Dokumentacja projekt PSZT

Autorzy: Piotr Żelazko, Daniel Bigos, Marcin Dziedzic

# Treść zadania

TT.NN.6

Stwórz sieć neuronową, która przewidzi cenę̨ sprzedaży domu na podstawie jego parametrów.

# Zestawienie kluczowych decyzji projektowych

Pierwszą kluczową decyzją projektową był wybór języka implementacji. Po zgłębieniu wiedzy w głównej problematyce związanej z sieciami neuronowymi dowiedzieliśmy się, że powinniśmy zwrócić uwagę na szybkość działania algorytmów trenowania SN. Decyzja w tym przypadku była dość oczywista – C++.

Drugim problemem było określenie interfejsu programu oraz prezentacji wyników jego działania. Biorąc pod uwagę konieczną liczbę testów konfiguracji SN przywiązaliśmy dużą wagę do optymalizacji procesu uruchomienia programu. Stwierdziliśmy więc, że implementacja GUI nie jest wskazana, a program powinien być aplikacją konsolową przyjmującą argumenty uruchomienia programu. Prezentacja wyników uczenia się sieci zapisywana jest do pliku wskazanego przez użytkownika.

Doszliśmy do wniosku, że:

Aby wybrać najlepszą konfigurację SN należy wyselekcjonować:

* topologię (liczbę warstw ukrytych i neuronów w każdej z nich.),
* funkcję aktywacji,
* współczynnik uczenia,
* rozmiar paczki danych (branych pod uwagę przy propagacji wstecznej)

Aby proces uczenia ukazał najlepsze rezultaty należy określić:

* liczbę epok,
* stosunek danych testowych do treningowych,
* liczby wątków (równolegle uruchamianych procesów uczenia)

Wszystkie ww. założenia zostały przez nas zaimplementowane.

# Instrukcja użytkownika (Instrukcję dla użytkownika wystarczająca do uruchomienia programu i jego głównych funkcji)

Program uruchamiany jest z wiersza poleceń.

Dostępne są następujące parametry programu:

Informacyjne:

-h [ --help ] Wyświetlona zostaje wiadomość pomocnicza zawierająca opis i listing argumentów wywołania.

-v [ --version ] Wyświetlona zostaje aktualna wersja programu.

Uruchomieniowe:

-m [ --mode ] arg Określenie trybu uruchomieniowego programu.

-d [ --input data ] arg Określenie nazwy pliku z danymi wejściowymi.

-r [ --result-path ] arg (=./) Określenie ścieżki do katalogu w którym będą serializowane SN

-l [ --logger ] arg (=./logger.csv) Określenie pliku do którego będą zapisywane rezultaty działania programu.

-e [ --epoch ] arg Określenie ilości epok uczenia się SN.

-t [ --topology ] arg Określenie topologii warstw ukrytych SN

-c [ --eta ] arg Określenie współczynnika uczenia się SN

-n [ --neural-net ] arg Wskazanie pliku, w którym znajduje się zapisana sieć neuronowa.

-p [ --pack ] arg Określenie wielkości paczki danych branych pod uwagę przy propagacji wstecznej

-f [ --function ] arg Określenie funkcji aktywacji.

-b [ --tolerance ] arg (=-1) Tryb treningu - Stosunek danych testowych do treningowych w procentach.

Tryb testowy – tolerancja błędu

-w [ --threads ] arg (=0) Określenie liczby wątków równolegle uruchamianych podczas procesu uczenia.

-k [ --k-fold ] arg (=0) Określenie współczynnika k w procesie k-fold

-q [ --timer ] arg (=0) Określenie czasu treningu SN

Dostępne są następujące tryby uruchomienia [--mode]:

1 - train

2 - train with timer

3 - test

4 - train with k-fold

Dostępne są następujące topologie sieci neuronowych [--topology] (liczba neuronów w warstwach ukrytych):

0 - 5,5

1 - 10, 10

2 - 15, 5

3 - 15, 10

4 - 20, 10

5 - 20, 20

6 - 30, 15

7 - 5, 5, 5

8 - 10, 10, 10

9 - 500

10 - 1000

11 - 1500

12 - 2000

Dostępne są następujące funkcje aktywacji:

0 - step

1 - fast sigmoid

2 - logistic

3 - tanh

4 - specific algebraic

Wymagane prametry dla poszczególnego trybu uruchomienia:

- TRAIN

- epoch

- pack

- function

- eta

- topology

- percentage

- threads

- TRAIN WITH TIMER

- pack

- function

- eta

- topology

- percentage

- timer

- TEST

- pack

- neural-net

- percentage

- K-FOLD

- epoch

- pack

- function

- eta

- topology

- k

Szybki przewodnik:

Odpalając projekt po raz pierwszy zalecane jest, aby użyć parametru –h. Po zapoznaniu się z instrukcją pomocniczą jesteśmy gotowi do przeprowadzania eksperymentów.

Aby rozpocząć pierwszy trening sieci neuronowej ustawiamy tryb treningu ( -m 1) oraz podajemy wszystkie parametry.

Parametry –e, –c, –p, –f, –b, –w możemy podać w formie listy. Wówczas program uruchomi równoległy trening sieci neuronowych wszystkich kombinacji parametrów.

Wynik uczenia się sieci (MSE) możemy przeglądać w pliku zwanym loggerm.

# Opis struktury programu

Program składa się z 5 głównych modułów:

* house – moduł umożliwiający zmapowanie danych wejściowych na struktury używane w implementacji. Klasy:
  + House
  + NormalizedValuesHouse
* neural network – moduł zawierający klasy opisujące strukturę sieci neuronowej oraz jej komponenty
  + neurons – zbiór klas reprezentujących neurony oraz ich komponenty
    - BiasNeuron
    - HiddenLayerNeuron
    - InputNeuron
    - Neuron
    - Neuron
    - OutputNeuron
    - Synapse
  + ActivationFunctionsBank
  + NeuralNetwork
  + NeuralNetworkTopology
* Program – moduł odpowiedzialny za odpowiednia inicjalizacje programu zważając na tryb uruchomienia
  + program\_initialization
    - ProgramInitializer
  + Program
  + TestProgram
  + TrainAndTestProgram
  + TraninProgram
* Progress – moduł odpowiedzialny za wyświetlania paska postępu nauki sieci neuronowej
  + ProgressStatus
  + ProgressStatusBar
  + ProgressStatusInfo
  + ProgressStatusManager
* klasy wspomagające procesy I/O
  + FileReader
  + RandomNumberGenerator
  + Serializator
  + TrainingDataFactory
* Tests – testy jednostkowe klas
  + SampleTests
  + test\_main

# Wnioski dotyczące osiągniętych rezultatów

Po udanej implementacji sieci neuronowej przystąpiliśmy do wybrania najlepszych parametrów sieci. Początkowo niektórych z nas zdziwił fakt, iż nie udało nam się zmniejszyć błędu średniokwadratowego (dalej MSE) do wartości piątego rzędu po przecinku w notacji wykładniczej. Przy zastosowanym mapowaniu zakresu cen domów z 0 - 10,000,000 do 0-1 uzyskaliśmy dokładność wyceny rzędu 100,000 zł. Zastanawialiśmy się dlaczego tak ciężko przełamać tą granicę.

Przeanalizowaliśmy dane wsadowe narzędziem SAS

../../Downloads/Histogram-results.pdf../../Downloads/Box%20Plot-results.pdf

Z rozkładu cen domów wynika, że większość z nich oscyluje w granicach 500,000 ze stosunkowo niewielkim odchyleniem. Na tym etapie właśnie to potrafiła wskazać nasza sieć.

Przystąpiliśmy do wyszukania najlepszej topologii sieci.

Program został uruchomiony z następującymi parametrami, co zapewniło, że każda sieć uczyła się przez taką samą jednostkę czasu:

-m 2 -d kc\_house\_data.csv -r ./out -l ./topologiesOneMinute.csv -p 1 -f 2 -c 0.05 -t 8 -b 5 -w 2 -q 1

Część wyników została przedstawiona na wykresie. W tym przypadku mse to średnia z dwóch prób dla tej samej topologii. Zgodnie z przewidywaniami najlepiej wypadła topologia o architekturze warstw ukrytych 40.

../../../Downloads/Bar%20Chart-results.pdf

*Przy zbyt malej liczbie neuronów sieć może nie mieć na tyle „pojemności” aby poprawnie ocenić sytuacje decyzyjną. Przy zbyt dużej liczbie węzłów tracimy czas obliczeniowy, gdyż niektóre z neuronów są niewykorzystywane. Zbadaliśmy też, że nasz system decyzyjny nie wymaga użycia większej ilości niż jednej warstwy ukrytej (funkcja decyzyjna nie jest na tyle skomplikowana).*