Friedrich-Alexander-Universität
Technische Fakultät

Machine Learning
Data Analytics

Machine Learning for Time Series

(MLTS or MLTS-Deluxe Lectures)

Dr. Dario Zanca

Machine Learning and Data Analytics (MaD) Lab
Friedrich-Alexander-Universität Erlangen-Nürnberg

25.10.2022

Topics overview

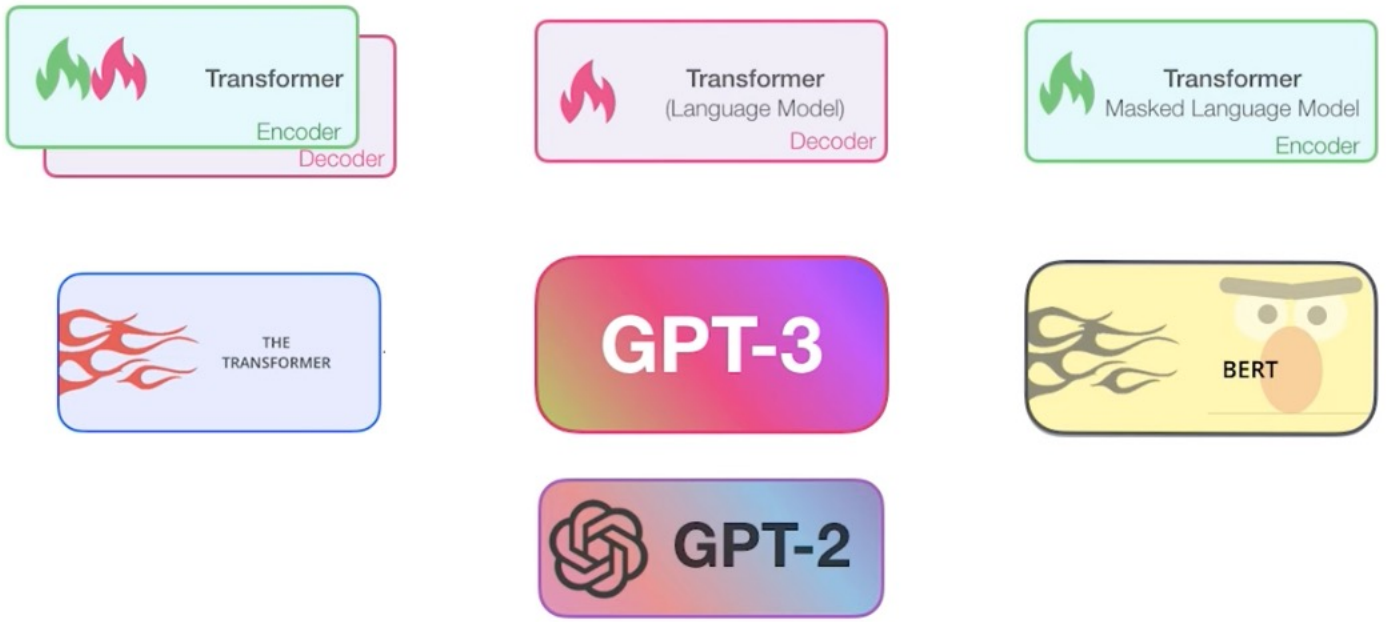
- Time series fundamentals and definitions (2 lectures)
- Bayesian Inference (1 lecture)
- Gussian processes (2 lectures)
- State space models (2 lectures)
- Autoregressive models (1 lecture)
- Data mining on time series (1 lecture)
- Deep learning on time series (4 lectures) ←
- Domain adaptation (1 lecture)

Recap: Recurrent neural networks

RNN / LSTM limitations:

- Non-parallelism → Long training time
- Difficulties with long sequences
  - Large memory usage
  - Difficult to train (vanishing/exploding gradients)
  - Hard to learn long-term dependencies (mitigated by LSTMs)

Motivation



- Transformers *perceive* the entire sequence at the same time.
- They are based on the seq2seq concept, i.e., transforming sequences into other sequences.
- State of the art in many NLP tasks.

https://www.youtube.com/watch?v=-QH8fRhqFHM&t=422s

In this lecture…

- Attention models
- The transformers architecture

Deep Learning for Time Series – Attention models

Attention models

Sequence-to-sequence models

Sequence-to-sequence (seq2seq) models aim at transforming an input sequence to an output sequence

- E.g., machine translation.

Seq2seq models generally have an encoder-decoder architecture, composed by:

- An encoder that processes the input sequence and compress the information into a context vector (also said embedding).
- A decoder that processes the context vector and produces the transformed output.

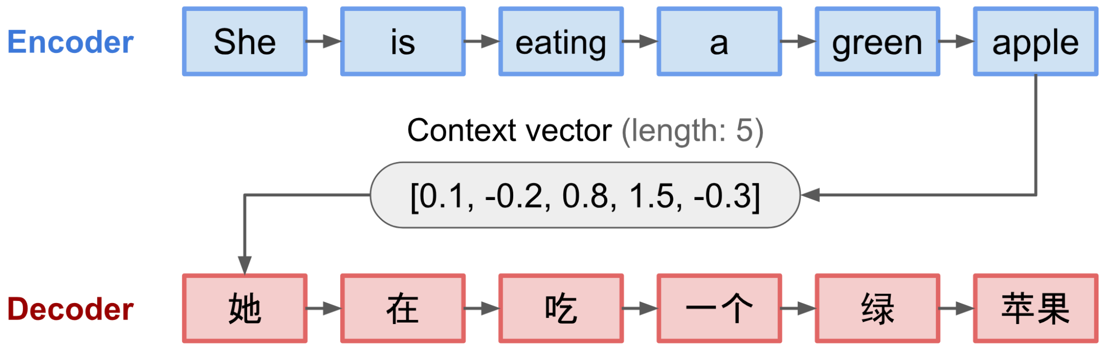
Sequence-to-sequence models

Disadvantages of the fixed length context vector design are:

- Incapability of remembering long sequences
- Too complex dynamics to be encoded in the hidden state

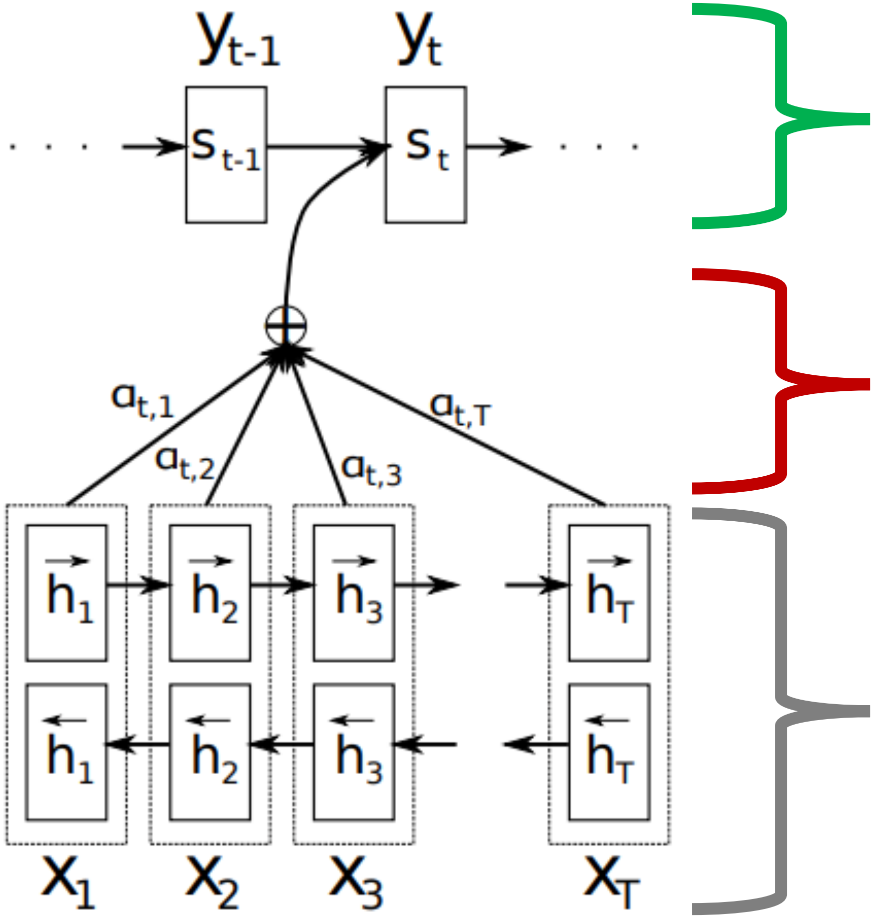
→ Attention mechanisms are proposed to solve this problem.

- Originated for machine translation [1]



[1] “Neural machine translation by jointly learning to align and translate”, Bahdanau et al.
Image from: https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

Attention mechanim



The **decoder** is an RNN, which receives a context vector as input

The **attention** (called "global alignment weights" by the authors) is a weighted sum parametrized by a FF-NN

The **encoder** is a bi-directional RNN

[1] "Neural machine translation by jointly learning to align and translate", Bahdanau et al.

Attention mechanim: formalization

Let x be the input sequence of length n, and y the output sequence of length m.

Let $h_i = [\overrightarrow{h_i}, \overleftarrow{h_i}]$ be the encoder state, given by the concatenation of the forward and backward hidden states of the bidirectional RNN.

Le denote with s_t the decoder state, for the output at position t. Then, the context vector is defined as the sum of the encoder states, weighted by the alignment scores, i.e.,

$$c_t = \sum_{i=1}^{n} a_{t,i} h_i$$

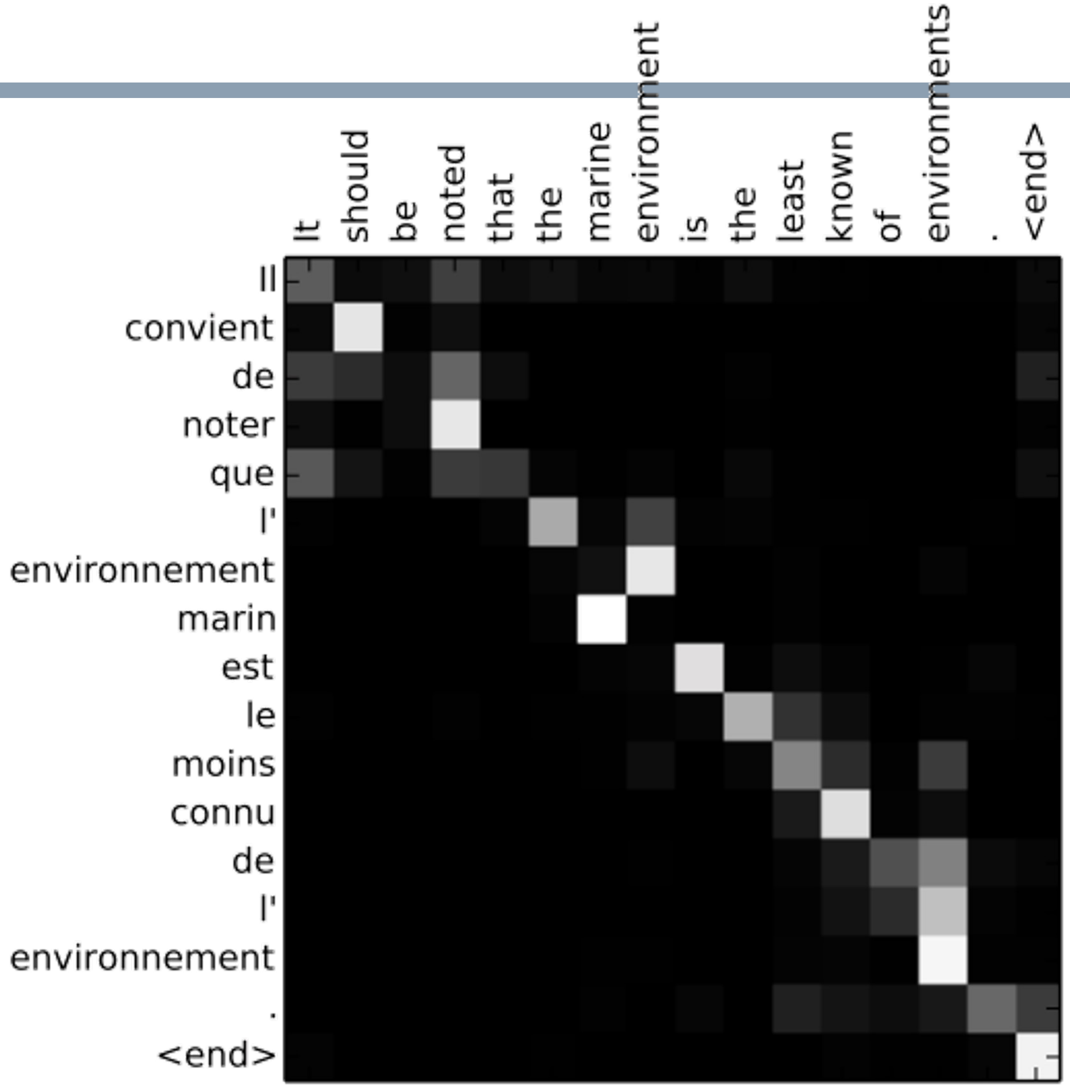
where $a_{t,i} = softmax(score(s_{t-1}, h_i))$, and $score(s_{t-1}, h_i) = \mathrm{v_a} \tanh(\mathrm{W_a}[s_{t-1}; h_i])$.

[1] “Neural machine translation by jointly learning to align and translate”, Bahdanau et al.

Attention mechanim: example

In the example on the side, the x-axis representes the source sentence and the y-axis the generated sentence.

Every pixel (i, j), at the i-th row and j-th column, indicates the weight the j-th source word embedding when generating the i-th word.



[1] “Neural machine translation by jointly learning to align and translate”, Bahdanau et al.

Attention mechanim

Name	Alignment score function
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.

Table from: https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.htm

Attention mechanism

Attention mechanisms can be characterised according to their function or design as:

- **Self-attention**
- **Soft/Hard attention**
- **Global/Local attention**

Global / local attention

Self-attention, also said intra-attention, is an attention mechanisms which relates different positions of an input sequence to generate a representation of the same sequence.

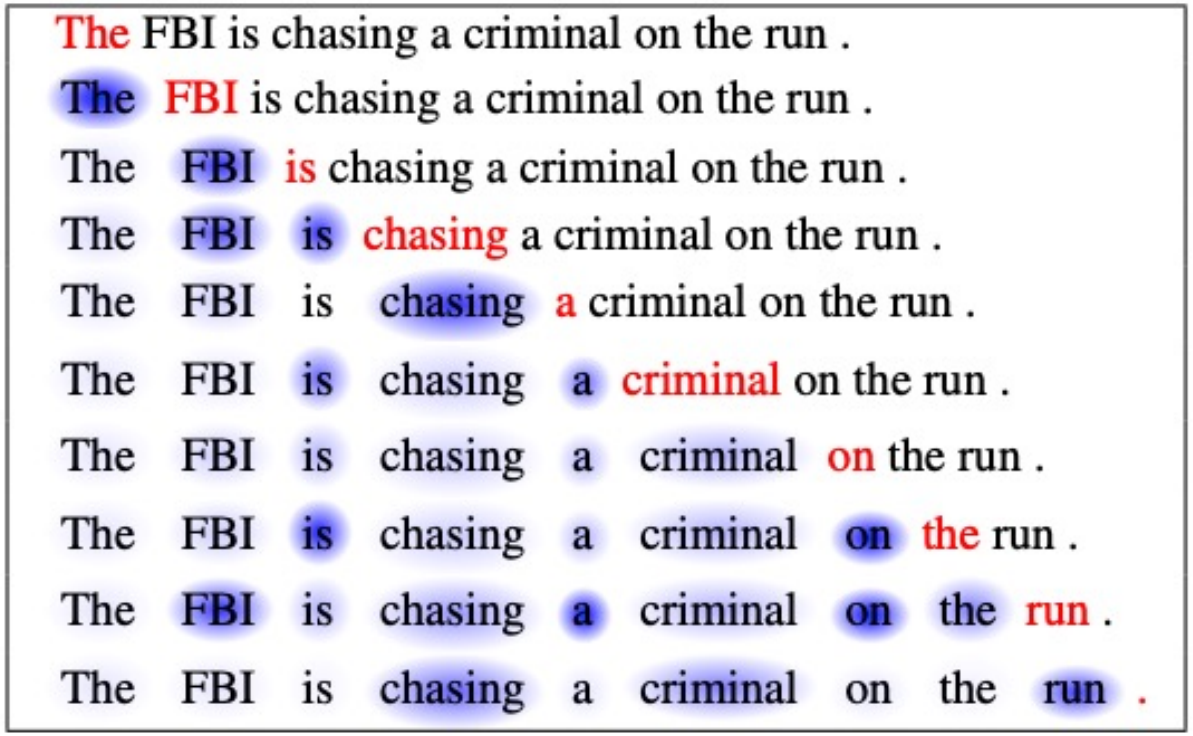


Figure from: https://arxiv.org/pdf/1601.06733.pdf

Soft / hard attention

Soft attention applies lerned weights to input parts, while **hard attention** select a single input part at the time.

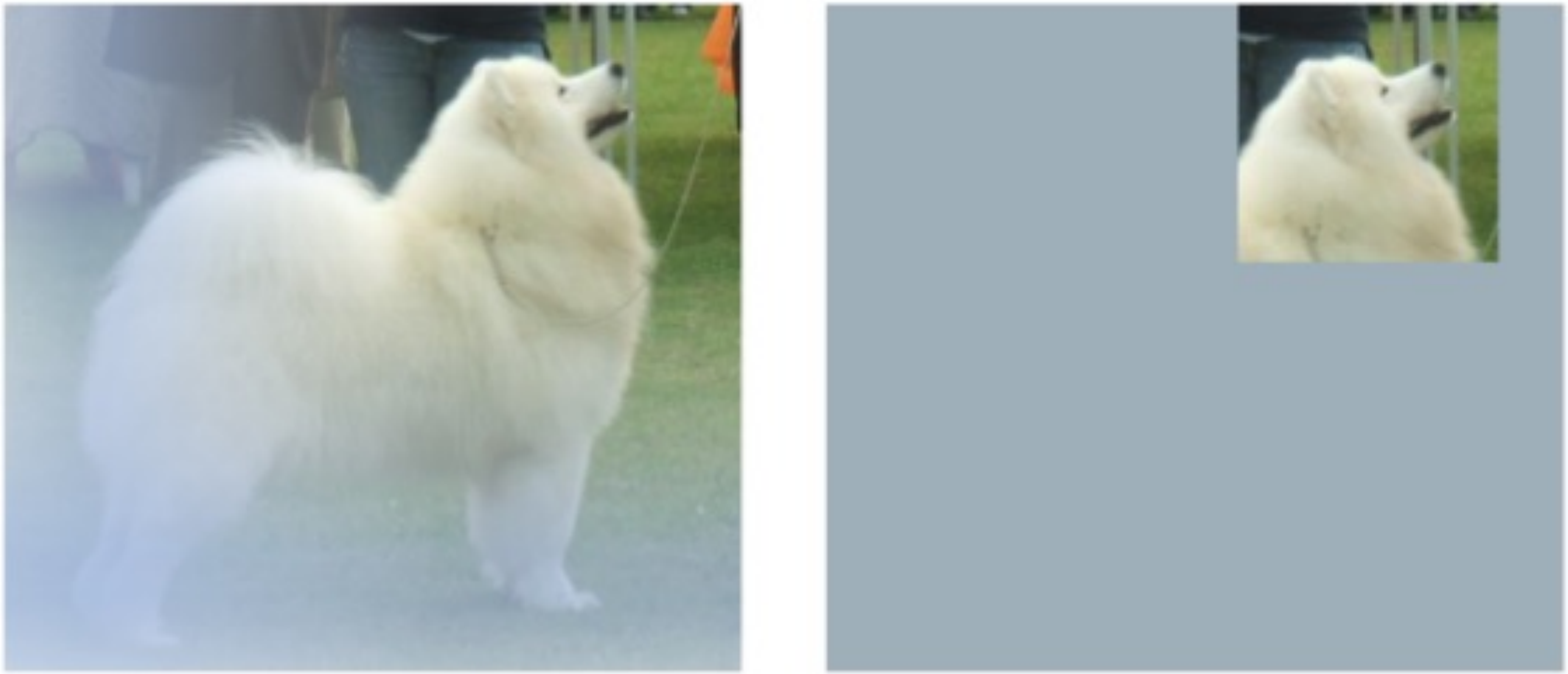


Image from: https://glassboxmedicine.com/2019/08/10/learn-to-pay-attention-trainable-visual-attention-in-cnns/

Global / local attention

In **global attention**, the context vector depends on the whole input sequence. The **local attention**, generates an input vector which depends only on a subset of the input sequence, corresponding to a window centered on the current position.

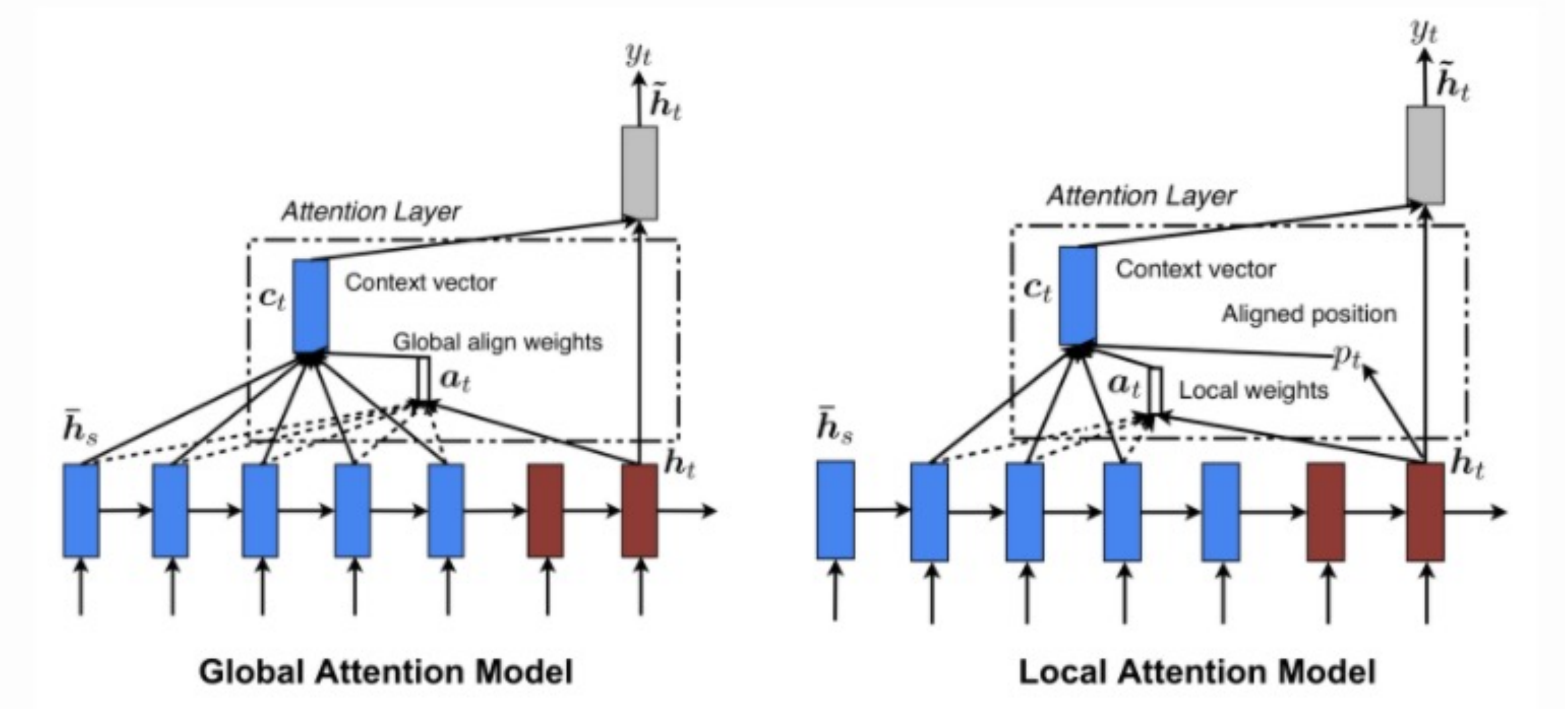


Figure from: https://arxiv.org/pdf/1508.04025.pdf

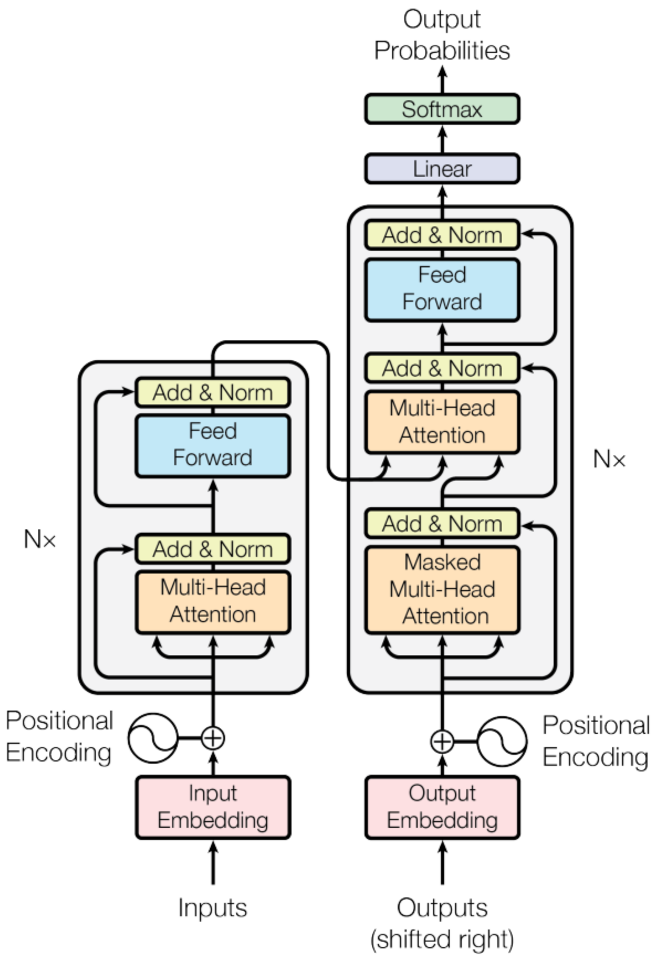
Deep Learning for Time Series – Attention models

The Transformer architecture

The Transformer architecture

The Transformer architecture, introduced in 2017 [2]:

- Completely built on self-attention
  - Do not use sequence aligned recurrent architecture
- Replaced all the RNNs

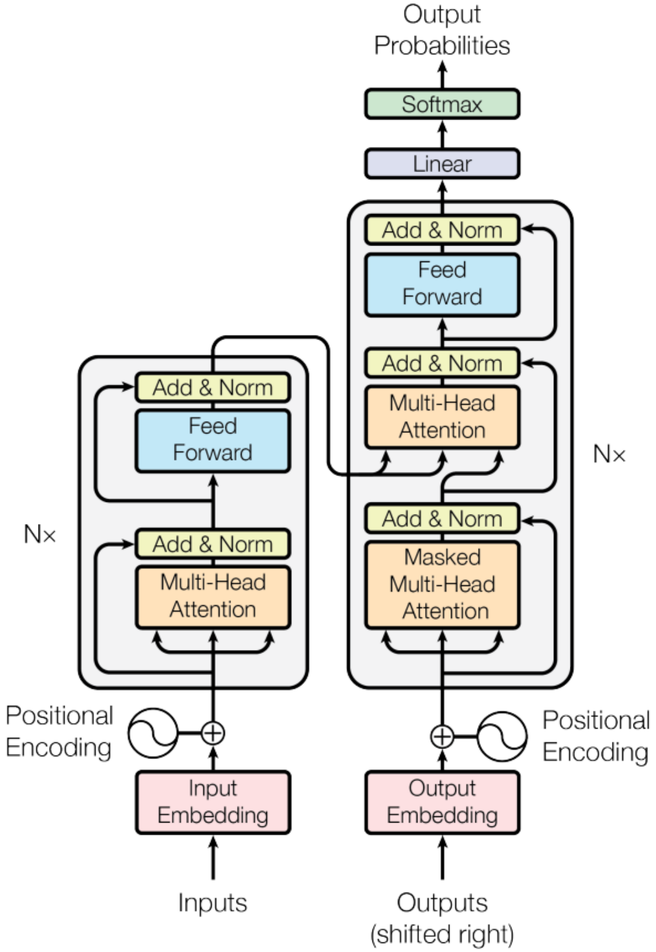


[2] “Attention is all you need”, Vaswani, et al.

The Transformer architecture

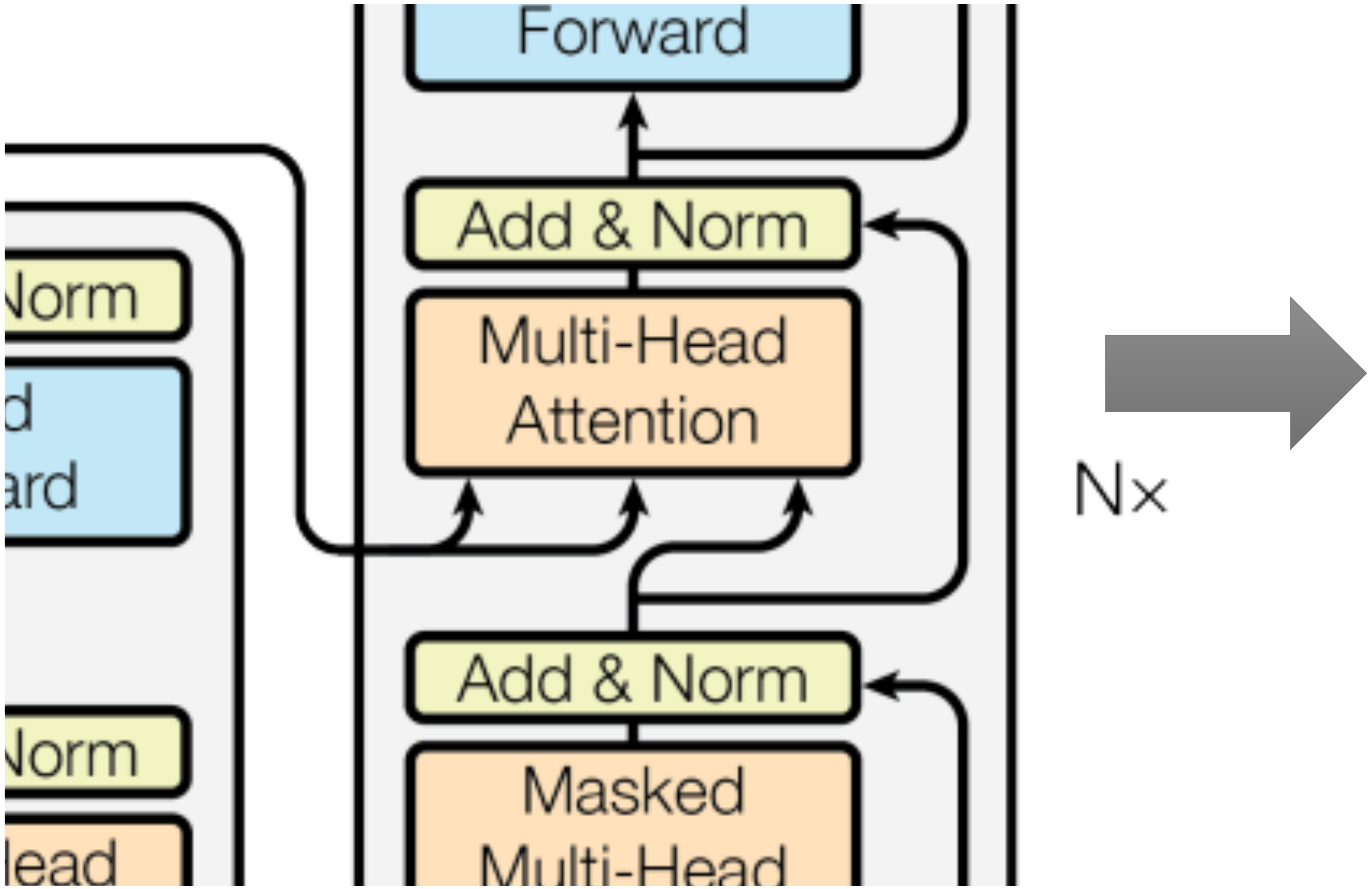
The fundamentals components of the transformer architectures are:

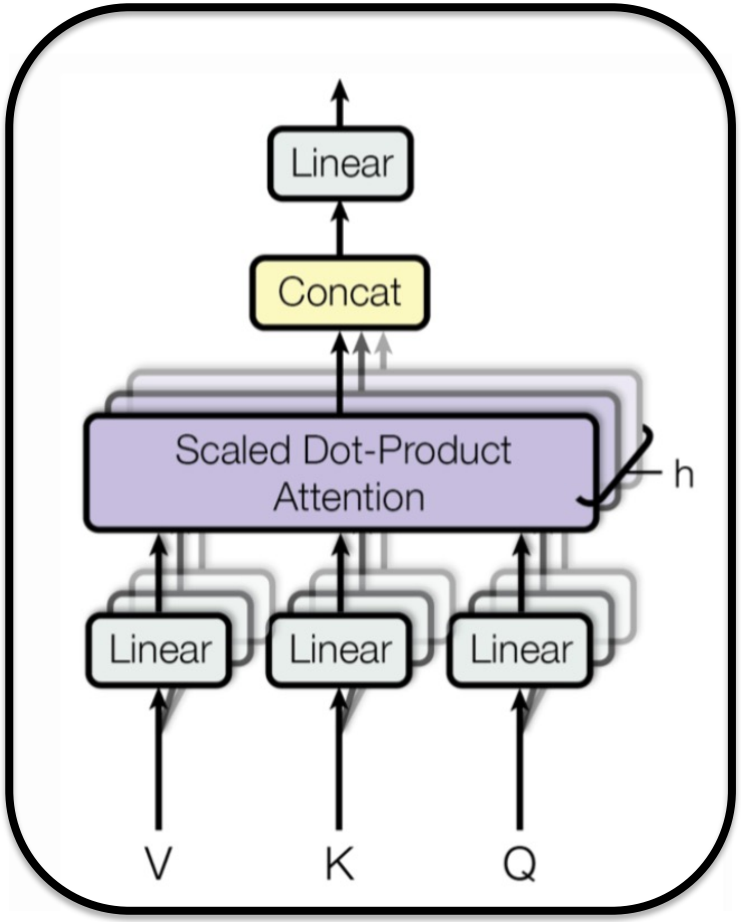
- Positional encoding
- Multi-head self attention
    - Based on K, V, and Q matrices
- An encoder-decoder architecture



[2] “Attention is all you need”, Vaswani, et al.

Transformer: multi-head attention

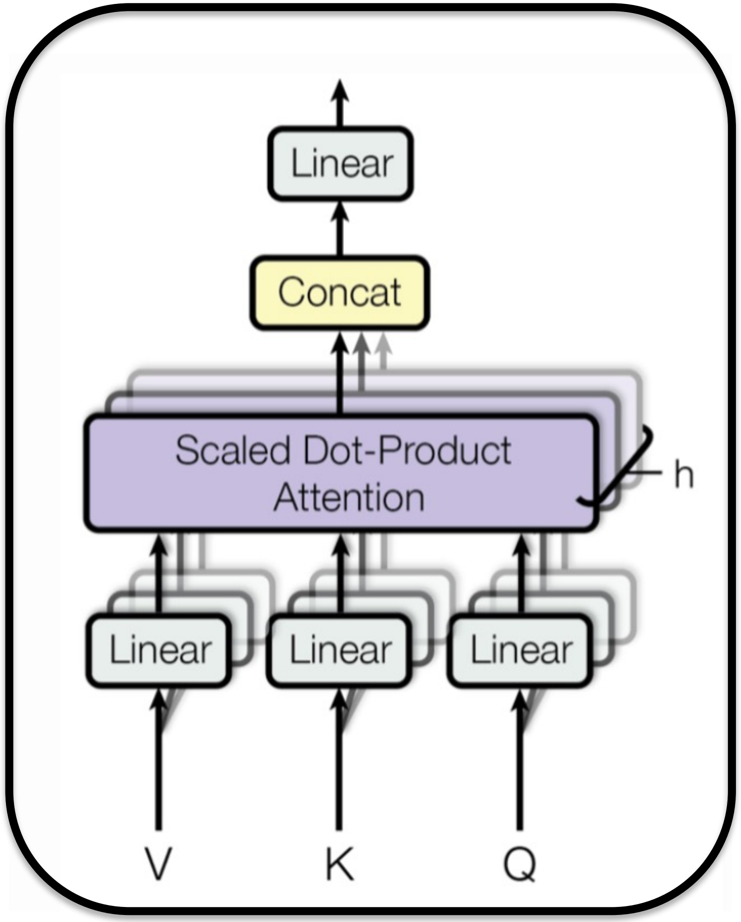




Multi-head attention

[2] “Attention is all you need”, Vaswani, et al.

Transformer: multi-head self-attention



Multi-head attention

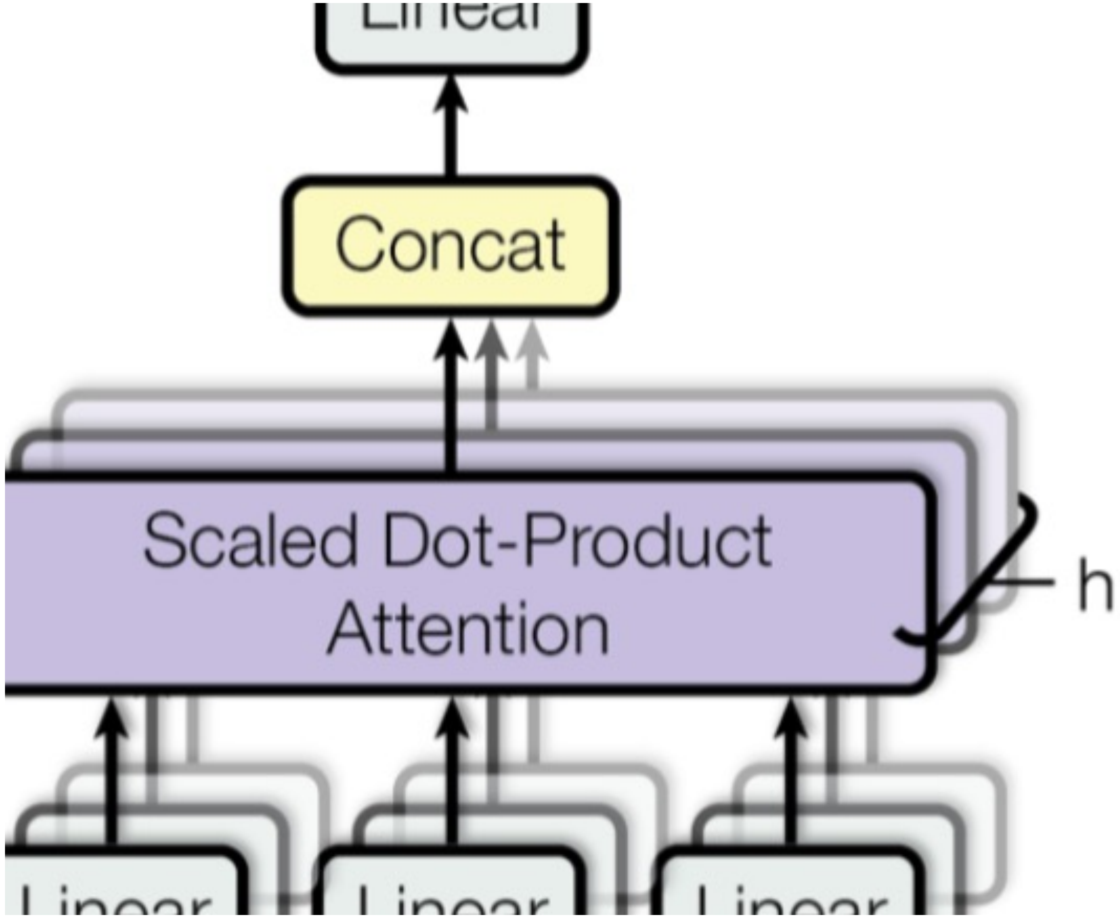
[2] “Attention is all you need”, Vaswani, et al.

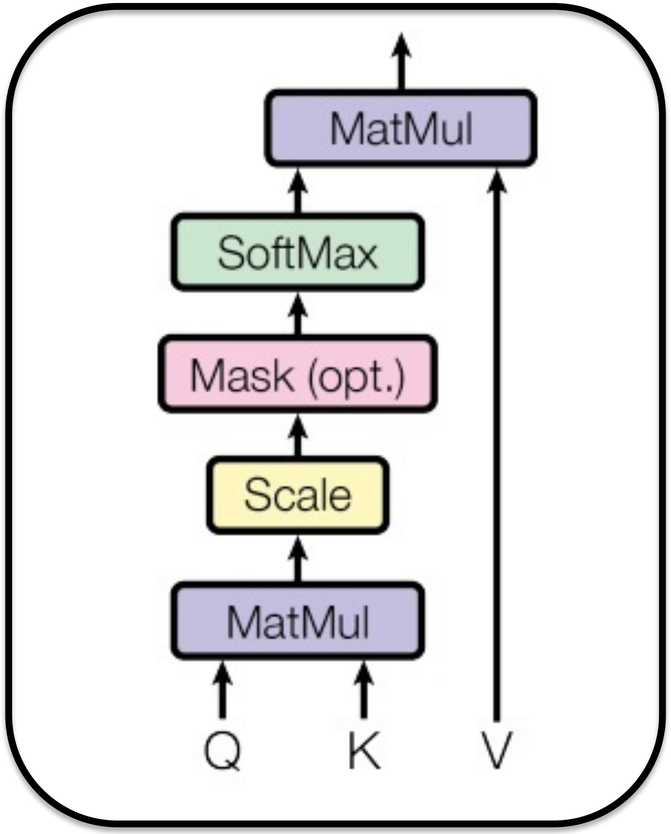
The first step in calculating self-attention consists of calculating three vectors from each input vector (i.e., each element of the input sequence).

- Query, $\boldsymbol{Q}$
- Key, $\boldsymbol{K}$
- Value, $\boldsymbol{V}$

These vector are calculated by multiplying the input vector by a corresponding matrix, resp., W^Q, W^K, and W^V.

Transformer: multi-head self-attention

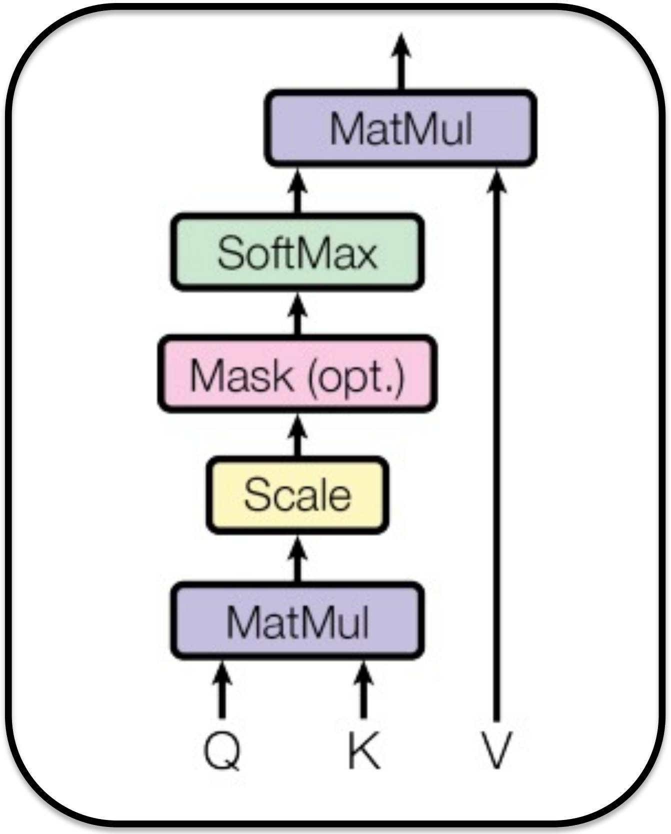




Scaled Dot-Product Attention

[2] “Attention is all you need”, Vaswani, et al.

Transformer: multi-head self-attention



Scaled Dot-Product Attention

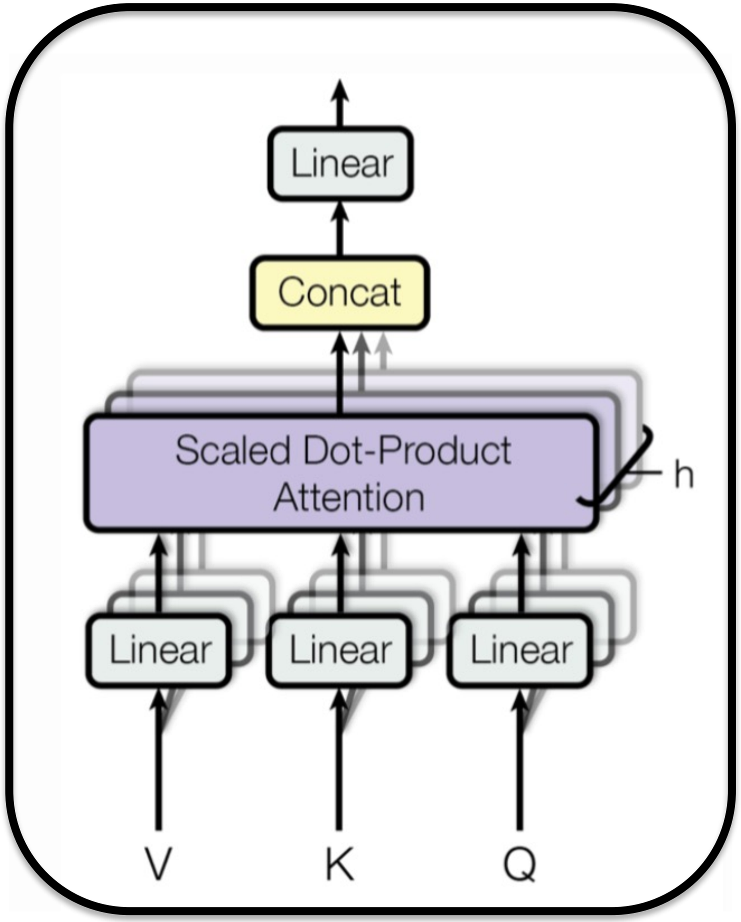
Then, the scaled dot-product attention is computed by

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

where d_k is the dimension of queries and keys.

[2] "Attention is all you need", Vaswani, et al.

Transformer: multi-head self-attention



Multi-head attention

[2] "Attention is all you need", Vaswani, et al.

Instead of performing a single attention function with a d_{model}-dimensional keys, values and queries, it is beneficial to linearly project queries, keys and values with **different linaer projections**.

Each of this projections is processed **in parallel** and then **concatenated**.

$MultiHead(Q, K, V) = concat(head_1, \ldots, head_h)$

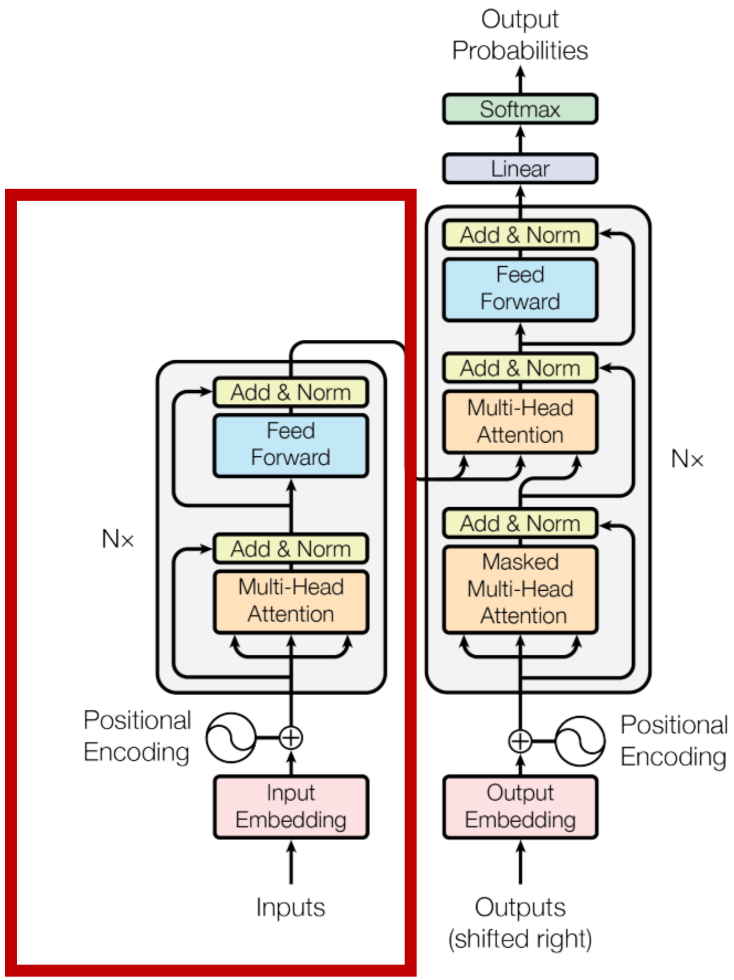
where each of the heads is a scaled dot-product attention, i.e.,

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

Transformer: the encoder

The encoder generates an attention-based representation of the input sequence.

- Capability of locate a specific information from a (potentially) very large context.
- It is made of N_x identical layers,, each composed of
  - A multi-head attention layer
  - A positional FF-NN
  - A residual connection and layer normalization

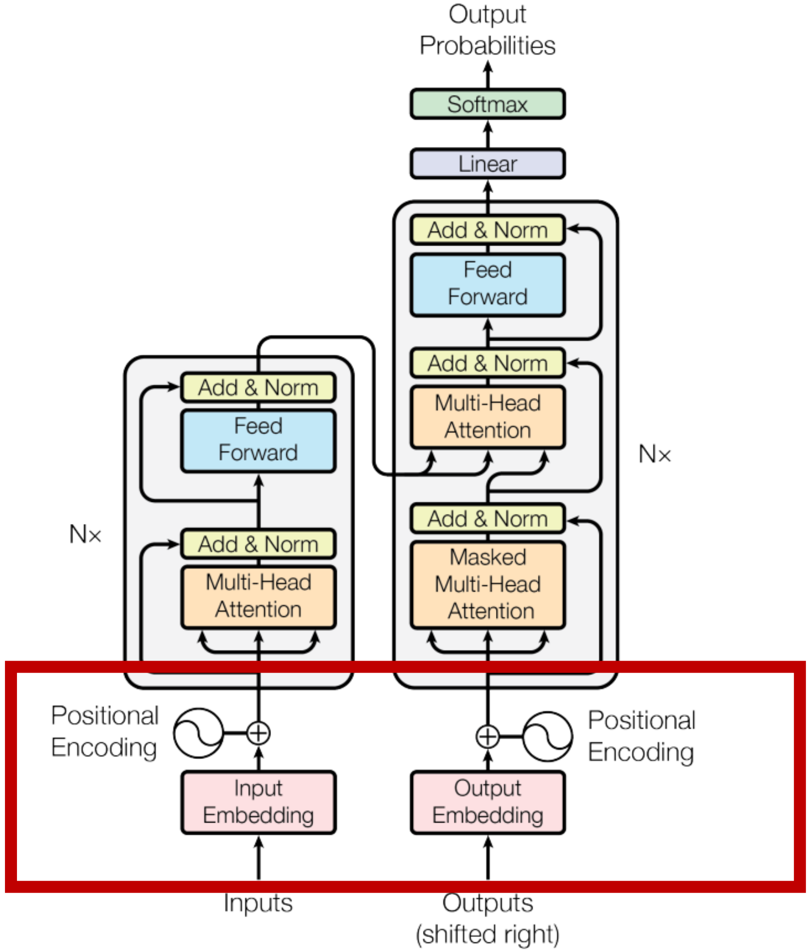


[2] “Attention is all you need”, Vaswani, et al.

Transformer: the positional encoding

Since the model contains no recurrence nor convolution, in order to make use of the order of the sequence, positional information is injected by mean of a **positional encoding.**

The positional encoding is **added to the input vectors** (for both the encoder and the decoder networks).



[2] “Attention is all you need”, Vaswani, et al.

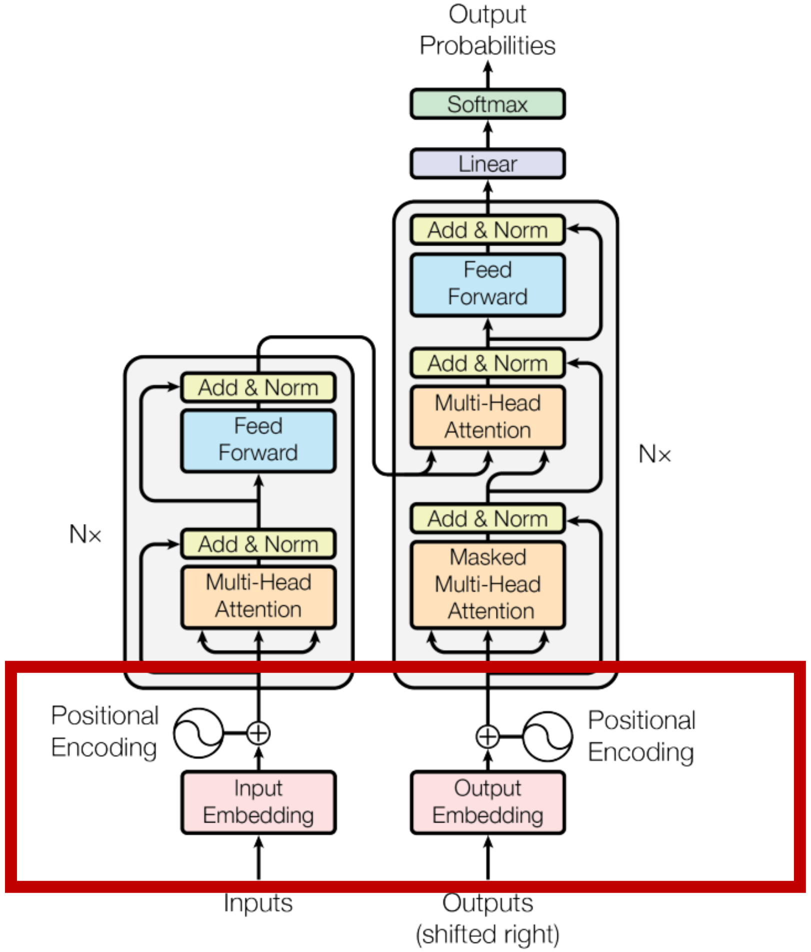
Transformer: the positional encoding

In the original implementations, the positional encoding is defined by using sine and cosine functions of different frequencies:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

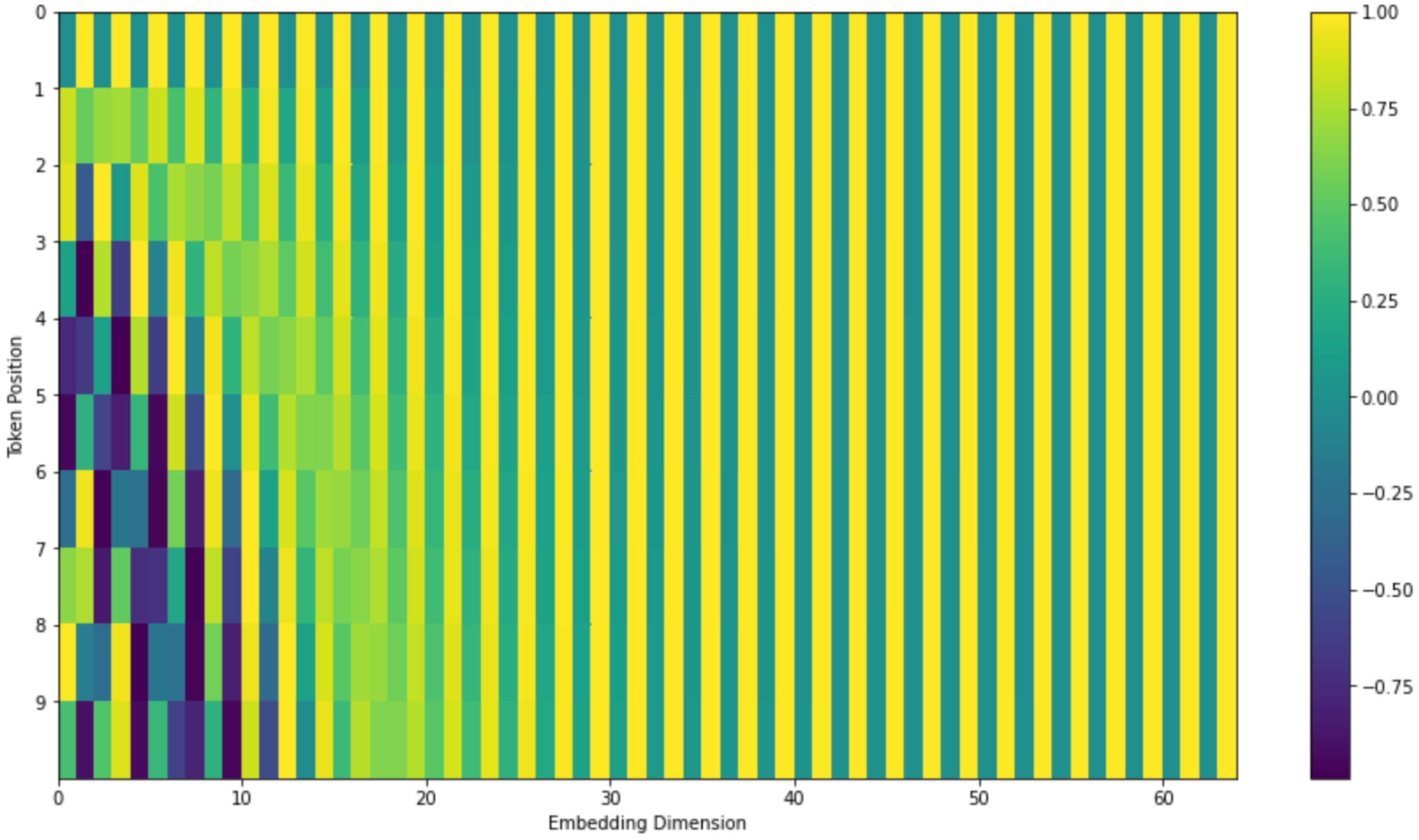
$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

where pos is the position and i is the dimension.



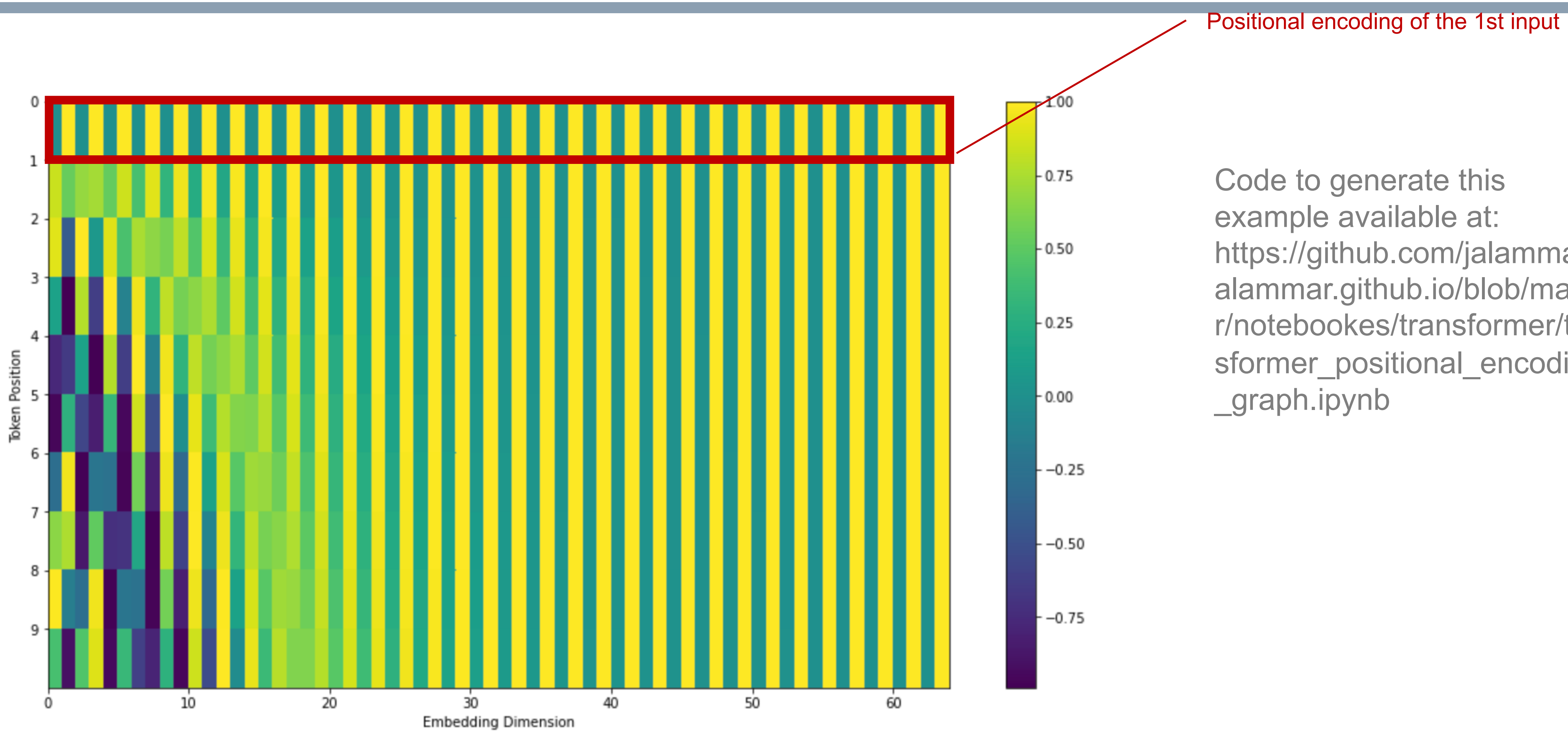
[2] “Attention is all you need”, Vaswani, et al.

Transformer: the positional encoding



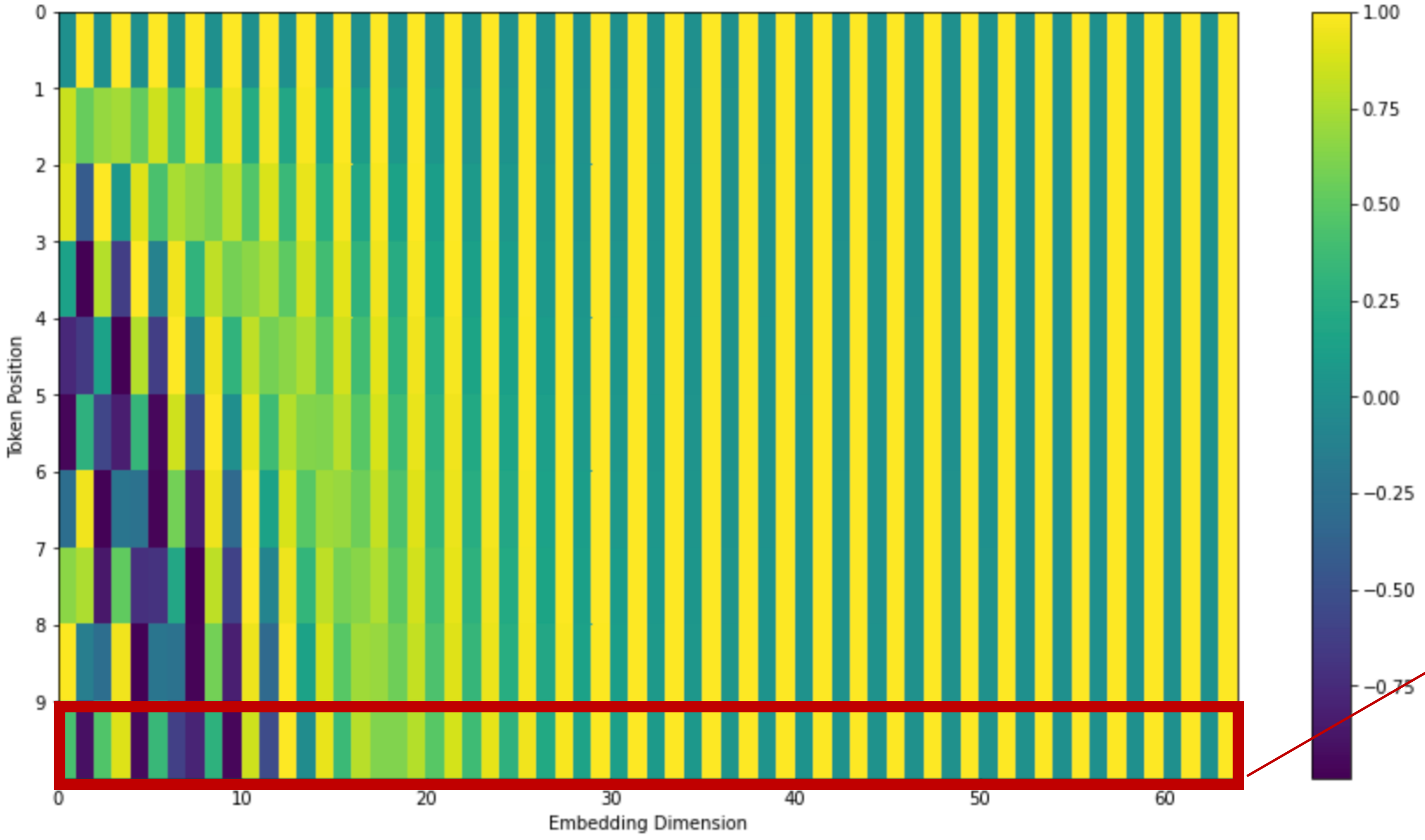
Code to generate this example available at: https://github.com/jalammar/jalammar.github.io/blob/master/notebookes/transformer/transformer_positional_encoding_graph.ipynb

Transformer: the positional encoding



Code to generate this example available at: https://github.com/jalammar/jalammar.github.io/blob/master/notebookes/transformer/transformer_positional_encoding_graph.ipynb

Transformer: the positional encoding



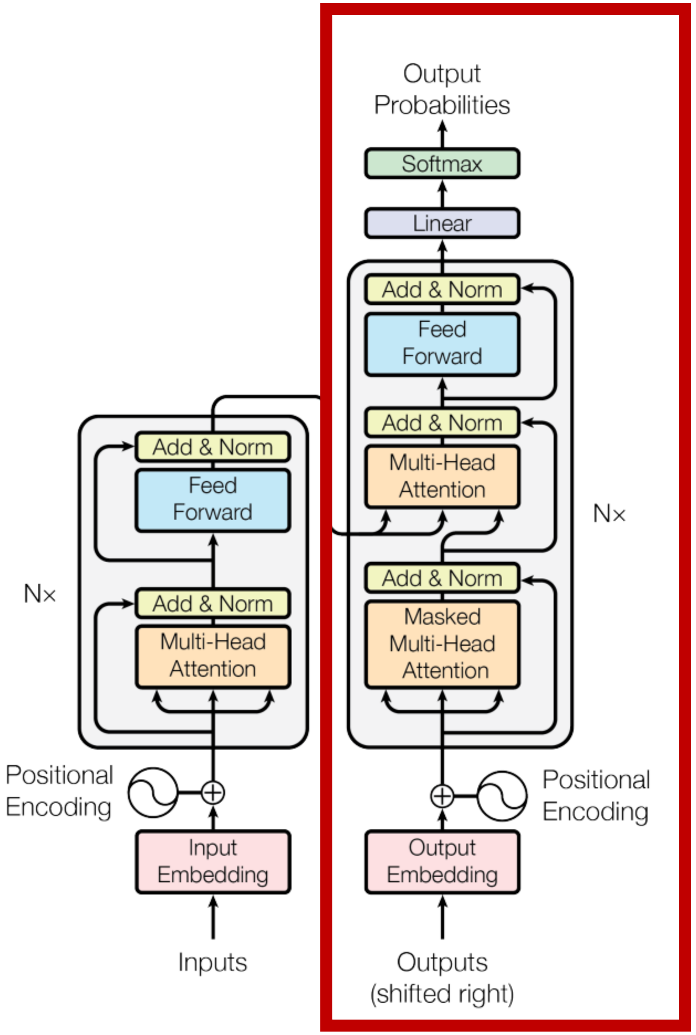
Positional encoding of the 10th input

Code to generate this example available at: https://github.com/jalammar/jalammar.github.io/blob/master/notebookes/transformer/transformer_positional_encoding_graph.ipynb

Transformer: the decoder

The decoder network is able to retrival information from the encoded representation.

- As the encoder, it is made of N_x identical layers,, each composed of
  - A multi-head attention layer
  - A positional FF-NN
  - A residual connection and layer normalization
- The first multi-head self-attention is properly masked to prevent information leakage from future positions.



[2] "Attention is all you need", Vaswani, et al.

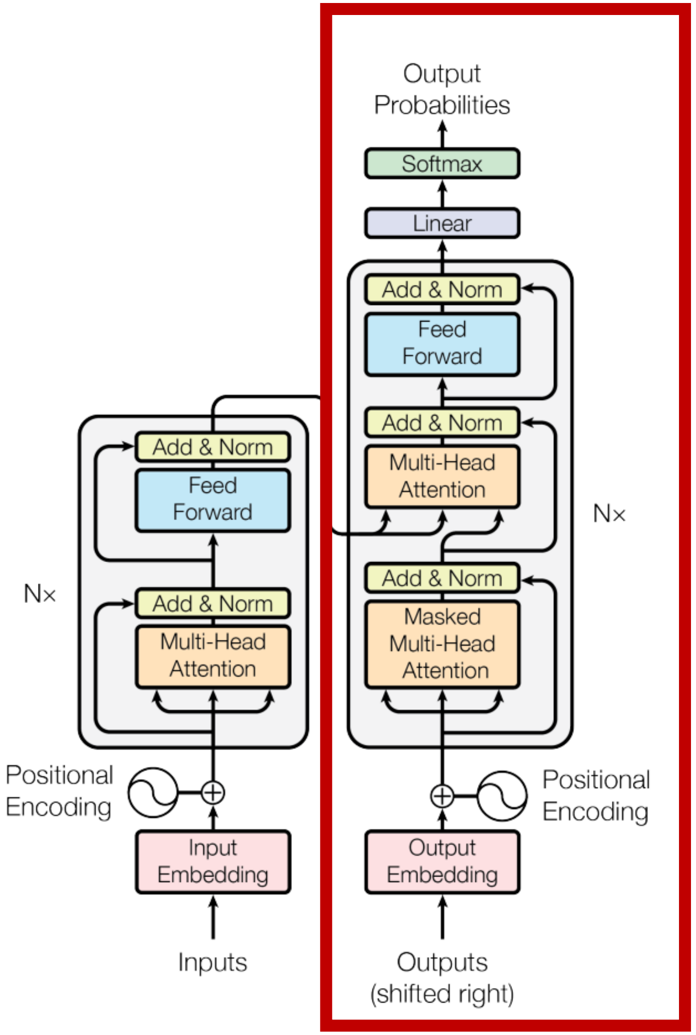
Transformer: the decoder

The deconding works as follows:

1. The output of the top encoder is transformed into a set of attention vectors $\boldsymbol{K}$ and $\boldsymbol{V}$.

2. These are used by each decoder
→ It helps the decoder focus on appropriate parts

3. The process is repeated until a special symbol of <end> is generated.



[2] “Attention is all you need”, Vaswani, et al.

Lecture title

Recap

In this lecture

- Attention models
    - Seq2Seq
    - Encoder-decoder
        - All hidden states are passed
    - Self-attention, soft/hard, local/global
- Transformers
    - Multi-head Self-attention
    - Positional encoding
    - Encoder and decoder networks

Transformers: pros and cons

Pros:

- Transformers can learn direct access to potentially very far input parts
- Many degree of freedom → Can learn complex dependecies
- Feed-forward model provides high performance on moder hardware

Cons:

- Quadratic time and memory complexity
- Many degree of freedom → Data hugry behavior during training
- Training is insidious
  - Hard integration with other architecture, not easy learning rate policy, use custom data loader, …
- Still limited research on time series data.

Other attention-based approaches

- Neural Turing Machines [3]
    - Couples NNs with external memory
    - Mimics reading and writing operations in Turing machines
    - Exploit content-based attention
- Pointer networks [4]
    - Solves problems where the outputs are positions in an input sequence
    - It applies attention over the input elements to pick one as the output at each decoder step.
- …