Attention Is All You Need

PDF : https://arxiv.org/pdf/1706.03762

Abstract

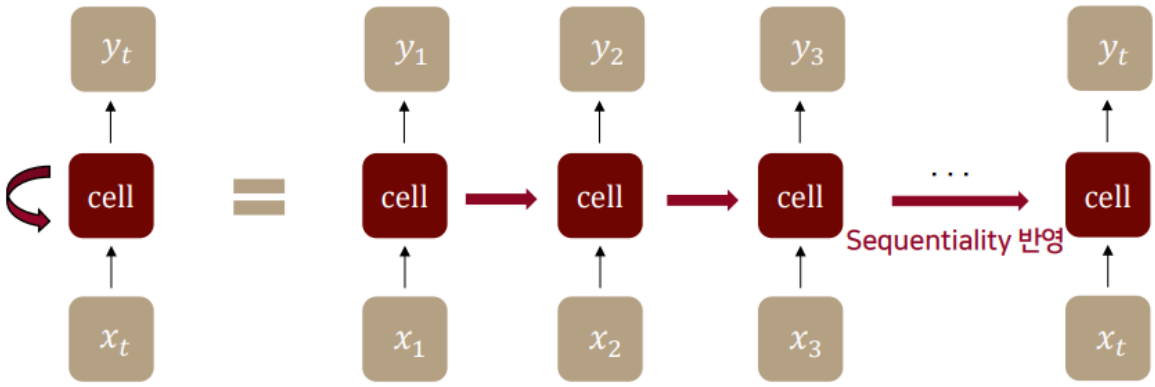
Existing sequence transduction models: complex RNN/CNN encoder–decoder models

Transformer: replaces recurrence/convolutions with attention-only computation

Introduction

RNN (LSTM, GRU) → SOTA in sequence modeling & transduction problems

Recurrent models : By aligning positions to computation steps, recurrent models handle sequential data effectively, **but this makes parallelization impossible**



Attention Mechanism : allows for modeling dependencies **regardless of their distance** in the input or output sequences

Transformer : based solely on **Attention mechanisms**, which allows it to model **global dependencies** and be **parallelized**

Background

Extended Neural GPU, ByteNet, ConvS2S : CNN based models
The number of operations required to relate signals from two arbitrary positions **grows with the distance** between them

- **ConvS2S** : Linear ($O(n)$)
- **ByteNet** : Logarithmic ($O(\log n)$)

→ Learning dependencies between distant point becomes hard

Transformer :

Pro : Constant number of operations ($O(1)$)

Con : reduced effective resolution (due to averaging attention-weighted position)

→ Implements **Multi-Head Attention** to mitigate this issue

Self-attention (Intra-attention)

: Attention mechanism relating different positions of a **single sequence** in order to compute a representation of the sequence

End-to-end memory networks

: **Recurrent attention mechanism.** They use attention, they still operate within a recurrent framework

Transformer : first transduction model **relying entirely on self-attention** precluding RNNs or CNNs

Model Architecture

Encoder : input symbol representation sequence → continuous representation

$(x_1, \ldots, x_n) \rightarrow \mathbf{z} = (z_1, \ldots, z_n)$

Decoder : given z → generate symbol representation sequence

$\mathbf{z} \rightarrow (y_1, \ldots, y_m)$

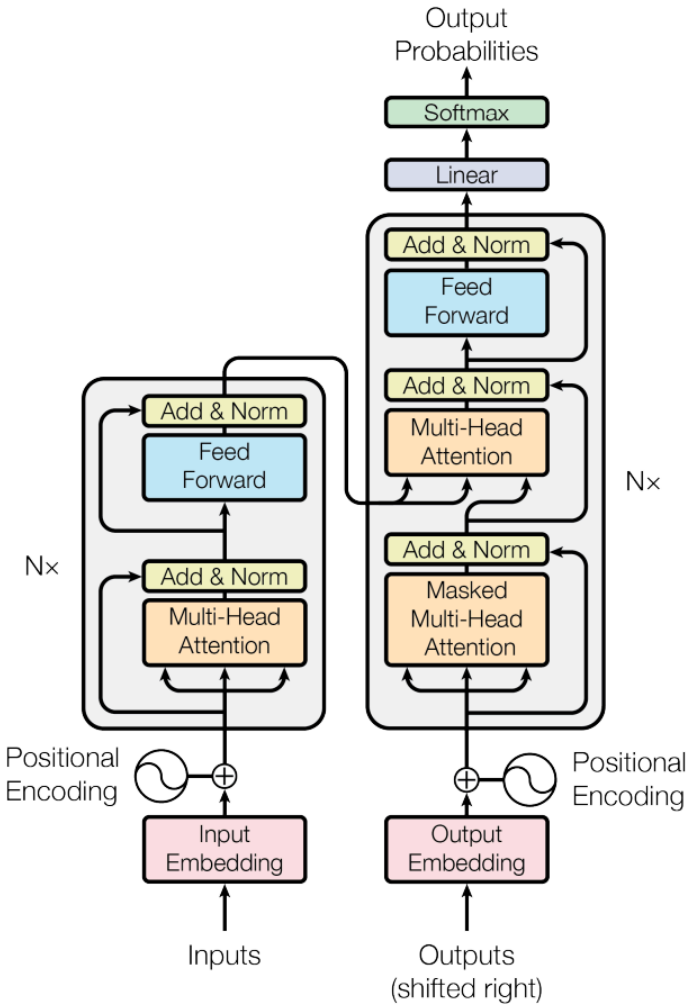


Figure 1: The Transformer - model architecture.

Encoder and Decoder Stacks

Encoder :

Composed of 6 stacks of identical layers, each having 2 sub-layers.

1. **Multi-head self-attention** mechanism
2. Position-wise fully connected **Feed-forward network**

Residual connection, followed by **layer normalization**

output :$\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$ → output dimension; $d_{model} = 512$

Decoder :

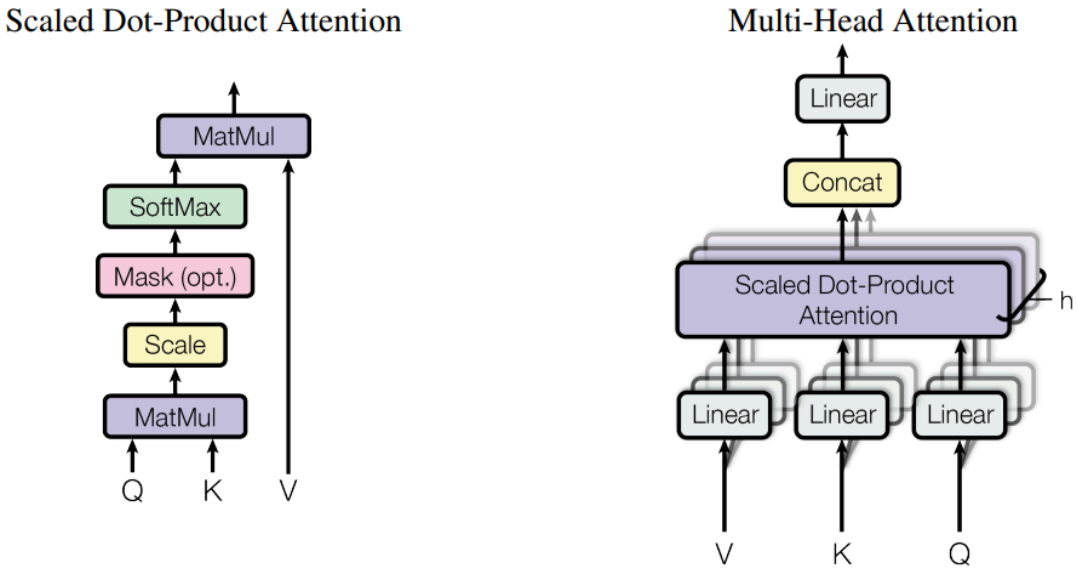
Composed of 6 stacks of identical layers, encoder layer + 1 sub-layers

1. **Masked Multi-Head Attention** Layer
    - To prevent positions from attending to subsequent positions
    - Ensures that the predictions for position i can depend only on the known outpus at positions less than i
2. **Multi-head self-attention** mechanism
    - Attention over the output of the encoder stack
3. Position-wise fully connected **Feed-forward network**

Residual connection, followed by **layer normalization**

Attention

Mapping a **query** and a set of **key-value pairs** to an output, where **keys,query,values** are all vectors



output : weighted sum of the values

→ weight assigned to each value is computed by a compatibility function of the query with the corresponding key

Scaled Dot-Product Attention

d_k : dimension of queries and keys

d_v: dimension of values

Q : Query, K : Key, V : Value

$\langle Q, K \rangle$: Dot product of Key and Query (To check the similarity)

Normalize by dividing $\sqrt{d_k}$ to prevent **exploding values**

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

Additive attention vs Dot-product attention

Additive attention : computes the compatibility function using a **feed-forward network with a single hidden layer**

Dot-product attention : identical to Transformer's algorithm except its scaling factor

Highly optimized matrix multiplication code makes dot-product attention much faster in practice, although their theoretical computational complexities are similar.

Multi-Head Attention

To linearly project queries, keys, values **multiple times** (h times) was beneficial

→ Concatenate the outputs and **apply a linear projection** to obtain the final values

→ Allows the model to jointly attend to information from **different representation subspaces** at **different positions**

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O,$$
$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Projections are **parameter matrices**

$W_i^Q \in \mathbb{R}^{d_{model} \times d_k}, W_i^K \in \mathbb{R}^{d_{model} \times d_k}, W_i^V \in \mathbb{R}^{d_{model} \times d_v}, W^O \in \mathbb{R}^{hd_v \times d_{model}}$

8-head parallel attention layers were employed, $d_k = d_v = d_{model}/h = 64$

→ Total computation cost is similar to single-head attention with full dimensionality due to the reduced dimension of each head

Applications of Attention in our Model

1. Encoder-Decoder attention
    - Queries : previous decoder layer
    - Keys & Values : output of the encoder
    - Every position in the decoder **attend over all positions in the input sequence**
2. Self-attention layers in encoder
    - Keys & Queries & Values : output of the previous layer in the encoder
    - Each position in the encoder can **attend to all positions in the previous layer of the encoder**
3. Self-attention layers in decoder
    - Similar to the self-attention layers in encoder
    - Prevent leftward information flow by **masking out**

Position-wise Feed-Forward Networks

Each of the layers in encoder and decoder contains a **fully connected feed-forward network**

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

↔ Two convolutions with kernel size 1

Dimensionality of input and output is $d_{model} = 512$

Inner-layer has dimensionality $d_{ff} = 2048$

Embeddings and Softmax

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Learned embeddings: **input & output tokens → vectors** (dim : d_{model})

Learned linear transformation and softmax function : **decoder output → next-token probabilities**

Encoder embedding layer, Decoder embedding layer, Pre-softmax linear transformation **shares** the weight matrix

→ Embedding layers, multiply those weights by $\sqrt{d_{model}}$

Positional Encoding

To inject relative (or absolute) position information to the model

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

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

pos : position of the token, i : dimension

→ for any fixed offset k, PE_{pos+k} can be represented as a **linear function** of PE_{pos}

→ Sinusoidal version allow the model to **extrapolate to sequence lengths longer** than the ones encountered during training

Why Self-Attention

1. Total **computational complexity** per layer

- (If sequence length is smaller than the representation dimensionality, i.e. $n < d$)
- $n^2 d < nd^2$, so faster than the RNN based models
- For very long sequences, **restricted self-attention** - only attending r size nbhd - can be considered (Max. path length increases to $O(n/r)$)

2. Amount of computation that can be **parallelized**
    - Connects all positions with a **constant number** of sequentially executed operations
3. **Path length** between long-range dependencies
    - **Shorter path** b/w input and output sequences → easier to learn **long-range dependencies**

Additionally, self-attention could yield **more interpretable models**

Conclusion

Transformer : sequence transduction model based entirely on attention mechanism

We plan to extend the Transformer to problems involving input and output modalities other than text and to investigate local, restricted attention mechanisms to efficiently handle large inputs and outputs such as images, audio and video