Kamil Wasilonek

**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 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 oraz ilość epok. Realizacja zautomatyzowanego uczenia realizowana jest w dwóch pętlach „for” (pierwszej zależnej od podanej liczby powtórzeń oraz drugiej zagnieżdżonej zależnej od długości tablicy zawierającej dane wejściowe).

**2.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. „learn\_rate” został ustawiony na 0.1.

Dla wag:

* **Dla wagi 0.3 prawidłowy wynik otrzymałem po 3 krokach**

Dla powyższego przykładu prezentuje wykres zależności epok uczenia od procentu popełnionego błędu:

*Wykres.1 Zależność epok uczenia od procentu popełnionego błedu*

* Dla wagi 0.1 prawidłowy wynik otrzymałem po 5 krokach
* Dla wagi 0.8 prawidłowy wynik otrzymałem po 5 krokach
* Dla wagi 1.5 prawidłowy wynik otrzymałem po 9 krokach
* Dla wagi 3 prawidłowy wynik otrzymałem po 17 krokach

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

• Dla „learn\_rate” 0.1 prawidłowy wynik otrzymałem po 3 krokach

• **Dla „learn\_rate” 0.3 prawidłowy wynik otrzymałem po 1 krokach**

• Dla „learn\_rate” 0.5 prawidłowy wynik otrzymałem po 5 krokach

• Dla „learn\_rate” 0.8 prawidłowy wynik otrzymałem po 7 krokach

• **Dla „learn\_rate” 1.5 prawidłowy wynik otrzymałem po 1 krokach**

Następnie przeprowadziłem testy podając różną ilość poszczególnych danych uczących

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

|  |  |  |  |
| --- | --- | --- | --- |
| Input\_1 | Input\_2 | Liczba wprowadzeń danego zestawu | Otrzymany wynik |
| 0 | 0 | 1 | Błędny |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |
|  |  |  |  |
| Input\_1 | Input\_2 | Liczba wprowadzeń danego zestawu | Otrzymany wynik |
| 0 | 0 | 1 | Błędny |
| 0 | 1 | 3 |
| 1 | 0 | 1 |
| 1 | 1 | 2 |
|  |  |  |  |
| Input\_1 | Input\_2 | Liczba wprowadzeń danego zestawu | Otrzymany wynik |
| 0 | 0 | 2 | Błędny |
| 0 | 1 | 4 |
| 1 | 0 | 2 |
| 1 | 1 | 1 |
|  |  |  |  |
| Input\_1 | Input\_2 | Liczba wprowadzeń danego zestawu | Otrzymany wynik |
| 0 | 0 | 3 | Prawidłowy |
| 0 | 1 | 2 |
| 1 | 0 | 3 |
| 1 | 1 | 2 |
| Input\_1 | Input\_2 | Liczba wprowadzeń danego zestawu | Otrzymany wynik |
| 0 | 0 | 5 | Prawidłowy |
| 0 | 1 | 6 |
| 1 | 0 | 3 |
| 1 | 1 | 0 |

**3.Wnioski**

Na podstawie wykonanego ćwiczenia które wymagało zaimplementowania w oparciu o wybrany algorytm perceptronu obserwujemy w jaki sposób modyfikacja podstawowych danych wykorzystywanych do uczenia neuronu wpływa na ostateczny wyniki programu.

Jednym z najważniejszych czynników są wagi dla odpowiednich danych wejściowych. Głównym założeniem uczenia 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. W moim przypadku najlepszą wagą była wartość 0.5 która przy odpowiedniej wartości „learn\_rate” tj. 0.1 lub 1.5, pozwalała uzyskać prawidłowy wynik już po pierwszej epoce. Należy jednak pamiętaj aby wartości te również były odpowiednio dobrane ponieważ może to wpłynąć na czas uczenia się perceptronu. Mój program opiera się na metodzie uczenia z nauczycielem, dlatego kolejnym ważnym czynnikiem 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 danych w takiej samej ilości. Przeprowadzone testy ukazują ,że istnieje możliwość nauczenia perceptronu bez podawania pełnego zestawu danych dla funkcji „AND”. Udało się znaleźć kombinacje danych wejściowych która prowadzi do poprawnego wyniku, jednak pomija dane (1,1). Przeprowadzone ćwiczenie mimo iż wykorzystuje najprostszą implementację neuronu bardzo dobrze prezentuje ideę „uczenia neuronu”. Ukazuje jak modyfikacja użytych danych(np. „learn\_rate”) może wpłynąć na czas jaki będzie potrzebny do uzyskania prawidłowego wyniku.

**4.Listing całego kodu**

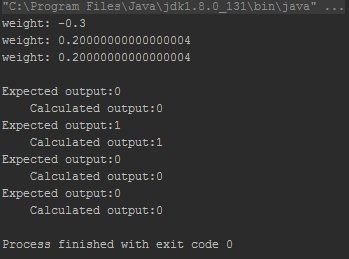
**Perceptron:**

import java.util.Random;  
  
public class Perceptron {  
  
 public double[] getWeights() {  
 return weights;  
 }  
  
 private double[] weights;  
 private int numerOfInput;  
 Random random = new Random();  
  
 public Perceptron(int numberOfInput) {  
 this.numerOfInput = numberOfInput;  
 weights = new double[numberOfInput];  
 for (int i = 0; i < weights.length; i++) {  
 weights[i] = 0.5;  
 }  
 }  
  
 public void randWeight(double[] weight) {  
 for (int i = 0; i < weight.length; i++) {  
 weight[i] = random.nextDouble();  
 System.*out*.println("Weight " + i + " " + weight[i]);  
 }  
 }  
  
 public int activationFunction(double output) {  
 if (output < 0) return 0;  
 else return 1;  
 }  
  
 public int calculateOutput(int[] input) {  
 double output = 0;  
 for (int i = 0; i < this.numerOfInput; i++) {  
 output += (input[i] \* weights[i]);  
 }  
 return activationFunction(output);  
 }  
  
 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:**

public class Main {  
  
 public static void main(String[] args) {  
  
 Perceptron perceptron = new Perceptron(3);  
  
 int howManyStepsToLearn = 2;  
 double learn\_rate = 0.1;  
  
 int bias = 1;  
 int[] input1 = { 0, 1, 0, 1 };  
 int[] input2 = { 0, 1, 1, 0 };  
  
 int[] expectedData = {0, 1, 0, 0}; //AND  
  
 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);  
 }  
 }  
   
   
 double weight[] = perceptron.getWeights();  
 for(int i = 0 ; i < weight.length ; i++ ){  
 System.*out*.println("weight: " + weight[i]);  
 }  
  
 System.*out*.println();  
  
 System.*out*.println("Expected output:" + expectedData[0]);  
 System.*out*.println("\tCalculated output:" + perceptron.calculateOutput(new int[] {bias, 0, 0}));  
 System.*out*.println("Expected output:" + expectedData[1]);  
 System.*out*.println("\tCalculated output:" + perceptron.calculateOutput(new int[] {bias, 1, 1}));  
 System.*out*.println("Expected output:" + expectedData[2]);  
 System.*out*.println("\tCalculated output:" + perceptron.calculateOutput(new int[] {bias, 0, 1}));  
 System.*out*.println("Expected output:" + expectedData[3]);  
 System.*out*.println("\tCalculated output:" + perceptron.calculateOutput(new int[] {bias, 1, 0}));  
 }  
}

**Przykładowy wynik:**

****

Bibliografia

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