Bartosz Budnik 8.06.2016 r.

Piotr Komar

Sebastian Grzesiak

Michał Szymański

Mateusz Walczak

***Zastosowania informatyki w gospodarce  
Projekt***

***Deeplearning – ner (nazwy własne)***

Prowadzący:

Dr inż. Tomasz Walkowiak

Politechnika Wrocławska

Spis treści

[1. Wstęp 3](#_Toc453004618)

[2. Instalacja środowiska 3](#_Toc453004619)

[2.1. Wykorzystane środowisko 3](#_Toc453004620)

[2.2. Instalacja 3](#_Toc453004621)

[2.3. Inne wersje 5](#_Toc453004622)

[2.4. Rozwiązywanie problemów 6](#_Toc453004623)

[3. Rezultat prac nad projektem 7](#_Toc453004624)

[4. Instrukcja obsługi 10](#_Toc453004625)

[5. Podsumowanie 12](#_Toc453004626)

[6. Spis ilustracji, listingu i tabel 12](#_Toc453004627)

[7. Literatura 13](#_Toc453004628)

# Wstęp

Celem tego projektu było napisanie programu, który będzie w stanie zidentyfikować nazwy własne w tekście oraz, w wersji bardziej zaawansowanej, dodatkowe informacje o wyrazach czy związkach frazeologicznych. Program miał nauczyć się identyfikować wyrazy z dostarczonej bazy i przekazywać informację użytkownikowi, w jakim stopniu nauczył się poprawnie ustalać nazwy własne. Dodatkowo po nauczeniu programu miała istnieć możliwość podawania dowolnych nazw w postaci parametru. W związku z dużą ilością danych, projekt oparty był o GPU, aby przyspieszyć działanie.

W celu napisania programu, należało wykorzystać rozwiązania z dziedziny deeplearning’u. Jest to gałąź uczenia maszynowego oparta o struktury i algorytmy, które modelują dane na wysokim poziomie abstrakcji przez wiele warstw procesujących, o złożonej strukturze składającej się z wielu nieliniowych transformacji[1].

# Instalacja środowiska

## Wykorzystane środowisko

Projekt został napisany w języku programowania python w wersji 2.7. Program działa w oparciu o framework Keras, korzystający z silnika Tensorflow.

TensorFlow to rozwiązanie od Google typu open source przeznaczone do obliczeń numerycznych w oparciu o CPU lub GPU[2]. Framework ten powstał z myślą o machine learning’u, celem przygotowania sieci neuronowych do wykrywania i rozszyfrowywania wzorców oraz korelacji. Keras z kolei to framework z bibliotekami modularnych sieci neuronowych przeznaczony do projektów machine i deep- learning’owych, umożliwiający korzystanie z silnika innych framework’ów, takich jak Theano czy TensorFlow[3].

Działanie programu zostało sprawdzone na systemie Linux, wersjach Fedora oraz Ubuntu. Do instalacji konieczne będzie połączenie z siecią oraz karta graficzna obsługująca CUDA. Wymagane również będzie konto na stronie Nvidia Developer w celu pobrania dodatkowych bibliotek cudNN[4].

## Instalacja

Instrukcja instalacji zakłada, że w systemie Linux nie ma poprzednich wersji TensorFlow, CUDA lub cudNN. Instalacja została wykonana na „czystym” systemie, więc obejmuje również podstawowe kroki. Wszystkie etapy przedstawione w tej instrukcji, zostały wykonane w takiej samej kolejności jak w środowisku testowym. Poniższe komendy zostały wykonane na systemie Linux 64-bit v14.04. Należy pamiętać, że w przypadku różnicy w wersji oprogramowania pobranego z Internetu, trzeba zmodyfikować odpowiednio niektóre komendy.

Instalację środowiska rozpoczęto od aktualizacji sterowników karty graficznej.

sudo apt-get install nvidia-340

Listing 1 Komenda instalująca sterownik graficzny.

W tym wypadku została zainstalowana wersja 340, jednak można zainstalować inną wersję obsługującą CUDA 7.5. Listę aktualnych wersji sterownika karty graficznej można znaleźć na stronie internetowej <http://www.nvidia.com/object/unix.html> (Stan na dzień 02.05.2016 r.).

Kolejnym krokiem była instalacja PIP.

sudo apt-get install python-pip python-dev

Listing 2 Komenda instalująca PIP.

Następnie instalacja TensorFlow z obsługą GPU; wymaga: Python 2.7, CUDA toolkit 7.5, CuDNN v5. Instalacja ostatnich dwóch elementów od Nvidii zostanie przedstawiona w dalszej części.

sudo pip install --upgrade https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow-0.8.0-cp27-none-linux\_x86\_64.whl

Listing 3 Komenda instalująca TensorFlow.

Są również dostępne inne wersje instalacyjne, jednak zostaną one przedstawione w dalszej części, jako opcjonalne zmiany. W projekcie została wykorzystana komenda umieszczona powyżej.

Po tym kroku zainstalowano CUDA 7.5 oraz cudNN 5.0. Aby pobrać, należy się zarejestrować na stronie <https://developer.nvidia.com/> (Stan na dzień 02.05.2016 r.). Najpierw zainstalowano CUDA 7.5 pobrane ze strony <https://developer.nvidia.com/cuda-downloads> (Stan na dzień 02.05.2016 r.). Należy wybrać kolejno:

Linux → x86\_64 → Ubuntu → 14.04 → deb (local)

Pobrany został plik **cuda-repo-ubuntu1404-7-5-local\_7.5-18\_amd64.deb** ważący w przybliżeniu 1.9 GB. Instalacja CUDA przebiegła w następujący sposób.

sudo dpkg -i cuda-repo-ubuntu1404-7-5-local\_7.5-18\_amd64.deb

sudo apt-get update

sudo apt-get install cuda

Listing Komendy potrzebne do instalacji CUDA.

Na stronie internetowej, z której pobrano ten plik znajduje się również kopia tych komend wzbogacona o obszerniejszy opis w pliku .pdf Installation Guide for Linux po adresem <http://developer.download.nvidia.com/compute/cuda/7.5/Prod/docs/sidebar/CUDA_Installation_Guide_Linux.pdf> (Stan na dzień 02.05.2016 r.), wraz z informacjami o rozwiązaniach możliwych do wystąpienia problemów. W przypadku tej instalacji ten pakiet nie sprawił problemów.

Kolejnym krokiem była instalacja cudNN, które jest dostępne pod adresem <https://developer.nvidia.com/cudnn> (Stan na 02.05.2016 r.). Po kliknięciu przycisku *Download*, wyświetlona została krótka ankieta, na którą można było odpowiedzieć. Następnie po kliknięciu *Proceed to download* i zaznaczniu opcji „*I Agree To the Terms…”* wyświetlona została lista z wyborem odpowiedniej wersji. Wybrano *cuDNN v5 Library for Linux*. Pobrany został plik **cudnn-7.5-linux-x64-v5.0-rc** ważący w przybliżeniu 75 MB, który wypakowano i skopiowano, jak wskazują komendy.

tar xvzf cudnn-7.5-linux-x64-v5.0-rc.tgz

sudo cp cudnn-7.5-linux-x64-v5.0-rc/cudnn.h /usr/local/cuda/include

sudo cp cudnn-7.5-linux-x64-v5.0-rc/libcudnn\* /usr/local/cuda/lib64

sudo chmod a+r /usr/local/cuda/lib64/libcudnn\*

Listing Komendy potrzebne do instalacji cudNN.

Powyższe komendy zakładają, że CUDA została zainstalowana w **/usr/local/cuda**. W przypadku instalacji w innej lokalizacji należy odpowiednio zmodyfikować.

Ostatnim krokiem było ustawienie **LD\_LIBRARY\_PATH** i **CUDA\_HOME** do ***~/.bashrc***. Wykonano ten krok korzystając z komend.

export LD\_LIBRARY\_PATH="$LD\_LIBRARY\_PATH:/usr/local/cuda/lib64"

export CUDA\_HOME=/usr/local/cuda

Listing Komenda ustawiająca zmienne środowiskowe.

W tym kroku również należy zmodyfikować komendy, jeśli lokalizacja CUDA jest inna, aniżeli ta założona w instalacji.

Ten krok kończy instalację. Potem sprawdzono czy TensorFlow został prawidłowo zainstalowany poprzez wykonanie następującego kroku.

python

>>> import tensorflow as tf

Listing Weryfikacja poprawności instalacji TensorFlow poprzez import bibliotek.

Ten etap zweryfikował poprawną instalację, więc można było wykonać kolejny krok jakim było uruchomienie testowego programu, dostarczonego wraz z framework’iem TensorFlow.

python /usr/local/lib/python2.7/disst-packages/tensorflow/models/image/mnist/convolutional.py

Listing Komenda uruchamiająca przykładowy program.

Program prawidłowo się uruchomił i testowa sieć uczyła się z domyślnego zbioru. Krok ten zakończył instalację środowiska.

## Inne wersje

TensorFlow jest elastycznym framework’iem. Umożliwia pracę na *python’ie 2.7–3.5*, z wykorzystaniem CPU lub GPU, jak również pracę na Linux’ie I Mac’u, (z wyłączniem GPU, dla systemów Mac OS X). Poniżej przedstawione komendy, które można zmienić w pierwszych krokach instalacyjnych (tylko dla GPU), z zaznaczeniem, że nie były testowane.

Python 3.4:

sudo apt-get install python3-pip python3-dev

Listing Komenda instalacji python 3.

TensorFlow, dla Python 3.4

sudo pip3 install --upgrade https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow-0.8.0-cp34-cp34m-linux\_x86\_64.whl

Listing Komenda instalacji TensorFlow, dla python 3.4.

## Rozwiązywanie problemów

W tym miejscu znajdują się sugestie co do możliwych napotkanych problemów przy instalacji środowiska. Opisane zostały jedynie te problemy na które osoby pracujące nad projektem się natknęły.

* Pomimo prawidłowej instalacji CUDA nie jest wykrywana przez TensorFlow.

Tutaj należy zacząć od sprawdzenia czy prawidłowo dodane zostały informacje LD\_LIBRARY\_PATH i CUDA\_HOME. Może się tak zdarzyć, że nie. Należy wtedy jedną z dwóch komend wyświetlić zmienne środowiskowe i zweryfikować poprawność dodania.

printenv

echo $PATH

Listing Komendy wyświetlające zmienne środowiskowe.

Pierwsza z nich wyświetla wszystkie zmienne środowiskowe, a druga tylko PATH. Ponowne dodanie zmiennych tym razem już z prawidłową lokalizacją powinno rozwiązać problem.

* Zmienne środowiskowe znikają po zamknięciu terminala.

Ten problem wymaga utrwalenia tych zmiennych poprzez dodanie do *~/.bashrc* informacji o tych zmiennych i ponowne wpisanie komendy.

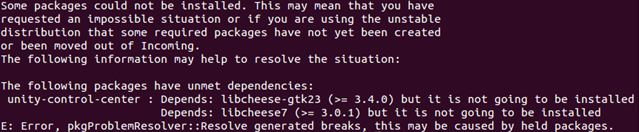
LD\_LIBRARY\_PATH=/usr/local/cuda/lib64:$LD\_LIBRARY\_PATH

Listing Dodanie zmiennej środowiskowej.

Wcześniej prawdopodobnie zmienne nie zostały dodane do *~/.bashrc*.

* Problem z „zależnościami” przy instalacji CUDA.

Przykładowy komunikat:



Rysunek Problem z zależnościami przy instalacji plików CUDA.

Jest to informacja o problemie z samym systemem Linux, a nie z instalacją. Należy wtedy wykonać następującą komendę.

sudo apt-get -f install

Listing Komenda naprawiająca błędne zależności.

Parametr –f oznacza naprawę. Komenda ta przeprowadzi próbę naprawy błędnych zależności. Następnie, dla pewności, można przeprowadzić prostą weryfikację, następującymi komendami.

sudo apt-get update

sudo apt-get upgrade

Listing Komendy sprawdzające poprawność repozytorium i cache.

Te komendy sprawdzają czy repozytorium oraz cache są pozbawione błędów. To powinno rozwiązać problem.

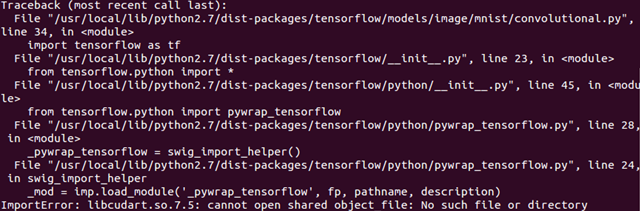
* Testowy program działa, ale wyświetla się o komunikat na początku o braku bibliotek cudNN.

Najprawdopodobniej wystąpił błąd przy instalacji. Należy sprawdzić lokalizację skopiowanych plików oraz zmienne środowiskowe czy prawidłowo CUDA została dopisana.

Nie należy się jednak przejmować tym problemem, ponieważ cudNN nie jest wymagane do poprawnego działania. Biblioteki te mają za zadanie m. in. przyspieszyć trenowanie sieci, więc można pominąć ten element środowiska, jeśli poprawienie lokalizacji skopiowanych plików nie pomaga.

* Problem z brakiem bibliotek CUDA przy próbie uruchomienia programu.

Możliwy błąd:



Rysunek Problem z importem bibliotek CUDA.

Ostatni wiersz:

*ImportError: libcudart.so.7.5: cannot open shared object file: No such file or directory*

Oznacza to, że gdzieś został popełniony błąd przy instalacji bibliotek CUDA. Należny ponownie prześledzić proces instalacji w poszukiwaniu ewentualnych błędów. *No such file or directory* sugeruje, że nie zostały podlinkowane lub zainstalowane niektóre elementy.

# Rezultat prac nad projektem

Projekt został zakończony z satysfakcjonującymi wynikami. Program określa nazwy własne z dokładnością na poziomie 70% - 75%. Zaimplementowane zostało rozróżnianie dużych i małych liter, interpretacja pojedynczych słów oraz wyrażeń złożonych z kilku słów.

W projekcie uczenie sztucznej sieci neuronowej odbywa się z wykorzystując model *Convolutional Layer*[5][6]. Jest to typ sieci *feed-forward*, gdzie połączenia pomiędzy jednostkami nie tworzą cykli oraz neurony są ułożone w podobny sposób jak w zwierzęcej korze wzrokowej, czyli indywidualne neurony pokrywają przestrzeń nakładając się, aby uzyskać lepszą reprezentację wartości. Za konfigurację *Convolution1D* odpowiedzialny jest fragment kodu:

Convolution1D(nb\_filter=nb\_filter, filter\_length=filter\_length, border\_mode='valid', activation='relu', subsample\_length=1)

Listing Funkcja Convolution1D.

Istotnym elementem projektu było właściwe przetworzenie danych, aby uzyskać informacje potrzebne do trenowania. W tym celu został napisany prosty kod przetwarzający dane. W osobnym pliku zapisywane były wyrazy wraz z zero-jedynkową informacją o tym, czy słowo jest nazwą własną czy nie. Znalezienie zależności dla innych własności, jak np. państwo okazało się bardzo ciężkie, w związku z czym obecna wersja programu przewiduje rozróżnianie tej informacji z rezultatem na poziomie 5%. Ogrom dostarczonych danych nie wymagał analizy ich całości, lecz fragmentu, jednak pomimo tego parametryzacja własności słów okazała się trudniejsza niż mogło się wydawać.

Tabela Przykład przefiltrowanych danych.

|  |  |  |
| --- | --- | --- |
|  | ***Wyraz*** | ***Nazwa własna*** |
| ***1*** | Szczecin | 1 |
| ***2*** | płaczliwy | 0 |
| ***3*** | Wadżrajany | 1 |
| ***4*** | klatka | 0 |
| ***5*** | Rugia | 1 |
| ***6*** | nyski | 0 |
| ***7*** | telewizor | 0 |

Powyższe dane prezentują proste kodowanie wyrazów, gdzie zawarta jest informacja o tym czy dane słowo stanowi nazwę własną czy nie. Surowe dane znajdują się w pliku *.txt*, przy czym każde słowo jest w nowym wierszu a informacja 0/1 występuje po słowie i jest oddzielona przecinkiem.

Kodowanie danych w programie wykorzystywało funkcję hashującą. Użytą funkcją kodowania słów do postaci przyjmowanej przez sieć neuronową jest funckja *one\_hot* udostępniona w frameworku Keras. Po podaniu do niej słowa lub zdania w wyniku otrzymujemy listę liczb całkowitych o rozmiarze słownika n. Jedyność nie jest zagwarantowana dlatego różne słowa mogą zostać zakodowane do takiej samej liczby.

keras.preprocessing.text.one\_hot(text, n, filters=base\_filter(), lower=True, split=" ")

Listing Funkcja one\_hot do kodowania posatci słów.

W polu *encoding* na poniższym rysunku pierwszy parametr to słowo a druga liczba reprezentuje cechę wielkości pierwszej litery słowa zakodowaną również poprzez funkcję *one\_hot* jako słowo 0 lub 1.



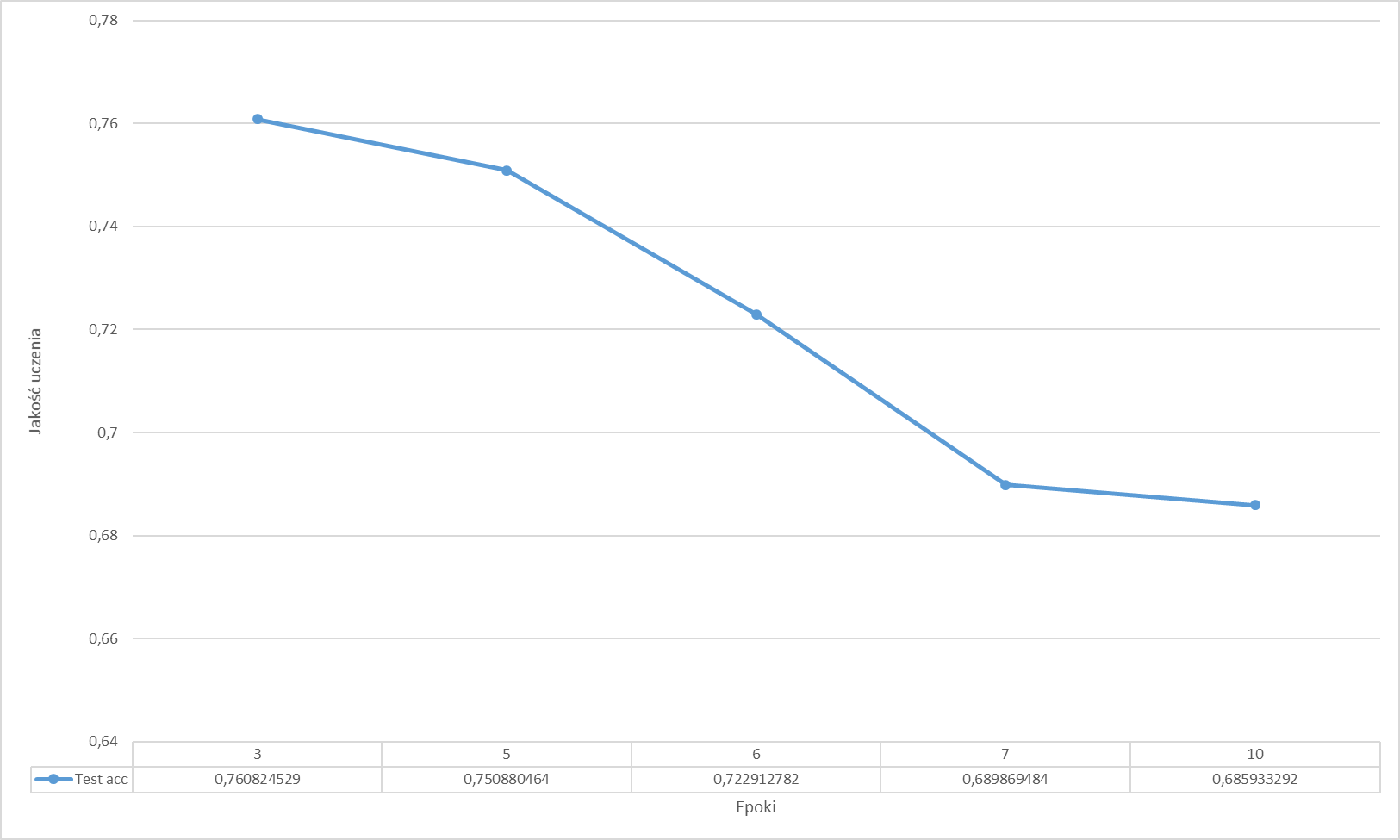
Kodowanie

Rysunek Przykład kodowania danych.

Wyniki jakości uczenia sieci neuronowej zależą w dużym stopniu od ilości epok, czyli powtórzeń uczenia się. Wyniki testu są z zakresu od 0 do 1. Prezentują się one następująco:

Tabela Wyniki uczenia.

|  |  |
| --- | --- |
| ***Epoki*** | ***Jakość uczenia*** |
| 3 | 0,760824529 |
| 5 | 0,750880464 |
| 6 | 0,722912782 |
| 7 | 0,689869484 |
| 10 | 0,685933292 |



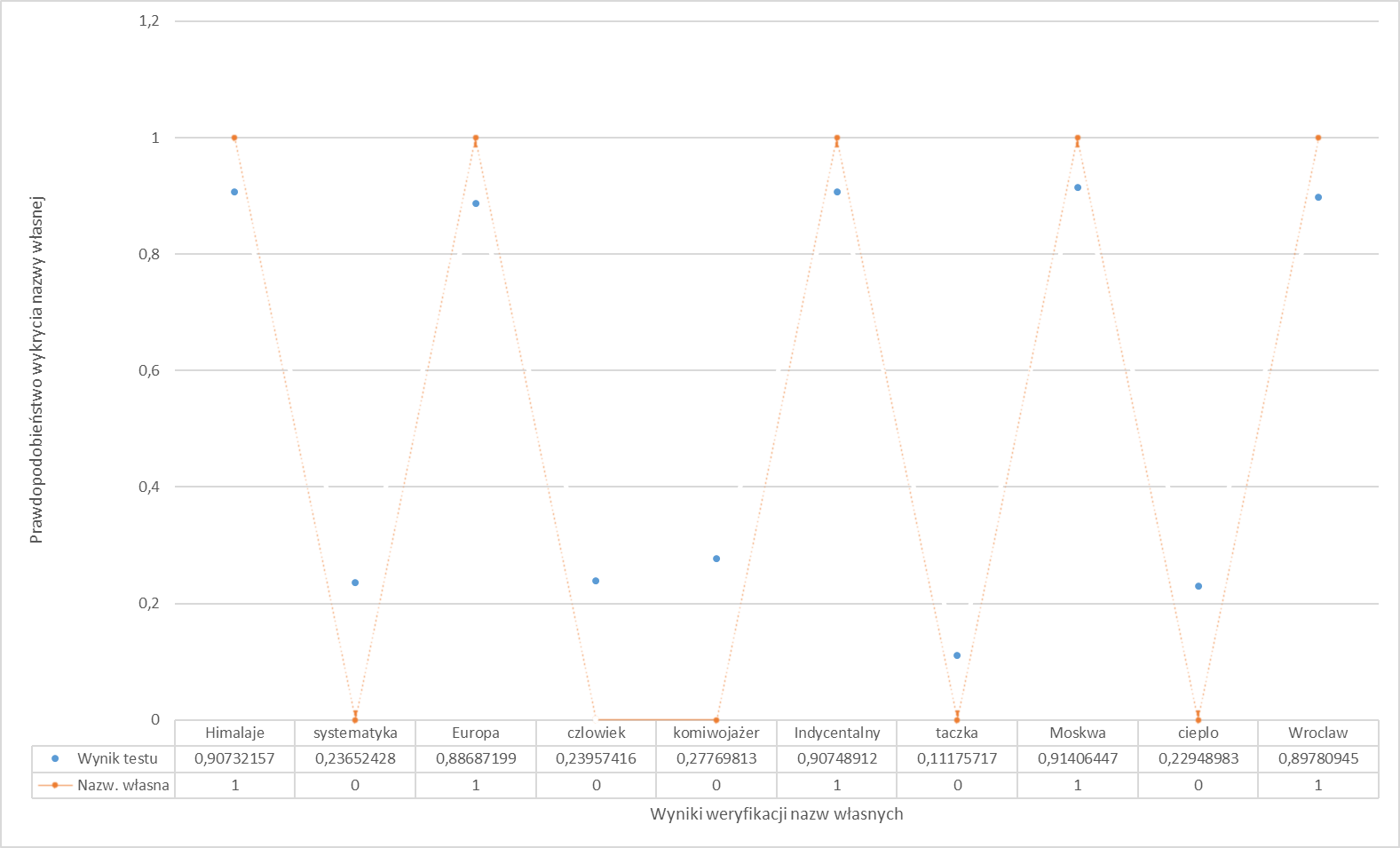
Rysunek Jakość uczenia.

Aby zaprezentować jakość wykrywania nazwa własnych zostały przeprowadzone proste testy. Poniżej zamieszczone zostaną rezultaty, dla dziesięciu przykładowych słów.

Tabela Uzyskane wyniki, dla 10 przykładowych wyrazów.

|  |  |  |  |
| --- | --- | --- | --- |
|  | ***Słowo*** | ***Uzyskany wynik*** | ***Nazwa własna*** |
| ***1*** | Wroclaw | 0,89780945 | Tak |
| ***2*** | cieplo | 0,22948983 | Nie |
| ***3*** | Moskwa | 0,91406447 | Tak |
| ***4*** | taczka | 0,11175717 | Nie |
| ***5*** | Indycentalny | 0,90748912 | Tak |
| ***6*** | komiwojażer | 0,27769813 | Nie |
| ***7*** | czlowiek | 0,23957416 | Nie |
| ***8*** | Europa | 0,88687199 | Tak |
| ***9*** | systematyka | 0,23652428 | Nie |
| ***10*** | Himalaje | 0,90732157 | Tak |

Wynik interpretuje się w następujący sposób. Im wartość jest bliższa jedynki, tym większe prawdopodobieństwo, że dany wyraz jest nazwą własną. Wyrazy nieistniejące, jak np. Indycentalny, wykrywane są jako nazwy własne. Na poniższym wykresie przedstawiono rezultaty w postaci graficznej.

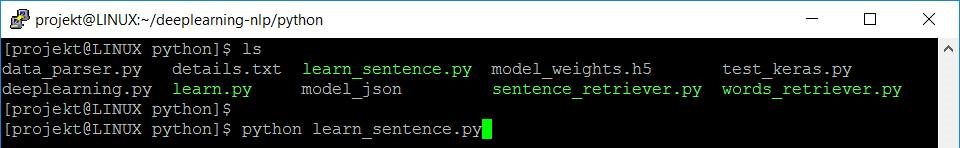


Rysunek Wykres przedstawiający porównanie uzyskanych wyników testu.

Niebieskie punkty oznaczają uzyskany wynik, natomiast pomarańczowe faktyczną informację o tym, czy dany wyraz jest nazwą własną lub nie jest. Pomarańczową linię przerywaną dorysowano, dla lepszego zobrazowania.

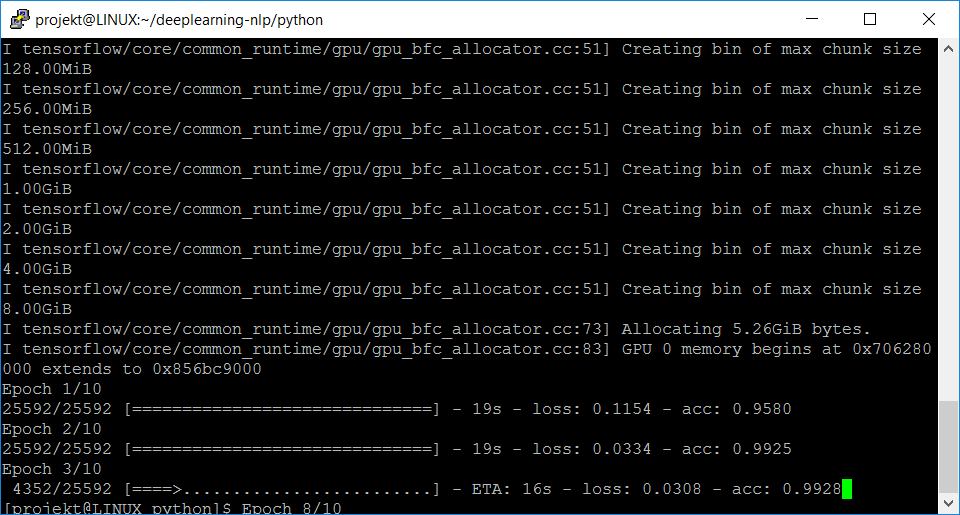
# Instrukcja obsługi

Program uruchamia się poziomu konsoli. Najpierw należy wytrenować sieć korzystając z komendy przestawionej na poniższym zrzucie ekranu.



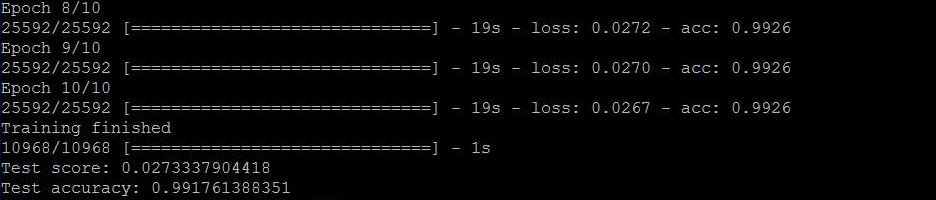
Rysunek Komenda odpowiedzialna za uczenie programu.

Rezultatem jest alokacja pamięci na karcie graficznej i rozpoczęcie procedury trenowania.



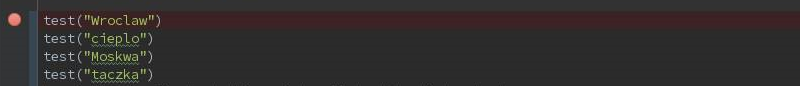
Rysunek Alokacja pamięci na GPU i uczenie.

Operacja kończona jest informacją o jakości trenowania.



Rysunek Informacja o ukończeniu procesu trenowania.

Dysponując tak wytrenowaną siecią można zacząć testy weryfikujące rozpoznawanie nazw własnych. Wykonuje się je wywołując metodę *test* i jako parametr podając żądane słowo. Na poniższym zrzucie ekranu uruchomiono program w trybie *debug* i zatrzymano działanie przy funkcji *test*. Testy po nauczeniu to podawanie słów jako parametr funkcji.



Rysunek Przykład testu wyrazów.

Po wykonaniu powyższych funkcji uzyskuje się następujące rezultaty.



Rysunek Rezultat wykonania funkcji z rysunku nr 6.

Odczytuje się je w następujący sposób. Szansa na to, że wyraz jest nazwą własną jest większa, im bliżej wartość *result* znajduje się jedynki. Na przykład słowo *Wroclaw* uzyskało wynik w przybliżeniu 0,9. Oznacza to, że jest duża szansa na to, iż jest to nazwa własna. Słowo *taczka* uzyskało wynik w przybliżeniu tylko 0,11, co za tym idzie jest to niewielka szansa na to, że jest to nazwa własna. Dodatkowo jest wyświetlane kodowanie danego wyrazu.

Powyższe działania w krótki sposób opisują podstawowe informacje o obsłudze programu i uzyskanych wynikach.

# Podsumowanie

Projekt pozwolił na poznanie charakterystyki programowania przy przetwarzaniu dużej ilości danych, w celu stworzenia modelu analizy informacji dostarczanych do programu, i ich automatycznej interpretacji. W tym wypadku rozpoznawaniu nazw własnych.

Program, który na podstawie wprowadzanych danych ma automatycznie decydować czy słowo, bądź związek frazeologiczny, jest nazwą własną osiąga skuteczność na poziomie 70% - 75%. Wykorzystywany jest przy tym procesor GPU, dzięki czemu czas potrzebny na trenowanie sieci został znacząco skrócony. Udało się zaimplementować wykrywanie dużej litry jak również odczyt, nie tylko wyrazów, ale i nazw składających się z kilku słów.

Projekt umożliwił poszerzenie umiejętności programowania w języku Python oraz zdobyto wiedzę na temat framework’ów Keras i TensorFlow.

# Spis ilustracji, listingu i tabel

[Tabela 1 Przykład przefiltrowanych danych. 8](#_Toc453004588)

[Tabela 2 Wyniki uczenia. 8](#_Toc453004589)

[Tabela 3 Uzyskane wyniki, dla 10 przykładowych wyrazów. 9](#_Toc453004590)

[Rysunek 1 Problem z zależnościami przy instalacji plików CUDA. 6](#_Toc453004592)

[Rysunek 2 Problem z importem bibliotek CUDA. 7](#_Toc453004593)

[Rysunek 3 Przykład kodowania danych. 8](#_Toc453004594)

[Rysunek 4 Jakość uczenia. 9](#_Toc453004595)

[Rysunek 5 Wykres przedstawiający porównanie uzyskanych wyników testu. 10](#_Toc453004596)

[Rysunek 6 Komenda odpowiedzialna za uczenie programu. 10](#_Toc453004597)

[Rysunek 7 Alokacja pamięci na GPU i uczenie. 11](#_Toc453004598)

[Rysunek 8 Informacja o ukończeniu procesu trenowania. 11](#_Toc453004599)

[Rysunek 9 Przykład testu wyrazów. 11](#_Toc453004600)

[Rysunek 10 Rezultat wykonania funkcji z rysunku nr 6. 12](#_Toc453004601)

[Listing 1 Komenda instalująca sterownik graficzny. 3](#_Toc453004602)

[Listing 2 Komenda instalująca PIP. 4](#_Toc453004603)

[Listing 3 Komenda instalująca TensorFlow. 4](#_Toc453004604)

[Listing 4 Komendy potrzebne do instalacji CUDA. 4](#_Toc453004605)

[Listing 5 Komendy potrzebne do instalacji cudNN. 4](#_Toc453004606)

[Listing 6 Komenda ustawiająca zmienne środowiskowe. 5](#_Toc453004607)

[Listing 7 Weryfikacja poprawności instalacji TensorFlow poprzez import bibliotek. 5](#_Toc453004608)

[Listing 8 Komenda uruchamiająca przykładowy program. 5](#_Toc453004609)

[Listing 9 Komenda instalacji python 3. 5](#_Toc453004610)

[Listing 10 Komenda instalacji TensorFlow, dla python 3.4. 5](#_Toc453004611)

[Listing 11 Komendy wyświetlające zmienne środowiskowe. 6](#_Toc453004612)

[Listing 12 Dodanie zmiennej środowiskowej. 6](#_Toc453004613)

[Listing 13 Komenda naprawiająca błędne zależności. 6](#_Toc453004614)

[Listing 14 Komendy sprawdzające poprawność repozytorium i cache. 6](#_Toc453004615)

[Listing 15 Funkcja Convolution1D. 7](#_Toc453004616)

[Listing 16 Funkcja one\_hot do kodowania posatci słów. 8](#_Toc453004617)

# Literatura

1. Ian Goodfellow, Yoshua Bengio, Aaron Courville (2016), „Deep Learning” MIT Press. Dostęp online: <http://www.deeplearningbook.org/>, Dostęp na dzień 24.05.2016 r.
2. <https://www.tensorflow.org/> - strona główna framework’u, typu *open source*, TensorFlow. Stan na dzień 02.05.2016 r.
3. <http://keras.io/> - strona główna framework’u Keras. Stan na dzień 02.05.2016 r.
4. <https://developer.nvidia.com/cudnn> - strona główna bibliotek cudNN. Stan na dzień 02.05.2016 r.
5. <https://en.wikipedia.org/wiki/Convolutional_neural_network> - definicja i opis jednego z typów sztucznej sieci neuronowej. Dostęp na dzień 03.06.2016 r.
6. <http://keras.io/layers/convolutional/> - opis funkcji *Convolutional1D* oraz innych z grupy *Convolutional Layers*. Dostęp na dzień 03.06.2016 r.