Sprawozdanie - MLP

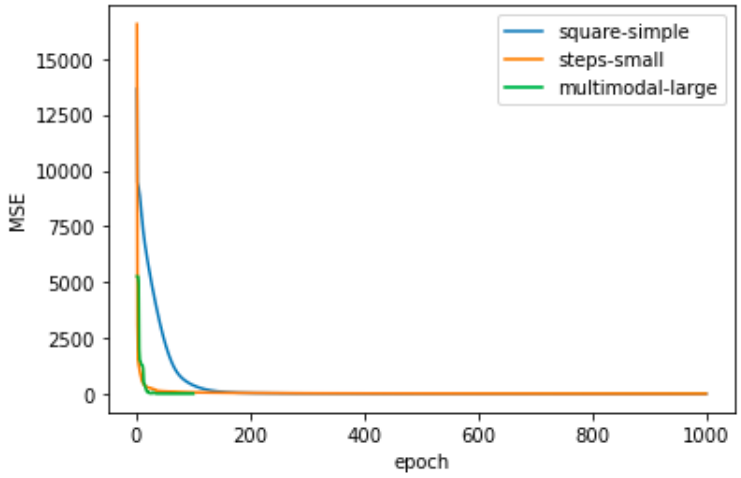
Jakub Lis
305744

Streszczenie

Laboratorium dotyczyło budowy modelu perceptronu wielowarstwowego (multilayer perceptron, MLP). W kolejnych tygodniach dodawane były nowe funkcjonalności poprawiające działanie tego typu sieci i możliwości eksperymentowania z różnymi architekturami oraz parametrami. W sprawozdaniu został opisany wpływ między innymi zmian architektur, funkcji aktywacji czy parametrów przy procesie uczenia poparty doświadczeniem zdobytym na przestrzeni tych tygodni oraz eksperymentami stworzonymi na potrzeby tego raportu.

1 Bazowa implementacji oraz propagacja wsteczna błędu

Początkowym zdaniem było zaimplementowanie sieci neuronowej typu feedforward, a następnie dodanie możliwości uczenia jej, tzn. dodanie propagacji wstecznej błędu. Wówczas jedynymi używanymi funkcjami aktywacji była funkcja sigmoid oraz liniowa. Wobec tego dla uzyskania niskich wyników MSE na zbiorach testowych danych square-simple, steps-small i multimodal-large trzeba było odnaleźć odpowiednie architektury sieci, dobrać krok uczenia (learning rate) oraz odpowiedni batch_size.



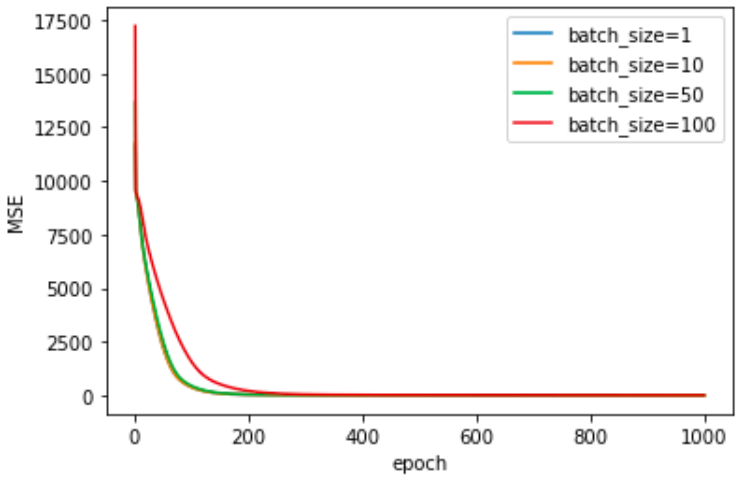
Rysunek 1: Uczenie sieci na podanych zbiorach

Wykres przedstawia jak zbiegał błąd (MSE) w procesie uczenia dla architektur podanych w tabeli niżej. Potwierdza on działanie propagacji wstecznej. Widać także, że dla zbioru multimodal-large wystarczyło 100 epok aby uzyskać satysfakcjonujący wynik.

Zbiór	architektura sieci	batch_size	MSE na zbiorze testowym
square-simple	1-200-1	50	2.99
steps-small	1-200-1	10	94.76
multimodal-large	1-32-32-1	10	2.57

Duży błąd w przypadku zbioru steps-small na zbiorze testowym wynikał z pojawienia się w testach punktów, które nie były zbliżone do punktów ze zbioru treningowego. Z tego powodu nie udało się uzyskać dobrego wyniku MSE, pomimo tego, że sieć dobrze dopasowywała się do danych.

Aby sprawdzić wpływ batch_size na uczenie przeprowadziłem test na jednym ze zbiorów. Wybrałem square-simple, do każdego batch_size ręcznie dobierałem krok uczenia, tak aby postarać się jak najbardziej minimalizować końcowy błąd.

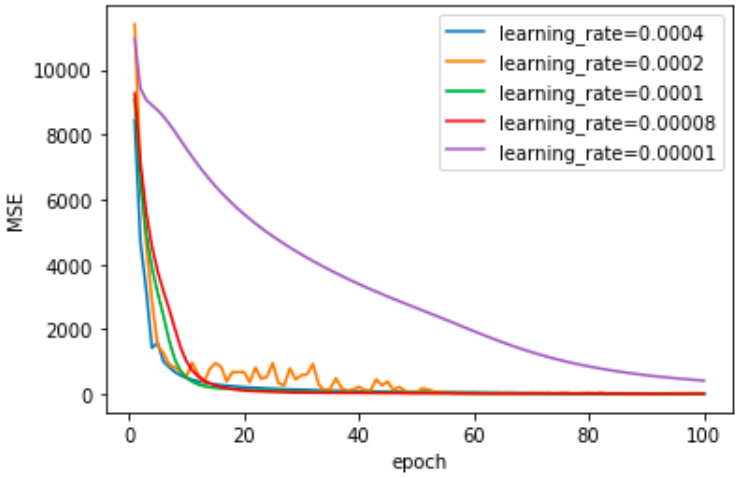


Rysunek 2: Uczenie sieci z podziałem na batch_size

Widać, że niezależnie od rozmiaru batcha wszystkie treningi zbiegły do podobnie małego błędu (ok. 2). Jedyna zauważalna różnica to szybsza zbieżność dla mniejszych batchy na początku uczenia, ale później się to wyrównało.

2 Moment i normalizacja gradientu

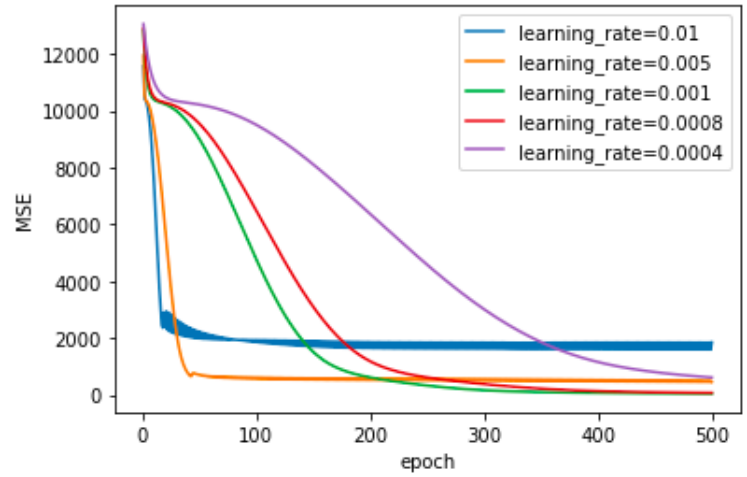
Implementacja momentu i normalizacji gradientu miała pomóc przy uczeniu naszej sieci MLP, tzn. przyspieszyć ją, gdy w danych epokach błąd rzeczywiście malał i powodować mniejsze zmiany wag, gdy błąd "skakał" lub rozbiegał.



Rysunek 3: Uczenie sieci z momentem

Widzimy, że uczenie z momentem pozwala na wybór kroku uczenia z dosyć szerokiego zakresu, a dla wszystkich wyborów sieć i tak się dobrze dopasowywuje do danych treningowych. Szczególnie, że w eksperymencie było tylko 100 epok. Wykorzystana architektura sieci to 1-100-1 z sigmoidalną funkcją aktywacji i rozmiarem batcha równym 100. Parametr λ używany w uczeniu z momentem wynosił 0.5.

Podobny eksperyment dla RMSProp prezentuje wykres poniżej.

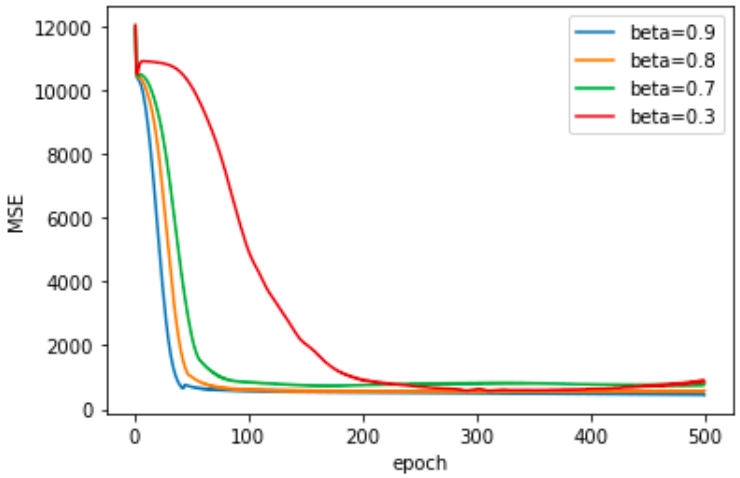


Rysunek 4: Uczenie sieci z normalizacją gradientu

Przy normalizacji gradientu użyta była identyczna architektura jak przy uczeniu z momentem. Różnicą jest parametr β zamiast λ, która wynosiła 0.9 dla wszystkich kroków uczenia. Użyte kroki

uczenia są także większe niż przy uczeniu z momentem, dla jeszcze mniejszych RMSProp zbiegał bardzo wolno w porównaniu z uczeniem z momentem.

Warto także sprawdzić jak zmiany parametru β wpływają na uczenie sieci, zostańmy więc przy kroku uczenia równym 0.005 i pozmieniajmy betę.



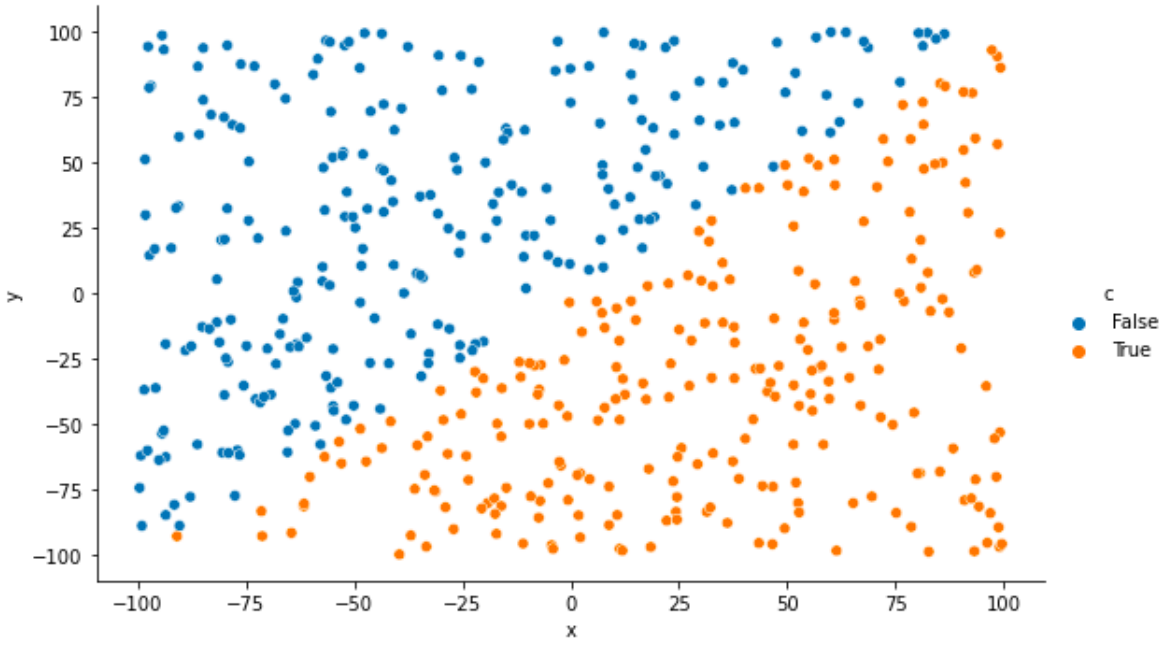
Rysunek 5: Uczenie sieci z normalizacją gradientu w zależności od β

Widać, że $\beta = 0.9$ była w tym przypadku najlepszym wyborem. Czym mniejsza wartość parametru β, tym wolniejsze zbieganie MSE. Ponadto dla małej wartości takiej jak 0.3 widać pod koniec mało stabilne zachowanie uczenia.

Porównując uczenie z momentem i RMSProp - to z momentem uczyło się szybciej. Już przy 100 epokach mamy błąd bardzo bliski 0 (Rysunek 3), dla RMSProp potrzeba na pewno więcej niż 300 epok, aby taki błąd osiągnąć (Rysunek 4).

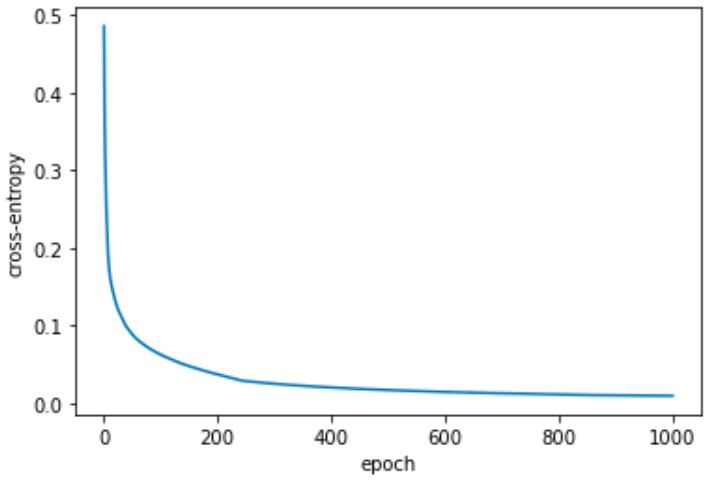
3 Klasyfikacja

Kolejnym etapem poprawy sieci neuronowej było dodanie funkcji softmax dla warstwy wyjściowej pozwalającej na efektywniejsze uczenie przy zadaniach klasyfikacji. Eksperymenty miały być przeprowadzone na zbiorach rings3-regular, easy, xor3. Niestety, satysfakcjonujący wynik udało się uzyskać jedynie na zbiorze easy.



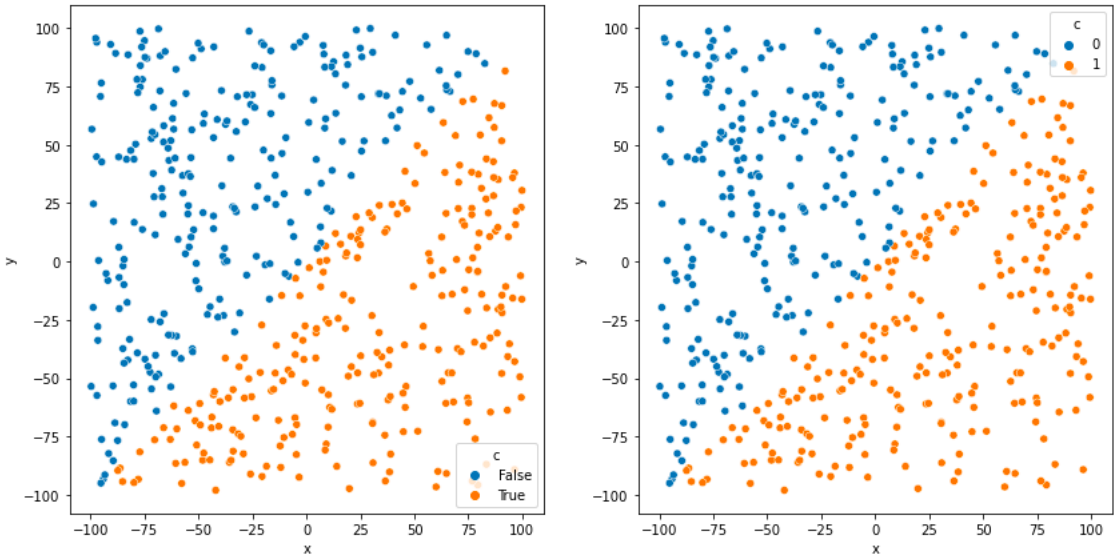
Rysunek 6: Zbiór easy; kolor punktu - klasa, do której przynależy punkt

Wykres powyżej przedstawia zbiór treningowy zbioru easy. Kolorami zaznaczone zostały klasy. Proces uczenia z wykorzystaniem funkcji strasy cross-entropy oraz z funkcją softmax na końcu sieci przebiegał tak jak poniżej.



Rysunek 7: Uczenie sieci na zbiorze easy

Wystarczyło więc 1000 epok, aby uzyskać bardzo niską wartość cross-entropy na zbiorze treningowym. Warto więc sprawdzić jak poradziła sobie sieć na zbiorze testowym.



Rysunek 8: Z lewej strony zbiór testowy z prawdziwymi klasami, z prawej predykowane klasy na zbiorze testowym przez nauczoną sieć neuronową

Otrzymane F1-score na zbiorze testowym ostatecznie było wyższe niż 0.99.

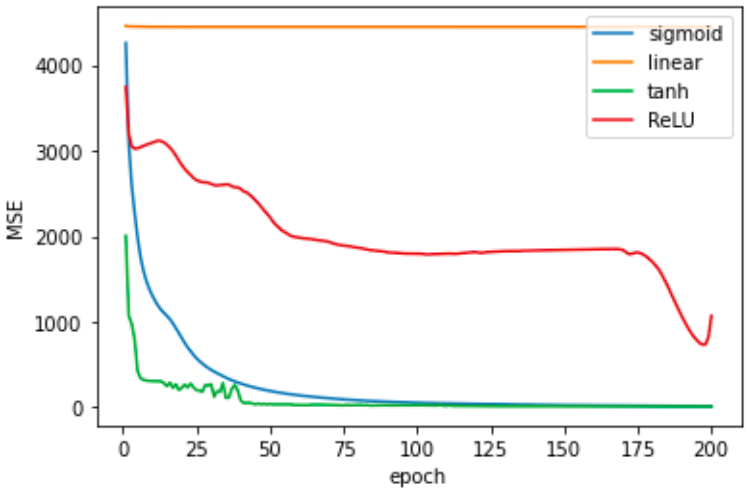
Przy pozostałych zbiorach nie udało się nauczyć sieci. Wartości metryki F1 były ostatecznie równe 0.61 dla zbioru xor3 i 0.68 dla zbioru rings3-regular.

4 Testowanie różnych funkcji aktywacji

Należało rozszerzyć implementację sieci o nowe funkcje aktywacji. Opiszę jak wyglądały testy wstępne dla zbioru multimodal-large takich funkcji aktywacji jak sigmoid, liniowa, tanh oraz ReLU. W przypadku ReLU implementacja pochodnej to zwracanie 1, gdy $x > 0$ i 0 w przeciwnym przypadku.

4.1 Jedna warstwa ukryta

Testy funkcji aktywacji podzielimy na trzy części: kiedy jest jedna warstwa ukryta, kiedy są dwie i kiedy były trzy. Zacznijmy od przypadku z jedną warstwą ukrytą.



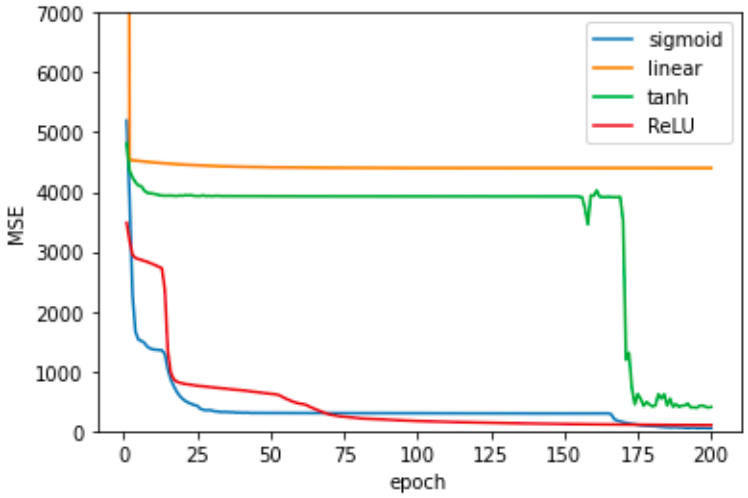
Rysunek 9: Uczenie sieci z jedną warstwą ukrytą w zależności od użytej funkcji aktywacji

Widać, że dla jednej warstwy ukrytej najlepiej poradziła sobie sieć z funkcjami aktywacji sigmoid oraz tanh. We wszystkich przypadkach na warstwie wyjściowej jest funkcja liniowa. Poza tym widać pewną zbieżność ReLU, ale pod koniec nie zachowuje się stabilnie, natomiast w przypadku funkcji liniowej po prostu nie zbiegała. Poniższa tabelka przedstawia użyte architektury, rozmiary batchu oraz końcowy błąd na zbiorze testowym.

Funkcja aktywacji	architektura sieci	batch_size	MSE na zbiorze testowym
sigmoid	1-50-1	50	10.6
liniowa	1-50-1	50	4434.4
tanh	1-50-1	50	8.4
ReLU	1-200-1	10	1302.6

4.2 Dwie warstwy ukryte

Następnie przetestowane zostało uczenie sieci z dwiema warstawmi ukrytmi podczas 200 epok. Na wyjściu w każdym przypadku używana była liniowa funkcja.



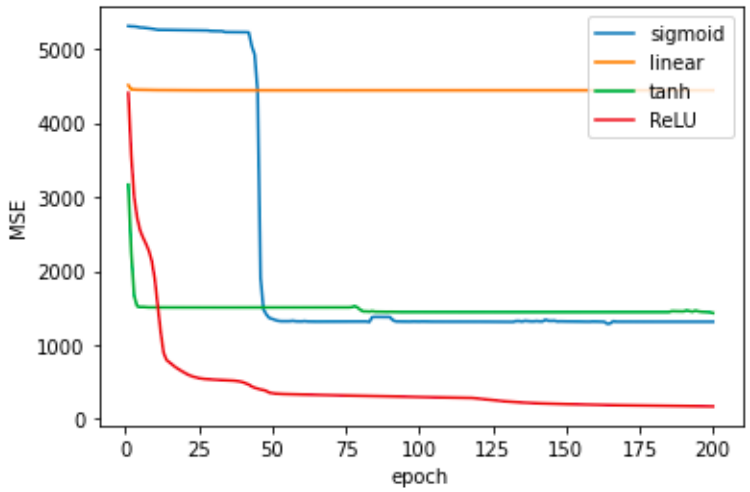
Rysunek 10: Uczenie sieci z dwiema warstwami ukrytymi w zależności od używanych funkcji aktywacji

Tym razem najlepiej poradziły sobie funkcje sigmoid oraz ReLU. Liniowa ponownie nie zbiegała poza pierwszymi epokami, natomiast tanges hiperboliczny długo pozostawał na jednym poziomie, później miał duży skok do niższego błędu, ale wciąż nie uzyskał lepszych wartości MSE niż sigmoid i ReLU. Ponownie wyniki MSE na zbiorze testowym umieśćmy w tabeli.

Funkcja aktywacji	architektura sieci	batch_size	MSE na zbiorze testowym
sigmoid	1-20-30-1	10	48.7
liniowa	1-100-50-1	50	4434.1
tanh	1-10-40-1	50	390.5
ReLU	1-20-10-1	5	83.7

4.3 Trzy warstwy ukryte

Ostatnie testy ze zbiorem multimodal-large były z trzema warstwami ukrytymi. Uczenie takich architektur przebiegało jak poniżej.



Rysunek 11: Uczenie sieci z trzema warstwami ukrytymi w zależności od używanych funkcji aktywacji

Tym razem przy 200 epokach jedynie w przypadku ReLU widać dobrą zbieżność, ale testowane było jedynie 200 epok, więc ciężko stwierdzić jak przy dużo większej liczbie epok zachowałyby się sieci z sigmoid i tanh - w szczególności tutaj w przypadku sigmoidu widzimy nagły spadek w okolicach 50-tej epoki. Może dalej pojawiłyby się takie kolejne.

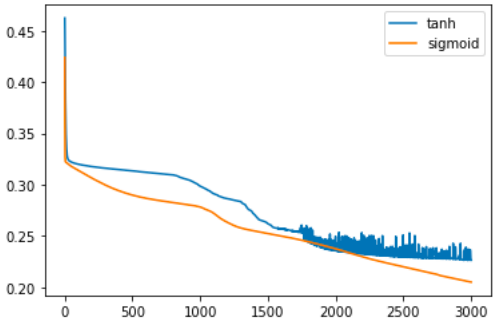
Funkcja aktywacji	architektura sieci	batch_size	MSE na zbiorze testowym
sigmoid	1-20-20-30-1	10	1312
liniowa	1-10-10-10-1	50	4434.3
tanh	1-5-10-50-1	50	1536.3
ReLU	1-32-16-8-1	5	119.5

4.4 Wnioski dotyczące funkcji aktywacji

W przypadku sieci z jedną warstwą ukrytą najlepiej działała funkcja tanh, jeżeli chodzi o końcowy błąd. Wykres jednak pokazuje, że sigmoid stopniowo dopasowywał się i nie miał takich niestabilnych skoków jak tanges hiperboliczny - z tego powodu użycie funckji sigmoid wydaje się być bezpieczniejszą opcją. Dla sieci z dwiema warstawmi ukrytymi najlepszy wynik na zbiorze testowym uzyskała sieć z funkcjami sigmoidalnymi. Wobec tego ponownie można uznać sigmoid za najlepszy. Natomiast w przypadku trzech warstw najlepszą i właściwie jedyną funkcją, która sobie radziła było ReLU. Ogólnie, w przypadku ReLU na podstawie wykresów (9-11) widzimy, że czym więcej warstw, tym sieć zachowywała się stabilniej (tzn. funkcja błędu od epoki była bardziej wygładzona). Funkcją, która na żadnej z architektur nie nadawała się do uczenia sieci neuronowej była funkcja liniowa. Jej używanie ma sens tylko na ostatniej warstwie. Widać także zależność szybkości zbiegania od liczby warstw ukrytych. Czym więcej warstw dodamy tym uczenie jest wolniejsze, tzn. wartość błędu z każdą epoką maleje wolniej, ale też czas wykonywania kodu był dłuższy.

4.5 Funkcje aktywacji w zadaniu klasyfikacji

W zadaniu klasyfikacji testowane były najlepsze architektury ustalone na zbiorze multimodal-large. Wobec tego wybrałem sigmoid oraz tanh. Zaprezentuję uzyskany wykres na zbiorze rings5-regular.

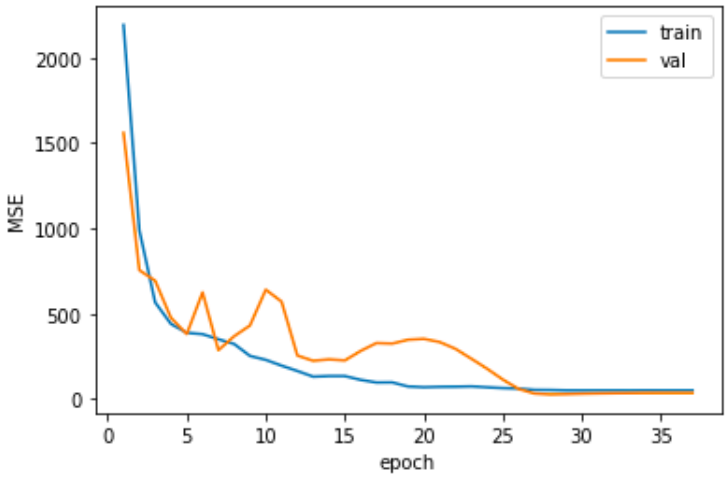


Rysunek 12: Uczenie sieci neuronowej z wybranymi funkcjami klasyfikacji przy zadaniu klasyfikacji

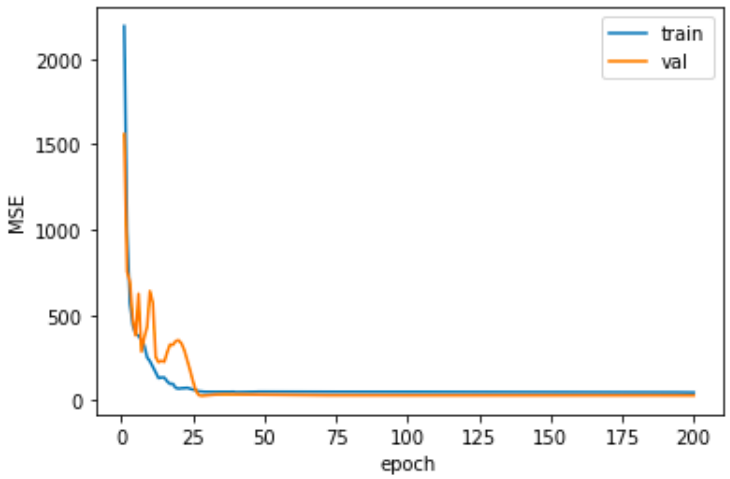
Jak widać, przy architekturze z dwiema warstwami ukrytymi, lepiej radził sobie sigmoid i ciągle stabilnie zbiegał, natomiast tanh pod koniec zaczął wariować.

5 Mechanizm zatrzymywania uczenia przy wzroście błędu na zbiorze walidacyjnym oraz regularyzacja wag w sieci

Do sieci dodana została opcja early stoppingu, gdy błąd na podanym zbiorze walidacyjnym zaczynał rosnąć lub utrzymywał się na identycznym poziomie. Przetestujemy jego działanie na sieci uczącej się na zbiorze zbiorze multimodal-large. Sieć ma dwie warstwy ukryte z funkcji aktywacji tanh.



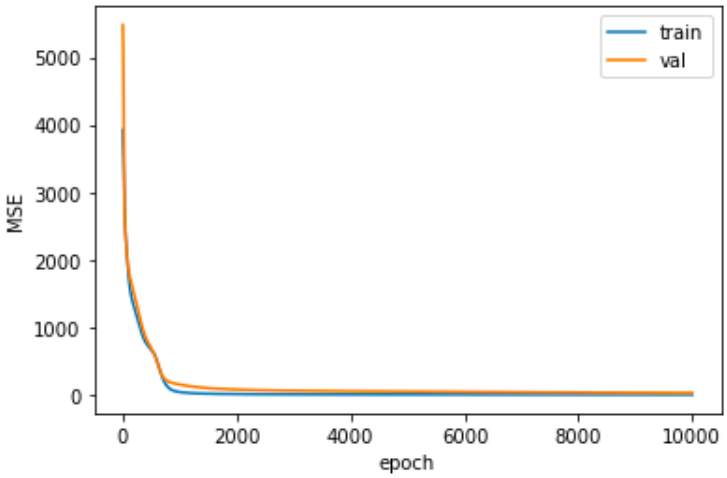
Rysunek 13: Zatrzymanie uczenia przy wzroście błędu na zbiorze walidacyjnym



Rysunek 14: Uczenie na ustalonych 200 epokach

Widzimy, że wcześniejsze zatrzymanie uczenia odbyło się przed 40-tą epoką. Na drugim wykresie jest przedstawiony proces uczenia 200 epok. Co prawda, nie mamy do czynienia z przeuczeniem w przypadku tego zbioru, ale warto było zatrzymać uczenie wcześniej dla oszczędności czasu.

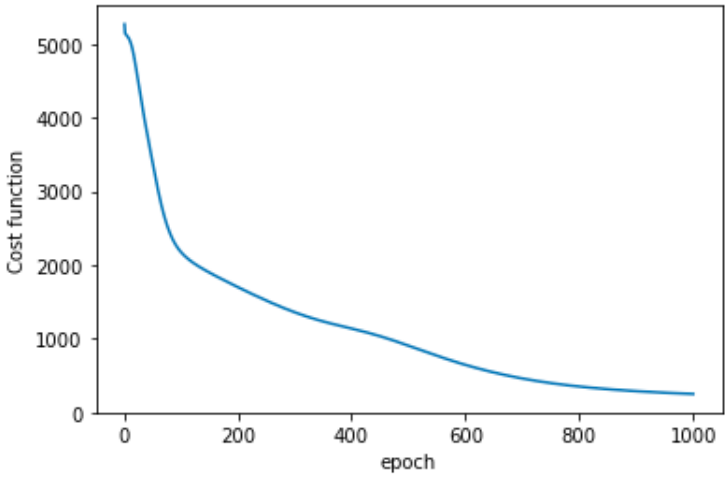
Przetestujemy wcześniejsze zatrzymywanie jeszcze na zbiorze multimodal-sparse.



Rysunek 15: Uczenie sieci na zbiorze multimodal-sparse

Niestety zbiór walidacyjny wyodrębniony ze zbioru treningowego dopasowywał się równie idealnie, co zbiór treningowy. Wobec tego wcześniejsze zatrzymanie (ustawione jedynie na 4 braki popraw MSE) nie zadziałało, tzn. po prostu nie odbyło się, gdyż z każdą epoką błąd na zbiorze walidacyjnym malał. Ostatecznie uzsykane MSE na zbiorze testowym było równe 689.2, pomimo tego, że na zbiorze treningowym i walidacyjnym było mniejsze od 10.

Przetestujmy więc działanie regularyzacji na tym zbiorze. Skorzystałem z regularyzacji L2 i uczyłem sieć na 1000 epokach.



Rysunek 16: Uczenie sieci na zbiorze multimodal-sparse z regularyzacją L2

Na osi y jest tym razem funkcja straty będąca sumą MSE oraz karą L2 za wagi w modelu.

Otrzymane MSE na zbiorze testowym wynosi 528.2 i jest rzeczywiście lepsze od nieudanego early stoppingu, ale wciąż duże.