Automatic Song Lyric Generation and Classification with Long Short-Term Networks

Amit Tallapragada
Arizona State University
Barrett Honors Thesis
Heni Ben Amor (Thesis Director)
Jorge Caviedes (Second Committee Member)

I. Abstract

Lyric classification and generation are trending in topics in the machine learning community. Long Short-Term Networks (LSTMs) are effective tools for classifying and generating text. We explored their effectiveness in the generation and classification of lyrical data and proposed methods of evaluating their accuracy. We found that LSTM networks with dropout layers were effective at lyric classification. We also found that Word embedding LSTM networks were extremely effective at lyric generation.

II. Introduction

A. Lyric Generation and Classification

In recent years, music has become extremely prevalent in machine learning papers. Because the availability of license-free songs has increased, researchers have been able to apply existing models to new data. This paper will focus particularly on the classification of lyrics by genre as well as the generation of lyrics by artist. There have been a significant amount of papers on text classification and generation, but only a select few that zone specifically into lyrics. This paper aims to bring four contributions to the data science community: (1) Determine how effective a Long Short-Term Memory network is at classifying songs by genre using their lyrics; (2) Present a method to evaluate the effectiveness of a lyric classification model; (3) Present a Long Short-Term Memory network that can generate lyrics in the writing style of a particular artist; (4) Present a method to evaluate the effectiveness of a lyrical generation model.

B. LSTM Background

Long Short-Term Memory (LSTM) networks are an extension of Recurrent Neural Networks (RNN). A RNN is a type of neural network that can operate on sequence vectors by factoring in it's own output from previous time-steps when computing its current output. Because of this ability, researchers frequently use RNNs to find patterns within sequential data. In theory a RNN can remember dependencies from an infinite amount of previous time-steps. In application however, RNNs are limited by the vanishing gradient problem. During training, gradient values can come close to zero as they are multiplied against each weight layer. This in turn makes it difficult for a RNN's weights to adjust and learn new information.

C. LSTMs and the Vanishing Gradient

One way to solve the vanishing gradient problem is through the use of Long Short-Term Memory (LSTM) units when building a RNN [9]. LSTM units work on the premise of using gates (a unit that returns either a pass or block value) instead of activation layers in a neural network. A LSTM cell has three gates: a forget, update, and output gate. Each of these gates are sigmoid level neural networks that output values between 0 and 1; indicating whether data should be allowed to flow or be stopped. The formulae that make up an LSTM are shown below and were presented by Andrew Ng [12]:

$$\tilde{c}^{<t>} = tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c) \tag{1}$$
$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u) \tag{2}$$
$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f) \tag{3}$$
$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o) \tag{4}$$
$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t>} \tag{5}$$
$$a^{<t>} = \Gamma_o * tanh(c^{<t>}) \tag{6}$$

Γu, Γf, and Γo represent the update, forget and output gates. And the c^t represents a LSTM cell at time-step t and a^t represents the hidden layer output used to compute a cell's output at time t. As explained by equations 5 and 6, the LSTM solves the vanishing gradient problem by using addition to update gradients instead of multiplication. This means gradients can flow through gates and retain memory from previous timesteps as shown in Fig 1.

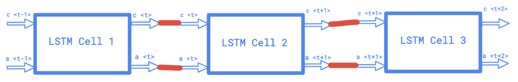

Fig. 1. Computations can be retained from previous time-steps using LSTM cells

D. Common LSTM Use Cases

LSTMs are commonly used to model sequential data such as time series [5]. LSTMs are extremely versatile and have been used for many natural language processing (NLP) problems. In 2017, Facebook computed over 4.5 billion language translations a day in 2017 using LSTMs [13]. This, among other applications have shown the versatility of LSTM cells in NLP. The scope of this paper will be limited to the use of LSTMs for text classification and generation.

E. LSTMs for Text Classification

There is a considerable amount of research being done in text classification and even some specifically for lyric classification. In [17], Tsaptsinos discusses the effectiveness of four different machine learning techniques and their effectiveness in lyric classification. In this study, LSTMs were one of the most effective techniques tested; when given over a million data points and 20 genres to classify by, an LSTM network could predict a lyric's genre with a 49.77% accuracy. Outside of using a normal LSTM, researchers have also used different variations of the cell to improve classification accuracy. In their paper Zhou et. al. creates a Bidirectional LSTM for text classification which outperforms a normal LSTM cell [19]. Our paper will use Word Embeddings and dropout to create a novel LSTM network for text classification for four genres.

1) Brief Overview of Dropout: Dropout is a technique introduced by Srivastava et. al. that multiplies mask vectors to randomly drop network cells. It is used to prevent a network from overfitting on a dataset and helps the network to find robust connections [16]. We can configure the amount of cells we want to drop by adjusting a dropout function's drop rate. For example, if our drop rate is 20%, our network will randomly drop 1 in every 5 cells during weight updates. Brownlee recommends using a dropout rate between 20% - 50%, suggesting starting with a lower rate to prevent having a rate that would either have minimal or overly aggressive effects on a network's training [3]. Because of this, anytime we use dropout in our models, we start with a dropout rate of 20%.

2) Brief Overview of Word Embeddings: Word Embeddings are a way to represent a corpus of text in an n-dimensional continuous vector space. The location of a word in the vector space is learned from a given corpus of text and can help mathematically represent relationships between words.

Fig 2 shows how Word Embeddings can pick up on relationships between words in it's vector space. Representing words in a continuous space allow networks to draw robust relations between words. They have been used in the past for text generation and have shown positive results [2].

F. LSTMs for Text Generation

In, [14] Potash et. al. trained an LSTM to generate lyrics in the writing styles of different rap artists and got positive results. The lyrics their model generated were very similar in rhyme density and lyrical similarity to songs written by the artist they tested against. Aside from lyrical generation, LSTMs have also been used to create actual sheet music [4]. In both these papers and others in the text generation space,

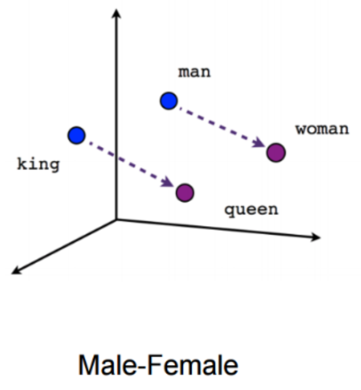

Fig. 2. Word Embedding depicting semantic relationships between words [7]

the two most common approaches are character generation, where the LSTM predicts the next character given a sequence of previous characters; and word generation, where the LSTM does the same prediction but on a word level. This paper will focus on word generation to see it's effectiveness in creating coherent lyrics.

III. Methods

A. Building Our Datasets

1) Lyric Classification Dataset: As mentioned by Tsaptsinos, given a large amount of data and genres, a LSTM network could classify songs with a 49.77% accuracy. We wanted to see if giving a LSTM network less genres to classify by could improve it's accuracy. For our classification problem, we obtained our lyrics dataset from Kaggle.com [11]. The original dataset contained over 360,000 songs that unequally spanned 12 genres. We reduced the scope of our problem to focus on four genres. From here, we chose the four most common genres for our dataset (Pop, Hip-Hop, Rock, and Metal). Metal, the least common of the four, had 2000 data points. In order to prevent classifier bias, the remaining genres were each reduced down to 2000 points – resulting in a total dataset of 8000 points.

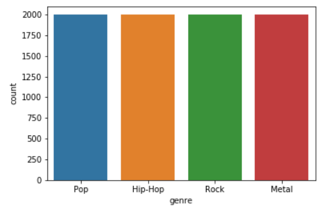

Fig. 3. Our Revised Four Genre Lyric Dataset

In order to train our classification model, we had to preprocess our data. To create our feature set, we extracted the lyrics from the Kaggle dataset and tokenized them. As Howard et. al. found that stop-word removal adversely affected models at lyric classification [10], we retained all stop-words in our dataset. Next, we converted our lyrics into numeric sequences such that each word was encoded to a numerical value. The longest song in our dataset was 8196 words, so we padded the rest our lyrics to be of the same length. To create our target variable, we extracted each song's genre from the Kaggle dataset and used one-hot encoding to represent each genre as a vector. Finally we split the dataset into two groups: 80% of it was used for training while the remaining 20% was for testing. All of our preprocessing was done using Python. We used Keras to tokenize and convert our text to sequences, and Sklearn to create our train/test split.

2) Lyric Generation Dataset: We chose to emulate the song writing styles of Taylor Swift, Kanye West, and Green Day. All three datasets were pre-processed and passed into our LSTM network. We wrote a Python script that downloaded 100 songs from each artist from AZlyrics.com. We then extracted the lyrics from this dataset and tokenized them once again allowing stop-words. Generative text LSTMs work by predicting the next word or character given a sequence of n values from prior timesteps. To do this we made one string of all a particular artist's songs and broke the song up into 50 word chunks with an overlap from the previous chunk of 3 words. The target variable for each of these songs was the next word in the song. This process is visualized in Fig 4.

```
Sequence:  ['said', 'the', 'way', 'my', 'blue'] Next Char: eyes
Sequence:  ['my', 'blue', 'eyes', 'shined', 'put'] Next Char: those
Sequence:  ['shined', 'put', 'those', 'georgia', 'stars'] Next Char: to
```

Fig. 4. An example of 5 word sequences with a 3 word step

By creating a dictionary mapping words to numbers, we were able to encode each sentence. We created a three dimensional matrix that contained one-hot encoding representations of each 50 word sequence by looping through our word sequences and populating a matrix of size (50, 3076). Because we had 12266 word sequences, our final matrix was of size (12266, 50, 3076).We also one-hot encoded our target variables. We used this same process to encode all three of our artist's lyrics.

B. Building our LSTM Models

1) Building our Classification LSTM Models: Our LSTM network consisted of one LSTM layer with 128 LSTM cells. We used 128 cells as a starting point because this is the default amount of cells for a hidden layer in Keras. The output of this layer was passed to a dense layer that used a softmax activation. We used softmax because the output of a softmax layer will be a vector the size of the number of genres we are classifying by where each value represents the probability value of a given input being a particular genre. Because we used softmax for our output activation and we used one-hot encoding for our target variable, the Keras documentation recommended using categorical crossentropy for our loss function [1]. We used RMSProp with a .001 learning rate as our optimizer. RMSProp as explained by Hinton et. al., is a mini-batch version of the rprop optimizer and works by keeping a moving average of the squared gradient for every weight in a network [8]. By computing the moving average, we normalize our gradient and in turn balance weight updates. RMSProp decreases the update size, when a gradient is too large, and increases it when a gradient is too small. Thus, this optimizer is helpful in preventing the vanishing gradient problem. We also created two more networks with different parameters to the network described above to see if they would increase prediction accuracy. We created a network with an additional LSTM layer with 64 cells. We tried this because increasing the amount of hidden layers can potentially increase the capacity of a model. As described by Goodfellow et. al. capacity is a model's ability to fit a wide variety of functions. Models with low capacity can struggle to fit to a dataset whereas models with a high capacity can overfit on a dataset [6, p. 110]. Comparing between a single and multi-layer LSTM network allowed us to compare which network had better capacity. We created another network with a dropout layer with a 20% drop rate. All of these networks were created using Keras.

2) Building our Generative LSTM Models: We created an LSTM network with a word-embedding layer for our input layer. We used a Keras embedding layer in our network which trained itself on input sequences of a particular artist and generated a 50-dimensional representation of each word. This vector representation of our input sequence was then passed into a 128 cell LSTM layer. We once again added a dropout with a 20% drop rate. The output of this layer was passed to a dense layer that used a softmax activation where the output was a vector where each value represented the probability of a particular word being next given an input sequence. For the same reasons explained for building our classification model, we used categorical crossentropy for our loss function and RMSProp as our opitmizer. Both of our networks were created using Keras. We trained our network at an arbitrarily large value of 200 epochs on the Google Cloud Platform on a Nvidia Tesla K80 GPU.

C. Evaluation Metrics

1) Classification LSTM Model Evaluation Metrics: We evaluated our classification LSTM by using the predicted outputs for our testing dataset. We also compared the results of both classification networks against a Feed-Forward Network to determine whether an LSTM provided any advantages for improving classification accuracy.

2) Generative LSTM Model Evaluation Metrics: Determining how similar a generated song is to an artist's actual songs is a difficult task. Parameters such as genre, length, and context all make up the meta-data of an artist's song. Creating a way to account for all of these is out of the scope of this paper. Ultimately, we decided to treat each song as a document and ran a popular document similarity algorithm called cosine similarity TFIDF (term frequency-inverse document frequency) on all of a particular artist's lyrics and a LSTM generated song. The algorithm works by counting the terms in a document and creating a regularized vector for each document in a corpus. Each value in the vectors represents the prevalence of a particular word in the document and the entire corpus. Sindhuja et. al. describes the equations below that make up cosine similarity TFIDF [15].

$$tf_{i,j} = n_{i,j} / \sum_{k} n_{k,j} \tag{7}$$

$$\big(tf/idf\big)_{i,j} = tf_{i,j} * idf_i \tag{8}$$

$$d_j * idf_i = log\Big(\frac{\mid D \mid /}{\mid d : t_i \epsilon d \mid}\Big) \tag{9}$$

$$X_j = [x_{1j}, x_{2j}, ..., x_{nj}] \tag{10}$$

$$corr(a,b) = \langle \frac{a}{\mid\mid a \mid\mid} * \frac{b}{\mid\mid b \mid\mid} \rangle \tag{11}$$

$$cos\theta = corr(a,b) \tag{12}$$

t_i represents a particular word in the corpus and d_j represents a particular document in the corpus. Equation 7 shows how to calculate term frequency for a particular word where $n_{i,j}$ represents the number of occurrences of a term t_i in some particular document d_j. $n_{j,k}$ is the frequency t_i in d_j. Equation 8 shows how to compute the TFIDF vector which represents each term's importance in a document. Equation 9 shows how to compute the inverse document frequency of some document d. Here D is the number of documents in a corpus and the bottom computation in the fraction is the number of documents where t_i appears. Equation 10 explains the creation of matrix X_j where each value is a a tf/idf vector as computed in Equation 8. We then create a frequency matrix X_j for each document in our corpus. Now we can compute the same matrix for our generated text, and compute it's similarity to each document in our corpus using cosine similarity (equations 11 and 12) and generate a similarity vector. We can take the average of this vector to estimate the average similarity between a generated song and the artist's actual songs.

IV. Results

A. Classification LSTM Results

The single layer LSTM network (LSTM 1), Multi-layer LSTM network (LSTM 2), the single layer LSTM with one dropout layer network (LSTM 2), and our Feed Forward network (FFN) accuracy results are shown below in Table 1.

	Test Accuracy	**Train Accuracy**
FFN	**33.73%**	**30.31%**
LSTM 1	**71.63%**	**71.64%**
LSTM 2	**52.00%**	**95.07%**
LSTM 1D	**72.31%**	**72.51%**

TABLE I
TEST AND TRAIN ACCURACY FOR OUR FOUR NETWORKS

The test and train accuracy values are percentages that indicate how many values our networks accurately predicted on our test and train data. We included the training data accuracy as it will allow us to analyze potential over fitting. Training LSTMs are a computationally heavy task so all the networks were trained on Google Cloud Platform with a Nvidia Tesla K80 GPU for 100 epochs.

B. Generation LSTM Results

We generated 10 songs with our Taylor Swift lyrics generator (LSTM 1) each at the average length of one of her songs (307 words). We computed the average cosine similarity TFIDF of each of these songs in respect to our corpus. We then averaged out these values. We computed the same value for 10 songs generated by textgenrnn (Textgen), a robust open-source text generator library which uses a series of LSTMs and optimizers [18]. The table below shows the average of each categories 10 songs.

	Similarity Score
LSTM 1	**08.917%**
Textgen	**09.567%**

TABLE II
COSINE SIMILARITY ON THE TFIDF OF OUR LSTM 1, TEXTGENRNN, AND ARTIST LYRICS

V. Discussion

Overall, our research has shown us that LSTMs are effective tools in the classification and generation of lyrics. We hope to continue our research by implementing the future improvements discussed in this section.

A. Classification LSTM Discussion

All three of our LSTM models significantly outperformed our baseline FFN Model. This indicates that LSTM networks can be an effective model for lyric classification. The LSTM network with a dropout layer performed the best of our four models with a 72.31% accuracy. This approach proved to be effective in improving the network's accuracy.

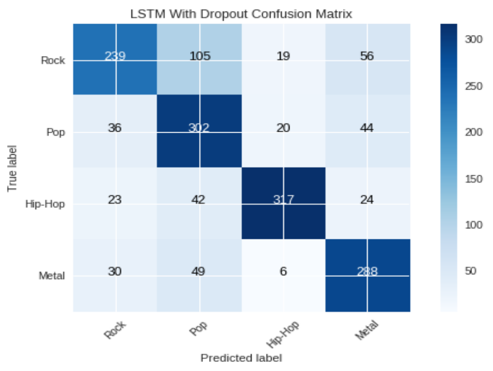

Fig. 5. A confusion matrix of the LSTM 1D depicting the number of songs the network properly classified into the four genres

As seen in Fig 5, our LSTM 1D network was best at classifying hip-hop music. This could be because of the distinct lyrical patterns rap songs have in comparison to our other three categories. The genre our network performed worst on was the rock genre. This could be because many rock songs bear similarities to pop and metal as well. This is evidenced in the confusion matrix as the network's second most predicted value for actual rock songs was pop. The line between genres can get murky in these scenarios because inevitably, they are human-made concepts and can sometimes be arbitrary.

1) Future Improvements: We could have potentially increased the accuracy of our LSTM 1D model by increasing our dropout rate. Another interesting observation from our results was that our network with 2 LSTM layers seemed to have over-fitted to the training data as it had a 95.07% accuracy on it in comparison to its 52% accuracy on the testing data. As mentioned above, this is likely because the capacity of a 2 layer LSTM network for the task of classifying 8000 songs to 4 genres was higher than needed. If we were to continue using 2 LSTM layers for this problem, we would also try a more aggressive dropout rate.

B. Generation LSTM Discussion

Table 2 above showed that our network performed a little worse in terms of similarity scores when compared against Textgenrnn. That being said, it did exceptionally well without having optimizations parameters that Textgenrnn automatically applies. Below is a sample of our network's generated Taylor Swift lyrics:

> *[album version line:] that's fine I'll tell you like my life makes life good mad mad way it through the way you feel like he should see it like a violin to like you're laughing through me like a wish they are right through me more take it right back from everything this time you were romeo i understand it all too well leave my own mind died i have you didn't know for you are right that way time you hadn't mine*

This is a sample of the network's generated Kanye West lyrics:

> *just made me you more with let its hate that real ridiculous felt and on i know him act these ridiculous take its oh to me, never dont seen stop jokes me before how stop (jesus news? up no in the lights where thats may and when your desert workout god and just somebody that make about its long [g1] breathe) for these cold at pull famous ocean] for many keep southside with since told drop it we go out i new tryna about west]*

1) Future Improvements: In it's current state the network on occasion outputs the same word repeatedly. We could implement a diversity setting similar to Textgenrnn's. Our network uses a softmax activation layer for it's output which means it returns an array of probabilities where the index of each value represents the word. A diversity parameter would enable us to generate songs by randomly picking values from a top-n amount of the most probable next words instead of always defaulting to the most probable word. This would help us reduce repetition. Furthermore, we could also work on improving our evaluation metric. One particular issue with our evaluation metric is that it is designed for document similarity rather than lyric similarity. It is based on the fact that two documents are similar if they have similar words at a similar frequency. With lyrics however, we need to account for verses and rhythm as well. Polash et. al. recommended using a modified version of the TFIDF cosine similarity method that also took into account the number of verses in each song. This could be a potential first step in creating a more effective lyric similarity metric.

References

[1] Losses. *Keras Documentation*.

[2] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Józefowicz, and Samy Bengio. Generating sentences from a continuous space. *CoRR*, abs/1511.06349, 2015.

[3] Jason Brownlee. Dropout regularization in deep learning models with keras, Mar 2017.

[4] Keunwoo Choi, George Fazekas, and Mark B. Sandler. Text-based LSTM networks for automatic music composition. *CoRR*, abs/1604.05358, 2016.

[5] Felix A. Gers, Douglas Eck, and Jürgen Schmidhuber. Applying lstm to time series predictable through time-window approaches. In Roberto Tagliaferri and Maria Marinaro, editors, *Neural Nets WIRN Vietri-01*, pages 193–200, London, 2002. Springer London.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[7] Google. Vector representations of words. Accessed: 2019-02-18.

[8] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Lecture 6a overview of mini–batch gradient descent, 2012.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[10] Sam Howard, Carlos N. Silla Jr, and Colin G. Johnson. Automatic lyrics-based music genre classification in a multilingual setting. In *Thirteenth Brazilian Symposium on Computer Music*, 2011.

[11] Gyanendra Mishra. Kaggle: 380,000 lyrics from metrolyrics. Accessed: 2019-02-18.

[12] Andrew Ng. Sequence models: Long short term memory (lstm). Accessed: 2019-02-18.

[13] Thuy Ong. Facebook's translations are now powered completely by ai, Aug 2017.

[14] Peter Potash, Alexey Romanov, and Anna Rumshisky. Ghostwriter: Using an lstm for automatic rap lyric generation. pages 1919–1924, 01 2015.

[15] B Sindhuja. Usage of cosine similarity and term frequency count for textual document clustering.

[16] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

[17] Alexandros Tsaptsinos. Lyrics-based music genre classification using a hierarchical attention network. *CoRR*, abs/1707.04678, 2017.

[18] Max Woolf. How to quickly train a text-generating neural network for free. Accessed: 2019-02-18.

[19] Peng Zhou, Zhenyu Qi, Suncong Zheng, Jiaming Xu, Hongyun Bao, and Bo Xu. Text classification improved by integrating bidirectional LSTM with two-dimensional max pooling. *CoRR*, abs/1611.06639, 2016.