Wydział Matematyki i Nauk Informacyjnych Politechniki Warszawskiej

Dokumentacja

Multilayer Perceptron

Karol Ulanowski
Jakub Waszkiewicz

24 marca 2020

Spis treści

1 Opis programu

Program jest implementacją wielowarstwowego perceptronu uczonego za pomocą algorytmu propagacji wstecznej błędu. Pozwala na stworzenie sieci o wybranej strukturze, wytrenowanie jej na postawie zadanych parametrów oraz zbioru treningowego. Pozwala też na sprawdzenie skuteczności sieci na podstawie zbioru testowego.

Stworzona aplikacja pozwala rozwiązywać zarówno problemy regresji, jak i klasyfikacji. Została napisana w języku Python przy wykorzystaniu podstawowych bibliotek takich jak Pandas oraz NumPy.

2 Konfiguracja

W celu uruchomienia programu należy uruchomić podać wszystkie następujące parametry:

- --layers (short: -s): liczba neuronów w warstwach oddzielana przecinkami. Pierwsza liczba oznacza rozmiar warstwy wejściowej ostatni rozmiar warstwy wyjściowej. Przykład "2,10,5,2"
- --activation_function (short: -f): funkcja aktywacji. Do wyboru:
  - 0 - Leaky ReLU z parametrem 0.01
  - 1 - Sigmoid
  - 2 - Tanh
  - 3 - Identity
- --bias (short: -b): obecność biasu.
  - 0 - brak
  - 1 - jest
- --batch-size (short: -s): rozmiar batcha 1,2,3...
- --number_of_iterations (short: -n): liczba iteracji 1,2,3...
- --learning_rate (short: -r)
- --momentum (short: -m)
- --problem (short: -p): rodzaj problemu
  - 0 - klasyfikacja
  - 1 - regresja
  - 2 - kaggle
- --input (short: -i): ścieżka do pliku treningowego
- --test (short: -t): ścieżka do pliku testowego
- --seed (short: -d): ziarno losowości
- --viusalizer (short: -v): obecność wizualizacji
  - 0 - brak wizualizacji
  - 1 - wizualizacja
- --collect_results_for_iterations (short: -a): co ile iteracji program ma wypisywać aktualny błąd
- --split (short: -x): podział treningowego pliku kaggle na zbiór testowy i treningowy. Wartość 0.7 oznacza że 70% próbek zostanie użytych do treningu.

3 Testowane parametry

Dla problemu regresji poprawności rozwiązania oceniano na podstawie średniego błędu dla próbki uzyskanego na danych testowych. Z kolei dla klasyfikacji wyznacznikiem poprawności rozwiązania była skuteczność wytrenowanej sieci, tzn. stosunek poprawnie sklasyfikowanych obserwacji do ich łącznej liczby, wyrażona w procentach. Podczas testów użyto stałego ziarna losowości równego 1000.

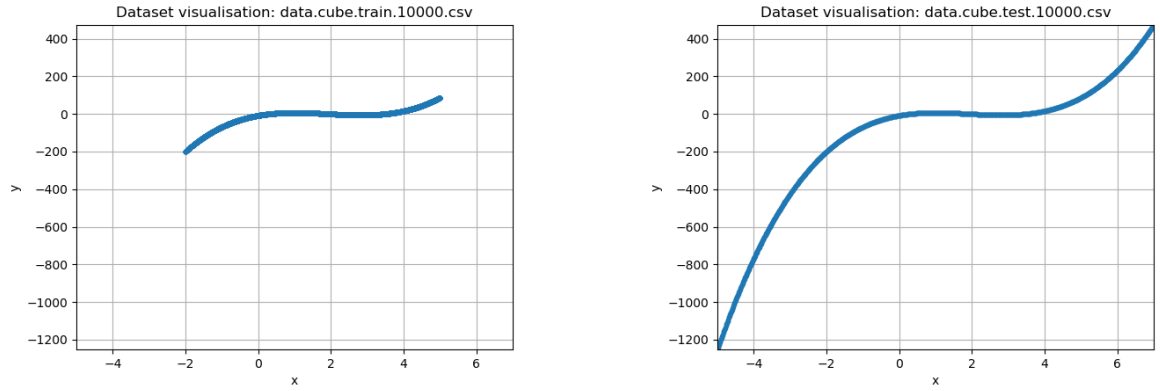
Przetestowano napisaną sieć pod względem następujących parametrów:

- rozmiaru warstw,
- funkcji kosztu,
- funkcji aktywacji,
- współczynnika nauki,
- współczynnika bezwładności,
- rozmiaru batcha,
- obecność biasu.

4 Regesja

Do inicjalizacji wag w sieci zostało wykorzystane stałe ziarno 1000, aby umożliwić porównanie wyników. We wszystkich testach regresji na ostatniej warstwie użyto jako funkcję aktywacji Identity ze względu na jej zbiór wartości $[-\infty, \infty]$. Trening przeprowadzono na danych przedstawionych na rysunku 1a. Dane testowe zostały przedstawione na rysunku 1b. Są to zbiory:

- *data.cube.train.10000.csv*,
- *data.cube.test.10000.csv*.



(a) Dane treningowe

(b) Dane testowe

Rysunek 1: Wizualizacja danych dla regresji

4.1 Rozmiar warstw

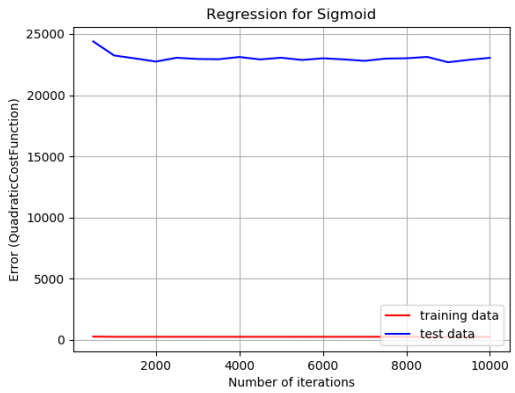
Przetestowano wpływ rozmiaru różnych sieci na otrzymaną dokładność. Sprawdzono rozmiary:

- brak warstw ukrytych,
- jedna warstwa ukryta z 7 neuronami [7],

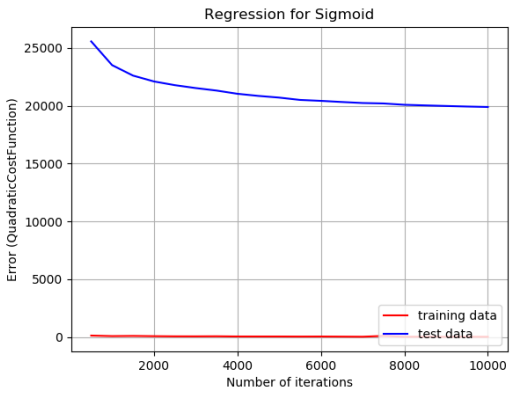
- dwie warstwy ukryte po 7 neurony [7, 7],
- trzy warstwy ukryte po 7 neurony [7, 7, 7].

Pozostała konfiguracja była następująca:

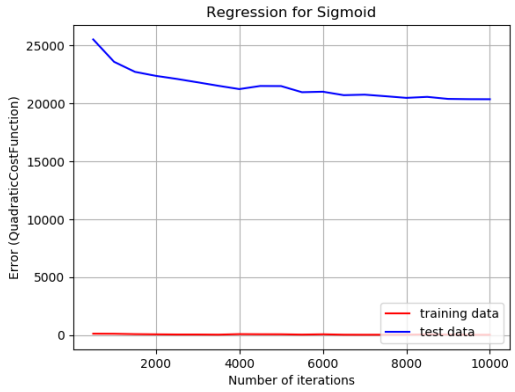
- liczba iteracji - 10000,
- funkcja aktywacji - Sigmoid,
- funkcja kosztu - Quadratic,
- obecność biasu - tak,
- rozmiar batcha - 10,
- współczynnik nauki - 0.005,
- współczynnik bezwładności - 0.



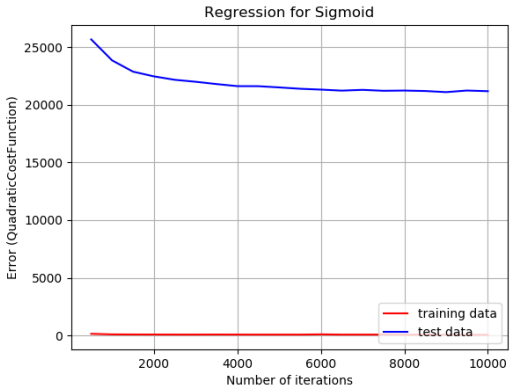
(a) brak warstw ukrytych



(b) 1 warstwa ukryta



(c) 2 warstwy ukryte



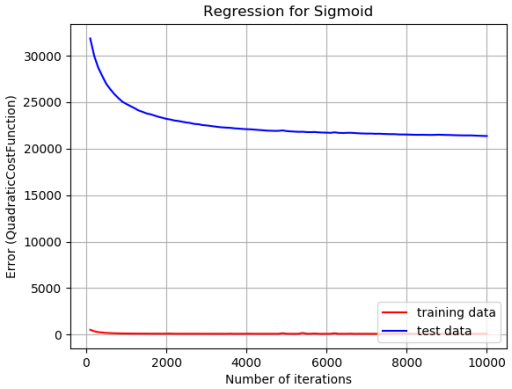
(d) 3 warstwy ukryte

Rysunek 2: Zależność kosztu od liczby warstw ukrytych

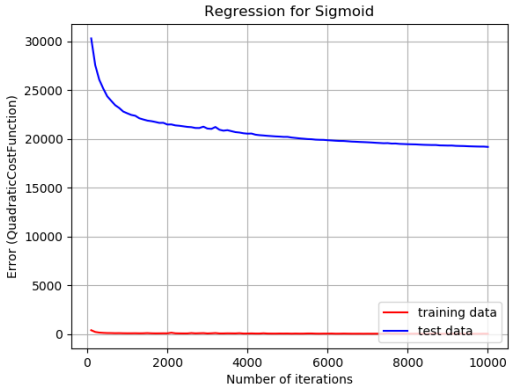
Najgorsze wyniki uzyskała sieć bez warstw ukrytych, a najlepsze - z jedną warstwą ukrytą. Dodanie kolejnych warstw sieci nie obniżyło kosztu, ale też nie pogorszyło go w znaczący sposób.

Dla sieci o 1 warstwie ukrytej można też sprawdzić, jaki ma wpływ rozmiar tej warstwy. Zostały zbadane warstwy o rozmiarze:

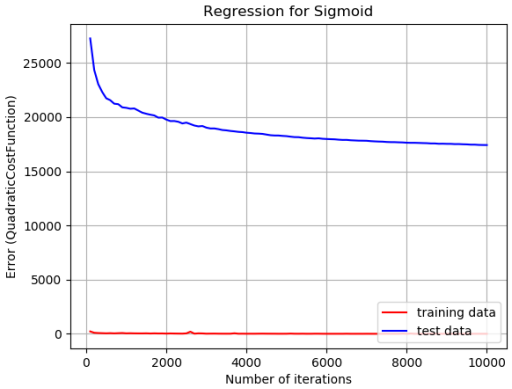
- 5,
- 10,
- 30,
- 100.



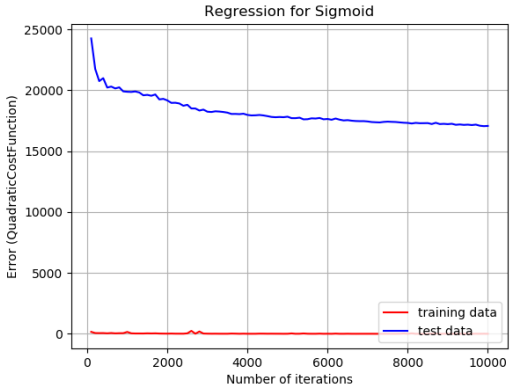
(a) warstwa o 5 neuronach



(b) warstwa o 10 neuronach



(c) warstwa o 30 neuronach



(d) warstwa o 100 neuronach

Rysunek 3: Zależność kosztu od rozmiaru warstwy ukrytej

Zwiększenie liczby neuronów na warstwie pozwoliło w znaczący sposób zmniejszyć wartość błędu.

4.2 Funkcja kosztu

Sprawdzano dwie funkcje dla problemu regresji: Quadratic Cost i Cross Entropy, jednak funkcja Cross Entropy nie działała w naszym przypadku dla tego problemu. Działanie funkcji Quadratic Cost przedstawiono więc na pozostałych wykresach w rozdz. 4 gdzie jest ona testowana w zależności od różnych parametrów.

4.3 Funkcja aktywacji

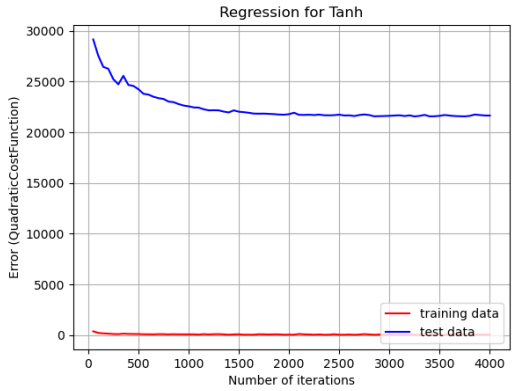
Zostały przetestowane funkcje aktywacji:

- Leaky ReLU z parametrem 0.01,
- Sigmoid,

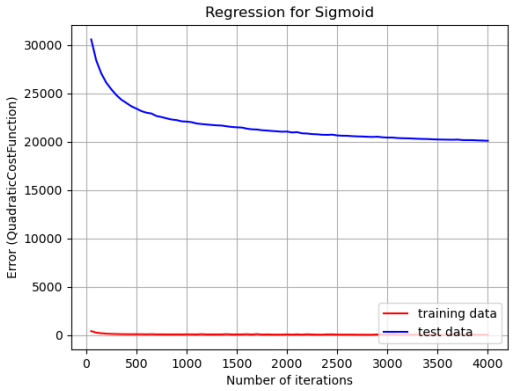
- Tangens hiperboliczny,
- Identity.

Pozostała konfiguracja była następująca:

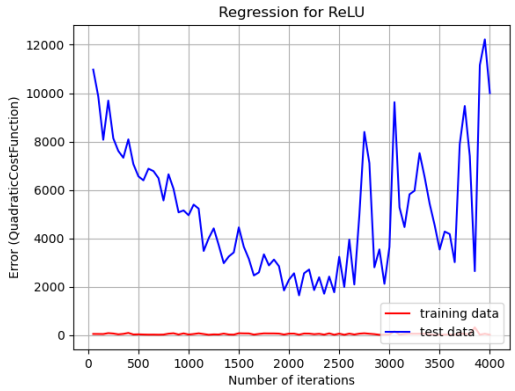
- rozmiar warstw ukrytych - [7],
- liczba iteracji - 4000,
- funkcja kosztu - Quadratic,
- obecność biasu - tak,
- rozmiar batcha - 100,
- współczynnik nauki - 0.5,
- współczynnik bezwładności - 0.1.



(a) Tanh



(b) Sigmoid



(c) Leaky ReLU z parametrem 0.01

(d) Identity

Rysunek 4: Zależność skuteczności od funkcji aktywacji

Funkcja Leaky ReLU osiągnęła zdecydowanie najlepszą dokładność po około 2000 iteracjach, później powstało zjawisko przeuczenia. Zdecydowaną przewagą tej funkcji jest jej przeciwdziedzina z przedziału $[-\infty, \infty]$. Z kolei w przeciwieństwie do funkcji Identity wprowadza ona dodatkowo nieliniowość. Widać jednak, że w przypadku funkcji ReLU zastosowano zbyt duży współczynnik nauki i bezwładności, co charakteryzuje się gwałtownymi skokami.

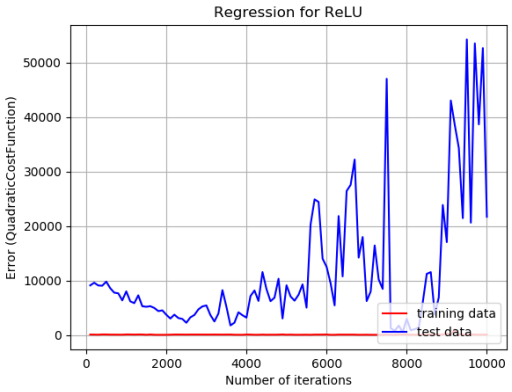
4.4 Współczynnik nauki

Sprawdzono 3 różne współczynniki nauki:

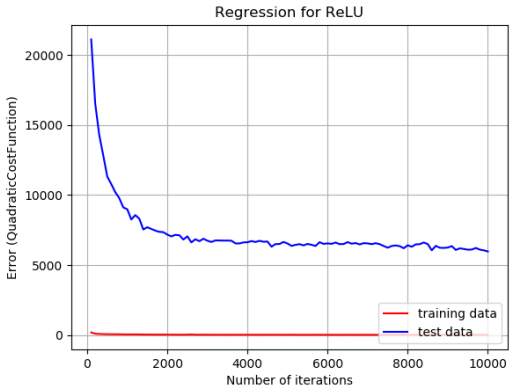
- 0.01,
- 0.001,
- 0.0001.

Pozostała konfiguracja była następująca:

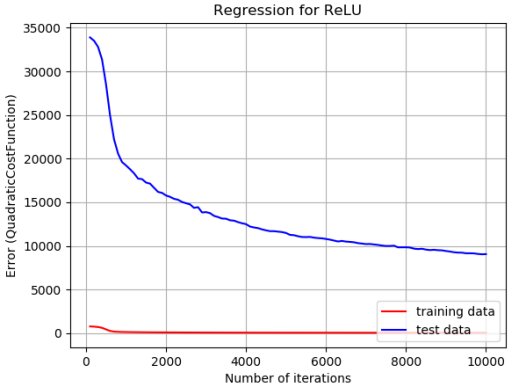
- rozmiar warstw ukrytych - [10],
- liczba iteracji - 10000,
- funkcja aktywacji - Leaky ReLU z parametrem 0.01,
- funkcja kosztu - Quadratic,
- obecność biasu - tak,
- rozmiar batcha - 10,
- współczynnik bezwładności - 0.



(a) Współczynnik nauki 0.01



(b) Współczynnik nauki 0.001



(c) Współczynnik nauki 0.0001

Rysunek 5: Zależność kosztu od współczynnika nauki

Jak widać z wykresów, dobór współczynnika nauki nie jest oczywisty. Przy jego zbyt wysokiej wartości możemy mieć sytuację widoczną na rysunku 5a. Zbyt szybkie uczenie może spowodować "przeskoczenie" szukanego minimum funkcji kosztu, a zatem oddalenie się od szukanego rozwiązania. Nadanie współczynnikowi zbyt małej wartości może spowolnić zbieżność, co oznacza, że przy danej liczbie iteracji sieć może nie nauczyć się wystarczająco. Dla testowanych przypadków najlepsze wyniki uzyskała sieć z współczynnikiem 0.001.

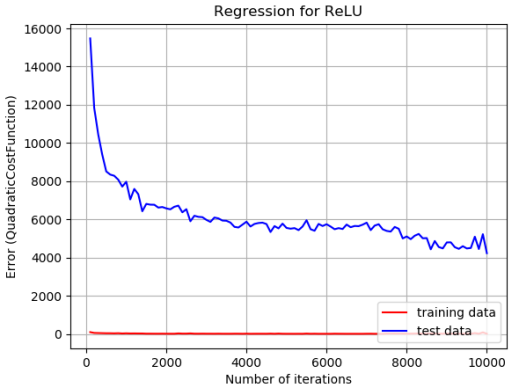
4.5 Współczynnik bezwładności

Sprawdzono 3 różne współczynniki bezwładności:

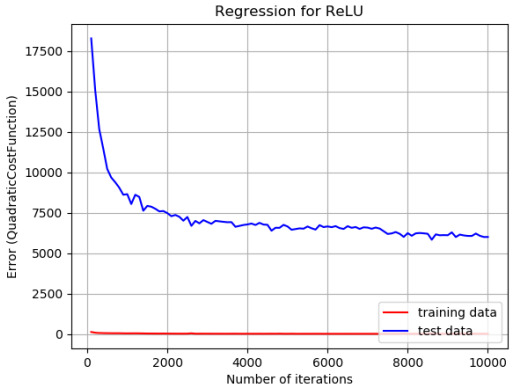
- 0.5,
- 0.1,
- 0.0001.

Pozostała konfiguracja była następująca:

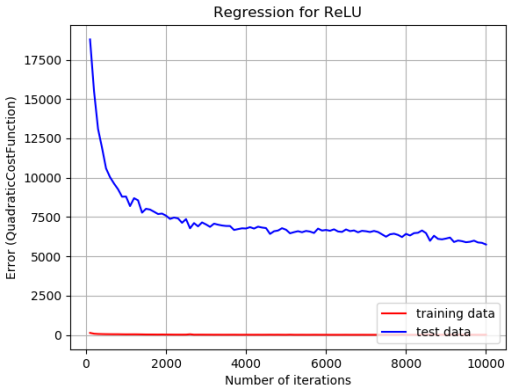
- rozmiar warstw ukrytych - [15],
- liczba iteracji - 10000,
- funkcja aktywacji - Leaky ReLU z parametrem 0.01,
- funkcja kosztu - Quadratic,
- obecność biasu - tak,
- rozmiar batcha - 10,
- współczynnik nauki - 0.001.



(a) Współczynnik bezwładności 0.5



(b) Współczynnik bezwładności 0.1



(c) Współczynnik bezwładności 0.001

Rysunek 6: Zależność kosztu od współczynnika bezwładności

Współczynnik bezwładności jest elementem, który ma wpływ na tempo zbieżności sieci. W jego przypadku mogą się pojawić podobne problemy, jak przy doborze współczynnika nauki. Jednak jest on wyliczany na podstawie zmiany wag z poprzedniej iteracji, zatem poziom jego wpływu jest mniejszy, niż współczynnika nauki.

4.6 Rozmiar batcha

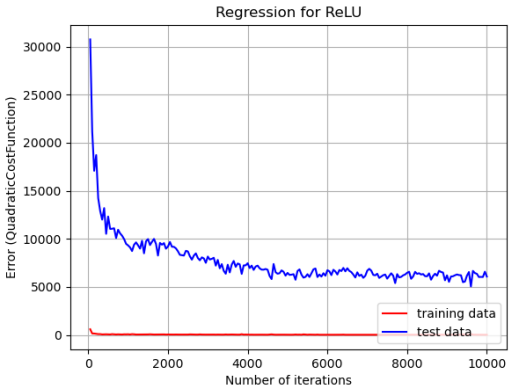
Sprawdzono następujące rozmiary:

- 1,
- 10,
- 100,
- 1000.

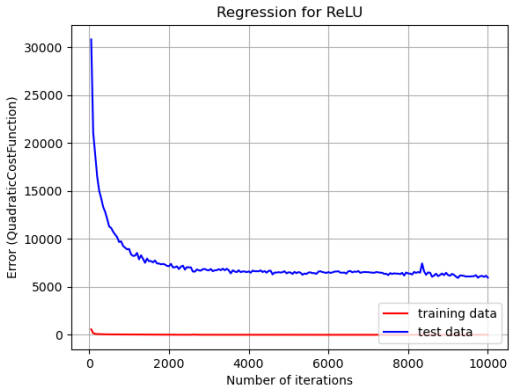
Pozostała konfiguracja była następująca:

- rozmiar warstw ukrytych - [7],
- liczba iteracji - 1000,
- funkcja aktywacji - Leaky ReLu z parametrem 0.01,
- funkcja kosztu - Quadratic,

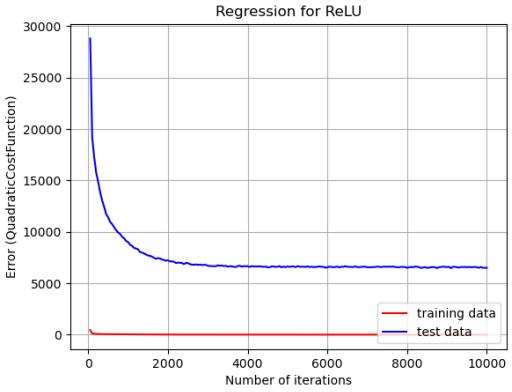
- obecność biasu - tak,
- współczynnik nauki - 0.001,
- współczynnik bezwładności - 0.1.



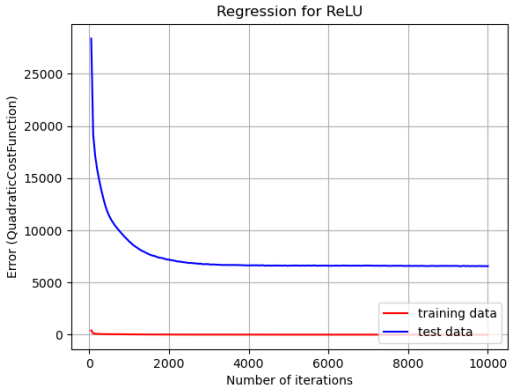
(a) Rozmiar batcha 1



(b) Rozmiar batcha 10



(c) Rozmiar batcha 100



(d) Rozmiar batcha 1000

Rysunek 7: Zależność skuteczności od rozmiaru batcha

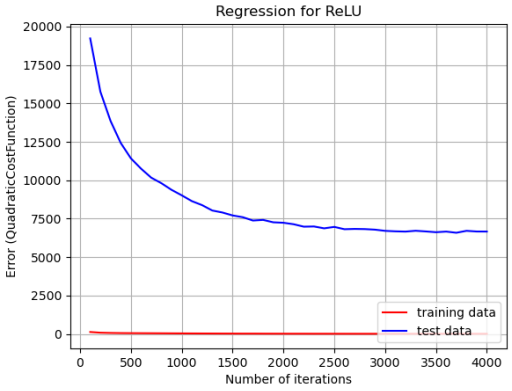
Zwiększenie rozmiaru batcha poprawia kierunek zbieżności sieci. Im większy batch tym poprawa wag i biasu w modelu jest dokładniejsza, jako że gradient jest uśredniany z większej próbki. Warto też zaznaczyć, że zwiększając tylko rozmiar batcha i zostawiając taka sama liczbę iteracji zwiększamy też czas trwania uczenia.

4.7 Obecność biasu

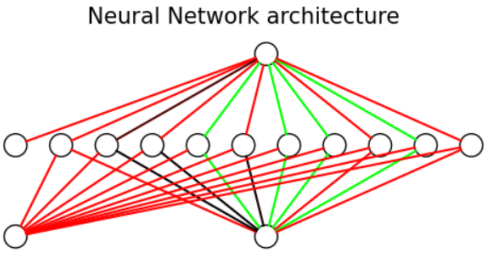
Przeprowadzono dwa testy w których sprawdzono wpływ biasu na proces uczenia się sieci. Przetestowano tą samą sieć z użyciem biasu i bez. Konfiguracja sieci:

- rozmiar warstw ukrytych - [7],
- liczba iteracji - 1000,
- funkcja aktywacji - Leaky ReLU z parametrem 0.01,
- funkcja kosztu - quadratic cost,
- rozmiar batcha - 100,

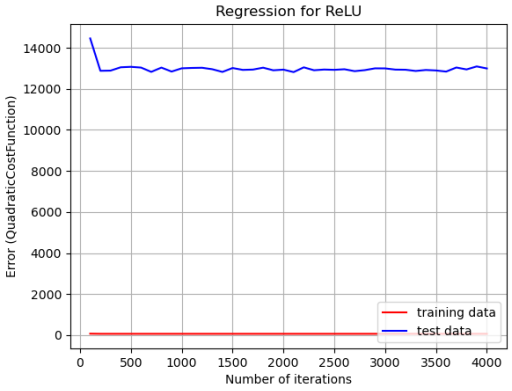
- współczynnik nauki - 0.01,
- współczynnik nauki - 0.05.



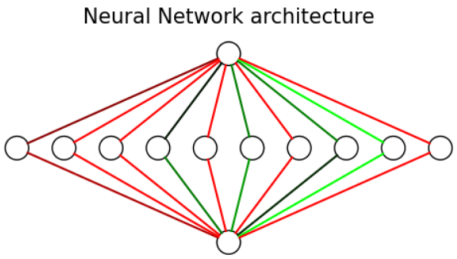
(a) Sieć z biasem



(b) Struktura sieci z biasem



(c) Sieć bez biasu



(d) Struktura sieci bez biasu

Rysunek 8: Zależność skuteczności od obecności biasu

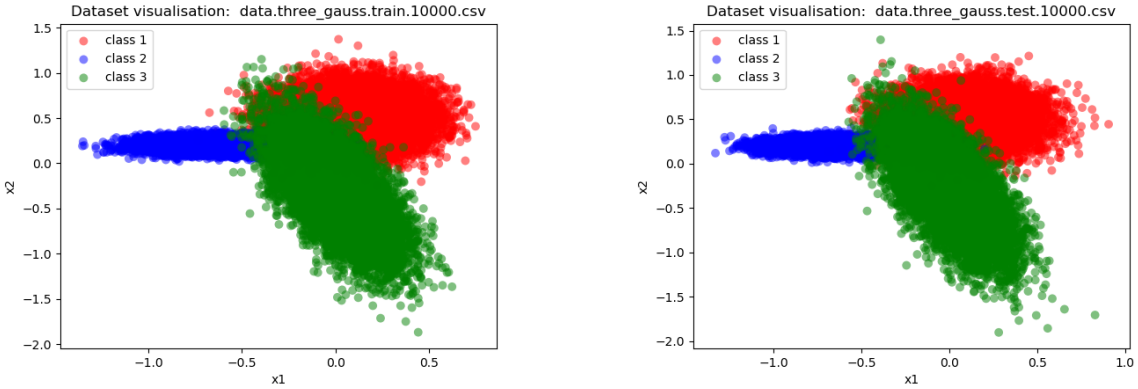
Na wykresach przedstawiono również struktury sieci. Jaśniejszy kolor zielony oznacza większą dodatnią wagę, jaśniejszy czerwony dużą ujemną wagę. Czarne krawędzie oznaczają wartości bliskie zera. Sieć z biasem dała dużo lepsze wyniki zbieżności. Wartości, które przewidujemy są w głównej mierze ujemne, dlatego też bias, który widać jest jasno czerwono co oznacza dużą ujemną wagę krawędzi.

5 Klasyfikacja

We wszystkich testach klasyfikacji na ostatniej warstwie użyto jako funkcję aktywacji Sigmoid ze względu na jej zbiór wartości $[0, 1]$ Trenowanie i testy dla klasyfikacji przeprowadzono na zbiorach:

- *data.three_gauss.train.10000.csv*,
- *data.three_gauss.test.10000.csv*.

Dane te przedstawiono na wykresach poniżej na rysunkach 9a oraz 9b.



(a) Dane treningowe

(b) Dane testowe

Rysunek 9: Wizualizacja danych dla klasyfikacji

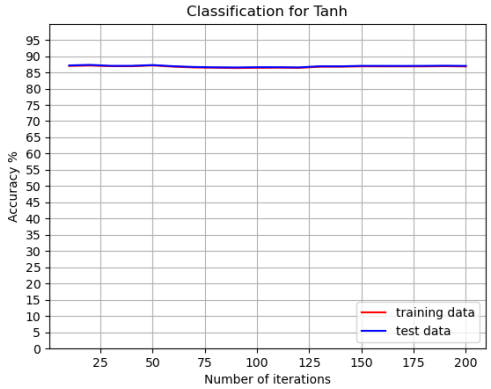
5.1 Rozmiar warstw

Przetestowano wpływ rozmiaru różnych sieci na otrzymaną dokładność. Sprawdzono rozmiary:

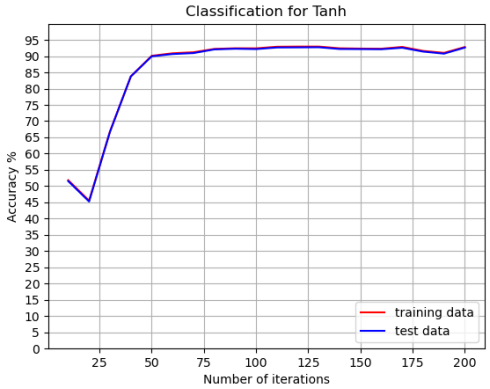
- brak warstw ukrytych,
- jedna warstwa ukryta z 3 neuronami [3],
- dwie warstwy ukryte po 3 neurony [3, 3],
- trzy warstwy ukryte po 3 neurony [3, 3, 3].

Pozostała konfiguracja była następująca:

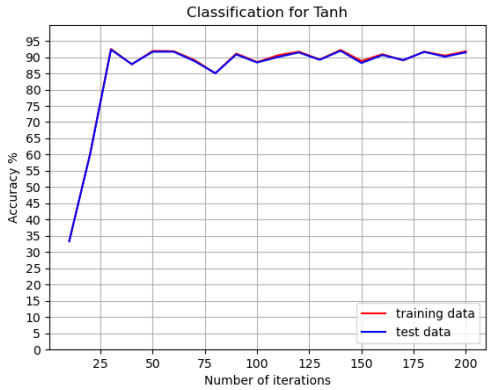
- liczba iteracji - 200,
- funkcja aktywacji - Tangens hiperboliczny,
- funkcja kosztu - Cross Entropy,
- obecność biasu - tak,
- rozmiar batcha - 10,
- współczynnik nauki - 0.5,
- współczynnik bezwładności - 0.1.



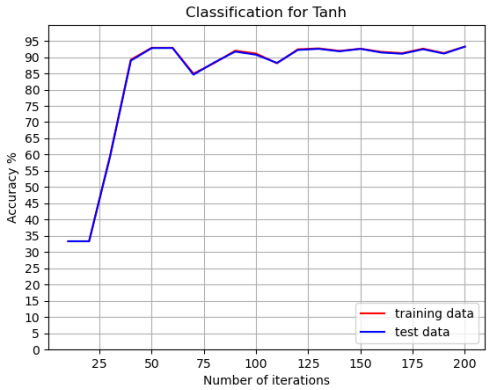
(a) brak warstw ukrytych



(b) 1 warstwa ukryta



(c) 2 warstwy ukryte



(d) 3 warstwy ukryte

Rysunek 10: Zależność skuteczności od liczby warstw ukrytych

Najsłabszy wynik dała sieć bez warstw ukrytych jednak najszybciej była zbieżna. Pomijając ten przykład, że dodanie większej liczby warstw do modelu powoduje szybszą zbieżność sieci, ale przez to, że sieć jest bardzo mała, widać skoki dokładności w późniejszych iteracjach.

5.2 Funkcja kosztu

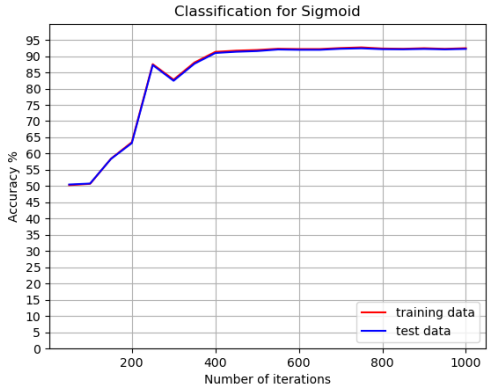
Sprawdzono dwie funkcje kosztu:

- Quadratic,
- Cross Entropy.

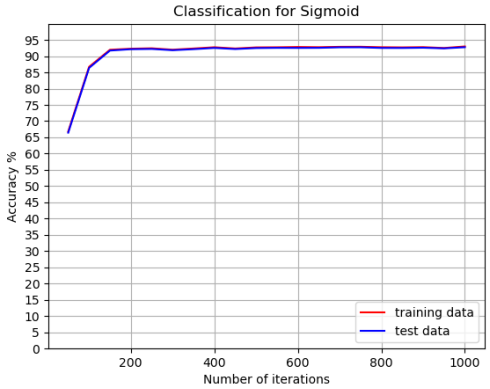
Pozostała konfiguracja była następująca:

- rozmiar warstw ukrytych - [7] i [7, 7],
- liczba iteracji - 200,
- funkcja aktywacji - Sigmoid,
- obecność biasu - tak,
- rozmiar batcha - 10,
- współczynnik nauki - 0.5,

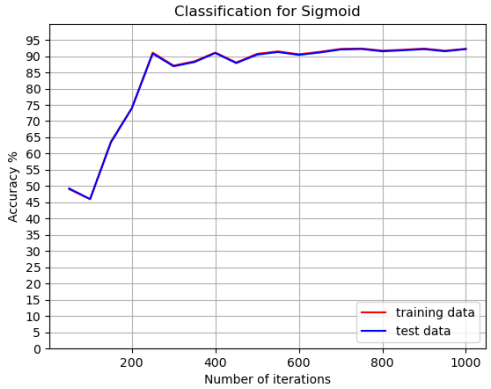
- współczynnik bezwładności - 0.1.



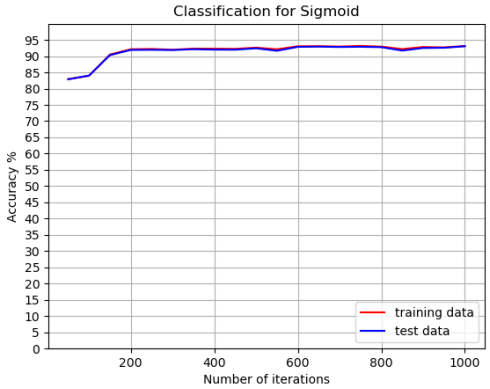
(a) Quadratic cost function - sieć (7)



(b) Cross entropy function - sieć (7)



(c) Quadratic cost function - sieć (7,7)



(d) Cross entropy function - sieć (7,7)

Rysunek 11: Zależność skuteczności od funkcji kosztu

Zastosowanie funkcji Cross Entropy daje dużo lepsze wyniki zbieżności modelu, jednak funkcja ta jest trudniejsza do obliczenia. Ponadto w przypadku wyliczania jej pochodnej wykonuje się dzielenie, co może w skrajnych przypadkach przy dzieleniu przez 0 spowodować wyjątek w programie.

5.3 Funkcja aktywacji

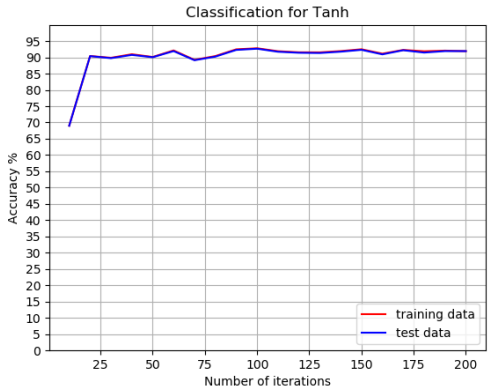
Zostały przetestowane funkcje aktywacji:

- Leaky ReLU z parametrem 0.01,
- Sigmoid,
- Tangens hiperboliczny,
- Identity.

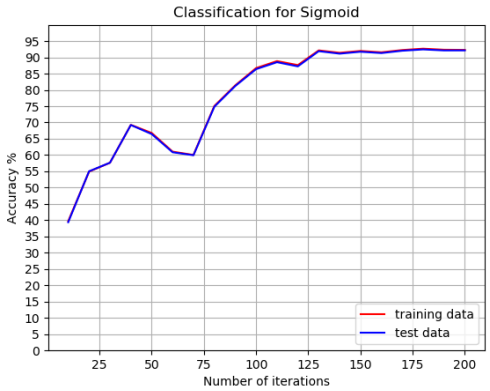
Pozostała konfiguracja była następująca:

- rozmiar warstw ukrytych - [7],
- liczba iteracji - 200,
- funkcja kosztu - Cross Entropy,

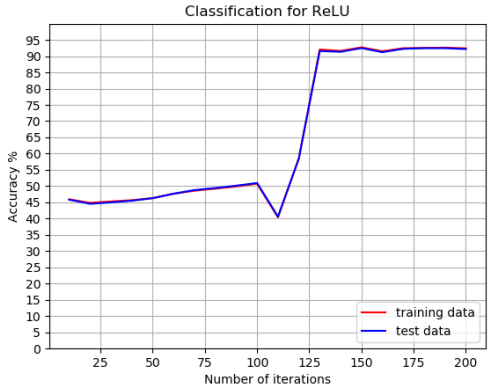
- obecność biasu - tak,
- rozmiar batcha - 10,
- współczynnik nauki - 0.5,
- współczynnik bezwładności - 0.1.



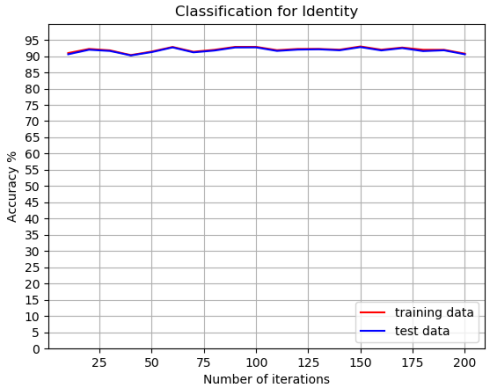
(a) Tanh



(b) Sigmoid



(c) Leaky ReLU z parametrem 0.01



(d) Identity

Rysunek 12: Zależność skuteczności od funkcji aktywacji

Jak widać na rysunku 12, dla testowanych sieci już po około 150 iteracjach skuteczność wynosi ponad 90% dla każdej funkcji. Dla mniejszej liczby iteracji najlepsza okazała się funkcja Identity oraz tangens hiperboliczny. Funkcja Leaky ReLU początkowo osiąga dość słabą skuteczność, lecz po 100 iteracjach zauważalny jest gwałtowny wzrost.

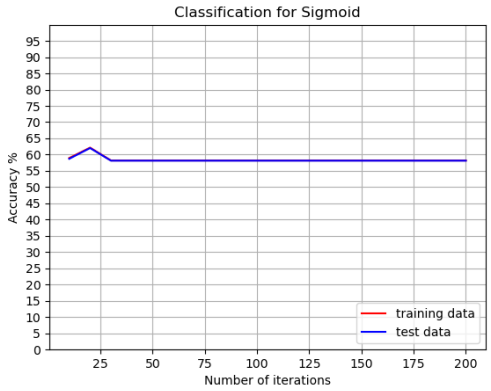
5.4 Współczynnik nauki

Sprawdzono 4 różne współczynniki nauki:

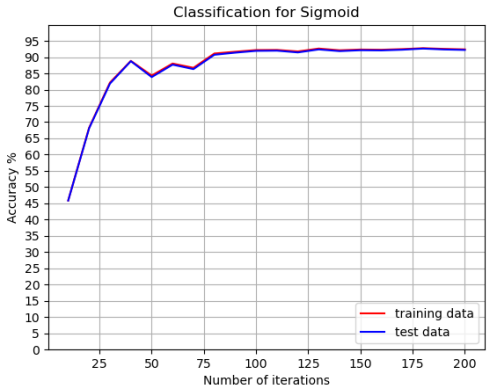
- 10,
- 1,
- 0.1,
- 0.001.

Pozostała konfiguracja była następująca:

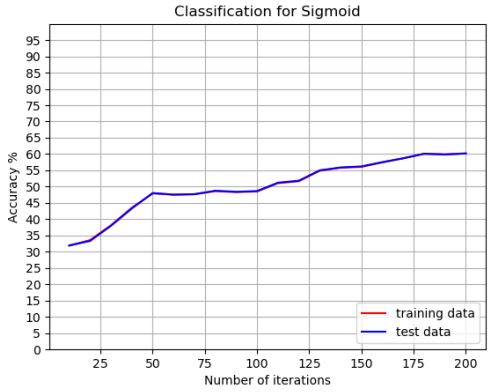
- rozmiar warstw ukrytych - [7],
- liczba iteracji - 200,
- funkcja aktywacji - Sigmoid,
- funkcja kosztu - Cross Entropy,
- obecność biasu - tak,
- rozmiar batcha - 10,
- współczynnik bezwładności - 0.



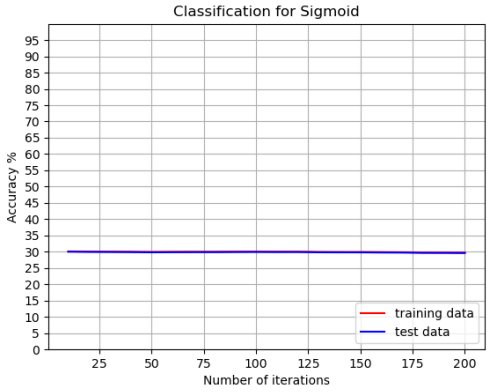
(a) Współczynnik nauki 10



(b) Współczynnik nauki 1



(c) Współczynnik nauki 0.1



(d) Współczynnik nauki 0.001

Rysunek 13: Zależność skuteczności od współczynnika nauki

Dobranie właściwej wartości współczynnika nauki jest jedną z kluczowych rzeczy podczas konfiguracji sieci. Zbyt duży współczynnik powoduje, że sieć nie potrafi osiągnąć minimum. Z kolei dla zbyt małych wartości sieć uczy się zbyt wolno i również nie widać postępu w nauce.

5.5 Współczynnik bezwładności

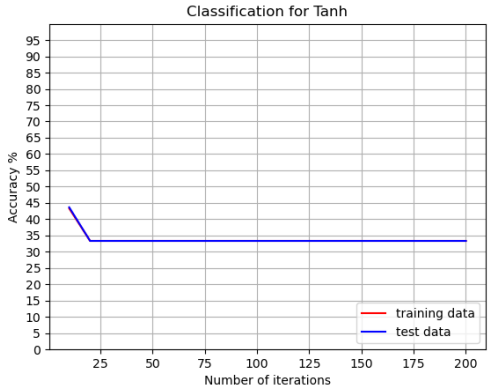
Sprawdzono 4 różne współczynniki bezwładności:

- 2,

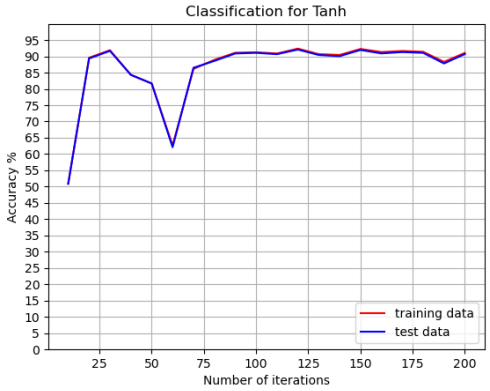
- 1,
- 0.1,
- 0.001.

Pozostała konfiguracja była następująca:

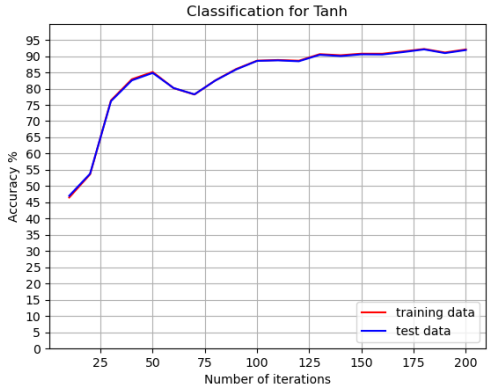
- rozmiar warstw ukrytych - [7],
- liczba iteracji - 200,
- funkcja aktywacji - Tangens hiperboliczny,
- funkcja kosztu - Cross Entropy,
- obecność biasu - tak,
- rozmiar batcha - 10,
- współczynnik nauki - 0.1.



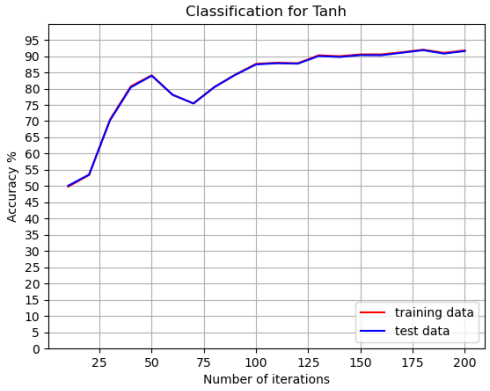
(a) Współczynnik bezwładności 2



(b) Współczynnik bezwładności 1



(c) Współczynnik bezwładności 0.1



(d) Współczynnik bezwładności 0.001

Rysunek 14: Zależność skuteczności od współczynnika bezwładności

Współczynnik bezwładności jest kolejnym elementem, który ma wpływ na tempo zbieżności sieci. Jeśli będzie zbyt duży, sieć może się przestać uczyć. Duża wartość tego współczynnika zwiększa tempo poprawiania sieci w kierunku ostatnio wyliczonych zmian gradientu. Dobrze jest to widoczne w przypadku około 70 iteracji kiedy większy współczynnik spowodował większy spadek dokładności, ale też szybszy powrót.

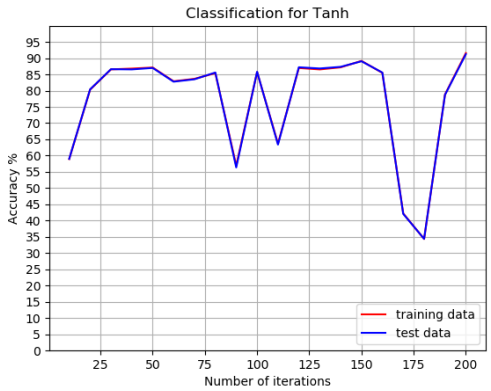
5.6 Rozmiar batcha

Sprawdzono następujące rozmiary:

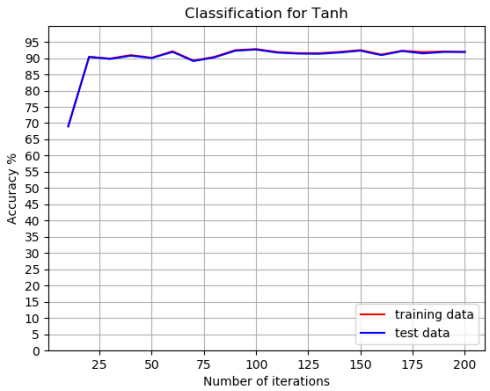
- 1,
- 10,
- 100,
- 1000.

Pozostała konfiguracja była następująca:

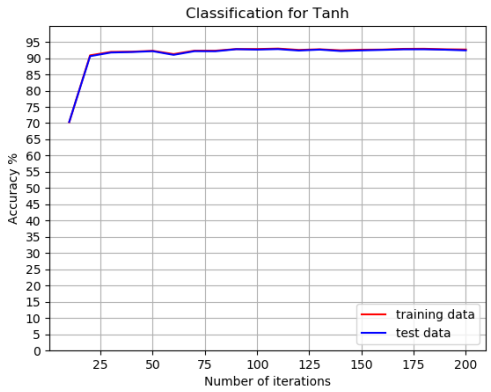
- rozmiar warstw ukrytych - [7],
- liczba iteracji - 200,
- funkcja aktywacji - Tangens hiprtboliczny,
- funkcja kosztu - Cross Entropy,
- obecność biasu - tak,
- współczynnik nauki - 0.5,
- współczynnik bezwładności - 0.1.



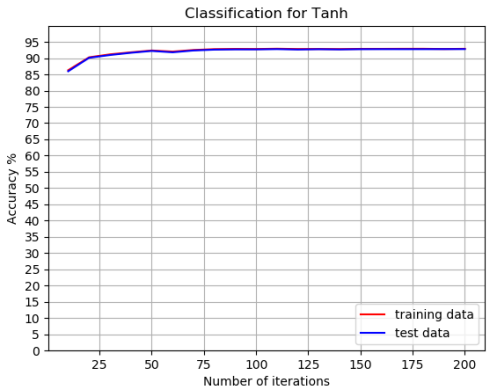
(a) Rozmiar batcha 1



(b) Rozmiar batcha 10



(c) Rozmiar batcha 100



(d) Rozmiar batcha 1000

Rysunek 15: Zależność skuteczności od rozmiaru batcha

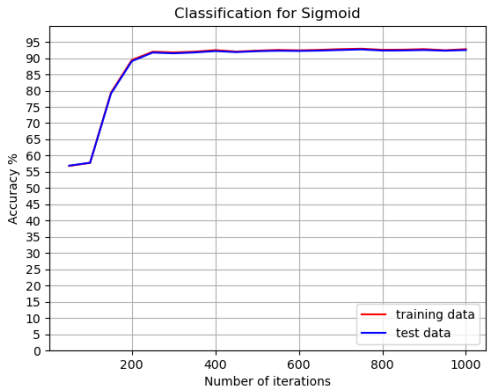
Zwiększenie rozmiaru batcha poprawia skuteczność sieci. Jest to spowodowane faktem, że przy tej samej liczbie iteracji poprawa wag jest wyliczana na podstawie większej liczby obserwacji. Dla batcha o rozmiarze 1 niestabilność skuteczności może być spowodowana stosunkowo małą liczbą obserwacji, na podstawie których wyliczana jest sieć. Przy maksymalnie 200 iteracjach daje to jedynie 200 obserwacji dla 10000-elementowego zbioru.

5.7 Obecność biasu

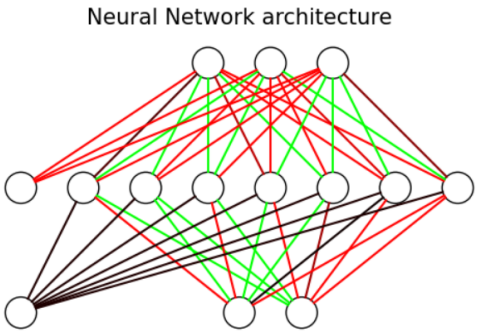
Przeprowadzono dwa testy w których sprawdzono wpływ biasu na proces uczenia się sieci. Przetestowano tą samą sieć z użyciem biasu i bez.

Konfiguracja sieci:

- rozmiar warstw ukrytych - [7],
- liczba iteracji - 1000,
- funkcja aktywacji - Sigmoid,
- funkcja kosztu - Quadratic,
- rozmiar batcha - 10,
- współczynnik nauki - 1.
- współczynnik bezwładności - 0.05.

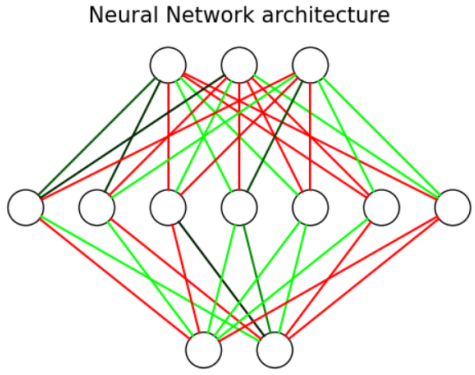


(a) Sieć z biasem



(b) Struktura sieci z biasem

(c) Sieć bez biasu



(d) Struktura sieci bez biasu

Rysunek 16: Zależność skuteczności od obecności biasu

Sieci z biasem i bez biasu otrzymały dla tego przykładu podobne wyniki. Na wykresach przedstawiono również struktury tych sieci. Jaśniejszy kolor zielony oznacza większą dodatnią wagę, jaśniejszy czerwony dużą ujemną wagę. Czarne krawędzie oznaczają wartości bliskie zera. Widać, że bias dodawany do warstwy wejściowej był równy zero lub bliski 0, prawdopodobnie z tego też powodu wagi na pierwszej warstwie mają podobne kolory - wartości. Jeśli chodzi o drugą warstwę wag bias odegrał na pewno dużą rolę i spowodował różnicę w wagach krawędzi.

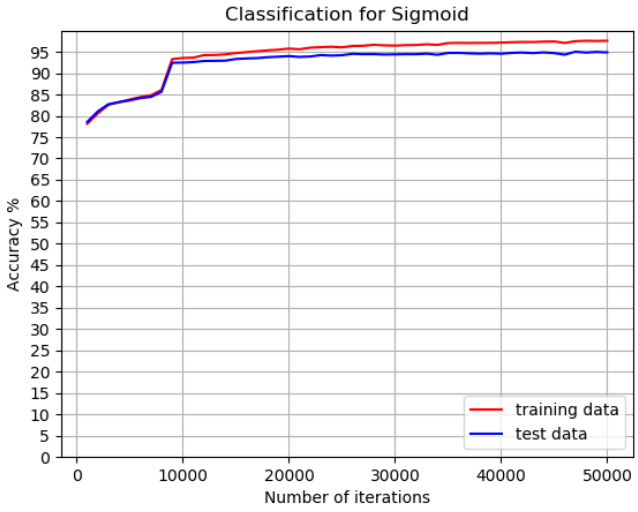
6 Rozpoznawanie cyfr - zbiór Kaggle Digit Recognizer

Korzystając z opinii odnośnie tego zbioru danych w internecie i w dużym stopniu testując losowo parametry udało się dobrać tak konfigurację, że skuteczność sieci jest na poziomie około 93%. Ostateczny najlepszy wynik jaki się udało osiągnąć na platformie Kaggle wyniósł 94.957%.

Ostatecznie najlepsze wyniki osiągnięte zostały dla sieci o następujących parametrach:

- jedna warstwa ukryta o rozmiarze 50,
- 50000 iteracji,
- Sigmoid jako funkcja aktywacji,
- Quadratic jako funkcja kosztu,
- obecny bias,
- batch o rozmiarze 10,
- współczynnik nauki o wartości 1,
- współczynnik bezwładności o wartości 0.05.

Skuteczność wybranej sieci został przedstawiony na rysunku 17.



Rysunek 17: Wykres skuteczności najlepszej sieci

Nicki na kaggle:

- ladmeerkat - Karol Ulanowski
- waszkiewiczj - Jakub Waszkiewicz

Literatura

[1] M. Nielsen, *Neural Networks and Deep Learning*

[2] 3Blue1Brown, *Backpropagation calculus — Deep learning, chapter 4*, https://www.youtube.com/watch?v=tIeHLnjs5U8

[3] A list of cost functions used in neural networks, https://stats.stackexchange.com/questions/154879/a-list-of-cost-functions-used-in-neural-networks-alongside-applications