Tudududu: Generating Music with RNNs

Bhanu Renukuntla, Saket Sharma, Swaroop Gadiyaram, Venkatesh Elango, Vikas Sakaray
Department of Electrical and Computer Engineering
University of California, San Diego

Abstract

This work deals with using *Recurrent Neural Networks* to generate music. A simple character-level Recurrent Neural Network (RNN) is trained on an ABC notation music file using the *Keras* library in Python. The trained model is then primed to generate music. Further, the effects of *dropout*, varying number of hidden nodes and different optimizers used for training are investigated on music generated. The activations of the hidden layer neurons of the trained model are visualized for a sample music (generated) input in the form of a heatmap. The highest achieved validation accuracy was 47.68 % and the network could generate decent music.

1 Introduction

Recurrent Neural Networks (RNNs) are a class of Neural Networks wherein the connections between units form directed cycle. This creates an internal state of the network which allows it to exhibit dynamic temporal behavior. Unlike feedforward neural networks (ex. CNNs), RNNs can use their internal *memory* to process arbitrary sequences of inputs. This makes them applicable to tasks such as unsegmented connected handwriting recognition, speech recognition as well as the tasks similar to that one addressed in this work. These networks are trained using the *backpropagation* algorithm. The whole network still expresses a single differentiable score function: from the raw input on one end to class scores at the other.

In this work, we train a simple RNN on a corpus of music in ABC format and the trained model is used to 'generate' music. Since this is an elementary model, we don't expect amazing music but instead have some reasonably good tunes spread through the generated sequence.

2 Music Generation

2.1 Dataset & Network

The dataset is a corpus of music tunes in the ABC format. Each sample music tune consists of a header followed by the body. A simple RNN with a single hidden layer with *tanh* activated neurons is used - we experiment using different number of neurons in the hidden layers. The network is implemented in Keras.

2.2 Training

We use a 80:20 split to divide the data into training and validation sets. All the characters are converted to *one-hot* encoded representations. Since we are interested in implementing a character-level RNN, the network is fed a slice 25 characters of the training set as input and employ *teacher forcing* (i.e, given the n^{th} character, the network is trained to predict the next ($(n+1)^{th}$)). We use Keras to implement and train the network. The network has a softmax output, trained to minimize the cross-entropy loss and *Adam* optimizer is used . We track the loss on training and validation sets.

Once the network is trained, we use it to generate music by priming it with a sequence of characters to get it started. The generated probability distribution at the softmax layer is used to sample a character that is to be fed back to the input layer and this is repeated to generate a tune of reasonable length. In other words, we flip an n-sided coin based on the generated probability distribution to sample a character that is to be fed back to the RNN. (Note that the network generates music in the ABC notation which is to be converted to a playable audio format (eg. .midi) for listening.)

We carry out the following experiments:

- With a network consisting of 100 hidden layer neurons, we generate 2 sample pieces for each of T=0.5, T=1 and T=2, where T is the temperature parameter. The slightly modified softmax function incorporating a *temperature* term (T) is given below. Using a higher value for T produces a softer probability distribution over classes.

$$y_i = \frac{exp(a_i/T)}{\sum_j exp(a_j/T)} \tag{1}$$

- We alter the network by changing the number of hidden layer units: networks with 50, 75 and 150 hidden layer neurons are trained and the effects on training/ validation loss are observed.
- For the network with 100 hidden layer units, we try using dropout with p=0.1, 0.2 and 0.3. For each case, a sample music tune is generated. The effect of dropout on training speed and generated results is studied.
- With 100 hidden layer neurons, we train the network with different optimizers: Adagrad and RMSProp besides Adam.
- **Feature Evaluation:** Finally, we visualize the activations of a neuron in the hidden layer in response to a generated music sequence by forward-propagating it through the trained network. This is visualized as a heatmap.

3 Results and Discussion

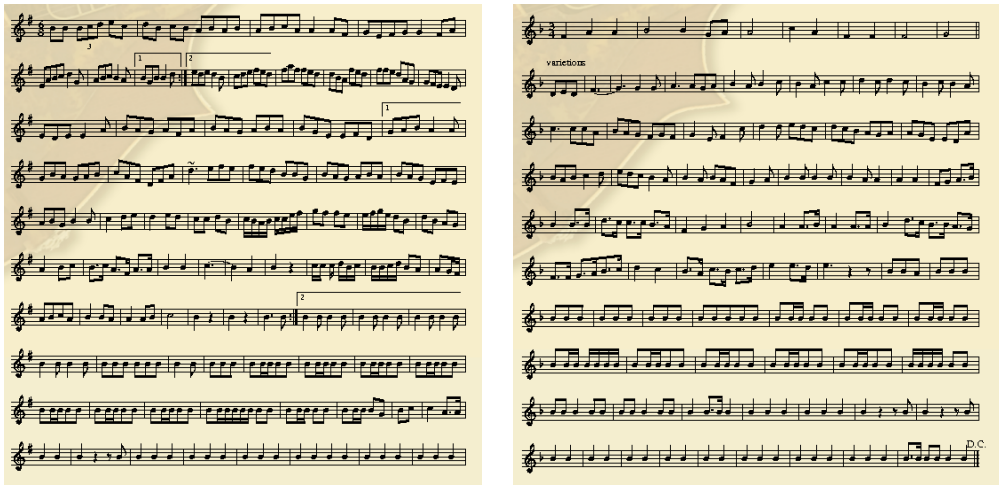


Figure 1: Two music files generated by using temperature 0.5

The music representations of generated music for different values of the temperature parameter (T) are shown in (Figures 1, 2, 3). The ABC notations can be found as separate attachments submitted along with this report. We observed that the tunes for T=0.5 and T=1.0, the generated tunes were relatively pleasant sounding than those generated using T=2.0. Also, higher T seemed to generate output that contained shorter length good sounding tunes in between some not so good tunes i.e,

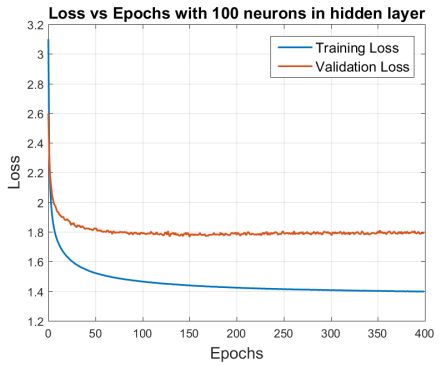


Figure 4: Loss vs Epochs with 100 neurons in hidden layer

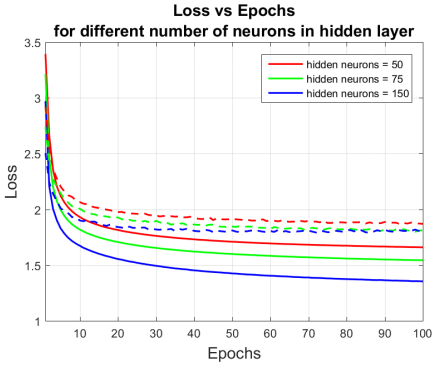


Figure 5: Loss vs Epochs for different number of neurons in hidden layer

the fraction of good sounding tunes generated seemed to be better for lower T. The network rarely produces good music for T=2.0. Other details and hyperparameters for training: batch size=50, epochs=100, optimizer= adam and default values for rest.

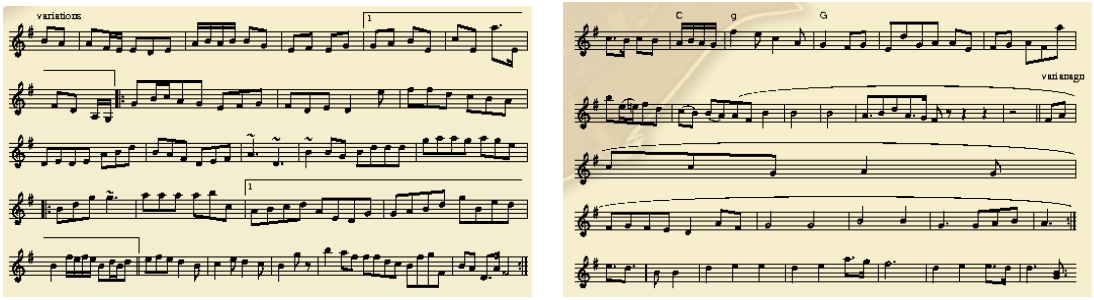


Figure 2: Two music files generated by using temperature 1.0

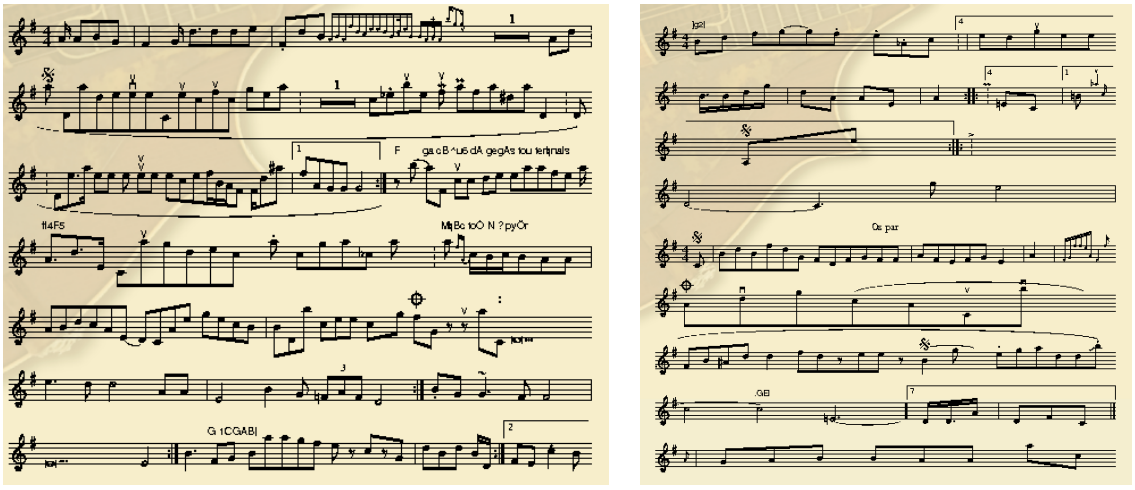


Figure 3: Two music files generated by using temperature 2.0

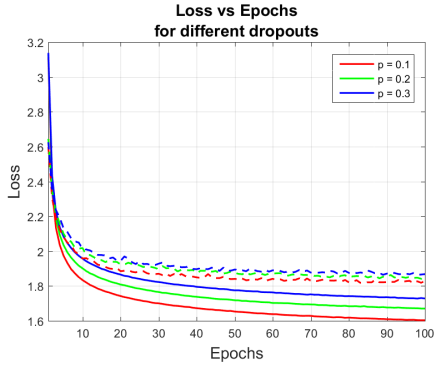


Figure 6: Loss vs Epochs for different dropouts

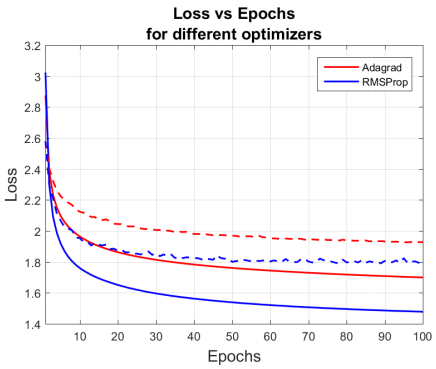
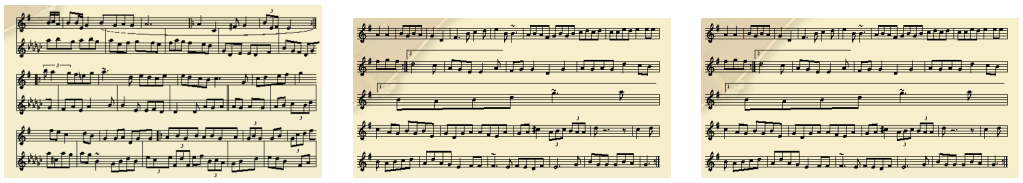


Figure 7: Loss vs Epochs for different optimizers

The training loss decreases monotonically with number of epochs. However, the validation loss doesn't follow a similar trend and we observe over-fitting phenomenon after some epochs (Figure 4). Also, as expected the loss on the validation set was higher than training loss. Validation losses in the plots are indicated by dotted lines and training losses are given by solid lines.

The loss on training and validation sets seem to change with number of units in the hidden layer. The magnitude of the loss goes down with increase in number of hidden layer neurons, and the network with 150 hidden layer units has the lowest loss, and the one with 50 hidden units has the highest. The trend of training and validation loss for a given number of hidden layer neurons was same as above. (Figure 5)

Increasing the dropout parameter leads to slower training speed and convergence as seen in Figure 6. The loss is higher for higher dropout. However, the music generated for dropout 0.1 was the best and sounded good to us (subjective). (Fig 8)



a b c

Figure 8: Music generated by using dropout a) $p = 0.1$ b) $p = 0.2$ c) $p = 0.3$

Interestingly it was observed that the behavior of loss also depends on the choice of optimizer used while training. *RMSProp* seemed to outperform *Adagrad* and contained longer/shorter length tunes and has a lower loss on both training and validation sets (Figure 7). Additionally, we noted that *Adam* performs the best (see plot in Figure 4). Also validation accuracies of 47.68 %, 47.03 % and 43.59 % were obtained with Adam, RMSProp and Adagrad respectively, a trend similar to that for model loss.

From the heatmap in Figure 9, we observe that the hidden layer neuron 1 has learned to recognize the header of the input music sequence. In the heatmap, *red indicates higher* activation and blue is lower activation. That is, it has higher activations for characters that form the header compared to those in the body. Figure 10 shows activation of the same neuron in response to a sequence from the training set.

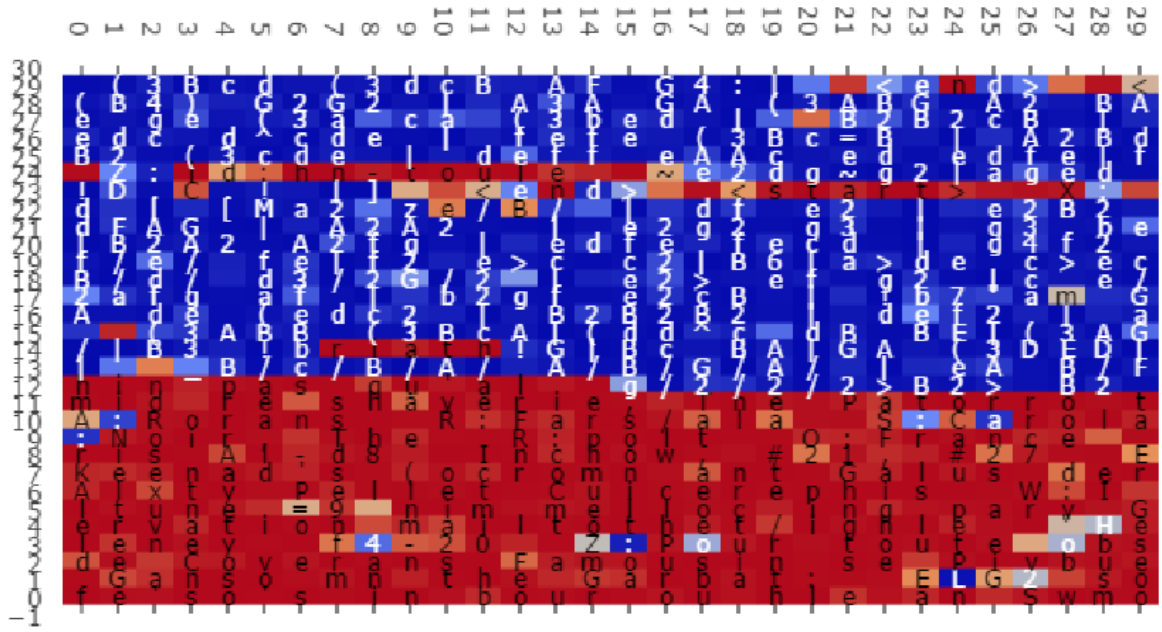


Figure 9: Heatmap of neuron 1 for generated music

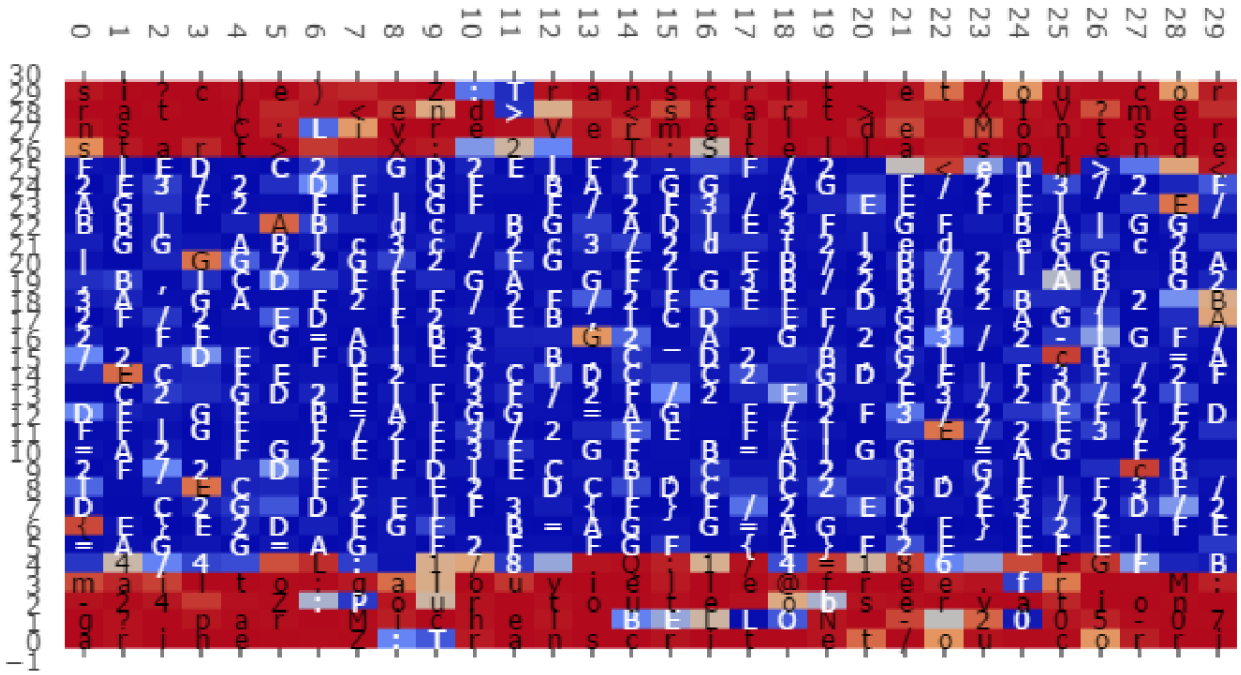


Figure 10: Heatmap of neuron 1 for sequence from training set

4 Summary

In this work, we used recurrent neural networks to generate music. A simple character-level Recurrent Neural Network (RNN) was trained on an ABC notation music corpus using the *Keras* library in Python. The trained model was then primed to generate music. We observed that using a value of temperature parameter greater than 1 gave more random outputs. However for T=0.5 and T=1.0, the network could generate good tunes. The validation loss was greater than training loss, as expected. Introducing more number of neurons in the hidden layer decreased loss while increasing the dropout parameter yielded slower training speed and convergence, with an increase in loss. The music sounded the most pleasant for dropout of 0.1 (subjective). Validation accuracies of 47.68 %, 47.03 % and 43.59 % were obtained with *Adam, RMSProp* and *Adagrad* respectively, a trend similar to that for model loss. We expect to get much better results by using LSTM based network.

5 References

1. *Efficient Backpropagation* by Yann LeCun, Leon Bottou, Genevieve B. Orr and Klaus-Robert Miller.
2. *Long Short-Term Memory* by Sepp Hochreiter and Jurgen Schmidhuber.
3. *Distilling the Knowledge in a Neural Network* by Geoffrey Hinton, Oriol Vinyals and Jeff Dean. eprint arXiv:1503.02531
4. *A Theoretically Grounded Application of Dropout in Recurrent Neural Networks* by Yarin Gal and Zoubin Ghahramani arXiv:1512.05287.
5. *Supervised Sequence Labelling with Recurrent Neural Networks* by Alex Graves.