Generative AI for Spotify Playlists

Shashidhar Gollamudi, Sunny Huang

Artist's Statement

Music is a lens that can color our world, and exposure to new music can change our perspectives and broaden our horizons. In an era dominated by algorithms and data-driven recommendations, our project strives to infuse the art of curation with the precision of artificial intelligence. While the world of streaming platforms has provided unprecedented access to an immense musical library, the process of playlist creation often relies on predetermined factors, such as listening history and existing playlists. Our aim is to introduce a novel approach that delves into the uncharted territory of playlist generation based solely on the evocative power of playlist titles. Although unconventional compared to art forms like oil on canvas or literature, playlists have become an important part of many people's lives, and a good playlist can invoke emotions just like other pieces of art. They can set the mood for parties, help improve focus when studying, or lift up moods on a bad day.

Since the emergence of Sequence to Sequence transformers, we've seen a proliferation in their usage for Machine Translation applications. However, these transformers aren't just limited to translation, they've also been used for image captioning, conversation, and text summarization (1). In this project, we apply these transformers to the generation of Spotify playlists from playlist titles. Currently, Spotify has the ability to suggest songs based on your listening history and the songs in a playlist, however it lacks the ability to make inferences from *only* the titles of those playlists.

We implement a Recurrent Neural Network (RNN) using Keras, a Transformer with Attention, and finally an N-Gram model for comparison purposes, which we detail in further sections.

How Models Work

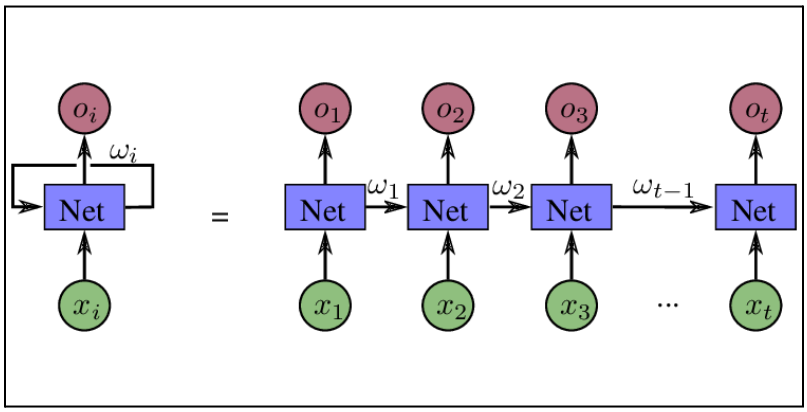
Three different models–N-grams, recurrent neural networks, and transformers–were used to train and generate playlists. All of them were able to generate playlists consisting of a list of Spotify track IDs, or URIs. All three are trained on the Spotify Million Playlist Dataset, which is a large database of a million playlists, including names and tracks.

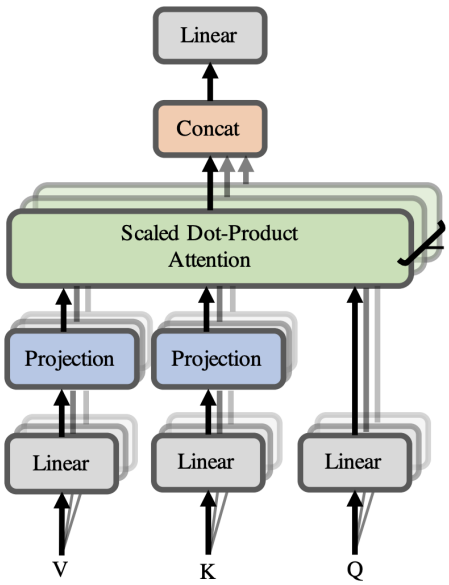
The N-gram model function is very similar to the language model in homework 3. One of the main differences between the models is how the data is processed. The playlist data is in JSON format, as shown on the right, and some of the input playlists also contain no songs, so additional preprocessing was required to extract the track information and skip over these empty playlists. After this, the training process was the same as with text generation. Sentences


```
{
    "name": "Happy Daze",
    "num_holdouts": 31,
    "pid": 1003326,
    "num_tracks": 36,
    "tracks": [
        {
            "pos": 0,
            "artist_name": "Bruno Mars",
            "track_uri": "spotify:track:7l1qvxWjxcKpB9PCtBuTbU",
            "artist_uri": "spotify:artist:0du5cEVh5yTK9QJze8zA0C",
            "track_name": "Count On Me",
            "album_uri": "spotify:album:1uyf3l2d4XYwiEqAb7t7fX",
            "duration_ms": 197373,
            "album_name": "Doo-Wops & Hooligans"
        },
        {
            "pos": 1,
            "artist_name": "Gavin DeGraw",
            "track_uri": "spotify:track:1ccNXmmgYnajJ8ulYYEK9Y",
            "artist_uri": "spotify:artist:5DYAABs8rkY9VhwtENoQCz",
            "track_name": "I Don't Want to Be",
            "album_uri": "spotify:album:70ZsiClM7RBUsoTERq2qXu",
            "duration_ms": 219080,
            "album_name": "Chariot"
        },
```

were created, with the playlist name at the start of the sentence, followed by all the track URIs. These are then tokenized and trained on. Finally, playlists are generated by passing a playlist title, padded by sentence start tokens, into the model, and converting the generated "sentence" into a playlist with song titles using a mapping from URI to song title we created during the preprocessing stage. We also tried using the Spotify API to retrieve the song information again after we generated playlists, but the API would sometimes throw errors, and since we ignore unknown tokens, all the generated songs would be songs that are already in our mapping.

The next model we built was the recurrent neural network. It uses an encoder-decoder architecture and is a sequence-to-sequence (Seq2Seq) model that can generate multiple outputs from multiple inputs. The process of building this model was similar to that of the N-gram model. We preprocessed data in a similar way. First, we read the JSON data and converted it into a list of playlist titles and playlist tracks. One extra step that was necessary for this model was padding both the titles and tracks so that all inputs and outputs were the same length. After this, we built the layers of the model which contain an embedding layer and an encoder layer, followed by a RepeatVector layer, which, from what the name implies, will repeatedly process an input vector a set number of times, as shown in the image above. Finally, we have a decoder layer and a logits layer, which will be the output of the model. The RNN is then trained using Adam as the optimizer. In order to generate output for a single playlist title, we used top-k sampling in order to get the sequences with the highest probability. One issue we found was that the model typically returned the same few tracks, regardless of the playlist title. This is likely due to the lack of training data since there were only 7,000 non-empty playlists, and some songs only occur a few times, so it will not be favored by the model. One potential fix to reduce the bias is to upsample sample playlists of certain genres so that they are more balanced with other genres. However, this might be difficult, since it would require another model that is able to determine the genre of a playlist.





The last model that we build is a transformer model. Similar to the RNN, it is also a Seq2Seq model. At first, we attempted to fine-tune BERT, but the base model is more suited for classification, and we were struggling to train it to generate text. We found another model called BERTGeneration, which is able to generate sentences in the way we wanted, but we ended up struggling to get our training data in the correct format, and the model was running out of input during training. Finally, we decided to build our own model using PyTorch. We used an encoder-decoder architecture similar to the RNN and also uses

multi-head attention (shown on the left) to focus on specific parts of the input sentence (2). Each of these layers was built as a subclass to the Module class in the torch.nn library. The preprocessing of the data was similar to that of the RNN. We read the data in the JSON, and then padded both the titles and the tracks. Since we were working with PyTorch for this model, we also needed to convert the data into LongTensors, so that they could be trained. After training for 20 epochs, the loss was still very high, at around 10. This is likely due to the same issue we saw with the RNN, where a few songs were repeatedly being selected. We saw the same issue when generating playlists. We were not able to implement a top-k sampling the same way as in the RNN, and when we tried using the maximum probability, the same song was returned regardless of the playlist title. This is an issue we need to work on in the future.

How our Art was Produced

Because of the subjective nature of our project and results, the exploration we did in tuning and adjusting our models was also largely subjective. However we'll briefly go over some of the parameters we changed.

Beginning with our N-Gram model, we trained the model using 2 all the way up to 10 grams. We found that performance was not extremely different between these variations, we believe this is due to the fact that the size of the data set was too low for these patterns to be captured. In our transformer RNN, we experimented with the size of the Top-K sample that our model chooses from. Increasing the Top-K helped to artificially increase the variation in our playlist generation, increasing the likelihood that more accurate but less popular songs would be generated. For both our transformer RNN and transformer with attention we experimented with training epochs, finding that increasing our training repetitions did not necessarily improve our models output. We believe that because the training set was so small, increasing the number of epochs beyond around 3-5 causes our models to 'overfit,' choosing the same popular songs again and again, even in the same playlist. Additionally, we explored variations in the model architecture, including adding more layers with different numbers of nodes. However, we found that these variations had little impact on our overall results. As mentioned before, we believe that the main bottleneck was the size and balance of the dataset.

Future Experiments

Although we've developed our models to a significant extent, we were limited by both time and computing constraints, and there are a few additions that we would have liked to implement.

In our current implementation we only used a small subset (~10,000) of the playlists from our Million Playlist data set. Because of this, our models were more susceptible to variations in the dataset. For example, although the playlists have different songs based on their titles, certain popular songs are more likely to appear regardless. As a result, our models more often predict these songs even if a title may not suggest it. Additionally, songs that appear less in all datasets are less likely to be generated, even if they better fit the title of a playlist. We believe that with a larger sample of the data set, our models would be better equipped to predict a larger variation of songs that can better fit the provided title. Another possible solution could be to upsample some of these less seen songs, or do the opposite with the popular songs. The results of our project, while limited by these constraints, inspired us to look to possible future applications as well.

One enhancement that could improve our models is providing them with artist and genre information as well. Our current implementations work solely on the track IDs of each song, however, with additional track meta-data, our models can incorporate other dimensions of playlists, and consider more complex relationships between songs in those playlists.

Additionally, while these predictions are somewhat useful as executable code, we believe that a user interface would allow our project to be more user-friendly. An interface solution would allow us to also incorporate the Spotify API to retrieve track and artist names, album covers, and other metadata and also actually create these playlists for users in Spotify.

With these changes, our project could be more accurate to titles as well as be applied outside of the academic setting. And although our implementations were partially constrained, we feel we've made significant progress in music discovery and exploration.

Works Cited

1. Radhakrishnan, Pranoy. “Sequence to Sequence Learning.” *Medium*, Towards Data Science, 10 Oct. 2017, towardsdatascience.com/sequence-to-sequence-learning-e0709eb9482d.
2. “Papers with Code - Multi-Head Linear Attention Explained.” *Explained | Papers With Code*, paperswithcode.com/method/multi-head-linear-attention. Accessed 6 Dec. 2023.
3. “Recurrent Neural Network Structure.” *Research Gate*, www.researchgate.net/figure/Recurrent-Neural-Network-Structure-The-left-is-the-typical-RNN-structure-The-right-part_fig3_311805526. Accessed 6 Dec. 2023.