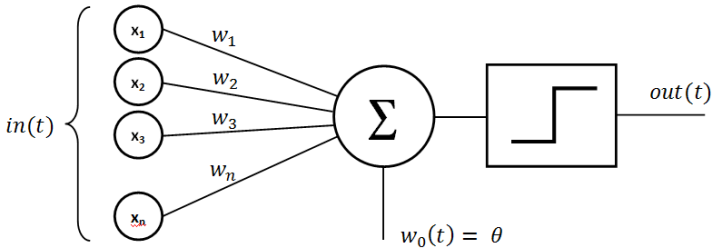
Michał Habigier gr. 1 lab
Scenariusz 1: Budowa i działanie perceptronu.

Źródła:
http://www.cs.stir.ac.uk/courses/ITNP4B/lectures/kms/2-Perceptrons.pdf
https://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15381-f01/www/handouts/110601.pdf

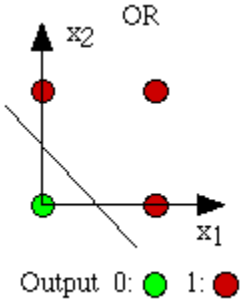
Opis:



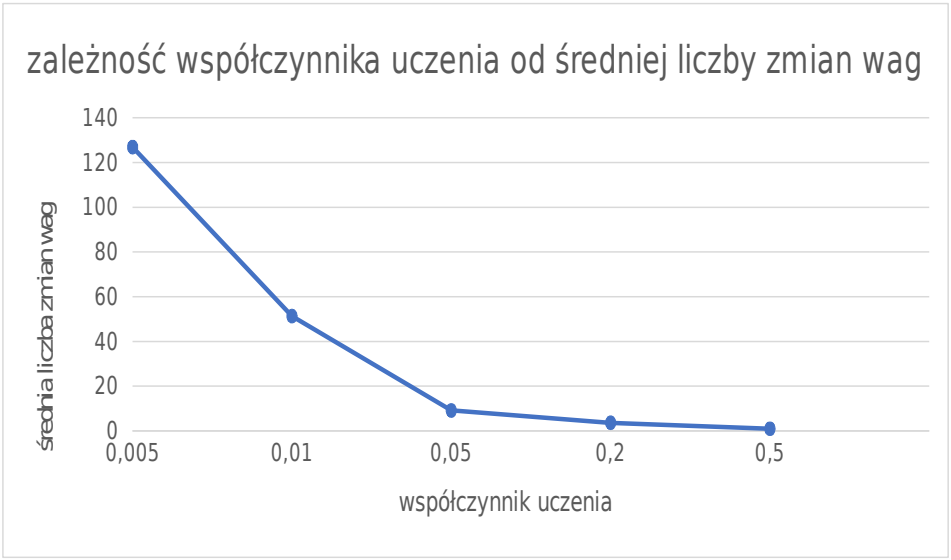
Stworzony przeze mnie perceptron zbudowany jest z neuronów wejściowych x1,x2 i wag w1,w2; sumatora realizującego działanie $\sum_{1}^{n}(wi*xi)$ oraz funkcji nieciągłej typu skokowego (jak w modelu McCullocha-Pittsa), która otrzymuje wynik sumatora i zwraca 0 lub 1.

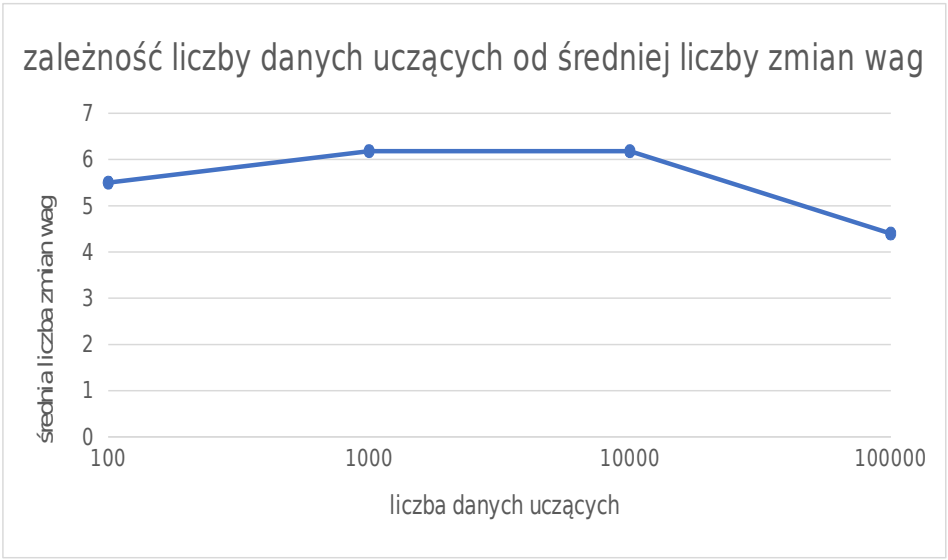
$$y = \begin{cases} 1 & \text{if } u \geq \theta \\ 0 & \text{if } u < \theta \end{cases}$$

Uczenie polega na porównywaniu wartości zwróconej przez funkcję aktywacji z rzeczywistym wynikiem. Jeśli są one różne następuje aktualizacja wag. Proces ten trwa do momentu, aż dla wszystkich wygenerowanych danych uczących aktualizacja wag nie nastąpi – to znaczy, że wagi są odpowiednio dobrane. Perceptron uczony będzie realizacji funkcji logicznej OR. Jest to uczenie z nauczycielem



Wyniki:


zależność współczynnika uczenia od średniej liczby zmian wag
140
120
100
80
60
40
20
0
średnia liczba zmian wag
0,005
0,01
0,05
0,2
0,5
współczynnik uczenia


zależność liczby danych uczących od średniej liczby zmian wag
7
6
5
4
3
2
1
0
średnia liczba zmian wag
100
1000
10000
100000
liczba danych uczących

Analiza:

Wszystkie testy zakończyły się w 100% poprawnymi odpowiedziami.
Wagi były losowane z przedziału <-1,1>. Nauczone wagi zawsze mieściły się w przedziale (0,1).
Dokładne wyniki, na podstawie których zostały wygenerowane wykresy znajdują się w arkuszu kalkulacyjnym.

Z wykresu zależności współczynnika uczenia od średniej liczby zmian wag (średniej z 10 prób) wynika, że im większy współczynnik uczenia, tym szybciej postępuje uczenie.

Z wykresu zależności liczby danych uczących od średniej liczby zmian wag (średniej z 10 prób) wynika, że liczba danych nie wpływa na tempo uczenia. Na wykresie widać bardzo małe odchylenie od średniej, które mieści się w błędzie statystycznym.

Wnioski:

Sposób w jaki zbudowany jest perceptron powoduje szerokie pole do jego zastosowań.

Byłem zaskoczony tym, że pierwszy z wykresów przybrał kształt funkcji hiperbolicznej. Byłem przekonany, że istnieje idealna wartość współczynnika nauczania, jednak jak okazało się im jest on wyższy tym perceptrony uczą się skuteczniej

Stuprocentowa skuteczność testów wydaje się dziwna, jednak mogła mieć na to wpływ prostota testowanej funkcji. Po prostu łatwo jest dobrać takie wagi, aby przy testowaniu za każdym razem otrzymać dobry wynik.

Listing kodu:

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Random;

public class Perceptron {
    private ArrayList<Integer> x1 = new ArrayList<>();
    private ArrayList<Integer> x2 = new ArrayList<>();
    private ArrayList<Integer> expectedResult = new ArrayList<>();
    private double learningRate;
    private Integer dataAmount;
    private Random random = new Random();
    private int weightChangesAmount;
    private final double []weights = {(random.nextDouble() * 2) - 1,
(random.nextDouble() * 2) - 1};

// zapis do pliku wygenerowanych losowych

    public void saveGeneratedData()throws IOException {
        FileWriter fileWriter = new FileWriter("data.txt");
        BufferedWriter bufferedWriter = new BufferedWriter(fileWriter);
        for(int i = 0; i< dataAmount; i++){
            bufferedWriter.write(x1.get(i)+" "+x2.get(i)+" "+
expectedResult.get(i)+"\n");
        }
        bufferedWriter.close();
    }

    public void saveTestResults(int testResult)throws IOException{

        FileWriter fw = new FileWriter("testResults.txt",true);
        BufferedWriter bw = new BufferedWriter(fw);
        bw.write(dataAmount +" "+ learningRate+" "+ testResult+ "\n");
        bw.close();
    }

    public void saveLearningResults() throws IOException{

        FileWriter fw = new FileWriter("learningResults.txt",true);
        BufferedWriter bw = new BufferedWriter(fw);
        bw.write(dataAmount +" "+ learningRate+" "+ weightChangesAmount +
"\n");
        bw.close();
    }
//losowanie dwóch wartości i obliczanie dla nich funkcji OR
    public void generateData(){
        x1.clear();
        x2.clear();
        expectedResult.clear();
        for(int i = 0; i< dataAmount; i++) {
            x1.add(random.nextInt(2));
            x2.add(random.nextInt(2));
            if (x1.get(i)==0 && x2.get(i)==0)
                expectedResult.add(0);
            else
                expectedResult.add(1);
```

```
        }
    }
//sumator
    public double calculateSum(int x1, int x2, double[]weights){
        double sum = x1*weights[0]+x2*weights[1];
        return sum;
    }

//funkcja aktywacji
    public int perceptronActivation(double sum){
        if(sum>0)
            return 1;
        else
            return 0;
    }

//aktualizacja wartości wag
//dodanie do wagi - wsp uczenia * błąd(zawsze 1) * wartość wejściowa
    public void adjustWeights(double[] weights, double error, int i){
        weights[0] += learningRate * error *  x1.get(i);
        weights[1] += learningRate * error *  x2.get(i);
    }

/*Obliczana jest suma iloczynów danych wejściowych i ich wag.
Wywoływana jest następnie funkcja aktywacyjna perceptronu, która w
zależności od obliczonej sumy zwraca 0 lub 1
Jeśli rzeczywisty wynik nie zgadza się z wartością zwracaną przez funkcję
aktywacji perceptronu, następuje
aktualizacja wag i inkrementyowana jest zmiana licznika wag.*/
    public void learn(){
        int changeInWeights=1;
        while(changeInWeights != 0){
            changeInWeights=0;
            for(int i = 0; i< dataAmount; i++){
                double calculatedSum = calculateSum(x1.get(i), x2.get(i),
weights);
                int result = perceptronActivation(calculatedSum);
                int error = expectedResult.get(i)-result;
                if(error!=0) {
                    adjustWeights(weights, error, i);
                    changeInWeights++;
                }
            }
            weightChangesAmount +=changeInWeights;
        }
    }

/*test polega na wyliczeniu sumy i aktywacji funkcji perceptronu, podobnie
jak w uczeniu.
Podczas sumowania iloczynów wykorzystywane są wagi otrzymane w procesie
uczenia.
Jeśli funkcja perceptronu zwróci wartość inną niż właściwa, liczone jest to
jako błąd. W przeciwnym wypadku jako poprawny wynik.
Test oblicza procentowy stosunek poprawnych odpowiedzi do błędnych.*/
    public int test(){
        int correct=0;
        int wrong=0;
        for(int i = 0; i< dataAmount; i++){
            double calculatedSum = calculateSum(x1.get(i), x2.get(i),
weights);
            int result = perceptronActivation(calculatedSum);
```

```
            int error = expectedResult.get(i)-result;
            if(error==0)
                correct++;
            else
                wrong++;
        }
        if(wrong==0)
            return 100;
        else
            return (correct / wrong) * 100;
    }

    public double getLearningRate() {
        return learningRate;
    }

    public void setLearningRate(Double learningRate) {
        this.learningRate = learningRate;
    }

    public int getDataAmount() {
        return dataAmount;
    }

    public void setDataAmount(Integer dataAmount) {
        this.dataAmount = dataAmount;
    }
}

//Funkcja main
import java.io.IOException;

public class Main {
    public static void main(String[] args) throws IOException{
        Perceptron perceptron = new Perceptron();
        perceptron.setDataAmount(1000);
        perceptron.setLearningRate(0.005);
        perceptron.generateData();
        perceptron.saveGeneratedData();
        perceptron.learn();
        perceptron.saveLearningResults();
        perceptron.saveTestResults(perceptron.test());
    }
}
```