Building a GPT-2 Transformer-Based Model from Scratch

Pattern Recognition

May 13, 2025

Project Overview

The goal of this project is to deepen your understanding of transformer architectures by implementing a GPT-2-like model from scratch using either PyTorch or TensorFlow. You will build the model without using pre-defined transformer blocks provided by these libraries (e.g., `torch.nn.Transformer`). The model will be trained on a text-generation dataset from Hugging Face, and you will evaluate its performance on a text-generation task.

Objectives

- Implement a GPT-2 transformer model from scratch, including multi-head self-attention, feed-forward networks, positional encodings, and layer normalization.
- Train the model on a suitable text-generation dataset from Hugging Face.
- Generate coherent text samples and evaluate the model's performance.
- Document your implementation, experiments, and findings in a detailed report.

Dataset

You will use the **TinyStories** dataset available on Hugging Face (`https://huggingface.co/datasets/roneneldan/TinyStories`). This dataset contains approximately 2.7 million short, simple stories written for young children, totaling around 10 million tokens. It is ideal for training a small language model due to its manageable size and coherent narrative structure. You may preprocess the dataset as needed (e.g., tokenization, creating fixed-length sequences).

Tasks

1. Model Implementation

Implement the following components of the GPT-2 architecture from scratch:

- **Positional Encoding**: Add positional information to input embeddings to capture word order.
- **Multi-Head Self-Attention**: Implement the self-attention mechanism with multiple heads, including scaled dot-product attention.
- **Feed-Forward Neural Network**: Include a position-wise feed-forward network for each transformer layer.
- **Layer Normalization**: Apply layer normalization after attention and feed-forward sub-layers.
- **Residual Connections**: Incorporate residual connections around attention and feed-forward sub-layers.
- **Decoder Stack**: Stack multiple transformer decoder layers (e.g., 12 layers for a small GPT-2 model).
- **Embedding Layer**: Create token embeddings and map them to the model's hidden size.
- **Output Layer**: Project the decoder output to the vocabulary size for next-token prediction.

Use either PyTorch or TensorFlow, but do not use their built-in transformer modules. You may use basic linear layers, matrix operations, and activation functions provided by these libraries.

2. Training

- Preprocess the TinyStories dataset (e.g., tokenize, create input-target pairs for next-token prediction).
- Define a suitable loss function (e.g., cross-entropy loss for next-token prediction).
- Implement a training loop with an appropriate optimizer (e.g., AdamW).
- Use a small model configuration for feasibility (e.g., 12 layers, 768 hidden size, 12 attention heads).
- Train the model for at least 5 epochs or until reasonable convergence.

3. Evaluation

- Generate text samples using the trained model (e.g., using greedy decoding or sampling).
- Compute the perplexity of the model on a held-out test set from TinyStories.
- Qualitatively analyze the generated text for coherence and relevance.

4. Report

Write a report (max 5 pages, excluding references) that includes:

- A detailed explanation of your implementation, including mathematical formulations of key components (e.g., self-attention).
- Description of the dataset preprocessing and training setup.
- Results, including perplexity scores and sample generated texts.
- Discussion of the model's strengths, weaknesses, and potential improvements.

Deliverables

Upload the source code to Github, then submit the repository link to Google Classroom. The repository should include the following:

1. Source code (well-documented, including a README with instructions to run the code).
2. Trained model weights (or a link to download them).
3. A PDF of your report.

Grading Criteria

- **Correctness of Implementation (30%)**: Accurate implementation of GPT-2 components.
- **Training and Evaluation (30%)**: Proper training setup, preprocessing, and evaluation metrics.
- **Report Quality (30%)**: Clarity, depth, and completeness of the report.
- **Code Documentation and Demo (10%)**: Well-documented code and clear screenshots showing your model is working.

Resources

- Original GPT-2 paper: Radford et al., 2019.
- Attention mechanism: Vaswani et al., 2017.
- Build LLM from Scratch Github Repo.: `https://github.com/rasbt/LLMs-from-scratch`.
- Hugging Face Datasets: `https://huggingface.co/datasets`.
- PyTorch documentation: `https://pytorch.org/docs/stable/index.html`.
- TensorFlow documentation: `https://www.tensorflow.org/api_docs`.

Notes

- Each team should consist of 3 to 5 students.
- Every team member must actively contribute to the project and be familiar with their specific contributions, as they may be asked about them during the discussion.
- Start early—training language models can be computationally intensive and time-consuming.
- If you encounter resource limitations, it is acceptable to reduce the dataset size or use a smaller model.
- Make use of GPU resources when available (e.g., Google Colab).
- Write modular, reusable code to enhance clarity and maintainability.