Sztuczne sieci neuronowe

Opis i implementacja

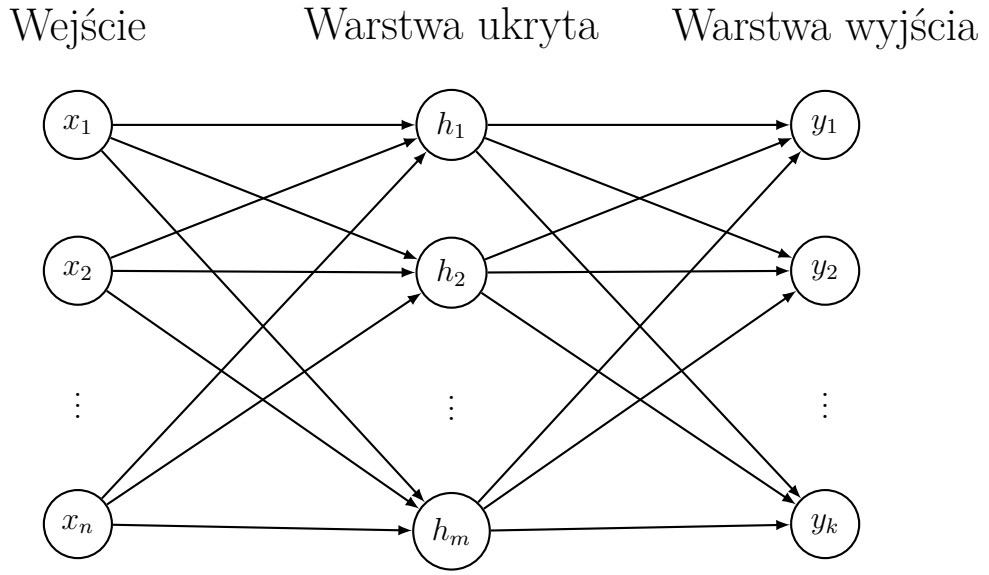
Wstęp

Sztuczne sieci neuronowe, a w szczególności oparty na nich perceptron wielowarstwowy (*ang. Multilayer Perceptron, MLP*), są kluczowymi narzędziami w dziedzinie uczenia maszynowego służącymi między innymi do aproksymacji nieliniowych funkcji. Perceptron dwuwarstwowy, składający się z warstwy ukrytej i wyjściowej, posiada zdolność przybliżania dowolnej funkcji ciągłej, o czym mówi twierdzenie o uniwersalnej aproksymacji [1]. Dzięki temu jest on szeroko stosowany w zadaniach praktycznych, takich jak właśnie przybliżanie funkcji matematycznych, rozpoznawanie wzorców czy klasyfikacja danych.

1. Opis perceptronu dwuwarstwowego

1.1 Architektura sieci neuronowej

Perceptron dwuwarstwowy jest strukturą hierarchiczną złożoną z połączonych ze sobą jednostek przetwarzających, zwanych neuronami. Sieć składa się z trzech głównych elementów: wektora wejść neuronu, warstwy ukrytej oraz warstwy wyjściowej. Informacja przepływa jednokierunkowo od wejścia do wyjścia, a każdy neuron realizuje operację ważonego sumowania sygnałów wejściowych oraz zastosowania funkcji aktywacji.



Każde połączenie pomiędzy neuronami posiada przypisaną wagę określającą wpływ sygnału wejściowego na aktywację kolejnych neuronów. Proces przetwarzania danych w sieci może zostać opisany w postaci wektorowo-macierzowej, co zostanie przedstawione wraz z głębszym wyjaśnieniem w następnej sekcji.

[1] Universal approximation theorem

1.2 Matematyczny opis perceptronu

Zamiast liczyć każdy neuron osobno, co byłoby bardzo nieefektywne, stosuje się zapis wektorowo-macierzowy. Pozwala on wykonywać obliczenia dla całej warstwy neuronów jednocześnie, w postaci jednego działania algebraicznego.

W tym celu definiujemy trzy podstawowe elementy: wektor wejściowy, wektor aktywacji warstwy ukrytej oraz wektor wyjściowy:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \qquad \mathbf{h} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_m \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{bmatrix}$$

Każda warstwa neuronowa składa się z połączeń pomiędzy neuronami, które opisujemy za pomocą wag oraz obciążeń (biasów). Wagi określają, jak silnie dany neuron wejściowy wpływa na neuron w kolejnej warstwie.

Macierz wag W zawiera wszystkie te połączenia jednocześnie. Każdy wiersz odpowiada jednemu neuronowi warstwy (tu: warstwy ukrytej), a każda kolumna odpowiada jednemu wejściu:

$$W = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1n} \\ w_{21} & w_{22} & \dots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \dots & w_{mn} \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

Działanie warstwy neuronowej można teraz opisać w prosty sposób: każdy neuron oblicza ważoną sumę wszystkich wejść, dodaje do niej bias, a następnie przepuszcza wynik przez funkcję aktywacji ψ. Zapisać to można w postaci jednego równania dla całej warstwy:

$$\mathbf{h} = \psi\left(W\mathbf{x} + \mathbf{b}\right)$$

gdzie $W\mathbf{x}$ oznacza jednoczesne obliczenie wszystkich sum ważonych, a $\mathbf{b}$ przesuwa wartości aktywacji neuronów w sposób niezależny od wejścia. Funkcja ψ jest stosowana osobno do każdego elementu wektora i wprowadza nieliniowość, dzięki czemu sieć może modelować bardziej złożone zależności niż zwykłe przekształcenia liniowe.

Dokładnie ten sam mechanizm powtarza się w warstwie wyjściowej. Wektor ukryty $\mathbf{h}$ jest przekształcany w końcowy wynik $\mathbf{y}$:

$$\mathbf{y} = W_2\mathbf{h} + \mathbf{b}_2$$

W ten sposób sieć neuronowa działa jako sekwencja dwóch operacji: najpierw liniowego mieszania informacji (macierze wag i biasy), a następnie nieliniowego przekształcenia (funkcja aktywacji). To właśnie połączenie tych dwóch kroków pozwala modelowi uczyć się złożonych zależności i schematów w danych.

2. Uczenie sieci neuronowej

W celu zobrazowania procesu uczenia sieci neuronowej rozważmy problem aproksymacji funkcji matematycznej, gdzie celem jest nauczenie modelu przewidywania wartości funkcji na podstawie jej argumentów.

2.1 Wstęp i założenia

Przyjmijmy, że chcemy nauczyć sieć neuronową aproksymacji funkcji opisującej rozkład Laplace'a, zdefiniowanej wzorem:

$$f(x) = \frac{1}{2b} e^{-\frac{|x-\mu|}{b}}$$

gdzie μ oznacza parametr położenia, natomiast b parametr skali. W rozważanym przypadku przyjęto:

$$\mu = 0, \qquad b = 1, \qquad x \in [-8, 8]$$

W warstwie ukrytej zastosowano funkcję aktywacji sigmoidalnej jako:

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

Wagi sieci zostały zainicjalizowane losowo przy użyciu metody rozkładu jednostajnego z przedziału zależnego od liczby neuronów wejściowych n, której celem jest utrzymanie stabilnych wartości sygnałów podczas propagacji w przód oraz propagacji wstecznej gradientu. Biasy zainicjalizowano zerami, natomiast losowa inicjalizacja wag powoduje, że początkowe predykcje sieci mają charakter losowy.

$$W_{ij} \sim \mathcal{U}\left(-\sqrt{\frac{1}{n}}, \sqrt{\frac{1}{n}}\right), \qquad b_j = 0$$

2.2 Funkcja straty

Aby możliwe było uczenie sieci neuronowej, konieczne jest zdefiniowanie funkcji straty określającej różnicę pomiędzy wartościami przewidywanymi przez model a rzeczywistymi wartościami funkcji. To jej wartości będziemy chcieli zminimalizować

W problemach często stosowany jest błąd średniokwadratowy (*Mean Squared Error*, MSE), definiowany wzorem:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

gdzie y_i oznacza rzeczywistą wartość funkcji, $\hat{y}_i$ wartość przewidywaną przez model, natomiast n liczbę próbek.

Kwadratowa natura funkcji MSE sprawia, że model jest szczególnie czuły na większe błędy predykcji, co pozwala skuteczniej minimalizować duże odchylenia od rzeczywistych wartości funkcji.

2.3 Mini-Batch Stochastic Gradient Descent

Nauka sieci neuronowej sprowadza się do optymalizacji parametrów modelu w celu minimalizacji funkcji straty. W procesie uczenia zastosowano metodę mini-batch stochastic gradient descent (mini-batch SGD), będącą wariantem stochastycznego spadku gradientu. W podejściu tym gradient funkcji straty nie jest liczony na pojedynczej próbce, lecz na losowych podzbiorach - pakietach (batchach), a następnie uśredniany.

Dla pojedynczego batcha gradient funkcji straty można zapisać jako:

$$\nabla L_{batch} = \frac{1}{m} \sum_{i=1}^{m} \nabla L_i$$

gdzie m oznacza rozmiar batcha, a ∇L_i gradient błędu dla pojedynczej próbki.

Parametry sieci aktualizowane są iteracyjnie według reguły:

$$\theta \leftarrow \theta - \eta \nabla L_{batch}$$

gdzie θ oznacza zbiór wszystkich wag i biasów w sieci, natomiast η to współczynnik uczenia (*ang. learning rate*).

Zastosowanie mini-batchy pozwala na zmniejszenie wariancji estymacji gradientu, co stabilizuje proces uczenia oraz przyspiesza jego przebieg w porównaniu do pełnego spadku gradientu.

Dodatkową zastosowaną modyfikacją jest przycinanie gradientu (*ang. gradient clipping*), które polega na ograniczeniu jego wartości składowych do przedziału $[-1, 1]$, co zapobiega problemowi eksplodujących gradientów.

2.4 Propagacja wsteczna gradientu

Propagacja wsteczna gradientu (*ang. backpropagation*) jest algorytmem służącym do efektywnego obliczania gradientów funkcji straty względem parametrów sieci neuronowej. Metoda polega na propagacji błędu od warstwy wyjściowej do warstwy wejściowej z wykorzystaniem reguły łańcuchowej rachunku różniczkowego.

Przebieg algorytmu można opisać następującymi krokami dla pojedynczej próbki pakietu:

- Obliczenie predykcji sieci dla wejścia x w procesie propagacji w przód (forward pass).
- Wyznaczenie błędu na warstwie wyjściowej jako różnicy pomiędzy wartością przewidywaną $\hat{y}$ a wartością rzeczywistą y.
- Propagacja błędu do warstwy ukrytej poprzez przemnożenie go przez transponowaną macierz wag warstwy wyjściowej.
- Uwzględnienie pochodnej funkcji aktywacji sigmoidalnej w warstwie ukrytej w celu przeskalowania gradientu.
- Obliczenie gradientów wag i biasów dla warstwy ukrytej oraz warstwy wyjściowej.
- Aktualizacja ∇L_{batch} o obliczone gradienty próbki.

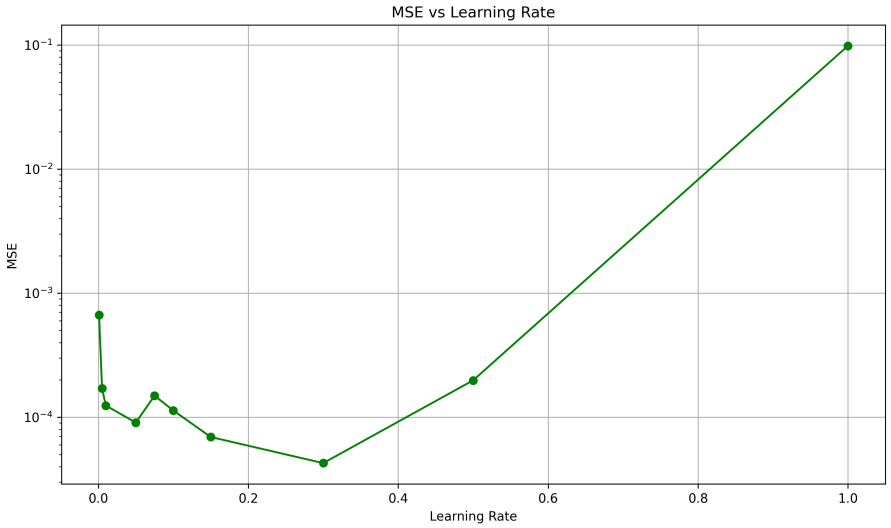
3. Algorytm w działaniu

W celu uzyskania możliwie dokładnej aproksymacji funkcji przeprowadzono walidację hiperparametrów sieci neuronowej, analizując wpływ poszczególnych parametrów na jakość aproksymacji oraz stabilność procesu uczenia. Na ich podstawie zostanie wytrenowany finalny model, którego wyniki zostaną porównane z funkcją rzeczywistą. Zbiór danych podzielono na część treningową (70%), walidacyjną (15%) oraz testową (15%).

Ze względu na losowy charakter przypisywania wag oraz stochastyczną naturę procesu uczenia, dla każdego testowanego współczynnika uczenia przeprowadzono wiele eksperymentów i uśredniono uzyskane wyniki, aby uzyskać wiarygodną ocenę wpływu tego parametru na jakość aproksymacji.

3.1 Wybór współczynnika uczenia

Współczynnik uczenia (*ang. learning rate*) określa długość kroku wykonywanego podczas aktualizacji parametrów modelu metodą mini-batch SGD. Parametr ten ma kluczowy wpływ na szybkość oraz stabilność procesu uczenia. Zbyt małe wartości prowadzą do bardzo wolnej zbieżności, natomiast zbyt duże mogą powodować oscylacje funkcji straty oraz przeskakiwanie poszukiwanego minimum.

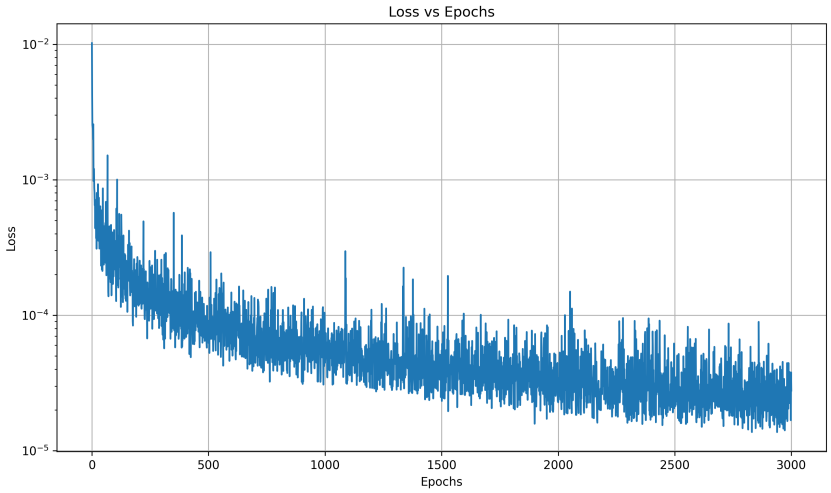


Na podstawie wykresu można zauważyć, że dla małych wartości η model osiąga relatywnie wysoki końcowy błąd, ponieważ proces optymalizacji przebiega zbyt wolno. Wraz ze wzrostem współczynnika uczenia jakość aproksymacji poprawia się, osiągając najniższe wartości w okolicach $\eta = 0.3$. Dalsze zwiększanie wartości parametru prowadzi jednak do pogorszenia wyników i niestabilności procesu uczenia.

W dalszych eksperymentach przyjęto wartość $\eta = 0.3$, która zapewniała najlepszy kompromis pomiędzy szybkością zbieżności a końcową wartością błędu. Wartość ta jest relatywnie wysoka w porównaniu do typowych konfiguracji stosowanych w gradientowych metodach optymalizacji, jednak w analizowanym przypadku okazała się skuteczna prawdopodobnie ze względu na niewielką złożoność aproksymowanej funkcji oraz zastosowanie mechanizmu przycinania gradientu.

3.2 Dobór liczby epok treningowych

Liczba epok określa, ile razy cały zbiór treningowy zostaje wykorzystany podczas procesu uczenia sieci neuronowej. Parametr ten wpływa bezpośrednio na stopień dopasowania modelu do danych oraz końcową jakość aproksymacji funkcji.



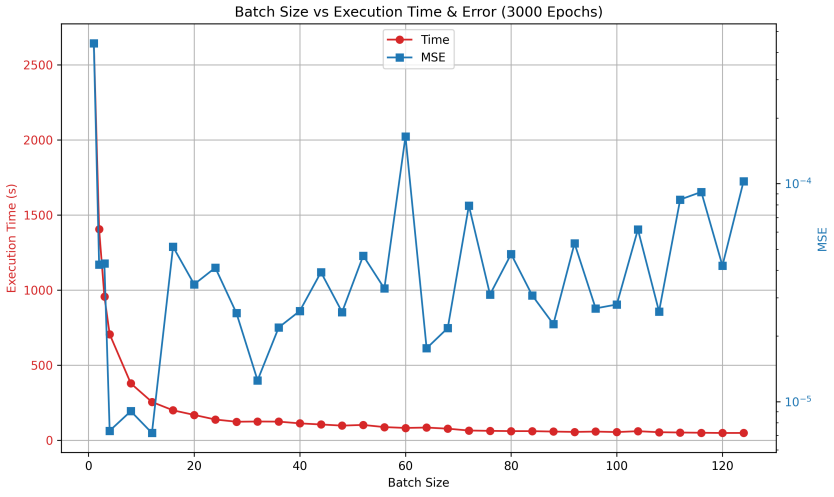
Na podstawie wykresu można zauważyć szybki spadek funkcji straty w początkowej fazie treningu, co świadczy o efektywnym uczeniu podstawowych zależności. W kolejnych epokach tempo poprawy maleje, choć ogólny trend pozostaje spadkowy.

Krzywa błędu jest niemonotoniczna i zawiera oscylacje wynikające ze stochastycznego charakteru mini-batch SGD, jednak mimo tego model zachowuje stopniową zbieżność.

W okolicach 3000 epoki dalsza poprawa jest już niewielka, dlatego przyjęto tę wartość jako kompromis między dokładnością a czasem treningu.

3.3 Wpływ wielkości pakietu treningowego

W implementacji wykorzystano mini-batch SGD, w którym gradient obliczany jest na niewielkich podzbiorach danych treningowych. Rozmiar batcha wpływa zarówno na stabilność estymacji gradientu, jak i na czas wykonywania pojedynczej epoki.

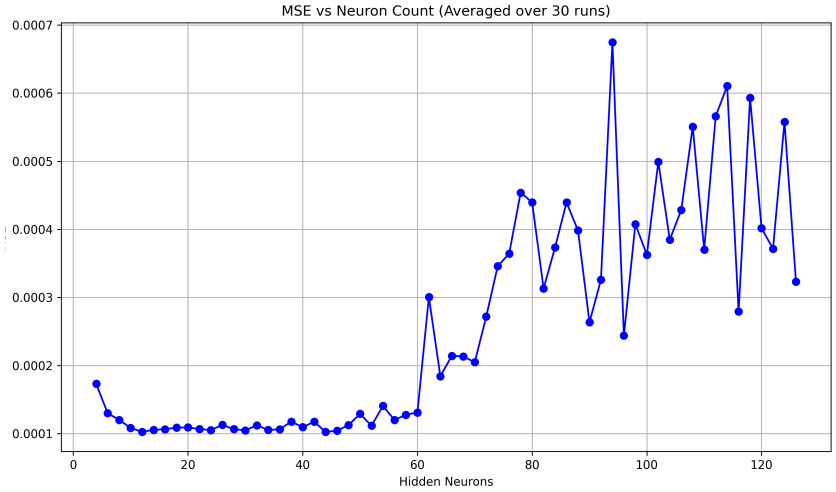


Dla bardzo małych batchy czas treningu był większy ze względu na dużą liczbę aktualizacji parametrów. Z kolei bardzo duże batch size prowadziły do pogorszenia jakości aproksymacji, wynikającego z nadmiernego wygładzenia gradientu.

Najlepszy kompromis pomiędzy czasem działania a dokładnością uzyskano dla batch size równego 16, który został przyjęty w finalnym modelu.

3.4 Wpływ liczby neuronów w warstwie ukrytej

Ostatnim analizowanym parametrem była liczba neuronów w warstwie ukrytej, odpowiadająca za pojemność modelu i jego zdolność do aproksymowania bardziej złożonych zależności.

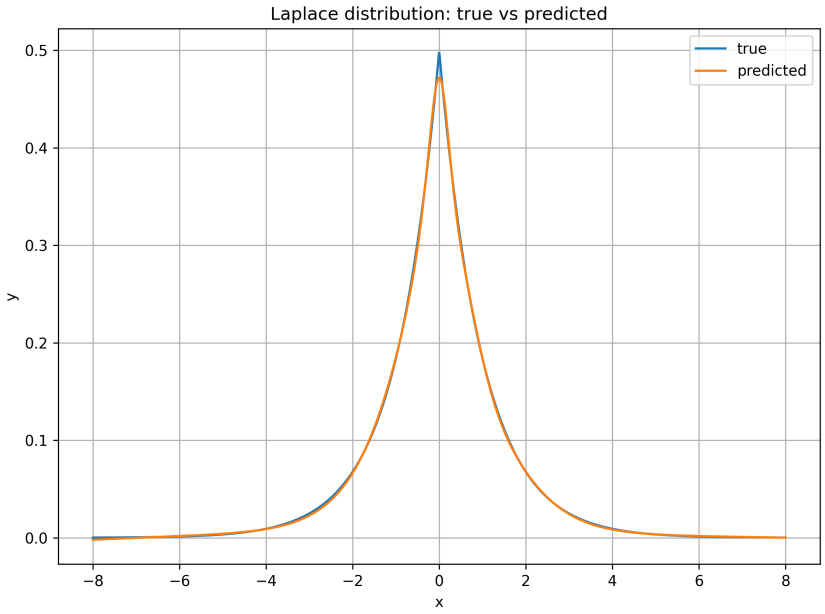


Na podstawie wykresu można zaobserwować kilka charakterystycznych zależności:

- Dla bardzo małej liczby neuronów model osiąga relatywnie wyższy błąd aproksymacji. Wynika to z niewystarczającej pojemności sieci, która nie jest w stanie dokładnie odwzorować analizowanej funkcji.
- W zakresie do około 60 neuronów wyniki pozostają stosunkowo stabilne i utrzymują niski poziom błędu. Oznacza to, że nawet relatywnie niewielka sieć neuronowa jest wystarczająca do aproksymacji funkcji rozkładu Laplace'a.
- Wraz ze wzrostem liczby neuronów rośnie liczba parametrów modelu, co zwiększa wrażliwość procesu uczenia na losową inicjalizację wag oraz stochastyczny charakter mini-batch SGD.
- Dla większych sieci widoczne są coraz silniejsze oscylacje błędu oraz pogorszenie stabilności treningu. Model zaczyna wykazywać oznaki nadmiernej złożoności, co może prowadzić do częściowego przeuczenia oraz trudniejszej optymalizacji przestrzeni parametrów.
- Najbardziej stabilne i powtarzalne wyniki uzyskano dla około 16 neuronów w warstwie ukrytej, dlatego wartość tę wykorzystano w końcowej konfiguracji modelu.

4. Wyniki

4.1 Porównanie funkcji rzeczywistej i aproksymowanej



Poniżej przedstawiono uzyskane wartości błędów na zbiorze testowym:

- **MSE = 0.000005** - średni błąd kwadratowy, silnie penalizujący większe odchylenia predykcji od wartości rzeczywistych
- **MAE = 0.001188** - średni błąd bezwzględny, określający przeciętną różnicę pomiędzy wartością przewidywaną a rzeczywistą

4.2 Wnioski

Na podstawie przeprowadzonych eksperymentów oraz uzyskanych wyników można sformułować następujące wnioski:

- **Wysoka jakość aproksymacji:** Model osiągnął bardzo niski poziom błędu na zbiorze testowym, co świadczy o skutecznym odwzorowaniu funkcji rozkładu Laplace'a.
- **Wpływ hiperparametrów:** Odpowiedni dobór współczynnika uczenia, liczby epok oraz batch size miał istotny wpływ na stabilność i szybkość procesu uczenia.
- **Stabilność mini-batch SGD:** Pomimo widocznych oscylacji funkcji straty proces uczenia zachowywał stabilną zbieżność i prowadził do stopniowej poprawy jakości predykcji.
- **Pojemność modelu:** Przeprowadzona analiza wykazała, że do skutecznej aproksymacji analizowanej funkcji wystarcza relatywnie niewielka liczba neuronów w warstwie ukrytej.
- **Zdolność do generalizacji:** Model poprawnie aproksymował dane testowe niewykorzystywane podczas treningu, co wskazuje na dobrą zdolność generalizacji.