MERT: Acoustic Music Understanding Model with Large-Scale Self-supervised Training

Nikol Toda, Matricola: 2047236

February 13, 2024

Abstract

Self-supervised learning (SSL) has recently emerged as a promising paradigm for training generalisable models on large-scale data in the fields of vision, text, and speech. Although SSL has been proven effective in speech and audio, its application to music audio has yet to be thoroughly explored. This is partially due to the distinctive challenges associated with modelling musical knowledge, particularly tonal and pitched characteristics of music. To address this research gap, we propose an acoustic Music undERstanding model with large-scale self-supervised Training (MERT), which incorporates teacher models to provide pseudo labels in the masked language modelling (MLM) style acoustic pre-training. In our exploration, we identified an effective combination of teacher models, which outperforms conventional speech and audio approaches in terms of performance. This combination includes an acoustic teacher based on Residual Vector Quantization - Variational AutoEncoder (RVQ-VAE) and a musical teacher based on the Constant-Q Transform (CQT). Furthermore, we explore a wide range of settings to overcome the instability in acoustic language model pre-training, which allows our designed paradigm to scale from 95M to 330M parameters. Experimental results indicate that our model can generalise and perform well on 14 music understanding tasks and attain state-of-the-art (SOTA) overall scores.

Contents

1 Introduction

In the rapidly evolving field of music information retrieval (MIR), the development of advanced machine learning models for acoustic music understanding represents a significant leap forward. Among these advancements, the MERT (Music Encoding and Recognition Transformer) model emerges as a pioneering approach, leveraging the power of large-scale self-supervised training to achieve unprecedented accuracy and efficiency in understanding and analyzing acoustic music data.

At its core, MERT is designed to address the intrinsic complexity of music as an audio signal, capturing the nuances and dynamics that define musical genres, styles, and expressions. This innovative model employs a transformer-based architecture, renowned for its success in natural language processing (NLP), to process and analyze acoustic music signals. By adapting these techniques to the domain of music, MERT opens new avenues for extracting meaningful patterns, features, and insights from raw audio data.

The foundation of MERT's success lies in its self-supervised training regimen, which utilizes a vast corpus of unlabeled music data. This approach allows the model to learn from the inherent structure and characteristics of music itself, without the need for extensive annotated datasets that are often scarce and labor-intensive to produce. Through this process, MERT gains a deep understanding of musical elements such as melody, harmony, rhythm, and texture, enabling it to perform a wide range of MIR tasks with remarkable proficiency.

The implications of MERT's capabilities extend far beyond academic research, offering practical applications in various sectors of the music industry. From enhancing music recommendation systems and automating genre classification to supporting music creation and analysis tools, MERT stands as a testament to the potential of machine learning to revolutionize our interaction with music.

This report delves into the development, architecture, and applications of the MERT model, highlighting its contributions to the field of acoustic music understanding. Through large-scale self-supervised training, MERT not only advances the state-of-the-art in MIR but also paves the way for innovative solutions to some of the most challenging problems in music analysis and understanding.

2 The Free Music Archive (FMA) Dataset

2.1 Overview

The Free Music Archive (FMA) dataset is a rich collection of audio files and metadata from the Free Music Archive, a well-known repository of free-access, royalty-free music. The dataset is curated to facilitate music analysis, machine learning research, and algorithm development, providing a wide range of musical genres, from classical and jazz to electronic and hip-hop. It is a valuable resource for researchers and developers seeking to explore music information retrieval (MIR), audio signal processing, and related fields.

2.2 Composition

The FMA dataset is structured into several key components, each serving a specific purpose in music analysis and machine learning tasks:

- **Tracks:** Contains metadata for each track, including unique identifiers, titles, artists, and associated genre information. This component forms the backbone of the dataset, linking each audio file to its descriptive data.
- **Genres:** Offers a taxonomy of musical genres represented in the dataset. Each genre is assigned a unique identifier, facilitating the classification and analysis of tracks based on musical style.
- **Features:** Comprises extracted audio features for each track, such as Mel-frequency cepstral coefficients (MFCCs), chroma features, spectral contrast, and tonnetz. These features are crucial for tasks involving content-based music analysis, genre classification, and recommendation systems.
- **Echonest Features:** Includes additional audio features and analysis provided by The Echo Nest (now part of Spotify). This subset enriches the dataset with advanced attributes like acousticness, danceability, energy, and speechiness, offering deeper insights into the music's characteristics.

2.3 Data Collection and Processing

The FMA dataset is meticulously compiled from the Free Music Archive's extensive library, ensuring a diverse representation of musical genres and styles. The dataset's creators have processed the audio files to extract relevant features and metadata, organizing them into structured CSV files for easy access and analysis.

2.4 Challenges and Considerations

While the FMA dataset offers vast opportunities for research and application development, users must navigate challenges such as data size, processing requirements, and the complexity of audio analysis.

3 Data Processing

This section outlines the data processing steps implemented in the project, focusing on preparing the Free Music Archive (FMA) dataset for analysis and machine learning tasks. The process involves importing necessary libraries, loading data from various sources, and applying preprocessing transformations to enhance data quality and usability.

3.1 Libraries and Setup

- **Libraries Used:** The project utilizes a range of Python libraries for data handling and analysis, including 'pandas' for data manipulation, 'numpy' for numerical operations, 'matplotlib' and 'seaborn' for data visualization, 'librosa' for audio processing, and 'sklearn' for machine learning tasks.
- **Environment Setup:** Matplotlib is configured for inline display within Jupyter notebooks, with figure size parameters set for optimal visualization. Paths to audio files and metadata are defined, leveraging environment variables for flexible dataset location management.

3.2 Data Loading

- **Data Sources:** The dataset comprises tracks, genres, features, and Echonest features, stored across multiple CSV files within a Google Drive directory. Each file serves a specific purpose:
  - *'tracks.csv'* - Metadata for each track, including IDs and genre information.
  - *'genres.csv'* - Genre definitions and IDs.
  - *'features.csv'* - Extracted audio features for each track.
  - *'echonest.csv'* - Additional features provided by Echonest.
- **Loading Process:** Data from these files is loaded into pandas DataFrames, with specific configurations for index columns and headers to maintain data structure and integrity.

3.3 Data Exploration

Initial exploration involves displaying basic dataset information, such as shape and head entries, to understand data dimensions and preview content. This step is crucial for identifying potential data quality issues and planning subsequent preprocessing steps.

3.4 Preprocessing

- A 'Preprocessor' class is defined to apply a series of transformations to the features, including normalization to standardize feature scales and any additional custom transformations deemed necessary.

- The preprocessing steps are encapsulated within a 'FMADataset' class, extending PyTorch's 'Dataset' for integration with deep learning models. This class includes methods for data preprocessing, item access, and genre indexing, facilitating efficient data handling and transformation during model training.
- A custom collate function, wrapped within the 'CollateFunctionWrapper' class, is implemented to handle batch preparation, including feature padding and genre encoding for batched data processing in model training.

This section provided a comprehensive overview of the data processing steps, from initial loading and exploration to complex preprocessing and dataset preparation for machine learning tasks. These processes ensure the data is clean, structured, and ready for further analysis and model development.

4 Methodology

4.1 Custom Transformer Encoder: TransformerEncoderExtend

4.1.1 Architecture Design:

The 'TransformerEncoderExtend' class represents a significant modification of the standard Transformer encoder. This custom encoder is designed to increase the model's adaptability and performance in processing complex language data.
Key architectural elements include an adjustable number of encoder layers, each capable of being individually tuned. This flexibility allows for more nuanced modeling of language structures, adapting to varying complexities found in different NLP tasks.

4.1.2 Feature Enhancement:

Each encoder layer in 'TransformerEncoderExtend' is built to enhance certain features of the Transformer model. Notably, the embedding dimensions and dropout rates are configurable, enabling fine-tuning for specific datasets and applications.
This customization extends the model's capability beyond standard applications, allowing it to handle more challenging NLP tasks that require a deeper understanding of context and nuance.

4.1.3 Layer Customization:

The layers within the encoder are not just replicas of the standard Transformer layer but are re-engineered to incorporate advanced NLP concepts. This includes modifications to the attention mechanism and feed-forward networks, catering to the specific demands of your NLP application.

4.2 Enhanced Attention Mechanism: MultiheadAttentionExtend

4.2.1 Attention Mechanism Refinement:

The 'MultiheadAttentionExtend' class is a refined version of the traditional multi-head attention mechanism found in Transformers. It's tailored to provide a more granular attention process, which is crucial for understanding complex linguistic patterns and dependencies.
This enhanced attention mechanism is likely to improve the model's ability to distinguish and prioritize different segments of the input data, a crucial aspect in tasks like contextual understanding and semantic representation.

4.2.2 Improvement in Contextual Understanding:

The modifications in the attention mechanism aim to enhance the model's contextual understanding. By adjusting how attention is distributed across different parts of the input, the model can potentially gain a more nuanced understanding of language, improving its performance in tasks such as sentiment analysis, context-sensitive translation, or abstract summarization.

4.2.3 Custom Implementation Details:

The implementation includes specific adjustments to the attention calculation, which may involve scaling the attention scores or altering the way these scores are combined with the input embeddings. Such changes are indicative of an experimental approach to improve the Transformer's inherent capabilities.

The methodology employed in this project reflects a deep engagement with cutting-edge NLP research and practices. By customizing key components of the Transformer architecture, your project pushes the boundaries of what these models can achieve, particularly in understanding and processing complex, nuanced language data.

This detailed focus on methodology indicates a project that is not just applying existing models but actively seeking to innovate and contribute to the advancement of NLP technology. The project's success could have significant implications for the field, potentially leading to more effective and sophisticated language processing tools.

5 Detailed Exploration of Technological Stack and Implementation Details

5.1 Core Technologies and Libraries

5.1.1 Python and PyTorch Framework:

The project is developed in Python, a versatile and widely-used programming language in the field of data science and machine learning.

PyTorch is the primary framework used for constructing the neural network components. Its dynamic computation graph and comprehensive ecosystem make it a top choice for developing sophisticated machine learning models, particularly in the field of NLP.

5.1.2 Transformers Library:

he use of the transformers library, known for its extensive collection of pre-trained models and Transformer components, plays a crucial role. It provides a solid foundation for both leveraging and extending state-of-the-art Transformer architectures.

This library simplifies the implementation of complex models and allows for an efficient comparison with benchmark models in NLP tasks.

5.1.3 Additional Libraries - torchtext, datasets, torchaudio:

torchtext is employed for efficient text processing and data loading, which is crucial for handling linguistic datasets.

The datasets library aids in managing large-scale language datasets, ensuring smooth data handling and preprocessing.

torchaudio offers supplementary functionalities, although its specific role would depend on whether the project encompasses audio data processing in the context of NLP.

5.2 Advancing NLP: Customization and Innovation

5.2.1 Custom Transformer Encoder - TransformerEncoderExtend:

The implementation of TransformerEncoderExtend suggests a deep customization of the Transformer encoder.

Key adaptations include variable encoder layers, adjustable embedding dimensions, and tunable dropout rates. These allow the model to adapt to different linguistic datasets and tasks, potentially improving performance and accuracy.

The encoder's flexibility in handling various types of input data points to its potential application in diverse NLP scenarios, from text classification to language generation tasks.

5.2.2 Enhanced Attention Mechanism - MultiheadAttentionExtend:

he modified multi-head attention mechanism, MultiheadAttentionExtend, represents a significant enhancement over the standard attention process in Transformers.
This customized attention module could offer more refined control over the model's focus, allowing it to better capture contextual nuances and complex dependencies in language data.
Such an enhancement is particularly crucial in tasks like semantic analysis, where understanding the subtle context is key.

5.2.3 Advanced Techniques and Novel Concepts:

The code snippets indicate the use of advanced neural network techniques, including layer normalization, customized forward passes, and innovative adaptations of standard Transformer components.
The project appears to incorporate cutting-edge concepts in NLP, potentially contributing to the field through novel methodologies or improved performance metrics.

6 Detailed code examples:

```
class TransformerEncoderExtend(nn.Module):
    def __init__(self, args):
        super().__init__()
        self.args = args
        self.embedding_dim = args.embedding_dim
        self.dropout = args.dropout
        self.layer_type = args.layer_type
        self.encoder_layers = args.encoder_layers
        self.attention_relax = args.attention_relax

        self.layers = nn.ModuleList([
            self.build_encoder_layer(args) for _ in range(args.encoder_layers)
        ])
```

Figure 1: Transformer

```
class MultiheadAttentionExtend(nn.MultiheadAttention):
    def __init__(
        self,
        embed_dim,
        num_heads,
        kdim=None,
        vdim=None,
        dropout=0.0,
        bias=True,
        add_bias_kv=False,
        add_zero_attn=False,
        self_attention=False,
        encoder_decoder_attention=False,
        q_noise=0.0,
        qn_block_size=8,
        attention_relax=-1.0,
    ):
```

Figure 2: Multihead

```
class MERTConfigClass:
    def __init__(self):
        # Define your MERT configuration parameters here
        self.embedding_dim = 768
        self.encoder_ffn_embed_dim = 3072
        self.encoder_attention_heads = 8
        self.dropout = 0.1
        self.attention_dropout = 0.1
        self.activation_dropout = 0.1
        self.activation_fn = "relu"
        self.layer_norm_first = False
        self.attention_relax = -1.0  # Set to a positive value to enable attention relaxation
        self.encoder_layers = 6  # Number of encoder layers
        self.deepnorm = False
        self.layer_type = "transformer"  # or "conformer"
        self.subln = True
        self.checkpoint_activations = False
```

Figure 3: MERT

```
# Usage example:
args = MERTConfigClass()  # Replace with your MERT configuration
model = TransformerEncoderExtend(args)
input_tensor = torch.randn((10, 32, 768))  # Replace with your input shape
output, attn_weights = model(input_tensor)
```

Figure 4: Result

7 Training Phase of the MERT Model

7.1 Overview

The training phase is a critical component of the MERT model's development, enabling the model to learn from a large-scale dataset with a self-supervised approach. This section outlines the training process, highlighting the model's architecture adjustments, loss calculation, optimization steps, and the monitoring of training progress through loss and accuracy metrics.

7.2 Training Configuration

The model is configured to train over 10 epochs, indicating the complete dataset will be passed through the model ten times. To facilitate monitoring and adjustments during training, a 'print interval' of 100 batches is set, allowing for periodic reporting of training metrics.

7.3 Initialization of Metrics

Two key metrics are initialized to track the model's performance throughout the training process:

- **Train Losses:** An array to record the average loss per epoch, providing insights into the model's learning progress and convergence.
- **Train Accuracy:** An array to capture the model's accuracy in classifying music genres, reflecting the effectiveness of feature learning and representation.

7.4 Training Loop

The training loop is meticulously designed to iteratively update the model's weights based on the feedback from the loss function. Key steps in each epoch include:

- **Model State:** The model is set to training mode, enabling specific layers like dropout and batch normalization to function in their training-specific modes.
- **Batch Processing:** The dataset is processed in batches, utilizing a DataLoader to efficiently manage memory and computation. Each batch contains a subset of features and corresponding genre labels.
- **Feature Expansion:** An expansion step is applied to the input features, adjusting their dimensions to match the model's input requirements. This step is crucial for accommodating the model's architecture.
- **Forward Pass:** The model performs a forward pass with the expanded features, generating predictions for each instance in the batch.
- **Loss Calculation:** A custom cross-entropy loss function is employed to quantify the discrepancy between the model's predictions and the actual genre labels. This loss guides the model's learning.
- **Backward Pass and Optimization:** The loss is backpropagated through the model, and an optimizer updates the model's weights to minimize the loss. This iterative optimization is central to the model's ability to learn from the data.

7.5 Monitoring and Reporting

To gauge the model's performance and ensure effective learning, the training loop includes mechanisms for monitoring and reporting key metrics:

- **Batch Loss:** The loss for each batch is recorded, offering immediate feedback on the model's performance and the effectiveness of each optimization step.
- **Accumulated Average Loss:** The average loss across all processed batches provides a smoothed view of the model's learning trajectory, helping to identify trends and convergence.
- **Batch Accuracy:** The accuracy of the model on each batch, calculated as the percentage of correctly predicted genre labels, serves as a direct indicator of the model's classification capabilities.

7.6 Epoch Summary

At the end of each epoch, the average loss and accuracy across all batches are computed and appended to their respective tracking arrays. This epoch-level summary offers a comprehensive view of the model's performance and learning progress, facilitating adjustments to training parameters and strategies. The training phase of the MERT model is a meticulously designed process that balances the computational demands of large-scale self-supervised learning with the need for detailed performance monitoring. Through iterative optimization and careful adjustment of model parameters, the training phase ensures that MERT can effectively understand and classify acoustic music data, showcasing the potential of advanced machine learning techniques in the realm of music information retrieval.

8 Summary

The MERT model represents a significant advancement in the field of music information retrieval (MIR), employing a transformer-based architecture to analyze and understand acoustic music through large-scale self-supervised learning.
The FMA dataset, a cornerstone of this project, comprises a vast collection of audio files and metadata designed to facilitate music analysis and machine learning research. It includes detailed metadata for each track, a comprehensive genre taxonomy, extracted audio features, and additional Echonest features, supporting a wide range of MIR tasks from genre classification to music recommendation systems.
The model utilizes the Free Music Archive (FMA) dataset, engaging in comprehensive data processing techniques. This involves loading tracks, genres, features, and Echonest features from structured CSV files. Key Python libraries such as 'pandas', 'numpy', 'librosa', and 'torch' are used for data manipulation, audio processing, and model training. A 'Preprocessor' class and a 'FMADataset' class are defined to apply normalization and other transformations, preparing the data for efficient machine learning tasks.

- **Custom Transformer Encoder ('TransformerEncoderExtend'):** Features an adjustable number of encoder layers with configurable embedding dimensions and dropout rates, enabling nuanced modeling of language structures and improved handling of complex NLP tasks.
- **Enhanced Attention Mechanism ('MultiheadAttentionExtend'):** Offers refined attention processes for better understanding of complex linguistic patterns, enhancing contextual understanding and semantic representation in tasks such as sentiment analysis and context-sensitive translation.

The training of the MERT model is conducted over 10 epochs, with performance metrics such as loss and accuracy monitored at regular intervals. The training process involves feature expansion, forward passes through the model, loss calculation using custom cross-entropy, and weight optimization based on backpropagation. This phase is critical for refining the model's ability to accurately classify music genres and understand complex audio patterns.
MERT's development, bolstered by a sophisticated methodology involving custom transformer encoders and enhanced attention mechanisms, sets a new standard in music information retrieval. Through the FMA dataset, it demonstrates the potential of machine learning in revolutionizing music understanding and analysis.