Podstawy Sztucznej Inteligencji – Laboratorium nr 1

Wykonał: Paweł Nowak

Temat ćwiczenia: Budowa i działanie perceptronu

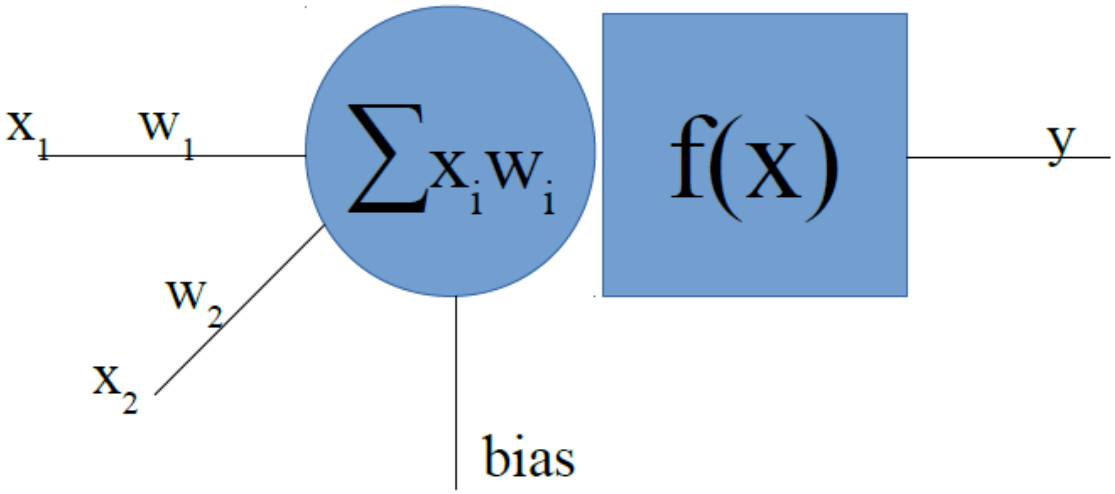
1. Cel ćwiczenia:

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

2. Realizacja ćwiczenia:

Wykorzystane materiały: http://edu.pjwstk.edu.pl/wyklady/nai/scb/rW2.html

Schemat perceptronu:



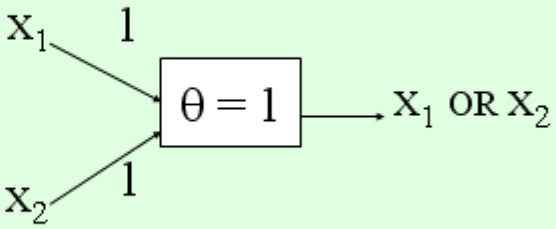
x – wejście
w – waga wejścia
bias – waga wejścia pobudzenia perceptronu
$\sum x_i w_i$ _ suma iloczynów wag z wartościami wejścia
f(x) – funkcja aktywacji progowa unipolarna dla a = 0
y – wyjście

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

Jeżeli *s* jest większa lub równa 0, to ustawiamy wyjście *y=1*, zaś w przeciwnym przypadku ustawiamy *y=-1 (funkcja bipolarna)*. Perceptron opisany jest więc jednoznacznie przez zbiór wag $w_1,...,w_n$ oraz wartość odchylenia *b*.

Jako funkcję logiczną wybrałem bramkę logiczną OR, która przedstawia się w następujący sposób:

x_1	x_2	$x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1



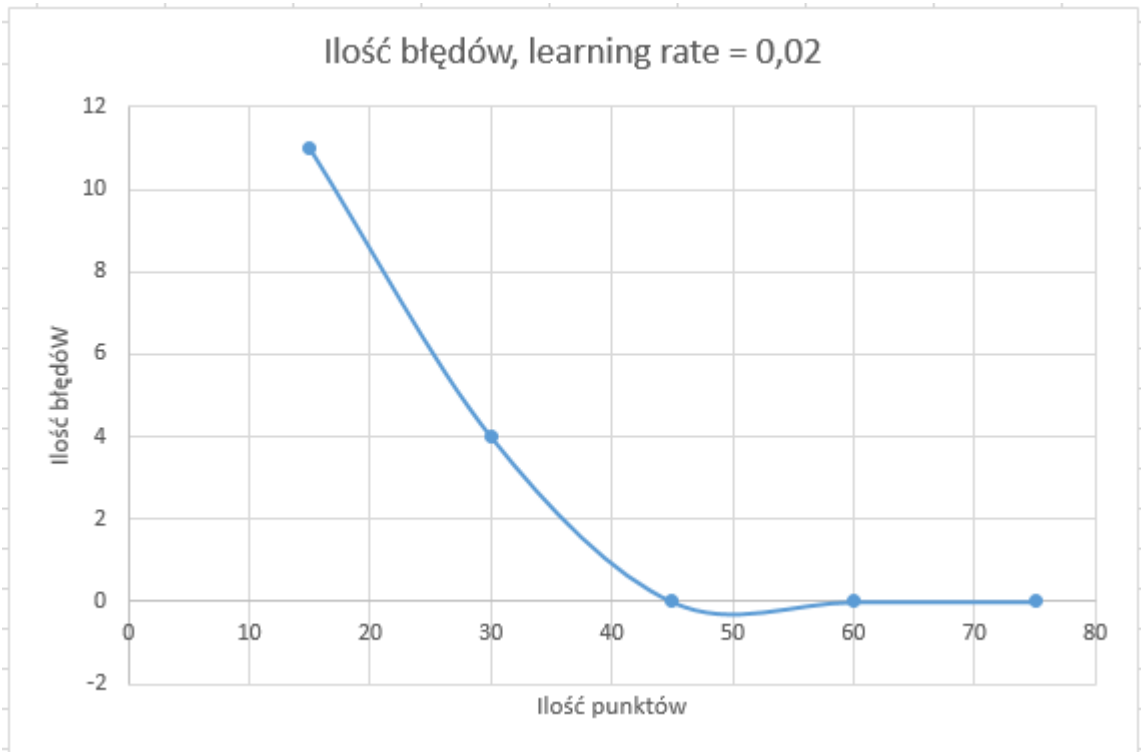
Jak widać na wejściu są liczby x1 i x2, których wartości są generowane z zakresu <0,1> jako liczby całkowite. Na podstawie pary dobranych liczb określany jest wynik (odpowiedź).

Statystyki nauki perceptronu:

Learning Rate	Epoka	Ilość punktów	Ilość błędów
0,01	1	15	11
	2	30	5
	3	45	5
	4	60	5
	5	75	5
0,02	1	15	11
	2	30	4
	3	45	0
	4	60	0
	5	75	0

0,03	1	15	5
	2	30	1
	3	45	0
	4	60	0
	5	75	0
0,1	1	15	0
	2	30	0
	3	45	0
	4	60	0
	5	75	0

Jak widać w wynikach zamieszczonych w tabeli wartość learning rate ogromnie wpływa na ilość błędów popełnianych przez perceptron. Im mniejsza wartość tej zmiennej tym perceptron częściej się myli. Za to jak widać przy wartości learning rate 0,1 już przy 15 punktach nie pomylił się ani razu.



Widać zauważalną różnice w obu wykresach, następne wykresy niewiele wpłynęłyby na przedstawienie wyników.

3. Wnioski:

Perceptron szybko i na małej ilości danych jest w stanie szybko nauczyć się rozwiązywać zadany mu problem. Wraz ze wzrostem wartości learning rate nauka przebiega szybciej oraz sprawniej. Im więcej wprowadzimy danych testowych tym perceptron będzie sprawniej odpowiadał. Zagadnienie bramki logicznej OR jest w gruncie rzeczy problemem prostym, nie wymagającym dużej złożoności. Model perceptronu nie jest skomplikowany, jest dość prosty do implementacji. Jednocześnie jest to odzwierciedlenie najprostszej sieci neuronowej, która może składać się z jednego lub wielu niezależnych węzłów. Do rozwiązania zadanego problemu wystarczył jeden węzeł. Wykorzystany język programowania przeze mnie to Python.

Kod programu:

Perceptron.py

```
import random
# bramka logiczna OR
class Perceptron:

    def __init__(self):
        self.learning_speed = 0.01

        self.weight = [0,0,0]
        self.weight[0] = random.uniform(-1,1)
        self.weight[1] = random.uniform(-1,1)
        self.weight[2] = random.uniform(-1,1)

        self.bias = 1

    def display(self, x1, x2):
        if x1 == 1:
            print ("TRUE or", end='')
        else:
            print ("FALSE or", end='')
        if x2 == 1:
            print (" TRUE = ", end='')
        else:
            print (" FALSE = ", end='')

        self.guess(x1, x2)
        if self.result == 1:
            print ("TRUE")
        else:
            print ("FALSE")

    def guess(self, x1, x2):
        self.result = x1*self.weight[0] + x2*self.weight[1] + self.bias*self.weight[2]

        if self.result >= 0:
            self.result = 1
        else:
            self.result = -1
        return self.result

    def train(self, x1, x2):
        if x1 == -1 and x2 == -1:
            self.desired = -1
        else:
            self.desired = 1

        error = self.desired - self.guess(x1,x2)
        self.weight[0] += self.learning_speed * error * x1
        self.weight[1] += self.learning_speed * error * x2
        self.weight[2] += self.learning_speed * error * self.bias
```