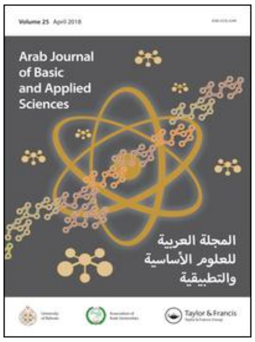

Arab Journal of Basic and Applied Sciences
المجلة العربية للعلوم الأساسية والتطبيقية
Taylor & Francis

Taylor & Francis
Taylor & Francis Group

Arab Journal of Basic and Applied Sciences

ISSN: (Print) 2576-5299 (Online) Journal homepage: https://www.tandfonline.com/loi/tabs20

Generation of music pieces using machine learning: long short-term memory neural networks approach

Nabil Hewahi, Salman AlSaigal & Sulaiman AlJanahi

ARAB JOURNAL OF BASIC AND APPLIED SCIENCES
2019, VOL. 26, NO. 1, 397–413
https://doi.org/10.1080/25765299.2019.1649972

University of Bahrain

Taylor & Francis
Taylor & Francis Group

ORIGINAL ARTICLE

OPEN ACCESS Check for updates

Generation of music pieces using machine learning: long short-term memory neural networks approach

Nabil Hewahi, Salman AlSaigal and Sulaiman AlJanahi

Department of Computer Science, College of Information Technology, University of Bahrain, Zallaq, Bahrain

ABSTRACT

In this article, we explore the usage of long short-term memory neural network (NN) in generating music pieces and propose an approach to do so. Bach's musical style has been selected to train the NN to make it able to generate similar music pieces. The proposed approach takes midi files, converting them to song files and then encoding them to be as inputs for the NN. Before inputting the files into the NNs, an augmentation process which augments the file into different keys is performed then the file is fed into the NN for training. The last step is the music generation. The main objective is to provide the NN with an arbitrary note and then the NN starts amending it gradually until producing a good piece of music. Various experiments have been conducted to explore the best values of parameters that can be selected to obtain good music generations. The obtained generated music pieces are accepted in terms of rhythm and harmony; however, some other problems exist such as in certain cases the tone stops or in some other cases getting short melodies that do not change.

ARTICLE HISTORY
Received 20 January 2019
Revised 8 June 2019
Accepted 26 July 2019

KEYWORDS
Music pieces; long short-term memory neural networks; machine learning

1. Introduction

Music generation is very important nowadays. It can be used in many applications. Musicians or artists build on what is generated by the machine and produce their own original work. The music or art generated by software is also sometimes sold by the companies or individuals who designed them. They can be licensed to corporations and retail for use in advertising, shop music etc. An example of this is *Aiva*, an artificial intelligence music generator which has already released its own copyrighted albums filled with generated soundtracks in the electronic music genre (Travers, 2018). For generating art, a painting generated by a GAN network named Portrait of Edmond Belamy was auctioned for $432,500; this is an indication of the value in using neural networks (NNs) for generating art or music (Christies.com, 2018). Another example is Jukedeck which can produce music based on the genre and beats per minute specified by the user. These all of the tracks generated by these AI composer products can be licensed for corporate or personal use (Travers, 2018). Automating the creative process of the human means that companies can get multimedia products faster and cheaper. That is why the music generator is such an important application of machine learning techniques.

Automated music composition has been a widely researched method of producing new music for many years now. Prior to machine learning the most popular approach to generate music was generative grammars. These previous methods produced simplified music pieces with noncomplex melodies and many inadequacies like predictable repeating melodies. Currently, the most recent (and arguably the best) tool for generating music is NNs (Agarwala, Inoue, & Sly, 2017). NNs can be trained with music like Bach's in order to generate its own original music pieces; the network learns the pattern of the inputted music and then uses that to generate its own. A good example of this is DeepBach (Hadjeres, Pachet, & Nielsen, 2017) and the BachBot (Liang, 2016) that utilize recurrent NNs. These music generators were able to produce music similar to Bach's musical style after being trained with hundreds of his chorales (Hadjeres et al., 2017). This article to implement a piano music generator based on Bach's style using a long-short term memory (LSTM) NN. We limit ourselves to Bach's musical style because Bach's style is a well-known and famous style. Considering more than one style will make the learning process more complicated and the NN might lose a style or part of it during learning another style, this might happen because of the big

CONTACT Nabil Hewahi nhewahi@uob.edu.bh Department of Computer Science, College of Information Technology, University of Bahrain, Zallaq, Bahrain

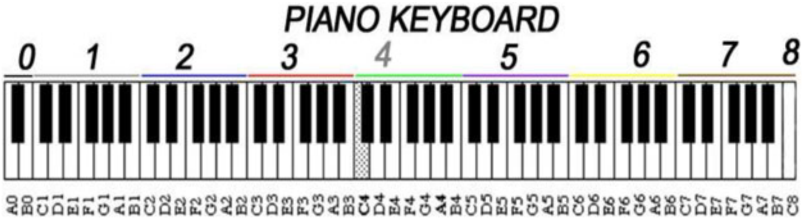

PIANO KEYBOARD
0 1 2 3 4 5 6 7 8
A0 B0 C1 D1 E1 F1 G1 A1 B1 C2 D2 E2 F2 G2 A2 B2 C3 D3 E3 F3 G3 A3 B3 C4 D4 E4 F4 G4 A4 B4 C5 D5 E5 F5 G5 A5 B5 C6 D6 E6 F6 G6 A6 B6 C7 D7 E7 F7 G7 A7 B7 C8

Figure 1. Piano keyboard (Tornblom, 2008).

Figure 2. Synthesia screenshot.

variations in different musical styles. The obtained results will be later presented and discussed in lights of the results of other approaches. In this introduction, we introduce some basic concepts of music and principles of LSTM to pave our explanation for the proposed approach.

1.1. Music background information

Understanding how our music generator works requires background knowledge in some basic concepts of music theory. This section entails a very brief summary of essential music terminologies. The term pitch is used to describe the highness or lowness of a sound. In music there are seven notes (or 'sounds') which are A, B, C, D, E, F, G. Each set of notes (A to G) belongs to an octave. Octaves are used to designate which pitch range the notes are in. For example, A1 (A in octave 1) will be much deeper and lower than A7 (A in octave 7) which is higher pitch and sharper sounding (Richer, 2010). A common piano keyboard consists of 88 playable keys and seven whole octaves as is shown by the colours in Figure 1.

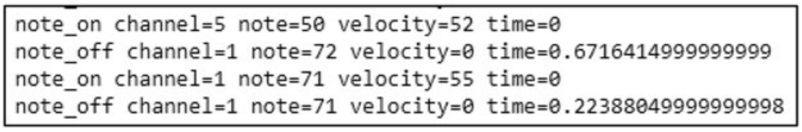
The majority of music pieces are in a key. A key is the note which forms the basis (or 'home') for the music in the piece. A piece played in the key of E means that the note E is the base of the music in the song. A scale is a specific set of notes built on the base of the key. This can be described as a specific toolset of notes with which to create a song (Learningmusic.ableton.com, 2018). Scales are different from each other in that some are more suitable to 'happier' or 'brighter' music (C major scales) and some are better for 'sad' or 'dark' pieces like the D or E minor scales.

A chord is a group of notes played simultaneously producing a distinct sound as a group; for example, the A Major chord is composed of an A, a C sharp and an E note (Richer, 2010). In music, transposition is the process of 'shifting a melody, a harmonic progression or an entire musical piece to another key, while maintaining the same tone structure' (Koch & Dommer, 1865). This means that the song's structure is preserved while the key is changed.

1.2. MIDI File format

The Musical Instrument Digital Interface format (MIDI or .mid) is used to store message instructions which contain note pitches, their pressure/volume, velocity, when they begin and when they end, phrases etc. It does not store sound like audio formats, but rather instructions on how to produce sound (Midi.org, 2018). These instructions can be interpreted by a sound card which uses a wavetable (table of recorded sound waves) to translate the MIDI messages into actual sound (Rouse & Good, 2005). It can also be interpreted by midi player studio software like Fruity Loops (FL) Studio or popular sequencers like Synthesia (see Figure 2). Musical notation software like MuseScore or Finale can translate midi into editable sheet music; this allows users to write music in normal music notation on their computers and then listen to it by MIDI players without ever needing an instrument.

MIDI messages are separated into system and channel messages. Since we are primarily concerned with the notes, we focus only on the channel messages. There are 16 possible MIDI channels (16 streams of channel messages containing notes to be played). Each MIDI channel message contains the type of the message, the time and the note number. The messages may contain other information as well, however, for this paper only the basic note messages are considered. The large bulk of MIDI



```
note_on channel=5 note=50 velocity=52 time=0
note_off channel=1 note=72 velocity=0 time=0.6716414999999999
note_on channel=1 note=71 velocity=55 time=0
note_off channel=1 note=71 velocity=0 time=0.22388049999999998
```

Figure 3. MIDI messages.

messages are mostly note-ons and offs. A note-on indicates that a key is being pressed (a note is being played). That note will continue to play until the note-off message is received, which indicates that a key is no longer being pressed. Within the channel message is the note number (i.e. middle C = 60) and other data like the velocity (volume of the note). It is worth noting that some MIDI messages are written with note-on data type and velocity = 0 as an alternative to note-off, however both mean that the note has stopped playing (Bello, 2018).

In Figure 3, the first message indicates that a note having pitch = 50 and velocity = 52 is being played. The time in note-on messages is zero because time indicates how much time the key has been playing. The note-off message after it indicates that note 72 has stopped after taking 0.67 time (custom midi time, not seconds). After that a note on message plays note = 71. It is then stopped by the note-off message after since it contains note = 71. The time this note took was 0.223.

1.3. Neural networks

A NN is defined as 'an interconnected assembly of simple processing elements (units or nodes)'. The processing in the network relies on the weights of the connections between those nodes which are trained by or adapt to the training dataset; this process is what is often called training the network or learning (Gurney, 2004). NNs are an ideal tool for classification problems. The benefits of NNs in classification are due to several things. NNs are adaptive because they adjust according to the data inputted. They are capable of identifying the functions which relate attributes to categories with some degree of accuracy. They are flexible enough to model complex real world classification problems like bankruptcy prediction, image and speech recognition etc. (Zhang, 2000). Not all NNs are the same however, and the most common types of NNs are split into two broad categories: feed-forward (FF) and recurrent (RNN). In FF NNs the activation flows via the input layer to the output layer without any form of recursion. There is only one hidden layer in between them; the name 'deep' FF NNs mean that there are more hidden layers but with the same concept (Tchircoff, 2017). The nature of FF NNs is unsuitable to music generation because there is no context/memory. No past state of the NN can affect the future states of the weights in the network. Musical melodies are essentially contextual and rely on memory of past notes in order to generate new notes that are coherent and tonal (pleasant sounding). This is why FF NNs are completely unsuitable for music generators.

1.3.1. Recurrent NNs

RNNs have recurrent connections within the hidden layers between previous and current states in the NN. This means that it is storing information in the form of activations thereby offering a kind of memory. This capability of storage makes it very useful for things like speech processing and music composition (Wiest, 2017). The main problem with a regular RNN is that it stores the information of only the state before; this means the context extends only one generation back. This is not very useful in music composition where the beginning of the song plays a role in the middle and the end as well.

1.3.2. LSTM NNs

LSTM networks solve the problem of a lack of long-term memory in regular RNNs. LSTM networks add a special kind of cells named memory cells. These memory cells consist of three gates: input, output and forget. Input gates control how much data is inputted into memory, output gate controls the data passed to next layer and the forget gate controls the loss or tearing in the stored memory (Tchircoff, 2017). These gates contain sigmoid and hyperbolic tangent functions with a low computational complexity (i.e. O(1)) meaning it will not be slow during training (Wiest, 2017). The structure of a memory cell can be seen in Figure 4. Each gate contains a function, the middle circle is the memory stored and S circles are sigmoid activation functions.

The function of these memory cells is that they extend the memory of a regular RNN and allow the network to read, write and delete stored information based on importance (Donges, 2018). This allows the networks to store long-term memory which is especially useful in speech recognition and music composing, making it ideal for most music generators. It will help in recognition of melody structure and proper note sequencing.

Figure 5 shows a basic LSTM network structure where each h node represents an LSTM memory cell and each column represents a stage of the NN. In this network, the output of each stage is mapped to

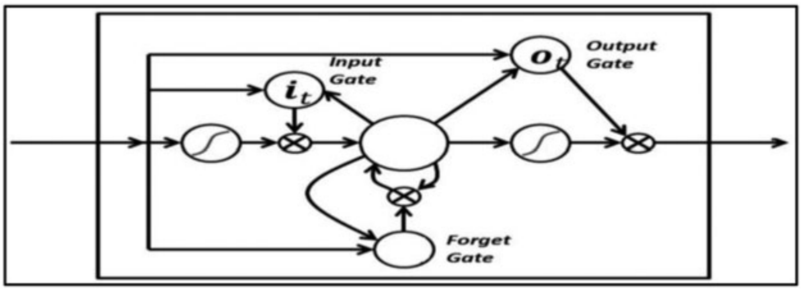


Figure 4. LSTM memory cell (Donges, 2018).

the input of the next stage, and the arrow between the memory cells represents the stored data kept between stages. This structure is ideal for music generators where a random note is inputted, and the predicted output note is used as input for the next stage thereby creating a sequence of notes.

2. Literature review and background

The automation of music composition has been implemented in several different ways over the past decade. Some of the oldest methods used were rule-based systems (grammars) and algorithms that utilize randomness like the 'riffology' algorithm presented by Langston (1988). One popular form of grammars used for music was L-Systems. These string-rewriting grammars were capable of producing relatively simple note sequences (McCormack, 1996). The focus for music composition and music generation in more recent times has shifted towards machine learning and artificial intelligence techniques. This is due to the capability of NNs and deep learning to produce less predictable and more melodically complex melodies than their random algorithm and grammar systems counterparts. The majority of music generators in the past few years have been created using LSTM networks. They differ, however, in the encoding of inputs from MIDI formats and some aspects of the NN structure. Eck and Schmidhuber (2002) were the first to switch from regular RNNs to LSTMs to generate blues music. Blues is a genre of music similar to jazz containing a lot of guitar solo note sequences known as riffs. The network was successful in generating music pieces with a proper structure resembling normal blues music. They did not map this to MIDI files but instead used their own format for representing note sequences. Their LSTM network proved to be capable of learning chord sequences and recognizing blues chord structure. More recently, Skuli (2017) created a music generator with a dataset consisting of Final Fantasy (the video game) MIDI tracks. This is similar to our work in that Skuli used MIDI's to train an LSTM NN and then generate his own MIDIs. He used Keras, which is a high-level deep learning Python API that runs on top of Tensorflow which simplifies interactions with the machine-learning library. The encoding of MIDI files to inputs for the network was done via a Python toolkit named Music21, whereas our encoding to our own format was entirely manual. Skuli inputted note and chord object sequences and used the network to predict the next object (note or chord) within the sequence. His results were relatively good since the music had structure and very few atonal notes. Another approach to using LSTMs for music was done by Walder and Kim (2018). They recognized that human music is largely self-similar in that a motif is repeated and transformed throughout a song. A motif is a theme expressed as a short musical phrase (a succession of notes) which can be found repeating throughout a given song. The goal of their motif network was to recognize motifs and motif transformations within a song sequence using concepts of motif transposition and shifting. This approach led to a higher rate of recognition in the regularities in a music sequence. A more successful but less automated approach to use LSTM networks for generating music is to start with a pre-defined musical rhythm and use the network to choose the pitches of the notes within it. This was done by Walder (2016) where he addressed probabilistic modelling of music data. One technique he used to augment the dataset was to transpose the training data into all 12 keys, which he did to avoid the non-trivial problem of recognizing music keys in the network. This was a technique which we used for our dataset as well. Our LSTM network structure has some common features with Skuli (2017) and Walder's (2016) LSTM NNs. Skuli's method of making the LSTM network predict the next note based on the previous note sequence is similar to ours, but the encoding of the data is different. It differs however in the layers/layer types (drop out layers, LSTM layers etc.) and activation functions as well as the note range (layer lengths), and method of encoding the data.

To sum up, in our work, we avoided to use concepts of Generative Adversarial Network (GAN) (Skymind, 2017). GAN is composed of a generative network and discriminator network. The generator receives random vectors as an input and then generates a new output. The discriminator has a sample

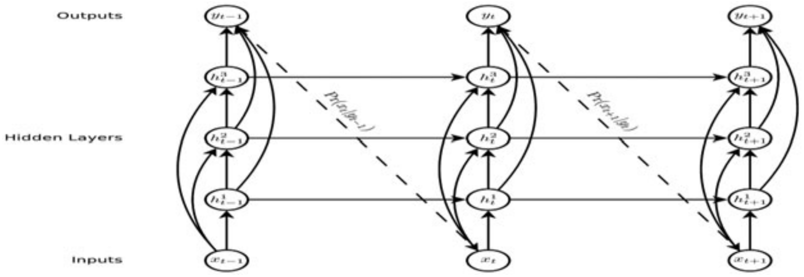


Figure 5. Three-layer LSTM network (Graves, 2014).

dataset which then determines whether the generated output is considered a sample or a fake result; it predicts the category of this generated output. The goal of this network structure is to generate outputs that are passable as part of the original dataset (Skymind, 2017). The main problem of GAN is the instability of training between the two networks. This was demonstrated by Agarwala et al. (2017) where the GAN network did not generate any meaningful output because the model was considered too unstable. It will only work if there is some kind of training balance (Tchircoff, 2017).

Eck and Schmidhuber (2002) earlier demonstrated the suitability of an LSTM network by proving its capability to recognize chords. However, for simplifying the interface with the machine-learning library Tensorflow, Skuli (2017) demonstrated how Keras could be used to create a music generator with LSTMs. We follow Skuli (2017) technique because of the demonstrated quality of music produced. The network model for recognizing motifs by Walder and Kim (2018) would have been a logical next step for work because of its capability to produce more structured melodies. One of the major drawbacks of Skuli's network was that the notes had no varying durations; the pieces had fixed time intervals and offsets. Our network, on the other hand, had varying note durations to increase musical depth and complexity. A major difference between our approach for encoding the MIDI data was the interpretation of chords. The models proposed by Skuli (2017) and Eck and Schmidhuber (2002) encode MIDI data into a sequence of note and chord objects before being inputted to the network. Our custom encoding method increased the complexity of the model by using a sliding window to capture given notes within a time interval without making a distinction between single notes or chords.

3. The proposed architecture

In this paper, we propose the flow shown in Figure 6.

3.1. MIDI files conversion to song format

All midi files have been converted into an intermediary format (Song Format) that we developed which helps us deal with the notes in a more natural way. While looping over the events of each file, notes being played in all the 16 MIDI channels are recorded into the Song Format. It is worth noting that sometimes MIDI files would use a 'note_on' message with 'velocity' 0 to indicate a 'note_off'. Each MIDI event contains a 'time' attribute which holds a number corresponding to the time (in ticks) since that last message. The Song Format is a set of two classes that have the job of holding the note information and converting to and from MIDI files. So, the functionality of exporting and importing MIDI into the encoded format is done inside the Song Format class. Moreover, since MIDI has a note range of [0, 127], and we are only interested in the subset that is the piano keys (the subset [21, $21+88$] in MIDI), the song format conversion process handles this shift from MIDI space to Piano space. The process is shown in Figure 7.

3.2. Encoding

Since there is no way to feed the network with midi values directly and be able to generate music, it becomes important to encode the music data into a format that is usable by the network. The output of the encoding method used is a matrix of binary vectors of size $N \times 89$ for each MIDI file, where N is the number of vectors in the piece. The first 88 bits in the vector correspond to the state of the piano keyboards 88 notes. A zero indicates that a note is not being played at this instant, while a one indicates the opposite. The 89th bit in the vector is the repetition count. The repetition count tells us how many times this vector is repeated; that is, how long this current state of the keyboard is preserved until it changes. In our initial experiments, the repetition count was a natural number. However, having a natural number that is not normalized will mess up the loss calculation and convergence. So, at the end we used a real number in the range (0,1]. Where a number close to 0 meaning less repetitions, and

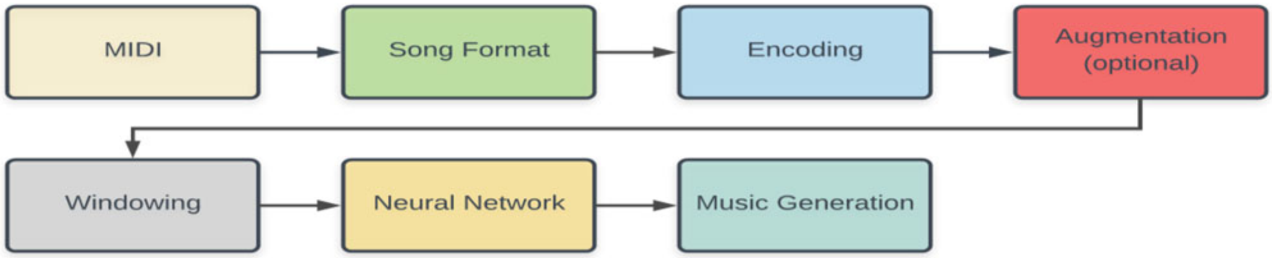
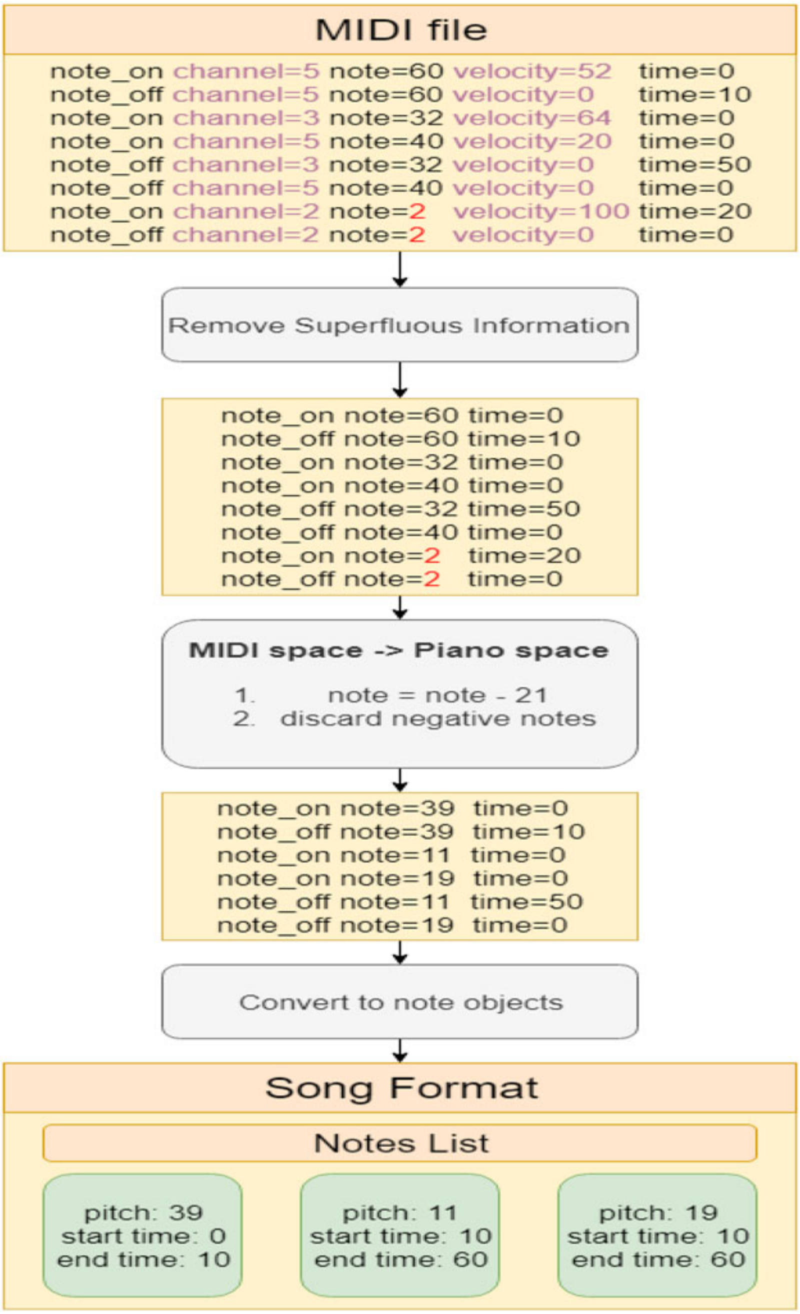


Figure 6. Proposed architecture.



```
note_on channel=5 note=60 velocity=52  time=0
note_off channel=5 note=60 velocity=0  time=10
note_on channel=3 note=32 velocity=64  time=0
note_on channel=5 note=40 velocity=20  time=0
note_off channel=3 note=32 velocity=0  time=50
note_off channel=5 note=40 velocity=0  time=0
note_on channel=2 note=2  velocity=100 time=20
note_off channel=2 note=2  velocity=0  time=0
```

```
note_on note=60 time=0
note_off note=60 time=10
note_on note=32 time=0
note_on note=40 time=0
note_off note=32 time=50
note_off note=40 time=0
note_on note=2  time=20
note_off note=2  time=0
```

```
note_on note=39  time=0
note_off note=39  time=10
note_on note=11  time=0
note_on note=19  time=0
note_off note=11  time=50
note_off note=19  time=0
```

Figure 7. Song format conversion process.

1 meaning it has the most inside it's MIDI file. The intervals are half-open because no vector should have a repetition value of zero, since it is semantically wrong. The least repetition value for a file will be $\frac{1}{\text{Maximum Repetition}}$.

We consciously chose to make the repetition count local instead of global; that is to have 1 be the largest note in the file and not in the whole dataset. Our reasoning is that the repetition count semantically answers 'How long is this note being

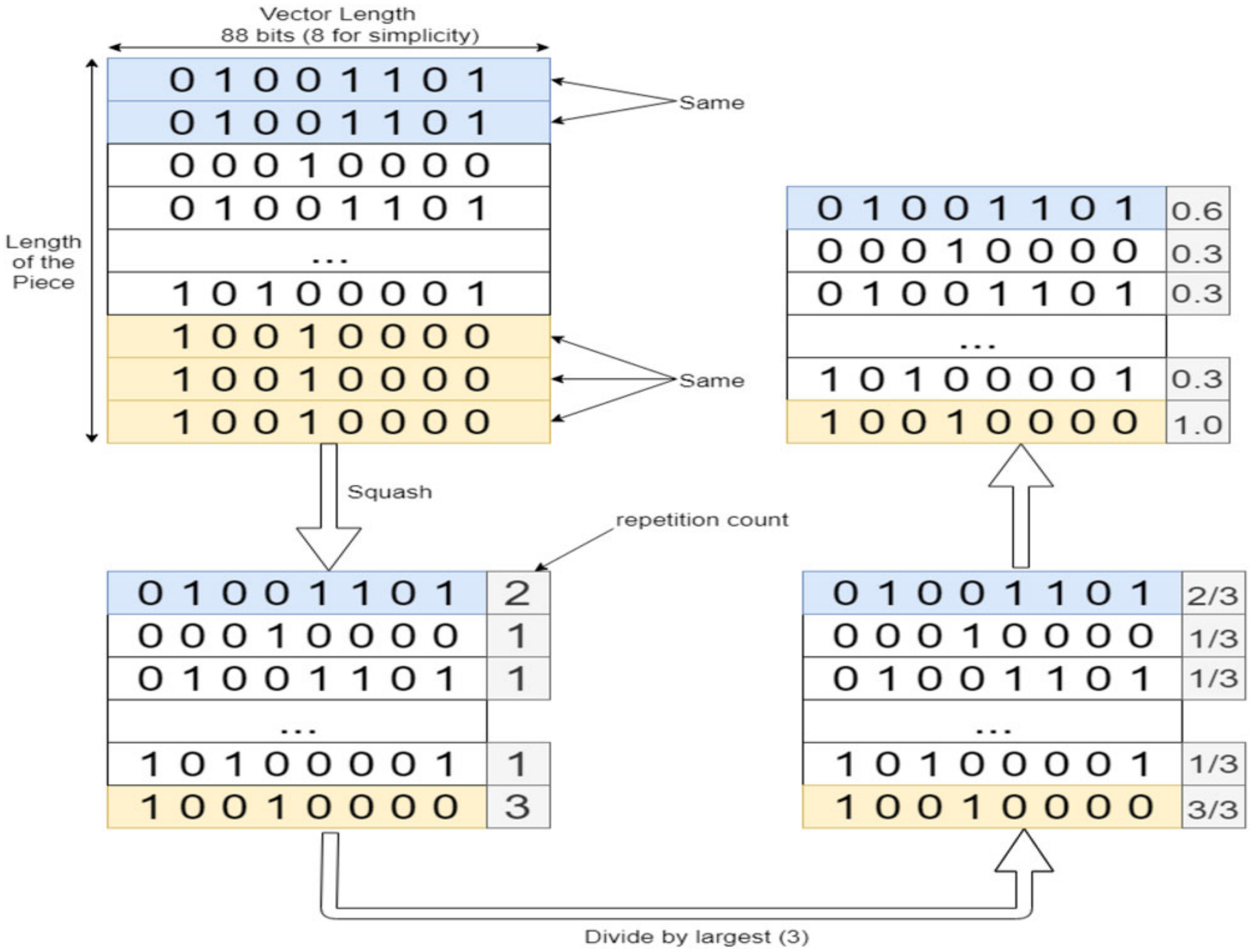


Figure 8. Encoding method.

played relative to other notes in its file?', and this number shouldn't be affected by other vectors in other pieces, even though all these vectors will enter the training process. That is to say, a long note in a file of short notes should not have a value less than a long note in a file of long notes. Figure 8 shows the used encoding method and shows how repeated vectors (The colored vectors) are getting squashed together. The number gets recorded in the repetition count (grey). The repetition count is then normalized by dividing it by the largest repetition count.

3.3. Data augmentation

A certain melody can be played in any different key with very different notes but have the same semantic meaning and be heard as the same melody. Figure 9 shows in musical notation how the same melody can be played in two musical keys. In the first line, the melody is played in the 'C major' musical key, while in the second it is played in the 'F major' musical key. These two melodies are equivalent if heard separately, even though they have different notes entirely. Figure 9 shows in musical notation how the same melody can be played in two musical keys. In the first line, the melody is played in the 'C major' musical key, while in the second it is played in the 'F major' musical key. These two melodies are equivalent if heard separately, even though they have different notes entirely. To solve this problem, augmentation has been used on our pieces.

As shown in Figure 10, after encoding, we make Augmentation Count (Ac) copies of each melody and transpose them randomly by doing a right shift. That is, we change the key of each copy such that we get Ac distinct copies each in a different key. The reason for the use of random permutations is to be able to do less than 11 augmentations without biasing the keys in the dataset. That is, if we did three augmentations without permutations, we would end up with data that is transposed to keys +1, +2 and +3, and this would create an imbalance in the dataset where the keys [+4, +11] would be underrepresented.

3.4. Timesteps

Since LSTMs expect the input shape to be in the format (timesteps, feature count), we must transform our data to be in this format. To do this we use a 'Sliding Window' method so that at each instance in the training, the sample contains itself and the previous N samples as shown in Figure 11.

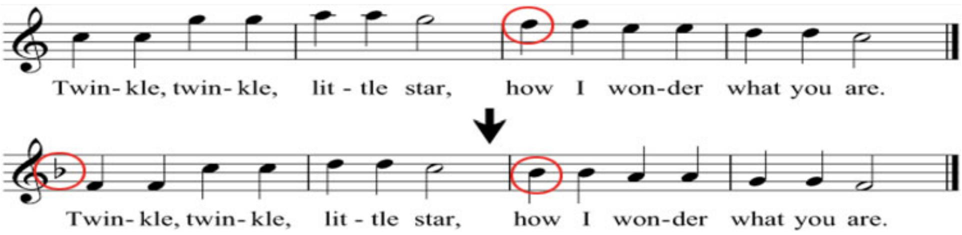

Twin- kle, twin- kle, lit - tle star, how I won- der what you are.
Twin- kle, twin- kle, lit - tle star, how I won- der what you are.

Figure 9. Transposition from C major to F major (Clementstheory.com, 2018).

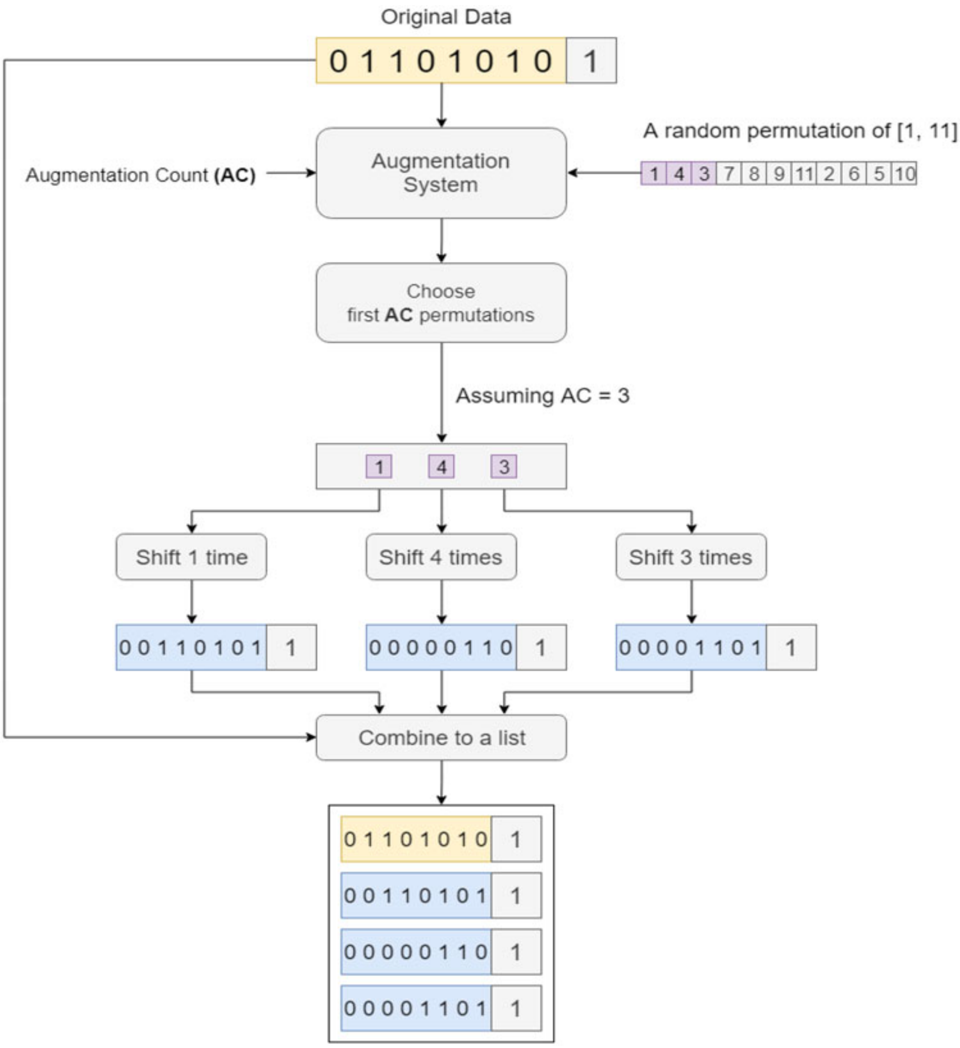


Figure 10. Augmentation process.

3.5. Neural network

Having the windowed encoded dataset, it is fed into the NN. We separate the examples and labels such that we give an example at sample n with its timesteps in the range $[n - w, n - 1]$ and let the NN to predict the sample at $n+1$.

Figure 12 uses a window from the data generated from the windowing in Figure 11 and shows how this window will be fed into the network.

3.6. Generating music

At fixed intervals during the network training, the network generates music and then save it in MIDI format for the purpose of listening to it and analyzing it. Figure 13 shows how the music gets generated after training. To generate music, the network is fed with a matrix of either noise, or a random matrix from the dataset and ask it to predict the next vector. After predicting the next vector, we append this vector to the matrix as well as the list of generated vectors and pop the last vector from the matrix. Similarly, the same process is performed to predict the next vector. This is repeated several times until a piece of the desired length is obtained.

4. Experimental results

In this section, we present our used datasets, NN architecture and the experimentations.

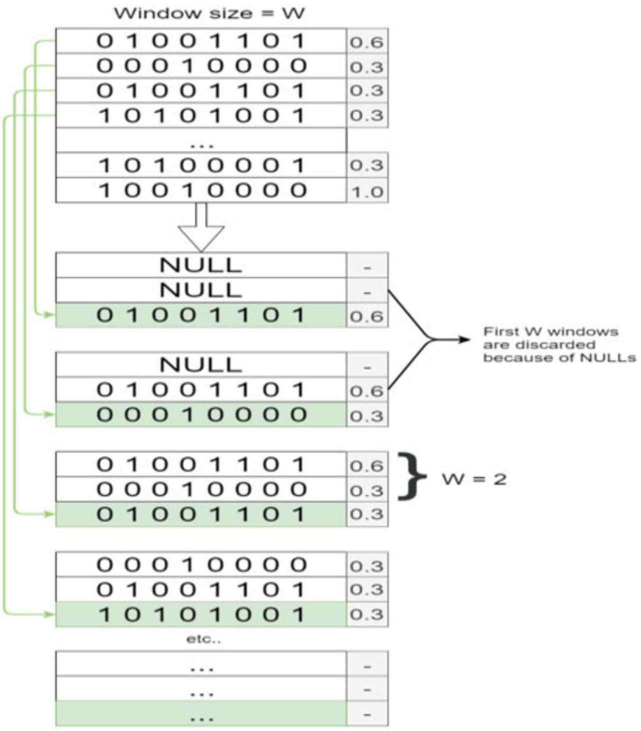


Figure 11. Sliding window.

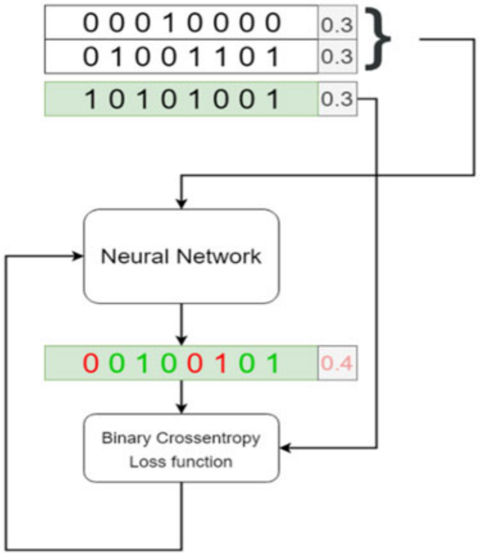


Figure 12. NN training.

4.1. Dataset

Bach's 'Well-Tempered Clavier Book II' has been chosen as our dataset (Classicalarchives.com, 2018). This is because of the systematic style of Bach's music, as well as the special nature of this work in which Bach writes two pieces of music for every key. The dataset contains 24 files, each file containing two pieces. Figure 14 shows a normalized histogram of the count of the 12 keys in the dataset. Each occurrence of a key is counted regardless of the octave. This shows that all the keys occur with the same degree, that is there are no predominant keys in the dataset. The standard deviation of the normalized histogram is 0.02683. This shows that on average the difference in the number of occurrences between any two keys in the dataset is around 2.6%. Figure 15 shows the occurrences of each of the 88 notes. We can observe that most of the notes lie in the middle range of the piano which is usually expected in all music. Due to these characteristics, this dataset is well-balanced and suits our needs perfectly.

4.2. Tools used

The tools used to implement the proposed approach are:

a. *Keras on the Tensorflow backend*. Keras is a high-level library for machine learning with NN that allows us to prototype quickly and change parameters without rewriting a lot of code. It runs on Google's tensorflow which is a library for computations on graphs that suits NN perfectly and executes efficiently on the GPU.
b. *TensorBoard*. A utility from tensorflow to visualize learning.
c. *NumPy*. NumPy is a scientific linear algebra library that deals efficiently with matrices and vectors. This library is used in Keras and the training and testing data have to be NumPy arrays before feeding them to any network. Moreover, we use NumPy in the encoding and decoding part to deal with matrices (e.g. adding a vector to multiple rows in a matrix).

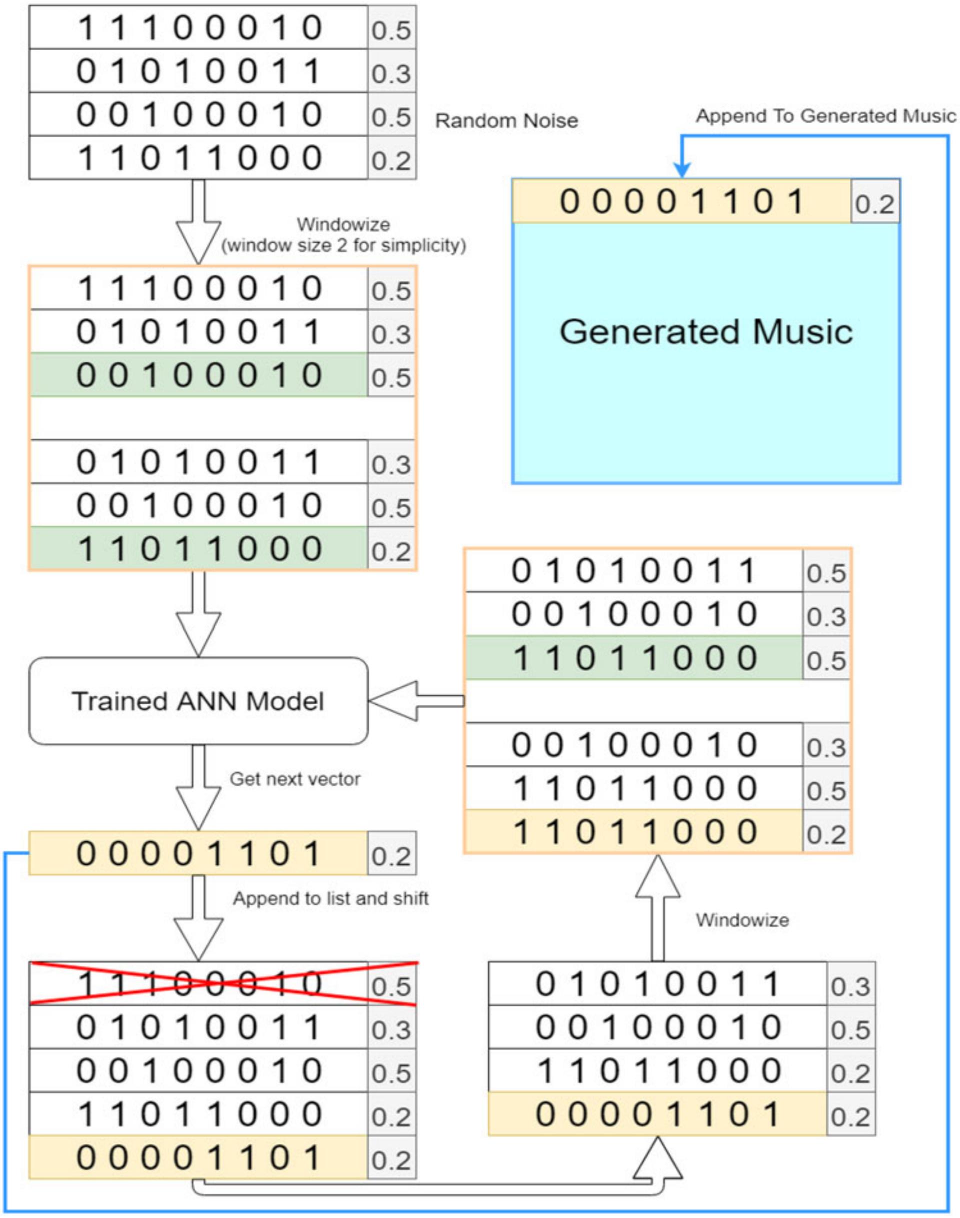


Figure 13. Music generating method.

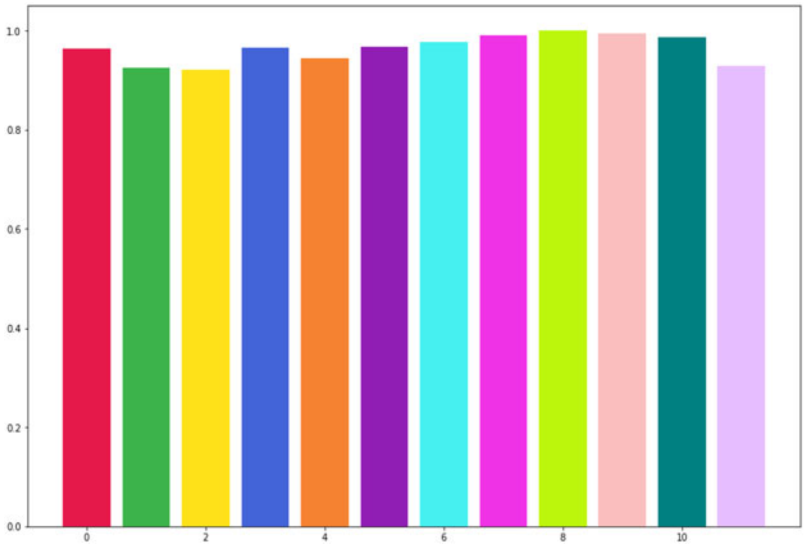


Figure 14. Dataset key count. *X*-axis: Key in numbers, *y*-axis: Normalized frequency.

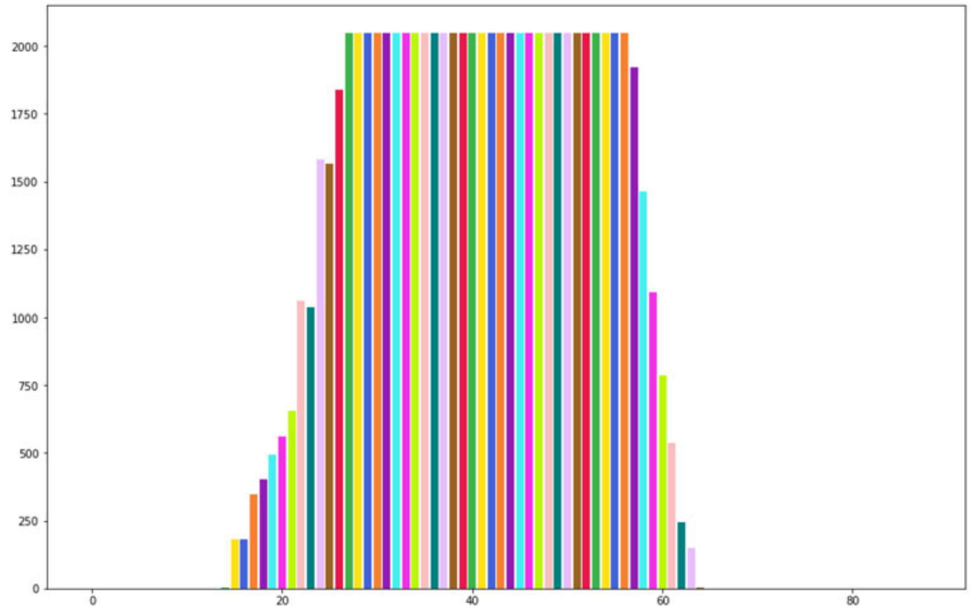


Figure 15. Note occurrences. *X*-axis: Piano key in numbers, *y*-axis: Un-normalized frequency.

d. *Mido*. Mido is a python library that helps reading MIDI files and writing them.

e. *Jupyter Notebook*. A web application for data visualization and statistical modelling. We use Jupyter Notebook in experimentation to be able to execute blocks of code in any sequence we choose. This helps to rapidly prototype experiments and NN models.

4.3. Neural network

NN has been created in Keras and train it on the data obtained from the windowing process. As shown in Figure 16, the NN consists of one or multiple LSTM layers each followed by a dropout layer that helps regularize the training and prevent overfitting. The dropout procedure is for dropping some nodes randomly. It is followed by a flattening operation that reduces the dimensionality of the matrix from 3D as output by the LSTM layers to 2D for the fully connected layers. Then we add fully connected layers each followed by a dropout layer. Finally, the output of the NN is a fully connected layer with size 89 with a sigmoid activation function applied to it. To guide the training of the NN, we opted to use the Binary Cross-entropy (Log-Loss) loss function instead of the classical mean squared error. This is because of the nature of our problem being close to a multi-label classification problem with 88 labels and since in multi-label classification, Binary Cross-Entropy gives a loss score more representative of the accuracy of the model (Kline & Berardi, 2005). Because of the 89th bit being a real value and not a binary value, this loss function is not perfect. However, it is close enough for our purpose.

To judge the performance of the network, we split the data into 70% training data and 30% validation data. We then use *F*-measure, which is the harmonic mean of precision and recall, to calculate the accuracy of the network.

4.4. Experimentations

Figures in the following subsections each includes four graphs as below:

- Epoch *F*-measure: The *F*-measure as calculated on the training data (*x*-axis) plotted against epoch count (*y*-axis).
- Epoch loss: The loss value as calculated on the training data (*x*-axis) plotted against epoch count (*y*-axis).
- Epoch Val *F*-measure: The *F*-measure as calculated on the testing data (*x*-axis) plotted against epoch count (*y*-axis).
- Epoch Val Loss: The loss value as calculated on the testing data (*x*-axis) plotted against epoch count (*y*-axis).

4.4.1. Number of LSTM layers

We have found out that having many LSTM layers made the learning progress slower and less accurate than having one or two layers with a sufficient number of neurons. This might be because of the 'Vanishing Gradient' problem where the depth of the network prevents the layers near the input to update their weights in an effective manner. We might have been able to get a deeper network if we found a way to combat this problem. One solution we didn't get to try was using 'Leaky ReLu' activation

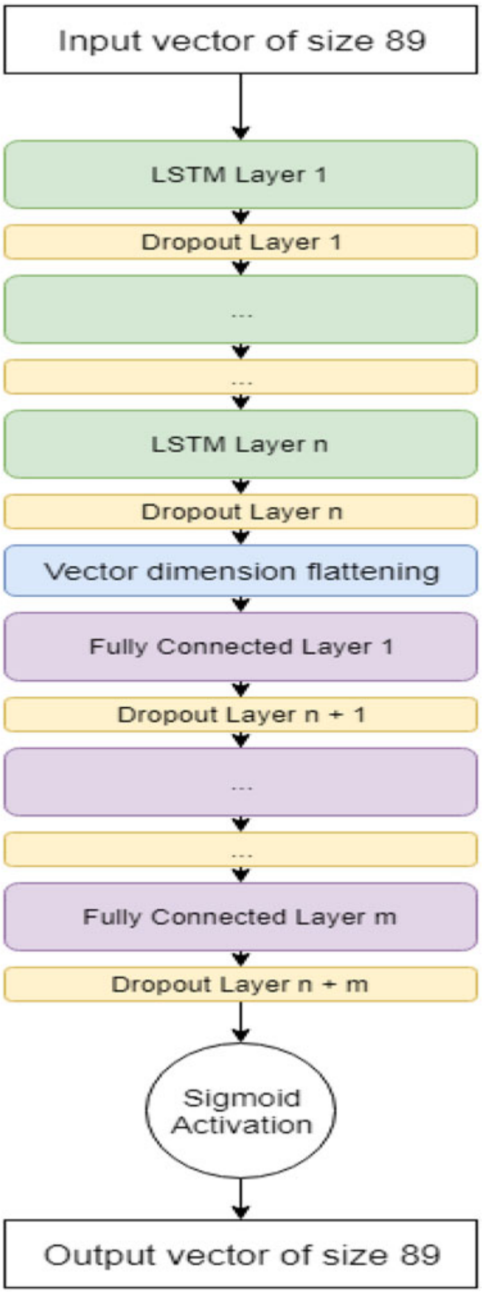


Figure 16. Architecture of the NN.

functions. We ended up using LSTM layers in the range of 2–4 layers.

4.4.2. Size of LSTM layers

The size of the LSTM layers had a big impact on the result. The bigger the size, the faster it converged, however this came with overfitting. We expect this was because having more neurons permitted the network to save more of the data of the training set into the weights instead of optimizing a way to generalize the overall patterns of music. This can be seen in Figure 17.

Figure 17 shows graphs for LSTM layer sizes 32 (light blue), 64 (pink), 128 (green), 256 (grey) and 512 (orange).

4.4.3. Dropout value

Dropout is a technique used in training NNs to regularize the training and prevent overfitting by randomly disconnecting a percentage of synapses between two layers. This forces the network to not use only specific paths and distributes the information throughout the various paths leading to better generalization with a smaller model. The dropout helped our network to not overfit. We thought it would eliminate overfitting entirely, however, this was not the case. We noticed from the graphs that it merely delayed overfitting. Although, this might have allowed the network some time to learn the patterns more effectively.

In Figure 18 dropout values are 0.0 (blue), 0.2 (red), 0.4 (light blue) and 0.8 (pink). It is clearly noticed that adding dropout will make the network converge slower in the training data. This is desirable because it reduces the memorization of the model and increases the generalization.

4.4.4. Optimizers

Keras optimizers are the engines that drive the backpropagation algorithm. Each optimizer has different characteristics (e.g. some optimizers include momentum while descending the gradient). Different optimizers that Keras offer have been used and results are shown in Figure 19. The optimizers used are Adam (red), RMSProp (light blue), Stochastic Gradient Descent (pink) and ADAGrad (green). The results show that RMSProp is the most suitable for our problem. Although they tend to overfit when the others don’t (the validation loss starts to increase), this is because they arrived earlier at the learning stage where the overfitting occurs.

4.4.5. Augmentations

As shown in Figure 20, we tested our augmentation technique with augmentation values 0 (green), 2 (grey), 4 (orange), 8 (blue), 11 (red). Each augmentation number corresponds to the number of transpositions applied to the dataset before training. The score for the training data shows that no augmentation produces the best accuracy. However, the testing accuracy shows that the more augmentations we have, the better validation accuracy we get. More importantly, the validation loss graph shows that the model with no augmentation hits a minimum loss, then starts to increase rapidly, while the models with higher augmentations resist this change. This sudden increase in the validation loss is the result of overfitting. So, while the network performs better in training for low augmentation, this does not generalize to data from outside the training set.

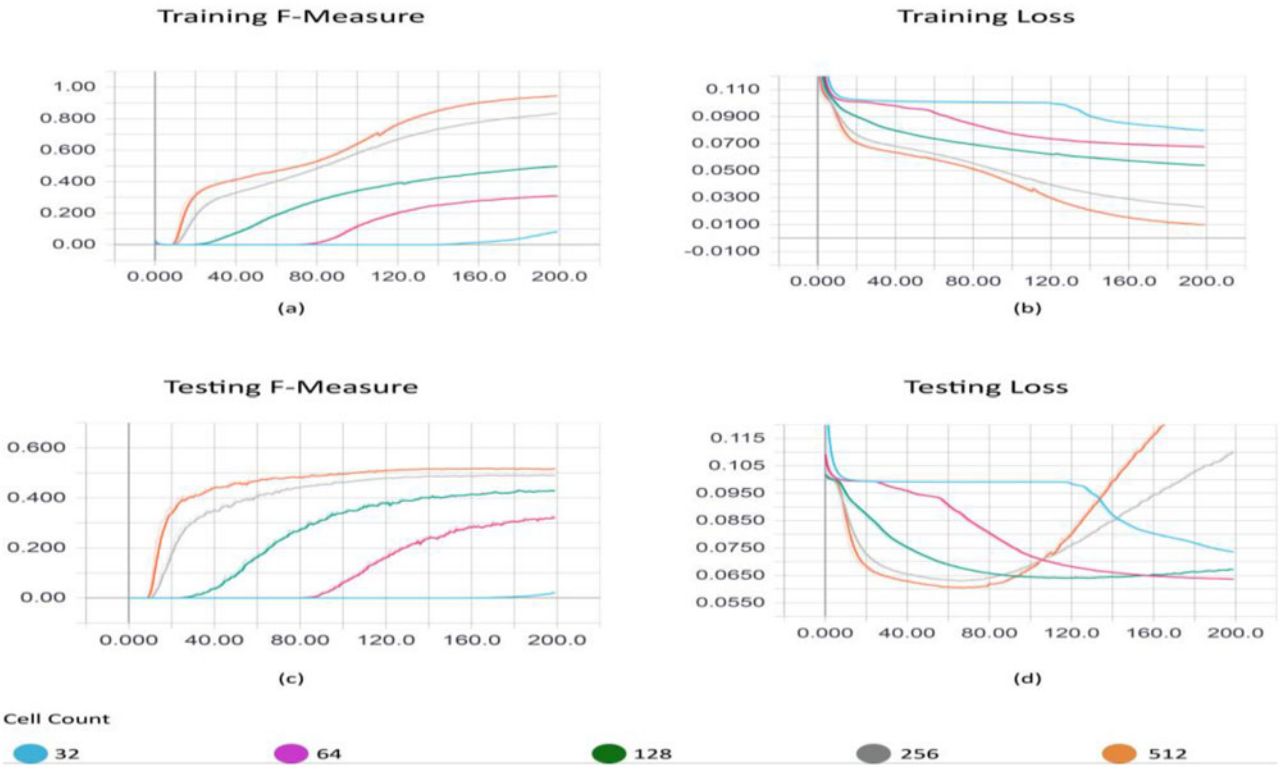


Figure 17. Testing LSTM layer sizes. (a) Training *F*-measure, (b) training loss, (c) testing *F*-measure, (d) testing loss.

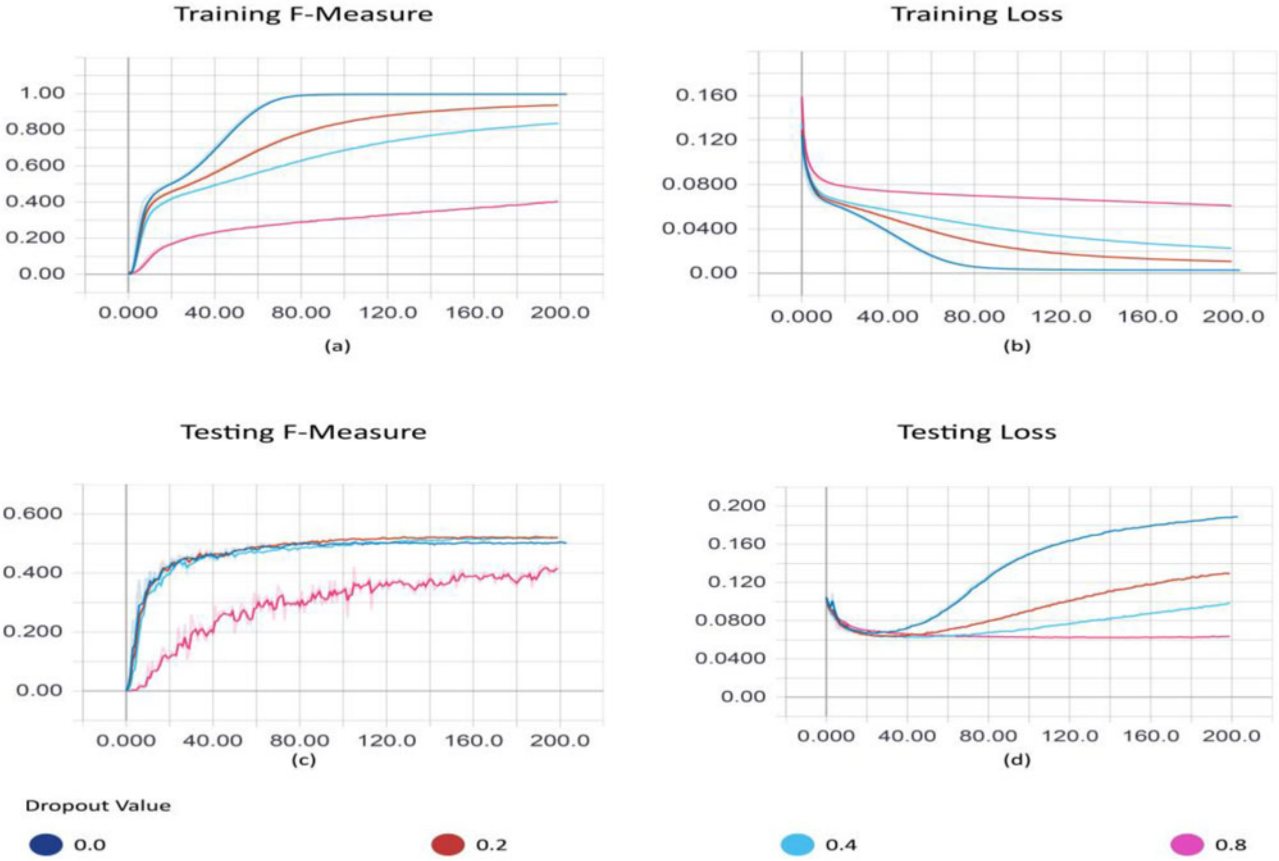


Figure 18. Testing dropout values. (a) Training *F*-measure, (b) training loss, (c) testing *F*-measure, (d) testing loss.

4.4.6. Batch size

The training data has to be split into batches that the network trains upon. After each batch, the network's LSTM state is reset, and the weights are updated. As shown in Figure 21, various values for the batch size have been tried and found that

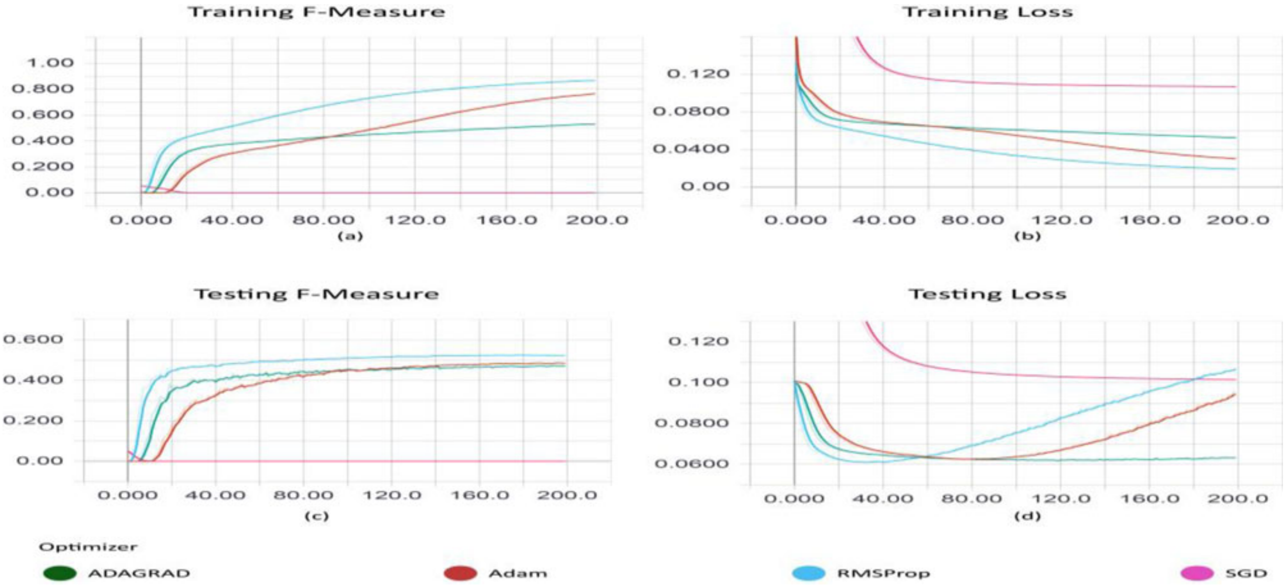


Figure 19. Testing different optimizers. (a) Training *F*-measure, (b) training loss, (c) testing *F*-Measure, (d) testing loss.

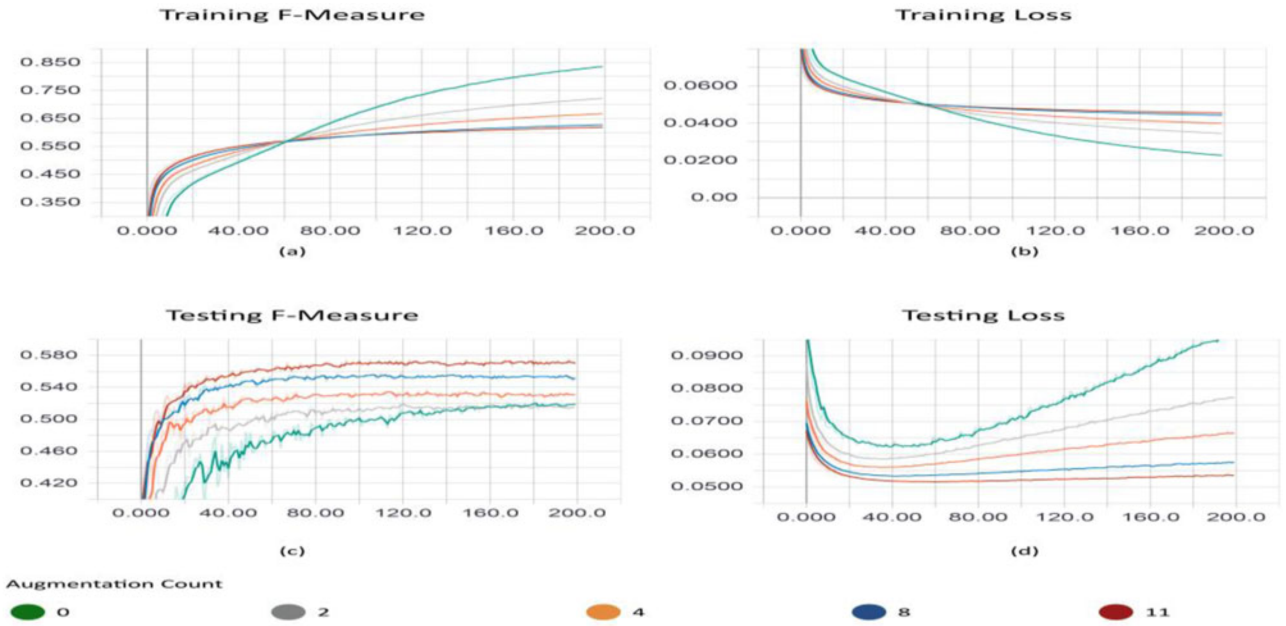


Figure 20. Testing augmentation values. (a) Training *F*-measure, (b) training loss, (c) testing *F*-measure, (d) testing Loss.

having smaller batch sizes makes the model perform better. However, having low batch sizes makes the training take a very long time. We found that a batch size of 512 gave the best result while maintaining relatively fast training.

4.4.7. Timesteps

The timesteps, or window size, had an impact on the performance of the model. We initially thought that having a larger window will definitely increase the accuracy, however the experimentation yielded results that suggests that there is an optimal size and having a larger window value will not contribute to better results. In Figure 22, window sizes 1 (light blue), 2 (pink), 4 (green), 8 (grey), 16 (orange), 32 (blue) are tested. Based on the *F*-measure score, window size 16 performed better than window size 32.

5. Discussion

It has been found that the most optimal configuration for the proposed model is to have 4 LSTM layers, with 512 LSTM cells each, a dropout value of 0.6 after each LSTM layer, 3 fully connected layers after the LSTM layers, with 256 neurons each, each of the fully connected layers followed by 0.6 dropout, RMSProp as optimizer, 512 batch size and a window size of 16. Using this configuration, we got a training *F*-measure of 0.83 and a testing *F*-measure of 0.71. The problem with generating music is the difficulty of measurement of the quality of the music.

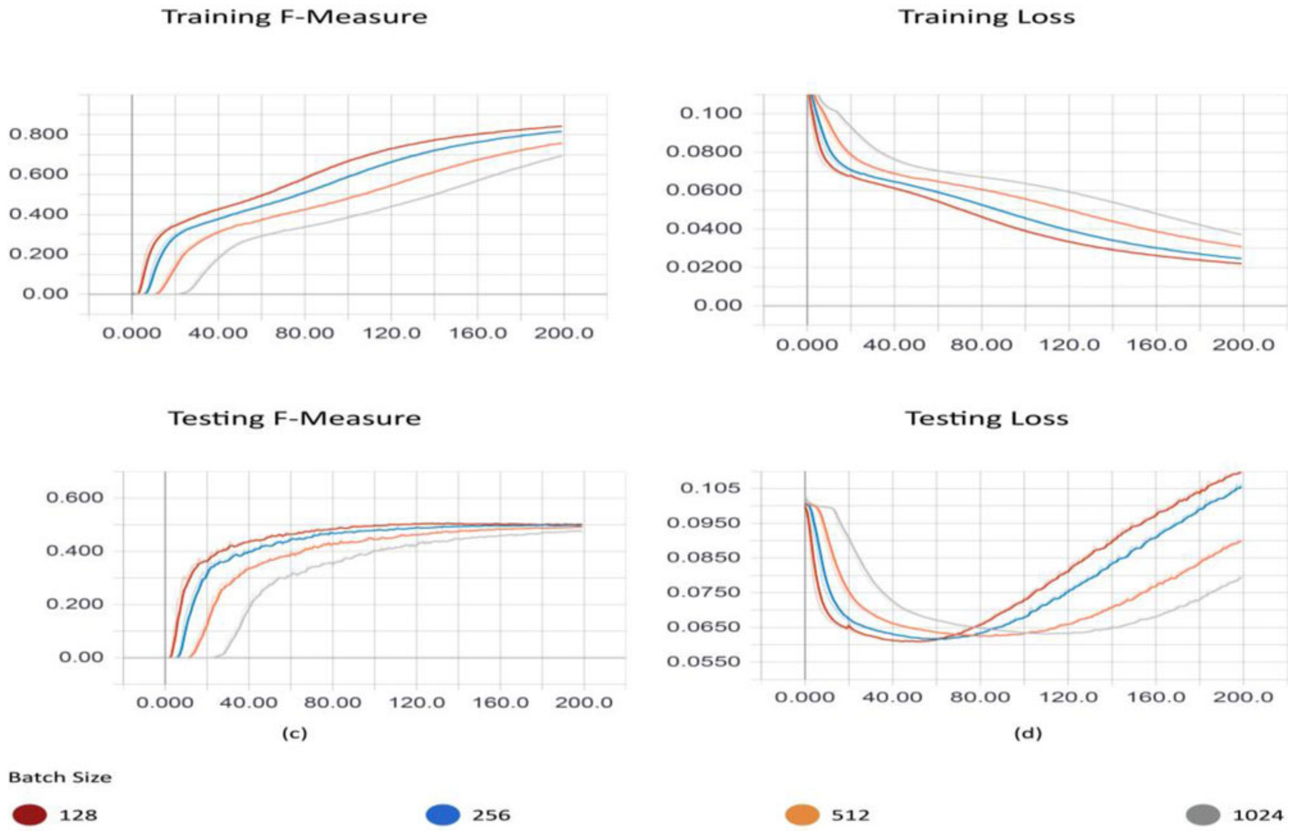


Figure 21. Testing batch sizes. (a) Training *F*-measure, (b) training Loss, (c) testing *F*-measure, (d) testing loss batch sizes 128 (red), 256 (blue), 512 (orange), and 1024 (grey).

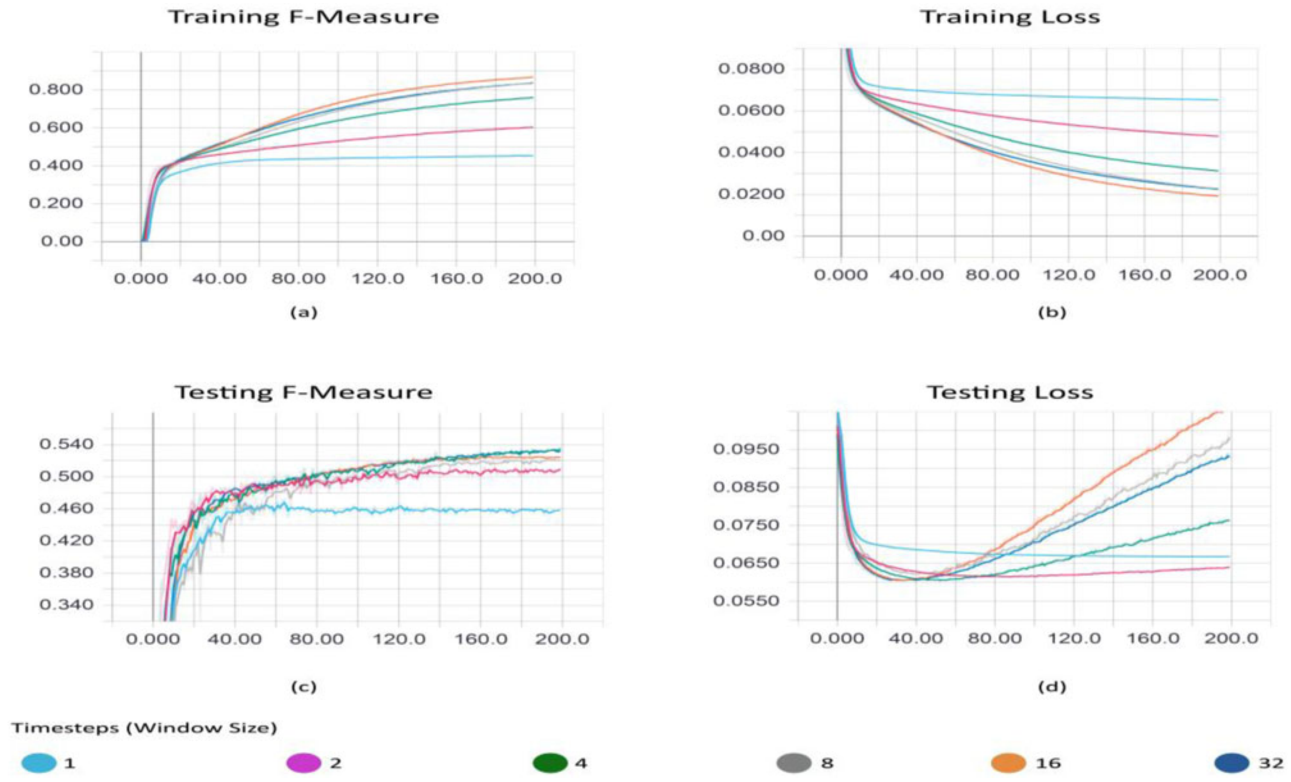


Figure 22. Testing window sizes. (a) Training *F*-measure, (b) training loss, (c) testing *F*-measure, (d) testing loss.

Our generated pieces had several advantages and disadvantages compared to the other music generators mentioned before. One advantage of our model was that the notes had varying durations/offsets compared to LSTM-based generators like Skuli (2017), which increased the complexity of the music.

Another is that it accounted for a much larger range of possible notes than Skuli (2017) or Chen (2001); Skuli had limited the range to the notes in his Final Fantasy dataset and Chen did not account for many chords. However, this increased complexity led to a lack of melodic structure. While there was a discernable melody in the generated pieces, there were a number of shortcomings compared to other generators. First, the generated pieces lacked the musical clarity (no cluttering of notes) generated by most other models like generated by Agarwala et al. (2017) or Skuli's. Second, the quality of the melody and rhythm within the pieces was low compared to Walder and Kim's (2018) model with its focus on recognizing motifs. Their generated pieces had far more rhythm and coherence because the model was capable of identifying motifs. Finally, while our generated pieces did have some note sequences which could be attributed or found to be related to the Bach dataset, they did not have the same level of fidelity to Bach's musical style like the BachBot model by Liang (2016).

6. Conclusion

In this study, we attempted to explore the behaviour of LSTM to generate music by developing an approach composed of six steps, namely, MIDI format conversion to song format, encoding, augmentation, windowing, learning using LSTM and music generation. Various experiments have been conducted with various parameters to investigate the improvement of learning. The results generated by our network were not very impressive from a music perspective, but not bad either. Our model shows a basic understanding of rhythm and harmony. The pieces generated do not have a large structure; beginning, ending and motif repetition. Some of this could have been addressed by a motif network similar to Walder's (2018) model. Some of the problems that remain unsolved are that the model in certain cases outputs a few good notes and then all zeros. It also sometimes produces short melodies which repeat infinitely. These two problems need to be deeply investigated and a post process might be enforced to avoid the occurrences of such rarely happening cases. The third problem was that the loss function does not take into account the 89th bit being a float value. If we omit the first two problems because they do not always occur, the model performance is musically acceptable and gives good harmony with less sound. For the third problem, the structure of the LSTM needs to give more importance to the 89th bit by somehow connecting the input directly to output so it can have a clear impact on the NN and in the loss function.

Possible future work for our work could be the use of custom loss functions that better model the sparse nature of the vectors. We could also use an extra 90th bit to represent the position inside the musical piece, which could help introduce larger structure in the pieces generated. Another possible improvement would be to use models other than LSTMs such as the Generative Adversarial Networks. We could also experiment with other types of musical datasets with different characteristics to the one we used. Finally, the problem could be extended to multi-instrument files and multi-instrument generation. Hopefully, these possible improvements could inspire further experimentation by others.

Disclosure statement

No potential conflict of interest was reported by the authors.

ORCID

Nabil Hewahi http://orcid.org/0000-0002-5870-2609
Sulaiman AlJanahi http://orcid.org/0000-0002-1781-4831

References

Agarwala, N., Inoue, Y., & Sly, A. (2017). Music composition using recurrent neural networks. [online]. Web.stanford.edu. Retrieved from https://web.stanford.edu/class/cs224n/reports/2762076.pdf.

Bello, J. (2018). MIDI Code. [ebook]. NYU. Retrieved from https://www.nyu.edu/classes/bello/FMT_files/9_MIDI_code.pdf.

Chen, C. (2001). *Creating melodies with evolving recurrent neural networks.* [ebook]. Washington, DC: IEEE. Retrieved from http://nn.cs.utexas.edu/downloads/papers/chen.ijcnn01.pdf.

Christies.com. (2018). Is artificial intelligence set to become art's next medium? | Christie's. [online]. Retrieved from https://www.christies.com/features/A-collaboration-between-two-artists-one-human-one-a-machine-9332-1.aspx.

Classicalarchives.com. (2018). Johann Sebastian Bach – Classical archives. [online]. Retrieved from https://www.classicalarchives.com/midi/composer/2113.html.

Clementstheory.com. (2018). Transposition. [image]. Retrieved from https://www.clementstheory.com/study/transposition/.

Donges, N. (2018). Recurrent neural networks and LSTM – Towards data science. [online]. Retrieved from https://towardsdatascience.com/recurrent-neural-networks-and-lstm-4b601dd822a5.

Eck, D., & Schmidhuber, J. (2002). *A first look at music composition using LSTM recurrent neural networks.* [ebook]. Manno, Switzerland: Instituto Dalle Molle di Studi Sull' Intelligenza Artificiale. Retrieved from http://people.idsia.ch/~juergen/blues/IDSIA-07-02.pdf.

Graves. (2014). Three-layer RNN. [image]. Retrieved from https://stackoverflow.com/questions/45223467/how-does-lstm-cell-map-to-layers.

Gurney, K. (2004). *An introduction to neural networks* (2nd ed., p. 13). [ebook]. London: UCL Press. Retrieved from https://www.inf.ed.ac.uk/teaching/courses/nlu/assets/reading/Gurney_et_al.pdf.

Hadjeres, G., Pachet, F., & Nielsen, F. (2017). *DeepBach: A steerable model for Bach Chorales Generation* (pp. 1–4.). [ebook]. Retrieved from https://pdfs.semanticscholar.org/d557/fdaa39c0047ac76618d018f1dad4e94aa508.pdf.

Kline, D. M. & Berardi, (2005). Revisiting squared-error and cross-entropy functions for training neural network classifiers. *Neural Computing & Application* [online], *14*, 310. Retrieved from https://link.springer.com/article/10.1007/s00521-005-0467-y.

Koch, H., & Dommer, A. (1865). *H. Ch. Koch's Musikalisches lexicon*. Heidelberg: J.C.B. Mohr.

Langston, P. (1988). *Six Techniques for Algorithmic Music Composition*. [ebook]. Morristown, New Jersey: Bellcore. Retrieved from http://peterlangston.com/Papers/amc.pdf.

Learningmusic.ableton.com. (2018). Keys and scales | Learning Music (Beta). [online]. Retrieved from https://learningmusic.ableton.com/notes-and-scales/keys-and-scales.html.

Liang, F. (2016). *BachBot: Automatic composition in the style of Bach chorales* (M. Phil in Machine Learning, Speech, and Language Technology). Cambridge: University of Cambridge.

McCormack, J. (1996). *Grammar based music composition*. [ebook]. Clayton, Victoria: Monash University. Retrieved from http://users.monash.edu/~jonmc/research/Papers/L-systemsMusic.pdf

Midi.org. (2018). About MIDI-Part 4: MIDI files. [online]. Retrieved from https://www.midi.org/articles-old/about-midi-part-4-midi-files

Richer, M. (2010). *Understand music theory: Teach yourself* (3rd ed.). London: Hachette UK.

Rouse, M., & Good, R. (2005). What is wavetable? – Definition from WhatIs.com. [online] WhatIs.com. Retrieved from https://whatis.techtarget.com/definition/wavetable.

Skuli, S. (2017). How to generate music using a LSTM neural network in Keras. [online]. Towards Data Science. Retrieved from https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-68786834d4c5.

Skymind. (2017). A beginner's guide to Generative Adversarial Networks (GANs). [online]. Retrieved form https://skymind.ai/wiki/generative-adversarial-network-gan.

Tchircoff, A. (2017). The mostly complete chart of neural networks, explained. [online]. Towards Data Science. Retrieved from https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464.

Tornblom, M. (2008). Piano keyboard. [image]. Retrieved form http://www.harmoniumnet.nl/klavier-keyboard-ENG.html.

Travers, J. (2018). Can AI write music well? | Technology. [online]. LabRoots. Retrieved from https://www.labroots.com/trending/technology/8810/listen-ai-writing-music-decide.

Walder, C. (2016). *Modelling symbolic music: Beyond the piano roll*. [ebook]. Cornell University. Retrieved form https://arxiv.org/abs/1606.01368.

Walder, C., & Kim, D. (2018). *Neural dynamic programming for musical self similarity*. [ebook]. Cornell University. Retrieved from https://arxiv.org/abs/1802.03144.

Wiest, L. (2017). Recurrent neural networks – Combination of RNN and CNN – Convolutional neural networks for image and video processing – TUM Wiki. [online]. Wiki.tum.de. Retrieved from https://wiki.tum.de/display/lfdv/Recurrent+Neural+Networks+-+Combination+of+RNN+and+CNN.

Zhang, G. (2000). Neural networks for classification: A survey. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, *30*(4), 451–462. [online]. Retrieved from https://ieeexplore.ieee.org/document/897072 doi:10.1109/5326.897072