Metody Inteligencji Obliczeniowej w Analizie Danych
Sprawozdanie NN

Filip Langiewicz
nr indeksu 327293

Politechnika Warszawska
Wydział Matematyki i Nauk Informacyjnych
15 kwietnia 2025

Spis treści

1. Wstęp

W ramach laboratoriów poświęconych sieciom neuronowym skupialiśmy się na budowie i działaniu perceptronu wielowarstwowego (MLP) – sieci neuronowej o strukturze feedforward. Głównym celem ćwiczeń było nie tylko osiągnięcie dobrych wyników na prostych zadaniach klasyfikacyjnych czy regresyjnych, ale przede wszystkim zrozumienie mechanizmów działania sieci, w tym obserwacja zmian parametrów w trakcie procesu uczenia.

W trakcie kolejnych zajęć rozwijaliśmy własną implementację MLP, korzystając wyłącznie z bibliotek do obliczeń macierzowych, takich jak NumPy – bez wykorzystywania gotowych frameworków do uczenia maszynowego. Każde laboratorium poświęcone było innemu zagadnieniu związanemu z budową i uczeniem sieci, a zdobyta wiedza była weryfikowana poprzez eksperymenty na przygotowanych zbiorach danych.

W niniejszym sprawozdaniu przedstawiam kolejne etapy implementacji sieci MLP oraz analizę wpływu różnych hiperparametrów i elementów architektury na efektywność procesu uczenia. Doświadczenia zdobyte podczas eksperymentów służą jako podstawa do sformułowania wniosków dotyczących praktycznych aspektów trenowania perceptronów wielowarstwowych.

2. NN1: Bazowa implementacja

2.1. Opis tematu

Pierwszy etap laboratorium polegał na stworzeniu od podstaw prostej sieci neuronowej typu MLP (Multi-Layer Perceptron), z możliwością łatwej konfiguracji architektury. Implementacja miała umożliwiać zmianę liczby warstw, liczby neuronów w każdej warstwie, a także ręczne ustawianie wag i biasów. Jako funkcję aktywacji zastosowano klasyczną funkcję sigmoidalną, natomiast na warstwie wyjściowej dopuszczono użycie funkcji liniowej.

Celem zadania było wykorzystanie przygotowanej sieci do rozwiązania problemu regresji na dwóch zbiorach danych: *square-simple* oraz *steps-large*. Eksperymenty przeprowadzono dla trzech różnych architektur:

- jedna warstwa ukryta z 5 neuronami,
- jedna warstwa ukryta z 10 neuronami,
- dwie warstwy ukryte, po 5 neuronów każda.

Wszystkie sieci należało dobrać ręcznie (bez automatycznego strojenia hiperparametrów), tak aby błąd średniokwadratowy (MSE) na nieznormalizowanym zbiorze testowym nie przekraczał wartości 9. Głównym celem tego etapu było zapoznanie się ze strukturą sieci oraz przygotowanie elastycznego kodu, który będzie rozwijany w kolejnych tygodniach.

2.2. Opis wykonanej pracy i wyniki eksperymentów

W ramach laboratorium zaimplementowano prostą wersję sieci neuronowej typu *Multi-Layer Perceptron* (MLP) bez algorytmu wstecznej propagacji błędu (ang. *backpropagation*). Klasa `MLPNoBackprop` umożliwia ręczne ustawianie wag i biasów, a także wybór funkcji aktywacji dla warstw ukrytych oraz wyjściowych (`sigmoid` lub `linear`).

Główne metody klasy to:

- `forward(X)` – propagacja sygnału przez kolejne warstwy sieci,
- `predict(X)` – uzyskanie końcowej predykcji modelu,
- `set_weights_and_biases(layer_idx, W, b)` – ręczne ustawienie wag i biasów w zadanej warstwie,
- `mse(y_true, y_pred)` – obliczenie średniego błędu kwadratowego.

2.2.1. Eksperyment 1: Square-simple dataset

Pierwszy eksperyment przeprowadzono na zbiorze danych `square-simple`, który zawierał jedną zmienną wejściową oraz jedną zmienną wyjściową. Wykorzystano sieć o architekturze $[1, 5, 1]$ (jedna warstwa ukryta z 5 neuronami).

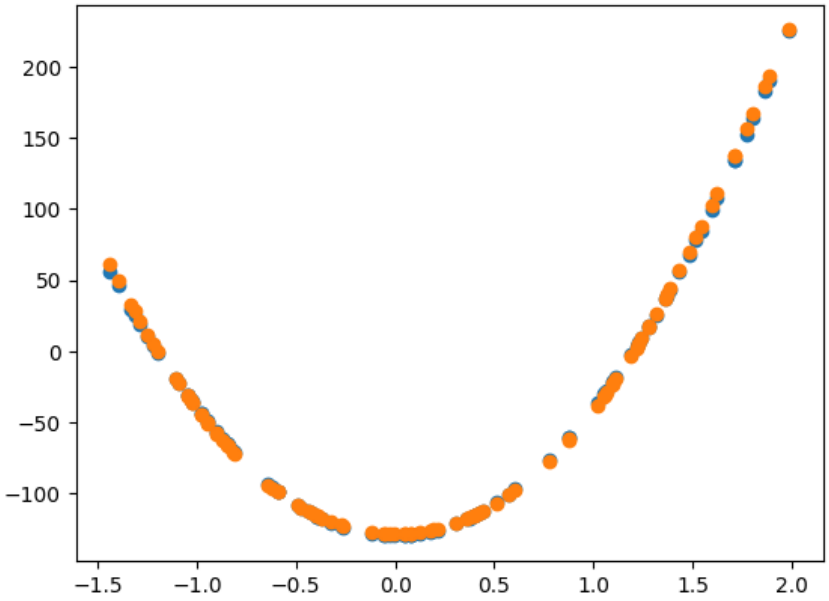
Konfiguracja sieci:

- Wagi warstwy 1: $\begin{bmatrix} 1.50 & -1.50 & 2.30 & 2.30 & 2.30 \end{bmatrix}$
- Biasy warstwy 1: $\begin{bmatrix} -3 & -3 & 1 & 1 & 1 \end{bmatrix}$
- Wagi warstwy 2: $\begin{bmatrix} 880 \\ 900 \\ 1 \\ 1 \\ 1 \end{bmatrix}$
- Bias warstwy 2: -215.20

Przy ustalaniu wag i biasów kierowano się chęcią dopasowania funkcji sigmoidalnych do kształtu funkcji kwadratowej.

Wyniki:

- Obliczono wartość błędu MSE między wartościami rzeczywistymi a predykcją modelu. Wynosiło ono około 2.65.
- Wyniki dopasowania przedstawiono graficznie na wykresie 2.1.



Rysunek 2.1: Dopasowanie przewidzianych wartości do faktycznych danych

2.2.2. Eksperymenty na zbiorze steps-large

W tej części wykonano trzy eksperymenty z różnymi architekturami sieci, bazując na ręcznie ustawionych wagach i biasach. Ponownie starano się wagi dobrać tak, żeby możliwie najlepiej odwzorowały funkcję schodkową.

- **Eksperyment 1:** Sieć $[1, 5, 1]$

  Wagi aktywne w 3 pierwszych neuronach warstwy ukrytej. Uzyskano dokładność predykcji na poziomie:

  $$\text{MSE} = 4{,}02$$

- **Eksperyment 2:** Sieć $[1, 10, 1]$

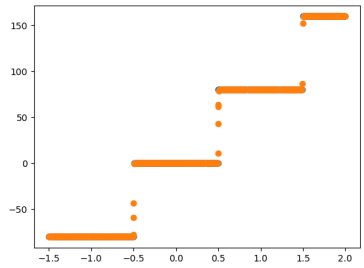
  Rozszerzona warstwa ukryta (10 neuronów), ale tylko 3 z nich aktywne. Wystarczyło rozszerzyć działającą architekturę z 3 neuronami o kolejne nieaktywne. Wynik identyczny jak w poprzednim eksperymencie:

  $$\text{MSE} = 4{,}02$$

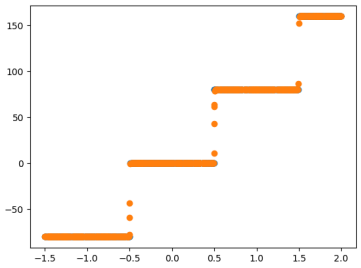
- **Eksperyment 3:** Sieć $[1, 5, 5, 1]$

  Dodano drugą warstwę ukrytą. Dane były propagowane przez aktywne 3 neurony, a następnie liniowo połączone. Błąd wzrósł:

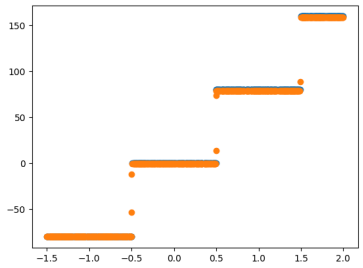
  $$\text{MSE} = 6{,}08$$



Rysunek 2.2: Dopasowanie dla eksperymentu 1



Rysunek 2.3: Dopasowanie dla eksperymentu 2



Rysunek 2.4: Dopasowanie dla eksperymentu 3

Zwiększanie liczby neuronów i warstw bez ich aktywacji (niezerowych wag) nie poprawiło jakości predykcji. Wręcz przeciwnie – zbyt głęboka architektura bez odpowiedniego ustawienia parametrów pogorszyła wynik.

2.3. Wnioski

Mimo że sieć nie była trenowana, uzyskano zadowalające dopasowanie do danych. Poprzez odpowiedni dobór wag sieć skutecznie odwzorowała funkcję nieliniowe. Było to jednak zadanie nad wyraz żmudne i czasochłonne oraz wymagało przemyślenia, jakie wagi i biasy dobrać.

3. NN2: Implementacja propagacji wstecznej błędu

3.1. Opis tematu

W ramach drugiego etapu należało rozszerzyć bazową implementację sieci MLP o mechanizm uczenia przy użyciu algorytmu propagacji wstecznej błędu (backpropagation). Kluczowym elementem było poprawne wyliczanie gradientów błędu oraz aktualizacja wag na podstawie pochodnych funkcji aktywacji.

Zadanie obejmowało:

- implementację uczenia sieci metodą propagacji wstecznej błędu,
- przygotowanie dwóch wariantów uczenia:
  - batch learning – aktualizacja wag po każdej pełnej epoce,
  - mini-batch learning – aktualizacja wag po każdej porcji danych,
- inicjalizację wag z rozkładu jednostajnego na przedziale $[0, 1]$, z możliwością zastosowania alternatywnych metod inicjalizacji, takich jak Xavier lub He,
- opracowanie narzędzia do wizualizacji zmian wag sieci w kolejnych iteracjach treningu – pomocne przy analizie problemów z uczeniem (np. stagnacja błędu),
- porównanie efektywności oraz szybkości uczenia pomiędzy obiema metodami aktualizacji wag.

Aby zweryfikować poprawność implementacji, należało przeprowadzić eksperymenty na trzech zbiorach danych:

- `square-simple` (oczekiwane MSE $\leqslant 4$),
- `steps-small` (oczekiwane MSE $\leqslant 4$),
- `multimodal-large` (oczekiwane MSE $\leqslant 40$).

Wszystkie wartości błędu średniokwadratowego (MSE) należało obliczać na nieznormalizowanych danych testowych.

3.2. Opis wykonanej pracy i wyniki eksperymentów

Zrealizowano pełną implementację wielowarstwowego perceptronu (MLP) z wykorzystaniem biblioteki `NumPy`, obejmującą:

- propagację sygnału w przód,
- propagację wsteczną błędu z aktualizacją wag,
- możliwość wyboru funkcji aktywacji (`sigmoid`, `relu`, `tanh`, `linear`) - dodatkowo, żeby móc osiągać lepsze wyniki,
- dwie wersje algorytmu uczącego: z aktualizacją wag po każdej epoce (full-batch) oraz po każdej porcji danych (mini-batch),
- inicjalizację wag z rozkładu jednostajnego $U(0, 1)$ oraz dodatkowo metodą Xavier, He oraz z rozkładu normalnego,
- wizualizację zmian wag w kolejnych epokach, wspierającą analizę procesu uczenia.

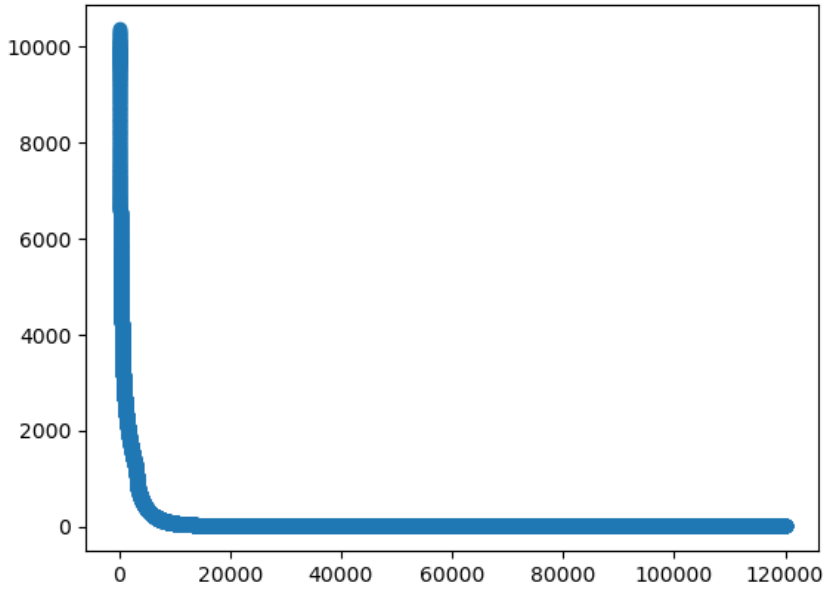
Zaprojektowany model umożliwia ręczne konfigurowanie architektury sieci oraz eksperymentowanie z różnymi parametrami i funkcjami aktywacji.

Testy i walidacja Zgodnie z wymaganiami, przeprowadzono testy uczenia na dostarczonych zbiorach danych:

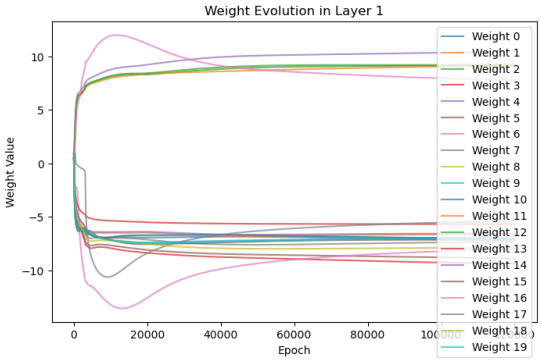
- `square-simple` – osiągnięto MSE poniżej 4,

$$\text{TRAIN MSE} = 1.3276979668856685$$

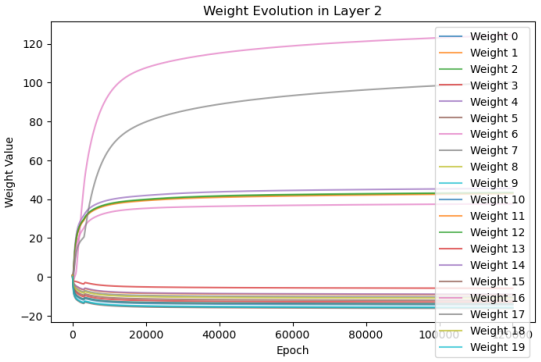
$$\text{TEST MSE} = 2.2589804633740136$$



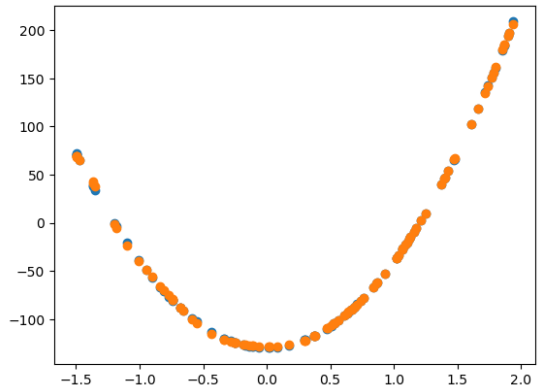
Rysunek 3.1: Zmiana MSE w trakcie uczenia sieci



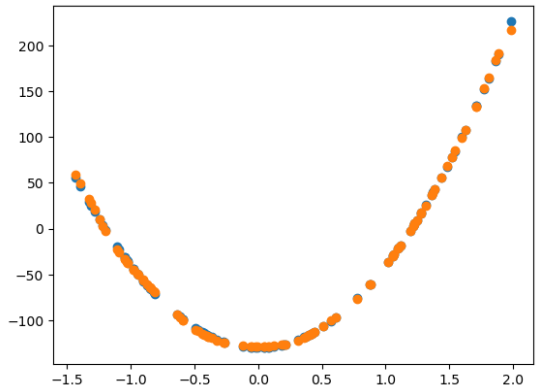
Rysunek 3.2: Zmiana wag w warstwie 1



Rysunek 3.3: Zmiana wag w warstwie 2



Rysunek 3.4: Dopasowanie dla eksperymentu na zbiorze treningowym

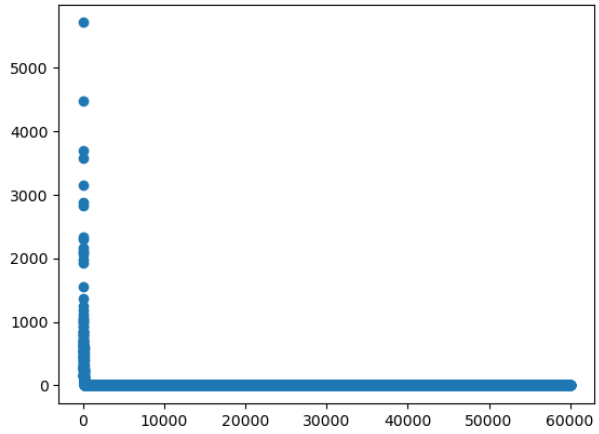


Rysunek 3.5: Dopasowanie dla eksperymentu na zbiorze testowym

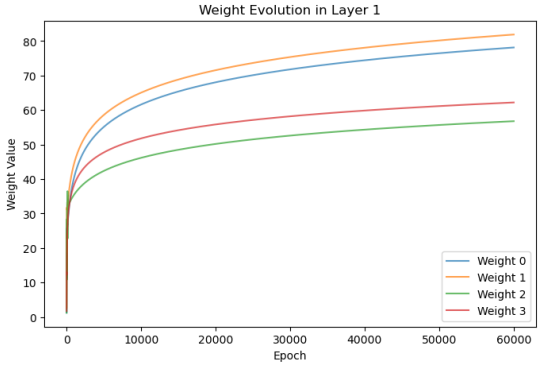
- `steps-small` – osiągnięto MSE poniżej 4,

$$\text{TRAIN MSE} = 0.06394917974933745$$

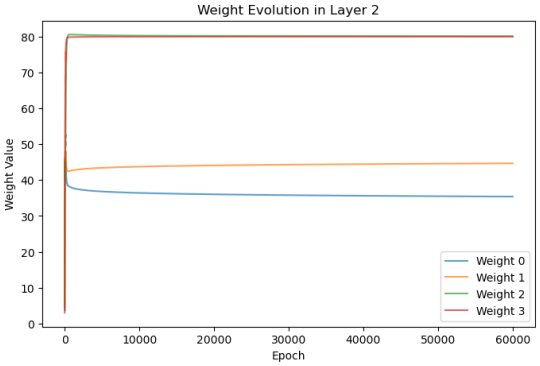
$$\text{TEST MSE} = 88.80559994456716$$



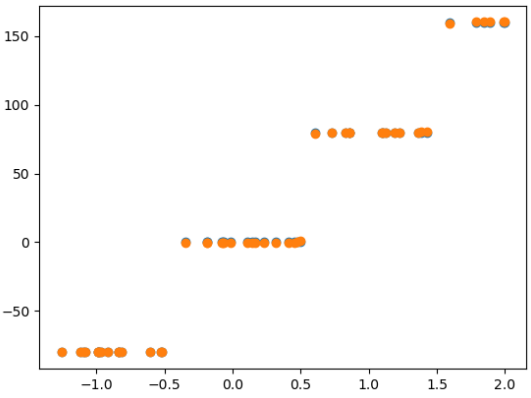
Rysunek 3.6: Zmiana MSE w trakcie uczenia sieci



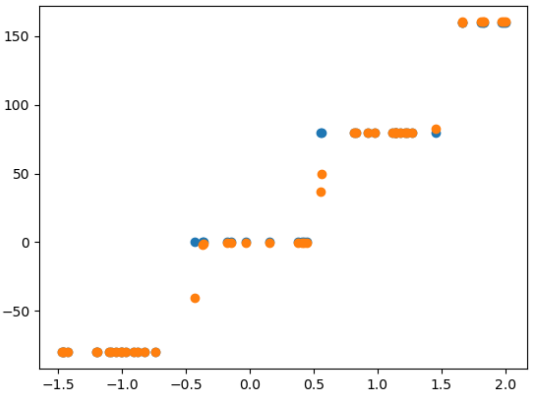
Rysunek 3.7: Zmiana wag w warstwie 1



Rysunek 3.8: Zmiana wag w warstwie 2



Rysunek 3.9: Dopasowanie dla eksperymentu na zbiorze treningowym



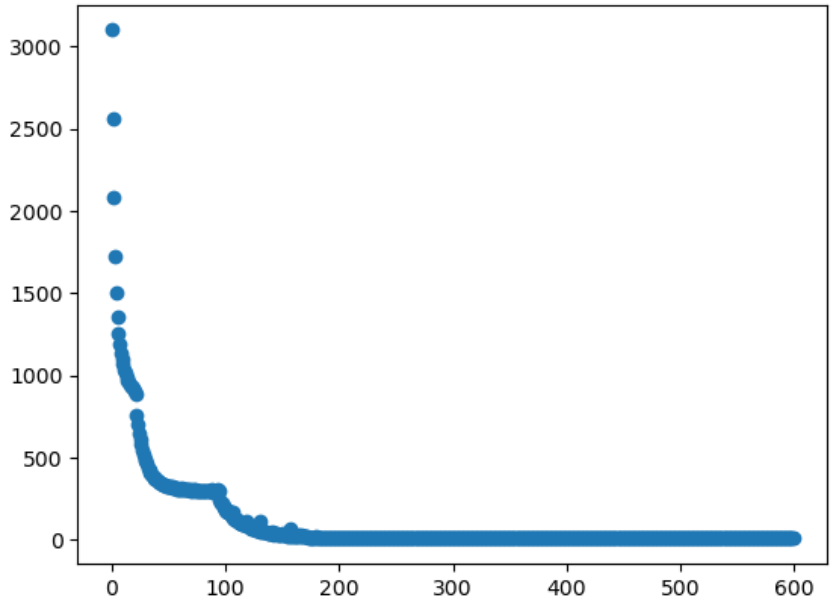
Rysunek 3.10: Dopasowanie dla eksperymentu na zbiorze testowym

Z powodu innego rozkładu na zmiennej x, nie udało się uzyskać MSE mniejszego niż 4 na zbiorze testowym.

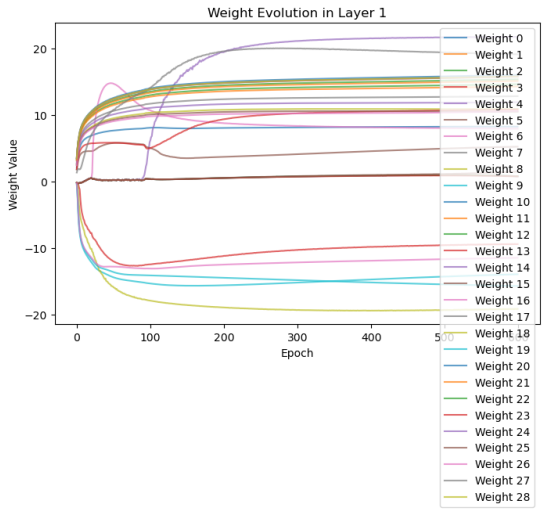
- `multimodal-large` – osiągnięto MSE poniżej 40,

$$\text{TRAIN MSE} = 8.06491650040556$$

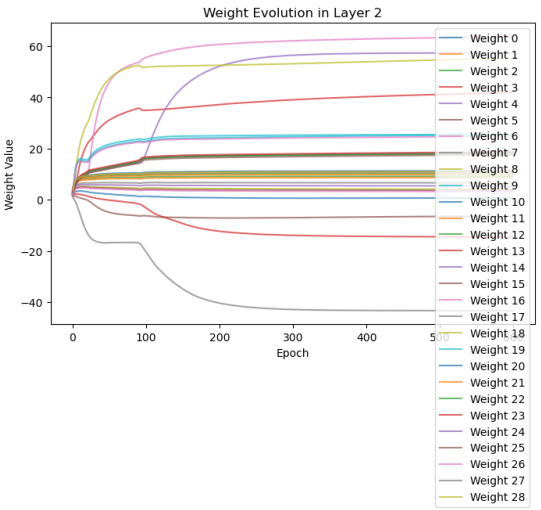
$$\text{TEST MSE} = 3.006926668875326$$



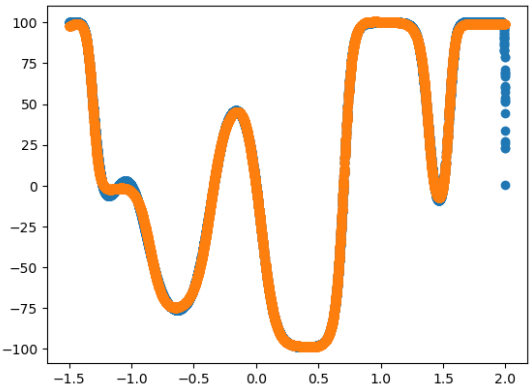
Rysunek 3.11: Zmiana MSE w trakcie uczenia sieci



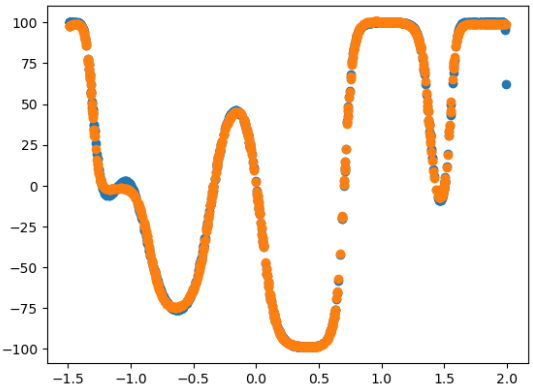
Rysunek 3.12: Zmiana wag w warstwie 1



Rysunek 3.13: Zmiana wag w warstwie 2



Rysunek 3.14: Dopasowanie dla eksperymentu na zbiorze treningowym



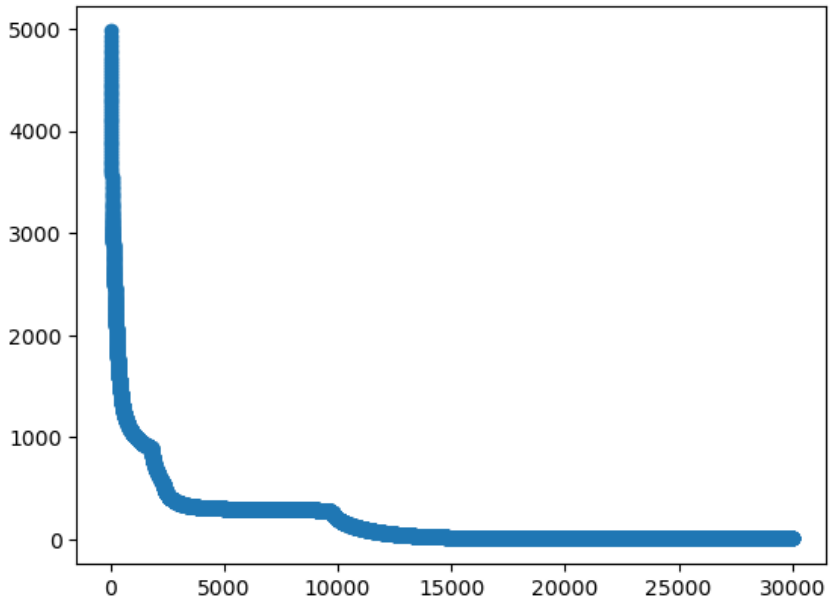
Rysunek 3.15: Dopasowanie dla eksperymentu na zbiorze testowym

co potwierdza poprawność działania implementacji.

Dodatkowo wykonano porównanie wersji z uczeniem pełnym i mini-batch na zbiorze `multimodal-large`, obserwując różnice w szybkości i stabilności konwergencji. Uzyskano wyniki:

$$\text{TRAIN MSE} = 8.252847512039889$$

$$\text{TEST MSE} = 3.1679614911161305$$



Rysunek 3.16: Zmiana MSE w trakcie uczenia sieci

Wizualizacja wag pozwoliła również na identyfikację problemów z zanikającym gradientem w niektórych przypadkach, co zostało skorygowane przez odpowiedni dobór funkcji aktywacji i inicjalizacji wag.

3.3. Wnioski

Na podstawie przeprowadzonych eksperymentów można sformułować następujące wnioski:

- Sieć neuronowa z poprawnie zaimplementowaną propagacją wsteczną błędu jest w stanie skutecznie nauczyć się funkcji z dostarczonych zbiorów danych, osiągając wartości MSE znacznie poniżej progów wymaganych w zadaniu.
- Wykorzystanie wizualizacji wartości wag w kolejnych epokach umożliwia lepsze zrozumienie procesu uczenia i może być pomocne w diagnozowaniu problemów z konwergencją. W przeprowadzonych eksperymentach ewolucja wag przebiegała stabilnie, co potwierdza poprawność implementacji.
- Różne podejścia do inicjalizacji wag (np. Xavier, He) mogą mieć znaczący wpływ na tempo i skuteczność uczenia.
- Porównanie wariantów z pełnym batchowaniem oraz mini-batch pokazało, że przy odpowiednim dostrojeniu hiperparametrów (rozmiar batcha, learning rate) mini-batch może prowadzić do efektywnego uczenia przy zachowaniu stabilności. Przy pełnym batchowaniu sieć zbiegała dużo, dużo wolniej.
- Dla prostych zbiorów, takich jak `square-simple`, nawet sieci o relatywnie prostej architekturze (jedna warstwa ukryta, sigmoid + linear) są wystarczające do uzyskania bardzo dobrych wyników.

Podsumowując, wszystkie kluczowe elementy zadania zostały zrealizowane: zaimplementowano uczenie z propagacją wsteczną błędu, przygotowano wizualizacje wag, zrealizowano zarówno uczenie pełne, jak i mini-batch oraz uzyskano poprawne wyniki dla wymaganych zbiorów danych.

4. NN3: Implementacja momentu i normalizacji gradientu

4.1. Opis tematu

W ramach trzeciego etapu należało usprawnić proces uczenia sieci MLP poprzez implementację dwóch technik modyfikujących działanie algorytmu gradientowego:

- momentu – czyli uwzględnienia poprzedniego kroku aktualizacji wagi, co przyspiesza zbieżność i stabilizuje uczenie,
- normalizacji gradientu metodą RMSProp – pozwalającą na dynamiczne dostosowywanie kroku uczenia dla każdego parametru.

Celem było porównanie szybkości zbieżności obu podejść oraz ich wpływu na jakość treningu. Eksperymenty należało przeprowadzić na następujących zbiorach danych:

- `square-large` (oczekiwane MSE $\leqslant 1$),
- `steps-large` (MSE $\leqslant 3$),
- `multimodal-large` (MSE $\leqslant 9$).

4.2. Opis wykonanej pracy i wyniki eksperymentów

Zgodnie z treścią zadania, w ramach ćwiczenia zaimplementowano dwa usprawnienia algorytmu gradientowego stosowanego do uczenia sieci neuronowej typu MLP:

- **Momentum** – mechanizm uwzględniający poprzednie aktualizacje wag, co pozwala na przyspieszenie procesu uczenia oraz zmniejszenie oscylacji gradientów. Implementacja opiera się na przechowywaniu wektorów prędkości dla wag i biasów, aktualizowanych w każdej iteracji zgodnie z wybranym współczynnikiem β.
- **RMSProp** – metoda normalizacji gradientu polegająca na śledzeniu średniej kroczącej kwadratów gradientów dla każdej wagi, co pozwala na dynamiczne dostosowywanie tempa

uczenia w różnych wymiarach przestrzeni parametrów. Dla stabilności obliczeń zastosowano dodatkowy parametr ϵ.

Obie metody zostały zaimplementowane jako opcje w funkcji `train()` klasy `MLP`, sterowane przez parametr `optimizer`, przyjmujący wartości `'momentum'` lub `'rmsprop'` (domyślnie `None` - brak optymalizacji).

W celu porównania skuteczności i szybkości zbieżności obu metod, przeprowadzono eksperymenty na trzech zbiorach danych wskazanych w poleceniu:

- `square-large` (docelowe MSE $\leqslant 1$),
- `steps-large` (docelowe MSE $\leqslant 3$),
- `multimodal-large` (docelowe MSE $\leqslant 9$).

Dla każdego zbioru sieć trenowano z użyciem obu technik optymalizacji. Wyniki przedstawiono w tabeli 4.1. Zauważalna jest znaczna przewaga RMSProp w przypadku zbioru `square-large`, gdzie udało się osiągnąć bardzo niski błąd MSE. Natomiast dla zbioru `steps-large`, lepsze wyniki uzyskano stosując momentum. W przypadku zbioru `multimodal-large`, obie metody dały poprawne rezultaty (poniżej progu 9.0), przy czym momentum osiągnął zadowalający wynik znacznie szybciej (już po 400 epokach).

Tabela 4.1: Porównanie skuteczności metod Momentum i RMSProp na różnych zbiorach danych

Zbiór danych	**Metoda**	**Epoki**	**Train MSE**	**Test MSE**
square-large	Momentum	2000	55.18	1339.04
square-large	RMSProp	3000	0.0897	33.54
steps-large	Momentum	15055	2.79	0.87
steps-large	RMSProp	45000	5.73	5.35
multimodal-large	Momentum	400	7.91	3.93
multimodal-large	RMSProp	1000	6.94	2.52

4.3. Wnioski

Na podstawie przeprowadzonych eksperymentów (tabela 4.1) można zauważyć, że skuteczność metod usprawniających uczenie gradientowe – *momentum* oraz *RMSProp* – jest silnie zależna od charakterystyki danych.

W przypadku zbioru `square-large`, metoda RMSProp zdecydowanie przewyższyła momentum, osiągając bardzo niski błąd MSE przy rozsądnej liczbie epok. Z kolei momentum nie zdołało doprowadzić do zbieżności w zadanej liczbie epok, co sugeruje, że w tym przypadku adaptacyjna natura RMSProp lepiej radzi sobie z niestabilnym krajobrazem błędu.

Dla zbioru `steps-large` sytuacja była odwrotna – momentum pozwoliło osiągnąć niski błąd testowy przy mniejszej liczbie epok niż RMSProp. Może to świadczyć o tym, że przy bardziej „stopniowym" charakterze funkcji celu, momentum efektywnie akumuluje gradienty i prowadzi do szybszej konwergencji.

W zbiorze `multimodal-large`, obie metody uzyskały wyniki spełniające kryteria jakości (MSE < 9). RMSProp osiągnął nieco niższy błąd testowy, jednak potrzebował ponad dwa razy więcej epok niż momentum, co może wskazywać na jego większą stabilność kosztem szybkości.

Podsumowując:

- **RMSProp** jest skuteczniejszy w sytuacjach, gdy krajobraz błędu jest bardziej złożony i trudny do optymalizacji.
- **Momentum** sprawdza się lepiej w zadaniach o bardziej regularnej strukturze, pozwalając na szybszą konwergencję.
- Nie istnieje jedna dominująca metoda – wybór strategii optymalizacji powinien być dostosowany do konkretnego problemu oraz jego właściwości.

5. NN4: Rozwiązywanie zadania klasyfikacji

5.1. Opis tematu

Celem czwartego etapu było dostosowanie sieci MLP do rozwiązywania problemów klasyfikacyjnych. W tym celu należało zaimplementować funkcję `softmax` w warstwie wyjściowej oraz odpowiednio zmodyfikować algorytm propagacji wstecznej, uwzględniając pochodną tej funkcji.

W ramach zadania należało:

- zaimplementować wariant sieci z funkcją `softmax` w warstwie wyjściowej,
- porównać skuteczność i szybkość uczenia w porównaniu z siecią używającą standardowej funkcji aktywacji (np. sigmoidalnej),
- przeprowadzić eksperymenty klasyfikacyjne na trzech zbiorach danych:
  - `rings3-regular` (wymagane F-measure $\geqslant 0.75$),
  - `easy` (F-measure $\geqslant 0.99$),
  - `xor3` (F-measure $\geqslant 0.97$).

5.2. Opis wykonanej pracy i wyniki eksperymentów

Zaimplementowano funkcję `softmax` jako jedną z funkcji aktywacji, która może być używana w warstwie wyjściowej sieci neuronowej. Funkcja ta została dodana do zestawu dostępnych aktywacji w klasie `Layer`, z odpowiednim uwzględnieniem jej specyfiki działania.

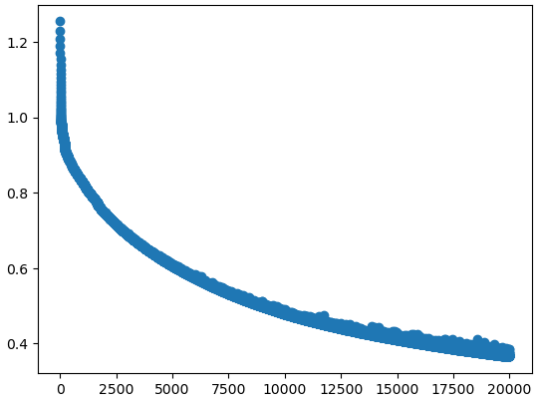
Wprowadzono również niezbędne zmiany w algorytmie uczenia – w przypadku klasyfikacji wieloklasowej (czyli gdy używana jest funkcja `softmax` na warstwie wyjściowej), zamiast klasycznego błędu MSE wykorzystywana jest funkcja straty `cross-entropy`, a propagacja błędów w ostatniej warstwie została uproszczona do formy `(y_pred - y_true)`, co wynika bezpośrednio z pochodnej funkcji softmax skompensowanej przez pochodną entropii krzyżowej.

Aby sprawdzić wpływ użycia funkcji `softmax` w warstwie wyjściowej, przeprowadzono eksperymenty porównawcze: trenowano sieć zarówno z klasyczną funkcją aktywacji (`sigmoid`), jak i z `softmax`. Dla obu wariantów mierzono skuteczność klasyfikacji przy użyciu metryki F-measure.

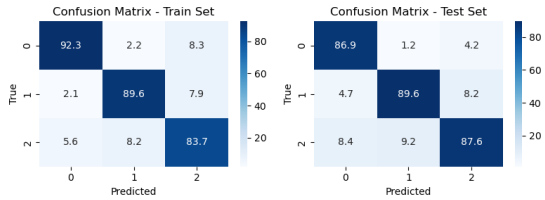
Eksperymenty przeprowadzono na trzech zestawach danych:

- `rings3-regular` – wymagany F-measure: 0.75.

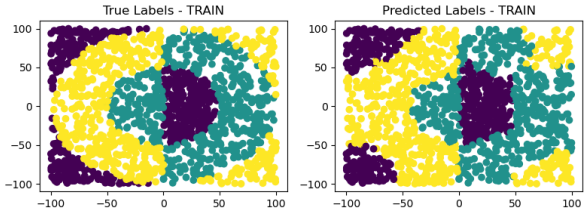
  Dla funkcji `softmax` na ostatniej warstwie:



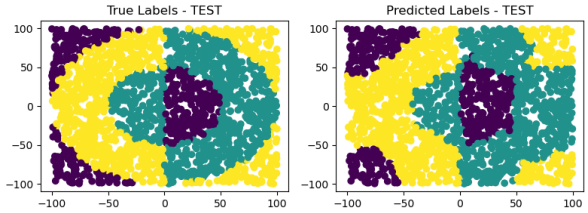
Rysunek 5.1: Wartości funkcji straty podczas eksperymentu



Rysunek 5.2: Macierz pomyłek

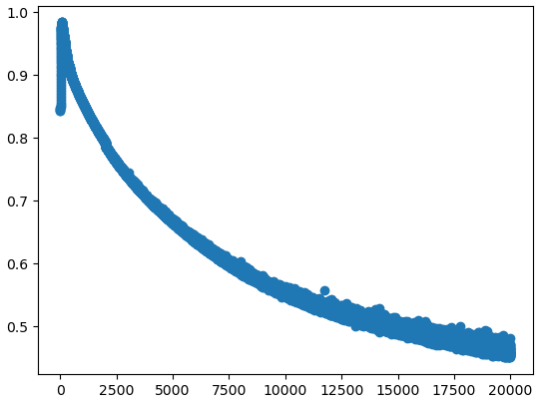


Rysunek 5.3: Dopasowanie dla eksperymentu na zbiorze treningowym

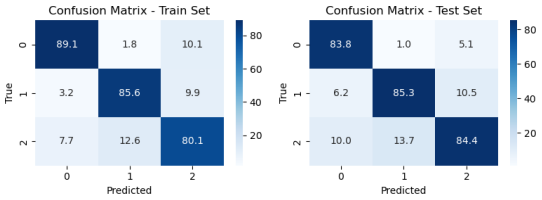


Rysunek 5.4: Dopasowanie dla eksperymentu na zbiorze testowym

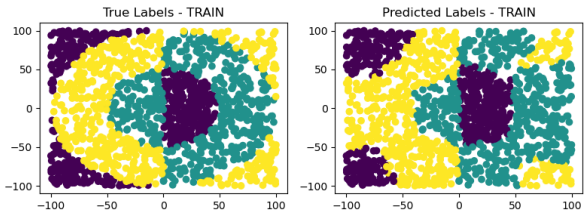
Dla funkcji `sigmoid` na ostatniej warstwie:



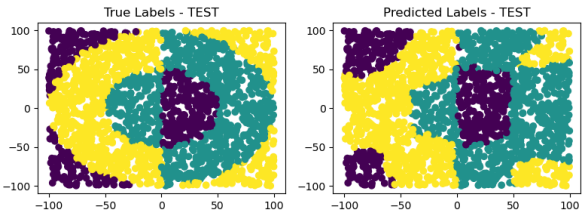
Rysunek 5.5: Wartości funkcji straty podczas eksperymentu



Rysunek 5.6: Macierz pomyłek



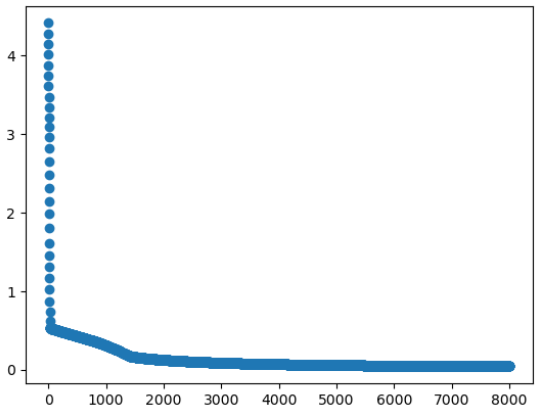
Rysunek 5.7: Dopasowanie dla eksperymentu na zbiorze treningowym



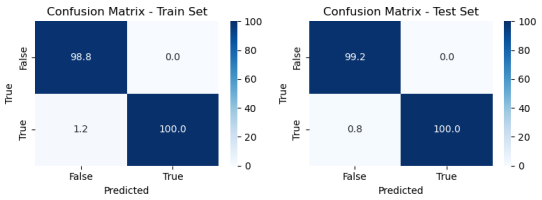
Rysunek 5.8: Dopasowanie dla eksperymentu na zbiorze testowym

- `easy` – wymagany F-measure: 0.99,

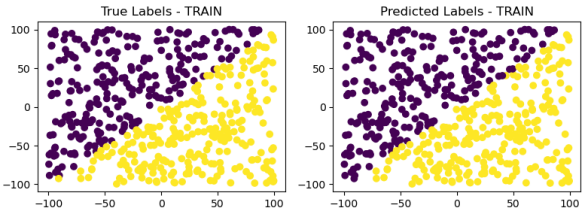
  Dla funkcji `softmax` na ostatniej warstwie:



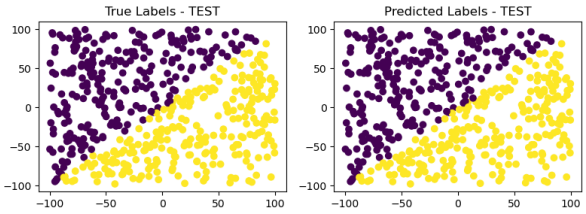
Rysunek 5.9: Wartości funkcji straty podczas eksperymentu



Rysunek 5.10: Macierz pomyłek

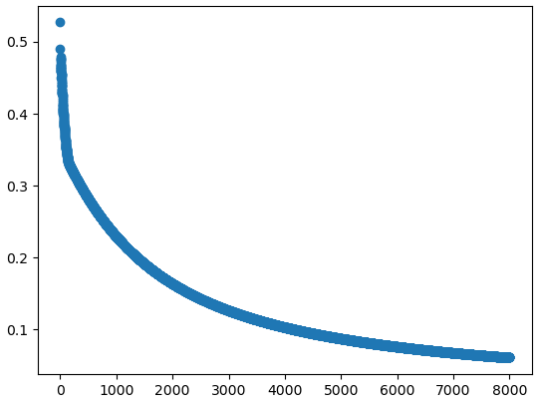


Rysunek 5.11: Dopasowanie dla eksperymentu na zbiorze treningowym

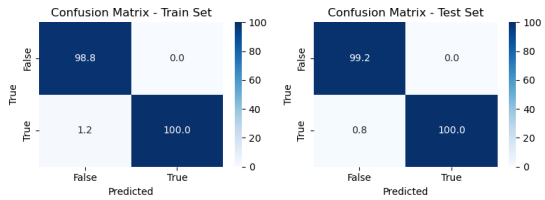


Rysunek 5.12: Dopasowanie dla eksperymentu na zbiorze testowym

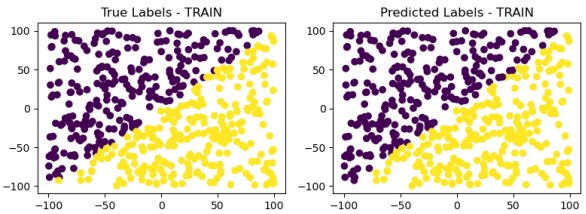
Dla funkcji `sigmoid` na ostatniej warstwie:



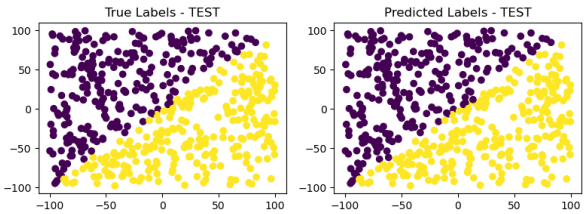
Rysunek 5.13: Wartości funkcji straty podczas eksperymentu



Rysunek 5.14: Macierz pomyłek



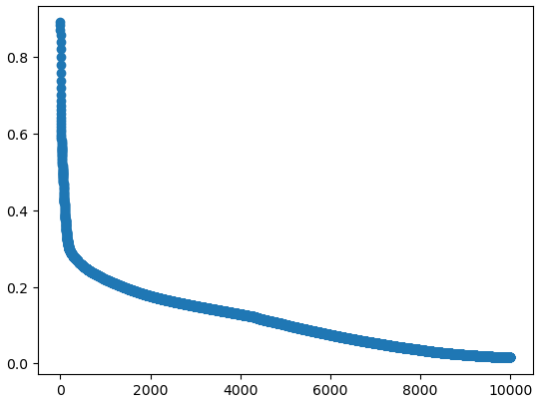
Rysunek 5.15: Dopasowanie dla eksperymentu na zbiorze treningowym



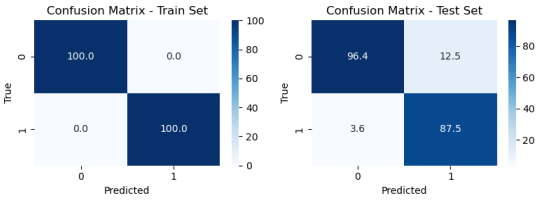
Rysunek 5.16: Dopasowanie dla eksperymentu na zbiorze testowym

- `xor3` – wymagany F-measure: 0.97.

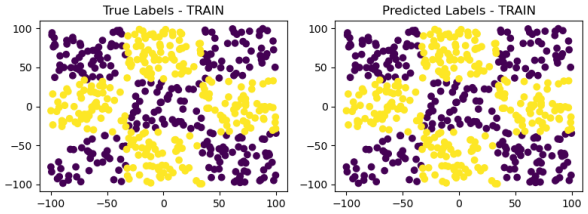
Dla funkcji `softmax` na ostatniej warstwie:



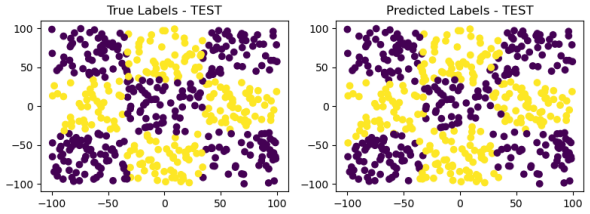
Rysunek 5.17: Wartości funkcji straty podczas eksperymentu



Rysunek 5.18: Macierz pomyłek

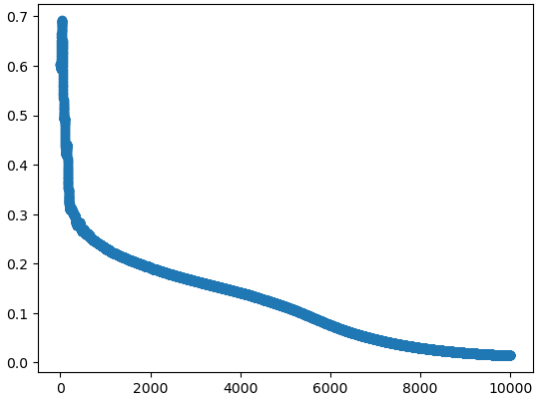


Rysunek 5.19: Dopasowanie dla eksperymentu na zbiorze treningowym

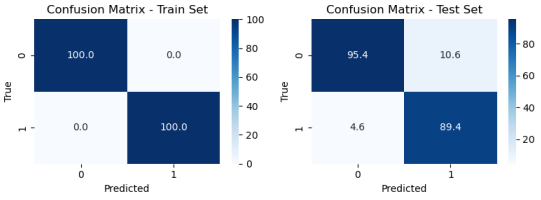


Rysunek 5.20: Dopasowanie dla eksperymentu na zbiorze testowym

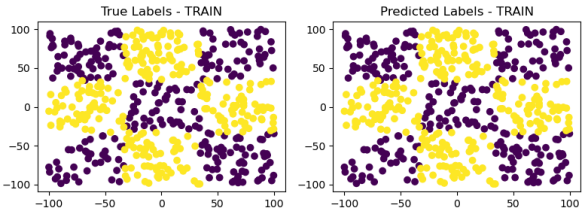
Dla funkcji `sigmoid` na ostatniej warstwie:



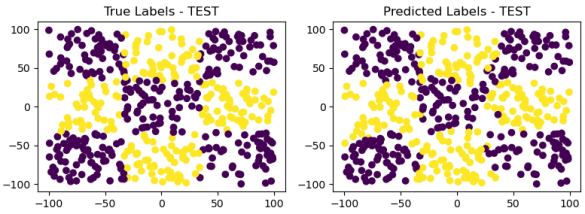
Rysunek 5.21: Wartości funkcji straty podczas eksperymentu



Rysunek 5.22: Macierz pomyłek



Rysunek 5.23: Dopasowanie dla eksperymentu na zbiorze treningowym



Rysunek 5.24: Dopasowanie dla eksperymentu na zbiorze testowym

Zestawienie uzyskanych wyników przedstawiono w tabeli 5.1. Eksperymenty wykazały, że funkcja `softmax` daje wyraźną poprawę skuteczności w przypadku bardziej złożonych zbiorów (np. `rings3-regular`), natomiast w zbiorach prostych (takich jak `easy` i `xor3`) różnice są marginalne lub niezauważalne.

Tabela 5.1: Porównanie skuteczności (F1-score) dla funkcji `softmax` i `sigmoid` na trzech zbiorach danych.

Zbiór danych	**Funkcja**	**F1-score (train)**	**F1-score (test)**	**Epoki**
`rings3-regular`	softmax	0.885	0.883	20000
	sigmoid	0.849	0.848	20000
`easy`	softmax	0.994	0.996	8000
	sigmoid	0.994	0.996	8000
`xor3`	softmax	1.000	0.922	10000
	sigmoid	1.000	0.926	10000

5.3. Wnioski

Przeprowadzone eksperymenty wykazały znaczną różnicę w skuteczności między siecią neuronową z funkcją aktywacji `softmax` a siecią z funkcją `sigmoid` na różnych zestawach danych. W szczególności, zastosowanie funkcji `softmax` na warstwie wyjściowej przyczyniło się do poprawy wyników w bardziej złożonych problemach klasyfikacyjnych, takich jak zbiór `rings3-regular`. W tym przypadku sieć z funkcją `softmax` osiągnęła wyższe wartości F1-score zarówno na zbiorze treningowym, jak i testowym, w porównaniu do sieci z funkcją `sigmoid`, co sugeruje, że `softmax` lepiej radzi sobie z klasyfikacją wieloklasową.

W przypadku zbiorów danych o prostszej strukturze, takich jak `easy` i `xor3`, różnice w skuteczności między funkcjami `sigmoid` a `softmax` były marginalne. Dla obu funkcji uzyskano bardzo

wysokie wyniki (bliskie 1.0), zarówno w przypadku zbioru treningowego, jak i testowego, co wskazuje, że w takich problemach wybór funkcji aktywacji ma mniejsze znaczenie.

Pomimo że funkcja `softmax` wykazała przewagę w bardziej złożonych zadaniach, czas treningu był zbliżony dla obu funkcji, co może sugerować, że różnica w efektywności wynika głównie z poprawy jakości klasyfikacji, a nie z szybkości uczenia. Z kolei w prostych zadaniach funkcja `sigmoid` okazała się wystarczająca i zapewniała równie dobre wyniki, co funkcja `softmax`, co może być korzystne w przypadku mniejszych zbiorów danych lub mniej złożonych problemów.

Podsumowując, wyniki eksperymentów sugerują, że dobór funkcji aktywacji zależy od charakterystyki problemu. W przypadku złożonych zadań klasyfikacyjnych, takich jak `rings3-regular`, warto zastosować `softmax`, podczas gdy w prostszych problemach, takich jak `easy` i `xor3`, różnice w skuteczności mogą być znikome, a funkcja `sigmoid` jest wystarczająca.

6. NN5: Testowanie różnych funkcji aktywacji

6.1. Opis tematu

Piąty etap laboratorium polegał na rozszerzeniu implementacji sieci MLP o możliwość wyboru jednej z kilku funkcji aktywacji:

- `sigmoid`,
- `liniowa`,
- `tanh`,
- `ReLU`.

Wymagało to dostosowania procesu uczenia do obliczania pochodnych tych funkcji, szczególnie dla `ReLU`, gdzie należy zwrócić uwagę na poprawną implementację gradientu.

Następnie należało:

- porównać szybkość i skuteczność uczenia sieci w zależności od funkcji aktywacji i liczby neuronów w warstwach ukrytych,
- przetestować sieci z jedną, dwiema i trzema warstwami ukrytymi,
- przeprowadzić testy dla zbioru `multimodal-large` (regresja) z wszystkimi kombinacjami funkcji aktywacji i architektur,
- wybrać dwa najlepsze zestawy (funkcja + architektura) i zbadać ich skuteczność na dodatkowych zbiorach:
  - regresja: `steps-large`,
  - klasyfikacja: `rings5-regular`, `rings3-regular`.

6.2. Opis wykonanej pracy i wyniki eksperymentów

W istniejącej klasie `Layer` dodano już wcześniej obsługę czterech funkcji aktywacji: `sigmoid`, `tanh`, `ReLU` oraz `liniowej (linear)`. Dla każdej z tych funkcji zaimplementowano zarówno samą funkcję aktywacji, jak i jej pochodną, co umożliwia poprawne działanie algorytmu wstecznej propagacji błędów (backpropagation). Obsługa została zrealizowana poprzez mapowanie nazw funkcji do odpowiednich implementacji w konstruktorze warstwy.

W klasie `MLP` (czyli całej sieci) niezbędne zmiany dotyczyły jedynie przekazywania konfiguracji poszczególnych warstw, które teraz mogą korzystać z dowolnej wspieranej funkcji aktywacji.

Zgodnie z treścią zadania, przeprowadzono testy porównawcze dla:

- czterech funkcji aktywacji: `sigmoid`, `tanh`, `ReLU`, `linear`,
- dziewięciu architektur sieci: z jedną, dwiema oraz trzema warstwami ukrytymi i z pięcioma, piętnastoma lub trzydziestoma neuronami w każdej,
- zbioru danych: `multimodal-large` (regresja) – jako zestaw testów wstępnych.

W dalszej części przeanalizowano skuteczność dwóch najlepiej działających kombinacji (architektura + funkcja aktywacji) na innych zestawach:

- **regresja**: `steps-large`,
- **klasyfikacja**: `rings5-regular` oraz `rings3-regular`.

W każdej konfiguracji rejestrowano wartości metryk: dla klasyfikacji była to miara F1, natomiast dla regresji – błąd średniokwadratowy (MSE).

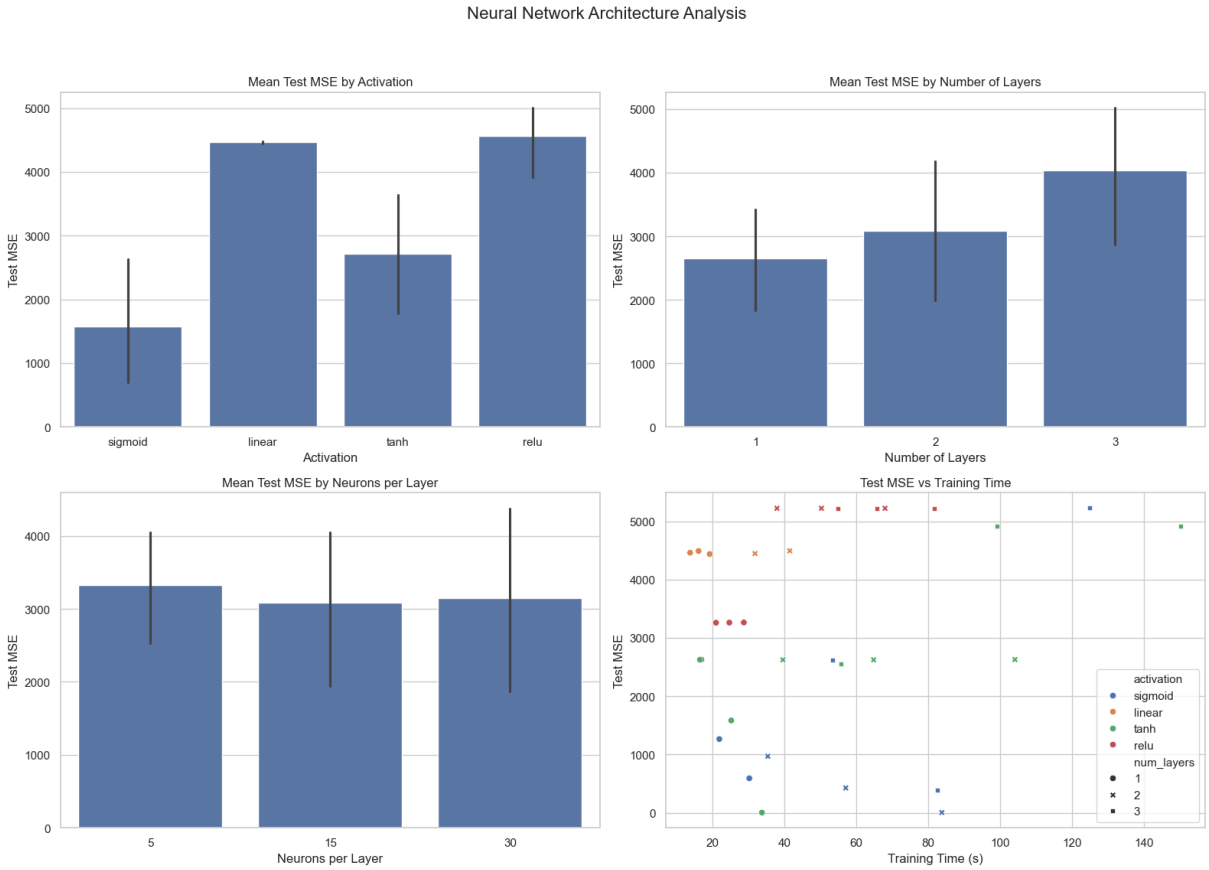
W przypadku funkcji `ReLU` należy zwrócić uwagę na jej nieciągłość i niedefiniowalność pochodnej w punkcie zero. W implementacji zastosowano podejście uproszczone, w którym pochodna w zerze przyjmuje wartość 0. Takie rozwiązanie jest powszechnie stosowane w praktyce i wystarczające do działania algorytmu uczącego.

Sieć została przetestowana na zbiorze `multimodal-large` w kontekście regresji. Przeprowadzono eksperymenty dla różnych architektur, tj. z 1, 2 oraz 3 warstwami ukrytymi, a także z różną liczbą neuronów (5, 15, 30) w każdej z warstw. Porównano wpływ wyboru funkcji aktywacji oraz rozmiaru architektury na jakość modelu (przy pomocy metryki MSE) oraz czas treningu.

Poniżej przedstawiono wyniki w tabeli 6.1. Wyniki obejmują błąd średniokwadratowy (MSE) dla zbioru treningowego i testowego oraz czas treningu w sekundach.

Index	Activation	Layers	Neurons	Train MSE	Test MSE	Time (s)
0	sigmoid	1	5	2524.2491	2626.9964	16.8452
1	sigmoid	1	15	1162.8615	1265.1260	21.8260
2	sigmoid	1	30	566.8165	591.7993	30.1676
3	sigmoid	2	5	897.2541	969.8620	35.3060
4	sigmoid	2	15	383.3314	427.2207	57.0049
5	sigmoid	2	30	10.2833	5.4283	83.7310
6	sigmoid	3	5	2519.5291	2623.9695	53.4086
7	sigmoid	3	15	349.6628	391.9772	82.4637
8	sigmoid	3	30	5180.1813	5225.2175	124.9105
9	linear	1	5	4425.7446	4459.5987	13.7048
10	linear	1	15	4450.7512	4489.6896	16.0564
11	linear	1	30	4402.1740	4438.1446	19.1577
12	linear	2	5	4410.1969	4446.0900	31.7616
13	linear	2	15	4455.1488	4492.2951	41.4196
14	linear	2	30	–	–	49.0842
15	linear	3	5	–	–	43.7652
16	linear	3	15	–	–	55.3203
17	linear	3	30	–	–	72.7604
18	tanh	1	5	2521.2707	2624.6958	16.4338
19	tanh	1	15	1444.0571	1583.4117	25.1404
20	tanh	1	30	8.7729	3.9785	33.6823
21	tanh	2	5	2519.4741	2623.5453	39.5176
22	tanh	2	15	2520.9421	2623.7528	64.7595
23	tanh	2	30	2521.3772	2627.4165	104.0645
24	tanh	3	5	2457.8098	2551.5236	55.6650
25	tanh	3	15	4902.6178	4913.0372	99.1389
26	tanh	3	30	4908.7384	4916.6090	150.1471
27	relu	1	5	3203.7676	3260.1158	20.9089
28	relu	1	15	3203.7546	3261.5652	24.6123
29	relu	1	30	3204.9487	3263.8422	28.6432
30	relu	2	5	5177.2785	5221.3858	37.8660
31	relu	2	15	5177.2802	5221.3736	50.2355
32	relu	2	30	5177.2810	5221.3704	67.9352
33	relu	3	5	5177.2810	5221.3703	54.8858
34	relu	3	15	5177.2819	5221.3667	65.7672
35	relu	3	30	5177.2807	5221.3716	81.6498

Tabela 6.1: Porównanie wyników sieci neuronowych dla różnych funkcji aktywacji, liczby warstw i liczby neuronów



Rysunek 6.1: Zagregowane wyniki dla zbioru multimodal

Najlepszy jakościowo model to ten z funkcją aktywacji `tanh`, jedną warstwą i 30 neuronami. Drugim najlepszym modelem jest sieć z funkcją aktywacji `sigmoid`, dwoma warstwami i 30 neuronami.

Następnie przeprowadzono eksperymenty na pozostałych zbiorach. Wyniki eksperymentów dla różnych architektur sieci neuronowych przedstawiono w Tabeli 6.2. Dla dwóch głównych typów architektur (arch1 i arch2) uzyskano różne wyniki zależnie od liczby epok i funkcji aktywacji.

Architektura 1 jest siecią o dwóch warstwach:

- Pierwsza warstwa ma 1 neuron na wejściu i 30 na wyjściu, z funkcją aktywacji *tanh*.
- Druga warstwa ma 30 neuronów na wejściu i 1 neuron na wyjściu, z funkcją aktywacji *linear*.

Architektura 2 to sieć o trzech warstwach:

- Pierwsza warstwa ma 1 neuron na wejściu i 30 na wyjściu, z funkcją aktywacji *sigmoid*.
- Druga warstwa ma 30 neuronów na wejściu i 30 na wyjściu, również z funkcją aktywacji *sigmoid*.

- Trzecia warstwa ma 30 neuronów na wejściu i 1 neuron na wyjściu, z funkcją aktywacji *linear*.

Architektura	Epoki	Czas Trenowania [s]	TRAIN SCORE	TEST SCORE
steps-large arch1	1000	610.67	30.11 (MSE)	30.28 (MSE)
steps-large arch2	1000	610.67	17.93 (MSE)	16.52 (MSE)
rings3-regular arch1	10000	192.68	0.615 (F1)	0.609 (F1)
rings3-regular arch2	10000	227.39	0.371 (F1)	0.419 (F1)
rings5-regular arch1	10000	474.32	0.847 (F1)	0.751 (F1)
rings5-regular arch2	10000	610.67	0.628 (F1)	0.541 (F1)

Tabela 6.2: Wyniki testów dla różnych architektur sieci neuronowych

6.3. Wnioski

W przeprowadzonych eksperymentach z użyciem sieci neuronowych dla różnych funkcji aktywacji, architektur i zbiorów danych uzyskano interesujące wyniki, które pozwalają na ocenę wpływu różnych parametrów na efektywność modeli.

- **Wpływ funkcji aktywacji:** Na podstawie wyników eksperymentów na zbiorze `multimodal-large` oraz innych testach, najlepsze wyniki uzyskano dla funkcji aktywacji `tanh`, szczególnie w przypadku modeli z 1 warstwą i 30 neuronami. Funkcja `sigmoid` również okazała się skuteczna, szczególnie w architekturach z dwiema warstwami i większą liczbą neuronów, gdzie uzyskano lepsze wyniki na zbiorze testowym w porównaniu do innych funkcji aktywacji.

- **Wpływ liczby warstw i neuronów:** Zwiększenie liczby warstw oraz neuronów w każdej z warstw nie zawsze prowadziło do poprawy wyników. Zauważono, że większe architektury, zwłaszcza te z 3 warstwami, nie zawsze dawały lepsze wyniki w odniesieniu do jakości modelu. W przypadku zbioru `steps-large` i architektur `arch1` oraz `arch2` uzyskano najlepsze wyniki dla mniejszej liczby warstw (1-2 warstwy), co może sugerować, że zbyt rozbudowane sieci mogą prowadzić do przeuczenia (overfitting) na danych testowych, zwłaszcza w przypadku funkcji aktywacji `ReLU`, gdzie zauważono stabilność wyników w czasie treningu.

- **Czas treningu:** Zauważono również, że czas treningu zależał głównie od liczby warstw oraz neuronów w każdej z nich. Modele o mniejszej liczbie neuronów i warstw (np. `arch1`)

osiągały lepsze wyniki w krótszym czasie, podczas gdy bardziej rozbudowane architektury, takie jak `arch2`, wymagały znacznie więcej czasu na trening.

- **Wnioski ogólne:**
  - Najlepsze wyniki w przypadku regresji uzyskano dla modelu z funkcją aktywacji `tanh`, jedną warstwą i 30 neuronami, co sugeruje, że nie zawsze rozbudowane modele są najlepsze, szczególnie w kontekście regresji.
  - W przypadku klasyfikacji, architektura `arch1` z mniejszą liczbą warstw okazała się bardziej skuteczna w większości przypadków.
  - Wyniki wskazują na to, że zarówno funkcja aktywacji, jak i architektura sieci mają kluczowy wpływ na jakość modelu i czas treningu, dlatego dobór odpowiednich parametrów w zależności od zadania (regresja lub klasyfikacja) jest kluczowy.

Podsumowując, przeprowadzone eksperymenty pozwoliły na określenie najlepszych kombinacji architektury i funkcji aktywacji dla różnych typów danych, co stanowi ważny krok w kierunku optymalizacji sieci neuronowych dla konkretnych zadań.

7. NN6: Zjawisko przeuczenia + regularyzacja

7.1. Opis tematu

Szósty i ostatni etap laboratorium dotyczył problemu przeuczenia (overfittingu) w sieciach neuronowych. W celu przeciwdziałania temu zjawisku należało zaimplementować dwa mechanizmy:

- regularyzację wag (np. L2),
- wczesne zatrzymanie treningu (early stopping) na podstawie wzrostu błędu na zbiorze walidacyjnym.

Następnie należało przeprowadzić eksperymenty na wybranych zbiorach danych i porównać skuteczność modeli na zbiorze testowym w zależności od zastosowanego wariantu przeciwdziałania przeuczeniu.

Zbiory danych użyte w testach:

- `multimodal-sparse`,
- `rings5-sparse`,
- `rings3-balance`,
- `xor3-balance`.

7.2. Opis wykonanej pracy i wyniki eksperymentów

W ramach zadania zaimplementowano mechanizmy **regularyzacji wag** oraz **zatrzymywania uczenia** (early stopping) w celu przeciwdziałania przeuczeniu sieci neuronowej.

7.2.1. Regularyzacja wag

Do implementacji regularyzacji zastosowano metody **L1** i **L2**. Regularyzacja jest stosowana podczas obliczania gradientów w procesie *wstecznej propagacji błędu*. Funkcja `regularization`

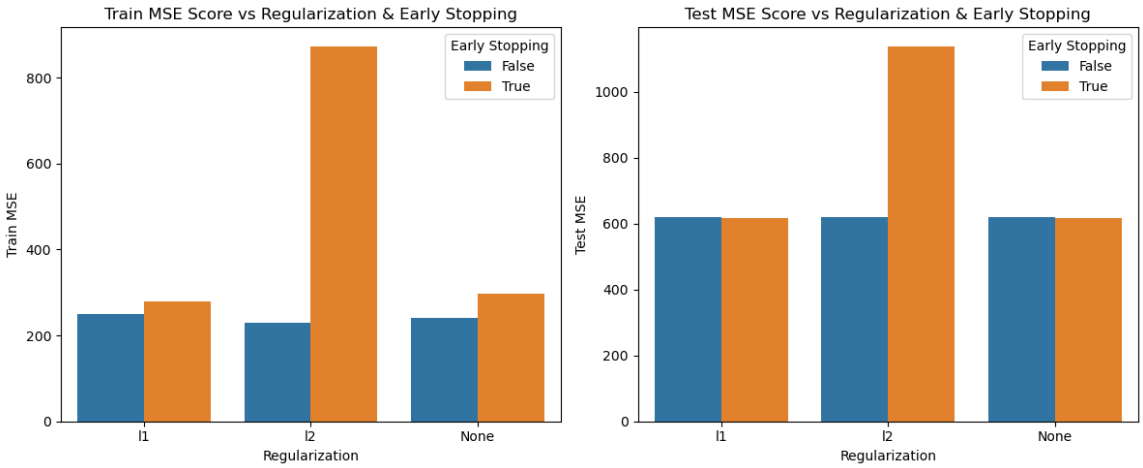
modyfikuje gradienty wag, co zapobiega ich nadmiernemu wzrostowi, a tym samym przeuczeniu modelu.

7.2.2. Zatrzymywanie uczenia (Early Stopping)

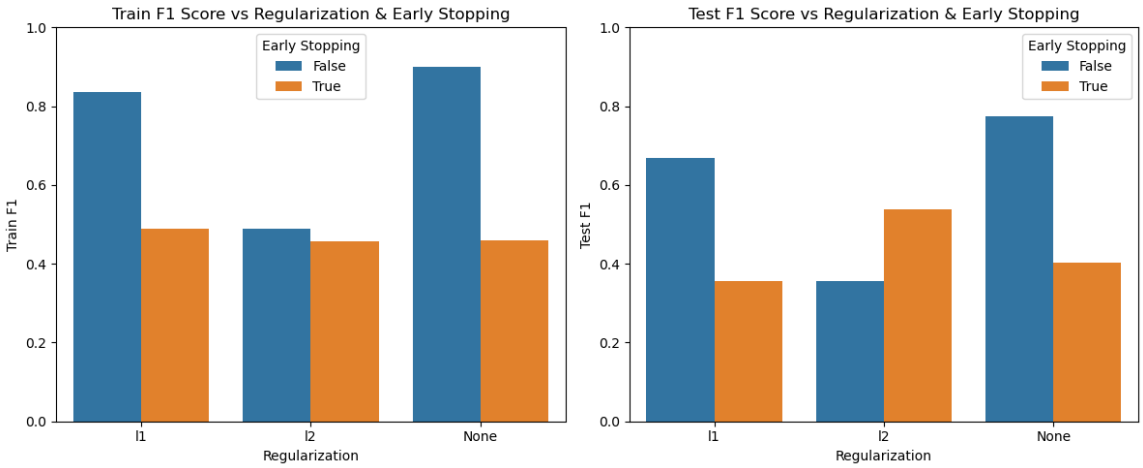
Zaimplementowano mechanizm *early stopping*, który zatrzymuje uczenie, gdy błąd na zbiorze walidacyjnym przestaje maleć przez określoną liczbę epok. Po zatrzymaniu treningu, wagi modelu są przywracane do najlepszych uzyskanych wartości.

7.2.3. Eksperymenty

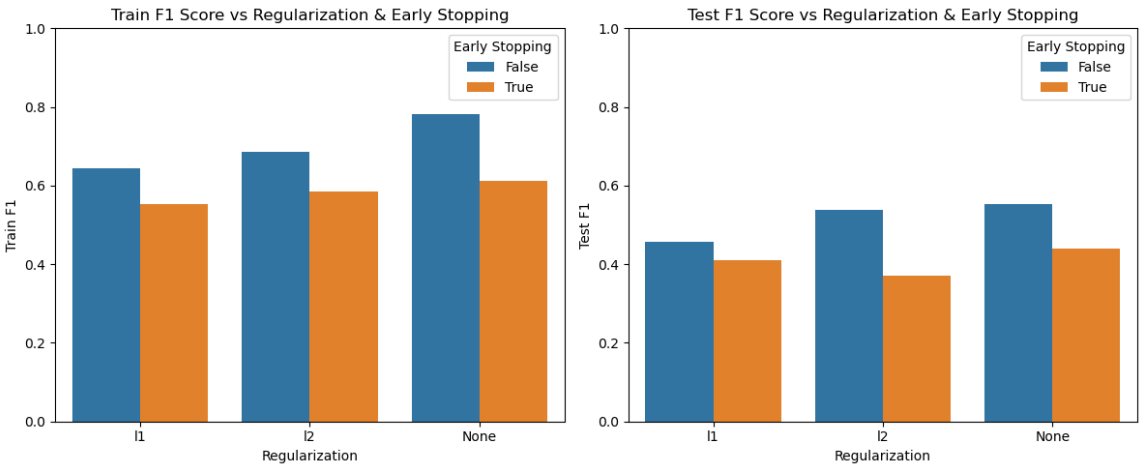
Przeprowadzono eksperymenty na czterech zestawach danych: `multimodal-sparse`, `rings5-sparse`, `rings3-balance` oraz `xor3-balance`, porównując skuteczność zastosowanych mechanizmów przeciwdziałania przeuczeniu. Podsumowane są one na wykresach 7.1 - 7.4.



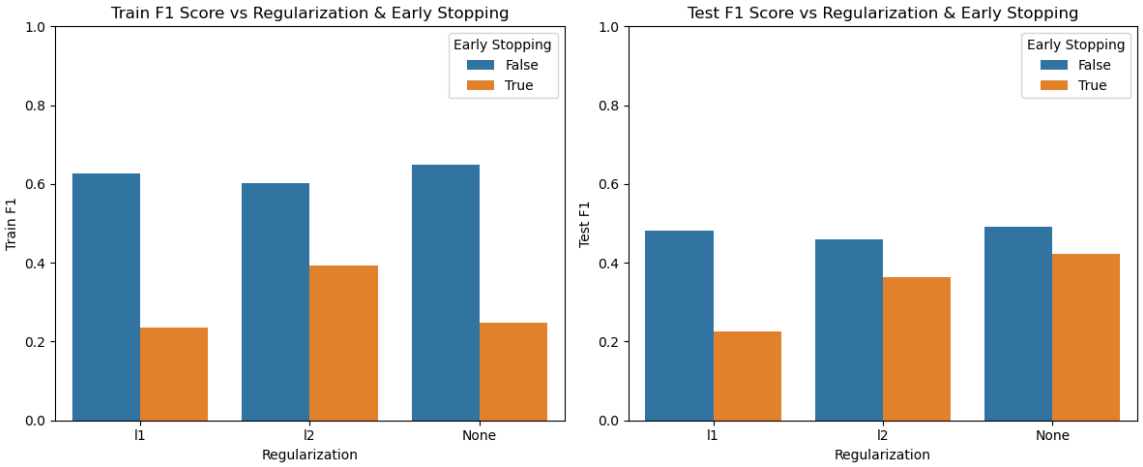
Rysunek 7.1: Zagregowane wyniki dla zbioru multimodal-sparse



Rysunek 7.2: Zagregowane wyniki dla zbioru xor3-balance



Rysunek 7.3: Zagregowane wyniki dla zbioru rings5-sparse



Rysunek 7.4: Zagregowane wyniki dla zbioru rings3-balance

7.3. Wnioski

Na podstawie analizy wyników przedstawionych na wykresach można sformułować kilka istotnych obserwacji dotyczących wpływu regularyzacji oraz zastosowania mechanizmu *early stopping* na jakość modeli.

- **Wpływ regularyzacji na jakość modeli:**
  - W przypadku zbioru *multimodal-squares* (Rys. 7.1), zastosowanie regularyzacji typu L2 lekko poprawiło wyniki MSE, zwłaszcza bez użycia *early stopping*. Regularyzacja L1 i brak regularyzacji byli mniej skuteczni.
  - Dla zbioru *xor3-balance* (Rys. 7.2), najlepsze wyniki osiągnięto bez użycia regulary-

zacji. Regularyzacja L2 doprowadziła do znacznego pogorszenia wyniku.

- W zbiorze *rings5-sparse* (Rys. 7.3), obserwowany był podobny trend – brak regularyzacji wyszedł najkorzystniej.

- **Wpływ *early stopping*:**
  - Zastosowanie *early stopping* nie przyniosło jednoznacznej poprawy – w niektórych przypadkach obserwowano za to spadek jakości modelu.
  - W przypadku zbioru `xor3-balance` po użyciu *early stopping* wyniki okazały się być znacząco gorsze niż bez.
  - Może to sugerować, że parametr *early stopping* został ustawiony zbyt restrykcyjnie (np. zbyt niski próg – około 100 epok), przez co model nie zdążył odpowiednio nauczyć się wzorców. W przyszłości warto rozważyć zwiększenie liczby epok lub bardziej elastyczne podejście do zatrzymywania treningu.

- **Równowaga między niedouczeniem a przeuczeniem:**
  - Wyniki wskazują, że zbyt silna regularyzacja (np. L2) może prowadzić do niedouczenia modelu, szczególnie gdy dodatkowo stosowany jest *early stopping*.
  - Z kolei brak regularyzacji i brak *early stopping* może prowadzić do przeuczenia, co widoczne było w wyższych wartościach MSE dla niektórych konfiguracji.

Podsumowując, najlepsze rezultaty w większości przypadków osiągnięto bez użycia regularyzacji i bez *early stopping*. Warto jednak przeprowadzić dalsze eksperymenty z lepiej dopasowanymi parametrami *early stopping*, gdyż jego potencjał w zapobieganiu przeuczeniu nie został w pełni wykorzystany.