Deep Learning for Music Generation

Isaac Tham
iztham@wharton.upenn.edu

Matthew Kim
mattmkim@seas.upenn.edu

This project aims to use deep learning techniques to generate unique music. Our baseline method used a transformer architecture to perform sequence generation using next token prediction. Using the same idea of next token generation, we implemented an RNN model to predict the next note given an input sequence for a single track and a five-instrument track. We also implemented a CNN model to produce a more well-formed arrangement between the five instruments using both a melody CNN and a conditional harmony CNN. To incorporate more randomness into the generated output, we used a VAE to encode sequences into a latent space and added noise to create variation in the generated output. Lastly, we attempted to use a GAN, but had difficulty in training the model and ran into mode collapse. Overall, we found the most success in our VAE model, with the outputs of this model being the most well-formed and having significant variation.

Introduction

Deep learning has radically transformed the fields of computer vision and natural language processing, in not just classification but also generative tasks, enabling the creation of unbelievably realistic pictures as well as artificially generated news articles. But what about the *field* of audio – or more specifically – music? In this project, we aim to create novel neural network architectures to generate new music, using 20,000 MIDI samples of different genres from the Lakh Piano Dataset, a popular benchmark dataset for recent music generation tasks.

Background

Music generation using deep learning techniques has been a topic of interest for the past two decades. Music proves to be a different challenge compared to images, among three main dimensions: Firstly, music is temporal, with a hierarchical structure with dependencies across time. Secondly, music consists of multiple instruments that are interdependent and unfold across time. Thirdly, music is grouped into chords, arpeggios and melodies – hence each time-step may have multiple outputs.

However, audio data has several properties that make them familiar in some ways to what is conventionally studied in deep learning (computer vision and natural language processing, or NLP). The sequential nature of the music reminds us of NLP, which we can use Recurrent Neural Networks for. There are also multiple 'channels' of audio (in terms of tones, and instruments), that are reminiscent of images that Convolutional Neural Networks can be used for. Additionally, deep generative models are exciting new areas of research, with the potential to create realistic synthetic data. Some examples are Variational Autoencoders (VAEs) and Generative Adversarial Neworks (GANs), as well as language models in NLP.

Most early music generation techniques have used Recurrent Neural Networks (RNNs), which naturally incorporates dependencies across time. Skuli (2017) used LSTMs to generate single-instrument music in the same fashion as language models. This same method was used by Nelson (2020), who adapted this to generate lo-fi music.

Recently, Convolutional Neural Networks (CNNs) have been used to generate music with great success, with DeepMind in 2016 showing the effectiveness of WaveNet, which uses dilated convolutions to generate raw audio. Yang (2017) created MidiNet, which uses Deep Convolutional Generative Adversarial Networks (DCGANs) to generate multi-instrument music sequences that can be conditioned both on the previous bar's music, as well as the chord of the current bar. The GAN concept was taken further by Dong in 2017 with MuseGAN, which uses multiple generators to achieve synthetic multi-instrument music that respects dependencies between instruments. Dong used the Wasserstein-GAN with Gradient Penalty (WGAN-GP) for greater training stability.

Lastly, as the latest advances in NLP have been made with attention networks and transformers, attempts have been similarly made to apply transformers to music generation. Shaw (2019) created MusicAutobot, which uses a combination of BERT, Transformer-XL and Seq2Seq to create a multi-task engine that can both generate new music as well as create harmony conditional on other instruments.

Dataset

Our data came from the Lakh Pianoroll Dataset, a collection of 174,154 multitrack pianorolls derived from the Lakh MIDI Dataset, and was curated by the Music and AI Lab at the Research Center for IT Innovation, Academia Sinica. We used the LPD-5 version of the dataset, which includes tracks for piano, drums, guitar, bass, and strings, allowing us to generate complex and rich music and to demonstrate the ability of our generative models to arrange music across different instruments. We used the cleansed subset of the Lakh Pianoroll Dataset, which includes 21,245 MIDI files. Each of the files had corresponding metadata, allowing us to determine information about each file such as the artist and title name.

Baseline Method: Next-Note Prediction with RNNs

To establish a baseline of music generation that we can improve on, we used recurrent neural networks (RNN), an existing and easily-replicable method. Generating music is formulated as a next-note prediction problem. (This method is very similar to recurrence-based language models that are used in NLP) This would allow us to generate as much music as we wanted by continuously passing in the note that was generated back into the model.

In terms of implementation, we used the Gated Recurrent Unit (GRU) instead of the vanilla RNN, because of its better ability to retain long-term dependencies. Each GRU would take in the previous layer's activation and output as input, and the output would be the next note given the previous activation and input.

To create the data needed to train our recurrent neural network, we first parsed the piano notes of our dataset, representing each file as a list of notes found in the file. We then created the training input sequences by taking subsets of the list representation for each song and created the corresponding training output sequences by simply taking the next note of each subset. With this training input and output, the model would be trained to predict the next note, which would then allow us to pass in any sequence of notes and get a prediction of the next note. Each input sequence was passed into an embedded layer that created embeddings of a size of 96. This embedding was then passed into a gated recurrent unit with a single layer, which was then passed to a fully connected layer to output a probability distribution of the next note. We could pick the note with the highest probability as the next predicted note, but that would lead to deterministic sequences with no variation. Hence, we sample the next note from a multinomial distribution with the output probabilities.

Evaluating

While the RNN next-note prediction model is easy and clean to implement, the generated music sounds far from ideal and there is very limited utility. Because we encode every single note into a token and predict a probability distribution over the encodings, we can only really do this for one instrument, because for multiple instruments, the number of combinations of notes increases exponentially. Also, the assumption that every note is of the same length definitely does not reflect most musical works.

Multi-instrument RNN

Hence, we sought to explore other methods to generate music for multiple instruments at the same time, and came up with the **Multi-instrument RNN.**

Instead of encoding the music into unique notes/chords like we did in the initial idea, we worked directly with the 5 x 128 multi-instrument piano roll at each time-step, flattening it to become a 640-dimensional vector that represents the music at every time-step. Then, we trained an RNN to predict the next time-step's 640-dimensional vector, given the previous length-32 sequence of 640-dimensional vectors.

While this method would theoretically make sense, it was challenging to produce satisfactory results due to the difficulty in generating variety that was complementary across all the instruments.

- In the single-instrument set-up, we sampled from a multinomial distribution with probability weighted by the output softmax scores to generate the next note. However, since all instruments are placed together in the 640-dimensional vector, generating the next note using softmax-ed scores over the entire 640d vector could mean that some instruments could potentially have multiple notes while some instruments have none.
- We attempted to solve this problem by running the softmax function separately for each of the 5 instrument's 128-dimensional vectors, so we could ensure we generate a certain number of notes for each instrument.
- However, this meant that the sampling for each instrument was independent from each other. This means that the generated piano sequence would not be complementary to the other instruments' sequences. For example, if the C-E-G chord is sampled from the sequence, the bass' has no way of incorporating this, and could sample the D-F-A chord, which is harmonically dissonant and not complementary.
- Moreover, there was the problem of not knowing how many notes to be sampled for each instrument at each time. This problem was not present in the single-instrument set-up because single notes and multi-note chords are all encoded as integer representations. We addressed this issue by sampling a specified number of notes for each time step (e.g. 2 for piano, 3 for guitar) from the multinomial. But this was unsuccessful as the generated music sounded highly random and unmusical.

Audio samples of generated music from these two songs can be found in the Generated Music Files folder with the name RNN Multitrack'.

Moving from Recurrent to Convolutional

From this point on, we decided to focus on **Convolutional Neural Networks (CNNs)** rather than RNNs to generate sequences of music. The CNN would directly generate a length-32 sequence by outputting a 5 x 32 x 128 3-dimensional tensor. This would solve the problem of not knowing how many notes to generate and having to use multinomial sampling. CNN architectures, such as WaveNet, have been shown to achieve just as good, if not better performance as RNNs in sequence generation. Additionally, they are much faster to train due to performance optimizations with convolutional operations.

MelodyCNN and Conditional HarmonyCNNs

In order to generate multiple instrument tracks compatible with each other, we tried a two-part generation model that comprises a **MelodyCNN** for next-time-step melody generation, as well as a **Conditional-HarmonyCNN** to generate the non-piano instruments, given the melody for the same time-step as well as that instrument's music for the last time step.

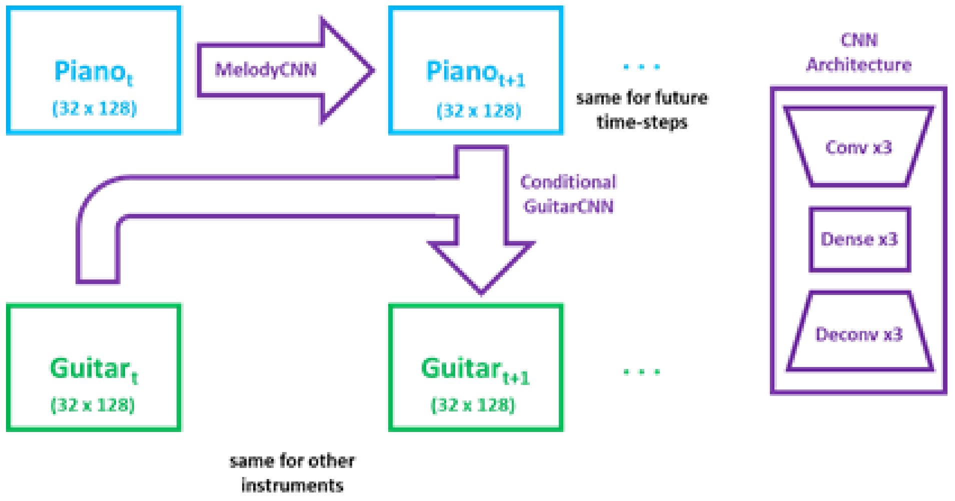


Figure 3: Architecture of the MelodyCNN + Conditional HarmonyCNN used to generate music.

Since the input and output sizes are the same (32 x 128), the MelodyCNN architecture used was symmetric, with 3 convolutional layers, 3 dense layers and 3 deconvolutional layers. The Conditional HarmonyCNN used 3 convolutional layers for each of the inputs (piano as well as instrument's previous), then concatenated the resulting tensors before passing through dense and deconvolutional layers.
Hence, the MelodyCNN learns a mapping between piano sequences in successive time steps, while the Conditional HarmonyCNNs maps from piano music space to the other instruments.

Using the 5 CNNs in total (one for each instrument), new music can be generated iteratively given a starting multi-instrument sequence. First, MelodyCNN is used to predict the next piano sequence, and the Conditional HarmonyCNNs are used to predict the other instruments.

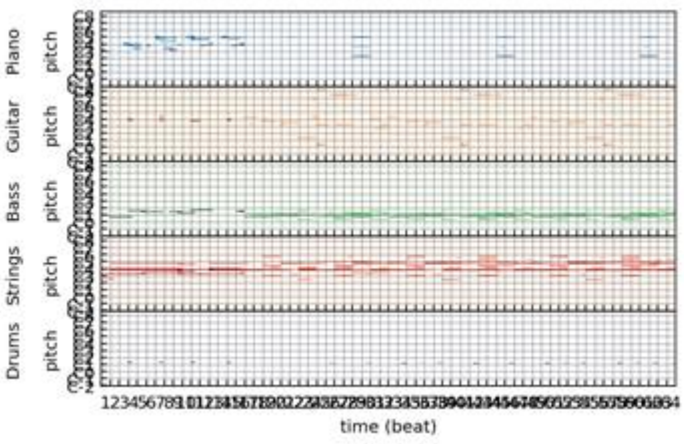


Figure 4: Pianoroll of music generated by the MelodyCNN + Conditional Harmony CNN.

This framework was successful in generating multi-instrument music sequences where the instruments sound musically complementary. However, varying the starting sequence from which the music is generated only led to very little variation in the generated music, as shown in the pianoroll above: the three generated sequences are nearly identical to each other.

Audio samples of generated music from these two songs can be found in the Generated Music Files folder with the name 'MelodyCNN+ConditionalCNN', 'MelodyCNN All Same', 'MelodyCNN All Same2'.

This shows that the CNNs likely converged on outputting only a small subset of common sequences in the training data that minimized the training loss. Another method needs to be found to generate some variety in the output music, given the same input, and to achieve this, we turn to VAEs.

Using Variational Autoencoders (VAEs)

Background to VAEs

A variational autoencoder (VAE) is an autoencoder in which training is regularized to ensure that the latent space has good properties that allow a generative process. Two such properties are continuity – close points in latent space should give similar points once decoded, and completeness – a point sampled from latent space should give meaningful content once decoded.

A vanilla autoencoder encodes the inputs into a vector in latent space, but has no guarantee that the latent space satisfies continuity and completeness that allow new data to be generated. In contrast, a VAE encodes an input as a distribution over latent space. Specifically, we assume the latent distribution to be distributed Gaussian, hence the encoder encoding a distribution is equivalent to the encoder outputting the mean and standard deviation parameters of the normal distribution.

To train the VAE, a two-term loss function is used: a reconstruction error (difference between decoded outputs and inputs), as well as a regularization term (KL-divergence between the latent distribution and standard Gaussian) to regularize the latent distribution to be as close to standard normal as possible.

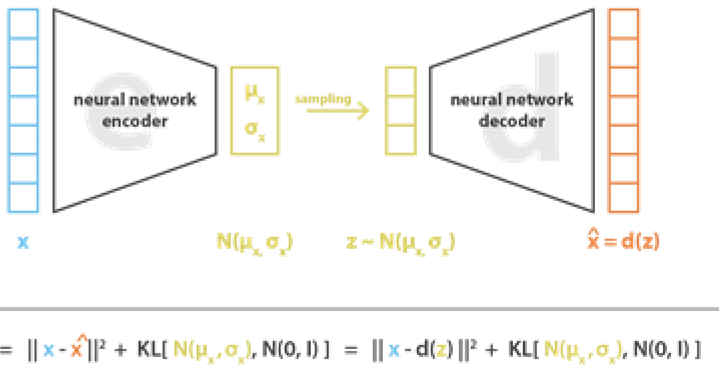
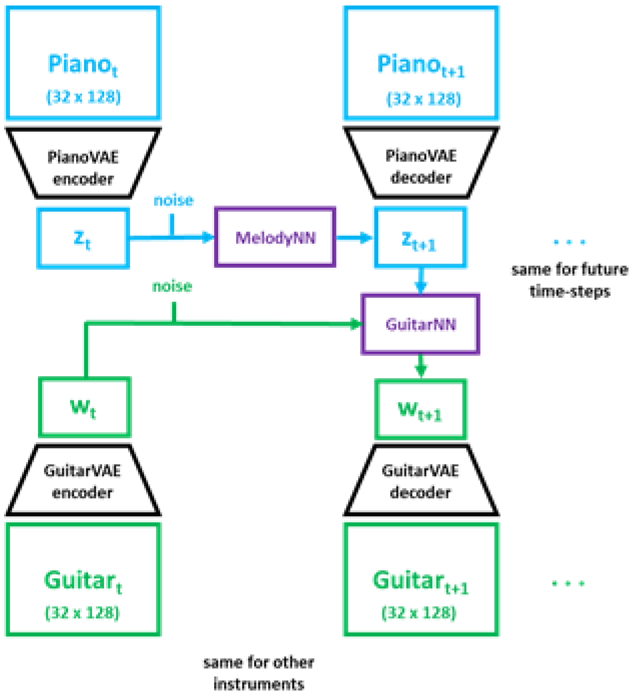

neural network encoder
μ_x
σ_x
sampling
neural network decoder
x
$N(\mu_x, \sigma_x)$
$z \sim N(\mu_x, \sigma_x)$
$\hat{x} = d(z)$
$\text{loss} = \|x - \hat{x}\|^2 + KL[N(\mu_x, \sigma_x), N(0, I)] = \|x - d(z)\|^2 + KL[N(\mu_x, \sigma_x), N(0, I)]$

Figure 5: Illustration of how a Variational Autoencoder (VAE) works.

Application



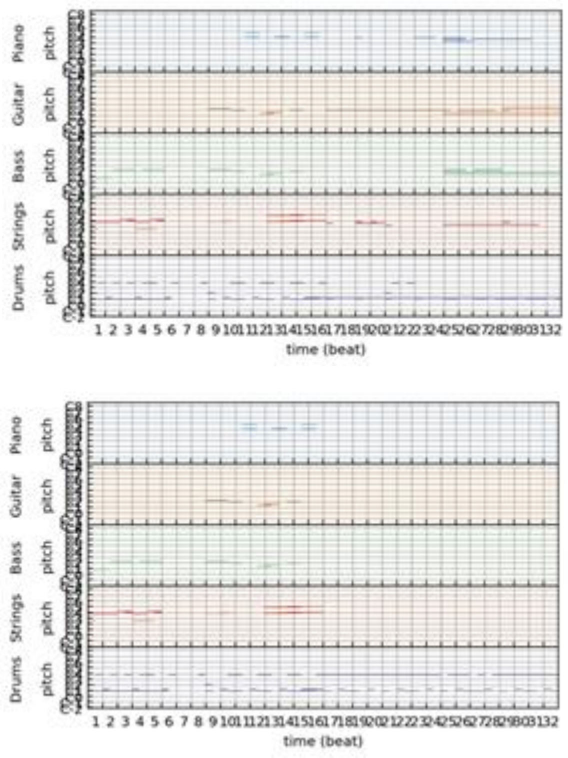


Figure 6: Architecture of VAE-NN used to generate music.
Figure 7: Two pianorolls generated from the same starting sequence. One example of variation shown in the music output is shown above. Both tracks above had the same starting sequence, but the generated drumbeats were slightly different. Also, the first track had a piano section toward the end while the second track did not, and the conditional NNs responded by varying the accompanying instrumental tracks that were generated.

We hence apply VAEs to the music generation task. The previous piano input is encoded by the piano VAE into a latent piano encoding of dimension K, z_t. Then, random noise is added to the encoded latent distribution's mean parameters. The standard deviation of this random noise is a hyperparameter that the user can tune based on the amount of variation he desires. The latent parameters z_t are then input to MelodyNN, a Multi-Layer Perceptron that learns a mapping from the previous piano sequence's latent distribution to the next piano sequence's latent distribution. The output z_{t+1} is then decoded to become generated next piano output.
Instrument-specific VAEs are also trained on the other four instruments (guitar, bass, strings, drums).

Then, similar to the ConditionalCNN earlier, we use a ConditionalNN, another MLP that takes in the generated next-period piano latent parameters z_{t+1} as well as the previous-period guitar latent parameters w_t, and learns a mapping to the next-period guitar latent parameters w_{t+1}. w_{t+1} is then decoded by the instrument-specific VAE's decoder to produce the next-period guitar output. 4 ConditionalNNs are trained, one for each non-piano instrument, which allow the next 5-insturment sequence to be generated.

Hence, by mapping the musical inputs into latent distributions with VAEs, we can introduce variation to the generated music output by adding random noise to the encoded latent distribution's parameters. Due to continuity, this ensures that after adding random noise, the decoded inputs are similar yet different from the original inputs, and due to completeness, it ensures that they give meaningful music outputs that are similar to the input music distribution.

<u>Results</u>

VAEs of latent dimensionality 8, 16, 32 and 64 were trained. In the end, a 16-dimensional latent space was used to train the conditional NNs, since the music samples are relatively sparse in music space.
After training the conditional NNs, we find that the VAE+NN method is successful in creating multi-instrument outputs that sound coherent, as well as having appropriate amounts of variation to be aesthetically pleasing. Random noise of standard deviations between 0.5 to 1.0 were found to generate the best amount of variation.

Audio samples of generated music using the VAEs can be found in the Generated Music Files folder with names from 'VAE Good 1' to 4.

Generating Music in certain Styles

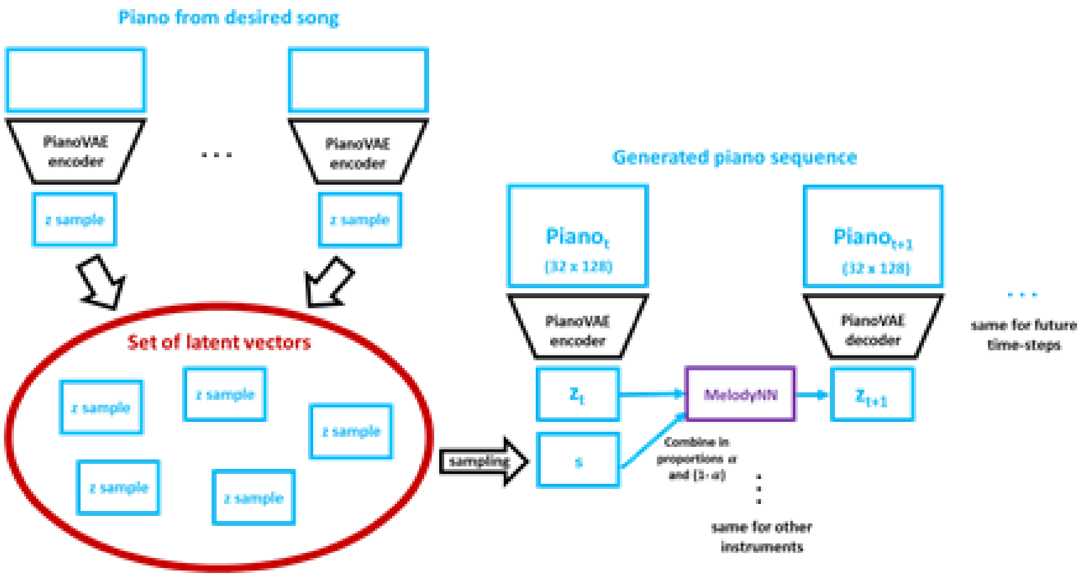


Figure 8: Method to generate music conditional on specific styles

The VAE-NN framework explained above allows us a straightforward method to generate music based on specific styles, such as a certain artist, genre, or year. For example, if we wanted to generate music in the style of *Thriller* by Michael Jackson, we could:

1. Break the song into 32-step sequences and encode each sequence's pianoroll into the latent space using each instrument's VAE encoder. Store the unique sequences in a **set** for each instrument.
2. When generating music from a starting sequence, one latent vector per instrument is sampled from this set. This sampled latent vector (from our desired song) s is then interpolated with the previous sequence's latent vector z_t to generate a new latent vector $z_t' = \alpha s + (1-\alpha)z_t$, with α being the **latent sample factor**, which is a hyperparameter that can be tuned. (Choose higher values of α to the generated music to be more significantly conditioned on the desired style)
3. Use z_t' instead of z_t as the input for the MelodyNN to generate the new latent vector and hence the generated piano sequence.

Using this method and α=0.5, we generated new music conditioned on several songs, some examples being *Thriller* by Michael Jackson and *I Want It That Way* by Backstreet Boys. This was successful in generating audio samples that have some similarity to the original song, but with some variation as well. (Once again, the extent of variation can be tuned with the *noise_sd* hyperparameter as well). One can even generate music based on samples which are a hybrid of different artists or styles, hence allowing music enthusiasts to synthesize music combining the styles of different music stars.

Audio samples of generated music from these two songs can be found in the Generated Music Files folder with the name 'VAE - Thriller' and 'VAE - I Want It That Way'.

Bloopers: GANs

Inspired by the success of MidiNet, which used Deep Convolutional Generative Adversarial Networks (DCGANs) to produce realistic-sounding music, we attempted to use GANs as well to generate music. GANs have been known to generate highly realistic synthetic samples in the computer vision field, better than VAEs. This is because GANs do not estimate the explicit probability density of the underlying distribution, while VAEs attempt to optimize the lower variational bound. However, GANs have been known to be very difficult to train successfully.

We used a generator with 6 de-convolutional layers, taking in a noise 100-dimensional vector and generating a 5 x 32 x 128 multi-instrument music sequence. The discriminator has the opposite architecture, taking in a 5 x 32 x 128 music sequence, passing it through 6 convolutional layers and outputting a probability of the sample being real.

For both generator and discriminator, PReLU activation was used, as well as batch normalization for the convolutional layers. Adam optimizer was used for both.

The following methods were attempted to improve the stability of the GAN:

- Label smoothing: Instead of using hard labels 0 or 1 for the generated or real images respectively, we add random noise to the label (so that the generated images have label between 0 and 0.1, and real images have label between 0.9 and 1).
- Feature matching: Adding L2 regularizers to enforce the distributions of the real and generated data to be close. Two regularizers were used: the first one on the absolute difference of expected value of real vs generated image inputs, and the second one on the absolute difference of expected value of the outputs of the first convolutional layer for the real and generated images inputs.
- Two Time-Scale Update Rule (TTUR): Using a higher learning rate for the discriminator compared to the generator.
- Tuning the learning rate

Despite several attempts, training the GAN proved unsuccessful in generating a variety of realistic-sounding music. There were instances of mode collapse, such as the generated audio sample below, which is 100 samples generated from different noise vectors concatenated together. The generated samples are mostly similar. Audio samples of generated music can be found in the Generated Music Files folder with the name 'GAN Mode Collapse'. Other attempts failed to learn anything substantial.

Another Baseline: Transformers

A second, more sophisticated, baseline method that we used was the transformer architecture. Transformers have found a lot of success in NLP, capable of training much more quickly and having much better long-term memory than older recurrence-based language models. We used the **Music Autobot project's Transformer-XL architecture** due to its extremely reproducible code – we thank Andrew Shaw for this and encourage you to check out his brilliant series of articles.

In a vanilla transformer model, direct connections between data units provide the chance to capture long-term dependencies. However, these vanilla transformers are implemented with a fixed-length context, hence the transformers are unable to model dependencies that are longer than the fixed length, and context fragmentation occurs.

The Transformer-XL architecture provides techniques to solve these issues. First, it has a segment-level recurrence mechanism. While training, representations computed for the previous segment are cached such that they can be used as an extended context when the model processes the next segment. Thus, information can now flow through segment boundaries, and solves the context fragmentation problem as well. Secondly, it has a relative positional encoding scheme. This allows the model to understand not only the absolute position of each token, but the position of each token relative to one another as well, which is extremely important in music.

Unlike text, tokenizing music is much trickier. A single music note represents two different values - a pitch as well as a duration (it can represent many more things as well, such as loudness and timing, but these are less important for our purposes). As a result, each note needs to be encoded into a sequence of tokens. Fortunately, the *MusicAutobot* project handles the tokenization of MIDI files.

We find the generated music to be relatively good as well. Audio samples of generated music from these two songs can be found in the Generated Music Files folder with the name 'Michael Jackson Output'.

Conclusion

Overall, we have implemented a variety of deep learning methods to the music generation problem with varying levels of success. Our baseline method utilized a recurrent neural network model for both a single track and multiple tracks. While this model found more success with regards to the musicality of the produced notes, it was very limited in utility as it could only produce notes on the quarter note beats. We then moved on to a convolutional neural network model, using a vanilla CNN to produce the piano track, and a conditional CNN that used the piano track to produce the other instrument tracks. We found the arrangements produced by the CNN models to be much more well-formed and coherent, because we were using conditional models.

The novel VAE-based architecture that we devised was the most successful contribution for our project. By encoding sequences into a latent space using a VAE, we can then add noise in the latent space to increase the variation of the generated output in a controllable way, while maintaining the similarity between previous sequence, ultimately improving the uniqueness of our generated music.

You can find our codebase here – feel free to use it for your own music generating adventures. Do leave a comment or reach out to me personally if you have any questions and I'll be glad to help! What truly makes the field of deep learning amazing is the open-source collaborative culture – our work would never have been possible without many generous contributors before us, and we hope this project has been a small but meaningful contribution to the deep learning space.

Special thanks for Professors Lyle Ungar and Konrad Kording who taught this Deep Learning course, and TA Pooja Counsul for her guidance throughout the semester.

References

Code Sources

- VAE: helper functions and architecture adapted from Week 8 Tutorial 1 of CIS522
- GAN:
    - training code adapted from Week 8 Tutorial 2 of CIS522 as well as MidiNet (https://github.com/annahung31/MidiNet-by-pytorch)
    - feature matching, label smoothing code adapted from MidiNet
- Next-note prediction:
    - Training code adapted from https://github.com/Skuldur/Classical-Piano-Composer
    - RNN language model evaluation and multinomial code adapted from Week 9 Tutorial 2 of CIS522
- Pre-processing Lakh Midi Dataset
    - Code adapted from https://nbviewer.jupyter.org/github/craffel/midi-dataset/blob/master/Tutorial.ipynb
    - Code to convert from MIDI to music21 adapted from https://github.com/Skuldur/Classical-Piano-Composer

Data Sources

- Lakh Pianoroll Dataset: https://salu133445.github.io/lakh-pianoroll-dataset/
- Genre Labels for songs in the Million Song Dataset: https://www.tagtraum.com/msd_genre_datasets.html

References

- https://ai.plainenglish.io/building-a-lo-fi-hip-hop-generator-e24a005d0144
- https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-68786834d4c5
- https://towardsdatascience.com/creating-a-pop-music-generator-with-the-transformer-5867511b382a
- https://towardsdatascience.com/practical-tips-for-training-a-music-model-755c62560ec2
- https://towardsdatascience.com/a-multitask-music-model-with-bert-transformer-xl-and-seq2seq-3d80bd2ea08e
- https://towardsdatascience.com/how-to-remix-the-chainsmokers-with-a-music-bot-6b920359248c
- MuseGAN: https://arxiv.org/abs/1709.06298
- MidiNet: https://arxiv.org/abs/1703.10847
- https://github.com/ashishpatel26/Best-Audio-Classification-Resources-with-Deep-learning#dl4m-details