Raport: Sieci MLP

Tymoteusz Urban 320665

15 kwietnia 2024

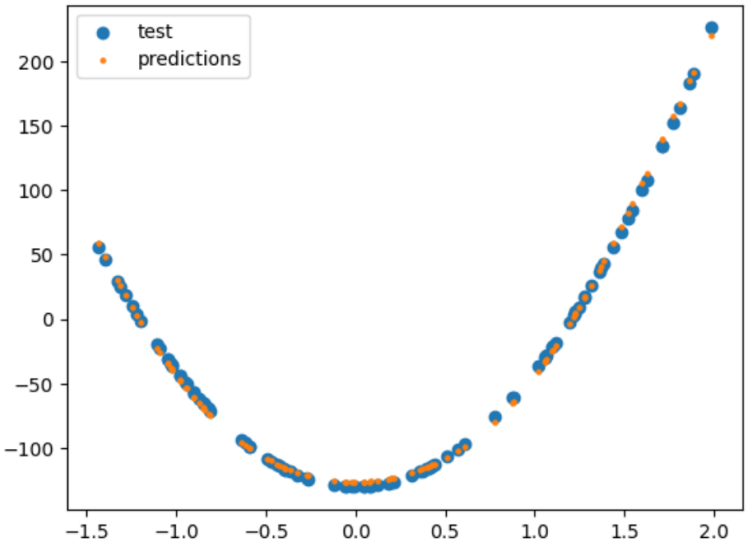
1 Wstęp

W niniejszym raporcie przedstawione zostały wyniki i analiza modelu perceptronu wielowarstwowego. Model sieci został zaimplementowany od zera, bez użycia gotowych bibliotek implementujących rozwiązania typu DL i ML. Celem było zrozumienie działania sieci neuronowych i wpływu hiperparametrów na proces uczenia.

2 NN1

Pierwsze zadanie polegało na stworzeniu modelu sieci MLP w której można ustawić liczbę warstw, liczbę neuronów, wagi (w tym biasy), a następnie na ręcznym dopasowaniu wag, tak by sieć dobrze estymowała zadane funkcje. Na warstwach ukrtych użyta została funkcja sigmoidalna, a na ostatniej warstwa liniowa.

2.1 Funkcja kwadratowa



Rysunek 1: Zbiór square-simple i predykcje dla architektury [1,5,1].

Intuicyjnie, dla $x < 0$ chcielibyśmy otrzymać coś podobnego do $sigm(-x)$, a dla $x > 0$ do $sigm(x)$. Potrzebujemy jedynie fragmentów, w których funkcja sigmoidalna zaczyna wzrastać (wtedy ma kształt podobny do funkcji kwadratowej). Po serii eksperymentów udało się ustalić wektory wag i biasów (zależne od współczynników a, b, c, d), przy użyciu których da się stworzyć funkcję zbliżoną do zbioru square-simple dla architektury z 1 warstwą ukrytą z 5 neuronami:

$weights = [[[a, -a, 0, 0, 0]], [[b], [b], [0], [0], [0]]]$
$biases = [[-c, -c, 0, 0, 0], [-d]],$

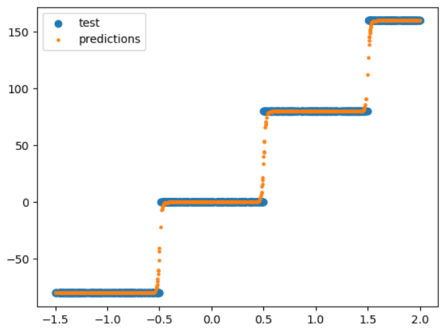
gdzie a odpowiada za nachylenie, b za skalowanie, c wyznacza punkt rozpoczęcia nachylania, d to przesunięcie. Następnie, przy odpowiednim ustawieniu wartości tych 4 współczynników, udało mi się uzyskać MSE równe 8.586: $a = 2, b = 600, c = 3.4, d = 165$.

Można zauważyć, że gdy udało nam się dostać MSE < 9 dla warstwy z 5 neuronami, to możemy tych wag użyć dla architektury z 10 neuronami, a pozostałe ustawić na 0. Wtedy te dodatkowe neurony nie będą jakkolwiek wpływać na wyniki, i otrzymamy bliźniaczą sieć do tej z 5 neuronami.

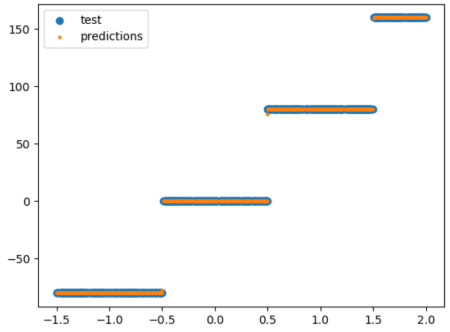
2.2 Funkcja schodkowa

Jeśli chodzi o funkcję schodkową, możemy sobie ją wyobrazić jako złożenie trzech funkcji sigmoidalnych, odpowiednio przesuniętych i przeskalowanych. Początkowo ustawiłem pierwsze 3 wagi na 100, a biasy na 50, -50 i -150 w celu stworzenia przesunięć funkcji sigmoidalnej. W drugim wektorze wag pierwsze 3 wagi ustawiłem na 80 żeby uzyskać odpowiedni przedział wartości, a ostatnim biasem przesunąłem wszystko w dół o 80.

$weights = [[[100, 100, 100, 0, 0]], [[80], [80], [80], [0], [0]]]$
$biases = [[50, -50, -150, 0, 0], [-80]]$



(a) Sieć z pierwotnymi wartościami wag.



(b) Sieć z odpowiednio zwiększonymi wartościami wag.

Takie wartości jednak nie wystarczały do uzyskana odpowiednio małego MSE. Zwiększyłem zatem znacząco wartości w pierwszym wektorze wag i biasów, żeby moje złożenie funkcji sigmoidalnych było bardziej kanciaste, a co za tym idzie lepiej dopasowane do zbioru steps large. Uzyskane MSE wynosiło 0.019.

3 NN2

Zadanie drugie polegało na zaimplementowaniu mechanizmu uczenia sieci korzystając z propagacji wstecznej. Należało przeprowadzić eksperymenty związane z różnymi wartościami hiperparametrów. Wszystkie dane przed treningiem były standaryzowane. Bez standaryzacji wektor y ma stosunkowo duże wartości, przez co sieć potrzebuje bardzo dużo epok, aby wagi się dostosowały do danych.

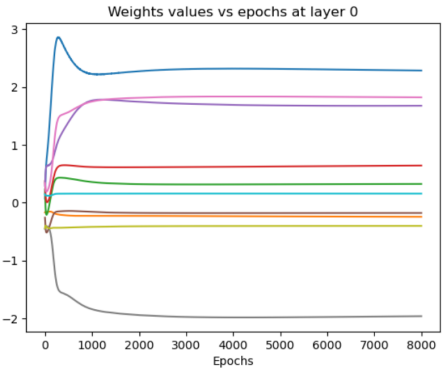
3.1 Kryteria MSE

Zbiór	warstwy	Inicjalizacja	Krok uczenia	Rozmiar porcji	Liczba epok	MSE
square simple	[1,10,1]	he	0.1	25	8000	2.545
steps small	[1,5,5,1]	he	1	100	1503	3.997
multimodal large	[1,10,10,1]	xavier	1	Pełne dane	59404	4.0

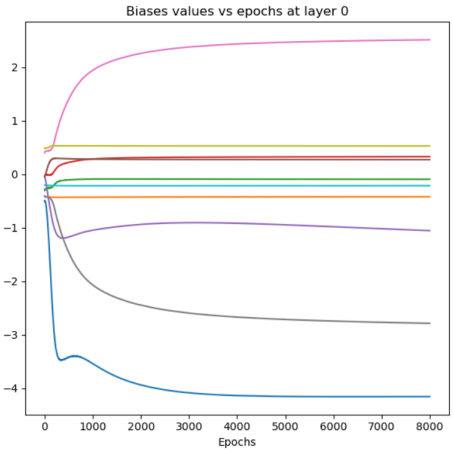
Tabela 1

3.2 Wizualizacja wag

W celu kontroli nauki sieci stworzona została funkcja przedstawiająca historię wartości wszystkich wag. Przykładowa wizualizacja dotyczy wspomnianej wcześniej architektury dla zbioru square simple. Na jednym wykresie widoczne są tylko wagi z jednej wybranej warstwy.



(c) Wagi na pierwszej warstwie.



(d) Biasy na pierwszej warstwie.

Widać, że jest zbieżność. Na początku treningu wartości wag się mocno zmieniają, jednak z czasem się stablizują, a zmiany są bardzo nieznaczne.

3.3 Rozmiar batcha

W celu przetestowania jak rozmiar batcha wpływa na zbieżność sieci, obliczyłem czas potrzebny do uzyskania MSE mniejszego równego 5. Eksperyment był przeprowadzany na zbiorze square simple i architekturze [1, 10, 1], 'xavier', lr=0.1:

size 1	Epoch 480	Loss = 5.357	Test Loss = 4.84	7.0 s
size 5	Epoch 519	Loss = 4.414	Test Loss = 4.713	0.9 s
size 10	Epoch 1740	Loss = 4.255	Test Loss = 4.993	2.0 s
size 25	Epoch 5622	Loss = 4.375	Test Loss = 4.987	4.5 s
size 50	Epoch 27252	Loss = 3.861	Test Loss = 4.938	18.7 s
size full	Epoch 19899	Loss = 3.773	Test Loss = 5.0	3.0 s

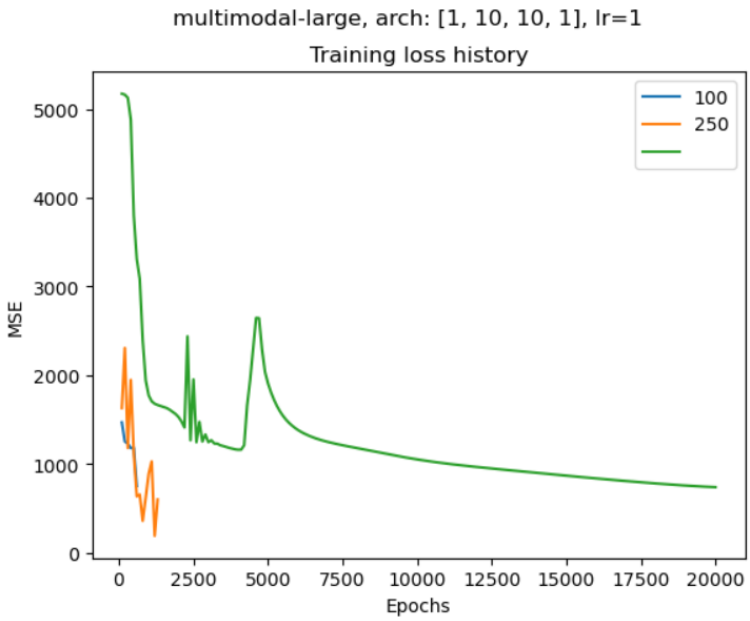
Tabela 2: Wyniki eksperymentu dla zbioru square simple

Widzimy, że w większości przypadków im mniejszy rozmiar batcha, tym mniej epok jest potrzebnych do osiągnięcia założonego MSE. Oczywiście to co nas interesuje to czas osiągnięcia celu. Dla tego zbioru i architektury najlepiej spisują się rozmiary 5, 10, i pełny rozmiar danych (bez dzielenia na porcje). Kiepskie wyniki z kolei daje dzielenie zbioru na bardzo duże części, albo liczenie gradientu na pojedynczych próbkach. To ostatnie podejście może skutkować trudnościami z uogólnieniem funkcji i brak odporności na outliery.

W przypadku zbioru multimodal jest bardzo zdecydowana przewaga mniejszych batchów nad brakiem tworzenia batchy. Po ponad 100 sekundach i 20000 epokach algorytm bez dzielenia danych na porcje nie był nawet blisko zakładanego poziomu MSE, podczas gdy sieci z batchami osiągnęły ten poziom dużo szybciej.

size 100	Epoch 649	Loss = 84.222	Test Loss = 78.77	51.8 s
size 250	Epoch 1352	Loss = 66.941	Test Loss = 61.55	102.4 s
size full	-	-	-	105.5 s

Tabela 3: Wyniki eksperymentu dla zbioru multimodal large



Rysunek 2: Historia uczenia

3.4 Inicjalizacja

Sprawdziłem także jak różne inicjalizowanie wag może wpływać na zbieżność. Dla każdego typu inicjalizacji sieć była tworzona i trenowana po 5 razy. Następnie zostało obliczone średnie MSE po 8 tysiącach epok.

Architektura: square simple, [1,10,1], lr=0.1, batch size = 25:

xavier	3.55
he	2.78
uniform	3.76

Tabela 4: średnie MSE po 8 tys. epok

Widzimy, że tutaj najlepiej spisuje się algorytm 'he'. Zobaczmy co się dzieje dla zbioru steps large.

Architektura: steps large, [1,5,5,1], lr=1, batch size = 50:

xavier	38.88
he	37.53
uniform	79.69

Tabela 5: średnie MSE po 200 epokach

Tutaj 'he' i 'xavier' wypadają w zasadzie identycznie. Widzimy, że te metody inicjowania wag są o wiele wydajniejsze, niż używanie do tego rozkładu jednostajnego.

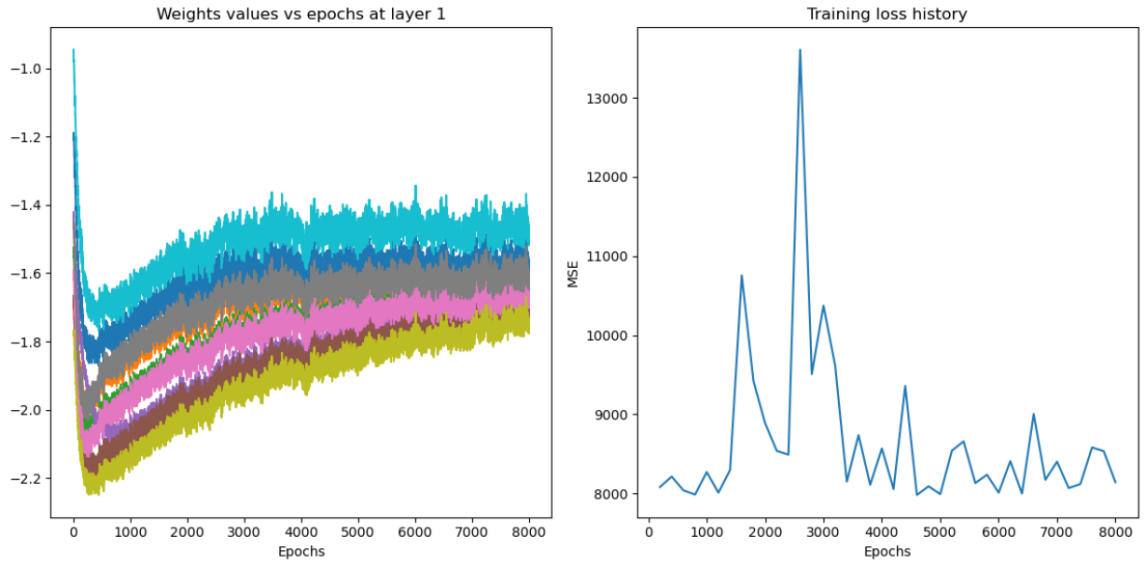
3.5 Learning rate

Ostatnim eksperymentem było sprawdzenie zachowania się sieci przy różnych krokach uczenia. Eksperyment został przeprowadzony na takiej samej zasadzie jak poprzednie.

krok uczenia	MSE
0.001	5077.19
0.01	76.0
0.1	3.08
1	2229.39

Tabela 6: średnie MSE po 8 tys. epok

Dla zbioru square-simple 0.1 jest najlepszą wartością learning rate. Dla 0.01 też jest zbieżność, ale zajęło by to nieco dłużej. Z kolei learning rate równy 1 jest za duży aby być zbieżny, wagi i funkcja straty nie potrafią się ustabilizować. Możemy to zwizualizować:



Rysunek 3: Historia wag i funkcji straty dla zbioru square simple i kroku uczenia równego 1.

4 NN3

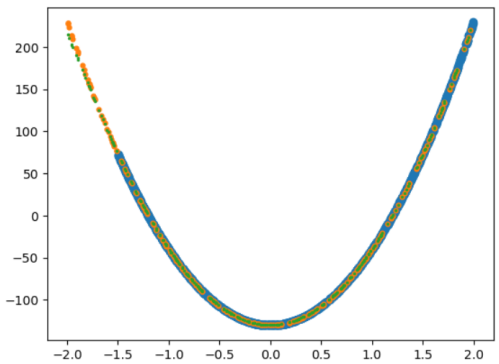
W tym zadaniu należało zaimplementować usprawnienia treningu sieci - uczenie z momentem i normalizację gradientu. Obie metody powinny znacząco przyspieszać zbieganie sieci.

4.1 Kryteria MSE

Zbiór	warstwy	Krok uczenia	Rozmiar porcji	Metoda	MSE
square large	[1,10,1]	0.1	50	rmsprop	0.997
steps-large	[1,5,5,1]	0.2	10	rmsprop	2.473

Tabela 7

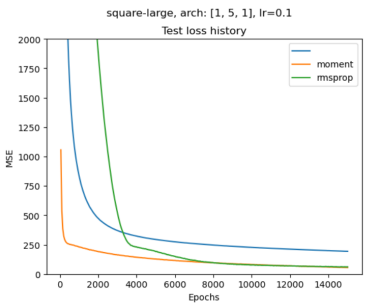
W przypadku square-large zbiór testowy różnił się od treningowego. Parabola była znacząco wydłużona. Przez to osiągnięcie MSE mniejszego od 1 było bardzo czasochłonne (ponad 55 tysięcy epok), podczas gdy funkcja straty na zbiorze treningowym była mniejsza od 1 już po 1700 epokach. Na zbiorze multimodal-large już wcześniej zostało uzyskane oczekiwane kryterium.



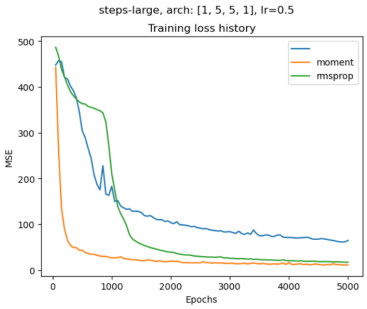
Rysunek 4: Square-large, zbiory treningowy i testowy.

4.2 Eksperymenty

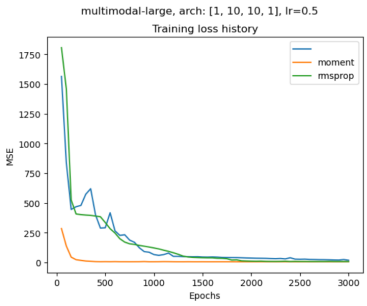
Eksperymenty polegały na trenowaniu różnych wariantów sieci na różnych zbiorach danych.



(a)



(b)



(c)

Widzimy, że uczenie z momentem daje najlepsze wyniki (i najszybciej). Normalizacja gradientu w ogólności też daje lepsze wyniki niż zwykła propagacja wsteczna, ale dopiero od pewnego momentu.

5 NN4

Następnym zadaniem było dodanie funkcji aktywacji softmax, w celu przeprowadzania klasyfikacji. Softmax jest używana oczywiście jedynie na ostatniej warstwie, a żeby wszystko dobrze działało to przeprowadzany jest także one hot encoding na zmiennej objaśnianej.

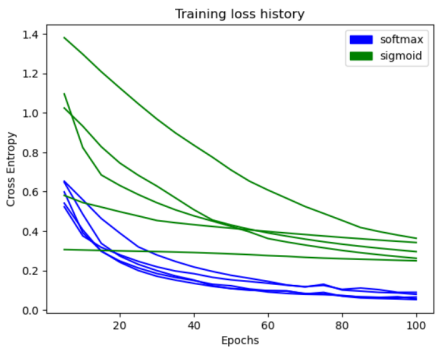
5.1 Kryteria miary F1

Zbiór	warstwy	Krok uczenia	Rozmiar porcji	Metoda	Liczba epok	F1 score
easy	[2,2,2]	0.1	25	-	12	0.998
rings3 regular	[2,5,5,3]	0.01	125	moment	1123	0.752
xor	[2,16,16,2]	0.001	125	moment	$>$ 100000	0.93

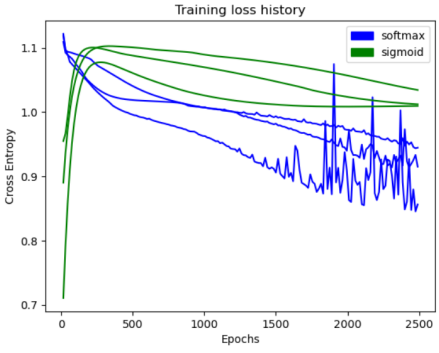
Tabela 8

5.2 Eksperymenty

Przeprowadzone eksperymenty polegały na porównaniu szybkości i skuteczności uczenia przy korzystaniu na ostatniej warstwie z softmax i funkcji sigmoidalnej.



(d) Klasyfikacja binarna na zbiorze easy.



(e) Klasyfikacja wieloklasowa na zbiorze rings3.

Widzimy, że softmax radzi sobie o wiele lepiej, zbieżność jest o wiele szybsza. W klasyfikacji wieloklasowej początkowe wartości cross entropii mogą się wydawać dziwne (małe wartości dla sigmoidalnej), ale wynikają one z tego, że wartości w softmax zawsze się sumują do 1, a w sigmoidzie nie. Przez to może zdarzyć się sytuacja, że dla sigmoidy wartości wszystkich neuronów na ostatniej warstwie będą duże, a wtedy wartość entropii krzyżowej będzie mała (pomimo tego, że wartości neuronów będą do siebie podobne i klasy będą nierozróżnialne). W

przypadku softmax taka sytuacja nie może wystąpić, gdyż prawdziwy neuron będzie miał dużą wartość wtedy i tylko wtedy gdy pozostałe będą małe. Przykładowo:

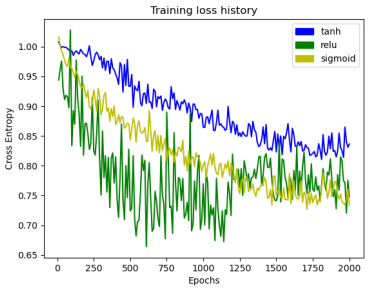
```
in [1]: y_true = np.array([[0,0,1]])
        y_pred_sig = np.array([[0.8, 0.8, 0.8]])
        print("sigmoid cross entropy:", cross_entropy(y_true, y_pred_sig))
        y_pred_soft = np.array([[0.25, 0.25, 0.5]])
        print("softmax cross entropy:", cross_entropy(y_true, y_pred_soft))
out [1]: sigmoid cross entropy: 0.2231435513142097
        softmax cross entropy: 0.6931471805599453
```

Sigmoid ma mniejszą wartość cross entropii, choć klasy są nierozróżnialne w przeciwieństwie do softmax, gdzie cross entropia jest wyższa, ale klasyfikator dobrze przyporządkowuje do klasy. Najprawdopobniej taka sytuacja ma miejsce w początkowym etapie uczenia naszej sieci.

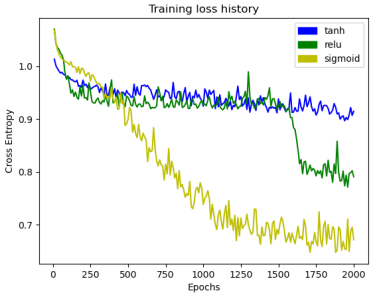
6 NN5

W tej części należało dodać i wytestować różne funkcje aktywacji. W ramach eksperymentów zostało przygotowane 36 wykresów. Dla każdego zbioru zostało wytestowane zachowanie sieci przy korzystaniu z różnych funkcji aktywacji (sigmoidalna, tanh, ReLU) dla jednej, dwóch i trzech warstw ukrytych.

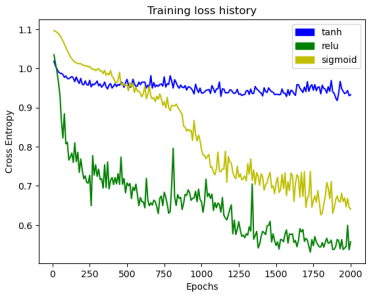
6.1 Klasyfikacja



(f) 1 warstwa ukryta

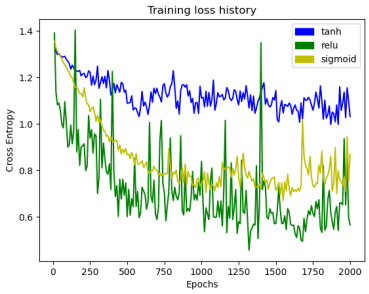


(g) 2 warstwy ukryte

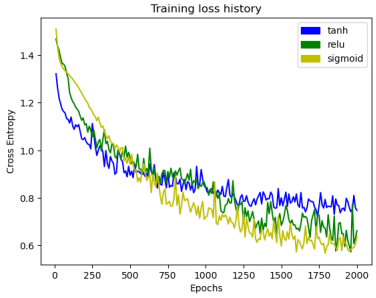


(h) 3 warstwy ukryte

Rysunek 5: **Rings 3 regular** - warstwy ukryte rozmiaru 5, learning rate = 0.01, batch size = 125, moment



(a) 1 warstwa ukryta



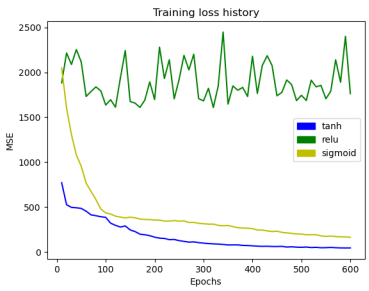
(b) 2 warstwy ukryte

(c) 3 warstwy ukryte

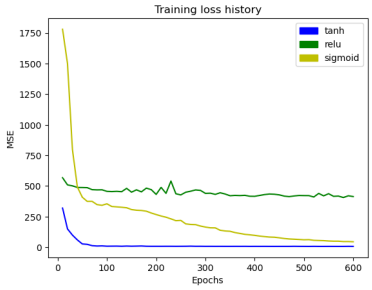
Rysunek 6: **Rings 5 regular** - warstwy ukryte rozmiaru 10, learning rate = 0.01, batch size = 125, moment

Co pierwsze rzuca się w oczy, to że przy małym rozmiarze sieci funkcja straty jest bardziej niestabilna i są dużo większe skoki funkcji straty. Widzimy, że przy klasifykacji najlepiej sprawdzają się funkcja sigmoidalna i ReLU. Użycie tangensa hiperbolicznego czasami powoduje, że funkcja straty się zupełnie nie zmniejsza i jest bliska stałej.

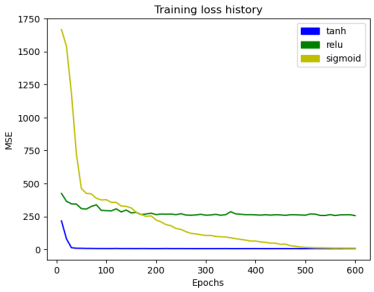
6.2 Regresja



(a) 1 warstwa ukryta

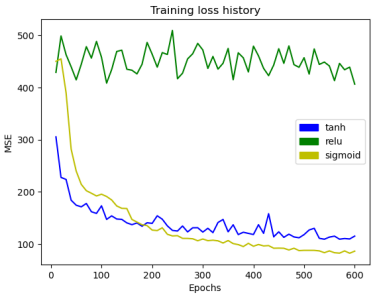


(b) 2 warstwy ukryte

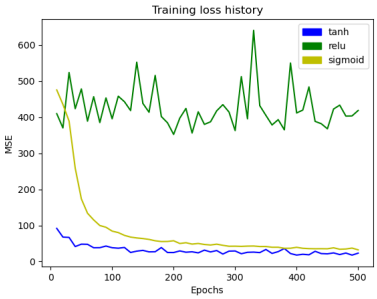


(c) 3 warstwy ukryte

Rysunek 7: **Multimodal large** - warstwy ukryte rozmiaru 10, learning rate = 0.1, batch size = 125, moment



(a) 1 warstwa ukryta



(b) 2 warstwy ukryte

(c) 3 warstwy ukryte

Rysunek 8: **Steps large** - warstwy ukryte rozmiaru 5, learning rate = 0.01, batch size = 25, moment

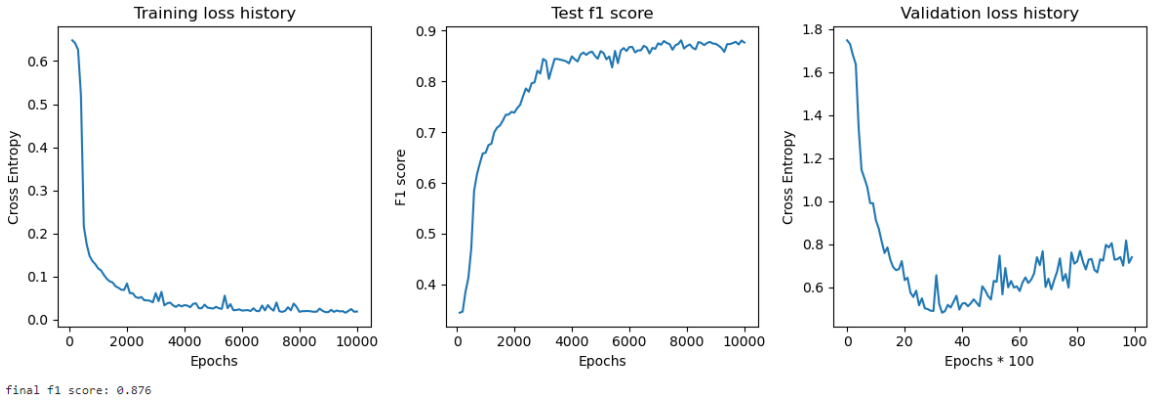
Odwrotnie jest w przypadku regresji. Tutaj zdecydowanie najlepiej wypada tangens hiperboliczny. W większości przypadków uzyskuje najmniejsze MSE i robi to najszybciej (przy użyciu funkcji sigmoidalnej czasami da się uzyskać lepszy wynik niż przy tanh, ale po pewnej liczbie epok). Funkcja ReLU jak widzimy zupełnie się do tego problemu nie nadaje.

7 NN6

7.1 Metody regularyzacji

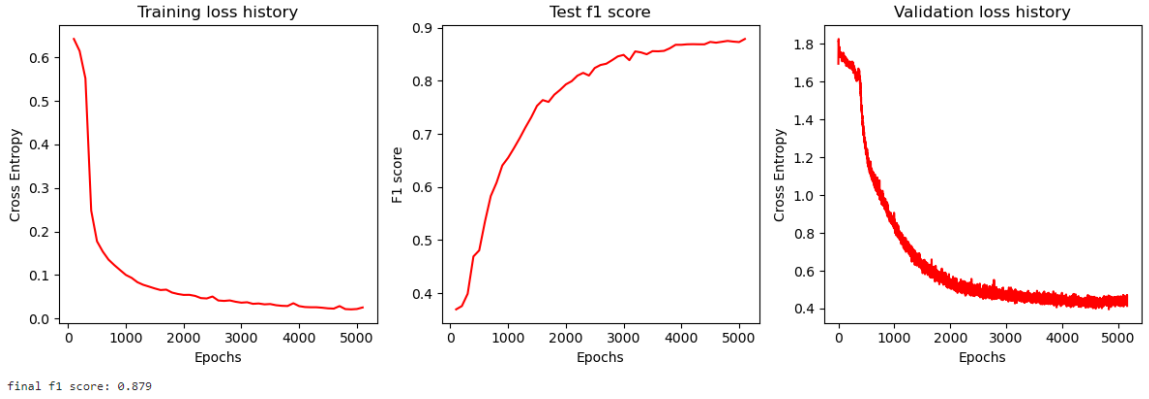
W tej sekcji dla każdego zbioru porównywane są wyniki i funkcje straty sieci dla modeli bez regularyzacji, z early stoppingiem na zbiorze walidacyjnym i z regularyzacją L2. W eksperymentach używany współczynnik regularyzacji wynosi 0.001.

Rings 3



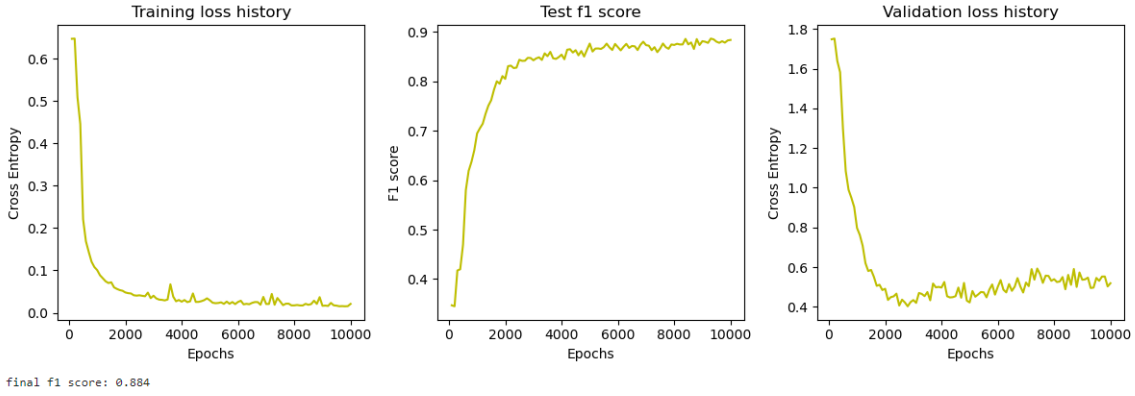
Rysunek 9: Model bez regularyzacji

Widzimy, że po około 6500 epokach f1 score przestaje wzrastać, z kolei funkcja straty na zbiorze walidacyjnym zaczyna wzrastać już od około 4000 epoki.



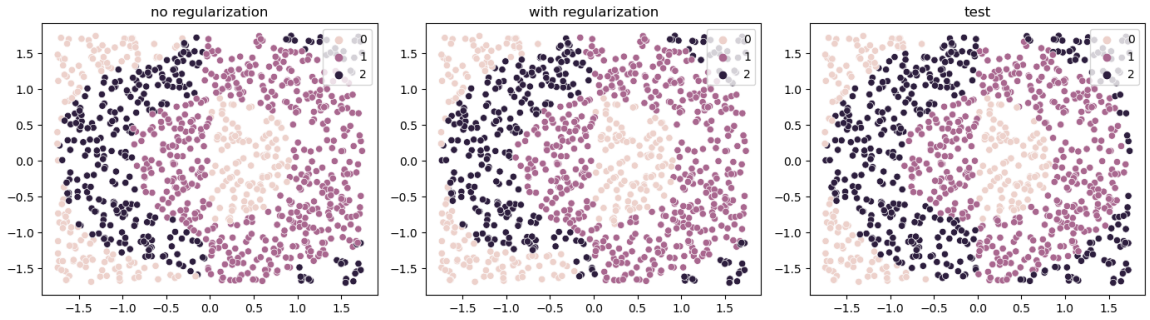
Rysunek 10: Model z early stoppingiem

F1 score jest w zasadzie identyczny, jak w przypadku treningu bez early stoppingu, jednakże uzyskaliśmy ten wynik bez zbędnego trenowania kilku tysięcy epok.



Rysunek 11: Model z regularyzacją L2

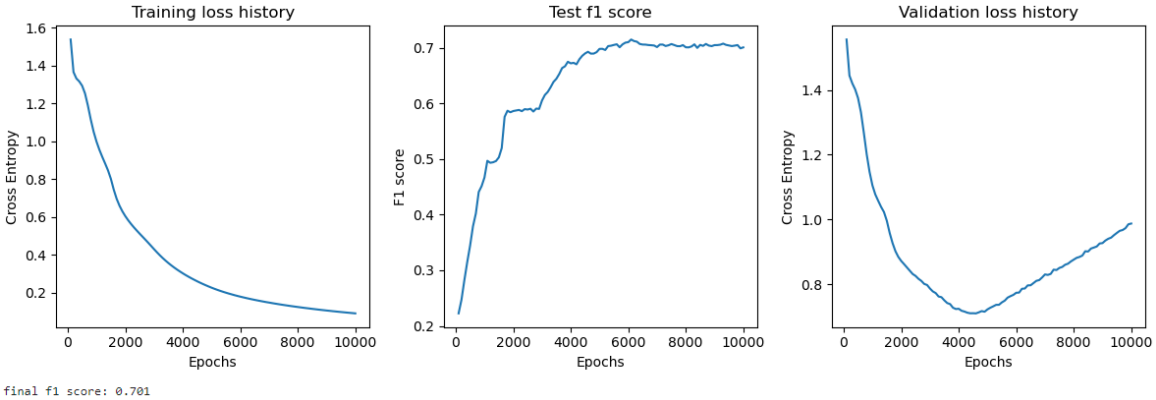
Wykresy metryk wyglądają w zasadzie identycznie jak bez regularyzacji. Zobaczmy jak wygląda podział klas:



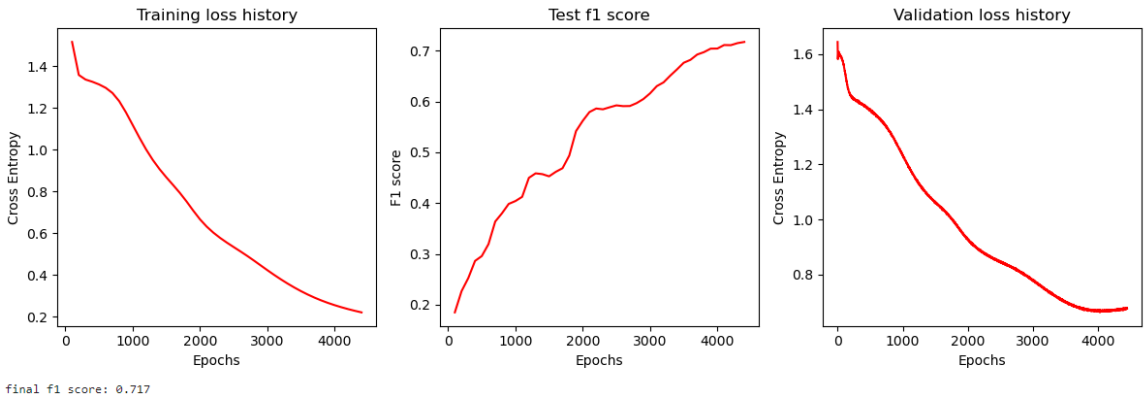
Rysunek 12: Predykcje zbiorze testowym

Nie widać jakichś szczególnych, znaczących różnić z których możnaby wyciągnąć jakieś wnioski. W niektórych miejscach model bez regularyzacji robi lepsze predykcje, a w niektórach z regularyzacją, a F1 score jest tylko odrobinę wyższy dla regularyzacji.

Rings 5

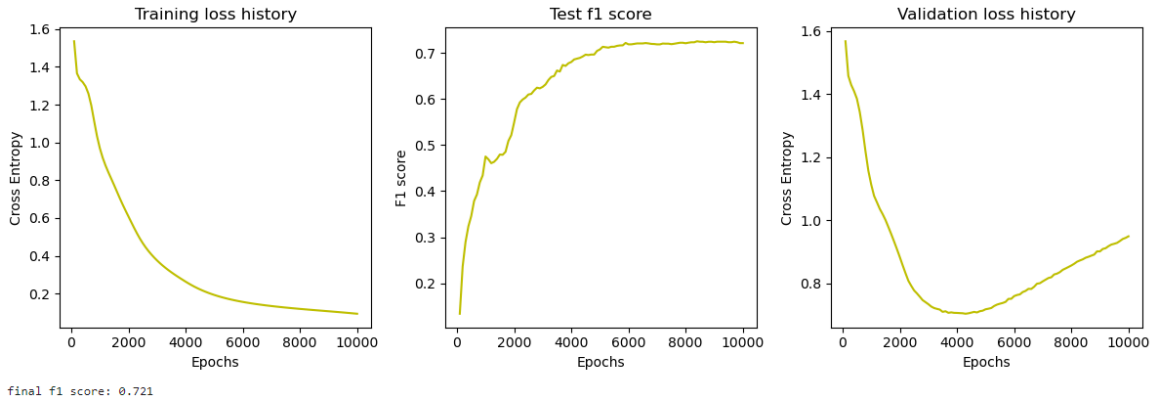


Rysunek 13: Model bez regularyzacji



Rysunek 14: Model z early stoppingiem

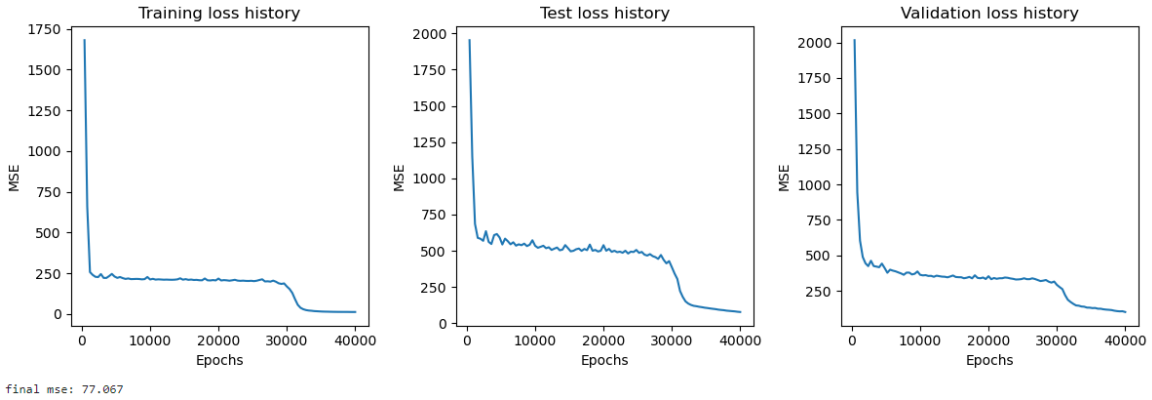
Uzyskaliśmy lepszy f1 score, dzięki temu że zatrzymaliśmy trening odpowiednio wcześnie.



Rysunek 15: Model z regularyzacją L2

Otrzymaliśmy najlepszy f1 score. Można zauważyć, że gdybyśmy zastosowali teraz early stopping, to trening zatrzymałby się około 4500 tysięcznej epoki, a w tym miejscu f1 score nie jest optymalny, więc early stopping nie zawsze daje najlepsze możliwe wyniki. Ostatecznie wszystkie metody poradziły sobie podobnie, jednakże z regularyzacją otrzymaliśmy najlepszy f1 score, z kolei z early stoppingiem trening trwał 3 razy szybciej.

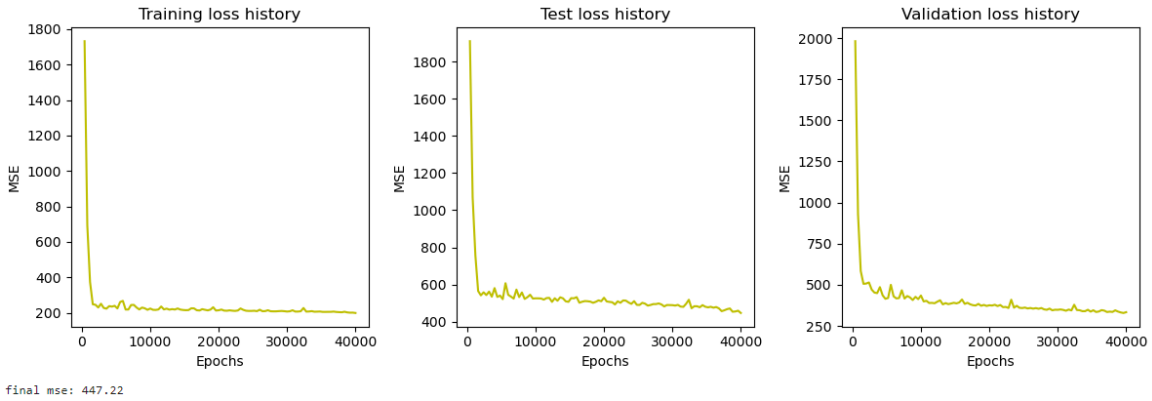
Multimodal large



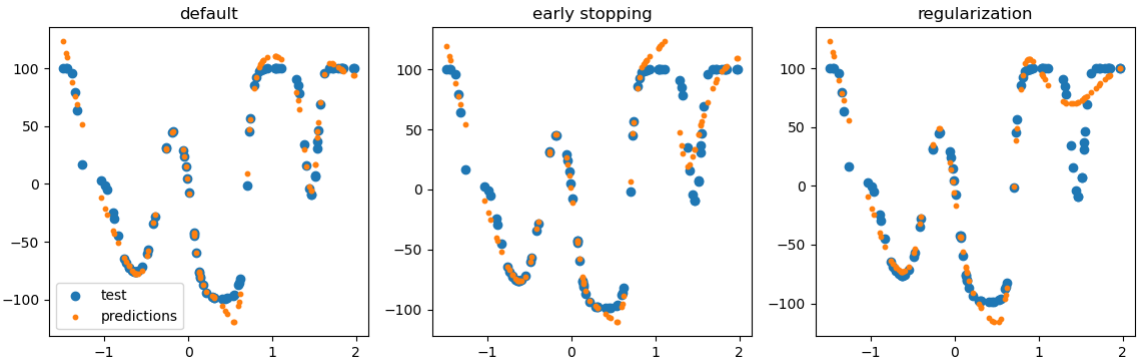
Rysunek 16: Model bez regularyzacji

Rysunek 17: Model z early stoppingiem

Widzimy, że early stopping nie zawsze się sprawdza. Pomimo tego, że długo nic się nie zmienia, to około 30000 epoki MSE drastycznie spada (w modelu bez regularyzacji). Early stopping jednak zatrzymuje trening nieco wcześniej, przez to nie dochodzi do optymalnej wartości.

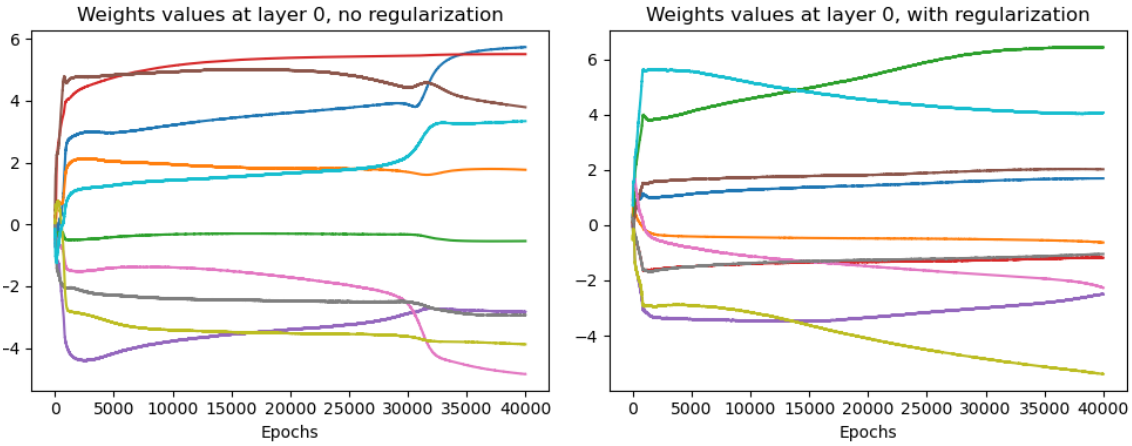


Rysunek 18: Model z regularyzacją L2



Rysunek 19: Predykcje na zbiorze testowym

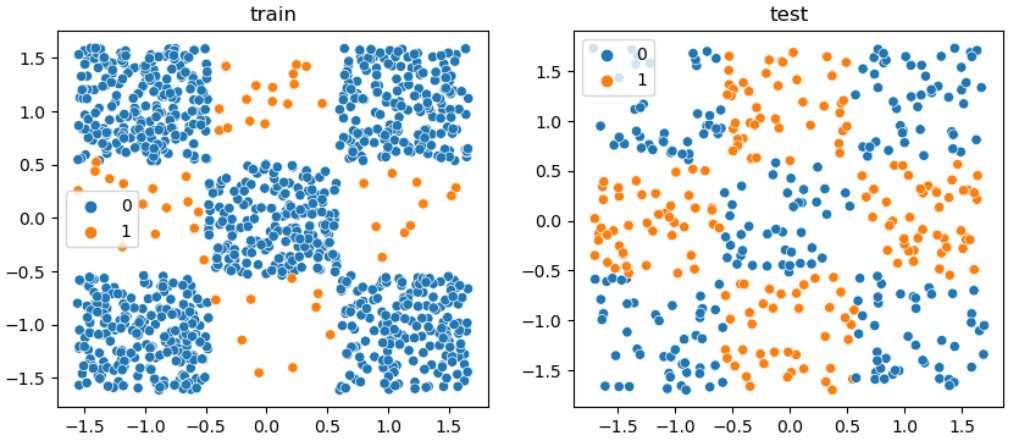
Niestety, regularyzacja nie sprawdza się w tym przypadku, a early stopping tylko utrudnił odpowiednie dopasowanie się sieci.



Rysunek 20: Historia wag na pierwszej warstwie

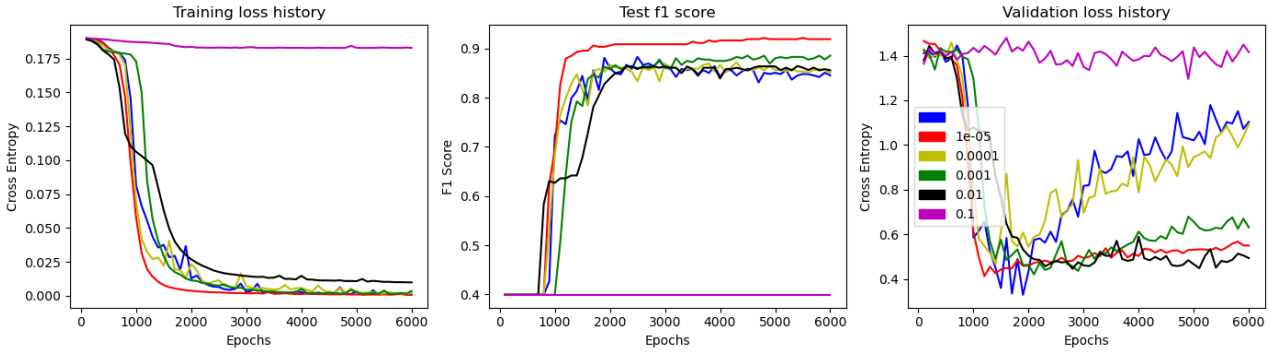
7.2 Regularyzacja L2

Dla zestawu danych **xor3** sprawdzimy różne wartości regularyzacji. Będziemy korzystać z architektury o warstwach [2,10,10,2], kroku uczenia 0.05, rozmiaru porcji danych 50 i optymalizacji z momentem.



Rysunek 21: **xor3** - zbiór treningowy i testowy

Na oko zależność w danych jest bardzo prosta, ale przez specyfikę zbioru testowego i treningowego zadanie nie jest takie trywialne. Potrzebujemy tutaj w pewien sposób uogólnić te zależności, gdyż jest wysokie ryzyko overfittingu. Może nam w tym pomóc regularyzacja.



Rysunek 22: Historia wag na pierwszej warstwie

Widzimy, że w przypadku gdy nasze dane potrzebują uogólnienia, najlepsze wyniki daje nieduża wartość współczynnika regularyzacji. Brak używania regularyzacji też daje dobre efekty, ale z czasem następuje zbyt mocne dopasowanie modelu do danych treningowych i przez to validation loss rośnie, a f1 score spada. Z kolei przy zbyt dużym współczynniku sieć przestaje się czegokolwiek uczyć.