Comparing Three Models for Natural Language Lyric Generation

Daniel Saunders

December 18, 2015

1 Abstract

We consider three models for generating new song lyrics based on song lyrics from a specific artist. Our word-level and character-level n-gram models and recurrent neural network model are trained on a set of lyrics of a given artist, and produces a new song-length lyric set in the style of said artist. We are looking to generate lyrics that utilize word choice, line length and structure, and end-of-line and in-line rhyme similar to that of the source artist. Although we were not able to generate satisfactory lyrics, the results from the attempts are promising and show different strengths based on their implementations.

2 Introduction

The problem of generating lyrics that successfully mimic the style of its source artist is a difficult one, largely due to the haphazard structure present in lyric-writing (repetitive choruses, common slang and abbreviation usage, annotation strings such as "Verse" or "Repeat 3x"). Studying this will allow us to understand just how difficult it is to write

quality lyrics, and, if enough progress is made, will allow us to use NLP methods in creating lyrics for actual commercial use, especially if our models can produce "better" lyrics than humans typically do.

Taking inspiration from poetry generation and political speech generation, my first approach was to use a word-based n-gram model, which ended up reproducing small chunks of the artists' song lyrics, especially with large n, and otherwise did not produce many interesting - or grammatical - lyrics.

Secondly, I implemented a recurrent neural network, since these are famous for learning syntactic structure in text, since it creates an internal state of the network, reinforced by training on multiple times on the input corpus. This model, instead of reproducing large chunks of the input lyrics, produced a large amount of ungrammatical jargon, and I was forced to abandon this as well.

For my third attempt, I implemented a character-based n-gram model, which works somewhat better than the previous two with careful selection of n (in the range of 7-9 seems to work best), but I was unable to include the calculation of goodness metrics or structuring the output into lines of lyrics. This model simply outputs a paragraph of text (1000 characters) which has been trained on the input lyrics, treated as a paragraph. If I had more time, I would restructure this implementation to work as the previous ones did, and possibly get better results.

I also implemented a goodness metrics method, and so, for each line of the generated lyrics, the program would generate $K = 10$ candidate lines, and then score each based on the goodness metrics, which include end-of-line rhyme, word frequency probability, and line length probability. The candidate with the highest score would be placed into the lyric set as the next line.

We pose the question: Are any of these models, combined with the goodness evaluation method, sufficient to produce high-quality lyrics in the style of their source text?

3 Related Work

There are a number of mildly successful attempts at generating song lyrics using natural language processing techniques.

For example, Hieu Nguyen and Brian Sa, of Stanford University, implemented a rap lyric generator, for which they utilized a linear-interpolated quad-gram model with absolute discounting for uni-gram, bi-gram, tri-gram, and quad-gram models according to their own hand-set weights. They also generated candidate lines (K = 30), and ranked each according to a handful of goodness metrics. Their results were promising on simple lyric sets, but oftentimes, the lyrics would be ungrammatical or make no sense as a whole, especially when the source lyrics were noisier (i.e., less structured). The key to the success of their model, however, was their use of hand-set weights, a form of *a priori* knowledge which biased the text generation.

Another attempt, by Wu et al., was to treat lyric generation as a machine translation problem. They implemented a challenge-response system, which would respond to a given line of rap to a "machine translation" response. The model was trained off of pairs of lines of rap lyrics, and thus would given many rhyming and relevant responses to challenge lines. Wu et al., implemented training without any supervision, (i.e., no a priori phonetic or linguistic knowledge), and stochastic transduction grammars in order to accomplish this. They also showed that they could improve the training data by implementing an unsupervised rhyme scheme detection algorithm. Their model produced surprising fluent and rhyming responses, despite the noisy input data, and they were safe in claiming that their model was superior to a number of off-the-shelf natural language generation packages.

4 Data

My dataset is sourced from "rap.genius.com", otherwise known as Genius, which I was able to obtain by implementing a web scraper that finds an artist's most popular 27 songs given the artist's name. Unfortunately, I was unable to use the Genius API to grab all of an artist's songs, due to the valuable nature of song lyrics, and so I was only able to train my model and generate lyrics based on the amount of songs mentioned above. For example, when the artist 'Drake' is input to the web scraper, we find that there are 15129 tokens and 2729 unique tokens in his most popular 27 songs. Genius has millions of lyrics from thousands of artists from a wide range of genres, including rap, pop, rock, country, rhythm and blues, and foreign music (from France, Germany, etc.)

Below are some statistics drawn from the 27-song corpus of a few artists:

Table 1: Artist Statistics Examples

Table 2: Jay-Z

Token Count	17941
Unique Tokens	4059
Average Line Length	8.8 words
Average Song Length	73 lines

Table 3: Beyonce

Token Count	10719
Unique Tokens	2218
Average Line Length	8.6 words
Average Song Length	44 lines

Below is a small sample of the combined unigram, bigram, and trigram model trained off of Fetty Wap lyrics, taken from the word-based n-gram model. The tokens in the parentheses are a sequences of words that appear in the corpus, while the following tokens in brackets are the tokens that follow such a sequence at any point in his songs:

('I', 'guess'): ['this', 'you', 'its', 'I', 'it', 'Im'], ('Imagine', 'how'): ['it'], ('ends',): ['I'], ('I', 'fly'): ['private'], ('Im', 'owed'): ['You'], ('high', 'school'): ['games'], ('out', 'Girl'): ['you'], ('didnt', 'have'): ['money'], ('the', 'wine'): ['is'], ('it', 'my'): ['way'], ('church', 'on'): ['Sunday', 'Sunday'], ('tired',): ['of'], ('gas', 'with'): ['a'], ('easily', 'influenced'): ['by'], ('that', 'nigga'): ['Six', 'cant', 'always'], ('building',): ['but', 'then'], ('me', 'stay'): ['true', 'true'], ('flow', 'and'): ['uh', 'I'], ('on', 'the'): ['dance', 'counter', 'floor', 'cover', 'phone', 'gas', 'street', 'beach', 'Lamborghini', 'floor', 'street', 'gas', 'street', 'phones', 'move', 'mirror', 'Takeout', 'Jaguar', 'lease', 'top', 'champagne', 'floor', 'team', 'shelf', 'road', 'world', 'double', 'beat', 'seats', 'lease', 'star', 'kid', 'kid', 'road', 'floor']

From this sample, you can see that many n-grams have only a few possible following words, and therefore, the lyric generation will not be able to generate unique lyrics easily by simply choosing random next words in the n-gram process.

5 Methods

5.1 Word-Level n-gram Model

I implemented a combined unigram, bigram, and trigram model, which was trained on the 27-song dataset mentioned above. The model accounts for the frequency of words in the corpus, and the frequency of unigrams, bigrams, and trigrams, as well as the goodness metrics of between-line rhymes.

In order to generate new lyrics, the model first randomly chooses a relatively common token from the bag-of-words created from the training lyrics. Then, the model randomly chooses a token that follows from the possible n-grams that precedes it in the currently generating line, and adds it to the line. After a certain number of words have been generated (equal to the some number of words on the normal distribution

determined from the training corpus), the program terminates the line and begins generating the next. The program halts after generating 30 lines of lyrics, which is a new implementation, as the old version of the model would generate about the average number of lines in the source artist's lyric set.

As for performance, the lyric generator seems to be over-fitting to the training data, in the sense that small chunks of the original lyrics tend to come out in the lyric generation process. This tends to vary with the size of choice of n for the length of the grams; with n = 2 or 3, or combined, the model produces much less of the source text than with n ¿ 4, but tends to be less grammatical and coherent.

I believe this over-fitting occurs since I have not scraped enough lyrics from Genius, and unfortunately, this cannot be helped because of the cost of getting these lyrics and the inability to scrape the rest off of their website. This may also be occurring because of the relative uniqueness of the lyrics written by the artists I have trained my model on so far. Further, the word-level n-gram model is not very intelligent in that it does not really "learn" much of anything, even with the include goodness metrics it fails to understand syntactic structure, but instead, only learns how to sequence words together.

Upon implementing features such as end-of-line and within-line rhyme scheme, The lyric-generation program generates a number of "candidate" lines (K = 10) for each new line in the song, and the program ranks each candidate according to the trained model and the metrics we choose to care about. The best candidate line will become the next line in the generating lyrics. This helped to create more rhyming lines and lyrics that used words that their source artist used most frequently.

5.2 Recurrent Neural Network Model

For my second attempt, the recurrent neural network model will predict the probability of a line in the lyrics given the words in the line. Specifically, the probability of

seeing a line of lyrics $\vec{w}$ is $P(w_1, w_2, ..., w_n) = \prod_{i=1}^{n} P(w_i|w_1, ..., w_{i-1})$, or in words, the probability of a sentence is the product of the probabilities of each word, given the words that came before it. Intuitively, it seems that sentences with high probabilities are those that are grammatically correct. Since we are able to predict the probability of a word given previous words, we are able to generate new text. The source code for this model is located in "rnn_generate.py".

Some data pre-processing had to be done before we could begin using our RNN model. We prepend a dummy "LINESTART" token and append a dummy "LINEEND" token to each line, since we would like our model to know about tokens that typically start or end lines. Because of this, we can ask questions like: Given that we have m tokens $w_1, w_2, ..., w_m$, what is the probability of w_{m+1} = "LINEEND"? We also create word-to-index and index-to-word vectors, which act as input to our RNN model, and denote the "LINESTART" token index as 0, and "LINEEND" token index as 1. So a given training example might be [0, 123, 432, 84, 6], and our corresponding next sequence might be [123, 432, 84, 6, 1]. Since our goal is to generate the next token in the sequence, the vector is simply shifted over by one when the prediction is made.

The input $\vec{w}$ is a sequence of words, and each w_i is a single word. We represent each word as a vector of length equal to the vocabulary size. So, the word with index 84 would be a vector of all 0's, except for a 1 at postion 84. This was used in the RNN code instead of doing it during the pre-processing. The output vector $\vec{o}$ of the RNN has the same vector format, where each $\vec{o_t}$ is a vector of length equal to vocabulary size, and each index represents the probability of that word being the next in the line.

At each time step t, we are concerned with the values of the parameters U, V, and W, which together represent the internal state of the neural network. We may think of the parameter U as a mapping from the word vector to the hidden state of the RNN, V as a mapping from the hidden state of RNN to the word vector, and W as the determiner for how the hidden state evolves over time. Now, with these in mind,

we further define $s_t = \tanh(Ux_t + Ws_{t-1})$, where s_t is the value of the hidden state at time t, and $o_t = \text{softmax}(Vs_t)$, where o_t is the value of the output at time t.

For the initialization of the neural network, we must randomly assign values to the parameters U, V, and W, and we do so in the range of $[-1/\sqrt{n}, 1/\sqrt{n}]$, a typically recommend approach. This is done in the "init()" method in the "RNN" class. For the forward propagation (the prediction of word probabilities), I implemented the "forward_propagation()" method. This returns the calculated outputs o, as well as the values of the hidden states s, which we shall use later in the computation of gradients. I also implemented the method "predict()", which calls our forward propagation method and returns o_t such that the output vector at time t has the greatest probability over all vectors in the output. We may use this when evaluating our model. Running this prediction method will give us a random result as of right now, since we have only just initialized U, V, and W to random values.

In order to train the network, we must define a way to calculate the errors it makes. We denote the loss by L, and we wish to find U, V, and W in a way that minimizes this L for the training corpus. One way to model this is by the cross-entropy loss function, given by $L(y, o) = -\frac{1}{N} \sum_{n \in N} y_n \log o_n$, where we have N words in our corpus, and where y is the expected value and o is our training example. This is loss with respect to our predictions, or the sequence of words we expect to see. This loss function is implemented in "calculate_loss", where the loss is printed to the console during the model's training period. The loss L for random predictions, in a vocabulary of size K, should be given by $-(1/N)N \cdot log(1/K) = log(K)$, since each next word will be predicted correctly with probability 1/K.

Now, to train the RNN, I chose the standard stochastic gradient descent algorithm, in order to find U, V, and W such that they minimize the total loss on the training data. We iterate over each training example, and on each, we move the parameters in a direction that reduces to total error. The directions are given by the gradients on

the loss: $\frac{\partial L}{\partial U}, \frac{\partial L}{\partial V}, \frac{\partial L}{\partial W}$. The algorithm also needs a learning rate, which defines how big the move towards the error reduction is on each iteration. In order to calculate the gradients mentioned about, we use an algorithm called back-propogation through time, or BPTT. Since the parameters are used in all time steps in the network, the gradient at each output depends on the the calculations of the current time step and all its preceding steps, using the chain rule from calculus. I implemented this in the "bptt()", where, in simple terms, the algorithm take in a training example, (x, y), and returns the gradients $\frac{\partial L}{\partial U}, \frac{\partial L}{\partial V}, \frac{\partial L}{\partial W}$. I also implemented gradient checking in "gradient_check", which uses the "bptt()' method so as to make sure the the SGD algorithm is actually reducing the errors that our neural network makes in training predictions.

As for the actual implementation of the SGD algorithm, this was carried out in two steps: (1) The method "sgd_step" calculates the gradients above and updates U, V, and W, and (2) An outer loop, "train_with_sgd" that iterates through all the training examples and adjusts the network's learning rate. This concludes the training portion of the recurrent neural network. We run the training loop on our dataset for a default of 100 epochs before generating new text. The program allows the user to input his or her preferred number of epochs, but with fewer than 100, the model tends to output complete gibberish, and with greater than 100, the model tends to over-train on the input corpus.

Lastly, I implemented the method "generate_line()", which takes the model as a parameter and generates sequences of words according to the probabilities provided by the neural network model. Each line generation halts when the "LINEEND" token is found, and so there may be lines of many different lengths that are generated. For this model, I decided only to generate 30 lines of lyrics, as the training, generating, and goodness-checking takes quite a while, even with the small training sets that were provided by Genius. Furthermore, the same goodness-checking function that I implemented for the word-level n-gram model was used, and the same number of

candidate lines were generated on each iteration of the "generate_line()" function (K = 10).

5.3 Character-Level n-gram Model

The character-level model was implemented much like the word-level model, but with the necessary inclusion of a string from which to start generating text, which the model treated at its "history". Given n, the history string must be of length at least n, and the last n characters of the string must be included in the model's n-gram mapping structure. Otherwise, the program would have to start generating based off a random length-n string in the mapping structure.

The mapping structure itself is comprised of length-n strings, each mapped to a list of their possible following characters which are couple with their probability of being next in the sequence. In this way, the model will calculate the most likely sequence of 1000 characters from the starting "history" string, and output the result.

I was unable to implement this model as I did with the previous two, since it only generates one long string based on the input as a single line, instead of generating based on the input as a collection of lines as lyrics and outputting a collection of lines scored based on goodness metrics.

6 Results

Overall, the results from all three of the models were fairly discouraging. A big part of this was the unavailability of a large source corpus, the lack of intelligent NLG models, and the noisiness and structural quality of the domain of song lyrics.

6.1 Word-Level n-gram Model

This model rarely produced interesting results, since it mostly reproduced phrases in the source lyrics in different orders. I found that giving an n equal to 2 or 3, or combining the model for both of these would produce decent results, but would rarely produce anything but small chunks of different songs strung together. Since the lines each began with a newly selected random word, each line typically had at least one chunk from one of the source artist's songs in it (see appendix A).

6.2 Recurrent Neural Network Model

The recurrent neural network model differed greatly from the other two models, as it produced a large amount of ungrammatical nonsense. Occasionally, it captures some syntactic structure, but, failing to teach this model anything about the underlying parts of speech, the model falls flat in producing grammatical sentences. I believe the reason for this poor performance is due in part to the length of time the model was trained for (after 30 epochs, the loss was only reduced from about log(K) to roughly $(1/2)log(K)$, indicating poor prediction accuracy, and after 150 epochs, roughly $(1/4)log(K)$), as well as RNNs inherent inability to learn dependencies between words that are several indexes apart.

6.3 Character-Level n-gram Model

There is not much to say about this model, since its implementation is incomplete compared to the previous two. However, the phrases that this model was able to generate were often much better than those of the word-level n-gram implementation, depending on the given n and the artist in question. For example, giving an n of 6 or 7 on rap lyric domains often produced interesting results (see appendix C), whereas an n of 8 or 9 on alternative or rock music typically worked better. I believe that small

changes like these point out major differences in lyric-writing styles between genres, and more experimentation with different features could help separate out lyrics by genre.

7 Discussion and Future Work

The best way to augment these models is first to attack it from a parts-of-speech, or grammatical perspective. Perhaps each line should be generated from one of several very general context free grammars, and we may treat the parts of speech as features on which to train a new model. This could work well in conjunction with the RNN model, as that is where it specifically fails to perform well. Another possible augmentation is to experiment with the goodness metrics, particularly in finding a good way to implement the within-line rhyme metric, since that had to be omitted because of the toll it took on the run-time of the systems, and finding proper weights for end-of-line rhymes.

Another model to consider is the Long Short Term Memory network, or LSTM for short. This is a special type of recurrent neural network which is capable of learning long-term dependencies, one which our neural network fails to capture. The key difference is in the RNNs parameter W, which is a single repeating layer in the standard neural network model, whereas the LSTM has an interactive, four-layer repeating module, governing by several complex functions. According to research into neural networks, this kind of network is ideal for natural language generation, and would lend itself well to our lyric-generation problem.

8 Appendices

Below I've included some generated lyrics from the models that I have implemented.

8.1 A: Word-Level n-gram Lyrics

Artist: Snoop Dogg, n-gram Order: 3

say i am so original pop twice then lean back
sitting by a tree nigga
yo bus boy dont miss that bus at ya mothafuckin school yo
over the dire reap while i stay king by any means scream on it my nigga turn that
pretty smart how i live in fear of a nigga with the
lbsee yall prayin that a g thang east side
party goin dumb put it back together like my nigga turn that shit up
their breasts literally i can switch it deliver this shit is aight for your motherfucking
too caught up with you nigga this that mothafuckin straight crack
account 14 carrots in ya favorite girl account 14 carrots in ya mothafuckin
were waking em up yeah and thats realer than the real 20s 20 minutes holla
aint funny i do what i say and do it moving real slick
field no mistakin and crack the little homie told me you thinkin bout snatchin jewelry
then brag to the sizacks my nigga turn that shit is
masses knockin niggas like cassius dogg pound gangsta assassin its like
spinach chew it up just blaze that
pain in the llac with doc in the
gators when i take it all around the town in 21st street snoop dogg it all

nervous fuck you him and him i was close to the aston nigga yo get
cheese remember days homey i aint done yet i lbsee yall prayin that a g
first lady sell a little misty here tonight this is the label that pays me unfadeable so
casino like bugsy siegel and do it moving real slick with it pass it to the
israeli machettes is cris swish some baileys snuggled
your service produced by metro boomin it was but it wont stop until the other side
youve got the
bitches in heat we tear the beat treat rap like cali weed
yet think shit over hell naw i aint finna just lie real recognize
one for the next goarounds a bitch to me shit dont
live with five shots niggas is hard to kill on my back
and g the key to me i bust and flee these niggas must be
side its california love this california bud got a red

Artist: Young the Giant, n-gram Order: 2

with the beat of my eyes you
silence you carved a boat to sail my shadow now i
love without leaving at all oh im coming up
the weight of your crimes but when the sun gets carried the
nobody calls anymore talking is dead with
in your kiss oh alls divine in desire with
break ill be shakin it cause im a young man
me from my disaster my disaster its heavy
blow pistols pistols pistols pistols lights
will stars align im thinking that its
riding down to the sunrise cause i want

cool kid hey is it my fault just

quits cold hard stare i wont quit cause i want

alright i hit the sidewalk and this is

head cause these days im awaken ive been thinking im

a riddle not a game of dice not for a sign of

how it starts hide in a slipstream oh

we built still here is the house we built still

and breathe everythings in its obvious to climb in

change i missed that train new york city it

road trick up your heart beats just like i wanted it to

a young man built to fall i missed that train new york

sit quiet inside youre so warm your

im in a dream and the rains begin

like its gon na break this time

baby ive been thinking im not feeling anything

soviet cries in the wild

happier i could find a way to see this straight id run

dice im feeling like its gon na

ride sherry ride sherry ride i will win it all

8.2 B: Recurrent Neural Network Lyrics

Artist: Taylor Swift, Training Epochs: 100

cause baby now weve got bad blood hey

follow you remember you and to know clothes

hes its bad and he as room

cause baby now weve got bad blood hey

cause baby now weve got bad blood hey

bullet in live is cheeks say youll see to in

and baby tall nice deep wounds last kiss

hes so og but handsome as done

remember drove make is cheeks more at see

and baby dont think hindsight familiar my iraq

cause baby now weve got bad blood

you know with in but be my clothes

amnesia in with youve cheeks say whyd the are replace

it forgive forever had mad your at the sad replace

hes so of and handsome as hell his

so lips a look dress say at the lasts replace

and baby got nice city very at kiss

and i voice incredible a your at nothing sunset babe

you so tall in to as hell so sunset overrate

so take a look what youve done

and baby now think our youve from iraq

you so me in in as it so we do

and its weve is cheeks more at kiss sunset iraq voice

cause baby now weve got bad blood town

so this a look scars staring done

you say of just hindsight your what love you again

so take a look what youve done

so take a look what youve done

producer you live you and your youll what his again

cause baby now weve got bad blood

Artist: Katy Perry, Training Epochs: 150

its a yes or a no

you were ta beg for it

i dont need a pretty poet

you aint got ta wine and dine me no

got me bow them my panties because my ass

let me see on your face

i dont need a pretty poet

if you choose to walk please

just etc ass at a cage

you got ta beg for it beg for it

you aint got ta wine and dine me no

so treasure wan sure a like you

i wan laughing see you lookin up

baby ima need you to beg for it

i dont need a pretty poet

boy you should a of away

but love will a the youre

you aint got ta wine and dine me no

i dont will a dozen roses

let me sit on your face

you got ta beg for it beg for it

make me your enemy walk position

you got gon to this dr is

its non the or a no

i wan na see you lookin up
so you make know mistake youre magic
theres you gooder a what position and for
its before yes palm of your hand now baby
but you make a all im
life walk will without you i

8.3 C: Character-Level n-gram Lyrics

Artist: Fetty Wap, n-gram Order: 7

at 56 a gram 5 a 100 grams though girl you a baaaaaaaah baby this is something fast 200 all on my seats shit i kills the shoots and i cant even lie shawty you my kind of lady so what these niggas beggin but time baby ay where the gold at baby and i want you to be mine again baby i keep my pussy in my whip with a cover low thread count up benjimins mins then leave her with that just speak on real she sing along like and thats just ask what the price of them bitches you cant go remy boyz got this sewed up remy boyz got the same time she loves a nigga living like a water slide when shell be mine again baby im the leader hits be coming right im tryna finish when i get high with my baby yeah i let you comin home with piranhas bitch he a bit what you to be mine again baby i set em down to the mansion and comparing it to you ra ra round baby i love these knots in my benz benz i skirt off imma start to wonder when they scream for my girl you the times you can stay the same time i see you and i

Artist: 2Pac, n-gram Order: 9

touch play the rules i shed tears but i dont cry dry your eyes swirl you doin your job

every day gotta roll on and then yaknahmsayin and theres no limit to the brothers or the sisters on welfare 2pac cares if dont nobody else care and i know the rules little accidentmurderer and i realize the precious time that bitch niggas be they want from us motherfuckers catchin cases bitches storm cause niggas knockin up the same ho i make a promise if you got bills to pay nigga go all out we get out they gon feel it we the realest is we bring on horror like tales from the vallejo where sellin narcotics is all i know yall got me under surveillance huh all eyes on me heeyyy to my niggas wanna be involved see i was out there now do you wanna ride or die lalalalalalalalalalalala they got money for war let my balls hang bust it i took apart your whole style when i was sick as a little peace its war on the street slugger my heat seeks suckers started smackin talkin that slack and till he see my