Modele perceptronu wielowarstwowego - sprawozdanie z implementacji

Paweł Koźmiński

9 kwietnia 2021r,

Spis treści

1 Wstęp

Pierwsza z trzech części laboratorium przeprowadzanych w ramach przedmiotu Metody Inteligencji Obliczeniowej w Analizie Danych polegała na samodzielnej implementacji sieci neuronowej typu feedforward - modelu perceptronu wielowarstwowego oraz rozmaitych do niej dodatków wspierających proces trenowania, dopasowania do prezentownych danych. W kolejnych sekcjach zostaną zaprezentowane efekty prac z pierwszych 6 tygodni semestru letniego.

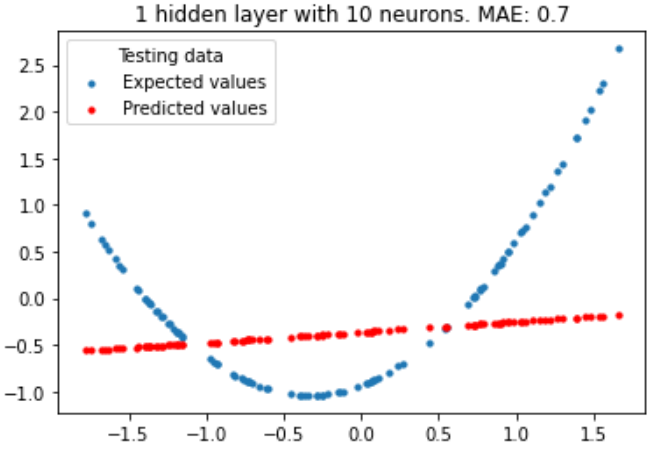
2 Bazowa implementacja sieci

Proces budowania implementacji modeli sieci neuronowych rozpoczął się od zaprogramowania podstawowej wersji rozwiązania. Pierwsza wersja umożliwiała użytkownikowi modyfikację niewielu parametrów, takich jak:

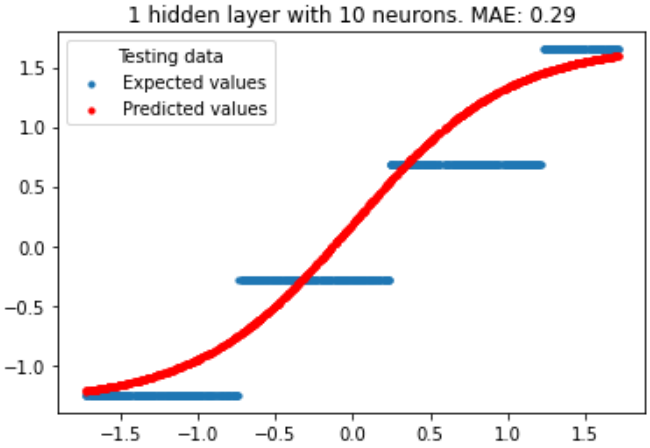
- liczba wejść do warstwy,
- liczba neuronów w warstwie,
- wagi przypisane odpowiednim wejściom do neuronów,
- wartości obciążeń (ang. bias),
- funkcja aktywacji

Kluczowym dla kolejnych tygodni było również zaplanowanie kształtu poszczególnych macierzy oraz wektorów wag i obciążeń w sieci. Zadaniem pierwszych laboratoriów było również pierwsze przetestowanie różnych architektur sieci.

Ze względu na prostotę pierwszej implementacji wyniki zwracane przez sieć były dalekie od ideału. Należy pamiętać, że wartości wag były dobierane ręcznie. Modele wyłącznie symbolicznie dopasowywały się do danych, co można zaobserwować na poniższych wykresach prezentujących najlepsze dla każdego ze zbiorów wyniki.



Rysunek 1: Model składający się z 1 warstwy z 10 neuronami. Dane po standaryzacji.

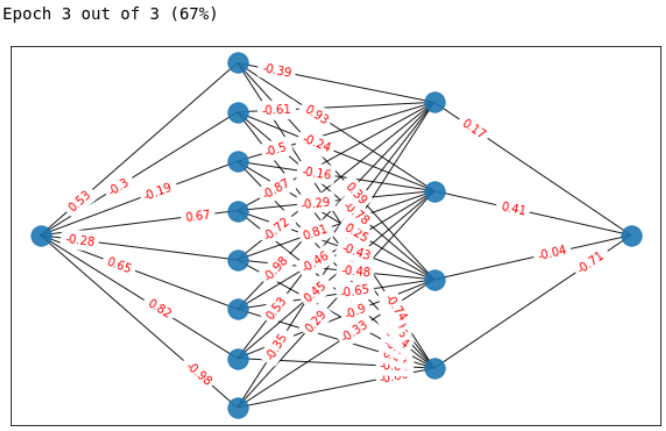


Rysunek 2: Model składający się z 1 warstwy z 10 neuronami. Dane po standaryzacji.

3 Propagacja wsteczna błędu

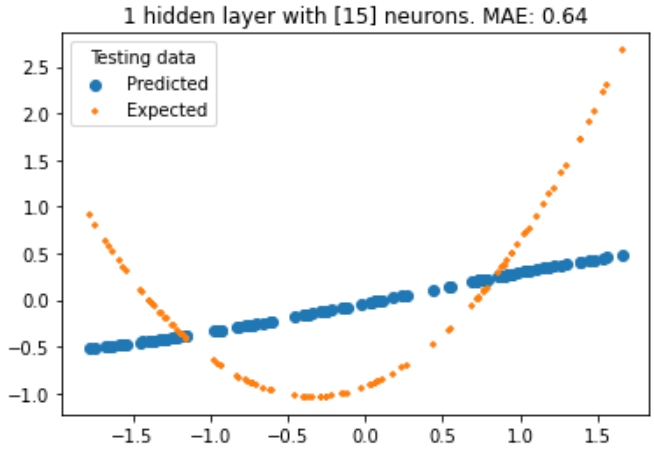
Po niezadowalających wynikach osiąganych przez sieci stworzone w trakcie pierwszego laboratorium priorytetem stało się poprawienie jakości dopasowania modelu do danych. Można to uczynić przy pomocy propagacji wstecznej błędu. Na podstawie wartości błędu predykcji, gradientu funkcji straty (we wszystkich rozważaniach w poniższym raporcie było nią MSE - błąd średniokwadratowy) oraz gradientów funkcji aktywacji dokonywane są modyfikacje wartości wag wej-

ścia w kolejnych warstwach. Istotnym elementem tworzenia sieci jest metoda inicjalizacji wag. Zaimplementowano trzy sposoby: metodę He, metodą Xaviera oraz losowanie wag z rozkładu jednostajnego, domyślnie na przedziale [-1, 1).
Aby kontrolować proces uczenia, czyli w gruncie rzeczy zmiany wag po prezentacji sieci kolejnych wzorców, dodałem możliwość graficznego sprawdzenia aktualnych wartości. Udało mi się to przy pomocy biblioteki `networkx` służącej do reprezentacji grafów. Dzięki umieszczeniu wag w różnych położeniach jest możliwość w miarę czytelnej reprezentacji wartości na każdym połączeniu między warstwami.

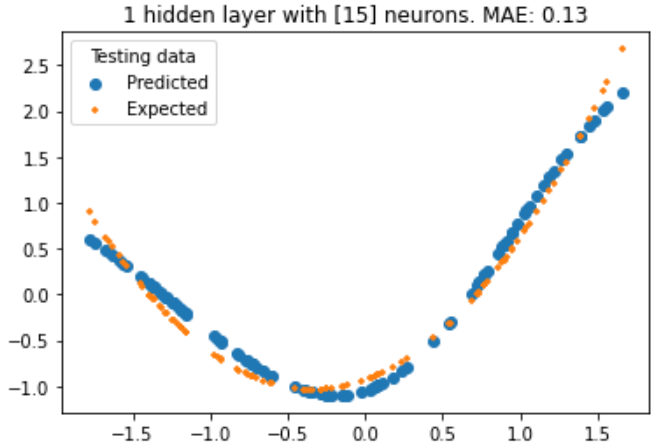


Rysunek 3: Przykładowa wizualizacja sieci składającej się z 2 warstw ukrytych z 8 i 4 neuronami.

Ważnym usprawnieniem procesu uczenia jest dodanie podejścia batchowego. W przypadku użycia tego mechanizmu, wagi w sieci modyfikowane są po prezentacji określonej liczby wzorców, mniejszej niż rozmiar całego zbioru treningowego. Skorzystanie z tego mechanizmu powoduje częstszą aktualizację wag, co zazwyczaj pociąga za sobą wyższe wyniki osiągane przez sieć.
Mechanizmem dodanym w trakcie drugich zajęć powodującym największą poprawę skuteczności modeli okazało się umożliwienie przeprowadzenia wielu epok uczenia. Epoka jest iteracją, w trakcie której prezentuje się wszystkie wzorce ze zbioru treningowego. Poniższe grafiki wskazują modele o tej samej wartości wszystkich parametrów oprócz liczby epok - na pierwszej widać efekty działania modelu, którego uczenie trwało 50 epok, na drugiej - takiego, którego trenowanie trwało 800.

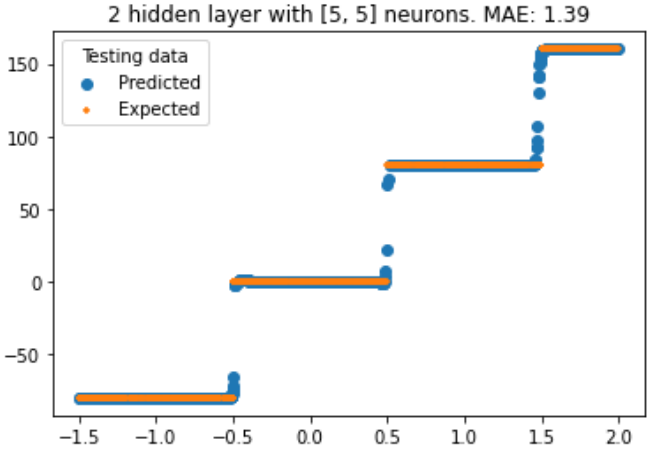


Rysunek 4: Predykcje modelu trenowanego przez 50 epok.



Rysunek 5: Predykcje modelu trenowanego przez 800 epok.

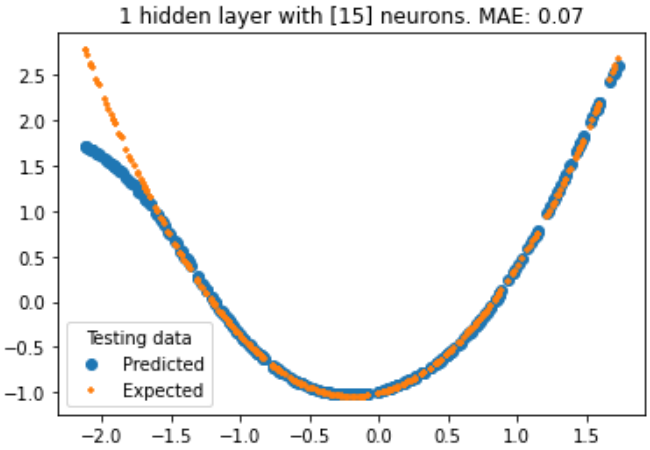
Modele po drugim laboratorium miały już znacznie wyższe możliwości dopasowania się do prostych zbiorów danych.



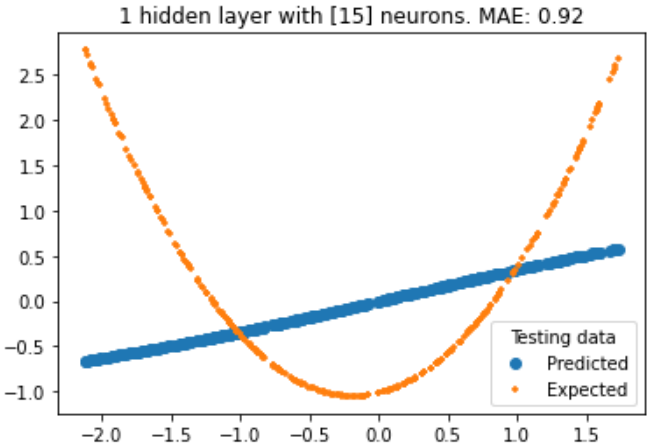
Rysunek 6: Model składający się z 2 warstw z 5 neuronami. Dane bez standaryzacji.

4 Implementacja optymalizatorów - momentu oraz normalizacji gradientu RMSProp

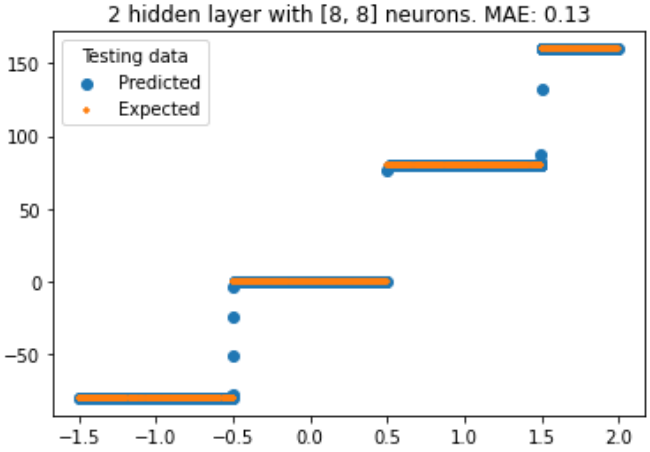
Kolejnym zadaniem było dodanie możliwości korzystania z dwóch metod usprawnień procesu uczenia - metody Momentum oraz RMSProp. Powyższe modyfikacje algorytmu odpowiadają za przyspieszenie procesu uczenia oraz zmniejszają szansę na utknięcie w minimum lokalnym zadania. Oba sposoby przyjmują współczynnik wygaszania $\lambda \in (0, 1)$, zazwyczaj w moich eksperymentach współczynnik ten wynosił więcej niż 0.9 - takie podejście zwracało zadowalające wyniki. Efekt poprawy uczenia został sprawdzony na kilku zbiorach danych. Poniżej znajduje się porównanie skuteczności sieci uczonych z tymi samymi parametrami - z optymalizatorami i bez.



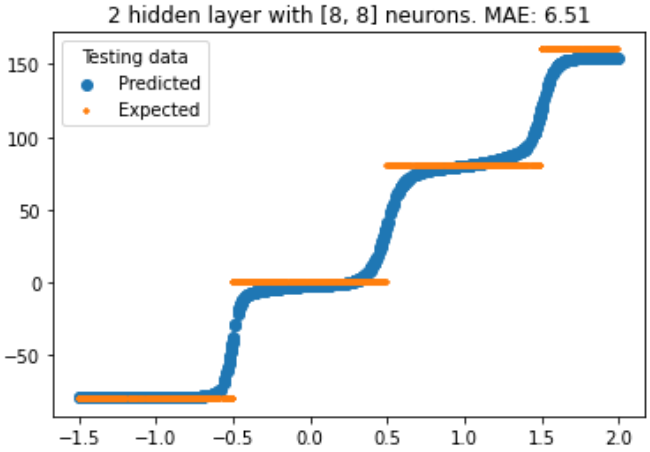
Rysunek 7: Model składający się z 1 warstwy z 15 neuronami z optymalizatorem RMSProp ($\lambda = 0.999$).



Rysunek 8: Model składający się z 1 warstwy z 15 neuronami bez optymalizatora.



Rysunek 9: Model składający się z 2 warstw z 8 neuronami z optymalizatorem Momentum ($\lambda = 0.99$).



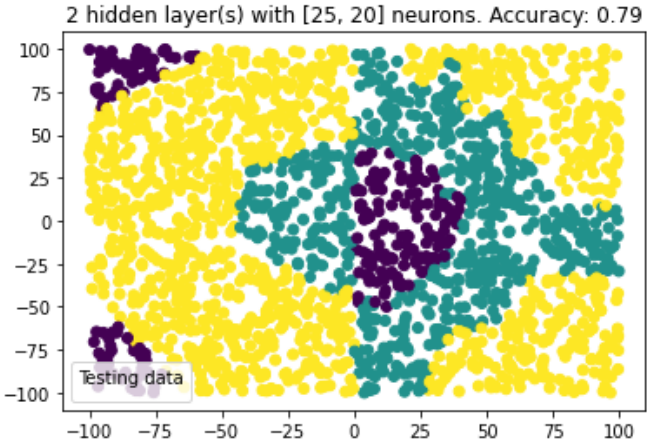
Rysunek 10: Model składający się z 2 warstw z 8 neuronami bez optymalizatora.

Korzystanie z optymalizatorów Momentum i RMSProp przy jednocześnie relatywnie wysokiej wartości współczynnika uczenia pozwala na uzyskanie modeli o wyższej skuteczności, trenowanych w znacznie mniejszej liczbie epok.

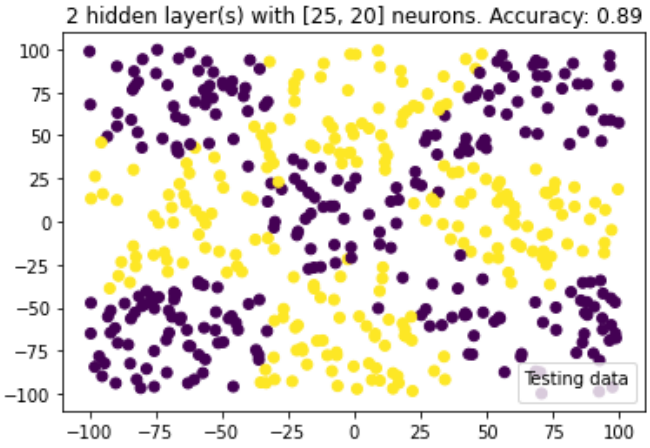
5 Rozwiązywanie zadania klasyfikacji

Wszystkie dotychczas prezentowane efekty działań modeli sieci neuronowych MLP opierały się na rozwiązaniu zadania regresji - przyporządkowaniu pre-

zentowanemu wzorcowi dowolnej liczby rzeczywistej. Kolejnym dodatkiem do istniejącej implementacji sieci było umożliwienie rozwiązywania zadań klasyfikacji, także wieloklasowej. Zostało to uczynione przy pomocy dodania nowej funkcji aktywacji warstwy wyjściowej - softmax. Jest to różniczkowalna funkcja zwracająca nieujemne wartości sumujące się do 1, dzięki czemu mogą być interpretowane jako prawdopodobieństwa przypisania do poszczególnych klas. Danej obserwacji jest oczywiście przypisywana klasa o najwyższej wartości prawdopodobieństwa.
Funkcja softmax jest funkcją wektorową - zarówno na jej wyjściu jak i wejściu mogą się znajdować wektory tego samego rozmiaru a każda z wartości wyjścia jest zależna od każdej wartości wejścia. Z tego powodu jej pochodna jest Macierzą Jacobiego, co wymagało sporych zmian w implementacji w porównaniu z pozostałymi funkcjami aktywacji.



Rysunek 11: Efekt działania sieci neuronowej z wyjściową funkcją aktywacji softmax na zbiorze rings3-regular.

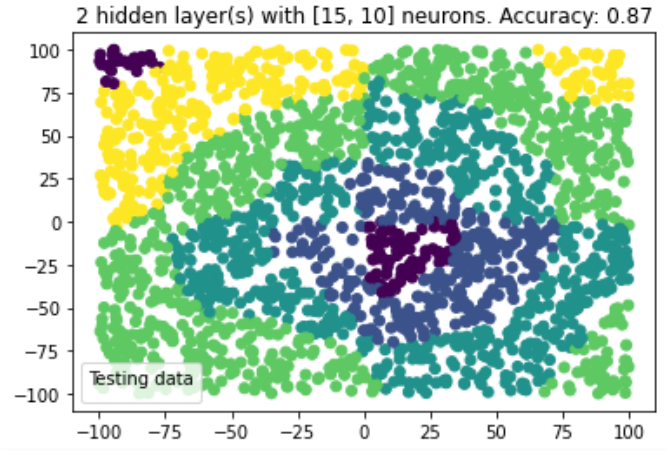


Rysunek 12: Efekt działania sieci neuronowej z wyjściową funkcją aktywacji softmax na zbiorze xor3.

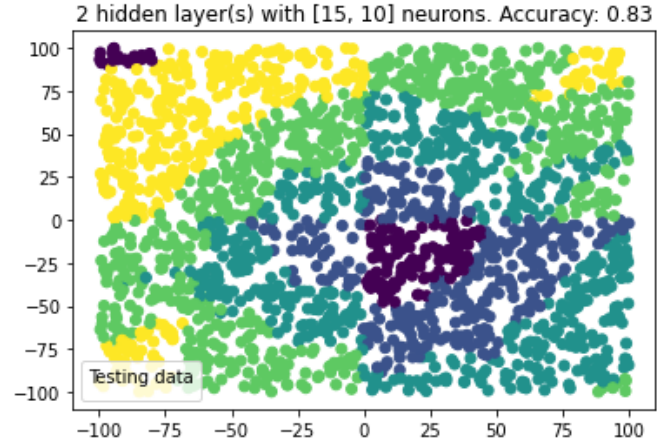
Przeprowadzone eksperymenty porównujące efekty trenowania sieci z funkcją liniową oraz softmax na warstwie wyjściowej nie wykazały znacznej różnicy w skuteczności algorytmów na korzyść którejkolwiek ze stron.

6 Inne funkcji aktywacji

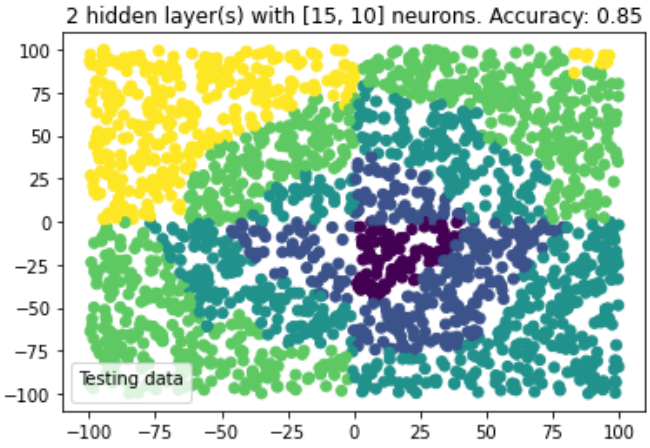
Kolejnym wyzwaniem było dodanie innych, szeroko używanych w świecie sieci neuronowych, funkcji aktywacji warstw ukrytych. Były nimi: tangens hiperboliczny, liniowa (identyczność) oraz ReLU. Niezbędnym elementem było dodanie właściwych funkcji pochodnych w celu wstecznej propagacji błędu. Spośród powyższych oraz zaimplementowanej wcześniej funkcji logistycznej, zdecydowanie najgorzej spisywała się identyczność, co zgadza się z informacjami przekazanymi na wykładzie - przestrzeń przekształceń liniowych może się okazać zbyt uboga do rozwiązywania trudniejszych zadań. W celu porównania prezentuję wykresy przedstawiające predykcje na zbiorze testowym rings5-regular w zależności od różnych funkcji aktywacji.



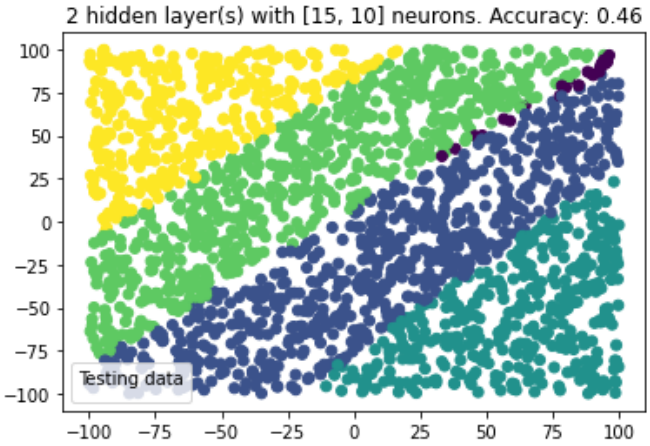
Rysunek 13: Efekt klasyfikacji wieloklasowej sieci neuronowej z funkcją aktywacji ReLU.



Rysunek 14: Efekt klasyfikacji wieloklasowej sieci neuronowej z funkcją aktywacji tanh.



Rysunek 15: Efekt klasyfikacji wieloklasowej sieci neuronowej z funkcją aktywacji sigmoid.

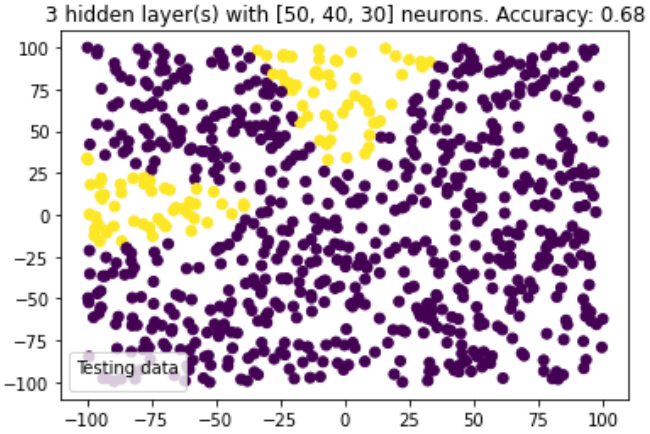


Rysunek 16: Efekt klasyfikacji wieloklasowej sieci neuronowej z funkcją aktywacji identyczność.

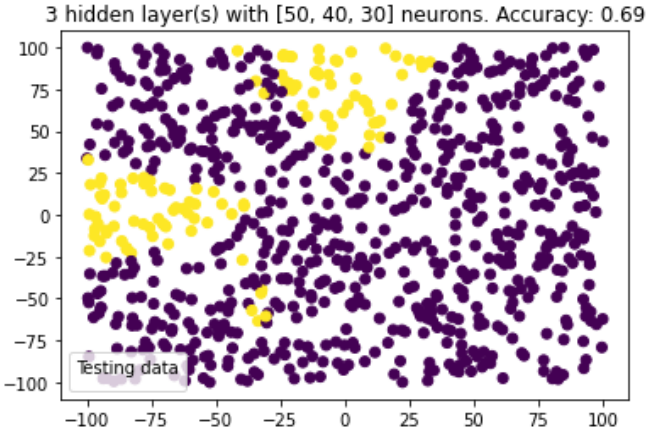
7 Zapobieganie zjawisku przeuczenia

Sporym zagrożeniem przy tworzeniu modeli predykcyjnych jest nadmierne dopasowanie obiektów do danych wykorzystywanych w trakcie uczenia. Głównym celem tworzenia sieci neuronowych jest możliwość wykorzystania rozwiązania do właściwego przewidywania wartości zmiennych objaśnianych dla nowych wzorców, wobec czego muszą one posiadać umiejętność generalizacji swoich ocen. Nadmierne dopasowanie owocowałoby wysoką wartością funkcji straty na zbiorze wzorców wcześniej niewidzianych przez model - np. na tzw. zbiorze

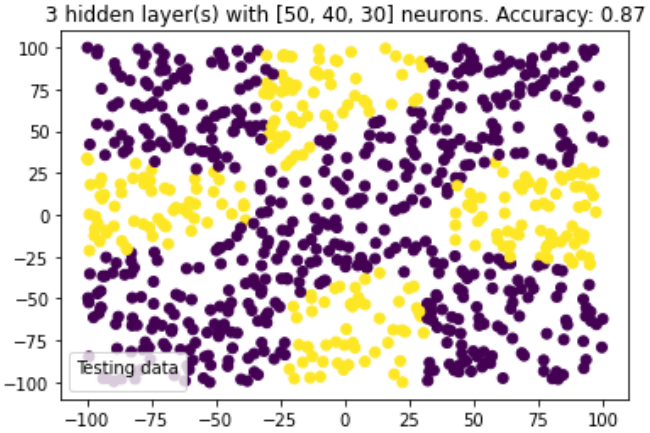
testowym.
Istnieje kilka mechanizmów ochrony przed nadmiernym dopasowaniem modelu. Niektóre z nich to regularyzacja wag, kontrola wartości funkcji straty na zbiorze walidacyjnym, walidacja krzyżowa czy też dropout - kontrolowane porzucanie części neuronów w trakcie treningu. W rozwijanej implementacji dodałem dwa pierwsze z nich: regularyzację L^2 oraz kontrolę funkcji straty na zbiorze walidacyjnym.
Regularyzacja wag L^2 polega na dodaniu do standardowej funkcji straty błędu średniokwadratowego wyrazu zależnego od miary $||x||_2$ wszystkich użytych w sieci wag, przemnożonego przez $\frac{2 \cdot \lambda}{m}$, gdzie m to liczba prezentowanych wzorców a λ to współczynnik regularyzacji. Taka modyfikacja funkcji straty karze wysokie wartości wag, zapobiega zjawisku ich eksplozji oraz nadmiernego dopasowania.
Zatrzymanie procesu uczenia na podstawie wartości funkcji straty możliwe jest po co najmniej 5 pierwszych epokach. Aby stwierdzić, że sieć zbyt mocno dopasowuje się do danych treningowych muszą jednocześnie zajść dwa warunki: wartość funkcji celu na zbiorze walidacyjnym musi być o ponad 10% wyższa niż w dotychczas optymalnym momencie oraz w 5 ostatnich zarejestrowanych epokach musi stale rosnąć.
Zastosowanie różnych metod zapobiegania przeuczeniu zwracało różne rezultaty, które przetestowałem między innymi na zbiorze xor3-balance (cechującym się wysokim niezbalansowaniem klas) przy zastosowaniu jednakowych parametrów uczenia niezwiązanych z zapobieganiem przeuczenia.



Rysunek 17: Efekt uczenia przy jednoczesnej regularyzacji wag oraz monitorowania wartości funkcji straty na zbiorze walidacyjnym.



Rysunek 18: Efekt uczenia przy zastosowaniu wyłącznie monitorowania wartości funkcji straty na zbiorze walidacyjnym.



Rysunek 19: Efekt uczenia przy zastosowaniu wyłącznie regularyzacji wag.

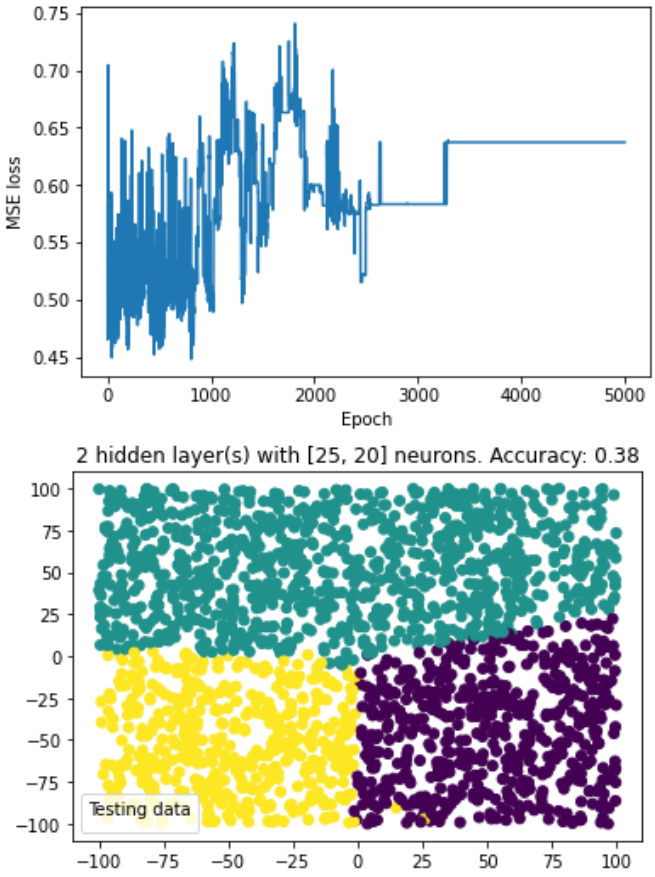
W powyższym eksperymencie najwyższe wyniki zostały osiągnięte przez sieć, która korzystała wyłącznie z mechanizmu regularyzacji wag w procesie uczenia.

8 Dodatkowe eksperymenty

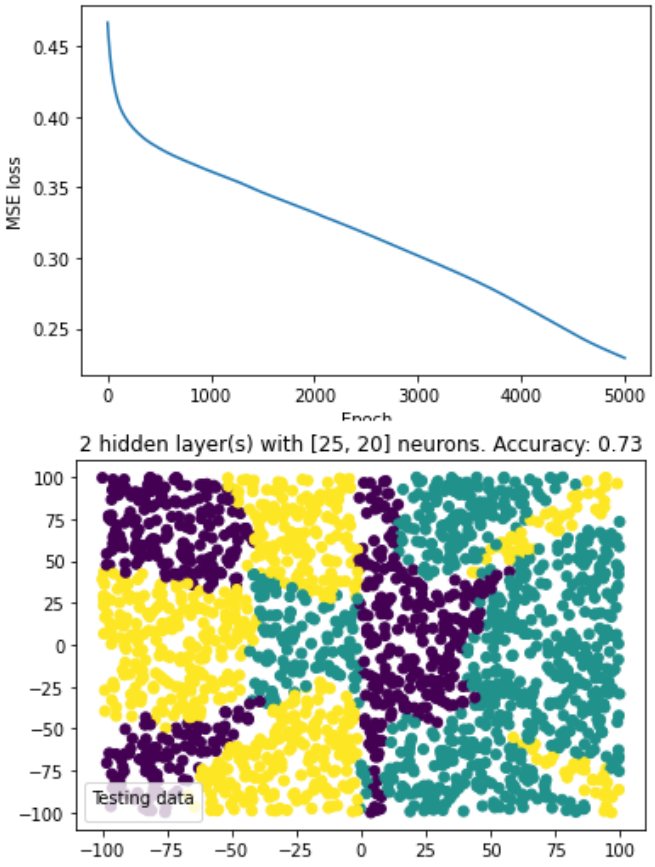
W trakcie wykonywania zadań laboratoryjnych przeprowadziłem rócenież szereg eksperymentów, które nie były celem żadnego konkretnego etapu, a mogą mieć istotny wpływ na proces uczenia.

8.1 Wpływ współczynnika uczenia

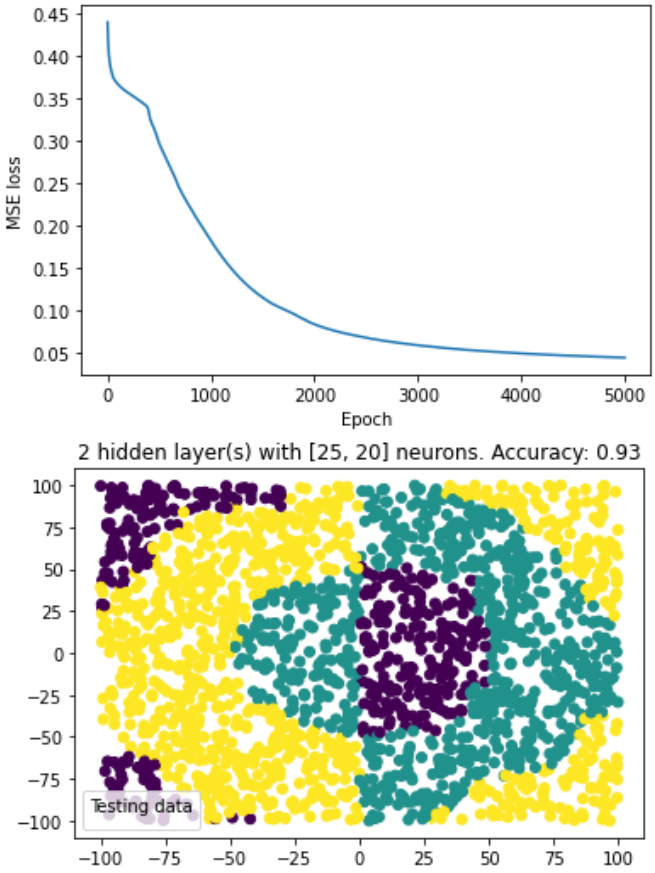
Współczynnik uczenia odpowiada za stopień poziom modyfikacji wag w propagacji wstecznej błędu po zaprezentowaniu każdego zestawu wzorców - im wyższa jest jego wartość, tym większy wpływ ma zmiana. Zbyt mała wartość tego parametru może powodować bardzo wolne zbieganie do optymalnej wartości funkcji straty, zbyt duża może jednak prowadzić do niestabilnego, a co za tym idzie, nieskutecznego procesu rozwiązywania. Poniższe grafiki prezentują efekty procesu uczenia z funkcją aktywacji tanh oraz optymalizatorem RMSProp, bez mechanizmów zapobiegających nadmiernemu dopasowaniu.



Rysunek 20: Efekt uczenia przy zastosowaniu zbyt dużego współczynnika uczenia, równego 0.05



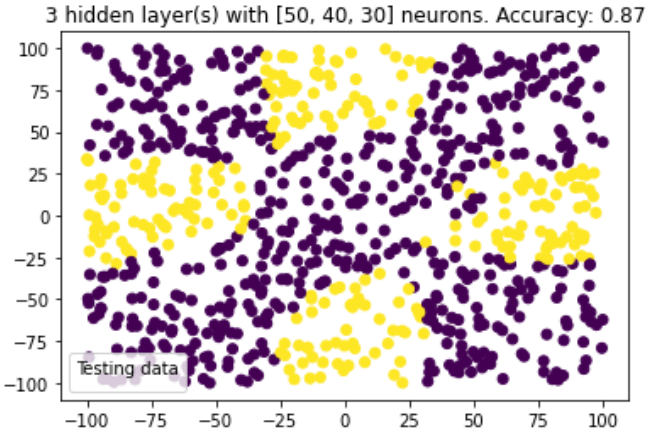
Rysunek 21: Efekt uczenia przy zastosowaniu zbyt małego współczynnika uczenia, równego 0.00001



Rysunek 22: Przykład efektu uczenia przy zastosowaniu odpowiedniego współczynnika uczenia, równego w tym przypadku 0.0001

8.2 Sigmoid jako wyjściowa funkcja aktywacji w zadaniu klasyfikacji binarnej

François Chollet, francuski badacz sztucznej inteligencji i twórca biblioteki Keras w Pythonie informuje w [1], że dla problemu klasyfikacji binarnej rekomendowaną funkcją aktywacji warstwy wyjściowej jest funkcja logistyczna sigmoid. W dotychczasowych rozważaniach w każdym problemie klasyfikacji wykorzystywałem w tym miejscu funkcję softmax. Ciekawy realnego wpływu tej zmiany dokonwałem testu sieci o tych samych parametrach z różnymi funkcjami aktywacji na wyjściu.



Rysunek 23: Efekt uczenia problemu klasyfikacji binarnej z logistyczną funkcją aktywacji na wyjściu.

W porównaniu z funkcją softmax, która była wykorzystywana przy uczeniu na grafice (link), sieć z sigmoidem stabilniej dążyła do optimum. Ponadto, przy wielu uruchomieniach, częściej udawało jej się poprawnie zidentyfikować wszystkie 4 obszary klasy w kolorze żółtym.

9 Podsumowanie

Zajęcia laboratoryjne z Metod Inteligencji Obliczeniowej w Analizie Danych były okazją do stworzenia pierwszych modeli predykcyjnych od zera. Wykonana w ciągu pierwszych 6 tygodni semestru praca zaowocowała stworzeniem narzędzia posiadającego stosunkowo wysokie możliwości predykcyjne, przynajmniej w odniesieniu do załączonych zbiorów danych. Powyższe sprawozdanie zawiera opis iteracyjnego procesu tworzenia modelu perceptronu wielowarstwowego składającego się z stworzenia bazowej implementacji, dodanie wstecznej propagacji błędu, dodania algorytmów wspierających optymalizację, wprowadzenia innych funkcji aktywacji, również dedykowanych do zadania klasyfikacji oraz konfigutacji mechanizmów zapobiegających proces uczenia. Dodatkowo zawarłem opis wpływu współczynnika uczenia na proces trenowania sieci oraz alternatywną funkcję aktywacji w problemie klasyfikacji binarnej.

Literatura

[1] Chollet, F. (2019). Deep learning. Praca z językiem Python i biblioteką Keras. Wydawnictwo Helion.