Skład zespołu: Illia Yatskevich
Paweł Maśluch

Prowadzący: Mikołaj Markiewicz

[PSZT] Podstawy Sztucznej Inteligencji
Semestr 2020Z
Projekt nr 2
MM.SN.R2 Perceptron wielowarstwowy

Treść zadania

Zaproponować architekturę, zaimplementować, wytrenować i przeprowadzić walidację sieci neuronowej do predykcji ceny domu.
Zbiór danych do użycia: Boston Housing - https://www.kaggle.com/c/boston-housing / http://lib.stat.cmu.edu/datasets/boston.
Porównać wyniki dla różnej liczby ukrytych neuronów.
// TO ISTOTNY LINK --> Więcej info na http://home.elka.pw.edu.pl/~mmarkiew/2020Z-pszt.html

Podział odpowiedzialności w zespole

Takie czynności, jak pisanie programu i testowanie jakości rozwiązania dla różnych parametrów, realizowaliśmy wspólnie.
Illia Yatskevich przygotował wykresy i umieścił pliki projektu na GitHubie,
Paweł Maśluch zredagował sprawozdanie.

Zwięzły opis algorytmu/architektury i uzasadnienie sposobu realizacji

Na początku działania algorytmu ładujemy zbiór danych do stworzonych przez nas struktur danych w Pythonie. Dalej, skalujemy dane (nie licząc rzeczywistych cen), dodając jednocześnie kolumnę jedynek (aby mieć możliwość późniejszego wyznaczenia biasu).
Następnie zmieniamy losowo kolejność wierszy w danych (żeby przy każdym uruchomieniu programu mieć inny zbiór uczący).
W naszym algorytmie podzieliliśmy dane na dane treningowe (ok. 80% wszystkich danych) oraz dane testowe (ok. 20% wszystkich danych).
Później definiujemy podstawowe parametry sieci, jak liczba iteracji algorytmu uczącego (n_iterations) oraz parametr dopuszczalnej zmiany wag w pojedynczej iteracji (l_rate).
W dalszej kolejności uruchamiamy algorytm uczący dla naszej sieci, a na końcu obliczamy, ile procent obliczonych wartości (dla danych testowych) różni się o co najwyżej 20% od wyniku prawidłowego.

Opis algorytmu uczącego:

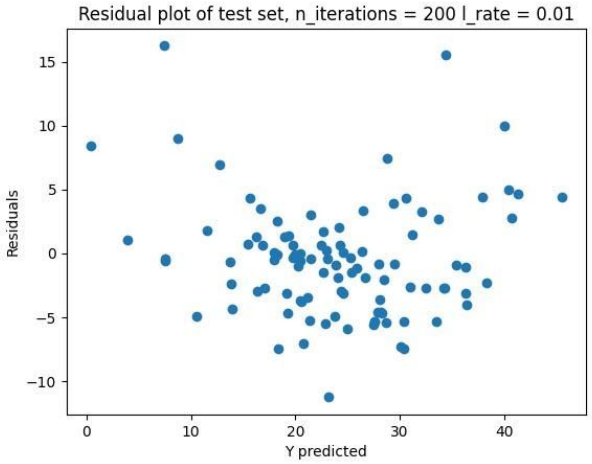
1) Inicjujemy wagi sieci liczbami losowymi o rozkładzie normalnym z wartością oczekiwaną 0 oraz odchyleniem standardowym sqrt(2/14).
2) Dla każdej iteracji i dla każdego wiersza danych uczących:
   a) obliczamy iloczyn macierzy wag przez transponowany wiersz wejścia
   b) sumujemy elementy tak powstałej macierzy
   c) następnie obliczamy predykcję, obliczając wartość funkcji przejścia (dla neuronu) dla obliczonej sumy (jako funkcję przejścia wzięliśmy funkcję RELU(x), zwracającą x dla nieujemnych x, lub 0 w p.p.)
   d) obliczamy błąd predykcji jako różnicę wartości dokładnej i wartości predykcji
   e) aktualizujemy wagi sieci, dodając do aktualnych wag iloczyn parametru l_rate, błędu predykcji, wiersza danych oraz pochodnej macierzy obliczonej w punkcie a)

Raport z przeprowadzonych eksperymentów, sposób testowania, tabelki, wykresy etc.

W eksperymentach zmieniamy liczbę iteracji algorytmu uczącego (n_iterations) oraz parametr l_rate. Dla danych ustawień uruchomiliśmy program 5 razy i obliczyliśmy średni procent trafień (procent trafień, czyli jaki procent predykcji różni się od wartości dokładnych o co najwyżej 20%) oraz średni czas treningu. Zamieszczamy również wykresy obrazujące, jak bardzo obliczona wartość różni się od poprawnej.

Przypadek 1

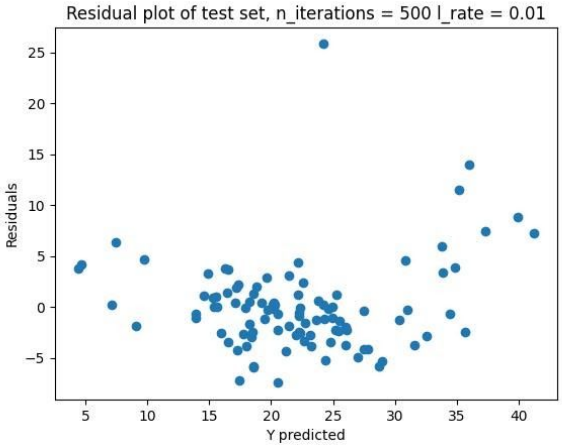
Średni procent trafień wynosi 64.6%, natomiast średni czas treningu wynosi ok. 1.33 sekundy.



Procent trafień wynosi 70%, natomiast czas treningu wynosi ok. 1.23 sekundy.

Przypadek 2

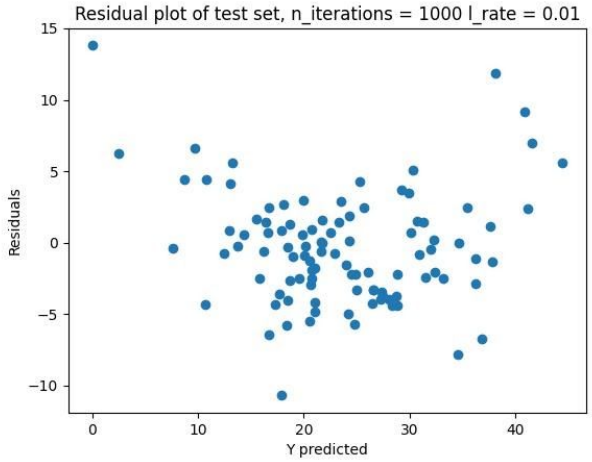
Średni procent trafień wynosi 76.6%, natomiast średni czas treningu wynosi ok. 3.19 sekundy.



Procent trafień wynosi 80%, natomiast czas treningu wynosi ok. 3.22 sekundy.

Przypadek 3

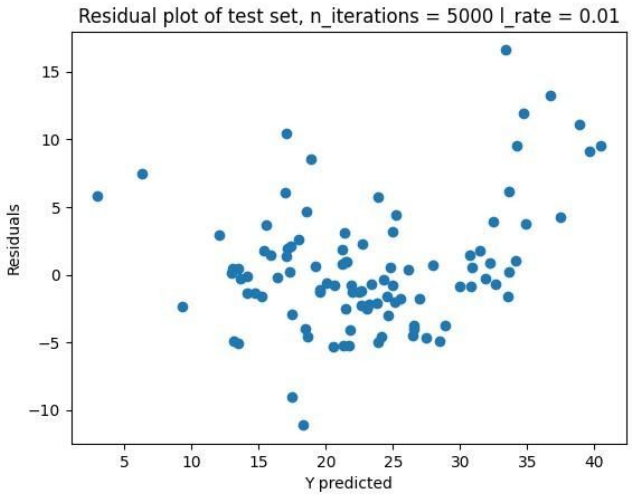
Średni procent trafień wynosi 72.6%, natomiast średni czas treningu wynosi ok. 6.1 sekundy.



Procent trafień wynosi 78%, natomiast czas treningu wynosi ok.6.1 sekundy.

Przypadek 4

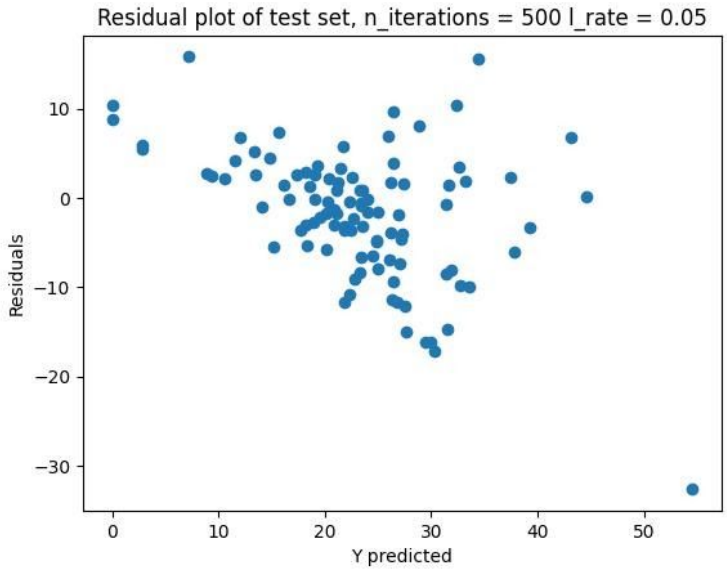
Średni procent trafień wynosi 69%, natomiast średni czas treningu wynosi ok. 31.63 sekundy.



Procent trafień wynosi 72%, natomiast czas treningu wynosi ok. 31 sekund.

Przypadek 5

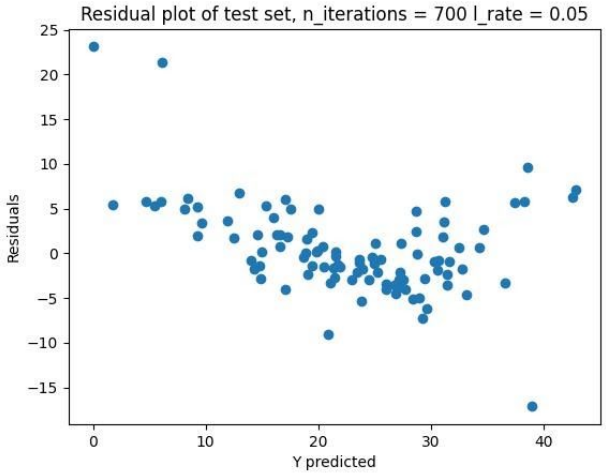
Średni procent trafień wynosi 50%, natomiast średni czas treningu wynosi ok. 3.07 sekundy.



Procent trafień wynosi 52%, natomiast czas treningu wynosi ok. 3.08 sekundy.

Przypadek 6

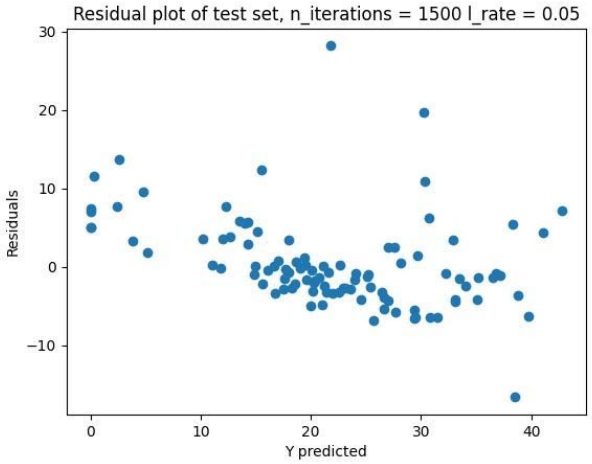
Średni procent trafień wynosi 63.2%, natomiast średni czas treningu wynosi ok. 4.47 sekundy.



Procent trafień wynosi 74%, natomiast czas treningu wynosi ok.4.38 sekundy.

Przypadek 7

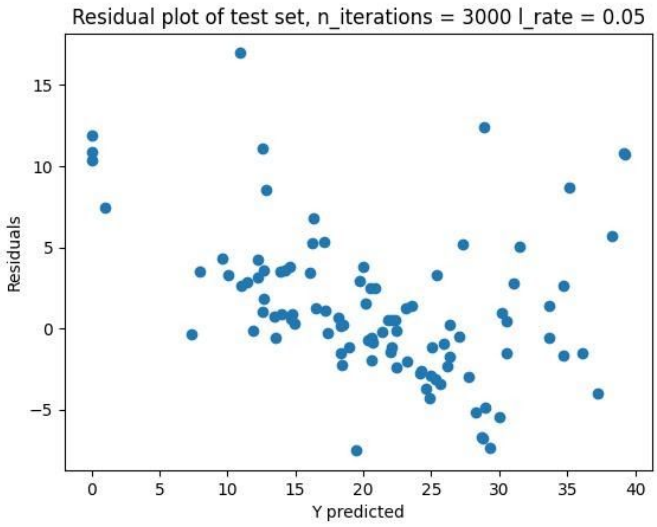
Średni procent trafień wynosi 60.6%, natomiast średni czas treningu wynosi ok. 9.45 sekundy.



Procent trafień wynosi 65%, natomiast czas treningu wynosi ok. 9.51 sekundy.

Przypadek 8

Średni procent trafień wynosi 63.8%, natomiast średni czas treningu wynosi ok. 18.34 sekundy.



Procent trafień wynosi 69%, natomiast czas treningu wynosi ok. 18.28 sekundy.

Wnioski, przemyślenia, ew. możliwe usprawnienia

Zauważmy, że dla przypadków z l_rate = 0.01 (czyli przypadków 1-4) mamy wzrost średniego procentu trafień aż do liczby iteracji równej 500 (przypadek 2), a następnie obserwujemy spadek tejże wartości. Można to uzasadnić faktem, iż początkowo mieliśmy niedotrenowany model, a po osiągnięciu maksimum średniego procentu trafień, zwiększenie liczby iteracji skutkowało przetrenowaniem sieci.

Dla przypadków z l_rate = 0.05 (czyli przypadków 5-8) obserwujemy wzrost średniego procentu trafień aż poziomu sześćdziesięciu-sześćdziesięciu kilku procent.
Może to wynikać z wpływu większej wartości parametru l_rate.

Możemy również zaobserwować inny wpływ parametru l_rate.
Spójrzmy na przypadki 2 oraz 5. Widzimy, że średni procent trafień zmniejszył się, kiedy zwiększyliśmy wartość parametru l_rate z 0.01 do 0.05 (liczba iteracji wynosi w obydwu przypadkach 500).

Zagadnienie, którym się zajmowaliśmy, miało charakter ciągły (gdyż możemy przyjąć, że wartości cen, które chcieliśmy przewidzieć, są zmiennymi ciągłymi).
Twierdzimy, że to utrudniało osiągnięcie bardzo wysokich średnich procentów trafień.

Możliwym usprawnieniem sieci jest wykonanie większej liczby testów (dla różnych parametrów sieci), aby znaleźć takie ich wartości, które dają lepszy średni procent trafień.

Zwięzła instrukcja dla użytkownika programu

Na początku należy zainstalować biblioteki języka Python: numpy, pandas, matplotlib, sklearn, statsmodels.
Aby zmienić parametry sieci neuronowej (czyli liczbę iteracji n_iterations oraz parametr l_rate), wystarczy zmienić wartości tych zmiennych w linijkach 88. oraz 89. programu perceptron.py.
Aby uruchomić program w Linuxie, należy w terminalu napisać komendę
python3 perceptron.py