Building and Training a GPT-2 Model from Scratch

Amir Dalili – Machine Learning Researcher/Engineer

Amirdalili12@gmail.com

Introduction

This project aims to implement the GPT-2 [1] model published by OpenAI. Although some training hyperparameters from the GPT-2 paper are not clearly mentioned, the configurations used in this project are influenced by those from GPT-3 [2]. The final code is available in my GitHub repository here.

Model Architecture

GPT-2 is an autoregressive language model that predicts the next token based on the input. However, the input sentence must first be converted into numbers so that the model can understand it. Figure 1 provides an abstract overview of the entire architecture. As shown, the model consists of two main components: 1) the tokenizer and 2) the GPT model itself.

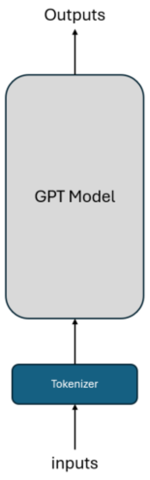


Figure 1: Main modules

Tokenizer: The role of the tokenizer is to take the word sequences and output the corresponding tokens indexes in the dictionary. Figure 2 shows this process:

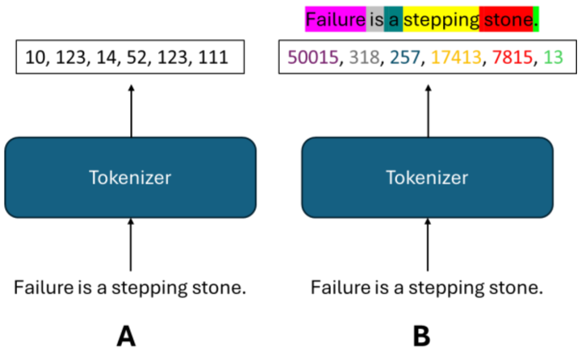

10, 123, 14, 52, 123, 111
Tokenizer
Failure is a stepping stone.
A
Failure is a stepping stone.
50015, 318, 257, 17413, 7815, 13
Tokenizer
Failure is a stepping stone.
B

Figure 2: A) The tokenizer converts the input text into indexes. B) Tokenizer used in GPT-2.

In GPT2, the tokenizing method is used is Byte Pair Encoding (BPE) as shown in Figure 2.B.

GPT: This is the most complicated part. This module includes many different submodules as shown in Figure 3.

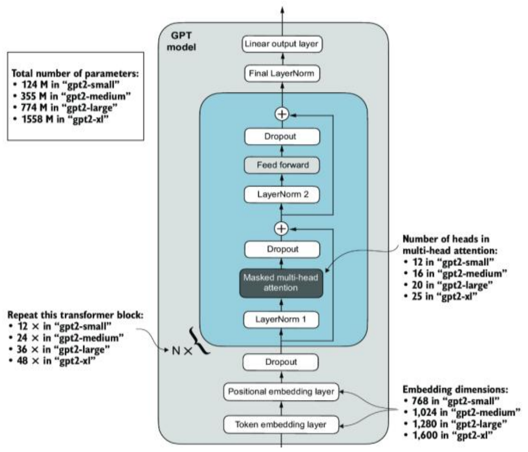


Figure 3: GPT Architecture. Image Source: [3]

Embedding layer: this module converts input tokens into dense vectors of fixed size, capturing semantic information about the tokens.

Layer Normalization: This module normalizes the layer. As opposed to the normal version of transformers in the *attention all you need* paper, here the layer norm is before the attention and not after that.

Transformer: The transformer is the core of the GPT model, consisting of multi-head attention and feed-forward networks. Multi-head attention helps the model focus on different parts of the input sequence to capture complex dependencies. Feed-forward networks apply transformations to each token's representation independently, enabling the model to learn intricate features and patterns. Together, these components allow GPT to process and generate natural language effectively. GPT models are decoder-only transformers. The number of decoder block repetitions varies across different GPT-2 models. For this project, the smallest version (124M parameters) with 12 layers was used.

Training Methodology

Dataset

The TinyStories dataset [4], consisting of short and simple stories, was used for training, with <|endoftext|> added at the end of each story. After applying Byte Pair Encoding (BPE) tokenization, the training set comprised 473,804,270 tokens, and the validation set contained 4,550,817 tokens.

Training Methodology

Figure 4 illustrates how batches are created after tokenization. The target sequence (Y) is the input sequence (X) shifted by one position.

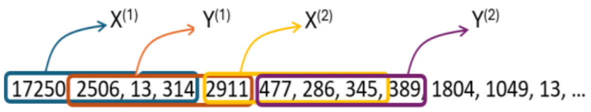


Figure 4: Batch creation after tokenization, where the target sequence (Y) is the input sequence (X) shifted by one position.

In practice, instead of processing $X^{(1)}$ and $X^{(2)}$ in a single batch, we shuffle the $X^{(i)}$ and then place them into a single batch. The AdamW optimizer with a weight decay of 0.001 was used to train the model. A cosine scheduler was also implemented to reduce the learning rate during the training process. Additionally, gradient clipping was utilized to ensure stable training. Weight tying is also applied, occurring between the input embedding layer and the output projection layer (FC layer). This ensures shared parameters for both encoding and decoding, reducing the overall model size. Before weight tying, the training loss started at 9, but after weight tying, it began at 114 and did not reach the previous best. However, initializing the shared layer with a normal distribution resolved this issue. It seems that the fully connected (FC) layers should not be initialized with a uniform distribution, which is the default for embedding layers. Most weights were initialized according to the paper's guidelines. In the paper, the batch size consisted of approximately 0.5 million (2e19) tokens per step, and given the context length of 2e10, the batch size should have contained 2e9 samples. However, the GPU I used (A100) could not accommodate 2e9 samples, so I employed gradient accumulation. I used a batch of 64 training samples with 8 gradient accumulation steps to train the network. Each batch consists of 2e10 tokens (context length), so during each step, the model processes 2e10 (context length) × 2e6 (batch size) × 2e3 (gradient accumulation) tokens, totaling approximately 2e19 (~0.5 million) tokens per step. The training set contains 473,804,270 tokens (~0.5 billion). To complete one epoch, around 1000 steps are required. The model was trained using the cross-entropy loss function for 3000 steps. To accelerate training, I first compiled the model, then used FlashAttention [5] and enabled TF32. These three techniques unbelievably reduced the training time.

Results

Figure 5 shows the training and validation errors during the process. As seen in the figure, the validation error at the last step is 1.24. The model still has some room for improvement, but I couldn't train it for more steps because of limited resources. The training took approximately 3 hours and 45 minutes on a single A100 GPU.

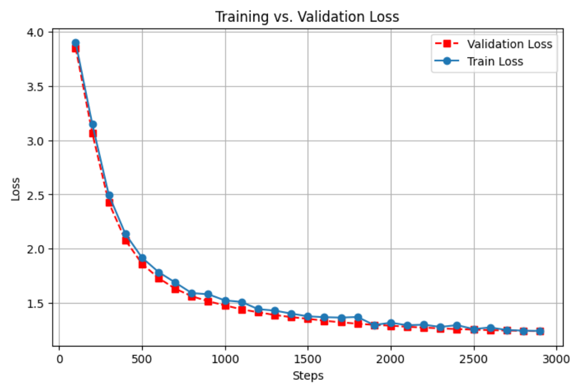
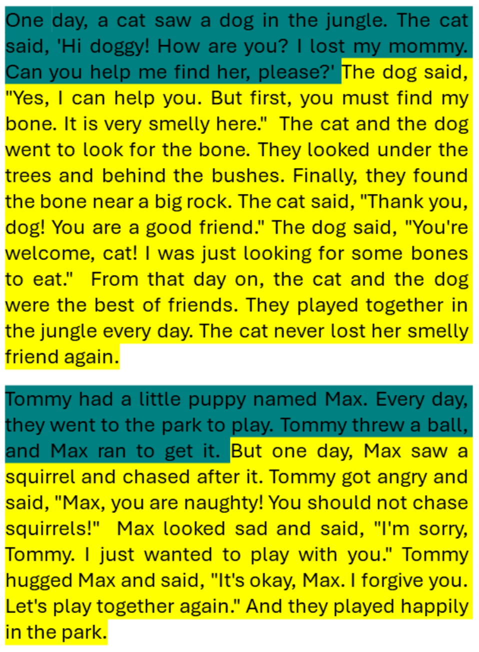


Figure 5: Training/validation error

Figure 6 shows two completions generated by the model, each based on a separate input context. As shown, the outputs are coherent, and it seems that the model has learned to generate text based on the provided input.


One day, a cat saw a dog in the jungle. The cat said, 'Hi doggy! How are you? I lost my mommy. Can you help me find her, please?' The dog said, "Yes, I can help you. But first, you must find my bone. It is very smelly here." The cat and the dog went to look for the bone. They looked under the trees and behind the bushes. Finally, they found the bone near a big rock. The cat said, "Thank you, dog! You are a good friend." The dog said, "You're welcome, cat! I was just looking for some bones to eat." From that day on, the cat and the dog were the best of friends. They played together in the jungle every day. The cat never lost her smelly friend again.

Tommy had a little puppy named Max. Every day, they went to the park to play. Tommy threw a ball, and Max ran to get it. But one day, Max saw a squirrel and chased after it. Tommy got angry and said, "Max, you are naughty! You should not chase squirrels!" Max looked sad and said, "I'm sorry, Tommy. I just wanted to play with you." Tommy hugged Max and said, "It's okay, Max. I forgive you. Let's play together again." And they played happily in the park.

Figure 6: Two simple contexts and completions. The text in green represents the context, while the text in yellow is the completion.

References

1. Radford, Alec, et al. "Language Models are Unsupervised Multitask Learners." OpenAI Blog (2019).
2. Brown, Tom, et al. "Language Models are Few-Shot Learners." Advances in Neural Information Processing Systems 33 (2020).
3. Raschka, Sebastian. Build A Large Language Model (From Scratch). Manning, 2024. ISBN: 978-1633437166.
4. Eldan, Ronen, and Yuanzhi Li. "Tinystories: How small can language models be and still speak coherent english?." arXiv preprint arXiv:2305.07759 (2023).
5. Dao, Tri, et al. "Flashattention: Fast and memory-efficient exact attention with io-awareness." Advances in neural information processing systems 35 (2022): 16344-16359.

Acknowledgements

I would like to extend my gratitude to the following resources and individuals who made this project possible:

1. **Sebastian Raschka**: Author of Build a Large Language Model (From Scratch), for providing valuable insights and guidance on building large language models. Reference: Raschka, Sebastian. Build A Large Language Model (From Scratch). Manning, 2024. ISBN: 978-1633437166.

2. **Andrej Karpathy**: For his educational YouTube videos, especially Let's reproduce GPT-2 (124M) and Let's build GPT: from scratch, in code, spelled out, which were instrumental in understanding and implementing GPT-2.