



**WYDZIAŁ
ELEKTROTECHNIKI
I INFORMATYKI**
POLITECHNIKI RZESZOWSKIEJ

Mateusz Fesz

Implementacja wielowarstwowej sieci neuronowej oraz algorytmu wstecznej propagacji błędów z adaptacyjnym współczynnikiem uczenia. Badanie wpływu metaparametrów sieci na efektywność uczenia

Praca projektowa z modułu „Sztuczna inteligencja”

Rzeszów, 2022

Spis treści

1. Opis projektu	6
1.1. Cel projektu	6
1.2. Opis i przygotowanie wykorzystanych danych	6
2. Zagadnienia teoretyczne związane z wykorzystaną siecią neuronową oraz algorytmem uczenia	8
2.1. Model neuronu	8
2.2. Sieć neuronowa jednowarstwowa	9
2.3. Sieć neuronowa wielowarstwowa	10
2.4. Proces uczenia sieci z wykorzystaniem metody gradientowej	11
2.4.1. Ogólny algorytm gradientowy	12
2.4.2. Zastosowanie algorytmu gradientowego w sieciach wielowarstwowych	14
2.4.3. Metoda stochastycznego spadku gradientu	14
2.4.4. Adaptacyjny współczynnik uczenia	15
3. Implementacja algorytmów w języku Rust	16
3.1. Implementacja struktury sieci	16
3.1.1. Podstawowa struktura reprezentująca sieć	16
3.1.2. Algorytm uczenia sieci	18
3.1.3. Skrypt ładujący dane	26
4. Eksperymenty	30
4.1. Seria 1 - Testy poprawności działania implementacji sieci	33
4.2. Seria 2 - Badanie wpływu metaparametrów sieci na przebieg oraz wynik procesu uczenia	34
4.2.1. Wpływ liczby neuronów na efektywność uczenia	36
4.2.2. Wpływ modyfikatorów współczynnika uczenia na poprawność klasyfikacji	37
5. Wstęp/wprowadzenie	40
6. Tekst zasadniczy – I	43
6.1. Formatowanie rozdziałów i podrozdziałów	43
7. Tekst zasadniczy – II	44

7.1. Formatowanie tekstu. Należy pamiętać, że na końcu tytułu rozdziału, podrozdziału i zakresu nie umieszcza się kropki	44
7.1.1. Marginesy i akapity	44
7.1.2. Zalecenia co do sposobu pisania jednostek i symboli wielkości fizycznych	45
7.1.3. Rysunki i tabele	47
7.1.4. Wzory matematyczne	48
7.1.5. Listingi programów	50
7.1.6. Numerowanie i punktowanie	50
7.2. Wykaz literatury	51
7.3. Wydruk pracy	51
8. Podsumowanie i wnioski końcowe	53
Załączniki	54
Literatura	55

1. Opis projektu

1.1. Cel projektu

Celem projektu jest realizacja uniwersalnej sieci neuronowej wielowarstwowej oraz zbadanie wpływu metaparametrów sieci na przebieg i efektywność procesu uczenia. Metaparametry poddane badaniom to:

- S1 - liczba neuronów w pierwszej warstwie sieci
- S2 - liczba neuronów w drugiej warstwie sieci
- lr - współczynnik uczenia sieci
- er - współczynnik maksymalnego dopuszczalnego przyrostu błędu, oznaczany również jako MAX_PERF_INC
- lr_{dec} - modyfikator współczynnika uczenia w przypadku przekroczeniu maksymalnego dopuszczalnego przyrostu błędu
- lr_{inc} - modyfikator współczynnika uczenia w przypadku spadku błędu

1.2. Opis i przygotowanie wykorzystanych danych

W celu przeprowadzenia badań, wykorzystany został zbiór danych dostępny pod adresem: <https://archive.ics.uci.edu/ml/datasets/zoo>. Składa się na niego 101 rekordów opisanych 18 parametrami. Opisują one następujące cechy zwierzęcia:

- 1) animal name - wartość tekstowa, stanowiąca nazwę zwierzęcia. Nieuwzględniana w procesie uczenia
- 2) hair - wartość logiczna, określająca występowanie owłosienia na ciele zwierzęcia
- 3) feathers, wartość logiczna, stwierdzająca występowanie piór na ciele zwierzęcia
- 4) eggs - wartość logiczna, niosąca informację o składaniu przez zwierzę jaj
- 5) milk - wartość logiczna, określająca zdolność zwierzęcia do wytwarzania mleka
- 6) airborne - wartość logiczna, stwierdzająca zdolność zwierzęcia do lotu

- 7) aquatic - wartość logiczna, niosąca informację o zdolności zwierzęcia do funkcjonowania w środowisku wodnym
- 8) predator - wartość logiczna, informująca czy zwierzę jest drapieżnikiem
- 9) toothed - wartość logiczna, określająca występowanie uzębienia u zwierzęcia
- 10) backbone - wartość logiczna, stwierdzająca występowanie kręgosłupa
- 11) breathes - wartość logiczna, niosąca informację o sposobie oddychania zwierzęcia
- 12) venomous - wartość logiczna, określająca czy zwierzę jest jadowite
- 13) fins - wartość logiczna, stwierdzająca występowanie płetw
- 14) legs - wartość liczbowa, informująca o liczbie nóg danego zwierzęcia
- 15) tail - wartość logiczna, niosąca informację o występowaniu ogona
- 16) domestic - wartość logiczna, określająca czy zwierzę jest uznawane za domowe
- 17) catsize - wartość logiczna, o niesprecyzowanej informacji
- 18) type - wartość liczbowa, określająca przynależność zwierzęcia do jednej z siedmiu klas

W procesie uczenia należy dokonać dopasowania zwierzęcia do odpowiedniej klasy. Jako parametry przyjmuje się kolumny 2-17. Kolumna 18 zawiera oczekiwaną wartość klasyfikacji. Kolumna pierwsza nie jest wykorzystywana w procesie uczenia, gdyż nie opisuje ona fizycznej cechy zwierzęcia, a jedynie jego nazwę.

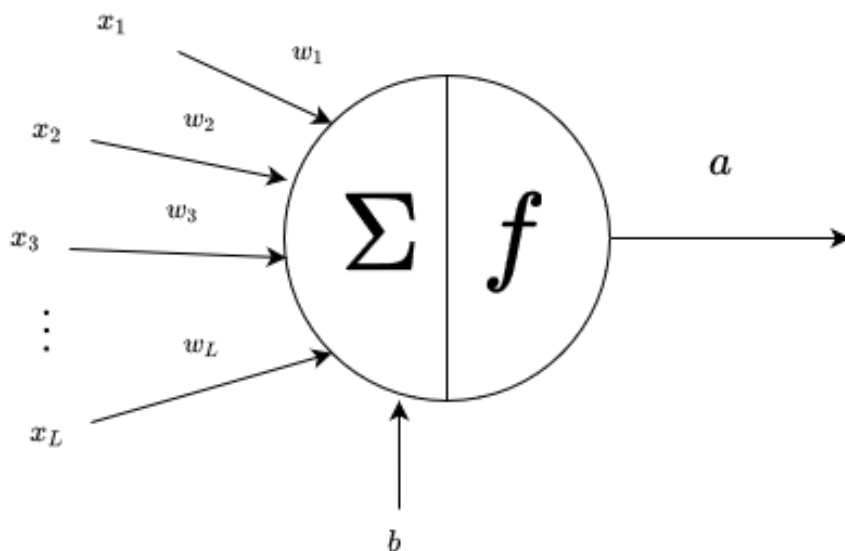
Przed przystąpieniem do procesu uczenia, przeprowadzono normalizację danych wejściowych. Wymagała jej jedynie kolumna „legs” przyjmująca wartości numeryczne z zakresu $[0; 8]$. Numer klasy stanowiący jedyną daną wyjściową został natomiast przekształcony do postaci siedmio-elementowego wektora, zawierającego wartość 1 na pozycji odpowiadającej danej klasie oraz wartości 0 na pozostałych pozycjach. W ostatnim kroku, dokonano podziału danych na zbiór uczący oraz zbiór walidacyjny. W tym celu przydzielono 25% rekordów należących do danej klasy do zbioru walidacyjnego, zaś pozostałą część do zbioru uczącego.

2. Zagadnienia teoretyczne związane z wykorzystaną siecią neuronową oraz algorytmem uczenia

2.1. Model neuronu

Neuron jest podstawowym elementem sieci neuronowej. Jego elementami są:

- Zbiór wejść x długości L
- Zbiór powiązanych z wejściami wag w długości L
- Wartość przesunięcia b
- Blok sumujący
- Funkcja aktywacji f
- Wyjście a



Rysunek 2.1: Graficzny model neuronu

Wartość sygnału wyjściowego neuronu możemy określić wzorem:

$$a = f \left(\sum_{j=1}^L w_j x_j + b \right) \quad (2.1)$$

W powyższym zapisie, x_j oraz w_j oznaczają odpowiednio kolejne wartości wejściowe i powiązane z nimi wagi (współczynniki wagowe). Zapis ten możemy jednakże uprościć, zakładając że $x = [x_1, x_2, \dots, x_L]^T$ będzie wektorem kolumnowym wejść, $w = [w_1, w_2, \dots, w_L]$ - macierzą wierszową powiązanych z wejściami wag, natomiast wartości a oraz b - skalarami. Wtedy równanie 2.1 przyjmie postać:

$$a = f(wx + b) \quad (2.2)$$

Ponadto istotnym w dalszych rozważaniach elementem modelu neuronu jest wyjście z bloku sumującego. Oznaczając je jako z otrzymamy:

$$z = \sum_{j=1}^L w_j x_j + b \quad (2.3)$$

Obliczona w ten sposób wartość, stanowi argument funkcji aktywacji neuronu. Funkcja ta przekształca wartość z na inną wartość uzależnioną od jej charakterystyki. Na potrzeby projektu, przyjęto że rolę tę będzie pełniła funkcja sigmoidalna (logsig). Jej wartość możemy obliczyć zgodnie ze wzorem:

$$f(z) = \frac{1}{1 + \exp(-z)} \quad (2.4)$$

W procesie uczenia istotna będzie także wartość pochodnej tej funkcji w danym punkcie. W przypadku funkcji sigmoidalnej można ją sprowadzić do następującej postaci:

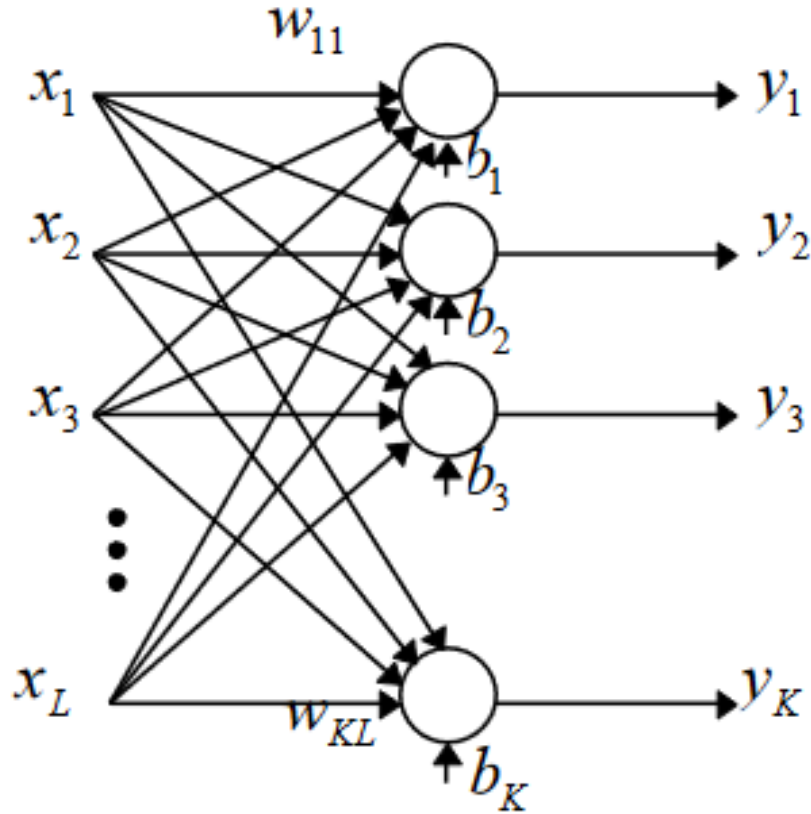
$$f'(z) = f(z) * 1 - f(z) \quad (2.5)$$

Co istotne, argumentami tej funkcji są stałe oraz wyliczona wcześniej pochodna w punkcie. Właściwość ta może być użyta do zmniejszenia liczby wykonywanych przez algorytm operacji.

2.2. Sieć neuronowa jednowarstwowa

Siecią neuronową jednowarstwową nazywamy taką sieć, w której neurony nie są połączone ze sobą bezpośrednio, a jedynie otrzymują dane na wejściu i podają wynik na wyjście.

Wejściem każdego neuronu jest wektor sygnałów wejściowych x , zaś wyjściem sieci wektor $y = [y_1, y_2, y_3, \dots, y_L]^T$. Ponadto możemy zdefiniować również wektor przesunięć $b = [b_1, b_2, \dots, b_K]$ Liczba wyjść z sieci jest tożsama z liczbą neuronów.



Rysunek 2.2: Model prostej sieci jednowarstwowej - Placeholder

Wagi przyporządkowane wejściom można wyrazić w postaci macierzy o rozmiarach $K \times L$ gdzie L oznacza liczbę wejść sieci, a K liczbę neuronów.

$$\mathbf{w} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1L} \\ w_{21} & w_{22} & \cdots & w_{2L} \\ \vdots & \vdots & & \vdots \\ w_{K1} & w_{K2} & \cdots & w_{KL} \end{bmatrix}$$

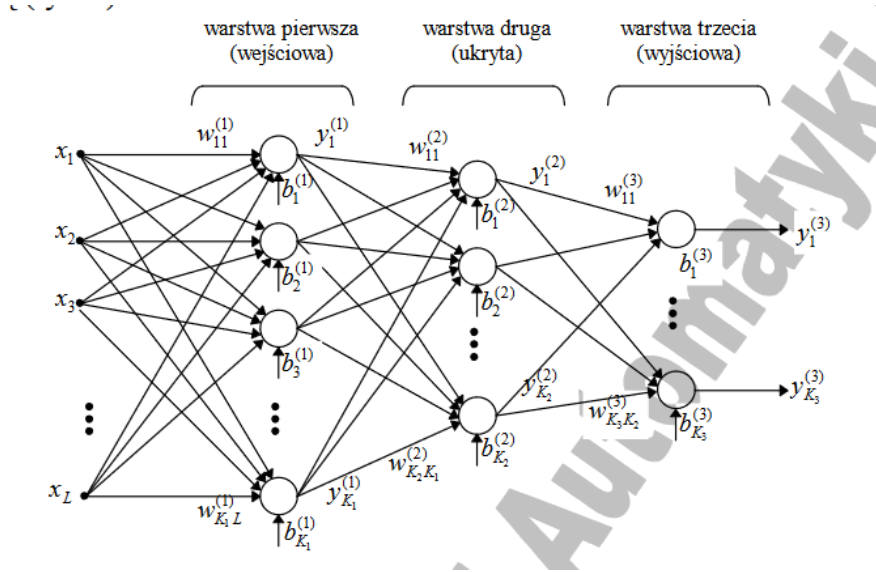
Przy założeniu że każdy neuron realizuje tę samą funkcję aktywacji, działanie sieci jednowarstwowej wielowarstwowej możemy wyrazić w postaci macierzowej:

$$y = f(wx + b) \quad (2.6)$$

2.3. Sieć neuronowa wielowarstwowa

Siecią neuronową wielowarstwową, nazywamy taką sieć w której neurony ułożone są w dwóch lub więcej połączonych ze sobą warstwach. Możemy zatem powiedzieć

że jest to pewna liczba połączonych kaskadowo sieci jednowarstwowych, a wyjście pojedynczej warstwy jest równocześnie wejściem warstwy następnej.



Rysunek 2.3: Model prostej sieci trójwarstwowej - Placeholder

W przypadku tego typu sieci ilość macierzy wag jest równa liczbie warstw neuronów. Oznaczając $y^{(l)}$ jako wyjście, $w^{(l)}$ jako macierz wag, $b^{(l)}$ jako macierz biasów, a $f^{(l)}$ jako funkcję aktywacji neuronów l -tej warstwy oraz wykorzystując równanie 2.6, w przypadku sieci trójwarstwowej otrzymamy równania:

$$\begin{aligned} y^{(1)} &= f^{(1)}(w^{(1)}x + b^{(1)}) \\ y^{(2)} &= f^{(2)}(w^{(2)}y^{(1)} + b^{(2)}) \\ y^{(3)} &= f^{(3)}(w^{(3)}y^{(2)} + b^{(3)}) \end{aligned} \quad (2.7)$$

Następnie, wykorzystując równania 2.7, możemy opisać działanie całej sieci trójwarstwowej równaniem:

$$y^{(3)} = f^{(3)}(w^{(3)}f^{(2)}(w^{(2)}f^{(1)}(w^{(1)}x + b^{(1)}) + b^{(2)}) + b^{(3)}) \quad (2.8)$$

2.4. Proces uczenia sieci z wykorzystaniem metody gradientowej

Celem określenia sposobu uczenia sieci, koniecznym jest uprzednie zdefiniowanie tego, co oznacza określenie sieci „nauczoną” bądź „nienauczoną”. W tym celu definiujemy tzw. funkcję kosztu, określającą poziom rozbieżności pomiędzy wartością otrzymaną na wyjściu sieci, a wartością oczekiwaną. Przykładem takiej funkcji, jest

funkcja błędu średniokwadratowego (MSE):

$$C(w, b, x, a) = \frac{1}{2n} \sum_x ||y(x) - a||^2 \quad (2.9)$$

W powyższym równaniu, w określa zbiór wszystkich wag wewnątrz sieci, b wszystkich jej biasów, x zbiór danych wejściowych, n ilość danych wejściowych, $y(x)$ zbiór oczekiwanych danych wyjściowych, natomiast a zbiór oczekiwanych wyjść z sieci. Zakładając, że w procesie uczenia wartości x oraz a pozostają stałe, możemy uprościć oznaczenie funkcji do postaci $C(w, b)$. Funkcja ta dąży do 0 gdy wartości otrzymywane na wyjściu sieci są bliskie wartościom oczekiwany, i rośnie wraz ze wzrostem różnicy pomiędzy nimi. W związku z powyższym, należy znaleźć taką metodę, która pozwoli na minimalizację wartości funkcji $C(w, b)$.

2.4.1. Ogólny algorytm gradientowy

Celem zobrazowania problemu rozważmy prostą funkcję $C(v_1, v_2)$ zobrazowaną na rysunku 2.4. Naszym celem jest znalezienie jej globalnego minimum. Teoretycznie możliwe jest jego wyznaczenie metodą analityczną, jednakże według literatury [3] jest to rozwiązanie mało wydajne, szczególnie gdy rozważymy funkcje więcej niż kilku zmiennych.

Obserwując rysunek 2.4 możemy jednak dojść do znacznie prostszej obliczeniowo metody. Przyjmując bliskie zeru wartości Δv_1 oraz Δv_2 możemy przyjąć zmianę wartości funkcji $C(v_1, v_2)$ na poziomie:

$$\Delta C \approx \frac{\delta C}{\delta v_1} \Delta v_1 + \frac{\delta C}{\delta v_2} \Delta v_2 \quad (2.10)$$

Na podstawie powyższego równania możemy zdefiniować wektor zmian:

$$\Delta v = (\Delta v_1, \Delta v_2)^T \quad (2.11)$$

oraz tzw. wektor gradientu:

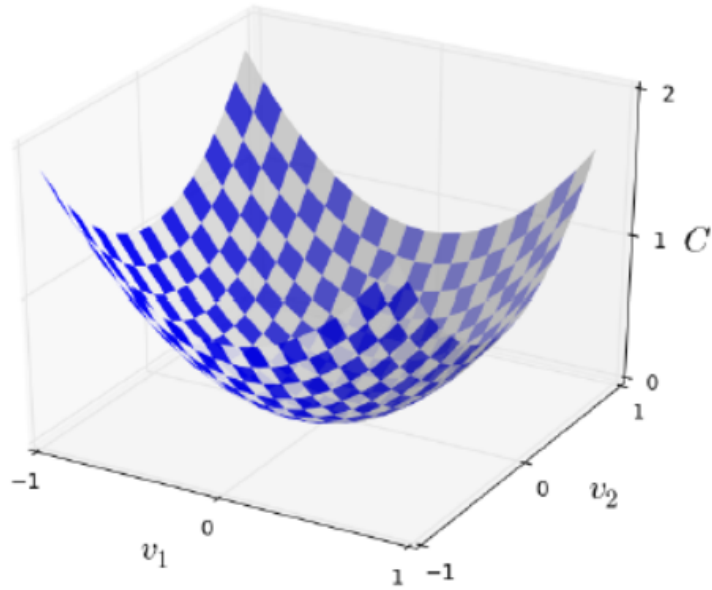
$$\nabla C = \left(\frac{\delta C}{\delta v_1}, \frac{\delta C}{\delta v_2} \right)^T \quad (2.12)$$

Wykorzystując powyższe definicje, równanie 2.12 możemy zapisać w postaci:

$$\Delta C \approx \nabla C \cdot \Delta v \quad (2.13)$$

Problemem przy równaniu 2.13 jest wyznaczenie optymalnego wektora Δv , tak aby zagwarantować ujemną wartość ΔC . W tym celu, możemy przyjąć wartości opisywane równaniem 2.11 jako równe:

$$\Delta v = -\eta \nabla C \quad (2.14)$$



Rysunek 2.4: Przykładowa funkcja 2 zmiennych

Wartość η nazywana jest „współczynnikiem uczenia” i przyjmuje bliskie zeru, dodatnie wartości. Wykorzystując powyższą definicję, możemy zapisać równanie 2.13 w postaci:

$$\Delta C \approx -\eta \nabla C \cdot \nabla C$$

$$\Delta C \approx -\eta \|\nabla C\|^2$$

W tej postaci równania widzimy, że dla odpowiednio niskiego współczynnika η zagwarantowany jest spadek wartości funkcji kosztu $C(v)$. Wykorzystując właściwość opisaną równaniem 2.14 możemy wyznaczyć nowe wartości zmiennych zawartych w wektorze v :

$$\begin{aligned} v \rightarrow v' &= v - \eta \nabla C \\ v' &= v + \Delta v \end{aligned} \tag{2.15}$$

Wielokrotnie aplikując powyższą regułę, jesteśmy w stanie podążać za gradientem aż do osiągnięcia minimum. Literatura [3] definiuje ją jako „algorytm spadku gradient”, ale jednocześnie wspomina że w niektórych sytuacjach, może nie być w pełni skuteczna.

2.4.2. Zastosowanie algorytmu gradientowego w sieciach wielowarstwowych

Algorytm gradientowy może zostać wykorzystany do minimalizacji funkcji przedstawionej równaniem 2.10, poprzez odnajdowanie wartości wag oraz biasów minimalizujących funkcję kosztu. Wykorzystując oznaczenia $w_{ij}^{(l)}$ jako j-tą wagę i-tego neuronu l-tej warstwy oraz $b_i^{(l)}$ jako bias i-tego neuronu w l-tej warstwie, możemy wyznaczyć ich wartości w kolejnych iteracjach (epokach) procesu uczenia:

$$w_{ij}^{(l)} \rightarrow w'_{ij}{}^{(l)} = w_{ij}^{(l)} - \eta \frac{\delta C}{\delta w_{ij}^{(l)}} \quad (2.16)$$

$$b_i^{(l)} \rightarrow b'_i{}^{(l)} = b_i^{(l)} - \eta \frac{\delta C}{\delta b_i^{(l)}} \quad (2.17)$$

Przyjmując jako funkcję kosztu błąd średniokwadratowy, przy założeniu wystąpienia wielu danych uczących, możemy ją zapisać w postaci:

$$C = \frac{1}{n} \sum_x C_x \quad (2.18)$$

czyli średniej wartości błędów $C_x = \frac{\|y(x)-a\|^2}{2}$ dla poszczególnych par uczących.

TODO: Rozpisać wzory na poszczególne pochodne - instrukcja [1]

2.4.3. Metoda stochastycznego spadku gradientu

Jednym z problemów algorytmu spadku gradientu, objawiającym się przy dużej liczbie rekordów uczących [3] jest długi czas potrzebny na obliczanie wartości pochodnych w równaniu 2.12. Celem zniwelowania tego problemu można wykorzystać metodę zwaną „stochastycznym spadkiem gradientu”. Polega ona na oszacowaniu rzeczywistego gradientu ∇C poprzez obliczenie ∇C_x dla losowo wybranej próbki danych uczących. Poprzez odpowiednie dobranie rozmiaru próbek, oraz ich odpowiednie uśrednienie, możemy uzyskać dobre przybliżenie rzeczywistego gradientu ∇C . [3] Elementy pojedynczej próbki możemy oznaczyć jako X_1, X_2, \dots, X_m . Zakładając odpowiednio duży rozmiar m możemy przyjąć za prawdziwe równanie:

$$\frac{\sum_{j=1}^m \nabla C_{X_j}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C \quad (2.19)$$

Przenosząc czynnik $\frac{1}{m}$ przed znak sumy, otrzymamy:

$$\nabla C \approx \frac{1}{m} \sum_{j=1}^m \nabla C_{X_j} \quad (2.20)$$

Wartym zauważenia jest też fakt występowania różnych konwencji uśredniania czy też skalowania gradientu stochastycznego. [3] Niektóre źródła mówią o braku konieczności ich stosowania gdyż wpływają na działanie sieci jedynie pośrednio, poprzez

modyfikację wartości współczynnika uczenia. Z tego też powodu, w dalszej implementacji przyjęto współczynnik równy stosunkowi rozmiaru próbki do rozmiaru całego zbioru uczącego.

2.4.4. Adaptacyjny współczynnik uczenia

W równaniu 2.14 wprowadzono pojęcie współczynnika uczenia. Standardowa metoda gradientowa zakłada że jest on stały przez cały okres uczenia sieci, a jego odpowiedni dobór jest jednym z kryteriów koniecznych do poprawnego przebiegu procesu uczenia. Nie jest to jednak rozwiązanie optymalne, gdyż zbyt duża jego wartość może znacząco utrudnić osiągnięcie minimum funkcji błędu a w skrajnych przypadkach spowodować rozbiegnięcie procesu uczenia. Zbyt mała jego wartość również jest niekorzystna gdyż znacznie wydłuża czas potrzebny na osiągnięcie celu uczenia - definiowanego jako minimalizacja funkcji wyrażonej równaniem 2.9.

Jedną z metod pozwalających zniwelować ten problem, jest metoda adaptacyjnej korekty współczynnika uczenia η [4]. Przyjmując wartość błędu w chwili czasu t :

$$MSE(t) = \frac{1}{2n} \sum_{i=1}^M ||y_i(t) - a(t)||^2 \quad (2.21)$$

Możemy zdefiniować sposób korekty współczynnika uczenia jako:

$$\eta(t+1) = \begin{cases} \eta(t) \cdot \xi_d & \text{gdy } MSE(t) > er \cdot MSE(t-1) \\ \eta(t) \cdot \xi_i & \text{gdy } MSE(t) < MSE(t-1) \\ \eta(t) & \text{gdy } MSE(t-1) \leq MSE(t) \leq er \cdot MSE(t-1) \end{cases} \quad (2.22)$$

gdzie ξ_d ξ_i są odpowiednio współczynnikami zmniejszania i zwiększania wartości współczynnika uczenia, zaś er określa dopuszczalną krotność przyrostu błędu. Podejście takie powoduje wprowadzenie konieczności ustalenia dodatkowych parametrów w procesie uczenia, lecz eliminuje problem doboru współczynnika uczenia. Jeśli będzie on zbyt duży, zostanie zredukowany do pożądanego poziomu ze względu na wzrastający poziom błędu. Jednocześnie zbyt niska wartość η zostanie zwiększona kiedy proces uczenia natrafi na bardziej optymalne warunki.

Literatura [5] podaje także bardziej skomplikowany wariant zastosowania zmiennego współczynnika uczenia. Polega on na dodatkowym zapisywaniu stanu wag oraz biasów przed ich zmianą wynikającą z algorytmu spadku gradientu. Wagi te mogą następnie zostać przywrócone jeśli uczenie przyniosło efekt odwrotny do zamierzonego

i doszło do wzrostu wartości błędu. W praktyce oznacza to przywrócenie zachowanych wag oraz biasów w momencie zmniejszenia η o współczynnik ξ_d .

3. Implementacja algorytmów w języku Rust

Przed wykonaniem badań operujących na przedstawionych algorytmach, konieczna była ich implementacja w postaci pozwalającej na proste ustalanie parametrów sieci, oraz jej ewentualne modyfikacje na potrzeby eksperymentów. W tym celu wykorzystany został język programowania ogólnego przeznaczenia - Rust, oraz następujące biblioteki:

- ndarray 0.15.4 - wykorzystywana do obliczeń związanych z algebrą liniową, przede wszystkim operacji na macierzach
- rand 0.8.5 - wykorzystywana do generowania liczb losowych

Pełny kod gotowy do kompilacji i uruchomienia znajduje się na repozytorium: https://github.com/MatiF100/AI_Project.

3.1. Implementacja struktury sieci

3.1.1. Podstawowa struktura reprezentująca sieć

Pierwszym krokiem koniecznym do realizacji projektu było wykonanie ogólnej struktury sieci, pozwalającej na ustalenie jej parametrów, oraz funkcji pozwalającej na jej inicjalizację.

```
1 #[derive(Debug, Clone)]
2 struct Network {
3     layers: Vec<usize>,
4     biases: Vec<Array2<f64>>,
5     weights: Vec<Array2<f64>>,
6     name: String,
7 }
```

Listing 1: Podstawowa struktura sieci neuronowej

```
1 fn new(layers: Vec<usize>) -> Self {
2     let mut rng = rand::thread_rng();
3     let layers = layers
4         .into_iter()
5         .filter(|x| *x > 0)
6         .collect::<Vec<usize>>();
7     Self {
8         name: String::from("Network 0"),
9         //Biases are initialized with random values
10        biases: layers
```



```

11     .iter()
12     .skip(1)
13     .map(|&s| {
14         (0..s)
15         .map(|_| rng.gen_range::<f64, std::ops::Range<f64
>>(-1.0..1.0))
16         .collect::<Vec<f64>>>()
17     })
18     .map(|v| Array2::from_shape_vec((v.len(), 1), v).unwrap())
19     .collect(),
20
21     //Weights are also initialized with random values
22     weights: layers
23         .windows(2)
24         .map(|x| {
25             (
26                 (x[0], x[1]),
27                 (0..x[0] * x[1])
28                 .map(|_| rng.gen_range::<f64, std::ops::Range<f64
>>(-1.0..1.0))
29                 .collect::<Vec<f64>>>(),
30             )
31         })
32         .map(|(x, v)| Array2::from_shape_vec((x.1, x.0), v).unwrap
33         ())
34         .collect::<Vec<_>>(),
35     //Layers are moved in from the argument
36     layers,
37 }

```

Listing 2: Konstruktor struktury sieci neuronowej

Sieć została zbudowana w sposób dający dużą swobodę w ustalaniu jej parametrów - zarówno liczba warstw jak i neuronów w poszczególnych warstwach może być łatwo modyfikowana przez podanie odpowiedniego wektora jako argumentu.

```

1 fn main(){
2     let (R, S1, S2, S3) = (16, 8, 6, 2);
3     let x = Network::new(vec![R, S1, S2, S3]);
4 }

```

Listing 3: Utworzenie przykładowej struktury sieci

Powyższy kod tworzy sieć neuronową o 16 wejściach, 8 neuronach w warstwie 1, 6 neuronach w warstwie drugiej oraz 2 neuronach w warstwie trzeciej. Dla uproszczenia kodu i przyspieszenia obliczeń, wejścia sieci są reprezentowane jako dodatkowa warstwa neuronów, których wyjście w trakcie działania sieci jest ustalane do wartości znajdujących się w wektorze wejściowym. Również dla uproszczenia kodu, przyjęto stałą funkcję aktywacji dla każdego z neuronów, opisaną równaniem 2.4.

```

1 //Sigmoidal function - basic activation function for neurons
2 fn sigmoid<D>(z: Array<f64, D>) -> Array<f64, D>
3 where
4     D: Dimension,
5 {
6     let mut z = z;
7     z.iter_mut().for_each(|f| *f = 1.0 / (1.0 + (-*f).exp()));
8     z
9 }
10
11 //Derivative of sigmoidal function
12 fn sigmoid_prime<D>(z: Array<f64, D>) -> Array<f64, D>
13 where
14     D: Dimension,
15 {
16     let val = sigmoid(z);
17     &val * (1.0 - &val)
18 }

```

Listing 4: Funkcja sigmoidalna oraz jej pochodna

Powyższa implementacja nie jest optymalna, gdyż zakłada obliczanie wartości funkcji aktywacji przy każdym wywołaniu funkcji zwracającej jej pochodną, podczas gdy zwykle wartość ta była już obliczona wcześniej. Możliwe jest zatem dalsza optymalizacja kodu, co jednakże nie jest konieczne dla przeprowadzenia większości eksperymentów.

```

1 fn feed_forward(&self, mut a: Array2<f64>) -> Array2<f64> {
2     for (b, w) in self.biases.iter().zip(self.weights.iter()) {
3         a = sigmoid(w.dot(&a) + b);
4     }
5     return a;
6 }

```

Listing 5: Realizacja funkcji feed-forward

Ostatnią funkcją konieczną do działania sieci jest funkcja feedforward, pobierająca dane z wejścia sieci, i zwracająca wynikowy wektor stopnia aktywacji neuronów wyjściowych.

3.1.2. Algorytm uczenia sieci

```

1 fn sgd(
2     &mut self,
3     training_data: &mut Vec<(Array2<f64>, Array2<f64>)>,
4     epochs: usize,
5     mini_batch_size: usize,
6     mut eta: f64,
7     test_data: Option<&Vec<(Array2<f64>, usize)>>,
8     eta_mod: Option<(f64, f64)>,
9     target_cost: f64,
10    report_interval: usize,

```

```

11 ) {
12   let mut rng = rand::thread_rng();
13
14   //Main loop performing learning step with each epoch
15   for j in 1..=epochs {
16     //Randomization of data for usage of mini-batch
17     training_data.shuffle(&mut rng);
18
19     //Generation of mini-batch vector. This is basically a
20     collection of smaller datasets
21     let mut mini_batches = training_data
22       .windows(mini_batch_size)
23       .step_by(mini_batch_size)
24       .map(|s| s.to_vec())
25       .collect::<Vec<Vec<_>>>();
26
27     let batch_count = mini_batches.len();
28     // Branching based on existance of adaptive learning rate
29     parameters
30     match eta_mod {
31       Some((dec, inc)) => {
32         // Saving state of network before readjustment of
33         weights and biases
34         let saved_weights = self.weights.clone();
35         let saved_biases = self.biases.clone();
36         let previous_error = self.mse(training_data);
37
38         //Sub-loop performing learning step for each of the mini
39         -batch
40         for mini_batch in &mut mini_batches {
41           //dbg!(&mini_batch);
42           self.update_mini_batch(mini_batch, eta, batch_count)
43         }
44
45         // Verification of newly achieved Mean Square Error
46         let new_error = self.mse(training_data);
47         if new_error < target_cost {
48           if let Some(data) = &test_data {
49             let output = self.evaluate(data);
50             println!("{}", self.name, output.1 as f64 /
51               output.0 as f64);
52           }
53           break;
54         }
55         if new_error > previous_error * MAX_PERF_INC {
56           // Restoring backup
57           self.weights = saved_weights;
58           self.biases = saved_biases;
59
60           // Adaptation - learning rate decreases
61           eta *= dec;
62         } else if new_error < previous_error {
63           // Adaptation - learning rate increases
64           eta *= inc;
65         }
66         // else statement does nothing - ommited

```

```

62     }
63     None => {
64         //Sub-loop performing learning step for each of the mini
        -batch
65         for mini_batch in &mut mini_batches {
66             //dbg!(&mini_batch);
67             self.update_mini_batch(mini_batch, eta, batch_count);
68             let new_error = self.mse(training_data);
69             if new_error < target_cost {
70                 if let Some(data) = &test_data {
71                     let output = self.evaluate(data);
72                     println!("{}", self.name, output.1 as f64 /
output.0 as f64);
73                 }
74                 break;
75             }
76         }
77     }
78 }
79
80 //Data verification. Can be ommited
81 if let Some(data) = &test_data {
82     if j + 1 % report_interval == 0 && report_interval != 0 {
83         let output = self.evaluate(data);
84         println!("{}", self.name, output.1 as f64 / output.0
as f64);
85     }
86 } else {
87     //println!("Epoch {} complete!", j);
88 }
89 }
90 }

```

Listing 6: Realizacja funkcji stochastycznego spadku gradientu

Powyższy kod realizuje uogólniony przypadek funkcji spadku gradientu. W jego ramach można wyróżnić kilka bloków funkcjonalnych. Pierwszym z nich jest definicja funkcji spadku gradientu oraz jej parametrów:

- mut self - mutowalna referencja do struktury Network, słowo kluczowe self oznacza że funkcja stanowi metodę struktury Network
- training_data - mutowalna referencja do wektora przechowującego pary uczące
- epochs - maksymalna liczba epok przez które należy wykonywać algorytm
- mini_batch_size - rozmiar próbki w metodzie stochastycznej (ustawiona na wartość równą długości wektora training_data będzie równoważne z brakiem podziału danych uczących)
- mut eta - mutowalna wartość zmiennoprzecinkowa, współczynnik uczenia

- test_data - opcjonalna referencja do wektora przechowującego dane wykorzystywane do walidacji krzyżowej
- eta_mod - opcjonalne parametry wykorzystywane w ramach adaptacyjnego współczynnika uczenia
- target_cost - docelowy błąd
- report_interval - liczba epok co którą sieć wykonuje walidację krzyżową i wyświetla raport

Następnym etapem, działającym już wewnątrz pętli jest wykonanie podziału zbioru uczącego na mniejsze próbki (o ile następuje taka konieczność). Dalsza część funkcji opiera się na opcjonalnym parametrze eta_mod. Jeśli jest on obecny, wykonywany jest algorytm powiązany z adaptacyjnym współczynnikiem uczenia. W każdej iteracji pętli, parametry sieci oraz poprzedni błąd są zapisywane przed wykonaniem operacji uczenia, a następnie w zależności od uzyskanego wyniku, wykonywana jest jedna z operacji opisana w równaniu 2.22. Jeżeli natomiast nie podano parametrów modyfikacji współczynnika uczenia, wykonywany jest klasyczny algorytm bez przyspieszeń. W trakcie uczenia, wykorzystywana jest funkcja update_mini_batch(...), którą przedstawiono na listingu 8.

Ostatnim elementem pętli jest natomiast walidacja krzyżowa. Zachodzi jeśli podano zostały podane dane weryfikacyjne, a numer obecnej epoki jest wielokrotnością zadanego interwału raportu. Działanie algorytmu walidacji krzyżowej prezentuje listing 7

```

1 fn evaluate(&self, test_data: &Vec<(Array2<f64>, usize)>) -> (
2     usize, usize) {
3     let mut local_data = test_data.clone();
4     let x = local_data
5         .iter_mut()
6         .map(|(x, y)| {
7             (
8                 self.feed_forward(x.clone())
9                 .iter()
10                .enumerate()
11                .max_by(|(_, a), (_, b)| {
12                    a.partial_cmp(b).unwrap_or(std::cmp::Ordering::Equal
13                ))
14            ),
15            y,
16        })
17     }

```

```

17     .filter(|(a, b)| a.unwrap_or(0) == **b)
18     .count();
19     (test_data.len(), x)
20 }

```

Listing 7: Realizacja funkcji walidacji krzyżowej

Ponieważ w przyjętej implementacji sieci neuronowej, każdy z neuronów posiada sigmoidalną funkcję aktywacji, konieczne jest zdefiniowanie jednoznacznego sposobu określenia, do jakiej klasy sieć przypisuje obiekt o zadanych parametrach. Powyższa implementacja zakłada w tym celu wykorzystanie n neuronów wyjściowych, gdzie n oznacza liczbę istniejących klas. Określenie wyniku klasyfikacji odbywa się poprzez funkcję maksimum, zwracającą indeks neuronu wyjściowego o najwyższym stopniu aktywacji. Warto zaznaczyć że w powyższej implementacji nie jest istotna sama wartość aktywacji neuronu, a jedynie jej stosunek względem pozostałych neuronów warstwy wyjściowej.

```

1 fn update_mini_batch(
2     &mut self,
3     mini_batch: &Vec<(Array2<f64>, Array2<f64>)>,
4     eta: f64,
5     batches_count: usize,
6 ) {
7     // Allocation of gradient vectors
8     let mut nabla_b = self
9         .biases
10        .iter()
11        .map(|b| Array::zeros(b.raw_dim()))
12        .collect::<Vec<Array2<f64>>>();
13     let mut nabla_w = self
14         .weights
15         .iter()
16         .map(|w| Array::zeros(w.raw_dim()))
17         .collect::<Vec<Array2<f64>>>();
18
19     // Loop performing learning iteration over all mini_batches
20     for (x, y) in mini_batch {
21         // Getting updated gradients from backpropagation algorithm
22         let (delta_nabla_b, delta_nabla_w) = self.backprop(x, y);
23
24         // Calculating new gradients with respect to ones created in
25         // first steps and also newly calculated ones
26         nabla_b = nabla_b
27             .iter()
28             .zip(delta_nabla_b.iter())
29             .map(|(nb, dnb)| nb + dnb)
30             .collect();
31         nabla_w = nabla_w
32             .iter()
33             .zip(delta_nabla_w.iter())
34             .map(|(nw, dnw)| nw + dnw)
35             .collect();
36     }
37 }

```

```

36
37 // Calculating new values for weights and biases based on
    recieved gradients with respect to batch size and learning
    rate
38 self.weights = self
39     .weights
40     .iter()
41     .zip(nabla_w.iter())
42     .map(|(w, nw)| w - nw * (eta / batches_count as f64) as f64)
43     .collect();
44 self.biases = self
45     .biases
46     .iter()
47     .zip(nabla_b.iter())
48     .map(|(b, nb)| b - nb * (eta / batches_count as f64) as f64)
49     .collect();
50 }

```

Listing 8: Realizacja funkcji `update_mini_batch`

Powyższa funkcja oblicza wartości gradientów dla wyodrębnionych w poprzednim etapie algorytmu próbek. Parametry przyjmowane przez funkcję to:

- `mut self` - mutowalna referencja do struktury Newtork
- `mini_batch` - referencja do próbki danych uczących
- `eta` - współczynnik uczenia
- `batches_count` - liczba wszystkich próbek wyodrębnionych z danych uczących

Funkcja ta alokuje bufor o stałym rozmiarze, a następnie używa go do sumowania gradientów obliczanych dla poszczególnych par uczących w funkcji `backprop(...)` opisanej listingiem 9. Następnie na podstawie obliczonego gradientu wykonywana jest aktualizacja wag. Procedurę realizowaną przez tą funkcję opisują równania 2.16 oraz 2.17.

```

1 fn backprop(&self, x: &Array2<f64>, y: &Array2<f64>) -> (Vec<
    Array2<f64>>, Vec<Array2<f64>>) {
2 //Initialization of gradient vectors.
3 let mut nabla_b = self
4     .biases
5     .iter()
6     .map(|b| Array::zeros(b.raw_dim()))
7     .collect::<Vec<Array2<f64>>>();
8 let mut nabla_w = self
9     .weights
10    .iter()
11    .map(|w| Array::zeros(w.raw_dim()))
12    .collect::<Vec<Array2<f64>>>();

```

```

13
14 // Preparing initial information for forward network pass
15 // Because of lifetimes and loop scope further in the function
16 // , it is best to make copies of input matrix here
17 let mut activation = x.clone();
18 let mut activations = vec![activation];
19
20 // zs is a Vector of neurons non linear blocks inputs - these
21 // will be calculated in the following loop
22 let mut zs: Vec<Array2<f64>> = Vec::new();
23
24 // Performing feedforward operation
25 for (b, w) in self.biases.iter().zip(self.weights.iter()) {
26     // z is the input of non-linear block, necessary in the
27     // following gradient calculation
28     let z = w.dot(activations.iter().last().unwrap()) + b;
29     zs.push(z.clone());
30
31     // As the current matrix of non-linear block inputs is
32     // calculated, it is passed as argument to the activation
33     // function
34     // TODO: in this place other activation functions can be
35     // implemented
36     activation = sigmoid(z);
37
38     // Saving the outputs of neuron layer, for later use
39     activations.push(activation);
40 }
41
42 // Calculating per_class or per_output cost of the result at
43 // this point, it is also worth noting that the "delta" is only
44 // partially calculated
45 // With used notation, the delta itself does not include the
46 // eta, or learning rate
47 // Cost derivative function only calculates error, or
48 // difference between achieved and expected output
49 // TODO: possibly unnecessary function call
50 let mut delta = Self::cost_derivative(activations.last().
51     unwrap().clone(), y.clone())
52     * sigmoid_prime(zs.last().unwrap().clone());
53
54 // Setting up known values in gradient vectors
55 // Last layer is easiest to calculate, as it does not require
56 // any data not available at the moment
57 // We they will be used as we perform the backward pass,
58 // calculating bias and weight gradients for every layer
59 *nabla_b.last_mut().unwrap() = delta.clone();
60 *nabla_w.last_mut().unwrap() = delta.dot(&activations[
61     activations.len() - 2].t());
62
63 // Performing backward network pass
64 // Side note: if the book gives example of any identifier as "
65 // l" "I" or "L" one should never follow the book and come up
66 // with anything that differs from l
67 for idx in 2..self.layers.len() {
68     // Getting the input of non-linear block for the idx-th

```



```

53     layer counting from the end
54     let z = &zs[zs.len() - idx];
55
56     // Calculating the derivative of activation function for
57     // given input
58     let derivative = sigmoid_prime(z.clone());
59
60     // Calculating delta - gradient for given layer
61     // TODO: Include the generic formula into readme
62     delta = self.weights[self.weights.len() - idx + 1].t().dot(&
63     delta) * derivative;
64
65     // Boilerplate forced by borrow-checker. Since .len() uses
66     // immutable reference, it would block the assignment operation
67     // if used inline
68     // Works fine this way though, since usize implements "Copy"
69     let b_len = nabla_b.len();
70     let w_len = nabla_w.len();
71
72     // Actual gradient for biases and weights is pretty similar
73     // The difference is that weight gradient is additionally
74     // multiplied by the activation state of given layer
75     nabla_b[b_len - idx] = delta.clone();
76     nabla_w[w_len - idx] = delta.dot(&activations[activations.
77     len() - idx - 1].t());
78 }
79
80 // Returning calculated gradient vectors
81 (nabla_b, nabla_w)
82 }
83
84 fn cost_derivative(output_activations: Array2<f64>, y: Array2<
85 f64>) -> Array2<f64> {
86     output_activations - y
87 }

```

Listing 9: Realizacja funkcji wstecznej propagacji błędów

Argumentami powyższej funkcji `backprop(...)` są kolejno:

- `self` - referencja do struktury `Network`
- `x` - wektor parametrów pary uczącej
- `y` - oczekiwane wyjście dla zadanych parametrów

Dodatkowo zdefiniowana została funkcja pomocnicza `cost_derivative(...)` zwracająca wektor pochodnych funkcji błędów dla każdego z wyjść, względem wyjść neuronów warstwy wyjściowej. Dla przyjętej funkcji błędów przyjmuje ona postać różnicy pomiędzy wyjściem sieci, a wyjściem oczekiwanym.

W pierwszej kolejności, podobnie jak w funkcji 8 alokowana jest pamięć potrzebna na przechowanie obliczonych gradientów. Następnie zostają przygotowane

zmienne przechowujące wartości wyjścia (aktywacji) neuronów oraz wejścia funkcji aktywacji. Kolejnym krokiem jest obliczenie oraz zachowanie wartości wejść oraz wyjść funkcji aktywacji neuronów w poszczególnych warstwach. Po wykonaniu pętli, obliczana jest wartość delta, a na jej podstawie wartości gradientów dla ostatniej warstwy wag oraz biasów. W ostatnim kroku, ponownie wykonywana jest pętla, która powtarza poprzedni krok dla pozostałych warstw, obliczając wartości gradientów dla warstw od przedostatniej do pierwszej (wejściowej). Obliczone w ten sposób wektory gradientów są następnie zwracane z funkcji.

3.1.3. Skrypt ładujący dane

Elementem niezbędnym do wykonania badań było dostosowanie zadanego zbioru danych ZOO do wymagań sieci. W tym celu wykonano dodatkowy moduł programu, w którym zdefiniowane zostały odpowiednie funkcje. Zawartość modułu przedstawia listing 10.

```
1 use ndarray::{arr2, Array2};
2
3 #[derive(Debug, Default, Clone)]
4 pub struct Animal {
5     name: String,
6     hair: f64,
7     feathers: f64,
8     eggs: f64,
9     milk: f64,
10    airborne: f64,
11    aqatic: f64,
12    predator: f64,
13    toothed: f64,
14    backbone: f64,
15    breathes: f64,
16    venomous: f64,
17    fins: f64,
18    legs: f64,
19    tail: f64,
20    domestic: f64,
21    catsize: f64,
22    ani_type: u8,
23 }
24
25 impl Animal {
26     #[allow(dead_code)]
27     pub fn even_list(mut vector: Vec<Self>) -> Vec<Self> {
28         let mut classes = [0; 7];
29         let limit = 9;
30
31         vector = vector
32             .into_iter()
33             .filter(|a| {
34                 classes[a.ani_type as usize] += 1;
```

```

35         classes[a.ani_type as usize] <= limit
36     })
37     .collect::

```

```

(max - min) as f64);
85
86     values
87 }
88 fn from_str(data: Vec<&str>) -> Self {
89     let mut an: Self = Default::default();
90     let mut it = data.iter();
91     an.name = it.next().unwrap().to_string();
92     an.hair = match *it.next().unwrap() {
93         "0" => 0.0,
94         _ => 1.0,
95     };
96     an.feathers = match *it.next().unwrap() {
97         "0" => 0.0,
98         _ => 1.0,
99     };
100    an.eggs = match *it.next().unwrap() {
101        "0" => 0.0,
102        _ => 1.0,
103    };
104    an.milk = match *it.next().unwrap() {
105        "0" => 0.0,
106        _ => 1.0,
107    };
108    an.airborne = match *it.next().unwrap() {
109        "0" => 0.0,
110        _ => 1.0,
111    };
112    an.aquatic = match *it.next().unwrap() {
113        "0" => 0.0,
114        _ => 1.0,
115    };
116    an.predator = match *it.next().unwrap() {
117        "0" => 0.0,
118        _ => 1.0,
119    };
120    an.toothed = match *it.next().unwrap() {
121        "0" => 0.0,
122        _ => 1.0,
123    };
124    an.backbone = match *it.next().unwrap() {
125        "0" => 0.0,
126        _ => 1.0,
127    };
128    an.breathes = match *it.next().unwrap() {
129        "0" => 0.0,
130        _ => 1.0,
131    };
132    an.venomous = match *it.next().unwrap() {
133        "0" => 0.0,
134        _ => 1.0,
135    };
136    an.fins = match *it.next().unwrap() {
137        "0" => 0.0,
138        _ => 1.0,
139    };

```

```

140     an.legs = it.next().unwrap().parse::<f64>().unwrap();
141     an.tail = match *it.next().unwrap() {
142         "0" => 0.0,
143         _ => 1.0,
144     };
145     an.domestic = match *it.next().unwrap() {
146         "0" => 0.0,
147         _ => 1.0,
148     };
149     an.catsize = match *it.next().unwrap() {
150         "0" => 0.0,
151         _ => 1.0,
152     };
153     an.ani_type = it.next().unwrap().parse::<u8>().unwrap()
- 1;
154
155     an
156 }
157 pub fn into_training_arr2(&self) -> (Array2<f64>, Array2<f64
>) {
158     (
159         arr2(&[[
160             self.hair,
161             self.feathers,
162             self.eggs,
163             self.milk,
164             self.airborne,
165             self.aqatic,
166             self.predator,
167             self.toothed,
168             self.backbone,
169             self.breathes,
170             self.venomous,
171             self.fins,
172             self.legs as f64,
173             self.tail,
174             self.domestic,
175             self.catsize,
176         ]])
177         .t()
178         .to_owned(),
179         Array2::from_shape_vec(
180             (7, 1),
181             (0..7)
182                 .map(|x| if x == self.ani_type { 1.0 } else
{ 0.0 })
183                 .collect::<Vec<f64>>()),
184         )
185         .unwrap(),
186     )
187 }
188 }

```

Listing 10: Moduł odpowiedzialny za załadowanie oraz przygotowanie danych

Struktura Animal przedstawiona na listingu 10 pozwala na przechowanie rekor-

dów uczących w postaci obiektów, co ułatwia dalszą manipulację nimi. Powiązane z nią funkcje pozwalają na utworzenie nowej listy zwierząt na bazie danych odczytanych z pliku oraz jej konwersję do formatu wymaganego przez sieć neuronową zaprezentowaną na listingu 1.

4. Eksperymenty

Eksperymenty którym poddawane były sieci, miały na celu zbadanie poprawności implementacji algorytmów, a także późniejsze badania ich wydajności oraz wpływu szeregu zmiennych na proces uczenia. Celem przygotowania struktury sieci do eksperymentów, zmodyfikowane zostały jej elementy mogące wpływać niekorzystnie na powtarzalność wyników. Przy uczeniu sieci o stałej liczbie neuronów wykonywana była głęboka kopia całej struktury, co pozwoliło na zapewnienie jednolitych warunków startowych. Ponadto pseudolosowy generator wag, został zastąpiony generatorem pseudolosowym o ustalonym ziarnie. Pozwoliło to na zapewnienie zbliżonych warunków startowych również dla sieci o różnej liczbie neuronów.

Ze względu na dużą liczbę planowanych eksperymentów, przystosowano kod testujący do równoczesnego działania na wielu rdzeniach procesora. W tym celu wykorzystano mechanizm wątków oraz system komunikacji kolejkowej „Multi-producer, single-consumer” co prezentuje listing 11.

```
1  fn main() {
2
3      //Vector for learning data
4      let t_data: Vec<(Array2<f64>, Array2<f64>)>;
5      //Vector for validation data
6      let v_data: Vec<(Array2<f64>, Array2<f64>)>;
7
8      let data = std::fs::read_to_string("zoo.data");
9
10     let animal_list = data::Animal::new_list(&data.unwrap());
11     let animal_list = data::Animal::partitioned_list(animal_list
12 , 0.75);
13     t_data = animal_list
14         .0
15         .iter()
16         .map(|a| a.into_training_arr2())
17         .collect::<Vec<_>>();
18
19     v_data = animal_list
20         .1
21         .iter()
22         .map(|a| a.into_training_arr2())
23         .collect::<Vec<_>>();
```

```

24     let data_len = t_data.len();
25     let test_data = v_data
26         .iter()
27         .map(|(input, output)| {
28             (
29                 input,
30                 output
31                     .iter()
32                     .enumerate()
33                     .max_by(|(a, _), (b, _)| a.partial_cmp(b).
unwrap_or(std::cmp::Ordering::Equal))
34                     .unwrap(),
35             )
36         })
37         .map(|(a, s)| (a.clone(), s.0))
38         .collect::<Vec<_>>()
39         .to_owned();
40
41     println!("Learning record count: {}", t_data.len());
42     let lr_step = vec![
43         0.0001, 0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7,
44         0.8, 0.9, 0.92, 0.94, 0.95, 0.96,
45         0.97, 0.98, 0.99, 10.0,
46     ];
47     let lr_inc_step = vec![
48         1.01, 1.03, 1.04, 1.05, 1.06, 1.07, 1.08, 1.1, 1.15,
49         1.2, 1.25, 1.3, 1.4,
50     ];
51     let lr_dec_step = vec![
52         0.99, 0.98, 0.97, 0.95, 0.93, 0.92, 0.9, 0.85, 0.8,
53         0.75, 0.7, 0.65, 0.6,
54     ];
55     let er_step = vec![
56         1.0, 1.001, 1.01, 1.02, 1.03, 1.04, 1.05, 1.06, 1.07,
57         1.08, 1.09, 1.1, 1.15, 1.2, 1.25,
58         1.3, 1.4, 1.5,
59     ];
60
61     let threads = std::sync::Arc::new(std::sync::Mutex::new(0));
62     let (sync_tx, sync_rx) = mpsc::sync_channel(16);
63     let (tx, rx) = mpsc::channel();
64
65     let tmp_threads = threads.clone();
66     thread::spawn(move || {
67         for s1 in 0..t_data.len() / 3 - 7 {
68             for s2 in 0..(t_data.len() / 3 - 7 - s1) {
69                 if s2 == 0 && s1 != 0 {
70                     continue;
71                 }
72                 let x = Network::new(vec![16, s1, s2, 7]);
73                 //for lr in &lr_step {
74                 for er in &er_step {
75                     let lr = 0.3;
76                     for lr_dec in &lr_dec_step {
77                         for lr_inc in &lr_inc_step {
78                             let mut net = x.clone();

```

```

75         net.name = format!("{}", s1, s2, lr, lr_dec, lr_inc, er);
76         let mut t_data = t_data.clone();
77         let test_data = test_data.clone();
78
79         let local_lrs = (lr, *lr_dec, *
lr_inc);
80         let local_sync_tx = sync_tx.clone();
81         let local_tx = tx.clone();
82         let local_er = *er;
83
84         local_sync_tx.send(()).unwrap();
85         let inner_threads = std::sync::Arc::
clone(&tmp_threads);
86
87         *inner_threads.lock().unwrap() += 1;
88         thread::spawn(move || {
89             net.sgd(
90                 &mut t_data,
91                 1000,
92                 data_len,
93                 local_lrs.0,
94                 Some(&test_data),
95                 Some((local_lrs.1, local_lrs
.2)),
96                 0.25 / t_data.len(),
97                 1000,
98                 Some(local_er),
99             );
100             local_tx.send(()).unwrap();
101         });
102     }
103 }
104 }
105 }
106 }
107 });
108
109 loop {
110     let threads = std::sync::Arc::clone(&threads);
111     //println!("{}", threads.lock().unwrap());
112     rx.recv().unwrap();
113     //println!("{}", threads.lock().unwrap());
114     sync_rx.recv().unwrap();
115     *threads.lock().unwrap() -= 1;
116     if *threads.lock().unwrap() <= 0 {
117         break;
118     }
119 }
120
121 }

```

Listing 11: Przykładowy skrypt eksperymentalny

4.1. Seria 1 - Testy poprawności działania implementacji sieci

Celem pierwszej serii eksperymentów była weryfikacja poprawności implementacji struktury sieci oraz algorytmów jej uczenia. W tym celu zastosowano proces uczenia dla prostych przypadków 2-wejściowych bramek logicznych AND, OR oraz XOR. W jej trakcie odnotowano niespodziewaną, istotną dla dalszych testów cechę implementacji. Prawdopodobnie z powodu przyjęcia funkcji sigmoidalnej dla neuronów warstwy wyjściowej oraz sposobu interpretacji zwracanej przez sieć wyników, ustalenie granicy błędu docelowego na równą

$$\text{err_goal} = 0.25/n$$

Przy n oznaczającym długość zbioru uczącego, powodowało występowanie dużej liczby błędów podczas walidacji działania sieci, nawet na zbiorze uczącym. Dalsze testy wykazały zwiększenie efektywności wskazań sieci przy kilkukrotnym zmniejszeniu wartości błędu docelowego. Wynik 100% poprawnej klasyfikacji był osiągany regularnie, dla różnych warunków startowych, przy wartości $\text{err_goal} = \frac{0.25}{2n}$. Z tego powodu, w części kolejnych eksperymentów również przyjmowano niższe wartości błędu docelowego.

```
1  fn main() {
2      let mut x = Network::new(vec![2, 4, 2]);
3      let xor = vec![
4          (
5              Array2::from_shape_vec((2, 1), vec![0.0, 0.0]).
6              unwrap(),
7              Array2::from_shape_vec((2, 1), vec![0.0, 1.0]).
8              unwrap(),
9              (
10                 Array2::from_shape_vec((2, 1), vec![0.0, 1.0]).
11                 unwrap(),
12                 Array2::from_shape_vec((2, 1), vec![1.0, 0.0]).
13                 unwrap(),
14                 (
15                     Array2::from_shape_vec((2, 1), vec![1.0, 0.0]).
16                     unwrap(),
17                     Array2::from_shape_vec((2, 1), vec![1.0, 0.0]).
18                     unwrap(),
19                     (
20                         Array2::from_shape_vec((2, 1), vec![1.0, 1.0]).
21                         unwrap(),
22                         Array2::from_shape_vec((2, 1), vec![0.0, 1.0]).
23                         unwrap(),
24                     ),
25                 ),
26             ),
27          ];
28      let test_data = xor
```

```

22     .iter()
23     .map(|(input, output)| {
24         (
25             input,
26             output
27                 .iter()
28                 .enumerate()
29                 .max_by(|(_, a), (_, b)| a.partial_cmp(b).
unwrap_or(std::cmp::Ordering::Equal))
30                 .unwrap(),
31         )
32     })
33     .map(|(a, s)| (a.clone(), s.0))
34     .collect::

```

Listing 12: Skrypt uczący dla bramki logicznej XOR

Powyższy eksperyment wykazał również konieczność zastosowania więcej niż 1 warstwy neuronów dla problemów nieseparowalnych liniowo. Zastosowanie jedynie 1 warstwy sprawdziło się zarówno w przypadku bramki AND jak i OR, lecz w przypadku XOR sieć utrzymywała poziom 50% poprawności klasyfikacji.

4.2. Seria 2 - Badanie wpływu metaparametrów sieci na przebieg oraz wynik procesu uczenia

W tej serii eksperymentów przetestowano 520 520 różnych konfiguracji sieci neuronowych. Przyjęto też arbitralne ziarno generatora liczb losowych, dla którego sieć wykazywała nienajlepsze efekty uczenia. Takie podejście pozwoliło na lepszą obserwację wpływu parametrów na przebieg uczenia, niż w przypadku zestawu parametrów sieci gwarantujących pełną poprawność w większości sytuacji. Modyfikację przykładowego kodu wykorzystaną do ich przeprowadzenia prezentuje listing 13.

```

1  thread::spawn(move || {
2      for s1 in 0..t_data.len() / 3 - 7 {
3          for s2 in 0..(t_data.len() / 3 - 7 - s1) {

```

```

4         if s2 == 0 && s1 != 0 {
5             continue;
6         }
7         let x = Network::new(vec![16, s1, s2, 7]);
8         for lr in &lr_step {
9             let lr = 0.3;
10            for lr_dec in &lr_dec_step {
11                for lr_inc in &lr_inc_step {
12                    let mut net = x.clone();
13                    net.name = format!("{}", s1, s2, lr, lr_dec, lr_inc, er);
14                    let mut t_data = t_data.clone();
15                    let test_data = test_data.clone();
16
17                    let local_lrs = (lr, *lr_dec, *
18                        lr_inc);
19
20                    let local_sync_tx = sync_tx.clone();
21                    let local_tx = tx.clone();
22                    let local_er = *er;
23
24                    local_sync_tx.send(()).unwrap();
25                    let inner_threads = std::sync::Arc::
26                        clone(&tmp_threads);
27
28                    *inner_threads.lock().unwrap() += 1;
29                    thread::spawn(move || {
30                        net.sgd(
31                            &mut t_data,
32                            1000,
33                            data_len,
34                            local_lrs.0,
35                            Some(&test_data),
36                            Some((local_lrs.1, local_lrs
37                                .2)),
38                            0.25 / t_data.len(),
39                            1000,
40                            None
41                        );
42                        local_tx.send(()).unwrap();
43                    });
44                }
45            }
46        }
47    }
48}

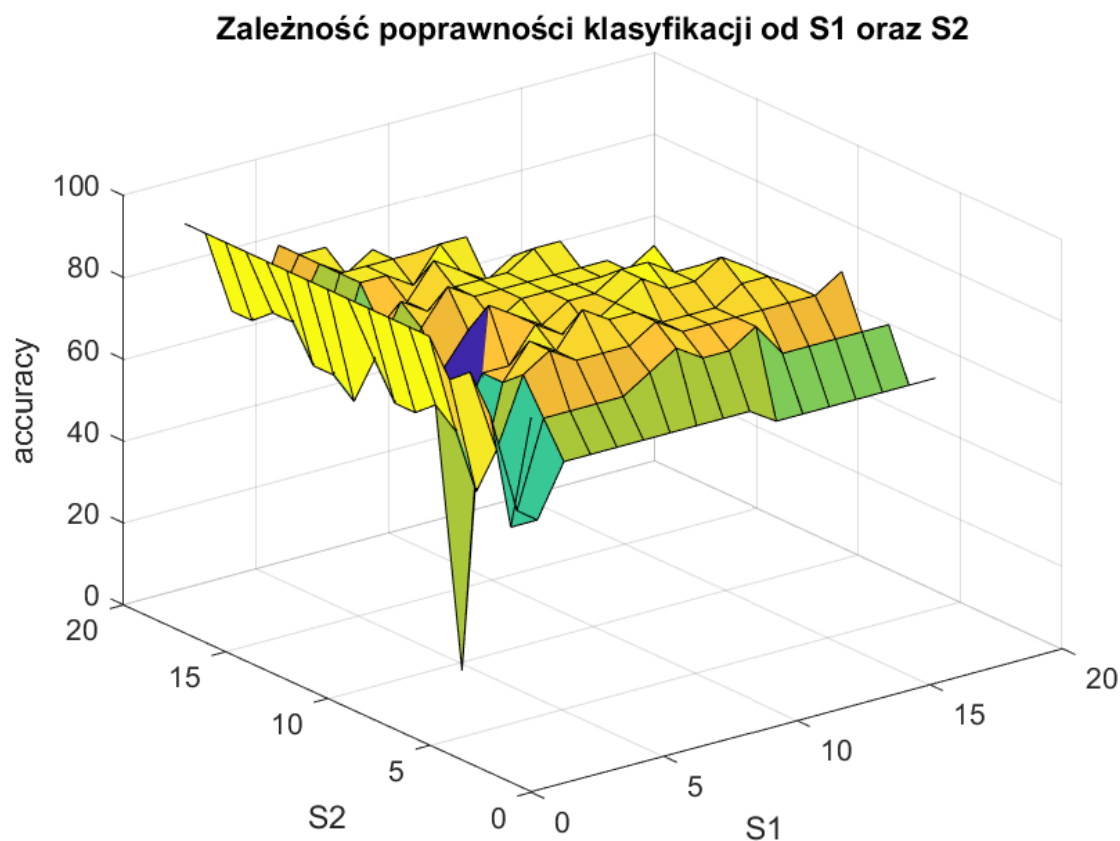
```

Listing 13: Modyfikacja kodu tworzącego instancje sieci na potrzeby eksperymentu

Ponieważ w trakcie tego eksperymentu nie sprawdzano parametru MAX_PERF_INC celem uniknięcia dalszego zwiększania liczby sieci do przetestowania, jego wartość przyjęto jako stałą równą 1.05

4.2.1. Wpływ liczby neuronów na efektywność uczenia

Dane pozyskane w wyniku eksperymentu pozwoliły na wygenerowanie 2889 wykresów zależności poprawności klasyfikacji w zależności od S1 oraz S2, przy zadanych parametrach współczynnika uczenia. Analizując je widzimy dużą skuteczność procesu uczenia dla sieci 2-warstwowych.

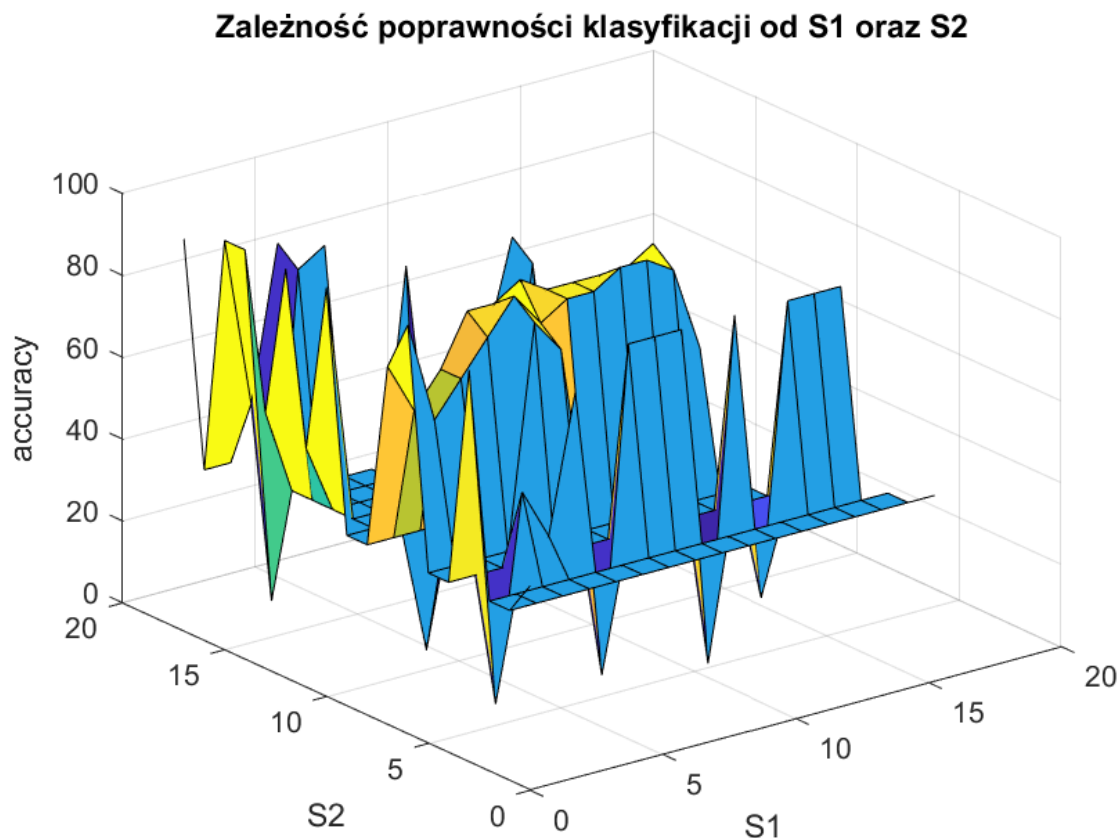


Rysunek 4.5: Wykres zależności poprawności klasyfikacji od S1 oraz S2 przy $lr = 0.2$, $lr_inc = 1.05$, $lr_dec = 0.9$

W przypadku sieci 2-warstwowych, pełną poprawność klasyfikacji zazwyczaj obserwujemy przy mniej niż 1000 epokach, co nie ma miejsca w przypadku sieci 3-warstwowej. Sieć 3-warstwowa osiągała 100% poprawność klasyfikacji jedynie w pojedynczych przypadkach. Może to sugerować że następuje przewymiarowanie sieci i traci ona zdolność uogólniania. Alternatywnie, możliwym jest także konieczność dłuższego uczenia takiej sieci. Wizualizację wyników testu widzimy na rysunku ???. Podobne wyniki uzyskano dla znacznej większości spośród wszystkich przetestowanych sieci.

Interesujący efekt został natomiast uzyskany dla niskiego wyjściowego współ-

czynnika uczenia. Charakterystyczny dla sieci z niskim współczynnikiem uczenia jest wykres 4.6.



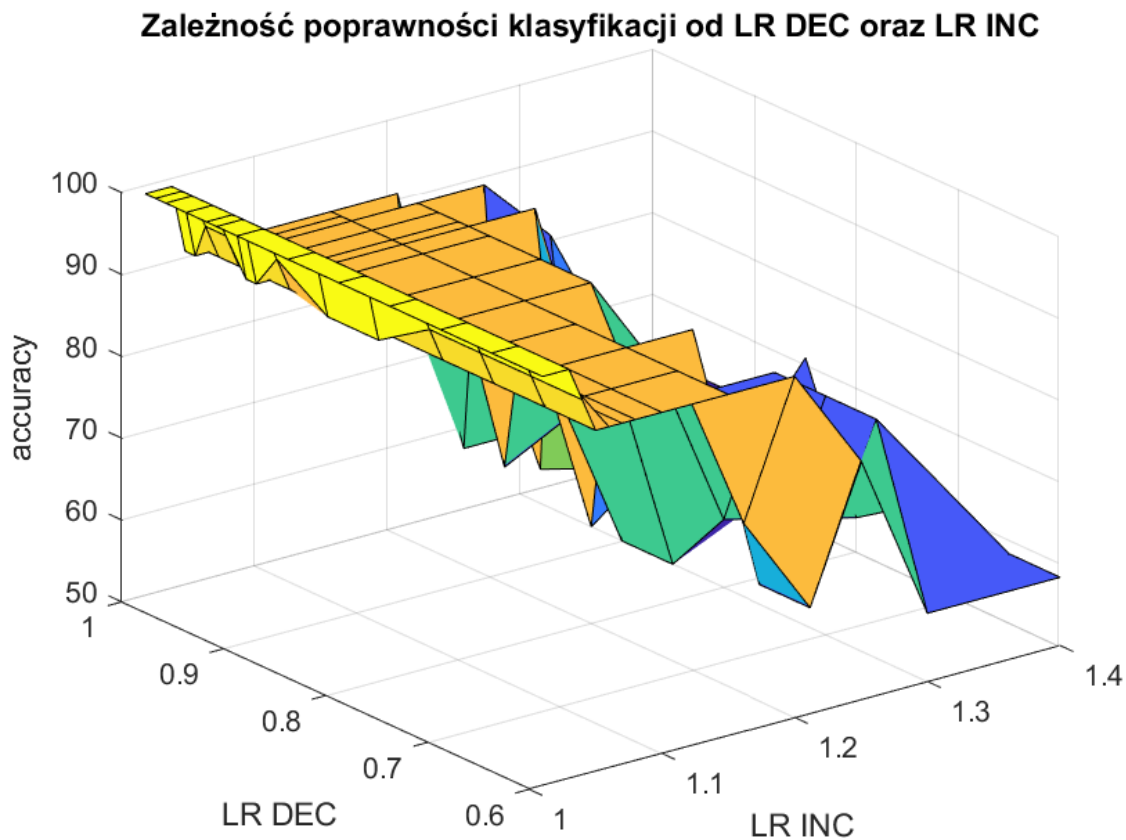
Rysunek 4.6: Wykres zależności poprawności klasyfikacji od S1 oraz S2 przy $lr = 0.0001$, $lr_inc = 1.25$, $lr_dec = 0.93$

W jego przypadku widzimy silną punktowość sieci. Poprawność klasyfikacji przy niektórych ilościach neuronów w poszczególnych warstwach jest podobna jak w eksperymentach z wyższym współczynnikiem uczenia, zaś w innych spada poniżej 40%. Prawdopodobną przyczyną jest duża podatność takiej sieci na parametry początkowe, zwłaszcza dobór wag. Mimo inicjalizacji generatora pseudolosowego stałym ziarnem, stan wag może być mniej lub bardziej odległy od optymalnego w zależności od liczby neuronów w poszczególnych warstwach.

4.2.2. Wpływ modyfikatorów współczynnika uczenia na poprawność klasyfikacji

Poprzez modyfikatory współczynnika uczenia rozumie się parametry lr_inc oraz lr_dec , wykorzystywane przy adaptacyjnej aktualizacji współczynnika uczenia. Po-

wstałe na ich podstawie wykresy pokazują ponadto poziom efektywności sieci o zadanej liczbie neuronów. Rysunek 4.7 prezentuje rozkład charakterystyczny dla sieci jednowarstwowych

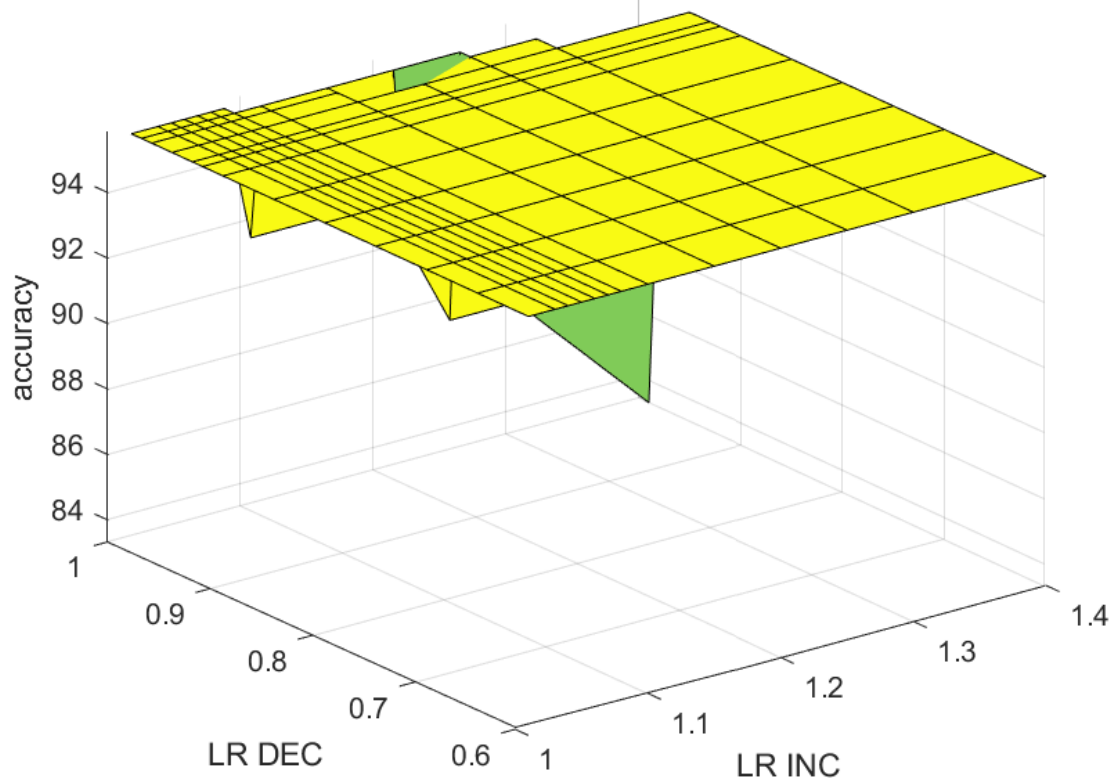


Rysunek 4.7: Wykres zależności poprawności klasyfikacji od modyfikatorów lr, przy $lr = 0.7$ dla sieci jednowarstwowej

Widzimy że najlepsza poprawność klasyfikacji wystąpiła dla małych wartości współczynnika lr_inc . Współczynnik lr_dec w tym przypadku nie miał zauważalnego wpływu. Podobnie w przypadku większości sieci dwuwarstwowych, efekt modyfikatorów był mało zauważalny, co przedstawia rysunek 4.8. W tym przypadku jednakże, powodem jest ogólnie duża skuteczność sieci jednowarstwowych. Większość z nich z łatwością osiąga pełną poprawność klasyfikacji, występują jedynie pojedyncze odchyły bez widocznej zależności.

Interesujące zjawisko możemy jednak zaobserwować przy liczbie neuronów większej od 13, co zaprezentowano na rysunku 4.9. W takiej sytuacji, następuje gwałtowny spadek poprawności klasyfikacji przez sieć dwuwarstwową. Prawdopodobną przyczyną

Zależność poprawności klasyfikacji od LR DEC oraz LR INC



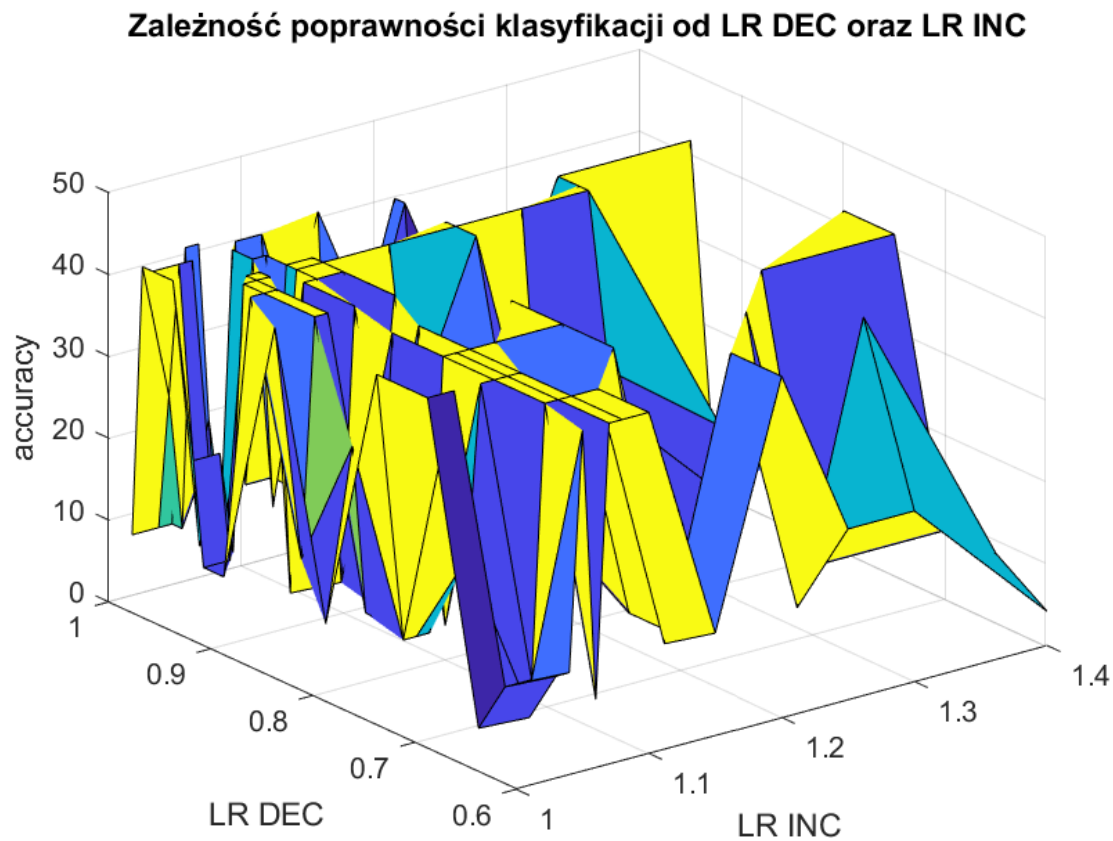
Rysunek 4.8: Wykres zależności poprawności klasyfikacji od modyfikatorów lr, przy $lr = 0.7$ dla sieci dwuwarstwowej przy $S1 = 3$

jest przewymiarowanie sieci i jej zbyt duża specjalizacja w klasyfikacji. W efekcie sieć traci zdolność uogólniającą i nie reaguje poprawnie na dane których nie wykorzystywano podczas uczenia.

Powyższy problem występuje jednakże

Również interesujące zjawisko prezentuje rysunek 4.10. Widzimy na nim całkowicie stałą poprawność klasyfikacji niezależnie od pozostałych parametrów. Co więcej, taki wykres występuje wielokrotnie i wyłącznie przy wartościach początkowych współczynnika uczenia mniejszych niż 0,001. Sugeruje to że wartość taka jest już zbyt mała, i pomimo zastosowania metody adaptacyjnej sieć nie jest w stanie pokonać minimum lokalnego funkcji błędu, bądź też zadana liczba epok jest zbyt mała aby mogło dojść do efektywnego nauczania sieci.

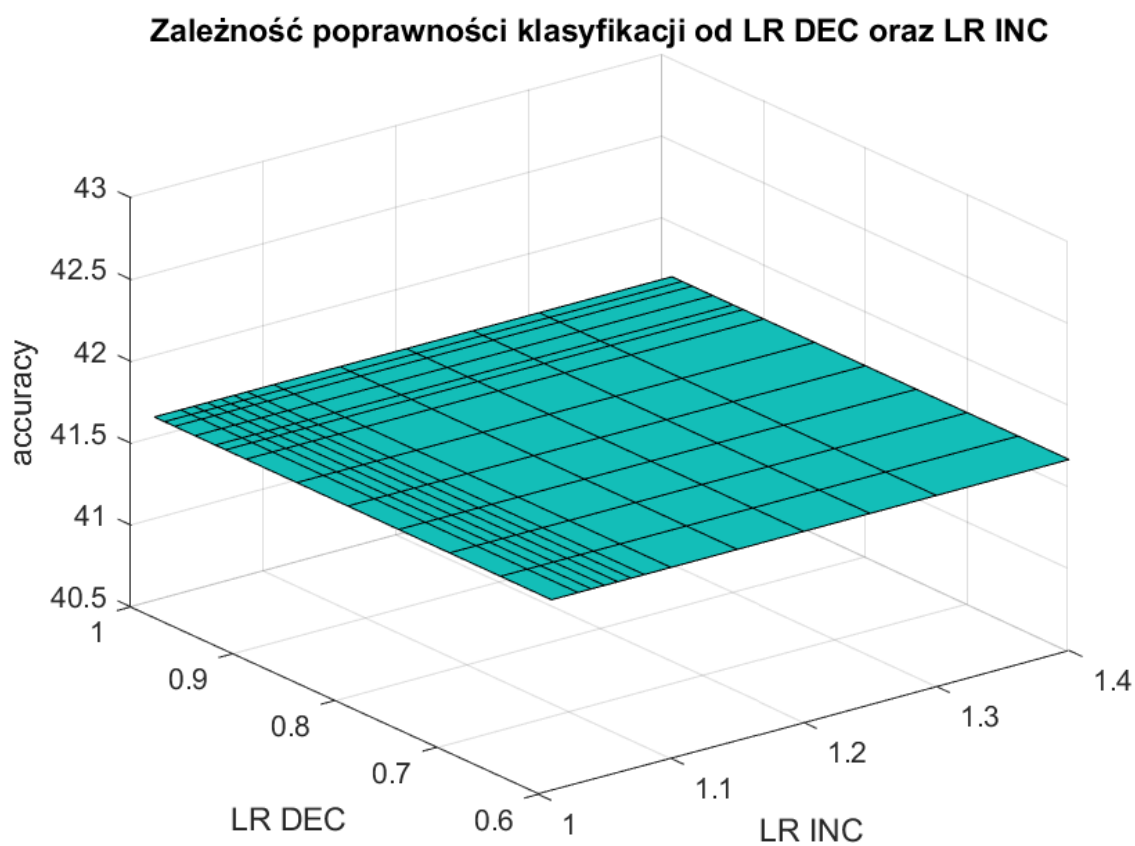
Skuteczność adaptacyjnego współczynnika uczenia objawia się najbardziej w sieciach 3-warstwowych. Przykładowy wynik uczenia dla takiej sieci obrazuje wykres



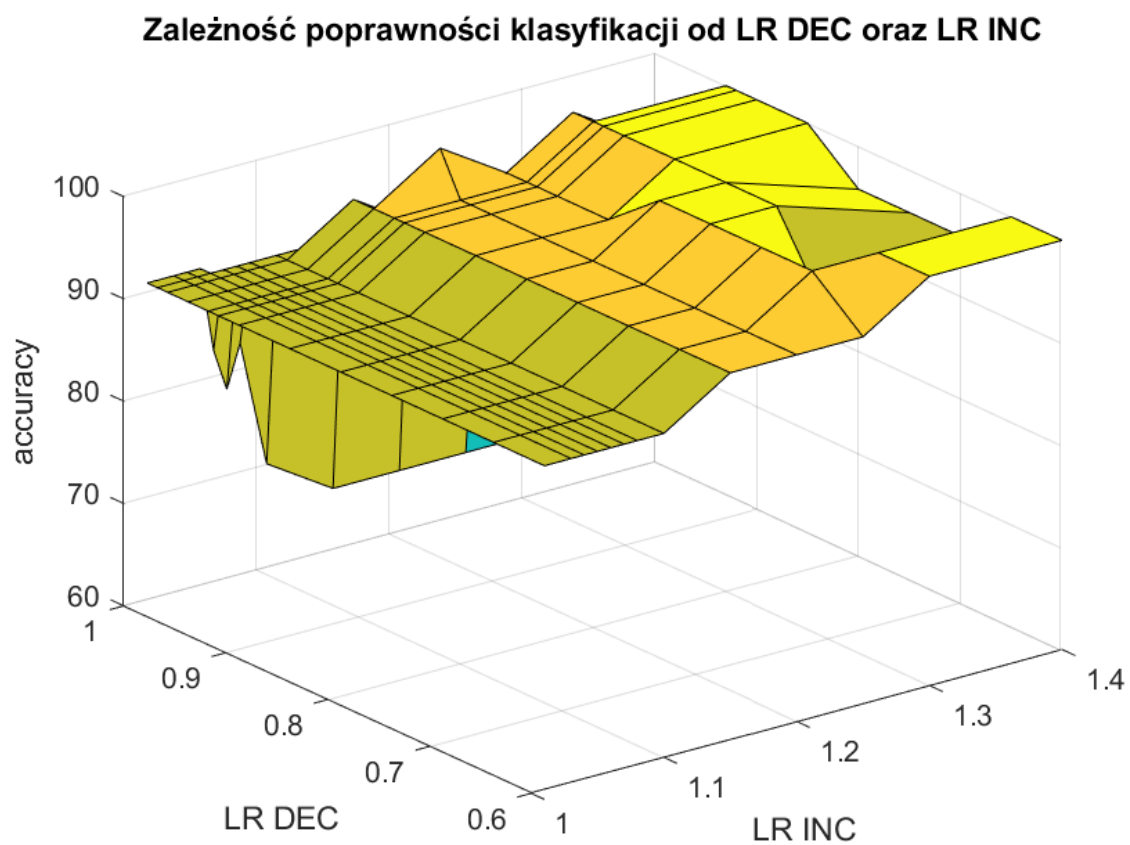
Rysunek 4.9: Wykres zależności poprawności klasyfikacji od modyfikatorów lr, przy $lr = 0.96$ dla sieci dwuwarstwowej przy $S1 = 14$

4.11. W tym przypadku wpływ wartości parametrów adaptacyjnych jest dużo bardziej widoczny niż w poprzednich przykładach.

Widzimy że dla osiągnięcia najwyższej skuteczności, istotnym jest ustalenie odpowiedniej wartości obu parametrów. Jeśli będziemy modyfikować tylko jeden z nich, a drugi pozostawimy bliski jedności, widzimy brak wymiernych zysków, a nawet pogorszenie dokładności klasyfikacji. Ponadto widzimy że niewielkie wartości współczynników nie wpływały w stopniu znacznym na ostateczny wynik testu.



Rysunek 4.10: Płaski wykres występujący przy niskim początkowym współczynniku uczenia



Rysunek 4.11: Zależność poprawności klasyfikacji od parametrów adaptacyjnych przy $lr = 0.2$, $S1 = 3$ oraz $S2 = 10$

5. Wstęp/wprowadzenie

1 ÷ 5 stron charakterystyka problematyki w świetle aktualnego stanu wiedzy i techniki, ze wskazaniem na zagadnienia istotne z punktu widzenia realizowanej pracy. Na trzeciej stronie można zamieścić podziękowania dla osób, które przyczyniły się do powstania pracy dyplomowej. Na kolejnej stronie nieparzystej rozpoczyna się spis treści. Po spisie treści zalecane jest umieszczenie wykazu użytych symboli, oznaczeń i akronimów. Od tego miejsca rozpoczyna się numeracja rozdziałów. Na następnej stronie umieszcza się wprowadzenie do pracy (scharakteryzowanie problematyki pracy, uzasadnienie wyboru tematyki) oraz przedstawia: cel i/lub tezę pracy, zakres pracy, przyjęte założenia itp. Ostatni akapit wstępu musi zawierać zwięzłe sformułowanie celu i zakresu pracy.

Uwaga:

Jeżeli decydujesz się wykorzystywać \LaTeX 'a, ignoruj ogólny dokument dotyczący formatowania pracy dyplomowej na WEiI - jest przeznaczony dla użytkowników innych edytorów tekstu. Korzystaj z załączonego arkusza stylu, stosuj formatowanie znaczeniowe (nie wymuszaj formatowania), a wynikowa praca będzie zgodna z wymaganiami. Zachęcamy do używania \LaTeX 'a, czas poświęcony na jego przyswojenie, zwróci się z nawiązką nawet w trakcie tworzenia pracy dyplomowej.

Niniejszy tekst, wykorzystujący styl `weiszablon.sty` zawiera informacje o formatowaniu, wielkości czcionek, wyrównania... , ale **uwaga**, sama treść nie jest istotna (np. opis wielkości czcionek), formatowanie wykona się automatycznie, tu te zapisy są tylko po to, aby dostarczyć dokument zawierający jak najwięcej przykładów użycia \LaTeX 'a.

6. Tekst zasadniczy – I

Do 20% objętości pracy. W zależności od charakteru pracy ten rozdział powinien zawierać:

- a) opis tematyki zagadnienia – aktualny stan zagadnienia,
- b) metody i rozwiązania,
- c) dyskusja i krytyczna ocena stanu aktualnego,
- d) podsumowanie stanu wiedzy, techniki literaturowe itp.

6.1. Formatowanie rozdziałów i podrozdziałów

Rozdziały zaczynają się u góry nowej strony (parzystej lub nieparzystej). Podrozdziały i zakresy mogą zaczynać się w dowolnym miejscu strony. Przy końcu pracy zamieszcza się podsumowanie i wnioski. Ostatni akapit podsumowania musi zawierać wyszczególnienie własnej pracy Autora i zaczynać się od sformułowania: „Autor za własny wkład pracy uważa:”. W tym miejscu kończy się numeracja rozdziałów.

Ewentualne listingi programów, instrukcje obsługi stanowisk lub inne tego rodzaju materiały zaleca się zamieścić w formie dodatków. Kolejno zamieszcza się: wykaz literatury, spis rysunków/tabel oraz streszczenie (zgodne ze „Wzorem streszczenia”). Wykaz literatury rozpoczyna od strony nieparzystej.

Opisując własne dokonania, stosuje się formę bezosobową w czasie przeszłym np. celem pracy było zaprojektowanie..., zakres pracy obejmował wyznaczenie..., w ramach pracy wykonano model... itp.

7. Tekst zasadniczy – II

Ponad 50% objętości pracy – część autorska:

- a) założenia – dane,
- b) opis zastosowanej metody rozwiązania lub analizy,
- c) opis proponowanego rozwiązania, wyniki analizy teoretycznej, obliczenia, projekt konstrukcyjny, procesowy, technologiczny,
- d) wyniki badań analitycznych, symulacyjnych lub eksperymentalnych itp.

Przy stosowaniu podziału na rozdziały i podrozdziały zaleca się unikać podziału więcej niż trzystopniowego. Podział tekstu, szczególnie na rozdziały główne, wynikać powinien z zakresu i charakterystyki realizowanej pracy.

7.1. Formatowanie tekstu. Należy pamiętać, że na końcu tytułu rozdziału, podrozdziału i zakresu nie umieszcza się kropki

7.1.1. Marginesy i akapity

Marginesy deklaruje się jako „lustrzane” i ustawia na 2 cm, na oprawę 1,5 cm. Nagłówek i stopka 1,25 cm. Tekst podstawowy akapitu: czcionka szeryfowa, styl Times (Times New Roman, Liberation Serif itp.), rozmiar 12 punktów, interlinia 1,5 wiersza. Akapit wyjustowany, wcięcie pierwszego wiersza 1,25 cm.

Na końcu każdego akapitu, którego tekst zaczerpnięto z literatury, musi znajdować się odnośnik do właściwej pozycji w wykazie literatury. W pracy nie stosuje się odnośników w formie przypisów. Liczby w nawiasie kwadratowym oznaczają kolejny numer pozycji w wykazie, np. [1] lub [1, 4, 7] lub [1, 6-8] itp.

Cytaty (dosłowne przytoczenie obcego tekstu w pracy) pisze się czcionką pochylą (kursywą) i ujmuje w cudzysłów. Przykład: „*Współpracując z jednostkami gospodarczymi działającymi w kraju, kształci wysokokwalifikowaną kadrę inżynierów*”.

Fragmenty kodów programów pisze się czcionką o stałej szerokości, styl Courier (Courier New, Liberation Mono itp.) o rozmiarze 10 punktów.

7.1.2. Zalecenia co do sposobu pisania jednostek i symboli wielkości fizycznych

Poniższy podrozdział opracowano na podstawie [?]. W trakcie pisania pracy należy zwracać uwagę na sposób oznaczania jednostek i symboli wielkości fizycznych. Przy zapisywaniu jednostek i symboli wielkości fizycznych można wyróżnić zapis w postaci kursywy (pismo pochyle) oraz antykwy (pismo proste).

1) Kursywę należy stosować w następujących przypadkach:

- symboli wielkości fizycznych niezależnie od tego czy jest to litera alfabetu greckiego (np. przenikalność magnetyczna μ) czy też łacińskiego (np. rezystancja R). Należy przestrzegać tej zasady niezależnie od miejsca, w którym pojawia się symbol tj. tekst, wzory matematyczne, rysunki, tabele,
- ogólny symbol zapisu funkcji czyli np. f , a nie f . Nie dotyczy to jednak zapisu konkretnych funkcji np. $\cos\omega t$ a nie $\cos\omega t$,
- macierze, wektory, których elementami są wielkości fizyczne należy zapisywać dodatkowo czcionką półgrubą (bold) np. $\mathbf{R} = \begin{bmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{bmatrix}$, $\mathbf{U} = \begin{bmatrix} U_1 \\ U_2 \end{bmatrix}$,
- wskaźnik dolny, górny, prawo- i lewostronny, ale tylko gdy odnosi się do konkretnej wielkości fizycznej, czyli np. składowa x -owa indukcji magnetycznej B_x , a nie B_x ,
- wskaźniki górne i dolne oznaczające dowolną liczbę np. R_j , I^k , ale nie R_1 , I^2 .

2) Antykwy należy stosować w następujących sytuacjach:

- wszystkie cyfry,
- symbole konkretnych funkcji np. $\tan\omega t$, a nie $\tan\omega t$,
- operatory operacji matematycznych np. pochodne zwyczajne $\frac{dx}{dt}$, a nie $\frac{dx}{dt}$,
- symbole liczb o konkretnej wartości np. przenikalność elektryczna próżni $\varepsilon_0 = 8,8542 \cdot 10^{-12} \text{ F} \cdot \text{m}^{-1}$, a nie $\varepsilon_0 = 8,8542 \cdot 10^{-12} \text{ F} \cdot \text{m}^{-1}$,

- indeksy, jeżeli odnoszą się do: obiektów (fizycznych, geometrycznych), czyli, np. natężenie pola elektrycznego w punkcie A to E_A , a nie E_A , zjawisk lub stanów fizycznych, np. moment obciążenia to T_L , a nie T_L , do nazwisk czy też oznaczeń pierwiastków, np. straty w miedzi to P_{Cu} a nie P_{Cu} , do charakteru wielkości symbolizowanej przez literę źródłową, np. wartość maksymalna siły to F_{\max} , a nie F_{max} , oznaczeń jednostek miary np. $M\Omega$, a nie $M\Omega$.
- 3) W przypadku jednostek miar (które zawsze należy pisać antykwą) zapisując konkretną wartość liczbą należy podać jej wartość i jednostkę z zachowaniem następujących zasad:
- zapisując wartość liczbową wielkości fizycznej po spacji należy podać jej jednostkę, ale nie nazwę jednostki np. 10A, ale nie 10 amper czy też 10 amperów,
 - zapisując wartość liczbową słownie należy w tej konwencji podać też jednostkę np. dziesięć omów, ale nie dziesięć Ω
 - do oznaczeń jednostek nie wolno dopisywać indeksów, np. moc wyjściowa silnika wynosi $P = 100 \text{ kW}_{\text{out}}$. W takim przypadku należy zapisać $P_{\text{out}} = 100 \text{ kW}$,
 - jednostek nie należy umieszczać w nawiasach kwadratowych, np. $I = 1 \text{ [A]}$. Odstępstwem od tej zasady mogą być tabele, nagłówki kolumn, opisy osi na wykresach oraz w sporadycznych sytuacjach we wzorach matematycznych (ale tylko wówczas, gdy zależność matematyczna nie wskazuje w jakiej jednostce wystąpi wartość liczbowa). Przykłady odstępstw zamieszczono w podrozdziale 7.1.3.
- 4) W trakcie zapisu symboli wielkości matematycznych można stosować również szereg znaków diakrytycznych, jak również należy przestrzegać następujących zaleceń:
- wartości chwilowe podstawowych wielkości fizycznych używanych np. w elektrotechnice należy zapisać małymi literami, np. u , i , lub stosować zapis np. $u(t)$, lub stosować indeks „t” przy wielkości, np. U_t ,
 - wartości skuteczne wielkości okresowych należy zapisać dużą literą np. U , I ,

- wartości szczytowe funkcji zmiennej, amplitudę funkcji sinusoidalnej czasu należy zapisać jako np. U_m ,
- podkreślenie symboli reprezentujących wielkości fizyczne, których wartość liczbową jest liczbą zespoloną, przy czym podkreślenie dotyczy tylko literki źródłowej np. \underline{Z}_1 , a nie $\underline{Z_1}$,
- kreska nad literą źródłową oznacza wartość średnią, np. \bar{I} co jest równoważne I_{av} .

7.1.3. Rysunki i tabele

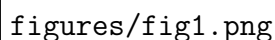
Tekst podstawowy w tabeli pisze się czcionką o rozmiarze 10 punktów, pojedyncza interlinia. Dane liczbowe – wyśrodkowane, dane tekstowe – wyrównane do lewej. Rysunki i tabele zamieszcza się wyśrodkowane na stronie, bez wcięcia pierwszego wiersza.

W akapicie poprzedzającym rysunek lub tabelę musi znajdować się krótki opis, czego dotyczy dany rysunek/tabela (odniesienie do rysunku/tabeli). Tytuły numeruje się zgodnie z kolejnością w danym rozdziale: numer_rozdziału.numer_tabeli/rysunku (np. rys. 2.1, tabela 3.5). W tytule rysunku/tabeli, zaczerpniętych z literatury, podaje się odnośnik do właściwej pozycji. Należy zadbać o to, aby opisy na rysunkach były czytelne (czcionka 8 punktów lub większa). Staraj się nie wymuszać numeracji, pozwól aby robił to za Ciebie L^AT_EX. Stosuj `\label` do znakowania obiektów, do których być może w tekście się będziesz odwoływał (rozdziały, rysunki, tabele, wzory, listingi ...). Odwołuj się do nich w tekście za pomocą funkcji `\ref{NazwaObiektu}`. Pamiętaj, że L^AT_EX korzystając z polecenia `latex` nie odczytuje z plików .jpg, .png ich wielkości. Polecenie `latex` generuje plik DVI. Jeżeli chcesz go używać zgłoś stosowny błąd. Aby się go pozbyć zdefiniuj wielkość natywną pliku grafiki. Polecamy jednak używanie zamiast polecenia `latex`, polecenie `pdflatex`, wówczas problem nie wystąpi.

Przykład: [...] co umożliwia wyznaczenie wartości napięcia. Na rys. 7.12 przedstawiono schemat obwodu z równolegle dołączoną pojemnością C_p .

Przykład: [...] Na rysunku 7.13 pokazano przykładową zależność prądów pasmowych i_{ph} bezszczotkowego silnika prądu stałego z magnesami trwałymi w funkcji położenia wirnika θ .

Przykład: [...] oraz indukcyjności wzajemnej. W tabeli 7.1 przedsta-



Rysunek 7.12: Tytuł rysunku, rozmiar 11 pkt., pojedyncza interlinia, akapit wyśrodkowany, bez wcięcia pierwszego wiersza. Na końcu tytułu rysunku/tabeli nie stawia się kropki [8]

wiono podstawowe parametry obwodu nieliniowego, zasilanego napięciem trójfazowym.

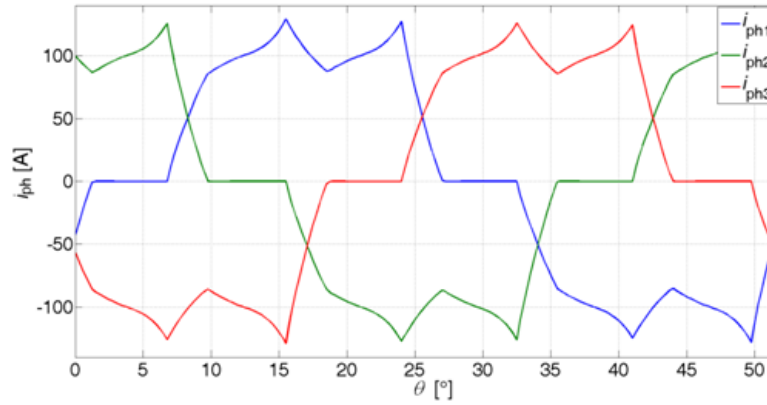
Tablica 7.1: Tytuł tabeli, rozmiar 11 pkt., pojedyncza interlinia, akapit wyrównany do lewej

U [V]	I [mA]	R , [k Ω]	L [mH]	R/R_{20}
13,6	7,29	3,94	100	1,25

7.1.4. Wzory matematyczne

Zmienne we wzorach pisze się czcionką pochyłą (styl edytora równań „Matematyka”) natomiast symbole, nie będące zmiennymi, czcionką prostą (styl „Tekst”). Rozmiary czcionek: normalny 12 punktów, indeks dolny/górny 9 pkt., indeks podrzędny 7 pkt., symbol 24 pkt., podsymbol 12 pkt. Separatorem dziesiętnym w liczbach jest przecinek, a nie kropka (dotyczy to również liczb pisanych w tekście akapitu). Poddawaj się w tym zakresie L^AT_EX’owi - pisz wzór, a poprawnie się utworzy.

Pod wzorem należy zamieścić objaśnienia użytych symboli (chyba, że znajdują się w wykazie na początku pracy). Wzory umieszcza się wyśrodkowane i numeruje zgodnie z kolejnością w danym rozdziale: (numer_rozdziału.numer_wzoru). Numery wzorów wyrównuje się do prawego marginesu. W akapicie poprzedzającym wzór musi znajdować się krótki opis, czego dotyczy dany wzór i – jeżeli potrzeba – odwołanie do literatury.



Rysunek 7.13: Tytuł rysunku, rozmiar 11 pkt., pojedyncza interlinia, akapit wyśrodkowany, bez wcięcia pierwszego wiersza. Na końcu tytułu rysunku/tabeli nie stawia się kropki [8]

Przykład: [...] wyznacza się, na podstawie wyrażenia (7.23). W nawiasach podano rozmiary czcionek używanych we wzorach

$$A(12) = \sum (24)m_{s(9)}N^{k_p(\tau)} \quad (7.23)$$

gdzie: m_s – masa próbki, N – natężenie oświetlenia, k_p – wykładnik potęgi ($k_p = 1,3 - 2,1$).

7.1.5. Listingi programów

W pracy dyplomowej możesz umieszczać fragmenty programów. Pamiętaj, aby umieszczać krótkie, tylko najważniejsze fragmenty kodów źródłowych. Zawsze je komentuj w treści pracy dyplomowej. Typowo w L^AT_EX kody źródłowe umieszczane są w środowisku `verbatim` (`\begin{verbatim}...\end{verbatim}`). Obecnie istnieje jednak bardziej nowoczesne i bardziej funkcjonalne środowisko `lstlisting` (wymaga zainstalowanego w systemie pakietu `listings`). Zwróć uwagę, że możesz kolorować składnię automatycznie za pomocą parametru `language`. W niniejszym dokumencie przedstawiono dwa przykłady listingów, Listing 14 to przykład kodu źródłowego Matlab, a poniżej Listing 15 dla Perl'a.

```
1 i = 1
2 p = 3
3 for i = 1:10
4     if i > 3
5         i=i+p
6     else
7         i=i+1
8     end
9 end
```

Listing 14: Listing programu Matlab

```
1 my $url = 'http://pei.prz.edu.pl';
2 use LWP::Simple;
3 my $content = get $url;
4 die "Couldn't get $url" unless defined $content;
5 print $content;
6 print "\n";
7 print "Length " + length($content)
```

Listing 15: Listing programu Perl

Z pewnością przeglądając źródło tego dokumentu zobaczysz, że kody źródłowe powinny mieć zdefiniowane parametry `label`, aby łatwo w tekście do nich się odwoływać. Numeracja linii jest w stylu domyślnie włączona (to przydatne, bo w treści pracy łatwo odwołać się dzięki temu do konkretnego wiersza w kodzie źródłowym), możesz je wyłączyć podając jako parametr `numbers=none`. Więcej szczegółów możesz odnaleźć w sekcji `\lstset` pliku arkusza styli.

7.1.6. Numerowanie i punktowanie

- 1) Pierwszy poziom (stosuje się numerowanie lub punktowanie). Formatowanie: akapit wyjustowany, wcięcie od lewej 0,75 cm, wysunięcie co 0,5 cm.

- 2) Znakiem numerowania jest liczba (z kropką lub nawiasem).
 - drugi poziom (stosuje się wyłącznie punktowanie). Formatowanie: akapit wyjustowany, wcięcie od lewej 1,25 cm, wysunięcie co 0,5 cm,
 - znakiem punktowania jest łącznik lub mała litera alfabetu (z nawiasem). Nie zaleca się stosowania kropek, strzałek itp.,
 - punktowane akapity rozpoczyna się minuskułą (małą literą), na końcu akapitu stawia się przecinek, ostatni punktowany akapit kończy się kropką.
- 3) Numerowane akapity rozpoczyna się majuskułą (wielką literą) i kończy kropką.
- 4) Należy zwrócić uwagę, aby nie rozdzielać numerowania/punktowania pomiędzy kolejnymi stronami tekstu.

7.2. Wykaz literatury

W wykazie literatury zamieszcza się wyłącznie pozycje, na które powołano się w pracy. Kolejność numerów w wykazie – zgodna z kolejnością pojawiania się danej pozycji w tekście.

Format akapitu: akapit wyjustowany, wysunięcie 0,75 cm. Prawidłowo opracowany wykaz został zaprezentowany w niniejszym dokumencie w odpowiednim rozdziale, oznaczonym jako „Literatura” (pozycja nr [?] to zasoby internetowe, [?] – książka, [?] – artykuł w czasopiśmie, [?] – karta katalogowa).

7.3. Wydruk pracy

Przed wydrukiem należy usunąć ewentualne błędy literowe i sprawdzić prawidłową interpunkcję. Przykładowo, łącznik zapisuje się za pomocą krótkiego minusa (np. badawczo-rozwojowy) natomiast myślnik – stosowany w zdaniach wtrąconych – zapisuje się za pomocą długiej pauzy. Dzielenie wyrazów według uznania Autora (można podzielić długie wyrazy, powodujące duże „rozstrzelenie” tekstu w poprzedzającym wierszu. Zaleca się usunięcie pojedynczych znaków na końcu wiersza oraz podwójnych spacji w tekście. Dla przedrostka „mikro” należy unikać stosowania litery „u” zamiast „μ”. Znak „μ” można otrzymać przytrzymując lewy Alt i wpisując na klawiaturze numerycznej 0181 (podobnie „stopień”: Alt-0176). W celu uniknięcia „rozstrzelenia” liczb i ich jednostek zaleca się używanie „twardej” spacji pomiędzy

liczbą i jednostką. Należy sprawdzić, czy tytuły podrozdziałów/zakresów nie zostały jako pojedyncze wiersze na poprzedniej stronie oraz czy rysunki/tabele i ich tytuły nie zostały rozdzielone pomiędzy kolejnymi stronami.

Pracę drukuje się dwustronnie. Zaleca się wydruk w kolorze. Przed wydrukiem należy ponumerować strony (czcionka 10 pkt., dół strony, akapit wyśrodkowany). Strony tytułowej oraz strony z podziękowaniem nie numeruje się. Spis treści rozpoczyna się od strony numer 3 (lub 5, jeżeli zamieszczono podziękowania).

8. Podsumowanie i wnioski końcowe

1 ÷ 3 stron merytorycznie podsumowanie najważniejszych elementów pracy oraz wnioski wynikające z osiągniętego celu pracy. Proponowane zalecenia i modyfikacje oraz rozwiązania będące wynikiem realizowanej pracy.

Ostatni akapit podsumowania musi zawierać wykaz własnej pracy dyplomanta i zaczynać się od sformułowania: „Autor za własny wkład pracy uważa: ...”.

Załączniki

Według potrzeb zawarte i uporządkowane uzupełnienie pracy o dowolny materiał źródłowy (wydruk programu komputerowego, dokumentacja konstrukcyjno-technologiczna, konstrukcja modelu – makiety – urządzenia, instrukcja obsługi urządzenia lub stanowiska laboratoryjnego, zestawienie wyników pomiarów i obliczeń, informacyjne materiały katalogowe itp.).

Literatura

- [1] <http://materialy.prz-rzeszow.pl/pracownik/pliki/34/sztuczna-inteligencja-cw9-siec-wielowarstw.pdf> (Dostęp 05.06.2022r.)
- [2] <http://materialy.prz-rzeszow.pl/pracownik/pliki/34/sztuczna-inteligencja-cw8-siec-jednowarstw.pdf> (Dostęp 05.06.2022r.)
- [3] Nielsen M.: Neural Networks and Deep Learning.
<http://neuralnetworksanddeeplearning.com/> (Dostęp. 05.06.2022r.)
- [4] <http://materialy.prz-rzeszow.pl/pracownik/pliki/34/sztuczna-inteligencja-cw10-przysp-uczenia.pdf> (Dostęp 13.06.2022r.)
- [5] Hagan T.M., Demuth H.B., Beale M.H.: Neural Network Design.
<https://hagan.okstate.edu/NNDesign.pdf> (Dostęp 13.06.2022r.)

STRESZCZENIE PRACY DYPLOMOWEJ WPISZ-RODZAJ-PRACY

**IMPLEMENTACJA WIELOWARSTOWEJ SIECI NEURONOWEJ
ORAZ ALGORYTMU WSTECZNEJ PROPAGACJI BŁĘDU Z
ADAPTACYJNYM WSPÓŁCZYNNIKIEM UCZENIA. BADANIE
WPŁYWU METAPARAMETRÓW SIECI NA EFEKTYWNOŚĆ
UCZENIA**

Autor: Mateusz Fesz, nr albumu: EF-167788

Opiekun: (tytuł naukowy przed) Imię i nazwisko opiekuna (tytuł po)

Słowa kluczowe: (max. 5 słów kluczowych w 2 wierszach, oddzielanych przecinkami)

Treść streszczenia po polsku

WPISZ-RODZAJ-PRACY THESIS ABSTRACT

TEMAT PRACY PO ANGIELSKU

Author: Mateusz Fesz, nr albumu: EF-167788

Supervisor: (academic degree) Imię i nazwisko opiekuna

Key words: (max. 5 słów kluczowych w 2 wierszach, oddzielanych przecinkami)

Treść streszczenia po angielsku