LSTM-Based Classical Piano Composition

Oliver Li, Shuhao Jiang, Guo Chen, Xindong Zhou
College of Engineering, Boston University, Boston, MA 02215
wsoxl@bu.edu, shuhaoj@bu.edu, bcsy@bu.edu, zxd10032@bu.edu

April 2025

1 Task

The goal of this project is to train a neural network to generate classical piano compositions. The model will learn from existing classical piano pieces and generate new compositions that resemble the style of composers such as Bach, Beethoven, and Mozart.

We aim to explore and develop music generation based on multiple models to achieve some of following capabilities after complete mimicking the style of different composers which includes:

- **Extend an existing melody**:

  Give an input sequence (the first few notes of a Beethoven piece), the AI will continue composing the rest of the melody in a coherent manner.

- **Generate original piano melodies**:

  The LSTM model will generate entirely new piano melodies based on learned patterns from classical compositions.

Challenges include:

- Modeling long-term dependencies in music sequences.
- Representing musical structures such as harmony, rhythm, and dynamics.
- Handling MIDI file processing and feature extraction for training.

2 Related Work

In addition to LSTM and Transformer-based models, Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs) have also been applied to symbolic music generation. Mogren introduced *C-RNN-GAN*, a recurrent GAN framework that produces sequences of real-valued MIDI events and learns temporal musical structure through adversarial training [1]. Yang et al. developed *MidiNet*, a conditional GAN model using convolutional layers for melody generation, allowing stylistic control and structure [2]. On the VAE side, Roberts et al. proposed *MusicVAE*, a hierarchical variational autoencoder capable of modeling long-term dependencies in musical sequences and interpolating between musical phrases in latent space [3]. These models offer powerful alternatives to RNNs and Transformers by emphasizing latent space structure and generative diversity.

3 Approach

The project uses these key models to evaluate on music generation, each model will be trained on the same dataset and evaluated with identical metrics:

3.1 LSTM

We use a multi-layer LSTM to capture long-range dependencies while mitigating the vanishing gradient problem inherent in vanilla RNNs. The LSTM model is then tested on the MAESTRO dataset.

3.1.1 Layer Formation

The model consists of multiple layers:

- **Embedded Layer:** We create embedded tokens for each note and represent them into a vector space.
- **LSTM Layer:** We build 2 LSTM layers with a hidden layer size of 256.
- **Dropout Layer:** The Dropout layer is applied between layers to reduce overfitting.
- **Loss Function:** MLE loss to optimize sequence prediction. As the final output is linear, we cannot use Cross-Entropy Loss here.
- **Optimization:** Adam optimizer with scheduled learning rate decay.

3.1.2 Test Results and Future Work

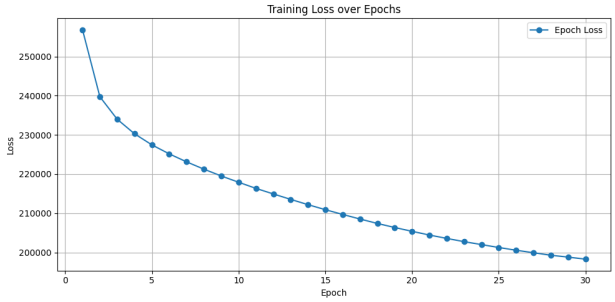

Figure 1: Loss over iterations for a multi-layer LSTM model

Although achieving a converging total training loss over epochs, the perplexity remains high at over 21, and the average Validation loss is 3.1405. Hence the next step is to provide improvement to the structure, testing different layers and learning rate for the model. The single-layer LSTM also performs bad resulting in a perplexity over 31.

3.2 RNN

While Long Short-Term Memory(LSTM) networks are widely used for music generation due to their ability to mitigate the vanishing gradient problem and capture long-range dependencies, we also implement a simpler RNN as a comparison, which allows us to compare and evaluate how LSTM improves performance in AI composition as an improved version of RNN.

3.2.1 Layer Formation

We train the model using cross-entropy loss and the Adam optimizer with the following layers:

- **Embedding Layer:**Each pitch is converted to a readable vector representation.
- **RNN Layer:**A 3-layer unidirectional RNN with hidden size 128 models temporal relationships
- **Dropout Layer:**Applied after the RNN to prevent overfitting
- **Output Layer:**A linear layer projects the final hidden state to the proper size, and we use softmax layer to produce a probability distribution over the next note.

During generation, the model is seeded with a fixed-length melody fragment and produces new notes one step at a time.

3.2.2 RNN Architecture Potential Enhancements

To explore the full potential of tradition RNN in the composition of AI, there are several ways to improve the model in future work:

- **Multi-feature input:**Compared with the traditional one using pitch only, we could incorporate pitch, duration, and velocity as separate independent input features.
- **Independent embedded layers:**Each input feature is embedded separately and connects into a unified input vector.

- **Layer normalization:** We could apply a normalization to the concatenated embedded layer to stabilize the input distribution.
- **Residual connections:** The Residual connections could be added between RNN layers to improve gradient flow in deeper stacks.

3.3 Variational Autoencoder (VAE)

We implement a Conditional Variational Autoencoder (CVAE) to generate pitch sequences conditioned on composer style. The model learns to reconstruct musical patterns while modeling stylistic variations in a continuous latent space.

3.3.1 Model Architecture

The CVAE consists of an encoder that takes a pitch sequence and composer label (one-hot) to produce latent mean and variance vectors. Using the reparameterization trick, a latent vector is sampled and passed to the decoder along with the same label to reconstruct the original sequence.

The loss function includes **binary cross-entropy** for reconstruction and **KL divergence** for latent regularization:

$$\mathcal{L} = \text{BCE}(x, \hat{x}) + \text{KL}(q(z|x, c)||p(z))$$

3.3.2 Training and Generation

MIDI files from the MAESTRO dataset are converted into normalized pitch sequences. We train on data from the top 10 composers, using fixed-length input windows of 64 notes. Training is performed on GPU over 15 epochs using the Adam optimizer.

To generate music, we sample a latent vector and decode it with a chosen composer label. The output pitch sequence is converted to MIDI using PrettyMIDI.

3.3.3 Future Work

Future improvements include adding rhythm and velocity features, extending sequence length, and exploring Transformer-based decoders for better structure.

3.3.4 Training Pipeline

The training process includes:

- **Data Preprocessing:** Parsing MIDI files to extract pitch, duration, velocity, and timing features.
- **Sequence Encoding:** Converting MIDI notes into integer-based sequences, normalized for consistency.
- **Training:** Using mini-batch training with GPU acceleration to optimize sequence modeling.
- **Generation:** Sampling an initial note sequence and predicting subsequent notes iteratively to create full-length compositions.
- **CVAE Training:** A VAE architecture is trained on grouped composer data to learn conditional latent representations and generate stylistically matched compositions.

3.4 GAN

The model consists of a generator G and a discriminator D, both conditioned on composer labels.

The generator takes a noise vector $z \in R^d$ and a label embedding $E_y(y) \in R^d$, concatenates them into $x \in R^{2d}$, and uses a multi-layer perceptron (MLP) with three `Linear` layers and `ReLU`/`Sigmoid` activations to produce a piano-roll of shape $[96, 88]$.

The discriminator flattens the piano-roll input, adds a label embedding of the same shape, and processes it through an MLP consisting of three `Linear` layers with `LeakyReLU` and `Sigmoid` activations to output a scalar probability.

3.4.1 Dataset and Metric

To train the GAN effectively, symbolic music data is converted into a structured piano-roll representation. The `midi_to_pianoroll` function transforms a MIDI file into a binary matrix of shape $[T, 88]$, with T time steps and 88 standard piano keys (MIDI 21–108).

3.4.2 Preliminary Results

To prevent the discriminator from dominating training, we apply label smoothing by adding noise to real samples. During early epochs (1–5), generator and discriminator losses fluctuated widely (20–100), reflecting unstable adversarial dynamics. In later epochs (10–20), losses began to stabilize or decline, though oscillations remained.

The cosine similarity (CosSim) occasionally exceeded 0.2 but typically hovered around 0.15. While useful as a rough alignment metric, CosSim alone does not reflect musical quality or stylistic fidelity..

Figure 2: Loss over iterations for GAN model

3.5 Tools and Libraries

- **Music21 [4] or Cubase**: For MIDI file parsing and feature extraction.
- **TensorFlow/Keras or PyTorch**: Framework for LSTM training and model implementation.
- **Fluidsynth [5]**: Converts MIDI output to WAV for audio playback.
- **PrettyMidi [6]**: Reads MIDI files and outputs the notes to pitch, instrument, and duration.

4 Dataset and Metrics

4.1 Dataset

The project will use the following datasets:

- **MAESTRO [7]**: A dataset of MIDI-aligned piano performances. It consists of 1276 music pieces from 51 different composers (Including duo composers).
- Custom-curated classical piano MIDI files.

4.2 Metrics

Model performance will be evaluated using:

- **Perplexity**: Measures sequence prediction confidence.
- **Style Similarity**: Compares generated pieces to training data using cosine similarity of feature embeddings.
- **Human Evaluation**: Expert musicians rate composition quality.

4.3 Preliminary Code

Please find the code in the SCC directory:
"/projectnb/ec523/projects/Proj_MusicGen"

5 Detailed Timeline and Roles

Task	Deadline	Who
Database Preprocessing	Week 1	All members
LSTM model design	Week 3	Oliver
RNN model design	Week 3	Shuhao
GAN model design	Week 3	Guo
VAE model design	Week 3	Xindong
Transformer model design	Week 5	Oliver
Hyperparameter tuning	Week 6	All members
Music generation and analysis	Week 5	Shuhao
Report and presentation	Week 6	All members

References

[1] Olof Mogren. C-rnn-gan: Continuous recurrent neural networks with adversarial training. *arXiv preprint arXiv:1611.09904*, 2016.

[2] Li-Chia Yang, Szu-Yu Chou, and Yi-Hsuan Yang. Midinet: A convolutional generative adversarial network for symbolic-domain music generation. In *Proceedings of the International Society for Music Information Retrieval (ISMIR)*, 2017.

[3] Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. A hierarchical latent vector model for learning long-term structure in music. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

[4] Michael Cuthbert and Christopher Ariza. music21: A toolkit for computer-aided musicology and symbolic music data processing. *Proceedings of the International Society for Music Information Retrieval (ISMIR)*, 2010.

[5] Peter Hanappe et al. Fluidsynth: A software synthesizer based on the soundfont 2 specification. *Proceedings of the Linux Audio Conference*, 2000.

[6] Colin Raffel and Daniel PW Ellis. An intuitive interface for music transcription with deep learning. In *Proc. of the 15th International Society for Music Information Retrieval Conference (ISMIR)*, pages 639–644, 2014.

[7] Curtis Hawthorne, Ian Simon, Anna Huang, et al. Maestro: A dataset for music generation and automatic transcription. *arXiv preprint arXiv:1810.12247*, 2018.