A Mathematical Blueprint for a Generative Pre-trained Transformer

Sankalp Yadav

July 18, 2025

Abstract

This document provides a formal blueprint for the implementation of a decoder-only Generative Pre-trained Transformer (GPT). I frame the project from a mathematical and algorithmic perspective. I define language modeling as a probabilistic sequence generation task and detail the underlying mathematical operations of the Transformer architecture, including token and positional embeddings, the scaled dot-product self-attention mechanism, multi-head attention, and position-wise feed-forward networks. Furthermore, I specify the optimization objective using the cross-entropy loss function and outline the auto-regressive process for text generation. This paper serves as the foundational plan for a principled and reproducible implementation.

1 Problem Formulation: Autoregressive Language Modeling

The fundamental goal of a language model is to assign a probability to a sequence of tokens. Given a sequence of tokens $X = (x_1, x_2, \ldots, x_T)$, its probability is factorized autoregressively as a product of conditional probabilities:

$$P(X) = \prod_{t=1}^{T} P(x_t | x_1, x_2, \ldots, x_{t-1}) \tag{1}$$

Our objective is to build a neural network, parameterized by θ, that models this conditional probability $P_\theta(x_t | x_{<t})$. The model takes a sequence of context tokens $x_{<t}$ and outputs a probability distribution over the entire vocabulary for the next token x_t.

2 Model Architecture: The Decoder-Only Transformer

I will implement a decoder-only Transformer model. The model is a stack of identical blocks, each performing a sequence of computations.

2.1 Input Embedding

The model cannot operate on raw text. Tokens are first mapped to dense vectors.

1. **Token Embedding:** Each token x_i from a vocabulary V is mapped to a continuous vector $e_{x_i} \in \mathbb{R}^{d_{\text{model}}}$ via a learnable embedding matrix W_e.

2. **Positional Embedding:** To provide the model with information about the order of tokens, a learnable positional embedding $p_i \in \mathbb{R}^{d_{\text{model}}}$ is added to each token embedding.

The final input representation for a token x_i is $h_i^{(0)} = e_{x_i} + p_i$.

2.2 The Transformer Block

The core of the model is a stack of N identical blocks. Each block has two main sub-layers.

2.2.1 Masked Multi-Head Self-Attention

The self-attention mechanism allows tokens to interact and aggregate information from each other. For a sequence of input vectors $H = (h_1, \ldots, h_T)$, I compute Query (Q), Key (K), and Value (V) matrices:

$$Q = HW_Q, \quad K = HW_K, \quad V = HW_V \tag{2}$$

where $W_Q, W_K, W_V \in \mathbb{R}^{d_{\text{model}} \times d_k}$ are learnable projection matrices. The scaled dot-product attention is then calculated as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T + M}{\sqrt{d_k}}\right) V \tag{3}$$

Here, d_k is the dimension of the key vectors. The mask matrix M is a lower-triangular matrix of zeros and $-\infty$ that prevents positions from attending to subsequent positions, preserving the autoregressive property.

Multi-Head Attention consists of running h attention mechanisms ("heads") in parallel and concatenating their results:

$$\text{MultiHead}(H) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h) W_O \tag{4}$$

$$\text{where } \text{head}_i = \text{Attention}(HW_Q^{(i)}, HW_K^{(i)}, HW_V^{(i)}) \tag{5}$$

Here, $W_Q^{(i)}, W_K^{(i)}, W_V^{(i)}$ are the projection matrices for the i-th head, and W_O is an output projection matrix.

2.2.2 Position-wise Feed-Forward Network

This is a two-layer multi-layer perceptron (MLP) applied independently to each position:

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{6}$$

2.2.3 Residual Connections and Layer Normalization

Each sub-layer in the block is wrapped with a residual connection and layer normalization:

$$H' = \mathrm{LayerNorm}(H + \mathrm{MultiHead}(H)) \tag{7}$$

$$H_{\mathrm{out}} = \mathrm{LayerNorm}(H' + \mathrm{FFN}(H')) \tag{8}$$

2.3 Output Layer

After the final Transformer block, a linear layer followed by a softmax function is used to produce the probability distribution over the vocabulary for the next token:

$$P(x_t|x_{<t}) = \mathrm{softmax}(H_{\mathrm{out}}^{(t-1)} W_e^T) \tag{9}$$

Note the weight tying between the input embedding matrix W_e and the final output layer.

3 Training and Optimization

The model parameters θ are trained to minimize the negative log-likelihood of the training data.

3.1 Objective Function

For a training corpus $\mathcal{D}$, the objective is to minimize the cross-entropy loss:

$$\mathcal{L}(\theta) = -\sum_{X \in \mathcal{D}} \sum_{t=1}^{T} \log P_\theta(x_t|x_{<t}) \tag{10}$$

3.2 Optimizer

I will use the AdamW optimizer, a variant of Adam that decouples weight decay from the gradient update, which often leads to better generalization.

4 Inference: Text Generation

New text is generated autoregressively.

1. Provide the model with a starting context sequence (a prompt).
2. The model computes the probability distribution for the next token.
3. A token is sampled from this distribution (e.g., via top-k sampling or nucleus sampling) and appended to the sequence.
4. This new sequence becomes the context for the next step.
5. Repeat until a desired length or an end-of-sequence token is generated.

5 Conclusion

This document has laid out the mathematical and algorithmic foundation for our project. By adhering to this formal blueprint, I will implement a GPT model, ensuring that each component is grounded in the established principles of the Transformer architecture and probabilistic language modeling.

References

[1] Karpathy, A. (2023). *Let's build GPT: from scratch, in code, spelled out.* [Video]. YouTube. https://www.youtube.com/watch?v=kCc8FmEb1nY

[2] Vaswani, A., et al. (2017). *Attention is all you need.* Advances in neural information processing systems, 30.