Sprawozdanie z projektu - Implementacja sieci neuronowych

Mateusz Deptuch

kwiecień 2025

Spis treści

1 Wprowadzenie

Głównym celem zadania było zapoznanie się z architekturą i działaniem perceptronu wielowarstwowego. Było ono realizowane sekwencyjnie: utworzenie architektury sieci i udoskonalanie jej poprzez dodawanie licznych usprawnień działania, a następnie testowanie rezultatów ich działania.

2 Realizacja zadań i wyniki

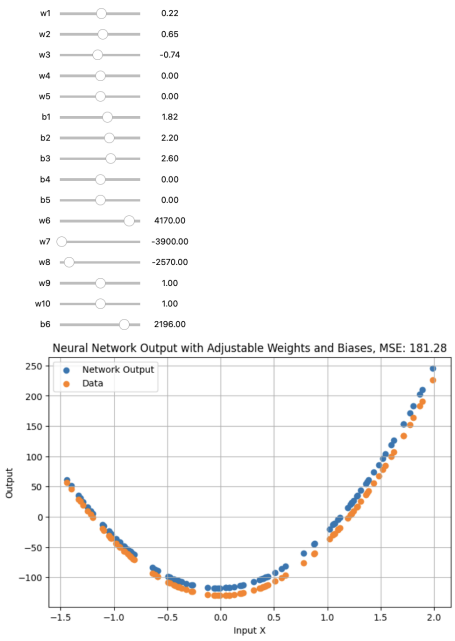
2.1 Laboratorium 1.

2.1.1 Opis zadania

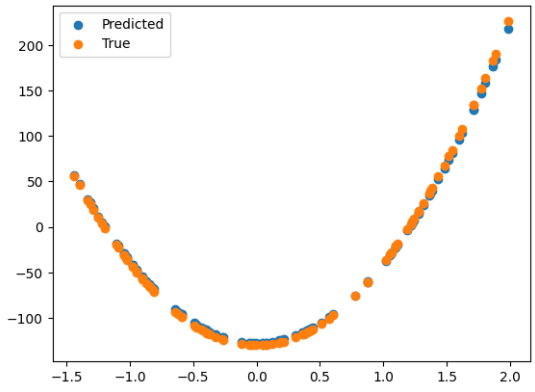
Zadanie polegało na zaimplementowaniu struktury sieci neuronowej MLP, a następnie ręcznym dopasowaniu wag w celu osiągnięcia zadowalających wyników działania modelu dla dwóch zbiorów danych.

2.1.2 Wyniki eksperymentu

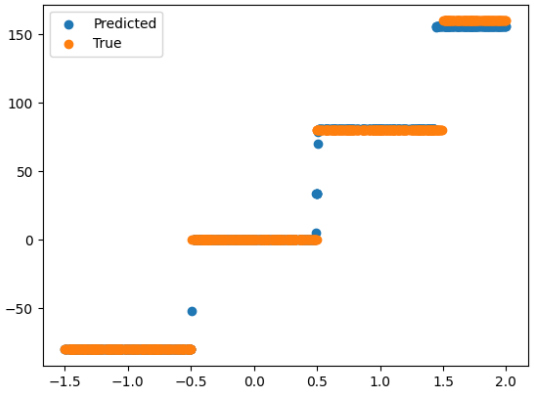
Model został zaimplementowany jako klasa w języku `python` przyjmująca wagi, liczbę warstw oraz liczbę neuronów na każdej z nich. Konstruktor przyjmował także funkcję aktywacji używaną przy wyliczaniu wyników modelu. Wagi dobrane zostały następnie ręcznie, za pomocą systemu suwaków i aktualizowanego na bieżąco wykresu, dla architektur sieci o jednej, dwóch i trzech warstwach ukrytych. Zadowalające wyniki zostały osiągnięte na jednej pięcioneuronowej warstwie ukrytej. Osiągnięte wyniki MSE dla zbiorów *square-simple* oraz *steps-large* wyniosły odpowiednio 7.06 i 6.95.



Rysunek 1: System suwaków do wyznaczania wartości wag i bias



Rysunek 2: Działanie sieci dla zbioru square-simple



Rysunek 3: Działanie sieci dla sbioru steps-large

Warstwa	**Liczba neuronów**	**Wagi**	**Biasy**
Wejściowa	1	-	-
Ukryta 1	5	$\begin{bmatrix} 0.22 & 0.65 & -0.74 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1.82 & 2.20 & 2.65 & 0 & 0 \end{bmatrix}$
Wyjściowa	1	$\begin{bmatrix} 4170.00 \\ -3900.50 \\ -2557.60 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 2184.30 \end{bmatrix}$

Tabela 1: Finalna architektura sieci neuronowej dla funkcji square-simple

2.1.3 Wnioski

Ręczne dopasowanie sieci neuronowej do funkcji o różnych postaciach jest możliwe, aczkolwiek bardzo czasochłonne, a przy bardziej złożonych funkcjach zapewne bardzo trudne, gdyż im bardziej złożona jest struktura sieci, tym mniej intuicyjne staje się ustawianie poszczególnych wag.

2.2 Laboratorium 2.

2.2.1 Opis zadania

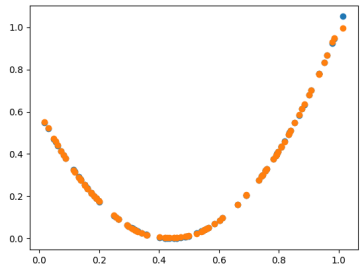
Głównym celem zadania było zaimplementowanie procesu uczenia sieci neuronowej z wykorzystaniem propagacji wstecznej błędu (backpropagation). Należało zaimplementować dwie metody aktualizacji wag: po prezentacji wszystkich wzorców oraz po prezentacji kolejnych porcji danych (batch). Zadanie obejmowało także naukę sieci na 3 zbiorach danych: *square-simple*, *steps-small* oraz *multimodal-large*.

2.2.2 Wyniki eksperymentu

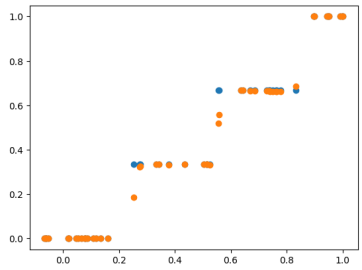
Zaimplementowana została metoda treningu, używająca propagacji wstecznej błędu, we wcześniej napisanej klasie zawierającej model MLP. Przyjmuje ona zarówno architekturę sieci, jak i współczynnik uczenia oraz funkcję aktywacji i jej pochodną. Istotne dla wyników okazało się użycie odpowiedniej metody inicjalizacji wag. Zaimplementowana została zarówno metoda jednostajnego losowania, jak i metoda *Xavier* o bardziej złożonym rozkładzie. Metoda Xavier okazała się bardziej skuteczna, i to na niej prowadzone były dalsze eksperymenty. Osiągnięte wyniki wyglądały następująco:

Zbiór danych	**Architektura**	**Train MSE**	**Test MSE**
square-simple	[1, 6, 6, 6, 1]	0,62	3,84
steps-small	[1, 5, 5, 5, 1]	0,32	63
multimodal-large	[1, 20, 20, 1]	10,01	5,25

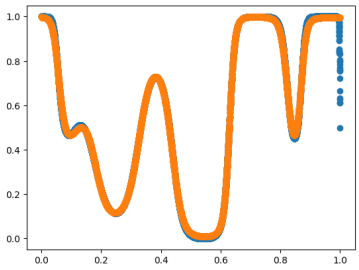
Tabela 2: Osiągnięte wyniki uczenia sieci dla różnych zbiorów danych



Rysunek 4: Przewidywania wytrenowanego modelu (square-simple)

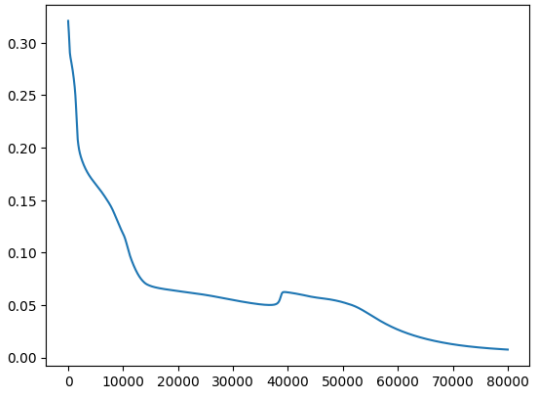


Rysunek 5: Przewidywania wytrenowanego modelu (steps-small)

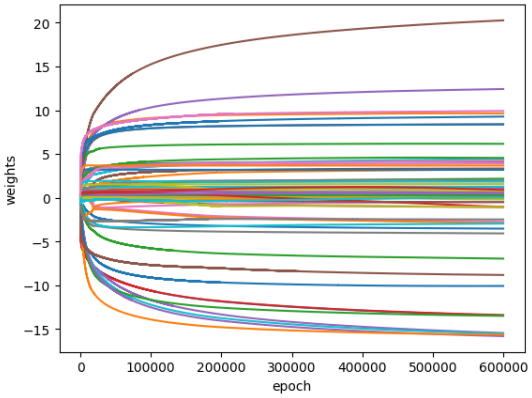


Rysunek 6: Przewidywania wytrenowanego modelu (multimodal-large)

Ponadto zaimplementowany został system umożliwiający analizę tempa uczenia się sieci (wykres błędu dla każdej z epok) oraz wizualizację zmieniania się wag sieci.



Rysunek 7: Wykres błędu od liczby epok dla multimodal-large



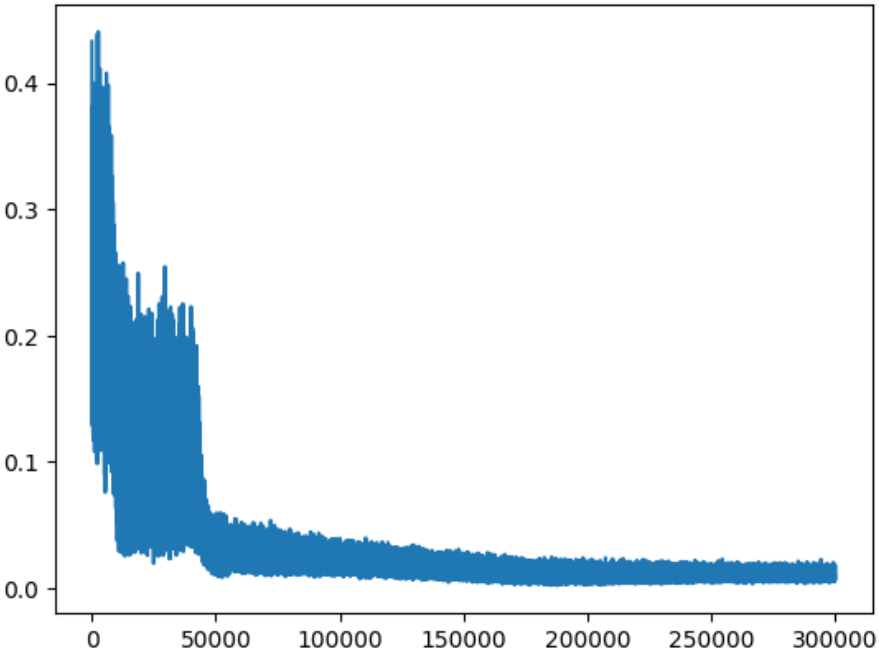
Rysunek 8: Wykres wartości wag od liczby epok dla square-simple

Drugą częścią zadania było zaimplementowanie propagacji wstecznej z aktualizacją wag po prezentacji wszystkich wzorców oraz po prezentacji kolejnych porcji danych (batch). Dla tej metody modele nie musiały być już trenowane tak dokładnie, ponieważ pożądane wyniki zostały osiągnięte wyżej.

Zbiór danych	**Architektura**	**Train MSE**	**Test MSE**
square-simple	[1, 6, 6, 6, 1]	19,62	23,49
steps-small	[1, 5, 5, 5, 1]	9,40	96
multimodal-large	[1, 20, 20, 1]	9,53	4,48

Tabela 3: Osiągnięte wyniki uczenia sieci dla różnych zbiorów danych przy użyciu metody batch

Jak widać, metoda batch poradziła sobie lepiej dla, mogłoby się wydawać, najbardziej złożonej funkcji *multimodal-large*. Można było także zauważyć, że ta metoda jest zdecydowanie mniej stabilna, co widać dobrze na wykresach błędu od numeru epoki. Wynika to bezpośrednio ze struktury metody, która polega na wyliczaniu kierunku skoku na podstawie próbki, a nie całego zbioru na raz.



Rysunek 9: Wykres błędu od liczby epok dla square-simple, metoda batch

2.2.3 Wnioski

Propagacja wsteczna błędu pozwala na dużo prostsze i efektywniejsze trenowanie sieci neuronowych. Zarówno metoda aktualizacji wag po prezentacji wszystkich wzorców, jak i metoda batch dobrze przybliżają funkcje różnej postaci, przy czym metoda batch jest dużo mniej stabilna, jednakże także dąży do optymalnego przybliżenia.

2.3 Laboratorium 3.

Zaimplementować dwa usprawnienia uczenia gradientowego sieci neuronowej: moment, normalizację gradientu RMSProp. Porównać szybkość zbieżności procesu uczenia dla obu wariantów.

2.3.1 Opis zadania

Zadanie polegało na implementacji dwóch usprawnień uczenia gradientowego sieci neuronowej:

- uczenie z momentem
- RMSProp

Uczenie z momentem polega na uwzględnieniu poprzedniego kierunku aktualizacji wag, co pozwala przyspieszyć zbieżność i zmniejszyć ryzyko utknięcia w lokalnym minimum. Normalizacja gradientu RMSProp polega na dzieleniu wartości gradientu przez średnią ruchomą jego kwadratu, co stabilizuje i dostosowuje tempo uczenia dla każdego parametru osobno. Następnie należało przetestować powyższe metody na trzech zbiorach danych:

- square-large
- steps-large
- multimodal-large

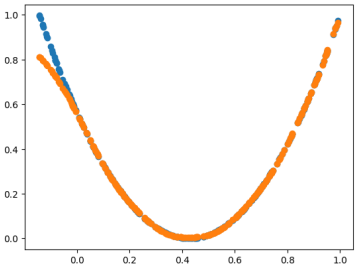
2.3.2 Wyniki eksperymentu

Zaimplementowane metody znacząco poprawiły wydajność uczenia się sieci oraz ich wyniki. Zauważalna też była mniejsza zależność wyniku od punktu startowego. Poniżej zobaczyć można wyniki dla uczenia z momentem:

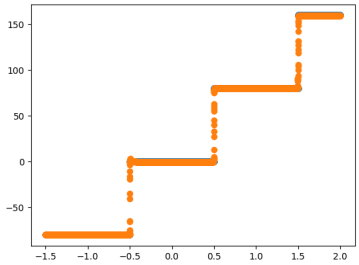
Zbiór danych	Architektura	Train MSE	Test MSE
square-large	[1, 4, 4, 4, 1]	0,71	126,38
steps-large	[1, 5, 5, 5, 5, 1]	2,58	1,83
multimodal-large	[1, 20, 20, 1]	9,21	4,09

Tabela 4: Osiągnięte wyniki uczenia sieci dla różnych zbiorów danych przy użyciu metody uczenia z momentem

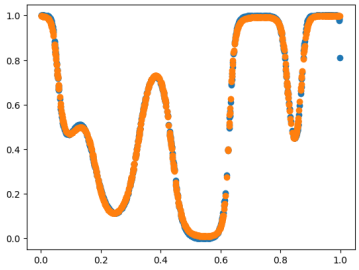
Duży błąd MSE na zbiorze *square-large* wynika ze struktury zbioru testowego, który posiadał większy zakres danych niż zbiór treningowy, co dobrze widać na pierwszym z poniższych wykresów.



Rysunek 10: Przewidywania wytrenowanego modelu (square-large)

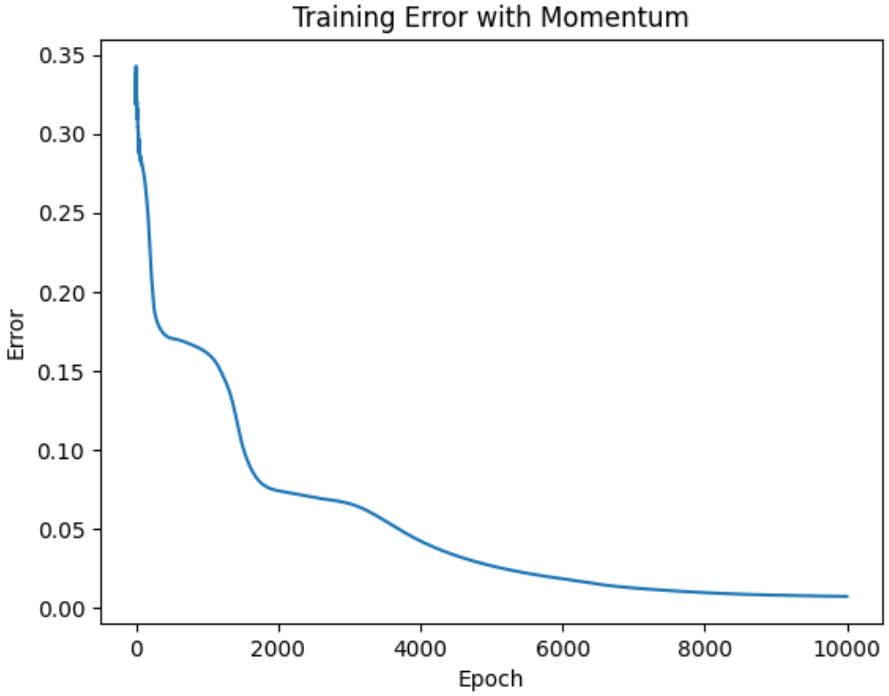


Rysunek 11: Przewidywania wytrenowanego modelu (steps-large)



Rysunek 12: Przewidywania wytrenowanego modelu (multimodal-large)

Przy użyciu tej metody można było zaobserwować dość szybki spadek błędu przy uczeniu się sieci.



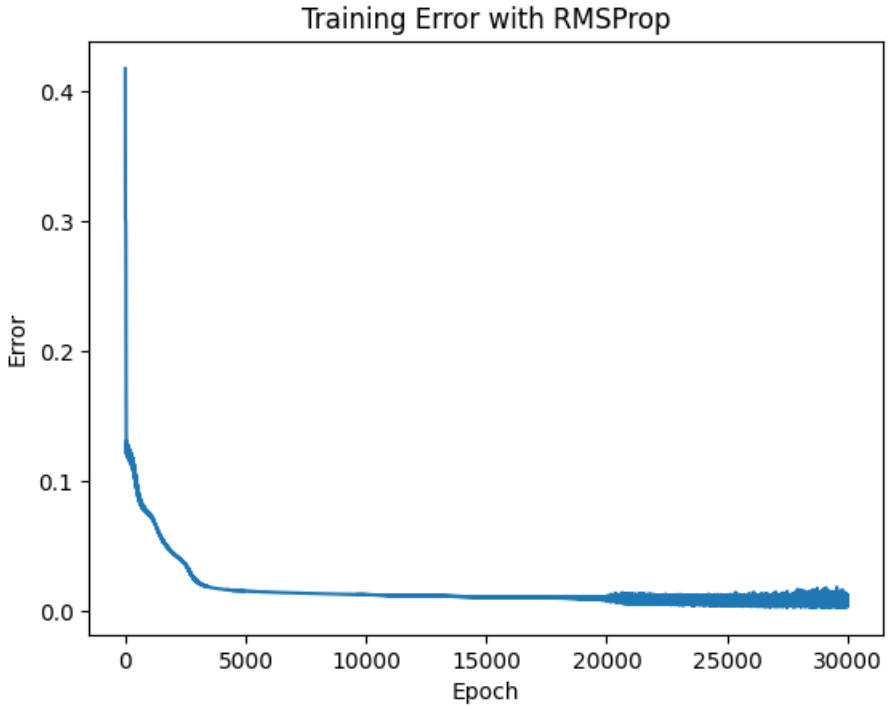
Rysunek 13: Wykres błędu od liczby epok dla multimodal-large, metoda uczenia z momentem

Drugim testowanym wariantem ucznia był RMSProp. Poniżej można zaobserwować osiągnięte wyniki.

Zbiór danych	Architektura	Train MSE	Test MSE
square-large	[1, 4, 4, 4, 1]	4,14	305
steps-large	[1, 5, 5, 5, 5, 1]	38	31
multimodal-large	[1, 20, 20, 1]	228	250

Tabela 5: Osiągnięte wyniki uczenia sieci dla różnych zbiorów danych przy użyciu metody RMSProp

Można zaobserwować, że metoda ta poradziła sobie znacznie gorzej na zbiorze multimodal-large. Mimo, że trenowana była na większej liczbie epok, nie zdołała osiągnąć tak dobrego wyniku jak uczenie z momentem. To co można było zobaczyć, to też mniejsza stabilność tej metody w dalszych etapach uczenia (wykres staje się pod koniec dość "poszarpany")



Rysunek 14: Wykres błędu od liczby epok dla steps-large, metoda RMSProp

2.3.3 Wnioski

Algorytmy usprawnienia uczenia gradientowego mogą znacznie poprawić szybkość i efektywność uczenia się sieci neuronowej MLP. Z dwóch testowanych algorytmów, na wszystkich zbiorach danych, uczenie z momentem wypadło znacznie lepiej niż RMSProp.

2.4 Laboratorium 4.

2.4.1 Opis zadania

Zadanie polegało na przygotowaniu swojej implementacji perceptronu do problemu klasyfikacji przy użyciu funkcji *softmax*. Oraz porównanie działania sieci używającej softmax i zwykłej funkcji aktywacji (sigmoid). Gotowy model należało przetestować na trzech zbiorach danych:

- rings3-regular
- easy
- xor3

oraz osiągnąć wymaganą wartość F-measure (F1 score).

$$F_1 = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}$$

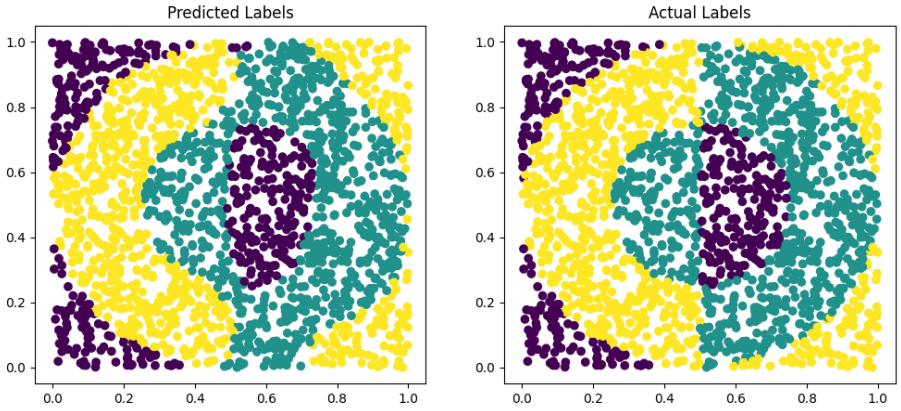
2.4.2 Wyniki eksperymentu

Zaimplementowanie użycia funkcji softmax wymagało zmiany struktury sieci, aby poprawnie wyliczała się pochodna przy propagacji wstecznej gradientu. Należało również dokonać one-hot enkodowania zmiennej celu oraz dostosowanie architektury sieci tak, aby liczba neuronów w warstwie wyjściowej równała się liczbie klas w klasyfikacji. Jako odpowiedź klasyfikacji brany zostaje ten neuron z warstwy wyjściowej, który ma najwyższą wartość. Poniżej zobaczyć możemy osiągnięte wyniki.

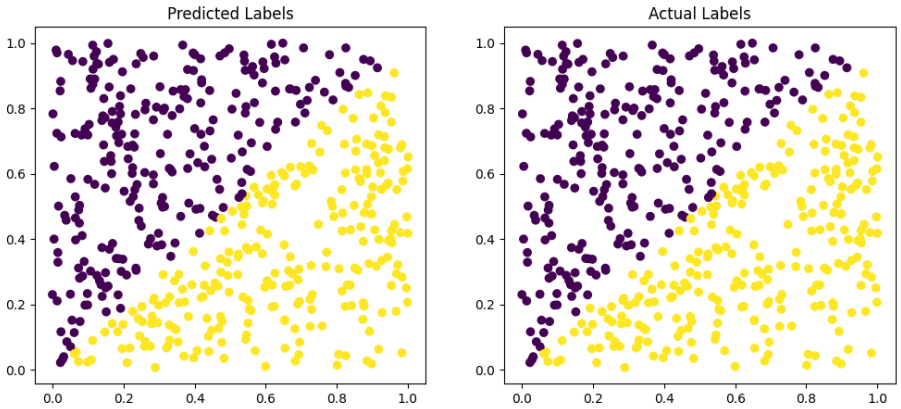
Zbiór danych	Architektura	Train F1	Test F1	Train F1 (softmax)	Test F1 (softmax)
rings3	[2, 10, 3]	0,87	0,84	0,96	0,94
easy	[2, 10, 2]	0,994	0,995	0,998	0,998
xor3	[2,12,12,2]	0,94	0,92	1,0	0,97

Tabela 6: Osiągnięte wyniki uczenia sieci dla różnych zbiorów danych

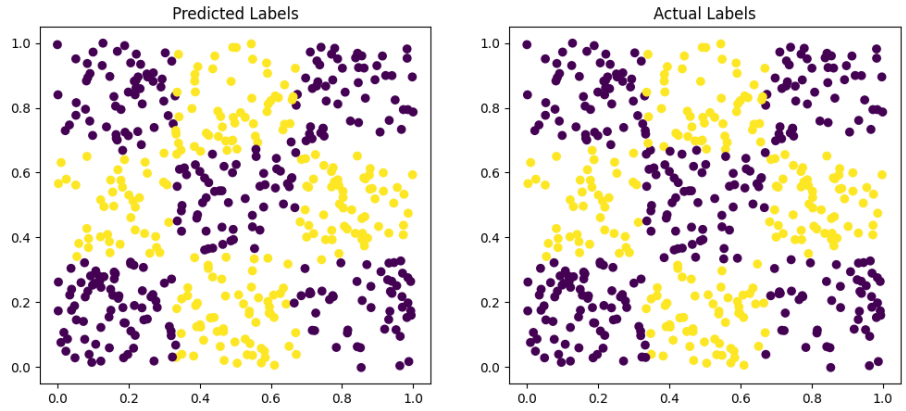
Jak widać, użycie funkcji softmax znacznie poprawiło wyniki.



Rysunek 15: Przewidywania sieci na zbiorze danych rings3 (softmax)



Rysunek 16: Przewidywania sieci na zbiorze danych easy (softmax)



Rysunek 17: Przewidywania sieci na zbiorze danych xor3 (softmax)

2.4.3 Wnioski

Perceptrony wielowarstwowe zdecydowanie nadają się do zadania klasyfikacji, przy czym użycie funkcji softmax znacznie poprawia jej wyniki w porównaniu z sigmoidalną funkcją aktywacji na wyjściu. Sieć radzi sobie z wielowymiarowymi, wieloklasowymi zadaniami klasyfikacji.

2.5 Laboratorium 5.

2.5.1 Opis zadania

Zadanie polegało na rozszerzeniu implementacji sieci neuronowej o 4 funkcje aktywacji:

- sigmoidalną,
- liniową,
- tanh,
- ReLU.

Następnie należało porównać skuteczność modelu dla różnych architektur i różnych funkcji aktywacji

2.5.2 Wyniki eksperymentu

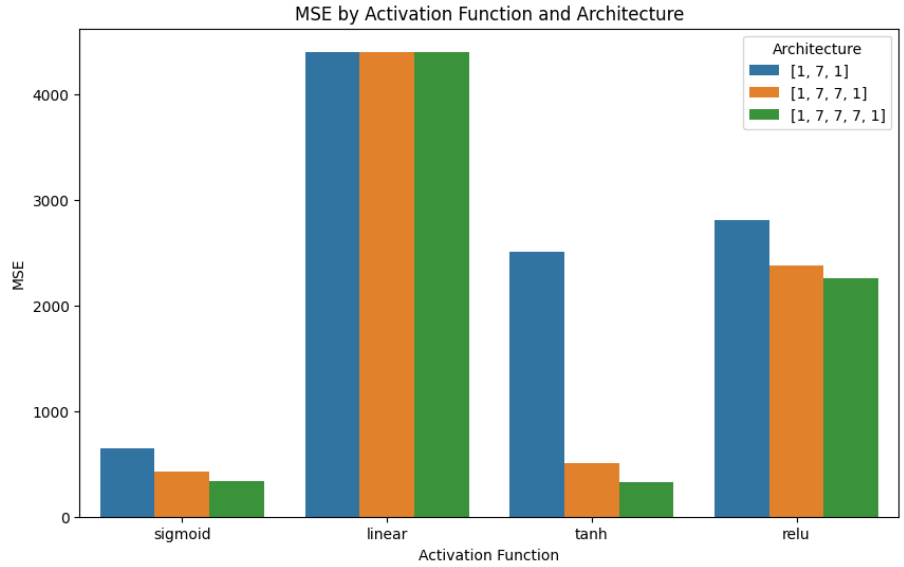
Na początek wykonany został test na zbiorze *multimodal-large*: Dla każdej funkcji aktywacji wytrenowane zostały trzy sieci o architekturach o jednej, dwóch i trzech warstwach ukrytych, przy czym każda z tych warstw zawierała po 7 neuronów. Następnie zostały wyłonione dwie pary (funkcja aktywacji, architektura) i zostały one poddane testom na trzech innych zbiorach:

- steps-large
- rings5-regular
- rings3-regular

Wyniki pierwszego, przesiewowego testu wyłoniły jako zwycięskie:

- ★ funkcja *tanh* dla trzech warstw ukrytych
- ★ funkcja *sigmoid* dla trzech warstw ukrytych

Wszystkie wyniki zobaczyć można na wykresie poniżej. Jak widać najgorzej poradziła sobie funkcja liniowa, a najbardziej uniwersalną funkcją wydaje się być *sigmoid*.



Rysunek 18: Wyniki testów dla różnych architektur i funkcji aktywacji

Następnie dla dwóch wybranych par architektura - funkcja aktywacji zostały przeprowadzone testy na 3 pozostałych zbiorach:

funkcja aktywacji	**Steps MSE**	**Rings5 F1**	**Rings3 F1**
sigmoid	339,5	**0,95**	**0,91**
tanh	**261,6**	0,61	0,48

Tabela 7: Osiągnięte wyniki uczenia sieci dla różnych konfiguracji sieci i zbiorów danych

W zadaniu regresji (steps) lepiej poradziła sobie siec z funkcją aktywacji *tanh*, jednak w zadaniu klasyfikacji zdecydowanie lepsze wyniki daje sieć z funkcją aktywacji *sigmoid*

2.5.3 Wnioski

Zarówno różne architektury, jak i różne funkcje aktywacji sprawdzają się dla różnych rodzajów problemów. Architekturę należy zatem dobierać do zadania indywidualnie, gdyż nie ma jednej najlepszej funkcji aktywacji ani architektury. Najbardziej uniwersalną funkcją aktywacji, przynajmniej dla danych z powyższych zadań, wydaje się być funkcja sigmoid, jednak nie osiąga ona najlepszych wyników w każdym przypadku.

2.6 Laboratorium 6.

2.6.1 Opis zadania

Celem zadania było zaimplementowanie dwóch metod działających przeuczeniu się modelu:

- early-stopping (zatrzymanie uczneia, gdy błąd na zbiorze walidacyjnym zaczyna rosnąć)
- regularyzacja (karanie za zbyt duże wagi sieci)

Następnie należało przetestować te metody na czterech zbiorach danych:

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

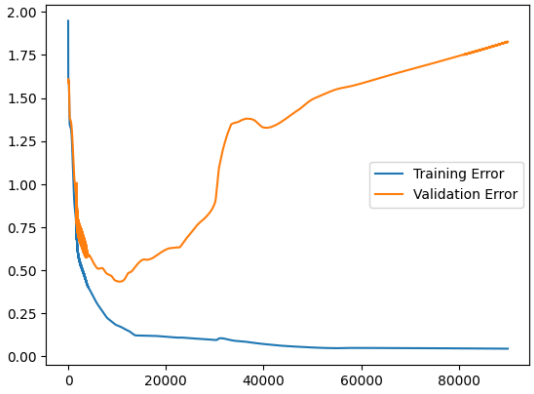
2.6.2 Wyniki eksperymentu

Zaimplementowana została regularyzacja l2 nakładająca karę za zbyt duże wagi sieci, poprzez pomniejszenie ich wpływu na dalszą naukę. Wprowadzony został również mechanizm early stoppingu, który przerywa naukę sieci, gdy błąd na zbiorze uczącym rósł przez zadaną liczbę epok wstecz. W poniższej tabeli można zobaczyć wyniki testów.

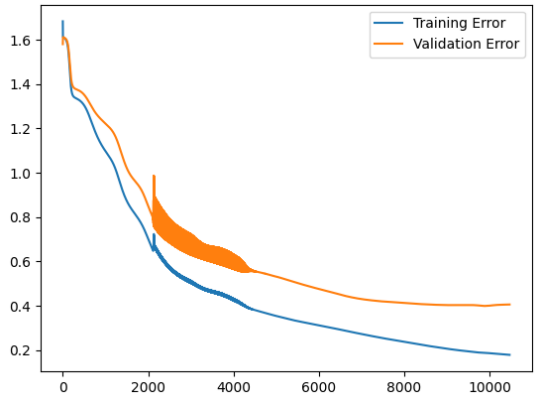
Przeciwdziałanie przeuczeniu	Multimodal train MSE	Multimodal test MSE	Rings5 train F1	Rings5 test F1	Rings3 train F1	Rings3 test F1	Xor3 train F1	Xor3 test F1
-	278	673	0,98	0,76	0,90	0,81	0,98	**0,89**
early-stopping	272	**606**	0,88	**0,80**	0,93	**0,82**	0,88	0,74
regularyzacja	255	620	0,97	0,78	0,89	**0,82**	0,97	0,85

Tabela 8: Osiągnięte wyniki uczenia sieci dla różnych zbiorów danych i metod przeciwdziałania przeuczeniu

Jak widać metody przeciwdziałania przeuczeniu poprawiły wyniki na zbiorze testowym w większości przypadków. Na poniższych wykresach można zauważyć, jak early stopping zapobiegł przeuczeniu sieci w porównaniu z normalnym uczeniem:



Rysunek 19: Wykres błędu od liczby epok dla rings5 (normalny trening)



Rysunek 20: Wykres wartości wag od liczby epok dla rings5 (early stopping)

2.6.3 Wnioski

Zarówno regularyzacja jak i early stopping okazują się być skutecznymi metodami zapobiegania przeuczenia się sieci neuronowej. Nie dość, że oba polepszają wyniki treningu, to early stopping dodatkowo może znacznie skrócić czas treningu. Należy jednak uważać, gdyż early stopping może zatrzymać trening zbyt wcześnie.

3 Podsumowanie

W trakcie wykonywania wszystkich zadań powstała kompletna implementacja sieci neuronowej MLP z wieloma usprawnieniami, takimi jak usprawnienia gradientowe, poprawa działania klasyfikacji, czy przeciwdziałania przeuczeniu. Mimo swojej złożoności i długiego treningu, sieć taka okazuje się być dobrym rozwiązaniem do różnorodnych zadań uczenia maszynowego.