

Metody inteligencji obliczeniowej w analizie danych

Sprawozdanie z budowy modelu perceptronu wielowarstwowego - sieci neuronowej typu feedforward

Damian Skowroński — 313506

11 kwietnia 2023

1 Wprowadzenie

Celem projektu była implementacja sieci neuronowej typu MLP (*Multi-Layer Perceptron*), którą można wykorzystać do prostych zadań klasyfikacji i regresji. Implementacja sieci była kolejno rozszerzana o następujące funkcjonalności:

- uczenie poprzez propagację wsteczną błędu
- wizualizacja wag
- uczenie z momentem i/lub normalizacją błędu
- rozwiązywanie zadań klasyfikacji
- regularyzacja wag i zatrzymywania uczenia przy wzroście błędu na zbiorze walidacyjnym

To sprawozdanie będzie się skupiać głównie na wnioskach wyciągniętych podczas pracy z użyciem sieci na prostych zbiorach, przy różnym stopniu rozwinięcia implementacji sieci, a nie na samym kodzie i sposobie implementacji sieci.

2 Bazowa implementacja

Pierwszy krok polegał po prostu na implementacji sieci, w której można ustawać ręcznie liczbę warstw, liczbę neuronów w każdej z warstw i wagi poszczególnych połączeń, w tym biasów. Na razie wykorzystywana funkcje aktywacji na warstwach ukrytych to funkcja sigmoidalna $\sigma(x) = \frac{e^x}{e^x + 1} = \frac{1}{1 + e^{-x}}$, natomiast na wyjściu jest funkcja liniowa. Sieć w obecnym stanie nie potrafi się uczyć, a jedynie na podstawie ustawionych wag i biasów z inputu dać jakiś output.

2.1 Test działania

Należało dla architektur sieci:

- 1 warstwa ukryta o 5 neuronach
- 1 warstwa ukryta o 10 neuronach
- 2 warstwy ukryte po 5 neuronów

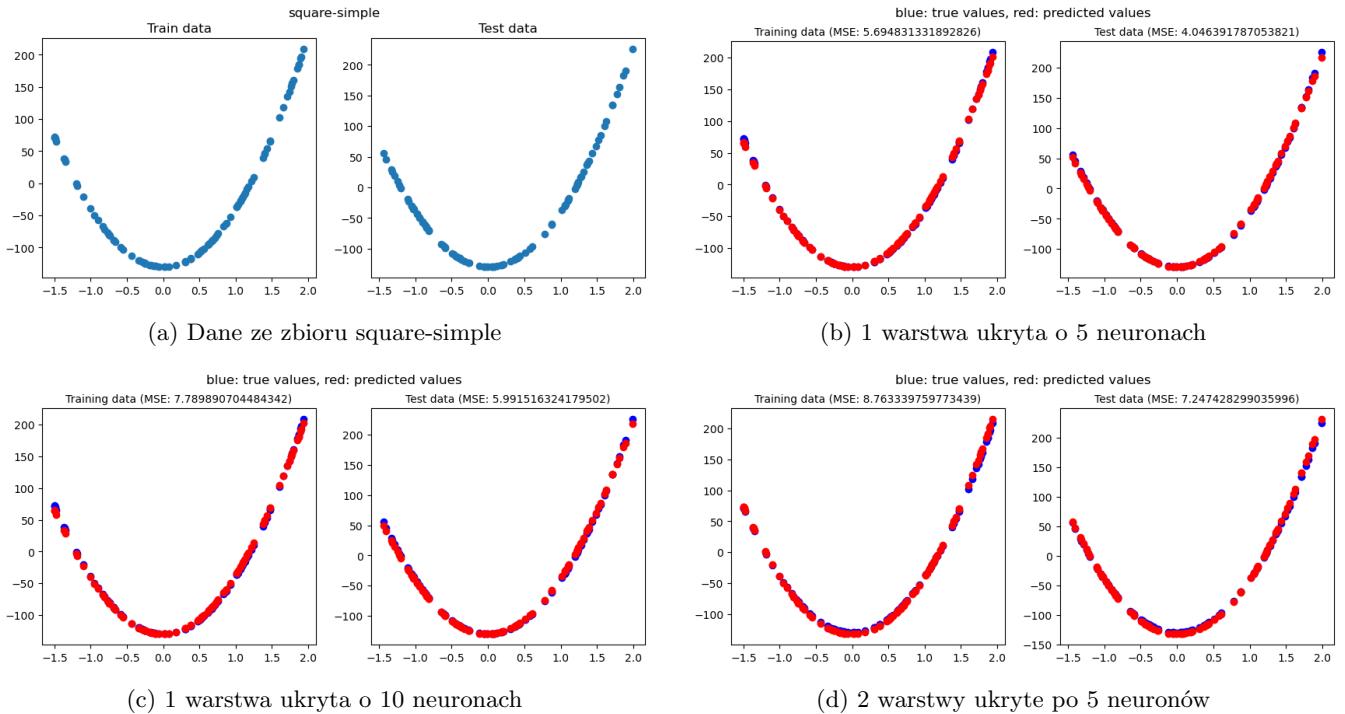
dopasować ręcznie wagi i biasy dla dwóch prostych zbiorów, tak aby MSE (*Mean Squared Error*) były wystarczająco niskie. Oba zbiory mają jeden input (przedstawiany na wykresach na poziomej osi) i jeden przewidywany output (na osi pionowej).

2.1.1 zbiór square-simple

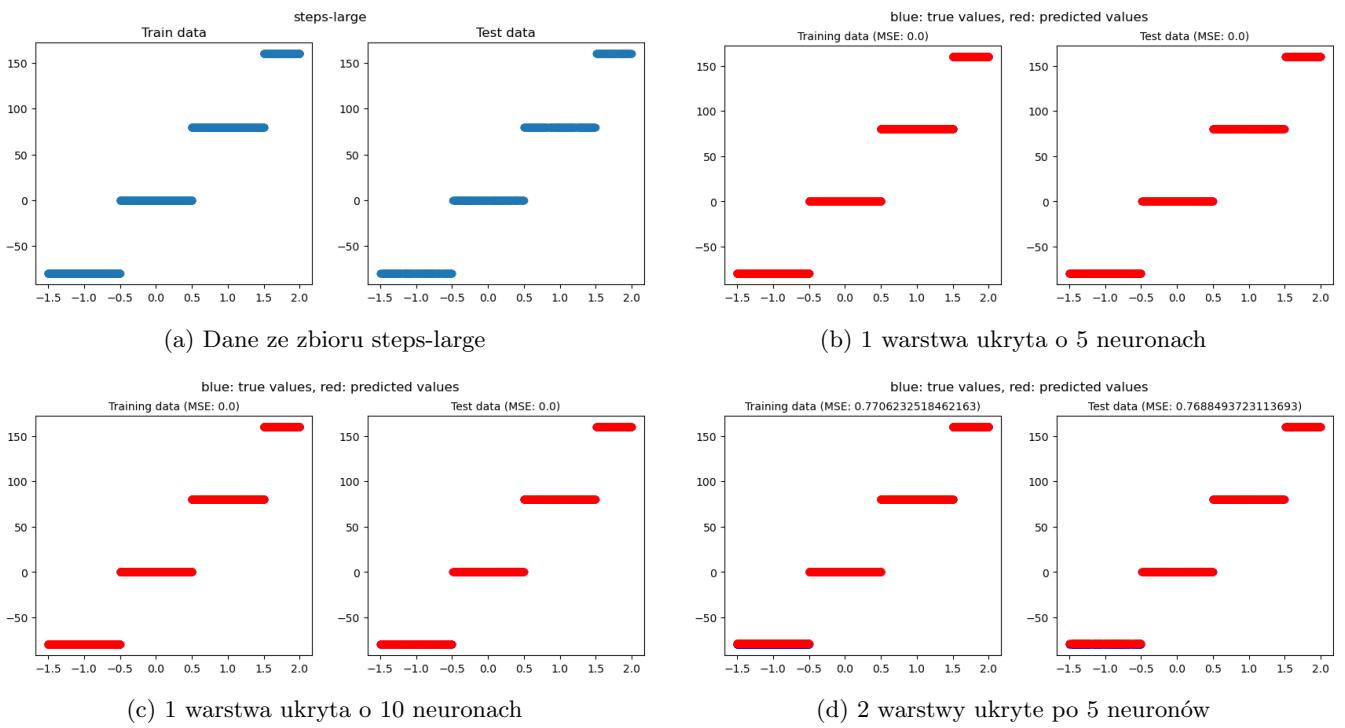
Zbiór zawiera wartości z funkcji kwadratowej dla pewnej próbki argumentów. W zbiorze treningowym i testowym jest po 100 obserwacji. Wykresy poniżej (Rysunek 1) przedstawiają wyniki z ręcznego dopasowywania wag i biasów dla wcześniej opisanych architektur. Punkty na niebiesko oznaczają prawdziwe wartości, a na czerwono wartości przewidywane przez sieć (na wszystkich wykresach za punkty niebieskie, tylko w dużej części są zasłonięte przez punkty czerwone). Nad wykresami napisane są wartości MSE dla kolejnych dopasowań.

2.1.2 zbiór steps-large

Wartości zbioru przypominają schody. W zbiorze treningowym jest 10000 obserwacji, a w testowym jest ich 1000. Wykresy na rysunku 2 przedstawiają informacje tak samo jak wcześniej opisane wykresy. Tutaj widać, że da się dobrze wagi tak, żeby punkty idealnie się pokrywały. W architekturach o jednej warstwie wystarczyłyby 3 neurony, bo tak naprawdę wagi pozostałych są ustalone na 0, czyli są po prostu wyłączone.



Rysunek 1: Ręczne dopasowanie do zbioru square-simple



Rysunek 2: Ręczne dopasowanie do steps-large

2.2 Wnioski

Wnioski z tego kroku są następujące:

- Dla prostych zbiorów da się dopasować wagi i biasy ręcznie, aby sieć dobrze przewidywała wartości, jednak nie jest to łatwe, ani przyjemne.
- Im więcej neuronów tym więcej parametrów do ustawiania, a co za tym idzie sieć może być dopasowana do bardziej skomplikowanych zadań.
- Większa ilość neuronów oznacza również zwiększoną trudność w ustawianiu poprawnych wag.
- Dużo trudniej było ustawić wagi dla 2 warstw niż dla jednej, ponieważ oprócz tego, że było więcej parametrów (na drugiej warstwie była macierz 5×5 , czyli 25 elementów), to też trudniej jest zrozumieć ich wpływ na output.

3 Propagacja wsteczna błędu

Następny krok implementacji to dodanie możliwości uczenia poprzez propagację wsteczną błędu (*backpropagation*). Oprócz tego do sieci dodalem inicjalizację wag i biasów jako obserwacji z rozkładu normalnego o średniej w 0 i odchyleniu standardowym równym $\sqrt{\frac{2}{d_i}}$, gdzie d_i oznacza liczbę zmiennych objaśniających. Zaimplementowana propagacja wsteczna pozwala na trenowanie sieci z aktualizacją wag i biasów po:

- wszystkich wzorcach (*Batch Gradient Descent*)
- pojedynczym wzorcu (*Stochastic Gradient Descent*)
- porcji wzorców (*Minibatch Gradient Descent*)

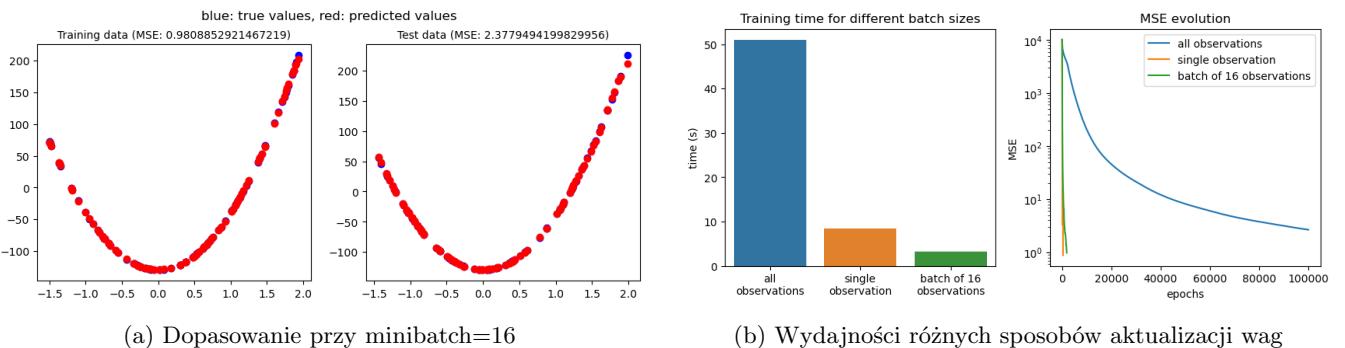
Sieć nie ma żadnych zabezpieczeń na sytuację eksplodujących gradientów, więc w celu weryfikowania takich sytuacji do implementacji doszła również możliwość wizualizacji wag.

3.1 Test działania

Żeby sprawdzić działanie uczenia sieci należało wykonać testy dla każdej z wcześniej wspomnianej aktualizacji wag. Podczas eksperymentu starałem się ręcznie dobierać odpowiednie parametry, aby mieć dobre porównanie.

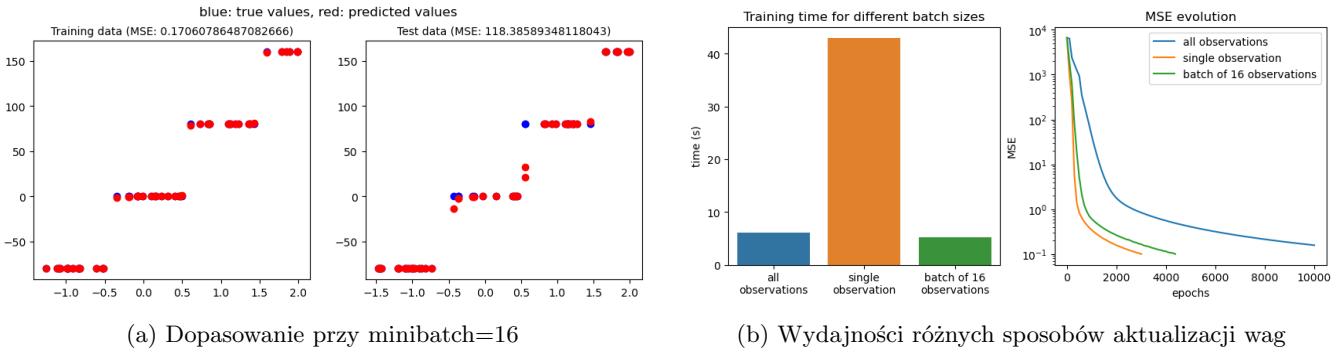
3.1.1 square-simple

Pierwszy test uczenia został wykonany na tym samym zbiorze, do którego dopasowywałem wcześniej w sekcji 2 wagi. Zbiór nie jest skomplikowany, więc sieć nie miała problemu z dopasowaniem. Wyniki uczenia przedstawione są na rysunku 3.



Rysunek 3: Uczenie sieci do zbioru square-simple

Na lewym wykresie (a) widać, że sieci udało się dobrze dopasować do danych. Wyniki są dużo lepsze, niż kiedy próbowałem to wykonać ręcznie. Na wykresie prawym (b) widać, że aktualizacja wag po prezentacji wszystkich wzorców nie poradziła sobie dobrze. Nawet po 100000 epoch nie udało się otrzymać tak dobrego MSE, jakie udało się pozostałym sposobom w dużo krótszym czasie.



Rysunek 4: Uczenie sieci do zbioru steps-small

3.1.2 steps-small

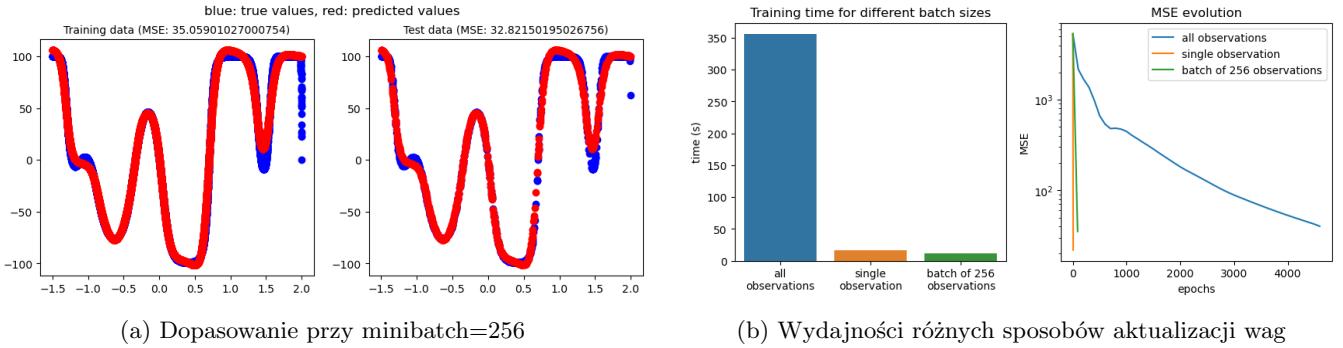
Drugi test uczenia został przeprowadzony na zbiorze steps-small, który różni się od zbioru steps-large z sekcji 2 tym, że ma dużo mniej obserwacji. Na rysunku 4 widać wyniki z uczenia sieci.

Na lewym wykresie (a) widać, że sieci udało się dobrze dopasować do danych treningowych ($MSE \approx 0.17$), natomiast dla danych testowych widać, że wynik nie jest tak samo dobry ($MSE \approx 118$). Myślę, że takie wyniki są spowodowane faktem, że w zbiorze testowym niektóre obserwacje wychodzą poza zakres tych z treningowego, przez co sieć nie mogła się do nich dobrze przystosować. Co prawda dopasowanie i tak wydaje się bardzo dobre, po prostu MSE jest znacznie większe, bo zakres przewidywanej wartości jest duży, więc jeśli model się pomylił dla którejś wartości (tutaj przewidział, że wartość jest gdzieś pomiędzy schodami) to ten błąd jest duży.

Dla tego zbioru prawy wykres jest ciekawszy niż dla poprzedniego. Widzimy, że czasowo trenowanie najdłużej zajęło aktualizacji wag po każdym wzorcu, natomiast potrzeba było na to najmniejszej ilości epoch i wynik MSE jest dobry. Pozostałe dwa sposoby aktualizacji wag zajęły podobną względem siebie ilość czasu, jednak aktualizacja po wszystkich wzorach znowu potrzebowała dużo większej ilości epoch, i ostatecznie otrzymała lekko gorszy wynik MSE .

3.1.3 multimodal-large

Trzeci test uczenia został wykonany na zbiorze multimodal-large, który jest większym i bardziej skomplikowanym zbiorem niż poprzednie, przez co dopuszczalne MSE na zbiorze treningowym przy uczeniu było mniejsze. Wyniki przedstawione są na rysunku 5.



Rysunek 5: Uczenie sieci do zbioru multimodal-large

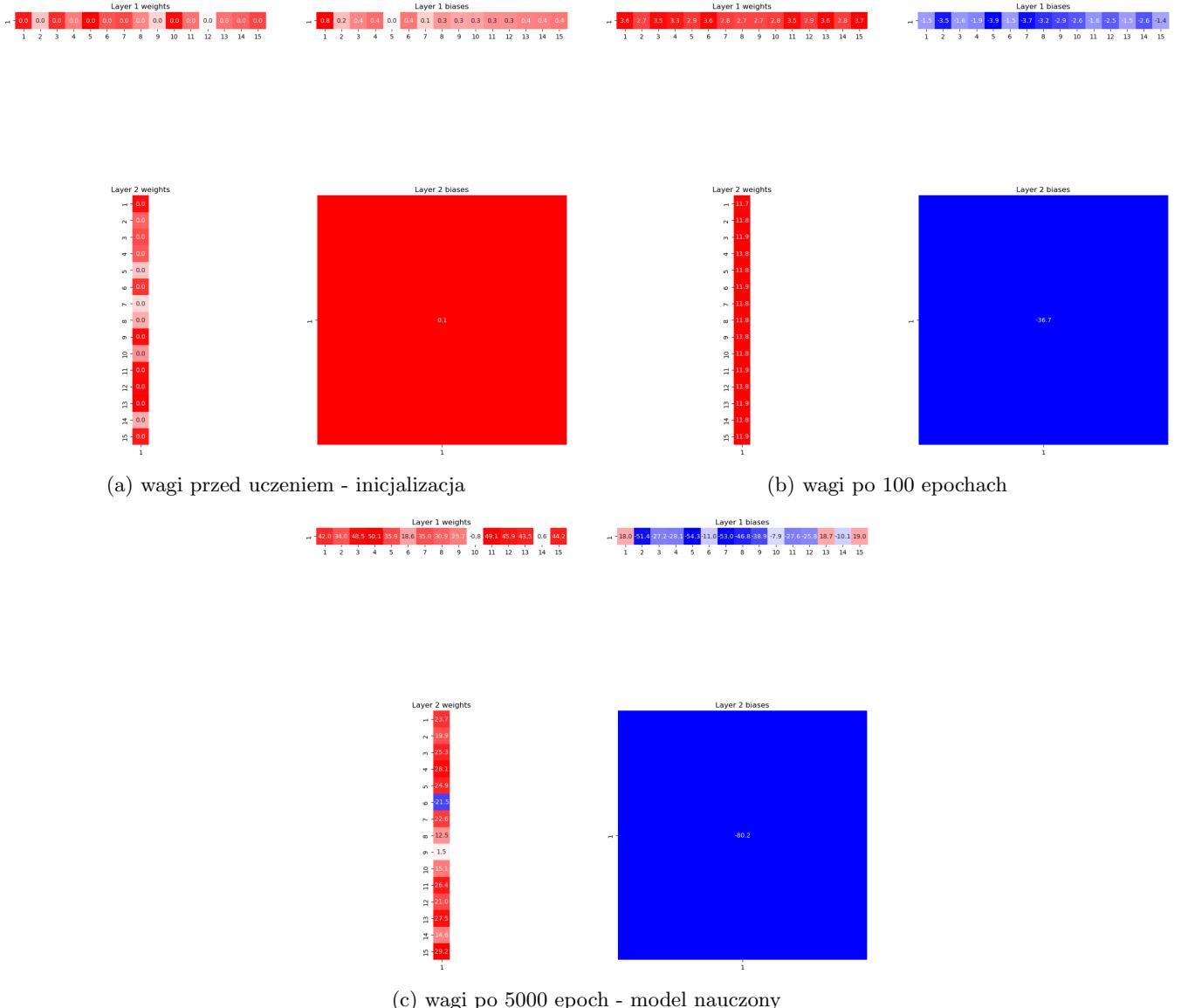
Na lewym wykresie (a) widać, że sieć mogłaby się trochę lepiej dopasować (chodzi o część z prawej danych treningowych), jednak nie ma to dużego wpływu na dane testowe, które nie mają tak dużo wartości z tego przedziału.

Na prawym wykresie (b) widać podobne wyniki co przy pierwszym zbiorze. Aktualizacja wag po wszystkich wzorcach wyszła najgorzej pod względem czasu, epoch i MSE . Pozostałe dwie metody wydają się działać równie dobrze.

3.2 Wizualizacja wag

Do lepszego zrozumienia uczenia i wybierania odpowiedniego *learning_rate* czasem pomagała mi wcześniej wspomniana wizualizacja wag. Rysunek 6 pokazuje, jak taka wizualizacja mogła wyglądać na prostym modelu (niestety wykresy są duże, i zmniejszone o tyle, że nie widać dokładnie napisów). Dla każdego z 3 wykresów wagi są z lewej strony, a biasy z prawej, każdy rząd to inną warstwę. Wobec tego widać, że ta sieć ma dwie warstwy ukryte, na pierwszej jest 10 neuronów, a na drugiej 1 neuron. Na wykresie (a) widać inicjalizację wag (wówczas z rozkładu uniform na przedziale od 0 do 1). Na wykresie (b) widać, że wagi i biasy pozmieniały się, szczególnie widać to na biasach, które są teraz na minusie (na załączonych wykresach nie widać dokładnych wartości, ale ważne jest, że wartości czerwone są dodatnie,

a niebieskie - ujemne, im mocniejszy odcień danego koloru tym dalej od 0 jest dana wartość). Na wykresie (c) widać już sieć po dłuższym uczeniu.



Rysunek 6: Wizualizacja wag modelu o 2 warstwach - pierwszej z 10 neuronami, drugiej z 1 neuronem

3.3 Wnioski

Wnioski z tego kroku są następujące:

- Nie ma większych problemów z dopasowaniem sieci przy użyciu samej (w sensie bez ulepszeń) propagacji wstecznej na prostych zbiorach.
- Na różnych zbiorach sposoby aktualizowania wag działały różnie, czasem jedno podejście było znaczowo gorsze niż inne.
- Aktualizacja wag po prezentacji wszystkich wzorców potrzebowała bardzo wielu epoch.
- Aktualizacja wag po prezentacji pojedynczych rezultatów potrzebowała najmniejszej liczby epoch, jednak często, dla większych zbiorów, wiązało się to z bardzo długim czasem poświęconym na każdą z nich.
- Z wykonanych testów wydaje się, że najlepszym podejściem była aktualizacja po jakimś minibatchu. We wszystkich 3 testach dawało to bardzo dobre wyniki w krótkim czasie i w niewielkiej ilości epoch.
- Wybranie odpowiedniego *learning_rate* było kluczowe dla dobrego uczenia sieci.
- Nie wszystkie sposoby aktualizowania wag miały to samo optymalne *learning_rate*, zwykle dla aktualizacji po prezentacji pojedynczego wzorca potrzebna było podzielić *learning_rate* przez 10 w porównaniu do pozostałych dwóch sposobów.

4 Uczenie z momentem i normalizacja gradientu

W celu polepszenia szybkości i zbieżności procesu uczenia sieci, kolejny krok implementacji polegał na dodaniu możliwości skorzystania z:

- uczenia z momentem (*momentum*)
- normalizacji gradientu (*RMSProp*)
- optymalizatora Adam (połączenia momentum i RMSProp)

4.1 Test działania

W celu sprawdzenia i porównania tych metod przeprowadziłem na kilku zbiorach eksperyment który wyglądał następująco:

- dla każdej metody wybieram odpowiednie, optymalne parametry
- uczę sieć z danym parametrem 10 razy, dla maksymalnej liczby epoch równej 1000
- porównuję sieci, których MSE jest medianą dla danej metody, porównuję tu też bazową implementację

Przy wszystkich metodach wykorzystywałem aktualizację minibatch. Wielkość minibatchy zależała od wielkości zbioru, ale na poziomie zbioru jej nie zmieniałem, czyli nie powinno mieć to znaczenia w porównywaniu metod.

4.1.1 square-large

Ten zbiór jest podobny do wcześniej wykorzystawanego square-simple. Różni się tym, że obserwacji jest dużo więcej i część testowa i treningowa nie zostały dobrze podzielone (widać na rysunku 7 wykresie (a), że w zbiorze treningowym nie ma wcale wartości poniżej -1.5, a testowym jest ich wiele), przez co nie ma sensu zajmować się MSE na zbiorze testowym.

Z rysunku 7 widać, że RMSProp i Adam poradziły sobie dużo lepiej niż bazowa sieć, czy sieć z uczeniem z momentem. Na wykresie zbieżności MSE widać też, że RMSProp i Momentum robiły "skoki" (tutaj kolejne punkty są co 10 epoch), natomiast base i Adam działały bardziej stabilnie (w sensie ich wykresy są bardziej płaskie, a nie takie ze szczytami i dolinami).

Z tabeli 1, w której są wyniki eksperymentu widać, że Adam był bardzo niezawodny (w sensie za każdym razem dawał dobre wyniki w krótkim czasie). RMSProp czasem potrafił być bardzo szybki (wyniki w 63, 81, 89 epoch i czasie poniżej sekundy). Bazowa implementacja zwykle osiągała dobry wynik, ale długo jej to zajmowało. Uczenie z momentem otrzymywało wyniki szybciej niż bazowa implementacja, ale czasem się "gubiło" i kończyło z większym MSE.

4.1.2 steps-large

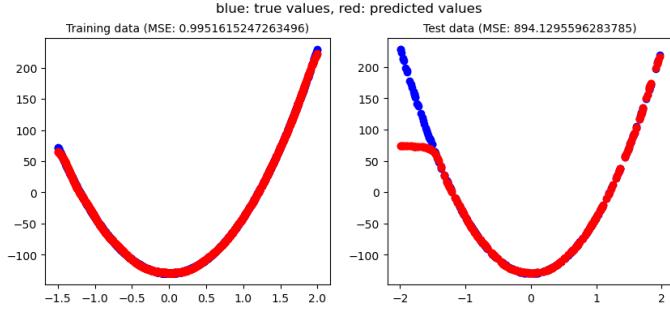
Ten zbiór był już używany w sekcji 2. Na rysunku 8 z wykresu (b), na którym jest porównanie czasów uczenia widać, że Adam spisał się najlepiej. Wykres zbieżności MSE jest tym razem bardziej chaotyczny. Tutaj też krzywa dla Adama wygląda najlepiej, jest najbardziej płaska i przede wszystkim najszybciej dochodzi do wyniku. Pomiędzy pozostałymi metodami nie było tu dużej różnicy.

Z tabeli 2 można wyczytać, że wszystkie metody oprócz uczenia z momentem okazały się solidne (pod tym względem, że każdej z nich udawało się dojść do MSE mniejszego niż 10). Widać również, że Adam zadziałał tutaj zdecydowanie najlepiej. Oprócz dwóch sytuacji za każdym razem udało mu się nauczyć w poniżej 200 epoch i 5 sekund, a kilka razy nawet poniżej 2 sekund.

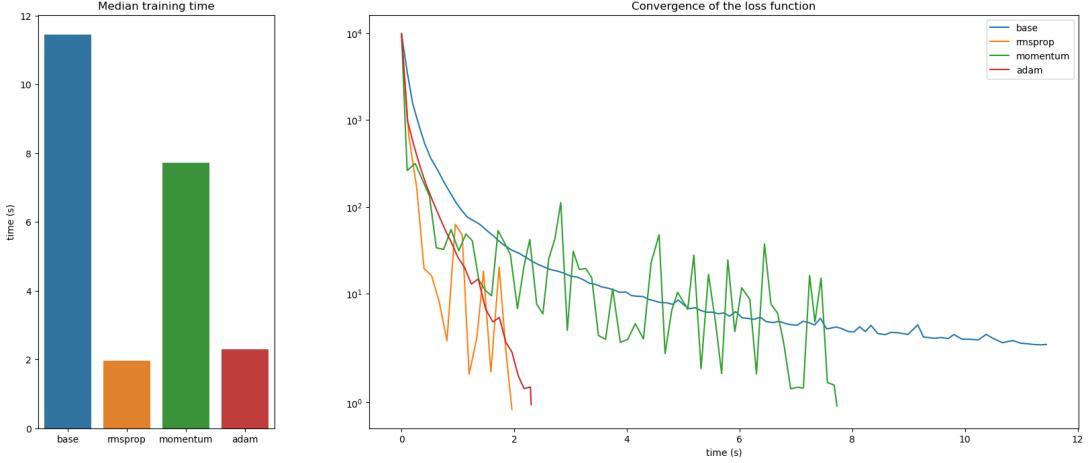
4.1.3 multimodal-large

Ten zbiór też był już używany w sekcji 3. Z rysunku 9 z wykresu porównywanego czasów widać, że znowu najlepiej spisał się Adam. Podobnie na wykresie obok widać, że krzywa odpowiadająca Adama wygląda najlepiej. Krzywe pozostałych metod wyglądają podobnie do siebie i nie ma jakichś bardzo znaczących różnic.

Z tabeli 3 można odczytać, że wszystkie sieci, oprócz jednej z Adamem, osiągnęły oczekiwane MSE (mniejsze niż 9) w liczbie epoch dużo mniejszej niż maksymalna (1000). Czasy uczenia i liczba epoch były bardzo podobne dla sieci nie używających Adama i były zwykle trochę gorsze niż sieci używające Adama. Adam znowu wydaje się otrzymywać wyniki najszybciej, oprócz tej jeden wcześniej wspomnianej sieci, gdzie MSE nie zeszło poniżej 9 w 1000 epoch.

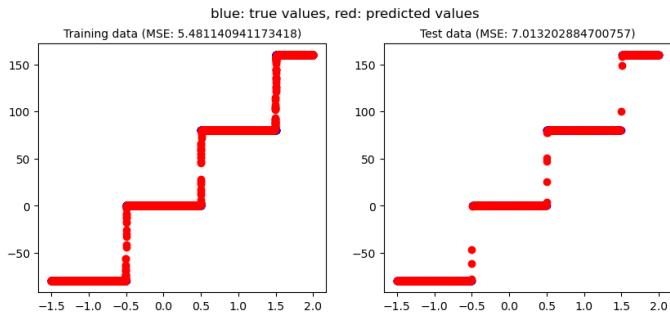


(a) Dopasowanie przy minibatch=32, 1 warstwa 30 neuronów

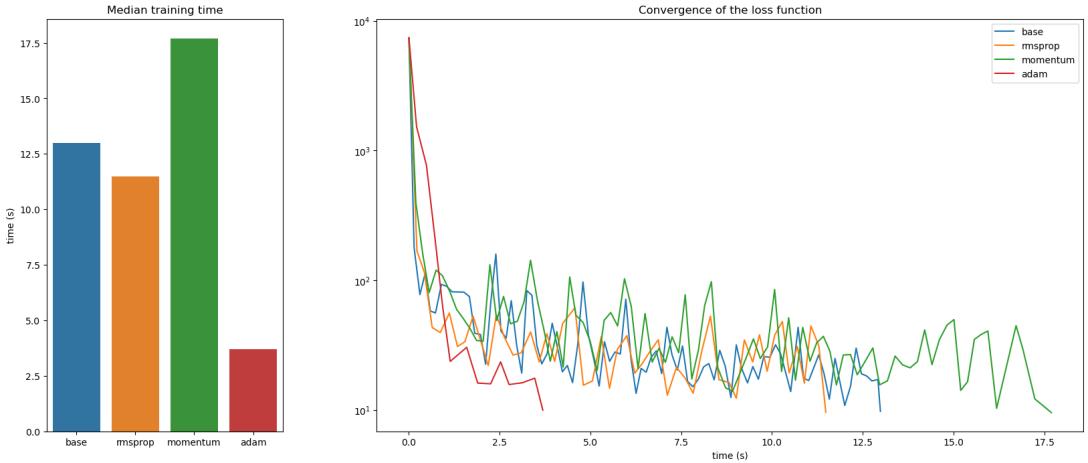


(b) Porównanie wydajności różnych metod uczenia

Rysunek 7: Uczenie sieci do zbioru square-large



(a) Dopasowanie przy minibatch=500, 2 warstwy po 10 neuronów



(b) Porównanie wydajności różnych metod uczenia

Rysunek 8: Uczenie sieci do zbioru steps-large

TIME				EPOCH				MSE			
base	rmssp.	mmnt.	adam	base	rmssp.	mmnt.	adam	base	rmssp.	mmnt.	adam
13.04	5.66	13.78	2.37	1000	333	1000	202	5.03	0.98	11.63	0.94
11.29	4.01	13.74	2.45	1000	285	1000	210	2.67	0.97	34.65	0.88
11.97	1.72	10.58	2.40	1000	148	1000	207	2.59	0.85	21.26	0.94
10.17	2.02	1.50	2.34	1000	181	149	197	2.76	0.82	0.98	0.96
9.95	2.01	2.93	2.29	1000	190	288	201	1.05	0.98	0.96	0.95
10.17	0.88	6.57	1.82	974	81	634	154	0.99	0.97	0.93	0.88
10.56	0.98	3.20	2.45	1000	89	286	206	1.36	0.73	0.87	0.94
10.18	10.68	4.81	1.43	1000	1000	486	135	3.01	8.88	0.96	0.90
9.90	0.67	10.59	2.34	1000	63	1000	206	2.47	0.93	30.64	0.91
9.78	1.30	10.37	1.10	927	116	1000	92	0.99	0.97	5.05	0.82

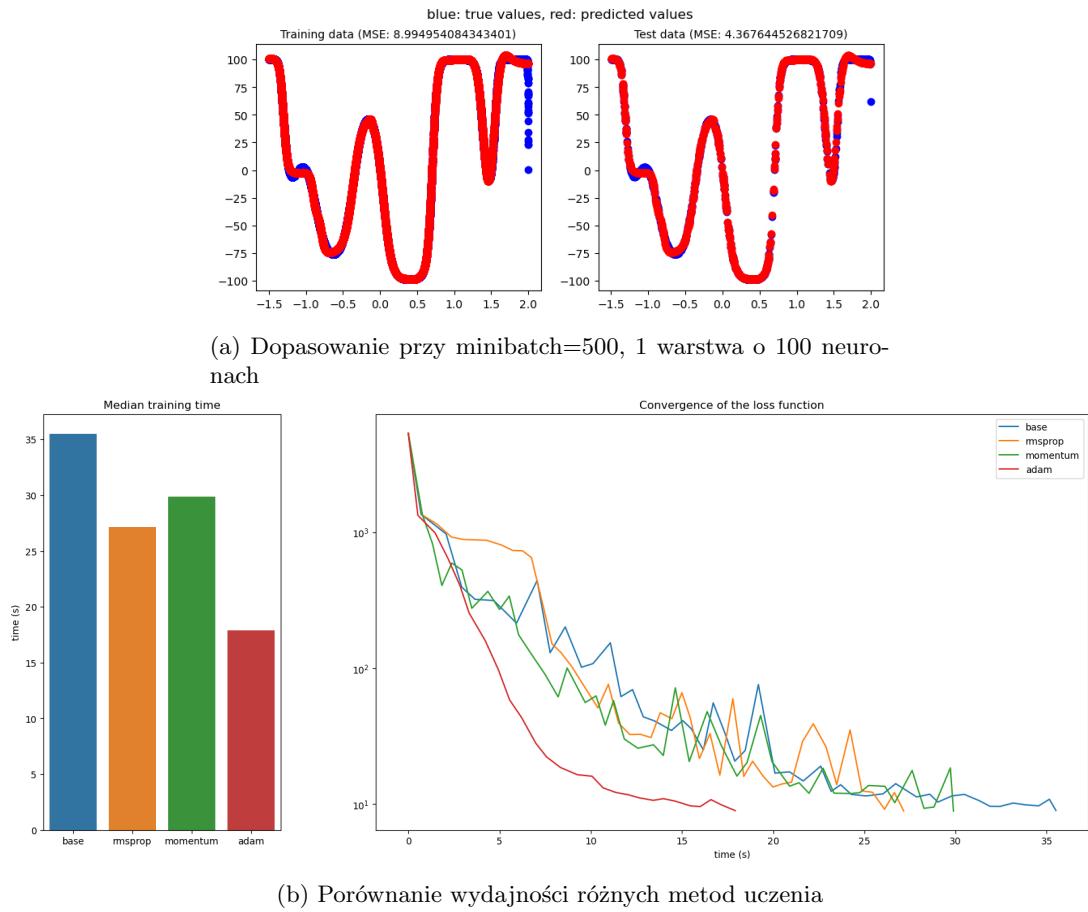
Tabela 1: Wyniki benchmarku metod usprawniających uczenie dla zbioru square-large

TIME				EPOCH				MSE			
base	rmssp.	mmnt.	adam	base	rmssp.	mmnt.	adam	base	rmssp.	mmnt.	adam
17.53	11.48	23.27	4.67	666	509	1000	224	9.83	9.62	30.53	9.63
12.15	14.54	17.70	6.85	616	721	876	165	9.89	9.15	9.54	9.88
11.11	13.87	55.92	14.64	554	578	991	258	9.34	9.26	9.41	9.82
28.66	11.58	19.63	3.93	704	560	1000	182	9.94	9.52	16.28	9.90
22.43	10.21	15.20	4.83	760	478	810	192	9.97	9.96	9.19	9.86
20.88	12.69	27.57	3.69	802	476	1000	127	9.40	9.48	19.77	9.97
15.74	11.65	36.28	3.35	719	458	1000	174	9.53	9.65	39.82	9.80
11.44	9.94	15.29	1.86	641	562	972	112	9.89	9.72	9.99	9.44
12.48	7.91	15.18	1.76	834	496	1000	111	8.58	9.75	31.36	9.65
12.99	10.25	12.16	2.65	845	652	827	169	9.75	9.98	9.76	9.71

Tabela 2: Wyniki benchmarku metod usprawniających uczenie dla zbioru steps-large

TIME				EPOCH				MSE			
base	rmssp.	mmnt.	adam	base	rmssp.	mmnt.	adam	base	rmssp.	mmnt.	adam
42.25	29.26	27.19	20.40	414	428	388	317	8.88	8.71	8.97	8.85
40.61	25.61	29.89	11.27	585	357	443	167	8.96	8.77	8.89	8.95
28.30	25.63	27.57	17.92	400	386	430	260	8.88	8.99	8.81	8.97
28.98	26.09	32.53	18.98	460	401	499	263	8.96	8.93	8.84	8.94
26.60	24.87	27.58	68.65	381	393	405	1000	8.86	8.53	8.96	548.06
51.95	32.71	28.85	17.66	376	520	445	266	8.98	8.57	8.85	8.99
44.47	28.93	40.55	17.20	681	419	449	207	8.99	8.93	8.91	8.56
35.50	57.82	35.27	15.96	475	545	419	188	8.99	8.74	8.99	8.87
44.56	39.92	42.27	21.01	415	413	454	255	8.99	8.98	8.94	8.99
28.70	27.16	32.33	18.02	435	427	399	271	8.91	8.90	8.95	8.85

Tabela 3: Wyniki benchmarku metod usprawniających uczenie dla zbioru multimodal-large



Rysunek 9: Uczenie sieci do zbioru multimodal-large

4.2 Wnioski

Wnioski z tego kroku są następujące:

- RMSProp zwykle osiągało oczekiwane MSE szybciej niż bazowa implementacja zarówno jeśli chodzi o epoki jak i o czas.
- Uczenie z momentem czasem dawały lepsze wyniki, jednak nie jakoś znacząco.
- Sieci wykorzystujące metodę Adam do optymalizacji prawie zawsze dawały najlepsze wyniki pod względem wartości MSE, szybkości, liczby epoch i niezawodności.

5 Klasyfikacja

Do tej pory sieć radziła sobie tylko z problemami regresji. W tym kroku zmodyfikowałem ją, aby radziła sobie również z klasyfikacją, czyli na wyjściu zostały dodane prawdopodobieństwa liczone funkcją softmax, a funkcja straty to cross entropia.

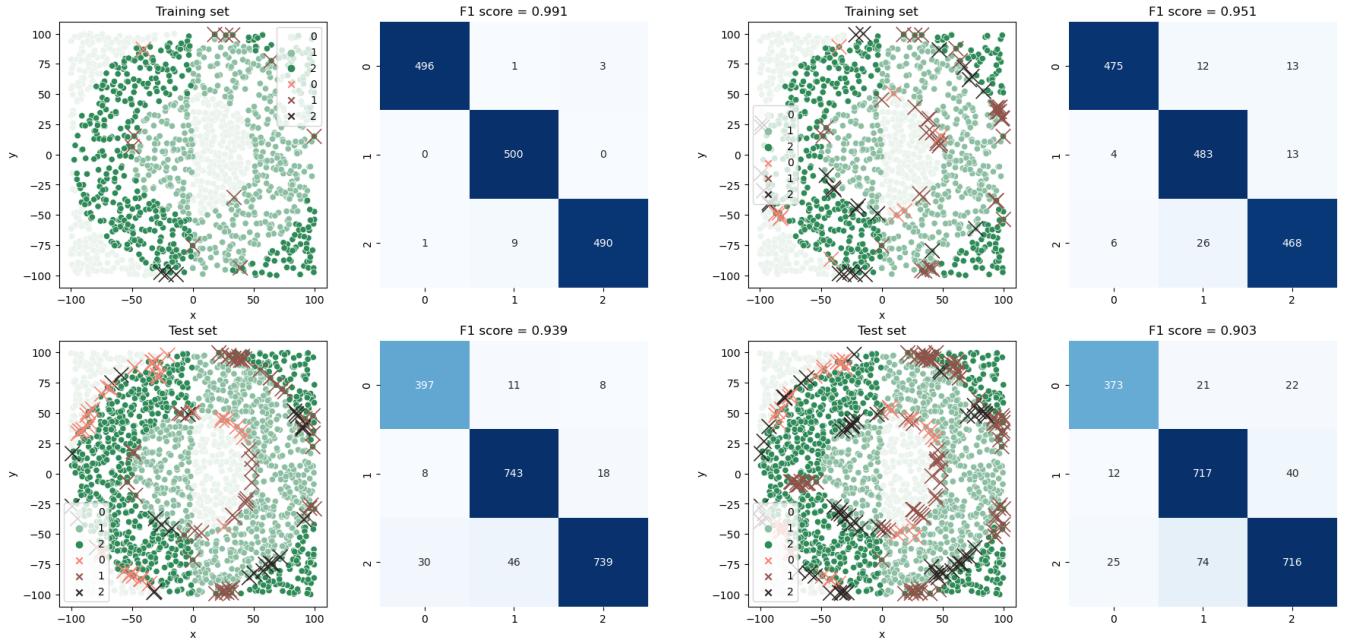
5.1 Test działania

Aby sprawdzić działanie klasyfikacji przeprowadziłem uczenie na 3 zbiorach. Zbiory mają dwie zmienne objaśniające reprezentowane na osiach wykresów, oraz zmienną objaśnianą, która na wykresach jest kolorem punktów. Sieci były trenowane z funkcją aktywacji na warstwie wyjścia softmax lub liniową, aby sprawdzić, czy softmax daje lepsze wyniki. Główną miarą do porównywania będzie *F1 score* na zbiorze treningowym. Wszystkie sieci miały architekturę 3 warstw ukrytych po 100 neuronów z sigmoidalną funkcją aktywacji.

5.1.1 rings3-regular

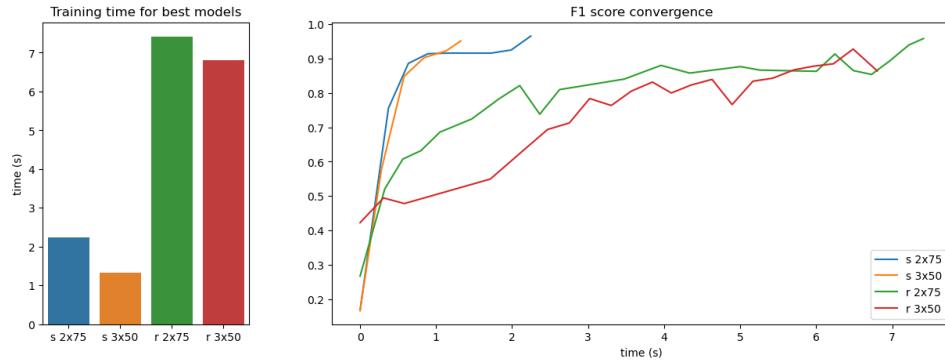
Zbiór ma 3 klasy, które układają się w takie przecinające się pierścienie. Na rysunku 10, patrząc na wykres (a) widać, że sieć bardzo dobrze nauczyła się do zbioru treningowego, i zwróciła całkiem dobry wynik dla zbioru testowego. Tak naprawdę pomyliła się tylko na granicach klas. Natomiast na wykresie (b) widać, że sieć wykorzystująca funkcję liniową na warstwie wyjścia gorzej przystosowała się do zbioru. Widać, że miała gorszy F1 score, a także na zbiorze

testowym myliła się w miejscach bardziej w głębi pola klasy, a nie tylko na granicach. Na wykresie (c) widać, że sieć z funkcją liniową potrzebowała dużo więcej czasu i więcej epoch, a i tak dała gorsze wyniki.



(a) funkcja aktywacji warstwy wyjściowej - softmax

(b) funkcja aktywacji warstwy wyjściowej - liniowa



(c) porównanie wyników w zależności od funkcji aktywacji funkcji wyjściowej

Rysunek 10: Uczenie sieci do zbioru rings3-regular

5.1.2 easy

Prosty zbiór z dwiema klasami rozdzielonymi prostą linią. Na rysunku 11 widać, że na tym zbiorze obie sieci nie miały problemu z nauczeniem się i osiągnęły te same wyniki. Na wykresie (c) widać jednak, że sieć z funkcją liniową potrzebowała więcej czasu. Nie widać tego na wykresie, ale sieć z softmaxem potrzebowała 5 epoch, a liniowa potrzebowała 10.

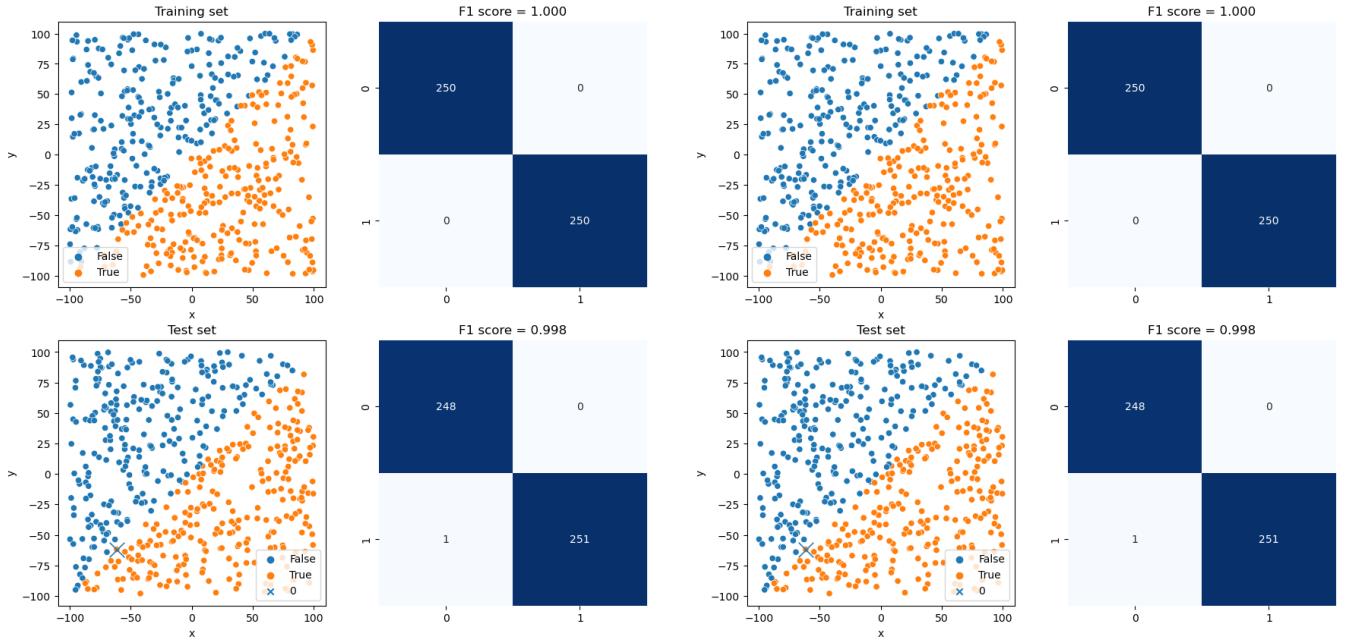
5.1.3 xor3

Tutaj znowu była klasyfikacja binarna, ale na trochę bardziej skomplikowanym zbiorze. Patrząc na rysunek 12, można zauważyć, że obie sieci otrzymały ten sam F1 score na zbiorze treningowym, jednak na zbiorze testowym sytuacja wygląda trochę lepiej dla funkcji softmax, która mniej się myliła. Na wykresie (c) widać podobną sytuację, co dla poprzednich 2 zbiorów, czyli funkcja liniowa potrzebowała więcej czasu i epoch.

5.2 Wnioski

Wnioski z tego kroku są następujące:

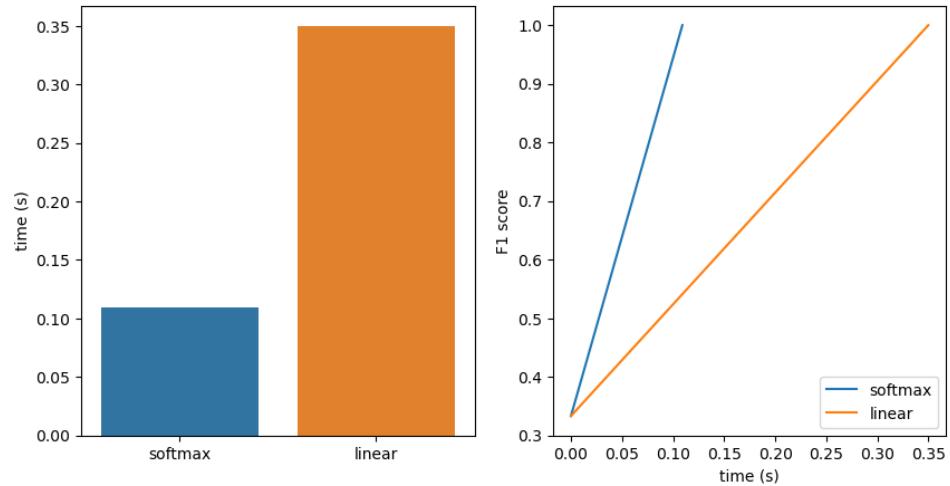
- Nie jest trudno zmodyfikować sieć rozwiązyującą problemy regresji, aby mogła rozwiązywać też problemy klasyfikacji (przynajmniej na ciągłych zmiennych objaśniających, nie testowałem na zmiennych np. binarnych).
- Funkcja softmax na wyjściu na pewno nie jest jedyną funkcją która nadaje się do klasyfikacji, zwykła funkcja liniowa też potrafi się dopasować do zbioru.



(a) funkcja aktywacji warstwy wyjściowej - softmax

(b) funkcja aktywacji warstwy wyjściowej - liniowa

Comparison for different output activation functions



(c) porównanie wyników w zależności od funkcji aktywacji funkcji wyjściowej

Rysunek 11: Uczenie sieci do zbioru easy

- Funkcja softmax na wyjściu sprawowała się dużo lepiej niż funkcja liniowa. Uczenie zajmowało jej mniej czasu i zwykle dawała też lepsze wyniki.

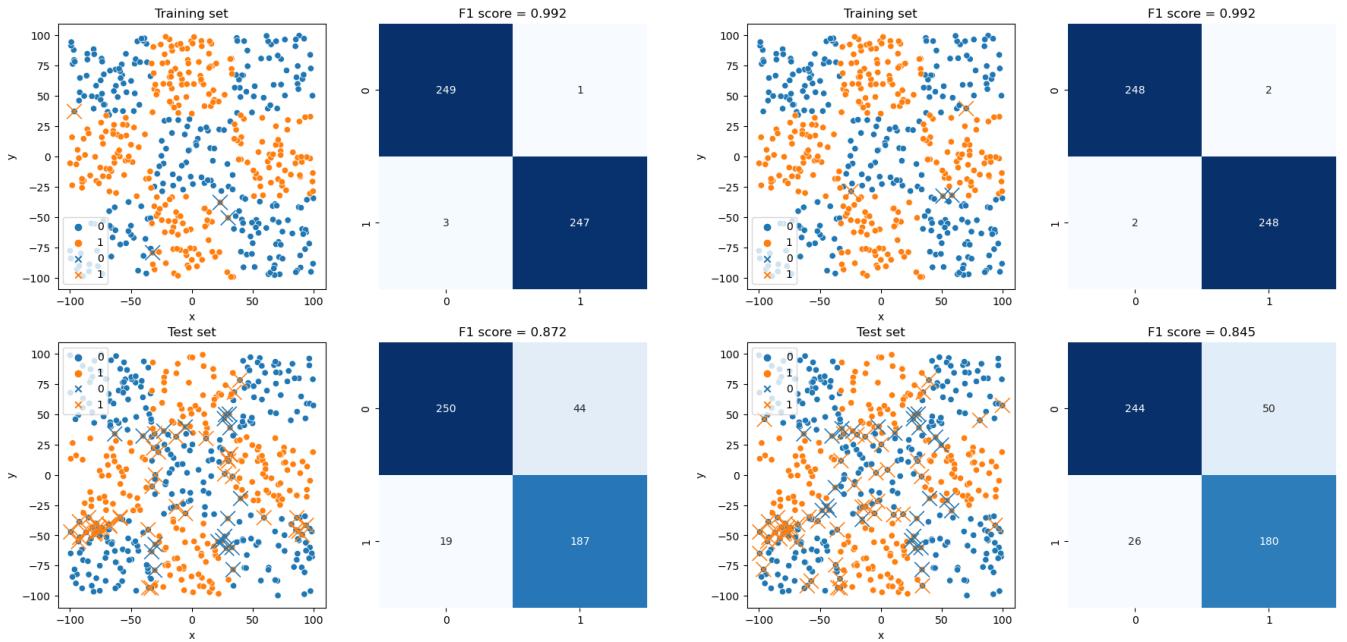
6 Różne funkcje aktywacji na warstwach ukrytych

Do tej pory sieć zawsze używała funkcji sigmoid na warstwach ukrytych. W tym kroku sprawdziłem, czy inne funkcje mogą okazać się lepsze. Funkcje wykorzystywane w tym porównaniu to:

- sigmoid (wykorzystywana do tej pory)
- ReLU (Leaky ReLU w problemach klasyfikacji, bo lepiej działała)
- tanh (tangens hiperboliczny)
- liniowa

6.1 Test działania

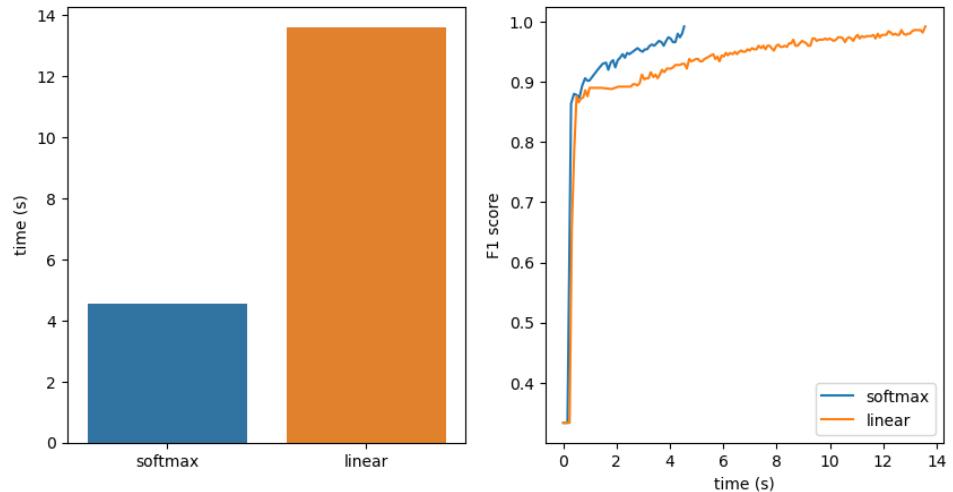
Funkcje aktywacji były porównywane na dwóch zbiorach z problemem regresji i na dwóch zbiorach z problemem klasyfikacji. Ponieważ niektóre funkcje spisują się lepiej gdy sieć ma więcej warstw, porównywałem sieci o 1, 2 i 3



(a) funkcja aktywacji warstwy wyjściowej - softmax

(b) funkcja aktywacji warstwy wyjściowej - liniowa

Comparison for different output activation functions



(c) porównanie wyników w zależności od funkcji aktywacji funkcji wyjściowej

Rysunek 12: Uczenie sieci do zbioru xor3

warstwach ukrytych. Eksperyment porównujący funkcje aktywacji składał się z następujących kroków oddzielnie dla każdego zbioru, każdej funkcji, i każdej architektury:

- znalezienie odpowiednich parametrów dla Adam i learning rate w celu sprawiedliwego porównania (mają one bardzo duże znaczenie dla wydajności sieci)
- uczenie sieci do danego zbioru przy wcześniej znalezionych optymalnych parametrach, dla każdej z architektur po 5 razy
- wybranie najlepszych wyników i porównanie ich

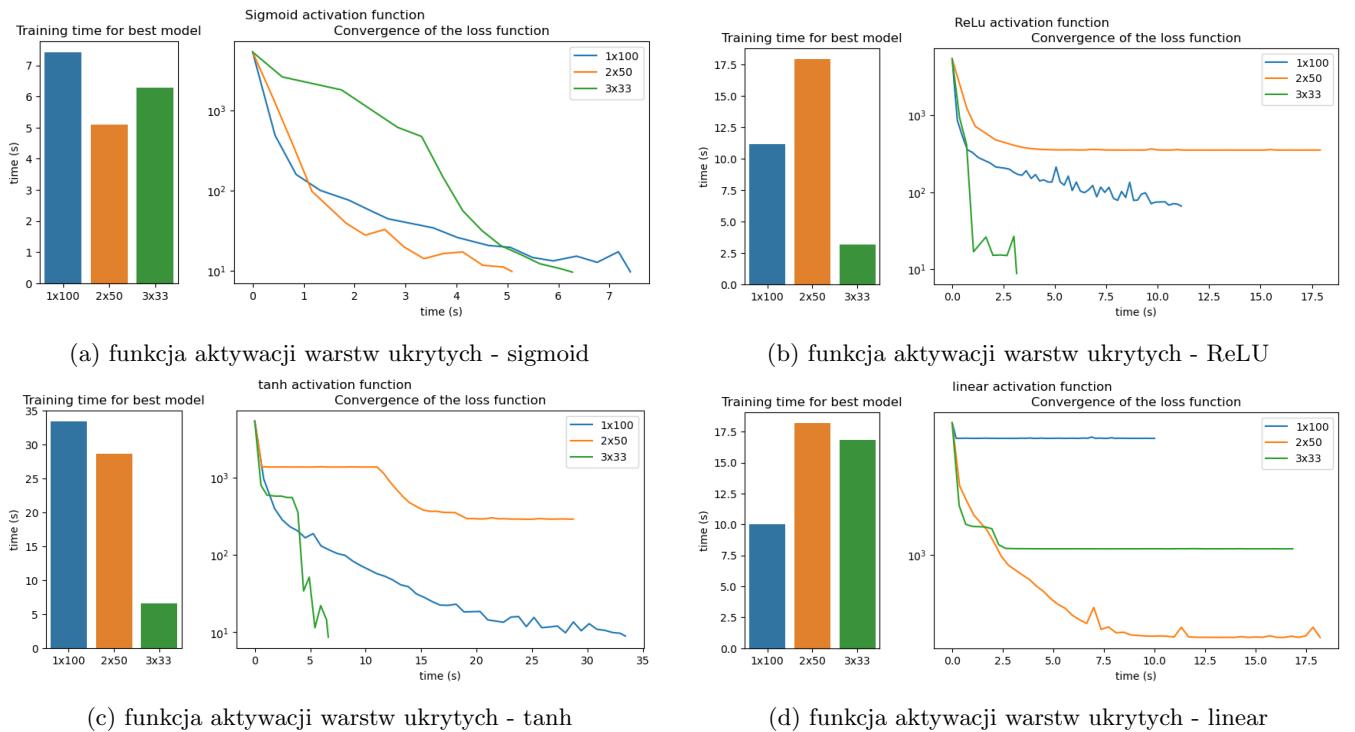
Sprawdzałem wszystkie funkcje tylko na pierwszym zbiorze. Dla następnych zbiorów wybrałem dwie funkcje, które wydawały mi się działać najlepiej i skupiłem się tylko na nich.

6.1.1 multimodal-large

Ten zbiór był już omawiany wcześniej. Na rysunku 13 przedstawione są wykresy zbieżności najlepszych sieci dla poszczególnych funkcji aktywacji. Na wykresach widać:

- (a) Funkcja sigmoid potrafiła sobie dobrze poradzić na wszystkich 3 architekturach, przy czym dla architektury 2x50 osiągnęła cel najszybciej.

- (b) Funkcja ReLU radziła sobie najlepiej dla większej ilości warstw i osiągnęła cel bardzo szybko, w pozostałych dwóch architekturach nie udało się osiągnąć celu w określonej maksymalnej liczbie epoch.
- (c) Funkcja tanh szybko osiągnęła cel przy architekturze 3x33, a w architekturze 1x100 też się to udało, ale wolniej.
- (d) Funkcja liniowa poradziła sobie tylko dla architektury 2x50, ale zajęło jej to dosyć długo.



Rysunek 13: Uczenie sieci do zbioru multimodal-large w zależności od funkcji aktywacji na warstwach ukrytych

W tabeli 4 przedstawione są wyniki benchmarku, z których wybieralem najlepsze modele do wykresów. Tutaj można zauważać, że najbardziej niezawodne (w sensie, zwykle osiągają dobre wyniki w szybkim czasie) były architektury sigmoid 1x100 i ReLU 3x33. Ponieważ wygląda na to, że sigmoid i ReLU działały najlepiej, w następnych zbiorach porównywałem tylko te funkcje.

multimodal-large						
1 x 100		2 x 50		3 x 33		
	time	loss	time	loss	time	loss
sigmoid	8.42	9.89	7.22	1429.41	6.27	9.70
	7.41	9.78	7.90	294.13	4.47	323.91
	6.86	14.14	7.06	1085.30	5.36	1013.12
	7.32	15.06	6.24	291.45	5.43	15.20
	10.97	17.62	5.09	9.89	11.76	1333.35
ReLU	14.83	72.37	20.10	353.20	3.81	8.80
	21.03	105.57	17.90	351.12	3.51	8.30
	13.07	170.37	15.89	372.15	3.14	8.79
	17.87	160.25	15.63	349.91	3.05	17.69
	11.14	65.87	15.78	780.68	3.45	21.80
tanh	33.41	8.97	28.70	289.46	9.23	8.96
	32.72	15.09	26.97	356.06	9.13	562.63
	45.47	12.51	28.63	893.85	6.63	8.67
	47.24	12.68	26.57	291.81	6.70	1092.41
	439.51	16.08	27.41	356.55	7.16	1097.68
linear	10.11	4405.78	18.29	91.03	16.56	287.86
	10.41	4398.41	18.18	351.58	16.57	548.34
	10.27	4405.34	19.11	891.89	16.82	1083.16
	9.96	4398.93	18.47	830.88	16.75	359.39
	9.99	4398.49	18.42	351.39	4.76	8.79

steps-large						
1 x 100		2 x 50		3 x 33		
	time	loss	time	loss	time	loss
sigmoid	35.80	31.55	22.77	12.86	24.18	33.32
	23.57	33.57	22.47	11.23	24.61	16.93
	24.12	32.59	25.39	12.26	22.24	32.48
	22.09	32.70	22.62	14.32	22.78	16.70
	23.02	36.06	21.70	21.37	26.65	16.03
ReLU	12.87	38.81	19.30	16.46	19.44	21.87
	13.10	37.87	18.28	25.52	20.13	7.24
	12.62	55.50	17.49	14.91	22.29	8.40
	11.58	60.53	18.16	15.45	22.28	7.24
	12.25	42.40	17.66	30.25	19.85	6.87

Tabela 4: Wyniki benchmarku dla regresji na zbiorach multimodal-large i steps-large

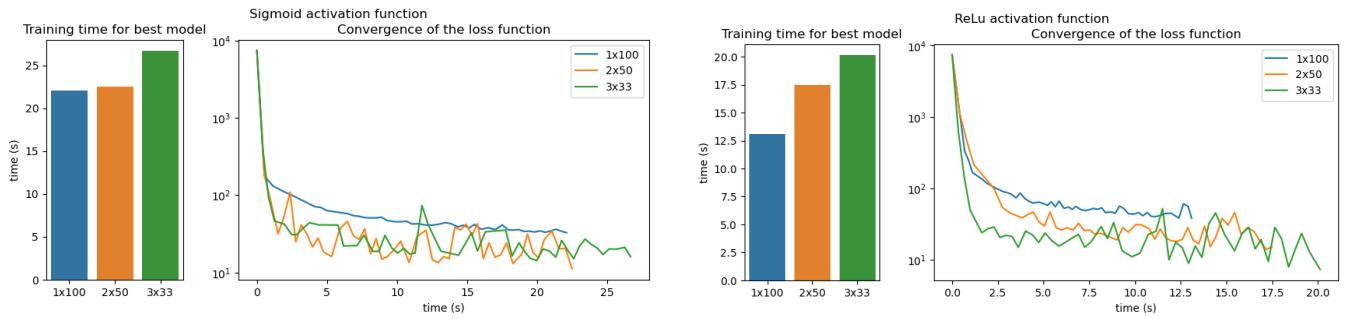
6.1.2 steps-large

Na rysunku 14 widać jak sieci poradziły sobie ze zbiorom. Na wykresach:

- (a) Sigmoid najlepiej zadziałał dla architektury 2x50, ale architektura 3x33 też osiągnęła cel
- (a) ReLU znowu najlepiej zadziałało przy architekturze 3x33.

Ogólnie można zauważyć z wykresów, że ReLU dla każdej architektury było trochę szybsze od sigmoida i osiągało podobne wyniki funkcji straty.

Z tabeli 4 można wyczytać, że ReLU dla każdej architektury za każdym razem okazał się szybszy niż sigmoid. Sigmoid zwykle osiągał lepsze MSE na architekturze 1x100 i 2x50, a ReLU dużo lepiej radziło sobie na architekturze 3x33.



(a) funkcja aktywacji warstw ukrytych - sigmoid

(b) funkcja aktywacji warstw ukrytych - ReLU

Rysunek 14: Uczenie sieci do zbioru steps-large w zależności od funkcji aktywacji na warstwach ukrytych

rings5-regular							
1 x 150		2 x 75		3 x 50			
	time	loss	time	loss	time	loss	
sigmoid	54.65	0.78	1.13	0.96	1.18	0.96	
	45.48	0.80	1.40	0.90	1.04	0.95	
	46.94	0.77	1.91	0.92	0.70	0.92	
	46.11	0.81	0.94	0.92	0.78	0.91	
	43.38	0.80	1.20	0.92	0.83	0.91	
ReLU	-	-	7.81	0.96	6.66	0.93	
	-	-	7.36	0.90	4.14	0.96	
	-	-	7.32	0.92	3.42	0.92	
	-	-	7.96	0.91	4.63	0.91	
	-	-	6.54	0.96	3.91	0.88	
	-	-	-	-	-	-	

rings3-regular							
2 x 75		3 x 50					
	time	loss	time	loss			
sigmoid	2.53	0.96	1.56	0.96			
	2.29	0.95	1.17	0.94			
	2.25	0.97	1.32	0.95			
	2.34	0.95	1.66	0.96			
	2.19	0.92	0.88	0.92			
	7.41	0.96	6.03	0.93			
ReLU	7.24	0.91	6.73	0.95			
	6.80	0.86	4.72	0.95			
	6.21	0.91	4.72	0.93			
	6.05	0.85	4.74	0.90			
	-	-	-	-			
	-	-	-	-			

Tabela 5: Wyniki benchmarku dla klasyfikacji na zbiorach rings5-regular i rings3-regular

6.1.3 rings5-regular

Zbiór podobny do rings3-regular, tylko zamiast 3 klas ma 5. Z wykresu na rysunku 15 można wyczytać, że sieci z funkcją sigmoid radziły sobie lepiej niż te z ReLU. Mimo, że na wykresie widać jakby ReLU potrzebowało dużo więcej czasu to nadal to było 4-6 sekund, więc wynik bardzo dobry.

W tabeli 5 pokazuję, dlaczego nie porównywałem architektury 1x150. Jest tak ponieważ dla obu funkcji F1 score zatrzymuje się w okolicy 0.8 i nie rusza się, nie ważne jakie są parametry. Podobna sytuacja zachodzi dla kolejnego zbioru rings3-regular. Informacje z tabeli pokazują, że obie funkcje dla architektur 2x75 i 3x50 dawały zawsze akceptowalne wyniki 0.8 w krótkim czasie. Sigmoid był zawsze znaczco szybszy.

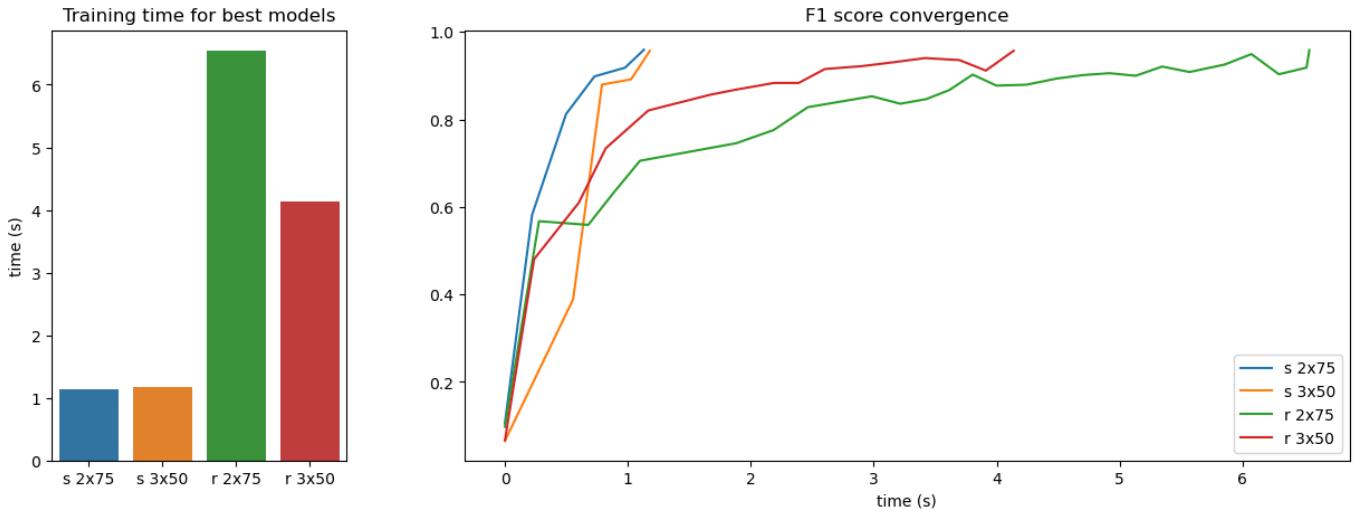
6.1.4 rings3-regular

Tutaj sytuacja jest prawie identyczna. Na rysunku 16 widać, że znowu sigmoid poradził sobie lepiej. Takie same wnioski widoczne w tabeli 5.

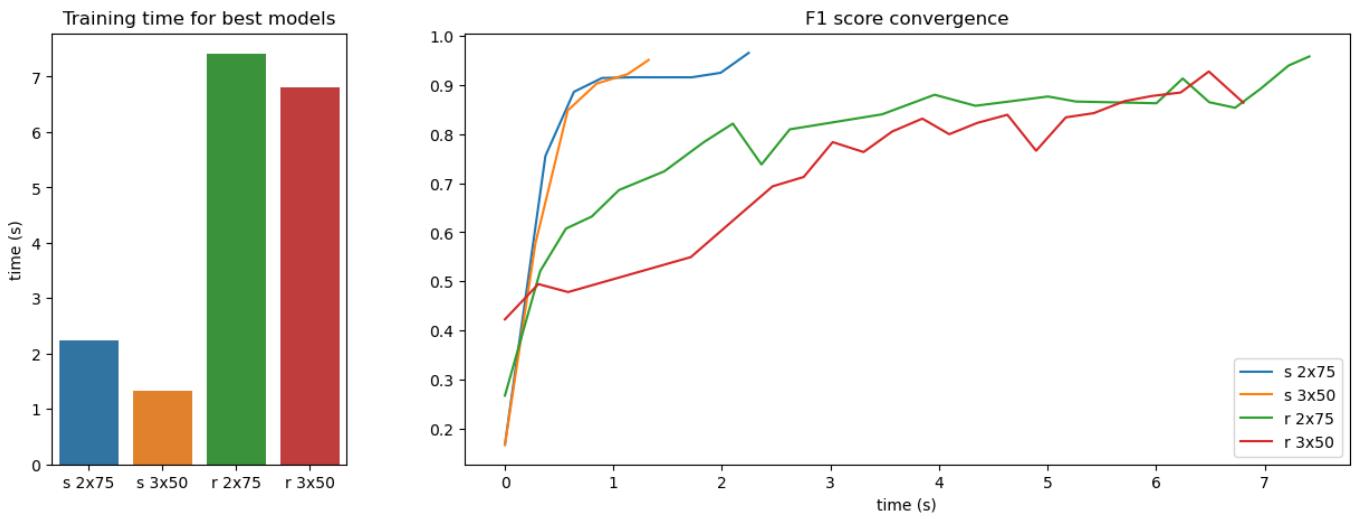
6.2 Wnioski

Wnioski z tego kroku są następujące:

- Dla różnych funkcji aktywacji na warstwach ukrytych ważne jest odpowiednie dobranie liczby warstw.



Rysunek 15: Uczenie sieci do zbioru rings5-regular w zależności od funkcji aktywacji na warstwach ukrytych



Rysunek 16: Uczenie sieci do zbioru rings3-regular w zależności od funkcji aktywacji na warstwach ukrytych

- Niektóre funkcje aktywacji wcale nie potrafią się nauczyć jeśli tych warstw jest za mało.
- Może to być też zależne od zbioru. Sigmoid nie miał problemu z uczeniem się na różnej ilości warstw dla obu problemów regresji, jednak dla klasyfikacji nie udawało mu się na jednej warstwie.
- W zbiorach wykorzystanych do testów wydaje się, że sigmoid działał najlepiej, ale ReLU zwykle też dawało dobre wyniki.

7 Przeuczenie i regularyzacja

W tym kroku do implementacji sieci dodałem mechanizm regularyzacji wag za pomocą kary L2, oraz możliwość rozbicia zbioru treningowego na dodatkowy zbiór walidacyjny. Podczas uczenia sieć nie ma dostępu do zbioru walidacyjnego. Jeśli funkcja straty maleje dla zbioru treningowego, a rośnie dla zbioru walidacyjnego to sieć zatrzymuje uczenie w celu uniknięcia przeuczenia. Najlepsza sieć to ta która osiągnęła najmniejszy wynik straty dla zbioru walidacyjnego.

7.1 Test działania

W celu sprawdzania, czy nowe implementacje lepiej przeciwdziałyły przeuczeniu porównywałem na zbiorach 4 ustawienia:

1. bazowa sieć
2. sieć z regularyzacją wag
3. sieć z zatrzymywaniem uczenia w zależności od straty na zbiorze walidacyjnym

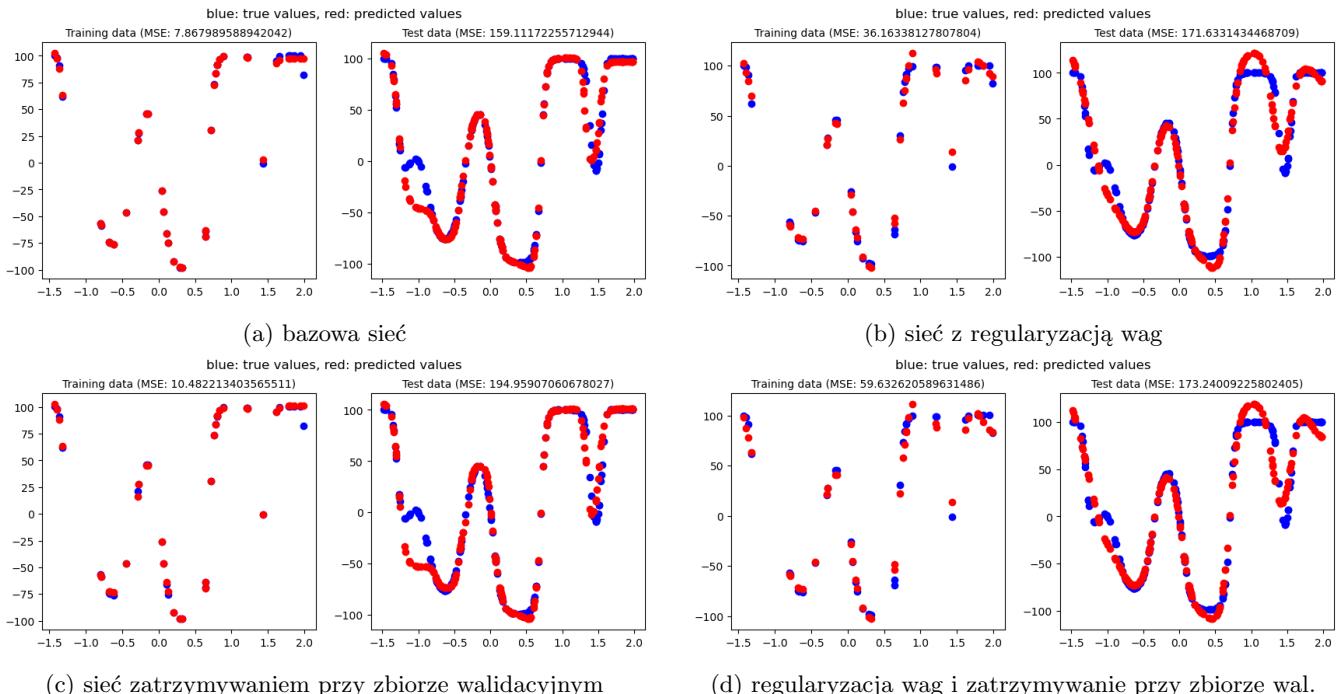
4. sieć z regularizacją wag i zatrzymaniem uczenia w zależności od straty na zbiorze walidacyjnym

Sieć przetestowałem raz na problemie regresji i 3 razy na problemie klasyfikacji, przy czym dwa zbiorów są *sparse* (rzadkie - w zbiorze treningowym jest bardzo mało obserwacji), a dwa zbiorów klasyfikacyjne są *balanced* (wartości z jednej klasy są bardzo rzadkie w porównaniu do reszty klas). Zaprezentowane wyniki nie były pierwszymi jakie otrzymałem. Uczyłem sieci wielokrotnie i wybierałem te, w których wynik wychodził najlepszy. Do zbioru walidacyjnego przydzielałem losowo 20% obserwacji ze zbioru treningowego.

7.1.1 multimodal-large-sparse

Zbior ma wartości z tej samej funkcji co zbiorów multimodal do tej pory. Różni się tym, że w zbiorze treningowym jest tylko 40 obserwacji. Na rysunku 17 widać kolejne wyniki uczenia sieci:

- (a) Dla zbioru treningowego bazowa sieć dopasowała się prawie idealnie. Na zbiorze testowym widać lepiej jak takie dopasowanie wyglądało, było prawie perfekcyjne. Widać z lewej strony wykresu testowego niedokładność związaną z brakiem danych. Najlepszy wynik MSE na zbiorze treningowym, ale widać przeuczenie w jednym miejscu.
- (a) Na wykresie zbioru treningowego widać, że MSE było dosyć wysokie, a punkty nie były tak dokładnie dopasowane (uczenie miało tyle samo epoch co bazowa sieć). Na zbiorze testowym widać ogólniejsze i lepsze dopasowanie w lewej części, ale gorsze w niektórych ostrych zagęściach.
- (a) Tu znowu niskie MSE na zbiorze treningowym związane z dokładnym dopasowaniem sieci do punktów, za wyjątkiem tych w zbiorze walidacyjnym. Wyniki uczenia takiej sieci w dużym stopniu zależy od wyboru obserwacji do zbioru walidacyjnego (szczególnie na takim małym zbiorze). Tutaj akurat losowo wybrały się obserwacje, które nie są wpływowe (mają bliskich sąsiadów więc nie mają tak dużego znaczenia). Na zbiorze testowym niewiele zmian w porównaniu do bazowej implementacji, lecz wynik trochę gorszy. Udało się go jednak otrzymać w znacząco mniejszej liczbie epoch.
- (a) Na zbiorze treningowym widać połączenie niedokładności trenowania z punktów (b) i (c). Na zbiorze testowym widać podobne dopasowanie co na wykresie (b). Ze względu na zatrzymanie przy braku polepszenia MSE dla walidacji proces uczenia był szybki, a wyniki zadowalające.

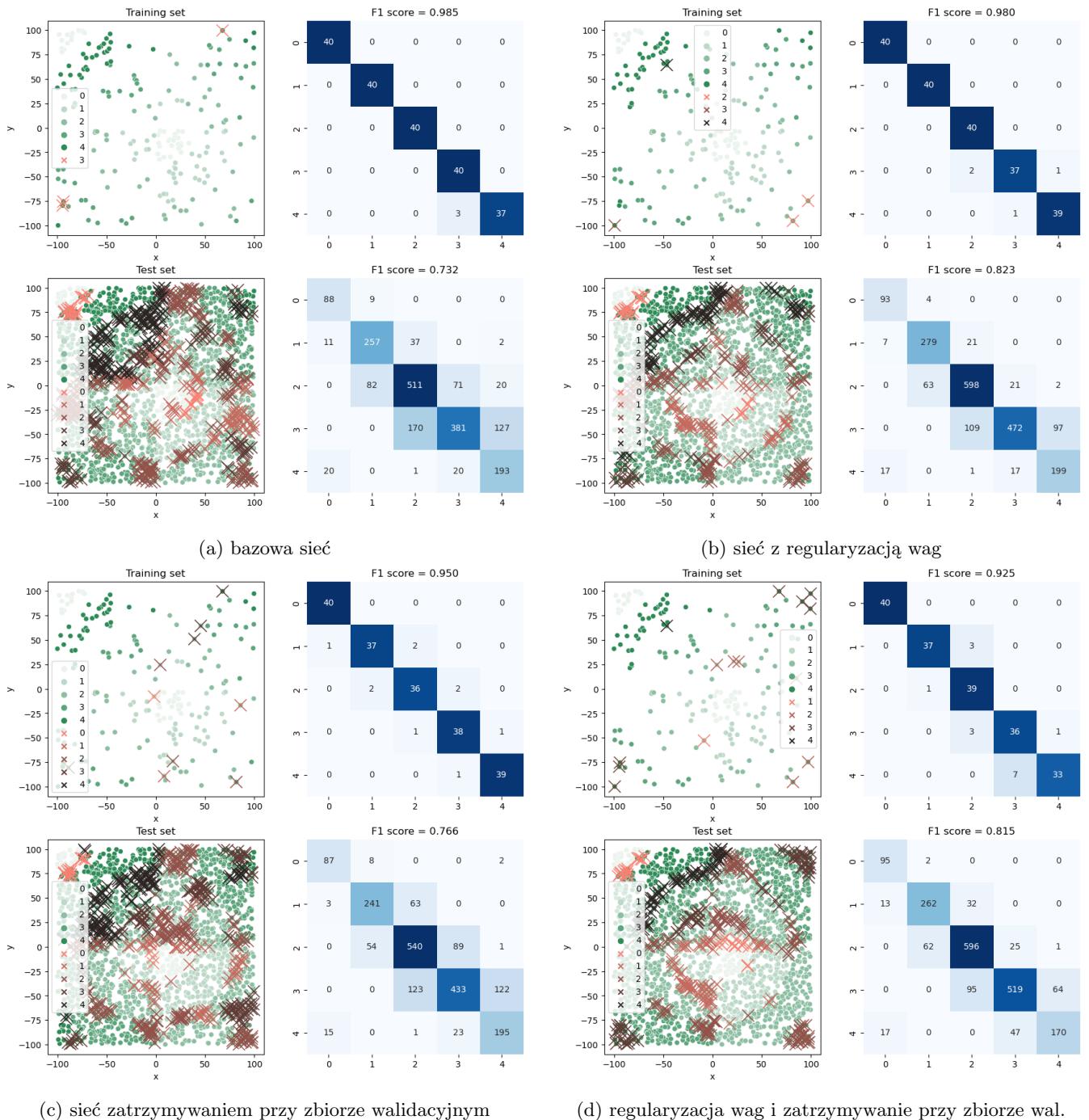


Rysunek 17: Uczenie sieci do zbioru multimodal-sparse w zależności od wyboru sposobu przeciwdziałania przeuczeniu

7.1.2 rings5-sparse

W tym zbiorze sytuacja jest podobna do poprzedniego. W zbiorze treningowym tym razem jest 200 obserwacji, przy czym są one podzielone po równo ze względu na klasy. Na rysunku 18 widać wyniki uczenia na zbiorze w zależności od wyboru sposobu przeciwdziałania przeuczeniu:

- (a) Bardzo dokładane dopasowanie do zbioru treningowego, jednak najgorszy wynik F1 na zbiorze testowym. Błędy widać przed wszystkim w lewym górnym rogu.
- (a) Równie dokładne dopasowanie do zbioru treningowego, ale tym razem trochę lepszy wynik F1 dla zbioru testowego.
- (a) Lekko gorsze dopasowanie do zbioru treningowego ze względu na stop braku polepszenia wyniku dla zbioru walidacyjnego. Ogólnie wynik porównywalny do wykresu (a), ale uczenie trwało znaczco krócej.
- (a) Bardziej ogólnie dopasowanie do zbioru treningowego przy najmniejszym do tej pory wyniku F1 (prawie 10% mniejszy od wyniku dla sieci bazowej). Wyniki dla zbioru testowego dobre, podobne do wykresu (b). Rozwiązań dało dobry wynik w krótkim czasie.

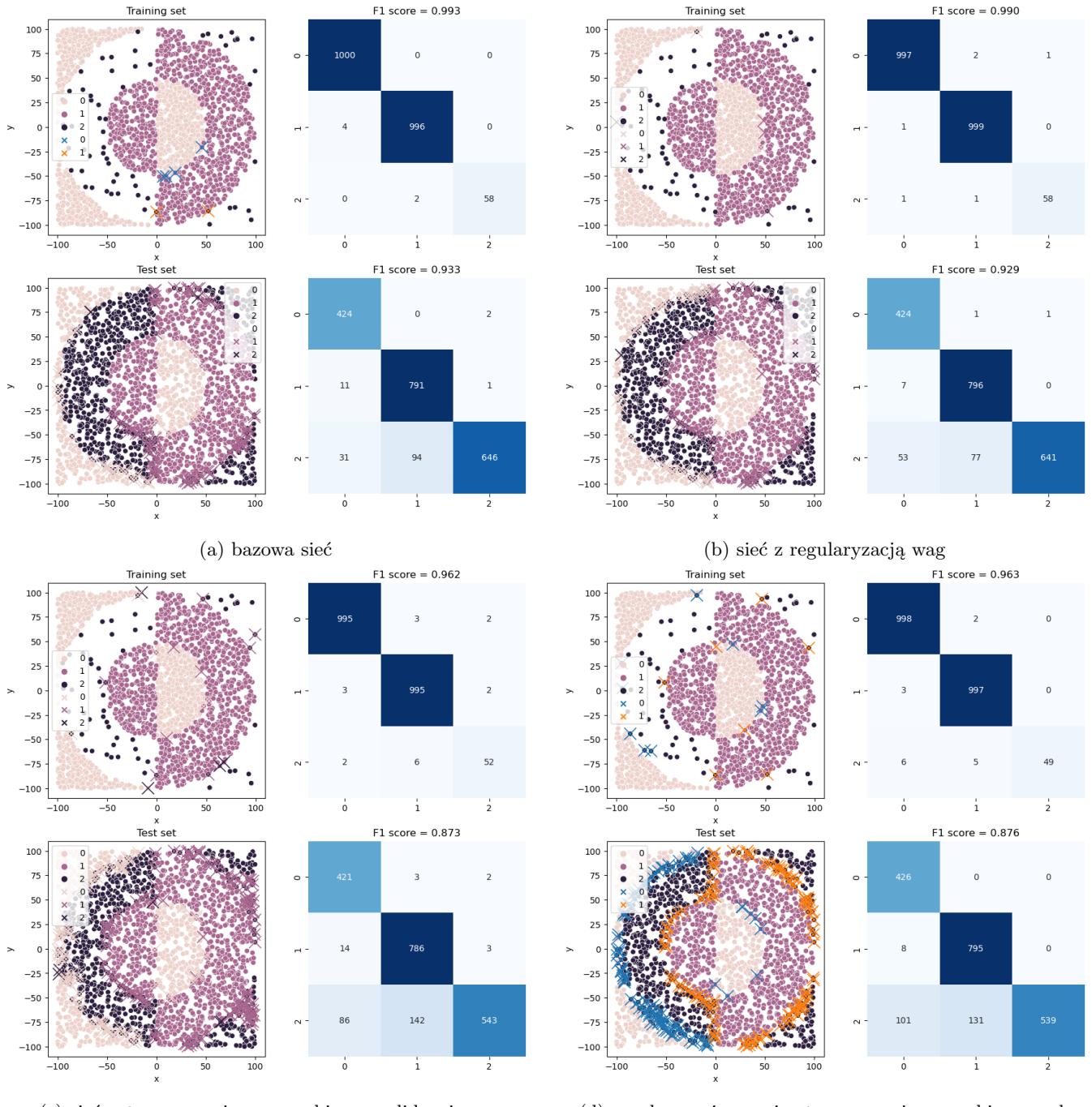


Rysunek 18: Uczenie sieci do zbioru rings5-sparse w zależności od wyboru sposobu przeciwdziałania przeuczeniu

7.1.3 rings3-balance

Ten zbiór ma już więcej obserwacji w zbiorze treningowym (ponad 2000), ale nie są one równomiernie rozłożone. Jest po 1000 obserwacji dla dwóch klas, a dla trzeciej jest tylko 60. Wyniki uczenia na tym zbiorze są na rysunku 19:

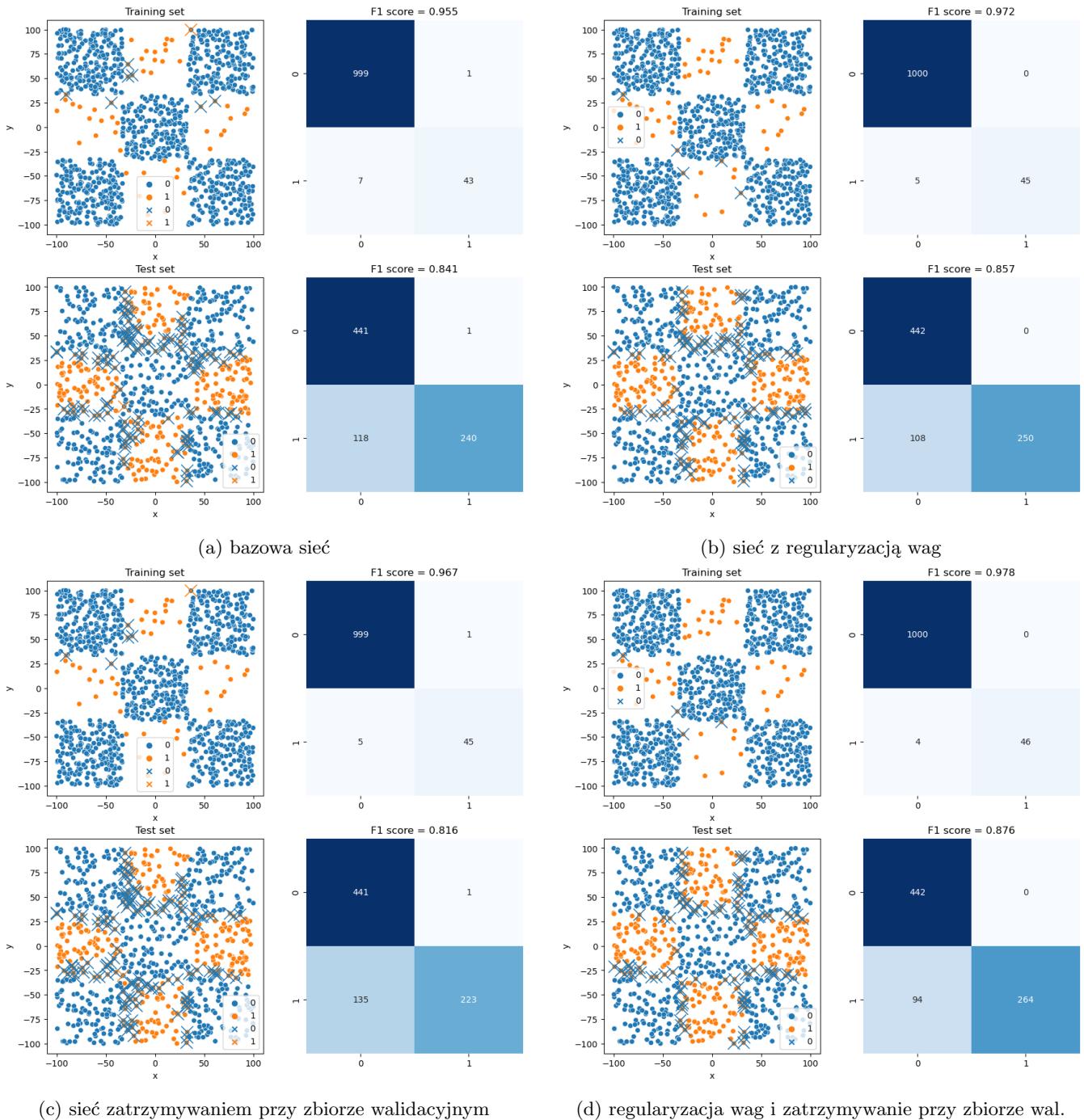
- Prawie idealne dopasowanie na zbiorze treningowym. Na zbiorze testowym wyjątkowo dobry wynik F1. Nie udało mi się osiągnąć lepszego stosując pozostałe sposoby przeciwdziałania przeuczeniu. Nie udało mi się też zbliżyć do takiego wyniku dla próbując na tych samych parametrach dla bazowej implementacji. Zwykle F1 było w okolicach 0.85.
- Równie silne dopasowanie do zbioru treningowego, a wynik F1 porównywalnie dobry do bazowej sieci.
- Gorszy wynik F1 na zbiorze treningowym i testowym. Tutaj wydaje mi się, że bardzo duże znaczenie miał znowu wybór obserwacji do zbioru walidacyjnego. Widać, że klika obserwacji z klasy 3, które znajdowały się w prawym dolnym rogu nie zostało dobrze zlabelowane na zbiorze treningowym, co zaskutkowało wielu błędem w tym obszarze.
- Podobne wyniki co na wykresie (c). Niezależnie jak zmieniałem parametry sieć nie potrafiła osiągnąć lepszych wyników niż warianty (a) i (b)



Rysunek 19: Uczenie sieci do zbioru rings3-balance w zależności od wyboru sposobu przeciwdziałania przeuczeniu

7.1.4 xor3-balanced

Na tym zbiorze przeprowadzona została klasyfikacja binarna, przy czym obserwacji z jednej klasu było 1000, a z drugiej 50. Ogólnie wyniki (Rysunek 20) różnią się od siebie mniej w porównaniu do poprzednich zbiorów. Najlepszą siecią okazała się ta z wykresu (d). Dała ona najlepsze wyniki F1, przy małej ilości epoch.



(c) sieć zatrzymywaniem przy zbiorze walidacyjnym

(d) regularizacja wag i zatrzymywanie przy zbiorze wal.

Rysunek 20: Uczenie sieci do zbioru xor3-balance w zależności od wyboru sposobu przeciwdziałania przeuczeniu

7.2 Wnioski

Wnioski z tego kroku są następujące:

- Często sposoby przeciwdziałania przeuczeniu dawały lepsze rezultaty poprzez albo lepsze wyniki na zbiorach testowych ze względu na pewną generalizację, albo znaczące zmniejszenie czasu uczenia przy równie dobrych wynikach.
- Regularizacja zwykle potrafiła generalizować dopasowanie, przez co zapobiegała przeuczeniu. Dobrze widać to na rysunku 17, gdzie output sieci wygląda bardziej jak łagodniejszy wielomian, niż ostre zmiany przy bazowej implementacji

- Z eksperymentu wychodzi, że zatrzymanie uczenia przy braku polepszenia funkcji straty dla zbioru walidacyjnego jest pozytywne głównie ze względu na polepszenie czasu uczenia sieci. To rozwiązanie wykorzystywane samotnie miało jednak duże minusy. Sieć traci dużą część obserwacji, na których mogłaby się uczyć, przez co wybór tych punktów, ma bardzo duże znaczenie. Oprócz tego zwykle otrzymywałem podobne, lub gorsze wyniki od podstawowej sieci, jeśli chodzi o ocenę na zbiorze testowym.
- Połączenie regularyzacji wag i zatrzymania w zależności od zbioru walidacyjnego często dawało najlepsze wyniki. Takie podejście łączyło generealizację dopasowania i szybkość uczenia.

8 Podsumowanie

W ramach tego projektu została zaimplementowana sieć neuronowa typu MLP. Sieć ta składa się z wielu warstw neuronów, które są połączone ze sobą za pomocą wag. Wagi te są modyfikowane w trakcie procesu uczenia, aby sieć mogła nauczyć się odpowiednich wzorców i przewidywać wyniki dla nowych danych. W trakcie pracy nad implementacją sieci zostały dodane różne funkcjonalności polepszające szybkość uczynienia i jakość wyników sieci. Praca z użyciem sieci na prostych zbiorach pozwoliła na wyciągnięcie cennych wniosków dotyczących wyboru architektury sieci, funkcji aktywacji, sposobów polepszenia zbieżności procesu uczenia oraz przeciwdziałania przeuczeniu. Ostatecznie wybory mocno zależą od danych, na których trenujemy sieć, jednak często w eksperymentach pojawiali się faworyci.