# Context-Based Music Composition using RNN Generative Model

**Apurva Pathak**
A53097569
appathak@ucsd.edu

**Kshitiz Gupta**
A53104364
ksg005@ucsd.edu

**Chaitanya Baratam**
A53104872
cbaratam@ucsd.edu

**Ritvik Jaiswal**
A53090298
rjaiswal@eng.ucsd.edu

**Puneeth BommiReddy**
A53093725
pbommire@eng.ucsd.edu

## Abstract

In this paper, we present a recurrent neural network that generates music compositions based on auxiliary information such as genre (Irish, Swedish and French). We represent the music in Abc notation, a text-based music notation system, so that a character-level recurrent neural network could be used for generating music composition. Our results are based on a corpus of over 2500 music files collected from Henrik Norbecks Abc Tunes. Our network, composed of LSTM units, produces music in Abc notation tailored to a genre. Quantitative and qualitative evaluation demonstrates that the model learns the rhythm, keys and tempo of music from the training music files. Feature visualization helped us interpret the activation levels of various neurons in the penultimate layer of the network when the music sequence were feeded to the network. Since our model generates music one character at a time, it also adds the metadata information into the generated music file like title, composer, etc. without any machinery explicitly dedicated to the purpose. A live demonstration of the project can be found on the project's website. [1]

## 1 Introduction

Our work is motivated by the success of *BeerMind* [5] which uses a character-level Recurrent Neural Network (RNN) to synthesize beer reviews tailored to star rating or category using an auxiliary input fed to the network at each time step. Our goal is to use a similar model to generate different genres of music using the context as the auxiliary input. Through a character based representation of musical notes as training data, we generate a sequence of notes using various RNN architectures and compare their performance. Further, we attempt to generate different genres of music using an auxiliary input that is fed to the network at each time step. We then visualize the learned parameters of our networks to understand what features the network has learnt.

Recurrent Neural Networks (RNNs) trained on the word-level i.e. with a word as input at each time-step suffer from the computational complexity that accompanies the very large vocabulary of legal words in any given context [5]. This is where character-level RNNs come in. They have been shown to generate coherent passages but suffer from being unable to decide which set of the previous inputs are relevant to the current output i.e. they are unable to effectively model the *context* [5][12]. They have been used to generate music with limited success. They fail to capture global structure, which most often manifests as generation of repetitive segments of text or music

[1]Live demonstration of context based music generation (`http://ec2-54-67-49-59.us-west-1.compute.amazonaws.com:8000/tunes`)

[1]. Being unable to decide when to stop generation is also one of the problems with this model. However, this can be combated by augmenting the vocabulary with suitable start and stop tags. Boulanger Lewandowski, Bengio, Vincent (2012) [3] talk about the difficulty in training RNNs and corresponding optimization techniques.

Character-level Long Short-Term Memory Networks (LSTMs) demonstrate the ability of RNNs to model sequences on multiple time scales simultaneously, i.e., they learn to form words, to form sentences, to generate paragraphs of appropriate length, etc. This makes them able to determine the context effectively as compared to a vanilla RNN. Nayebi and Vitelli (2015) [4] conclude that LSTMs can successfully model long term dependencies present in musical sequences effectively. Eck and Schmidhuber (2002) [2] used an LSTM based model for music composition to successfully learn the global structure of a musical form and use that information to compose new pieces in that form. The model was used to generate new instances of a bebop-jazz variation of standard 12-bar blues.

However, we wish to add an auxiliary input such as Swedish, French and Irish, etc. to determine the kind of music it should generate. Since the LSTM could possibly *forget* the auxiliary input in a few steps. Lipton, Vikram and McAuley propose that this context deciding auxiliary input be concatenated along with the input at all time steps. While it might seem redundant to replicate the input context at each sequence step, but by providing it, we eliminate pressure on the model to memorize it [5]. They have used such a model to generate beer reviews and it looks to be a plausible solution for the context based note-level music we are trying to generate.

However LSTMs are not without limitations. Boulanger, Lewandowski and Bengio, Vincent (2012) [3] talk about the difficulty in training RNNs and optimization techniques. The limitations of LSTMs include generating a single expected value, as opposed to a full conditional distribution as generated by RBMs (which allows for greater variety in generated music). They combined the time dependencies modelled by an RNN with an RBM as part of their training model.

In this paper, we have used LSTM with dropout to generate music one character at a time. The music was represented as a sequence of characters in the ABC format. We have used three types of music - Irish, French and Swedish tunes. The context is modelled as a concatenated input (eg. {(Irish: [1,0,0]}, {(French: [0,1,0]}). We then try to validate our results.

This section is followed by a discussion of the dataset and its representation. Following which, the architecture and mathematical details of the models used are discussed. Finally, we conclude the paper with a discussion of experiments, results and their analysis. Our quantitative and qualitative analysis shows that our model learnt the syntax of Abc notation and generated good quality music conditioned on the genre (Irish vs French vs Swedish).

## 2 Dataset and Representation

### 2.1 Dataset

We have scraped music files from *Henrik Norbeck's Abc Tunes* [14] and *Musique Abc Tunes*[15]. The former is a repository of 2180 Irish and 388 Swedish tunes in Abc notation (described in 2.2) made available on the web as a free resource whereas the later contains a collection of 553 French tunes in Abc notation. In addition to music notes and rhythm, these datasets also contain the metadata information like title, composer, source, etc. We have used all the tunes for training of our model and to perform experiments.

### 2.2 Data Representation

We represent the music in Abc notation [13]. Abc is a text-based music notation system designed to be comprehensible by both people and computers. Music notated in abc is written using characters - letter, digits and punctuation marks - on paper or in computer files. An abc tune itself consists of a tune header and a tune body, terminated by an empty line or the end of the file. The tune header is composed of several metadata information fields like title, composer, etc. The tune header should start with an X:(reference number) field followed by a T:(title) field and finish with a K:(key) field. The tune body, which contains the music code, follows immediately after. Figure 1 explains different sections of abc notation using a sample music in its abc notation from Henrik Norbeck's Abc Tunes [14].

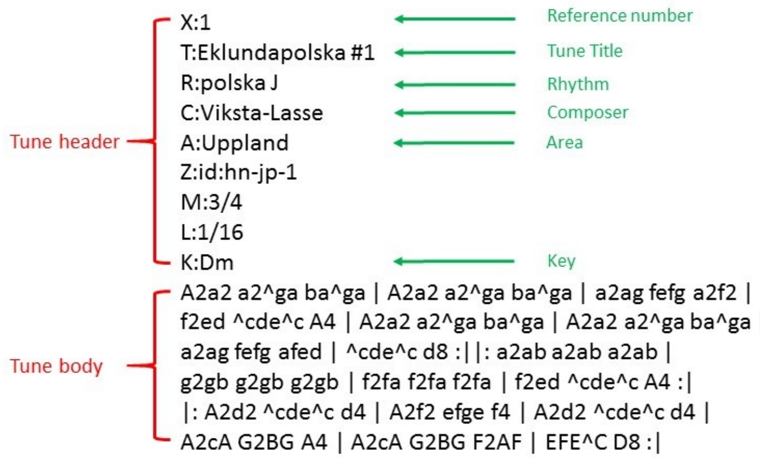



Figure 1: Abc notation of music file

## 3 Recurrent Neural Network Methodology

These differ from traditional feed-forward neural networks in the sense that the inputs and outputs are not independent. There exist feedback connections between nodes in a way that these connections form a directed cycle. RNNs have an internal memory which stores information about what has been calculated so far, thus giving them the capability of handling sequential data.

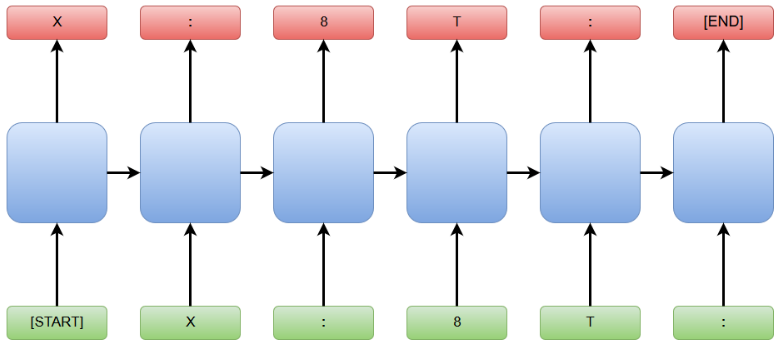
The problem with RNNs is that of the vanishing/exploding gradient. When trained on an arbitrarily long sequence, if the weights are small the gradients shrink exponentially and if the weights are large they grow exponentially. This problem is overcome by using LSTMs (Long Short Term Memory), introduced by Hochreiter and Schmidhuber (1997) [6]. Each memory cell has an internal state s in which activation is preserved along a self-connected recurrent edge. Each cell also has sigmoidal input, output and forget gates. Information gets into the cell whenever its input $(i)$ gate is on and it stays in the cell until its forget $(f)$ gate is turned on. The information can be read from the cell by turning on its output $(o)$ gate. Several layers of LSTMs can be stacked together (Graves- 2013) [7]. At step $t$, each LSTM layer $h_l^{(t)}$ receives as input the output from layer $h_{l-1}^{(t)}$ at step $t$ and the output from layer $h_l^{(t-1)}$ at step $t-1$. As a base case, we take $h_0^{(t)} = x^{(t)}$ and $h_l^{(0)} = 0$. The following are the equations to calculate the forward pass through an LSTM:

$$
\begin{aligned}
g_l^{(t)} &= \phi(W_l^{gx} h_{l-1}^{(t)} + W_l^{gh} h_l^{(t-1)} + b_l^g) \\
i_l^{(t)} &= \sigma(W_l^{ix} h_{l-1}^{(t)} + W_l^{ih} h_l^{(t-1)} + b_l^i) \\
f_l^{(t)} &= \sigma(W_l^{fx} h_{l-1}^{(t)} + W_l^{fh} h_l^{(t-1)} + b_l^f) \\
o_l^{(t)} &= \sigma(W_l^{ox} h_{l-1}^{(t)} + W_l^{oh} h_l^{(t-1)} + b_l^o) \\
s_l^{(t)} &= g_l^{(t)} \odot i_l^{(t)} + s_l^{(t-1)} \odot f_l^{(t)} \\
h_l^{(t)} &= \phi(s_l^{(t)}) \odot o_l^{(t)}
\end{aligned}
$$

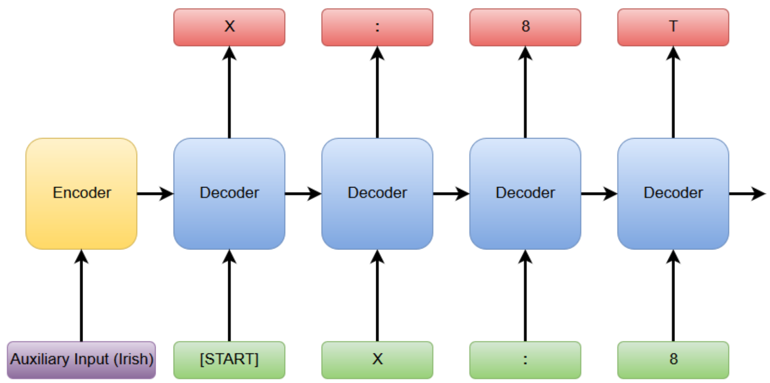
Here, $\sigma$ denotes an element-wise sigmoid function, $\phi$ denotes element-wise $tanh$, and $\odot$ is element-wise product.

### 3.1 Generative Recurrent Neural Networks

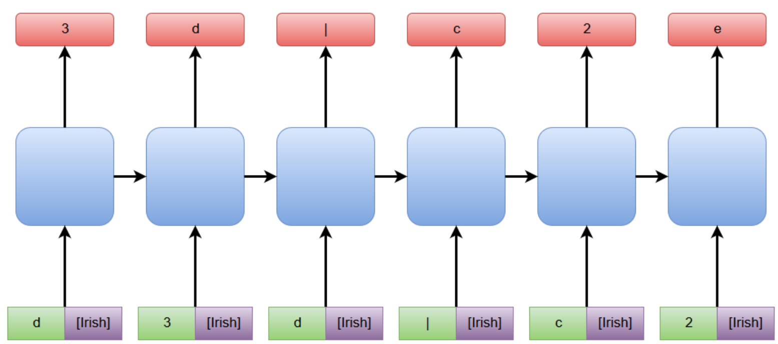
Similar to the beer review paper [5], we build on the generative RNN model of Sutskever et al. (2011; 2014) [8] [9]. A generative RNN predicts the next token in a sequence $(x^{t+1})$ given the




(a) Vanilla generative recurrent neural network




(b) RNN with LSTM units with no dropout




(c) RNN with LSTM units with dropout

Figure 2: Different types of RNNs

sequence up to that point $(x^1, ..., x^t)$. Thus, the output string is simply the input string shifted by one token. (Figure 2a). The output layer is fully connected with softmax activation. Cross entropy is the loss function used for training.

The model can then be used to generate arbitrarily long sequences given some starting token and state by passing the generated output as input in the subsequent step and repeating.

### 3.2 Concatenated Input Recurrent Neural Networks

In our experiment, we generate music based on context given in the form of an auxiliary input $x_{aux}$. In the model described by Sutskever et al. (2014) [9], and Karpathy and Fei-Fei (2015) [10], the auxiliary input is encoded and passed as an initial state to the decoder (Figure 2b). This auxiliary input has to be preserved across many sequence steps which is why it is successful in producing short image captions and the like, but impractical in generating long music notes.

To overcome this issue of having to remember the auxiliary input across each step, we simply concatenate $x_{aux}$ with each character representation in the sequence (Figure 2c). At train time, $x_{aux}$ is a feature of the training set; however, during the prediction phase, $x_{aux}$ is fixed and concatenated with each character in the sequence to produce the next character.

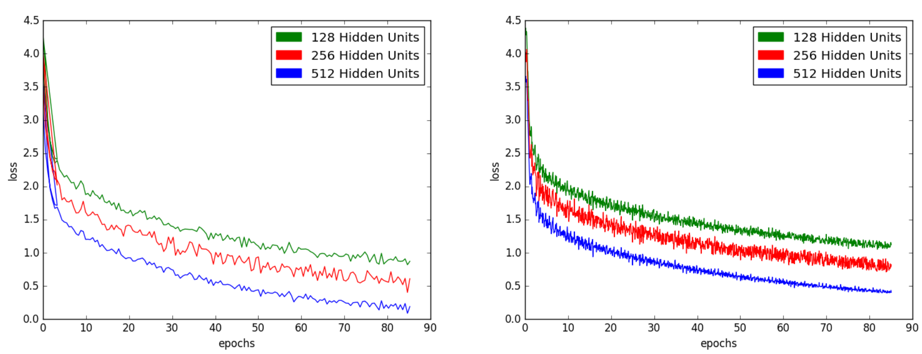
### 3.3 Weight Transplantation

Models with auxiliary inputs are considerably harder to train as compared to a typical unsupervised model. To overcome this problem, the model is first trained without the auxiliary input. Once the weights have converged, they are then used in the concatenated input model. The extra set of weights from the input layer to the first hidden layer are initialized to zero. This is similar to the pre-training common within the computer vision community (Yosinski et al., 2014) [11]. Here, instead of new output weights, we train new input weights.

## 4 Experiment and Results

All the experiments were performed using Lasagne framework [16] built over Theano. Before training, we concatenated all the tunes represented in ABC format, delimiting each of the tunes with <START> and <END> tags, to generate our training data. Before generating music based on genre we did feasibility study for music generation using recurrent neural networks. We trained a vanilla RNN to see if music generation in this way is possible. It had single hidden layer with 100 nodes. We split our training data into mini-batches of size 512, which are in turn split into segments of sequence length 25. The weights are updated using AdaGrad with a learning rate of 0.1. We found that the model generated good quality music in Abc format after some epochs of training. This gave us the confidence to move forward with the idea of music generation using RNN. Samples of music generated using vanilla RNNs can be found at `https://goo.gl/Iw7IU2`.

Once we established the feasibility of generating music using RNNs, we trained a LSTM network with 2 hidden layers, with 512 hidden nodes in each. We experimented with various size of hidden units before finalizing the number of units to 512. Decreasing the number of hidden nodes lead to a higher error rate, which lead to the interpretation that the network seems to underfit the data (Figure 3). Further increasing the number of units beyond 512 lead to GPU memory issues.




(a) LSTM units with no dropout for different hidden nodes

(b) LSTM units with dropout for different hidden nodes

Figure 3: Loss plotted against epochs for different no. of hidden nodes

We also experimented by varying the mini-batch sizes. As the average length of a melody in the training data was found to be around 400, we observed that a small mini-batch size could not accommodate a complete melody with <START> and <END> tags. With a small mini-batch the network couldnt learn how to start or end a melody, leading to abrupt starts and finishes to the generated tunes. A mini-batch size greater than 512 lead to memory errors on the GPU. So, we split the training data into mini-batches of size 512 which was in turn split into segments of sequence length 64. To generate music of a particular genre, we provided a one-hot encoded auxiliary input

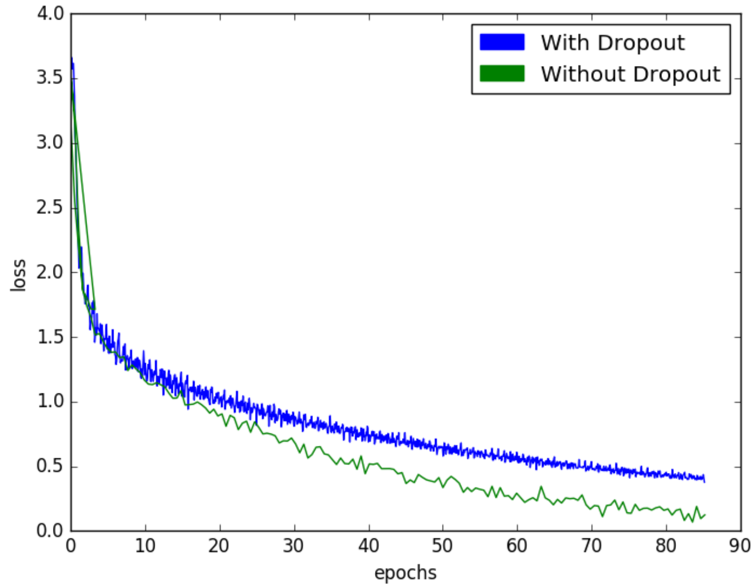



Figure 4: Plot of loss against epochs for LSTM with dropout and no dropout

representing Irish, French and Swedish melodies to the LSTM. Training was done in mini-batches and the weights were updated using AdaGrad with a learning rate of 0.1. To combat exploding gradients, we clip the elements of each gradient at $\pm 100$.

Next, we introduced dropout of 0.1 in LSTM. In the LSTM network with no dropout, training loss dropped quickly implying that the network is overfitting the training data (Figure 4). This lead to the network generating music very similar to the training melodies, defeating the purpose of generating new music. With the introduction of dropout, we were able to resolve this issue, thereby leading to generation of novel melodies.

Training the network with the auxiliary input leads to very slow learning. So, we first trained a character-level LSTM network with dropout (i.e. without the auxiliary input), on the combined Irish, French and Swedish music datasets. We then transplanted the weights of the trained network into the concatenated input network for fine-tuning. A live demonstration of the model can be found at `http://ec2-54-67-49-59.us-west-1.compute.amazonaws.com:8000/tunes`.

X:18

T:Solle else Toll Ta er ools To ole an er oe er or on er ool en er ool en er or on er oll an er on er or on on er or on er oo so er on er or on er oor War oo sar ar on er or ole Ar or on er or on er ool en er ols ar ool ia dor ian en er ole tar ion Ah ton

Z:id:hn-slide-

M:6/8

L:1/8

$\boxed{\text{LK:1}}$ Should be K:1

A2A A2A | A2A A2 | A2A A2A | A2A A2A | A2A A2A | A2A A2A | A2A A2A | A2A A2A | A2A A2A | A2A A2A | A2A A2A | A2A A2A | A2A A2A | A2A A2A | A2A A2A | A2A A2A | A2A A2A | A2A A2A | A2A A2A | A2A A2 | A2A A2 | A2 A2 A2 | A2A A2 | A2A A2 | A2A A2 | A2A A2 | A2 A2 | d2 d2 | d2 d2 | d2 d2 | e2 d2 | e2 d2 | d2 d2

Figure 5: Music generated after 10 epochs

```
X:8
T:Pols ar Rer Ror Roll Rar Rols an Rers on ser son sar son sols or
son son sor son son son son sor son sor son son son sor son
sors on sol sor son sons
Z:id:hn-sl-13
M:3/4
L:1/8
K:D
A2 A2 A2 | A2 A2 | A2 A2 | B2 A2 | B2 A2 | B2 A2 |
A2 A2 | B2 A2 | B2 A2 | B2 A2 | B2 A2 | B2 A2 | B2 A2 | B2 A2
| A2 A2 | B2 A2 | A2 A2 | A2 A2 | A2 A2 | B2 A2 |
A2 A2 | B2 A2 | B2 A2 | B2 A2 | B2 A2 | B2 A2 | A2 A2 | B2 A2
| A2 A2 | B2 A2 |
A2 A2 | B2 A2 | B2 A2 | B2 A2 | B2 A2 | B2 A2 | B2 A2 |
A2 A2 | B2 A2 | B2 A2 | A2 A2 | B2 A2 | A2 A2 | A2 A2 | A2 |
A2 A2 | A2 A2 |
A2 A2 | A2 A2 | A2 A2 | A2 A2 | A2 A2
```

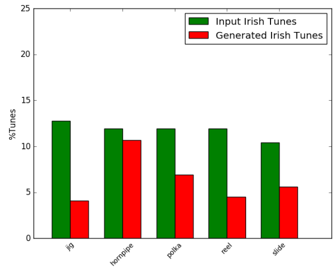
Figure 6: Music generated after 50 epochs

```
X:2
T:Polsk of The
R:slide
D:Seeral Ossen
Z:id:hn-slide-6
M:6/8
L:1/8
K:Ador
A2 | G2 A2 A2 | B2 GE GE | E2 A2 A2 | B2 G2 BG |
A2 A2 c2 | B2 BA BA G2 | B2 BA/G/ GE |
D2 G2 BG | B2 AB/c/ BA |1 G2 G2 :|2 G2 G2 B2 ||
```

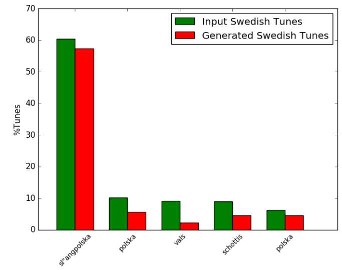
Figure 7: Music generated after 100 epochs

In the generation phase, the network initially generated random text. After 10 epochs, the network started generating text which followed the ABC format with some syntax errors (Figure 5). After 50 epochs the network started generating syntactically correct ABC text which can be played in the Midi format. However, this output had the problem of repeating notes for long duration (Figure 6). After 80 epochs, the network started generating ABC text which followed the chord structure of a real melody (Figure 7).

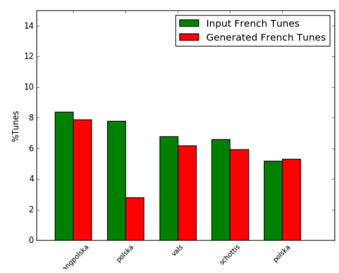
Although we could differentiate Irish, Swedish and French melodies by listening to it, there was no concrete metric in the literature to quantitatively differentiate them. So, we compared the top 5 rhythms (*R* field in the tune header in Abc notation) generated by the network for each tune and found that these rhythms are the same in the generated tune and the training tune for different genres of melodies. Figure 8a shows the top 5 rhythms of the training Irish data which are consistent with the top 5 rhythms of the generated data. The same was observed for the Swedish and French data which can be seen in Figure 8b and Figure 8c respectively. Further, we projected the activations of neurons in the last but one layer (the LSTM layer just before the softmax) into two dimensions using t-SNE. We observed several small clusters representing different note combinations within each genre, which seems to be clearly separable from the note combinations in other genres (Figure 9a). Although the genres (Irish, French and Swedish) seem to be separated from each other but in




(a) Top 5 Irish rhythms on training and generated data



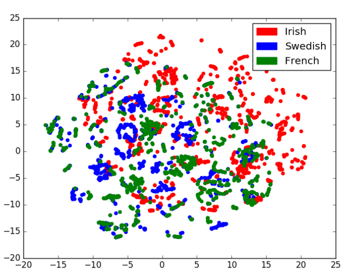

(b) Top 5 Swedish rhythms on training and generated data



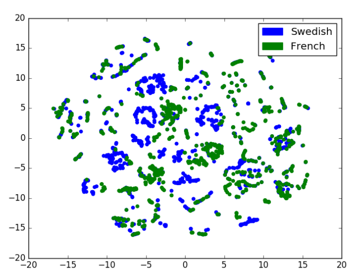

(c) Top 5 French rhythms on training and generated data

Figure 8: Plot of top 5 rhythms on training and generated data for different genres

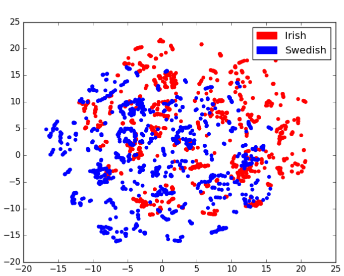
order to get a better visualization we plotted it two genres at a time as shown in Figure 9b 9c 9d. All clusters in these figures seem to have negligible overlap with one another which means that the network learnt different features for different auxiliary inputs. All these visualizations helped us verify that the network is actually generating music based on the genre provided as input.



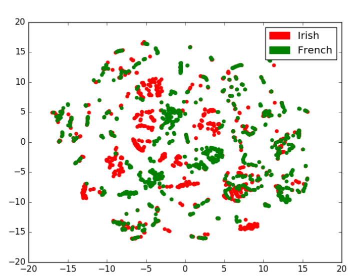

(a) t-SNE for Irish, French and Swedish music




(b) t-SNE for Swedish and French music




(c) t-SNE for Irish and Swedish music

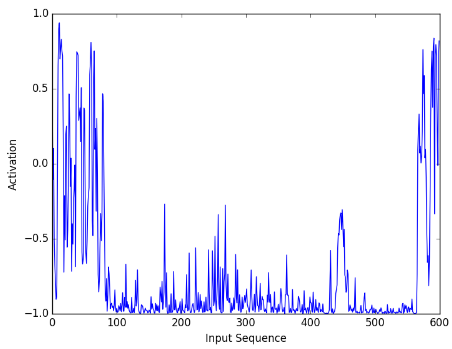



(d) t-SNE for Irish and French music

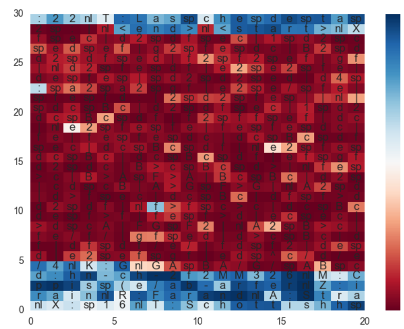
Figure 9: Projection of activations of neurons in the final LSTM layer in 2-d using t-SNE.

```
Header
X: 16
T:Schottishh rain
R:Farand
A:Strappis (e/ab-arfer
Z:id:hn-ch-2f2MM326
M:C/4
K:G
Body
GA BA/G/ AB cd|e2 ef gf|ed ^c/d/ef|df df/e/|fd f2|de fe/f/|gf ed|d>c Bc|d>d cA|FG F2|
A2 B>c|de f>f|fe f>d|ec ec|d2 df|
f>f c>c|dc Bc|d>f ec|dc Bc|df f>d|cd cB|A>G F>G|
A2 d>c|B>A F>A|Bc Bc|d2 d2 dc|B>c Bc d>|
fe dc Bc|dc Bc df|ef gf ec|dc Bc df|
e2 fe fe|fe fe dc|dc Bc df|
e2 fe fe|fe fe dc|dc Bc df|f2 ff ef|dc dc Bc|d2 df ec|1 d2 fe fd:|2 d2 fe ||
|: a2 a2 gf|e2 e/ fe|de fe | d2 d2 ed|d4 |
d2 fe df|e2 e2 fe|d2 df ed|f2 f2 ef|gf ed ef|g2 fe dc|B2 df ec|d2 df ec|1 d2 c2 :|
<end>
<start>
Header
X:22
T:Las che de ta S
```

Figure 10: Input text for feature visualization




(a) Activation of LSTM plotted against the index of the input sequence



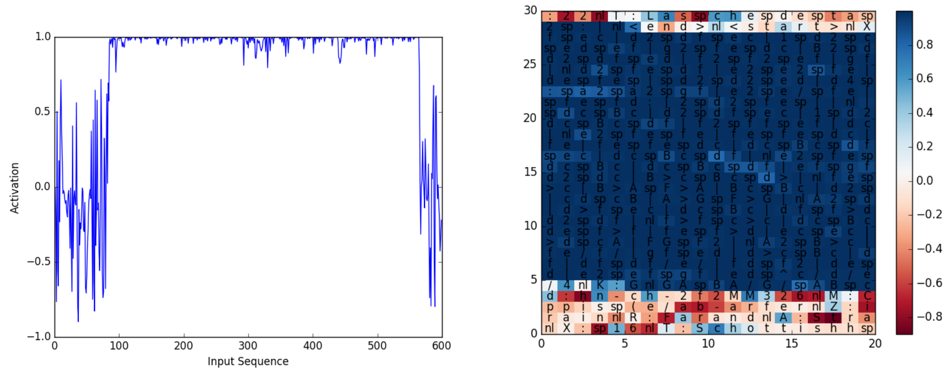
(b) Heatmap showing that the neuron gets fired for tune header

Figure 11: Activation of the neuron that represents the tune header.

## Feature Visualization

To understand the features generated at the last but one layer of the network (i.e. the LSTM layer immediately preceding the softmax output layer) feature visualization for this layer was performed. Figure 10 shows the input (of length 600) used for this task. The sections of the input representing the tune header and body are marked in the figure for clarity. Figure 11 12 13 are the heat map plots of different neurons in this layer. This layer contains 512 neurons and each heat map is a plot of the activity of one of the neurons as the input sequence passes through it. Of the 512 neurons, we were able to interpret the activity for only 2-3% of the neurons. The scale on the right indicates that a blue color implies an activity closer to 1 and a red color implies an activity closer to -1. The space and newline characters in the heat maps are represented by *sp* and *nl* respectively.

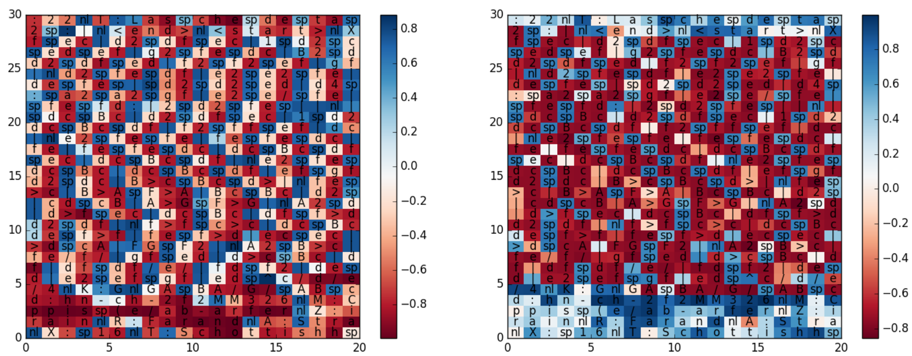
Figure 11b shows the activation levels for the neuron which inhibits the activation level of input characters belonging to the body. Characters in the header like 'X', ':', etc. show activation higher than those which belong to the body of the input. Figure 11a is another representation of this heat map, plotting activation as a function of the input sequence. Note that the start of the input sequence (shown in Figure 10) corresponds to the header with characters having high activation levels as compared to the middle part of the input sequence which corresponds to the body resulting in low




(a) Activation of LSTM plotted against the index of the input sequence

(b) Heatmap showing that the neuron gets fired for tune body

Figure 12: Activation of the neuron that represents the tune body.



(a) Heatmap showing that the neuron gets fired for space, newline and pipe (|) character.

(b) Heatmap showing that the neuron gets fired for spaces and newline character inside tune body

Figure 13: Heatmap showing that the neuron gets fired for various special characters.

activation levels, and the concluding part is the header for next generated tune which once again exhibits enhanced activity.

Similarly, Figure 12b and 12a plots the activity of the neuron which fires whenever an input character belonging to the body of the input sequence passes through it. It can be seen clearly that for characters that belong to the body of the sequence, therefore appearing in the middle part of the sequence, activation levels are closer to 1 as compared to characters in the header or footer.

Figure 13a and 13b correspond to neurons that fire for space and newline characters in the body of the input sequence, and newline, space and pipe characters appearing anywhere in the sequence, respectively.

## 5 Conclusion

In this paper, we demonstrate the ability of character-level LSTM network to generate relevant melodies conditioned on auxiliary input. Incorporation of dropout to the network was shown to improve the performance of the system. This is also the first work, to our knowledge, to generate the music conditioned upon genre. Our qualitative and quantitative analysis shows that our model learnt the syntax of Abc notation and generated good quality melodies. Further, our analysis support that the model was able to generate different music conditioned on the auxiliary input. In addition to this, the feature visualization help us interpret the features learnt by some of the neurons. The number of genres used for this project were restricted to three due to the unavailability of proper dataset, however, the same model can be extended for any number of the genres.

## 6 Contributions

Apurva and Kshitiz wrote the code for LSTM generative model. Puneeth and Ritvik wrote the code for character level vanilla RNN. Chaitanya wrote the code to scrape data and to add dropout. Apurva and Kshitiz wrote the code for feature visualization and generated the plots for results and experimentation. Ritvik, Puneeth and Chaitanya wrote the most of the parts of the report and were helped by Apurva and Kshitiz. Apurva and Kshitiz built the website.

## References


[1] Mozer, Michael C., (1994) Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing., *Connection Science*, pp. 247-280.

[2] Eck, Douglas., and Juergen Schmidhuber, (2002) A first look at music composition using lstm recurrent neural networks.

[3] Boulanger-Lewandowski, Nicolas and Bengio, Yoshua and Vincent, Pascal., (2012) Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription., *arXiv preprint arXiv:1206.6392*

[4] Nayebi, Aran and Vitelli, Matt., (2015) GRUV: Algorithmic Music Generation using Recurrent Neural Networks.

[5] Lipton, Zachary C and Vikram, Sharad and McAuley, Julian., (2015) Capturing Meaning in Product Reviews with Character-Level Generative Text Models., *arXiv preprint arXiv:1511.03683*

[6] Hochreiter, Sepp, and Jrgen Schmidhuber. "Long short-term memory." *Neural computation* 9, no. 8 (1997): 1735-1780.

[7] Graves, Alex. "Generating sequences with recurrent neural networks." *arXiv preprint arXiv:1308.0850* (2013).

[8] Sutskever, Ilya, James Martens, and Geoffrey E. Hinton. "Generating text with recurrent neural networks." In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 1017-1024. 2011.

[9] Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." In *Advances in neural information processing systems*, pp. 3104-3112. 2014.

[10] Karpathy, Andrej, and Li Fei-Fei. "Deep visual-semantic alignments for generating image descriptions." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3128-3137. 2015.

[11] Yosinski, Jason, Jeff Clune, Yoshua Bengio, and Hod Lipson. "How transferable are features in deep neural networks?." In *Advances in Neural Information Processing Systems*, pp. 3320-3328. 2014.

[12] Andrej Karapathy., The Unreasonable Effectiveness of Recurrent Neural Networks., `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`.

[13] The abc music standard 2.1, `http://abcnotation.com/wiki/abc:standard:v2.1`.

[14] Henrik Norbeck's Abc Tunes, `http://www.norbeck.nu/abc/`.

[15] Musique Abc Tunes, `http://michel.bellon.free.fr/Musique/abc/Provence2.abc`.

[16] Lasagne, `https://github.com/Lasagne/Lasagne`.