Lyric Bot - Stylized Lyric Generation

Ben Crystal & Alice Murphy

May 26, 2022

Contents

1 Introduction

As musicians, many artists inspire us to create songs in similar styles. It is relatively easy to loosely imitate the style of a pianist, a guitar player, or a beat-maker, but it can be very difficult and time consuming to emulate a lyricist. To come up with wordings in a similar style to an artist requires not only an understanding of the lyrics across their discographies, but also an understanding of what they are thinking and how they came up with the words that they have incorporated in many of their songs. Additionally, it can be very difficult to create lyrics for songs in general, especially given a small time frame. This project aimed to create a deep learning architecture that learn an artist's lyrical style and generate new lyrics based that artist.

The motivation for this project is to develop a neural network that will learn an artist's style from their work and have the ability to reproduce a new set of lyrics for the user. This project has many possible applications. Musicians could create new songs in the style of musicians who are deceased to pay tribute to them. They can train on multiple artists to blend their styles and create an entirely new approach to lyricism. People can also use newly created lyrics in their favorite artists' styles as inspiration for their own vocals, as well as as unique fillers when recording tracks as a placeholder to build other aspects of a song around. This can also be used to generate lyrics in a short amount of time, which is a situation that arises when artists need "filler" lyrics when composing the rest of a track. The range of creative uses of these lyrics is limitless. A potential commercial application of this process could be the implementation of an app that locally generates lyrics to new songs either based on user preferences on a streaming service or on text notes uploaded by the user themselves.

The team hypothesized that a computationally efficient network consisting of LSTM and fully connected nodes could effectively produce new song lyrics based on the style of an artist's existing corpus of lyrics. The proposed architecture proved to be unsuccessful in this pursuit, often generating repetitive, existing, or random lyrics, as will be discussed in the "Results" section. The team considers several next-steps to improve the network design for semantic, stylized text, which can be found in the "Discussion" section.

2 Problem Definition and Algorithm

2.1 Task Definition

The algorithm is tasked with tailoring itself to individual artists' writing styles in order to producing new content while mimicking lyrics based on the artist's existing songs. It must be able to take the input of a seed, or user-defined starting characters, and output a continuation of lyrics for a user-specified duration. Lyric generation networks are relatively new, but allowing users to customize the starting words and the artist's style is quite novel. This approach would assist in both creating tribute lyrics, for celebrating artists that no longer produce music, and inspiring an individual's attempt to write a song in an certain style, for new musicians who seek to imitate their idol's expression. This algorithm may also be helpful to lyricists that need to push through writer's block, which is the situation when creative people become stumped with what should come next in their art. This type of block cannot happen to this generative algorithm, so it provide the type of inspiration that a human collaborator would. Generative algorithms have the potential to revolutionize the creative process and have already begun to both compete with and assist human artists.

2.2 Algorithm Definition

The algorithm selected for the purpose of style-based lyric generation was a very small character-based approach. In particular, LSTM and bidirectional LSTM layers were chosen to "learn" how to generate text. These layers are accepted by the machine learning community as a better approach to lyric generation than RNNs, as the memory component allows them to consider information from "further back", generating a potentially more likely character. The bidirectional LSTM enables the algorithm to look both forward and backward at the training text, to better conceptualize the context of each character in terms of previous and future characters.

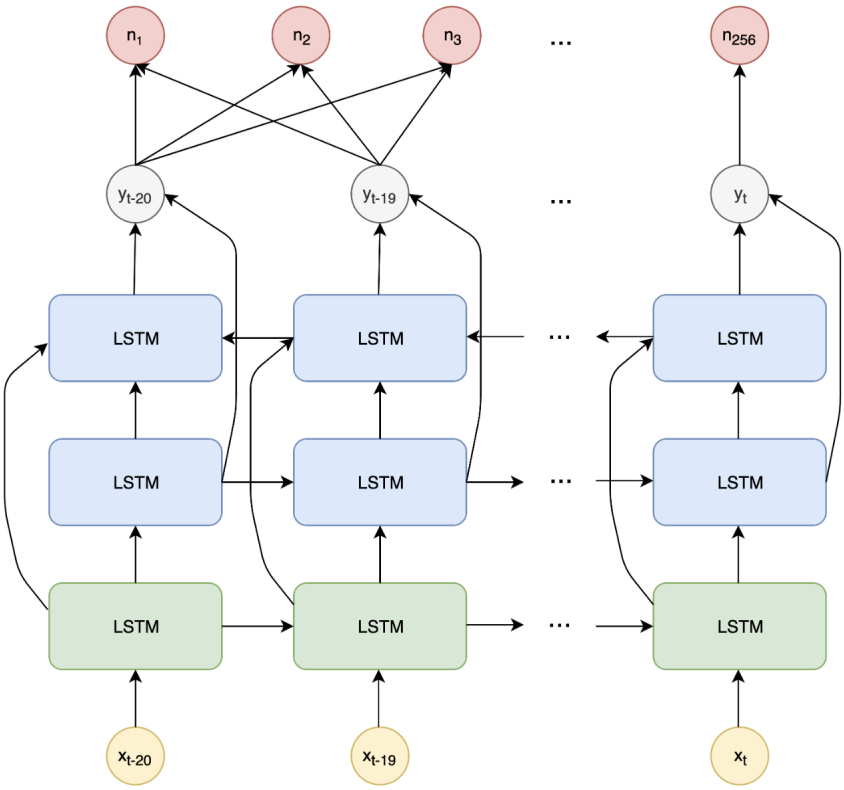

Figure 1: This figure shows the architecture used in this project. The yellow nodes represent the input sequence of 20 characters, the green LSTM blocks represent the first LSTM layer, the blue LSTM blocks are all part of the bidirectional LSTM layer, the gray nodes represent the outputs from the bidirectional LSTM layer, and the red nodes correspond to the fully connected output layer. Note, this fully connected output layer has 256 nodes, which can act as a one hot encoding for any ASCII character.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_126 (LSTM)              (None, 100, 256)          264192
_________________________________________________________________
dropout_126 (Dropout)        (None, 100, 256)          0
_________________________________________________________________
bidirectional_63 (Bidirectio (None, 512)               1050624
_________________________________________________________________
dropout_127 (Dropout)        (None, 512)               0
_________________________________________________________________
dense_63 (Dense)             (None, 256)               131328
=================================================================
Total params: 1,446,144
Trainable params: 1,446,144
Non-trainable params: 0
_________________________________________________________________
```

Figure 2: This figure is a summary of the architecture that trained its weights from scratch on the dataset. The model totals 1,446,144 trainable weights.

3 Experimental Evaluation

3.1 Methodology

This project involved gathering a dataset of lyrics, grouped by artist. The website Genius.com[1] acted as a valuable resource. This hub of lyrics benefits data scientists and music fanatics alike, providing accurate data with thorough notation. Due to the many applications of Genius.com's artist, song, and lyric data, a powerful python module was created to interface with the site. This package, lyricsgenius[2], helped gather the lyrics from artists' most popular songs. For each artist, the words and notations from their top 40 songs, as shown in Figure 3. These characters that make up the lyrics to these songs were then joined into one long string per artist. These strings were filtered to remove most of the musical notation or punctuation that would pose a challenge to the generative model. As seen in Figure 4, the text contained labels in brackets, that indicated the section of the song, like the chorus, the verses, and the bridge. These labels are helpful in sorting the song, but would be learned as parts of the lyrics if fed through the model. The raw text also contained apostrophes written as the string literal: backslash-apostrophe. The backslashes and apostrophes were removed to ensure that these were not viewed as valuable parts of words and that conjunctions (like "can't", "wasn't", and "don't") were not viewed as two words, separated by punctuation. After the cleaning of the strings, the numbers from the sections were retained, as seen in Figure 5. Characters were converted to ASCII numbers to be interpreted and predicted by the algorithm, in an effort to produce language from calculations.

```
artist = "Mac Miller"
max_songs = 40
genius = lyricsgenius.Genius("aQ7lefZGur5adkObCa4UZE9olb-x-xxXeyxHPyyL8k0r6jejtCZsAytNZ4ww67Od")
artist = genius.search_artist(artist, max_songs=max_songs, sort="popularity")
songs=artist.songs
```

```
Searching for songs by Mac Miller...

Song 1: "Self Care"
Song 2: "Weekend"
Song 3: "Objects In The Mirror"
Song 4: "2009"
Song 5: "Cinderella"
Song 6: "Donald Trump"
Song 7: "Loud"
Song 8: "Diablo"
Song 9: "My Favorite Part"
Song 10: "Best Day Ever"
```

Figure 3: The lyricsgenius package accesses song lyrics based on artist and enables the user to obtain the most popular tracks.

```
gave his seat to him  [Verse 31] He saw the starlight shining Streaming from the East Deat
h was on the rampage But his heart was now at peace  [Verse 32] They battened down the hat
ches But the hatches wouldn\'t hold They drowned upon the staircase Of brass and polished
gold  [Verse 33] Leo said to Cleo I think I\'m going mad But he\'d lost his mind already W
```

Figure 4: The raw lyric strings contained sections labels, brackets, backslashes, and apostrophes.

```
eat to him  31 He saw the starlight shining Streaming from the East Death was on the rampage
But his heart was now at peace  32 They battened down the hatches But the hatches wouldnt hol
d They drowned upon the staircase Of brass and polished gold  33 Leo said to Cleo I think Im
going mad But hed lost his mind already Whatever mind he had  34 He tried to block the doorwa
```

Figure 5: After filtering, the artist strings contained only letters, spaces, and numbers. Note, this text still poses problems, due to the repetition of spaces and meaningless numbers.

The lyrics were fed into the algorithm as normalized versions of ASCII integers. In other words, each character in each artist's string was converted to its ASCII code, an integer between 1 and 256. Then, these numbers were divided by 256 to convert them from integers to floats between zero and one, as shown in Figure 6. The model trained on sequences of length 20, meaning that the training data inputs were strings of 20 characters, with corresponding outputs of the one character following the sequence, as shown in Figure 7. This method allows users to input a string of 20 character and loop the prediction algorithm to output as many characters as they want.

```
Seed as floats between 0 and 1:
" [0.453125, 0.40625, 0.41015625, 0.44921875, 0.125, 0.41015625, 0.44921
875, 0.125, 0.453125, 0.40625, 0.39453125, 0.125, 0.39453125, 0.46875,
0.37890625, 0.42578125, 0.4375, 0.421875, 0.39453125, 0.125] "
Seed as characters:
" this is the example  "
```

Figure 6: After filtering, the artist strings contained only letters, spaces, and numbers. Note, this text still poses problems, due to the repetition of spaces and meaningless numbers.

```
I Made a promise to  -> m
 Made a promise to m -> y
Made a promise to my ->
ade a promise to my  -> m
```

Figure 7: The algorithm trains itself on strings of length 20 and outputs of length 1 that correspond to the next character in the lyrics.

In both human created and computer generated art, effective quantitative criteria to evaluate creative pursuits has yet to be found. For this project, the overall success is based on qualitative human satisfaction. Since songwriting, the human equivalent to this algorithm, is most often valued based on listener or reader enjoyment, using this measure is fitting. However, the algorithm must be able to optimize a quantitative function, without requiring constant human feedback when training. The algorithm will seek to minimize loss and, during validation, it calculates the accuracy of prediction. The loss function and accuracy metric used for optimization are *sparse categorical cross entropy* and *sparse categorical accuracy*, respectively. Categorical cross entropy is a performance evaluating parameter used in multiclass categorization. While categorical cross entropy works with one-hot encoded information, sparse categorical cross entropy takes integer inputs and assumes equal distance between each of the values. This function was chosen in an effort to minimize the

memory needed to run the model. It effectively replaced one-hot vectors of length 256 with single integers. The metric trained the algorithm's ability to mimic character sequencing in lyrics, by equally penalizing all incorrect predictions.

3.2 Results

3.3 Mac Miller: No Output

The network was trained on the lyrics of late rapper Mac Miller's top 40 songs. The network training ran for 20 epochs, which totalled roughly 12 hours of training. The network updated weights based on batch sizes of 512 and the Adam optimizer at a learning rate of 0.001. The network weights were saved every 4 epochs, so, over the 24 epochs of training, 6 network weight files were saved. Using the final weightings, the network generated 150 new characters based on the seed: "When I was young and". The resulting output can be seen in Figure 8. The algorithm failed to predict ASCII values and it was later discovered that the weights had reached "NaN" values even before the 4th epoch (the first saved set of weights). This discovery indicated the issue of exploding gradients.

```
[nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
 nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
 nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
 nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
 nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
 nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
 nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
 nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
 nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
 nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
 nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
 nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
 nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
 nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
 nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
 nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
 nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
 nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
 nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
 nan, nan, nan, nan, nan, nan, nan, nan, nan]], dtype=float32)
```

Figure 8: The network weights reached unreal values and, thus, the outputs were not characters.

3.4 Bob Dylan: Overfit or Nearly Random

To remedy the exploding gradient problem, gradient clipping was implemented. This method sets a maximum and minimum gradient value to ensure the values do not go outside of a useful range. For this experiment, the Adam optimizer with a learning rate of 0.001 was used again, but the gradients were constrained to ± 0.5. The model sought to optimize its weights to maximize accuracy, shown in Figure 9, and minimize loss, shown in Figure 10. The model trained on Bob Dylan's top 40 songs for 20 epochs, taking about 3 hours in total. The model weights were saved after every epoch.

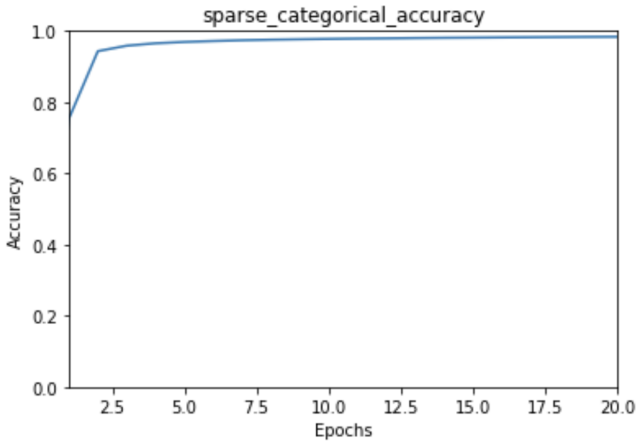

Figure 9: The sparse categorical accuracy reached a peak value of 0.98.

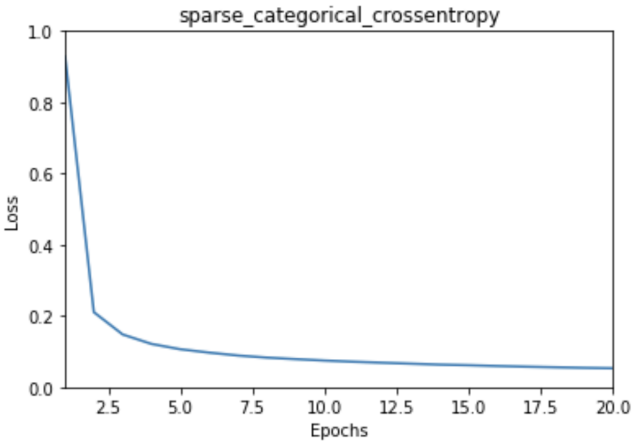

Figure 10: The sparse categorical cross entropy reached a minimum of 0.8.

The generating code can take in any file of the network weights, any ASCII character seed of length 20, and will output new text of any length. In the example shown in Figure 11, the seed "when i look up i see" was fed into the network with the weights from epoch 1, 5, 10, and 20. The outputs after epoch 1 and 5 both replicated existing Bob Dylan songs (*Tempest* and *It's Alright, Ma*). After epochs 10 and 20, the model began predicting new sequences of characters, but these patterns had no meaning to human users. Interestingly, when the user set the model to predict a long steam of generated text, the model seemed to produce some unique and some replicate lyrics. As shown in Figure 12, the seed "its a beautiful spri" led to mostly random outputs, with part of *Tempest* showing up again in a short burst. The model drifted between reproducing existing lyrics and producing indiscriminate strings.

Epoch 1

" **when i look up i see**ee These bnse bl ooe eoe tott ahye ohe the alld 33 Tee watchman he lay dreaming Of all the things that can be He dreamed the Titanic was sinking Int " (Tempest)

Epoch 5

" **when i look up i see**ee A lnwwer souonng to hear Alone you stand, with nobody near When a trembling distant voice, unclear Startles your sleeping ears to hear That somebo " [It's Alright, Ma (I'm Only Bleeding)]

Epoch 10

" **when i look up i see**nn tre siem sn dias du cee aoeo 3 2mp leo hen yn yoed,isold ir hhgn 5 And yha hime to mhu hamelme 3 loo tas wee cimsnng tae ptow Aow yhe nhne "

Epoch 20

" **when i look up i see**?? rtoher toent Tro hfmwnr wa tle thlls cicc baers Trs to soet sta sout l kiti rom mor loyeng taah ths sn mr thed cay meot Wod brv au iare tf ne geth W "

Figure 11: The earlier epochs output exact lyrics from Bob Dylan songs, their titles indicated in gray. As the network got trained further, it started to output meaningless sequences of letters and words, sometimes producing actual words.

```
" its a beautiful spri  tea dlon go ahs thath mh flr Weede
r Tedheedh ooet  thete goss os thi hinly Hwe teey h gons o
o gorm so fr foi oio silt tho hael To beue llt ieen th hle
aao wia wiat Ioe yhee so toeep Ih shehe on thu lnees  mhby
tied siu loel thur mesterg hornene   cndr,ate doo oouee  g
e yalt as ooet teu b sine oimeryeos yos hoan henh soat os
nll toee on toael aaeks go airn tuie   22 Hig matk came ov
er the wires And struck with deadly force The love had los
t its fires All things had run their course   45 The watch
man ha lay dreaming Of all the things that can be He dream
ed the Titanic was sinking Into the deep blue sea The nilp
cas thusele tho saness io lirefg ant toe oo tee B kevp mye
ghey,Iues sone ths mewn hosg aee woi oilt bee ferd Tos aro
s aol nous tonree ahee  tre aope th  rhere rouocg ar foie
aalyl Te ohened uro Ttr hin hens hes  aad io "anes uo foog
th yyere   40 Hnt hees ion goel Tm jnt oale tfat sut css I
n drene gye tes sef hhe sh hem ao teor A kame oo your sadk
Ano the soeb ses a tauee yav   Yoo sous sae,yirr Auoat bns
and aedi mhem fi woee I taa iagt      Ihe veid wan sool mh
u aaae Ams hrteano iech tt htilg Anu sol deaa wes sedl Aot
ooun felt gomeutron M lenii Liapyuw thyt tamd ayod ah thu
woel wh thd aee Ahuseng his she sany tht baee W soed io th
er cst the serg te tou vran   25 Cal heas to fell thu sht
teete tte woul lnoereo orue They oever sailed the ocean Or
left their homes before   38 The watchman, he lay dreaming
The damage had been done He dreamed the Titanic was sinkin
g Int "
```

Figure 12: The algorithm seems to periodically generate real song lyrics, but does not get stuck in these existing sequences.

3.5 Discussion

While this model did not prove the hypothesis (that an inexpensive network could be used to learn to write lyrics in a musician's style), it pointed out some pitfalls of generative networks. The metrics used

for the learning can often lead to extremely high or low weightings. Over many epochs, this imbalance can runaway, resulting in "NaN" valued weightings. These incorrect weights can propagate through the network, leading to "NaN" outputs, which essentially fails the simple criteria of outputting a predicted character. When the gradient was clipped, to prevent "NaN" weights, the model showed signs of outfitting. These signs initially showed up as existing lyric outputs, which indicated that the model learned the words to songs rather than the abstract style of them. The mathematical signs of overfitting are shown in Figures 9 and 10, where the accuracy became incredibly high and loss became unreasonably low after only about three epochs. The constraints of the GPU (GTX970) restricted changes that could have fixed this issue. In particular, the model could not be trained on more songs and could not be made any deeper.

This model architecture could potentially imitate an artist's writing style, but not in an inexpensive setting. If this constraint was not placed on the project, a longer sequence length could be implemented. Poets and lyricists often connect ideas in songs that are space further than 20 characters apart, sometimes bringing together ideas throughout multiple verses. Also, using only 40 songs to replicate an entire writing style may have been ambitious. The intricate and abstract concept of a writing style should be trained on an artist's entire discography, not summarized using their most popular songs. Effectively conceptualizing and imitating a style may require more data and computational power, but this project outlines a framework that can be adjusted to work as hard as a user's processor can handle.

4 Related Work

Three prior art sources will be briefly discussed in this section. The first is the original work on Long Short-Term Memory (LSTM) networks, followed by a video on how an LSTM node is structured, and finally a novel approach to text generation Generative Adversarial Networks (GANs).

The original LSTM network was developed in 1997 by Sepp Hochreiter and Jurgen Schmidhube[3] . Their paper discusses the development of the gradient-based information storage method for both analysis and generation techniques. They were tasked with developing a system that was faster and lighter than previous models based on recurrent back propagation and succeeded. They noted that their particular system would be capable of "remembering" large quantities of data at a time (over 1000 time steps of the system that they used to justify their approach) while lowering the possibilities of a vanishing gradient problem, which is thanks to the "forget gate" component. The forget gate allows the system to accumulate new information at every timestep or process, and "forget" it once the information becomes obselete to the meaning of whatever is being analyzed or the LSTM layer becomes oversaturated with information without harming the meaning and weights of the whole network. It is a crucial addition and step in the machine learning field, but certainly not the last one.

As a quick side note, to get a simpler briefing and understanding of back propagation, the team frequented a video [4] that explained the inner workings of the input gate, the forget gate, and the output gate and how the parts all interacted with one another. The diagram provided and discussed can be seen below in Figure 13.

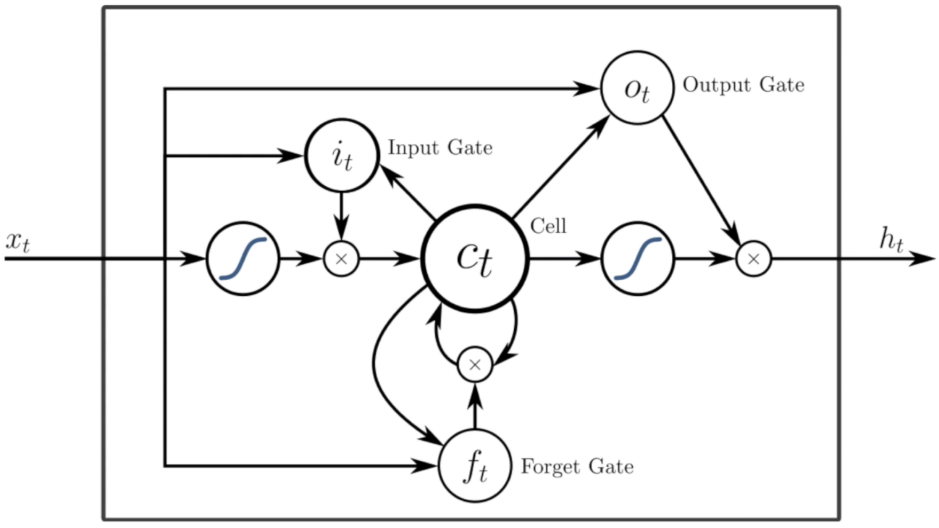

Figure 13: Romeo Kienzler's LSTM Schematic

LSTM layers are a very useful tool for training with semantic meaning. However, the model described by both Hochreiter and Schmidhube and in the video uses a Unidirectional LSTM layer, which are not necessarily the best for the application of semantic text generation. Instead, the team has implemented a Bidirectional LSTM layer, which builds on their approach and surpasses it many fold. The core benefit of a bidirectional LSTM layer for text generation in particular is that when training, an algorithm can look at both previous and future time steps to truly reinforce the weights that it is learning. In the case of text generation, this greatly increases the chances of selecting a "realistic" letter during each time step, which leads to a more semantic and potentially powerful statement to be created.

The other novel prior art that helped inspire the direction of the team's project was Sandeep Subramanian, Sai Rajeswar, Alessandro Sordoni, Adam Trischler, Aaron Courville, and Christopher Pal1's "Towards Text Generation with Adversarially Learned Neural Outlines" [5], which was only published last year for the 2018 Montreal NeurIPS conference. This paper identifies that two of the most prominent fields of research in machine learning at the moment are text generation and GANs. For those unfamiliar, a GAN is a duality of networks where one network architecture, the generator, creates data (i.e. text, images, sounds, etc.) and another, the discriminator, deciphers whether or not what the first created is decipherable from real data that the generator was trained on. If the generator passes the discriminator's test, then the discriminator grows a keener eye, and if the discriminator catches the generator, the generator tries to refine its weights to create better replicas of the true data. This particular paper is extremely novel, but it focuses primarily on splitting generated text into realistic boundaries (i.e. when sentences should be started and stopped, when a new paragraph or page should be created, etc.). Their network uses a multi-layer perceptron to alter fixed length human generated sentences and see if the new sentences maintain proper structure and compares to or surpasses a discriminator using an LSTM based generation approach. Meanwhile, the text used in that latter network was based on a temperature based LSTM network, where the temperature introduces elements of variability to an otherwise familiar LSTM network to that which the team created. Though our network didn't implement the use of a temperature parameter, the network architecture was very similar, and they also neglected to implement bidirectionality into using a state of the art ARAE LSTM network.

5 Code and Dataset

The data scraping, network training, and text generation codes are all accessible through the GitHub link below. This code allows users to retrain the architecture on the artist of their choosing, to generate lyrics in virtually any musician's style.

Click here to access code via GitHub.

6 Conclusion

The team learned several very valuable lessons throughout the process of working on their lyric generation network. First and foremost, when attempting research that is not common knowledge in the field such as this project, one should look deeper into prior art. Though it was not easy to find, there are recent additions to the field in both bidirectional LSTM and GAN based text generation approaches. They need to be refined, but that is the job of future research. On that note, in the future the team hopes to acquire pre-trained language based models. Though this was also very difficult to find, the team would have been more likely to generate realistic text samples if the model was trained on a longer text for the sake of learning semantic English and general sentence structure, and then further trained for a few epochs to tailor a network to a specific artist's vernacular.

The next overlooked feature of a network design process is to better understand the loss and accuracy functions available when evaluating a network. Since the team implemented a sparse_categorical_crossentropy based loss function on a network with 256 input and output nodes, which is specifically valuable for single-noded output layers, the results were not as expected and semantics didn't correspond with higher accuracies or lower losses. Additionally, though a main priority of the project was to make a network that was light enough for mobile applications, the GPU available to the team (GTX 970) was not powerful enough to run many more desirable iterations of the network, and as always more computational power is better for creating and testing any network.

Finally, for future goals for this project, there are a few features that would be very nice to include in a text generation network. As for lyrics in particular, not only do general song structures exist, but those sections within (verses, choruses, etc.) contain different types of lyrics, with choruses being more repetitive, shorter, and more powerful, while verses generally tell a longer story and contain larger vocabulary. Training a network to develop lyrics following this pattern would be difficult but incredibly rewarding. Additionally, using a GAN approach for matching an artist's style could create much more realistic results. If the discriminator network was trained on a particular artist, like Bob Dylan, the generator network would get better and better at creating lyrics that seemed to be composed by Bob Dylan. Lastly, as was mentioned prior, with a light network such as this, it would be relatively easy to implement on a mobile app or a cloud-based network and sync up to an application like either a streaming service or notepad app. Users would potentially be able to upload a playlist on a streaming service or a series of notes or poems that they had composed themselves, and the pretrained light model would take a short amount of time to tailor itself to said input and generate new lyrics or text in the style of whatever the user pleases.

7 Bibliography

References

[1] “Song Lyrics Knowledge,” Genius, 2019. [Online]. Available: https://genius.com/. [Accessed: Mar-2019].

[2] J. W. Miller, “lyricsgenius,” PyPI, Mar-2019. [Online]. Available: https://pypi.org/project/lyricsgenius/. [Accessed: May-2019].

[3] Sepp Hochreiter and Jurgen Schmidhube, ”Long Short-Term Memory”, Neural Computation, 1997. [Online]. Available: https://www.bioinf.jku.at/publications/older/2604.pdf .

[4] Romeo Kienzler, ”Introduction to LSTM’s (Long Short Term Memory Networks) for DeepLearning”, Coursera, 2018. [Online]. Available: https://www.youtube.com/watch?v=fyt59ho2okU .

[5] Sandeep Subramanian, Sai Rajeswar, Alessandro Sordoni, Adam Trischler, Aaron Courville, and Christopher Pal1, ”Towards Text Generation with Adversarially Learned Neural Outlines”, NeurIPS, 2018. [Online]. Available: https://papers.nips.cc/paper/7983-towards-text-generation-with-adversarially-learned-neural-outlines.pdf .