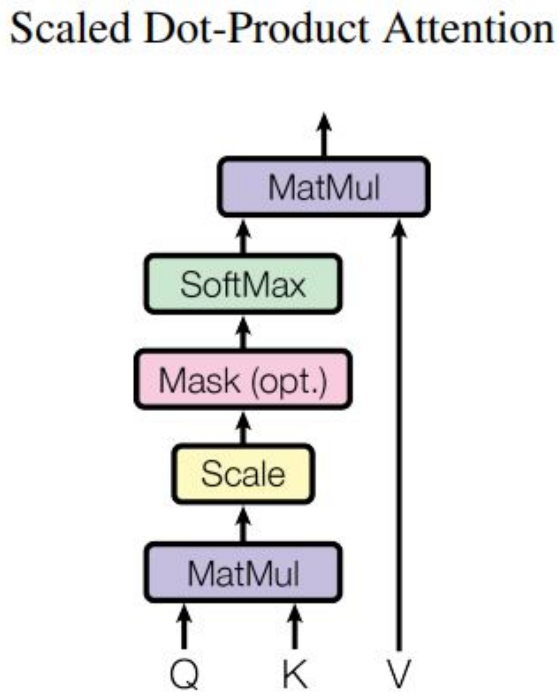
Attention, Transformers, GPT

The traditional **Attention** layer



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

https://arxiv.org/pdf/1706.03762.pdf , “Attention is All You Need”, by scientists from Google Brain and Google Research, in NeurIPS 2017.

The traditional **Attention** layer

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

1. Queries, keys and values are all row vectors.
2. Given a single query:
   a. Compute the similarity of the query with every key.
   b. Scale and softmax these similarity scores to obtain a discrete probability mass fn.
   c. Obtain a weighted sum of the value vectors– the weights are these probabilities computed earlier.

The traditional **Attention** layer

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

In practice, we perform many queries at once – by passing a Q matrix (many row vectors) instead of a single q vector.

Observe that Q and K need to have the same number of columns, while K and V need to have the same number of rows.

Multi-Head Attention

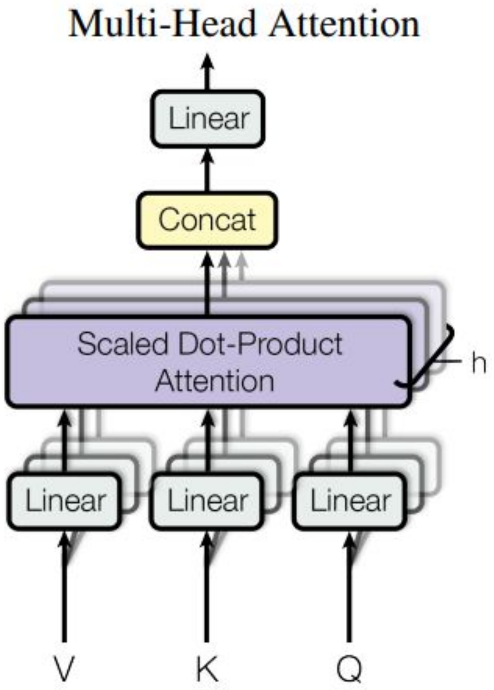
Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this.

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

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.

Multi-Head Attention



$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$
$$\text{where}\ \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Q, K, V ???

Depending on the training task, where we obtain Q, K and V varies.

Three cases are discussed:

1. Encoder-Decoder Transformers (such as in Machine Translation)
2. Decoder-only (like GPT)
3. Encoder-only (like BERT)

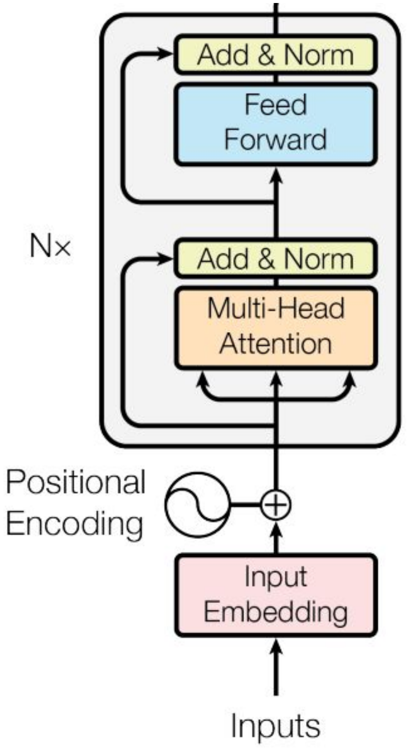
Encoder-Decoder Transformers for Machine Translation

Here, the network is in two parts:

1. Encoder side – generate a single vector to represent the input sentence.
2. Decoder side – generate the output sentence, conditioned on the encoding for the input sentence (and the output so far).

https://jalammar.github.io/illustrated-transformer/
– an excellent blog for the fine print, which is rare because most blogs only give the broad strokes!

Encoder of Encoder-Decoder



1. There are **N** TransformerEncoder blocks, and Q, K, V for each block are the [projected] output of the previous block.
2. Q, K, V for the first TransformerEncoder block come from the embeddings for the input sentence, albeit with a **positional encoding** added.

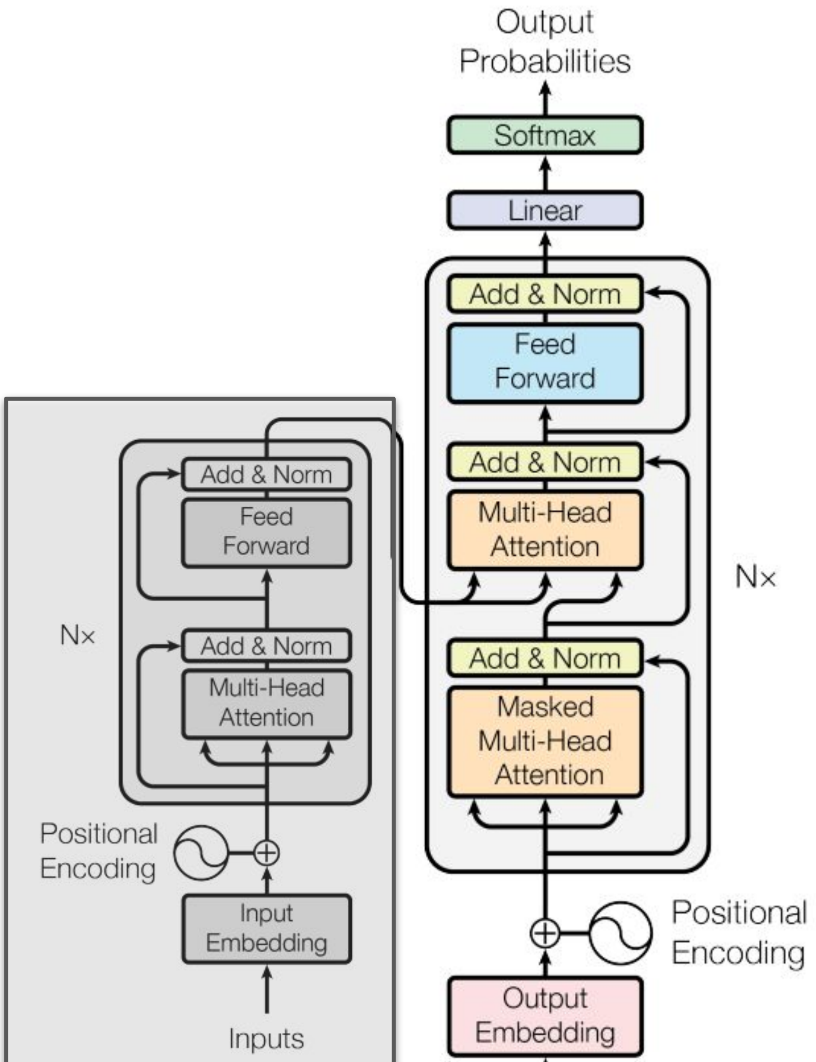
Positional Encoding: What and Why?

1. In the given form, the Transformer block maintains a kind of symmetry across timesteps. Words (or tokens) are given the same preference irrespective of their positions.
2. This is not desirable in sequential data contexts – hence, we add a positional encoding to the input.
3. The functional form of this input varies depending on the implementation. In some cases, this encoding is a learnable set of weights.

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

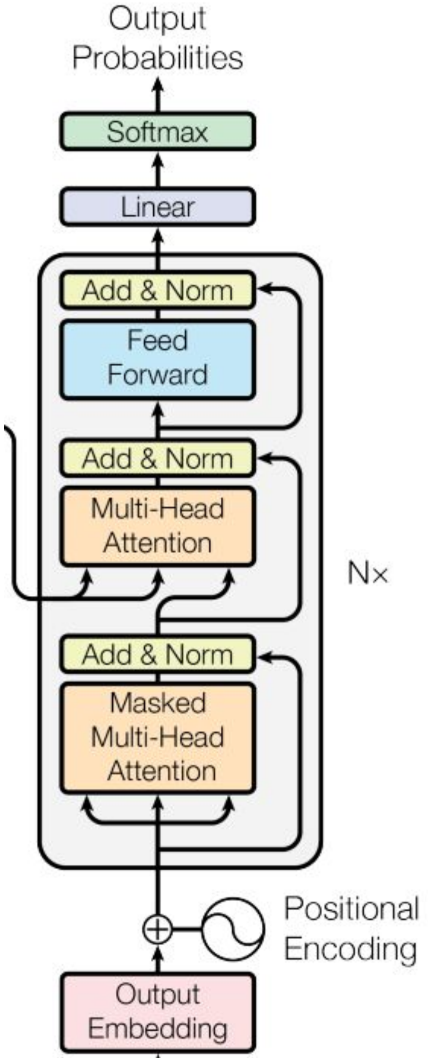
Decoder of Encoder-Decoder

1. In each TransformerDecoder block, there are two Attention units (there was only one in the TransformerEncoder).
2. For the first Attention unit, QKV are from the output of the previous TransformerDecoder block.
3. In case of the first TransformerDecoder block, QKV for the first Attention unit come from the **final output so far.**
4. For the second Attention layer, Q is from the first Attention layer while K and V are from **the last TransformerEncoder's output.**



Decoder of Encoder-Decoder: Masking

1. During training, we will be passing an entire output sequence to the decoder. However, what is desired is that the query should only attend to the output **so far –** otherwise, in the training, the future output would influence the past words predicted!
2. For this, a **causal mask** is applied. This is typically just a triangular matrix, and is not a major hurdle in training. It's more of an implementation detail to be watchful of.



Looking ahead: Transfer Learning – GPT and BERT

Transfer learning:

1. Take a large model.
2. Freeze all except the last few layers.
3. Optionally, remove the last few layers.
4. Add some layers of your own.
5. Train.

Using this, you can benefit from the experience a large model has on a **generic** task and apply that with minimal training to a **specific** task – with the hope of better results than using just a large task-specific model.

Two such large, generic models are **GPT** (OpenAI) and **BERT** (Google).

https://dailynous.com/2020/07/30/philosophers-gpt-3/