Konrad Tabiś

Laboratorium nr 1

Sprawozdanie - Podstawy Sztucznej Inteligencji – Scenariusz 1

Temat ćwiczenia: 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.

2.Zadania do wykonania

1. Implementacja sztucznego neuronu
2. Wygenerowanie danych uczących dla funkcji logicznej AND
3. Uczenie perceptronu dla różnej liczby danych uczących, różnych współczynników uczenia
4. Testowanie perceptronu

3.Sprawozdanie:

Syntetyczny opis budowy oraz wykorzystanego algorytmu uczenia

Źródło algorytmu wg którego zaimplementowałem sztuczny neuron:
http://edu.pjwstk.edu.pl/wyklady/nai/scb/wyklad3/w3.htm

Na początku inicjujemy wagi losowo, a następnie porównujemy wagi z wynikiem oczekiwanym i modyfikujemy je przy pomocy wzorów:

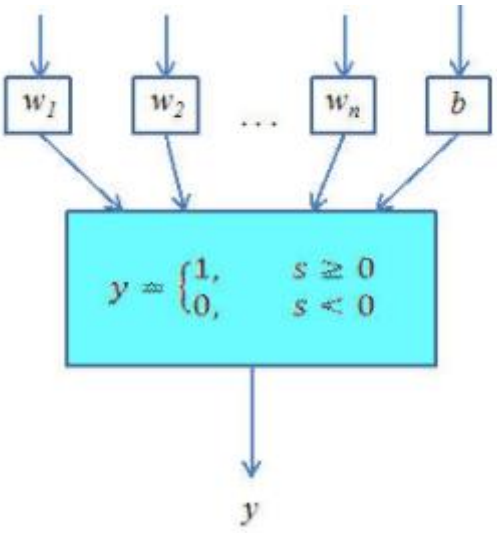
w_1+=n*(d-y) *x_1

w_2+=n*(d-y) *x_2

b +=n*(d-y)

gdzie w_1, w_2 to wagi, n jest niewielkim współczynnikiem uczenia(n>0), d-oczekiwaną odpowiedzią, y-odpowiedzią neuronu, b-progiem wzbudzenia perceptronu, a x_1, x_2 wartościami wejściowymi.

Funkcja na podstawie której obliczana jest wartość wyjściowa to unipolarna funkcja progowa, zwraca tylko dwie wartości co dobrze obrazuje problem rozwiązywany przez perceptron.



Zestawienie otrzymanych danych

Do uczenia perceptronu wybrałem funkcję logiczną AND.

Współczynnik uczenia	Epoka	Ilość punktów	Ilość błędów
0,05	1	4	3
	2	8	3
	3	12	3
	4	16	3
	5	20	3
	6	24	2
	7	28	1

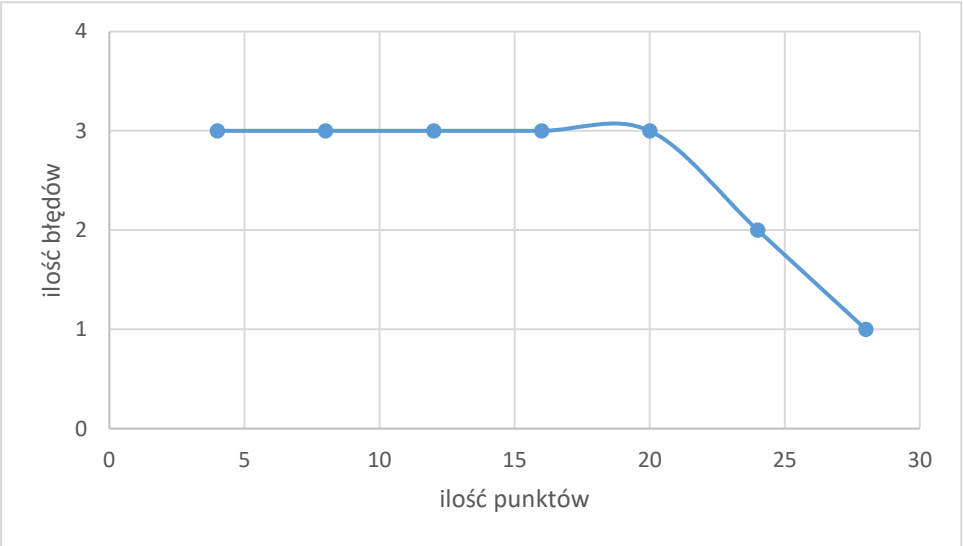
Współczynnik uczenia	Epoka	Ilość punktów	Ilość błędów
0,1	1	4	3
	2	8	3
	3	12	2
	4	16	0
	5	20	0
	6	24	0
	7	28	0

Współczynnik uczenia	Epoka	Ilość punktów	Ilość błędów
0,15	1	4	1
	2	8	1
	3	12	0
	4	16	0
	5	20	0
	6	24	0
	7	28	0

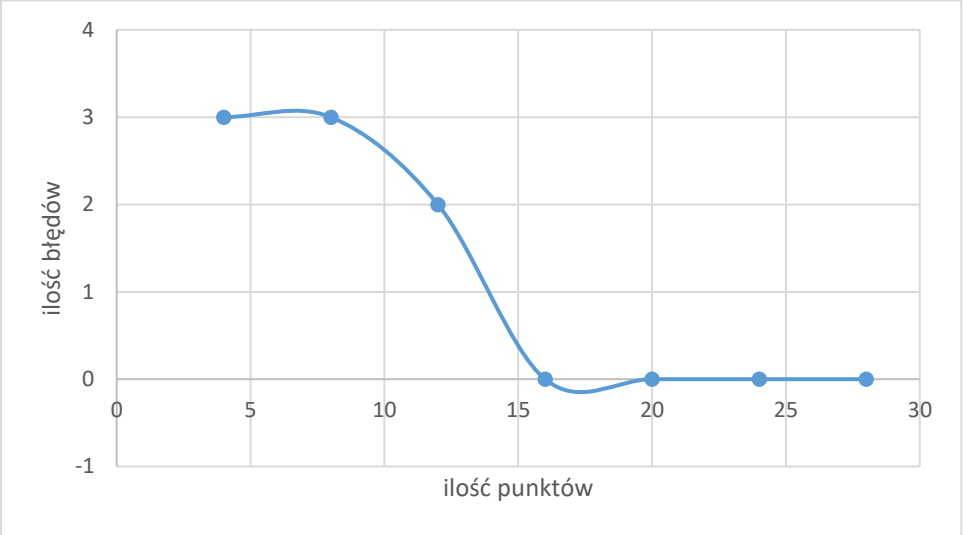
Współczynnik uczenia	Epoka	Ilość punktów	Ilość błędów
0,3	1	4	2
	2	8	0
	3	12	0
	4	16	0
	5	20	0
	6	24	0
	7	28	0

Wykresy ilości błędów od ilości punktów w zależności od współczynnika uczenia:

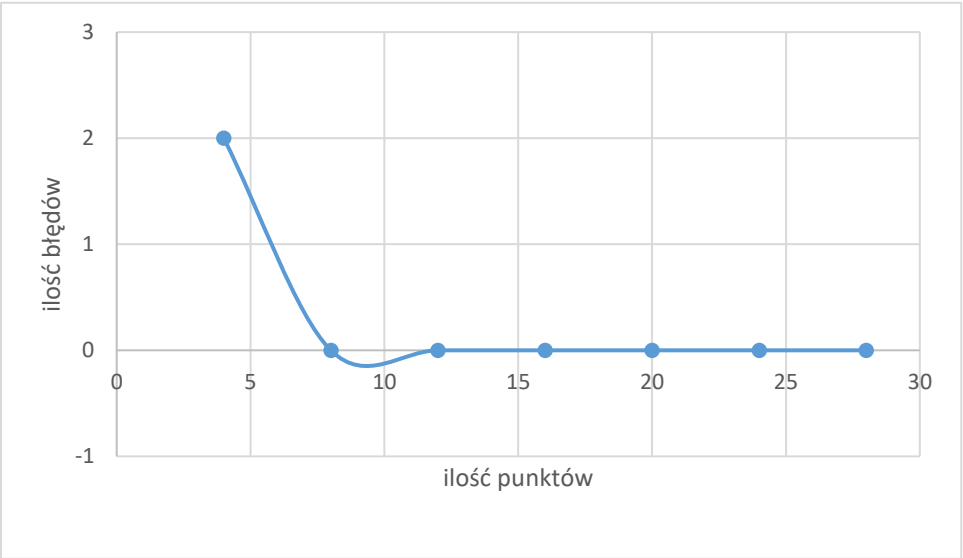
- Dla współczynnika uczenia równego 0,05



- Dla współczynnika uczenia równego 0,1



- Dla współczynnika uczenia równego 0,3



Wnioski:

Wszystkie powyższe wyniki są dla losowych wag. Na podstawie tych wyników możemy zauważyć, że bardzo ważnymi zmiennymi dla szybkości działania programu są współczynnik uczenia oraz ilość epok(punktów). Ważne jest także dobranie odpowiedniej funkcji progowej. Dla tego konkretnego przypadku funkcja progowa zwracająca tylko dwie wartości okazała się wystarczająca, a przy tym bardzo szybko i nieskomplikowana. Niestety nie każdy problem możemy rozwiązać tak prostą funkcją. Jak widać na powyższych wykresach dobranie odpowiedniego współczynnika uczenia bardzo wpływa na szybkość poprawnego działania programu. Przy współczynniku tym równym 0,05 nawet po 7 epokach pojawiały się błędy, natomiast dla 0,15 już po 3 epokach był bezbłędny.

Dla pojedynczego perceptronu proces uczenia nie jest skomplikowany, niestety nie jesteśmy tez nim w stanie dużo osiągnąć, bo już przy bramce logicznej XOR jesteśmy bezradni.

Listing kodu

```
import java.util.Random;

/**
 * Created by Dell on 13.10.2017.
 */
//klasa perceptronu
public class Perceptron {
    Random generator =new Random();

    //wagi poczatkowe
    //ustawione randomowo
    public double
weight0=generator.nextDouble(),weight1=generator.nextDouble()
,weight2=generator.nextDouble();

    //funkcja progowa
    public int perceptronOut (double perceptronOut)
    {
        if(perceptronOut<0)return 0;
        else return 1;
    }

    //funkcja sumująca iloczyny danych wejściowych i odpowiedających im wag
    public int process ( int x0, int x1, int x2 ) {
        double y_p = x0 * weight0 + x1 * weight1 + x2 * weight2;
        return perceptronOut( y_p );
    }

    //algorytm uczenia perceptronu
    public void learn_perceptron(int x0, int x1, int x2, double y, double
learning_factor)
    {
        double percOut=process(x0,x1,x2);
        weight0 += ( y - percOut ) * learning_factor * x0;
        weight1 += ( y - percOut ) * learning_factor * x1;
        weight2 += ( y - percOut ) * learning_factor * x2;
    }
}
```

```
/**
 * Created by Dell on 14.10.2017.
 */
public class Main {
    public static void main(String[] args) {
        Perceptron perceptron = new Perceptron();

        int bias = 1;
        int ilosc_epok=8;
        double learning_factor = 0.15;

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

        //tabela wynikowa dla AND
        int[] y = {0, 0, 0, 1};

        //pętle uczące perceptron z wyświetlaniem wyników po każdej epoce
```

```
        System.out.println("Wyniki uczenia perceptonu funkcji AND\n");
        for(int i =0;i<ilosc_epok;i++){
            for(int j=0;j<4;j++)
            {

perceptron.learn_perceptron(bias,x1[j],x2[j],y[j],learning_factor);
            }
            //Wyświetlanie wyników po każdej epoce
            System.out.println("Epoka numer: "+ i + "\n");
            System.out.println("x1="+x1[0]+" x2="+x2[0]+" y="+y[0]+" wynik
= "+ perceptron.process(bias,x1[0],x2[0])+"\n");
            System.out.println("x1="+x1[1]+" x2="+x2[1]+" y="+y[1]+" wynik
= "+ perceptron.process(bias,x1[1],x2[1])+"\n");
            System.out.println("x1="+x1[2]+" x2="+x2[2]+" y="+y[2]+" wynik
= "+ perceptron.process(bias,x1[2],x2[2])+"\n");
            System.out.println("x1="+x1[3]+" x2="+x2[3]+" y="+y[3]+" wynik
= "+ perceptron.process(bias,x1[3],x2[3])+"\n\n");
        }

        //Wyświetlenie wag końcowych
        System.out.println("WAGI:");
        System.out.println("W0 = " + perceptron.weight0);
        System.out.println("W1 = " + perceptron.weight1);
        System.out.println("W2 = " + perceptron.weight2);
    }
    }
```