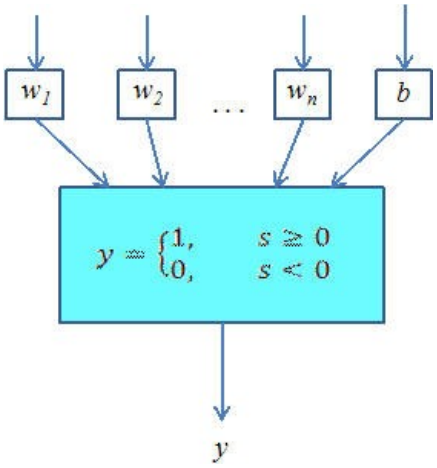
Barbara PArtyka
Sprawozdanie 1.
Temat ćwiczenia: Budowa i działanie perceptronu.
Cel ćwiczenia: Poznanie budowy i działania perceptonu poprzez implementację oraz uczenie perceptronu wybraną funkcję logiczną dwóch zmiennych.

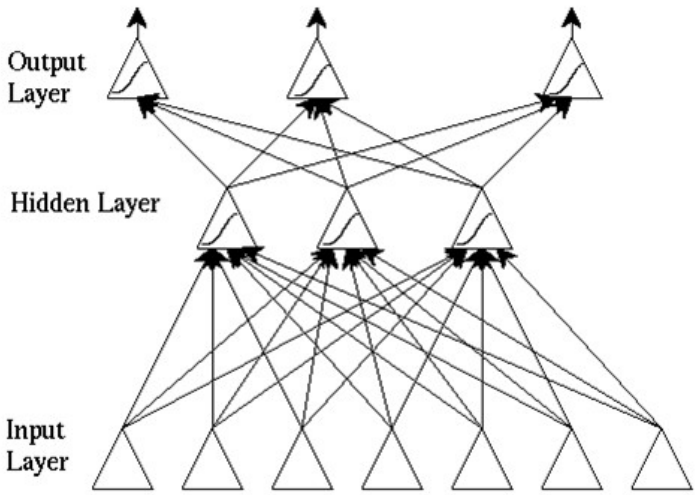
Wstęp teoretyczny:

Perceptronem nazywamy prosty element obliczeniowy, który sumuje ważone sygnały wejściowe i porównuje tę sumę z progiem aktywacji - w zależności od wyniku perceptron może być albo wzbudzony (wynik 1), albo nie (wynik 0).



Algorytm uczenia perceptronu - tzn. automatyczny dobor wag na podstawie napływających przykładów.
Algorytm : Inicjujemy wagi losowo. Dla każdego przykładu uczącego obliczamy odpowiedź perceptronu, jeśli odpowiedź perceptronu jest nieprawidłowa, to modyfikujemy wagi.

Sieci wielowarstwowe - algorytm ich uczenia (propagacja wsteczna). Architektura polegajaca na tym, że tworzymy kilka (zwykle 3) warstw neuronów połączonych w ten sposób, że wyjścia neuronów należących do warstwy niższej połączone są z wejściami neuronów należących do warstwy wyższej (każdy z każdym), pozwala na tworzenie sieci o niemal dowolnej charakterystyce. Działanie takiej sieci polega na liczeniu odpowiedzi neuronów w kolejnych warstwach - najpierw w pierwszej, do której trafiają sygnały z wejść sieci, potem (na podstawie wyników pierwszej warstwy) liczymy odpowiedzi drugiej warstwy neuronów itd., przy czym odpowiedzi ostatniej warstwy traktowane są jako wyjścia z sieci.



Podobnie jak w przypadku pojedynczego neuronu, główną zaletą sieci neuronowej jest to, że nie musimy "ręcznie" dobierać wag. Możemy te wagi wytrenować, czyli znaleźć ich w przybliżeniu optymalny zestaw za pomocą metody obliczeniowej zwanej wsteczną propagacją błędu. Jest to metoda umożliwiająca modyfikację wag w sieci o architekturze warstwowej , we wszystkich jej warstwach.

Ogólny schemat procesu trenowania sieci wygląda następująco:

1. Ustalamy topologię sieci, tzn. liczbę warstw, liczbę neuronów w warstwach.
2. Inicjujemy wagi losowo (na małe wartości).
3. Dla danego wektora uczącego obliczamy odpowiedź sieci (warstwa po warstwie).
4. Każdy neuron wyjściowy oblicza swój błąd, oparty na różnicy pomiędzy obliczoną odpowiedzią *y* oraz poprawną odpowiedzią *t*.
5. Błędy propagowane są do wcześniejszych warstw.
6. Każdy neuron (również w warstwach ukrytych) modyfikuje wagi na podstawie wartości błędu i wielkosci przetwarzanych w tym kroku sygnałów.
7. Powtarzamy od punktu 3. dla kolejnych wektorów uczących. Gdy wszystkie wektory zostaną użyte, losowo zmieniamy ich kolejność i zaczynamy wykorzystywać powtórnie.
8. Zatrzymujemy się, gdy średni błąd na danych treningowych przestanie maleć. Możemy też co jakiś czas testować sieć na specjalnej puli nieużywanych do treningu próbek testowych i kończyć trenowanie, gdy błąd przestanie maleć.

Plik trainingData.txt:

```
in: 1.0 0.0
out: 1.0
in: 1.0 1.0
out: 0.0
in: 1.0 0.0
out: 1.0
in: 0.0 1.0
out: 1.0
in: 0.0 1.0
out: 1.0
in: 0.0 1.0
out: 1.0
in: 0.0 1.0
out: 1.0
in: 1.0 1.0
out: 0.0
in: 1.0 1.0
out: 0.0
in: 0.0 1.0
out: 1.0
in: 0.0 0.0
```

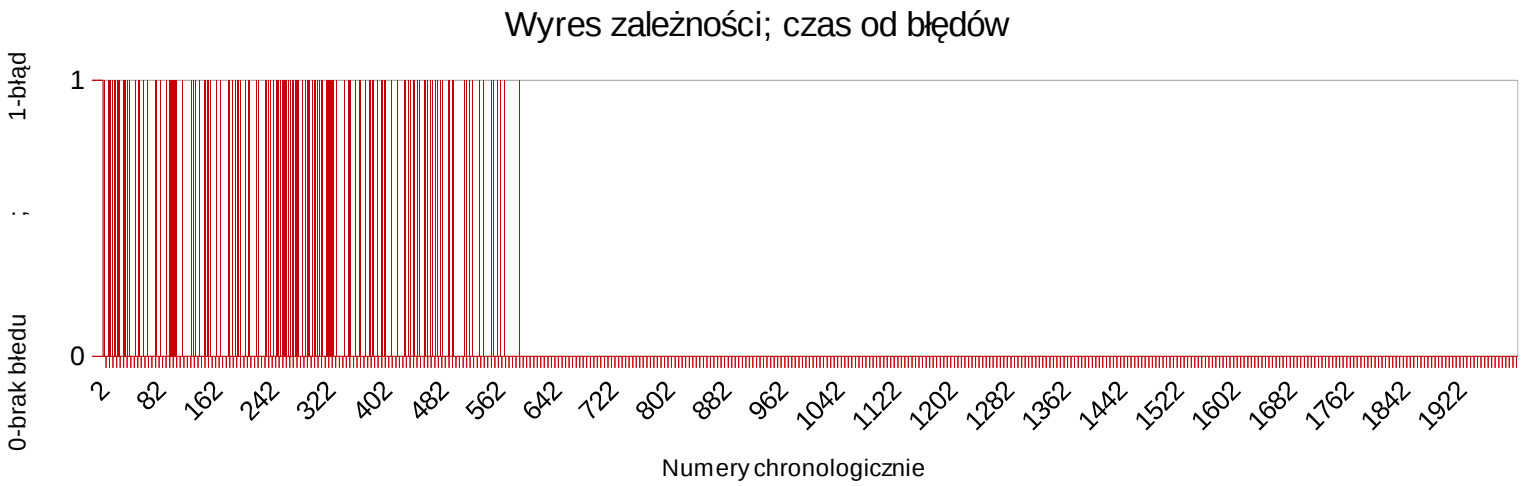
out: 0.0
in: 0.0 1.0
out: 1.0
in: 0.0 0.0
out: 0.0
in: 0.0 0.0
out: 0.0
in: 1.0 0.0
out: 1.0
in: 1.0 1.0
out: 0.0
in: 1.0 0.0
out: 1.0
in: 1.0 1.0
out: 0.0
in: 0.0 1.0
out: 1.0
in: 0.0 1.0
out: 1.0
in: 1.0 1.0
out: 0.0
in: 0.0 1.0
out: 1.0
in: 0.0 0.0
out: 0.0
in: 1.0 1.0
out: 0.0
in: 0.0 1.0
out: 1.0
in: 1.0 0.0
out: 1.0
in: 0.0 1.0
out: 1.0
in: 1.0 0.0
out: 1.0
in: 0.0 0.0
out: 0.0
in: 0.0 0.0
out: 0.0
in: 0.0 1.0
out: 1.0
in: 1.0 1.0
out: 0.0
in: 0.0 1.0
out: 1.0
in: 0.0 1.0
out: 1.0
in: 1.0 1.0
out: 0.0
in: 1.0 0.0
out: 1.0
in: 0.0 1.0

out: 1.0
in: 1.0 1.0
out: 0.0
in: 0.0 1.0
out: 1.0
in: 1.0 0.0
out: 1.0
in: 1.0 0.0
out: 1.0
in: 0.0 1.0
out: 1.0
in: 1.0 0.0
out: 1.0
in: 0.0 1.0
out: 1.0
in: 0.0 1.0
out: 1.0
in: 1.0 1.0
out: 0.0
in: 1.0 1.0
out: 0.0
in: 0.0 1.0
out: 1.0
in: 0.0 1.0
out: 1.0
in: 1.0 1.0
out: 0.0
in: 0.0 0.0
out: 0.0
in: 1.0 1.0
out: 0.0
in: 0.0 1.0
out: 1.0
in: 1.0 0.0
out: 1.0
in: 1.0 0.0
out: 1.0
in: 1.0 0.0
out: 1.0
in: 0.0 0.0
out: 0.0
in: 0.0 0.0
out: 0.0
in: 1.0 0.0
out: 1.0
in: 0.0 1.0
out: 1.0
in: 0.0 0.0
out: 0.0
in: 0.0 1.0
out: 1.0
in: 1.0 1.0

out: 0.0
in: 1.0 0.0
out: 1.0
in: 1.0 0.0
out: 1.0
in: 0.0 0.0
out: 0.0
in: 1.0 0.0
out: 1.0
in: 0.0 1.0
out: 1.0
in: 0.0 1.0
out: 1.0
in: 0.0 1.0
out: 1.0
in: 0.0 1.0
out: 1.0
in: 0.0 1.0
...

Program makeTrainingSamples.cpp tworzy dane na podstawie których sztuczny neuron w programie neural-net.cpp się uczy. Dane do uczenia neuronu zawarte są w pliku tekstowym trainingData. W moim przykładzie wykorzystałam alternatywe rozłączne XOR.

A	B	A **XOR** B
0	0	0
0	1	1
1	0	1
1	1	0



Na podstawie otrzymanych wyników, można stwierdzić, iż z czasem częstość występowania błędów zmierza do zera. Już około 600 wywołania błędy przestają się pojawiać, co pokazuje iż sztuczny neuron nauczył się funkcji logicznej XOR.

Listing kodu:
https://github.com/barbarapar/PSI_GCP03_zima_2017-2018_Barbara_Partyka