AI Project 3: Recurrent Neural Network Song Generator
Members: Trent Winger, Adam Weaver, and Kyle VanDenHeuvel

The goal of this project was to train a recurrent neural network to write song lyrics by acquiring song lyrics and utilizing a neural network built with TensorFlow to read the lyrics character-by-character. To achieve this, we first built a tool that utilizes the API of both Spotify and Genius. First, the user creates a Spotify playlist containing all of the tracks with which they wish to train the model. Next, the Spotify URL is given as input to a script that gathers both the track title and artist, creates a pairing out of the two elements, and searches for the corresponding track on Genius. With Genius' API, we are able to web scrape the lyrics and put them into a text file. In this process, however, we encountered one problem: noisy data. At times, the web scraping done by the API would gather nonsensical data and place it into the text document. Fortunately, scrolling through the text document allowed for us to easily "clean" the data, but in cases where we did not, such as Trial 5, the output was meaningless due to the unclean data.

Following the data-gathering step, we then pass the data to a recurrent neural network (RNN) to train a model. In order to do this, we used TensorFlow. This portion of the project was facilitated by TensorFlow's tutorial on character-based text generation (https://www.tensorflow.org/tutorials/text/text_generation), which provides the necessary code to achieve that which we set out to do. Though most of this code was provided for us, we did make one key change to force the model to finish its current line of text before completing its generation step. Initially, the code provided by TensorFlow stops text generation after a certain number of characters, regardless of the previous character. Since we wanted to avoid word or line fragments, we decided to add to the text generation function to force it to finish its line. We attained this result by modifying the code to check that it has either finished a word and reached a new-line character, and only end text generation on a new line or after doubling the initial number of characters. In doing so, the final line of the song does not end abruptly, much like many songs one may hear.

After the core components of this project were working, we began to experiment with three main features: epoch count, temperature value, and most importantly, input data. Changing the number of epochs resulted in mundane outcomes; a higher number of epochs resulted in highly-trained data, and a lower number of epochs resulted in barely-trained data. Examples of this include Trial 6, in which the model manages to write lines that make grammatical sense after 25 epochs of training, though nonsensical lines were still occasionally created. For example, the phrase "I got a ais canding a suard[Verse 1]". Intuitively, one would imagine that a higher count of epochs would result in "better" lyrics. However, this resulted in over-training the model to the point that it would completely copy lines or entire verses from the training data. As the amount of training data was increased, though, this became less of an issue when compared to datasets with lesser amounts of data when trained for a similar number of epochs. As a result, we struggled to find a "sweet spot" for creating coherent, yet unplagiarized song lyrics.

The temperature value was another parameter with which we experimented. This value is not used in the training of the model, but can vastly alter the generated text. As a default, the

temperature value was a 1.0. Lowering this value resulted in more predictable text to be generated, and higher values resulted in more dynamic text. In Trial 10, we trained a single model, and generated 15 different songs at 3 different temperature values: 1.0, 2.0, and .5. Reviewing the results indicates that this variable has significant impact on the text generated; songs made with a temperature value of 2.0 were so unpredictable that it was unclear an attempt at cohesion was even made. Songs generated with a temperature of .5 resulted in very grammatically-coherent lyrics, though this caused a repeat of words and a neural network that was much more enthusiastic about plagiarism. For example, one of the songs generated like this resulted in the word “no” being repeated for over 10 lines. With the results of both temperature values in mind, we deemed it sufficient to only use values close to 1.0 in order to both maintain cohesion and avoid lyric-stealing.

Last, but most certainly not least, was the starting data. In this case, starting data refers to the playlist (or playlists) used to train the model. In order to experiment with output, we examined collections of songs from different genres and at times mashed two genres together. Hip-Hop songs, for example, resulted in many explicatives and aggressive words. Country songs, as expected, made mentions of typical themes such as beer, bonfires, and trucks. Aside from country music, the playlists utilized were collections of songs that our group members were fans of, which made plagiarism identification much easier. As expected, albeit disappointing, songs did not make much semantic sense nor exceedingly-literate grammatical sense. At times, however, song lyrics generated humorous, nonsensical lines of text, such as “I’ve been swimming off you”, and “We counter anowhere, but I ball a lot (Ayy).” Mashing up genres simply results in a mix of words/short phrases found commonly in both songs, though that is also unsurprising given that common words and phrases are statistically more likely to show up in output data given how neural networks in general function. In addition to different genres, we briefly experimented with other languages such as Spanish as two of our group members are able to speak it in varying degrees. This experiment, however, did not provide much new insight into how RNNs work, as the results of either being plagiarized or nonsensical also appeared in these trials. In a similar vein, we also tested the effects of the starting string, generating multiple songs using the same model and starting string. These experiments resulted in completely different songs each time, despite the same model and starting string, proving that the RNN uses the starting string as a base rather than determining factor of the song’s contents akin to a “seed”.

Overall, our group is happy with the results of this project. Even if we will not be able to algorithmically create songs in order to turn a profit, we were still able to implement a random text generation to create songs. We explored a number of factors that influence the behavior of the RNN and the resulting songs. After training for as little as 25 epochs, our models were able to consistently match the formatting of the songs in our data while generating songs one character at a time. And although we struggled to keep our models from plagiarizing lines or verses, we were still impressed with their ability to learn the words, phrases, and formatting inherent to songs and reproduce them.