Project Title: Music Generation with LSTM Neural Network
Student Name: Matthew Gilmer
Mentor Name: Srdjan Santic

LSTMs (Long Short-Term Memory Neural Networks) are a special type of RNN (Recurrent Neural Network), a class of artificial neural networks with memory storage capabilities that allow them to take sequential data as input. LSTMs are even better suited than traditional RNNs when the gaps between important events in the sequence are variable. This is accomplished by a memory cell with three control gates (input, output, and forget) which is better suited to retaining the important information from previously processed events in the sequence. The basic task of an LSTM is to predict the next event in the sequence. One interesting open question is: Can LSTMs be used to generate realistic melodies or pieces of music in the style of the music it has been trained on? One particular attempt at such a LSTM, trained on the music of Johann Sebastian Bach, has sparked the genesis of this project (Hewahi, AlSaigal, and AlJanahi 2019, link).

Their model, which they arrived at after tuning several hyperparameters, consists of four LSTM layers of 512 cells each followed by three 'dense' layers of 256 neurons each. A dropout value of 0.6 was used after each layer. Their window size was 16, meaning each training and testing input was a sequence of 16 input vectors. Each input vector consisted of a mostly boolean vector where each column represented one of the keys on a piano. A 1, indicated that the key was pressed while a 0 indicated it was not. The final column in the vector was a float indicating the duration of the vector. This was an added level of complexity over another similar work (Skuli 2017, link). The F-measures for training and testing were 0.83 and 0.71, respectively, which is reasonably good. However, the generated pieces may have suffered in quality due to the aforementioned added complexity. The authors note that the generated pieces lacked 'musical clarity'. Additionally, the style of Bach learned by this model was not as true to Bach's musical style as the BachBot model (Liang 2016, link). It is evident from these results that LSTMs have not yet been successful in generating realistic melodies or pieces of music.

For this project, I have elected to use a dataset from Kaggle of classical music. It has separate folders for different composers giving me the freedom to choose any subset of these I wish. For folders sufficiently large, as may be the case for Chopin, there may be enough data to train exclusively from here and see if any elements of his style can be learned. The data are in MIDI (Musical Instrument Digital Interface) format, which encodes the instructions for playing a song. In the chosen dataset there is just one instrument, the piano. For extracting the notes and chords in these instructions, I will use the **music21** Python module. Then, I will encode each song into a series of vectors of the form used by Hewahi, AlSaigal, and AlJanahi 2019 (described above). The final initialization step will be to loop through each of these songs turning every continuous slice of size (window size + 1) into a train/test record. Thus, the network will be trained to predict the next vector in a sequence for a sequence of size (window size). I plan to implement the network with the keras module for running TensorFlowin Python. This is essentially a binary classification problem for every key on the piano and therefore a binary cross-entropy loss function is appropriate.

One interesting thing done by Hewahi, AlSaigal, and AlJanahi 2019 is the augmentation of training data by transposing songs into random keys. They argued that the semantic meaning of a melody is invariant across transposition to other musical keys. This gives me another idea in which I would find the key of the song and transpose all the songs to C (the choice of the specific key is arbitrary since any sequence generated by the network could then be transposed into the user's key of choice). Then the neural network would be encouraged (but not enforced) to play within the confines of this key. Where before there were some number of randomly transposed versions of each sequence in the training data, there is now just one version of each sequence. I like this idea because it will cut down on training time by having less training samples while still containing all the relative melodic information. I also think this will simplify the problem for the neural network since the training data will have less variation because the notes of the C major (or equivalently, the A minor) scale will always be the most popular ones.

The final deliverable will take the form of a web service with an API allowing users to upload their own MIDI sequences for the purposes of assisting with music composition, or request a piece of music created by the algorithm from scratch. In the latter scenario, a random sequence from the training or testing set will be used as input into the predict method of the model. The next vector will be predicted and appended. Then, the left-most vector will be dropped, retaining the original size. There will be some pre-trained models for immediate use, but the user will also be able to specify the composers they wish the model to be trained on as well as the window size. This will be desirable since the window size will determine (approximately) how long/complex the generated pieces are.

For training models, I will use Paperspace. I will test a model on both a CPU and GPU and note the improvement, if any, to find out which one is better suited for this problem. The data requirements should be small. I shall start with a subset of the songs of chopin, of which there are 48 total. Through exploratory data analysis, I found that there are 39 which remain in the same key throughout. These songs, in processing, will become about 20,000 sequences from iterating over the original sequence with a window function. This training data will amount to about 230 MB for a window size of 16 (the size of the data is proportional to the window size). The amount of CPU hours needed is difficult to estimate at this time. Once the data preparation pipeline is complete, I will begin tests on my laptop to benchmark the performance.