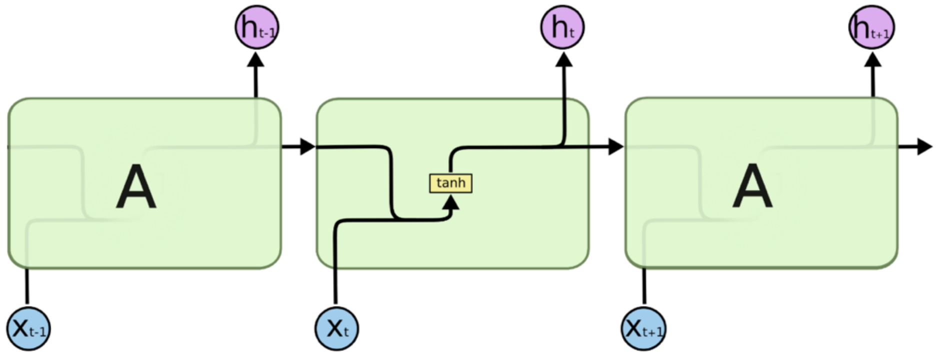
Generating Country Song Lyrics using Long Short-Term Memory Networks

1) Introduction

According to the Denver Country Bar/Nightclub “The Grizzly Rose”, “the country music fandom is getting younger and more diverse”. Indeed, the genre’s listener base between 18 and 24 years old has grown by 54% since 2006. However, Country music struggles to impose itself on streaming services like Spotify, as it “comprises only 5.6 percent of all streams” (As stated by the website Billboard). Many factors contribute to this lag, including the genre’s difficulty to generate new content at a pace that meets the consumers’ short attention spans. Therefore, this project aims to use Long short-term memory recurrent neural networks to rapidly generate lyrics for new country songs and exploit this genre’s potential. Keras “LSTM’s” have been used for text generation in texts like “Alice in Wonderland” (novel by Lewis Carroll) or Taylor Swift lyrics.

2) Review of RNN’s and LSTM’s

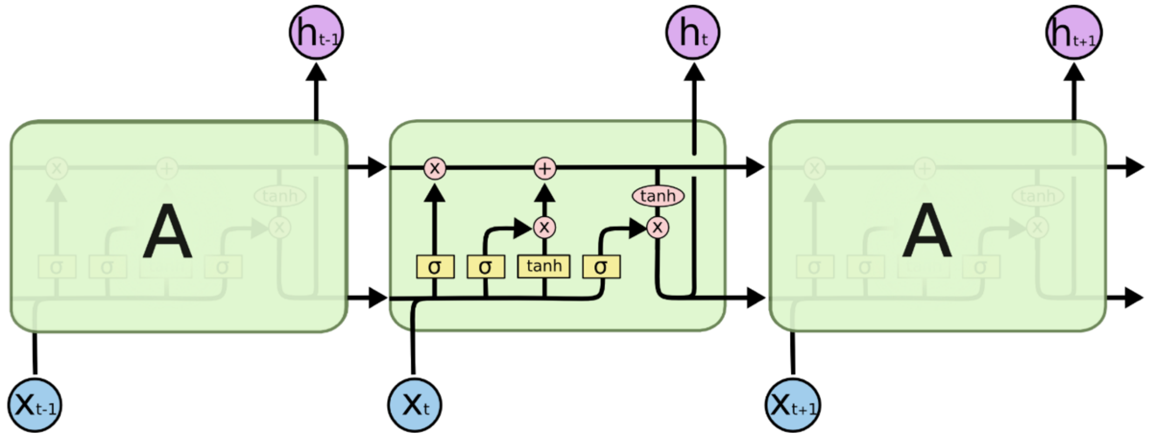
This project used Long Short-Term Memory Networks, which are a type of Recurrent Neural Network. Recurrent Neural Networks are specialized in processing a sequence of values $x^{(1)},\ldots, x^{(T)}$. As opposed to a traditional Feed-forward Neural Network which would have separate parameters for each element of the sequence, a Recurrent Neural Network would use the same weights across the different time steps. Each output is a function of the other outputs in the same sequence. The RNN can be decomposed into chunks “A_t”, which processes each input x_t and produces output h_t. However, each output h_t is not only a function of x_t but also of the previous output h_{t-1}, as shown in Figure 1. Both of these processes are combined with the use of pointwise operations between h_{t-1} and C_t (or the result of x_t going through an activation function such as tanh). This allows the Recurrent model to maintain the sequential logic of the input data.



The repeating module in a standard RNN contains a single layer.

Figure 1: Chunks A_{t-1}, A_t and A_{t+1} of a standard RNN

Following this model, we can easily see that Classic RNN's store a lot of information without much filtering. These simplistic models often run into a "long-term dependency problem". According to this theory, the distance between the predicted output and the relevant input is inversely related with the accuracy of the prediction. For instance, in the sentence *"I grew up in France,…….. I speak French fluently"*, the underlined word is the predicted output. To accurately predict the language, we need the word "France". If the gap between the two was large, the resulting gradient would either be excessively large ("exploding"), or small ("vanishing"). To overcome that problem would either take colossal computing time or a new RNN architecture. The Long-Short Term Memory network, shown in Figure 2 has four layers per chunk instead of 1.



The repeating module in an LSTM contains four interacting layers.

Figure 2: Chunks A_{t-1}, A_t and A_{t+1} of a LSTM

The Cell State acts as a "conveyor belt" to which information is added through "gates". The gates are each composed of activation functions such as tanh or sigmoid, as well as pointwise operations and regulate the flow of information of the cell state.

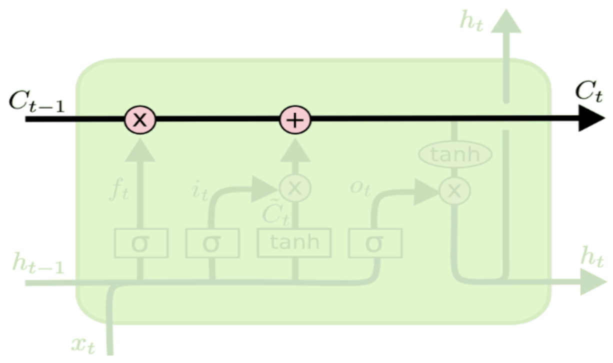
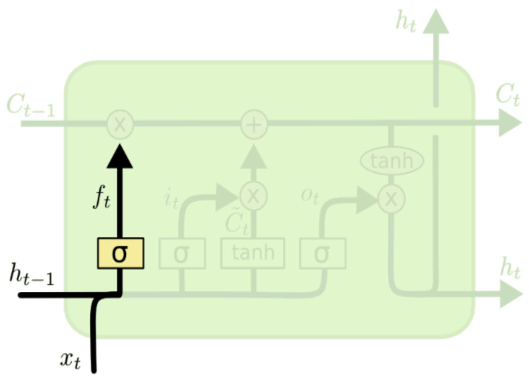


Figure 3: The Cell State in a LSTM

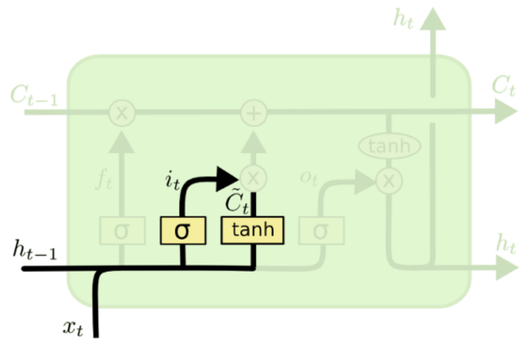
The "forget gate" decides how much information is kept from the previous cell state. The sigmoid function takes "x_t" and "h_{t-1}" as inputs, passes them through a set of Weights W_f, and outputs a number "f_t" between 0 (completely get rid) and 1 (completely keep). This gate serves allows the model to discard long-term dependencies.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure 4: The forget gate in a LSTM

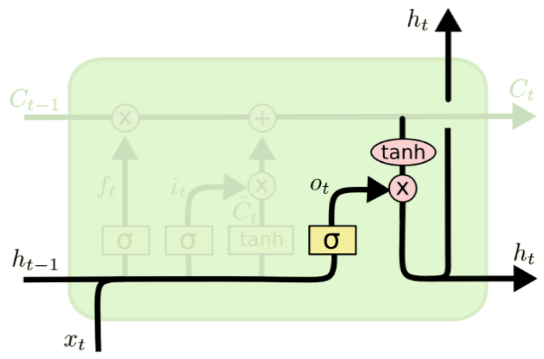
Furthermore, the input gate decides which values are updated (i_t) and creates a vector of new candidate values ($C^{\sim}_t$). The computation of it involves a sigmoid function and weights Wi while that of Ct uses a tanh function and weights W_c.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figure 5: The input gate in a LSTM

Lastly, the output gate functions like the forget gate, by filtering which parts cell state will be output (o_t) using weights W_o and a sigmoid function, then combined with an altered version of the Cell State ($\tanh(C_t)$) using pointwise multiplication (resulting in h_t).

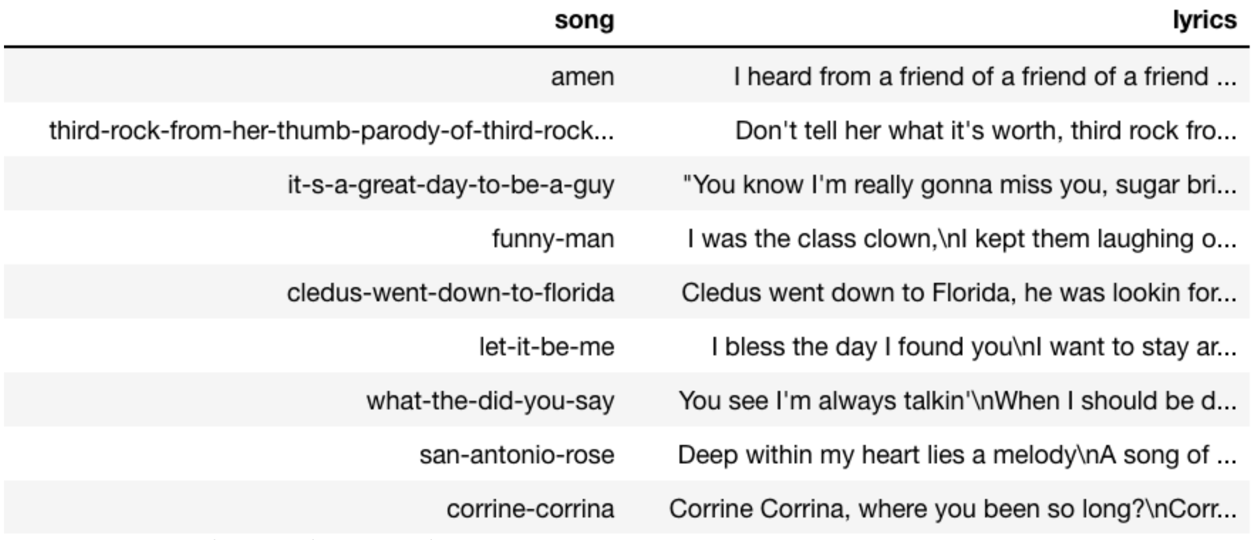


$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 6: The output gate in a LSTM

3) Overview and Processing of the Data Set

The Data Set was taken from a CSV file called “380,000+ lyrics from MetroLyrics” which was found on Kaggle. As shown in Figure 7, each row corresponded to a song classified by the “index”, “song name”, “year”, “artist”, “genre” and “lyrics” columns. It was then downloaded as a Pandas data frame, cleared of all NaN’s. Furthermore, only the rows whose genre was “country” were selected. The “index”, “year”, “artist”, “genre” columns were deemed irrelevant for this purpose and were therefore deleted. The resulting data frame had 2 columns (“song name” and “lyrics”) and 14 387 rows (or songs). The lyrics were then converted to a lower text .txt file.



	song	lyrics
	amen	I heard from a friend of a friend of a friend ...
	third-rock-from-her-thumb-parody-of-third-rock...	Don't tell her what it's worth, third rock fro...
	it-s-a-great-day-to-be-a-guy	"You know I'm really gonna miss you, sugar bri...
	funny-man	I was the class clown,\nI kept them laughing o...
	cledus-went-down-to-florida	Cledus went down to Florida, he was lookin for...
	let-it-be-me	I bless the day I found you\nI want to stay ar...
	what-the-did-you-say	You see I'm always talkin'\nWhen I should be d...
	san-antonio-rose	Deep within my heart lies a melody\nA song of ...
	corrine-corrina	Corrine Corrina, where you been so long?\nCorr...

Figure 7: Data frame after initial pre-processing

Since computing time would be too long for the whole data set (as will be further explored in section 4), N different songs were randomly selected from the data frame. The total number of characters "N_total" was counted, as well as the number "Q" of different characters. Each character was then mapped to an integer using a Python dictionary. As N increases, the number of different characters also increases. The resulting data was then split into an input X and output y.

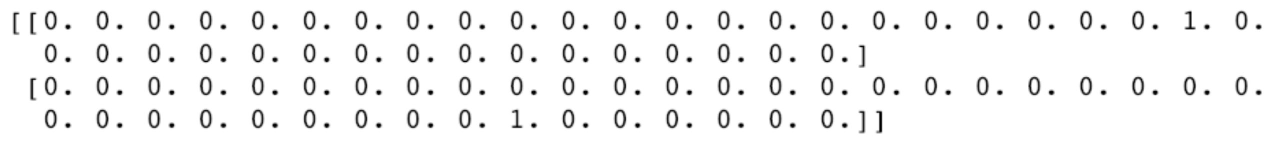
X was therefore any sequence of 100 characters in the data set and the corresponding output y was the following 1 character. Each character was then mapped to an integer using a python dictionary. (Another dictionary was created to map the integers back to their corresponding characters for printing the outputs). There were Q different possibilities for y and N_total training points. The y vector was then hot-encoded. For instance, the first two elements of the y vector (for training) are represented in their integer (a) and hot-encoded (b) form in figure 9. The input X is therefore a N_total x 100 matrix and the output y is now a N_total x Q matrix. The data was then split into training (80%) and testing (20%) sets.

```
['\n' ' ' '!' '"' "'" '(' ')' ',' '-' '.' ':' '?' '[' ']' 'a' 'b' 'c' 'd'
 'e' 'f' 'g' 'h' 'i' 'j' 'k' 'l' 'm' 'n' 'o' 'p' 'q' 'r' 's' 't' 'u' 'v'
 'w' 'x' 'y' 'z']
```

Figure 8: Q different characters in a 10-song dataset (Q=40)

```
[22, 33]
```

Figure 9 (a) integer form of y[:2]



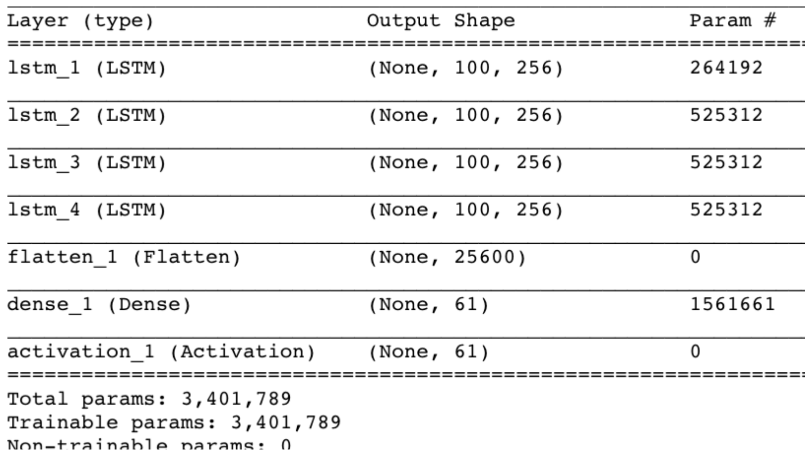
```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]]
```

Figure 9 (b) hot-encoded form of y[:2]

4) Training with different amounts of data and number of epochs

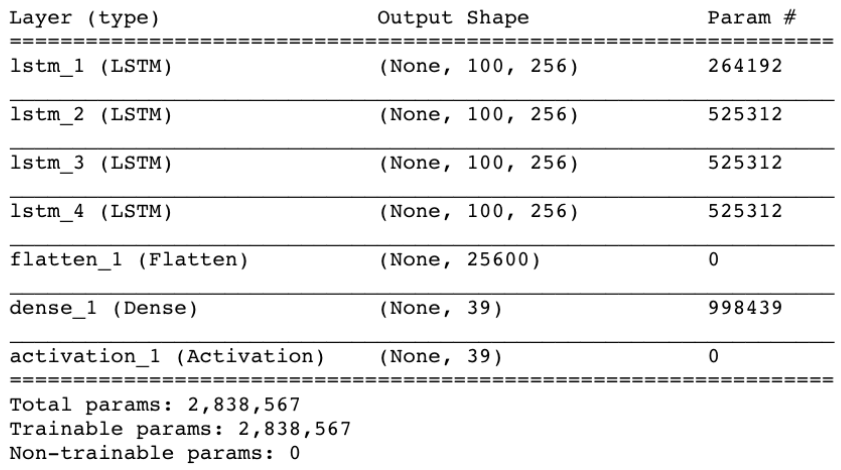
The LSTM was built using the Keras Sequential Model and trained using the Keras fit function. The LSTM had 4 layers with 256 nodes in each. For the first layer, the input shape was specified as a 100 x 1 argument. 3 hidden layers were added and the data was then flattened before being passed through a Softmax function. The cross-entropy loss was then calculated and passed through the Keras adam optimizer (with default parameters: $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 10^{-8}$).

Since processing the whole dataset (14 387 songs, N_total = 13 250 494, Q=115 different characters, 4 784 243 parameters) would take too much time (approximately 50 hours per epoch), 2 different data sets were created: one with 100 songs and another with 10 songs. For 100 songs (N_total = 87 978, Q = 61, 3 401 789 parameters) each epoch would take approximately 16 minutes. For 10 songs (N_total = 10 106, Q = 39, 2 838 567 parameters), each epoch would take approximately 2-3 minutes. Figures 10 (a) and 10 (b) show the summaries of the LSTMs for 100 and 10 songs. Since training time for the 10-song dataset is much shorter than that of 100-song dataset, we were able to train the 10-song data set for 10, 30 and 60 epochs versus 10 epochs only for the 100-song dataset.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 100, 256)          264192
_________________________________________________________________
lstm_2 (LSTM)                (None, 100, 256)          525312
_________________________________________________________________
lstm_3 (LSTM)                (None, 100, 256)          525312
_________________________________________________________________
lstm_4 (LSTM)                (None, 100, 256)          525312
_________________________________________________________________
flatten_1 (Flatten)          (None, 25600)             0
_________________________________________________________________
dense_1 (Dense)              (None, 61)                1561661
_________________________________________________________________
activation_1 (Activation)    (None, 61)                0
=================================================================
Total params: 3,401,789
Trainable params: 3,401,789
Non-trainable params: 0
```

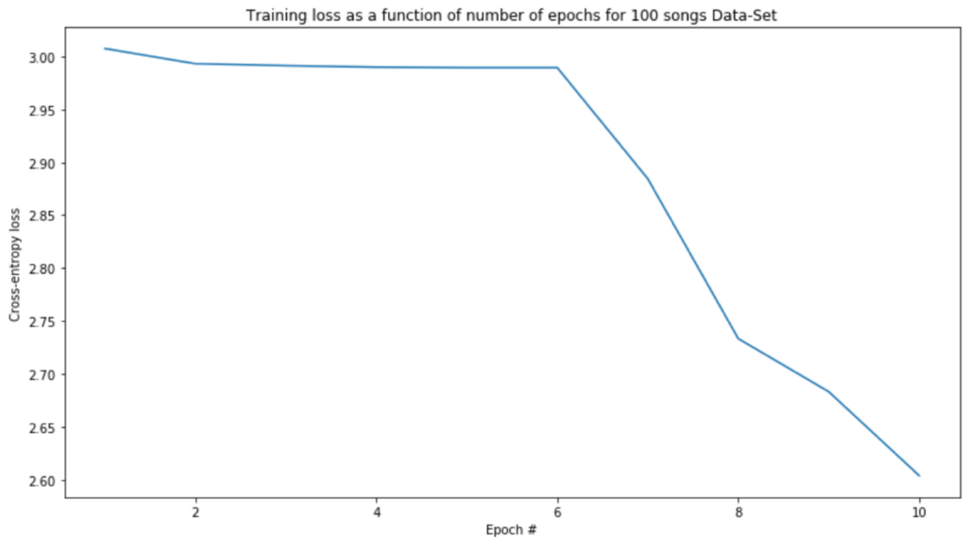
Figure 10 (a): Summary of the LSTM for 100 songs



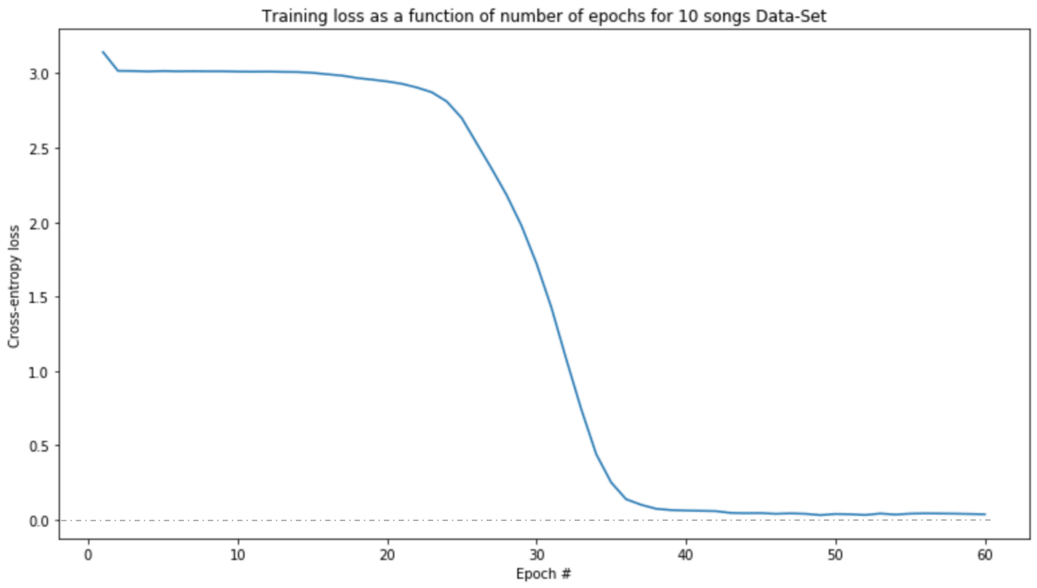
```
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 100, 256)          264192
_________________________________________________________________
lstm_2 (LSTM)                (None, 100, 256)          525312
_________________________________________________________________
lstm_3 (LSTM)                (None, 100, 256)          525312
_________________________________________________________________
lstm_4 (LSTM)                (None, 100, 256)          525312
_________________________________________________________________
flatten_1 (Flatten)          (None, 25600)             0
_________________________________________________________________
dense_1 (Dense)              (None, 39)                998439
_________________________________________________________________
activation_1 (Activation)    (None, 39)                0
=================================================================
Total params: 2,838,567
Trainable params: 2,838,567
Non-trainable params: 0
```

Figure 10 (b): Summary of the LSTM for 100 songs

We then compared the cross-entropy loss for the 100-song dataset (10 epochs) with that of the 10-song dataset (60 epochs). Overall, we can see that the smaller dataset was better at minimizing loss, lowering the loss by 3 points versus the larger dataset, which only lost 0.4 points. Both of the loss functions dropped after approximately half of the total epochs.



We can see that the loss drops from 3 to 2.75 between 6 and 8 epochs. This maximum loss has a slope of -0.125.



We can see that the loss drops from 3 to 0.0 between 25 and 35 epochs. This maximum loss has a slope of -0.3.

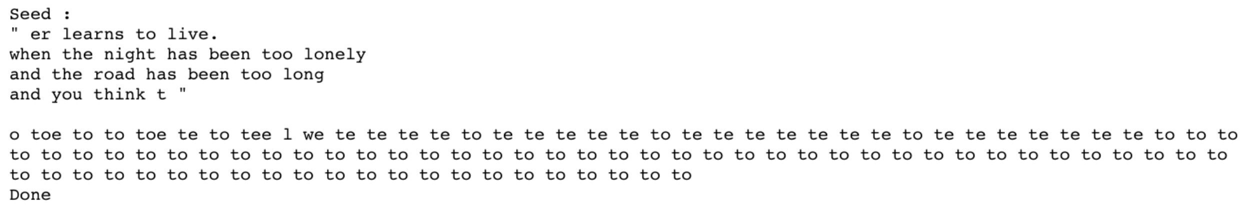
5) Testing with 100-song data-set vs 10-song data-set

For testing, we randomly selected 100 consecutive characters in our dataset as our initial input X_0 (The characters that come after "Seed:" in figure _). We then predicted the following character y_0 using the Keras "predict" function. As a result, we had 101 total characters. We then took the 2nd to 101st characters as our new input X_1. We then predicted the 102nd character and updated our dataset until n total characters were generated (achieved using a for loop for n_iterations, here n_iterations = 300).

Since we could only train our 100-song data set for 10 epochs, there was not much variation in our predicted output, as shown in figure 11. We can note that the only characters appearing are spaces " " and letters "t", "e" and "o". Words were at most 3 characters long in each case.

```
Seed :
"  you ain't woman enough to take my man
well women like you...
no you ain't woman enough to take my m "

e tee to tee to toe to to te te te te we te to te te te te te te to te te te te te te te te to te te te te te te to t
o to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to t
o to to to to to to to to to to to to to to to to to to to to t
Done
```



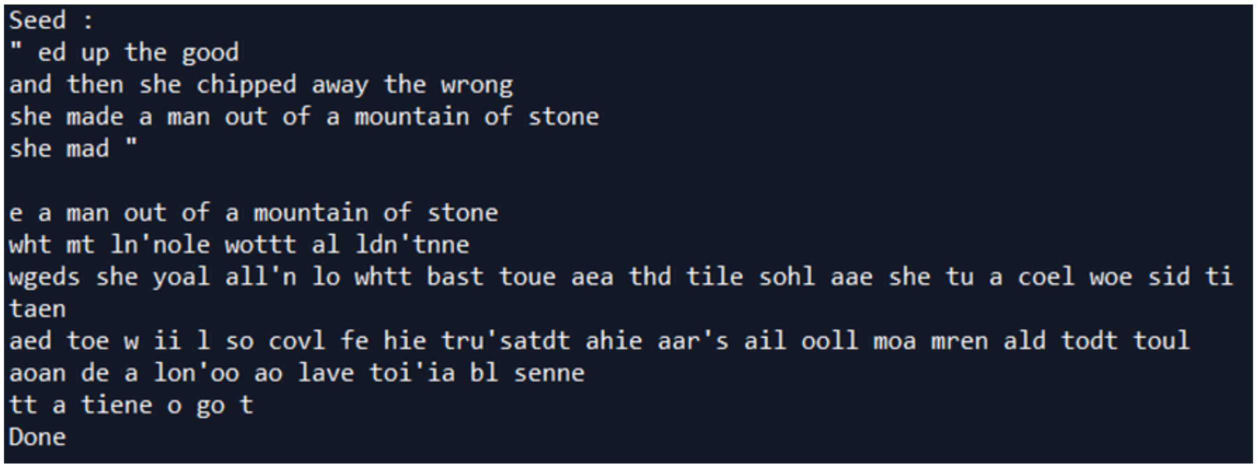
```
Seed :
" er learns to live.
when the night has been too lonely
and the road has been too long
and you think t "

o toe to to toe te to tee l we te te te te to te te te te te to te te te te te te te to te te te te te te te to to to
to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to
to to to to to to to to to to to to to to to to to to to to to to
Done
```

Figure 11: 2 different predicted outputs for 100-song data set (10 epochs)

For the 10-song dataset, the results were more varied according to the number of epochs. For instance, with 10 epochs, the model only predicted spaces " ", which makes sense since spaces are the most frequently appearing character in any text. Similarly to the 100-song 10 epoch trial, the model only seems to predict characters based on how often they appear in the dataset (their proportion to N_total), not based on the preceding characters in the input. Therefore, 10 epochs are not enough to predict full words, let alone sentences that make logical sense or follow any rhyme scheme.

Increasing the number of epochs to 30 seemed more promising, as shown in figure 12. The first predicted verse made logical sense but was merely a repetition of the previous verse. One can assume that it was able to predict this exactly because the input X_0 ended mid-verse (even mid-word). However, once it started to predict a new verse (after the character "\n"), the words stopped making sense. One can note that for 30 epochs, there seems to be a "structure" to these verses that was not present for the 10 epochs case.



```
Seed :
" ed up the good
and then she chipped away the wrong
she made a man out of a mountain of stone
she mad "

e a man out of a mountain of stone
wht mt ln'nole wottt al ldn'tnne
wgeds she yoal all'n lo whtt bast toue aea thd tile sohl aae she tu a coel woe sid ti
taen
aed toe w ii l so covl fe hie tru'satdt ahie aar's ail ooll moa mren ald todt toul
aoan de a lon'oo ao lave toi'ia bl senne
tt a tiene o go t
Done
```

Figure 12: predicted output for 10-song data set (30 epochs)

Increasing the number of epochs to 60 showed great results. All sentences made sense and were structured in verses. Common themes of country music (https://www.westword.com/music/the-ten-biggest-tropes-in-country-music-

5694846) were even respected with expressions like “soul”, “angel”, “stained glass”. However, one can see that both predictions contain identical phrases. Since the dataset is only 10 songs and the testing set represents 20%, we only have 2 different songs to test on.

```
Seed :
"  stained glass sunday morning
baby you light it up
shot some faith on my darkest places
like a risin "

g sun
filled my heart with a million amazing graces
your voice is like a choir
a match that starts a fire
deep down in my bones where only you can go
when my soul needs an angel's touch
you light it up
thare you are like a stained glass sunday morning
you bid the dark
with all your truth and all you
Done
```

```
Seed :
" s low beams, long stretch of highway
when the lonesome hits like a tidal wave
tossing and turning, t "

angled with my demons
making me doubt everything i believe in
there you are
like a stained glass sunday morning
baby you light it up
shot some faith on my darkest places
like a rising sun
filled my heart with a million amazing graces
your voice is like a choir
a match that starts a fire
deep down in
Done
```

Figure 13: 2 different predicted outputs for 10-song data set (60 epochs)

6) Conclusion

We can see that LSTM’s are great for sequential data. Many variables matter in determining the computational time vs. accuracy of a particular model, including batch size (batch size ~16 would maybe give a better chance of escaping a local minima for the loss function), number of layers, dimension of each layer, presence of a “dropout” etc. Due to training this model on a CPU and not a GPU, we narrowed our experiments to varying data size and number of epochs. Considering our results, we can conclude that a higher number of epochs is more important for a network to recognize and reproduce sequential patterns such as Country music lyrics.

References

- *https://www.westword.com/music/the-ten-biggest-tropes-in-country-music-5694846*
- *https://www.billboard.com/articles/business/7941122/country-music-streaming-digitalera*
- *https://towardsdatascience.com/ai-generates-taylor-swifts-song-lyrics-6fd92a03ef7e*
- *http://colah.github.io/posts/2015-08-Understanding-LSTMs/*