

---

# Attention Is All You Need

---

**Ashish Vaswani\***  
Google Brain  
avaswani@google.com

**Noam Shazeer\***  
Google Brain  
noam@google.com

**Niki Parmar\***  
Google Research  
nikip@google.com

**Jakob Uszkoreit\***  
Google Research  
usz@google.com

**Llion Jones\***  
Google Research  
llion@google.com

**Aidan N. Gomez\* †**  
University of Toronto  
aidan@cs.toronto.edu

**Lukasz Kaiser\***  
Google Brain  
lukaszkaiser@google.com

**Illia Polosukhin\* ‡**  
illia.polosukhin@gmail.com

## Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. 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. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

## 1 Introduction

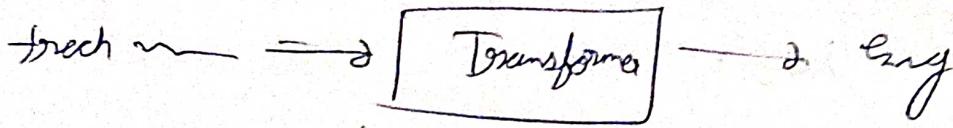
Recurrent neural networks, long short-term memory [12] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and transduction problems such as language modeling and machine translation [29, 2, 5]. Numerous efforts have since continued to push the boundaries of recurrent language models and encoder-decoder architectures [31, 21, 13].

---

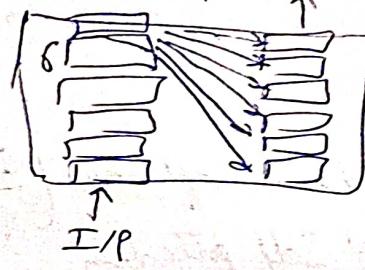
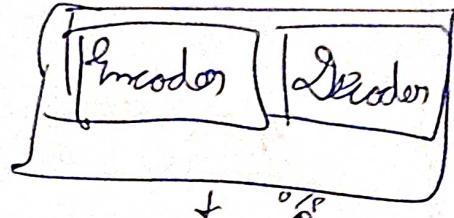
\*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

†Work performed while at Google Brain.

‡Work performed while at Google Research.



black box model



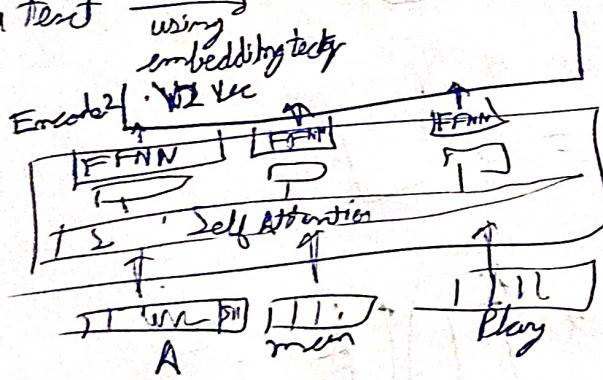
$G \in D$  for best result  
A hyperparameter



for next word

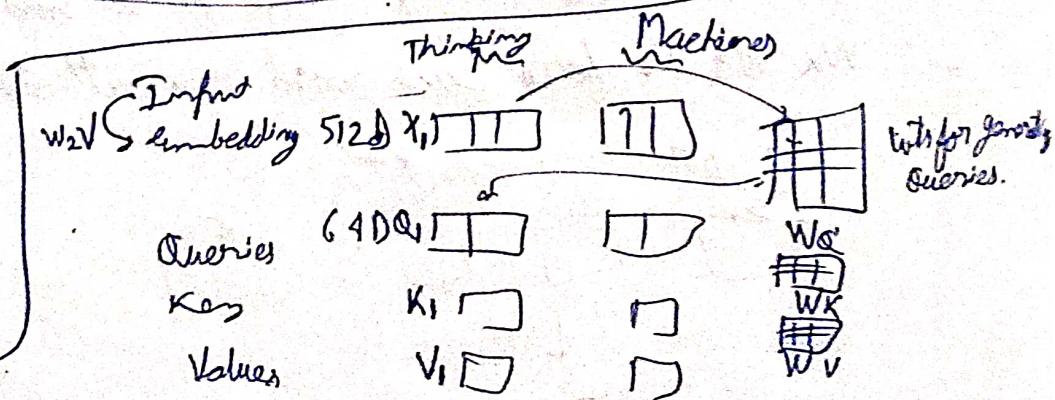
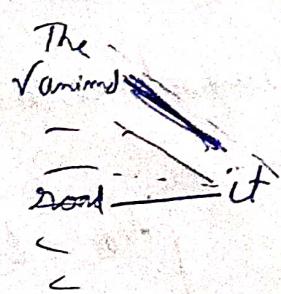
Attention is all you need

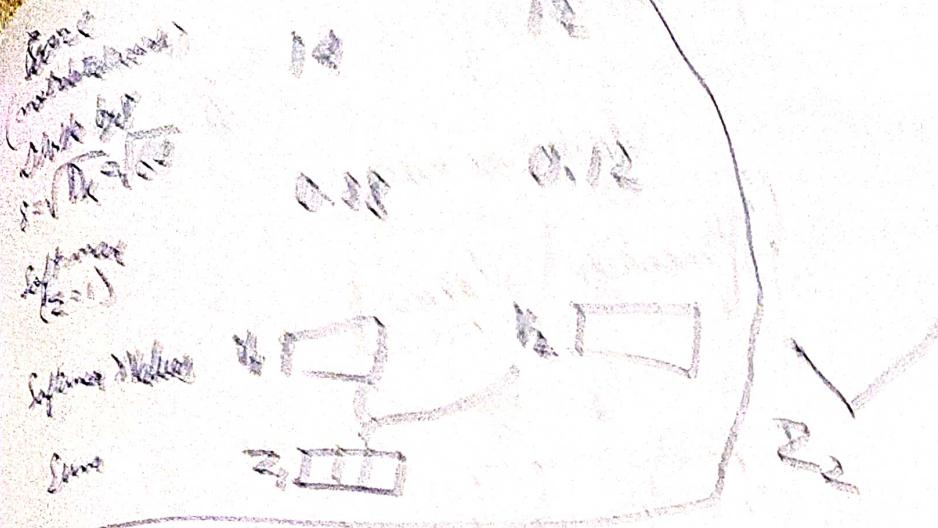
I/P a text converted using embedding tech 512 dimension vector



Self Attention -

The animal didn't cross road as it was too tired





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

Matrix Calculation of Self Attention

$$\begin{matrix} X \\ \begin{smallmatrix} 2 \times d \\ 2 \times 5 \times 2 \end{smallmatrix} \end{matrix} \times \begin{matrix} W_Q \\ \begin{smallmatrix} d \times d \\ 5 \times 2 \end{smallmatrix} \end{matrix} = \begin{matrix} Q \\ \begin{smallmatrix} 2 \times d \\ 2 \times 2 \end{smallmatrix} \end{matrix}$$

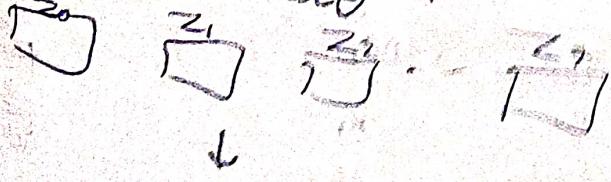
$$\begin{matrix} X \\ \begin{smallmatrix} 2 \times d \\ 2 \times 5 \times 2 \end{smallmatrix} \end{matrix} \times \begin{matrix} W_K \\ \begin{smallmatrix} d \times d \\ 5 \times 2 \end{smallmatrix} \end{matrix} = \begin{matrix} K \\ \begin{smallmatrix} 2 \times d \\ 2 \times 2 \end{smallmatrix} \end{matrix}$$

$$\begin{matrix} X \\ \begin{smallmatrix} 2 \times d \\ 2 \times 5 \times 2 \end{smallmatrix} \end{matrix} \times \begin{matrix} W_V \\ \begin{smallmatrix} d \times d \\ 5 \times 2 \end{smallmatrix} \end{matrix} = \begin{matrix} V \\ \begin{smallmatrix} 2 \times d \\ 2 \times 2 \end{smallmatrix} \end{matrix}$$

$$\text{Softmax} \cdot \left( \frac{Q \times K^T}{\sqrt{dk}} \right) V = Z$$

$W_Q, W_K, W_V \rightarrow$  If same for all word Single head attention  
 $W_Q, W_K, W_V \rightarrow$  for each word  $\rightarrow$  attention importance to each word

8 Attention head good.

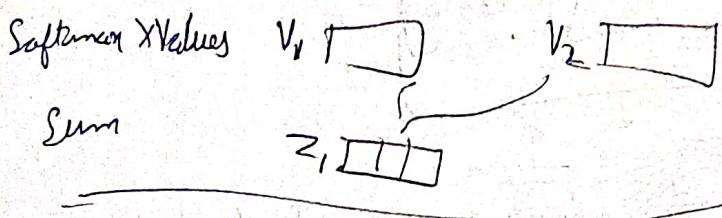


Score  
 (unnormalized score)  
 Divide by 8  
 $8 = \sqrt{d_K} = \sqrt{64}$   
 Softmax  
 $(\sum = 1)$

$q_1 \sim K_1$     $q_1 \not\sim K_2$     $q_2 \sim K_1$     $q_2 \not\sim K_3$

14      12

0.88      0.12



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

Matrix Calculation of Self Attention

$$\underbrace{\begin{matrix} x \\ \vdots \\ x \end{matrix}}_{2 \text{ word}} \times \underbrace{\begin{matrix} w_Q \\ \vdots \\ w_Q \end{matrix}}_{2 \times 5/12} = \underbrace{\begin{matrix} q \\ \vdots \\ q \end{matrix}}_{5/12}$$

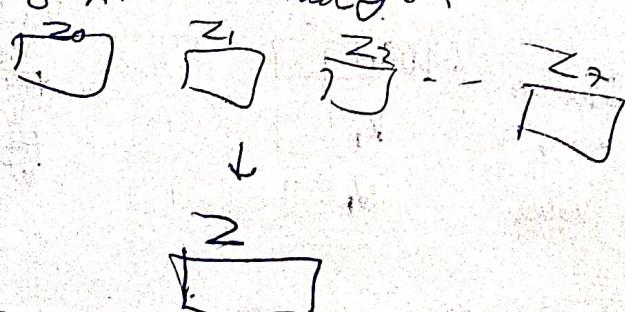
$$\underbrace{\begin{matrix} x \\ \vdots \\ x \end{matrix}}_{2 \text{ word}} \times \underbrace{\begin{matrix} w_K \\ \vdots \\ w_K \end{matrix}}_{2 \times 5/12} = \underbrace{\begin{matrix} k \\ \vdots \\ k \end{matrix}}_{5/12}$$

$$\underbrace{\begin{matrix} x \\ \vdots \\ x \end{matrix}}_{2 \text{ word}} \times \underbrace{\begin{matrix} w_V \\ \vdots \\ w_V \end{matrix}}_{2 \times 5/12} = \underbrace{\begin{matrix} v \\ \vdots \\ v \end{matrix}}_{5/12}$$

Softmax

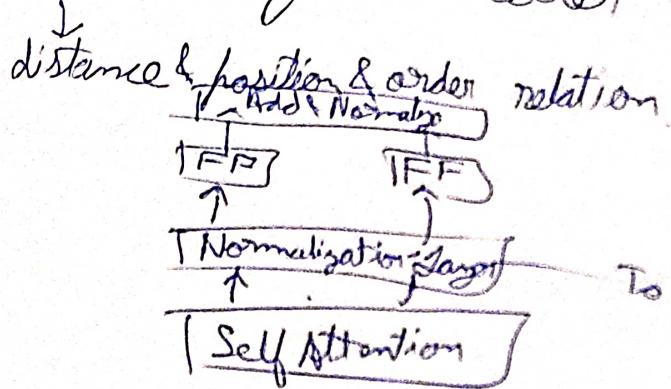
$$\left( \underbrace{\begin{matrix} q \\ \vdots \\ q \end{matrix}}_{5/12} \times \underbrace{\begin{matrix} k \\ \vdots \\ k \end{matrix}}_{5/12}^T \right) \underbrace{\begin{matrix} v \\ \vdots \\ v \end{matrix}}_{5/12} = \underbrace{\begin{matrix} z \\ \vdots \\ z \end{matrix}}_{5/12}$$

$w_K$     $w_V$     $w_Q \rightarrow$  If same for all word Single head attending  
 $w_K$     $w_V$     $w_Q$  for each word  $\rightarrow$  attention/importance to each word  
 8 Attention head good.



$w_O \rightarrow$  to combine

I/P  $\rightarrow$  Positional encoding  $\rightarrow$  Encoder



To skip layers FFNN or SA is not used

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  $h_t$ , as a function of the previous hidden state  $h_{t-1}$  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 [18] and conditional computation [26], 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 [2, 16]. In all but a few cases [22], however, such attention mechanisms are used in conjunction with a recurrent network.

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. The Transformer allows for significantly more parallelization and can reach a new state of the art in translation quality after being trained for as little as twelve hours on eight P100 GPUs.

## 2 Background

The goal of reducing sequential computation also forms the foundation of the Extended Neural GPU [20], ByteNet [15] and ConvS2S [8], all of which use convolutional neural networks 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 [11]. 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 as described in section 3.2.

Self-attention, sometimes called intra-attention is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence. Self-attention has been used successfully in a variety of tasks including reading comprehension, abstractive summarization, textual entailment and learning task-independent sentence representations [4, 22, 23, 19].

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 language modeling tasks [28].

To the best of our knowledge, however, 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. In the following sections, we will describe the Transformer, motivate self-attention and discuss its advantages over models such as [14, 15] and [8].

## 3 Model Architecture

Most competitive neural sequence transduction models have an encoder-decoder structure [5, 2, 29]. Here, the encoder maps an input sequence of symbol representations  $(x_1, \dots, x_n)$  to a sequence of continuous representations  $\mathbf{z} = (z_1, \dots, z_n)$ . Given  $\mathbf{z}$ , the decoder then generates an output sequence  $(y_1, \dots, y_m)$  of symbols one element at a time. At each step the model is auto-regressive [9], 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, shown in the left and right halves of Figure 1, respectively.

### 3.1 Encoder and Decoder Stacks

**Encoder:** The encoder is composed of a stack of  $N = 6$  identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-

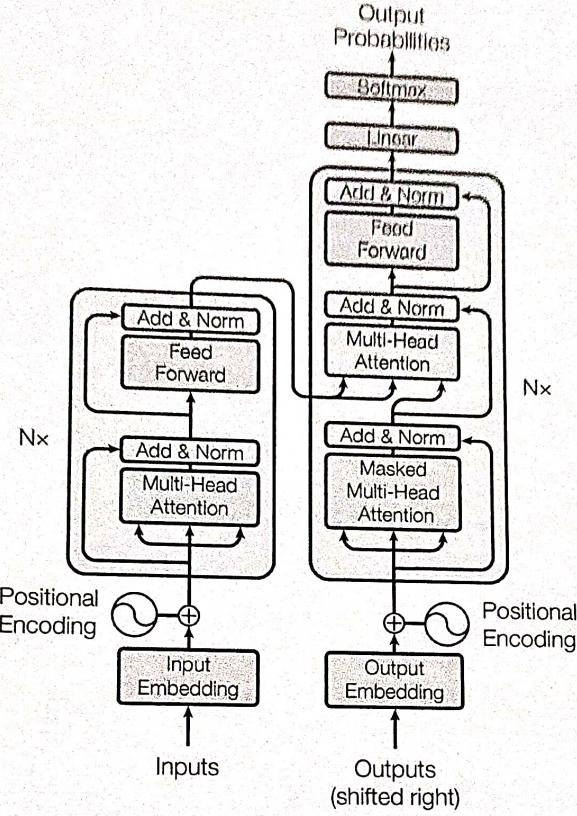


Figure 1: The Transformer - model architecture.

wise fully connected feed-forward network. We employ a residual connection [10] around each of the two sub-layers, followed by layer normalization [1]. That is, the output of each sub-layer is  $\text{LayerNorm}(x + \text{Sublayer}(x))$ , where  $\text{Sublayer}(x)$  is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension  $d_{\text{model}} = 512$ .

**Decoder:** The decoder is also 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  $i$  can depend only on the known outputs at positions less than  $i$ .

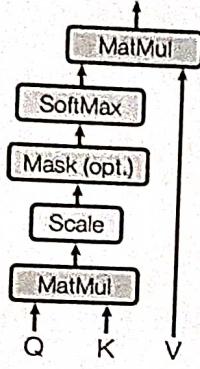
### 3.2 Attention

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.

#### 3.2.1 Scaled Dot-Product Attention

We call our particular attention "Scaled Dot-Product Attention" (Figure 2). The input consists of queries and keys of dimension  $d_k$ , and values of dimension  $d_v$ . We compute the dot products of the

Scaled Dot-Product Attention



Multi-Head Attention

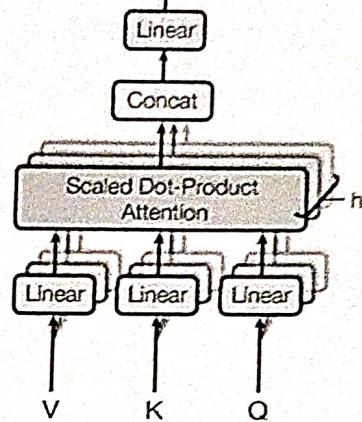


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

query with all keys, divide each by  $\sqrt{d_k}$ , and apply a softmax function to obtain the weights on the values.

In practice, we compute the attention function on a set of queries simultaneously, packed together into a matrix  $Q$ . The keys and values are also packed together into matrices  $K$  and  $V$ . We compute the matrix of outputs as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (1)$$

The two most commonly used attention functions are additive attention [2], and dot-product (multiplicative) attention. Dot-product attention is identical to our algorithm, except for the scaling factor of  $\frac{1}{\sqrt{d_k}}$ . 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  $d_k$  the two mechanisms perform similarly, additive attention outperforms dot product attention without scaling for larger values of  $d_k$  [3]. We suspect that for large values of  $d_k$ , the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients<sup>4</sup>. To counteract this effect, we scale the dot products by  $\frac{1}{\sqrt{d_k}}$ .

### 3.2.2 Multi-Head Attention

Instead of performing a single attention function with  $d_{\text{model}}$ -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  $d_k$ ,  $d_k$  and  $d_v$  dimensions, respectively. On each of these projected versions of queries, keys and values we then perform the attention function in parallel, yielding  $d_v$ -dimensional output values. These are concatenated and once again projected, resulting in the final values, as depicted in Figure 2.

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.

<sup>4</sup>To illustrate why the dot products get large, assume that the components of  $q$  and  $k$  are independent random variables with mean 0 and variance 1. Then their dot product,  $q \cdot k = \sum_{i=1}^{d_k} q_i k_i$ , has mean 0 and variance  $d_k$ .

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

where  $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

Where the projections are parameter matrices  $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$ ,  $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$ ,  $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$  and  $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$ .

In this work we employ  $h = 8$  parallel attention layers, or heads. For each of these we use  $d_k = d_v = d_{\text{model}}/h = 64$ . Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.

### 3.2.3 Applications of Attention in our Model

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

- In "encoder-decoder attention" layers, the queries 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. This mimics the typical encoder-decoder attention mechanisms in sequence-to-sequence models such as [31, 2, 8].
- 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  $-\infty$ ) all values in the input of the softmax which correspond to illegal connections. See Figure 2.

### 3.3 Position-wise Feed-Forward Networks

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.

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \quad (2)$$

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  $d_{\text{model}} = 512$ , and the inner-layer has dimensionality  $d_{ff} = 2048$ .

### 3.4 Embeddings and Softmax

Similarly to other sequence transduction models, we use learned embeddings to convert the input tokens and output tokens to vectors of dimension  $d_{\text{model}}$ . We also use the usual learned linear transformation and softmax function to convert the decoder output to predicted next-token probabilities. In our model, we share the same weight matrix between the two embedding layers and the pre-softmax linear transformation, similar to [24]. In the embedding layers, we multiply those weights by  $\sqrt{d_{\text{model}}}$ .

### 3.5 Positional Encoding

Since our model 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

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)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

bottoms of the encoder and decoder stacks. The positional encodings have the same dimension  $d_{\text{model}}$  as the embeddings, so that the two can be summed. There are many choices of positional encodings, learned and fixed [8].

In this work, we use sine and cosine functions of different frequencies:

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

where  $pos$  is the position and  $i$  is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from  $2\pi$  to  $10000 \cdot 2\pi$ . 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  $k$ ,  $PE_{pos+k}$  can be represented as a linear function of  $PE_{pos}$ .

We also experimented with using learned positional embeddings [8] instead, and found that the two versions produced nearly identical results (see Table 3 row (E)). We chose the sinusoidal version because it may allow the model to extrapolate to sequence lengths longer than the ones encountered during training.

## 4 Why Self-Attention

In this section we compare various aspects of self-attention layers to the recurrent and convolutional layers commonly used for mapping one variable-length sequence of symbol representations  $(x_1, \dots, x_n)$  to another sequence of equal length  $(z_1, \dots, z_n)$ , with  $x_i, z_i \in \mathbb{R}^d$ , such as a hidden layer in a typical sequence transduction encoder or decoder. Motivating our use of self-attention we consider three desiderata.

One is the total computational complexity per layer. Another is the amount of computation that can be parallelized, as measured by the minimum number of sequential operations required.

The third is the path length between long-range dependencies in the network. Learning long-range dependencies is a key challenge in many sequence transduction tasks. One key factor affecting the ability to learn such dependencies is the length of the paths forward and backward signals have to traverse in the network. The shorter these paths between any combination of positions in the input and output sequences, the easier it is to learn long-range dependencies [11]. Hence we also compare the maximum path length between any two input and output positions in networks composed of the different layer types.

As noted in Table 1, a self-attention layer connects all positions with a constant number of sequentially executed operations, whereas a recurrent layer requires  $O(n)$  sequential operations. In terms of computational complexity, self-attention layers are faster than recurrent layers when the sequence length  $n$  is smaller than the representation dimensionality  $d$ , which is most often the case with sentence representations used by state-of-the-art models in machine translations, such as word-piece [31] and byte-pair [25] representations. To improve computational performance for tasks involving very long sequences, self-attention could be restricted to considering only a neighborhood of size  $r$  in

Table 3: Variations on the Transformer architecture. Unlisted values are identical to those of the base model. All metrics are on the English-to-German translation development set, newstest2013. Listed perplexities are per-wordpiece, according to our byte-pair encoding, and should not be compared to per-word perplexities.

	$N$	$d_{\text{model}}$	$d_{\text{ff}}$	$h$	$d_k$	$d_v$	$P_{\text{drop}}$	$\epsilon_{ls}$	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
										5.01	25.4	60
(C)				2						6.11	23.7	36
				4						5.19	25.3	50
				8						4.88	25.5	80
				16	256	32	32			5.75	24.5	28
				32		128	128			4.66	26.0	168
				64		1024	1024			5.12	25.4	53
				128		4096	4096			4.75	26.2	90
(D)								0.0		5.77	24.6	
								0.2		4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3	300K		<b>4.33</b>	<b>26.4</b>	213

In Table 3 rows (B), we observe that reducing the attention key size  $d_k$  hurts model quality. This suggests that determining compatibility is not easy and that a more sophisticated compatibility function than dot product may be beneficial. We further observe in rows (C) and (D) that, as expected, bigger models are better, and dropout is very helpful in avoiding over-fitting. In row (E) we replace our sinusoidal positional encoding with learned positional embeddings [8], and observe nearly identical results to the base model.

## 7 Conclusion

In this work, we presented the Transformer, the first sequence 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. On both WMT 2014 English-to-German and WMT 2014 English-to-French translation tasks, we achieve a new state of the art. In the former task our best model outperforms even all previously reported ensembles.

We are excited about the future of attention-based models and plan to apply them to other tasks. 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. Making generation less sequential is another research goals of ours.

The code we used to train and evaluate our models is available at <https://github.com/tensorflow/tensor2tensor>.

**Acknowledgements** We are grateful to Nal Kalchbrenner and Stephan Gouws for their fruitful comments, corrections and inspiration.