz.

print("Witaj z Pythona!") #To też jest komentarz.
```

Komentarz jednowierszowy może pojawić się w wierszu sam lub po kodzie wykonywalnym. Pojawia się tylko w jednej linii. W przypadku krótkiego tekstu opisowego, takiego jak wyjaśnienie konkretnego fragmentu kodu, zazwyczaj używasz komentarza jednowierszowego. Notatnik pokazuje komentarze w charakterystycznym kolorze (zazwyczaj niebieskim) i kursywą. Python w rzeczywistości nie obsługuje bezpośrednio komentarza wielowierszowego, ale możesz go utworzyć za pomocą łańcucha w potrójnym cudzysłowie. Komentarz wielowierszowy zarówno zaczyna się, jak i kończy trzema podwójnymi cudzysłowami (""") lub trzema pojedynczymi cudzysłowami (''') w następujący sposób:

```
"""

Aplikacja: Comments.py

Napisane przez: John

Cel: Pokazuje, jak używać komentarzy.

"""
```

Te wiersze nie są wykonywane. Python nie wyświetli komunikatu o błędzie, gdy pojawią się w twoim kodzie. Notebook traktuje je jednak inaczej. Zauważ, że rzeczywiste komentarze Pythona, te poprzedzone znakiem hash (#) w komórce 1, nie generują żadnych danych wyjściowych. Jednak ciągi w potrójnym cudzysłowie generują dane wyjściowe. Ponadto, w przeciwieństwie do standardowych komentarzy, tekst w potrójnym cudzysłowie jest wyświetlany na czerwono (w zależności od edytora), a nie na niebiesko, a tekst nie jest pisany kursywą. Jeśli planujesz wydrukować notes jako raport, musisz unikać ciągów w potrójnym cudzysłowie. (Niektóre środowiska IDE, takie jak IDLE, całkowicie ignorują ciągi w potrójnym cudzysłowie.) Zwykle używa się komentarzy wielowierszowych, aby uzyskać dłuższe wyjaśnienia, kto utworzył aplikację, dlaczego została utworzona i jakie zadania wykonuje. Oczywiście nie ma sztywnych zasad dotyczących tego, w jaki sposób używasz komentarzy. Głównym celem jest dokładne poinformowanie komputera, co jest, a co nie jest komentarzem, aby nie próbował wchodzić w interakcje z komentarzem tak, jak kodowałby.

Używanie komentarzy do pozostawiania sobie przypomnień

Wiele osób tak naprawdę nie rozumie komentarzy i nie do końca wie, co zrobić z notatkami w kodzie. Pamiętaj, że możesz napisać kawałek kodu dzisiaj, a potem nie patrzeć na niego przez lata. Potrzebujesz notatek, aby pobudzić swoją pamięć, abyś pamiętał, jakie zadanie wykonuje kod i dlaczego go napisałeś. Oto kilka typowych powodów, dla których warto używać komentarzy w kodzie:

- Przypomnij sobie, co robi kod i dlaczego go napisałeś
- Powiedz innym, jak dbać o swój kod

- Udostępniaj swój kod innym programistom
- Lista pomysłów na przyszłe aktualizacje
- Podaj listę źródeł dokumentacji, których użyłeś do napisania kodu
- Utrzymuj listę wprowadzonych ulepszeń

Możesz używać komentarzy na wiele innych sposobów, ale są to najczęstsze sposoby. Przyjrzyj się, w jaki sposób komentarze są używane w przykładach w książce, zwłaszcza gdy przejdziesz do późniejszych rozdziałów, w których kod staje się bardziej złożony. W miarę jak Twój kod staje się coraz bardziej złożony, musisz dodawać więcej komentarzy i wprowadzać komentarze związane z tym, o czym musisz o tym pamiętać.

Używanie komentarzy, aby uniemożliwić wykonanie kodu

Deweloperzy czasami używają również funkcji komentowania, aby zapobiec wykonywaniu wierszy kodu (nazywane komentowaniem). Może być konieczne wykonanie tej czynności w celu ustalenia, czy wiersz kodu powoduje awarię aplikacji. Podobnie jak w przypadku każdego innego komentarza, możesz użyć komentowania jednowierszowego lub wielowierszowego. Jednak podczas korzystania z komentowania wielowierszowego, jako część danych wyjściowych widzisz kod, który nie jest wykonywany (i faktycznie może być pomocne sprawdzenie, gdzie kod wpływa na dane wyjściowe).

Uzyskiwanie pomocy z językiem Python

Ta książka nie nauczy Cię języka Python, który sam w sobie wymagałby całej książki. Oto najlepsze metody uzyskania pomocy:

- Wybierz jedną z opcji w menu Pomoc Notatnika.
- Otwórz monit Anacondy, uruchom kopię Pythona i użyj poleceń tekstowych, aby wyszukać pomoc.
- Pobierz dokumentację Pythona z https://docs.python.org/3.6/download.html.
- Przejrzyj dokumentację online na https://docs.python.org/3.6/.
- Skorzystaj z jednego z poniższych samouczków:

• Oficjalny samouczek: https://docs.python.org/3.6/

• TutorialsPoint: https://www.tutorialspoint.com/python/

• Szkoły W3Schools: https://www.w3schools.com/python/

• learnpython.org: https://www.learnpython.org/

• Akademia kodowania: https://www.codecademy.com/learn/learn-python

Chodzi o to, że ta książka zakłada, że umiesz już programować w Pythonie. Ten rozdział zawiera pewne pomoce związane z narzędziami, które ułatwiają przejście od narzędzi, których używałeś w przeszłości, do narzędzi używanych w tej książce.

Praca w chmurze

Mimo że przedstawiono podejście do przetwarzania lokalnego, może zaistnieć potrzeba interakcji z zasobami w chmurze w celu wykonania określonych zadań. W poniższych sekcjach omówiono dwie czynności związane z chmurą, które możesz wykonać podczas korzystania z tej książki. Pierwszym z nich jest dostęp do zasobów w chmurze dla różnych potrzeb. Drugi to użycie Google Colaboratory do pracy z przykładami na tablecie zamiast na komputerze stacjonarnym.

Korzystanie z zestawów danych i jąder Kaggle

Kaggle (https://www.kaggle.com/) to ogromna społeczność naukowców zajmujących się danymi i innych, którzy muszą pracować z dużymi zestawami danych, aby uzyskać informacje potrzebne do osiągnięcia różnych celów. Możesz tworzyć nowe projekty na Kaggle, przeglądać pracę wykonaną przez innych nad ukończonymi projektami lub brać udział w jednym z trwających konkursów. Jednak Kaggle to coś więcej niż tylko społeczność naprawdę inteligentnych ludzi, którzy lubią bawić się danymi; to także miejsce, w którym możesz uzyskać zasoby potrzebne do nauki wszystkiego o głębokim uczeniu się i tworzenia własnych projektów. Najlepszym miejscem, aby dowiedzieć się, jak Kaggle może pomóc Ci odkryć więcej o głębokim uczeniu się, jest https://www.kaggle.com/m2skills/datasets-and-tutorialkernels-dla początkujących. Ta witryna zawiera listę różnych zestawów danych i jąder samouczków udostępnianych przez Kaggle. Zbiór danych to po prostu rodzaj bazy danych informacji używanej do wykonywania standardowych testów kodu aplikacji. Jądro samouczka to rodzaj projektu, którego używasz, aby nauczyć się analizować dane na różne sposoby. Na przykład, możesz znaleźć jądro samouczka o klasyfikacji grzybów na https://www.kaggle.com/uciml/klasyfikacja grzybów.

Korzystanie z Google Colaboratory

Colaboratory (https://colab.research.google.com/notebooks/welcome.ipynb) lub w skrócie Colab to usługa Google oparta na chmurze, która replikuje Jupyter Notebook w chmurze. Jest to implementacja niestandardowa, więc może się zdarzyć, że Colab i Notatnik nie będą zsynchronizowane — funkcje w jednym mogą nie zawsze działać w drugim. Nie musisz niczego instalować w swoim systemie, aby z niego korzystać. W większości przypadków używasz Colab tak samo, jak instalacja Jupyter Notebook na komputerze stacjonarnym. Głównym powodem, dla którego warto dowiedzieć się więcej o Colab, jest chęć korzystania z urządzenia innego niż standardowa konfiguracja komputera stacjonarnego do pracy z przykładami. Na razie ta sekcja zawiera podstawy korzystania z istniejących plików. Możesz otwierać istniejące notatniki znalezione w pamięci lokalnej, na Dysku Google lub w serwisie GitHub. Możesz także otworzyć dowolny z przykładów Colab lub przesłać pliki ze źródeł, do których masz dostęp, takich jak dysk sieciowy w systemie. We wszystkich przypadkach należy rozpocząć od wyboru Plik ⇨ Otwórz notatnik. Widok domyślny pokazuje wszystkie ostatnio otwierane pliki, niezależnie od lokalizacji. Pliki są wyświetlane w kolejności alfabetycznej. Możesz filtrować liczbę wyświetlanych elementów, wpisując ciąg w filtrze notesów. U góry znajdują się inne opcje otwierania zeszytów. Nawet jeśli nie jesteś zalogowany, nadal możesz uzyskać dostęp do przykładowych projektów Colab. Te projekty pomagają zrozumieć Colab, ale nie pozwalają na robienie czegokolwiek z własnymi projektami. Mimo to nadal możesz eksperymentować z Colab bez uprzedniego logowania się do Google. Oto krótka lista sposobów korzystania z plików w Colab:

- Korzystanie z Dysku dla istniejących notebooków: Dysk Google jest domyślną lokalizacją dla wielu operacji w Colab i zawsze możesz wybrać go jako miejsce docelowe. Podczas pracy z Dyskiem zobaczysz listę plików. Aby otworzyć określony plik, kliknij jego łącze w oknie dialogowym. Plik otworzy się w bieżącej zakładce Twojej przeglądarki.
- Korzystanie z GitHub dla istniejących notebooków: Podczas pracy z GitHub, najpierw musisz podać lokalizację kodu źródłowego online. Lokalizacja musi wskazywać na projekt publiczny; nie możesz korzystać z Colab, aby uzyskać dostęp do swoich prywatnych projektów. Po nawiązaniu połączenia z GitHub zobaczysz listę repozytoriów (które są kontenerami na kod związany z konkretnym projektem) oraz gałęzi (które reprezentują poszczególne implementacje kodu). Wybranie repozytorium i oddziału wyświetla listę plików notatników, które można załadować do Colab. Po prostu kliknij wymagany link i ładuje się tak, jakbyś korzystał z Dysku Google.

- Korzystanie z pamięci lokalnej dla istniejących notatników: Jeśli chcesz użyć źródła do pobrania dla tej książki lub dowolnego źródła lokalnego, wybierz kartę Prześlij w oknie dialogowym. Pośrodku zobaczysz pojedynczy przycisk o nazwie Wybierz plik. Kliknięcie tego przycisku otwiera okno dialogowe Otwórz plik w przeglądarce. Zlokalizuj plik, który chcesz przesłać, tak jak zwykle w przypadku każdego pliku, który chcesz otworzyć. Wybranie pliku i kliknięcie Otwórz powoduje przesłanie pliku na Dysk Google. Jeśli wprowadzisz zmiany w pliku, zmiany te pojawią się na Dysku Google, a nie na dysku lokalnym.

Wykorzystanie struktury głębokiego uczenia się

Przyjrzymy się ramom uczenia głębokiego, ponieważ korzystanie z platformy uczenia głębokiego może znacznie skrócić czas, koszty i złożoność tworzenia rozwiązania do uczenia głębokiego. Oczywiście musisz zacząć od zdefiniowania terminu framework, który jest abstrakcją, która zapewnia ogólną funkcjonalność, którą modyfikuje kod aplikacji. W przeciwieństwie do biblioteki działającej w Twojej aplikacji, gdy używasz frameworka, Twoja aplikacja działa w nim. Nie możesz modyfikować podstawowej funkcjonalności frameworka, co oznacza, że masz stabilne środowisko do pracy, ale większość frameworków oferuje pewien poziom rozszerzalności. Struktury są zazwyczaj dostosowane do konkretnych potrzeb, takich jak struktury internetowe używane do tworzenia aplikacji online. W związku z tym, mimo że frameworki do uczenia głębokiego mają wiele cech charakterystycznych dla frameworków, zapewniają one również określone funkcje, które omawiamy. Nie wszyscy wykorzystują te same pomysły i koncepcje do uruchamiania aplikacji do uczenia głębokiego. Ponadto nie każda organizacja chce inwestować w złożoną strukturę uczenia głębokiego, gdy wystarczy tańsza i prostsza struktura. W związku z tym można znaleźć wiele struktur głębokiego uczenia się, które mogą zapewnić podstawową funkcjonalność, której można używać do eksperymentowania i do prostszych aplikacji. Ten rozdział bada niektóre z tych podstawowych struktur i porównuje je, aby mieć lepsze pojęcie o tym, co jest dostępne. Aby zapewnić najlepsze możliwe środowisko do nauki, w tej książce w przykładach wykorzystano framework TensorFlow. TensorFlow działa lepiej w sytuacjach przedstawionych u nas niż inne rozwiązania omówione wcześniej, a ta część wyjaśnia, dlaczego. Mówi również dokładnie, dlaczego TensorFlow jest dobrym ogólnym rozwiązaniem dla wielu scenariuszy głębokiego uczenia się.

Prezentacja frameworka

Jak wspomniano we wstępie, Twój kod działa w ramach frameworka. W środowisku frameworka Twój kod wysyła żądania do frameworka, który następnie spełnia je za Ciebie. W konsekwencji frameworki zapewniają rodzaj struktury do tworzenia aplikacji. Ze względu na tę strukturę, frameworki są specyficzne dla domeny, odpowiadając na określone rodzaje potrzeb związanych z tworzeniem aplikacji. W poniższych sekcjach omówiono struktury zarówno z perspektywy przeglądu, jak i bardziej szczegółowo jako rozwiązanie do uczenia głębokiego. Ważne jest, aby pamiętać, że te sekcje nie dostarczają pełnych informacji na temat frameworków, ale pomagają zrozumieć frameworki głębokiego uczenia się na tyle dobrze, aby podejmować dobre decyzje na ich temat.

Definiowanie różnic

Specyficzna dla domeny problematyka frameworków sprawia, że konieczne jest znalezienie odpowiedniego rodzaju frameworka dla Twoich potrzeb. (Domena problemowa to opis wiedzy i zasobów wymaganych do rozwiązania problemu. Na przykład nie idziesz do lekarza, aby rozwiązać swoje problemy z hydrauliką – zamiast tego udajesz się do hydraulika.) Zwyciężyło proste zapytanie o ogólne ramy nie zrobi ci wiele dobrego. Oto kilka przykładów typów ram, z których wszystkie mają specyficzne cechy, aby zaspokoić potrzeby ich problematycznej domeny:

- Struktura aplikacji (w rodzaju używanych do tworzenia aplikacji dla użytkowników końcowych)
- Artystyczne (rysunek, muzyka i inne formy twórcze)
- Cactus framework (wysokowydajne obliczenia naukowe)
- System wspomagania decyzji
- Modelowanie systemów ziemnych
- Modelowanie finansowe
- Struktura internetowa (w tym frameworki specyficzne dla języków takich jak AJAX i JavaScript)

Różnorodność frameworków oprogramowania jest niesamowita i prawdopodobnie nigdy nie będziesz ich potrzebować. Łączą je dwie ważne rzeczy. W każdym przypadku framework definiuje serię zamrożonych punktów, które definiują charakterystykę aplikacji i których programista nie może zmienić. Ponadto struktura definiuje punkty aktywne, których programista używa do definiowania specyfiki oprogramowania docelowego. Na przykład zamrożony punkt w aplikacji internetowej może definiować interfejs, na którym użytkownik polega na składaniu żądań, podczas gdy punkt aktywny może określać sposób realizacji tego żądania. Ktoś projektując aplikację do wyszukiwania książek skupiłby się na specyfice wyszukiwania książek, pomijając wymagania zarządzania państwem i obsługi wniosków.

Wyjaśnienie popularności frameworków

Myśląc o oprogramowaniu można łatwo zauważyć progres narzędzi wykorzystywanych do jego tworzenia. Kiedyś programiści musieli wprowadzać swój kod za pomocą kart z dziurkaczem, co było niezwykle czasochłonne i podatne na błędy. Edytory ułatwiają pracę, ponieważ teraz możesz wpisywać, co chcesz zrobić. Następne jest zintegrowane środowisko programistyczne (IDE). Korzystanie ze środowiska IDE umożliwia modelowanie, kompilację i testowanie kodu w jednym środowisku, a także wiele innych rzeczy. Wykorzystanie bibliotek umożliwia szybkie tworzenie dużych, złożonych aplikacji. Tak więc framework – który jest środowiskiem, w którym programista musi brać pod uwagę tylko specyfikacje konkretnej aplikacji – jest po prostu kolejnym krokiem w zwiększaniu produktywności programistów, a jednocześnie sprawia, że aplikacje są bardziej niezawodne i mniej podatne na błędy. Stąd popularność frameworków wśród programistów. Jednak framework to znacznie więcej niż tylko sposób na szybsze tworzenie kodu, przy mniejszym wysiłku i mniejszej liczbie błędów. Framework umożliwia tworzenie znormalizowanego środowiska, w którym wszyscy korzystają z tych samych bibliotek, narzędzi, interfejsów programowania aplikacji (API) i innych programów. Korzystanie ze znormalizowanego środowiska umożliwia przenoszenie kodu między systemami bez obawy o wprowadzenie dziwnych problemów z aplikacjami z powodu niespójności środowiskowych. Ponadto problemy z rozwojem zespołu są mniejsze, ponieważ środowisko współpracy jest uproszczone. Ponieważ framework obsługuje wszystkie szczegóły niskiego poziomu, należy również wziąć pod uwagę skład zespołu aplikacyjnego. W przeszłości zespół mógł potrzebować osób biegłych w interakcji ze sprzętem lub tworzeniu podstaw interfejsu użytkownika. Zastosowanie frameworka oznacza, że wszystkie te zadania są już wykonane, dlatego zespół składa się z ekspertów merytorycznych, którzy potrafią skutecznie komunikować się ze sobą, umożliwiając spójne podejście do tworzenia aplikacji. Najważniejszym powodem, dla którego frameworki są teraz tak popularne, jest sposób, w jaki kodowanie jest obecnie wykonywane. Kiedyś programiści musieli wiedzieć, jak komunikować się ze sprzętem i oprogramowaniem na bardzo niskim poziomie. Obecnie frameworki ułatwiają kodowanie w środowisku, w którym:

- Większość aplikacji składa się głównie z wywołań API połączonych ze sobą w celu osiągnięcia określonego celu.
- Ludzie muszą rozumieć, jak działają interfejsy API, a nie co robią lub jak to robią. Deweloper musi zastanowić się, jakie struktury danych akceptuje API i jak dobrze przetwarza dane pod presją.
- Ogromna zainstalowana baza istniejącego oprogramowania oznacza utrzymywanie tego kodu na miejscu i znajdowanie szybkich, wydajnych metod interakcji z nim.
- Nacisk kładziony jest na architekturę, a nie na detale. Ponieważ większość nowych aplikacji w dużym stopniu opiera się na istniejącym kodzie, do którego dostęp uzyskuje się za pośrednictwem bibliotek lub interfejsów API, programiści nie spędzają tyle czasu na nauce

specyfiki języka; lepiej jest odkryć, który stos kodu może wykonać pracę bez konieczności samodzielnego pisania kodu.

- Najważniejsze jest prawidłowe ustawienie algorytmu.
- Narzędzia stały się tak sprytne, że często korygują drobne błędy w kodowaniu i poprawnie interpretują niejasności w kodzie programisty, więc nacisk kładzie się na sprowadzenie pomysłów, a nie na pisanie doskonałego kodu.
- Języki wizualne, w których przeciągasz i upuszczasz obiekty w środowisku graficznym, stają się coraz bardziej powszechne. W pewnym momencie kod może faktycznie zniknąć (przynajmniej dla większości twórców aplikacji).
- Nie wystarczy znać jedną platformę. Większość dzisiejszych aplikacji musi działać bezbłędnie w systemach Windows, Linux, OS X, Android, większości smartfonów i niezliczonych innych platformach, ponieważ użytkownicy chcą oprogramowania w formie, którą rozumieją.

UWZGLĘDNIAJĄC WADY FRAMWOKA

W zależności od tego, z kim rozmawiasz, rozwiązanie ramowe nie zawsze jest panaceum, za jakie uznają go zwolennicy. Jednym z większych problemów podczas korzystania z frameworka jest to, że staje się on własną aplikacją. Zespół programistów musi nauczyć się zarówno frameworka, jak i wszystkich narzędzi używanych do napisania aplikacji. W związku z tym, jeśli większość członków zespołu zajmujących się programowaniem nie korzystała wcześniej z frameworka, będą potrzebować dodatkowego czasu, aby przezwyciężyć krzywą uczenia się frameworka. Jednak po tym, jak nauczą się korzystać z frameworka, z łatwością odzyskają część tej początkowej inwestycji w czasie dzięki wyższej ogólnej wydajności.

Kolejnym problemem związanym z frameworkami jest ich tendencja do nieefektywnego wykorzystywania zasobów. Rozmiar aplikacji frameworka, w tym framework, jest zazwyczaj większy niż aplikacji opracowanej przy użyciu bibliotek. Oczywiście aplikacje monolityczne są generalnie najbardziej wydajne, ponieważ mogą korzystać tylko z zasobów wymaganych dla tej aplikacji. Cały nadmiar kodu znaleziony we frameworkach pochodzi z próby stworzenia uniwersalnego rozwiązania. Wszystkie struktury omówione w tej książce to oferty publiczne. W rzeczywistości większość z nich to również open source. Jednak niektórzy zwolennicy frameworków uważają, że każde przedsiębiorstwo powinno mieć własny framework, który jest opracowywany przy użyciu wspólnego kodu z aplikacji w tym przedsiębiorstwie. Dzięki takiemu podejściu uzyskany framework ma spójny wygląd i działanie, które odpowiada aplikacjom sprzed frameworku, które przedsiębiorstwo musi utrzymywać. Jednak opracowanie niestandardowego frameworka dla konkretnego przedsiębiorstwa jest czasochłonne. Dlatego wiele osób wskazuje, że rozwiązanie oparte na frameworku nie jest tak przydatne ani łatwe do nauczenia, jak rozwiązania nieframeworkowe.

Definiowanie ram głębokiego uczenia się

Myśląc o frameworku do głębokiego uczenia się, tak naprawdę zastanawiasz się, w jaki sposób platforma zarządza zamrożonymi punktami i gorącymi punktami. W większości przypadków platforma uczenia głębokiego zapewnia zamrożone i gorące punkty w tych obszarach:

- Dostęp do sprzętu (np. łatwe korzystanie z GPU)
- Standardowy dostęp do warstwy sieci neuronowej
- Prymitywny dostęp do głębokiego uczenia się
- Zarządzanie wykresami obliczeniowymi
- Szkolenie modelarskie
- Wdrażanie modelu

- Testowanie modeli
- Budowanie i prezentacja wykresów
- Inferencja (propagacja w przód)
- Automatyczne różnicowanie (propagacja wsteczna)

Ramy dotyczą innych kwestii, a skupienie się na konkretnych kwestiach determinuje wykonalność określonych ram dla określonego celu. Podobnie jak w przypadku wielu form pomocy w tworzeniu oprogramowania, musisz starannie wybrać platformę, z której korzystasz.

Wybór konkretnego frameworka

Poprzednie sekcje omawiają ogólnie atrakcyjność frameworków i śledzą, w jaki sposób frameworki mogą tworzyć znacznie lepsze środowisko pracy dla programistów. Omówiono również funkcje, które sprawiają, że platforma głębokiego uczenia się jest wyjątkowa. Oczywiście ilość automatyzacji, jaką zapewnia framework i liczba obsługiwanych przez niego typowych funkcji, jest punktem wyjścia do znalezienia frameworka, który spełni Twoje potrzeby. Musisz również wziąć pod uwagę takie kwestie, jak krzywa uczenia się, jeśli chodzi o łatwość korzystania z frameworka. Jednym z ważniejszych rozważań przy wyborze frameworka jest pamiętanie, że frameworki są specyficzne dla domeny, co oznacza, że jeśli potrzebujesz stworzyć aplikację obejmującą domeny, na przykład aplikację do uczenia głębokiego, która zawiera interfejs sieciowy, potrzebujesz wielu frameworków. Uzyskanie struktur, które dobrze ze sobą współpracują, może mieć kluczowe znaczenie. Jeśli hostujesz również swoją aplikację w chmurze, musisz zastanowić się, które frameworki współpracują również z ofertą dostawcy chmury. Na przykład, jeśli zdecydujesz się użyć TensorFlow jako swojej platformy, możesz również polegać na Amazon Web Services (AWS) do hostowania swojej aplikacji. Inną opcją podczas korzystania z TensorFlow jest przejście bezpośrednio do Google Cloud, gdzie możesz trenować swoje rozwiązanie do uczenia głębokiego za pomocą procesorów GPU lub jednostek przetwarzania Tensor (TPU). TPU zostały opracowane przez Google specjalnie do uczenia maszynowego sieci neuronowych przy użyciu TensorFlow. TPU to układy scalone specyficzne dla aplikacji (ASIC) zoptymalizowane pod kątem konkretnego zastosowania. W tym przypadku służą one do przetwarzania sieci neuronowej za pomocą TensorFlow. Rozmiar i złożoność aplikacji również odgrywają rolę w wyborze platformy uczenia głębokiego, ponieważ często potrzebujesz platformy wyższej klasy do prawidłowej interakcji z dużymi aplikacjami. Konieczność radzenia sobie z różnego rodzaju aplikacjami jest równoważona przez zwykłe obawy dotyczące kosztów i dostępności. Wypróbowanie wielu low-endowych struktur głębokiego uczenia się w tym rozdziale nie będzie Cię kosztować i może zapewnić wszystko, co jest potrzebne do rozpoczęcia pracy.

Praca z low-end frameworkami

Low-endowe struktury uczenia głębokiego często zawierają wbudowany kompromis. Musisz wybrać między kosztami i złożonością użytkowania, a także potrzebą obsługi dużych aplikacji w wymagających środowiskach. Kompromisy, które chcesz ponieść, generalnie odzwierciedlają to, czego możesz użyć do ukończenia swojego projektu. Mając to na uwadze, w poniższych sekcjach omówiono szereg niskobudżetowych frameworków, które są niezwykle przydatne i dobrze sprawdzają się w małych i średnich projektach, ale wymagają również rozważenia kompromisów.

Caffe2

Caffe2 jest luźno oparty na Caffe, który został pierwotnie opracowany na Uniwersytecie Kalifornijskim w Berkeley. Jest napisany w C++ z interfejsem Pythona. Jednym z powodów, dla których ludzie naprawdę lubią Caffe2 jest to, że można trenować i wdrażać model bez pisania kodu. Zamiast tego wybierasz jeden z gotowych modeli i dodajesz go do pliku konfiguracyjnego (co wygląda niesamowicie

jak kod JSON). W rzeczywistości duży wybór wstępnie wytrenowanych modeli pojawia się w ramach Model Zoo, na których można polegać na wielu potrzebach. Pierwotny Caffe miał szereg problemów, które czynią go mniej atrakcyjnym niż Caffe2 dla naukowców zajmujących się danymi. Obecna wersja Caffe jest nadal popularna, ale tak naprawdę nie można jej używać do niczego skomplikowanego. Caffe2 ulepsza Caffe w następujący sposób:

- Lepsze wsparcie dla rozproszonych szkoleń na dużą skalę
- Rozwój mobilny
- Dodano obsługę procesora i obsługę GPU poprzez CUDA

MIGRACJA Z CAFFE DO CAFFE2

Mimo że Caffe wciąż istnieje i wiele osób z niej korzysta, może się okazać, że Caffe2 jest produktem, którego naprawdę potrzebujesz. Jeśli masz teraz jakieś aplikacje Caffe, możesz przenieść je do Caffe2 za pomocą technik dostępnych na https://caffe2.ai/docs/caffe-migration.html, więc każda inwestycja, którą poczyniłeś w Caffe, będzie nadal przydatna w Caffe2.

Inne dodatki znajdziesz w nowej wersji Caffe. Innym powodem popularności Caffe2 jest to, że może przetwarzać obrazy dość szybko i bez znaczących problemów ze skalowaniem. Został zaprojektowany tak, aby był lekki i szybki. Zwróć uwagę, że Caffe2 i PyTorch mają w przyszłości zjednoczyć się jako jeden produkt.

Chainer

Chainer to biblioteka napisana wyłącznie w Pythonie, która opiera się na bibliotekach NumPy i CuPy . Preferred Networks prowadzi rozwój tej biblioteki, ale IBM, Intel, Microsoft i NVIDIA również odgrywają pewną rolę. Głównym punktem tej biblioteki jest to, że pomaga wykorzystać możliwości CUDA twojego GPU, dodając tylko kilka linijek kodu. Innymi słowy, ta biblioteka zapewnia prosty sposób na znaczne zwiększenie szybkości kodu podczas pracy z ogromnymi zestawami danych. Wiele dzisiejszych bibliotek głębokiego uczenia, takich jak Theano i TensorFlow, korzysta ze statycznego podejścia do głębokiego uczenia o nazwie definiowanie i uruchamianie, w którym definiujesz operacje matematyczne, a następnie przeprowadzasz szkolenie w oparciu o te operacje. W przeciwieństwie do Theano i TensorFlow, Chainer wykorzystuje podejście definiowania po uruchomieniu, które opiera się na dynamicznym podejściu do uczenia głębokiego, w którym kod definiuje operacje matematyczne w trakcie szkolenia. Oto dwie główne zalety tego podejścia:

- Intuicyjne i elastyczne podejście: podejście definiowania po uruchomieniu może opierać się na natywnych możliwościach języka, a nie wymagać tworzenia specjalnych operacji w celu przeprowadzenia analizy.
- Debugowanie: Ponieważ podejście definiowania po uruchomieniu definiuje operacje podczas uczenia, można polegać na funkcjach wewnętrznego debugowania w celu zlokalizowania źródła błędów w zestawie danych lub kodzie aplikacji.

TensorFlow 2.0 może również korzystać z definiowania po uruchomieniu, opierając się na Chainer, aby zapewnić szybkie wykonanie.

PyTorch

PyTorch jest następcą Torcha (http://torch.ch/) napisanego w języku Lua Jedna z podstawowych bibliotek Torch (biblioteka PyTorch autograd) została uruchomiona jako rozwidlenie Chainer, co opisano w poprzedniej sekcji. Facebook początkowo opracował PyTorch, ale obecnie używa go wiele

innych organizacji, w tym Twitter, Salesforce i University Oxsford. Oto cechy, które sprawiają, że PyTorch jest wyjątkowy:

- Niezwykle przyjazny dla użytkownika
- Wydajne wykorzystanie pamięci
- Stosunkowo szybko
- Powszechnie używane do badań

Niektórzy ludzie lubią PyTorch, ponieważ jest łatwy do odczytania jak Keras, ale naukowiec nie traci umiejętności korzystania ze skomplikowanych sieci neuronowych. Ponadto PyTorch bezpośrednio obsługuje dynamiczne wykresy modeli obliczeniowych (więcej szczegółów na ten temat można znaleźć w sekcji „Pojęcie, dlaczego TensorFlow jest tak dobry" w dalszej części rozdziału), co czyni go bardziej elastycznym niż TensorFlow bez dodawania TensorFlow Fold.

MXNet

Największym powodem korzystania z MXNet jest szybkość. Może być trudno ustalić, czy MXNet czy CNTK jest szybszy, ale oba produkty są dość szybko i są często używane jako kontrast do spowolnienia, którego niektórzy ludzie doświadczają podczas pracy z TensorFlow. MXNet to produkt Apache, który obsługuje wiele języków, w tym Python, Julia, C++, R i JavaScript. Korzysta z niego wiele dużych organizacji, w tym Microsoft, Intel i Amazon Web Services. Oto aspekty, które sprawiają, że MXNet jest wyjątkowy:

- Funkcje zaawansowanej obsługi GPU
- Może być uruchomiony na dowolnym urządzeniu
- Zapewnia imperatywne API o wysokiej wydajności
- Oferuje łatwe serwowanie modeli
- Zapewnia wysoką skalowalność

Może to brzmieć jak idealny produkt dla Twoich potrzeb, ale MXNet ma co najmniej jedną poważną wadę - brakuje mu poziomu wsparcia społeczności, które zapewnia TensorFlow. Ponadto większość badaczy nie patrzy na MXNet przychylnie, ponieważ może stać się on złożony, a badacz w większości przypadków nie ma do czynienia ze stabilnym modelem.

Microsoft Cognitive Toolkit/CNTK

Jak wspomniano w poprzedniej sekcji, jego szybkość jest jednym z powodów korzystania z Microsoft Cognitive Toolkit (CNTK). Microsoft używa CNTK do dużych zbiorów danych — naprawdę dużych. Jako produkt obsługuje języki programowania Python, C++, C# i Java. W związku z tym, jeśli jesteś naukowcem, który polega na R, to nie jest to produkt dla Ciebie. Microsoft używał tego produktu w Skype, Xbox i Cortana. Specjalne cechy tego produktu to

- Wspaniały występ
- Wysoka skalowalność
- Wysoce zoptymalizowane komponenty
- Obsługa Apache Spark
- Obsługa Azure Cloud

Podobnie jak w przypadku MXNet, CNTK ma wyraźny problem z brakiem odpowiedniego wsparcia społeczności. Ponadto zwykle nie zapewnia zbyt wiele wsparcia stron trzecich, więc jeśli pakiet nie zawiera potrzebnych funkcji, możesz ich w ogóle nie uzyskać.

Zrozumienie TensorFlow

W tej chwili TensorFlow znajduje się na szczycie stosu pod względem frameworków głębokiego uczenia . Sukces TensorFlow wynika z wielu powodów, ale przede wszystkim wynika z zapewnienia solidnego środowiska w stosunkowo łatwym w użyciu pakiecie. Poniższe sekcje pomogą Ci zrozumieć, dlaczego ta książka wykorzystuje TensorFlow. Odkrywasz, co sprawia, że TensorFlow jest tak ekscytujący i jak dodatki sprawiają, że korzystanie z niego jest jeszcze łatwiejsze.

Zrozumieć, dlaczego TensorFlow jest tak dobry

Produkt musi oferować całkiem sporo pod względem funkcjonalności, łatwości użytkowania i niezawodności, aby wpłynąć na rynek, gdy ludzie mają wiele możliwości wyboru. Jednym z powodów sukcesu TensorFlow jest to, że obsługuje on kilka najpopularniejszych języków: Python, Java, Go i JavaScript. Ponadto jest dość rozszerzalny. Każde rozszerzenie jest operacją (tak jak w działaniu). Chodzi o to, że gdy produkt ma świetne wsparcie dla wielu języków i pozwala na znaczną rozszerzalność, produkt staje się popularny, ponieważ ludzie mogą wykonywać zadania w sposób, który najlepiej im odpowiada, a nie tak, jak zdaniem dostawcy potrzebuje użytkownik. Ważny jest również sposób, w jaki TensorFlow ocenia i wykonuje kod. Natywnie TensorFlow obsługuje tylko statyczne wykresy obliczeniowe. Jednak rozszerzenie TensorFlow Fold obsługuje również dynamiczne wykresy. Wykres dynamiczny to taki, w którym struktura wykresu obliczeniowego zmienia się w zależności od struktury danych wejściowych i zmienia się dynamicznie w miarę działania aplikacji. Korzystając z dynamicznego grupowania, TensorFlow Fold może stworzyć statyczny wykres z dynamicznych wykresów, który następnie może przesłać do TensorFlow. Ten statyczny wykres przedstawia transformację jednego lub więcej dynamicznych wykresów modelujących niepewne dane. Oczywiście możesz nawet nie potrzebować budować grafu obliczeniowego, ponieważ TensorFlow obsługuje również szybkie wykonywanie (natychmiastowe ocenianie operacji bez tworzenia grafu obliczeniowego), dzięki czemu może natychmiast ocenić kod Pythona (nazywane wykonywaniem dynamicznym). Włączenie tej dynamicznej funkcjonalności sprawia, że TensorFlow jest niezwykle elastyczny w zakresie danych, które może pomieścić.

WSPARCIE TENSORFLOW NA COLAB

Wielu programistów polega dziś na środowiskach online, takich jak Colab, przy wykonywaniu zadań, ponieważ instalowanie i konfigurowanie TensorFlow na komputerze stacjonarnym może okazać się trudne i musisz mieć procesor graficzny, który obsługuje TensorFlow, jeśli chcesz przyspieszyć przetwarzanie. Ponadto musisz wziąć pod uwagę wiele innych kwestii. Colab wydaje się ułatwiać sprawy. Aby uzyskać wsparcie dla procesora, wystarczy wybrać pole konfiguracji. Aby upewnić się, że masz odpowiednie wsparcie, po prostu uruchom trochę dodatkowego kodu specyficznego dla Colab https://colab.research.google.com/notebooks/gpu.ipynb). Jednak rzeczywistość rzadko działa tak samo jak teoria. Po pierwsze, musisz ponownie zainstalować wszystko za każdym razem, gdy rozpoczynasz nową sesję Colab, ponieważ obsługa bibliotek nie jest stała . Oczywiście możesz w ogóle nie mieć dostępu do GPU (według uznania Google) lub obsługa GPU może mieć ograniczenia. Aby zapewnić najlepszą możliwą naukę, w tej książce zastosowano niezwykle uproszczoną konfigurację TensorFlow, która pozwala uniknąć wielu pułapek, które napotykają inne środowiska. To środowisko będzie działać w przypadku książki, dowolnego doświadczenia edukacyjnego, które możesz mieć w szkole, małych projektów eksperymentalnych, a nawet projektów dla małych i średnich firm, które korzystają z małych i średnich zestawów danych. Nigdy nie możesz użyć tej konfiguracji do uruchomienia projektu typu Facebook. Oprócz różnych rodzajów obsługi dynamicznej, TensorFlow umożliwia również użycie GPU do przyspieszenia obliczeń. W rzeczywistości można korzystać z wielu procesorów graficznych i rozłożyć model obliczeniowy na kilka maszyn w klastrze. Możliwość

dostarczenia tak dużej mocy obliczeniowej do rozwiązania problemu sprawia, że TensorFlow jest szybszy niż większość konkurencji. Szybkość jest ważna, ponieważ odpowiedzi na pytania często mają krótką długość życia; uzyskanie odpowiedzi jutro na pytanie, które masz dzisiaj, nie zadziała w wielu scenariuszach. Na przykład lekarz, który polega na usługach sztucznej inteligencji, aby zapewnić alternatywy podczas operacji, potrzebuje natychmiastowych odpowiedzi lub pacjent może umrzeć. Funkcje obliczeniowe pomagają jedynie uzyskać rozwiązanie problemu. TensorFlow pomaga również w wizualizacji rozwiązania na różne sposoby za pomocą rozszerzenia TensorBoard. To rozszerzenie pomaga w

- Wizualizacji wykresu obliczeniowego
- Wskaźnikach wykonania wykresu
- Pokazać dodatkowe dane w razie potrzeby

Podobnie jak w przypadku wielu produktów, które zawierają wiele funkcji, TensorFlow oferuje stromą krzywą uczenia się. Jednak cieszy się również znacznym wsparciem społeczności, zapewnia dostęp do wielu praktycznych samouczków, ma świetne wsparcie innych firm dla kursów online i oferuje wiele innych pomocy, aby zmniejszyć krzywą uczenia się.

Ułatwianie TensorFlow przy użyciu TFLearn

Jedną z głównych skarg dotyczących bezpośredniego korzystania z TensorFlow jest to, że kodowanie jest zarówno niskie, jak i czasami trudne. Kompromis, jaki robisz z TensorFlow, polega na tym, że zyskujesz dodatkową elastyczność i kontrolę, pisząc więcej kodu. Jednak nie każdy potrzebuje głębi, którą TensorFlow może zapewnić, dlatego pakiety takie jak TFLearn (http://tflearn.org/), czyli TensorFlow Learn, są tak ważne. (Na rynku można znaleźć wiele pakietów, które próbują zmniejszyć złożoność; TFLearn jest tylko jednym z nich.) TFLearn ułatwia pracę z TensorFlow, ale w określony sposób:

- Wysoki poziom interfejsu programowania aplikacji (API) pomaga uzyskać wyniki przy mniejszej ilości kodu.
- Wysoki poziom API zmniejsza ilość standardowego (boilerplate) kodu, który piszesz.
- Prototypowanie jest szybsze, podobnie jak w przypadku Caffe2 (opisanej wcześniej w tym rozdziale).
- Przejrzystość z TensorFlow oznacza, że możesz zobaczyć, jak działają funkcje i korzystać z nich bezpośrednio, bez polegania na TFLearn.
- Użycie funkcji pomocniczych automatyzuje wiele zadań, które normalnie trzeba wykonać ręcznie.
- Zastosowanie doskonałej wizualizacji pozwala z większą łatwością zobaczyć różne aspekty aplikacji, w tym model obliczeniowy.

Otrzymujesz całą tę funkcjonalność i wiele więcej, nie rezygnując z aspektów, które sprawiają, że TensorFlow jest tak wspaniałym produktem. Na przykład nadal masz pełny dostęp do możliwości TensorFlow w zakresie korzystania z procesorów, układów GPU, a nawet wielu systemów, aby zapewnić większą moc obliczeniową w przypadku każdego problemu.

Używanie Keras jako najlepszego uproszczenia

Keras jest mniej frameworkiem, a bardziej API (zestawem specyfikacji interfejsów, których można używać z wieloma platformami jako backendami). Zazwyczaj jest to jednak platforma do głębokiego uczenia się, ponieważ tak właśnie z niej korzystają. Aby korzystać z Keras, musisz mieć również framework do uczenia głębokiego, taki jak TensorFlow, Theano, MXNet lub CNTK. Keras jest w

rzeczywistości dołączany do TensorFlow, co również sprawia, że jest to łatwe rozwiązanie do zmniejszenia złożoności TensorFlow. W tej książce założono, że używasz Keras z TensorFlow, ale wiedza, że możesz używać Keras z innymi frameworkami do uczenia głębokiego, jest zaletą. Dlatego ta książka nie korzysta z wersji Keras zawartej w TensorFlow, ale instaluje ją osobno. Możesz używać tego samego interfejsu z wieloma platformami, umożliwiając korzystanie z potrzebnego frameworka bez konieczności zajmowania się kolejną krzywą uczenia się. Największym atutem Keras jest to, że proces tworzenia aplikacji przy użyciu struktury głębokiego uczenia się umieszcza w paradygmacie, który większość ludzi dobrze rozumie. Nie da się stworzyć aplikacji, która byłaby zarówno łatwa w użyciu, jak i radziła sobie z naprawdę złożonymi sytuacjami — a jednocześnie byłaby elastyczna. Więc Keras niekoniecznie dobrze radzi sobie ze wszystkimi sytuacjami. Na przykład jest to dobry produkt, gdy Twoje potrzeby są proste, ale nie jest to dobry wybór, jeśli planujesz opracować nowy rodzaj sieci neuronowej. Siła Keras polega na tym, że umożliwia szybkie prototypowanie przy niewielkim wysiłku. API nie przeszkadza, gdy stara się zapewnić elastyczność, której możesz nie potrzebować w bieżącym projekcie. Ponadto, ponieważ Keras upraszcza sposób wykonywania zadań, nie można go rozszerzyć tak, jak w przypadku innych produktów, co ogranicza możliwość dodawania funkcjonalności do istniejącego środowiska.

Więcej niż kilka osób skarżyło się na czasami niejednoznaczne zgłaszanie błędów dostarczane przez Keras. Jednak Keras częściowo kompensuje ten problem, zapewniając silne wsparcie społeczności. Ponadto wiele osób narzekających na komunikaty o błędach najwyraźniej próbuje zrobić coś złożonego. Pamiętanie o szybkim prototypowaniu Keras może powstrzymać Cię od próbowania projektów, które mogą być zbyt trudne dla produktu.

Pobieranie kopii TensorFlow i Keras

Twoja kopia Pythona dostarczana z Anacondą nie zawiera kopii TensorFlow ani Keras; produkty te należy zainstalować osobno. Aby uniknąć problemów z integracją TensorFlow z narzędziami Anaconda, nie postępuj zgodnie z instrukcjami znajdującymi się na https://www.tensorflow.org/install/pip dotyczącymi instalacji produktu za pomocą pip. Podobnie, unikaj korzystania z instrukcji instalacji Keras na https://keras.io/#installation. Aby upewnić się, że Twoja kopia TensorFlow i Keras jest dostępna w programie Notebook, musisz otworzyć monit Anacondy, a nie standardowy wiersz poleceń lub okno terminala. W przeciwnym razie nie możesz upewnić się, że masz skonfigurowane odpowiednie ścieżki. Poniższe kroki pozwolą Ci rozpocząć instalację.

1. W wierszu polecenia Anaconda wpisz python - wersja i naciśnij klawisz Enter.

Zobaczysz aktualnie zainstalowaną wersję Pythona, która powinna być wersją 3.6.5. Ścieżka, którą widzisz w oknie, jest funkcją Twojego systemu operacyjnego, czyli w tym przypadku Windows, ale możesz zobaczyć inną ścieżkę podczas korzystania z monitu Anaconda. Następnym krokiem jest stworzenie środowiska do wykonywania kodu, który opiera się na TensorFlow i Keras. Zaletą korzystania ze środowiska jest zachowanie nieskazitelnego środowiska do późniejszego wykorzystania z innymi bibliotekami. Używasz conda, a nie innego produktu środowiskowego, takiego jak virtualenv, aby upewnić się, że oprogramowanie jest zintegrowane z narzędziami Anaconda.

2. Wpisz conda create -n DL4Denv python=3 anaconda=5.3.0 tensorflow=1.11.0 keras=2.2.4 nb_conda i naciśnij Enter. Możesz zobaczyć komunikat ostrzegawczy o dostępności nowszej wersji conda. Można bezpiecznie zignorować tę wiadomość (lub możesz zaktualizować conda za pomocą polecenia wyświetlonego później w ostrzeżeniu, jeśli chcesz). W razie potrzeby wpisz Y i naciśnij klawisz Enter, aby usunąć komunikat i kontynuować proces tworzenia. Wykonanie tego kroku może zająć trochę czasu, ponieważ Twój system będzie musiał pobrać TensorFlow 1.11.0 i Keras 2.2.4 ze źródła online. Kiedy pobieranie zostało zakończone, konfiguracja musi utworzyć dla Ciebie kompletną instalację. Po

wykonaniu wszystkich wymaganych kroków pojawi się monit Anaconda. W międzyczasie przeczytanie dobrego artykułu technicznego lub wypicie kawy pomoże zabić czas.

3. Wpisz conda, aktywuj DL4Denv i naciśnij Enter. Monit zmienia się i pokazuje środowisko DL4Denv, a nie środowisko podstawowe lub root. Wszelkie zadania, które teraz wykonujesz, będą miały wpływ na środowisko DL4D, a nie na oryginalne środowisko podstawowe.

4. Wpisz python -m pip install - uaktualnij pip i naciśnij Enter. Ten krok zajmie trochę czasu, ale nie tak długo, jak stworzenie środowiska. Celem tego kroku jest upewnienie się, że masz zainstalowaną najnowszą wersję pip, aby późniejsze.

5. Wpisz conda deactivate i naciśnij Enter. Dezaktywacja środowiska powoduje powrót do środowiska podstawowego. Wykonujesz ten krok, aby mieć pewność, że zawsze kończysz sesję w środowisku podstawowym.

6. Zamknij monit Anakondy. Twoje instalacje TensorFlow i Keras są teraz gotowe do użycia.

Naprawianie błędu narzędzi do budowania C++ w systemie Windows

Wiele funkcji Pythona wymaga do kompilacji narzędzi do budowania w języku C++, ponieważ programiści napisali kod w C++, a nie w Pythonie, aby uzyskać największą szybkość wykonywania niektórych rodzajów przetwarzania. Na szczęście zarówno Linux, jak i OS X są dostarczane z zainstalowanymi narzędziami do budowania C++. Tak więc nie musisz robić nic specjalnego, aby stworzyć polecenia kompilacji Pythona które działają. Użytkownicy systemu Windows muszą jednak zainstalować kopię narzędzia do kompilacji C++ 14 lub nowszego, jeśli jeszcze ich nie zainstalowali. W rzeczywistości środowisko Notebook jest dość wybredne - potrzebujesz Visual C++ 14 lub nowszego, a nie jakiejkolwiek wersji C++ (takiej jak GCC). Jeśli niedawno zainstalowałeś program Visual Studio lub inny produkt deweloperski firmy Microsoft, możesz mieć zainstalowane narzędzia do kompilacji i nie musisz instalować drugiej kopii. Uzyskanie samych narzędzi do budowania nic Cię nie kosztuje. Poniższe kroki pokazują krótką i łatwą metodę uzyskania wymaganych narzędzi do kompilacji, jeśli nie masz jeszcze zainstalowanego C++ 14 lub nowszego:

1. Pobierz instalator narzędzi do kompilacji offline . Twoja aplikacja pobiera kopię pliku vs_buildtools.exe. Próba korzystania z narzędzi do kompilacji online często wiąże się ze zbyt wieloma opcjami, a Microsoft oczywiście chce, abyś kupił jego produkt.

2. Zlokalizuj pobrany plik na dysku twardym i kliknij dwukrotnie vs_buildtools.exe. Zobaczysz okno dialogowe Instalator programu Visual Studio. Zanim będziesz mógł zainstalować narzędzia do budowania, musisz powiedzieć instalatorowi, co chcesz zainstalować.

3. Kliknij Kontynuuj. Instalator programu Visual Studio pobiera i instaluje dodatkowe pliki obsługi. Po zakończeniu tej instalacji zapyta, które obciążenie należy zainstalować.

4. Zaznacz opcję Visual C++ Build Tools, a następnie kliknij Install. Nie musisz instalować niczego poza domyślnymi funkcjami. Okienko Szczegóły instalacji po prawej stronie okna Instalatora programu Visual Studio zawiera mylący zestaw opcji, które nie będą potrzebne w tej książce. Proces pobierania około 1,1 GB rozpoczyna się natychmiast. Czekając, możesz napić się kawy. W oknie Instalator programu Visual Studio wyświetlany jest postęp pobierania i instalacji. W pewnym momencie zobaczysz komunikat informujący, że instalacja się powiodła.

5. Zamknij okno Instalator programu Visual Studio. Twoja kopia narzędzi kompilacji Visual C++ jest gotowa do użycia. Po wykonaniu instalacji może być konieczne ponowne uruchomienie systemu, zwłaszcza jeśli wcześniej zainstalowano program Visual Studio.

Dostęp do nowego środowiska w Notatniku

Kiedy otwierasz Notatnik, automatycznie wybiera on środowisko podstawowe lub root - domyślne środowisko dla narzędzi Anaconda.

Przeglądanie matematyki i optymalizacji macierzy

Część 1 mówi o podstawach głębokiego uczenia się i dlaczego jest to ważne dzisiaj. W Części 2 zagłębiłeś się w proces uczenia się czegoś z danych poprzez uczenie maszynowe. Kluczowym punktem z obu tych Części jest to, że twój komputer niczego nie rozumie, ale możesz dostarczyć mu dane, a to z kolei może pomóc ci zrozumieć coś nowego na podstawie tych danych. Na przykład możesz opisać do niej operację matematyczną, która pomoże Ci uzyskać wgląd lub zrozumieć dane w sposób, którego nie mogłeś inaczej. Komputer staje się narzędziem do wykonywania naprawdę zaawansowanych obliczeń matematycznych znacznie szybciej, niż można to zrobić ręcznie. Podstawą tych operacji matematycznych jest wykorzystanie określonych struktur danych, w tym macierzy. Musisz zrozumieć operacje skalarne, wektorowe i macierzowe w ramach odkrywania, w jaki sposób uczenie głębokie może znacząco zmienić sposób wyświetlania danych opisujących dzisiejszy świat. Łączenie danych znalezionych w określonych rodzajach struktur z algorytmami zaprojektowanymi do pracy z tymi strukturami jest podstawowym elementem uczenia głębokiego. Ten rozdział pomaga zrozumieć dane, struktury danych używane do ich przechowywania oraz sposób, w jaki można wykonywać proste zadania z tymi strukturami. Do tej pory tak naprawdę nie widziałeś niczego, co wygląda na jakąkolwiek naukę. Samo posiadanie struktur danych i odpowiednich operacji do interakcji z nimi nie wystarczy do rozważenia procesu uczenia się. Ostatnia część tego rozdziału pomoże ci powiązać wykonywanie tych operacji z ich szybkim wykonaniem przy użyciu optymalizacji. Czynność optymalizacji operacji wykonywanych na danych jest tym, co stanowi uczenie: komputer uczy się unikać niepotrzebnych opóźnień w wykonaniu analizy potrzebnej do wykonania zadań.

Ujawnianie matematyki, której naprawdę potrzebujesz

Świat jest niesamowicie złożonym miejscem, a próba przedstawienia go za pomocą danych i matematyki bardzo jasno pokazuje ten fakt. Dane wyrażają świat rzeczywisty jako abstrakcję przy użyciu wartości liczbowych lub innych jako środków do ilościowego określenia abstrakcji. Na przykład kolor niebieski może stać się wartością 1. Matematyka to sposób manipulacji tą wartością, aby lepiej je zrozumieć i rozpoznać wzorce, które w innym przypadku mogłyby być niejasne. Na przykład może się okazać, że większa część osób mieszkających na określonym obszarze woli kolor niebieski od innych. Poniższe sekcje pomagają zrozumieć dane i matematykę z perspektywy sztucznej inteligencji, która pozwala na interakcję ze światem w sposób zautomatyzowany (na przykład poprzez czyszczenie dywanu robotem lub prosząc system nawigacji samochodu o podanie wskazówek do miejsce, w którym nie byłeś wcześniej).

Praca z danymi

Bez danych niemożliwe jest przedstawienie bytów ze świata rzeczywistego w formie, którą komputer może pomóc w zrozumieniu i zarządzaniu. Komputer nie rozumie danych; po prostu przechowuje dane i umożliwia manipulowanie danymi za pomocą matematyki. Komputer też nie rozumie danych wyjściowych. Wynik manipulacji wymaga interpretacji przez człowieka, aby miał sens. Tak więc dane zaczynają się i kończą na ludzkiej interpretacji świata rzeczywistego przedstawionej jako abstrakcja. Tworząc dane, musisz podać pewną spójną miarę abstrakcji, w przeciwnym razie komunikacja stanie się niemożliwa. Na przykład, jeśli jeden zestaw danych przedstawia kolor niebieski jako liczbę całkowitą 1, inny zestaw danych przedstawia kolor niebieski jako liczbę rzeczywistą 2,0, a trzeci przedstawia kolor niebieski jako niebieski łańcuch, nie można połączyć informacji, chyba że utworzysz inny zbiór danych zawierający te same wartości dla każdego niebieskiego wpisu. Ponieważ ludzie są niespójni, dane również mogą być niespójne (zakładając, że są one przede wszystkim poprawne). Przekształcenie wartości między zestawami danych nie zmienia faktu, że ludzie, którzy je interpretują, nadal widzą kolor niebieski zakodowany w abstrakcji, którą są dane. Po zebraniu wystarczającej ilości danych

możesz nimi manipulować w sposób, który pozwoli komputerowi zaprezentować Ci wzorce, których być może wcześniej nie widziałeś. Jak zawsze, komputer nie rozumie danych ani ich interpretacji, ani nawet tego, że stworzył dla ciebie wzór. Matematyka zdefiniowana przez niezwykle inteligentnych naukowców manipuluje danymi we wzór za pomocą wyrażeń matematycznych. Z perspektywy uczenia głębokiego masz więc ludzkiego interpretera dostarczającego abstrakcje danych z rzeczywistych obiektów, komputer wykonujący jedną lub więcej manipulacji tymi danymi oraz dane wyjściowe, które ponownie wymagają ludzkiej interpretacji, aby w ogóle miała jakiekolwiek znaczenie. Głębokie uczenie, dla celów tego rozdziału, jest po prostu czynnością automatyzacji procesu manipulacji danymi przy użyciu tych samych technik, których może używać człowiek, w połączeniu z szybkością, jaką może zapewnić komputer. Akt uczenia się oznacza odkrycie, jak skutecznie wykonywać manipulacje, tak aby przydatne wzorce pojawiły się jako część wyników. Automatyzacja nie jest przydatna, jeśli nie jest kontrolowana, a głębokie uczenie zapewnia tę kontrolę za pomocą obliczeń macierzowych. Obliczenie macierzowe to seria mnożenia i sumowania uporządkowanych zbiorów liczb. Musisz zrozumieć, jak głębokie uczenie działa matematycznie, abyś mógł:

- Rozwiązać wszelkie fantazje, że głębokie uczenie działa w taki sam sposób jak ludzki mózg
- Zdefiniować narzędzia potrzebne później do stworzenia przykładu głębokiej sieci neuronowej

Tworzenie i operowanie macierzą

Zapewnienie, że wszystkie abstrakcje używane dla konkretnych obiektów w świecie rzeczywistym są takie same, nie wystarcza do stworzenia sensownego modelu. Zwykłe stwierdzenie, że liczba całkowita 1 reprezentuje kolor niebieski, nie zapewnia niezbędnej struktury do wykonania manipulacji matematycznych, chyba że taka manipulacja dotyczy pojedynczej wartości (skalaru). Grupa powiązanych wartości może pojawić się na liście (wektorze), ale tylko wtedy, gdy każda z wartości reprezentuje ten sam rodzaj obiektu. Na przykład możesz utworzyć listę kolorów, z których każdy ma określoną wartość. Aby były naprawdę przydatne, dane muszą pojawiać się w formie grupującej się jak wpisy w formularzu, który usprawnia automatyczne przetwarzanie. Ogólnie preferowaną formą jest tabela (macierz), która ma określone typy wartości obiektów w kolumnie i poszczególne wpisy w wierszach. Widzisz, że macierze są często używane w tej książce, ponieważ zapewniają wygodny sposób przenoszenia złożonych wpisów jako jednostki. Macierz nieruchomości w Bostonie może zawierać różnego rodzaju powiązane informacje, takie jak cena, liczba pokoi i charakterystyka środowiskowa każdego domu. Mimo że uzyskujesz dane w innej formie, proces importowania, który przekształca je w macierz, jest pierwszym krokiem w użyciu zestawu danych w celu zobaczenia przydatnych wzorców poprzez zastosowanie głębokiego uczenia. Matematyka, której potrzebujesz, sprowadza się do tych rzeczy:

- Proces, w tym matematyka, używa do przekształcania wszystkich elementów danych w podobną formę
- Proces, w tym matematyka, używany do umieszczania elementów danych w strukturze, takiej jak macierz, w celu pomocy w automatycznym przetwarzaniu danych
- Matematyka potrzebna do manipulowania macierzą, aby pojawiły się przydatne wzory
- Metodologia, w tym matematyka, wykorzystuje się do dostarczenia danych wyjściowych do ludzkiej interpretacji wzorców

Zrozumienie operacji skalarnych, wektorowych i macierzowych

Aby wykonać użyteczną pracę z Pythonem, często musisz pracować z większą ilością danych, które są dostarczane w określonych formularzach. Formy te mają dziwnie brzmiące nazwy, ale nazwy są dość ważne. Trzy terminy, które musisz znać, to:

- Skalar: Pojedynczy element danych podstawowych. Na przykład sama liczba 2 jest skalarem.
- Wektor: Jednowymiarowa tablica (zasadniczo lista) elementów danych. Na przykład tablica zawierająca liczby 2, 3, 4 i 5 będzie wektorem. Dostęp do elementów w wektorze uzyskuje się za pomocą indeksu liczonego od zera, czyli wskaźnika do żądanego elementu. Element o indeksie 0 jest pierwszym elementem w wektorze, który w tym przypadku wynosi 2.
- Macierz: tablica dwóch lub więcej wymiarów (zasadniczo tabela) elementów danych. Na przykład macierzą jest tablica zawierająca liczby 2, 3, 4 i 5 w pierwszym wierszu oraz 6, 7, 8 i 9 w drugim wierszu. Dostęp do elementów w macierzy można uzyskać za pomocą indeksu wierszy i kolumn liczonych od zera. Pozycja w wierszu 0, kolumnie 0 jest pierwszą pozycją w macierzy, czyli w tym przypadku 2.

Głębokie uczenie opiera się na macierzach. Używane źródła danych mają format wierszy i kolumn do opisywania atrybutów określonego elementu danych. Na przykład, aby opisać osobę, macierz może zawierać takie atrybuty, jak imię i nazwisko, wiek, adres i numer konkretnego przedmiotu kupowanego każdego roku. Znając te atrybuty, możesz przeprowadzić analizę, która dostarczy nowych rodzajów informacji i pomoże w uogólnieniu na temat określonej populacji. Python sam w sobie zapewnia interesujący zestaw funkcji, ale nadal będziesz musiał wykonać dużo pracy, aby wykonać niektóre zadania. Aby zmniejszyć ilość wykonywanej pracy, możesz polegać na kodzie pisanym przez inne osoby i znajdującym się w bibliotekach. Poniższe sekcje opisują, jak używać biblioteki NumPy do wykonywania różnych zadań na macierzach.

Tworzenie matrycy

Zanim będziesz mógł zrobić cokolwiek z macierzą, musisz ją stworzyć, co obejmuje wypełnienie jej danymi. Najłatwiejszym sposobem wykonania tego zadania jest skorzystanie z biblioteki NumPy, którą importujesz jako np. za pomocą poniższego kodu:

import numpy as np.

Aby utworzyć podstawową macierz, po prostu używasz funkcji tablicowej NumPy, tak jak w przypadku wektora, ale definiujesz dodatkowe wymiary. Wymiar to kierunek w macierzy. Na przykład dwuwymiarowa macierz zawiera wiersze (jeden kierunek) i kolumny (drugi kierunek). Tablica call myMatrix = np.array([[1,2,3], [4,5,6], [7,8,9]]) tworzy macierz zawierającą trzy wiersze i trzy kolumny:

array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

Zwróć uwagę, jak osadziłeś trzy listy na liście kontenerów, aby utworzyć dwa wymiary. Aby uzyskać dostęp do określonego elementu tablicy, należy podać wartość indeksu wiersza i kolumny, taką jak mojaMacierz[0, 0], aby uzyskać dostęp do pierwszej wartości równej 1. W podobny sposób można utworzyć macierze o dowolnej liczbie wymiarów. Na przykład mojaMacierz = np.array([[[1,2], [3,4]], [[5,6], [7,8]]]) tworzy trójwymiarową macierz z x, y i z oś, która wygląda tak:

array([[[1, 2],

[3, 4]],

[[5, 6],

[7, 8]]])

W takim przypadku osadzasz dwie listy, w ramach dwóch list kontenerów, w ramach jednej listy kontenerów, która zawiera wszystko razem. W takim przypadku musisz podać wartość indeksu x, y i z, aby uzyskać dostęp do określonej wartości. Na przykład mojaMacierz[0, 1, 1] uzyskuje dostęp do wartości 4. W niektórych przypadkach musisz utworzyć macierz, która ma określone wartości początkowe. Na przykład, jeśli na początku potrzebujesz macierzy wypełnionej jedynkami, możesz użyć funkcji jedynki. Wywołanie myMatrix = np.ones([4,4], dtype=np.int32) tworzy macierz zawierającą cztery wiersze i cztery kolumny wypełnione wartościami int32 w następujący sposób:

array([[1, 1, 1, 1],

[1, 1, 1, 1],

[1, 1, 1, 1],

[1, 1, 1, 1]])

Podobnie, wywołanie myMatrix = np.ones([4,4,4], dtype=np.bool) utworzy trójwymiarową tablicę. Tym razem macierz będzie zawierała wartości logiczne True. Dostępne są również funkcje do tworzenia macierzy wypełnionej zerami, macierzy tożsamości oraz do zaspokojenia innych potrzeb.

Biblioteka NumPy obsługuje rzeczywistą klasę macierzy. Klasa matrix obsługuje specjalne funkcje, które ułatwiają wykonywanie zadań specyficznych dla macierzy. Odkryjesz te cechy w dalszej części rozdziału. Na razie wszystko, co naprawdę musisz wiedzieć, to jak stworzyć macierz typu danych matrix. Najprostszą metodą jest wykonanie wywołania podobnego do tego, którego używasz dla funkcji tablicy, ale zamiast tego użyj funkcji mat, na przykład mojaMacierz = np.mat([[1,2,3], [4,5,6] , [7,8,9]]), co daje następującą macierz:

matrix([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

Możesz również przekonwertować istniejącą tablicę na macierz za pomocą funkcji asmatrix. Użyj funkcji asarray, aby przekonwertować obiekt macierzy z powrotem do postaci tablicy. Jedyny problem z klasą matrix polega na tym, że działa ona tylko na macierzach dwuwymiarowych. Jeśli spróbujesz przekonwertować trójwymiarową macierz na klasę macierzy, zobaczysz komunikat o błędzie informujący, że kształt jest zbyt duży, aby mógł być macierzą.

Wykonywanie mnożenia macierzy

Dwie popularne metody mnożenia macierzy to element po elemencie i iloczyn skalarny. Podejście element po elemencie jest proste. Poniższy kod tworzy mnożenie dwóch macierzy element po elemencie:

a = np.array([[1,2,3],[4,5,6]])

b = np.array([[1,2,3],[4,5,6]])

print(a*b)

W zamian otrzymujesz tablicę w rodzaju pokazanej tutaj:

[[1 4 9]

[16 25 36]]

Zauważ, że a i b mają ten sam kształt: dwa rzędy i trzy kolumny. Aby wykonać mnożenie element po elemencie, dwie macierze muszą mieć ten sam kształt. W przeciwnym razie zobaczysz komunikat o błędzie z informacją, że kształty są nieprawidłowe. Podobnie jak w przypadku wektorów, funkcja mnożenia również daje wynik element po elemencie. Niestety mnożenie element po elemencie może dawać nieprawidłowe wyniki podczas pracy z algorytmami. W wielu przypadkach to, czego naprawdę potrzebujesz, to iloczyn skalarny, który jest sumą iloczynów dwóch ciągów liczbowych.

Podczas wykonywania iloczynu skalarnego z macierzą liczba kolumn w macierzy a musi odpowiadać liczbie wierszy w macierzy b. Jednak liczba wierszy w macierzy a może być dowolną liczbą, a liczba kolumn w macierzy b może być dowolną liczbą, o ile tworzysz iloczyn skalarny a przez b. Na przykład poniższy kod tworzy poprawny iloczyn skalarny:

```
a = np.array([[1,2,3],[4,5,6]])

b = np.array([[1,2,3],[3,4,5],[5,6,7]])

print(a.dot(b))
```

Oto, co otrzymujesz jako dane wyjściowe w tym przypadku:

```
[[22 28 34]

[49 64 79]]
```

Należy zauważyć, że dane wyjściowe zawierają liczbę wierszy znalezionych w macierzy a oraz liczbę kolumn znalezionych w macierzy b. Jak to wszystko działa? Aby uzyskać wartość znalezioną w tablicy wyjściowej o indeksie [0,0] 22, sumujesz wartości a[0,0]*b[0,0] (czyli 1), a[0,1]* b[1,0] (czyli 6) i a[0,2]*b[2,0] (czyli 15), aby uzyskać wartość 22. Pozostałe wpisy działają dokładnie w ten sam sposób. Zaletą korzystania z klasy macierzy numpy jest to, że niektóre zadania stają się bardziej proste. Na przykład mnożenie działa dokładnie tak, jak tego oczekujesz. Poniższy kod tworzy iloczyn skalarny przy użyciu klasy matrix:

```
a = np.mat([[1,2,3],[4,5,6]])

b = np.mat([[1,2,3],[3,4,5],[5,6,7]])

print(a*b)
```

Dane wyjściowe z operatorem * są takie same, jak przy użyciu funkcji kropki z tablicą. Jednak mimo że wynik wygląda tak samo, jak przy użyciu funkcji kropki, nie jest dokładnie taki sam. Wyjściem poprzedniego kodu jest tablica, podczas gdy wyjściem tego kodu jest macierz. Ten przykład wskazuje również, że podczas wykonywania zadań, takich jak mnożenie dwóch macierzy, musisz wiedzieć, czy używasz tablicy, czy obiektu macierzy. Aby wykonać mnożenie element po elemencie przy użyciu dwóch obiektów macierzy, należy użyć funkcji mnożenia numpy.

Wykonywanie zaawansowanych operacji na macierzach

Przeprowadzimy Cię przez wszystkie rodzaje interesujących operacji na macierzach, ale niektórych z nich używasz często, dlatego pojawiają się tu. Podczas pracy z tablicami czasami otrzymujesz dane w kształcie, który nie działa z algorytmem. Na szczęście numpy ma specjalną funkcję zmiany kształtu, która pozwala umieścić dane w dowolnym wymaganym kształcie. W rzeczywistości możesz go użyć do przekształcenia wektora w macierz, jak pokazano w poniższym kodzie:

```
changeIt = np.array([1,2,3,4,5,6,7,8])
```

```
print(changeIt)

print(changeIt.reshape(2,4))

print(changeIt.reshape(2,2,2))
```

Ten kod generuje następujące dane wyjściowe, które pokazują postęp zmian wytworzonych przez funkcję zmiany kształtu:

```
[1 2 3 4 5 6 7 8]

[1 2 3 4]

[5 6 7 8]]

[[[1 2]

[3 4]]

[[5 6]

[7 8]]]
```

Wyjściowy kształt changeIt jest wektorem, ale użycie funkcji reshape zamienia go w macierz. Ponadto możesz ukształtować macierz w dowolną liczbę wymiarów, które współpracują z danymi. Musisz jednak zapewnić kształt pasujący do wymaganej liczby elementów. Na przykład wywołanie changeIt.reshape(2,3,2) zakończy się niepowodzeniem, ponieważ nie ma wystarczającej liczby elementów, aby zapewnić macierz o tym rozmiarze. W niektórych sformułowaniach algorytmów możesz napotkać dwie ważne operacje na macierzach. Są transpozycją i odwrotnością macierzy. Transpozycja ma miejsce, gdy macierz o kształcie n x m jest przekształcana w macierz m x n poprzez zamianę wierszy z kolumnami. Większość tekstów wskazuje tę operację za pomocą indeksu górnego T, tak jak w AT. Widzisz tę operację najczęściej używaną do mnożenia w celu uzyskania właściwych wymiarów. Podczas pracy z numpy używasz funkcji transpozycji, aby wykonać wymaganą pracę. Na przykład, zaczynając od macierzy, która ma dwa wiersze i cztery kolumny, można ją transponować tak, aby zawierała cztery wiersze, każdy z dwoma kolumnami, jak pokazano w tym przykładzie:

```
changeIt = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

print(np.transpose(changeIt))
```

Wynik pokazuje efekty transpozycji:

```
[1 5]

[2 6]

[3 7]

[48]]
```

Odwracanie macierzy stosuje się do macierzy o kształcie m x m, które są macierzami kwadratowymi o tej samej liczbie wierszy i kolumn. Ta operacja jest dość ważna, ponieważ pozwala na natychmiastowe rozwiązanie równań obejmujących mnożenie macierzy, takich jak y=bX, gdzie musisz odkryć wartości w wektorze b. Ponieważ większość liczb skalarnych (wyjątki obejmują zero) ma liczbę, której pomnożenie daje wartość 1, pomysł polega na znalezieniu macierzy odwrotnej, której pomnożenie da

w wyniku specjalną macierz zwaną macierzą jednostkową. Aby zobaczyć macierz tożsamości w numpy, użyj funkcji tożsamości, takiej jak ta:

print(np.tożsamość(4))

Oto wynik działania tej funkcji:

[1. 0. 0. 0.]

[0. 1. 0. 0.]

[0. 0. 1. 0.]

[0. 0. 0. 1.]]

Zauważ, że macierz jednostkowa zawiera wszystkie jedynki na przekątnej. Znalezienie odwrotności skalara jest dość łatwe (liczba skalarna n ma odwrotność n-1, czyli 1/n). To inna historia dla matrycy. Odwracanie macierzy wymaga dość dużej liczby obliczeń. Odwrotność macierzy A oznaczono jako A–1. Podczas pracy z numpy używasz funkcji linalg.inv, aby utworzyć odwrotność. Poniższy przykład pokazuje, jak utworzyć odwrotność, użyć jej do uzyskania iloczynu skalarnego, a następnie porównać ten iloczyn skalarny z macierzą jednostkową przy użyciu funkcji allclose:

a = np.array([[1,2], [3,4]])

b = np.linalg.inv(a)

print(np.allclose(np.dot(a,b), np.identity(2)))

Wyjście z tego kodu to

True

Czasami znalezienie odwrotności macierzy jest niemożliwe. Gdy macierz nie może być odwrócona, jest określana jako macierz osobliwa lub macierz zdegenerowana. Pojedyncze matryce nie są normą; są dość rzadkie.

Rozszerzenie analizy na tensory

Prostym sposobem na rozpoczęcie patrzenia na tensory jest to, że zaczynają się jako uogólniona macierz, która może mieć dowolną liczbę wymiarów. Mogą być 0-D (skalarne), 1-D (wektor) lub 2-D (macierz). W rzeczywistości tensory mogą mieć więcej wymiarów, niż można sobie wyobrazić. Tensory mają liczbę wymiarów potrzebną do przekazania znaczenia jakiegoś obiektu za pomocą danych. Mimo że większość ludzi postrzega dane jako macierz 2-D mając wiersze zawierające pojedyncze obiekty i kolumny, które mają indywidualne elementy danych, które definiują te obiekty, w wielu przypadkach macierz 2D nie wystarczy. Na przykład możesz potrzebować przetworzyć dane, które zawierają element czasu, tworząc macierz 2-D dla każdej obserwowanej chwili. Wszystkie te sekwencje macierzy 2-D wymagają do przechowywania struktury 3-D, ponieważ trzecim wymiarem jest czas. Jednak tensory są czymś więcej niż tylko wymyślnym rodzajem matrycy. Reprezentują jednostkę matematyczną, która żyje w strukturze wypełnionej innymi bytami matematycznymi. Wszystkie te byty oddziałują ze sobą w taki sposób, że transformacja bytów jako całości oznacza, że poszczególne tensory muszą przestrzegać określonej reguły transformacji. Dynamiczny charakter tensorów odróżnia je od standardowych macierzy. Każdy tensor w strukturze reaguje na zmiany w każdym innym tensorze, który występuje jako część transformacji. Aby zastanowić się, jak tensory działają w odniesieniu do głębokiego uczenia, rozważmy, że algorytm może wymagać trzech danych wejściowych do działania, co wyraża ten wektor:

inputs = np.array([5, 10, 15])

Są to pojedyncze wartości oparte na pojedynczym zdarzeniu. Być może stanowią pytanie o to, który detergent jest najlepszy na Amazon. Jednak zanim będzie można wprowadzić te wartości do algorytmu, należy zważyć ich wartości na podstawie szkolenia przeprowadzonego na modelu. Innymi słowy, biorąc pod uwagę detergenty kupowane przez dużą grupę ludzi, macierz pokazuje, który z nich jest faktycznie najlepszy przy określonych nakładach. Nie chodzi o to, że detergent jest najlepszy w każdej sytuacji, po prostu o to, że stanowi najlepszą opcję przy określonych danych wejściowych. Ważenie wartości pomaga odzwierciedlić to, czego aplikacja do uczenia głębokiego nauczyła się dzięki analizie ogromnych zbiorów danych. Dla celów argumentacji, możesz zobaczyć wagi w macierzy, która następuje jako wyuczone wartości:

weights = np.array([[.5,.2,-1], [.3,.4,.1], [-.2,.1,.3]])

Teraz, gdy dla danych wejściowych dostępne jest ważenie, możesz przekształcić dane wejściowe w oparciu o uczenie się algorytmu wykonanego w przeszłości:

result = np.dot(inputs, weights)

Wyjście

[2,5 6,5 0,5]

przekształca oryginalne dane wejściowe tak, aby teraz odzwierciedlały efekty uczenia się. Wektor wejściowy jest warstwą ukrytą w sieci neuronowej, a wynikowy wynik jest następną warstwą ukrytą w tej samej sieci neuronowej. Przekształcenia lub inne działania zachodzące w każdej warstwie określają, w jaki sposób każda ukryta warstwa wpływa na całą sieć neuronową, która w tym przypadku była ważona. Późniejsze rozdziały pomogą Ci zrozumieć koncepcje warstw, ważenia i innych czynności w sieci neuronowej. Na razie zastanów się po prostu, że każdy tensor oddziałuje ze strukturą na podstawie działań każdego innego tensora.

Efektywne wykorzystanie wektoryzacji

Wektoryzacja to proces, w którym aplikacja przetwarza wiele wartości skalarnych jednocześnie, a nie pojedynczo. Głównym powodem korzystania z wektoryzacji jest oszczędność czasu. W wielu przypadkach procesor będzie zawierał specjalną instrukcję związaną z wektoryzacją, taką jak instrukcja SSE w systemach x86 (https://docs.oracle.com/cd/E26502_01/html/E28388/eojde.html). Zamiast wykonywania pojedynczych instrukcji w pętli, podejście wektoryzacji wykona je jako grupę, znacznie przyspieszając proces. Podczas pracy z ogromnymi ilościami danych ważna jest wektoryzacja, ponieważ tę samą operację wykonuje się wiele razy. Wszystko, co możesz zrobić, aby utrzymać proces poza pętlą, przyspieszy wykonanie całego kodu. Oto przykład prostej wektoryzacji:

def doAdd(a, b):

return a + b

vectAdd = np.vectorize(doAdd)

print(vectAdd([1, 2, 3, 4], [1, 2, 3, 4]))

Po wykonaniu tego kodu otrzymasz następujące dane wyjściowe:

[2 4 6 8]

Funkcja vectAdd działała na wszystkich wartościach jednocześnie, w jednym wywołaniu. W związku z tym funkcja doAdd, która zezwala tylko na dwa wejścia skalarne, została rozszerzona, aby umożliwić jednocześnie cztery wejścia. Ogólnie wektoryzacja oferuje następujące korzyści:

- Kod, który jest zwięzły i łatwiejszy do odczytania
- Zredukowany czas debugowania z powodu mniejszej liczby linii kodu
- Środki do dokładniejszego przedstawiania wyrażeń matematycznych w kodzie
- Zmniejszona liczba nieefektywnych pętli

Interpretowanie uczenia się jako optymalizacji

Do tej pory omówiono dane jako abstrakcję, przekształcanie danych w użyteczne formy, przechowywanie danych w macierzy oraz podstawy manipulowania tą macierzą po zbudowaniu. Wszystkie te rzeczy prowadzą do możliwości zautomatyzowania przetwarzania danych, dzięki czemu można znaleźć przydatne wzorce. Na przykład zbiór pikseli, najmniejszy element obrazu, to po prostu seria liczb w macierzy. Zlokalizowanie określonej twarzy na tym obrazie wymaga manipulacji tymi liczbami, aby znaleźć określone sekwencje, które utożsamiają się z twarzą. Wkrótce zdajesz sobie sprawę, że znalezienie wzorca, a następnie jego poprawna interpretacja wymaga czasu, nawet dla komputera, aby wykonał go z jakąkolwiek dokładnością. Oczywiście zawsze liczy się czas. Odkrycie, że przestępca wszedł na lotnisko godzinę po fakcie, jest bezużyteczne – odkrycie musi nastąpić tak szybko, jak to możliwe. Aby tak się stało, manipulacja danymi i rozpoznawanie wzorców muszą nastąpić tak szybko, jak to możliwe, co oznacza optymalizację procesu. Optymalizacja oznacza po prostu znalezienie sposobów na szybsze wykonanie zadania bez utraty dużej lub niczego na drodze do dokładności. Uczenie się, z perspektywy komputera, ma miejsce, gdy aplikacja znajdzie sposób na skuteczne przeprowadzenie optymalizacji. Musisz pamiętać, że uczenie się na komputerze różni się od uczenia się przez człowieka tym, że komputer tak naprawdę tego nie robi aby zrozumieć wszystko, co nowe, kiedy ma miejsce nauka. Komputer może po prostu manipulować danymi z większą szybkością i dokładnością, aby zlokalizować interesujące wzorce. Pozostała część tej książki szczegółowo omawia koncepcję optymalizacji, ale kolejne sekcje dają szybki przegląd tego, co oznacza optymalizacja manipulacji.

Odkrywanie funkcji kosztów

Ludzie dość dobrze rozumieją ideę kosztów. Idziesz do jednego sklepu i stwierdzasz, że produkt kosztuje określoną kwotę. Wiesz jednak, że inny sklep sprzedaje dokładnie ten sam produkt za mniej. Produkty są takie same w obu przypadkach, więc kupujesz przedmiot w sklepie, który sprzedaje go za mniej. Ta sama zasada kosztów dotyczy uczenia się na komputerze. Komputer może zapewnić wiele metod znajdowania pożądanych wzorów, ale tylko jedna z tych metod da wynik o pożądanej dokładności w wymaganym czasie. Metoda, która działa najlepiej, ta, której ostatecznie użyjesz, ma najniższy koszt. Na przykład, aby rozwiązać problem, może być konieczne przewidzenie liczby lub klasy. Możliwe jest przekształcenie każdego z tych problemów w koszt, który algorytm głębokiego uczenia może wykorzystać do określenia, czy jego przewidywanie jest prawidłowe. Zadanie to wykonuje się za pomocą funkcji kosztu (zwanej również funkcją straty), która mierzy różnicę między poprawną odpowiedzią a odpowiedzią dostarczoną przez algorytm głębokiego uczenia się. Wynikiem funkcji kosztu jest różnica między wartością poprawną a wartością przewidywaną w postaci liczby. Funkcja kosztu jest tym, co naprawdę napędza sukces głębokiego uczenia, ponieważ decyduje o tym, czego algorytm się uczy. Musisz mądrze wybrać odpowiednią funkcję kosztową dla swojego problemu. Oto funkcje kosztów, które są często używane w przypadku uczenia głębokiego:

- Błąd średniokwadratowy: przyjmuje kwadrat różnicy między wartością poprawną a wartością przewidywaną przez algorytm. Gdy różnica jest duża, wartość podniesiona do kwadratu jest jeszcze większa, co podkreśla błąd algorytmu.
- Cross entropia lub log loss: Ocenia błędy przewidywania za pomocą logarytmu. Algorytmy głębokiego uczenia wykorzystują prawdopodobieństwa do udzielania odpowiedzi. (Nie podają prawdopodobieństwa, ale wynik ma pewne prawdopodobieństwo). Prawdopodobieństwa są oparte na ich poprawności i są przekształcane na miarę liczbową, która reprezentuje błąd.

Znajomość kosztów, jakie generuje algorytm głębokiego uczenia się podczas zgadywania wyników, to tylko jedna z części procesu. Tak jak ludzie uczą się na błędach, gdy są o nich świadome, uczenie głębokie uczy się na podstawie wyników funkcji kosztów. Koszt oznacza znalezienie metody, która wykonuje zadania w sposób optymalny. Słowo „optymalne" jest celowo nieprecyzyjne, ponieważ to, co może wydawać się optymalne w jednej sytuacji, może nie być optymalne w innej. Optymalnym rozwiązaniem jest takie, które będzie nadal lokalizować wymagane wzorce w minimalnym czasie z określoną dokładnością na dużej liczbie elementów danych. Stworzenie metody działającej z danymi, o których wiesz, nie popłaca. To, czego potrzebujesz, to optymalna metoda radzenia sobie z danymi, o których dzisiaj nie wiesz.

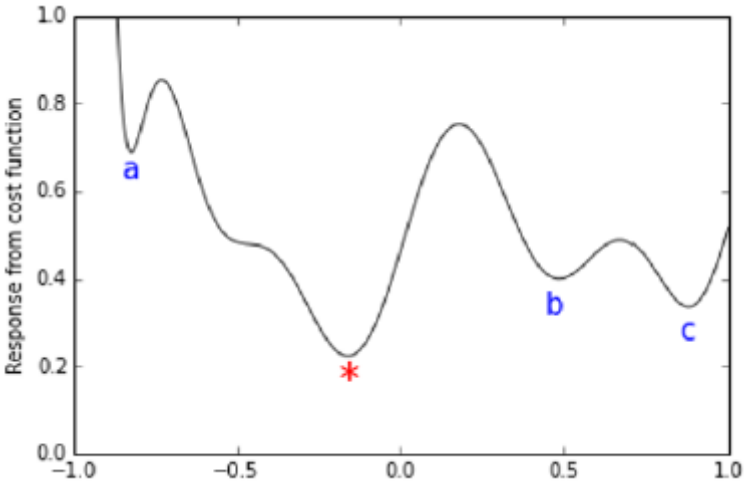
Opadająca krzywa błędu

Kiedy człowiek popełnia błąd i ktoś go widzi, druga osoba przekazuje informację zwrotną, aby pomóc pierwszej osobie zrozumieć naturę błędu i właściwe rozwiązanie. Pojedyncza sesja informacji zwrotnej może nie wystarczyć, aby pomóc osobie naprawić błąd; dlatego powtórzenie informacji zwrotnej może być konieczne, aby pomóc osobie w stopniowym naprawieniu błędu. Podobnie automatyzacja zapewniana przez głębokie uczenie wymaga korekty przez kolejne poprawki. Po wykryciu błędu automatyka zapewnia korektę algorytmów realizujących przetwarzanie. Ta pętla sprzężenia zwrotnego poprawia z czasem odpowiedzi udzielane przez rozwiązanie do uczenia głębokiego, co sprawia, że rozwiązanie jest dokładniejsze w znajdowaniu prawidłowych wzorców. W miarę postępu tego procesu poziom błędu mierzony przez funkcję kosztu zmniejsza się, rysując w ten sposób krzywą opadającą. Funkcja kosztu steruje właśnie opisanym procesem, ale wymaga innych algorytmów, takich jak optymalizacja i korekcja błędów, aby dokonać rzeczywistych zmian. Funkcja kosztu zgłasza poziom błędu tylko wtedy, gdy model głębokiego uczenia generuje prognozę. Na potrzeby tego tekstu różne algorytmy osiągają różne rodzaje optymalizacji. Gradient Descent, Stochastic Gradient Descent, Momentum, Adagrad, RMSProp, Adadelta i Adam to warianty tej samej koncepcji optymalizacji, którą omawiamy w dalszej części książki. Korekcja błędów opiera się na innym algorytmie zwanym propagacją wsteczną. Funkcja błędu wysyła informację zwrotną przez sieć neuronową w postaci wag, które wpływają na sposób, w jaki rozwiązanie przekształca dane wejściowe, aby zapewnić poprawność danych wyjściowych.

Nauka właściwego kierunku

Zejście gradientowe jest szeroko stosowanym podejściem do określania, jakie poprawki są potrzebne, aby model głębokiego uczenia się działał lepiej przy określonym błędzie. Zawsze zaczyna się od wstępnej konfiguracji sieci głębokiego uczenia i przekłada informację zwrotną z funkcji kosztów na ogólną poprawkę, która ma zostać rozesłana do głębokiego uczenia węzłów sieciowych. Ten proces wymaga wykonania kilku iteracji - do momentu, gdy wynik funkcji kosztu znajdzie się w żądanym zakresie. W przenośni możesz zobaczyć schodzenie w dół jako kapitana łodzi, która musi płynąć drogą wodną, aby ominąć liczne przeszkody, takie jak skały lub góry lodowe. Ponieważ kapitan dostrzega niebezpieczeństwo (błąd zgłaszany przez funkcję kosztów), wprowadza poprawkę na koło sterowe, która pozwala uniknąć kolizji. Oczywiście kapitan przekazuje poprawkę załodze. Załoga wykorzystuje

te informacje do sterowania silnikami i sterami statku, co jest częścią historii odgrywaną przez algorytm wstecznej propagacji błędów. W oparciu o funkcję kosztów sieć wymaga również optymalizacji w celu zminimalizowania błędu. Jednak optymalizacja odbywa się tylko na danych uczących. Niestety idealna optymalizacja danych treningowych może prowadzić do nadmiernego dopasowania. Rozpoznawanie problemów, takich jak nadmierne dopasowanie, jest miejscem, w którym pojawia się artystyczny aspekt głębokiego uczenia się; musisz zoptymalizować przy użyciu danych uczących, ale nie zoptymalizować całkowicie (przekroczenie), aby wynikowy model działał dobrze również na danych testowych. Ta równoważąca czynność polegająca na znalezieniu odpowiedniego poziomu optymalizacji jest uogólnieniem. Naprawienie ograniczonej liczby iteracji optymalizacji lub zatrzymanie optymalizacji, gdy zauważysz, że model zaczyna słabiej działać na danych testowych, które są oddzielone od danych uczących (proces zwany wczesnym zatrzymaniem) to typowe strategie osiągania optymalizacji uczenia głębokiego. Interesującą kwestią jest to, że seria poprawek dostarczanych przez algorytm gradientu może ostatecznie nie być optymalna. Ustalenie, jak skutecznie naprawić pojedynczy błąd, jest proste; poprawianie wielu błędów jednocześnie może okazać się trudne. W wielu przypadkach algorytm optymalizacji utknął w ślepym zaułku i nie może znaleźć właściwego sposobu na poprawę wydajności sieci neuronowej, jak pokazano na rysunku .



Ta sytuacja jest lokalnym minimem, w którym rozwiązanie wydaje się działać optymalnie, chociaż w rzeczywistości tak nie jest, ponieważ dalsze poprawki mogą nadal poprawiać wydajność.

Rysunek powyższy przedstawia przykład procesu optymalizacji z wieloma lokalnymi minimami (minimalne punkty na zakrzywieniu oznaczone literami), w których proces optymalizacji może zostać uderzony i nie może kontynuować zejścia w kierunku głębokiego minimum oznaczonego gwiazdką. W procesie optymalizacji dla modelu uczenia głębokiego, rozróżniasz różne wyniki optymalizacji. Możesz mieć globalne minimum, dobry model, który generuje prognozy z najmniejszym możliwym błędem dla problemu i wiele lokalnych minimów, rozwiązania, które wydają się zapewniać najlepszą korekcję błędów, ale w rzeczywistości tego nie robią. Oprócz lokalnych minimów, inne problemy, które możesz napotkać podczas optymalizacji, to punkty siodła. W punktach siodłowych nie masz minimum, ale Twoja optymalizacja gwałtownie spowalnia, skłaniając Cię do przekonania, że algorytm osiągnął minimum. W rzeczywistości punkty siodłowe stanowią jedynie pauzę optymalizacyjną. Upierając się, że algorytm zmierza w określonym kierunku optymalizacji, zapewniasz, że może on łatwo ominąć punkty siodłowe i kontynuować redukcję błędów. Oto sposoby na zwiększenie szans na uzyskanie zoptymalizowanych i dobrze działających algorytmów:

- Przygotuj dane do nauki zgodnie z potrzebami, aby odzwierciedlić problem

- Wybierz różne warianty optymalizacji i ustaw ich naukę według potrzeb
- Ustaw inne kluczowe cechy sieci głębokiego uczenia

Aktualizacja

Aktualizacja sieci neuronowej za pomocą wag może przybrać jedną z dwóch form: stochastyczną i wsadową. Podczas wykonywania aktualizacji stochastycznych każde wejście generuje indywidualnie korektę wagi. Takie podejście ma tę zaletę, że zmniejsza ryzyko, że algorytm utknie w lokalnych minimach. Podczas wykonywania aktualizacji wsadowych błąd kumuluje się w pewien sposób i następuje korekta masy gdy partia jest kompletna. Zaletą tego podejścia jest to, że nauka przebiega szybciej, ponieważ wpływ dostosowania wagi jest większy. Najlepszym sposobem uczenia się głębokiej sieci neuronowej jest próba zminimalizowania błędów wszystkich przykładów naraz. Ten cel nie zawsze jest możliwy, ponieważ dane mogą być zbyt duże, aby zmieścić się w pamięci. Aktualizacje wsadowe są najlepszą możliwą strategią w wielu przypadkach, przy czym rozmiary partii są największe z możliwych dla używanego sprzętu.

Tworzenie podstaw regresji liniowej

Termin regresja liniowa może wydawać się skomplikowany, ale tak nie jest. Regresja liniowa to zasadniczo linia prosta poprowadzona przez szereg współrzędnych x/y, które określają położenie punktu danych. Punkty danych mogą nie zawsze leżeć bezpośrednio na linii, ale linia pokazuje, gdzie punkty danych wypadłyby w idealnym świecie współrzędnych liniowych. Używając linii, możesz przewidzieć wartość y (zmienna kryterium) przy wartości x (zmienna predykcyjna). Kiedy masz tylko jedną zmienną predykcyjną, masz prostą regresję liniową. Dla kontrastu, gdy masz wiele predyktorów, masz wielokrotną regresję liniową, która nie opiera się na linii, ale raczej na płaszczyźnie rozciągającej się przez wiele wymiarów. Głębokie uczenie wykorzystuje dane wejściowe do odgadnięcia płaszczyzny nieliniowej, która najwłaściwiej przejdzie przez środek zestawu punktów danych w bardziej wyrafinowany sposób niż regresja liniowa. Ma pewne kluczowe cechy z regresją liniową, która jest głównym tematem tej Częsci: Aby opowiedzieć Ci o regresji liniowej i dostarczyć przydatnych pomysłów, które możesz później przenieść do głębokiego uczenia się. W pierwszej części omówiono zmienne i sposób, w jaki pracujesz z nimi, aby stworzyć regresję liniową. Idąc dalej, powiedzmy, że stworzyłeś model regresji liniowej, ale linia oddziela dwie kategorie. Punkty danych po jednej stronie linii to jedno, a punkty danych po drugiej stronie linii to co innego. Sieć neuronowa może wykorzystać regresję liniową do określenia prawdopodobieństwa, że punkt danych znajduje się po jednej lub drugiej stronie linii. Wiedząc, z jakim rodzajem obiektu (wyrażonym w punkcie danych) masz do czynienia, możesz kategoryzować obiekt - to znaczy określić, do jakiej grupy obiektów należy. Istotą wykonywania całej tej pracy jest wypracowanie rozwiązania problemu. Na przykład możesz mieć całą listę punktów danych i musisz wiedzieć, do której grupy należy każdy punkt danych, co byłoby żmudnym zadaniem bez jakiejś automatyzacji. Jednak, aby stworzyć prawidłowe rozwiązanie danego problemu, musisz mieć odpowiednie dane, co oznacza określenie właściwych danych wejściowych lub funkcji do użycia. W trzeciej części tego rozdziału omówiono, jak wybrać funkcje, które będą najlepsze aby odpowiedzieć na pytania, które musisz rozważyć. Wreszcie, wykorzystujemy to, co do tej pory odkryłeś, aby rozwiązać prosty problem za pomocą stochastycznego spadku gradientu (SGD). Złożenie wszystkiego razem sprawi, że zastosowanie regresji liniowej w rozwiązywaniu problemów stanie się jasne.

Łączenie zmiennych

Regresja ma długą historię w różnych dziedzinach: statystyce, ekonomii, psychologii, naukach społecznych i naukach politycznych. Regresja liniowa, oprócz możliwości dokonywania szerokiego zakresu przewidywań obejmujących wartości liczbowe, klasy binarne i wielokrotne, prawdopodobieństwa i dane liczbowe, pomaga również zrozumieć różnice grupowe, modelować preferencje konsumentów i określać ilościowo ważność funkcji w modelu. Pozbawiona większości właściwości statystycznych regresja liniowa pozostaje prostym, zrozumiałym, ale skutecznym algorytmem przewidywania wartości i klas. Szybka w szkoleniu, łatwa do wyjaśnienia osobom nietechnicznym i prosta do wdrożenia w dowolnym języku programowania, regresja liniowa i logistyczna to pierwszy wybór większości praktyków głębokiego uczenia się podczas budowania modeli do porównania z bardziej zaawansowanymi rozwiązaniami (model bazowy). Ludzie używają ich również do określania kluczowych cech problemu, eksperymentowania i uzyskiwania wglądu w tworzenie cech.

Praca poprzez prostą regresję liniową

Musisz odróżnić statystyczne zapisy regresji liniowej, które obejmują wykreślanie współrzędnych i rysowanie przez nie linii, od algorytmu używanego przez głębokie uczenie do przewidywania położenia tej linii na wykresie. Regresja liniowa działa poprzez łączenie cech liczbowych poprzez sumowanie.

Dodanie stałej liczby, zwanej odchyleniem, kończy sumowanie. Błąd systematyczny reprezentuje linię bazową predykcji, gdy wszystkie cechy mają wartości zero. Odchylenie może odgrywać ważną rolę w tworzeniu domyślnych prognoz, zwłaszcza gdy brakuje niektórych funkcji (a więc mają wartość zerową). Oto typowy wzór na regresję liniową:

$y = \beta X + \alpha$

W tym wyrażeniu y jest wektorem wartości odpowiedzi. Możliwe wektory odpowiedzi to ceny domów w mieście lub sprzedaż produktu, czyli po prostu dowolna odpowiedź liczbowa, taka jak miara lub ilość. Symbol X określa macierz cech, których należy użyć do odgadnięcia wektora y. X to macierz, która zawiera tylko wartości liczbowe. Grecka litera alfa (α) reprezentuje obciążenie, które jest stałą, podczas gdy litera beta (β) jest wektorem współczynników, które model regresji liniowej używa z obciążeniem do tworzenia prognozy. Używanie greckich liter alfa i beta w regresji jest szeroko rozpowszechnione do tego stopnia, że większość praktyków nazywa wektor współczynników regresją beta. Możesz nadać sens temu wyrażeniu na różne sposoby. Upraszczając, możesz sobie wyobrazić, że X w rzeczywistości składa się z jednej cechy (opisywanej jako predyktor w praktyce statystycznej), więc możesz ją przedstawić jako wektor o nazwie x. Gdy dostępny jest tylko jeden predyktor, obliczenie jest prostą regresją liniową. Teraz, gdy ty masz prostsze sformułowanie, twoja algebra i geometria ze szkoły średniej mówią ci, że sformułowanie y=bx+a jest prostą w płaszczyźnie współrzędnych utworzoną z osi x (odciętej) i osi y (rzędnej).

Przejście do wielokrotnej regresji liniowej

Świat rzadko oferuje problemy, które mają tylko jedną cechę. Przewidując ceny domów, należy wziąć pod uwagę różne kwestie, takie jak sąsiedztwo i liczba pokoi w domu. W przeciwnym razie ludzie mogliby rozwiązać większość problemów bez korzystania z automatyzacji, takiej jak głębokie uczenie. Kiedy masz więcej niż jedną cechę (wielokrotna regresja liniowa), nie możesz już używać prostej płaszczyzny współrzędnych złożonej z x i y. Przestrzeń obejmuje teraz wiele wymiarów, a każdy wymiar jest cechą. Teraz Twoja formuła jest bardziej skomplikowana, składa się z wielu wartości x, z których każda jest ważona własną wersją beta. Na przykład, jeśli masz cztery cechy (tak, że przestrzeń jest czterowymiarowa), formuła regresji, jak wyjaśniono na podstawie formy macierzowej, jest

$y = x_1b_1+x_2b_2+x_3b_3+x_4b_4+a$

Ta złożona formuła, która istnieje w przestrzeni wielowymiarowej, nie jest już linią, lecz płaszczyzną o tylu wymiarach, co przestrzeń. Jest to hiperpłaszczyzna, a jej powierzchnia indywidualizuje wartości odpowiedzi dla każdej możliwej kombinacji wartości w wymiarach cech. Ta dyskusja wyjaśnia regresję w jej znaczeniu geometrycznym, ale możesz również postrzegać ją jako dużą sumę ważoną. Możesz rozłożyć odpowiedź na wiele części, z których każda odnosi się do funkcji i przyczynia się do określonej części. Znaczenie geometryczne jest szczególnie przydatne do omawiania właściwości regresji, ale znaczenie sumowania ważonego pomaga lepiej zrozumieć praktyczne przykłady. Na przykład, jeśli chcesz przewidzieć model wydatków na reklamę, możesz użyć modelu regresji i utworzyć taki model:

sales = advertising*badv + shops*bshop + price*bprice + a

W tym ujęciu sprzedaż to suma wydatków na reklamę, liczby sklepów dystrybuujących produkt oraz ceny produktu. Możesz szybko zdemistyfikować regresję liniową, wyjaśniając jej składniki. Po pierwsze, masz odchylenie, stałą a, która działa jako punkt wyjścia. Następnie masz trzy wartości cech, każda wyrażona w innej skali (reklama to dużo pieniędzy, cena to przystępna wartość, a sklepy to liczba dodatnia), każda przeskalowana przez odpowiedni współczynnik beta. Każda beta przedstawia wartość liczbową, która opisuje intensywność związku z odpowiedzią. Posiada również znak, który pokazuje

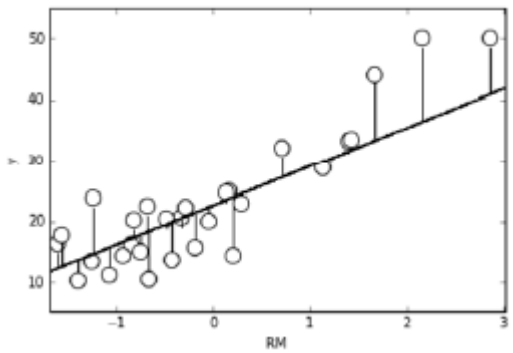
efekt zmiany funkcji. Kiedy współczynnik beta jest bliski zeru, wpływ cechy na odpowiedź jest słaby, ale jeśli jego wartość jest daleka od zera, dodatnia lub ujemna, efekt jest znaczący i cecha jest ważna dla modelu regresji. Aby uzyskać szacunkową wartość docelową, skalujesz każdą beta do miary obiektu. Wysoka beta zapewnia większy lub mniejszy wpływ na odpowiedź w zależności od skali funkcji. Dobrym zwyczajem jest standaryzacja cech (poprzez odjęcie średniej i podzielenie przez odchylenie standardowe), aby uniknąć zmylenia wysokich wartości beta na cechach na małą skalę i porównywania różnych współczynników beta. Otrzymane wartości beta są porównywalne, co pozwala określić, które z nich mają największy wpływ na odpowiedź (te o największej wartości bezwzględnej). Jeśli beta jest pozytywna, zwiększenie funkcji zwiększy odpowiedź, a zmniejszenie funkcji zmniejszy odpowiedź. I odwrotnie, jeśli beta jest ujemna, reakcja będzie działać wbrew tej funkcji: gdy jedna rośnie, druga maleje. Każda wersja beta w regresji reprezentuje wpływ.

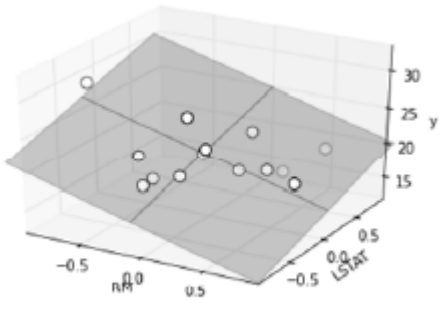
W tym zejście gradientowe

Korzystając z algorytmu opadania gradientu omówionego w dalszej części, regresja liniowa może znaleźć najlepszy zestaw współczynników beta (i obciążenia), aby zminimalizować funkcję kosztu określoną przez kwadrat różnicy między przewidywaniami a wartościami rzeczywistymi:

$$J(w) = \frac{1}{2n}\sum(Xw - y)^2$$

Ten wzór określa koszt J w funkcji w, wektora współczynników modelu liniowego. Koszt to zsumowana, podniesiona do kwadratu różnica wartości odpowiedzi od wartości przewidywanych (mnożenie Xw) podzielona przez dwukrotność liczby obserwacji (n). Algorytm dąży do znalezienia minimalnych możliwych wartości rozwiązania dla różnicy między rzeczywistymi wartościami docelowymi a przewidywaniami uzyskanymi z regresji liniowej. Wynik optymalizacji można wyrazić graficznie jako odległości w pionie między punktami danych a linią regresji. Linia regresji dobrze reprezentuje zmienną odpowiedzi, gdy odległości są małe, jak pokazano na rysunku (z prostą regresją liniową po lewej stronie i wielokrotną regresją liniową po prawej). Jeśli zsumujesz kwadraty odległości (długość linii łączącej punkt danych z linią regresji na rysunku), suma ta jest zawsze minimalna z możliwych, gdy poprawnie obliczysz linię regresji. (Żadna inna kombinacja wersji beta nie spowoduje mniejszego błędu).





W statystyce praktycy często wskazują na szacowanie rozwiązania regresji liniowej na podstawie rachunku macierzowego (tzw. rozwiązywanie przez formę zamkniętą). Stosowanie tego podejścia nie zawsze jest wykonalne, a obliczenia są dość powolne, gdy macierz wejściowa jest duża. W głębokim uczeniu uzyskujesz te same wyniki, korzystając z funkcji Gradient Descent Optimization (GDO), która łatwiej i szybciej obsługuje większe ilości danych, szacując w ten sposób rozwiązanie z dowolnej macierzy wejściowej.

Widząc regresję liniową w akcji

Poniższy przykład w Pythonie używa zestawu danych Boston z Scikit-learn, aby spróbować odgadnąć ceny mieszkań w Bostonie za pomocą regresji liniowej. Przykład próbuje również określić, które zmienne mają większy wpływ na wynik. Poza kwestiami obliczeniowymi, standaryzacja predyktorów okazuje się całkiem przydatna, jeśli chcesz określić wpływowe zmienne:

```
from sklearn.datasets import load_boston

from sklearn.preprocessing import scale

boston = load_boston()

X, y = scale(boston.data), boston.target
```

Klasa regresji w Scikit-learn jest częścią modułu linear_model. Ponieważ wcześniej skalowałeś zmienne X, nie musisz decydować o żadnych innych przygotowaniach ani specjalnych parametrach podczas korzystania z tego algorytmu:

```
from sklearn.linear_model import LinearRegression

regression = LinearRegression()

regression.fit(X, y)
```

Teraz, gdy algorytm jest już dopasowany, możesz użyć metody punktowej, aby zgłosić miarę R2:

```
print('R2 %0.3f' % regression.score(X, y))
```

R2 0.741

TROCHĘ LEPIEJ ZROZUMIEĆ R^2

R^2, znany również jako współczynnik determinacji, jest miarą z zakresu od 0 do 1. Pokazuje, w jaki sposób użycie modelu regresji jest lepsze w przewidywaniu odpowiedzi niż przy użyciu prostej średniej. Współczynnik determinacji wywodzi się z praktyki statystycznej i bezpośrednio odnosi się do sumy kwadratów błędów. Możesz również rozumieć R^2 jako ilość informacji wyjaśnionych przez model (tak samo jak kwadrat korelacji), więc zbliżenie się do 1 oznacza możliwość wyjaśnienia większości danych za pomocą modelu. Obliczanie R^2 na tym samym zestawie danych użytych do treningu jest powszechne w statystyce. W nauce o danych i uczeniu głębokim zawsze lepiej jest testować wyniki na danych, które nie są wykorzystywane do szkolenia. Złożone algorytmy mogą zapamiętywać dane, zamiast się z nich uczyć. W pewnych okolicznościach ten problem może również wystąpić, gdy używasz prostszych modeli, takich jak regresja liniowa.

Aby zrozumieć, co kieruje szacunkami w modelu regresji wielorakiej, należy przyjrzeć się atrybutowi współczynniki_, który jest tablicą zawierającą współczynniki beta regresji. Wyświetlając atrybut boston.DESCR, możesz zrozumieć odwołanie do zmiennej:

```
print([a + ':' + str(round(b, 1)) for a, b in

zip(boston.feature_names, regression.coef_)])

['CRIM:-0.9', 'ZN:1.1', 'INDUS:0.1', 'CHAS:0.7',

'NOX:-2.1', 'RM:2.7', 'AGE:0.0', 'DIS:-3.1',

'RAD:2.7', 'TAX:-2.1', 'PTRATIO:-2.1',
```

'B:0.9', 'LSTAT:-3.7']

Zmienna DIS, która zawiera ważone odległości do pięciu ośrodków zatrudnienia, ma największą bezwzględną zmianę jednostkową. W nieruchomościach dom, który jest zbyt daleko od zainteresowań ludzi (takich jak praca), obniża wartość. Zamiast tego WIEK lub INDUS, które są proporcjami opisującymi wiek budynku i to, czy w okolicy dostępne są zajęcia niehandlowe, nie wpływają tak bardzo na wynik; bezwzględna wartość ich współczynników beta jest znacznie niższa. Możesz się zastanawiać, dlaczego przykłady w rozdziale nie wykorzystują Keras i TensorFlow. Korzystanie z tych bibliotek jest możliwe, ale pakiety uczenia głębokiego najlepiej nadają się do rozwiązań z zakresu uczenia głębokiego. Używanie ich do prostszych modeli oznacza nadmierne komplikowanie rozwiązania. Scikit-learn oferuje przejrzyste i proste implementacje modeli regresji liniowej, które pomagają zrozumieć, jak te algorytmy działają lepiej.

Mieszanie typów zmiennych

Sporo problemów pojawia się z efektywnym, ale prostym narzędziem do regresji liniowej. Czasami, w zależności od danych, z których korzystasz, problemy są większe niż korzyści z używania tego narzędzia. Najlepszym sposobem określenia, czy regresja liniowa zadziała, jest użycie algorytmu i przetestowanie jego skuteczności na danych.

Modelowanie odpowiedzi

Regresja liniowa może modelować odpowiedzi tylko jako dane ilościowe. Kiedy potrzebujesz modelować kategorie jako odpowiedź, musisz przejść do regresji logistycznej. Pracując z predyktorami, najlepiej radzisz sobie, używając ciągłych zmiennych liczbowych; chociaż możesz dopasować zarówno liczby porządkowe, jak i, przy niektórych przekształceniach, kategorie jakościowe. Zmienna jakościowa może wyrażać cechę koloru, taką jak kolor produktu, lub cechę wskazującą zawód danej osoby. Dostępnych jest kilka opcji przekształcania zmiennej jakościowej przy użyciu techniki, takiej jak kodowanie binarne (najczęstsze podejście). Tworząc binarną zmienną jakościową, tworzysz tyle obiektów, ile klas w elemencie. Każda cecha zawiera zerowe wartości, chyba że jej klasa pojawia się w danych, gdy przyjmuje wartość jeden. Ta procedura jest nazywana kodowaniem jednorazowym. Prosty przykład Pythona wykorzystujący moduł przetwarzania wstępnego Scikit-learn pokazuje, jak wykonać kodowanie one-hot:

```
from sklearn.preprocessing import OneHotEncoder

from sklearn.preprocessing import LabelEncoder

lbl = LabelEncoder()

enc = OneHotEncoder()

qualitative = ['red', 'red', 'green', 'blue',

'red', 'blue', 'blue', 'green']

labels = lbl.fit_transform(qualitative).reshape(8,1)

print(enc.fit_transform(labels).toarray())

[[ 0. 0. 1.]

[ 0. 0. 1.]

[ 0. 1. 0.]
```

[1. 0. 0.]

[0. 0. 1.]

[1. 0. 0.]

[1. 0. 0.]

[0. 1. 0.]]

W tym przypadku zobaczysz, co wygląda na trzy kolumny: niebieską, zieloną i czerwoną. Na przykład zauważ, że w elemencie tablicy [0, 2] widzisz wartość 1., co odpowiada wartości czerwonego na tej pozycji. Teraz spójrz na oryginalną tablicę, gdzie widzisz, że quality[0] rzeczywiście jest „czerwone”.

Modelowanie funkcji

W statystyce, ponieważ regresję liniową rozwiązujesz za pomocą formy zamkniętej, gdy chcesz zrobić zmienną binarną ze zmiennej jakościowej, przekształcasz wszystkie poziomy oprócz jednego, ponieważ używasz wzoru na obliczenie macierzy odwrotnej, która ma sporo ograniczeń . W głębokim uczeniu się używasz gradientu, więc zamiast tego przekształcasz wszystkie poziomy. Jeśli w macierzy danych brakuje danych i nie radzisz sobie z nimi właściwie, model przestanie działać. W związku z tym należy podać brakujące wartości (na przykład zastępując brakującą wartość średnią wartością obliczoną na podstawie samej cechy). Innym rozwiązaniem jest użycie wartości zerowej dla brakującego przypadku i utworzenie dodatkowej zmiennej binarnej, której wartości jednostek wskazują brakujące wartości w elemencie. Ponadto wartości odstające (wartości poza normalnym zakresem) zakłócają regresję liniową, ponieważ model stara się zminimalizować kwadratową wartość błędów (zwanych również resztami). Wartości odstające mają duże reszty, zmuszając algorytm do skupienia się bardziej na nich niż na zwykłych punktach.

Radzenie sobie ze złożonymi relacjami

Największym ograniczeniem regresji liniowej jest to, że model jest sumą niezależnych składników, ponieważ każda cecha występuje w sumowaniu osobno, pomnożona tylko przez własną wartość beta. Ta matematyczna forma doskonale nadaje się do wyrażenia sytuacji, w której cechy nie są ze sobą powiązane. Na przykład wiek i kolor oczu osoby nie są ze sobą powiązane, ponieważ nie mają na siebie wpływu. W ten sposób możesz uznać je za niezależne terminy, a w podsumowaniu regresji pozostawienie ich osobno ma sens. Porównaj terminy niepowiązane z terminami pokrewnymi. Na przykład wiek osoby i kolor włosów są powiązane, ponieważ starzenie się powoduje wybielanie włosów. Gdy umieścisz te funkcje w podsumowaniu regresji, to tak, jakbyś podsumował te same informacje. Z powodu tego ograniczenia nie można określić, jak przedstawić wpływ kombinacji zmiennych na wynik. Innymi słowy, swoimi danymi nie możesz przedstawiać złożonych sytuacji. Ponieważ model składa się z prostych kombinacji cech ważonych, na jego przewidywania bardziej wpływa błąd systematyczny niż wariancja. W rzeczywistości, po dopasowaniu obserwowanych wartości wyników, rozwiązanie proponowane przez modele liniowe jest zawsze proporcjonalnie przeskalowaną mieszanką cech. Niestety, nie można wiernie przedstawić niektórych relacji między odpowiedzią a funkcją, używając proporcjonalnie przeskalowanej kombinacji funkcji. W wielu przypadkach odpowiedź zależy od cech w sposób nieliniowy: niektóre wartości cech działają jak przeszkody, po których reakcja nagle wzrasta lub maleje, wzmacnia się lub słabnie, a nawet odwraca. Jako przykład rozważmy, jak ludzie rosną od dzieciństwa. Jeśli obserwuje się je w określonym przedziale wiekowym, związek między wiekiem a wzrostem jest w pewien sposób liniowy: dziecko rośnie wraz z wiekiem. Jednak niektóre dzieci rosną bardziej (wzrost ogólny), a inne szybciej (wzrost w określonym czasie). Ta obserwacja jest aktualna, gdy oczekujesz, że model liniowy znajdzie średnią odpowiedź.

Jednak po pewnym wieku dzieci przestają rosnąć, a wzrost pozostaje stały przez długi okres życia, powoli spadając w starszym wieku. Oczywiście regresja liniowa nie może uchwycić takiej nieliniowej zależności. (W końcu możesz przedstawić to jako rodzaj paraboli). Ponieważ relacja między zmienną przewidywaną a każdą zmienną predykcyjną jest oparta na pojedynczym współczynniku, nie masz sposobu na przedstawienie złożonych relacji, takich jak parabola (unikatowa wartość x maksymalizująca lub minimalizująca odpowiedź), wykładniczy wzrost lub bardziej złożoną nieliniową krzywą, chyba że wzbogacisz tę cechę. Najłatwiejszym sposobem modelowania złożonych relacji jest zastosowanie matematycznych przekształceń predyktorów przy użyciu rozwinięcia wielomianowego. Rozszerzenie wielomianowe, przy określonym stopniu d, tworzy potęgi każdej cechy aż do d-potęgi i d-kombinacji wszystkich terminów. Na przykład, jeśli zaczniesz od prostego modelu liniowego, takiego jak:

y = b1x1 + b2x2 + a

a następnie użyj rozwinięcia wielomianowego drugiego stopnia, model ten staje się

y = b1x1 + b2x2 + a + b3x1**2+b4x2**2+b5x1x2

Wprowadzasz dodatek do oryginalnego sformułowania (rozwinięcie) za pomocą potęg i kombinacji istniejących predyktorów. Wraz ze wzrostem stopnia rozwinięcia wielomianu rośnie liczba wyrazów pochodnych. Używając rozwinięcia wielomianowego, zaczynasz umieszczać zmienne względem siebie. Właśnie to robią sieci neuronowe i głębokie uczenie na inną skalę; wiążą każdą zmienną ze sobą. Poniższy przykład Pythona używa zestawu danych Boston do sprawdzenia skuteczności techniki. Jeśli się powiedzie, rozwinięcie wielomianowe wykryje nieliniowe relacje w danych, które wymagają krzywej, a nie linii, aby poprawnie przewidzieć i przezwyciężyć wszelkie trudności w przewidywaniu kosztem zwiększonej liczby predyktorów.

```
from sklearn.preprocessing import PolynomialFeatures

from sklearn.model_selection import train_test_split

from sklearn.metrics import r2_score

pf = PolynomialFeatures(degree=2)

poly_X = pf.fit_transform(X)

X_train, X_test, y_train, y_test = (

train_test_split(poly_X,

y, test_size=0.33, random_state=42))

from sklearn.linear_model import Ridge

reg_regression = Ridge(alpha=0.1, normalize=True)

reg_regression.fit(X_train,y_train)

print ('R2: %0.3f'

% r2_score(y_test,reg_regression.predict(X_test)))

R2: 0.819
```

Ponieważ skale elementów są powiększane przez rozwinięcie potęgi, dobrą praktyką jest standaryzacja danych po rozwinięciu wielomianu. Rozszerzenie wielomianowe nie zawsze zapewnia korzyści pokazane w poprzednim przykładzie. Rozszerzając liczbę funkcji, zmniejszasz stronniczość prognoz kosztem potencjalnego nadmiernego dopasowania.

Przełączanie na prawdopodobieństwo

Do tej pory rozważano tylko modele regresji, które wyrażają wartości liczbowe jako dane wyjściowe z uczenia się danych. Większość problemów wymaga jednak również klasyfikacji. W poniższych sekcjach omówiono, w jaki sposób można odnieść się do wyników zarówno liczbowych, jak i klasyfikacji.

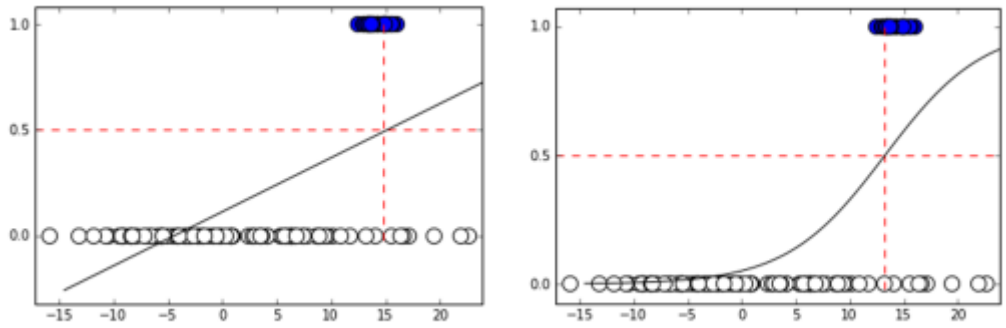
Określanie odpowiedzi binarnej

Rozwiązaniem problemu z odpowiedzią binarną (model musi wybierać spośród dwóch możliwych klas) byłoby zakodowanie wektora odpowiedzi jako ciągu jedynek i zer (lub wartości dodatnich i ujemnych). Poniższy kod Pythona udowadnia zarówno wykonalność, jak i ograniczenia używania odpowiedzi binarnej:

```
import numpy as np

a = np.array([0, 0, 0, 0, 1, 1, 1, 1])

b = np.array([1, 2, 3, 4, 5, 6, 7, 8]).reshape(8,1)

from sklearn.linear_model import LinearRegression

regression = LinearRegression()

regression.fit(b,a)

print (regression.predict(b)>0.5)

[False False False False True True True True]
```

W statystyce regresja liniowa nie może rozwiązać problemów z klasyfikacją, ponieważ spowodowałoby to powstanie serii naruszonych założeń statystycznych. Tak więc w przypadku statystyki wykorzystanie modeli regresji do celów klasyfikacji jest głównie problemem teoretycznym, a nie praktycznym. W głębokim uczeniu problem z regresją liniową polega na tym, że służy ona jako funkcja liniowa, która stara się zminimalizować błędy przewidywania; dlatego, w zależności od nachylenia obliczonej linii, może nie być w stanie rozwiązać problemu z danymi.

Gdy regresja liniowa otrzymuje zadanie przewidywania dwóch wartości, takich jak 0 i +1, które reprezentują dwie klasy, próbuje obliczyć linię, która zapewnia wyniki zbliżone do wartości docelowych. W niektórych przypadkach, mimo że wyniki są precyzyjne, dane wyjściowe są zbyt daleko od wartości docelowych, co wymusza dostosowanie linii regresji w celu zminimalizowania sumowanych błędów. Zmiana skutkuje mniejszą liczbą sumowanych błędów odchyleń, ale większą liczbą błędnie sklasyfikowanych przypadków. Regresja liniowa nie daje akceptowalnych wyników, gdy priorytetem jest dokładność klasyfikacji, jak pokazano na rysunku po lewej stronie.



Dlatego nie sprawdzi się w wielu zadaniach klasyfikacyjnych. Regresja liniowa działa najlepiej na kontinuum oszacowań liczbowych. Jednak w przypadku zadań klasyfikacyjnych potrzebna jest bardziej odpowiednia miara, taka jak prawdopodobieństwo posiadania klasy.

Przekształcanie oszacowań liczbowych w prawdopodobieństwa

Dzięki następującemu wzorowi możesz przekształcić oszacowania liczbowe regresji liniowej na prawdopodobieństwa, które lepiej opisują, jak klasa pasuje do obserwacji:

$$p(y=1)=\frac{\exp(r)}{(1+\exp(r))}$$

W tym wzorze celem jest prawdopodobieństwo, że odpowiedź y będzie odpowiadać klasie 1. Litera r jest wynikiem regresji, sumą zmiennych ważonych ich współczynnikami. Funkcja wykładnicza exp(r) odpowiada liczbie Eulera e podniesionej do potęgi r. Regresja liniowa wykorzystująca tę formułę przekształcenia (zwaną również funkcją łączenia) do zmiany jej wyników na prawdopodobieństwa jest regresją logistyczną. Regresja logistyczna (pokazana po prawej stronie na rysunku powyżej) jest taka sama jak regresja liniowa, z tym wyjątkiem, że dane y zawierają liczby całkowite wskazujące klasę względem obserwacji. Tak więc, używając zestawu danych Boston z modułu zestawów danych Scikit-learn, możesz spróbować zgadnąć, co sprawia, że domy w okolicy są zbyt drogie (wartości mediany >= 40):

```
from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split

binary_y = np.array(y >= 40).astype(int)

X_train, X_test, y_train, y_test = train_test_split(X,

binary_y, test_size=0.33, random_state=5)

logistic = LogisticRegression()

logistic.fit(X_train,y_train)

from sklearn.metrics import accuracy_score

print('In-sample accuracy: %0.3f' %

accuracy_score(y_train, logistic.predict(X_train)))

print('Out-of-sample accuracy: %0.3f' %
```

accuracy_score(y_test, logistic.predict(X_test)))

In-sample accuracy: 0.973

Out-of-sample accuracy: 0.958

Przykład dzieli dane na zestawy uczące i testowe, umożliwiając sprawdzenie skuteczności modelu regresji logistycznej na danych, których model nie używał do uczenia się. Otrzymane współczynniki informują o prawdopodobieństwie znalezienia się określonej klasy w klasie docelowej (która jest dowolną klasą zakodowaną przy użyciu wartości 1). Jeśli współczynnik zwiększa prawdopodobieństwo, będzie miał dodatni współczynnik; w przeciwnym razie współczynnik jest ujemny.

for var,coef in zip(boston.feature_names,

logistic.coef_[0]):

print ("%7s : %7.3f" %(var, coef))

CRIM : -0.006

ZN : 0.197

INDUS : 0.580

CHAS : -0.023

NOX : -0.236

RM : 1.426

AGE : -0.048

DIS : -0.365

RAD : 0.645

TAX : -0.220

PTRATIO : -0.554

B : 0.049

LSTAT : -0.803

Czytając wyniki na ekranie, widać, że w Bostonie przestępczość (CRIM) ma pewien wpływ na ceny. Jednak poziom ubóstwa (LSTAT), odległość od pracy (DIS) i zanieczyszczenie (NOX) mają znacznie większe skutki. Co więcej, w przeciwieństwie do regresji liniowej, regresja logistyczna nie wyświetla po prostu klasy wynikowej (w tym przypadku 1 lub 0), ale także szacuje prawdopodobieństwo, że obserwacja należy do jednej z dwóch klas:

print('\nclasses:',logistic.classes_)

print('\nProbs:\n',logistic.predict_proba(X_test)[:3,:])

classes: [0 1]

Probs:

[[0.39022779 0.60977221]

[0.93856655 0.06143345]

[0.98425623 0.01574377]]

W tej małej próbie tylko pierwszy przypadek ma 61 procent prawdopodobieństwa, że będzie kosztownym obszarem mieszkaniowym. Wykonując prognozy przy użyciu tego podejścia, znasz również prawdopodobieństwo, że Twoja prognoza jest dokładna i działasz odpowiednio, wybierając tylko prognozy o odpowiednim poziomie dokładności. (Na przykład możesz wybrać tylko prognozy, które przekraczają 80-procentowe prawdopodobieństwo.)

Zgadywanie właściwych funkcji

Posiadanie wielu funkcji do pracy może wydawać się odpowiedzią na potrzebę głębokiego uczenia się, aby w pełni zrozumieć problem. Jednak samo posiadanie funkcji niczego nie rozwiązuje; potrzebujesz odpowiednich funkcji do rozwiązywania problemów. W poniższych sekcjach omówiono, jak wybrać odpowiednie funkcje podczas wykonywania zadań uczenia głębokiego.

Definiowanie wyniku niezgodnych funkcji

O ile nie używasz walidacji krzyżowej, miary błędów, takie jak R^2, mogą być mylące, ponieważ liczba funkcji może łatwo ją zawyżyć, nawet jeśli funkcja nie zawiera odpowiednich informacji. Poniższy przykład pokazuje, co dzieje się z R2 po dodaniu tylko losowych funkcji:

```
from sklearn.model_selection import train_test_split

from sklearn.metrics import r2_score

X_train, X_test, y_train, y_test = train_test_split(X,

y, test_size=0.33, random_state=42)

check = [2**i for i in range(8)]

for i in range(2**7+1):

X_train = np.column_stack((X_train,np.random.random(

X_train.shape[0])))

X_test = np.column_stack((X_test,np.random.random(

X_test.shape[0])))

regression.fit(X_train, y_train)

if i in check:

print ("Random features: %i -> R2: %0.3f" % (i,

r2_score(y_train,regression.predict(X_train))))
```

```
Random features: 1 -> R2: 0.739

Random features: 2 -> R2: 0.740

Random features: 4 -> R2: 0.740

Random features: 8 -> R2: 0.743
```

Random features: 16 -> R2: 0.746

Random features: 32 -> R2: 0.762

Random features: 64 -> R2: 0.797

Random features: 128 -> R2: 0.859

To, co wydaje się zwiększoną zdolnością przewidywania, jest w rzeczywistości tylko iluzją. Możesz ujawnić, co się stało, sprawdzając zestaw testowy i odkrywając, że wydajność modelu spadła:

```
regression.fit(X_train, y_train)

print ('R2 %0.3f'

% r2_score(y_test,regression.predict(X_test)))

# Please notice that the R2 result may change from run to

# run due to the random nature of the experiment

R2 0.474
```

Rozwiązywanie problemu nadmiernego dopasowania za pomocą selekcji i regularyzacji

Regularyzacja jest skutecznym, szybkim i łatwym rozwiązaniem do wdrożenia, gdy masz wiele funkcji i chcesz zmniejszyć wariancję oszacowań z powodu współliniowości między predyktorami. Może to również pomóc, jeśli masz wartości odstające i szum w swoich danych. Regularyzacja działa poprzez dodanie kary do funkcji kosztu. Kara jest sumą współczynników. Jeśli współczynniki są podniesione do kwadratu (tak, że wartości dodatnie i ujemne nie mogą się nawzajem znosić), jest to regularyzacja L_2 (zwana również grzbietem). Kiedy używasz wartości bezwzględnej współczynnika, jest to regularyzacja L_1 (zwana również Lasso). Jednak regularyzacja nie zawsze działa idealnie. Regularyzacja L_2 zachowuje wszystkie cechy w modelu i równoważy udział każdej z nich. W rozwiązaniu L_2, jeśli dwie zmienne są dobrze skorelowane, każda z nich w równym stopniu przyczynia się do rozwiązania dla części, podczas gdy bez regularyzacji ich wspólny wkład byłby nierówno rozłożony. Alternatywnie, L_1 wydobywa z modelu wysoce skorelowane cechy, ustawiając ich współczynnik na zero, proponując w ten sposób rzeczywisty wybór między cechami. W rzeczywistości ustawienie współczynnika na zero jest jak wykluczenie cechy z modelu. Gdy współliniowość jest wysoka, wybór predyktora do ustawienia na zero staje się nieco losowy i, w zależności od próbki, można uzyskać różne rozwiązania charakteryzujące się różnie wykluczonymi cechami. Taka niestabilność rozwiązania może być uciążliwa, czyniąc rozwiązanie L_1 mniej niż idealnym. Naukowcy znaleźli rozwiązanie, tworząc różne rozwiązania oparte na regularyzacji L_1 a następnie przyjrzenie się zachowaniu współczynników w różnych rozwiązaniach. W tym przypadku algorytm wybiera tylko stabilne współczynniki (te, które rzadko są ustawione na zero). Poniższy przykład modyfikuje przykład rozwinięć wielomianowych przy użyciu regularyzacji L_2 (regresji Ridge'a) i zmniejsza wpływ nadmiarowych współczynników utworzonych przez procedurę rozwinięcia:

```
from sklearn.preprocessing import PolynomialFeatures

from sklearn.model_selection import train_test_split

pf = PolynomialFeatures(degree=2)

poly_X = pf.fit_transform(X)

X_train, X_test, y_train, y_test =
```

```
train_test_split(poly_X,

y, test_size=0.33, random_state=42)

from sklearn.linear_model import Ridge

reg_regression = Ridge(alpha=0.1, normalize=True)

reg_regression.fit(X_train,y_train)

print ('R2: %0.3f'

% r2_score(y_test,reg_regression.predict(X_test)))

R2: 0.819
```

Uczenie się jednego przykładu na raz

Znalezienie właściwych współczynników dla modelu liniowego to tylko kwestia czasu i pamięci. Czasami jednak system nie ma wystarczającej ilości pamięci do przechowywania ogromnego zestawu danych. W takim przypadku musisz uciec się do innych środków, takich jak uczenie się na jednym przykładzie na raz, zamiast ładowania ich wszystkich do pamięci. Poniższe sekcje przedstawiają podejście do nauki oparte na jednym przykładzie na raz.

Korzystanie z gradientu

Opadanie gradientu znajduje właściwy sposób na zminimalizowanie funkcji kosztu po jednej iteracji na raz. Po każdym kroku sprawdza wszystkie zsumowane błędy modelu i aktualizuje współczynniki, aby błąd był jeszcze mniejszy podczas następnej iteracji danych. Skuteczność tego podejścia wynika z rozważenia wszystkich przykładów w próbie. Wadą tego podejścia jest konieczność załadowania wszystkich danych do pamięci. Niestety nie zawsze możesz przechowywać wszystkie dane w pamięci, ponieważ niektóre zestawy danych są ogromne. Ponadto uczenie się z wykorzystaniem prostych uczniów wymaga dużej ilości danych do budowy efektywnych modeli (więcej danych pomaga w prawidłowym rozróżnieniu współliniowości). Pobieranie i przechowywanie porcji danych na dysku twardym jest zawsze możliwe, ale nie jest to wykonalne ze względu na konieczność wykonywania mnożenia macierzy, co wymaga wymiany danych z dysku w celu wybrania wierszy i kolumn. Naukowcy, którzy pracowali nad problemem, znaleźli skuteczne rozwiązanie. Zamiast uczyć się na podstawie wszystkich danych po obejrzeniu ich wszystkich (co nazywa się iteracją), algorytm uczy się z jednego przykładu na raz, wybranego z pamięci przy użyciu dostępu sekwencyjnego, a następnie uczy się na podstawie następnego przykładu. Gdy algorytm nauczy się wszystkich przykładów, zaczyna od początku, chyba że spełni jakieś kryterium zatrzymania (takie jak wykonanie określonej liczby iteracji).

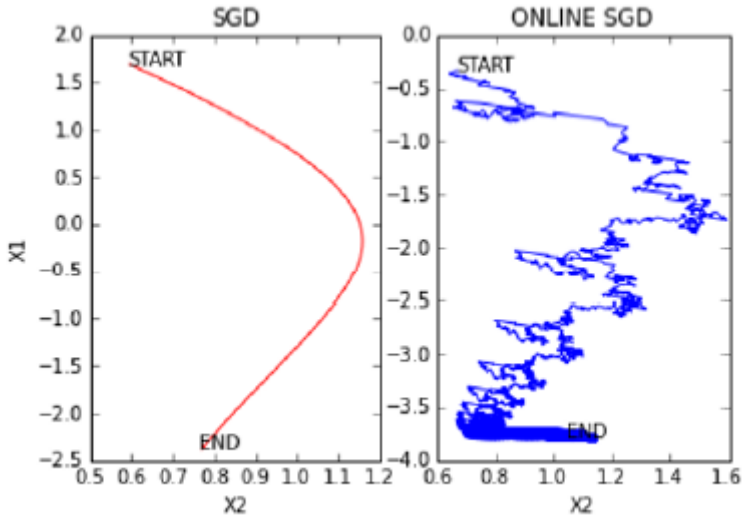
Zrozumienie, czym różni się SGD

Stochastic gradient descent (SGD) to niewielka odmiana algorytmu gradientu. Zawiera procedurę aktualizacji szacowania współczynników beta. Modele liniowe doskonale radzą sobie z tym podejściem. W SGD receptura pozostaje taka sama jak w standardowej wersji gradientu (zwanej wersją wsadową, w przeciwieństwie do wersji online), z wyjątkiem aktualizacji. W SGD aktualizacja jest wykonywana pojedynczo, co pozwala algorytmowi na pozostawienie podstawowych danych w pamięci i umieszczenie w pamięci tylko jednej obserwacji potrzebnej do zmiany wektora współczynnika:

$w_j = w_j – \alpha(wx - y)x_j$

Podobnie jak w przypadku algorytmu gradientu, algorytm aktualizuje współczynnik w cechy j, odejmując różnicę między przewidywaniem a rzeczywistą odpowiedzią. Następnie mnoży różnicę przez

wartość cechy j oraz przez współczynnik uczenia alfa (który może zmniejszyć lub zwiększyć wpływ aktualizacji na współczynnik). SGD oferuje inne subtelne różnice. Najważniejszą różnicą jest stochastyczny termin w nazwie tego algorytmu uczenia się online. W rzeczywistości SGD oczekuje losowo jednego przykładu z dostępnych przykładów (dobór losowy). Problem z nauką online polega na tym, że przykładowe porządkowanie zmienia sposób, w jaki algorytm odgaduje współczynniki beta. W przypadku częściowej optymalizacji jeden przykład może zmienić sposób, w jaki algorytm osiąga optymalną wartość, tworząc inny zestaw współczynników, niż miałoby to miejsce bez tego przykładu. Jako praktyczny przykład SGD może nauczyć się kolejności, w jakiej widzi przykłady. Jeśli algorytm dokonuje dowolnego porządku (historycznego, alfabetycznego lub, co gorsza, związanego ze zmienną odpowiedzi), to niezmiennie się tego uczy. Tylko losowe próbkowanie pozwala uzyskać wiarygodny model online, który działa skutecznie na niewidocznych danych. Podczas przesyłania strumieniowego danych musisz losowo zmienić kolejność danych (tasowanie danych). Algorytm SGD, w przeciwieństwie do uczenia wsadowego, wymaga znacznie większej liczby iteracji, aby uzyskać właściwy kierunek globalny, pomimo przeciwnych wskazań, które pochodzą z pojedynczych przykładów. W rzeczywistości algorytm aktualizuje się po każdym nowym przykładzie, a wynikająca z niego droga do optymalnego zestawu parametrów jest bardziej nieobliczalna w porównaniu z optymalizacją przeprowadzaną na partii, która od razu zmierza we właściwym kierunku, ponieważ pochodzi z danych jako całości , jak pokazano na rysunku



W tym przypadku szybkość uczenia się ma jeszcze większe znaczenie, ponieważ dyktuje, w jaki sposób procedura optymalizacji SGD może oprzeć się złym przykładom. W rzeczywistości, jeśli wskaźnik uczenia się jest wysoki, odstający przykład może całkowicie wykoleić algorytm, uniemożliwiając mu osiągnięcie dobrego wyniku. Z drugiej strony wysokie wskaźniki uczenia się pomagają utrzymać algorytm uczenia się na przykładach. Dobrą strategią jest zastosowanie elastycznego tempa uczenia się, to znaczy rozpoczynanie od elastycznego tempa uczenia się i usztywnianie go w miarę wzrostu liczby przykładów, które widział. Zarówno klasyfikacja SGD, jak i implementacje regresji w narzędziu Scikit-learn zawierają różne funkcje straty, które można zastosować do optymalizacji stochastycznego spadku gradientu. Tylko dwie z tych funkcji odnoszą się do metod omówionych w tej części:

- loss='squared_loss': zwykłe najmniejszych kwadratów (OLS) dla regresji liniowej
- loss='log': Klasyczna regresja logistyczna

Aby zademonstrować skuteczność uczenia się poza rdzeniem, poniższy przykład konfiguruje krótki eksperyment w Pythonie z użyciem regresji i squared_loss jako funkcji kosztu. Opiera się na zestawie

danych z Bostonu po przetasowaniu go i podzieleniu na zestawy treningowe i testowe. Przykład pokazuje, jak zmieniają się współczynniki beta, gdy algorytm widzi więcej przykładów. W przykładzie te same dane są przekazywane wielokrotnie, aby wzmocnić uczenie się wzorców danych. Użycie zestawu testowego gwarantuje rzetelną ocenę, zapewniając pomiary zdolności algorytmu do uogólniania danych poza próbką. Wynik pokazuje, ile czasu zajmie wzrost R^2 i ustabilizowanie wartości współczynników:

```
from sklearn.model_selection import train_test_split

from sklearn.linear_model import SGDRegressor

X_train, X_test, y_train, y_test = train_test_split(X,

y, test_size=0.33, random_state=42)

SGD = SGDRegressor(penalty=None,

learning_rate='invscaling',

eta0=0.01, power_t=0.25,

max_iter=5, tol=None)

power = 17

check = [2**i for i in range(power+1)]

for i in range(400):

for j in range(X_train.shape[0]):

SGD.partial_fit(X_train[j,:].reshape(1,13),

y_train[j].reshape(1,))

count = (j+1) + X_train.shape[0] * i

if count in check:

R2 = r2_score(y_test,SGD.predict(X_test))

print ('Example %6i R2 %0.3f coef: %s' %

(count, R2, ' '.join(map(

lambda x:'%0.3f' %x, SGD.coef_))))
```

```
Example 131072 R2 0.724 coef: -1.098 0.891 0.374 0.849

-1.905 2.752 -0.371 -3.005 2.026 -1.396 -2.011

1.102 -3.956
```

Bez względu na ilość danych, zawsze możesz dopasować prosty, ale skuteczny model regresji liniowej, korzystając z możliwości uczenia się online SGD

Przedstawiamy sieci neuronowe

Być może słyszałeś termin sieć neuronowa w odniesieniu do sztucznej inteligencji. Pierwszą rzeczą, którą musisz wiedzieć, jest to, że właściwym terminem jest sztuczna sieć neuronowa (SSN), ponieważ nikt nie odkrył żadnej metody odtwarzania prawdziwego mózgu, skąd pochodzi koncepcja sieci neuronowej. Część 2 opisuje różne podejścia do głębokiego uczenia się, z których jednymi są SSN. Tu termin ten jest skrócony, ponieważ wszyscy inni go używają, ale musisz wiedzieć, że ANN jest właściwie właściwym terminem i że jest dziełem plemienia koneksjonistów. Przynajmniej prawdziwy mózg – może zacząć doceniać perceptron, czyli najprostszy rodzaj sieci neuronowej. Perceptron jest w centrum wielu obrazów sieci neuronowych, które widzisz w Internecie, ale nie wszystkie sieci neuronowe naśladują perceptron. Sieć neuronowa może pracować ze złożonymi danymi, ponieważ umożliwia przepływ wielu danych wejściowych przez wiele warstw przetwarzania w celu uzyskania niezliczonych danych wyjściowych. (Perceptron może w rzeczywistości wybierać tylko między dwoma wyjściami.) Pomysł polega na tym, że każda ze ścieżek uruchamia się tylko wtedy, gdy faktycznie ma szansę odpowiedzieć na dowolne pytanie, które podasz swoimi danymi wejściowymi, w oparciu o wybrane algorytmy. W następnej części omówiono niektóre z tych metod radzenia sobie ze złożonymi danymi. Ponieważ sieci neuronowe mogą modelować niewiarygodnie złożone dane w sposób, który zdumiewa niektórych ludzi, można by pomyśleć, że mogą one korygować błędy w przetwarzaniu, takie jak nadmierne dopasowanie. Niestety, komputery naprawdę nie mają prawdziwych mózgów, więc nadmierne dopasowanie jest problemem, który trzeba rozwiązać. W ostatniej części tego rozdziału przyjrzymy się niektórym rozwiązaniom dotyczącym nadmiernego dopasowania i omówimy, dlaczego jest to tak duży problem.

Odkrywanie niesamowitego perceptronu

Mimo że tekst dotyczy głębokiego uczenia się, nadal musisz wiedzieć coś o poprzednich poziomach wdrażania uczenia maszynowego i sztucznej inteligencji. Perceptron jest w rzeczywistości rodzajem (implementacją) uczenia maszynowego dla większości ludzi, ale inne źródła podają, że jest to prawdziwa forma uczenia głębokiego. Możesz rozpocząć podróż w kierunku odkrycia, jak działają algorytmy uczenia maszynowego, przyglądając się modelom, które określają swoje odpowiedzi za pomocą linii i powierzchni, aby podzielić przykłady na klasy lub oszacować przewidywania wartości. Są to modele liniowe, a ten rozdział przedstawia jeden z najwcześniejszych algorytmów liniowych stosowanych w uczeniu maszynowym: perceptron. Późniejsze rozdziały pomogą Ci odkryć inne rodzaje modelowania znacznie bardziej zaawansowane niż perceptron. Jednak zanim będziesz mógł przejść do tych innych tematów, powinieneś zrozumieć ciekawą historię perceptronu.

Zrozumienie funkcjonalności perceptronu

Frank Rosenblatt z Cornell Aeronautical Laboratory opracował perceptron w 1957 pod patronatem United States Naval Research. Rosenblatt był psychologiem i pionierem w dziedzinie sztucznej inteligencji. Biegły w kognitywistyce, jego pomysł polegał na stworzeniu komputera, który mógłby uczyć się metodą prób i błędów, tak jak robi to człowiek. Pomysł został pomyślnie opracowany i na początku perceptron nie był pomyślany jako zwykłe oprogramowanie; został stworzony jako oprogramowanie działające na dedykowanym sprzęcie. Użycie tej kombinacji umożliwiło szybsze i dokładniejsze rozpoznawanie złożonych obrazów niż jakikolwiek inny komputer w tamtym czasie. Nowa technologia wzbudziła wielkie nadzieje i wywołała ogromne kontrowersje, gdy Rosenblatt stwierdził, że perceptron jest zalążkiem nowego rodzaju komputera, który będzie w stanie chodzić, mówić, widzieć, pisać, a nawet reprodukować się i być świadomym swojego istnienia. Jeśli to prawda, byłoby potężnym narzędziem i wprowadziło świat do sztucznej inteligencji. Nie trzeba dodawać, że perceptron nie spełnił oczekiwań swojego twórcy. Wkrótce wykazała ograniczoną zdolność, nawet w

swojej specjalizacji w rozpoznawaniu obrazów. Ogólne rozczarowanie zapoczątkowało pierwszą zimę AI (okres zmniejszonego finansowania i zainteresowania w większości z powodu nadmiernej hybrydy) i tymczasowego porzucenia koneksjonizmu do lat 80. XX wieku. Koneksjonizm to podejście do uczenia maszynowego oparte na neuronauce, a także na przykładzie biologicznie połączonych sieci. Możesz prześledzić korzeń koneksjonizmu z perceptronem. Perceptron jest algorytmem iteracyjnym, który dąży do określenia, poprzez kolejne i powtarzalne przybliżenia, najlepszego zestawu wartości dla wektora, w, który jest również nazywany wektorem współczynników. Kiedy perceptron osiągnie odpowiedni wektor współczynników, może przewidzieć, czy przykład jest częścią klasy. Na przykład jednym z zadań, które początkowo wykonywał perceptron, było ustalenie, czy obraz otrzymany z czujników wizualnych przypominał łódź (przykład rozpoznawania obrazu wymagany przez Biuro Badań Marynarki Wojennej Stanów Zjednoczonych, sponsor badań nad perceptronem). Kiedy perceptron zobaczył obraz jako część klasy łodzi, oznaczało to, że sklasyfikował obraz jako łódź. Wektor może pomóc przewidzieć klasę przykładu, gdy pomnożymy go przez macierz cech X, zawierającą informacje w wartościach liczbowych wyrażonych w wartościach liczbowych względem naszego przykładu, a następnie dodamy wynik mnożenia do stałego wyrazu o nazwie stronniczość, ur. Jeśli wynik sumy jest zerowy lub dodatni, perceptron klasyfikuje przykład jako część klasy. Gdy suma jest ujemna, przykład nie jest częścią klasy. Oto wzór na perceptron, w którym funkcja znaku wyprowadza 1 (gdy przykład jest częścią klasy), gdy wartość w nawiasie jest równa lub wyższa od zera; w przeciwnym razie zwraca 0:

y = znak(Xw + b)

Zauważ, że ten algorytm zawiera wszystkie elementy, które charakteryzują głęboką sieć neuronową, co oznacza, że wszystkie elementy budulcowe umożliwiające tę technologię były obecne od samego początku:

- Numeryczne przetwarzanie danych wejściowych: X zawiera liczby i żadne wartości symboliczne nie są używane jako dane wejściowe, dopóki nie zostaną przetworzone jako liczba. Na przykład nie możesz wprowadzać informacji symbolicznych, takich jak czerwony, zielony lub niebieski, dopóki nie przekonwertujesz tych wartości kolorów na liczby.
- Wagi i obciążenie: Perceptron przekształca X, mnożąc przez wagi i dodając obciążenie.
- Podsumowanie wyników: Wykorzystuje mnożenie macierzy podczas mnożenia X przez wektor w.
- Funkcja aktywacji: Perceptron aktywuje wynik wejścia w skład klasy, gdy suma przekracza próg - w tym przypadku, gdy wynikowa suma wynosi zero lub więcej.
- Iteracyjne uczenie się najlepszego zbioru wartości wektora w: Rozwiązanie opiera się na kolejnych przybliżeniach opartych na porównaniu między wyjściem perceptronu a oczekiwanym wynikiem.

Dotknięcie granicy nierozłączności

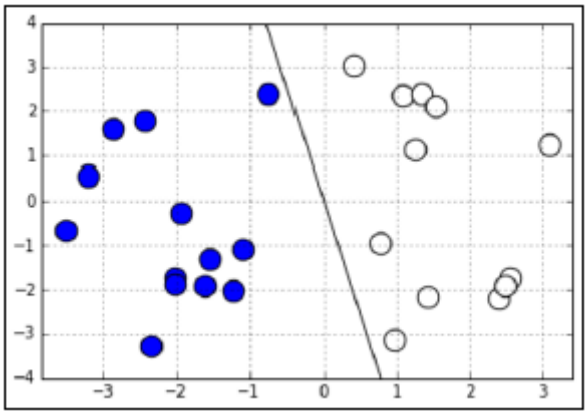
Sekret obliczeń perceptronowych polega na tym, jak algorytm aktualizuje wartości wektora w. Takie aktualizacje następują losowo wybierając jeden z błędnie sklasyfikowanych przykładów. Masz błędnie sklasyfikowany przykład, gdy perceptron stwierdza, że przykład jest częścią klasy, ale nie jest, lub gdy perceptron określa, że przykład nie jest częścią klasy, ale jest. Perceptron obsługuje jeden błędnie sklasyfikowany przykład na raz (nazwij go x_t) i działa poprzez zmianę wektora w za pomocą prostego dodawania ważonego:

$w = w + \eta(x_t * y_t)$

Ta formuła nazywa się strategią aktualizacji perceptronu, a litery oznaczają różne elementy liczbowe:

- Litera w to wektory współczynników, które są aktualizowane, aby poprawnie pokazać, czy błędnie sklasyfikowany przykład t jest częścią klasy, czy nie.
- Grecka litera eta (η) to wskaźnik uczenia się. Jest to liczba zmiennoprzecinkowa z zakresu od 0 do 1. Ustawienie tej wartości w pobliżu zera może ograniczyć zdolność formuły do prawie całkowitego zaktualizowania wektora w, podczas gdy ustawienie wartości bliskiej jedności sprawia, że proces aktualizacji w pełni wpływa na wartości wektora w. Ustawienie różnych szybkości uczenia się może przyspieszyć lub spowolnić proces uczenia się. Wiele innych algorytmów korzysta z tej strategii, a niższe eta służy do poprawy procesu optymalizacji poprzez zmniejszenie liczby nagłych skoków wartości w po aktualizacji. Kompromis polega na tym, że musisz dłużej czekać, zanim uzyskasz końcowe wyniki.
- Zmienna x_t odnosi się do wektora cech liczbowych dla przykładu t.
- Zmienna y_t odnosi się do podstawowej prawdy o tym, czy przykład t jest częścią klasy, czy nie. W przypadku algorytmu perceptronu y_t jest wyrażane umownie przez +1, gdy przykład jest częścią klasy i przez –1, gdy przykład nie jest częścią klasy.

Strategia aktualizacji zapewnia intuicję o tym, co dzieje się podczas używania perceptronu do nauki klas. Jeśli wyobrazisz sobie przykłady rzutowane na płaszczyznę kartezjańską, perceptron to nic innego jak linia próbująca oddzielić klasę pozytywną od klasy negatywnej. Jak być może pamiętasz z algebry liniowej, wszystko wyrażone w postaci y = xb+a jest w rzeczywistości prostą w płaszczyźnie. Perceptron wykorzystuje wzór y = xw + b, który używa różnych liter, ale wyraża tę samą formę, linię na płaszczyźnie kartezjańskiej. Początkowo, gdy w jest ustawione na zero lub na wartości losowe, linia oddzielająca jest tylko jedną z nieskończonych możliwych linii znalezionych na płaszczyźnie, jak pokazano na rysunku 7-1. Faza aktualizacji definiuje go, zmuszając go do zbliżenia się do błędnie sklasyfikowanego punktu. Gdy algorytm przechodzi przez błędnie sklasyfikowane przykłady, stosuje szereg poprawek. W końcu, używając wielu iteracji do zdefiniowania błędów, algorytm umieszcza linię oddzielającą na dokładnej granicy między dwiema klasami.



Pomimo tego, że był tak inteligentnym algorytmem, perceptron dość szybko pokazał swoje ograniczenia. Oprócz zdolności do odgadywania dwóch klas przy użyciu wyłącznie cech ilościowych, miał ważny limit: jeśli dwie klasy nie miały granic z powodu mieszania, algorytm nie mógł znaleźć rozwiązania i aktualizował się w nieskończoność. Jeśli nie możesz podzielić dwóch klas rozłożonych na co najmniej dwóch wymiarach za pomocą dowolnej linii lub płaszczyzny, można je rozdzielić nieliniowo. Przezwyciężanie nieliniowej separacji danych jest jednym z wyzwań, jakie musi pokonać uczenie maszynowe, aby skutecznie radzić sobie ze złożonymi problemami opartymi na rzeczywistych danych, a nie tylko na danych sztucznych tworzonych do celów akademickich. Kiedy sprawa nieliniowej rozdzielności znalazła się pod lupą i praktycy zaczęli tracić zainteresowanie perceptronem, eksperci szybko wysunęli teorię, że mogą rozwiązać problem, tworząc nową przestrzeń cech, w której wcześniej

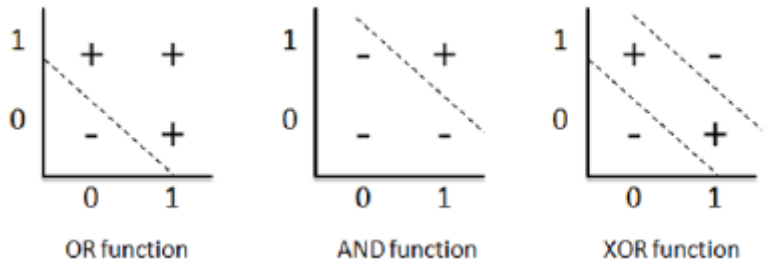
nierozdzielne klasy są dostrajane, aby stać się rozdzielnymi. W ten sposób perceptron byłby tak samo dobry jak wcześniej. Niestety tworzenie nowych przestrzeni fabularnych jest wyzwaniem, ponieważ wymaga mocy obliczeniowej, która jest dziś tylko częściowo dostępna dla społeczeństwa. Tworzenie nowej przestrzeni funkcji to zaawansowany temat omówiony w dalszej części książki przy badaniu strategii uczenia się algorytmów, takich jak sieci neuronowe i maszyny wektorów nośnych. W ostatnich latach algorytm odrodził się dzięki big data: perceptron w rzeczywistości nie musi pracować ze wszystkimi danymi w pamięci, ale radzi sobie dobrze na pojedynczych przykładach (aktualizacja wektora współczynników tylko wtedy, gdy błędnie sklasyfikowany przypadek sprawia, że jest to konieczne). Jest to zatem idealny algorytm do uczenia się online, na przykład uczenia się na podstawie dużych zbiorów danych na raz.

Uderzanie w złożoność za pomocą sieci neuronowych

Poprzednia część pomogła Ci odkryć sieć neuronową z perspektywy perceptronu. Oczywiście sieci neuronowe to coś więcej niż ten prosty początek. Wydajność i inne problemy, które nękają perceptron, są przynajmniej częściowo rozwiązywane w nowszych algorytmach. Poniższe sekcje pomagają zrozumieć sieci neuronowe w postaci, w jakiej istnieją.

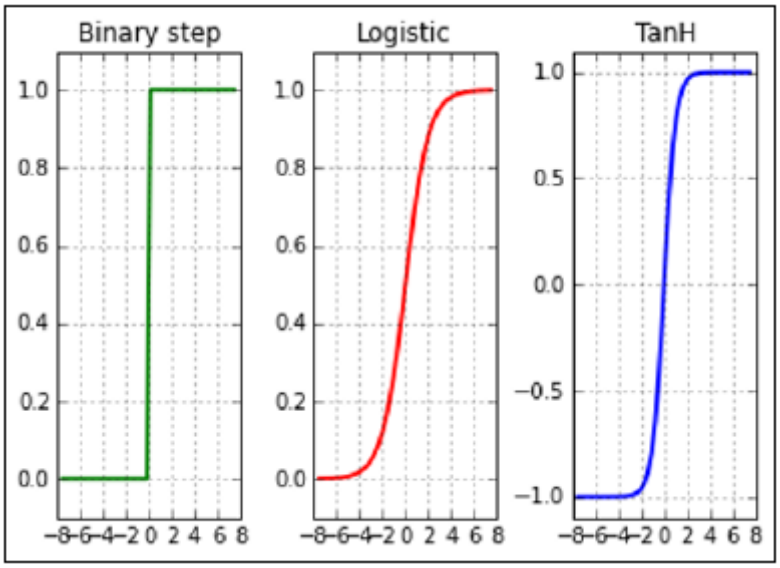
Biorąc pod uwagę neuron

Podstawowym składnikiem sieci neuronowej jest neuron (nazywany również jednostką). Wiele neuronów ułożonych w połączoną strukturę tworzy sieć neuronową, w której każdy neuron łączy się z wejściami i wyjściami innych neuronów. W ten sposób neuron może wprowadzać cechy z przykładów lub wyników innych neuronów, w zależności od jego lokalizacji w sieci neuronowej. Kiedy psycholog Rosenblatt wymyślił perceptron, myślał o nim jako o uproszczonej matematycznej wersji neuronu mózgowego. Perceptron przyjmuje wartości jako dane wejściowe z pobliskiego środowiska (zestawu danych), waży je (tak jak robią to komórki mózgowe, w oparciu o siłę połączeń przychodzących), sumuje wszystkie wartości ważone i aktywuje się, gdy suma przekracza próg. Ten próg generuje wartość 1; w przeciwnym razie jego przewidywanie wynosi 0. Niestety, perceptron nie może się uczyć, gdy klasy, które próbuje przetworzyć, nie są liniowo oddzielone. Jednak naukowcy odkryli, że chociaż pojedynczy perceptron nie mógł nauczyć się logicznej operacji XOR pokazanej na rysunku 7-2 (wyłącznej lub, co jest prawdziwe tylko wtedy, gdy dane wejściowe są różne), dwa perceptrony współpracujące ze sobą mogą.



Neurony w sieci neuronowej są dalszą ewolucją perceptronu: przyjmują wiele ważonych wartości jako dane wejściowe, sumują je i dostarczają sumę jako wynik, tak jak robi to perceptron. Zapewniają jednak również bardziej wyrafinowaną transformację sumowania, czego perceptron nie może zrobić. Obserwując przyrodę, naukowcy zauważyli, że neurony odbierają sygnały, ale nie zawsze same wysyłają sygnał. Zależy to od ilości odbieranego sygnału. Kiedy neuron otrzymuje wystarczającą ilość bodźców, wysyła odpowiedź; w przeciwnym razie milczy. W podobny sposób neurony algorytmiczne po otrzymaniu wartości ważonych sumują je i wykorzystują funkcję aktywacji do oceny wyniku, który przekształca go w sposób nieliniowy. Na przykład funkcja aktywacji może zwolnić wartość zerową,

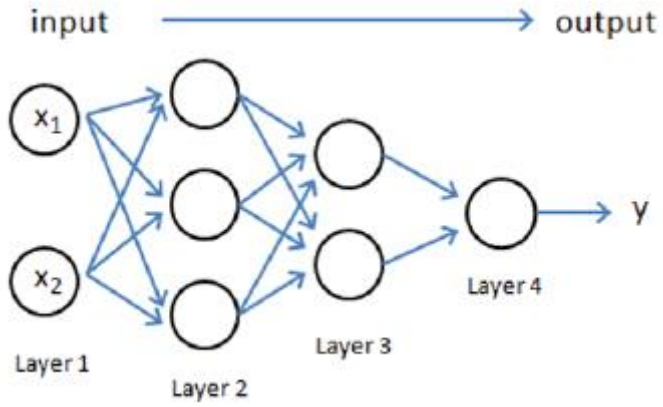
chyba że wejście osiągnie określony próg, lub może tłumić lub zwiększać wartość przez nieliniowe przeskalowanie jej, przesyłając w ten sposób przeskalowany sygnał. Sieć neuronowa ma różne funkcje aktywacji, jak pokazano na rysunku. Funkcja liniowa (oznaczona jako krok binarny) nie stosuje żadnej transformacji i jest rzadko używana, ponieważ redukuje sieć neuronową do regresji z przekształceniami wielomianowymi. Sieci neuronowe powszechnie wykorzystują funkcje aktywacji sigmoidu (oznaczonego jako Logistic) lub stycznego hiperbolicznego (oznaczonego TanH) lub ReLU (który jest obecnie zdecydowanie bardziej powszechny).



Rysunek pokazuje, w jaki sposób dane wejściowe (wyrażone na osi poziomej) mogą przekształcić dane wyjściowe w coś innego (wyrażone na osi pionowej). Przykłady pokazują krok binarny, logistykę (zwaną również sigmoidą) i styczną funkcję aktywacji hiperbolicznej (często określaną jako tanh). Więcej o funkcjach aktywacji dowiesz się w dalszej części rozdziału, ale póki co pamiętaj, że funkcje aktywacji działają dobrze w pewnych zakresach wartości x. Z tego powodu należy zawsze przeskalowywać dane wejściowe do sieci neuronowej za pomocą standaryzacji statystycznej (zerowa średnia i wariancja jednostkowa) lub normalizować dane wejściowe w zakresie od 0 do 1 lub od –1 do 1. Funkcje aktywacji sprawiają, że sieć neuronowa działa w klasyfikacji lub regresji; jednak początkowy wybór aktywacji sigmoid lub tanh dla większości sieci stanowi krytyczny limit w przypadku korzystania z bardziej złożonych sieci, ponieważ obie aktywacje działają optymalnie dla bardzo ograniczonego zakresu wartości.

Przesyłanie danych za pomocą feed-forward

W sieci neuronowej należy wziąć pod uwagę architekturę, czyli sposób rozmieszczenia elementów sieci neuronowej. W przeciwieństwie do innych algorytmów, które mają stały potok, który określa, w jaki sposób algorytmy otrzymują i przetwarzają dane, sieci neuronowe wymagają decyzji o przepływie informacji poprzez ustalenie liczby jednostek (neuronów) i ich rozkładu w warstwach, jak pokazano na rysunku.



Rysunek przedstawia prostą architekturę neuronową. Zwróć uwagę, jak warstwy filtrują informacje w sposób progresywny. Jest to wejście typu feed-forward, ponieważ dane są przesyłane do sieci w jedną stronę. Połączenia łączą wyłącznie jednostki w jednej warstwie z jednostkami w kolejnej warstwie (przepływ informacji od lewej do prawej). Nie istnieją żadne połączenia między jednostkami w tej samej warstwie lub z jednostkami spoza następnej warstwy. Co więcej, informacja przesuwa się do przodu (od lewej do prawej). Przetworzone dane nigdy nie wracają do poprzednich warstw neuronowych. Korzystanie z sieci neuronowej jest jak stosowanie warstwowego systemu filtrowania wody: wlewasz wodę z góry, a woda jest filtrowana na dole. Woda nie ma możliwości powrotu; po prostu idzie do przodu i prosto w dół, a nigdy na boki. W ten sam sposób sieci neuronowe wymuszają przepływ danych przez sieć i mieszanie się ze sobą tylko zgodnie z architekturą sieci. Wykorzystując najlepszą architekturę do miksowania funkcji, sieć neuronowa tworzy nowe złożone funkcje w każdej warstwie i pomaga uzyskać lepsze predykcje. Niestety nie ma sposobu na określenie najlepszej architektury bez empirycznego wypróbowania różnych rozwiązań i sprawdzenia, czy dane wyjściowe pomagają przewidzieć wartości docelowe po przepłynięciu przez sieć. Ważną rolę odgrywają pierwsza i ostatnia warstwa. Pierwsza warstwa, zwana warstwą wejściową, pobiera cechy z każdego przykładu danych przetwarzanych przez sieć. Ostatnia warstwa, zwana warstwą wyjściową, udostępnia wyniki. Sieć neuronowa może przetwarzać tylko numeryczne, ciągłe informacje; nie można ograniczyć do pracy ze zmiennymi jakościowymi (na przykład etykiety wskazujące jakość, taka jak czerwony, niebieski lub zielony na obrazie). Zmienne jakościowe można przetwarzać, przekształcając je w ciągłą wartość liczbową, taką jak seria wartości binarnych. Kiedy sieć neuronowa przetwarza zmienną binarną, neuron traktuje zmienną jako liczbę ogólną i przekształca wartości binarne w inne wartości, nawet ujemne, przetwarzając je w jednostkach. Zwróć uwagę na ograniczenia związane tylko z wartościami liczbowymi, ponieważ nie można oczekiwać, że ostatnia warstwa wygeneruje nienumeryczne przewidywanie etykiety. Kiedy mamy do czynienia z problemem regresji, ostatnia warstwa to pojedyncza jednostka. Podobnie, gdy pracujesz z klasyfikacją i masz dane wyjściowe, które muszą wybierać spośród liczby n klas, powinieneś mieć n jednostek końcowych, z których każda reprezentuje wynik powiązany z prawdopodobieństwem reprezentowanej klasy. Dlatego przy klasyfikowaniu problemu wieloklasowego, takiego jak gatunek tęczówki, ostatnia warstwa ma tyle samo jednostek, co gatunek. Na przykład w archetypowym przykładzie klasyfikacji Iris, stworzonym przez słynnego statystyka Fishera, mamy trzy klasy: setosa, versicolor i virginica. W sieci neuronowej opartej na zbiorze danych tęczówki masz zatem trzy jednostki reprezentujące jeden z trzech gatunków tęczówki. Dla każdego przykładu przewidywaną klasą jest ta, która na końcu otrzyma wyższy wynik. Niektóre sieci neuronowe mają specjalne warstwy końcowe, zbiorczo nazywane softmax, które mogą dostosowywać prawdopodobieństwo każdej klasy na podstawie wartości otrzymanych z poprzedniej warstwy. W klasyfikacji ostatnia warstwa może reprezentować zarówno podział prawdopodobieństw dzięki

softmaxowi (problem wieloklasowy, w którym suma prawdopodobieństw sumuje się do 100 procent), jak i niezależne przewidywanie wyniku (ponieważ przykład może mieć więcej klas, co jest problemem wieloetykietowym, w którym zsumowane prawdopodobieństwa mogą przekraczać 100 procent). Gdy problemem klasyfikacji jest klasyfikacja binarna, wystarczy jedno wyjście. Ponadto w regresji możesz mieć wiele jednostek wyjściowych, z których każda reprezentuje inny problem regresji. (Na przykład w prognozowaniu możesz mieć różne prognozy na następny dzień, tydzień, miesiąc itd.)

Wchodząc jeszcze głębiej w króliczą norę

Sieci neuronowe mają różne warstwy, z których każda ma własne wagi. Ponieważ sieć neuronowa segreguje obliczenia według warstw, znajomość warstwy odniesienia jest ważna, ponieważ można uwzględnić pewne jednostki i połączenia. Możesz odnieść się do każdej warstwy za pomocą określonego numeru i ogólnie mówić o każdej warstwie za pomocą litery l. Każda warstwa może mieć inną liczbę jednostek, a liczba jednostek znajdujących się między dwiema warstwami dyktuje liczbę połączeń. Mnożąc liczbę jednostek w warstwie początkowej przez liczbę w kolejnej warstwie, możesz określić całkowitą liczbę połączeń między nimi: liczba połączeń(l) = jednostki(l) * jednostki(l+1).

Połączenia reprezentuje macierz wag, zwykle nazywana wielką grecką literą Theta (θ). Dla ułatwienia czytania w książce użyto dużej litery W, co jest dobrym wyborem, ponieważ jest to macierz lub tablica wielowymiarowa. W ten sposób można użyć W1 do odniesienia się do wag połączeń od warstwy 1 do warstwy 2, W2 do połączeń od warstwy 2 do warstwy 3 i tak dalej. Wagi reprezentują siłę połączenia między neuronami w sieci.

Gdy waga połączenia między dwiema warstwami jest niewielka, oznacza to, że sieć zrzuca wartości przepływające między nimi i sygnalizuje, że wybranie tej trasy prawdopodobnie nie wpłynie na ostateczną prognozę. Alternatywnie, duża dodatnia lub ujemna wartość wpływa na wartości, które otrzymuje następna warstwa, zmieniając w ten sposób pewne przewidywania. To podejście jest analogiczne do komórek mózgowych, które nie są samotne, ale łączą się z innymi komórkami. W miarę zdobywania doświadczenia połączenia między neuronami mają tendencję do osłabiania lub wzmacniania, aby aktywować lub dezaktywować określone obszary komórek sieci mózgowej, powodując inne przetwarzanie lub aktywność (reakcja na niebezpieczeństwo, na przykład, jeśli przetworzona informacja sygnalizuje sytuację zagrażającą życiu).

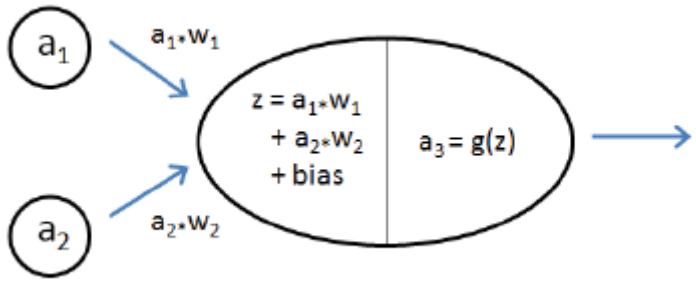
UKRYTE WARSTWY

Warstwy między wejściem a wyjściem są czasami nazywane warstwami ukrytymi, a liczenie warstw zaczyna się od pierwszej warstwy ukrytej. Jest to po prostu inna konwencja od tej zastosowanej w tej książce. Przykłady w książce zawsze zaczynają się od warstwy wejściowej, więc pierwsza ukryta warstwa to warstwa nr 2.

Teraz, gdy znasz już pewne konwencje dotyczące warstw, jednostek i połączeń, możesz rozpocząć szczegółowe badanie operacji wykonywanych przez sieci neuronowe. Na początek możesz wywołać wejścia i wyjścia na różne sposoby:

- a: Wynik przechowywany w jednostce w sieci neuronowej po przetworzeniu przez funkcję aktywacji (zwaną g). To jest końcowy wynik, który jest przesyłany dalej w sieci.
- z: Mnożenie między a i wagami z macierzy W. z reprezentuje sygnał przechodzący przez połączenia, analogicznie do wody w rurach, która płynie pod wyższym lub niższym ciśnieniem w zależności od średnicy rury. W ten sam sposób wartości odebrane z poprzedniej warstwy uzyskują wyższe lub niższe wartości ze względu na wagi połączeń użyte do ich transmisji.

Każda kolejna warstwa jednostek w sieci neuronowej progresywnie przetwarza wartości pobierane z cech (obraz taśmociągu). Gdy dane są przesyłane w sieci, docierają do każdej jednostki jako wartość wytworzona przez zsumowanie wartości obecnych w poprzedniej warstwie i ważonych połączeniami reprezentowanymi w macierzy W. Gdy dane z dodanym odchyleniem przekroczą pewien próg, aktywacja funkcja zwiększa wartość zapisaną w urządzeniu; w przeciwnym razie wygasza sygnał, redukując go. Po przetworzeniu przez funkcję aktywacji wynik jest gotowy do przekazania do połączenia połączonego z następną warstwą. Te kroki powtarzają się dla każdej warstwy, aż wartości osiągną koniec i otrzymasz wynik, jak pokazano na rysunku.



Rysunek pokazuje szczegół procesu, w którym dwie jednostki przekazują swoje wyniki do innej jednostki. To wydarzenie ma miejsce w każdej części sieci. Kiedy zrozumiesz przejście z dwóch neuronów do jednego, zrozumiesz cały proces sprzężenia do przodu, nawet jeśli zaangażowanych jest więcej warstw i neuronów. Aby uzyskać więcej wyjaśnień, oto siedem kroków używanych do stworzenia predykcji w sieci neuronowej złożonej z czterech warstw:

1. Pierwsza warstwa (zwróć uwagę na indeks górny 1 na a) ładuje wartość każdej cechy w innej jednostce:

$a^{(1)} = X$

2. Wagi połączeń mostkujących warstwę wejściową z warstwą drugą mnoży się przez wartości jednostek w warstwie pierwszej. Mnożenie macierzy waży i sumuje dane wejściowe dla drugiej warstwy razem.

$z^{(2)} = W^{(1)}a^{(1)}$

3. Algorytm dodaje stałą obciążenia do warstwy drugiej przed uruchomieniem funkcji aktywacji. Funkcja aktywacji przekształca dane wejściowe drugiej warstwy. Otrzymane wartości są gotowe do przekazania do połączeń.

$a^{(2)} = g(z^{(2)} + \text{odchylenie}^{(2)})$

4. Połączenia trzeciej warstwy ważą i sumują wyjścia warstwy drugiej.

$z^{(3)} = W^{(2)}a^{(2)}$

5. Algorytm dodaje stałą obciążenia do warstwy trzeciej przed uruchomieniem funkcji aktywacji. Funkcja aktywacji przekształca dane wejściowe warstwy trzeciej.

$a^{(3)} = g(z^{(3)} + \text{bias}^{(3)})$

6. Wyjścia warstwy trzeciej są ważone i sumowane przez połączenia do warstwy wyjściowej.

$z^{(4)} = W^{(3)})a^{(3)}$

7. Na koniec algorytm dodaje stałą obciążenia do warstwy czwartej przed uruchomieniem funkcji aktywacji. Jednostki wyjściowe otrzymują swoje dane wejściowe i przekształcają dane wejściowe za pomocą funkcji aktywacji. Po tej końcowej transformacji jednostki wyjściowe są gotowe do wydania wynikowych prognoz sieci neuronowej.

$a^{(4)} = g(z^{(4)} + bias^{(4)})$

Funkcja aktywacji pełni rolę filtra sygnału, pomagając wybrać odpowiednie sygnały i uniknąć słabych i zaszumionych (ponieważ odrzuca wartości poniżej pewnego progu). Funkcje aktywacji zapewniają również nieliniowość wyjścia, ponieważ wzmacniają lub tłumią wartości przechodzące przez nie w nieproporcjonalny sposób. Wagi połączeń umożliwiają mieszanie i komponowanie cech w nowy sposób, tworząc nowe cechy w sposób nie różniący się zbytnio od rozwinięcia wielomianowego. Aktywacja czyni nieliniową wynikową rekombinację cech przez połączenia. Oba te komponenty sieci neuronowej umożliwiają algorytmowi poznanie złożonych funkcji docelowych, które reprezentują związek między cechami wejściowymi a docelowym wynikiem.

Używanie wstecznej propagacji do dostosowania uczenia się

Z perspektywy architektonicznej sieć neuronowa świetnie sobie radzi z miksowaniem sygnałów z przykładów i przekształcaniem ich w nowe funkcje w celu uzyskania aproksymacji złożonych funkcji nieliniowych (funkcji, których nie można przedstawić jako linii prostej w przestrzeni funkcji). Aby stworzyć tę możliwość, sieci neuronowe działają jako uniwersalne aproksymatory, co oznacza, że może odgadnąć dowolną funkcję docelową. Należy jednak wziąć pod uwagę, że jednym aspektem tej funkcji jest zdolność do modelowania złożonych funkcji (zdolność do reprezentacji), a innym aspektem jest zdolność do efektywnego uczenia się na podstawie danych. Uczenie się zachodzi w mózgu z powodu tworzenia i modyfikacji synaps między neuronami w oparciu o bodźce otrzymane w wyniku doświadczeń metodą prób i błędów. Sieci neuronowe umożliwiają odtworzenie tego procesu w postaci matematycznej formuły zwanej propagacją wsteczną. Od momentu pojawienia się w latach 70. algorytm propagacji wstecznej otrzymał wiele poprawek. Każde ulepszenie procesu uczenia się sieci neuronowych skutkowało nowymi zastosowaniami i ponownym zainteresowaniem tą techniką. Ponadto obecna rewolucja w zakresie głębokiego uczenia, odrodzenie sieci neuronowych, które zarzucono na początku lat 90., wynika z kluczowych postępów w sposobie uczenia się sieci neuronowych na swoich błędach. Jak widać w innych algorytmach, funkcja kosztu aktywuje konieczność lepszego poznania pewnych przykładów (duże błędy odpowiadają wysokim kosztom). Gdy wystąpi przykład z dużym błędem, funkcja kosztu wyprowadza wysoką wartość, która jest minimalizowana przez zmianę parametrów w algorytmie. Algorytm optymalizacji określa najlepsze działanie w celu zmniejszenia wysokich wyników z funkcji kosztu. W regresji liniowej znalezienie reguły aktualizacji do zastosowania do każdego parametru (wektora współczynników beta) jest proste. Jednak w sieci neuronowej sprawy są nieco bardziej skomplikowane. Architektura jest zmienna, a współczynniki parametrów (połączenia) są ze sobą powiązane, ponieważ połączenia w warstwie zależą od tego, w jaki sposób połączenia w poprzednich warstwach rekombinowały dane wejściowe. Rozwiązaniem tego problemu jest algorytm wstecznej propagacji błędów. Propagacja wsteczna to sprytny sposób na propagację błędów z powrotem do sieci i sprawienie, aby każde połączenie odpowiednio dostosowało swoje wagi. Jeśli początkowo przekazujesz informacje przekazywane dalej do sieci, nadszedł czas, aby się cofnąć i przekazać opinię na temat tego, co poszło nie tak w fazie do przodu. Propagacja wsteczna to sposób, w jaki korekty wymagane przez algorytm optymalizacji są propagowane w sieci neuronowej. Ważne jest rozróżnienie między optymalizacją a propagacją wsteczną. W rzeczywistości wszystkie sieci neuronowe korzystają z propagacji wstecznej, ale w następnym rozdziale omówiono wiele różnych algorytmów optymalizacji. Odkrycie, jak działa propagacja wsteczna, nie jest skomplikowane, chociaż pokazanie, jak to działa za pomocą formuł i

matematyki, wymaga pochodnych i udowodnienia niektórych sformułowań, co jest dość skomplikowane i wykracza poza zakres tej książki. Aby zorientować się, jak działa propagacja wsteczna, zacznij od końca sieci, właśnie w momencie, gdy przykład został przetworzony i masz predykcję jako wyjście. W tym momencie możesz porównać go z rzeczywistym wynikiem i odejmując oba wyniki, uzyskać przesunięcie, które jest błędem. Teraz, gdy znasz niezgodność wyników w warstwie wyjściowej, możesz przejść wstecz, aby rozłożyć go na wszystkie jednostki w sieci. Funkcja kosztu sieci neuronowej do klasyfikacji opiera się na entropii krzyżowej (jak widać w regresji logistycznej):

$$Cost = y * \log(h_W(X)) + (1 - y)*\log(1 - h_W(X))$$

Jest to sformułowanie zawierające logarytmy. Odnosi się do predykcji wytwarzanej przez sieć neuronową i wyrażonej jako hW(X) (co odczytuje się jako wynik sieci podane połączenia W i X jako dane wejściowe). Aby uprościć sprawę, myśląc o kosztach, warto pomyśleć o tym jako o obliczaniu przesunięcia między oczekiwanymi wynikami a wyjściem sieci neuronowej. Pierwszy krok w przesłaniu błędu z powrotem do sieci polega na mnożeniu wstecznym. Ponieważ wartości podawane do warstwy wyjściowej składają się z wkładów wszystkich jednostek, proporcjonalnych do wagi ich połączeń, można redystrybuować błąd zgodnie z każdym wkładem. Na przykład wektor błędów warstwy n w sieci, wektor oznaczony grecką literą delta (δ), jest wynikiem następującego sformułowania:

$$\delta^{(n)} = W^{(n)T} * \delta^{(n+1)}$$

Ta formuła mówi, że począwszy od końcowej delty, możesz kontynuować redystrybucję delty, cofając się w sieci i używając wag, których użyłeś do przesunięcia wartości do przodu, aby podzielić błąd na różne jednostki. W ten sposób można rozesłać błąd terminala do każdej jednostki neuronowej i użyć go do ponownego obliczenia bardziej odpowiedniej wagi dla każdego połączenia sieciowego, aby zminimalizować błąd. Aby zaktualizować wagi W warstwy l, wystarczy zastosować następującą formułę:

$$W^{(1)} = W^{(1)} + \eta * \delta^{(1)} * g'(z^{(1)}) * a^{(1)}$$

Formuła może wydawać się zagadkowa na pierwszy rzut oka, ale jest to podsumowanie, a możesz odkryć, jak działa, patrząc na jej elementy. Najpierw spójrz na funkcję g'. Jest to pierwsza pochodna funkcji aktywacji g, obliczona przez wartości wejściowe z. W rzeczywistości jest to metoda zejścia gradientowego. Spadek gradientu określa sposób zmniejszenia miary błędu poprzez znalezienie, spośród możliwych kombinacji wartości, wag, które najbardziej redukują błąd.

Grecka litera eta (η), czasami nazywana również alfa (α) lub epsilon (ε), w zależności od podręcznika, z którym się zapoznałeś, to tempo nauki. Jak stwierdzono w innych algorytmach, zmniejsza to efekt aktualizacji sugerowanej przez pochodną gradientu opadającego. W rzeczywistości podany kierunek może być tylko częściowo poprawny lub tylko z grubsza poprawny. Wykonując wiele małych kroków w dół, algorytm może obrać bardziej precyzyjny kierunek w kierunku globalnego błędu minimalnego, który jest celem, który chcesz osiągnąć (czyli siecią neuronową wytwarzającą najmniejszy możliwy błąd predykcji). Dostępne są różne metody ustawienia właściwej wartości eta, ponieważ optymalizacja w dużej mierze zależy od tego. Jedna metoda ustawia wartość eta początkową wysoką i zmniejsza ją podczas procesu optymalizacji. Inna metoda zmiennie zwiększa lub zmniejsza eta na podstawie ulepszeń uzyskanych przez algorytm: duże ulepszenia nazywają większym eta (ponieważ zejście jest łatwe i proste); mniejsze ulepszenia wymagają mniejszego eta, aby optymalizacja działała wolniej, szukając najlepszych okazji do obniżenia. Pomyśl o tym jak na krętej ścieżce w górach: zwalniasz i starasz się nie zostać uderzonym lub zepchniętym z drogi podczas schodzenia. Większość wdrożeń oferuje automatyczne ustawienie poprawnego eta. Należy zwrócić uwagę na znaczenie tego ustawienia podczas uczenia sieci neuronowej, ponieważ jest to jeden z ważnych parametrów, które

należy dostosować, aby uzyskać lepsze prognozy, wraz z architekturą warstw. Aktualizacje wagi mogą odbywać się na różne sposoby w odniesieniu do zestawu szkoleniowego przykładów:

- Tryb online: aktualizacja wagi następuje po każdym przejściu przez sieć każdego przykładu. W ten sposób algorytm traktuje przykłady uczenia się jako strumień, z którego można się uczyć w czasie rzeczywistym. Ten tryb jest idealny, gdy musisz uczyć się out of core, czyli gdy zestaw treningowy nie mieści się w pamięci RAM. Jednak ta metoda jest wrażliwa na wartości odstające, więc musisz utrzymywać niski wskaźnik uczenia się. (W konsekwencji algorytm powoli zbliża się do rozwiązania).
- Tryb wsadowy: Aktualizacja wagi następuje po przetworzeniu wszystkich przykładów w zestawie treningowym. Ta technika sprawia, że optymalizacja jest szybka i mniej podatna na występowanie wariancji w przykładowym strumieniu. W trybie wsadowym propagacja wsteczna uwzględnia zsumowane gradienty wszystkich przykładów.
- Tryb mini-partii (lub stochastyczny): Aktualizacja wagi następuje po przetworzeniu przez sieć podpróbki losowo wybranych przykładów zestawów uczących. Podejście to łączy zalety trybu online (niskie zużycie pamięci) i trybu wsadowego (szybka konwergencja), wprowadzając element losowy (podpróbkowanie), aby uniknąć zatrzymania się gradientu w lokalnych minimach (spadek wartości, który nie jest prawdziwe minimum).

Zmaganie się z nadmiernym dopasowaniem

Biorąc pod uwagę architekturę sieci neuronowej, możesz sobie wyobrazić, jak łatwo algorytm może nauczyć się prawie wszystkiego z danych, zwłaszcza jeśli dodasz zbyt wiele warstw. W rzeczywistości algorytm działa tak dobrze, że na jego przewidywania często wpływa wysoka wariancja oszacowania zwana overfittingiem. Overfitting powoduje, że sieć neuronowa uczy się każdego szczegółu przykładów treningowych, co umożliwia ich replikację w fazie predykcji. Ale poza zestawem treningowym sieć nigdy nie przewidzi poprawnie niczego innego. W poniższych sekcjach bardziej szczegółowo omówiono niektóre problemy związane z nadmiernym dopasowaniem.

Zrozumienie problemu

Kiedy używasz sieci neuronowej do rzeczywistego problemu, stajesz się bardziej rygorystyczny i ostrożny w implementacji niż w przypadku innych algorytmów. Sieci neuronowe są słabsze i bardziej podatne na istotne błędy niż inne rozwiązania uczenia maszynowego. Starannie dzielisz swoje dane na zestawy treningowe, walidacyjne i testowe. Zanim algorytm nauczy się na podstawie danych, musisz ocenić wartość swoich parametrów:

- Architektura (liczba warstw i węzłów w nich)
- Funkcje aktywacji
- Parametr uczenia
- Liczba iteracji

W szczególności architektura oferuje ogromne możliwości tworzenia potężnych modeli predykcyjnych przy wysokim ryzyku nadmiernego dopasowania. Parametr uczenia kontroluje szybkość uczenia się sieci na podstawie danych, ale może nie wystarczyć, aby zapobiec nadmiernemu dopasowaniu danych uczących.

Otwarcie czarnej skrzynki

Masz dwa możliwe rozwiązania problemu nadmiernego dopasowania. Pierwsza to regularyzacja, jak w regresji liniowej i logistycznej. Możesz zsumować wszystkie współczynniki połączenia, do kwadratu lub w wartości bezwzględnej, aby ukarać modele ze zbyt dużą liczbą współczynników o wysokich

wartościach (uzyskiwane przez regularyzację L2) lub o wartościach różnych od zera (osiągane przez regularyzację L1). Drugie rozwiązanie jest również skuteczne, ponieważ kontroluje, kiedy dochodzi do nadmiernego dopasowania. Nazywa się to wczesnym zatrzymaniem i działa poprzez sprawdzenie funkcji kosztu w zbiorze walidacyjnym, gdy algorytm uczy się ze zbioru uczącego. (Sekcja „Uczenie się właściwego kierunku" w rozdziale 5 zawiera więcej szczegółów na temat wczesnego zatrzymywania się.) Możesz nie zdawać sobie sprawy, kiedy twój model zaczyna się przesadzać. Funkcja kosztu obliczona za pomocą zestawu treningowego poprawia się wraz z postępem optymalizacji. Jednak gdy tylko zaczniesz rejestrować szum z danych i przestaniesz uczyć się ogólnych zasad, możesz sprawdzić funkcję kosztów na danych poza próbką (próbka walidacyjna). W pewnym momencie zauważysz, że przestaje się poprawiać i zaczyna się pogarszać, co oznacza, że Twój model osiągnął limit uczenia się.

Budowanie podstawowej sieci neuronowej

Część 7 przedstawia sieci neuronowe wykorzystujące najprostszą i najbardziej podstawową sieć neuronową ze wszystkich: perceptron. Jednak sieci neuronowe występują w wielu formach, z których każda ma swoje zalety. Na szczęście wszystkie formy sieci neuronowych mają podstawową architekturę i polegają na określonych strategiach, aby osiągnąć to, co muszą. Jeśli dowiesz się, jak działa podstawowa sieć neuronowa, możesz dowiedzieć się, jak działają bardziej złożone architektury. W pierwszej części omówmy podstawy funkcjonalności sieci neuronowej – czyli to, co musisz wiedzieć, aby zrozumieć, jak sieć neuronowa wykonuje użyteczną pracę. Wyjaśnia funkcjonalność sieci neuronowej za pomocą podstawowej sieci neuronowej, którą można zbudować od podstaw za pomocą Pythona. Druga część zagłębia się w pewne różnice między sieciami neuronowymi. Na przykład w Części 7 odkryjesz, że poszczególne neurony aktywują się po osiągnięciu określonego progu. Funkcja aktywacji określa, kiedy sygnał wejściowy jest wystarczający do uruchomienia neuronu, więc wiedza o dostępnych funkcjach aktywatora jest ważna dla rozróżnienia między sieciami neuronowymi. Ponadto musisz wiedzieć o używanym optymalizatorze, aby zapewnić szybkie wyniki, które faktycznie modelują problem, który chcesz rozwiązać. Na koniec musisz zdecydować, jak szybko uczy się Twoja sieć neuronowa. Oszczędź czas i błędy związane z ręcznym wpisywaniem kodu.

Zrozumienie sieci neuronowych

W Internecie można znaleźć wiele dyskusji na temat architektur sieci. Problem, jednak jest to, że wszystkie szybko stają się szalenie złożone, przez co normalni ludzie chcą wyrywać sobie włosy. Pewne niepisane prawo zdaje się mówić, że matematyka musi natychmiast stać się abstrakcyjna i tak skomplikowana, że żaden zwykły śmiertelnik nie może jej zrozumieć, ale każdy może zrozumieć sieć neuronową. Materiał w Części 7 daje dobry początek. Chociaż Część 7 opiera się trochę na matematyce, aby przekazać swój punkt widzenia, matematyka jest stosunkowo prosta. Dowiesz się, jak umieścić w kodzie Pythona wszystkie podstawowe funkcje sieci neuronowej. To, co naprawdę reprezentuje sieć neuronowa, jest rodzajem filtra. Wlewasz dane na górę, te dane przenikają przez różne tworzone warstwy, a dane wyjściowe pojawiają się na dole. Rzeczy, które różnicują sieci neuronowe, to te same rzeczy, których możesz szukać w filtrze. Na przykład rodzaj wybranego algorytmu określa rodzaj filtrowania, jakie będzie wykonywać sieć neuronowa. Możesz chcieć odfiltrować ołów z wody, ale pozostawić wapń i inne korzystne minerały nienaruszone, co oznacza wybór rodzaju filtra, aby to zrobić. Jednak filtry mogą być dostarczane z elementami sterującymi. Na przykład możesz wybrać filtrowanie cząstek o jednym rozmiarze, ale przepuszczać cząstki o innym rozmiarze. Użycie wag i błędów w sieci neuronowej jest po prostu rodzajem kontroli. Regulujesz kontrolkę, aby dostroić otrzymywane filtrowanie. W tym przypadku, ponieważ używasz sygnałów elektrycznych wzorowanych na tych znalezionych w mózgu, sygnał może przejść, gdy spełnia określony warunek – próg zdefiniowany przez funkcję aktywacji. Aby jednak na razie wszystko było proste, pomyśl o tym tak, jak o dostosowaniu podstawowych operacji dowolnego filtra. Możesz monitorować aktywność swojego filtra. Jeśli jednak nie chcesz stać cały dzień, patrząc na to, prawdopodobnie polegasz na jakiejś automatyzacji, aby upewnić się, że wydajność filtra pozostaje stała. W tym miejscu do gry wkracza optymalizator. Optymalizując wyjście sieci neuronowej, widzisz potrzebne wyniki bez ciągłego ręcznego dostrajania. Wreszcie chcesz, aby filtr działał z szybkością i pojemnością, która pozwoli mu prawidłowo wykonywać swoje zadania. Zbyt szybkie wlewanie wody lub innej substancji przez filtr spowodowałoby jego przelanie. Jeśli nie będziesz nalewał wystarczająco szybko, filtr może się zatkać lub działać nieprawidłowo. Dostosowanie szybkości uczenia się optymalizatora sieci neuronowej umożliwia zapewnienie, że sieć neuronowa wygeneruje żądane dane wyjściowe. To jak dostosowanie szybkości nalewania filtra. Sieci neuronowe mogą wydawać się trudne do zrozumienia. Fakt, że wiele z tego, co robią, jest spowitych matematyczną złożonością, nie pomaga. Jednak nie

musisz być naukowcem od rakiet, aby zrozumieć, o co chodzi w sieciach neuronowych. Wszystko, co naprawdę musisz zrobić, to podzielić je na łatwe do opanowania kawałki i spojrzeć na nie z właściwej perspektywy. W poniższych sekcjach pokazano, jak od podstaw kodować każdą część podstawowej sieci neuronowej.

Definiowanie podstawowej architektury

Sieć neuronowa opiera się na wielu jednostkach obliczeniowych, neuronach, ułożonych w warstwy hierarchiczne. Każdy neuron akceptuje dane wejściowe od wszystkich swoich poprzedników i dostarcza dane wyjściowe do swoich następców, dopóki sieć neuronowa jako całość nie spełni wymagań. W tym momencie kończy się przetwarzanie sieciowe i otrzymujesz dane wyjściowe. Wszystkie te obliczenia występują pojedynczo w sieci neuronowej. Sieć przechodzi nad każdym z nich, wykorzystując pętle do iteracji pętli. Można również wykorzystać fakt, że większość tych operacji to zwykłe mnożenie, po którym następuje dodawanie, i skorzystać z obliczeń macierzowych przedstawionych w sekcji „Wykonywanie mnożenia macierzy" w Części 5. Przykład w tej sekcji tworzy sieć z danymi wejściowymi warstwa (której wymiary są określone przez dane wejściowe), ukryta warstwa z trzema neuronami i pojedyncza warstwa wyjściowa, która mówi, czy dane wejściowe są częścią klasy (w zasadzie binarna odpowiedź 0/1). Ta architektura zakłada utworzenie dwóch zestawów wag reprezentowanych przez dwie macierze (kiedy faktycznie używasz macierzy):

- Pierwsza macierz wykorzystuje wielkość określoną przez liczbę wejść x 3, przedstawia wagi, które mnożą wejścia i sumują je do trzech neuronów.
- Druga macierz używa rozmiaru 3 x 1, zbiera wszystkie dane wyjściowe z warstwy ukrytej i sprawia, że warstwa ta zbiega się w dane wyjściowe.

Oto wymagany skrypt Pythona (który może trochę potrwać, w zależności od szybkości twojego systemu):

```
import numpy as np

from sklearn.datasets import make_moons

from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt

%matplotlib inline

def init(inp, out):

return np.random.randn(inp, out) / np.sqrt(inp)

def create_architecture(input_layer, first_layer,

output_layer, random_seed=0):

np.random.seed(random_seed)

layers = X.shape[1], 3 , 1

arch = list(zip(layers[:-1], layers[1:]))

weights = [init(inp, out) for inp, out in arch]

return weights
```

Interesującym punktem tej inicjalizacji jest to, że wykorzystuje ona sekwencję macierzy do automatyzacji obliczeń sieci. Sposób, w jaki kod je inicjuje, ma znaczenie, ponieważ nie można używać zbyt małych liczb - sygnał będzie zbyt słaby, aby sieć mogła działać. Jednak należy również unikać liczb, które są zbyt duże, ponieważ obliczenia stają się zbyt nieporęczne. Czasami zawodzą, co powoduje problem z eksplodującym gradientem lub częściej powoduje nasycenie neuronów, co oznacza, że nie można prawidłowo wytrenować sieci, ponieważ wszystkie neurony są zawsze aktywowane. Inicjowanie sieci przy użyciu samych zer jest zawsze złym pomysłem, ponieważ jeśli wszystkie neurony mają tę samą wartość, zareagują w ten sam sposób na dane treningowe. Bez względu na to, ile neuronów zawiera architektura, działają one jako pojedynczy neuron. Prostszym rozwiązaniem jest rozpoczęcie od początkowych losowych wag, które mieszczą się w zakresie wymaganym dla funkcji aktywacji, czyli funkcji transformacji, które dodają elastyczności w rozwiązywaniu problemów przy użyciu sieci. Możliwym prostym rozwiązaniem jest ustawienie wag na średnią zero i jedno odchylenie standardowe, które w statystyce nazywa się standardowym rozkładem normalnym, a w kodzie występuje jako np.random. polecenie radn.

Co więcej, ponieważ każdy neuron akceptuje wejścia wszystkich poprzednich neuronów, kod przeskalowuje losowe wagi o rozkładzie normalnym, używając pierwiastka kwadratowego z liczby wejść. W konsekwencji neurony i ich funkcje aktywacji zawsze obliczają odpowiedni rozmiar, aby wszystko działało płynnie.

Dokumentowanie podstawowych modułów

Architektura to tylko część sieci neuronowej. Możesz to sobie wyobrazić jako strukturę sieci. Architektura wyjaśnia, w jaki sposób sieć przetwarza dane i dostarcza wyniki. Jednak, aby jakiekolwiek przetwarzanie miało miejsce, musisz również zakodować podstawowe funkcje sieci neuronowej. Pierwszym elementem składowym sieci jest funkcja aktywacji. Część 7 szczegółowo opisuje kilka funkcji aktywacji używanych w sieciach neuronowych, nie wyjaśniając ich szczegółowo. Przykład w tej sekcji zawiera kod funkcji sigmoid, jednej z podstawowych funkcji aktywacji sieci neuronowej. Funkcja sigmoidalna jest krokiem naprzód w stosunku do funkcji kroku Heaviside'a, która działa jak przełącznik, który aktywuje się przy pewnym progu. Funkcja kroku Heaviside'a wyprowadza 1 dla wejść powyżej progu i 0 dla wejść poniżej tego progu. Funkcje sigmoidalne wyprowadzają odpowiednio 0 lub 1 dla małych wartości wejściowych poniżej zera lub wysokie wartości powyżej zera. Dla wartości wejściowych z zakresu od –5 do +5 funkcja wyprowadza wartości w zakresie 0-1, powoli zwiększając wyjściową wartość uwalnianych wartości do około 0,2, a następnie szybko rosnąc liniowo do wartości 0,8. Następnie spada ponownie, gdy wskaźnik produkcji zbliża się do 1. Takie zachowanie reprezentuje krzywą logistyczną, która jest przydatna do opisywania wielu zjawisk naturalnych, takich jak wzrost populacji, która zaczyna powoli rosnąć, a następnie w pełni kwitnie i rozwija się, aż zwolni, zanim uderzy limit zasobów (takich jak dostępna przestrzeń życiowa lub żywność). W sieciach neuronowych funkcja sigmoidalna jest szczególnie przydatna do modelowania danych wejściowych, które przypominają prawdopodobieństwa, i jest różniczkowalna, co jest aspektem matematycznym, który pomaga odwrócić jej efekty i opracować najlepszą fazę wstecznej propagacji.

```
def sigmoid(z):

return 1/(1 + np.exp(-z))

def sigmoid_prime(s):

return s * (1 -s)
```

Po utworzeniu funkcji aktywacji można utworzyć procedurę przekazywania, która jest mnożeniem macierzy między danymi wejściowymi każdej warstwy a wagami połączenia. Po zakończeniu mnożenia kod stosuje funkcję aktywacji do wyników w celu przekształcenia ich w sposób nieliniowy. Poniższy kod osadza funkcję sigmoid w kodzie przekazywania sieci. Oczywiście w razie potrzeby możesz użyć dowolnej innej funkcji aktywacji.

```
def feed_forward(X, weights):

a = X.copy()

out = list()

for W in weights:

z = np.dot(a, W)

a = sigmoid(z)

out.append(a)

return out
```

Stosując feed forward do całej sieci, w końcu uzyskujesz wynik w warstwie wyjściowej. Teraz możesz porównać wyniki z rzeczywistymi wartościami, które chcesz uzyskać w sieci. Funkcja dokładności określa, czy sieć neuronowa dobrze wykonuje predykcje, porównując liczbę poprawnych domysłów z całkowitą liczbą dostarczonych predykcji.

```
def accuracy(true_label, predicted):

correct_preds = np.ravel(predicted)==true_label

return np.sum(correct_preds) / len(true_label)
```

Funkcja wstecznej propagacji jest następna, ponieważ sieć działa, ale wszystkie lub niektóre prognozy są nieprawidłowe. Korygowanie przewidywań podczas uczenia umożliwia tworzenie sieci neuronowej, która może przyjmować nowe przykłady i dostarczać dobrych przewidywań. Szkolenie jest uwzględniane w wagach połączeń jako wzorce obecne w danych, które mogą pomóc w prawidłowym przewidywaniu wyników. Aby wykonać propagację wsteczną, najpierw obliczasz błąd na końcu każdej warstwy (ta architektura ma dwie). Używając tego błędu, mnożysz go przez pochodną funkcji aktywacji. Wynik zapewnia gradient, czyli zmianę wag niezbędną do dokładniejszego obliczania predykcji. Kod zaczyna się od porównania wyniku z poprawnymi odpowiedziami (l2_error), a następnie oblicza gradienty, które są niezbędnymi poprawkami wagowymi (l2_delta). Następnie kod mnoży gradienty przez wagi, które kod musi skorygować. Operacja dystrybuuje błąd z warstwy wyjściowej do warstwy pośredniej (l1_error). Nowe obliczanie gradientu (l1_delta) zapewnia również korekty wagi do zastosowania w warstwie wejściowej, co kończy proces dla sieci z warstwą wejściową, warstwą ukrytą i warstwą wyjściową.

```
def backpropagation(l1, l2, weights, y):

l2_error = y.reshape(-1, 1) - l2

l2_delta = l2_error * sigmoid_prime(l2)

l1_error = l2_delta.dot(weights[1].T)

l1_delta = l1_error * sigmoid_prime(l1)
```

return l2_error, l1_delta, l2_delta

To jest tłumaczenie kodu Pythona, w uproszczonej formie, formuł z Części 7. Funkcja kosztu to różnica między danymi wyjściowymi sieci a poprawnymi odpowiedziami. Przykład nie dodaje błędów systematycznych podczas fazy sprzężenia do przodu, co zmniejsza złożoność procesu propagacji wstecznej i ułatwia zrozumienie. Po tym, jak wsteczna propagacja przypisze każdemu połączeniu jego część korekty, która powinna zostać zastosowana w całej sieci, należy dostosować początkowe wagi tak, aby reprezentowały zaktualizowaną sieć neuronową. Robisz to, dodając do wag każdej warstwy, mnożenie danych wejściowych dla tej warstwy oraz korekty delta dla warstwy jako całości. Jest to krok metody opadania gradientu, w którym zbliżasz się do rozwiązania, wykonując powtarzające się małe kroki we właściwym kierunku, więc może być konieczne dostosowanie rozmiaru kroku używanego do rozwiązania problemu. Parametry alfa ułatwiają zmianę wielkości kroku. Użycie wartości 1 nie wpłynie na wpływ poprzedniej korekty wagi, ale wartości mniejsze niż 1 skutecznie go zmniejszą.

def update_weights(X, l1, l1_delta, l2_delta, weights,

alpha=1.0):

weights[1] = weights[1] + (alpha * l1.T.dot(l2_delta))

weights[0] = weights[0] + (alpha * X.T.dot(l1_delta))

return weights

Sieć neuronowa nie jest kompletna, jeśli może uczyć się tylko na podstawie danych, ale nie może przewidywać. Ostatnia funkcja przewidywania przesyła nowe dane za pomocą sprzężenia do przodu, odczytuje ostatnią warstwę wyjściową i przekształca jej wartości w przewidywania problemów. Ponieważ funkcja aktywacji sigmoidy jest tak biegła w modelowaniu prawdopodobieństwa, kod używa wartości połowicznej od 0 do 1, czyli 0,5, jako próg dla dodatniego lub ujemnego wyjścia. Takie binarne dane wyjściowe mogą pomóc w zaklasyfikowaniu dwóch klas lub jednej klasy względem wszystkich pozostałych, jeśli zestaw danych zawiera trzy lub więcej typów wyników do sklasyfikowania.

def predict(X, weights):

_, l2 = feed_forward(X, weights)

preds = np.ravel((l2 > 0.5).astype(int))

return preds

W tym momencie przykład zawiera wszystkie części, dzięki którym sieć neuronowa działa. Potrzebujesz tylko problemu, który pokazuje, jak działa sieć neuronowa.

Rozwiązywanie prostego problemu

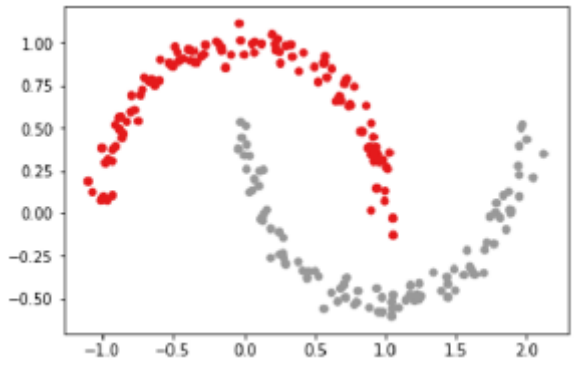
W tej sekcji testujesz napisany przez siebie kod sieci neuronowej, prosząc go o rozwiązanie prostego, ale nie banalnego problemu z danymi. Kod wykorzystuje funkcję make_moons pakietu Scikit-learn, aby utworzyć dwa przeplatające się okręgi punktów w kształcie dwóch półksiężyców. Rozdzielenie tych dwóch okręgów wymaga algorytmu zdolnego do zdefiniowania nieliniowej funkcji separacji, która uogólnia nowe przypadki tego samego rodzaju. Sieć neuronowa, taka jak ta przedstawiona wcześniej w rozdziale, z łatwością poradzi sobie z tym wyzwaniem.

np.random.seed(0)

coord, cl = make_moons(300, noise=0.05)

```
X, Xt, y, yt = train_test_split(coord, cl,

test_size=0.30,

random_state=0)

plt.scatter(X[:,0], X[:,1], s=25, c=y, cmap=plt.cm.Set1)

plt.show()
```

Kod najpierw ustawia losowy ziarno, aby generował ten sam wynik za każdym razem, gdy chcesz uruchomić przykład. Następnym krokiem jest utworzenie 300 przykładów danych i podzielenie ich na pociąg i testowy zestaw danych. (Zbiór danych testowych stanowi 30 procent całości). Dane składają się z dwóch zmiennych reprezentujących współrzędne x i y punktów na wykresie kartezjańskim. Rysunek 8-1 przedstawia wyniki tego procesu.



Ponieważ uczenie się w sieci neuronowej odbywa się w kolejnych iteracjach (zwanych epokami), po utworzeniu i zainicjowaniu zestawów wag kod zapętla 30 000 iteracji danych z dwóch półksiężyców (każdy fragment jest epoką). W każdej iteracji skrypt wywołuje niektóre z wcześniej przygotowanych podstawowych funkcji sieci neuronowej:

- Przesyłaj dane przez całą sieć.
- Przenieś błąd z powrotem do sieci.
- Aktualizuj wagi każdej warstwy w sieci na podstawie błędu wstecznej propagacji.
- Oblicz błędy szkolenia i walidacji.

Poniższy kod używa komentarzy, aby szczegółowo opisać, kiedy każda funkcja działa:

```
weights = create_architecture(X, 3, 1)

for j in range(30000 + 1):

# First, feed forward through the hidden layer

l1, l2 = feed_forward(X, weights)

# Then, error backpropagation from output to input

l2_error, l1_delta, l2_delta = backpropagation(l1,

l2, weights, y)

# Finally, updating the weights of the network

weights = update_weights(X, l1, l1_delta, l2_delta,
```

```
weights, alpha=0.05)

# From time to time, reporting the results

if (j % 5000) == 0:

train_error = np.mean(np.abs(l2_error))

print('Epoch {:5}'.format(j), end=' - ')

print('error: {:0.4f}'.format(train_error),

end= ' - ')

train_accuracy = accuracy(true_label=y,

predicted=(l2 > 0.5))

test_preds = predict(Xt, weights)

test_accuracy = accuracy(true_label=yt,

predicted=test_preds)

print('acc: train {:0.3f}'.format(train_accuracy),

end= ' | ')

print('test {:0.3f}'.format(test_accuracy))
```

Zmienna j zlicza iteracje. W każdej iteracji kod próbuje podzielić j przez 5000 i sprawdzić, czy z dzielenia wychodzi moduł. Gdy moduł ma wartość zero, kod wnioskuje, że od poprzedniego sprawdzenia minęło 5000 epok, a podsumowanie błędu sieci neuronowej jest możliwe poprzez zbadanie jego dokładności (ile razy predykcja jest poprawna w odniesieniu do całkowitej liczby predykcji) na zestawie treningowym i na zestawie testowym. Dokładność zbioru uczącego Zmienna j zlicza iteracje. W każdej iteracji kod próbuje podzielić j przez 5000 i sprawdzić, czy z dzielenia wychodzi moduł. Gdy moduł ma wartość zero, kod wnioskuje, że od poprzedniego sprawdzenia minęło 5000 epok, a podsumowanie błędu sieci neuronowej jest możliwe poprzez zbadanie jego dokładności (ile razy predykcja jest poprawna w odniesieniu do całkowitej liczby predykcji) na zestawie treningowym i na zestawie testowym. Dokładność na zbiorze uczącym pokazuje, jak dobrze sieć neuronowa dopasowuje dane, dostosowując jej parametry przez proces wstecznej propagacji błędów. Dokładność zestawu testowego daje wyobrażenie o tym, jak dobrze rozwiązanie uogólniło się na nowe dane, a zatem czy można je ponownie wykorzystać. Dokładność testu powinna mieć największe znaczenie, ponieważ pokazuje potencjalną użyteczność sieci neuronowej z innymi danymi. Dokładność treningu mówi tylko o tym, jak sieć wypada na podstawie aktualnych danych, z których korzystasz.

Spojrzenie pod maskę sieci neuronowych

Gdy już wiesz, jak zasadniczo działają sieci neuronowe, potrzebujesz lepszego zrozumienia, co je różnicuje. Oprócz różnych architektur, wybór funkcji aktywacji, optymalizatorów i szybkości uczenia się sieci neuronowej może mieć znaczenie. Znajomość podstawowych operacji nie wystarczy, ponieważ nie uzyskasz pożądanych rezultatów. Spojrzenie pod maskę pomaga zrozumieć, w jaki sposób można dostroić rozwiązanie sieci neuronowej do modelowania określonych problemów. Ponadto zrozumienie różnych algorytmów używanych do tworzenia sieci neuronowej pomoże Ci uzyskać lepsze wyniki przy mniejszym wysiłku i krótszym czasie. Poniższe sekcje skupiają się na trzech obszarach różnicowania sieci neuronowych.

Wybór odpowiedniej funkcji aktywacji

Funkcja aktywacji po prostu określa, kiedy neuron się uruchamia. Potraktuj to jako punkt krytyczny: wprowadzenie określonej wartości nie spowoduje pobudzenia neuronu, ponieważ to nie wystarczy, ale tylko trochę więcej danych wejściowych może spowodować uruchomienie neuronu. Neuron definiuje się w prosty sposób w następujący sposób:

y = Σ (waga * dane wejściowe) + odchylenie

Wyjście, y, może mieć dowolną wartość z zakresu od + nieskończoności do – nieskończoności. Problem polega więc na tym, aby zdecydować, jaka wartość y jest wartością strzału, czyli w tym miejscu w grę wchodzi funkcja aktywacji. Funkcja aktywacji określa, która wartość jest wystarczająco wysoka lub wystarczająco niska, aby odzwierciedlić punkt decyzyjny w sieci neuronowej dla konkretnego neuronu lub grupy neuronów. Jak wszystko inne w sieciach neuronowych, nie masz tylko jednej funkcji aktywacji. Korzystasz z funkcji aktywacji, która działa najlepiej w konkretnym scenariuszu. Mając to na uwadze, możesz podzielić funkcje aktywacji na następujące kategorie:

- Krok: Funkcja kroku (nazywana również funkcją binarną) opiera się na określonym progu przy podejmowaniu decyzji o aktywacji lub nie. Korzystanie z funkcji kroku oznacza, że wiesz, jaka konkretna wartość spowoduje aktywację. Jednak funkcje kroku są ograniczone, ponieważ są w pełni aktywowane lub całkowicie dezaktywowane – nie istnieją żadne odcienie szarości. W konsekwencji przy próbie określenia, która klasa jest najprawdopodobniej poprawna na podstawie danych wejściowych, funkcja kroku nie zadziała.
- Liniowa: Funkcja liniowa (A = cx) zapewnia prostoliniowe określenie aktywacji na podstawie danych wejściowych. Korzystanie z funkcji liniowej pomaga określić, które wyjście aktywować w oparciu o to, które wyjście jest najbardziej poprawne (wyrażone przez ważenie). Jednak funkcje liniowe działają tylko jako pojedyncza warstwa. Jeśli miałbyś ułożyć wiele warstw funkcji liniowych, wynik byłby taki sam jak za pomocą pojedynczej warstwy, co jest sprzeczne z celem korzystania z sieci neuronowych. W konsekwencji funkcja liniowa może występować jako pojedyncza warstwa, ale nigdy jako wiele warstw.
- Sigmoid: Funkcja sigmoidalna ($A = 1 / 1 + e^{-x}$), która tworzy krzywą w kształcie litery C lub S, jest nieliniowa. Rozpoczyna się czymś w rodzaju funkcji kroku, z wyjątkiem tego, że wartości między dwoma punktami faktycznie istnieją na krzywej, co oznacza, że można układać funkcje sigmoidalne w celu przeprowadzenia klasyfikacji z wieloma danymi wyjściowymi. Zakres funkcji sigmoidalnej wynosi od 0 do 1, a nie – nieskończoność do + nieskończoność jak w przypadku funkcji liniowej, więc aktywacje są ograniczone w określonym zakresie. Jednak funkcja sigmoidalna ma problem zwany znikającym gradientem, co oznacza, że funkcja odmawia uczenia się po pewnym punkcie, ponieważ propagowany błąd zmniejsza się do zera, gdy zbliża się do odległych warstw.
- Tanh: Funkcja tanh ($A = (2 / 1 + e^{-2x}) – 1$) jest w rzeczywistości skalowaną funkcją sigmoidalną. Ma zakres od –1 do 1, więc znowu jest to precyzyjna metoda aktywacji neuronów. Duża różnica między funkcjami sigmoidalnymi a funkcjami tanh polega na tym, że gradient funkcji tanh jest silniejszy, co oznacza, że wykrywanie małych różnic jest łatwiejsze, dzięki czemu klasyfikacja jest bardziej czuła. Jak funkcja esicy , tanh cierpi na znikające problemy z gradientem.
- ReLU: Funkcja ReLU lub Rectified Linear Units (A(x) = max(0, x)) daje wyjście w zakresie od 0 do nieskończoności, więc jest podobna do funkcji liniowej z tym wyjątkiem, że jest również nieliniowa, umożliwiając układanie funkcji ReLU. Zaletą ReLU jest to, że wymaga mniejszej mocy obliczeniowej, ponieważ mniej neuronów się uruchamia. Brak aktywności w miarę zbliżania się neuronu do części linii zerowej oznacza, że jest mniej potencjalnych wyjść, na

które można się przyjrzeć. Jednak ta zaleta może stać się również wadą, gdy masz problem zwany umierającym ReLU. Po pewnym czasie wagi sieci neuronowej nie dają już pożądanego efektu (po prostu przestają się uczyć) i dotknięte nią neurony umierają – nie reagują na żadne dane wejściowe.

Ponadto ReLU ma kilka wariantów, które powinieneś rozważyć:

- ELU (Exponential Linear Unit): Różni się od ReLU, gdy wejścia są ujemne. W tym przypadku wyjścia nie spadają do zera, ale powoli spadają wykładniczo do -1.
- PReLU (Parametric Rectified Linear Unit): Różni się od ReLU, gdy wejścia są ujemne. W tym przypadku wyjściem jest funkcja liniowa, której parametry są poznawane przy użyciu tej samej techniki, co każdy inny parametr sieci.
- LeakyReLU: Podobny do PReLU, ale parametr dla strony liniowej jest stały.

Poleganie na inteligentnym optymalizatorze

Optymalizator zapewnia, że sieć neuronowa szybko i poprawnie modeluje każdy problem, który chcesz rozwiązać, modyfikując obciążenia i wagi sieci neuronowej. Okazuje się, że to zadanie wykonuje algorytm, ale musisz wybrać odpowiedni algorytm, aby uzyskać oczekiwane rezultaty. Podobnie jak w przypadku wszystkich scenariuszy sieci neuronowych, masz do wyboru kilka opcjonalnych typów algorytmów:

- Stochastyczny spadek gradientowy (SGD)
- RMSPPro
- AdaGrad
- AdaDelta
- AMSGrad
- Adam i jego warianty, Adamax i Nadam

Optymalizator działa poprzez minimalizację lub maksymalizację wyniku funkcji celu (znanej również jako funkcja błędu) reprezentowanej jako E(x). Ta funkcja jest zależna od wewnętrznych parametrów uczących się modelu używanych do obliczania wartości docelowych (Y) z predyktorów (X). Dwa wewnętrzne parametry do uczenia się to wagi (W) i stronniczość (b). Różne algorytmy mają różne metody radzenia sobie z funkcją celu. Możesz skategoryzować funkcje optymalizatora według sposobu, w jaki radzą sobie z pochodną (dy/dx), która jest chwilową zmianą y względem x. Oto dwa poziomy obsługi instrumentów pochodnych:

- Pierwszego rzędu: Algorytmy te minimalizują lub maksymalizują funkcję celu przy użyciu wartości gradientu w odniesieniu do parametrów.
- Drugi rząd: Te algorytmy minimalizują lub maksymalizują funkcję obiektu przy użyciu wartości pochodnych drugiego rzędu w odniesieniu do parametrów. Pochodna drugiego rzędu może dać wskazówkę, czy pochodna pierwszego rzędu rośnie czy maleje, co dostarcza informacji o krzywiźnie linii.

Często stosuje się techniki optymalizacji pierwszego rzędu, takie jak Gradient Descent, ponieważ wymagają one mniejszej liczby obliczeń i stosunkowo szybko zbliżają się do dobrego rozwiązania podczas pracy z dużymi zestawami danych.

Ustalanie działającego tempa uczenia się

Każdy optymalizator ma zupełnie inne parametry do dostrojenia. Jedną ze stałych jest ustalenie szybkości uczenia się, która reprezentuje szybkość, z jaką kod aktualizuje wagi sieci (takie jak parametr

alfa użyty w przykładzie dla tego rozdziału). Szybkość uczenia się może wpływać zarówno na czas potrzebny sieci neuronowej do nauczenia się dobrego rozwiązania (liczba epok), jak i na wynik. W rzeczywistości, jeśli tempo uczenia się jest zbyt niskie, Twoja sieć będzie się uczyć w nieskończoność. Ustawienie zbyt dużej wartości powoduje niestabilność podczas aktualizowania wag, a sieć nigdy nie osiągnie dobrego rozwiązania. Wybór szybkości uczenia się, która działa jest zniechęcająca, ponieważ można skutecznie wypróbować wartości z zakresu od 0,000001 do 100. Najlepsza wartość różni się w zależności od optymalizatora. Wybrana wartość zależy od rodzaju posiadanych danych. Teoria może być tutaj mało pomocna; musisz przetestować różne kombinacje, zanim znajdziesz najbardziej odpowiednią szybkość uczenia się, aby pomyślnie wytrenować sieć neuronową. Pomimo całej otaczającej je matematyki, strojenie sieci neuronowych i zapewnienie ich najlepszego działania jest głównie kwestią empirycznych wysiłków w próbowaniu różnych kombinacji architektur i parametrów.

Przejście do głębokiego uczenia

Części 7 i 8 przedstawiają sztuczną inteligencję z perspektywy uczenia maszynowego, z dodatkowymi informacjami na temat uczenia głębokiego. Ta Część dotyczy wyłącznie głębokiego uczenia się, ponieważ tak naprawdę potrzebujesz rozwiązań do głębokiego uczenia się, aby pracować z dzisiejszą nadmiarem danych w inteligentny sposób. Chociaż uczenie maszynowe dodaje możliwość uczenia się do arsenału sztucznej inteligencji, należy od samego początku uświadomić sobie, że komputery mają ograniczenia – tak naprawdę nie rozumieją, co robią ludzie. Algorytmy, które są matematycznymi reprezentacjami różnych procesów interpretacji danych, kontrolują wszystko. Tak więc w pierwszej części przyjrzymy się danym z perspektywy uczenia głębokiego, ponieważ do skutecznego dopasowywania wzorców potrzebne są ogromne ilości danych. Wraz z przejściem od sztucznej inteligencji do uczenia maszynowego i głębokiego uczenia wzrastają wymagania obliczeniowe. W rzeczywistości jednym z głównych powodów zimy AI w przeszłości był brak mocy obliczeniowej. Obecnie do przetwarzania danych można używać procesorów graficznych, takich jak NVIDIA Titan V z 5120 rdzeniami Compute Unified Device Architecture (CUDA). w sposób, który nie był możliwy nawet kilka lat temu. Dlatego w drugiej części omówiono, w jaki sposób możesz poprawić swoje doświadczenie głębokiego uczenia się, dodając więcej sprzętu w nim lub przy użyciu innych strategii stosowanych obecnie przez naukowców zajmujących się danymi (m.in.). Trzecia część koncentruje się dokładnie na tym, jak głębokie uczenie się różni od uczenia maszynowego – różnica, która jest stałym źródłem problemów dla wielu osób. Znalezienie dokładnej definicji, z którą każdy może się zgodzić, jest prawie niemożliwe, więc jeśli jesteś już ekspertem w dziedzinie głębokiego uczenia się, możesz nie do końca zgadzać się ze wszystkim, co ten Część ma do powiedzenia. Opieramy się na tej definicji, aby przedstawić zasady i przykłady głębokiego uczenia się, więc musisz znać szczególny sposób postrzegania głębokiego uczenia się. Wreszcie, czwarta część zawiera wszystkie istotne elementy, które odkryjesz w pierwszych trzech częściach i poprawia je. Zaczynasz zdawać sobie sprawę, że uczenie głębokie ma wiele form i że niektóre formy są szczególnie odpowiednie do rozwiązywania konkretnych problemów. Obecnie nie istnieje jedno rozwiązanie, które rozwiązuje każdy problem, nawet niedostatecznie, więc znajomość właściwego zestawu rozwiązań do rozwiązania konkretnego problemu może zaoszczędzić dużo czasu i frustracji.

Widząc dane wszędzie

Big data to coś więcej niż modne hasło używane przez dostawców do proponowania nowych sposobów przechowywania danych i ich analizy. Rewolucja big data to codzienność i siła napędowa naszych czasów. Być może słyszeliście o big data w wielu specjalistycznych publikacjach naukowych i biznesowych i zastanawialiście się, co tak naprawdę oznacza ten termin. Z technicznego punktu widzenia big data odnosi się do dużych i złożonych ilości danych komputerowych, tak dużych i skomplikowanych, że aplikacje nie mogą sobie z nimi poradzić, wykorzystując dodatkową pamięć masową lub zwiększając moc komputera. Poniższe sekcje pomogą Ci zrozumieć, co sprawia, że dane są dziś uniwersalnym zasobem.

Biorąc pod uwagę efekty konstrukcji

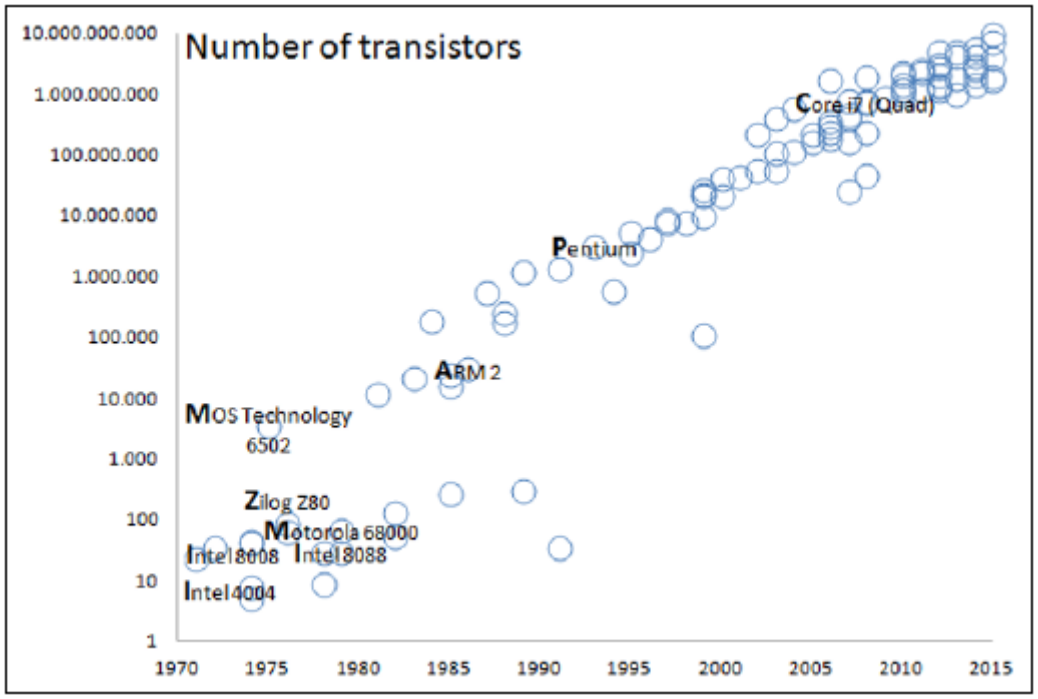
Big data oznacza rewolucję w przechowywaniu i manipulacji danymi. Wpływa na to, co możesz osiągnąć z danymi w kategoriach bardziej jakościowych (co oznacza, że oprócz robienia więcej, możesz lepiej wykonywać zadania). Komputery przechowują duże zbiory danych w różnych formatach z ludzkiego punktu widzenia, ale komputer postrzega dane jako strumień jedynek i zer (podstawowy język komputerów). Możesz przeglądać dane jako jeden z dwóch typów, w zależności od tego, jak je produkujesz i zużywasz:

- Zorganizowany: wiesz dokładnie, co zawiera i gdzie znaleźć każdy element danych. Typowymi przykładami danych strukturalnych są tabele bazy danych, w których informacje są uporządkowane w kolumny, a każda kolumna zawiera określony rodzaj informacji. Dane są często ustrukturyzowane zgodnie z projektem. Zbierasz go selektywnie i zapisujesz we właściwym miejscu. Na przykład możesz chcieć umieścić liczbę osób kupujących określony produkt w określonej kolumnie, w określonej tabeli, w określonej bazie danych. Podobnie jak w przypadku biblioteki, jeśli wiesz, jakich danych potrzebujesz, możesz je od razu znaleźć.
- Bez struktury: Masz pojęcie, co zawiera, ale nie wiesz dokładnie, jak to jest ułożone. Typowymi przykładami danych nieustrukturyzowanych są obrazy, filmy i nagrania dźwiękowe. Możesz użyć nieustrukturyzowanego formularza dla tekstu, aby oznaczyć go cechami, takimi jak rozmiar, data lub typ zawartości. Zwykle nie wiesz dokładnie, gdzie pojawiają się dane w nieustrukturyzowanym zestawie danych, ponieważ dane są wyświetlane jako sekwencje jedynek i zer, które aplikacja musi interpretować lub wizualizować.

Przekształcenie nieustrukturyzowanych danych w ustrukturyzowany formularz może kosztować dużo czasu i wysiłku oraz może wiązać się z pracą wielu osób. Większość danych z rewolucji big data jest nieustrukturyzowana i przechowywana w takiej postaci, w jakiej są, chyba że ktoś renderuje je w ustrukturyzowanej formie. Ten obszerny i wyrafinowany magazyn danych nie pojawił się nagle z dnia na dzień. Opracowanie technologii przechowywania takiej ilości danych wymagało czasu. Czasu zajęło również rozpowszechnienie technologii generującej i dostarczającej dane, a mianowicie komputerów, czujników, inteligentnych telefonów komórkowych, Internetu i usług World Wide Web.

Zrozumienie implikacji Moore’a

W 1965 r. Gordon Moore, współzałożyciel firm Intel i Fairchild Semiconductor, napisał w artykule zatytułowanym „Cramming More Components Onto Integrated Circuits” , że liczba komponentów znalezionych w układach scalonych obwody podwajałyby się co roku przez następną dekadę. W tym czasie w elektronice dominowały tranzystory. Możliwość umieszczenia większej liczby tranzystorów w układzie scalonym (IC) oznaczała możliwość uczynienia urządzeń elektronicznych bardziej wydajnymi i użytecznymi. Proces ten nazywa się integracją i oznacza silny proces miniaturyzacji elektroniki (sprawianie, że ten sam obwód jest znacznie mniejszy). Dzisiejsze komputery nie są dużo mniejsze niż komputery sprzed dekady, ale są zdecydowanie potężniejsze. To samo dotyczy telefonów komórkowych. Mimo że są tego samego rozmiaru co ich poprzednicy, są w stanie wykonywać więcej zadań. To, co Moore stwierdził w tym artykule, było w rzeczywistości prawdą od wielu lat. Branża półprzewodników nazywa to prawem Moore'a. Zgodnie z przewidywaniami, przez pierwsze dziesięć lat miało miejsce podwojenie. W 1975 roku Moore poprawił swoje oświadczenie, przewidując podwojenie co dwa lata. Rysunek 9-1 przedstawia skutki tego podwojenia. Ten wskaźnik podwojenia jest nadal aktualny, choć obecnie panuje opinia, że nie utrzyma się on dłużej niż do końca obecnej dekady (do około 2020 roku). Od 2012 r. zaczęła pojawiać się rozbieżność między spodziewanym wzrostem prędkości a tym, co firmy półprzewodnikowe mogą osiągnąć w zakresie miniaturyzacji.



Istnieją fizyczne bariery w integracji większej liczby obwodów w układzie scalonym przy użyciu obecnych komponentów krzemionkowych, ponieważ można zrobić tylko tak małe rzeczy. Jednak innowacje są kontynuowane. W przyszłości prawo Moore'a może nie mieć zastosowania. Stanie się tak, ponieważ przemysł przejdzie na nową technologię, taką jak wytwarzanie komponentów przy użyciu laserów optycznych zamiast tranzystorów. W końcu ludzie zlekceważą prawo Moore'a, ponieważ przemysł nie będzie w stanie nadążyć za tempem, jak to miało miejsce w przeszłości .

Biorąc pod uwagę zmiany w prawie Moore’a

Dla naukowców zajmujących się danymi i innych zainteresowanych głębokim uczeniem ważne jest to, że od 1965 r. podwojenie komponentów co dwa lata zapoczątkowało wielki postęp w elektronice cyfrowej, który miał daleko idące konsekwencje w pozyskiwaniu, przechowywaniu, manipulacji i zarządzaniu danych. Prawo Moore'a ma bezpośredni wpływ na dane. Zaczyna się od inteligentnych urządzeń. Im inteligentniejsze urządzenie, tym więcej osób polega na nim w celu interakcji z danymi na nowe sposoby (o czym świadczy fakt, że elektronika jest dziś wszędzie). Im większa dyfuzja tej mocy obliczeniowej, tym niższa staje się cena, tworząc niekończącą się pętlę, która napędza wszędzie użycie potężnych maszyn obliczeniowych i małych czujników. Przy dużej ilości dostępnej pamięci komputera i większych dyskach do przechowywania danych konsekwencją jest zwiększenie dostępności danych, takich jak strony internetowe, zapisy transakcji, pomiary, obrazy cyfrowe i inne rodzaje danych. Bez tych postępów dzisiejszy Internet nie byłby możliwy, ponieważ przepływ danych zależy od tak inteligentnych urządzeń. Dzięki połączonym z nim komputerom, urządzeniom mobilnym i czujnikom, Internet generuje i dystrybuuje nowe dane w dużych ilościach. Niektóre źródła szacują obecną dzienną produkcję danych na około 2,5 kwintyliona (liczba z 18 zerami) bajtów, przy czym lwia część przypada na dane nieustrukturyzowane, takie jak wideo i audio. Większość tych danych dotyczy typowych ludzkich działań, uczuć, doświadczeń i relacji, którym towarzyszy rosnący udział danych dotyczących funkcjonowania podłączonych maszyn, od złożonych maszyn przemysłowych po proste lampy inteligentnego domu (lampy, którymi można sterować zdalnie za pomocą Internet).

Odkrywanie korzyści płynących z dodatkowych danych

Wraz z eksplozją dostępności danych na urządzeniach cyfrowych, dane nabierają nowych niuansów wartości i użyteczności, wykraczając poza ich początkowy zakres nauczania (szkolenia) i przekazywania wiedzy (przekazywania danych). Obfitość danych, rozpatrywana w ramach analizy danych, zyskuje nowe funkcje odróżniające ją od informacyjnych:

- Dane lepiej opisują świat, przedstawiając szeroką gamę faktów i bardziej szczegółowo, dostarczając niuanse dla każdego faktu. Stało się tak obfite, że obejmuje każdy aspekt rzeczywistości. Możesz go użyć, aby odkryć, w jaki sposób nawet pozornie niepowiązane rzeczy i fakty rzeczywiście odnoszą się do siebie.
- Dane pokazują, jak fakty łączą się ze zdarzeniami. Możesz wyprowadzić ogólne zasady i dowiedzieć się, jak świat będzie się zmieniał lub przekształcał przy określonych przesłankach. Kiedy ludzie działają w określony sposób, dane zapewniają również pewną zdolność przewidywania.

W poniższych sekcjach omówiono, w jaki sposób posiadanie większej ilości danych jest zwykle lepsze. Mając więcej danych do pracy, Twój projekt głębokiego uczenia się może stać się bardziej dokładny, niezawodny i, w niektórych przypadkach, wykonalny

Definiowanie rozgałęzień danych

Pod pewnymi względami dane dostarczają nam nowych supermocy. Chris Anderson, poprzedni redaktor naczelny Wired, omawia, jak duże ilości danych mogą pomóc w odkryciach naukowych poza metodami naukowymi. Autor przytacza przykład osiągnięć Google w branży reklamowej i tłumaczeniowej, w której Google zyskało na znaczeniu nie poprzez stosowanie konkretnych modeli czy teorii, ale raczej poprzez zastosowanie algorytmów do uczenia się na podstawie danych. Podobnie jak w reklamie, dane naukowe (takie jak fizyka lub biologia) mogą wspierać innowacje, które pozwalają naukowcom podejść do problemów bez hipotez, zamiast tego uwzględniając różnice stwierdzone w dużych ilościach danych i stosując algorytmy odkrywania. Galileo Galilei oparł się na metodzie naukowej, aby stworzyć podstawy współczesnej fizyki i astronomii. Większość wczesnych postępów opiera się na obserwacjach i kontrolowanych eksperymentach, które określają przyczyny tego, jak i dlaczego coś się dzieje. Zdolność do innowacji przy użyciu samych danych to duży przełom w sposobie, w jaki możemy zrozumieć świat. W przeszłości naukowcy przeprowadzili niezliczone obserwacje i dokonali wielu dedukcji, aby opisać fizykę wszechświata. Ten ręczny proces pozwolił ludziom znaleźć podstawowe prawa świata, w którym żyjemy. Analiza danych poprzez parowane obserwacje wyrażone jako wejścia i wyjścia, pozwalają określić, jak rzeczy działają i zdefiniować, dzięki głębokiemu uczeniu, przybliżonym regułom lub prawom, świata bez konieczności uciekania się do ręcznych obserwacji i dedukcji. Proces jest teraz szybszy i bardziej automatyczny.

Biorąc pod uwagę terminowość i jakość danych

Dane umożliwiają nie tylko wspomaganie głębokiego uczenia się, ale także głębokie uczenie. Niektórzy powiedzieliby, że uczenie głębokie jest wynikiem wyrafinowanych algorytmów o podwyższonej złożoności matematycznej i to z pewnością prawda. Działania takie jak wizja i rozumienie języka wymagają algorytmów, które nie są łatwe do wyjaśnienia w terminach laika i wymagają milionów obliczeń do działania. (Sprzęt również odgrywa tutaj rolę). Głębokie uczenie to jednak coś więcej niż algorytmy. Dr Alexander Wissner-Gross, amerykański naukowiec, przedsiębiorca i pracownik Institute for Applied Computation Science na Harvardzie, przedstawia swoje spostrzeżenia na temat głębokich zarobków w niedawnym wywiadzie dla Edge. Wywiad pokazuje, dlaczego rozwój technologii głębokiego uczenia się trwał tak długo. Wissner-Gross konkluduje, że jakość i dostępność danych mogły być kluczowymi czynnikami, a nie tylko dostępnością algorytmiczną. Innymi słowy, posiadanie potężnych algorytmów jest konieczne, ale niewystarczające, jeśli nie masz odpowiednich danych do

ich uruchomienia. Wissner-Gross dokonuje przeglądu najbardziej przełomowych osiągnięć w dziedzinie głębokiego uczenia się w ostatnich latach, pokazując, w jaki sposób dane i algorytmy przyczyniają się do sukcesu każdego przełomu i podkreślając, jak każdy z nich był świeży w momencie, gdy społeczność AI osiągnęła kamień milowy. Wissner-Gross pokazuje, że dane są stosunkowo nowe i zawsze aktualizowane, podczas gdy algorytmy nie są nowymi odkryciami, ale opierają się na konsolidacji starszej technologii. Na przykład, jeśli weźmiesz pod uwagę ostatnie osiągnięcia uczenia głębokiego, prawie ludzka wydajność sieci GoogleLeNet w prawidłowym klasyfikowaniu obrazów do klas opiera się na starym algorytmie uruchamianym na najnowszych danych. Wykorzystuje Convolutional Neural Networks for Visual Recognition, algorytm opracowany w 1989 roku, który mógł pokazać swoją rzeczywistą skuteczność dopiero po przeszkoleniu z wykorzystaniem korpusu ImageNet zawierającego ponad 1,5 miliona obrazów, rozpowszechnianych ponad 1000 kategorii (korpus ImageNet został udostępniony w 2010 roku). Kolejnym osiągnięciem do rozważenia jest wynik zespołu Google DeepMind. Zespół wdrożył głęboką sieć neuronową, która osiąga takie same umiejętności, jak ludzie, w graniu w 29 różnych gier na Atari. Opierali się na algorytmie Q-Learning z 1992 roku, który mogli zastosować w grach na Atari dopiero po 2013 roku, kiedy splotowe sieci neuronowe stały się bardziej powszechne, oraz na kompletnym zbiorze danych zawierającym 50 gier na Atari 2600, zwanym Arcade Learning Environment. Wissner-Gross podaje inne przykłady tego samego rodzaju osiągnięć w zakresie uczenia głębokiego, na przykład kiedy IBM Deep Blue pokonał Garry'ego Kasparowa i kiedy IBM Watson staje się światowym zagrożeniem! mistrz. We wszystkich tych przypadkach Wissner-Gross stwierdza, że algorytm jest zwykle starszy od danych średnio o 15 lat. Wskazuje, że dane popychają osiągnięcia głębokiego uczenia się do przodu i pozostawia czytelnika zastanawiającego się, co mogłoby się stać, gdyby możliwe było zasilenie obecnie dostępnych algorytmów lepszymi danymi pod względem jakości i ilości.

Poprawa szybkości przetwarzania

Kiedy zajrzysz do głębokiego uczenia się, możesz być zaskoczony, że znajdziesz wiele starej technologii, ale co zaskakujące, wszystko działa tak, jak nigdy wcześniej. Ponieważ naukowcy w końcu odkryli, jak sprawić, by niektóre proste, dobre rozwiązania współpracowały ze sobą, duże zbiory danych mogą automatycznie filtrować, przetwarzać i przekształcać dane. Na przykład nowe aktywacje takie jak ReLU nie są wcale takie nowe; są znane od perceptronu. Funkcje rozpoznawania obrazów, które początkowo sprawiły, że uczenie głębokie stało się tak popularne, również nie są nowe. Początkowo głębokie uczenie nabrało ogromnego tempa dzięki Convolutional Neural Networks (CNN). Odkryta w latach 80. przez francuskiego naukowca Yann LeCun, takie sieci przynoszą teraz zdumiewające rezultaty, ponieważ wykorzystują wiele warstw neuronowych i dużą ilość danych. To samo dotyczy technologii, która pozwala maszynie rozumieć ludzką mowę lub tłumaczyć z jednego języka na inny. W każdym przypadku rozwiązanie opiera się na kilkudziesięcioletniej technologii, do której badacz powrócił i zaczął pracować w nowym paradygmacie głębokiego uczenia się. Jedynym problemem jest to, że całe to przetwarzanie danych wymaga bardzo wielu cykli przetwarzania, więc poniższe sekcje omawiają, jak poprawić szybkość przetwarzania, aby faktycznie można było zobaczyć wynik analizy danych w rozsądnym czasie.

Wykorzystując potężny sprzęt

Wykorzystanie niewiarygodnych ilości danych ma wpływ na dzisiejsze działanie algorytmu. Aby przetworzyć tak wiele danych, naukowcy różnego typu polegają na zwiększonym wykorzystaniu procesorów graficznych i sieci komputerowych, aby szybko uzyskać odpowiedzi. Wraz z równoległością (więcej komputerów umieszczonych w klastrach i działających równolegle) procesory GPU umożliwiają tworzenie większych sieci i skuteczne trenowanie ich na większej ilości danych. W rzeczywistości GPU może wykonywać pewne operacje 70 razy szybciej niż jakikolwiek procesor, co pozwala na skrócenie

czasu trenowania sieci neuronowych z tygodni do dni, a nawet godzin. Artykuł mówi, dlaczego potrzebujesz zarówno procesorów, jak i kart graficznych, aby stworzyć efektywne uczenie głębokie Układy GPU są potężnymi jednostkami obliczeniowymi macierzowymi i wektorowymi niezbędnymi do wstecznej propagacji błędów. Technologie te sprawiają, że treningowe sieci neuronowe są osiągalne w krótszym czasie i są dostępne dla większej liczby osób. Badania otworzyły także świat nowych zastosowań. Sieci neuronowe mogą uczyć się na ogromnych ilościach danych i wykorzystywać duże ilości danych (obrazy, tekst, transakcje i dane z mediów społecznościowych), tworząc modele, które stale działają lepiej, w zależności od przepływu danych, które je zasilasz.

Dokonywanie innych inwestycji

Wielcy gracze, tacy jak Google, Facebook, Microsoft i IBM, zauważyli nowy trend i od 2012 roku zaczęli przejmować firmy i zatrudniać ekspertów w nowych dziedzinach głębokiego uczenia się. Dwóch z tych ekspertów to Geoffrey Hinton, który jest najbardziej znany ze swojej pracy nad zastosowaniem algorytmu wstecznej propagacji błędów w wielowarstwowych sieciach neuronowych, a teraz współpracuje z Google, oraz Yann LeCun, twórca Convolutional Neural Networks, który obecnie prowadzi badania nad sztuczną inteligencją Facebooka. Dzisiaj każdy może uzyskać dostęp do sieci, a ludzie mogą również uzyskać dostęp do narzędzi, które pomagają tworzyć sieci głębokiego uczenia się. Dostęp ten wykracza poza czytanie publicznie dostępnych artykułów naukowych, które wyjaśniają, jak działa głębokie uczenie; zawiera również narzędzia do programowania sieci. We wczesnych dniach głębokiego uczenia naukowcy zbudowali każdą sieć od podstaw, używając języków takich jak C++. Niestety tworzenie aplikacji w tak niskopoziomowym języku ogranicza dostęp do danych kilku dobrze wyszkolonym specjalistom. Dzisiejsze możliwości tworzenia skryptów (na przykład przy użyciu Pythona) są lepsze ze względu na szeroką gamę platform uczenia głębokiego typu open source, takich jak TensorFlow firmy Google lub PyTorch Facebooka . Te struktury umożliwiają replikację najnowszych postępów w uczeniu głębokim za pomocą prostych poleceń.

Wyjaśnianie różnic w uczeniu głębokim od innych form sztucznej inteligencji

Biorąc pod uwagę zakłopotanie bogactw związanych z AI jako całością, takich jak duże ilości danych, nowy i potężny sprzęt obliczeniowy dostępny dla wszystkich oraz mnóstwo prywatnych i publicznych inwestycji, możesz być sceptycznie nastawiony do technologii głębokiego uczenia, która polega na sieci neuronowych, które mają więcej neuronów i ukrytych warstw niż w przeszłości. Głębokie sieci kontrastują z prostszymi, płytszymi sieciami z przeszłości, które zawierały w najlepszym razie jedną lub dwie ukryte warstwy. Wiele rozwiązań, które umożliwiają dziś głębokie uczenie się, wcale nie jest nowych, ale uczenie głębokie wykorzystuje je w nowy sposób. Głębokie uczenie nie jest po prostu rebrandingiem starej technologii, perceptronu, odkrytego w 1957 roku przez Franka Rosenblatta w Cornell Aeronautical Laboratory. Głębokie uczenie działa lepiej ze względu na dodatkowe wyrafinowanie, jakie dodaje dzięki pełnemu wykorzystaniu potężnych komputerów i dostępności lepszych (nie tylko więcej) danych. Głębokie uczenie oznacza również głęboką jakościową zmianę możliwości oferowanych przez technologię wraz z nowymi i zadziwiającymi zastosowaniami. Obecność tych możliwości modernizuje stare, ale dobre sieci neuronowe, przekształcając je w coś nowego. Poniższe sekcje opisują, jak głębokie uczenie się spełnia swoje zadanie.

Dodawanie kolejnych warstw

Możesz się zastanawiać, dlaczego głębokie uczenie się rozkwitło dopiero teraz, gdy technologia wykorzystywana jako podstawa głębokiego uczenia istniała dawno temu. Jak wspomniano wcześniej w tym rozdziale, komputery są dziś potężniejsze, a uczenie głębokie może uzyskać dostęp do ogromnych ilości danych. Jednak odpowiedzi te wskazują tylko na ważne problemy z głębokim uczeniem się w przeszłości, a mniejsza moc obliczeniowa i mniejsza ilość danych nie były jedynymi

przeszkodami nie do pokonania. Do niedawna w uczeniu głębokim wystąpił również kluczowy problem techniczny, który uniemożliwiał sieciom neuronowym posiadanie wystarczającej liczby warstw do wykonywania naprawdę złożonych zadań. Ponieważ może korzystać z wielu warstw, uczenie głębokie może rozwiązywać problemy, które są poza zasięgiem uczenia maszynowego, takie jak rozpoznawanie obrazów, tłumaczenie maszynowe i rozpoznawanie mowy. Sieć neuronowa wyposażona tylko w kilka warstw jest doskonałym uniwersalnym aproksymatorem funkcji, czyli systemem, który może odtworzyć dowolną możliwą funkcję matematyczną. Po wyposażeniu w wiele więcej warstw sieć neuronowa staje się zdolna do tworzenia, wewnątrz swojego wewnętrznego łańcucha multiplikacji macierzy, wyrafinowanego systemu reprezentacji do rozwiązywania złożonych problemów. Aby zrozumieć, jak działa złożone zadanie, takie jak rozpoznawanie obrazu, rozważ następujący proces:

1. System głębokiego uczenia wyszkolony do rozpoznawania obrazów (taki jak sieć zdolna do odróżnienia zdjęć psów od tych przedstawiających koty) określa wagi wewnętrzne, które są w stanie rozpoznać temat obrazu.

2. Po wykryciu każdego pojedynczego konturu i narożnika na obrazie, sieć głębokiego uczenia łączy wszystkie takie podstawowe cechy w złożone cechy charakterystyczne.

3. Sieć dopasowuje takie cechy do idealnej reprezentacji, która dostarcza odpowiedzi.

Innymi słowy, sieć głębokiego uczenia się może odróżnić psy od kotów, używając swoich wewnętrznych wag, aby określić, jak najlepiej wyglądać pies i kot. Następnie używa tych wewnętrznych wag, aby dopasować każdy nowy obraz, który mu dostarczysz. Jednym z najwcześniejszych osiągnięć głębokiego uczenia się, które uświadomiło opinii publicznej jego potencjał, jest koci neuron. Zespół Google Brain, kierowany w tym czasie przez Andrew Ng i Jeffa Deana, połączył 16 000 komputerów, aby obliczyć sieć głębokiego uczenia o ponad miliardzie wag, umożliwiając w ten sposób nienadzorowaną naukę z filmów z YouTube. Sieć komputerowa potrafiła nawet samodzielnie określić, bez żadnej interwencji człowieka, czym jest kot, a naukowcom Google udało się wykopać z sieci reprezentację tego, jak sama sieć oczekiwała od kota. W czasie, gdy naukowcy nie mogli układać więcej warstw w sieci neuronowej ze względu na ograniczenia sprzętu komputerowego, potencjał technologii pozostał pogrzebany, a naukowcy ignorowali sieci neuronowe. Brak sukcesu dołożył się do głębokiego sceptycyzmu, jaki pojawił się wokół technologii podczas ostatniej zimy AI. Jednak tym, co naprawdę uniemożliwiło naukowcom stworzenie czegoś bardziej wyrafinowanego, był problem z zanikającymi gradientami. Zanikający gradient pojawia się, gdy próbujesz przesłać sygnał przez sieć neuronową, a sygnał szybko zanika do wartości zbliżonych do zera; nie może przejść przez funkcje aktywacji. Dzieje się tak, ponieważ sieci neuronowe są mnożeniami łańcuchowymi. Każde mnożenie poniżej zera szybko zmniejsza przychodzące wartości, a funkcje aktywacji wymagają wystarczająco dużych wartości, aby przepuścić sygnał. Im dalej od wyjścia znajdują się warstwy neuronów, tym większe prawdopodobieństwo, że zostaną one zablokowane przed aktualizacjami, ponieważ sygnały są zbyt małe, a funkcje aktywacji je zatrzymają. W konsekwencji Twoja sieć przestaje się uczyć jako całość lub uczy się w niewiarygodnie wolnym tempie. Każda próba złożenia i przetestowania złożonych sieci kończyła się niepowodzeniem, ponieważ algorytm wstecznej propagacji błędów nie mógł zaktualizować warstw bliżej wejścia, co sprawiało, że nauka na podstawie złożonych danych, nawet jeśli takie dane były w danym momencie dostępne, była prawie niemożliwa. Głębokie sieci są dziś możliwe dzięki badaniom naukowców z University of Toronto w Kanadzie, takich jak Geoffrey Hinton, który nalegał na pracę nad neuronami. sieci, nawet jeśli wydawały się być staromodnym podejściem do uczenia maszynowego.

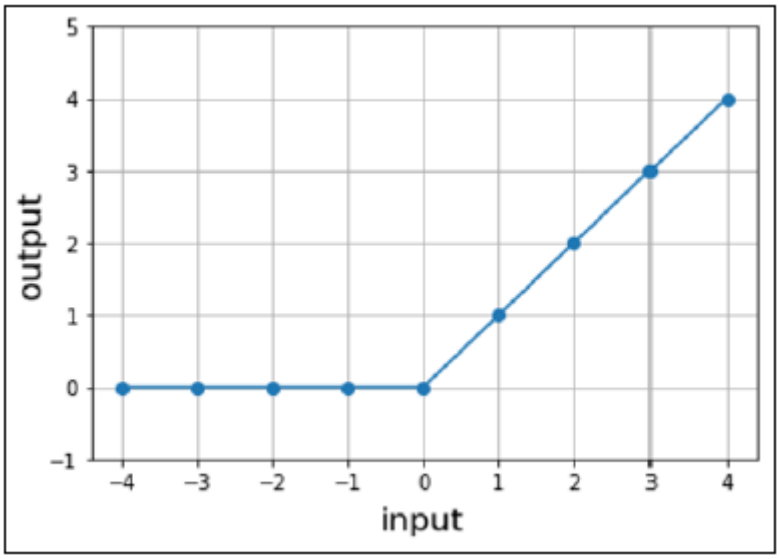
Profesor Hinton, weteran sieci neuronowych (przyczynił się do zdefiniowania algorytmu wstecznej propagacji błędów), wraz z zespołem w Toronto opracowali kilka metod obejścia problemu

zanikających gradientów. Otworzył pole do ponownego przemyślenia nowych rozwiązań, dzięki którym sieci neuronowe ponownie stały się kluczowym narzędziem w uczeniu maszynowym i sztucznej inteligencji. Profesor Hinton i jego zespół zapadają w pamięć również za to, że byli jednymi z pierwszych, którzy testowali użycie GPU w celu przyspieszenia treningu głębokiej sieci neuronowej. W 2012 roku wygrali otwarty konkurs organizowany przez firmę farmaceutyczną Merck and Kaggle (ta ostatnia jest stroną internetową dla konkursów data science), wykorzystując swoje najnowsze odkrycia dotyczące głębokiego uczenia się. To wydarzenie zwróciło dużą uwagę na ich pracę.

Zmiana aktywacji

Zespół Geoffreya Hintona był w stanie dodać więcej warstw do architektury neuronowej dzięki dwóm rozwiązaniom, które zapobiegły problemom z propagacją wsteczną:

- Zapobiega problemowi eksplodujących gradientów, używając inteligentniejszej inicjalizacji sieci. Gradient eksplodujący różni się od gradientu znikającego, ponieważ może spowodować eksplozję sieci, gdy gradient eksplodujący stanie się zbyt duży, aby go obsłużyć. Twoja sieć może eksplodować, chyba że poprawnie zainicjujesz sieć, aby zapobiec obliczaniu dużych liczb wagi. Następnie rozwiązujesz problem zanikania gradientów poprzez zmianę aktywacji sieci.
- Zespół zdał sobie sprawę, że przepuszczanie sygnału przez różne warstwy aktywacyjne ma tendencję do tłumienia sygnału wstecznej propagacji, aż staje się zbyt słaby, aby przejść dalej po zbadaniu, jak działa aktywacja sigmoidalna. Jako rozwiązanie tego problemu użyli nowej aktywacji. Wybór algorytmu do użycia spadł w kierunku starego typu aktywacji ReLU, który oznacza rektyfikowane jednostki liniowe. Aktywacja ReLU zatrzymywała odebrany sygnał, jeśli był poniżej zera, zapewniając nieliniową charakterystykę sieci neuronowych i przepuszczając sygnał tak, jakby był powyżej zera. (Stosowanie tego typu aktywacji jest przykładem połączenia starej, ale wciąż dobrej technologii z obecną). Rysunek pokazuje, jak działa ten proces.

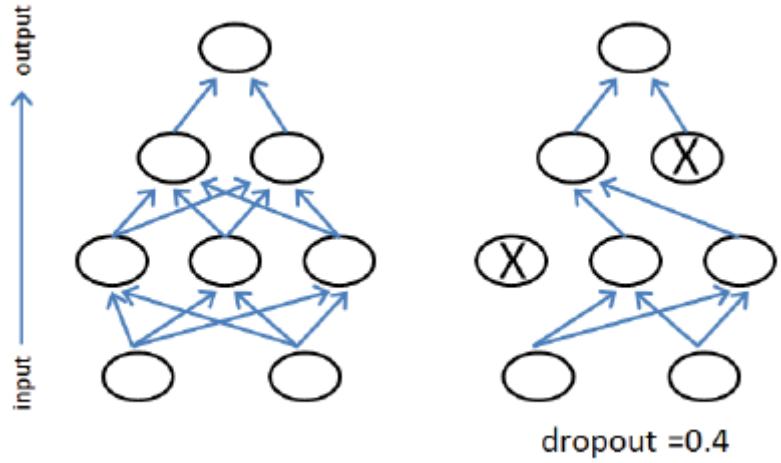


ReLU działał niesamowicie dobrze i pozwolił, aby sygnał wstecznej propagacji dotarł do początkowych głębokich warstw sieci. Gdy sygnał jest dodatni, jego pochodna wynosi 1. Dowód na pochodną ReLU można również znaleźć na rysunku 9-2. Należy zauważyć, że tempo zmian jest stałe i równoważne jednostce, gdy sygnał wejściowy jest dodatni (podczas gdy sygnał jest ujemny, pochodna wynosi 0, co uniemożliwia sygnał

od przejścia). Możesz obliczyć funkcję ReLU używając f(x)=max(0,x). Zastosowanie tego algorytmu znacznie zwiększyło prędkość treningu, umożliwiając szybkie trenowanie jeszcze głębszych sieci bez narażania się na martwe neurony. Martwy neuron to taki, którego sieć nie może aktywować, ponieważ sygnały są zbyt słabe.

Dodawanie regularyzacji przez odejście

Drugie wprowadzenie do głębokiego uczenia opracowane przez zespół Hintona w celu ukończenia wstępnego rozwiązania głębokiego uczenia mającego na celu uregulowanie sieci. Uregulowana sieć ogranicza wagi sieci, co uniemożliwia sieci zapamiętywanie danych wejściowych i uogólnianie obserwowanych wzorców danych. W poprzednich dyskusjach w tym rozdziale zauważono, że pewne neurony zapamiętują określone informacje i zmuszają inne neurony do polegania na tym silniejszym neuronie, powodując, że słabe neurony same rezygnują z uczenia się czegokolwiek użytecznego (sytuacja nazywana koadaptacją). Aby zapobiec koadaptacji, kod tymczasowo wyłącza aktywację losowej części neuronów w sieci. Jak widać z lewej strony rysunku, wagi normalnie działają poprzez pomnożenie ich wejść przez wyjścia dla aktywacji.



Aby wyłączyć aktywację, kod mnoży maskę złożoną z losowej mieszanki jedynek i zer z wynikami. Jeśli neuron zostanie pomnożony przez jeden, sieć przekazuje swój sygnał. Kiedy neuron jest mnożony przez zero, sieć zatrzymuje swój sygnał, zmuszając inne neurony, aby nie polegały na nim w procesie. Dropout działa tylko podczas treningu i nie dotyka żadnej części ciężarków. Po prostu maskuje i ukrywa część sieci, zmuszając niemaskowaną część do odgrywania bardziej aktywnej roli w uczeniu się wzorców danych. W czasie przewidywania dropout nie działa, a wagi są przeskalowywane numerycznie, aby uwzględnić fakt, że nie działały razem podczas treningu.

Znajdowanie jeszcze inteligentniejszych rozwiązań

Głębokie uczenie wpływa na skuteczność sztucznej inteligencji w rozwiązywaniu problemów związanych z rozpoznawaniem obrazu, tłumaczeniem maszynowym i rozpoznawaniem mowy. Problemy te były początkowo rozwiązywane przez klasyczną sztuczną inteligencję i uczenie maszynowe. Ponadto deep learning prezentuje nowe i korzystne rozwiązania w następujących obszarach:

- Uczenie ciągłe z wykorzystaniem nauki online
- Rozwiązania wielokrotnego użytku wykorzystujące transfer learning
- Proste, proste rozwiązania wykorzystujące uczenie od końca do końca

Poniższe sekcje pomogą Ci zrozumieć, na czym polega uczenie się online, uczenie transferowe i uczenie się od końca do końca.

Korzystanie z nauki online

Sieci neuronowe są bardziej elastyczne niż inne algorytmy uczenia maszynowego i mogą nadal trenować, pracując nad tworzeniem prognoz i klasyfikacji. Ta zdolność pochodzi z algorytmów optymalizacji, które umożliwiają sieciom neuronowym uczenie się, które mogą działać wielokrotnie na małych próbkach przykładów (nazywane uczeniem wsadowym) lub nawet na pojedynczych przykładach (nazywane uczeniem online). Sieci uczenia głębokiego mogą budować swoją wiedzę krok po kroku i pozostawać w kontakcie z nowymi informacjami, które mogą nadejść (tak jak umysł dziecka, który jest zawsze otwarty na nowe bodźce i doświadczenia związane z uczeniem się). Na przykład aplikacja do uczenia głębokiego w serwisie społecznościowym może trenować na obrazach kotów. Gdy ludzie publikują zdjęcia kotów, aplikacja rozpoznaje je i oznacza odpowiednią etykietą. Kiedy ludzie zaczną publikować zdjęcia psów w sieci społecznościowej, sieć neuronowa nie musi wznawiać treningu; może kontynuować naukę obrazów psów. Ta funkcja jest szczególnie przydatna przy radzeniu sobie ze zmiennością danych internetowych. Sieć głębokiego uczenia się może być otwarta na nowości i dostosowywać swoje wagi, aby sobie z nimi radzić.

Przenoszenie nauki

Elastyczność jest przydatna nawet po zakończeniu szkolenia sieci, ale należy ją ponownie wykorzystać do celów innych niż początkowe uczenie się. Sieci, które rozróżniają obiekty i poprawnie je klasyfikują, wymagają dużo czasu i dużej mocy obliczeniowej, aby nauczyć się, co robić. Rozszerzenie możliwości sieci na nowe rodzaje obrazów, które nie były częścią poprzedniego uczenia się, oznacza przeniesienie wiedzy do tego nowego problemu (uczenie transferowe). Na przykład możesz przenieść sieć, która jest w stanie rozróżnić psy i koty, aby wykonać pracę polegającą na wykrywaniu potraw z makaronu i sera. Korzystasz z większości warstw sieci takimi, jakimi są (zamrażasz je), a następnie pracujesz nad ostatecznymi warstwami wyjściowymi (dostrajanie). W krótkim czasie i przy mniejszej liczbie przykładów sieć zastosuje to, czego nauczyła się przy rozróżnianiu psów i kotów, do makaronu i sera. Będzie działać nawet lepiej niż sieć neuronowa wyszkolona tylko do rozpoznawania makaronu i sera. Transfer learning jest czymś nowym dla większości algorytmów uczenia maszynowego i otwiera możliwy rynek transferu wiedzy z jednej aplikacji do drugiej, z jednej firmy do drugiej. Google już to robi, faktycznie udostępniając swoje ogromne repozytorium danych, upubliczniając sieci, które na nim zbudowało (jak opisano w tym poście: https://techcrunch.com/2017/06/16/objectdetection-api/). Jest to krok w kierunku demokratyzacji głębokiego uczenia się poprzez umożliwienie każdemu dostępu do jego potencjału.

Nauka od końca do końca

Wreszcie, głębokie uczenie umożliwia uczenie się od końca do końca, co oznacza, że rozwiązuje problemy w łatwiejszy i bardziej bezpośredni sposób niż poprzednie rozwiązania głębokiego uczenia się. Ta elastyczność może mieć większy wpływ na rozwiązywanie problemów. Możesz chcieć rozwiązać trudny problem, na przykład pozwolić sztucznej inteligencji rozpoznawać znane twarze lub prowadzić samochód. Korzystając z klasycznego podejścia AI, trzeba było podzielić problem na łatwiejsze do opanowania podproblemy, aby osiągnąć akceptowalny wynik w wykonalnym czasie. Na przykład, jeśli chciałeś rozpoznać twarze na zdjęciu, poprzednie systemy sztucznej inteligencji podzieliły problem na części w następujący sposób:

1. Znajdź twarze na zdjęciu.

2. Przytnij twarze ze zdjęcia.

3. Przetwórz przycięte twarze, aby uzyskać pozę podobną do zdjęcia z dowodu osobistego.

4. Przekaż przetworzone przycięte twarze jako przykłady do nauki do sieci neuronowej w celu rozpoznawania obrazu.

Dziś możesz przesłać zdjęcie do architektury głębokiego uczenia, poprowadzić je, aby nauczyć się znajdować twarze na obrazach, a następnie użyć architektury głębokiego uczenia do ich klasyfikacji. Możesz użyć tego samego podejścia do tłumaczenia języka, rozpoznawania mowy, a nawet autonomicznych samochodów. We wszystkich przypadkach po prostu przekazujesz dane wejściowe do systemu głębokiego uczenia i uzyskujesz pożądany rezultat.

Wyjaśnienie splotowych sieci neuronowych

Kiedy zajrzysz do głębokiego uczenia się, możesz być zaskoczony, że znajdziesz wiele starych technologii, ale co zdumiewające, wszystko działa tak, jak nigdy dotąd, ponieważ naukowcy w końcu wiedzą, jak sprawić, by niektóre proste, starsze rozwiązania współpracowały ze sobą. W rezultacie big data może automatycznie filtrować, przetwarzać i przekształcać dane. Na przykład nowe aktywacje, takie jak Rectified Linear Units (ReLU), omówione w poprzednich rczęściach, nie są nowe, ale widać, że są używane w nowy sposób. ReLU to funkcja sieci neuronowej, która pozostawia niezmienione wartości dodatnie i zamienia ujemne na zero; pierwszą wzmiankę o ReLU można znaleźć w artykule naukowym Hahnlosera z 2000 roku. Również możliwości rozpoznawania obrazów, które sprawiły, że uczenie głębokie stało się tak popularne kilka lat temu, również nie są nowe. W ostatnich latach uczenie głębokie nabrało ogromnego rozpędu dzięki możliwości zakodowania pewnych właściwości w architekturze za pomocą Convolutional Neural Networks (CNN), które są również nazywane ConvNet. Francuski naukowiec Yann LeCun i inni znani naukowcy opracowali pomysł CNN pod koniec lat 80. i w pełni rozwinęli swoją technologię w latach 90. XX wieku. Ale dopiero teraz, około 25 lat później, takie sieci zaczynają przynosić zdumiewające rezultaty, osiągając nawet lepsze wyniki niż ludzie w poszczególnych zadaniach rozpoznawania. Zmiana nastąpiła, ponieważ możliwe jest skonfigurowanie takich sieci w złożone architektury, które mogą udoskonalić ich naukę na podstawie wielu przydatnych danych. CNN mocno podsyciły niedawny renesans głębokiego uczenia się. W poniższych sekcjach omówiono, w jaki sposób CNN pomagają w wykrywaniu krawędzi i kształtów obrazu w zadaniach takich jak odszyfrowywanie odręcznego tekstu, dokładne lokalizowanie określonego obiektu na obrazie lub oddzielanie różnych części złożonej sceny obrazu.

Rozpoczęcie trasy CNN od rozpoznawania znaków

CNN nie są nowym pomysłem. Pojawiły się pod koniec lat 80. jako rozwiązanie problemów z rozpoznawaniem postaci. Yann LeCun opracował CNN, gdy pracował w AT&T Labs Research wraz z innymi naukowcami, takimi jak Yoshua Bengio, Leon Bottou i Patrick Haffner w sieci o nazwie LeNet5. Zanim zagłębimy się w technologię tych wyspecjalizowanych sieci neuronowych, poświęcimy czas na zrozumienie problemu rozpoznawania obrazów. Cyfrowe obrazy są dziś wszędzie ze względu na wszechobecność aparatów cyfrowych, kamer internetowych i telefonów komórkowych z aparatami. Ponieważ przechwytywanie obrazów stało się tak łatwe, obrazy dostarczają nowy, ogromny strumień danych. Możliwość przetwarzania obrazów otwiera drzwi do nowych zastosowań w dziedzinach takich jak robotyka, autonomiczna jazda, medycyna, bezpieczeństwo i nadzór.

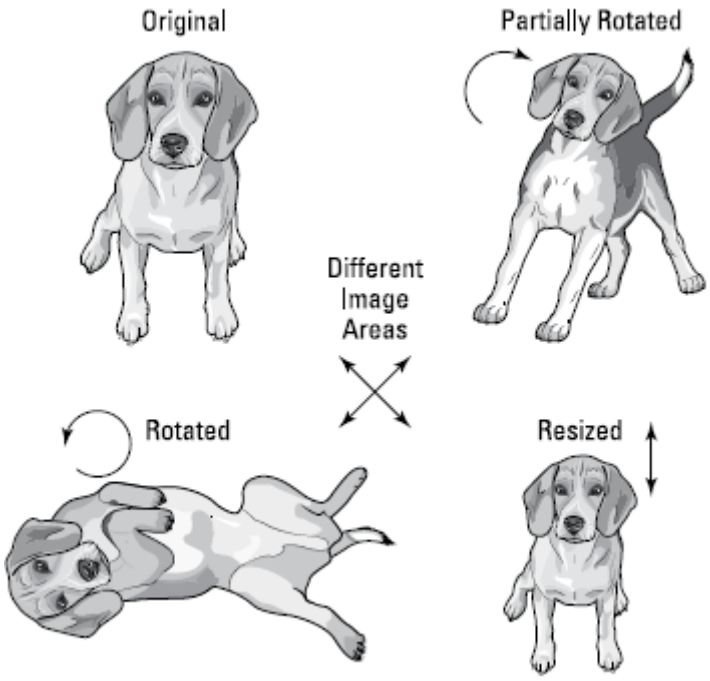
Zrozumienie podstaw obrazu Przetwarzanie obrazu do użytku przez komputer przekształca go w dane. Komputery wysyłają obrazy do monitora jako strumień danych złożony z pikseli, więc obrazy komputerowe są najlepiej reprezentowane jako macierz wartości pikseli, przy czym każda pozycja w macierzy odpowiada punktowi obrazu. Nowoczesne obrazy komputerowe przedstawiają kolory przy użyciu serii 32 bitów (po 8 bitów dla koloru czerwonego, niebieskiego, zielonego i przezroczystości - kanał alfa). Możesz jednak użyć tylko 24 bitów, aby stworzyć obraz w prawdziwym kolorze. Obrazy komputerowe przedstawiają kolory za pomocą trzech nakładających się macierzy, z których każda dostarcza informacji dotyczących jednego z trzech kolorów: czerwonego, zielonego lub niebieskiego (nazywanego również RGB). Mieszanie różnych ilości tych trzech kolorów umożliwia reprezentowanie dowolnego standardowego koloru widocznego dla ludzi, ale nie tych, które widzą ludzie o niezwykłej percepcji. (Większość ludzi może zobaczyć maksymalnie 1 000 000 kolorów, co mieści się w zakresie 16 777 216 kolorów oferowanych przez kolor 24-bitowy. Tetrachromaty mogą zobaczyć 100 000 000 kolorów, więc nie można użyć komputera do analizy tego, co widzą. Ogólnie rzecz biorąc, obraz jest zatem manipulowany przez komputer jako trójwymiarowa macierz składająca się z wysokości, szerokości i liczby kanałów - czyli trzy dla obrazu RGB, ale może być tylko jeden dla obrazu czarno-

białego. (Skala szarości to specjalny rodzaj obrazu RGB, dla którego każdy z trzech kanałów ma ten sam numer. W przypadku obrazu w skali szarości pojedyncza macierz może wystarczyć, ponieważ pojedyncza liczba reprezentuje 256 kolorów w skali szarości, jak pokazano na przykładzie na rysunku. Tu każdy piksel obrazu liczby jest określany ilościowo przez jego wartości macierzy.

255	255	170	34	102	238	255	255
255	255	34	0	85	0	170	255
255	204	0	221	255	68	119	255
255	187	51	255	255	119	119	255
255	170	119	255	255	102	119	255
255	187	68	255	238	51	136	255
255	221	17	170	85	51	255	255
255	255	153	34	85	255	255	255

Biorąc pod uwagę fakt, że obrazy są pikselami (reprezentowanymi jako dane wejściowe numeryczne), praktycy sieci neuronowych początkowo osiągnęli dobre wyniki, łącząc obraz bezpośrednio z siecią neuronową. Każdy piksel obrazu podłączony do węzła wejściowego w sieci. Następnie jedna lub więcej kolejnych warstw ukrytych uzupełniało sieć, w wyniku czego powstała warstwa wyjściowa. Podejście to sprawdziło się w przypadku małych obrazów i rozwiązywania małych problemów, ustępując miejsca różnym podejściom do rozwiązywania problemów z rozpoznawaniem obrazów. Jako alternatywę naukowcy wykorzystali inne algorytmy uczenia maszynowego lub zastosowali intensywne tworzenie funkcji, aby przekształcić obraz w nowo przetworzone dane, które mogą pomóc algorytmom lepiej rozpoznawać obraz. Przykładem tworzenia cech obrazu są Histogramy Zorientowanych Gradientów (HOG), które są obliczeniowym sposobem wykrywania wzorów na obrazie i przekształcania ich w macierz numeryczną.

Praktycy sieci neuronowych stwierdzili, że tworzenie cech obrazu jest wymagające dużej mocy obliczeniowej i często niepraktyczne. Łączenie pikseli obrazu z neuronami było trudne, ponieważ wymagało obliczenia niewiarygodnie dużej liczby parametrów, a sieć nie mogła osiągnąć niezmienności translacji, czyli możliwości rozszyfrowania reprezentowanego obiektu w różnych warunkach wielkości, zniekształcenia lub położenia na obrazie, jak pokazano na rysunku.



Sieć neuronowa, która jest zbudowana z gęstych warstw, jak opisano w poprzednich częściach, może wykrywać tylko obrazy podobne do tych używanych do uczenia - te, które widziała wcześniej - ponieważ uczy się, dostrzegając wzorce w określonych lokalizacjach obrazu. Ponadto sieć neuronowa może popełniać wiele błędów. Przekształcenie obrazu przed przesłaniem go do sieci neuronowej może częściowo rozwiązać problem poprzez zmianę rozmiaru, przesuwanie, czyszczenie pikseli i tworzenie specjalnych fragmentów informacji w celu lepszego przetwarzania sieci. Ta technika, zwana tworzeniem funkcji, wymaga wiedzy na temat niezbędnych przekształceń obrazu, a także wielu obliczeń w zakresie analizy danych. Ze względu na intensywny poziom wymaganej pracy niestandardowej zadania rozpoznawania obrazów są bardziej pracą rzemieślnika niż naukowca. Jednak ilość pracy niestandardowej zmniejszyła się z czasem, ponieważ wzrosła baza bibliotek automatyzujących niektóre zadania.

Wyjaśnienie, jak działają sploty

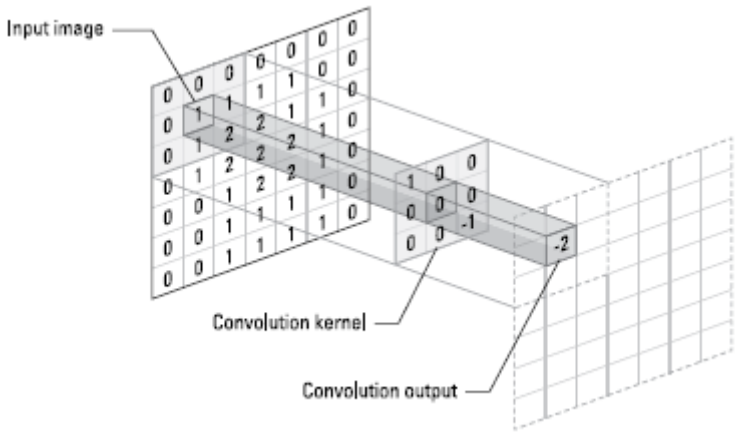
Sploty z łatwością rozwiązują problem niezmienności translacji, ponieważ oferują inne podejście do przetwarzania obrazu w sieci neuronowej. Pomysł powstał z biologicznego punktu widzenia od obserwacji tego, co dzieje się w korze wzrokowej człowieka. Eksperyment z 1962 przeprowadzony przez zdobywców Nagrody Nobla, Davida Huntera Hubela i Torstena Wiesela, wykazał, że tylko niektóre neurony aktywują się w mózgu, gdy oko widzi pewne wzorce, takie jak krawędzie poziome, pionowe lub ukośne. Ponadto obaj naukowcy odkryli, że neurony organizują się pionowo, w hierarchię, co sugeruje, że percepcja wzrokowa opiera się na zorganizowanym udziale wielu pojedynczych, wyspecjalizowanych neuronów. (Możesz dowiedzieć się więcej o tym eksperymencie, czytając artykuł na ttps://knowingneurons.com/2014/10/29/hubel-and-wiesel-heneural-basis-of-visual-perception/.) Sploty po prostu przyjmują to pomysł i, używając matematyki, zastosować go do przetwarzania obrazów w celu zwiększenia zdolności sieci neuronowej do dokładnego rozpoznawania różnych obrazów.

Zrozumienie splotów

Aby zrozumieć, jak działają konwolucje, zaczynasz od danych wejściowych. Wejście to obraz złożony z jednej lub więcej warstw pikseli, zwanych kanałami, a obraz używa wartości od 0, co oznacza, że pojedynczy piksel jest całkowicie wyłączony, do 255, co oznacza, że pojedynczy piksel jest włączony. (Zazwyczaj wartości są przechowywane jako liczby całkowite w celu zaoszczędzenia pamięci.) Jak wspomniano w poprzedniej części tego rozdziału, obrazy RGB mają oddzielne kanały dla kolorów czerwonego, zielonego i niebieskiego. Mieszanie tych kanałów generuje paletę kolorów taką, jaką widzisz na ekranie. Splot działa, operując jednocześnie na małych fragmentach obrazu we wszystkich kanałach obrazu. (Wyobraź sobie kawałek ciasta warstwowego, przy czym każdy kawałek pokazuje wszystkie warstwy). Fragmenty obrazu to po prostu ruchome okno obrazu: okno splotu może być kwadratem lub prostokątem i zaczyna się od lewego górnego rogu obrazu i przesuwa się od lewej do prawej i od góry do dołu. Cała wycieczka po oknie nad obrazem nazywana jest filtrem i oznacza całkowitą transformację obrazu. Należy również zauważyć, że kiedy nowy fragment jest otoczony przez okno, okno przesuwa się o określoną liczbę pikseli; wysokość slajdu nazywa się krokiem. Krok równy 1 oznacza, że okno przesuwa się o jeden piksel w prawo lub w dół; krok 2 oznacza ruch dwóch pikseli; i tak dalej. Za każdym razem, gdy okno splotu przesuwa się do nowej pozycji, następuje proces filtrowania, aby utworzyć część filtra opisanego w poprzednim akapicie. W tym procesie wartości w oknie konwolucji są mnożone przez wartości w jądrze (mała macierz używana do rozmycia, wyostrzania, wypukłości, wykrywania krawędzi i nie tylko - wybierasz jądro, którego potrzebujesz do danego zadania). Jądro ma taki sam rozmiar jak okno konwolucji. Mnożenie każdej części obrazu przez jądro tworzy nową wartość dla każdego piksela, co w pewnym sensie jest nową przetworzoną cechą obrazu. Konwolucja wyprowadza wartość piksela, a kiedy przesuwane okno zakończy objazd po obrazie, obraz został przefiltrowany. W wyniku splotu znajdujesz nowy obraz o następujących cechach:

- Jeśli używasz pojedynczego procesu filtrowania, wynikiem jest przekształcony obraz pojedynczego kanału.
- Jeśli używasz wielu jąder, nowy obraz ma tyle kanałów, ile jest filtrów, z których każdy zawiera specjalnie przetworzone wartości nowych funkcji. Liczba filtrów to głębokość filtra splotu.
- Jeśli użyjesz kroku 1, otrzymasz obraz o takich samych wymiarach jak oryginał.
- Jeśli użyjesz kroków o rozmiarze większym niż 1, wynikowy zawiły obraz będzie mniejszy niż oryginał (krok o rozmiarze dwa oznacza zmniejszenie rozmiaru obrazu o połowę).
- Wynikowy obraz może być mniejszy w zależności od rozmiaru jądra, ponieważ jądro musi rozpocząć i zakończyć swoją podróż na granicach obrazu. Podczas przetwarzania obrazu jądro pochłonie swój rozmiar minus jeden. Na przykład jądro 3 x 3 piksele przetwarzające obraz o wymiarach 7 x 7 pikseli zje 2 piksele z wysokości i szerokości obrazu, a wynikiem splotu będzie wyjście o rozmiarze 5 x 5 pikseli . Masz możliwość uzupełnienia obrazu zerami na granicy (co w istocie oznacza umieszczenie czarnej ramki na obrazie), aby proces splotu nie zmniejszył ostatecznego rozmiaru wyjściowego. Ta strategia nazywa się tym samym dopełnieniem. Jeśli po prostu pozwolisz jądru zmniejszyć rozmiar obrazu początkowego, nazywa się to prawidłowym dopełnieniem.

Przetwarzanie obrazu od dawna opiera się na procesie konwolucji. Filtry konwolucji mogą wykryć krawędź lub uwydatnić pewne cechy obrazu. Rysunek przedstawia przykład niektórych zwojów przekształcających obraz.



Problem z używaniem zwojów polega na tym, że są one wykonane przez człowieka i wymagają wysiłku, aby je rozgryźć. Używając zamiast tego splotu sieci neuronowej, po prostu ustaw następujące opcje:

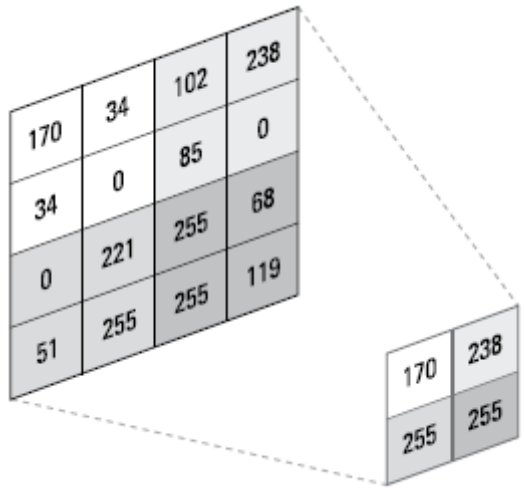
- Liczba filtrów (liczba jąder operujących na obrazie będącym jego kanałami wyjściowymi)
- Rozmiar jądra (ustaw tylko jeden bok dla kwadratu; ustaw szerokość i wysokość dla prostokąta)
- Kroki (zwykle 1- lub 2-pikselowe)
- Czy obraz ma być obramowany czarną obwódką (wybierz prawidłowe wypełnienie lub to samo wypełnienie)

Po określeniu parametrów przetwarzania obrazu proces optymalizacji określa wartości jądra wykorzystywane do przetwarzania obrazu w sposób umożliwiający najlepszą klasyfikację końcowej warstwy wyjściowej. Każdy element macierzy jądra jest zatem neuronem sieci neuronowej i modyfikowany podczas uczenia przy użyciu wstecznej propagacji dla najlepszej wydajności samej sieci. Innym interesującym aspektem tego procesu jest to, że każde jądro specjalizuje się w znajdowaniu określonych aspektów obrazu. Na przykład jądro wyspecjalizowane w filtrowaniu cech typowych dla kotów może znaleźć kota bez względu na to, gdzie się znajduje na obrazie, a jeśli użyjesz wystarczającej liczby jąder, wykryty zostanie każdy możliwy wariant obrazu (przeskalowany, obrócony, przetłumaczony) , czyniąc sieć neuronową wydajnym narzędziem do klasyfikacji obrazów i rozpoznawania granic obrazu, które można łatwo wykryć po zastosowaniu jądra o wymiarach 3 x 3 piksele. To jądro specjalizuje się w znajdowaniu krawędzi, ale inne jądro może wykrywać inne cechy obrazu. Zmieniając wartości w jądrze, tak jak robi to sieć neuronowa podczas wstecznej propagacji, sieć znajduje najlepszy sposób przetwarzania obrazów do celów regresji lub klasyfikacji.

Jądro jest macierzą, której wartości są zdefiniowane przez optymalizację sieci neuronowej, pomnożone przez małą łatkę o tym samym rozmiarze poruszającą się po obrazie, ale może być zamierzona jako warstwa neuronowa, której wagi są dzielone między różne neurony wejściowe. Możesz zobaczyć łatę jako nieruchomą warstwę neuronową połączoną z wieloma częściami obrazu zawsze przy użyciu tego samego zestawu wag. To dokładnie ten sam wynik. Keras oferuje warstwę konwolucyjną Conv2D po wyjęciu z pudełka. Ta warstwa Keras może pobierać dane wejściowe bezpośrednio z obrazu (w krotce musisz ustawić input_shape szerokość, wysokość i liczbę kanałów obrazu) lub z innej warstwy (takiej jak inny splot). Możesz także ustawić filtry, kernel_size, strides i padding, które są podstawowymi parametrami dla każdej warstwy splotowej, jak opisano wcześniej w rozdziale. Podczas ustawiania warstwy Conv2D możesz również ustawić wiele innych parametrów, które w rzeczywistości są zbyt techniczne i mogą nie być konieczne do twoich pierwszych eksperymentów z CNN. Jedyne inne parametry, które mogą być teraz przydatne, to aktywacja, która może dodać wybraną aktywację, oraz nazwa, która określa nazwę warstwy.

Uproszczenie korzystania z poolingu

Warstwy splotowe przekształcają oryginalny obraz przy użyciu różnych rodzajów filtrowania. Każda warstwa znajduje na obrazie określone wzory (szczególne zestawy kształtów i kolorów, dzięki którym obraz jest rozpoznawalny). W miarę postępu tego procesu złożoność sieci neuronowej rośnie, ponieważ liczba parametrów rośnie wraz ze wzrostem liczby filtrów. Aby utrzymać złożoność w zarządzaniu, musisz przyspieszyć filtrowanie i zmniejszyć liczbę operacji. Warstwy puli mogą uprościć dane wyjściowe otrzymane z warstw splotowych, zmniejszając w ten sposób liczbę kolejnych wykonywanych operacji i wykorzystując mniej operacji splotowych do wykonania filtrowania. Działając w sposób podobny do konwolucji (stosując rozmiar okna dla filtra i krok do jego przesunięcia), warstwy puli operują na łatach otrzymywanych danych wejściowych i redukują łatkę do pojedynczej liczby, skutecznie zmniejszając w ten sposób dane przepływające przez sieć neuronowa. Rysunek 10-5 przedstawia operacje wykonywane przez warstwę puli, odbierającą jako dane wejściowe przefiltrowane dane reprezentowane przez lewą macierz 4-x-4: operując na niej przy użyciu okna o rozmiarze 2 piksele i poruszając się o 2 piksele.



W rezultacie warstwa puli generuje właściwy wynik: macierz 2x-2. Sieć stosuje operację łączenia na czterech łatach reprezentowanych przez cztery różnokolorowe części macierzy. Dla każdej poprawki warstwa puli oblicza maksymalną wartość i zapisuje ją jako dane wyjściowe. Bieżący przykład opiera się na warstwie max pooling, ponieważ używa maksymalnej transformacji w oknie przesuwnym. W rzeczywistości masz dostęp do czterech głównych typów warstw puli:

- Maksymalne łączenie
- Średnia pula
- Globalne maksymalne łączenie
- Globalna średnia pula

Ponadto te cztery typy warstw puli mają różne wersje, w zależności od wymiaru danych wejściowych, które mogą przetwarzać:

- Połączenie 1-D: Działa na wektorach. Tak więc łączenie jednowymiarowe jest idealne dla danych sekwencyjnych, takich jak dane czasowe (dane reprezentujące zdarzenia następujące po sobie w czasie) lub tekst (reprezentowane jako sekwencje liter lub słów). Pobiera maksimum lub średnią z sąsiednich części sekwencji.

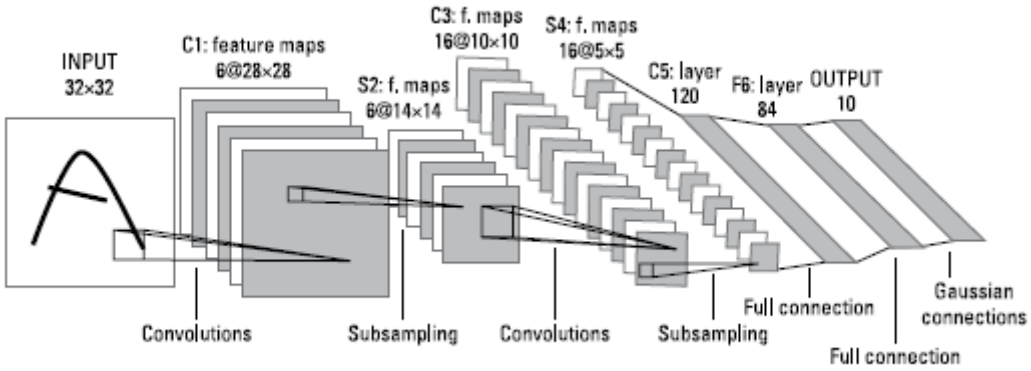
- Pulacje 2-D: Dopasowuje dane przestrzenne, które pasują do macierzy. Możesz użyć puli 2D dla obrazu w skali szarości lub każdego kanału obrazu RBG osobno. Pobiera maksimum lub średnią małych fragmentów (kwadratów) danych.
- Pule 3-D: Dopasowuje dane przestrzenne reprezentowane jako dane przestrzenno-czasowe. Możesz użyć puli 3D dla zdjęć zrobionych w czasie. Typowym przykładem jest zastosowanie obrazowania metodą rezonansu magnetycznego (MRI) do badania lekarskiego. Radiolodzy używają rezonansu magnetycznego do badania tkanek ciała za pomocą pól magnetycznych i fal radiowych.

Opis architektury LeNet

Być może zdziwił Cię opis CNN w poprzedniej sekcji i sposób działania jej warstw (sploty i maksymalne łączenie), ale jeszcze bardziej możesz być zdumiony odkryciem, że nie jest to nowa technologia; zamiast tego pojawił się w latach 90. XX wieku. Poniższe sekcje opisują bardziej szczegółowo architekturę LeNet.

Biorąc pod uwagę podstawową funkcjonalność

Kluczową osobą stojącą za tą innowacją był Yann LeCun, który pracował w AT&T Labs Research jako kierownik działu badań nad przetwarzaniem obrazu. LeCun specjalizował się w optycznym rozpoznawaniu znaków i wizji komputerowej. Yann LeCun to francuski informatyk, który wraz z Léon . stworzył splotowe sieci neuronowe Bottou, Yoshua Bengio i Patricka Haffnera. Obecnie jest głównym naukowcem AI w Facebook AI Research (FAIR) i srebrnym profesorem na New York University (głównie związany z NYU Center for Data Science). Jego osobista strona domowa znajduje się pod adresem http://yann.lecun.com/. Pod koniec lat 90. firma AT&T wdrożyła LeNet5 firmy LeCun, aby odczytywać kody pocztowe dla Poczty Stanów Zjednoczonych. Firma wykorzystała również LeNet5 do czytników czeków do bankomatów, które mogą automatycznie odczytywać kwotę czeku. System nie zawodzi, jak donosi LeCunn. Jednak sukces LeNetu minął w tamtym czasie prawie niezauważony, ponieważ sektor sztucznej inteligencji przeżywał zimę sztucznej inteligencji: zarówno opinia publiczna, jak i inwestorzy byli znacznie mniej zainteresowani i zwracali uwagę na ulepszenia technologii neuronowej niż teraz. Jednym z powodów zimy AI jest to, że wielu badaczy i inwestorów straciło wiarę w ideę, że sieci neuronowe zrewolucjonizują sztuczną inteligencję. Dane z tamtych czasów nie były tak skomplikowane, aby taka sieć działała dobrze. (Bankomaty i USPS były godnymi uwagi wyjątkami ze względu na ilość przetwarzanych danych). Przy braku danych, zwoje tylko nieznacznie przewyższają zwykłe sieci neuronowe złożone z połączonych warstw. Ponadto wielu badaczy osiągnęło wyniki porównywalne z LeNet5, korzystając z zupełnie nowych algorytmów uczenia maszynowego, takich jak Support Vector Machines (SVM) i Random Forests, które były algorytmami opartymi na zasadach matematycznych innych niż te stosowane w sieciach neuronowych. W tamtym czasie posiadanie maszyny zdolnej do odczytywania liczb pisanych na maszynie i odręcznie było nie lada wyczynem. Jak pokazano na rysunku, architektura LeNet5 składa się z dwóch sekwencji warstw splotowych i średnich, które wykonują przetwarzanie obrazu.



Ostatnia warstwa sekwencji jest następnie spłaszczana; to znaczy, że każdy neuron z powstałej serii zawiłych tablic 2-D jest kopiowany do pojedynczej linii neuronów. W tym momencie dwie w pełni połączone warstwy i klasyfikator softmax uzupełniają sieć i zapewniają wynik pod względem prawdopodobieństwa. Sieć LeNet5 jest tak naprawdę podstawą wszystkich kolejnych CNN. Odtworzenie architektury za pomocą Keras wyjaśni ją warstwa po warstwie i zademonstruje, jak zbudować własne sieci konwolucyjne.

Budowanie własnej sieci LeNet5

Sieć ta zostanie przeszkolona na odpowiedniej ilości danych (zestaw cyfr dostarczony przez Keras, składający się z ponad 60 000 przykładów). Dlatego możesz mieć przewagę, jeśli uruchomisz go na Colab, jak wyjaśniono w rozdziale 3, lub na komputerze lokalnym, jeśli masz dostępny GPU. Po otwarciu nowego notatnika zaczynasz od zaimportowania niezbędnych pakietów i funkcji z Keras za pomocą następującego kodu:

```
import keras

import numpy as np

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Conv2D, AveragePooling2D

from keras.layers import Dense, Flatten

from keras.losses import categorical_crossentropy
```

Po zaimportowaniu niezbędnych narzędzi należy zebrać dane:

(X_train, y_train), (X_test, y_test) = mnist.load_data() . Przy pierwszym wykonaniu tego polecenia polecenie mnist pobierze wszystkie dane z Internetu, co może chwilę potrwać. Pobrane dane składają się z jednokanałowych obrazów o wymiarach 28 x 28 pikseli, reprezentujących odręcznie napisane liczby od zera do dziewięciu. W pierwszym kroku musisz przekonwertować zmienną odpowiedzi (y_train dla fazy uczenia i y_test dla testu po zakończeniu modelu) na coś, co sieć neuronowa może zrozumieć i nad czym pracować:

```
num_classes = len(np.unique(y_train))

print(y_train[0], end=' => ')
```

y_train = keras.utils.to_categorical(y_train, 10)

y_test = keras.utils.to_categorical(y_test, 10)

print(y_train[0])

Ten fragment kodu tłumaczy odpowiedź z liczb na wektory liczb, gdzie wartość na pozycji odpowiadającej liczbie odgadniętej przez sieć to 1, a pozostałe to 0. Kod wygeneruje również transformację dla pierwszego przykładu obrazu w pociągu ustawić:

5 => [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]

Zauważ, że dane wyjściowe są oparte na 0, a 1 pojawia się w pozycji odpowiadającej liczbie 5. To ustawienie jest używane, ponieważ sieć neuronowa potrzebuje warstwy odpowiedzi, która jest zbiorem neuronów (stąd wektor), które powinny zostać aktywowane, jeśli udzielona odpowiedź jest poprawna. W tym przypadku widzisz dziesięć neuronów, a w fazie uczenia kod aktywuje poprawną odpowiedź (wartość we właściwej pozycji jest ustawiona na 1) i wyłącza pozostałe (ich wartości wynoszą 0). W fazie testowej sieć neuronowa wykorzystuje swoją bazę przykładów, aby włączyć właściwy neuron lub przynajmniej włączyć więcej niż prawidłowy. W poniższym kodzie kod przygotowuje dane treningowe i testowe:

X_train = X_train.astype(np.float32) / 255

X_test = X_test.astype(np.float32) / 255

img_rows, img_cols = X_train.shape[1:]

X_train = X_train.reshape(len(X_train),

img_rows, img_cols, 1)

X_test = X_test.reshape(len(X_test),

img_rows, img_cols, 1)

input_shape = (img_rows, img_cols, 1)

Liczby pikseli z zakresu od 0 do 255 są przekształcane na wartość dziesiętną z zakresu od 0 do 1. Pierwsze dwa wiersze kodu optymalizują sieć pod kątem poprawnej pracy z dużymi liczbami, które mogą powodować problemy. Kolejne linie zmieniają kształt obrazów, tak aby miały wysokość, szerokość i kanały. Poniższy wiersz kodu definiuje architekturę LeNet5. Zaczynasz od wywołania funkcji sekwencyjnej, która udostępnia pusty model:

lenet = Sequential()

Pierwsza dodana warstwa to warstwa splotowa o nazwie „”:

lenet.add(Conv2D(6, kernel_size=(5, 5), activation='tanh',

input_shape=input_shape, padding='same', name='C1'))

Splot działa z filtrem o rozmiarze 6 (co oznacza, że utworzy sześć nowych kanałów utworzonych przez sploty) i rozmiarze jądra 5 x 5 pikseli. Aktywacja dla wszystkich warstw sieci, z wyjątkiem ostatniej, to tanh (funkcja Hyperbolic Tangent), funkcja nieliniowa, która była najnowocześniejszą funkcją aktywacji w czasie, gdy Yann LeCun stworzył LetNet5. Funkcja jest dziś przestarzała, ale przykład używa jej w celu zbudowania sieci przypominającej oryginalną architekturę LetNet5. Aby wykorzystać taką sieć do

własnych projektów, należy ją zastąpić nowoczesnym ReLU. Przykład dodaje warstwę puli o nazwie S2, która używa jądra 2x-2-pixel:

```
lenet.add(AveragePooling2D(pool_size=(2, 2), strides=(1, 1),

padding='valid'))
```

W tym momencie kod kontynuuje sekwencję, zawsze wykonywaną ze splotem i warstwą puli, ale tym razem przy użyciu większej liczby filtrów:

```
lenet.add(Conv2D(16, kernel_size=(5, 5), strides=(1, 1),

activation='tanh', padding='valid'))

lenet.add(AveragePooling2D(pool_size=(2, 2), strides=(1, 1),

padding='valid'))
```

LeNet5 zamyka się stopniowo, stosując splot ze 120 filtrami. Ten splot nie ma warstwy zbiorczej, ale raczej spłaszczoną warstwę, która rzutuje neurony do ostatniej warstwy splotu jako gęstą warstwę:

```
lenet.add(Conv2D(120, kernel_size=(5, 5), activation='tanh',

name='C5'))

lenet.add(Flatten())
```

Zamknięcie sieci to sekwencja dwóch gęstych warstw, które przetwarzają wyjścia splotu za pomocą aktywacji tanh i softmax. Te dwie warstwy zapewniają końcowe warstwy wyjściowe, w których neurony aktywują sygnał wyjściowy, aby zasygnalizować przewidywaną odpowiedź. Warstwa softmax jest w rzeczywistości warstwą wyjściową określoną przez name='OUTPUT':

```
lenet.add(Dense(84, activation='tanh', name='FC6'))

lenet.add(Dense(10, activation='softmax', name='OUTPUT'))
```

Gdy sieć jest gotowa, do jej skompilowania potrzebny jest Keras. (Za całym kodem Pythona kryje się jakiś kod języka C.) Keras kompiluje go w oparciu o optymalizator SGD:

```
lenet.compile(loss=categorical_crossentropy, optimizer='SGD',

metrics=['accuracy'])

lenet.summary()
```

W tym momencie możesz uruchomić sieć i poczekać, aż przetworzy obrazy:

```
batch_size = 64

epochs = 50

history = lenet.fit(X_train, y_train,

batch_size=batch_size,

epochs=epochs,

validation_data=(X_test,

y_test))
```

Ukończenie cyklu zajmuje 50 epok, a każda epoka przetwarza partie 64 obrazów jednocześnie. (Epoka to jednorazowe przejście całego zestawu danych przez sieć neuronową, podczas gdy partia jest częścią zestawu danych, co w tym przypadku oznacza rozbicie zestawu danych na 64 porcje). W każdej epoce (trwa około 8 sekund, jeśli użyj Colab), możesz monitorować pasek postępu informujący o czasie wymaganym do ukończenia tej epoki. Możesz również przeczytać miary dokładności dla obu zestawów uczących i zestaw testowy (bardziej realistyczny widok). W ostatniej epoce powinieneś przeczytać, że LeNet5 zbudowany w kilku krokach osiąga dokładność 0,989, co oznacza, że na każde 100 odręcznych liczb, które próbuje rozpoznać, sieć powinna poprawnie odgadnąć około 99.

Wykrywanie krawędzi i kształtów z obrazów

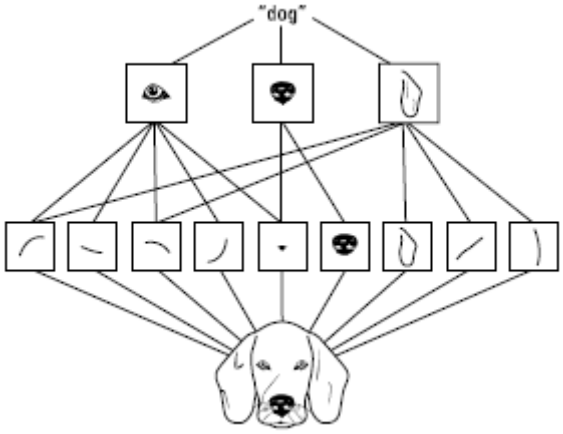
Sploty przetwarzają obrazy automatycznie i działają lepiej niż gęsto połączona warstwa, ponieważ uczą się wzorców obrazu na poziomie lokalnym i mogą je odtworzyć w dowolnej innej części obrazu (cecha zwana niezmiennością translacji). Z drugiej strony, tradycyjne gęste warstwy neuronowe mogą w sztywny sposób określać ogólną charakterystykę obrazu bez korzyści wynikających z niezmienności translacji. To jak różnica między uczeniem się książki poprzez zapamiętywanie tekstu w znaczących kawałkach, a zapamiętywaniem go słowo po słowie. Uczeń (sploty), który nauczył się kawałek po kawałku, może lepiej wyabstrahować treść książki i jest gotowy do zastosowania tej wiedzy w podobnych przypadkach. Uczeń (gęsta warstwa), który nauczył się tego słowo po słowie, stara się wydobyć coś pożytecznego. CNN nie są ani magią, ani czarną skrzynką. Możesz je zrozumieć dzięki przetwarzaniu obrazu i wykorzystać ich funkcjonalność, aby rozszerzyć ich możliwości na nowe problemy. Ta funkcja pomaga rozwiązać szereg problemów z widzeniem komputerowym, które naukowcy zajmujący się danymi uznali za zbyt trudne do złamania przy użyciu starszych strategii.

Wizualizacja zwojów

CNN wykorzystuje różne warstwy do wykonywania określonych zadań w sposób hierarchiczny. Yann LeCun (zobacz sekcję „Rozpoczynanie trasy CNN z rozpoznawaniem znaków" na początku tego rozdziału) zauważył, jak LeNet najpierw przetwarzał krawędzie i kontury, potem motywy, potem kategorie, a na końcu obiekty. Ostatnie badania pokazują, jak naprawdę działają konwolucje:

- Warstwy początkowe: Odkryj krawędzie obrazu
- Warstwy środkowe: wykrywaj złożone kształty (utworzone przez krawędzie)
- Ostateczne warstwy: Odkryj charakterystyczne cechy obrazu charakterystyczne dla typu obrazu, który chcesz sklasyfikować w sieci (na przykład nos psa lub uszy kota)

Ta hierarchia wzorców wykrytych przez sploty wyjaśnia również, dlaczego głębokie sieci splotowe działają lepiej niż płytkie: im więcej jest ułożonych w stos splotów, tym lepiej sieć może uczyć się coraz bardziej złożonych i użytecznych wzorców skutecznego rozpoznawania obrazów. Rysunek pokazuje, jak wszystko działa.



Wizerunek psa jest przetwarzany przez sploty, a pierwsza warstwa chwyta wzory. Druga warstwa przyjmuje te wzory i składa je w kota. Jeśli wzorce przetwarzane przez pierwszą warstwę wydają się zbyt ogólne, aby mogły być przydatne, wzorce ujawnione przez drugą warstwę odtwarzają bardziej charakterystyczne cechy psa, które zapewniają sieci neuronowej przewagę w rozpoznawaniu psów.

Trudność w określeniu, jak działa splot, polega na zrozumieniu, w jaki sposób jądro (macierz liczb) tworzy sploty i jak działają one na łatach obrazu. Kiedy masz wiele zwojów działających jeden po drugim, określenie wyniku poprzez bezpośrednią analizę jest trudne. Jednak technika zaprojektowana do zrozumienia takich sieci tworzy obrazy, które aktywują najwięcej splotów. Kiedy obraz silnie aktywuje określoną warstwę, masz wyobrażenie o tym, co ta warstwa postrzega najbardziej.

Analiza splotów pomaga zrozumieć, jak wszystko działa, zarówno w celu uniknięcia błędów w przewidywaniu, jak i opracowania nowych sposobów przetwarzania obrazów. Na przykład możesz odkryć, że Twoja CNN odróżnia psy od kotów, aktywując je w tle obrazu, ponieważ obrazy, których użyłeś do szkolenia, przedstawiają psy na zewnątrz, a koty w domu. Możesz nawet samodzielnie sprawdzić obrazy, klikając i wskazując warstwy GoogleLeNet, CNN zbudowanej przez Google. Obrazy z wizualizacji funkcji mogą przypominać obrazy deepdream, jeśli miałeś okazję je zobaczyć, gdy były hitem w sieci. To ta sama technika, ale zamiast szukać obrazów, które najbardziej aktywują warstwę, wybierasz warstwę splotową i pozwalasz jej przekształcić obraz. Można również kopiować styl dzieł wielkiego artysty z przeszłości, takiego jak Picasso czy van Gogh, stosując podobną technikę polegającą na wykorzystaniu zwojów do przekształcania istniejącego obrazu, proces zwany artystycznym transferem stylu. Powstały obraz jest nowoczesny, ale styl już nie. Ciekawe przykłady transferu stylu artystycznego można znaleźć w oryginalnym artykule „A Neural Algorithm of Artistic Style" autorstwa Leona Gatysa, Alexandra Eckera i Matthiasa Bethge. Na rysunku oryginalny obraz został przekształcony stylistycznie poprzez zastosowanie cech rysunkowych i kolorystycznych z japońskiego Ukiyo-e „Wielka fala z Kanagawa", drzeworytu wykonanego przez japońskiego artystę Katsushika Hokusai, który żył od 1760 do 1760 roku. 1849.

Prezentacja udanych architektur

W ostatnich latach naukowcy zajmujący się danymi osiągnęli ogromny postęp dzięki głębszemu badaniu działania CNN. Inne metody również przyczyniły się do postępu w działaniu CNN. Konkursy obrazów odegrały ważną rolę, zmuszając naukowców do ulepszania swoich sieci, co umożliwiło udostępnienie dużych ilości obrazów. Proces aktualizacji architektury rozpoczął się podczas ostatniej zimy AI. Fei-Fei Li, profesor informatyki na Uniwersytecie Illinois w Urbana Champaign (a obecnie główny naukowiec w Google Cloud, a także profesor w Stanford) postanowił dostarczyć więcej zestawów danych ze świata rzeczywistego, aby lepiej testować algorytmy sieci neuronowych. Zaczęła gromadzić niesamowitą liczbę obrazów reprezentujących dużą liczbę klas obiektów. Ona i jej zespół wykonali tak ogromne zadanie, korzystając z Amazon Mechanical Turk, usługi, której używasz, aby prosić ludzi o wykonanie dla Ciebie mikrozadań (takich jak klasyfikowanie obrazu) za niewielką opłatą. Powstały zbiór danych, ukończony w 2009 roku, nazywał się ImageNet i początkowo zawierał 3,2 miliona oznaczonych obrazów (obecnie zawiera ponad 10 milionów obrazów) ułożonych w 5247 hierarchicznie zorganizowanych kategorii.

ImageNet wkrótce pojawił się na konkursie w 2010 roku, w którym sieci neuronowe, wykorzystując sploty (stąd odrodzenie i dalszy rozwój technologii opracowanej przez Yanna LeCuna w latach 90.), udowodniły swoją zdolność do poprawnej klasyfikacji obrazów uporządkowanych w 1000 klas. W ciągu siedmiu lat rywalizacji (wyzwanie zamknięte w 2017 r.) zwycięskie algorytmy poprawiły dokładność przewidywania obrazów z 71,8 proc. do 97,3 proc., co przewyższa ludzkie możliwości (ludzie popełniają błędy w klasyfikowaniu obiektów). Oto kilka godnych uwagi architektur CNN, które zostały opracowane na potrzeby konkursu:

- AlexNet (2012): Stworzony przez Alexa Krizhevsky'ego z Uniwersytetu w Toronto. Wykorzystał CNN z filtrem 11 x 11 pikseli, wygrał konkurs i wprowadził wykorzystanie GPU do trenowania sieci neuronowych wraz z aktywacją ReLU w celu kontrolowania overfittingu.
- VGGNet (2014): Pojawił się w dwóch wersjach, 16 i 19. Został stworzony przez Visual Geometry Group na Uniwersytecie Oksfordzkim i zdefiniował nowy standard wielkości filtra 3x-3 dla CNN.
- ResNet (2015): Stworzony przez Microsoft. Ta CNN nie tylko rozszerzyła ideę różnych wersji sieci (50, 101, 152), ale także wprowadziła warstwy pomijane, sposób łączenia głębszych warstw z płytszymi, aby zapobiec problemowi znikającego gradientu (patrz rozdziały 8 i 9, aby uzyskać więcej informacji na temat ten problem) i umożliwiają znacznie głębsze sieci, które są w stanie lepiej rozpoznawać wzorce na obrazach.

Możesz skorzystać ze wszystkich innowacji wprowadzonych przez konkurencję ImageNet, a nawet korzystać z każdej z sieci neuronowych. Ta dostępność pozwala na odtworzenie wydajności sieci widocznej w konkursach i pomyślne rozszerzenie ich o niezliczone inne problemy.

Omówienie transferu uczenia się

Sieci, które rozróżniają obiekty i poprawnie je klasyfikują, wymagają dużej ilości obrazów, długiego czasu przetwarzania i ogromnej mocy obliczeniowej, aby nauczyć się, co robić. Dostosowanie możliwości sieci do nowych typów obrazów, które nie były częścią wstępnego szkolenia, oznacza przeniesienie istniejącej wiedzy do nowego problemu. Ten proces dostosowywania możliwości sieci nazywa się uczeniem transferu, a sieć, którą dostosowujesz, jest często określana jako sieć wstępnie wytrenowana. Nie można zastosować uczenia transferu do innych algorytmów uczenia maszynowego; tylko głębokie uczenie ma możliwość przeniesienia tego, czego się nauczyło na jeden problem, na inny. Transfer learning to coś nowego dla większości algorytmów uczenia maszynowego i otwiera możliwy rynek transferu wiedzy z jednej aplikacji do drugiej oraz z jednej firmy do drugiej. Google już to robi; udostępnia swoje ogromne repozytorium danych, upubliczniając sieci, które zbudował na TF Hub (https://www.tensorflow.org/hub). Na przykład możesz przenieść sieć, która jest w stanie rozróżnić psy i koty, aby wykonać pracę polegającą na wykrywaniu potraw z makaronu i sera. Z technicznego punktu widzenia zadanie to można wykonać na różne sposoby, w zależności od tego, jak podobny jest nowy problem z obrazem do poprzedniego i ile nowych obrazów masz do treningu. (Mały zestaw danych obrazu to kilka tysięcy obrazów, czasem nawet mniej). Jeśli problem z nowym obrazem jest podobny do poprzedniego, sieć może znać wszystkie niezbędne sploty (krawędź, kształt i warstwy obiektów wysokiego poziomu) rozszyfrować podobne obrazy i sklasyfikować je. W takim przypadku nie musisz wprowadzać zbyt wielu obrazów do treningu, dodawać dużej mocy obliczeniowej ani zbyt głęboko dostosowywać wstępnie wytrenowaną sieć. Ten rodzaj transferu jest najczęstszym zastosowaniem uczenia transferu i zwykle stosuje się go wykorzystując sieć przeszkoloną w konkursie ImageNet (ponieważ te sieci zostały przeszkolone na tak wielu obrazach, że prawdopodobnie masz wszystkie sploty potrzebne do transferu wiedzy do innych zadań). Powiedzmy, że zadanie, które chcesz rozszerzyć, obejmuje nie tylko dostrzeganie psów na zdjęciach, ale także określanie rasy psa. Korzystasz z większości warstw sieci ImageNet, takich jak VGG16, bez dalszych zmian. W transferze ucząc się, zamrażasz wartości wstępnie wytrenowanych współczynników splotów, aby nie miały na nie wpływu żadne dalsze trenowanie, a sieć nie nadawała się do danych, które posiadasz, jeśli jest ich za mało.

Z nowymi obrazami, następnie trenujesz warstwy wyjściowe ustawione na nowym problemie (proces znany jako dostrajanie). W krótkim czasie i za pomocą zaledwie kilku przykładów sieć zastosuje to, czego nauczyła się przy rozróżnianiu psów i kotów, w odniesieniu do ras psów. Będzie działać nawet lepiej niż sieć neuronowa wyszkolona tylko do rozpoznawania ras psów, ponieważ podczas dostrajania wykorzystuje to, co ma sieć nauczyła się wcześniej z milionów obrazów. Sieć neuronowa zidentyfikuje tylko te obiekty, do których identyfikacji została wyszkolona. W związku z tym, jeśli wyszkolisz CNN w rozpoznawaniu głównych ras psów, takich jak Labrador Retriever lub Husky, CNN nie rozpozna mieszanek tych dwóch ras, takich jak Labsky. Zamiast tego CNN wygeneruje najbliższe dopasowanie na podstawie wewnętrznych wag, jakie wypracuje podczas treningu. Jeśli zadanie, które musisz przenieść do istniejącej sieci neuronowej, różni się od zadania, do którego zostało przeszkolone, czyli odnajdywania dań z makaronu i sera, zaczynając od sieci używanej do identyfikacji psów i kotów, masz kilka opcji:

- Jeśli masz mało danych, możesz zamrozić pierwszą i środkową warstwę wstępnie wytrenowanej sieci i odrzucić ostatnie warstwy, ponieważ zawierają one funkcje wysokiego poziomu, które prawdopodobnie nie są przydatne w przypadku Twojego problemu. Zamiast końcowych zwojów, dodajesz warstwę odpowiedzi odpowiednią do Twojego problemu. Dostrajanie pozwoli ustalić najlepsze współczynniki dla warstwy odpowiedzi, biorąc pod uwagę dostępne wstępnie wytrenowane warstwy splotowe.

- Jeśli masz dużo danych, dodajesz odpowiednią warstwę odpowiedzi do wstępnie wytrenowanej sieci, ale nie zamrażasz warstw splotowych. Używasz wstępnie wytrenowanych wag jako punktu wyjścia i pozwalasz sieci dopasować swój problem w najlepszy sposób, ponieważ możesz trenować na dużej ilości danych.

Pakiet Keras oferuje kilka wstępnie wytrenowanych modeli, których możesz użyć do nauki transferu.

Przedstawiamy rekurencyjne sieci neuronowe

Tu omówiono, w jaki sposób uczenie głębokie może radzić sobie z przepływającymi informacjami. Rzeczywistość nie jest po prostu zmienna, ale zmienia się w sposób progresywny, który można przewidzieć dzięki obserwacji przeszłości. Jeśli zdjęcie jest statyczną migawką chwili, wideo, składające się z sekwencji powiązanych ze sobą obrazów, jest płynną informacją, a film może powiedzieć znacznie więcej niż pojedyncze zdjęcie lub seria zdjęć. Podobnie dla krótkich i długich danych tekstowych (od tweetów do całych dokumentów lub książek) oraz dla wszystkich serii liczbowych, które reprezentują coś, co dzieje się na osi czasu (na przykład seria o sprzedaży produktu lub jakości powietrza w ciągu dnia w Miasto). Opisano serię nowych warstw, sieci rekurencyjne i wszystkie ich ulepszenia, takie jak warstwy LSTM i GRU. Te technologie stoją za najbardziej zdumiewającymi aplikacjami do uczenia głębokiego, z którymi możesz dziś eksperymentować. Często widzisz je używane w telefonie komórkowym lub w domu. Na przykład używasz tego rodzaju aplikacji podczas rozmowy z inteligentnymi głośnikami, takimi jak Siri, Google Home lub Alexa. Inna aplikacja tłumaczy Twoją rozmowę na inny język za pomocą Tłumacza Google. Za każdą z tych technologii kryje się odrębna architektura neuronowa i dane specyficzne dla aplikacji używane do szkolenia - niektóre publiczne, a niektóre zastrzeżone. Nawet przy tych różnicach w źródle danych i technice warstwy, które sprawiają, że wszystko jest możliwe, są dokładnie tymi samymi warstwami, które importujesz z Tensor-Flow i Keras.

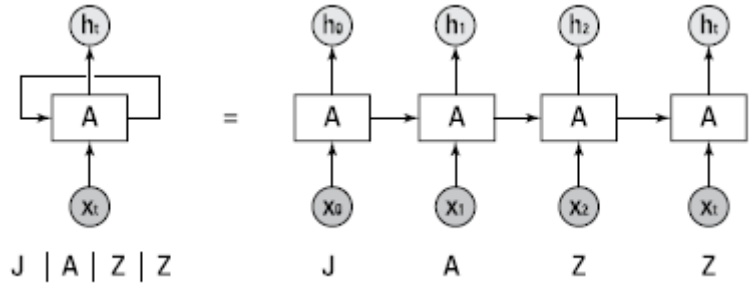
Przedstawiamy sieci rekurencyjne

Sieci neuronowe zapewniają przekształcenie danych wejściowych w pożądane dane wyjściowe. Nawet w przypadku uczenia głębokiego proces jest taki sam, chociaż transformacja jest bardziej złożona. W przeciwieństwie do prostszej sieci neuronowej składającej się z kilku warstw, głębokie uczenie opiera się na większej liczbie warstw do wykonywania złożonych transformacji. Wyjście ze źródła danych łączy się z warstwą wejściową sieci neuronowej, a warstwa wejściowa rozpoczyna przetwarzanie danych. Ukryte warstwy mapują wzorce i wiążą je z konkretnym wynikiem, którym może być wartość lub prawdopodobieństwo. Proces ten doskonale sprawdza się w przypadku każdego rodzaju danych wejściowych, a szczególnie dobrze sprawdza się w przypadku obrazów, jak opisano w rozdziale 10. Po przetworzeniu danych przez każdą warstwę przesyła przekształcone dane do następnej warstwy. Ta następna warstwa przetwarza dane w sposób całkowicie niezależny od warstw poprzednich. Użycie tej strategii oznacza, że jeśli przesyłasz wideo do swojej sieci neuronowej, sieć będzie przetwarzać każdy obraz pojedynczo, jeden po drugim, a wynik w ogóle się nie zmieni, nawet jeśli przetasujesz kolejność dostarczanych obrazów . Prowadząc sieć w taki sposób, używając architektur opisanych w poprzednich rozdziałach tej książki, nie uzyskasz żadnej korzyści z kolejności przetwarzania informacji. Jednak doświadczenie uczy również, że aby zrozumieć proces, czasami trzeba obserwować zdarzenia po kolei. Kiedy wykorzystasz doświadczenie zdobyte na poprzednim etapie, aby zbadać nowy krok, możesz skrócić krzywą uczenia się i skrócić czas i wysiłek potrzebny do zrozumienia każdego kroku.

Modelowanie sekwencji z wykorzystaniem pamięci

Tego rodzaju architektury neuronowe, jakie widzieliśmy do tej pory, nie pozwalają na jednoczesne przetwarzanie sekwencji elementów przy użyciu jednego wejścia. Na przykład, gdy masz serię miesięcznych sprzedaży produktów, uwzględniasz dane dotyczące sprzedaży za pomocą dwunastu danych wejściowych, po jednym na każdy miesiąc, i pozwalasz sieci neuronowej na jednoczesną ich analizę. Wynika z tego, że gdy masz dłuższe sekwencje, musisz dostosować je do większej liczby wejść, a twoja sieć staje się dość duża, ponieważ każde wejście powinno łączyć się z każdym innym wejściem. W efekcie otrzymujesz sieć charakteryzującą się dużą liczbą połączeń (co przekłada się również na wiele wag). Rekurencyjne sieci neuronowe (RNN) są alternatywą dla rozwiązań z poprzednich części, takich

jak perceptron czy CNN. Po raz pierwszy pojawiły się w latach 80., a różni badacze pracowali nad ich ulepszeniem, aż do niedawna zyskały popularność dzięki rozwojowi głębokiego uczenia się i mocy obliczeniowej. Pomysł stojący za RNN jest prosty, badają każdy element sekwencji raz i zachowują o nim pamięć, aby mogli ponownie wykorzystać go podczas badania następnego elementu w sekwencji. Jest to podobne do tego, jak działa ludzki umysł podczas czytania tekstu: osoba czyta tekst litera po literze, ale rozumie słowa, pamiętając każdą literę w słowie. W podobny sposób RNN może powiązać słowo z wynikiem, pamiętając sekwencję otrzymywanych liter. Rozszerzenie tej techniki umożliwia poproszenie RNN o ustalenie, czy fraza jest pozytywna, czy negatywna - jest to szeroko stosowana analiza zwana analizą sentymentu. Sieć łączy pozytywną lub negatywną odpowiedź z pewnymi sekwencjami słów, które widziała w przykładach szkoleniowych. RNN reprezentujesz graficznie jako jednostkę neuronową (znaną również jako komórka), która łączy wejście z wyjściem, ale także łączy się ze sobą, jak pokazano na rysunku. To samopołączenie reprezentuje koncepcję rekurencji, która jest funkcją, która jest stosowana do siebie, dopóki nie osiągnie określonej wartości wyjściowej. Jednym z najczęściej używanych przykładów rekurencji jest obliczanie silni, jak opisano na https://www.geeksforgeeks.org/recursion/. Rysunek przedstawia konkretny przykład RNN używający sekwencji liter do stworzenia słowa jazz. Prawa strona rysunku przedstawia reprezentację zachowania jednostki RNN odbierającej jazz jako dane wejściowe, ale w rzeczywistości jest tylko jedna jednostka, jak pokazano po lewej stronie.



Rysunek pokazuje rekurencyjną komórkę po lewej stronie i rozwija ją jako rozwiniętą serię jednostek, do której trafiają pojedyncze litery słowa jazz po prawej stronie. Zaczyna się od j, po którym następują inne litery. Gdy ten proces zachodzi, RNN emituje dane wyjściowe i modyfikuje swoje wewnętrzne parametry. Modyfikując swoje wewnętrzne parametry, urządzenie uczy się na podstawie danych, które otrzymuje oraz z pamięci poprzednich danych. Sumą tego uczenia się jest stan komórki RNN. Omawiając sieci neuronowe w poprzednich rozdziałach, ta książka mówi wyłącznie o wagach. W przypadku RNN musisz również znać termin stan. Wagi pomagają przetworzyć dane wejściowe na dane wyjściowe w RNN, ale stan zawiera ślady informacji, które RNN widział do tej pory, więc stan wpływa na funkcjonowanie RNN. Stan jest rodzajem pamięci krótkotrwałej, która resetuje się po zakończeniu sekwencji.

Gdy komórka RNN pobiera fragmenty sekwencji, wykonuje następujące czynności:

1. Przetwarza je, zmieniając stan przy każdym wejściu.

2. Emituje dane wyjściowe.

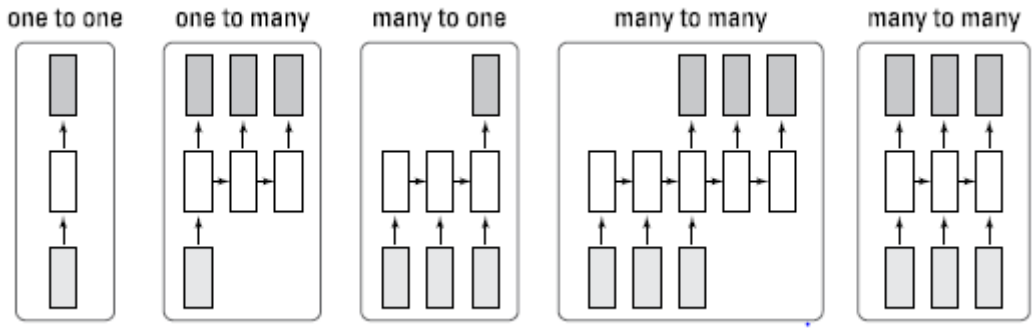
3. Po obejrzeniu ostatniego wyjścia, RNN uczy się najlepszych wag do mapowania wejścia do prawidłowego wyjścia za pomocą wstecznej propagacji.

Rozpoznawanie i tłumaczenie mowy

Zdolność do rozpoznawania i tłumaczenia między językami staje się każdego dnia coraz ważniejsza, ponieważ gospodarki na całym świecie stają się coraz bardziej zglobalizowane. Tłumaczenie językowe to obszar, w którym sztuczna inteligencja ma zdecydowaną przewagę nad ludźmi – do tego stopnia, że niektórzy zaczynają kwestionować, jak długo ludzki tłumacz pozostanie opłacalny. Oczywiście musisz sprawić, by proces tłumaczenia był opłacalny dzięki głębokiemu uczeniu. Z perspektywy architektury neuronowej masz kilka możliwości:

- Zachowaj wszystkie wyjścia dostarczone przez komórkę RNN
- Zachowaj ostatnie wyjście komórki RNN

Ostatnim wyjściem jest wyjście całego RNN, ponieważ powstaje po zakończeniu badania sekwencji. Możesz jednak użyć poprzednich danych wyjściowych, jeśli chcesz przewidzieć inną sekwencję lub zamierzasz ułożyć więcej komórek RNN po bieżącej, na przykład podczas pracy z Convolutional Neural Networks (CNN). Pionowe tyczenie RNN pozwala sieci uczyć się złożonych wzorców sekwencji i efektywniej tworzyć predykcje. RNN można również układać poziomo w tej samej warstwie. Umożliwienie wielu RNN uczenia się na podstawie sekwencji może pomóc im uzyskać więcej z danych. Korzystanie z wielu RNN jest podobne do CNN, w którym każda pojedyncza warstwa wykorzystuje głębokość zwojów, aby poznać szczegóły i wzory z obrazu. W przypadku wielu RNN, warstwa może uchwycić różne niuanse badanej sekwencji. Projektowanie siatek sieci RNN, zarówno w poziomie, jak i w pionie, poprawia wydajność predykcyjną. Jednak decyzja o sposobie korzystania z danych wyjściowych określa, co może osiągnąć architektura głębokiego uczenia zasilana przez RNN. Kluczem jest liczba elementów używanych jako dane wejściowe i długość sekwencji oczekiwana jako dane wyjściowe. Ponieważ sieć uczenia głębokiego synchronizuje wyniki RNN, uzyskujesz pożądany rezultat. W przypadku korzystania z wielu sieci RNN masz kilka możliwości, jak pokazano na rysunku:



- Jeden do jednego: Gdy masz jedno wejście i oczekujesz jednego wyjścia. Przykłady w tej książce do tej pory stosują to podejście. Biorą jeden przypadek, składający się z pewnej liczby zmiennych informacyjnych, i zapewniają oszacowanie, takie jak liczba lub prawdopodobieństwo.
- Jeden do wielu: Tutaj masz jedno wejście i w rezultacie oczekujesz sekwencji wyjść. Sieci neuronowe z automatycznymi napisami wykorzystują następujące podejście: wprowadzasz pojedynczy obraz i tworzysz frazę opisującą zawartość obrazu.
- Wiele do jednego: klasyczny przykład dla sieci RNN. Na przykład wprowadzasz sekwencję tekstową i oczekujesz pojedynczego wyniku jako wyjścia. Widzisz to podejście używane do tworzenia oszacowania analizy nastrojów lub innej klasyfikacji tekstu.
- Wiele do wielu: Podajesz sekwencję jako dane wejściowe i oczekujesz sekwencji wynikowej jako danych wyjściowych. Jest to podstawowa architektura wielu najbardziej imponujących aplikacji wykorzystujących sztuczną inteligencję wykorzystującą głębokie uczenie. To podejście jest stosowane do tłumaczenia maszynowego (takiego jak sieć, która może automatycznie

tłumaczyć frazę z angielskiego na niemiecki), chatbotów (sieć neuronowa, która może odpowiadać na twoje pytania i kłócić się z tobą) oraz etykietowania sekwencji (klasyfikowanie każdego z obrazów w wideo).

Tłumaczenie maszynowe to zdolność maszyny do poprawnego i sensownego tłumaczenia jednego języka ludzkiego na inny. Ta zdolność jest czymś, do czego naukowcy od dawna starali się osiągnąć, zwłaszcza do celów wojskowych. Prawdziwy przełom nastąpił dopiero po tym, jak Google uruchomiło Google Neural Machine Translation (GNMT), o którym można przeczytać więcej na blogu Google AI. GNMT opiera się na serii RNN (przy użyciu paradygmatu wiele do wielu), aby odczytać sekwencję słów w języku, z którego chcesz tłumaczyć (zwanym warstwą kodera) i zwrócić wyniki do innej warstwy RNN (warstwy dekodera), która przekształca go w przetłumaczone wyjście. Neuronowe tłumaczenie maszynowe wymaga dwóch warstw, ponieważ gramatyka i składnia jednego języka mogą się różnić od drugiego. Pojedyncza sieć RNN nie może jednocześnie obsługiwać dwóch systemów językowych, więc do obsługi tych dwóch języków potrzebna jest para koder-dekoder. System nie jest doskonały, ale jest to niesamowity krok naprzód w stosunku do poprzednich rozwiązań opisanych w artykule Wasilija Zubariewa, znacznie redukujący błędy w kolejności słów, błędy leksykalne (wybrany wyraz w tłumaczeniu) i gramatykę (w jaki sposób słowa są używane). Ponadto wydajność zależy od zestawu treningowego, różnic między zaangażowanymi językami i ich specyficznymi cechami. Na przykład ze względu na sposób, w jaki zbudowana jest struktura zdań w języku japońskim, japoński rząd inwestuje obecnie w tłumacza głosu w czasie rzeczywistym, aby pomóc podczas Igrzysk Olimpijskich w Tokio w 2020 r. i pobudzić turystykę poprzez opracowanie zaawansowanego rozwiązania sieci neuronowej. RNN są powodem, dla którego Twój asystent głosowy może Ci odpowiedzieć, a automatyczny tłumacz może dać Ci tłumaczenie na język obcy. Ponieważ RNN to po prostu powtarzająca się operacja mnożenia i sumowania, sieci głębokiego uczenia się nie mogą tak naprawdę zrozumieć żadnego znaczenia; po prostu przetwarzają słowa i frazy na podstawie tego, czego nauczyli się podczas treningu.

Umieszczanie prawidłowego podpisu na zdjęciach

Innym możliwym zastosowaniem RNN wykorzystującym podejście wiele do wielu jest generowanie podpisów, które obejmuje dostarczenie obrazu do sieci neuronowej i otrzymanie opisu tekstowego wyjaśniającego, co dzieje się na obrazie. W przeciwieństwie do chatbotów i tłumaczy maszynowych, których dorobek jest konsumowany przez ludzi, generowanie napisów działa w robotyce. To coś więcej niż tylko generowanie opisów zdjęć lub filmów. Generowanie napisów może pomóc osobom z wadami wzroku postrzegać swoje otoczenie za pomocą urządzeń takich jak Horus do noszenia lub zbudować pomost między obrazami a bazami wiedzy (które są oparte na tekście) dla robotów - umożliwienie im lepszego zrozumienia otoczenia. Zaczynasz od specjalnie opracowanych zbiorów danych, takich jak Pascal Sentence Dataset; Flickr 30K , który składa się z obrazów Flickr z adnotacjami za pomocą crowdsourcingu; lub zbiór danych MS Coco . We wszystkich tych zestawach danych każdy obraz zawiera co najmniej jedną frazę wyjaśniającą treść obrazu. Na przykład w zestawie danych MS Coco numer próbki 5947 widzisz cztery latające samoloty, które możesz poprawnie opisać jako:

- Cztery samoloty na niebie w pochmurny dzień
- Cztery jednosilnikowe samoloty w powietrzu w pochmurny dzień
- Grupa czterech samolotów lecących w szyku
- Grupa samolotów lecących po niebie
- Flota samolotów latających po niebie

Dobrze wyszkolona sieć neuronowa powinna być w stanie generować analogiczne frazy, jeśli zostanie zaprezentowana z podobnym zdjęciem. Google po raz pierwszy opublikował artykuł na temat

rozwiązania tego problemu, nazwany siecią Show and Tell lub Neural Image Caption (NIC), w 2014 r., a następnie zaktualizował go rok później.

Od tego czasu Google udostępnia kartę sieciową typu open source i oferuje ją jako część struktury TensorFlow. Jako sieć neuronowa składa się z przeszkolonej sieci CNN (takiej jak Google LeNet, zwycięzca konkursu ImageNet 2014; szczegółowe informacje można znaleźć w sekcji „Opisywanie architektury LeNet" w rozdziale 10), która przetwarza obrazy podobnie do przekazywania uczenia się. Obraz jest przekształcany w sekwencję wartości reprezentujących cechy obrazu wysokiego poziomu wykryte przez CNN. Podczas uczenia osadzony obraz przechodzi do warstwy RNN, które zapamiętują charakterystykę obrazu w swoim stanie wewnętrznym. CNN porównuje wyniki otrzymane przez RNN ze wszystkimi możliwymi opisami dostarczonymi dla obrazu treningowego i obliczany jest błąd. Następnie błąd powraca do części sieci RNN, aby dostosować wagi RNN i pomóc jej nauczyć się, jak poprawnie opisywać obrazy. Po wielokrotnym powtórzeniu tego procesu przy użyciu różnych obrazów, sieć jest gotowa do obejrzenia nowych obrazów i dostarczenia opisu tych nowych obrazów.

Wyjaśnienie pamięci długoterminowej krótkotrwałej

Wykorzystanie pamięci krótkotrwałej w RNN może wydawać się w stanie rozwiązać każdy możliwy problem głębokiego uczenia się. Jednak RNN nie są całkowicie pozbawione wad. Problem z RNN wynika z ich kluczowej cechy, jaką jest rekurencja tych samych informacji w czasie. Te same informacje, przechodzące wiele razy przez te same komórki, mogą być stopniowo tłumione, a następnie znikać, jeśli waga komórek jest zbyt mała. Jest to tak zwany problem znikającego gradientu, kiedy sygnał korygujący błędy propagowany wstecznie znika po przejściu przez sieć neuronową. Ze względu na problem znikającego gradientu nie można układać zbyt wielu warstw RNN, ponieważ ich aktualizacja staje się trudna. RNN napotykają na jeszcze trudniejsze problemy. W propagacji wstecznej gradient (korekta) dotyczy korekcji błędów, którą wytwarzają sieci podczas prognozowania. Warstwy przed prognozą rozprowadzają gradient do warstw wejściowych i zapewniają prawidłową aktualizację wagi. Warstwy osiągnięte przez małą aktualizację gradientu skutecznie przestają się uczyć. W rzeczywistości, wewnętrznie propagowane wstecznie sygnały RNN mają tendencję do zanikania po kilku rekurencjach, więc sekwencje, które sieć neuronowa aktualizuje i uczy się lepiej, to te najnowsze. Sieć zapomina o wczesnych sygnałach i nie może powiązać wcześniej widzianych sygnałów z nowszymi sygnałami wejściowymi. Dlatego RNN może łatwo stać się zbyt krótkowzroczny i nie można go z powodzeniem zastosować do problemów wymagających dłuższej pamięci. Propagacja wsteczna w warstwie RNN działa zarówno poprzez warstwę w kierunku innych warstw, jak i wewnętrznie, wewnątrz każdej komórki RNN, dostosowując jej pamięć. Niestety, nieważne jak silny jest sygnał, po chwili gradient słabnie i zanika. Krótka pamięć i zanikający gradient utrudniają RNN przyswajanie dłuższych sekwencji. Aplikacje, takie jak napisy do obrazów lub tłumaczenie maszynowe, wymagają dobrej pamięci we wszystkich częściach sekwencji. W związku z tym większość zastosowań wymaga alternatywy, a podstawowe RNN zostały zastąpione różnymi komórkami rekurencyjnymi.

Definiowanie różnic w pamięci

Dwóch naukowców zbadało problem zanikającego gradientu w RNN i opublikowało przełomowy artykuł w 1997 roku, w którym zaproponowano rozwiązanie dla RNN. Sepp Hochreiter, informatyk, który wniósł duży wkład w dziedziny uczenia maszynowego, głębokiego uczenia i bioinformatyki, oraz Jürgen Schmidhuber, pionier w dziedzinie sztucznej inteligencji, opublikowali „Pamięć długotrwała krótkoterminowa". . W artykule przedstawiono nową koncepcję komórki rekurencyjnej, która obecnie służy jako podstawa wszystkich niesamowitych aplikacji do głębokiego uczenia się wykorzystujących sekwencje. Pierwotnie odrzucona, ponieważ była zbyt innowacyjna (wyprzedzała swoje czasy), nowa koncepcja komórki, zaproponowana w artykule, nazwana LSTM (skrót od długiej pamięci

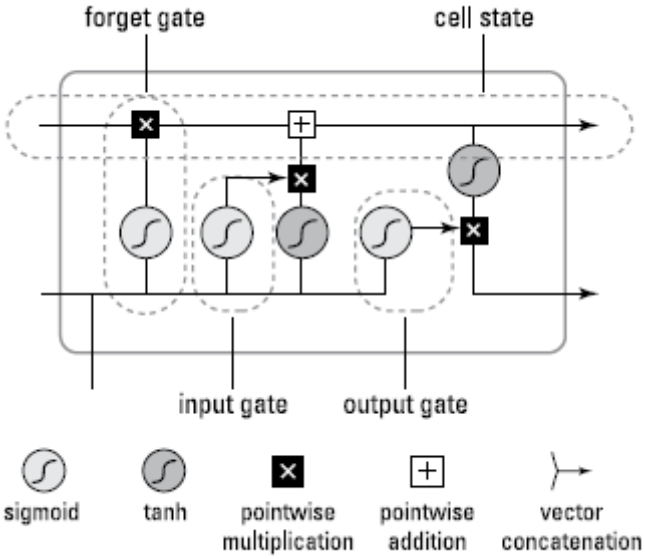
krótkotrwałej) jest obecnie wykorzystywana do wykonywania ponad 4 miliardów operacji neuronowych dziennie. LTSM jest uważany za standard tłumaczenia maszynowego i chatbotów. Google, Apple, Facebook, Microsoft i Amazon opracowały produkty oparte na technologii LSTM opracowanej przez Hochreitera i Schmidhubera. Produkty takie jak inteligentni asystenci głosowi i tłumacze maszynowi działaliby inaczej, gdyby nie wynaleziono LSTM. Podstawową ideą stojącą za LSTM jest rozróżnianie przez RNN stanu na krótko- i długoterminowe. Stan jest pamięcią komórki, a LSTM dzieli się na różne kanały:

- Krótki termin: dane wejściowe mieszają się bezpośrednio z danymi pochodzącymi z sekwencji
- Długotrwały: pobiera z pamięci krótkotrwałej tylko te elementy, które należy zachować przez długi czas

Co więcej, kanał pamięci długotrwałej ma mniej parametrów do dostrojenia. Pamięć długotrwała wykorzystuje pewne dodatki i mnożenia z elementami pochodzącymi z pamięci krótkotrwałej i nic więcej, co czyni ją prawie bezpośrednią autostradą informacyjną. (Znikający gradient nie może zatrzymać przepływu informacji.)

Spacer po architekturze LSTM

LSTM są rozmieszczone wokół bramek, które są wewnętrznymi mechanizmami, które wykorzystują sumowanie, mnożenie i funkcję aktywacji do regulowania przepływu informacji wewnątrz komórki LSTM. Regulując przepływ, bramka może utrzymywać, wzmacniać lub odrzucać informacje, które dotarły z sekwencji zarówno w pamięci krótko-, jak i długoterminowej. Ten przepływ przypomina obwód elektryczny. Rysunek pokazuje wewnętrzną strukturę LSTM.



Różne korzenie i bramy mogą początkowo wydawać się nieco skomplikowane, ale następująca sekwencja kroków pomaga je zrozumieć:

1. Pamięć krótkotrwała pochodząca z poprzedniego stanu (lub z losowych wartości) spotyka nowo wprowadzoną część ciągu i mieszają się razem, tworząc pierwszą derywację.

2. Sygnał pamięci krótkotrwałej, przenoszący zarówno sygnał wychodzący, jak i sygnał nowo wprowadzony, próbuje dotrzeć do pamięci długotrwałej, przechodząc przez bramkę zapominania,

która służy do zapominania pewnych danych. (Technicznie widać rozgałęzienia, w których sygnał jest duplikowany.)

3. Bramka zapominania decyduje, jakie informacje krótkoterminowe powinna odrzucić przed przekazaniem ich do pamięci długotrwałej. Aktywacja esicy anuluje sygnały, które nie są przydatne i wzmacnia to, co zamiast tego wydaje się ważne do zapamiętania.

4. Informacje przechodzące przez bramkę zapominania docierają do kanału pamięci długotrwałej przenoszącego informacje z poprzednich stanów.

5. Wartości pamięci długotrwałej i wyjście z bramki zapominania są mnożone przez siebie.

6. Pamięć krótkotrwała, która nie przeszła przez bramkę zapominania, jest ponownie duplikowana i zajmuje kolejną gałąź; oznacza to, że jedna część przechodzi do bramki wyjściowej, a druga jest skierowana w stronę bramki wejściowej.

7. W bramce wejściowej dane pamięci krótkotrwałej przechodzą oddzielnie przez funkcję sigmoidalną i funkcję tanh. Wyjścia tych dwóch funkcji są następnie najpierw mnożone przez siebie, a następnie dodawane do pamięci długotrwałej. Wpływ na pamięć długotrwałą zależy od esicy, która działa w celu zapomnienia lub zapamiętania, jeśli sygnał zostanie uznany za ważny.

8. Po dodaniu wyjściami z bramki wejściowej pamięć długotrwała nie ulega zmianie. Składająca się z wybranych danych wejściowych z pamięci krótkotrwałej pamięć długotrwała przenosi informacje dłużej w sekwencji i nie reaguje na przerwę czasową między nimi.

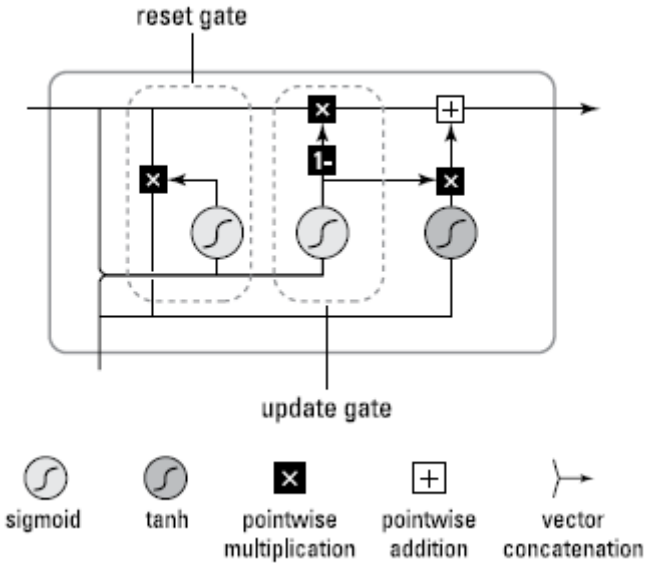
9. Pamięć długotrwała dostarcza informacji bezpośrednio do następnego stanu. Jest również wysyłany do bramki wyjściowej, gdzie zbiega się również pamięć krótkotrwała. Ta ostatnia bramka normalizuje dane z pamięci długotrwałej za pomocą aktywacji tanh i filtruje pamięć krótkotrwałą za pomocą funkcji sigmoidalnej. Oba wyniki są mnożone przez siebie, a następnie przesyłane do kolejnego stanu.

LSTM używają zarówno aktywacji sigmoidalnej, jak i tanh dla swoich bramek. Zasadą, o której należy pamiętać, jest to, że funkcja tanh normalizuje dane wejściowe między –1 a 1, a funkcja sigmoidalna zmniejsza je między 0 a 1. Dlatego, podczas gdy funkcja aktywacji tanh utrzymuje dane wejściowe między możliwym do zastosowania zakresem wartości, sigmoid może przełączać wejście jest wyłączone, ponieważ przesuwa słabsze sygnały w kierunku zera, wygaszając je. Innymi słowy, funkcja esicy pomaga zapamiętywać (wzmacniać sygnał) i zapominać (tłumić sygnał).

Odkrywanie ciekawych wariantów

LSTM ma kilka wariantów, wszystkie wywoływane z dodatkowymi cyframi lub literami w nazwie, takie jak LSTM4, LSTM4a, LSTM5, LSTM5a i LSMT6, aby pokazać, że mają zmodyfikowaną architekturę, chociaż podstawowe koncepcje rozwiązania pozostają. Popularną i istotną modyfikacją, którą można znaleźć w tych wariantach, jest użycie połączeń typu wizjer, które są po prostu potokami danych, które pozwalają wszystkim lub niektórym warstwom bramek na spojrzenie na pamięć długotrwałą (w terminologii RNN stan komórki). Pozwalając na podglądanie pamięci długoterminowej, RNN może podejmować decyzje krótkoterminowe na podstawie wcześniej zaobserwowanych wzorców, które utrwaliły się podczas biegu. Na Keras można znaleźć zwykłą implementację LSTM za pomocą polecenia keras.layers.LSTM (keras.layers.CuDNNLST to wersja GPU), które wystarcza w większości zastosowań. Jeśli chcesz przetestować warianty wizjera, możesz zapoznać się z implementacją TensorFlow, który oferuje więcej opcji na poziomie architektury komórki LSTM. Inny wariant jest bardziej radykalny. Gated Recurrent Units (znane również jako GRU) po raz pierwszy pojawiły się w artykule zatytułowanym „Uczenie się reprezentacji fraz przy użyciu kodera-dekodera RNN do statystycznego

tłumaczenia maszynowego”. GRU działają jako uproszczenie architektury LSTM. W rzeczywistości działają przy użyciu bramek informacyjnych, których parametry można nauczyć się w taki sam sposób, jak LSTM. Ogólnie rzecz biorąc, przepływ informacji w komórce GRU odbywa się drogą liniową, ponieważ GRU używa tylko pamięci roboczej (odpowiednik pamięci długoterminowej w kategoriach LSTM). Ta pamięć robocza jest odświeżana przez bramkę aktualizacji z wykorzystaniem aktualnych informacji dostarczonych do sieci. Zaktualizowane informacje są następnie sumowane ponownie z oryginalną pamięcią roboczą w bramce łączącej te dwa elementy, co nazywa się bramką resetowania, ponieważ wybiera informacje o pamięci roboczej, aby skutecznie zachować pamięć danych zwalnianych do następnej sekwencji kroków. Możesz zobaczyć prosty schemat przepływu na rysunku.



W przeciwieństwie do LSTM, GRU używają bramki resetowania, która zatrzymuje informacje, które powinny zostać zapomniane. GRU używają również bramki aktualizacji, która utrzymuje użyteczne sygnały. GRU mają unikalną pamięć, bez rozróżnienia na długą i krótką. Możesz używać zarówno warstw GRU, jak i LSTM w swoich sieciach bez zbytniej zmiany kodu. Zaimportuj warstwę za pomocą keras.layers.GRU (lub keras.layers. CuDNNGRU dla wersji tylko dla GPU, która opiera się na bibliotece NVIDIA CuDNN;) i współdziałaj z nim jako warstwą LSTM. Jednostki parametrów określa się, definiując liczbę jednostek GRU potrzebnych w jednej warstwie. Przejście z LSTM na GRU zapewnia te korzyści, a także pewne kompromisy:

- GRU traktują sygnały tak jak LSTM i potencjalnie unikają problemu znikającego gradientu, ale nie rozróżniają pamięci długiej i krótkiej, ponieważ opierają się na jednej pamięci roboczej - stan komórki przetwarzany wielokrotnie przez komórkę GRU.
- GRU są mniej złożone niż LSTM, ale są też mniej zdolne do zapamiętywania przeszłych sygnałów, dlatego LSTM mają przewagę, gdy mają do czynienia z dłuższymi sekwencjami.
- GRU trenują szybciej niż LSTM (mają mniej parametrów do dostosowania).
- GRU działają lepiej niż LSTM, gdy masz mniej danych treningowych, ponieważ jest mniej prawdopodobne, że przewyższą otrzymywane informacje.

Uzyskanie niezbędnej uwagi

Czytając o warstwach LSTM i GRU stosowanych do problemów językowych, często uważasz, że mechanizm uwagi jest wymieniany jako najskuteczniejszy sposób rozwiązywania złożonych problemów, takich jak

- Zadanie sieci neuronowej Odpowiedz na pytania
- Frazy klasyfikujące
- Tłumaczenie tekstu z jednego języka na inny

Mechanizm uwagi jest uważany za najnowocześniejsze rozwiązanie do rozwiązywania tych złożonych problemów i pomimo nieobecności w obecnie dostępnych warstwach w pakietach TensorFlow i Keras, znalezienie działającej implementacji open source lub nawet samodzielne zaprogramowanie nie jest trudne. Możesz zacząć tworzyć swój własny mechanizm uwagi, patrząc na implementację open source opracowaną przez Philippe'a Rémy'ego, inżyniera ds. badań, pod adresem. Po raz pierwszy ujawnione w artykule „Neural machine translation by together learning to align and translation" autorstwa Dzmitry Bahdanau, Kyunghyun Cho i Yoshua Bengio w 2014 roku, warstwy uwagi, które implementują mechanizm uwagi to wektory wag wyrażające wagę elementu w zbiorze przetwarzanym przez głęboką sieć neuronową. Często zestaw elementów zawiera sekwencję przetwarzaną przez RNN, ale może to być również obraz. W rzeczywistości warstwa uwagi może rozwiązać dwa rodzaje problemów:

- Podczas przetwarzania długich sekwencji słów, powiązane słowa mogą pojawiać się daleko od siebie w sekwencji. Na przykład zaimki są zazwyczaj trudne do obsługi przez RNN, ponieważ nie mogą powiązać zaimka z elementami przekazanymi wcześniej w sekwencji. Warstwa uwagi może wyróżnić kluczowe elementy we frazie, zanim RNN zacznie przetwarzać sekwencję.
- Podczas przetwarzania dużych obrazów wiele obiektów pojawiających się na obrazie może odwracać uwagę sieci neuronowej od uczenia się prawidłowej klasyfikacji obiektów docelowych. Przykładem jest budowanie sieci do rozpoznawania punktów orientacyjnych na zdjęciach z wakacji. Warstwa uwagi może wykryć, jaką część zdjęcia powinna przetworzyć sieć neuronowa, i zasugerować, aby RNN zignorowała nieistotne elementy, takie jak osoba, pies lub samochód obecne na zdjęciu.

W sieci neuronowej warstwa uwagi jest zwykle umieszczana za warstwą powtarzalną, taką jak LSTM lub GRU. W 2017 roku badacze z Google stworzyli samodzielny mechanizm uwagi, który może działać bez polegania na poprzednich powtarzających się warstwach i który działa znacznie lepiej niż poprzednie rozwiązania. Nazwali taką architekturę Transformer

Wykonywanie klasyfikacji obrazu

Zrozumienie, jak działają warstwy splotowe, jak pokazano w Części 10, to tylko punkt wyjścia. Teoria może tylko wyjaśnić, jak to wszystko działa, ale nie może odpowiednio opisać sukcesu głębokich rozwiązań sieci neuronowych w dziedzinie rozpoznawania obrazów. Ogromna część sukcesu tej technologii, zwłaszcza w zastosowaniach AI, wynika z dostępności odpowiednich danych do trenowania i testowania sieci obrazów, ich zastosowania do różnych problemów dzięki transferowi uczenia się oraz dalszego zaawansowania technologii, która pozwala odpowiadać na złożone pytania o treści obrazu. Tu zagłębisz się w temat wyzwań związanych z klasyfikacją obiektów i wykrywaniem, aby odkryć ich wkład w fundament obecnego renesansu uczenia głębokiego. Konkursy, takie jak te oparte na zbiorze danych ImageNet, nie tylko dostarczają odpowiednich danych do trenowania sieci wielokrotnego użytku do różnych celów (dzięki transferowi uczenia się), ale także zachęcają naukowców do znajdowania inteligentniejszych nowych rozwiązań zwiększających możliwości sieci neuronowej do zrozumienia obrazów. Moduły lokalnej normalizacji odpowiedzi i inicjacji są rozwiązaniami technologicznymi zbyt złożonymi, aby je omawiać tutaj, ale należy mieć świadomość, że są one rewolucyjne. Wszystkie zostały wprowadzone przez sieci neuronowe, które zwyciężyły w konkursie ImageNet: AlexNet (w 2012 r.), GoogleLeNet (w 2014 r.) i ResNet (w 2015 r.). Dzięki zestawowi danych niemieckiego testu porównawczego znaków drogowych, dostarczonemu przez Institute für NeuroInformatik w Ruhr-Universität Bochum w Niemczech, zamykamy przykładem wykorzystania zestawu danych obrazu. Korzystając z zestawu danych, budujesz własną sieć CNN do rozpoznawania znaków drogowych za pomocą powiększania obrazu i ważenia w celu równoważenia częstotliwości różnych klas w przykładach.

Korzystanie z wyzwań związanych z klasyfikacją obrazów

Warstwy CNN do rozpoznawania obrazów zostały po raz pierwszy opracowane przez Yanna LeCuna i zespół naukowców. AT&T faktycznie wdrożyło LeNet5 (sieć neuronową do odręcznych numerów opisaną w rozdziale 10) w czytnikach czeków bankomatów. Jednak wynalazek nie powstrzymał kolejnej zimy AI, która rozpoczęła się w latach 90., kiedy wielu badaczy i inwestorów ponownie straciło wiarę w to, że komputery mogą osiągnąć jakikolwiek postęp w prowadzeniu znaczącej rozmowy z ludźmi, tłumaczenia z różnych języków, rozumienia obrazów i rozumowania w sposób ludzi. W rzeczywistości systemy eksperckie już podkopały zaufanie publiczne. Systemy eksperckie to zestaw automatycznych reguł ustalanych przez ludzi, aby umożliwić komputerom wykonywanie określonych operacji. Niemniej jednak nowa zima AI uniemożliwiła rozwój sieci neuronowych na rzecz różnych rodzajów algorytmów uczenia maszynowego. W tamtych czasach komputery nie miały mocy obliczeniowej i miały pewne ograniczenia, takie jak problem znikającego gradientu. (Rozdział 9 omawia znikający gradient i inne ograniczenia, które uniemożliwiały głębokie architektury neuronowe.) Danym również brakowało wówczas złożoności, a w konsekwencji złożona i rewolucyjna CNN, taka jak LeNet5, która już działała z technologią i ograniczeniami tamtych czasów, miała niewiele okazja do pokazania swojej prawdziwej mocy. Tylko garstka badaczy, takich jak Geoffrey Hinton, Yann LeCun, Jürgen Schmidhuber i Yoshua Bengio, rozwijała technologie sieci neuronowych, dążąc do przełomu, który zakończyłby zimę AI. Tymczasem w 2006 roku Fei-Fei Li, profesor informatyki na Uniwersytecie Illinois Urbana-Champaign (obecnie profesor nadzwyczajny w Stanford, a także dyrektor Stanford Artificial Intelligence Lab i Stanford Vision Lab) podjął wysiłek, aby dostarczać więcej zestawów danych ze świata rzeczywistego, aby lepiej testować algorytmy. Zaczęła gromadzić niesamowitą liczbę obrazów reprezentujących dużą liczbę klas obiektów. Proponowane klasy obejmują różne rodzaje obiektów, zarówno naturalnych (na przykład 120 ras psów), jak i stworzonych przez ludzi (takich jak środki transportu. Korzystając z tego ogromnego zestawu danych obrazu do treningu, naukowcy zauważyli, że ich algorytmy zaczęły działać

lepiej (nic podobnego ,ImageNet istniał w tym czasie), a następnie zaczęli testować nowe pomysły i ulepszać architektury sieci neuronowych.

Zagłębianie się w ImageNet i MS COCO

Można podsumować wpływ i znaczenie konkursu ImageNet (znanego również jako ImageNet Large Scale Visual Recognition Challenge lub ILSVRC; http://image-net. org/challenges/LSVRC/) na rozwój rozwiązań głębokiego uczenia do rozpoznawania obrazów w trzech kluczowych punktach:

- Pomoc w głębokim renesansie sieci neuronowych: Architektura AlexNet CNN (opracowana przez Alexa Krizhevsky′ego Ilyę Sutskevera i Geoffreya Hintona) zwyciężyła w konkursie ILSVRC 2012 z dużą przewagą nad innymi rozwiązaniami.
- Popychanie różnych zespołów naukowców do opracowywania bardziej wyrafinowanych rozwiązań: ILSVRC poprawiło wydajność sieci CNN. VGG16, VGG19, ResNet50, Inception V3, Xception i NASNet to wszystkie sieci neuronowe testowane na obrazach ImageNet, które można znaleźć w pakiecie Keras . Każda architektura reprezentuje ulepszenie w stosunku do poprzednich architektur i wprowadza kluczowe innowacje w zakresie uczenia głębokiego.
- Umożliwienie transferu uczenia się: Konkurs ImageNet pomógł udostępnić zestaw wag, który umożliwił im pracę. 1,2 miliona obrazów szkoleniowych ImageNet, rozmieszczonych w 1000 oddzielnych klasach, pomogło w stworzeniu splotowych sieci, których górne warstwy mogą w rzeczywistości uogólniać problemy inne niż ImageNet.

Ostatnio kilku badaczy zaczęło podejrzewać, że nowsze architektury neuronowe przepełniają zestaw danych ImageNet. W końcu ten sam zestaw testowy był używany od wielu lat do wyboru najlepszych sieci, jak spekulują badacze Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt i Vaishaal. Inni badacze z zespołu Google Brain (Simon Kornblith, Jonathon Shlens i Quoc V.Le) odkryli korelację między dokładnością uzyskaną w ImageNet a wydajnością uzyskaną przez transfer uczenia się tej samej sieci w innych zestawach danych. Opublikowali swoje odkrycia w artykule „Do Better ImageNet Models Transfer Better?”. Co ciekawe, zwrócili również uwagę, że jeśli sieć zostanie przestrojona w ImageNet, mogą wystąpić problemy z uogólnianiem. Dlatego dobrą praktyką jest testowanie uczenia się transferu w oparciu o najnowszą i najlepiej działającą sieć znalezioną w ImageNet, ale nie na tym. Może się okazać, że niektóre mniej wydajne sieci są w rzeczywistości lepsze dla Twojego problemu.

Inne zastrzeżenia dotyczące korzystania z ImageNet polegają na tym, że typowe obrazy w codziennych scenach zawierają więcej obiektów i że obiekty te mogą nie być wyraźnie widoczne, gdy są częściowo zasłonięte innymi obiektami lub ponieważ mieszają się z tłem. Jeśli chcesz używać wstępnie przeszkolonej sieci ImageNet w codziennym kontekście, na przykład podczas tworzenia aplikacji lub robota, wydajność może Cię rozczarować. W konsekwencji, odkąd zakończył się konkurs ImageNet (twierdząc, że poprawa wydajności poprzez kontynuowanie pracy nad zbiorem danych nie byłaby możliwa), badacze coraz bardziej koncentrowali się na wykorzystaniu alternatywnych publicznych zbiorów danych do kwestionowania własnych CNN i ulepszania najnowocześniejszych w rozpoznawaniu obrazu. Oto dotychczasowe alternatywy:

* PASCAL VOC (Visual Object Classes) : pascal/VOC/: Opracowany przez Uniwersytet Oksfordzki, ten zbiór danych wyznacza standard uczenia sieci neuronowych do oznaczania wielu obiektów na tym samym obrazie, standard PASCAL VOC xml. Konkurencja związana z tym zbiorem danych została wstrzymana w 2012 roku.

* SUN https://groups.csail.mit.edu/vision/SUN/: Ten zestaw danych, stworzony przez Massachusetts Institute of Technology (MIT), zawiera testy porównawcze, które pomogą Ci określić wydajność CNN. Nie wiąże się z tym żadna konkurencja.

* MS COCO http://cocodataset.org/: Ten zestaw danych przygotowany przez Microsoft Corporation oferuje szereg aktywnych konkursów.

W szczególności obiekty wspólne firmy Microsoft w zestawie danych kontekstowych (stąd nazwa MS COCO) oferują mniej obrazów szkoleniowych dla modelu niż w ImageNet, ale każdy obraz zawiera wiele obiektów. Ponadto wszystkie obiekty pojawiają się w realistycznych pozycjach (nie inscenizowanych) i ustawieniach (często na świeżym powietrzu i w miejscach publicznych, takich jak drogi i ulice). Aby odróżnić obiekty, zestaw danych zapewnia zarówno kontury we współrzędnych w pikselach, jak i etykiety w standardzie PASCAL VOC XML, przy czym każdy obiekt jest zdefiniowany nie tylko przez klasę, ale także przez jego współrzędne na obrazach (prostokąt obrazu, który pokazuje, gdzie go znaleźć) . Prostokąt ten nazywa się obwiednią, definiowaną w prosty sposób przy użyciu czterech pikseli, w przeciwieństwie do wielu pikseli potrzebnych do zdefiniowania obiektu za pomocą jego konturów. Zestaw danych ImageNet niedawno zaczął oferować, w co najmniej milionie obrazów, wiele obiektów do wykrywania i ich obwiedni.

Nauka magii powiększania danych

Nawet jeśli masz dostęp do dużych ilości danych dla swojego modelu uczenia głębokiego, takich jak zestawy danych ImageNet i MS COCO, może to nie wystarczyć ze względu na mnogość parametrów występujących w większości złożonych architektur neuronowych. W rzeczywistości, nawet jeśli używasz technik takich jak dropout , overfitting jest nadal możliwy. Overfitting występuje, gdy sieć zapamiętuje dane wejściowe i nie uczy się ogólnie użytecznych wzorców danych. Oprócz przerywania, inne techniki, które mogą pomóc w walce z nadmiernym dopasowaniem sieci, to LASSO, Ridge i ElasticNet. Jednak nic nie jest tak skuteczne dla zwiększenia możliwości predykcyjnych sieci neuronowej poprzez dodanie większej liczby przykładów do harmonogramu treningów. Pierwotnie LASSO, Ridge i ElasticNet były sposobami ograniczania wag modelu regresji liniowej, który jest algorytmem statystycznym do obliczania szacunków regresji. W sieci neuronowej działają one w podobny sposób, wymuszając jak najniższą sumę wag w sieci, bez szkody dla poprawności predykcji. LASSO stara się zredukować wiele wag do zera, osiągając w ten sposób wybór najlepszych wag. Natomiast Ridge ma tendencję do tłumienia wszystkich wag, unikając wyższych wag, które mogą generować nadmierne dopasowanie. Wreszcie ElasticNet jest połączeniem podejścia LASSO i Ridge, co stanowi kompromis między strategią selekcji i tłumienia. Augmentacja obrazu stanowi rozwiązanie problemu braku przykładów do zasilania sieci neuronowej w celu sztucznego tworzenia nowych obrazów z już istniejących. Rozszerzanie obrazu składa się z różnych operacji przetwarzania obrazu, które są wykonywane oddzielnie lub łącznie, aby uzyskać obraz inny niż początkowy. Wynik pomaga sieci neuronowej lepiej uczyć się zadania rozpoznawania. Na przykład, jeśli masz obrazy treningowe, które są zbyt jasne lub zbyt rozmyte, przetwarzanie obrazu modyfikuje istniejące obrazy na ciemniejsze i ostrzejsze wersje. Te nowe wersje ilustrują cechy, na których sieć neuronowa musi się skoncentrować, a nie przykłady, które skupiają się na jakości obrazu. Ponadto obracanie, cięcie lub wyginanie obrazu może pomóc, ponieważ ponownie zmuszają sieć do uczenia się przydatnych funkcji obrazu, niezależnie od tego, jak wygląda obiekt. Najczęstsze procedury powiększania obrazu to:

- Flip: Obracanie obrazu wokół jego osi testuje zdolność algorytmu do znalezienia go niezależnie od perspektywy. Ogólny sens Twojego obrazu powinien pozostać nawet po odwróceniu. Niektóre algorytmy nie mogą znaleźć obiektów do góry nogami lub nawet w odbiciu lustrzanym, zwłaszcza jeśli oryginał zawiera słowa lub inne specyficzne znaki.
- Obrót: Obracanie obrazu umożliwia testowanie algorytmu pod pewnymi kątami; symulowanie różnych perspektyw lub nieprecyzyjnie skalibrowanych wizualizacji.

- Przycinanie losowe: przycinanie obrazu zmusza algorytm do skupienia się na komponencie obrazu. Wycięcie obszaru i rozszerzenie go do tego samego rozmiaru co standardowy obraz umożliwia przetestowanie rozpoznawania częściowo ukrytych cech obrazu.
- Przesunięcie koloru: zmiana niuansów kolorów obrazu uogólnia Twój przykład, ponieważ kolory mogą się zmieniać lub być inaczej rejestrowane w rzeczywistym świecie.
- Dodawanie szumu: dodanie losowego szumu testuje zdolność algorytmu do wykrywania obiektu, nawet gdy jakość obiektu jest mniej niż idealna.
- Utrata informacji: losowe usuwanie części obrazu symuluje niedrożność wzroku. Pomaga także sieci neuronowej polegać na ogólnych cechach obrazu, a nie na szczegółach (które można losowo wyeliminować).
- Zmiana kontrastu: zmiana jasności powoduje, że sieć neuronowa jest mniej wrażliwa na warunki oświetleniowe (na przykład światło dzienne lub sztuczne).

Nie musisz specjalizować się w przetwarzaniu obrazu, aby wykorzystać tę potężną technikę powiększania obrazu. Keras oferuje sposób na łatwe włączenie augmentacji do dowolnego szkolenia za pomocą funkcji ImageDataGenerator. Głównym celem ImageDataGenerator jest generowanie partii danych wejściowych do zasilania sieci neuronowej. Oznacza to, że możesz uzyskać dane jako porcje z tablicy NumPy przy użyciu metody .flow. Ponadto nie musisz mieć wszystkich danych treningowych w pamięci, ponieważ metoda .flow_from_directory może je pobrać bezpośrednio z dysku. Ponieważ ImageDataGenerator pobiera partie obrazów, może je przekształcić za pomocą przeskalowania (obrazy składają się z liczb całkowitych, z zakresu od 0 do 255, ale sieci neuronowe działają najlepiej z liczbami zmiennoprzecinkowymi w zakresie od zera do jednego) lub stosując pewne przekształcenia, takie jak:

- Standaryzacja: Uzyskanie wszystkich danych w tej samej skali poprzez ustawienie średniej na zero i odchylenia standardowego na jeden (jako standaryzacja statystyczna), w oparciu o średnią i odchylenie standardowe całego zestawu danych (w ujęciu cech) lub osobno dla każdy obraz (przykładowo).
- Wybielanie ZCA: usuwanie wszelkich zbędnych informacji z obrazu przy zachowaniu oryginalnego podobieństwa obrazu.
- Losowy obrót, losowe przesunięcia i losowe przewroty: Orientacja, przesuwanie i odwracanie obrazu tak, aby obiekty pojawiały się w innej pozie niż oryginał.
- Ponowna kolejność wymiarów: dopasowanie wymiarów danych między obrazami. Na przykład konwertowanie obrazów BGR (format obrazu kolorowego, który był wcześniej popularny wśród producentów aparatów fotograficznych) na standardowe RGB.

Kiedy używasz ImageDataGenerator do przetwarzania partii obrazów, nie jesteś związany rozmiarem pamięci komputera w systemie, ale raczej rozmiarem pamięci (na przykład rozmiarem dysku twardego) i szybkością przesyłania. Możesz nawet uzyskać potrzebne dane w locie z Internetu, jeśli Twoje połączenie jest wystarczająco szybkie. Możesz uzyskać jeszcze bardziej zaawansowane rozszerzenia obrazu, korzystając z pakietu takiego jak albumentations (https://github.com/albu/albumentations). Alexander Buslaev, Alex Parinov, Vladimir I. Iglovikov i Evegene Khvedchenya stworzyli go na podstawie swoich doświadczeń z wieloma wyzwaniami związanymi z wykrywaniem obrazów. Pakiet oferuje niesamowitą gamę możliwych narzędzi do przetwarzania obrazu w zależności od zadania do wykonania i rodzaju używanej sieci neuronowej.

Rozróżnianie znaków drogowych

Po omówieniu podstaw teoretycznych i cech charakterystycznych CNN, możesz spróbować je zbudować. TensorFlow i Keras mogą skonstruować klasyfikator obrazu dla określonego, oddzielonego problemu. Specyficzne problemy nie oznaczają uczenia się wielu różnych funkcji obrazu, aby pomyślnie

wykonać zadanie. Dlatego można je łatwo rozwiązać za pomocą prostych architektur, takich jak LeNet5 (CNN, który zrewolucjonizował rozpoznawanie obrazów neuronowych, omówiony w rozdziale 10) lub coś podobnego. Ten przykład wykonuje interesujące, realistyczne zadanie przy użyciu niemieckiego rozpoznawania znaków drogowych (GTSRB) znaleziony w Instytucie Neuroinformatyki na stronie Ruhr-Universität Bochum. Odczytywanie znaków drogowych jest trudnym zadaniem ze względu na różnice w wyglądzie w rzeczywistych warunkach. GTSRB zapewnia punkt odniesienia do oceny różnych algorytmów uczenia maszynowego zastosowanych do zadania. O budowie tej bazy danych można przeczytać w artykule J. Stallkampa i innych pt. „Man vs. computer: Benchmarking machine learning Algorytmy do rozpoznawania znaków drogowych".

Zbiór danych GTSRB oferuje ponad 50 000 obrazów uporządkowanych w 42 klasach (znaki drogowe), co pozwala na stworzenie problemu klasyfikacji wieloklasowej. W zadaniu z klasyfikacją wieloklasową podajesz prawdopodobieństwo, że obraz należy do klasy i przyjmujesz najwyższe prawdopodobieństwo jako poprawną odpowiedź. Na przykład znak „Uwaga: plac budowy" spowoduje, że algorytm klasyfikacji wygeneruje wysokie prawdopodobieństwa dla wszystkich znaków uwagi. (Największe prawdopodobieństwo powinno odpowiadać swojej klasie.) Rozmycie, rozdzielczość obrazu, różne warunki oświetlenia i perspektywy sprawiają, że zadanie to stanowi wyzwanie dla komputera (a czasem także dla człowieka), jak widać na niektórych przykładach wyodrębnionych z zestaw danych na rysunku.

Przygotowuję dane obrazu

Przykład zaczyna się od skonfigurowania modelu, ustawienia optymalizatora, wstępnego przetwarzania obrazów i utworzenia splotów, puli i gęstych warstw, jak pokazano w poniższym kodzie.

```
import numpy as np

import zipfile

import pprint

from skimage.transform import resize

from skimage.io import imread

import matplotlib.pyplot as plt

% matplotlib inline

import warnings

warnings.filterwarnings("ignore")
```

```
from keras.models import Sequential

from keras.optimizers import Adam

from keras.preprocessing.image import ImageDataGenerator

from keras.utils import to_categorical

from keras.layers import Conv2D, MaxPooling2D

from keras.layers import (Flatten, Dense, Dropout)
```

Zbiór danych zawiera ponad 50 000 obrazów, a powiązana sieć neuronowa może osiągnąć niemal ludzki poziom dokładności rozpoznawania znaków drogowych. Taka aplikacja będzie wymagała dużej ilości obliczeń komputerowych, a uruchomienie tego kodu lokalnie może zająć dużo czasu na twoim komputerze, w zależności od rodzaju posiadanego komputera. Podobnie, Colab może potrwać dłużej, w zależności od zasobów udostępnianych przez Google, w tym od tego, czy faktycznie masz dostęp do procesora graficznego, jak wspomniano w rozdziale 4. Ustawienie czasu tej początkowej aplikacji w konfiguracji pomoże Ci określić, czy Twój komputer lokalny czy Colab to najszybsze środowisko do uruchamiania większych zestawów danych. Jednak najlepsze środowisko to takie, które zapewnia najbardziej spójne i wiarygodne wyniki. Możesz nie mieć dobrego połączenia z Internetem, przez co Colab jest gorszym wyborem. W tym momencie przykład pobiera zbiór danych GTSRB z jego lokalizacji w Internecie (witryna INI Benchmark, na Ruhr-Universität Bochum określonym wcześniej). Poniższy fragment kodu pobiera go do tego samego katalogu, co kod Pythona. Pamiętaj, że proces pobierania może zająć trochę czasu, więc teraz może być dobry moment na uzupełnienie filiżanki.

```
import urllib.request

url = "http://benchmark.ini.rub.de/Dataset/\

GTSRB_Final_Training_Images.zip"

filename = "./GTSRB_Final_Training_Images.zip"

urllib.request.urlretrieve(url, filename)
```

Po pobraniu zestawu danych jako pliku .zip z Internetu kod ustawia rozmiar obrazu. (Wszystkie obrazy są zmieniane na obrazy kwadratowe, więc rozmiar reprezentuje boki w pikselach). Kod ustawia również część danych, które mają być przechowywane do celów testowych, co oznacza wykluczenie niektórych obrazów z uczenia, aby uzyskać bardziej wiarygodną miarę tego, jak działa w sieci. Pętla przez pliki przechowywane w pobranym pliku .zip pobiera poszczególne obrazy, zmienia ich rozmiar, przechowuje etykiety klas i dołącza obrazy do dwóch oddzielnych list: jednej do celów szkoleniowych i jednej do celów testowych. Sortowanie wykorzystuje funkcję mieszającą, która tłumaczy nazwę obrazu na liczbę i na podstawie tej liczby decyduje, gdzie dołączyć obraz.

```
IMG_SIZE = 32

TEST_SIZE = 0.2

X, Xt, y, yt = list(), list(), list(), list()

archive = zipfile.ZipFile(

'./GTSRB_Final_Training_Images.zip', 'r')

file_paths = [file for file in archive.namelist()
```

```
if '.ppm' in file]

for filename in file_paths:

img = imread(archive.open(filename))

img = resize(img,

output_shape=(IMG_SIZE, IMG_SIZE),

mode='reflect')

img_class = int(filename.split('/')[-2])

if (hash(filename) % 1000) / 1000 > TEST_SIZE:

X.append(img)

y.append(img_class)

else:

Xt.append(img)

yt.append(img_class)

archive.close()
```

Po zakończeniu zadania kod zgłasza spójność przykładów nauki i testów.

```
test_ratio = len(Xt) / len(file_paths)

print("Train size:{} test size:{} ({:0.3f})".format(len(X),

len(Xt),

test_ratio))
```

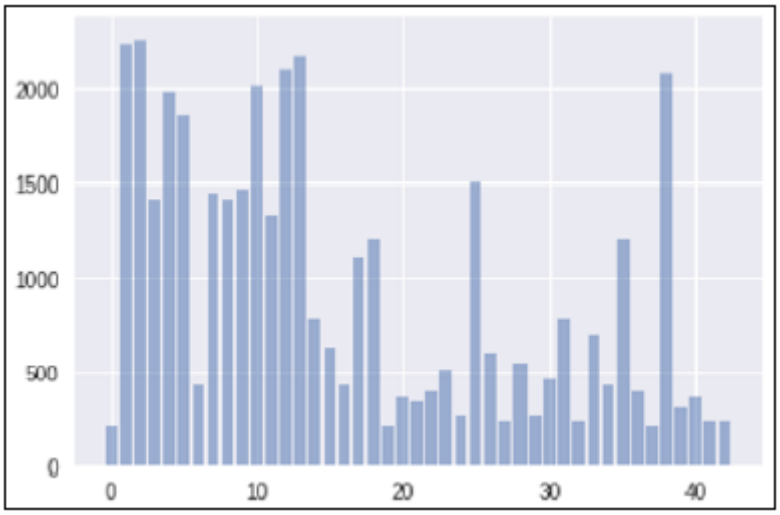
Rozmiar treningu to ponad 30 000 obrazów, a test prawie 8000 (20 procent całości):

Rozmiar treningu: 31344 Rozmiar testowy: 7865 (0,201)

Twoje wyniki mogą się nieco różnić od przedstawionych. Na przykład inny przebieg przykładu dał pociąg o rozmiarze 31 415, a rozmiar testowy o rozmiarze 7794. Sieci neuronowe mogą lepiej uczyć się problemów wieloklasowych, gdy klasy są liczbowo podobne lub mają tendencję do skupiania uwagi na uczeniu się tylko bardziej zaludnionych klas. Poniższy kod sprawdza dystrybucję klas:

```
classes, dist = np.unique(y+yt, return_counts=True)

NUM_CLASSES = len(classes)

print ("No classes:{}".format(NUM_CLASSES))

plt.bar(classes, dist, align='center', alpha=0.5)

plt.show()
```

Rysunek poniższy pokazuje, że klasy nie są zrównoważone. Niektóre znaki drogowe pojawiają się częściej niż inne (na przykład podczas jazdy znaki stopu są spotykane częściej niż znaki dla jeleni).



Jako rozwiązanie kod oblicza wagę, która jest stosunkiem opartym na częstotliwościach klas, które sieć neuronowa wykorzystuje do zwiększenia sygnału, który otrzymuje z rzadszych przykładów i zrzucenia częstszych:

class_weight = {c:dist[c]/np.sum(dist) for c in classes}

Uruchamianie zadania klasyfikacyjnego

Po ustawieniu wag kod definiuje generator obrazów, część kodu, która pobiera obrazy w partiach (próbki o predefiniowanym rozmiarze) w celu trenowania i walidacji, normalizuje ich wartości i stosuje augmentację do walki z nadmiernym dopasowaniem poprzez lekkie przesuwanie i obracanie ich. Zwróć uwagę, że poniższy kod stosuje rozszerzenie tylko w generatorze obrazów szkoleniowych, a nie w generatorze walidacji, ponieważ konieczne jest przetestowanie tylko oryginalnych obrazów.

batch_size = 256

tgen=ImageDataGenerator(rescale=1./255,

rotation_range=5,

width_shift_range=0.10,

height_shift_range=0.10)

train_gen = tgen.flow(np.array(X),

to_categorical(y),

batch_size=batch_size)

vgen=ImageDataGenerator(rescale=1./255)

val_gen = vgen.flow(np.array(Xt),

to_categorical(yt),

batch_size=batch_size)

The code finally builds the neural network:

def small_cnn():

```
model = Sequential()
model.add(Conv2D(32, (5, 5), padding='same',
input_shape=(IMG_SIZE, IMG_SIZE, 3),
activation='relu'))
model.add(Conv2D(64, (5, 5), activation='relu'))
model.add(Flatten())
model.add(Dense(768, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(NUM_CLASSES, activation='softmax'))
return model
model = small_cnn()
model.compile(loss='categorical_crossentropy',
optimizer=Adam(),
metrics=['accuracy'])
```

Sieć neuronowa składa się z dwóch splotów, jeden z 32 kanałami, drugi z 64, obie pracujące z jądrem o rozmiarze (5,5). Po zwojach następuje gęsta warstwa 768 węzłów. Dropout (porzucenie 40 procent węzłów) reguluje tę ostatnią warstwę, a softmax ją aktywuje (więc suma prawdopodobieństw wyjściowych wszystkich klas będzie sumować się do 100 procent).

BIORĄC POD UWAGĘ KOSZTY REALISTYCZNEJ WYDAJNOŚCI

Jak już kilkakrotnie wspomniano, szkolenie głębokiego uczenia się może zająć dużo czasu. Za każdym razem, gdy widzisz w kodzie funkcję dopasowania, taką jak model.fit_generator, prawdopodobnie prosisz system o przeprowadzenie szkolenia. Przykładowy kod zawsze będzie starał się zapewnić realistyczne wyniki - to znaczy to, co naukowiec w prawdziwym świecie uzna za akceptowalne. Niestety, realistyczny wydruk może kosztować zbyt dużo czasu. Nie każdy ma dostęp do najnowszego, zaawansowanego technologicznie systemu i nie każdy dostanie GPU w Colab. Przykład w tym rozdziale w niektórych przypadkach zajmuje dużo czasu na szkolenie. Na przykład testowanie kodu w Colab zajęło nieco ponad 16 godzin, gdy Colab nie dostarczył GPU. Ten sam przykład może działać w ciągu zaledwie godziny, jeśli Colab dostarcza procesor graficzny. (Rozdział 4 zawiera więcej informacji o problemie z procesorem graficznym). Podobnie w przypadku systemu z samym procesorem 16-rdzeniowy system Xeon wymagał 4 godzin i 23 minut do ukończenia szkolenia, ale procesor Intel i7 z 8 rdzeniami wymagał nieco ponad 9 godzin na zrobienie tego samego. Jednym ze sposobów obejścia tego problemu jest zmiana liczby epok używanych do trenowania modelu. Ustawienie epochs=100 użyte w przykładzie w tym rozdziale zapewnia dokładność wyjściową nieco ponad 99 procent. Jeśli jednak czas ma znaczenie, możesz chcieć użyć niższego ustawienia epok podczas uruchamiania tego przykładu, aby skrócić czas oczekiwania na zakończenie przykładu. Inną alternatywą dla uniknięcia problemu jest użycie obsługi GPU na komputerze lokalnym. Aby jednak skorzystać z tej alternatywy, musisz mieć kartę graficzną z odpowiednim rodzajem chipa. Ponieważ konfiguracja jest złożona i prawdopodobnie nie będziesz mieć odpowiedniego procesora graficznego, ta książka dotyczy tylko procesora. Jednak z pewnością możesz zainstalować poprawną obsługę, używając Części 4 jako punktu

wyjścia, a następnie dodając obsługę CUDA. Po stronie optymalizacji stratą do zminimalizowania jest kategoryczna entropia krzyżowa. Kod mierzy sukces na dokładności, która jest procentem poprawnych odpowiedzi udzielonych przez algorytm. (Odpowiedzią jest klasa znaków drogowych o najwyższym przewidywanym prawdopodobieństwie).

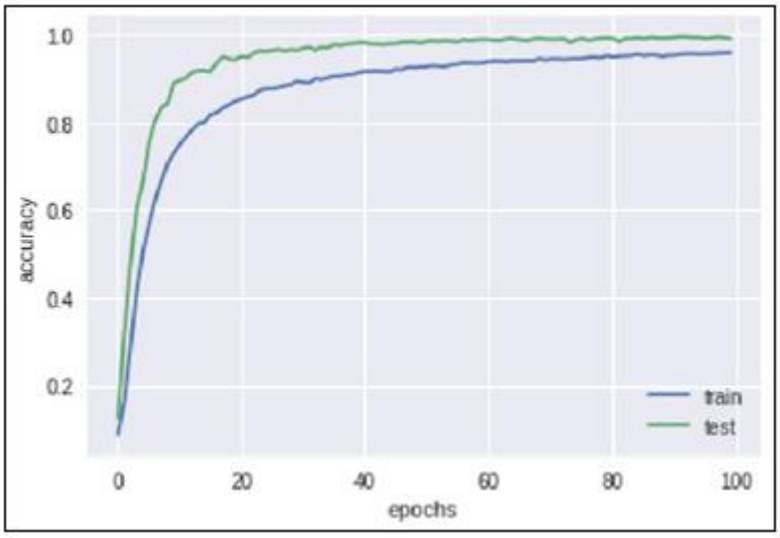
```
history = model.fit_generator(train_gen,

steps_per_epoch=len(X) // batch_size,

validation_data=val_gen,

validation_steps=len(Xt) // batch_size,

class_weight=class_weight,

epochs=100,

verbose=2)
```

Używając fit_generatora w modelu, partie obrazów zaczynają być losowo wyodrębniane, normalizowane i powiększane na potrzeby fazy uczenia. Po wyciągnięciu wszystkich obrazów treningowych kod widzi epokę (iterację treningową z pełnym przebiegiem zestawu danych) i oblicza wynik walidacji na obrazach walidacyjnych. Po przeczytaniu 100 epok trening i model są zakończone.

Jeśli nie korzystasz z żadnego ulepszeń, możesz trenować swój model w około 30 epokach i osiągnąć wydajność swojego modelu prawie porównywalną do umiejętności kierowcy w zakresie rozpoznawania różnych rodzajów znaków drogowych (co daje około 98,8 procent dokładności). . Im bardziej agresywnie stosujesz augmentację, tym więcej epok jest potrzebnych, aby model osiągnął swój najwyższy potencjał, chociaż wydajność dokładności również będzie wyższa. W tym momencie kod tworzy wykres przedstawiający zachowanie dokładności uczenia i walidacji podczas uczenia:

```
print("Best validation accuracy: {:0.3f}"

.format(np.max(history.history['val_acc'])))

plt.plot(history.history['acc'])

plt.plot(history.history['val_acc'])

plt.ylabel('accuracy'); plt.xlabel('epochs')

plt.legend(['train', 'test'], loc='lower right')

plt.show()
```

Kod zgłosi najlepszą zarejestrowaną dokładność walidacji i wykreśli krzywe dokładności uzyskane dla danych dotyczących pociągu i walidacji podczas coraz większych epok uczenia się, jak pokazano na rysunku . Zauważ, że dokładność treningu i walidacji jest prawie podobna pod koniec szkolenia, chociaż walidacja jest zawsze lepsza niż szkolenie. Łatwo to wyjaśnić, ponieważ obrazy walidacyjne są w rzeczywistości „łatwiejsze” do odgadnięcia niż obrazy treningowe, ponieważ nie stosuje się do nich żadnych rozszerzeń.



Biorąc pod uwagę, że kod może inicjować sieć neuronową na różne sposoby, możesz zobaczyć różne najlepsze wyniki na końcu optymalizacji uczenia. Jednak pod koniec 100 epok określonych w kodzie dokładność walidacji powinna przekroczyć 99 procent (próbne przebiegi osiągnęły do 99,5% w Colab). Istnieje różnica między wydajnością uzyskiwaną na danych o pociągu (która często jest mniejsza) i na podzbiorze walidacyjnym, ponieważ dane o pociągu są bardziej złożone i zmienne niż dane walidacyjne, biorąc pod uwagę rozszerzenia obrazu ustawione w kodzie. Powinieneś uznać ten wynik za całkiem doskonały, opierając się na najnowocześniejszych benchmarkach, o których możesz przeczytać w artykule „HALOI, Mrinal. Klasyfikacja znaków drogowych przy użyciu sieci konwolucyjnych opartych na głębokiej incepcji.

Nauka zaawansowanych CNN

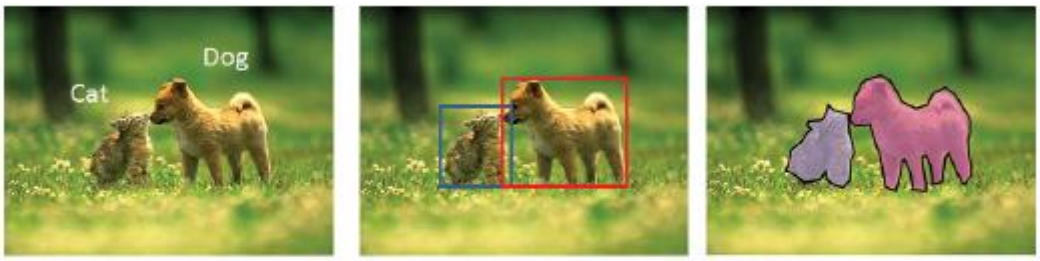
Rozwiązania do głębokiego uczenia się do rozpoznawania obrazów stały się tak imponujące pod względem wydajności na poziomie ludzkim, że można je zobaczyć w opracowywanych lub już wprowadzonych na rynek aplikacjach, takich jak autonomiczne samochody i urządzenia do nadzoru wideo. Urządzenia do monitoringu wizyjnego wykonują już takie zadania, jak automatyczne monitorowanie obrazu satelitarnego, wykrywanie twarzy oraz lokalizacja i liczenie osób. Jednak nie można sobie wyobrazić skomplikowanej aplikacji, w której sieć oznacza obraz tylko jedną prognozą. Nawet prosty wykrywacz psów lub kotów może nie okazać się przydatny, gdy analizowane zdjęcia zawierają wiele psów i kotów. Rzeczywisty świat jest bałaganiarski i złożony. Nie można oczekiwać, z wyjątkiem ograniczonych i kontrolowanych przypadków, obrazów w stylu laboratoryjnym, które składają się z pojedynczych, wyraźnie przedstawionych obiektów. Konieczność obsługi złożonych obrazów utorowała drogę wariantom splotowych sieci neuronowych (CNN). Takie warianty oferują wyrafinowanie, które wciąż jest rozwijane i udoskonalane, takie jak wykrywanie i lokalizacja wielu obiektów. Wykrywanie wielu obiektów może zajmować się jednocześnie wieloma różnymi obiektami. Lokalizacja może wskazać, gdzie się znajdują na obrazie, a segmentacja może znaleźć ich dokładne kontury. Te nowe możliwości wymagają złożonych architektur neuronowych i bardziej zaawansowanego przetwarzania obrazu niż podstawowe sieci CNN omówione w poprzednich rozdziałach. W tym rozdziale przedstawiono podstawy działania tych rozwiązań, wymienia kluczowe podejścia i architektury, a na koniec testuje jedną z najlepiej działających implementacji wykrywania obiektów. Rozdział kończy się ujawnieniem oczekiwanej słabości w skądinąd niewiarygodnej technologii. Ktoś może złośliwie nakłonić CNN do zgłaszania mylących wykryć lub zignorować widziane obiekty przy użyciu odpowiednich technik manipulacji obrazem. To zagadkowe odkrycie otwiera nowy front badawczy, który pokazuje, że wydajność uczenia głębokiego musi również uwzględniać bezpieczeństwo do użytku prywatnego i publicznego.

Rozróżnianie zadań klasyfikacyjnych

Sieci CNN są budulcem rozpoznawania obrazów opartego na głębokim uczeniu, ale odpowiadają tylko na podstawową potrzebę klasyfikacji: mając obraz, mogą określić, czy jego treść może być powiązana z konkretną klasą obrazu, której nauczyliśmy się na poprzednich przykładach. Dlatego, gdy trenujesz głęboką sieć neuronową do rozpoznawania psów i kotów, możesz nakarmić ją zdjęciem i uzyskać dane wyjściowe, które powiedzą ci, czy zdjęcie zawiera psa czy kota. Jeśli ostatnią warstwą sieciową jest warstwa softmax, sieć wyprowadza prawdopodobieństwo, że zdjęcie zawiera psa lub kota (dwie klasy, w rozpoznawaniu których został wyszkolony), a wynik sumuje się do 100 procent. Kiedy ostatnia warstwa jest warstwą aktywowaną sigmoidą, otrzymujesz wyniki, które możesz zinterpretować jako prawdopodobieństwa zawartości należącej do każdej klasy niezależnie. Wyniki niekoniecznie sumują się do 100 procent. W obu przypadkach klasyfikacja może się nie powieść, gdy:

- Głównym obiektem nie jest to, co nauczyłeś rozpoznawać sieć, na przykład prezentując przykładową sieć neuronową ze zdjęciem szopa pracza. W takim przypadku sieć wygeneruje nieprawidłową odpowiedź psa lub kota.
- Główny obiekt jest częściowo zasłonięty. Na przykład Twój kot bawi się w chowanego na zdjęciu, które pokazujesz sieci, a sieć nie może go wykryć.
- Zdjęcie zawiera wiele różnych obiektów do wykrycia, być może w tym zwierzęta inne niż koty i psy. W takim przypadku dane wyjściowe z sieci będą sugerować pojedynczą klasę zamiast obejmować wszystkie obiekty.

Rysunek



Multiple detection | Localization by bounding boxes | Semantic Segmentation

przedstawia obraz 47780 (http://cocodataset.org/#explore?id=47780) pobrany ze zbioru danych MS Coco (wydanego jako część licencji Creative Commons Attribution 4.0 o otwartym kodzie źródłowym). Seria trzech danych wyjściowych pokazuje, w jaki sposób CNN wykryło, zlokalizowało i posegmentowało obiekty pojawiające się na obrazie (kociaka i psa stojącego na polu trawy). Zwykła sieć CNN nie może odtworzyć przykładów z rysunku 13-1, ponieważ jej architektura wyświetli cały obraz jako należący do określonej klasy. Aby przezwyciężyć to ograniczenie, naukowcy rozszerzają podstawowe możliwości sieci CNN, aby były zdolne do następujących czynności:

- Wykrywanie: Określanie, kiedy obiekt jest obecny na obrazie. Wykrywanie różni się od klasyfikacji, ponieważ obejmuje tylko część obrazu, co oznacza, że sieć może wykrywać wiele obiektów tego samego i różnych typów. Możliwość wykrywania obiektów na częściowych obrazach nazywana jest wykrywaniem instancji.
- Lokalizacja: dokładne określenie miejsca, w którym wykryty obiekt pojawia się na obrazie. Możesz mieć różne typy lokalizacji. W zależności od ziarnistości rozróżniają część obrazu zawierającą wykryty obiekt.
- Segmentacja: Klasyfikacja obiektów na poziomie pikseli. Segmentacja doprowadza lokalizację do skrajności. Ten rodzaj modelu neuronowego przypisuje każdy piksel obrazu do klasy, a nawet jednostki. Na przykład sieć zaznacza wszystkie piksele na zdjęciu w stosunku do psów i rozróżnia każdy z nich za pomocą innej etykiety (tzw. segmentacja instancji).

Wykonywanie lokalizacji

Lokalizacja jest prawdopodobnie najłatwiejszym rozszerzeniem, jakie można uzyskać ze zwykłego CNN. Wymaga to trenowania modelu regresora wraz z modelem klasyfikacji uczenia głębokiego. Regresor to model, który zgaduje liczby. Zdefiniowanie położenia obiektu na obrazie jest możliwe przy użyciu współrzędnych piksela narożnika, co oznacza, że można nauczyć sieć neuronową, aby wyprowadzała kluczowe miary, które ułatwiają określenie, gdzie sklasyfikowany obiekt pojawia się na obrazie za pomocą ramki ograniczającej. Zwykle obwiednia używa współrzędnych x i y lewego dolnego rogu, wraz z szerokością i wysokością obszaru, który otacza obiekt.

Klasyfikowanie wielu obiektów

CNN może wykryć (przewidując klasę) i zlokalizować (podając współrzędne) tylko jeden obiekt na obrazie. Jeśli masz wiele obiektów na obrazie, nadal możesz użyć CNN i zlokalizować każdy obiekt obecny na obrazie za pomocą dwóch starych rozwiązań do przetwarzania obrazu:

- Okno przesuwne: analizuje tylko część (nazywaną obszarem zainteresowania) obrazu na raz. Gdy obszar zainteresowania jest wystarczająco mały, prawdopodobnie zawiera tylko jeden obiekt. Mały obszar zainteresowania umożliwia CNN prawidłową klasyfikację obiektu. Ta technika nazywana jest przesuwanym oknem, ponieważ oprogramowanie wykorzystuje okno

obrazu, aby ograniczyć widoczność do określonego obszaru (tak jak robi to okno w domu) i powoli przesuwa to okno wokół obrazu. Ta technika jest skuteczna, ale może wielokrotnie wykryć ten sam obraz lub może się okazać, że niektóre obiekty pozostaną niewykryte na podstawie rozmiaru okna, które zdecydujesz się użyć do analizy obrazów.

- Piramidy obrazu: Rozwiązuje problem korzystania z okna o stałym rozmiarze, ponieważ generuje ono coraz mniejsze rozdzielczości obrazu. Dlatego możesz zastosować małe okno przesuwne. W ten sposób przekształcasz obiekty na obrazie, a jedna z redukcji może dokładnie pasować do użytego okna przesuwnego.

Techniki te są intensywne obliczeniowo. Aby je zastosować, musisz wielokrotnie zmieniać rozmiar obrazu, a następnie podzielić go na kawałki. Następnie przetwarzasz każdą porcję za pomocą klasyfikacji CNN. Liczba operacji dla tych czynności jest tak duża, że renderowanie wyników w czasie rzeczywistym jest niemożliwe. Przesuwane okno i piramida obrazów zainspirowały badaczy głębokiego uczenia się do odkrycia kilku koncepcyjnie podobnych podejść, które są mniej intensywne obliczeniowo. Pierwsze podejście to wykrywanie jednoetapowe. Działa poprzez podzielenie obrazów na siatki, a sieć neuronowa dokonuje prognozy dla każdej komórki siatki, przewidując klasę znajdującego się w niej obiektu. Przewidywanie jest dość przybliżone, w zależności od rozdzielczości siatki (im wyższa rozdzielczość, tym bardziej złożona i wolniejsza sieć głębokiego uczenia). Wykrywanie jednoetapowe jest bardzo szybkie i ma prawie taką samą prędkość jak zwykła CNN do klasyfikacji. Wyniki muszą zostać przetworzone w celu zebrania razem komórek reprezentujących ten sam obiekt, co może prowadzić do dalszych nieścisłości. Architektury neuronowe oparte na tym podejściu to Single-Shot Detector (SSD), You Only Look Once (YOLO) i RetinaNet. Detektory jednostopniowe są bardzo szybkie, ale nie tak precyzyjne. Drugie podejście to wykrywanie dwuetapowe. W tym podejściu wykorzystuje się drugą sieć neuronową w celu udoskonalenia przewidywań pierwszej. Pierwszym etapem jest sieć propozycji, która wyprowadza swoje prognozy na siatkę. Drugi etap dopracowuje te propozycje i generuje ostateczną detekcję i lokalizację obiektów. R-CNN, Fast R-CNN i Faster R-CNN to dwuetapowe modele wykrywania, które są znacznie wolniejsze niż ich jednoetapowe odpowiedniki, ale bardziej precyzyjne w swoich przewidywaniach.

Dodawanie adnotacji do wielu obiektów na obrazach

Aby wytrenować modele uczenia głębokiego w celu wykrywania wielu obiektów, musisz podać więcej informacji niż w przypadku prostej klasyfikacji. Dla każdego obiektu podajesz zarówno klasyfikację, jak i współrzędne w obrazie za pomocą procesu adnotacji, co kontrastuje z etykietowaniem używanym w prostej klasyfikacji obrazu. Etykietowanie obrazów w zbiorze danych jest trudnym zadaniem nawet przy prostej klasyfikacji. Mając obraz, sieć musi zapewnić poprawną klasyfikację faz szkolenia i testów. W etykietowaniu sieć decyduje o właściwej etykiecie dla każdego obrazu i nie każdy będzie odbierał przedstawiony obraz w ten sam sposób. Osoby, które stworzyły zbiór danych ImageNet, skorzystały z klasyfikacji dostarczonej przez wielu użytkowników z platformy crowdsourcingowej Amazon Mechanical Turk (ImageNet korzystał z usługi Amazon na tyle, że w 2012 roku okazał się najważniejszym klientem akademickim Amazona). przy dodawaniu adnotacji do obrazu za pomocą obwiedni polegasz na pracy wielu osób. Adnotacja wymaga nie tylko oznaczenia każdego obiektu na obrazie, ale także określenia najlepszego pola, w którym każdy obiekt będzie otoczony. Te dwa zadania sprawiają, że adnotacja jest jeszcze bardziej złożona niż etykietowanie i bardziej podatna na generowanie błędnych wyników. Prawidłowe wykonanie adnotacji wymaga pracy większej liczby osób, które mogą zapewnić konsensus co do dokładności adnotacji. Niektóre oprogramowanie typu open source może pomóc w dodawaniu adnotacji do wykrywania obrazu (a także do segmentacji obrazu, omówionej w następnej sekcji). Szczególnie skuteczne są dwa narzędzia:

- LabelImg, stworzony przez TzuTa z samouczkiem.

- LabelMe to potężne narzędzie do segmentacji obrazów, które zapewnia usługę online.
- FastAnnotationTool, oparty o bibliotekę wizji komputerowej OpenCV. Pakiet jest mniej dobrze utrzymany, ale nadal opłacalny.

Segmentacja obrazów

Segmentacja semantyczna przewiduje klasę dla każdego piksela na obrazie, co jest inną perspektywą niż etykietowanie lub adnotacja. Niektórzy nazywają to zadanie gęstym przewidywaniem, ponieważ przewiduje ono każdy piksel na obrazie. Zadanie nie rozróżnia konkretnie różnych obiektów w predykcji. Na przykład segmentacja semantyczna może pokazać wszystkie piksele należące do klasy cat, ale nie dostarczy żadnych informacji o tym, co kot (lub koty) robią na zdjęciu. Możesz łatwo uzyskać wszystkie obiekty na segmentowanym obrazie przez przetwarzanie końcowe, ponieważ po wykonaniu przewidywania możesz uzyskać obszary pikseli obiektu i rozróżnić ich różne wystąpienia, jeśli w ramach tej samej przewidywanej klasy istnieje wiele oddzielonych obszarów. Segmentację obrazu mogą osiągnąć różne architektury uczenia głębokiego. Do najskuteczniejszych należą sieci w pełni splotowe (FCN) i U-NET. FCN są zbudowane dla pierwszej części (zwanej koderem), która jest taka sama jak CNN. Po początkowej serii warstw splotowych, FCN kończą się kolejną serią CNN, które działają w odwrotny sposób jako koder (co czyni je dekoderem). Dekoder jest skonstruowany tak, aby odtworzyć oryginalny rozmiar obrazu wejściowego i wyprowadzić jako piksele klasyfikację każdego piksela na obrazie. W ten sposób FCN uzyskuje semantyczną segmentację obrazu. FCN są zbyt intensywne obliczeniowo dla większości aplikacji czasu rzeczywistego. Ponadto wymagają dużych zestawów szkoleniowych, aby dobrze nauczyć się swoich zadań; w przeciwnym razie ich wyniki segmentacji są często zgrubne. Znalezienie części kodera FCN wstępnie przeszkolonej w ImageNet, która przyspiesza szkolenie i poprawia wydajność uczenia się, jest powszechne. Sieci U-NET są ewolucją FCN opracowaną przez Olafa Ronnebergera, Philippa Fischera i Thomasa Broxa w 2015 roku do celów medycznych (patrz https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/). Sieci U-NET mają przewagę w porównaniu z sieciami FCN. Kodowanie (nazywane również skróceniem) i części dekodujące (nazywane również rozszerzaniem) są idealnie symetryczne. Ponadto sieci U-NET wykorzystują połączenia skrótowe między warstwami kodera i dekodera. Skróty te umożliwiają łatwe przejście szczegółów obiektów z kodowania do części dekodujących U-NET, a wynikowa segmentacja jest precyzyjna i drobnoziarnista.

Budowanie modelu segmentacji od podstaw może być trudnym zadaniem, ale nie musisz tego robić. Możesz użyć kilku wstępnie wytrenowanych architektur U-NET i od razu zacząć korzystać z tego rodzaju sieci neuronowej, wykorzystując zoo modeli segmentacji (termin używany do opisania kolekcji wstępnie wytrenowanych modeli oferowanych przez wiele platform; zobacz https://modelzoo.co/ szczegóły) oferowane przez modele segmentacji, pakiet oferowany przez Pawła Jakubowskiego.

Postrzeganie przedmiotów w ich otoczeniu

Zintegrowanie funkcji widzenia z systemem czujników autonomicznego samochodu może zwiększyć pewność i bezpieczeństwo jego jazdy. Algorytm segmentacji może pomóc samochodowi odróżnić pasy ruchu od chodników, a także od innych przeszkód, które samochód powinien zauważyć. Samochód mógłby nawet być wyposażony w kompletny system typu end-to-end, taki jak NVIDIA, która steruje kierowaniem, przyspieszaniem i hamowaniem w sposób reaktywny na podstawie wizualnych danych wejściowych. System wizualny może wykryć na drodze pewne obiekty związane z prowadzeniem pojazdu, takie jak znaki drogowe i sygnalizacja świetlna. Może wizualnie śledzić trajektorie innych samochodów. We wszystkich przypadkach rozwiązaniem może być sieć głębokiego uczenia się. W sekcji „Wyróżniające zadania klasyfikacji" omówiono, w jaki sposób wykrywanie obiektów poprawia klasyfikację pojedynczego obiektu oferowaną przez CNN. W tej sekcji wyjaśniono również architektury

i obecne modele dwóch głównych podejść: wykrywanie jednoetapowe (lub wykrywanie jednoetapowe) i wykrywanie dwuetapowe (znane również jako propozycja regionu). Ta sekcja opisuje, jak działa jednostopniowy system wykrywania i zapewnia pomoc dla pojazdu autonomicznego. Programowanie takiego systemu wykrywania od podstaw byłoby trudnym zadaniem, wymagającym całej własnej książki. Na szczęście możesz korzystać z projektów open source na GitHub, takich jak Keras-RetinaNet. Keras-RetinaNet to implementacja Keras modelu RetinaNet zaproponowana przez Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He i Piotra Dollára w artykule „Focal Loss for Dense Object Detection" opublikowanym w sierpniu 2017 r. Isaac Newton stwierdził: „Jeżeli widziałem dalej, to przez stanie na ramionach Gigantów". Podobnie, możesz osiągnąć więcej w głębokim uczeniu, korzystając z istniejących architektur neuronowych i wstępnie wytrenowanych sieci.

Odkrywanie, jak działa RetinaNet

RetinaNet to wyrafinowany i interesujący model wykrywania obiektów, który stara się być tak szybki, jak inne jednoetapowe modele wykrywania, jednocześnie osiągając dokładność przewidywania pola ograniczającego w dwuetapowych systemach wykrywania, takich jak Faster R-CNN (ang. Faster R-CNN). model o najwyższej wydajności). Dzięki swojej architekturze RetinaNet osiąga swoje cele, wykorzystując techniki podobne do architektury U-NET omawianej dla segmentacji semantycznej. RetinaNet należy do grupy modeli zwanych sieciami piramid funkcji (FPN). RetinaNet zawdzięcza swoje działanie swoim autorom, Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He i Piotrowi Dollárowi, którzy zauważyli, że jednostopniowe modele wykrywania nie zawsze wykrywają obiekty dokładnie, ponieważ mają na nie wpływ przytłaczająca obecność rozpraszające elementy na obrazach wykorzystywanych do treningu. Ich artykuł „Focal Loss for Dense Object Detection" zawiera szczegółowe informacje na temat technik stosowanych przez RetinaNet. Problem w tym, że obrazy przedstawiają niewiele obiektów do wykrycia. W rzeczywistości sieci detekcji onestage są wytrenowane w odgadywaniu klasy każdej komórki na obrazie podzielonym przez ustaloną siatkę, w której większość komórek jest pozbawiona obiektów zainteresowania. W segmentacji semantycznej celami klasyfikacji są pojedyncze piksele. W jednoetapowym wykrywaniu celami są zestawy przylegających do siebie pikseli, wykonujące podobne zadanie do segmentacji semantycznej, ale na innym poziomie szczegółowości. Oto, co się dzieje, gdy masz taką przewagę przykładów zerowych w obrazie i używają podejścia szkoleniowego, które analizuje wszystkie dostępne komórki jako przykłady. Sieć z większym prawdopodobieństwem przewidzi, że nic nie znajduje się w przetworzonej komórce obrazu, niż zapewni poprawnie przewidzianą klasę. Sieci neuronowe zawsze wybierają najbardziej wydajną ścieżkę uczenia się, a w tym przypadku łatwiej jest przewidzieć tło niż cokolwiek innego. W tej sytuacji, która nosi nazwę niezrównoważonego uczenia się, wiele obiektów jest niewykrywanych przez sieć neuronową przy użyciu podejścia pojedynczego strzału do wykrywania obiektów.

W uczeniu maszynowym, gdy chcesz przewidzieć dwie różne liczbowo klasy (jedna to klasa większościowa, a druga to klasa mniejszościowa), pojawia się problem z niezrównoważoną klasyfikacją. Większość algorytmów nie działa poprawnie, gdy klasy są niezrównoważone, ponieważ preferują one klasę większościową. Dostępnych jest kilka rozwiązań tego problemu:

- Sampling: Wybieranie niektórych przykładów i odrzucanie innych.
- Downsample: Zmniejszenie efektu klasy większości poprzez wybranie tylko jej części, co równoważy przewidywania większości i mniejszości. W wielu przypadkach jest to najłatwiejsze podejście.
- Upsample: Zwiększenie efektu klasy mniejszości poprzez wielokrotne powielanie jej przykładów, aż klasa mniejszości będzie miała taką samą liczbę przykładów jak klasa większości.

Twórcy RetinaNet obierają inną drogę, jak zauważają we wcześniej wspomnianym artykule Focal Loss for Dense Object Detection. Odrzucają większość przykładów klas, które są łatwiejsze do sklasyfikowania i koncentrują się na komórkach, które są trudne do sklasyfikowania. W rezultacie funkcja kosztów sieci koncentruje się bardziej na dostosowaniu swoich wag do rozpoznawania obiektów w tle. Jest to rozwiązanie polegające na utracie ogniskowej i stanowi inteligentny sposób, aby jednoetapowe wykrywanie działało bardziej poprawnie, ale szybciej, co jest wymogiem aplikacji w czasie rzeczywistym, na przykład do wykrywania przeszkód lub obiektów w samochodach autonomicznych lub przetwarzania dużych ilości obrazów w monitoringu wideo.

Korzystanie z kodu Keras-RetinaNet

Wydany na licencji open source Apache License 2.0, Keras-RetinaNet jest projektem sponsorowanym przez holenderską firmę robotyczną Fitz i możliwym dzięki wielu współpracownikom (najlepszymi współpracownikami są Hans Gaiser i Maarten de Vries). Jest to implementacja sieci neuronowej RetinaNet napisana w Pythonie przy użyciu Keras . Znajdziesz Keras-RetinaNet wykorzystywane z powodzeniem w wielu projektach, z których najbardziej godnym uwagi i imponującym jest zwycięski model w konkursie NATO Innovation Challenge, którego zadaniem było wykrywanie samochodów na zdjęciach lotniczych.

Kod sieciowy wykrywania obiektów jest zbyt skomplikowany, aby można go było opisać na kilku stronach, a ponadto możesz użyć istniejącej sieci do skonfigurowania rozwiązań do uczenia głębokiego, dlatego w tej sekcji pokazano, jak pobrać i używać Keras-RetinaNet na swoim komputerze. Przed wypróbowaniem tego procesu upewnij się, że komputer został skonfigurowany zgodnie z opisem w Części 4 i rozważ kompromisy związane z użyciem różnych opcji wykonania opisanych w pasku bocznym „Rozważanie kosztów realistycznego wyniku” w Częsci 12. Na początek kroku, przesyłasz niezbędne pakiety i zaczynasz pobieranie spakowanej wersji repozytorium GitHub. W tym przykładzie użyto wersji 0.5.0 Keras-RetinaNet.

```
import os

import zipfile

import urllib.request

import warnings

warnings.filterwarnings("ignore")

url = "https://github.com/fizyr/\

keras-retinanet/archive/0.5.0.zip"

urllib.request.urlretrieve(url, './'+url.split('/')[-1])
```

Po pobraniu skompresowanego kodu przykładowy kod automatycznie wyodrębnia go za pomocą tych poleceń:

```
zip_ref = zipfile.ZipFile('./0.5.0.zip', 'r')

for name in zip_ref.namelist():

zip_ref.extract(name, './')

zip_ref.close()
```

Wykonanie tworzy nowy katalog o nazwie keras-retinanet-0.5.0, który zawiera kod do konfiguracji sieci neuronowej. Następnie kod wykonuje kompilację i instalację pakietu za pomocą polecenia pip:

```
os.chdir('./keras-retinanet-0.5.0')

!python setup.py build_ext --inplace

!pip install .
```

Wszystkie te polecenia właśnie pobrały kod, który buduje architekturę sieci. Przykład wymaga teraz wstępnie wytrenowanych wag i opiera się na wagach wytrenowanych w zestawie danych MS Coco przy użyciu ResNet50 CNN, sieci neuronowej, której firma Microsoft użyła do wygrania konkursu ImageNet 2015.

```
os.chdir('../')

url = "https://github.com/fizyr/\

keras-retinanet/releases/download/0.5.0/\

resnet50_coco_best_v2.1.0.h5"

urllib.request.urlretrieve(url, './'+url.split('/')[-1])
```

Pobranie wszystkich wag zajmuje trochę czasu, więc teraz jest dobry moment na uzupełnienie kawy. Po zakończeniu tego kroku przykład jest gotowy do zaimportowania wszystkich niezbędnych poleceń i zainicjowania modelu RetinaNet przy użyciu wstępnie wytrenowanych wag pobranych z Internetu. Ten krok ustawia również słownik do konwersji liczbowych wyników sieci na zrozumiałe klasy. Wybór klas jest przydatny dla detektora w samojezdnym samochodzie lub innego rozwiązania, które ma rozumieć obrazy zrobione z drogi lub skrzyżowania.

```
import os

import numpy as np

from collections import defaultdict

import keras

from keras_retinanet import models

from keras_retinanet.utils.image import (read_image_bgr,

preprocess_image, resize_image)

from keras_retinanet.utils.visualization import (draw_box,

draw_caption)

from keras_retinanet.utils.colors import label_color

import matplotlib.pyplot as plt

%matplotlib inline

model_path = os.path.join('.',

'resnet50_coco_best_v2.1.0.h5')
```

```
model = models.load_model(model_path,

backbone_name='resnet50')

labels_to_names = defaultdict(lambda: 'object',

{0: 'person', 1: 'bicycle', 2: 'car',

3: 'motorcycle', 4: 'airplane', 5: 'bus',

6: 'train', 7: 'truck', 8: 'boat',

9: 'traffic light', 10: 'fire hydrant',

11: 'stop sign', 12: 'parking meter',

25: 'umbrella'})
```

Aby przykład był użyteczny, potrzebujesz przykładowego obrazu do testowania modelu RetinaNet. Przykład opiera się na darmowym obrazie z Wikimedia przedstawiającym skrzyżowanie z ludźmi spodziewającymi się przejść przez jezdnię, niektórymi zatrzymanymi pojazdami, sygnalizacją świetlną i znakami drogowymi.

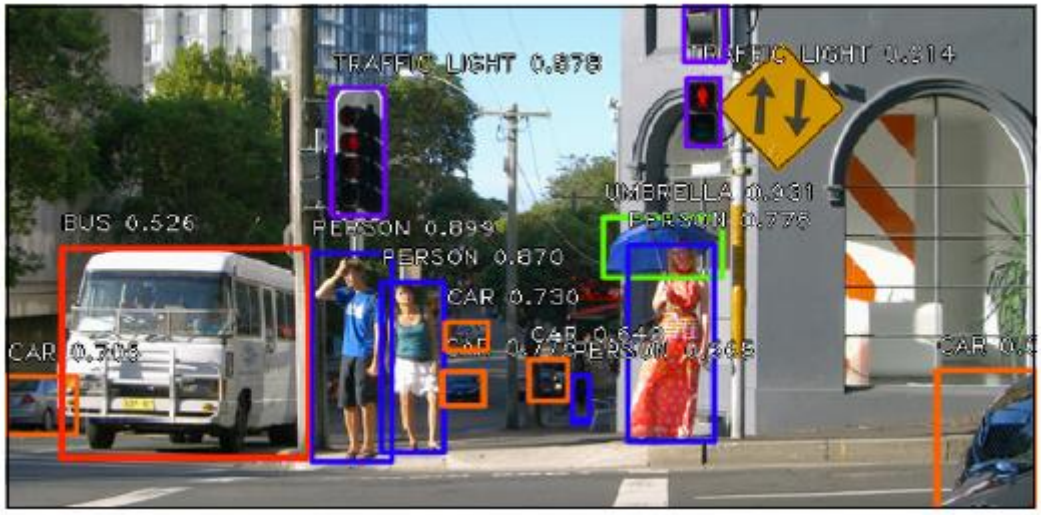
```
url = "https://upload.wikimedia.org/wikipedia/commons/\

thumb/f/f8/Woman_with_blue_parasol_at_intersection.png/\

640px-Woman_with_blue_parasol_at_intersection.png"

urllib.request.urlretrieve(url, './'+url.split('/')[-1])
```

Po zakończeniu pobierania obrazu czas przetestować sieć neuronową. We fragmencie kodu następującym po tym wyjaśnieniu kod odczytuje obraz z dysku, a następnie zamienia kanały z niebieskiego na czerwone (ponieważ obraz jest przesyłany w formacie BGR, ale RetinaNet działa z obrazami RGB). Na koniec kod wstępnie przetwarza i zmienia rozmiar obrazu. Wszystkie te kroki kończą się przy użyciu dostarczonych funkcji i nie wymagają specjalnych ustawień. Model wyświetli wykryte ramki ograniczające, poziom ufności (wynik prawdopodobieństwa, że sieć rzeczywiście coś wykryła) oraz etykietę kodu, która zostanie przekonwertowana na tekst przy użyciu wcześniej zdefiniowanego słownika etykiet. Pętla filtruje pola wydrukowane na obrazku według przykładu. W kodzie zastosowano próg ufności równy 0,5, co oznacza, że przykład zachowa wszelkie wykrycia, których ufność wynosi co najmniej 50 procent. Użycie niższego progu ufności skutkuje większą liczbą wykryć, zwłaszcza tych obiektów, które na obrazie wydają się małe, ale także zwiększa liczbę błędnych wykryć (na przykład niektóre cienie mogą zacząć być wykrywane jako obiekty). W zależności od Twoich celów przy użyciu RetinaNet, możesz zdecydować, że użycie niższego progu ufności jest w porządku. Zauważysz, że gdy obniżysz pewność, proporcja wynikowych dokładnych domysłów (tych z prawie 100-procentową pewnością) zmniejszy się. Taka proporcja nazywana jest precyzją wykrywania, a decydując, jaką precyzję możesz tolerować, możesz ustawić najlepszą pewność dla swoich celów.

```
image = read_image_bgr('640px-Woman_with_blue_parasol_at_

intersection.png')

draw = image.copy()

draw[:,:,0], draw[:,:,2] = image[:,:,2], image[:,:,0]
```

```
image = preprocess_image(image)
image, scale = resize_image(image)
boxes, scores, labels = model.predict_on_batch(np.expand_
dims(image, axis=0))
boxes /= scale
for box, score, label in zip(boxes[0], scores[0], labels[0]):
if score > 0.5:
color = label_color(label)
b = box.astype(int)
draw_box(draw, b, color=color)
caption = "{} {:.3f}".format(labels_to_names[label],
score)
draw_caption(draw, b, caption.upper())
plt.figure(figsize=(12, 6))
plt.axis('off')
plt.imshow(draw)
plt.show()
```

Pierwsze uruchomienie kodu może trochę potrwać, ale po kilku obliczeniach powinieneś otrzymać wynik przedstawiony na rysunku. Sieć może z powodzeniem wykrywać różne obiekty, niektóre bardzo małe (np. osoba w tle), niektóre częściowo widoczne (np. nos samochodu po prawej stronie obrazu). Każdy wykryty obiekt jest ograniczony przez jego obwiednię, co tworzy szeroki zakres możliwych zastosowań.



Na przykład możesz użyć sieci do wykrycia, że ktoś używa parasola lub jakiegoś obiektu. Podczas przetwarzania wyników można powiązać fakt, że dwa pola ograniczające nakładają się na siebie, przy czym jedno to parasol, a drugie to osoba, oraz że pierwsze pole jest umieszczone nad drugim, aby wywnioskować, że dana osoba jest trzymając parasol. Nazywa się to wizualnym wykrywaniem relacji. W ten sam sposób, poprzez ogólne ustawienie wykrytych obiektów i ich względne położenie, możesz wytrenować drugą sieć głębokiego uczenia się, aby wywnioskować ogólny opis sceny.

Pokonywanie wrogich ataków na aplikacje do głębokiego uczenia się

Ponieważ uczenie głębokie znajduje wiele zastosowań w samochodach autonomicznych, takich jak wykrywanie i interpretacja znaków drogowych i świateł; wykrywanie drogi i jej pasów; wykrywanie przechodzących pieszych i innych pojazdów; sterowanie samochodem przez kierowanie i hamowanie w podejściu od końca do końca do jazdy automatycznej; i tak dalej, mogą pojawić się pytania o bezpieczeństwo autonomicznego samochodu. prowadzenie samochodu nie jest jedyną powszechną czynnością, która przechodzi rewolucję. ze względu na aplikacje do głębokiego uczenia się. Niedawno wprowadzone aplikacje, które są dostępne publicznie, obejmują rozpoznawanie twarzy w celu zapewnienia bezpieczeństwa. Innym przykładem aplikacji do głębokiego uczenia się jest rozpoznawanie mowy używane w systemach głosowych (VCS), dostarczane przez wiele firm, takich jak Apple, Amazon, Microsoft i Google, w szerokiej gamie aplikacji, w tym Siri, Alexa i Strona główna Google.

Niektóre z tych aplikacji do głębokiego uczenia się mogą powodować szkody ekonomiczne lub nawet zagrażać życiu, jeśli nie podają prawidłowej odpowiedzi. W związku z tym możesz być zaskoczony odkryciem, że hakerzy mogą celowo oszukać głębokie sieci neuronowe i naprowadzić je na błędne przewidywania za pomocą określonych technik zwanych przykładami kontradyktoryjności. Przykładem kontradyktoryjnym jest ręcznie wykonana część danych, która jest przetwarzana przez sieć neuronową jako dane wejściowe treningowe lub testowe. Haker modyfikuje dane, aby wymusić niepowodzenie algorytmu w jego zadaniu. Każdy kontradyktoryjny przykład nosi modyfikacje, które są rzeczywiście drobne, subtelne i celowo niedostrzegalne dla ludzi. Modyfikacje, choć nieskuteczne dla ludzi, nadal są dość skuteczne w zmniejszaniu efektywności i użyteczności sieci neuronowej. Często takie złośliwe przykłady mają na celu doprowadzenie sieci neuronowej do awarii w przewidywalny sposób, aby stworzyć coś nielegalnego na korzyść dla hakera. Oto tylko kilka złośliwych zastosowań przykładów przeciwnika (lista nie jest wyczerpująca):

- Wprowadzenie w błąd autonomicznego samochodu do wypadku
- Uzyskiwanie pieniędzy z oszustwa ubezpieczeniowego poprzez posiadanie fałszywych zdjęć roszczenia, które automatyczne systemy uznają za prawdziwe
- Oszukanie systemu rozpoznawania twarzy w celu rozpoznania niewłaściwej twarzy i przyznania dostępu do pieniędzy na koncie bankowym lub danych osobowych na urządzeniu mobilnym

Część 16 omawia generatywne sieci adwersarzy (GAN) i szkolenia adwersarzy, które mają zupełnie inny cel niż przykłady adwersarzy. Techniki te są sposobem na wytrenowanie głębokiej sieci neuronowej w celu generowania z niej nowych przykładów dowolnego rodzaju.

Oszukiwanie pikseli

Po raz pierwszy ujawnione w artykule „Intriguing Properties of Neural Networks” skłoniły wielu badaczy do opracowania szybszych i skuteczniejszych sposobów tworzenia takich przykładów, niż wskazywał oryginalny artykuł.

ODKRYCIE, ŻE MUFFIN NIE JEST CHIHUAHUA

Czasami klasyfikacja obrazów głębokiego uczenia nie daje właściwej odpowiedzi, ponieważ obraz docelowy jest z natury niejednoznaczny lub renderowany dla zagadkowych obserwatorów. Na przykład niektóre obrazy są tak mylące, że mogą nawet na chwilę zmylić ludzkiego egzaminatora, takie jak memy internetowe Chihuahua kontra Muffin lub Labradoodle kontra smażony kurczak. Sieć neuronowa może źle zrozumieć mylące obrazy, jeśli jej architektura nie jest adekwatna do zadania, a jej szkolenie nie było wyczerpujące pod względem widzianych przykładów. Mariya Yao, felietonistka poświęcona technologii sztucznej inteligencji, porównała różne interfejsy API wizji komputerowej i stwierdziła, że są one nawet pełne. Wzorcowe produkty wizyjne mogą zostać oszukane przez niejednoznaczne zdjęcia. Ostatnio inne badania rzuciły wyzwanie głębokim sieciom neuronowym, proponując nieoczekiwane perspektywy znanych obiektów. W artykule zatytułowanym „Strike (with) a Pose: Neural Networks Are Easy Fooled by Strange Poses of Familiar Objects" , naukowcy odkryli, że prosta niejednoznaczność może oszukać stan techniki klasyfikatory obrazu i detektory obiektów wyszkolone na dużych zbiorach danych obrazu. Często obiekty są uczone przez sieci neuronowe na podstawie zdjęć zrobionych w kanonicznych pozach (czyli w zwykłych i zwyczajnych sytuacjach). W obliczu obiektu w nietypowej pozie lub poza zwykłym otoczeniem niektóre sieci neuronowe nie mogą kategoryzować powstałego obiektu. Na przykład spodziewasz się, że autobus szkolny będzie jeździł po drodze, ale jeśli obrócisz go i skręcisz w powietrzu, a następnie wylądujesz na środku drogi, sieć neuronowa z łatwością zobaczy go jako śmieciarkę, uderzenie. worek, a nawet pług śnieżny. Możesz argumentować, że błędna klasyfikacja wynika z błędu uczenia się (uczenia sieci neuronowej przy użyciu tylko obrazów w kanonicznych pozach). Oznacza to jednak, że obecnie nie należy polegać na tej technologii w każdych okolicznościach, zwłaszcza, jak wskazali autorzy artykułu, w zastosowaniach samochodów autonomicznych, ponieważ obiekty mogą nagle pojawiać się na drodze w nowych pozach lub okolicznościach.

Przykłady kontradyktoryjne wciąż ograniczają się do laboratoriów badawczych głębokiego uczenia się. Z tego powodu można znaleźć wiele prac naukowych cytowanych w tych akapitach, odwołując się do różnego rodzaju przykładów. Jednak nigdy nie należy lekceważyć przykładów kontradyktoryjności jako pewnego rodzaju akademickiej dywersji, ponieważ ich potencjał szkód jest wysoki. U podstaw wszystkich tych podejść leży idea, że mieszanie z obrazem pewnych informacji liczbowych, nazywanych perturbacją, może sprawić, że sieć neuronowa będzie zachowywać się inaczej niż oczekiwano, chociaż w sposób kontrolowany. Tworząc przykład kontradyktoryjny, dodajesz specjalnie wymyślony szum (wyglądający na coś, co wydaje się być losowymi liczbami) do istniejącego obrazu, a to wystarczy, aby oszukać większość CNN (ponieważ często ta sama sztuczka działa z różnymi architekturami, gdy jest wytrenowana przez tę samą dane). Ogólnie rzecz biorąc, możesz odkryć takie perturbacje, mając dostęp do modelu (jego architektury i wag). Następnie wykorzystujesz jego algorytm wstecznej propagacji błędów, aby systematycznie odkrywać najlepszy zestaw informacji liczbowych do dodania do obrazu, aby móc zmutować jedną przewidywaną klasę w inną. Efekt perturbacji można uzyskać, zmieniając pojedynczy piksel na obrazie. Dzięki temu podejściu badacze uzyskali doskonale działające, kontradyktoryjne przykłady, które odkryli badacze z Kyushu University i opisali w artykule „One Pixel Attack for Fooling Deep Neural Networks".

Hakowanie za pomocą naklejek i innych artefaktów

Większość przeciwstawnych przykładów to eksperymenty laboratoryjne dotyczące odporności widzenia, a przykłady te mogą zademonstrować wszystkie swoje możliwości, ponieważ są tworzone przez bezpośrednią modyfikację danych wejściowych i testowanych obrazów podczas fazy uczenia. Jednak wiele aplikacji opartych na głębokim uczeniu działa w świecie rzeczywistym, a wykorzystanie technik laboratoryjnych nie zapobiega złośliwym atakom. Takie ataki nie wymagają dostępu do

podstawowego modelu neuronowego, aby były skuteczne. Niektóre przykłady mogą mieć postać naklejki lub niesłyszalnego dźwięku, z którym sieć neuronowa nie potrafi sobie poradzić. Artykuł zatytułowany „Przykłady kontradyktoryjne w świecie fizycznym" pokazuje, że różne ataki są również możliwe w warunkach nielaboratoryjnych. Wystarczy wydrukować przykłady kontradyktoryjności i pokazać je kamerze zasilającej sieć neuronową (np. za pomocą kamery w telefonie komórkowym). To podejście pokazuje, że skuteczność przykładu kontradyktoryjnego nie jest ściśle związana z danymi liczbowymi wprowadzanymi do sieci neuronowej. To zestaw kształtów, kolorów i kontrastu obecny na obrazie, który spełnia tę sztuczkę, i nie potrzebujesz bezpośredniego dostępu do modelu neuronowego, aby dowiedzieć się, który zestaw działa najlepiej. Jak sieć może pomylić obraz pralki z sejfem lub głośnikiem bezpośrednio z tego filmu nakręconego przez autorów, którzy oszukali demo kamery TensorFlow, aplikacji na urządzenia mobilne, która dokonuje klasyfikacji obrazu w locie.

Inni badacze z Carnegie Mellon University znaleźli sposób na oszukanie wykrywania twarzy, aby uwierzyć, że dana osoba jest celebrytą, produkując oprawki okularów, które mogą wpływać na rozpoznawanie instancji przez głęboką sieć neuronową. W miarę upowszechniania się automatycznych systemów bezpieczeństwa, możliwość oszukania systemu za pomocą prostych dodatków, takich jak okulary, mogą stać się poważnym zagrożeniem bezpieczeństwa. Artykuł zatytułowany „Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition" opisuje, w jaki sposób akcesoria mogą pozwolić zarówno na uniknięcie osobistego rozpoznania, jak i podszywania się. Wreszcie, inne niepokojące użycie w świecie rzeczywistym kontradyktoryjnego przykładu pojawia się w artykule „Robust Physical-World Attacks on Deep Learning Visual Classification". Zwykłe czarno-białe naklejki umieszczone na znaku stopu mogą wpływać na to, jak autonomiczny samochód rozumie sygnał, błędnie rozumiejąc go jako inny znak drogowy. Kiedy użyjesz bardziej kolorowych (ale też bardziej widocznych) naklejek, takich jak te opisane w artykule „Adversarial Patch" możesz kierować przewidywaniami sieci neuronowej w określonym kierunku, ignorując wszystko poza naklejką i wprowadzającymi w błąd informacjami. Jak wyjaśniono w artykule, sieć neuronowa może przewidzieć, że banan to cokolwiek innego, po prostu umieszczając w pobliżu odpowiednią oszukańczą naklejkę.

W tym momencie możesz się zastanawiać, czy możliwa jest jakakolwiek obrona przed wrogimi przykładami, czy też prędzej czy później zniszczą one zaufanie publiczne do zastosowań głębokiego uczenia się, zwłaszcza w dziedzinie samochodów autonomicznych. Intensywnie badając, jak wprowadzić w błąd sieć neuronową, naukowcy odkrywają również, jak chronić ją przed niewłaściwym użyciem. Po pierwsze, sieci neuronowe mogą aproksymować dowolną funkcję. Jeśli sieci neuronowe są wystarczająco złożone, mogą również samodzielnie określić, w jaki sposób wykluczyć wrogie przykłady, gdy są nauczane przez inne przykłady. Po drugie, nowatorskie techniki, takie jak ograniczanie wartości w sieci neuronowej lub zmniejszanie rozmiaru sieci neuronowej po jej wyuczeniu (technika zwana destylacją, używana wcześniej w celu zapewnienia żywotności sieci na urządzeniach z małą ilością pamięci) zostały pomyślnie przetestowane w wielu różnych rodzajach ataki przeciwnika.

Praca nad przetwarzaniem języka

Komputer nie rozumie języka; przetwarza język tylko dla określonych aplikacji. Ponadto komputer nie może przetwarzać języka, chyba że jest wysoce formalny i precyzyjny, jak na przykład język programowania. Sztywne reguły składni i gramatyka umożliwiają komputerowi przekształcenie programu napisanego przez programistę w języku komputerowym, takim jak Python, w język maszynowy, który określa, jakie zadania wykona komputer. Język ludzki wcale nie jest podobny do języka komputera. Język ludzki często nie ma precyzyjnej struktury i jest pełen błędów, sprzeczności i niejasności, a mimo to działa dobrze dla ludzi, przy pewnym wysiłku ze strony słuchacza, aby służyć społeczeństwu ludzkiemu i postępowi wiedzy. Programowanie komputera do przetwarzania ludzkiego języka jest zatem zniechęcającym zadaniem, które jest możliwe dopiero od niedawna przy użyciu przetwarzania języka naturalnego (NLP), głębokiego uczenia się sieci neuronowych rekurencyjnych (RNN) i osadzania słów. Osadzanie słów to nazwa techniki modelowania języka i uczenia się funkcji w NLP, która odwzorowuje słownictwo na wektory liczb rzeczywistych przy użyciu produktów takich jak Word-2vec, GloVe i fastText. Widzisz również, że jest używany we wstępnie przeszkolonych sieciach, takich jak BERT o otwartym kodzie źródłowym Google. W tym rozdziale zaczniesz od podstaw potrzebnych do zrozumienia NLP i zobaczysz, jak może ono pomóc w budowaniu lepszych modeli głębokiego uczenia się dla problemów językowych. Rozdział następnie wyjaśnia osadzanie słów, w jaki sposób wstępnie wytrenowane sieci zrewolucjonizują głębokie uczenie się oraz jak komputery mogą komunikować się za pośrednictwem chatbotów. Rozdział zamyka przykład modelu głębokiego uczenia stosowanego do analizy sentymentu, który odkrywa opinie w tekście. Nie musisz ręcznie wpisywać kodu źródłowego tego rozdziału. W rzeczywistości jest to o wiele łatwiejsze, jeśli korzystasz ze źródła do pobrania.

Język przetwarzania

Dla uproszczenia możesz wyświetlić język jako sekwencję słów złożoną z liter (a także znaków interpunkcyjnych, symboli, emotikonów itd.). Głębokie uczenie najlepiej przetwarza język przy użyciu warstw RNN, takich jak LSTM lub GRU . Jednak wiedza o używaniu RNN nie mówi ci, jak używać sekwencji jako danych wejściowych; musisz określić rodzaj sekwencji. W rzeczywistości głębokie uczenie sieci akceptuje tylko numeryczne wartości wejściowe. Komputery kodują zrozumiałe sekwencje liter na liczby zgodnie z protokołem, takim jak 8-bitowy format transformacji Unicode (UTF-8). UTF-8 to najczęściej używane kodowanie. (Możesz przeczytać elementarz o kodowaniu na https://www.alexreisner.com/code/character-encoding.)

Głębokie uczenie może również przetwarzać dane tekstowe za pomocą splotowych sieci neuronowych (CNN) zamiast RNN, reprezentując sekwencje jako macierze (podobnie jak przetwarzanie obrazów). Keras obsługuje warstwy CNN, takie jak Conv1D, które mogą operować na uporządkowanych w czasie cechach - czyli sekwencjach słów lub innych sygnałów. Po wyjściu konwolucji 1D zwykle następuje warstwa MaxPooling1D, która podsumowuje dane wyjściowe. CNN zastosowane do sekwencji znajdują granicę w swojej niewrażliwości na globalny porządek sekwencji. (Mają tendencję do dostrzegania lokalnych wzorców.) Z tego powodu najlepiej nadają się do przetwarzania sekwencji w połączeniu z RNN, a nie jako ich zamiennik. Przetwarzanie języka naturalnego (NLP) składa się z szeregu procedur, które poprawiają przetwarzanie słów i fraz do analizy statystycznej, algorytmów uczenia maszynowego i głębokiego uczenia. NLP zawdzięcza swoje korzenie językoznawstwu obliczeniowemu, które napędzało systemy oparte na regułach sztucznej inteligencji, takie jak systemy eksperckie, które podejmowały decyzje w oparciu o komputerowe tłumaczenie ludzkiej wiedzy, doświadczenia i sposobu myślenia. NLP przetworzył informacje tekstowe, które są nieustrukturyzowane, na bardziej ustrukturyzowane dane, aby systemy eksperckie mogły z łatwością nimi manipulować i oceniać. Głębokie uczenie się wzięło dziś górę, a systemy eksperckie ograniczają się do konkretnych

zastosowań, w których interpretacja i kontrola procesów decyzyjnych mają pierwszorzędne znaczenie (na przykład w zastosowaniach medycznych i systemach decyzyjnych dotyczących zachowania kierowcy w niektórych samojezdnych samochodach). Jednak potok NLP jest nadal dość istotny dla wielu aplikacji głębokiego uczenia

Definiowanie rozumienia jako tokenizacji

W potoku NLP pierwszym krokiem jest uzyskanie nieprzetworzonego tekstu. Zwykle przechowujesz go w pamięci lub uzyskujesz do niego dostęp z dysku. Gdy dane są zbyt duże, aby zmieścić się w pamięci, utrzymujesz wskaźnik do nich na dysku (taki jak nazwa katalogu i nazwa pliku). W poniższym przykładzie używasz trzech dokumentów (reprezentowanych przez zmienne łańcuchowe) przechowywanych na liście (kontener dokumentów to korpus w lingwistyce obliczeniowej):

```
import numpy as np

texts = ["My dog gets along with cats",

"That cat is vicious",

"My dog is happy when it is lunch"]
```

Po uzyskaniu tekstu przetwarzasz go. Podczas przetwarzania każdej frazy wyodrębniasz odpowiednie cechy z tekstu (zwykle tworzysz macierz torby słów) i przekazujesz wszystko do modelu uczenia, takiego jak algorytm głębokiego uczenia się. Podczas przetwarzania tekstu możesz używać różnych transformacji do manipulowania tekstem (przy czym tokenizacja jest jedyną obowiązkową transformacją):

- Normalizacja: usuń wielkie litery.
- Czyszczenie: Usuń elementy nietekstowe, takie jak interpunkcja i cyfry.
- Tokenizacja: Podziel zdanie na pojedyncze słowa.
- Zatrzymaj usuwanie słów: usuń popularne, mało pouczające słowa, które nie dodają znaczenia zdaniu, takie jak przedimki a i a. Usunięcie negacji, takich jak not, może być szkodliwe, jeśli chcesz odgadnąć sentyment.
- Rdzenie: Zredukuj słowo do jego rdzenia (czyli formy słowa przed dodaniem afiksów fleksyjnych. Algorytm zwany stemmerem może to zrobić w oparciu o szereg reguł.
- Lematyzacja: Przekształć słowo w jego formę słownikową (lemat). Jest to alternatywa dla stemmingu, ale jest bardziej złożona, ponieważ nie używasz algorytmu. Zamiast tego używasz słownika, aby przekształcić każde słowo w jego lemat.
- Pos-tagowanie: Oznacz każde słowo we frazie wraz z jego rolą gramatyczną w zdaniu (np. tagowanie słowa jako czasownika lub rzeczownika).
- N-gramy: Powiąż każde słowo z określoną liczbą (n w n-gramach) kolejnych słów i traktuj je jako unikalny zestaw. Zwykle do celów analizy najlepiej sprawdzają się bigramy (seria dwóch sąsiednich elementów lub tokenów) i trigramy (seria trzech sąsiednich elementów lub tokenów).

Aby osiągnąć te przekształcenia, możesz potrzebować wyspecjalizowanego pakietu Pythona, takiego jak NLTK lub Scikit-learn. Pracując z głębokim uczeniem i dużą liczbą przykładów, potrzebujesz tylko podstawowych przekształceń: normalizacji, czyszczenia i tokenizacji. Warstwy uczenia głębokiego mogą określić, jakie informacje należy wyodrębnić i przetworzyć. Pracując z kilkoma przykładami, musisz zapewnić jak najwięcej przetwarzania NLP, aby pomóc sieci głębokiego uczenia się określić, co zrobić, pomimo niewielkich wskazówek zawartych w kilku przykładach. Keras oferuje funkcję keras.preprocessing.text.Tokenizer, która normalizuje (używając dolnego parametru ustawionego na

True), czyści (parametr filters zawiera ciąg znaków do usunięcia, zwykle: '!"#$%&( )*+,-./:;<=>?@ [\]^_`{|}~ ') i tokenizuje.

Włożenie wszystkich dokumentów do torby

Po przetworzeniu tekstu należy wydobyć odpowiednie cechy, co oznacza przekształcenie pozostałego tekstu na informację liczbową do przetworzenia przez sieć neuronową. Zwykle robi się to za pomocą podejścia bag-of-words, które uzyskuje się poprzez kodowanie częstotliwościowe lub kodowanie binarne tekstu. Ten proces oznacza przekształcenie każdego słowa w kolumnę macierzy o szerokości odpowiadającej liczbie słów, które musisz przedstawić. Poniższy przykład pokazuje, jak osiągnąć ten proces i co on oznacza. W przykładzie wykorzystano listę tekstów utworzoną wcześniej w rozdziale. W pierwszym kroku przygotowujesz podstawową normalizację i tokenizację za pomocą kilku poleceń Pythona do określenia rozmiaru słownictwa do przetwarzania:

unique_words = set(word.lower() for phrase in texts for

word in phrase.split(" "))

print(f"There are {len(unique_words)} unique words")

Kod zgłasza 1/4 słowa. Teraz możesz załadować funkcję Tokenizer z Keras i ustawić ją tak, aby przetwarzała tekst, podając oczekiwany rozmiar słownictwa:

from keras.preprocessing.text import Tokenizer

vocabulary_size = len(unique_words) + 1

tokenizer = Tokenizer(num_words=vocabulary_size)

Użycie vocabulary_size, który jest zbyt mały, może wykluczyć ważne słowa z procesu uczenia się. Zbyt duży może bezużytecznie zużywać pamięć komputera. Musisz podać Tokenizerowi prawidłowe oszacowanie liczby odrębnych słów zawartych na liście tekstów. Zawsze dodajesz także 1 do vocabulary_size, aby zapewnić dodatkowe słowo na początek frazy (termin, który pomaga sieci głębokiego uczenia się). W tym momencie Tokenizer mapuje słowa obecne w tekstach na indeksy, które są wartościami liczbowymi reprezentującymi słowa w tekście:

tokenizer.fit_on_texts(texts)

print(tokenizer.index_word)

The resulting indexes are as follows:

{1: 'is', 2: 'my', 3: 'dog', 4: 'gets', 5: 'along',

6: 'with', 7: 'cats', 8: 'that', 9: 'cat', 10: 'vicious',

11: 'happy', 12: 'when', 13: 'it', 14: 'lunch'}

Indeksy reprezentują numer kolumny, w której znajdują się informacje o słowie:

print(tokenizer.texts_to_matrix(texts))

Here's the resulting matrix:

[[0. 0. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0.]

[0. 1. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0.]

[0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1.]]

Macierz składa się z 15 kolumn (14 słów plus wskaźnik początku frazy) i trzech wierszy reprezentujących trzy przetworzone teksty. Jest to macierz tekstowa do przetworzenia przy użyciu płytkiej sieci neuronowej (RNN wymagają innego formatu, jak omówiono później), która zawsze ma rozmiar vocabulary_size według liczby tekstów. Liczby wewnątrz macierzy oznaczają, ile razy słowo pojawia się w wyrażeniu. Nie jest to jednak jedyna możliwa reprezentacja. Oto inne:

- Kodowanie częstotliwości: Zlicza liczbę wystąpień słów we frazie.
- Kodowanie one-hot lub kodowanie binarne: zwraca uwagę na obecność słowa we frazie, niezależnie od tego, ile razy się pojawia.
- Termin Frequency-Inverse Document Frequency score (TF-IDF): Koduje miarę określającą, ile razy słowo pojawia się w dokumencie w stosunku do ogólnej liczby słów w macierzy. (Słowa z wyższymi wynikami są bardziej charakterystyczne; słowa z niższymi wynikami mają mniej informacji.)

Możesz użyć transformacji TF-IDF bezpośrednio z Keras. Tokenizer oferuje metodę texts_to_matrix, która domyślnie koduje twój tekst i przekształca go w macierz, w której kolumny to twoje słowa, wiersze to twoje teksty, a wartości to częstotliwość słów w tekście. Jeśli zastosujesz transformację przez określenie mode='tfidf', transformacja użyje TF-IDF zamiast częstotliwości słów do wypełnienia wartości macierzy:

print(np.round(tokenizer.texts_to_matrix(texts,

mode='tfidf'), 1))

Zauważ, że używając reprezentacji macierzowej, bez względu na to, czy używasz binarnej, częstotliwościowej, czy bardziej wyrafinowanego TF-IDF, straciłeś sens porządkowania słów, który istnieje we frazie. Podczas przetwarzania słowa rozpraszają się w różnych kolumnach, a sieć neuronowa nie może odgadnąć kolejności słów we frazie. Ten brak porządku jest powodem, dla którego nazywasz to podejściem typu „bag-of-words”. Podejście bag-of-words jest stosowane w wielu algorytmach uczenia maszynowego, często z wynikami od dobrych do sprawiedliwych, i można je zastosować do sieci neuronowej przy użyciu gęstych warstw architektury. Transformacje słów zakodowanych w n_gramach (omówione w poprzednim akapicie jako transformacje przetwarzania NLP) dostarczają więcej informacji, ale znowu nie możesz powiązać słów. RNN śledzą sekwencje, więc nadal używają kodowania „one-hot”, ale nie kodują całej frazy, a raczej indywidualnie kodują każdy token (którym może być słowo, znak, a nawet kilka znaków). Z tego powodu oczekują ciągu indeksów reprezentujących frazę:

print(tokenizer.texts_to_sequences(texts))

Gdy każda fraza przechodzi do wejścia sieci neuronowej jako sekwencja liczb indeksu, liczba jest zamieniana w zakodowany z jednym gorącym wektorem wektor. Zakodowane w jednym czasie wektory są następnie wprowadzane do warstw RNN pojedynczo, dzięki czemu są łatwe do nauczenia. Na przykład, oto transformacja pierwszej frazy w macierzy:

[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]]

W tej reprezentacji otrzymujesz odrębną macierz dla każdego fragmentu tekstu. Każda macierz przedstawia poszczególne teksty jako odrębne słowa przy użyciu kolumn, ale teraz wiersze reprezentują kolejność występowania słów. (Pierwszy wiersz to pierwsze słowo, drugi wiersz to drugie słowo itd.)

Zapamiętywanie ważnych sekwencji

Praca z TF-IDF i n-gramami (literowymi lub słownymi) umożliwia tworzenie modeli językowych na kilku przykładach. Kodowanie fraz jako sekwencji jednowyrazowych kodowań jednowyrazowych pomaga w efektywnym korzystaniu z RNN. Jednak lepszym sposobem przetwarzania danych tekstowych z większą szybkością (sposób, który tworzy zaawansowane modele uczenia głębokiego) jest użycie osadzania. Osadzania mają długą historię. Pojęcie osadzeń pojawiło się w statystycznej analizie wielowymiarowej pod nazwą wielowymiarowej analizy korespondencji. Od lat 70. Jean-Paul Benzécri, francuski statystyk i językoznawca, wraz z wieloma innymi francuskimi badaczami z Francuskiej Szkoły Analizy Danych odkryli, jak mapować ograniczony zestaw słów na przestrzenie niskowymiarowe (zazwyczaj reprezentacje 2D, takie jak jako mapa topograficzna). Proces ten zamienia słowa w znaczące liczby i prognozy, odkrycie, które przyniosło wiele zastosowań w językoznawstwie i naukach społecznych oraz utorowało drogę dla ostatnich postępów w przetwarzaniu języka z wykorzystaniem głębokiego uczenia się.

Zrozumienie semantyki przez osadzanie słów

Sieci neuronowe są niezwykle szybkie w przetwarzaniu danych i znajdowaniu odpowiednich wag, aby osiągnąć najlepsze prognozy, podobnie jak wszystkie warstwy głębokiego uczenia omówione do tej pory: od CNN po RNN. Te sieci neuronowe mają limity wydajności oparte na danych, które muszą przetworzyć, takie jak normalizowanie danych, aby umożliwić prawidłowe działanie sieci neuronowej lub wymuszanie zakresu wartości wejściowych od 0 do +1 lub od -1 do +1 w celu zmniejszenia problemów podczas aktualizacji wagi sieci. Normalizacja odbywa się wewnętrznie w sieci za pomocą funkcji aktywacji, takich jak tanh, która ściska wartości w zakresie od –1 do +1, które stosują statystyczną transformację wartości przenoszonych z jednej warstwy do drugiej. Innym rodzajem problematycznych danych, z którymi sieć neuronowa jest trudna do obsłużenia, są dane rzadkie. Masz rzadkie dane, gdy twoje dane składają się głównie z wartości zerowych, co dzieje się dokładnie, gdy przetwarzasz dane tekstowe przy użyciu kodowania częstotliwościowego lub binarnego, nawet jeśli nie używasz TF-IDF. Podczas pracy z rzadkimi danymi sieć neuronowa nie tylko będzie miała trudności ze znalezieniem dobrego rozwiązania , ale będziesz potrzebować również ogromnej liczby wag dla warstwy wejściowej, ponieważ macierze sparse są zwykle dość szerokie (mają wiele kolumn). Rzadkie problemy z danymi motywowały użycie osadzania słów, co jest sposobem na przekształcenie rzadkiej macierzy w gęstą. Osadzanie słów może zmniejszyć liczbę kolumn w macierzy z setek tysięcy do kilkuset. Ponadto nie pozwalają na żadne wartości zerowe wewnątrz macierzy. Proces osadzania słów nie jest wykonywany losowo, ale jest opracowany tak, aby słowa miały podobne wartości, gdy mają to samo znaczenie lub znajdują się w tych samych tematach. Innymi słowy, jest to złożone mapowanie; każda kolumna osadzania jest mapą specjalizacji (lub skalą, jeśli wolisz), a podobne lub powiązane słowa gromadzą się blisko siebie.

Osadzanie słów nie jest jedyną zaawansowaną techniką, której możesz użyć, aby rozwiązania do uczenia głębokiego błyszczały w nieustrukturyzowanym tekście. Ostatnio pojawiła się seria przeszkolonych sieci, które jeszcze bardziej ułatwiają modelowanie problemów językowych. Na

przykład jednym z najbardziej obiecujących jest Google Bidirectional Encoder Representations from Transformers (BERT). Jako inny przykład możesz mieć osadzanie, które przekształca nazwy różnych produktów spożywczych w kolumny wartości liczbowych, co jest macierzą osadzonych słów. Na tej macierzy słowa przedstawiające owoce mogą mieć podobny wynik w określonej kolumnie. W tej samej kolumnie warzywa mogą mieć różne wartości, ale niezbyt daleko od owoców. Wreszcie, nazwy dań mięsnych mogą być dalece wartościowe od owoców i warzyw. Osadzanie wykonuje tę pracę poprzez konwersję słów na wartości w macierzy. Wartości są podobne, gdy słowa są synonimami lub odnoszą się do podobnego pojęcia. (Nazywa się to podobieństwem semantycznym, z semantycznym odnoszeniem się do znaczenia słów.) Ponieważ to samo znaczenie semantyczne może występować w różnych językach, możesz użyć starannie zbudowanych osadzeń, aby pomóc w tłumaczeniu z jednego języka na inny: słowo w jednym języku będzie miało te same osadzone wyniki jako to samo słowo w innym języku. Naukowcy z laboratorium Facebook AI Research (FAIR) znaleźli sposób na synchronizację różnych osadzeń i wykorzystanie ich do dostarczania wielojęzycznych aplikacji opartych na głębokim uczeniu.

Ważnym aspektem, o którym należy pamiętać podczas pracy z osadzaniami słów, jest to, że są one produktem danych, a zatem odzwierciedlają zawartość danych użytych do ich utworzenia. Ponieważ embedingi słów wymagają do prawidłowego wygenerowania dużej ilości przykładów tekstowych, treść tekstów wprowadzanych do embedingów podczas szkolenia jest często pobierana automatycznie z sieci i nie jest w pełni analizowana. Korzystanie z niezweryfikowanych danych wejściowych może prowadzić do błędów związanych z osadzaniem słów. Na przykład możesz być zaskoczony, gdy odkryjesz, że osadzenie słów tworzy niewłaściwe skojarzenia między słowami. Powinieneś być świadomy takiego ryzyka i dokładnie przetestować swoją aplikację, ponieważ konsekwencją jest dodanie tych samych nieuczciwych uprzedzeń do tworzonych aplikacji do uczenia głębokiego. Na razie najpopularniejszymi osadzaniami słów powszechnie używanymi w aplikacjach do głębokiego uczenia się są

- Word2vec: Stworzony przez zespół badaczy kierowany przez Tomáša Mikolova w Google. Opiera się na dwóch płytkich warstwach sieci neuronowych, które próbują nauczyć się przewidywać słowo, znając słowa poprzedzające je i następujące po nim. Word2vec występuje w dwóch wersjach: opartej na czymś w rodzaju modelu worka słów (zwanym ciągłym workiem słów lub CBOW), który jest mniej wrażliwy na kolejność słów; a inny oparty na n-gramach (zwany ciągłym pomijaniem gramów), który jest bardziej wrażliwy na kolejność. Word2vec uczy się przewidywać słowo na podstawie jego kontekstu za pomocą hipotezy dystrybucji, co oznacza, że podobne słowa pojawiają się w podobnych kontekstach słów. Ucząc się, jakie słowa powinny pojawiać się w różnych kontekstach, Word2vec uwewnętrznia konteksty. Obie wersje nadają się do większości zastosowań, ale wersja z pominięciem grama jest w rzeczywistości lepsza w przedstawianiu nieczęstych słów.
- GloVe (Global Vectors): Opracowany jako projekt open source na Uniwersytecie Stanforda (https://nlp.stanford.edu/projects/glove/), podejście GloVe jest podobne do statystycznych metod lingwistycznych. Pobiera statystyki współwystępowania słowo-słowo z korpusu i redukuje wynikową macierz rzadką do gęstej za pomocą faktoryzacji macierzy, która jest metodą algebraiczną szeroko stosowaną w statystyce wielowymiarowej.
- fastText: Stworzony przez laboratorium AI Research (FAIR) Facebooka, fastText to osadzanie słów, dostępne w wielu językach, które działa z podsekwencjami słów zamiast pojedynczych słów. Dzieli słowo na wiele kawałków liter i osadza je. Ta technika ma ciekawe implikacje, ponieważ fastText oferuje lepszą reprezentację rzadkich słów (które często składają się z podciągów, które nie są rzadkie) i określa sposób wyświetlania błędnie napisanych słów. Możliwość obsługi błędów ortograficznych i błędów pozwala na efektywne wykorzystanie

osadzania z tekstem pochodzącym z sieci społecznościowych, e-maili i innych źródeł, z którymi ludzie zwykle nie używają sprawdzania pisowni.

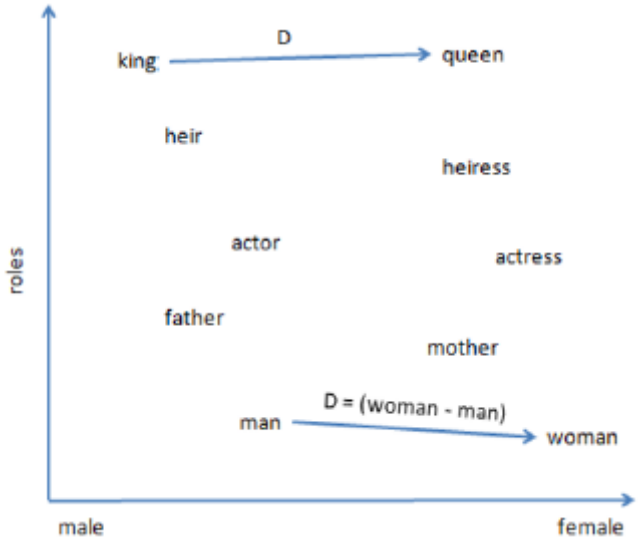
WYJAŚNIENIE DLACZEGO (KRÓL – MĘŻCZYZNA) + KOBIETA = KRÓLOWA

Osadzanie słów przekłada słowo na serię liczb reprezentujących jego pozycję w samym osadzeniu. Ta seria liczb to wektor słów. Zwykle składa się z około 300 wektorów (liczba wektorów używanych przez Google w ich modelu wyszkolonych w zestawie danych Google News), a sieci neuronowe używają go do lepszego i bardziej efektywnego przetwarzania informacji tekstowych. W rzeczywistości słowa o podobnym znaczeniu lub używane w podobnym kontekście mają podobne wektory słów; w związku z tym sieci neuronowe mogą łatwo rozpoznać słowa za pomocą podobnego znaczenia. Ponadto sieci neuronowe mogą pracować z analogiami, manipulując wektorami, co oznacza, że można uzyskać niesamowite wyniki, takie jak

• król – mężczyzna + kobieta = królowa

• paryż – francja + polska = warszawa

Może wydawać się magią, ale to prosta matematyka. Możesz zobaczyć, jak wszystko działa, patrząc na poniższy rysunek, który reprezentuje dwa wektory Word2vec.



Każdy wektor w Word2vec reprezentuje inną semantykę; może to być rodzaj żywności, jakość osoby, narodowość lub płeć. Istnieje wiele semantyk i nie są one z góry zdefiniowane; szkolenie z osadzania stworzyło je automatycznie na podstawie przedstawionych przykładów. Rysunek przedstawia dwa wektory z Word2vec: jeden reprezentujący jakość osoby ; inny reprezentujący płeć osoby. Pierwszy wektor definiuje role, zaczynając od króla i królowej z wyższymi wynikami, przechodząc przez aktora i aktorkę, a na końcu kończąc na niższych wynikach mężczyzny i kobiety. Jeśli dodasz ten wektor do wektora płci, zobaczysz, że warianty męskie i żeńskie rozdzielają się różnymi punktacjami na tym wektorze. Teraz, kiedy odejmujesz mężczyznę i dodajesz kobietę do króla, po prostu odsuwasz się od współrzędnych króla i przesuwasz wzdłuż wektora płci, aż osiągniesz pozycję królowej. Ta prosta sztuczka ze współrzędnymi, która nie implikuje żadnego zrozumienia słów przez Word2vec, jest możliwa, ponieważ wszystkie wektory osadzenia słów są zsynchronizowane, reprezentując znaczenie języka, i możesz w sposób sensowny przechodzić z jednej współrzędnej do drugiej zmieniali pojęcia w rozumowaniu.

Wykorzystanie AI do analizy nastrojów

Analiza sentymentu wywodzi się obliczeniowo z tekstu pisanego przy użyciu nastawienia pisarza (pozytywnego, negatywnego lub neutralnego) do tematu tekstu. Tego rodzaju analiza jest przydatna dla osób zajmujących się marketingiem i komunikacją, ponieważ pomaga im zrozumieć, co klienci i konsumenci myślą o produkcie lub usłudze, a tym samym działać odpowiednio (na przykład próbując odzyskać niezadowolonych klientów lub zdecydować się na inną strategię sprzedaży). Każdy przeprowadza analizę sentymentu. Na przykład podczas czytania tekstu ludzie w naturalny sposób starają się określić sentyment, który poruszył osobę, która go napisała. Jednak gdy liczba tekstów do przeczytania i zrozumienia jest zbyt duża, a tekst ciągle się kumuluje, jak w mediach społecznościowych i mailach klientów, ważna jest automatyzacja zadania. Nadchodzącym przykładem jest testowe uruchomienie RNN przy użyciu Keras i TensorFlow, które budują algorytm analizy sentymentu zdolny do klasyfikowania postaw wyrażonych w recenzji filmu. Dane to próbka zbioru danych IMDb, która zawiera 50 000 recenzji (podzielonych na pół między zestawy pociągowe i testowe) filmów, którym towarzyszy etykieta wyrażająca opinię na temat opinii (0=negatywna, 1=pozytywna). IMDb (https://www.imdb.com/) to duża internetowa baza danych zawierająca informacje o filmach, serialach telewizyjnych i grach wideo. Pierwotnie utrzymywany przez fanów, teraz jest prowadzony przez spółkę zależną Amazon. Na IMDb ludzie znajdują potrzebne informacje o swoim ulubionym programie, a także zamieszczają swoje komentarze lub piszą recenzję, którą mogą przeczytać inni odwiedzający. Keras oferuje do pobrania opakowanie dla danych IMDb. Przygotowujesz, tasujesz i układasz te dane w pociąg i zestaw testowy. Ten zestaw danych pojawia się wśród innych przydatnych zestawów danych na https://keras.io/datasets/. W szczególności dane tekstowe IMDb oferowane przez Keras są oczyszczone ze znaków interpunkcyjnych, znormalizowane do małych liter i przekształcone w wartości liczbowe. Każde słowo jest zakodowane w liczbę reprezentującą jego ranking częstotliwości. Najczęściej używane słowa mają małe liczby; rzadsze słowa mają wyższe liczby.

Na początek kod importuje funkcję imdb z Keras i używa jej do pobierania danych z Internetu (pobieranie około 17,5 MB). Parametry używane w przykładzie obejmują tylko 10 000 najpopularniejszych słów, a Keras powinien przetasować dane przy użyciu określonego losowego ziarna. (Znajomość ziarna umożliwia odtworzenie losowania w razie potrzeby). Funkcja zwraca dwa zestawy pociągów i testów, oba złożone z sekwencji tekstowych i wyniku tonacji.

```
from keras.datasets import imdb

top_words = 10000

((x_train, y_train),

(x_test, y_test)) = imdb.load_data(num_words=top_words,

seed=21)
```

Po zakończeniu poprzedniego kodu możesz sprawdzić liczbę przykładów za pomocą następującego kodu:

```
print("Training examples: %i" % len(x_train))

print("Test examples: %i" % len(x_test))
```

Po zapytaniu o liczbę przypadków dostępnych do wykorzystania w fazie uczenia i testowania sieci neuronowej, kod wyświetla odpowiedź w postaci 25 000 przykładów dla każdej fazy. (Ten zestaw danych jest stosunkowo mały w przypadku problemu językowego; najwyraźniej zestaw danych służy głównie do celów demonstracyjnych). Ponadto kod określa, czy zestaw danych jest zrównoważony, co oznacza, że ma prawie równą liczbę pozytywnych i negatywnych przykładów tonacji.

```
import numpy as np

print(np.unique(y_train, return_counts=True))
```

Wynik, array([12500, 12500]), potwierdza, że zbiór danych jest podzielony równomiernie na wyniki pozytywne i negatywne. Taka równowaga między klasami odpowiedzi wynika wyłącznie z demonstracyjnego charakteru zbioru danych. W prawdziwym świecie rzadko można znaleźć zrównoważone zbiory danych. Następnym krokiem jest utworzenie kilku słowników Pythona, które mogą konwertować między kodem używanym w zbiorze danych a prawdziwymi słowami. W rzeczywistości zestaw danych użyty w tym przykładzie jest wstępnie przetworzony i zawiera sekwencje liczb reprezentujących słowa, a nie same słowa. (Algorytmy LSTM i GRU, które można znaleźć w Keras, oczekują ciągów liczb jako liczb.)

```
word_to_id = {w:i+3 for w,i in imdb.get_word_index().items()}

id_to_word = {0:'<PAD>', 1:'<START>', 2:'<UNK>'}

id_to_word.update({i+3:w for w,i in imdb.get_word_index().

items()})

def convert_to_text(sequence):

return ' '.join([id_to_word[s] for s in sequence if s>=3])

print(convert_to_text(x_train[8]))
```

Poprzedni fragment kodu definiuje dwa słowniki konwersji (ze słów na kody numeryczne i odwrotnie) oraz funkcję, która tłumaczy przykłady zestawu danych na czytelny tekst. Jako przykład, kod wyświetla dziewiąty przykład: „ten film był jak wrak złego pociągu, tak okropny jak był. . .”. Z tego fragmentu można łatwo przewidzieć, że sentyment do tego filmu nie jest pozytywny. Słowa takie jak zły, zniszczony i okropny przekazują silne negatywne uczucia, co ułatwia odgadnięcie właściwego sentymentu. W tym przykładzie otrzymujesz sekwencje liczbowe i zamieniasz je z powrotem w słowa, ale przeciwieństwo jest powszechne. Zwykle otrzymujesz frazy złożone ze słów i przekształcasz je w sekwencje liczb całkowitych, które zasilają warstwę RNN. Keras oferuje wyspecjalizowaną funkcję, Tokenizer, która może to zrobić za Ciebie. Wykorzystuje metody fit_on_text, aby nauczyć się mapować słowa na liczby całkowite z danych treningowych, oraz texts_to_matrix, aby przekształcić tekst w sekwencję. Jednak w innych wyrażeniach możesz nie znaleźć takich odkrywczych słów. Uczucie jest wyrażane w bardziej subtelny lub pośredni sposób, a zrozumienie sentymentu na początku tekstu może nie być możliwe, ponieważ ujawniające zwroty i słowa mogą pojawić się znacznie później w dyskursie. Z tego powodu musisz również zdecydować, jaką część frazy chcesz przeanalizować. Tradycyjnie bierzesz początkową część tekstu i używasz jej jako reprezentatywnej dla całej recenzji. Czasami wystarczy kilka początkowych słów - na przykład pierwszych 50 słów - aby zrozumieć sens; czasami potrzebujesz więcej. Szczególnie długie teksty nie ujawniają wcześnie swojej orientacji. Dlatego to od Ciebie zależy, czy zrozumiesz rodzaj tekstu, z którym pracujesz, i zdecydujesz, ile słów chcesz przeanalizować za pomocą głębokiego uczenia się. Ten przykład uwzględnia tylko pierwsze 200 słów, co powinno wystarczyć. Zauważyłeś, że kod zaczyna podawać kod słowom zaczynającym się od cyfry 3, pozostawiając kody od 0 do 2. Niższe cyfry są używane do specjalnych znaczników, takich jak sygnalizacja początku frazy, wypełnianie pustych spacji w celu ustalenia sekwencji o określonej długości i zaznaczając słowa, które są wykluczone, ponieważ nie występują wystarczająco często. Ten przykład pokazuje tylko najczęstsze 10 000 słów. Używanie tagów do wskazywania początku, końca i

ważnych sytuacji to sztuczka, która działa w przypadku RNN, zwłaszcza w przypadku tłumaczenia maszynowego.

```
from keras.preprocessing.sequence import pad_sequences

max_pad = 200

x_train = pad_sequences(x_train,

maxlen=max_pad)

x_test = pad_sequences(x_test,

maxlen=max_pad)

print(x_train[0])
```

Używając funkcji pad_sequences firmy Keras z parametrem max_pad ustawionym na 200, kod pobiera pierwsze dwieście słów każdej recenzji. W przypadku, gdy przegląd zawiera mniej niż dwieście słów, tyle wartości zerowych, ile jest konieczne, poprzedza sekwencję, aby osiągnąć wymaganą liczbę elementów sekwencji. Przycinanie sekwencji do określonej długości i wypełnianie pustych przestrzeni wartościami zerowymi nazywa się wypełnianiem wejściowym, co jest ważną czynnością przetwarzania podczas korzystania z RNN, takich jak algorytmy głębokiego uczenia. Teraz kod projektuje architekturę:

```
from keras.models import Sequential

from keras.layers import Bidirectional, Dense, Dropout

from keras.layers import GlobalMaxPool1D, LSTM

from keras.layers.embeddings import Embedding

embedding_vector_length = 32

model = Sequential()

model.add(Embedding(top_words,

embedding_vector_length,

input_length=max_pad))

model.add(Bidirectional(LSTM(64, return_sequences=True)))

model.add(GlobalMaxPool1D())

model.add(Dense(16, activation="relu"))

model.add(Dense(1, activation="sigmoid"))

model.compile(loss='binary_crossentropy',

optimizer='adam',

metrics=['accuracy'])

print(model.summary())
```

Poprzedni fragment kodu definiuje kształt modelu uczenia głębokiego, w którym wykorzystuje kilka wyspecjalizowanych warstw do przetwarzania języka naturalnego firmy Keras. Przykład wymagał

również podsumowania modelu (polecenie model.summary()), aby określić, co dzieje się z architekturą przy użyciu różnych warstw neuronowych. Masz warstwę Osadzanie, która przekształca sekwencje liczbowe w gęste osadzanie słów. Ten rodzaj osadzania słów jest bardziej odpowiedni do uczenia się przez warstwę RNN, co omówiono w poprzednim akapicie tego rozdziału. Keras udostępnia warstwę osadzania, która nie tylko musi być pierwszą warstwą sieci, ale może realizować dwa zadania:

- Zastosowanie wstępnie wytrenowanego osadzania słów (takich jak Word2vec lub GloVe) do wprowadzania sekwencji. Wystarczy przekazać macierz zawierającą osadzanie do jej wag parametrów.
- Tworzenie osadzonego słowa od podstaw na podstawie otrzymanych danych wejściowych.

W tym drugim przypadku osadzanie musi tylko wiedzieć:

- input_dim: Rozmiar słownika oczekiwany od danych
- output_dim: Wielkość przestrzeni osadzenia, która zostanie wytworzona (tzw. wymiary)
- input_length: oczekiwany rozmiar sekwencji

Po ustaleniu parametrów, Embedding znajdzie lepsze wagi, aby podczas treningu przekształcić sekwencje w gęstą matrycę. Rozmiar gęstej matrycy wynika z długości sekwencji i wymiarowości osadzenia. Jeśli korzystasz z warstwy The Embedding dostarczonej przez Keras, musisz pamiętać, że funkcja udostępnia tylko macierz wagową wielkości słownika przez wymiar pożądanego osadzenia. Odwzorowuje słowa na kolumny macierzy, a następnie dostosowuje wagi macierzy do podanych przykładów. To rozwiązanie, choć praktyczne dla niestandardowych problemów językowych, nie jest analogiczne do omawianych wcześniej osadzania słów, które są szkolone w inny sposób i na milionach przykładów. W przykładzie zastosowano zawijanie dwukierunkowe — warstwę LSTM składającą się z 64 komórek. Dwukierunkowy przekształca normalną warstwę LSTM, podwajając ją: po pierwszej stronie stosuje normalną sekwencję wprowadzonych danych; na drugim przechodzi odwrotność sekwencji. Stosujesz to podejście, ponieważ czasami używasz słów w innej kolejności, a budowanie warstwy dwukierunkowej wyłapuje każdy wzorzec słów, bez względu na kolejność. Implementacja Keras jest rzeczywiście prosta: po prostu stosujesz ją jako funkcję na warstwie, którą chcesz renderować dwukierunkowo. Dwukierunkowy LSTM jest ustawiony na sekwencje zwracane (return_sequences=True); oznacza to, że dla każdej komórki zwraca wynik dostarczony po obejrzeniu każdego elementu sekwencji. Wynikiem dla każdej sekwencji jest macierz wyjściowa 200 x 128, gdzie 200 to liczba elementów sekwencji, a 128 to liczba komórek LSTM użytych w warstwie. Ta technika zapobiega pobieraniu przez RNN ostatniego wyniku każdej komórki LSTM. Wskazówki dotyczące sentymentu tekstu mogą w rzeczywistości pojawić się w dowolnym miejscu osadzonej sekwencji słów. Krótko mówiąc, ważne jest, aby nie brać ostatniego wyniku każdej komórki, ale raczej najlepszy z niej. Dlatego kod opiera się na następnej warstwie, GlobalMaxPool1D, aby sprawdzić każdą sekwencję wyników dostarczanych przez każdą komórkę LSTM i zachować tylko maksymalny wynik. Powinno to zapewnić, że przykład wybierze najsilniejszy sygnał z każdej komórki LSTM, która, miejmy nadzieję, jest wyspecjalizowana przez swoje szkolenie w wybieraniu znaczących sygnałów. Po przefiltrowaniu sygnałów neuronowych przykład ma warstwę 128 wyjść, po jednym dla każdej komórki LSTM. Kod redukuje i miesza sygnały przy użyciu kolejnej gęstej warstwy 16 neuronów z aktywacją ReLU (w ten sposób przechodzą tylko pozytywne sygnały; zobacz sekcję „Wybór właściwej funkcji aktywacji” w rozdziale 8, aby uzyskać szczegółowe informacje). Architektura kończy się końcowym węzłem wykorzystującym aktywację sigmoid, która ściśnie wyniki do zakresu 0–1 i sprawi, że będą wyglądały jak prawdopodobieństwa. Po zdefiniowaniu architektury możesz teraz trenować sieć. Wystarczą trzy epoki (trzykrotne przekazanie danych przez sieć, aby nauczyć się wzorców). Kod używa za każdym razem partii 256 recenzji, co pozwala sieci zobaczyć wystarczającą różnorodność słów i sentymentów

za każdym razem przed aktualizacją wag przy użyciu propagacji wstecznej. Wreszcie kod koncentruje się na wynikach dostarczanych przez dane walidacyjne (które nie są częścią danych uczących). Uzyskanie dobrego wyniku z danych walidacyjnych oznacza, że sieć neuronowa prawidłowo przetwarza dane wejściowe. Kod informuje o danych walidacyjnych tuż po zakończeniu każdej epoki

history = model.fit(x_train, y_train,

validation_data=(x_test, y_test),

epochs=3, batch_size=256)

Uzyskanie wyników zajmuje trochę czasu, ale jeśli korzystasz z procesora graficznego, zakończy się w czasie, jaki zajmuje wypicie filiżanki kawy. W tym momencie możesz ocenić wyniki, ponownie korzystając z danych walidacyjnych. (Wyniki nie powinny mieć żadnych niespodzianek ani różnic w stosunku do tego, co kod zgłaszał podczas szkolenia.)

loss, metric = model.evaluate(x_test, y_test, verbose=0)

print("Test accuracy: %0.3f" % metric)

Dokładność końcowa, czyli odsetek poprawnych odpowiedzi z głębokiej sieci neuronowej, wyniesie około 85-86 proc. Wynik będzie się nieznacznie zmieniał za każdym razem, gdy przeprowadzisz eksperyment z powodu randomizacji podczas budowania sieci neuronowej

To zupełnie normalne, biorąc pod uwagę niewielki rozmiar danych, z którymi pracujesz. Jeśli zaczniesz z odpowiednimi szczęśliwymi ciężarkami, nauka będzie łatwiejsza podczas tak krótkiej sesji treningowej. Ostatecznie Twoja sieć jest analizatorem nastrojów, który może prawidłowo odgadnąć nastroje wyrażone w recenzji filmu w około 85 procentach przypadków. Mając jeszcze więcej danych treningowych i bardziej wyrafinowane architektury neuronowe, możesz uzyskać jeszcze bardziej imponujące wyniki. W marketingu podobne narzędzie służy do automatyzacji wielu procesów wymagających czytania tekstu i podejmowania działań. Ponownie, możesz połączyć taką sieć z siecią neuronową, która nasłuchuje głosu i zamienia go w tekst. (To kolejna aplikacja RNN, która teraz obsługuje Alexę, Siri, Google Voice i wiele innych osobistych asystentów.) Przejście umożliwia aplikacji zrozumienie sentymentu nawet w wypowiedziach głosowych, takich jak rozmowa telefoniczna od klienta.

Generowanie muzyki i sztuk wizualnych

W Internecie można znaleźć wiele dyskusji na temat tego, czy komputery mogą być kreatywne dzięki zastosowaniu głębokiego uczenia się. Dialog dotyczy samej istoty tego, co to znaczy być kreatywnym. Filozofowie i inni dyskutowali na ten temat bez końca w historii ludzkości, nie dochodząc do konkluzji, co dokładnie oznacza twórczość. Jednak, aby zapewnić podstawę do dyskusji, ta książka definiuje kreatywność jako umiejętność definiowania nowych pomysłów, wzorców, relacji i tak dalej. Nacisk kładziony jest na nowości: oryginalność, progresywność i wyobraźnię, które zapewniają ludzie. Nie obejmuje kopiowania czyjegoś stylu i nazywania go swoim. Oczywiście ta definicja prawie na pewno wzbudzi irytację niektórych, podczas gdy inni przyjmą akceptujące skinienia głowy, ale aby dyskusja w ogóle działała, potrzebna jest definicja. Pamiętaj, ta definicja nie wyklucza kreatywności nieludzi. Na przykład, niektórzy ludzie mogą postarać się o kreatywne małpy .Ta Część pomoże ci zrozumieć, w jaki sposób kreatywność i komputery mogą łączyć się w fascynującą współpracę. Po pierwsze, musisz wziąć pod uwagę, że komputery opierają się na matematyce, aby zrobić wszystko, a sztuka i muzyka nie są wyjątkiem. Komputer może przenieść istniejącą sztukę lub wzory muzyczne do sieci neuronowej i wykorzystać wynik do wygenerowania czegoś, co wygląda na nowe, ale w rzeczywistości opiera się na istniejącym wzorcu. Jednak wraz z tym odkryciem drugą kwestią jest to, że człowiek zaprojektował algorytm używany do przeprowadzenia statystycznej analizy wzoru, a następnie wytworzenia nowej sztuki. Innymi słowy, komputer sam nie wykonał tego zadania; polegał na człowieku, aby zapewnić środki do wykonania zadania. Co więcej, człowiek zdecyduje, jaki styl naśladować i określi, jaki rodzaj twórczości może się podobać estetycznie. Krótko mówiąc, komputer staje się narzędziem w rękach wyjątkowo inteligentnego człowieka, który automatyzuje proces tworzenia czegoś, co można by uznać za nowe, ale tak nie jest. W ramach procesu definiowania, w jaki sposób niektórzy mogą postrzegać komputer jako kreatywny, rozdział ten określa również, w jaki sposób komputery naśladują ustalony styl. Przekonasz się na własne oczy, że głębokie uczenie się opiera się na matematyce, aby wykonać zadanie na ogół nie związane z matematyką. Artysta lub muzyk nie polega na obliczeniach, aby stworzyć coś nowego, ale może polegać na obliczeniach, aby zobaczyć, jak inni wykonali swoje zadanie. Kiedy artysta lub muzyk wykorzystuje matematykę do studiowania innego stylu, proces ten nazywa się uczeniem się, a nie tworzeniem. Oczywiście cała ta książka dotyczy tego, jak uczenie głębokie wykonuje zadania związane z uczeniem się, a nawet ten proces znacznie różni się od tego, jak uczą się ludzie.

Nauka naśladowania sztuki i życia

Prawdopodobnie widziałeś ciekawe wizje sztuki AI. Sztuka niezaprzeczalnie ma walor estetyczny. W rzeczywistości artykuł wspomina, że Christie′s, jeden z najsłynniejszych domów aukcyjnych na świecie, pierwotnie spodziewał się sprzedać dzieło sztuki za 7000 do 10 000 USD, ale w rzeczywistości sprzedano je za 432 000 USD, według Guardiana i New York Times. Tak więc rodzaj sztuki jest nie tylko atrakcyjny, ale może również generować dużo pieniędzy. Jednak w każdej bezstronnej historii, którą czytasz, pozostaje pytanie, czy sztuka AI rzeczywiście jest sztuką. Poniższe sekcje pomagają zrozumieć, że generowanie komputerów nie ma związku z kreatywnością - przekłada się to na niesamowite algorytmy wykorzystujące najnowsze statystyki.

Przenoszenie stylu artystycznego

Jednym z wyróżników sztuki jest styl artystyczny. Nawet jeśli ktoś robi zdjęcie i wyświetla je jako sztukę, sposób, w jaki zdjęcie jest robione, przetwarzane i opcjonalnie poprawiane, określa szczególny styl. W wielu przypadkach, w zależności od umiejętności artysty, nie można nawet powiedzieć, że patrzysz na fotografię ze względu na jej elementy artystyczne. Niektórzy artyści stają się tak sławni dzięki swojemu szczególnemu stylowi, że inni poświęcają czas na dogłębne przestudiowanie go, aby poprawić własną technikę. Na przykład często naśladuje się unikalny styl Vincenta van Gogha. Styl Van Gogha – jego

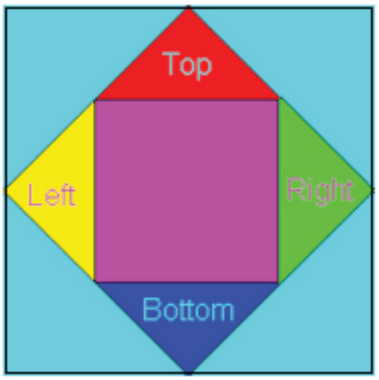
użycie kolorów, metod, mediów, tematyki i wielu innych rozważań – wymaga intensywnych badań, aby ludzie mogli się powtórzyć. Ludzie improwizują, więc przyrostek przymiotnikowy esque często pojawia się jako deskryptor stylu danej osoby. Krytyk mógłby powiedzieć, że konkretny artysta posługuje się metodologią van Goghesque. Aby tworzyć dzieła sztuki, komputer wykorzystuje określony styl artystyczny do modyfikowania wyglądu obrazu źródłowego. W przeciwieństwie do człowieka, komputer może doskonale odtworzyć określony styl, mając wystarczająco spójnych przykładów. Oczywiście można było stworzyć rodzaj mieszanego stylu, posługując się przykładami z różnych okresów w życiu artysty. Chodzi o to, że komputer nie tworzy nowego stylu ani nie improwizuje. Obraz źródłowy również nie jest nowy. Podczas pracy z komputerem widzisz doskonale skopiowany styl i doskonale skopiowany obraz źródłowy, a następnie przenosisz styl do obrazu źródłowego, aby utworzyć coś, co wygląda trochę jak oba. Proces używany do przeniesienia stylu do obrazu źródłowego i uzyskania wyniku jest złożony i generuje wiele dyskusji. Na przykład ważne jest rozważenie, gdzie kończy się kod źródłowy i zaczynają się elementy, takie jak szkolenie. Pamiętaj, że cała dyskusja skupia się na ludziach, którzy tworzą kod i przeprowadzają szkolenie komputera; sam komputer nie bierze udziału w dyskusji, ponieważ komputer po prostu analizuje liczby.

INNE RODZAJE SZTUKI GENEROWANEJ

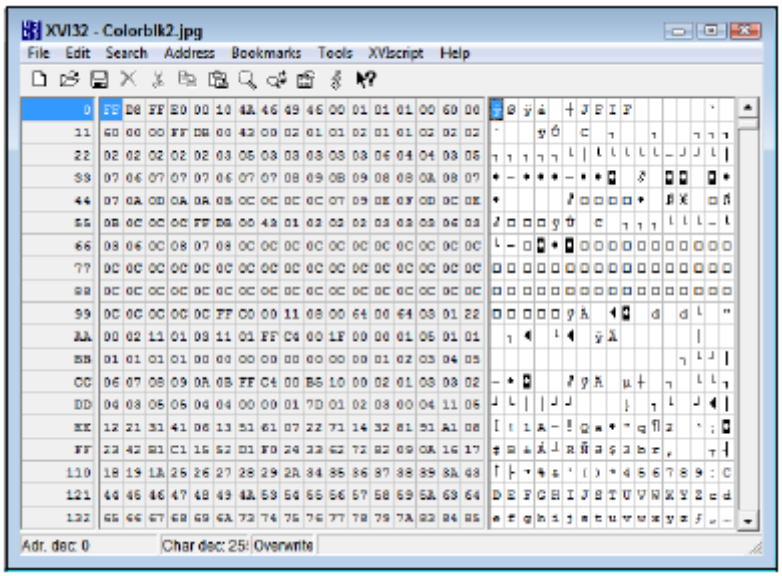
Należy pamiętać, że omawiamy szczególny rodzaj sztuki komputerowej - rodzaj generowany przez sieć głębokiego uczenia się. Możesz znaleźć wszelkiego rodzaju inne dzieła sztuki generowane komputerowo, które niekoniecznie opierają się na głębokim uczeniu się. Jednym z wcześniejszych przykładów generowanej sztuki jest fraktal, utworzony za pomocą równania. Pierwszym z tych fraktali jest zbiór Mandelbrota stworzony w 1980 roku przez polskiego matematyka Benoita B. Mandelbrota. Niektóre fraktale są dziś całkiem piękne, a nawet zawierają elementy ze świata rzeczywistego. Mimo to twórczość nie należy do komputera, który po prostu przetwarza liczby, ale do matematyka lub artysty, który projektuje algorytm używany do generowania fraktala. Kolejnym krokiem w generowanej sztuce jest Computer Generated Imagery (CGI). Prawdopodobnie widziałeś niesamowite przykłady grafiki CGI w filmach, ale dziś pojawia się ona prawie wszędzie. Niektórzy ograniczają CGI do grafiki 3D, a niektórzy do dynamicznej grafiki 3D, takiej jak używana w grach i filmach video. Bez względu na to, jakie ograniczenia nałożysz na grafikę CGI, proces jest zasadniczo taki sam. Artysta decyduje się na serię przekształceń, aby stworzyć efekty na ekranie komputera, takie jak woda, która wygląda na mokrą i mgłę, która wygląda na mglistą. CGI widzi również zastosowanie w budowaniu modeli opartych na projektach, takich jak rysunki architektoniczne . Modele te pomogą Ci zobrazować, jak będzie wyglądał gotowy produkt na długo przed obróceniem pierwszej łopaty ziemi. Jednak w końcu widzisz kreatywność artysty, architekta, matematyka lub innej osoby, która mówi komputerowi, aby wykonał różnego rodzaju obliczenia, aby przekształcić projekt w coś, co wygląda naprawdę. Komputer nic z tego nie rozumie.

Sprowadzenie problemu do statystyk

Komputery właściwie nic nie widzą. Ktoś wykonuje cyfrowy obraz obiektu ze świata rzeczywistego lub tworzy fantazyjny rysunek, taki jak na rysunku ,



a każdy piksel na tym obrazie pojawia się jako krotki liczb reprezentujących wartości czerwone, niebieskie i zielone każdego piksela, jak pokazano na rysunku.



Z kolei te liczby są tym, z czym komputer wchodzi w interakcję za pomocą algorytmu. Komputer nie rozumie, że liczby tworzą krotkę - to ludzka konwencja. Wie tylko, że algorytm definiuje operacje, które muszą zajść na szeregu liczb. Krótko mówiąc, sztuka staje się kwestią manipulowania liczbami przy użyciu różnych metod, w tym statystyki.

Głębokie uczenie opiera się na szeregu algorytmów, które manipulują pikselami w rysunku źródłowym na różne sposoby, aby odzwierciedlić konkretny styl, którego chcesz użyć. W rzeczywistości można znaleźć oszałamiający wachlarz takich algorytmów, ponieważ wydaje się, że każdy ma inny pomysł na to, jak zmusić komputer do tworzenia określonych rodzajów sztuki. Chodzi o to, że wszystkie te metody opierają się na algorytmach, które działają na serii liczb, aby wykonać zadanie; komputer nigdy nie bierze pędzla do ręki, aby stworzyć coś nowego. Wydaje się jednak, że dwie metody napędzają obecne strategie:

- Konwolucyjne sieci neuronowe (CNN): przegląd w rozdziale 10; zobacz także sekcję „Definiowanie nowego utworu na podstawie jednego artysty” w dalszej części tego rozdziału, aby zapoznać się z perspektywą artystyczną
- Generative Adversarial Networks (GAN): przegląd w rozdziale 16; sprawdź także sekcję „Wizualizacja, jak sieci neuronowe śnią”, w dalszej części tego rozdziału, ponownie dla artystycznej perspektywy

Zrozumienie, że głębokie uczenie się nie tworzy

W przypadku sztuki tworzonej przez głębokie uczenie obrazy są pożyczane, komputer w ogóle ich nie rozumie, a komputer polega na algorytmach, które wykonują zadanie modyfikacji obrazów. Głębokie uczenie nie wybiera nawet metody poznawania obrazów – robi to człowiek. Krótko mówiąc, głębokie uczenie się to interesująca metoda manipulowania obrazami stworzonymi przez kogoś innego przy użyciu stylu, który również stworzyła inna osoba. To, czy głębokie uczenie się może coś stworzyć, nie jest prawdziwym pytaniem. Ważnym pytaniem jest to, czy ludzie potrafią docenić skutki głębokich wyników uczenia się. Pomimo niezdolności do zrozumienia lub tworzenia, głębokie uczenie może przynieść niesamowite rezultaty. W związku z tym kreatywność najlepiej pozostawić ludziom, ale głębokie uczenie się może dać każdemu ekspresyjne narzędzie, nawet osobom, które nie są artystami. Na przykład możesz użyć głębokiego uczenia się, aby stworzyć wersję ukochanej osoby van Gogha do powieszenia na ścianie. Fakt, że uczestniczyłeś w procesie i masz coś, co wygląda na profesjonalnie narysowane, jest kwestią do rozważenia - nie czy komputer jest kreatywny.

Naśladowanie artysty

Głębokie uczenie pomaga naśladować konkretnego artystę. Możesz naśladować dowolnego artystę, którego chcesz, ponieważ komputer nie rozumie nic na temat stylu ani rysunku. Algorytm głębokiego uczenia wiernie odtworzy styl na podstawie wprowadzonych danych. W związku z tym naśladowanie jest elastycznym sposobem tworzenia określonych danych wyjściowych, jak opisano w poniższych sekcjach.

Definiowanie nowego utworu w oparciu o jednego artystę

Splotowe sieci neuronowe (CNN) pojawiają się w wielu zastosowaniach w aplikacjach głębokiego uczenia. Na przykład są używane do autonomicznych samochodów i systemów rozpoznawania twarzy. Rozdział 10 zawiera kilka dodatkowych przykładów tego, jak CNN wykonują swoją pracę, ale chodzi o to, że CNN może dobrze wykonywać zadania związane z rozpoznawaniem po odpowiednim przeszkoleniu. Co ciekawe, stacje CNN sprawdzają się szczególnie dobrze w rozpoznawaniu stylu artystycznego. Możesz więc połączyć dwa dzieła sztuki w jeden kawałek. Jednak te dwie części dostarczają dwa różne rodzaje danych wejściowych dla CNN:

- Treść: obraz, który definiuje żądany wynik. Na przykład, jeśli podasz obraz treści przedstawiający kota, wynik będzie wyglądał jak kot. Nie będzie to ten sam kot, z którym zacząłeś, ale treść określa pożądany efekt w odniesieniu do tego, co zobaczy człowiek.
- Styl: Obraz definiujący pożądaną modyfikację. Na przykład, jeśli podasz przykład obrazu van Gogha, wynik będzie odzwierciedlał ten styl.

Ogólnie rzecz biorąc, widzisz CNN, które opierają się na jednym obrazie treści i jednym obrazie stylu. Użycie tylko dwóch obrazów, takich jak ten, pozwala zobaczyć, jak treść i styl współdziałają w celu uzyskania określonego wyniku. Przykład na https://medium.com/mlreview/making-ai-art-with-style-transfer-using-keras-8bb5fa44b216 przedstawia metodę łączenia w ten sposób dwóch obrazów. Oczywiście musisz zdecydować, jak połączyć obrazy. W rzeczywistości tutaj w grę wchodzą statystyki głębokiego uczenia się. Aby wykonać to zadanie, użyj transferu stylu neuronowego, jak opisano w artykule „ Styl algorytmu neuronowy sztuki" Leona A. Gatysa, Alexandra S. Eckera i Matthiasa Bethge.

Algorytm działa z następującymi rodzajami obrazów: obraz treści, który przedstawia obiekt, który chcesz reprezentować; obraz stylu, który zapewnia styl artystyczny, który chcesz naśladować; oraz obraz wejściowy, który jest obrazem do przekształcenia. Obraz wejściowy jest zwykle losowym obrazem lub tym samym obrazem co obraz zawartości. Przeniesienie stylu oznacza zachowanie treści

(to znaczy, jeśli zaczynasz od zdjęcia psa, wynik nadal będzie przedstawiał psa). Jednak przekształcony obraz wejściowy jest bliższy obrazowi stylu w prezentacji. Algorytm, którego używasz, zdefiniuje dwie miary straty:

- Utrata treści: Określa ilość oryginalnego obrazu używanego przez CNN do dostarczania danych wyjściowych. Większa strata oznacza, że wynik będzie lepiej odzwierciedlał styl, który zapewniasz. Możesz jednak dojść do punktu, w którym strata jest tak duża, że nie możesz już zobaczyć treści.
- Utrata stylu: określa sposób, w jaki styl jest stosowany do treści. Wyższy poziom strat oznacza, że treść zachowuje więcej swojego oryginalnego stylu. Utrata stylu musi być na tyle niska, abyś mógł otrzymać nowe dzieło sztuki, które odzwierciedla pożądany styl.

Posiadanie tylko dwóch obrazów nie pozwala na intensywne szkolenie, więc korzystasz z przeszkolonej sieci uczenia głębokiego, takiej jak VGG-19 (zwycięzca wyzwania ImageNet z 2014 r. stworzonego przez Visual Geometry Group, VGG, na Uniwersytecie Oksfordzkim). Wstępnie przeszkolona sieć głębokiego uczenia już wie, jak przetworzyć obraz na cechy obrazu o różnej złożoności. Algorytm przesyłania stylu neuronowego wybiera CNN VGG-19, z wyłączeniem końcowych w pełni połączonych warstw. W ten sposób masz sieć, która działa jako filtr przetwarzania obrazów. Gdy wysyłasz obraz, VGG-19 przekształca go w reprezentację sieci neuronowej, która może być zupełnie inna niż oryginał. Jeśli jednak jako filtrów obrazu używasz tylko górnych warstw sieci, sieć przekształca wynikowy obraz, ale nie zmienia go całkowicie.

Wykorzystując takie transformacyjne właściwości sieci neuronowej, styl transferu neuronowego nie wykorzystuje wszystkich splotów w VGG-19. Zamiast tego monitoruje je za pomocą dwóch miar strat, aby upewnić się, że pomimo przekształceń zastosowanych w obrazie sieć zachowuje treść i stosuje styl. W ten sposób, gdy kilka razy prześlesz obraz wejściowy przez VGG-19, jego wagi dostosowują się, aby wykonać podwójne zadanie zachowania treści i uczenia się stylu. Po kilku iteracjach, które w rzeczywistości wymagają wielu obliczeń i aktualizacji wagi, sieć przekształca obraz wejściowy w oczekiwany obraz i styl graficzny. Często widzisz wyjście z CNN określane jako pastisz. To fantazyjne słowo, które ogólnie oznacza dzieło artystyczne złożone z elementów zapożyczonych z motywów lub technik innych artystów. Biorąc pod uwagę naturę sztuki głębokiego uczenia się, termin ten jest odpowiedni.

Łączenie stylów w celu tworzenia nowej sztuki

Jeśli naprawdę chcesz wymyślić, możesz stworzyć pastisz oparty na wielu obrazach stylów. Na przykład możesz wytrenować CNN, używając wielu prac Moneta, aby pastisz wyglądał bardziej jak kawałek Moneta w ogóle. Oczywiście równie łatwo można było połączyć style wielu malarzy impresjonistycznych, aby stworzyć to, co się wydaje być wyjątkowym dziełem sztuki, które ogólnie odzwierciedla styl impresjonistyczny.

Wizualizacja, jak marzą sieci neuronowe

Korzystanie z CNN jest zasadniczo procesem ręcznym w odniesieniu do wyboru funkcji strat. Sukces lub porażka CNN zależy od tego, jak ludzie ustalają różne wartości. GAN ma inne podejście. Opiera się na dwóch interaktywnych, głębokich sieciach, które automatycznie dostosowują wartości, aby zapewnić lepsze wyniki. Możesz zobaczyć te dwie głębokie sieci o tych nazwach:

- Generator: Tworzy obraz na podstawie wprowadzonych danych. Obraz musi zachowywać oryginalną treść, ale z odpowiednim poziomem stylu, aby stworzyć pastisz trudny do odróżnienia od oryginału.

- Dyskryminator: Określa, czy dane wyjściowe generatora są wystarczająco rzeczywiste, aby mogły zostać uznane za oryginał. Jeśli nie, dyskryminator zapewnia informację zwrotną, informując generator, co jest nie tak z pastiszem.

Aby ta konfiguracja działała, w rzeczywistości trenujesz dwa modele: jeden dla generatora, a drugi dla dyskryminatora. Obydwa działają wspólnie, przy czym generator tworzy nowe próbki, a dyskryminator mówi generatorowi, co jest nie tak z każdą próbką. Proces toczy się tam i z powrotem między generatorem a dyskryminatorem, aż pastisz osiągnie określony poziom doskonałości. W rozdziale 16 znajdziesz jeszcze bardziej szczegółowe wyjaśnienie działania sieci GAN.

Takie podejście jest korzystne, ponieważ zapewnia wyższy poziom automatyzacji i większe prawdopodobieństwo uzyskania dobrych wyników niż przy użyciu CNN. Wadą jest to, że to podejście również wymaga dużo czasu na wdrożenie, a wymagania dotyczące przetwarzania są znacznie większe. W związku z tym korzystanie z podejścia CNN jest często lepsze, aby osiągnąć wystarczająco dobry wynik.

Korzystanie z sieci do komponowania muzyki

Ta Część skupia się głównie na sztuce wizualnej, ponieważ możesz łatwo ocenić subtelne zmiany, które w niej zachodzą. Jednak te same techniki działają również w przypadku muzyki. Możesz używać CNN i GAN do tworzenia muzyki w oparciu o określony styl. Komputery nie widzą sztuki wizualnej ani nie słyszą muzyki. Tony muzyczne stają się liczbami, którymi manipuluje komputer, tak jak manipuluje liczbami powiązanymi z pikselami. Komputer w ogóle nie widzi różnicy. Jednak głębokie uczenie wykrywa różnicę. Tak, używasz tych samych algorytmów do muzyki, co do sztuk wizualnych, ale ustawienia, których używasz, są inne, a trening również jest wyjątkowy. Ponadto niektóre źródła podają, że trening do muzyki jest o wiele trudniejszy niż do sztuki. Oczywiście część trudności wynika z różnic między ludźmi słuchającymi muzyki. Jako grupa, ludziom wydaje się, że trudno jest zdefiniować estetycznie przyjemną muzykę, a nawet ludzie, którzy lubią określony styl lub konkretnych artystów, rzadko lubią wszystko, co ci artyści tworzą. Pod pewnymi względami narzędzia używane do komponowania muzyki przy użyciu sztucznej inteligencji są bardziej sformalizowane i dojrzałe niż te używane w sztuce wizualnej. Nie oznacza to, że narzędzia do komponowania muzyki zawsze dają świetne rezultaty, ale oznacza to, że możesz łatwo kupić pakiet do wykonywania zadań związanych z komponowaniem muzyki. Oto dwie najpopularniejsze oferty dzisiaj:

- Ampermusic
- Jukedeck

Kompozycja muzyczna AI różni się od generowania sztuk wizualnych, ponieważ narzędzia muzyczne są dostępne od dłuższego czasu. Nieżyjący już autor piosenek i wykonawca David Bowie używał starszej aplikacji o nazwie Verbasizer w 1995 roku do pomocy w jego pracy. Kluczową ideą jest tutaj to, że to narzędzie wspomagało, a nie produkowało pracę. Człowiek jest talentem twórczym; AI służy jako kreatywne narzędzie do tworzenia lepszej muzyki. W związku z tym muzyka nabiera charakteru współpracy, zamiast stawiać na pierwszym planie sztuczną inteligencję.

Budowanie generatywnych sieci przeciwników

Głębokie uczenie stało się popularną technologią, a nowe badania cały czas dostarczają coraz bardziej imponujących odkryć. Odkrycia zawsze pojawiają się w jeszcze szybszym tempie podczas konferencji Neural Information Processing Systems (NeurIPS), która służy jako scena dla wszystkiego, co dotyczy głębokiego uczenia się. Konferencja odbywa się co roku w innym miejscu na świecie. Konferencja zawsze udostępnia ludziom nowe technologie, ale kilka dziedzin przyciągnęło całą uwagę. Wśród imponującej różnorodności aplikacji i nowych technologii związanych z głębokim uczeniem przedstawionych ostatnio na konferencji, oto te, na które należy zwrócić największą uwagę: Przetwarzanie języka naturalnego (zwłaszcza w przypadku wstępnie przeszkolonych osadzeń, takich jak BERT); Uczenie się przez wzmacnianie; oraz generatywne sieci przeciwstawne (GAN). GAN to myślenie poza schematem. Yann LeCun, obecnie dyrektor Facebook AI, określa go jako „najciekawszy pomysł ostatnich dziesięciu lat w uczeniu maszynowym". Ta Część opisuje, czym są GAN i pokazuje, w jaki sposób są one w stanie generować nowe dane, zwłaszcza obrazy, z już istniejących. Część uzupełnia przegląd sieci GAN, budując sieć przy użyciu Keras i TensorFlow Po zobaczeniu GAN w akcji, część przechodzi do omówienia najciekawszych osiągnięć i osiągnięć GAN. Oszczędź czas i błędy związane z ręcznym wpisywaniem kodu.

Tworzenie konkurencyjnych sieci

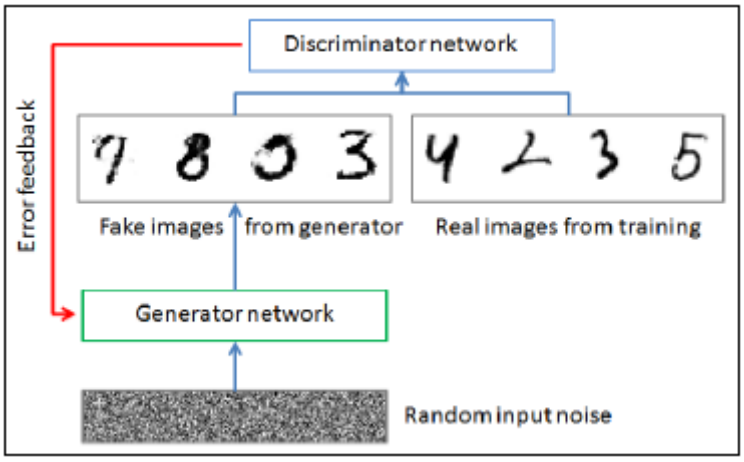
W 2014 r. w Department d'informatique et de recherche opérationnelle na Uniwersytecie w Montrealu Ian Goodfellow i inni badacze (wśród nich Yoshua Bengio, jeden z najbardziej znanych kanadyjskich naukowców zajmujących się sztucznymi sieciami neuronowymi i głębokim uczeniem) opublikowali pierwszy artykuł na GAN. W kolejnych miesiącach artykuł przyciągnął uwagę i został uznany za innowacyjny ze względu na proponowane połączenie głębokiego uczenia się i teorii gier. Pomysł stał się powszechny ze względu na jego dostępność jeśli chodzi o architekturę sieci neuronowej: Możesz wytrenować działający GAN przy użyciu standardowego komputera. (Technika działa lepiej, jeśli można zainwestować dużo mocy obliczeniowej). W przeciwieństwie do innych sieci neuronowych głębokiego uczenia, które klasyfikują obrazy lub sekwencje, specjalnością sieci GAN jest ich zdolność do generowania nowych danych poprzez czerpanie inspiracji z danych treningowych. Ta zdolność staje się szczególnie imponująca, gdy mamy do czynienia z danymi obrazu, ponieważ dobrze wyszkolone GAN mogą generować nowe dzieła sztuki, które ludzie sprzedają na aukcjach (takie jak dzieła sztuki sprzedane w Christie's za prawie pół miliona dolarów). Wyczyn ten jest jeszcze bardziej niesamowity, ponieważ wcześniejsze wyniki uzyskane przy użyciu innych technik matematycznych i statystycznych były dalekie od wiarygodności i użyteczności.

Znalezienie klucza w konkursie

Nazwa GAN zawiera w sobie termin adwersarz, ponieważ kluczową ideą GAN jest rywalizacja między dwiema sieciami, które grają przeciwko sobie jako adwersarze. Ian Goodfellow, główny autor oryginalnego artykułu o GAN, użył prostej metafory, aby opisać, jak wszystko działa. Goodfellow opisał ten proces jako niekończące się wyzwanie między fałszerzem a detektywem: fałszerz musi stworzyć fałszywe dzieło sztuki, kopiując prawdziwe arcydzieło sztuki, więc zaczyna coś malować. Gdy fałszerz zakończy fałszywy obraz, detektyw bada go i decyduje, czy fałszerz stworzył prawdziwe dzieło sztuki, czy po prostu podróbkę. Jeśli detektyw zobaczy podróbkę, fałszerz otrzymuje powiadomienie, że coś jest nie tak z pracą (ale nie gdzie leży wina). Kiedy fałszerz pokazuje, że sztuka jest prawdziwa pomimo negatywnej opinii detektywa, detektyw otrzymuje powiadomienie o błędzie i zmienia technikę wykrywania, aby uniknąć niepowodzenia podczas następnej próby. W miarę jak fałszerz kontynuuje próby oszukania detektywa, zarówno fałszerz, jak i detektyw zdobywają coraz większą wiedzę o swoich

obowiązkach. Z biegiem czasu sztuka wytworzona przez fałszerza staje się niezwykle wysokiej jakości i jest prawie nie do odróżnienia od prawdziwej rzeczy, chyba że ktoś ma oko eksperta.

Rysunek



ilustruje historię sieci GAN jako prostego schematu, w którym wejścia i architektury neuronowe współdziałają ze sobą w zamkniętej pętli wzajemnych sprzężeń zwrotnych. Sieć generatorów pełni rolę fałszerza, a sieć dyskryminacyjna - detektywa. GAN używają terminu dyskryminator ze względu na podobieństwo przeznaczenia do obwodów elektronicznych, które akceptują lub odrzucają sygnały na podstawie ich właściwości. Dyskryminator w GAN akceptuje (niesłusznie) lub odrzuca (poprawnie) pracę stworzoną przez generator. Interesującym aspektem tej architektury jest to, że generator nigdy nie widzi ani jednego przykładu uczącego. Tylko dyskryminator ma dostęp do takich danych podczas swojego szkolenia. Generator otrzymuje losowe dane wejściowe (szum), aby za każdym razem zapewnić losowy punkt początkowy, co zmusza go do uzyskania innego wyniku. Wydaje się, że generator nabiera całej chwały (w końcu generuje produkt danych). Jednak prawdziwą siłą napędową architektury jest dyskryminator. Dyskryminator oblicza błędy, które są propagowane wstecznie do własnej sieci, aby dowiedzieć się, jak najlepiej odróżnić dane rzeczywiste od fałszywych. Błędy propagują się również do generatora, który optymalizuje się tak, że dyskryminator zawiedzie podczas następnej rundy.

PROBLEM Z FAŁSZYWYMI DANYMI

Tak jak GAN może generować imponującą sztukę, tak może generować fałszywych ludzi. Spójrz na https://www.thispersondoesnotexist.com/, aby zobaczyć osobę, która nie istnieje. Jeśli nie wiesz, gdzie szukać, zdjęcia są naprawdę przekonujące. Jednak na razie zdradzają je drobne szczegóły:

• Tła wyglądają na zabłocone lub w jakiś sposób brakuje im tego prawdziwego wrażenia.

• Ci, którzy obejrzeli film Matrix, będą zaznajomieni z epizodycznymi usterkami, które pojawiają się na niektórych obrazach.

• Tekstura pikseli pierwszego planu może nie być właściwa. Na przykład możesz zobaczyć wzory mory tam, gdzie nie są oczekiwane

Jednak rozpoznanie tego rodzaju problemów wymaga człowieka. Ponadto różne problemy w końcu znikną, gdy GAN ulegnie poprawie. GAN może fałszować więcej niż tylko zdjęcia. Możesz stworzyć całkowicie fałszywą ludzką tożsamość w niewiarygodnie krótkim czasie przy niewielkim wysiłku. GAN mogą mieć wszystkie właściwe rekordy we wszystkich właściwych miejscach. Technologia istnieje

dzisiaj do tworzenia fałszywych tożsamości ludzkich, które mogłyby pojawić się w miejscach, w których wykorzenienie ich byłoby wyjątkowo niewygodne. Jest to rodzaj problemu, o którym musisz wiedzieć - nie zabójcze roboty.

GAN-y mogą wydawać się kreatywne. Jednak bardziej poprawnym terminem byłoby to, że są generatywne: uczą się na przykładach, jak różnią się dane i mogą generować nowe próbki, tak jakby zostały pobrane z tych samych danych. GAN uczy się naśladować wcześniej istniejącą dystrybucję danych; nie może stworzyć czegoś nowego. Jak stwierdzono w innych rozdziałach, uczenie głębokie nie jest kreatywne.

Osiąganie bardziej realistycznych wyników

Nawet jeśli koncepcja GAN jest jasna, jego architektura może początkowo wydawać się skomplikowana. Stworzenie podstawowego przykładu GAN stało się całkiem przystępne przy użyciu Keras z TensorFlow, a uczenie się przez działanie to dobry sposób na wyjaśnienie szczegółów technologii, które w przeciwnym razie pozostałyby teoretyczne. Proces składa się z kilku skomplikowanych części, ale ostatecznie wszystko jest dokładnie takie, jak opisano w poprzednich akapitach, używając metafory Iana Goodfellowa. Na kolejnych stronach zbudujesz prosty GAN, który uczy się, jak odtwarzać odręcznie napisane liczby od zera do dziewięciu po nauczeniu się ich ze zbioru danych MNIST. Zbiór danych MNIST to zestaw zdigitalizowanych, znormalizowanych, odręcznych próbek o wymiarach 28 x 28 pikseli (napisanych zarówno przez uczniów szkół średnich, jak i pracowników American Census Bureau), które są często wykorzystywane do szkolenia systemów obrazowania. Przykład zaczyna się od zaimportowania niezbędnych funkcji i klas. Do tego zadania nie potrzebujesz niczego wymyślnego, a już poradziłeś sobie lub nawet przetestowałeś wszystko, co importuje kod:

```
import numpy as np

from keras.datasets import mnist

from keras.models import Sequential, Model

from keras.layers import Input, Dense, Dropout

from keras.layers import BatchNormalization

from keras.layers.advanced_activations import LeakyReLU

from keras.optimizers import Adam

import matplotlib.pyplot as plt

%matplotlib inline
```

Zauważ, że kod pobiera zbiór danych MNIST przy użyciu funkcji Keras mnist. Odrębne macierze obrazów 28 x 28 pikseli i wyrażają wartości pikseli od 0 do 255. Kod przetwarza je, aby były przydatne w sieci głębokiego uczenia, wykonując następujące czynności:

1. Uczyń je wektorem, czyli listą wartości, zmieniając kształt danych.

2. Przekonwertuj ich wartości na typ zmiennoprzecinkowy, używając 32-bitowej precyzji odpowiedniej dla procesorów graficznych, ponieważ wersja 64-bitowa ma zastosowanie tylko do przetwarzania przez procesor.

3. Przeskaluj ich wartości w zakresie 0–1.

Normalizacja to proces przekształcania danych obrazu przed przetwarzaniem głębokiego uczenia. Możesz użyć różnych rodzajów normalizacji, takich jak przeskalowanie zakresu od 0 do 1 do –1 do 1 lub zastosowanie normalizacji statystycznej przez odjęcie średniej i podzielenie przez odchylenie standardowe. Zwykle przeskalowanie wszystkich wartości z zakresu od 0 do 1 jest dobrym rozwiązaniem.

```
def normalize(X):

X = X.reshape(len(X), 784)

X = X.astype('float32')/255

return X

(X_train, Y_train), (X_test, Y_test) = mnist.load_data()

X_train = normalize(X_train)
```

Po przygotowaniu zestawu danych do nauki sieci neuronowej można rozpocząć przygotowywanie architektury GAN. Rozpoczynasz od zdefiniowania kilku parametrów, takich jak typ danych wejściowych dostarczanych do GAN w celu wygenerowania jego obrazów. Dobrym wyborem dla tego projektu jest użycie listy liczb losowych. Wyobraź sobie te losowe liczby jako instrukcje dostarczone do GAN, aby zdecydować, co reprezentować. Masz niewielką kontrolę nad tym, co GAN robi z liczbami, ale w innych modelach możesz efektywnie wykorzystać dane wejściowe i uzyskać pożądane dane wyjściowe. Ustawiasz również optymalizator (w tym przypadku optymalizator Adam) i definiujesz pierwszą część architektury, generator. Generator pobiera losowe dane wejściowe i przepuszcza je przez serię czterech gęstych warstw. Jedynym godnym uwagi aspektem tego procesu jest to, że z wyjątkiem ostatniej warstwy, LeakyReLU zasila wszystkie warstwy, co jest aktywacją, która tłumi negatywne sygnały wejściowe. Normalizacja wsadowa kontroluje dystrybucję wyników poprzez zastosowanie do nich normalizacji statystycznej. Dzięki takiemu podejściu unika się sytuacji, w której podczas treningu wyskakuje ekstremalna liczba. Warto zauważyć, że ostatnia warstwa jest inna; wykorzystuje aktywację sigmoidalną do generowania wyjść od zera do jednego. Ta ostatnia warstwa uwalnia obraz wytwarzany przez GAN, czyniąc z niego generatorową część architektury. Ponieważ generuje 784 wyjścia, których wartości mieszczą się w zakresie od 0 do 1, wyjścia można łatwo zmienić i przeskalować na Tablicę 28 x 28 pikseli o wartościach od 0 do 255 (to jest obraz MNIST).

```
input_dim = 100

np.random.seed(42)

optimizer = Adam(lr=0.0002, beta_1=0.5)

gen = Sequential()

gen.add(Dense(256, input_dim=input_dim))

gen.add(LeakyReLU(alpha=0.2))

gen.add(BatchNormalization())

gen.add(Dense(512))

gen.add(LeakyReLU(alpha=0.2))

gen.add(BatchNormalization())
```

```
gen.add(Dense(1024))

gen.add(LeakyReLU(alpha=0.2))

gen.add(BatchNormalization())

gen.add(Dense(784, activation='sigmoid'))

gen.compile(loss='binary_crossentropy',

optimizer=optimizer)
```

Druga część architektury, dyskryminator, jest konstrukcyjnie podobna do generatora. Znowu ma cztery gęste warstwy, a wszystkie oprócz ostatniej są zasilane przez funkcje aktywacji LeakyReLU. Dyskryminator nie korzysta z normalizacji wsadowej, ale ma przerwanie, aby uniknąć nadmiernego dopasowania, ponieważ ta część wykonuje nadzorowane zadanie klasyfikacji. W rzeczywistości wyjście to pojedynczy węzeł, który wyprowadza wartość prawdopodobieństwa od 0 do 1. Celem tej części sieci neuronowej jest odróżnienie fałszywych obrazów wytwarzanych przez część generatora od rzeczywistych obrazów.

```
dsc = Sequential()

dsc.add(Dense(1024, input_dim=784))

dsc.add(LeakyReLU(alpha=0.2))

dsc.add(Dropout(0.3))

dsc.add(Dense(512))

dsc.add(LeakyReLU(alpha=0.2))

dsc.add(Dropout(0.3))

dsc.add(Dense(256))

dsc.add(LeakyReLU(alpha=0.2))

dsc.add(Dropout(0.3))

dsc.add(Dense(1, activation='sigmoid'))

dsc.compile(loss='binary_crossentropy',

optimizer=optimizer)
```

W tym momencie trudna część została wykonana, połączyłeś pierwszą i drugą połowę sieci i upewniłeś się, że współpracują ze sobą. Korzystanie z funkcjonalnego API Keras, ustawiasz architektury, które są bardziej złożone niż architektury sekwencyjne używane wcześniej w tej sekcji. Podsumowując, część generatora przetwarza dane wejściowe i przekazuje wynik do części dyskryminacyjnej. Dyskryminator działa jak funkcja matematyczna zastosowana do innych funkcji - to znaczy jest dyskryminatorem funkcji (generator funkcji [wejście]). W ten sposób kontrolujesz również optymalizację sieci, ponieważ możesz zamrozić jej część za pomocą funkcji make_trainable def make_trainable(dnn, flag). (W rzeczywistości szkolisz generator i dyskryminator w celu maksymalizacji różnych celów.)

```
dnn.trainable = flag

for l in dnn.layers:
```

```
l.trainable = flag

make_trainable(dsc, False)

inputs = Input(shape=(input_dim, ))

hidden = gen(inputs)

output = dsc(hidden)

gan = Model(inputs, output)

gan.compile(loss='binary_crossentropy',

optimizer=optimizer)
```

Teraz możesz przetestować konfigurację. Przygotowujesz dwie przydatne, przydatne funkcje do generowania szumu wejściowego i wykreślania wyników generatora: def create_noise(n, z):

```
return np.random.normal(0, 1, size=(n, z))

def plot_sample(n, z):

samples = gen.predict(create_noise(n, z))

plt.figure(figsize=(15,3))

for i in range(n):

plt.subplot(1, n, (i+1))

plt.imshow(samples[i].reshape(28, 28),

cmap='gray_r')

plt.axis('off')

plt.show()
```

Właściwy test rozpoczyna się od skonfigurowania kodu na 100 epok treningu i ustawienia partii treningu na 128 obrazów. Kod zaczyna iterować przez liczbę epok i partii niezbędnych do przekazania wszystkich obrazów szkoleniowych do GAN. Podobnie jak w przypadku innych przykładów, uruchomienie tego zajmuje trochę czasu. Jeśli możesz uzyskać dostęp do GPU, lepiej uruchomić go w Google Colab lub na komputerze z kartą GPU. Kiedy możesz uzyskać dostęp do GPU, zaplanuj odczekanie pół godziny na jego uruchomienie w Google Colab. (Twoja własna lokalna konfiguracja GPU może działać lepiej). Przykładowy przykład może znacznie przekroczyć kilka godzin w systemie CPU.

```
epochs = 100

batch_size = 128

batch_no = int(len(X_train) / batch_size)

gen_errors, dsc_errors = (list(), list())

for i in range(0, epochs):

for j in range(batch_no):

# Drawing a random sample of the training set
```

```
rand_sample = np.random.randint(0, len(X_train),
size=batch_size)
image_batch = X_train[rand_sample]
# Creating noisy inputs for the generator
input_noise = create_noise(batch_size, input_dim)
# Generating fake images from the noisy input
generated_images = gen.predict(input_noise)
X = np.concatenate((image_batch,
generated_images))
# Creating somehow noisy labels
y = np.concatenate([[0.9]*batch_size,
[0.0]*batch_size])
# Training discriminator to distinguish fakes from
# real ones
make_trainable(dsc, True)
dsc_loss = dsc.train_on_batch(X, y)
make_trainable(dsc, False)
# Trainining generating fakes
input_noise = create_noise(batch_size, input_dim)
fakes = np.ones(batch_size)
for _ in range(4):
gen_loss = gan.train_on_batch(input_noise,
fakes)
# Recording the losses
gen_errors.append(gen_loss)
dsc_errors.append(dsc_loss)
# Showing intermediate results
if i % 10 == 0:
print("Epoch %i" % i)
plot_sample(10, input_dim)
```

Gdy kod zakończy wykonywanie wielu obliczeń, może zmienić kroki, które podejmuje:

1. Wygeneruj kilka fałszywych obrazów, wywołując samą funkcję generatora. Ponieważ jest to prosta prognoza, bez konieczności uczenia się, obrazy wyjściowe z generatora będą na początku wyglądały zupełnie losowo.

2. Połącz fałszywe obrazy z partią prawdziwych obrazów.

3. Przekaż obrazy dyskryminatorowi, aby określić, czy dyskryminator może oddzielić fałszywe obrazy od prawdziwych. Jest to ćwiczenie szkoleniowe, a dyskryminator uczy się oddzielać najnowocześniejsze obrazy od generatora od prawdziwych obrazów źródłowych.

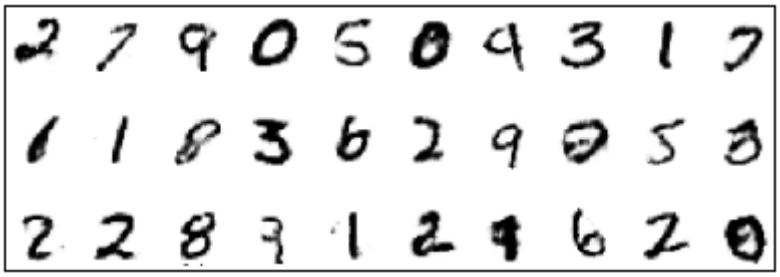
4. Zatrzymaj dyskryminator po zakończeniu uczenia się, aby kod mógł uruchomić go razem z generatorem, ale tym razem tylko generator będzie się uczyć. Na tym etapie kod wprowadza kilka losowych danych wejściowych do generatora, aby przekształcić je w obrazy, a następnie przekazuje fałszywki dyskryminatorowi, aby ustalić, czy dyskryminator można oszukać, aby uwierzył, że są to prawdziwe obrazy.

Gdy dyskryminator może określić, że te obrazy są iloczynem generatora, kod użyje wyniku dyskryminatora jako błędu, z którego generator ma się uczyć (sukces dyskryminatora jest porażką generatora).

Kod zawiera kilka sztuczek, dzięki którym GAN zawsze daje dobre wyniki:

- Kiedy trenujesz dyskryminator, wprowadzasz pewną niepewność do etykiet prawdy, dzięki czemu dyskryminator jest mniej dotkliwy.
- Za każdym razem, gdy trenujesz dyskryminator, czterokrotnie trenujesz również generator. Dzieje się tak, ponieważ nauka generowania obrazów jest w rzeczywistości dłuższym procesem, a zastosowanie tego podejścia przyspiesza ten proces.

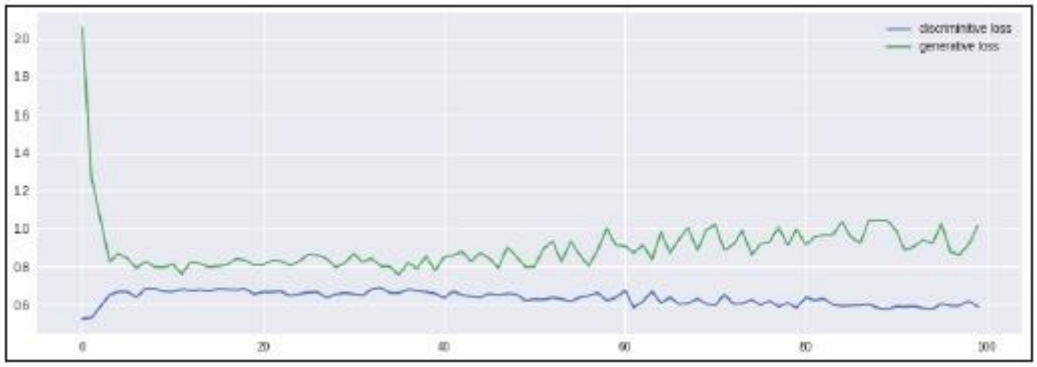
To są dwie najskuteczniejsze sztuczki, ale o jeszcze więcej z nich możesz przeczytać na tej stronie prowadzonej przez Soumith Chintala pod adresem https://github.com/soumith/ganhacks. Jasne jest, że głębokie uczenie się jest nadal bardziej sztuką (możliwą do wyjaśnienia) niż nauką. Wykreślenie niektórych wyników, jak pokazano w poniższym kodzie, pokazuje, że GAN nauczył się generować niemal wiarygodne odręcznie liczby, chociaż nie są one doskonałe. Patrząc na rysunek 16-2, możesz zobaczyć, co GAN może osiągnąć w tak krótkim czasie uczenia się.



Można również zaobserwować błędy, jakie podczas szkolenia wytworzyły dwie sieci tworzące GAN. Użyj następującego kodu (Rysunek poniżej przedstawia dane wyjściowe):

```
# Plotting the errors

plt.figure(figsize=(15, 5))

plt.plot(dsc_errors, label='discriminitive loss')

plt.plot(gen_errors, label='generative loss')
```

```
plt.legend()

plt.show()
```



Rysunek pokazuje, że błędy są w innej skali, ponieważ błąd dyskryminatora jest zawsze niższy niż błąd generatora. Ponadto błąd dyskryminatora ma tendencję do zmniejszania się, ponieważ więcej przykładów pomaga dyskryminatorowi oddzielić fałszywe obrazy od rzeczywistych, nawet jeśli generator poprawi swoje możliwości. Jeśli chodzi o generator, błędy są początkowo silnie zmniejszane, ale potem mają tendencję do ponownego narastania, ponieważ dyskryminator zyskuje doświadczenie w wykrywaniu. Jeśli uruchomisz więcej epok, zobaczysz, że błąd generatora przybiera kształt sinusoidalny, ponieważ regularnie zwiększa swój współczynnik błędów przez pewien czas (w miarę, jak generator staje się bardziej sprawny), a następnie ponownie maleje (po znalezieniu nowej sztuczki, aby oszukać dyskryminator). To niekończąca się walka między dwiema częściami sieci GAN - walka, która zawsze daje bardziej realistyczne obrazy w miarę kontynuowania treningu.

Biorąc pod uwagę rosnące pole

Po rozpoczęciu implementacji typu plain-vanilla, podobnej do tej właśnie zakończonej, badacze rozwinęli ideę GAN do wielu wariantów, które umożliwiają wykonywanie zadań bardziej złożonych niż zwykłe tworzenie nowych obrazów. Lista sieci GAN i ich aplikacji rośnie z dnia na dzień, a nadążanie za nimi jest trudne. Avinash Hindupur zbudował „GAN Zoo", śledząc wszystkie warianty, co z każdym dniem staje się coraz trudniejsze. Zheng Liu zamiast tego preferuje podejście historyczne, a oś czasu GAN, którą utrzymuje. Bez względu na to, jak podchodzisz do GAN, sprawdzanie, jak każdy nowy pomysł wyrasta z poprzednich, jest przydatnym ćwiczeniem.

Wymyślanie realistycznych zdjęć celebrytów

Głównym zastosowaniem GAN jest tworzenie obrazów. Pierwszą siecią GAN, która wyewoluowała z oryginalnego artykułu Goodfellow i innych, jest DCGAN, która została oparta na warstwach konwolucyjnych. Przykład daje wiarygodne proste obrazy, ale opiera się na użyciu gęstych warstw, a nie CNN, które działają lepiej podczas pracy z danymi obrazu. DCGAN znacznie poprawił możliwości generacyjne oryginalnych GAN i wkrótce zrobili wrażenie na wszystkich, tworząc fałszywe obrazy twarzy, biorąc przykłady ze zdjęć celebrytów. Oczywiście nie wszystkie twarze stworzone przez DCGAN były realistyczne, ale wysiłek był tylko początkiem pośpiechu, aby stworzyć bardziej realistyczne obrazy. EBGAN-PT, BEGAN i Progressive GAN to ulepszenia, które osiągają wyższy stopień realizmu. Możesz przeczytać artykuł przygotowany przez firmę NVIDIA na temat progresywnych sieci GAN, aby uzyskać dokładniejsze wyobrażenie o jakości osiąganej przez takie najnowocześniejsze techniki. Kolejnym wielkim ulepszeniem GAN jest warunkowy GAN (CGAN). Chociaż posiadanie sieci wytwarzającej realistyczne obrazy wszelkiego rodzaju jest interesujące, jest mało przydatne, gdy nie możesz w jakiś sposób kontrolować rodzaju otrzymywanych danych wyjściowych.

CGAN manipulują danymi wejściowymi i siecią, aby zasugerować GAN, co powinien on wyprodukować. Teraz, na przykład, masz sieci, które tworzą obrazy twarzy osób, które nie istnieją, w oparciu o twoje preferencje dotyczące wyglądu włosów, oczu i innych szczegółów.

Poprawa szczegółów i tłumaczenie obrazu

Tworzenie obrazów o wyższej jakości i ewentualnie kontrolowanie generowanych danych wyjściowych otworzyło drogę do większej liczby zastosowań. W tym rozdziale nie ma miejsca na omówienie ich wszystkich, ale poniższa lista zawiera przegląd tego, co można znaleźć:

- Cykl GAN: Stosowany do stylu transferu neuronowego (jak omówiono w rozdziale 10). Na przykład możesz zamienić konia w zebrę lub obraz Moneta w taki, który wydaje się pochodzić od van Gougha. Eksplorując projekt na https://github.com/junyanz/CycleGAN, możesz zobaczyć, jak to działa i zastanowić się, jakie przekształcenia może zastosować do obrazów.
- Super Resolution GAN (SRGAN): Przekształca obrazy, zmieniając rozmazane obrazy o niskiej rozdzielczości w wyraźne obrazy o wysokiej rozdzielczości. Zastosowanie tej techniki w fotografii i kinie jest interesujące, ponieważ poprawia niską jakość obrazu prawie bez żadnych kosztów..
- Generowanie obrazu osoby sterowanej pozy: kontroluje pozę osoby przedstawionej na tworzonym obrazie.
- Pix2Pix: Tłumaczy szkice i mapy na rzeczywiste obrazy i odwrotnie. Możesz użyć tej aplikacji do przekształcenia szkiców architektonicznych w obraz rzeczywistego budynku lub do przekształcenia zdjęcia satelitarnego w narysowaną mapę.
- Naprawa obrazu: naprawia lub modyfikuje istniejący obraz, określając, czego brakuje, co zostało usunięte lub zasłonięte.
- Starzenie się twarzy: Określa, jak twarz będzie się starzeć.
- Midi Net: Tworzy muzykę w Twoim ulubionym stylu.

Zabawa z uczeniem głębokiego wzmacniania

Oprócz przykładu GAN, możesz pokusić się o utożsamienie uczenia głębokiego z przewidywaniami uczenia nadzorowanego. Jednak uczenie głębokie jest również używane do uczenia się nienadzorowanego i uczenia się przez wzmacnianie (RL). Nauka nienadzorowana obsługuje wiele uznanych technik, takich jak autokodery i mapy samoorganizujące się (SOM), których nie obejmuje ta książka. Nienadzorowane techniki mogą pomóc w segmentacji danych na jednorodne grupy lub w celu wykrycia anomalii w zmiennych. Techniki RL są nawet bardziej popularne niż techniki uczenia się bez nadzoru wśród praktyków. Ostatnio jako obiekt intensywnych badań, RL osiąga mądrzejsze rozwiązania problemów, takich jak parkowanie samochodu, nauka jazdy w zaledwie dwadzieścia minut , sterowanie robotem przemysłowym i nie tylko. Ta Część opisuje niektóre z tych technik, w tym jedną o nazwie AlphaGo, która została opisana na wiadomość po tym, jak został pierwszym algorytmem, który pokonał profesjonalnego gracza w Go (starożytna chińska gra planszowa) w grze parzystej. Zdobędziesz również praktyczne doświadczenie, pracując z kilkoma przykładami, które wprowadzają Cię w OpenAI Gym , kompletny zestaw narzędzi do eksperymentowania z głębokim uczeniem, oraz keras-rl, gotowa do użycia implementacja najnowocześniejszych algorytmów RL, takich jak Google Deep Q-Network (DQN). DQN to algorytm używany do grania w klasyczne gry na Atari 2600 na poziomie eksperta i wygrywania. DQN to tylko jedno z możliwych zastosowań tej techniki, opatentowanej przez Google DeepMind). Po pokazaniu, jak zbudować działającą przykładową sieć głębokiego uczenia, zdolną do pomyślnego grania w prostą grę omówiono, jak działa AlphaGo i dlaczego jego zwycięstwo jest kamieniem milowym dla głębokiego uczenia się i ogólnie sztucznej inteligencji.

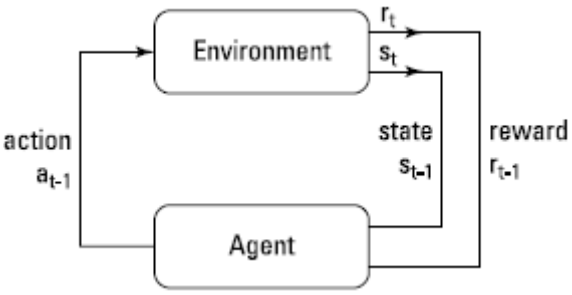
Granie w gry z sieciami neuronowymi

Jako małe dziecko być może lubiłeś odkrywać otaczający cię świat i podejmować ryzyko, aby sprawdzić swoje umiejętności pod czujnym okiem rodziców. Dopiero później zastąpiłeś wiedzę zbudowaną na bezpośrednim doświadczeniu wiedzą otrzymaną od innych. Tak jak nadzorowany algorytm uczenia maszynowego przypomina studenta uczącego się o świecie na podstawie czyichś przeszłych doświadczeń opisanych w książkach (w tej metaforze doświadczenia są danymi), algorytm RL jest bardziej jak maluch – czysta tablica, która gromadzi wiedzę, próbując coś i sprawdzanie, czy ta wiedza zapewnia nagrodę lub karę. RL zapewnia kompaktowy sposób uczenia się bez gromadzenia dużej masy danych, ale obejmuje również złożoną interakcję ze światem zewnętrznym. Ponieważ RL zaczyna się bez żadnych danych, interakcja ze światem zewnętrznym i otrzymywanie informacji zwrotnej określa metodę używaną do uzyskania wymaganych danych. Możesz zastosować to podejście do robota poruszającego się w świecie fizycznym lub do bota wędrującego w świecie cyfrowym. W szczególności RL wydaje się kuszące w przypadku problemów, które nie są łatwe do złamania przy użyciu samych danych statycznych (dostarczonych). Przykładami takich problemów jest nauczenie komputera samodzielnego grania w grę lub wypracowanie najlepszego możliwego wyniku w niepewnych sytuacjach, takich jak optymalizacja reklam online. Reklama jest jednym z najlepszych przykładów, ponieważ aplikacja musi dostarczać odpowiednie kampanie odpowiednim odbiorcom, ale brakuje wcześniejszego doświadczenia (dla danych statycznych lub istniejących), ponieważ wszystkie kampanie są nowe.

Przedstawiamy uczenie się przez wzmacnianie

W RL masz agenta (którym może być robot w świecie rzeczywistym lub bot w świecie cyfrowym) wchodzący w interakcję ze środowiskiem, które może obejmować wirtualny lub inny świat z własnymi regułami. Agent może odbierać informacje z otoczenia (zwanego stanem) i może na nim działać, czasami je zmieniając. Co ważniejsze, agent może otrzymać informację z otoczenia, pozytywną lub

negatywną, na podstawie sekwencji działań lub zaniechań. Dane wejściowe są nagrodą, nawet jeśli są negatywne. Celem RL jest nauczenie agenta, jak się zachowywać, aby zmaksymalizować łączną sumę nagród otrzymanych podczas jego doświadczenia w środowisku. Relację między agentem a środowiskiem można określić na rysunku . Zwróć uwagę na indeksy czasu. Jeśli potraktujesz obecną chwilę w czasie jako t, poprzednią chwilą jest t–1. W czasie t–1 agent działa, a następnie otrzymuje od otoczenia zarówno stan, jak i nagrodę. Na podstawie zbiorów wartości odnoszących się do działania w czasie t, stanu w czasie t-1 i nagrody w czasie t, algorytm RL może nauczyć się działania, aby uzyskać określony stan środowiska.



Ian Goodfellow, naukowiec zajmujący się badaniami nad sztuczną inteligencją odpowiedzialny za tworzenie GAN, uważa, że lepsza integracja między RL a głębokim uczeniem jest jednym z głównych priorytetów dalszych postępów w uczeniu głębokim. Lepsza integracja prowadzi do inteligentniejszych robotów. Integracja jest obecnie gorącym tematem, ale do niedawna RL zazwyczaj silniej łączyła się ze statystykami i algorytmami niż sieci neuronowe. Niektórzy ludzie wcześniej próbowali zmusić ich do działania razem. We wczesnych latach 90. Gerald Tesauro z IBM Research Center wymyślił sposób, w jaki komputer może nauczyć się grać w Backgammona i pokonać przeciwnika. mistrz świata (człowieka). Z powodzeniem wykorzystał sieć neuronową do zasilania algorytmu RL, tworząc program komputerowy, który nazwał TD-Gammon. TD-Gammon wzbudził szerokie zainteresowanie zastosowaniem sieci neuronowych do problemów RL, więc wielu ludzi próbowało po Tesauro pokazać inne możliwe zastosowanie tej techniki, ale wszystkim się nie udało i pomysł umarł. Później niektórzy badacze zauważyli, że Backgammon to gra oparta częściowo na przypadku. Inne gry (takie jak szachy lub Go) i problemy w świecie rzeczywistym, które nie reagowały dobrze na połączenie głębokiego uczenia się i RL, nie są zależne od szczęścia. Brak elementu szczęścia tylko częściowo wyjaśnia problem z uzyskaniem dobrego uczenia głębokiego w niektórych grach (na przykład poker jest grą losową, ale od jakiegoś czasu jest poza zasięgiem RL i głębokiego uczenia). Pomimo tego spostrzeżenia (tj. głębokie uczenie działa lepiej w przypadku niepewności), naukowcy wciąż nie mogli znaleźć rozwiązania, które umożliwiłoby sieciom neuronowym obsługę RL w przypadku nowych problemów, aż do kilku lat później, kiedy zespół badawczy Google ds. głębokiego uczenia udowodnił, że przeciwnie. W Google DeepMind wykorzystali dobrze znaną technikę RL o nazwie Q-learning i sprawili, że działa ona z głębokim uczeniem, a nie z klasycznym algorytmem obliczeniowym. Nowy wariant, nazwany Deep Q-Learning, wykorzystuje zarówno sploty, jak i regularne gęste warstwy, aby uzyskać dane wejściowe problemu i je przetworzyć. To rozwiązanie nie tylko ponownie połączyło głębokie uczenie i RL, ale także zaowocowało nadludzkimi możliwościami grania w niektóre gry na Atari 2600. Algorytm nauczył się grać w stosunkowo krótkim czasie i znalazł sprytne strategie, z których korzystają tylko najbardziej doświadczeni gracze. Zespół DeepMind opublikował również artykuł zatytułowany „Kontrola na poziomie człowieka poprzez głębokie uczenie wzmacniające”. Mimo wysoce technicznego tematu artykuł jest dość czytelny. Pokazuje, dlaczego Deep Q-Learning działa z niektórymi grami, a źle z innymi. Problem pojawia się, gdy sieć neuronowa musi opracować złożone i długoterminowe strategie.

Symulowanie środowisk gry

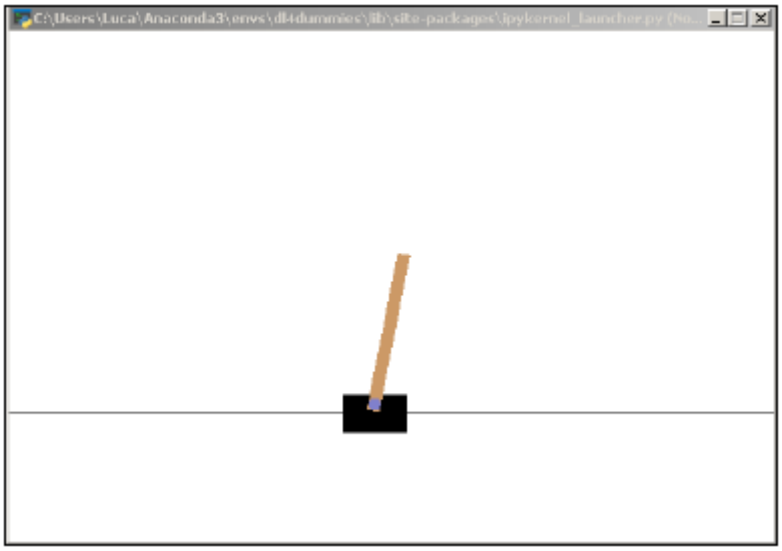
Nawet jeśli nie pracujesz z prekonstytuowanymi zestawami danych podczas pracy z RL (co oznacza, że nie musisz zbierać i oznaczać danych), musisz wziąć pod uwagę interakcje między algorytmem a światem zewnętrznym, co jest innym wyzwaniem. Na przykład, jeśli chcesz zbudować algorytm RL, który może cię pokonać w szachach, musisz najpierw zbudować szachową grę komputerową, która zawiera wszystkie zasady gry. Algorytm połączy się z tym zestawem reguł jako część danych wejściowych. Aby umożliwić większej liczbie badaczy i praktyków osiągnięcie tego warunku wstępnego, OpenAI ,organizacja non-profit zajmująca się badaniami nad sztuczną inteligencją, opracowała pakiet Gym o otwartym kodzie źródłowym. Siłownia to kompletny zestaw narzędzi, który pomoże każdemu w rozwoju Algorytmy RL stosowane zarówno do podstawowych, jak i trudnych problemów, oferując gotowe do użycia środowiska. OpenAI Gym pozwala zweryfikować, czy twoje algorytmy mają ogólny zakres, ponieważ wszystkie środowiska używają tego samego interfejsu poleceń. Po prostu zmieniasz nazwę środowiska, aby przetestować rozwiązanie RL w innej sytuacji. Pakiet zawiera również stronę internetową, na której możesz opublikować swoje wyniki, porównując, jak Twój algorytm RL radzi sobie z innymi rozwiązaniami. Możesz łatwo zainstalować pakiet siłowni i jego wymagania wstępne na swoim komputerze lokalnym (pakiet h5py) z powłoki Anacondy używając tych poleceń (pip połączy się z Internetem, aby pobrać pakiety i zainstalować je lokalnie):

pip zainstaluj h5py

pip zainstaluj siłownię

conda install -c menpo ffmpeg

W przeciwieństwie do innych przykładów książek, przykłady nie mogą działać w Google Colab z przyczyn technicznych - procedury są zbyt skomplikowane. Musisz uruchomić kod na swoim komputerze lokalnym. Korzystając z Gym nie musisz się już martwić o środowisko. Dostępne są różne środowiska, niektóre prezentujące zadania algorytmiczne (takie jak nauka kopiowania sekwencji), niektóre oparte na tekście, inne związane z robotami (np. sterowanie ramieniem robota) i większa liczba oparta na starych grach arkadowych Atari, takich jak Space Invaders lub Breakout. . Zaczynasz od klasycznego środowiska, jak opisano w literaturze naukowej RL, ale możesz także zbadać inne możliwości oferowane przez pakiet. Uczenie się, jak rozwiązywać gry za pomocą RL, pomaga również opracowywać lepsze rozwiązania problemów w świecie rzeczywistym. W Uber, firmie zajmującej się siecią transportową, inżynierowie studiują algorytmy RL, zastanawiają się, jak działa RL, i analizują sposób, w jaki RL podejmuje decyzje mające na celu rozwijanie zaufania i zaufania do sztucznej inteligencji, co można przeczytać na blogu inżynierskim Ubera .Siłownia jest zbudowana wokół podstawowych zasad RL, więc znajdziesz funkcje i metody do opisania agenta i środowiska. Możesz także poprosić agenta o wykonanie akcji lub zaniechanie działania w środowisku. Środowisko odpowie, przekazując informację zwrotną w dwóch formach: nowy stan, którego możesz użyć do podsumowania nowej sytuacji w środowisku; oraz nagrodę, która jest wynikiem pokazującym sukces lub porażkę. Jedyną częścią, którą musisz zakodować, jest RL, a podstawowy przykład możesz zacząć od kilku linijek Pythona. Środowisko dla eksperymentu RL to problem CartPole. Kij swobodnie przyczepia się do wózka poruszającego się po torze (nie uwzględnia się tarcia). Wahadło zaczyna się pionowo, w niestabilnej równowadze, a celem otoczenia jest zapobieganie jego przewróceniu (co wymaga kąta większego niż 15 stopni od pionu). W przypadku akcji określasz, czy zwiększyć lub zmniejszyć prędkość wózka w jednym lub drugim kierunku. Rysunek przedstawia reprezentację środowiska zapewnianego przez pakiet OpenAI Gym. Możesz również zobaczyć przykład, jak zrównoważyć CartPole w tym rzeczywistym eksperymencie przez Wydział Inżynierii Technologicznego Instytutu Edukacji Kretetu



Środowisko CartPole działa poprzez zgłaszanie obserwacji następujących stanów:

- Pozycja koszyka
- Prędkość wózka
- Kąt biegunowy
- Prędkość bieguna na czubku

Możesz manipulować środowiskiem w oparciu o te stany poprzez:

- Pchanie wózka w lewo
- Pchanie wózka w prawo

Poniższy kod tworzy środowisko i testuje z nim kilka losowych poleceń:

```
import numpy as np

import gym

env = gym.make('CartPole-v0')

np.random.seed(42), env.seed(42)

nb_actions = env.action_space.n

input_shape = (1, env.observation_space.shape[0])
```

Tworzysz środowisko za pomocą pojedynczego polecenia make, które zwraca klasę Pythona używaną do uzyskiwania ogólnych informacji o środowisku (na przykład o akcjach, które możesz wykonać za pomocą env.action_space), kontrolowaniu upływu czasu lub wykonaniu określonej akcji wewnątrz środowisko. Kilka następnych linijek resetuje właśnie utworzone środowisko. Wszystko jest uruchamiane od nowa w początkowej pozycji (niektóre aspekty środowiskowe są ustalane losowo). Kod wykorzystuje pętlę 200 iteracji do wykonywania różnych losowych akcji, próbkowanych z zakresu możliwych dostępnych akcji (siła przyłożona do wózka w zakresie od -1 do +1). Po zakończeniu iteracji gra kończy się niepowodzeniem (gdy drążek znajduje się więcej niż 15 stopni od pionu) lub wózek przesunie się o więcej niż 2,4 jednostki od środka, zmienna done staje się prawdziwa i eksperyment kończy się (liczba kroków różnią się, ponieważ jest to losowy proces wyborów).

```
observation = env.reset()

for t in range(200):

env.render()

act = env.action_space.sample()

obs, rwrd, done, info = env.step(act)

if done:

print("Episode concluded after %i timesteps" % (t+1))

break

env.close()
```

Prezentacja Q-learningu

Budowanie rozwiązania RL opartego na głębokim uczeniu się wymaga sporego wysiłku w kodowaniu, ale można wykorzystać istniejący pakiet, keras-rl, który zawiera najnowszy stan- najnowocześniejsze algorytmy RL. Ten pakiet, opracowany przez Matthiasa Plapperta, naukowca pracującego w OpenAI, może bezproblemowo integrować się z sieciami neuronowymi zbudowanymi z Keras i środowiskami OpenAI. Instalujesz pakiet, wydając w powłoce to polecenie:

```
pip install keras-rl
```

Po zainstalowaniu keras-rl importujesz niezbędne funkcje z Keras (używasz sieci neuronowej dla swojego rozwiązania RL) oraz wyspecjalizowane funkcje keras-rl do tworzenia agenta RL.

```
from keras.models import Sequential

from keras.layers import Dense, Activation

from keras.layers import Flatten, Dropout

from keras.optimizers import Adam

from rl.agents.dqn import DQNAgent

from rl.policy import EpsGreedyQPolicy

from rl.memory import SequentialMemory
```

Pierwszym krokiem jest zbudowanie sieci zdolnej do określenia wyniku w postaci nagrody z określonego stanu środowiska. Jest to podejście do uczenia się oparte na wartościach i jest to idea stojąca za Deep Q-Network i Deep Q-Learning: aby w przybliżeniu określić nagrodę po wykonaniu określonego działania, biorąc pod uwagę obecny stan. Ta technika nie uwzględnia bezpośrednio przeszłych działań i związanego z nimi stanu lub pełnej sekwencji działań, które agent powinien wykonać, ale działa skutecznie w przypadku wielu problemów, wskazując najlepsze pojedyncze działanie do podjęcia spośród alternatyw.

```
model = Sequential()

model.add(Flatten(input_shape=input_shape))

model.add(Dense(12))
```

```
model.add(Activation('relu'))

model.add(Dense(nb_actions))

model.add(Activation('linear'))

print(model.summary())
```

Sieć neuronowa, którą tworzy kod, jest prosta, składa się z trzech warstw o malejącej liczbie neuronów. Wszystkie warstwy są aktywowane przez funkcję ReLU, ale ostatnia warstwa aktywuje się liniowo, aby uzyskać wartość wyjściową, która jest używana jako akcja, którą wykona bot. Algorytm DQN nie rozumie, jak działa środowisko. W ludzkim sensie algorytm po prostu łączy stan i działania z oczekiwanymi nagrodami, co odbywa się za pomocą funkcji matematycznej. Dlatego algorytm nie może zrozumieć, czy gra w konkretną grę; jego rozumienie otoczenia ogranicza się do wiedzy o zgłaszanym stanie wynikającej z podejmowanych działań. Ta sieć neuronowa zasila algorytm DQN wraz z polityką (polityka to funkcja, która wybiera sekwencję działań) oraz pamięcią poprzednich działań i stanów. Pamięć jest niezbędna, aby przykład mógł wytrenować sieć neuronową. Rejestruje poprzednie doświadczenia agentów ze środowiskiem, a kod może próbkować je w celu wyodrębnienia serii działań o danym stanie. Sieć neuronowa wykorzystuje pamięć, aby nauczyć się oszacować prawdopodobną nagrodę z działania podjętego w stanie.

W przypadku zasad polityka EPS Greedy Q wykonuje jedną z następujących czynności:

- Wykonuje losową akcję z prawdopodobieństwem epsilon
- Wykonuje aktualnie najlepszą akcję z prawdopodobieństwem (1 – epsilon)

Te dwie polityki pokazują kompromis między poszukiwaniem a eksploatacją. Kiedy funkcja zasad Eps Greedy Q wybiera losowo wykonanie losowej akcji, algorytm bada, ponieważ może zdecydować się na nieoczekiwaną akcję, która może prowadzić do interesującego wyniku. Na przykład w grze Atari Breakthrough kopanie dziury w ścianie i doprowadzanie piłki do szaleństwa, niszczenie ściany od góry, jest wyraźnie strategią, która pojawiła się losowo podczas eksploracji i którą algorytm RL zapisał i nauczył się jako niezwykle przydatny.

```
policy = EpsGreedyQPolicy(eps=0.3)

memory = SequentialMemory(limit=50000,

window_length=1)

dqn = DQNAgent(model=model,

nb_actions=nb_actions,

memory=memory,

nb_steps_warmup=50,

target_model_update=0.01,

policy=policy)

dqn.compile(Adam(lr=0.001))

training = dqn.fit(env, nb_steps=30000,

visualize=False, verbose=1)
```

System sam się szkoli, stosując to samo podejście, które jest stosowane w innych sieciach uczenia głębokiego. Po ukończeniu nauki na 30 000 przykładów jest gotowy do testowania:

```
env = gym.make('CartPole-v0')

mon = gym.wrappers.Monitor(env,

"./gym-results",

force=True)

mon.reset()

dqn.test(mon, nb_episodes=1, visualize=True)

mon.close()

env.close()
```

Test powinien zakończyć się wysoką nagrodą (oczekiwany wynik to około 200, ale może być inny, ponieważ test ma losowy element treningowy). Możesz przejrzeć zachowanie koszyka za pomocą dyrektyw DQN znajdujących się w filmie nagranym podczas testu:

```
import io

import base64

from IPython.display import HTML

template =

'./gym-results/openaigym.video.%s.video000001.mp4'

video = io.open(template % mon.file_infix, 'r+b').read()

encoded = base64.b64encode(video)

HTML(data='''

<video width="520" height="auto" alt="test" controls>

<source src="data:video/mp4;base64,{0}"

type="video/mp4"/>

</video>'''.format(encoded.decode('ascii')))
```

System sam się szkoli, stosując to samo podejście, które jest stosowane w innych sieciach uczenia głębokiego. Po ukończeniu nauki na 30 000 przykładów jest gotowy do testowania:

Test powinien zakończyć się wysoką nagrodą (oczekiwany wynik to około 200, ale może być inny, ponieważ test ma losowy element treningowy). Możesz przejrzeć zachowanie koszyka za pomocą dyrektyw DQN znajdujących się w filmie nagranym podczas testu:

Wyjaśnienie Alpha-Go

Szachy i Go to popularne gry planszowe, które mają wspólne cechy, takie jak granie w nią przez dwóch graczy, którzy poruszają się na zmianę i nie mają elementu losowego (nie rzuca się kostkami, jak w tryktraku). Poza tym mają różne zasady gry i złożoność. W szachach każdy gracz ma do dyspozycji 16

pionów w zależności od typu, a gra kończy się, gdy król jest patowy (zaszachowany) - nie może się ruszyć dalej. Eksperci szacują, że możliwych jest około 10123 różnych partii szachów, co jest dużą liczbą, biorąc pod uwagę, że naukowcy szacują liczbę atomów w znanym wszechświecie na około 1080. Jednak komputery mogą opanować pojedynczą partię szachów, określając przyszłe możliwe ruchy na tyle daleko, by mieć przewagę nad każdym ludzkim przeciwnikiem. W 1997 roku Deep Blue, superkomputer IBM zaprojektowany do gry w szachy, pokonał Garry'ego Kasparowa, mistrza świata w szachach. Komputer nie może przewidzieć pełnej partii szachów używając brutalnej siły (obliczając każdy możliwy ruch od początku do końca partii). Wykorzystuje pewne heurystyki i jego zdolność do przyjrzenia się pewnej liczbie przyszłych ruchów. Deep Blue był komputerem o wysokiej wydajności obliczeniowej, który mógł przewidzieć więcej przyszłych ruchów w grze niż jakikolwiek poprzedni komputer. W Go masz siatkę linii 19 x 19 zawierającą 361 miejsc, na których każdy gracz umieszcza kamień (zwykle w kolorze czarnym lub białym) za każdym razem, gdy gracz wykonuje swoją turę. Celem gry jest zamknięcie w kamieniach większej części planszy niż przeciwnika. Biorąc pod uwagę, że każdy gracz ma średnio około 250 możliwych ruchów w każdej turze, a gra składa się z około 150 ruchów, komputer potrzebowałby wystarczającej ilości pamięci, aby pomieścić 150250 gier, czyli około 10360 plansz. Z perspektywy zasobów Go jest bardziej złożone niż szachy, a eksperci wierzyli, że żadne oprogramowanie komputerowe nie będzie w stanie pokonać ludzkiego mistrza Go w ciągu następnej dekady przy użyciu tego samego podejścia, co Deep Blue. Jednak AlphaGo dokonał tego za pomocą technik RL. DeepMind, londyńskie centrum badawcze należące do Google, opracowało w 2016 r. system komputerowy o nazwie AlphaGo, który zawierał umiejętności gry w Go, których nigdy wcześniej nie osiągnęło żadne rozwiązanie sprzętowe i programowe. Po skonfigurowaniu systemu Deep-Mind przetestował AlphaGo z najsilniejszym mistrzem Go żyjącym w Europie, Fan Gui, który trzykrotnie był mistrzem Europy w Go. DeepMind rzucił mu wyzwanie w meczu za zamkniętymi drzwiami, a AlphaGo wygrał wszystkie mecze, pozostawiając Fan Gui zdumiony stylem gry wyświetlanym przez komputer. Następnie, po tym, jak Fan Gui pomógł udoskonalić umiejętności AlphaGo, zespół DeepMind, kierowany przez ich dyrektora generalnego Demisa Hassabisa i głównego naukowca Davida Silvera, rzucił wyzwanie Lee Sedolowi, południowokoreańskiemu profesjonalnemu graczowi Go, który ma dziewiąty dan, najwyższy poziom, jaki może osiągnąć mistrz. . AlphaGo wygrał serię czterech meczów z Lee Sedolem i przegrał tylko jeden. Oprócz meczu, który przegrał z powodu nieoczekiwanego ruchu mistrza, faktycznie prowadził inne gry i zadziwił mistrza, grając nieoczekiwane, uderzające ruchy. W rzeczywistości obaj gracze, Fan Gui i Lee Sedol, uważali, że granie przeciwko AlphaGo jest jak granie przeciwko zawodnikowi pochodzącemu z innej rzeczywistości: ruchy AlphaGo nie przypominały niczego, co widzieli wcześniej.

Ustalanie, czy masz zamiar wygrać

W szachach możesz odkrywać przyszłe ruchy i zajść daleko z odpowiednim komputerem. Liczba pionów, ich ograniczone ruchy i stan planszy ułatwiają ustalenie, co może się wydarzyć. Co więcej, możesz uzyskać miarę postępów gry lub oceny ruchu ze względu na charakter samej gry (na przykład figury szachowe mają wartość). W Go nie możesz dokonać tych ustaleń, ponieważ liczba możliwych ruchów eksploduje zaledwie kilka ruchów do przodu. Ponadto nie możesz określić wartości ruchu, ponieważ musisz zobaczyć ukończoną grę, zanim zrozumiesz, w jaki sposób każdy ruch przyczynił się do końca gry. Ponieważ podstawowa strategia Go różni się od szachów, programy komputerowe grające w Go używają innego podejścia, aby określić, jakie ruchy wykonać. Takie podejście nazywa się Monte Carlo Tree Search (MCTS). W MCTS komputer symuluje wiele kompletnych gier z istniejącego stanu planszy, najpierw używając losowych ruchów, a następnie wykorzystując najbardziej udane ruchy, które znajdzie podczas losowej gry. Nie różni się to zbytnio od podejścia do eksploracji/eksploatacji w RL. Korzystając z tego podejścia, komputer może określić, czy ruch w Go jest dobry, czy nie, symulując wystarczającą liczbę gier, aby uzyskać wiarygodną odpowiedź. AlphaGo

używa MCTS, ale obsługuje przetwarzanie algorytmu za pomocą sieci neuronowych. System składa się z dwóch elementów:

- Spojrzenie na system przyszłych ruchów: Metoda prognozowania podobna do tej stosowanej przez Deep Blue. Jest to system wyszukiwania drzewa, ponieważ rozgałęzia się przez możliwe gry i polega na MCTS, aby to zrobić.
- Niektóre sieci CNN: zapewniają wskazówki dotyczące systemu wyszukiwania drzew.

Sieci głębokiego uczenia się są dwojakiego rodzaju: sieci polityk i sieci wartości. Obie sieci przetwarzają obraz tablicy, szukając lokalnych i ogólnych wzorców, takich jak te używane w przetwarzaniu obrazu używanym do rozróżniania psa od kota. Rola dwóch sieci politycznych (jednej wolniejszej, ale bardziej precyzyjnej, drugiej szybszej, ale bardziej brutalnej) ma kierować wyborem działań. Te sieci strategii generują prawdopodobieństwo dla każdego możliwego ruchu, więc MCTS może symulować realistyczne gry na podstawie ich sugestii, a nie losowo. Sieć wartości zapewnia prawdopodobieństwo wygranej, biorąc pod uwagę stan zarządu. Wykorzystując zarówno sieci wartości, które zapewniają intuicję sytuacji w grze, jak i sieć zasad, która pomaga komputerowi przewidywać przyszłe ruchy, AlphaGo może dostarczyć najlepszą strategię i ruchy podczas gry. Biorąc pod uwagę, że taka architektura nie jest tak naprawdę kompleksowa, ponieważ obejmuje tak wiele różnych systemów, inżynierowie z Deep Mind najpierw przeszkolili AlphaGo, używając gier, w które grają mocni amatorzy, aby uruchomić sieci neuronowe. (Wykorzystali 160 000 amatorskich gier zebranych od internetowej społeczności Go.) Wreszcie, pozwolili AlphaGo grać przeciwko sobie, aby nauczyć się, jak poprawić i udoskonalić swoje umiejętności gry. Tutaj techniki RL odegrały kluczową rolę: nauczyli komputery grać w tryktraka, szachy, pokera, scrabble i wreszcie Go, dzięki AlphaGo rzucając sobie wyzwania miliony razy, pracując w szybkim i intensywnym środowisku budującym doświadczenie, które potrafią ludzie". uchwyt. David Silver, główny badacz projektu AlphaGo, stwierdził, że samouczenie się jest tak skuteczne w budowaniu inteligentnych systemów, ponieważ przeciwnik, z którym mają do czynienia te systemy, ma zawsze odpowiedni poziom umiejętności – nigdy za niski ani za wysoki. Pozwalanie systemowi uczyć się przez samo granie jest czymś, co można było zobaczyć w TD-Gammon w 1992 roku, a także w komputerze WOPR w filmie WarGames z 1983 roku. (W tym sensie komputer WOPR jest tak samo symboliczny dla sztucznej inteligencji, jak HAL9000 w 2001: Odyseja kosmiczna).

Stosowanie samokształcenia na dużą skalę

Zespół DeepMind, który stworzył AlphaGo, nie zatrzymał się po sukcesie swojego rozwiązania; wycofał AlphaGo i stworzył jeszcze bardziej niesamowite systemy. Najpierw zespół zbudował AlphaGo Zero, które jest trenowane przez AlphaGo, grając przeciwko sobie. Następnie stworzył Alpha Zero, czyli ogólny program, który sam może nauczyć się grać w szachy i shogi, japońską grę w szachy. Jeśli AlphaGo zademonstrowało, jak rozwiązać problem uważany za niemożliwy dla komputerów, AlphaGo Zero pokazał, że komputery mogą osiągnąć superumiejętności dzięki samouczeniu się (co jest w istocie RL). Ostatecznie jego wyniki były nawet lepsze niż te, które zaczynały się od ludzkiego doświadczenia: AlphaGo Zero rzucił wyzwanie AlphaGo i wygrał 100 meczów, nie przegrywając ani jednego.

Artykuł opublikowany w Nature wyjaśnia, że AlphaGo Zero rozpoczął naukę od wykonywania losowych ruchów. To ćwiczenie jest podobne do tego, jak algorytm wzmacniający DQN nauczył się równoważyć wózek w przykładzie kodowania. W około 29 milionach samodzielnych gier AlphaGo Zero osiągnął poziom przewyższający poprzedni system AlphaGo. Co więcej, AlphaGo Zero jest mniej złożona pod względem modeli głębokiego uczenia i wymaganego sprzętu. Potrzebuje jednego komputera i czterech niestandardowych chipów TPU Google, podczas gdy oryginalny AlphaGo wymagał kilku maszyn i 48 TPU. AlphaGo, AlphaGo Zero i Alpha Zero reprezentują nową granicę RL, a także nadzieję na przyszłe zastosowania. W rzeczywistości, poza grą w Go, szachy i shogi, te systemy nie są zdolne do niczego

innego. Podobnie jak Deep Blue, systemy te koncentrują się na jednym zadaniu, które mogą wykonać na jakościowo nadludzkim poziomie. Naukowcy z DeepMind przewidują dalsze możliwe zastosowania, które są obecnie trudne i wymagające dla ludzi, takie jak zwijanie białek, optymalizacja zużycia energii w sieci lub odkrywanie nowych materiałów w chemii.

UJĘCIE ZNACZENIA ALFA ZERO

Wyczyn Alpha Zero jest nawet ważniejszy niż to, co osiągnął AlphaGo. Ta książka często wspomina o roli danych w otwieraniu drogi do dobrego funkcjonowania uczenia głębokiego. Więcej danych przy prostym modelu może pokonać sprytny algorytm wykorzystujący mniej danych. Jednak Alpha Zero udało się osiągnąć szczyt wydajności, zaczynając od zerowych danych. Ta zdolność wykracza poza ideę, że dane mogą osiągnąć każdy cel. Alpha Zero jest możliwe, ponieważ znamy procesy generatywne stosowane przez graczy w Go, a badaczom DeepMind udało się odtworzyć idealne środowisko Go. Jeśli chodzi o prawa, można zdefiniować znacznie więcej sytuacji niż Go. Na przykład naukowcy znają podstawowe prawa działania świata fizycznego, ponieważ ludzie spędzili wieki na badaniu ich, a najjaśniejsze umysły starały się je zrozumieć – od Isaaca Newtona po Alberta Einsteina i Stephena Hawkinga. Ta wiedza otwiera drzwi do tworzenia modeli generatywnych, które mogą replikować i symulować procesy myślowe wykorzystywane do tworzenia danych potrzebnych do uczenia się przez głębokie uczenie i modele sztucznej inteligencji. Jeśli ten proces wydaje się naprawdę zaawansowany, zwróć uwagę: już tu jest. Ludzie już dyskutują, w jaki sposób gra wideo może pomóc w budowaniu lepszych samojezdnych samochodów.