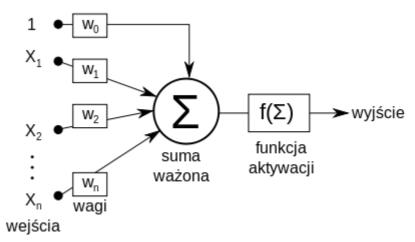
Tomasz Tomala Podstawy Sztucznej Inteligencji Sprawozdanie z projektu nr 2

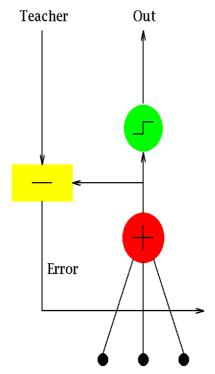
Celem ćwiczenia buło poznanie budowy i działania jednowarstwowych sieci neuronowych oraz uczenie rozpoznawania wielkości liter.

### 1) Syntetyczny opis budowy wykorzystanego algorytmu uczenia:

Aby wykonać ćwiczenie stworzyłem dwie jednowarstwowe sieci – każdą z wykorzystaniem innego algorytmu. Pierwsza z nich korzysta z modelu perceptronu McCullocha-Pittsa, druga z kolei wykorzystuje neuron typu Adaline.

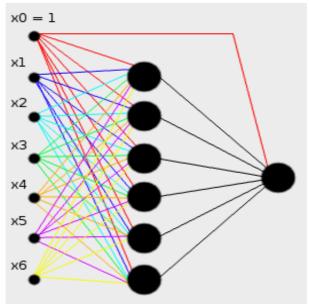


Ilustracja 1: Model perceptronu McCullocha-Pittsa



Ilustracja 2: Model neuronu Adaline

Obie sieci składają się z siedmiu neuronów. Sześć pierwszych stanowi pierwszą warstwę i przesyła sygnały wyjściowe do siódmego wyjściowego neuronu. Każdy z neuronów otrzymuje po siedem sygnałów wejściowych. Sposób w jaki wszystko zostało połączone przedstawiam na poniższej grafice:



Ilustracja 3: Graficzne zilustrowanie połaczeń w sieci

Do nauki perceptronów oraz adaline wykorzystałem algorytm Widrow-Hoffa.

Do budowy perceptronu wykorzystałem podany na wykładzie model McCullocha-Pittsa. Zaimplementowana przeze mnie klasa Perceptron składa się z trzech metod: active, process oraz learn.

Metoda *active* wykorzystuje unipolarną funkcję progową która zwraca wynik 0 lub 1 dla podanego argumentu.

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

Metoda *process* sumuje iloczyn sygnałów wejściowych i odpowiadających im wag. Uruchamia metodę activate z otrzymaną sumą iloczynów jako parametrem, oraz zwraca wynik tej metody.

$$\sum_{i=1}^m w_i x_i$$

Metoda *learn* wywołuje metodę process dla otrzymanych wejść jako parametrów, po czym na podstawie otrzymanego wyniku modyfikuje wszystkie wagi dla odpowiednich wejść.

$$W_i = W_i + (y - y') * x_i * \alpha$$

Budowa adaline jest bardzo podobna do budowy perceptronu. Różni się tylko tym, że modyfikowanie wag wykonuje się z pominięciem funkcji aktywacji. Zaimplementowana przeze mnie klasa Adaline składa się z czterech metod: active, process, learn oraz test.

Metoda *active* wykorzystuje unipolarną funkcję progową która zwraca wynik -1 lub 1 dla podanego argumentu.

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 1 & \text{otherwise} \end{cases}$$

Metoda *process* sumuje iloczyn sygnałów wejściowych i odpowiadających im wag, oraz zwraca ten wynik.

$$\sum_{i=1}^m w_i x_i$$

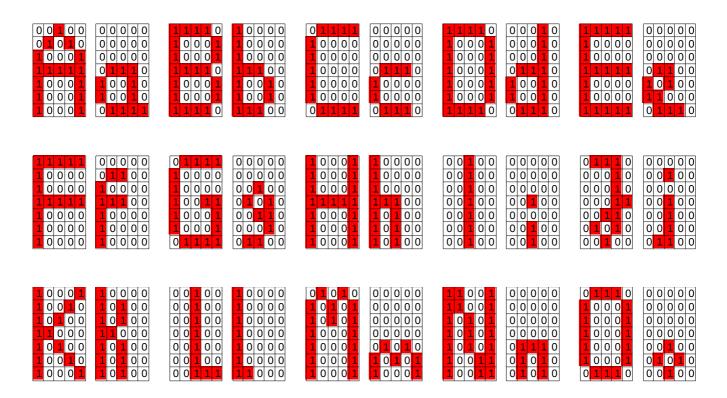
Metoda *learn* wywołuje metodę process dla otrzymanych wejść jako parametrów, po czym na podstawie otrzymanego wyniku modyfikuje wszystkie wagi dla odpowiednich wejść.

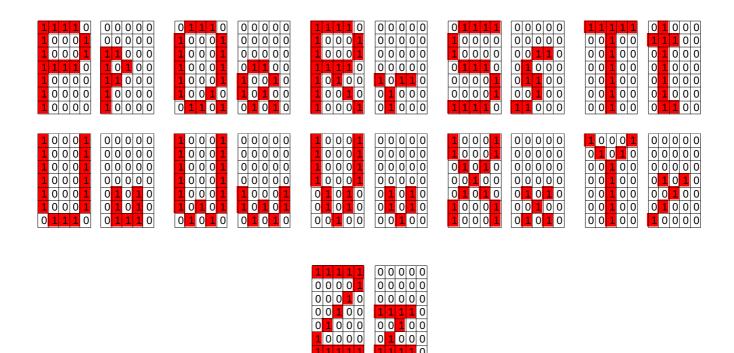
$$W_i = W_i + (y - y') * x_i * \alpha$$

Metoda *test* uruchamia metodę active z wynikiem metody process jako parametrem.

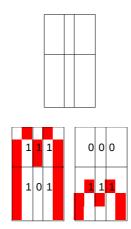
## 2) Zestawienie otrzymanych wyników:

Jako dane uczące i testujące wykorzystałem własnoręcznie stworzone litery alfabetu. Są one przedstawione jako dwuwymiarowa tablica o rozmiarach 7 x 5. Przedstawiam je poniżej:



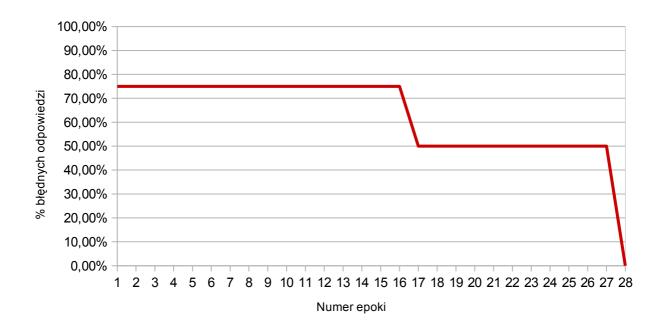


Obszar 7 x 5 podzieliłem na 6 obszarów. Każdy obszar to jeden sygnał wejściowy do neuronów. Jeżeli w danym obszarze pojawił się choćby jeden piksel z litery to obszar zwracał sygnał 1, w przeciwnym wypadku zwracał sygnał 0. Poniżej sposób podziału tablicy na obszary, oraz przykład dla dużej i małej litery "M":



Do uczenia perceptronu wybrałem funkcje logiczna AND.

Na początku zacząłem uczyć perceptron w taki sposób, że przesyłałem mu wszystkie kombinacje danych wejściowych taką samą ilość razy. Wszystkie wagi początkowe ustawiłem na wartość równą 0.5. Z kolei współczynnik uczenia się ustawiłem na wartość 0.01. Proces uczenia wymagał 28 powtórzeń. Poniżej prezentacja na wykresie:



Następnie zacząłem eksperymentować z różnymi wagami początkowymi. W każdym kolejnym teście sprawdzałem jak długo zajmie nauka w zależności od wag początkowych. Warunki testu identyczne jak do tego powyżej. Poniżej wyniki:

Waga początkowa	Ilość powtórzeń		
0	5		
0.05	3		
0.08	5		
0.13	8		
0.17	10		
0.2	11		

Kolejnym krokiem było sprawdzenie jak będzie wyglądał proces uczenia się dla różnych współczynników uczenia się. Test taki sam jak powyżej, jednak wagi początkowe są równe 0.5. Poniżej przedstawiam wyniki:

Współczynnik uczenia	Ilość powtórzeń
0.01	28
0.1	3
0.5	5
1.2	1
5	5

Następnie sprawdziłem jak będą wyglądać wyniki dla różnej ilości danych uczących. Tym razem jednak ustaliłem stałą wartość dla wszystkich wag początkowych równą 0.5 a współczynnik uczenia wynosił 0.1. Poniżej przedstawiam wyniki dla różnych kombinacji danych wejściowych i ilości ich powtórzeń:

<b>x1</b>	x2	Ilość powtórzeń	wynik	x1	x2	Ilość powtórzeń	wynik
0	0	1		0	0	2	
0	1	1	błędny	0	1	2	la la duas
1	0	1		1	0	2	błędny
1	1	1		1	1	2	

x1	x2	Ilość powtórzeń	wynik	x1	<b>x2</b>	Ilość powtórzeń	wynik
0	0	4		0	0	4	
0	1	3	poprawny	0	1	3	
1	0	2		1	0	2	poprawny
1	1	1		1	1	0	

x1	x2	Ilość powtórzeń	wynik
0	0	6	
0	1	3	
1	0	1	poprawny
1	1	0	

# 3) Analiza i dyskusja błędów uczenia i testowania opracowanego perceptronu w zależności od wartości współczynnika uczenia oraz liczby danych uczących:

Uczenie się perceptronu zależy od trzech czynników: wag początkowych, współczynnika uczenia oraz od danych uczących. Jak widać na powyższych danych, wagi początkowe wpływają na szybkość uczenia się jednak współczynnik uczenia wpływa na to o wiele bardziej, ponieważ wagi i tak ulegają ciągłej modyfikacji. Jednak w głównej mierze to od danych uczących zależy czy perceptron będzie w stanie się nauczyć czy też nie, co wyraźnie widać na ostatnim teście. W przypadku funkcji logicznej AND nie musiałem podawać perceptronowi do nauczenia się danych wejściowych [1,1] ani razu, jednak musiałem go nauczyć pozostałych trzech kombinacji odpowiednią ilość razy, w przeciwnym wypadku wynik był błędny.

#### 4) Sformułowanie wniosków:

Dla pojedynczego perceptronu proces uczenia jest bardzo prosty. Jesteśmy w stanie nauczyć go prostej funkcji logicznej w zaledwie kilku krokach. Całość opiera się o wykonywanie tych samych operacji dla różnych danych wejściowych i odpowiednim modyfikowaniu wag.

W całym procesie uczenia się najważniejsze są dane uczące. Należy dobrać je w odpowiedni sposób i powtórzyć uczenie się na ich podstawie odpowiednią ilość razy. W przeciwnym wypadku wyniki będą niepoprawne. Wagi początkowe również wpływają na proces uczenia, jednak mają one raczej marginalne znaczenie. O wiele większe znaczenie ma współczynnik uczenia. Jeżeli wybierzemy za mały, to perceptronowi zajmie bardzo długo, aż uzyskamy odpowiednie wyniki. Z kolei za duża wartość tego współczynnika może sprawić, że nie będziemy w stanie w ogóle trafić w odpowiednie wagi, lub też zajmie to bardzo dużo czasu. Dlatego właśnie musimy dokonać optymalnego wyboru współczynnika uczenia.

#### 5) Listing całego kodu

```
package perceptron;
import java.util.Random;
public class Perceptron {
    private int noi; //ilość wejść
    private double[] w; //wagi

    public Perceptron ( int numbers_of_inputs ) {
        noi = numbers_of_inputs;
        w = new double[noi];

        for ( int i = 0; i < noi; i++)
            w[i] = new Random().nextDouble(); //wagi początkowe są losowane
}

//funkcja aktywująca
private int active ( double y_p ) {
        return y_p < 0 ? 0 : l;
}

//sumator
public int process ( int[] x ) {
        double y_p = 0;

        for ( int i = 0; i < noi; i++)
            y_p += x[i] * w[i];

        return active( y_p);
}

//uczenie
public void learn ( int[] x, double y, double lr ) {
        double y_p = process( x );

        for ( int i = 0; i < noi; i++)
            w[i] += ( y - y_p ) * lr * x[i];
}

public double getW ( int i ) {
        return w[i];
}
</pre>
```

# Bibliografia:

https://pl.wikipedia.org/wiki/Neuron\_McCullocha-Pittsa https://en.wikipedia.org/wiki/ADALINE https://en.wikipedia.org/wiki/Perceptron