Jakub Piasecki 175864

Maciej Wróblewski 175822

Generowanie muzyki klasycznej z użyciem sztucznych sieci neuronowych

Opis

Celem projektu jest stworzenie sztucznej sieci neuronowej i wyuczenie jej utworami muzyki klasycznej w taki sposób, aby była ona w stanie generować własną kompozycję. W sprawozdaniu przedstawiliśmy najlepszy znaleziony przez nas model, ale testowaliśmy wiele innych na wielu różnych zbiorach danych.

Przygotowanie danych

Jako format danych wejściowych przyjęliśmy wyłącznie pliki o rozszerzeniu mdi. Jest to format zapisu nutowego, który zajmuje mniej pamięci, jest dużo łatwiejszy do przetwarzania od typowych plików dźwiękowych, a dane w nim zawarte są lepsze w procesie uczenia. Ze stron [1] i [2] pozyskaliśmy utwory Ludwiga van Beethovena, Wolfganga Amadeusza Mozarta oraz Josepha Haydna, które zostały przez nas wykorzystane jako dane wejściowe do sieci. Ograniczenie liczby kompozytorów wynika z potrzeby zachowania podobnej stylistyki utworów wejściowych. Ich większa liczba spowodowałaby zatracenie stylu generowanej muzyki.

Sam format mdi nie jest wystarczającym do skutecznej nauki sieci. Dane należało przetworzyć. W tym celu skorzystaliśmy z biblioteki MIDO, która umożliwiła nam pracę z formatem. Przetwarzanie danych zostało zrealizowane w poniższy sposób:

- Scalenie wszystkich utworów w jeden plik.
- Usunięcie znaków kontrolnych i zastąpienie ich pustymi nutami o tym samym czasie trwania w celu uniknięcia desynchronizacji kompozycji.
- Usunięcie nut o zerowym czasie trwania ponieważ nie mają one wpływu na brzmienie utworu.
- Zapis danych do pliku csv o kolumnach: wartość nuty, intensywność, czas trwania.

Model sieci

Modelem sieci neuronowej z którego skorzystaliśmy jest model sekwencyjny. Główną bazą naszego projektu jest sztuczna rekurencyjna sieć neuronowa Long Short-Term Memory (LSTM).

Sieć rekurencyjna

Rekurencyjne sieci neuronowe (RSN; ang. recurrent neural networks) to odmiana sieci używana do analizy danych z szeregów czasowych, mogąca pracować na sekwencjach danych o dowolnej długości. To, czym RSN wyróżnia się od standardowych sieci neuronowych jest możliwość "pamiętania" poprzednich stanów. Dzięki możliwości pracy na sekwencji danych, sieci rekurencyjne są stosowane do przewidywania lub generowania kolejnego stanu na podstawie dostarczonego ciągu danych wejściowych.

Każdy neuron rekurencyjny składa się z dwóch zbiorów wag, jeden z nich jest przeznaczony dla danych wejściowych - x_t, a drugi dla danych wyjściowych z poprzedniego taktu - y_{t-1}. Wyniki warstwy neuronów rekurencyjnych dla pojedynczego przykładu są obliczane według poniższego wzoru.

$$y_t = \phi(W_x * x_t + W_y * y_{t-1} + b)$$

Gdzie:

Y_t - dane wyjściowe w takcie t.

X_t - dane wejściowe w takcie t

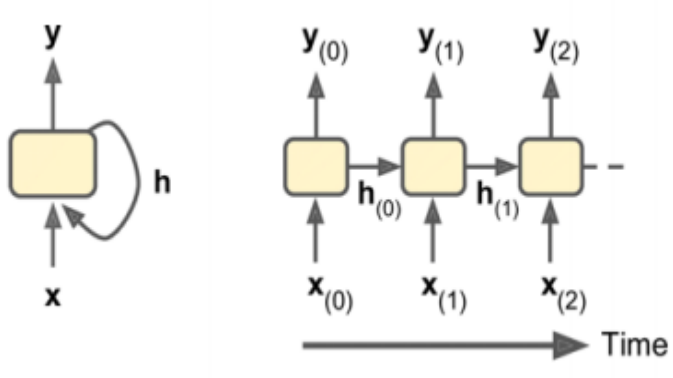
W_x – macierz wag połączeń dla wejść w bieżącym takcie.

W_y – macierz wag połączeń dla danych wyjściowych z poprzedniego taktu.

ϕ – funkcja aktywacji

b – wektor obciążenia

Komórka pamięci sieci rekurencyjnej



Wyjściem neuronu rekurencyjnego w ramce t jest funkcja wszystkich danych wejściowych z poprzednich ramek czasowych, dlatego można stwierdzić, że sieć rekurencyjna zawiera pewną "pamięć". Fragment sieci zachowujący informacje o stanie w poszczególnych ramkach czasowych jest nazywany komórką pamięci. Stan komórki w takcie t oznaczany jest symbolem *h*. W przypadku komórek podstawowych wyjście i stan są identyczne, ale w bardziej zaawansowanych typach komórek nie zawsze tak jest. W przeciwieństwie do sieci feedforward, w których aktywacja rozprzestrzeniała się tylko od warstwy wejściowej do warstwy wyjściowej, rekurencyjna sieć neuronowa polaryzuje połączenia w przeciwnym kierunku. Ponadto w każdym kroku czasowym neuron rekurencyjny oprócz danych wejściowych x_t otrzymuje również własne wyniki z poprzedniego taktu y_{t-1}.

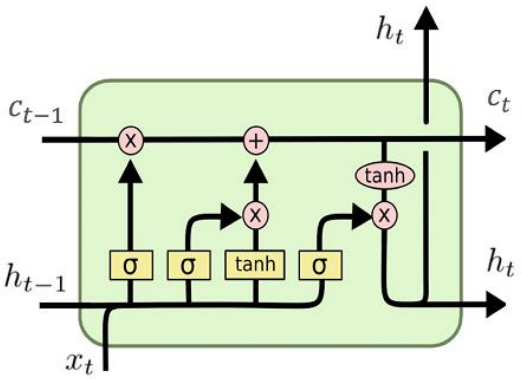
LSTM

Long-Short Term Memory to rodzaj sieci rekurencyjnej, różniący się od standardowych sieci RSN dużo większymi możliwościami pracy na długoterminowych zależnościach. Standardowe sieci RSN nie radzą sobie z tym zadaniem ze względu na problem zanikającego gradientu, czyli wielokrotne przemnażanie pochodnych, skutkujące bardzo małymi liczbami, które w efekcie nie pozwalają na skuteczne uczenie sieci. Ponadto w przeciwieństwie do standardowych sieci RNN, które na bieżąco nadpisują swoją pamięć, LSTM ma możliwość "zadecydowania" o tym, czy informacja powinna zostać zachowana w sieci, czy należy ją usunąć.

LSTM dzięki dodaniu bramki zapominającej jest w stanie dużo efektywniej kontrolować zarządzanie pamięcią, a w efekcie przechowywać informacje tak długo, dopóki są potrzebne. Ta właściwość czyni tę sieć niesamowicie skuteczną w wychwytywaniu długoterminowych wzorców w szeregach czasowych, takich jak między innymi nagrania dźwiękowe.

Sposób działania komórki

Stan długoterminowy C_{t-1} przechodzi przez bramkę zapominającą, tam niektóre wspomnienia są porzucane, a następnie dodawane są nowe, wyselekcjonowane przez bramkę wejściową. Później w niezmienionej formie stan ten przesyłany jest dalej. Stan krótkotrwały h_t jest przefiltrowaną i przepuszczoną przez funkcję tangensa hiperbolicznego kopią stanu długotrwałego.



LSTM
(Long-Short Term Memory)

Zmieniane przez nas parametry warstw LSTM to:

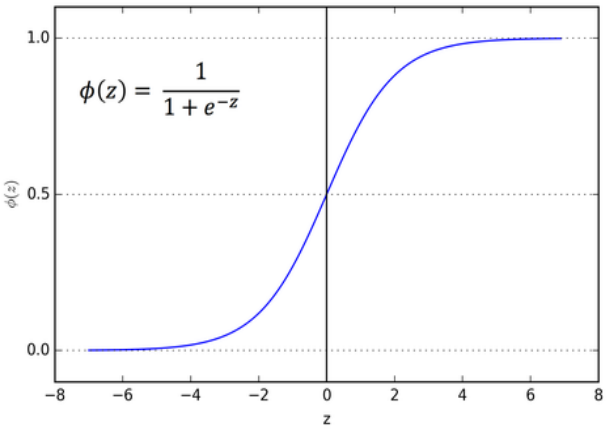
- *dropout* – wpływa na to, ile komórek w warstwie będzie aktywnych w danym cyklu
- *Zwracanie sekwencji* – ustawiony sprawia, że warstwa przekazuję całą sekwencję danych, zamiast pojedynczego wektora.

Inne wykorzystane warstwy

Poza warstwami sieci LSTM skorzystaliśmy również z warstw gęstych. Są to proste sieci neuronowe, w których każda wartość wejściowa połączona jest z każdym neuronem tej warstwy. Korzystają one z funkcji aktywacji – w naszym przypadku jest to aktywacja liniowa.

Ze względu na to, że mamy do czynienia z niewielkim zasobem danych uczących, dużą rolę w zapobieganiu przetrenowaniu odgrywają warstwy opuszczeń (dropout). W procesie uczenia, losowo z zadaną częstotliwością, ustawiają one wartości wejściowe na 0. Pozostałe skalowane są odpowiednio tak, aby suma wartości wejściowych pozostała nienaruszona.

Ostatnia warstwa jest warstwą aktywacji i korzysta z funkcji Softmax. Jest to funkcja wykładnicza, której wartość jest normalizowana w taki sposób, aby suma aktywacji dla całej warstwy była równa 1. Dzięki zastosowanej normalizacji wartości wyjściowe mogą być interpretowane jako oszacowane prawdopodobieństwo przynależności danego sygnału wejściowego do poszczególnych klas. Jej wykres przedstawiono obok.



Optymalizator i funkcja kosztu

Aby sieć była w stanie się uczyć potrzebny jest optymalizator, który dobiera parametry wagowe sieci. Robi to z użyciem tak zwanej funkcji kosztu, której globalne minimum stara się znaleźć, aktualizując współczynniki w każdym kroku uczenia. Najpopularniejszymi optymalizatorami są te, które rozszerzają metodę spadku wzdłuż gradientu. Głównym parametrem optymalizatorów jest tempo uczenia, które wpływa na wielkość zmian wprowadzanych w pojedynczej iteracji uczenia.

Początkowo wykorzystywaliśmy optymalizator Adam – Adaptive Movement Estimation. Cechuje się użyciem wykładniczej średniej kroczącej gradientu w celu przeskalowania tempa uczenia. Niestety w naszym projekcie osiągał on nieco gorsze wyniki niż Root Mean Square Propagation, który trafił do finalnej wersji modelu. W przeciwieństwie do Adam'a wykorzystuje on zwykłą średnią kroczącą do obliczania tempa nauki. Gromadzi wyłącznie gradienty z najbardziej aktualnych przebiegów, dzięki czemu jest w stanie sprawnie osiągnąć zbieżność z minimum globalnym.

Wykorzystana przez nas funkcja kosztu to entropia krzyżowa. Stanowi najbardziej rozpowszechnioną funkcję kosztu stosowaną w problemach klasyfikacji. Jest wypukła i łatwa do optymalizacji za pomocą technik stochastycznego spadku wzdłuż gradientu. Mierzy wydajność metod klasyfikacji modelu których wynikiem jest wartość probabilistyczna z zakresu <0;1>, z tego powodu idealnie łączy się z funkcją straty Softmax.

Model

Najlepszy znaleziony przez nas model składa się kolejno z warstw:

1. Warstwa gęsta (108 neuronów)
2. Warstwa LSTM (134 neurony, dropout = 0.06, zwracająca sekwencję)
3. Warstwa dropout (współczynnik = 0.10)
4. Warstwa LSTM (106 neuronów, dropout = 0.10, zwracająca sekwencję)
5. Warstwa dropout (współczynnik = 0.10)
6. Warstwa LSTM (88 neuronów, dropout = 0.05)
7. Warstwa dropout(współczynnik = 0.08)
8. Warstwa gęsta (108 neuronów)
9. Warstwa aktywacji Softmax

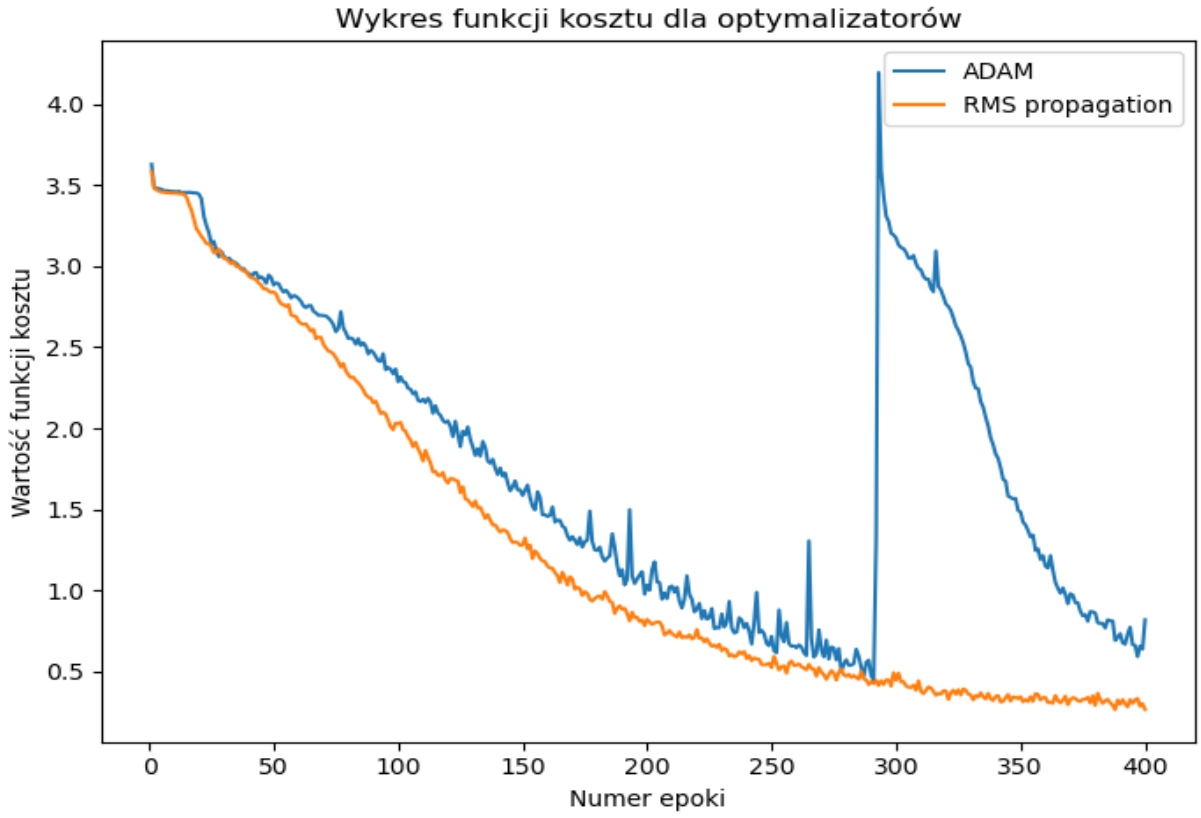
Proces uczenia i testowania

Ze względu na możliwości, ograniczyliśmy się do przewidywania wyłącznie nut, bez czasu trwania i intensywności. Wartości te zostały ustawione jako stałe. Próba dodania kolejnej zmiennej powodowała znaczny rozrost liczby klas, co zwiększyło trudność zadania, a i wszelkie próby stworzenia sensownie działającego modelu okazały się nieudane. Wyniki były gorsze niż przewidywanie tylko nut.

Zbiór danych podzielony został na zbiór uczący w postaci kompozycji Beethovena i Mozarta oraz na zbiór testowy składający się z utworów Haydna. Dane wejściowe ostatecznie nie są skalowane, próby skalowania do przedziału od 0 do 1 nie przynosiły zadowalających rezultatów.

Uczenie sieci polega na dostarczaniu jej próbek w postaci wektora N (dobrane przez nas N w najlepszym modelu wynosi 50) następujących po sobie nut, a także etykiety nuty kolejnej. W jednym kroku uczenia wykorzystujemy 12 przetasowanych z każdą epoką próbek wejściowych. Tworzenie utworu następuje poprzez podanie N początkowych nut ze zbioru testowego, na ich podstawie generowaniu kolejnej, wybierając klasę o najwyżej ocenionym prawdopodobieństwie wystąpienia i przesuwaniu okna nut o jeden w każdym kroku, dostarczając na koniec najnowszy dźwięk.

Wykres wartości funkcji kosztu w kolejnych etapach uczenia najlepszego modelu prezentuje się następująco:



Ulepszenia i problemy

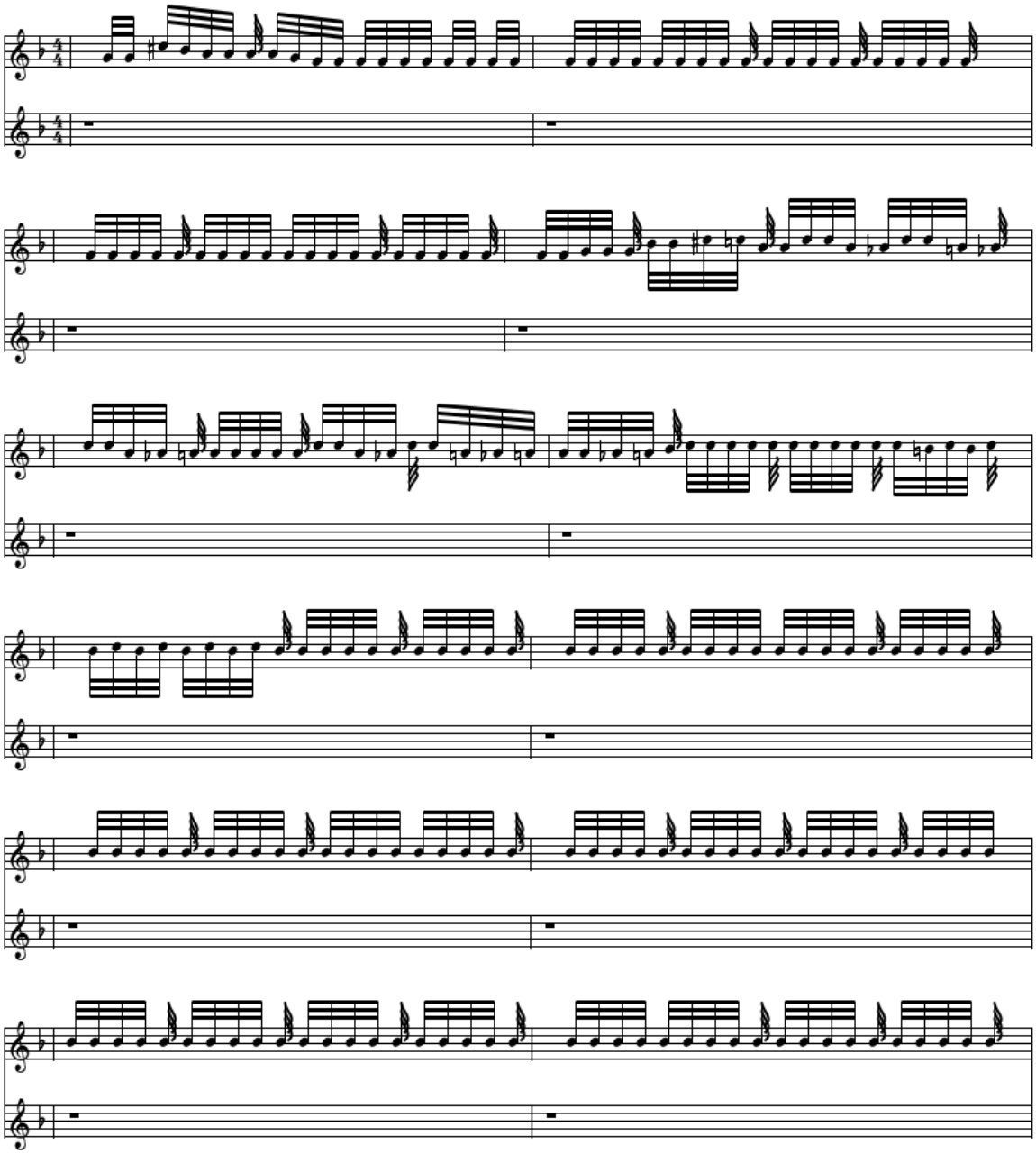
1. Znaczącym krokiem w przód, jeżeli chodzi o rozwój modelu, okazało się dodanie warstwy gęstej na początek. Sprawiła ona, że sieć lepiej dopasowała się do danych oraz zaczęła korzystać z szerszej gamy dźwięków procesie generowania.
2. Pierwsze wersje modelu miały wyraźną tendencję do zapętlania się, tj. generowały one cyklicznie te same dźwięki. Drobne ulepszenia w doborze parametrów sieci, sprawiały, że problem stawał się mniejszy, jednakże największą rolę w jego niwelowaniu odegrało wybieranie nuty na podstawie odwrotnej dystrybuanty względem tablicy predykcji. Nie jest to rozwiązanie niedoskonałości modelu, ale służy zaleczeniu tego skutków.
3. Problemem nierozwiązanym jest brak różnorodności i niewykształtowany styl w generowanych utworach.

Obserwacje

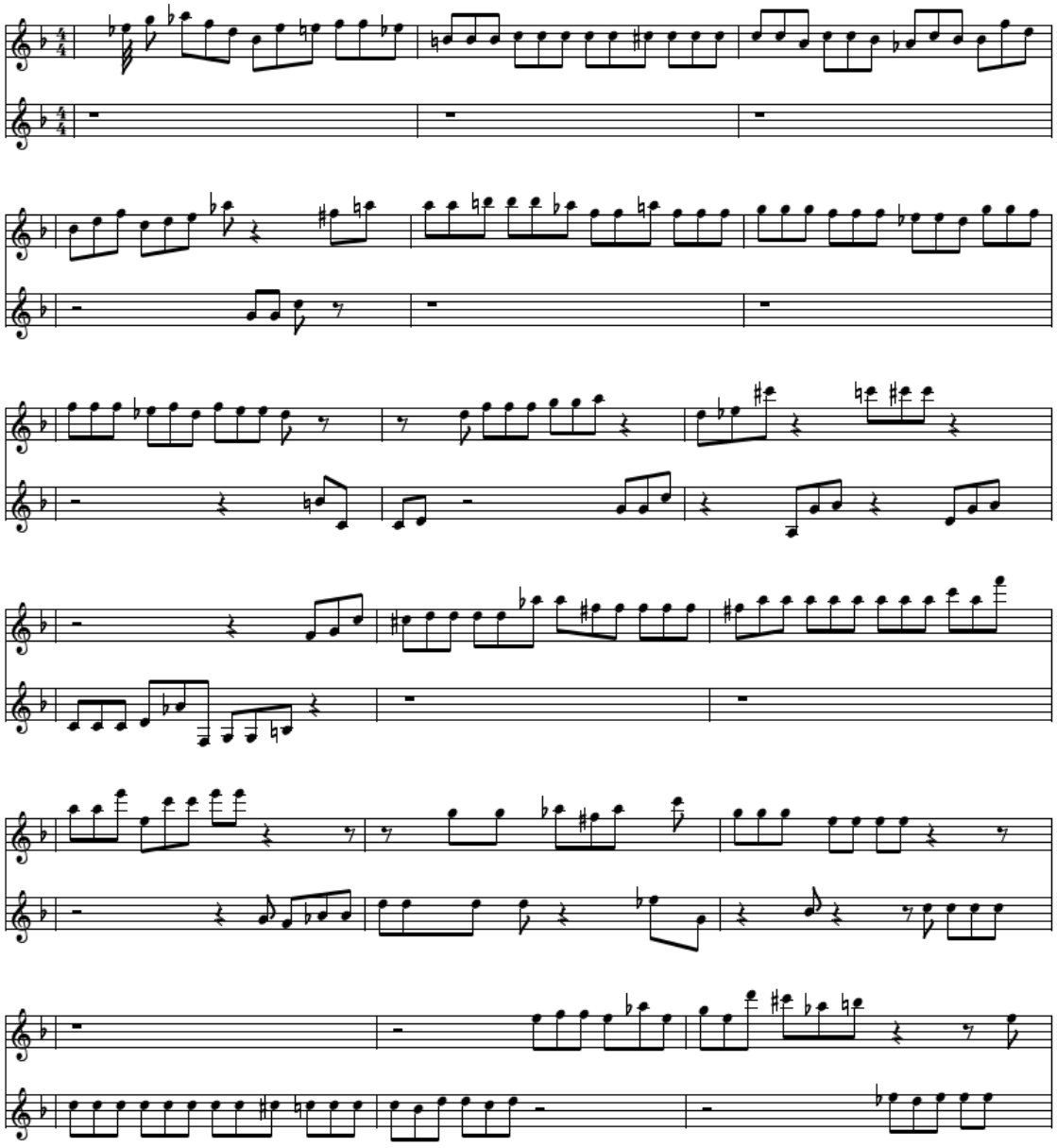
1. Im większa niż wystarczająca długość wektora nut wejściowych, tym większa szansa na to, że utwór zapętli się.
2. Próby wyuczenia modelu utworami wielu różnych kompozytorów sprawiają, że generowanym utworom brakuje spójności.
3. Struktura modelu sieci oraz jej parametry mają znaczący wpływ na jej działanie.

Różnica w zapisie nutowym pomiędzy początkowym modelem a końcowym

Model starszy



Model końcowy

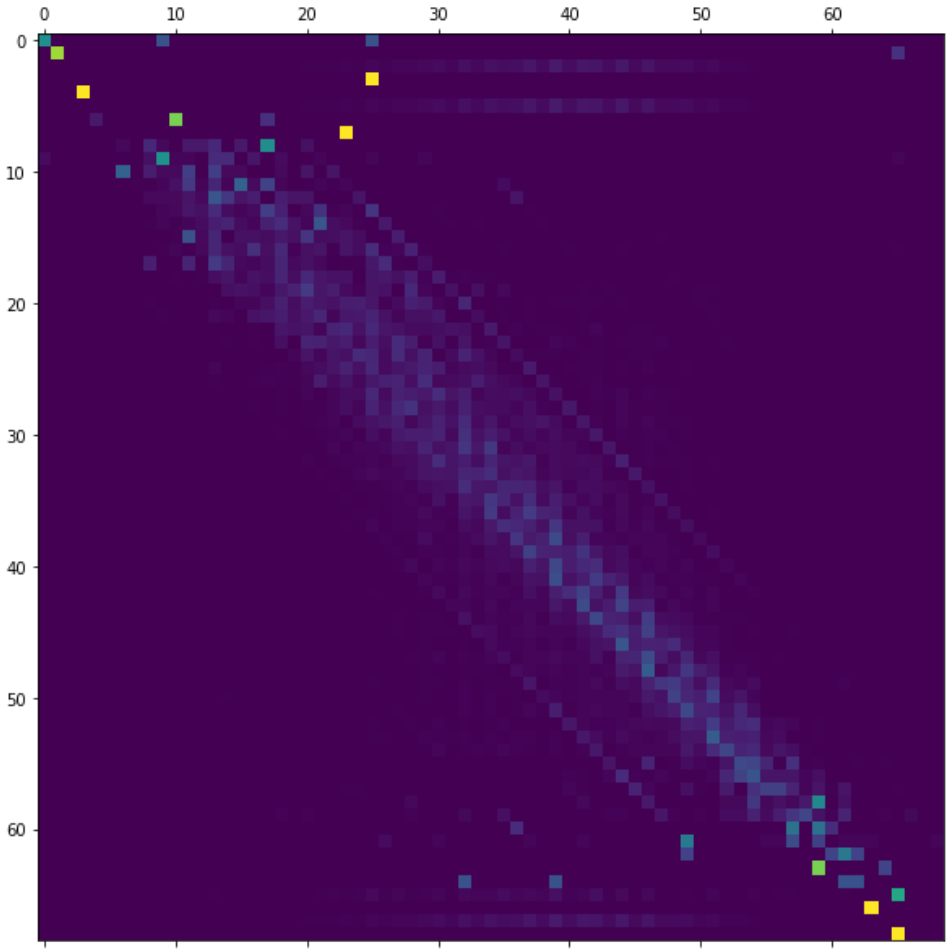


Podejście probabilistyczne

W projekcie sprawdziliśmy również podejście czysto probabilistyczne. Polega ono na generowaniu następnego dźwięku, wyłącznie na podstawie obecnego stanu, którym może być K poprzednich nut utworu. Ze względów technicznych, K nie może być większe niż 4, gdyż liczba kombinacji wszystkich stanów byłaby zbyt duża, żeby zmieścić je w pamięci.

Na podstawie danych uczących generowana jest macierz, w której indeks wiersza oznacza stan obecny, a wartości w tych wierszach oznaczają prawdopodobieństwo wystąpienia nuty następnej, oznaczonej przez indeks kolumny. Wartości w każdym wierszu sumują się do 1. Jeżeli w macierzy znajdzie się pusty wiersz, co jest bardzo prawdopodobne w przypadku, gdy obecny składa się z kilku dźwięków, to zastępowany jest on wierszem o rozkładzie takim samym jak cały zbiór uczący.

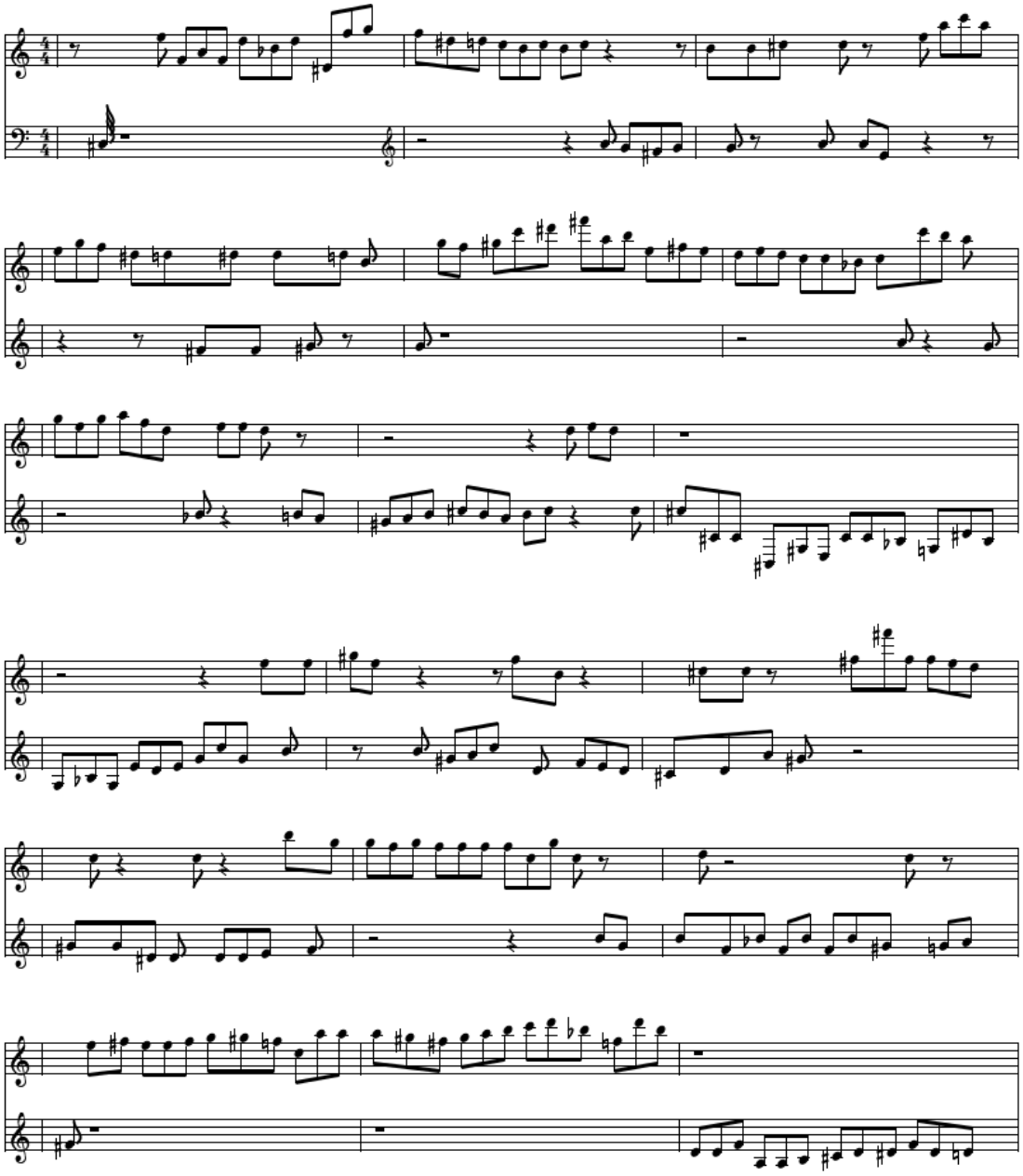
Poniżej prezentuje się macierz prawdopodobieństwa dla stanu obecnego składającego się z jednej nuty. Indeksy nut przesunięte są o stałą wartość. Kolor jaśniejszy oznacza większe prawdopodobieństwo:



Poza paroma jasnymi punktami, które dominowały w zbiorze uczącym, widać wyraźną zależność: dźwięk następny, z dużym prawdopodobieństwem jest bardzo podobny do poprzedniego.

Generowanie utworów

Muzyka generowana jest bardzo podobnie jak w podejściu z wykorzystaniem sztucznych sieci neuronowych, z tym, że w tym wypadku pierwsze dźwięki są wybierane losowo. Wygenerowane utwory w pewnym stopniu przypominają muzykę, cechują się szeroką gamą dźwięków, natomiast brakuje w nich jakkolwiek wyczuwalnej kompozycji, którą da się odczuć w muzyce generowanej z użyciem sieci LSTM. Tak prezentuje się zapis nutowy dla K=2 (wybrane z różnych K – subiektywnie najlepsze, choć różnice są minimalne):



Technologie

Projekt został napisany w języku Python z wykorzystaniem bibliotek tensorflow (konkretnie Keras API), numpy, pandas oraz Mido.

Źródła

Do sprawozdania dołączamy również najlepsze utwory oraz wyuczone modele sieci w etapach uczenia. Wyższy numer oznacza późniejszy czas generacji.

[1] http://www.piano-midi.de

[2] https://www.midiworld.com/mozart.htm

[3] Aurélien Géron - Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow