# Recurrent Generation of Lyrics for "4 Chord Songs"

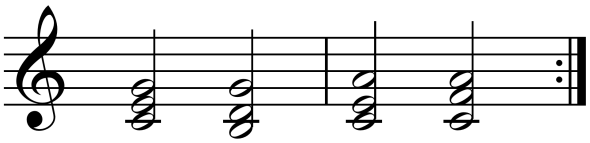
**Aditya Parmar**
Northeastern University
360 Huntington Avenue
Boston, Massachusetts 02115

### Abstract

The purpose of this project was to attempt to create an ai interface which can procedurally generate lyrics for a particular subset of songs. The structure I am using is based upon a traditional LSTM (long short term memory) model recurrent network. Additionally, I'm implementing one-hot encoding for characters. The particular implementation I'm using involves a user provided seed to act as a driver for lyrics prediction. Overall, the lyrics that are generated are nonsense, but do seem to have some semblance of meaning in them.

## Introduction

Nowadays, thanks to the popularization of public radio and audio media distribution, the music industry is more prosperous than ever. They have earned it, as it seems that these days people quickly get addicted to hearing catchy music and can always count on the radio to provide them, and every few weeks, we're provided new music to dig in to as much as we'd like. But just how new is that music? You may be surprised (or not) to learn that an overwhelming majority of songs on the radio have the exact same chord structure. Yes indeed, the secret to your eternal captivation is in right here.



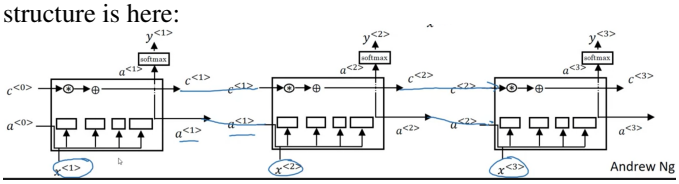
What you are looking at is known as the "I V vi IV". A potent combination of major and minor chords in harmony. It rose in popularity ever since the 1960s. A now popular youtube band known as "Axis of Awesome" released a medley of such "I V vi IV" songs called "4 chords" (which is why, for brevity, I am calling them 4 chord songs henceforth). Now that the ubiquity of this chord progression in modern music has been shown, here is where I come in. I made a hypothesis that if the chord progression is given to us, all we need to do to make a popular pop beat is to come up with the right lyrics. With this in mind, I thought, what better a source than other 4 chord songs, of which there is a large list of documented examples. In this project, I endeavor to attempt this by means of an ai network that should procedurally generate lyrics for me. After looking at several possible solutions, I eventually landed on the LSTM network; Long Short Term Memory. This is the same type of network employed for a variety of text prediction software and a couple other auto-generation related projects. In order to keep my project fresh, rather than just implementing the LSTM network described on the internet and plugging in my data, I decided to add the element of being able to provide your very own seeds to drive the prediction. In addition, my model will work on a character-by-character basis, as opposed to a word-by-word basis. The reason for this is that in addition to being easier to code, I found that the results are actually more coherent than a word based model, possibly due to the variety of different lyrics sources I am drawing information from. Additionally, in order to implement this, I will be using Keras, as its api is quite simpler than TensorFlow, which I am still trying to learn. In order to gather the necessary data, I will be using BeautifulSoup and Pandas, both of which are commonly used for web scraping and data entry.



## Background

As stated earlier, this approach will be based off of the LSTM recurrent layer. This is well documented on wikipedia and other resources but a brief description is this. First off, LSTM is typically used for prediction and classification of data in the form of a time series. Other typical applications include handwriting recognition, semantic parsing, and rhythm detection. They were developed to deal with datasets which feature gradients that rise and fall extremely quickly. In dissecting its name, the "long term" refers to the network's ability to recall data from an arbitrary length of time. the "short term" refers to the ability of the network to "forget" after a period of time, which can help stabilize the previously mentioned rising and falling gradients, which would typically throw off a traditional recurrent neural network. The fact that this type of network can remember data from long periods of time is actually what makes it so powerful and is why it's adopted by so many technologies today. The vanishing gradient mentioned before has been a problem for a while in the deep learning community and LSTM was one of the solutions that sprung up. The training of LSTM's involves backpropogation through time. The specific architecture of LSTM I am using

is the one that Keras uses, which is the kind that Hochreiter and Schmidhuber described in 1997. An overview of the structure is here:




This image was taken from Andrew Ng's deep learning coursera course.

Apart from LSTMs, my model is relatively simple, with the network itself only composing of 3 layers, one of those being LSTM. The other layers are a "Dense" layer, which is Keras' implementation of a typical densely connected RNN node, and a "softmax" activation layer.

Additionally, my algorithm employs "one-hot" encoding, which is a way to turn categorical variables, which, in my case, are all the letters and symbols contained in the song lyrics, into a structure that is more suited to machine learning algorithms. In a nutshell, the reason we need this is because it doesn't make sense to just encode each character to a number like 0,1,2 because there's not actually any sense of ordering between letters. Instead, we convert each character into a binary column with a length of the number of characters we're encoding. We then set all values to zero except for one, which represents the character we are looking at (hence, one-hot encoding).

Lastly, my project involves web scraping, which is a way of programming a call to a webpage and extracting its contents. In this project, I have done so in a very controlled way, to minimize service overload to the website.

## Related Work

As mentioned before, the crux of this project is the LSTM layer of the neural network that provides the necessary ability to predict values. The LSTM is not, however, the only candidate for this job. In the most basic sense, my job could be accomplished through a simple recurrent neural net, however the results would not be optimal at all due to the above mentioned vanishing gradient problem. Then, the main contenders for the project algorithm were LSTM and GRU. GRU, Gated Recurrent Unit, is a brand new type of recurrent layer which works in a similar way to LSTM, but is lacking a memory unit; the content that was previously hidden by LSTMs "forget" mechanism is on full display, but operated on in a different way. All in all, it's been shown that the results of GRU and LSTM can be about identical, but LSTM has the advantage of producing slightly better results for larger datasets. Aside from that, both LSTM and GRU have a number of variants such as LSTMCell, ConvLSTM2D, CuDNNLSTM, and even more if you consider all the tensorflow classes, I decided to stick with the simple LSTM layer because it seemed to satisfy all my needs and was the simplest to work with.

## Project Description

**Part 1: Gathering the Data** The first thing I would need was the data itself in a format that can easily be fed into a machine learning algorithm. I decided to use wikipedia's list of 4 chord songs, as it was the largest and most parsing friendly, as all the data was in a table. From there, BeautifulSoup was employed to gather the titles and artists of the songs and put them all into a csv. From there, I employed pandas to read that csv and made calls to MetroLyrics.com to scrape the individual lyrics. The reason I used metrolyrics was due to it having an immense amount of songs and also having a very simple url naming format, which I was able to guess, given the artist and title of a song. All the lyrics were then put into another csv. Once I acquired all the lyrics that I could, I began the one hot encoding process. This involved with creating 2 dictionaries, one which mapped characters to indices, and one with mapped indices to characters (for reverse lookup later on).

**Part 2: Training Data Generation** In order to train the model, I needed to come up with a ton of possible sequences to work with. I decided that since I had a ton of lyrics to work with, this could be done with a simple traversal of the concatenation of all lyrics and extraction in the following manner:

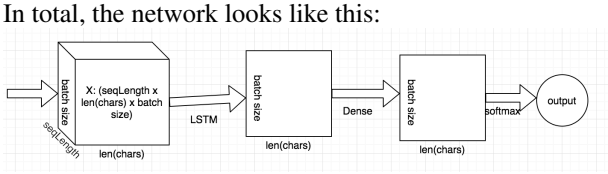
```
seqLength = 30

sequences = []
predicted = []

textLen = len(lyrics)

for i in range(0, textLen - seqLength, 1):
    sequences.append(lyrics[i:i+seqLength])
    predicted.append(lyrics[i+seqLength])
```

seqLength: this is the length of the seed sequence and thus the length of each training example, 30
sequences: a list which will contain all the data that could possibly have been seen previously by the algorithm (obviously, not an exhaustive list, but given that there will eventually be nearly as many sequences as the combined length of all lyrics, it seemed sufficient)
predicted: a list which will contain all of the "next" characters, according to the ordering of the lyrics
textLen: The total length of all lyrics combined
gll (not shown): this is the concatenation of all lyrics. In the code, it is known as giantLyricsList

The contents of sequences and predicted will then serve as part of our x and y for the model, when combined with the respective one hot encoding method.

**Part 3: The Model** The model is a simple 3 layer neural network, also described above. The first layer is the LSTM layer. Almost all inputs are as they are by default, so all I had to describe was the input shape, which is a matrix with dimensions seqLength and the number of unique characters. This allows us to deal with all situations and all types of character prediction. The output of this layer is then fed into

a standard densely connected layer. This densely connected layer also has most settings set to default, and outputs a number of units equivalent to the total size of the alphabet. Finally, these units are driven through a softmax activation layer, the traditional final choice to a neural net, which applies the logistic function to properly classify our variables. In total, the network looks like this:

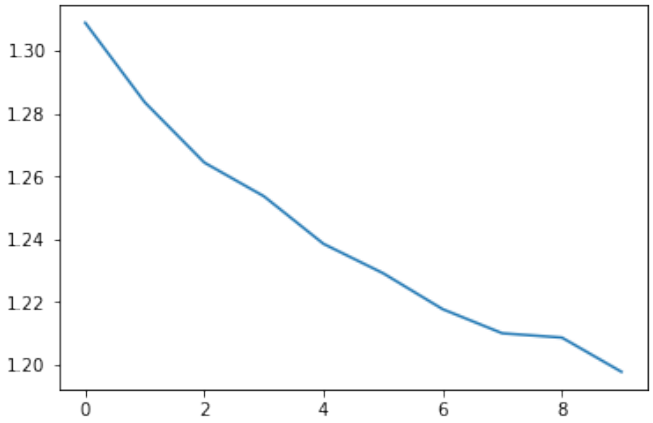



I finally trained the data with a batch size of 128, with 10 epochs.

**Part 4: Generating Output** After the model is trained, the output generation can finally begin. A call to the user is made to input a seed for the program. The seed is then used to generate one character. Next, the first character of the seed is chopped off, the generated character appended, and this becomes the new seed. I repeat this process 360 times, the number of characters I wanted to generate.

## Experiments

This project was not without its failures. I initially had a word-by-word model, which would output a word given an input. Identifying input by words was a bit of a hassle, but the real issue was that the output was simply not good enough. I also had not much of a clue how to make proper encoding for it. Even with 10 epochs I had a very high loss and the output frequently had loops of ”the” or would fail to return any words at all for some non english inputs or unrecognized words. I decided to scrap that and work with the character model instead. This ended up being much more successful.




The model achieved a loss of about 1.2 after 10 epochs. Here is a demonstration of the improvement of the lyrics for the seed ”i am a big big dog and i am in”:

### 1 epoch

i am a big big dog and i am in the the so the the the dond to
the sing the sout you not in the so the the sour the the sing
the wing the sond to to the to the so the so me to the the so
the sous the wand i me the cand the sout the sout to sand i
me the the the the the me stong
the the so the the to the to the some
i me to the the me the wond you the the thethe the the to the
sout the th

As you can see, there are still many ”the” loops and nonsense words.

### 5 epochs

i am a big big dog and i am in the say
when you can’t be the seart
it all the way you feel the rear you storing for your better
i see i want the say
when i want to be the way i’m the way
i wanna do you to can the brow
i can’t want the way you and me
i’mto the way i’m like you all and you stand
the way i wanna the way you to the back
in the say of the way of the words are the way the rea

A more structured view is becoming clear. Less nonsense words and some actual rhyming at the ends of lines.

### 10 epochs

i am a big big dog and i am in the carr
i was here i see
you got me in your eyes
i was here that i say
i was so think i’ll be good to be alone
you and i want to be forever
i was hore to see
you want to be a mata too mood back and me
that i don’t wanna be a shore
what i see
you and i was so show it starts
i don’t have to take you to be you reallyed for you
i’m stand the world wondere
want

The final result.

Another interesting outcome is that the character level model works with all types of inputs, including something as useless as 30 0s consecutively, which it seems to think is spanish:

### Seed: 000000000000000000000000000000

000000000000000000000000000000xxxxi desmasia
don besta mos tu mi vama
ye bay bay ta bos ma musa tum ata
mas a no pi ma ma
sa bai tu musa nos mu mas tu mu mba mama
y sa no mu mamama

sa ba ba bacra moravavando mombebe de mos ta pose to
sas res y es meste take ma heres a sando
ma be tu masa to
s mi te tu mila da tecata
sa no mi tu tucata tacata
na la ntacatatcrad
to manda tala tale to tale f

I don't think all of those are real words, but it does seem to pose some sort of structure regardless.

It is also worth noting that prior to adding the final softmax layer, my model had a significantly higher loss and there were more long words and nonsense words.

## Conclusion

In the end, my model did not seem to produce the quality of lyrics that could be used for a proper song. The results were, however, quite interesting in that the beginnings of an ai that can accomplish this goal seem visible. Perhaps with a more focused dataset, such as a particular artist's songs, a more realistic result could be produced, as the songs in this list were taken from all genres and between 1960 to 2018. It's also possible that my data wasn't big enough, at just about 200 songs, while most professional AI, such as the one that writes screenplays, has hundreds of thousands of training data. Additionally, my model only consists of 3 basic layers, while most professional models have several times that many. All facts considered, I am satisfied with the output and that I could output anything at all. Through this project, I learned more than I thought I would about LSTMs and their history and application and also about machine learning frameworks in general, which are fantastically well documented. Designing one's own neural network as opposed to following instructions and demos is truly quite remarkable. While my model was simple I really wonder how people come up with models consisting of 50+ layers. I do plan to touch up on this project in the future, possibly implementing GRU instead of LSTM and seeing how that goes, and probably also adding several more intermittent hidden layers.