|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Jarosz Wojciech | Gr. 1 | PSI | Sprawozdanie 1 | Temat: Budowa i działanie perceptronu |

1. Cel ćwiczenia:

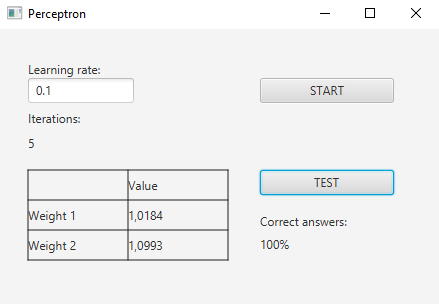
Celem ćwiczenia jest poznanie budowy i działania perceptronu poprzez implementację oraz uczenie perceptronu realizującego wybraną funkcję logiczną dwóch zmiennych.

1. Wykonane zadania:
   1. Zaimplementowano sztuczny neuron.
   2. Jako funkcję logiczną wybrano funkcję OR a następnie wygenerowano dane uczące
   3. Przeprowadzono etap uczenia dla różnych współczynników uczenia.
   4. Przetestowano perceptron
2. Syntetyczny opis budowy oraz wykorzystanego algorytmu uczenia:
3. Inicjalizacja losowych wag z zakresu (0,1)
4. Obliczenie sumy ze wzoru:
5. Wykorzystanie funkcji aktywacyjnej (w tym przypadku progowej):
6. Obliczenie błędu:
7. Wyznaczenie nowych wag:

*wagai+1 = (współczynnik\_uczenia) \* błąd \* xi + wagai*

1. Powtarzanie kroków 2-5, dopóki błąd jest równy zero dla wszystkich wektorów wejściowych {x1,x2} tzn. otrzymany wynik musi być równy oczekiwanemu wynikowi dla każdej pary zmiennych.
2. Testowanie perceptronu: sprawdzenie ilości poprawnych odpowiedzi dla N losowych danych wejściowych.
3. Zestawienie wyników:

Przykładowe dane wyjściowe:



Itr: 1

0,6184 0,6993 0 0 0 0 0.0 0,0000 0,6184 0,6993

0,6184 0,6993 0 1 1 0 1.0 0,6993 0,6184 0,7993

0,6184 0,7993 1 0 1 0 1.0 0,6184 0,7184 0,7993

0,7184 0,7993 1 1 1 1 0.0 1,5177 0,7184 0,7993

Itr: 2

0,7184 0,7993 0 0 0 0 0.0 0,0000 0,7184 0,7993

0,7184 0,7993 0 1 1 0 1.0 0,7993 0,7184 0,8993

0,7184 0,8993 1 0 1 0 1.0 0,7184 0,8184 0,8993

0,8184 0,8993 1 1 1 1 0.0 1,7177 0,8184 0,8993

Itr: 3

0,8184 0,8993 0 0 0 0 0.0 0,0000 0,8184 0,8993

0,8184 0,8993 0 1 1 0 1.0 0,8993 0,8184 0,9993

0,8184 0,9993 1 0 1 0 1.0 0,8184 0,9184 0,9993

0,9184 0,9993 1 1 1 1 0.0 1,9177 0,9184 0,9993

Itr: 4

0,9184 0,9993 0 0 0 0 0.0 0,0000 0,9184 0,9993

0,9184 0,9993 0 1 1 0 1.0 0,9993 0,9184 1,0993

0,9184 1,0993 1 0 1 0 1.0 0,9184 1,0184 1,0993

1,0184 1,0993 1 1 1 1 0.0 2,1177 1,0184 1,0993

Itr: 5

1,0184 1,0993 0 0 0 0 0.0 0,0000 1,0184 1,0993

1,0184 1,0993 0 1 1 1 0.0 1,0993 1,0184 1,0993

1,0184 1,0993 1 0 1 1 0.0 1,0184 1,0184 1,0993

1,0184 1,0993 1 1 1 1 0.0 2,1177 1,0184 1,0993

Tabela Zależność ilości iteracji od współczynnik uczenia.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Współczynnik uczenia | 1. | 2. | 3. | 4. | 5. | 6. | 7. | Średnia |
| 0,05 | 17 | 10 | 16 | 18 | 11 | 14 | 16 | 15 |
| 0,1 | 10 | 5 | 4 | 9 | 9 | 8 | 5 | 7 |
| 0,4 | 4 | 3 | 2 | 4 | 3 | 3 | 2 | 3 |

1. Analiza i dyskusja błędów uczenia i testowania opracowanego perceptronu w zależności od wartości współczynnika uczenia oraz liczby danych uczących:

Algorytm uczenia musi obliczyć korekty wag. Te korekty są mnożone przez współczynnik uczenia. Określa on jak na błędy uczenia ma reagować uczeń (sieć), jeśli współczynnik jest wysoki( bliski 1), błędy są nietolerowane i sieć szukając rozwiązania przechodzi od skrajności w skrajność, natomiast przy niskim współczynniku uczenia błędy uczenia są bardziej tolerowane i proces uczenia przebiega powoli. Widać to w przeprowadzonych testach perceptronu, zestawionych w Tabeli 1

1. Wnioski:
   1. Przeprowadzony proces uczenia jest procesem jednowarstwowym tzn. wyznaczana jest „granica” pomiędzy odpowiedziami poprawnymi a błędnymi.
   2. aby funkcja mogła być realizowana przez perceptron prosty sygnały wejściowe dla których funkcja aktywacji przyjmuje wartość 0 muszą leżeć w innej półpłaszczyźnie niż te sygnały wejściowe dla których funkcja ta przyjmuje wartość 1. Mówi się że muszą one być liniowo separowalne.
   3. Im wyższy współczynnik uczenia tym mniejsza ilość iteracji, natomiast im niższy współczynnik tym większa oraz uzyskane wagi mają wartość bliżej 1.
   4. Uczenie trwa do tej chwili aż dal wszystkich danych wejściowych błąd będzie równy 0.
   5. Dla testowania końcowego dla różniej ilości danych wejściowych nie otrzymywano błędów, ze względu na trywialną funkcję logiczną, jaką jest OR.
2. Lisning:

Controller.java

**package** sample;  
**import** javafx.fxml.FXML;  
**import** javafx.scene.control.\*;  
**import** java.util.Random;  
  
**public class** Controller {  
 **double weight1**, **weight2**;  
 @FXML  
 Label **weight1Label**;  
 @FXML  
 Label **weight2Label**;  
 @FXML  
 Label **correctAnswearsLabel**;  
 @FXML  
 TextField **learningRateTextField**;  
 @FXML  
 Label **iterationsLabel**;  
  
 @FXML  
 **private void** initialize() {**learningRateTextField**.setText(Double.*toString*(0.1));}  
  
 @FXML  
 **public void** startButton(){  
 *//int[][] truthTable = Perceptron.andTruthTable;* **int**[][] truthTable = Perceptron.***orTruthTable***;  
 **double**[] weights = {Math.*random*(), Math.*random*()};  
 Perceptron perceptron = **new** Perceptron();  
 **double** learning\_rate = 0.1;  
 **try** {  
 learning\_rate = Double.*parseDouble*(**learningRateTextField**.getText());  
 } **catch** (NumberFormatException e) {  
 **learningRateTextField**.setText(String.*valueOf*(0.1));  
 learning\_rate=0.1;  
 }  
 **int** itrNumber = 0;  
 **boolean** isErrorNotEqualToZero = **true**;  
 **double** error = 0;  
 **double**[] newWeights;  
 **while** (isErrorNotEqualToZero){  
 itrNumber++;  
 System.***out***.println(**"\nItr: "**+itrNumber);  
 isErrorNotEqualToZero = **false**;  
 error =0;  
 **for** (**int** x=0;x<truthTable.**length**;x++){  
 **int**[] tmp={truthTable[x][0],truthTable[x][1]};  
 **double** sum = perceptron.calculateSum(tmp ,weights);  
 **int** result = perceptron.activeationFunction(sum);  
 error = truthTable[x][2] - result;  
 **if**(error !=0) isErrorNotEqualToZero=**true**;  
 newWeights = perceptron.calculateWeights(tmp, learning\_rate,weights,error);  
 System.***out***.println(String.*format*(**"%.4f"**,weights[0])+**" "**+String.*format*(**"%.4f"**,weights[1])+**"\t"**+truthTable[x][0]+**"\t"**+truthTable[x][1]+**"\t"**+truthTable[x][2]+  
 **"\t"**+result+**"\t"**+error+**"\t"**+String.*format*(**"%.4f"**,sum)+**"\t"**+String.*format*(**"%.4f"**,newWeights[0])+**"\t"**+String.*format*(**"%.4f"**,newWeights[1]));  
 **weight1Label**.setText(String.*format*(**"%.4f"**,newWeights[0]));  
 **weight1**=newWeights[0];  
 **weight2Label**.setText(String.*format*(**"%.4f"**,newWeights[1]));  
 **weight2**=newWeights[1];  
 weights =newWeights;  
 }  
  
 }  
 **iterationsLabel**.setText(String.*valueOf*(itrNumber));  
  
 }  
  
 @FXML  
 **public void** startTest() {  
 **if**(**weight1Label**.getText().isEmpty()){  
 **correctAnswearsLabel**.setText(**"Calculation first!!!"**);  
 }**else**{  
 **int** x1, x2;  
 **int** answer=0;  
 **int** N=10000;  
 **double**[] weights ={**weight1**,**weight2**};  
 Random generator = **new** Random();  
 Perceptron perceptron = **new** Perceptron();  
 **for**(**int** i=0; i<N;i++){  
 x1=generator.nextInt(2);  
 x2=generator.nextInt(2);  
 **int**[] inData={x1,x2};  
  
 **double** sum = perceptron.calculateSum(inData,weights);  
 **int** result = perceptron.activeationFunction(sum);  
 *//AND TEST  
 /\* if(result==1){  
 if(x1==1&&x2==1) answear++;  
 }else {  
 if (x1 != 1 || x2 != 1) answear++;  
 }\*/  
 //OR TEST* **if**(result==0){  
 **if**(x1==0&&x2==0) answer++;  
 }**else** {  
 **if** (x1 != 0 || x2 != 0) answer++;  
 }  
  
 }  
 **correctAnswearsLabel**.setText(String.*valueOf*((answer/N)\*100)+**"%"**);  
 }  
 }  
}

Perceptron.java

**package** sample;  
  
*/\*\*  
 \* Created by Wojtek on 18.10.2017.  
 \*/***public class** Perceptron {  
 **public static final int**[][] ***andTruthTable*** = {{0,0,0},{0,1,0},{1,0,0},{1,1,1}};  
 **public static final int**[][] ***orTruthTable*** = {{0,0,0},{0,1,1},{1,0,1},{1,1,1}};  
 **public double** calculateSum(**int**[] truthTable, **double**[] weights){  
 **double** sum = 0;  
 **for**(**int** x=0; x < truthTable.**length**; x++)  
 sum += truthTable[x] \* weights[x];  
 **return** sum;  
 }  
 **public int** activeationFunction(**double** sum){  
 **int** result;  
 **if**(sum > 1) result = 1;  
 **else** result=0;  
 **return** result;  
 }  
 **public double**[] calculateWeights(**int**[] truthTable, **double** learning\_rate, **double**[] weights, **double** error){  
 **double**[] newWeights = **new double**[weights.**length**];  
 **for**(**int** x=0; x < weights.**length**;x++)  
 newWeights[x] = learning\_rate \* error \* truthTable[x] + weights[x];  
 **return** newWeights;  
 }  
}