XI. Wykształcone odgadywanie ze Statystyką i Prawdopodobieństwem

Quake III to jedna z moich ulubionych gier. Dzięki mojemu doświadczeniu Quake / Quake II mogłem zacząć od drugiego najwyższego poziomu botów po raz pierwszy grałem w Quake III; ostatecznie stałem się na tyle pewny siebie, aby poradzić sobie z koszmarnymi robotami. Początkowo te boty zabrały mnie dość szybko (jak tylko jeden z nich chwycił ten karabin, miałem poważne kłopoty). Co ciekawe, patrząc na punkt widzenia bota w grze Quake III, możesz określić, w jaki sposób boty celują w Ciebie. Wygląda na to, że gra określa dokładność bota poprzez losowe przesuwanie kamery po małym kółku. Dla początkujących robotów ten krąg jest ogromny, w związku z czym boty rzadko strzelają tam, gdzie powinny. Z drugiej strony, koszmarne boty mają tak małe kółko, że dosłownie za każdym razem trafiają w trupa (co wyjaśnia również, dlaczego czasami tęsknią za tobą na szynie, ale rzadko). Pojawia się jeden bardzo interesujący problem. Początkowo, gdy zacząłem walczyć z koszmarnymi botami Quake'a III, nie mogłem ich pokonać. Teraz, na niektórych poziomach, mogę podnieść do sześciu i wciąż wygrywać. W jakiś sposób wątpię, że gdybym grał przeciwko sześciu ludziom, wygrałbym. Jednym z największych problemów ze sztuczną inteligencją (AI) jest to, że komputery są zbyt inteligentne lub zbyt głupie. W przypadku Quake III, inteligencja botów jest wszędzie na mapie, z wyjątkiem tego, gdzie powinna być. Czy nie byłoby dobrze, gdyby komputer mógł być dokładnie na twoim poziomie umiejętności? W ten sposób, gdy twoje umiejętności wzrosną, tak samo będzie z komputerem. W końcu gra przeciwko komputerowi powinna być uczciwą walką dla obu stron, a nie sytuacją, w której jedna strona zostanie poważnie pobita. Wszystko wskazuje na to, że istnieją dwa główne typy sztucznej inteligencji:

■ Wszechwiedząca sztuczna inteligencja, która zna całą mapę i dokładnie to, co porusza komputer który powinien zrobić, aby wygrać. Ten typ sztucznej inteligencji nie jest szczególnie interesujący z matematycznego punktu widzenia, ponieważ dotyczy głównie wyszukiwania min / maks w strukturach lub czymś wzdłuż tych linii.

■ Adaptacyjna sztuczna inteligencja, która umożliwia komputerowi dostosowanie się do otoczenia. Quake to dobry przykład gry, która korzysta z tego typu sztucznej inteligencji, ponieważ w Quake, komputer nie może wyliczyć wszystkich możliwych scenariuszy. Jak dokładnie działa sztuczna inteligencja? Sztuczna inteligencja działa głównie poprzez wykorzystanie prawdopodobieństwa i statystyk, które są przedmiotem niniejszego rozdziału. W tym rozdziale omawiamy następujące zagadnienia:

■ Prawdopodobieństwo

■ Permutacje

■ Kombinacje

■ Generowanie liczb losowych

■ Dystrybucja

■ Algorytmy genetyczne

■ Logika rozmyta

■ Sieci neuronowe

Podstawowe zasady statystyki

Statystyka jest interesującym rozdziałem w matematyce, ponieważ dotyczy nieznanego. Ideą danych statystycznych jest umożliwienie zgrania kilku liczb w celu przewidywania przyszłych wyników. Co

dziwne, statystycznie poprawne wyniki rzadko kiedy pasują do rzeczywistości. Na przykład, jeśli uczeń ma średnią ocenę 2,346, czy spodziewałbyś się, że w następnym kursie uzyska ocenę 2,346? Prawdopodobnie nie. Można jednak powiedzieć z pewną dozą pewności, że jego ocena będzie w przybliżeniu równa tej wartości punktowej. Chodzi o to, że odpowiedź w rzeczywistości nie odbiegnie gwałtownie od oczekiwanego rezultatu. Dziedzina statystyki opiera się na następujących narzędziach, z których każdy jest omawiany w następujących sekcjach:

■ Prawdopodobieństwo

■ Permutacja

■ Kombinacje

Prawdopodobieństwo

Prawdopodobieństwo to ocena prawdopodobieństwa wystąpienia określonego zdarzenia. Na przykład rozważ grę w szachy, w której komputer musi ustalić, czy przeniesienie rycerza z jednej pozycji do drugiej jest dobrym posunięciem. Aby zdecydować, komputer może spojrzeć na swoje przeszłe doświadczenia. Jeśli w przeszłości wykonanie identycznego ruchu zakończyłoby się sukcesem, komputer mógłby zdecydować, że powtórzenie tego samego ruchu byłoby dobrym pomysłem. Innymi słowy, komputer może zdecydować, że jest wysoce prawdopodobne, że jest to dobry ruch. Z drugiej strony, jeśli przeniesienie tego utworu w przeszłości doprowadziło do wzięcia królowej komputera, a nawet króla, komputer mógłby zdecydować się na rozważenie alternatyw. Oznacza to, że komputer może zdecydować, że jest to wysoce nieprawdopodobne, że jest to dobry ruch. Jednym ze sposobów zmierzenia tych abstrakcyjnych pojęć - prawdopodobnych i nieprawdopodobnych - jest użycie teorii mnogości i zapisanie ich w kategoriach procentowych. Jako przykład wykorzystać grę szachową, przypuśćmy, że z biegiem czasu komputer wykonał ten sam ruch 100 razy i że udało się to 80 razy. W takim przypadku można powiedzieć, że ruch ma wskaźnik sukcesu równy 80%. Kolejny przykład pochodzi z gier typu shoot-em-up. Załóżmy na przykład, że bot trzyma strzelanie do ciebie podczas gry, i że gra ma narzędzie, którego możesz użyć do śledzenia liczby strzałów w ciebie strzelonych i jak wiele z nich faktycznie cię trafiło. Jeśli pod koniec gry ustalisz, że bot wystrzelił na ciebie 1000 razy i uderzył cię 300 razy, możesz obliczyć współczynnik - i, przez rozszerzenie, procent - aby opisać prawdopodobieństwo uderzenia bota w przyszłość:

300/1000 = 0,3 = 30%

Matematyczne prawdopodobieństwo, że p (x) zdarzenia x jest spełnione, jest następujące:

p(x) = Powodzenie dla x / Całkowite Próby dla x

Mając tę definicję w zasięgu ręki, możesz przyjrzeć się zakresowi, który powinna wykazywać p. Na początek wiesz, że prawdopodobieństwo nie może być ujemne, ponieważ ani całkowita liczba prób, ani całkowita liczba sukcesów nie mogą być ujemne. Ponadto wiesz, że nie możesz mieć więcej sukcesów niż prób. Najlepszym możliwym scenariuszem byłaby sytuacja, w której wszystkie próby zakończyły się sukcesem. W takim przypadku otrzymasz prawdopodobieństwo 1 lub 100%. Jeśli z drugiej strony wszystkie próby były błędami, otrzymalibyśmy prawdopodobieństwo 0. W konsekwencji zakres p (x) wynosi [0, 1]. Jak byś obliczył prawdopodobieństwo nieosiągnięcia przez x, znanego również jako obliczanie dopełnienia prawdopodobieństwa? Możesz znaleźć uzupełnienie odejmując sukcesy od całkowitych prób, zasadniczo odwracając procent:

$$p(\overline{x}) = 1 - p(x)$$

W teorii mnogości pasek służy do reprezentowania dopełnienia zbioru. Uzupełnienie może być uważane za "wszystko oprócz". Wracając do przykładu na bota, możesz określić szanse, że bot nie uderzy Cię w następujący sposób:

1? 0,3 = 0,6 = 60%

Prawdopodobieństwo zdarzeń symultanicznych

Załóżmy, że twoja gra nie ma botów, ale zamiast tego ma zautomatyzowane wieżyczki, które strzelają. Każda wieżyczka ma własne statystyki. Oznacza to, że każdy z nich wie, jakie są możliwe strzały w celu trafienia w cel. To jest twoje pierwsze prawdopodobieństwo. Twoje drugie prawdopodobieństwo jest, czy, biorąc pod uwagę cel trafienia, strzał jest śmiertelny. Możesz również być zainteresowany określeniem prawdopodobieństwa, że dana wieża strzela śmiertelnie w inny sposób, prawdopodobieństwo, że wieżyczka trafi i zabije jej cel. Jeśli prawdopodobieństwo jest niskie, możesz rozważyć cel innej wieży dla obiektu, jeśli znajduje się on w przestrzeni krytycznej. Matematycznie można to wyrazić w następujący sposób: gdzie x oznacza wieżę lądującą na hicie, a y oznacza trafienie śmiertelne:

$$p(x \cap y)$$

Jeśli myślisz o tym tylko w kategoriach równania prawdopodobieństwa, możesz zauważyć, że wskaźnik sukcesu rzeczywiście się pomnożył. Dla każdego przypadku, w którym x jest sukcesem, y także musi odnieść sukces, aby całe prawdopodobieństwo mogło odnieść sukces. Jeśli więc pomnożymy prawdopodobieństwo sukcesu dla x przez prawdopodobieństwo dla y, otrzymamy prawdopodobieństwo, że oba zdarzenia wystąpią w tym samym czasie.Matematycznie, wyraża się to w następujący sposób:

$$p\left(\bigcap_{i=1}^{n} x_i\right) = \prod_{i=1}^{n} p(x_i)$$

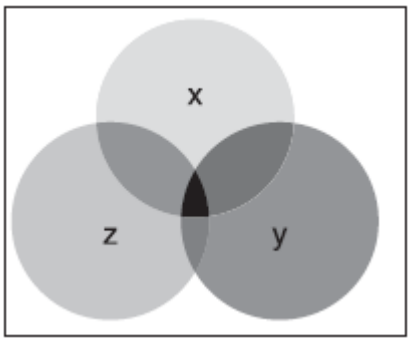
Jeśli szanse na postawienie danej wieży osiągnęły cel 0,2, a szanse na zabicie celu wynosiły 0,5, to szanse na trafienie a zabicie celu wynosiłoby 0,1 lub 10%. Tak więc można rozsądnie przewidzieć, że jeden strzał na 10 zabije cel. Ma to sens, gdy weźmiesz pod uwagę, że jeden strzał z pięciu trafi w cel i że potrzeba średnio dwóch trafień, by zabić cel. Aby komplikować sytuację, załóżmy, że jedna wieża jest w tyle, więc zdecydowałeś się umieścić dwie wieże na tym samym celu. W tym przypadku chcesz obliczyć szanse, że jedna z dwóch wież trafi w cel. Z matematycznego punktu widzenia to, czego szukasz, to :

$$p(x \cup y)$$

Na początku możesz uwieść myśl, że rozwiązanie jest niezwykle łatwe - wystarczy, że dodasz szanse x i szanse na trafienie celu. Niestety nie zawsze tak jest. Na przykład, co się stanie, jeśli obie wieże trafią w cel? Ponieważ suma szans na trafienie celu przez pierwszą wieżę i szanse na drugą wieżę trafiającą w cel, razem wzięto pod uwagę możliwość, że obie wieże osiągnęły dwukrotnie sukces. W związku z tym, jeśli odejmiesz szanse obu wież trafiających w cel, otrzymasz prawidłowe równanie, które oblicza szanse, że którakolwiek z wież trafi w cel:

$$p\left(\bigcup_{i=1}^{n} x_i\right) = \sum_{i=1}^{n} p(x_i) - \sum_{i=1}^{n}\sum_{j=2}^{n-1} p(x_i \cap x_j) + \cdots \pm p\left(\bigcap_{i=1}^{n} x_i\right)$$

Jeśli zastosujesz ten pomysł rekursywnie, dojdziesz do wniosku, że jeśli masz trzy elementy, będziesz musiał obliczyć przypadek, w którym każda para wież zakończy się sukcesem. Robiąc to jednak, będziesz musiał przesadzić z przypadkiem, w którym wszystkie trzy wieżyczki odniosły sukces. W związku z tym w tym przypadku należy odjąć od poprzedniego odejmowania lub, jeśli wolisz, dodać ponownie prawdopodobieństwo przecięcia wszystkich trzech elementów. Zastosuj tę logikę do n równoczesnych elementów i uzyskaj poprzednie równanie. Jest to powszechnie określane jako zasada wyłączenia z włączenia. Możesz sobie wyobrazić, co się dzieje, badając przecięcie okręgów (zwane diagramem Venna), jak pokazano na rysunku 11.1.



Aby pomóc ci lepiej zrozumieć ten pomysł, załóżmy, że masz trzy wieżyczki z wskaźnikami sukcesu odpowiednio 0,2, 0,4 i 0,6. Możesz obliczyć szanse, że przynajmniej jedna wieżyczka trafi w cel w następujący sposób:

$$\begin{aligned} p(x \cup y \cup z) &= p(x) + p(y) + p(z) - p(x \cap y) - p(x \cap z) - p(y \cap z) + p(x \cap y \cap z) \\ &= 0.2 + 0.4 + 0.6 - 0.2 \cdot 0.4 - 0.2 \cdot 0.6 - 0.4 \cdot 0.6 + 0.2 \cdot 0.4 \cdot 0.6 \\ &= 0.808 \end{aligned}$$

Nie jest źle! To około 80%, co oznacza, że pomiędzy trzema wieżami cel zostanie zabity cztery razy na pięć. Jako interesujący zwrot, przypuśćmy, że wiesz, że wieżyczka trafiła i zabiła cel. Co więcej, znasz prawdopodobieństwo śmierci po trafieniu, ale nie prawdopodobieństwo trafienia w wieżę. Dedukowanie to jest podstawową algebrą i jest czytane jako "prawdopodobieństwo zdarzenia b wiedząc, że zdarzenie miało miejsce":

$$p(a \cap b) = p(a)\,p(b)$$

$$p(b \mid a) = \frac{p(a \cap b)}{p(a)}$$

Permutacja

Permutacja jest podstawowym narzędziem prawdopodobieństwa, które pomaga ci policzyć sposoby, w jakie można zamówić lub wybrać przedmioty. Może to być przydatne, gdy chcesz obliczyć prawdopodobieństwo, ale musisz wiedzieć, ile sposobów możesz uporządkować swoje elementy. Jest to również dobre narzędzie do obliczania złożoności problemu, jak zobaczysz, kiedy zagłębisz się w

część statystyki statystyki sztucznej inteligencji. Aby to zilustrować, załóżmy, że masz trzy obiekty - a, b, i c- oraz trzy pola, i chcesz określić liczbę sposobów, w jakie obiekty i pola mogą być sparowane. Podczas wybierania obiektu, który ma zostać umieszczony w pierwszym polu do wyboru masz trzy obiekty. Przy wyborze obiektu, który ma zostać umieszczony w drugim polu, masz do wyboru dwa obiekty, ponieważ jeden obiekt został już umieszczony w pudełku i dlatego nie jest już wybierana .Kiedy wybierając obiekt do umieszczenia w trzecim polu, masz tylko jeden obiekt do wyboru, ponieważ pozostałe dwa obiekty są już w pudełku. Matematycznie można określić, że masz 3 • 2 • 1 = 6 sposobów lub permutacji parowania obiektów i pól. Jeśli je wyliczysz, zobaczysz, że tak właśnie jest: {a, b, c}, {a, c, b}, {b, a, c}, {b, c, a}, {c, a, b}, {c, b, a}. Co by się stało, gdybyś miał tylko dwie skrzynki, ale nadal trzy przedmioty? W takim przypadku do wyboru byłyby trzy obiekty, decydujące o tym, który obiekt zostanie umieszczony w pierwszym polu, oraz dwa obiekty do wyboru przy podejmowaniu decyzji, który obiekt zostanie umieszczony w drugim polu. W ten sposób można użyć równania 3 • 2 = 6, aby ustalić, że liczba permutacji się nie zmienia. Aby to sprawdzić, wystarczy spojrzeć na pierwsze dwa elementy zestawu trzech; każdy jest wyjątkowy. Mówi ci to, że ogólnie permutacja r elementów wybranych z n elementów jest następująca, w której ja! dla danej liczby i jest zdefiniowany jako iloczyn i, i? 1, ja? 2, aż do 1:

$$P(n,r)=\frac{n!}{(n-r)!}$$

Działa to tylko wtedy, gdy pola są rozpoznawalne. Jeśli po prostu twierdzisz, że pudełko jest pudełkiem, kolejność nie ma znaczenia, a ty naprawdę patrzysz na kombinowany problem. Matematycznie, co następuje:

$$i!=\prod_{j=1}^{i} j$$

Kombinacje

Kombinacje są podobne do permutacji, ale nie stanowią porządku. Innymi słowy, kombinacja jest permutacją, w której żadne dwa zestawy nie zawierają dokładnie tych samych elementów. Załóżmy na przykład, że jesteś na kolacji i masz możliwość wyboru dwóch dań z czterech dostępnych. W takim przypadku kolejność selekcji nie ma znaczenia i można sprawdzić, czy istnieje sześć możliwości. Jednak powracając do przykładu obiektu / pudełka z poprzedniej sekcji, załóżmy, że masz tyle obiektów, ile robisz. W takim przypadku kombinacja równa się permutacji i podobnie, jeśli wybrałeś tylko jeden obiekt z całego pakietu. Jest to oczywiste, jeśli po prostu myślisz o definicji kombinacji w odniesieniu do permutacji. Przypomnijmy, że kombinacje są naprawdę permutacjami, w których kolejność nie ma znaczenia. Jeśli musisz wybrać jeden z n, to w takim przypadku kolejność nie ma znaczenia. Podobnie, jeśli chcesz pominąć jeden obiekt, obowiązuje ta sama logika, ale w odwrotnej kolejności. Pominięcie jednego obiektu nie ma sensu, jeśli usuniesz jeden obiekt (lub wybierzesz n - 1 z n). Ale jak ogólnie rozwiązać problem kombinacji? To naprawdę nie jest takie skomplikowane. W permutacji P (n, r) oblicza się liczbę sposobów pozycjonowania r elementów wybranych z grupy n. Następnie musisz usunąć zestawy w permutacji, które są takie same, ale uporządkowane inaczej. Co ciekawe, już wiesz, że r elementów można ustawić na r! sposoby. Tak więc, dla każdej możliwości, r! istnieją sposoby pozycjonowania wybranych obiektów. W konsekwencji, jeśli podzielisz tę kwotę, usuniesz wszystkie zestawy inaczej uporządkowane:

$$C(n,r)=\binom{n}{r}=\frac{P(n,r)}{r!}=\frac{n!}{r!(n-r)!}$$

Możesz zastosować ten pomysł rekurencyjnie, aby uzyskać formułę wielomianową, która jest jedynie uogólnioną kombinacją. Załóżmy na przykład, że chcesz utworzyć cztery grupy jednostek powietrznych do swojej gry. Chcesz ustalić, ile sposobów możesz to osiągnąć; oczywiście nie zależy ci na kolejności wyborów, ponieważ są to grupy bojowe. Jeśli wykonasz matematykę dla 16 samolotów z grupami 5, 4, 3 i 2, otrzymasz:

$$\begin{aligned}\binom{16}{5}\binom{11}{4}\binom{7}{3}\binom{4}{2}&=\frac{16!}{5!11!}\frac{11!}{4!7!}\frac{7!}{3!4!}\frac{4!}{2!2!}\\&=\frac{16!}{5!4!3!2!2!}\\&=\binom{16}{5,4,3,2}\end{aligned}$$

Kombinacje stają się również dość potężnym narzędziem do obliczania współczynnika mocy wielomianowej. Współczynnik wielomianu dla danego zestawu mocy można obliczyć w następujący sposób:

Dla $\left(x_1+x_2+x_3+\cdots\right)^n$

współczynnik z $x_1^{a_1}x_2^{a_2}x_3^{a_3}\cdots$

to $\binom{n!}{a_1,a_2,a_3,\ldots,a_i}$

To może być całkiem skutecznym narzędziem rozwiązywania problemów. Na przykład współczynnik wielomianu $(x + y + z)^8$ dla $x^2y^2z^4$ wynosi 8! / (2! 2! 4!). Możesz rozwiązać mnóstwo problemów z tego typu narzędziami, ale ponieważ nie zawsze jest to użyteczne w grach, opisałem je krótko.

Generowanie liczb losowych (Jednolite Odchylenia)

Czym dokładnie jest przypadkowość? Słownik definiuje go jako pozbawiony planu, celu lub wzorca. Jako matematyk wolę myśleć o przypadkowości jako braku postrzeganego porządku. Mówię o tym, ponieważ wiele rzeczy, które uważasz za losowe, takie jak wynik rzutu kostką, można wyjaśnić, obliczyć i powtórzyć. W końcu czy tak trudno jest uwierzyć, że jeśli ktoś pozycjonuje, trzęsie i wypuszcza parę kostek dokładnie w taki sam sposób jak poprzedni rzut, to nie może osiągnąć tego samego wyniku? Robiąc krok dalej, czy osoba ta nie mogłaby powtórzyć ćwiczenia kilka razy, zestawić tabele i ocenić wyniki, i wykorzystać te informacje do przewidywania wyniku kolejnych rzutów? Oczywiście, jest tak wiele parametrów i zmiennych związanych z rolowaniem kości, siłą, kątem, i tym podobnych - że prawie niemożliwe byłoby pozycjonowanie, potrząsanie i wypuszczenie pary kości dokładnie w taki sam sposób wiele razy. Chodzi jednak o to, że nie ma czegoś takiego jak prawdziwa liczba losowa. W kontekście gry nie jesteś zainteresowany uzyskaniem prawdziwego generatora liczb losowych w takim samym stopniu, jak uzyskiwanie liczb pseudolosowych (to jest liczb, które wydają się losowe, ale w

rzeczywistości nie są, sposób, w jaki wynik rzutu kostką wydaje się losowy, ale nie jest to przypadek losowy) t) w określonym zakresie i, co najważniejsze, szybko osiągnąć to zadanie. Problem w tym, że starzejące się standardy językowe nie uwzględniają pełnych możliwości programistów, które teraz posiadają, a zatem penalizują programistów. Na przykład standardowe funkcje srand () / rand () są pozostałością, która może generować tylko 16-bitowe wartości. Co się dzieje, gdy potrzebujesz wartości pływającej, podwójnej, 32-bitowej lub nawet 64-bitowej? Używasz jednolitych odchyleń - to znaczy liczb losowych z równym prawdopodobieństwem w danym zakresie - które są najszybszym sposobem wygenerowania rozsądnego zestawu losowego. Ponadto, jeśli znasz SIMD, możesz skorzystać z tej wiedzy, aby czterokrotnie zwiększyć prędkość generowania losowych liczb. W statystyce generowanie liczb losowych można wykonać na różnych dystrybucjach. Rozkład definiuje prawdopodobieństwo, że liczba zostanie wybrana.

Uwaga

Co się dzieje, gdy potrzebujesz wartości pływającej, podwójnej, 32-bitowej lub nawet 64-bitowej? Oczywiście, są na to sposoby, ale uważam te techniki za hacki. Nie są one optymalne, ale techniki wymienione w tej książce zapewniają zasłużoną aktualizację do standardowego generatora liczb losowych bibliotek C / C ++.

Całkowita Kongruencyjna Standardowa Metoda Liniowa

Jak wspomniano wcześniej, równomierne odchylenia są liczbami losowymi, które są równie prawdopodobne w danym zakresie. Istnieją inne rodzaje liczb losowych, ale nie obchodzi ich to, ponieważ ich celem jest zwykle odchylenie pewnych liczb ze średnią i odchyleniem standardowym, takim jak rozkład Gaussa. Istnieje jednak metoda generowania liczb pseudolosowych, zwana standardową metodą kongruencji liniowej. W skrócie, kongruencja oznacza, że występuje działanie modułu; liniowy oznacza, że każda liczba jest równie prawdopodobna (odchylenia równomierne). Ta metoda w szczególności dotyczy liczb całkowitych. Tylko co sprawia, że dobry generator liczb losowych w kontekście gry?

■ Na początek chcesz, żeby liczby wyglądały dobrze, przypadkowo. Innymi słowy, ty nie chcę, by generowanie liczb było zgodne z oczywistymi wzorcami.

■ Chcesz również, aby liczby obejmowały cały zakres. W końcu generator liczb losowych, który może generować tylko 10 liczb w zakresie 10K, nie będzie zbyt dobry dla większości projektów.

■ Niezwykle ważne jest, aby numer był generowany szybko. Nie ma nic bardziej frustrująca niż gra, która zatrzymuje się podczas generowania losowych cząstek wybuchu.

Co zatem oznacza standard ANSI C, który sugeruje, co większość obecnie wprowadzanych bibliotek C / C ++ w odniesieniu do generowania liczb losowych zaleca? Prawda jest taka, że komitet nie zaleca niczego. Daje jedynie przykład i zauważa, że jakakolwiek funkcja jest zaimplementowana, powinna wytworzyć dobrą losową sekwencję. Ich przykładem był górny 16-bitowy równanie cykliczne:

$$N_i = (1103515245N_{i-1} + 12345) \bmod 32768$$

W skrócie, moduł gwarantuje, że liczba losowa będzie zawarta w zakresie [0, 32767]; jego bardzo uproszczona forma stanowi raczej atrakcyjną opcję. Mimo to, ma kilka pytań. Dlaczego wybierasz te liczby i dlaczego one działają? Nie licząc zbyt wiele na matematykę, liczba losowa zawsze musi być ograniczona; stąd moduł tutaj jest oczywistym wyborem, ponieważ wiesz na pewno, że reszta nie może być większa niż dzielnik. Jeśli chodzi o liniową funkcję poprzedzającą moduł, istnieje w rzeczywistości mnóstwo funkcji, które można tu wstawić. Innym popularnym wyborem, który jest tutaj pokazany, jest

rodzina opóźnionych serii Fibonacciego, która jest niczym innym jak sekwencją dodatków buforowych. Niemniej jednak wszystkie metody mają problem nie być tak szybki, jak liniowy kongruencji dla równych entropii.

$$N_i = (N_{i-10} + N_{i-7}) \bmod 24$$

Najprostszym (najszybszym) nierównomiernym równaniem, które można zastosować, jest (Ni-1 + b), gdzie b jest stałą, ale terminy te dają bardzo przewidywalny i ciągły zbiór, który nie jest pożądany. Następnym logicznym krokiem jest próba (aNi-1 + b), gdzie a i b są stałymi. Ta funkcja jest wciąż ciągła i przewidywalna, ale jeśli twoja zmienna a jest wystarczająco duża, zostanie cofnięta w dół przez moduł, który da niezbyt przewidywalny zbiór. Niektórzy teoretycy pokazali, że całkowite usunięcie b może również przynieść dobre wyniki przy ostrożnym wyborze, ale ponieważ instrukcja dodawania jest dość tania, zachowasz ją. Ważne jest, aby pamiętać, że ze względu na naturę modułu terminy w sekwencji ostatecznie się powtórzą, a długość cykli będzie mniejsza lub równa dzielnikowi. Ogólna postać liniowego kongruencyjnego generatora liczb losowych jest podana tutaj:

$$N_i = (aN_{i-1} + b) \bmod c$$

Wracając do przykładu ANSI / C i rzeczywistych wartości a = 1103515245 i b = 12345, istnieje wiele innych liczb, z których można wybrać, chociaż nie można wybrać dowolnej liczby. Na przykład w latach sześćdziesiątych IBM rozprowadził dobrze znany algorytm o wartościach a = 65539, b = 0, c = 231. W przypadku dystrybucji 1D wszystko wygląda dobrze. Rozkład 2D również wygląda całkiem nieźle. Patrząc na rozkład 3D pod określonym kątem, szybko zauważasz, że pojawia się wyraźny wzorzec. Lekcja: Aby uzyskać dobrą sekwencję, musisz przestrzegać kilku zasad; nawet wtedy nic nie gwarantuje, że użyte liczby będą dobre dla wszystkich aplikacji. Ponieważ pracujesz z komputerami, twoje idealne liczby dla c zawsze będą logiczną granicą zmiennej (to jest 8- / 16- / 32- / 64- / 128- / ... bit). Jaka jest zatem dobra wartość dla aib? Na początek wartość a lub b powinna być duża lub wyraźna funkcja liniowa będzie zauważalna. Dowód: W 1961 roku Greenberger udowodnił, że sekwencja ma maksymalny okres c dla liczb 2n wtedy i tylko wtedy, gdy:

■ b nie jest nieparzysty

■ (a - 1) to wielokrotność 4

Microsoft wydaje się używać następującego równania dla jego funkcji Visual int C ++ .NET int rand ():

int rand () = ((Seed = (Seed * 214013) + 2531011) / 65536) mod 32768

Równanie to jest idealnie zgodne z poprzednimi twierdzeniami o wartościach a = 214013, b = 2531011 i c = 215. Ciekawostką jest tutaj dodany podział. W dowolnej losowej sekwencji, która wykorzystuje liniowe równanie kongruencji, najniższe bity zawsze mają najgorszą entropię. Z tego powodu równanie najpierw dzieli wartość 65 356 przed obliczeniem pozostałej części równania. Podczas ustalania losowej liczby pasującej do zakresu zaleca się podzielenie liczby tak, aby pasowała do zakresu zamiast stosowania innego modulo i pobierania niższych bitów liczby. W scenariuszu gry nie interesuje cię jednak tworzenie lepszych (czyli losowych) liczb losowych, więc z przyjemnością puścisz ten dodatkowy podział. Teraz, gdy rozumiesz teorię za tym, przejdźmy do problemu. Pamiętasz wspomniany wcześniej hack, którego możesz użyć do uzyskania 32-bitowych liczb losowych? Proces obejmuje wyłącznie obliczenie dwóch liczb losowych; przypisujesz 16 pierwszych bitów do pierwszej liczby, a dolne 16 bitów do dolnej liczby. To jest złe z dwóch powodów:

■ Obliczasz dwie liczby losowe - dwie multiplikacje, dwa przesunięcia w prawo i dwa ands w przypadku Microsoftu

■ Wydajesz tylko 32 768 możliwych liczb, podczas gdy 32-bitowa liczba całkowita powinna generować 2^{32} unikalne liczby

Teraz wiesz, jak utworzyć liniową funkcję kongruencji, ponieważ jedyną rzeczą, którą ty potrzebujesz są wartości dla a i b do podłączenia do równania dla wartości 32-bitowych. Dla 32 bitów, c = 2^{32}; jeśli zastosujesz się do twierdzenia Knutha, że a = 1,664,525 i b = 1,013,904,223, utworzysz 32-bitowy generator liczb losowych. Ta sekwencja ma się dobrze i ma pełne cykle 2^{32} według powyższego twierdzenia.

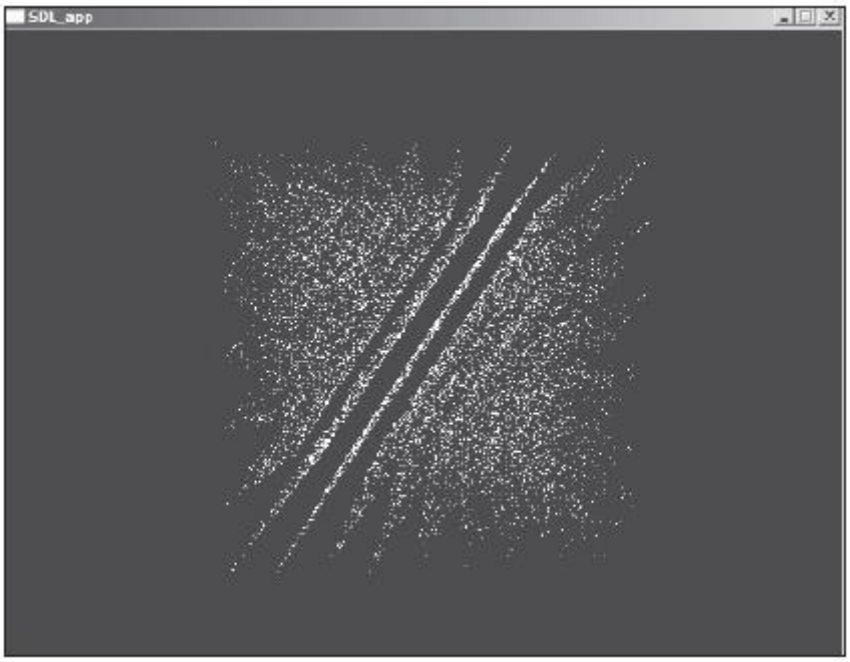
int rand () = Seed = (Seed * 1664525) + 1013904223

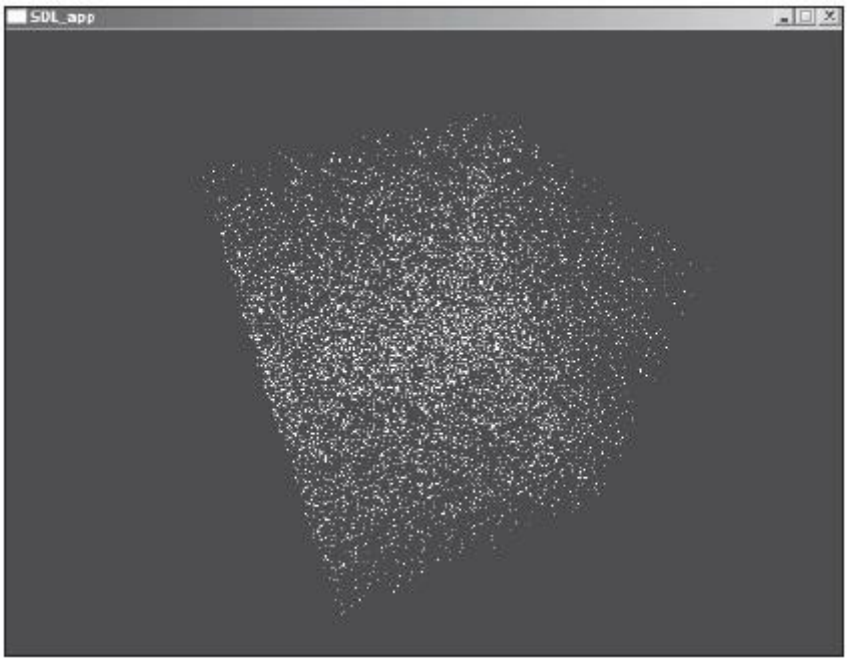
To, co tutaj zrobiłeś, jest dwojakie:

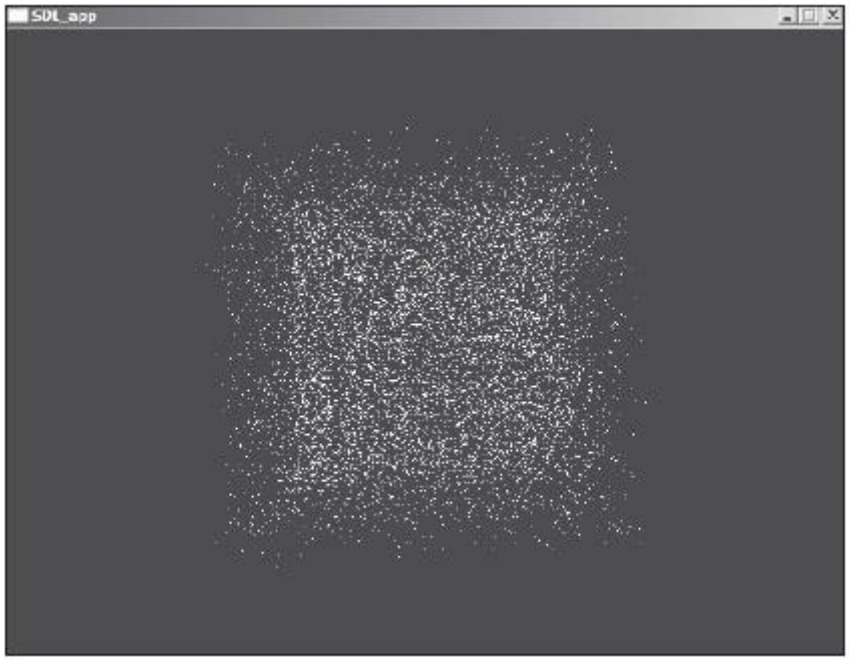
■ Utworzono funkcję, która ma więcej możliwości.

■ Utworzono funkcję szybszą niż jej 16-bitowy odpowiednik.

W odniesieniu do 64 bitów nie wydaje się, aby istniała jakakolwiek zgoda lub przypuszczenie co do dobrego zestawu wartości dla a i b . Dlatego, dla waszych celów, przypuszczam, że a = 49.948,198,753,856,273 i b = 7,478,206,353,254,095,475 są wartościami dobrymi. Zostały one przeanalizowane za pomocą różnych dobrze znanych testów, takich jak test spektralny, obliczanie entropii, korelacja szeregowa, średnia arytmetyczna, Chi-kwadrat, Monte Carlos i kompresja, z których wszystkie dały wystarczająco dobre wyniki dla twoich celów. Najważniejszym testem dla gier jest jednak test spektralny, który ma na celu określenie, jak dobry losowy rozkład wygląda, gdy zostanie narysowany w n wymiarach. Aby to zilustrować, załóżmy, że wykorzystujemy losowe liczby, aby określić lokalizację i prędkość generatora twoich cząsteczek. Oczywiście, jeśli wszystkie wierzchołki generowane przez układ cząsteczek leżą na równoległych płaszczyznach, będzie to wyglądało raczej niezręcznie. Jest to również miejsce, w którym nie działa generator losowy IBM. Zawarte z książką jest program do renderowania widmowego. "Losowo" może renderować losowe zestawy w maksymalnie sześciu wymiarach bez utraty informacji przez kodowanie informacji przez zmianę koloru pikseli. Możesz przetestować swój własny zestaw parametrów; jeśli działa w twoim przypadku, to tylko to się liczy. Test jest dość prosty. Wykreśla wierzchołki, dla których współrzędne są losowo wybierane (co jest dość podobne do układu cząsteczek, który generuje losowe położenia lub prędkości losowe). Na koniec zakończmy tę część ilustrując kilka widm (patrz rysunki 11.2, 11.3 i 11.4) z generatorami liczb losowych przedstawionymi w tej sekcji.


SDL_app


SDL_app



Zmiennopzecinkowa kongruencyjna metoda liniowa

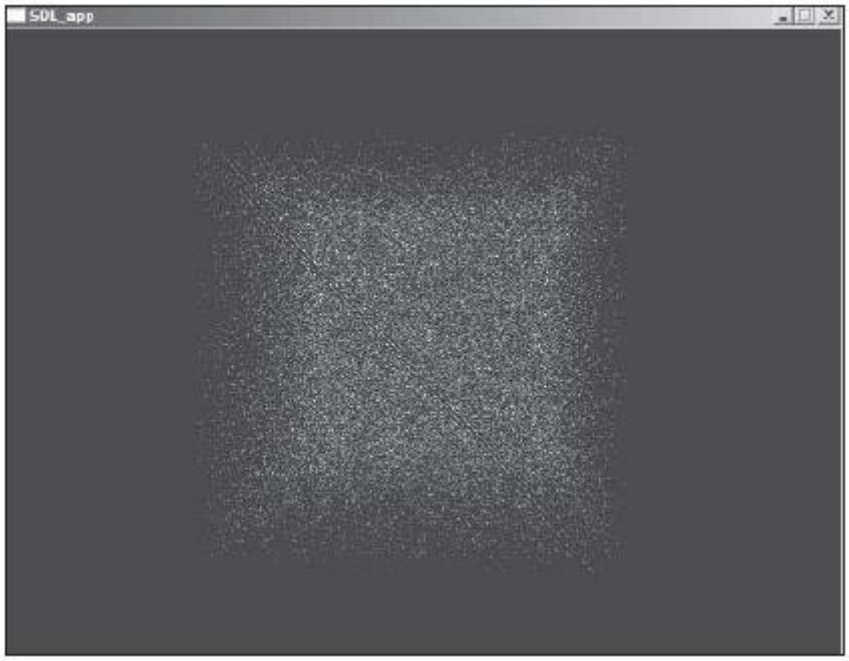
Liczby całkowite są ładne, ale nie wystarczająco dobre. Jeśli poważnie myślisz o grach, to wiesz, że wiele ciekawych grafik korzysta w dużej mierze z wartości zmiennoprzecinkowych. Typowym hackerem do osiągnięcia tego jest wybranie losowej liczby całkowitej, podzielenie liczby przez maksymalną liczbę, którą można wygenerować (stąd utworzenie zakresu od 0 do 1), a następnie pomnożenie liczby w celu dopasowania do zakresu zmiennoprzecinkowego. Nie trzeba dodawać, że pozostawia to sporo miejsca na pewne optymalizacje. Wiesz już, jaka jest liniowa metoda kongruencji (przynajmniej dla liczb całkowitych); teraz wystarczy zastosować podobną koncepcję do pływaków. A może ty? Optymalizacja to jedna z dziedzin wymagających głębokiej wiedzy na temat maszyny, z którą pracujesz. W tym przypadku interesuje Cię bitowa prezentacja wartości zmiennoprzecinkowych na komputerze. Na szczęście dla ciebie ten format jest znormalizowany i dość łatwy. W skrócie, każdy numer zmiennoprzecinkowy można zapisać w następującej formie:

$$\pm(1 + 2^{-mantisa}) * 2^{exponent-127}$$

Jeśli ustalisz wykładnik na 0, abyś pomnożył liczbę przez 1, możesz losować mantysę, aby uzyskać liczbę losową. Pozostaje tylko jeden mały diabeł, którego musisz egzorcyzmować, zanim będziesz mógł nazwać tę metodę kompletną. Przypomnijmy, że w mantysie pierwszy bit jest domyślnie 1, co oznacza, że twoje liczby będą miały postać 1.xxx. Wartość xxx, która jest mantysą, będzie zawierała się w przedziale od 0 do 1, ale nigdy nie osiągnie 1, ponieważ dodanie 1 / 2i dla wszystkich dodatnich liczb całkowitych będzie miało tendencję do 1, ale nigdy do niego nie dotrze. Nie powinno to stanowić problemu, ponieważ każda kolejna operacja najprawdopodobniej naprawi nieskończenie mały brak precyzji. Ponieważ generujesz liczbę losową między 1,0 a 2,0, możesz po prostu odjąć 1, aby uzyskać zakres [0, 1], jak pokazuje poniższy kod:

```
unsigned long Result = (Seed i 0x007FFFFF) | 0x3F800000;

float RandomFloat = ((float) * (float *) i wynik) - 1.0f;
```

Okazuje się, że technika ta jest ostatnią niespodzianką. Patrzenie na widmo tych liczb wartościami Knutha ujawnia, że czwarty wymiar tych wartości eksponuje wzór, co oznacza, że te wartości nie są najlepsze wokół. Rysunek 11.5, który został udoskonalony cyfrowo, ilustruje to.



(Będziesz mógł zobaczyć znacznie wyraźniej, że widmo tego uporządkowanego zestawu nie jest zbyt dobre, jeśli uruchomisz aplikację samodzielnie.) Nie twierdzę, że znalazłem idealne rozwiązanie, ponieważ nie uważam, że istnieje . Przynajmniej widma wartości a = 85899573 ib = 101390421 nie wydają się generować żadnych widocznych wzorów. Czy to oznacza, że powinieneś zmienić wartości 32-bitowego generatora liczb losowych? Nie do końca. Widmo nie ujawniło niczego niepokojącego, gdy zajmowałoby się 32-bitowymi liczbami całkowitymi, ale te wartości mogłyby być lepsze. Przed zakończeniem tej sekcji przyjrzyjmy się podwójnym zmiennym zmiennoprzecinkowym. Podwójne punkty zmiennoprzecinkowe mają ten sam pomysł, co pojedyncze punkty zmiennoprzecinkowe, ale liczba bitów przydzielonych do sekcji jest różna. W podwójnym, wykładnik ma 11 bitów, mantys 52 i znak 1. Jeśli szukasz dokładnego randomizatora, wtedy masz mały wybór, ale musisz obliczyć 64-bitowy, a następnie zastosować maski, a następnie odejmowanie 1. W większości przypadków nie potrzebujesz faktycznie 52-bitowej dokładności dziesiętnej. 32 bity powinny wystarczyć w większości - jeśli nie we wszystkich sytuacjach. W związku z tym, zamiast generowania losowych bitów dla wszystkich 52 bitów, można po prostu wygenerować losowe bity dla pierwszych 32 bitów i pozostawić pozostałe bity na 0. To jest tylko 32-bitowa liczba losowa całkowita, która jest przesunięta, aby dopasować do wymaganego bitu pozycja. Poniższy fragment kodu ilustruje to:

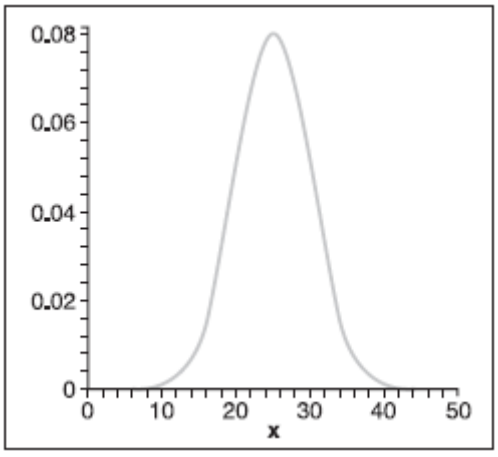
```
unsigned long Wynik [2] = {

((unsigned long) Seed << 20),

((unsigned long) Seed >> 12) | 0x3FF00000,

};

double RandomDouble = (double) * (double *) Wynik - 1,0;
```

Generowanie liniowych nieprzewidywalnych liczb losowych

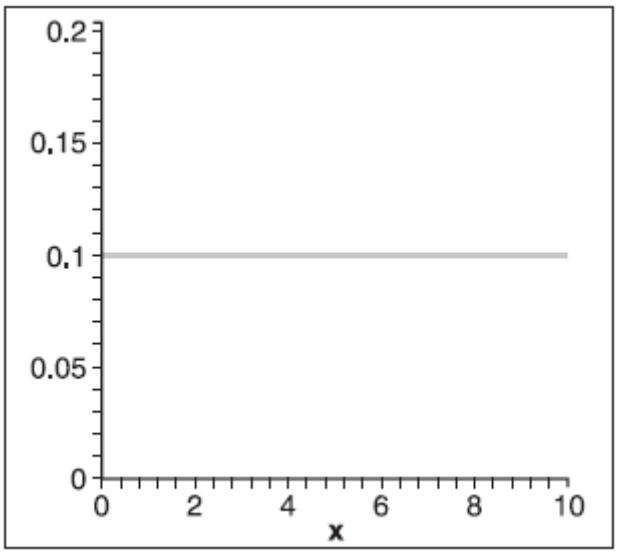
Możesz zdecydować, że interesujące jest generowanie losowej liczby, która nie jest przewidywalna liniowo. Dlaczego? Ponieważ hackowanie w grach wieloosobowych się zwiększyło i wydaje się, że pogarsza się. Tak więc, w niektórych rodzajach gier, jeśli gracz może zdobyć ziarno losowej liczby innego gracza, może w zasadzie przewidzieć liczbę i wykorzystać tę informację na swoją korzyść. Jedną z najlepszych technik generowania liniowo nieprzewidywalnych liczb losowych jest generator białego szumu. Przykłady białego szumu to śnieg, który pojawia się w telewizorze na niezajętych kanałach, hałas kondensatora termicznego i tak dalej. Oczywiście nie masz dostępu do takich materiałów na komputerze, ale możesz używać urządzeń zewnętrznych, aby przesuwać ziarno raz na jakiś czas. Na przykład, w każdej sekundzie można odczytać pozycję myszy i dodać najmniej znaczący bit tej wartości do nasiona. Inne źródło przypadkowości może pochodzić z joysticka lub klawiatury. Twoja wyobraźnia jest granicą. Walcz jednak z hackami, ponieważ naprawdę obniżają współczynnik zabawy.

Rozkład

Rozkład jest ważnym aspektem statystyk, który analizuje, w jaki sposób próbki pobrane z wydarzenia są powiązane. Jak wiesz, gromadzenie statystyk wymaga obserwacji; aby gromadzić dane, musisz pobrać próbki. Czasami będziesz mógł wziąć każdą możliwą próbkę, dając kompletny model. Na przykład możesz wypróbować ocenę każdego ucznia w klasie i wygenerować rozkład tych wartości, a następnie użyć wykresu rozkładu, aby pokazać relatywnie ile osób ma określoną ocenę. Możesz myśleć o grafie rozkładu jako dużej strukturze typu Connect 4. W tym przypadku wykres ma pięć kolumn: A, B, C, D i F. Jeśli odsetek uczniów, którzy otrzymali A, wynosił od 0 do 25%, dodaj jeden "token" do kolumny A. Jeśli odsetek uczniów, którzy otrzymali A wynosił od 25 do 50%, dodaj dwa "tokeny" do kolumny A. Następnie przejdź do kolumny B i dodaj odpowiednio żetony. Kiedy skończysz, będziesz mógł zobaczyć, jak zostały podzielone stopnie. Wykres rozkładu nie jest ograniczony do czterech klasyfikacji (0-25%, 25-50%, 50-75%, 75-100%). W rzeczywistości możesz wykreślić wykres, który ma nieskończoną liczbę podziałów. Wykres rozkładu można zatem postrzegać jako gęstość wartości w zależności od jej możliwych wartości, jak pokazano na rysunku 11.8.



Taki wykres jest zazwyczaj znormalizowany, tak że obszar pod wykresem jest równy 1. W ten sposób całka z jednej granicy do drugiej daje procent próbek w tym zakresie, jak pokazano na rysunku 11.7



Właściwości dystrybucji

Dystrybucja jest zwykle przypisywana szeregowi dobrze znanych czynników, które pomagają

wizualizuj, jak powinna wyglądać dystrybucja:

■ Średnia

■ Wariancja

■ Odchylenie standardowe

Średnia

Średnia jest średnią ważoną możliwych wartości, gdzie waga jest prawdopodobieństwem. W typowej średniej wszystkie wartości mają taką samą wagę / prawdopodobieństwo. Jednak w średniej ważonej różne masy przypisuje się różnym wartościom. Załóżmy na przykład, że masz zestaw liczb {1, 2, 2, 3}. Średnią można obliczyć za pomocą równania (1 + 2 + 2 + 3) / 4. Z drugiej strony można również reprezentować to samo w bardziej dogodny sposób za pomocą wag (1 * 1 + 2 * 2 + 1 * 3) / 4. W drugim przypadku stosowana jest średnia ważona, w której liczba 2 ma wagę lub, jeśli wolisz, prawdopodobieństwo, które jest dwa razy większe od innych liczb. W przypadku funkcji ciągłej można zdefiniować średnią jako taką:

$$\mu = E(X) = \int_{-\infty}^{\infty} x \cdot p(x)\, dx$$

Wariancja

Wariancja jest wartością, która daje wyobrażenie, jak bardzo wartości próbki w zbiorze odbiegają od średniej. Na przykład, jeśli wielu uczniów ma taką samą ocenę, wariancja będzie niska. Z drugiej strony, gdyby wielu uczniów nie zdało egzaminu, a wielu innych go zablokowało, wariancja byłaby wysoka. Jednym ze sposobów określenia wariancji jest obliczenie średniej różnicy między elementem x i μ, Ale ta metoda może być problematyczna, jeśli znak staje się problemem. Zamiast tego po prostu wyrównaj różnicę, aby uzyskać wynik bez znaku. W konsekwencji wariancja jest zdefiniowana jako taka:

$$\begin{aligned}\sigma^2 &= E\left(\left(X-\mu\right)^2\right)\\ &= E\left(X^2\right)-\mu^2\\ &= \left(\int_{-\infty}^{\infty} x^2 p(x)\right)-\mu^2\end{aligned}$$

Odchylenie standardowe

Dla własnych potrzeb odchylenie standardowe można zdefiniować jako pierwiastek kwadratowy odmiany. Odchylenie standardowe jest, ogólnie rzecz biorąc, bardziej użyteczne niż wariacja, głównie dlatego, że działa w tej samej skali, co dane. Kwadrat pierwiastkowy wprowadzony w wariancji zmienia skalę, ale obliczenie pierwiastka kwadratowego z ostatecznych wyników pomaga przywrócić wszystko na tej samej stopie. Innymi słowy, to porównywanie pomarańczy z pomarańczami zamiast porównywania liczb kwadratowych ze zwykłymi liczbami.

Kowariancja

Już widziałeś, czym jest wariancja. Kowariancja jest bardzo ściśle powiązaną zasadą, która jest po prostu bardziej ogólna. Biorąc pod uwagę n zbiorów $\{X_1\}$, ..., $\{X_n\}$, kowariancja $[s]_{ij}$ z x_i i x_j jest matematycznie określona przez

$$\sigma_{ij} = \overline{\left(x_i-\mu_i\right)\left(x_j-\mu_j\right)}$$

Tutaj μ_i reprezentuje średnią i^{tego} zbioru, a słupek przedstawia średnią wszystkich możliwych wartości. Ten rodzaj analizy jest bardzo użyteczny do określenia kierunku zbioru próbek. Na przykład, jeśli chcesz określić kierunek formułowania dla grupy czołgów, możesz zbudować macierz kowariancji. Jest to po prostu macierz, w której każdy element ij reprezentuje wartość kowariancyjną dla ij. Ta macierz przechodzi przez każdą wartość $[s]_{ij}$, jak w tabeli. Ogólnie można zapisać macierz kowariancji C w następujący sposób:

$$\mathbf{C} = \frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{x}_i-\mu\right)\left(\mathbf{x}_i-\mu\right)^{\mathrm{T}}$$

Jeśli napiszesz tę macierz dla zbioru 3-wektorów, otrzymasz następującą macierz:

$$\mathbf{C} = \frac{1}{n}\begin{bmatrix} \sum_{i=1}^{n}\left(x_i-\mu_x\right)^2 & \sum_{i=1}^{n}\left(x_i-\mu_x\right)\left(y_i-\mu_y\right) & \sum_{i=1}^{n}\left(x_i-\mu_x\right)\left(z_i-\mu_z\right) \\ \sum_{i=1}^{n}\left(y_i-\mu_y\right)\left(x_i-\mu_x\right) & \sum_{i=1}^{n}\left(y_i-\mu_y\right)^2 & \sum_{i=1}^{n}\left(y_i-\mu_y\right)\left(z_i-\mu_z\right) \\ \sum_{i=1}^{n}\left(z_i-\mu_z\right)\left(x_i-\mu_x\right) & \sum_{i=1}^{n}\left(z_i-\mu_z\right)\left(y_i-\mu_y\right) & \sum_{i=1}^{n}\left(z_i-\mu_z\right)^2 \end{bmatrix}$$

Pierwszą rzeczą, którą możesz zauważyć, jest to, że macierz jest symetryczna. Tak więc, tylko połowa z nich musi zostać obliczona. Ale w jaki sposób można użyć tej macierzy do określenia, na przykład, ogólnego kierunku grup czołgów? Cóż, powinno być całkiem jasne, że kowariancja jest ściśle związana z wariancją. W rzeczywistości i-ta wariancja dla i-tej grupy składa się z elementów w przekątnej macierzy. W związku z tym ta macierz daje poczucie delt między komponentami (lub offset, jeśli wolisz). Przesunięcie jest wektorem, a zatem daje ogólny kierunek, do którego zmierza grupa. Jeśli masz macierz diagonalną, oznacza to, że wartości są równomiernie rozłożone na każdej osi, a zatem ogólny kierunek to nic innego jak osie. Gdy dowolna kowariancja wynosi 0, oznacza to, że nie ma żadnej korelacji między tymi dwoma elementami. Nadal musisz ustalić podstawę (zestaw trzech wektorów), która da ci ogólny kierunek formowania czołgów. Kluczem do rozwiązania tego problemu jest to, że macierz przekątnej kowariancji ma swoje osie w trywialnej podstawie {<0, 0, 0>, <0, 1, 0>, <0, 0, 1>}. W konsekwencji, jeśli możesz znaleźć podstawę B (macierz rotacji), która przekształci macierz kowariancji w macierz diagonalną, podstawą B będzie twój zestaw kierunkowych wektorów. Wynika to z faktu, że rotacja jest po prostu liniową kombinacją zbioru wektorów, które przedefiniowują osie przestrzeni. Jeśli zastosujesz tę logikę, otrzymasz następujące informacje:

$$\begin{aligned}\mathbf{C}' &= \frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{B}\mathbf{x}_i - \mathbf{B}\ldots_r\right)\left(\mathbf{B}\mathbf{x}_i - \mathbf{B}\ldots_r\right)^{\mathrm{T}} \\ &= \frac{1}{n}\sum_{i=1}^{n}\mathbf{B}\left(\mathbf{x}_i - \ldots_r\right)\left(\mathbf{x}_i - \ldots_r\right)^{\mathrm{T}}\mathbf{B}^{\mathrm{T}} \\ &= \mathbf{B}\mathbf{C}\mathbf{B}^{\mathrm{T}}\end{aligned}$$

Teraz jest dobry moment, aby przypomnieć sobie, czego nauczyliście się w części 3 " - czyli o wektorach własnych i procesie diagonalizacji. Właśnie tam chcesz zacząć korzystać z tej techniki. Aby rozwiązać problem, wystarczy znaleźć macierz B wektorowej matrycy, która spełnia powyższą instrukcję; ta macierz będzie podstawą kierunku zbioru jednostek. Odkrycie to będzie niezwykle ważne w części 15, , kiedy obliczysz objętość graniczną różnych elementów. W tym momencie będziesz chciał poznać ogólny kierunek zestawu wierzchołków, aby określić zasięg i kierunek objętości.

Jednolita dystrybucja

Metody, które wcześniej pokazano, aby generować liczby losowe, mogą generować równomierny rozkład. Jednolita dystrybucja daje równe prawdopodobieństwo każdej próbce. Jeśli spojrzysz na wykres takiego rozkładu, pokazanego na rysunku 11.7, zauważysz, że jest to linia prosta biegnąca od a do b, z dwóch granic. Oznacza to, że prawdopodobieństwo wystąpienia jednej określonej wartości poza zdarzeniem jest równie prawdopodobne, jak prawdopodobieństwo wystąpienia innej konkretnej wartości z tego samego zdarzenia. Jeśli spojrzysz na generowanie liczb losowych, możesz sprawdzić, czy tak właśnie jest. Generator przejdzie przez każdą wartość od a do b, zanim wybierze ponownie tę samą wartość. Każda możliwość ma takie samo prawdopodobieństwo. W prawdziwym życiu, jeśli celujesz w bota, istnieje duże prawdopodobieństwo, że trafisz w okolicę bota, jeśli strzelasz do niego. Ponieważ istnieje większe prawdopodobieństwo, że uderzysz w jeden obszar bardziej niż inny (być może skrajne strony), ten rodzaj dystrybucji nie byłby jednolity. Zazwyczaj, patrząc na rozkład, w rzeczywistości patrzysz na funkcję, która daje gęstość prawdopodobieństwa. Wraca to do przykładu gry Connect 4. Im więcej elementów masz w jednej kolumnie, tym gęstsza staje się ta kolumna. Jeśli spojrzysz na funkcję gęstości dla tego rozkładu, zauważysz, że jest to linia płaska. To nie powinno dziwić, ponieważ każdy element y ma takie samo prawdopodobieństwo. Prawdopodobieństwo, że wynik zdarzenia znajduje się w zasięgu [a, b], jest zatem łatwe do obliczenia. Jedyne, co musisz określić,

to wysokość funkcji gęstości. Jeśli wymusisz obszar pod krzywą na równe 1, otrzymasz następującą logikę dla gęstości h:

$$\begin{aligned} 1 &= p_u(a,b) \\ &= hb - ha \\ h &= \frac{1}{b-a} \end{aligned}$$

Wynika z tego, że prawdopodobieństwo, że wynik zdarzenia mieści się w zakresie [c, d] wynosi

$$p_u(a,b) = \frac{1}{b-a}(d-c)$$

Zwróć uwagę na coś interesującego w tej dystrybucji (i dla każdej ciągłej dystrybucji). Jeśli c = d, otrzymasz 0. Innymi słowy, każda pojedyncza liczba rzeczywista w funkcji ciągłej ma prawdopodobieństwo 0. Ma to sens, jeśli myślisz o problemie w realnej przestrzeni. Jeśli wybierzesz pojedynczą liczbę rzeczywistą w zakresie [a, b], istnieje nieskończona liczba wartości, które możesz zbudować w tym segmencie. Nieskończoność w porównaniu do jednej liczby jest dość słaba, a 1 ponad nieskończoność jest oczywiście 0. Z drugiej strony, jeśli wybierzesz zakres [c, d], masz także nieskończoną liczbę wartości w swoim zakresie, więc jeśli obliczysz stosunek (w nieortodoksyjnej matematyce), symbol nieskończoności zostaje anulowany i otrzymasz procent.

Uwaga

Idea podziału symboli nieskończoności jako taka nie jest jednak ogólnie poprawna; został tu użyty tylko po to, by pokazać, że jedna prawdziwa wartość na nieskończonym zasięgu nie ma szans na bycie wybranym.

Z oczywistych względów średnią tego rozkładu podaje średnia typowa:

$$\mu = \frac{a+b}{2}$$

Podobnie możesz spojrzeć na odmianę, która jest nieco bardziej złożona, ale podana przez:

$$\sigma^2 = \frac{(b-a)^2}{12}$$

Rozkład dwumianowy

Często przydaje się klasyfikowanie wydarzenia po prostu tak udanego, czy nie. W takim przypadku masz dwa możliwe wyniki, a prawdopodobieństwo prawdopodobieństwa dla każdego z nich to p i q. Powszechnie zdarza się, że wydarzenie jest powtarzane do momentu osiągnięcia sukcesu. Wracając do przykładu wieży, wieżyczka wystrzeli, dopóki nie zabije celu. Jedną z rzeczy, której nie uwzględnił ostatni model, jest to, że jeśli wieżyczka nie trafi, ma lepszą szansę trafienia w cel przy następnym wystrzale. Ma to sens, ponieważ ma współczynnik trafień 60%, co oznacza, że można się spodziewać, że w większości przypadków trafią trzy strzały na pięć. Jeśli wiesz, że stracił szansę na strzał, to szanse na trafienie w następny strzał są zwiększone do trzech na cztery. Pomysł, prawdopodobieństwo dwóch

wyników lub powtarzane dwumianowe zdarzenie jest tym, co próbuje modelować dwumianowa dystrybucja. Możesz w rzeczywistości spojrzeć na to jako dwumianowy, gdzie n jest liczbą powtórzeń eksperymentu:

$(p+q)^n$,

Fajną rzeczą w tym zapisie jest to, że jeśli spojrzysz na współczynnik rozszerzenia dla których moc p równa się wielkości sukcesu, dla której chcesz obliczyć prawdopodobieństwo, reprezentuje prawdopodobieństwo otrzymania x sukcesów z n. Na przykład, jeśli uderzy cię AI, jego sukces wzrośnie. Sukces jest zatem procentem sukcesu, jakiego oczekujesz od zdarzenia. Przy bardziej rygorystycznej definicji liczba sposobów na uzyskanie x sukcesów spośród n prób wystąpienia zdarzenia X jest następująca:

$$p_b(X=x) = (\text{Liczba sposobów uzyskania sukcesu x z n}) p^x q^{n-x}$$
$$= \binom{n}{x} p^x q^{n-x}$$

Więc tak naprawdę to rozszerzenie mówi: "Chcę x sukcesów (a więc p^x) i (n - X) niepowodzeń (a więc q^{n-x}). Aby to zilustrować, spróbuj obliczyć szanse wieży z dokładnością 60% oceń, by uzyskać pięć doskonałych ujęć

$$p_b(X=5) = \binom{5}{5} 0.6^5 0.4^0$$
$$= 0.07776$$
$$= 7.7\%$$

Dość szczupłe szanse, jak widać. Zwiększ liczbę od 5 do czegoś wyższego, a otrzymasz jeszcze gorsze wyniki. Jeśli, z drugiej strony, chciałbyś obliczyć szanse posiadania 40% wieżyczki zabijając cel w pięciu strzałach, naprawdę próbowałbyś określić prawdopodobieństwo, że zdarzenie X ma jakąś wartość {1, 2, 3, 4 , 5}. Podsumowując te warunki dasz odpowiedź:

$$p_b(X \le 5) = P(X=5) + P(X=4) + P(X=3) + P(X=2) + P(X=1)$$
$$= \binom{5}{5} 0.4^5 0.6^0 + \binom{5}{4} 0.4^4 0.6^1 + \binom{5}{3} 0.4^3 0.6^2 + \binom{5}{2} 0.4^2 0.6^3 + \binom{5}{1} 0.4^1 0.6^4$$
$$= 0.01024 + 0.0768 + 0.2304 + 0.3456 + 0.2592$$
$$= 0.92224 = 92.2\%$$

Ogólnie rzecz biorąc, nawet cel z dokładnością tylko 40% ma dość dobre szanse na zabicie celu w ciągu pięciu strzałów. Bardziej inteligentnym sposobem rozwiązania tego problemu byłoby po prostu odwrócenie jego uwagi. Innymi słowy, obliczyć prawdopodobieństwo, że wieża straciła wszystkie swoje cele, a następnie obliczyć odwrotność tego prawdopodobieństwa (q = 1 - p). Rozkład dwumianowy nie jest rozkładem ciągłym. Dlatego zamiast używać całek, musisz użyć skończonych sum. To nie powinno stanowić problemu. Prawdopodobieństwo sukcesu jest zawsze takie samo dla tego rozkładu, więc łatwo jest obliczyć średnią i wariancję funkcji - proces, który pozostanie jako ćwiczenie:

$$\mu = np$$
$$\sigma^2 = npq$$

Rozkład Poissona

Rozkład dwumianowy przypisuje pojedynczemu prawdopodobieństwu pojedynczą jednostkę. Inne podejście polega na przyjęciu wielu podobnych jednostek i przyjrzeniu się prawdopodobieństwom dotyczącym grupy jednostek. To jest idea dystrybucji Poissona. Załóżmy na przykład, że korzystasz z garażu czołgów w nocy, a ponieważ tanio korzystasz z materiałów i napraw, części czołgów zaczynają zawodzić. Czy przyjęta z góry stawka zadająca 1,02 czołgów na batalion (złożona ze 100 czołgów) na bitwę oznacza, że przynajmniej jeden czołg na batalion zawiedzie podczas bitwy? Nie. Ta stopa nie różni się od wcześniej omówionych współczynników prawdopodobieństwa. Stopień prawdopodobieństwa, taki jak 60%, jest tak naprawdę tylko średnią; jest całkiem możliwe, aby przejść bitwę, nie tracąc żadnych czołgów. Z drugiej strony możesz stracić trzy lub więcej czołgów podczas pojedynczej bitwy. Oto prawdopodobieństwo: zwykłe odgadnięcie. Wiele złożonej matematyki zajmuje się dystrybucją (funkcją gęstości), więc po prostu wskoczmy do niej. Rozkład procesu Poissona (rozkład Poissona) dla stopy [L] zweryfikowanej dla jednostek x jest następujący:

$$p_p(x) = \frac{\lambda^y e^{-\lambda}}{y!}$$

Wróćmy do przykładu czołgu. Jakie są szanse, że przynajmniej jeden czołg zawiedzie podczas bitwy? Matematyka jest całkiem prosta:

$$\begin{aligned} p_p(X>0) &= 1 - p_p(X=0) \\ &= 1 - \frac{1.02^0 e^{-1.02}}{0!} \\ &= 1 - e^{-1.02} \\ &= 0.6394 = 63.94\% \end{aligned}$$

Jeśli przeliczysz formułę, aby sprawdzić szanse na więcej niż jeden tank, otrzymasz zaledwie 0,72%. Więc jeśli możesz poświęcić jeden czołg, być może wybór na skąpe materiały i naprawy był tego wart. Ponieważ szybkość jest podana jako warunek wstępny procesu Poissona, średnia funkcji jest tym samym stawką. Jeśli wykonasz matematykę dla wariancji, przekonasz się, że to także jest równe stawce. W kategoriach matematycznych:

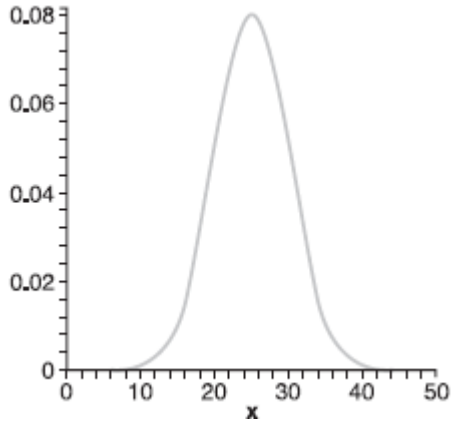
$$\mu = \sigma^2 = \lambda$$

Normalna dystrybucja

Rozkład normalny jest powszechnie stosowany w symulacjach, ponieważ wynika z wielu zjawisk fizycznych. Fizycy czasami nazywają rozkład normalny rozkładem Gaussa, podczas gdy socjolodzy wolą nazywać go krzywą Bell. Ta dystrybucja jest interesująca w grach, ponieważ oferuje dużą elastyczność w symulacji. Piękno tego polega na tym, że masz kontrolę nad zmiennością. Załóżmy na przykład, że grasz w shoot-em-up 3D, a twoje boty mają średnią dokładność 60%. Jeśli napiszesz swój kod tak, że

co sześć pocisków na dziesięć trafi gracza, gracz wkrótce się zorientuje co się dzieje i szybko nauczy się ukrywać podczas szkodliwych strzałów, drastycznie zmniejszając "prawdziwą" dokładność bota. Bardziej realistycznym podejściem jest nadanie botowi odmienności. Na przykład możesz traktować krzywą rozkładu jako funkcję odległości strzału od ciebie. Możesz wtedy powiedzieć, że rozpiętość [a, b] jest rozpiętością, która jest uważana za trafienie (sukces), podczas gdy wszystko poza tym zakresem jest brakiem. Od tego momentu możesz pisać różne wykwalifikowane strzelanki, po prostu zmieniając wariancję botów. Zabójcze boty będą miały bardzo małą wariancję, podczas gdy głupie boty będą miały bardzo długą wariancję. Zaletą tego jest to, że niezależnie od tego, jak dobry jest ten bot, możliwość, że go przegapi, jest zawsze obecna. Oznacza to, że zamiast generować super-bota, generujesz bardziej "ludzkiego" bota. Funkcja rozkładu normalnego jest niczym innym, jak znormalizowanym ciągłym rozkładem Poissona w zakresie od [-∞, ∞], Jak przedstawiono tutaj i zilustrowano na rysunku 11.8:

$$p_n(x) = \frac{e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}}$$



Inteligencja wszędzie

Głównym celem statystyki w odniesieniu do programowania gier jest możliwość wykorzystania go w jakiś sposób do celów sztucznej inteligencji. Podaliśmy już kilka przykładów na temat tego, w jaki sposób możesz zrobić to, aby twoje jednostki były inteligentniejsze i bardziej wydajne, ale można zrobić o wiele więcej dzięki statystyce, aby wykorzystać sztuczną inteligencję. W poniższej sekcji omówiono kilka dobrze znanych technik sztucznej inteligencji i ich dopasowanie do statystycznego modelu gry.

Funkcja wykładnicza

Problem z poprzednimi metodami prawdopodobieństwa polega na tym, że nie pozwalają one na znaczną zmianę. Gra jest dynamiczna, ponieważ zmienia się z czasem. Załóżmy więc, że chcesz zbudować sztuczną inteligencję, która będzie mniej więcej pasować do umiejętności gracza. Na przykład, jeśli ustawisz dokładność bota w Quake na określoną wartość, gracz najprawdopodobniej osiągnie punkt, w którym będzie w stanie pokonać bota na poziomie trudności n, ale w rzeczywistości nie będzie w stanie pokonać bota w trudnej sytuacji poziom n + 1. Rozwiązanie jest zatem proste: Dlaczego w ogóle istnieją jakieś poziomy? Zamiast tego możesz po prostu dopasować umiejętności bota do umiejętności gracza, obliczając niektóre statystyki na odtwarzaczu i dostarczając dane do bota. Załóżmy na przykład, że masz dokładność 20%. Idealnie, bot powinien starać się dopasować tę dokładność, aby dać ci przyzwoite wyzwanie. Po prostu obliczenie średniej dokładności gracza w czasie jest nieskutecznym sposobem określenia poziomu umiejętności, na którym komputer powinien grać. Dlaczego? Załóżmy, że gracz przełącza się na pewną broń, która nagle poprawia (lub pogarsza) jego

grę. Jeśli dokładność gracza jest wykreślona jako średnia, może zająć trochę czasu, zanim zmieniona dokładność zostanie przetłumaczona na dane, które zmieniają poziom umiejętności komputera. Aby to zilustrować, załóżmy, że masz dokładność średnio 8 000 000/10 000 000 lub 80%. Załóżmy, że nowa próbka jest generowana w przeliczeniu na 100. Jeśli gwałtownie spadniesz do 20% dokładności, następna ocena Twojej dokładności wyniesie 8000,020 / 10,000,100, czyli 79,999% - ledwo dostrzegalna różnica. W rezultacie poziom dokładności bota pozostanie mniej więcej statyczny, nawet gdy twoje spadnie; minie trochę czasu, zanim dwie się wyrównają. Wprowadź funkcję wykładniczą. Ideą tej funkcji jest przypisanie określonej wagi do ostatniego przybliżenia; suma wszystkich starszych przybliżeń połączony jest następnie przydzielony pozostały procent. Na przykład, aby podać wagę 50% do bieżącego przybliżenia, można po prostu uruchomić następującą formułę:

$$Size_n = \frac{NewSample}{2} + \frac{Size_{n-1}}{2}$$

Aby to zilustrować, załóżmy, że twoje próbki są opisane w zbiorze {4, 7, 2, 6, 3, 7}. Biorąc pod uwagę powyższe równanie, twoje przybliżenie przyniosłoby {4, 5.5, 3.75, 4.87, 3.93, 5.46}. Jeśli chcesz mniej perturbacji, możesz przypisać mniejszą wartość procentową do nowej wartości; i odwrotnie, jeśli chcesz, aby przybliżenie było bardziej przekrzywione, możesz podać bardziej powszechną wartość. Jeśli wartość próbkowana ulegnie zmianie, niski procent zajmie dużo czasu na adaptację (zmianę), podczas gdy wyższy nie. Z drugiej strony wyższy odsetek może zmienić się zbyt szybko i może nie być w stanie odpowiednio wchłonąć perturbacje. Znasz swoje oprogramowanie; powinieneś wiedzieć, ile próbek powinno się zmieniać w czasie i jak szybko. Jednym z problemów z funkcjami wykładniczymi jest to, że często przewidują liczbę mniejszą niż średnie maksimum. Innymi słowy, jeśli gracz z czasem staje się lepszy, bot nigdy nie spróbuje poziomu dokładności, który jest większy od gracza. Staje się to problemem, ponieważ nie zmusza gracza do poprawy; może wygrać dość łatwo, ponieważ zawsze walczy z gorszym botem. Na przykład, jeśli odwołasz się do dwóch zestawów zamieszczonych powyżej, zauważysz, że przewidywania są mniejsze niż rzeczywista wartość trzech razy na pięć. W swojej grze możesz śledzić minimalny i maksymalny poziom. Chcesz dążyć bardziej do maksimum, ponieważ prawdopodobnie wolisz nieco rzucić wyzwanie graczowi. Innym ważnym faktem jest to, że jeśli gracz osiągnie określoną dokładność w pewnym momencie, prawdopodobnie zawsze będzie mógł powrócić do tego poziomu dokładności w przyszłości, nawet jeśli jego poziom dokładności zostanie przesunięty w czasie. Aby sobie z tym poradzić, oszukuj trochę. Zamiast śledzić maksimum, użyj funkcji wykładniczej na samym maksimum, a następnie możesz zastosować średnią ważoną z maksymalną i aktualną wartością dokładności. Mogą być trochę mylące, więc przyjrzyjmy się głębiej. Załóżmy, że masz taką samą serię liczb całkowitych, jak podano powyżej - {4, 7, 2, 6, 3, 7} - dla dokładności w skali dziesięciu. (Przy okazji, możesz uzyskać te liczby, obliczając liczbę trafień na łączną liczbę trafień dla danego ustalonego interwału.) W ten sposób możesz zdefiniować, że będziesz sprawdzać, ile trafień odniesie sukces na każde 10 strzałów. Za każdym razem, gdy dziesiąte trafienie jest wysyłane, natychmiastowe prawdopodobieństwo powraca do 0/0. Oczywiście, gracz może mieć ogromne wahania dokładności w stosunku do dziesięciu strzałów - zachowanie, którego nie chcesz, aby bot adoptował. Aby tego uniknąć, użyj funkcji wykładniczej, takiej jak poprzednio podany przykład, który dał {4, 5.5, 3.75, 4.87, 3.93, 5.46}. Jednak w wielu przypadkach możesz sprawdzić, czy gracz może zrobić coś lepszego. Jednocześnie zwróć uwagę na maksimum. Jeśli obecne prawdopodobieństwo jest wyższe niż maksimum, możesz ustawić nowe maksimum. Jeśli prawdopodobieństwo jest niższe, w tym miejscu można uruchomić funkcję wykładniczą. Na przykład, jeśli założysz, że używasz wartości procentowej 5% dla maksimum, otrzymasz {4, 7, 6.75, 6.71, 6.52, 7}. Teraz, gdy masz dwa zestawy prawdopodobieństwa, możesz połączyć je po raz ostatni ze średnią ważoną, aby uzyskać ostateczną

dokładność bota. Na przykład, jeśli bieżąca wartość wynosi 80%, a maksymalna wynosi 20%, otrzymasz {4, 5,8, 4,35, 5,23, 4,44, 5,76}. Jak widać, ta nowa metoda rzuca wyzwanie graczowi, aby zrobił to lepiej.

Uwaga

Oczywiście można również zastosować odchylenie do równania, ale problem z uprzedzeniem polega na tym, że może on wiele znaczyć, jeśli dokładność jest bardzo zła i może oznaczać bardzo mało, jeśli dokładność jest już dobra. Dzięki funkcji wykładniczej uzyskujesz całkiem dobrą równowagę.

Logika Rozmyta / Sieci Neuronowe

Czy naprawdę istnieje coś takiego jak absolutna prawda i absolutne kłamstwo? Czy istnieje coś takiego jak nigdy i zawsze? Prawie każda reguła ma wyjątek. Jest to szczególnie prawdziwe w przypadku sztucznej inteligencji. Dobra SI zazwyczaj ewoluuje, doskonali się z biegiem czasu. Proces, w którym się to odbywa, jest czasem nazywany logiką rozmytą, a ideą jest to, że gromadzi zestaw hipotez, weryfikuje je i zamiast mówić, że jest "prawdziwy" lub "fałszywy", daje zbiór hipotez. prawdopodobieństwo prawdomówności. Weźmy na przykład bota, który śledzi ruchy wysokiego poziomu gracza, takie jak skakanie i strzelanie. Zamiast mówić, że gracz zawsze skacze przed strzelaniem, bot oblicza średnią, a nawet lepiej, funkcję wykładniczą (w ten sposób, jeśli nagle przestaniesz skakać i strzelać, AI nie będzie zdezorientowana). Z tym, sztuczna inteligencja może śledź czas, w którym skoczyłeś przed strzelaniem i ile z tych uderzeń się powiodło. Jeśli prawdopodobieństwo przekroczy pewien określony wstępnie próg, wówczas sztuczna inteligencja może spojrzeć na środki zaradcze. Tutaj możesz wstawić jeszcze jedną rozmytą instrukcję. Bot może losowo wypróbować kilka kombinacji i zamiast znaleźć jedną unikalną kombinację, którą zapewnia, że jest "ostateczną prawdą", może śledzić skuteczność swoich broni przeciwko graczowi i może zacząć dopasowywać ataki gracza do różne zabezpieczenia lub kontrataki, które może wykonać bot. Ponieważ ten system jest dynamiczny (z funkcją wykładniczą lub średnią), bot dostosuje się do nowych technik, które na niego rzucasz. To nasuwa kolejne pytanie: Jak wymyślić skuteczny atak lub obronę? W następnej sekcji zasugerowano jedną metodę.

Algorytmy ewolucyjne

Algorytmy ewolucyjne są w dużym stopniu inspirowane przez darwinowski sposób myślenia. Darwin zapewnił, że życie ewoluuje według prawa przetrwania najsilniejszego. Najsilniejsi z paczek żyli, a słabsi umierali. W rezultacie to, co dziś masz, to niektóre z najsilniejszych gatunków, a dokładniej te, które najlepiej się dostosowują. Możesz użyć tego samego pomysłu na oprogramowanie. Jeśli powrócisz do pomysłu bota, który próbuje nowych taktyk, dobrym podejściem, jeśli nie chcesz, aby twój bot przechodził od jednej dobrej taktyki do jednej głupiej taktyki, jest użycie mutacji. Początkowo możesz wygenerować zestaw losowych taktyk (skakać, strzelać, strzelać), (kaczka, ukryć), (strzelać, strzelać) i tak dalej. Następnie możesz ocenić skuteczność tych taktyk. Kiedy wiesz, które z nich działają najlepiej, możesz odrzucić te, które nie zakończyły się sukcesem i wygenerować mutacje tych, które zadziałały. Na przykład, jeśli uznałeś, że (strafe, shoot) była najlepsza opcja, być może następny zestaw mutacji miałby {{ostrzeliwanie, strzelanie, kaczka}, {kaczka, strzelanie, strzelanie} i {strzelanie, ostrzeliwanie}). Zakres mutacji należy do Ciebie. Jeśli chcesz drastycznych huśtawki, to jest droga, ale jeśli wolisz bardziej konserwatywne podejście, możesz robić mniejsze mutacje w dłuższych odstępach czasu.