Anna Budzoń

Sprawozdanie 1

Data oddania: 20.10.2017r.

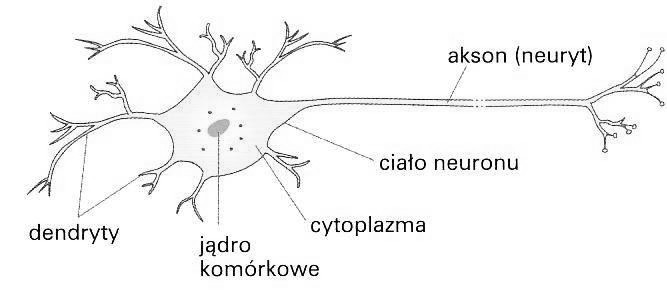
1. Temat ćwiczenia:

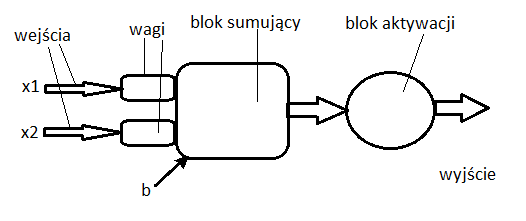
Budowa i działanie perceptronu prostego, jednowarstwowego.

1. Cel ćwiczenia:

Poznanie budowy i działania perceptronu poprzez implementację i uczenie perceptronu realizacji wybranej funkcji logicznej.

1. Porównanie budowy neuronu ze sztucznym neuronem:





Na pierwszym rysunku przedstawiona jest budowa komórki nerwowej, a na drugim sztuczny neuron. Możemy zaobserwować wyraźne podobieństwo:

1. Komórka nerwowa odbiera sygnały za pomocą dendrytów (wypustek komórki nerwowej stykających się z innymi komórkami), a później przekazuje je do aksonu. W sztucznym neuronie dendryty przedstawione są jako liczne wejścia – dane wejściowe xn. W naszym przypadku będziemy potrzebować jedynie dwa sygnały wejściowe – x1 i x2.
2. Ponadto odwzorowaniem aksonu (często bardzo długa wypustka przekazująca sygnał do kolejnych neuronów) jest wartość wyjściowa ( w tym zadaniu wynosi 0 lub 1).
3. Parametry wewnętrzne perceptronu to wagi połączeń, symulujące wagi synaps łączących naturalne neurony,
4. Gdy sygnał przechodzi przez komórkę nerwową następuje nieliniowa obróbka sygnału: stan wyjścia nie jest prostą sumą tego, co wpłynęło na wejścia. Podobne zdarzenie zachodzi w sztucznym neuronie.
5. Operacje zachodzące wewnątrz sztucznego neuronu:
   1. wymnożenie danych wejściowym przez odpowiadające im wagi - to właśnie one są zasadniczym elementem w procesie uczenia perceptronu. Początkowo są wybrane losowo, a następnie w kolejnych epokach, ich wartość jest optymalizowana.
   2. następnie w bloku sumującym iloczyny *xn* i wag *w1,w2,…wn* są sumowane ze sobą. Musimy także uwzględnić w sumie bias (*b*), czyli wartość odchylenia odpowiadająca za nieliniowe przekształcenie wejść w wyjście. Bias jest stałą losową na wejściu, tak jak w przypadku wag.
   3. w bloku aktywacji określa się funkcję aktywacji, która jest prostą służącą do dzielenia przestrzeni, klasyfikująca dane wejściowe do odpowiednich grup. Do naszego zadania wykorzystamy funkcję progową unipolarną:

gdzie θ jest zadaną wartością progową (zmienna threshold) i przyjmujemy, że wynosi 0.

* 1. W ten sposób otrzymujemy wartość wyjściową y, która wynosi 0 lub 1, zależną od sumy ∑.

1. Realizacja projektu:
2. Wykorzystanie algorytmu szukania wag – reguła perceptronu, czyli szczególny przypadek reguły Widrowa-Hoffa.
3. Język implementacji : Java.
4. Funkcja logiczna: OR.
5. **Przebieg programu:**

- określenie stałych:

1. LEARNING RATE – współczynnik uczenia, od którego zależy jak szybko neuron będzie się uczył, tzn. jak szybko dobierze odpowiednie wagi,
2. MAX\_ITERATION – maksymalna liczba wykonania się pętli do-while, czyli maksymalna ilość epok uczenia się,
3. NUM\_X – ilość zestawów uczących,
4. NUM\_WEIGHTS – liczba wag, zależna od ilości danych wejściowych xn,
5. threshold – wartość progowa w funkcji aktywacji.

- wylosowanie wartości początkowych wag w przedziale od 0 do 1 za pomocą funkcji randomNumber,

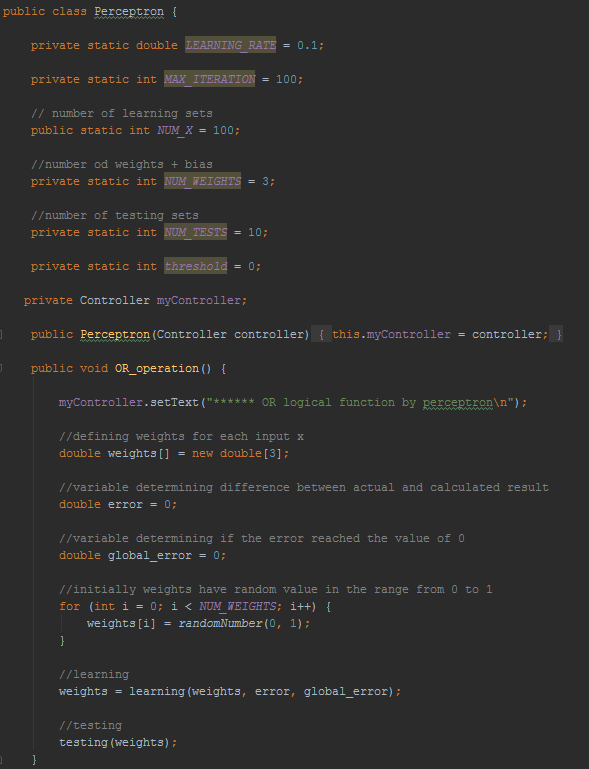
- w procesie uczenia się perceptronu pobieramy z pliku dane uczące i z odpowiednią liczbą zestawów (NUM\_X). Zawarte są wszystkie kombinacje danych wejściowym wraz z rzeczywistymi wynikami funkcji logicznej OR (actual\_outputs),

- najważniejszą częścią całego programu jest pętla do-while, w której sztuczny neuron uczy się:

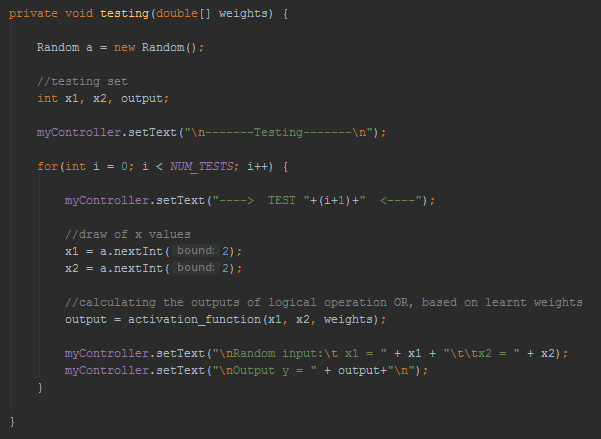
1. wywołujemy funkcję activation\_function, która przyjmuje jako argumenty dane wejściowe i wagi: według algorytmu oblicza ona sumę ∑ i zwraca wartość wyjściową 0 lub 1, zależnie od wartości funkcji aktywacji,
2. obliczamy różnicę między rzeczywistymi wynikami funkcji logicznej OR, a obliczonymi przez perceptron (error),
3. uaktualniamy wagi (weights): dodajemy do nich współczynnik uczenia wymnożony przez error i wartość odpowiadającej zmiennej wejściowej. W ten sposób dążymy do tego, by perceptron dobrał takie wagi, dzięki którym otrzymalibyśmy wyniki najbardziej zbliżone, dążące do prawidłowych,
4. global\_error służy do kontrolowania czy błąd obliczeń nie zbliża się do 0, a więc jest warunkiem zakończenia się pętli.
5. Gdy osiągniemy maksimum iteracji albo global\_error będzie równy zero, procedura uczenia się perceptonu zakończy się. W innym wypadku jest kontynuowana.

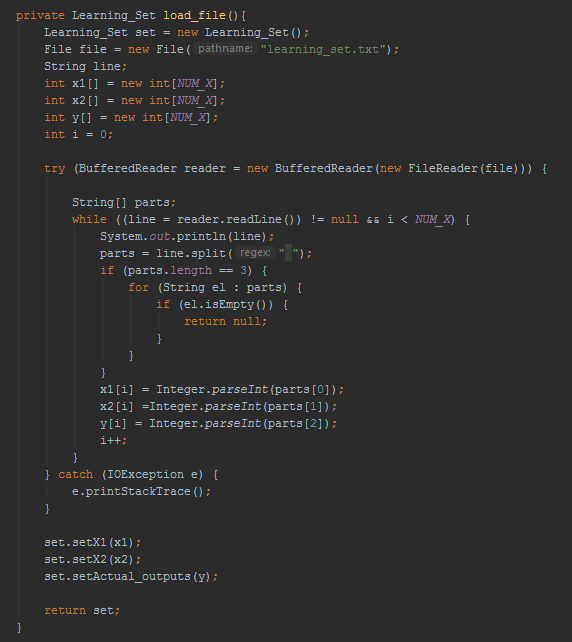
- po fazie uczenia się następuje faza testowania. W tym celu losowane są wartości wejściowe, wynoszące 0 lub 1, które wraz z określonymi wcześniej wagami przesyłane są do funkcji activation\_function.

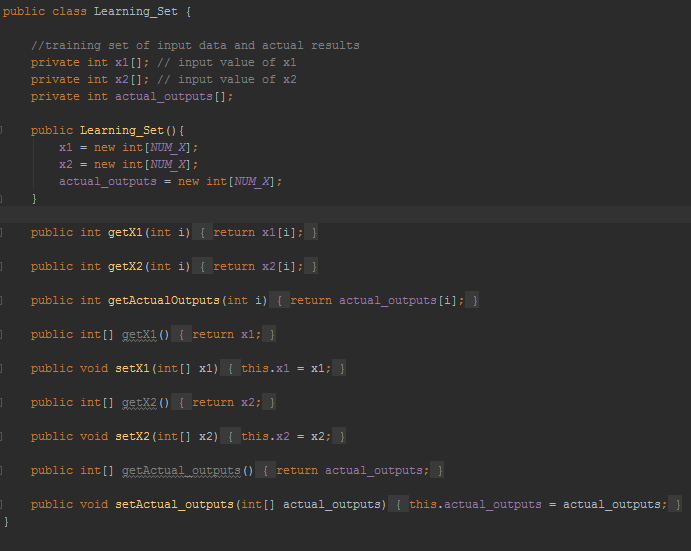
1. **Listing kodu programu:**



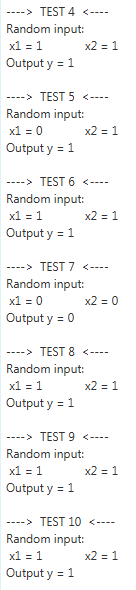
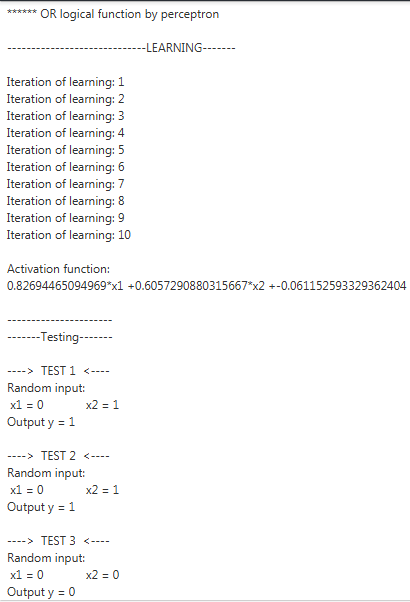








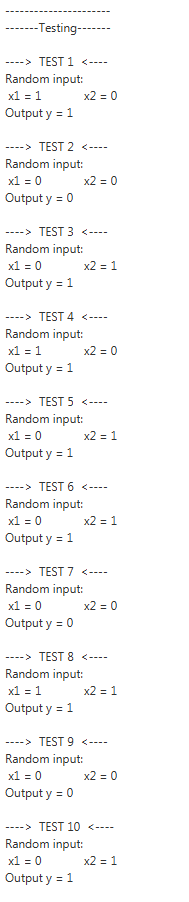
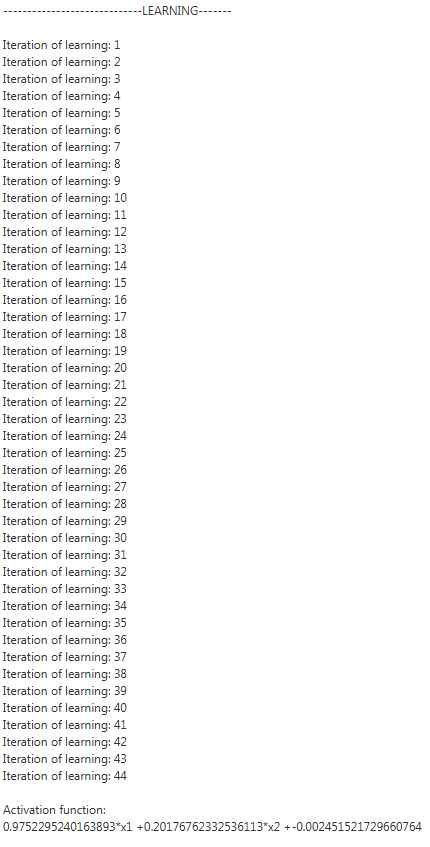
1. Wyniki:
   1. Dla LEARNING\_RATE = 0.1 i NUM\_X = 4:



Liczba epok uczenia się: 10

Liczba błędnych wyników: 0

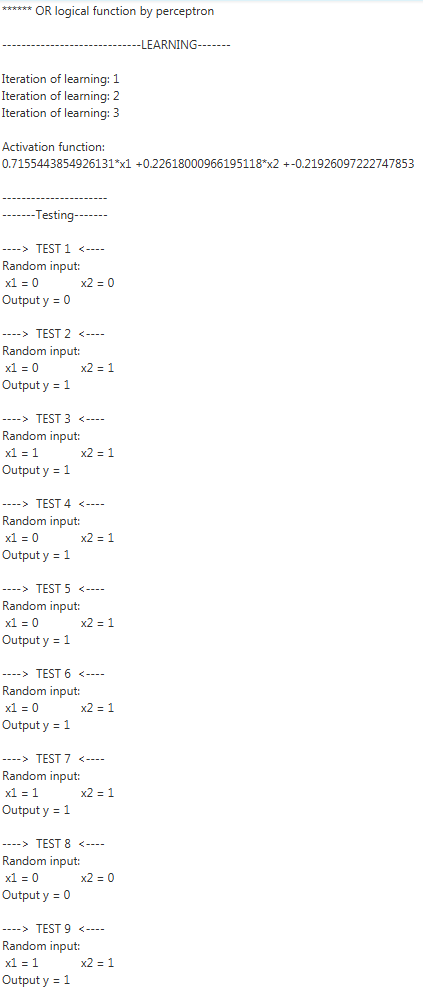
* 1. Dla LEARNING\_RATE = 0.01 i NUM\_X = 4:



Liczba epok uczenia się: 44

Liczba błędnych wyników: 0

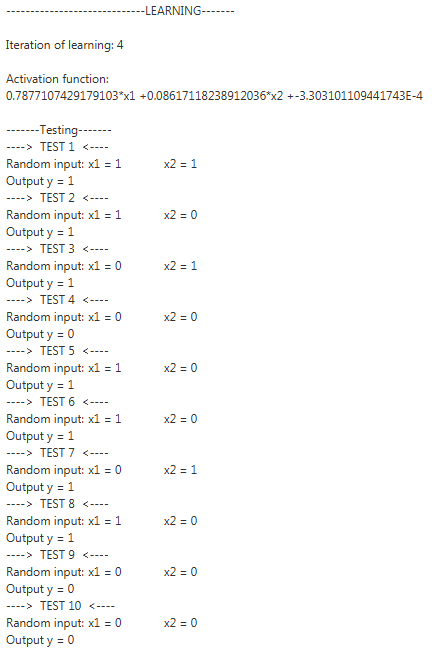
* 1. LEARNING\_RATE = 0.5 i NUM\_X = 4:



Liczba epok uczenia się: 3

Liczba błędnych wyników: 0

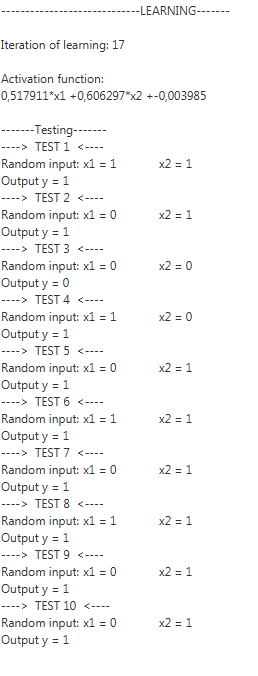
* 1. LEARNING\_RATE = 0.01 i NUM\_X = 40:



Liczba epok uczenia się: 4

Liczba błędnych wyników: 0

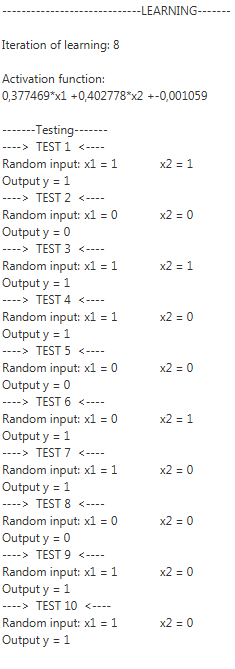
* 1. LEARNING\_RATE = 0.005 i NUM\_X = 40:



Liczba epok uczenia się: 2

Liczba błędnych wyników: 0

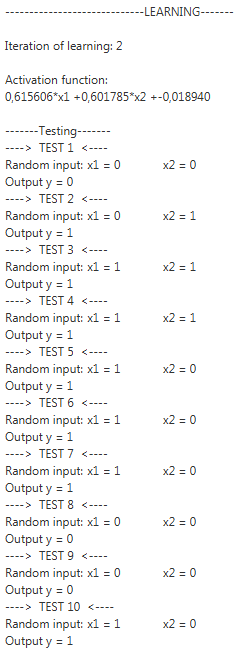
* 1. LEARNING\_RATE = 0.002 i NUM\_X = 60:



Liczba epok uczenia się: 8

Liczba błędnych wyników: 0

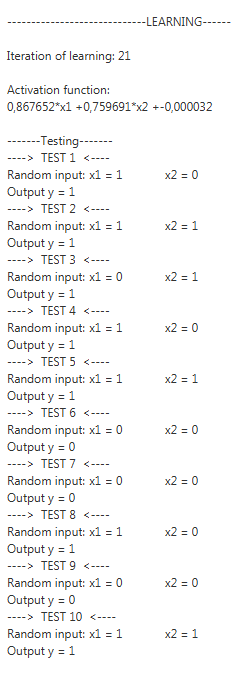
* 1. LEARNING\_RATE = 0.02 i NUM\_X = 80:



Liczba epok uczenia się: 2

Liczba błędnych wyników: 0

* 1. LEARNING\_RATE = 0.001 i NUM\_X = 100:



Liczba epok uczenia się: 21

Liczba błędnych wyników: 0

1. Wnioski:
   1. Współczynnik uczenia się perceptronu ma decydujący wpływ na szybkość jego uczenia się - im wyższy tym szybciej ten proces przebiega. Przy LEARNING\_RATE = 0.5 lub LEARNING\_RATE = 0.2 sztuczny neuron potrzebował niewiele iteracji, by wyliczane dane wyjściowe były prawidłowe, nawet przy małej liczbie danych wejściowych.
   2. Na szybkość uczenia miała też ogromny wpływ liczba danych wejściowych, uczących. Im była większa tym perceptron potrzebował mniej epok uczenia się. Największy wpływ możemy zaobserwować przy małym współczynniku uczenia się, kiedy to następuje bardzo duży rozrzut pomiędzy liczbą epok dla 4, a 100 danych wejściowych.
   3. Z wykresów możemy wywnioskować, że najszybciej neuron będzie uczył się dla jak największej liczby danych uczących oraz przy dużym współczynniku uczenia się. Przy zbyt małej liczbie danych wejściowych perceptron może podawać niewłaściwe odpowiedzi.
   4. Perceptron bardzo szybko jest w stanie nauczyć się funkcji logicznych. Ponadto bardzo łatwo jest zmodyfikować program w taki sposób, by działał dla różnych funkcji (np. OR czy AND). Wystarczy zmienić zestaw uczący, a zostanie określona inna funkcja klasyfikująca.
   5. Podczas każdego wykonanego testowania, nie wykryto błędu w obliczeniach perceptronu. Można podejrzewać, że stało się tak, ponieważ funkcje logiczne są jednymi z najprostszych operacji i prawdopodobieństwo otrzymania dobrego wyniku jest bardzo duże.