POLITECHNIKA RZESZOWSKA
im. IGNACEGO ŁUKASIEWICZA

WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI
POLITECHNIKI RZESZOWSKIEJ

Projekt z przedmiotu
Sztuczna Inteligencja

Jakub Duda 169532 2 EF-DI P1

Rzeszów, 2023

Spis treści

1. Opis projektu

1.1 Założenia projektu

Założeniem projektu jest zrealizowanie perceptronu trójwarstwowego z przyśpieszaniem metodą momentum i adaptacyjnym współczynnikiem uczenia uczącym się klasyfikacji zwierząt do poszczególnych klas na podstawie ich cech oraz zbadaniem wpływu zmiany parametrów sieci na proces uczenia. Projekt został zrealizowany w języku Python z użyciem biblioteki nnet_jit.

1.2 Zestaw danych

Zestaw danych uczących [8] zawiera 101 instancji, 18 cech oraz 7 klas. W owym zestawie nie ma danych brakujących oraz nieokreślonych, dane są nieuporządkowane.

Cechy zwierząt:

1. nazwa zwierzęta,
2. posiadanie włosów,
3. posiadanie piór,
4. jajorodność,
5. posiadanie mleka,
6. posiadanie zdolności do lotu,
7. żyjące w wodzie,
8. drapieżność,
9. posiadanie uzębienia,
10. posiadanie kręgosłupa,
11. oddychanie,
12. jadowitość,
13. posiadanie płetw,
14. ilość odnóży,
15. posiadanie ogona,
16. udomowienie,
17. kotowatość,
18. przynależność do klasy

Klasy:

1. ssaki (41),
2. ptaki (20),
3. gady (5),
4. ryby (13),
5. płazy (4),
6. owady (8),
7. mięczaki (10)

Liczba w nawiasach obok klasy zwierząt określa liczność każdego z zbiorów. Zbiór wejściowy będą stanowiły cechy od 2 do 17, natomiast zbiór wyjściowy cecha 18. Cecha pierwsza nie będzie używane do zbudowania zbioru uczącego. Wszystkie wartości wejściowe poza ilością odnóży nie wymagają normalizacji.

1.3 Przygotowanie danych

Dane wejściowe przekształcamy do postaci 16 wierszy odpowiadających cechom oraz 101 kolumn będących instancjami. Następnie przeprowadzamy normalizację danych do przedziału [-1,1].

Normalizowanie danych odbywa się poprzez zależność:

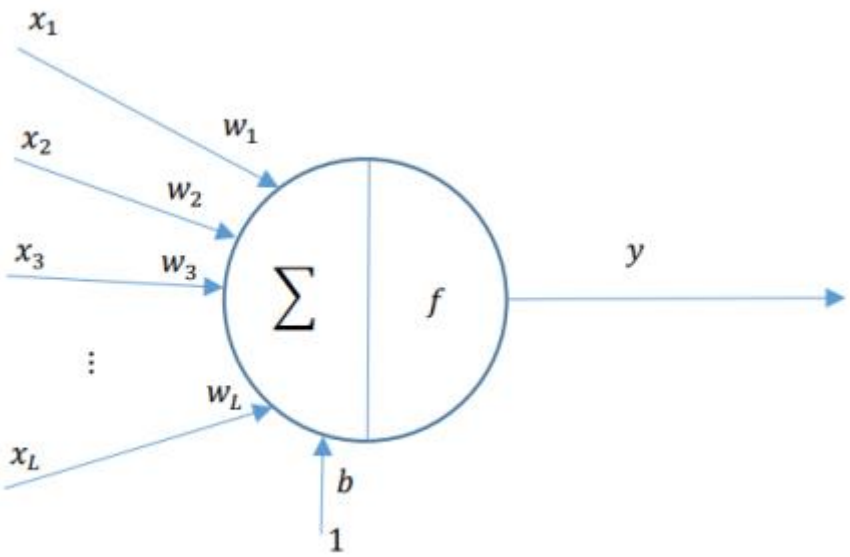
$$Xnorm = \frac{(Ymax - Ymin) * (X - Xmin)}{(Xmax - Xmin)} + Ymin$$

Gdzie:

- X_{norm} - wartość znormalizowana cechy
- X - wartość nieznormalizowana cechy
- X_{max} - maksymalna wartość nieznormalizowana cechy
- X_{min} - minimalna wartość nieznormalizowana cechy
- Y_{max} - maksymalna wartość po normalizacji
- Y_{min} – minimalna wartość po normalizacji

2. Zagadnienia teoretyczne

2.1. Model neuronu



Rys. 1. Model neuronu McCullocha-Pittsa.[4]

Neuron McCullocha-Pittsa jest złożony z dwóch części. Pierwsza z nich składa się z sumy iloczynów poszczególnych wejść z odpowiadającą mu wagą, oraz biasu. Jest ona nazywana łącznym pobudzeniem neuronu. Drugą częścią neuronu jest funkcja przejścia, która określa wyjście neuronu, zależnie od wartości łącznego pobudzenia neuronu.

Sygnał wyjściowy neuronu możemy określić następująco:

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

j-ty sygnał wejściowy neuronu (j = 1,2,...L) - x_j,
współczynnik wagowy poszczególnego wejścia (waga) - w_j,
sygnał wyjściowy neuronu - y,
bias - b,
funkcja przejścia neuronu – f

Przy przyjęciu zapisu macierzowego:

- wektor sygnałów wejściowych - $x = [x_1, x_2, \ldots, x_L]^T$,
- macierz wierszowa wag - $w = [w_1, w_2, \ldots, w_L]^T$

Otrzymujemy następujący wzór:

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

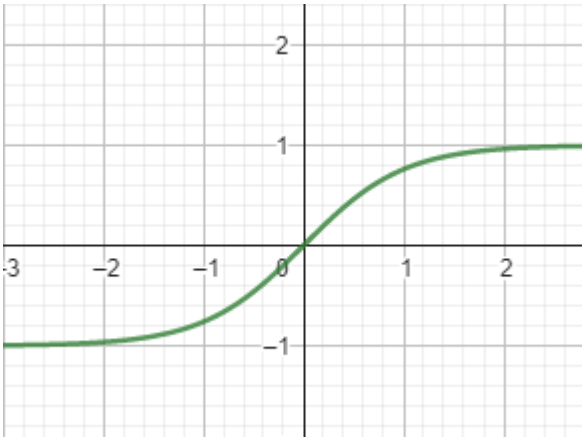
„Ważona suma wejść wraz z przesunięciem często bywa nazywana łącznym pobudzeniem neuronu i w dalszych rozważaniach oznaczana będzie symbolem z.”[3]

Otrzymujemy następujący wzór:

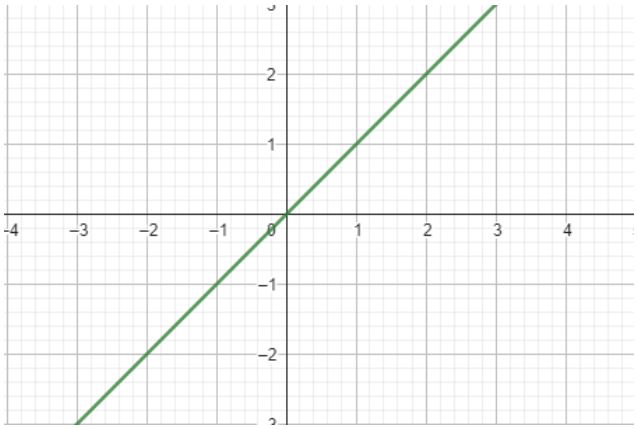
$$y = f(z)$$

2.2. Funkcje przejścia

W sztucznej inteligencji, funkcja przejścia neuronu (określana również jako funkcja aktywacji neuronu), przyjmuje jako argument łączne pobudzenie neuronu i zwraca wyjście zgodnie z charakterystyką danej funkcji przejścia. Sieci jednokierunkowe wielowarstwowe najczęściej wykorzystują funkcje przejścia typu sigmoidalnego na przykład tangens hiperboliczny, w warstwie wejściowej oraz ukrytych. Ze względu na zbyt wąski zakres (-1,1), nie stosuje się ich do warstwy wyjściowej, tylko korzysta się z funkcji liniowej, która nie posiada takich ograniczeń.[4]



Rys. 2. Wykres funkcji przejścia tangensa hiperbolicznego.



Rys. 3. Wykres funkcji przejścia liniowej.

Wzór funkcji tansig:

$$f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Wzór funkcji liniowej:

$$f(z) = z$$

W sieciach neuronowych wykorzystujemy pochodną funkcji przejścia w algorytmie wstecznej propagacji, dlatego zostanie teraz wyprowadzona:

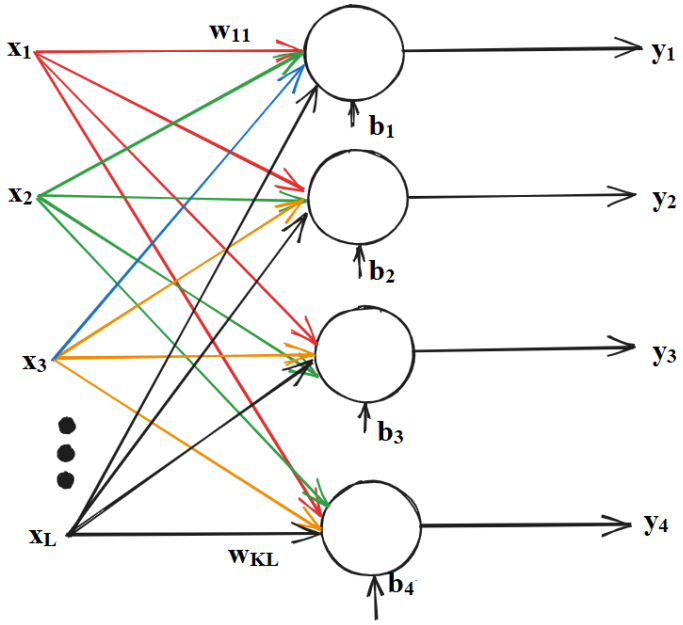
Pochodna funkcji przejścia tansig:

$$f'(z) = \frac{(e^z + e^{-z}) - (-e^{-z})(e^z - e^{-z})}{(e^z + e^{-z})^2} = \frac{(e^z + e^{-z})^2 - (e^z - e^{-z})^2}{(e^z + e^{-z})^2} =$$
$$= 1 - \frac{(e^z - e^{-z})^2}{(e^z + e^{-z})^2} = 1 - f(z)^2$$

Pochodna funkcji przejścia purelin (liniowej)

$$f'(z) = 1$$

2.3. Sieci neuronowe jednokierunkowe jednowarstwowe



Rys. 4. Sieć jednokierunkowa jednowarstwowa.

Sieć jednokierunkowa jednowarstwowa ma następujące cechy: występuje w niej jedna warstwa neuronów oraz przepływ sygnałów odbywa się od wejścia do wyjścia.[5]
Wzór opisujący wyjście jednokierunkowej, jednowarstwowej sieci neuronowej wyraża się następującym wzorem:

$$y = f(wx + b)$$

Gdzie:

Wektor sygnałów wejściowych – $\mathbf{x} = [x_1, x_2,\ldots,x_K]^T$,
wektor przesunięć - $\mathbf{b} = [b_1,b_2,\ldots b_K]^T$,
wektor sygnałów wyjściowych – $\mathbf{y} = [y_1,y_2,\ldots,y_K]^T$,
prostokątna macierz wag o wymiarach K na L - **w**
K - numeruje kolejne neurony w warstwie,
L – numeruje sygnały wejściowe

Uczenie pod nadzorem pojedynczej sieci neuronowej polega na takim doborze wag, aby uzyskać odwzorowanie danych wejściowych **x** z określonym błędem. Uczenie pod nadzorem oznacza, że każdemu wektorowi wejściowemu odpowiada pożądany wektor sygnałów wyjściowych. Zmiana wag odbywa się w kolejnych epokach **t**. Dla j-tej wagi i-tego neuronu, zmianę wag opisujemy zależnością:

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

Błąd popełniany przez sieć wyraża się następująco:

$$e = y - \acute{y}$$

Gdzie:

pożądane wyjście – ý
wyjście sieci – y

Aby doprowadzić do uzyskania odpowiednich wyników na wyjściu, minimalizujemy funkcję celu E, którą przyjmujemy jako sumaryczny błąd kwadratowy SSE.

$$E = \frac{1}{2}\sum_{i=1}^{K}(e_i)^2$$

Poszukujemy minimum metodą gradientową:

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

Wyznaczając wzór uczenia gradientowego dla i-tego neuronu i j-tej wagi mamy:

$$\Delta w = -\eta \frac{\partial E}{\partial w_{ij}}$$

Dzięki zależności

$$\mathrm{E} = \mathrm{E}(\,\mathrm{y_i}\,(\,\mathrm{z_i}\,(\,\mathrm{w_{ij}}\,)\,)\,)$$

Otrzymujemy:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial z_i}\frac{\partial z_i}{\partial w_{ij}}$$

Po dokonaniu analizy poprzednich wzorów, możemy dojść do wniosku, iż:

$$\frac{\partial E}{\partial y_i} = y_i - \acute{y} = e_i$$

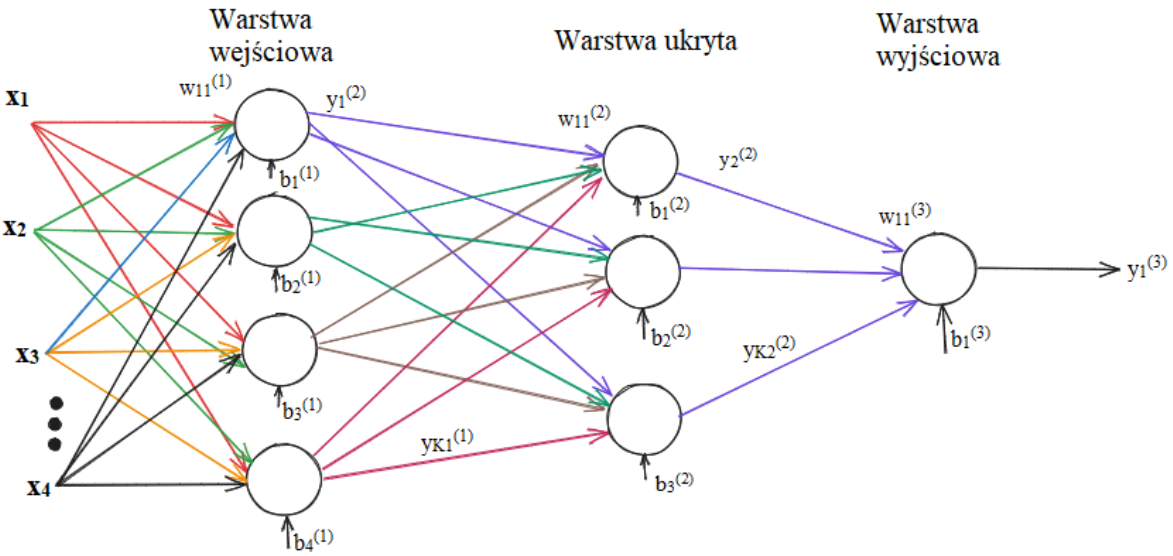
Następnie obliczamy:

$$\frac{z_{ij}}{\partial w_{ij}} = x_j$$

Podstawiając zależności otrzymamy regułę uczenia Widrowa-Hoffa w postaci ogólnej:

$$\Delta w_{ij}(t) = \eta(\acute{y} - y_i)\frac{\partial y_i}{\partial z_i}x_j$$

2.4. Sieci neuronowe jednokierunkowe wielowarstwowe



Rys. 5. Sieć neuronowa jednokierunkowa wielowarstwowa.

Sieć jednokierunkowa wielowarstwowa (MLP) jest jednym z podstawowych modeli sztucznych sieci neuronowych. Składa się z co najmniej trzech warstw: warstwy wejściowej, jednej lub więcej warstw ukrytych oraz warstwy wyjściowej. Sieć ta jest "jednokierunkowa", ponieważ sygnały przepływają przez sieć od warstwy wejściowej do warstwy wyjściowej bez żadnych cykli.

Warstwa wejściowa jest to pierwsza warstwa sieci, która przyjmuje wejście lub dane wejściowe. Każdy neuron w tej warstwie reprezentuje jedną zmienną wejściową. MLP może mieć jedną lub więcej warstw ukrytych. Neurony w tych warstwach nie mają bezpośredniego dostępu do danych wejściowych ani do wyników końcowych. Są one odpowiedzialne za przetwarzanie informacji i ekstrakcję cech z danych wejściowych. Warstwa wyjściowa jest to ostatnia warstwa sieci, która generuje wyniki predykcji lub klasyfikacji.

Sieci jednokierunkowe wielowarstwowe używają funkcji sigmoidalnych w warstwie wejściowej oraz ukrytych i funkcji liniowej w warstwie wyjściowej, ze względu na fakt zbyt małego zakresu funkcji sigmoidalnych.

Działanie poszczególnych warstw opisane jest przez:

$$y^{(1)}=f^{(1)}(w^{(1)}x+b^{(1)}),$$

$$y^{(2)}=f^{(2)}(w^{(2)}y^{(1)}+b^{(2)}),$$

$$y^{(3)}=f^{(3)}(w^{(3)}y^{(2)}+b^{(3)}).$$

Działanie całej sieci jest opisane wzorem:

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

2.5. Algorytm wstecznej propagacji błędu

Proces uczenia się sieci neuronowej za pomocą algorytmu wstecznej propagacji błędu składa się z dwóch głównych faz: propagacji w przód i propagacji wstecznej.

W fazie propagacji w przód, dane wejściowe są przekazywane przez sieć neuronową, a każdy neuron oblicza ważoną sumę swoich wejść i przekazuje wynik przez funkcję aktywacji, generując wartość wyjściową. Proces ten powtarza się warstwa po warstwie, aż dotrzemy do warstwy wyjściowej, gdzie uzyskujemy ostateczne wyjście sieci. Po uzyskaniu wyjścia sieci, następuje faza propagacji wstecznej, w której błąd między otrzymanym wyjściem a oczekiwanym wyjściem jest obliczany. Ten błąd jest następnie propagowany wstecz przez sieć, a każdy neuron oblicza swoją cząstkową pochodną błędu względem swoich wag.

Algorytm wstecznej propagacji błędu umożliwia sieciom neuronowym "uczenie się" na podstawie przykładów treningowych i dostosowywanie swoich wag w celu wykonania pożądanych predykcji.

Proces ten matematycznie wygląda następująco [4]:

Będziemy korzystali poniższego wzoru:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$$
$$w_{ij}(t+1) = w_{ij}(t) + -\eta \frac{\partial E}{\partial w_{ij}}$$

Funkcją celu jest SSE.

$$E = \frac{1}{2}\sum_{i_3=1}^{K_3}(e_{i_3})^2 = \frac{1}{2}\sum_{i_3=1}^{K_3}(y_{i3}^{(3)} - \acute{y}_{i3})^2 =$$

$$= \frac{1}{2}\sum_{i_3=1}^{K_3}(f^{(3)}\sum_{i_2=1}^{K_2} w_{i3i2}^{(3)} y_{i2} + b_{i3}^{(3)}) - \acute{y}_{i3})^2 =$$

$$= \frac{1}{2}\sum_{i_3=1}^{K_3}(f^{(3)}\sum_{i_2=1}^{K_2} w_{i3i2}^{(3)} f^{(2)}(\sum_{i_1=1}^{K_1} w_{i2i1}^{(2)} f^{(1)}(\sum_{\mathrm{j}=1}^{\mathrm{L}} w_{i1j}^{(1)} x_j + b_{i1}^{(1)}) + b_{i2}^{(2)}) + b_{i3}^{(3)} - \acute{y}_{i3})^2$$

Wyznaczenie gradientu względem warstwy wejściowej:
Gdzie:
Liczba neuronów warstwy pierwszej - i_1
Liczba neuronów warstwy drugiej – i_2
Liczba neuronów warstwy trzeciej – i_3
Liczba wejść – j

$$\frac{\partial E}{\partial w_{i1j}^{(1)}} = \frac{\partial E}{\partial f_{i3}^{(3)}} \frac{\partial f^{(3)}(z_{i3}^{(3)})}{\partial z_{i3}^{(3)}} \frac{\partial z_{i3}^{(3)}}{\partial f_{i2}^{(2)}} \frac{\partial f^{(2)}(z_{i2}^{(2)})}{\partial z_{i2}^{(2)}} \frac{\partial z_{i2}^{(2)}}{\partial f_{i1}^{(1)}} \frac{\partial f^{(1)}(z_{i1}^{(1)})}{\partial z_{i1}^{(1)}} \frac{\partial z_{i1}^{(1)}}{\partial w_{i1j}^{(1)}} =$$

$$= \sum_{i_3=1}^{K_3} (y_{i3}^{(3)} - \acute{y}_{i3}) \frac{\partial f^{(3)}(z_{i3}^{(3)})}{\partial z_{i3}^{(3)}} \sum_{i_2=1}^{K_2} w_{i3i2}^{(3)} \frac{\partial f^{(2)}(z_{i2}^{(2)})}{\partial z_{i2}^{(2)}} w_{i2i1}^{(2)} \frac{\partial f^{(1)}(z_{i1}^{(1)})}{\partial z_{i1}^{(1)}} x_j$$

Gdzie:

$$\frac{\partial z_{i2}^{(2)}}{\partial f_{i1}^{(1)}} = w_{i2i1}^{(2)}$$

$$\frac{\partial f^{(3)}(z_{i3}^{(3)})}{\partial z_{i3}^{(3)}} = 1$$

$$\frac{\partial f^{(2)}(z_{i1}^{(2)})}{\partial z_{i1}^{(2)}} = (1 - (y_{i1}^{(2)})^2)$$

Przykład 1

Policzymy wartość wagi $w_{22}^{(2)}$ w kolejnej epoce:

$$\frac{\partial E}{\partial w_{22}^{(2)}} = \frac{\partial E}{\partial f^{(3)}} \frac{\partial f^{(3)}(z_{i3}^{(3)})}{\partial z_{i3}^{(3)}} \frac{\partial z_{i3}^{(3)}}{\partial f^{(2)}} \frac{\partial f^{(2)}(z_2^{(2)})}{\partial z_2^{(2)}} \frac{\partial z_2^{(2)}}{\partial w_{22}^{(2)}} =$$

$$= (y_1^{(3)} - \acute{y}_1) \frac{\partial f^{(3)}(z_1^{(3)})}{\partial z_1^{(3)}} w_{12}^{(3)} \frac{\partial f^{(2)}(z_2^{(2)})}{\partial z_2^{(2)}} y_2^{(2)} =$$

$$= e_1^{(3)} \cdot 1 \cdot w_{12}^{(3)} (1 - (y_2^{(2)})^2)\, y_2^{(2)}$$

Podstawiamy do wzoru:

$$w_{22}^{(2)}(t+1) = w_{22}(t) + \, -\eta (e_1^{(3)} \cdot 1 \cdot w_{12}^{(3)} (1 - (y_2^{(2)})^2)\, y_2^{(2)})$$

Przykład 2

Policzymy wartość wagi $w_{12}^{(1)}$ w kolejnej epoce:

$$\frac{\partial E}{\partial w_{12}^{(1)}} = \frac{\partial E}{\partial f_{i3}^{(3)}} \frac{\partial f^{(3)}(z_{i3}^{(3)})}{\partial z_{i3}^{(3)}} \frac{\partial z_{i3}^{(3)}}{\partial f_{i2}^{(2)}} \frac{\partial f^{(2)}(z_{i2}^{(2)})}{\partial z_{i2}^{(2)}} \frac{\partial z_{i2}^{(2)}}{\partial f_{1}^{(1)}} \frac{\partial f^{(1)}(z_{1}^{(1)})}{\partial z_{1}^{(1)}} \frac{\partial z_{1}^{(1)}}{\partial w_{12}^{(1)}} =$$

$$= \sum_{i_3=1}^{K_3} (y_{i3}^{(3)} - \acute{y}_{i3}) \frac{\partial f^{(3)}(z_{i3}^{(3)})}{\partial z_{i3}^{(3)}} \sum_{i_2=1}^{K_2} w_{i3i2}^{(3)} \frac{\partial f^{(2)}(z_{i2}^{(2)})}{\partial z_{i2}^{(2)}} w_{i21}^{(2)} \frac{\partial f^{(1)}(z_{1}^{(1)})}{\partial z_{1}^{(1)}} x_2 =$$

$$= e_1 \cdot 1 \cdot (\ w_{11}^{(3)} \cdot \left(1 - \left(y_2^{(2)}\right)^2\right) \cdot w_{11}^{(3)} + \ldots) \cdot (1 - (y_1^{(1)})^2)\ \cdot x_2$$

Podstawiamy do wzoru:

$$w_{12}^{(1)}(t+1) = e_1 \cdot 1 \cdot (\ w_{11}^{(3)} \cdot \left(1 - \left(y_2^{(2)}\right)^2\right) \cdot w_{11}^{(3)} + \ldots) \cdot (1 - (y_1^{(1)})^2)\ \cdot x_2$$

2.6 Adaptacyjny współczynnik uczenia

Adaptacyjny współczynnik uczenia jest techniką, która polega na dynamicznym dostosowaniu wartości współczynnika uczenia w trakcie uczenia. Zamiast ustalać stałą wartość współczynnika uczenia, adaptacyjna metoda dostosowuje go w oparciu o obserwacje dotyczące tempa zmiany funkcji celu. W praktyce oznacza to, że w trakcie uczenia, algorytm samodzielnie reguluje wartość współczynnika uczenia w zależności od sytuacji.

„Analogie do niej można znaleźć w postępowaniu ludzi – w tym wypadku nauczycieli, którzy przyspieszają proces uczenia, gdy mają zdolnych uczniów, szybko „chwytających" wiedzę, a łagodnieją, gdy natrafią na bardziej „oporną" klasę lub skomplikowane zagadnienie, przy którym nawet zdolni uczniowie mają problemy ze zrozumieniem".[6]

W tej metodzie, będziemy używać SSE w celu porównywania różnic między epokami:

$$\eta(t+1) = \begin{cases} \eta(t) \cdot dec, gdy\ SSE(t) > er \cdot SSE(t-1) \\ \eta(t) \cdot inc, gdy\ SSE(t) < er \cdot SSE(t-1) \end{cases}$$

Gdzie:

współczynnik uczenia – (**η**),
epoka uczenia – (**t**),
współczynnik zmniejszania stałej uczenia – (**dec**),
współczynnik zwiększania stałej uczenia – (**inc**),
dopuszczalna krotność błędu – (**er**)

2.7. Metoda momentum

Metoda momentum jest popularnym algorytmem optymalizacji używanym do uczenia sieci trójwarstwowych i innych rodzajów sieci neuronowych. Służy do przyspieszenia procesu uczenia, poprzez wprowadzenie momentu do aktualizacji wag. Podczas aktualizacji wag, obliczany jest gradient funkcji celu dla obecnych danych uczących. Następnie, używając współczynnika momentum, obliczane jest również "momentum" - skumulowany krok, który bierze pod uwagę wcześniejsze aktualizacje wag.

"Często porównuje się proces uczenia sieci neuronowych do toczenia kamienia po górskim zboczu. Metoda momentum można sobie wyobrazić jako nadanie kamieniowi pewnego impetu. Gdy kamień zbliża się do wąskich dolin, impet pomaga mu przejść przez te obszary płaskie, które mogą spowolnić jego ruch, i kontynuować szybsze opadanie w dół zbocza, osiągając lepsze minimum lokalne".[7]

$$w_{ij}(t+1) = w_{ij}(t) + \ -\eta \frac{\partial E}{\partial w_{ij}} + \alpha(w_{ij}(t) - w_{ij}(t-1))$$

Gdzie **α** to współczynnik momentum, który przyjmuje wartości w zakres [0,1]. Najczęściej wykorzystuje się ten współczynnik na poziomie 0.95. Natomiast **t** to kolejna epoka uczenia.

3. Skrypt programu

3.1 Treść skryptu

```
import hickle as hkl
import numpy as np
import nnet_jit as net
import matplotlib.pyplot as plt
from sklearn.model_selection import StratifiedKFold

class mlp_ma_3w:
def __init__(self, x, y_t, K1, K2, K3, lr, err_goal, \
                disp_freq, mc, ksi_inc, ksi_dec, er, max_epoch):
        self.x          = x
        self.L          = self.x.shape[0]
        self.y_t        = y_t
        self.K1         = K1
        self.K2         = K2
        self.K3         = K3
        self.lr         = lr
        self.err_goal   = err_goal
        self.disp_freq  = disp_freq
        self.mc         = mc
        self.ksi_inc    = ksi_inc
        self.ksi_dec    = ksi_dec
        self.er         = er
        self.max_epoch  = max_epoch
        self.SSE_vec = []
        self.PK_vec = []

        self.w1, self.b1 = net.nwtan(self.K1, self.L)
        self.w2, self.b2 = net.nwtan(self.K2, self.K1)
        self.w3, self.b3 = net.rands(self.K3, self.K2)
 hkl.dump([self.w1,self.b1,self.w2,self.b2,self.w3,self.b3],
'wagi3w.hkl')
         self.w1,self.b1,self.w2,self.b2,self.w3,self.b3 =
hkl.load('wagi3w.hkl')
        self.w1_t_1, self.b1_t_1, self.w2_t_1, self.b2_t_1, self.w3_t_1,
self.b3_t_1 = \
            self.w1, self.b1, self.w2, self.b2, self.w3, self.b3
        self.SSE = 0
        self.lr_vec = list()


    def predict(self,x):
        n = np.dot(self.w1, x)
        self.y1 = net.tansig( n,  self.b1*np.ones(n.shape))
        n = np.dot(self.w2, self.y1)
        self.y2 = net.tansig( n,  self.b2*np.ones(n.shape))
        n = np.dot(self.w3, self.y2)
        self.y3 = net.purelin(n, self.b3*np.ones(n.shape))
        return self.y3
```

```
def train(self, x_train, y_train):
        for epoch in range(1, self.max_epoch+1):
            self.y3 = self.predict(x_train)
            self.e = y_train - self.y3

            self.SSE_t_1 = self.SSE
            self.SSE = net.sumsqr(self.e)
            self.PK = (1 -
sum((abs(self.e)>=0.5).astype(int)[0])/self.e.shape[1] ) * 100
            self.PK_vec.append(self.PK)
            if self.SSE < self.err_goal or self.PK == 100:
                break

            if np.isnan(self.SSE):
                break
            else:
                if self.SSE > self.er * self.SSE_t_1:
                    self.lr *= self.ksi_dec
                elif self.SSE < self.SSE_t_1:
                    self.lr *= self.ksi_inc
            self.lr_vec.append(self.lr)

            self.d3 = net.deltalin(self.y3, self.e)
            self.d2 = net.deltatan(self.y2, self.d3, self.w3)
            self.d1 = net.deltatan(self.y1, self.d2, self.w2)
            self.dw1, self.db1 = net.learnbp(self.x,  self.d1, self.lr)
            self.dw2, self.db2 = net.learnbp(self.y1, self.d2, self.lr)
            self.dw3, self.db3 = net.learnbp(self.y2, self.d3, self.lr)

            self.w1_temp, self.b1_temp, self.w2_temp, self.b2_temp,
self.w3_temp, self.b3_temp = \
            self.w1.copy(), self.b1.copy(), self.w2.copy(),
self.b2.copy(), self.w3.copy(), self.b3.copy()

            self.w1 += self.dw1 + self.mc * (self.w1 - self.w1_t_1)
            self.b1 += self.db1 + self.mc * (self.b1 - self.b1_t_1)
            self.w2 += self.dw2 + self.mc * (self.w2 - self.w2_t_1)
            self.b2 += self.db2 + self.mc * (self.b2 - self.b2_t_1)
            self.w3 += self.dw3 + self.mc * (self.w3 - self.w3_t_1)
            self.b3 += self.db3 + self.mc * (self.b3 - self.b3_t_1)

            self.w1_t_1, self.b1_t_1, self.w2_t_1, self.b2_t_1,
self.w3_t_1, self.b3_t_1 = \
            self.w1_temp, self.b1_temp, self.w2_temp, self.b2_temp,
self.w3_temp, self.b3_temp

            self.SSE_vec.append(self.SSE)
```

```
x,y_t,x_norm,x_n_s,y_t_s = hkl.load('zoo.hkl')

max_epoch = 2000
err_goal = 0.25
disp_freq = 10
lr = 1e-4
mc = 0.95
ksi_inc = 1.05
ksi_dec = 0.7
er = 1.04
K1 = 7
K2 = 3

data = x_n_s.T
target = y_t_s
K3 = target.shape[0]

CVN = 4
skfold = StratifiedKFold(n_splits=CVN)
PK_vec = np.zeros(CVN)
```

Skrypt używany do eksperymentów. [9]

3.2 Analiza skryptu

Część konstruktora klasy:
class mlp_ma_3w: - konstruktor klasy

self.x = x - macierz danych wejściowych

self.L = self.x.shape[0]-ustawienie atrybutu 'L' klasy na liczbę wierszy w macierzy x

self.y_t = y_t - wektor docelowy (oczekiwany wynik) dla każdej próbki w danych wejściowych 'x'

self.K1 = K1-liczba neuronów w: pierwszej warstwie neuronów,

self.K2 = K2-drugiej warstwie neuronów

self.K3 = K3-trzeciej warstwie neuronów

self.lr = **lr**-szybkość uczenia

self.err_goal = err_goal-docelowy poziom błędu

self.disp_freq = disp_freq-częstotliwość wyświetlania postępu uczenia (liczba epok między kolejnymi wyświetleniami)

self.mc = mc-stała uczenia momentum

self.ksi_inc = ksi_inc-współczynnik przyrostu dla stałej uczenia

self.ksi_dec = ksi_dec-współczynnik spadku dla stałej uczenia

self.er = er-wariancja startowych wartości wag

self.max_epoch = max_epoch-maksymalna liczba epok uczenia

self.SSE_vec = []-pusty wektor do przechowywania błędów

self.PK_vec = []-pusty wektor do przechowywania wskaźników trafienia podczas uczenia sieci,

self.w1, self.b1 = net.nwtan(self.K1, self.L)-tworzenie wagi i biasu dla warstwy pierwszej sieci neuronowej, funkcja nwtan zwraca te wagi i biasy jako losowe,

self.w2, self.b2 = net.nwtan(self.K2, self.K1)- tworzenie wagi i biasu dla warstwy drugiej sieci neuronowej, funkcja nwtan zwraca te wagi i biasy jako losowe,

self.w3, self.b3 = net.rands(self.K3, self.K2)-net.rands zwraca losowe wagi i biasy dla warstwy trzeciej

**self.w1_t_1, self.b1_t_1, self.w2_t_1, self.b2_t_1, self.w3_t_1, self.b3_t_1 = **

self.w1, self.b1, self.w2, self.b2, self.w3, self.b3-tworzy kopie wag i biasów dla każdej z trzech warstw sieci neuronowej i przypisuje je do zmiennych z sufiksem _t_1, co oznacza, że są to wartości z poprzedniej iteracji trenowania.

self.SSE = 0-ustawia sumę kwadratów na początkową wartość równą 0

self.lr_vec = list()-tworzy pustą listę, która będzie przechowywać wartości szybkości uczenia (**lr**) w kolejnych iteracjach trenowania.

Część metody predict:
def predict(self,x):-definiowanie 3 warstw sieci neuronowych, zwraca wyjście sieci

n = np.dot(self.w1, x)-mnoży macierz wag w1 przez dane wejściowe x i przypisuje wynik do n

self.y1 = net.logsig(n, self.b1*np.ones(n.shape))-wywołuje funkcję aktywacji tansig z pakietu net na n, wraz z wektorem biasu b1 i przypisuje wynik do y1

n = np.dot(self.w2, self.y1)-mnoży macierz wag w2 przez wyniki z poprzedniej warstwy y1 i przypisuje wynik do n

self.y2 = net.logsig(n, self.b2*np.ones(n.shape))-wywołuje funkcję aktywacji tansig z pakietu net na n, wraz z wektorem biasu b2 i przypisuje wynik do y2.

n = np.dot(self.w3, self.y2)-mnoży macierz wag w3 przez wyniki z poprzedniej warstwy y2 i przypisuje wynik do n
self.y3 = net.purelin(n, self.b3*np.ones(n.shape))-wywołuje funkcję aktywacji purelin z pakietu net na n, wraz z wektorem biasu b3 i przypisuje wynik do y3

return self.y3-zwraca wyniki z ostatniej warstwy y3 jako macierz danych wyjściowych.

Część metody train:
def train(self, x_train, y_train):-przeprowadza uczenie sieci neuronowej na podstawie danych wejściowych i wejściowych

for epoch in range(1, self.max_epoch+1):-iteracja po epokach treningowych

self.y3 = self.predict(x_train)-wywołanie metody predict, przewidywanie wyników na podstawie danych wejściowych

self.e = y_train - self.y3-obliczenie błędu

self.SSE_t_1 = self.SSE-przypisanie aktualnej wartości SSE do zmiennej pomocniczej

self.SSE = net.sumsqr(self.e)- obliczenie błędu kwadratowego średniokwadratowego

self.PK = (1 - sum((abs(self.e)>=0.5).astype(int)[0])/self.e.shape[1]) * 100-obliczenie PK,

oblicza się sumę przypadków, w których wartość bezwzględna błędu jest większa lub równa 0.5, dzieli przez ilość wszystkich przypadków i odejmuje się tę wartość od 1. Następnie wynik jest mnożony przez 100 i przypisywany do self.PK

self.PK_vec.append(self.PK)-obliczone PK jest dodawane do listy

if self.SSE < self.err_goal or self.PK == 100:
break -warunek zakończenia treningu: Jeśli wartość SSE jest mniejsza od docelowego celu błędu (self.err_goal) lub stopień poprawności wynosi 100, trening zostaje przerwany.
if np.isnan(self.SSE):
break - obsługa błędu : Jeśli wartość SSE jest NaN (Not a Number), trening zostaje przerwany.
else:
if self.SSE > self.er * self.SSE_t_1:
self.lr *= self.ksi_dec
elif self.SSE < self.SSE_t_1:
self.lr *= self.ksi_inc
self.lr_vec.append(self.lr)-Aktualizacja współczynnika uczenia (lr): W zależności od porównania bieżącego SSE (self.SSE) z poprzednim SSE (self.SSE_t_1), współczynnik uczenia (self.lr) jest dostosowywany. Jeśli SSE jest większe niż iloczyn self.er i self.SSE_t_1, to self.lr zostaje pomnożone przez self.ksi_dec. Jeśli SSE jest mniejsze niż self.SSE_t_1, to self.lr zostaje pomnożone przez self.ksi_inc. Wartość self.lr jest dodawana do listy self.lr_vec.

self.d3 = net.deltalin(self.y3, self.e)-oblicza gradient błędu dla warstwy wyjściowej (trzeciej warstwy) sieci neuronowej przy użyciu funkcji deltalin z modułu net. Przekazuje self.y3 (wyniki z trzeciej warstwy) i self.e (różnice między oczekiwanymi wynikami a przewidywanymi wynikami) jako argumenty.

self.d2 = net.deltalog(self.y2, self.d3, self.w3)-oblicza gradient błędu dla drugiej warstwy sieci neuronowej przy użyciu funkcji deltatan z modułu net. Przekazuje self.y2 (wyniki z drugiej warstwy), self.d3 (gradient błędu z trzeciej warstwy) oraz self.w3 (macierz wag między drugą a trzecią warstwą) jako argumenty.

self.d1 = net.deltalog(self.y1, self.d2, self.w2)-Oblicza gradient błędu dla pierwszej warstwy sieci neuronowej przy użyciu funkcji deltatan z modułu net. Przekazuje self.y1 (wyniki z pierwszej warstwy), self.d2 (gradient błędu z drugiej warstwy) oraz self.w2 (macierz wag między pierwszą a drugą warstwą) jako argumenty.

self.dw1, self.db1 = net.learnbp(self.x, self.d1, self.lr)-oblicza zmiany wag i biasu dla pierwszej warstwy sieci neuronowej przy użyciu funkcji learnbp z modułu net. Przekazuje self.x (dane wejściowe), self.d1 (gradient błędu dla pierwszej warstwy) oraz self.lr (współczynnik uczenia) jako argumenty. Zwraca zmiany wag (self.dw1) i biasu (self.db1)

self.dw2, self.db2 = net.learnbp(self.y1, self.d2, self.lr)-Oblicza zmiany wag i biasu dla drugiej warstwy sieci neuronowej przy użyciu funkcji learnbp z modułu net. Przekazuje self.y1 (wyniki z pierwszej warstwy), self.d2 (gradient błędu dla drugiej warstwy) oraz self.lr (współczynnik uczenia) jako argumenty. Zwraca zmiany wag (self.dw2) i biasu (self.db2).

self.dw3, self.db3 = net.learnbp(self.y2, self.d3, self.lr)-oblicza zmiany wag i biasu dla trzeciej warstwy sieci neuronowej przy użyciu funkcji learnbp z modułu net. Przekazuje self.y2 (wyniki z drugiej warstwy), self.d3 (gradient błędu dla trzeciej warstwy) oraz self.lr (współczynnik uczenia) jako argumenty. Zwraca zmiany wag (self.dw3) i biasu (self.db3).

**self.w1_temp, self.b1_temp, self.w2_temp, self.b2_temp, self.w3_temp, self.b3_temp = **
self.w1.copy(), self.b1.copy(), self.w2.copy(), self.b2.copy(), self.w3.copy(), self.b3.copy()-
tworzy kopie aktualnych wag i biasów (tymczasowe zmienne) i przypisuje je do odpowiednich zmiennych. Te tymczasowe zmienne będą używane do aktualizacji wag i biasu, jeśli zostaną spełnione odpowiednie warunki.

self.w1 += self.dw1 + self.mc * (self.w1 - self.w1_t_1)
self.b1 += self.db1 + self.mc * (self.b1 - self.b1_t_1)
self.w2 += self.dw2 + self.mc * (self.w2 - self.w2_t_1)
self.b2 += self.db2 + self.mc * (self.b2 - self.b2_t_1)
self.w3 += self.dw3 + self.mc * (self.w3 - self.w3_t_1)
self.b3 += self.db3 + self.mc * (self.b3 - self.b3_t_1)
aktualizowanie wag i biasów, uwzględniając metodę momentum

**self.w1_t_1, self.b1_t_1, self.w2_t_1, self.b2_t_1, self.w3_t_1, self.b3_t_1 = **
self.w1_temp, self.b1_temp, self.w2_temp, self.b2_temp, self.w3_temp, self.b3_temp -
aktualizuje poprzednie wartości wag i biasu, przypisując im aktualne wartości (tymczasowe zmienne).
self.SSE_vec.append(self.SSE)-dodaje aktualne SSE (suma kwadratów błędów) do listy SSE_vec, która przechowuje SSE dla kolejnych epok uczenia.

Pozostała część kodu:

x,y_t,x_norm,x_n_s,y_t_s = hkl.load('zoo.hkl')-załadowanie pliku i wyłuskanie danych, gdzie:

- x-zbiór cech wyjściowych, cechy w wierszach, 16 wierszy 101 kolumn z wartościami,
- y_t-wyjście pożądane, jednowierszowa tablica 101 kolumn,
- x_norm-dane wejściowe znormalizowane [-1;1],
- x_n_s-dane wejściowe posortowane, znormalizowane,
- y_t_s-wyjście pożądane, posortowane(klasy)

data = x_n_s.T-przypisuje dane wejściowe x_n_s do zmiennej data, przy czym x_n_s jest transponowane

target = y_t_s-przypisuje dane wyjściowe y_t_s do zmiennej target

K3 = target.shape[0]-liczba neuronów w warstwie wyjściowej (1)

CVN = 4-ilość podziałów

skfold = StratifiedKFold(n_splits=CVN)- inicjalizuje obiekt StratifiedKFold z liczbą podziałów równą CVN. StratifiedKFold to metoda walidacji krzyżowej, która zachowuje proporcje klas w każdym podziale, zapewniając bardziej reprezentatywne wyniki.

PK_vec = np.zeros(CVN)-tworzy tablicę zerowej długości CVN i przypisuje ją do zmiennej PK_vec. Ta tablica będzie przechowywać wyniki miary wydajności (PK) dla każdego podziału walidacji krzyżowej.

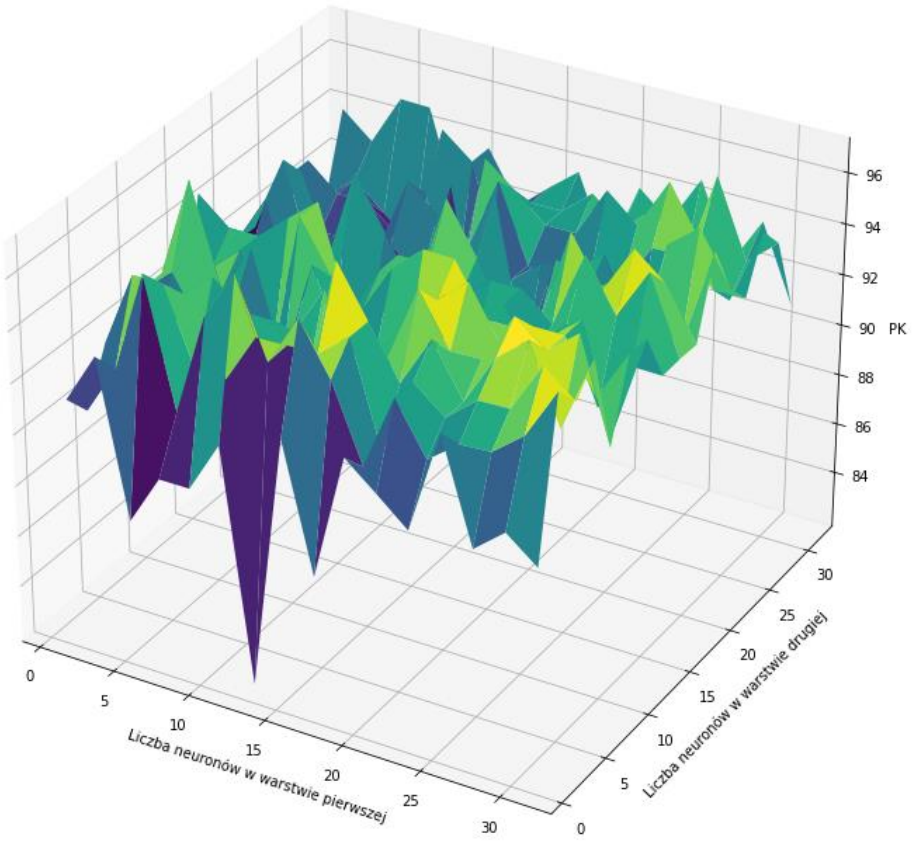
4. Eksperymenty

4.1. Eksperyment 1

Założeniem eksperymentu pierwszego jest wyznaczenie średniej precyzji klasyfikacji dla każdego przypadku w zależności od liczby neuronów w warstwie pierwszej oraz drugiej. Eksperyment został przeprowadzony dla szesnastu pierwszych liczb nieparzystych. Warstwa wejściowa neuronów oraz ukryta zawierają liczby z zakresu 1 – 31.

Parametry określone w eksperymencie:

maksymalna liczba epok uczenia - (**max_epoch**) = 1000,
stała uczenia - (**lr**) = 0.000001,
parametr metody momentum - (**mc**) = 0.95,
liczba neuronów w warstwie pierwszej - (**K1**) = 1-31 z krokiem 2,
liczba neuronów w warstwie ukrytej – (**K2**) = 1-31 z krokiem 2



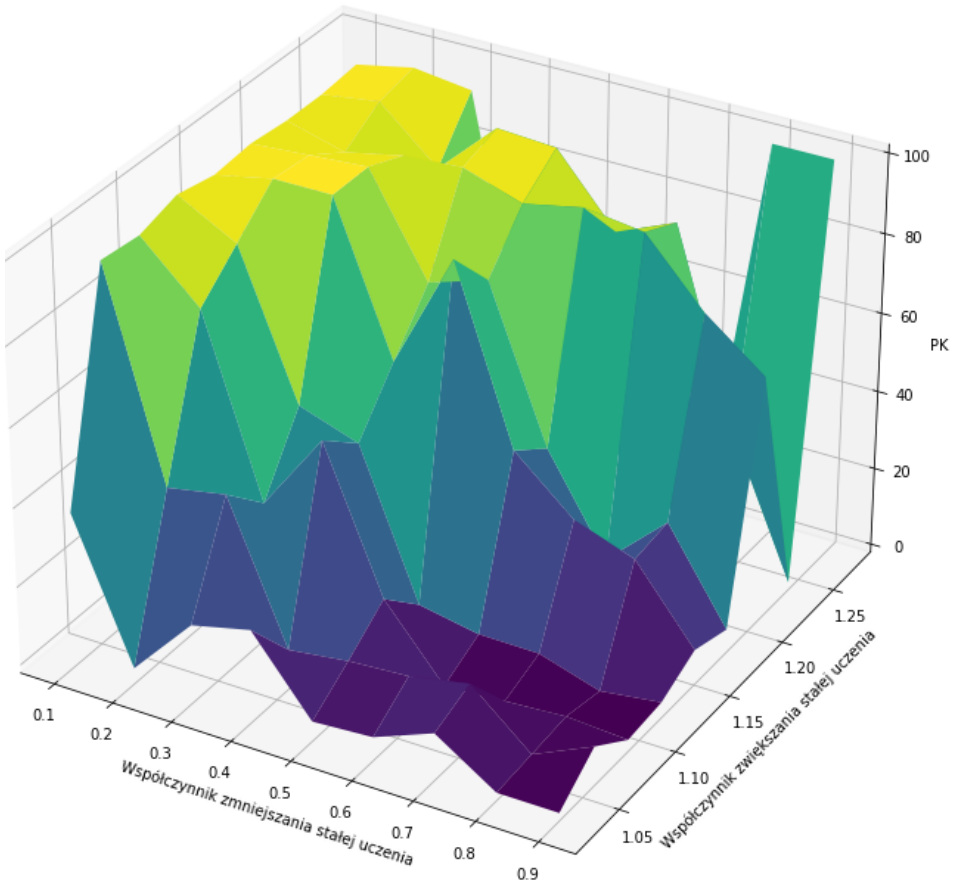
Rys .6. Zależność poprawności klasyfikacji od liczby neuronów w warstwie wejściowej oraz ukrytej

Dla określonych wyżej parametrów wykres jest nieregularny, średnia precyzja klasyfikacji dla wszystkich przypadków wynosi **93,16%**, co jest przyzwoitą skutecznością zakładając ustawiony parametr stałej uczenia na tak niskim poziomie. Wartością maksymalną średniej precyzji klasyfikacji jest **97.45%**.

4.2. Eksperyment 2

Założeniem eksperymentu drugiego jest wyznaczenie średniej precyzji klasyfikacji przy zmianach współczynników zwiększania stałej uczenia oraz zmniejszania stałej uczenia. Parametr zmniejszania stałej uczenia ksi_dec przyjmuje wartości od 0.1 do 0.9 z krokiem 0.1. Parametr zwiększania stałej uczenia ksi_inc przyjmuje wartości od 1.03 do 1.27 z krokiem 0.03.

Parametry określone w eksperymencie:
maksymalna liczba epok uczenia - (**max_epoch**) = 2000,
stała uczenia - (**lr**) = 0.0001,
parametr metody momentum - (**mc**) = 0.95,
liczba neuronów w warstwie pierwszej - (**K1**) = 7,
liczba neuronów w warstwie ukrytej - (**K2**) = 3
współczynnik zmniejszania stałej uczenia - **(ksi_dec**) =od 0.1 do 0.9 z krokiem 0.1
współczynnik zwiększania stałej uczenia - **(ksi_inc**) =od 1.03 do 1.27 z krokiem 0.03



Rys .7. Zależność poprawności klasyfikacji od współczynników zwiększania i zmniejszania stałej uczenia.

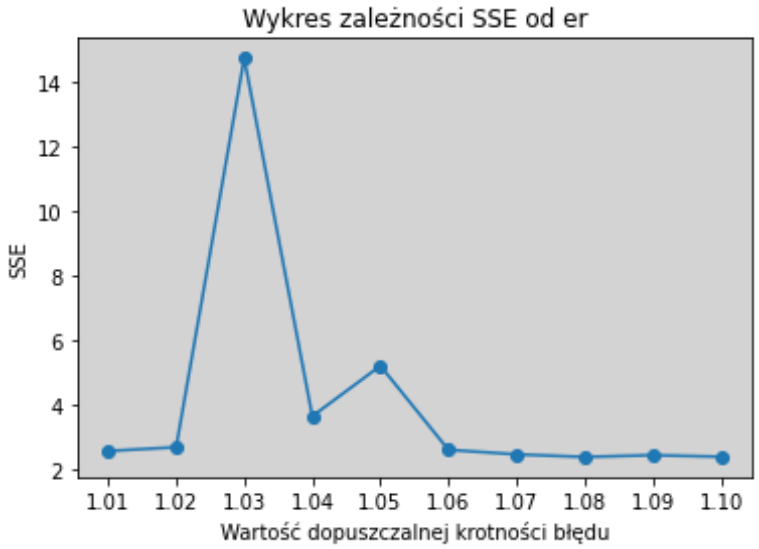
W przypadku eksperymentu drugiego, wartość średnia precyzji klasyfikacji wyniosła **47.9%**. Największą wartość średniej precyzji klasyfikacji **100%**, uzyskano dla par ksi_inc, ksi_dec: **1.24**, **0.9**; **1.27**, **0.9**. Najmniejsza wartość średniej precyzji klasyfikacji wyniosła **0%**, co wynika z szerokiego zakresu zmiany współczynnika zmniejszania stałej uczenia.

Otrzymany wykres zależności PK od ksi_inc oraz ksi_dec jest jednorodny i stopniowy, jednakże sięga on dwóch ekstremów poziomu średniej precyzji klasyfikacji.

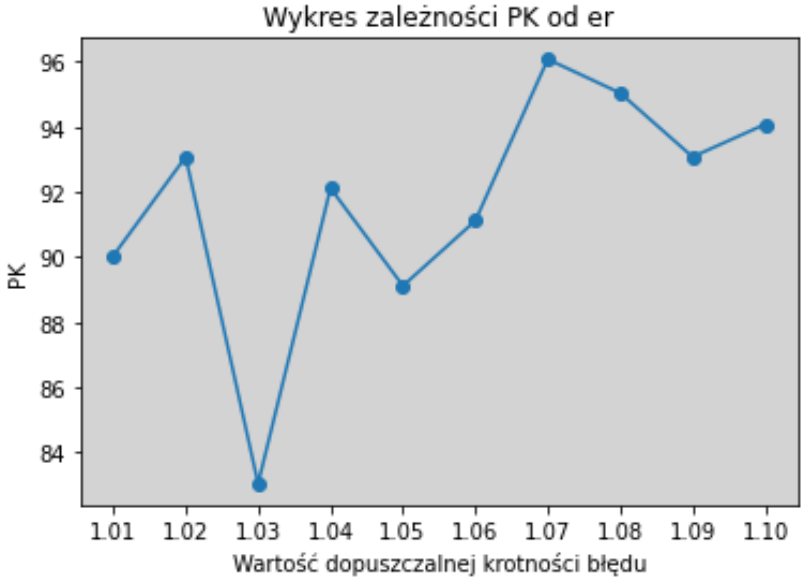
4.3. Eksperyment 3

Eksperyment 3 polega na określeniu średniego SSE oraz PK dla różnych parametrów dopuszczalnej krotności przyrostu błędu. Ten parametr będzie przyjmował wartości od 1.01 do 1.1 z krokiem 0.1.

Parametry określone w eksperymencie:
maksymalna liczba epok uczenia - (**max_epoch**) = 2000,
stała uczenia - (**lr**) = 0.0001,
parametr metody momentum - (**mc**) = 0.95,
liczba neuronów w warstwie pierwszej - (**K1**) = 7,
liczba neuronów w warstwie ukrytej - (**K2**) = 3
współczynnik zmiany błędu - (**er**) = od 1.01 do 1.1 z krokiem 0.1



Rys .8. Zależność sumarycznego błędu kwadratowego (SSE) od dopuszczalnej krotności błędu.



Rys .9. Zależność precyzji klasyfikacji (PK) od dopuszczalnej krotności błędu.

Średnia wartość SSE wyniosła **4.12**, maksymalną wartość sumarycznego błędu kwadratowego uzyskano dla er równego **1.03**, owa wartość wyniosła **14.78**. Najmniejsza

wartość średnia SSE została uzyskana przy er równym **1.08** i wyniosła **2.39**. Podczas analizy wykresu możemy dojść do wniosku, iż, od wartości parametru er równego **1.06** błąd utrzymywał się na zadowalającym poziomie.

Analizując otrzymany wykres zależności średniej precyzji klasyfikacji (PK) od dopuszczalnej krotności błędu (er) możemy zaobserwować, iż dla wartości er równej **1.07**, uzyskujemy największą dokładność, równą **96.07%**. Natomiast najmniejsza dokładność równa **83.03%** została uzyskana dla er równego **1.03**. Średnia wartość PK została wyznaczona na poziomie **91.68%**.

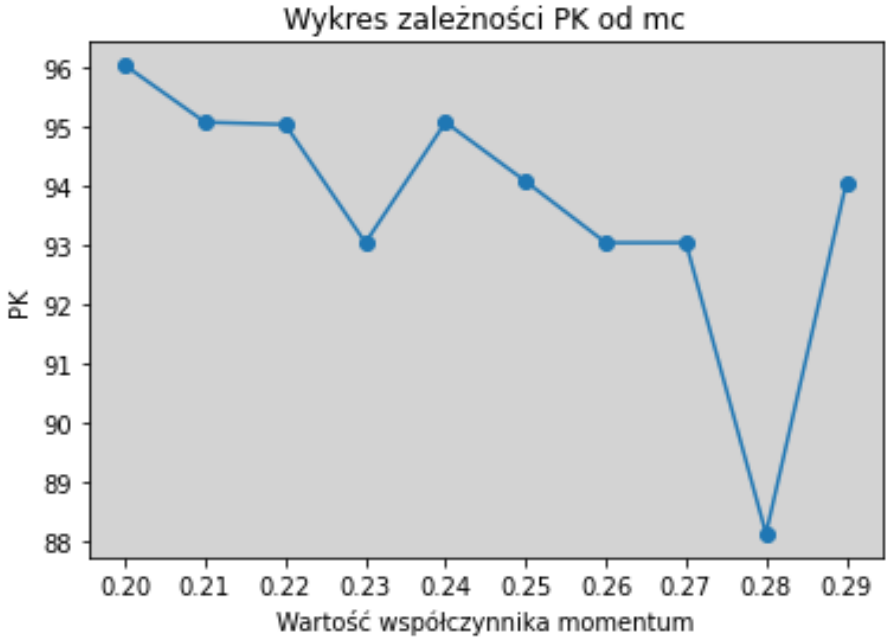
4.4. Eksperyment 4

Eksperyment czwarty będzie dotyczył zmiany parametru PK w zależności od stałej uczenia momentum (mc). Wartość stałej momentum może przyjmować wartości od 0 do 1. Eksperyment został podzielony na trzy etapy: sprawdzanie dokładności uczenia się sieci dla mc należącego do przedziału od 0.2 do 0.29 z krokiem 0.01, następnie od 0.5 do 0.59 z krokiem 0.01 oraz od 0.9 do 0.99. Celem podziału eksperymentu na 3 osobne wykresy jest dokładna analiza 3 zakresów wartości współczynnika mc, który ma spory wpływ na parametr PK.

Parametry określone w eksperymencie:
maksymalna liczba epok uczenia - (**max_epoch**) = 2000,
stała uczenia - (**lr**) = 0.0001,
liczba neuronów w warstwie pierwszej - (**K1**) = 7,
liczba neuronów w warstwie ukrytej - (**K2**) = 3

Etap 1 eksperymentu

Parametr metody momentum - (**mc**) = od 0.20 do 0.29 z krokiem 0.01

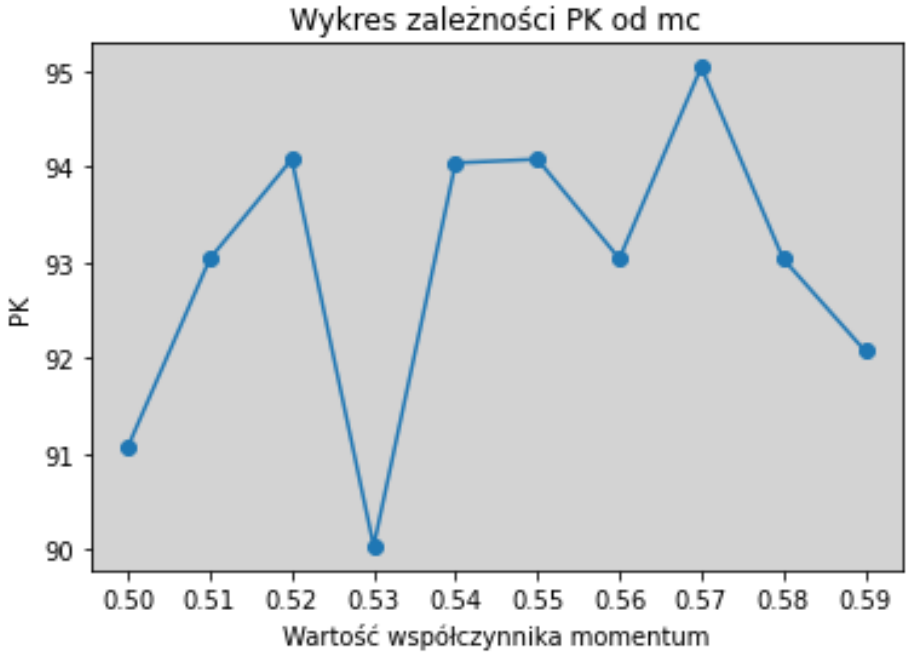


Rys .10. Zależność precyzji klasyfikacji (PK) wartości współczynnika momentum w zakresie [0.20,0.29].

Dokonawszy analizy wykresu, możemy stwierdzić, iż wartość parametru PK przyjmuje wartości na zadowalającym poziomie. Wartość maksymalna współczynnika PK wynosi **96.03%** dla parametru mc równej **0.20**. Wartość minimalna parametru PK wyniosła **88.12%**, przy wartości współczynnika momentum równego **0.28**. Wartość średnia uzyskana dla tego przedziału wartości wyniosła **93.66%**.

Etap 2 eksperymentu

Parametr metody momentum - (**mc**) = od 0.5 do 0.59 z krokiem 0.01

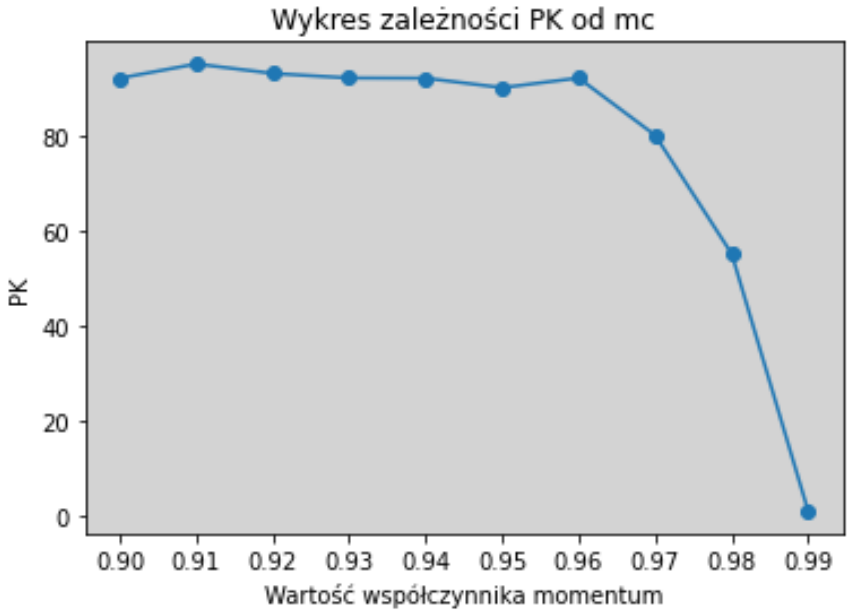


Rys .11. Zależność precyzji klasyfikacji (PK) wartości współczynnika momentum w zakresie [0.50,0.59].

Po dokonaniu analizy wykresu dla drugiej części eksperymentu możemy zauważyć wysoką wartość parametru PK, osiągającą maksymalnie **97.04%** przy współczynniku uczenia równym **0.57**.Wartość minimalna parametru PK wyniosła **92.07%** dla wartości parametru mc równego **0.53**. Wartość średnia parametru PK wyniosła **94.64%**.

Etap 3 eksperymentu

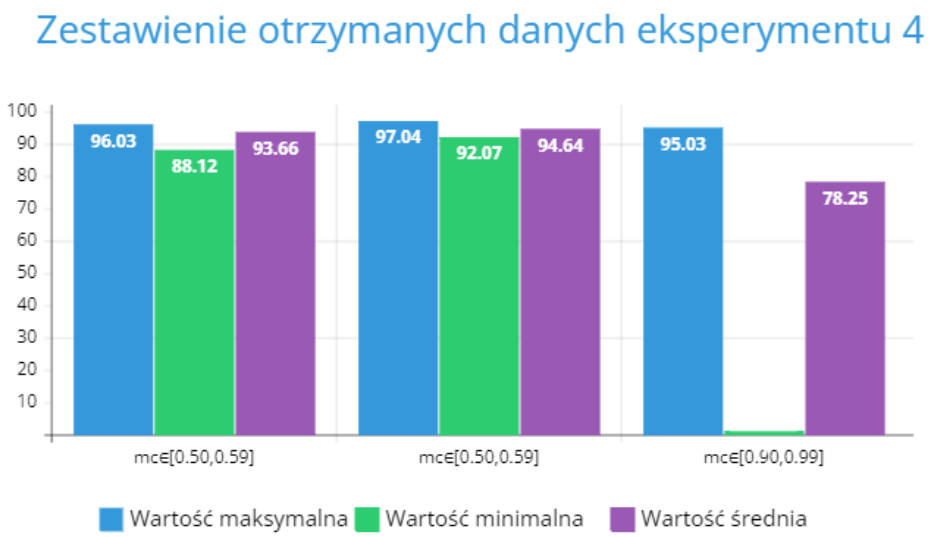
Parametr metody momentum - (**mc**) = od 0.9 do 0.99 z krokiem 0.01



Rys .12. Zależność precyzji klasyfikacji (PK) wartości współczynnika momentum w zakresie [0.90,0.99].

Analizując otrzymany wykres możemy stwierdzić, iż do wartości momentum równej **0.96**, wartość parametru PK pozostaje na podobnym poziomie, jednak powyżej wspomnianej wartości, średnia precyzja klasyfikacji zaczyna drastycznie się zmniejszać. Zakres wartości parametru mc, przy którym uzyskujemy wysoką skuteczność uczenia się sieci możemy określić na **0.90** do **0.96**. Wartość maksymalną PK, uzyskaliśmy dla wartości parametru mc równego **0.91** i jest ona równa **95.03%**. Wartość minimalna, wynosząca **1%** została uzyskana dla wartości współczynnika momentum równej **0.99**. Wartość średnia parametru PK wyniosła **78.25%**.

Wnioski do Eksperymentu 4



Rys .13. Zestawienie wyników szczątkowych eksperymentu 4.

Zgodnie z treścią [2], współczynnik momentum przyjmuje wartości od 0 do 1. Poszukując odpowiedniego współczynnika momentum zostało wybrane po 10 wartości z umownych granic podziału współczynnika momentum w zależności od przeznaczenia. Niskie wartości umownie przyjęte jako zakres: [0,0.3], są używane na początku procesu uczenia, aby umożliwić sieci powolne dostosowanie się do danych i uniknięcie oscylacji. Średnie wartości umownie przyjęte jako zakres [0.4,0.6], są używane w trakcie uczenia, gdy sieć zbliża się do optymalnego rozwiązania. Pomagają przyspieszyć proces uczenia i mogą pomóc w uniknięciu lokalnych minimów. Wysokie wartości określane jako zakres [0.7,1], mogą być używane, gdy sieć utknie w lokalnym minimum i potrzebuje większej dynamiki, aby z niego wyjść.

Gdy dokonamy analizy zestawionych wyników, może się okazać, że największą wartość średnią parametru PK, uzyskaliśmy dla danych wartości współczynnika mc mieszczących się w przedziale [0.50,0.59]. Gdy przyjrzymy się wykresom pierwszej oraz drugiej części eksperymentu, możemy dostrzec najmniejszy rozstrzał w wartościach PK, w przeciwieństwie do eksperymentu trzeciego, gdzie występuje on bardzo duży.

Największa wartość maksymalna została uzyskana w przypadku drugiej części eksperymentu i wyniosła **97.04%**, dla współczynnika momentum równego **0.57**.

Najmniejsza wartość minimalna wynosi **1%** i wystąpiła ona przy trzeciej części eksperymentu, przy założeniu współczynnika momentum równym **0.99**, który jest jednak dużą skrajnością.

Największą średnią z precyzji klasyfikacji uzyskano dla drugiej części eksperymentu i wyniosła ona **96.64%**, a najmniejszą w trzeciej części eksperymentu, wynoszącą **78.25%**.

4.5 Badanie wpływu użycia funkcji przejścia logsig na parametr PK

W eksperymencie piątym, zostanie zbadany wpływ zmiany funkcji przejścia z tansig na logsig na parametr PK. Zostanie przeprowadzone 10 testów, w celu porównania wpływu tej funkcji aktywacji na poprawność klasyfikacji sieci.

Funkcja logsig generuje wartości z zakresu [0,1], natomiast funkcja tansig z zakresu [-1,1]. Podczas propagacji wstecznej, różnice w zakresie wartości mogą wpływać na efektywność propagacji gradientu. Funkcja logsig ma lepsze właściwości gradientowe w pobliżu skrajnych wartości (0 i 1), podczas gdy funkcja tansig ma większą stabilność gradientową, czyli gradienty są w miarę jednorodne i nie podlegają skokom w zakresie [-1, 1].

Wzór funkcji logsig:

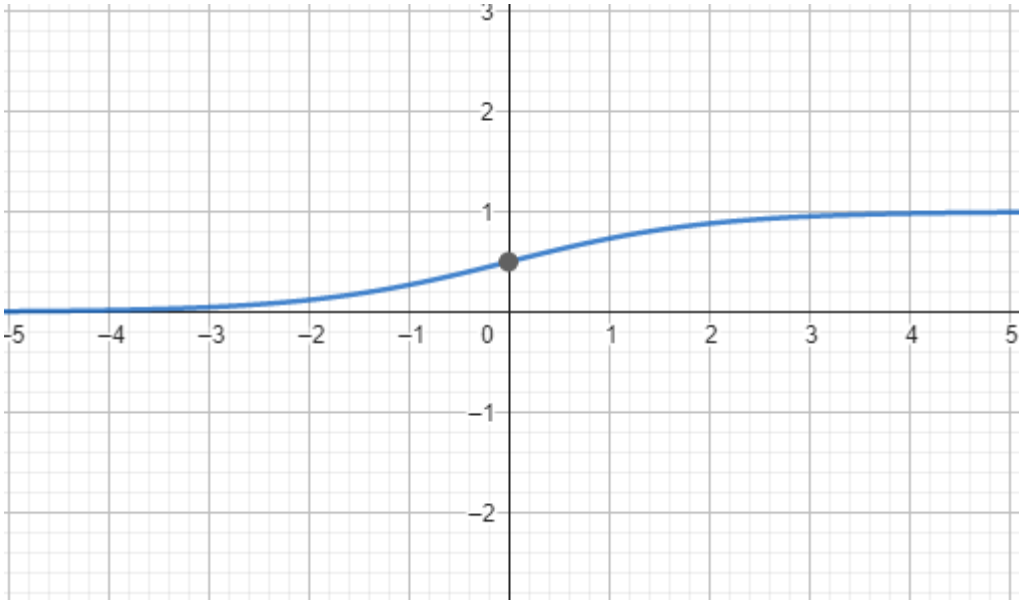
$$f(z) = \frac{1}{1+e^{-z}} = \frac{e^{z}}{(1\cdot e^{z}) + (e^{-z}\cdot e^{z})} = \frac{e^{z}}{1+e^{z}}$$

Pochodna funkcji logsig:

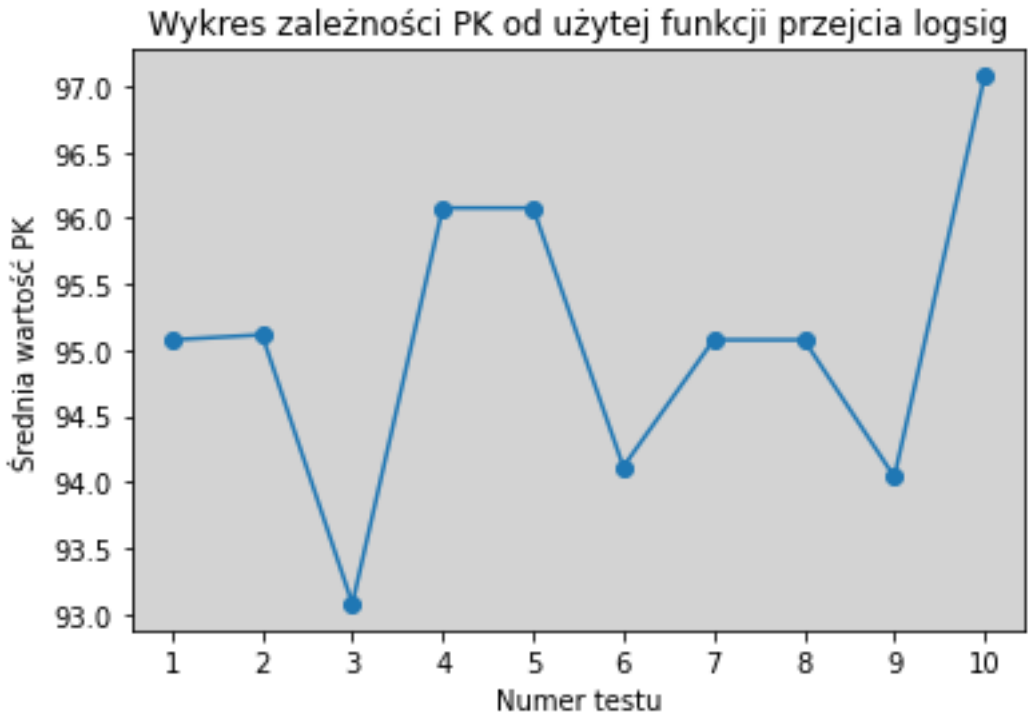
$$f'(z) = \left(\frac{e^{z}}{1+e^{z}}\right)' = \frac{e^{z}(1+e^{z}) - e^{z}\cdot e^{z}}{(1+e^{z})^{2}} =$$

$$= \frac{e^{z} + e^{2z} - e^{2z}}{(1+e^{z})^{2}} = \frac{e^{z}}{(1+e^{z})^{2}} = \frac{e^{z}}{1+e^{z}} \cdot \left(1 - \frac{e^{z}}{1+e^{z}}\right) =$$

$$= f(z)(1-f(z))$$

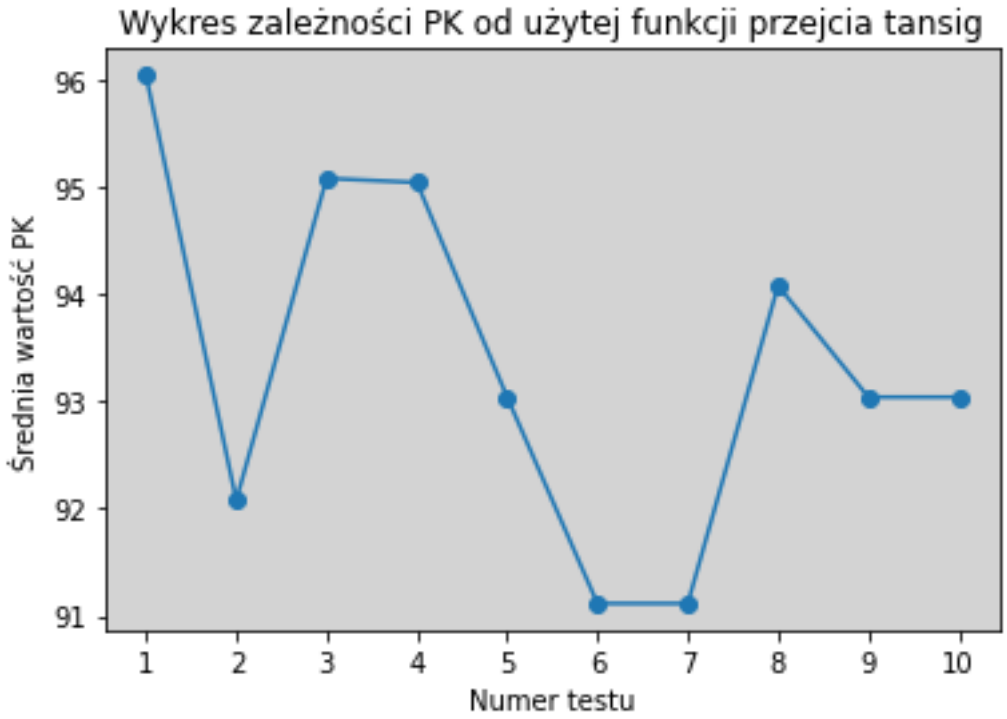


Rys .14. Wykres funkcji logsig.



Rys .15. Wykres zależności parametru PK od funkcji przejścia logsig.

Średnia precyzja klasyfikacji została uzyskana na poziomie **95.08%**. Wartością maksymalną, uzyskaną w tym eksperymencie jest wartość **97.07%**. Uzyskana wartość minimalna została wyznaczona na poziomie **93.07%**.



Rys .16. Wykres zależności parametru PK od użytej funkcji przejścia tansig.

Średnia precyzja klasyfikacji została uzyskana na poziomie **93.36%**. Wartością maksymalną, uzyskaną w tym eksperymencie jest wartość **96.03%**. Uzyskana wartość minimalna została wyznaczona na poziomie **91.12%**.

W eksperymencie otrzymano większą średnią precyzję klasyfikacji dla funkcji przejścia logsig. Analizując dane wejściowe, możemy dojść do wniosku, iż są one w miarę liniowe, co może skutkować faktem większej efektywności użycia funkcji przejścia logsig.

4.6 Wyznaczenie optymalnych współczynników a, b oraz c, funkcji kwadratowych, opisujących liczbę neuronów w warstwie wejściowej oraz ukrytej

Założeniem eksperymentu szóstego, jest znalezienie jednej wartości współczynników funkcji kwadratowych oraz konkretną wartość z osi odciętych, które pozwolą na wyznaczenie takiej liczby neuronów, aby parametr PK, był jak największy. Ze względu na ograniczoną możliwość obliczeniową, funkcje zostały wyznaczone dla zakresu odciętych [1,29], dla zakresu współczynnika a: [1,19], b: [1,19] oraz c: [1,19]. W trakcie obliczeń, zostały pominięte wartości nienaturalne.

Wzór wyznaczający liczbę neuronów w warstwie wejściowej:

$$K1(x) = a * x^2 - b * x + c$$

Wzór wyznaczający liczbę neuronów w warstwie ukrytej:

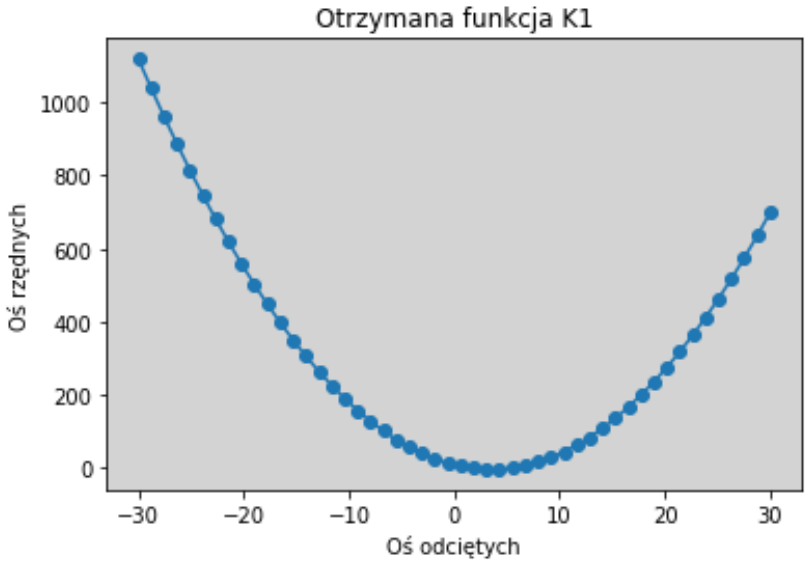
$$K2(x) = -a * x^2 + b * x + c$$

Po wykonaniu długotrwałych obliczeń otrzymano kilka przypadków wartości poszczególnych współczynników, dla których PK wyszło maksymalne, czyli równe **98.04%**. Są to wartości odpowiednio:
{a,b,c}:{1,4,7},{1,7,9},{1,13,18},{2,10,11},{3,6,10},{4,1,4},{4,7,18},{5,3,3},{9, 4, 19},{9,5,11}. W tym projekcie zostanie szczegółowo opisany jeden przypadek.

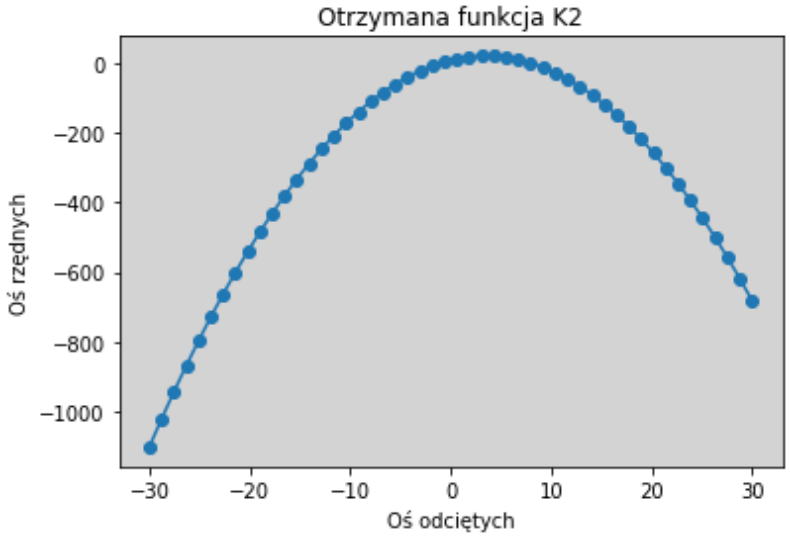
Przyjmując parametry a, b, c, jako {1,13,18}, otrzymujemy następujące wzory funkcji:

$$K1(x) = x^2 - 13 * x + 18$$

$$K2(x) = -x^2 + 13 * x + 18$$

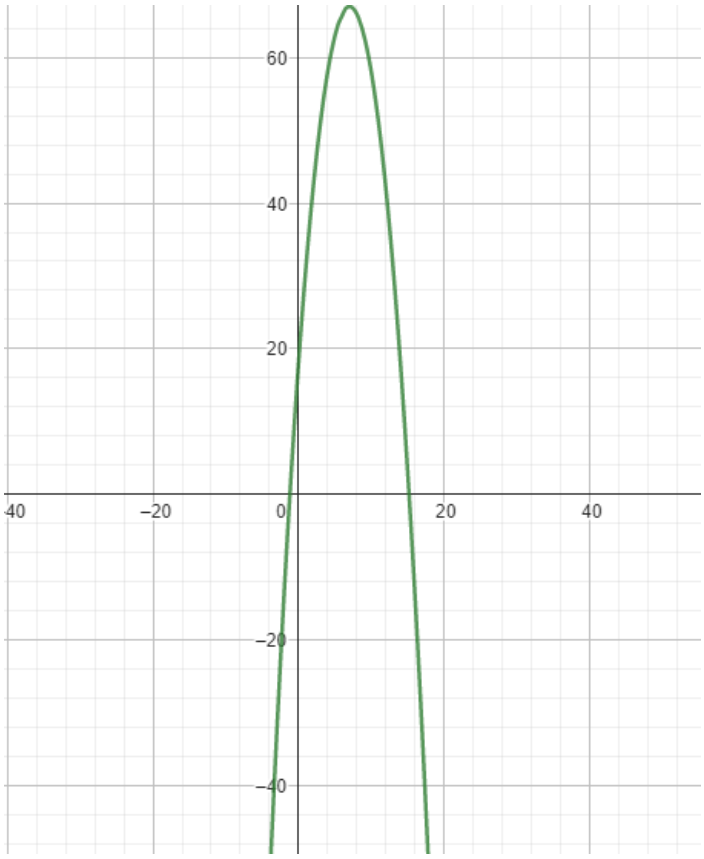


Rys .17. Wykres zależności liczby neuronów warstwy wejściowej od wartości odciętych z zakresu [-30,30].



Rys .18. Wykres zależności liczby neuronów warstwy ukrytej od wartości odciętych z zakresu [-30,30].

Ze względu na zbyt duży rozstrzał wartości, wykonano wykres funkcji K2(x) w innym programie w celu dokładniejszej obserwacji wartości dodatnich.



Rys .. Wykres zależności liczby neuronów warstwy ukrytej od wartości odciętych.

Dla wartości z osi odciętych: **x=1**, otrzymujemy najefektywniejszą parę neuronów równą odpowiednio:

$$K1(1) = 1 - 13 * 1 + 18 = 6$$

$$K2(1) = -1 + 13 * 1 + 18 = 30$$

Możemy wyrazić liczbę neuronów jednej warstwy przez liczbę neuronów drugiej warstwy:

$$K1(x) = x^2 - 13 * x + 18$$

$$x^2 = -K2(x) + 13 * x + 18$$

Podstawiając:

$$K1(x) = (-K2(x) + 13 * x + 18) - 13 * x + 18 =$$

$$K1(x) = -K2(x) + 36$$

5. Wnioski

Na podstawie przeprowadzonych eksperymentów, można stwierdzić, iż perceptron trójwarstwowy z przyśpieszeniem metodą momentum oraz adaptacyjnym współczynnikiem uczenia jest bardzo elastyczny oraz wydajny w przypadku klasyfikacji zwierząt do konkretnych klas. Jego elastyczność wynika z ilości możliwych zmiennych do manipulacji i przy określeniu pewnych parametrów sieć może działać z bardzo wysoką skutecznością. Jednakże, przy takiej ilości parametrów, perceptron wymaga rozważnego doboru poszczególnych współczynników, na przykładzie przytoczonego eksperymentu czwartego, gdzie parametr mc został ustalony na poziomie 0.99, co spowodowało spadek parametru PK do 1%. Z drugiej strony możemy przytoczyć przykładowy eksperyment szósty, w którym przy określonej liczbie neuronów, wartość parametru PK została obliczona na poziomie 98.04%, co jest bardzo wysoką dokładnością klasyfikacji.

Literatura

[1] Zajdel R., *Procedura przygotowania danych dla sieci neuronowych na potrzeby projektu z modułu sztuczna inteligencja.* - http://materialy.prz-rzeszow.pl [10.05.2023]

[2] Zajdel R., *Przyśpieszanie procesu uczenia* - http://materialy.prz-rzeszow.pl [11.05.2023]

[3] Zajdel R., *Model neuronu* - http://materialy.prz-rzeszow.pl [12.05.2023]

[4] Zajdel R., *Sieć jednokierunkowa wielowarstwowa* - http://materialy.prz-rzeszow.pl [13.05.2023]

[5] Zajdel R., *Sieć jednokierunkowa jednowarstwowa* - http://materialy.prz-rzeszow.pl

[13.05.2023]

[6] Cisowski P., *Poznawanie własności sieci neuronowych w środowisku MATLAB- https://dbc.wroc.pl/Content/19826/PDF/ciskowski_poznawanie.pdf* [16.05.2023]

[7] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning. MIT Press*. [16.05.2023]

[8] Źródło zbioru danych uczących: *http://archive.ics.uci.edu/ml/datasets/Zoo-*

[11.05.2023]

[9] Zajdel R., Skrypt programu używanego do przeprowadzenia eksperymentów –

http://materialy.prz-rzeszow.pl [12.05.2023]

Opracowanie tematu teoretycznego pt. „*Sieci jednowarstwowe jednokierunkowe*”, zostało w dużej mierze oparte na [5].