THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Deep Learning
Final Project
March 18, 2024
Amir Jafari
Due: *Check the Syllabus*

Music Generation with Deep Neural Networks

Project Report (team 7)

for the course

DATS 6303 'Deep Learning'

Spring 2024

Team Members:

Aman Jaglan

Aneri Patel

Smit Pancholi

Tushar Sharma

Table of Content

Introduction

Music generation through artificial intelligence creates a unique intersection of creativity and computation, pushing the boundaries of both technological innovation and artistic expression. This project seeks to explore how music and machine learning can intersect by utilizing advanced deep learning techniques to autonomously generate melodies. The initiative is anchored in the history of computational creativity and is particularly motivated by recent advances in deep learning that have significantly enhanced the quality and complexity of generated music.

The project's framework is based on time series prediction, which treats music generation as a sequential process where future notes are predicted based on preceding sequences. This method is especially suitable for music, where recognizing and predicting patterns is key to composition. The project utilizes the music21 Python library, which provides robust tools for music analysis and manipulation. This library allows for the exploration of musical structures and their transformation into data formats that are compatible with machine learning models.

The deep learning component of the project is powered by PyTorch, a versatile and powerful machine learning library that supports the fast development and modification of complex neural network architectures. PyTorch's capability for dynamic computation graphs allows for the creation of customized architectures that can handle the specific demands of musical data. The project investigates several neural network designs, including a GRU (Gated Recurrent Unit) with an integrated attention layer, designed to enhance prediction accuracy by focusing on critical parts of the input sequence. A Variational Autoencoder (VAE), known for its effectiveness in generating new data samples that embody learned characteristics. A multi-head attention LSTM (Long Short-Term Memory), which employs several attention mechanisms to more effectively understand the connections between different music segments.

To display the generated music scores and support interactive exploration, MuseScore is utilized. MuseScore offers a visual representation of the music, simplifying the evaluation of the generated melodies' aesthetic qualities. To further improve accessibility and user interaction, a web application interface is developed using Streamlit. This interface permits users to directly interact with the model by providing inputs and receiving generated music in real time.

Project flow

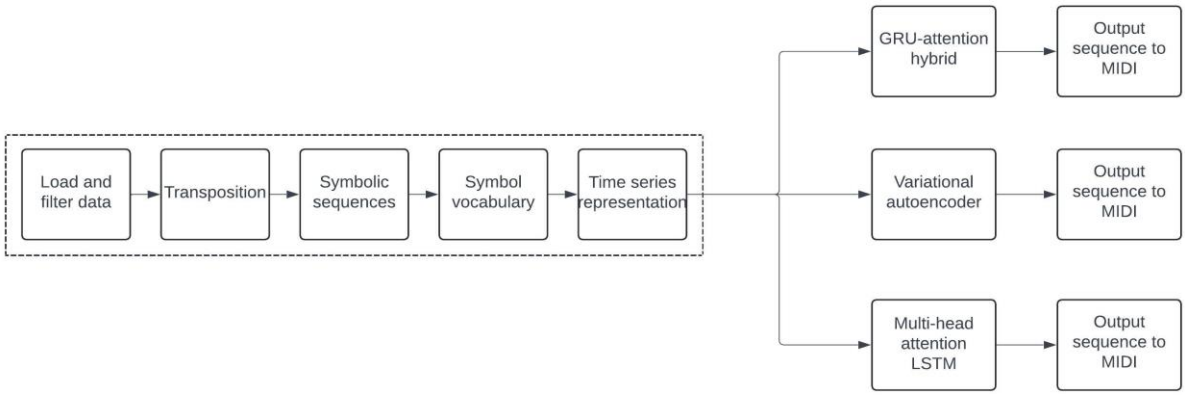


Figure 2. Schematic block diagram for the project workflow.

Music theory fundamentals

- Pitch: The perceived highness or lowness of a sound.
- Key/Tonic: The central note and its corresponding scale, establishing a sense of musical home.
- Monophonic/Polyphonic Melody: A monophonic melody is a single line of notes, while polyphonic melodies feature multiple independent melodies played simultaneously.
- Tempo: The speed of the music measured in beats per minute (BPM).
- Time Signature: A rhythmic structure organizing beats into measures, indicated as a fraction on the score (e.g., 4/4, 9/8, etc).

Dataset Description

The dataset comprises over 5,000 melodies from the ESAC collection [1], specifically focusing on German folk songs. This extensive compilation offers a broad spectrum of traditional German music, serving as a vital resource for the study and analysis of folk melodies. The ESAC (European Song Archive Collection) datasets are distinguished by their structured approach to musical archiving, where each melody is precisely encoded to facilitate digital analysis. This allows researchers and enthusiasts to thoroughly examine musical patterns, historical evolutions, and cultural significances across different pieces.

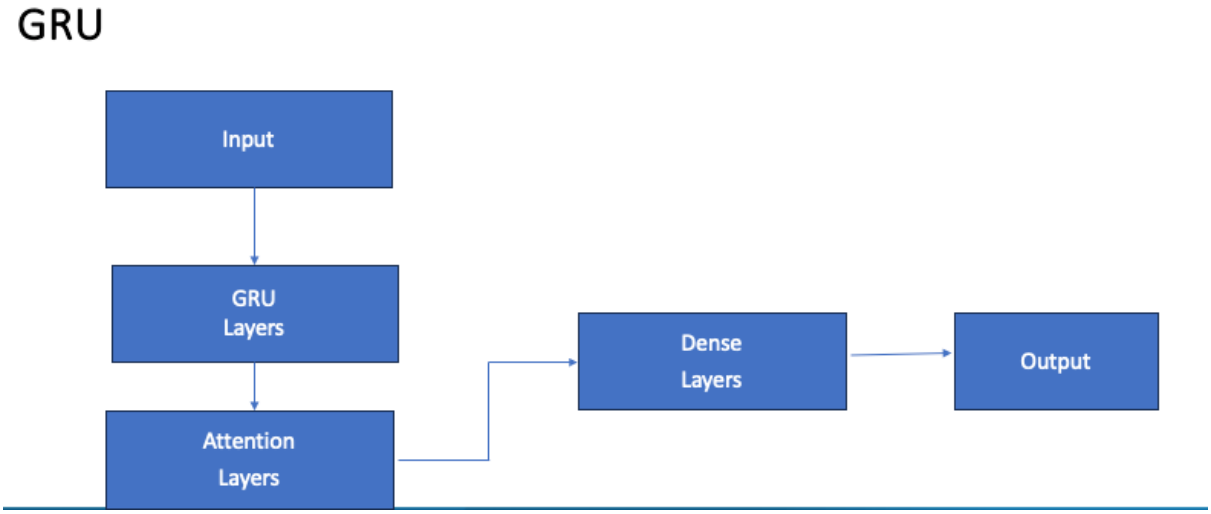
The melodies in this dataset are annotated with musical and contextual metadata, which includes information on the song's origins, lyrics, and historical background. This makes the dataset a comprehensive repository of melodies and a crucial source of ethnomusicological data.

DL Network and Training Algorithm

1. Gated Recurrent Units

Background Information:

The MusicModel is a sophisticated deep learning model designed primarily for processing and generating musical data. It integrates several advanced neural network techniques, including bidirectional Gated Recurrent Units (GRUs), an advanced attention mechanism, and multiple layers of processing. These components make the model highly effective for tasks that involve complex sequential input, such as music, where understanding both the temporal dynamics and the contextual nuances is crucial.



Network Architecture:

1. GRU Layer

Purpose: Captures both long-term and short-term dependencies within the input sequence, crucial for modeling time-related dynamics in data like music.

Architecture Detail: The model employs a bidirectional GRU, which processes data both forwards and backwards, providing a richer contextual understanding.

Equation:

For each gate in GRU:

$$
\begin{aligned}
r_t &= \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr}) \\
z_t &= \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz}) \\
n_t &= \tanh(W_{in}x_t + b_{in} + r_t \odot (W_{hn}h_{(t-1)} + b_{hn})) \\
h_t &= (1 - z_t) \odot n_t + z_t \odot h_{(t-1)}
\end{aligned}
$$

Dropout (0.5) is applied within the GRU to enhance regularization by randomly omitting units from the learning process during training.

2. Advanced Attention Mechanism :

Purpose: Dynamically focuses the model on the most informative parts of the input sequence to enhance feature extraction for subsequent prediction tasks.

Architecture Detail:

attention_fc: Transforms input features into an intermediate attention space.

value_fc: Provides a complementary transformation used to enrich the attention process.

query_fc: Calculates final attention scores for weighting input features.

Equation:

$$
\begin{aligned}
\text{Attention Scores} &= \text{query\_fc}(\tanh(\text{attention\_fc}(x) + \text{value\_fc}(x))) \\
\text{Attention Weights} &= \text{softmax}(\text{Attention Scores}, \text{dim} = 1) \\
\text{Weighted Sum} &= \sum(x \odot \text{Attention Weights})
\end{aligned}
$$

The attention scores determine how much weight each part of the input should receive, and the softmax function normalizes these scores to ensure they sum to one, allowing them to be used as a probabilistic measure of importance.

3. Batch Normalization

Purpose: Normalizes the outputs of the attention mechanism to improve the stability and speed of training by reducing internal covariate shift.

Architecture Detail: Applied to the batch dimension of the processed data.

Equation:

$$y = \frac{x - \mu_{\text{batch}}}{\sqrt{\sigma^2_{\text{batch}} + \epsilon}}\gamma + \beta$$

Where μbatch and σ2batch are the mean and variance of the batch, γ andβ are learnable parameters, and ϵ is a small constant for numerical stability.

4. Dense Layers and Dropout

Purpose: Further process the features to map them to the desired output space, applying non-linear transformations and regularization.

Architecture Detail:

dense1: Maps attention-processed features to a reduced feature space.

dense2: Further reduces feature dimensionality.

dense3: Outputs the final predictions.

Dropout (0.6): Applied after dense1 to prevent overfitting.

Equations:

Each dense layer applies:

$$y = Wx + b$$

ReLU activations are used between layers to introduce non-linearity:

$$\text{ReLU}(x) = \max(0, x)$$

Training Algorithm

Initialization: Initialize all weights using a strategy like He initialization, which is suitable for layers followed by ReLU activations.

Optimization: Use the Adam optimizer for its adaptive learning rate capabilities, which helps in dealing with the sparse gradients often encountered in recurrent network training.

Loss Function: Employ cross-entropy loss for classification tasks or mean squared error for regression tasks, depending on the specific objectives of the music processing task.

Regularization: Implement dropout within the GRU layers and after dense layers to prevent overfitting and to promote generalization across unseen musical data.

Epochs and Batches: Train the model for several epochs, using mini-batches of data to optimize the memory usage and computational efficiency. Regularly evaluate the model on a validation set to monitor the training progress and adjust hyperparameters as needed.

2. Variational Autoencoder (VAE).

Description of the Deep Learning Network: Variational Autoencoder (VAE)

Background Information

A Variational Autoencoder (VAE) is a generative model that learns to represent high-dimensional data in a lower-dimensional latent space, and then reconstructs the original data from this compressed representation. VAEs differ from traditional autoencoders by introducing a probabilistic approach to the encoding process, where the encoder outputs parameters to a probability distribution representing the data. This probabilistic approach enables the model to generate new data points by sampling from the latent space.

Network Architecture

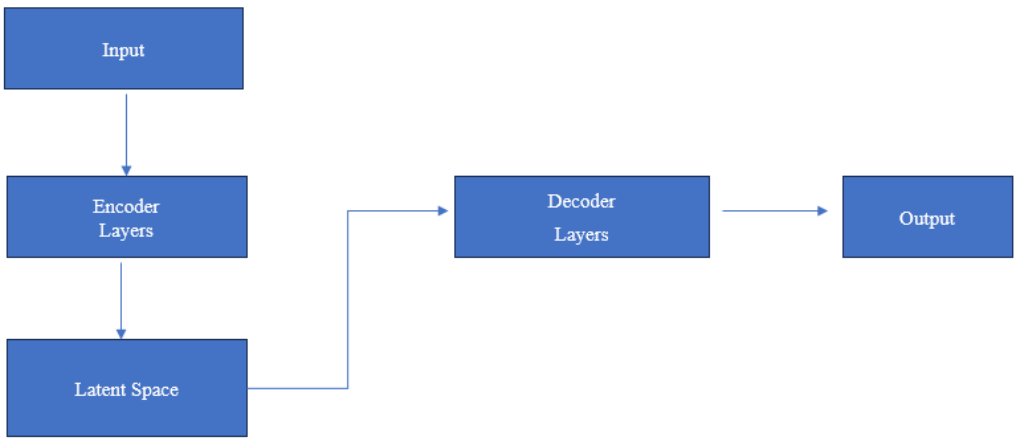


Figure 2. Network Architecture of VAE.

The VAE is comprised of two main parts: the encoder and the decoder.

Encoder:

Purpose: Maps the input data to a probability distribution in the latent space.

The encoder in the network transforms the input data into two separate latent space representations: mean (μ) and log variance ($\log(\sigma^2)$). These parameters define a Gaussian distribution from which latent variables are sampled. This stochastic process is facilitated by the reparameterization trick, which maintains the differentiability of the model by expressing the random variable as a deterministic variable plus noise. The

sampled latent variables are subsequently processed by the decoder, which aims to accurately reconstruct the original input data.

Components:

fc1 and fc2: These are fully connected (dense) layers that progressively transform the input into a higher abstraction, capturing complex relationships in the data.

relu1 and relu2: Non-linear activation functions that introduce non-linearities into the model, allowing it to learn more complex patterns.

dropout1: A regularization technique that helps prevent overfitting by randomly dropping units during training.

fc21 and fc22: These layers output the mean (μ) and the logarithm of the variance ($\log(\sigma^2)$) of the latent variables, respectively.

Equation: Mean (μ) and variance (σ^2) are computed as:

$$\mu = \text{fc21}(h), \quad \log(\sigma^2) = \text{fc22}(h)$$

Where h, is the output of the previous hidden layer.

Reparameterization Trick:

Purpose: Allows the backpropagation of gradients directly through the stochastic sampling, making it feasible to optimize the parameters of the probability distribution using gradient descent.

Equation:

$$z = \mu + \sigma \odot \epsilon, \quad \text{where } \epsilon \sim \mathcal{N}(0, I)$$

Decoder:

Purpose: Reconstructs the input data from the latent variables.

Components:

fc3 and fc4: Dense layers that transform the latent variables back into the space of the original data.

relu3 and relu4, dropout2: Similar to the encoder, these layers apply non-linear transformations and regularization.

fc5: The final layer that outputs the reconstruction of the original data.

Output activation (torch.sigmoid): Ensures the output values are in a valid range (e.g., 0-1 for normalized data).

Equation:

$$\hat{x} = \sigma(\text{fc5}(h'))$$

Where h′, is the output of the last hidden layer in the decoder, and

σ is the sigmoid function.

Loss Function

The primary objective during training is to minimize the reconstruction loss, typically measured by binary cross entropy, and a regularization term known as the Kullback-Leibler divergence (KLD). The KLD imposes a penalty on the deviation of the learned distribution in the latent space from a standard Gaussian distribution, effectively controlling the distribution of latent variables.

Loss = BCE+β×KLD, where β is a hyperparameter that balances the two loss components.

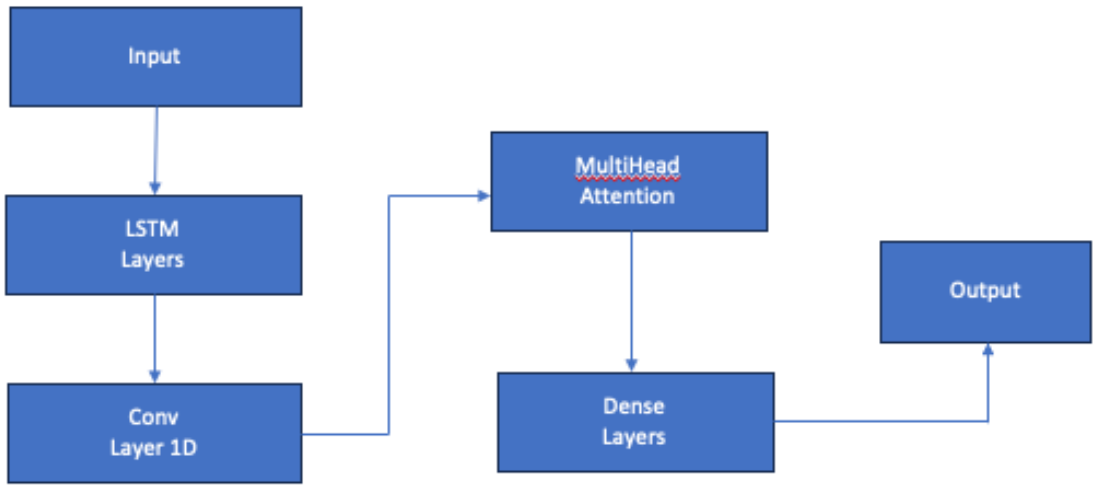
The overall loss function is a weighted sum of these two terms, where the weight, referred to as beta (β), balances the two competing aspects of the training objective. The output of the VAE is the reconstructed version of the original input data, produced by the decoder. The quality of the output would be assessed based on how closely the generated melodies resemble the original melodies in terms of structure, rhythm, and harmony, among other musical attributes.

3. LSTM with Multi Head Attention

Description of the Deep Learning Network: MusicModel

Background Information:

The MusicModel is a sophisticated neural network that combines several state-of-the-art techniques in deep learning to process sequential data, specifically designed for music. The architecture leverages the strengths of Long Short-Term Memory (LSTM) networks, convolutional neural networks (CNNs), attention mechanisms, and dense layers, making it highly effective for tasks that require understanding the context and nuances in music sequences.



Network Architecture:

LSTM Layers:

Purpose: To capture long-range dependencies within the music data, crucial for maintaining the temporal dynamics of melodies and rhythms.

Architecture Detail: The model uses a bidirectional LSTM, allowing it to gather context from both past and future data points within a sequence. This is particularly useful in music where the anticipation of upcoming notes and the resonance of previous notes both influence the current note.

Equation:

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{(t-1)} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho}) \\
c_t &= f_t \odot c_{(t-1)} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

Dropout (0.5): Applied within the LSTM to prevent overfitting by randomly omitting units during training.

Convolutional Layer (Conv1D):

Purpose: To detect local patterns such as specific rhythmic or harmonic features within the sequences.

Architecture Detail: Utilizes a kernel of size 3, which slides across the time steps to extract features from the data.

Equation:

$$y_t = \mathrm{ReLU}(w * x_t + b)$$

Multi-Head Attention Mechanism:

Purpose: To allow the model to focus on important parts of the input sequence when making predictions, mimicking how a musician might focus on different motifs or themes.

Architecture Detail: Splits the embedding into multiple 'heads', each of which performs attention independently to blend different learned aspects.

Equation:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

Dropout (0.5): Applied after attention to further regularize the model.

Dense Layer:

Purpose: To map the extracted and attended features to the desired output size, which could correspond to different potential next notes or other musical attributes.

Equation: $y = wx + b$

Layer Normalization and Residual Connections:

Purpose: To stabilize the learning and integrate information efficiently across the network.

Equation for Layer Normalization:

$$\hat{x} = \frac{x-\mu}{\sigma}$$

Equation for Residual Connection: $y=x+\text{LayerNorm}(x)$

Training Algorithm:

The MusicModel is typically trained using backpropagation through time (BPTT), a variant of the standard backpropagation used specifically for training networks that process sequential data. This involves:

Loss Calculation: Depending on the task, common choices are cross-entropy loss for classification (e.g., next note prediction) or mean squared error for regression (e.g., continuous features of music like pitch).

Optimizer: Often, an Adam optimizer is used due to its efficiency in handling sparse gradients and adaptive learning rate capabilities.

Data Preprocessing

1. Loading and Filtering the Dataset

The process began by loading musical pieces in Kern (.krn) format from a designated dataset directory. The music21 library was used to parse these files into a format the code could work with.

To simplify the learning process, songs were filtered to ensure they only contained notes of specific rhythmic durations (e.g., quarter notes, eighth notes, etc.).

2. Data Augmentation by Transposition

Each song in the dataset was transposed into all 24 keys (12 major and 12 minor). This technique augmented the data by exposing the models to the same melodies in different musical keys, helping them learn in a way that wasn't tied to a specific key.

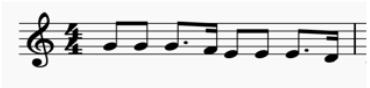
3. Symbolic Music Representation

Pitches were represented by **MIDI numbers** (here taken as strings) which range from **0** to **127**. For example, middle C (C4) on an 88-key piano was encoded as 60. Sustain of a note for a quarter length duration of time was encoded as "**_**". Rests were denoted by a specific symbol like "**r**".

The durations of the notes were quantized based on a quarter-note time step. This meant that they were rounded to the nearest multiple of the time step, making the representation more manageable. Fig. 2 shows an example of an encoded sequence.

[67 , _ , 67 , _ , 67 , _ , _ , 65 , 64 , 64 , 64]

Sequence encoding



Score

Figure 2. Sequence encoding example for a 7-note sequence.

4. Symbol Vocabulary Preparation

All the individual encoded songs were combined into a single, large dataset file. A special delimiter sequence was added between each song to help the models distinguish where one melody ended and another began.

A vocabulary was constructed containing all the unique musical symbols found within the dataset. Each symbol was then assigned a numerical ID. This mapping was saved as a Json file, as it was crucial for converting between symbolic and numerical representations of the music.

5. Time Series Data Preparation

The single dataset file was loaded, and symbolic content was converted to sequences of integers using the previously created mapping.

The long sequence of integers was divided into smaller training sequences of a fixed length. This was how the model would be fed data as a time series – it examined these short sequences and learned to predict the next note (or symbol) in the sequence using a **sliding window approach** as shown in Fig. 3.

Finally, one-hot encoding was applied to transform both the input sequences and expected outputs into a categorical format that neural networks readily worked with.

input: [60, _ 61, _ 62, _ …] target: [13]

input: [61, _ 62, _ 63, _ …] target: [14]

. .
. .
. .

Figure 3. Time series music representation based on the sliding window approach.

Experimental Setup

1.GRU

Overview

The goal of this experiment is to assess the performance of a novel music generation model that incorporates GRU layers and an advanced attention mechanism. This model, MusicModel, is designed to handle complex dependencies in musical data effectively, aiming to produce sequences that are both harmonically complex and expressive.

Model Architecture

The architecture of MusicModel reflects a strategic layering of various neural network technologies, each contributing uniquely to the model's overall capability:

Gated Recurrent Unit (GRU) Layers:

Configuration: The model includes 4 GRU layers with 128 hidden units each, a reduction from the initially considered 256 units. This choice reflects a balance between computational efficiency and the capacity to capture relevant musical patterns.

Bidirectional Processing: Each GRU layer is bidirectional, enhancing the model's ability to learn from both past and future context simultaneously—crucial for understanding the full musical structure.

Advanced Attention Mechanism:

Custom Design: Following the GRU layers, an AdvancedAttention mechanism is employed. This layer computes attention scores based on the transformed inputs, allowing the model to focus dynamically on the most relevant parts of the input sequence for predicting each subsequent note.

Components:

attention_fc and value_fc: These layers transform the input to create a new representation space where attention scores are calculated.

query_fc: Computes a single attention score for each feature, allowing the model to weigh different features based on their relevance to the output.

Batch Normalization:

Applied after the attention mechanism, this layer normalizes the activations from the GRU and attention layers, helping to stabilize the learning process and improve convergence speed.

Dense Layers and Non-linear Activation:

The model includes multiple dense layers that progressively reduce the dimensionality from the high-dimensional feature space to the output size, which matches the number of unique MIDI notes in the dataset.

Dropout: A dropout rate of 0.6 is applied after the first dense layer to prevent overfitting by randomly omitting features during training.

Training and Operational Details

Loss Function: The model uses a Cross-Entropy Loss, ideal for classification tasks where each output can be treated as a discrete class (MIDI note).

Optimizer and Learning Rate: An Adam optimizer with a learning rate of 0.0005 is used for training, providing an adaptive learning rate mechanism that helps in fine-tuning the weights more effectively.

Epochs and Batch Size: Training is conducted over 8 epochs with a batch size of 100, optimizing the trade-off between learning speed and model stability.

Data Handling and Processing

Data Representation: Input data is preprocessed into sequences of MIDI notes, where each note is encoded into a one-hot vector using mappings provided by mapping.json.

Output Configuration: The output units of the model are set to the size of the dataset's unique MIDI notes, as determined from the mapping file, ensuring that the model can output any note in the dataset.

2. Variational Autoencoder

The experimental setup for the Variational Autoencoder (VAE) designed for melody generation involved a meticulous process to ensure effective training and validation of the model. Data consisting of preprocessed musical sequences was used to train the network. This data was systematically encoded into a numerical format that reflected musical information such as pitch and rhythm, which the VAE learned to encode and decode.

The network was implemented using the PyTorch framework, a choice driven by its flexibility and efficiency in handling dynamic computational graphs, which are essential for implementing the stochastic gradient descent necessary for training VAEs. The architecture of the VAE included layers for encoding the input into a latent space and subsequently decoding it to reconstruct the input data. The training process was operationalized by first initializing the network weights using the Xavier uniform method, enhancing the model's ability to converge faster and more reliably during training.

Training parameters, such as the learning rate and batch size, were determined based on preliminary experiments and literature benchmarks. The initial learning rate was set at 0.005, with adjustments made via a learning rate scheduler that reduced the rate by a factor of 0.1 every 10 epochs to fine-tune the network's training as it progressed. This method helped in stabilizing the learning process in later stages of training when finer adjustments to weights were necessary.

The performance of the model was primarily judged by the loss during training, which comprised two parts: the binary cross-entropy loss that measured the difference between the original and reconstructed data, and the Kullback-Leibler divergence that ensured the distribution of the latent variables was as expected. A combination of these metrics provided a comprehensive view of both the accuracy of reconstruction and the effectiveness of the latent space modeling.

To detect and prevent overfitting, several strategies were implemented. Dropout layers were included in both the encoder and decoder to introduce regularization by randomly omitting a subset of features during each training pass. This approach encouraged the model to learn more robust features that generalized better to new data.

Moreover, the experiment included mechanisms to handle extrapolation by ensuring that the model did not generate implausible melodies. This was managed by sampling the latent space carefully and ensuring that the distribution parameters (mean and variance) learned by the model did not deviate significantly from those seen during training, thus maintaining the integrity and musicality of the generated outputs.

3. LSTM with Multihead Attention

Overview

In this study, we evaluate a sophisticated neural network architecture designed for the task of music generation. The architecture, named MusicModel, employs a hybrid approach combining Long Short-Term Memory (LSTM) units, convolutional layers, and multi-head attention mechanisms. This combination aims to leverage the strengths of each component to enhance the model's ability to understand and generate complex musical sequences with high fidelity to the training data.

Model Architecture Details

The MusicModel consists of multiple layers, each tailored to process different aspects of musical information:

LSTM Layers: The core of the model is formed by eight LSTM layers, each with 256 hidden units. These layers are bidirectional, meaning they process the input data in both forward and reverse directions to better capture the context surrounding each note in a sequence. This bidirectional approach is particularly effective in modeling music, where the context of a note can significantly influence its interpretation. To mitigate the risk of overfitting, a dropout rate of 0.5 is applied between LSTM layers, randomly omitting a portion of the features during the training phase.

Convolutional Layer: Following the LSTM layers, a 1-dimensional convolutional layer with 256 filters of size 3 is applied. This layer's purpose is to detect local patterns and relationships in the data that may span several time steps, which are essential for recognizing rhythmic and textural motifs in music. The convolutional layer is complemented by a subsequent dropout layer and layer normalization, which standardize the outputs to have zero mean and unit variance, thus stabilizing the learning process.

Multi-Head Attention Mechanism: To enhance the model's ability to focus on relevant parts of the input sequence when making predictions, a multi-head attention mechanism with 8 heads is integrated. This component allows the model to attend to different segments of the input sequence simultaneously, which is crucial for capturing the complex structure of musical compositions where multiple voices or instruments interact. The attention mechanism processes the outputs of the LSTM and convolutional layers, dynamically weighting the significance of each part of the sequence in the generation of the next note.

Output Layers: The final part of the network consists of a linear transformation followed by another dropout layer. This configuration processes the attended features to generate the output sequence. The output layer specifically converts the high-dimensional feature representations into the final sequence of music notes, represented as MIDI values.

Training and Implementation

The model is trained using a cross-entropy loss function, which is standard for classification tasks where each output from the model can be treated as a class label—in this case, specific MIDI notes. The optimizer used is Adam, with a learning rate of 0.0001, which is chosen for its effective handling of sparse gradients and its adaptability in learning rates across different parameters. Training is conducted over 20 epochs with a batch size of 100, balancing the need for computational efficiency and the benefit of gradient averaging.

Data Handling

Input to the model consists of MIDI note sequences derived from a dataset of music pieces, preprocessed and encoded into a format suitable for neural network training. A JSON file, mapping.json, is used to map between MIDI note numbers and their corresponding indices in the model's output layer, ensuring that each note is appropriately represented in both the input and the target sequences.

4. Stacked bidirectional LSTM architecture building and training

Framework Shift: The most fundamental change is the transition from TensorFlow (Keras) to PyTorch. This brings a different approach to defining model architecture, handling data, optimization, etc.

Bidirectional LSTMs: The core model architecture now employs bidirectional LSTMs, which process the input sequence both in the forward and reverse directions. This potentially allows the model to better capture long-range dependencies in musical sequences.

Stacked Layers: The model has been made deeper with a stacked LSTM architecture. Two LSTM layers are used, with the output of the first layer feeding into the second. This can help the model learn more complex hierarchical representations of the music.

1. Setup

• Constants: Defines parameters like hidden layer sizes (NUM_UNITS), loss function (nn.CrossEntropyLoss), learning rate, epochs, batch size, and the model save path.

• Device Check: Checks if a GPU (CUDA) is available and uses it for training if so. Otherwise, defaults to CPU.

3. Dataset Class

• MusicDataset: A PyTorch Dataset class is defined to handle the input and target sequences. It provides methods for getting the length (_len) and retrieving individual samples (getitem_). The snippet for the MusicDataset class is shown in Fig. 4.

4. Model Architecture (LSTMModel)

• Bidirectionality: Each LSTM layer explicitly sets the bidirectional=True parameter.

• Stacking: Two consecutive nn.LSTM layers are used, creating the stacked architecture. The output of the first layer is passed as input to the second.

• Dropouts: Dropout layers (nn.Dropout) are added after each LSTM layer to help prevent overfitting.

• Output Layer: A linear layer (nn.Linear) maps the final hidden state to the output units, corresponding to the size of the music vocabulary.

5. Model Building and Training Loop

- build_model: Helper function to instantiate the LSTMModel, define the loss function (nn.CrossEntropyLoss) and optimizer (torch.optim.Adam). The model is moved to the appropriate device (GPU in this case).
- **Train Function:**
- Loads input and target sequences (assumed to already be PyTorch Tensors).
- Creates the MusicDataset and a DataLoader to handle mini-batches.
- Calls build_model to get the model, its loss function, and the optimizer.
- Iterates for the specified number of epochs:
- Processes mini-batches from the DataLoader.
- Moves data to the correct device.
- Zeros gradients, calculates output, loss, and performs backpropagation.
- Updates weights with the optimizer.
- Prints the loss for monitoring.
- Saves the trained model state to SAVE_MODEL_PATH

Results

Through a series of experiments involving three distinct music generation models, we have observed diverse capabilities and results. The first model, utilizing basic neural network principles, demonstrated an ability to generate melodies with clear scale ascension and descents, hinting at an understanding of basic musical structures but limited by a tendency for repetitiveness and predictability. Notable was the model's use of repetitive motifs, which, while providing thematic consistency, also suggested a potential lack of creative depth in the generated sequences. The visual representation on the music sheet highlighted rhythmic variations that added dynamism to the compositions, although these often adhered to simpler, more predictable patterns.

The second model, a Variational Autoencoder (VAE), took a more advanced approach by incorporating latent space representations to predict and generate musical sequences. This model showcased enhanced rhythmic development and harmonic complexity, managing to introduce accidentals and modal interchanges that suggested a nuanced understanding of tonality. The structural coherence and emotional impact of the compositions were notably improved, with melodies that traversed diverse emotional landscapes, offering both engagement and unpredictability. However, the dependency on the seed melody's quality and the model's struggle with maintaining musical coherence over extended compositions marked critical areas needing refinement.

The third model combined advanced attention mechanisms with GRU layers, aiming to focus more precisely on relevant parts of the input data. This approach resulted in music that, while structurally coherent in short bursts, often failed to develop more complex musical narratives over longer sequences. The model's performance highlighted the ongoing challenge of balancing novelty and musicality, where the output sometimes bordered on randomness rather than deliberate creative expression.

The fourth model was able to capture some of the more complex musical ideas in the training data, but was not able to correctly stay in key and changed key from B minor to C major, and then G major towards the end. This could be due to the nature of preprocessing (especially the bypassing of transposition) and model architecture.

Conclusion

Collectively, these experiments with different models underscore the vast potential of machine learning in music generation, alongside significant challenges that remain. Each model brought us closer to replicating the nuanced craft of human-composed music, yet each also revealed critical limitations, particularly in areas like long-term structural development, emotional expressiveness, and creative unpredictability. The repetitive patterns and reliance on simpler musical forms suggest a need for models to learn deeper, more abstract aspects of musical composition. Moreover, the consistent issue across models—their dependence on the quality of input for generating compelling music—calls for more robust training datasets and potentially more sophisticated neural network architectures.

Improvements could include integrating techniques that allow for more user control over the generative process, enhancing the ability to specify desired musical styles or thematic elements. Additionally, exploring hybrid models that combine the strengths of LSTM, VAE, and attention mechanisms might yield a more versatile tool capable of handling the complex layers of musical creativity. Ultimately, the journey towards an ideal music generation model remains a vibrant field of exploration that blends technology, creativity, and a deep understanding of musical artistry.

References:

[1] MuseScore. (n.d.). Retrieved April 26, 2024, from https://musescore.com/

[2] Cai, B., Yang, S., Gao, L., & Xiang, Y. (2023). Hybrid Variational Autoencoder for Time Series Forecasting. Knowledge-Based Systems, 281, 111079. https://doi.org/10.1016/j.knosys.2023.111079

[3] Kingma, D. P., & Welling, M. (2019). An Introduction to Variational Autoencoders. Foundations and Trends in Machine Learning, 12(4), 307-392. https://doi.org/10.1561/2200000056

[4] Yu, T., & Zhu, H. (2020). Hyper-Parameter Optimization: A Review of Algorithms and Applications. https://doi.org/10.48550/arXiv.2003.05689

[5] Odaibo, S. (2019). Tutorial: Deriving the Standard Variational Autoencoder (VAE) Loss Function. https://doi.org/10.48550/arXiv.1907.08956

[6] ESAC Folk Song Dataset

[7] Bank, D., Koenigstein, N., & Giryes, R. (2021). Autoencoders. https://doi.org/10.48550/arXiv.2003.05991

[8] Wu, J., Hu, C., Wang, Y., Hu, X., & Zhu, J. (2019). A Hierarchical Recurrent Neural Network for Symbolic Melody Generation. IEEE Transactions on Cybernetics, 50(12), 3751 3761. https://doi.org/10.1109/TCYB.2019.2953194

[9] Lu, P., Tan, X., Yu, B., Qin, T., Zhao, S., & Liu, T.-Y. (2022). MeloForm: Generating Melody with Musical Form based on Expert Systems and Neural Networks. https://doi.org/10.48550/arXiv.2208.14345

[10] Dey, R., & Salem, F. M. (Year). Gate-Variants of Gated Recurrent Unit (GRU) Neural Networks. Circuits, Systems, and Neural Networks (CSANN) LAB Department of Electrical and Computer Engineering, Michigan State University. East Lansing, MI 48824-1226, USA.

[11] Zhou, Y. (n.d.). Music Generation Based on Bidirectional GRU Model. Department of Statistics, University of Michigan.

[12] Jozefowicz, R., Zaremba, W., & Sutskever, I. (2015). An Empirical Exploration of Recurrent Network Architectures. Proceedings of the 32nd International Conference on Machine Learning (ICML-15), 2342-2350. https://proceedings.mlr.press/v37/jozefowicz15.html

[13] Kang, Q., Chen, E. J., Li, Z.-C., Luo, H.-B., & Liu, Y. (Year). Attention-based LSTM predictive model for the attitude and position of shield machine in tunneling. State Key Laboratory of Water Resources Engineering and Management, Wuhan University, Wuhan, China. School of Civil & Hydraulic Engineering, Huazhong University of Science & Technology, Wuhan, China. Tunnel Engineering Company, Wuhan Municipal Construction Group Co. Ltd, Wuhan, China.

[14] Staudemeyer, R. C., & Morris, E. R. (Year). Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks. Faculty of Computer Science, Schmalkalden University of Applied Sciences, Germany.