Politechnika Wrocławska

	Podstawy sieci neuronowych
Autor sprawozdania: **Igor Frysiak, nr albumu 272548** **Oskar Krupski, nr albumu 272511**	**Projekt – wariant:** **1**
Tytuł sprawozdania: Sprawozdanie z postępów prac – wariant 1.	**Etap nr:** **1**
Data oddania sprawozdania: **19 stycznia 2025**	**Ocena:**

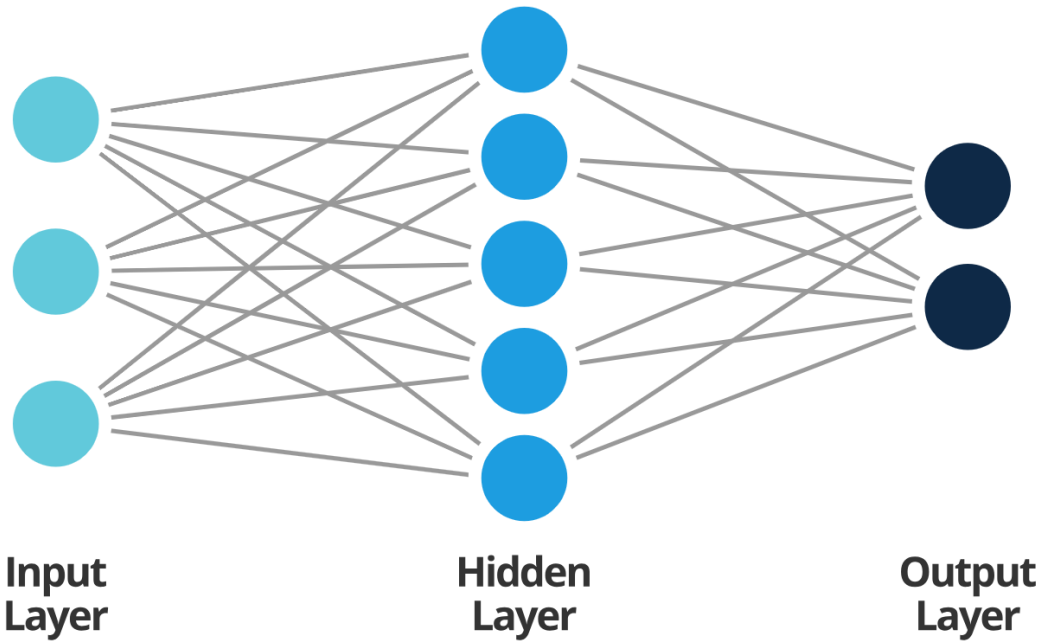
1. Wprowadzenie zadanie 1

Zadanie 1 z wariantu 1 polegało na zaprojektowaniu sztucznej sieci neuronowej typu MLP (Multi-Layered Perceptron, wielowarstwowa) do rozwiązania problemu klasyfikacji obrazów. Zestaw danych, dla którego wykonywane jest zadanie klasyfikacji to zbiór danych MNIST. Jest to dużych rozmiarów zestaw zawierający ręcznie pisane cyfry, który jest powszechnie używany w trenowaniu różnych sieci klasyfikujących obrazy, takich jak na przykład w naszym zadaniu. Zawiera on 60,000 przykładów do trenowania i 10,000 przykładów do testowania.

2. Realizacja zadania 1

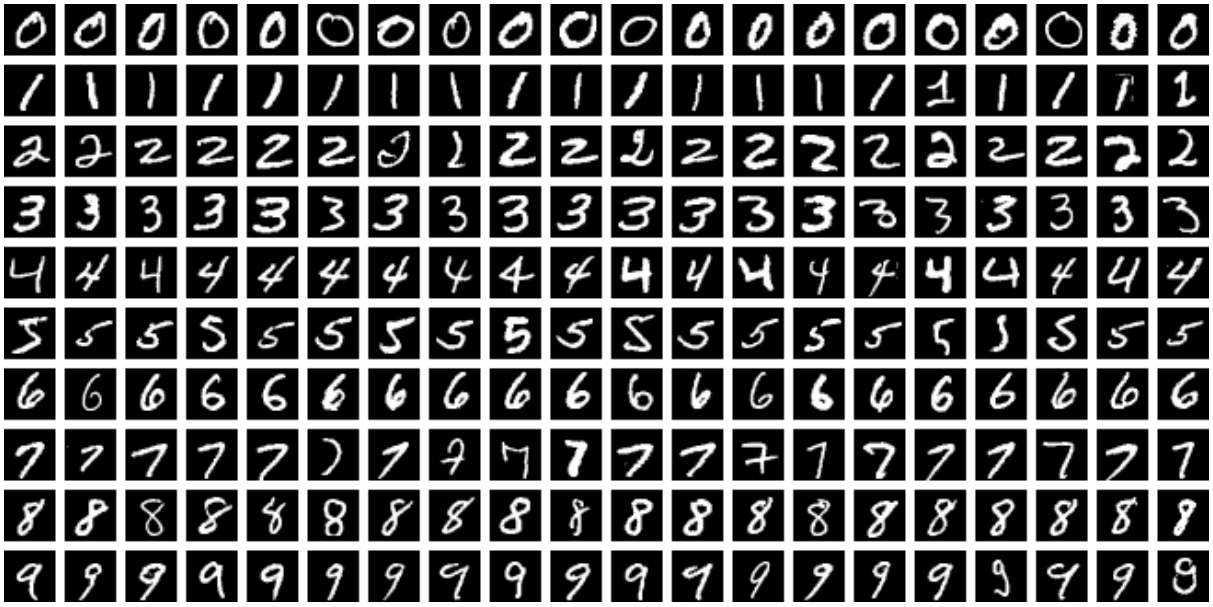
2.1. Wstęp teoretyczny

Nasze zadanie, tak jak zostało już wcześniej wspomniane, polega na zaprojektowaniu sieci neuronowej do klasyfikacji danych z zestawu MNIST. W prostszych słowach, chcemy utworzyć program, który będzie losowo pobierał z zestawu testowego cyfry i poprawnie je rozpoznawał. Nie mamy tutaj za cel postawionej 100% dokładności, co byłoby bardzo trudne do osiągnięcia bez korzystania z gotowych, specjalnie do tego przeznaczonych bibliotek. Naszym postawionym sobie celem jest osiągnięcie dokładności na poziomie co najmniej 90%, co wydawało się nam realistycznym wynikiem.



Rysunek 1. Przykładowy model poglądowy sieci neuronowej.

Nasza sieć będzie wyglądała podobnie do tej powyżej, z różnymi ilościami neuronów w poszczególnych warstwach. W warstwie wejściowej dla naszego zadania będzie to zawsze 784 neuronów. Musi tak być, ponieważ każdy neuron w tej warstwie odpowiada pojedynczemu pikselowi w obrazie na wejściu. Mamy tutaj na myśli obrazy z ręcznie narysowanymi cyframi, które nasza sieć ma na celu rozpoznać. Obrazy te mają rozmiar 28x28 pikseli, które są „spłaszczane" do postaci wektorowej o rozmiarze 784.
Obrazy zostały pobrane ze strony:
https://www.kaggle.com/datasets/hojjatk/mnist-dataset?resource=download



Rysunek 2. Przykładowe obrazy z bazy MNIST.

W naszej warstwie ukrytej ustaliliśmy 20 neuronów (choć liczbę tę można łatwo konfigurować w skrypcie), a w warstwie wyjściowej 10 neuronów. Innymi parametrami, które można zmieniać w celu badań, jest współczynnik uczenia oraz ilość przejść przez dane (epoki).

2.2. Realizacja zadania

W tej sekcji opiszemy dokładniej budowę oraz działanie naszej sieci.

Zaczynając od początku, biblioteki z których korzystaliśmy to:

- numpy – w celu obsługi tabeli, macierz i wykonywania działań matematycznych,
- matplotlib – w celu tworzenia wykresów (MSE od learning rate),
- idx2numpy – w celu wczytywania danych z formatu IDX.

Aby nauczyć naszą sieć do rozpoznawania danych, potrzebujemy funkcje aktywacji. Są to funkcje matematyczne, które stosuje się na wyjściu każdego neuronu. Potrzebujemy ich w naszym programie, ponieważ bez nich sieć otrzymywałaby pomiędzy warstwami jedynie zestawy równań liniowych, co nie przynosiłoby żadnych właściwych efektów.

W naszym skrypcie zaimplementowaliśmy kilka funkcji aktywacji, w zależności od których otrzymujemy różne dokładności:

- Sigmoidalna,
- ReLU (Rectified Linear Unit),
- Leaky ReLU,
- Softmax.

gdzie pierwsze 3 funkcje wykorzystujemy w warstwie ukrytej, a softmax wykorzystujemy w warstwie wyjściowej. Funkcja softmax często stosowana jest właśnie w warstwie wyjściowej, aby przekształcić surowe wyniki na prawdopodobieństwa.

Proces trenowania sieci składa się z kilku etapów:

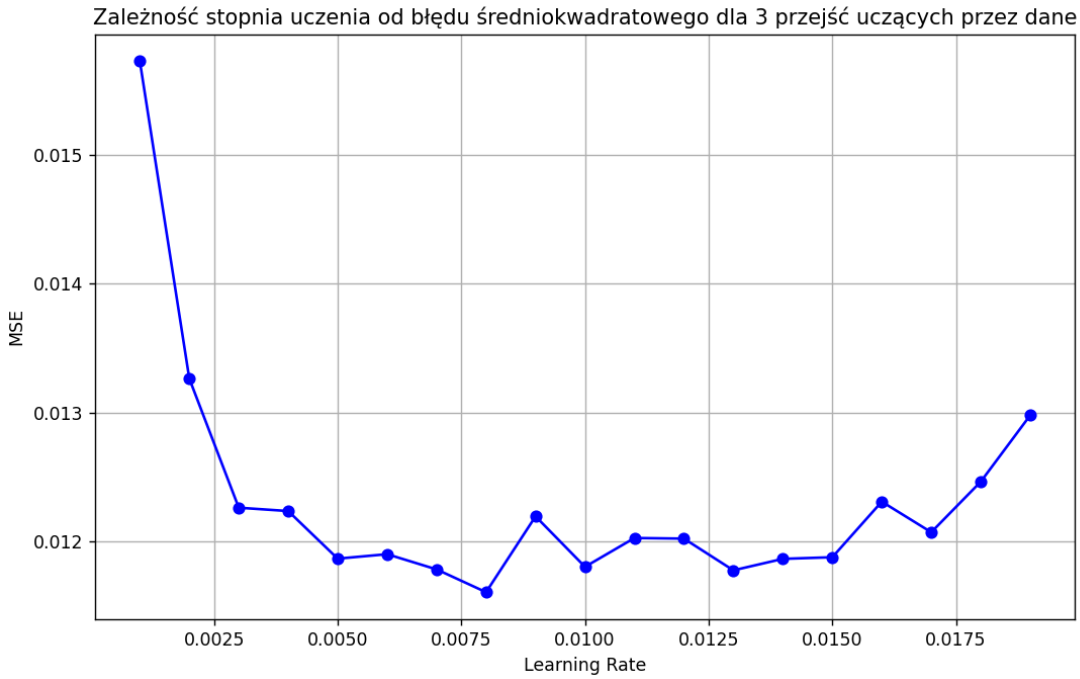
- Na początku inicjalizujemy wagi oraz biasy dla naszych warstw. Wagi generowane są losowo, używając rozkładu normalnego, a biasy inicjalizowane jako 0.
- Forward propagation – dla obrazów na wejściu propagujemy w przód, obliczając wartości w warstwie ukrytej i wyjściowej, korzystając przy z odpowiednich funkcji aktywacji.
- Obliczanie błędu – przy pomocy błędu średniokwadratowego MSE obliczamy błąd naszej sieci.
- Back propagation – przechodzimy przez sieć w kierunku wstecznym, dostosowując przy tym wagi oraz biasy w celu zminimalizowania błędu. Korzystamy tutaj z metody gradient descent.

Liczba epok określa ilość pełnych przejść przez dane, dla których powtarzane są powyższe działania.

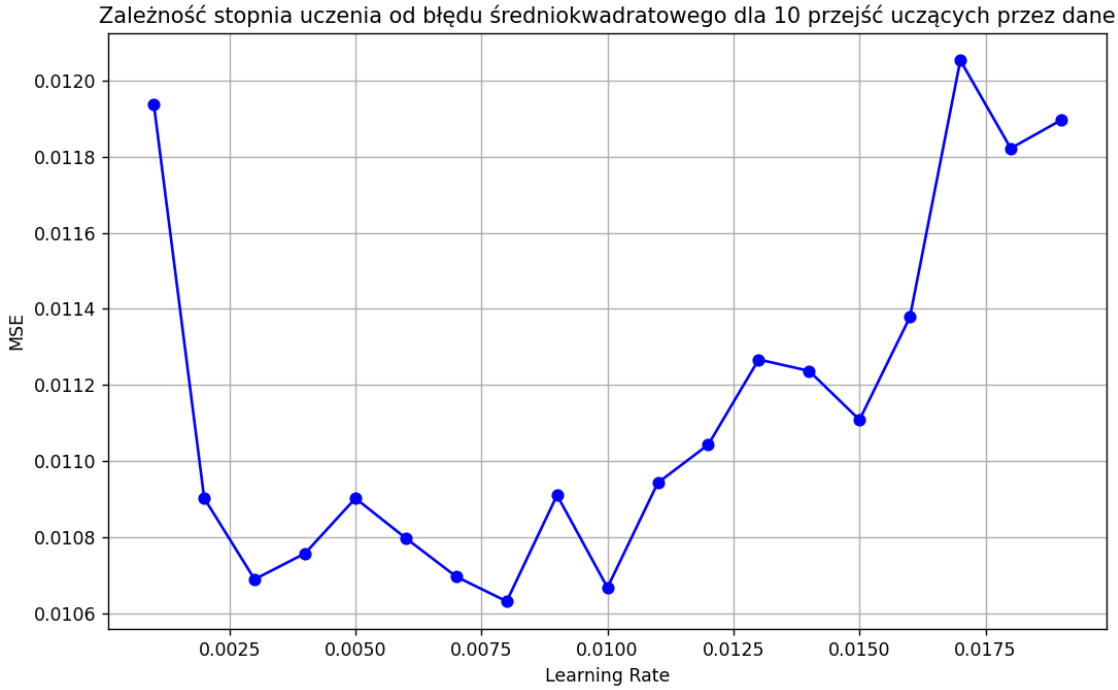
Na sam koniec przeprowadzamy również ocenę naszych wyników, które wizualizowane są za pomocą wykresu MSE od współczynnika uczenia. Pozwala nam to na analizę sieci i wyciągnięcie wniosków, dla jakich wartości współczynnika uczenia błąd średniokwadratowy jest najmniejszy.

2.3. Wyniki

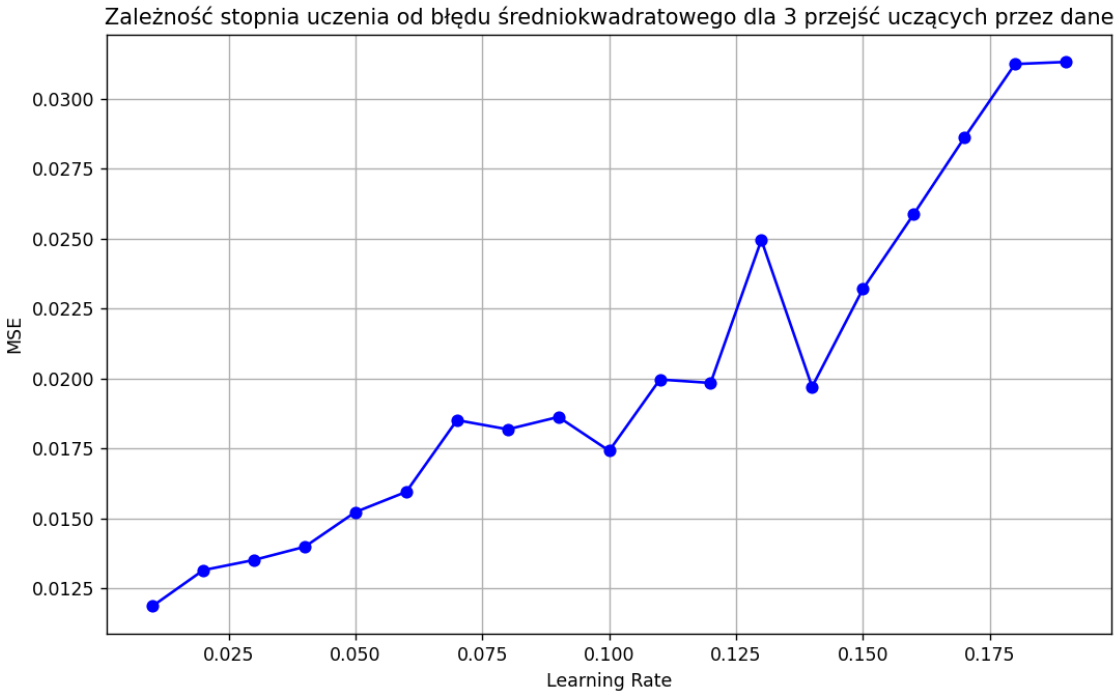
- Funkcja aktywacji sigmoidalna:



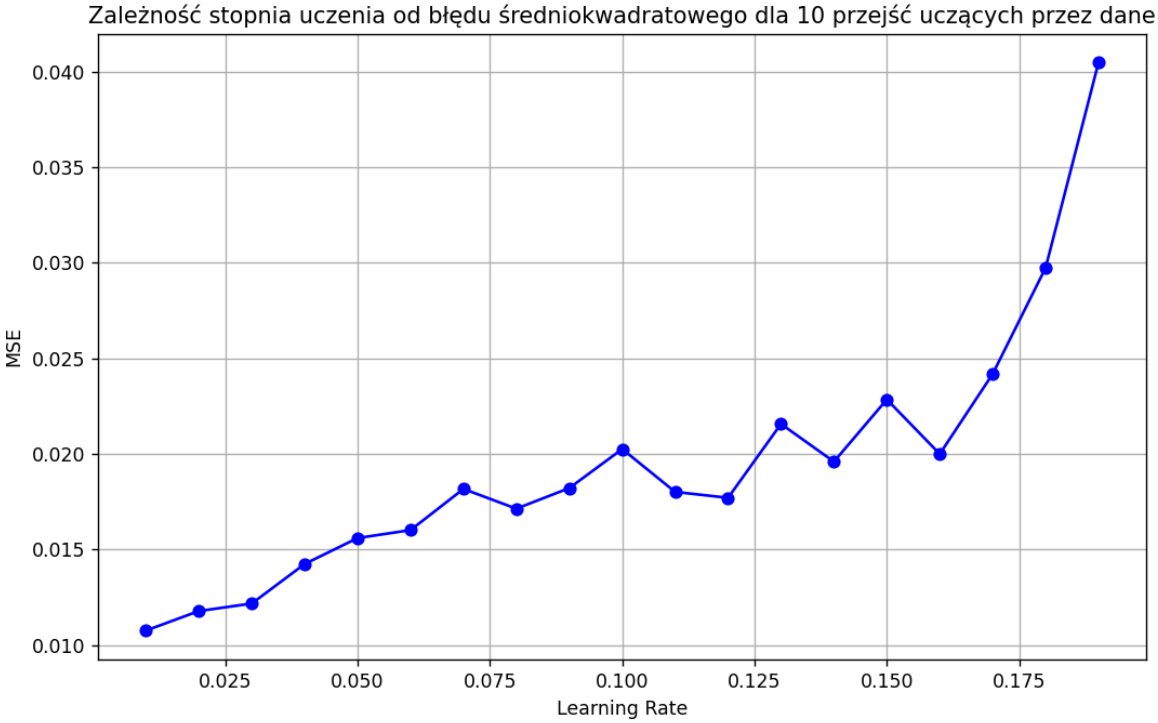
Rysunek 3. Funkcja aktywacji sigmoidalna, współczynnik nauczania = 0.001, 3 epoki. Najwyższa osiągnięta dokładność: 93.37%



Rysunek 4. Funkcja aktywacji sigmoidalna, współczynnik nauczania = 0.001, 10 epok. Najwyższa osiągnięta dokładność: 93.78%

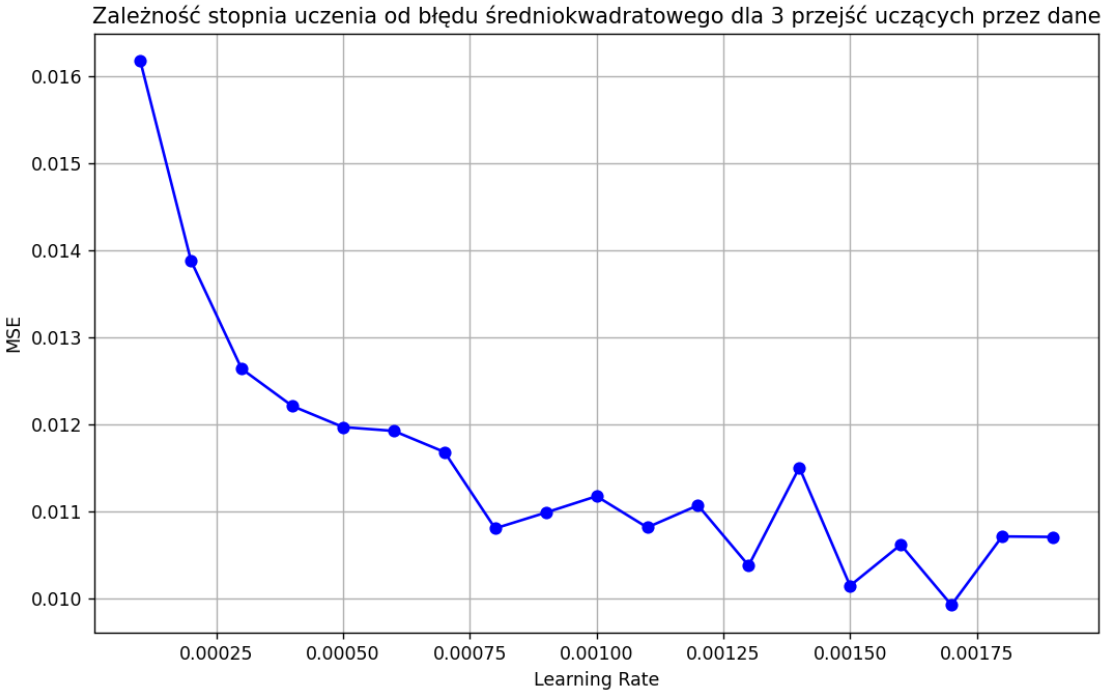


Rysunek 5. Funkcja aktywacji sigmoidalna, współczynnik nauczania = 0.01, 3 epoki. Najwyższa osiągnięta dokładność: 93.22%

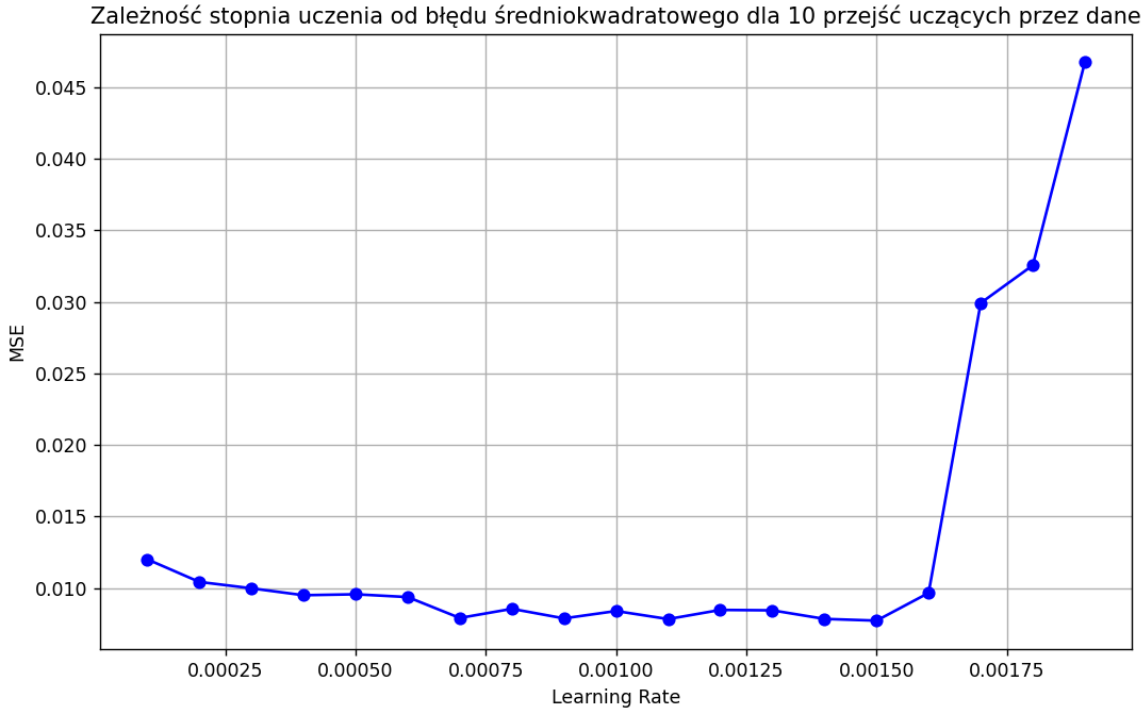


Rysunek 6. Funkcja aktywacji sigmoidalna, współczynnik nauczania = 0.01, 10 epok. Najwyższa osiągnięta dokładność: 94.01%

- Funkcja aktywacji ReLU:

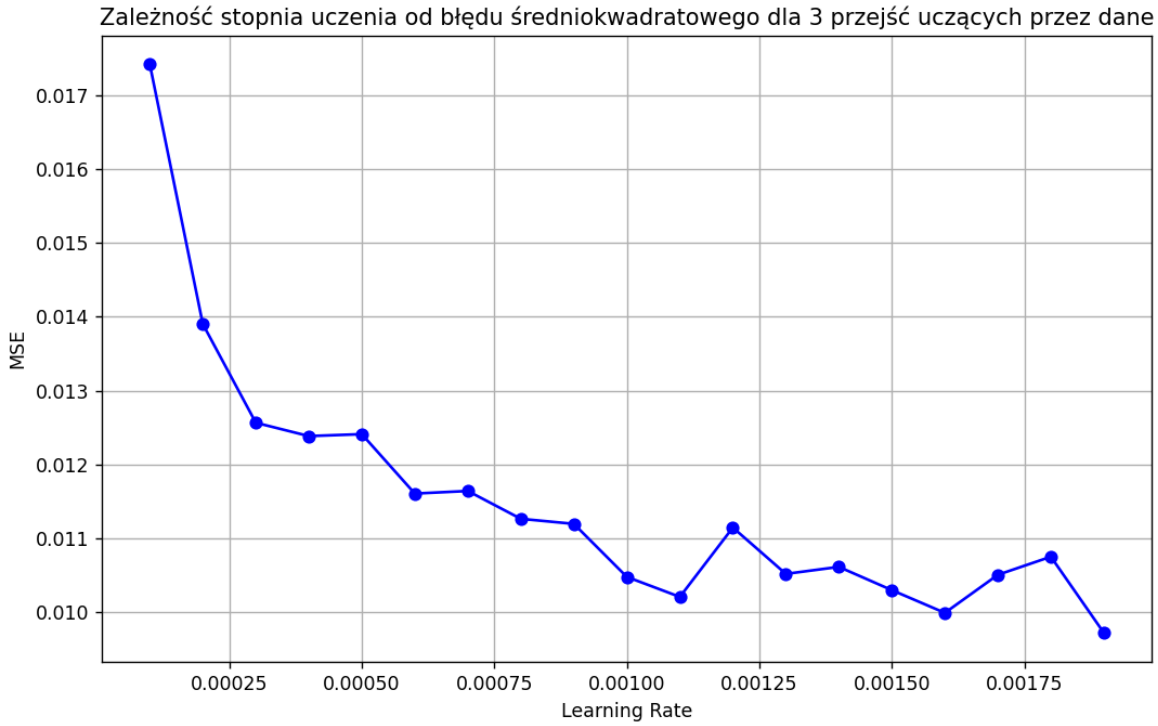


Rysunek 7. Funkcja aktywacji ReLU, współczynnik nauczania = 0.0001, 3 epoki. Najwyższa osiągnięta dokładność: 93,95%

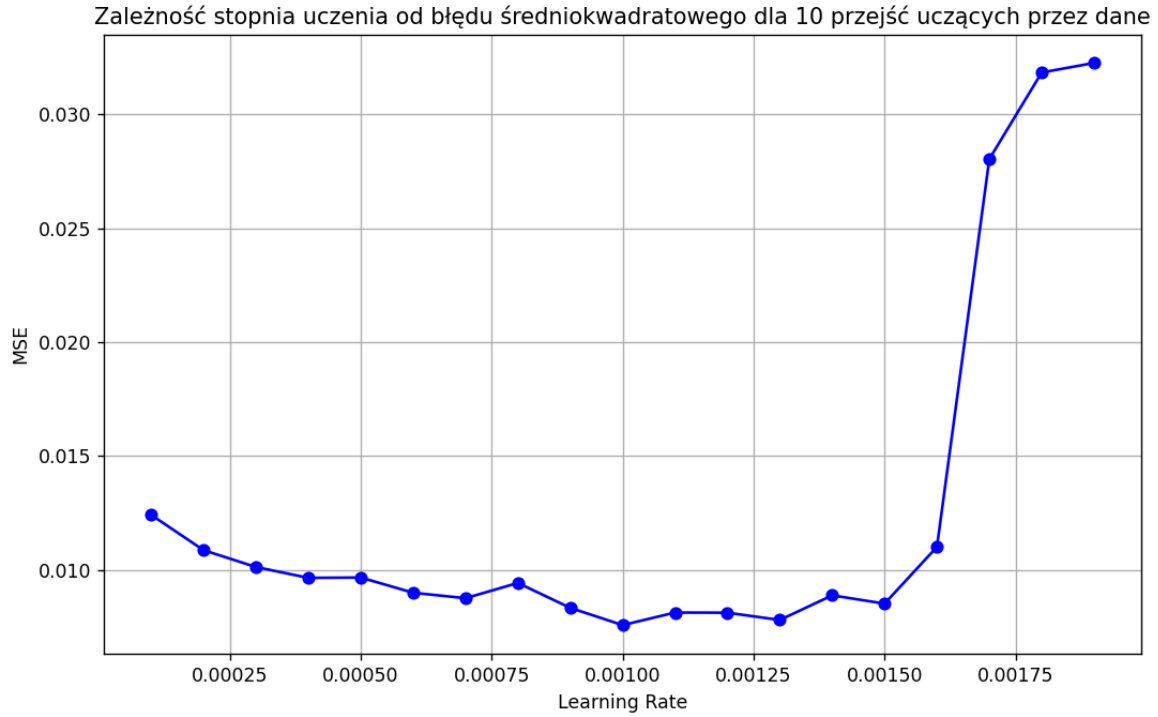


Rysunek 8. Funkcja aktywacji ReLU, współczynnik nauczania = 0.0001, 10 epoki. Najwyższa osiągnięta dokładność: 95.49%

- Funkcja aktywacji Leaky ReLU:



Rysunek 9. Funkcja aktywacji Leaky ReLU, współczynnik nauczania = 0.0001, 3 epoki. Najwyższa osiągnięta dokładność: 93.75%



Rysunek 10. Funkcja aktywacji Leaky ReLU, współczynnik nauczania = 0.0001, 10 epoki. Najwyższa osiągnięta dokładność: 95.5%

3. Wprowadzenie zadania 2

Zadanie 2 polegało na zaprojektowaniu sieci neuronowej w celu aproksymacji dwuwymiarowej funkcji Ackley'a, na podstawie zadanego zestawu losowych punktów i wartości funkcji w punkcie. Odwzorowania wartości tej funkcji należało dokonać na zakresie wejściowym od -2 do 2.

4. Realizacja zadania 2

4.1. Wstęp teoretyczny

Funkcja Ackley'a definiowana jest wzorem:

$$f_A(x_1, x_2) = -20\exp\left[-0.2\sqrt{0.5(x_1^2 + x_2^2)}\right] - \exp\left[0.5(\cos(2\pi x_1) + \cos(2\pi x_2))\right] + \exp[1] + 20$$

Jest to funkcja dwuwymiarowa, w której wartość docelowa zależy od dwóch zmiennych wejściowych. Zakres tej funkcji był ograniczony do $[-2, 2]$ w obu wymiarach, a wartości funkcji zostały znormalizowane do zakresu $[-1, 1]$, co ułatwiło proces trenowania modelu neuronowego.

Do realizacji użyto biblioteki:

- numpy – w celu obsługi tabeli, macierz i wykonywania działań matematycznych,
- matplotlib – w celu tworzenia wykresów 2D i 3D.

4.2. Budowa sieci neuronowej

Nasza sieć neuronowa składała się z:

- Warstwy wejściowej o 2 neuronach.
- Jednej warstwy ukrytej z 20 neuronami.
- Warstwy wyjściowej z 1 neuronem.

4.3. Proces trenowania

1. **Generowanie danych**: Wygenerowano 1000 przykładów treningowych, gdzie wejścia zostały losowo wygenerowane z przedziału $[-2, 2]$, a wartości funkcji Ackley'a obliczono i znormalizowano.
2. **Funkcja aktywacji**: Dostępne w kodzie funkcje aktywacji to ReLU, Leaky ReLU oraz Sigmoid, Swish, Elu.
3. **Propagacja w przód**: Obliczano wartości w warstwie ukrytej i wyjściowej, uwzględniając funkcje aktywacji.

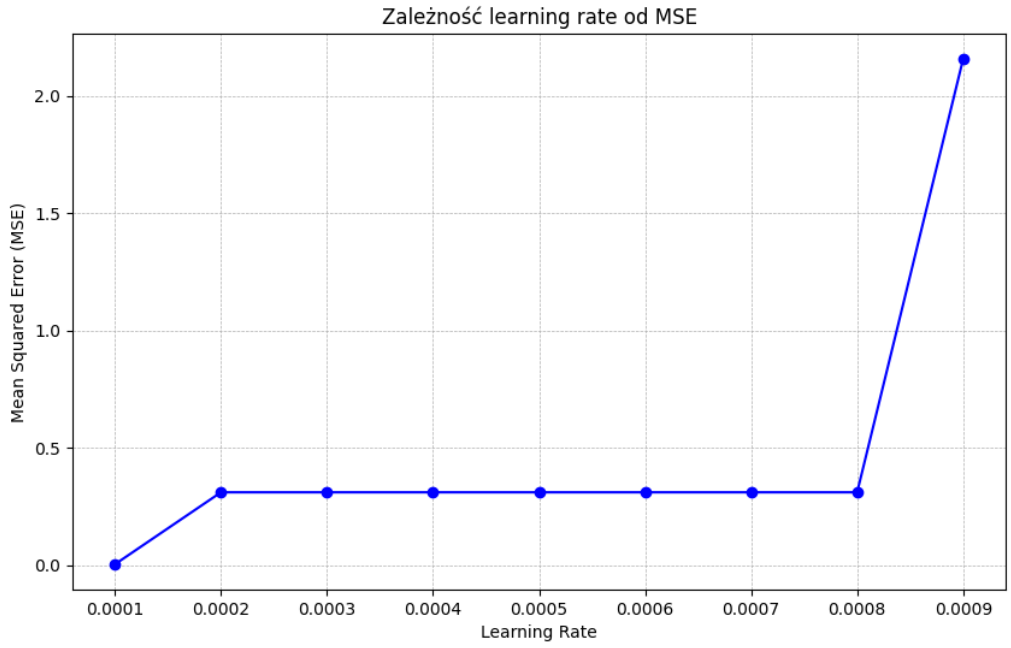
4. **Obliczanie błędu**: Używano funkcji błędu MSE (Mean Squared Error) do oceny wyników sieci.

5. **Propagacja wsteczna**: Wagi i biasy były aktualizowane metodą gradient descent. W celu zminimalizowania nadmiernego dopasowania modelu do danych, uwzględniono regularyzację L2.

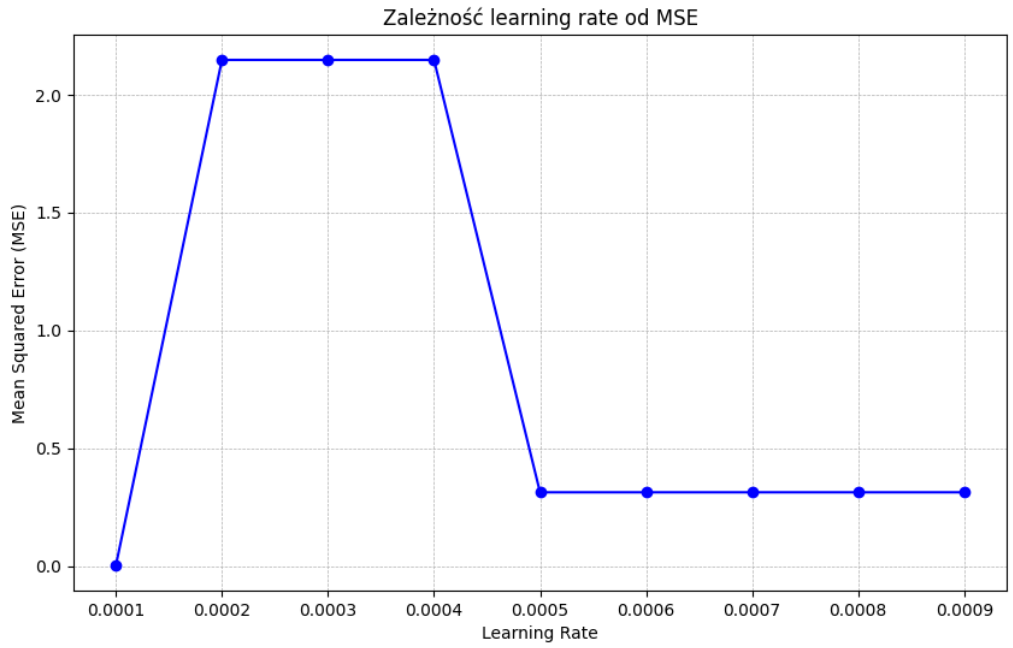
6. **Epoki**: Proces powtarzano przez 3000 epok.

4.4. Wyniki

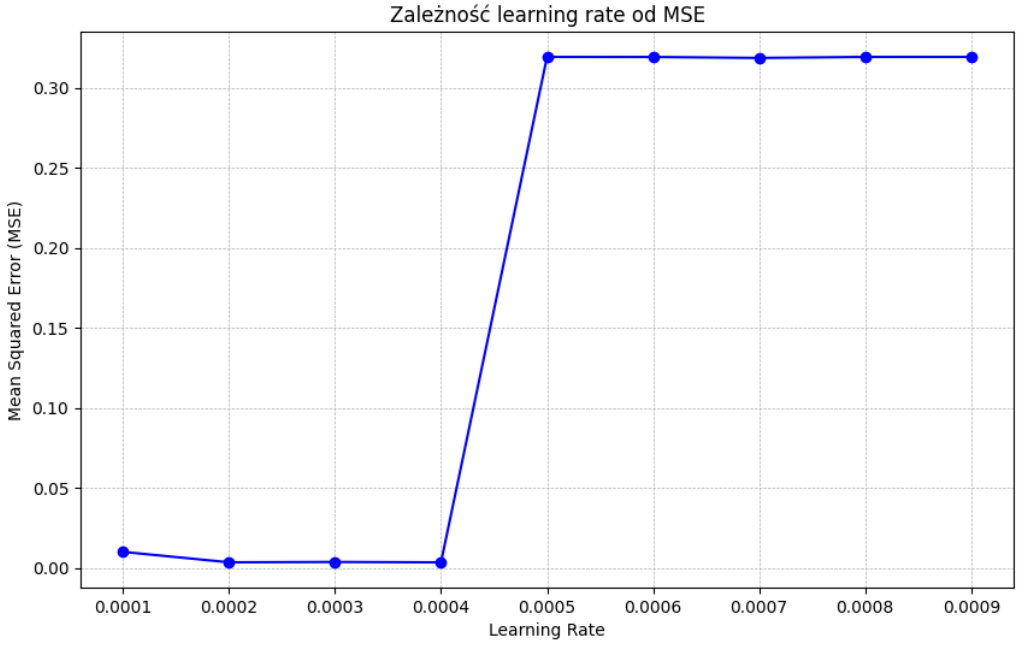
Model testowany jest w regularnych odstępach 0.01.
Wyniki predykcji porównano z rzeczywistymi wartościami funkcji Ackley'a, wizualizując je na wykresach 2D oraz 3D.



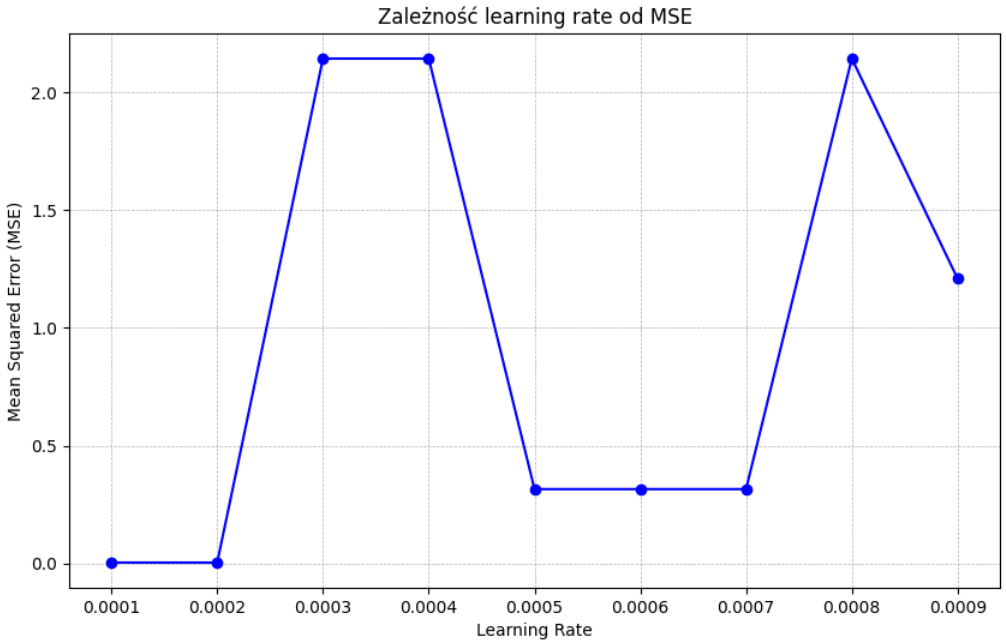
Rysunek 11 Zależność learning rate od MSE dla funkcji aktywacji ReLU, epochs 3000. Dla learning rate 0.0001, MSE = 0.0035.



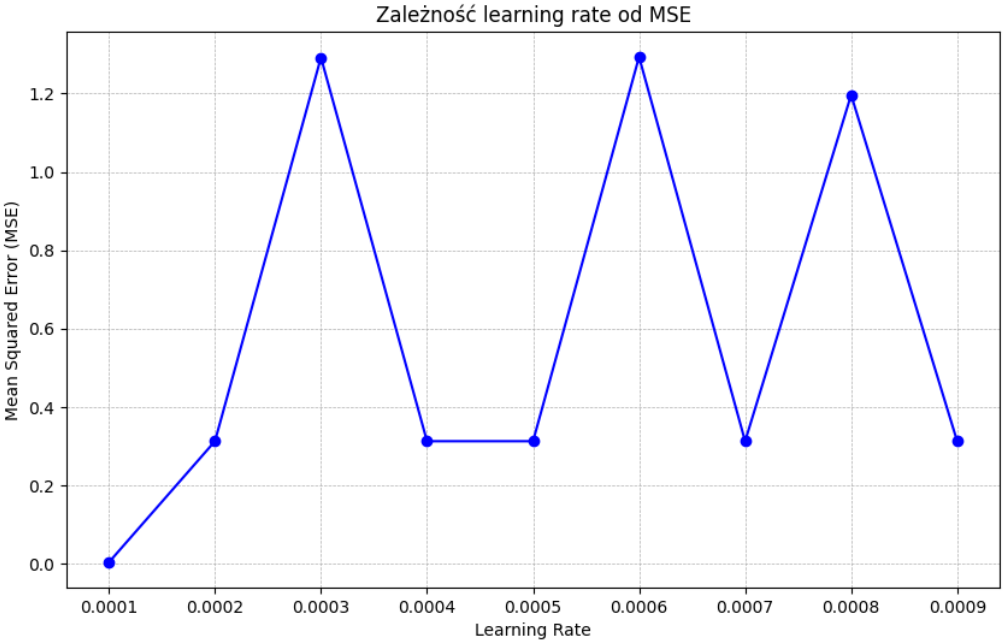
Rysunek 12 Zależność learning rate od MSE dla funkcji aktywacji LeakyReLU, epochs 3000. Dla learning rate 0.0001, MSE = 0.0037.



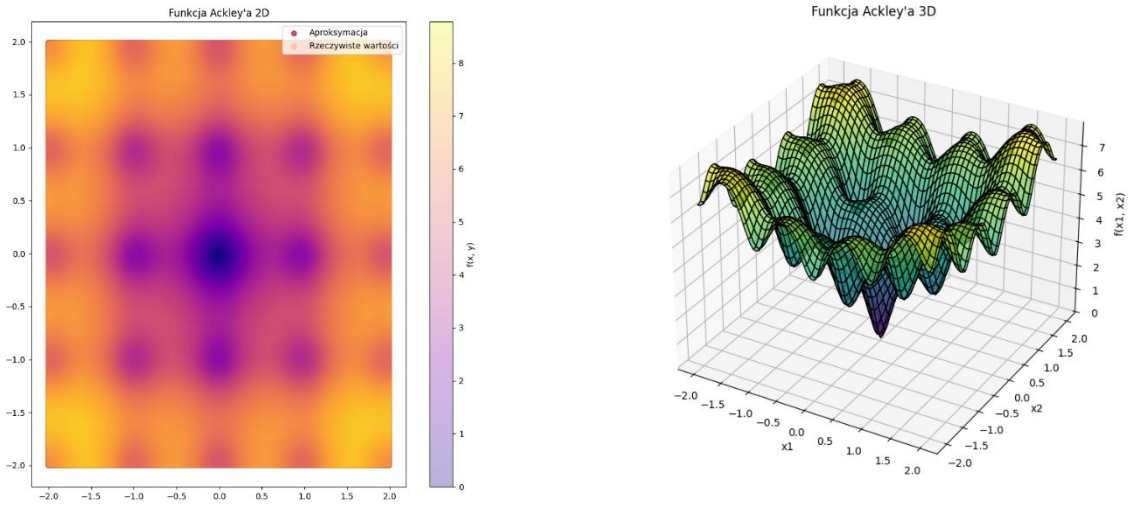
Rysunek 13 Zależność learning rate od MSE dla funkcji aktywacji Sigmoid, epochs 3000. Dla learning rate 0.0003, MSE = 0.0037.



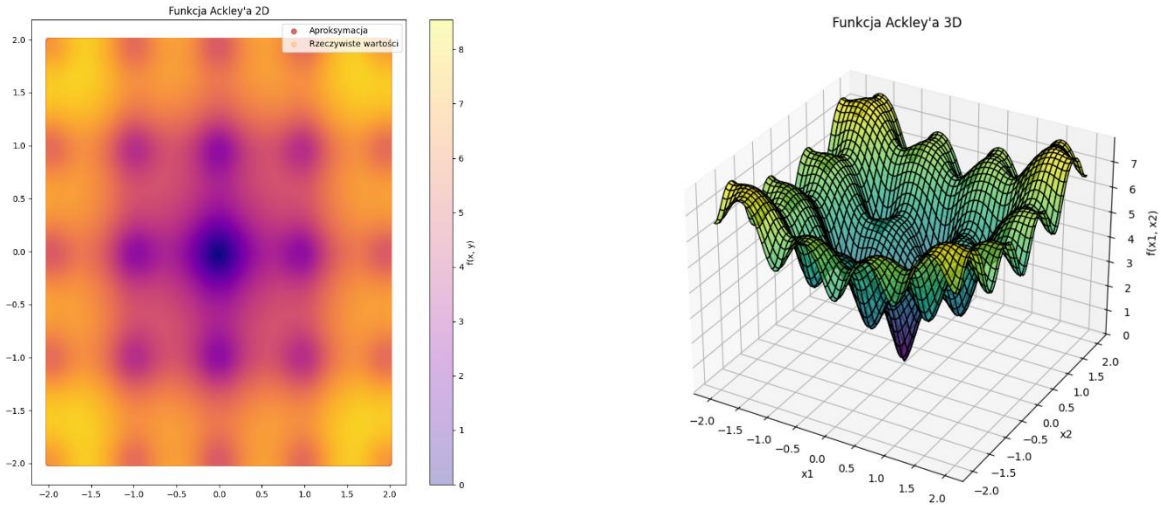
Rysunek 14 Zależność learning rate od MSE dla funkcji aktywacji Swish, epochs 3000. Dla learning rate 0.0001, MSE = 0.0041.



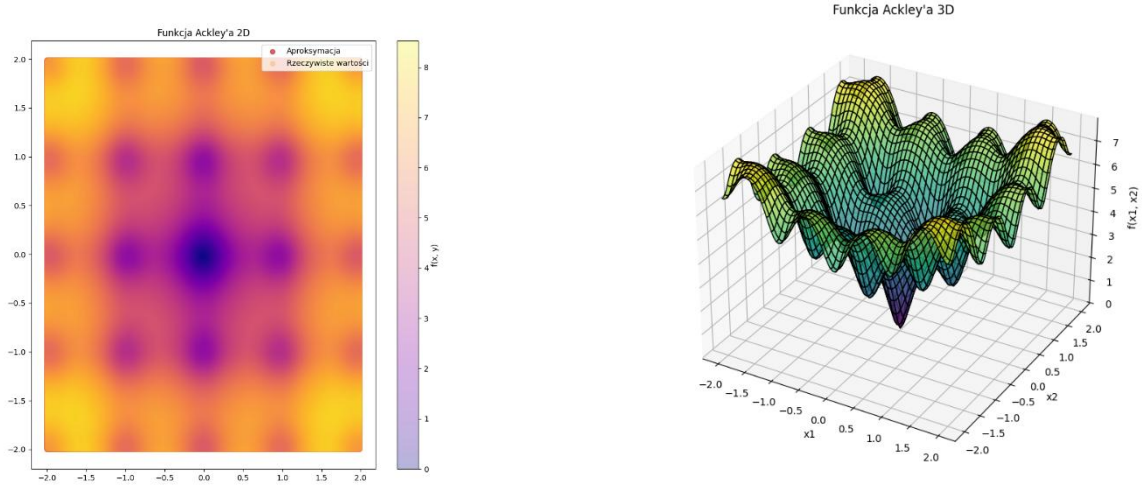
Rysunek 15 Zależność learning rate od MSE dla funkcji aktywacji Elu, epochs 3000. Dla learning rate 0.0001, MSE = 0.0044.



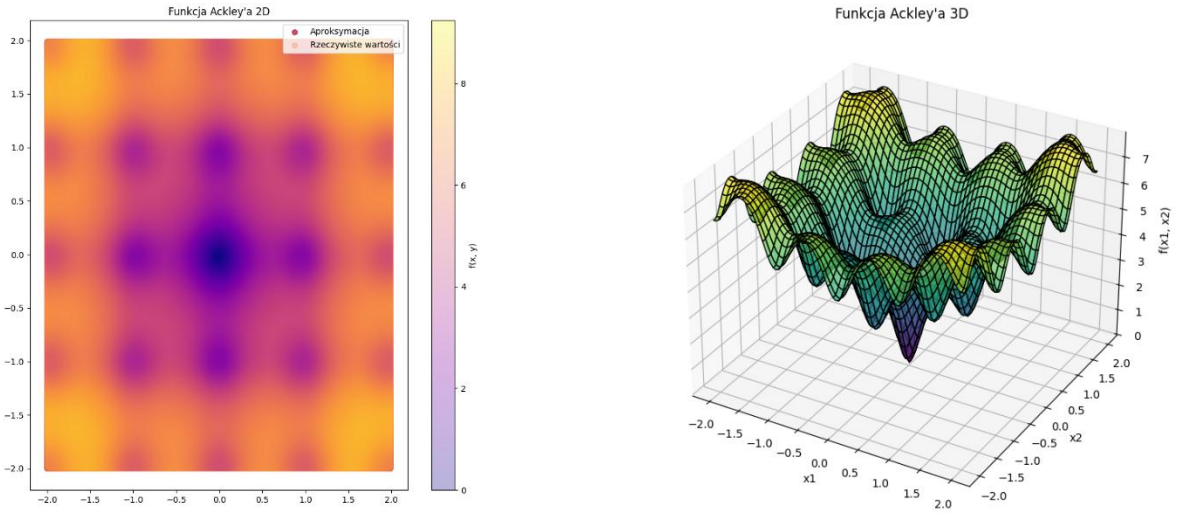
Rysunek 16 Wykresy predykcji 2D i 3D dla funkcji aktywacji ReLU. Epochs 3000.



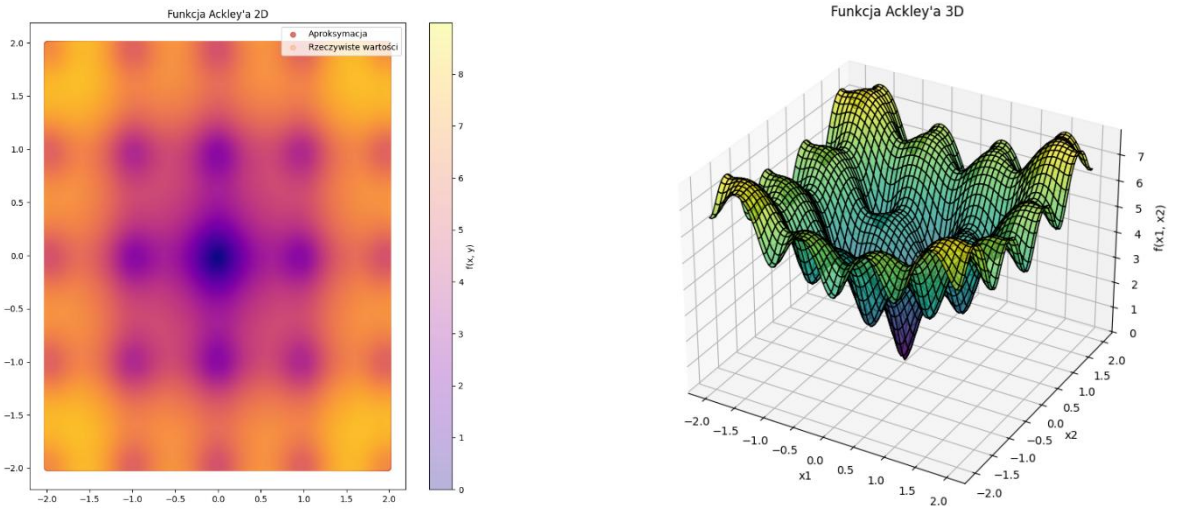
Rysunek 17 Wykresy predykcji 2D i 3D dla funkcji aktywacji LeakyReLU. Epochs 3000.



Rysunek 18 Wykresy predykcji 2D i 3D dla funkcji aktywacji Sigmoid. Epochs 3000.

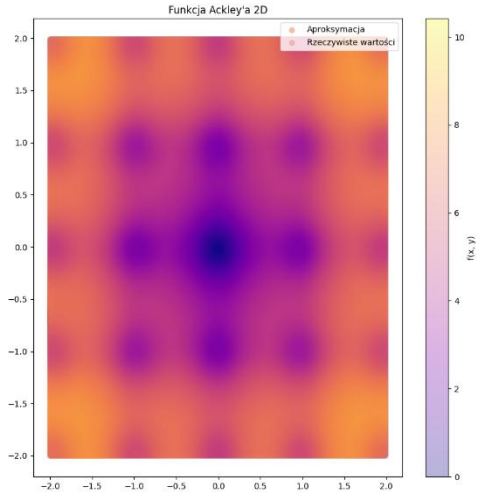


Rysunek 19 Wykresy predykcji 2D i 3D dla funkcji aktywacji Swish. Epochs 3000..

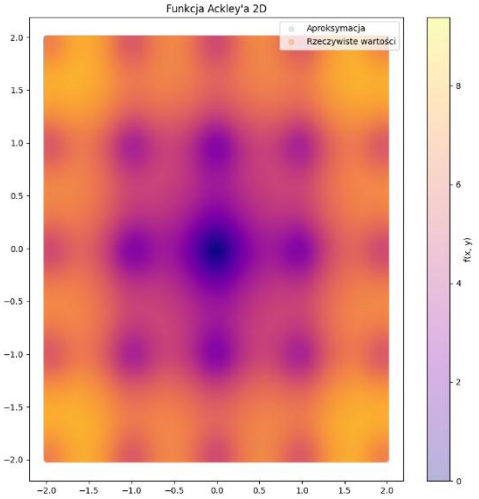


Rysunek 20 Wykresy predykcji 2D i 3D dla funkcji aktywacji Elu. Epochs 3000.

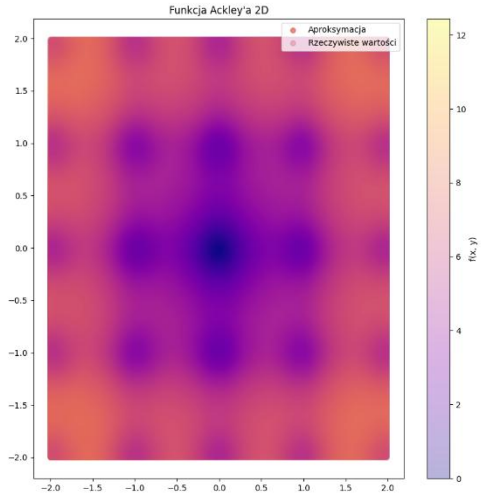
Wyniki dla 5 epochs:



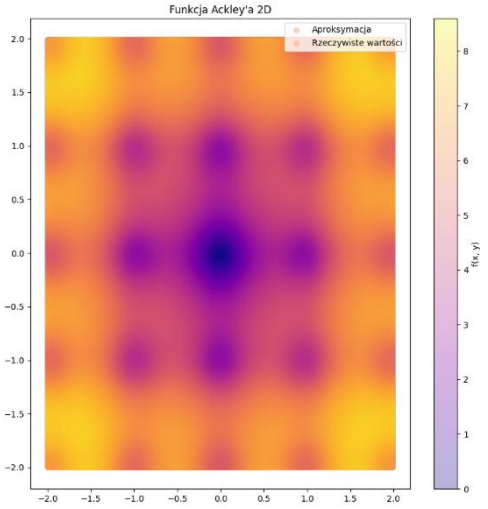
Rysunek 21 Wykres predykcji 2D dla funkcji aktywacji ReLU. Epochs 5. MSE = 0.1.



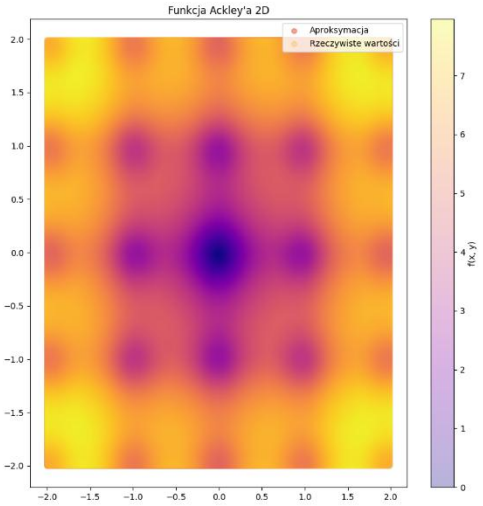
Rysunek 22 Wykres predykcji 2D dla funkcji aktywacji LeakyReLU. Epochs 5. MSE = 0.078.



Rysunek 23 Wykres predykcji 2D dla funkcji aktywacji Sigmoid. Epochs 5. MSE = 0.03.



Rysunek 24 Wykres predykcji 2D dla funkcji aktywacji Swish. Epochs 5. MSE = 0.088.



Rysunek 25 Wykres predykcji 2D dla funkcji aktywacji Elu. Epochs 5. MSE = 0.065.

5. Analiza i wnioski

- Sigmoidalna funkcja aktywacji pozwala na dobre dopasowanie, jednak dla małych współczynników uczenia lepsze są funkcje ReLU i Leaky ReLU.
- Wysokie współczynniki uczenia negatywnie wpływają na stabilność działania funkcji ReLU i Leaky ReLU.
- Liczba epok znacząco wpływa na końcową dokładność modelu. Wyższa liczba epok daje bardziej dokładne rezultaty niż mniejsza, co pokazuje, że sieć potrzebuje odpowiedniego czasu, aby się nauczyć. Można to zaobserwować w obu sieciach.
- W zadaniu 2 sieć dla małej ilości epok uczy się najszybciej dla funkcji aktywacji Sigmoid, a następnie Elu, LeakyReLU, Swish, a najwolniej dla ReLU.
- Współczynnik uczenia wymaga precyzyjnego doboru. Dla zbyt niskich wartości proces uczenia jest niewystarczający, a dla zbyt wysokich funkcje aktywacji przestają działać poprawnie.
- W zadaniu 1 najlepsze wyniki osiągnięto dla funkcji aktywacji Leaky ReLU z dokładnością 95.5% przy współczynniku uczenia 0.0001 i 10 epokach.
- W zadaniu 2 najlepsze wyniki osiągnięto dla funkcji aktywacji ReLU, następnie Leaky ReLU, Sigmoidalna, Swish, Elu.
- Przy stosunkowo niewielkiej liczbie neuronów, w warstwie ukrytej, obie sieci były w stanie wykonać oba zadania poprawnie.
- W zadaniu 2 dla funkcji aktywacji: ReLU, LeakyReLU, Elu większa ilość neuronów w warstwie ukrytej (50) wprowadza niestabilność w predykcji, błąd MSE zachowuje się nieprzewidywalnie, na zmianę rośnie i maleje i ostatecznie sieć nie jest nauczona.
- W zadaniu 2 wykresy dla 3000 epok świadczą o bardzo dobrym dopasowaniu aproksymacji do rzeczywistej wartości.