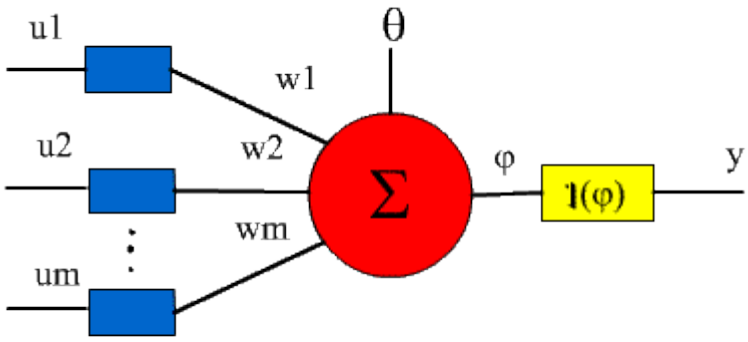
Rafał Szyszka

Sprawozdanie 1 – Budowa i działanie perceptronu.

1.Wstęp.

Budowa perceptronu:



Perceptron prosty zbudowany jest jedynie z warstwy wejściowej i warstwy wyjściowej. Ponieważ nie istnieją połączenia pomiędzy elementami warstwy wyjściowej, każdy z tych elementów może być traktowany niezależnie jako osobna sieć o m+1 wejściach i jednym wyjściu.

Algorytm uczenia polega na dobraniu wag początkowych losowo, podaniu wektora uczącego, obliczeniu wartości wyjściowej perceptronu, porównaniu wartości wyjściowej z oczekiwaną a następnie dokonania modyfikacji według zależności danej wzorami: $w_{i+1} = w_i + (d_i - y_i) * lr * x_i$

2.Cel ćwiczenia.

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

3. Przebieg ćwiczenia.

a) Implementacja sztucznego neuronu.

b) Wygenerowanie danych uczących i testujących dla funkcji logicznej AND dwóch zmiennych.

c) uczenie perceptronu

d) testowanie perceptronu

4. Opracowanie wyników.

a) Losowe wagi początkowe a ilość iteracji uczenia. Learn rate 0.1

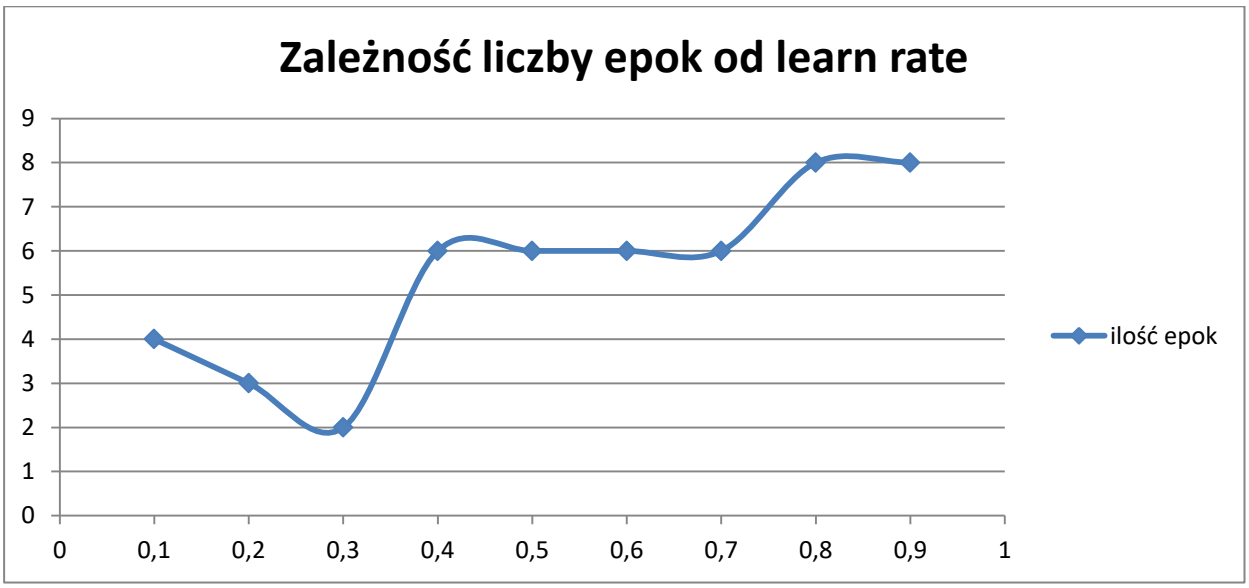
Lp.	wagi początkowe	ilość epok
1	0.12015262896618861	2
	0.23780034806802242	
	0.25472953745674487	
2	0.9296562618641151	7
	0.9131129376265016	
	0.8235758313903179	
3	0.3797022294489887	9
	0.9045790562184314	
	0.10697300133287235	
4	0.5525917580944772	5
	0.20586320859358243	
	0.4063476424365402	
5	0.10136696752631413	3
	0.44870285605283666	
	0.1651100873973096	

b)Stałe wagi początkowe a ilość iteracji uczenia. Learn rate 0.1

Lp.	wagi początkowe	ilość epok
1	0.1	5
2	0.2	3
3	0.3	4
4	0.4	4
5	0.5	4
6	0.6	5
7	0.7	5
8	0.8	6
9	0.9	7

c)zależność liczby epok od learn rate przy stałej wadze początkowej 0.5

Lp.	learn rate	ilość epok
1	0.1	4
2	0.2	3
3	0.3	2
4	0.4	6
5	0.5	6
6	0.6	6
7	0.7	6
8	0.8	8
9	0.9	8

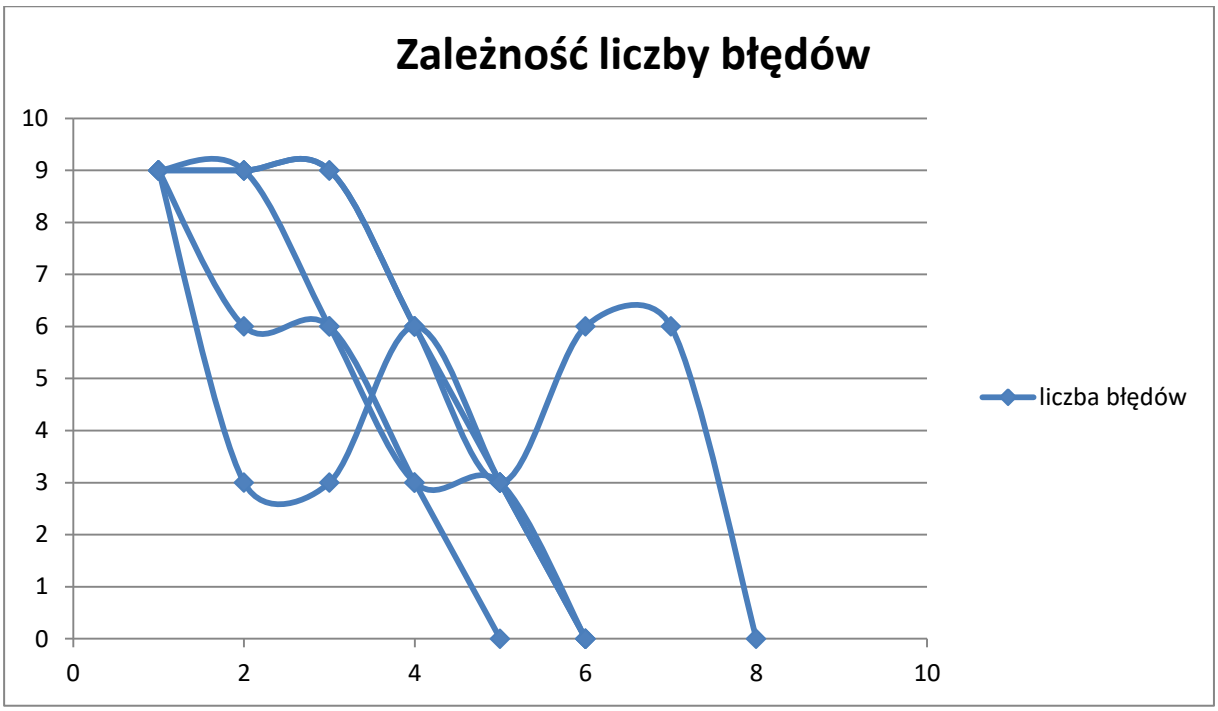


Wykonałem także eksperymenty dla innych wag początkowych do punktu c) i za każdym razem wychodziło że learn rate na poziomie 0.3 ma najmniejszą

potrzebną liczbę epok do poprawnego wyuczenia perceptronu, a wartości 0.8 oraz 0.9 najwyższą.

d) numer epoki a ilość popełnionych błędów

nr epoki	**liczba błędów**
1	9
2	9
3	9
4	6
5	3
6	0
1	9
2	9
3	9
4	6
5	3
6	6
7	6
8	0
1	9
2	9
3	6
4	3
5	3
6	0
1	9
2	6
3	6
4	3
5	0
1	9
2	3
3	3
4	6
5	3
6	0



5. Wnioski

Na podstawie otrzymanych wyników można stwierdzić, że najlepsza efektywność uczenia perceptronu jest przy zróżnicowanych wagach początkowych, ale odpowiednio niskich w przedziale <0.1;0.3> oraz przy learn rate na poziomie 0.3. Oznacza to, że współczynnik uczenia nie powinien być ani za mały ani za duży oraz nasuwa się wniosek, że lepiej różnicować początkowe wagi. Nauczenie perceptronu prostej funkcji logicznej było stosunkowo bardzo proste, bo w najlepszym przypadku otrzymywaliśmy dobre wyniki już w drugiej iteracji, a najgorszym w dziewiątej.

6. Źródła

http://www.cs.put.poznan.pl/rklaus/assn/percep.htm

http://home.agh.edu.pl/~horzyk/lectures/biocyb/BIOCYB-SieciNeuronowe.pdf

https://pl.wikipedia.org/wiki/Perceptron

http://edu.pjwstk.edu.pl/wyklady/nai/scb/wyklad3/w3.htm

7. Listing kodu

```java
/*
 * Copyright (C) 2017 Szysz
 */
package main;

/**
 *
 * @author Szysz
 */
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        int n = 3;
        Perceptron perc = new Perceptron(n);

        perc.showWages();

        int inputSize = 4;
        double learnRate = 0.1;

        int bias = 1;

        int x1[] = {1, 0, 1, 0};
        int x2[] = {0, 0, 1, 1};

        int d[] = {0, 0, 1, 0};

        double error;
        do{
            error = 0;
            for (int j = 0; j < inputSize; j++) {
                error+=perc.learn(new int[]{bias, x1[j], x2[j]}, d[j], learnRate);
            }
            System.out.println("error: "+error);
        }while(error!=0);


        System.out.println(perc.process(new int[]{bias, 1, 0}));
        System.out.println(perc.process(new int[]{bias, 0, 1}));
        System.out.println(perc.process(new int[]{bias, 1, 0}));
        System.out.println(perc.process(new int[]{bias, 1, 1}));
    }

}
```

```java
*/
public class Perceptron {

    private final int rozmiar;
    private final double wage[];

    public Perceptron(int n) {
        rozmiar = n;
        wage = new double[n];
        Random rand = new Random();
        for (int i = 0; i < n; i++) {
            wage[i] = rand.nextDouble();
            // wage[i] = 0.1;
        }

    }

    private int activate(double y) {
        if (y < 0) {
            return 0;
        } else {
            return 1;
        }
    }

    public int process(int[] x) {
        double y = 0;
        for (int i = 0; i < rozmiar; i++) {
            y += x[i] * wage[i];
        }

        return activate(y);
    }

    public double learn(int x1[], double d, double learnRate) {
        double y = process(x1);
        double error = 0;
        for (int i = 0; i < rozmiar; i++) {
            wage[i] += (d - y) * learnRate * x1[i];
            error+=Math.abs(d-y);
        }
        return error;
    }

    public void showWages(){
        for (int i = 0 ; i<rozmiar; i++)
            System.out.println("w"+i+": "+wage[i]);
    }

}
```