DS 5983: Large Language Models (Summer 2025) — Roi Yehoshua

Student name: — (Due) June 26, 2025

PA3: Build GPT-2 From Scratch

In this assignment, you will develop a GPT-2 like model from scratch using PyTorch. You will implement a Byte-Pair Encoding (BPE) tokenizer, construct a decoder-only Transformer-based language model, train it on a next-word prediction task, and evaluate its performance.

1. Before starting, review the following resources to familiarize yourself with GPT-2's architecture and training principles:
   - GPT-2 Blog Post (OpenAI): Overview of GPT-2's architecture, capabilities, and results.
   - GPT-2 Paper: Technical details on GPT-2.
   - GPT-2 GitHub Repository: GPT-2 codebase, training scripts, and additional documentation.
   - Hugging Face GPT-2: Pre-trained models, usage examples, and training tips.
   - Andrej Karpathy's YouTube Tutorial: A hands-on walkthrough of implementing a GPT model from scratch.
2. Choose a dataset that is manageable for training on a single GPU. Recommended options:
   - OpenWebText: A dataset similar to GPT-2's training corpus.
   - Wikitext-2: A lightweight text dataset via Hugging Face.
3. Implement a tokenizer based on Byte-Pair Encoding (BPE):
   (a) Define an appropriate vocabulary size (e.g., 32,000 tokens).
   (b) Train the tokenizer on your dataset.
   (c) Save the trained tokenizer for later use.
4. Implement a decoder-only Transformer architecture, including the following components:
   (a) Token and position embeddings. Use learned position embedding, as done in GPT-2, instead of sinusoidal positional encodings.
   (b) Multi-head masked self-attention layers. For this assignment, you may use the MultiheadAttention class from PyTorch. Ensure you apply causal (unidirectional) attention, preventing tokens from attending to future positions (e.g., using the `attn_mask` parameter).
   (c) Feedforward layers. Implement two linear layers with a GELU non-linearity.
   (d) Layer normalization and residual connections. Apply pre-layer normalization before attention and feedforward layers, as in GPT-2.

5. Model hyperparameters:
   (a) Define hyperparameters for the model size, number of layers, embedding dimension, number of attention heads, and hidden layer size.
   (b) Use GPT-2 standard configurations (GPT-2 Small, Medium, Large, or XL) for your experiments.
6. Train the model on a next-word prediction task using your selected dataset:
   - Use the AdamW optimizer with weight decay for stable training.
   - Implement a learning rate scheduling consisting of warm-up followed by cosine decay for improved convergence.
   - Limit sequence length to 128 tokens to reduce memory usage.
   - Choose a batch size that fits your GPU memory (e.g., 8 or 16 on a single GPU).
   - If memory is limited, use gradient accumulation to simulate larger batches.
   - Report the cross-entropy loss on the training and validation sets at regular intervals (e.g., every X gradient updates).
   - Implement model checkpointing (save model weights periodically) to prevent loss of progress due to unexpected interruptions.
   - If you do not have a dedicated GPU, use free cloud-based resources such as Google Colab or Kaggle Notebooks.
7. Model evaluation:
   (a) Evaluate your trained model on unseen test data using the perplexity metric.
   (b) Experiment with different model sizes and compare their performance in terms of training efficiency, perplexity, and text generation quality.
8. Generate text samples using your trained model. Experiment with different decoding strategies, such as:
   - Greedy decoding: Always selects the highest probability token.
   - Top-k sampling: Limits choices to the top $k = 50$ highest probability tokens.
   - Nucleus sampling: Samples from the smallest set of tokens whose cumulative probability exceeds $p = 0.9$.

   Analyze how different sampling strategies affect fluency and diversity of generated text.
9. Submit a short report that includes:
   (a) Training loss curve showing both training and validation loss over time.
   (b) Perplexity results on the test dataset.
   (c) Generated text samples, showcasing the model's ability to complete sentences.
10. (Bonus) Explore more efficient attention mechanisms to speed up training, such as FlashAttention or using sparse attention methods.