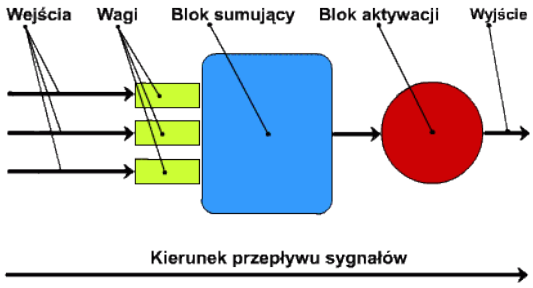
Sprawozdanie

Małgorzata Bień IS gr.1

Temat ćwiczenia: Budowa i działanie perceptronu

W moim programie zaimplementowałam sztuczny neuron według modelu McCullocha-Pittsa.



W dendrytach(wejścia) umieszcza się dane wejściowe, które zostają przemnożone przez wartości wag w odpowiadających im synapsach. Dla każdego powiązania dendryt<→synapsa zachodzi: R=d*w, gdzie R to wynik połączenia, d informacja w dendrycie a w to wartość wagi. Wyniki wszystkich połączeń trafiają do bloku sumującego. Rezultat tej operacji przechodzi przez blok aktywacyjny . W tym miejscu zostaje poddany funkcji aktywacyjnej, która może mieć charakter liniowy lub przyjmować postać bipolarnej funkcji skoku jednostkowego. Na wyjście zostaje oddana wartość zmodyfikowana w bloku aktywacyjnym.
Aby przeprowadzić uczenie potrzebny jest nam zestaw wzorców. Czyli pary danych wejściowych i pożądanych odpowiedzi od neuronu. Dla każdego wzorca liczymy błąd, który z każdą iteracją będzie coraz mniejszy, jest on równy różnicy oczekiwanej wartości i wartości która pojawiła się na wyjściu sieci. Kiedy mamy policzony błąd neuronu dla każdego wzorca wtedy trzeba nanieść na synapsy(wagi) odpowiednie poprawki. Aby to zrobić musimy wcześniej ustalić jeszcze jedną składową procesu uczenia – współczynnik uczenia z zakresu (0-1). Odpowiada on za prędkość uczenia sieci.
Dokonujemy poprawek w synapsach neuronu: do wag na poszczególnych synapsach neuronu dodajemy iloczyn współczynnika uczenia, błędu neuronu i wartości wejściowej na dendrycie.

Opis kodu programu:

Perceptron.java | NeuralFXMLController.java

```
2
3  import java.util.Random;
4
5  public class Perceptron {
6
7      NeuralFXMLController controller = new NeuralFXMLController();
8
9      public static final int[][] ANDdata = {
10             {0,0,0},
11             {0,1,0},
12             {1,0,0},
13             {1,1,1}};
14
15     public static final double LEARNING_RATE=0.01;
16     public static final double[] WEIGHTS = {Math.random(),Math.random()};
17
18     public Perceptron(NeuralFXMLController neuralFXMLController) {
19         this.controller=neuralFXMLController;
20     }
21
22
23     public double processCellNode(int[]dendrites,double[]synapses){
24         double sum=0;
25         for(int i=0; i<dendrites.length; i++){
26             sum += dendrites[i]*synapses[i];
27         }
28         return sum;
29     }
```

W klasie Perceptron na początku ustalam w tablicy ANDdata zestaw wzorców, wejścia i odpowiedzi funkcji logicznej AND. Ustawiam współczynnik uczenia w zmiennej LEARNING_RATE oraz umieszczam losowe wartości wag w synapsach neuronu. W funkcji processCellNode() obliczam sumę połączenia dendryt-synapsa.

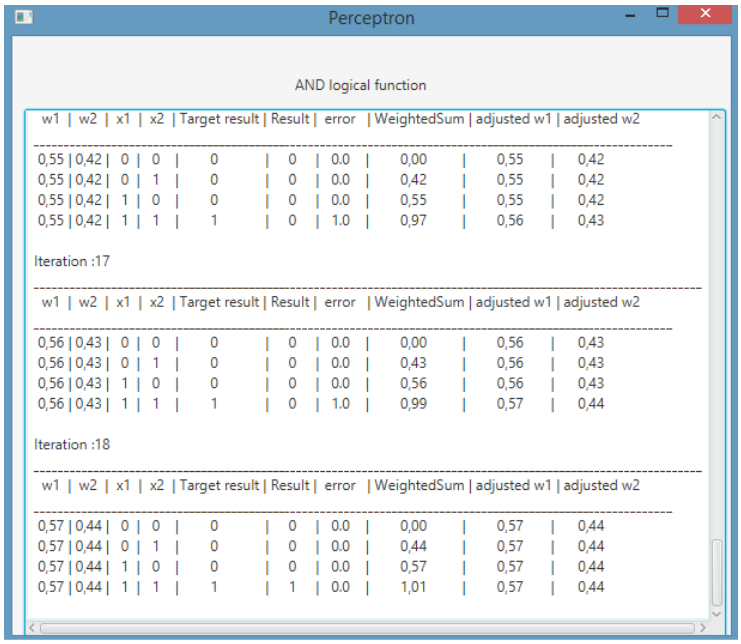
```
public int activationFunction(double cellNode){
    int result=0;
    if(cellNode > 1) {
        result = 1;
    }
    return result;
}

public double[] calculateWeight(int[]data, double[]weights, double error){
    double[] inputWeights =  new double[weights.length];
    for(int i=0; i<weights.length; i++){
        inputWeights[i] = LEARNING_RATE *error * data[i] + weights[i];
    }
    return inputWeights;
}
```

Suma tych połączeń jest przekazywana do funkcji aktywacji – activationFunction(). Metoda calculateWeight() służy do obliczenia korekty wag.

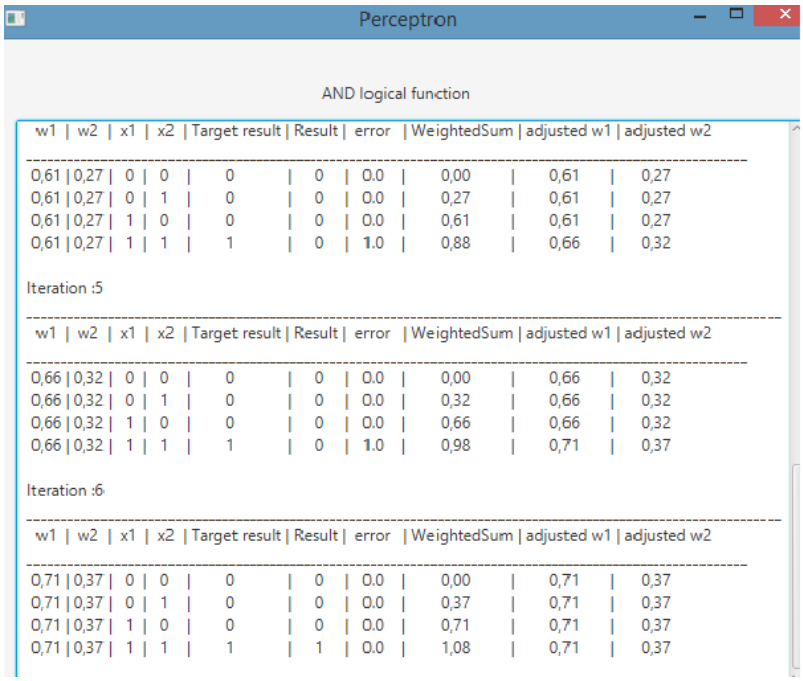
```
double[]weights = Perceptron.WEIGHTS;
int iter = 0;
boolean errorFlag = true;
double error = 0;
double[] adjustedWeights = null;

public void logicalFunction(){
    while(errorFlag){
        controller.setText("\n");
        controller.setText("Iteration :" + iter+"\n");
        controller.setText("_____________________________________________________________________________");
        controller.setText("  w1   |   w2   |   x1   |   x2   | Target result | Result |   error    | Weighted
        controller.setText("_____________________________________________________________________________");
        errorFlag=false;
        error = 0;
        for(int i=0; i<ANDdata.length; i++){
            int[]input={ANDdata[i][0],ANDdata[i][1]};
            double calculateInput =  processCellNode(input,weights);
            int result = activationFunction(calculateInput);
            error = ANDdata[i][2] - result;
            if(error != 0)
                errorFlag = true;

            adjustedWeights = calculateWeight(input, weights, error);
            controller.setText(ANDdata, weights, result, error, calculateInput, adjustedWeights);
            controller.setText("\n");
            weights=adjustedWeights;

        }iter++;
```

W funkcji logicalFunction() zachodzi cały proces uczenia. Wypisujemy numer iteracji by zobaczyć w którym kroku błąd neuronu wyniósł wartość 0.

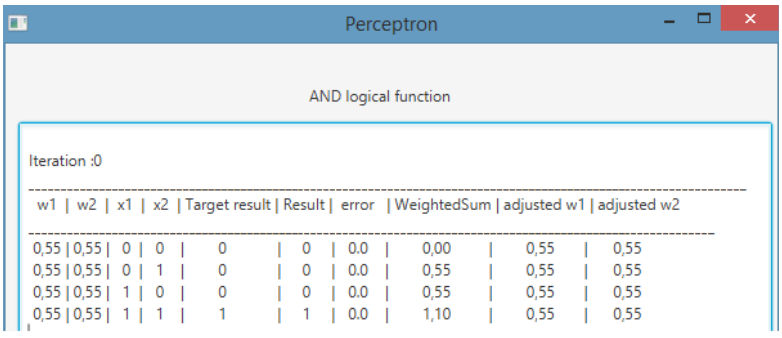
Przykładowe działania programu:
LEARNING_RATE=0.01



LEARNING_RATE=0.05



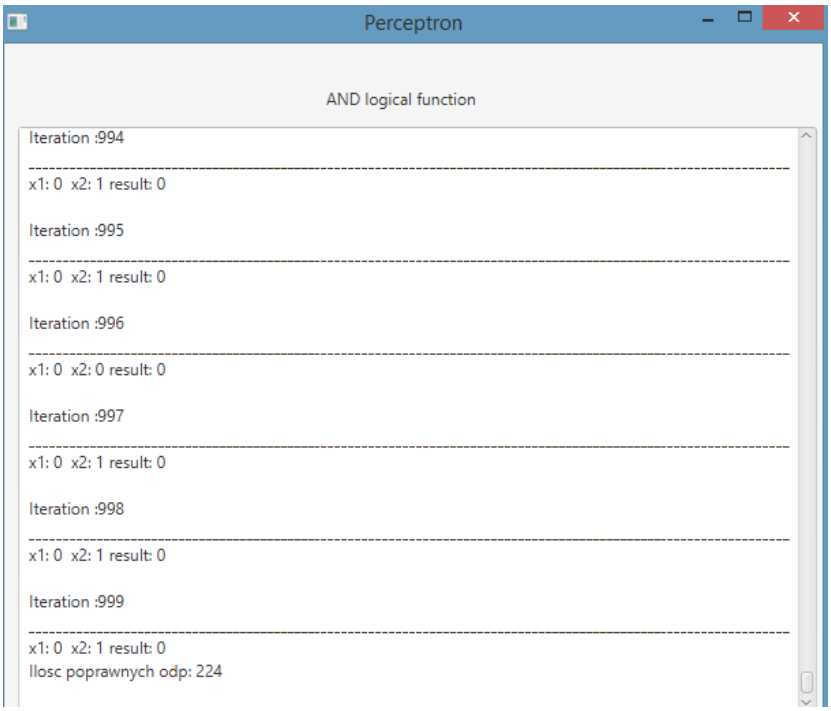
LEARNING_RATE = 0.1



```
public void Testing(){

    int testIter=0;
    int result=0;
    Random r = new Random();
    int x=0,y=0;

    for(int i=0; i<1000; i++){

        controller.setText("\n");
        controller.setText("Iteration :" + testIter+"\n");
        controller.setText("_____________________________________");

        x = r.nextInt(2);
        y = r.nextInt(2);

        int[]testData = { x,y };
        double weightedSum =  processCellNode(testData,weights);
        int doTest = activationFunction(weightedSum);
        if(doTest==1){
            if(x==1 && y==1)
            result++;           //ilość poprawnych odp
        }
        controller.setTestText(x,y,doTest);
        controller.setText("\n");

        testIter++;
        }controller.setText("Ilosc poprawnych odp: "+result+"\n");
```

Funkcja Testing() służy do testowania neuronu dla większej ilości danych wejściowych z zakresu 0 lub 1, już bez pożądanej odpowiedzi.

Przykładowy wynik testowania:



Na 1000 iteracji otrzymaliśmy 224 poprawne odpowiedzi.

Wnioski:

Z otrzymanych wyników można łatwo zauważyć, że prędkość uczenia zależy od współczynnika uczenia. Im większy tym uczenie przebiega szybciej. Gdy ustawiliśmy go na wartość 0.01 błąd neuronu osiągnął wartość 0 dopiero w 18 iteracji a gdy współczynnik był równy 0.1 już w pierwszej iteracji. Niższe wartości spowodują uczenie wolniejsze, mozolne, ale za to dokładne i precyzyjne.