Previous lecture/practical

Any question?

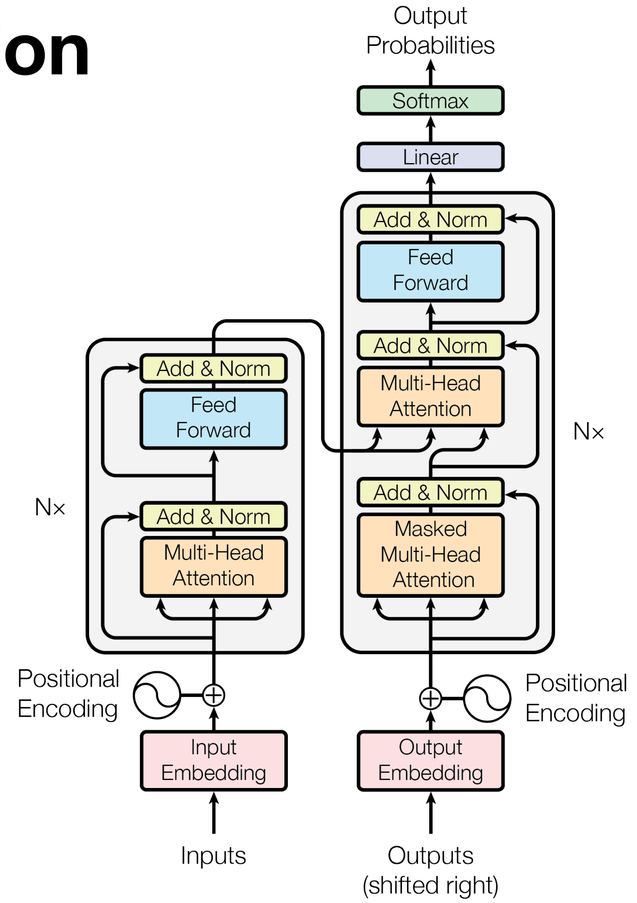
Previous lecture/practical

Any question?

Recap: you will have to implement, train and evaluate RNN models during the final project —> Important to understand the underlying mechanisms…

Introduction to Deep Learning

Lecture 3: Transformers and Attention



Course Overview

1. Convolutional Neural Networks (1h lecture + 3h practical)
2. Recurrent Neural Networks (1h lecture + 3h practical)
3. Transformers and Attention (1h lecture + 3h practical)
4. Project (12h)

Course Overview

1. Convolutional Neural Networks (1h lecture + 3h practical)
2. Recurrent Neural Networks (1h lecture + 3h practical)
3. Transformers and Attention (1h lecture + 3h practical)
4. Project (12h)

Useful resources about Attention/Transformers

- Vaswani et al., *Attention is all you need*, NeurIPS 2017
- Dosovitskiy et al., *An image is worth 16x16 words: Transformers for image recognition at scale*, ICLR 2021

- The Illustrated Transformer: https://jalammar.github.io/illustrated-transformer/
- The annotated Transformer: https://nlp.seas.harvard.edu/2018/04/03/attention.html

The motivation behind attention

MLP —> Each neuron gets **information from all others from the previous layer**
CNN —> Each neuron gets **information from only neurons in its neighbourhood**
RNN —> Information is propagated inside a **hidden state memory**

Attention —> Can a neuron get **information from neurons depending on the similarity between their associated values**?

Attention in recurrent networks

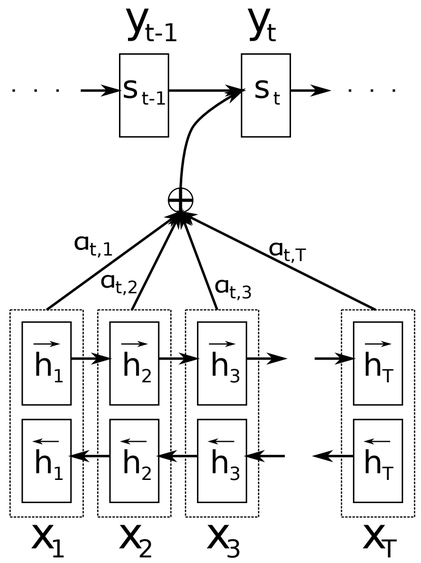
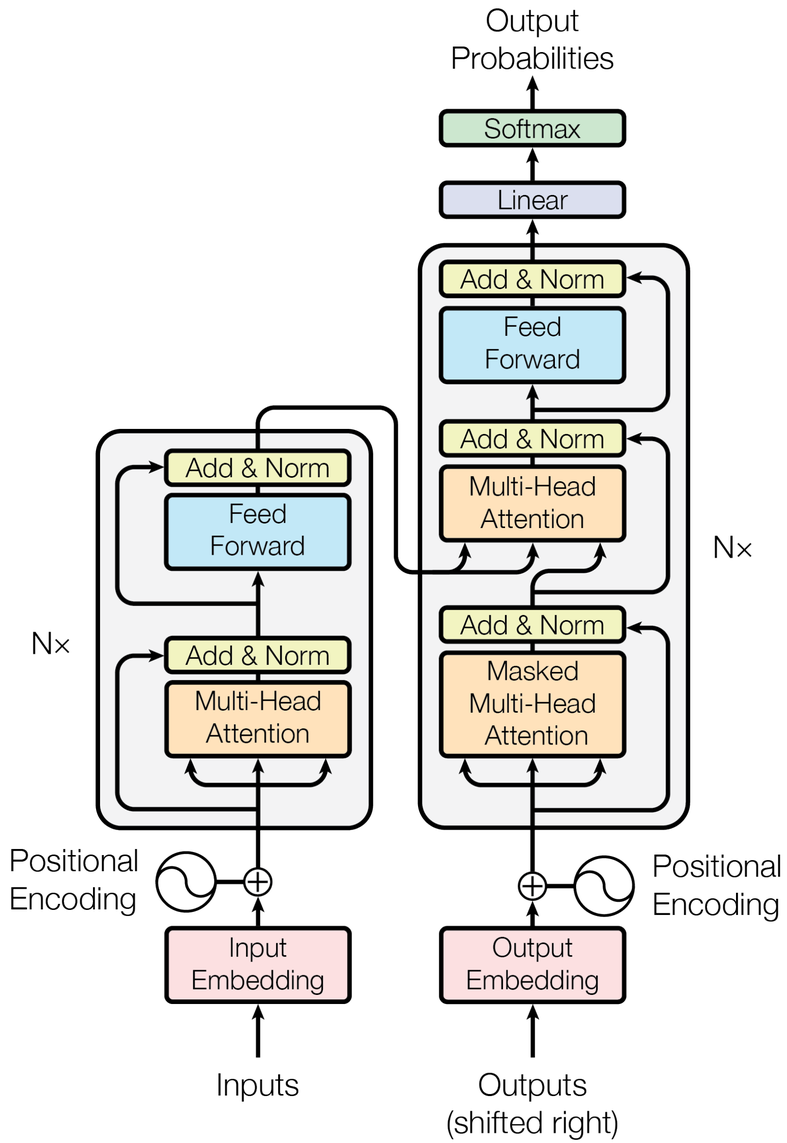


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

Bahdanau et al., Neural machine translation by jointly learning to align and translate, ICLR 2015

Attention is all you need



Vaswani et al. Attention is all you need, NeurIPS 2017

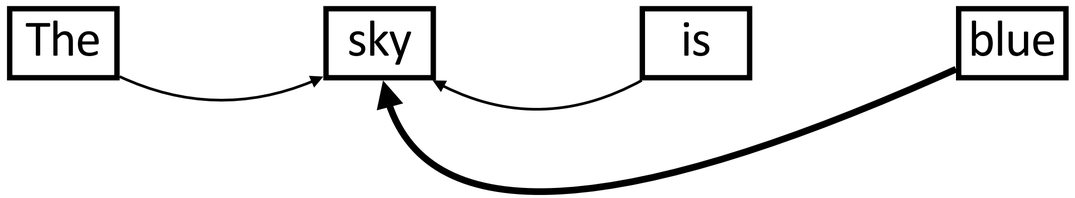
Attention is all you need — Self-attention

Let's consider a set $X = \{x_i\}$ of vectors. These elements will be called **tokens** in the remaining of the course.

An important assumption is that **tokens are related in a certain way**.

The idea behind self-attention is to **enrich the representation of each item with the information contained in other vectors**.

Example: A sentence where each word is a token. The representation of the word "sky" gets enriched by the information brought by other tokens (e.g. the "blue" token can be used to enrich "sky" as it provides colour information). All tokens do not enrich "sky" in the same way (see arrow thickness).



Attention is all you need — Self-attention

Let's consider a set $X = \{x_i\}$ of vectors. These elements will be called **tokens** in the remaining of the course.

An important assumption is that **tokens are related in a certain way**.

The idea behind self-attention is to **enrich the representation of each item with the information contained in other vectors**.

In self-attention, **each item attends to all items in the set** (even to itself).

Attention is all you need — Self-attention

Let's consider a set $X = \{x_i\}$ of vectors. These elements will be called **tokens** in the remaining of the course.

An important assumption is that **tokens are related in a certain way**.

The idea behind self-attention is to **enrich the representation of each item with the information contained in other vectors**.

In self-attention, **each item attends to all items in the set** (even to itself).

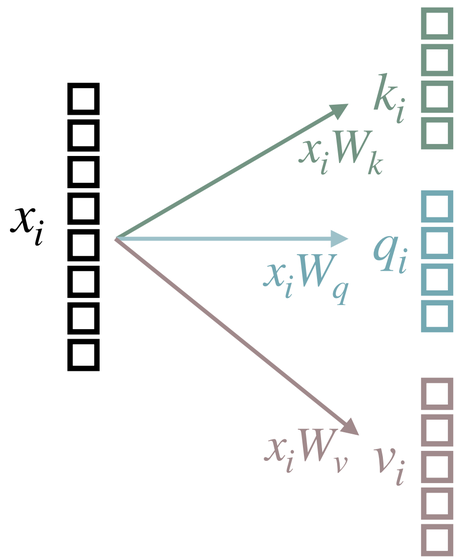
But, what do we mean by "attend"?

Attention is all you need — Self-attention

We predict 3 quantities for each token $x_i \in \mathbb{R}^a$:

1. **Key** $k_i \in \mathbb{R}^{d_k}$
2. **Query** $q_i \in \mathbb{R}^{d_k}$
3. **Value** $v_i \in \mathbb{R}^{d_v}$

We thus learn 3 linear layers ($W_k \in \mathbb{R}^{(a+1)\times d_k}$, $W_q \in \mathbb{R}^{(a+1)\times d_k}$, $W_v \in \mathbb{R}^{(a+1)\times d_v}$) to predict quantities from input x_i.

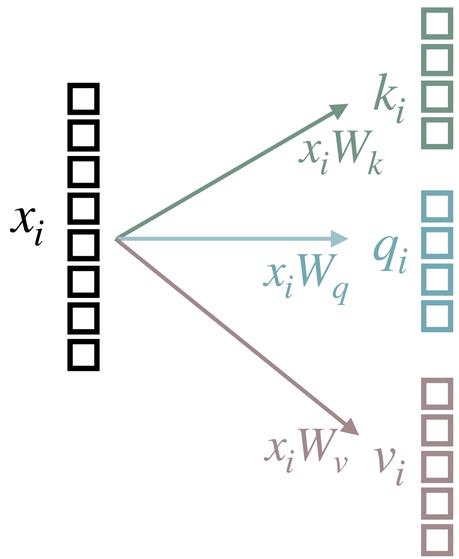


Attention is all you need — Self-attention

We predict 3 quantities for each token $x_i \in \mathbb{R}^a$:

1. **Key** $k_i \in \mathbb{R}^{d_k}$
2. **Query** $q_i \in \mathbb{R}^{d_k}$
3. **Value** $v_i \in \mathbb{R}^{d_v}$

We thus learn 3 linear layers ($W_k \in \mathbb{R}^{(a+1)\times d_k}$, $W_q \in \mathbb{R}^{(a+1)\times d_k}$, $W_v \in \mathbb{R}^{(a+1)\times d_v}$) to predict quantities from input x_i.



If we write X as a matrix where each row is an x_i item we can compute all keys K, queries Q , and values V as,

$$K = XW_k$$
$$Q = XW_q$$
$$V = XW_v$$

Attention is all you need — Self-attention

We predict 3 quantities for each token $x_i \in \mathbb{R}^a$:

1. **Key $k_i \in \mathbb{R}^{d_k}$ —> My summary when items search for information in me.**
2. **Query $q_i \in \mathbb{R}^{d_k}$ —> What I want to search for in other items.**
3. **Value $v_i \in \mathbb{R}^{d_v}$ —> The information retrieved from me by other items.**

Attention is all you need — Self-attention

We predict 3 quantities for each token $x_i \in \mathbb{R}^a$:

1. **Key $k_i \in \mathbb{R}^{d_k}$ —> My summary when items search for information in me.** (K is the matrix of all keys)
2. **Query $q_i \in \mathbb{R}^{d_k}$ —> What I want to search for in other items.** (Q is the matrix of all queries)
3. **Value $v_i \in \mathbb{R}^{d_v}$ —> The information retrieved from me by other items.** (V is the matrix of all values)

We compute a **scaled similarity matrix S between all queries Q and keys K**, by performing a dot product and a scaling operation:

$$S = \frac{QK^T}{\sqrt{d_k}}$$

Attention is all you need — Self-attention

We predict 3 quantities for each token $x_i \in \mathbb{R}^a$:

1. **Key $k_i \in \mathbb{R}^{d_k}$ —> My summary when items search for information in me.** (K is the matrix of all keys)
2. **Query $q_i \in \mathbb{R}^{d_k}$ —> What I want to search for in other items.** (Q is the matrix of all queries)
3. **Value $v_i \in \mathbb{R}^{d_v}$ —> The information retrieved from me by other items.** (V is the matrix of all values)

We compute a **scaled similarity matrix S between all queries Q and keys K**, by performing a dot product and a scaling operation:

$$S = \frac{QK^T}{\sqrt{d_k}}$$

For each query, we compute a **distribution over keys** by performing a **softmax operation**. The final matrix is A:

$$A = softmax(S)$$

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K$$

From *https://en.wikipedia.org/wiki/Softmax_function*

Attention is all you need — Self-attention

We predict 3 quantities for each token $x_i \in \mathbb{R}^a$:

1. **Key $k_i \in \mathbb{R}^{d_k}$ —> My summary when items search for information in me.** (K is the matrix of all keys)
2. **Query $q_i \in \mathbb{R}^{d_k}$ —> What I want to search for in other items.** (Q is the matrix of all queries)
3. **Value $v_i \in \mathbb{R}^{d_v}$ —> The information retrieved from me by other items.** (V is the matrix of all values)

We compute a **scaled similarity matrix S between all queries Q and keys K**, by performing a dot product and a scaling operation:

$$S = \frac{QK^T}{\sqrt{d_k}}$$

For each query, we compute a **distribution over keys** by performing a **softmax operation**. The final matrix is A:

$$A = softmax(S)$$

Finally, the final representation of all tokens T is done by performing a linear combination of values V weighted by A:

$$T = AV$$

Attention is all you need — Multi-head Self-attention

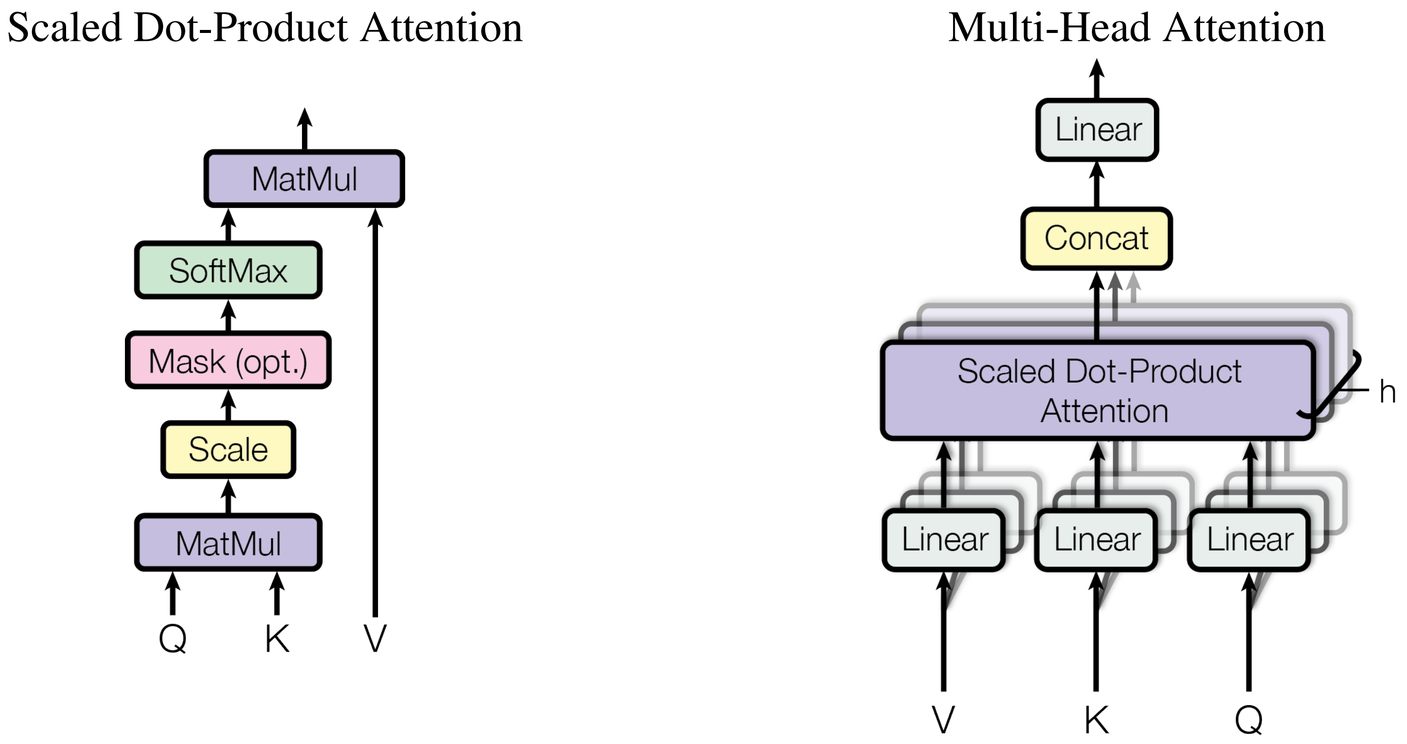


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

Figure reproduced from Vaswani et al.

Vaswani et al. Attention is all you need, NeurIPS 2017

Attention is all you need — Multi-head Self-attention

Simple yet elegant idea: Multi-head attention allows **projecting the same token into different query, key, and value spaces**.

This is a way for each token to **query different information**, but also to **retrieve different information from a same token**.

Attention is all you need — Encoder-Decoder

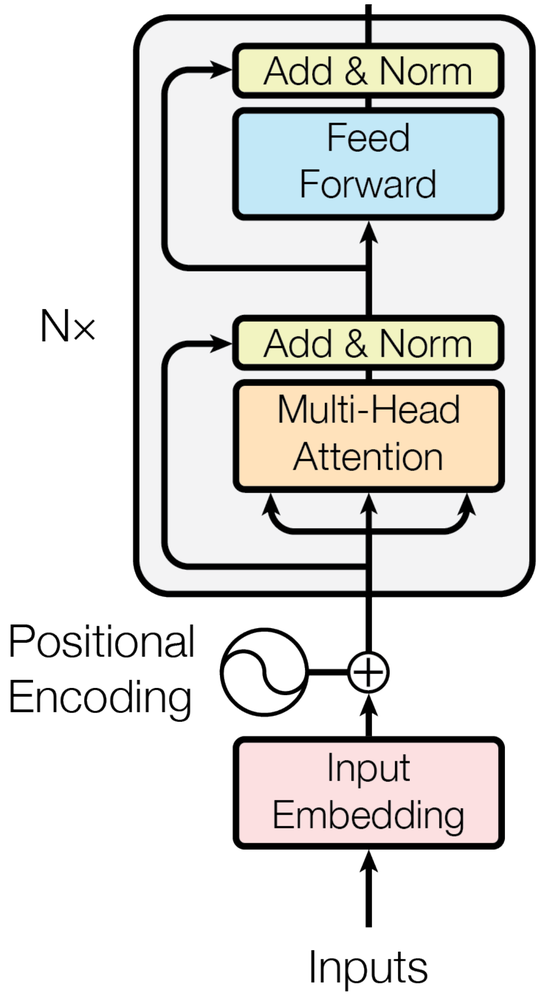
The Encoder-Decoder architecture takes a set of tokens as input and outputs another set of token. More specifically, it was applied to sentence translation: it is given a sentence in French as input and outputs a sentence in English.

A **fundamental difference** between encoder and decoder is the **type of attention**:

- **Encoder**: **Bi-directional** attention, i.e. each item attends to items before and after in the sequence.
- **Decoder**: **Causal** attention, i.e. each item only attends to items before in the sentence. The decoder generates a new sentence so tokens after are simply not known yet!

Let’s now look at the encoder and the decoder separately.

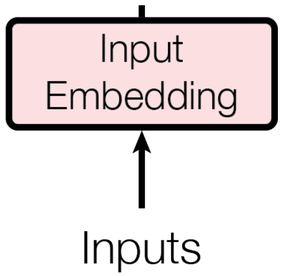
Attention is all you need — Encoder



Vaswani et al. Attention is all you need, NeurIPS 2017

Attention is all you need — Encoder

In the text of sentence translation, the input to the encoder is a sequence of words. But, this **input has to be turned into a numerical form**, i.e. a **sequence of embeddings**. This is an important step!



Vaswani et al. Attention is all you need, NeurIPS 2017

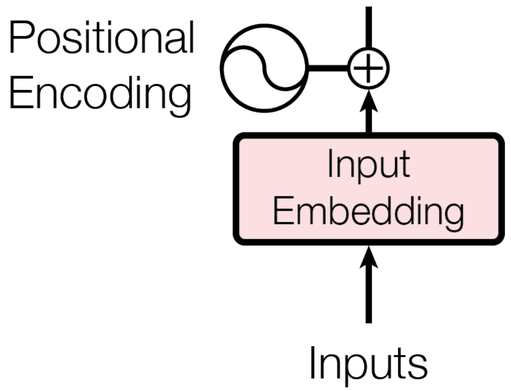
Attention is all you need — Encoder

Recap: a self-attention layer is applied to a **set of items**, i.e. an **<u>unordered</u> collection of vectors**.

But, as humans, we know that **tokens in a sentence have an order**!

We thus **augment each embedding** with **information about where it is localized in the sentence**.

This is done by **adding a positional encoding**.



Vaswani et al. Attention is all you need, NeurIPS 2017

Attention is all you need — Encoder

In the paper, authors experimented with:
- learned positional embeddings
- Sine and cosine functions of different frequencies

$$PE_{(pos,2i)} = \sin\left(pos/10000^{2i/d}\text{model}\right)$$

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

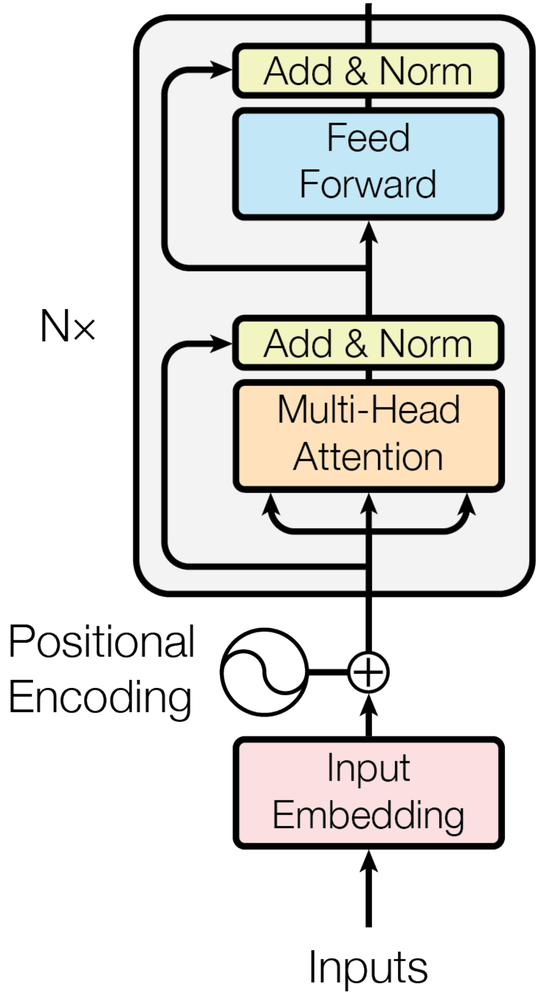
Positional encodings have the same dimension as the input tokens as they will be summed. In the above equations, pos is the position index in the sequence, and i is the dimension.

Vaswani et al. Attention is all you need, NeurIPS 2017

Attention is all you need — Encoder

The encoder is a sequence of N identical (different weights!) layers:

1. Multi-Head attention
2. Normalization with residual connection (Add & Norm)
3. Feed Forward layer (simple sequence of linear layer)
4. Normalization with residual connection (Add & Norm)

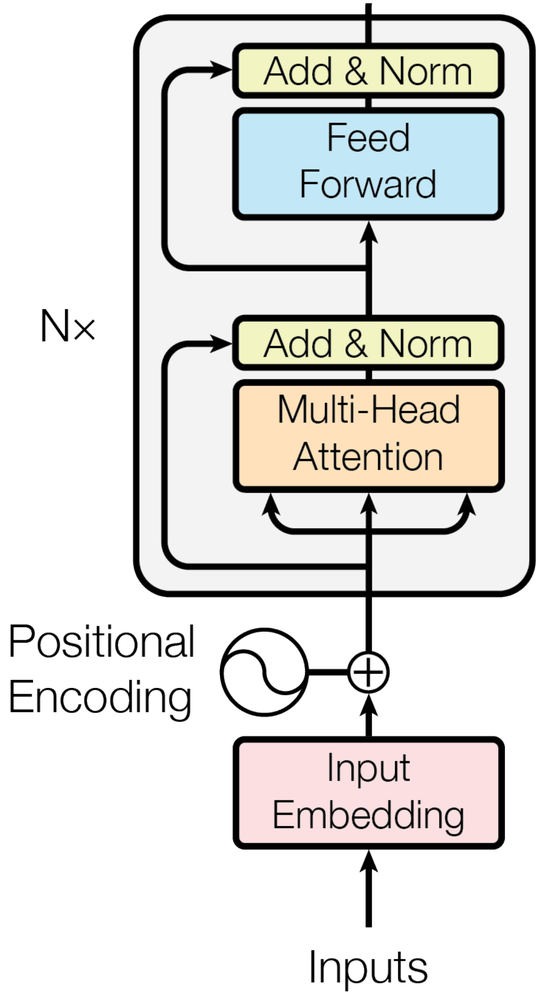


Vaswani et al. Attention is all you need, NeurIPS 2017

Attention is all you need — Encoder

The encoder is a sequence of N identical (different weights!) layers:

1. Multi-Head attention
2. Normalization with residual connection (Add & Norm)
3. Feed Forward layer (simple sequence of linear layer)
4. Normalization with residual connection (Add & Norm)



The used normalization operation is the **Layer Normalization**: normalizes output values from all neurons in a layer for a given training sample, i.e. output mean and standard deviation are computed across all neurons and used to normalize outputs (subtracting the mean and dividing by the std).

Multi-Head Attention:

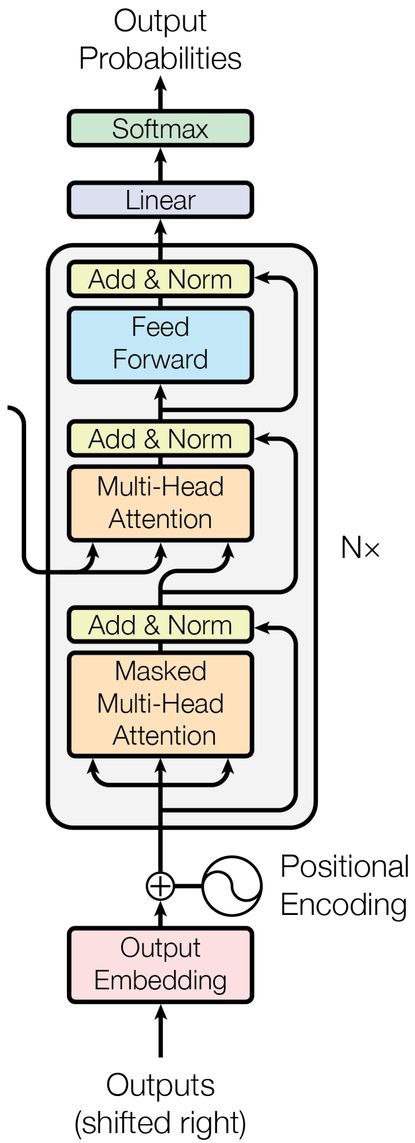
$$out = MultiHeadAttention(x)$$

Add & Norm:

$$out = LayerNorm(out + x)$$

Vaswani et al. Attention is all you need, NeurIPS 2017

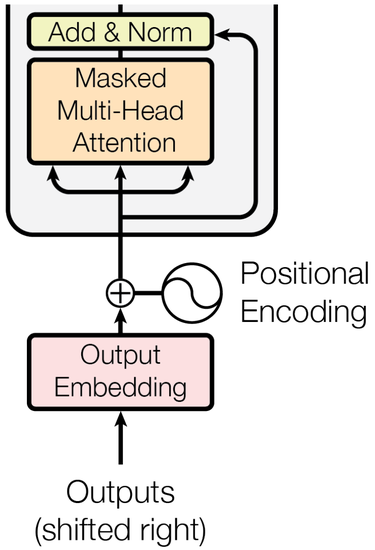
Attention is all you need — Decoder



Vaswani et al. Attention is all you need, NeurIPS 2017

Attention is all you need — Decoder

This is a **standard Multi-Head self-attention** block. It is called **Masked** Multi-Head Attention as it is a **causal attention layer**, each token only attends to the ones before.



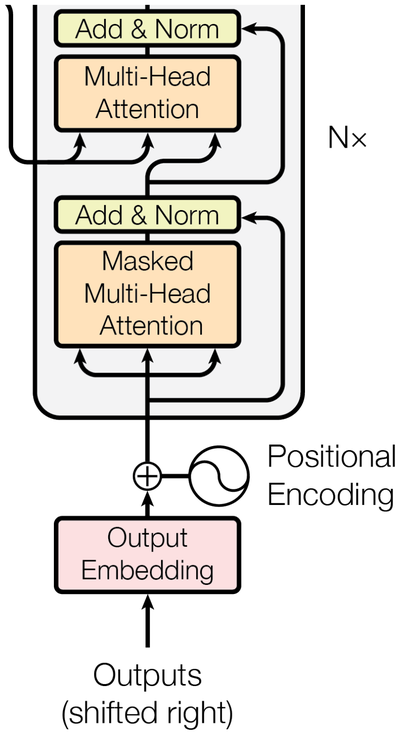
Vaswani et al. Attention is all you need, NeurIPS 2017

Attention is all you need — Decoder

Encoder-Decoder Attention

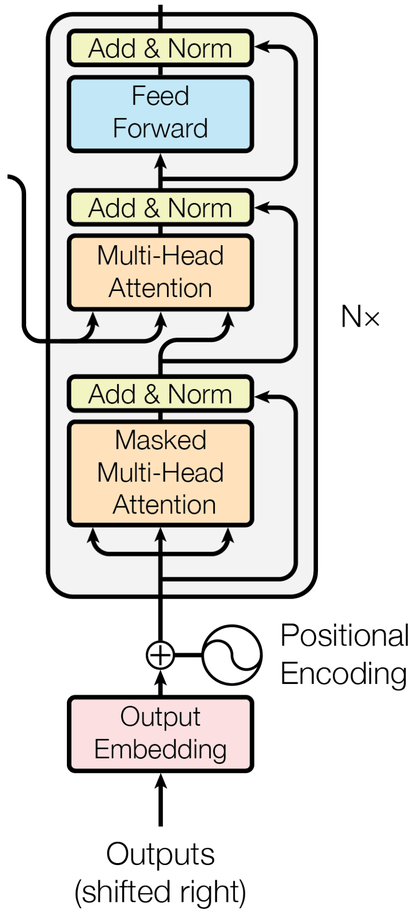
This is also a **standard Multi-Head attention** block. The difference is that it is **not a self-attention layer** as the queries, keys and values are not predicted from the same set.

Queries are predicted from the output of Masked Multi-Head attention layer in the decoder, while keys and values are predicted from the output of the encoder.



Vaswani et al. Attention is all you need, NeurIPS 2017

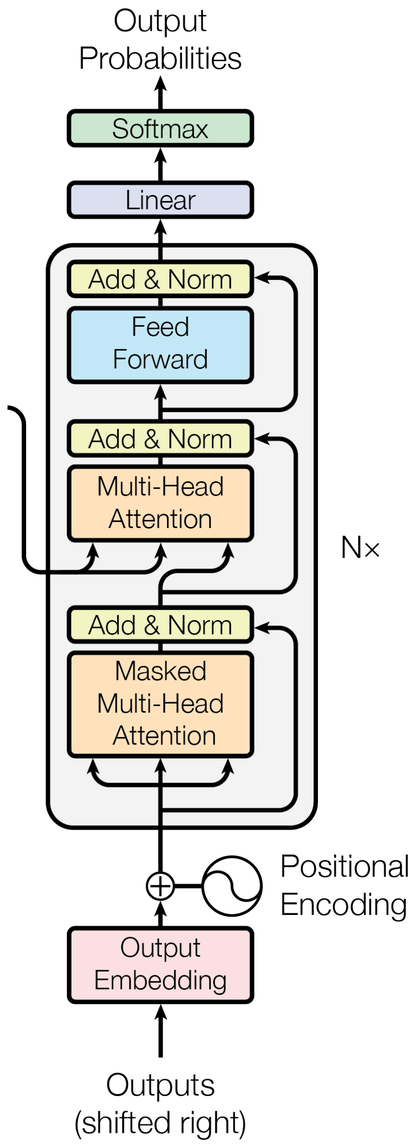
Attention is all you need — Decoder



Vaswani et al. Attention is all you need, NeurIPS 2017

Attention is all you need — Decoder

To generate the next token, we use a linear layer followed by softmax activation function to obtain a distribution over all possible tokens.



Vaswani et al. Attention is all you need, NeurIPS 2017

Vision Transformer (ViT)

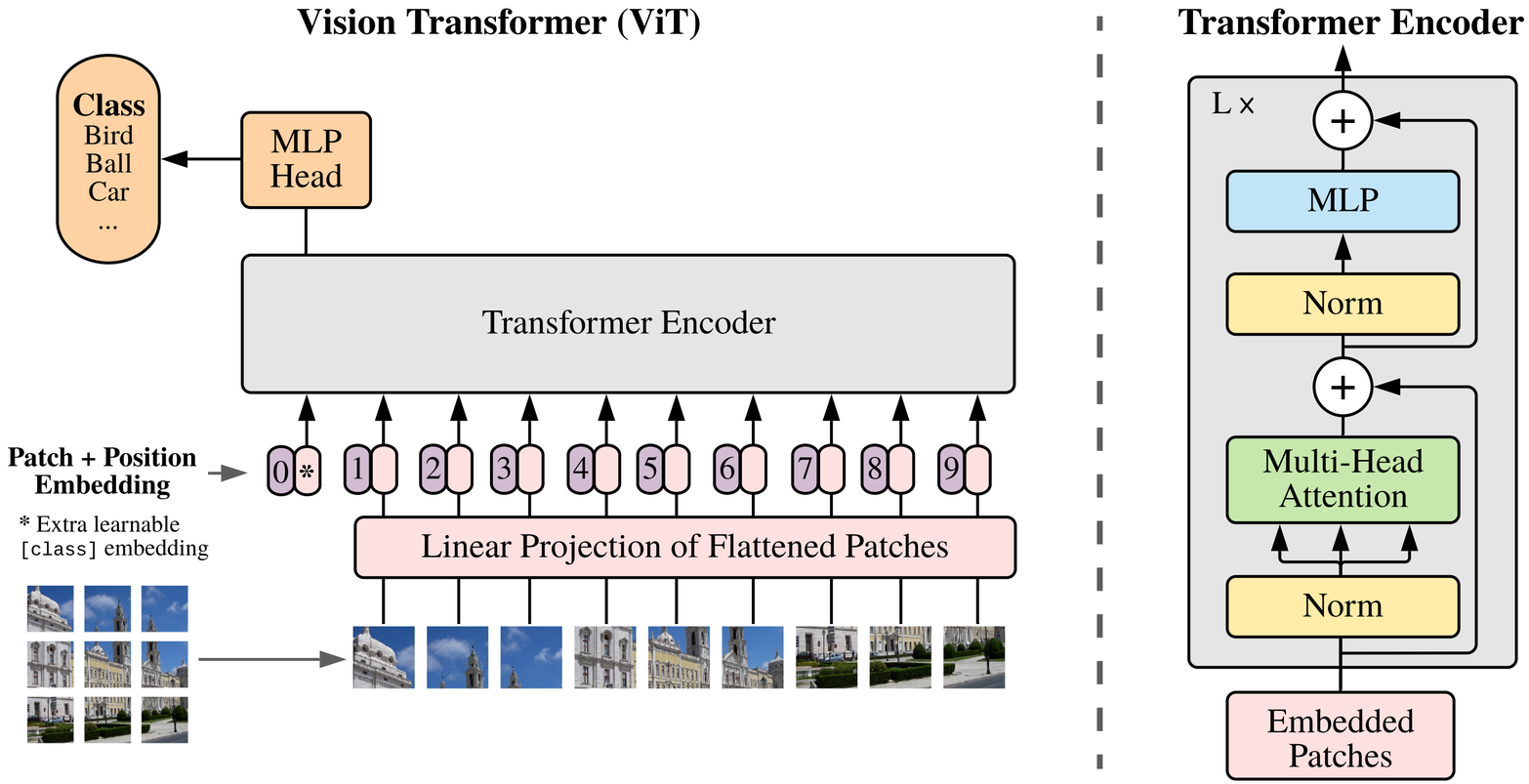


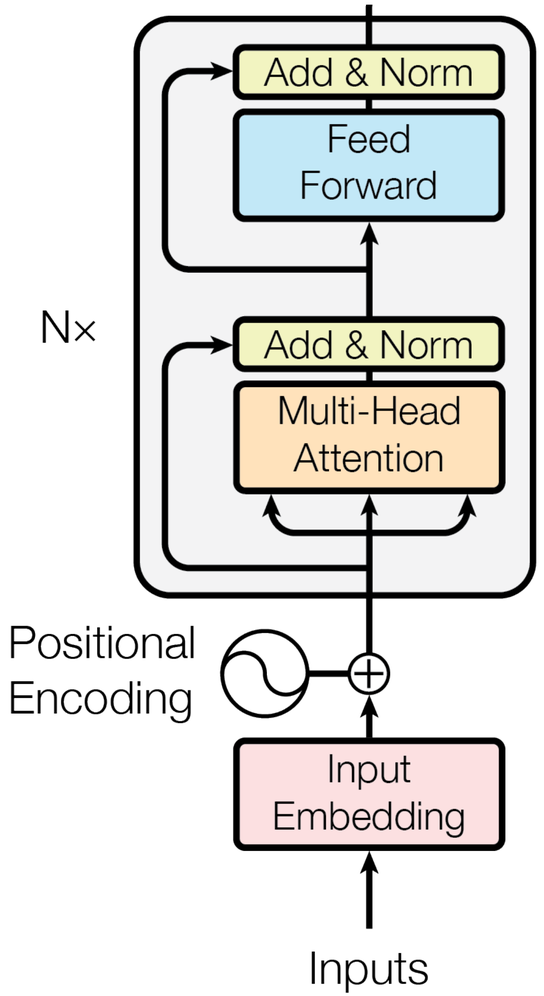
Figure reproduced from Dosovitskiy et al.

Dosovitskiy et al. An image is worth 16x16 words: Transformers for image recognition at scale, ICLR 2021

Transformers and Attention — Practical

Goals:

1. Implementing a Transformer encoder from scratch (following the architecture introduced in the *Attention is all you need* paper)
2. Understanding the involved computations
3. Building a full Deep Learning pipeline in PyTorch to train a model on a given dataset



Vaswani et al. Attention is all you need, NeurIPS 2017