**Attention Is All You Need**

Ashish Vswani et al.

**Abstract**

**The dominant sequence transduction models** are based on complex **recurrent** or **convolutional neural networks** that include an **encoder** and a **decoder**.We propose a new simple network architecture, **the Transformer**, **based solely on attention mechanisms**, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being **more parallelizable** and **requiring significantly less time to train**.

**Introduction**

Numerous efforts have since continued to push the boundaries of recurrent language models and encoder-decoder architectures.

**Recurrent models** typically factor computation along the **symbol positions** of the input and output sequences. Aligning the positions to steps in computation time, they generate a sequence of **hidden states** , as a function of the **previous hidden state** and the input for position t. **This inherently sequential nature** **precludes parallelization** within training examples, which becomes **critical at longer sequence lengths**, as **memory constraints limit** batching across examples. Recent work has achieved significant improvements in computational efficiency through **factorization tricks** and **conditional computation**, while also improving model performance in case of the latter. The fundamental constraint of **sequential computation**, however, **remains**.

**Attention mechanisms** have become an integral part of compelling sequence modeling and transduction models in various tasks, **allowing modeling of dependencies** **without regard to their distance** in the input or output sequences.

In this work we propose the **Transformer**, a model architecture eschewing recurrence and instead relying **entirely on an attention mechanism** to draw global dependencies between input and output.

**Background**

The goal of **reducing sequential computation** also forms the foundation of the Extended Neural GPU, ByteNet and ConvS2S, all of which use **CNN** as basic building block, **computing hidden representations in parallel** for all input and output positions. In these models, the number of operations required to relate signals from two arbitrary input or output positions grows in the distance between positions, **linearly** for ConvS2S and **logarithmically** for ByteNet. This makes it more difficult to learn dependencies between **distant positions**. In the **Transformer** this is reduced to a **constant** number of operations, albeit at the cost of reduced effective resolution due to averaging attention-weighted positions, an effect we counteract with **Multi-Head Attention**.

**Self-attention** is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence.

**End-to-end memory networks** are based on a **recurrent attention mechanism** instead of sequence-aligned recurrence and have been shown to perform well on simple-language question answering and langage modeling tasks.

The **Transformer** is the first transduction model **relying entirely on self-attention** to compute representations of its input and output without using sequence-aligned RNNs or Convolution.

**Model Architecture**

Most competitive neural sequence transduction models have an **encoder-decoder structure**. The encoder maps an **input sequence** of symbol representations ( to a sequence of **continuous representations**

. Given , the decoder then generates an **output sequence** of symbols one element **at a time**. At each step the model is **auto-regressive**, consuming the **previously** generated symbols as additional input when generating the next.

The Transformer follows this overall architecture using **stacked self-attention** and **point-wise**, **fully connected layers for both the encoder and decoder**.

A picture containing text, diagram, screenshot, plan

Description automatically generated

**The Transformer – model architecture**

**1. Encoder and Decoder Stacks**

**Encoder**: A stack of **N = 6 identical layers**. Each layer has **two sub-layers**.

* The first is **a multi-head self-attention** mechanism.
* The second is a simple, **position-wise fully connected feed-forward network**.
* Employing **a residual connection** around each of the two sub-layers, followed by **layer normalization**.
* The output of each sub-layer is **LayerNorm**(x + Sublayer(x)), where Sublayer(x) is the function **implemented by the sub-layer itself**.
* **To facilitate these residual connections**, all sub-layers in the models, as well as the embedding layers, produce outputs of **dimension** .

**Decoder**: Composed of a stack of **N = 6 identical layers**. In addition to the **two sub-layers in each encoder layer**, the decoder inserts **a third sub-layer**, which performs **multi-head attention** over the **output of the encoder stack**. Similar to the encoder, we employ **residual connections** around each of the sub-layers, followed by **layer normalization**. We also **modify the self-attention sub-layer** in the decoder stack **to prevent positions from attending to subsequent positions**. This masking, combined with fact that **the output embeddings** are **offset by one position**, ensures that the **predictions** for **position**  can depend only on the known outputs **at positions less than** .

**2. Attention**

An attention function can be described as **mapping a query** and **a set of key-value pairs** **to an output**, where the query, keys, values, and output are all vectors. The output is computed as **a weighted sum of the values**, where the weight assigned to each value is **computed by** **a compatibility function** of the query with the corresponding key.

**Scaled Dot-Product Attention**

The input consists of **queries** and **keys** of **dimension** , and **values** of **dimension** . We compute the **dot products** of the query with all keys, **divide each by** , and apply a **softmax function** to obtain the weights on the values.

The matrix of outputs as:

A picture containing font, text, white, line

Description automatically generated

The two most commonly used attention fuctions are **additive attention**, and **dot-product** (**multiplicative**) **attention**.

* **Dot-product attention** is identical to our algorithm, except for the scaling factor of .
* **Additive attention** computes the compatibility function using a feed-forward network with a single hidden layer.

While the two are similar in theoretical complexity, dot-product attention is **much faster** and **more space-efficient** in practice, since it can be implemented using highly optimized matrix multiplication code.

While **for small values of** the two mechanisms perform **similarly**, **additive attention outperforms** dot product attention **without scaling** for larger values of . To counteract this effect, we **scale the dot products** by .

**Multi-Head Attention**

Instead of performing a single attention function with -dimensional keys, values and queries, we found it **beneficial** **to linearly project the queries, keys and values h times with different**, learned linear projections to , and dimensions, respectively. On each of these projected versions of queries, keys and values we then perform the attention function **in parallel**, yielding -dimensional output values. These are concatenated and once again projected, resulting in the final values, as depicted in Figure 2.

**A picture containing diagram, screenshot, text, plan

Description automatically generated**

**Multi-head attention** allows the model **to jointly attend to information from different representation subspaces at different positions**. With a single attention head, averaging inhibits this.

A picture containing text, font, white, receipt

Description automatically generated

In this work, we employ **h = 8 parallel attention layers**, or heads. For each of these we use = = = 64. **Due to the reduced dimension** of each head, the total computational cost is **similar to** that of **single-head attention** with full dimensionality.

**Applications of Attention in our Model**

The Transformer uses multi-head attention in three different ways:

* In “**encoder-decoder attention**” layers, the **quries** come from **the previous decoder layer**, and the memory **keys and values** come from **the output of the encoder**. This **allows** every position in the **decoder** **to attend over all positions in the input sequence**.
* The **encoder** contains **self-attention layers**. In a self-attention layer **all of the keys, values and queries come from** the same place, in this case, **the 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**.
* Similarly, self-attention layers in the **decoder** allow each position in the decoder to attend to all positions in the decoder up to and including that position. We **need to prevent leftward information flow in the decoder to preserve the auto-regressive property**. We implement this inside of **scaled dot-product attention** by masking out (setting to ) all values in the input of the softmax which correspond to illegal connections.

**3. Position-wise Feed-Forward**

In addition to attention sub-layers, each of the layers in our **encoder** and **decoder** contains **a fully connected feed-forward network**, which is applied to each position **separately** and **identically**. This consists of **two linear transformations** with a **ReLU** activation in between.



While **the linear transformations** are the same across different positions, they use **different parameters** from layer to layer. Another way of describing this is **as two convolutions with kernel size 1**. The dimensionality of input and output is = 512, and the inner-layer has dimensionality = 2048.

**4. Embeddings and Softmax**

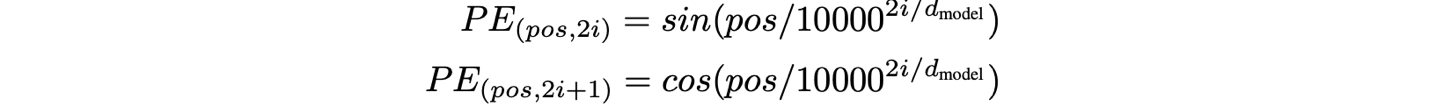
Similarly to other sequence transduction models

* **Using** **learned embeddings** to convert the input tokens and output tokens **to vectors of dimension** .
* **Using** **the usual learned linear transformation** and **softmax function** to convert the decoder output **to predicted next-token probabilities**.
* **Sharing the same weight matrix** between the two embedding layers and the pre-softmax linear transformation.
* **Multiplying** those weights by .

**5. Positional Encoding**

*The Transformer* contains **no recurrence** and **no convolution**, in order for the model to make use of the order of the sequence, we must inject some information about the **relative** or **absolute position of the tokens** in the sequence. To this end, we add “**positional encodings**” to the input embeddings at the bottoms of the encoder and decoder stacks. **The positional encodings** have the same dimension as the embeddings, so that the two can be summed. There are many choices of positional encodings, learned and fixed.

In this work, **sine and cosine functions** of different frequencies are used.



where is the position and is the dimension

That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form **a geometric progression** from to 10000. We chose this function because **we hypothesized it would allow the model to easily learn to attend by relative positions**, since for any fixed offset ,  **can be represented as a linear function of** .

We also experimented with using learned positional embeddings instead, and found that the two versions produced nearly identical results. We chose **the sinusoidal version** because it may allow the model **to extrapolate to sequence lengths longer than the ones encountered during training**.

**Why Self-Attention**

Comparing various aspects of **self-attention layers** to the **recurrent** and **convolutional layers** commonly used for mapping one variable-length sequence of symbol representations to another sequence of equal length , with , such as a hidden layer in a typical sequence transduction encoder or decoder. Motivating our use of self-attention we consider three desiderata.

1. **The total computational complexity** per layer
2. **The amount of computation** that can be **parallelized**, as measured by the minimum number of sequential operations required.
3. **The path length** between long-range dependencies in the network, as measured by the maximum path length between any two input and output positions in networks composed of the different layer types.

A picture containing text, receipt, font, screenshot

Description automatically generated

**Training**

**Results**

**Conclusion**

In this work, the Transformer, the first sequencec transduction model based entirely on attention, replacing the recurrent layers most commonly used in encoder-decoder architectures with multi-headed self-attention.

For translation tasks, the Transformer can be trained significantly faster than architectures based on recurrent or convolutional layers.

**Mentioned Models in this Paper**

**Language modeling & Machine translation**: Recurrent neural networks, long short-term memory and gated recurrent neural networks

**Reference**