Mateusz Nogieć

Scenariusz 2-sprawozdanie

Celem projektu było poznanie budowy i działania jednowarstwowych sieci nieuronowych oraz uczenie rozpoznawania wielkości liter.

Opis budowy wykorzystanych sieci i algorytmów uczenia.

1. Perceptron- składający się z neuronów McCullocha-Pittsa, posiadających wiele wejść i jedno wyjście. Każdemu z wejść przyporządkowana jest liczba rzeczywista - waga wejścia. Wartość

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

   na wyjściu neuronu obliczana jest w następujący sposób
2. Adaline (ADAptive LInear Neuron)-swoją budową bardzo przypomina budowę perceptronu z tą różnicą, że porównuje się sygnał wejściowy z sygnałem na wyjściu liniowej części neuronu (sumatora), a więc błąd opisany jest wzorem: $\boldsymbol{\delta} = \boldsymbol{d} - \mathbf{s}$.

$$g(\mathbf{z}) = \begin{cases} 1 & \text{if } \mathbf{z} \geq \theta \\ -1 & \text{otherwise.} \end{cases}$$

Opis danych uczących oraz testujących.

W pliku z danymi uczącymi znajduje się 10 małych i 10 dużych liter. Wektor każdej litery kończy oczekiwany output. Jedna litera to matryca zer i jedynek 5x7.

Plik testujący składa się z 3 małych i 3 dużych liter. Oczekiwany output na końcu wektora.

```
0,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1
1,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,1,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,1,1,1,0,1
0,1,1,1,0,1,0,0,0,1,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,1,0,1,1,1,0,1
1,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,1,1,1,0,1
1,1,1,1,1,1,0,0,0,0,1,0,0,0,0,1,1,1,1,1,0,1,0,0,0,0,1,0,0,0,0,1,1,1,1,1
1,1,1,1,1,1,0,0,0,0,1,0,0,0,0,1,1,1,1,1,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1
0,1,1,1,0,1,0,0,0,1,1,0,0,0,0,1,0,1,1,1,1,0,0,0,1,1,0,0,0,1,0,1,1,1,0,1
1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,1,1,1,1,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1
0,1,1,1,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,1,1,1,0,1
1,1,1,1,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,0,0,0,1,0,1,1,1,0,1
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,1,0,0,1,0,1,0,0,1,0,1,1,1,1,1,0
0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,1,1,1,1,0,0,0,1,1,1,1,1,1,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,0,0,0,1,0,0,0,0,1,1,1,1,1,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,1,1,1,1,1,1,0,0,0,1,1,1,1,1,0
0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,0,0,0,1,1,1,1,1,1,1,0,0,0,0,1,1,1,1,0
0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,0,0,0,1,1,1,1,1,1,1,0,0,0,0,1,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,0,0,0,1,1,1,1,1,1,1,0,0,0,0,1,1,1,1,0
0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,0,1
0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,1,0
```

Wektor litery

Oczekiwany output

Dla każdej sieci wykonałem uczenie dla następujących **współczynników uczenia**: 0.01, 0.1, 0.5, 1.

Poniżej prezentuje otrzymane wyniki.

Perceptron:

lRate 0.01

```
Input: 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 1; 1; 1; 0; 0; 0; 1; 0; 0; 1; 0; 0; 0; 1; 1; 1; 1; 1;  Output: 0; Desired output: 0;  Error: 0;
Input: 1; 1; 1; 1; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1;  Output: 1; Desired output: 1;  Error: 0;
Input: 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 1; 1; 1; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1;  Output: 0; Desired output: 0;  Error: 0;
Input: 1; 1; 1; 1; 1; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 1; 1; 1; 1; 1; 0; 0; 0; 1; 0; 0; 0; 0; 1; 1; 1; 1; 1; 1;  Output: 1; Desired output: 1;  Error: 0;
Input: 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 1; 1; 1; 1; 1; 0; 0; 0; 1; 1; 0; 0; 0; 1; 1; 1; 1; 1; 1;  Output: 0; Desired output: 0;  Error: 0;
Input: 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 1; 1; 1; 1;  Output: 1; Desired output: 1;  Error: 0;
Total Mean Square Error: 0.0
```

0.1

```
Input: 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 1; 1; 1; 0; 0; 0; 1; 0; 0; 1; 0; 0; 0; 1; 1; 1; 1; 1;  Output: 0; Desired output: 0;  Error: 0;
Input: 1; 1; 1; 1; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1;  Output: 1; Desired output: 1;  Error: 0;
Input: 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 1; 1; 1; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1;  Output: 0; Desired output: 0;  Error: 0;
Input: 1; 1; 1; 1; 1; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 1; 1; 1; 1; 1; 0; 0; 0; 1; 0; 0; 0; 0; 1; 1; 1; 1; 1; 1;  Output: 1; Desired output: 1;  Error: 0;
Input: 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 1; 1; 1; 1; 1; 0; 0; 0; 1; 1; 0; 0; 0; 1; 1; 1; 1; 1; 1;  Output: 0; Desired output: 0;  Error: 0;
Input: 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 1; 1; 1; 1;  Output: 1; Desired output: 1;  Error: 0;
Total Mean Square Error: 0.0
```

0.5

```
Input: 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 1; 1; 1; 0; 0; 0; 1; 0; 0; 1; 0; 0; 0; 1; 1; 1; 1; 1;  Output: 0; Desired output: 0;  Error: 0;
Input: 1; 1; 1; 1; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1;  Output: 1; Desired output: 1;  Error: 0;
Input: 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 1; 1; 1; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1;  Output: 0; Desired output: 0;  Error: 0;
Input: 1; 1; 1; 1; 1; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 1; 1; 1; 1; 1; 0; 0; 0; 1; 0; 0; 0; 0; 1; 1; 1; 1; 1; 1;  Output: 1; Desired output: 1;  Error: 0;
Input: 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 1; 1; 1; 1; 1; 0; 0; 0; 1; 1; 0; 0; 0; 1; 1; 1; 1; 1; 1;  Output: 0; Desired output: 0;  Error: 0;
Input: 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 1; 1; 1; 1;  Output: 1; Desired output: 1;  Error: 0;
Total Mean Square Error: 0.0
```

1

```
Input: 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 1; 1; 1; 0; 0; 0; 1; 0; 0; 1; 0; 0; 0; 1; 1; 1; 1; 1;  Output: 0; Desired output: 0;  Error: 0;
Input: 1; 1; 1; 1; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1;  Output: 1; Desired output: 1;  Error: 0;
Input: 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 1; 1; 1; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1;  Output: 0; Desired output: 0;  Error: 0;
Input: 1; 1; 1; 1; 1; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 1; 1; 1; 1; 1; 0; 0; 0; 1; 0; 0; 0; 0; 1; 1; 1; 1; 1; 1;  Output: 1; Desired output: 1;  Error: 0;
Input: 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 1; 1; 1; 1; 1; 0; 0; 0; 1; 1; 0; 0; 0; 1; 1; 1; 1; 1; 1;  Output: 0; Desired output: 0;  Error: 0;
Input: 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 1; 1; 1; 1;  Output: 1; Desired output: 1;  Error: 0;
Total Mean Square Error: 0.0
```

Adaline:

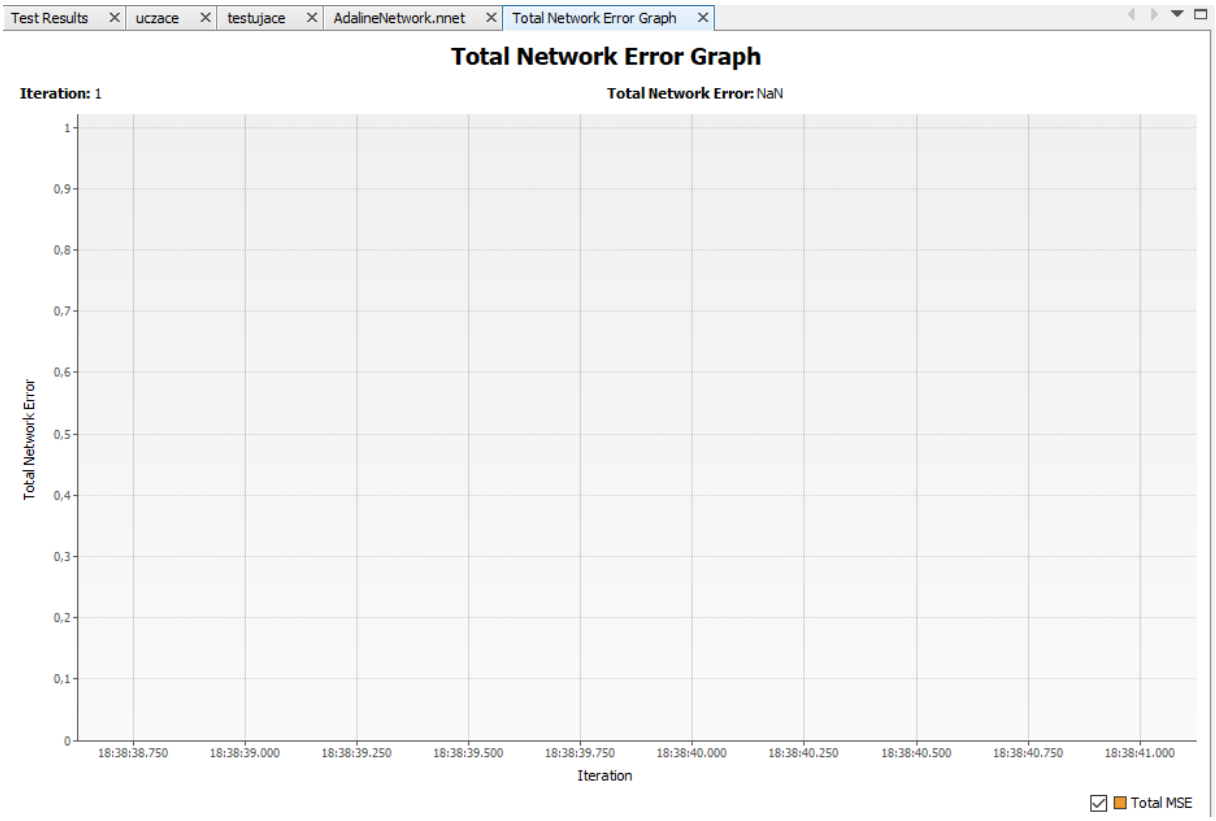
0.01

Input: 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 1; 1; 1; 0; 0; 0; 1; 0; 0; 1; 0; 0; 0; 1; 1; 1; 1; 1; Output: -0,2096; Desired output: 0; Error: -0,2096;
Input: 1; 1; 1; 1; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; Output: 2,5026; Desired output: 1; Error: 1,5026;
Input: 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 1; 1; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; 1; 0; 1; Output: 0,8741; Desired output: 0; Error: 0,8741;
Input: 1; 1; 1; 1; 1; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 1; 1; 1; 1; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 1; 1; 1; 1; 1; Output: 0,9549; Desired output: 1; Error: -0,0451;
Input: 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 1; 1; 1; 1; 1; 0; 0; 0; 0; 1; 1; 0; 0; 0; 0; 1; 1; 1; 1; 1; Output: 0,0993; Desired output: 0; Error: 0,0993;
Input: 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 1; 1; 1; 1; Output: 0,3184; Desired output: 1; Error: -0,6816;
Total Mean Square Error: 0.5903852749181672

0.1

Input: 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 1; 1; 1; 0; 0; 0; 1; 0; 0; 1; 0; 0; 0; 1; 1; 1; 1; 1; Output: -0,1097; Desired output: 0; Error: -0,1097;
Input: 1; 1; 1; 1; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; Output: 2,2204; Desired output: 1; Error: 1,2204;
Input: 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 1; 1; 1; 1; 0; 1; 0; 1; 1; 0; 1; 0; 1; 1; 0; 1; Output: 0,6848; Desired output: 0; Error: 0,6848;
Input: 1; 1; 1; 1; 1; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 1; 1; 1; 1; 1; 0; 0; 0; 1; 0; 0; 0; 0; 1; 1; 1; 1; 1; Output: 1,0757; Desired output: 1; Error: 0,0757;
Input: 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 1; 1; 1; 1; 0; 0; 0; 1; 1; 0; 0; 0; 1; 1; 1; 1; Output: 0,0995; Desired output: 0; Error: 0,0995;
Input: 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 1; 1; 1; 1; Output: 0,3913; Desired output: 1; Error: -0,6087;
Total Mean Square Error: 0.39274117056942415

0.5, 1



Dyskusja błędów, wnioski.

Perceptron okazał się bezbłędny dla każdego z testowanych współczynników uczenia. Na taki wynik może wpływać format, w jakim zapisywałem małe litery. Jest dość oczywisty dla programu, dlatego jego skuteczność jest aż tak wysoka.

Skuteczność sieci neuronów Adeline jest dużo niższa. W programie, którego używałem do realizacji projektu dla neuronu zwracającego output nie działa poprawnie próg aktywacji, a jedynie zwraca sumę. Dla wpółczynnika uczenia 0.01 i 0.01 skuteczność wyniosła ~67%. Warta zauważenia jest tendencja, że przy wyższym współczynniku sumy odpowiadające następnie za próg aktywacji coraz bardziej odbiegają od pożądanych rezultatów. Dla współczynnika 0.5 oraz 1 program wyrzucał błąd, którego powodem była błędna ilość iteracji. Niestety nie wiem czym może być to spowodowane.

Do realizacji projektu wykorzystałem program NeurophStudio. Z niego pochodzą powyższe screenshoty.