**Lyrics Generator**

**NLP Final Project Report and Ethical Statement**

Veronica Child, Maryam Hedayati, Sasha Mayn, and Chelsea Ying

November 20, 2017

**Ethical Statement**

While our generated lyrics are currently often silly and nonsensical, with improvement our lyrics generator has the potential to create lyrics that could seem as if a human wrote them. The ability to generate lyrics close to human-likeness has many benefits and ethical considerations around accountability, uniqueness, and what it means to “be human.”

Our generator could begin a golden age of song lyrics. If we trained our generated on all the best songs and or top hits ever produced, our generator would theoretically learn what constitutes a “good” song. It would then have the capability of creating thousands of new song lyrics based on “good” songs and invent the next greatest hit that humans may never have thought of. The training process of our generator could easily be abused though depending on what it trains on. The quality of the training lyrics greatly influences the quality of our generator’s output. For example, if our generator trained on lyrics data filled with negative gender roles or identities, then it could easily perpetuate the existing biases found in many of today’s pop lyrics on a highly scalable level and greatly add to the problem of negative stereotypes.

Our generator has the potential to make a musician’s life easier. A song requires a lot of components, from rhythm, to vocals, to instruments, and lyrics are just a single part of a complex process. Having a lyrics generator that creates the ideal lyrics for a song removes one step of the song creation process, allowing artists to focus on other components of the song they might be more interested in. However, there is the potential for lyricists to lose their jobs if our generator is good enough, thus resulting in added unemployment. A lyrics generator could potentially govern the song creation process by changing how artists think. Artists may alter their creative processes to fit the generated lyrics instead of using the generator to fit their own needs. Furthermore, there would be the question of accountability of a song with generated lyrics; who should take the credit of a song’s success or flop if it used generated lyrics? Lastly, a generator would contribute to the mechanization of the music industry. Society can already generate musical accompaniment with digital audio workstations and enhance or mimic human vocals with audio-altering and text-to-speech technology. Having a lyrics generator may be one of the last steps in total mechanization and automation of the music industry, thereby entirely removing the human touch out of music.

To avoid some of the ethical dilemmas, we would implement certain safeguards and restrictions on our generator before releasing it. Within our code, we would implement checks to make sure no one particular song is being weighed. We would also require a licensing based on training such that users of our generator would only train on song lyrics that agree to be a part of our database. Our database would be open to the public to view to see the exact songs our generator is training off of. We would also require for each song that uses our generator’s outputted lyrics to be labeled in some way to differentiate when the lyrics were generated and when they were developed by humans to ensure a “human-made” feeling.

**Project Report**

**Research Question/Statement**

The purpose of our project is to build a system that generates plausible customized pop lyrics using a number of NLP tools like a POS-tagger, templates, context free grammar, and a Tensorflow neural sequence-to-sequence model. We wanted to explore the possible ways of combining these very different tools, learn more about automatic language generation, and create a collection of generated lyrics that could be seen as potentially human in likeness. Our research question is what approaches can generate plausible, relatively high quality pop song lyrics, and which approach will yield the best results? To answer this question, we surveyed three different approaches: templates, CFG and ngram model, and neural language generation system.

**Background Information**

Our data comes primarly from a lyrics dataset available on github (please see<https://github.com/walkerkq/musiclyrics/blob/master/billboard_lyrics_1964-2015.csv>). It is a collection of over 50 years of pop music in a .csv file. Regarding notable issues, there are “fair use” issues for using song lyrics in work if the artist is not cited. However, since we are not displaying an artist’s exact lyrics for a song but are instead compiling lyrics, we are not subject to “fair use” and copyright issues.

Our project uses multiple tools and libraries. The most notable modules are TensorFlow and NLTK. The RNN component of our project uses TensorFlow (please see<https://www.tensorflow.org/tutorials/recurrent#language_modeling>) to build a neural sequence-to-sequence language model that generates “song-like things” from a few POS-tagged content words (basically a translation system). In particular, we use seq2seq, a sequence to sequence learning module for TensorFlow (please see <https://github.com/google/seq2seq>), to train our language model on the tokenized lyrics data. Our CFG and template components use NLTK (<http://www.nltk.org/>) to easily parse and tag text as well as provide access to a wide variety of lexicons.

**Methodology - Template Approach**

Our template approach is arguably the most structured of the three approaches since it is the least dependent on computer generation. For our templates, we based the component off of mad-libs, a phrasal template word game where one player prompts others for a list of words to substitute for blanks in a story before reading the - often comical or nonsensical - story aloud.

Similarly, our template program maps user input to a specific spot in the hard-coded template. When run, our template requests user input. The input is checked to make sure it’s valid (i.e. only alphanumeric characters), and then it is given a similarity score against a happy or sad sentiment. Depending on how similar the collective input is, a template is randomly chosen from either the happy or sad genre. The program then calculates the mapping of the user input to the template. If no enough input is available to fill the needed number of blanks in the template, the system uses the WordNet package of NLTK’s corpus to access words similar to the user’s input through ‘synsets’, which are sets of synonyms. The completed lyric is then printed onto the screen for the user to enjoy.

**Methodology - CFG Approach**

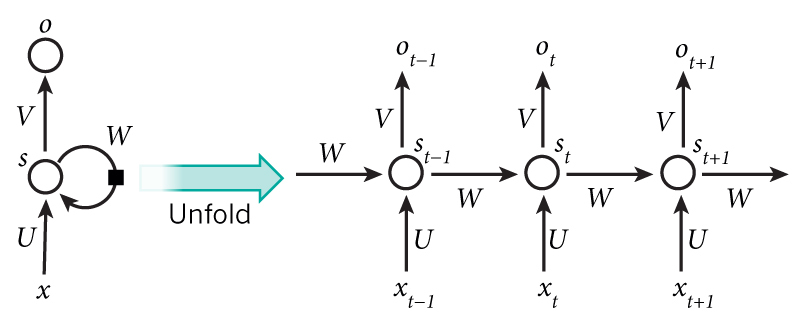
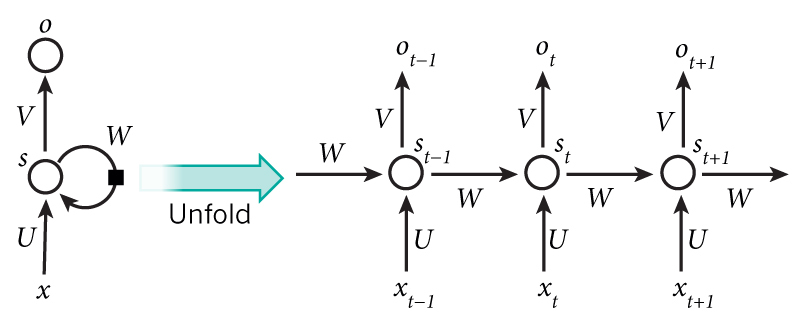
For CFG approach, we used CFG rules and the lyrics dataset to generate lyrics. The dataset that we used required some preprocessing: the text of each song was a string with no line breaks, punctuation was often missing and a lot of the words were stuck together. The first thing we did was break the lyrics into lines of 7 words each (the approximate length of a pop lyric) and then wrote a function to separate the words that were struck together by checking possible groupings of two words.

Then we POS-tagged the whole corpus and added all words to a dictionary of terminals. Later we pull 10 words per part of speech from the dictionary for our CFG to use as terminals. This resulted in pretty ungrammatical sentences because the POS-tagger is not foolproof and got some of the classification wrong; also, our rules did not take tense, person and number agreement into account (that would have been hard to do since the POS-tagger did not distinguish between those categories). So we decided to augment the CFG with a 4-gram model that would score and pick out the most grammatical sentences. For the 4-grams we used 90% of the lyrics data - the remaining 10 being used as vocabulary for the CFG - and a corpus of 1 million 4-grams from the Brigham Young University Ngram Corpus. We were hoping that this way we would get some of the word order and vocabulary that is specific to pop lyrics, as well as some higher quality 4-grams that would perhaps be out-of-domain. We let the CFG generate a large number of lyrics (say, 500 thousand or 1 million) and then randomly pick from the 1000 with the highest probability. We also implemented very rudimentary postprocessing of generated lyrics to account for agreement of pronouns (e.g., “I” becomes “me” when following a verb).

**Methodology - Recurrent Neural Net Approach**

The idea behind RNNs is to make use of sequential information in lyrics in generation. In a traditional neural network, we assume that all inputs (and outputs) are independent of each other. But for many tasks, such as text generation, that is not a good approach since the appearances of words depend a lot on contextual information. If you want to predict the next word in a sentence you better know which words came before it.

RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Another way to think about RNNs is that they have a “memory”, denoted by St in Figure 1, which captures information about what has been calculated so far. In theory, RNNs can make use of information in arbitrarily long sequences, but in practice they are limited to looking back only a few steps.



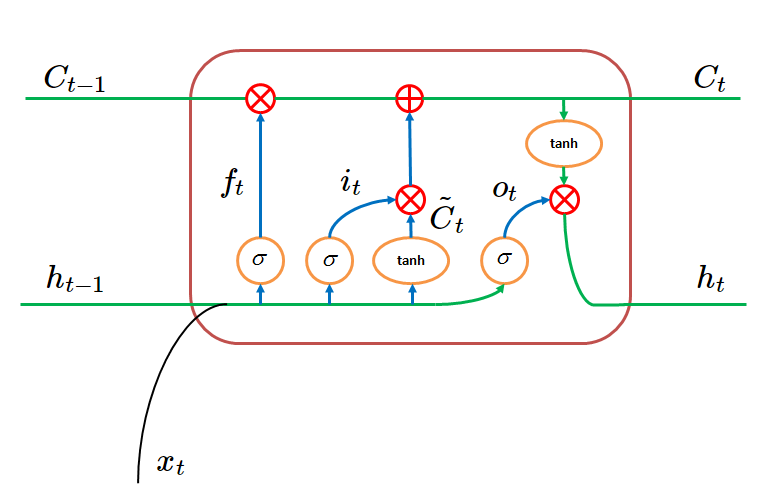
RNN unfold into

traditional neural network

==============>

**Figure 1**: Illustration for recurrent neural network.

To solve the limitation in sequence memorization for basic RNN, we selected a special twist to RNN, Long Short Term Memory cells, capable of learning long-term dependencies and yield better generation result. Networks with LSTM cells also have the form of a chain of repeating modules that RNN has, but instead of having simple structure in its repeating modules (one layer of activation function), there are four layers in its module, giving the network the ability to “forget”, which mean it can decide whether to forget the previous hidden state.

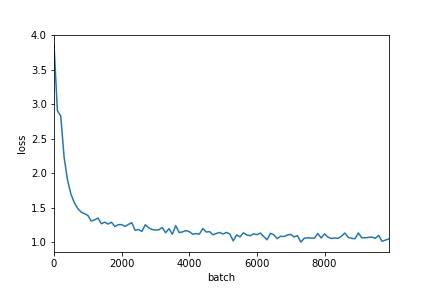


**Figure 2**: Illustration for one repeating module in Long Short Term Memory Network

The special forget layer (ft) looks at the hidden layer from previous step (ht-1) and input (xt), and outputs a number between 0 and 1 for each result in the cell state from the previous step (Ct-1). A 1 represents “completely keep this” while a 0 represents “completely get rid of this”. Then it updates the current cell state (Ct) and hidden layer (ht) accordingly to forward the calculation.

In terms of the implementation, we adapted the open-source character sequence to character sequence RNN text generation model from [spiglerg](https://github.com/spiglerg/RNN_Text_Generation_Tensorflow)’s github repository. The model uses TensorFlow to build up an RNN with 2 LSTM layers, 256 LSTM cells on each layer, and linear activation function for its output.

We used lyrics from the billboard dataset to constructed a vocabulary of unique characters from the lyrics. The size of the vocabulary will also be the input and output size of our RNN. Then we concatenated all lyrics from the dataset and embedded the character sequence to vocabulary using one-hot encoding. To train the network, we fed character sequence from the lyrics and use the remainder of the sequence as expected output for supervised learning. Softmax\_cross\_entropy\_with\_logits is used to calculate loss and back propagate. We trained the model for 10,000 batches with a batch size of 64 to reach a final loss of 1.05.



**Figure 3**: Measured loss at each batch for training the RNN generation network

Once the model is trained, it can generate character sequences in the style of the training lyrics with any specified length given a seed input, a few starting characters for the lyrics the user wants. Since the training lyrics dataset we obtained doesn’t have any line breaks, the generated sequence will also not contain line breaks, so line breaks are manually added as a part of postprocessing.

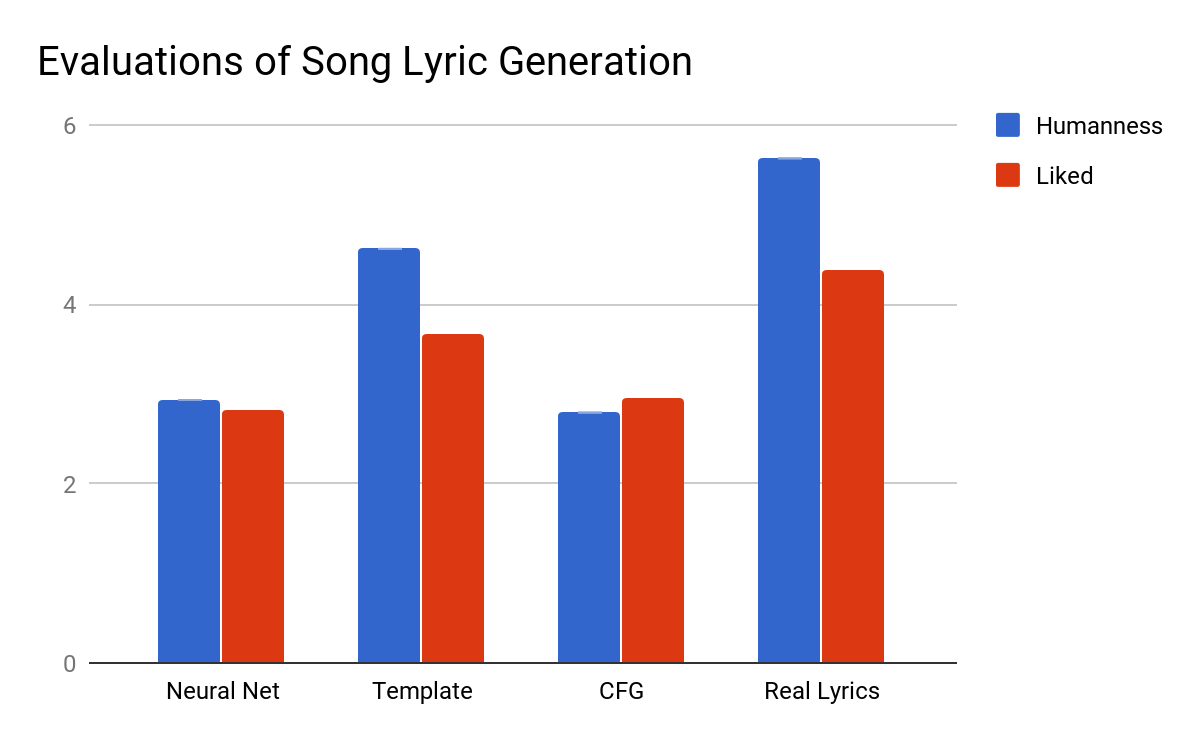
**Evaluation of Results**

In order to evaluate the quality of our results, we conducted an informal survey (linked [here](https://goo.gl/forms/B3TRl5Jnrl0LdX3n1)), in which people were given excerpts of lyrics generated via the different methods. They were asked to rate each lyric excerpt on a scale of 1 to 7 based on its “human-ness”, and how much they liked the excerpt (regardless of how human or not it seemed). There were 10 lyric excerpts: 2 from real songs, 2 generated using the template approach, 4 generated using the neural net approach, and 2 generated using the CFG approach. Our survey was completed by 55 participants. Table 1 below contains the mean results for each lyric excerpt, and Figure 4 shows the mean scores for each generation strategy. As expected, real song lyrics score the highest on how “human” they seem, as well as on how much they are liked. Templates scored second highest on both categories, which is also unsurprising given that they are primarily written by humans. The neural net and CFG approach scored similarly, although when looking at the individual sample breakdown, two of the neural net samples scored extremely poorly, while the other two scored much higher. Thus, there seems to be more variance in the quality of Neural Net based generations.

This evaluation strategy is very informal, and does not necessarily provide an accurate representation of each of these methods, given that each of the samples was hand picked, and each participant received the same samples in the same order. In order to improve the evaluation, we would include the same number of samples from each strategy, randomize the order, and ideally have our system generate a new sample for each participant. However, given our limited time and resources, we were not able to create a survey with this level of complexity.

**Table 1**: Mean scores for each lyrics sample

|  |  |  |
| --- | --- | --- |
| Type | Humanness | Liked |
| Neural Net | 1.56 | 2.27 |
| Neural Net | 1.6 | 2.18 |
| Neural Net | 4.51 | 3.56 |
| Neural Net | 4.05 | 3.27 |
| Template | 4.73 | 3.45 |
| Template | 4.51 | 3.89 |
| CFG | 2.8 | 2.76 |
| CFG | 2.78 | 3.13 |
| Real Lyrics | 4.67 | 4.16 |
| Real Lyrics | 6.58 | 4.62 |

**Figure 4**: Mean scores for each generation strategy

**Table 2**: Mean and standard deviation for each generation strategy

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Humanness | Liked | Humanness Std | Liked std |
| **Neural Net** | 2.931818182 | 2.822727273 | 1.570201686 | 0.698140387 |
| **Template** | 4.618181818 | 3.672727273 | 0.1555634919 | 0.3111269837 |
| **CFG** | 2.790909091 | 2.945454545 | 0.01414213562 | 0.261629509 |
| **Real Lyrics** | 5.627272727 | 4.390909091 | 1.350573952 | 0.3252691193 |

**Conclusion**

In this project we explored three different approaches to generating pop lyrics: templates with user input, CFG, and Neural Net. As shown in Figure 4, real lyrics were still consistently ranked higher on both humanness and likability (mean score of about 5.5 and 4.5 respectively), which means that none of these approaches as they are is quite advanced enough to rival human creativity and mastery of language.

On the bright side, the mean scores for all three strategies were significantly higher than 1 (about 2.75 for Neural Net and CFG and 4.5 for templates), which means that the approaches are able to generate data that at least partially resembles natural language; this is promising and with more postprocessing and taking grammar into account (and, perhaps, a better training dataset), results that are much closer to human-written lyrics would be possible. As shown in Table 2, Neural Net also has high standard deviations in both Humanness and Liked scores. Although with a relatively low mean, Neural Net produces results with more extreme ratings. So with more postprocessing and hand picking the better ones, it will be able to yield similar results as the templative approach without requiring any human input in generation process.