GENERATING MUSIC WITH LSTM NETWORKS

Chengyu Dong
Computer Science and Engineering
cdong@eng.ucsd.edu

Rui Yang
Computer Science and Engineering
r3yang@eng.ucsd.edu

Shang Gao
Computer Science and Engineering
shgao@eng.ucsd.edu

Weifeng Hu
Computer Science and Engineering
weh031@eng.ucsd.edu

Ye Fan
Computer Science and Engineering
y1fan@eng.ucsd.edu

April 5, 2019

ABSTRACT

Recent development in Recurrent Neural Network such as LSTM has shown promising achievement in time-sensitive sequential data. This paper presents a LSTM model that generates music. We trained two recurrent network with one hidden layer, a Vanilla RNN and a LSTM network. It can be found that the LSTM outperforms the Vanilla RNN in terms of validation cross entropy loss as well as test cross entropy loss. The optimal hidden layer size is also found by changing the number of neurons in the hidden unit. During the generation stage, we experimented with different Temperature parameter T in softmax. We also noticed significant differences in music quality with temperature-based sampling and Arg-Max sampling.

***K*eywords** Recurrent Network · LSTM · Music Generation

1 Introduction

Deep learning has become a fast developing domain in recent years and is now used in the fields of predicting, classification and generating. Among these tasks, developing models to extract, analyze and generate music gains more and more interest considering generating good music might has not only artistic value but also commercial potential. Because of the sequential correlation between musical notes, models like Recurrent Neural Networks (RNNs) and Long short-term Memory (LSTMs) which can memorize former input are widely used to modeling the process of generating music.

In this paper, we would like to explore how Recurrent Neural Networks (RNNs) can deal with music generation. We concatenated a series of music files in ABC format as shown in Figure 1 to one single ABC file containing multiple tunes and divided it into a lot of chunks of fixed length. We passed these chunks in sequence to our LSTM and generated music with this trained LSTM model. We also compared the different architectures of RNNs based on their performance.

2 Methods

2.1 Data Preprocessing

As described in **Introduction** section, the music files are given in ABC format with music tunes concatenated. Each tune is delineated by <start> and <end> tags. In order to convert text data into features for our model, we perform

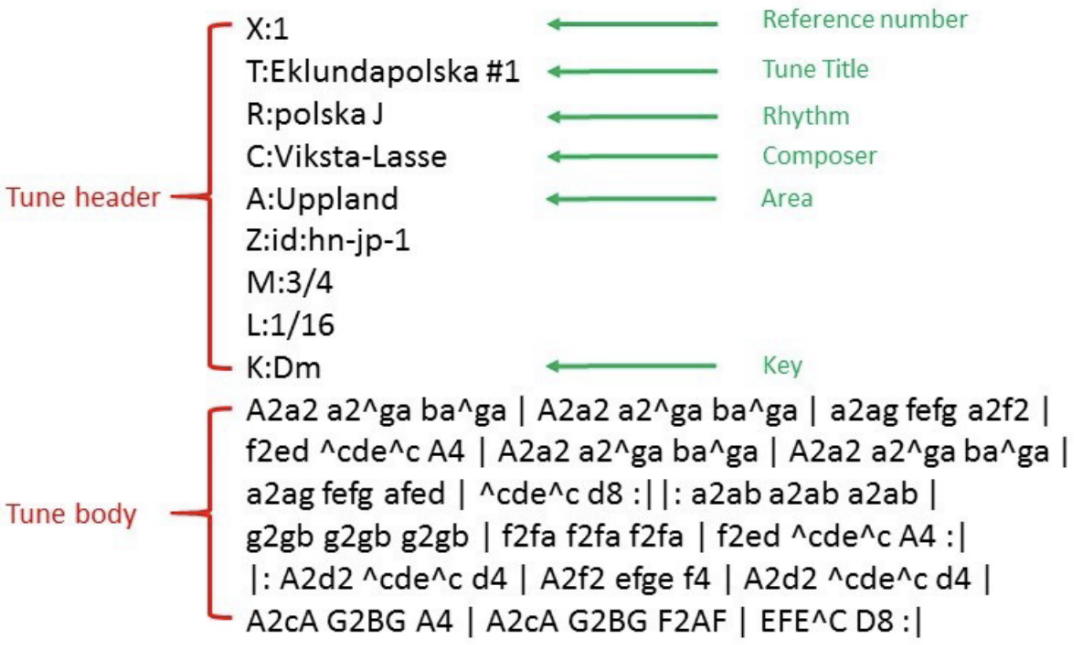


Figure 1: The example music data in ABC format

one-hot encoding to the characters at tune header, tune body as well as the <start> and <end> tags. The entire text in training data is scanned and 93 unique characters are counted. Therefore, each character in the music data is encoded as a 1x93 one-hot vector with 1 at the index corresponding to that character and 0 elsewhere.

The music data is also divided into chunks of fixed size (i.e. 100 characters per chunk). Therefore, the music data are stored as minibatches of size 1, where each minibatch contains a chunk of 100 characters being one-hot encoded. When training the recurrent network, a chunk is passed into the model as input and the same chunk (left-shifted by 1) is passed as the target. The temporal order is kept by passing the chunks in order as they appear in the data.

2.2 Model

In this work, we are going to use a particular RNN architecture, i.e. LSTM, which was first proposed by Hochreiter & Schmidhuber [1]. Like standard RNNs, LSTM consists of repeating modules, but in each module the structure is more sophisticated. Fig. 2[1] shows the structure of the repeating module.

The core idea of LSTM is the cell state, which runs straight through the entire recurrent network, allowing information flow along it unchanged. The gate is another key idea to manipulate the information flow based on previous state and current input. An LSTM module is composed of three kinds of gates, namely forget gate, write gate and read gate, corresponding to three kinds of manipulations to the information flow, namely erase, append and output.

In forget gate layer, the model decides which information to throw away. The weighted concatenation of previous output information and current input information is transformed by a sigmoid function to get a number between 0 and 1 corresponding to each neuron in the cell state, namely

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f). \tag{1}$$

Next the model decides what new information to put in the cell state. The weighted concatenation of previous information and current input transformed by a tanh function serves as the new information to be written. And an additional gate do the same thing as the forget gate, but this time is used to control which new information will be written at last. The manipulation can be expressed as

$$\begin{aligned} \widetilde{C_t} &= \tanh(W_c[h_{t-1}, x_t] + b_c), \\ i_t &= \sigma(W_i[h_{t-1}, x_t] + b_i). \end{aligned} \tag{2}$$

[1] `http://colah.github.io/posts/2015-08-Understanding-LSTMs`

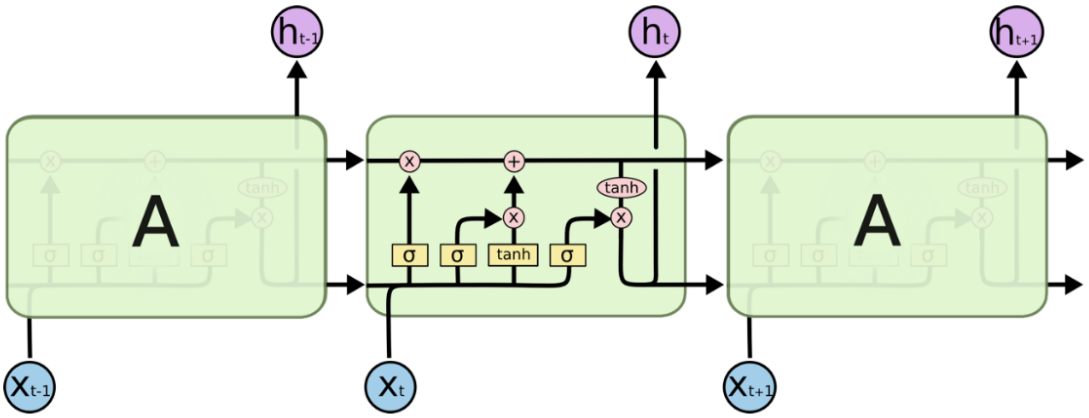


Figure 2: A repeating module in LSTM

So now the cell state is updated as

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C_t}, \tag{3}$$

where $*$ denotes element wise multiplication. Finally the model decides which information to output. As always, a sigmoid layer produces a weight matrix based on the previous and current information, and uses it to filter the output. This can be expressed as

$$\begin{aligned} o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o), \\ h_t &= o_t * \tanh(C_t). \end{aligned} \tag{4}$$

Note that the cell state is transformed by a tanh function to enforce its values between -1 and 1 before output.

There are many variant version of LSTM, including but not limit to "peephole connections"[2], Gated Recurrent Unit (GRU)[3] and Depth Gated RNNs[4]. There are also some techniques to make LSTM more sophisticated, including stacking and bi-direction. But in this work since we are dealing with a small and simple dataset, we will only use the original LSTM, and properly tune the hyperparameters to get a satisfactory performance.

3 Results

3.1 Sample Music

We use the best model that we trained to generate music. The hyperparameters of this model are as follows. We will use temparture based sampling, which randomly selects the character based on the softmax of the scaled output. The scaled factor is called temperature because it determines the magnitude of difference between the highest and lowest outputs. A high temperature therefore allows more randomness and dynamics into the decision because the probabilities associated with the characters are more close to each other.

Table 1: Hyperparameters

Hyperparameter	Value
Chunk Size	200
Number of Hidden Units	150
Learning Rate	0.01
Optimizer	Adam
Number of epochs	50

Here with the best model, we generate 6 music pieces with T=1, 2 and 0.5. The initial characters are set to be the header of an arbitrary music piece in test set. We will generate a reasonable length of characters, and pick a subset that is relatively integral and clean.

Here are 6 music pieces we generated. Two at T=1, two at T=2, and two at T=0.5.

```
<start>
X:16
T:Callatt of Boleal The
R:mazurka
H:Aire Broamand Breat : Ganellam #2
Z:id:hn-mazurka-7
M:3/4
K:G
DG|E2 BA GB|
d2 BA GB|d2 d2 dB|A2 A2 A2|
BA BA GB|d2 BA GB|A2 BA GF|
AG FD G2|AB cB AB|
d2 ed BA|
```

Figure 3: First Music Piece ABC Notation of T=1

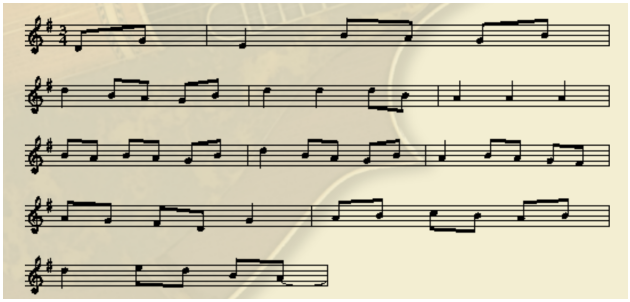


Figure 4: First Music Piece Representation of T=1

```
<start>
X:16
T:Chanter's Mazurka
R:mazurka
D:Mary Bergin: Feabonas of Satiegh the Mosssain
Z:id:hn-mazurka-1
M:3/4
L:1/8
K:Amix
d3 Bcd | e3 ecA | ABc dcd | cdc Bcd | e2c Bcd |
e2A Bcd | e2d cBA | B2B dBG | A2
```

Figure 5: Second Music Piece ABC Notation of T=1

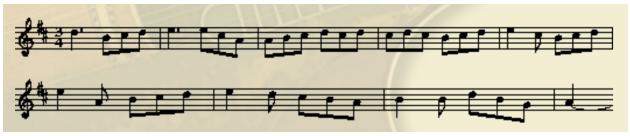


Figure 6: Second Music Piece Representation of T=1

```
<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
K:D
~A3 AGE|Oc?c c3A^C|cm"GDB BGB-|
[O>gBA GG]K,F=D({F}CF^A|BAWEc/f/ |~a>ag +b=fd|ledB "caog||
<And>A<opion pa clanal:'i Blfo(bamB,V]
|:
Imanscm'}:B[c:El]ssion) a>d'la,4 _oy
AA | Bece ag}al d'b2 a>b->a^fgd #2e'
```

Figure 7: First Music Piece ABC Notation of T=2

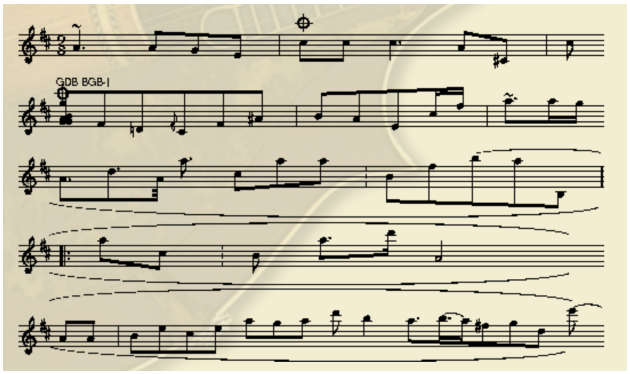


Figure 8: First Music Piece Representation of T=2

```
<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
K:D
~A3 AFd|eGE FeH|f2d6 a,fg|vbg <end>2V: Il perrio, alsA=p'sfxl\'uti
|:Grdpt=empzp+{c}A"NC"AD ED "AF"A4/ E/AG |e2g M(3fva ab | aAf7 e2 (3gf( a |
|: B2: D|E^F GBdc | ^GDG)Bcfd | df~g3|A2d2 Gagf|:
~a3g f_EF/A/B
```

Figure 9: Second Music Piece ABC Notation of T=2

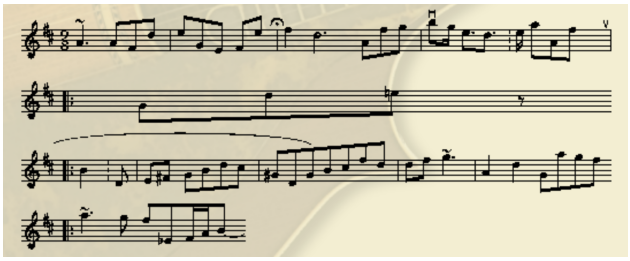


Figure 10: Second Music Piece Representation of T=2

```
<start>
X:16
T:Pillye Rackie
D:Da Danaine Collach Fidhe
Z:id:hn-mazurka-16
M:3/4
K:G
DG|B2 BA GB|d2 dB AG|F2 E2 FA|d2 df gf|
B2 B2 ed|e2 e2 d2|d2 d2 d2|d2 df d2|
B3 BA B2|d2 d2 d2|B2 d2 A2|B2 A2 A2|B2 B2 AB|d
```

Figure 11: First Music Piece ABC Notation of T=0.5

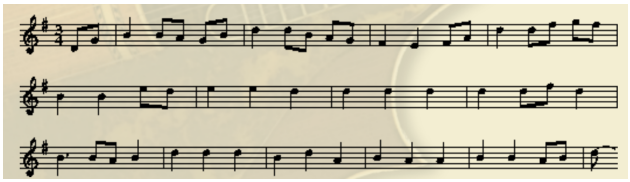


Figure 12: First Music Piece Representation of T=0.5

```
<start>
X:12
T:Lassin O'Conan's Mazurka
R:mazurka
D:Mary Bergin: The Forghe
Z:id:hn-mazurka-8
M:3/4
K:G
GB|e2 dB GA|B2 B2 AG|F2 GA BG|A2 BA GA|B2 B2 d2:|
|:fg|af df ed|c2 A2 BA|d2 d2 dc|B2 A2 GA|B2 A2 AG||
<e
```

Figure 13: Second Music Piece ABC Notation of T=0.5

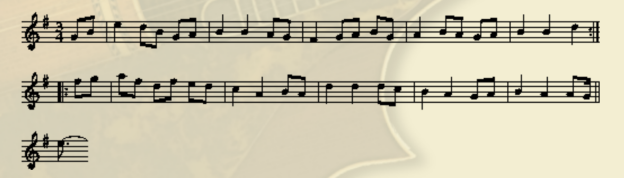


Figure 14: Second Music Piece Representation of T=0.5

If we compare these music pieces, we can find that, when T=2, every time we generate a music piece, they would be very different. However, when T=0.5, the difference would not be so obvious. As we can see, the temperature T can influence the results.

Generally speaking, if the temperature becomes larger, then the model becomes more open. If the temperature becomes smaller, then the model becomes more conservative.

3.2 Loss Curve

To get the best model, we use the validation set for tuning three hyper-parameters, i.e. number of hidden units, learning rate and chunk length. We train the model for 20 epochs using Adam optimizer. The best set of hyper-parameters for the LSTM model is that Learning rate: 0.01, Hidden units: 150, Chunk length: 200. Fig. 15 shows training loss and validation loss vs. number of epochs on training data. We find that both training loss and validation loss decrease significantly before the epoch 8, but training loss keeps decreasing while validation loss stays stable and even has a slight rebound. The gap between two curves increases which means the model goes over-fitting. We save the best weights at epoch 8 and the loss on test data is 1.63154715.

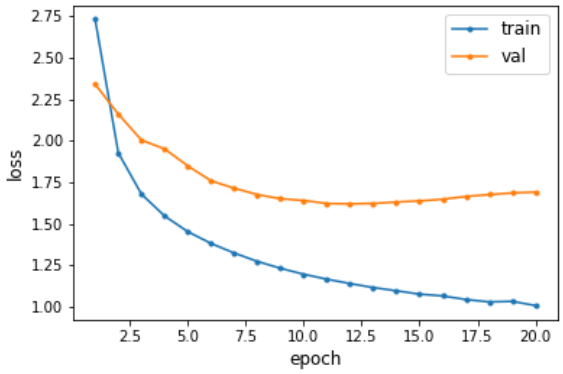


Figure 15: Training and Validation Loss vs. Number of Epochs for the Best Model

4 Experiments

4.1 Hidden Layer Size

We change the number of neurons in the hidden layer for 50, 100 and 200. The other hyper-parameters are the same as those mentioned in Section 3.2. We plot the loss curves corresponding to each model in Fig. 16.

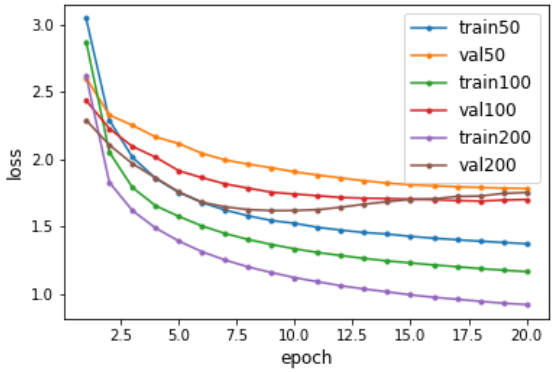


Figure 16: Training and Validation Loss vs. Number of Epochs for the Models with Different Number of Neurons

We find that the model with more neurons in the hidden layer tends to reach a lower training loss. Fig. 17 shows that the model with 200 neurons has the lowest training loss, while the model with 50 neurons has the highest training loss. It demonstrates that it is easier to fit the training data with more hidden units. However, when we pay attention to validation curves in Fig. 18, we find that the lowest validation loss does not decrease when we change the neuron number form 150 to 200. Besides, over-fitting is more obvious for the model with 200 neurons.

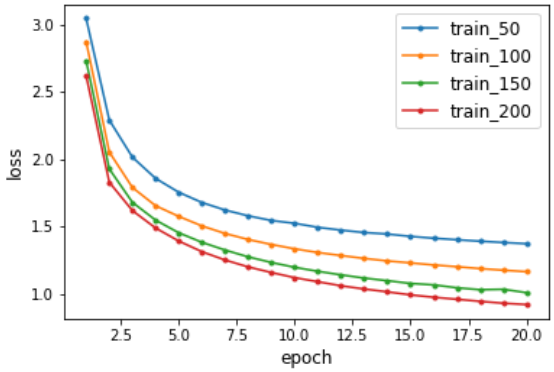


Figure 17: Training Loss vs. Number of Epochs for the Models with Different Number of Neurons

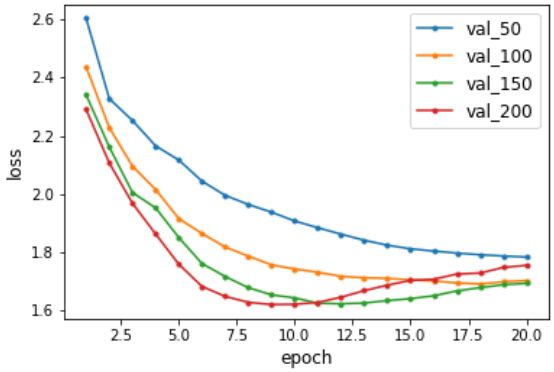


Figure 18: Validation Loss vs. Number of Epochs for the Models with Different Number of Neurons

4.2 Sampling

First, we can get the ABC notation and music representation of a music piece generated by temperature based sampling with T=0.7.

```
<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
K:D
~A3 AB/d/|e2 ef/g/|af ed/c/|d2 df|d2 dB|d2 d2|
BB Bd|B2 d2|B4 dc|d2 B2 dB|AG F2 A2|
AG FD DE|B2 BA B/A/G/G/|F2 DG|Bd cd ef|1 a2 (3fed ^cd (3efg|a4 ag|ab a2 ed|cB cA GB|
A>A BG AB|c2 Bc df|e2 d2 dc|B2 A2 AB|
```

Figure 19: ABC Notation of Music Piece Generated with Temperature Based Sampling

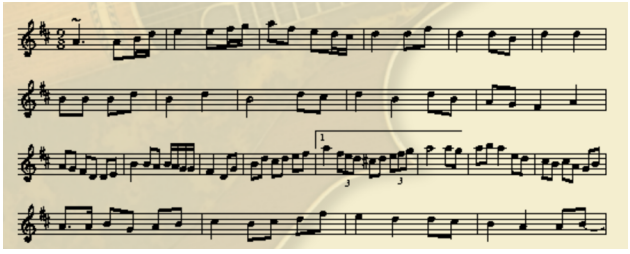


Figure 20: Music Representation of Music Piece Generated with Temperature Based Sampling

Then, we change the sampling policy. We instead use arg-max sampling, which simply picks the character corresponding to the largest probability output by softmax, rather than pick it randomly. The ABC notation and music representation is as follows:

```
<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
K:D
~A3 AB|d2 dB | B2 BA | B2 BA | B2 B2 | B2 B2 | d2 d2 |
B2 B2 | B2 B2 | B2 B2 | B2 B2 | B2 B2 | B2 B2 ||
<end>
```

Figure 21: ABC Notation of Music Piece Generated with Arg-Max Sampling

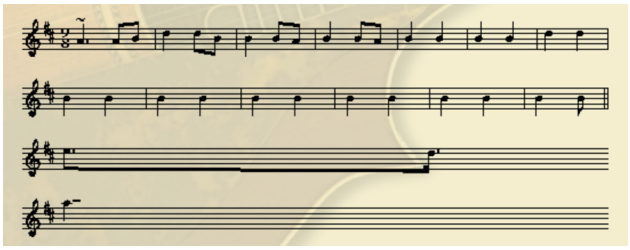


Figure 22: Music Representation of Music Piece Generated with Arg-Max Sampling

We have discussed temperature based sampling above. As we can see from both the ABC notation and music representation, if we use Arg-Max sampling, there might be "dead loops", that is, the model constantly output a same note, or a same note combinations. From the above figure, After d2, it generates B2 all the time. But the same phenomenon never happens when we use temperature based sampling.

The reason lies that, when we apply Arg-Max sampling, it is too deterministic, and can be bound by some 'easy' characters. If a piece of characters is highly correlated, and happens to have the same head and tail character, then the model will be stuck in the loop of this piece, because the probability of a subsequent character will always be the highest resulting from the same previous characters. On the contrast, if we introduce some randomness into the decision, then this trivial repetition can be avoided. Some different output other than the high correlated characters will have the chance to emerge. And once a new character is fed into the chain, the tie can be broken and the model can be escaped from a constantly same pattern. That should be something called music.

4.3 Vanilla RNN

In order to compare the performance of the LSTM model, a Vanilla RNN model is used to fit the same data. We use one hidden layer and the same number of hidden units as the LSTM model. The RNN architecture is shown in Figure 23, where x_t is the input one-hot encoded character at timestamp t, h_t is the hidden state at timestamp t, and y_t is the output at timestamp t. The calculations of the hidden state is shown in the equation 5, where h_0 is initialized as zero. The calculations of the output is shown in equation 6. Notice that W_{hh}, W_{xh} and W_{hy} are learned parameters and initialized using Xavier initialization [5].

$$h_t = f_W(h_{t-1}, x_t) = \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \tag{5}$$

$$y_t = W_{hy}h_t \tag{6}$$

The same learning rate of 0.01 and Adam optimizer are used during the training of the Vanilla RNN model. Figure

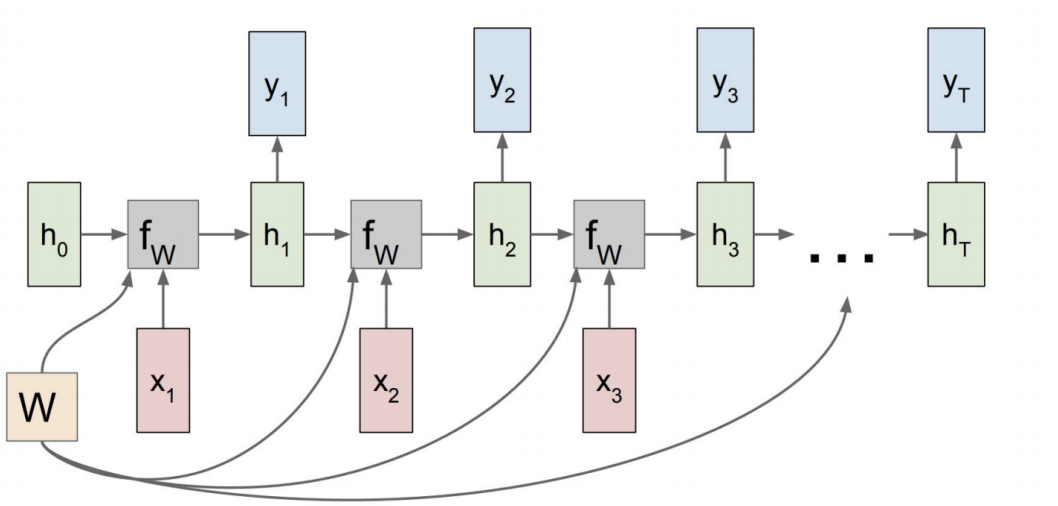
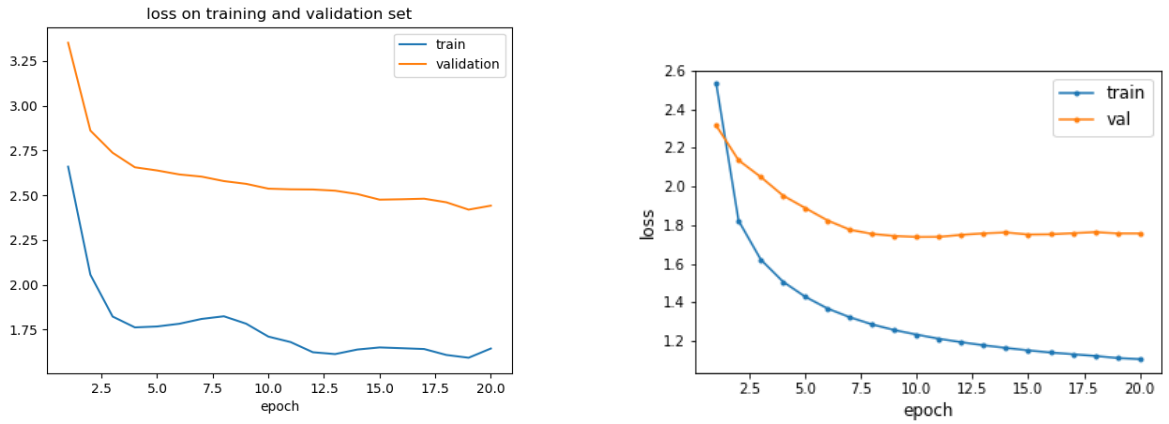


Figure 23: The architecture of the Vanilla RNN model



(a) Cross entropy loss curve for Vanilla RNN

(b) Cross entropy loss curve for LSTM

Figure 24: Cross entropy loss comparison between Vanilla RNN and LSTM

24 shows the comparison of loss curves between the Vanilla RNN model and the LSTM model. We can see that the LSTM model has a smaller cross entropy loss on the validation set. Moreover, the final cross entropy loss on the test set is 2.148 for Vanilla RNN and 1.784 for LSTM. Moreover, the Vanilla RNN model is more likely to suffer overfitting problem, as the gap between the training loss and the validation loss curves is larger. Therefore, it leads to our conclusion that the LSTM model performs much better in predicting music sequence than a Vanilla RNN model.

4.4 Feature Extraction

Here we extract some features to show how our model learns to generate music. We simply give the initial notes as '<start>', and use the best model from above to generate a length of 393 characters, such that the overall length is 400 for convenience sake. Then we feed the characters to the model and forward propagate one by one, to obtain the respective activation of each neuron.

Here we choose one neuron in the hidden state as example. As shown in Fig. 25, we reshape the characters into a 20×20 array, and plot the activation heatmap of this neuron responding to each of the character. We clearly see from '<start>' to 'K:G'[2], namely the header of the music, the neuron barely activates. But for the body, the neuron fires, especially for the notes, excluding the bar line '|', till the end of this sheet '<end>'. So this particular neuron is able to recognize the notes in the music.

Another example is for a neuron in the cell state. As shown in Fig. 26, it memorizes the start of a music sheet. Interestingly, it also fires heavily for 'Mazurka', a particular music style. After playing this generated music, we found that it is indeed close to 'Mazurka' music in terms of rhythms.

[2]i.e. a key of G major

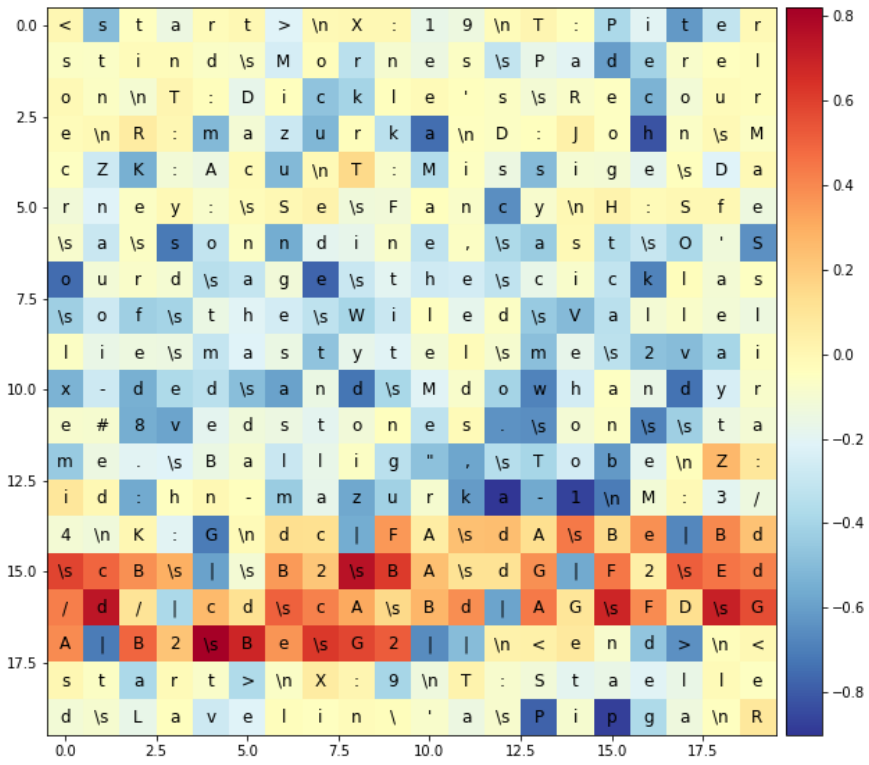


Figure 25: Activation heatmap showing this particular neuron fires for the body of the music

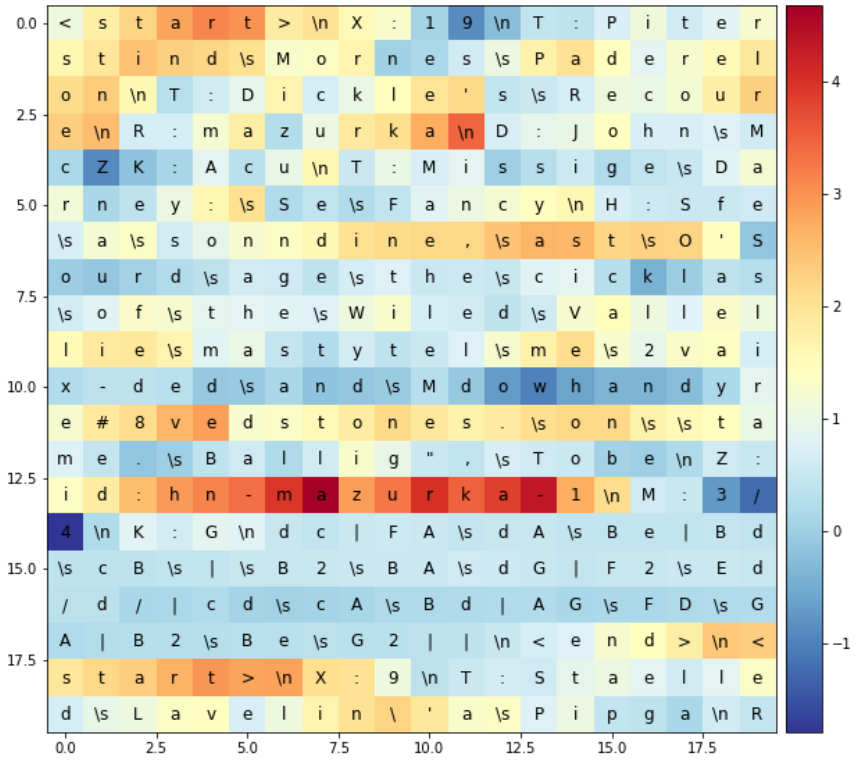


Figure 26: Activation heatmap showing this particular neuron fires when the music starts. It also favors a particular music style ‘Mazurka’, which names after a fast traditional Polish dance.

5 Contribution

Rui Yang worked on the data preprocessing step including converting text data into one-hot encoded features and putting them into the dataloader. Rui also worked on building, training and testing the Vanilla RNN model.

Weifeng Hu worked on the data preprocessing step including converting text data into one-hot encoded features and putting them into the dataloader. Weifeng also worked on building, training and testing the Vanilla RNN model.

Ye Fan worked on tuning hyper-parameters, training models, collecting and analyzing results of models with different parameters.

Shang Gao worked on generating music pieces, selecting, comparing and analyzing these music pieces, testing argmax sampling.

Chengyu Dong is responsible for the implementation of the LSTM model, as well as training and testing the model. Chengyu Dong also worked on the implementation of the music generation, and the visualization of the feature map. The corresponding parts in the report include Model and Feature extraction.

References

[1] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[2] Felix A. Gers and Juergen Schmidhuber. Recurrent nets that time and count. Technical report, 2000.

[3] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[4] Kaisheng Yao, Trevor Cohn, Katerina Vylomova, Kevin Duh, and Chris Dyer. Depth-gated LSTM. *CoRR*, abs/1508.03790, 2015.

[5] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Artificial Intelligence and Statistics (AISTATS)*, pages 249–256, 2010.