

# Raport projektu z przedmiotu sztuczna inteligencja

**Temat:** Zrealizować sieć neuronową uczoną algorytmem wstecznej propagacji błędów z przyspieszeniem metodą „momentum” (MLP\_M) uczącą się identyfikacji szkła.

**Wykonał:**  
Kamil Polit  
Nr albumu: 168159  
3EF-ZI  
Grupa:2  
Rok 2023

## **Spis treści:**

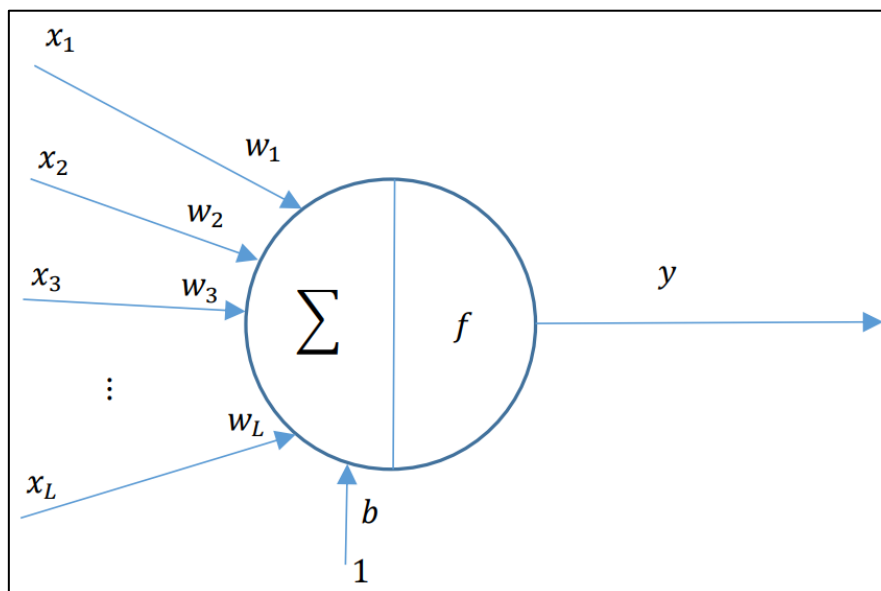
<b>1. Opis problemu .....</b>	<b>3</b>
<b>2. Część teoretyczna .....</b>	<b>3</b>
<b>2.1 Model neuronu.....</b>	<b>3</b>
<b>2.2 Sieci neuronowe jednokierunkowe wielowarstwowe .....</b>	<b>4</b>
<b>2.3 Algorytm wstecznej propagacji błędu z przyśpieszeniem         metodą „momentum” .....</b>	<b>6</b>
<b>3. Analiza danych .....</b>	<b>9</b>
<b>4. Kod programu.....</b>	<b>10</b>
<b>5. Eksperymenty.....</b>	<b>13</b>
<b>5.1 Wyznaczenie optymalnych wartości K1 oraz K2.....</b>	<b>13</b>
<b>5.2 Wyznaczenie optymalnych wartości lr oraz mc.....</b>	<b>14</b>
<b>6. Podsumowanie i wnioski .....</b>	<b>16</b>
<b>7. Bibliografia .....</b>	<b>17</b>

## 1. Opis problemu

Praca nad projektem zakłada utworzenie sieci neuronowej przy użyciu algorytmu wstecznej propagacji błędów. Następnie sieć ta zostanie poddana procesowi nauki, w którym zastosowane zostanie przyspieszenie za pomocą metody "momentum" w celu identyfikacji szkła. Przy wykorzystaniu tej sieci przeprowadzone zostaną eksperymenty, których celem będzie ustalenie, jak poszczególne współczynniki wpływają na skuteczność procesu identyfikacji.

## 2. Część teoretyczna

### 2.1. Model neuronu



Rys. 1. Model neuronu

Sygnał wyjściowy neuronu  $y$  określony jest zależnością:

$$y = f\left(\sum_{j=1}^L w_j x_j + b\right) \quad (1)$$

gdzie  $x_j$  jest  $j$ -tym ( $j = 1, 2, \dots, L$ ) sygnałem wejściowym, a  $w_j$  - współczynnikiem wagowym (wagą). Ze względu na skrócenie zapisu wygodnie będzie stosować zapis macierzowy do opisu działania neuronu. Niech  $x = [x_1, x_2, x_L]^T$  będzie wektorem sygnałów wejściowych,  $w = [w_1, w_2, \dots, w_L]^T$  - macierzą wierszową wag, a  $y$  i  $b$  - skalarami.

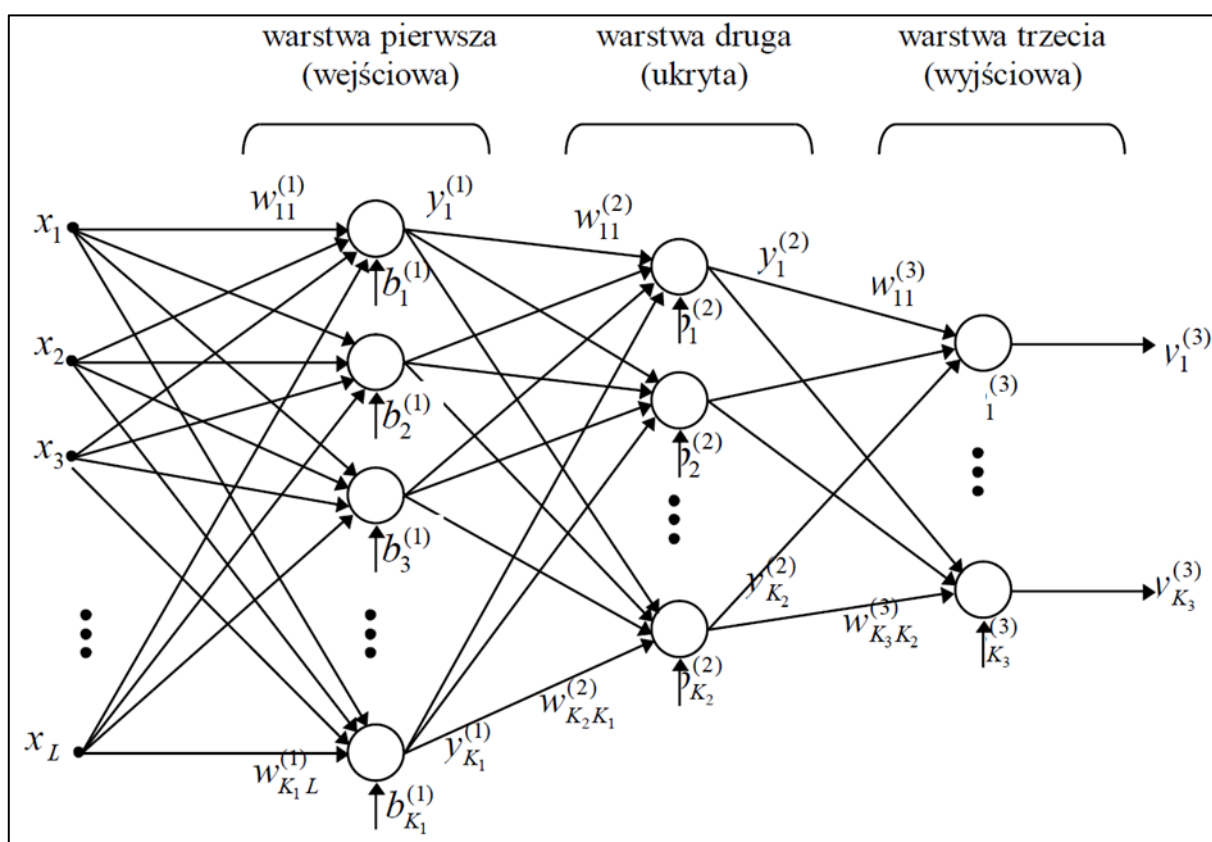
Wówczas

$$y = f(w \cdot x + b) \quad (2)$$

Ważona suma wejść wraz z przesunięciem często bywa nazywana łącznym pobudzeniem neuronu i w dalszych rozważaniach oznaczana będzie symbolem  $z$

$$z = \sum_{j=1}^L w_j x_j + b \quad (3)$$

## 2.2. Sieci neuronowe jednokierunkowe wielowarstwowe



Rys. 2. Sieć jednokierunkowa wielowarstwowa

Taką sieć nazywa się trójwarstwową. Występują tu połączenia pomiędzy warstwami neuronów typu każdy z każdym. Sygnały wejściowe podawane są do warstwy wejściowej neuronów, których wyjścia stanowią sygnały źródłowe dla kolejnej warstwy. Można wykazać, że sieć trójwarstwowa nieliniowa jest w stanie odwzorować praktycznie dowolne odwzorowanie nieliniowe.

Każda warstwa neuronów posiada swoją macierz wag  $\mathbf{w}$ , wektor przesunięć  $\mathbf{b}$ , funkcje aktywacji  $f$  i wektor sygnałów wyjściowych  $\mathbf{y}$ . Aby je rozróżniać w dalszych rozważaniach do każdej z wielkości dodano numer warstwy, której dotyczą. Na przykład dla warstwy drugiej (ukrytej) macierz wag oznaczana będzie symbolem  $\mathbf{w}(2)$ . Działanie każdej z warstw można rozpatrywać oddzielnie. I tak np. warstwa druga posiada:  $L = K1$  sygnałów wejściowych,  $K = K2$  neuronów i macierz wag  $\mathbf{w} = \mathbf{w}(2)$  o rozmiarach  $K2 \times K1$ . Wejściem warstwy drugiej jest wyjście warstwy pierwszej  $\mathbf{x} = \mathbf{y}(1)$ , a wyjściem  $\mathbf{y} = \mathbf{y}(2)$ . Działanie poszczególnych warstw dane jest przez

$$\mathbf{y}^{(1)} = f^{(1)}(\mathbf{z}^{(1)}) = f^{(1)}(\mathbf{w}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \quad (4)$$

$$\mathbf{y}^{(2)} = f^{(2)}(\mathbf{z}^{(2)}) = f^{(2)}(\mathbf{w}^{(2)}\mathbf{y}^{(1)} + \mathbf{b}^{(2)}) \quad (5)$$

$$\mathbf{y}^{(3)} = f^{(3)}(\mathbf{z}^{(3)}) = f^{(3)}(\mathbf{w}^{(3)}\mathbf{y}^{(2)} + \mathbf{b}^{(3)}) \quad (6)$$

Działanie całej sieci można, więc opisać, jako:

$$\mathbf{y}^{(3)} = f^{(3)}(\mathbf{w}^{(3)}f^{(2)}(\mathbf{w}^{(2)}\mathbf{y}^{(1)}(\mathbf{w}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}))) \quad (7)$$

Sieci jednokierunkowe wielowarstwowe wykorzystują najczęściej w warstwach wejściowej i ukrytej funkcje aktywacji typu sigmoidalnego. Typ funkcji aktywacji w warstwie wyjściowej zależy od przeznaczenia sieci. W pewnych praktycznych zastosowaniach (np. w sieciach używanych do sterowania) zastosowanie funkcji aktywacji typu sigmoidalnego może być niekorzystne. Istotne znaczenie ma tutaj zbyt wąski zakres  $(-1,1)$  sygnału wyjściowego. W związku z tym, w wielu zastosowaniach używa się funkcji liniowej w warstwie wyjściowej, która nie posiada takich ograniczeń.

## 2.3. Algorytm wstecznej propagacji błędu z przyspieszeniem metodą „momentum”

Uczenie pod nadzorem sieci wielowarstwowej zostanie przeprowadzone metodą wstecznej propagacji błędu. W przypadku sieci trójwarstwowej z rys.2 funkcja celu (w tym przypadku sumarycznego błędu kwadratowego SSE)  $E = SSE = \frac{1}{2} \sum_{i=1}^K e_i^2$  po uwzględnieniu zależności (4) - (7) przyjmie postać

$$\begin{aligned}
 E = SSE &= \frac{1}{2} \sum_{i_3=1}^{K_3} e_{i_3}^2 = \\
 &= \frac{1}{2} \sum_{i_3=1}^{K_3} (y_{i_3}^{(3)} - \hat{y}_{i_3})^2 = \frac{1}{2} \sum_{i_3=1}^{K_3} (f^{(3)}(\sum_{i_2=1}^{K_2} w_{i_3 i_2}^{(3)} y_{i_2}^{(2)} + b_{i_3}^{(3)}) - \hat{y}_{i_3})^2 = \\
 &= \frac{1}{2} \sum_{i_3=1}^{K_3} (f^{(3)}(\sum_{i_2=1}^{K_2} w_{i_3 i_2}^{(3)} f^{(2)}(\sum_{i_1=1}^{K_1} w_{i_2 i_1}^{(2)} y_{i_1}^{(1)} + b_{i_2}^{(2)}) + b_{i_3}^{(3)}) - \hat{y}_{i_3})^2 = \\
 &= \frac{1}{2} \sum_{i_3=1}^{K_3} (f^{(3)}(\sum_{i_2=1}^{K_2} w_{i_3 i_2}^{(3)} f^{(2)}(\sum_{i_1=1}^{K_1} w_{i_2 i_1}^{(2)} f^{(1)}\left(\sum_{j=1}^L w_{i_1 j}^{(1)} x_j + b_{i_1}^{(1)}\right) + b_{i_2}^{(2)}) + b_{i_3}^{(3)}) - \hat{y}_{i_3})^2
 \end{aligned} \tag{8}$$

gdzie:  $j = 1, \dots, L$  oznacza numer wejścia warstwy pierwszej,  $i_1 = 1, \dots, K_1$ ,  $i_2 = 1, \dots, K_2$ ,  $i_3 = 1, \dots, K_3$  – oznaczają odpowiednio numer wyjścia warstwy pierwszej, drugiej i trzeciej.

Odpowiednie składniki gradientu funkcji celu względem wag neuronów poszczególnych warstw otrzymuje się przez różniczkowanie zależności (8). Obliczanie wag neuronów rozpoczyna się od warstwy wyjściowej, dla której mamy

$$\frac{\partial E}{\partial w_{i_3 i_2}^{(3)}} = \frac{\partial E}{\partial y_{i_3}^{(3)}} \frac{\partial y_{i_3}^{(3)}(z_{i_3}^{(3)})}{\partial z_{i_3}^{(3)}} \frac{\partial z_{i_3}^{(3)}}{\partial w_{i_3 i_2}^{(3)}} = (y_{i_3}^{(3)} - \hat{y}_{i_3}) \frac{\partial y_{i_3}^{(3)}(z_{i_3}^{(3)})}{\partial z_{i_3}^{(3)}} y_{i_2}^{(2)} \tag{9}$$

gdzie:

$$z_{i_3}^{(3)} = \sum_{i_2=1}^{K_2} (w_{i_3 i_2}^{(3)} y_{i_2}^{(2)} + b_{i_3}^{(3)}) \tag{10}$$

jest łącznym pobudzeniem  $i_3$ -tego neuronu warstwy wyjściowej. Stosując podstawienie:

$$\delta_{i_3}^{(3)} = (y_{i_3}^{(3)} - \hat{y}_{i_3}) \frac{\partial y_{i_3}^{(3)}(z_{i_3}^{(3)})}{\partial z_{i_3}^{(3)}} \quad (11)$$

zależność przyjmie postać:

$$\frac{\partial E}{\partial w_{i_3 i_2}^{(3)}} = \delta_{i_3}^{(3)} y_{i_2}^{(2)} \quad (12)$$

Podobnie można wyznaczyć elementy gradientu względem wag warstwy ukrytej i wejściowej:

$$\frac{\partial E}{\partial w_{i_2 i_1}^{(2)}} = \frac{\partial E}{\partial y_{i_3}^{(3)}} \frac{\partial y_{i_3}^{(3)}(z_{i_3}^{(3)})}{\partial z_{i_3}^{(3)}} \frac{\partial z_{i_3}^{(3)}}{\partial y_{i_2}^{(2)}} \frac{\partial y_{i_2}^{(2)}(z_{i_2}^{(2)})}{\partial z_{i_2}^{(2)}} \frac{\partial z_{i_2}^{(2)}}{\partial w_{i_2 i_1}^{(2)}} = z_{i_3}^{(3)} \quad (13)$$

$$= \sum_{i_3=1}^{K_3} (y_{i_3}^{(3)} - \hat{y}_{i_3}) \frac{\partial y_{i_3}^{(3)}(z_{i_3}^{(3)})}{\partial z_{i_3}^{(3)}} w_{i_3 i_2}^{(3)} \frac{\partial y_{i_2}^{(2)}(z_{i_2}^{(2)})}{\partial z_{i_2}^{(2)}} y_{i_1}^{(1)} =$$

$$\frac{\partial E}{\partial w_{i_1 j}^{(1)}} = \frac{\partial E}{\partial y_{i_3}^{(3)}} \frac{\partial y_{i_3}^{(3)}(z_{i_3}^{(3)})}{\partial z_{i_3}^{(3)}} \frac{\partial z_{i_3}^{(3)}}{\partial y_{i_2}^{(2)}} \frac{\partial y_{i_2}^{(2)}(z_{i_2}^{(2)})}{\partial z_{i_2}^{(2)}} \frac{\partial z_{i_2}^{(2)}}{\partial y_{i_1}^{(1)}} \frac{\partial y_{i_1}^{(1)}(z_{i_1}^{(1)})}{\partial z_{i_1}^{(1)}} \frac{\partial z_{i_1}^{(1)}}{\partial w_{i_1 j}^{(1)}} =$$

$$\sum_{i_3=1}^{K_3} (y_{i_3}^{(3)} - \hat{y}_{i_3}) \frac{\partial y_{i_3}^{(3)}(z_{i_3}^{(3)})}{\partial z_{i_3}^{(3)}} \sum_{i_2=1}^{K_2} w_{i_3 i_2}^{(3)} \frac{\partial y_{i_2}^{(2)}(z_{i_2}^{(2)})}{\partial z_{i_2}^{(2)}} w_{i_2 i_1}^{(2)} \frac{\partial y_{i_1}^{(1)}(z_{i_1}^{(1)})}{\partial z_{i_1}^{(1)}} x_j \quad (14)$$

Analogicznie można wyznaczyć elementy gradientu względem przesunięć w poszczególnych warstwach sieci.

Zależności (11), (13) i (14) po podstawieniu do:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t) \quad (15)$$

pozwalają na uczenie odpowiednio warstw trzeciej, drugiej i pierwszej trójwarstwowej sieci neuronowej.

Do przyspieszania uczenia zostanie wykorzystana metoda „momentum”. W przypadku uczenia pod nadzorem modyfikacji gradientowej reguły uczenia (15) do postaci przyspieszania metodą momentum może wyglądać następująco:

$$w_{ij}(t + 1) = w_{ij}(t) + \Delta w_{ij}(t) + m_c M_{ij}(t) \quad (16)$$

gdzie  $\Delta w_{ij}(t)$  jest określone zależnością:

$$\Delta w = -\eta \nabla E(w) \quad (17)$$

w której  $\partial E / \partial w_{ij}$  dla poszczególnych warstw sieci wielowarstwowej uwzględnia (9), (13) i (14), zaś składnik momentum  $M_{ij}(t)$  jest wyliczany z zależności

$$M_{ij}(t) = w_{ij}(t) - w_{ij}(t - 1) \quad (18)$$

$m_c \in [0,1]$  jest współczynnikiem momentum. Jeżeli  $m_c = 0$ , to reguła (16) upraszcza się do (15). Najczęściej przyjmuje się  $m_c = 0.95$ . Wprowadzenie składnika momentum zdecydowanie wpływa na zwiększenie szybkości uczenia.



### 3. Analiza danych

Informacje na temat danych:

- Ilość przypadków: 214
- Ilość atrybutów: 9
- Zadanie: Klasyfikacja
- Brakujących danych: 0

Dane zostały podzielone na 11 kolumn, z których pierwsza zawiera unikalne identyfikatory przypadków. W drugiej kategorii znajduje się współczynnik załamania. Kolumny od 4 do 10 przedstawiają procentowe udziały poszczególnych składników chemicznych (sodu, magnezu, aluminium, silikonu, potasu, wapnia, baru oraz żelaza). Jedenasta kolumna zawiera informacje o rodzaju szkła. Możemy sklasyfikować 7 różnych rodzajów szkła: procesowane szkło do budowy okien, nieprocesowane szkło do budowy okien, procesowane szkło używane w pojazdach, nieprocesowane szkło używane w pojazdach, pojemniki, zastawa stołowa, szkło do reflektorów.

<b>1. Tabela - Dystrybucja w klasach</b>	
<b>Nazwa klasy</b>	<b>Ilość elementów</b>
<b>Używanych w oknach</b> (budynków lub pojazdów)	163
-Procesowanych	87
--Szkło do budowy okien	70
--Szkło używane w pojazdach	17
-Nieprocesowanych	76
--Szkło do budowy okien	76
--Szkło używane w pojazdach	0
<b>Nieprocesowanych</b>	51
-Szkło do budowy okien	13
-Szkło używane w pojazdach	9
-Nie jest używanych w oknach	29

Link do strony, z której pochodzą dane znajduje się w bibliografii.

## 4. Kod programu (fragmenty)

```
1) import hickle as hkl1 # Biblioteka do obsługi plików binarnych
2) import numpy as np # Biblioteka do obliczeń naukowych i numerycznych
3) import nnet as net # Biblioteka zawierająca funkcje sieci neuronowych
4) import matplotlib.pyplot as plt # Biblioteka do tworzenia wykresów
5) from sklearn.model_selection import StratifiedKFold # Klasa do podziału danych na zbiory testowe i treningowe
6) from timeit import default_timer as timer # Klasa timer do sprawdzania sekund od startu
7) # Linie 1-6: implementacja bibliotek i metod
8) class mlp_m3w:
9)     def __init__(self, x, y_t, K1, K2, lr, err_goal, \
10)                 disp_freq, mc, max_epoch, initialize): # Linia 9: Inicjalizacja konstruktora
11)                 # Linia 10: klasa z przekazanymi argumentami
12)         self.x = x # Dane wejściowe
13)         self.L = self.x.shape[0] # Liczba cech wejściowych
14)         self.y_t = y_t # Oczekiwane dane wyjściowe
15)         self.K1 = K1 # Liczba neuronów w pierwszej warstwie ukrytej
16)         self.K2 = K2 # Liczba neuronów w drugiej warstwie ukrytej
17)         self.lr = lr # Współczynnik uczenia (wpływa na tempo aktualizacji wag sieci)
18)         self.err_goal = err_goal # Wartość błędu, której chcemy osiągnąć w trakcie uczenia
19)         self.disp_freq = disp_freq # Częstotliwość wyświetlania informacji o postępie uczenia
20)         self.mc = mc # Współczynnik momentu
21)         self.max_epoch = max_epoch # Maksymalna liczba epok (iteracji) uczenia
22)         self.K3 = y_t.shape[0] # Liczba neuronów w warstwie wyjściowej
23)         self.SSE_vec = [] # Wektor przechowujący wartości błędu SSE w kolejnych epokach
24)         self.PK_vec = [] # Wektor przechowujący wartości błędu PK w kolejnych epokach
25)         self.data = self.x.T # Dane wejściowe przekształcone (transponowane)
26)         self.target = self.y_t # Oczekiwane dane wyjściowe
27)         self.initialize = initialize # Flaga wskazująca, czy należy zainicjalizować wagi
28)
29)         if self.initialize: # Inicjalizacja wag w przypadku, gdy flaga initialize jest ustawiona
30)             self.w1, self.b1 = net.nwtan(self.K1, self.L)
31)             self.w2, self.b2 = net.nwtan(self.K2, self.K1) # Linie 30-32: inicjalizacja wag
32)             self.w3, self.b3 = net.rands(self.K3, self.K2)
33)             # hkl1.dump([self.w1,self.b1,self.w2,self.b2,self.w3,self.b3], 'wagi3w.hkl1')
34)         else: # Wczytanie wag z pliku w przeciwnym razie
35)             self.w1,self.b1,self.w2,self.b2,self.w3,self.b3 = hkl1.load('wagi3w.hkl1')
36)             # Inicjalizacja poprzednich wag jako aktualnych wag
37)             self.w1_t_1, self.b1_t_1, self.w2_t_1, self.b2_t_1, self.w3_t_1, self.b3_t_1 = \
38)                 self.w1, self.b1, self.w2, self.b2, self.w3, self.b3
39)             self.SSE = 0 # Zainicjowanie sumy kwadratów błędów
40)             self.lr_vec = list() # Stworzenie pustej listy dla współczynników uczenia
41)             # Linie 12-40: zdefiniowanie parametrów klasy
42)
43)     def predict(self,x): # Funkcja obliczająca wartość wyjściową sieci neuronowej dla podanego
44)                         # wektora wejściowego x poprzez przekształcenie go przez warstwy sieci
45)         n = np.dot(self.w1, x) # Obliczanie iloczynu skalarnego warstwy 1 i danymi wejściowymi
46)         self.y1 = net.tansig( n, self.b1*np.ones(n.shape)) # Oblicza wartość funkcji tansig dla
47)                     # pierwszej warstwy
48)         n = np.dot(self.w2, self.y1) # Obliczanie iloczynu skalarnego warstwy 2 i wyjścia pierwszej warstwy y1
49)         self.y2 = net.tansig( n, self.b2*np.ones(n.shape)) # Oblicza wartość funkcji tansig dla
50)                     # drugiej warstwy
51)         n = np.dot(self.w3, self.y2) # Obliczanie iloczynu skalarnego warstwy 3 i wyjścia drugiej warstwy y2
52)         self.y3 = net.purelin(n, self.b3*np.ones(n.shape)) # Oblicza wartość funkcji purelin
53)                     # dla trzeciej warstwy
54)         return self.y3 # Zwrócenie wyjścia z trzeciej warstwy jako wynik
55)
56)     def train(self, x_train, y_train): # Funkcja train przeprowadza trening sieci neuronowej
57)                                       # przy użyciu algorytmu wstecznej propagacji błędu
58)         for epoch in range(1, self.max_epoch+1):
59)             self.y3 = self.predict(x_train) # Obliczanie wartości wyjściowej sieci dla danych treningowych
60)             self.e = y_train - self.y3 # Obliczanie błędu predykcji
61)
62)             self.SSE_t_1 = self.SSE
63)             self.SSE = net.sumsq(self.e) # Obliczanie sumy kwadratów błędów
64)             self.PK = sum((abs(self.e)<0.5).astype(int)[0])/self.e.shape[1] * 100 # Oblicza stopień
```

65)

poprawności predykcji

```

66) self.PK_vec.append(self.PK) # Dodanie stopnia poprawności do listy
67) if self.SSE < self.err_goal or self.PK == 100:
68)     break # Zatrzymanie treningu, jeśli osiągnięto docelowy
69) # błąd lub stopień poprawności wynosi 100%
70) if np.isnan(self.SSE):
71)     break # Zatrzymanie treningu, jeśli suma kwadratów
72) # błędów jest nieokreślona
73) self.d3 = net.deltalin(self.y3, self.e) # Obliczanie gradientu dla warstwy wyjściowej
74) self.d2 = net.deltatan(self.y2, self.d3, self.w3) # Obliczanie gradientu dla warstwy ukrytej 2
75) self.d1 = net.deltatan(self.y1, self.d2, self.w2) # Obliczanie gradientu dla warstwy ukrytej 1
76) self.dw1, self.db1 = net.learnbp(x_train, self.d1, self.lr) # Obliczanie zmiany wag i
77) # progów dla w1 i b1
78) self.dw2, self.db2 = net.learnbp(self.y1, self.d2, self.lr) # Obliczanie zmiany wag i
79) # progów dla w2 i b2
80) self.dw3, self.db3 = net.learnbp(self.y2, self.d3, self.lr) # Obliczanie zmiany wag i
81) # progów dla w3 i b3
82) # Próg jest parametrem dodatkowym w sieciach neuronowych, który wpływa na aktywację neuronów
83) self.w1_temp, self.b1_temp, self.w2_temp, self.b2_temp, self.w3_temp, self.b3_temp = \
84) self.w1.copy(), self.b1.copy(), self.w2.copy(), self.b2.copy(), self.w3.copy(), self.b3.copy()
85) # Tworzy kopie wag i progów
86) # Próg jest parametrem dodatkowym w sieciach neuronowych, który wpływa na aktywację neuronów.
87) self.w1 += self.dw1 + self.mc * (self.w1 - self.w1_t_1) # Aktualizuje wagi w1 z uwzględnieniem
88) # momentum
89) self.b1 += self.db1 + self.mc * (self.b1 - self.b1_t_1) # Aktualizuje wagi b1 z uwzględnieniem
90) # momentum
91) self.w2 += self.dw2 + self.mc * (self.w2 - self.w2_t_1) # Aktualizuje wagi w2 z uwzględnieniem
92) # momentum
93) self.b2 += self.db2 + self.mc * (self.b2 - self.b2_t_1) # Aktualizuje wagi b2 z uwzględnieniem
94) # momentum
95) self.w3 += self.dw3 + self.mc * (self.w3 - self.w3_t_1) # Aktualizuje wagi w3 z uwzględnieniem
96) # momentum
97) self.b3 += self.db3 + self.mc * (self.b3 - self.b3_t_1) # Aktualizuje wagi b3 z uwzględnieniem
98) # momentum
99)
100) self.w1_t_1, self.b1_t_1, self.w2_t_1, self.b2_t_1, self.w3_t_1, self.b3_t_1 = \
101) self.w1_temp, self.b1_temp, self.w2_temp, self.b2_temp, self.w3_temp, self.b3_temp
102) # Zapisuje poprzednie wagi i progi
103) self.SSE_vec.append(self.SSE) # Dodaje sumę kwadratów błędów do listy
104)
105) def train_CV(self, CV, skfold): # Funkcja wykonuje walidację krzyżową (CV) dla sieci neuronowej
106)     PK_vec = np.zeros(CVN) # Wektor do przechowywania wartości poprawności (PK) dla każdego foldu CV
107)
108)     for i, (train, test) in enumerate(skfold.split(self.data, np.squeeze(self.target)), start=0):
109)         x_train, x_test = self.data[train], self.data[test]
110)         y_train, y_test = np.squeeze(self.target)[train], np.squeeze(self.target)[test]
111)         # Linia 101-103 podział danych na zbiory treningowe i testowe
112)         self.train(x_train.T, y_train.T) # Trenowanie sieci na zbiorze treningowym
113)         result = self.predict(x_test.T) # Prognozowanie wyników dla zbioru testowego
114)         n_test_samples = test.size # Liczba próbek w zbiorze testowym
115)         PK_vec[i] = sum((abs(result - y_test)<0.5).astype(int)[0])/n_test_samples * 100 # Obliczenie PK
116)
117)     PK = np.mean(PK_vec) # Obliczenie średniej poprawności dla wszystkich foldów CV
118)     return PK # Zwrócenie średniej poprawności klasyfikacji
119)
120) x,y_t,x_norm,x_n_s,y_t_s = hkl.load('glass.hkl') # Wczytanie danych z pliku 'glass.hkl'
121)
122) max_epoch = 2000 # Maksymalna liczba epok treningu
123) err_goal = 0.25 # Docelowa wartość błędu (suma kwadratów błędów)
124) disp_freq = 200 # Częstotliwość wyświetlania informacji diagnostycznych
125)
126) lr_vec = np.array([1e-2, 1e-3, 1e-4, 1e-5]) # Wektor współczynnika uczenia
127) mc_vec = np.array([0.2, 0.5, 0.6, 0.7, 0.9]) # Wektor współczynnika momentu
128) K1_vec = np.array([1,3,5,7]) # Wektor wartości K1 do testów
129) K2_vec = K1_vec # Wektor wartości K1 do testów
130)

```

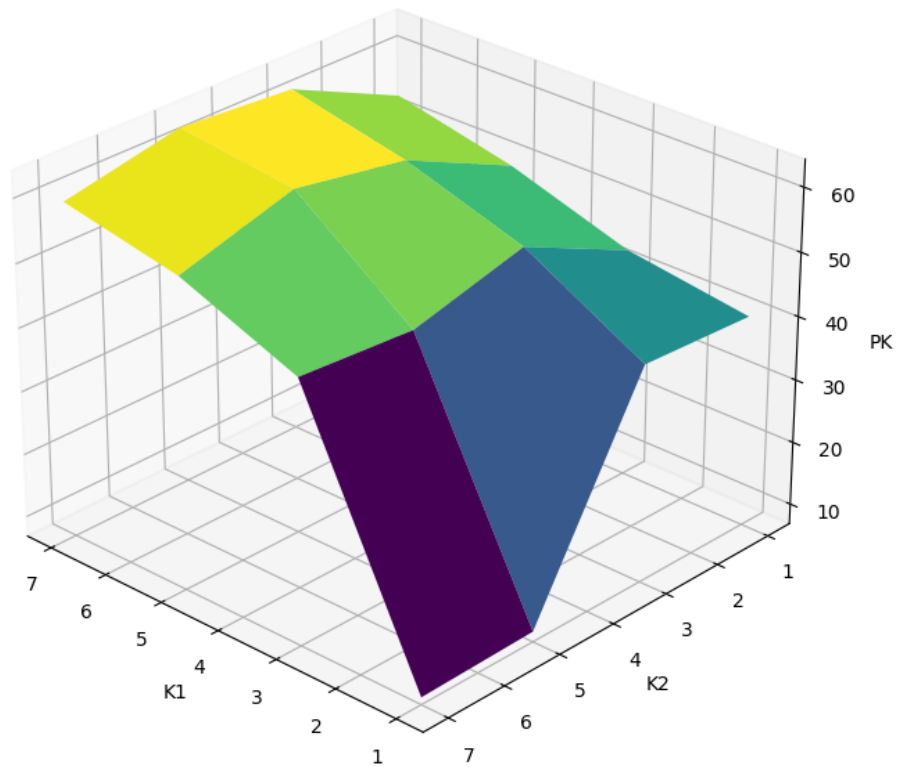
```

131) start = timer()                                # Wywołanie funkcji rozpoczynającej timer
132) CVN = 10                                       # Liczba podziałów Cross-Validation
133) skfold = StratifiedKFold(n_splits=CVN)         # Inicjalizacja obiektu StratifiedKFold z
134)                                                  podziałem na CVN części
135) PK_2D_K1K2 = np.zeros([len(K1_vec),len(K2_vec)]) # Inicjalizacja macierzy PK_2D_K1K2
136) PK_2D_lrmc = np.zeros([len(lr_vec),len(mc_vec)]) # Inicjalizacja macierzy PK_2D_lrmc
137) PK_2D_K1K2_max = 0                            # Inicjalizacja maksymalnej wartości PK
138)                                                  z macierzy PK_2D_K1K2
139) k1_ind_max = 0                                 # Indeks K1 dla maksymalnej wartości PK
140) k2_ind_max = 0                                 # Indeks K2 dla maksymalnej wartości PK
141) for k1_ind in range(len(K1_vec)):              # Pętle iterujące po indeksach w wektorach K1_vec oraz K2_vec
142)     for k2_ind in range(len(K2_vec)):
143)         mlpnet = mlp_m_3w(x_norm, y_t, K1_vec[k1_ind], K2_vec[k2_ind], \
144)             lr_vec[1], err_goal, disp_freq, mc_vec[1], \
145)             max_epoch, True)                    # Inicjalizacja obiektu mlpnet klasy mlp_a_3w
146)         PK = mlpnet.train_CV(CVN, skfold)        # Wywołanie metody train_CV obiektu mlpnet do obliczenia PK
147)         print("K1 {} | K2 {} | PK {}".format(K1_vec[k1_ind], K2_vec[k2_ind], PK))
148)                                                  # Wyświetlenie informacji o wartości PK dla danego K1 i K2
149)         PK_2D_K1K2[k1_ind, k2_ind] = PK         # Wpisanie wartości PK do odpowiedniej komórki w macierzy PK_2D_K1K2
150)         if PK > PK_2D_K1K2_max:
151)             PK_2D_K1K2_max = PK
152)             k1_ind_max = k1_ind
153)             k2_ind_max = k2_ind                  # Aktualizacja maksymalnej wartości PK oraz indeksów K1 i K2
154)
155) PK_2D_lrmc_max = 0                              # Inicjalizacja maksymalnej wartości wskaźnika PK
156) lr_ind_max = 0                                  # Inicjalizacja indeksu najlepszego współczynnika uczenia
157) mc_ind_max = 0                                  # Inicjalizacja indeksu najlepszego współczynnika momentu
158)
159) for lr_ind in range(len(lr_vec)):                # Pętle iterujące po indeksach w wektorach lr_vec oraz mc_vec
160)     for mc_ind in range(len(mc_vec)):
161)         mlpnet = mlp_m_3w(x_norm, y_t, K1_vec[k1_ind_max], K2_vec[k2_ind_max], \
162)             lr_vec[lr_ind], err_goal, disp_freq, mc_vec[mc_ind], \
163)             max_epoch, True)                    # Inicjalizacja obiektu mlpnet klasy mlp_a_3w
164)         PK = mlpnet.train_CV(CVN, skfold)        # Wywołanie metody train_CV obiektu mlpnet do obliczenia PK
165)         print("lr {} | mc {} | PK {}".format(lr_vec[lr_ind], mc_vec[mc_ind], PK))
166)                                                  # Wyświetlenie informacji o wartości PK dla danego lr i mc
167)         PK_2D_lrmc[lr_ind, mc_ind] = PK         # Wpisanie wartości PK do odpowiedniej komórki w macierzy PK_2D_lrmc
168)         if PK > PK_2D_lrmc_max:
169)             PK_2D_lrmc_max = PK
170)             lr_ind_max = lr_ind
171)             mc_ind_max = mc_ind                  # Aktualizacja maksymalnej wartości PK oraz indeksów lr i mc
172)
173) print("OPTYMALNE WARTOŚCI: K1={} | K2={} | lr={} | mc={} | PK={}.\
174)     format(K1_vec[k1_ind_max], K2_vec[k2_ind_max], lr_vec[lr_ind_max], \
175)     mc_vec[mc_ind_max], PK_2D_lrmc[lr_ind_max, mc_ind_max])) # Wyświetlanie danych
176)     optymalnych
177) print("Execution time:", timer()-start)

```

## 5. Eksperymenty

### 5.1 Wyznaczenie optymalnych wartości K1 oraz K2

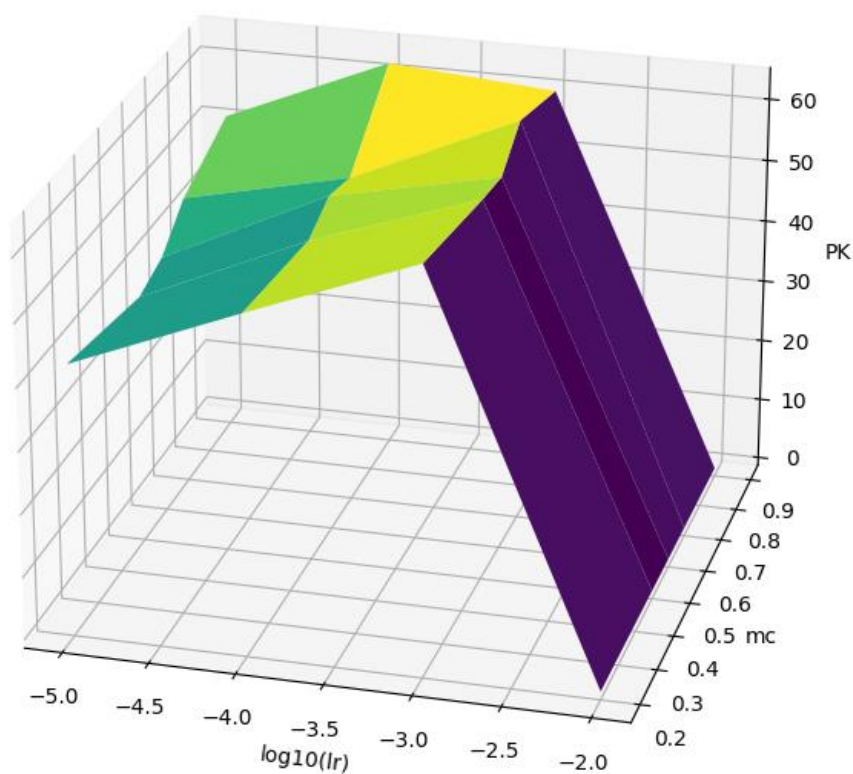


Rys. 3. Płaszczyzna wartości K1,K2 oraz PK

2. Tabela zależności PK od wartości K1 oraz K2				
K1	K2	lr_vec	mc_vec	PK
1	3	1e-3	0.5	41.0822510822511
1	5	1e-3	0.5	8.4199134199134
1	7	1e-3	0.5	7.9437229437229
5	3	1e-3	0.5	57.5974025974026
5	5	1e-3	0.5	60.8658008658009
7	5	1e-3	0.5	63.1168831168831

Z powyższego zdjęcia oraz tabeli wywnioskować możemy tendencję, że im większe wartości  $K1$  oraz  $K2$  przydzielimy tym lepszemu wyniku PK możemy się spodziewać. W większości przypadków będzie to prawda, aczkolwiek zauważyć musimy bardzo niskie wyniki dla wartości  $K1=1$  i  $K2=5$  oraz  $K1=1$  i  $K2=7$  ponieważ mogą one przeczyć naszemu wnioskowi. Możemy jednak stwierdzić że są one wynikiem czynników losowych, ponieważ taka tendencja nie pojawiają się w dalszych wynikach.

## 5.2 Wyznaczenie optymalnych wartości $l_r$ oraz $m_c$



Rys. 4. Płaszczyzna wartości  $\log_{10}(l_r)$ ,  $m_c$  oraz PK

<b>3. Tabela zależności PK od wartości <math>l_r</math> oraz <math>m_c</math></b>				
K1	K2	$l_r\_vec$	$m_c\_vec$	PK
7	5	1e-2	0.2	0.0
7	5	1e-3	0.6	59.0259740259740
7	5	1e-3	0.7	63.7662337662337
7	5	1e-3	0.9	60.2813852813852
7	5	1e-4	0.6	53.3549783549783
7	5	1e-4	0.7	51.9264069264069
7	5	1e-4	0.9	62.3160173160173
7	5	1e-5	0.5	38.7445887445887
7	5	1e-5	0.7	45.9090909090909

Jak widać na powyższym wykresie oraz tabeli najlepszy wynik PK otrzymujemy gdy  $l_r$  przyjmuje wartość 1e-3 lub 1e-4. Widzimy również, że zwiększanie lub zmniejszanie wartości  $l_r$  (1e-2 i 1e-5) drastycznie zmniejszają naszą dokładność. Nie jesteśmy jednak w stanie stwierdzić czy istnieje tendencja dla wartości  $m_c$ . Trudno jest więc stwierdzić jak dokładnie wartość  $m_c$  wpływa na naszą skuteczność, możemy się jednak domyślać, że powinniśmy próbować dopasować wartość  $m_c$  do innych zmiennych.

## 6. Podsumowanie i wnioski

Podsumowując cel projektu został zrealizowany, utworzyliśmy sieć neuronową tak aby była ona w stanie identyfikować szkło na podstawie podanych atrybutów szkła. Osiągnęła ona skuteczność wynoszącą (przy optymalnych zmiennych) w przybliżeniu 63.766%. W naszych eksperymentach wykazaliśmy zależność, w której ilości neuronów koreluje ze zwiększeniem dokładności skryptu. Upraszczając zwiększanie ilości neuronów może prowadzić do lepszych wyników PK. Jednakże mogą wystąpić przypadki gdy nasza skuteczność drastycznie spadnie (tak jak w przypadku 1 testu dla wartości  $K1=1$  i  $K2=5$  oraz  $K1=1$  i  $K2=7$ ) pomimo zwiększenia ilości neuronów otrzymaliśmy drastycznie mniejszą skuteczność. Takie zachowanie możemy jednak wytłumaczyć czynnikiem losowym na co wskazywał by fakt, że taka anomalia nie pojawia się w dalszych wynikach. W przypadku  $lr$  widzimy, że nasza skuteczność jest największa dla wartości  $lr=1e-3$  oraz  $lr=1e-4$ . Zauważyć też możemy, że ani zmniejszanie ani zwiększanie tych wartości nie poprawia naszej skuteczności (test 2 z wartościami  $lr=1e-2$  oraz  $lr=1e-5$ ). Możemy więc założyć, że są to najbardziej optymalne wartości dla naszych danych wejściowych. Takich zależności nie możemy jednak zauważyć w przypadku  $mc$ . Z naszych eksperymentów wynika że wpływ czynnika momentum na naszą skuteczność nie przedstawia konkretnego trendu. Możliwe jest więc że aby uzyskać optymalny wynik PK musimy dopasować wartość  $mc$  do pozostałych zmiennych.



## 7. Bibliografia

Dr hab. inż. prof. PRz Roman Zajdel – instrukcje do laboratoriów:

<http://materialy.przrzeszow.pl/pracownik/pliki/34/%C4%86WICZENIE%204.pdf>

(dostęp: 12.06.2023)

<http://materialy.przrzeszow.pl/pracownik/pliki/34/%C4%86WICZENIE%207.pdf>

(dostęp: 12.06.2023)

<http://materialy.przrzeszow.pl/pracownik/pliki/34/%C4%86WICZENIE%208.pdf>

(dostęp: 12.06.2023)

Dane użyte w testach pobrane zostały ze strony:

<http://archive.ics.uci.edu/dataset/42/glass+identification>

(dostęp: 12.06.2023)

Link do repozytorium z plikami projektowymi:

<https://github.com/KamilZemo/Studia/tree/main/SI>