
SPRAWOZDANIE Z PROCESU IMPLEMENTACJI SIECI MLP

9 kwietnia 2021

Opis tematu

Cele

Założeniem przeprowadzonych rozważań było zapoznanie się z budową modelu perceptronu wielowarstwowego – sieci neuronowej typu feedforward. Głównym celem była wizualizacja budowy sieci i zmian wartości parametrów w trakcie procesu uczenia w prostych problemach predykcyjnych.

Krótkie wprowadzenie teoretyczne

Multilayer Perceptron, MLP to typ sztucznych sieci neuronowych składający się zwykle z jednej warstwy wejściowej, kilku warstw ukrytych oraz jednej warstwy wyjściowej. Warstwa wyjściowa może składać się z neuronów liniowych (w przypadku regresji) lub neuronów nielinowych (w przypadku klasyfikacji). Na ogólnym poziomie sieć powinna móc uczyć się na podstawie danych tzn. przyjmować porcje danych reprezentujące obserwacje i odpowiadające im odpowiedzi i na tej podstawie być w stanie ustalić funkcję, która potrafi przekształcać dane wejściowe na predykcje jak najbardziej zbliżone do wartości zmiennej odpowiedzi.

Etap 1.: Bazowa implementacja

Opis wykonanej pracy

W ramach pierwszego etapu przeprowadzonych rozważań należało zaimplementować sieć neuronową typu MLP, parametryzowaną przez: liczbę warstw, liczbę neuronów w każdej z warstw oraz wagi poszczególnych połączeń, w tym biasów. Zgodnie z narzuconym założeniem sieć miała używać sigmoidalnej funkcji aktywacji, na wyjściu natomiast dopuszczana była funkcja liniowa. Na potrzeby realizacji tego zadania zaimplementowana została klasa *MLP*, której obiekty były parametryzowane wymienionymi wcześniej atrybutami. W szczególności liczby neuronów w kolejnych warstwach miały być podawane w formie tablicy, której długość powinna odpowiadać liczbie warstw w żądanej architekturze sieci, pierwszy element musiał być tożsamy z wymiarem danych wejściowych, a ostatni z oczekiwana liczbą wyjść sieci. Omawiana implementacja obejmowała metodę *forward()* przyjmującą dane wejściowe i przekazującą je w serii operacji wprzód do wszystkich kolejnych warstw.

Wyniki eksperymentów

Przygotowaną implementację należało wykorzystać do rozwiązania zadania regresji na dostarczonych danych przy zadanych architekturach sieci. W ramach eksperymentów należało rozważyć architektury:

- ↪ z jedną warstwą ukrytą o 5 neuronach,
- ↪ z jedną warstwą ukrytą o 10 neuronach,
- ↪ z dwoma warstwami ukrytymi o 5 neuronów każda.

i zbudować po jednej sieci dla zbiorów: *square-simple* i *steps-large* ręcznie dobierając wartości parametrów. Uzyskane wyniki zostały przedstawione w tabeli 1.

MSE dla dostarczonych zbiorów danych i zadanych architektur		
Zbiór danych	Architektura	MSE
<i>Square-simple</i>	1 warstwa ukryta o 5 neuronach	8.366
<i>Square-simple</i>	1 warstwa ukryta o 10 neuronach	8.366
<i>Square-simple</i>	2 warstwy ukryte po 5 neuronów	871.105
<i>Steps-large</i>	1 warstwa ukryta o 5 neuronach	0.95045
<i>Steps-large</i>	1 warstwa ukryta o 10 neuronach	0.95045
<i>Steps-large</i>	2 warstwy ukryte po 5 neuronów	77.633

Tabela 1: Wartości miary MSE dla sieci z ręcznie dobranymi wartościami wag.

Wnioski

Ze względu na nieefektywność procesu ręcznego doboru wag oraz proporcjonalność stopnia trudności ich doboru do poziomu skomplikowania stosowanej architektury, podstawową konkluzją była potrzeba znalezienia sposobu automatyzacji procesu parametryzacji.

Etap 2.: Implementacja propagacji wstecznej błędu

Opis wykonanej pracy

Na potrzeby bieżącego etapu należało rozszerzyć przygotowaną wcześniej implementację sieci neuronowej o możliwość przeprowadzenia procesu uczenia z propagacją wsteczną błędu. W tym celu do istniejącej implementacji dodano funkcje *backprop()* oraz *train()*. Pierwsza z nich odpowiadała za przeprowadzenie porównania predykcji z właściwymi wartościami zmiennej odpowiedzi, obliczenie wartości straty i wygenerowanie jej gradientu. Gradient ten to pochodna cząstkowa wartości straty względem każdego elementu z ostatniej warstwy sieci. W ostatnim kroku funkcja *backprop()* przekazywała wyznaczony gradient wstecz przez wszystkie warstwy. Obliczano przy tym gradienty dla parametrów i te gradienty były zapisywane w odpowiednich operacjach. Funkcja *train()* odpowiadała za aktualizację wag i biasów na podstawie wyniku przeprowadzonej propagacji wstecznej błędu. Implementacja funkcji *train* objęła dwa warianty aktualizacji wag w procesie uczenia: wersję z aktualizacją wag po prezentacji wszystkich wzorców i drugą z aktualizacją po prezentacji kolejnych porcji (ang. *batch*).

Wyniki eksperymentów

Na drodze eksperymentów należało porównać szybkość uczenia dla każdego z wariantów oraz rozważyć trzy metody inicjowania wag do procesu uczenia. Efektywność uczenia sieci należało przetestować na zadanych zbiorach: *square-simple*, *steps-small* i *multimodal-large*.

Eksperymenty dla wszystkich kombinacji zbiorów, wariantów inicjalizacji i aktualizacji zostały powtórzone 50 razy, otrzymane rezultaty zostały zagregowane (obliczono wartości podstawowych statystyk) i przedstawione w zamieszczonych niżej tabelach (tabele 2-4).

Zbiór *square-simple*

Wagi z rozkładu jednostajnego na przedziale [0, 1]					
Batch	Średnie MSE	Minimum MSE	Maksimum MSE	Odchylenie std. MSE	Średni czas uczenia
10.0	12.643634	5.912571	33.312178	5.335768	0.506815
20.0	13.460079	5.873654	21.805111	4.325653	0.597413
50.0	13.889574	8.892049	32.434780	4.022268	0.523701
100.0 (całość)	12.485091	7.892613	20.493086	3.084025	0.556966

Wagi z rozkładu jednostajnego na przedziale [-1, 1]					
Batch	Średnie MSE	Minimum MSE	Maksimum MSE	Odchylenie std. MSE	Średni czas uczenia
10.0	9.799406	5.047427	21.933413	3.556502	0.468309
20.0	9.860416	4.871330	19.811685	3.340989	0.539376
50.0	9.955390	4.490533	20.926792	3.416104	0.490538
100.0 (całość)	10.358034	4.496592	21.307828	3.522311	0.504609

Wagi z rozkładu normalnego standardowego					
Batch	Średnie MSE	Minimum MSE	Maksimum MSE	Odchylenie std. MSE	Średni czas uczenia
10.0	8.049504	4.523435	16.315374	2.507950	0.423929
20.0	7.910050	4.044170	18.897859	2.975172	0.443620
50.0	7.093778	4.133146	11.163663	1.821443	0.528831
100.0 (całość)	7.657029	4.846737	16.723033	2.424914	0.609939

Tabela 2: Statystyki dla zbioru *square-simple*

Zbiór *steps-small*

Wagi z rozkładu jednostajnego na przedziale [0, 1]					
Batch	Średnie MSE	Minimum MSE	Maksimum MSE	Odchylenie std. MSE	Średni czas uczenia
5.0	12.568424	9.986691	14.983697	1.206049	0.381441
10.0	12.831569	9.766502	14.864000	1.037339	0.442443
25.0	12.692304	9.785290	14.020400	1.055783	0.438197
50.0 (całość)	12.823889	10.080947	13.645882	0.663135	0.490067
Wagi z rozkładu jednostajnego na przedziale [-1, 1]					
5	10.476052	9.155576	12.181807	0.751752	0.442858
10.0	10.451113	9.606521	12.668735	0.536183	0.452118
25.0	10.496054	9.777637	11.295098	0.322044	0.493117
50.0 (całość)	10.675836	9.940731	11.206103	0.258057	0.525219
Wagi z rozkładu normalnego standardowego					
5	10.353590	8.891288	12.832247	0.680927	0.387595
10.0	10.292974	9.324535	11.908711	0.474993	0.395539
25.0	10.283892	9.723963	11.067686	0.329195	0.464543
50.0 (całość)	10.492616	9.610139	11.142139	0.299499	0.533022

Tabela 3: Statystyki dla zbioru *steps-small*

Zbiór *multimodal-large*

Wagi z rozkładu jednostajnego na przedziale [0, 1]					
Batch	Średnie MSE	Minimum MSE	Maksimum MSE	Odchylenie std. MSE	Średni czas uczenia
10	15.706951	13.062040	20.762024	2.670063	40.645264
100	11.165347	9.713305	13.446099	1.161271	43.980804
1000	11.109059	9.398124	12.462422	1.015080	29.927933
10000 (całość)	10.664581	10.082848	11.215142	0.387911	45.858199
Wagi z rozkładu jednostajnego na przedziale [-1, 1]					
10	10.372553	6.606052	14.898121	2.389733	29.246943
100	7.173462	5.161227	11.052933	2.063298	29.443649
1000	7.901490	5.289080	10.355405	1.991918	29.552150
10000 (całość)	8.778320	5.461490	11.678431	2.399101	48.865691
Wagi z rozkładu normalnego standardowego					
10	10.817211	6.755954	17.260537	3.240975	29.799897
100	7.264094	4.672700	12.540535	2.636687	28.711503
1000	7.797173	4.835234	10.545822	2.152942	29.064719
10000 (całość)	7.078950	5.468716	11.143545	1.933736	49.700941

Tabela 4: Statystyki dla zbioru *multimodal-large*

Wnioski

- Wybór sposobu inicjalizacji wag ma znaczący wpływ na przebieg trenowania sieci (w szczególności na czas uczenia) oraz na finalną wartość błędu średniokwadratowego predykcji.
- Zastosowanie podziału na batche w procesie uczenia istotnie go przyspiesza. Na mocy przeprowadzonych eksperymentów można sformułować wniosek, że zasadnym wyborem jest długość batch'a wynosząca ok. 10% liczności zbioru treningowego.

Etap 3.: Implementacja momentu i normalizacji gradientu

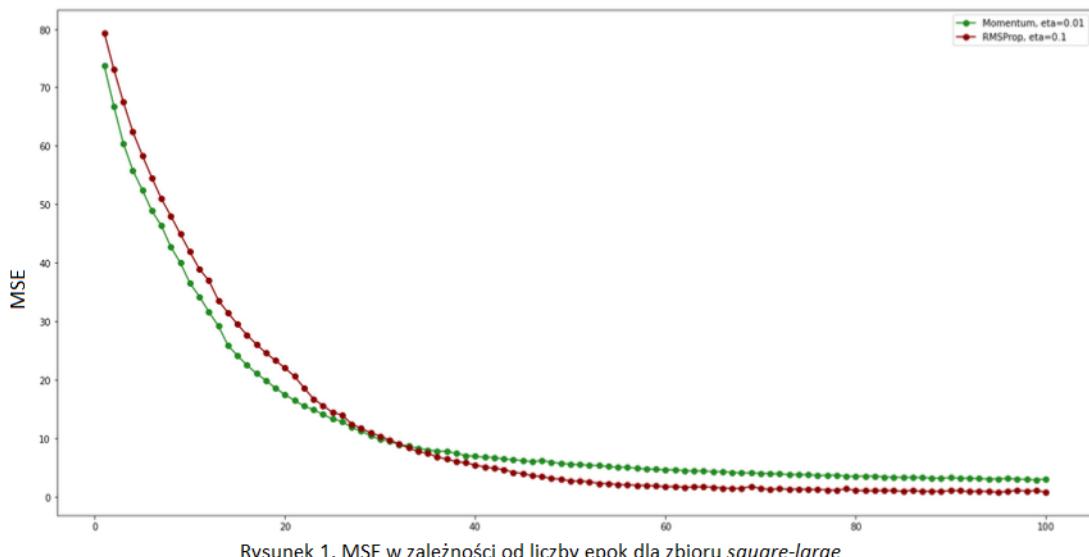
Opis wykonanej pracy

Założeniem trzeciego etapu ewaluacji była implementacja dwóch usprawnień uczenia gradientowego sieci neuronowej: momentu i normalizacji gradientu RMSProp. W związku z tym zaimplementowano dwie dedykowane funkcje, odpowiednio: `trainWithMomentum()`, `trainWithRMSPprop()` realizujące zadane cele i przeprowadzono porównanie szybkości zbieżności procesu uczenia dla obydwu wariantów.

Wyniki eksperymentów

Eksperymenty zostały przeprowadzone na zbiorach: *square-large*, *steps-large* i *multimodal-large*. Zmienności osiąganych wartości miary *MSE* w zależności od liczby epok dla poszczególnych zbiorów przedstawiono na kolejnych wykresach natomiast ostateczne wartości błędów średnio-kwaratowych osiągnięte w procesie uczenia w odpowiadających im tabelach.

Zbiór *square-large*



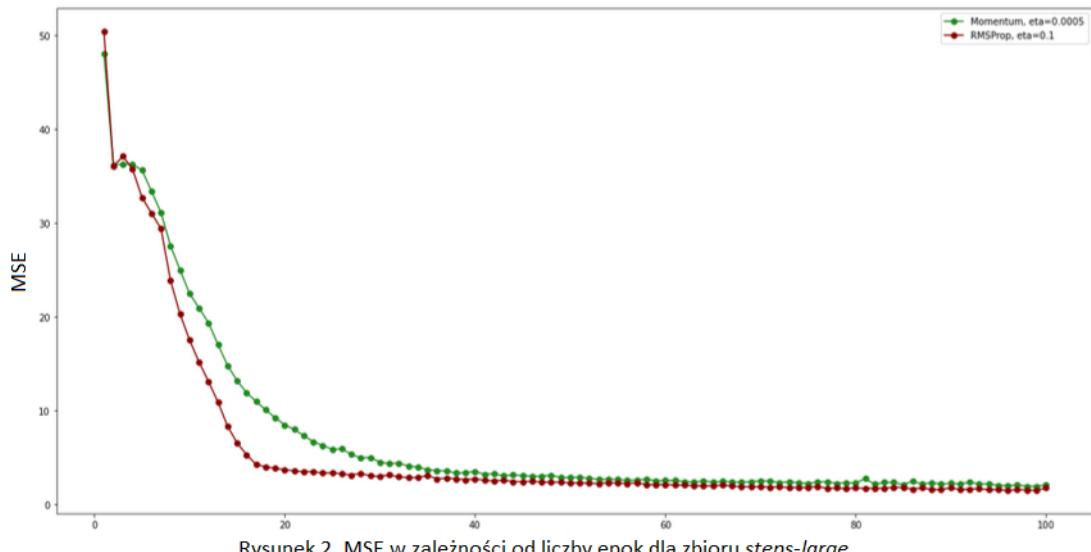
Rysunek 1. MSE w zależności od liczby epok dla zbioru *square-large*

Rysunek 1: Wartości miary *MSE* w zależności od liczb epok dla zbioru *square-large*.

Zbiór <i>square-large</i>		
Metoda uczenia	<i>eta</i>	<i>MSE</i>
Moment	0.0005	1.9904
RMSProp	0.1	1.7352

Tabela 5: Wartości miary *MSE* dla sieci wytrenowanej na 100 epokach

Zbiór *steps-large*



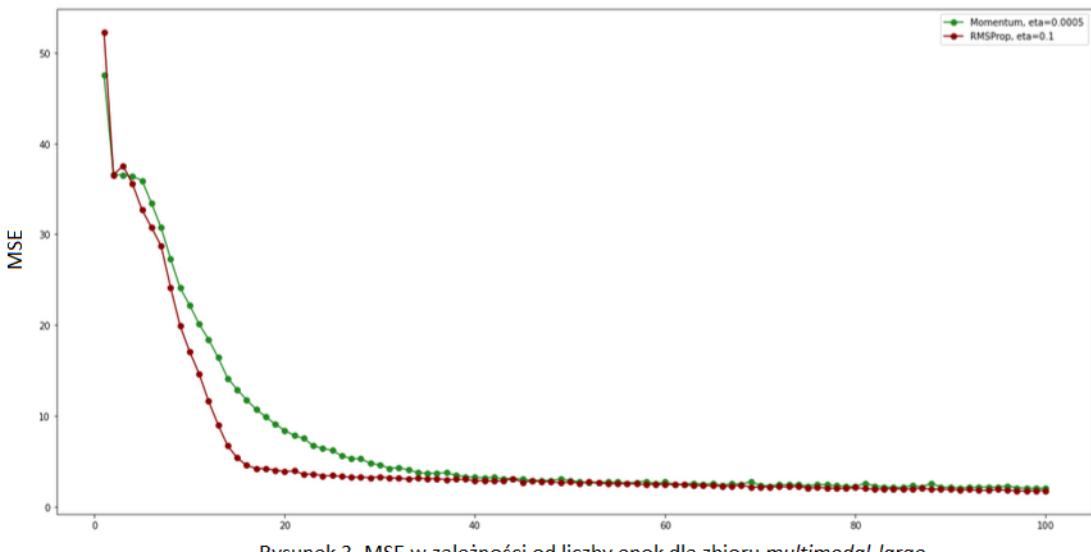
Rysunek 2. Wartości miary MSE w zależności od liczb epok dla zbioru *steps-large*

Rysunek 2: Wartości miary MSE w zależności od liczb epok dla zbioru *steps-large*.

Zbiór <i>steps-large</i>		
Metoda uczenia	<i>eta</i>	<i>MSE</i>
Moment	0.001	3.0165
RMSProp	0.1	0.776

Tabela 6: Wartości miary MSE dla sieci wytrenowanej na 100 epokach

Zbiór *multimodal-large*



Rysunek 3. Wartości miary MSE w zależności od liczb epok dla zbioru *multimodal-large*

Rysunek 3: Wartości miary MSE w zależności od liczb epok dla zbioru *multimodal-large*.

Zbiór <i>multimodal-large</i>		
Metoda uczenia	<i>eta</i>	<i>MSE</i>
Moment	0.0005	2.089
RMSProp	0.1	1.781

Tabela 7: Wartości miary *MSE* dla sieci wytrenowanej na 100 epokach

Wnioski

- Wykorzystanie momentu oraz normalizacji gradientu RMSProp ma istotne znaczenie dla poprawy szybkości zbieżności sieci i jej skuteczności.
- Zarówno momentum, jak i RMSProp miały szybsze tempo zbieżności niż podstawowa metoda uczenia, więc jeśli jesteśmy ograniczeni liczbą epok, użycie jednego z tych algorytmów może być korzystne.
- Trudno jednoznacznie wyłonić lepsze rozwiązańe jednak łatwo zauważać, że w ogólności dla dostatecznie dużej ilości epok wersja z normalizacją gradientu *RMSProp* osiągnęła wyniki co najmniej nie gorsze niż *momentum*.
- W trakcie prowadzenia eksperymentów zauważono, że modyfikacja współczynnika wygaszania momentu *lambda* w metodzie *trainWithMomentum* ma rzeczywisty wpływ na wartość MSE. Zmniejszenie wartości parametru *lambda* powoduje zmniejszenie ostrości aktualizacji wag, co z kolei skutkuje wolniejszym tempem uczenia się i jednocześnie zapobiega zagłuszeniu wpływu nowych danych przez poprzednio uzyskane rezultaty. W związku z tym dla większej liczby epok zmniejszenie wartości parametru *lambda* może mieć pozytywny wpływ na moc predykcyjną sieci.
- Wpływ modyfikacji wartości współczynnika *beta* na zbieżność procesu uczenia i efektywność sieci jest niejednoznaczny a jego dobór jest uwarunkowany naturą rozważanego problemu - zalecane wartości wyznaczone na drodze przeprowadzonych eksperymentów zawierają się w przedziale $[0.8, 0.9]$.
- Normalizacja gradientu *RMSProp* faktycznie zapobiega problemowi eksplozji wag - dla *RMSProp* można było użyć wyższej wartości parametru *eta* (krok uczenia) i nie tylko wyniki były znacznie lepsze, ale także dla dwóch spośród trzech rozważanych zbiorów wariant z *RMSProp* zbiegał szybciej niż pozostałe algorytmy.

Etap 4.: Rozwiązywanie zadania klasyfikacji

Opis wykonanej pracy

W kolejnym tygodniu celem była implementacja funkcji softmax dla warstwy wyjściowej sieci neuronowej oraz badanie szybkości uczenia i skuteczności sieci w wariancie, gdy funkcja ta była używana na ostatniej warstwie i gdy stosowana była tam zwykła funkcja aktywacji.

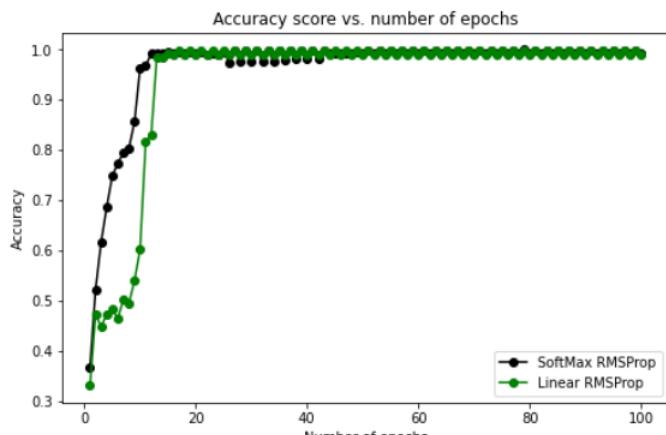
Bieżące zadanie wymagało wprowadzenia zmian wewnątrz funkcji inicjalizującej obiekty klasy *MLP* (gdzie należało dodać funkcję *softmax()* jako wariant funkcji możliwy do zastosowania na ostatniej warstwie) oraz w samym algorytmie uczenia (gdzie z kolei należało uwzględnić zmianę pochodnej wynikającą z wprowadzonej modyfikacji funkcji *_init_()*).

W związku z trudnościami, które pojawiły się przy próbach zmiany liczby neuronów na wyjściu sieci, która w problemach klasyfikacji miała odpowiadać liczbę klas, zaistniała konieczność modyfikacji istniejącej implementacji sieci. Ponieważ wprowadzone zmiany były czysto techniczne (dotyczyły np. zmiany typów obiektów) a ogólna koncepcja pozostała niezmieniona nie zostaną one przytoczone w niniejszym sprawozdaniu.

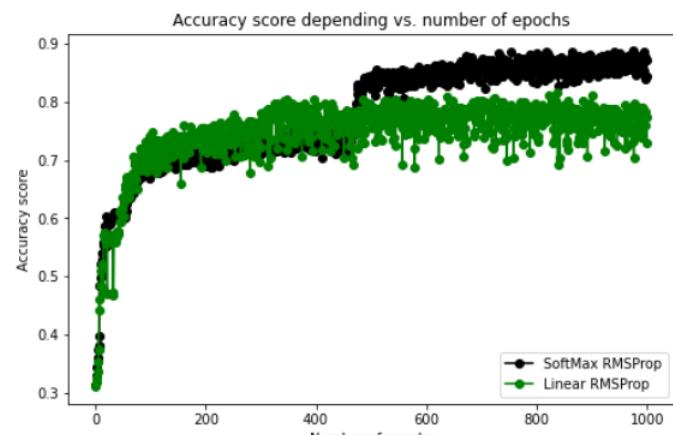
Biorąc pod uwagę przygotowanie danych do podania na wejściu sieci istotną zmianą względem dotychczasowej metodologii była konieczność zakodowania zmiennej odpowiedzi metodą *one-hot*.

Wyniki eksperymentów

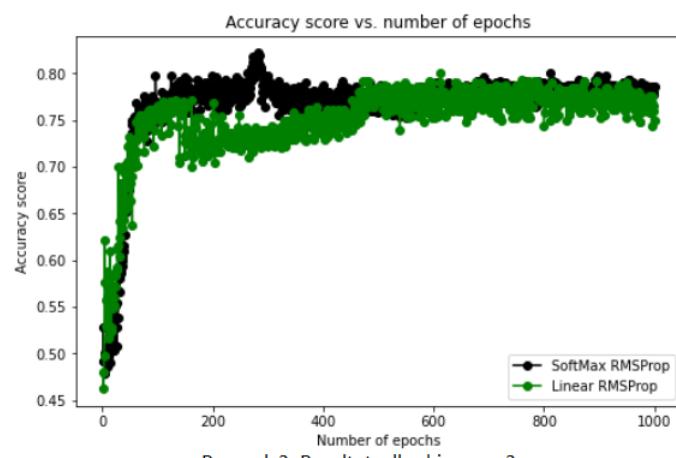
Eksperymenty przeprowadzone zostały na zadanych zbiorach danych: *rings3-regular*, *easy*, *xor3*. Moce predykcyjne dla poszczególnych problemów klasyfikacji oceniano względem miary *Accuracy* a osiągnięte wyniki zostały zagregowane i zwizualizowane na poniższych wykresach.



Rysunek 1. Wyniki eksperymentów dla zbioru easy-dataset



Rysunek 2. Wyniki eksperymentów dla zbioru rings3-regular



Rysunek 3. Wyniki eksperymentów dla zbioru xor3

Rysunek 4: Wartości miary *Accuracy* w zależności od liczb epok dla poszczególnych zbiorów danych

Wnioski

- ↪ Zadowalające rezultaty względem miary *Accuracy* są osiągalne dla obydwu rozważanych wariantów funkcji aktywacji na ostatniej warstwie. W szczególności dla odpowiednio dużej liczby epok wyniki wersji z funkcją softmax są lepsze lub co najmniej niegorsze niż rezultaty osiągnięte dla wariantu z funkcją liniową.
- ↪ Szybkość zbieżności dla obydwu wariantów jest porównywalna jednak wersja z softmaxem na ostatniej warstwie wydaje się bardziej niezawodna.
- ↪ Wraz ze wzrostem stopnia skomplikowania zbioru coraz bardziej uwidaczniają się różnice pomiędzy wynikami poszczególnych wariantów. W szczególności wydaje się, że dla prostych zbiorów (*easy-dataset*) funkcje liniową i softmax można stosować zamiennie natomiast dla bardziej złożonych (*rings3-regular*, *xor3*) funkcja softmax na ostatniej warstwie sprawdza się lepiej.

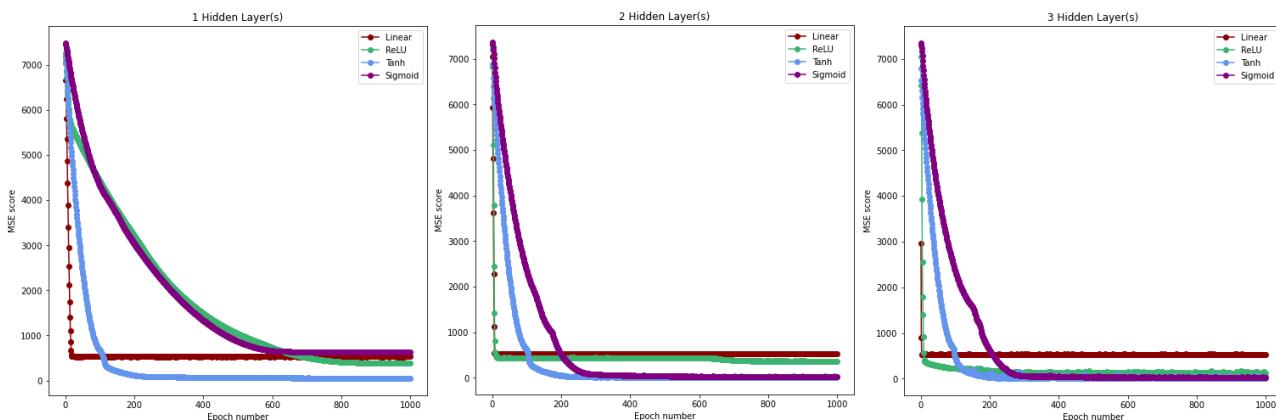
Etap 5.: Testowanie różnych funkcji aktywacji

Opis wykonanej pracy

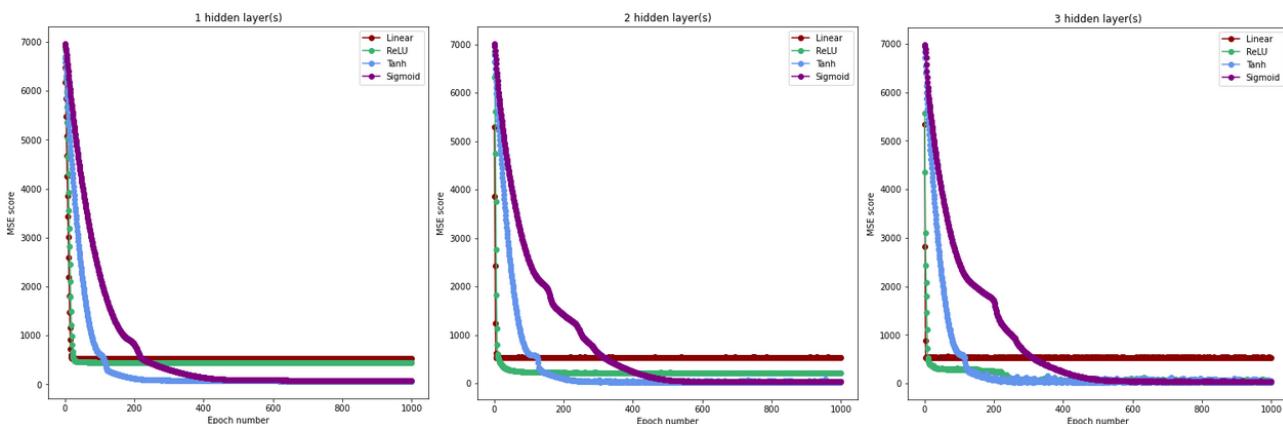
W piątym tygodniu prac należało rozszerzyć istniejącą implementację sieci i metody uczącej o możliwość wyboru funkcji aktywacji spośród funkcji: sigmoidalnej, liniowej, tanh oraz ReLU a następnie porównać szybkości uczenia i skuteczności sieci w zależności od liczby neuronów w poszczególnych warstwach i rodzaju funkcji aktywacji. Zmiany w algorytmie uczenia były analogiczne do wprowadzanych w ramach poprzedniego etapu i obejmowały uwzględnienie zmian funkcji aktywacji i ich pochodnych. Zgodnie ze wskazaniami eksperymenty przeprowadzono dla architektur z jedną, dwiema i trzema warstwami ukrytymi z uwzględnieniem faktu, że różne funkcje aktywacji mogły dawać różne skuteczności w zależności od liczby neuronów i liczby warstw. Ze względu na niezadowalające wyniki klasyfikacji osiągane dla stosowanych dotychczas wariantów inicjowania wag, na potrzeby przeprowadzenia procesu uczenia dla tego rodzaju problemów, jako alternatywny sposób inicjalizacji zaimplementowana została metoda **xavier**.

Wyniki eksperymentów

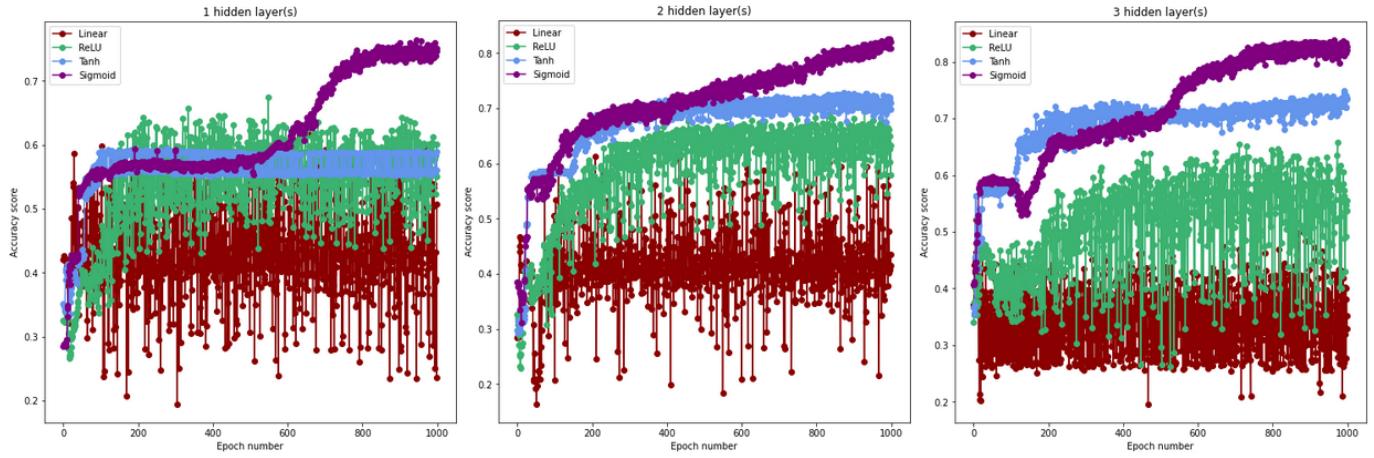
Wspomniane już eksperymenty przeprowadzono dla problemów regresji na zbiorach: *steps-large*, *multimodal-large* i klasyfikacji z wykorzystaniem zbiorów: *rings5-regular*, *rings3-regular*.



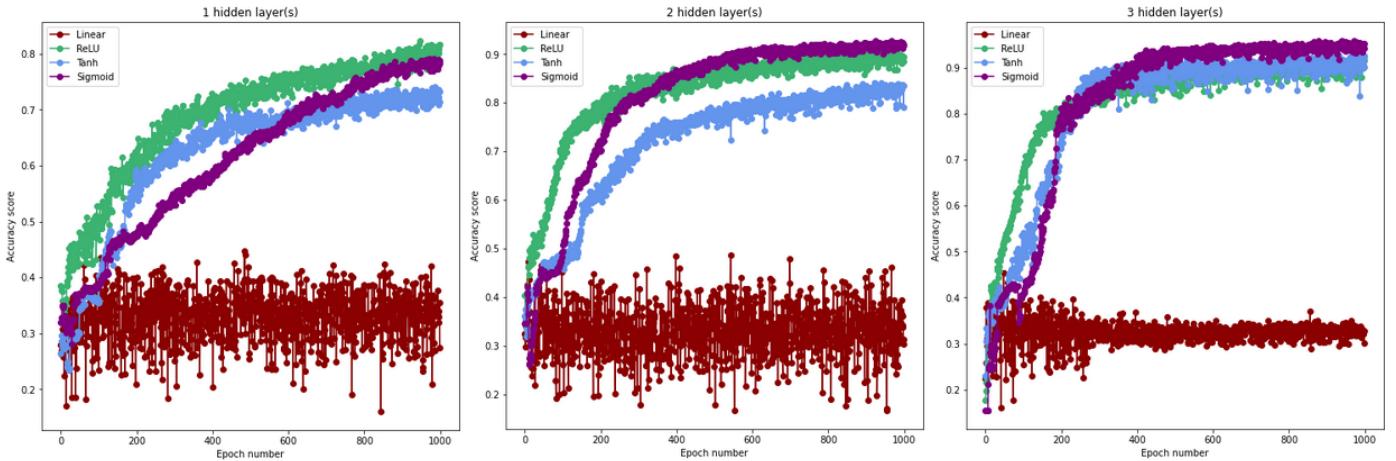
Rysunek 5: Wartości miary MSE w zależności od liczb epok dla poszczególnych architektur w problemie regresji dla zbioru: *steps-large*



Rysunek 6: Wartości miary MSE w zależności od liczb epok dla poszczególnych architektur w problemie regresji dla zbioru: *multimodal-large*



Rysunek 7: Wartości miary *Accuracy* w zależności od liczb epok dla poszczególnych architektur w problemie klasyfikacji dla zbioru: *rings3-regular*



Rysunek 8: Wartości miary *Accuracy* w zależności od liczb epok dla poszczególnych architektur w problemie klasyfikacji dla zbioru: *rings5-regular*

Wnioski

- Architektura tj. liczba warstw i neuronów rzeczywiście miała znaczący wpływ na efektywność predykcji.
- Funkcje sigmoidalna i Tanh są akceptowalne, zarówno w problemach regresji, jak i klasyfikacji.
- Łatwo zauważać, że funkcje ReLU i liniowa nie sprawdzały się dobrze jako funkcje aktywacyjne w problemach regresji. Jednak funkcja ReLU działała całkiem dobrze w zadaniu klasyfikacyjnym dla dużego zbioru danych (*rings5-regular*), nawet przy niezbyt skomplikowanych architekturach (rys.8.).
- Funkcja liniowa nie jest adekwatna do problemów klasyfikacyjnych - nie ma możliwości, aby regularny model regresji liniowej mógł być efektywny dla problemów klasyfikacyjnych (problem błędnej specyfikacji).

Etap 6.: Zjawisko przeuczenia + regularyzacja

Opis wykonanej pracy

W ostatnim etapie prac zaimplementowano mechanizm regularyzacji wag w sieci oraz mechanizm zatrzymywania uczenia przy wzroście błędu na zbiorze walidacyjnym. Na drodze eksperymentów przeprowadzonych na zbiorach: *multimodal-sparse*, *rings5-sparse*, *rings3-balance*, *xor3-balance* porównano skuteczność na zbiorze testowym dla różnych wariantów przeciwdziałania przeuczeniu. Modyfikacje istniejącej implementacji zostały wdrożone w oparciu o wyprowadzenie pochodnej dla nowej funkcji kosztu z regularyzacją L_2 i polegały na modyfikacji sposobu aktualniania wag wewnątrz procedury uczącej. Konieczne było zastosowanie podziału na zbiory treningowy i walidacyjny wewnątrz procedury *trainWithRMSPProp()* tak, aby móc śledzić przyrosty błędów predykcji na zbiorze walidacyjnym w kolejnych epokach. Przyjęto założenie, że proces uczenia zostanie zatrzymany natychmiast po zaobserwowaniu 10%-owej różnicy między bieżącą a minimalną/maksymalną wartością błędu oraz, że do zatrzymania może dojść najwcześniej po 50 epokach.

Wyniki eksperymentów

Przeprowadzone eksperymenty polegały na pięćdziesięciokrotnym przeprowadzeniu procesu uczenia sieci neuronowej z regularizacją i bez oraz każdorazowym wyliczeniu wartości błędu średniokwadratowego lub dokładności w zależności od natury rozważanego problemu. Porównywane sieci były jednakowo sparameryzowane a jedynym czynnikiem je odróżniającym było wdrożenie regularyzacji. Wyniki zostały następnie zagregowane a podstawowe statystyki dla otrzymanych rezultatów umieszczone w poniższej tabeli.

Problem regresji			
Zbiór <i>multimodal-sparse</i>			
Regularizacja	Średnie MSE	Maksymalne MSE	Odchylenie std. MSE
TAK	7.255764	7.868503	0.1852
NIE	8.016637	8.955756	0.253122
Problem klasyfikacji			
Zbiór <i>rings3</i>			
Regularizacja	Średnie Acc	Maksymalne Acc	Odchylenie std. Accuracy
TAK	0.72509	0.7655	0.014742
NIE	0.71712	0.7455	0.012418
Zbiór <i>rings5</i>			
TAK	0.75934	0.819	0.028582
NIE	0.75169	0.7995	0.024809
Zbiór <i>xor3</i>			
TAK	0.723525	0.77125	0.016355
NIE	0.725775	0.76	0.018085

Tabela 8: Podstawowe statystyki wyznaczone dla wartości miar względem, których oceniano otrzymane predykcje sieci.

Wnioski

- Regularizacja chroni nas przed overfittingiem, dokładając w każdej iteracji "karę" za zbyt duże co do wartości bezwzględnej wagi.
- W przeprowadzonych eksperymentach przyrost błędu na zbiorze walidacyjnym nigdy nie był wystarczająco duży, aby spowodować wcześniejsze zakończenie uczenia sieci.

Nie mniej jednak, mechanizm zatrzymywania trenowania sieci przy wzroście błędu jest wartym stosowania usprawnieniem. Jest on bowiem dość prosty w implementacji, ułatwia czasochłonny proces estymacji ilości epok potrzebnych do wytrenowania się danej sieci a w szczególnych przypadkach może istotnie przyspieszyć proces uczenia.

- Z porównując wyniki z powyższej tabeli (tab. 8.) można dość do wniosku, że zastosowanie regularyzacji poprawiło trafność predykcji na zbiorze testowym. Aby wyciągnąć jednoznaczne wnioski należałoby lepiej sparametryzować sieć tak, aby wyjściowa (bez regularyzacji) dokładność jej predykcji, mierzona względem odpowiednich miar była większa. Wówczas bowiem można byłoby mieć pewność, że poprawa w istocie wynika z zastosowania regularyzacji a nie z obecnej w procesie uczenia losowości.