Transformer

Vaswani et al. "Attention is all you need." (2017).

2021 DIYA Time Series Team
Yejoon Lee

First draft on Sept 19, 2021
Revised on Apr 24, 2022

Why Transformer?

- Large-scale Language Models (BERT, GPT)
- Computer Vision (Vision Transformer)
- Reinforcement Learning (AlphaStar)

https://www.rottentomatoes.com/m/transformers_the_movie

https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii#block-8

Quick Review

- Language Model
- Seq2seq
- RNN Encoder-decoder (Cho et al., 2014)
- RNN Encoder-decoder with attention (Bahdanau et al., 2014)

Language Model

- At each timestep t, the output of the language model is the conditional distribution

$$p(x_t|x_{t-1}, \dots, x_1)$$

- Then we can compute the probability of sequence $\mathbf{x} = (x_1, \dots, x_T)$ using

$$p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t|x_{t-1}, \dots, x_1)$$

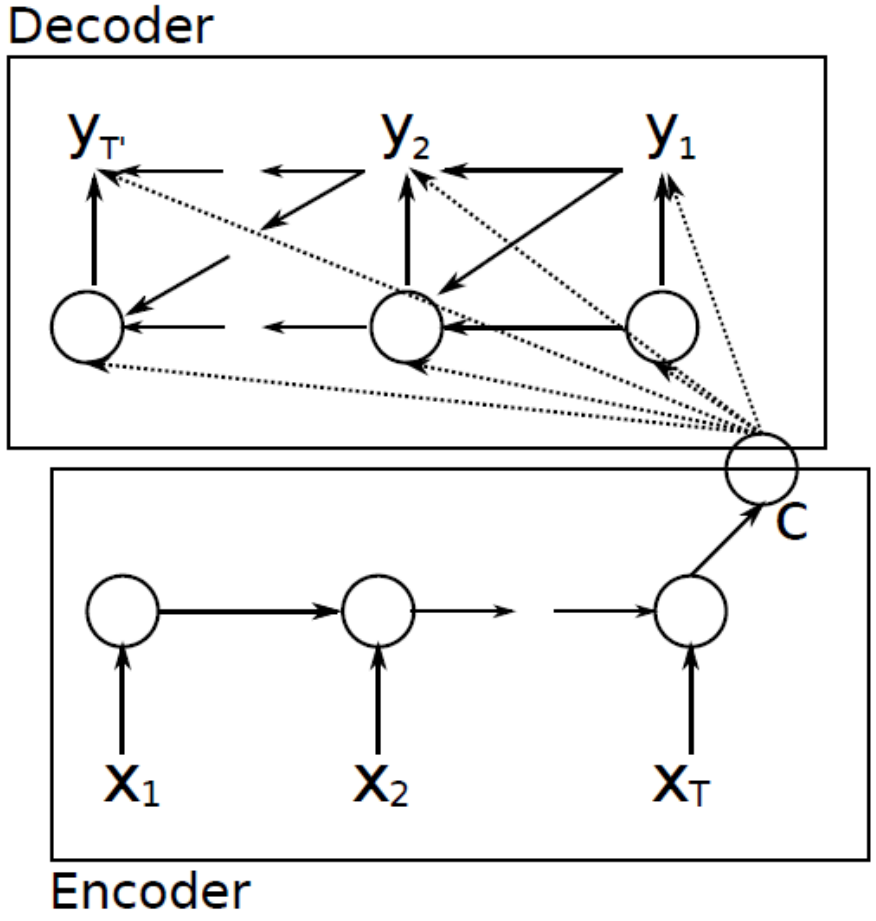
Motivation of Seq2seq

Variable-length input and output

"DNNs can only be applied to problem whose inputs and target can be sensibly encoded with vectors and fixed dimensionality. It is a significant limitation, since many important problems are best expressed with sequences whose lengths are not known a-priori. For example, speech recognition and machine translation are sequential problems."

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." (2014).

Quick review of RNN Encoder-decoder



- Encoder is a simple RNN.
- Decoder is RNN with a fixed-length context vetor *c* :

$$\mathbf{h}_{\langle t \rangle} = f\left(\mathbf{h}_{\langle t-1 \rangle}, y_{t-1}, \mathbf{c}\right),$$

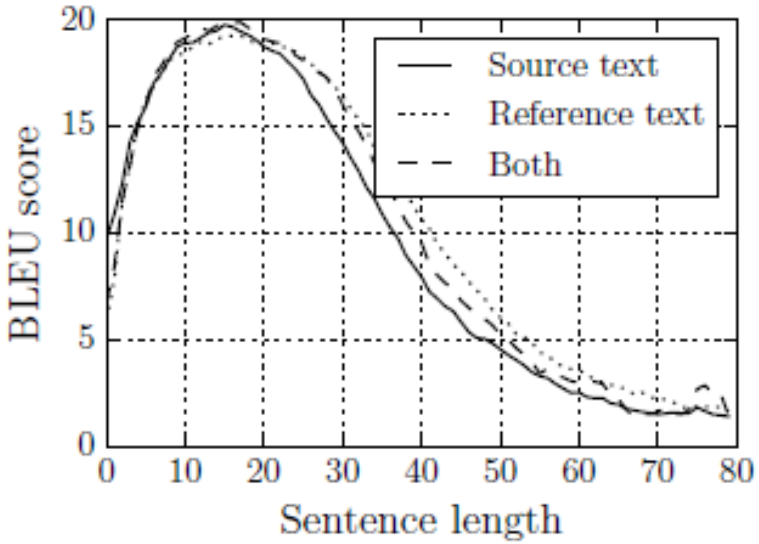
$$P(y_t | y_{t-1}, y_{t-2}, \ldots, y_1, \mathbf{c}) = g\left(\mathbf{h}_{\langle t \rangle}, y_{t-1}, \mathbf{c}\right).$$

where $\mathbf{h}_{<t>}$ is the hidden state of the decoder at time **t**.

Cho, Kyunghyun, et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation." (2014).

Quick review of RNN Encoder-decoder

Limit: Long Sentence



(a) RNNenc

Cho, Kyunghyun, et al. "On the properties of neural machine translation: Encoder-decoder approaches.“ (2014).

Quick review of RNN Encoder-decoder with attention

Just add subscript i to context vector c

Decoder Structure

Vanilla RNN Encoder-Decoder
(Cho et al. 2014.)

$$\mathbf{h}_{\langle t \rangle} = f\left(\mathbf{h}_{\langle t-1 \rangle}, y_{t-1}, \mathbf{c}\right),$$

$$P(y_t|y_{t-1}, y_{t-2}, \ldots, y_1, \mathbf{c}) = g\left(\mathbf{h}_{\langle t \rangle}, y_{t-1}, \mathbf{c}\right).$$

RNN Encoder-Decoder with attention
(Bahdanau et al. 2014.)

$$s_i = f(s_{i-1}, y_{i-1}, c_i).$$

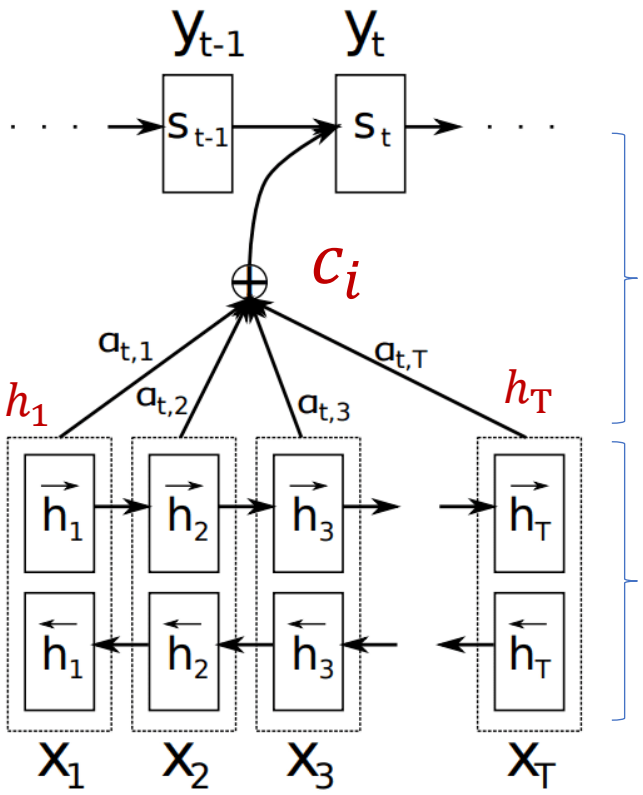
$$p(y_i|y_1, \ldots, y_{i-1}, \mathbf{x}) = g(y_{i-1}, s_i, c_i),$$

Single context vector c for all target word y_i

Distinct context vector c_i for every target word y_i

Cho, Kyunghyun, et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation." (2014).
Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." (2014).

Quick review of RNN Encoder-decoder with attention



Decoder (index i)

Encoder (index j)

Annotation not from the paper marked red

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j.$$

Context vector c_i is a weighted sum of annotation h_j.

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})},$$

Ensure that $\sum_j \alpha_{ij} = 1$ by softmax.

$$e_{ij} = a(s_{i-1}, h_j)$$

Alignment model a scores how well the inputs around position j and output at position i match.

- Annotation could be understood as a hidden state of encoder RNN though this is not precisely true.
- **Score** refers to either a or e_{ij}, depending on the context.
- Alignment model a is a feedforward neural network which is jointly trained (i.e., additive style). This only holds true in this particular attention model(i.e., Bahdanau attention).
- (Why s_{i-1} instead of s_i? s_i cannot be known when calculating e_{ij}.)

Quick review of RNN Encoder-decoder with attention

Why is this effective?

"By letting the decoder have an attention mechanism, we relieve the encoder from the burden of having to encode all information in the source sentence into a fixed-length vector.

With this new approach the information can be spread throughout the sequence of annotations, which can be selectively retrieved by the decoder accordingly."

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." (2014).

The Transformer

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

Łukasz 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 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

~~RNN(CNN)~~

+

Attention

"… the first sequence transduction model based entirely on attention, replacing the recurrent layers…"

Vaswani, Ashish, et al. "Attention is all you need." (2017).

Query? Key? Value?

You are not the only one!!

What exactly are keys, queries, and values in attention mechanisms?

Asked 1 year, 10 months ago Active today Viewed 43k times

114

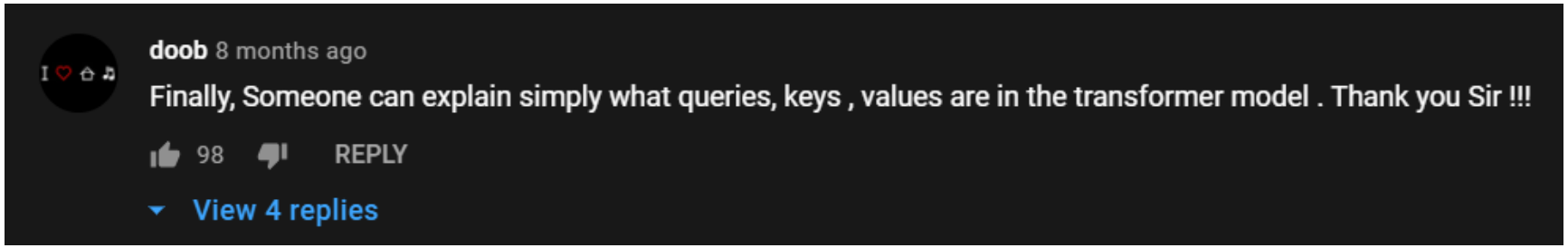
How should one understand the keys, queries, and values that are often mentioned in attention mechanisms?

I've tried searching online, but all the resources I find only speak of them as if the reader already knows what they are.

Featured on

Commu

3-vote



The Transformer from "Attention is All You Need" has been on a lot of people's minds over the last year. Besides producing major improvements in translation quality, it provides a new architecture for many other NLP tasks. The paper itself is very clearly written, but the conventional wisdom has been that it is quite difficult to implement correctly.

Query? Key? Value?

- Why do these feel so absurd at first?

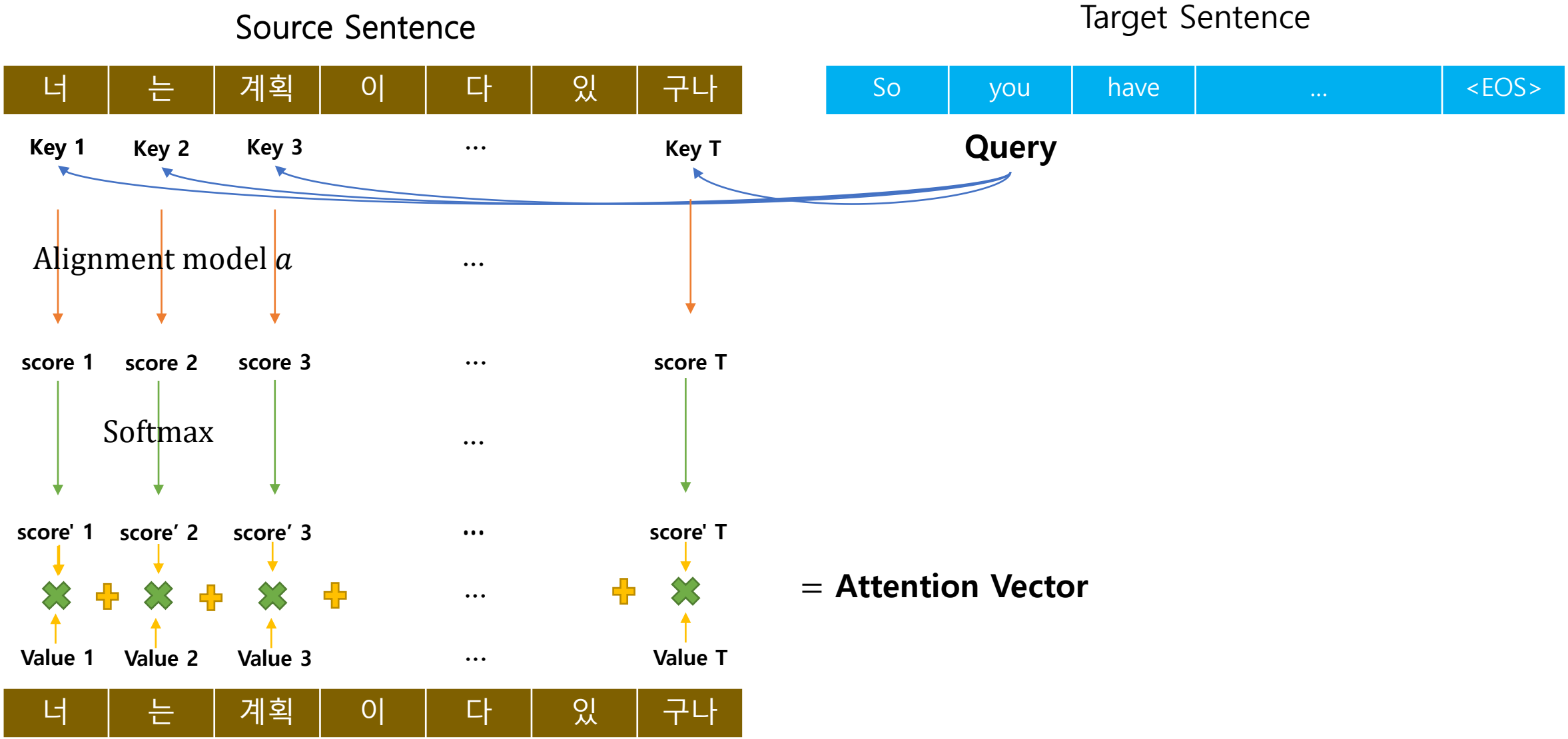
1. "Attention is all you need" assumes that readers already know.

2. Query, key, values are **conceptually different**, but **could be same (and for many cases they are) in implementation.** (K=V or Q=K=V; latter only possible in self-attention)

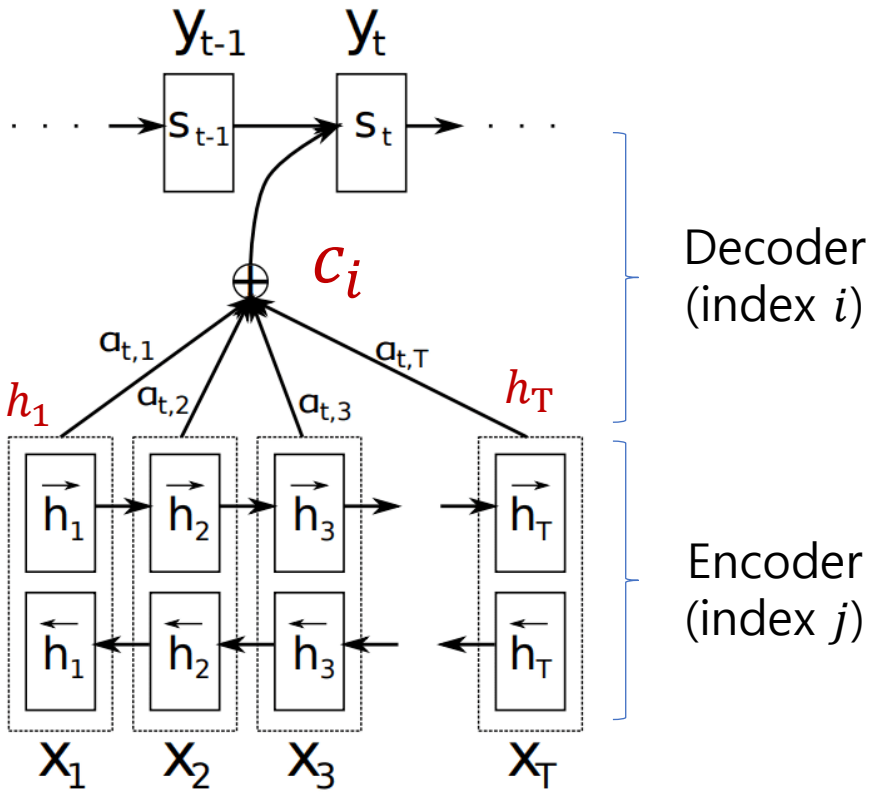
- Query, key, value can be used to describe any general attention mechanism, not only the transformer.

Query? Key? Value?

* Vectors are capitalized, scalars are lowercased.



Query? Key? Value? – apply to Bahdanau Attention



Annotation not from the paper marked red

*This is the case where key = value.

Value

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j.$$

Context vector c_i is a weighted sum of annotation h_j.

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})},$$

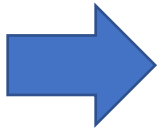
Ensure that $\sum_j \alpha_{ij} = 1$ by softmax.

$$e_{ij} = a(s_{i-1}, h_j)$$

Alignment model a scores how well the inputs around position j and output at position i match.

Query Key

Inside the Transformer



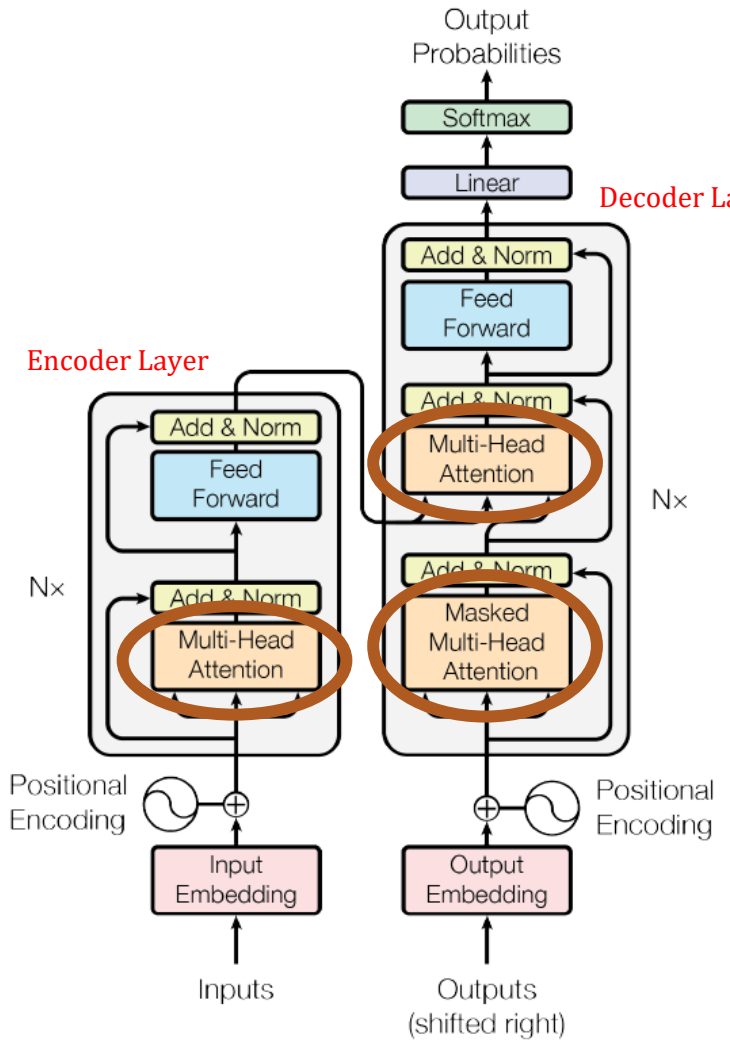


Figure 1: The Transformer - model architecture.

Annotation not from the paper marked red

Scaled Dot-Product Attention

There are attention(alignment) styles other than Bahdanau's.

$$\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \bar{\boldsymbol{h}}_s & \textit{dot} \\ \boldsymbol{h}_t^\top \boldsymbol{W}_a \bar{\boldsymbol{h}}_s & \textit{general} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W}_a[\boldsymbol{h}_t; \bar{\boldsymbol{h}}_s]\right) & \textit{concat} \end{cases}$$

The Transformer

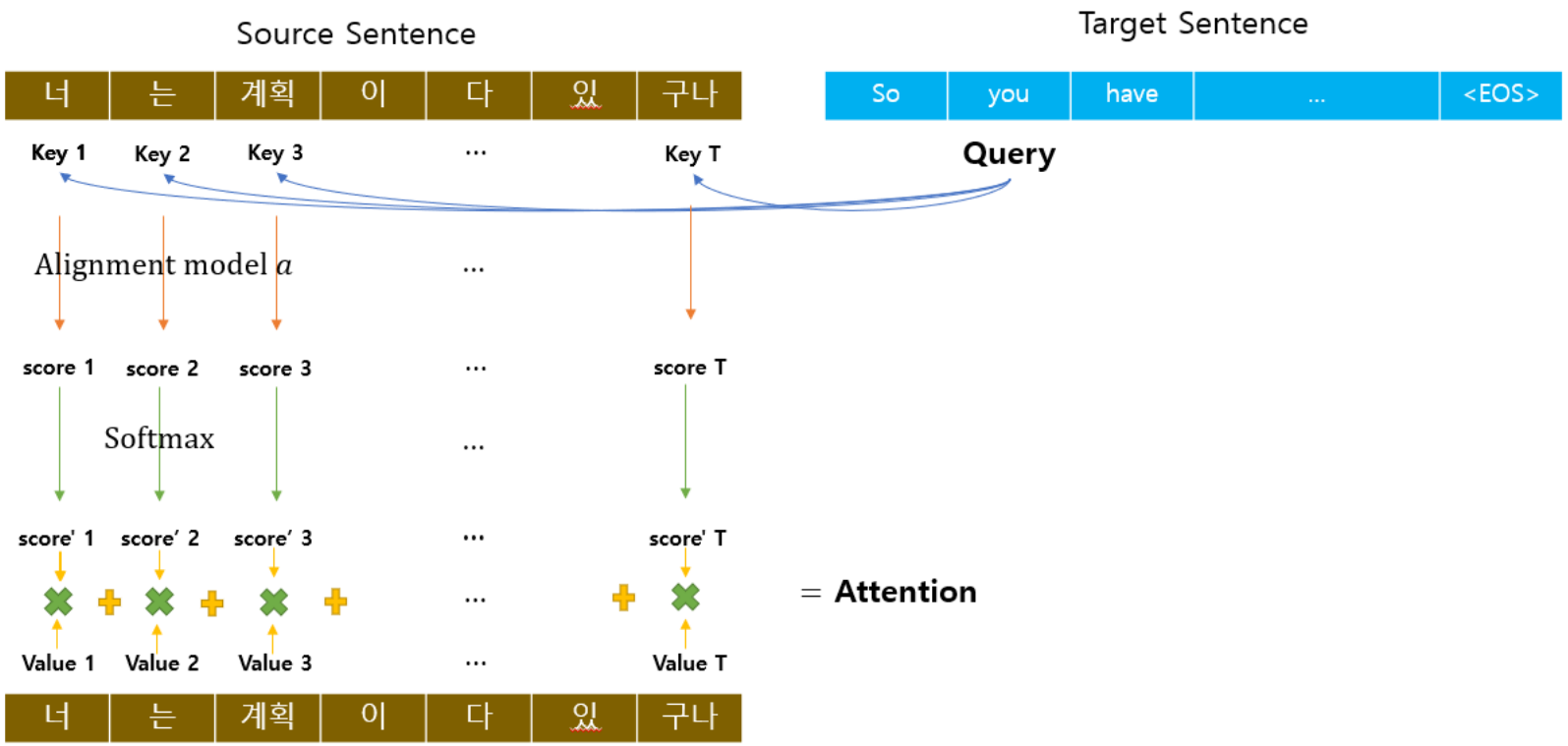
Bahdanau's style

Luong, Minh-Thang, Hieu Pham, and Christopher D. Manning. "Effective approaches to attention-based neural machine translation." (2015).

Scaled Dot-Product Attention

Alignment model

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



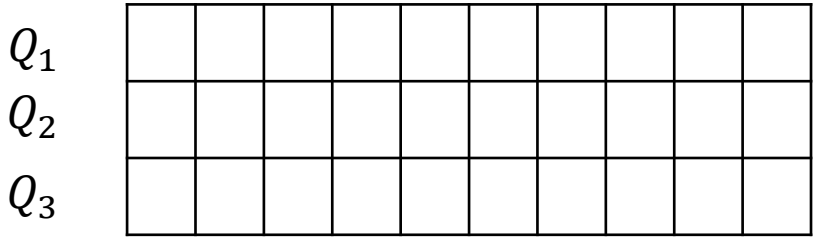
$$Q \in \mathbb{R}^{l_{tar} \times d_{model}}$$
$$K \in \mathbb{R}^{l_{src} \times d_{model}}$$
$$V \in \mathbb{R}^{l_{src} \times d_{model}}$$

where l_{src} is the length of source setence,
l_{tar} is the length of target sentence,
d_{model} is the embedding dimension.

Scaled Dot-Product Attention

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

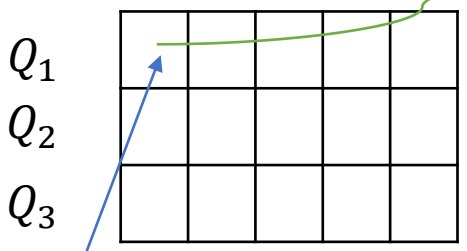
Q



* *Assume* $l_{tar} = 3, l_{src} = 5, d_{model} = 10$

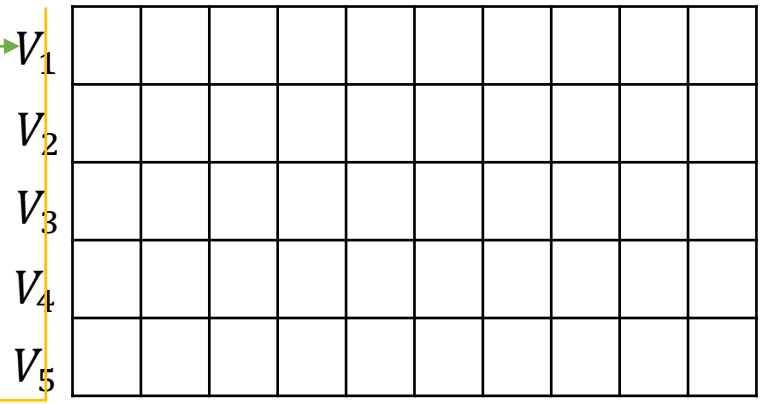
QK^T

(Attention plot)



Given Q_1, Weight of V_1

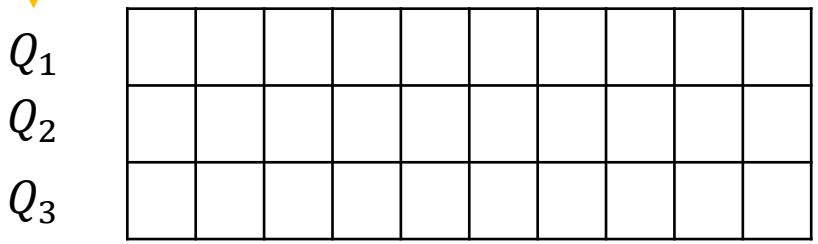
V



Alignment score between Q_1 *and* K_1

Weighted sum of V

$Attention(Q, K, V)$



$$Q \in \mathbb{R}^{l_{tar} \times d_{model}}$$
$$K \in \mathbb{R}^{l_{src} \times d_{model}}$$
$$V \in \mathbb{R}^{l_{src} \times d_{model}}$$

where l_{src} *is the length of source setence,* l_{tar} *is the length of target sentence,* d_{model} *is the embedding dimension.*

$$Attention(Q, K, V) \in \mathbb{R}^{l_{tar} \times d_{model}}$$

Shape of query is preserved.

=> Result is better embedding.

Different Notation: Single projection, then divide

$$QW^Q = [QW_1^Q, QW_2^Q, \ldots, QW_h^Q]$$

Multi-Head Attention

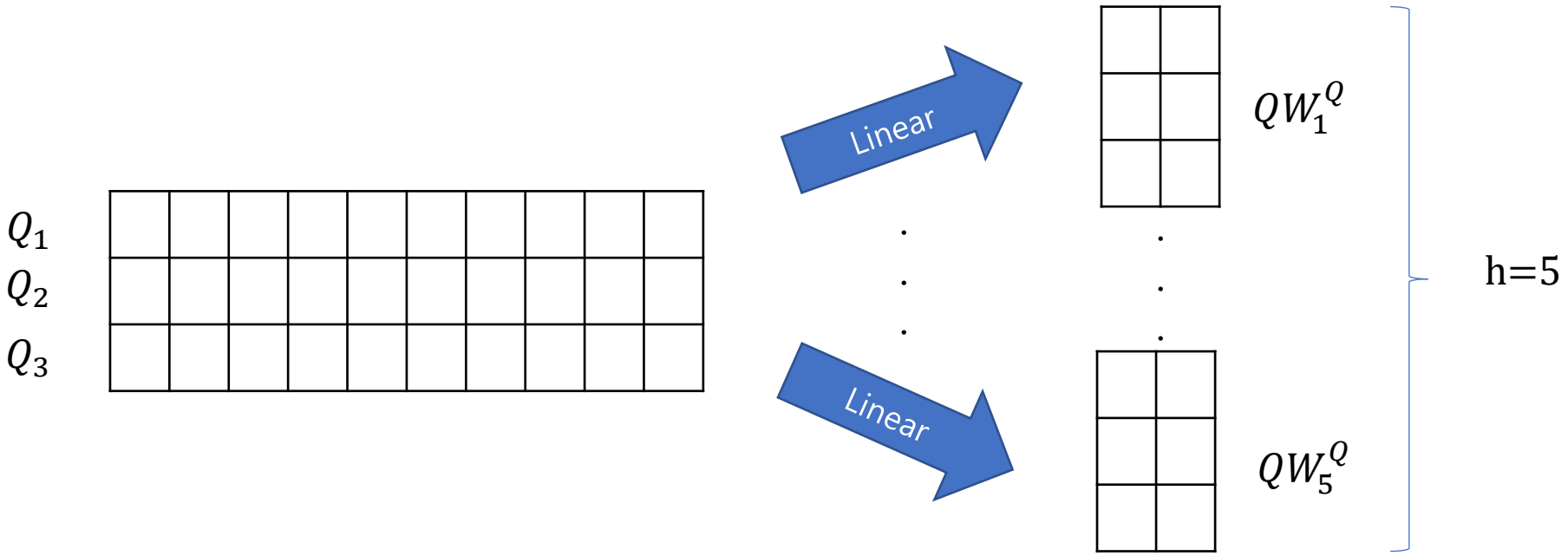
$$Q \in \mathbb{R}^{l_{tar} \times d_{model}}$$
$$K \in \mathbb{R}^{l_{src} \times d_{model}}$$
$$V \in \mathbb{R}^{l_{src} \times d_{model}}$$

where l_{src} is the length of source setence, l_{tar} is the length of target sentence, d_{model} is the embedding dimension.

Linear

$$QW_i^Q \in \mathbb{R}^{l_{tar} \times d_k}$$
$$KW_i^K \in \mathbb{R}^{l_{src} \times d_k}$$
$$VW_i^V \in \mathbb{R}^{l_{src} \times d_v}$$

where $W_i^{Q,K,V}$ are learnable projections at head i, d_k and d_v is usually set to d_{model}/h , where h is a number of heads .



Multi-Head Attention

Different Notation: Project separately, then sum

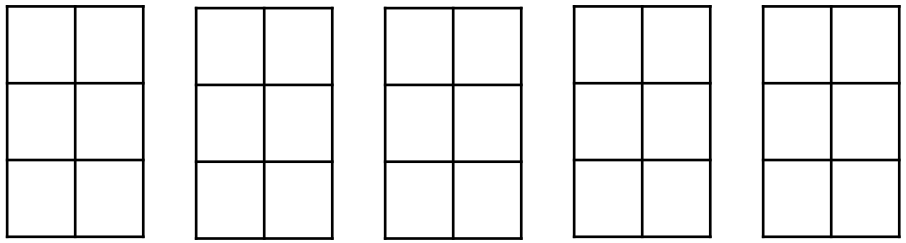
$$\text{MultiHead}(Q,K,V) = \sum_{i=0}^{h} \text{head}_i W_i^O$$

$$where\ W_i^O \in \mathbb{R}^{d_v \times d_{model}}$$

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h}) W^O$$

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

h=5

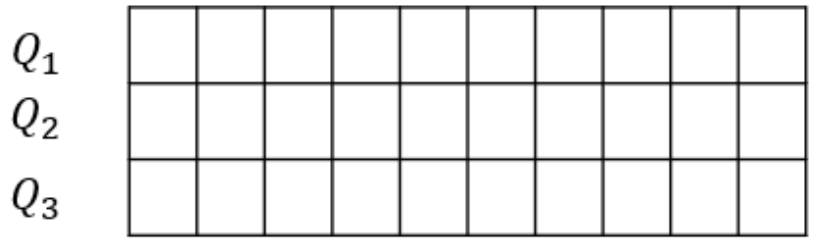


$head_1$ $head_2$ ... $head_5$

(After attention)

Concat

Linear



$MultiHead(Q,K,V) \in \mathbb{R}^{l_{tar} \times d_{model}}$

Shape of query is preserved.

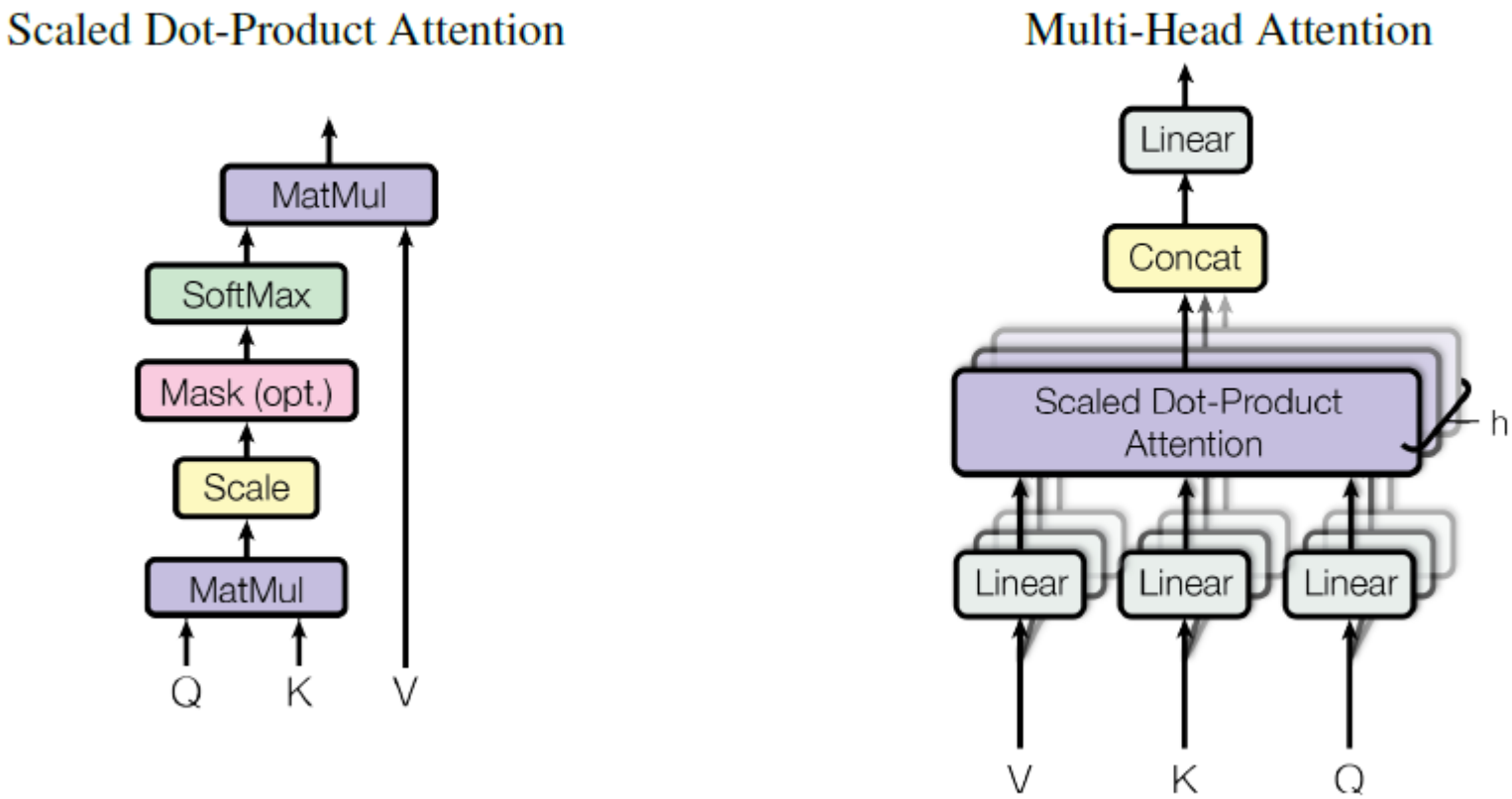
=> Result is better embedding.

Why Multi-Head?

"Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions."

Vaswani, Ashish, et al. "Attention is all you need." (2017).

Recap



Vaswani, Ashish, et al. "Attention is all you need." (2017). (Transformer)

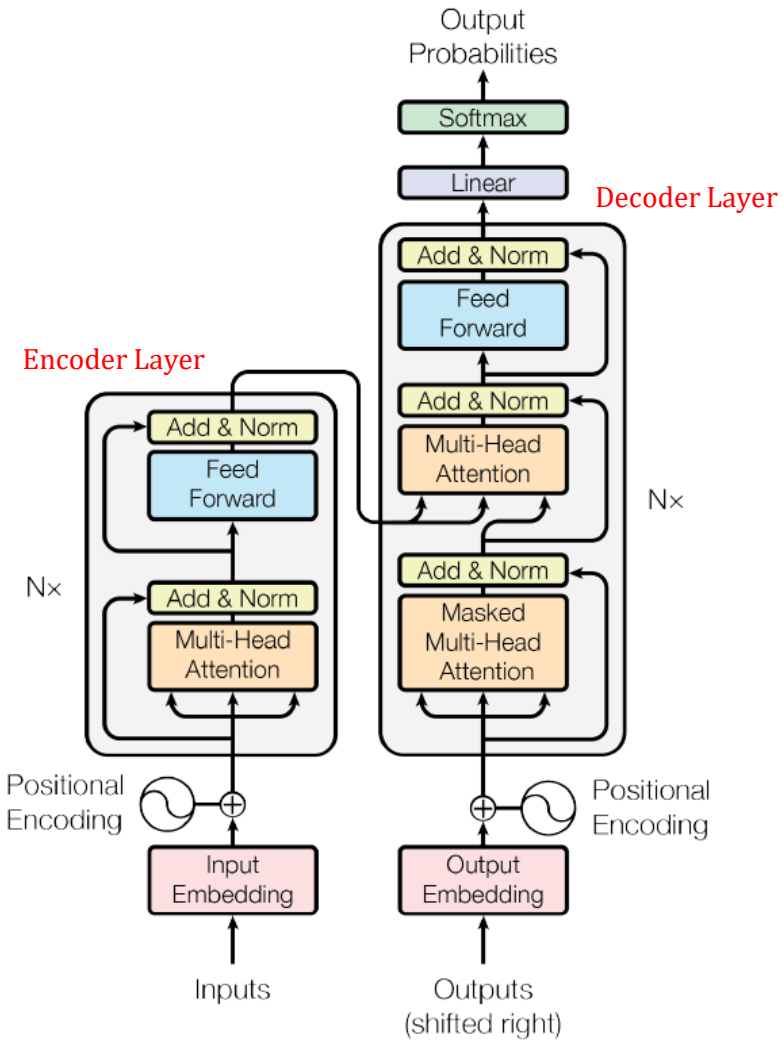


Figure 1: The Transformer - model architecture.

Annotation not from the paper marked red

Encoder(decoder) =
stacked encoder(decoder) layer

Three different types of attention in Transformer

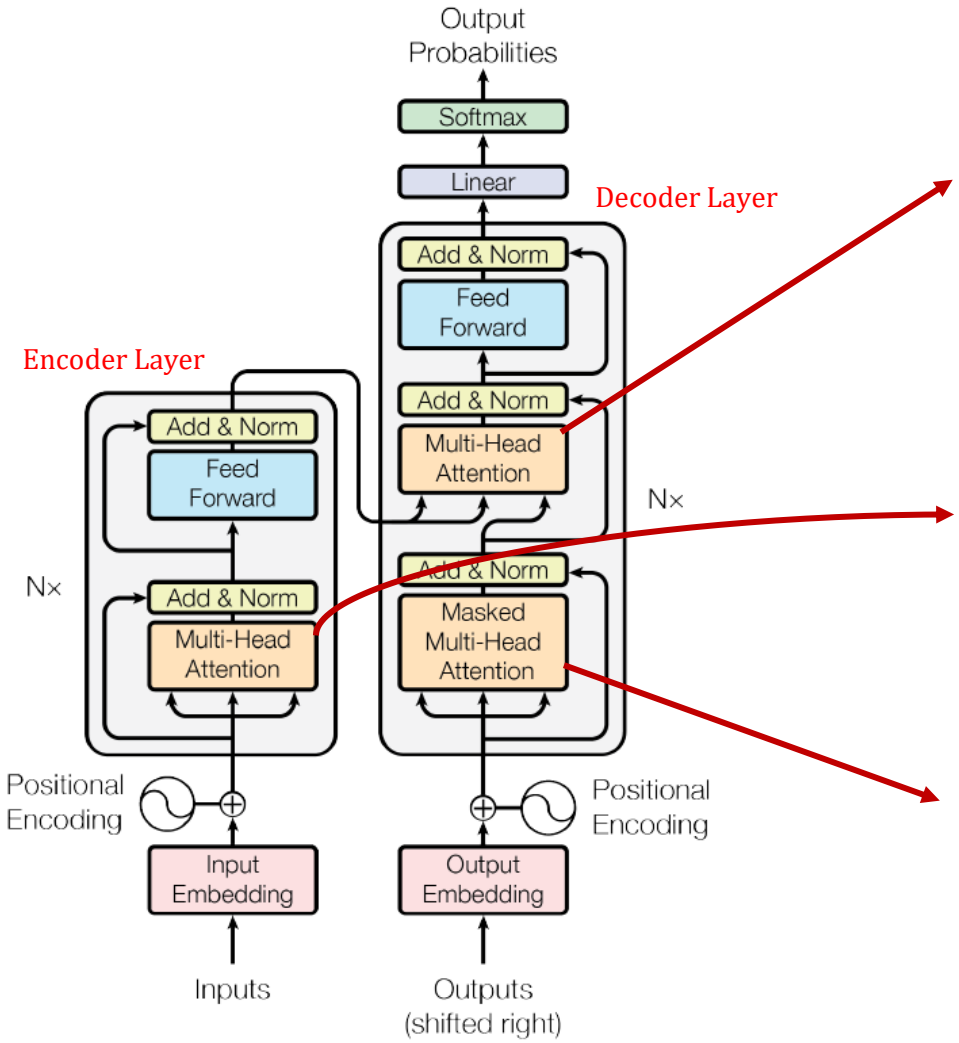


Figure 1: The Transformer - model architecture.

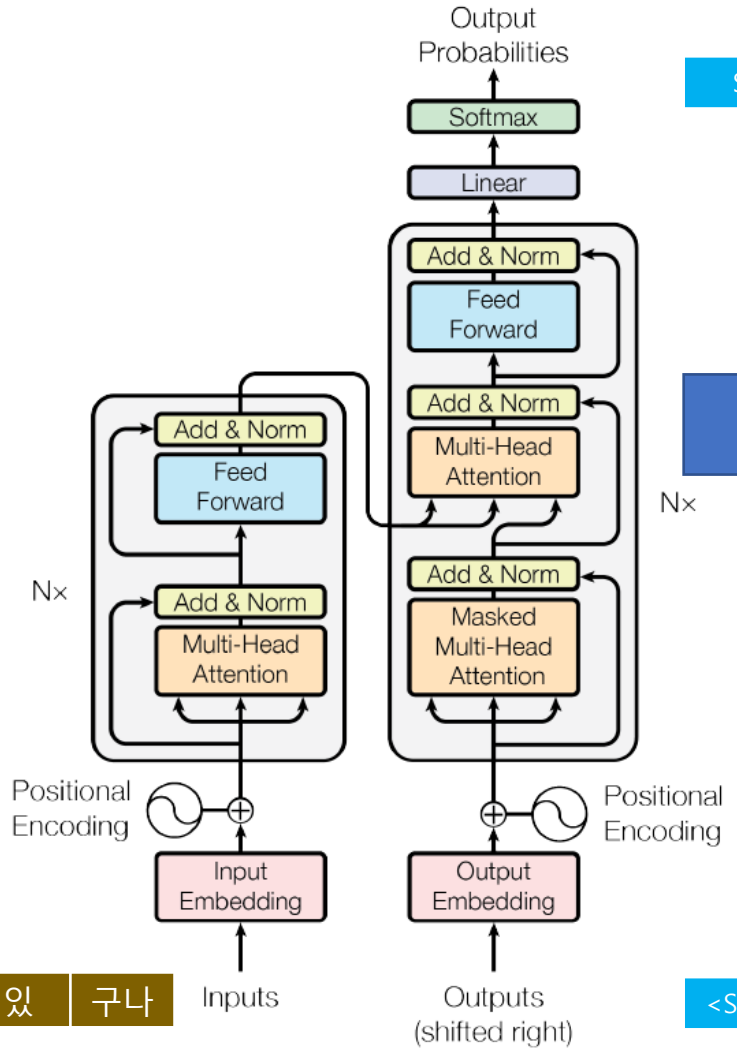
Annotation not from the paper marked red

Encoder-decoder attention:
Q is from the previous decoder layer, K and V is from the output of the encoder.

Self-attention in encoder:
Q, K, V is all from the output of the previous layer in the encoder.

Self-attention in decoder:
Q, K, V is all from the output of previous layer in the decoder. *Mask is applied.*

Inputs and Outputs in the Transformer



Source Sentence

너	는	계획	이	다	있	구나

Target Sentence

So	you	have	...	<EOS>

Done parallel in training (Teacher-forcing)

...

Illegal

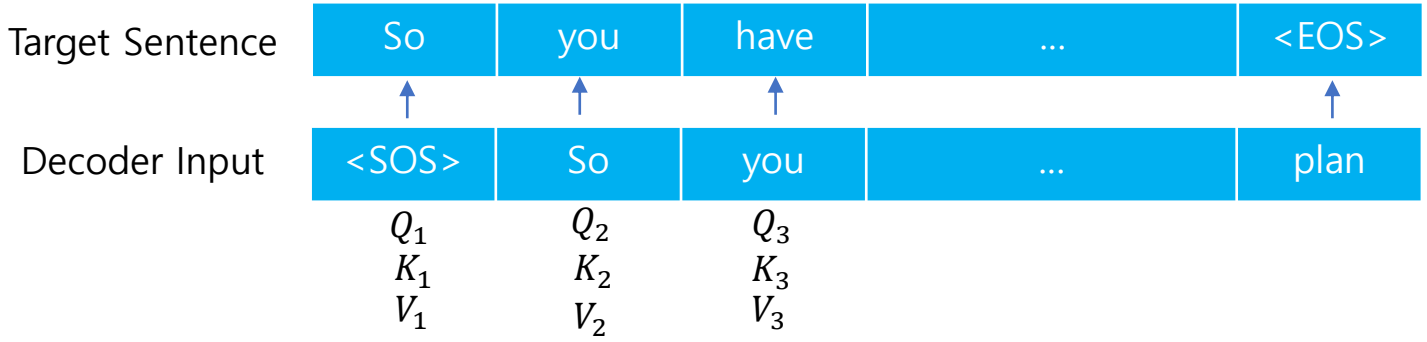
<SOS>	So	you	...	plan

Target Sentence (shifted)

Figure 1: The Transformer - model architecture.

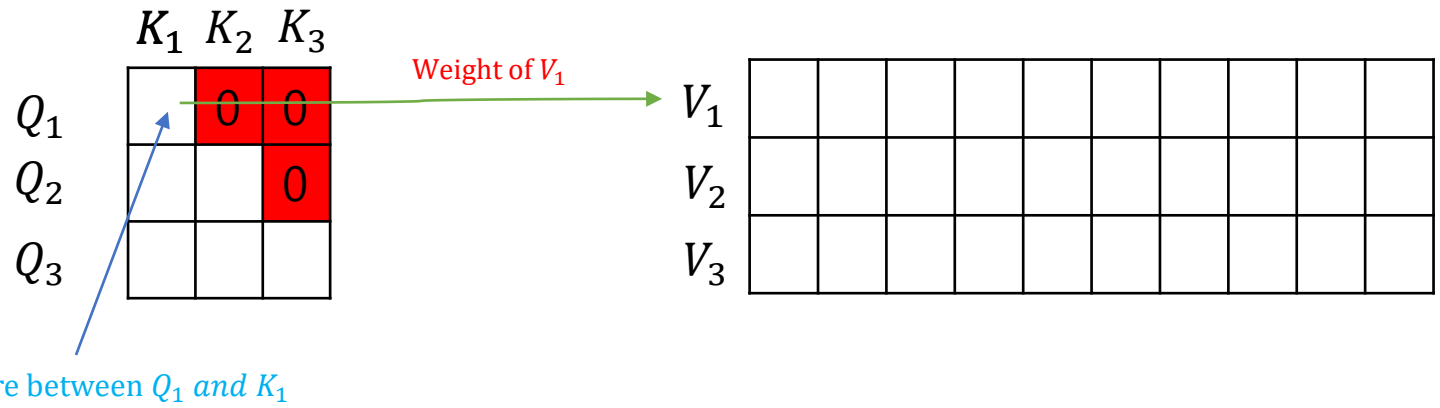
Masked Scaled Dot-Product Attention

Red blocks are the illegal connections; in these blocks, score → 0



*Self-attention: Q, K, V all comes from the same embedding.

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



Position-wise Feed-Forward Networks

Sub-layers are circled.

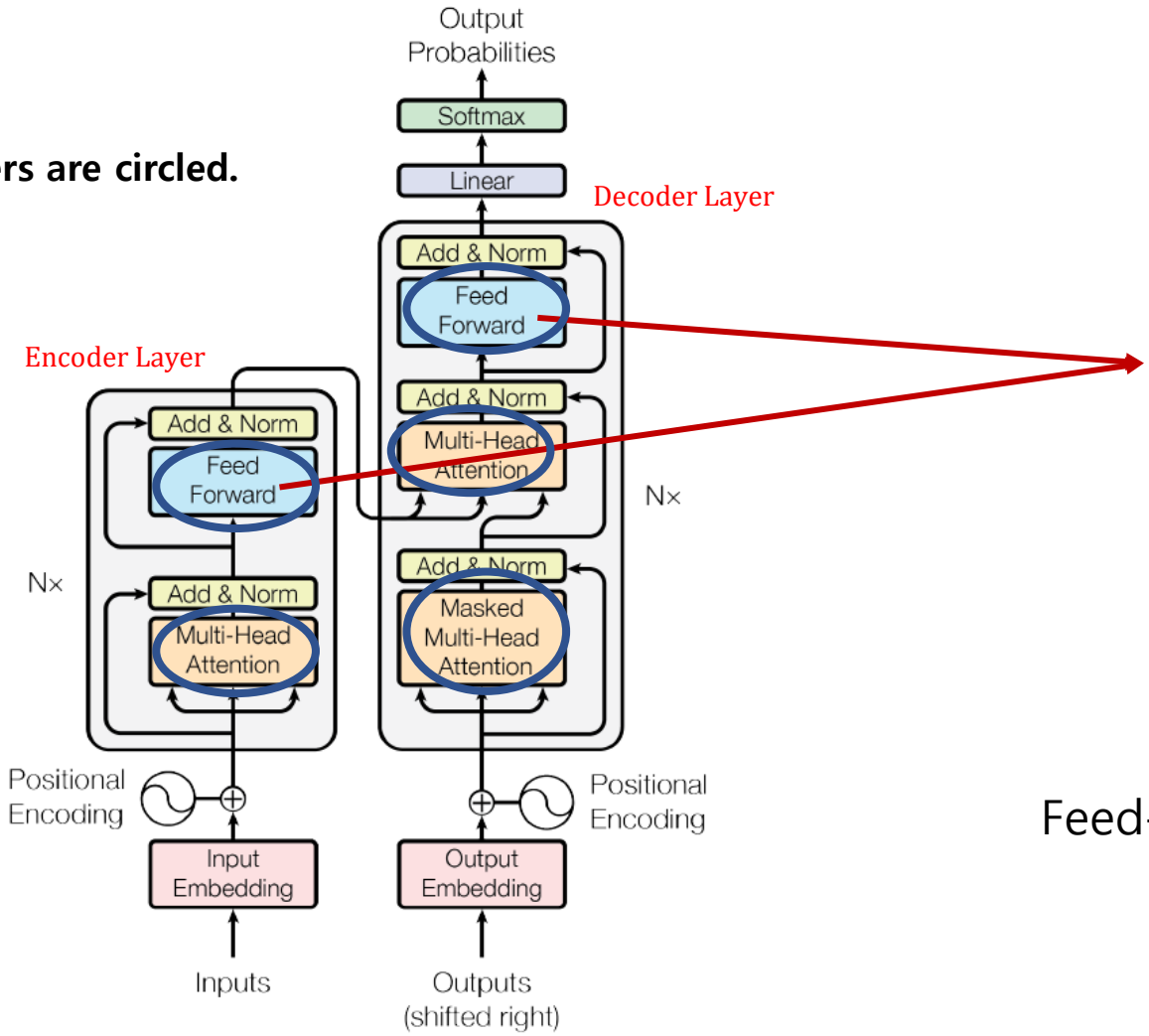


Figure 1: The Transformer - model architecture.

Annotation not from the paper marked red

Position-wise Feed-Forward Networks

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

$Dimesionality: d_{model} \rightarrow d_{ff} \rightarrow d\_model$

Feed-forward networks do not affect the shape of data.

More on layers (other than sub-layers)

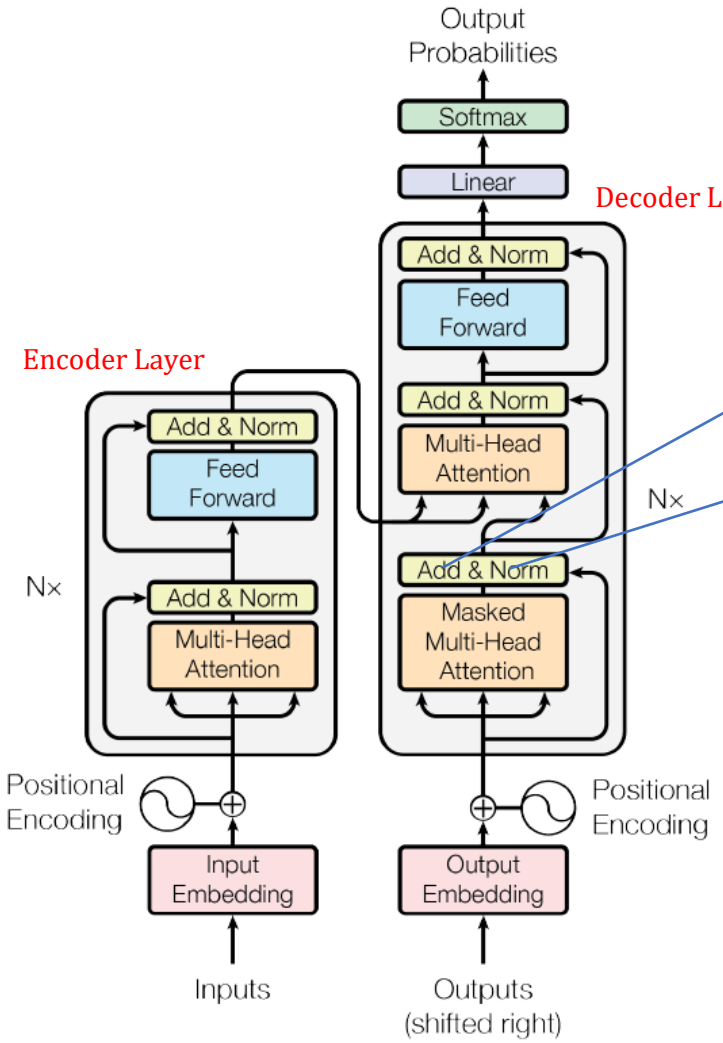


Figure 1: The Transformer - model architecture.

Annotation not from the paper marked red

Residual Connection

Primary reason why attention and feed-forward networks are set to have no affect on the shape of data.
He, Kaiming, et al. "Deep residual learning for image recognition." (2016).

Layer Normalization

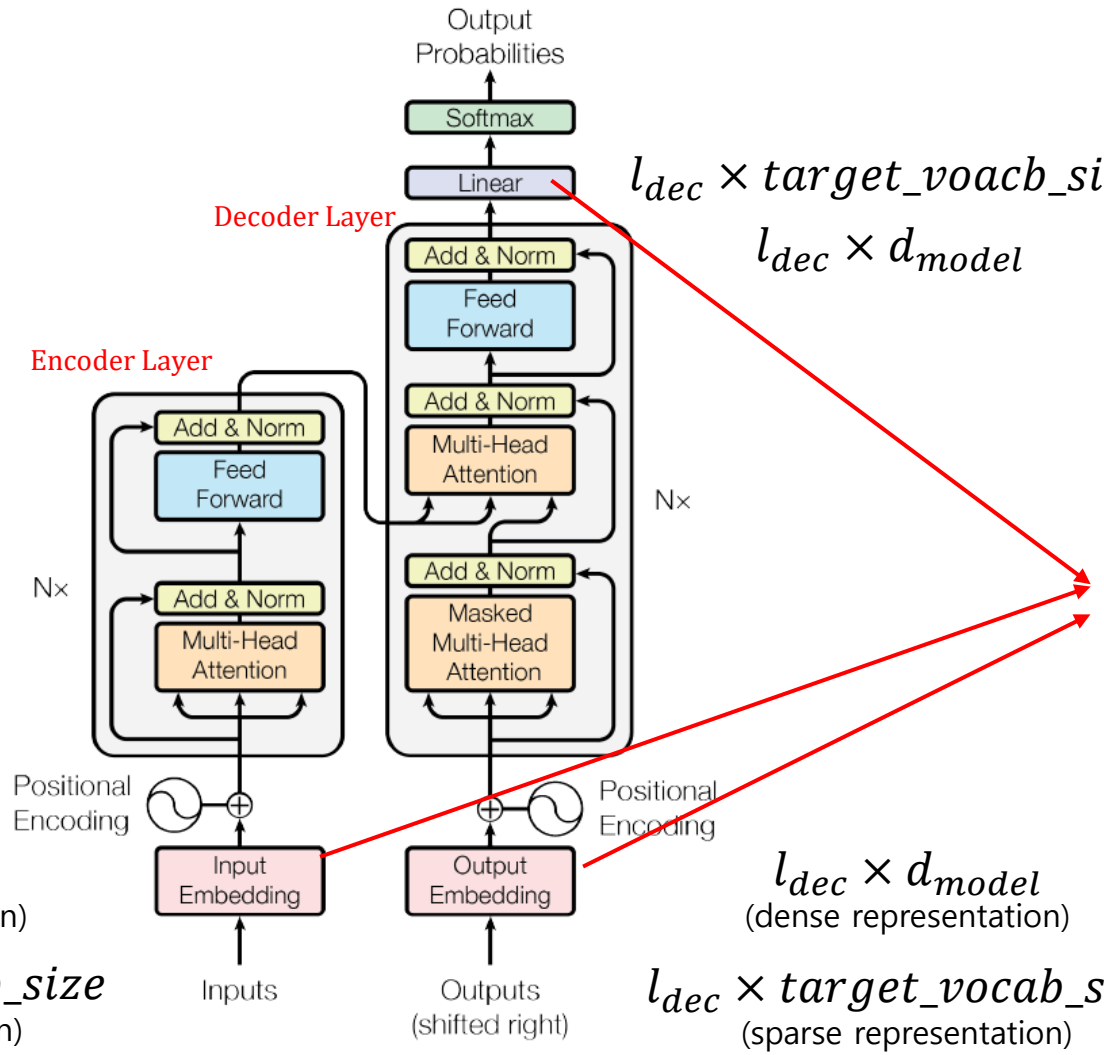
Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton. "Layer normalization." (2016).

Dropout

Output of every sub-layers & sum of embedding and positional encoding

These do not affect the shape of data.
So do attentions. So do feed-forward networks.
Thus, encoder and decoder have no affect on the shape of data.

Embedding and Softmax



$l_{dec} \times target\_voacb\_size$

$l_{dec} \times d_{model}$

$l_{enc} \times d_{model}$
(dense representation)

$l_{enc} \times input\_vocab\_size$
(sparse representation)

$l_{dec} \times d_{model}$
(dense representation)

$l_{dec} \times target\_vocab\_size$
(sparse representation)

Figure 1: The Transformer - model architecture.

Original paper suggests sharing the same weight matrix, though famous codes on the Internet seems to ignore this, perhaps due to the difficulty of implementation.

Annotation not from the paper marked red

Positional Encoding

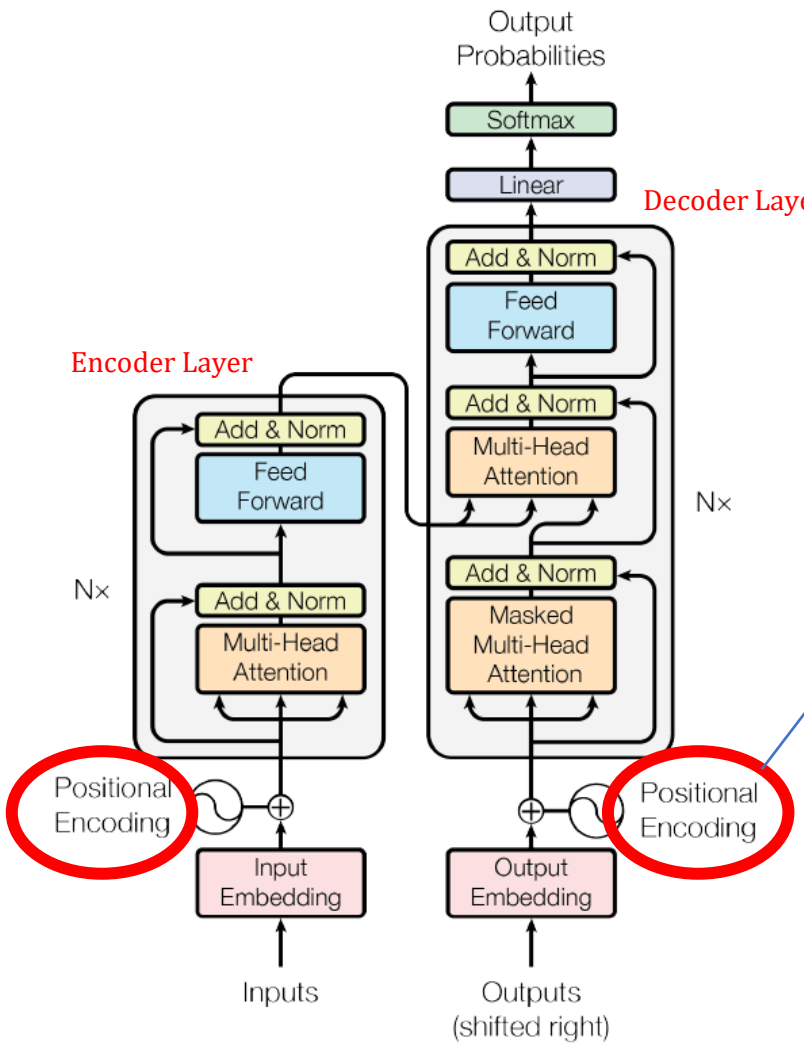


Figure 1: The Transformer - model architecture.

Annotation not from the paper marked red

Why positional encoding?

"… 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."

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

$$PE \in \mathbb{R}^{l \times d_{model}},$$

PE is summed with embedding

Position embedding can also be learned.

"… also experimented with using learned positional embeddings instead, and found that the two versions produced nearly identical results"

Pseudocode

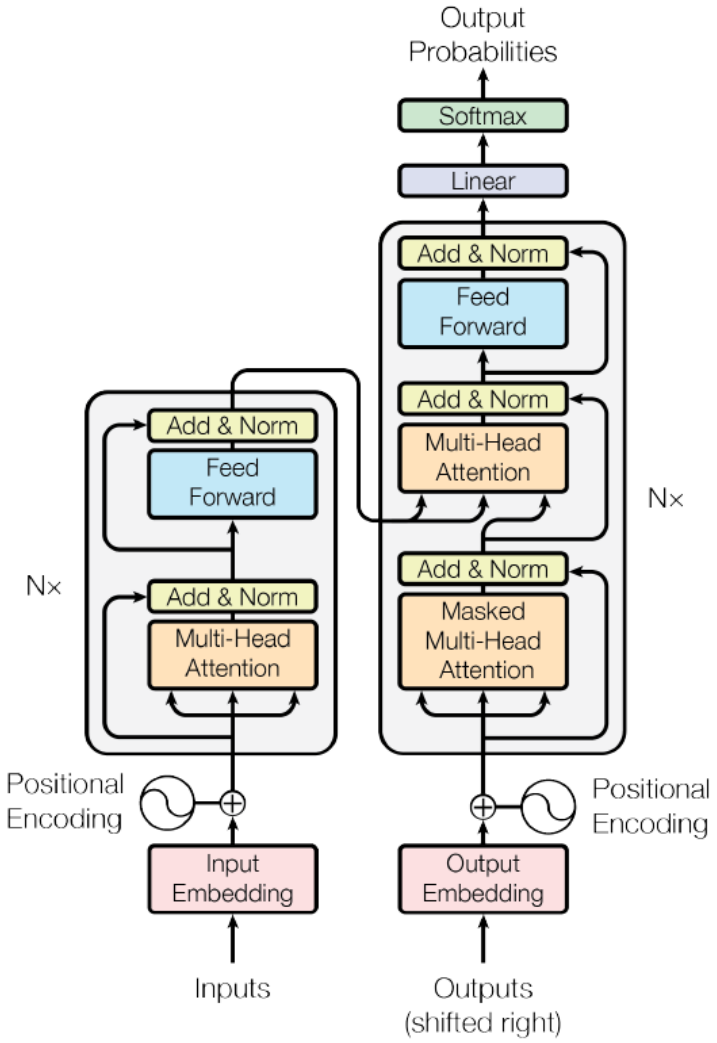


Figure 1: The Transformer - model architecture.

```
def vanilla_transformer(enc_inp, dec_inp):
    """Transformer variant known as the "vanilla" transformer."""
    x = embedding(enc_inp) * sqrt(d_m)
    x = x + pos_encoding(x)
    x = dropout(x)
    for _ in range(n_enc_layers):
        attn = multi_head_attention(x, x, x, None)
        attn = dropout(attn)
        attn = layer_normalization(x + attn)

        x = pointwise_ff(attn)  # ReLU activation(affine map(attn))
        x = layer_normalization(x + attn)

    # x is at this point the output of the encoder
    enc_out = x

    x = embedding(dec_inp) * sqrt(d_m)
    x = x + pos_encoding(x)
    x = dropout(x)
    mask = causal_mask(x)
    for _ in range(n_dec_layers):
        attn1 = multi_head_attention(x, x, x, mask)
        attn1 = layer_normalization(attn1 + x)

        attn2 = multi_head_attention(attn1, enc_out, enc_out, None)
        attn2 = dropout(attn2)
        attn2 = layer_normalization(attn1 + attn2)

        x = pointwise_ff(attn2)
        x = layer_normalization(attn2 + x)
    return dense(x)
```

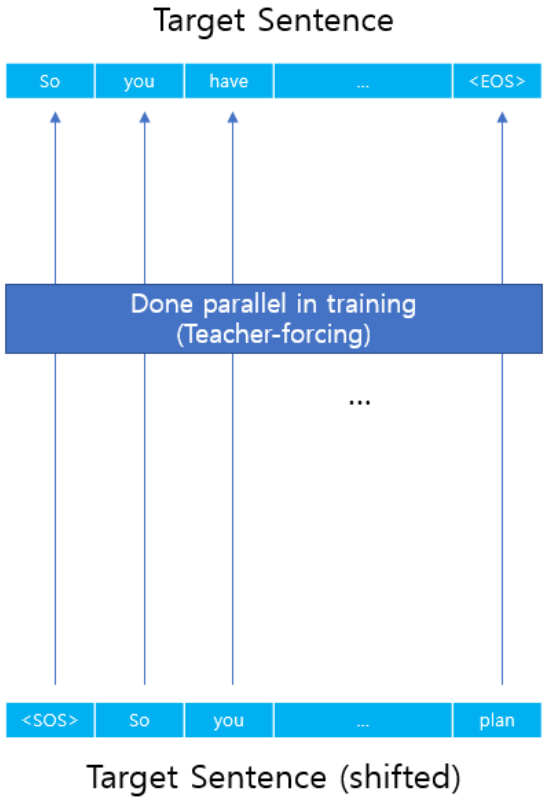
https://en.wikipedia.org/wiki/Transformer_(machine_learning_model)

Why are transformers so nice?

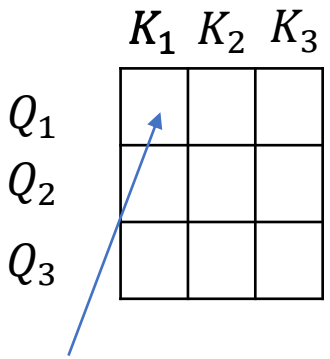
1. More Parallelization

"The lack of recurrence enables greater within-training-example parallelization."

Liu, Peter J., et al. "Generating wikipedia by summarizing long sequences." (2018).



2. Long-range Dependency



Resources relating to Transformer

- Code
- (Tensorflow) https://www.tensorflow.org/text/tutorials/transformer
- (PyTorch) http://nlp.seas.harvard.edu/2018/04/03/attention.html
- Resources
- https://www.youtube.com/watch?v=OyFJWRnt_AY&t=1760s
- https://jalammar.github.io/illustrated-transformer/

Notable papers

- Cho, Kyunghyun, et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation." (2014). (RNN Encoder-Decoder; GRU)
- Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks.“ (2014). (RNN Encoder-Decoder)
- Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate.“ (2014). (Bahdanau Attention)
- Luong, Minh-Thang, Hieu Pham, and Christopher D. Manning. "Effective approaches to attention-based neural machine translation.“ (2015). (Luong Attention)
- **Vaswani, Ashish, et al. "Attention is all you need." (2017). (Transformer)**