Generating Music with LSTM Networks

Gokce Sarar
Department of Electrical and Computer Engineering
University of California, San Diego
San Diego, CA 92093
gsarar@ucsd.edu

Gitika Meher
Department of Electrical and Computer Engg.
University of California San Diego
9500 Gilman Dr, La Jolla, CA 92093
gkarumur@eng.ucsd.edu

Nasha Meoli
Department of Electrical and Computer Engg.
University of California San Diego
9500 Gilman Dr, La Jolla, CA 92093
nmeoli@eng.ucsd.edu

Ambareesh S J
Department of Electrical and Computer Engg.
University of California San Diego
9500 Gilman Dr, La Jolla, CA 92093
asreekum@ucsd.edu

Farheen Ahluwalia
Department of Computer Science Engg.
University of California San Diego
9500 Gilman Dr, La Jolla, CA 92093
fahluwal@ucsd.edu

Abstract

Recurrent Neural Networks can be trained to produce sequences, as exemplified by recent results in machine learning research, particularly in machine translation and sequence modeling. While vanilla RNNs work in these settings, they are posed with certain problems like vanishing gradients that make learning of deep models difficult. LSTM (Long Short Term Memory Networks) are a variant of RNNs that overcome these issues[7]. Their architecture, enable easy propagation of gradient while retaining memory sequences for predictions. In this paper, we explore LSTMs and how they can be used for music generation. We conduct a variety of experiments with different architectures, and sampling strategies to better understand the behavior of LSTMs in these settings. We also experiment and compare LSTM performance with performance of vanilla RNNs. Finally, we provide music samples generated by our LSTM models.

1 Introduction

Using Long Short Term Memory networks it is possible to generate new sequential data by giving it a considerable of amount of training data, and in this project the effectiveness of LSTM is explored with music sequences. The LSTM network is trained using characters from a music database, where the music notes are given in ABC format.

Recurrent neural networks make use of sequential information. They have a "memory" which is used to capture information about the previous states. Given an input sequence, the recurrent neural network applies a recurrence formula at every time step based on the previous state of the current network and current input to produce an output and the new state. The same parameter and same recurrence formula is used at every time step.

Long Short Term Memory (LSTM) units are a variation of RNN which use cell states and gates to capture long-term dependencies in data. A typical LSTM unit consists of 4 gates which control the

extent of previous information that needs to be forgotten , new information that must be stored in the cell state, the output to be produced and the update to hidden state. The cell state provides a mechanism for gradient to be backpropagated uninterrupted. Hence, LSTMs are able to resolve the vanishing gradient problem and learn long-term dependencies in data.

In this paper, we explore LSTMs and how they can be used for music generation. We conduct a variety of experiments with different architectures, and sampling strategies to better understand the behavior of LSTMs in these settings.

2 Methods

2.1 Vanilla RNN

Traditional neural network are unable to use reasoning about previous events in sequential data to predict or generate later information. RNNs are a way of solving this problem.

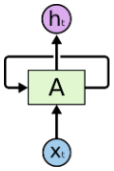

Figure 1: Basic RNN unit

A recurrent neural network intuitively contain multiple copies of the same network, each passing a message to a successor. A representation of the same on unrolling the loop:

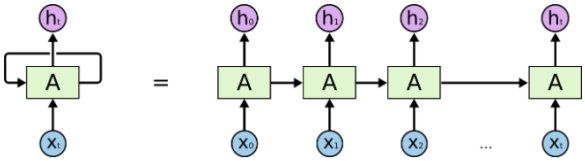

Figure 2: Unrolled RNN Unit

Unlike normal Neural Networks (or Convolutional Networks) RNN's are not too constrained: they accept different sized input. Even though RNNs can theoretically connect and predict a sequence of any lengths, sometimes when the gap between the relevant information and the place that it's needed is large RNNs can't seem to learn to use the past information, very well. We used a Vanilla RNN model to generate music data, and the parameters were the same as that of the later LSTM models, to compare and contrast the performance of the model.

2.2 LSTM

Long Short Term Memory networks are first introduced in [4] and they are a special kind of RNN, capable of learning long-term dependencies. All RNNs have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single $tanh$ layer.The LSTMs have the ability to remove or add information to the cell state, carefully regulated by structures called gates. The first step in LSTM is to decide what information to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer." It looks at previous hidden output and current input, and outputs a number between 0 and 1 for each number in the cell state.

The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next,

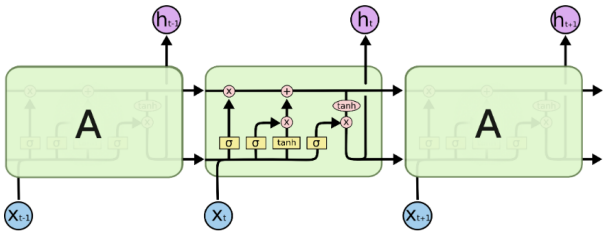

Figure 3: Basic LSTM unit

a tanh layer creates a vector of new candidate values, that could be added to the state. In the next step, we these two are combined to create an update to the state. Then the old state is multiplied by ft, forgetting the things we decided to forget earlier. We add a new candidate value, scaled by how much we decided to update each state value. Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a point-wise multiplication operation.

LSTMs use this architecture to retain previous information and also incorporate new information as well. To tackle vanishing gradients the information is fed in chunks and the hidden state is carried forward to the next chunk. There have been last of proposed variants of the architecture as can be seen in [3]. LSTMs are very effective in problems that involve remembering context and predicting sequences.

2.3 Sampling

During the generation stage, we “prime” the network with a sequence, and then let the network run on its own, predicting the next character, and then using the network’s output as the next input. There are two ways of producing the network output, the first being, taking the maximum probability element from the softmax output which takes the maximum output. However this leads to the same output multiple times, making the result deterministic. The second way is sampling, where you sample the output from the softmax probabilities using a Gaussian probability distribution, which generally results in a more "musical" output[6].

2.4 Baseline Model

2.4.1 Description

We used PyTorch’s builtin LSTM module for creating our model. The model has two blocks. The input is fed into the LSTM layer, which has one hidden layer with 100 cells. The input dimension of LSTM layer is 94. After the LSTM layer, there is one Linear Layer, which maps the 100 hidden state output to 94 dictionary classes. After that we give the logits to Crossentropy loss, whose implementation in Pytorch includes the softmax implementation.

2.4.2 Training Procedure

We trained the LSTM neural network for 100 epochs and chose the best model for evaluation of the test set. We used an initial learning rate of 0.001 and a batch size of 1. We used Adam optimizer. We split the datasets in to chunks of 100 characters. There were 93 characters in the dictionary and we also added a new character, $, to pad the last chunk in each epoch, that’s why our dictionary size is 94. Following that our input has the one-hot encoding for 94 characters and the softmax output is also 94 dimensional.

In Fig. 4, the validation and training losses during training can be seen. We evaluated after each epoch. As can be seen training loss got really lower than validation loss, where the validation loss began over-fitting after 25th epoch, which is an early stage. The best model had a validation loss of 1.778, and when we evaluated it on the test set, the test set loss was 1.990.

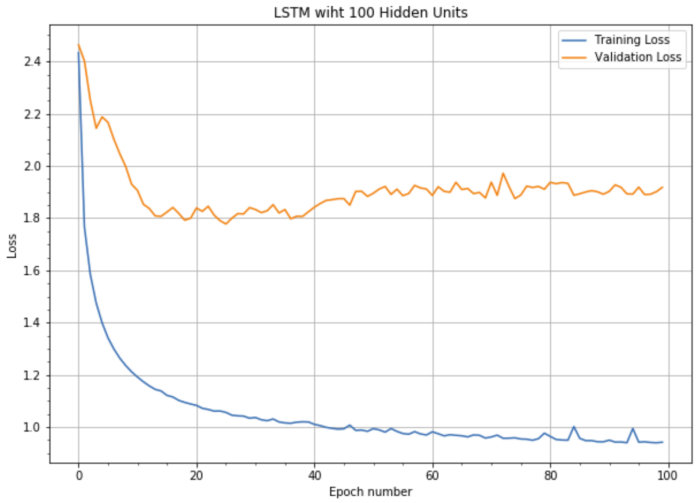

Figure 4: Training and Validation Set Losses during training

3 Results

3.1 Generated Pieces with different Temperatures

In Fig. 5, the prime we fed to the network can be seen. We chose a prime of length 100, so that the hidden states would reach to a good condition for generation. We chose 100 characters for the prime, since we trained LSTM with a memory of 100, that's why we thought that after seeing 100 proper characters, model would be in a state of generating good-quality pieces. We let the model to generate 5000 characters and chose the good ones for reporting purposes. We applied sampling from softmax with 3 different temperatures: 1, 0.5 and 2.

```
<start>
X:19
T:Dusty Miller, The
R:hop jig
D:Tommy Keane: The Piper's Apron
Z:id:hn-slipjig-19
M:9/8
```

Figure 5: Prime fed to the network

3.1.1 Temperature = 1

With a softmax sampling with a temperature of 1, we managed to generate really good pieces. We had both long and short pieces, but we didn't encounter an example, which is not convertible (it might have occurred if we generated more, but among the few times we generated 5000 characters, we didn't encounter not convertible ones). The generated samples are different, when we generated 5000 characters couple of times due to random sampling from softmax. In Fig. 6, the first piece we generated can be seen in ABC format and in Fig. 7, its converted version can be seen. In Fig. 8, the ABC format of our second generated piece can be seen, which was the longest piece we generated. It's converted version can be seen in Fig. 9.

3.1.2 Temperature = 0.5

Changing the temperature, allows us the changing the probability distribution towards to a more uniformly distribution or to an impulse-like distribution[1]. By decreasing temperature we are moving towards to an impulse-like distribution, where the generated music gets less random and

```
<start>
X:88
T:Derrof Cackiam Donke's And
R:polka
D:#arit Martin Keels: Trane
Z:id:hn-polka-77
M:2/4
L:1/8
K:A
A>B cB|A2 A2:|
|:Bd B/c/d/e/|dB cB|Af f/f/a/f/|ae ed | cB e2|A>B GB|
cB AE/A/|BA cB|AD FE|D>A BA|BA Be|de fa|ag fe|d2 d2||
<end>
```

Figure 6: Generated music1 with T=1, in abc notation

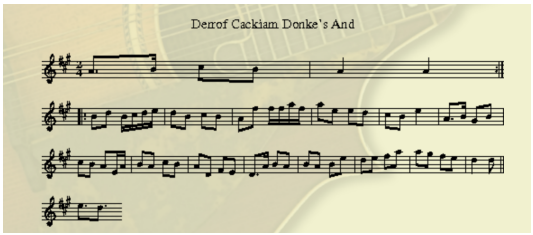

Figure 7: Generated music1 with T=1, in converted version

```
<start>
X:47
T:Prion Doulkad Coottlendy's
R:polka
K:G
DD DD | Ad fg | BB BA | fd ed/e/ | fe dB/A/ |1 G3 :|2
A^c cA | d^c ed | ed (3ege | d2 B2 |
ef af | fa fe | fg fe | f3 gd ||
P:variations
|: a2 | afdf gedB | defd c2BA | BABd e2ce |1 fdfd cege ||
|: dedB gedB | AGFE DBdf |
gddB Agfe | dGFE gC |
EG B/d/e/G/ cB/c/ | cc/B/ A>B | A2 ef/c/ | fe g>a | e>f e>e | fe c2 |
B3 A/B/ | AG EF | B/c/d B2 | ec AG | B2 AG | B2 cB/A/ |
BA A>B/A/ | GF Bd |1 A2 A>B :|2 A2 AG/A/ |
d/f/e/d/ B/d/e |1 dA AG ||
P:Variations varie ::
DF (3ddd | =c2df | f2ef | fa (3g ef |
gedB D2DE |
eAAB GAGB | A2AA BAGF |1 GABc de dc :|
<end>
```

Figure 8: Generated music2 with T=1, in abc notation

when temperature is 0, then we have argmax for choosing the next sample, where everything is completely deterministic. When we decreased temperature to 0.5, it didn't degrade the performance with respect to T=1, since it still provides the necessary randomness. We were able to generate plausible pieces and couldn't detect much of a difference from T=1. In Fig. 10, the first piece we generated can be seen, whereas its converted version can be seen in Fig. 11. Similarly, in Fig. 12, the second piece we generated can be seen, where it's converted version is in Fig. 13.

3.1.3 Temperature = 2

When temperature is set to 2, the generated pieces become meaningless. Most of the time, the first piece after prime begins with $< start >$ and goes on for a very long time, since for most of the samples the probabilities are similaryly and highly likely due to the fact that the probability distribution became more uniform. So the pieces don't necessarily follow the logical order, which the model is trained on. The most obvious observation is with $T = 1$ and $T = 0.5$, we observed that almost always $< start >$s followed $< end >$s (we never encountered otherwise, but in case we missed one or two examples, we state as "almost aslways"). Nevertheless. with T=2, there is for example the following generated sequences: $\backslash n < end > \backslash n < saart, < end > \backslash n < spart, n < end > \backslash nM : 4/$. That's why we needed to run multiple times to get a meaningful results. Most of the time ABC converter couldn't convert them. That's why even if we ran a large amount of time, we just encountered one proper piece and a second one which ended with $< enc >$ instead of $< end >$. However, this second one could be converted by abcConverter if we just changed the end of the piece. That's why here we are giving three examples: First, in Fig.14, you can see a piece which hast start and end but couldn't be converted by abcConverter. Secondly, you can see the example in Fig. 15, where we changed $< enc >$ to $< end >$, but didn't touch the rest and in

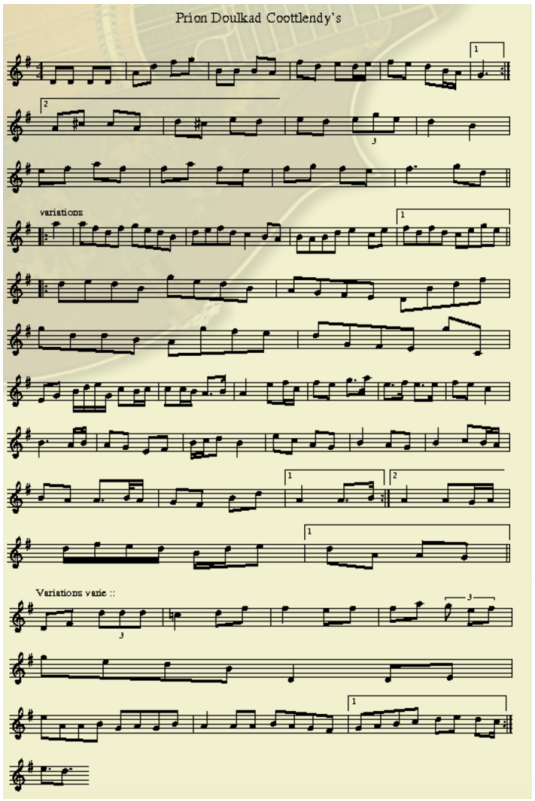

Figure 9: Generated music1 with T=1, in converted version

```
<start>
X:88
T:Cowas Mazurka, The
T:Aun Berceth Belly, The
R:polka
H:Also played in Amill to Magains du parled with the Bally at the Dann Learche Kenfed aid rassion the Daverite/d/ly:
16
M:2/4
L:1/8
K:D
d2 BA|d2 d2|d2 d2|c2 AF|AB AB|AB AB|AB AF|1 G2 Bd:|2 G2 G2||
|:e2 ge|fd Bc/d/|ed BA|d2 d>B|AF DE|DE DE||
<end>
```

Figure 10: Generated music1 with T=0.5, in ABC format

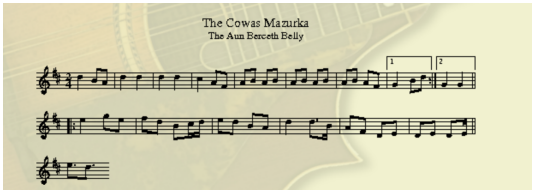

Figure 11: Generated music1 with T=0.5, in converted version

```
<start>
X:7
T:Wardon's Polka
R:polka
H:Also played in A millina be the part in the moulsish in cret the camat ard #54
D:Micheas: The Froet Irlin: uree and Crolkale
S:idihn-polka-77
M:2/4
L:1/8
K:D
A>B AF|AB cA|B>A BA|G>A GB|cB AB/A/|GF ED/E/|FA AB|cB AB|cB A2|1 G2 G2:|2 G2 GA||
P:Version 2:
AB cB/A/|1 AG A>B||cA AB/c/|de dB|dB BA|BA AB|ce ed|e/f/g fe|de fa|fa fe|fe d2|ef/e/ dB|Bc d2|ef/e/ dB|A/B/c/d/ eg/e/
|df ed|e2 ge|ce e>e|1 ed B2:|2 BG GB||
|:fe fe|d2 Bc|d2 d2|d2 d2|ed ed|ed ef|g>a ga|ba gf|ed BA|FA BA|1 BA AB:|2 AG G2||
|:ec ed|ef ed|e2 ed/c/|BA AB/c/|d2 d2:|
|:ed cB|A/F/E/F/ GE|AD D2:|
<end>
```

Figure 12: Generated music2 with T=0.5, in ABC format

Fig.16, you can see its converted version. Finaly, in Fig. 17, you can see a good generated piece in abc notation and in Fig. 18 you can see its converted version. We don't give example of long, non-ending sequences here, since for them it was obvious that the generation failed.

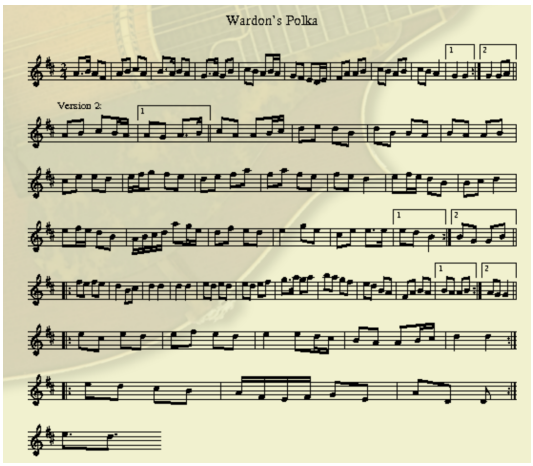

Figure 13: Generated music2 with T=0.5, in converted version

```
<start>
S3MrMion- #55)
I:aic: Wfituc3>c NeobhD/D^ca."Gk 4G6:Gd e2ece | {
CFA^fw'a'7'] REF]06Jz f2ie |
Pevd (serle "A"s_st.e2
Z:-ls-vart Mi-)gh/B/@/2&.BrovEn.Ge,d
Z:Transcrit dfaie feLvc/rgiomby.
M:",3 lackadeat>soLvor'swhal.WB-007-93
z 4 A z FEC: DEAE>D :|
<end>
```

Figure 14: Generated music1 with T=2, which couldn't be converted by abcConverter

```
<start>
X:879
Af)D^AD.E C:ZFAB .D? g4|
B2B ata |g (b/af|g/e/d a|
DML.a/y nave, h?n oast, Wi)fu-be
T:?llt <xtarap
K:G
Bc:|BdBd eceA|~G12 GBAG||ABf dFD |1 GEcd A2:|
|]fg|!AAc-fuf|A2c BA^A|^cGB EcA BB|FEGc A2fg|
<enc>
```

Figure 15: Generated music2 with T=2, in abc notation

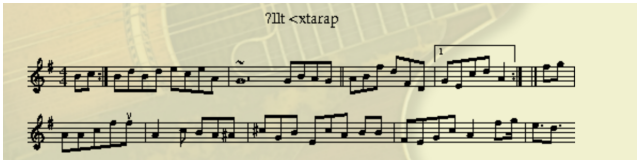

Figure 16: Generated music2 with T=2, converted version

3.2 Changing number of neurons

The number of hidden neurons in an LSTM is an important hyperparameter in the design of the model[2], three different numbers of hidden units were used 50,75 and 150.
The curves for the training loss shown in Figures 19 to 21 show similar patters for different numbers of hidden units.

```
<start>
X:164
Z:ijsfdiyn
adfa edfamed A>dbro::al Ton, veglanaa:
H:imioc. Ipel..a.ne,t cordcNon.
W:Trz
StZPierral Afey~] Chegla
Z:ThorymFr toty "Fom
K:Bman
A|Bc cA|cB e/g/f/e/|12D::AG ||
<end>
```

Figure 17: Generated music3 with T=2, in abc notation

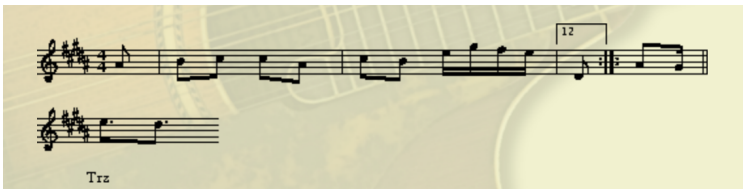

Figure 18: Generated music3 with T=2, converted version

A test accuracy of 62.5%, 66% and 72.6% were obtained for the 50, 75 and 150 neurons respectively.

The validation loss curves show that the number of neurons affects the shape of the validation loss curve. As the number of hidden units increase then the minimum validation loss is reached in fewer epochs. In Figure 21 the minimum validation loss is reached in under 10 epochs whereas in Figure 19 it takes over 20 epochs to converge.

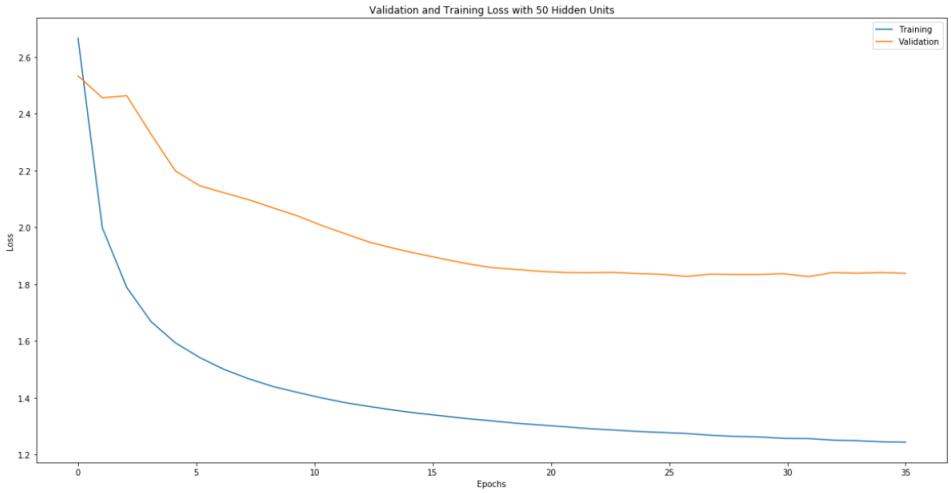

Figure 19: Loss with 50 neurons in hidden layer

3.3 Argmax Sampling

3.3.1 With Argmax Sampling

When Sampling is done choosing the largest index of the softmax output, all the tunes generated are similar and there is not a lot of variance in a single tune either. We observe a lot of repeating characters. The tune titles, composers across different different tunes remain the same too. Sizes of

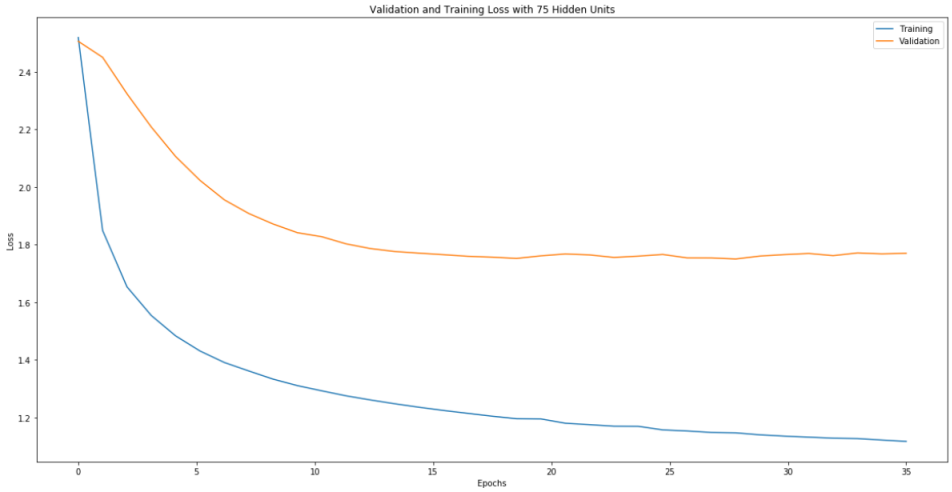

Figure 20: Loss with 75 neurons in hidden layer

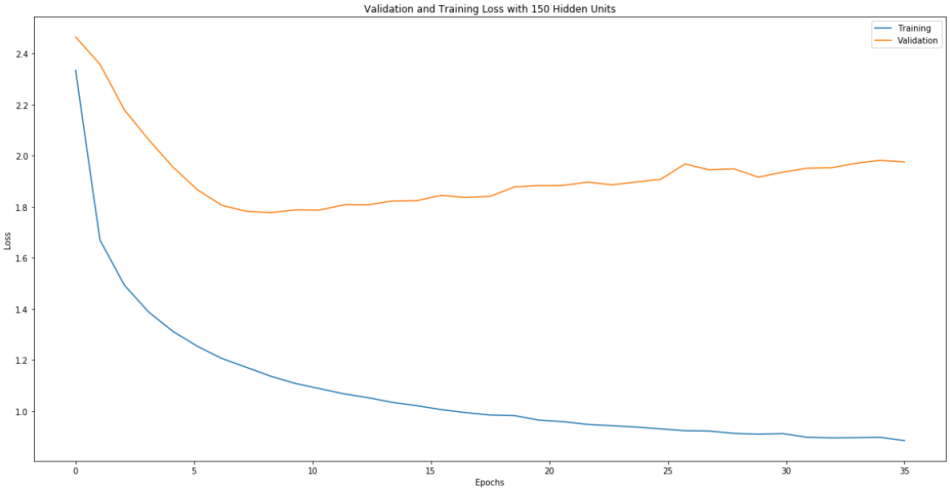

Figure 21: Loss with 150 neurons in hidden layer

different generated tunes also look similar. Sampling using the argmax function is very deterministic.

<start>
X:79
T:John Maidh Mazurka, The
T:Wallow the Bally to Thein Boom
R:polka
Z:id:hn-polka-77
M:2/4
L:1/8
K:D
A>B AF|AF AF|AB AF|AB AB|AB AB|cB AB|AB AB|cB AB|AB AB|cB AB|AB AB|cB AB|AB AB|AB AB|AB AB|AB
AB:|
<end>

Figure 22: Figure showing the midi tune generated using argmax sampling in abc format

Figure 23: Figure showing the midi tune generated using argmax sampling

```
<start>
X:79
T:John Maidh Mazurka, The
T:Wallow the Bally to Thein Boom
R:polka
Z:id:hn-polka-77
M:2/4
L:1/8
K:D
A>B AF|AF AF|AB AF|AB AB|AB AB|cB AB|AB AB|cB AB|AB AB|cB AB|AB AB|cB AB|AB AB|AB AB|AB AB|AB
AB:|
<end>
```

Figure 24: Figure showing the midi tune generated using argmax sampling in abc format

Figure 25: Figure showing the midi tune generated using argmax sampling

3.3.2 Sampling with T=0.7

When sampling is done using the temperature based softmax sampling method, the obtained tunes are less deterministic and vary among themselves still capturing the underlying features. As shown in the figures, the author names, regional origins, length of the tunes are considerable different for all the generated tunes with the same primer characters.

```
<start>
X:46
T:Mist Conke, The
T:Filly as Chavere #1
T:Flick's Cornas The
R:polka
Z:id:hn-polka-77
M:2/4
L:1/8
K:A
A>B AF|(3B,D GE D2||
<end>
```

Figure 26: Figure showing the midi tune generated using temperature based softmax sampling in abc format

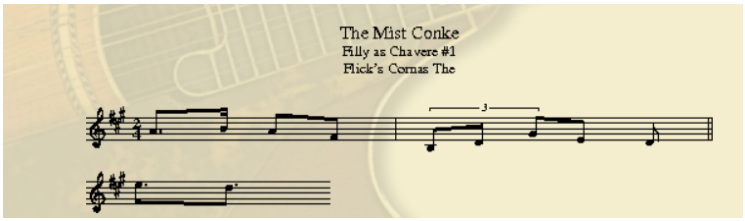

Figure 27: Figure showing the midi tune generated using temperature based softmax sampling

```
<start>
X:14
T:Crigal Polka
R:polka
Z:id:hn-polka-72
M:2/4
L:1/8
K:D
A>B FE|DF Ac|AB/A/ GE|DD FA|DF FA|AF EF|GB/A/ GF|1 DD DF:|2 D2 AF||
|:fe fe|dB AB|cB AB/c/|dB BA|dd d2|de/f/ ge|fd cB|AF DF|1 AG Gz:|2 AG G2||
<end>
```

Figure 28: Figure showing the midi tune generated using temperature based softmax sampling in abc format

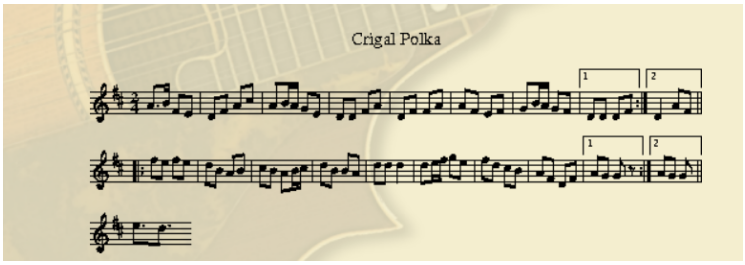

Figure 29: Figure showing the midi tune generated using temperature based softmax sampling

3.4 Vanilla RNN

RNNs, expectantly produced less effective results compared to LSTMs. The training, and validation loss curves were higher, and so was the Test Loss.

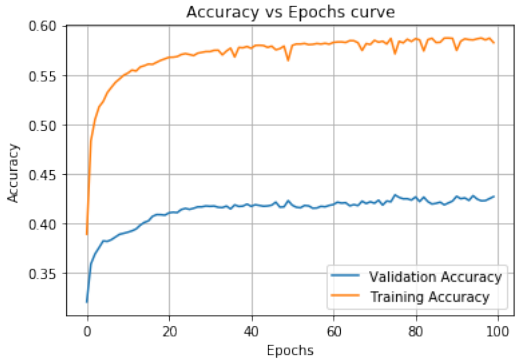

Figure 30: Accuracy vs Epochs for RNN

The Test results from the best model after 100 epochs with 0.001 learning rate and 100 hidden units, was the following:

Test Loss : 1.9697

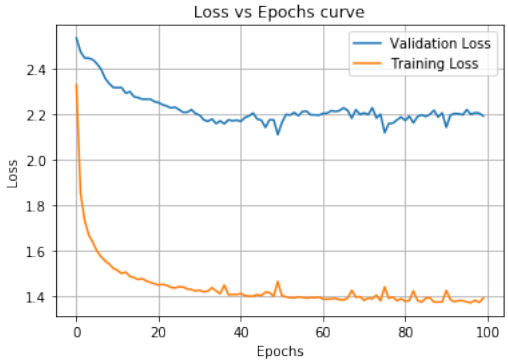

Figure 31: Loss vs Epochs for RNN

The loss was higher as compared to the LSTM and also the music generated was of poorer quality in two aspects: (1) The tunes generated were very short in general, and there were only few of considerable length (2) The tunes were at times jarring, with sudden spiky notes in between, making it sound amateurish, even to the untrained listener. The notes produced by LSTM were much more regular and sounded better. the resuls prove that LSTM are much better in preserving long term dependenices and solving vanishing gradient problems. Compared to LSTMs, Vanilla RNNs suffer, making them less effective of sequential data generation.

Tunes generated were the following:

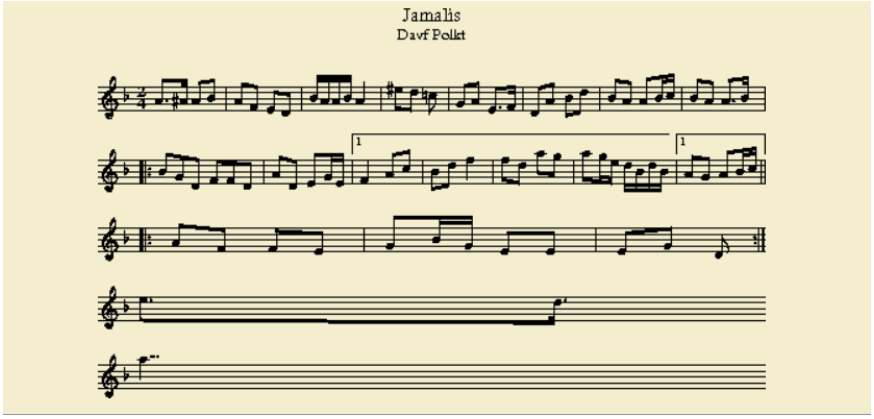

Figure 32: Generated Music RNN

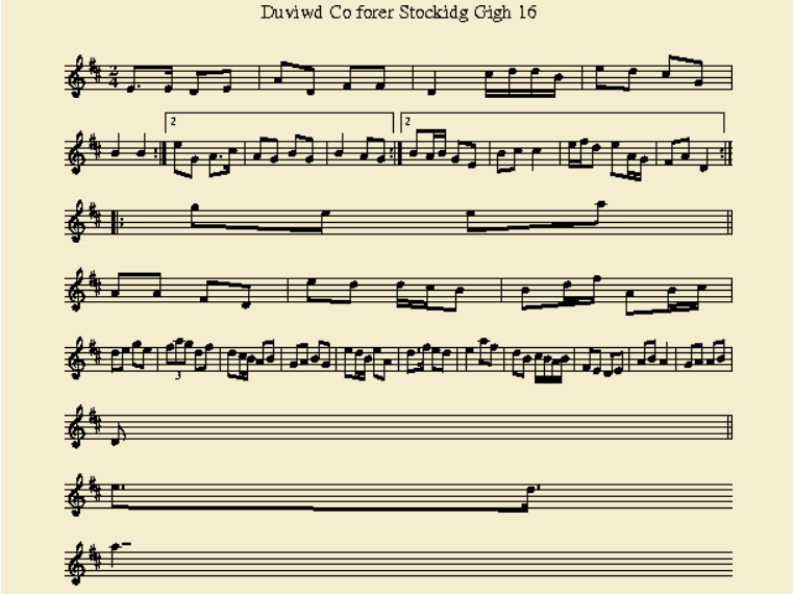

Figure 33: Generated Music RNN

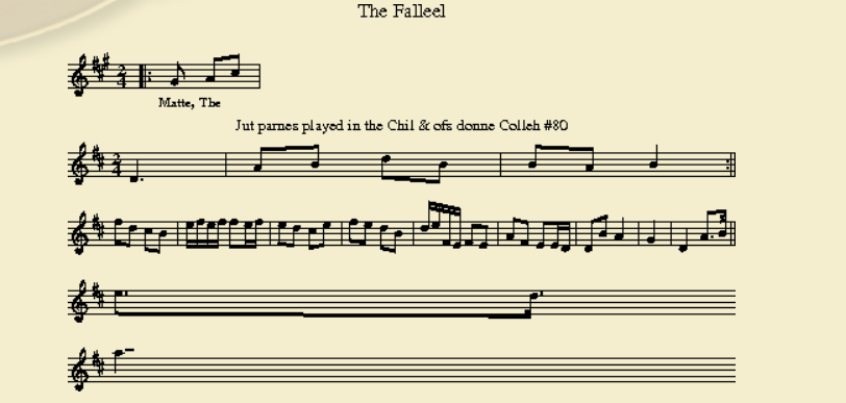

Figure 34: Generated Music RNN

3.5 Feature Evaluation

An interesting way to look at how the LSTM is able to predict the music sequence is to look at each of the hidden unit activations during a forward propagation and a good example can be seen in [5]. The color of the hidden unit corresponds to its activation. Blue corresponds to the least active unit and red corresponds to an most active unit. Most of these hidden units follow some pattern in their activation which helps in producing music. The heatmap of the the neuron in Figure 35 exhibits high activations for the tune body and low activations for the header and other tune elements. The heatmap in Figure 36 shows high activations for the start delimiter and the reference keys in the tune. In Figure 37 The heatmap for the particular neuron chosen suggests that it is detecting just the title of the tune.

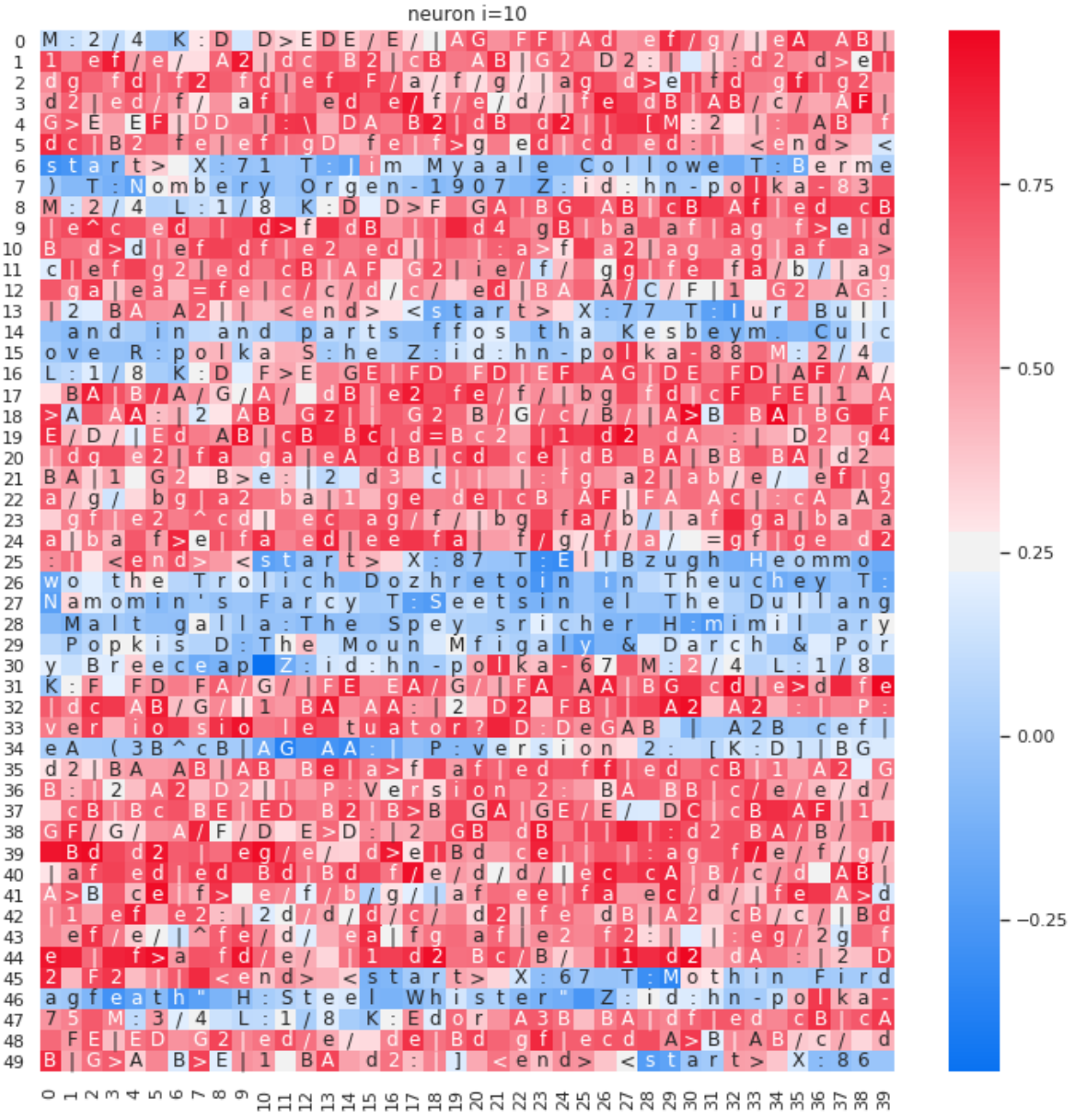

Figure 35: Heatmap for a neuron activating only for tune body

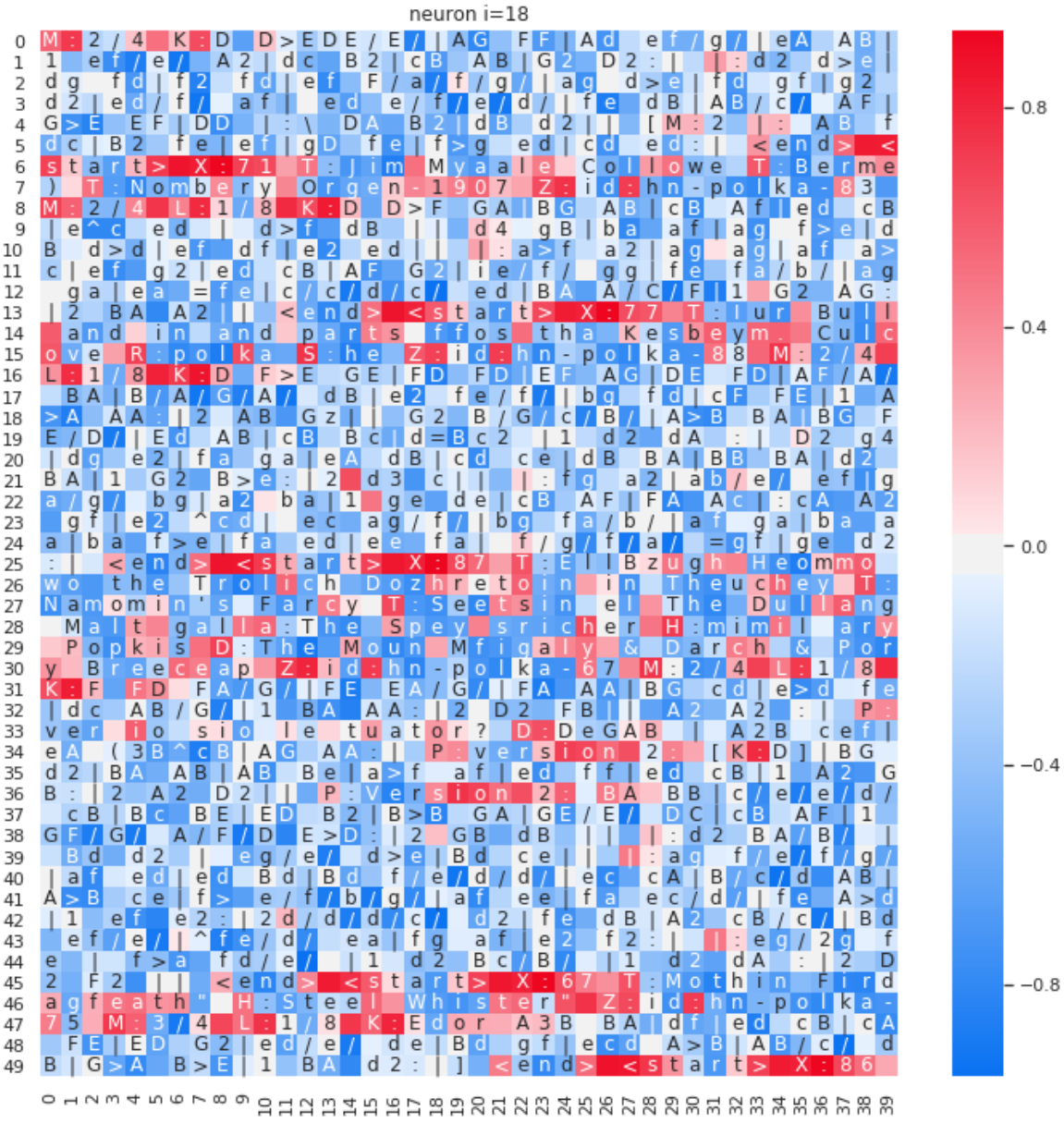

Figure 36: Heatmap for a neuron maximally activating for the the beginning of start delimiter and the reference number of the tune

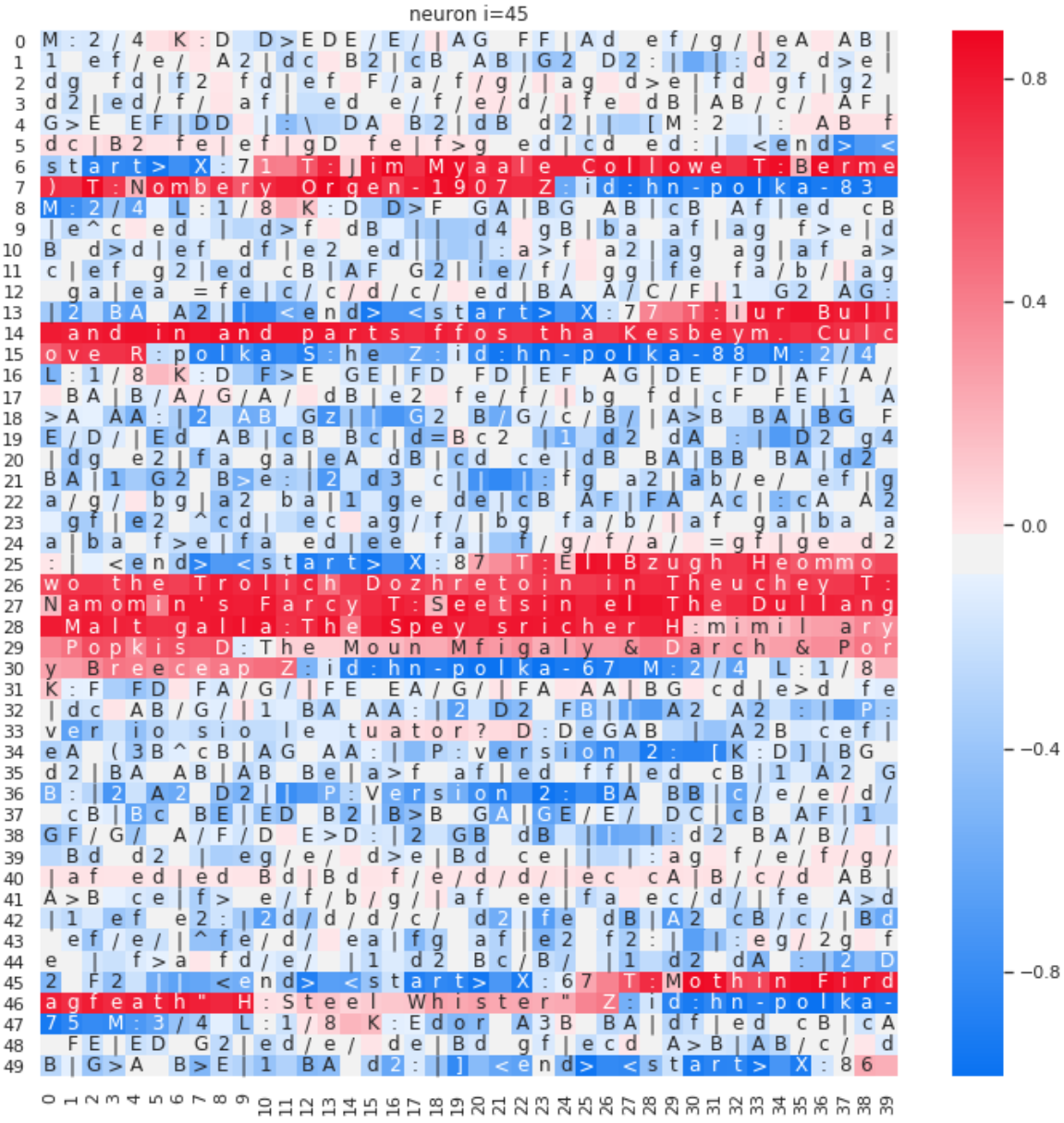

Figure 37: Heatmap for a neuron maximally activating for the tune title

4 Contributions

All author's implemented their own versions of LSTM, which was later combined into a single source code to set up baselines for all other experiments. The work on different experiments was similarly divided equally. Ambareesh wrote the code for Vanilla RNN. Various experiments with temperature were carried out by Nasha. Gitika was responsible for implementing the sampling module. Farheen was responsible to write code for generating heatmaps. Gokce worked on the coding of dataloader, model, training and generating music.

References

[1] Florian Colombo and Wulfram Gerstner. Bachprop: Learning to compose music in multiple styles. *CoRR*, abs/1802.05162, 2018.

[2] Karsten Eckhardt. Choosing the right hyperparameters for a simple lstm using keras. `https://towardsdatascience.com/choosing-the-right-hyperparameters-for-a-simple-lstm-using-keras-f8e9ed76f046`. Accessed: 2019-03-01.

[3] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber. Lstm: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, Oct 2017.

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[5] Andrej Karpathy, Justin Johnson, and Fei-Fei Li. Visualizing and understanding recurrent networks. *CoRR*, abs/1506.02078, 2015.

[6] Nikhil Kotecha and Paul Young. Generating music using an LSTM network. *CoRR*, abs/1804.07300, 2018.

[7] Zachary Chase Lipton. A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019, 2015.