# LSTM based music generation

Falak Shah Twisha Naik Nisarg Vyas
*InFoCusp Innovations Private Limited*
Ahmedabad, India
{falak, twisha, nisarg}@infocusp.in

***Abstract*—Use of deep learning techniques for generating music has gained popularity in the recent years owing to the high amount of compute power being available and the evolution of deep learning architectures which are well suited for learning patterns from sequential data. In this paper, our goal is to generate musical notes that will follow a given input primer sequence such that the entire generated musical sequence sounds continuous and melodious. For this, we leverage Google Magenta's inbuilt models and propose new methods for data ingestion on Long Short Term Memory (LSTM) based models. In the most basic magenta model, the current note is passed as an input and the distribution of the next note is predicted as the output, this then is used to generate the next note. We propose variants of this off-the-shelf Magenta model. One of the proposed variants feeds in extra information about the targets into the model to aid the prediction process. Another variant is based on the observation that the melody is relatively independent of the starting note, and the difference in pitches of notes from the start note preserves the characteristics of a song in a compact manner, amenable for machine learning modeling. Along with introducing new models, the existing work explores the inbuilt magenta model with different hyperparameters settings, presenting quantitative results that serve as a benchmark.**

***Index Terms*—Music , LSTM , Deep Learning**

## I. Introduction

### A. Deep Learning based music generation approaches

Formulation of music sequence generation as a deep learning problem has been a much researched upon topic since past few years. A wide variety of approaches have been proposed in literature that employ deep learning for music generation [1] [2]. Some of these approaches make use of the raw music waveform, while others are based on structured formats such as MIDI [3] or MusicXML [4].

The goal of MIDI or MusicXML based methods is to predict the next notes in the sequence from the combination of current note and (optionally) chord information. A "note" here indicates a sound generated by a single midi pitch of a given musical instrument. The notes define the overall melody of the song. The chords represent a combination of 3 or more notes played simultaneously and they act in a supporting role forming the song's harmony. These notes and chords are converted into vector representations and input to the machine learning models that generate the next notes. Sequence-based models such as Long-Short Term Memory (LSTM) [5] models and Recurrent Neural Networks (RNNs) are most commonly used for these type of problems as they have shown promising results in learning sequence information from time series data [6] [7].

### B. Literature survey

The first direction of research is on using direct audio waveform for training models. The WaveNet architecture by van der Oord et al. was aimed at generating raw audio waveforms. It uses a convolutional feedforward network without pooling layer [8]. In [9], Gruv et al compare LSTM vs Gated Recurrent Unit (GRU) based models for music models showing that LSTM outperforms GRU at audio waveform generation.

Choi et al. [7] have explored applications of character and word-based RNNs with LSTM units for automatic generation of jazz chord progressions and rock music drum tracks. Briot et al. [10] identify issues in direct deep learning based music generation such as generated music mimicking the music it is trained on and lack of control over generated patterns and suggest solutions for the same. An unsupervised learning approch called Audio Word2Vec [11] was proposed to generate vectors from analog audio data without human annotation using Sequence-to-sequence Autoencoder.

Generative models have also been explored for music generation. Dong et al. proposed three models for symbolic multi-track music generation under the framework of Generative Adversarial Networks (GANs) [12]. It combines music-specific sequence learner using an LSTM network with an autocorrelation-based predictor. Such a model is able to learn arbitrary long-timescale correlations in music while avoiding overfitting. [13].

The MidiNet model [14] explores the use of Convolutional Neural Networks (CNNs) [15] for generating a series of MIDI notes one bar [VIII-A] after another. In addition to the generator, a discriminator is trained to learn the distributions of melodies, making it a generative adversarial network (GAN). Conditional mechanisms have been developed to exploit available prior knowledge, so that the model can generate melodies either from scratch, by following a chord sequence, or by conditioning on the melody of previous bars (e.g. a priming melody).

In this paper, we focus on LSTM based generation of melodies given the primer sequence. We train on music sequences in the form of MusicXML or MIDI format. We describe the formulation of midi based music generation as a deep learning problem in the next subsection.

### C. Formulating Music generation as deep learning problem

Melody is a continuous and pleasant sounding sequence of notes. MIDI and MusicXML files are digital representations of melodies that store information about which note is being

978-1-7281-6119-8/19/$31.00 ©2019 IEEE
DOI 10.1109/iCMLDE49015.2019.00020

played at what time, with what intensity and for how long. Every melody encoded in these formats is a time sequence and thus, sequential models like RNNs and LSTMs can be used to identify patterns in the data. The aim of the models described in this paper is to predict notes following a given primer sequence, based on the patterns that the model has learned from the data.

The primer sequence provided as an initial input is used to set the internal states of recurrent models that are further used to predict the next notes in the sequence. The set of notes that can occur in the output is a finite set and hence the prediction problem is formulated as a classification problem. At each time t, a softmax vector giving the probabilities of the next note in the output is generated and then a note is picked based on the softmax probabilty distribution. The basic model for music generation takes the note at time t as an input and uses hidden state from time t-1, to predict the distribution of notes at time t+1.

For our experiments in this paper, we have used the magenta library by Google [16] and the Wikifonia dataset [17]. The in-built magenta models were trained using different hyper-paramaters on Wikifonia data to get comparable results to the pre-trained models. Further, changes were made in the provided inputs to improve results. The advantages of using magenta include: the ability to handle different input formats like MusicXML and MIDI; easy pre-processing of the data and generation of reproducible results. Magenta code is fast and supports GPU usage. It also supports tensorboard [18] which makes the interpretation of training easy through visualization.

In this paper, Section-II describes the analysis of data set used and pre-preprocessing steps for converting the MusicXML files to a format suitable for feeding it into the models. Section-III and IV provide details of the proposed variants and the model architecture used, respectively. Section-V has the results we observed during various experiments.

## II. Data Analysis and Preprocessing

Wikifonia dataset [17] which has 6403 scores, was used for training the deep learning model. The raw data is in the form of MusicXML files. We limit ourselves to just monophonic (single instrument playing) melodies for the training process, with piano being the selected instrument.

The data undergoes some pre-processing steps before it is used for training the models. In the first step, the note information extracted from the MusicXML files. The extracted notes have the pitch of the note, start time and end time. To digitize the analog time variable, quantization is performed. The quantized sequence of notes is then encoded using different encoders and fed into the model. Scores having longer than a certain period of silence in between are split into two independent training examples at that point, resulting in more scores than the input dataset. After undergoing these pre-processing steps, a total of 7442 samples are used for training. Each of these steps are explained in detail in the following subsections.

### A. Note to pitch mapping

There are twelve notes [VIII-A] in one octave [VIII-A] on the piano. The white keys are described by seven alphabets A to F and there are 5 alter notes (black keys on piano) described by placing **'#'** after the note (Table I). These piano notes are mapped to unique integers, denoted as midi_pitch using the formulae in equation 1 and 2.

| Pitch identifier | C | C# | D | D# | E | F |
|---|---|---|---|---|---|---|
| Base pitch class | 0 | 1 | 2 | 3 | 4 | 5 |
| Pitch identifier | F# | G | G# | A | A# | B |
| Base pitch class | 6 | 7 | 8 | 9 | 10 | 11 |

TABLE I
Pitch Identifier to Base Pitch Class

| Sharp # | Double sharp ## | No alter | Flat b | Double Flat bb |
|---|---|---|---|---|
| 1 | 2 | 0 | -1 | -2 |

TABLE II
Alter Types

$$pitch\_class = (base\_pitch\_class + alter)\%12 \quad (1)$$

$$midi\_pitch = (12 + pitch\_class) + octave * 12 \quad (2)$$

**Example:** Note **C#4** can be converted to midi_pitch as follows:
From table I and II it can be seen that,
base_pitch_class of C = 0 and alter of # = 1, octave = 4
Using equation 1, **pitch_class of C#** = (0 + 1)%12 = 1
Using equation 2, **midi_pitch** = (12 + 1) + 4*12 = 61

### B. Restricting note pitches within a range

Since majority of notes of the dataset fall in octave 3-6, the notes within the corrsponding midi_pitch range [48, 84) are kept as-is. The rest are clipped to fall within this range using the below algorithm. The key idea is to keep the offset of the note from the start of the octave the same as before while clipping to the given range.

```
if note < min_note:
  updated_note = min_note +
  (note - min_note) % notes_per_octave
elif note >= max_note:
  updated_note = (max_note
  - notes_per_octave) +
  (note - max_note) % notes_per_octave
```

where $notes\_per\_octave = 12$, $min\_note = 48$ and $max\_note = 84$.
**Example:** If the note value is 20 (offset 8 from the octave starting at 12), we set it to 56 (offset 8 from the $min\_note$ which is 48) or if the note value is 90 (offset 6 from the octave starting at 84), we set it to 78 (offset 6 from start of last octave in range at 72).

### C. Time Split

In case of multiple time signatures[VIII-A] within a single sample, it is split into separate samples from the point of change in time signature. This was done to ensure a unique time signature per sample. After the time split, a total of 8910 melodies are generated from the original 6403 melodies.

### D. Quantization

After time splitting, we have a sequence of notes each with corresponding start and end times and single time signature. To digitize the start and end timings, we first fix a quantization interval, the tempo - quarters per minute (qpm) and set the quantization steps per quarter note (4 in default case). Based on these divisions, we move the start or end times of the notes to the start/ end of the closest quantization step.

**Example:** Suppose qpm = 120 and steps_per_quarter = 4
By definition, we have 120 quarters in 1 minute i.e. 2 quarters per second. In other words, one quarter note would span a fixed interval of 0.5 seconds. As there are 4 steps in a quarter, the duration of one step would be one fourth of the duration of a quarter. Hence, one step spans 0.125 seconds i.e. when 0.125 seconds have passed, one step has to be incremented. We would then shift the start/end time of any given note to the beginning/end of the closest step.

### E. Melody Extraction

A single note in a quantized sequence could span multiple steps. We use two special events to better represent notes spanning multiple steps / periods of silence:

- no_event (-2): A note that has been playing will continue playing or if there is silence, silence will continue. This event is useful when there are notes spanning multiple steps. All the steps following such note's onset would be marked as -2. A note 67 starting at time step 4 and ending at 8 would be rerpresented as [67, -2, -2, -2] in event space.
- note_off_event (-1): Intervals where no notes are being played are marked by -1.

There are some default magenta parameters for melody extraction by which sequences can be discarded, truncated and split in different melodies. They are intuitive in their purpose and are listed below:

1) min_bars = 5: If the melody is shorter than the given number of bars, discard it.
2) max_steps = 512: If the melody is longer than the given number of steps, truncate it.
3) min_unique_pitches = 5: If the melody has less than the given number of unique pitches, discard it.
4) gap_bars = 1.0: If the melody has a gap (silence) of more than the given bars, split the melody from that point.

### F. Encoding

The role of encoding is to convert the pitches extracted above into a form that can then be input to the model. We have a set of pitches from a fixed pitch range [48-84) and two special events. The data is encoded to one-hot vectors for passing in as input to the model. The default Magenta encoder directly converts these notes into one-hot vectors belonging to one of the 38 classes. (48-84: 36 pitches from 3 octaves and 2 extra classes for -1 and -2 events described above). We propose different variants to this encoding mechanism which are described in the next section.

## III. Proposed encoding variants

### A. Neighbours distribution

The idea behind this encoding is to provide additional information to the model about the target. The model is trying to learn the output note (i.e the next note) distribution given a particular input node. Using the dataset, we can pre-compute this information. For instance, consider note C4. We can compute the probability distribution of next note given that this is the current note. This would be a vector of size $notes$. This vector would be used as an input in addition to the one hot vector denoting the current note.

To compute the next note distribution, counts of each of the next notes for every current note was calculated. By analysisng the data, it was observed that most of the notes (around 70%) in the training data were no_event. While calculating the distribution, if we divide count for each of the notes by total notes, the non special events (i.e. notes other than no event) would get normalised to very small values. To avoid this, no event and the other events were normalised separately using the formulae below.
For no_event:

$$no\_event\_dist = no\_event\_count/total\_count \quad (3)$$

For other events:

$$event\_dist = event\_count/(total\_count - no\_event\_count) \quad (4)$$

We pre-compute this distribution for all notes and pass it along with the one-hot vector indicating the current note being played.

As a variant of this, we ran an experiment where the distributions of the next 5 notes were passed in the model. For passing the next 5 notes distribution, we added the one-hot vector of current note to the immediate next neighbor distribution and appended the next 4 neighbours distributions as is. The resulting input vector was 5 times the original vector size.

### B. Consecutive pitch difference

The central idea behind this model is that any melody is defined by the difference of consecutive notes and not particular notes. Suppose we have a sequence, **[C4 E4 G4 A4]**. This can be seen in the form of difference of semitones as below.
[start_note, (+4 semitones), (+3 semitones), (+2 semitones)]

The start note can be any of the 12 root notes. Considering the above example, this same sequence can be played using a different combination of notes: **[D4 F4# A4 B4]**. Even after this change, the resulting note sequences would have very similar flavour.

While encoding, the special note events had to be kept as it is because pressing a key again to play the same note is different from long pressing a particular key.

The encoding mainly depends on the current event. If the previous event is ”continuation” or ”silence”, we take the last note played before it to take the difference.

**Example**: Let maximum allowed difference be 4.

| Notes | -2 | -2 | 72 | -2 | 72 | -2 | 71 | -2 | 74 |
|---|---|---|---|---|---|---|---|---|---|
| Diff | -2 | -2 | 0 | -2 | 0 | -2 | -1 | -2 | 3 |
| Encoded diff | 0 | 0 | 6 | 0 | 6 | 0 | 5 | 0 | 9 |

The special events remain unchanged and the rest of the notes get encoded as differences. The first note after silence is encoded as difference 0 as there is no reference to take a difference. This model did not work as expected and we explain reason for its failure in the next subsection.

### C. Pitch difference w.r.t. start note

Different note sequences with differences of consecutive notes from the start note remaining same sound very similar. Based on this observation, we can reduce the variations in the input data by encoding the notes information in terms of difference from the start note. So, if the start note is say C5 and the following sequence is [C5, D5, F5, B5, C5] - we encode this sequence as [0,2,5,11,0]. The start note is left out when creating training data.

The previous variant used the difference of consecutive notes instead of difference from the start note. However, that did not work out as expected. Two reasons for that are: 1. Multiple inputs mapping to the same set of outputs. 2. Within a song, 2 different sequences having same difference, such as [C4, C5, C8] and [D4, D5, D8] would be encoded in a similar way when they are quite different. This was resolved by using the difference from the start note as mentioned above.

### D. Bunch notes in input

The music data is a structured data. Instead of looking at just one previous note while predicting the next note, it would be helpful if more than one previous notes are sent as an input. When the input is just one note, the information about previous notes in the sequence is stored in the hidden states of LSTMs in the model to utilize for prediction. Instead of relying on the model's hidden states for storing the information, here we explicitly provide them in the model input.

We perform one-hot encoding of each of the notes in the bunch (here we take a bunch of 4 notes) and then append those one-hot vectors to form a final encoded input vector. Output of the model is next note in the sequence after the bunch of notes.

One-hot encoding is used to encode the output in all the above encoding variants.

## IV. Model Architecture

### A. Long Short Term Memory (LSTM) Networks

LSTMs [5] are special type of Recurrent Neural Networks which are capable of learning longer sequences. The recurrent networks store information about history of inputs in their internal states and use this information about the past input sequence while predicting the future sequence. LSTMs have a special structure with four gates (Figure 1) which gives them this powerful learning ability.

- i: Input gate, Learns whether to write to cell
- f: Forget gate, Learns whether to erase cell state
- o: Output gate, Learns how much to reveal to the cell state
- g: Update gate, Learns how much to write to cell state

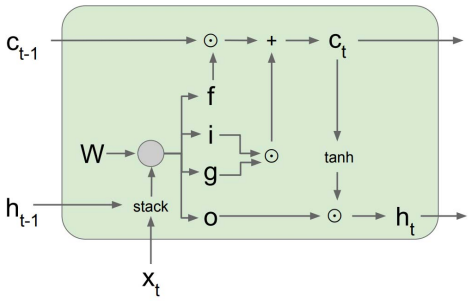


Fig. 1. LSTM Structure [19]

**LSTM Equations:**

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \quad (5)$$
$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \quad (6)$$
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \quad (7)$$
$$g_t = \tanh(W_{xg}x_t + W_{hg}h_{t-1} + b_g) \quad (8)$$

Cell State: $c_t = f_t \cdot c_{t-1} + i_t \cdot g_t$
Output State: $h_t = o_t \cdot \tanh(c_t)$

The equations above describe a single layer of LSTM. We can have a stacked LSTM structure where the output of layer-1 acts like the input of layer-2 and so on. The stacked structure helps to learn more complex patterns in the data.

### Melody RNN

We followed the same architecture as done in magenta basic RNN model which uses a 2 layered LSTM with 128 nodes each (shown in Figure 2). It uses an LSTM as the recurrent layer and with dynamic unrolling so it goes back to the full length of the input song (upto 512 steps) while back propagating.

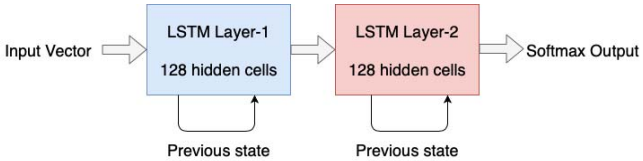


Fig. 2. Melody RNN architecture

*Training Procedure*

Following the steps of the pipeline explained in the previous section, training samples are generated which are the songs encoded as inputs and labels in form of one hot vectors.

While training the model (Figure 3), we provide the encoded input, predict the softmax probabilty for next note and try to minimize the softmax loss. The softmax loss given the true output and predicted class probabilities is given by

$$loss = -log(p_{correct\_class}) \tag{9}$$

The class having maximum softmax probabilty is treated as the predicted class for accuracy calculation while training.

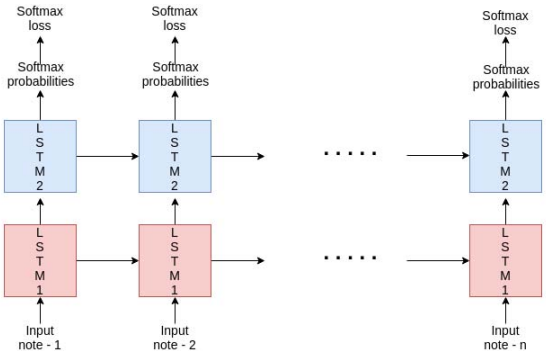


Fig. 3. Training LSTM on music sequence

*B. Generating multiple possible output sequences given the same primer*

For any classifier, given the current input and state, the LSTM provides an output distribution which is indicative of the probability of each note being the next note in the sequence. As an example, say there are 3 classes A, B and C and the predicted output is 0.4 for A, 0.2 for B and 0.1 for C. Then, these can be interpreted as probabilities of each of these being the next notes in the sequence. The usual procedure for classifiers is to pick the predicted output as the one with the maximum probability (A in the above case) at each step. Doing this would make the procedure deterministic and would not allow generation of different sequences from the same starting note.

Since we wish to generate different possible yet likely outcomes starting from the same primer sequence, we follow the probability based procedure as done in the magenta library. Instead of picking out the class with the max probability, we sample based on the distribution. This means for the example shown above - A is 4 times as likely and B is twice as likely to get picked as output as compared to C. This ensures that A does not get picked every time and also that the outputs are being picked as per the distribution predicted by the model. Process is shown in Figure 4.

## V. Results

Table III describes the training accuracy and training loss for all the different models used for music generation. Each

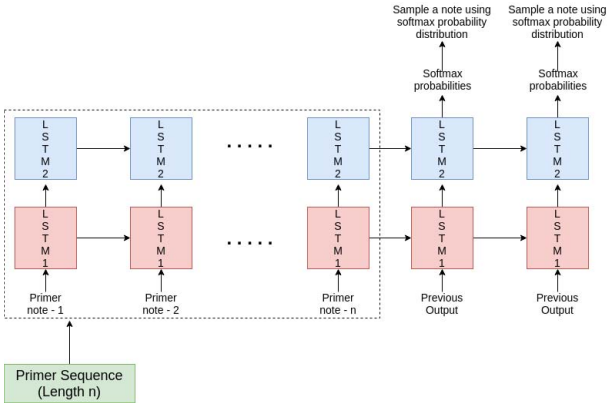


Fig. 4. Generating sequences using trained model

| Model name | Model | Acc | EA | NEA | Loss | PPL |
|---|---|---|---|---|---|---|
| Melody RNN | [128, 128] | 0.79 | 0.25 | 0.98 | 0.69 | 1.99 |
| Melody RNN | [256, 256] | 0.84 | 0.41 | 0.98 | 0.52 | 1.68 |
| Melody RNN | [512, 512] | 0.97 | 0.89 | 0.99 | 0.09 | 1.09 |
| Next Note | [128, 128] | 0.68 | 0.019 | 0.89 | 1.63 | 5.10 |
| Next Note | [256, 256] | 0.85 | 0.49 | 0.98 | 0.49 | 1.63 |
| Neighbor Dist | [128, 128] | 0.79 | 0.20 | 0.98 | 0.73 | 2.07 |
| Neighbor Dist | [512, 512] | 0.95 | 0.83 | 0.99 | 0.14 | 1.15 |
| Pitch Diff | [128, 128] | 0.78 | 0.25 | 0.98 | 0.76 | 2.15 |
| Start note Diff | [128, 128] | 0.81 | 0.27 | 0.99 | 0.64 | 1.89 |
| Bunch Note | [256, 256] | 0.87 | 0.51 | 0.99 | 0.43 | 1.53 |
| Bunch Note | [512, 512] | 0.97 | 0.88 | 0.99 | 0.11 | 1.11 |

TABLE III
RESULTS TABLE

of them trained for 100,000 steps. The melody RNN is the basic model and all other models are our proposed variations. Samples generated by these models can be found on this link.

Following are the evaluation metrics calculated:

- Accuracy (Acc): The percent of predictions matching the original output.
- Event accuracy (EA): The accuracy of events other than "no event". It is separately computed as the "no event" class forms a large part of the prediction sequence.
- No event accuracy (NEA): This metric describes the percent of no events in the original sequence that are correctly predicted. This value is generally high because most of the notes in the input data and prediction are "no event".
- Loss: The loss function used is softmax cross entropy loss function shown in Eq 9.
- Perplexity (PPL): It is defined as the exponential of softmax loss.

$$perplexity = e^{loss} \tag{10}$$

*Subjective Analysis*

Apart from the quantitative metrics, we have employed a subjective analysis method to evaluate different models. For this we took 4 samples using different primer sequence from each of the 4 models and asked 10 subjects (a mix of music domain experts and non-experts) to rate these melodies on a

| Model name | Model architecture | Average Rating |
|---|---|---|
| Melody RNN | [128, 128] | 3.315 |
| Next Note Distribution | [128, 128] | 3.215 |
| Neighbor Distribution | [128, 128] | 2.865 |
| Start note Pitch Diff | [128, 128] | 2.615 |

TABLE IV
SUBJECTIVE ANALYSIS OF MODELS

scale of 1 to 5 (least melodious to most melodious). All the melodies generated are 32s (256 steps) long with a primer which is 8.75s (70 steps) long. The average ratings of the different models by the curators is shown in Table IV.

## VI. CONCLUSION

We presented baseline results on magenta models and variants of the models trained on different hyperparameters which can be helpful for quantitative analysis. We also introduced variants of the encoding techniques based on a) Additional information which can be useful to LSTMs for predictions and b) The observation that the melody is relatively independent of the starting note, and the difference in pitches of notes from the start note preserves the characteristics of a song. These variants produce melodies which are comparable or better in perceptual quality compared to the baseline magenta models.

## VII. DISCUSSION AND FUTURE PERSPECTIVES

### A. Attention RNN

Attention mechanism in RNNs has proved effective at learning the alignment between modalities such as objects in images to corresponding captions [20] or part of sentences during translation [21]. Same can be utilized in music generation models.

### B. Beam Search for sequence generation

Beam search [22] can be used to get different output sequences from the same primer sequence. It generates multiple beams of next outputs at each step depending on the beam width. For example, for a beam width of 3, at any time t, we would pick the top 3 most likely next notes for all 3 most likely sequences till that time. Then, from all 9 outputs, we would pick the 3 with the highest likelihood and repeat the procedure iteratively for the next steps.

## REFERENCES


[1] J.-P. Briot, G. Hadjeres, and F. Pachet, "Deep learning techniques for music generation-a survey," *arXiv preprint arXiv:1709.01620*, 2017.

[2] C. Donahue, H. H. Mao, Y. E. Li, G. W. Cottrell, and J. McAuley, "Lakhnes: Improving multi-instrumental music generation with cross-domain pre-training," *arXiv preprint arXiv:1907.04868*, 2019.

[3] "MIDI Documentation." https://mido.readthedocs.io/en/latest/about_midi.html. [Online; accessed 19-July-2019].

[4] "MusicXML Documentation." http://usermanuals.musicxml.com/MusicXML/MusicXML.htm, 2015. [Online; accessed 19-July-2019].

[5] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[6] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[7] K. Choi, G. Fazekas, and M. Sandler, "Text-based lstm networks for automatic music composition," *arXiv preprint arXiv:1604.05358*, 2016.

[8] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[9] A. Nayebi and M. Vitelli, "Gruv: Algorithmic music generation using recurrent neural networks," *Course CS224D: Deep Learning for Natural Language Processing (Stanford)*, 2015.

[10] J.-P. Briot and F. Pachet, "Music generation by deep learning-challenges and directions," *arXiv preprint arXiv:1712.04371*, 2017.

[11] Y.-A. Chung, C.-C. Wu, C.-H. Shen, H.-Y. Lee, and L.-S. Lee, "Audio word2vec: Unsupervised learning of audio segment representations using sequence-to-sequence autoencoder," *arXiv preprint arXiv:1603.00982*, 2016.

[12] H.-W. Dong, W.-Y. Hsiao, L.-C. Yang, and Y.-H. Yang, "Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[13] D. Eck and J. Lapalme, "Learning musical structure directly from sequences of music," *University of Montreal, Department of Computer Science, CP*, vol. 6128, 2008.

[14] L.-C. Yang, S.-Y. Chou, and Y.-H. Yang, "Midinet: A convolutional generative adversarial network for symbolic-domain music generation," *arXiv preprint arXiv:1703.10847*, 2017.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[16] "Magenta Documentation." https://magenta.tensorflow.org. [Online; accessed 30-July-2019].

[17] "Wikifonia Dataset." http://www.synthzone.com/files/Wikifonia/Wikifonia.zip. [Online; accessed 19-July-2019].

[18] "Summaries and Tensorboard." https://www.tensorflow.org/guide/summaries_and_tensorboard. [Online; accessed 19-July-2019].

[19] "CS231n lecture notes." http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf, 2017. [Online; accessed 19-July-2019].

[20] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *International conference on machine learning*, pp. 2048–2057, 2015.

[21] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[22] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3156–3164, 2015.


## VIII. APPENDIX

### A. Music Terminology

- note: In music, a note is the pitch and duration of a sound, and its representation in musical notation. For example, C4.
- Octave: An octave is the interval between one musical pitch and another with double its frequency.
- bar: It is a segment of time corresponding to specific number of beats. Typically, a piece consists of several bars of the same length, and in modern musical notation the number of beats in each bar is specified at the beginning of the score by the time signature.
- time_signature: A number denoted in form numerator over denominator - numerator showing number of beats in a bar and denominator showing what kind of notes constitute one beat (1 quarter note constitutes a beat if denominator is 4).