Podstawy Sztucznej Inteligencji – Laboratorium nr 2

Wykonał: Paweł Nowak

Temat ćwiczenia: Budowa i działanie sieci jednowarstwowej

1. Cel ćwiczenia

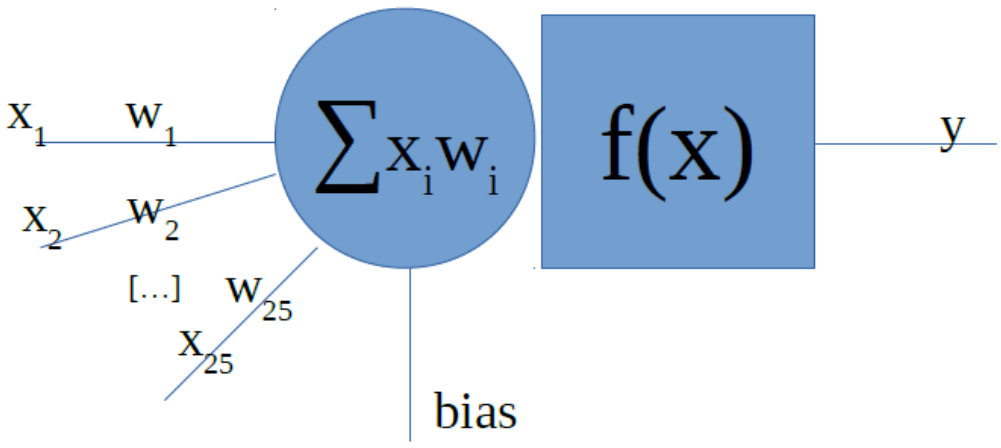
Celem ćwiczenia jest poznanie budowy i działania jednowarstwowych sieci neuronowych oraz uczenie rozpoznawania wielkości liter

2. Realizacja ćwiczenia

Wybrany przeze mnie język programowania do realizacji laboratorium to **Python**.

Do realizacji ćwiczenia stworzyłem dwie sieci jednowarstwowe wykorzystując z modelu neuronu McCullocha – Pittsa, natomiast druga implementacja to Adaline.

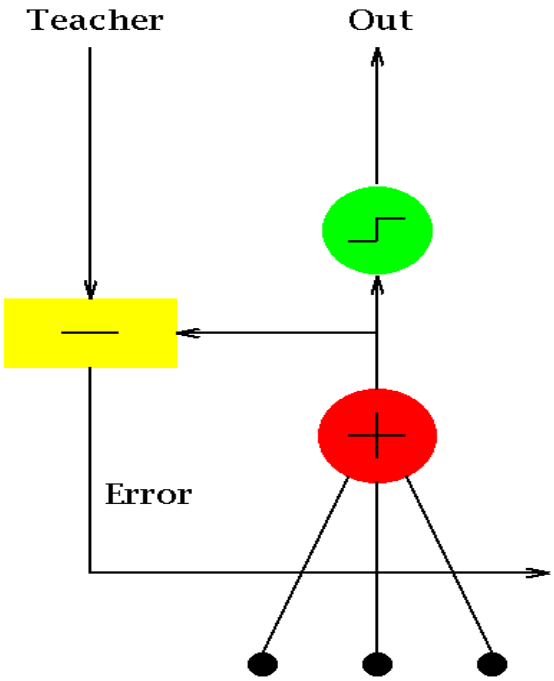
Graficzne przedstawienie modelu McCullocha – Pittsa:



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

Algorytm uczenia:

Graficzne przedstawienie modelu Adaline:



Nauka sieci opiera się na algorytmie Widrowa – Hoffa.

Progowa funkcja aktywacji dla modelu McCullocha-Pittsa wygląda następująco:

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

Natomiast dla modelu Adaline:

$$y(x) = \begin{cases} -1 & dla \quad x < a \\ 1 & dla \quad x \geq a \end{cases}$$

Metoda sumowania sygnałów wejściowych wygląda następująco:

$$y = \sum w_i x_i$$

gdzie w_i – waga, x_i – sygnał wejściowy.

Metoda uczenia opisana jest w następujący sposób:

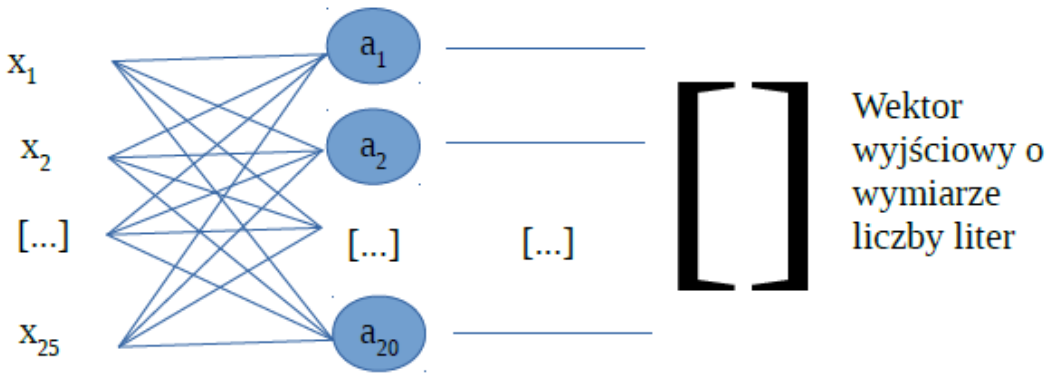
$$W_i = w_i + (y - y') * learning\_rate * x_i$$

Gdzie:

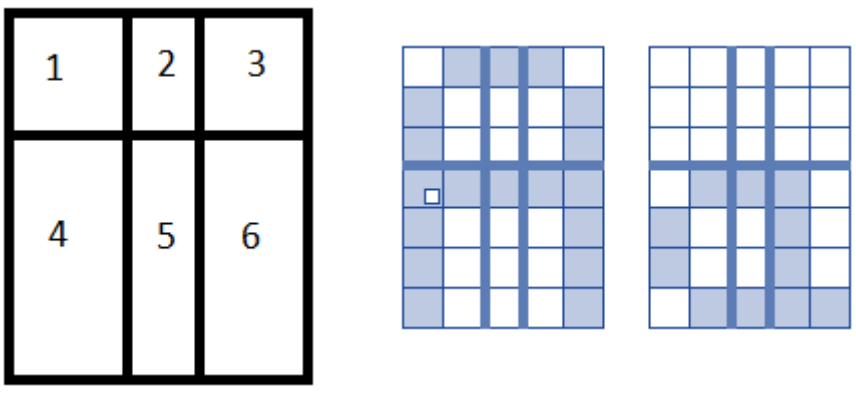
- w – dana waga,
- y –wielkość litery (0, 1),
- y' – wielkość litery „nauczona",
- learning_rate – współczynnik szybkości nauki,
- x_i – sygnał wejściowy

Budowa Adaline i Perceptronu niewiele się różni tzn. głównie różnica jest przy funkcji aktywacji oraz Adaline ma dodatkowo metodę test, która zwraca funkcję aktywacji z parametrem sumowania wag (jako argument).

Sieci składają się z siedmiu neuronów, 6 neuronów wysyła sygnały wyjściowe do neuronu nr 7 jako do neuronu wyjściowego. Każdy neuron dostaje 7 sygnałów wejściowych (pierwszy to bias, pozostałe to obliczenia czy w danym sektorze znajduje się fragment litery). Schemat połączeń:

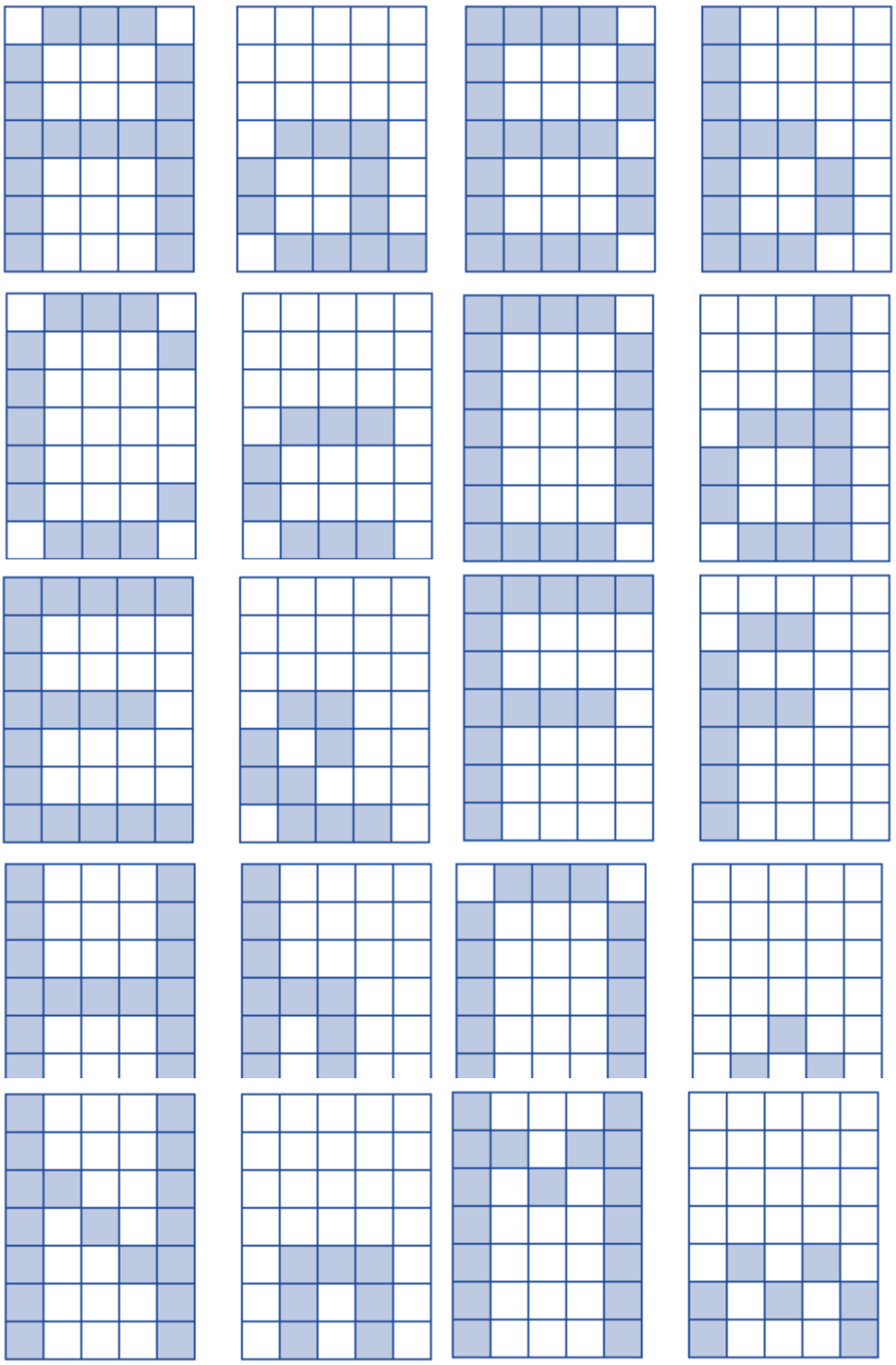


Przygotowany przeze mnie zestaw danych do nauki sieci to macierze 5x7 - 10 dużych i 10 małych liter alfabetu. Każda litera została podzielona na ponumerowane obszary wg. schematu:



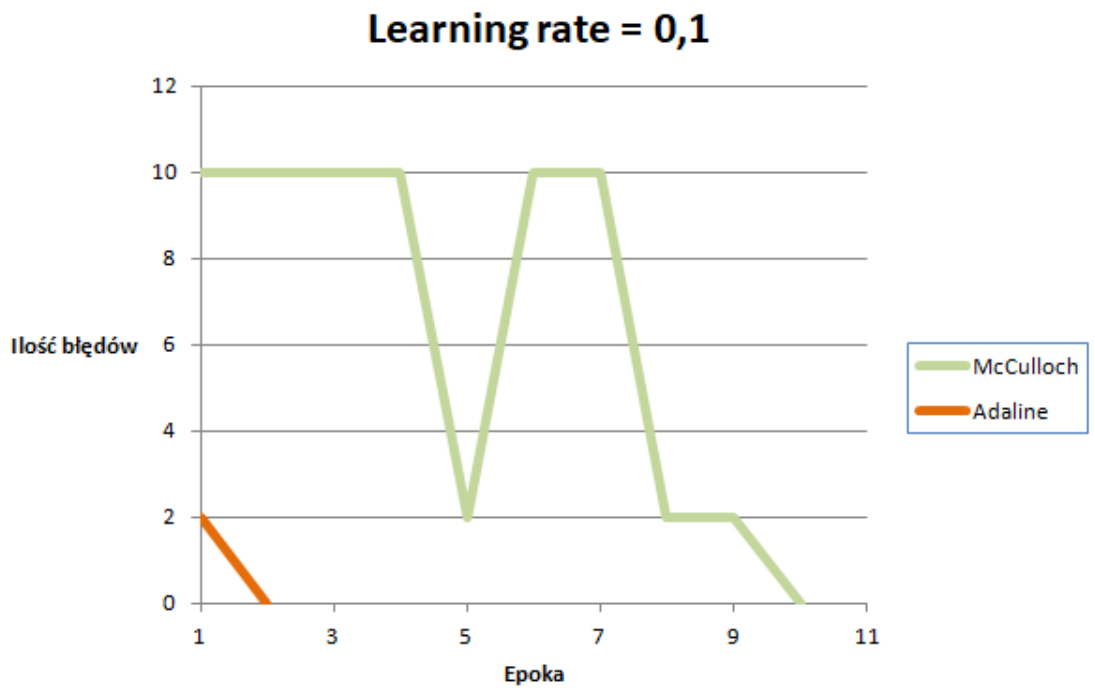
Pogrubionymi liniami został zaznaczony rzeczywisty podział na przykładzie liczby „A” i „a”.

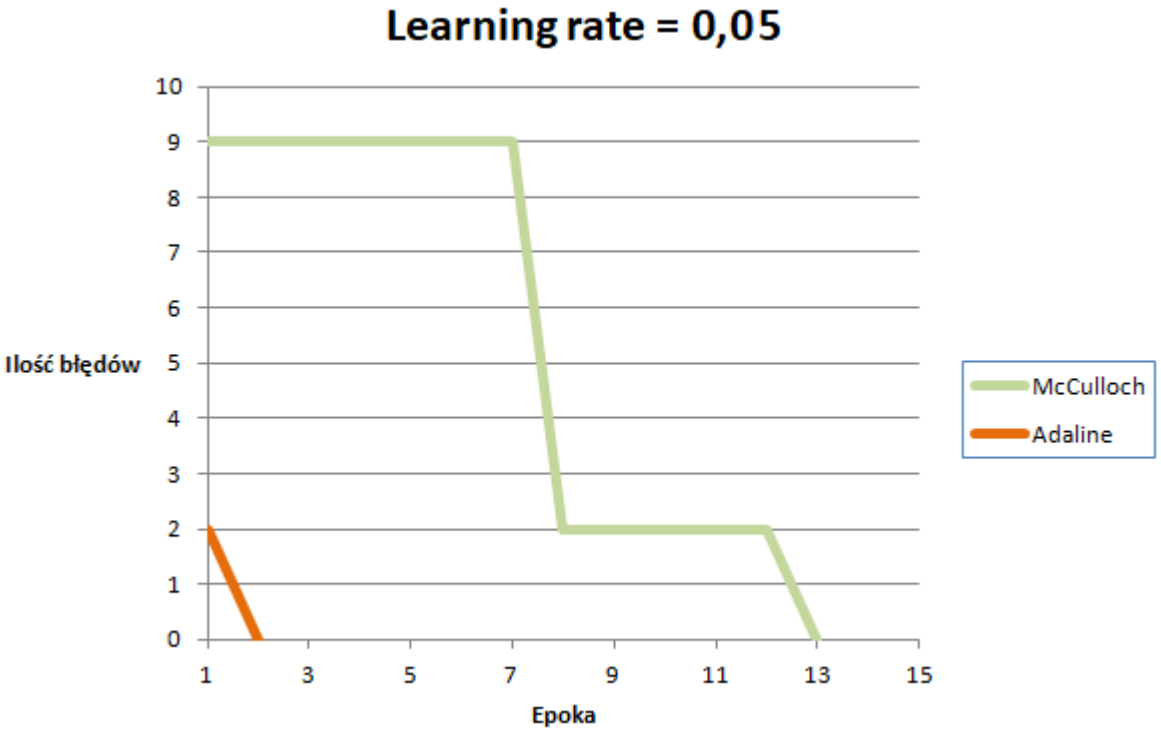
Schematy liter:

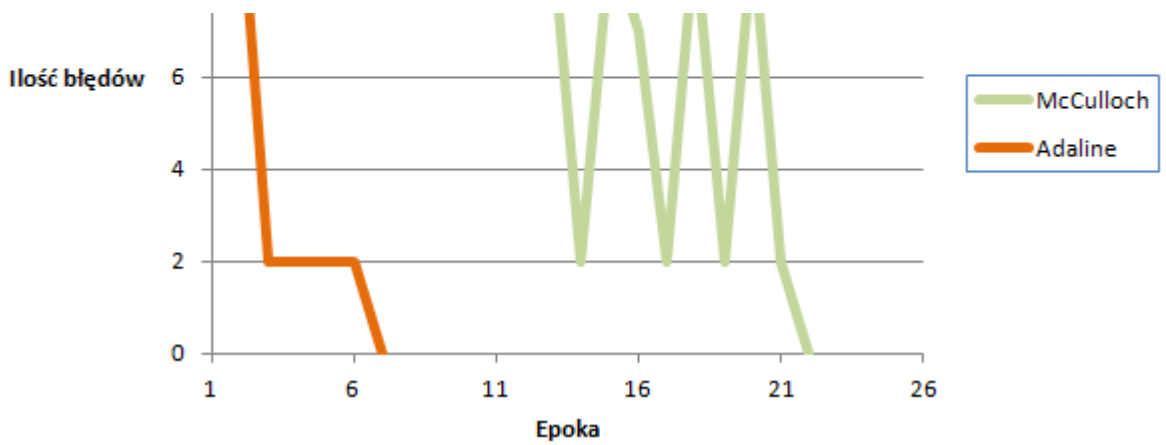


Niebieskie obszary zostały opisane wartością 1, natomiast białe (puste) wartością 0.

3. Przedstawienie wyników na wykresach:







Learning rate = 0,001

4. Analiza wyników

Jak wynika z wykresów powyżej współczynnik uczenia znacząco wpływa na szybkość uczenia się. Im większa wartość learning_rate tym szybciej przebiega nauka. Model Adaline jest również dużo bardziej wydajny (od 2 do 10-krotnie) od modelu McCullocha-Pittsa.

Dla wysokich wartości learning rate Adaline potrzebował zaledwie 1-2 epoki do nauki przy każdej próbie. Przy czym dla tej samej wartości McCulloch-Pitts potrzebował około 10 epok, aby nauczyć się prawidłowo rozróżniać litery. Dla wartości 0,01 Adaline potrzebował 7 epok do nauczenia się, a McCulloch aż 22. Następnie dla 0,001 Adaline 36, McCulloch – 84, dla 0,0005 -> Adaline – 125 epok, McCulloch – 279.

5. Wnioski

Model Adaline jest znacznie wydajniejszy i efektywniej się uczy. Zestawiając różne współczynniki uczenia zawsze był w stanie nauczyć się znacznie szybciej od modelu McCullocha-Pittsa. Możliwość zmniejszania wartości wag w modelu Adaline oraz funkcja aktywacji dla wartości -1 i 1 zwiększa efektywność nauki neuronu. Wprowadzane dane testowe również mają ogromne znaczenie (empirycznie przekonałem się, że budowa litery „L" oraz „b" jest dla modelów prawie tym samym przez co nie można było rozpoznać, czy wprowadzona litera jest duża, czy też nie). W przypadku niepoprawnego wprowadzenia danych możliwe jest zapętlenie programu w nieskończoność.

6. Listing kodu

```
"""Main
Perceptron
class """
            import random
            class Perceptron:
                "Klasa perceptronu"
                def __init__(self, inputs):
                    self.inputs = inputs
                    self.weights = []
                    for i in range(0, inputs):
                        self.weights.append(random.uniform(0, 1))
                @staticmethod
                def activation(y_p):
                    "funkcja aktywacji"
                    if y_p < 0:
                        return 0
                    return 1
                def sum(self, vector):
                    "Sumowanie wag"
```

```
                    y_p = 0
                    for i in range(0, self.inputs):
                        y_p += vector[i] * self.weights[i]
                    return self.activation(y_p)
                def learn(self, vector, y_value, learning_rate):
                    "Uczenie"
                    y_p = self.sum(vector)
                    for i in range(0, self.inputs):
                        self.weights[i] += (y_value - y_p) * learning_rate * vector[i]
```

```
"Losowanie
typu
double"
            from random import uniform
            class Adaline:
                """Adaline"""
                def __init__(self, inputs):
                    "Konstruktor"
                    self.weights = []
                    self.inputs = inputs
                    for i in range(0, inputs):
                        self.weights.append(uniform(0, 1))
                @staticmethod
                def activation(y_p):
                    "Funkcja aktywacji"
                    if y_p <= 0:
                        return -1
                    return 1
                def sum(self, vector):
                    "Sumowanie"
                    y_p = 0
                    for i in range(0, self.inputs):
                        y_p += vector[i] * self.weights[i]
                    return y_p
                def learn(self, vector, y_value, learning_rate):
                    "Metoda uczenia"
                    y_p = self.sum(vector)
                    for i in range(0, self.inputs):
                        self.weights[i] += (y_value - y_p) * learning_rate * vector[i]
                def test(self, vector):
                    "Metoda testująca"
                    return self.activation(self.sum(vector))
```

```
from
Perceptron
import
Perceptron
from Letters import getLetter
### funkcje
def learn(perceptrons, inputs, learning_rate, i, j):
    vector = getLetter(i, j)
    vector_p = []
    vector_p.append(1)   # bias
    for k in range(0, inputs-1):
        perceptrons[k].learn(vector, i, learning_rate)
        vector_p.append(perceptrons[k].sum(vector))

    perceptrons[inputs-1].learn(vector_p, i, learning_rate)
def test(perceptrons, letter_number, inputs):
    results = []
    for i in range(0, letter_number * 2):
        results.append(0)
    vector = []
    vector_p = []
    vector_p.append(1)   # bias
    for i in range(0, inputs-1):
        vector_p.append(0)
    for i in range(0, 2):
        for j in range(0, letter_number):
            vector = getLetter(i, j)
            for k in range(0, inputs-1):
                vector_p[k+1] = perceptrons[k].sum(vector)
            results[i * letter_number + j] = perceptrons[inputs -
1].sum(vector_p)
    return results
 ### funkcje
ERAS = 0
INPUTS = 7
LETTER_NUMBER = 10
LEARNING_RATE = 0.00001
PERCEPTRONS = []
Y = []
RESULT = []
## wypelnienie danymi
for i in range(0, INPUTS):
    PERCEPTRONS.append(Perceptron(INPUTS))
for i in range(0, LETTER_NUMBER):
    Y.append(0)
for i in range(LETTER_NUMBER, LETTER_NUMBER*2):
```

```
            Y.append(1)
        for i in range(0, LETTER_NUMBER*2):
            RESULT.append(0)
        ## Uczenie
        while Y != RESULT:
            # i = 0 -> duze litery, 1 -> male litery
            for i in range(0, 2):
                for j in range(0, LETTER_NUMBER):
                    learn(PERCEPTRONS, INPUTS, LEARNING_RATE, i, j)
            RESULT = test(PERCEPTRONS, LETTER_NUMBER, INPUTS)
            ERAS += 1

        print("Ilosc krokow do nauki:", ERAS)
```

```
"Import
Adaline"
        from Adaline import Adaline
        "Import liter"
        from Letters import getLetter
        def learn(adaline, inputs, learning_rate, i, j):
            "Funkcja uczenia"
            vector = getLetter(i, j)
            vector_formatter(vector)
            vector_p = []
            vector_p.append(1) #bias
            if i==0:
                letter_size = -1
            else:
                letter_size = 1
            for i in range(0, inputs - 1):
                adaline[i].learn(vector, letter_size, learning_rate)
                vector_p.append(adaline[i].test(vector))

            adaline[inputs - 1].learn(vector_p, letter_size, learning_rate)
        def test(adaline, letter_number, inputs):
            "Funkcja testujaca"
            result = []
            for i in range(0, letter_number * 2):
                result.append(0)
            vector_p = []
            vector_p.append(1) #bias
```

```
    for i in range(0, inputs - 1):
        vector_p.append(0)
    for i in range(0, 2):
        for j in range(0, letter_number):
            vector = getLetter(i, j)
            vector_formatter(vector)
            for k in range(0, inputs - 1):
                vector_p[k + 1] = adaline[k].test(vector)
            result[i * letter_number + j] = adaline[inputs - 1].test(vector_p)
    return result
def vector_formatter(vector):
    "Zamiana 0 na -1 w wektorze"
    for i in range(0, INPUTS):
        if vector[i] == 0:
            vector[i] = -1
### dane wejsciowe
INPUTS = 7
LETTTER_NUMBER = 10
ERAS = 0
LEARNING_RATE = 0.00001
ADALINES = []
CORRECT = []  # -1 --> duża litera, 1 --> mała litera
RESULT = []  # dane wyjsciowe (wyniki testowania)
for i in range(0, INPUTS):
    ADALINES.append(Adaline(INPUTS))
for i in range(0, LETTTER_NUMBER):
    CORRECT.append(-1)
for i in range(LETTTER_NUMBER, 2 * LETTTER_NUMBER):
    CORRECT.append(1)
for i in range(0, 2 * LETTTER_NUMBER):
    RESULT.append(0)
while RESULT != CORRECT:
    for i in range(0, 2):
        for j in range(0, LETTTER_NUMBER):
            learn(ADALINES, INPUTS, LEARNING_RATE, i, j)
    RESULT = test(ADALINES, LETTTER_NUMBER, INPUTS)
    ERAS += 1
print("Ilość potrzebnych kroków do nauczenia:", ERAS)
```

```
letters
= [
            [
                    [[0,1,1,1,0], [1,0,0,0,1], [1,0,0,0,1], [1,1,1,1,1], [1,0,0,0,1],
          [1,0,0,0,1], [1,0,0,0,1]],   #A
                    [[1,1,1,1,0], [1,0,0,0,1], [1,0,0,0,1], [1,1,1,1,0], [1,0,0,0,1],
          [1,0,0,0,1], [1,1,1,1,0]],   #B
                    [[0,1,1,1,0], [1,0,0,0,1], [1,0,0,0,0], [1,0,0,0,0], [1,0,0,0,0],
          [1,0,0,0,1], [0,1,1,1,0]],   #C
                    [[1,1,1,1,0], [1,0,0,0,1], [1,0,0,0,1], [1,0,0,0,1], [1,0,0,0,1],
          [1,0,0,0,1], [1,1,1,1,0]],   #D
                    [[1,1,1,1,1], [1,0,0,0,0], [1,0,0,0,0], [1,1,1,1,0], [1,0,0,0,0],
          [1,0,0,0,0], [1,1,1,1,1]],   #E
                    [[1,1,1,1,1], [1,0,0,0,0], [1,0,0,0,0], [1,1,1,1,0], [1,0,0,0,0],
          [1,0,0,0,0], [1,0,0,0,0]],   #F
                    [[1,0,0,0,1], [1,0,0,0,1], [1,0,0,0,1], [1,1,1,1,1], [1,0,0,0,1],
          [1,0,0,0,1], [1,0,0,0,1]],   #H
                    [[0,1,1,1,0], [1,0,0,0,1], [1,0,0,0,1], [1,0,0,0,1], [1,0,0,0,1],
          [1,0,0,0,1], [0,1,1,1,0]],   #O
                    [[1,0,0,0,1], [1,1,0,1,1], [1,0,1,0,1], [1,0,0,0,1], [1,0,0,0,1],
          [1,0,0,0,1], [1,0,0,0,1]],   #M
                    [[1,0,0,0,1], [1,0,0,0,1], [1,1,0,0,1], [1,0,1,0,1], [1,0,0,1,1],
          [1,0,0,0,1], [1,0,0,0,1]]    #N
            ],
            [
                    [[0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0], [0,1,1,1,0], [1,0,0,1,0],
          [1,0,0,1,0], [0,1,1,1,1]],   #a
                    [[1,0,0,0,0], [1,0,0,0,0], [1,0,0,0,0], [1,1,1,0,0], [1,0,0,1,0],
          [1,0,0,1,0], [1,1,1,0,0]],   #b
                    [[0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0], [0,1,1,1,0], [1,0,0,0,0],
          [1,0,0,0,0], [0,1,1,1,0]],   #c
                    [[0,0,0,1,0], [0,0,0,1,0], [0,0,0,1,0], [0,1,1,1,0], [1,0,0,1,0],
          [1,0,0,1,0], [0,1,1,1,0]],   #d
                    [[0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0], [0,1,1,0,0], [1,0,1,0,0],
          [1,1,0,0,0], [0,1,1,1,0]],   #e
                    [[0,0,0,0,0], [0,1,1,0,0], [1,0,0,0,0], [1,1,1,0,0], [1,0,0,0,0],
          [1,0,0,0,0], [1,0,0,0,0]],   #f
                    [[1,0,0,0,0], [1,0,0,0,0], [1,0,0,0,0], [1,1,1,0,0], [1,0,1,0,0],
          [1,0,1,0,0], [1,0,1,0,0]],   #h
                    [[0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0], [0,0,1,0,0],
          [0,1,0,1,0], [0,0,1,0,0]],   #o
                    [[0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0], [0,1,0,1,0],
          [1,0,1,0,1], [1,0,0,0,1]],   #m
                    [[0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0], [0,1,1,1,0],
          [0,1,0,1,0], [0,1,0,1,0]]    #n
            ]
```

```
            ]
def getLetter(size, letter):
    "Zwraca wektor dla danej litery (podzial na 6 obszarow), przy sektorach i - kolumny, j - wiersze"
    result = []
    for i in range (0,7):
        result.append(0)
    result[0] = 1
    #obszar nr1 - lewy dolny rog
    for i in range (3, 7):
        for j in range(0,2):
            if letters[size][letter][i][j] == 1:
                result[4] = 1
    #obszar nr2 - srodek na dole
    for i in range (3, 7):
        for j in range(2, 3):
            if letters[size][letter][i][j] == 1:
                result[5] = 1

    #obszar nr3 - prawy dolny rog
    for i in range (3, 7):
        for j in range(3, 5):
            if letters[size][letter][i][j] == 1:
                result[6] = 1
    #obszar nr4 - lewy gorny rog
    for i in range (0,3):
        for j in range(0,2):
            if letters[size][letter][i][j] == 1:
                result[1] = 1
    #obszar nr5 - srodek na gorze
    for i in range (0,3):
        for j in range(2, 3):
            if letters[size][letter][i][j] == 1:
                result[2] = 1

    #obszar nr6 - prawy gorny rog
    for i in range (0,3):
        for j in range(3, 5):
            if letters[size][letter][i][j] == 1:
                result[3] = 1
    return result
```

7. Bibliografia

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