The GPT Transformer: A Complete Mathematical and Architectural Analysis

Sean Diab

18 June 2025

Abstract

This paper presents a comprehensive, technical analysis of the decoder-only, autoregressive transformer architecture used in state-of-the-art chatbot models. Our goal is to cover most implementation and mathematical details while maintaining readability. We begin with a detailed overview of the forward pass, followed by technical operation steps, and conclude with the derivation of all backpropagation equations. While many resources provide an overview of GPT architecture, a comprehensive, technical source is virtually nonexistent. This paper aims to fill that gap by serving as a standalone reference for students, researchers, and practitioners seeking a full understanding of how state-of-the-art language models like GPT operate and learn. In particular, it is targeted towards researchers aiming to improve upon current implementations, as advancements are best made from a solid foundation.

1 Introduction

Despite the widespread use and interest in Large Language Models (LLMs) such as ChatGPT (Brown et al., 2020), obtaining a complete technical understanding of how they work proves to be more challenging than necessary. Introductory materials such as videos or articles offer intuitive explanations of the structure and architecture of transformers, and many technical sources might outline specific components, such as the forward pass of a transformer, or code for an open source model. There is a noticeable absence, however, of a mathematically precise explanation that covers everything from token input to final prediction and backpropagation.

This paper addresses that gap by providing a comprehensive walkthrough of the autoregressive, decoder-only GPT architecture, covering embeddings, attention, layer normalization, FFN blocks, output layers, etc. We begin with a detailed overview of the feedforward processes, then provide technical operation steps of a single forward pass, concluding with a readable derivation of all gradient descent equations used for backpropagation (Rumelhart et al., 1986). Figure 1 provides a visual of the forward pass and backpropagation. Our aim is to make GPT fully understandable to practitioners, students, and researchers seeking more than a surface-level understanding of state-of-the-art chatbot models. The final result is a standalone technical reference for the inner workings of modern autoregressive transformers.

It is important to note that many of the most recent state-of-the-art models have been closed-source. This paper adapts to this by adhering to the most recent top GPT implementations that have technical reports describing their architectures, and notifies the reader when there are various successful implementations for a certain section.

Contents

2 GPT FeedForward Overview

Parameter(s)	Shape	Description
x	$\mathbb{Z}^n$	Input, a vector of token indices
E_{token}	$\mathbb{R}^{V \times d_{\text{model}}}$	Token embedding matrix
E_{pos}	$\mathbb{R}^{n_{\max} \times d_{\text{model}}}$	Positional embedding matrix
W_Q, W_K, W_V	$\mathbb{R}^{d_{\text{model}} \times h \cdot d_{\text{head}}}$	Attention projection weights
$\beta_Q, \beta_K, \beta_V$	$\mathbb{R}^{h \cdot d_{\text{head}}}$	Biases for Q/K/V projections
W_O	$\mathbb{R}^{(h \cdot d_{\text{head}}) \times d_{\text{model}}}$	Output projection weight matrix after concatenation
β_O	$\mathbb{R}^{d_{\text{model}}}$	Bias for output projection
γ_1, β_1	$\mathbb{R}^{d_{\text{model}}}$	Scale and shift parameters for LayerNorm 1
γ_2, β_2	$\mathbb{R}^{d_{\text{model}}}$	Scale and shift parameters for LayerNorm 2
W_1	$\mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}$	First FFN weight matrix
β_1^{ffn}	$\mathbb{R}^{d_{\text{ff}}}$	Bias for first FFN layer
W_2	$\mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}$	Second FFN weight matrix
β_2^{ffn}	$\mathbb{R}^{d_{\text{model}}}$	Bias for second FFN layer
$\gamma_{final}, \beta_{final}$	$\mathbb{R}^{d_{\text{model}}}$	Scale and shift parameters for Final LayerNorm
W_{lm}	$\mathbb{R}^{d_{\text{model}} \times V}$	Final language model head weight matrix
β_{lm}	$\mathbb{R}^V$	Bias for final language model head

Table 1: Parameter definitions. Shape variables with base d refer to the dimensionality of the subscript, n is the input length, V is the vocabulary size, and h is the number of heads.

Note on Per-Block Parameters: The model architecture consists of L transformer blocks stacked sequentially with input $X^{(l)}$. The parameters listed above from the attention weights through β_2^{ffn} represent the parameters required for one such block.

Note on Per-Head Parameters: Within each block are h attention heads. As described in section 2.2, the weights and biases for Q, K, and V listed in Table 1 represent the parameters required for one block, and are split up column-wise for each head.

Note on Biases: The biases are added row-wise. Formally, if $C = A + \beta$, then $C_i = A_i + \beta$.

2.1 Processing the Input

The initial input x to a large language model is a string of text, which gets converted into a string of tokens via a tokenizer. Tokens are pieces of text such as words, word parts, numbers, etc. For example, `"curiosity"` might get converted into:

`[<SOS>, "cur", "ios", "ity", <EOS>]`.

Here, `<SOS>` and `<EOS>` are the start of sentence and end of sentence tokens, respectively. In order to convert this input into numerical format, each token is replaced with its corresponding index. Since transformers take in a fixed length, the input is padded if it is too short. The result is something like (where 0 is the padding token ID):

`[1, 1543, 8821, 327, 2, 0, ..., 0]`.

The numerized and padded input is now fed into two trained embedding matrices: a matrix that embeds each token and a matrix that embeds each position, and are added together token-wise:

$$X_i^{(0)} = E_{\text{token}}[x_i] + E_{\text{pos}}[i].$$

Next, the embedded input goes through a LayerNorm (Ba et al., 2016), which is applied row-wise (per token). Layernorm normalizes each token to have zero mean and unit variance, then multiplies the result row-wise by a scaling factor γ and shifting bias β, which are both row vectors of shape $\mathbb{R}^{d_{\text{model}}}$.

2.2 Attention

The output of the LayerNorm, X_{ln1}, is fed through multiple transformer blocks sequentially, where the output of one block is the input to the next (Vaswani et al., 2017). Each block has multiple heads, which are computed in parallel and then concatenated together.

Each transformer block has an attention mechanism, which has three weight matrices and three bias vectors, one for Q, K, and V (the query, key, and value matrices). X_{ln1} is now multiplied and added to the corresponding weights and biases (e.g., $Q = X_{\text{ln1}}W_Q + \beta_Q$) to obtain Q, K, and $V \in \mathbb{R}^{n \times d_{\text{model}}}$. Since there are multiple attention heads in each block, Q, K, and V are split up column-wise, for example:

$$Q = \text{Concat}(Q_1, \ldots, Q_h), \quad Q_j \in \mathbb{R}^{n \times d_{\text{head}}}, \quad \text{where } d_{\text{head}} = d_{\text{model}}/h.$$

Next, we calculate attention scores. The description below will be for one head, implicitly assuming its index j, but the calculation is done for each head in parallel. Matrix multiply Q and $K^\top$ together, then scale by $1/\sqrt{d_{\text{head}}}$. Since we don't want the padding tokens to affect other tokens during attention, we apply a mask that turns all padding tokens to -10^9 (or another large negative number), then softmax the result across rows, which turns them to zero.

$$A = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_{\text{head}}}} + M\right).$$

Note: M is actually the addition of the mask described above and another, which is discussed in the paragraph titled "Note on Masks":

$$M = M_{\text{padding}} + M_{\text{causal}},$$

The output of the attention head is calculated by matrix multiplying $A \in \mathbb{R}^{n \times n}$ with V:

$$\text{AttentionHead} = AV.$$

All attention head outputs for a particular block are concatenated together, resulting in a matrix $H \in \mathbb{R}^{n \times d_{\text{model}}}$. A linear layer (affine transformation) is then applied, preceding a residual connection and a second LayerNorm.

Note on Attention: Since $A \propto Q \cdot K$, it is tempting to assume that Q_i encodes information about token i and K_j encodes information about token j, and their dot product measures how much token i attends to token j, which is used to scale the value matrix. However, this interpretation is overly simplistic. Each set of Q, K, and V belongs to a single attention head within one transformer block, and as such is a single component of a deeper computation which captures complex, high-dimensional relationships. What makes attention so powerful is that, since the input to attention turns into three matrices which are multiplied together, the attention mechanism gives the model flexibility to exchange information across that matrix.

Note on Masks: Since language models cannot see future tokens, another mask $M \in \mathbb{R}^{n \times n}$ is applied before the softmax so that each token can only attend to tokens after it (or so each token can only pay attention to tokens before it). This is done by adding the mask to the padding mask described earlier. The typical mask that achieves such effect is called a causal mask.

M_{causal} is defined such that:

$$(M_{\text{causal}})_{i,j} = \begin{cases} -\infty & \text{if } i < j \\ 0 & \text{otherwise} \end{cases}.$$

Another popular mask used instead of a causal mask is a sparse mask (Child et al., 2019), which only lets each token pay attention to itself and $w-1$ tokens before it (where w is the window size). Sparse masks reduce attention computation while maintaining similar performance (or sometimes even improving it). M_{sparse} is defined such that:

$$(M_{\text{sparse}})_{i,j} = \begin{cases} 0 & \text{if } j \in [i-w+1, i] \\ -\infty & \text{otherwise} \end{cases}.$$

In practice, a large negative constant like -10^9 is used instead of $-\infty$ to avoid NaNs. There are many different masks that are used in GPTs, but due to the recent closed-source nature of top models, it is difficult to find out exactly which masks are being used. Mixing masks can also prove promising. For example, GPT-3 alternated between causal and sparse masks per transformer block (Brown et al., 2020).

2.3 Through a Feedforward Neural Network

After the second LayerNorm, X_{ln2} is fed into a FFN with two layers that has a GELU activation function after the first (Hendrycks and Gimpel, 2016). The first linear layer is applied independently to each row and expands the dimensionality from d_{model} to a higher hidden dimension, often $4\times$ larger. After the GELU, the second layer projects back to the regular dimension. The output, X_{ffn}, then goes through a second residual connection.

Note: The GELU activation function is originally defined as $\text{GELU}(x) = x \cdot \Phi(x)$, where $\Phi(x)$ is the CDF of the normal distribution. However, due to its calculation being resource-intensive, most implementations use an approximation.

All steps from the first LayerNorm to the second residual connection are repeated for each transformer block.

2.4 Output

After the last transformer block, there is a final LayerNorm, then a final linear projection that maps the matrix to $\mathbb{R}^{n \times V}$. Often this linear projection "unembedding" matrix is the transpose of the embedding matrix (Press and Wolf, 2016). This reduces the number of trainable parameters, which improves generalization and also saves compute and memory. After a softmax is applied row-wise, the output is a resulting probability distribution over all possible tokens. For training, this entire output is used for error correction. For inference (predicting the next token in practice), the model does not simply output the token with the highest associated probability. Instead, a technique called top-p sampling is used to make the model nondeterministic (Holtzman et al., 2019), which improves diversity in the model's outputs. Top-p sampling takes the smallest subset of tokens such that:

$$\sum_{t \in \text{top-p}} P_n[t] \geq p$$

for a threshold p (a common value for p is 0.9). The probabilities of this subset are renormalized and then sampled from a categorical distribution.

For inference, temperature is also applied. The goal of temperature is to either widen or narrow the outputted probability distribution. To implement this, the input is divided by T (the temperature scalar) right before being fed into the final softmax. A temperature less than 1 sharpens the probability distribution, lowering the entropy of the model. A temperature greater than 1 flattens the probability distribution, allowing for more diverse outputs. This makes the model more creative. A typical temperature used in practice is $T = 0.7$. For training, we are only concerned with the accuracy of the outputted probability distribution, so temperature is not used.

2.5 Training

The parameters of a language model take on matrix and vector forms, which consist of a collection of numbers. These numbers must be "trained" in order to produce optimal results. For now, gradient descent has proven to be the best way to train language models.

Consider a graph relating all trainable parameters to a loss $\mathcal{L}$ calculated using a loss function, which measures how incorrect an output is. The points on this high-dimensional graph that minimize $\mathcal{L}$ would then offer strong performance. The idea of gradient descent is to find a combination of parameters that result in a minimal $\mathcal{L}$ by first finding the gradient of the parameters with respect to $\mathcal{L}$, then taking steps in the direction of steepest descent (the negation of the gradient).

Note: The most common loss function is cross entropy:

$$\mathcal{L} = -\frac{1}{n} \sum_{i=1}^{n} \log P_i[y_i].$$

In practice, gradient descent is implemented using the backpropagation algorithm, which first finds the gradient of the loss with respect to the parameters that affect it directly, at the very end of the transformer. Then, using the chain rule, this gradient gets backpropagated to the parameters at the beginning. Once the gradient of the loss $\mathcal{L}$ with respect to all trainable parameters has been found, they are updated simultaneously by the update rule:

$$\theta \leftarrow \theta - \eta \nabla_\theta,$$

where θ represents the parameters being updated, and η is the learning rate (a small positive scalar). Throughout this paper, we implicitly infer that the gradient is with respect to the loss $\mathcal{L}$. There has been extensive research done on optimizing the update rule. At the time of this paper, the industry standard is AdamW (Kingma and Ba, 2014) (Loshchilov and Hutter, 2017), defined as:

$$\theta \leftarrow \theta - \eta \cdot \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon} + \lambda \theta \right),$$

where $\lambda\theta$ is the weight decay term, providing a penalty proportional to that of the parameter size. This is a regularization technique because it encourages parameter size reduction, which helps prevent overfitting.

The second main regularization technique in transformers is dropout (Srivastava et al., 2014). Dropout randomly zeroes out p percent of the components of a matrix. Randomly dropping a subset of neurons prevents the network from relying on specific neurons, forcing it to develop robust representations. Dropout is only used during training, and turned off during inference. It is typically applied after the attention output, feedforward layers, residual connections, and sometimes after the embeddings, but the exact placements can vary between implementations.

2.6 Beyond Next-Token Prediction

In addition to the supervised learning techniques described above, modern language models typically employ two techniques to allow them to reason further.

The first is hidden reasoning, where the model is allowed to "think" before responding. Forcing the model to respond without reasoning beforehand can often result in a spontaneous, inaccurate answer. For example, if a non-thinking model is asked a multiple choice question, it will often respond with an answer right away without allowing some of its response to constitute reasoning.

It is possible to stop the LLM from spontaneously answering right away by fine-tuning it on data that meticulously reasons through a question before answering it, or by prompting it to do so. This is a form of chain-of-thought (CoT) reasoning, but it requires extensive amounts of specialized data. A more efficient solution is to reserve space in the model's output not directly seen by the user that the model can use to think, for example between special delimiters like `<think>...</think>` (exact implementations can vary) (Goyal et al., 2023). The model then chooses when to switch to the actual response seen by the user. In training, only the visible response is graded. This indirectly improves the model's thoughts, as better thoughts will lead to a better response.

Hidden reasoning allows a model to train its thoughts without explicit data telling it how to think. In this regard, a hidden reasoning model is given more autonomy over its own optimization, as it is allowed to "think for itself". The model is still autoregressive, but its thoughts aren't directly supervised.

Note: There is a wide array of different prompt engineering techniques meant to serve similar purposes to hidden reasoning. Apart from hidden reasoning, chain-of-thought (CoT) reasoning (Wei et al., 2022) is popular.

The second advanced reasoning technique is to fine-tune the model using reinforcement learning, which we discuss conceptually. A common way to implement this is by using a Markov Decision Process (MDP) (Puterman, 1990), where each token represents an action and the model state is its partial (current) output. The reward can be programmatically assigned, but reinforcement learning with human feedback (RLHF) is most common.

Reinforcement learning samples outputs (typically using a categorical distribution) until the entire response is generated. It then calculates a scalar reward to evaluate the response. Next, advantages are calculated for the action taken at each time step (using a learned value function) to score the performance of each output. Tokens with more (or less) advantage are made more (or less) likely by parameter updates.

In order to update the parameters, the gradients of the (log) probabilities of each chosen action in the entire response with respect to each parameter are calculated (via backpropagation) and averaged per parameter. The parameters are typically updated using a proximal policy optimization (PPO) algorithm (Schulman et al., 2017), which increases the log-probability of chosen tokens proportional to their advantage.

Described above is how reinforcement learning is used to critique every action, but other implementations (such as the REINFORCE algorithm (Williams, 1992)) grades the entire response only once by calculating a total advantage for the entire response and broadcasting it to all time steps.

Note: Since parameters are updated after the entire response is finished, reinforcement learning is typically used to fine-tune the model after supervised learning. Using reinforcement learning from the beginning of training is computationally inefficient, especially if using RLHF.

The true power behind reinforcement learning, especially when coupled with hidden reasoning, is the ability to "move past" the limited 'predict the next token in a given text' technique. The bottom line of traditional supervised learning is to maximize corpus likelihood by predicting the next token. This concept can be molded to make language models intelligent by tweaking the training data. However, the goal of reinforcement learning, especially RLHF, is to generate a more favorable response that will result in a higher reward. This is more directly aligned with the main goal of making language models smarter.

3 Operation Steps

In this section, we cover the forward-pass for inference formally, breaking it down into operation steps. Steps $2-10$ detail the operations within a single transformer block $l \in \{1, \ldots, L\}$, implicitly using its corresponding parameters, and steps 3 and 4 outline the computations for one such head inside of that block. The input to block l is $X^{(l-1)}$ (where $X^{(0)}$ is the initial embedding), and the output is $X^{(l)}$.

1. **Token and Positional Embeddings:**

$$X_i^{(0)} \in \mathbb{R}^{d_{\text{model}}} = E_{\text{token}}[x_i] + E_{\text{pos}}[i] \quad \text{for each } i \in \{1, \ldots, n\} \tag{1}$$

2. **First LayerNorm (applied row-wise):**

$$X_{\text{ln1}} = \text{LayerNorm}(X^{(l-1)}; \gamma_1, \beta_1) = \gamma_1 \odot \frac{X^{(l-1)} - \mu(X^{(l-1)})}{\sqrt{\sigma^2(X^{(l-1)}) + \varepsilon}} + \beta_1 \tag{2}$$

3. **Compute Q/K/V projections:**

$$Q = X_{\text{ln1}} W_Q + \beta_Q, \quad K = X_{\text{ln1}} W_K + \beta_K, \quad V = X_{\text{ln1}} W_V + \beta_V \tag{3}$$

4. **Scaled Dot-Product Attention (per head):**

$$\text{AttentionHead}_j(Q, K, V) = \text{softmax}\left(\frac{Q_j K_j^\top}{\sqrt{d_{\text{head}}}} + M\right) V_j \tag{4}$$

5. **Concatenate Heads:**

$$H = \text{Concat}(\text{AttentionHead}_1, \ldots, \text{AttentionHead}_h) \in \mathbb{R}^{n \times (h \cdot d_{\text{head}})} \tag{5}$$

6. **Output Projection:**

$$X_{\text{attn}} = H W_O + \beta_O \tag{6}$$

7. **First Residual Connection:**

$$X_{\text{add1}} = X^{(l-1)} + X_{\text{attn}} \tag{7}$$

8. **Second LayerNorm:**

$$X_{\text{ln2}} = \text{LayerNorm}(X_{\text{add1}}; \gamma_2, \beta_2) \tag{8}$$

9. **Feed-Forward Network:**

$$X_{\text{ffn}} = \left(\text{GELU}\left(X_{\text{ln2}} W_1 + \beta_1^{\text{ffn}}\right)\right) W_2 + \beta_2^{\text{ffn}} \tag{9}$$

10. **Second Residual Connection:**

$$X^{(l)} = X_{\text{add1}} + X_{\text{ffn}} \tag{10}$$

- Repeat steps 2–10 for each block $l = 1, \ldots, L$.

11. **Final LayerNorm:**

$$X_{\text{final\_ln}} = \text{LayerNorm}(X^{(L)}; \gamma_{\text{final}}, \beta_{\text{final}}) \tag{11}$$

12. **Final Linear Projection:**

$$L = X_{\text{final\_ln}} W_{\text{lm}} + \beta_{\text{lm}} \in \mathbb{R}^{n \times V} \tag{12}$$

13. **Softmax (per token):**

$$P_i = \text{softmax}\left(\frac{L_i}{T}\right) \quad \text{for each } i \in \{1, \ldots, n\} \tag{13}$$

14. **Predict Next Token:**

$$\text{return} \quad x_{n+1} \sim \text{Categorical}(\text{Top}{-}p(P_n)) \tag{14}$$

4 Backpropagation through Final LM Head

To train a GPT model, we must calculate the gradient of all trainable parameters with respect to the loss $\mathcal{L}$. The remainder of this paper derives these gradients. We begin at the output layer and calculate the gradients of parameters as we flow backward through the model. To maintain readability, most derivations include the relevant forward equation(s), the incoming gradient from the previous layer, and our goal. Additionally, the end of each section contains a summary of derived gradients.

4.1 From Loss to Logits

Forward Equation: $\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n} \log P_i[y_i]$, where $P_i = \text{softmax}(L_i)$ (Eq. 13).

Goal: Calculate ∇_L, the gradient of the loss with respect to the logits L.

The backward pass starts from the scalar loss $\mathcal{L}$. We will find ∇_{L_i} for each row L_i. Since $\log P_i[y_i]$ only depends on L_i, we treat all $j \neq i$ terms as constants. Thus, for a specific element $L_i[v]$ (the logit for vocabulary term v at sequence position i):

$$\frac{\partial \mathcal{L}}{\partial L_i[v]} = -\frac{1}{n}\frac{\partial \log P_i[y_i]}{\partial L_i[v]}.$$

Using the chain rule for the logarithm:

$$\frac{\partial \log P_i[y_i]}{\partial L_i[v]} = \frac{1}{P_i[y_i]}\frac{\partial P_i[y_i]}{\partial L_i[v]}.$$

Recall the softmax definition:

$$P_i[y_i] = \frac{e^{L_i[y_i]}}{\sum_{u=1}^{V} e^{L_i[u]}}.$$

We evaluate the derivative $\frac{\partial P_i[y_i]}{\partial L_i[v]}$ using the quotient rule, considering two cases:

4.1.1 Case 1 ($v = y_i$):

The derivative is w.r.t. the logit corresponding to the correct token.

$$\frac{\partial P_i[y_i]}{\partial L_i[y_i]} = \frac{\frac{\partial (e^{L_i[y_i]})}{\partial L_i[y_i]}(\sum_u e^{L_i[u]}) - e^{L_i[y_i]}\frac{\partial (\sum_u e^{L_i[u]})}{\partial L_i[y_i]}}{(\sum_u e^{L_i[u]})^2}$$

$$= \frac{e^{L_i[y_i]}(\sum_u e^{L_i[u]}) - e^{L_i[y_i]}e^{L_i[y_i]}}{(\sum_u e^{L_i[u]})^2}$$

$$= \frac{e^{L_i[y_i]}}{\sum_u e^{L_i[u]}} \cdot \frac{(\sum_u e^{L_i[u]}) - e^{L_i[y_i]}}{\sum_u e^{L_i[u]}} = P_i[y_i]\left(1 - \frac{e^{L_i[y_i]}}{\sum_u e^{L_i[u]}}\right)$$

$$= P_i[y_i](1 - P_i[y_i]).$$

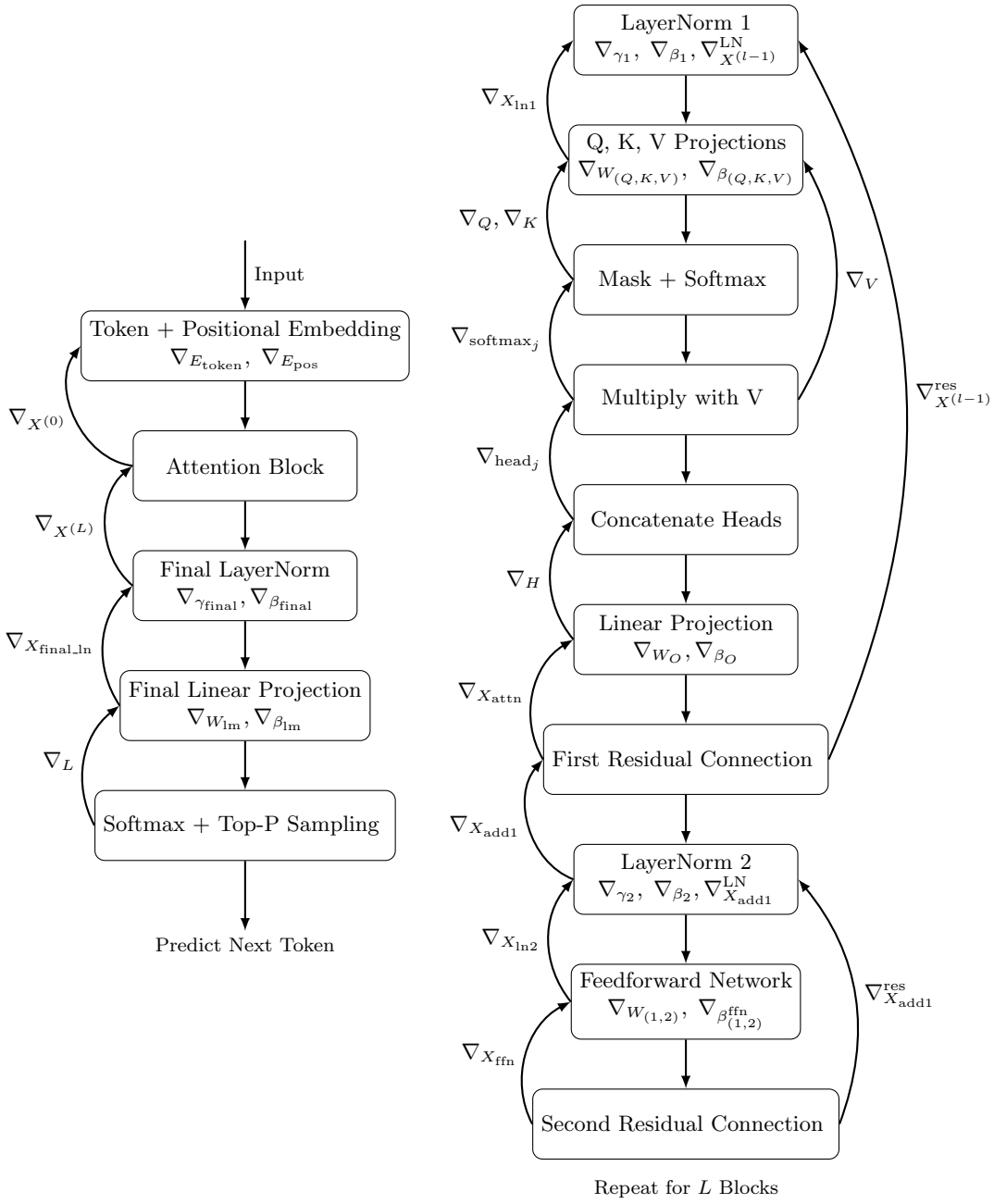

Figure 1: Overview of the GPT transformer architecture. (Left) High-level structure of the autoregressive transformer; (Right) internal breakdown of the attention mechanism, this fits into the attention block on the left. The arrows pointing downward signify the forward pass while the arrows pointing upward signify the backward propagation of gradients. The gradients listed in each block represent the gradients that are calculated for that block but aren't directly sent to previous blocks.

4.1.2 Case 2 ($v \neq y_i$):

The derivative is w.r.t. a logit corresponding to an incorrect token.

$$\frac{\partial P_i[y_i]}{\partial L_i[v]} = \frac{\frac{\partial (e^{L_i[y_i]})}{\partial L_i[v]}(\sum_u e^{L_i[u]}) - e^{L_i[y_i]}\frac{\partial (\sum_u e^{L_i[u]})}{\partial L_i[v]}}{(\sum_u e^{L_i[u]})^2}.$$

Since $v \neq y_i$, the first term's derivative is 0.

$$= \frac{0 - e^{L_i[y_i]}e^{L_i[v]}}{(\sum_u e^{L_i[u]})^2} = -\frac{e^{L_i[y_i]}}{\sum_u e^{L_i[u]}} \cdot \frac{e^{L_i[v]}}{\sum_u e^{L_i[u]}}$$

$$= -P_i[y_i]P_i[v].$$

Substituting these back into the expression for $\frac{\partial \log P_i[y_i]}{\partial L_i[v]}$:

$$\frac{\partial \log P_i[y_i]}{\partial L_i[v]} = \frac{1}{P_i[y_i]} \times \begin{cases} P_i[y_i](1 - P_i[y_i]), & \text{if } v = y_i \\ -P_i[y_i]P_i[v], & \text{if } v \neq y_i \end{cases} = \begin{cases} 1 - P_i[y_i], & \text{if } v = y_i \\ -P_i[v], & \text{if } v \neq y_i \end{cases}$$

$$\Rightarrow \frac{\partial \log P_i[y_i]}{\partial L_i[v]} = \mathbf{1}_{v=y_i} - P_i[v].$$

Substituting this into the gradient of the loss $\mathcal{L}$:

$$\frac{\partial \mathcal{L}}{\partial L_i[v]} = -\frac{1}{n}(\mathbf{1}_{v=y_i} - P_i[v]) = \frac{1}{n}(P_i[v] - \mathbf{1}_{v=y_i}).$$

For the entire matrix L, this means the gradient ∇_L has entries $(\nabla_L)_{iv} = \frac{1}{n}(P_{iv} - \mathbf{1}_{v=y_i})$.

$$\nabla_L = \frac{1}{n}(P - Y) \tag{15}$$

(Where Y is a matrix of the same shape as P, with $Y_{iv} = 1$ if v is the correct token for position i, and 0 otherwise).

4.2 Final Linear Projection

Forward Equation: $L = X_{\text{final\_ln}}W_{\text{lm}} + \beta_{\text{lm}}$ (Eq. 12).

Incoming Gradient: ∇_L.

Goal: Calculate $\nabla_{W_{\text{lm}}}$, $\nabla_{\beta_{\text{lm}}}$, and the gradient flowing back to the previous layer, $\nabla_{X_{\text{final\_ln}}}$.

By the chain rule,

$$\frac{\partial \mathcal{L}}{\partial X_{\text{final\_ln}}} = \frac{\partial \mathcal{L}}{\partial L}\frac{\partial L}{\partial X_{\text{final\_ln}}}, \quad \frac{\partial \mathcal{L}}{\partial W_{\text{lm}}} = \frac{\partial \mathcal{L}}{\partial L}\frac{\partial L}{\partial W_{\text{lm}}}, \quad \frac{\partial \mathcal{L}}{\partial \beta_{\text{lm}}} = \frac{\partial \mathcal{L}}{\partial L}\frac{\partial L}{\partial \beta_{\text{lm}}}.$$

Using the Vector-Jacobian product rule for matrix multiplication $Y = XW \implies \nabla_W = X^\top \nabla_Y$:

$$\nabla_{W_{\text{lm}}} = X_{\text{final\_ln}}^\top \nabla_L, \quad \nabla_{X_{\text{final\_ln}}} = \nabla_L W_{\text{lm}}^\top. \tag{16}$$

Since β_{lm} is added row-wise, $\nabla_{\beta_{\text{lm}}}$ sums the incoming gradients ∇_L across rows:

$$\nabla_{\beta_{\text{lm}}} = \sum_{i=1}^{n} (\nabla_L)_i \tag{17}$$

4.3 Final LayerNorm

Forward Equation: $X_{\text{final\_ln}} = \text{LayerNorm}(X^{(L)}; \gamma_{\text{final}}, \beta_{\text{final}})$ (Eq. 11).

Incoming Gradient: $\nabla_{X_{\text{final\_ln}}}$.

Goal: Calculate gradients for $\nabla_{\gamma_{\text{final}}}, \nabla_{\beta_{\text{final}}}$, and the gradient flowing back to the second residual connection, $\nabla_{X^{(L)}}$.

We derive the LayerNorm gradients in Section 5.3. The derivation is not repeated here for brevity.

$$\nabla_{\gamma_{\text{final}}} = \sum_{i=1}^{n} \nabla_{\tilde{x}_i} \odot \hat{x}_i, \quad \nabla_{\beta_{\text{final}}} = \sum_{i=1}^{n} \nabla_{\tilde{x}_i} \tag{18}$$

$$(\nabla_{X^{(L)}})_i = \frac{1}{\sqrt{\sigma_i^2 + \varepsilon}} \Big[g - \tfrac{1}{d_{\text{model}}} \big(\sum_{j=1}^{d_{\text{model}}} g_j \big) \mathbf{1} - \hat{x} \odot \Big(\tfrac{1}{d_{\text{model}}} \sum_{j=1}^{d_{\text{model}}} g_j \, \hat{x}_j \Big) \mathbf{1} \Big], \tag{19}$$

applied row-wise for $i = 1, \ldots, n$, where:

$$\tilde{x} = (X_{\text{final\_ln}})_i, \quad g = (\nabla_{X_{\text{final\_ln}}})_i \odot \gamma_{\text{final}}, \quad x = (X^{(L)})_i, \text{ and } \hat{x} = \frac{(x - \mu_i)}{\sqrt{\sigma_i^2 + \varepsilon}}.$$

4.4 Gradients Summary (Final LM Head)

4.4.1 From Loss to Logits:

$$\nabla_L = \frac{1}{n}(P - Y)$$

4.4.2 Final Linear Projection:

$$\nabla_{W_{\text{lm}}} = X_{\text{final\_ln}}^{\top} \nabla_L, \quad \nabla_{\beta_{\text{lm}}} = \sum_{i=1}^{n} (\nabla_L)_i, \quad \nabla_{X_{\text{final\_ln}}} = \nabla_L W_{\text{lm}}^{\top}$$

4.4.3 Final LayerNorm:

$$\nabla_{\gamma_{\text{final}}} = \sum_{i=1}^{n} \nabla_{\tilde{x}_i} \odot \hat{x}_i, \quad \nabla_{\beta_{\text{final}}} = \sum_{i=1}^{n} \nabla_{\tilde{x}_i}$$

$$(\nabla_{X^{(L)}})_i = \frac{1}{\sqrt{\sigma_i^2 + \varepsilon}} \Big[g - \tfrac{1}{d_{\text{model}}} \big(\sum_{j=1}^{d_{\text{model}}} g_j \big) \mathbf{1} - \hat{x} \odot \Big(\tfrac{1}{d_{\text{model}}} \sum_{j=1}^{d_{\text{model}}} g_j \, \hat{x}_j \Big) \mathbf{1} \Big]$$

5 Backprop through Second Residual and FFN

5.1 Second Residual Connection

Forward Equation: $X^{(l)} = X_{\text{add1}} + X_{\text{ffn}}$ (Eq. 10).

Incoming Gradient: $\nabla_{X^{(l+1)}}$ (from the block above, or from the final LayerNorm if this is the last block).

Goal: Calculate $\nabla^{\text{res}}_{X_{\text{add1}}}$ (gradient component for X_{add1} from this path) and $\nabla_{X_{\text{ffn}}}$.

Since the partial derivatives are identity matrices with respect to $\nabla_{X^{(l)}}$:

$$\nabla_{X_{\text{ffn}}} = \nabla_{X^{(l)}}, \qquad \nabla^{\text{res}}_{X_{\text{add1}}} = \nabla_{X^{(l)}}. \tag{20}$$

Note: $\nabla_{X_{\text{add1}}}$ flows back here as $\nabla^{\text{res}}_{X_{\text{add1}}}$, but it also flows back to LayerNorm 2 in section 5.3, defined as $\nabla^{\text{LN}}_{X_{\text{add1}}}$. The total gradient $\nabla_{X_{\text{add1}}}$ is the sum of the two components, calculated in section 5.4.

5.2 Feed-Forward Network

5.2.1 From X_{ffn} to $H, W_2, \beta_2^{\text{ffn}}$

Forward Equation: $X_{\text{ffn}} = H\,W_2 + \beta_2^{\text{ffn}}$ (Second part of Eq. 9).

Incoming Gradient: $\nabla_{X_{\text{ffn}}}$.

Goal: Calculate $\nabla_{W_2}, \nabla_{\beta_2^{\text{ffn}}}$, and the gradient flowing back to the GELU output, ∇_H.

This is another standard affine transformation:

$$\nabla_{W_2} = H^\top \nabla_{X_{\text{ffn}}}, \quad \nabla_{\beta_2^{\text{ffn}}} = \sum_{i=1}^{n} (\nabla_{X_{\text{ffn}}})_i, \quad \nabla_H = \nabla_{X_{\text{ffn}}} W_2^\top. \tag{21}$$

5.2.2 GELU Activation

Forward Equation: $H = \text{GELU}(A)$ (Intermediate part of Eq. 9).

Incoming Gradient: ∇_H.

Goal: Calculate ∇_A, the gradient flowing back to the input of the GELU function.

Since GELU is an elementwise function, $H_{ij} = \text{GELU}(A_{ij})$, the chain rule applies elementwise:

$$\frac{\partial \mathcal{L}}{\partial A_{ij}} = \frac{\partial \mathcal{L}}{\partial H_{ij}} \frac{\partial H_{ij}}{\partial A_{ij}} = (\nabla_H)_{ij} \cdot \text{GELU}'(A_{ij}).$$

In matrix form:

$$\nabla_A = \nabla_H \odot \text{GELU}'(A). \tag{22}$$

Note: As explained in section 2.3, the GELU is typically approximated in practice. We do not define the derivative here explicitly since several different approximations exist.

5.2.3 To $W_1, \beta_1^{\text{ffn}}, X_{\text{ln2}}$

Forward Equation: $A = X_{\text{ln2}} W_1 + \beta_1^{\text{ffn}}$ (First part of Eq. 9).

Incoming Gradient: ∇_A.

Goal: Calculate $\nabla_{W_1}, \nabla_{\beta_1^{\text{ffn}}}$, and the gradient flowing back to the output of the second LayerNorm, $\nabla_{X_{\text{ln2}}}$.

This is again a standard affine transformation:

$$\nabla_{W_1} = X_{\text{ln2}}^\top \nabla_A, \quad \nabla_{\beta_1^{\text{ffn}}} = \sum_{i=1}^{n} (\nabla_A)_i, \quad \nabla_{X_{\text{ln2}}} = \nabla_A \, W_1^\top. \tag{23}$$

5.3 Second LayerNorm

Forward Equation: $X_{\text{ln2}} = \text{LayerNorm}(X_{\text{add1}}; \gamma_2, \beta_2)$ (Eq. 8).

Incoming Gradient: $\nabla_{X_{\text{ln2}}}$.

Goal: Calculate gradients for parameters $\nabla_{\gamma_2}, \nabla_{\beta_2}$, and the gradient component flowing back to the first residual connection $\nabla^{\text{LN}}_{X_{\text{add1}}}$.

5.3.1 Gradients w.r.t. Scale (γ_2) and Shift (β_2) Parameters

The LayerNorm operation for a single row i is:
$X_{\text{ln2,i}} = \hat{X}_{\text{add1,i}} \odot \gamma_2 + \beta_2$, where $\hat{X}_{\text{add1,i}} = \frac{X_{\text{add1,i}} - \mu_i}{\sqrt{\sigma_i^2 + \varepsilon}}$.

Using the chain rule, summing contributions across all rows $i = 1...n$:

$$\nabla_{\beta_2} = \sum_{i=1}^{n} \frac{\partial \mathcal{L}}{\partial X_{\text{ln2,i}}} \frac{\partial X_{\text{ln2,i}}}{\partial \beta_2} = \sum_{i=1}^{n} (\nabla_{X_{\text{ln2}}})_i \odot 1$$

$$\nabla_{\gamma_2} = \sum_{i=1}^{n} \frac{\partial \mathcal{L}}{\partial X_{\text{ln2,i}}} \frac{\partial X_{\text{ln2,i}}}{\partial \gamma_2} = \sum_{i=1}^{n} (\nabla_{X_{\text{ln2}}})_i \odot \hat{X}_{\text{add1,i}}$$

$$\nabla_{\gamma_2} = \sum_{i=1}^{n} (\nabla_{X_{\text{ln2}}})_i \odot \frac{X_{\text{add1,i}} - \mu_i}{\sqrt{\sigma_i^2 + \varepsilon}}, \quad \nabla_{\beta_2} = \sum_{i=1}^{n} (\nabla_{X_{\text{ln2}}})_i. \tag{24}$$

5.3.2 Gradient w.r.t. LayerNorm Input X_{add1}

Goal: Calculate $\nabla^{\text{LN}}_{X_{\text{add1}}} := \frac{\partial \mathcal{L}}{\partial X_{\text{add1}}}\Big|_{\text{through LayerNorm}}$. This will be done by decomposing the derivative into three parts and solving for each one separately:

$$\frac{\partial \mathcal{L}}{\partial \hat{x}_j} = \frac{\partial \mathcal{L}}{\partial \tilde{x}_j} \frac{\partial \tilde{x}_j}{\partial \hat{x}_j} \frac{\partial \hat{x}_j}{\partial x_j}.$$

We now calculate the gradient for a single row r: Let $x = (X_{\text{add1}})_r, \tilde{x} = (X_{\text{ln2}})_r$, and the incoming gradient row be $\nabla_{\tilde{x}} = (\nabla_{X_{\text{ln2}}})_r$.

5.3.2.1 Gradient w.r.t. Normalized Input $\hat{x}$:

By the chain rule, since $\tilde{x}_j = \gamma_{2,j}\,\hat{x}_j + \beta_{2,j}$,

$$\frac{\partial \mathcal{L}}{\partial \hat{x}_j} = \frac{\partial \mathcal{L}}{\partial \tilde{x}_j}\frac{\partial \tilde{x}_j}{\partial \hat{x}_j} = \nabla_{\tilde{x}_j}\,\gamma_{2,j}.$$

Let $g_j = \nabla_{\tilde{x}_j}\gamma_{2,j}$.

5.3.2.2 LayerNorm Gradient $\frac{\partial \hat{x}_j}{\partial x_i}$

Forward equations:

$$\hat{x}_j = \frac{x_j - \mu}{\sqrt{\sigma^2 + \varepsilon}},$$

$$\mu = \frac{1}{d_{\text{model}}}\sum_{k=1}^{d_{\text{model}}} x_k, \qquad \sigma^2 = \frac{1}{d_{\text{model}}}\sum_{k=1}^{d_{\text{model}}} (x_k - \mu)^2.$$

We set $d = d_{\text{model}}$ to keep notation light. Define $\hat{x}_j = \frac{a}{b}$. By the quotient rule:

$$\frac{\partial \hat{x}_j}{\partial x_i} = \frac{b\,\partial_i a \;-\; a\,\partial_i b}{b^2}.$$

We need to find $\frac{\partial a}{\partial x_i}$ and $\frac{\partial b}{\partial x_i}$.
Since $a = x_j - \mu$ and $\frac{\partial \mu}{\partial x_i} = \frac{1}{d}$,

$$\frac{\partial a}{\partial x_i} = \delta_{ij} - \frac{1}{d}.$$

b $= \sqrt{\sigma^2 + \varepsilon}$. To compute $\frac{\partial b}{\partial x_i}$, first note:

$$\frac{\partial \sigma^2}{\partial x_i} = \frac{\partial}{\partial x_i}\Big[\tfrac{1}{d}\sum_k (x_k - \mu)^2\Big] = \frac{1}{d}\sum_k 2(x_k - \mu)\left(\delta_{ik} - \frac{1}{d}\right) = \frac{2}{d}(x_i - \mu).$$

Then:

$$\frac{\partial b}{\partial x_i} = \frac{\partial}{\partial x_i}(\sigma^2 + \varepsilon)^{1/2} = \frac{1}{2\,b}\,\frac{\partial \sigma^2}{\partial x_i} = \frac{x_i - \mu}{d \cdot b}.$$

Plugging in to the quotient rule:

$$\frac{\partial \hat{x}_j}{\partial x_i} = \frac{b\,(\delta_{ij} - \frac{1}{d}) \;-\; (x_j - \mu)\,\frac{x_i - \mu}{d \cdot b}}{b^2}.$$

Since:

$$\hat{x}_j = \frac{x_j - \mu}{b} \quad \Rightarrow \quad x_j - \mu = b\,\hat{x}_j,$$

$$(x_j - \mu)(x_i - \mu) = (b\,\hat{x}_j)(b\,\hat{x}_i) = b^2 \hat{x}_j \hat{x}_i.$$

The second term in the numerator simplifies to:

$$\frac{(x_j - \mu)(x_i - \mu)}{d \cdot b} = \frac{b^2 \hat{x}_j \hat{x}_i}{d \cdot b} = \frac{b \hat{x}_j \hat{x}_i}{d}.$$

The numerator now becomes:

$$b\left(\delta_{ij} - \frac{1}{d}\right) - \frac{b\hat{x}_j\hat{x}_i}{d} = b\left[\delta_{ij} - \frac{1}{d} - \frac{\hat{x}_j\hat{x}_i}{d}\right].$$

Combining with the denominator:

$$\frac{\partial \hat{x}_j}{\partial x_i} = \frac{1}{\sqrt{\sigma^2+\varepsilon}}\Big[\delta_{ij} - \tfrac{1}{d_{\text{model}}} - \frac{\hat{x}_j\hat{x}_i}{d_{\text{model}}}\Big].$$

5.3.2.3 Gradient w.r.t. Input x_i (for the row):

$$\nabla_{x_i} = \frac{\partial \mathcal{L}}{\partial x_i} = \sum_{j=1}^{d_{\text{model}}} \frac{\partial \mathcal{L}}{\partial \hat{x}_j}\frac{\partial \hat{x}_j}{\partial x_i} = \sum_{j=1}^{d_{\text{model}}} g_j \frac{\partial \hat{x}_j}{\partial x_i}$$

Substituting the Jacobian:

$$\nabla_{x_i} = \sum_{j=1}^{d_{\text{model}}} g_j \frac{1}{\sqrt{\sigma^2+\varepsilon}}\Big[\delta_{ij} - \frac{1}{d_{\text{model}}} - \frac{\hat{x}_j\hat{x}_i}{d_{\text{model}}}\Big]$$

$$= \frac{1}{\sqrt{\sigma^2+\varepsilon}}\left[g_i - \frac{1}{d_{\text{model}}}\sum_j g_j - \hat{x}_i \cdot \frac{1}{d_{\text{model}}}\sum_j g_j\hat{x}_j\right].$$

Writing this result in matrix form:

$$(\nabla^{\text{LN}}_{X_{\text{add1}}})_i = \frac{1}{\sqrt{\sigma^2+\varepsilon}}\Big[g - \tfrac{1}{d_{\text{model}}}\Big(\sum_{j=1}^{d_{\text{model}}} g_j\Big)\mathbf{1} - \hat{x}\odot\Big(\tfrac{1}{d_{\text{model}}}\sum_{j=1}^{d_{\text{model}}} g_j\,\hat{x}_j\Big)\mathbf{1}\Big],$$

applied row-wise for $i = 1, \ldots, n$, where:

$$\tilde{x} = (X_{\text{ln2}})_r, \quad g = \nabla_{\tilde{x}} \odot \gamma_2, \quad x = (X_{\text{add1}})_r, \quad \text{and} \quad \hat{x} = \frac{x-\mu}{\sqrt{\sigma^2+\varepsilon}}.$$

5.4 Final Gradient Accumulation for X_{add1}

The total gradient for X_{add1} is the sum of the component from the residual connection (Section 5.1) and the component from LayerNorm 2 (Section 5.3.2):

$$\nabla_{X_{\text{add1}}} = \nabla^{\text{res}}_{X_{\text{add1}}} + \nabla^{\text{LN}}_{X_{\text{add1}}}. \tag{25}$$

5.5 Gradients Summary (Second Residual and FFN Gradients)

5.5.1 Output Gradients

$$\nabla_{X_{\text{ffn}}} = \nabla_{X^{(l)}}, \quad \nabla^{\text{res}}_{X_{\text{add1}}} = \nabla_{X^{(l)}}$$

5.5.2 FFN Layer 2 (Linear → Output)

$$\nabla_{W_2} = H^\top \nabla_{X_{\text{ffn}}}, \quad \nabla_{\beta_2^{\text{ffn}}} = \sum_{i=1}^{n} (\nabla_{X_{\text{ffn}}})_i, \quad \nabla_H = \nabla_{X_{\text{ffn}}} W_2^\top$$

5.5.3 GELU Nonlinearity

$$\nabla_A = \nabla_H \odot \text{GELU}'(A)$$

5.5.4 FFN Layer 1 (Input → Hidden)

$$\nabla_{W_1} = X_{\text{ln2}}^\top \nabla_A, \quad \nabla_{\beta_1^{\text{ffn}}} = \sum_{i=1}^{n} (\nabla_A)_i, \quad \nabla_{X_{\text{ln2}}} = \nabla_A \, W_1^\top$$

5.5.5 LayerNorm Gradients

$$\nabla_{\gamma_2} = \sum_{i=1}^{n} (\nabla_{X_{\text{ln2}}})_i \odot \frac{X_{\text{add1},i} - \mu_i}{\sqrt{\sigma_i^2 + \varepsilon}}, \quad \nabla_{\beta_2} = \sum_{i=1}^{n} (\nabla_{X_{\text{ln2}}})_i$$

$$(\nabla_{X_{\text{add1}}}^{\text{LN}})_i = \frac{1}{\sqrt{\sigma^2 + \varepsilon}} \Big[g - \tfrac{1}{d_{\text{model}}} \Big(\sum_{j=1}^{d_{\text{model}}} g_j \Big) \mathbf{1} - \hat{x} \odot \Big(\tfrac{1}{d_{\text{model}}} \sum_{j=1}^{d_{\text{model}}} g_j \, \hat{x}_j \Big) \mathbf{1} \Big]$$

5.5.6 Final Residual Combination

$$\nabla_{X_{\text{add1}}} = \nabla_{X_{\text{add1}}}^{\text{res}} + \nabla_{X_{\text{add1}}}^{\text{LN}}$$

6 Backprop through First Residual and MHA

6.1 First Residual Connection

Forward Equation: $X_{\text{add1}} = X^{(l-1)} + X_{\text{attn}}$ (Eq. 7).

Incoming Gradient: $\nabla_{X_{\text{add1}}}$ (from Section 5.4).

Goal: Calculate $\nabla_{X_{\text{attn}}}$ (passed to the attention block output) and $\nabla_{X^{(0)}}^{\text{res}}$ (the gradient component for the initial embeddings $X^{(0)}$).

Since the partial derivatives are identity matrices with respect to $\nabla_{X_{\text{add1}}}$:

$$\nabla_{X_{\text{attn}}} = \nabla_{X_{\text{add1}}}, \quad \nabla_{X^{(0)}}^{\text{res}} = \nabla_{X_{\text{add1}}}. \tag{26}$$

Note: $\nabla_{X^{(0)}}^{\text{res}}$ will be combined with the gradient component coming through LayerNorm 1 in Section 6.7.

6.2 Attention Output Projection

Forward Equation: $X_{\text{attn}} = HW_O + \beta_O$ (Eq. 6).

Incoming Gradient: $\nabla_{X_{\text{attn}}}$.

Goal: Calculate gradients for $\nabla_{W_O}, \nabla_{\beta_O}$, and the gradient flowing back to the concatenated attention heads ∇_H.

This is a standard affine transformation:

$$\nabla_{W_O} = H^\top \nabla_{X_{\text{attn}}}, \quad \nabla_{\beta_O} = \sum_{i=1}^{n} (\nabla_{X_{\text{attn}}})_i, \quad \nabla_H = \nabla_{X_{\text{attn}}} W_O^\top. \tag{27}$$

6.3 Concatenation

Forward Equation: $H = \text{Concat}(\text{AttentionHead}_1, \ldots, \text{AttentionHead}_h)$ (Eq. 5).

Incoming Gradient: ∇_H.

Goal: Distribute the gradient ∇_H back to the individual attention heads, calculating $\nabla_{\text{AttentionHead}_j}$ for each head j.

The backward pass through concatenation involves slicing the incoming gradient matrix ∇_H column-wise according to the dimensions of each head.

$$\nabla_{\text{AttentionHead}_j} = \text{slice}_j\big(\nabla_H\big) \text{ for each head } j \in \{1, \ldots, h\} \tag{28}$$

(Where slice_j extracts the columns of ∇_H corresponding to the output dimensions of AttentionHead_j.)

6.4 Scaled-Dot-Product Attention (per head)

The derivation below is for an individual head j, implicitly using its notation. The gradient is calculated for each head. Forward equations (Eq. 4):

$$S = \frac{QK^\top}{\sqrt{d_{\text{head}}}} + M, \quad A = \text{softmax}(S), \quad \text{AttentionHead} = AV.$$

6.4.1 From Attention Head Output to Attention Weights and Value

Forward Equation: $\text{AttentionHead} = AV$.

Incoming Gradient: $\nabla_{\text{AttentionHead}}$.

Calculate ∇_A and ∇_V.

This is a standard matrix multiplication:

$$\nabla_A = \nabla_{\text{AttentionHead}} V^\top, \quad \nabla_V = A^\top \nabla_{\text{AttentionHead}}. \tag{29}$$

6.4.2 Through Softmax

Forward Equation: $A = \text{softmax}(S)$ (applied row-wise).

Incoming Gradient: ∇_A.

Goal: Calculate ∇_S.

We need the Jacobian of the softmax function applied row-wise. For a single row vector $a = \text{softmax}(s)$, the Jacobian matrix J has elements $J_{ik} = \frac{\partial a_i}{\partial s_k}$. We derived earlier (Section 4.1, applied here per row):
$J_{ik} = a_i(\delta_{ik} - a_k)$. This can be written as $J = \text{diag}(a) - aa^\top$.
The gradient $\nabla_s = (\frac{\partial a}{\partial s})^\top \nabla_a = J^\top \nabla_a = J\nabla_a$ (since J is symmetric).

$$\nabla_{s_i} = \sum_k J_{ik}\nabla_{a_k} = \sum_k a_i(\delta_{ik} - a_k)\nabla a_k = a_i\nabla_{a_i} - a_i\sum_k a_k\nabla_{a_k} = a_i(\nabla a_i - \sum_k a_k \nabla a_k).$$

Applying this row-wise to the matrices A and S, with incoming gradient ∇_A:

$$\nabla_S = (\nabla_A \odot A) - A \odot \text{rowsum}(\nabla_A \odot A). \tag{30}$$

Note: $\text{rowsum}(M) \in \mathbb{R}^{n\times 1}$ computes the sum of each row of M.

6.4.3 From Scores to Query and Key

Forward Equation: $S = \frac{QK^\top}{\sqrt{d_{\text{head}}}} + M$.

Incoming Gradient: ∇_S.

Goal: Calculate ∇_Q and ∇_K.

This is another matrix multiplication:

$$\nabla_Q = \frac{1}{\sqrt{d_{\text{head}}}}\nabla_S\, K, \quad \nabla_K = \frac{1}{\sqrt{d_{\text{head}}}}(\nabla_S)^\top Q. \tag{31}$$

Note: The mask automatically affects the gradient by modifying the softmax output in the forward pass.

6.5 QKV Projections

Forward Equations:
$Q = X_{\text{ln1}}W_Q + \beta_Q, \quad K = X_{\text{ln1}}W_K + \beta_K, \quad V = X_{\text{ln1}}W_V + \beta_V$ (Eq. 3).

Incoming Gradients: $\nabla_{Q_j}, \nabla_{K_j}, \nabla_{V_j}$ for each head j.

Goal: Calculate gradients $\nabla_{W_Q}, \nabla_{\beta_Q}, \ldots, \nabla_{W_V}, \nabla_{\beta_V}$, and the gradient flowing back to the output of the first LayerNorm, $\nabla_{X_{\text{ln1}}}$.

First, reconstruct the full gradients for the combined Q, K, V matrices by concatenating the gradients from individual heads along the feature dimension:

$$\nabla_Q = \text{Concat}(\nabla_{Q_1}, \ldots, \nabla_{Q_h}), \quad \nabla_K = \text{Concat}(\nabla_{K_1}, \ldots, \nabla_{K_h}), \quad \nabla_V = \text{Concat}(\nabla_{V_1}, \ldots, \nabla_{V_h}).$$

Since Q, K and V are calculated via affine transformations:

$$\nabla_{W_Q} = X_{\text{ln1}}^\top \nabla_Q, \quad \nabla_{W_K} = X_{\text{ln1}}^\top \nabla_K, \quad \nabla_{W_V} = X_{\text{ln1}}^\top \nabla_V \tag{32}$$

$$\nabla_{\beta_Q} = \sum_{i=1}^{n} (\nabla_Q)_i, \quad \nabla_{\beta_K} = \sum_{i=1}^{n} (\nabla_K)_i, \quad \nabla_{\beta_V} = \sum_{i=1}^{n} (\nabla_V)_i. \tag{33}$$

Gradient w.r.t. Input X_{ln1}: The gradient contributions from the Q, K, and V paths are summed.

$$\nabla_{X_{\text{ln1}}} = \nabla_Q W_Q^\top + \nabla_K W_K^\top + \nabla_V W_V^\top. \tag{34}$$

6.6 First LayerNorm

Forward Equation: $X_{\text{ln1}} = \text{LayerNorm}(X^{(0)}; \gamma_1, \beta_1)$ (Eq. 2).

Incoming Gradient: $\nabla_{X_{\text{ln1}}}$.

Goal: Calculate gradients for parameters $\nabla_{\gamma_1}, \nabla_{\beta_1}$, and the gradient component flowing back to the initial embeddings $\nabla_{X^{(0)}}^{\text{LN}}$.

The derivation is identical in form to the second LayerNorm (Section 5.3).

$$\nabla_{\gamma_1} = \sum_{i=1}^{n} (\nabla_{X_{\text{ln1}}})_i \odot \frac{X_i^{(0)} - \mu_i}{\sqrt{\sigma_i^2 + \varepsilon}}, \quad \nabla_{\beta_1} = \sum_{i=1}^{n} (\nabla_{X_{\text{ln1}}})_i, \tag{35}$$

$$(\nabla_{X^{(0)}}^{\text{LN}})_i = \frac{1}{\sqrt{\sigma_i^2 + \varepsilon}} \Big[g - \tfrac{1}{d_{\text{model}}} \big(\sum_{j=1}^{d_{\text{model}}} g_j \big) \mathbf{1} - \hat{x} \odot \Big(\tfrac{1}{d_{\text{model}}} \sum_{j=1}^{d_{\text{model}}} g_j \hat{x}_j \Big) \mathbf{1} \Big], \tag{36}$$

applied row-wise for $i = 1, \ldots, n$, where:

$$\tilde{x} = (X_{\text{ln1}})_i, \quad g = (\nabla_{X_{\text{ln1}}})_i \odot \gamma_1 \quad x = (X^{(0)})_i, \text{ and } \hat{x} = \frac{x - \mu_i}{\sqrt{\sigma_i^2 + \varepsilon}}.$$

6.7 Final Gradient Accumulation for $X^{(0)}$

The total gradient for the initial embedding $X^{(0)}$ combines the gradient flowing directly from the first residual connection (Section 6.1) and the gradient flowing through the first LayerNorm (Section 6.6):

$$\nabla_{X^{(0)}} = \nabla_{X^{(0)}}^{\text{res}} + \nabla_{X^{(0)}}^{\text{LN}}. \tag{37}$$

6.8 Gradients Summary (First Residual and MHA)

6.8.1 First Residual Connection

$$\nabla_{X_{\text{attn}}} = \nabla_{X_{\text{add1}}}, \quad \nabla_{X^{(0)}}^{\text{res}} = \nabla_{X_{\text{add1}}}$$

6.8.2 Output Projection

$$\nabla_{W_O} = H^\top \nabla_{X_{\text{attn}}}, \quad \nabla_{\beta_O} = \sum_{i=1}^{n} (\nabla_{X_{\text{attn}}})_i, \quad \nabla_H = \nabla_{X_{\text{attn}}} W_O^\top$$

6.8.3 Attention Head Concatenation

$$\nabla_{\text{AttentionHead}_j} = \text{slice}_j(\nabla_H) \quad \text{for } j = 1, \ldots, h$$

6.8.4 Scaled-Dot-Product Attention (per head)

6.8.4.1 Output to Attention Weights and Value

$$\nabla_{A_j} = \nabla_{\text{AttentionHead}_j} V_j^\top, \quad \nabla_{V_j} = A_j^\top \nabla_{\text{AttentionHead}_j}$$

6.8.4.2 Softmax

$$\nabla_{S_j} = (\nabla_{A_j} \odot A_j) - A_j \odot \text{rowsum}(\nabla_{A_j} \odot A_j)$$

6.8.4.3 Attention Score to Query and Key

$$\nabla_{Q_j} = \frac{1}{\sqrt{d_{\text{head}}}} \nabla_{S_j} K_j, \quad \nabla_{K_j} = \frac{1}{\sqrt{d_{\text{head}}}} (\nabla_{S_j})^\top Q_j$$

6.8.5 QKV Projections (All Heads Combined)

Concatenate Per-Head Gradients

$$\nabla_Q = \text{Concat}(\nabla_{Q_1}, \ldots, \nabla_{Q_h}), \quad \nabla_K = \text{Concat}(\nabla_{K_1}, \ldots, \nabla_{K_h}), \quad \nabla_V = \text{Concat}(\nabla_{V_1}, \ldots, \nabla_{V_h})$$

Gradients w.r.t. QKV Parameters

$$\nabla_{W_Q} = X_{\text{ln1}}^\top \nabla_Q, \quad \nabla_{W_K} = X_{\text{ln1}}^\top \nabla_K, \quad \nabla_{W_V} = X_{\text{ln1}}^\top \nabla_V$$

$$\nabla_{\beta_Q} = \sum_{i=1}^{n} (\nabla_Q)_i, \quad \nabla_{\beta_K} = \sum_{i=1}^{n} (\nabla_K)_i, \quad \nabla_{\beta_V} = \sum_{i=1}^{n} (\nabla_V)_i$$

Gradient to Input of QKV Projections

$$\nabla_{X_{\text{ln1}}} = \nabla_Q W_Q^\top + \nabla_K W_K^\top + \nabla_V W_V^\top$$

6.8.6 First LayerNorm

$$\nabla_{\gamma_1} = \sum_{i=1}^{n} (\nabla_{X_{\text{ln1}}})_i \odot \frac{X_i^{(0)} - \mu_i}{\sqrt{\sigma_i^2 + \varepsilon}}, \quad \nabla_{\beta_1} = \sum_{i=1}^{n} (\nabla_{X_{\text{ln1}}})_i$$

$$(\nabla_{X^{(0)}}^{\text{LN}})_i = \frac{1}{\sqrt{\sigma_i^2 + \varepsilon}} \Big[g - \tfrac{1}{d_{\text{model}}} \big(\sum_{j=1}^{d_{\text{model}}} g_j \big) \mathbf{1} - \hat{x} \odot \Big(\tfrac{1}{d_{\text{model}}} \sum_{j=1}^{d_{\text{model}}} g_j \, \hat{x}_j \Big) \mathbf{1} \Big]$$

6.8.7 Final Gradient Accumulation

$$\nabla_{X^{(0)}} = \nabla^{\text{res}}_{X^{(0)}} + \nabla^{\text{LN}}_{X^{(0)}}$$

7 Backprop through Embeddings

Forward Equation: $X_i^{(0)} = E_{\text{token}}[x_i, :] + E_{\text{pos}}[i, :]$ for each row $i \in \{1, \ldots, n\}$ (Eq. 1).

Incoming Gradient: $\nabla_{X^{(0)}}$.

Goal: Calculate $\nabla_{E_{\text{token}}}$ and $\nabla_{E_{\text{pos}}}$.

For the positional embedding E_{pos}, the i-th row $E_{\text{pos}}[i, :]$ only contributes to $X_i^{(0)}$. Therefore:

$$\frac{\partial \mathcal{L}}{\partial E_{\text{pos}}[i, :]} = \frac{\partial \mathcal{L}}{\partial X_i^{(0)}} \frac{\partial X_i^{(0)}}{\partial E_{\text{pos}}[i, :]} = \nabla_{X_i^{(0)}} \cdot 1$$

$$\nabla_{E_{\text{pos}}}[i, :] = \nabla_{X_i^{(0)}}. \tag{38}$$

For E_{token}, the specific embedding vector used at position i is $E_{\text{token}}[x_i, :]$, where x_i is the token index at that position. If multiple tokens in the input are the same, their gradient gets accumulated:

$$\frac{\partial \mathcal{L}}{\partial E_{\text{token}}[x_i, :]} \mathrel{+}= \frac{\partial \mathcal{L}}{\partial X_i^{(0)}} \frac{\partial X_i^{(0)}}{\partial E_{\text{token}}[x_i, :]} = \nabla_{X_i^{(0)}} \cdot 1$$

$$\nabla_{E_{\text{token}}}[x_i, :] \mathrel{+}= \nabla_{X_i^{(0)}}. \tag{39}$$

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer mormalization. *arXiv preprint arXiv:1607.06450*, 2016.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 2020.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens. *arXiv preprint arXiv:2310.02226*, 2023.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Ofir Press and Lior Wolf. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*, 2016.

Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science*, 1990.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 1986.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 2014.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, 2017. URL `http://arxiv.org/abs/1706.03762`.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 1992.