Perceptron wielowartwowy - Metody inteligencji obliczeniowej w analizie danych

Piotr Bielecki,nr albumu 313320

1 Wstęp

1.1 Cel

Raport poświęcony jest opracowaniu wyników pracy z przedmiotu "Metody inteligencji obliczeniowej w analizie danych. W poniższej pracy poruszone są tematy:

- perceptronu wielowarstwowego z dobieraną architekturą
- propagacja wstecznej błędu
-

2 NN1 - bazowa implementacja

2.1 cel ćwiczenia

celem tej pracy domowej, było stworzenie podstawowej wersji perceptronu wielowarstwowego, w którym można wybrać:

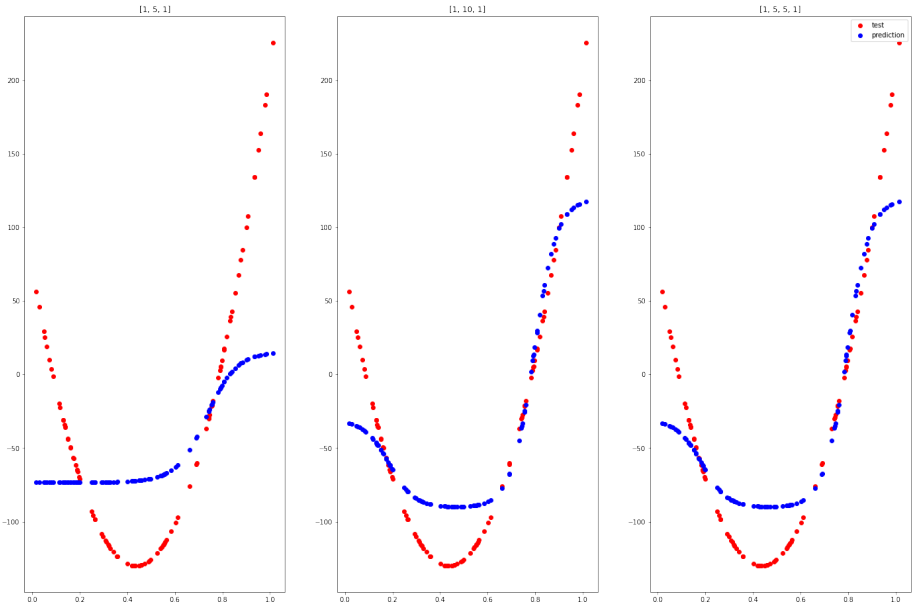
- architekturę wejścia
- funkcję aktywacji

należało także sprawdzić działanie sieci dla różnych architektur:

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

2.2 co to perceptron wielowarstwowy?

Perceptron wielowarstwowy to rodzaj sieci neuronowej, składającej się z co najmniej trzech warstw neuronów: wejściowej, ukrytej i wyjściowej. Każda warstwa składa się z wielu neuronów połączonych - każde z każdym - wewnątrz sąsiadujących warstw, a połączenia te mają przypisane wagi, które są dostosowywane w trakcie uczenia sieci.



Rysunek 1: od lewej, 5 neuronów - 1 warstwa, 10 neuronów - 1 warstwa, 5 neuronów - 2 warstwy.

2.3 opracowanie, wnioski, wyniki

Architektura z pojedynczą warstwą ukrytą z 5 neuronami jest zbyt mała i nie pozwala sieci na odpowiednie dopasowanie się do danych. Większe architektury są w stanie "dogiąć się" - większa liczba neuronów pozwala sieci na wychwycenie bardziej złożonych wzorców w danych.

3 NN2 - Implementacja propagacji wstecznej błędu

3.1 Cel ćwiczenia

- implemntacja propagacji wstecznej błędu i sprawdzenie jej działania
- następnie należało zaimplementować metodę wizualizacji wartości wag sieci w kolejnych iteracjach
- 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

3.2 opracowanie, wnioski, wyniki

wielkość batch ma wpływ na sposób, w jaki sieć jest uczona poprzez regulowanie kroku uczenia (ang. learning rate) i zmniejszanie zmienności gradientu. Batch to porcja przykładów uczących przetwarzanych jednocześnie przez sieć neuronową w jednej iteracji procesu uczenia. Proces uczenia w porcjach działa poprzez uśrednienie błędu z całego batcha przed propagacją. Dzięki temu można zwiększyć stabilność i skuteczność uczenia się sieci, ponieważ metoda ta pomaga zredukować zmienność gradientu. W praktyce, większe batche zazwyczaj prowadzą do szybszego uczenia się sieci, ale mniejsze batche mogą prowadzić do lepszej jakości wyników, szczególnie w przypadku zbiorów danych o wysokiej zmienności, gdzie utracilibyśmy pewne informacje w procesie uśredniania.

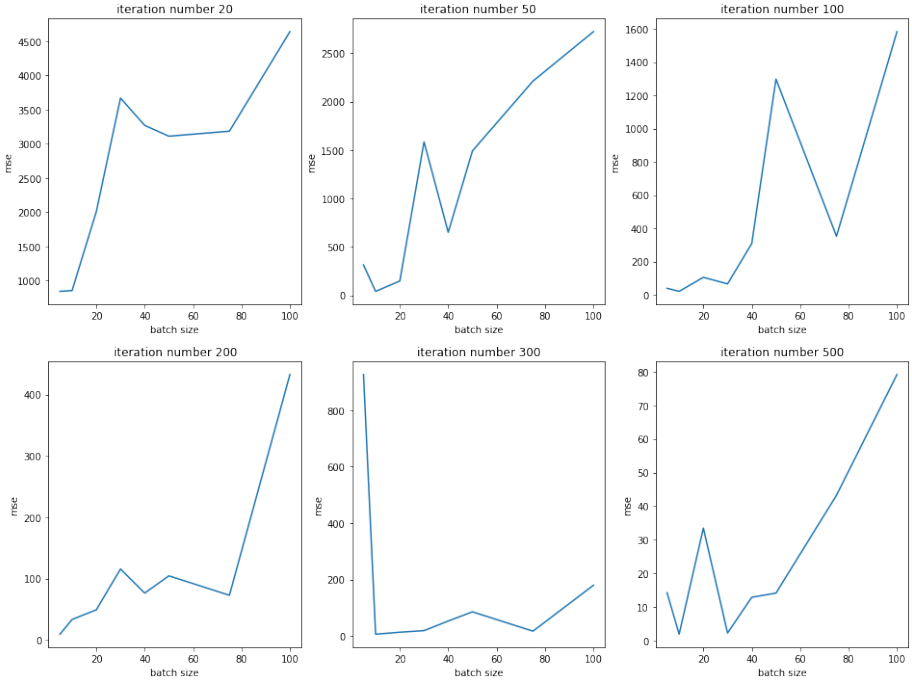
Gdy przez sieć przepuszczamy całą porcję danych, aby dokonać jednej modyfikacji wag, taki proces nazywamy "gradient descent". Gdy jednak porcja jest mniejsza niż całość danych, mówimy o "stochastic gradient descent".

square-simple	steps-small	multimodal-large
0.34199	23.5145	2.3422

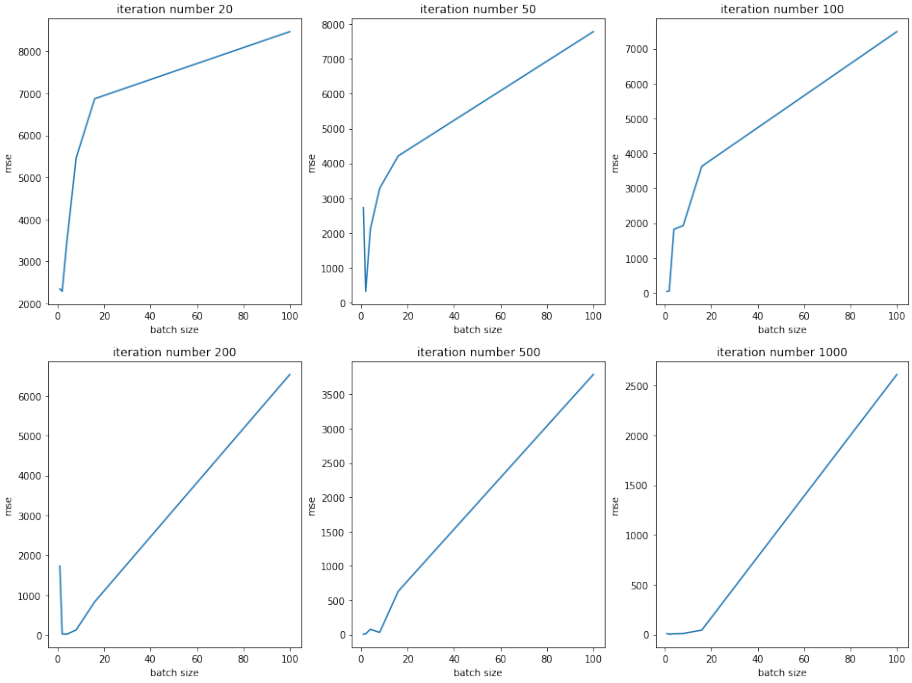
Tabela 1: Wyniki już po doborze batch_size

3.2.1 Wpływ batch-size

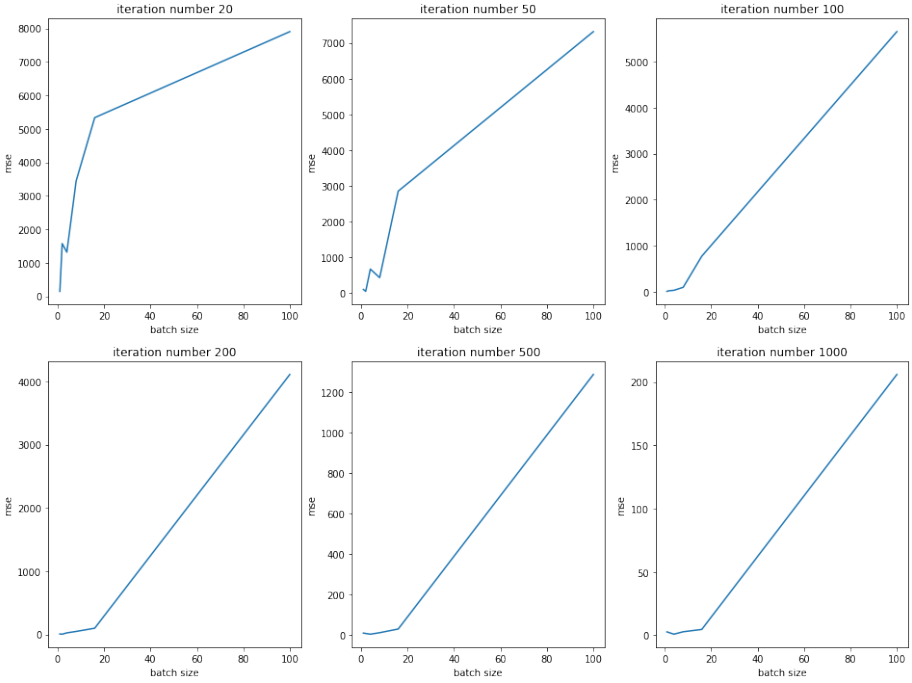
Poniższe wykresy prezentują jak wielkość porcji danych wpływa na jakość uczenia. Jak widać, najlepiej modele zachowują się przy wielkości porcji na poziomie kilku(nastu) obserwacji.



Rysunek 2: dataset - square-simple



Rysunek 3: dataset - steps-small



Rysunek 4: dataset - multimodal

4 NN3 - Implementacja momentu i normalizacji gradientu

4.1 cel ćwiczenia

w tej pracy domowej należało zaimplementować usprawnienia uczenia gradientowego

- metoda momentum
- metoda RMSprop

Należało również porównać szybkość zbieżności procesu uczenia dla tych wariantów, i dla wariantu bez usprawnień.

4.2 opracowanie, wnioski, wyniki

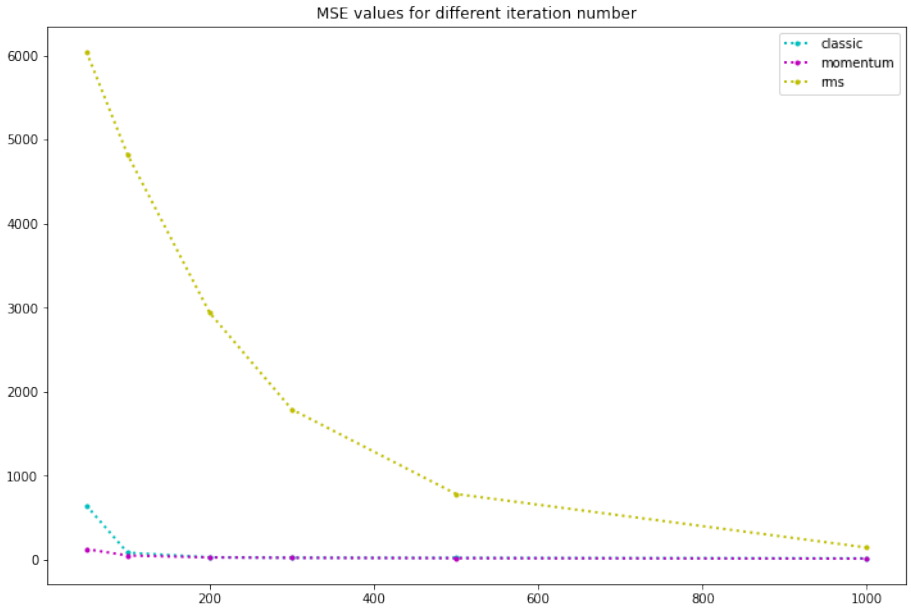
Metoda momentów pomaga w poruszaniu się po powierzchni funkcji celu, przyspieszając proces uczenia się i unikając zbyt wielkich kroków, które mogą prowadzić do oscylacji i stagnacji w procesie uczenia się. W metodzie tej wprowadza się pojęcie pędu (podobnego do tego fizycznego). Pęd służy do przyspieszenia procesu uczenia się, ponieważ gradienty poprzednich kroków są brane pod uwagę przy aktualizacji wag.

Algorytm RMSprop, z kolei, polega na adaptacyjnym zmniejszaniu kroku uczenia na podstawie pierwiastka ze średniego kwadratu gradientów. Metoda ta zapobiega zbyt dużym aktualizacjom wag w przypadku dużych gradientów, co prowadzi do szybszego i bardziej stabilnego procesu uczenia się.

W obu algorytmach wymagany jest dodatkowy hiperparametr kontrolujący wielkość wpływu metody na adaptacje wag.

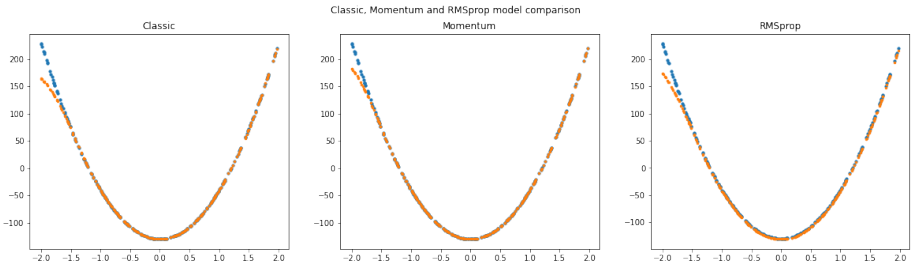
problem	metoda klasyczna	metoda Momentum	metoda RMSprop
square-large	108.185	49.716	85.089
steps-large	8.138	2.998	19.231
multimodal	0.656	1.442	10.528

Tabela 2:

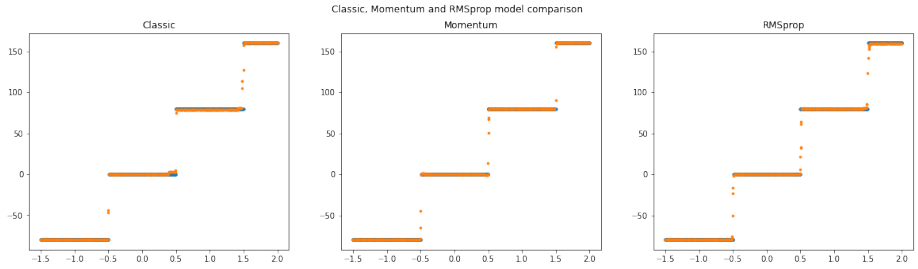


Rysunek 5: zachowanie MSE dla jednego z problemów przy zastosowaniu różnych metod

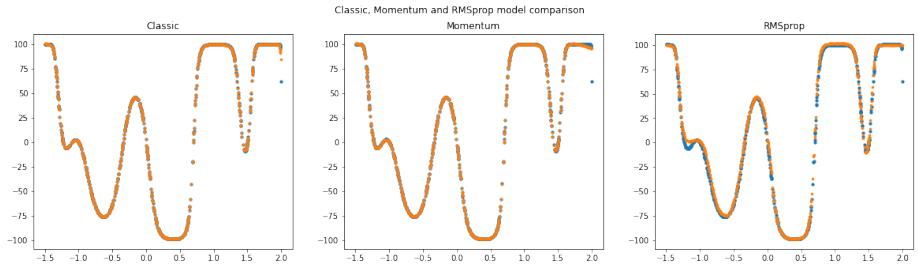
Poniżej wykresy pokazujące dopasowanie poszczególnych modeli do danych



Rysunek 6: dataset - square



Rysunek 7: dataset - steps



Rysunek 8: dataset - multimodal

Metoda momentum jednogłośnie zwycięża w tej konkurencji, według przeprowadzonych testów prowadzi do najlepszych wyników, jak i zbiega najszybciej.

5 NN4 - Rozwiązywanie zadania klasyfikacji

5.1 cel ćwiczenia

W czwartej pracy domowej należało:

- Zaimplementować funkcję softmax dla warstwy wyjściowej sieci neuronowej
- porównać jej działanie z wersją z liniową funkcją wyjścia

należało także wprowadzić zmiany uwzględniające pochodną funkcji wyjścia, natomiast okazało się to niekonieczne, ponieważ przy odpowiednim zaimplementowaniu one_hot_encodingu pochodna nie zmienia postaci. (link do artykułu)

Funkcja softmax jest często używana jako ostatnia warstwa w sieciach neuronowych do przekształcenia wartości wyjściowych na wartości "prawdopodobieństw" (nie są to prawdopodobieństwa per se). Działanie funkcji softmax polega na przekształceniu wektora liczb rzeczywistych na wektor, w którym suma

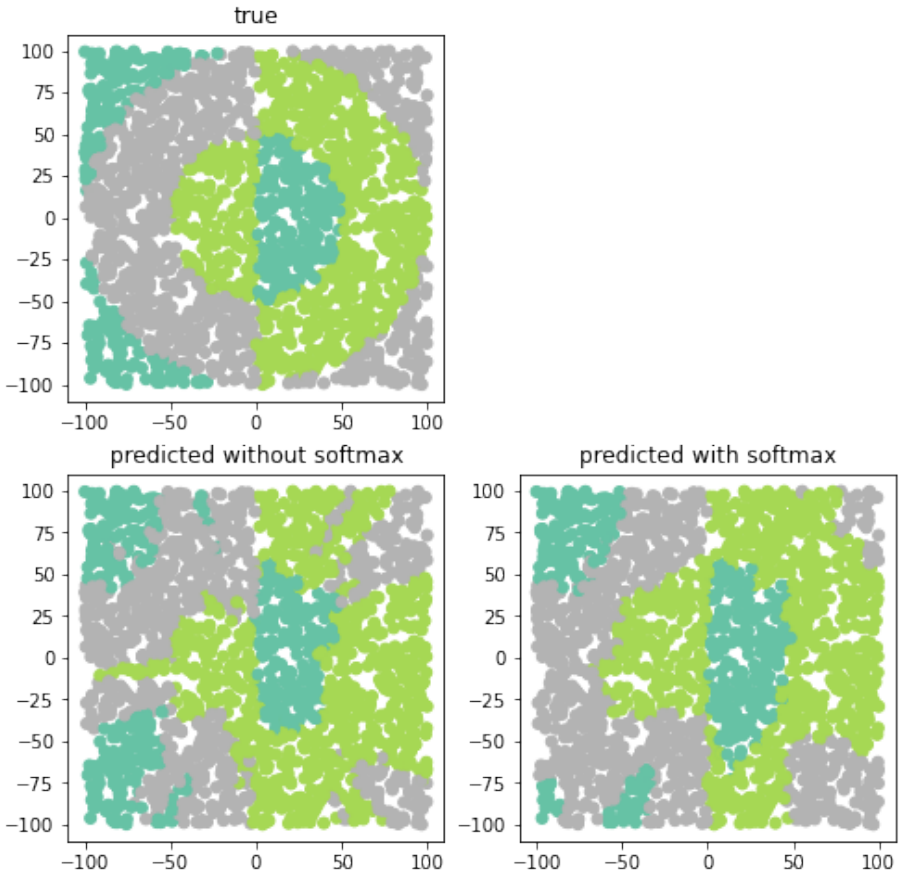
wszystkich wartości w wektorze wynosi 1. Konkretnie, jeśli mamy wektor wartości rzeczywistych o wymiarze n, to wartości te są przekształcane w wektor prawdopodobieństw o wymiarze n zgodnie z poniższym wzorem:

$$softmax(x_i) = \frac{exp(x_i)}{\sum_{j=1}^{n}(exp(x_j))} \tag{1}$$

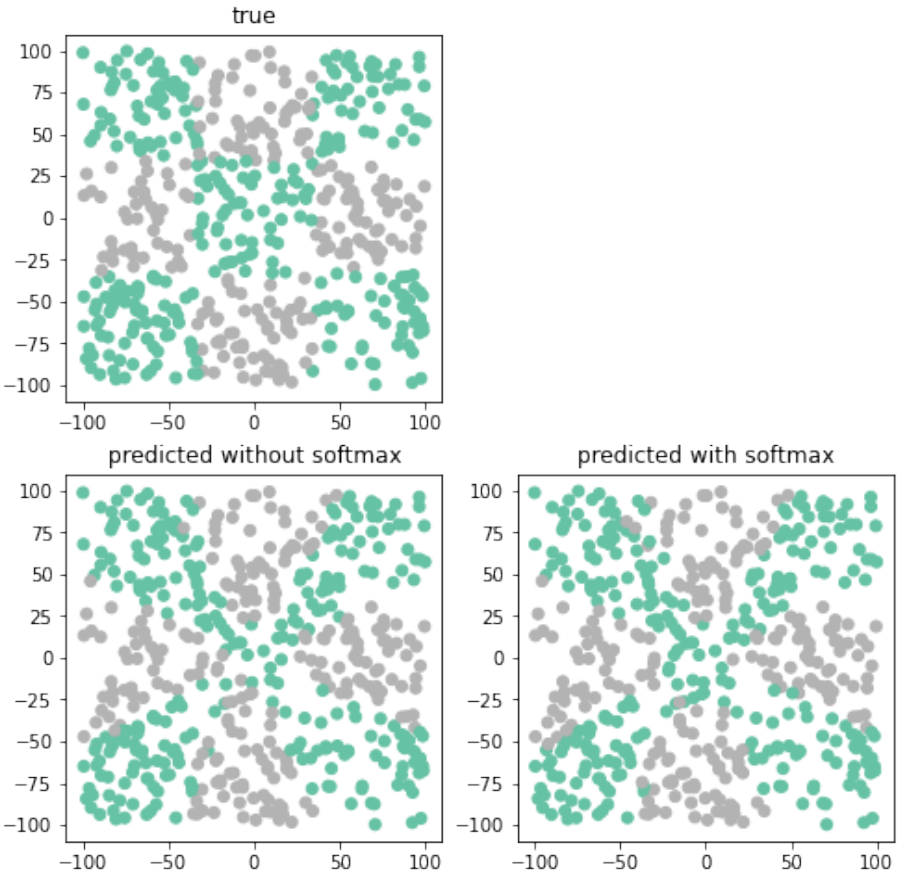
Problem	Bez Softmax	Softmax
Rings	0.7675	0.8005
Easy	0.998	0.998
XOR3	0.833	0.836

Tabela 3: wartość f-score

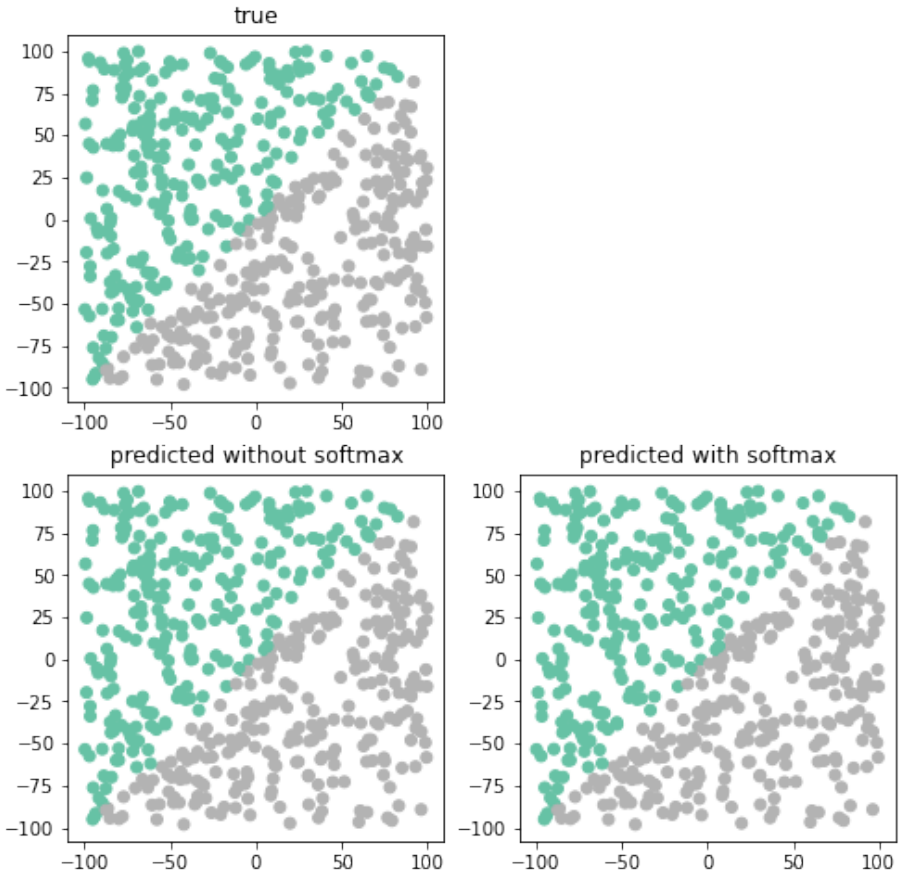
Poniższe wykresy przedstawiają porównania klasyfikacji z funkcją liniową na wyjściu sieci, jak i z wykorzystaniem funkcji softmax wykorzystaniem.



Rysunek 9: dataset - rings



Rysunek 10: dataset - xor



Rysunek 11: dataset easy

6 NN5 - Testowanie różnych funkcji aktywacji

6.1 cel ćwiczenia

w tej pracy domowej należało rozszerzyć implementację o różne funkcje aktywacji:

- sigmoid
- liniowa
- tanh
- ReLU

oraz porównać szybkość uczenia w zależności od funkcji aktywacji.

dla zbioru multimodal przeprowadzić analizę wszystkich czterech funkcji aktywacji dla architektur (zrozumiałem z treści polecenia, że mają to być architektury jak na pierwszej pracy domowej, dlatego wyniki nie powalają), dla pozostałych zbiorów należało wybrać dwa zestawy (architektura + funkcja aktywacji) i porównać ich wyniki.

6.2 opracowanie, wnioski, wyniki

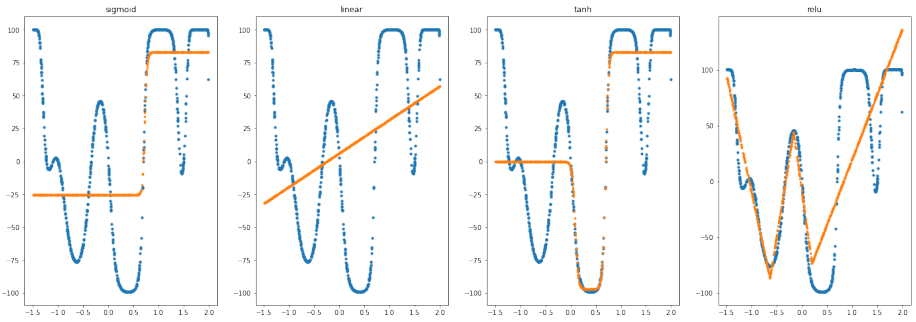
architektura	sigmoid	liniowa	tanh	ReLU
10 neuronów, 1 warstwa	2629	4440	1655	1640
5 neuronów, 2 warstwy	2832	4434	2719	1513
10 neuronów, 3 warstwy	2624	4441	1112	127

Tabela 4: Wyniki dla problemu multimodal(MSE)

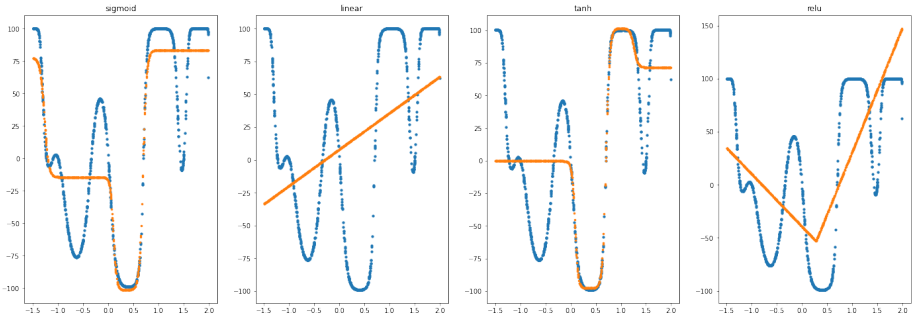
problem	tanh	ReLU
steps-large	27	457
rings5-regular	0.547	0.6565
rings3	0.571	0.8715

Tabela 5: Wyniki dla pozostałych problemów (MSE lub F1)

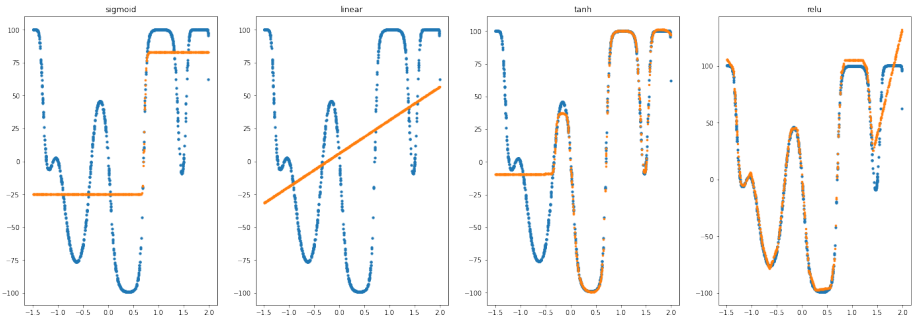
Architektury były mniejsze niż w poprzednich pracach domowych, dlatego wyniki są niższe. Poniżej porównanie dopasowań do danych testowych.



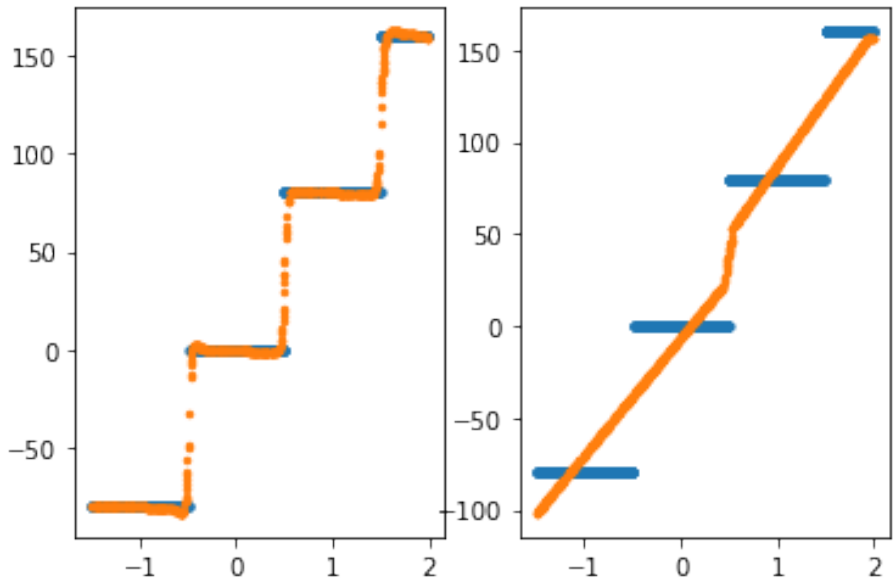
Rysunek 12: dataset - multimodal, 5 neuronów, 1 warstwa



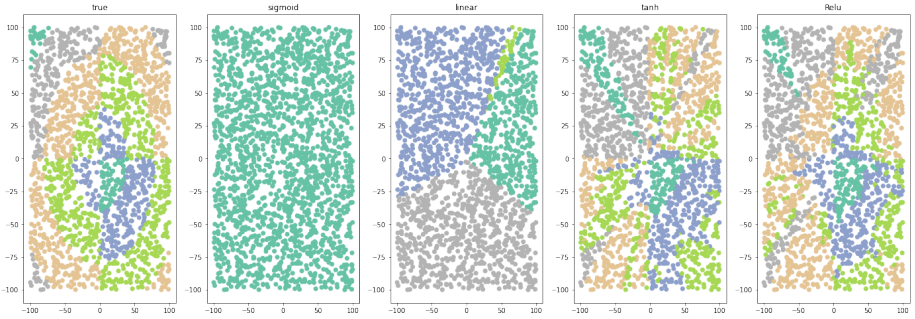
Rysunek 13: dataset - multimodal, 10 neuronów, 1 warstwa



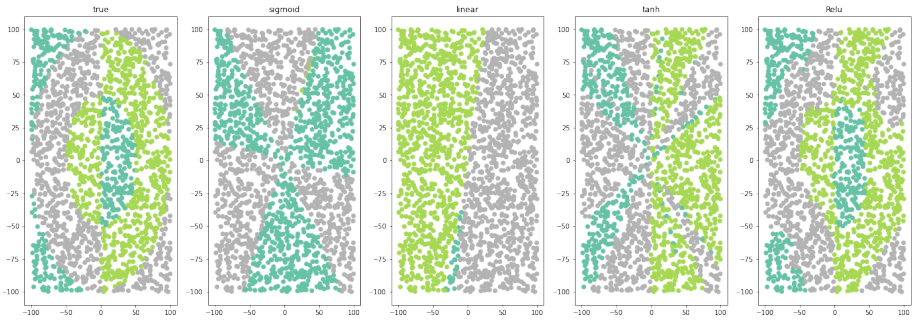
Rysunek 14: dataset - multimodal, 10 neuronów, 3 warstwy



Rysunek 15: dataset - steps, tanh po lewej, ReLU po prawej



Rysunek 16: dataset - rings 5, porównanie różnych f. aktywacji



Rysunek 17: dataset - rings 3,

Liniowa funkcja aktywacji nie jest zazwyczaj stosowana w warstwach ukrytych sieci neuronowych, ponieważ nie wprowadza nieliniowości do sieci. W związku z tym, sieć neuronowa z liniową funkcją aktywacji byłaby równoważna z jedną warstwą liniową (regresja liniowa).

Funkcja sigmoid jest wykorzystywana w sieciach w problemach klasyfikacji binarnej, lub kiedy chcemy ograniczyć wyjścia funkcji do przedziału [0;1]

Funkcja tanh jest podobna do sigmoid, z tym, że jej wyjścia nie ograniczają się do dodatnich, jej wyjściem są liczby z przedziału [-1;1], co pozwala modelowi na większą ekspresję. Bywa wykorzystywana, jeśli chcemy uchwycić bardziej złożone relacje między wejściami i wyjściami sieci.

ReLU jest funkcją, która dla każdego wejścia ujemnego zwróci wartość 0, bywa wykorzystywana w niezbyt skomplikowanych nieliniowych modelach, ponieważ zdaje radzić sobie z nimi lepiej niż sigmoid czy tanh.

7 NN6 - Zjawisko przeuczenia + regularyzacja

7.1 cel ćwiczenia

należało zaimplementować mechanizm regularyzacji wag oraz mechanizm zatrzymywania uczenia przy wzroście błędu na zbiorze walidacyjnym.

7.2 opracowanie, wnioski, wyniki

Regularyzacja wag jest techniką używaną w sieciach neuronowych w celu zapobiegania przeuczeniu modelu. Przeuczenie (overfitting) występuje, gdy sieć neuronowa jest w stanie doskonale dopasować się do danych treningowych, ale nie generalizuje się dobrze na nowe dane. Regularyzacja wag pomaga zminimalizować ryzyko przeuczenia poprzez wprowadzenie pewnego rodzaju kary za duże wartości wag.

Istnieją różne metody regularizacji wag, ale jedną z najpopularniejszych jest L2-regularizacja, która polega na dodaniu do funkcji błędu sieci neuronowej wy-

rażenia opartego na kwadracie normy wektora wag (tzw. norma L2). W ten sposób, wagi są zmuszane do przyjmowania mniejszych wartości, co z kolei zmniejsza skłonność modelu do przeuczenia się.

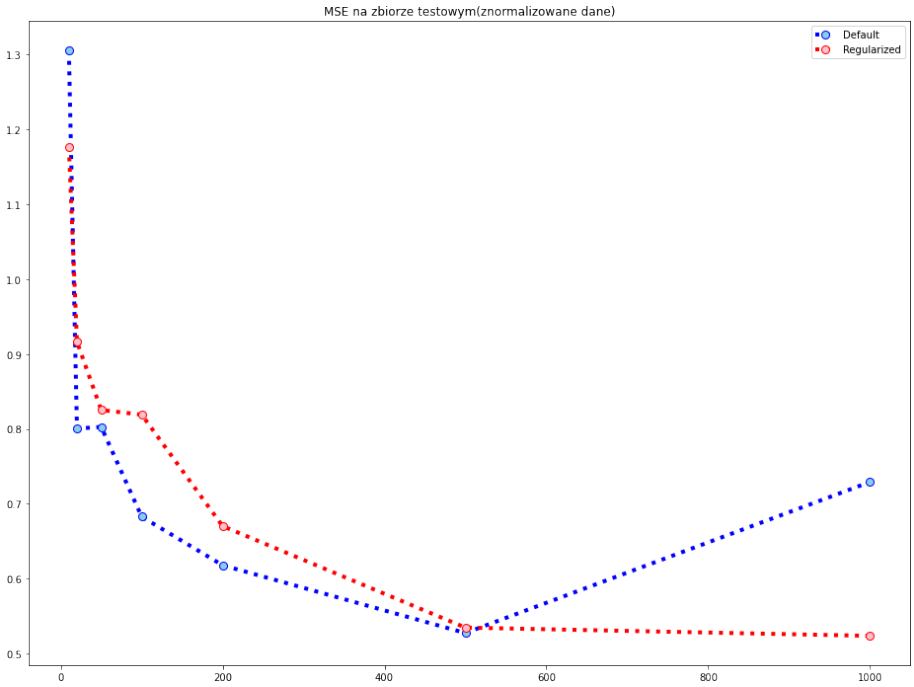
Konkretnie, funkcja błędu sieci neuronowej z zastosowaną L2-regularizacją może być wyrażona następująco:

$$Loss_{regularized} = Loss + \lambda \|w\|^2$$

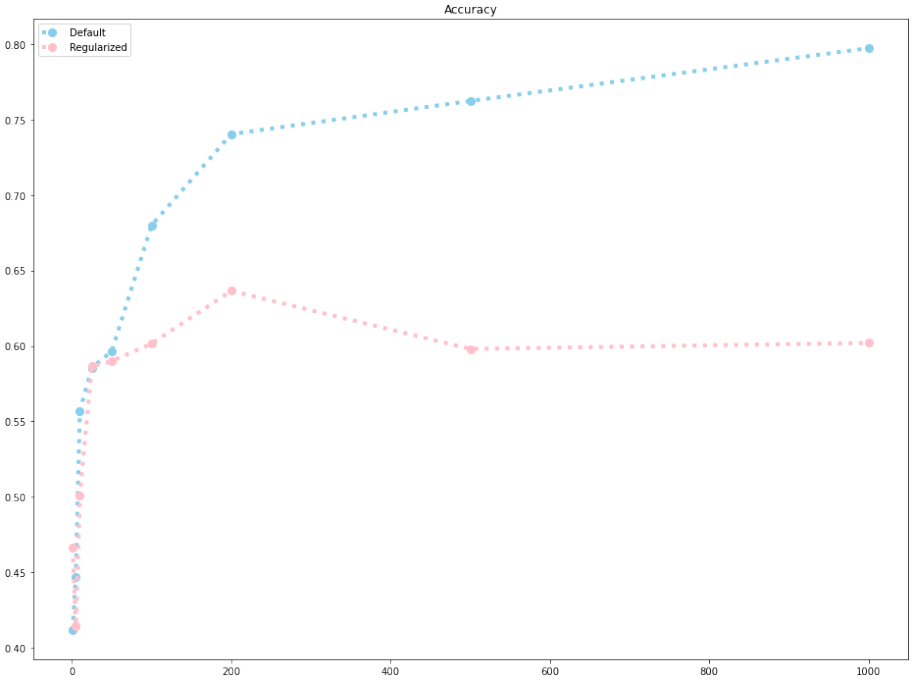
gdzie Loss to pierwotna funkcja kosztu, λ to współczynnik regularyzacji, a $\|w\|^2$ to kwadrat normy wektora wag.

Współczynnik regularyzacji λ jest parametrem, który kontroluje wpływ regularyzacji na wyniki modelu. Im większa wartość λ, tym większy wpływ regularyzacji na wyniki modelu. Z drugiej strony, jeśli λ jest zbyt duże, to model może zbytnio skupić się na zmniejszaniu wag kosztem dokładności.

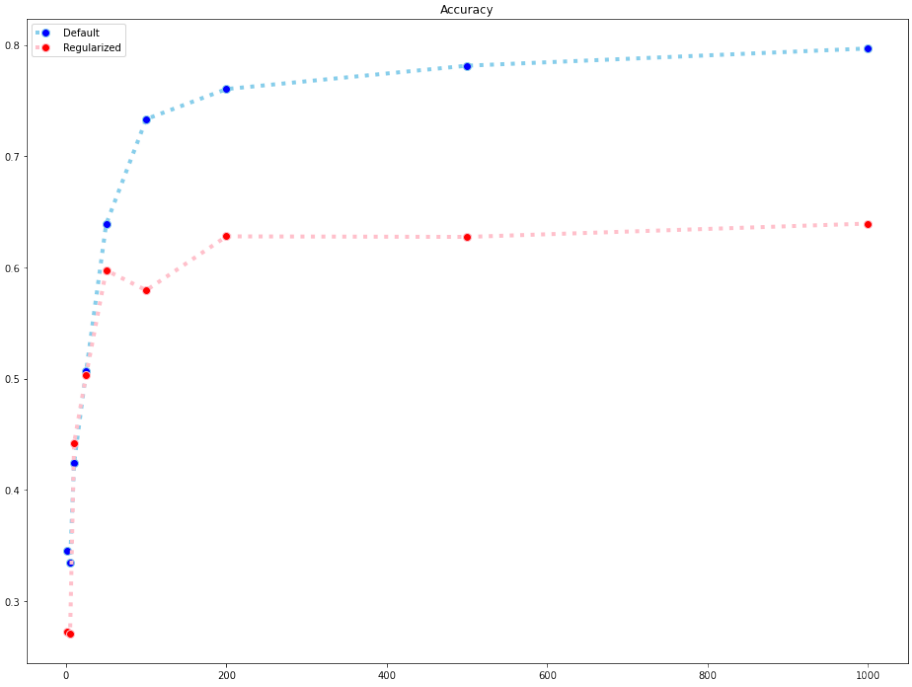
Implementacja regularyzacji wag jest korzystna, ponieważ pomaga zwiększyć zdolność modelu do generalizacji, co jest kluczowe dla skutecznego wykorzystania sieci neuronowych w praktyce.



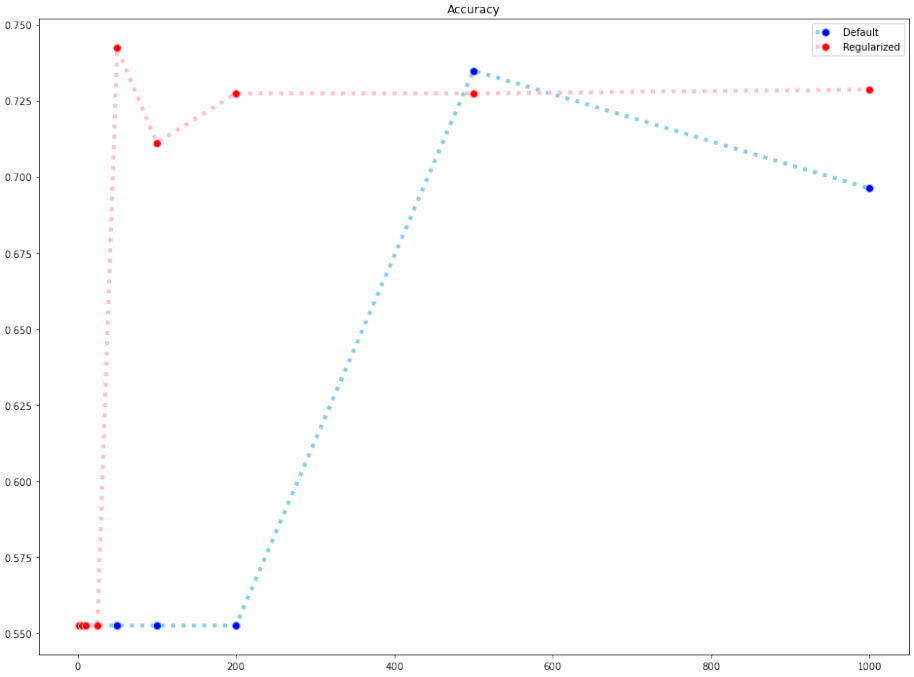
Rysunek 18: dataset - multimodal



Rysunek 19: dataset - rings3



Rysunek 20: dataset - rings5



Rysunek 21: dataset - XOR

- Dla problemu multimodal obie sieci radziły sobie podobnie, natomiast sieć z regularyzacją była w stanie nauczyć się lepiej, niż ta bez regularyzacji
- W przypadku rings-5 (z zaimplementowanym softmax) sieć z regularyzacją poradziła sobie gorzej niż sieć bez regularyzacji. być może jest to efekt specyficznej struktury zbioru
- Rings-3 - Dla tego zbioru, sieć bez regularyzacji osiągnęła delikatnie wyższy wynik, szybkość zbiegania obu sieci była podobna
- XOR-3 - Sieć bez regularyzacji ma problem z nauczeniem się tego zbioru, natomiast sieć z regularyzacją wychodzi z eksperymentu z tarczą.