Jest to rozszerzony zapis odczytu wygłoszonego na seminarium *Popularyzacja Matematyki* na Uniwersytecie Warszawskim w roku akademickim 2006/2007.

Sieci neuronowe

Marek GRABOWSKI, Warszawa

Wstęp

Ostatnio dużo mówi się o „inteligentnych" i „uczących się" programach. Spróbujemy tutaj wyjaśnić, na czym polega „inteligentne" działanie i „uczenie się" sieci neuronowych – chyba najstarszego i najpopularniejszego rodzaju systemów uczących się. Na początek – trochę historii i ideologii, stojącej za sieciami neuronowymi.

1. Podstawy

Sztuczna sieć neuronowa z założenia jest uproszczonym modelem sieci naturalnej, jaką jest m.in. ludzki mózg. Przypomnijmy sobie trochę elementarnej biologii. Mózg składa się z komórek nerwowych zwanych neuronami. Pojedynczy neuron to komórka, która jest zbudowana z ciała komórkowego, dendrytów (wejść) i aksonu (wyjścia). Działanie neuronu jest proste – jeśli suma napięć na dendrytach przekroczy pewien próg, to generowany jest sygnał wyjściowy, który przez synapsy (połączenia aksonu z dendrytami) przekazywany jest do następnych komórek (oczywiście jest to bardzo uproszczony model). Akson jednego neuronu łączy się z dendrytami innych i tym sposobem neurony są połączone w szarą sieć. Kora mózgowa człowieka składa się z około 10^{10} (dziesięciu miliardów) komórek nerwowych, między którymi jest mniej więcej 10^{15} (miliona miliardów) połączeń. Czas reakcji pojedynczego neuronu to około 3 milisekund, a czas reakcji mózgu na bodziec to 300 milisekund (można z tego wyciągnąć wniosek, że informacja przechodzi w mózgu przez 100 warstw neuronów). Zasadniczą różnicą między mózgiem a współczesnym komputerem jest to, że „obliczenia" w mózgu prowadzone są równolegle (wiele neuronów „obrabia" tę samą informację jednocześnie), a programy komputerowe są mniej lub bardziej sekwencyjne (robią wszystko po kolei).

Sztuczne sieci neuronowe składają się z (programowych lub sprzętowych) modeli pojedynczych neuronów, w jakiś sposób połączonych ze sobą. Taka modelowa komórka nerwowa ma ileś „wejść" i jedno „wyjście". W mózgu niektóre synapsy są mocniejsze, inne słabsze. Modeluje się to, każdemu wejściu przypisując wagę przez jaką będzie przemnażane „napięcie", które się na nim pojawi. Poza wejściami od innych neuronów, model ma także jedno wejście służące do przesuwania progu aktywacji – czyli wzmacnianiu lub osłabianiu działania neuronu. To „sztuczne" wejście zawsze ma „napięcie" równe jeden, więc rzeczywisty wkład w działanie neuronu zależy od wagi, jaką jemu przypiszemy (oznaczmy tę wagę przez w_0).

Oznaczmy:

- $\overrightarrow{x} = (x_1, x_2, \cdots, x_n)$ – wektor wejść,
- $\overrightarrow{w} = (w_0, w_1, w_2, \cdots, w_n)$ – wektor wag.

Modelowy neuron oblicza najpierw ważoną sumę wejść I ($I = w_0 + \overrightarrow{x} \circ \overrightarrow{w} = w_0 + \sum_{i=1}^{n} x_i w_i$), a następnie wartość jakiejś prostej funkcji (ustalonej dla danego neuronu), zwanej funkcją aktywacji, przy argumencie równym I. Najprostszym przykładem takiej funkcji jest tzw. funkcja progowa – wzorowana na rzeczywistej komórce nerwowej:

$$f(I) = \begin{cases} 1 & \text{gdy} \quad I > 0 \\ -1 & \text{gdy} \quad I \leqslant 0 \end{cases}$$

Funkcja ta odpowiada generowaniu sygnału, gdy tylko suma napięć na dendrytach przekroczy pewną wartość. Popatrzmy na to, co robi taka funkcja w przypadku, gdy $\overrightarrow{x} = (x_1, x_2)$, czyli gdy dane zawierają się w płaszczyźnie. Mamy wtedy $I = w_0 + w_1 x_1 + w_2 x_2$. Po chwili zastanowienia widzimy, że zbiór tych x_1 i x_2, które spełniają równanie $I = 0$, tworzy prostą. Po jednej stronie tej prostej znajdują się takie (x_1, x_2), że $I > 0$, a po drugiej takie, że $I < 0$. Czyli nasza funkcja progowa, dla ustalonego $\overrightarrow{w}$ dzieli płaszczyznę na dwie „połowy".

Łatwo wymyślić takie wagi, żeby nasz neuron działał jak logiczna koniunkcja lub alternatywa (przyjmujemy, że prawda = 1, a fałsz = −1):

$$\mathrm{AND}(x_1, x_2) = f(x_1 + x_2 - 1, 5),$$
$$\mathrm{OR}(x_1, x_2) = f(x_1 + x_2 + 0, 5).$$

Jeśli weźmiemy dwa neurony, z których każdy wybiera jakąś połowę płaszczyzny, to jeśli wyjścia tych neuronów podłączymy do neuronu AND (OR), otrzymamy przecięcie (sumę) tych półpłaszczyzn. Nic nie stoi na przeszkodzie, żeby połączyć wyjście tego przecięcia z czymś innym, kolejnym AND-em lub OR-em i otrzymać kolejne przecięcie. W takim razie, możemy zbudować sieć, która będzie dawała odpowiedź 1 na dowolnym, wybranym przez nas, „porządnym" (cokolwiek słowo porządny by w tym miejscu znaczyło) podzbiorze płaszczyzny, a −1 na całej reszcie.

Funkcja progowa jest łatwa do zrozumienia i napisania, ma jednak jedną wadę – jest nieciągła, co, jak się okaże później, będzie nam przeszkadzać. Z tego powodu, w praktycznych zastosowaniach zastępuje się funkcję progową jakimiś gładkimi przybliżeniami, takimi jak funkcja sigmoidalna

$$\sigma(x) = \frac{1}{1 + e^{-x}},$$

czy tangens hiperboliczny

$$\mathrm{tgh}(x) = \frac{e^x - e^{-1}}{e^x + e^{-x}}.$$

Główna różnica między nimi jest taka, że $\sigma(x)$ przyjmuje wartości od 0 do 1, a $\mathrm{tgh}(x)$ od −1 do 1. Obie te funkcję od funkcji progowej różnią się tym, że nie dzielą one płaszczyzny „dokładnie" jedną kreską, tylko „mniej więcej", a kreska jest rozmyta.

2. Budowa sieci

Wiemy już jak wyglądają i jak działają pojedyncze neurony, jednak od budowy neuronu do budowy całej, działającej sieci jeszcze daleka droga. Zanim nauczymy naszą sieć wykonywać konkretne zadanie, musimy ustalić jej strukturę – ile będzie w niej neuronów, jak będą połączone i jaka będzie topologia sieci.

Na początek musimy uświadomić sobie jedną rzecz. Prawda generalna, mówiąca, że warto dokładnie wiedzieć czego się chce, w wypadku sieci neuronowych jest szczególnie ważna. Projektując sieć, musimy dokładnie określić, na jakie pytania sieć ma odpowiadać. Pojedynczy neuron potrafi odpowiedzieć na pytanie tak/nie (w wypadku gładkiego przybliżenia funkcji progowej – „jak bardzo tak"), więc jeśli chcemy, żeby sieć mówiła nam np. „jaka cyfra jest tutaj napisana", musimy stworzyć wyjściową warstwę sieci (czyli tę, z której będziemy odczytywać odpowiedź), składającą się z jednego neuronu na każdą możliwą odpowiedź: „jak bardzo ta cyfra jest jedynką", „jak bardzo ta cyfra jest dwójką" itd.

Drugim, trudniejszym pytaniem jest „jakie informacje będę dawał sieci". W niektórych przypadkach, jak np. rozpoznawanie pisma, odpowiedź jest stosunkowo łatwa – w tym przypadku będzie to trochę przerobiona bitmapa z obrazkiem, który chcemy rozpoznawać. Niestety, nie zawsze sytuacja jest tak prosta, a na niektóre pytania, takie jak „jakie informacje powinna mieć sieć grająca na giełdzie", do tej pory nikt nie zna odpowiedzi (a przynajmniej nikt się do tego nie przyznaje). Wszystkie dane muszą być w postaci zrozumiałej dla komputera, czyli liczbowej. Jak już uporamy się z wyborem rodzaju informacji i przekształceniem ich na język matematyki (czyli zapiszemy je w postaci wektora $(x_1, x_2, \cdots, x_n)$), będziemy wiedzieć, ile „wejść" będą miały neurony z najwyższej warstwy budowanej sieci.

W ogólności, sieć może mieć postać dowolnego grafu skierowanego. Jeśli taki graf zawiera cykle, sieć nazywamy rekurencyjną, jeśli jest acykliczny to sieć nazywamy jednokierunkową (feedforward network). My zajmiemy się sieciami jednokierunkowymi. Acykliczne grafy skierowane nazywamy dagami (Directed Acyclic Graph), ale z przyzwyczajenia będziemy stosować nazwy wywodzące się z normalnych (ale informatycznych) drzew. Dla ustalenia uwagi przyjmijmy, że „korzeń" naszej sieci będzie odpowiadał „wejściu" i będzie „na górze", a liście będą

„wyjściami" i znajdować się będą „na dole". Wiemy już, że będziemy budować sieć jednokierunkową, wiemy, ile nasze drzewo będzie miało liści i ile wyjść będzie miał korzeń. Trzeba jeszcze ustalić, co będzie między górą a dołem. Będą tam jakieś neurony, które możemy pogrupować w „warstwy" w zależności od odległości od korzenia – wszystkie warstwy ponad liśćmi, a pod korzeniem, nazywamy warstwami ukrytymi. Ogólna odpowiedź na pytanie „ile warstw ukrytych i jakiej wielkości będzie zawierała nasza sieć" jest nieznana i w każdym przypadku może być inna. Znajdowanie najlepszej budowy drzewa polega na przeprowadzeniu wielu eksperymentów i wyciągnięciu wniosków. Dąży się do tego, żeby sieć dawała jak najbliższe prawdy odpowiedzi, ale jednocześnie była na tyle mała, by zdążyć się „nauczyć" w skończonym czasie.

Dla przykładu mogę powiedzieć, że do podstawowego rozpoznawania ręcznie pisanych cyfr wystarczy jedna warstwa ukryta (taka sieć daje błąd rzędu 2%). Podobno nie są znane zastosowania, do których potrzebne są sieci o więcej niż trzech warstwach ukrytych.

3. Uczenie sieci

Zastanówmy się, czym może być „uczenie" sieci. Wiemy już, że działanie sieci to nic innego jak znajdowanie pewnych podzbiorów przestrzeni danych i przypisywanie im jakichś odpowiedzi. W miarę jasny jest fakt, że by sieć w ogóle miała szansę przyzwoicie odpowiadać na pytania, dane dla których prawidłowa jest jakaś odpowiedź muszą stanowić na tyle porządny zbiór, żeby dało się go odciąć od innych skończoną liczbą „prostych" (ściśle – hiperpłaszczyzn). Jeśli spojrzymy na sieć neuronową geometrycznie, to proces uczenia się będzie po prostu przesuwaniem tych „prostych" tak, aby jak najlepiej wycinały szukany zbiór. Pojawiło się magiczne słówko: „najlepiej" – właśnie w tym miejscu przychodzi z pomocą matematyka.

Zanim wyjaśnimy dokładnie, jaki jest problem ze słowem „najlepiej", popatrzmy na coś innego – w jaki sposób my, ludzie, uczymy się. Po pierwsze zdarza się, że mamy jakiegoś nauczyciela, kogoś kto powie nam, co zrobiliśmy dobrze, a co zrobiliśmy źle. Sieć też można uczyć w ten sposób (nazwany uczeniem z nauczycielem – supervised learning). Polega to na tym, że dajemy sieci zbiór danych treningowych z odpowiedziami. Sieć odpowiada na pytania po kolei, porównuje odpowiedź, do której sama doszła, z prawidłową i przesuwa „proste" (czyli zmienia wagi $\overrightarrow{w}$) tak, żeby w przyszłości zmniejszyć błąd, jaki popełniła. Taki sposób uczenia jest stosowany np. przy sieciach stosowanych w OCR-ach (Optical Character Recognition – systemach rozpoznających pismo).

Czasami jednak nie mamy do dyspozycji nauczyciela, ale wiemy mniej więcej co chcemy osiągnąć (np. chcemy dobrze biegać). W takim przypadku sami jesteśmy w stanie stwierdzić, jak bardzo źle nam, na razie, idzie. Tak samo jest z sieciami, one też czasami potrafią stwierdzić jak dobrze rozwiązały problem, wiedząc tylko, jakie dane dostały i jaki wyszedł im wynik (taki sposób uczenia nazywamy uczeniem bez nauczyciela – unsupervised learning). Po stwierdzeniu, co mogłaby robić lepiej, także poprawia się wagi w neuronach. Tak uczy się np. sieci używane w kompresji danych.

Ostatnią metodą uczenia jest opracowywanie wzorców zachowań, kiedy prawidłowy wynik nie jest określony, ale można wnioskować jakie strategie prowadzą do najlepszych efektów. Człowiek uczy się prawidłowego zachowania metodą prób i błędów – jeśli starsza pani będzie okładała parasolką młodzieńca za to, że nie ustąpił jej miejsca, to będzie on wiedział, że jeśli nie chce zostać pobity, to musi ustępować miejsca starszym paniom. W wypadku sieci jest podobnie. Sieć może znajdować się w jakimś stanie (czyli np. siedzieć w tramwaju) i musi zdecydować, jaką z możliwych akcji ma podjąć (ustąpić miejsca lub nie). Po podjęciu decyzji przechodzi do następnego stanu (siedzi dalej, albo stoi) i dostaje „nagrodę" (lub „karę") za to, co zrobiła. Uczenie polega na znalezieniu najlepszej strategii zachowania. Oczywiście, znowu sieć poprawia swoje wagi tak, żeby maksymalizować nagrodę. W terminologii sieci neuronowych, taką metodę nazywamy uczeniem ze wspomaganiem – reinforcement learning. Ta metoda jest najlepsza dla sieci mających uczyć się „na bieżąco", np. sieci grające w różne gry.

Widać pewną cechę wspólną tych wszystkich metod nauki. Jest nią chęć minimalizacji (maksymalizacji) pewnej funkcji błędu (zysku). Statystyka zna wiele różnych miar błędu, czy „ryzyka" i każda z nich może być z powodzeniem stosowana w różnych przypadkach. Każda mierzy jakość w inny sposób – nie ma jednej, uniwersalnej definicji jakości. Najpopularniejszą miarą jest błąd średniokwadratowy, który – ogólnie rzecz biorąc – jest odpowiednikiem wariancji. W uproszczeniu, jest to miara rozrzucenia odpowiedzi wokół prawidłowej – im mniejsza – tym odpowiedzi są bardziej skupione wokół dobrej, czyli bardzo często podajemy prawie dobrą odpowiedź, a im większa – tym większą mamy szansę na duży błąd. Oznaczmy:

- t_i - prawidłowa odpowiedź i-tego neuronu z danej warstwy,
- z_i - ważona suma wejść i-tego neuronu z danej warstwy,
- f - funkcja aktywacji neuronu.

Przy takich oznaczeniach, błąd średniokwadratowy to $E = \sum_{i=1}^{n}(t_i - f(z_i))^2$. Jeśli przyjrzymy się temu wzorowi i przypomnimy sobie, że f i $\overrightarrow{x}$ są określone (dla danego przykładu), zauważymy, że E jest funkcją zależną wyłącznie od $\overrightarrow{w}$. Czyli jesteśmy w stanie stwierdzić jak, przy tych konkretnych danych, błąd zależy od wag. Sytuacja jest taka, że nasza sieć „stoi" w pewnym miejscu wykresu zależności błędu od wag – gdzieś w górach. Jej celem jest zejście do najgłębszej kotliny na całej Ziemi (zminimalizowanie funkcji błędu). Pierwszą przymiarką do osiągnięcia tego celu jest po prostu schodzenie w dół, póki jest to możliwe. Oczywiście, żeby zejść jak najszybciej na dół, musimy poruszać się w kierunku, w którym jest najstromiej. W matematyce ten kierunek jest przeciwny do gradientu, który pokazuje, w którą stronę jest najbardziej do góry. Żeby móc obliczyć ów gradient, potrzebujemy, żeby góry były stosunkowo gładkie i nie mały przepaści (dlatego też zamiast brać zwykłą funkcję progową bierzemy jakieś porządne przybliżenia). Najważniejsze twierdzenie sztucznej inteligencji mówi, że nieważne ilowymiarowe będą góry (mogą mieć nawet nieskończoną ilość wymiarów): jeśli będziemy zawsze szli najbardziej w dół jak to możliwe, to w końcu dojdziemy do minimum lokalnego (dna jakiejś kotliny).

Niestety, jak już stoimy na dnie, to nie jesteśmy w stanie stwierdzić, czy jest to ta „najgłębsza". Żeby zwiększyć swoje szanse na trafienie tam, gdzie chcemy, stosuje się różne strategie. Najprostszą jest wielokrotny start: na początku losujemy miejsce, z którego zaczynamy, schodzimy tak głęboko, jak się da z tego miejsca i zapamiętujemy głębokość. Jak już zrobimy to wystarczająco dużo razy, to wybieramy najgłębsze miejsce, jakie znaleźliśmy.

Innym problemem jest ustalenie wielkości kroków, jakie będziemy robić (czyli ustalenie stałej uczenia). Jak schodzimy sobie po wykresie funkcji, to nie możemy co chwilę stawać i od nowa sprawdzać, w którą stronę jest „w dół". Musimy ustalić sobie jakąś długość kroku. To jest kolejna rzecz, którą trzeba ustalić eksperymentalnie. Jeśli kroki będą za małe, to możemy schodzić bardzo, bardzo długo, za to jeśli będą za duże, to może nam się zdarzyć, że „przeskoczymy" kilka szczytów jednym susem. Dosyć popularną strategią jest zmniejszanie długości kroku w trakcie zbliżania się do dna. Zaczynamy od sporych skoków, żeby w miarę szybko znaleźć się w okolicach dna, a później zwalniamy, żeby precyzyjniej trafić najniżej położone miejsce.

Są różne sposoby przyspieszenia „schodzenia" w głąb doliny. Jedną z nich jest stosowanie „pędu" (momentum) – pomysł polega na zastąpieniu człowieka w górach przez piłkę. Piłka, gdy się stacza, to nie robi tego po gradiencie – ma jakiś swój pęd, który trochę zakrzywia tor jej ruchu. Jak wiemy kuleczka zazwyczaj się stacza na sam dół, a tylko w wyjątkowo złośliwych przypadkach może „przeskoczyć" dołek, więc i tutaj dodanie „pędu" nie powinno psuć w większości przypadków, mimo że teoretycznie jest to możliwe. Praktyka pokazuje, że dodanie „pędu" znacznie przyspiesza uczenie się sieci praktycznie go nie zakłócając.

Przedstawiliśmy tutaj tylko ogólną ideę stojąca za uczeniem i budową sieci neuronowych – konkretnych algorytmów, o ich implementacjach nie mówiąc, są setki. Niektóre z algorytmów są stosowane częściej, inne rzadziej, ale nie ma na razie żadnego, o którym można by śmiało powiedzieć, że we wszystkich przypadkach jest najlepszy.