Raport projektu z przedmiotu

sztuczna inteligencja

Temat: Zrealizować sieć neuronową uczoną algorytmem wstecznej propagacji błędu z przyśpieszeniem metodą adaptacyjnego współczynnika uczenia (MLP_A) uczącą się identyfikacji szkła.

Wykonał:

Tomasz Polit

Nr albumu: 168160

Grupa:2

3EF-ZI

Rok: 2023

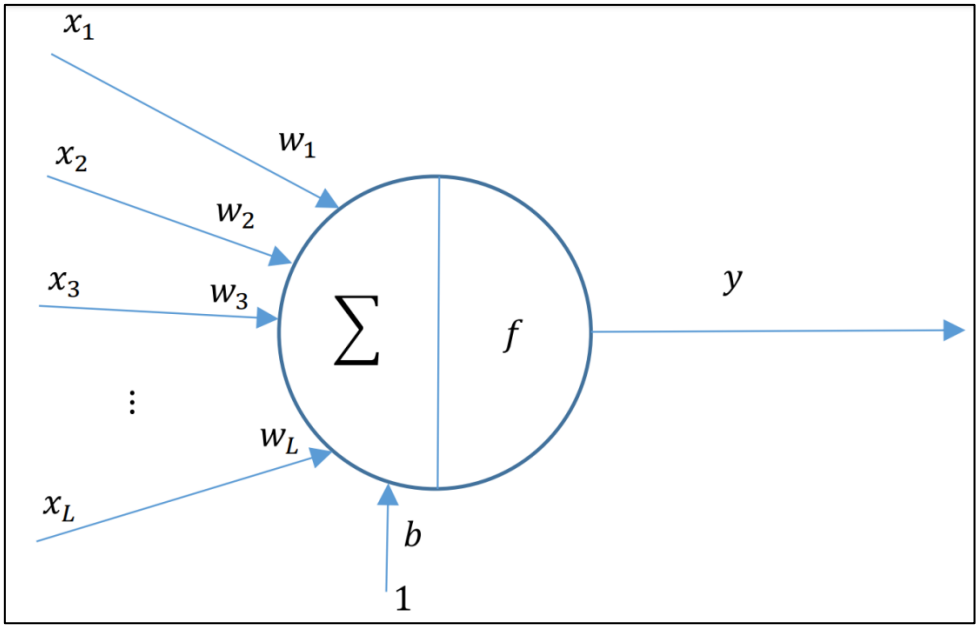
Spis treści:

1. Opis problemu

Realizacja projektu polega na stworzeniu sieci neuronowej algorytmem wstecznej propagacji błędu, która następnie zostanie poddana procesowi uczenia z przyśpieszeniem metodą adaptacyjnego współczynnika uczenia w celu identyfikacji szkła. Przy użyciu tej, że sieci zostaną przeprowadzone eksperymenty, których celem będzie ustalenie wpływu poszczególnych współczynników na poprawność identyfikacji.

2. Część teoretyczna

2.1. Model neuronu



Rys. 1. Model neuronu

Sygnał wyjściowy neuronu y określony jest zależnością:

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

gdzie x_j jest j-tym (j = 1,2,…,L) sygnałem wejściowym, a w_j - współczynnikiem wagowym (wagą). Ze względu na skrócenie zapisu wygodnie będzie stosować zapis macierzowy do opisu działania neuronu.

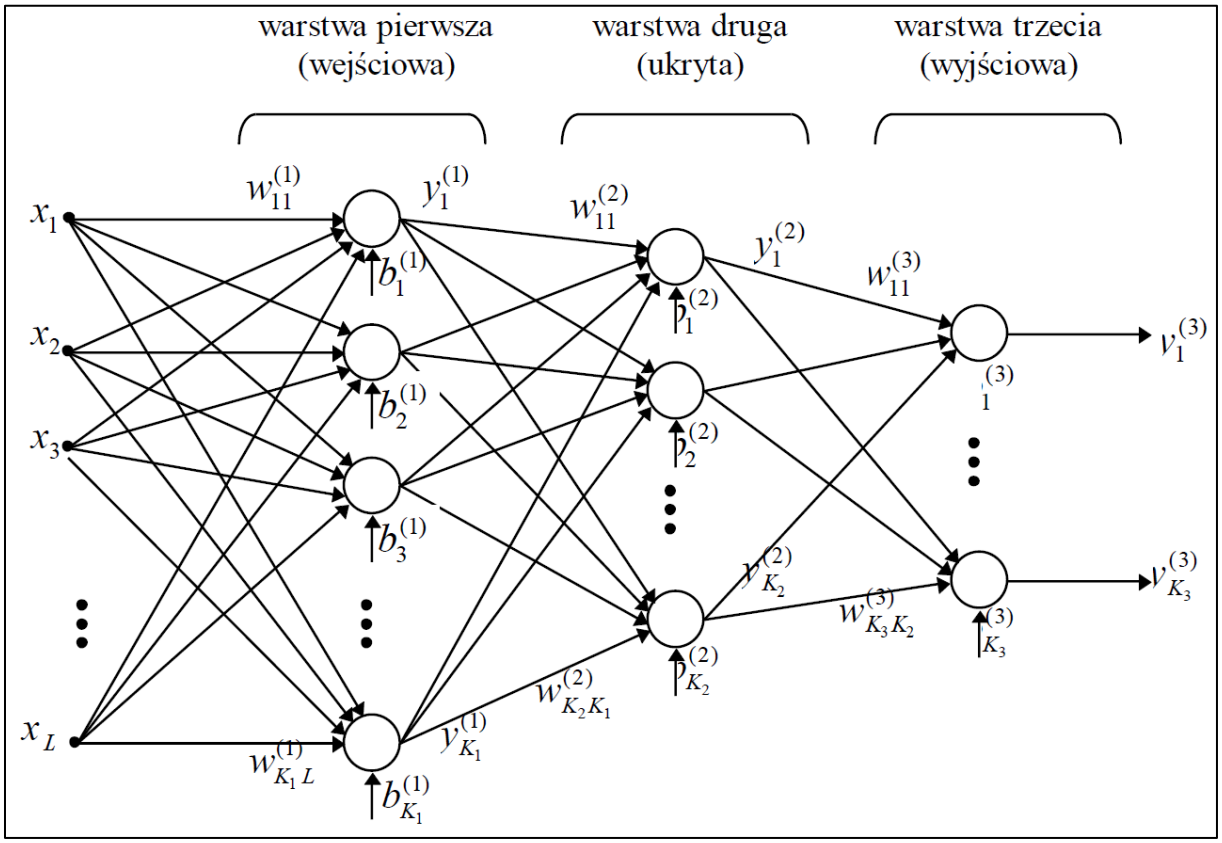
Niech $x = [x_1, x_2, x_L\ ]^T$ będzie wektorem sygnałów wejściowych, $w = [x_1, x_2, \dots, x_L]^T$ - macierzą wierszową wag, a y i b - skalarami. Wówczas

$$y = f(wx + b) \tag{2}$$

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

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

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 $\boldsymbol{w}$, wektor przesunięć $\boldsymbol{b}$, funkcje aktywacji $\boldsymbol{f}$ i wektor sygnałów wyjściowych $\boldsymbol{y}$. 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 $\boldsymbol{w}^{(2)}$. Działanie każdej z warstw można rozpatrywać oddzielnie. I tak np. warstwa druga posiada: $L = K1$ sygnałów wejściowych, $K = K2$ neuronów i macierz wag $\boldsymbol{w} = \boldsymbol{w}^{(2)}$ o rozmiarach $K2 \times K1$. Wejściem warstwy drugiej jest wyjście warstwy pierwszej $\boldsymbol{x} = \boldsymbol{y}^{(1)}$, a wyjściem $\boldsymbol{y} = \boldsymbol{y}^{(2)}$. Działanie poszczególnych warstw dane jest przez

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

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

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

Działanie całej sieci można, więc opisać, jako:

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

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 przyśpieszeniem metodą adaptacyjnego współczynnika uczenia

Uczenie pod nadzorem sieci wielowarstwowej odbywać się będzie 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

$$E = SSE = \frac{1}{2}\sum_{i_3=1}^{K_3} e_{i_3}^2 =$$

$$= \frac{1}{2}\sum_{i_3=1}^{K_3}\left(y_{i_3}{}^{(3)} - \hat{y}_{i_3}\right)^2 = \frac{1}{2}\sum_{i_3=1}^{K_3}(f^{(3)}(\sum_{i_2=1}^{K_2} w_{i_3 i_2}{}^{(3)} y_{i_2}{}^{(2)} + b_{i_3}^{(3)}) - \hat{y}_{i_3})^2 =$$

$$= \frac{1}{2}\sum_{i_3=1}^{K_3}(f^{(3)}(\sum_{i_2=1}^{K_2} w_{i_3 i_2}{}^{(3)} f^{(2)}(\sum_{i_1=1}^{K_1} w_{i_2 i_1}{}^{(2)} y_{i_1}{}^{(1)} + b_{i_2}^{(2)}) + b_{i_3}^{(3)}) - \hat{y}_{i_3})^2 =$$

$$= \frac{1}{2}\sum_{i_3=1}^{K_3}(f^{(3)}(\sum_{i_2=1}^{K_2} w_{i_3 i_2}{}^{(3)} f^{(2)}(\sum_{i_1=1}^{K_1} w_{i_2 i_1}{}^{(2)} f^{(1)}\left(\sum_{j=1}^{L} w_{i_1 j}{}^{(1)} x_j + b_{i_1}^{(1)}\right) + b_{i_2}^{(2)}) + b_{i_3}^{(3)}) - \hat{y}_{i_3})^2 \quad (8)$$

gdzie: j = 1, … , L oznacza numer wejścia warstwy pierwszej, i_1 = 1, … , K_1, i_2 =1, … , K_2, i_3 = 1, … , 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

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

gdzie:

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

jest łącznym pobudzeniem i3-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:

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

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

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

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

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

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

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

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

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

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

Do przyspieszania uczenia zostanie wykorzystana metoda adaptacyjnej korekty współczynnika uczenia η. W metodzie tej na podstawie porównania miary błędu (np. sumarycznego błędu kwadratowego) w chwili czasu t ($SSE(t)$) z jej poprzednią wartością ($SSE(t-1)$), sposób zmiany współczynnika uczenia η definiuje się jako:

$$\eta(t+1) = \begin{cases} \eta(t)\xi_d & gdy\ SSE(t) > er \cdot SSE(t-1) \\ \eta(t)\xi_i & gdy\ SSE(t) < SSE(t-1) \\ \eta(t) & gdy\ SSE(t-1) \le SSE(t) \le er \cdot SSE(t-1) \end{cases}$$

gdzie ξ_d, ξ_i są odpowiednio współczynnikami zmniejszania i zwiększania wartości η, a er dopuszczalną krotnością przyrostu błędu.

3. Analiza danych

Informacje na temat danych:

- Ilość przypadków: 214
- Ilość atrybutów: 9
- Zadanie: Klasyfikacja
- Brakujące dane: Nie występują

Dane zostały podzielone na 11 kolumn, z których pierwsza to unikalny identyfikator przypadku. Kolumna druga to współczynnik załamania. Kolumny od 4 do 10 to procent wagi odpowiednich cząsteczek chemicznych (sodu, magnezu, aluminium, silikonu, potasu, wapnia, baru i żelaza). Kolumna jedenasta to rodzaj szkła. Wyróżniamy 7 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.

Tabela 1 - Dystrybucja w klasach	
Klasa	**Liczba przypadków**
1. Używane w oknach (budynków i pojazdów)	163
1.1. Procesowane	87
1.1.1. Szkło do budowy okien budynków	70
1.1.2. Szkło używane w pojazdach	17
1.2. Nieprocesowane	76
1.2.1. Szkło do budowy okien budynków	76
1.2.2. Szkło używane w pojazdach	0
2. Nieużywane w oknach (budynków i pojazdów)	51
2.1. Pojemniki	13
2.2. Zastawa stołowa	9
2.3. Szkło do reflektorów	29

Link do danych użytych w eksperymentach znajdują się w bibliografii.

4. Skrypt programu

```
1.  import hickle as hkl                                # 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 oraz treningowe
6.                                                      # Linie 1-5: implementacja bibliotek i metod
7.  class mlp_a_3w:
8.     def __init__(self, x, y_t, K1, K2, lr, err_goal, disp_freq, ksi_inc, ksi_dec, er, max_epoch):
9.                                                  # Linia 8: Inicjalizacja konstruktora klasy z przekazanymi argumentami
10.       self.x          = x                       # Dane wejściowe
11.       self.L          = self.x.shape[0]         # Liczba cech wejściowych
12.       self.y_t        = y_t                     # Oczekiwane dane wyjściowe
13.       self.K1         = K1                      # Liczba neuronów w pierwszej warstwie ukrytej
14.       self.K2         = K2                      # Liczba neuronów w drugiej warstwie ukrytej
15.       self.K3         = y_t.shape[0]            # Liczba neuronów w warstwie wyjściowej
16.       self.lr         = lr                      # Współczynnik uczenia (wpływa na tempo aktualizacji wag sieci)
17.       self.err_goal   = err_goal                # Wartość błędu, której chcemy osiągnąć w trakcie uczenia
18.       self.disp_freq  = disp_freq               # Częstotliwość wyświetlania informacji o postępie uczenia
19.       self.ksi_inc    = ksi_inc                 # Wartość o jaką zwiększane są wagi pozytywnie wpływające na wynik
20.       self.ksi_dec    = ksi_dec                 # Wartość o jaką zmniejszane są wagi negatywnie wpływające na wynik
21.       self.er         = er                      # Procentowy limit błędu, który determinuje koniec uczenia
22.       self.max_epoch  = max_epoch               # Maksymalna liczba epok (iteracji) uczenia
23.       self.data       = self.x.T                # Dane wejściowe przekształcone (transponowane)
24.       self.target     = self.y_t                # Oczekiwane dane wyjściowe
25.       self.SSE_vec    = []                      # Wektor przechowujący wartości błędu SSE w kolejnych epokach
26.       self.PK_vec     = []                      # Wektor przechowujący wartości błędu PK w kolejnych epokach
27.
28.       self.w1, self.b1 = net.nwtan(self.K1, self.L)
29.       self.w2, self.b2 = net.nwtan(self.K2, self.K1)
30.       self.w3, self.b3 = net.rands(self.K3, self.K2)
31.                                                 # Linie 28-30: inicjalizacja wag
32.       self.SSE = 0                              # Zainicjowanie sumy kwadratów błędów
33.       self.lr_vec = list()                      # Stworzenie pustej listy dla współczynników uczenia
34.                                                 # Linie 10-34: zdefiniowanie parametrów klasy
```

```
35.    def predict(self,x):              # Funkcja obliczająca wartość wyjściową sieci neuronowej dla podanego
36.                                       wektora wejściowego x poprzez przekształcenie go przez warstwy sieci
37.       n = np.dot(self.w1, x)         # Obliczanie iloczynu skalarnego warstwy 1 i danymi wejściowymi
38.       self.y1 = net.tansig( n,  self.b1*np.ones(n.shape)) # Oblicza wartość funkcji tansig dla pierwszej warstwy
39.       n = np.dot(self.w2, self.y1)  # Obliczanie iloczynu skalarnego warstwy 2 i wyjścia pierwszej warstwy y1
40.       self.y2 = net.tansig( n,  self.b2*np.ones(n.shape)) # Oblicza wartość funkcji tansig dla drugiej warstwy
41.       n = np.dot(self.w3, self.y2)  # Obliczanie iloczynu skalarnego warstwy 3 i wyjścia drugiej warstwy y2
42.       self.y3 = net.purelin(n, self.b3*np.ones(n.shape)) # Oblicza wartość funkcji purelin dla trzeciej warstwy
43.       return self.y3                                      #Zwrócenie wyjścia z trzeciej warstwy jako wynik
44.
45.    def train(self, x_train, y_train):             # Funkcja train przeprowadza trening sieci neuronowej
46.                                                      przy użyciu algorytmu wstecznej propagacji błędu
47.       for epoch in range(1, self.max_epoch+1):
48.          self.y3 = self.predict(x_train)          # Obliczanie wartości wyjściowej sieci dla danych treningowych
49.          self.e = y_train - self.y3               # Obliczanie błędu predykcji
50.
51.          self.SSE_t_1 = self.SSE
52.          self.SSE = net.sumsqr(self.e)            # Obliczanie sumy kwadratów błędów
53.          self.PK = sum((abs(self.e)<0.5).astype(int)[0])/self.e.shape[1] * 100 # Oblicza stopień poprawności
54.                                                                                   predykcji
55.          self.PK_vec.append(self.PK)                     # Dodanie stopnia poprawności do listy
56.          if self.SSE < self.err_goal or self.PK == 100:
57.             break     # Zatrzymanie treningu, jeśli osiągnięto docelowy błąd lub stopień poprawności wynosi 100%
58.          if np.isnan(self.SSE):
59.             break     # Zatrzymanie treningu, jeśli suma kwadratów błędów jest nieokreślona
60.          else:
61.              if self.SSE > self.er * self.SSE_t_1:   # Sprawdzenie czy obecny błąd nie przekroczył iloczynu
62.                                                        poprzedniego błędu i dopuszczalnej krotności błędu
63.                  self.lr *= self.ksi_dec             # Aktualizacja wartości lr jako iloczyn lr i ksi_dec
64.              elif self.SSE < self.SSE_t_1:           # Sprawdzenie czy obecny błąd jest mniejszy od poprzedniego
65.                  self.lr *= self.ksi_inc             # Aktualizacja wartości lr jako iloczyn lr i ksi_inc
66.          self.lr_vec.append(self.lr)                 # Dodanie wartości lr na koniec wektora lr_vec
67.
68.          self.d3 = net.deltalin(self.y3, self.e)                # Obliczanie gradientu dla warstwy wyjściowej
69.          self.d2 = net.deltatan(self.y2, self.d3, self.w3)      # Obliczanie gradientu dla warstwy ukrytej 2
70.          self.d1 = net.deltatan(self.y1, self.d2, self.w2)      # Obliczanie gradientu dla warstwy ukrytej 1
71.          self.dw1, self.db1 = net.learnbp(x_train, self.d1, self.lr) # Obliczanie zmiany wag i progów dla w1 i b1
72.          self.dw2, self.db2 = net.learnbp(self.y1, self.d2, self.lr) # Obliczanie zmiany wag i progów dla w2 i b2
73.          self.dw3, self.db3 = net.learnbp(self.y2, self.d3, self.lr) # Obliczanie zmiany wag i progów dla w3 i b3
74.              # Próg jest parametrem dodatkowym w sieciach neuronowych, który wpływa na aktywację neuronów
75.          self.w1 += self.dw1                                    # Aktualizacja wagi w1
76.          self.b1 += self.db1                                    # Aktualizacja progu b1
77.          self.w2 += self.dw2                                    # Aktualizacja wagi w2
78.          self.b2 += self.db2                                    # Aktualizacja progu b2
79.          self.w3 += self.dw3                                    # Aktualizacja wagi w3
80.          self.b3 += self.db3                                    # Aktualizacja progu b3
81.          self.SSE_vec.append(self.SSE)                          # Dodanie sumy kwadratów błędów do listy
```

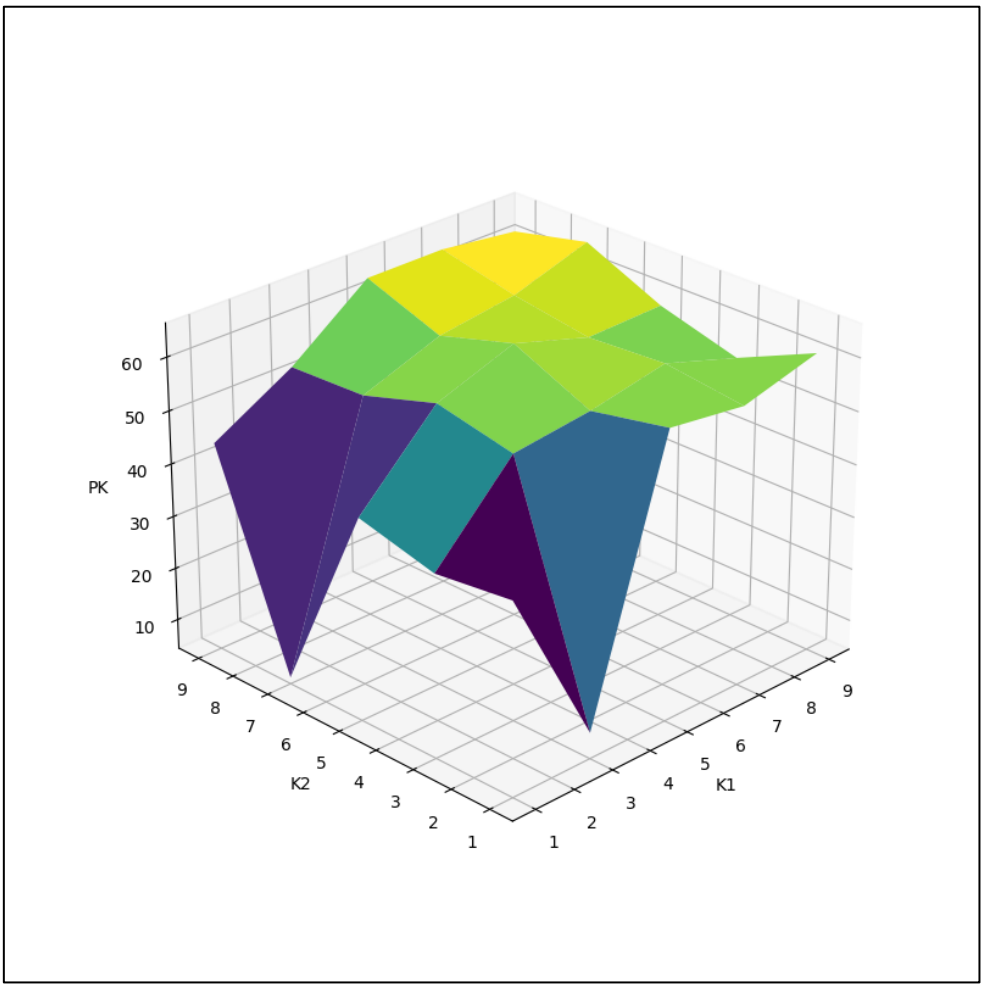
```
82.     def train_CV(self, CV, skfold):  # Funkcja wykonuje walidację krzyżową (CV) dla sieci neuronowej
83.         PK_vec = np.zeros(CVN)     # Wektor do przechowywania wartości poprawności (PK) dla każdego foldu CV
84.         for i, (train, test) in enumerate(skfold.split(self.data, np.squeeze(self.target)), start=0):
85.             x_train, x_test = self.data[train], self.data[test]
86.             y_train, y_test = np.squeeze(self.target)[train], np.squeeze(self.target)[test]
87.                                                  # Linia 84-86 podział danych na zbiory treningowe i testowe
88.             self.train(x_train.T, y_train.T)      # Trenowanie sieci na zbiorze treningowym
89.             result = self.predict(x_test.T)       # Prognozowanie wyników dla zbioru testowego
90.             n_test_samples = test.size            # Liczba próbek w zbiorze testowym
91.             PK_vec[i] = sum((abs(result - y_test)<0.5).astype(int)[0])/n_test_samples * 100  # Obliczenie PK
92.             PK = np.mean(PK_vec)                  # Obliczenie średniej poprawności dla wszystkich foldów CV
93.             return PK                             # Zwrócenie średniej poprawności klasyfikacji
94.
95. x,y_t,x_norm,x_n_s,y_t_s = hkl.load('glass.hkl')   # Wczytanie danych z pliku 'glass.hkl'
96.
97. max_epoch = 2000                          # Maksymalna liczba epok treningu
98. err_goal = 0.25                           # Docelowa wartość błędu (suma kwadratów błędów)
99. disp_freq = 100                           # Częstotliwość wyświetlania informacji diagnostycznych
100. lr = 1e-4                               # Współczynnik uczenia
101.
102. CVN = 9                                 # Liczba podziałów Cross-Validation
103. skfold = StratifiedKFold(n_splits=CVN)   # Inicjalizacja obiektu StratifiedKFold z podziałem na CVN części
104.
105. K1_vec = np.array([1,3,5,7,9])                  # Wektor wartosci K1 do testów
106. K2_vec = K1_vec                                 # Wektor wartosci K2 do testów
107. PK_2D_K1K2 = np.zeros([len(K1_vec),len(K2_vec)])  # Inicjalizacja macierzy PK_2D_K1K2
108.
109. PK_2D_K1K2_max = 0                        # Inicjalizacja maksymalnej wartości PK z macierzy PK_2D_K1K2
110. k1_ind_max = 0                           # Indeks K1 dla maksymalnej wartości PK
111. k2_ind_max = 0                           # Indeks K2 dla maksymalnej wartości PK
112.
113. ksi_inc_vec = np.array(np.arange(1.01,1.10,0.02))              # Wektor wartosci ksi_inc do testów
114. ksi_dec_vec = np.array(np.arange(0.50,0.99,0.1))               # Wektor wartosci ksi_dec do testów
115. PK_2D_IncDec = np.zeros([len(ksi_inc_vec),len(ksi_dec_vec)])    # Inicjalizacja macierzy PK_2D_ IncDec
116.
117. PK_2D_IncDec_max = 0                      # Inicjalizacja maksymalnej wartości PK z macierzy PK_2D_ IncDec
118. ksi_inc_ind_max = 0                      # Indeks ksi_inc dla maksymalnej wartości PK
119. ksi_dec_ind_max = 0                      # Indeks ksi_dec dla maksymalnej wartości PK
120.
121. er_vec = np.array(np.arange(1.00,1.10,0.02))      # Wektor wartosci er do testów
122. PK_er = np.zeros([len(er_vec)])                   # Inicjalizacja macierzy PK_2D_er
123. PK_er_max = 0                                    # Inicjalizacja maksymalnej wartości PK z macierzy PK_2D_er
124. er_ind_max = 0                                   # Indeks er dla maksymalnej wartości PK
```

```
for k1_ind in range(len(K1_vec)):
 for k2_ind in range(len(K2_vec)): # Pętle iterujące po indeksach w wektorach K1_vec oraz K2_vec
    mlpnet = mlp_a_3w(x_norm, y_t, K1_vec[k1_ind], K2_vec[k2_ind], lr, err_goal, disp_freq, ksi_inc_vec[2],
                      ksi_dec_vec[2], er_vec[2], max_epoch)      # Inicjalizacja obiektu mlpnet klasy mlp_a_3w
    PK = mlpnet.train_CV(CVN, skfold)      # Wywołanie metody train_CV obiekctu mlpnet do obliczenia PK
    print("K1 {} | K2 {} | PK {}". format(K1_vec[k1_ind], K2_vec[k2_ind], PK))
                                           # Wyświetlenie informacji o wartości PK dla danego K1 i K2
    PK_2D_K1K2[k1_ind, k2_ind] = PK        # Wpisanie wartości PK do odpowiedniej komórki w macierzy PK_2D_K1K2
    if PK > PK_2D_K1K2_max:
       PK_2D_K1K2_max = PK
       k1_ind_max = k1_ind
       k2_ind_max = k2_ind                 # Aktualizacja maksymalnej wartości PK oraz indeksów K1 i K2

fig = plt.figure(figsize=(8, 8))                                    # Utworzenie figury
ax1 = fig.add_subplot(111, projection='3d')                         # Dodanie figury do podplotu
X, Y = np.meshgrid(K1_vec, K2_vec)                                  # Utworzenie siatki z wartości K1 i K2
surf = ax1.plot_surface(X, Y, PK_2D_K1K2.T, cmap='viridis')      # Utworzenie powierzchni z wartości K1,K2 i PK

ax1.set_xlabel('K1')
ax1.set_ylabel('K2')
ax1.set_zlabel('PK')
                                                                    # Linie 143-145 Dodanie oznaczeń osi
plt.savefig("Fig.1_PK_K1K2_glass.png",bbox_inches='tight')      # Zapisanie zdjęcia figury

print("OPTYMALNE WARTOŚCI: K1={} | K2={}".
      format(K1_vec[k1_ind_max], K2_vec[k2_ind_max]))     # Wyświetlenie optymalnych wartości K1 i K2

for ksi_inc_ind in range(len(ksi_inc_vec)):
 for ksi_dec_ind in range(len(ksi_dec_vec)): # Pętle iterujące po indeksach w wektorach ksi_inc_vec oraz
                                             ksi_dec_vec
    mlpnet = mlp_a_3w(x_norm, y_t, K1_vec[k1_ind_max], K2_vec[k2_ind_max], lr, err_goal, disp_freq,
                          ksi_inc_vec[ksi_inc_ind], ksi_dec_vec[ksi_dec_ind], er_vec[2], max_epoch)
                                           # Inicjalizacja obiektu mlpnet klasy mlp_a_3w
    PK = mlpnet.train_CV(CVN, skfold)      # Wywołanie metody train_CV obiekctu mlpnet do obliczenia PK
    print("ksi_inc {} | ksi_dec {} | PK {}". format(ksi_inc_vec[ksi_inc_ind], ksi_dec_vec[ksi_dec_ind], PK))
                                           # Wyświetlenie informacji o wartości PK dla danego ksi_inc i ksi_dec
    PK_2D_IncDec[ksi_inc_ind, ksi_dec_ind] = PK
                                           # Wpisanie wartości PK do odpowiedniej komórki w macierzy PK_2D_IncDec
    if PK > PK_2D_IncDec_max:
       PK_2D_IncDec_max = PK
       ksi_inc_ind_max = ksi_inc_ind
       ksi_dec_ind_max = ksi_dec_ind       # Aktualizacja maksymalnej wartości PK oraz indeksów ksi_inc i ksi_dec
```

```
fig = plt.figure(figsize=(8, 8))                                    # Utworzenie figury
ax2 = fig.add_subplot(111, projection='3d')                         # Dodanie figury do podplotu
X, Y = np.meshgrid(ksi_inc_vec, ksi_dec_vec)                        # Utworzenie siatki z wartości ksi_inc i ksi_dec
surf = ax2.plot_surface(X, Y, PK_2D_IncDec.T, cmap='cool')          # Utworzenie powierzchni z wartości ksi_inc,
                                                                      ksi_dec i PK
ax2.set_xlabel('ksi_inc')
ax2.set_ylabel('ksi_dec')
ax2.set_zlabel('PK')
                                                                    # Linie 172-174 Dodanie oznaczeń osi
plt.savefig("Fig.2_PK_IncDec_glass.png",bbox_inches='tight')        # Zapisanie zdjęcia figury

print("OPTYMALNE WARTOŚCI: ksi_inc={} | ksi_dec={}".
      format(ksi_inc_vec[ksi_inc_ind_max], ksi_dec_vec[ksi_dec_ind_max]))
                                          # Wyświetlenie optymalnych wartości ksi_inc i ksi_dec
for er_ind in range(len(er_vec)):         # Pętle iterujące po indeksach w wektorze er_vec
    mlpnet = mlp_a_3w(x_norm, y_t, K1_vec[k1_ind_max], K2_vec[k2_ind_max], lr, err_goal, disp_freq,
                      ksi_inc_vec[ksi_inc_ind_max], ksi_dec_vec[ksi_dec_ind_max], er_vec[er_ind], max_epoch)
                                          # Inicjalizacja obiektu mlpnet klasy mlp_a_3w
    PK = mlpnet.train_CV(CVN, skfold)     # Wywołanie metody train_CV obiekctu mlpnet do obliczenia PK
    print("er {} | PK {}". format(er_vec[er_ind], PK)) # Wyświetlenie informacji o wartości PK dla danego er
    PK_er[er_ind] = PK                    # Wpisanie wartości PK do odpowiedniej komórki w macierzy PK_2D_er
    if PK > PK_er_max:
       PK_er_max = PK
       er_ind_max = er_ind                # Aktualizacja maksymalnej wartości PK oraz indeksu er

fig = plt.figure(figsize=(8, 8))                            # Utworzenie figury
ax3 = fig.add_subplot(111)                                  # Dodanie figury do podplotu
ax3.set_xlabel('er')
ax3.set_ylabel('PK')
ax3.plot(er_vec, PK_er)
                                                            # Linie 192-196 Dodanie oznaczeń osi
plt.savefig("Fig.3_PK_er_glass.png",bbox_inches='tight') # Zapisanie zdjęcia figury

print("OPTYMALNE WARTOŚCI: K1={} | K2={} | ksi_inc={} | ksi_dec={} | er={} | PK={}".
      format(K1_vec[k1_ind_max], K2_vec[k2_ind_max], ksi_inc_vec[ksi_inc_ind_max], ksi_dec_vec[ksi_dec_ind_max],
             er_vec[er_ind_max], PK_er_max))             # Wyświetlanie danych optymalnych
```

5. Eksperymenty

5.1. Wyznaczenie optymalnych wartości K1 oraz K2

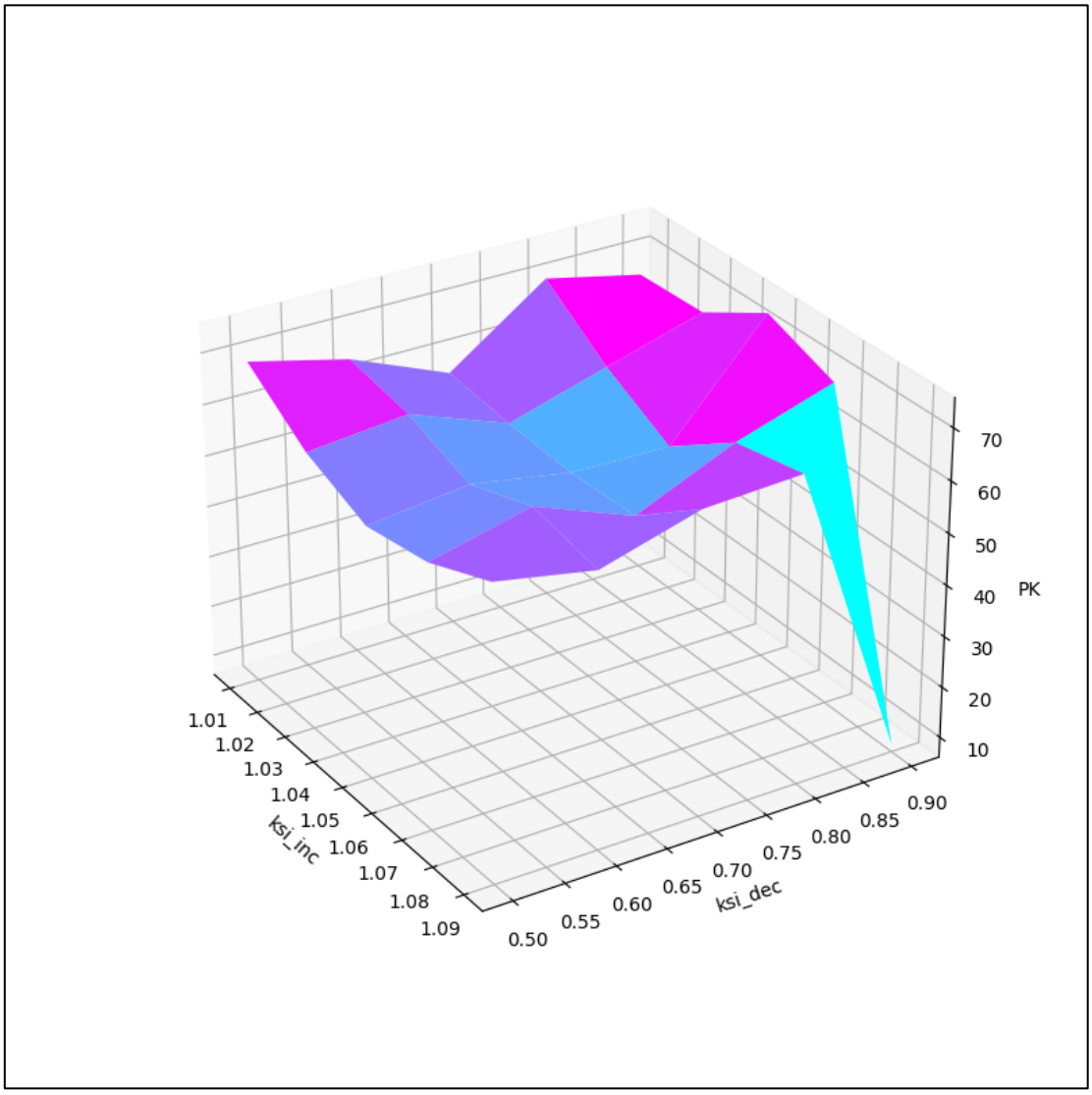


Rys. 3. Płaszczyzna wartości K1,K2 oraz PK

Tabela 2 – zależność poprawności klasyfikacji od zmiennych K1 i K2					
K1	K2	ksi_inc	ksi_dec	er	PK
1	5	1,05	0,70	1,04	42.57246376811594
1	7	1,05	0,70	1,04	5.595813204508856
1	9	1,05	0,70	1,04	43.94122383252818
3	1	1,05	0,70	1,04	9.25925925925926
3	3	1,05	0,70	1,04	54.32769726247987
5	5	1,05	0,70	1,04	62.600644122383265

Na zamieszczonych powyżej rysunku i tabeli zauważyć możemy bardzo ciekawą anomalię. Widzimy, że wartość PK dla K1=1 i K2=7 oraz K1=3 i K2=1 odnotowaliśmy bardzo duży spadek dokładności. Możemy również odnotować pewną tendencję. Im większe są wartości K1 oraz K2 tym większe są wartości, wokół, których wartość poprawności klasyfikacji się znajduje.

5.2. Wyznaczenie optymalnych wartości ksi_inc i ksi_dec

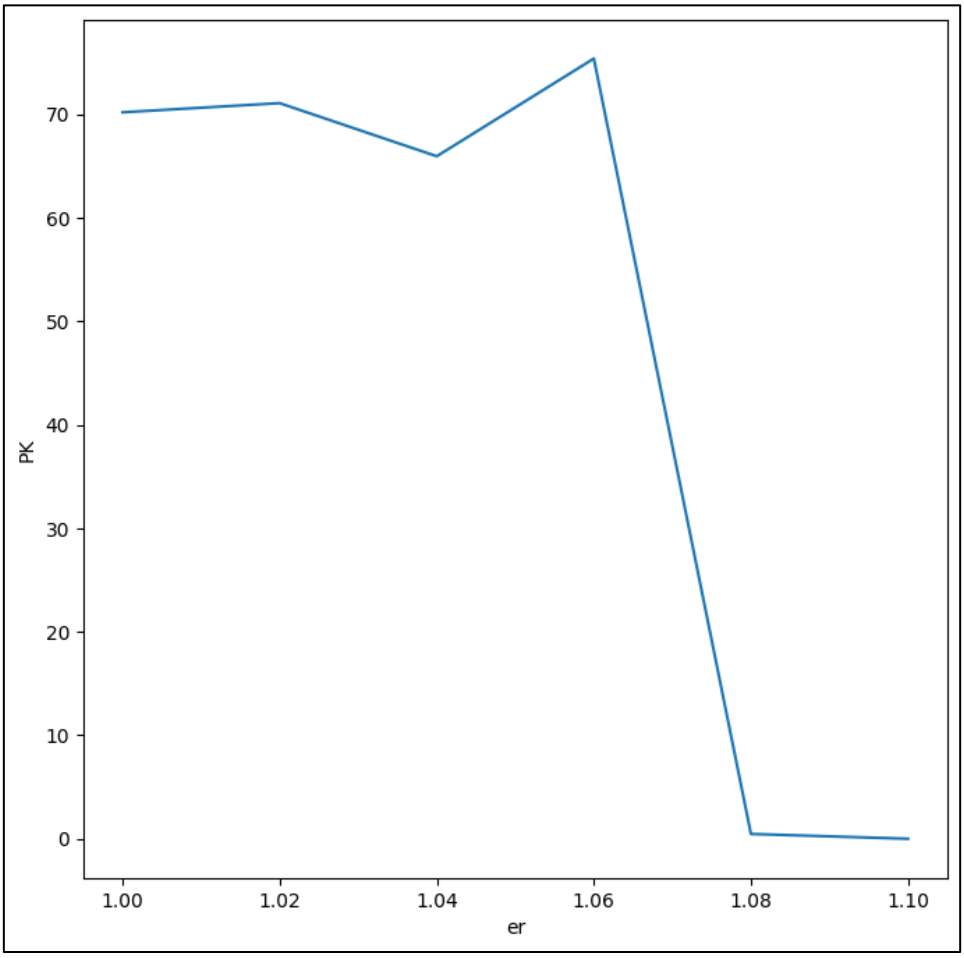


Rys. 4. Płaszczyzna wartości ksi_inc,ksi_dec oraz PK

Tabela 3 – zależność poprawności klasyfikacji od zmiennych ksi_inc i ksi_dec					
K1	K2	ksi_inc	ksi_dec	er	PK
9	7	1.01	0.50	1,04	69.24315619967794
9	7	1.01	0.70	1,04	56.60225442834138
9	7	1.05	0.90	1,04	74.29549114331724
9	7	1.09	0.80	1,04	65.9621578099839
9	7	1.09	0.90	1,04	7.407407407407408

Z zamieszonych powyżej zdjęcia i tabeli widzimy, że ciężko jest wyznaczyć jeden postarzający się schemat zmian PK w zależności od ksi_inc oraz ksi_dec. Możliwym jest, że ważniejszym aspektem jest dostrojenie zmiennych względem siebie. Ponownie możemy zauważyć bardzo ciekawą anomalię. W ostatnim przypadku ponownie zauważyliśmy wyjątkowy spadek dokładności klasyfikacji.

5.3. Eksperyment dla najlepszych wartości er



Rys. 5. Graf zależności er oraz PK

Tabela 4 – zależność poprawności klasyfikacji od zmiennej er					
K1	K2	ksi_inc	ksi_dec	er	PK
9	7	1.05	0.90	1.04	65.94202898550725
9	7	1.05	0.90	1.06	75.38244766505636
9	7	1.05	0.90	1.08	0.4629629629629629
9	7	1.05	0.90	1.10	0.0

Ponownie obserwując powyższy rysunek oraz tabelę ciężko jest oszacować wzór, który sugerowałby zależność poprawności klasyfikacji od wartości er. Jest to prawdopodobnie kolejny przypadek, w którym aby zwiększyć wartość PK należy dopasować wartość er do pozostałych zmiennych. Ponownie również widzimy, że otrzymaliśmy niskie wartości PK dla dwóch ostatnich przypadków.

6. Podsumowanie i wnioski

Cel projektu został zrealizowany, ponieważ utworzona sieć neuronowa była wstanie identyfikować szkło na podstawie podanych atrybutów. Skuteczność algorytmu przy najbardziej optymalnych zmiennych wniosła w przybliżeniu 75.382448 %. Eksperymenty wykazały, że zwiększenie ilości neuronów koreluje ze zwiększeniem dokładności skryptu, lecz nie jest zasadą. Na każdym z wykresów odnotowaliśmy przynajmniej jeden przypadek, w którym poprawność klasyfikacji drastycznie spada. Jak widzimy w dwóch przypadkach, gdy K1=1 a K2=7 oraz gdy K1=3 a K2=1 odnotowaliśmy duży spadek dokładności skryptu. Możemy to spróbować wytłumaczyć wpływem losowym, jest to tym bardziej możliwe, jeżeli spojrzymy na kolej wartości gdzie taka anomalia nie wystąpiła. Podobnie po oszacowaniu wartości optymalnych dla K1 i K2 odnotowaliśmy podobną sytuację dla ksi_inc=1,09 oraz ksi_dec=0,90 oraz następnie dla er=1,08 oraz er=1,10. Nie można jednak wykluczyć, że w tych przypadkach również czynniki losowe mogły spowodować tak drastyczną różnicę. W przypadku ksi_inc oraz ksi_dec trudno oszacować czy zwiększenie wartości powoduje polepszenie dokładności. Ważniejszym aspektem w tym przypadku wydaje się odpowiednie dostrojenie wartości względem siebie. W przypadku szacowania optymalnej wartości er zauważyć możemy sytuację, w której widzimy, że zwiększenie wartości er powoduje spadek poprawności klasyfikacji. Drastyczną zmianę możemy podobnie jak pozostałe wartości uznać za wynik obciążony wpływem czynników losowych.

7. Bibliografia

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

http://materialy.prz-rzeszow.pl/pracownik/pliki/34/%C4%86WICZENIE%204.pdf (dostęp: 09.06.2023)

http://materialy.prz-rzeszow.pl/pracownik/pliki/34/%C4%86WICZENIE%207.pdf (dostęp: 09.06.2023)

http://materialy.prz-rzeszow.pl/pracownik/pliki/34/%C4%86WICZENIE%208.pdf (dostęp: 09.06.2023)

Dane użyte w eksperymentach pochodzą ze strony:

http://archive.ics.uci.edu/dataset/42/glass+identification (dostęp: 09.06.2023)