WORDS CAN LOOK AT ALL OTHER WORDS
MODEL LEARNS WHICH WORDS ARE IMPORTANT
TO WHICH

A Comprehensive Analysis of "Attention Is All You Need"

[redacted]

September 20, 2025

Introduction

The 2017 paper "Attention Is All You Need" by Vaswani et al. introduced the **Transformer**, a novel network architecture that fundamentally shifted the paradigm of sequence modeling in Natural Language Processing (NLP). By dispensing with recurrence and convolutions entirely, the Transformer proposed a model based solely on attention mechanisms. This design not only achieved a new state-of-the-art in machine translation but also offered massive parallelization, drastically reducing training times. This report provides a detailed deconstruction of the concepts, architecture, and mechanisms presented in the paper, incorporating in-depth explanations of its foundational and technical components, built from an extensive discussion of each element.

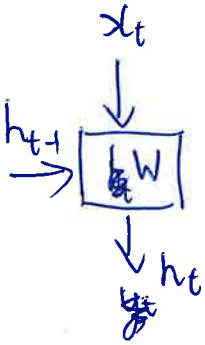
1 The Pre-Transformer Landscape - Recurrent Models and Their Limitations

To appreciate the Transformer's innovation, one must first understand the architecture that dominated sequence transduction tasks: the Recurrent Neural Network (RNN).

1.1 The Mechanics of Recurrent Neural Networks (RNNs)

RNNs are a class of neural networks designed specifically to handle sequential data. Unlike standard feed-forward networks, which process inputs independently, RNNs have a "loop" that allows them to persist information from one step of a sequence to the next. This persisted information is called the **hidden state**, which acts as the network's memory.

- **Sequential Computation:** An RNN processes a sequence token-by-token. The hidden state at a given step, h_t, is a function of the input at that step, x_t, and the hidden state from the previous step, h_{t-1}. The update is governed by the following equation:



$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

  - h_t: The new hidden state (a vector) at time step t, representing the network's memory after seeing the current input.
  - h_{t-1}: The hidden state from the previous step.
  - x_t: The input vector (e.g., a word embedding) for the current step.
  - W_{hh}, W_{xh}, b_h: These are the weight matrices and bias vector that the network learns during training. Crucially, these same weights are used at every single time step.

Example: Processing "the cat sat"

IF ATTN., ALL HIDDEN STATES ARE AN RNN'S OUTPUT VECTOR[1]

THE OUTPUT OF THE RNN IS THE FINAL HIDDEN STATE (IF NO ATTENTION)

1. **Time Step 1 ("the"):** The network receives the vector for "the" (x_1) and an initial hidden state (h_0, usually zeros). It calculates h_1, which now represents "the".
2. **Time Step 2 ("cat"):** The network receives the vector for "cat" (x_2) and the previous state (h_1). It combines them to produce h_2, which now represents "the cat".
3. **Time Step 3 ("sat"):** It receives "sat" (x_3) and h_2 to produce h_3, a representation of the full phrase "the cat sat".

This inherent sequentiality means the computation cannot be parallelized within a single sequence, making it slow for long inputs.

- **Long-Range Dependencies:** The dependency on the immediately preceding step makes it difficult for information and gradients to travel over long distances in the sequence, posing a challenge for learning relationships between distant words (the vanishing gradient problem).

1.2 The RNN Encoder-Decoder Architecture and the Decoding Process

For sequence-to-sequence tasks like machine translation, RNNs were used in an **encoder-decoder** framework.

- **Encoder:** An RNN reads the input sentence one word at a time, updating its hidden state at each step. The final hidden state of the encoder is treated as a summary of the entire input sentence, known as the **context vector**.
- **Decoder:** A separate RNN is initialized with the encoder's context vector. Its task is to generate the output sentence word-by-word in an **autoregressive** manner. The detailed process is as follows:

  1. **Initialization:** The decoder's first hidden state (d_0) is initialized with the encoder's final context vector. This is the crucial step where the summary of the input sentence is transferred.
  2. **First Word Generation:** The decoder takes a special `<start-of-sequence>` token as its first input. It processes this input and its initial hidden state to produce an output vector, which is passed through a softmax layer to create a probability distribution over the entire vocabulary. The word with the highest probability is chosen as the first output word (e.g., "die").
  3. **Subsequent Word Generation:** For each subsequent step, the word generated in the previous step is used as the input (e.g., "die" is fed in to generate "Katze"). The decoder updates its hidden state and generates the next word. This process is repeated until an `<end-of-sequence>` token is generated.

1.3 The Information Bottleneck and the Rise of Attention

This classic architecture suffered from a critical **information bottleneck**. The encoder had to compress the meaning of a potentially long, complex sentence into a single, fixed-size context vector.

The solution was the **attention mechanism**, which was used *in conjunction with* RNNs. Instead of just using the encoder's final hidden state, attention allowed the decoder, at every step, to look back at *all* of the encoder's hidden states.

- **Mechanism:** For each output word it generated, the decoder would:

  ↳ THIS IS LEARNT WHILE TRAINING

  1. Use its current hidden state to score every hidden state from the encoder. This comparison produces a set of **attention scores**, indicating the relevance of each input word for the current decoding step.
  2. Convert these scores into **attention weights** (probabilities that sum to 1) using a softmax function.
  3. Create a **dynamic context vector** by taking a weighted sum of all encoder hidden states. This vector is custom-tailored for the current decoding step, focusing only on relevant parts of the input.
  4. Use this dynamic vector (along with its own hidden state) to generate the next word.

- **Remaining Limitation:** While attention solved the information bottleneck, the encoder process itself remained sequential. The hidden states were still generated one after another, as the calculation for h_t depended on h_{t-1}. This computational limitation was the final problem the Transformer aimed to solve.

2 The Transformer Revolution - A New Architecture

The Transformer's central thesis is that attention is not just a helpful add-on for RNNs; it is powerful enough to replace them entirely.

2.1 The Core Idea: Self-Attention

The foundational block of the Transformer is **self-attention**. This mechanism allows every word in a sequence to directly relate to and weigh the importance of every other word in the *same* sequence. This results in context-aware word representations.

- **Parallelization:** Unlike an RNN, self-attention has no sequential dependencies. The representation for every word can be calculated simultaneously.
- **Constant Path Length:** The path for information to travel between any two words is of length one, making it trivial to model long-range dependencies.
- **Technical Implementation (Query, Key, Value):** To achieve this, each input word embedding is transformed into three distinct vectors by multiplying it with three learned weight matrices (W^Q, W^K, W^V):

  ALSO LEARNED ←

  $\vec{w}$: WORD VECTOR

  - **Query (q_i):** A representation of a word used to score its relevance against all other words. It effectively acts as a probe to find relevant information from the rest of the sequence. $\vec{w_i} \cdot W^Q$
  - **Key (k_j):** A representation of another word used for establishing compatibility. Its role is to be compared against the query vector. The dot product of q_i and k_j produces a raw compatibility score. $\vec{w_j} \cdot W^K$
  - **Value (v_j):** A representation of a word that contains its actual content or meaning. This is the vector that will be used in the final weighted sum if its corresponding key is deemed relevant. $\vec{w_j} \cdot W^V$

  The attention score is the dot product of a query and a key, and the final output is a weighted sum of all value vectors in the sequence.

INPUT EMBEDDINGS = $\vec{W}$ + POSITONAL ENCODINGS

2.2 The Transformer's Encoder-Decoder Structure

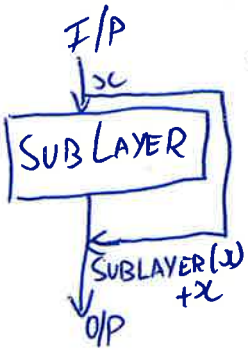
The Transformer retains the high-level encoder-decoder framework but replaces the recurrent layers with stacks of self-attention and feed-forward layers.

ENCODER O/P SHAPE DEPENDS ON No. OF WORDS

- **The Encoder Stack:** Composed of a stack of **N=6** identical layers. Each layer contains two primary sub-layers: SECTION 3.3
  1. A **Multi-Head Self-Attention Mechanism**, where the input sequence communicates with itself.
  2. A simple, **Position-wise Fully Connected Feed-Forward Network**, which processes each position's representation independently.
- **The Decoder Stack:** Also composed of a stack of **N=6** identical layers. Each layer has **three** sub-layers: SECTION 3.3
  1. A **Masked Multi-Head Self-Attention Mechanism**, for the decoder to look at previously generated words in the output sequence.
  2. A **Multi-Head Attention Mechanism** over the Encoder's output. This is where the decoder gets information from the input sentence. The queries come from the previous decoder layer, while the keys and values come from the encoder stack's output.
  3. A **Position-wise Fully Connected Feed-Forward Network.**

2.3 Essential Components for Deep Architectures

Training a deep stack of layers is made possible by two critical techniques applied after each sub-layer.



- **Residual Connections:** To combat the vanishing gradient problem, the Transformer uses residual (or "skip") connections. The input to a sub-layer, x, is added directly to the output of that sub-layer, 'Sublayer(x)'. The resulting operation is 'output = x + Sublayer(x)'. This structure means the layer only needs to learn the **residual** ('F(x) = output - x'). This makes it much easier to learn identity mappings (by driving the layer's weights to zero) and allows gradients to flow unimpeded through the network via the skip connection "highway," enabling stable training of very deep models.

- **Layer Normalization:** To stabilize the training process, the output of the residual connection is immediately normalized. Layer Normalization rescales the activations within a layer for each training example to have a mean of 0 and a standard deviation of 1. This ensures a consistent distribution of inputs to subsequent layers, accelerating training. The full operation is 'LayerNorm(x + Sublayer(x))'.

2.4 The Decoder's Masked Self-Attention

The decoder's self-attention layer has a crucial modification: it is **masked**.

- **Purpose:** To preserve the auto-regressive property. When predicting the word at position i, the model must only be allowed to see the words at positions less than i. It cannot "cheat" by looking at future words, which would make learning trivial and cause it to fail during real-world inference.

- **Mechanism:** The mask is implemented directly within the attention calculation before the softmax step.

1. **Score Calculation:** The raw attention scores are calculated via the matrix multiplication QK^T.
2. **Mask Application:** A mask is added to the score matrix, setting all values corresponding to future positions (the upper triangle of the matrix) to negative infinity ($-\infty$).
3. **Softmax:** When the softmax function is applied to these scores, e raised to $-\infty$ becomes 0. This ensures that the attention weights for all future words are zero, and they contribute nothing to the representation of the current word.

3 The Core Mechanisms of the Transformer in Detail

3.1 Word Embeddings and Positional Encodings

- **Learned Embeddings:** The Transformer creates its own vocabulary and learns embedding vectors from scratch during training. Unlike pre-trained embeddings (e.g., Word2Vec), which are trained on a general corpus and loaded in, these embeddings are initially random and are optimized to be specifically useful for the task at hand (e.g., translation).

- **Positional Encodings:** Since self-attention is permutation-invariant, the model has no inherent sense of word order. To solve this, **Positional Encodings** are added to the input embeddings. The paper uses fixed sinusoidal functions:

$$PE_{(pos,2i)} = \sin(\text{pos}/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = \cos(\text{pos}/10000^{2i/d_{\text{model}}})$$

The pairing of sine (for even dimensions) and cosine (for odd dimensions) is a critical design choice. It allows the positional encoding for any position $pos + k$ to be represented as a linear transformation (a **rotation**) of the encoding at pos. This property makes it easy for the model's linear layers to learn about relative positions.

3.2 Scaled Dot-Product Attention

This is the core computational unit. For a set of queries (Q), keys (K), and values (V), the output is computed as:

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

- **Query, Key, and Value Vectors:** Each input word embedding is projected into three vectors:
  - **Query (Q):** A representation of the current word used to score its relevance against all other words.
  - **Key (K):** A representation of all words in the sequence used to be scored against a query.

- **Value (V):** A representation of all words that gets aggregated in the final weighted sum.

- **Scaling:** The dot-product scores are scaled by $\sqrt{d_k}$ (the dimension of the key vectors) to prevent the softmax function from entering regions with small gradients, thus stabilizing training.

3.3 Multi-Head Attention

Instead of performing attention once, the Transformer uses **Multi-Head Attention** (with h=8 heads).

- **Motivation:** A single attention head might average different types of relationships (e.g., syntactic and semantic), diluting the information. Multiple heads allow the model to jointly attend to information from different "representation subspaces," with each head specializing in capturing different types of relationships. For example, one head might learn to track syntactic dependencies while another tracks semantic similarity.

- **Mechanism:** BASICALLY, 8 SETS OF W^Q, W^K, W^V (LEARNED WEIGHTS)

  1. The Q, K, and V matrices are linearly projected 8 times into lower-dimensional subspaces using different weight matrices for each head. This creates 8 different "lenses" through which to view the data.
  2. Scaled Dot-Product Attention is applied in parallel to each of these 8 projections.
  3. The 8 resulting output vectors are concatenated into a single larger vector.
  4. This concatenated vector is passed through a final linear layer to produce the final, unified output that combines the insights from all heads.

4 Validation and Impact

- **State-of-the-Art Performance:** The Transformer set new state-of-the-art BLEU scores on both English-to-German (28.4) and English-to-French (41.8) machine translation tasks, outperforming all previous models and ensembles.

- **Training Efficiency:** Due to its parallelizable nature, the Transformer was trained for a fraction of the cost of competing models, achieving superior results in just 3.5 days on 8 GPUs.

- **Generalization:** The model also performed exceptionally well on English constituency parsing, proving its general applicability beyond translation.

- **Lasting Impact:** The Transformer architecture has become the foundation for nearly all modern NLP models, including large language models like BERT, GPT, and T5. Its introduction marked a pivotal moment, shifting the field's focus from recurrence to attention-based architectures.

Conclusion

"Attention Is All You Need" presented a paradigm shift in sequence modeling. By demonstrating that a model based entirely on self-attention could not only match but significantly outperform recurrent architectures in both accuracy and efficiency, the paper laid the groundwork for the

next generation of models in artificial intelligence. The Transformer's elegant design and powerful mechanisms have defined the state-of-the-art and continue to be the dominant architecture in the field today.