**SPRAWOZDANIE**

**Scenariusz 1:**

**Temat**: Budowa i działanie perceptronu **Autor:** Tomasz Zapiórkowski

**Cel Ćwiczenia:**

Celem ćwiczenia było poznanie budowy i działania perceptronu poprzez implementację oraz uczenie perceptronu realizującego wybraną funkcję logiczną dwóch zmiennych.

**Algorytm uczenia perceptronu:**

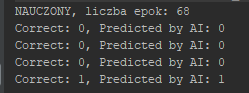
* Zadanie właściwych danych wejściowych i wyjściowych służących do uczenia.
* Zadanie współczynnika uczenia, liczby danych uczących oraz threshold.
* Inicjalizacja wag losowymi liczbami zmiennoprzecinkowymi z zakresu 0-1
* Wykonywanie poniższych formuł, aż do nauczenia lub gdy skończą się dane uczące:
  + Sprawdzenie poprawności (threshold computations).
  + Wyliczenie błędu lokalnego i globalnego.
  + Zaktualizowanie wag poprzez dodanie przemnożonych przez siebie: współczynnika uczenia, danych wejściowych i błędu lokalnego.
  + Sprawdzenie czy nie skończyły się dane uczące albo nie osiągnęliśmy zadowalającej poprawności.
* Zadanie nowych danych testowych i sprawdzenie poprawności przewidywań.

**Wyniki:**

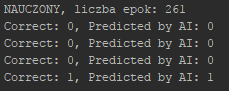
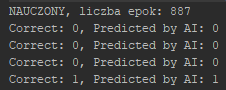
Niezmiennymi danymi były dane wejściowe odpowiadające bramce AND:



W zależności od próby zmieniano liczbę epok, współczynnik uczenia oraz threshold.

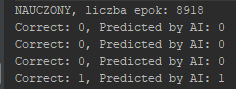
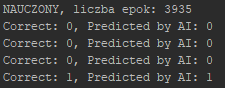


Przy każdej zmianie parametru robiono 10 prób i zapisywano wyniki. Metodą porównawczą było sprawdzanie ile epok program potrzebuje by nauczyć się bezbłędnie przewidywać wyniki.

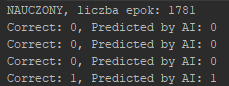
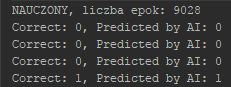
 

Pierwsze dwie próby wykonano przy stałym threshold wynoszącym 0,01 oraz maksymalnej liczbie epok 10000. Zmieniał się jedynie lrate. W pierwszych 10 pomiarach wynosił od 0,001 a następnie 0,0001.

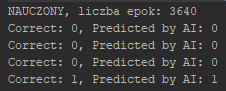
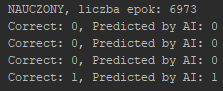
Analizując powyższe dane możemy bez wątpienia stwierdzić, że zmniejszony lrate zwiększa ilość epok wymaganych do nauczenia perceptronu.

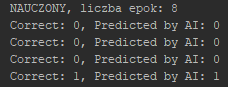
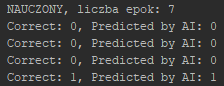
Następnym testem było zbadanie ile epok będzie koniecznych przy stałym lrate wynoszącym 0,0001 ale zmieniającym się threshold, odpowiednio: 0,01 i 0,2.

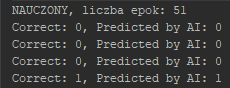
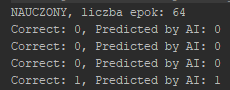
 

W tym przypadku różnice nie są już tak duże jak przy zmianie lrate, ale widać wyraźną tendencję: większy threshold zmniejsza ilość niezbędnych epok.

Kolejny test polegał na optymalizacji dobieranych wartości przy jednoczesnym zapewnieniu poprawności otrzymanych wyników. Threshold ustalono na 0,2 lrate zmieniano i przyjmował wartości 0,1 i 0,01.

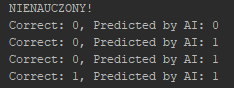
 

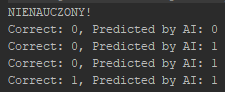
 

Wyniki pokazują że dość wysoki lrate i tak zapewnia poprawność wyników końcowych przy stosunkowo małej liczbie epok potrzebnej do nauki.

**Analiza:**

Najważniejszą wartością niezbędną do nauczenia perceptronu była ilość epok. Kiedy zadano ich zbyt mało i perceptron nie osiągał odpowiednio niskiego błędu, to nie potrafił przewidywać poprawnych wyników i bardzo często się mylił.

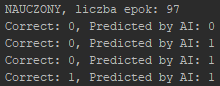
 

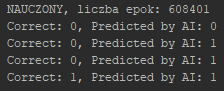




*Źle dobrane warunki uczenia się (za mało epok „NUM\_EPOCHS” w stosunku do pozostałych współczynników*

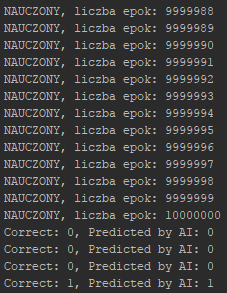
Kolejnym istotnym wyznacznikiem jest stosunek lrate do threshold. Aby perceptron działał prawidłowo musi on wynosić <1, tzn. lrate musi być zawsze mniejszy niż threshold. Przeprowadzono kilka testów, które jednoznacznie to pokazały.

*Źle dobrane warunki uczenia się (zły stosunek LEARNING\_RATE/THRESHOLD)*

Uruchamiając program bez warunku błędu minimalnego zauważono też, że po pewnym czasie nauka przebiega nieefektywnie lub też nawet jest zaburzana przez zbyt dużą ilość danych.



**Wnioski:**

Ćwiczenie zapoznało mnie z podstawami zagadnień sztucznej inteligencji w szczególności perceptronu. Elementarna wiedza zdobyta podczas tego ćwiczenia z pewnością będzie niezbędna przy projektowaniu bardziej zaawansowanych sieci neuronowych.

Perceptron jest bardzo skutecznym narzędziem, gdy nasze dane wejściowe nie są skomplikowane czy trudno wyznaczalne. Dla funkcji logicznej dwóch zmiennych sprawdził się znakomicie i już po kilku epokach potrafił poprawnie przewidywać wyniki (przy odpowiednim dobraniu parametrów uczenia).

Jednakże zastosowany algorytm nie ustrzegł się błędów. Dość łatwo można było go zepsuć stosując nieodpowiedni współczynnik uczenia bądź nie zadając wystarczająco danych uczących.

Reasumując: dzięki tym ćwiczeniom poznałem zasadę działania perceptronu, a także udało mi się ją zastosować w praktyce. Zastosowany algorytm nauczył mnie jakich błędów nie popełniać by perceptron działał prawidłowo.

**Listing:**

Perceptron.java

*/\*\*  
 \* Created by madrimas on 15.10.2017.  
 \*/*import java.util.Random;  
  
public class Perceptron {  
 double[] weightVector;  
 double thresholdComputation;  
  
 public void perceptron(double[][] inputData, int[] outputData, double thresholdComputation, double learningRate, int numberOfEpochs) {  
 this.thresholdComputation = thresholdComputation;  
 int lengthOfInputs = inputData[0].length;  
 int lengthOfOutputs = outputData.length;  
 weightVector = new double[lengthOfInputs]; //creating weightVector of input data size  
 Random randomNumbers = new Random(); //creating random number generator  
  
 int epochsCounter = 0;  
 int globalError;  
 int localError;  
 int outcome;  
  
 for (int i = 0; i < lengthOfInputs; i++) {  
 weightVector[i] = randomNumbers.nextDouble(); //complete vector with random doubles (0-1)  
 }  
  
 do {  
 epochsCounter++;  
 globalError = 0;  
 for (int i = 0; i < lengthOfOutputs; i++) {  
 outcome = calculateThreshold(inputData[i]);//threshold computations  
 localError = outputData[i] - outcome;//compute local error  
 globalError += localError;//adding local error to global error  
 for (int j = 0; j < lengthOfInputs; j++) {  
 double delta = learningRate \* inputData[i][j] \* localError;//compute new weight  
 weightVector[j] += delta;  
 }  
 }  
 } while ((globalError != 0) && (epochsCounter < numberOfEpochs));//  
 }  
  
 public int calculateThreshold(double[] inputData) {  
 double sum = 0.0;  
 for (int i = 0; i < inputData.length; i++) {  
 sum += weightVector[i] \* inputData[i];  
 }  
 return sum >= thresholdComputation ? 1 : 0;//validation test  
 }  
}

PerceptronTest.java

*/\*\*  
 \* Created by madrimas on 15.10.2017.  
 \*/*public class PerceptronTest {  
  
 static int *NUM\_EPOCHS* = 1000000000;//maximum numbers of epochs  
 static double *LEARNING\_RATE* = 0.01;  
 static double *THRESHOLD* = 0.2;  
  
 public static void main(String[] args) {  
 Perceptron perceptron = new Perceptron();  
 double inputData[][] = {{0, 0}, {0, 1}, {1, 0}, {1, 1}};//learning data  
 int outputData[] = {0, 0, 0, 1};//learning data  
  
 perceptron.perceptron(inputData, outputData, *THRESHOLD*, *LEARNING\_RATE*, *NUM\_EPOCHS*);  
 System.*out*.println("Correct: 0, Predicted by AI: " + perceptron.calculateThreshold(new double[]{0, 0}));  
 System.*out*.println("Correct: 0, Predicted by AI: " + perceptron.calculateThreshold(new double[]{1, 0}));  
 System.*out*.println("Correct: 0, Predicted by AI: " + perceptron.calculateThreshold(new double[]{0, 1}));  
 System.*out*.println("Correct: 1, Predicted by AI: " + perceptron.calculateThreshold(new double[]{1, 1}));  
 }  
}

*Materiały źródłowe:*[*http://www.codebytes.in/2015/07/perceptron-learning-algorithm-java.html*](http://www.codebytes.in/2015/07/perceptron-learning-algorithm-java.html)[*http://edu.pjwstk.edu.pl/wyklady/nai/scb/wyklad3/w3.htm*](http://edu.pjwstk.edu.pl/wyklady/nai/scb/wyklad3/w3.htm)[*https://www.ii.uni.wroc.pl/~aba/teach/NN/w4.pdf*](https://www.ii.uni.wroc.pl/~aba/teach/NN/w4.pdf)