Sztuczna Inteligencja

Soma Dutta

Wydział Matematyki i Informatyki, UWM w Olsztynie
soma.dutta@matman.uwm.edu.pl

Wykład - 11: Uczenie się na przykładach

Semestr letni 2022

Uczenie

- Uczenie się jest procesem ukierunkowanym na osiąganie celów bazując na dostępie do danych (wiedzy, obserwacji) fragmentarycznych. Powinna być zapewniona możliwość doskonalenia się uczenia.
- Agent może się nauczyć różnych aspektów. Niektóre z nich są następujące.
  - Jaką wcześniejszą wiedzę agent już ma?
  - Jak dane są reprezentowane?
  - Jak uczyć się na podstawie opinii?
- Rozmawialiśmy już o reprezentowaniu wiedzy agenta przy użyciu języka rachunku zdań lub pierwszego rzędu.
- Omówimy teraz różne sposoby uczenia się na podstawie faktoryzowanej reprezentacji (factored representation) danych gdzie dane wejściowe są reprezentowane za pomocą wektorów wartości atrybutów, a dane wyjściowe mogą być liczbami rzeczywistymi (odpowiadającymi wartościom pewnej funkcji ciągłej) lub mają postać dyskretnej wartości.

Unsupervised learning

- Wśród różnych sposobów uczenia się jest prawdopodobnie poprawnej uczenie się funkcji ogólnej lub reguł na podstawie zadanych par wejścia-wyjścia. Nazywa się to uczeniem indukcyjnym.
- Istnieją trzy rodzaje informacji zwrotnych, które określają trzy główne typy uczenia się.
  - Uczenie się bez nadzoru: W takim przypadku agent uczy się w postaci skupisk obiektów, nawet jeśli nie jest dostarczana jawna informacja o tych skupiskach. Najczęstszym zadaniem uczenia się bez nadzoru jest grupowanie (clustering), co oznacza wykrywanie skupisk zbiorów przykładów.
  - Przykład: Taksówkarz może stopniowo opracować koncepcję 'dobrych dni dla jazdy' i 'złych dni dla jazdy', nie mając podanych przykładów etykietowanych przez nauczyciela jako dobre lub złe dni dla jazdy.

Reinforcement learning (Uczenie się ze wzmocnieniem)

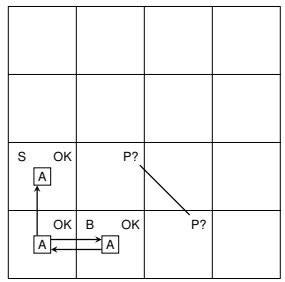
- Uczenie się ze wzmocnieniem: W takim przypadku agent uczy się na podstawie serii wzmocnień - w postaci nagród lub kar.
- Przykład: Dwa punkty za zwycięstwo na końcu gry w szachy mówi agentowi, że zrobił coś dobrze. Agent musi zdecydować, które z ruchów wcześniej wykonanych w grze były najbardziej odpowiedzialne za to.

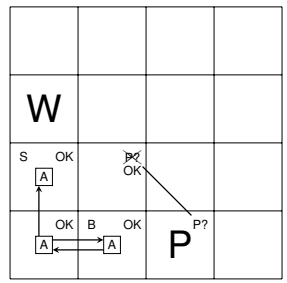
Supervised learning (Uczenie nadzorowane)

- Uczenie nadzorowane: W tym przypadku agent otrzymuje niektóre przykłady par wejścia-wyjścia i uczy się funkcji, która odwzorowuje wejścia na wyjścia (również dla nieznanych dotąd wejść).
- Przykład: Rozważmy na przykład szkolenie kandydatów na taksówkarzy. Z każdym razem gdy instruktor woła 'Hamulec' kandydat może nauczyć się warunku - zasady działania określającej warunek kiedy ma nastąpić hamowanie.
- W praktyce rozróżnienie to nie zawsze jest tak wyraźne. W uczeniu semi-nadzorowanym danych jest niewiele przykładów etykietowanych (oznakowanych decyzją) i należy oznakować (decyzją) duży zbiór danych obiektów, które nie są jeszcze oznakowane. Należy tego dokonać przy założeniu, że zadane etykiety mogą nie być absolutnie (zawsze) prawdziwe (ponieważ dane mogą zawierać zaszumione i/lub nieprecyzyjne dane).

Przykłady różnych rodzajów uczenia się

- W systemach decyzyjnych widzieliśmy, że zaczynamy od par wejście-wyjście, gdzie wektor wartości dla atrybutów warunkowych jest traktowany jako dane wejściowe, a przypisana decyzja jest odpowiednim wyjściem. Naszym celem było znalezienie reguł spełniających wszystkie przypadki par wejście-wyjście. Dotyczy to uczenia się nadzorowanego.
- W świecie Wumpusa agent uczy się poprzez dedukcyjne uczenie się, a także uczenie się ze wzmocnieniem.





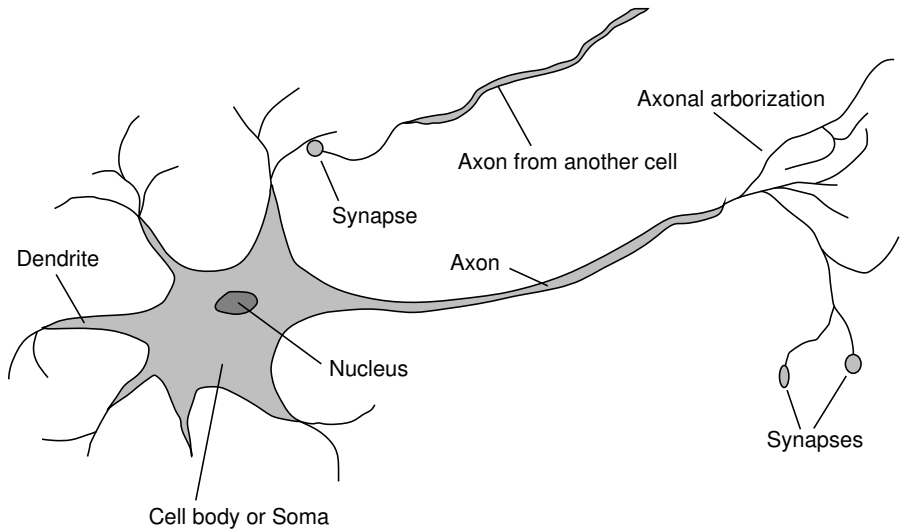
Uczenie nadzorowane

- Zadanie: Mając zadany zbiór treningowy N przykładowych par wejście-wyjście $(x_1, y_1), \ldots, (x_N, y_N)$, gdzie każdy został wygenerowany przez nieznaną funkcję $y = f(x)$, należy odkryć funkcję h, która jest bliska do prawdziwej funkcji f.
- Tutaj x i y mogą mieć dowolną wartość; nie muszą to być liczby. Funkcja h jest hipotezą.
- Uczenie się to przeszukiwanie przestrzeni możliwych hipotez w celu znalezienia takiej hipotezy, która dobrze się sprawdzi, nawet na nowych przykładach poza zbiorem treningowym.

Sieci neuronowe

- Pomysły techniczne, które omówiliśmy do tej pory, okazują się przydatne przy budowaniu modeli matematycznych aktywności mózgu; i odwrotnie, rozważania o budowie mózgu mogą pomóc w rozszerzeniu zakresu pomysłów technicznych.

  Naśladowanie mózgu działającego jako sieć komórek neuronowych.



  Sygnały to zaszumione “bodźce trenujące” poziom potencjału elektrycznego komórek.

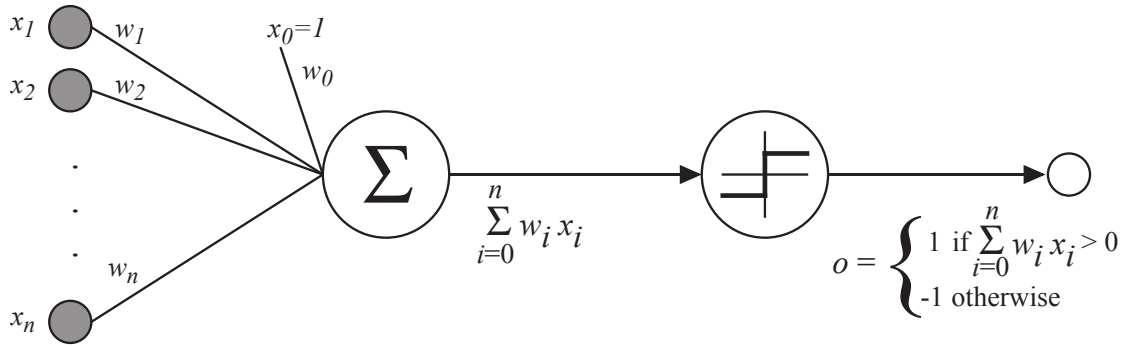
Sieci neuronowe: stuczne i naturalne

	Komputer	M.zg
Jednostki obliczeniowe	1 CPU 10^8 bramek logicznych	10^{11} neuronów > 20 typów
Jednostki pamięciowe	10^{10} bitów RAM 10^{11} bitów dysku	10^{11} neuronów 10^{14} synaps
Czas cyklu	1 ns (10^{-9} sek.)	1–10 ms (10^{-3} sek.)
Szerokość pasma	10^{10} bitów/sek	10^{14} bitów/sek
Zmiany stanów/sek	10^9	10^{14}

Równoległość daje wciąż umysłowi ludzkiemu ogromną przewagę nad komputerem pomimo dużo wolniejszego czasu przetwarzania informacji.

Perceptron

$\vec{x} = (x_1, x_2, \ldots, x_n)$ — wektor wejściowy (obiekt danych)

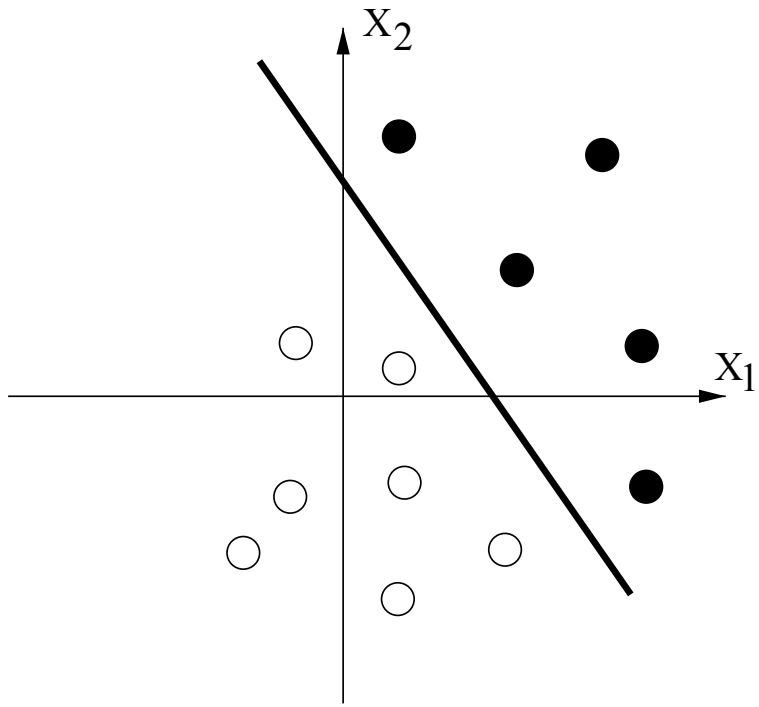


$\vec{w} = (w_0, w_1, \ldots, w_n)$ — wektor wag perceptronu
w_0 – waga przesunięcia (“przesuwa” próg funkcji aktywacji)
σ – progowa (skokowa) funkcja aktywacji perceptronu
$o(\vec{x})$ – wartość wyjścia perceptronu dla wektora $\vec{x}$

$$o(\vec{x}) = \sigma(\vec{w} \cdot \vec{x}) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \ldots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

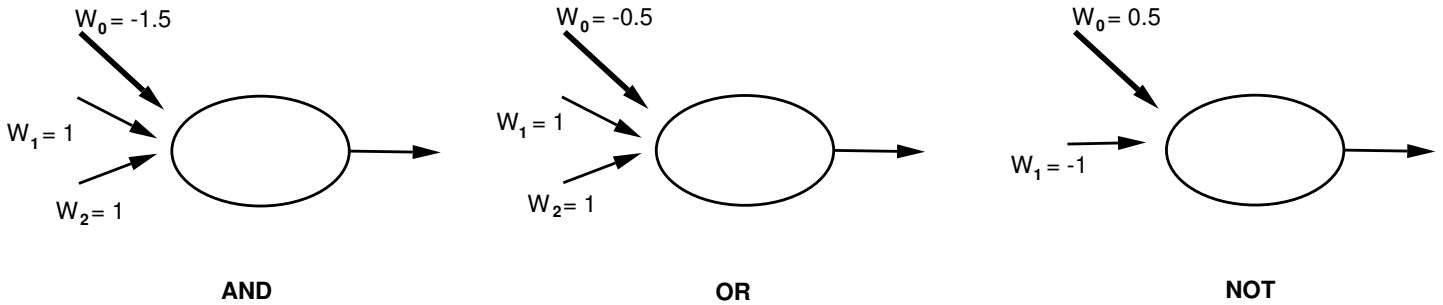
Perceptron: wyrażalność

Perceptron reprezentuje liniowe cięcie w przestrzeni wejść

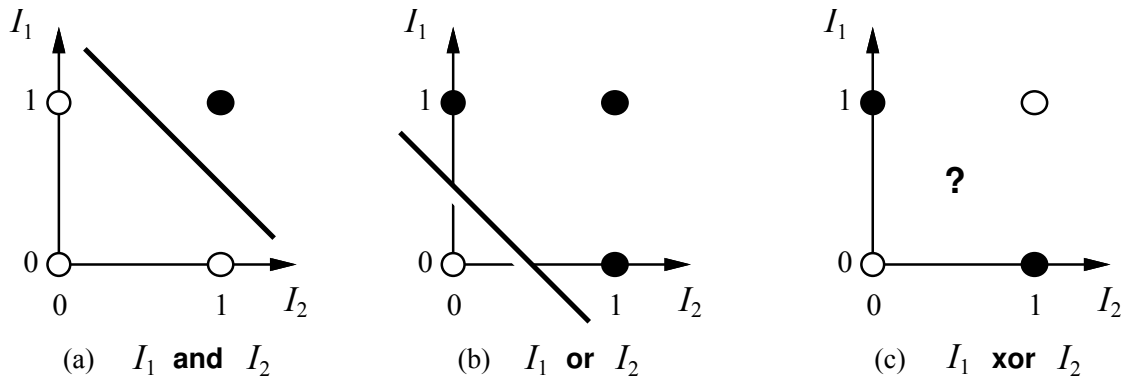


Perceptron: wyrażalność

Można wyrazić funkcje logiczne AND, OR, NOT



ale nie da się dobrać wag do funkcji XOR



- Funkcja AND:

  $w_0 + w_1 + w_2 > 0$ bo $x_1 = x_2 = 1$
  $w_0 + w_1 \leq 0$ bo $x_1 = 1, x_2 = 0$
  $w_0 + w_2 \leq 0$ bo $x_1 = 0, x_2 = 1$
  $w_0 \leq 0$ bo $x_1 = x_2 = 0$

- Funkcja OR:

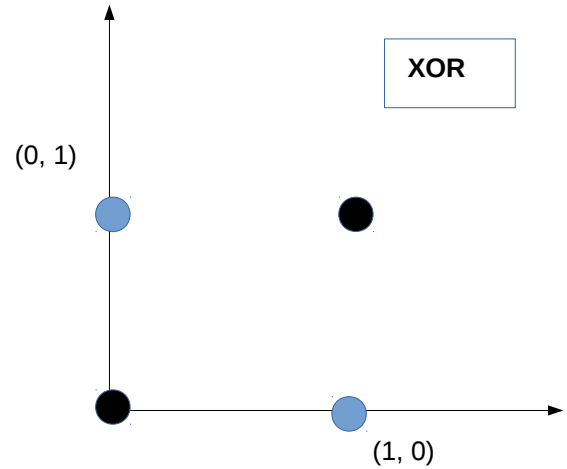
  $w_0 + w_1 + w_2 > 0$ bo $x_1 = x_2 = 1$
  $w_0 + w_1 > 0$ bo $x_1 = 1, x_2 = 0$
  $w_0 + w_2 > 0$ bo $x_1 = 0, x_2 = 1$
  $w_0 \leq 0$ bo $x_1 = x_2 = 0$

- Funkcja NOT:

  $w_0 > 0$ bo $x_1 = 0$
  $w_0 + w_1 \leq 0$ bo $x_1 = 1$

Ograniczenie perceptronu

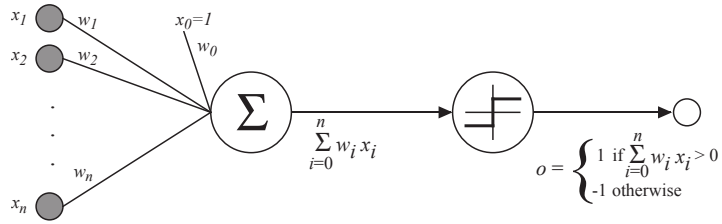
- Możemy postrzegać perceptron jako reprezentujący hiperpłaszczyznową powierzchnię decyzyjną w n-wymiarowej przestrzeni instancji (tj. punktów). Wyjścia Perceptronu 1 dla wystąpienia obiektu po pozytywnej stronie hiperpłaszczyzny i -1 dla wystąpienia obiektu po drugiej stronie hiperpłaszczyzny.
- Nie wszystkie zbiory pozytywnych i negatywnych przykładów można oddzielić (separować) hiperpłaszczyzną. Te, które można rozdzielić, są nazywane liniowo separowalnymi zbiorami przykładów (linearly separable examples).
- Perceptrony mogą reprezentować wszystkie prymitywne funkcje boolowskie AND, OR, NAND ($\neg$ AND) i NOR ($\neg$ OR). Niestety niektórych funkcji boolowskich nie można reprezentować za pomocą pojedynczego perceptronu. Na przykład taką funkcją jest XOR której wartość wynosi 1 wtedy i tylko wtedy, gdy $x_1 \neq x_2$.



- Zdolność perceptronów do reprezentowania AND, OR, NAND i NOR jest ważna, ponieważ każda funkcja boolowska może być reprezentowana przez pewną sieć połączonych ze sobą jednostek bazującą na tych funkcjach prymitywnych.
- W rzeczywistości każda funkcja boolowska może być reprezentowana przez pewną sieć perceptronów o głębokości wyznaczonej przez zaledwie dwa poziomy, w których dane wejściowe są podawane do wielu jednostek pierwszego poziomu, a dane wyjściowe z tych jednostek są następnie wprowadzane do drugiej, ostatniej warstwy. Jest to możliwe, ponieważ każda formuła boolowska może być reprezentowana w postaci koniunkcji klauzul, z których każda jest alternatywą literałów.

Uczenie się perceptronu

$\vec{x} = (x_1, x_2, \ldots, x_n)$ — wektor wejściowy (obiekt danych)



$\vec{w} = (w_0, w_1, \ldots, w_n)$ — wektor wag perceptronu
w_0 – waga przesunięcia (“przesuwa” próg funkcji aktywacji)
σ – progowa (skokowa) funkcja aktywacji perceptronu
$o(\vec{x})$ – wartość wyjścia perceptronu dla wektora $\vec{x}$

$$o(\vec{x}) = \sigma(\vec{w} \cdot \vec{x}) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \ldots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Przykładowy algorytm oparty na idei iteracyjnego poprawiania, celem jest optymalizacja (minimalizacja błędu) modelu.

function PERCEPTRON-LEARN(*perceptron*, *examples*, α) **returns** a perceptron
  inputs: *examples* - a set of examples, each with input $\vec{x}$ and output $y(\vec{x})$
    perceptron - a perceptron with weights $\vec{w} = (w_0, \ldots, w_n)$
    α - the learning rate
  repeat
    for each $\vec{x} = (x_1, \ldots, x_n)$ in *examples* **do**
      $\Delta\vec{w} \leftarrow \alpha\,(y(\vec{x}) - \vec{w} \cdot \vec{x})\,\vec{x}$
      $\vec{w} \leftarrow \vec{w} + \Delta\vec{w}$
    end for
  until some stopping criterion is satisfied
  return *perceptron*
end function

Uczenie się perceptronu

- Skoncentrujmy się na tym, jak wyuczyć perceptron dla danej funkcji, czyli jak nauczyć się wag dla jednego perceptronu. Tutaj, dokładniej problemem uczenia się jest określenie wektora wag, który powoduje że perceptron generuje poprawne wyjścia $+1/-1$ dla każdego z podanych przykładów treningowych.
- Jednym ze sposobów nauczenia się akceptowalnego wektora wag jest rozpoczęcie od losowych wag, następnie iteracyjnie stosuje się perceptron do każdego przykładu treningowego, modyfikując, wprowadzane wagi perceptronu za każdym razem, gdy błędnie klasyfikuje on przykład.
- Ten proces jest powtarzany iteracyjnie dla przykładów treningowych tyle razy, ile potrzeba aby perceptron poprawnie sklasyfikował wszystkie przykłady treningowe.
- Wagi są modyfikowane w każdym kroku zgodnie z regułą treningową perceptronu, która zmienia wagę w_i związaną z wejściem x_i zgodnie z regułą $w_i \leftarrow w_i + \Delta w_i$ gdzie $\Delta w_i = \alpha(y(\overrightarrow{x}) - O(\overrightarrow{x})).x_i = \alpha(y(\overrightarrow{x}) - \sigma(\overrightarrow{w}.\overrightarrow{x})).x_i$ i α nazywa się współczynnikiem uczenia się.

Zbieżność: przykład

- Przykład wyjaśniający, w jaki sposób iteracja zbiega do poprawnego rozwiązania.
  - Załóżmy, że przykład treningowy jest poprawnie sklasyfikowane już przez perceptron. W tej sytuacji $(y(\overrightarrow{x}) - \sigma(\overrightarrow{w}.\overrightarrow{x})) = 0$. Więc $\Delta w_i = 0$ i żadne wagi nie są aktualizowane.
  - Załóżmy, że perceptron wyprowadza wartość -1, gdy docelowy wynik wynosi +1. Dlatego wagi muszą zostać zmienione, aby zwiększyć wartość $\overrightarrow{w}.\overrightarrow{x}$. Na przykład jeśli $x_i > 0$, zwiększenie w_i zbliży perceptron do prawidłowej klasyfikacji przykładu. Na przykład, jeśli $x_i = 0{,}8$, $\alpha = 0{,}1$, $y(\overrightarrow{x}) = 1$, a $\sigma(\overrightarrow{w}.\overrightarrow{x}) =$ -1, wtedy aktualizacja wagi będzie $\Delta w_i = \alpha.(y(\overrightarrow{x} - \sigma(\overrightarrow{w}.\overrightarrow{x})).x_i = 0.16\ (> 0)$.
  - Z drugiej strony, jeśli $y(\overrightarrow{x}) =$ -1 i $\sigma(\overrightarrow{w}.\overrightarrow{x}) = 1$, to wagi dla $x_i > 0$ będą raczej zmniejszać się niż zwiększać.

Uczenie się perceptronu dla funkcji AND

- $\overrightarrow{x} = (x_1, x_2)$ to wektor wejściowy, x_0 to zmienna fikcyjna, której wartości jest zawsze ustawiona jako $x_0 = 1$, i $\overrightarrow{w} = (w_0, w_1, w_2)$ to wektor wag.
- Musimy wybrać w_0, w_1, w_2 w taki sposób, aby gdy $x_1 = x_2 = 1$ wyjście perceptronu wynosi 1, a we wszystkich innych przypadkach wyjście to 0.
- Próbki wejściowe-wyjściowe do treningu to: ((0, 0), 0), ((1, 0), 0), ((0, 1), 0) i ((1, 1), 1).
- Załóżmy, że α jest ustawiona na 0.1.
- Proces może rozpocząć od dowolnych losowych wartości dla w_0, w_1, w_2. (Np, może być że wybrane jest $w_0 = w_1 = w_2 = 1$)
- Wyjście perceptronu dla pierwszego przykładu: $w_0.1 + w_1.0 + w_2.0 = 1$, i oczekiwany wynik to 0. Tak więc zgodnie z zasadą uczenia się wagi musimy zmieniać każdą wagę tak, że
  $w_i \leftarrow w_i + \alpha(y(\overrightarrow{x}) - \overrightarrow{w}.\overrightarrow{x}).x_i$.
  $w_0 \leftarrow 1 + 0.1(0 - 1).1$, a dla i=1, 2 mamy $w_i \leftarrow 1 + 0.01(0 - 1).0$
- Więc w następnej iteracji mamy $w_0 = 0.9$, $w_1 = w_2 = 1$.

Uczenie się z regułą delta

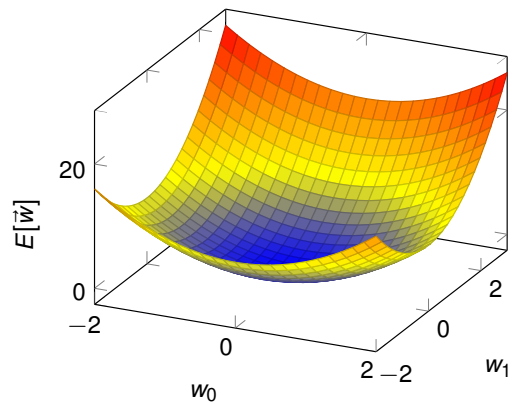
- Chociaż reguła perceptronu znajduje poprawny wektor wag podczas treningu gdy przykłady można rozdzielić liniowo, może jednak nie zapewnić zbieżności, jeśli przykłady nie są liniowo separowalne.
- Druga reguła treningowa, zwana regułą delta, jest przeznaczona aby przezwyciężyć tę trudność. Kluczową ideą tej reguły delta jest użycie spadku gradientu wektora wag do przeszukania przestrzeni hipotez odpowiadającym możliwym wektorom wag, aby znaleźć wagi, które najlepiej pasują do przykładów treningowych.
- Regułę treningową delta najlepiej zrozumieć, biorąc pod uwagę zadanie treningu perceptronu bez progu; jest to jednostka liniowa (linear unit), dająca wynik jako $o(\overrightarrow{x}) = \overrightarrow{w}.\overrightarrow{x}$.
- Aby wyprowadzić regułę uczenia się wag dla jednostek liniowych, zacznijmy od określenie miary błędu hipotezy (wektor wag), względem przykładów treningowych.

Błąd sredniokwadratowy

- Błąd sredniokwadratowy dla zbioru treningowego U: $E(\vec{w}) = \frac{1}{2}\Sigma_{\vec{x}\in U}(y(\vec{x}) - \vec{w}.\vec{x})^2$
- $E(\vec{w})$ to po prostu połowa kwadratowej różnicy między docelowym wyjściem $y(\vec{x})$ a wyjściem jednostki liniowej $\vec{w}.\vec{x}$, zsumowana po wszystkich przykładach treningowych.
- Tak więc dla każdego wektora w przestrzeni hipotez możliwych wektorów wag istnieje odpowiadająca wartość E, i poszukujemy hipotezy wyznaczającej minimalny błąd na zbiorze treningowym.

Wyszukiwanie z wykorzystaniem spadku gradientu

- Możemy obliczyć kierunek najbardziej stromego zejścia wzdłuż powierzchni błędu, obliczając pochodną E względem każdej składowej wektora $\overrightarrow{w}$.



Gradient błędu średniokwadratowego

$$\nabla E[\vec{w}] = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$$

$\Longrightarrow$ wskazuje kierunek, w którym błąd $E[\vec{w}]$ rośnie

- Widzimy, że $\bigtriangledown E(\overrightarrow{w})$ jest wektorem, którego składowymi są pochodne cząstkowe E względem każdego z w_i.
- Gradient interpretowany jest jako wektor w przestrzeni wag, gradient określa kierunek, największego wzrostu E. Zmiana zwrotu tego wektora daje zatem kierunek wektor największego spadku wartości E.

- Wyszukiwanie z wykorzystaniem gradientu wag określa wektor wag, który minimalizuje E zaczynając od dowolnego początkowego wektora wag, a następnie wielokrotnie go modyfikując małymi.
- Ujemna wartość tego wektora wskazuje kierunek największego spadku. Ponieważ gradient określa kierunek najbardziej stromego wzrostu E, regułą uczącą dla spadku gradientu jest $w_i \leftarrow w_i + \Delta w_i$, gdzie $\Delta w_i = \text{-}\alpha \frac{\partial E}{\partial w_i}$.
- Znak minus jest obecny, ponieważ chcemy przesunąć wektor wagi w kierunku, w którym zmniejsza się E.
- Na każdym etapie wektor wag jest zmieniany w kierunku, najbardziej stromego zejścia wzdłuż powierzchni błędu.
- Ten proces trwa do momentu osiągnięcia minimum globalnego błędu.
- Aby zbudować praktyczny algorytm iteracyjnego aktualizowania wag, jak wspomniano, potrzebny jest skuteczny sposób obliczania gradientu dla wszystkich kroków.

Najbardziej stromego zejścia

Błąd średniokwadratowy $E[\vec{w}] = \frac{1}{2}\sum_{\vec{x}\in U}(y(\vec{x}) - \vec{w}\cdot\vec{x})^2$

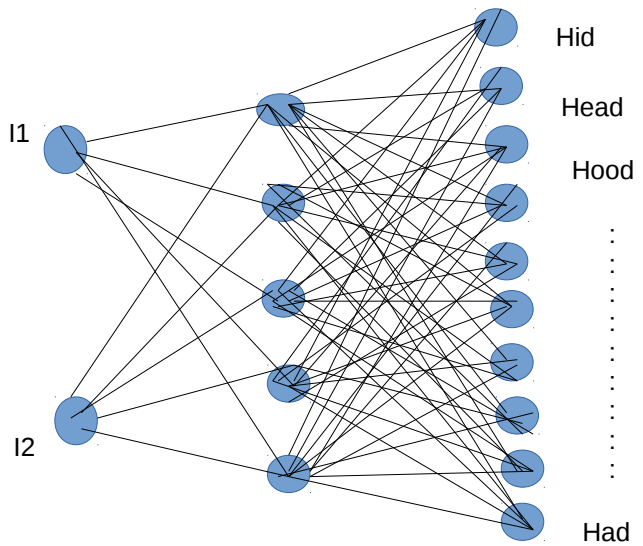
$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{\vec{x}\in U}(y(\vec{x}) - \vec{w}\cdot\vec{x})^2 \\
&= \frac{1}{2}\sum_{\vec{x}\in U}\frac{\partial}{\partial w_i}(y(\vec{x}) - \vec{w}\cdot\vec{x})^2 \\
&= \frac{1}{2}\sum_{\vec{x}\in U} 2(y(\vec{x}) - \vec{w}\cdot\vec{x})\frac{\partial}{\partial w_i}(y(\vec{x}) - \vec{w}\cdot\vec{x}) \\
&= \sum_{\vec{x}\in U}(y(\vec{x}) - \vec{w}\cdot\vec{x})\frac{\partial}{\partial w_i}(y(\vec{x}) - \vec{w}\cdot\vec{x}) \\
&= \sum_{\vec{x}\in U}(y(\vec{x}) - \vec{w}\cdot\vec{x})(-x_i) \\
&= \sum_{\vec{x}\in U} -(y(\vec{x}) - \vec{w}\cdot\vec{x})x_i
\end{aligned}$$

Stąd $\nabla E[\vec{w}] = \sum_{\vec{x}\in U} -(y(\vec{x}) - \vec{w}\cdot\vec{x})\vec{x}$.

- Tak więc reguła treningowa, aby nauczyć się wektora wag na podstawie najbardziej stromego spadku gradientu to $w_i \leftarrow w_i + \Delta w_i$ gdzie $\Delta w_i = \text{-}\alpha\frac{\partial(E)}{\partial(w_i)} = \alpha.\Sigma_{\overrightarrow{x}\in U}(y(\overrightarrow{x}) - \overrightarrow{w}.\overrightarrow{x}).x_i$.
- α jest dodatnią stałą zwaną współczynnikiem uczenia się.

Wielowarstwowa sieć neuronowa

- Jak wspomniano wcześniej, pojedyncze perceptrony mogą wyrażać jedynie decyzyjne liniowe powierzchnie.
- Istnieją proste funkcje, takie jak XOR, a także złożone zadania, takie jak rozpoznawanie mowy, które obejmuje nieliniową reprezentację.
- Przedstawiona tutaj sieć została wyuczona rozpoznawania 1 z 10 dźwięków samogłoskowych występujących w kontekście 'h...d' (np. had, hid). Wejście sieci składa się z dwóch parametrów, I_1 i I_2, uzyskanych z analizy spektralnej dźwięku. Dziesięć wyjść sieci odpowiada 10 możliwym dźwiękom samogłosek. Decyzją sieci jest wyjście, którego wartość jest najwyższa.



Wielowarstwowa sieć neuronowa

Jednostki:

Jednostki podzielone są na warstwy, każda jednostka przyporządkowana jest do dokładnie jednej warstwy

Wejścia:

Wejścia podłączone są wyłącznie do jednostek znajdujących się w najniższej warstwie

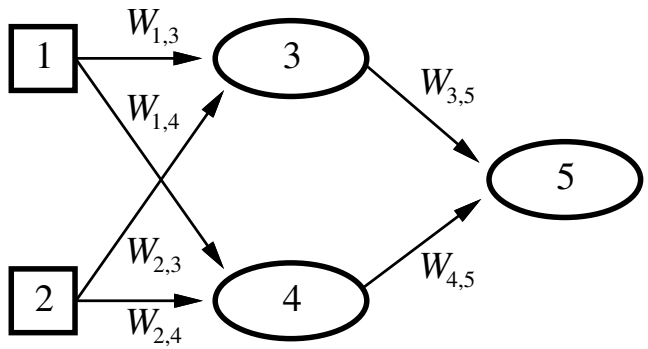
Połączenia:

Połączenia występują wyłącznie pomiędzy jednostkami z sąsiednich warstw, łączą zawsze wyjścia jednostek z warstwy niższej z wejściami do jednostek w warstwie wyższej

Wyjście:

Typowa sieć z jedną wartością funkcji ma tylko jedną jednostkę w najwyższej warstwie, wyjście z tej jednostki jest wyjściem całej sieci

Wielowarstwowa sieć neuronowa: ewaluacja



$$x_5 = \sigma(w_{3,5} \cdot x_3 + w_{4,5} \cdot x_4)$$
$$= \sigma(w_{3,5} \cdot \sigma(w_{1,3} \cdot x_1 + w_{2,3} \cdot x_2) + w_{4,5} \cdot \sigma(w_{1,4} \cdot x_1 + w_{2,4} \cdot x_2))$$

Uwaga:

Zastosowanie liniowych funkcji przejścia w warstwach pośrednich byłoby nieefektywne - warstwy takie można pominąć, poprzez odpowiednie przeliczenie wag i bezpośrednie połącznie warstw poprzedającej i następującej.

Dziękuję za uwagę