Question-1
Implement Transformer from Scratch

Overview

In this assignment we have to implement a Transformer model from scratch for language modeling on Shakespearean text and then preprocess the dataset (tokenization, handling special tokens, padding/truncation). After that we have to build core Transformer components:

- Self-attention & multi-head attention.
- Positional encoding, layer normalization, residual connections.
- Scaled dot-product attention and feed-forward networks.

After that we have to train the model with proper loss computation (cross-entropy) and have to track training/validation loss for each epoch and should have periodic text generation to monitor progress.

Here we have to evaluate using perplexity, ensuring padding tokens are excluded.

For testing we have to create an inference function which will load the model, process test.txt, generate text, and will compute perplexity.

Explanation of Preprocessing Steps

1. **Tokenization**:
   - We are splitting sentences into tokens using whitespace.
   - Then we are adding special tokens <START>, <STOP>, and <PAD> to mark sentence boundaries and handle padding.
   - A vocabulary is built from training data which mapping each unique token to an index and an inv which do vice versa

2. **Padding/Truncation**:

   - We are padding sequences to a fixed length (MAX_LEN=128) using <PAD> tokens to ensure we have uniform input dimensions.

    - If we have excess tokens (greater than MAX_LEN) then we are truncating them.

3. **Dataset Preparation**:
    - Input-target pairs are created by shifting token sequences (e.g., tokens[:-1] as input, tokens[1:] as target).
    - We are converting data to PyTorch tensors for model compatibility and easy computations.

Model Architecture

The Transformer model consists of:

1. Embeddings:
    - We have Token embeddings (nn.Embedding) to convert token IDs to vectors.
    - and positional embeddings (nn.Embedding) to encode positional information.

2. Transformer Layers:
    - Multi-Head Attention: Here we are doing scaled dot-product attention with causal masking for autoregressive generation.
    - Feed-Forward Networks: then two linear layers with ReLU activation and dropout.
    - Residual Connections: we are also applying residual conn after attention and feed-forward layers.
    - Layer Normalization: Used before and after each sub-layer for better results.

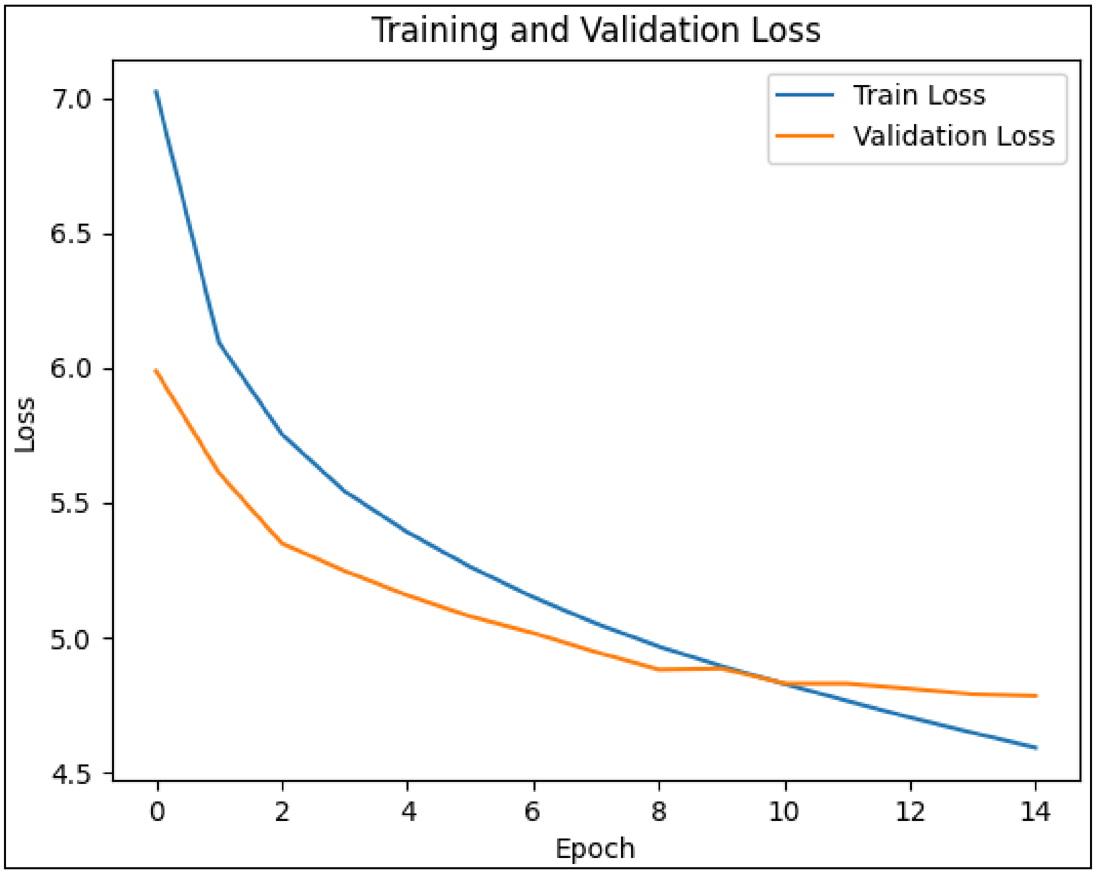
3. Output Head:
    - Final linear layer to project embeddings back to vocabulary size.

Key Hyperparameters:

- Embedding dimension: 256
- Number of heads: 4

- Transformer layers: 6
- Feed-forward hidden dim: 1024
- Dropout: 0.1
- Batch size: 32
- Learning rate: 1e-4
- Epochs: 15

Training and Validation Loss Plots



Observations:

- Training and validation losses are decreasing steadily which indicates effective learning and no overfitting.
- No severe overfitting is there which suggests proper regularization (dropout, layer norm).
- Final validation perplexity for dev file is **80.21** which shows reasonable model performance.

Text Generation Examples

The model generates coherent Shakespeare-style text:

Context: DUKE VINCENTIO : And you , good brother father .
Generated: <START> , if there be no remedy for it , but that you will needs buy and sell men and women like beasts , we shall have all the world drink brown and white bastard .

The outputs shows old-fashioned vocabulary with dramatic tone which shows the model’s ability to learn stylistic patterns.

Conclusion

The implemented shows that this decoder only transformer model successfully learns Shakespearean language pattern and achieves low perplexity and is generates contextually relevant text.