Math Underlying Transformers

Promothesh Chatterjee*

Initial Setup

- **Sentence:** "The cat sat on the mat."
- **Embeddings:** Each word is represented by a 2-dimensional vector. Suppose we have 2-dimensional embeddings for simplicity:
  - "The" = [1, 0]
  - "cat" = [0, 1]
  - "sat" = [1, 1]
  - "on" = [0, 0]
  - "the" = [1, 0]
  - "mat" = [0, 1]
- **Positional Embeddings:** Representing positions as 2-dimensional vectors for simplicity:
  - Position 0 = [0, 0]
  - Position 1 = [0, 1]
  - Position 2 = [1, 0]
  - Position 3 = [1, 1]
  - Position 4 = [0, 2]
  - Position 5 = [2, 0]
- **Combined Embeddings:** Adding the positional embeddings to the word embeddings:
  - "The" at position 0 = [1, 0] + [0, 0] = [1, 0]
  - "cat" at position 1 = [0, 1] + [0, 1] = [0, 2]
  - "sat" at position 2 = [1, 1] + [1, 0] = [2, 1]
  - "on" at position 3 = [0, 0] + [1, 1] = [1, 1]
  - "the" at position 4 = [1, 0] + [0, 2] = [1, 2]
  - "mat" at position 5 = [0, 1] + [2, 0] = [2, 1]
- **Weight Matrices:** Initialized randomly.

Step-by-Step Training Process

1. **Forward Pass:**

   For each word, compute the Q, K, and V vectors:

$$Q_{word} = X_{word} \times W_Q$$
$$K_{word} = X_{word} \times W_K$$
$$V_{word} = X_{word} \times W_V$$

   For simplicity, let's focus on a single word, "cat".

   - Combined embedding for "cat": [0, 2]

- Initial W_Q:
$$\begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix}$$
- Initial W_K:
$$\begin{bmatrix} 0.5 & 0.6 \\ 0.7 & 0.8 \end{bmatrix}$$
- Initial W_V:
$$\begin{bmatrix} 0.9 & 1.0 \\ 1.1 & 1.2 \end{bmatrix}$$

Compute Q, K, and V for "cat":

$$Q_{cat} = [0, 2] \times \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix} = [0.6, 0.8]$$

$$K_{cat} = [0, 2] \times \begin{bmatrix} 0.5 & 0.6 \\ 0.7 & 0.8 \end{bmatrix} = [1.4, 1.6]$$

$$V_{cat} = [0, 2] \times \begin{bmatrix} 0.9 & 1.0 \\ 1.1 & 1.2 \end{bmatrix} = [2.2, 2.4]$$

2. **Self-Attention Mechanism:**

   Compute attention scores, apply softmax, and compute weighted sum of values.

Objective

The self-attention mechanism allows each word in a sentence to focus on other words in the sentence to understand the context better. This is done by computing attention scores that determine how much each word should focus on every other word.

Detailed Steps

1. **Compute Attention Scores:**

   - For each word, compute the dot product of its Query vector with the Key vectors of all words in the sentence. This gives us a score that indicates how much attention the word should pay to each other word.

   For example, consider the word "cat" with Query vector Q_{cat}:

   - $Q_{cat} = [0.6, 0.8]$

   Compute the scores for "cat" attending to "cat" and "sat":

   - For "cat" attending to "cat":

     $$\text{Score}(Q_{cat}, K_{cat}) = Q_{cat} \cdot K_{cat} = [0.6, 0.8] \cdot [1.4, 1.6] = (0.6 \times 1.4) + (0.8 \times 1.6) = 0.84 + 1.28 = 2.12$$

- For "cat" attending to "sat":

$$\text{Score}(Q_{cat}, K_{sat}) = Q_{cat} \cdot K_{sat} = [0.6, 0.8] \cdot [2.8, 2.8] = (0.6 \times 2.8) + (0.8 \times 2.8) = 1.68 + 2.24 = 3.92$$

2. **Apply Softmax to Scores:**

- Apply the softmax function to the attention scores to convert them into probabilities (attention weights). This normalization ensures that the scores add up to 1, which makes them easier to interpret.

$$\text{Softmax(Score)} = \frac{e^{\text{Score}_i}}{\sum_j e^{\text{Score}_j}}$$

For our example, the scores for "cat" attending to "cat" and "sat" are 2.12 and 3.92, respectively:

- Compute exponentials:

$$e^{2.12} \approx 8.32, \quad e^{3.92} \approx 50.63$$

- Normalize using softmax:

$$\text{Attention Weight}_{cat \to cat} = \frac{e^{2.12}}{e^{2.12} + e^{3.92}} = \frac{8.32}{8.32 + 50.63} \approx 0.14$$

$$\text{Attention Weight}_{cat \to sat} = \frac{e^{3.92}}{e^{2.12} + e^{3.92}} = \frac{50.63}{8.32 + 50.63} \approx 0.86$$

These weights indicate that "cat" pays more attention to "sat" than to itself.

3. **Compute Weighted Sum of Values:**

- Multiply each Value vector by its corresponding attention weight to get the weighted values.
- Sum these weighted values to get the new representation of the word.

For "cat", with Value vectors $V_{cat} = [2.2, 2.4]$ and $V_{sat} = [2.2, 2.4]$, and the attention weights computed earlier:

- Weighted Value for "cat" attending to "cat":

$$\text{Weighted Value}_{cat \to cat} = \text{Attention Weight}_{cat \to cat} \times V_{cat} = 0.14 \times [2.2, 2.4] = [0.308, 0.336]$$

- Weighted Value for "cat" attending to "sat":

$$\text{Weighted Value}_{cat \to sat} = \text{Attention Weight}_{cat \to sat} \times V_{sat} = 0.86 \times [2.2, 2.4] = [1.892, 2.064]$$

- Sum of weighted values:

$$\text{Output}_{cat} = \text{Weighted Value}_{cat \to cat} + \text{Weighted Value}_{cat \to sat} = [0.308, 0.336] + [1.892, 2.064] = [2.2, 2.4]$$

4. **Loss Calculation:**

Compare the model's output with the target output and compute the loss (e.g., cross-entropy loss).

5. **Backpropagation:**

   - Compute gradients of the loss with respect to W_Q, W_K, and W_V.
   - For W_Q:

$$\frac{\partial \text{Loss}}{\partial W_Q}$$

6. **Update Weights:**

   Adjust the weight matrices using gradient descent:

   $$ W_Q ←W_Q - \frac{ \text{Loss

}}{ W_Q}

W_K ←W_K - $\eta \frac{\partial \text{Loss}}{\partial W_K}$

W_V ←W_V - $\eta \frac{\partial \text{Loss}}{\partial W_V}$ $$

Summary

The weight matrices W_Q, W_K, and W_V are learned during training through the process of backpropagation and gradient descent. This iterative process adjusts the weights to minimize the loss, enabling the model to learn optimal transformations of input embeddings into Q, K, and V vectors, thereby improving the model's performance on the given task.

This process is repeated for every word in the sentence, allowing each word to focus on different words to varying degrees based on their contextual relevance. The learned weight matrices W_Q, W_K, and W_V enable the model to adaptively transform the input embeddings and capture complex dependencies and relationships within the data.

After the sum of weighted values is computed in the self-attention mechanism, the resulting vectors undergo several additional transformations before becoming the final output of the transformer layer. Here's a detailed breakdown of the steps that follow:

Steps After the Sum of Weighted Values

1. **Concatenation of Multi-Head Attention Outputs:**

   - In practice, transformers use multi-head attention, where multiple self-attention operations (heads) are performed in parallel. Each head operates independently with its own set of weight matrices W_Q, W_K, and W_V.
   - The outputs from each head are concatenated to form a single combined output. This allows the model to capture information from different representation subspaces.

$$\text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)$$

2. **Linear Transformation:**

   - The concatenated output is then passed through a linear transformation using a weight matrix W_O to combine the information from all heads.

$$\text{Output}_{\text{multi-head}} = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h) \times W_O$$

3. **Residual Connection and Layer Normalization:**

   - A residual connection (skip connection) is added, where the input to the multi-head attention layer is added to the output of the linear transformation. This helps with the flow of gradients during training and stabilizes the learning process.

$$\text{Output}_{\text{residual}} = \text{Input} + \text{Output}_{\text{multi-head}}$$

   - Layer normalization is applied to the result to standardize the output.

$$\text{Output}_{\text{normalized}} = \text{LayerNorm}(\text{Output}_{\text{residual}})$$

4. **Feed-Forward Network (FFN):**

   - The normalized output is passed through a position-wise feed-forward network. This consists of two linear transformations with a ReLU activation in between. This network applies the same transformation to each position separately.

$$\text{FFN}(x) = \text{ReLU}(x \times W_1 + b_1)$$
$$\text{Output}_{\text{ffn}} = \text{FFN}(\text{Output}_{\text{normalized}}) \times W_2 + b_2$$

5. **Second Residual Connection and Layer Normalization:**

   - Another residual connection is added, where the input to the feed-forward network is added to its output.

$$\text{Output}_{\text{residual, ffn}} = \text{Output}_{\text{normalized}} + \text{Output}_{\text{ffn}}$$

   - Layer normalization is applied again to standardize the output.

$$\text{Output}_{\text{final}} = \text{LayerNorm}(\text{Output}_{\text{residual, ffn}})$$