Comprehensive Step-by-Step Explanation of Self-Attention with Tables

Objective

To provide a detailed, step-by-step explanation of the self-attention mechanism used in Transformers, utilizing tables extensively to enhance clarity.

Introduction

Transformers have transformed natural language processing by enabling models to understand context more effectively. The self-attention mechanism is at the heart of this innovation. We'll walk through an example sentence to illustrate how self-attention works, focusing on each step and using tables to make the information clear.

Example Sentence

Let's use the simple sentence:

"The cat sat on the mat."

This sentence contains six words.

1 Step 1: Assign Word Embeddings

Purpose: Convert each word into a numerical vector (embedding) that represents its meaning.

Embedding Dimension: We'll use 2-dimensional embeddings for simplicity.

Assigned Embeddings

Position	**Word**	**Embedding** $[e_1, e_2]$
1	The	$[1, 0]$
2	cat	$[0, 1]$
3	sat	$[1, 1]$
4	on	$[0, -1]$
5	the	$[1, 0]$
6	mat	$[-1, 0]$

Table 1: Word Embeddings

Note: "The" and "the" are assigned the same embedding.

2 Step 2: Compute Queries (Q), Keys (K), and Values (V)

Purpose: Transform embeddings into queries, keys, and values to facilitate the attention mechanism.
Weight Matrices: For simplicity, we'll use identity matrices for W^Q, W^K, and W^V.

Weight Matrices

$$W^Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad W^K = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad W^V = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Compute Q, K, V for Each Word

Since we're using identity matrices:

$$\begin{aligned} Q_{\text{word}} &= W^Q \times \text{Embedding}_{\text{word}} = \text{Embedding}_{\text{word}} \\ K_{\text{word}} &= W^K \times \text{Embedding}_{\text{word}} = \text{Embedding}_{\text{word}} \\ V_{\text{word}} &= W^V \times \text{Embedding}_{\text{word}} = \text{Embedding}_{\text{word}} \end{aligned}$$

Resulting Q, K, V Vectors

Position	**Word**	**Query** Q	**Key** K	**Value** V
1	The	$[1, 0]$	$[1, 0]$	$[1, 0]$
2	cat	$[0, 1]$	$[0, 1]$	$[0, 1]$
3	sat	$[1, 1]$	$[1, 1]$	$[1, 1]$
4	on	$[0, -1]$	$[0, -1]$	$[0, -1]$
5	the	$[1, 0]$	$[1, 0]$	$[1, 0]$
6	mat	$[-1, 0]$	$[-1, 0]$	$[-1, 0]$

Table 2: Queries, Keys, and Values for Each Word

3 Step 3: Calculate Attention Scores

Purpose: Determine how much each word should pay attention to every other word.

Compute Raw Attention Scores

For each word, compute the dot product between its Query vector and the Key vectors of all words.
Formula:

$$\text{Score}(\text{word}_i, \text{word}_j) = Q_{\text{word}_i} \cdot K_{\text{word}_j}^{\top}$$

Let's compute the scores for **"cat" (Position 2)** attending to all words.

Queries and Keys for "cat"

$$Q_{\text{cat}} = [0, 1]$$

Compute Scores

word_j	Calculation	Score
The (1)	$[0, 1] \cdot [1, 0]^\top = (0 \times 1) + (1 \times 0)$	0
cat (2)	$[0, 1] \cdot [0, 1]^\top = (0 \times 0) + (1 \times 1)$	1
sat (3)	$[0, 1] \cdot [1, 1]^\top = (0 \times 1) + (1 \times 1)$	1
on (4)	$[0, 1] \cdot [0, -1]^\top = (0 \times 0) + (1 \times -1)$	-1
the (5)	Same as "The"	0
mat (6)	$[0, 1] \cdot [-1, 0]^\top = (0 \times -1) + (1 \times 0)$	0

Table 3: Raw Attention Scores for "cat"

4 Step 4: Scale the Scores

Purpose: Prevent large dot product values that could make the softmax function produce very small gradients.

Scaling Factor

- Key vector dimension $d_k = 2$
- Scaling factor: $\sqrt{d_k} = \sqrt{2} \approx 1.4142$

Compute Scaled Scores

Divide each score by 1.4142.

word_j	Raw Score	Scaled Score $\frac{\text{Score}}{\sqrt{2}}$
The (1)	0	$\frac{0}{1.4142} = 0$
cat (2)	1	$\frac{1}{1.4142} \approx 0.7071$
sat (3)	1	≈ 0.7071
on (4)	-1	$\frac{-1}{1.4142} \approx -0.7071$
the (5)	0	0
mat (6)	0	0

Table 4: Scaled Attention Scores for "cat"

5 Step 5: Apply the Softmax Function

Purpose: Convert the scaled scores into probabilities that sum to 1.

Compute Exponentials

$\textbf{word}_j$	**Scaled Score**	exp(**Scaled Score**)
The (1)	0	$e^0 = 1$
cat (2)	0.7071	$e^{0.7071} \approx 2.028$
sat (3)	0.7071	≈ 2.028
on (4)	-0.7071	$e^{-0.7071} \approx 0.493$
the (5)	0	1
mat (6)	0	1

Table 5: Exponentials of Scaled Scores

Compute Sum of Exponentials

$$\text{Sum} = 1 + 2.028 + 2.028 + 0.493 + 1 + 1 = 7.549$$

Compute Attention Weights

$$\text{Attention Weight} = \frac{\exp(\text{Scaled Score})}{\text{Sum}}$$

$\textbf{word}_j$	exp(**Scaled Score**)	**Attention Weight**
The (1)	1	$\frac{1}{7.549} \approx 0.1325$
cat (2)	2.028	$\frac{2.028}{7.549} \approx 0.2687$
sat (3)	2.028	≈ 0.2687
on (4)	0.493	$\frac{0.493}{7.549} \approx 0.0653$
the (5)	1	≈ 0.1325
mat (6)	1	≈ 0.1325

Table 6: Attention Weights for "cat"

Attention Weights for "cat"

Word	**Attention Weight (cat $\rightarrow$ word)**
The	0.1325
cat	0.2687
sat	0.2687
on	0.0653
the	0.1325
mat	0.1325

Table 7: Final Attention Weights for "cat"

6 Step 6: Compute the Output Vector

Purpose: Generate a new, context-aware representation for "cat" by combining the value vectors of all words, weighted by the attention weights.

Value Vectors

Word	Value Vector V
The	$[1, 0]$
cat	$[0, 1]$
sat	$[1, 1]$
on	$[0, -1]$
the	$[1, 0]$
mat	$[-1, 0]$

Table 8: Value Vectors

Compute Weighted Value Vectors

Multiply each value vector by the corresponding attention weight.

Word	Attention Weight	Value Vector V	Weighted Value Vector
The	0.1325	$[1, 0]$	$[0.1325, 0]$
cat	0.2687	$[0, 1]$	$[0, 0.2687]$
sat	0.2687	$[1, 1]$	$[0.2687, 0.2687]$
on	0.0653	$[0, -1]$	$[0, -0.0653]$
the	0.1325	$[1, 0]$	$[0.1325, 0]$
mat	0.1325	$[-1, 0]$	$[-0.1325, 0]$

Table 9: Weighted Value Vectors for "cat"

Sum the Weighted Value Vectors

Add up all the weighted value vectors component-wise.

First Component ($y_{\text{cat},1}$):

$$y_{\text{cat},1} = 0.1325 + 0 + 0.2687 + 0 + 0.1325 - 0.1325 = 0.4012$$

Second Component ($y_{\text{cat},2}$):

$$y_{\text{cat},2} = 0 + 0.2687 + 0.2687 - 0.0653 + 0 + 0 = 0.4721$$

Output Vector for "cat":

$$\mathbf{y}_{\text{cat}} = [0.4012, 0.4721]$$

7 Interpreting the Output Vector

Positive and Negative Contributions

First Component ($y_{\text{cat},1}$):

- **Positive Contributions**:
  - "The": +0.1325
  - "sat": +0.2687
  - "the": +0.1325
- **Negative Contribution**:

- "mat": -0.1325

- **Net Result**:
$$0.1325 + 0.2687 + 0.1325 - 0.1325 = 0.4012$$

Second Component ($y_{\text{cat},2}$):

- **Positive Contributions**:
  - "cat": $+0.2687$
  - "sat": $+0.2687$

- **Negative Contribution**:
  - "on": -0.0653

- **Net Result**:
$$0.2687 + 0.2687 - 0.0653 = 0.4721$$

Understanding Positive and Negative Values

- **Positive Values**: Indicate that the word contributes features that enhance the representation in that dimension.

- **Negative Values**: Indicate that the word contributes features that reduce or oppose the representation in that dimension.

8 Why We Did Each Step

1. **Assign Word Embeddings**:
   - **Purpose**: Convert words into numerical form for processing.
   - **Reason**: Embeddings capture semantic meanings in vector space.
2. **Compute Queries, Keys, and Values**:
   - **Purpose**: Prepare the data for the attention mechanism.
   - **Reason**: Queries and keys determine attention scores; values are the data to be combined.
3. **Calculate Attention Scores**:
   - **Purpose**: Measure the relevance between words.
   - **Reason**: Allows each word to focus on important words in the context.
4. **Scale the Scores**:
   - **Purpose**: Prevent large scores that could cause numerical instability.
   - **Reason**: Ensures the softmax function operates effectively.
5. **Apply the Softmax Function**:
   - **Purpose**: Convert scores into probabilities.
   - **Reason**: Normalizes attention scores so they sum to 1.
6. **Compute the Output Vector**:
   - **Purpose**: Generate context-aware word representations.
   - **Reason**: Combines information from relevant words, emphasizing important features.

9 Visual Summary Using Tables

Table of Attention Weights for "cat"

Word	Attention Weight	Value Vector V	Weighted Value Vector
The	0.1325	$[1, 0]$	$[0.1325, 0]$
cat	0.2687	$[0, 1]$	$[0, 0.2687]$
sat	0.2687	$[1, 1]$	$[0.2687, 0.2687]$
on	0.0653	$[0, -1]$	$[0, -0.0653]$
the	0.1325	$[1, 0]$	$[0.1325, 0]$
mat	0.1325	$[-1, 0]$	$[-0.1325, 0]$
Sum	**1.0**		**[0.4012, 0.4721]**

Table 10: Summary of Attention Weights and Weighted Values for "cat"

10 Connecting Back to the Limitations of RNNs

Why RNNs Struggle

- **Sequential Processing**: RNNs process words one at a time, making parallelization difficult.
- **Long-Term Dependencies**: Information can be lost over long sequences due to vanishing gradients.
- **Inefficient for Long Sentences**: Performance degrades as sentence length increases.

How Transformers Address These Issues

- **Parallel Processing**: Self-attention allows processing all words simultaneously.
- **Capturing Global Context**: Each word can attend to all other words, capturing long-range dependencies.
- **Efficiency**: Better utilization of computational resources and faster training times.

11 Key Takeaways

- **Self-Attention Mechanism**: Enables the model to focus on relevant parts of the input sequence.
- **Queries, Keys, and Values**: Fundamental components that facilitate the attention calculations.
- **Positive and Negative Contributions**: Allow the model to fine-tune word representations by adding or subtracting features.
- **Use of Tables**: Helps visualize the computations and understand the flow of information.

Final Thoughts

By working through this example step by step and utilizing tables, we've demystified the self-attention mechanism. Each component plays a crucial role in enabling Transformers to understand context and relationships within sequences effectively.

—

Note: If you have any further questions or need additional clarification on any part of this explanation, please feel free to ask!