Implementation of a GPT-2 model report

Pattern Recognition

Dr. Saleh Shehaby , Ts : Ziad Shokry

May 27 –2025

Team members :

1. *Roaa Othman Mehanna 2205176*
2. *Jolie Nabil Zaki 2205016*
3. *Maya Hazem Nseef 2205042*

1. Introduction

This report details the implementation of a GPT-2-like language model trained on the TinyStories dataset. The model is designed to generate coherent short stories while being computationally efficient enough to train on limited hardware resources. The implementation includes all key components of the transformer architecture, including multi-head self-attention and position-wise feed-forward networks

---

2. Model Architecture

2.1 Key Components

The model follows the decoder-only transformer architecture used in GPT-2, with the following key components:

- Multi-head self-attention
- Layer normalization
- Feed-forward neural network
- Residual connections

2.1.1 Multi-Head Self-Attention

The self-attention mechanism allows the model to weigh the importance of different words in the input sequence when generating each output token.

Formula :

1. **Query, Key, Value projections:**

$Q=XWQ,K=XWK,V=XWVQ=XWQ,K=XWK,V=XWV$

where XX is the input matrix and $WQ,WK,WVWQ,WK,WV$ are learned projection matrices.

2. **Scaled dot-product attention:**

$Attention(Q,K,V)=softmax(QKTdk)V$Attention($Q$,$K$,$V$)=softmax($dk$ QKT)V

where $dkdk$ is the dimension of the key vectors.

3. **Multi-head attention concatenates multiple attention heads:**

$MultiHead(Q,K,V)=Concat(head1,...,headh)WO$MultiHead($Q$,$K$,$V$)=Concat(head1 ,...,head$h$)$W O$

where each head performs attention independently.

2.1.2 Position-wise Feed-Forward Network :

The FFN applies two linear transformations with a GELU activation in between:

$FFN(x)=W2(GELU(W1x+b1))+b2$FFN($x$)=$W$2 (GELU($W$1 x+b1))+b2

2.1.3 Layer Normalization and Residual Connections :

Each sub-layer (attention and FFN) is followed by layer normalization and residual connections: $\boldsymbol{x}out$ =x+Dropout(SubLayer(LayerNorm(x)))

2.2 Model Configuration

The implemented model uses:

- Vocabulary size: 10,000 tokens
- Model dimension (d_model): 512

- Number of layers: 6
- Number of attention heads: 8
- Feed-forward dimension: 2048
- Maximum sequence length: 256 tokens

---

3. Dataset and Preprocessing

3.1 TinyStories Dataset

The TinyStories dataset consists of short, simple stories written in English, designed for training small language models. The dataset was split into:

- Training set: 90% of stories
- Validation set: 10% of stories

3.2 Tokenization

A Byte Pair Encoding (BPE) tokenizer was trained with:

- Vocabulary size: 10,000 tokens
- Special tokens: [UNK], [PAD], [BOS], [EOS]
- Pre-tokenization: Whitespace splitting

3.3 Training Setup

- Batch size: 16
- Learning rate: 3e-4 (AdamW optimizer)
- Training epochs: 5
- Gradient clipping: 1.0
- Dropout rate: 0.1

- Training subset: 10% of full dataset (for efficiency)

---

4. Results

Sample Generated Text :

Prompt: "Once upon a time"

Output:

Once upon a time, there was a little girl named Lily. She loved to read stories and dream about magical lands. One day, she found a glowing stone...

---

5. Discussion

5.1 Strengths

- **Interpretability**: Implementing from scratch provides deep insight into model internals.
- **Performance**: Reasonable perplexity and coherence despite small dataset.
- **Flexibility**: Architecture is modular and extendable for larger tasks.

5.2 Weaknesses

- **Limited Training Data**: Small dataset limits generalization and long-range coherence.
- **Compute Constraints**: Small batch size and short training time affect model performance.
- **Tokenization**: A more robust pre-trained tokenizer could improve consistency.

References

1. HuggingFace Tokenizers: https://huggingface.co/docs/tokenizers
2. TinyStories Dataset: https://huggingface.co/datasets/roneneldan/TinyStories

3. PyTorch documentation: PyTorch documentation — PyTorch 2.7 documentation