
	<p>Instytut Informatyki Politechniki Śląskiej Zespół Mikroinformatyki i Teorii Automatów Cyfrowych <b>Projekt BIAI</b></p>			
Rok akademicki:	Rodzaj studiów*: SSI/NSI/NSM	Przedmiot:	Grupa:	Sekcja:
2016/2017	SSI	BIAI	BDIS	3
Skład sekcji:	Bartłomiej Gruba	<b>Prowadzący:</b> OA/JP/KT/GD/BSz/GB	GB	
	Łukasz Kozień			
<h2 style="text-align: center;">Sprawozdanie</h2>				
<p>Temat:</p> <h1 style="text-align: center;">Nauka sieci neuronowej tworzenia prostych melodii</h1> <p style="text-align: center;">Adres repozytorium: <a href="https://github.com/bartekgruba/biai">https://github.com/bartekgruba/biai</a></p>				
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.

## 2. 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.

## 3. 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 zdecydować, 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.

## 4. 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.

 Christmas_Carols_-_12_Days_Of_Christmas	Sekwencja MIDI	6 KB
 Christmas_Carols_-_All_I_Want_For_Christma...	Sekwencja MIDI	16 KB
 Christmas_Carols_-_Angels_We_Have_Heard_...	Sekwencja MIDI	10 KB
 Christmas_Carols_-_Auld_Lang_Syne	Sekwencja MIDI	44 KB
 Christmas_Carols_-_Carol_Of_The_Bells	Sekwencja MIDI	27 KB
 Christmas_Carols_-_Chestnuts_Roasting_On_...	Sekwencja MIDI	10 KB
 Christmas_Carols_-_Deck_The_Halls	Sekwencja MIDI	37 KB
 Christmas_Carols_-_Do_You_Hear_What_I_Hear	Sekwencja MIDI	27 KB
 Christmas_Carols_-_Frosty_The_Snowman	Sekwencja MIDI	45 KB
 Christmas_Carols_-_Have_Yourself_A_Merry_L...	Sekwencja MIDI	31 KB
 Christmas_Carols_-_Holly_Jolly_Christmas	Sekwencja MIDI	29 KB
 Christmas_Carols_-_I_Saw_Mommy_Kissing_S...	Sekwencja MIDI	30 KB
 Christmas_Carols_-_It_Came_Upon_A_Midnig...	Sekwencja MIDI	12 KB
 Christmas_Carols_-_Jingle_Bells	Sekwencja MIDI	39 KB
 Christmas_Carols_-_Joy_To_The_World	Sekwencja MIDI	84 KB
 Christmas_Carols_-_Let_It_Snow	Sekwencja MIDI	26 KB
 Christmas_Carols_-_Little_Drummer_Boy	Sekwencja MIDI	9 KB
 Christmas_Carols_-_Nightmare_Before_Christ...	Sekwencja MIDI	18 KB

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 się 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.

## **5. Pliki użyte w programie**

### a) 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

### b) 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

### c) 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

### d) 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

### e) 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

## 6. Uruchamianie i testowanie

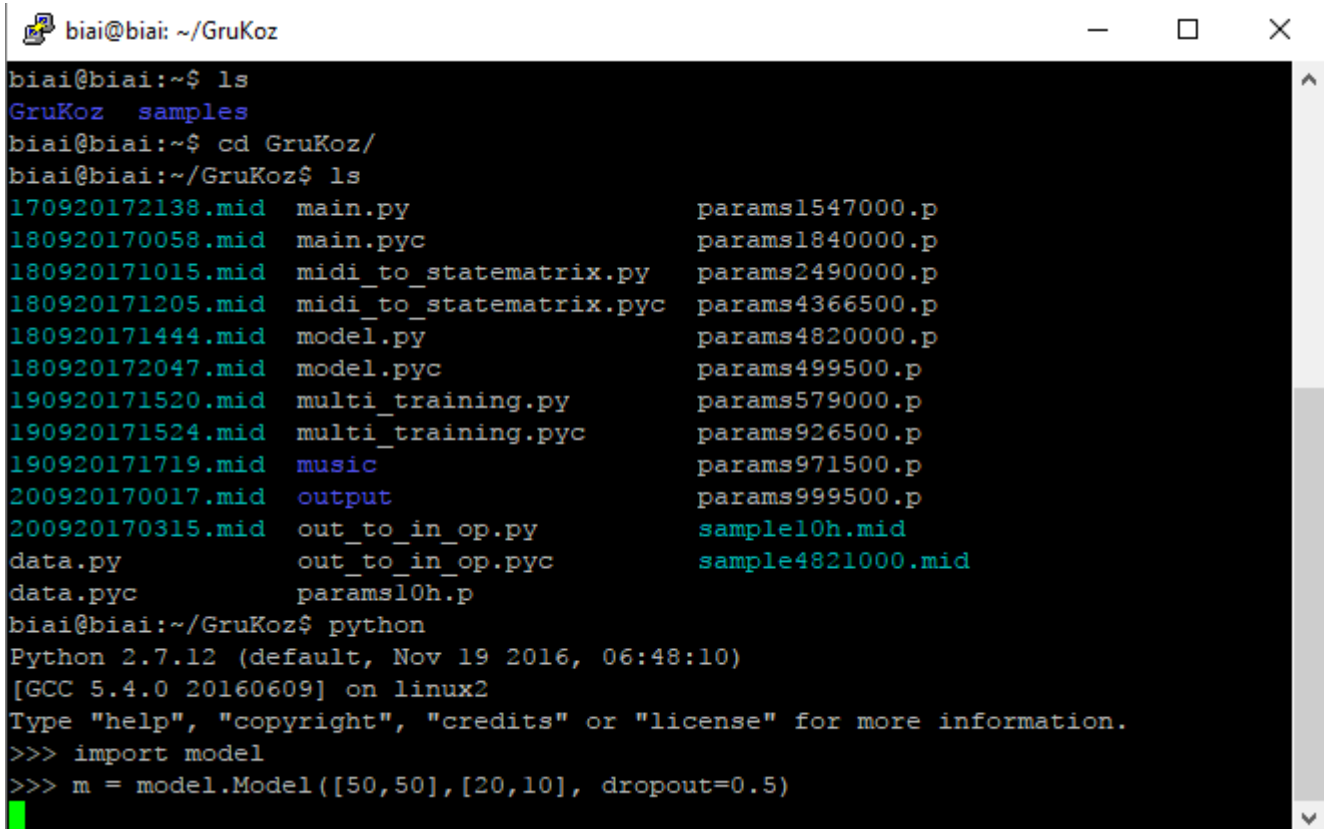
Do uruchomienia potrzebne było zainstalowanie programu Python w dowolnej wersji. Z racji na to, że wersja 2.7 jest wciąż najczęściej używaną pomimo faktu, że istnieje już nowsza, zdecydowaliśmy się właśnie na nią. Python 2.7 dostarcza zaawansowane biblioteki, służące do operowania na plikach muzycznych.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

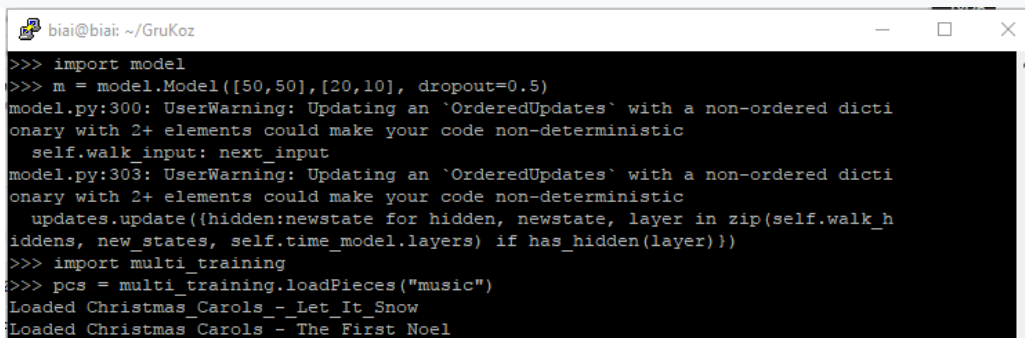
A terminal window titled 'biai@biai: ~/GruKoz' showing a file listing of the 'GruKoz' directory. The files include MIDI files, Python scripts, and parameter files. Below the listing, the user runs 'python' and enters a series of commands to import the 'model' module and create a neural network model 'm' with two hidden layers of 50 and 50 nodes, an output layer of 20 nodes, and a dropout rate of 0.5.

```
biai@biai:~/GruKoz$ ls
GruKoz  samples
biai@biai:~/GruKoz$ cd GruKoz/
biai@biai:~/GruKoz$ ls
170920172138.mid  main.py          params1547000.p
180920170058.mid  main.pyc         params1840000.p
180920171015.mid  midi_to_statematrix.py  params2490000.p
180920171205.mid  midi_to_statematrix.pyc  params4366500.p
180920171444.mid  model.py         params4820000.p
180920172047.mid  model.pyc        params499500.p
190920171520.mid  multi_training.py  params579000.p
190920171524.mid  multi_training.pyc  params926500.p
190920171719.mid  music            params971500.p
200920170017.mid  output           params999500.p
200920170315.mid  out_to_in_op.py   sample10h.mid
data.py          out_to_in_op.pyc  sample4821000.mid
data.pyc         params10h.p
biai@biai:~/GruKoz$ python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import model
>>> 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")
```

A terminal window showing the execution of the 'multi\_training' module. It imports the 'model' module and the 'multi\_training' module. Then, it calls 'multi\_training.loadPieces("music")', which loads two MIDI files: 'Christmas\_Carols\_-\_Let\_It\_Snow' and 'Christmas\_Carols\_-\_The\_First\_Noel'. There are also some warnings about 'OrderedUpdates' being updated with a non-ordered dictionary.

```
>>> import model
>>> m = model.Model([50,50],[20,10], dropout=0.5)
model.py:300: UserWarning: Updating an 'OrderedUpdates' with a non-ordered dicti
onary with 2+ elements could make your code non-deterministic
  self.walk_input: next_input
model.py:303: UserWarning: Updating an 'OrderedUpdates' with a non-ordered dicti
onary with 2+ elements could make your code non-deterministic
  updates.update(({hidden:newstate for hidden, newstate, layer in zip(self.walk_h
iddens, new_states, self.time_model.layers) if has_hidden(layer)}))
>>> import multi_training
>>> pcs = multi_training.loadPieces("music")
Loaded Christmas_Carols_-_Let_It_Snow
Loaded Christmas_Carols_-_The_First_Noel
```

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>

Microsoft Azure biai-vnet > biai572

Wyszukaj zasoby

biai572  
Network interface

Wyszukaj (Ctrl+ /)

Overview

Dziennik aktywności

Kontrola dostępu (IAM)

Tagi

USTAWIENIA

IP configurations

DNS servers

Przenieś Delete

Podstawowe elementy ^

Grupa zasobów (zmień)	Private IP address
biai	10.0.0.4
Lokalizacja	Virtual network/subnet
Europa Północna	biai-vnet/default
Nazwa subskrypcji (zmień)	Public IP address
Bezpłatna wersja próbna	52.138.150.164 (biai-ip)
Identyfikator subskrypcji	Network security group
eca63435-62f3-42ca-a5ef-16a2af8808f7	biai-nsg
	Attached to
	biai

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 przerw.

```
biai@biai: ~/GruKoz
>>> import model
>>> m = model.Model([50,50],[20,10], dropout=0.5)
model.py:300: UserWarning: Updating an `OrderedUpdates` with a non-ordered dicti
onary with 2+ elements could make your code non-deterministic
  self.walk input: next_input
model.py:303: UserWarning: Updating an `OrderedUpdates` with a non-ordered dicti
onary with 2+ elements could make your code non-deterministic
  updates.update({hidden:newstate for hidden, newstate, layer in zip(self.walk_h
iddens, new_states, self.time_model.layers) if has_hidden(layer)})
>>> import multi_training
>>> pcs = multi_training.loadPieces("music")
Loaded Christmas_Carols_-_Let_It_Snow
Loaded Christmas_Carols_-_The_First_Noel
Loaded Christmas_Carols_-_Angels_We_Have_Heard_On_High
Loaded Christmas_Carols_-_Do_You_Hear_What_I_Hear
Loaded Christmas_Carols_-_I_Saw_Mommy_Kissing_Santa_Claus
Loaded Christmas_Carols_-_Santa_Claus_Is_Coming_To_Town
Loaded Christmas_Carols_-_Holly_Jolly_Christmas
Loaded Christmas_Carols_-_Chestnuts_Roasting_On_An_Open_Fire
Loaded Christmas_Carols_-_Have_Yourself_A_Merry_Little_Christmas
Loaded Christmas_Carols_-_The_Christmas_Song_(John_Lennon)
Loaded Christmas_Carols_-_Frosty_The_Snowman
Loaded Christmas_Carols_-_Deck_The_Halls
Loaded Christmas_Carols_-_12_Days_Of_Christmas
Loaded Christmas_Carols_-_All_I_Want_For_Christmas_(Is_My_Two_Front_Teeth)
Loaded Christmas_Carols_-_Auld_Lang_Syne
>>> import cPickle as pickle
>>> m.learned_config = pickle.load(open( "params4820000.p", "rb" ) )
>>>
```

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.

```
Loaded Christmas_Carols_-_12_Days_Of_Christmas
Loaded Christmas_Carols_-_All_I_Want_For_Christmas_(Is_My_Two_Front_Teeth)
Loaded Christmas_Carols_-_Auld_Lang_Syne
>>> import cPickle as pickle
>>> m.learned_config = pickle.load(open( "params4820000.p", "rb" ) )
>>> multi_training.trainPiece(m, pcs, 200000)
epoch 0
epoch 100
epoch 200
epoch 300
epoch 400
```

Pierwszy cykl nauczania trwał niespełna 20 minut i wyniki były bardzo niesatysfakcjonujące. Po wygenerowaniu pierwszego pliku midi, przeformatowaliśmy 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.

```
1, 55, Note_on_c, 0, 57, 40
1, 55, Note_on_c, 0, 58, 40
1, 55, Note_on_c, 0, 59, 40
1, 55, Note_on_c, 0, 61, 40
1, 55, Note_on_c, 0, 64, 40
1, 55, Note_on_c, 0, 65, 40
1, 55, Note_on_c, 0, 68, 40
1, 55, Note_on_c, 0, 69, 40
1, 55, Note_on_c, 0, 70, 40
1, 55, Note_on_c, 0, 71, 40
1, 55, Note_on_c, 0, 73, 40
1, 55, Note_on_c, 0, 74, 40
1, 55, Note_on_c, 0, 75, 40
1, 55, Note_on_c, 0, 78, 40
1, 55, Note_on_c, 0, 79, 40
```

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 puszczoney.

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 ddmrrrrrgss

- Dzień
- Miesiąc
- Rok
- Godzina
- Sekunda



## 7. 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 poznać 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.

## 8. 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/>