Sprawozdanie z realizacji zadania

Implementacja sieci neuronowych od zera

wersja 2

Tymoteusz Kwieciński

320637

Czerwiec 2024

1 Cel zadania

Głównym celem tego zadania było zapoznanie się z budową modelu perceptrony wielowarstwowego oraz jego implementacja bez użycia przeznaczonych do tego bibliotek. Zadanie zostało podzielone na 6 części, które stopniowo pozwalały wgłębić się w szczegóły działania sieci oraz usprawnić działanie zaimplementowanego modelu. Początkowo testowaliśmy działanie sieci w celu poznania struktury oraz działania funkcji *forwad pass*, następnie należało zaimplementować mechanizm uczenia sieci - propagację wsteczną. Następnie dla działającej sieci dokładaliśmy dodatkowe elementy usprawniające jej wyniki, takie jak algorytmy usprawniające działanie algorytmu *gradient descent*, dostosowanie architektury sieci do zadania klasyfikacji, a także kontrolowanie zjawiska przeuczenia przez regularyzacje i kontrole wartości *loss* na zbiorze walidacyjnym w trakcie treningu.

2 Realizacja działań i wyniki eksperymentów

2.1 Laboratorium 1

2.1.1 Opis zadania

Zadania w tych laboratoriach polegało na zbudowaniu bazowej struktury sieci neuronowej MLP oraz ręcznym dostosowaniu jej wag. Działanie sieci mieliśmy przetestować na dwóch zbiorach danych - prostym zbiorze, który reprezentuje funkcję kwadratową *square simple* oraz zbiorze który reprezentuje funkcję schodkową - *steps large*.

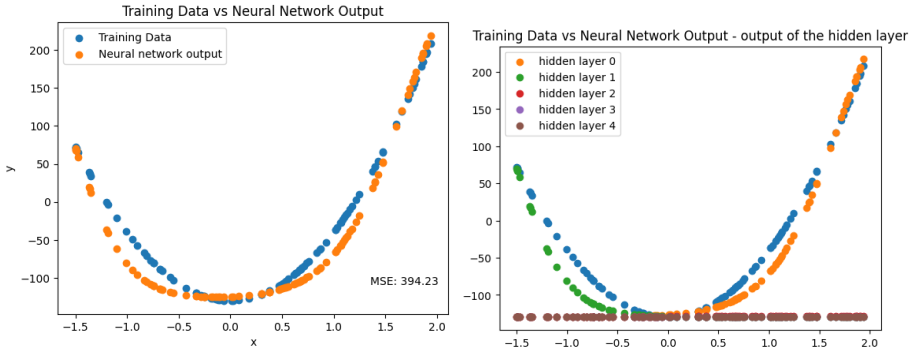
2.1.2 Wyniki eksperymentów

Początkowo wagi dobierałem ręcznie, próbując *trafić* w odpowiednie wagi. Jednak dokładne dopasowanie wag modelu było możliwe jedynie dzięki odpowiedniej wizualizacji wyników warstw ukrytych sieci. Wyniki kolejnych neuronów sieci umieściłem na wykresie, a następnie końcową funkcję próbowałem *złożyć* jako suma tych funkcji. Ostatecznie dzięki odpowiedniemu dopasowania parametrów testowanych sieci udało uzyskać mi się odpowiednią jakość modelów.

Poniżej znajduje się tabela 1 z wynikami sieci neuronowej z ręcznie dostoswanymi wagami dla zbioru testowego *square-simple*.

W tabeli 2 poniżej znajdują się wyniki sieci na zbiorze *steps-large*.

Podobne wyniki przy jednej warstwie dla różnych liczb neuronów wynika z tego, że dla ułatwienia ręcznego dobierania wag, większość neuronów miała wyzerowane wagi.



Rysunek 1: Próby ręcznego dostosowania wag sieci ręcznie

l. warstw ukrytych	l. neuronów w warstwie	treningowe MSE	test MSE
1	5	1.186	1.376
1	10	1.186	1.376
2	5	0.680	0.844

Tabela 1: Wyniki sieci neuronowej

2.1.3 Wnioski

Dobieranie ręczne wag sieci jest możliwe, ale zdecydowanie bardzo czasochłonne. O ile eksperyment pozwolił mi na lepsze zrozumienie w jaki sposób sieć manipuluje funkcjami aktywacji, to nie warto korzystać z tej naiwnej metody *trenowania* sieci neuronowych.

2.2 Laboratorium 2

2.2.1 Opis zadania

Celem drugiego laboratorium było zaimplementowanie mechanizmu uczenia w sieci neuronowej zwanej propagacją wsteczną. Mechanizm ten pozwala w łatwy i szybki sposób aktualizować wagi sieci, a co za tym idzie, w lepszy sposób przybliżać skomplikowane funkcje. Propagacja wsteczna opiera się na łańcuchowym wyliczaniu pochodnych (ang. *chain rule*), które mogą być wykorzystane w gradientowych algorytmach optymalizacyjnych, takich jak *gradient descent* oraz jego warianty. [2]

Dodatkowo, w ramach tego laboratorium należało zaimplementować inicjalizację wag sieci i sprawdzić działanie napisanego kodu na kilku prostych przykładach.

l. warstw ukrytych	l. neuronów w warstwie	treningowe MSE	test MSE
1	5	2.090	2.756
1	10	2.090	2.756
2	5	0.598	0.578

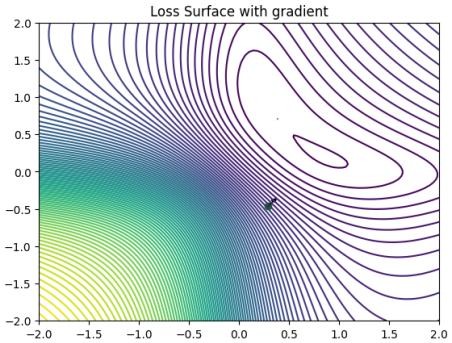
Tabela 2: Wyniki sieci neuronowej dla zbioru *steps-large*

2.2.2 Podjęte kroki i wyniki eksperymentów

To ćwiczenie okazało się dość czasochłonne, ze względu na konieczność debugowania działania sieci. Ostatecznie jednak wspierając się literaturą udało się zaimplementować propagację wsteczną.

Istotne dla wyników sieci okazało się odpowiednie inicjalizowanie wag sieci, w związku z tym zaimplementowałem metody *Xavier* oraz *He*, a także prostą inicjalizację wag z rozkładu standardowego normalnego i jednostajnego na przedziale $[0, 1]$. Inicjalizacja wag sieci z rozkładu jednostajnego na przedziale $[0, 1]$ okazała się negatywnie wpływać na to, jak szybko uczyła się sieć, a czasami nawet doprowadzała do zatrzymywania się procesu treningu w nieoptymalnym lokalnym minimum. W związku z tym we wszystkich dalszych eksperymentach wykorzystywałem dla warstw ukrytych funkcję inicjalizującą *He*, zaś dla ostatnich warstw sieci - liniowych oraz softmax - losowy wybór wag z rozkładu normalnego standardowego. Zaimplementowany algorytm gradient descent miał możliwość podziału na batche, o ile nie napisano inaczej wykorzystywałem dość standardowy rozmiar batchu - 32. [2]

Aby zwizualizować odpowiednie działanie poszukiwania gradientu wag oraz sprawdzić, czy faktycznie udaje się algorytmowi poszukiwać minimum, stworzyłem wykres poziomicowy 2, który ilustruje, że gradient wskazuje minimum lokalne funkcji straty.



Rysunek 2: Wizualizacja funkcji straty sieci z 1 warstwą ukrytą i 2 neuronami

W tabeli 3 poniżej znajdują się wyniki sieci na różnych zbiorach danych. Do zbioru *square-simple* wykorzystałem 2 warstwy po 5 neuronów, do zbioru *steps-small* - 3 warstwy po 5 neuronów, zaś do zbioru *multimodal-large* - ponownie 2

warstwy po 5 neuronów, co okazało się w zupełności wystarczające do uzyskania satysfakcjonujących wyników.

zbiór danych	f. aktywacji	l. warstw ukrytych	test MSE	min test MSE	max test MSE
square-simple	ReLU	2	3.94±0.77	3.24	5.42
steps-small	sigmoid	3	111.53±12.09	96.61	125.06
multimodal-large	ReLU	2	1.67±1.15	3.15	0.21

Tabela 3: Wyniki sieci neuronowej po backpropagacji

Tak duży błąd w przypadku zbioru *steps-small* wynikał z problemy z podziałem zbioru na treningowy i testowy. Jak widać na wykresie 3 w zbiorze treningowym brakowało kilku punktów, które odbiegały od tych w zbiorze treningowym, ale pojawiły się w testowym zbiorze. Być może należałoby zastosować odpowiednią regularyzację, aby zapobiec temu zjawisku, szczególnie, że MSE na zbiorze treningowym każdorazowo było mniejsze niż 1. Jednak z uwagi na bardzo charakterystyczny rodzaj funkcji, proste regularyzacje nie przyniosły oczekiwanych rezultatów i podejmowałem żadnych dalszych kroków w celu polepszeniu wyników sieci na tym zbiorze.

2.2.3 Wnioski

Dzięki temu ćwiczeniu pomyślnie udało mi się zapoznać z działaniem propagacji wstecznej w sieciach neuronowych. Ręczne wyprowadzenie pochodnych w trakcie pracy nad implementacją okazało się bardzo wartościowym elementem ćwiczenia. Co więcej, w związku z problemami z działaniem sieci po poprawnym zaimplementowaniu propagacji, przekonałem się na własnej skórze jak istotne jest poprawne inicjowanie wag sieci. Inicjalizacja wag z rozkładu jednostajnego $[0, 1]$ dla warstw ukrytych okazało się znacznie pogarszać wyniki backpropagacji. Miałem również zapoznać się z naukowymi pracami dotyczącymi inicjalizacji wag - metody *Xavier* [1] oraz *He* [3].

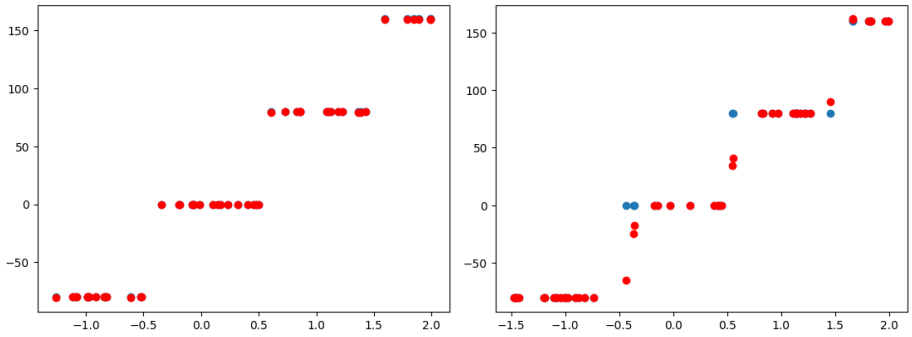
2.3 Laboratorium 3

2.3.1 Opis zadania

W ramach 3 ćwiczenia, mieliśmy za zadanie zaimplementować modyfikacje metody poszukiwania lokalnych minimum funkcji za pomocą gradientu - gradient descent. Nasza sieć miała mieć możliwość wykorzystywania metody RMSprop oraz algorytmu wykorzystującego moment. [6]

2.3.2 Podjęte kroki i wyniki eksperymentów

Poza zaimplementowaniem wyżej wspomnianych algorytmów przejrzałem literaturę i zaimplementowałem również algorytm *Adam*, który okazał się bardzo



Rysunek 3: Wizualizacja działania sieci na zbiorze treningowym i testowym od lewej. Jak widać przewidywania sieci są bardzo dobre. Na niebiesko oznaczone są rzeczywiste wartości, zą na czerwono - predykcje

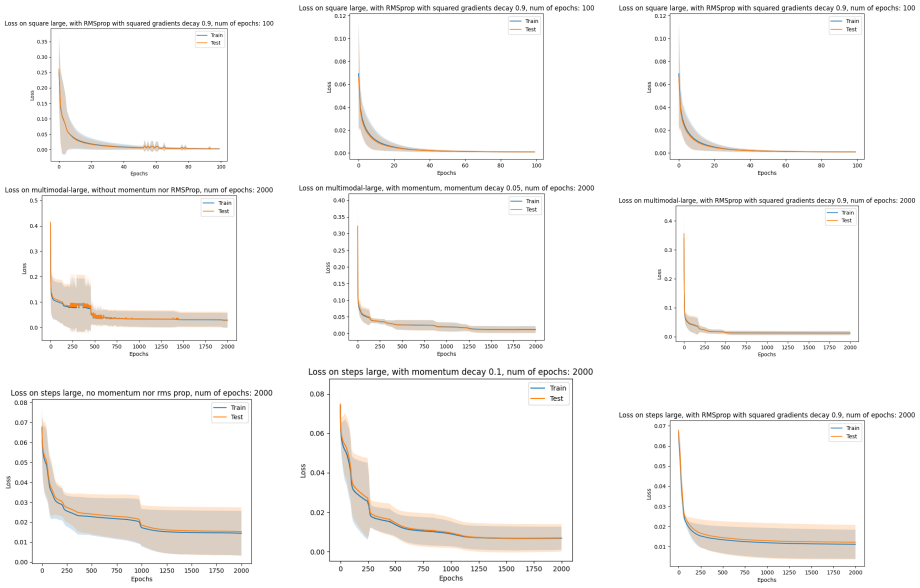
podobny do pozostałych, wymaganych algorytmów. [4] Działanie sieci przetestowałem ponownie na sugerowanych 3 zbiorach danych. W tabeli 4 prezentuje wyniki porównania. Przetestowałem wyniki działania algorytmów bez zmodyfikowanej funkcji gradientu, metody RMSprop z parametrami *squared gradients* równym 0.9 oraz algorytmu wykorzystujego moment z parametrem 0.1.

	test MSE		
zbiór danych	domyślny gradient descent	momentum 0.1	RMSProp 0.9
square-simple	9.32±2.38	7.88±3.54	16.55±9.65
steps-large	103.91±74.29	46.95±42.61	82.85±52.34
multimodal-large	149.97±148.14	56.69±48.68	60.17±32.40

Tabela 4: Wyniki eksperymentu dla różnych rodzajów algorytmu gradient descent

2.3.3 Wnioski

Jak się okazuje, trening przy użyciu wybranych algorytmów optymalizacji przyspiesza proces uczenia sieci neuronowej - zbieżność następuje szybciej i jest dużo bardziej gładka. Niestety, w przypadku rozważanych, małych zbiorów danych, modyfikacje Gradient Descent nie poprawia znacząco wyników, a nawet może je nawet pogorszyć.



Rysunek 4: Porównanie zbieżności dla różnych modyfikacji algorytmu Gradient Descent na różnych zbiorach danych. Widać, że zbieżność algorytmu jest najlepsza dla modyfikacji RMSProp, sieć najwolniej zbiega dla tradycyjnego algorytmu Gradient Descent bez modyfikacji

2.4 Laboratorium 4

2.4.1 Opis zadania

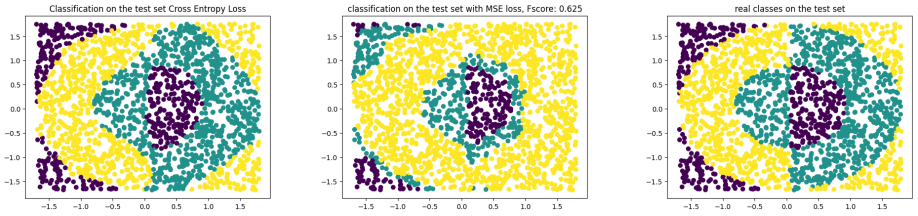
4 zadanie polegało na dostosowaniu sieci do zadania klasyfikacji. Oznaczało to dodanie innej funkcji straty do sieci - *cross-entropy*. Co więcej, aby dostosować sieć do działania w przypadku wielu klas, należało uwzględnić dodatkową funkcję aktywacji dla ostatniej warstwy sieci - softmax. Funkcja ta pozwala przeskalować dowolną liczbę zmiennych, tak aby znajdowały się w przedziale $[0, 1]$ i sumowały się do 1. Można zinterpretować taki wynik jako rozkład prawdopodobieństwa.

2.4.2 Podjęte kroki i wyniki eksperymentów

Początkowo, aby odpowiednio dostosować sieć do zadania klasyfikacji planowałem w zwykły dla backpropagacji sposób wyliczyć pochodną funkcji straty, aby następnie na podstawie niego liczyć gradient po kolejnych warstwach. Okazało się to jednak nietrywialnym zadaniem i po obliczeniach doszedłem do wniosku, że można ominąć ten problem przez pominięcie etapu wyliczania gradientu dla outputu ostatniej warstwy.

Ponownie jak poprzednio działanie sieci, w szczególności nowych elementów zweryfikowałem na zadanych zbiorach danych. Aby zweryfikować, czy w zadaniu klasyfikacji można wykorzystać funkcję straty wykorzystywaną w zadaniu

regresji - MSE przetestowałem wyniki sieci z sigmoidem jako funkcja aktywacji w ostatniej warstwie oraz funkcją starty MSE na zbiorze danych *rings3*. Wizualne porównanie działania sieci można zaobserwować na 5. Wyniki pełnych eksperymentów do zadań klasyfikacji można zobaczyć w tabeli 5.



Rysunek 5: Wyniki klasyfikacji przy różnych funkcjach starty

zbiór danych	l. warstw ukrytych	train Fscore	test Fscore	min test Fscore	max test Fscore
easy	0	0.99±0.00	0.99±0.01	0.98	1
rings3	3	0.91±0.02	0.91±0.02	0.87	0.92
xor3	2	0.99±0.00	0.96±0.02	0.93	0.98

Tabela 5: Wyniki sieci neuronowej dla zadania klasyfikacji

Podobnie jak w przypadku zadania regresji dla zbioru *steps-small*, w zbiorze *xor* również sieć neuronowa sprawdziła się znacznie gorzej na zbiorze testowym w porównaniu do treningowego. Regularyzacja przy pomocy norm L1 oraz L2 ponownie nie poprawiły rezultatów. Aby otrzymać lepsze rezultaty próbowałem również ręcznie zastosowałem metodę analogiczną do *cosine schedule* - rozpoczynając od dużej wartości learning rate kolejno zwiększałem ją i zmniejszałem, to jednak tylko zwiększyło przetrenowanie modelu. Aby zweryfikować, czy dane w zbiorze testowym i treningowym faktycznie są z tego samego rozkładu, złączyłem datasety treningowy i testowy, a następnie ponownie je podzieliłem. Zbiór testowy stanowił teraz 20% złączonego zbioru To pomogło polepszyć wyniki sieci, ostatecznie uzyskując średnio 0.99 i 0.98 Fscore na zbiorze odpowiednio treningowym i testowym. Minimalne uzyskane wartości Fscore uzyskane na tym ponownie podzielonym zbiorze nie były niższe niż 0.98.

2.4.3 Wnioski

Do zadania klasyfikacji potrzebne było zmodyfikowanie architektury sieci. Jak się okazuje nie było to trywialne zadanie, ze względu na konieczność zmiany sposbu wyznaczania pochodnych oraz dodania nowej funkcji straty - *Cross-entropy*, która działa znacznie lepiej dla zadania klasyfikacji, gdyż przy jej zastosowaniu nie jest aż tak widoczne zjawisko saturacji. Ostatecznie sieć sprawuje się znacznie lepiej w zadaniach klasyfikacji, niż gdyby stosować podejście z zadania regresji.

2.5 Laboratorium 5

2.5.1 Opis zadania

W ramach przedostatniego laboratorium naszym zadaniem było przetestowanie różnych architektur sieci, skupiając się na kilku funkcjach aktywacji. Mieliśmy poznać wpływ wyboru funkcji aktywacji na jakość predykcji oraz czasu treningu w zadaniu regresji oraz klasyfikacji.

2.5.2 Podjęte kroki i wyniki eksperymentów

Aby przetestować działanie różnych funkcji aktywacji, wpierw zaimplementowałem je, odpowiednio modyfikując architekturę sieci. Następnie przeprowadziłem proste przeszukiwanie hiperparametrów - grid search liczby warstw oraz liczby neuronów w warstwie. Eksperymenty zostały przeprowadziłem na zbiorze *multimodal-large* Następnie rozważając różne funkcje aktywacji: liniową, hiperboliczny tangens, ReLU oraz sigmoid, przeprowadziłem trening na zbiorze treningowym oraz ewaluacje działania sieci z danymi hiperparametrami na wydzielonym wcześniej ze zbioru treningowego zbioru walidacyjnego. [5]

Zbiór walidacyjny został wydzielony, aby następnie przetestować działanie najlepszych modelów na oddzielnym zbiorze testowym, który nie był znany przez modele i na którym nie odbywało się strojenie hiperparametrów. Zależało mi również na ilościowym wyznaczeniu najlepszych hiperparametrów, w związku z czym chciałem wyznaczyć estymacje MSE, która podlegałaby jak najmniejszemu błędowi.

Ostatecznie najlepsze okazały się największe architektury - 3 warstwy ukryte po 10 neuronów z funkcjami aktywacji relu oraz tangens hiperboliczny, które osiągnęły najmniejsze wartości MSE na zbiorze walidacyjnym - odpowiednio: 41.6±1.35, 7.74±0.65

Jak się okazało, podobnie dobre wyniki modele te otrzymały na zbiorze testowym - odpowiednio: 33.9±1.9, 5,23±0.5. Po wybraniu najlepszych architektur sieci, dalsze treningi były przeprowadzane na złączonym zbiorze treningowym i walidacyjnym.

Tak wybrane dwie architektury porównałem na tym oraz pozostałych zbiorach danych proponowanych w zadaniu, uwzględniając czas treningu, który był wykonywany na pełnym zbiorze treningowym, zaś miary były liczone na zbiorze testowym. Wyniki tego eksperymentu są w tabelach 6 oraz 7.

	ReLU			tanh		
zbiór danych	train MSE	test MSE	czas treningu [s]	train MSE	test MSE	czas treningu[s]
steps-large	22.32±0.10	23.22±0.35	43.1±1.5	14.74±1.63	9.95±0.31	98.84±2.7
multimodal-large	10.69±0.43	7.44±0.35	57.51±3.2	6.19±0.23	1.68±0.04	87±4.1

Tabela 6: Wyniki eksperymentu dla zadania regresji

	ReLU			tanh		
zbiór danych	train Fscore	test Fscore	czas treningu [s]	train Fscore	test Fscore	czas treningu[s]
rings3	0.887±0.04	0.871±0.13	9.1±2.4	0.902±0.02	0.907±0.06	12.3±2.4
rings5	0.855±0.05	0.832±0.17	8.4±1.4	0.92±0.03	0.87±0.03	13.1±3.2

Tabela 7: Wyniki eksperymentu dla zadania klasyfikacji

Wszystkie eksperymenty były przeprowadzane na 1000 epok, wartości learning-rate równej 0.05, 3 warstwach ukrytych po 10 neuronów każda. Poniższe wyniki zostały zaprezentowane uśredniając wyniki po 5 uruchomieniach algorytmu.

2.5.3 Wnioski

Jak się okazuje, funkcjami aktywacji, które dają najlepsze wyniki predykcyjne są relu oraz tangens hiperboliczny, w zadaniach klasyfikacji oraz regresji. Jest to w pewien sposób zaskakujące, z uwagi na to, że historycznie to sigmoid był najczęściej używaną funkcją aktywacji w sieciach neuronowych, okazuje się jednak, że sigmoid lepiej działa jako funkcja aktywacji w bardzo małych sieciach. [2] W trakcie ćwiczenia sprawdziłem eksperymentalnie, że w istocie liniowa funkcja aktywacji nie ma sensu. Po zastosowaniu liniowej funkcji aktywacji, zgodnie z przewidywaniami cała sieć zachowywała się jak model regresji liniowej - złożenie funkcji liniowych jest funkcją liniową. Dodatkowo okazało się, że mimo iż tangens hiperboliczny czasami lepiej sprawuje się w predykcji przy podobnych pozostałych hiperparametrach, to jego trening jest często dłuższy.

2.6 Laboratorium 6

2.6.1 Opis zadania

Ostatnie zadanie polegało na zaimplementowaniu metod przeciwdziałania zjawisku przeuczenia w sieci neuronowej. Zastosowane metody to regularyzacja oraz wczesne zatrzymywanie treningu. Pierwsza z nich polega na dodaniu kary do funkcji straty sieci, tak aby zminimalizować dopasowanie sieci do szumu znajdującego się w danych. Druga metoda polega na kontroli wartości funkcji straty w zbiorze walidacyjnym do którego sieć ma dostęp w trakcie treningu. Jeżeli wartość funkcji straty nie będzie malała lub nawet rosła w trakcie kolejnych epok to trening należy zakończyć wcześniej. [2]

2.6.2 Podjęte kroki i wyniki eksperymentów

Po zaimplementowaniu wyżej wspomnianych metod przetestowałem je na trzech rekomendowanych zbiorach - wykonałem jedno zadanie regresji oraz trzy klasyfikacji.

Aby odpowiednio sprawdzić działanie wszystkich metod, przetestowałem wszystkie możliwe kombinacje regularyzacji oraz wczesnego zatrzymania trenin-

gu. Wyniki tych eksperymentów znajdują się w tabelach poniżej. W przypadku jednego zbioru danych wszystkie hiperparametry były zamrożone, ustalone na uniwersalne wartości.

Parametr patience został ustawiony na domyślną wartość 5. Zbiór walidacyjny we wszystkich eksperymentach został wydzielony jako 20% danego zbioru testowego. Zbiór walidacyjny był małego rozmiaru, aby nadmiernie nie obciążać czasowo i obliczeniowo eksperymentu - obliczanie wartości estymatorów dopasowania odbywa się po każdej iteracji propagacji wstecznej. W kolejnych tabelach 8, 9, 11, 12 znajdują się wyniki kolejnych eksperymentów dotyczących badania metod regularyzacji.

W tabeli 8 widzimy porównanie dla zadania regresji na zbiorze *multimodal-sparse*. Ciekawym zjawiskiem są tutaj znacznie niższe wartości

regularyzacja	early stop	liczba epok	train MSE	val MSE	test MSE
-	-	10000	5713±149	6057±163	5628±157
-	tak	227±21	**4571±91**	**4453±83**	**3901±79**
L1	-	10000	5987±143	6101±147	5323±139
L1	tak	2493±199	6179±131	6497±137	5753±131
L2	-	10000	5927±131	5909±127	5378±131
L2	tak	383±29	4693±97	4618±89	4567±83

Tabela 8: Wyniki eksperymentu dla zadania regresji - zbiór *multimodal-sparse*. Co ciekawe, wartości funkcji straty dla zbioru testowego są tutaj na ogół niższe niż dla zbioru treningowego. Ta różnica może wynikać z bardzo małego rozmiaru zbiorów treningowego i walidacyjnego, co pociąga za sobą dużą wariancję estymatorów błędu [5]

regularyzacja	early stop	liczba epok	train Fscore	val Fscore	test Fscore	test acc
-	-	10000	**0.97±0.02**	**0.92±0.02**	**0.93±0.02**	0.94±0.01
-	tak	6533±1840	**0.96±0.01**	**0.91±0.02**	**0.93±0.01**	**0.95±0.02**
L1	-	10000	0.90±0.03	0.87±0.02	0.90±0.02	0.91±0.02
L1	tak	9134±1032	0.88±0.04	0.86±0.03	0.88±0.03	0.89±0.02
L2	-	10000	0.91±0.02	0.85±0.02	0.87±0.02	0.88±0.01
L2	tak	10000	0.92±0.03	0.84±0.02	0.86±0.02	0.87±0.02

Tabela 9: Wyniki eksperymentu dla zadania klasyfikacji - zbiór *rings3-balanced*

2.6.3 Wnioski

Metody przeciwdziałania zjawisku przeuczenia są niesamowicie ważne w sieciach neuronowych. Zastosowane oba mechanizmy regularyzacji dość skutecznie

regularyzacja	early stop	liczba epok	train Fscore	val Fscore	test Fscore	test acc
-	-	10000	**0.99±0.00**	**0.82±0.01**	**0.82±0.01**	**0.83±0.01**
-	tak	4848±2266	0.94±0.04	0.79±0.04	0.79±0.04	0.81±0.02
L1	-	10000	0.97±0.01	0.81±0.01	0.81±0.01	0.82±0.01
L1	tak	4151±3015	0.92±0.03	0.78±0.04	0.78±0.04	0.82±0.03
L2	-	10000	0.94±0.01	**0.82±0.01**	0.81±0.01	0.82±0.01
L2	tak	3046±1741	0.89±0.04	0.77±0.03	0.77±0.03	0.81±0.01

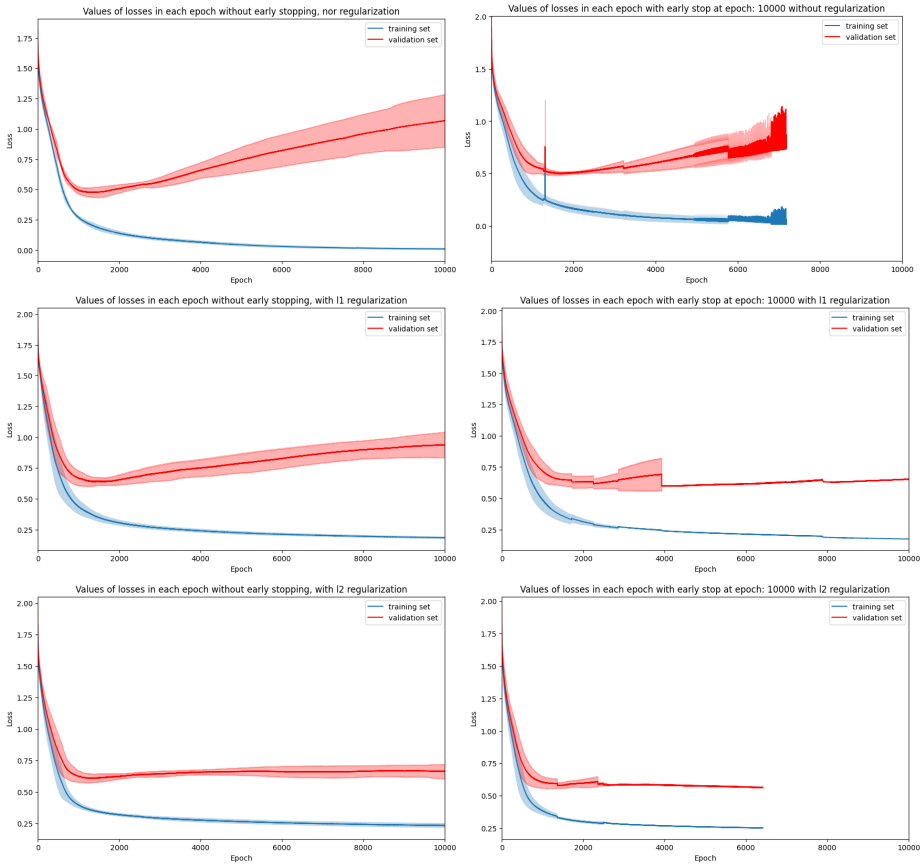
Tabela 10: Wyniki eksperymentu dla zadania klasyfikacji - zbiór *rings5-sparse*

regularyzacja	early stop	liczba epok	train Fscore	val Fscore	test Fscore	test acc
-	-	10000	**1.00±0.00**	**0.89±0.01**	**0.91±0.00**	**0.91±0.01**
-	tak	5	0.87±0.01	0.56±0.01	0.55±0.00	0.55±0.00
L1	-	10000	0.99±0.01	**0.89±0.02**	0.92±0.01	0.93±0.01
L1	tak	5	0.86±0.01	0.56±0.01	0.55±0.01	0.55±0.00
L2	-	10000	0.98±0.00	0.85±0.01	0.88±0.01	0.90±0.02
L2	tak	5	0.87±0.01	0.57±0.01	0.55±0.01	0.55±0.00

Tabela 11: Wyniki eksperymentu dla zadania klasyfikacji - zbiór *xor*. Niestety w przypadku tego zbioru early stopping nie pomógł w zwalczaniu overfittingu, na początkowym etapie treningu bez znaczenia jaka regularyzacja zosstała zastosowana, wartość funkcji straty na zbiorze testowym wzrastała - stąd szybkie kończenie działania funkcji

pomogły przeciwdziałać temu zjawisku, mimo że wykorzystywane sieci nie były zbyt skomplikowane, a różnice między zbiorami testowymi i treningowymi zbyt duże. Ostatecznie jedynie w niektórych przypadkach poprawa wyników była istotnie zauważalna, co implikuje, że podobnie jak w wielu innych obszarach uczenia maszynowego, nie istnieje jedno, najlepsze i uniwersalne rozwiązanie dla każdego przypadku. [7]

Stosowanie jednocześnie regularyzacji i metody wczesnego zatrzymania, nie przyniosło znacznie lepszych rezultatów. Okazuje się, że metoda wczesnego zatrzymania i regularyzacja z normą L2 ma podobne własności *ściągania współczynników* do zera, więc równoczesne ich stosowanie nie jest tak skuteczne [2].



Rysunek 6: Wyniki klasyfikacji przy różnych rodzajach regularyzacji na zbiorze *rings5*. Na powyższych wykresach widać bardzo dobrze zjawisko przeuczenia - w prrzypadku niezastosowania metody wczesnego zatrzymania, nawet w przypadku regularyzacji wykres funkcji straty przypomina literę U. Po zastosowaniu metody wczesnego zatrzymania, trening jest zatrzymywany odpowiednio wcześniej - gdy wartość funkcji straty dla zbioru walidacyjnego zaczyna rosnąć. Dziwne zachowanie na wykresach przy wczesnym zatrzymaniu wynikają z uśredniania wartości pod koniec treningu. Widać jednak, że przy wczesnym zatrzymaniu, przeuczenie jest mniej intensywne.

3 Podsumowanie

W ciągu 6 laboratoriów udało zaimplementować mi się sieć neuronową MLP w Pythonie. Wykorzystywałem jedynie podstawowe biblioteki, które pozwalają na operacje matematyczne i wczytywanie oraz wizualizacje danych - nie użyłem bardziej zaawansowanych bibliotek, które oferują już zaimplementowane sieci neuronowe. Sieć neuronowa, wraz z pomocniczymi funkcjami, została zaimple-

regularyzacja	early stop	liczba epok	train Fscore	val Fscore	test Fscore
-	-	10000	**0.99**	0.88	0.91
-	tak	10	0.88	0.57	0.55
L1	-	10000	0.98	**0.90**	**0.92**
L1	tak	2320	0.97	0.84	0.89
L2	-	10000	0.98	0.88	0.90
L2	tak	5	0.86	0.56	0.54

Tabela 12: Wyniki eksperymentu dla zadania klasyfikacji - zbiór *xor3-balanced*

mentowana w stworzonym przeze mnie pakiecie *networks*, stosując odpowiednie obiektowe podejście.

Uważam, że realizacja tego zadania była dla mnie niesamowicie wartościowa i wiele się w trakcie tego ćwiczenia nauczyłem. Mam nadzieje, że zrozumienie podwalin budowy podstawowej sieci neuronowej, pozwoli mi na lepsze zrozumienie bardziej skomplikowanych struktur i rozwój w dziedzinie uczenia maszynowego i deep learningu.

Literatura

[1] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[2] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. `http://www.deeplearningbook.org`.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015.

[4] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[5] prof. dr. hab. Jan Mielniczuk. slides on advanced machine learning. dostęp: 2024–03-16.

[6] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning*, Proceedings of Machine Learning Research, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[7] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.