Breaking Down the Attention is All You Need Paper

Introduction

Recurrent Models and Their Limitations

1. **Recurrent Models (RNNs):**
   - **Computation Factoring:** These models process input and output sequences by aligning the symbol positions with computation time steps.
   - **Hidden States Sequence:** They generate a series of hidden states (ht), where each state ht is a function of the previous state ht-1 and the current input at position t.
2. **Sequential Nature:**
   - **Sequential Computation:** Due to their inherently sequential nature, RNNs compute one step at a time, making it difficult to parallelize computations within a single training example.
   - **Impact on Training:** This sequential processing becomes problematic with longer sequences because it limits the ability to batch process multiple examples simultaneously, leading to memory constraints.
3. **Recent Improvements:**
   - **Factorization Tricks and Conditional Computation:** Some recent advancements have improved computational efficiency and model performance. For example, factorization tricks (as mentioned in reference [18]) and conditional computation methods (reference [26]) have helped.
   - **Fundamental Constraint:** Despite these improvements, the core issue of sequential computation in RNNs remains unsolved.

Attention Mechanisms

4. **Introduction of Attention:**
   - **Importance:** Attention mechanisms have become crucial in sequence modeling and transduction tasks because they can model dependencies in sequences without considering the distance between elements.
   - **Implementation with RNNs:** Most attention mechanisms are used alongside recurrent networks (RNNs) to enhance their performance (with a few exceptions noted in reference [22]).

Proposal of the Transformer Model

5. **Transformer Model:**
   - **Eschewing Recurrence:** The Transformer architecture proposed in this work eliminates the use of recurrence entirely.
   - **Reliance on Attention:** Instead of relying on sequential processing, the Transformer uses an attention mechanism to capture global dependencies between input and output sequences.
   - **Benefits:**

        - **Parallelization:** This design allows for much greater parallelization compared to RNNs.
        - **Training Efficiency:** The model can achieve state-of-the-art performance in translation tasks after only twelve hours of training on eight P100 GPUs.

Summary

The passage discusses the limitations of traditional recurrent models (RNNs) in handling long sequences due to their sequential nature, which hampers parallelization and efficient training. It highlights the role of attention mechanisms in overcoming these limitations to some extent. The authors then propose the Transformer model, which completely abandons recurrence in favor of attention mechanisms, enabling significant parallelization and improved training efficiency, leading to superior performance in sequence modeling tasks like translation.

This new approach marks a significant shift from the traditional methods and offers a more efficient solution for handling long sequences in various applications.

Background

1. Reducing Sequential Computation in Models:

- **Extended Neural GPU, ByteNet, ConvS2S:**
    - These models aim to reduce the sequential nature of computation using convolutional neural networks.
    - They parallelize the computation of hidden representations across all input and output positions.
    - **Operational Complexity:**
        - ConvS2S requires a number of operations that grows linearly with the distance between positions.
        - ByteNet requires a number of operations that grows logarithmically with the distance.
        - This increasing complexity makes it difficult to learn dependencies between distant positions.
- **Transformer Model:**
    - The Transformer reduces the number of operations to a constant, regardless of distance between positions.
    - This results in reduced resolution due to averaging attention-weighted positions.
    - The Transformer uses Multi-Head Attention to counteract this reduced resolution.

2. Self-Attention:

- **Definition:**
    - Self-attention relates different positions within a single sequence to compute its representation.

- **Applications:**
  - Successfully applied in reading comprehension, abstractive summarization, textual entailment, and learning task-independent sentence representations.

3. End-to-End Memory Networks:

- **Recurrent Attention Mechanism:**
  - These networks use a recurrent attention mechanism instead of sequence-aligned recurrence.
- **Performance:**
  - Effective in simple-language question answering and language modeling tasks.

4. Significance of the Transformer:

- **First Transduction Model with Self-Attention:**
  - The Transformer uniquely relies entirely on self-attention for computing input and output representations.
  - It does not use sequence-aligned RNNs or convolutional networks.

Transformer Model Structure

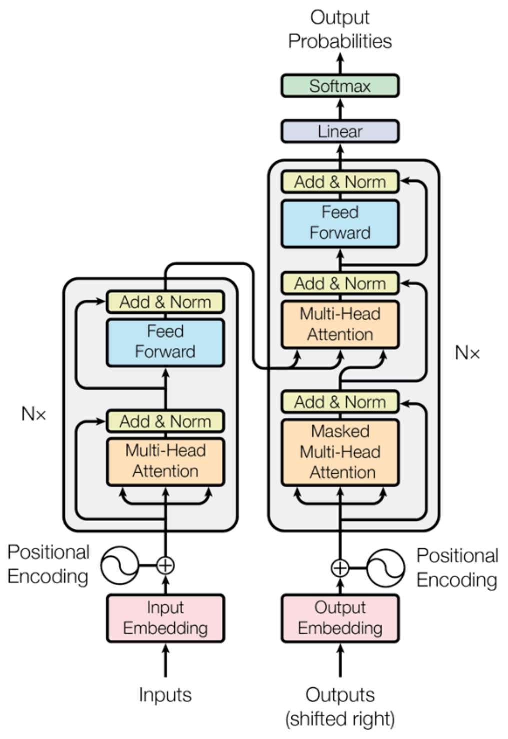


Figure 1: The Transformer - model architecture.

Sequence Transduction Models: These models are used for tasks that convert an input sequence to an output sequence. Examples include machine translation (translating sentences from one language to another), text summarization, and speech recognition.

The Transformer model follows the general encoder-decoder architecture but introduces significant innovations using self-attention mechanisms.

1. **Encoder in the Transformer:**
    - **Stacked Self-Attention Layers:** The Transformer encoder consists of multiple layers of self-attention mechanisms stacked on top of each other. Self-attention allows each symbol in the input sequence to attend to all other symbols, capturing dependencies regardless of their distance in the sequence.
    - **Point-Wise Fully Connected Layers:** After the self-attention mechanism, each symbol's representation is passed through fully connected feed-forward networks applied independently to each position in the sequence.
    - **Residual Connections and Layer Normalization:** These techniques are used to stabilize and enhance the training process.
2. **Decoder in the Transformer:**
    - **Stacked Self-Attention Layers:** Similar to the encoder, the decoder also has multiple self-attention layers. However, it has an additional layer of attention that allows it to attend to the encoder's output representations z.
    - **Masked Self-Attention:** In the decoder, the self-attention mechanism is masked to prevent attending to future positions. This ensures that the prediction for a given position can only depend on previous positions.
    - **Auto-Regressive Generation:** The decoder generates the output sequence one symbol at a time, using the previously generated symbols and the encoder's output.

Summary

- **Overall Architecture:**
    - The encoder converts an input sequence of symbols into continuous representations.
    - The decoder generates an output sequence from these continuous representations, one symbol at a time, in an auto-regressive manner.
- **Innovations in the Transformer:**
    - Utilizes self-attention mechanisms to capture dependencies across the entire sequence.
    - Employs pointwise fully connected layers for transformation and feature extraction.
    - Uses stacked layers in both the encoder and decoder to enhance model capacity and performance.

By using self-attention and fully connected layers, the Transformer model improves on traditional sequence transduction models, allowing for better handling of long-range dependencies and more efficient parallel processing.

Encoder Structure

1. **Stack of Identical Layers:**

- The encoder consists of N=6N = 6N=6 identical layers. This means there are six layers in the encoder, each designed in the same way.

2. **Sub-Layers in Each Layer:**
   - Each of these layers has two sub-layers:
     1. **Multi-Head Self-Attention Mechanism:**
        - **Self-Attention:** This mechanism allows the model to focus on different parts of the input sequence when producing each part of the output sequence.
        - **Multi-Head:** Instead of having a single attention mechanism, multiple attention mechanisms (heads) run in parallel. This allows the model to capture different types of relationships and features from different parts of the sequence simultaneously.
     2. **Position-Wise Fully Connected Feed-Forward Network:**
        - **Position-Wise:** This network applies the same feed-forward network to each position (word) in the sequence independently.
        - **Fully Connected:** This means that every neuron in one layer is connected to every neuron in the next layer.
3. **Residual Connections:**
   - Residual connections are employed around each of the two sub-layers. This technique was introduced by He et al. in their ResNet architecture.
   - **Purpose:** Residual connections help in mitigating the vanishing gradient problem, making it easier to train deep neural networks. They allow the model to learn identity mappings, making it easier for the network to learn the desired function.
   - **Implementation:** For each sub-layer, the input to the sub-layer xxx is added to the output of the sub-layer. This sum is then passed through a normalization step. Mathematically, this can be represented as:

     LayerNorm(x+Sublayer(x))

     where Sublayer(x) is the function implemented by the sub-layer itself.

4. **Layer Normalization:**
   - After adding the residual connection, layer normalization is applied.
   - **Purpose:** Layer normalization helps in stabilizing and accelerating the training process. It ensures that the outputs have a mean of zero and a variance of one, which helps in maintaining the numerical stability of the model.
   - **Implementation:** This involves computing the mean and variance of the activations, and then normalizing each activation accordingly.

How Does Layer Normalization Work?

1. **Calculate Mean and Variance:**
   - For each data point (e.g., each position in a sentence), calculate the mean and variance of the activations in that layer.
2. **Normalize:**

- Subtract the mean and divide by the standard deviation (which is the square root of the variance). This scales the activations to have a mean of zero and a variance of one.

3. **Scale and Shift:**
   - Apply learnable parameters to scale and shift the normalized values. This step ensures that the network can still learn the best representation for the data.

Simple Example

Imagine you have a list of numbers: [3, 5, 2, 8].

1. **Calculate Mean:**
   - $(3 + 5 + 2 + 8) / 4 = 4.5$
2. **Calculate Variance:**
   - $[(3-4.5)^2 + (5-4.5)^2 + (2-4.5)^2 + (8-4.5)^2] / 4 = 5.25$
3. **Calculate Standard Deviation:**
   - $\sqrt{5.25} \approx 2.29$
4. **Normalize:**
   - $(3 - 4.5) / 2.29 \approx -0.65$
   - $(5 - 4.5) / 2.29 \approx 0.22$
   - $(2 - 4.5) / 2.29 \approx -1.09$
   - $(8 - 4.5) / 2.29 \approx 1.53$

Now the numbers are normalized, having a mean of zero and a variance of one.

Why It's Helpful

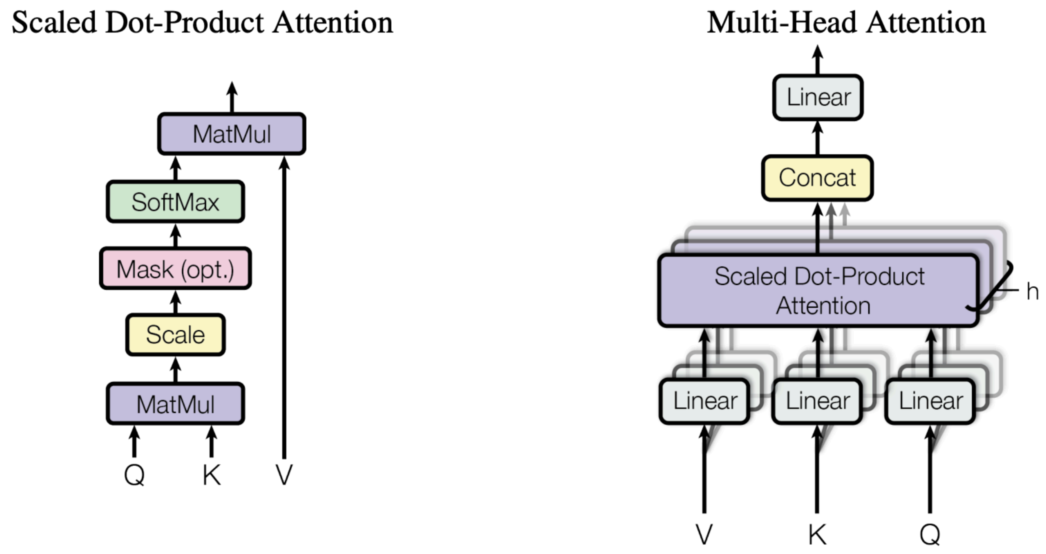
- **Consistent Activation Ranges:** Ensures that activations are in a consistent range, which makes it easier for the network to learn.
- **Reduces Sensitivity:** Reduces the sensitivity of the model to the scale of the inputs, making the training process smoother and more reliable.

Conclusion

Layer normalization helps neural networks train faster and more reliably by standardizing the inputs to each layer. It ensures that the activations have a mean of zero and a variance of one, making the training process more stable and efficient.

5. **Dimensional Consistency:**
   - **Output Dimension:**
     - Ensuring all layers and sub-layers have the same output dimension (d_model w= 512) is crucial for the residual connections to work properly.
   - **Embeddings:**
     - The initial step converts each word into a vector of dimension 512.
   - **Consistency:**

- This dimension is maintained throughout the network, ensuring smooth and



  stable transformations.

Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

Breaking down the attention mechanism equation

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Inputs:

- **Q (Query):** A matrix representing the queries. Each query corresponds to an element in the input sequence that is seeking to understand its relationships with other elements.
- **K (Key):** A matrix representing the keys. Each key corresponds to an element in the input sequence that other elements will query against.
- **V (Value):** A matrix representing the values. Each value corresponds to an element in the input sequence that holds the information to be retrieved based on the queries and keys.

Steps in the Attention Mechanism:

1. **Dot Product of Query and Key Transpose:**
   - Compute the dot product of the query matrix Q with the transpose of the key matrix K^T.

- This operation produces a score matrix that represents the similarity between each query and key pair.

$$QK^T$$

2. **Scale by Square Root of Key Dimension:**

   - Scale the dot product result by dividing it by the square root of the dimension of the keys, $sqrt(dk)$. This scaling helps in stabilizing the gradients during training.

$$\frac{QK^T}{\sqrt{d_k}}$$

3. **Softmax Function:**

   - Apply the softmax function to the scaled dot product result. The softmax function converts the scores into probabilities that sum to 1. This step ensures that the weights used to combine the values are normalized.

$$\text{softmax}(\frac{QK^T}{\sqrt{d_k}})$$

4. **Weighting the Values:**

   - Multiply the resulting attention weights with the value matrix VVV. This step produces a weighted sum of the values, where the weights are determined by the similarity between the queries and keys.

$$\text{Attention}(Q,K,V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

5. Summary

- **Query (Q):** Represents what each element in the sequence wants to know.
- **Key (K):** Represents the elements to be queried against.
- **Value (V):** Contains the information to be retrieved based on the query-key relationship.

6. **Process Flow:**

   - Compute the dot product of Q and the transpose of K.
   - Scale the dot product result by dividing by the square root of the dimension of the keys (dk).
   - Apply the softmax function to get the attention weights.
   - Multiply the attention weights with V to get the final attention output.

This attention mechanism allows the model to focus on different parts of the input sequence when processing each element, capturing dependencies and relationships more effectively.

Below I am going to add my conceptual understanding of why it is important to have multi-attention blocks rather than having a singular attention block

Capturing Diverse Features and Relationships

- **Multiple Perspectives:** Each attention head learns to focus on different parts of the sequence and different types of relationships. This means that while one head might focus on short-term dependencies (nearby words), another might capture long-term dependencies (words far apart in the sequence).
- **Richer Representations:** By capturing various aspects of the data, multi-head attention provides a more comprehensive understanding of the sequence. For instance, in a sentence, different heads might capture different syntactic structures

Improving Model Robustness and Generalization

- **Redundancy and Complementarity:** Different heads provide complementary information, which enhances the robustness of the model. If one head misses a specific dependency, another might capture it. This redundancy helps in making the model less sensitive to noise or variations in the data.
- **Better Generalization:** The diverse features captured by multiple heads help the model generalize better to new, unseen data. It can handle various patterns and structures that might not have been present in the training data.

3. Reducing Information Bottlenecks

- **Parallel Processing:** Each attention head operates independently, allowing the model to process multiple parts of the sequence simultaneously. This parallelism helps in reducing the risk of information bottlenecks, where critical information might be lost or overlooked if processed sequentially.
- **Efficient Computation:** Multi-head attention distributes the computational load, making it easier to handle large and complex sequences. Each head processes a smaller subset of the data, which can be more manageable and efficient.

Process Breakdown

1. **Linear Projection:**
   - **Queries, Keys, and Values:**
     - In the Transformer, the input data is first converted into queries (Q), keys (K), and values (V). Each of these is typically of dimension modeld.
   - **Linear Projection:**
     - Instead of directly using these dmodel-dimensional vectors, the model linearly projects the queries, keys, and values h times using different learned linear projections.
     - These projections transform the queries, keys, and values into smaller dimensions: dk for keys and queries, and dv for values.
     - This means that each query, key, and value is split into h smaller vectors, each of dimension dk or dv .
2. **Parallel Attention Function:**
   - **Performing Attention:**
     - The attention function is then performed in parallel on each of these hhh projected versions of queries, keys, and values.
     - For each of the h heads, this results in dv-dimensional output values.
   - **Parallel Processing:**
     - By performing attention in parallel, the model can capture various aspects of the input data simultaneously.
3. **Concatenation and Final Projection:**
   - **Concatenation:**
     - The h sets of dv-dimensional output values from the parallel attention heads are concatenated.
   - **Final Projection:**
     - This concatenated vector is then linearly projected again to form the final output values.
   - **Dimensionality:**
     - This final projection ensures that the output dimensions match the original input dimensions, maintaining consistency in the model.

Benefits of Multi-Head Attention

1. **Jointly Attending to Information:**
   - **Different Representation Subspaces:**
     - Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions in the input sequence.
     - Each head can focus on different aspects of the input, capturing a richer set of features and dependencies.
2. **Avoiding Averaging Inhibition:**
   - **Single Head Limitation:**

        - With a single attention head, the model would average the attention over all positions, which can inhibit the model's ability to focus on specific details.
    - **Multiple Heads:**
        - Multiple attention heads prevent this issue by allowing the model to attend to multiple aspects and relationships simultaneously, without averaging them into a single representation.

Summary

Process Flow:

1. Linearly project the queries, keys, and values h times using different learned projections, resulting in dk_-dimensional queries and keys, and dv-dimensional values.
2. Perform the attention function in parallel on each of these projected versions, yielding dv_xdimensional output values.
3. Concatenate the output values from all heads.
4. Linearly project the concatenated output to obtain the final values.

Key Advantages:

- **Richer Representations:** Each attention head can focus on different parts of the sequence, capturing a wider range of dependencies and features.
- **Improved Performance:** By attending to various aspects of the input data simultaneously, the model can better understand and process complex patterns.
- **Enhanced Flexibility:** The model can adapt to different types of relationships and structures within the data, making it more robust and effective.

Multi-head attention significantly enhances the Transformer model's ability to process and understand sequences, leading to improved performance in various tasks such as machine translation, text summarization, and more.

$$\begin{aligned} \mathrm{MultiHead}(Q, K, V) &= \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O \\ \text{where}\ \mathrm{head_i} &= \mathrm{Attention}(QW^Q_i, KW^K_i, VW^V_i) \end{aligned}$$

Where the projections are parameter matrices $W^Q_i \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W^K_i \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W^V_i \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

Step by Step Explanation

Input Embeddings:

- The input sequence is first embedded into a 512-dimensional space, resulting in an embedding matrix X with dimensions (T,512)), where T is the sequence(token) length.

Linear Projections:

- The input embeddings are linearly projected to obtain the queries Q, keys K, and values V. This is done using learned weight matrices, where T is the sequence length.

Each attention head has its own set of projection matrices:

- W_Q ∈ R^512x64
- W_K ∈ R^512x64
- W_V ∈ R^512x64

The projections are computed as:

- Q = X * W_Q (dimensions T x 64)
- K = X * W_K (dimensions T x 64)
- V = X * W_V (dimensions T x 64)

$$\text{Attention}(Q,K,V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Concatenating Multiple Heads:t

If there are h attention heads, each head produces an output of dimensions (T,64). These outputs are concatenated to form a single matrix of dimensions (T,64×h). For example, with 8 heads, the concatenated output will have dimensions (T,512).

🎬 The concatenated output is then linearly projected back to the original model dimension d_model using a final projection matrix W_O ∈ R^512x512:

- Final Output = Concatenated Output * W_O

This final output has dimensions (T, 512), maintaining consistency with the input dimension.

Summary

- **Initial Embedding:** Input sequence embedded to 512 dimensions.
- **Projection:** Queries, keys, and values projected to 64 dimensions for each head.
- **Dot Product:** Queries and keys dot product yields attention scores (T, T).

- **Scaling and Softmax:** Scaled scores converted to attention weights.
- **Weighted Sum:** Attention weights applied to values to get attention output (T, 64).
- **Concatenation:** Outputs from multiple heads concatenated to (T, 512).
- **Final Projection:** Concatenated output projected back to (T, 512).

This multi-head attention mechanism allows the Transformer to capture diverse patterns and dependencies from different subspaces, improving its ability to model complex relationships in the data.