Kamil Wasilonek

**PSI – SPRAWOZDANIE**

LABORATORIUM NR 1

1. **Cel​ ​ćwiczenia:**

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

1. **Opis wykonanych zadań:**

Zaimplementowałem sztuczny neuron w oparciu o model McCullocha-Pittsa.Sygnały dochodzące do neuronu (I1, I2, ...IN) mnożone są przez tzw. wagi Wi(i - numer neuronu). Dodatkowy sygnał jednostkowy nazwano biasem. Na podstawie wag i dochodzących sygnałów obliczany jest stan wewnętrzny neuronu. Odpowiedź neuronu zależy od tego czy jego stan wewnętrzny (pobudzenie) przekracza pewien poziom. Odpowiada za to unipolarna funkcja aktywacji która w modelu McCullocha-Pittsa jest funkcją progową.



Rys. 1 Schemat neuronu McCullocha-Pittsa

Klasa „Perceptron” odpowiadająca za perceptron składa się z 3 funkcji:

* **„calculateOutput”** odpowiada za sumowanie iloczynów wartości „input” podanych na wejściu oraz wag „weights” .
* **„activationFunction”** jest aktywacyjną funkcja progową unipolarną. Na jej podstawie obliczana jest wartość wyjścia perceptronu.
* **„learnFunction”** wykonuje modyfikacje wag wykorzystując dane otrzymane z powyższych funkcji. W argumentach metody podajemy również oczekiwaną prawidłową wartość oraz „krok uczenia”. Obie te czynniki wykorzystujemy do modyfikacji wag.

Kolejną klasą wykorzystaną w programie jest „Main” w której podajemy dane wykorzystywane do uczenia perceptronu oraz informacje takie jak liczba powtórzeń uczenia i ilość epok. Nauka realizowana jest w dwóch pętlach „for” (pierwszej zależnej od podanej liczby powtórzeń oraz drugiej zależnej od długości tablicy zawierającej dane wejściowe). Stworzona została również pętla „while” realizująca proces uczenia i testowania perceptronu do momentu uzyskania oczekiwanych wartości.

1. **Zestawienie otrzymanych wyników:**

W programie wykorzystałem funkcje „AND”. Początkowo podaje każdy zestaw prawidłowych danych jednokrotnie, a następnie testuje uczenie się perceptronu dla różnych wag i obserwuje po ilu krokach otrzymam prawidłowe wyniki. Podczas analizy parametr „learn\_rate” został ustawiony na 0.1.

*Tab.1 Zależność wag od ilości potrzebnych kroków*

|  |  |
| --- | --- |
| Waga | Ilość potrzebnych kroków |
| **0.3** | **3** |
| **0.1** | **5** |
| **0.8** | **5** |
| **1.5** | **9** |
| **3** | **17** |

*Wykres.1 Zestawienie dla „leran\_rate”=0.1 oraz wagi= 3*

Kolejną modyfikacja sprawdzającą działanie programu była modyfikacja „learn\_rate”. Użyte wagi początkowe wynosiły 0.5.

*Tabela.2 Zależność „learn\_rate” o ilości potrzebnych kroków*

|  |  |
| --- | --- |
| learn\_rate | Ilość potrzebnych kroków |
| **0.1** | **3** |
| **0.3** | **1** |
| **0.5** | **5** |
| **0.8** | **7** |
| **1.5** | **1** |

Trzeci przeprowadzony test polegał na losowaniu wag początkowych oraz zadawaniu różnych „learn\_rate”, a nastepnie obserwacji ile epok będzie niezbędne do nauczenia perceptronu i otrzymania oczekiwanych wartości. Na podstawie 20 prób wybrałem najmniejsza liczbę epok.

*Tabela.3 Zależność „learn\_rate” o ilości potrzebnych kroków*

|  |  |
| --- | --- |
| learn\_rate | Ilość potrzebnych kroków |
| 0.01 | 16 |
| 0.8 | 1 |
| 1.5 | 1 |

*Wykres.2 Zestawienie dla „leran\_rate” = 0.01*

Następnie przeprowadziłem testy podając różną ilość poszczególnych danych uczących

(przy ustawieniu wag równych 0.5 oraz „learn\_rate” wynoszącym 0.1).

*Tabela.4 wyniki przeprowadzonych testów dla poszczególnych danych*

|  |  |  |  |
| --- | --- | --- | --- |
| Input\_1 | Input\_2 | Liczba wprowadzeń danego zestawu | Otrzymany wynik |
| 0 | 0 | 1 | Błędny |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

*Tabela.5* *wyniki przeprowadzonych testów dla poszczególnych danych*

|  |  |  |  |
| --- | --- | --- | --- |
| Input\_1 | Input\_2 | Liczba wprowadzeń danego zestawu | Otrzymany wynik |
| 0 | 0 | 1 | Błędny |
| 0 | 1 | 3 |
| 1 | 0 | 1 |
| 1 | 1 | 2 |

*Tabela.6 wyniki przeprowadzonych testów dla poszczególnych danych*

|  |  |  |  |
| --- | --- | --- | --- |
| Input\_1 | Input\_2 | Liczba wprowadzeń danego zestawu | Otrzymany wynik |
| 0 | 0 | 2 | Błędny |
| 0 | 1 | 4 |
| 1 | 0 | 2 |
| 1 | 1 | 1 |

*Tabela.7 wyniki przeprowadzonych testów dla poszczególnych danych*

|  |  |  |  |
| --- | --- | --- | --- |
| Input\_1 | Input\_2 | Liczba wprowadzeń danego zestawu | Otrzymany wynik |
| 0 | 0 | 3 | Prawidłowy |
| 0 | 1 | 2 |
| 1 | 0 | 3 |
| 1 | 1 | 2 |

*Tabela.8 wyniki przeprowadzonych testów dla poszczególnych danych*

|  |  |  |  |
| --- | --- | --- | --- |
| Input\_1 | Input\_2 | Liczba wprowadzeń danego zestawu | Otrzymany wynik |
| 0 | 0 | 5 | Prawidłowy |
| 0 | 1 | 6 |
| 1 | 0 | 3 |
| 1 | 1 | 0 |

1. **Wnioski**

Ćwiczenie które wykonałem wymagało zaimplementowania w oparciu o wybrany algorytm perceptronu. Dzięki wykonanym operacjom obserwujemy w jaki sposób modyfikacja podstawowych danych wykorzystywanych do nauki neuronu wpływa na ostateczne wyniki programu.

Jednym z najważniejszych czynników są wagi dla odpowiednich danych wejściowych. Głównym założeniem jest taka ich modyfikacja, aby błąd popełniany przez perceptron był jak najmniejszy, zaś obliczony wynik był jak najbardziej zbliżony do oczekiwanej wartości. Dobór wag początkowych nie ma istotnego wypływu na działanie programu, ponieważ w czasie jego trwania są modyfikowane w celu uzyskania poprawnego wyniku. Należy jednak pamiętać aby wartości te były odpowiednio dobrane, ponieważ może to wpłynąć na czas uczenia się perceptronu. Duży wpływ na proces nauki ma „współczynnik uczenia”. Na podstawie zestawionych wyników testów obserwujemy jak niewielka jego zmiana może doprowadzić do zwiększenia liczby potrzebnych epok. Gdy współczynnik jest zbyt mały , czas poszukiwania wyniku będzie za długi , zaś przy zbyt dużym możemy pominąć odpowiednie wartości. Mój program opiera się na metodzie uczenia z nauczycielem, dlatego kolejnym ważnym parametrem są dane uczące jakie wykorzystujemy do nauki. Dla funkcji logicznej „AND” przy jednokrotnym podaniu każdego zestawu danych udało się osiągnąć prawidłowy wynik już po jednej epoce. Warto jednak zauważyć, że nie jest wymagane podawanie ich w takiej samej ilości. Przeprowadzone testy ukazują ,że istnieje możliwość nauczenia perceptronu bez podawania pełnego zestawu informacji. Udało się znaleźć kombinacje danych wejściowych, która prowadzi do poprawnego wyniku, jednak pomija pare liczb (1,1). Przeprowadzone ćwiczenie mimo iż opiera się na prostej implementacji bardzo dobrze prezentuje ideę „uczenia neuronu”. Ukazuje jak modyfikacja użytych danych (np. „współczynnika uczenia”) może wpłynąć na czas jaki będzie potrzebny do uzyskania prawidłowego wyniku.

1. **Listing całego kodu**

**Perceptron:**

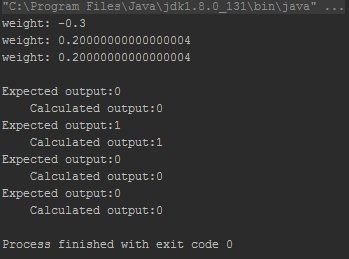
import java.util.Random;  
  
// Klasa implementująca perceptron  
public class Perceptron {  
public double[] getWeights() {  
return weights;  
}  
  
// Pola używane w klasie  
private double[] weights;  
 private intnumerOfInput;  
Random random= new Random();  
  
  
// Konstruktor klasy nadający początkowe wagi  
public Perceptron(intnumberOfInput) {  
this.numerOfInput= numberOfInput;  
weights = new double[numberOfInput];  
 for (int i = 0; i<weights.length; i++) {  
//weights[i] = 3;  
weights[i] = random.nextDouble();  
System.*out*.println("Start weight:" + i + " " + weights[i]);  
System.*out*.println();  
}  
 }  
  
  
//Funkcja aktywacji oparta na unipolarnej funkcji progowej  
public int activationFunction(double output) {  
if (output <0) return 0;  
 else return 1;  
}  
  
  
//Funkcja sumująca  
public int calculateOutput(int[] input) {  
double output = 0;  
 for (int i = 0; i< this.numerOfInput; i++) {  
 output += (input[i] \* weights[i]);  
}  
return activationFunction(output);  
}  
  
  
// Funkcja ucząca, która modyfikuje wagi przy każdym kroku uczenia  
public void learnFunction(int[] input, double y, double learn\_rate) {  
double output = calculateOutput(input);  
 for (int i = 0; i<weights.length; i++) {  
weights[i] += (y - output) \* learn\_rate \* input[i];  
}  
 }  
}

**Main:**

import java.io.\*;  
import java.util.Arrays;  
  
public class Main {  
  
public static void main(String[] args) throws IOException {  
  
//Tworzenie perceptronu z nadaniem żądanej ilości wejść  
Perceptron perceptron = new Perceptron(3);  
  
//Podanie ilości epok i kroku uczenia  
int howManyStepsToLearn = 0;  
 double learn\_rate = 0.1;  
  
//Podanie danych uczących  
int bias = 1;  
int[] input1 = {0, 1, 0, 1};  
int[] input2 = {0, 1, 1, 0};  
  
//Podanie oczekiwanych wartości dla poszczególnych zestawów uczących  
int[] expectedData = {0, 1, 0, 0}; //AND  
int[] outputData = new int[4];  
  
  
// Pętla realizująca uczenie perceptronu założoną ilość epok  
// for (int i = 0; i<howManyStepsToLearn; i++) {  
// for (int j = 0; j < input1.length; j++) {  
// int[] inputData = {bias,input1[j],input2[j]};  
// perceptron.learnFunction( inputData , expectedData[j], learn\_rate);  
// }  
// }  
  
  
 // pętla sprawdzająca zgodnosc wartości wyliczonych z oczekiwanymi  
while (!Arrays.*equals*(expectedData, outputData)) {  
  
for (inti = 0; i<4; i++) {  
int[] inputData = {bias, input1[i], input2[i]};  
perceptron.learnFunction(inputData, expectedData[i], learn\_rate);  
}  
  
for (inti = 0; i<4; i++) {  
outputData[i] = perceptron.calculateOutput(new int[]{bias, input1[i], input2[i]});  
}  
  
howManyStepsToLearn++;  
  
 if (howManyStepsToLearn>1000) break;  
}  
  
// wypisanie końcowych wag  
double weight[] = perceptron.getWeights();  
 for (inti = 0; i<weight.length; i++) {  
System.*out*.println("\nEnd weight: " + weight[i]);  
}

// wypisanie końcowego wyliczonego wyniku  
for (int i = 0; i<outputData.length; i++) {  
System.*out*.println(outputData[i]);  
}  
  
System.*out*.println("\nIle kroków potrzebnych do nauki: " + howManyStepsToLearn);  
  
}  
}

**Przykładowy wynik działania programu:**

****

1. Bibliografia

http://wwwold.ece.utep.edu/research/webfuzzy/docs/kk-thesis/kk-thesis-html/img18.gif