Assignment 2 — Building mini-GPT Model Report

Student: Yuqi Lai

Course: CSYE7374

1. Model architecture and parameters

The model uses PyTorch's `nn.TransformerEncoder` backbone with **causal masking**, ensuring that each token can only attend to previous positions in the sequence. This enforces the autoregressive property required for next-token prediction.

Key Parameters

- Embedding Dimension: 256
- Transformer Layers: 2
- Attention Heads: 4
- Context Window (Block Size): 128 tokens
- Vocabulary Size: ~32,000 (from the pretrained `AutoTokenizer`)

Components

- **Embeddings:** Each token is represented through learned token embeddings (`nn.Embedding`), which are summed with learnable positional embeddings of the same dimensionality.
- **Causal Masking:** A triangular mask created via `torch.triu(tensor, diagonal=1)` assigns `-inf` to future positions, preventing information leakage during self-attention.
- **Normalization:** A final `nn.LayerNorm` ensures stable representations before projection.
- **Output Head:** A linear layer transforms hidden states into vocabulary-sized logits for next-token prediction.

2. Dataset Details

The model was trained on the **preprocessed dataset (`dataset_128`)** prepared in Assignment 1. All sequences were pre-tokenized to a **fixed length of 128 tokens**, enabling efficient batching and training.

Data Subsetting

Because the original dataset contained over **1.9 million samples**, a 10% subset was selected to ensure feasible training time and resource usage. The final working dataset contained **193,421 samples**.

Train/Validation Split

The subset was further divided using a 90/10 split:

- **Training Set:** 174,078 samples
- **Validation Set:** 19,343 samples

Preprocessing

- Token sequences were generated using the previously saved tokenizer.
- A custom `collate_fn` was implemented to efficiently stack `input_ids` into contiguous tensors during batching.

3. Training setup and hyperparameter experiments

The training loop was implemented using PyTorch, with optimization performed using AdamW and loss computed via Cross-Entropy Loss. Model evaluation was conducted using Perplexity (PPL), computed as `exp(avg_loss)` for each epoch.

Experimental Configuration

Two learning rate configurations were tested to evaluate the sensitivity of the model to optimization hyperparameters. All other hyperparameters remained constant.

Hyperparameters

- Batch Size: 128
- Epochs: 2
- Optimizer: AdamW
- Loss Function: Cross-Entropy
- Context Length: 128 tokens

Experiment A – Baseline

- Learning Rate: 5e-4

Experiment B – High Learning Rate

- Learning Rate: 1e-3

4. Observations and Analysis

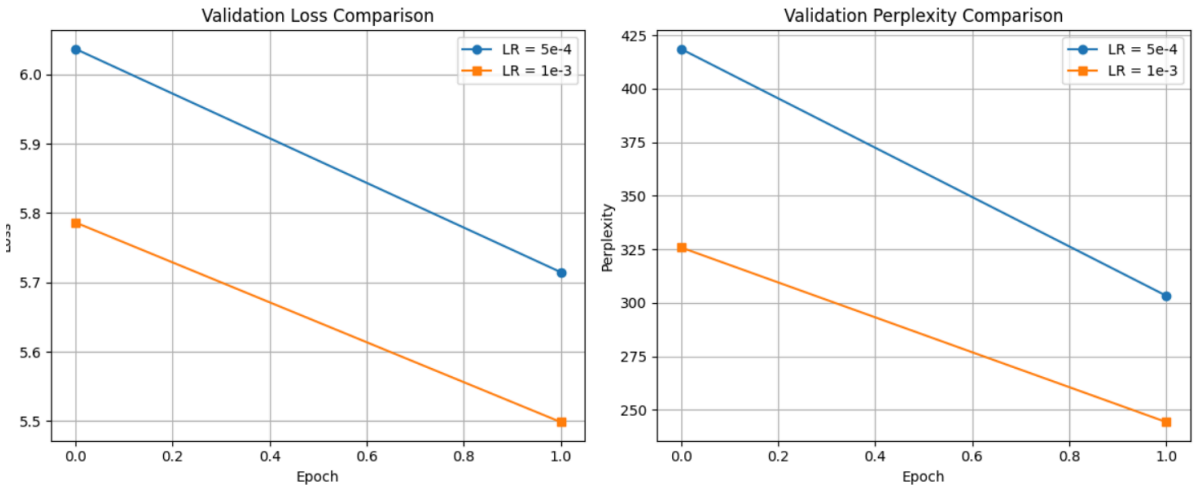
1. Impact of Learning Rate

Experiment B (LR = 1e-3) clearly outperformed the baseline in both validation loss and perplexity. The **orange curve** (High LR) shows a noticeably steeper downward trajectory compared to the **blue curve** (Baseline).

- **Observation:** Validation perplexity reduced from ~303 → ~244 (≈19% improvement).
- **Interpretation:** For a small-scale 2-layer Transformer trained for only two epochs, **5e-4 was too conservative**. The model benefited from larger updates and showed no signs of instability.

2. Generalization

The training–validation loss gap remained small across epochs, suggesting minimal overfitting within this early training regime.



5. Challenges and Solutions

5.1. Checkpoint Loading Consistency

- **Challenge:** Loading the `.pt` file initially failed when the inference-time architecture did not exactly match the training-time configuration.

- **Solution:** Re-instantiating `MiniGPT` using the same `DEFAULT_CONFIG` before calling `load_state_dict()` ensured architecture-weight compatibility.

5.2. Causal Masking Implementation

- **Challenge:** Ensuring that no future tokens were visible to the model required precise masking logic.

- **Solution:** Using `torch.triu(..., diagonal=1)` filled with `-inf` reliably masked out future positions, preventing "cheating" and ensuring valid causal modeling.

5.3. Resource Constraints

- **Challenge:** Training on the full 1.9M-row dataset was computationally too expensive.

- **Solution:** Using a **10% subset** allowed faster iteration and hyperparameter exploration while preserving dataset diversity.