CS 4100 Final Report: A Neural Embedding System for Playlist Generation

Maxwell Pinheiro, Olivia Floody, Krish Sharma

9 December 2020

1 Abstract

Capturing variation and similarity in artistic work forms is an increasingly difficult task. Music particularly is composed as a combination of sequential, contextual, and linguistic attributes which makes the task of learning relationships between different instances of songs a non-trivial task. In this report, we investigate a deep learning approach to provide recommendations for playlist generation through generating neural embeddings of songs to project the complex, high dimensional observation space into non-linear, low dimensional representations to determine similarity of songs in managable dimensions. Particularly, we generate a dataset of songs from various expert curated playlists which are similar in attributes rather than just drawn from the same genre. We leverage previous advances in sequential and natural language processing to extract sentiment features from songs using pretraining on a labelled Twitter sentiment dataset. Finally, we pass this constructed dataset to various model configurations to perform neural architecture search over three architectures. Our recommendations are provided as K-Nearest neighbors within the cluster corresponding to a sample's classification with respect to its embedding.

2 Introduction

In the past five years, there has been a rapid increase in the number of organizations leveraging artificial intelligence systems to improve recommendation systems. Across industries, the largest corporations are increasing their spending and efforts on developing machine learning techniques to provide consumers with better options from Amazon in retail, Netflix in entertainment, and Spotify in music. The pace of AI research in academia has rivaled this growth rate with breakthrough models such as BERT, GPT, and WaveNet being released and opening the gates to greater understanding of sequential data such as sound and language. At the intersection of language modeling and audio sequence modeling sits the challenge of understanding similarity and differences between songs to provide accurate recommendation in the form of playlist generation

given a starting point (i.e. an initial song as the seed).

The motivation to take on this project was primarily to provide ourselves with an automated service that could generate a playlist just given a starting point rather than having to rely on manually adding songs that felt similar or were captured as "also listened to by people who listened to X." We wanted to address the challenge of determining similarity in a high dimensional space by leveraging current data APIs that are available from Spotify for song data and Genius for lyrics data. The way we approached the problem was to pose a notion of similarity in a lower dimensional space by learning compressed, non-linear combinations of our initial dataset. This allows us to use algorithms that compute similarity, notably K-Nearest Neighbors, in order to determine the songs in our database that are similar to the song provided. We improve on simply applying the Spotify features to do this by incorporating multiple embeddings from a BERT semantic model that extract sentiment and lyrics similarity to make more informed recommendations.

Our initial proposal was to learn representations of songs from particular playlists and to use a generative adversarial network to create music from those playlist representations. This would have followed a similar pipeline as we ultimately use in our work. Rather than providing recommendations using K-Nearest Neighbors for recommendations, we would have instead used those latent embeddings and representations as input to a GAN that would generate output space samples (i.e. audio) from those latent variables. However, doing architecture search over this space would have been beyond the scope of our timeline. Instead, our work focuses on dataset collection, learning embeddings and semantic variables about lyrics using BERT, and projecting our high dimensional data into a reasonable space for K-Nearest Neighbors to provide recommendations. We perform architecture search over the depth and activations of our networks to determine optimal configurations for these networks.

3 Methodology

We framed our problem as an unsupervised and transfer learning approach to recommendation. In order for us to collect data about our domain, we leveraged the Spotify API which enabled us to extract data about specific playlists that we had selected on the basis of musical uniqueness between the playlist categories. The playlists used and their URLs are provided in Figure 1 below. After extracting the playlist data, particularly obtaining unique Spotify identifiers for each song in a playlist, we were able to query Spotify's analytics database which provides the analysis of audio features that have been analyzed for every track hosted on the platform. We use these audio features as part of our dataset. The list of features extracted from the Spotify API are provided below.

audio features = [‘danceability’, ‘energy’, ‘key’, ‘loudness’, ‘mode’, ‘speechiness’, ‘acousticness’, ‘instrumentalness’, ‘liveness’, ‘valence’, ‘tempo’, ‘time sig-

nature']

analysis features = ['num samples', 'analysis sample rate', 'analysis channels', 'end of fade in', 'start of fade out', 'loudness', 'tempo confidence', 'time signature confidence', 'key confidence', 'mode confidence']

While the API allowed us to determine whether a given song has lyrics or not, we were unable to obtain any direct lyrics analysis from the API. In order for us to gain insight into the semantic structure of the lyrics which are a main component in determining similarity between songs, we applied two BERT networks to our task at hand. Particularly, we use BERT for performing inference on the sentiment of the lyrics and to learn a high dimensional embedding of the text of the lyrics in order to determine textual similarity. To train BERT to predict the sentiment of the song, we attempt to use a network trained on a related textual domain as we do not have sentiment labels available for the songs. Specifically, we use the Twitter Sentiment Analysis dataset from the Kaggle challenge. We train BERT to predict the sentiment of a Tweet and use the learned network on our lyrics dataset. In order to obtain the embeddings of the text, we use a general pretrained BERT network from the Hugging Face team in order to extract a 768 dimensional vector representing the lyrics. However, this is of course far too high dimensional of a feature space and does not provide us with the ability to perform similarity calculations over the space, we outline an approach to address this problem downstream.

In line with our motivation to approach this task as the application of neural embeddings to determine recommendations in a reasonable comparison space, the next stage of our pipeline performs dimensionality reduction along our tabular audio features and language data. Audio data has relationships between features that are inherently non-linear which led us to not using PCA as a valid approach for us to reducing the number of features present in our dataset. Instead, we propose an Autoencoder approach to performing non-linear dimensionality reduction. We use a multi-layer network written in Keras with activations at every level and measure our reconstruction loss as a function of the Mean Squared Error over all our features. We exclude features such as the song name, artist, and unique URL from these embeddings as they are generally not informative for the similarity between two songs. It can be argued that an artist may produce the same type of music across multiple songs so that feature should be provided as an input to this dimensionality reduction and we include this in our architecture search. We use two separate Autoencoders for this dimensionality reduction. The first Autoencoder ingests the tabular data obtained from the Spotify API and the scores for sentiment classifications to learn a low dimensional representation of the data. The second works directly on the embeddings obtained from the the pre-trained base version of BERT as this data alone is more than 20 times larger than the Spotify and sentiment data in the feature space.

Finally, once we have obtained these latent features, we use a simple approach to determining similarity between a given sample and the dataset we

have of songs by using K-Nearest Neighbors over the low dimensional representation of the songs we have extracted so far. Due to computational resource restrictions, we operate on a dataset of about 700 songs, certainly to scale up to the millions of songs on Spotify, a localization process would be required to reduce millions of comparisons for every new sample.

4 Experiment and Results

In order to evaluate our system, we evaluated the individual networks and the end predictions. Particularly we perform architecture search over our two autoencoders to determine the optimal configurations for the layer sizes, latent space size, and activation functions. In order to do this, we abstracted our code to be able to pass in different sizes for the encoding dimension, an array of encoder layer sizes and their corresponding activation functions. Note that we did not have to pass in the layer specifications for the decoder as they were a mirror image of the encoder configuration.

We first had to obtain the embeddings of the text to perform dimensionality reduction so we used a base, uncased BERT model available from Hugging Face to obtain these embeddings. However, we wanted to decrease the size of these embeddings down from the 768 vector outputted by the model down to a reasonable size (i.e. less than 50 dimensions) in order to perform K-Nearest Neighbors on the embeddings. We did not have a metric that would enable us to evaluate the out of the box model however.

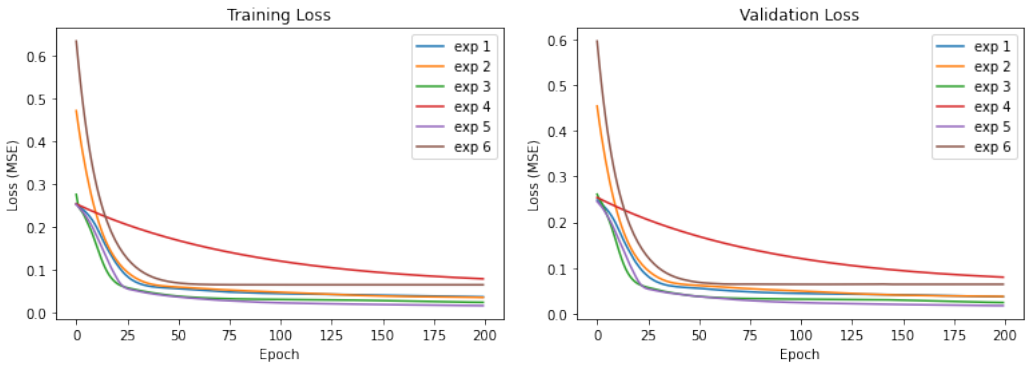
While we were fortunate to not have any significant architecture issues, we did encounter struggles with handling activation functions when projecting into the latent space. Our objective was to learn non-linear, low dimensional representations that accurately capture the observation space however, by using a ReLU activation, we lower bounded all latent variables to 0 due to the 0 setting nature of the ReLU when the input is less than 0. This resulted in some latent variables being zero across all samples that we were able to uncover by plotting the latent representations of our data. To resolve this, we attempted to use various activations but found that not applying an activation at the latent layer gave us the lowest Mean Squared Error.

Additionally, we trained our own sentiment extraction network as seen in BERTPretraining.ipynb but opted to obtain the embeddings using an out of the box network as we were unable to get the trained model to save from Google Colab as the model size was too large. This is certainly a point for improvement as we would like to instead train our data on lyrics data directly that is labeled with sentiment. Furthermore, we have no evaluation metric for how BERT trained on Twitter sentiment performs on the lyrics data in determining sentiment as we do not have labelled sentiment data for the lyrics.

We ran experiments on both our tabular encoder and our embeddings encoder using the specifications provided below. We evaluated the performance of these objective using a train-test split of 80:20 ratio on our original dataset and measured reconstruction using Mean Squared Error. We did not vary the

Table 1: Autoencoder architecture for tabular encoding

exp	encod. dim	layer sizes	activation fns	final training loss	final validation loss
1	4	15, 10, 5	all relu	0.0375	0.0374
2	5	20, 10	sigmoid, relu	0.0363	0.0381
3	4	20, 18, 12, 6	all tanh	0.0248	0.0249
4	1	24, 20, 16, 12, 8, 4, 2	all relu	0.0797	0.0802
5	10	20, 16	all relu	0.0170	0.0180
6	2	20, 18, 12, 6	all sigmoid	0.0658	0.0647

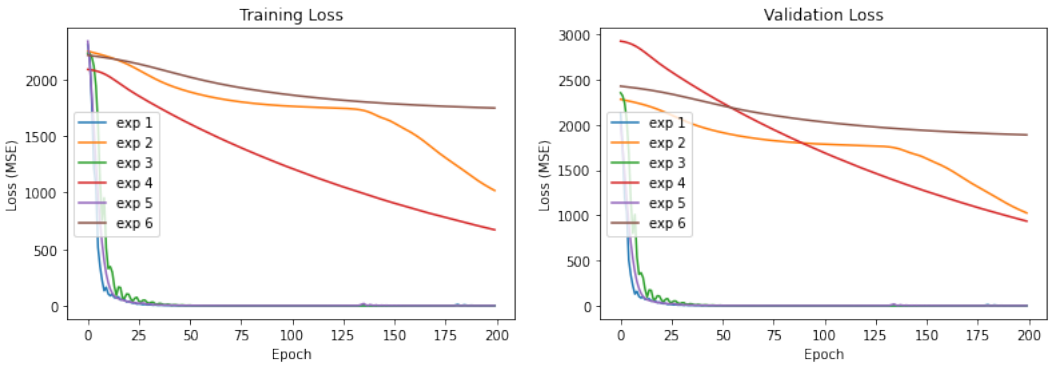


number of epochs which may have adversely impacted the convergence of deeper networks however we wanted to keep this term as a control for the search space as it was not directly a modification over the architecture. All of our tested architectures and their corresponding losses for both the training and validation set are provided in the tables and graphs below.

The loss charts shown here (Table 1) correspond directly to the experiments provided in the associated table. Evidently, the architecture does not severely affect the losses as they are all within the same range of 0.3 to 0.05 MSE which is not a considerable difference. However, it is important to note that these values were scaled using a min-max scaler so there are not values that are being mispredicted at a high magnitude that would significantly impact the loss. Additionally performance on the training set and validation set does not vary significantly as the general trend of the loss is similar as well. We find that the loss for experiment 4, even though the architecture tested is the deepest, performs significantly worse than the rest of our experiments, likely largely due to the fact that we attempted to squeeze variation across 26 variables into 1. We also noted that the loss for experiment 6 begins to rise after about epoch 50 for both the training and the validation set which indicates that we may be facing the problem of vanishing gradients due to the sigmoid activation where the activations tend to cluster along the extremes of the function. This may have been preventable by using early stopping.

Table 2: Autoencoder architecture for song similarity

exp	encod. dim	layer sizes	activation fns	final training loss	final validation los
1	30	500, 200, 100	all relu	0.7327	0.1102
2	5	200, 150, 100, 50, 10	alt. sigmoid & relu	1018.6	1025.0
3	15	250, 200, 175, 125, 50, 25	all relu	0.1697	0.0818
4	1	225, 150, 50, 15	all tanh	671.03	934.60
5	50	200, 100	all relu	0.0580	0.0604
6	2	100, 75, 25, 15, 5	all sigmoid	1748.5	1888.5



The two best performing networks came from experiment 5 and experiment 3. These networks had the largest latent space and a unique loss function respectively. While the best network came from experiment 5, it used more than twice as many latent features which suggests that the tanh activation function may be better suited for this problem, especially when it comes to capturing variation in the values of the latent space as the ReLU activation lower bounds values being passed forward at 0 which prevents potential variation along that axis from being leveraged to learn more informative latent variables.

These loss charts (Table 2) have much more variability than the charts above (Table 1). Experiments 2, 4, and 6 all have much higher loss than the other architecture configurations. The predominant factor explaining this is the dimensions of the latent space; the three experiments have encoding dimensions of 5, 1, and 2, respectively. This makes sense because the lyric data has a much higher dimension than the tabular data, with the lyric data having dimension 768 and tabular data 26. While we were able to successfully reduce to such low dimensions as 1, 2, and 5 with the tabular data, the lyric data will lose much more information if we reduce to those same dimensions. The autoencoder reported much lower loss when reducing to higher dimensions (15, 30, and 50) since these reductions are of similar scale as the reductions performed with the tabular data. It's also worth mentioning that the tabular data is largely preprocessed by the Spotify API, while the lyric data is not. This autoencoder

Ratings out of 10 for each Architecture Recommendation

Architecture 1	Architecture 2	Architecture 3
7.49	6.06	5.87

thus performed encoding on much more raw data, which would explain why it performs less optimally than the previous autoencoder.

We ran a brief survey using the predictions made for the same songs by different architectures. These different architectures were particular the combination of the two best autoencoders, two second best autoencoders, and two third best autoencoders based on Mean Squared Error. All other attributes of the system were held consistent. We found that the mean squared error was associated with the average rating that users gave the playlist for relative rankings of the recommendations by the system. The survey used can be found here: https://docs.google.com/forms/d/e/1FAIpQLSerHy3vX38vsfa-Yg0TYM8PCdIKfgBRwyYMsAZ0jxdPXzrdsg/viewform?usp=sf$_l ink$

5 Conclusions

Ultimately, we found that our approach allowed us to extract critical information about audio in the form of scalar values which provided reasonable suggestions and recommendations for the playlist. However, we also learned that in order to develop upon an initial architecture, developing that architecture requires a significant amount of ideation regarding the design of the codebase that allows for efficient experimentation. This can be see in our abstraction of creating an architecture using arguments rather than statically defining specific layers.

We additionally found that using autoencoders allowed us to learn effective low dimensional representations which another approach like PCA may not have allowed for as PCA is entirely linearly dependent. We noticed that there are benefits to using deeper neural networks that allows us to capture more intricate non-linear relationships over many layers. Additionally, we learned that being able to convert sequential data into low dimensions can abstract those complexities by using language models like BERT that provide embeddings.

Finally, we used a pretty simple algorithm to do the recommendation itself, K-Nearest Neighbors, but to allow this to be feasible with an original dataset that is near thousands of dimensions and in reality operates over millions of songs, learning these latent variables is quite important. Sometimes the simplest models require a significant amount of processing ahead of time.

Finally we learned the importance of starter code - having to work through and write large amounts of boilerplate code ate hours of our time which resulted in this project taking over 50 hours.

A Code Instructions

Our code can be found at: https://github.com/krisharma/spotify-recs

In order to streamline the exploration and running of our code, we created a main.py script that takes in command line arguments to provide the user the option to specify what playlists they would like to serve as their database of the content and playlists to be sampled from. We then automatically run the following processes:

1. Extract data from the Spotify API
2. Extract lyrics from Genius API for all songs in those playlists
3. Extract sentiments from all of the lyrics extracted
4. Extract text embeddings from the lyrics
5. Train tabular autoencoder on the Spotify data and obtain embeddings for the dataset
6. Train lyrics autoencoder and obtain embeddings of the lyrics
7. Obtain data necessary for the "seed" song to be passed to the tabular and lyrics encoder to obtain the latent features of the song to use as the first song in the playlist
8. Run K-Nearest Neighbors on the dataset with the new sample to determine the K most similar songs

In order to run this please follow the format of the command:

python3 main.py –pretrained False
–playlists 'https://open.spotify.com/playlist/37i9dQZF1DWWEJlAGA9gs0'
–seed 'Blinding Lights,The Weeknd'
–length 5

The playlist argument takes in a link to a Spotify playlist, you can add multiple playlists by separating them just by a comma. The seed takes in a string of SongName,Artist. Finally the length argument takes in an integer to represent the length of the playlist to be outputted. This should be between [1, 700].

If you want some clarification on how our program runs, the workspace.ipynb file demonstrates the recommendation software (predominantly at the bottom of the file). These would be the best way to see that our code is functional as the command line interface was only able to work on specific hardware due to dependency compatibility issues.

Our experiments that we ran to perform architecture search on can be found in the .ipynb files.