Sprawozdanie z sieci MLP

Aleksandra Wichrowska, 291140

14 kwiecień 2020

Spis treści

1 Bazowa implementacja

1.1 Krótki opis problemu

Pierwsze zadanie polegało na bazowej implementacji sieci neuronowej typu MLP. Użytkownik na wejściu może podać liczbę warstw, liczbę neuronów na każdej warstwie i wagi poszczególnych warstw. Sieć używa sigmoidalnej funkcji aktywacji, a na wyjściu funkcji liniowej.

1.2 Testy i wizualizacje

Testy przeprowadziłam na dwóch zbiorach danych:

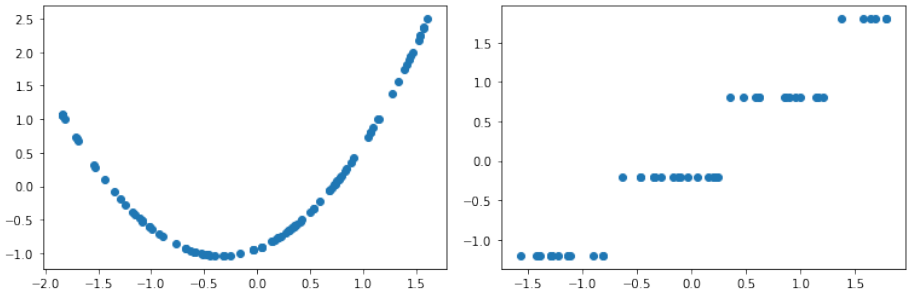
- *square-simple*
- *steps-large*

Dla każdego zbioru rozważałam trzy architektury sieci:

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

Dla pierwszej architektury podjęłam próbę "ręcznej" inicjalizacji wag i biasów dla poszczególnych warstw - na podstawie obserwacji, prób i błedów.

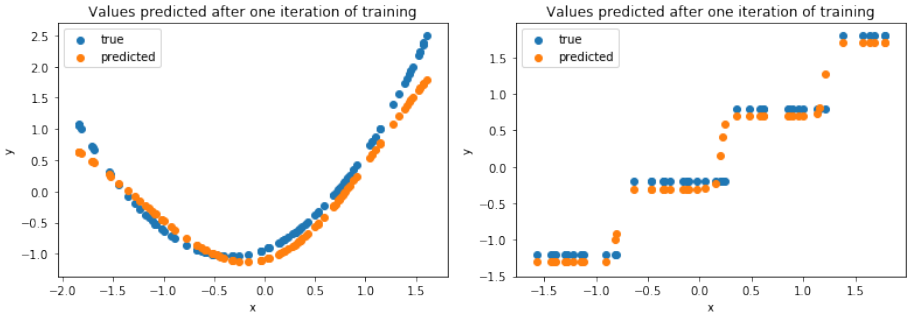
1.2.1 Wizualizacja zbiorów danych



(a) square-simple (b) steps-large

Rysunek 1

1.2.2 Sieć z jedną warstwą ukrytą o 5 neuronach



(a) square-simple (b) steps-large

Rysunek 2

1.3 Wnioski

Po wielu próbach i korygowaniu wartości wag i biasów (16 wartości) uzyskałam zadowalające wyniki. Wartości uzyskane na podstawie wektora wejściowego, macierzy wag i odpowiednich funkcji aktywacji są zbliżone do oczekiwanych (rys. 2).

2 Implementacja propagacji wstecznej błędu

2.1 Krótki opis problemu

Kolejnym etapem prac nad siecią neuronową była implementacja propagacji wstecznej błędu oraz metod inicjalizacji wag. Dzięki temu algorytm potrzebuje na wejściu jedynie danych oraz podania architektury sieci (liczby neuronów na poszczególnych warstwach). Zaimplementowane są dwie wersje aktualizacji wag:

- aktualizacja po prezentacji wszystkich wzorców
- aktulizacja po prezentacji kolejnych porcji (batch)

2.2 Testy i wizualizacje

Testy przeprowadziłam na trzech zbiorach danych:

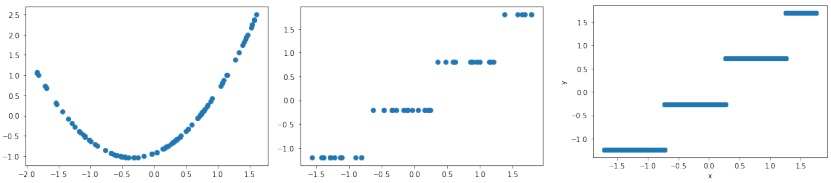
- *square-simple*
- *steps-small*
- *multimodal-large*

Przeprowadzone testy obejmują:

- sprawdzenie poprawności algorytmu (czy sieć neuronowa zbiega do poprawnego wyniku)
- porównanie uczenia na całym zbiorze i z wykorzystaniem mini-batchy
- porównanie metod inicjalizowania wag

Podczas wszystkich testów przyjęłam: $learning\_rate = 0.01$

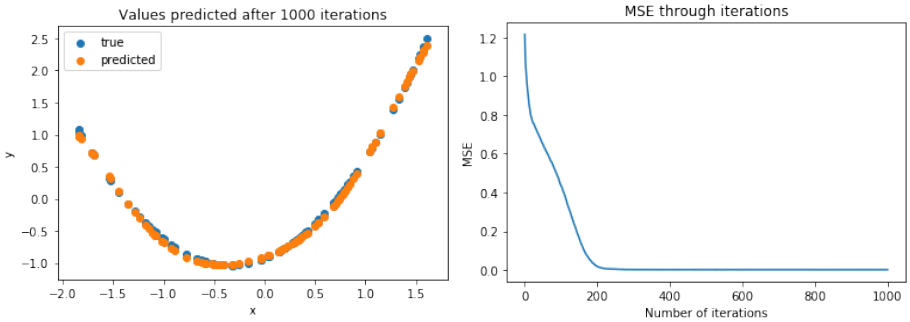
2.2.1 Wizualizacja zbiorów danych



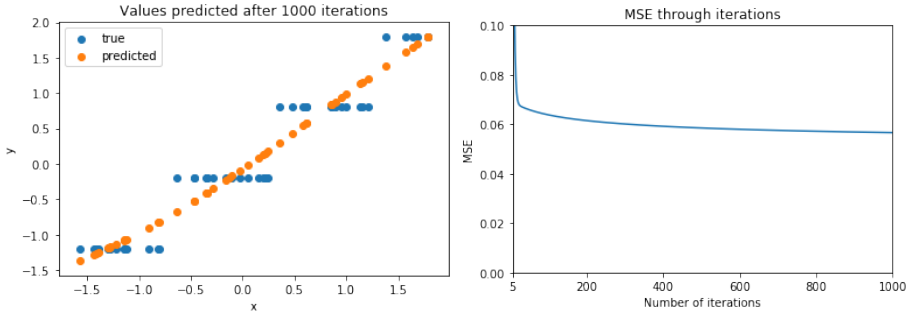
(a) square-simple (b) steps-large (c) multimodal-large

Rysunek 3

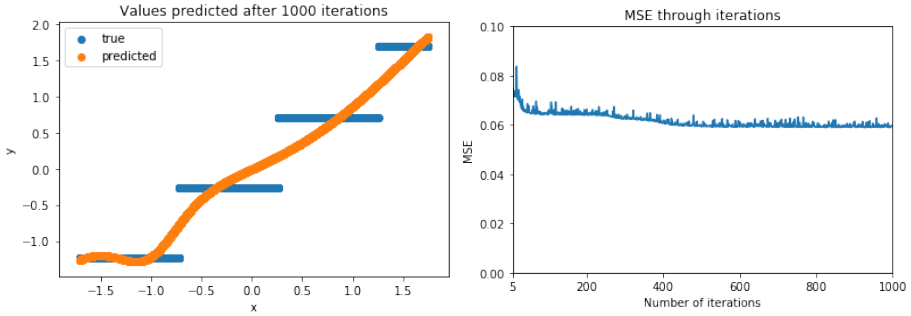
2.2.2 Sprawdzenie poprawności algorytmu



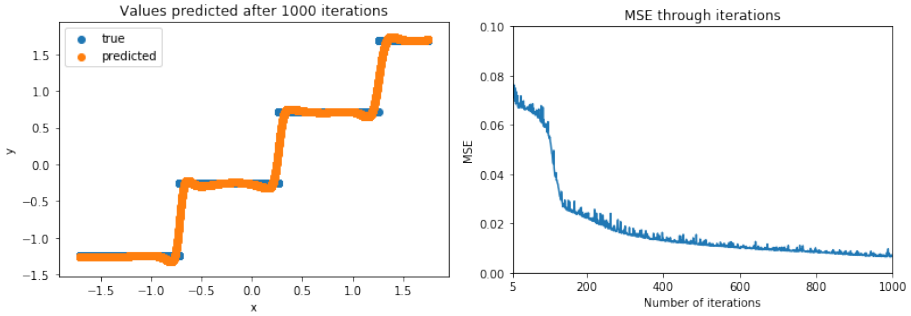
Rysunek 4: Wyniki uzyskane na zbiorze square-simple przez sieć neuronową z jedną warstwą ukrytą o 10 neuronach



Rysunek 5: Wyniki uzyskane na zbiorze steps-small przez sieć neuronową z jedną warstwą ukrytą o 10 neuronach

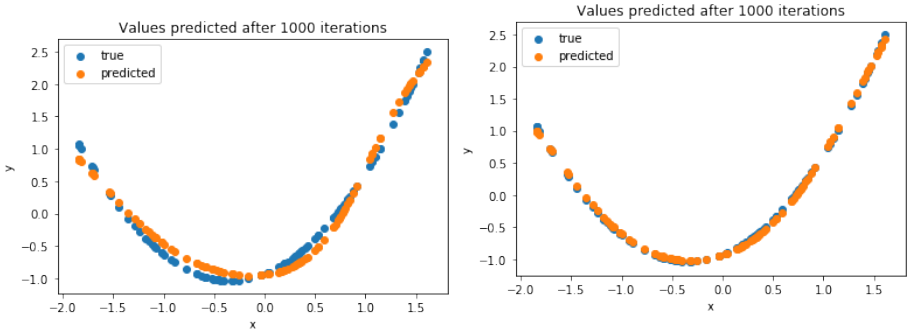


Rysunek 6: Wyniki uzyskane na zbiorze multimodal-large przez sieć neuronową z jedną warstwą ukrytą o 10 neuronach



Rysunek 7: Wyniki uzyskane na zbiorze multimodal-large przez sieć neuronową z dwiema warstwami ukrytymi: o 20 i 10 neuronach

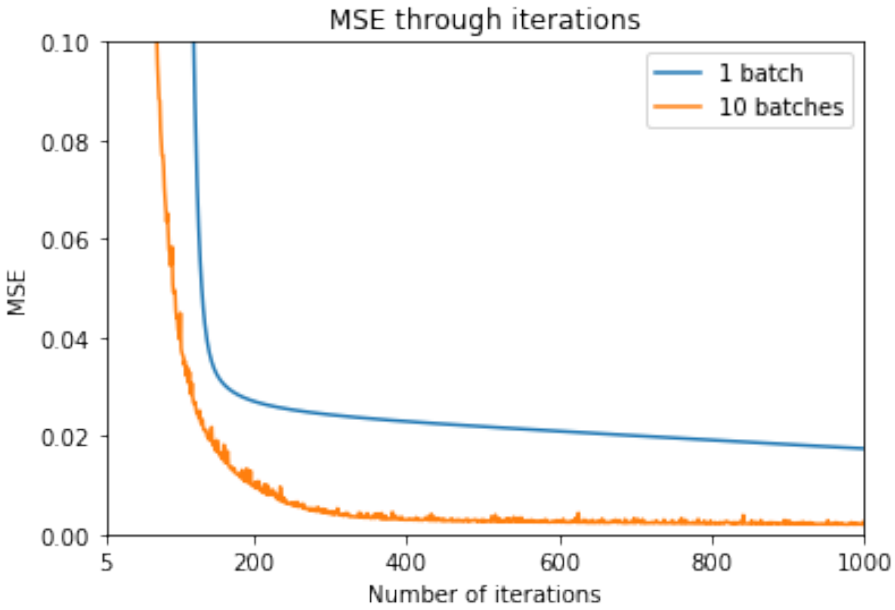
2.2.3 Uczenie na całym zbiorze vs. z wykorzystaniem mini-batchy



(a) Uczenie bez mini-batchy

(b) Uczenie z 10 mini-batchami po 10 obserwacji w każdym

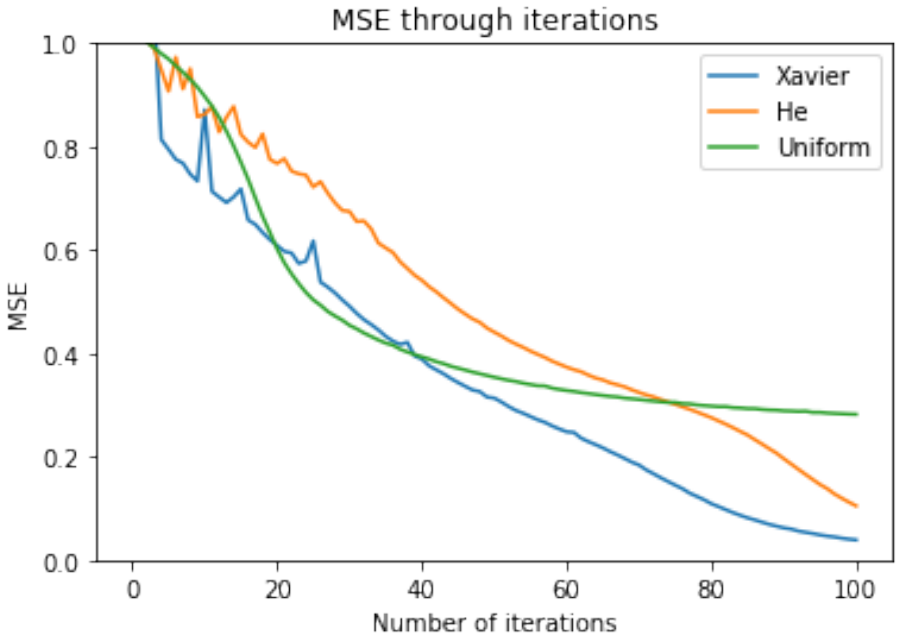
Rysunek 8: Porównanie dokładności uczenia bez i z mini-batchami



Rysunek 9: Porównanie szybkości zbieżności dla obu wariantów uczenia

2.2.4 Porównanie metod inicjalizowania wag

Poniższy test przeprowadziłam na zbiorze *square-simple*.



Rysunek 10: Porównanie zbieżności uczenia w 100 iteracjach dla różnych metod inicjalizowania wag

2.3 Wnioski

- Algorytm działa poprawnie - po odpowiednio dużej liczbie iteracji oraz dopasowaniu architektury sieci rozwiązanie zbiega (rys. 4-7).
- Uczenie z wykorzystaniem mini-batchy osiąga zdecydowanie lepsze wyniki (rys. 8,9). W dalszej części sprawozdania będę korzystać tylko z tego algorytmu.
- Sieć neuronowa najlepiej uczy się przy inicjalizacji wag metodą Xavier (rys.10). Przy inicjalizacji metodą He rozwiązanie również zbiega, tylko

wolniej. Inicjalizacja wag z rozkładu jednostajnego osiąga znacząco gorsze wyniki.

3 Implementacja momentu i normalizacji gradientu

3.1 Krótki opis problemu

Zadaniem tej części projektu była implementacja usprawnień uczenia gradientowego sieci neuronowej:

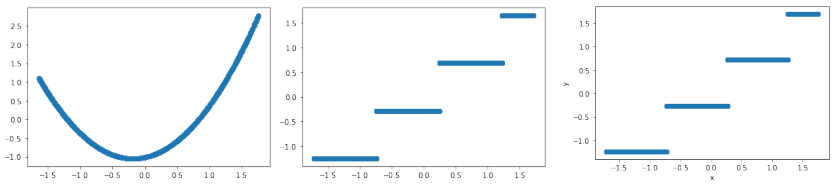
- metody momentów
- normalizacji gradientu RMSProp

3.2 Testy i wizualizacje

Przeprowadzone przeze mnie testy mają na celu porównanie szybkości zbieżności procesu uczenia dla obu wariantów.
Testy przeprowadziłam na trzech zbiorach danych:

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

3.2.1 Wizualizacja zbiorów danych



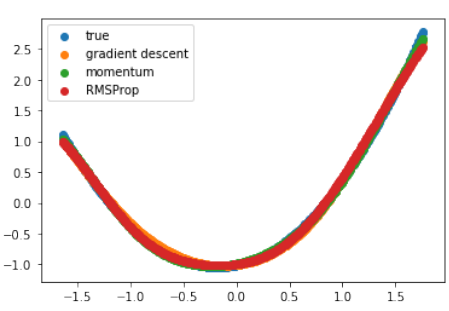
(a) square-large (b) steps-large (c) multimodal-large

Rysunek 11

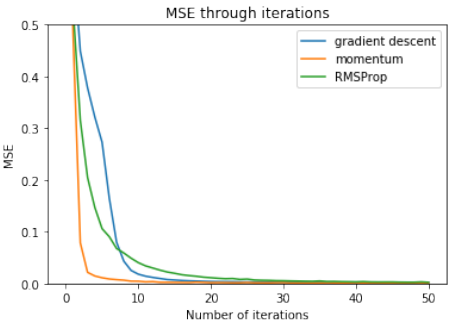
Wszystkie testy przeprowadziłam na sieciach neuronowych o jednej warstwie ukrytej z 20 neuronami. Dla metody *momentum* przyjęłam $\beta = 0.9$. Dla me-

tody *RMSProp* przyjęłam $\beta = 0.01$. Wszystkie wizualizacje wykonałam po 50 iteracjach algorytmu.

3.2.2 Porównanie

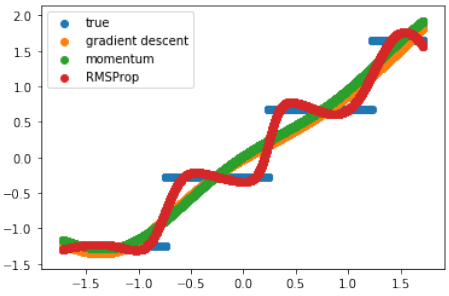


(a) Porównanie wyników

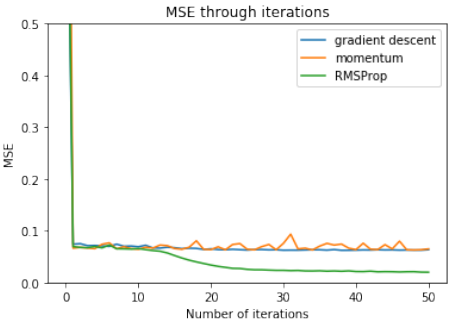


(b) Porównanie MSE

Rysunek 12: Porównanie metod dla zbioru *square-large*

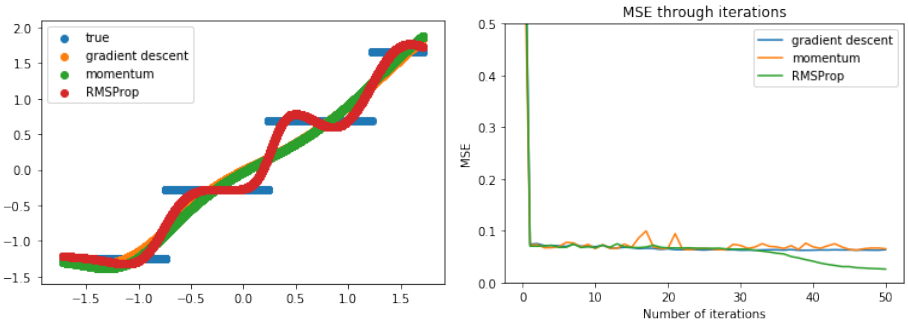


(a) Porównanie wyników



(b) Porównanie MSE

Rysunek 13: Porównanie metod dla zbioru *steps-large*



(a) Porównanie wyników

(b) Porównanie MSE

Rysunek 14: Porównanie metod dla zbioru *multimodal-large*

3.3 Wnioski

- Dla zbioru *squre-large* uzyskane wyniki są bardzo zbliżone (rys.12). Błąd dopasowania (MSE) najszybciej zbiega do 0 w przypadku metody *momentum*.
- Dla zbiorów *steps-large* i *multimodal-large* najlepsze wyniki osiąga metoda *RMSProp* (rys.13, 14).
- Ze względu na to, że wyniki w pierwszym teście są bardzo zbliżone, a w kolejnych odbiegają od siebie w sposób znaczący, możemy wysnuć wniosek, że najlepsze wyniki osiąga metoda *RMSProp*.

4 Rozwiązywanie zadania klasyfikacji

4.1 Krótki opis problemu

Na tym etapie rozwoju projektu przystosowywaliśmy nasze implementacje sieci neuronowych do rozwiązywania zadań klasyfikacji.

4.2 Testy i wizualizacje

Przeprowadzone testy obejmowały:

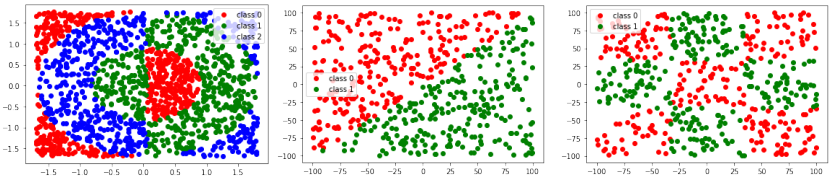
- sprawdzenie poprawności algorytmu (czy sieć neuronowa zbiega do poprawnego wyniku)
- porównanie wyników dla róznych funkcji aktywacji na ostatniej warstwie (softmax i funkcja liniowa)

Testy przeprowadziłam na trzech zbiorach danych:

- *rings3-regular*
- *easy*
- *xor3*

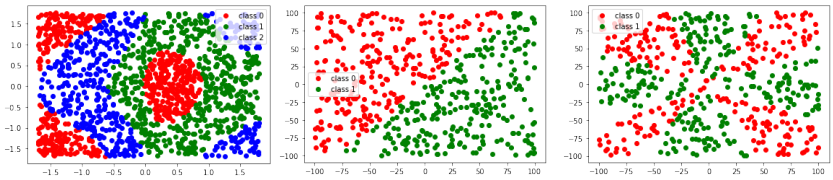
Wszystkie testy przeprowadziłam na sieciach neuronowych o dwóch warstwach ukrytych, po 30 neuronów każda. Wizualizacje wykonałam po 100 iteracjach algorytmu. Stosowałam metodę *RMSProp* z parametrem $\beta = 0.01$

4.2.1 Wizualizacja zbiorów danych



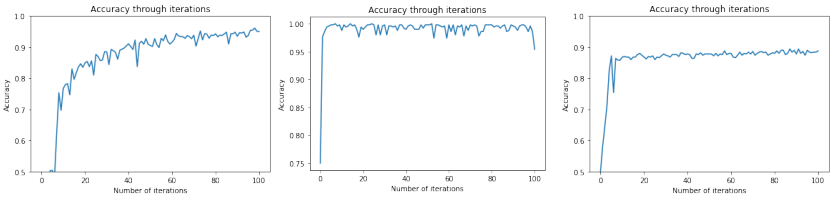
(a) ring3-regular (b) easy (c) xor3

4.2.2 Sprawdzenie poprawności algorytmu



(a) ring3-regular (b) easy (c) xor3

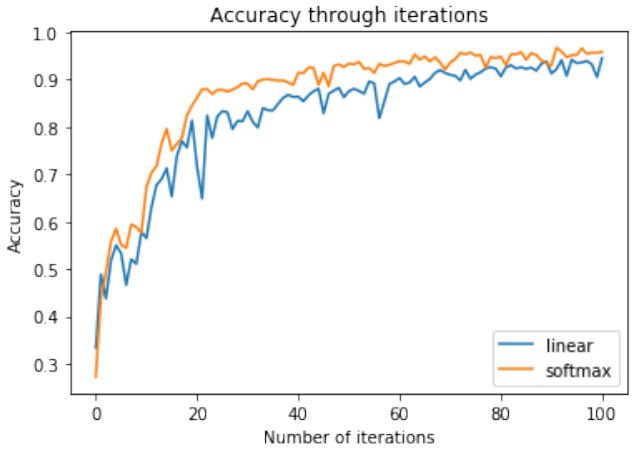
Rysunek 16: Przewidywane podziały na klasy po 100 iteracjach



(a) ring3-regular (b) easy (c) xor3

Rysunek 17: Dokładność przewidywania klasy (accuracy) w zależności od liczby iteracji

4.2.3 Porównanie funkcji aktywacji dla ostatniej warstwy



Rysunek 18: Porównanie szybkości zbieżności dla zbioru rings3-regular

4.3 Wnioski

- Po wprowadzeniu kilku poprawek algorytm działa poprawnie również dla zadań klasyfikacji. Dla każdego z trzech przeprowadzonych testów (rys. 17) sieć neuronowa osiąga accuracy z zakresu $(0.9, 1)$.
- Przyjęcie softmax jako funkcji aktywacji dla ostatniej warstwy poprawia szybkość zbieżności algorytmu (rys. 18).

5 Testowanie różnych funkcji aktywacji

5.1 Krótki opis problemu

Szybkość zbieżności sieci neuronowej zależy również od wyboru odpowiedniej funkcji aktywacji. W tej części sprawozdania zajmę się porównianiem następujący funkcji aktywacji:

- liniowa
- reLU
- sigmoid
- tanh

5.2 Testy i wizualizacje

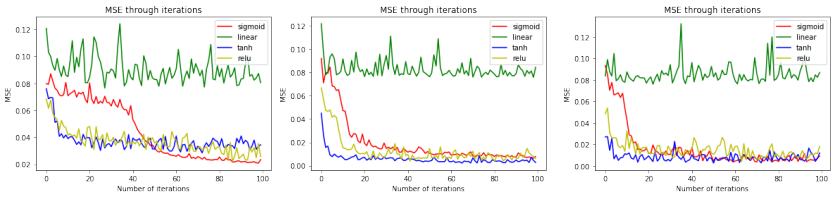
Testy przeprowadziłam na trzech zbiorach danych:

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

Dla każdego zbioru danych wykonałam testy dla trzech różnych architektur sieci:

- jednej warstwy ukrytej o 30 neuronach
- dwóch warstw ukrytych o 30 i 20 neuronach
- trzech warstw ukrytych o 30, 20 i 10 neuronach

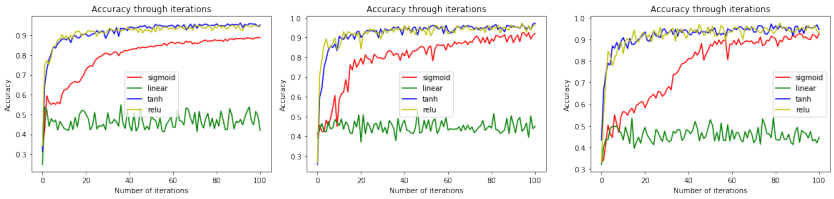
5.2.1 Porównanie funkcji aktywacji w zadaniu regresji



(a) jedna warstwa ukryta (b) dwie warstwy ukryte (c) trzy warstwy ukryte

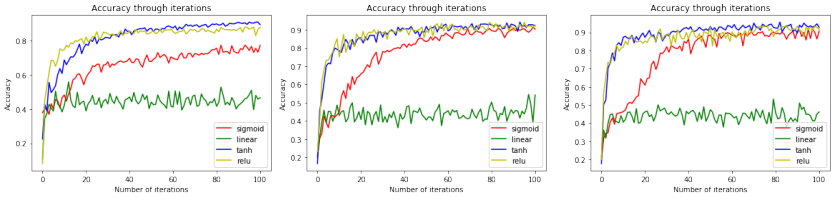
Rysunek 19: Porównanie funkcji aktywacji dla zbioru *steps-large*

5.2.2 Porównanie funkcji aktywacji w zadaniu klasyfikacji



(a) jedna warstwa ukryta (b) dwie warstwy ukryte (c) trzy warstwy ukryte

Rysunek 20: Porównanie funkcji aktywacji dla zbioru *rings3-regular*



(a) jedna warstwa ukryta (b) dwie warstwy ukryte (c) trzy warstwy ukryte

Rysunek 21: Porównanie funkcji aktywacji dla zbioru *rings5-regular*

5.3 Wnioski

- Funkcja liniowa osiąga najgorsze rezultaty, zarówno w zadaniu regresji, jak i klasyfikacji (rys. 19-21).
- Najlepsze rezultaty zapewnia zastosowanie jako funkcji aktywacji funkcji *tanh* lub $reLU$, w zadaniu regresji nieznacznie lepsze wyniki osiąga *tanh*.

6 Zjawisko przeuczenia i regularyzacja

6.1 Krótki opis problemu

Ostatnią częścią projektu było zaimplementowanie mechanizmu regularyzacji wag w sieci neuronowej. Regularyzacja przeciwdziała zjawisku przeuczenia sieci.

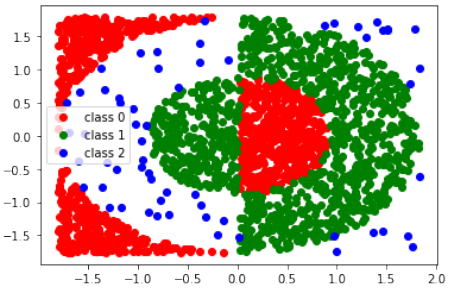
6.2 Testy i wizualizacje

Testy przeprowadziłam na trzech zbiorach danych:

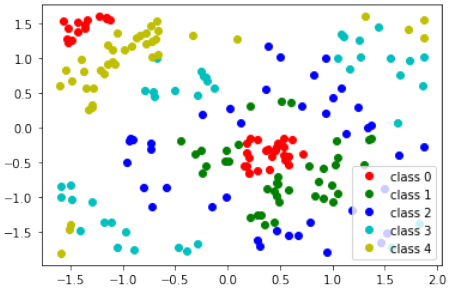
- *multimodal-sparse*
- *rings3-balance*
- *rings5-sparse*
- *xor3-balance*

Wszystkie testy przeprowadziłam na sieciach neuronowych o dwóch warstwach ukrytych po 30 neuronów każda.

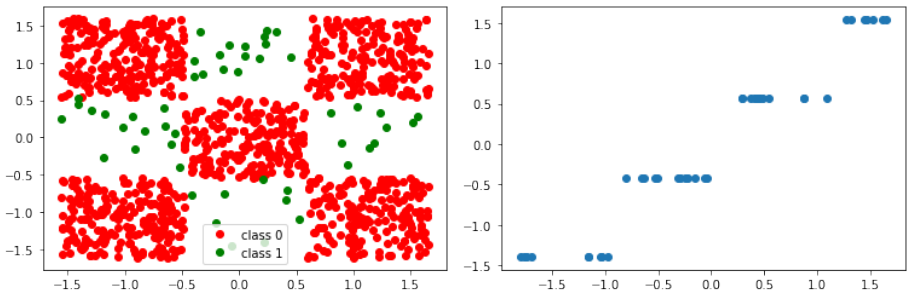
6.2.1 Wizualizacja zbiorów danych



(a) ring3-balance

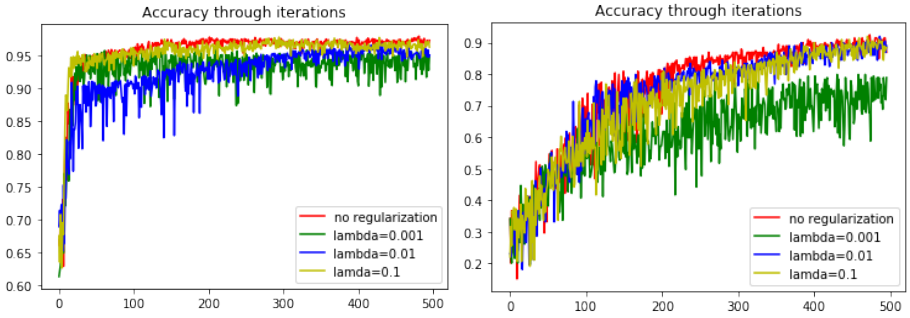


(b) rings5-sparse

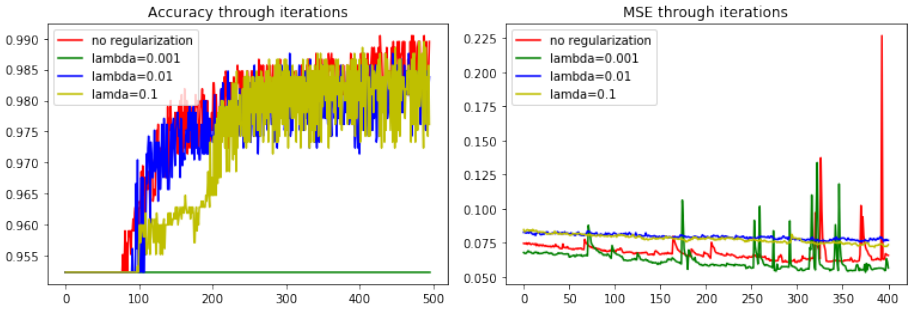


(a) xor3-balance (b) multimodal-sparse

6.2.2 Regularyzacja



(a) ring3-balance (b) rings5-sparse



(a) xor3-balance (b) multimodal-sparse

Rysunek 25: Wykresy miar (accuracy/mse) dla różnych wartości regularyzacji

6.3 Wnioski

W tym miejscu nie jestem pewna poprawności mojej implementacji i wyników. Analizując otrzymane wykresy można wysnuć wnioski:

1. Dla zadań klasyfikacji najlepsze wyniki osiąga model bez regularyzacji. Może to oznaczać przeuczenie tego modelu.

2. Dla zadania regresji najlepsze wyniki osiąga model z najmniejszą stałą regularyzacji - 0.001