Building a GPT-2 Transformer from Scratch

Pattern Recognition Project Report

---

1. Introduction

This project focuses on implementing a **GPT-2-style transformer language model from scratch** using PyTorch. Unlike typical approaches that rely on high-level transformer APIs (e.g., `torch.nn.Transformer` or Hugging Face's `transformers`), this implementation builds each core component manually. The model is trained on the **TinyStories** dataset, a corpus of short children's stories, and is evaluated based on its ability to generate coherent text.

Objectives

- Develop all fundamental components of GPT-2 manually: **multi-head self-attention, positional embeddings, feed-forward layers, and layer normalization**.
- Train the model on **next-token prediction** using the TinyStories dataset.
- Evaluate model performance both **quantitatively (via perplexity)** and **qualitatively (via generated outputs)**.

---

2. Methodology

2.1 Model Architecture

The model follows the **decoder-only GPT-2 architecture** with the following key components:

(1) Token and Positional Embeddings

- **Token Embedding:** Maps each token to a 768-dimensional vector.
- **Positional Embedding:** Learned embeddings are added to token embeddings to encode sequence order (instead of sinusoidal encodings).

(2) Multi-Head Self-Attention

- Implements **scaled dot-product attention** with 12 attention heads:
$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$
-
- A **causal mask** is applied to prevent tokens from attending to future positions (preserving the autoregressive property).

(3) Feed-Forward Network (FFN)

- Consists of two linear layers with a GELU activation in between:

$$\mathrm{FFN}(x) = W_2(\mathrm{GELU}(W_1 x + b_1)) + b_2$$

(4) Layer Normalization & Residual Connections

- Residual connections wrap both the attention and FFN modules, followed by **pre-layer normalization** for training stability.

(5) Decoder Stack

- The full model consists of **12 identical decoder blocks** stacked sequentially.

(6) Output Projection

- A linear layer projects the final hidden states to the vocabulary size (`50,257`) for next-token prediction.

2.2 Dataset and Preprocessing

- **Dataset:** TinyStories (from Hugging Face), ~2.7 million short stories (~10 million tokens).
- **Tokenization:** Byte-Pair Encoding (BPE) tokenizer using `tokenizers` library.
- **Input Preparation:**
    - Sequences truncated/padded to **512 tokens**.
    - Split: **90% training / 10% validation**.

2.3 Training Setup

- **Optimizer:** AdamW (`lr = 5e-5`, weight decay = 0.01).
- **Loss:** Cross-entropy over predicted token distributions.
- **Batch Size:** 32 (limited by ~16GB GPU memory).
- **Epochs:** 5 total (~12 hours on an NVIDIA T4 GPU).
- **Checkpointing:** Best model saved based on validation loss.

---

3. Results and Evaluation

3.1 Quantitative Evaluation

Metric	Training	Validation
Loss	2.31	2.89
Perplexity	10.1	18.0

- The model achieves reasonable perplexity on both sets.
- A moderate gap between training and validation suggests some **overfitting**.

3.2 Qualitative Evaluation

Prompt: *"Once upon a time, there was a"*
Generated Text:

"Once upon a time, there was a little rabbit who loved to explore the forest. One day, he found a shiny key and wondered what it could open."

Analysis:

- The output is grammatically correct and logically structured.
- Occasionally, the model **repeats phrases** or **loses context** in longer generations.

---

4. Discussion

Strengths

- ✅ Fully functional **GPT-2 architecture built from scratch**.
- ✅ Capable of generating **fluent and coherent** short-form text.
- ✅ Code is **modular and extensible** for experimentation.

Limitations

- ❌ Limited **long-range context retention**.
- ❌ Evidence of **overfitting** on the training set.
- ❌ **Slow training speed** due to hardware constraints.

Future Work

- 🔹 Scale up model depth and embedding size.
- 🔹 Incorporate **regularization techniques** (dropout, gradient clipping).
- 🔹 Experiment with **sampling strategies** (top-k, nucleus sampling) to improve generation diversity.

- 🔷 Fine-tune on larger datasets (e.g., OpenWebText) for more robust generalization.

---

5. Conclusion

This project successfully demonstrates the construction and training of a **GPT-2-style transformer model from first principles**. The model performs well on short-story generation tasks and provides a strong foundation for future exploration in generative language modeling.