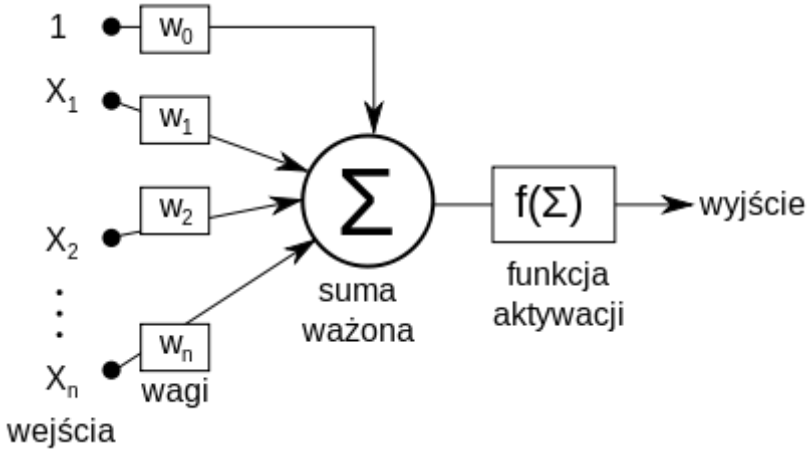
Daniel Wenecki

Podstawy Sztucznej Inteligencji

Sprawozdanie z projektu nr 1

1) Celem ćwiczenia było zapoznanie się z budową perceptronu i mechanizmem jego działania. W tym celu stworzyliśmy perceptron realizujący funkcje logiczną dwóch zmiennych.

2) Model perceptronu realizowany w tym ćwiczeniu:



Zaimplementowany przeze mnie model perceptronu nosi nazwę Neuron McCullocha-Pittsa. Jest to jeden z matematycznych modeli neuronu. Posiada wiele wejść i jedno wyjście. Każdemu z wejść przyporządkowana jest liczba rzeczywista - waga wejścia. Wartość na wyjściu neuronu obliczana jest w następujący sposób:

1. obliczana jest suma iloczynów wartości xi podanych na wejścia i wag wi wejść:

$$s = w_0 + \sum_{i=1}^{n} x_i w_i$$

2. na wyjście podawana jest wartość funkcji aktywacji f(s) dla obliczonej sumy

Funkcja aktywacji dla naszego perceptronu to funkcja progowa unipolarnama i ma ona postać:

$$y(x) = \begin{cases} 0 \; dla \; x < a \\ 1 \; dla \; x \geq a \end{cases}$$

3) Otrzymane wyniki:

Dla funkcji logicznej AND:

Początkowo ustawiłem krok uczenia na 0.2, następnie ustawiając różne wagi, uzyskiwałem poprawne wagi po różnych ilościach powtórzeń:

Wagi początkowe	-1	-0.5	0	0.5	1
Ilość wykonanych pętli	10	8	5	2	3

Następnie ustawiłem wszystkie wagi na 1 i modyfikowałem krok uczenia(learningRate).

Krok uczenia	0.01	0.05	0.1	0.2	0.5
Ilość wykonanych pętli	56	11	6	3	1

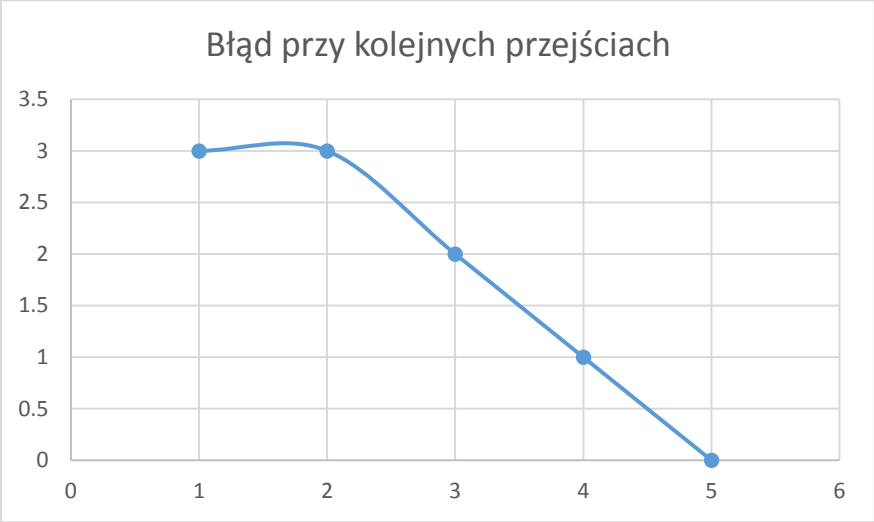
Przy zmianie wprowadzonych danych liczących, algorytm mógł zakończyć pracę w momencie, w którym nie dawał jeszcze poprawnych wyników.

Dla funkcji logicznej XOR:

Niezależnie od ustawionych parametrów i ilości wykonanych pętli, algorytm nie mógł znaleźć odpowiednich wag, co potwierdza że pojedynczy perceptron nie jest w stanie przedstawić funkcji logicznej XOR(jest to za to możliwe dla sieci neuronowych).

4) Analiza i dyskusja błędów uczenia

Dla wag początkowych 0.1 i krokiem nauczania równym 0.1 uzyskałem następujące błędy:



Jak widać błąd malał, wraz z kolejnymi epokami, aż osiągnął wystarczająco dla nas niską wartość. Podobnie jest przy innych ustawieniach wag. Jest to związane z wzorem zgodnie z którym nasze wagi się modyfikują. Uważać należy jednak na krok uczenia, ponieważ ustawienie zbyt dużej wartości może spowodować zwiększenie błędu, zamiast jego redukcję. (Wartość oczekiwana wag będzie znacznie przeskoczona przez algorytm).

5) Wnioski

Przy pomocy perceptronu możemy w prosty sposób przewidywać wyniki funkcji, o wprowadzonych przez nas punktach. Szybkość uczenia się perceptronu zależy od wartości takich jak krok uczenia, wagi początkowej i punktów na podstawie których perceptron się uczy. Dane do nauki powinny być odpowiednio dobrane, najlepiej żeby punkty wprowadzone były od siebie oddalone równomiernie, w przeciwnym razie perceptron może dawać wyniki ze znacznie większymi błędami. Wagi początkowe często ustawia się na losowe i daje to dość zadowalające wyniki. Jeśli chodzi o krok uczenia to powinien on być odpowiednio dobrany, ponieważ wartość za mała znacznie wydłuży poszukiwanie odpowiednich wag, za to wartość za duża może sprawić, że zmiany wag będą za duże. Będą one przestrzeliwały wartości optymalnych wag, przez co czas szukania również się wydłuży, lub nie będziemy w ogóle w stanie znaleźć zadowalającego nas przybliżenia.

Kod źródłowy:

```
public class Perceptron {
    double learningRate = 1;
    double bias = 1;
    int nrOfInstances = 200;
    double w1,w2,wb;
    public Perceptron (double x1[],double x2[],double y[])
    {
        Random r = new Random();
        w1 = 1;
        w2 = 1;
        wb = 1;
        learn(x1,x2,y);
    }
    public void learn(double x1[],double x2[],double y1[])
    {
        double global_error= 0;
        double local_error=0;
        double yloc =0;
        for(int i=0;i<nrOfInstances;i++)
        {
            global_error = 0;
            for(int j=0;j<4;j++) {
                yloc=treshhold(calculate(x1[j],x2[j]));
                local_error = (y1[j] - yloc);
                w1 = w1 + learningRate*local_error*x1[j];
                w2 = w2 + learningRate*local_error*x2[j];
                wb = wb + learningRate*local_error;
                global_error = global_error + local_error*local_error;
            }
            if(global_error < 0.001) break;
            System.out.println("Petla nr: " + (i + 1));
        }
    }
    public double treshhold(double sum)
    {
        if(sum>0) return 1;
        else return 0;
    }
    public double calculate(double x1,double x2)
    {
        return x1*w1 + x2*w2 + bias*wb;
```

```
    }
}


public class Main {

    public static void main(String[] args) {
        System.out.println("Dzialanie perceptonu: ");
        double x1[] = {0,0,1,1};
        double x2[] = {0,1,0,1};
        double y1[] = {0,0,0,1};
        double y2[] = {0,1,1,1};
        double y3[] = {0,1,1,0};
        //double bias[] = {1,1,1,1};
        //doPerceptonThnings(x1,x2,y2);
        Perceptron p1 = new Perceptron(x1,x2,y1);

        System.out.println("Wynik z 0 i 0  " +
p1.treshhold(p1.calculate(0,0)));
        System.out.println("Wynik z 0 i 1  " +
p1.treshhold(p1.calculate(0,1)));
        System.out.println("Wynik z 1 i 0  " +
p1.treshhold(p1.calculate(1,0)));
        System.out.println("Wynik z 1 i 1  " +
p1.treshhold(p1.calculate(1,1)));
    }

}
```

Źródła:

https://pl.wikipedia.org/wiki/Neuron_McCullocha-Pittsa