Sprawozdanie NN

Magdalena Jeczeń
nr indeksu: 320558

Spis treści

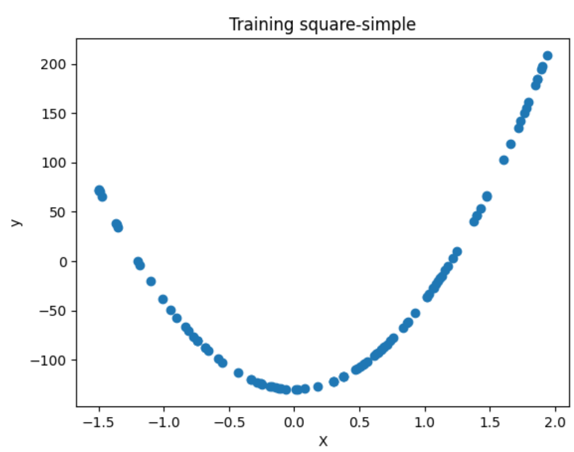
1 Wstęp

Całe laboratorium, składające się z zadań, bazowało na MLP (multilayer perceptron) - modelu perceptronu wielowarstwowego. Sieć ta jest typu feedforward. Głównym założeniem, poza nauką o MLP, było także badanie zachowania się procesu uczenia tej sieci w zależności od ustawień hiperparametrów, czy typu jej uczenia.

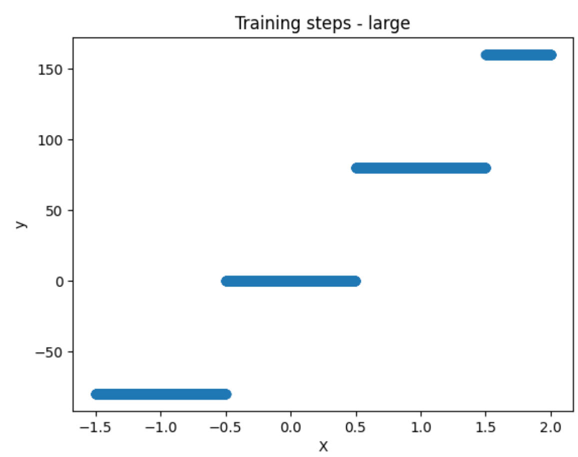
2 NN1: Bazowa implementacja

2.1 Opis zadania

Podstawą zadania było zaimplementowanie sieci neuronowej typu MLP, tak aby możliwe było ręczne ustawienie liczby warstw oraz neuronów na każdej z nich, wag na poszczególnych połączeniach oraz biasów. Wymaganiem zadania również było, aby stosowaną funkcją aktywacji była funkcja sigmoidalna. Funkcją stosowaną na wyjściu (na ostatniej warstwie) miała być funkcja liniowa.
Tak zaimplementowana sieć miała zostać następnie użyta do rozwiązania zadania regresji na zbiorach danych:



(a) Zbiór *square-simple*



(b) Zbiór *steps-large*

dla 3 architektur sieci:

- **Architektura 1:** jedna warstwa ukryta, 5 neuronów
- **Architektura 2:** jedna warstwa ukryta, 10 neuronów
- **Architektura 3:** dwie warstwy ukryte, po 5 neuronów na każdej

Celem zadania było ręczne dobranie wag oraz biasów sieci, tak aby uzyskać wartość MSE na nieznormalizowanym zbiorze testowym, wynoszące co najwyżej 9.

2.2 Moje rozwiązanie

Architektura została zaimplementowana w Pythonie, m.in. przy użyciu biblioteki Numpy. Ważnym jej elementem była normalizacja danych - bez niej nie byłam w stanie osiągać odpowiednio niskich, wymaganych MSE. Dane były normalizowane na wejściu, trenowane, denormalizowane przed liczeniem wartości funkcji kosztu.

2.2.1 Architektura 1:

W przypadku tej architektury, ze względu na tylko jedną warstwę ukrytą, w celu znalezienia odpowiednich wag oraz biasów, stwierdziłam, że zwizualizuję sobię jak wyglądają funkcje sigmoidalne wychodzące z warstwy ukrytej, a dokładniej:

$$\sigma(w_i x_i + b_i) \text{ , gdzie } i \in [5]$$

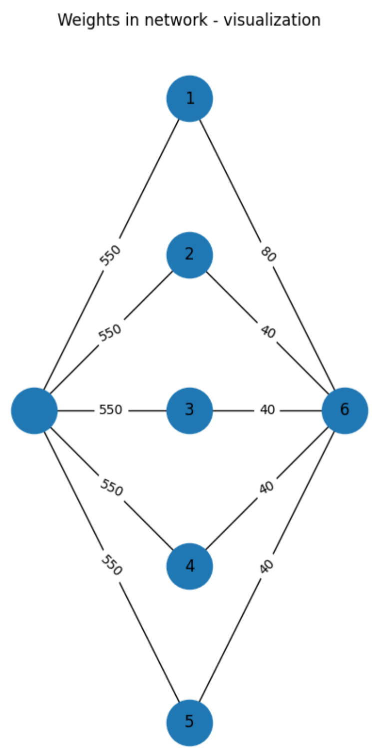
Na ostatniej warstwie stosowałam funkcję liniową $f(x) = x$, co sprawiało, że wartością predykowaną y_{pred} w przypadku regresji była po prostu suma sigmoid wychodzących z warstwy ukrytej. To był kolejny powód mojej wizualizacji - widząc wszystkie sigmoidy na jednym wykresie, byłam w stanie mniej więcej oszacować jak będzie wyglądać ich suma i adekwatnie do tego manipulować wagami oraz biasami.

a) Zbiór danych steps-large

Dane te przypominają schody - w związku z tym wiedziałam, że chcę jak najbardziej 'ostre' sigmoidy, czyli duże wartości wag na połączeniach wchodzących do warstwy ukrytej oraz następnie za pomocą biasów mogłam wykres przesuwać tak aby po zsumowaniu otrzymać kolejne stopnie.
Następnie za pomocą wag na połączeniach między warstwą ukrytą a wychodzącą, oraz biasu na warstwie wychodzącej odpowiednio przeskalować oraz przesunąć ostateczny wykres.

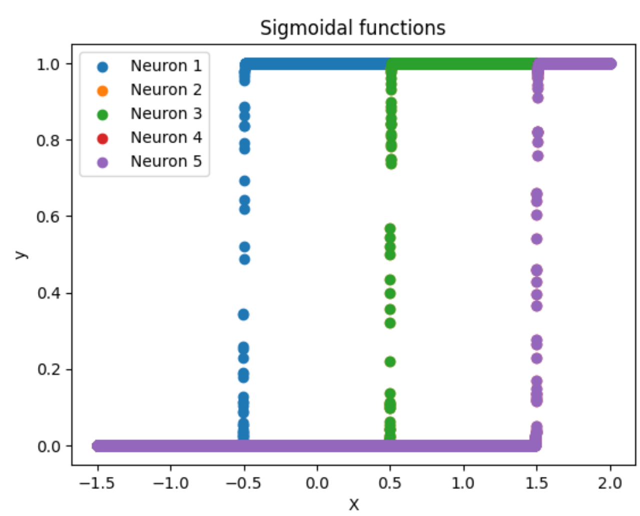
Finalnie dobrane wagi oraz biasy:

- wagi:



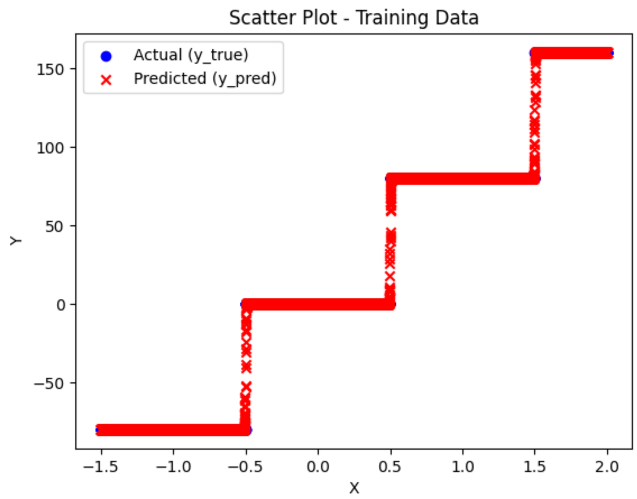
Rysunek 2: Wizualizacja ostatecznych wag w sieci dla architektury 1 i zbioru *steps-large*, (liczby na neuronach oznaczają numer neuronu, które będą użyte przy podaniu wartości biasów na tych neuronach)

- biasy:
  $b_1 = 275$, $b_2 = -275$, $b_3 - 275$, $b_4 = -825$, $b_5 = -825$, $b_6 = -80$



Rysunek 3: Sigmoidy wychodzące z warstwy ukrytej dla ostatecznie wybranych wag oraz biasów dla zbioru *steps-large*

Finalne wyniki:
Błąd średniokwadratowy (MSE) dla zbioru treningowego: 3.796509159113264
Błąd średniokwadratowy (MSE) dla zbioru testowego: 4.307871616653174



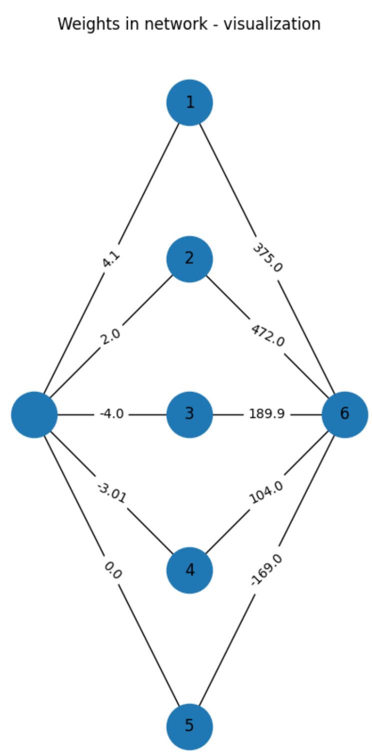
Rysunek 4: Wykres danych treningowych po wytrenowaniu wraz z prawdziwymi danymi przewidywanymi, dla zbioru *steps-large*, architektura 1

b) Zbiór danych square-simple

Dane te przedstawiają funkcję kwadratową - w związku z tym podczas dobierania wag na wejściu do warstwy ukrytej stosowałam małe wartości, aby uzyskane w ten sposób sigmoidy były bardziej zaokrąglone.

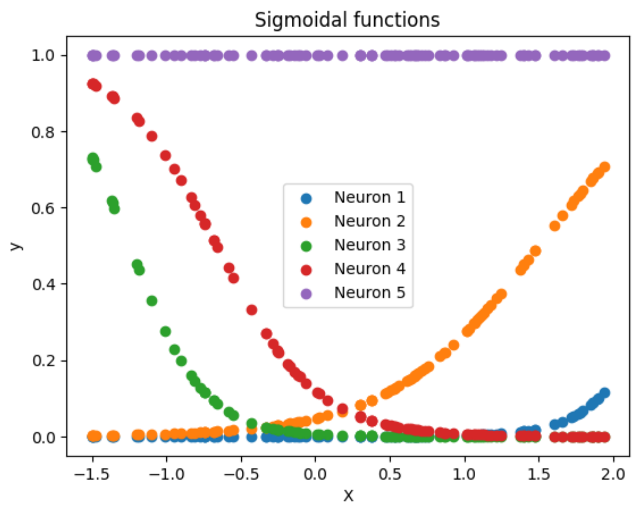
Finalnie dobrane wagi oraz biasy:

- wagi:



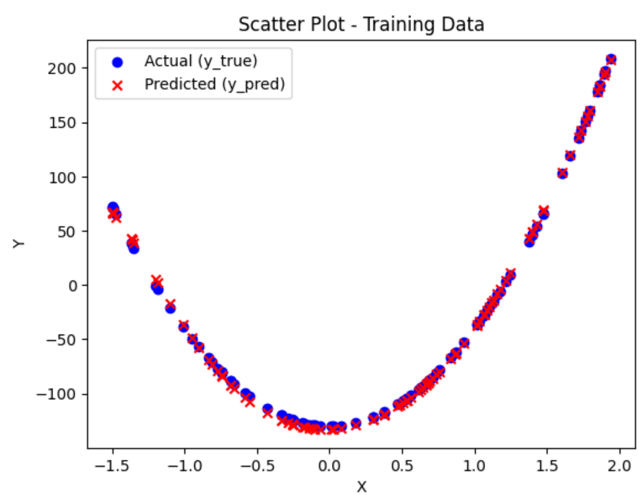
Rysunek 5: Wizualizacja ostatecznych wag w sieci dla architektury 1 i zbioru *steps-large*, (liczby na neuronach oznaczają numer neuronu, które będą użyte przy podaniu wartości biasów na tych neuronach)

- biasy:
  $b_1 = -10$, $b_2 = -3$, $b_3 - 5$, $b_4 = -2$, $b_5 = 7$, $b_6 = 0$



Rysunek 6: Sigmoidy wychodzące z warstwy ukrytej dla ostatecznie wybranych wag oraz biasów dla zbioru *square-simple*

Finalne wyniki:
Błąd średniokwadratowy (MSE) dla zbioru treningowego: 7.107663231792449
Błąd średniokwadratowy (MSE) dla zbioru testowego: 7.886637396889147



Rysunek 7: Wykres danych treningowych po wytrenowaniu wraz z prawdziwymi danymi przewidywanymi, dla zbioru *square-simple*, architektura 1

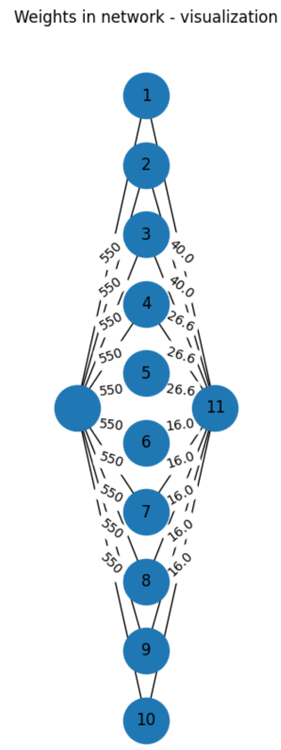
2.2.2 Architektura 2:

W przypadku tej architektury przyjęłam taką samą taktykę jak w przypadku poprzedniej - wizualizowałam sobie sigmoidy na wyjściu z warstwy ukrytej, tak aby odpowiednio dobrać wagi oraz biasy.

a) Zbiór danych steps-large

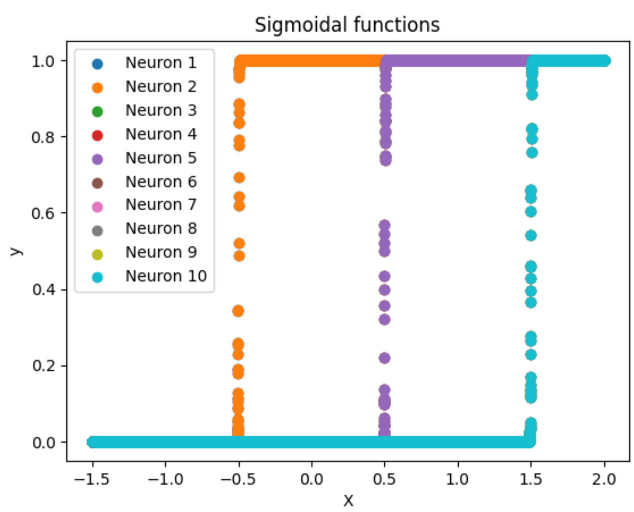
Finalnie dobrane wagi oraz biasy:

- wagi:



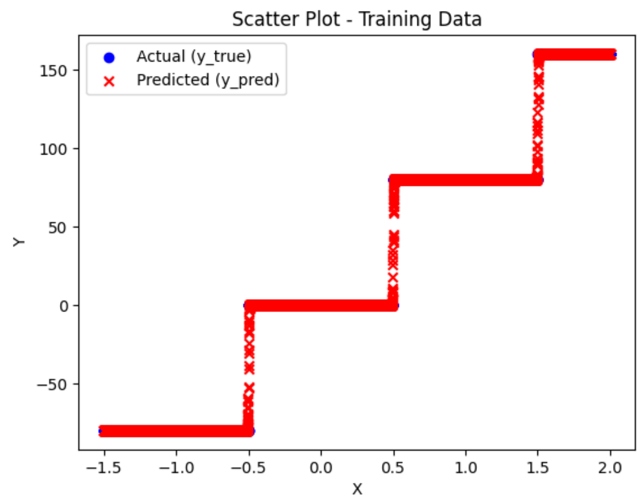
Rysunek 8: Wizualizacja ostatecznych wag w sieci dla architektury 2 i zbioru *steps-large*, (liczby na neuronach oznaczają numer neuronu, które będą użyte przy podaniu wartości biasów na tych neuronach)

- biasy:
  $b_1 = 275$, $b_2 = 275$, $b_3 = -275$, $b_4 = -275$, $b_5 = -275$, $b_6 = -825$, $b_7 = -825$, $b_8 = -825$, $b_9 = -825$, $b_{10} = -825$, $b_{11} = -80$



Rysunek 9: Sigmoidy wychodzące z warstwy ukrytej dla ostatecznie wybranych wag oraz biasów dla zbioru *steps-large*

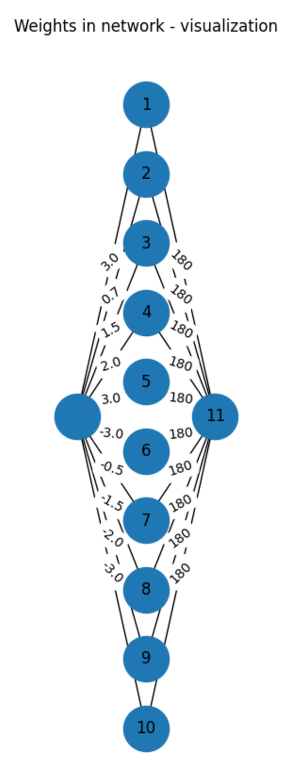
Finalne wyniki:
Błąd średniokwadratowy (MSE) dla zbioru treningowego: 3.81964831147679
Błąd średniokwadratowy (MSE) dla zbioru testowego: 4.34578385689489



Rysunek 10: Wykres danych treningowych po wytrenowaniu wraz z prawdziwymi danymi przewidywanymi, dla zbioru *steps-large*, architektura 2

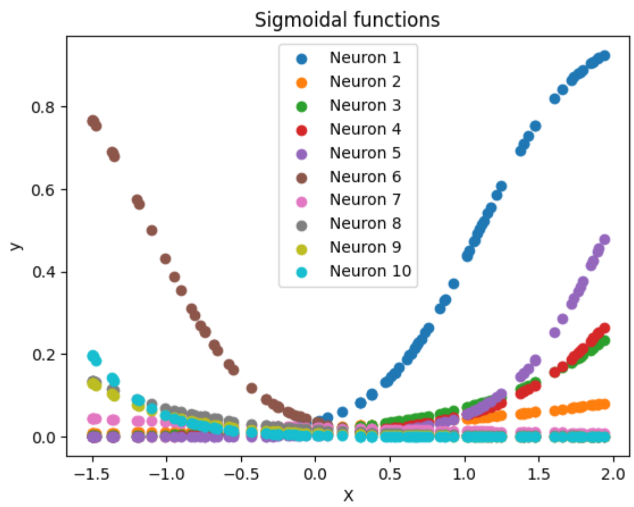
b) Zbiór danych square-simple
Dane te przedstawiają funkcję kwadratową - w związku z tym podczas dobierania wag na wejściu do warstwy ukrytej stosowałam małe wartości, aby uzyskane w ten sposób sigmoidy były bardziej zaokrąglone.
Finalnie dobrane wagi oraz biasy:

- wagi:



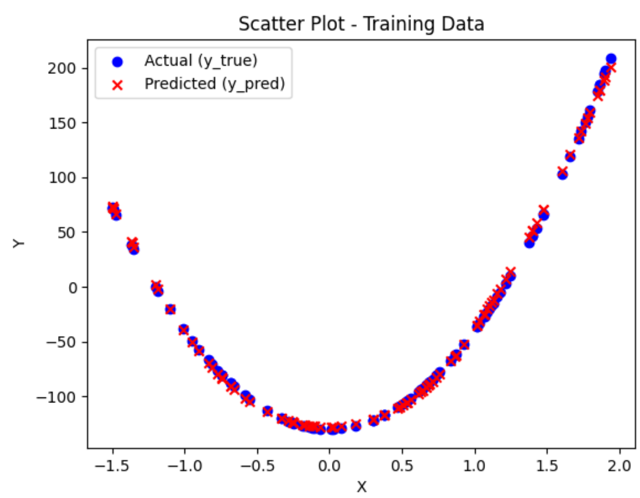
Rysunek 11: Wizualizacja ostatecznych wag w sieci dla architektury 2 i zbioru *square-simple*, (liczby na neuronach oznaczają numer neuronu, które będą użyte przy podaniu wartości biasów na tych neuronach)

- biasy:
  $b_1 = -3.3$, $b_2 = -3.8$, $b_3 = -4.1$, $b_4 = -4.9$, $b_5 = -5.9$, $b_6 = -3.3$, $b_7 = -3.8$, $b_8 = -4.1$, $b_9 = -4.9$, $b_{10} = -5.9$, $b_{11} = -158$



Rysunek 12: Sigmoidy wychodzące z warstwy ukrytej dla ostatecznie wybranych wag oraz biasów dla zbioru *square-simple*

Finalne wyniki:
Błąd średniokwadratowy (MSE) dla zbioru treningowego: 6.800043061688103
Błąd średniokwadratowy (MSE) dla zbioru testowego: 7.336642941501717



Rysunek 13: Wykres danych treningowych po wytrenowaniu wraz z prawdziwymi danymi przewidywanymi, dla zbioru *square-simple*, architektura 2

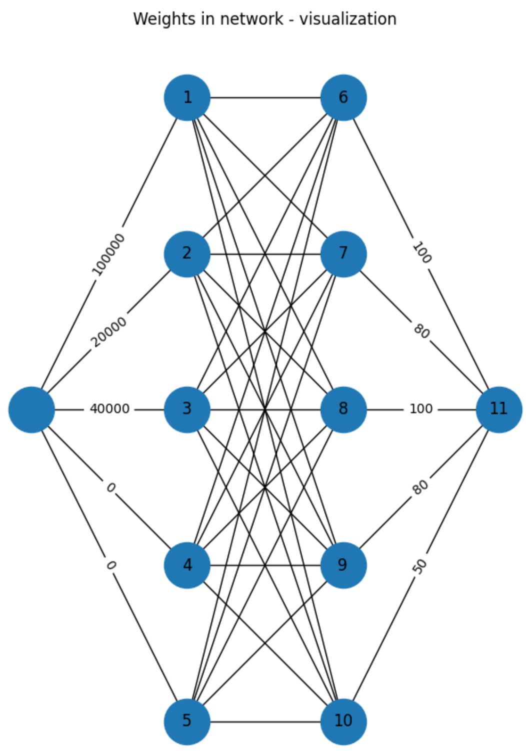
2.2.3 Architektura 3:

Dla tej architektury, która posiada już 2 warstwy ukryte, zrezygnowałam z rysowania sigmoid, ponieważ wyczułam jak one oraz ich suma się zachowują. Następnie metodą prób i błędów dobierałam wagi i biasy tak by uzyskać wymagane MSE.

a) Zbiór danych steps-large

Finalnie dobrane wagi oraz biasy:

- wagi:



Rysunek 14: Wizualizacja ostatecznych wag w sieci dla architektury 3 i zbioru *steps-large* między warstwą wejściową i pierwszą ukrytą oraz między drugą ukrytą a wyjściową. Liczby na neuronach oznaczają numer neuronu, które będą użyte przy podaniu wartości biasów na tych neuronach oraz pozostałych wag.

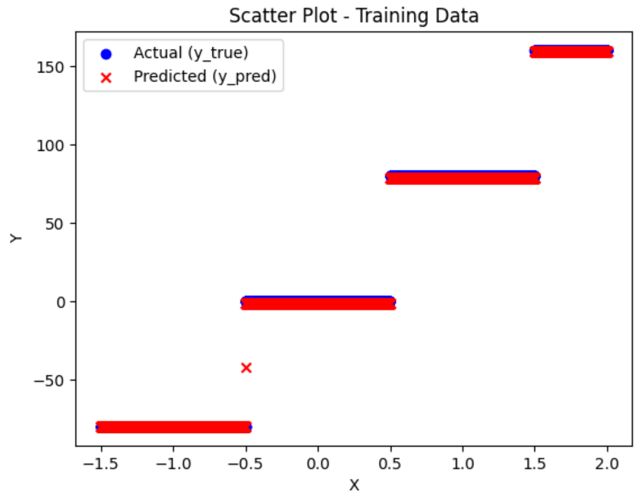
Wartości wag na połączeniach między warstwami ukrytymi, przedstawione w macierzy, gdzie W_{ij}, $i, j \in [5]$ to waga na połączeniu między neuronem i-tym a $(j+5)$-tym:

$$W = \begin{pmatrix} 50 & 0 & 0 & 0 & 0 \\ 0 & 40 & 0 & 0 & 0 \\ 1.4 & 0 & 50 & 0 & 0 \\ 0 & 0 & 0 & 50 & 0 \\ 0 & 0 & 0 & 0 & 50 \end{pmatrix}$$

- biasy:
  $b_1 = -150000$, $b_2 = -10000$, $b_3 = 20000$, $b_4 = 0$, $b_5 = 0$, $b_6 = -2.81$, $b_7 = -30$, $b_8 = -0.6$, $b_9 = 0$, $b_{10} = 0$, $b_{11} = -251$

Finalne wyniki:

Błąd średniokwadratowy (MSE) dla zbioru treningowego: 2.036506738889041
Błąd średniokwadratowy (MSE) dla zbioru testowego: 1.2300802548172247

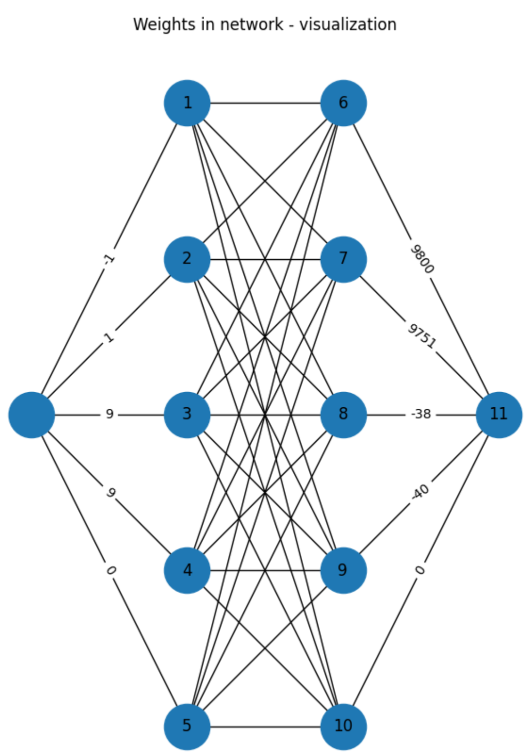


Rysunek 15: Wykres danych treningowych po wytrenowaniu wraz z prawdziwymi danymi przewidywanymi, dla zbioru *steps-large*, architektura 3

b) Zbiór danych square-simple

Finalnie dobrane wagi oraz biasy:

- wagi:



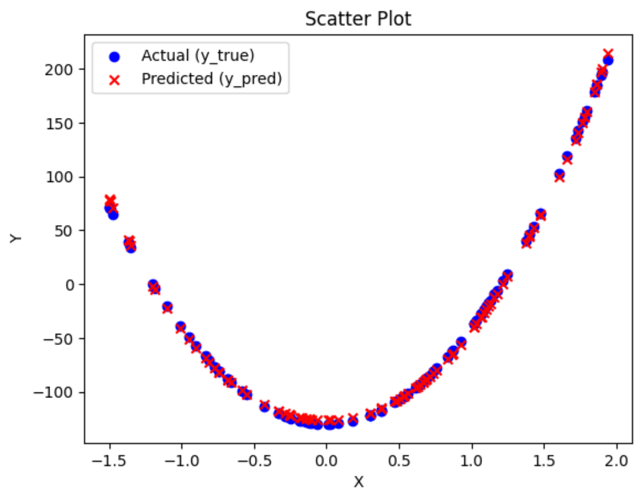
Rysunek 16: Wizualizacja ostatecznych wag w sieci dla architektury 3 i zbioru *square-simple* między warstwą wejściową i pierwszą ukrytą oraz między drugą ukrytą a wyjściową. Liczby na neuronach oznaczają numer neuronu, które będą użyte przy podaniu wartości biasów na tych neuronach oraz pozostałych wag

Wartości wag na połączeniach między warstwami ukrytymi, przedstawione w macierzy, gdzie W_ij, $i, j \in [5]$ to waga na połączeniu między neuronem i-tym a $(j+5)$-tym:

$$W = \begin{pmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 50 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

- biasy:
  $b_1 = -4.8$, $b_2 = -4.8$, $b_3 = -15$, $b_4 = -20$, $b_5 = 0$, $b_6 = 0$, $b_7 = 0$, $b_8 = 0$, $b_9 = 0$, $b_{10} = 0$, $b_{11} = -10022.5$

Finalne wyniki:
Błąd średniokwadratowy (MSE) dla zbioru treningowego: 7.464028687068637
Błąd średniokwadratowy (MSE) dla zbioru testowego: 6.701690034545383



Rysunek 17: Wykres danych treningowych po wytrenowaniu wraz z prawdziwymi danymi przewidywanymi, dla zbioru *square-simple*, architektura 3

2.3 Podsumowanie, wnioski

Dobieranie wag metodą prób i błedów było dosyć czasochłonne, szczególnie na początku. W przypadku architektur z 1 warstwą ukrytą bardzo pomocne była wizualizacja sigmoid wychodzących z warstwy ukrytej. Pomogło to również w nabraniu intuicji, jak zachowuje się funkcja sigmoidalna przy skalowaniu i przesuwaniu. Zbiorem, dla którego dobieranie wag i biasów było prostsze był zbiór *steps-large*. Ze względu na to, że przedstawiał schody, łatwo było zobaczyć jak chcę aby każda poszczególna sigmoida na wyjściu z warstwy ukrytej wyglądała. Ważnym aspektem całego rozwiązania zadania była również normalizacja danych. Dzięki temu wagi oraz biasy mogły być dobrane przeze mnie łatwiej i dokładniej, co pomogło osiągnąć wymagane MSE.

3 NN2: Implementacja propagacji wstecznej błędu

3.1 Opis zadania

Zadanie polegało na zaimplementowaniu uczenia sieci neuronowej propagacją wsteczną błędu:

- z aktualizacją parametrów po przejsciu przez wszystkie obserwacje zbioru treningowego
- z podziałem zbioru treningowego na batche i aktualizacją parametrów po przejściu przez batch

Celem było porównanie tych 2 metod pod względem szybkości uczenia, a także wpływu wielkości batchy na szybkość uczenia.
Mieliśmy także przetestować uczenie naszej sieci na zbiorach:

- *square-simple*, wymagane maksymalne MSE: 4
- *steps-small*, wymagane maksymalne MSE: 4
- *multimodal-large*, wymagane maksymalne MSE: 40

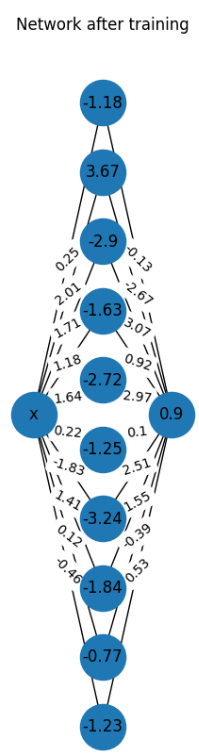
3.2 Moje rozwiązanie

Sieć z zadania NN1 powiększyła się o implementację propagacji wstecznej. Wagi początkowe były inicjalizowane z rozkładu jednostajnego na przedziale $[0, 1]$, natomiast początkowe biasy były równe 0.

3.2.1 Test uczenia sieci

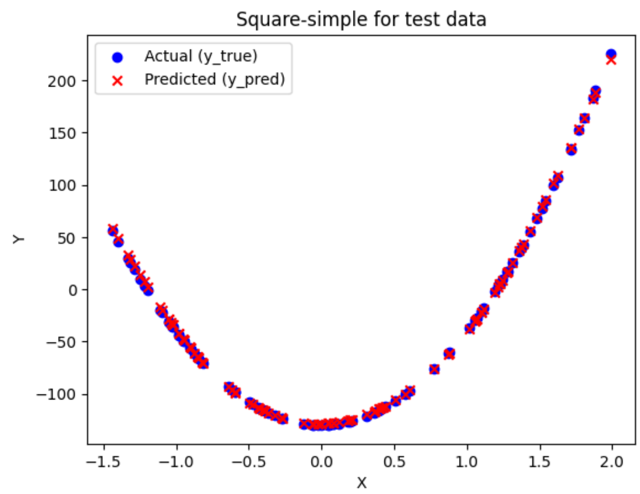
a) Zbiór danych square-simple
Użyta przeze mnie architektura sieci do tego zbioru miała jedną warstwę ukrytą, na której znajdowało się 10 neuronów.

Wagi i biasy po wytrenowaniu modelu:



Rysunek 18: Wagi oraz biasy po wytrenowaniu sieci dla zbioru *square-simple*

Wyniki:
Square-simple MSE dla train: 1.3068611743996954
Square-simple MSE dla test: 2.0884659423622183



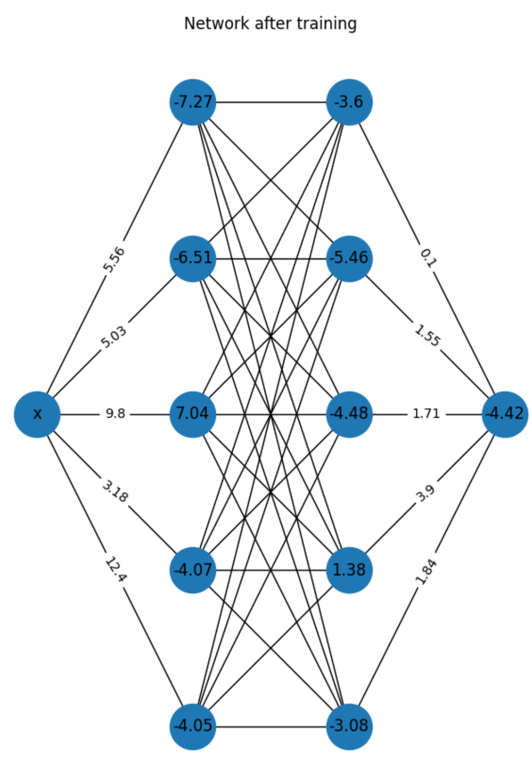
Rysunek 19: Wykres danych testowych przewidzianych przez wytrenowany model wraz z prawdziwymi danymi dla zbioru *square-simple*

Ten zbiór danych nie potrzebował wielu epok (użyłam 400), aby się wytrenować, ze względu na swoją prostotę.

b) Zbiór danych steps-small

Użyta przeze mnie architektura sieci do tego zbioru miała dwie warstwy ukryte, na których znajdowało się po 5 neuronów. Zdecydowałam się na architekturę z 2 warstwami ukrytymi, ponieważ pod względem propagacji wstecznej te dane nie są tak proste jak poprzednie (są bardzo 'ostre'). Próbowałam trenować na nich sieć z jedną warstwą ukrytą, jednak nie przyniosło to wymaganego, bądź blisko wymaganego MSE.

Wagi i biasy po wytrenowaniu modelu:

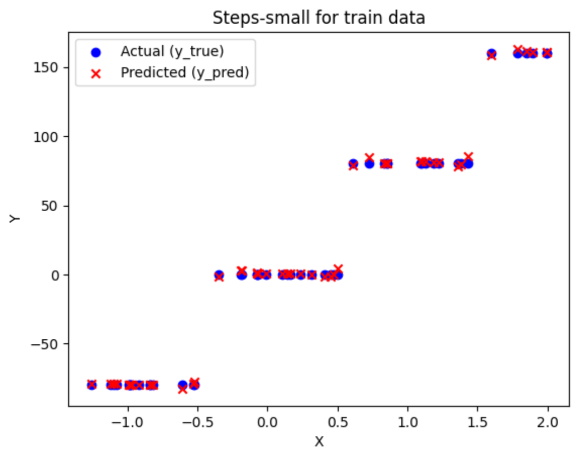


Rysunek 20: Wagi oraz biasy po wytrenowaniu sieci dla zbioru *steps-small*

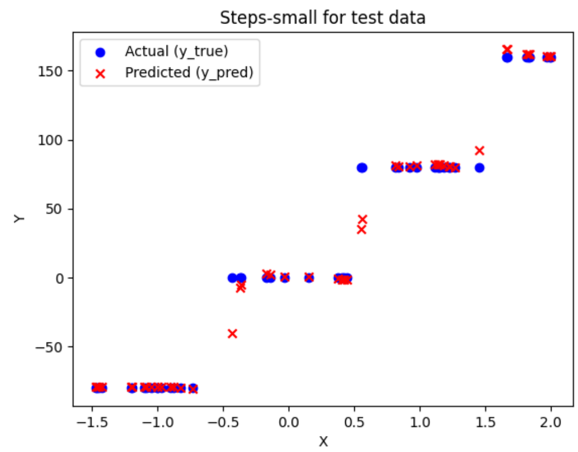
Wyniki:

Steps-small MSE dla train: 2.9708229519945344
Steps-small MSE dla test: 109.20024636254101

Niestety nie udało mi się wytrenować tego zbioru tak, aby MSE na zbiorze testowym było chociaż zbliżone do tego na treningowym. Myślę, że wynika to z liczby obserwacji w zbiorze treninigowym - jest ich zaledwie 50 i nie wystarcza to, aby wytrenowana sieć dobrze działała na danych testowych.



(a) Wykres danych treningowych przewidzianych przez wytrenowany model wraz z prawdziwymi danymi dla zbioru *steps-small*



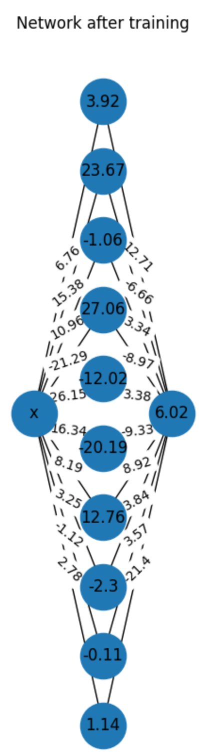
(b) Wykres danych testowych przewidzianych przez wytrenowany model wraz z prawdziwymi danymi dla zbioru *steps-small*

Widzimy, że sieć ma problem z dobrą predykcją na zbiorze testowym przede wszystkim na krańcach schodów.

c) Zbiór danych multimodal-large

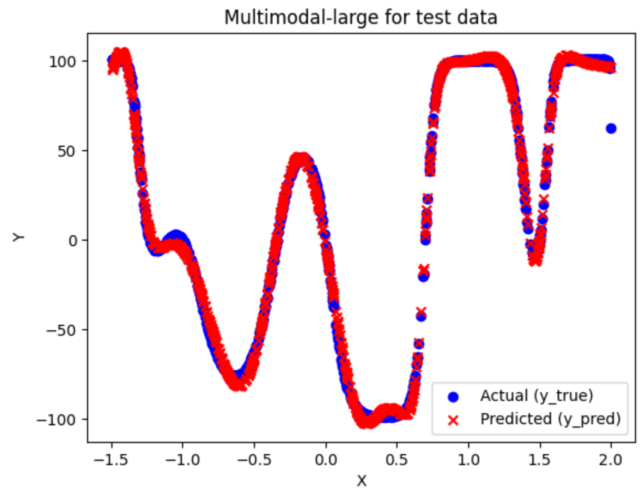
Dla tego zbioru danych użyłam sieci z jedną warstwą ukrytą, z 10 neuronami na niej.

Wagi po wytrenowaniu modelu:



Rysunek 22: Wagi oraz biasy po wytrenowaniu sieci dla zbioru *multimodal-large*

Wyniki:
Multimodal-large MSE dla train: 17.11008893640455
Multimodal-large MSE dla test: 12.853568732810311

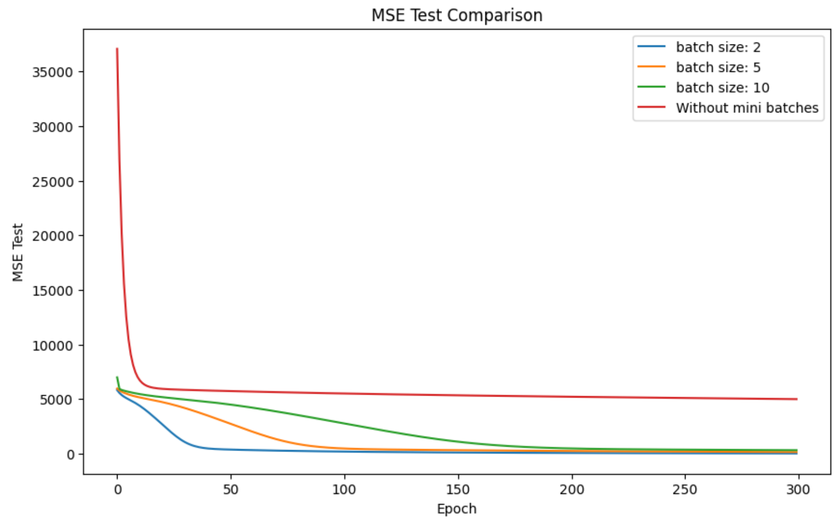


Rysunek 23: Wykres danych testowych przewidzianych przez wytrenowany model wraz z prawdziwymi danymi dla zbioru *multimodal-large*

Sieć dobrze poradziła sobie z tymi danymi, mimo nieregularności kształtu tej funkcji. Jednak potrzebowała sporo czasu na wytrenowanie, co może wynikać z dużej liczby obserwacji w zbiorze treningowym (jest ich 10000).

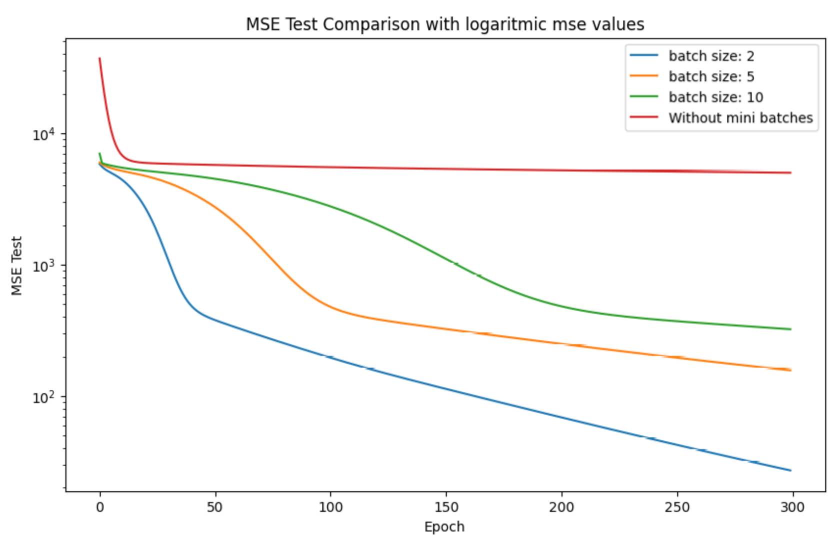
3.2.2 MSE względem liczby epok, w zależności od rozmiaru batchy

Ten eksperyment przeprowadziłam dla danych *square-simple* oraz dla sieci z jedną warstwą ukrytą, z 10 neuronami na niej.



Rysunek 24: Wartość MSE dla wytrenowanego modelu dla danych testowych zbioru *square-simple* w zależności od liczby batchy oraz od uczenia bez-batchowego

Aby lepiej zobaczyć różnice między uczeniem, szczególnie w końcowych epokach, posłużyłam się skalą logarytmiczną na osi y (z MSE).



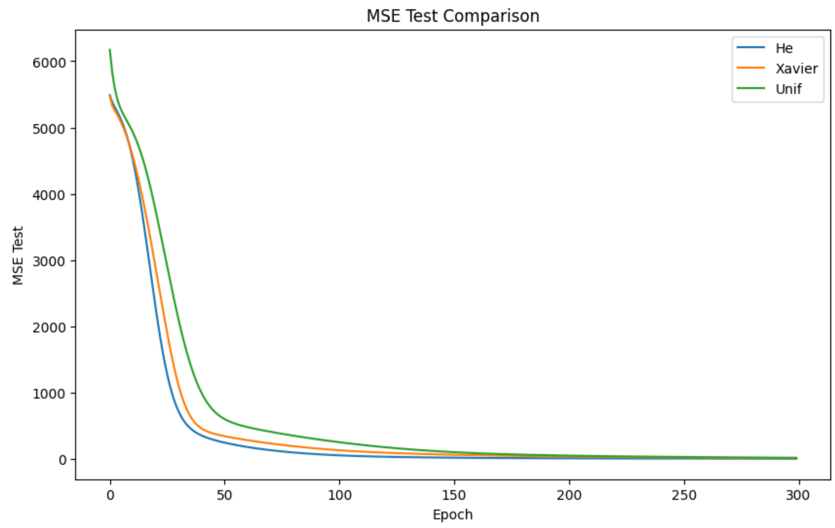
Rysunek 25: Wartość MSE dla wytrenowanego modelu dla danych testowych zbioru *square-simple* w zależności od liczby batchy oraz od uczenia bez-batchowego, ze zlogarytmowanymi wartościami MSE

Widzimy, że występuje tendencja - im więcej obserwacji w batchu, tym gorzej sieć radzi sobie z uczeniem. Gdy nie korzystamy w ogóle z mini-batchy, funkcja radzi sobie najgorzej. Myślę, że jest to spowodowane tym, że sieć podczas propagacji wstecznej robi wówczas 'krok' po przejściu przez wszystkie obserwacje. Krok ten jest również uśredniony po wszystkich obserwacjach, co przy wzięciu wszystkich obserwacji ze zbioru treningowego wpływa negatywnie na postęp uczenia w tym przypadku.

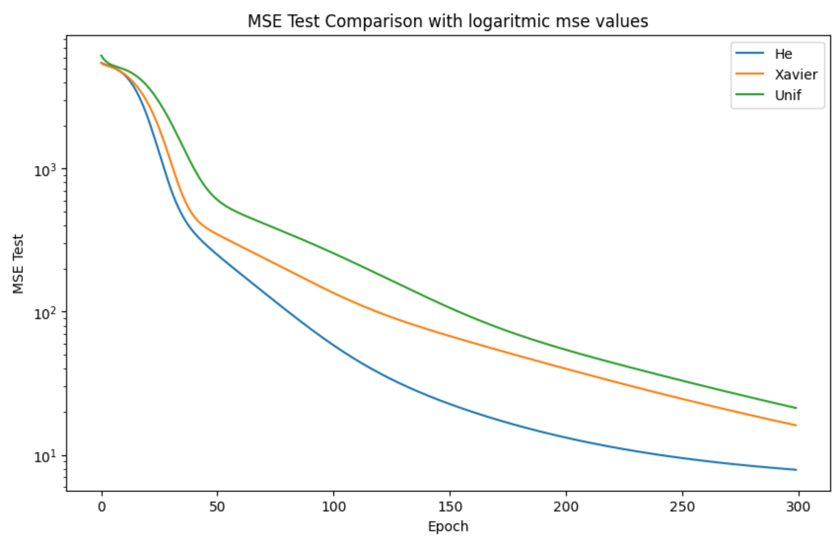
3.2.3 MSE względem liczby epok, w zależności od metody inicjalizacji wag początkowych

Sprawdziłam również, jak radzi sobie sieć, w zależności od metody inicjalizacji wag początkowych - z rozkładu jednostajnego $[0, 1]$, metoda He oraz metoda Xavier.
Dane użyte przeze mnie do tego eksperymentu to *Square-simple*, a sieć składała się z 1 warstwy ukrytej z 10 neuronami.

Wyniki prezentują się następująco.



Rysunek 26: MSE na zbiorze testowym względem liczby epok, w zależności od metody inicjalizacji wag początkowych dla zbioru *square-simple*



Rysunek 27: MSE z wartościami zlogarytmowanymi na zbiorze testowym względem liczby epok, w zależności od metody inicjalizacji wag początkowych dla zbioru *square-simple*

Widzimy, że sieć najlepiej radzi sobie, gdy wagi początkowe inicjalizowane są metodą He. Najgorzej radzą sobie wagi inicjalizowane z rozkładu jednostajnego $[0, 1]$. Jednak sa to niewielkie różnice i dla każdej z tych metod sieć radzi sobie podobnie.

3.2.4 Podsumowanie, wnioski

Sama implementacja propagacji wstecznej była czasochłonna, ponieważ jest to mimo wszystko rozbudowana matematycznie metoda.
Sieć radziła sobie dobrze z danymi, jedynie zbiór *steps-small* sprawił problem, co mogło wynikać z małej liczności zbioru treningowego.
Gdy porównałam uczenie batchowe z bez-batchowym, uczenie batchowe dało lepsze rezultaty. W dodatku, im mniej było obserwacji w batchu, tym lepiej sieć sobie radziła. Wynika to moim zdaniem z tego, iż gdy jest mniej obserwacji w batchu, sieć dopasowuje się bardziej lokalnie niż globalnie. W tym przypadku również częsciej robione są kroki w stronę uśrednionego gradientu, co zmniejsza dokładność uczenia, jednak znacznie go przyspiesza.

4 NN3: Implementacja momentu i normalizacji gradientu

4.1 Opis zadania

Celem zadania było zaimplementowanie 2 metod usprawnienia uczenia gradientowego:

- moment
- RMSProp

Następnie porównanie ich pod względem szybkości zbieżności procesu uczenia.
Mieliśmy także przetestować uczenie naszej sieci na zbiorach:

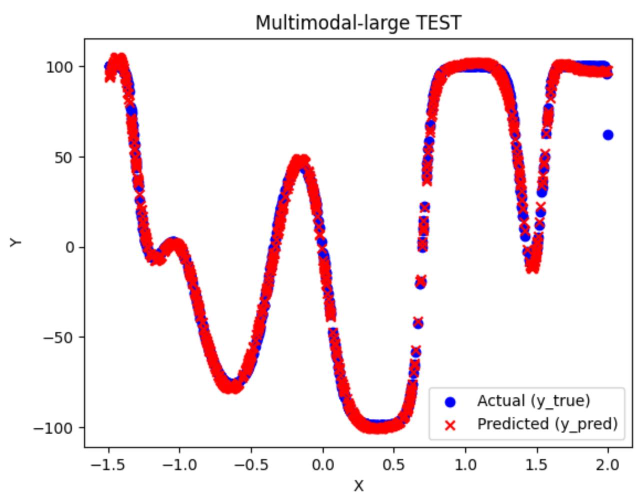
- *multimodal-large*, wymagane maksymalne MSE: 9
- *steps-large*, wymagane maksymalne MSE: 3
- *square-large*, wymagane maksymalne MSE: 1

4.2 Moje rozwiązanie

4.2.1 Test uczenia sieci

a) Zbiór danych multimodal-large
Dla tych danych użyłam sieci z jedną warstwą ukrytą, na której znajdowało się 10 neuronów. W celu osiągnięcia wymaganego MSE użyłam metody momentów.

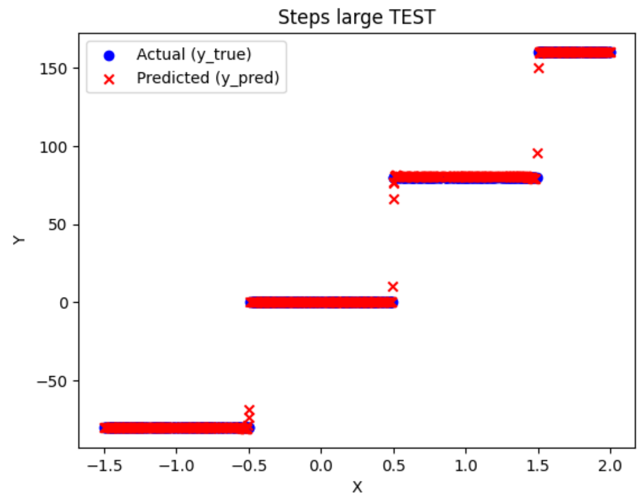
Wyniki:
Zbiór multimodal-large, MSE dla zbioru treningowego: 11.076350693688426
Zbiór multimodal-large, MSE dla zbioru testowego: 6.770157194289296



Rysunek 28: Wykres danych testowych przewidzianych przez wytrenowany model wraz z prawdziwymi danymi dla zbioru *multimodal-large*

b) Zbiór danych steps-large
Dla tych danych użyłam sieci z 3 warstwami ukrtymi, po 5 neuronów na każdej z nich. W celu osiągnięcia wymaganego MSE użyłam metody momentów.

Wyniki:
Zbiór steps-large, MSE dla zbioru treningowego: 5.650248541684378
Zbiór steps-large, MSE dla zbioru testowego: 0.9427801717397638

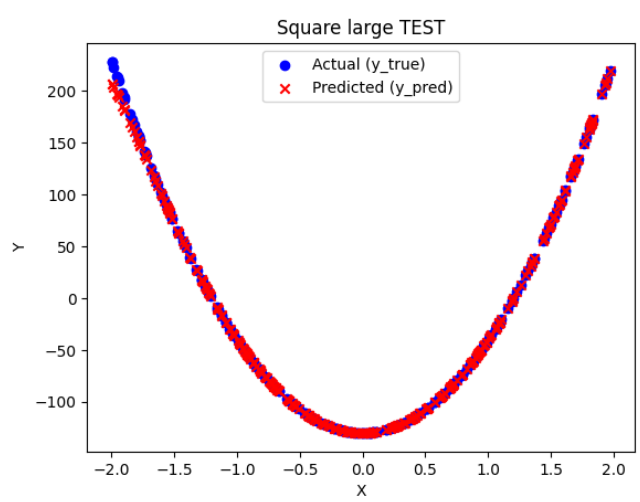


Rysunek 29: Wykres danych testowych przewidzianych przez wytrenowany model wraz z prawdziwymi danymi dla zbioru *steps-large*

c) Zbiór danych square-large
Dla tych danych użyłam sieci z dwiema warstwami ukrytymi, po 5 neuronów na każdej. W celu osiągnięcia wymaganego MSE użyłam metody momentów.

W przypadku tego zbioru danych, nie udało mi się ich wytrenować tak, aby uzyskać wymagane MSE albo chociaż zbliżone MSE między zbiorem testowym i treningowym.

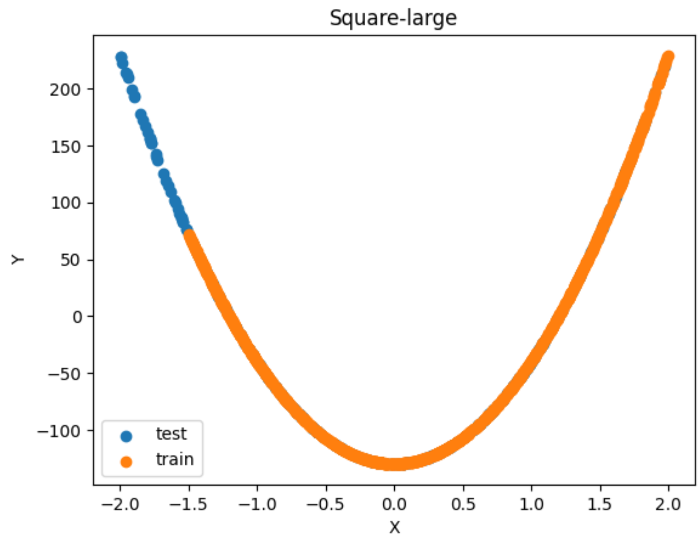
Wyniki:
Zbiór square-large najmniejsze uzyskane MSE test: 9.8551119177237
Zbiór square-large MSE train: 0.0003560064322213099



Rysunek 30: Wykres danych testowych przewidzianych przez wytrenowany model wraz z prawdziwymi danymi dla zbioru *square-large*

Widać, że sieć miała problem z lewą częścią wykresu oraz jest duża różnica między MSE na zbiorze treningowym i testowym.

Przyjrzymy się danym:



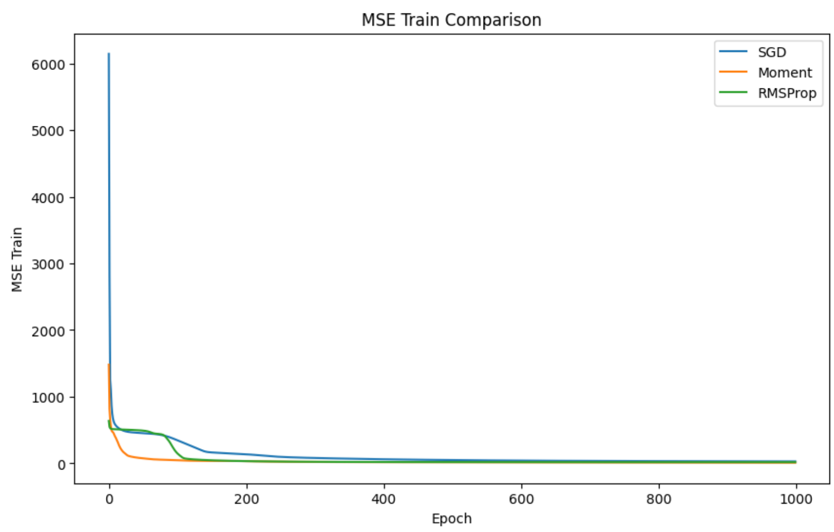
Rysunek 31: Dane testowe oraz treningowe dla zbioru *square-large*

Powyższy wykres przedstawia jak dane treningowe i testowe mają się do siebie. Widzimy, że z lewej strony wykresu, występują znaczące braki w zbiorze treningowym. W związku z tym sieć miała prawo mieć problemy z nauczeniem się tej lewej części wykresu, gdyż nie miała tam nawet jednego punktu zaczepienia. Mimo puszczenia sieci na bardzo dużo epok, nie była w stanie zejść aż tak nisko z MSE na zbiorze testowym. Być może puszczenie jej na jeszcze więcej epok poskutkowałoby dojściem do wymaganego MSE wynoszącego 1, jednak trwałoby to długo.

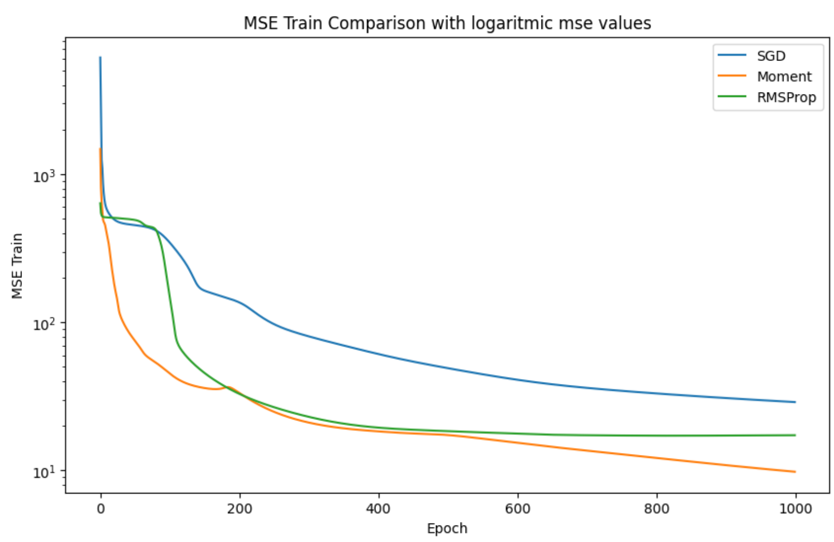
4.2.2 MSE względem liczby epok, w zależności od rodzaju uczenia gradientowego

Eksperyment ten wykonałam dla danych *steps-large* oraz *multimodal-large*.

a) Zbiór danych steps-large



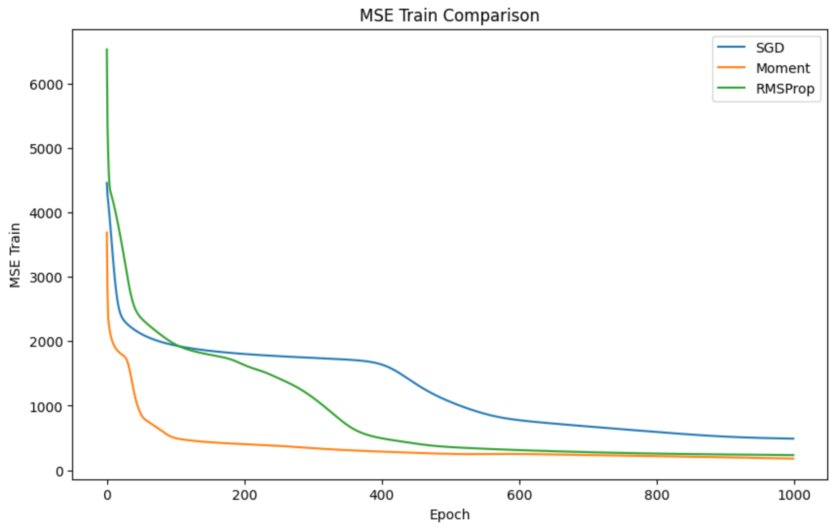
Rysunek 32: MSE na zbiorze testowym względem liczby epok, w zależności od rodzaju uczenia gradientowego dla zbioru *steps-large*



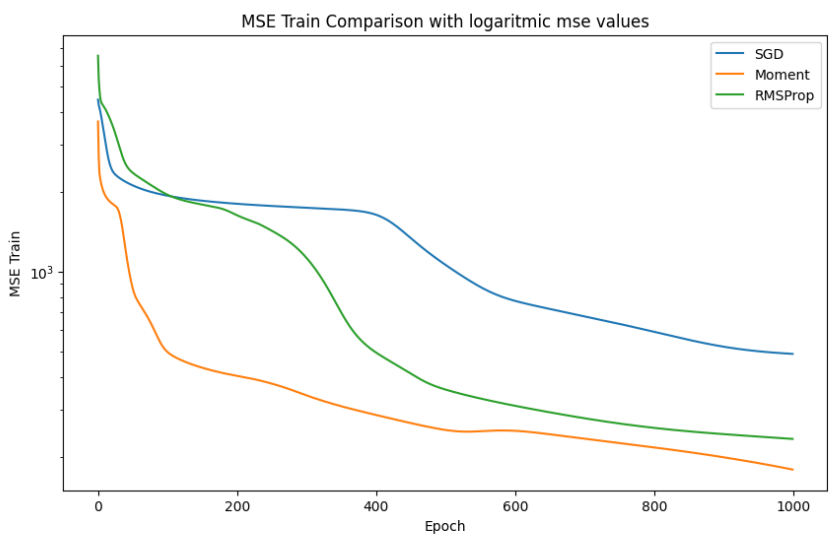
Rysunek 33: MSE z wartościami zlogarytmowanymi na zbiorze testowym względem liczby epok, w zależności od rodzaju uczenia gradientowego dla zbioru *steps-large*

Widzimy, że w przypadku danych *steps-large* najlepiej radzą sobie metody RMSProp oraz Momentu. Na początku radzą sobie podobnie, jednak RMSProp od pewnego momentu bardzo zwolnił, a Moment znalazł jeszcze mniejsze minimum.

b) Zbiór danych multimodal-large



Rysunek 34: MSE na zbiorze testowym względem liczby epok, w zależności od rodzaju uczenia gradientowego dla zbioru *multimodal-large*



Rysunek 35: MSE z wartościami zlogarytmowanymi na zbiorze testowym względem liczby epok, w zależności od rodzaju uczenia gradientowego dla zbioru *multimodal-large*

W przypadku tych danych również najlepiej poradziła sobie metoda Momentu. RMSProp poradził sobie niewiele gorzej. Natomiast uczenie gradientowe bez usprawnienia dało dużo gorsze wyniki.

4.2.3 Podsumowanie, wnioski

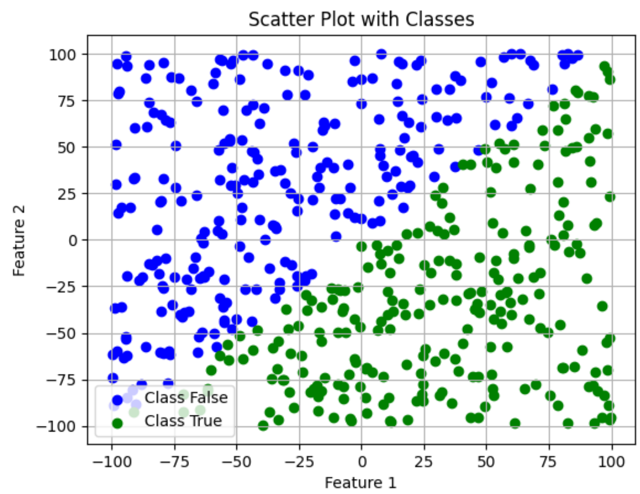
Obydwe metody usprawnień uczenia gradientowego - Moment oraz RMSProp nie były trudne do zaimplementowania. Za to znacznie poprawiły uczenie sieci. Testując model na danych *square-large* natrafiłam na problem z ich wytrenowaniem, ze względu na braki w obserwacjach w danych treningowych względem testowych. Sieć nie była w stanie w sensowym czasie wytrenować się na tyle, bo osiągnąć zbliżone MSE do danych treningowych.

5 NN4: Rozwiązywanie zadania klasyfikacji

5.1 Opis zadania

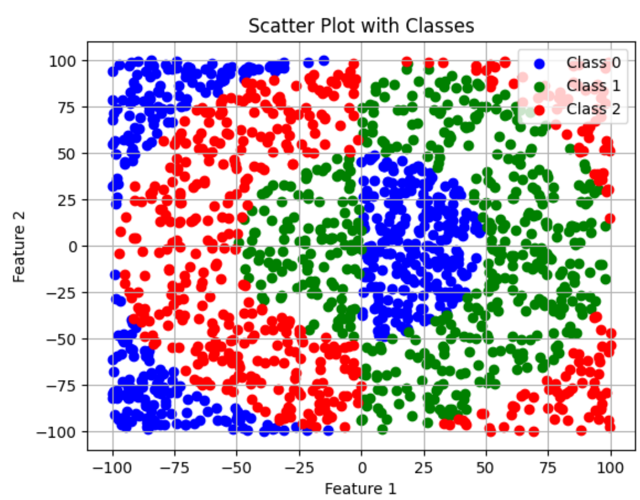
Celem zadania była implementacja funkcji softmax dla warstwy wyjściowej sieci neuronowej. Mieliśmy również sprawdzić jak sieć radzi sobie z uczeniem, gdy używamy na ostatniej warstwie funkcji softmax, a jak gdy używamy funkcji sigmoidalnej. Funkcją kosztu miało być F-measure, natomiast po ostatniej warstwie miało być zastosowane cross-entropy.
Mieliśmy także przetestować uczenie naszej sieci na zbiorach:

- *easy*, wymagane minimalne F-measure: 0.99



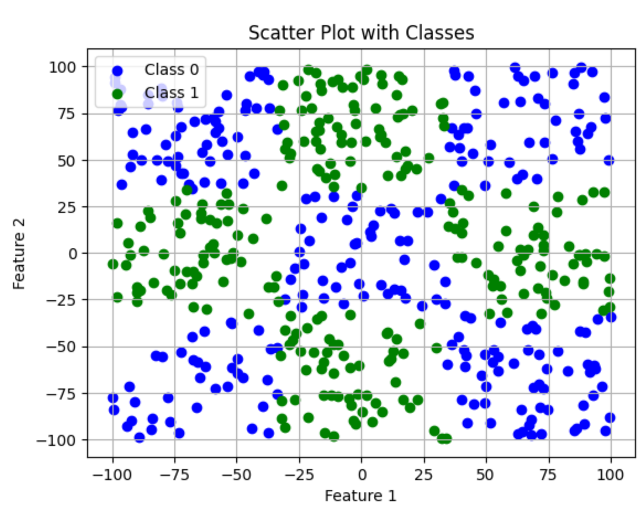
Rysunek 36: Zbiór danych *easy*

- *rings3-regular*, wymagane minimalne F-measure: 0.75



Rysunek 37: Zbiór danych *rings3-regular*

- *xor3*, wymagane minimalne F-measure: 0.97



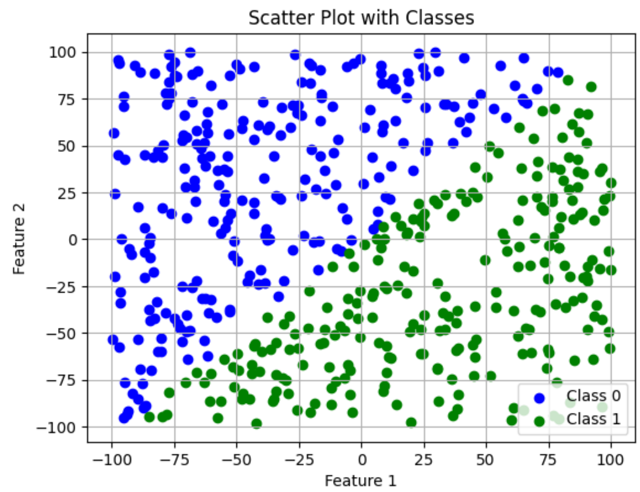
Rysunek 38: Zbiór danych *xor3*

5.2 Moje rozwiązanie

5.2.1 Zbiór danych easy

Test uczenia sieci
Dla tych danych użyłam sieci z jedną warstwą ukrytą, na której znajdowało się 5 neuronów. Użyłam także metody usprawnienia uczenia gradientowego RMSProp.

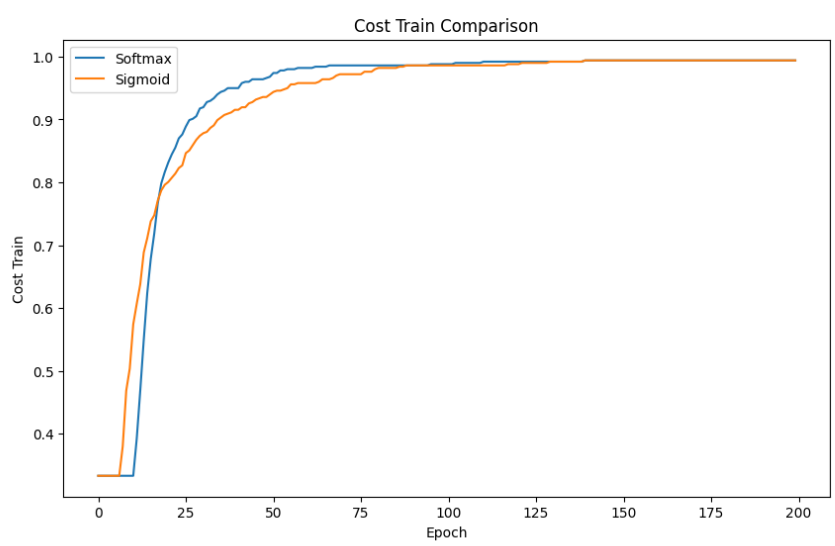
Wyniki:
F score dla zbioru easy dla zbioru treningowego: 0.9939999759999041
F score dla zbioru easy dla zbioru testowego: 0.9919998719979519



Rysunek 39: Dane testowe po przepuszczeniu przez wytrenowany model dla zbioru *easy*

Dane te, jak sama ich nazwa wskazuje są proste do wytrenowania - model szybko osiąga wysoki F score.

Wartość funkcji kosztu względem liczby epok, w zależności od funkcji aktywacji na ostatniej warstwie



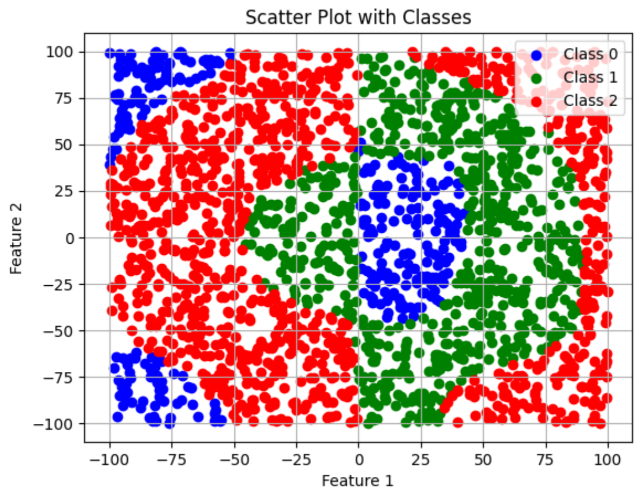
Rysunek 40: Wartość funkcji kosztu względem liczby epok, w zależności od funkcji aktywacji na ostatniej warstwie dla danych *easy*

Jak widać w przypadku tych prostych danych na samym początku lepiej radzi sobie model z funkcją sigmoidalną na końcu. Jednak szybko się to zmienia i lepiej zbiega model z funkcją softmax na końcu. Zatem dla tych danych lepiej radzi sobie funkcja softmax na końcu.

5.2.2 Zbiór danych rings3-regular

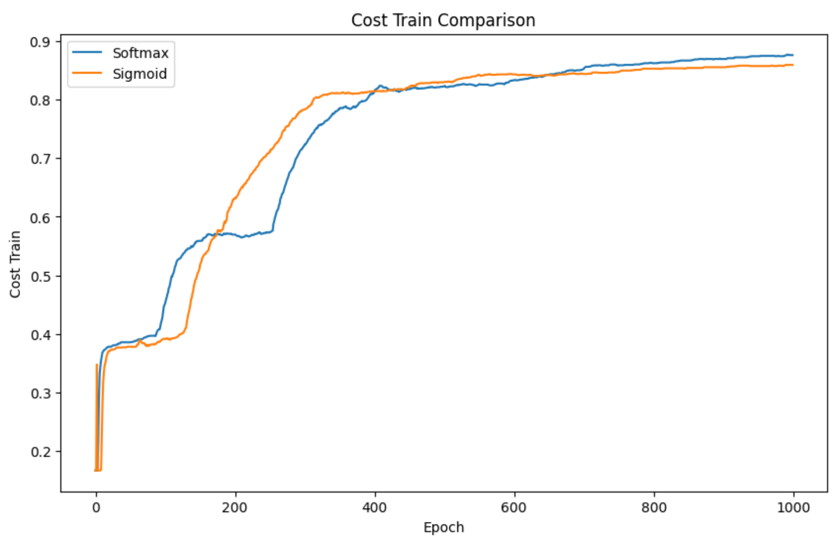
Test uczenia sieci
Dla tych danych użyłam sieci z dwiema warstwami ukrytymi, po 10 neuronów na każdej z nich. Użyłam także metody usprawnienia uczenia gradientowego RMSProp.

Wyniki:
F score dla zbioru rings3-regular dla zbioru treningowego: 0.8762070396818249
F score dla zbioru rings3-regular dla zbioru testowego: 0.8682398177560771



Rysunek 41: Dane testowe po przepuszczeniu przez wytrenowany model dla zbioru *rings3-regular*

Wartośc funkcji kosztu względem liczby epok, w zależności od funkcji aktywacji na ostatniej warstwie



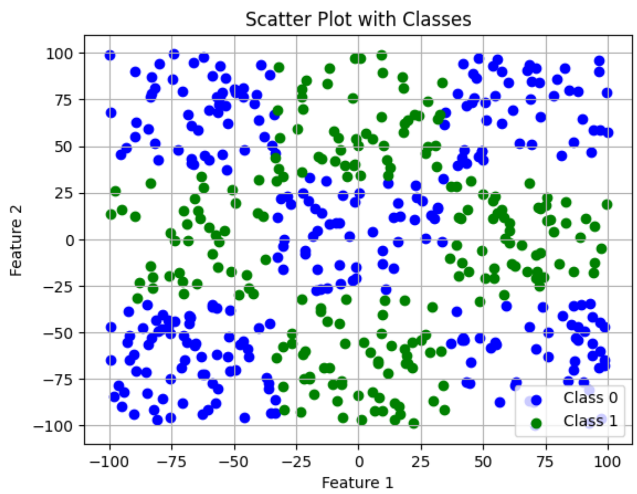
Rysunek 42: Wartość funkcji kosztu względem liczby epok, w zależności od funkcji aktywacji na ostatniej warstwie dla danych *rings3-regular*

Jak widać na wykresie, modele wymieniają się wczasie - w pewnym zakresie epok lepiej radzi sobie model z funkcją sigmoidalną na końcu, w innym zakresie - z funkcją softmax. Jednak na końcu trenowania lepiej radzi sobie model z funkcją softmax.

5.2.3 Zbiór danych xor3

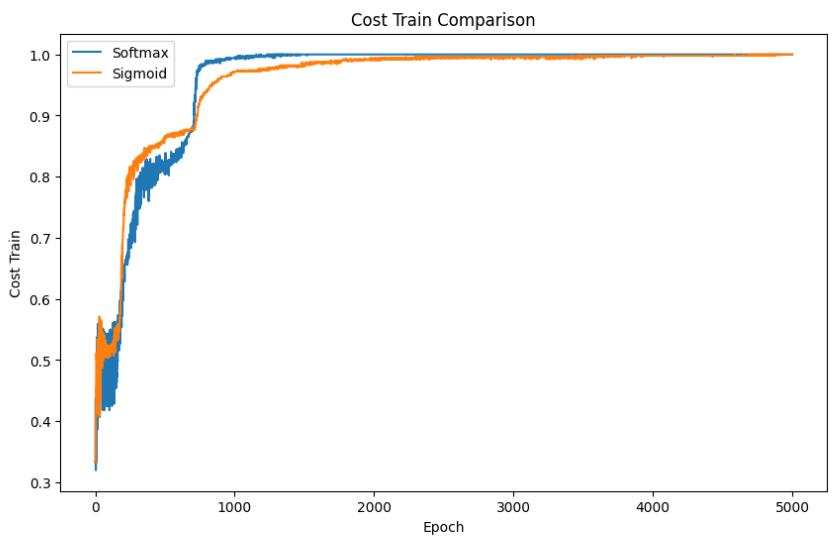
Test uczenia sieci
Dla tych danych użyłam sieci z dwiema warstwami ukrytymi, po 10 neuronów na każdej z nich. Użyłam także metody usprawnienia uczenia gradientowego RMSProp.

Wyniki:
F score dla zbioru xor3 dla zbioru treningowego: 1.0
F score dla zbioru xor3 dla zbioru testowego: 0.9795011397366307



Rysunek 43: Dane testowe po przepuszczeniu przez wytrenowany model dla zbioru *xor3*

Wartośc funkcji kosztu względem liczby epok, w zależności od funkcji aktywacji na ostatniej warstwie



Rysunek 44: Wartość funkcji kosztu względem liczby epok, w zależności od funkcji aktywacji na ostatniej warstwie dla danych *xor3*

W tym przypadku, jak i w tym wyżej modele się wymieniają - raz lepiej radzi sobie model z funkcją softmax na końcu, a raz z sigmoidalną, jednak przez znacznie dłuższy czas lepiej radzi sobie funkcja softmax.

5.3 Podsumowanie, wnioski

W związku z tym, że w 3 rozpatrywanych zbiorach lepiej radzi sobie funkcja softmax, bądź radzi sobie podobnie jak sigmoidalna, to do klasyfikacji na ostatniej warstwie lepiej stosować funkcję softmax. Sama sieć MLP dobrze radzi sobie w problemach klasyfikacji.

6 NN5: Testowanie różnych funkcji aktywacji

6.1 Opis zadania

Celem zadania było rozszerzenie istniejącej implementacji sieci i metody uczącej o możliwość wyboru funkcji aktywacji:

- sigmoid
- liniowa
- tanh
- ReLU

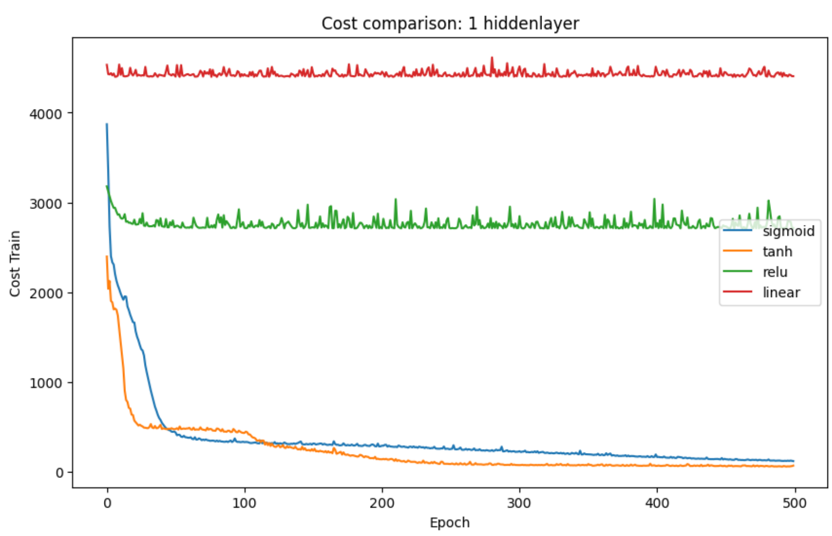
oraz porównanie szybkości uczenia i skuteczności sieci w zależności od liczby warstw oraz funkcji aktywacji. Dla danych *multimodal-large* mieliśmy przetestować powyższe funkcje aktywacji dla 3 architektur sieci: z jedną warstwą ukrytą, z dwiema warstwami ukrytymi i z trzema warstwami ukrytymi.
Następnie spośród tych 12 architektur z funkcjami aktywacji wybrać 2 które dały najlepsze wyniki i przetestować ich skuteczność na zbiorach danych:

- *steps-large* (regresja)
- *rings5-regular* (klasyfikacja)
- *rings3-regular* (klasyfikacja)

6.2 Moje rozwiązanie

6.2.1 Testy na zbiorze multimodal-large

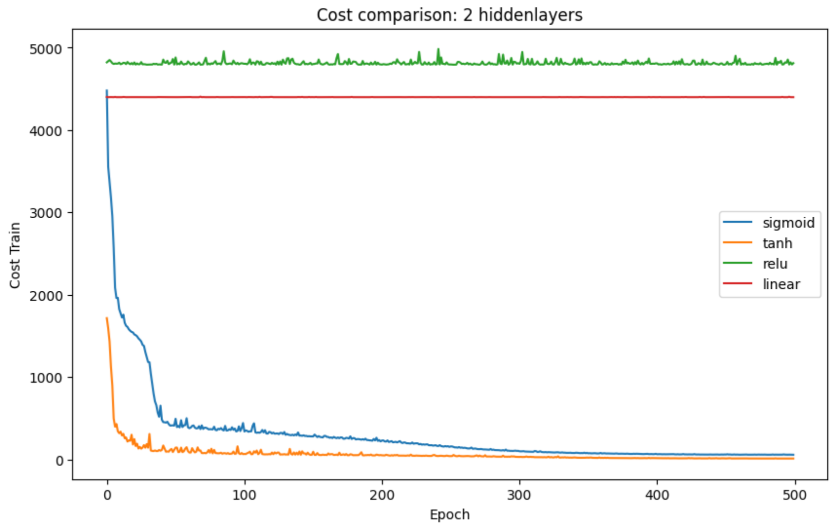
Architektura z 1 warstwą ukrytą:
Dla architektury z 1 warstwą ukrytą, zastosowałam na niej 10 neuronów. Użyłam także metody Momentu usprawnienia uczenia gradientowego.



Rysunek 45: Wartość funkcji kosztu względem liczby epok, w zależności od rodzaju funkcji aktywacji dla danych *multimodal-large* dla architektury z 1 warstwą ukrytą

Widzimy, że najlepiej radzą sobie funkcje: sigmoidalna oraz tanh, natomiast najlepszy wynik dała tanh. Znacznie gorzej radzi sobie ReLU, dla której MSE tylko na początku trochę spadło, a potem stało w miejscu. Najgorzej poradziła sobie funkcja liniowa, która praktycznie cały czas stała w miejscu.

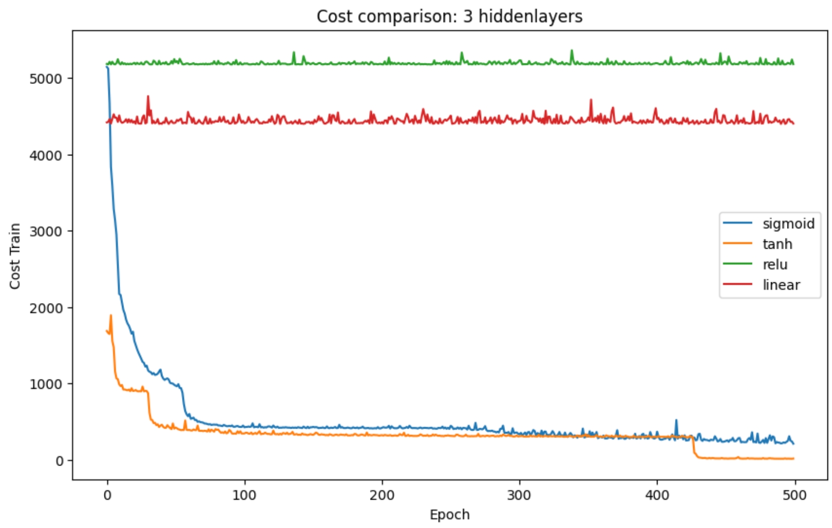
Architektura z 2 warstwami ukrtytymi:
Dla architektury z 2 warstwami ukrytymi zastoswałam na każdej z nich po 7 neuronów. Użyłam także metody Momentu usprawnienia uczenia gradientowego.



Rysunek 46: Wartość funkcji kosztu względem liczby epok, w zależności od rodzaju funkcji aktywacji dla danych *multimodal-large* dla architektury z 2 warstwami ukrytymi

Dla tej architektury również najlepiej poradziły sobie sigmoidalna oraz tanh, z czym ta druga nieco lepiej niż pierwsza. Liniowa oraz ReLU praktycznie stały w miejscu.

Architektura z 3 warstwami ukrytymi:
Dla architektury z 3 warstwami ukrytymi zastosowałam na każdej z nich 4 neurony. Użyłam także metody Momentu usprawnienia uczenia gradientowego.



Rysunek 47: Wartość funkcji kosztu względem liczby epok, w zależności od rodzaju funkcji aktywacji dla danych *multimodal-large* dla architektury z 3 warstwami ukrytymi

Tak jak w przypadku 2 poprzednich architektur, najlepiej poradziła sobie sigmoidalna oraz tanh. Tanh zaliczyła pod koniec znaczący spadek, przez co dała lepszy wynik od sigmoidy. Znów, funkcja liniowa oraz ReLU praktycznie stały w miejscu.

Podsumowanie:

	simgoid	tanh	relu	linear
1 warstwa ukryta	114.92	63.74	2797.34	4442.99
2 warstwy ukryte	50.41	5.55	4824.08	4433.59
3 warstwy ukryte	218.6	10.32	5228.5	4438.95

Tabela 1: Końcowe MSE na zbiorze testowym dla każdej z architektur

Widzimy zatem, że ze względu na MSE na zbiorze testowym, najlepsze zestawy to:

- 2 warstwy ukryte [7,7], funkcja aktywacji: tanh,
- 3 warstwy ukryte [4,4,4], funkcja aktywacji: tanh

6.2.2 Testy 2 najlepszych architektur

a) Zbiór danych steps-large
MSE test steps-large - zestaw 1: 17.80337314690982
MSE test steps-large - zestaw 2: 73.83116204354711

b) Zbiór danych rings5-regular
Fscore test rings5-regular - zestaw 1: 0.7633794311539064
Fscore test rings5-regular - zestaw 2: 0.5076734485587648

c) Zbiór danych rings3-regular
Fscore test rings3-regular - zestaw 1: 0.8489927902815868
Fscore test rings3-regular - zestaw 2: 0.6860232045170638

Widzimy, że najlepsze wyniki jeśli chodzi zestaw danych dla regresji dał zestaw 1. Natomiast dla 2 pozostałych zbiorów danych, do problemu klasyfikacji, najlepiej poradził sobie zestaw 2.

6.3 Podsumowanie, wnioski

Funkcją aktywacji która radziła sobie najlepiej dla zbioru *multimodal-large* była tanh. Niezależnie od liczby warstw ukrytych zawsze dawała nam najlepszy wynik. Funkcja sigmoidalna radziła sobie niewiele gorzej od niej, wciąż dając zadowalające wyniki. Funkcja relu nie poradziła sobie dobrze. Przy jej używaniu, w jednym z zestawów MSE na początku trochę spadało, jednak potem utrzymywało praktycznie stałą wartość. W pozostałych przypadkach MSE było praktycznie stałe. Funkcja liniowa poradziła sobie zdecydowanie najgorzej. MSE od samego początku było praktycznie stałe, podobnie jak w przypadku funkcji ReLU.

Architekturą która dała najlepsze wyniki, była architektura z 2 warstwami ukrytymi, w której umieściłam po 7 na każdej z warstw ukrytych. Architektury z 1 oraz z 3 warstwami ukrytymi radziły sobie gorzej, jednak która z nich najgorzej, to zależy od użytej funkcji aktywacji.

W związku z powyższym, do tematu regresji najlepiej nadaje się funkcja aktywacji tanh bądź sigmoid.

Być może w temacie klasyfikacji funkcja ReLU dałaby zadowalające wyniki, jednak iż w przypadku zbioru do regresji *multimodal-large* dawała złe wyniki, to nie została wybrana do dalszych testów.