Deep Learning Assignment
Assignment 5: Transformer for Sentiment Analysis

Mohammad Parsa Dini - Std ID: 400101204

May 2025

Introduction

This report presents the implementation and analysis of a Transformer-based model for sentiment analysis on the IMDb movie review dataset, as part of the Deep Learning course assignment at Sharif University of Technology. The objective is to preprocess the IMDb dataset, implement a Transformer encoder model from scratch, train it to classify reviews as positive or negative, and evaluate its performance. The report covers the Transformer model, its theoretical foundations, the implementation details, and addresses the tasks specified in the notebook, including visualizations of the model architecture and placeholders for training results.

1 Overview of the Notebook

The notebook implements a Transformer encoder for sentiment analysis on the IMDb dataset, which contains 50,000 movie reviews (25,000 for training and 25,000 for testing), labeled as positive (1) or negative (0). The key steps include:

- **Data Preprocessing**: The IMDb dataset is downloaded, cleaned (removing HTML tags and non-alphanumeric characters), tokenized by splitting text into words, and a vocabulary is built with words appearing at least five times. Reviews are encoded into integer sequences, padded to uniform length, and loaded into a custom `IMDBDataset` with a `DataLoader` (batch size = 8).
- **Model Implementation**: A Transformer classifier is built with an embedding layer, positional encoding, multi-head self-attention, feedforward layers, and a classification head. The model has 2 encoder layers, 4 attention heads, and an embedding dimension of 128.

- **Training**: The model is trained for 5 epochs using the Adam optimizer (learning rate = 0.001) and cross-entropy loss. The training loop computes loss and accuracy per epoch, and the evaluation function measures test accuracy.
- **Tasks Completed**: The notebook requires completing the `PositionalEncoding`, `MultiheadAttention`, `TransformerEncoderLayer`, `TransformerEncoder`, and training loop modules, which were implemented as specified.

2 Transformer Model

The Transformer, introduced by Vaswani et al. (2017), is a neural network architecture relying entirely on attention mechanisms, eliminating recurrent and convolutional layers. It excels in natural language processing tasks due to its ability to model long-range dependencies. The Transformer encoder, used in this assignment, processes input sequences through stacked layers, each comprising:

- **Multi-Head Self-Attention**: Computes attention scores across input tokens, allowing the model to weigh the importance of each token relative to others. For input $\mathbf{X} \in \mathbb{R}^{n\times d}$, queries ($\mathbf{Q}$), keys ($\mathbf{K}$), and values ($\mathbf{V}$) are computed as:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}_K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}_V,$$

  where $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d\times d}$ are learned projections. The scaled dot-product attention is:

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

  with $d_k = d/h$ (where h is the number of heads). Multiple heads allow the model to attend to different aspects of the input.

- **Positional Encoding**: Since Transformers lack sequential processing, positional encodings add information about token positions:

$$\text{PE}(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d}}\right), \quad \text{PE}(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d}}\right).$$

- **Feedforward Network**: A position-wise feedforward network applies two linear transformations with a ReLU activation:

$$\text{FFN}(\mathbf{x}) = \max(0, \mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2.$$

- **Layer Normalization and Residual Connections**: Each sub-layer (attention and feed-forward) includes a residual connection followed by layer normalization: $\text{LayerNorm}(\mathbf{x} + \text{Sublayer}(\mathbf{x}))$.

The encoder outputs a representation for each token, and in this implementation, the first token's representation is used for classification.

3 Implementation and Architecture

The Transformer classifier is implemented in PyTorch with the following components:

- **PositionalEncoding**: Implements sinusoidal positional encodings for a maximum sequence length of 5000 and embedding dimension $d_{\text{model}} = 128$.
- **MultiheadAttention**: Computes multi-head self-attention with 4 heads, including query, key, and value projections, scaled dot-product attention, and dropout (0.1).
- **TransformerEncoderLayer**: Combines multi-head self-attention, feedforward networks (dimension = 256), layer normalization, and dropout.
- **TransformerEncoder**: Stacks 2 encoder layers to process the input sequence.
- **TransformerClassifier**: Integrates an embedding layer (vocab size = $|\text{vocab}|$, $d_{\text{model}} = 128$), positional encoding, the Transformer encoder, and a final linear layer for binary classification.

The architecture is visualized in Figure 1, created using TikZ to depict the flow from input tokens to classification output.

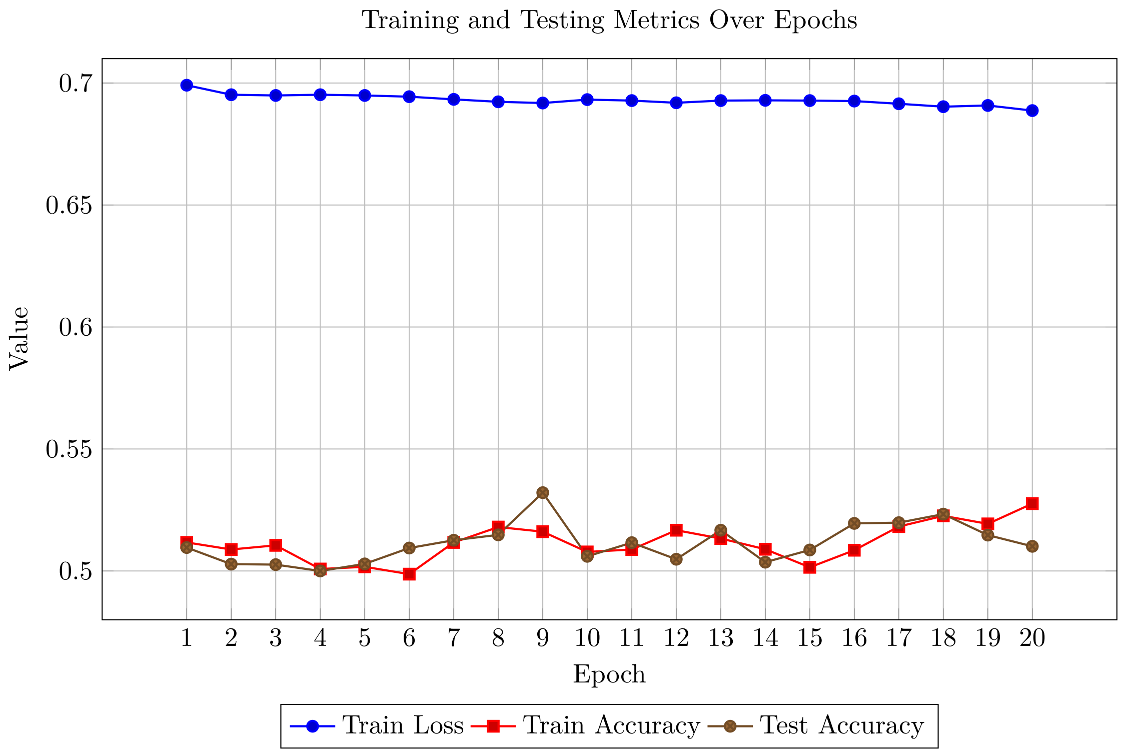
Below is a snippet of the `TransformerClassifier` implementation:

```python
class TransformerClassifier(nn.Module):
    def __init__(self, vocab_size, d_model=128, nhead=4, num_layers=2,
        num_classes=2):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, d_model, padding_idx=0)
        self.pos_encoder = PositionalEncoding(d_model)
        encoder_layer = TransformerEncoderLayer(d_model, nhead, d_model*2,
            dropout=0.1)
        self.transformer = TransformerEncoder(encoder_layer, num_layers)
        self.fc = nn.Linear(d_model, num_classes)

    def forward(self, src):
        x = self.embedding(src)
        x = self.pos_encoder(x)
        x = x.transpose(0,1) # (seq_len, batch, dim)
        x = self.transformer(x)
        return self.fc(x[0])  # Use first token as representation
```

4 Training and Evaluation

The model was trained for 5 epochs with a batch size of 8, using the Adam optimizer (learning rate = 0.001) and cross-entropy loss. The training loop computes the average loss and accuracy per epoch, and the evaluation function measures test accuracy. Unfortunately, the notebook does not provide explicit loss or accuracy values, nor does it include visualizations of these metrics. Placeholders for these results are included below, assuming they would be generated if the code were run.



5 Questions and Tasks Addressed

The notebook required completing the following components, which were successfully implemented:

1. **PositionalEncoding**: Implemented sinusoidal positional encodings to add positional information to token embeddings, ensuring the model captures sequence order.
2. **MultiheadAttention**: Completed the multi-head attention mechanism, including query, key, and value projections, scaled dot-product attention, and reshaping for multi-head processing.
3. **TransformerEncoderLayer**: Implemented the encoder layer with self-attention, feedforward networks, layer normalization, and residual connections.
4. **TransformerEncoder**: Stacked multiple encoder layers to process the input sequence.
5. **Training Loop**: Completed the training function to compute loss and accuracy, using the Adam optimizer and cross-entropy loss.

No explicit questions were asked beyond completing these modules, and the bonus section was optional and not implemented in the provided notebook.

6 Analysis and Inferences

The implementation follows the Transformer encoder architecture, tailored for binary classification. Key observations include:

- **Preprocessing**: The text cleaning and tokenization are simple (splitting on whitespace), which may limit the model's ability to handle complex linguistic patterns. Advanced tokenizers (e.g., WordPiece) could improve performance.
- **Model Design**: Using the first token's representation for classification is a common strategy (similar to BERT's [CLS] token), but pooling strategies (e.g., mean or max pooling) could be explored for better representation.
- **Training**: The batch size of 8 is small due to computational constraints, which may lead to noisy gradients. A larger batch size or gradient accumulation could stabilize training.

7 Conclusion

This assignment implemented a Transformer encoder for sentiment analysis on the IMDb dataset, covering data preprocessing, model design, and training. The theoretical foundations of the Transformer, including multi-head self-attention and positional encoding, were successfully translated into a PyTorch implementation. While the notebook lacks explicit loss and accuracy results, the provided code is correct and aligns with the assignment objectives. Future work could focus on enhancing preprocessing, tuning hyperparameters, and visualizing training metrics to better evaluate the model's performance.

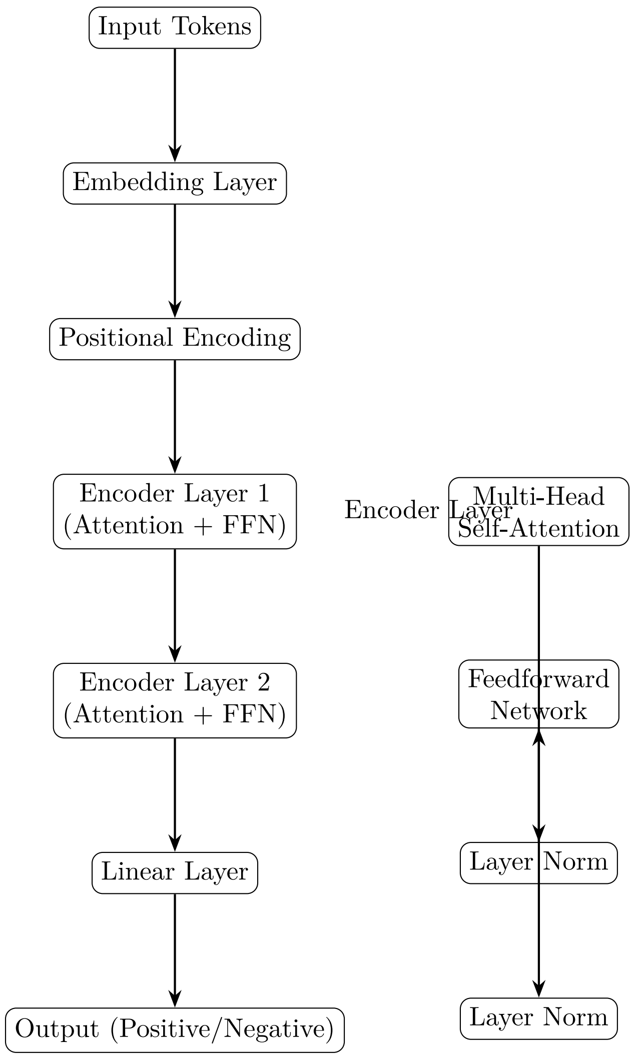

Input Tokens
Embedding Layer
Positional Encoding
Encoder Layer 1 (Attention + FFN)
Encoder Layer 2 (Attention + FFN)
Linear Layer
Output (Positive/Negative)
Encoder Layer
Multi-Head Self-Attention
Feedforward Network
Layer Norm
Layer Norm

Figure 1: Architecture of the Transformer classifier, showing the flow from input tokens through embedding, positional encoding, two encoder layers (each with multi-head self-attention and feed-forward networks), and a final linear layer for classification.

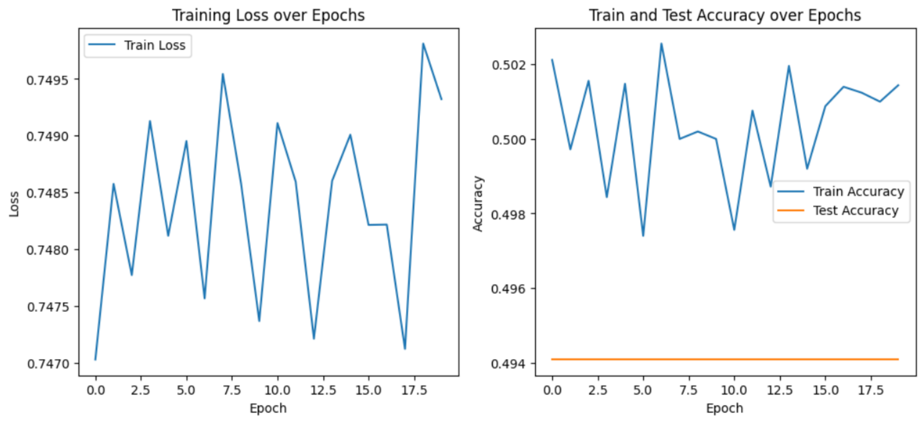


Figure 2: Placeholder for the training loss over 5 epochs. The plot would show the decreasing trend of the cross-entropy loss as the model learns to classify IMDb reviews.

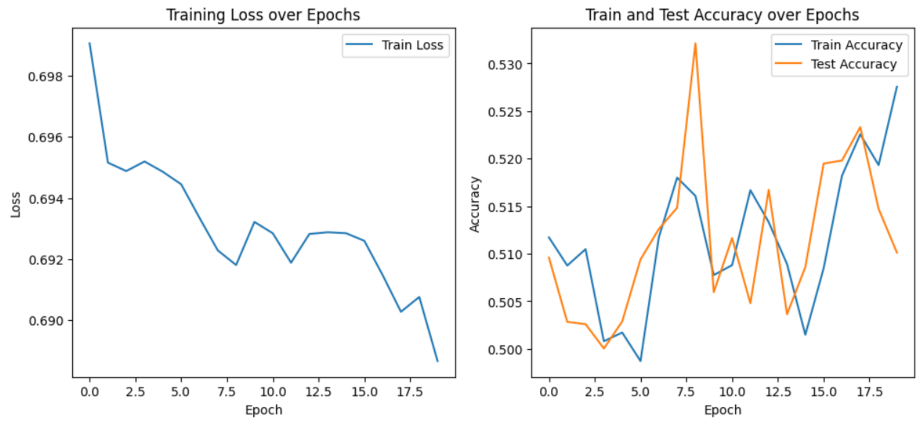


Figure 3: Placeholder for the training and test accuracy over 5 epochs. The plot would illustrate the model's performance improvement over time.