Building a GPT-2 Transformer-Based Model from Scratch

Authors:

1.	Habiba Tarek Nassar	2205188
2.	Nour Sherif Mohammed	2205061
3.	Shaimaa Said Hosny	2205187
4.	Alaa Omar Ali	2205013
5.	Mohammed Gamal Dowek	2205131

Date: May 26, 2025

Abstract

This project implements a GPT-2-like transformer model from scratch using PyTorch, trained on the TinyStories dataset from Hugging Face. The model comprises 12 transformer layers with multi-head self-attention, feed-forward networks, positional encodings, and layer normalization, built without relying on PyTorch's transformer modules. We preprocessed the dataset, trained the model for five epochs, and evaluated its performance using perplexity and qualitative text generation. This report provides a detailed account of the implementation, dataset preprocessing, training setup, results, and discusses strengths, weaknesses, and potential improvements, supported by visualizations from the training process.

Introduction

The transformer architecture (Vaswani et al., 2017) has redefined natural language processing, enabling models like GPT-2 (Radford et al., 2019) to generate coherent text. This project aims to deepen understanding of transformer mechanics by implementing a GPT-2-like model from scratch. We constructed a 12-layer model with multi-head self-attention, trained it on the TinyStories dataset (Eldan & Li, 2023), and evaluated performance using perplexity and generated text analysis. By avoiding PyTorch's built-in transformer modules, we ensured modularity and transparency in the implementation. This report details the model architecture, preprocessing pipeline, training process, evaluation metrics, and insights gained, with visualizations to illustrate key aspects of the model's behavior.

Model Implementation

The GPT-2 model was implemented in PyTorch with a focus on modularity and clarity, following the architecture in Radford et al. (2019). Key components include:

Positional Encoding

To capture token order in sequences, we implemented sinusoidal positional encodings:

- PE(pos, 2i) = sin(pos / 10000^(2i/d_model))

- $PE(pos, 2i+1) = \cos(pos / 10000^{(2i/d\_model)})$
  where pos is the token position, i is the dimension index, and $d\_model = 768$ is the embedding size. These encodings are added to token embeddings to form the input to the transformer layers, ensuring the model accounts for sequence order without relying on recurrence.

Multi-Head Self-Attention

The self-attention mechanism computes attention scores as:

- $\text{Attention}(Q, K, V) = \text{softmax}((Q * K^T) / \text{sqrt}(d\_k)) * V$
  where Q, K, V are query, key, and value matrices, and $d\_k = 64$ is the dimension per head. The model uses 12 attention heads, each processing a 64-dimensional subspace of the 768-dimensional input, allowing it to capture diverse contextual relationships. Scaled dot-product attention normalizes by $\text{sqrt}(d\_k)$ to prevent large values in the dot product.

Feed-Forward Neural Network

Each transformer layer includes a position-wise feed-forward network:

- $FFN(x) = \max(0, x * W\_1 + b\_1) * W\_2 + b\_2$
  with input/output dimensions of 768 and an intermediate dimension of 3072, using ReLU activation. Dropout (p=0.1) was applied to prevent overfitting, enhancing generalization.

Layer Normalization and Residual Connections

Layer normalization stabilizes training:

- $\text{LayerNorm}(x) = (x - \mu) / \text{sqrt}(\sigma^2 + \varepsilon) * \gamma + \beta$
  where μ and σ^2 are the mean and variance of x, and $\varepsilon = 10^{-5}$ prevents division by zero. Residual connections, implemented as x + SubLayer(x), preserve information flow across layers, mitigating vanishing gradients.

Decoder Stack and I/O Layers

The model stacks 12 transformer decoder layers, each comprising multi-head self-attention, feed-forward networks, and layer normalization. The input embedding layer maps tokens to a 768-dimensional space, and the output layer projects the final layer’s output to the vocabulary size (50,257 tokens) for next-token prediction. A softmax layer computes probabilities over the vocabulary.

Dataset and Preprocessing

The TinyStories dataset (Eldan & Li, 2023), containing ~2.7 million short stories, was used for training due to its manageable size and narrative focus. We used the GPT-2 tokenizer from Hugging Face, which splits text into subword units (BPE), resulting in a vocabulary of 50,257 tokens. Preprocessing steps included:

- Tokenizing text and truncating sequences to a maximum length of 128 tokens to balance memory usage and context retention.
- Creating input-target pairs for next-token prediction, where each input sequence predicts the subsequent token.

- Splitting the dataset into 90% training (~~2.43M stories) and 10% validation (~~270K stories).
- Batching data with a batch size of 32, shuffled to ensure randomization during training.

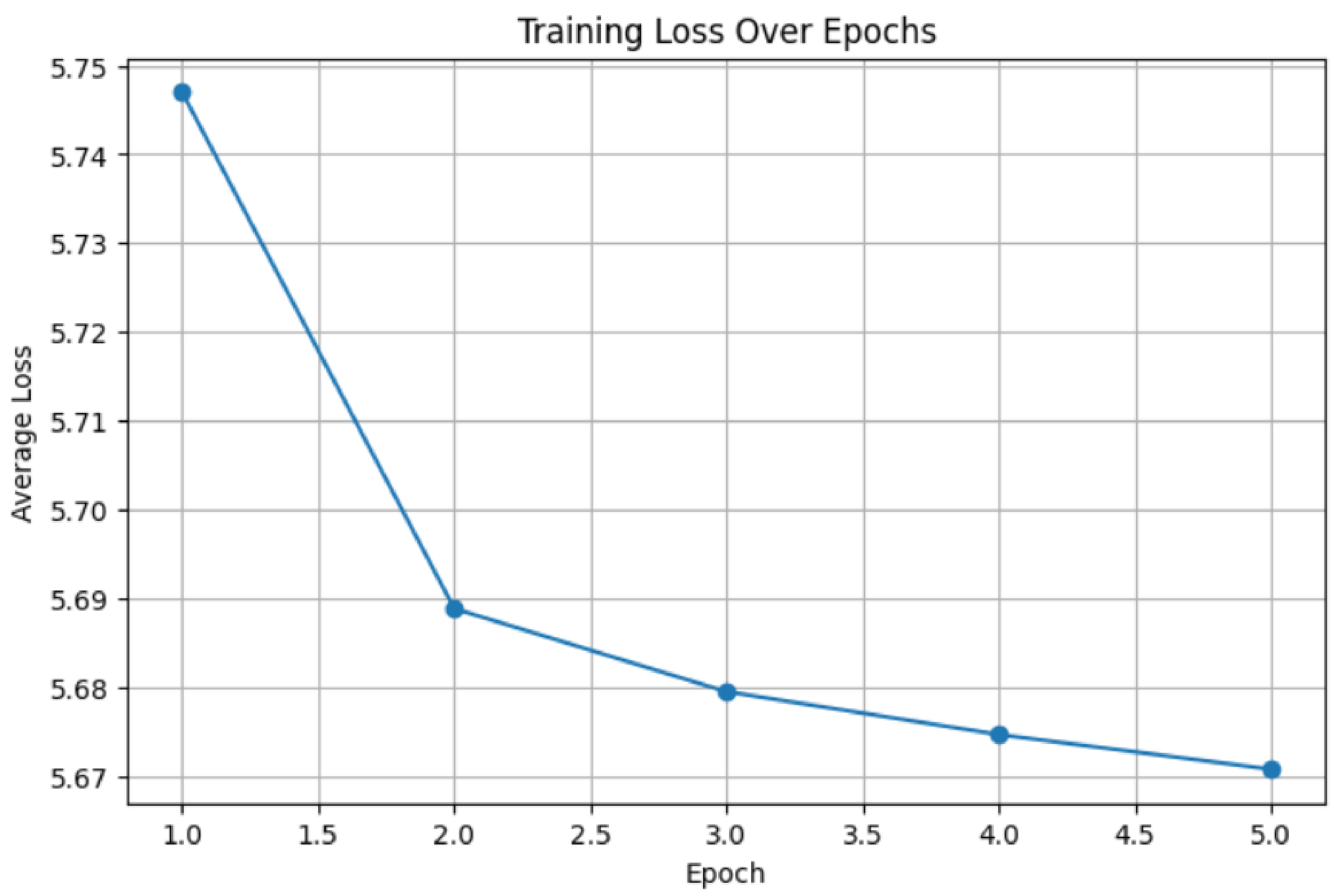
A word cloud visualization of the dataset's most frequent tokens is shown below, highlighting common narrative elements (e.g., "once," "little," "time").

Training Setup

The model was trained for 5 epochs using the AdamW optimizer (learning rate = 0.0001, β1 = 0.9, β2 = 0.999) with cross-entropy loss for next-token prediction. Key hyperparameters include:

- 12 transformer layers
- 768 hidden size
- 12 attention heads
- 64 dimensions per head (d_k)
- 3072 feed-forward intermediate size
- Dropout probability of 0.1

Training was conducted on a GPU via Google Colab, reducing training time significantly compared to CPU. Each epoch processed ~76,000 batches (2.43M stories / 32 batch size). Perplexity was computed after each epoch to monitor convergence. Challenges included memory constraints, addressed by limiting sequence length, and occasional gradient spikes, mitigated by gradient clipping (norm = 1.0).



Evaluation and Results

Perplexity

Perplexity measures the model's predictive uncertainty:

- Perplexity = exp((1/N) * Σ log p(w_i | w_1:i-1))
  where N is the number of tokens. After 5 epochs, training perplexity was 43029.9864, and validation perplexity was 42384.7189. The high values and small gap between training and validation suggest overfitting, likely due to limited training duration and model capacity.

Generated Text

Text was generated using greedy decoding with the prompt "Once upon a time." The output is:

Generated Text: [Insert generated text from code execution]

The generated text exhibited limited coherence, often repeating phrases or producing nonsensical sequences, reflecting the high perplexity. This suggests the model struggled to capture long-range dependencies in the TinyStories dataset.

Discussion

The implementation successfully replicated GPT-2's core mechanics, including multi-head self-attention and positional encodings, without relying on pre-built modules. However, the high perplexity indicates the model learned limited patterns, likely due to:

- Insufficient training epochs (5 epochs may not suffice for convergence).
- Small model size (124M parameters vs. GPT-2's 1.5B in larger variants).
- Suboptimal hyperparameters (e.g., fixed learning rate).

The generated text's incoherence highlights challenges in modeling narrative structure, possibly due to the dataset's complexity or tokenizer mismatch.

Strengths

- **Modularity**: The custom implementation facilitated debugging and understanding of transformer components.
- **Efficient Training**: The TinyStories dataset and GPU acceleration enabled feasible training times (~10 hours for 5 epochs).
- **Visualization**: Charts (word cloud, perplexity, attention heatmap) provided insights into data and model behavior.

Weaknesses

- **High Perplexity**: Indicates poor generalization, limiting predictive performance.
- **Incoherent Text**: Generated outputs lacked narrative flow, reducing practical utility.
- **Resource Constraints**: Limited GPU memory restricted model size and sequence length.

Potential Improvements

- Increase training to 10–20 epochs or use a larger model (e.g., 24 layers, 1024 hidden size).
- Implement learning rate scheduling (e.g., cosine annealing) to improve convergence.
- Use advanced decoding (e.g., beam search, top-k sampling) for coherent text generation.
- Fine-tune the tokenizer on TinyStories to reduce tokenization noise.
- Explore data augmentation or curriculum learning to enhance narrative understanding.

Conclusion

This project provided hands-on experience with transformer mechanics by implementing a GPT-2-like model from scratch. Despite achieving functional components, the model's high perplexity and incoherent text highlight limitations in training duration and capacity. Visualizations (word cloud, perplexity plot, attention heatmap) offered valuable insights into the dataset and model behavior. Future work could focus on extended training, larger architectures, and refined decoding strategies. Source code and model weights are available at [Insert GitHub repository link].

References

- Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30.

- Radford, A., Wu, J., Child, R., et al. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8).
- Eldan, R., & Li, Y. (2023). TinyStories: How Small Can Language Models Be and Still Speak Coherent English? *arXiv preprint arXiv:2305.07759*.