Sieci neuronowe - sprawozdanie

Michał Gromadzki, 313356

1 Bazowa implementacja

1.1 Opis tematu laboratoriów

Na tych zajęciach należy zaimplementować sieć neuronową typu MLP, w której można ustawić: liczbę warstw, liczbę neuronów w każdej z warstw i wagi poszczególnych połączeń, w tym biasów. Sieć ma używać sigmoidalnej funkcji aktywacji. Na wyjściu dopuszczana jest funkcja liniowa.

Implementacja sieci musi być przygotowana w taki sposób, żeby łatwo zmieniać:

1. Architekturę, to znaczy liczbę wejść, wyjść, neuronów w warstwach ukrytych.
2. Funkcję aktywacji.

Tak przygotowaną implementację należy następnie wykorzystać do rozwiązania zadania regresji na dostarczonych danych. Parametry sieci należy dobrać ręcznie, tak aby uzyskać możliwie dobrze wyniki na zbiorach danych (zbudować po jednej sieci dla każdego zbioru):

1. square-simple
2. steps-large

Rozważyć architektury sieci:

1. jedna warstwa ukryta, 5 neuronów
2. jedna warstwa ukryta, 10 neuronów
3. dwie warstwy ukryte, po 5 neuronów każda

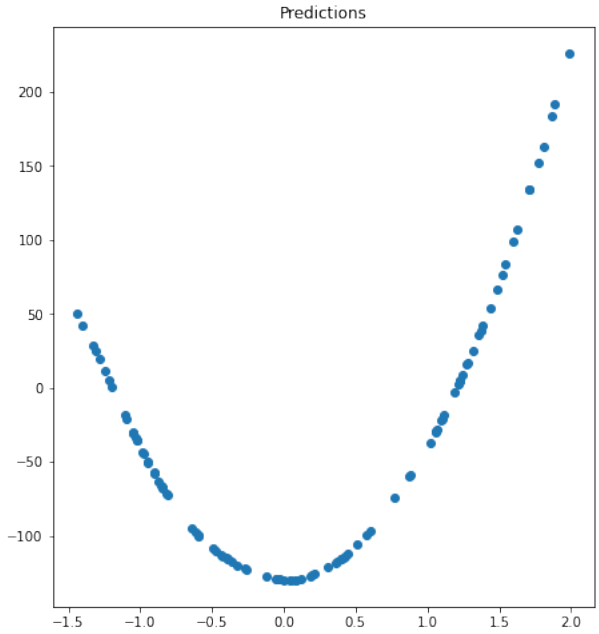
1.2 Opis wykonanej pracy i wyniki eksperymentów

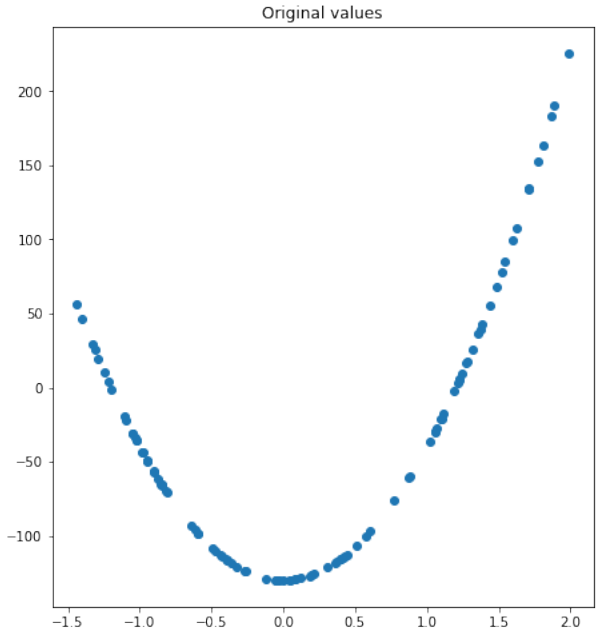
1.2.1 Wykonana praca

Zaimplementowanie funkcji liniowej i sigmoidalnej i dodanie ich do słownika w celu ułatwienia inicjalizacji sieci. W skład sieci będą wchodzić dwie klasy - Layer oraz NeuralNetwork. Klasa Layer zawiera wagi oraz biasy warstwy sieci, której odpowiada. Na razie wagi są inicjowane jako macierz jedynek a bias jako wektor zer. Dodatkowo posiada metodę feedforward pozwalającą na propagację w przód przez daną warstwę. Klasa NeuralNetwork będzie pozwalała na implementowania wszystkich pozostałych metod, na razie posiada jedynie metodę odpowiadającą za inicjalizacje odpowiednich klas Layer oraz metodę predict, która jest wywołaniem metody feedforward na kolejnych warstwach sieci. Podczas tworzenia klasy NeuralNetwork możemy wybrać wielkość wektora wejściowego, ilość warstw oraz liczbę neuronów i funkcję aktywacji w poszczególnej warstwie.

1.2.2 Eksperymenty

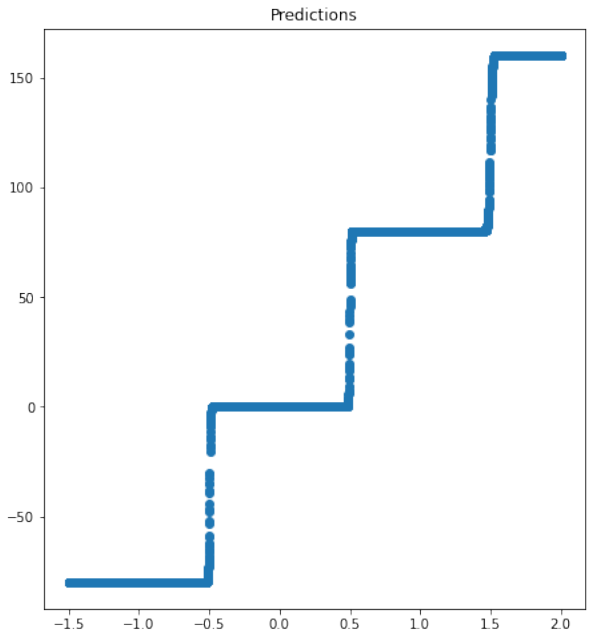
Sprawdzimy poprawność działania sieci dla jednej oraz dwóch ukrytych warstw, wagi zostały dobrane ręcznie. Dla zbioru square-simple została wykorzystana sieć z jedną warstwą ukrytą o 5 neuronach.

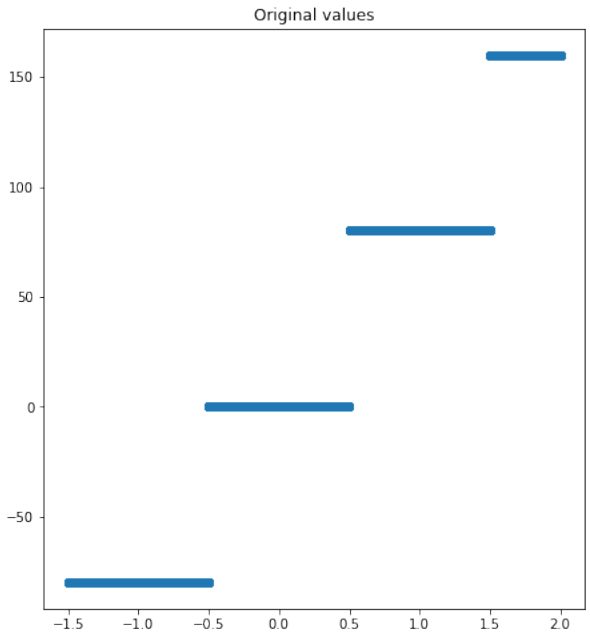




Otrzymaliśmy MSE równe 1.3.

Dla zbioru steps-large zostały wykorzystane 2 warstwy ukryte, obie o 5 neuronach.





Otrzymaliśmy MSE równe 8.9.

1.3 Wnioski na podstawie eksperymentów

Wykresy predykcji i oryginalnych wartości są wystarczająco podobne, aby stwierdzić, że propagacja w przód w zaimplementowanej sieci działa poprawnie na jednej oraz dwóch ukrytych warstwach.

2 Implementacja propagacji wstecznej błędu

2.1 Opis tematu laboratoriów

W ramach tego laboratorium trzeba zaimplementować uczenie sieci neuronowej propagacją wsteczną błędu. Aby sprawdzić implementację, należy wykonać uczenie na prostych danych do uczenia dostarczonych na zajęciach. Następnie należy zaimplementować metodę wizualizacji wartości wag sieci w kolejnych iteracjach i w przypadku gdy nie udaje się nauczyć sieci, spróbować wykorzystać te wizualizacje do ustalenia przyczyny problemu. Zaimplementować wersję z aktualizacją wag po prezentacji wszystkich wzorców i wersję z aktualizacją po prezentacji kolejnych porcji (batch). Porównać szybkość uczenia dla każdego z wariantów.

Inicjować wagi z rozkładu jednostajnego na przedziale [0,1]. Opcjonalnie zaimplementować inną metodą inicjowania wag. Albo metodę He albo Xavier.

Przetestować uczenie sieci na następujących zbiorach:

1. square-simple
2. steps-small
3. multimodal-large

2.2 Opis wykonanej pracy i wyniki eksperymentów

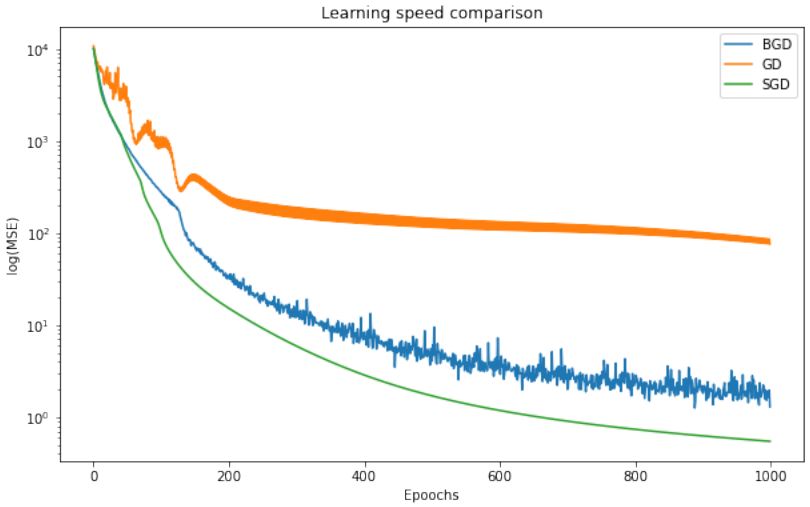
2.2.1 Wykonana praca

Zaimplementowanie gradientów funkcji aktywacji i dodanie ich do słownika. Zaimplementowanie funkcji straty MSE oraz jej gradientu. Klasa Layer teraz zapisuje wartości przed i po obliczeniu funkcji aktywacji, dodatkowo posiada metodę layer_grad, która zwraca gradient na danej warstwie. Klasa NeuralNetwork posiada metodę backpropagation, która na podstawie gradientu funkcji straty oraz gradientu kolejnych warstw zwraca ostateczny gradient. Metodę update_params która zmienia wartości wag i biasów na każdej warstwie. Dodatkowo zostały zaimplementowane trzy optimizery - Gradient Descent, Stochastic Gradient Descent oraz mini-batch Gradient Descent. Dodatkowo zostały zaimplementowane trzy sposoby inicjalizacji wag - z rozkładu jednostajnego, metoda He oraz metoda Xaviera.

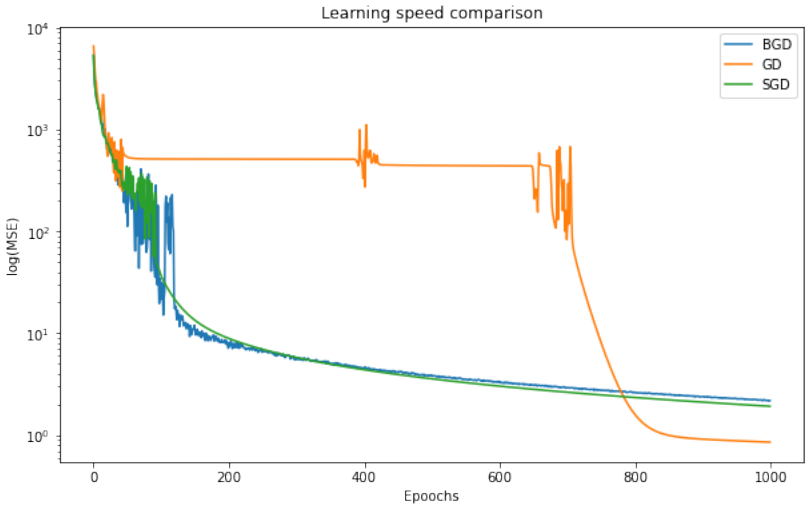
2.3 Wnioski na podstawie eksperymentów

2.3.1 Eksperymenty

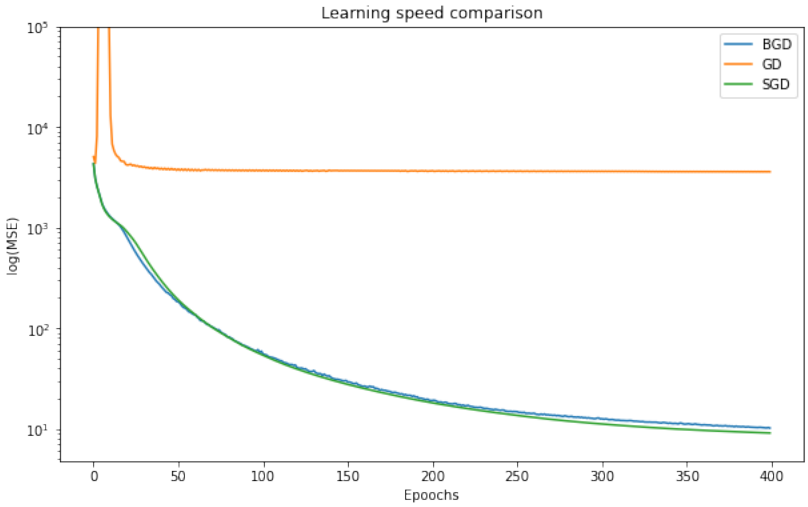
Przetestujemy szybkość uczenia każdego z optimizerów na każdym z 3 podanych zbiorów. Zaczniemy od zbioru square-simple, zostanie wykorzystana architektura z jedną warstwą ukrytą o 30 neuronach, a parametr learning_rate będzie równy 0.001, a batch_size = 32 dla mini-batch Gradient Descent.



Następnie sprawdzimy szybkość uczenia na zbiorze steps-large, wykorzystamy architekturę z jedną warstwą ukrytą o 5 neuronach, parametr learning_rate = 0.007, a batch_size = 32.



Ostatnim zbiorem będzie multimodal-large, wykorzystamy architekturę z jedną warstwą ukrytą o 50 neuronach, parametr learning_rate = 0.0001, a batch_size = 32.



Wyniki eksperymentów zostały zapisane w tabeli poniżej:

	Train set			Test set		
Datasets	GD	SGD	BGD	GD	SGD	BGD
square-simple	76.08	0.54	1.29	93.80	1.79	3.62
steps-small	0.85	1.91	2.18	94.45	101.2	96.37
multimodal-large	3219	9.49	10.15	3152	5.03	6.26

Warto zwrócić uwagę na, zbiór steps-small. Otrzymujemy małe MSE na zbiorze treningowym, lecz o wiele większe na testowym. Bardzo niewielki rozmiar sieci świadczy o braku możliwości przetrenowania sieci, rozbieżne wartości MSE mogą świadczyć o źle dobranych danych treningowych lub testowych.

2.3.2 Wnioski na podstawie eksperymentów

Na podstawie wyników eksperymentów możemy stwierdzić, że wszystkie optimizery zostały zaimplementowane poprawnie. Najlepiej sprawdzał się Stochastic Gradient Descent. Może to wynikać z niewielkiego rozmiaru zbiorów danych, dzięki największej częstotliwości zmiany parametrów osiągnęliśmy najmniejsze MSE, a mały rozmiar danych zagwarantował niewielki wzrost poświęconego czasu na trenowanie. Należy podkreślić, że wybór odpowiedniego optimizera zależy od danego zbioru danych oraz architektury sieci, jednak w większości większych zbiorów danych mini-batch gradient descent sprawdza się lepiej od pozostałych. Standardowa wielkość jednego batcha to z reguły 32, jednak przy większych zbiorach danych należy tą wartość zwiększyć.

3 Implementacja momentu i normalizacji gradientu

3.1 Opis tematu laboratoriów

Zaimplementować dwa usprawnienia uczenia gradientowego sieci neuronowej:

1. moment
2. normalizację gradientu RMSProp.

Porównać szybkość zbieżności procesu uczenia dla obu wariantów.

Przeprowadzić eksperymenty na zbiorach:

1. square-large
2. steps-large
3. multimodal-large

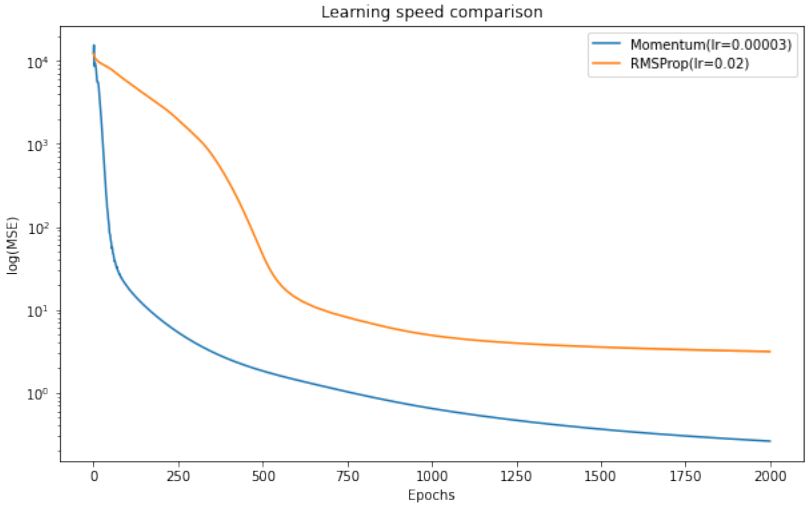
3.2 Opis wykonanej pracy i wyniki eksperymentów

3.2.1 Wykonana praca

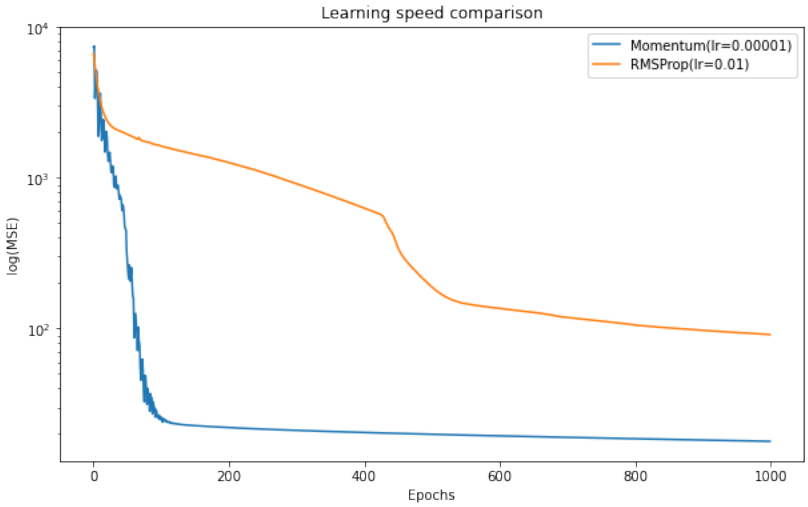
Zaimplementowanie momentu oraz normalizacja gradientu RMSProp zgodnie z wykładem.

3.2.2 Eksperymenty

Przetestujemy sprawność działania algorytmów na zbiorze square-large oraz steps-large. Na zbiorze square-large będziemy korzystać z architektury zawierającej jedną ukrytą warstwę o 100 neuronach, learning_rate będzie podany na wykresie. We wszystkich eksperymentach parametry optimizerów będą równe 0.9.



Na zbiorze steps-large wykorzystamy architekturę z jedną ukrytą warstwą o 40 neuronach, learning_rate również będzie podany na wykresie



Wyniki eksperymentów zostały zapisane w tabeli poniżej:

	Train set		Test set	
Datasets	Momentum	RMSProp	Momentum	RMSProp
square-large	2.03	5.93	231.83	462.33
steps-large	17.87	91.30	14.96	91.16

Duże MSE na square-large wynika z faktu, że x zawiera się w przedziale -1.5 do 2 na zbiorze treningowym, a w zbiorze testowym na przedziale -2 do 2.

3.3 Wnioski na podstawie eksperymentów

Na podstawie eksperymentów, możemy stwierdzić ze optimizery zostały zaimplementowane poprawnie. Należy zwrócić uwagę, że mechanizm momentum efektywnie zwiększa krok uczenia, co może pomóc pokonać wypłaszczenia funkcji straty oraz pomóc wyjść z minimów lokalnych. Metoda RMSProp pozwala na dostosowanie kroku uczenia dla każdego parametru oddzielnie. Wynika to z faktu, że dzieli krok uczenia przez średnią kwadratu gradientu. Dzięki temu dobrze sobie radzi z niestabilnymi oraz rzadkimi gradientami.

4 Rozwiązywanie zadania klasyfikacji

4.1 Opis tematu laboratoriów

Zaimplementować funkcję softmax dla warstwy wyjściowej sieci neuronowej. Sprawdzić szybkość i skuteczność w wariancie, gdy sieć używa funkcji softmax na ostatniej warstwie i gdy jest użyta zwykła funkcja aktywacji. Softmax wymaga też odpowiednich zmian w algorytmie uczenia, uwzględniających pochodną funkcji.

Przeprowadzić eksperymenty na zbiorach:

1. rings3-regular
2. easy
3. xor3

4.2 Opis wykonanej pracy i wyniki eksperymentów

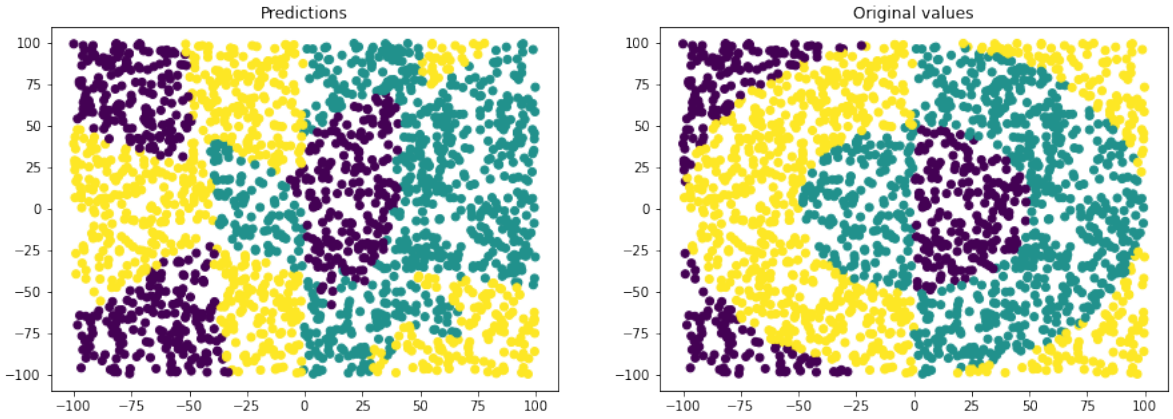
4.2.1 Wykonana praca

Zaimplementowanie funkcji softmax oraz jej gradientu i dodanie ich do słowników z funkcjami aktywacji i gradientami funkcji aktywacji. Zaimplementowanie funkcji straty cross-entropy oraz jej gradientu i dodanie do odpowiednich słowników. Dodatkowo przygotowanie funkcji, która zamienia pojedynczy label podany w zbiorach danych na wektor one-hot.

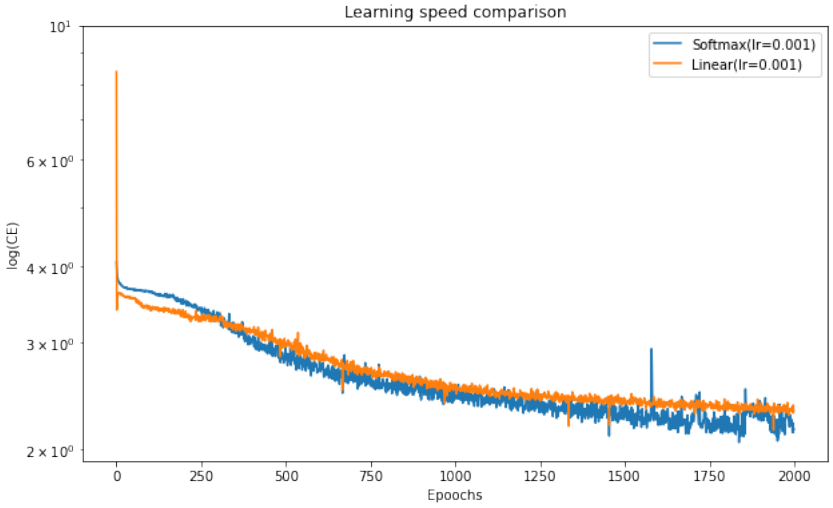
4.2.2 Eksperymenty

Przetestujemy efektywność funkcji softmax na zbiorze rings3-regular, będziemy porównywać z liniową funkcją aktywacji, wykorzystamy zwykły zbiór danych oraz znormalizowany. Wykorzystamy architekturę sieci z jedną ukrytą warstwą o 20 neuronach, wartości learning_rate będą podane na wykresach.

Dla zbioru nieznormalizowanego predykcje zostały zaprezentowane poniżej.

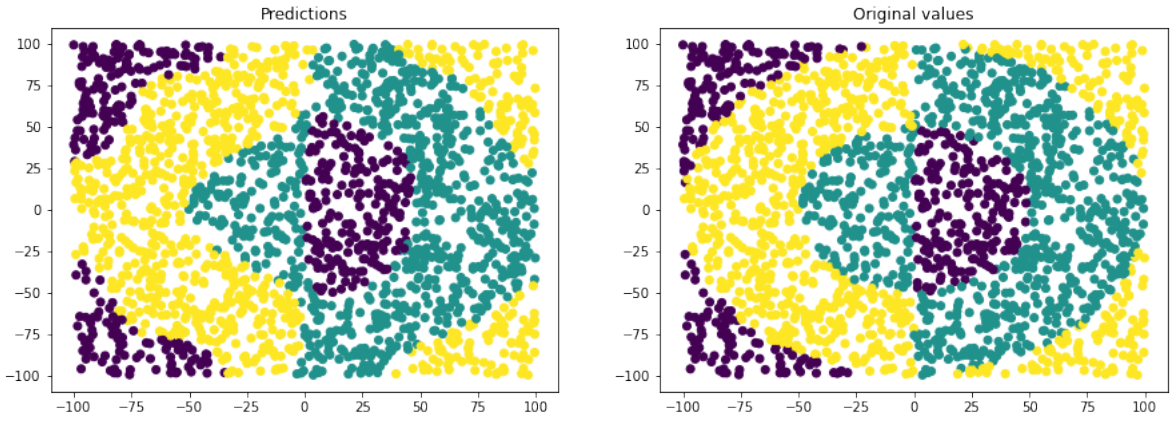


Otrzymaliśmy f1 na poziomie 0.75, jest to wartość na granicy akceptowalnej dla tego zbioru.

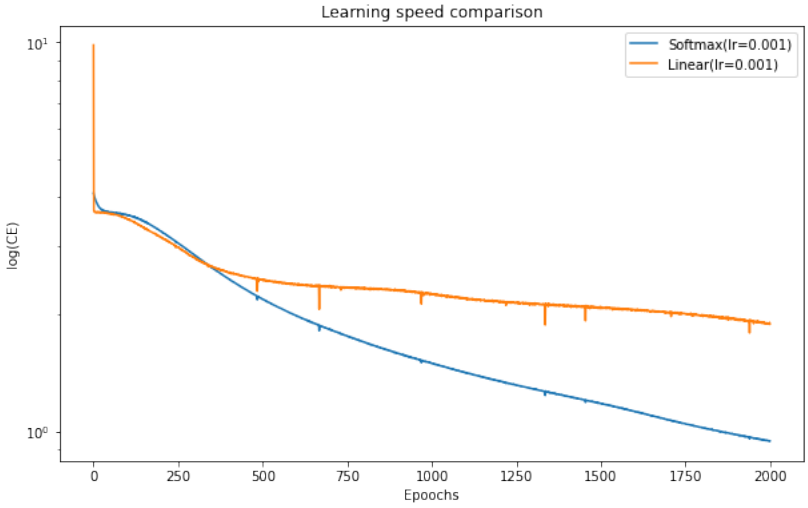


Widzimy, że softmax radzi sobie nieznacznie lepiej od liniowej funkcji kosztem bardziej niestabilnego spadku wartości funkcji straty.

Dla zbioru znormalizowanego predykcje zostały zaprezentowane poniżej.

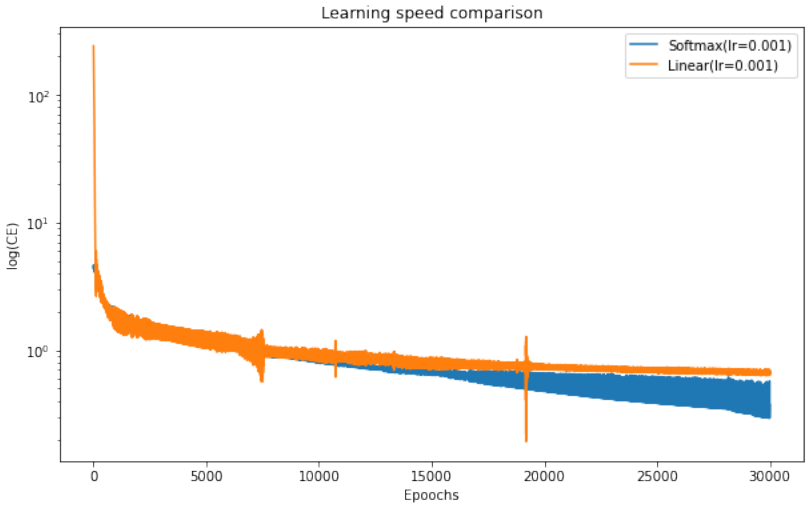


Na znormalizowanym zbiorze danych otrzymujemy f1 na poziomie 0.91 jest to dużo lepszy wynik niż na nieznormalizowanym zbiorze, jest ono o wiele powyżej akceptowalnego.



Na znormalizowanych danych różnica między funkcją softmax a liniową jest jeszcze większa niż na nieznormalizowanych. Warto zwrócić uwagę, że dzięki normalizacji otrzymaliśmy o wiele większe f1, dla obu funkcji aktywacji, a w zadaniu klasyfikacji lepiej sprawdza się funkcja softmax

Na zbiorze xor3 wykorzystaliśmy architekturę z jedną warstwą ukrytą o 100 neuronach, widzimy wyraźną przewagę funkcji softmax nad liniową.



4.3 Wnioski na podstawie eksperymentów

Na podstawie eksperymentów możemy stwierdzić, że mechanizm klasyfikacji została zaimplementowany poprawnie, a funkcja softmax radzi sobie lepiej od liniowej. Dodatkowo normalizacja danych wpłynęła bardzo pozytywnie na jakość predykcji.

5 Testowanie różnych funkcji aktywacji

5.1 Opis tematu laboratoriów

Należy rozszerzyć istniejącą implementację sieci i metody uczącej o możliwość wyboru funkcji aktywacji:

1. sigmoid
2. liniowa
3. tanh
4. ReLU

Porównać szybkość uczenia i skuteczność sieci w zależności od liczby neuronów w poszczególnych warstwach i rodzaju funkcji aktywacji. Należy wziąć pod uwagę fakt, że różne funkcje aktywacji mogą dawać różną skuteczność w zależności od liczby neuronów i liczby warstw. Sprawdzić sieci z jedną, dwiema i trzema warstwami ukrytymi. Podobnie jak w poprzednim tygodniu, trzeba dostosować proces uczenia do pochodnych nowych funkcji aktywacji.

Przeprowadzić testy wstępne dla zbioru multimodal-large (regresja), dla wszystkich trzech architektur i wszystkich czterech funkcji aktywacji.

Dla pozostałych zbiorów wybrać dwa najlepsze zestawy i zbadać ich skuteczność:

1. regresja
   (a) steps-large
2. klasyfikacja
   (a) rings5-regular
   (b) rings3-regular

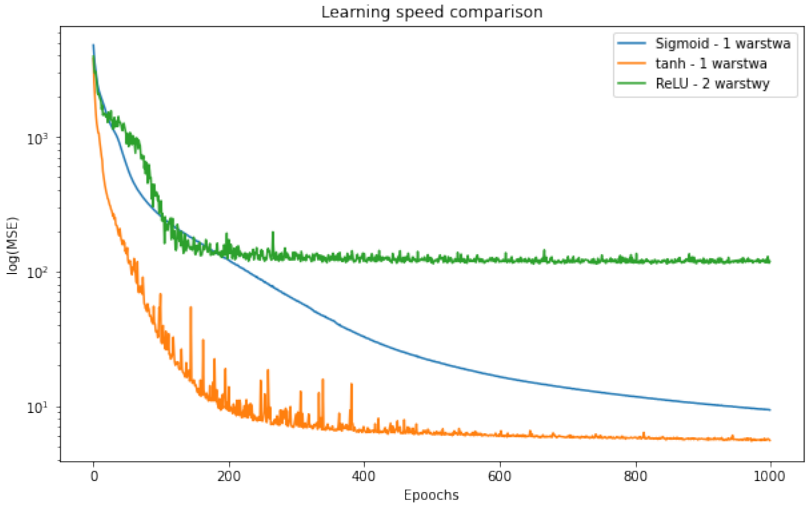
5.2 Opis wykonanej pracy i wyniki eksperymentów

5.2.1 Wykonana praca

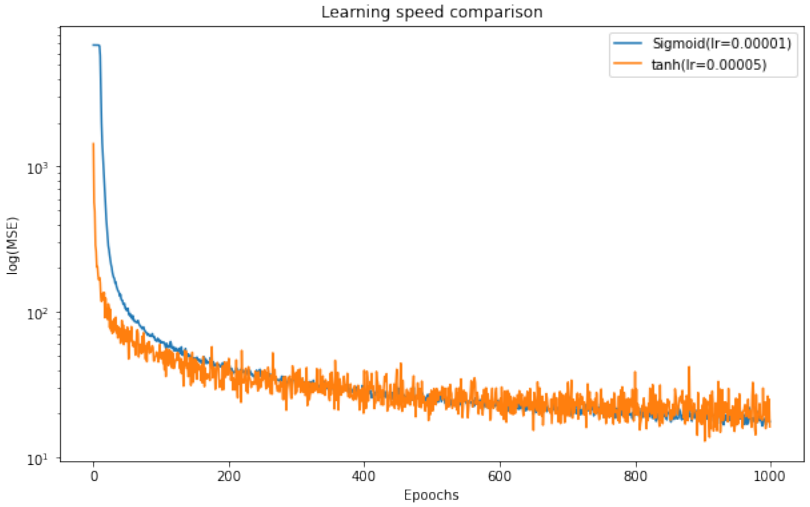
Zaimplementowanie funkcji aktywacji ReLU oraz tanh oraz ich gradientów, dodanie do odpowiednich słowników.

5.2.2 Eksperymenty

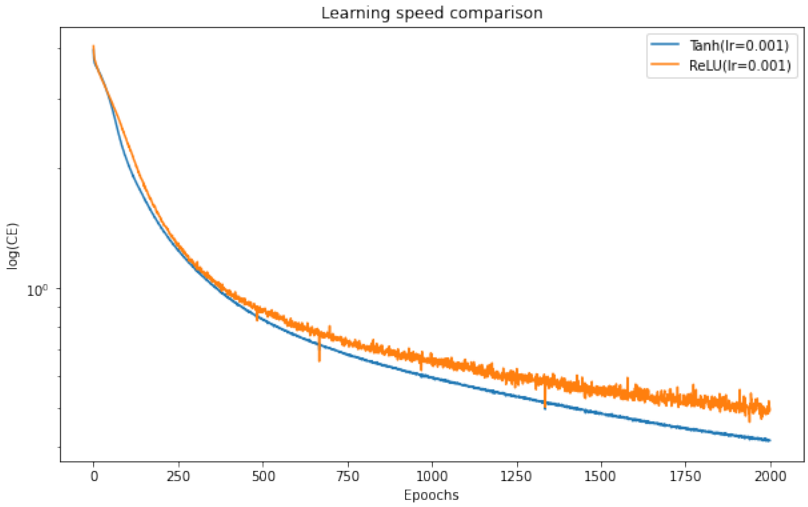
Przetestujemy działanie nowych funkcji aktywacji na 3 zbiorach - multimodal-large, rings5-regular oraz rings3-regular. Architektura każdej sieci będzie podana na wykresie.



Widzimy, że tanh oraz sigmoid są odpowiednimi funkcjami aktywacji dla danego problemu, podczas gdy sieć z funkcją aktywacji ReLU nie jest odpowiednia. Funkcja tanh nauczyła się szybciej kosztem większej niestabilności. Dla zbioru steps-large została wykorzystana architektura z dwiema ukrytymi warstwami o 32 neuronach.



Obie funkcje zbiegają do podobnego minimum, z tą różnicą ze tanh robi to dużo mniej stabilnie. Dla zbioru rings3-regular wykorzystamy sieć z 2 ukrytymi warstwami o 20 neuronach.



Tanh zbiega niewiele szybciej i do mniejszego minimum, możemy uznać, że jest lepsza funkcją aktywacji dla tego problemu.

5.3 Wnioski na podstawie eksperymentów

Na podstawie eksperymentów możemy stwierdzić, że funkcje aktywacji ReLU oraz tanh zostały poprawnie zaimplementowane. Należy podkreślić, że wybór odpowiedniej funkcji aktywacji zależy od wielu czynników, takich jak rodzaj problemu, który chcemy rozwiązać, liczba warstw w modelu, liczba neuronów w każdej warstwie. Nie ma jednoznacznej odpowiedzi na pytanie, która funkcja aktywacji jest najlepsza, ponieważ różne funkcje mogą działać lepiej w różnych scenariuszach.

6 Zjawisko przeuczenia + regularyzacja

6.1 Opis tematu laboratoriów

Zaimplementować mechanizm regularyzacji wag w sieci oraz mechanizm zatrzymywania uczenia przy wzroście błędu na zbiorze wariacyjnym.

Przeprowadzić eksperymenty na zbiorach i porównać skuteczność na zbiorze testowym dla różnych wariantów przeciwdziałania przeuczeniu sieci:

1. multimodal-sparse
2. rings5-sparse
3. rings3-balance
4. xor3-balance

6.2 Opis wykonanej pracy i wyniki eksperymentów

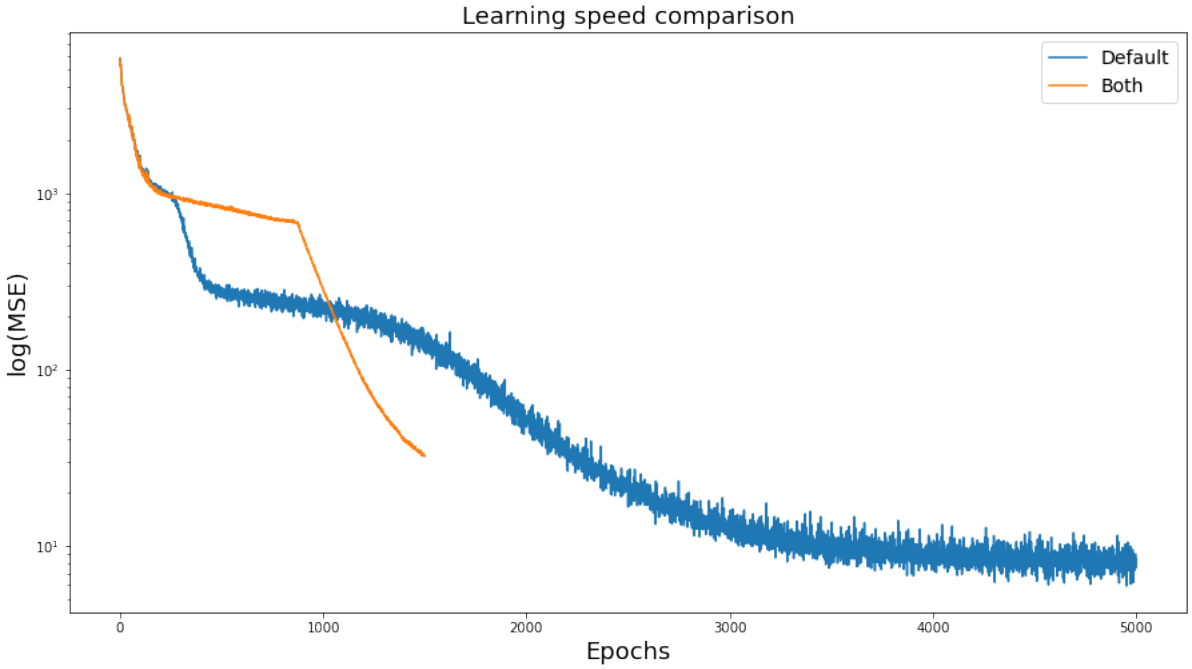
6.2.1 Wykonana praca

Zaimplementowanie pochodnej funkcji $L2$ regularyzacji wag i dodanie jej do zwykłego gradientu funkcji straty podczas propagacji wstecznej. Dodatkowo co epokę zostaje obliczana funkcja straty na zbiorze testowym, dzięki temu możemy zatrzymać uczenia, w momencie, kiedy otrzymamy zadowalający wynika na zbiorze testowym.

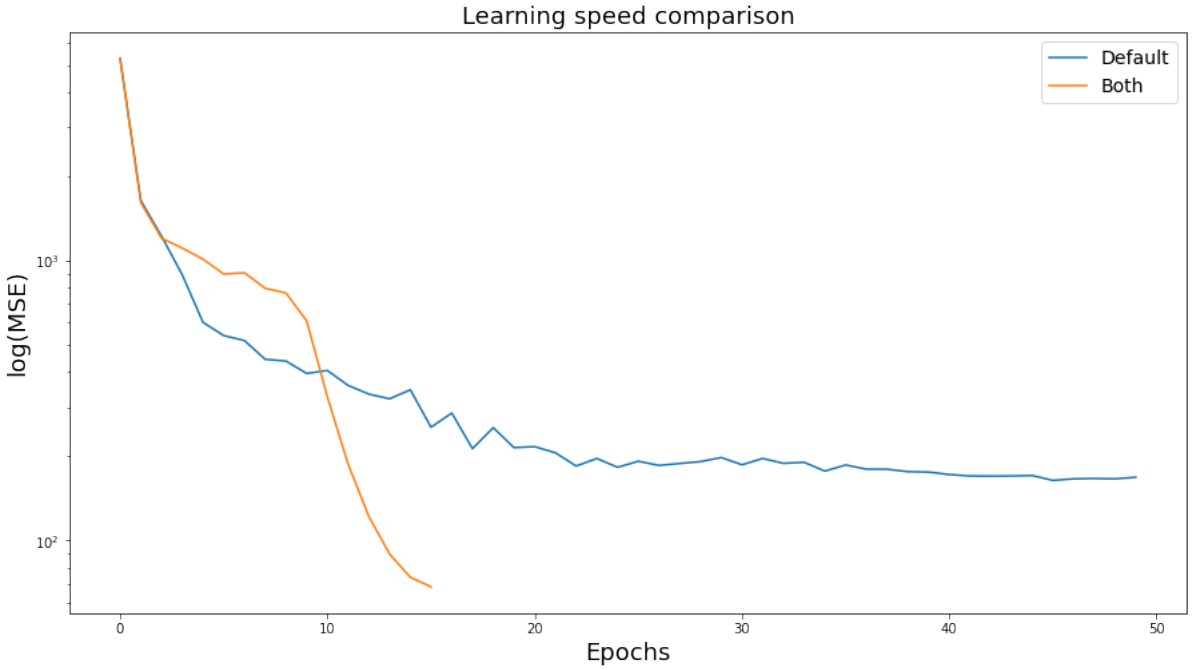
6.2.2 Eksperymenty

przetestujemy działanie regularyzacji wag oraz early-stopping na zbiorach multimodal-sparse oraz rings5-sparse. Na zbiorze multimodal-sparse mechanizm regularyzacji wag zadziałał bardzo skutecznie:

Na zbiorze treningowym:



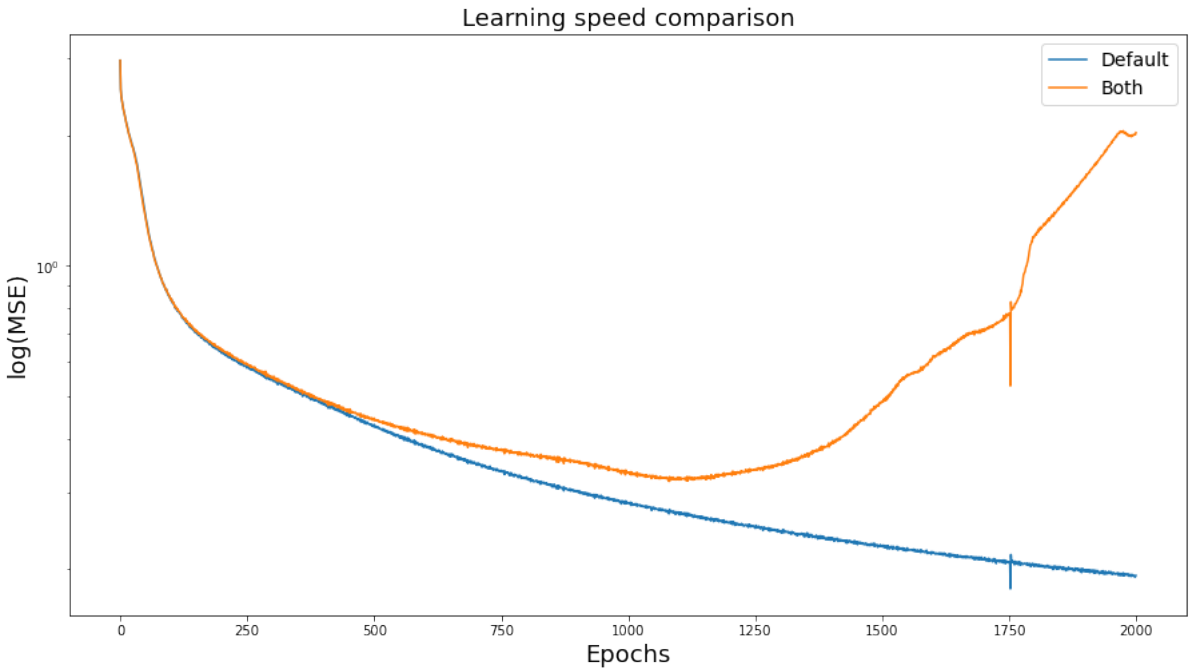
Na zbiorze testowym:



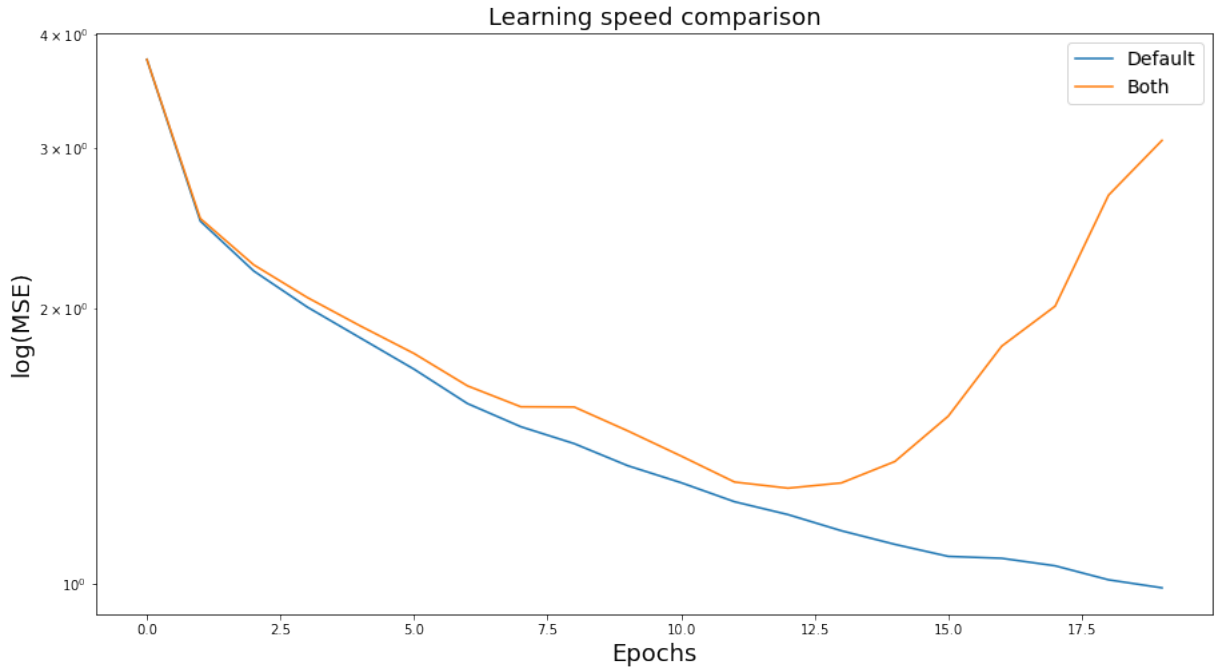
Dodatkowo na wykresach można zauważyć działanie mechanizmu early-stopping po osiągnięciu zadowalającego wyniku na zbiorze testowym proces uczenia został przerwany.

Na zbiorze rings5-sparse regularyzacja wag wpłynęła negatywnie na proces uczenia:

Na zbiorze treningowym:



Na zbiorze testowym:



6.3 Wnioski na podstawie eksperymentów

Na podstawie eksperymentów możemy stwierdzić mechanizm early-stopping oraz regularyzacja wag zostały zaimplementowane poprawnie. Należy podkreślić, że regularyzacja wag nie powinna być wykorzystywana w dowolnym problemie, a jedynie, kiedy istnieje możliwość przeuczenia.