|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| logowydzialu | Instytut Informatyki Politechniki Śląskiej  Zespół Mikroinformatyki i Teorii Automatów Cyfrowych  **Projekt BIAI** | | | logoii | |
| **Rok akademicki:** | **Rodzaj studiów\*: SSI/NSI/NSM** | **Przedmiot:** | | **Grupa:** | **Sekcja:** |
| 2016/2017 | SSI | BIAI | | BDIS | 3 |
| **Skład sekcji:** | Bartłomiej Gruba | **Prowadzący**:  OA/JP/KT/GD/BSz/GB | | GB | |
|  | Łukasz Kozień |
| *Sprawozdanie* | | | | | |
| Temat:  Nauka sieci neuronowej tworzenia prostych melodii  Adres repozytorium: <https://github.com/bartekgruba/biai> | | | | | |
| Data:  dd/mm/yyyy | | | 21/09/2017 | | |

1. *Temat projektu.*

Tematem naszego projektu jest stworzenie sieci neuronowej tworzącej proste melodie, które są zapisywane w formacie MIDI.

1. *Założenia projektu.*

1. Komputer przyjmuje pliki MIDI prostych utworów lub melodii.

2. Pliki MIDI są parsowane na łatwiej przyswajalny format pliku tekstowego

3. Sieć "uczy się" melodii dzięki LSTM (Long Short Term Memory) Andreja Karpathy aby następnie stworzyć plik tekstowy z melodią.

4. Plik tekstowy stworzony przez program jest parsowany na plik MIDI.

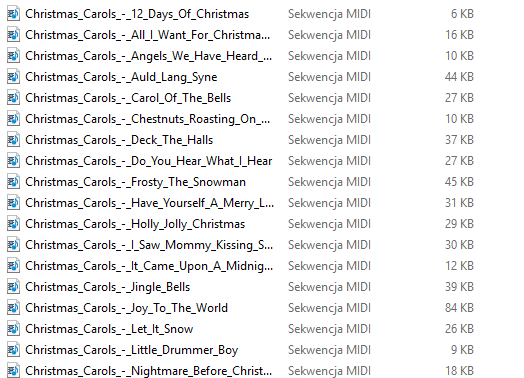
5. Plik MIDI jest odtwarzany.

1. *Analiza zadania.*

Podczas tworzenia programu, zastanawialiśmy się w jakiej technologii powinniśmy zaimplementować sieć i jej obsługę. Wahaliśmy się między implementowaniem sieci w Pythonie używając pakietów do obliczeń naukowych oraz użyciem gotowych bibliotek w Lua (bibliotece języka C, używanej także jako niezależny język). Po pierwotnych próbach implementacji w Lua, napotkaliśmy problemy techniczne, z powodu których ze względu na brak znajomości języka oraz małego dostępu do pomocnych źródeł zaczęliśmy zadecydowaliśmy, aby skorzystać z języka Python. Po utworzeniu modelu sieci program przyjmuje jako parametry folder plików midi, które są konwertowane aby były przyswajalne dla sieci. Po wykonaniu algorytmu tworzony jest wykonywalny plik o rozszerzeniu .mid oraz plik z parametrami sieci o rozszerzeniu .p.

1. *Struktura danych wejściowych/testowych*

Program uruchamiany jest z konsoli Pythona. Do uruchomienia skryptów niezbędne są biblioteki do funkcji naukowych takie jak: numpy, scipy, theano a także python-midi do otwierania plików midi. Danymi wejściowymi do uruchomienia programu są dowolne pliki midi. Ze względu na ograniczenia sprzętowe skupiliśmy się na małych plikach, aby nauczanie sieci wykonywało się w rozsądnym czasie.



Przykładowe pliki midi użyte do nauczania sieci wraz z ich rozmiarem

Pliki wejściowe powinny znajdować się w jednym folderze. Rozpoczęcie nauki odbywa się poprzez wywołanie funkcji loadPieces(ścieżkaDoFolderu) zaimplementowanej w skrypcie multi\_training, gdzie argumentem funkcji jest ścieżka do folderu, w którym znajdują cię pliki testowe. Przed rozpoczęciem nauki należy utworzyć folder o nazwie ‘output’ w lokalizacji, w której uruchamiana jest konsola Python’a. Funkcja trainPieces z pliku multi\_training zapisuje tam generowane pliki midi oraz pliki z parametrami do nauki. Brak folderu output spowoduje wygenerowanie błędu, a przez to żadne efekty nie zostaną uzyskane.

1. *Pliki użyte w programie*
2. model.py  
     
   Plik wykorzystujący bibliotekę theano do stworzenia sieci neuronowej LSTM (Long Short-Term Memory).

Składa się z dwóch klas:

* Model - klasa odpowiadająca za sieć neuronową wykorzystuje bibliotekę nauczania sieci theano\_lstm.

Najważniejsze funkcje:

* + def \_\_init\_\_(self, t\_layer\_sizes, p\_layer\_sizes, dropout=0) - funkcja inicjalizująca sieć neuronową, gdzie t\_layer\_sizes, p\_layer\_sizes to rozmiary powłok, a dropout - wartość dropoutu
  + def setup\_train(self) - funkcja ustawiająca parametry nauczania sieci neuronowej
  + def setup\_predict(self) - funkcja ustawiająca parametry przewidywania kolejnych stanów sieci
* PassthroughLayer - klasa odpowiadająca za pustą warstwę do otrzymania ostatecznego wyniku LSTM

1. multi\_training.py

Plik odpowiadający za nauczanie sieci.

Najważniejsze funkcje:

* def loadPieces(dirpath) - funkcja pobierająca próbki plików midi do nauki sieci, gdzie dirpath to ścieżka do folderu z plikami midi
* def trainPiece(model,pieces,epochs,start=0) - funkcja ucząca sieć, gdzie model to model sieci, pieces to pobrane wcześniej próbki muzyki próbki plików midi do nauki sieci, a epochs to ilość paczek próbek

1. midi\_to\_statematrix.py

Plik odpowiadający za konwertowanie plików midi.

Najważniejsze funkcje:

* def midiToNoteStateMatrix (midifile) - funkcja konwertująca pliki midi do nauki
* def noteStateMatrixToMidi (statematrix, name="example") - funkcja konwertująca stan sieci do pliku midi

1. data.py

Plik odpowiadający za obliczenia dla węzłów sieci neuronowej.

Najważniejsze funkcje:

* def startSentinel () – funkcja startująca sentinela, która oczekuje na informację czy rozmiar macierzy nie został przekroczony od funkcji noteSentinel()
* def noteSentinel () – funkcja pilnująca czy rozmiar macierzy nie został przekroczony

1. out\_to\_in\_op.py

Plik odpowiadający za tworzenie węzłów sieci neuronowej. Zawiera jedną klasę z dwoma funkcjami:

* def make\_node(self, state, time) – zwraca procedurę biblioteki theano tworzącą strukturę sieci
* def perform(self, node, inputs\_storage, outputstorage) – implementuje tworzenie sieci w języku Python

1. *Uruchamianie i testowanie*

Do uruchomienia potrzebne było zainstalowanie programu Python w dowolnej wersji. Z racji na to, że wersja 2.7 jest wciąż najczęściej używaną pomimo faktu, że isntnieje już nowsza, zdecydowaliśmy się właśnie na nią. Python 2.7 dostarcza zaawansowane biblioteki, służące do operowania na plikach muzycznych. Biblioteki te to theano (zawierająca logike LSTM) oraz numpy i scipy.

Aby użyć skryptów, należy uruchomić konsolę Python w folderze, w którym się one znajdują. Każdy skrypt należy przed użyciem zaimportować:

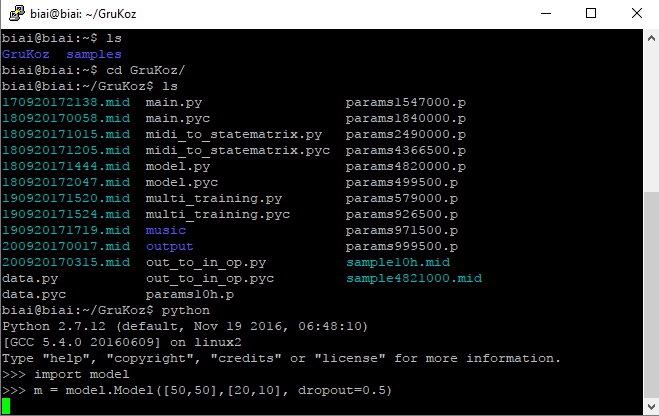
import model

import multi\_training

import main

Następnie do zmiennej m przypisujemy nasz model sieci:

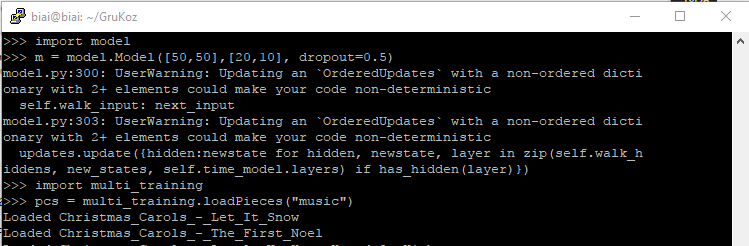
m = model.Model([50,50],[20,10], dropout=0.5)



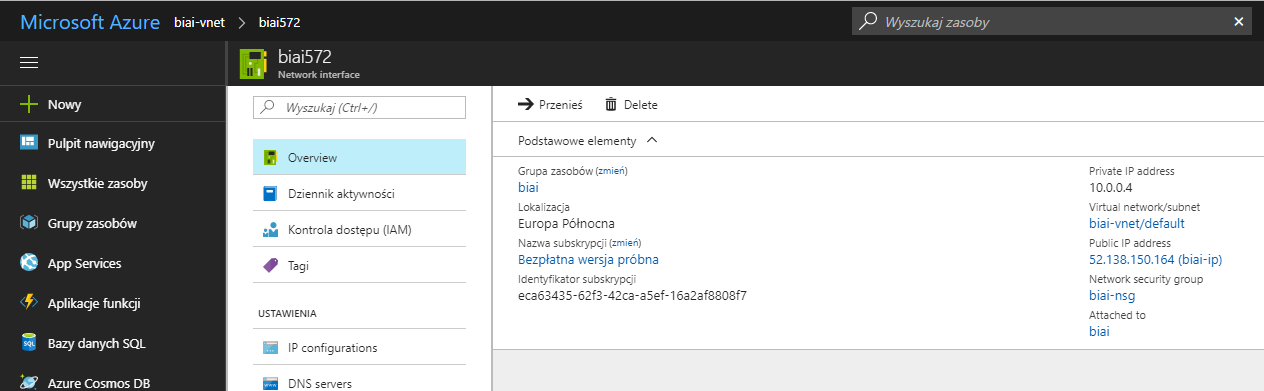
Liczby podane w kwadratowych nawiasach oznaczają rozmiary powłok sieci. Pierwsza z nich to powłoka czasu, druga tonów melodii. Parametr ‘dropout’ definiuje w jakim stopniu chcemy pomijać dane wejściowe przy uczeniu. 0 Oznacza w tym przypadku całkowity brak pomijania danych, 1 pomijanie niemalże stuprocentowe. Wyjaśnieniem dla obecności takiego parametru jest fakt, iż nie chcemy aby sieć tworzyła istniejące kompozycje, dostawała całkowicie gotowe wzorce. Chcemy aby tworzyła własne kompozycje, dlatego potrzebne jest pomijanie części danych wejściowych.

Po przypisaniu modelu sieci należy załadować pliki wejściowe, służące do nauki

pcs = multi\_training.loadPieces("music")



Z racji na to, że pliki wejściowe mogą być dowolnymi plikami o rozszerzeniu .midi, możliwości testowania sieci wydają się przeogromne. Z początku testowaliśmy uczenie wykorzystując tylko i wyłącznie twórczość Jana Sebastiana Bacha. Robiliśmy to w Windowsowym Powershell’u, który niestety działał bardzo opornie i próbki generowały się w zatrważającym tempie jednej na 2 ~ 3 godziny. Z racji na bardzo wolne uczenie się, próbki te nie były zadowalające, dlatego postanowiliśmy skorzystać z darmowej subskrypcji próbnej na portalu <https://portal.azure.com>

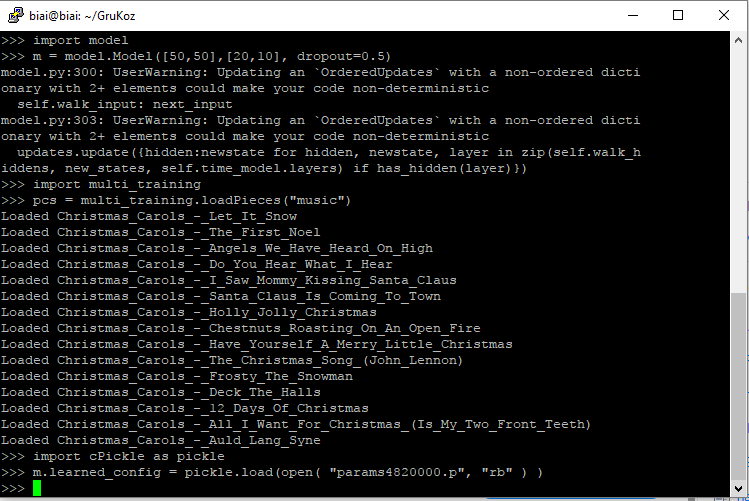


W ramach subskrypcji uzyskaliśmy dostęp do dodatkowej mocy obliczeniowej (czystych  
konsolowych Linux serwerów 16.04 oraz 17.04). Z nowymi możliwościami postanowiliśmy wypróbować naukę sieci na dwóch zupełnie różnych od siebie gatunkach muzycznych jakimi są kolędy i muzyka country.

Próbki generowały się niesamowicie szybko, co umożliwiło śledzenie postępów w nauce.  
Sieć po kilku godzinach nauki generowała już próbki nie tylko składające się z uderzania w całą klawiaturę fortepianu 20 razy na sekundę. Niestety nawet tak dobre środowisko okazało się zawodne z powodu restartu serwera w trakcie nauki. Szybko zorientowaliśmy się, że każdorazowe rozpoczynanie treningu od zera nie pozwoli nam uzyskać oczekiwanych rezultatów.

Na szczęście pomoc na rozwiązanie tego problemu odszukaliśmy w Internecie, a mianowicie wśród licznych bibliotek języka Python. Biblioteka cPickle udostępnia funkcję, umożliwiającą wczytanie konfiguracji modelu sieci (plik o rozszerzeniu .p) jako parametr modelu.

Terminal na stronie Microsoft Azure nie pozwalał wygenerować więcej niż jeden milion próbek. Proces przerywał się bez naszej ingerencji, dlatego zmuszeni byliśmy do skorzystania z programu Putty do połączenia z serwerem.Na szczęście połączenie to, pozwoliło swobodnie generować próbki bez jakichkolwiek niechcianych przerwań.

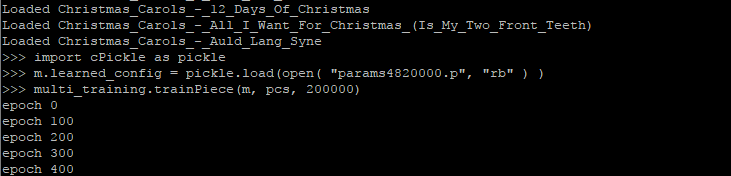


Ostatnim napotkanym przez nas problemem była niemożność zamknięcia okna programu Putty w trakcie nauki. Wraz z zamknięciem okna, proces się przerywał. Komputery działały z tego powodu dzień i noc ale zamierzone rezultaty zostały osiągnięte.

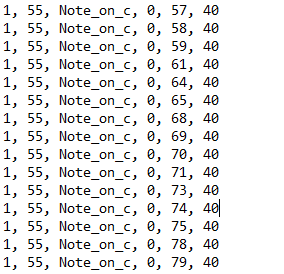
Po załadowaniu plików, można przystąpić nauki. Naukę rozpoczyna się za pomocą komendy:

multi\_training.trainPiece(m, pcs, 10000)

Parametrami tej funkcji są nasz model sieci (m), zmienna przechowująca próbki do nauki oraz ilość iteracji. Nowa próbka generowana jest co 500 iteracji, a składa się ona z pliku .mid oraz .p.



Pierwszy cykl nauczania trwał niespełna 20 minut i wyniki były bardzo niesatysfakcjonujące. Po wygenerowaniu pierwszego pliku midi, przeformatowywaliśmy go na plik tekstowy (za pomocą programu midicsv), aby obejrzeć pierwszy ‘wynik’ działania sieci. Na poniższym zdjęciu widać, że w jednej jednostce czasu (55) ‘naciskanych’ jest mnóstwo klawiszy na raz.



Pierwsze 1 oznacza numer ścieżki dźwiękowej. W przypadku, gdyby był to utwór polifoniczny, było by ich więcej.

55 to czas zabrzmienia dźwięku w milisekundach od rozpoczęcia utworu. Po tym widać, że w ciągu jednej jednostki czasu na raz nie naciskana jest cała klawiatura, a bardziej pojedyncze dźwięki.

Note\_on\_c, Note\_off\_c sygnalizuje że klawisz został ‘naciśnięty’ lub puszczony.

Wartości liczbowe to wysokość dźwięku, a ostatni argument oznacza długość, jaką dźwięk powinien trwać po naciśnięciu. Dla note\_off\_c jest to 0.

Wraz z mijającym czasem można było zauważyć progres. Wygenerowane pliki nie brzmiały już jakby ktoś naciskał całą dostępną klawiaturę fortepianu na raz. Niekontrolowanych dźwięków było coraz mniej, a w górnych rejestrach da się zauważyć formującą się melodię. Daleko jeszcze do właściwego brzmienia ale progres jest zauważalny. Dodatkowo nauka sieci tak kontrastującymi ze sobą gatunkami jak kolędy i muzyka country spowodowała, że na etapie kilkunastu godzin nauki widoczne są różnice w próbkach wynikowych.

Wszystkie próbki wynikowe dostępne są w załączonym linku do GitHuba. Podzielone są na wejściowe/wyjściowe oraz na gatunki. Pliki nazwane są formatem ddmmrrrrggss

* Dzien
* Miesiąc
* Rok
* Godzina
* Sekunda

1. *Wnioski*

Przed przystąpieniem do projektu zagadnienie sieci neuronowych było nam całkowicie obce. Ze względu na brak możliwości porównania z innymi projektami wykonywanymi przez nas wcześniej w toku studiów, wybrany temat okazał się dużo bardziej skomplikowany niż się spodziewaliśmy.

Po początkowej próbie implementacji w języku LUA (pochodna C) niestety zakończonej niepowodzeniem, zmieniliśmy podejście do problemu i spróbowaliśmy skorzystać z bibliotek języka Python. Jak szybko się okazało, była to dobra decyzja, ze względu na wsparcie dla LSTM zapewnione przez biblioteki Pythona. Wyniki uzyskane przez nas nie są idealne ale pozwalają zauważyć postępy nauki.

Mając na uwadze naszą muzyczną przeszłość (dyplom z fortepianu), temat, na który pomysł zaczerpnęliśmy z Internetu wydawał nam się doskonały. W dawnych jak i w dzisiejszych czasach tworzenie muzyki instrumentalnej uważane jest za przywilej ludzi wykształconych muzycznie, posiadających dobry słuch i ogromne doświadczenie w tej dziedzinie. Patrząc na to, że maszyna po odpowiednio długim treningu i odpowiednim materiale do nauki jest w stanie stworzyć utwór w stylu kompozytorów żyjących wieki temu rzuca zupełnie nowe spojrzenie na ten temat.

Wykonany projekt pozwolił nam zapoznać się z językiem Python, który coraz obszerniej używany jest w tworzeniu oprogramowania i aplikacji. Pozwolił nam również zpoznać się z platformą <http://platform.azure.com> udostępniającą serwery do wykorzystania w pracach naukowych jak i komercyjnie. Niewykluczone, że w przyszłości skorzystamy jeszcze z tej możliwości.

Program spełnia założenia zawarte w karcie projektu, a wyniki są zgodne z przewidywaniami. Wraz z wydłużaniem czasu uczenia sieci i zwiększaniem ilości danych do nauki, powstałe programy zauważalnie poprawiały swe brzmienie, harmonię, fakturę oraz melodykę. Do efektu końcowego jednak wciąż daleko dlatego zamierzamy kontynuować naukę sieci w przyszłości. Najistotniejszym problemem na przyszłość jest wzięcie pod uwagę, że sieć nie ma duplikować utworów, których się uczy. Musi na podstawie wzorców tworzyć własne kompozycje.

1. *Źródła*

* <http://www.hexahedria.com/2015/08/03/composing-music-with-recurrent-neural-networks/>
* <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>
* http://monik.in/a-noobs-guide-to-implementing-rnn-lstm-using-tensorflow/