Wykorzystanie sztucznej inteligencji do generowania treści muzycznych

Mateusz Dorobek

Wydział Elektroniki i Technik Informacyjnych,
Politechnika Warszawska, ul. Nowowiejska 15/19, 00-665 Warszawa

mateusz.dorobek@gmail.com

Streszczenie. Artykuł opisuje zastosowanie sieci neuronowych w muzyce, zawiera także przegląd dostępnych technologii oraz opis istniejących architektur sieci neuronowych, które znalazłyby zastosowanie w tej dziedzinie.

Słowa kluczowe: sztuczna inteligencja, uczenie maszynowe, uczenie głębokie, sieci neuronowe, muzyka, jazz.

1 Wstęp

Muzyka jest nieodzownym elementem naszego życia, towarzyszy nam na każdym jego etapie. Zarówno w dniu codziennym, jak i w wyjątkowych chwilach odpowiednia muzyka jest bardzo ważna. Historia muzyki zaczęła się intensywnie rozwijać dopiero od końca XVIII w. natomiast jej początki sięgają ponad 3000 lat p.n.e. [1] Muzyka przez setki lat bardzo się zmieniła i przyjmowała bardzo rożne formy, a jej rozwój jest bardzo dynamiczny nawet współcześnie. Pomimo tak wielu różnic pomiędzy różnymi gatunkami i stylami muzycznymi, nadal łączy je wiele wspólnych zasad. Teoria muzyki jest obszerną dziedziną nauki, która bardzo szczegółowo opisuje takie aspekty jak kompozycja, aranżacja, czy harmonia, natomiast praktyka dowodzi, że odchodzenie od tych zasad często daje ciekawe rezultaty. To jedynie potwierdza jak złożoną dziedziną sztuki jest muzyka, oraz jak bliska ona jest człowiekowi. Wszelkie reguły jedynie próbują opisać to co człowiek stworzył od praktycznie zera. Teoretycznie rzecz ujmując muzyka jest niczym więcej niż sumą okresowych zakłóceń ośrodka o różnych częstotliwościach. Muzyka jaka ukształtowała się w naszej kulturze nie jest jedynie losowym szumem, a stanowi harmoniczną spójną całość, którą można próbować modelować.

Wpływ technologii na muzykę jest ogromny. Jak każda dziedzina sztuki rozwija się pod wpływem wielu czynników zewnętrznych, a kierunku tego rozwoju praktycznie nie sposób przewidzieć. Stworzenie nowych i udoskonalenie dawnych instrumentów, czy wynalezienie płyty winylowej na przełomie XIX i XX wieku, powstanie radia w pierwszej połowie XX wieku, czy rozpowszechnienie się płyt CD w latach osiemdziesiątych. Te wszystkie technologie zrewolucjonizowały muzykę, rozwinął się profesjonalny rynek muzyczny. Wraz z powstaniem Internetu, portali muzycz-

nych, serwisów streaming-owych dostępność muzyki osiągnęła ogólnoświatowy poziom. W ułamku sekundy jesteśmy w stanie odsłuchać nagrania z dowolnej części świata, udostępnić swoją twórczość i być na bieżąco z najnowszymi trendami w muzyce. Ma to swoje dobre i złe stronny, natomiast niezaprzeczalny jest wpływ technologii na kształt muzyki zwłaszcza współcześnie.

Muzyka jazzowa znacząco wyróżnia się na tle innych gatunków. Przede wszystkim razem z poprzedzającym ją bluesem stanowi fundament całej muzyki rozrywkowej. Rozmaitość styli, długa historia i całe spektrum aparatu wykonawczego muzyki jazzowej sprawia, że ta muzyka zasługuję na szczególną uwagę. Istotny z punktu widzenia tej pracy jest aspekt złożoności tej muzyki. Z jednej strony nieograniczona niemal swoboda wykonawcza, z drugiej obszerna teoria harmonii jazzowej. To połączenie daje wyjątkowo złożoną strukturę muzyczną, której analiza, czy modelowanie wymaga nawet nie tyle dużej wiedzy co doświadczenia. Jest to zarazem skomplikowany problem i intrygujące wyzwanie.

2 Przegląd istniejących rozwiązań i (literatury)

W tym rozdziale opiszę część z istniejących rozwiązań dotyczących generowania i przetwarzania muzyki z wykorzystaniem sieci neuronowych. W szczególności zdolne do transkrybowania, komponowania czy tworzenia i łączenia brzmień.

2.1 Hexahedria - Summer Research on the HMC [2]

Celem tego projektu było stworzenie oprogramowania zdolnego do generowania solówek jazzowych na podstawie struktury harmonicznej utworu. Bardzo pożądaną cechą architektur stosowanych w tego typu problemach jest niezależność od tonacji, czyli uczenie sieci do generowania solówek w jednej tonacji pozwala wykorzystać tak zdobyte doświadczenie do generowania solówek również w innych tonacjach. Tego typu architektury mają wzgląd na harmonię nie tylko jako ciąg niezależnych akordów, lecz na powiązaną strukturę, gdzie każdy akord pełni jakąś funkcję względem swoich sąsiadów. Oczywiście jest to podejście które odzwierciedla rzeczywistą strukturę harmonii, która nie jest zależna od tonacji w której jest utwór. Aby to osiągnąć trzeba było uzależnić aktualną nutę od jakiegoś punktu odniesienia. Można to zrobić na dwa sposoby: uzależniając aktualną nutę od poprzedniej, lub od aktualnego akordu. Obydwa podejścia mają swoje niebywałe zalety, jedna gwarantuje melodyczność, drugie zgodność z harmonią utworu. Aby połączyć te dwa podejścia skorzystano z metody Product-of-Experts.

Product-of-Experts Generative Model.
Sposób na połączenie dwóch dystrybucji w jedną dzięki czemu możemy skutecznie wykorzystać wyniki uzyskane z dwóch różnych sieci.

$$P(x|E_1, E_2) = \frac{P(x|E_1)P(x|E_2)}{\sum_{y \in A} P(y|E_1)P(y|E_2)} \qquad (1)$$

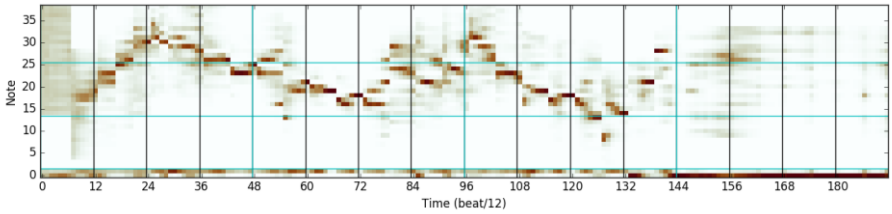


Fig. 1. Dystrybucja niezależna, bez wykorzystania metody.

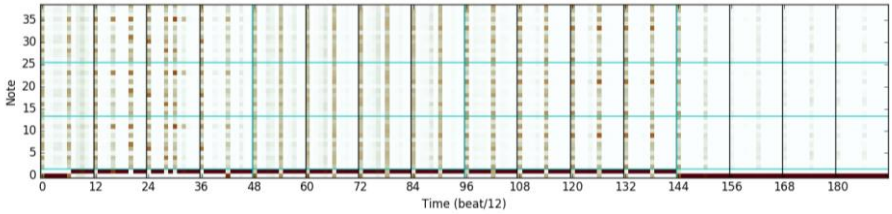


Fig. 2. Dystrybucja zależna od bieżącego akordu.

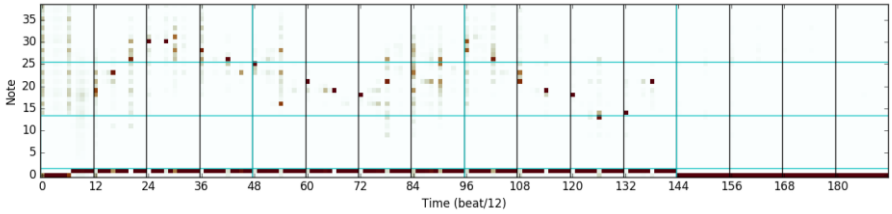


Fig. 3. Dystrybucja zależna od poprzedniego dźwięku.

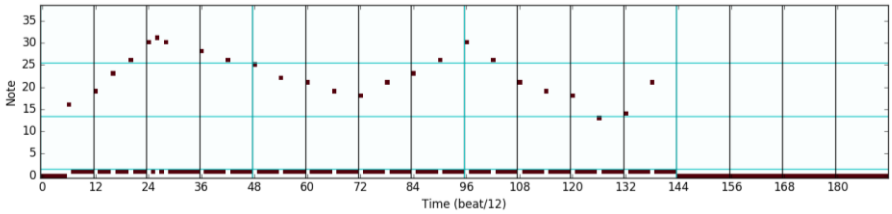


Fig. 4. Dystrybucja łączona z wykorzystaniem Product-of-Experts.

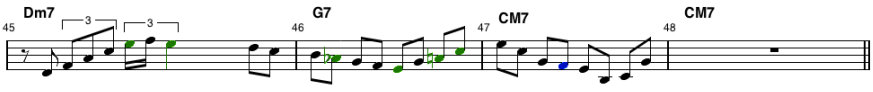


Fig. 5. Zapis nutowy dystrybucji łączonej

Łącząc dwie sieci z których pierwsza uzależnia rozkład prawdopodobieństwa wyboru nuty od poprzedzającej ją nuty i sieci, która uzależniała wybór nuty od akordu który aktualnie obowiązuje. Daje to możliwość analizy melodii muzycznej w poziomie (melodia) i w pionie (harmonia) co pozwoliło modelować muzykę w sposób bardziej zbliżony do rzeczywistości.

Compressing Sequence Autoencoder.

Istnieje specjalny typ improwizacji nazywane zawołanie i odpowiedź, lub inaczej czwórki, gdzie dwóch lub więcej muzyków na przemian gra krótkie improwizowane fragmenty, gdzie każdy jest pewną wariacją poprzedniego. Naiwnym podejściem byłaby lekka modyfikacja wysokości i czasu niektórych dźwięków. Odpowiednim podejściem okazują się autoenkodery [3], które pozwoliły by zenkodować fragmenty muzyczne i lekko modyfikować tak skompresowany wektor, dzięki czemu poruszamy się w przestrzeni dobrych solówek, a nie przypadkowo zmodyfikowanych.

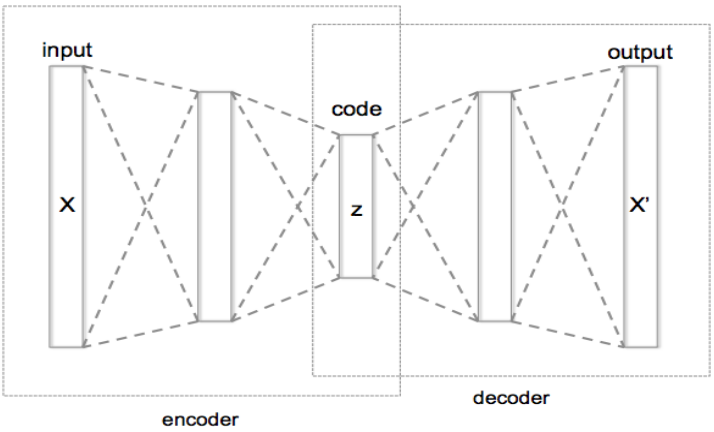


Fig. 6. Struktura autoenkodera.

Dwa główne podejścia:

Global Autoencoder - czyta całą sekwencję tworzy pojedynczy wektor i dekoduje z powrotem do całej sekwencji. Problemem jaki występuje przy tym rozwiązaniu jest to, że nie ono może zenkodować melodii o dowolnej długości do wektora o zdefiniowanej długości.

Timestep-Level Autoencoder - enkoduje pojedynczy odcinek czasowy jako wektor i później go dekoduje i bierze się za kolejny odcinek uzależniony już od wyniku pierwszego. Problemem jest to, że nie potrafi reprezentację sekwencji jako całość. Rozwiązaniem okazała się hybryda. Autoencoder produkuje sekwencje enkodowanych wektorów, ale produkowane w ten sposób sekwencje muszą być krótsze niż sekwencja wejściowa. Na przykład może zwracać pojedynczy wektor dla każdej połówki taktu na wejściu. To pozwala na przetrzymywanie dość długich solówek i jednocześnie uzyskiwanie sekwencji wektorów które razem mogą reprezentować kolejne części melodii wejściowej.

Generative Adversarial Model.
Była to próba rozszerzenia Generative Adversarial Networks (GAN) [4] tak by przystosować go do generowania sekwencyjnej muzyki. Podstawowym założeniem jest wykorzystanie dwóch elementów:

- Dyskryminatora, który próbuje odróżnić wytrenowane dane od wygenerowanych przez generator.
- Generatora, który generuje nowe dane i próbuje oszukać dyskryminator, żeby myślał, że to prawdziwe dane.

GAN-y najlepiej radzą sobie ze skomplikowanymi wielowymiarowymi danymi oraz z ciągłymi danymi wejściowymi, ale na potrzeby projektu spróbowano z dyskretnymi sekwencjami, których rozmiar i liczba jest mniejsza niż dla obrazów. Jednym z problemów jaki się pojawił była propagacja wsteczna wyjścia dyskryminatora. Rozwiązaniem było wzięcie wyjścia dyskryminatora dla konkretnego odcinka czasu i zastosowanie propagacji wstecznej korzystając z dystrybucji generatora dla tego odcinka. Dzięki czemu funkcja dyskryminatora będąca sumą ważoną stała się w pełni różniczkowalna względem współczynników generatora. Kolejnym problemem było to że w początkowej fazie generator generował losowe nuty, co dyskryminator wykrywał bardzo wcześnie, przez co nauczył się patrzeć jedynie na początek melodii. W efekcie nie był różniczkowalny po całości wejścia, a jedynie po jego początkowym fragmencie.

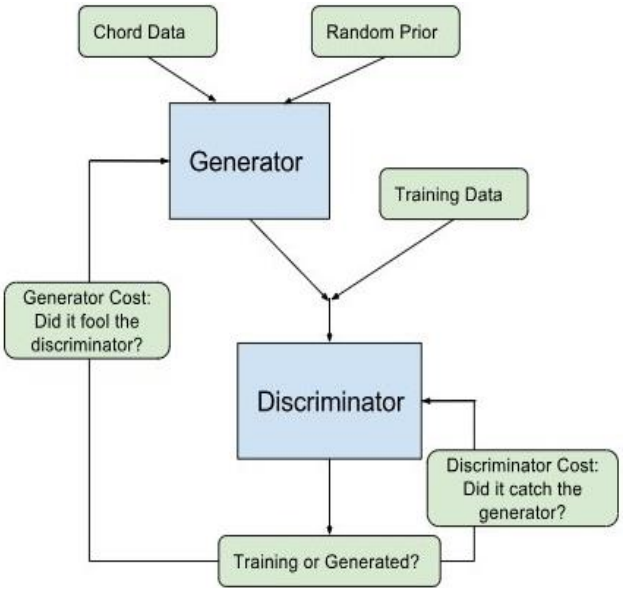


Fig. 7. Struktura GAN-a

2.2 Magenta [5]

Otwarto-źródłowy projekt zbudowany na TensorFlow. Magenta obejmuje głównie dwie dziedziny: muzyka i obraz. Niżej opisze szereg aplikacji, które powstały w oparciu o framework, jak zapewnia Magenta. Do każdego z projektów dodam komentarz odnośnie potencjału sztucznej inteligencji, jaki został w nim wykorzystany, oraz przydatności w muzyce. Mówiąc o zastosowaniu technologii w muzyce mam na myśli jej profesjonalny aspekt, pomijam możliwe zastosowanie w muzyce amatorskiej, czy hobbystycznej.

Melody Mixer [6] **i Latent Loops.** [6]
Prosta aplikacja pozwalająca z dwóch/czterech melodii utworzyć zbiór kilkunastu melodii zaczynający się od jednej a kończący na drugiej melodii, gdzie każdy z kolejnych fragmentów staje się coraz bardziej podobny do końcowej melodii. Program pokazuje możliwości AI w dziedzinie kompresowania dekodowania i enkodowania muzyki przez sieci neuronowe, oraz umieszczenia jej fragmentów w wektorowej przestrzeni, dzięki czemu możemy tak swobodnie się przemieszczać między fragmentami muzycznymi. Przydatność dla profesjonalnych zastosowań muzycznych nie jest natomiast zbyt duża. Można wykorzystać płynne przejścia melodii podczas przejść w muzyce elektronicznej, czy arpegiowanych motywach.

Beat Blender. [8]
Podobnie jak Latent Loops pozwala na płynne przechodzenie pomiędzy podanymi na wejściu, ale rytmami, a nie melodiami. Tutaj również jak w Latent Loops zastosowanie w muzyce elektronicznej i raczej nie do odgrywania na żywo, lecz w produkcji.

NSynth. [9]
Służy do syntezy brzmień na podstawie sampli podanych na wejście. W przeciwieństwie do standardowych narzędzi syntezy takich jak oscylatory, synteza FM, PCM, czy synteza tablicowa NSynth używa sieci neuronowych do syntezy na poziomie pojedynczych sampli, dając możliwość sterowania takimi (wysokopoziomowymi) parametrami, jak tembr i dynamika. Przy procesie uczenia wykorzystana była ponad 300 tysięczna baza dźwięków dostarczona również przez Magentę. Zastosowanie tego programu to przede wszystkim muzyka elektroniczna, czyli brzmienia do instrumentów klawiszowych, oraz postprodukcja.

Performance RNN. [10]
Jeden z ciekawszych projektów w tym zestawieniu. Aplikacja służy do generowania muzyki, która przypomina akompaniament fortepianu. Prosty interfejs pozwalający na wprowadzenie wag na odpowiednie składniki skali chromatycznej pozwala uzyskać odpowiedni nastrój w zależności od funkcji harmonicznej, którą przedstawiają wagi na kolejnych dźwiękach. Charakter tego akompaniamentu przypomina ten, który można usłyszeć w muzyce klasycznej. Wynik przypomina stylistycznie grę Chopina,

Debussy'ego, Mozarta. Aktualnie istnieją na rynku o wiele lepsze programy typu automatyczny akompaniator, z znacznie większą różnorodnością stylów i możliwością dokładnego odegrania harmonii utworu, lecz Performance RNN jako jeden z niewielu jest zbudowany na sieci neuronowej. Na razie nie stanowi dla nich zagrożenia, lecz ma potencjał, który sprawia, że warto śledzić ten i podobne mu projekty.

AI Duets. [11]
Aplikacja ta potrafi naśladować/odpowiadać na melodię zagrana na klawiaturze lub sterowniku MIDI. Wstępne wnioski:

- Jest świadoma tonacji w jakiej się znajduje.
- Jest świadoma jaka jest gęstość rytmu i rozpiętość melodii, potrafi na to reagować
- Nie potrafi grać harmonicznie, jedynie monofonicznie.
- Czasami cytuje bezpośrednio.
- Nie potrafi grać w tempie.

Na razie nie widzę żadnego profesjonalnego zastosowania, ale jest w tym potencjał, do grania na żywo improwizacji typu zawołanie i odpowiedź.

Neural Melody Autocompletion. [12]
Efekt działania podobny jak w A.I. Duets, aplikacja będąca rozszerzeniem wytrenowanej sieci ImprovRNN. Udało się również uzyskać świadomość harmonii, lecz melodie nadal są arytmiczne, i donikąd nie dążą. Wejściem jest akord grany na klawiaturze, lecz przy tak płytkiej (jeden akord) świadomości harmonii nie ma mowy o złożonej harmonii. Reakcja na zmianę harmonii też nie jest zbyt szybka ok. 2 sekund, więc na scenie okaże się bezużyteczny.

Onsets and Frames. [13]
Zdecydowanie najciekawszy projekt który bazuje na Magencie. Program potrafi transkrybować (tworzyć zapis nutowy utworu na podstawie zapisu audio) polifoniczne nagranie fortepianu, osiągając przy tym bardzo obiecujące wyniki. Wewnątrz posiada sieci konwolucyjne oraz LSTM. Autorzy za przyczynę tak dużej skuteczności uznają podział detekcji nut na dwa stosy sieci neuronowych. Jeden stos (Onset) wykrywa pierwsze kilka klatek w których nuta jest grana, a drugi stos (Frame) wykrywa te klatki, w których dźwięk jest nieaktywny. Surowe dane z części wykrywające początki nut są przekazywane jako dodatkowy argument do drugiego stosu.

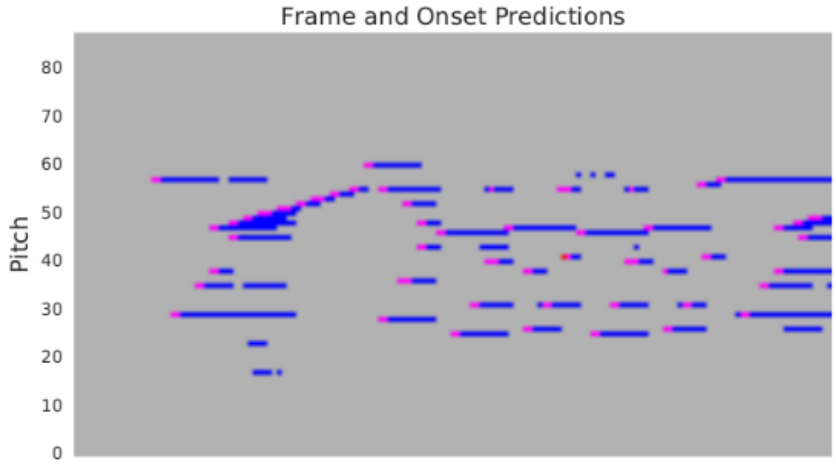


Fig. 8. Predykcje stosu Frames.

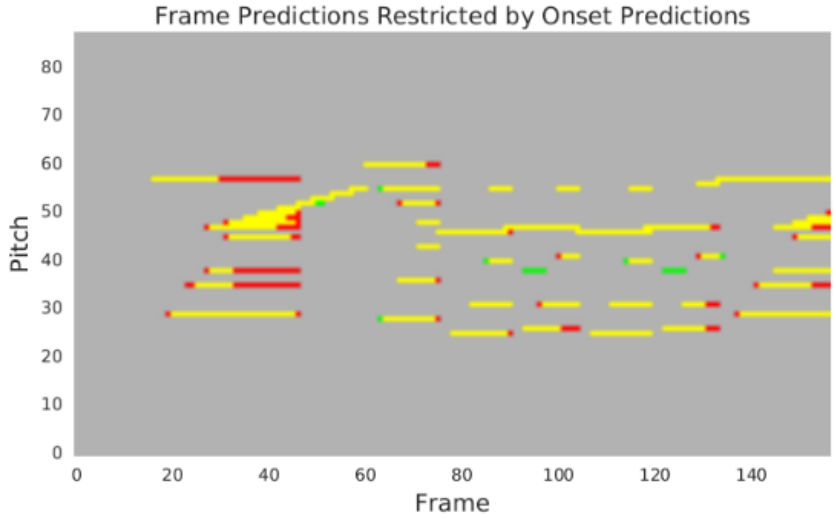


Fig. 9. Predykcje stosu Frames po korekcji przez predykcje stosu Onset

W **Fig 8.** nuty, które zaczynają się purpurowym kolorem znalazły potwierdzenie od stosu Onset, lecz nie wszystkie i to właśnie te które nie uzyskały tego potwierdzenia okazały się fałszywie wykryte, co potwierdza istotę tej drugiej warstwy. **Fig 9.** pokazuje na czerwono miejsca w których stos Frames nic nie wykrył, a na zielono te które wskazał błędnie. Żółte nuty to zgodność w obu stosach.

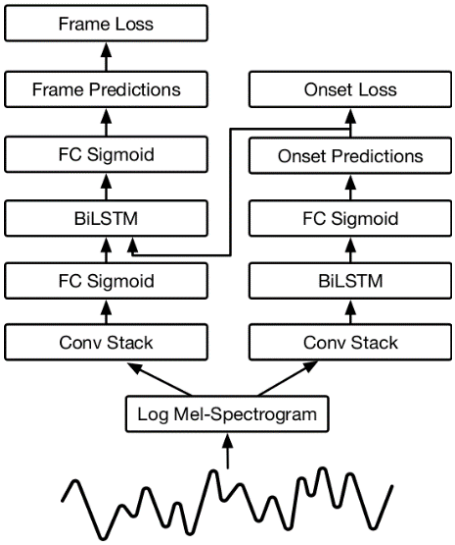


Fig. 10. Struktura modelu

Amper. [14]

Pozwala generować fragmenty muzyczne o bardzo różnej stylistyce, można manipulować takimi parametrami jak instrumentarium, tempo, tonacja, charakter i dynamika utworu. Długość może być teoretycznie dość dowolna, lecz czas generacji może stanowić pewne ograniczenie. 30 sekund nagrania generuje się około minuty, a na 90 sekund nagrania trzeba poczekać do 10 minut. Cechy wygenerowanej muzyki:

- Dobrze oddaje ustawiony nastrój
- Bardzo dobrze wyprodukowana muzyka
- Bardzo dobre sample
- Utrzymana tonacja i tempo
- Krótki czas trwania

Wygodny interfejs do generowania i dobra jakość powstałej muzyki to główne atuty tego projektu, niestety jest to komercyjny projekt, bez otwartych źródeł. Utwory tworzone przez Amper są wykorzystywane w reklamach telewizyjnych. Za 200$ można wykupić prawa do wygenerowanego utworu.

GRUV. [15]

Projekt Badawczy napisany w Pythonie z wykorzystaniem Kerasa i Theano, powstały w 2015 którego efektem było wygenerowanie bardzo zakłóconego, ale jednak audio w którym było słychać rytm harmonię i nawet przebijający się wokal. Struktura na jakiej się opiera to LSTM i CNN. Projekt nie jest już rozwijany, naukowcy, którzy się nim zajmowali nie trenowali go na dużym zbiorze i przeprowadzili jedynie 2000 iteracji.

DeepJazz. [16]

Projekt o architekturze LSTM korzystający z TensorFlow Theano i Kerasa. Model był trenowany na pojedynczym pliku MIDI. Po 128 cyklach treningowych na muzyce Pata Metheny-ego uzyskano rytmiczny utwór o złożonej harmonii z melodią, która korespondowała z akordowym akompaniamentem.

3 Sieci neuronowe

Niżej wymieniłem kilka architektur, który mogą znaleźć zastosowanie w tematyce, jaką porusza moja praca inżynierska, czyli w generacji treści muzycznych.

3.1 Rekurencyjne sieci neuronowe - RNN [17] [18]

Rekurencyjne sieci neuronowe w odróżnieniu od standardowych sieci neuronowych mają zdolność zapamiętania kontekstu, w którym się znajdują. Wiele zagadnień takich jak generowanie muzyki, rozpoznawanie tekstu, modelowanie języka, czy przetwarzanie obrazów, są zależne od poprzedniego stanu. Takie zjawiska próbują odtworzyć sieci rekurencyjne. Aby tego dokonać w ich strukturze pojawiają się pętle.

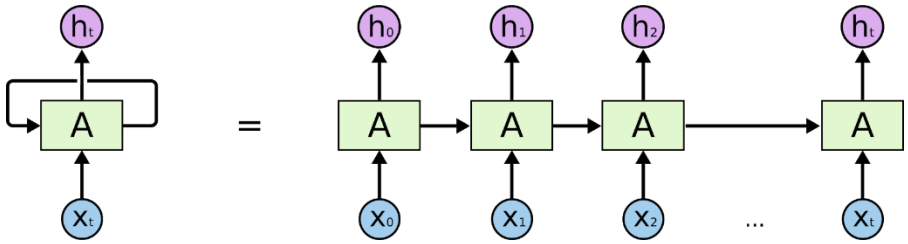


Fig. 11. Struktura sieci rekurencyjnych.

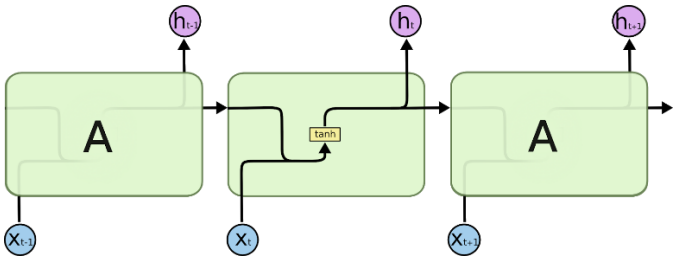


Fig. 12. Funkcja aktywacji tanh.

Oznacza to po prostu to, że pewne fragmenty sieci powtarzają się. Wiele problemów wymaga od naszej sieci zapamiętania krótkiego kontekstu, (**Fig 13)** np. szacowanie prawdopodobieństwa wystąpienia danego słowa po danej sekwencji. Z częścią z nich RNN radzą sobie bardzo dobrze.

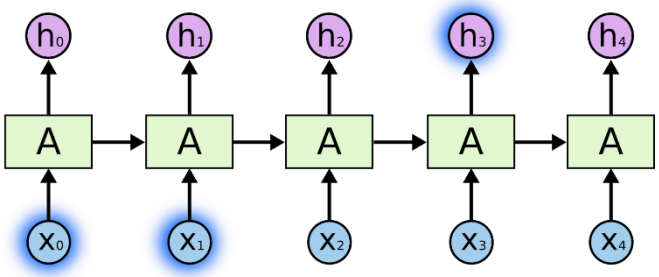


Fig. 13. Pamięć krótkotrwała.

Problem zaczyna pojawiać się, gdy problem potrzebuje znacznie szerszego kontekstu, (**Fig 14**) aby go rozwiązać. Teoretycznie standardowe sieci rekursywne (Vanilla RNN) mogą się tego nauczyć, ale praktyka tego nie potwierdza.

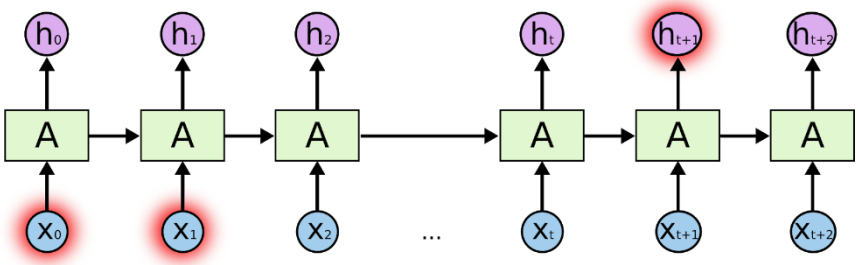


Fig. 14. Brak pamięci dla szerszego kontekstu.

Aby poradzić sobie z tym problemem powstały sieci LSTM, które o wiele lepiej uczą się wykorzystywać kontekst z nawet bardzo odległej przeszłości.

3.2 Long Short Term Memory [17] [18]

LSTM to specjalny rodzaj RNN, zdolny do uczenia się długo-terminowych zależności. Podobnie jak klasyczny RNN ma strukturę składającą się z powtarzających się elementów.

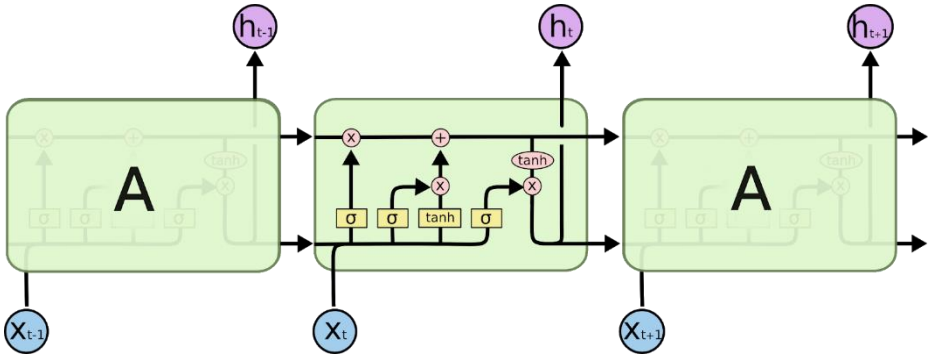
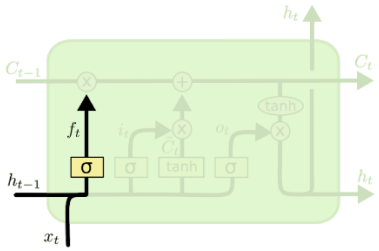


Fig. 15. Struktura LSTM.

Budowa podstawowego modelu LSTM:.

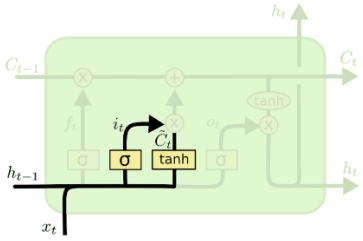
- **Forget gate layer (Fig 16)-** na tym poziomie decydujemy które informacje będą używane i w jakim stopniu, bramka nadaje każdej wartości na wejściu pewną wagę od 0 do 1.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Fig. 16. Forget gate layer.

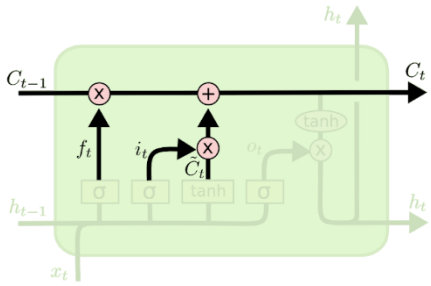
- **Input gate layer (Fig 17)** - w tym miejscu (warstwa sigmoid-owa) podejmuje się decyzje, które wartości będą aktualizowane. Warstwa tanh tworzy wektor wartości które będą dodane do stanu.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Fig. 17. Input gate layer.

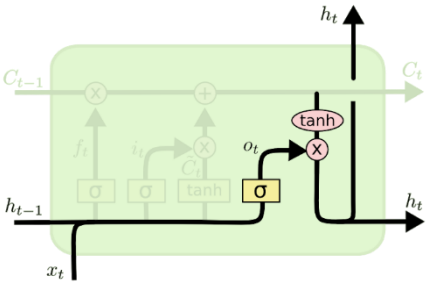
- Aktualizujemy stan naszej komórki wykonując macierzowe operacje mnożenia i dodawania. (**Fig 18**)



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Fig. 18. Operacje macierzowe.

- **Output gate layer (Fig 19)**- na końcu decydujemy które wartości będziemy dawać na wyjściu komórki.

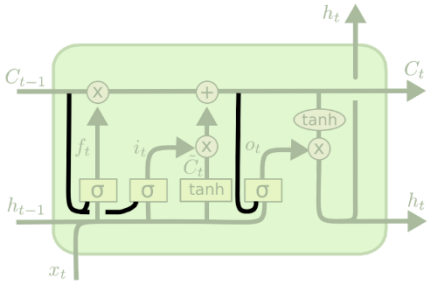


$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Fig. 19. Output gate layer.

Warianty LSTM.

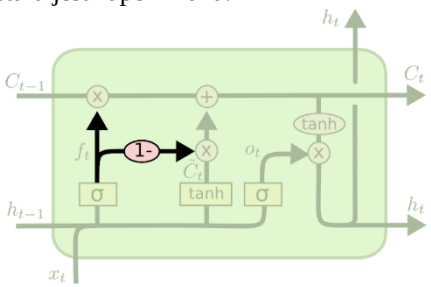
Przedstawiony wyżej model jest najbardziej podstawowym, istnieją natomiast zmodyfikowane odmiany, o których należy tu wspomnieć. Wersja z dodanym tzw. "peephole connections" (**Fig 20**) pozwalamy niektórym bramkom na wgląd do stanu komórki.



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t]\ +\ b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t]\ +\ b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t]\ +\ b_o\right)$$

Fig. 20. Peephole connections.

Połączone bramki forget i input. Tu poza pewnym uproszczeniem struktury pojawia się ciekawa zależność, a mianowicie możemy nauczyć się nowej rzeczy jedynie gdy stara jest zapominana.

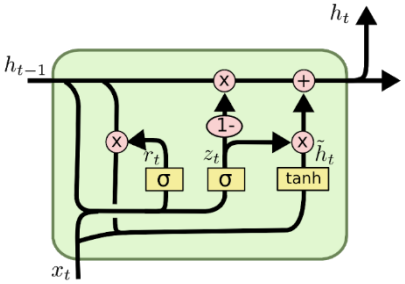


$$C_t = f_t * C_{t-1} + (\mathbf{1} - \boldsymbol{f_t}) * \tilde{C}_t$$

Fig. 21. Zapominanie starych zależności

3.3 Gated Recurent Unit

Gated Recurrent Unit (**Fig 22**) – wersja LSTM w której bramki forget i input są łączone w jedną bramkę update. Dodatkowo połączony jest stan komórki oraz jej stan ukryty, oraz występuje kilka innych połączeń wewnątrz struktury niż w standardowym LSTM. Rezultatem jest prostszy model, który jest ostatnio bardziej popularny.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Fig. 22. Struktura Gated Recurent Unit.

Przykładowa implementacja: CBLSTM - Andrej Karpathy [19]
Zalety sieci LSTM w dziedzinie generowania muzyki:

- świetnie generuje tekst - więc muzyka w formacie tekstowym MIDI
- szybko się uczy (w porównaniu do sieci rekurencyjnych)
- może generować dane bez końca - ważne, żeby nie istniało odgórne ograniczenie dla długości muzyki
- łatwo uczy się relacji przyczyna-skutek - co w muzyce pełni dużą rolę. Oprócz struktury pionowej - harmonia, posiada ona zależność od czasu, po dominancie następuje rozwiązania, a po idącej w górę melodii kierunek zmienia się w pewnym momencie na odwrotny.

- dyskretna reprezentacja danych - każda nuta ma konkretną wysokość, strój jest ustalony na konkretne wartości.

Wady

- musi przenieść format 2D na 1D - przez co zniekształca się obraz tego czym jest muzyka
- przez pamięć, którą posiada dochodzi do niekorzystnego zjawiska gradient explosion

Attention - Kolejny krok w rozwoju RNN i LSTM. [20]
Mechanizm, który pozwala obejść problem stałej długości wewnętrznej reprezentacji wektora danych w architekturze Encoder-Decoder w szczególności w przypadku gdy dane wejściowe są dłuższe niż dane treningowe. Jest to architektura, która działa na dwóch sieciach LSTM, z których jedna zakodowuje sekwencje wejściową na wektor o stałej długości, a druga go odkodowuje. Attention pozwala uwolnić się od problemu stałej długości wewnętrznej struktury. Osiągane jest to przez przechowywanie uśrednionych wyjść z enkodera LSTM dla każdego kawałka sekwencji wejściowej i trenowanie modelu tak, żeby zwracał selektywną uwagę na te wartości wejściowe i uzależniał od nich wartości sekwencji wyjściowej. Zwiększa to złożoność obliczeniową modelu, ale skutkuje to skuteczniejszym jego działaniem.

Podobna technika jest stosowana w przypadku dużych obrazów i nazywa się ją glimps-based modification (modyfikacja bazowana na przebłyskach) polega na tym że zanim zostanie wykonana finalna predykcja dokonuje się serii mniejszych pomiarów skupionych na mniejszych fragmentach, które później nakierowują na poprawną odpowiedź.

3.4 Generative Adversarial Network [4]

Istotą GANów są dwa konkurujące modele sieci. Jedna z nich to generator próbujący naśladować prawdziwe dane. Druga - dyskryminator otrzymuje dane zarówno te prawdziwe jak i wytwarzane przez generator i próbuje ocenić ich prawdziwość.
Przykładowa implementacja: HyperGAN - 255Bits.
Zalety sieci typu GAN w dziedzinie generowania muzyki:

- Jest przeznaczony do generowania twarzy - wyniki w tej dziedzinie są bardzo obiecujące
- Dobrze radzi sobie z danymi przestrzennymi - oznacza to że jest lepiej przystosowany do struktury wejściowej jaką jest muzyka.
- HyperGAN wykonując na GPU równoległe obliczenia jest bardzo szybki.

Wady

- Muzykę możemy zamienić na obraz 2D 96x64 pikseli i po zakodowaniu na RGB daje to jedynie 9,6s muzyki, co dla mnie jest dosyć krótkim czasem.
- Nie potrafi nauczyć się “co jest przed, a co po” - co jest dużą wadą jeśli chodzi o charakter muzyki.

- Nie działa na dyskretnych danych - muzykę można kwantyzować, ponieważ w tekstowym lub graficznym formacie zapisu jest dyskretna, nuty się zaczynają i kończą w konkretnych miejscach i mają konkretną wysokość, która należy do niedużego zbioru.
- Spektrum kolorów pikseli jest szerokie, więc obraz będzie zamazany - w związku z powyższym punktem jeżeli wyjście jest niedyskretne a wejście dyskretne muzyczny obraz jaki uzyskamy będzie zamazany.
- Proces uczenia, może się odwrócić - generator i deskryptor mają niechlubną cechę która w pewnych momentach fazy uczenia może spowodować odwrócenie tego procesu.

3.5 Convolutional Neural Network [21]

Są to sieci neuronowe najczęściej wykorzystywane w rozpoznawaniu obrazów. Ważną ich cechą jest niezależność od przesunięcia i od przestrzeni, co oznacza że nie dla lekko innych danych nie będzie konieczne ponowne uczenie się sieci. Ich struktura wymusza wykształcenie swego rodzaju filtrów, które analizują obraz.
Przykładowa implementacja: Pixel CNN - Aaron van den Oord
Zalety sieci konwolucyjnych w dziedzinie generowania muzyki:

- Uczy się kierunkowo, czyta dane wejściowe jako sekwencję, ale nie szkodzi, bo muzyka ma charakterystykę kierunkową, rozchodzi się w czasie w jednym kierunku.
- Tę sieć wykorzystuje się aby uzupełnić brakujące części obrazów, ponieważ generuje ona dalszą część danych na podstawie tego co już ma, więc w muzyce miałaby zastosowanie na przykład w generowaniu muzyki na żywo.

Wady:

- Działa bardzo wolno, ponieważ nic nie może zostać wyprodukowane, póki bezpośredni poprzednik (piksel) nie jest stworzony.

3.6 Auto-Encoder [22]

Sieć neuronowa wykorzystywana w nienadzorowanym uczeniu maszynowym.
Jest to algorytm kompresji danych, który jest dostosowany do danych na których operuje. Składa się z 3 elementów: kodera, dekodera, funkcji straty.
Koder, dekoder i parametryczne funkcje (sieci neuronowe) muszą być różniczkowalne względem funkcji straty będą optymalizowane aby zmniejszyć stratę wynikające z przeprowadzanej przez nie rekonstrukcji za pomocą metody propagacji wstecznej gradientu, dzięki temu sieć uczy się rozkładać dane wejściowe, oraz ignorować szum informacyjny w nich zawarty.

4 Narzędzia

Implementacja sieci neuronowych to złożone zadania z pomocą przychodzą pomocne narzędzia, z których większość jest darmowa, dobrze udokumentowana z setkami przykładów dostępnych w Internecie. Niektóre z nich posiadają wysokopoziomowe API pozwalające bardzo szybko implementować i modyfikować sieci neuronowych.

4.1 TensorFlow. [23]

Stworzona w 2015 roku przez Google Brain Team biblioteka do uczenia maszynowego i głębokich sieci neuronowych. Cechuje się dobrą wydajnością i skalowalnością. W celu zwiększenia wydajności napisano ją w C++, oraz stworzono API w Pythonie. Ponadto framework posiada dobrej jakości dokumentację i skuteczne narzędzie do wyświetlania i analizowania diagramów przepływu danych opisujących wykonywane obliczenia - TensorBoard. Istnieją także wyższe warstwy API takie jak Keras, tf.contrib.learn, TF-Slim, czy Estimators, które pozwalają testować i porównywać modele. Pół roku temu wprowadzono interfejs Eager Execution, który pozwala uruchamiać kod źródłowy TensorFlow bezpośrednio, tworząc grafy w późniejszym czasie, dzięki czemu uproszczone jest debugowanie i przyspieszone działanie w początkowej fazie.

TensorFlow można uruchomić na karcie graficznej, co jest jedna z ważniejszych jego go cech, ponieważ obliczenia dotyczące głębokich sieci neuronowych przyspieszają zwykle o jeden rząd wielkości na układach GPU (w szczególności na architekturze obliczeniowej CUDA od firmy Nvidia o ogólnym zastosowaniu, w przypadku większości frameworków) w porównaniu do układów CPU. Prostsze metody maszynowego uczenia się zasadniczo nie wymagają przyspieszenia oferowanego przez układy GPU. Poniżej porównanie trzech popularnych API do TensorFlow.

4.2 Keras [24]

Łatwo zacząć z nim pracę, ale jest mniej wydajny niż TFlearn, czy TensorLayer. Oryginalnie był stworzony do pracy z Theano, więc tam posiada lepsze wsparcie, ale po ogromnym sukcesie TensorFlow, również do niego zostało stworzone. Kerasa można obsługiwać bez znajomości TF co jest dużą zaletą, ponieważ w TFlearn i TensorLayer trzeba wykazać się pewną sprawnością w posługiwaniu się TensorFlow. Na początku 2017 roku Google wybrało Kerasa jako domyślne API TensorFlow.

4.3 TFlearn [25]

Biblioteka wspierająca jedynie TensorFlow. Jest przezroczysta dla TF, można w niej pisać wyrażenia symboliczne dzięki czemu można pisać bardzo efektywne metody TensorFlow.

4.4 TensorLayer [26]

Podobnie jak TFlearn wspiera tylko TensorFlow i jest dla niego jeszcze bardziej przezroczysty niż Kreas i TFlearn. Jest to w miarę nowa biblioteka (2016) autorzy stworzyli poręczne poradniki, ale środowisko użytkowników jest jeszcze dość małe.

4.5 Scikit-learn [27]

Framework Scikit-learn jest napisany w języku Python oferuje szeroki wybór skutecznych algorytmów uczenia maszynowego. Dla użytkowników języka Python Scikit-learn to z pewnością najlepsze rozwiązanie spośród wszystkich prostych bibliotek.

Scikit-learn to mocna i sprawdzona biblioteka uczenia maszynowego napisana w języku Python z szerokim asortymentem ugruntowanych algorytmów i zintegrowanych grafik. Program jest stosunkowo łatwy w instalacji, nauce i obsłudze oraz dysponuje dobrymi przykładami i dobrej jakości samouczkami.

Z drugiej strony, framework Scikit-learn nie obejmuje implementacji uczenia głębokiego ani uczenia przez wzmacnianie, nie oferuje modeli graficznych ani możliwości prognozowania sekwencji, nie może być też stosowany w przypadku języków innych niż Python. Nie obsługuje również PyPy, kompilatora „na czas" napisanego w Pythonie, ani procesorów graficznych. Nie sprawia w rzeczywistości żadnych problemów, jeśli chodzi o szybkość uczenia się. Korzysta z kompilera Cython (Python-to-C) dla funkcji, które muszą być wykonywane błyskawicznie, takich jak pętle wewnętrzne.

4.6 Caffe [28]

Biblioteka do implementacji głębokiego uczenia, który początkowo służył jako silny framework do klasyfikacji obrazów. Rozwój Caffe nieco zwolnił, biorąc pod uwagę nieustannie pojawiające się błędy, zatrzymanie rozwoju na wersji 1.0 sprzed ponad roku i opuszczenie projektu przez jego inicjatorów. Framework nadal jednak dysponuje dobrej jakości splotowymi sieciami neuronowymi do rozpoznawania obrazów i dobrym wsparciem dla układów GPU o nazwie CUDA od firmy Nvidia, a także prostym formatem opisu sieci. Z drugiej strony, Caffe często wymagają znacznych ilości pamięci GPU (ponad 1 GB) do ich uruchomienia, jego dokumentacja nadal zawiera błędy i jest problematyczna, trudno uzyskać wsparcie, a proces instalacji budzi wątpliwości, szczególnie w zakresie obsługi notebooka Python.

4.7 Theano [29]

Biblioteka konkurencyjna dla TensorFlow, wyszła w 2016 ostatnia aktualizacja do wersji 1.0.0. w listopadzie 2017. Narzędzie wydaje się być bardziej niskopoziomowe niż TF, możliwe że nawet bardziej wydajne, ale w porównaniu do konkurenta ma mniejsze wsparcie i nie jest tak intensywnie rozwijane.

4.8 Microsoft Azure ML Studio [30]

Jest to narzędzie do współpracy, obsługiwane metodą „przeciągnij i upuść", które służy do budowania, testowania i wdrażania rozwiązań z zakresu analizy predykcyjnej na podstawie posiadanych danych. Usługa Machine Learning Studio publikuje modele jako usługi sieci Web, które mogą być łatwo używane w niestandardowych aplikacjach albo narzędziach do analiz biznesowych. Machine Learning Studio to połączenie analiz danych, analiz predykcyjnych, zasobów w chmurze oraz samych danych. W ramach darmowego konta można uczyć sieć w ich chmurze, przez pewien ograniczony czas w miesiącu, posiadają graficzne API, w którym taką sieć można zbudować.

5 Bibliografia

[1] J. Chołmiński i K. Wilkowska-Chołmińska, Historia muzyki cz. I, PWM, 1989.

[2] R. Keller and D. Johnson, "Hexahedria," 08 08 2016. [Online]. Available: http://www.hexahedria.com/2016/08/08/summer-research-on-the-hmc-intelligent-music-software-team.

[3] D. P. Kingma i M. Welling, „Auto-Encoding Variational Bayes," *arXiv:1312.6114*, 12 2013.

[4] I. J. Goodfellow, J. Pouget-Abadie i M. Mirza, „Generative Adversarial Networks," *arXiv:1406.2661*, 01 2014.

[5] Google, „Magenta," Google, [Online]. Available: https://magenta.tensorflow.org/.

[6] Google, „Melody Mixer," Google, [Online]. Available: https://experiments.withgoogle.com/ai/melody-mixer/view/.

[7] Google, „Latent Loops," Google, [Online]. Available: https://teampieshop.github.io/latent-loops/.

[8] Google, „Beat Blender," Google, [Online]. Available: https://experiments.withgoogle.com/ai/beat-blender/view/.

[9] Google, „NSynth," Google, [Online]. Available: https://magenta.tensorflow.org/nsynth.

[10] Google, „Performance RNN," Google, [Online]. Available: https://magenta.tensorflow.org/demos/performance_rnn/index.html.

[11] Google, „AI Duets," Google, [Online]. Available: https://experiments.withgoogle.com/ai/ai-duet/view/.

[12] Google, „Neural Melody Autocomletion," Google, [Online]. Available: https://codepen.io/teropa/full/gvwwZL/.

[13] Google, „Onset and Frames," Google, [Online]. Available:

https://magenta.tensorflow.org/onsets-frames.

[14] Amper Music, „Amper," Amper, [Online]. Available: https://www.ampermusic.com/.

[15] M. Vitelli i A. Nayebi, „Github," GRUV, [Online]. Available: https://github.com/MattVitelli/GRUV.

[16] J. S. Kim, „GitHub," DeepJazz, [Online]. Available: https://github.com/jisungk/deepjazz.

[17] A. Sherstinsky, „Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term," *arXiv:1808.03314*, 08 2018.

[18] C. Olah , „colah.github," Understanding LSTMs, 27 08 2015. [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[19] A. Karpathy, „GitHub," 08 2015. [Online]. Available: https://github.com/karpathy/char-rnn.

[20] J. Brownlee, „machinelearningmastery," 30 06 2017. [Online]. Available: https://machinelearningmastery.com/attention-long-short-term-memory-recurrent-neural-networks/.

[21] A. Tejumade, „medium," 13 02 2018. [Online]. Available: https://medium.com/ai-saturdays/aisaturdaylagos-basic-overview-of-cnn-cd354470e2bb.

[22] F. Chollet, „blog.keras.," 14 05 2016. [Online]. Available: https://blog.keras.io/building-autoencoders-in-keras.html.

[23] Google, „tensorflow," Google, [Online]. Available: https://www.tensorflow.org/.

[24] Keras, „Keras," Keras, [Online]. Available: http://keras.io/.

[25] TFlearn, „TFlearn," TFlearn, [Online]. Available: http://tflearn.org/.

[26] TensorLayer, „TensorLayer," TensorLayer, [Online]. Available: https://tensorlayer.readthedocs.io/en/stable/.

[27] Scikit-Learn, „Scikit-Learn," Scikit-Learn, [Online]. Available: http://scikit-learn.org/stable/.

[28] Caffe, „Caffe," Caffe, [Online]. Available: http://caffe.berkeleyvision.org/.

[29] Theano, „Theano," Theano, [Online]. Available: http://deeplearning.net/software/theano/.

[30] Microsoft, „Microsoft Azure ML Studio," Microsoft, [Online]. Available: https://azure.microsoft.com/pl-pl/services/machine-learning-studio/.