Deep Learning

Lecture 7: Attention and transformers

Prof. Gilles Louppe
g.louppe@uliege.be

Today

Attention is all you need!

- Encoder-decoder
- Bahdanau attention
- Attention layers
- Transformers

Encoder-decoder

Many real-world problems require to process a signal with a sequence structure.

- Sequence classification:
    - sentiment analysis in text
    - activity/action recognition in videos
    - DNA sequence classification
- Sequence synthesis:
    - text synthesis
    - music synthesis
    - motion synthesis
- Sequence-to-sequence translation:
    - speech recognition
    - text translation
    - time series forecasting

Given a set $\mathcal{X}$, if $S(\mathcal{X})$ denotes the set of sequences of elements from $\mathcal{X}$,

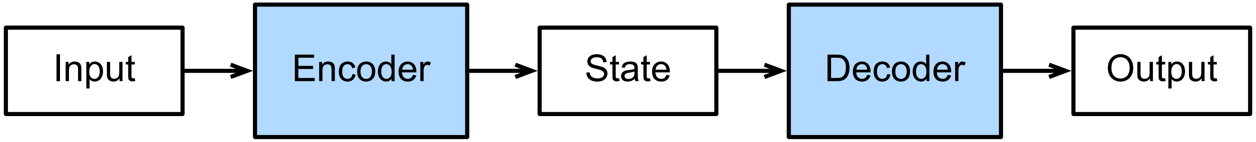
$$S(\mathcal{X}) = \cup_{t=1}^{\infty} \mathcal{X}^t,$$

then we formally define:

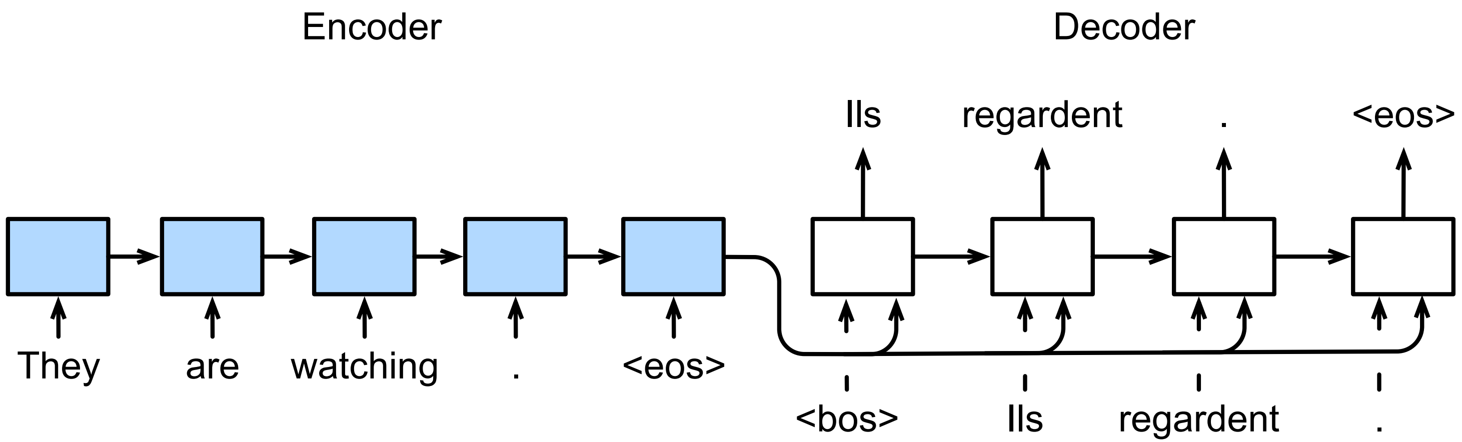
Sequence classification	$f : S(\mathcal{X}) \to \triangle^C$
Sequence synthesis	$f : \mathbb{R}^d \to S(\mathcal{X})$
Sequence-to-sequence translation	$f : S(\mathcal{X}) \to S(\mathcal{Y})$

In the rest of the slides, we consider only time-indexed signal, although it generalizes to arbitrary sequences.

When the input is a sequence $\mathbf{x} \in S(\mathbb{R}^p)$ of variable length, the historical approach is to use a recurrent **encoder-decoder** architecture that first compresses the input into a single vector v and then uses it to generate the output sequence.



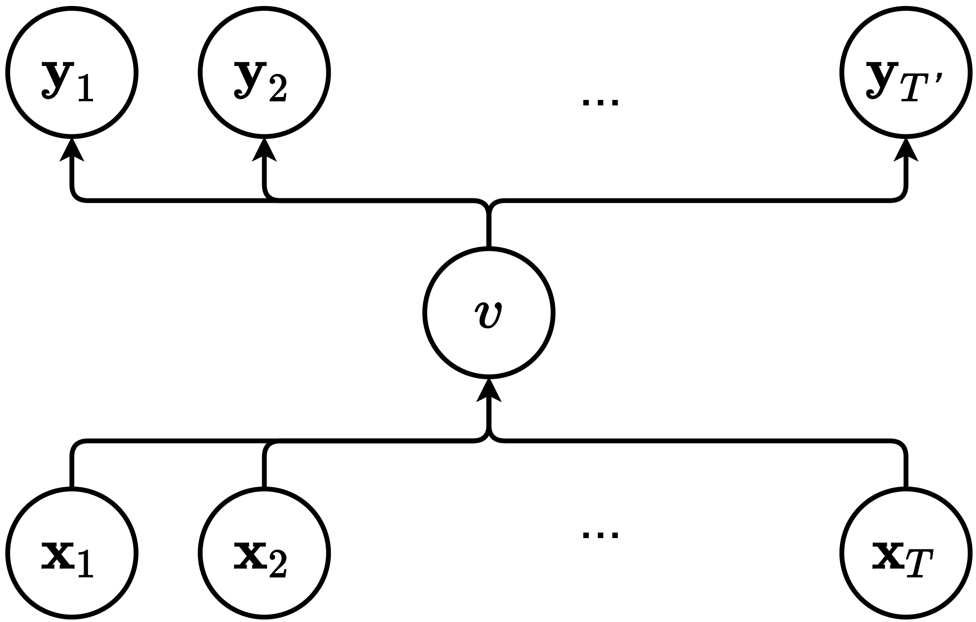
Credits: Dive Into Deep Learning, 2023.



Recurrent encoder-decoder models compress an input sequence $\mathbf{x}_{1:T}$ into a single thought vector v, and then produce an output sequence $\mathbf{y}_{1:T'}$ from an autoregressive generative model

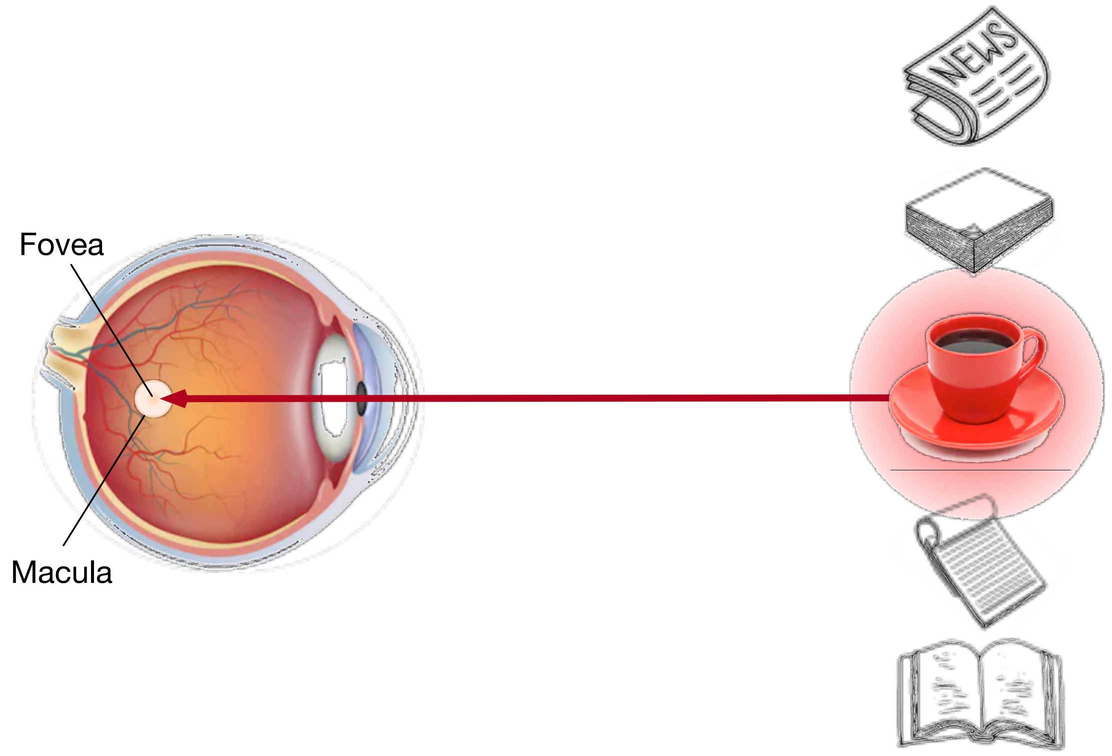
$$\begin{aligned}\mathbf{h}_t &= \phi(\mathbf{x}_t, \mathbf{h}_{t-1})\\ v &= \mathbf{h}_T\\ \mathbf{y}_i &\sim p(\cdot|\mathbf{y}_{1:i-1}, v).\end{aligned}$$

—
Credits: Dive Into Deep Learning, 2023.



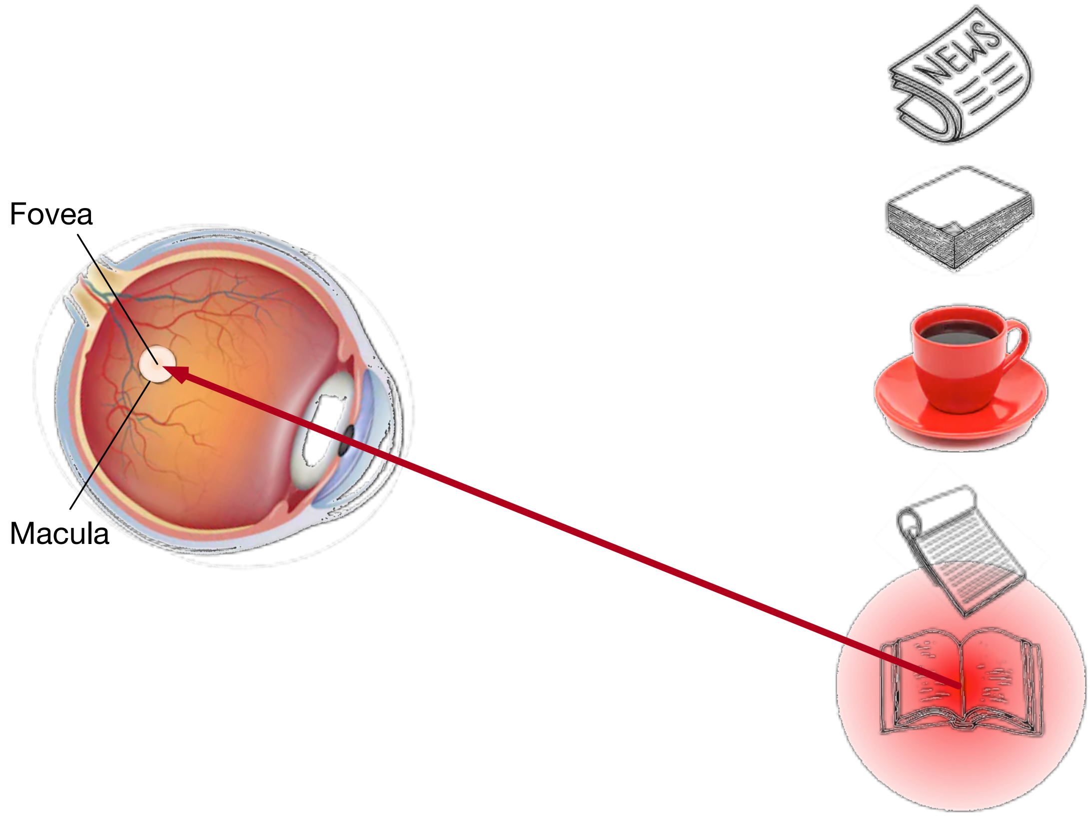
This architecture assumes that the sole vector v carries enough information to generate entire output sequences. This is often challenging for long sequences.

Bahdanau attention



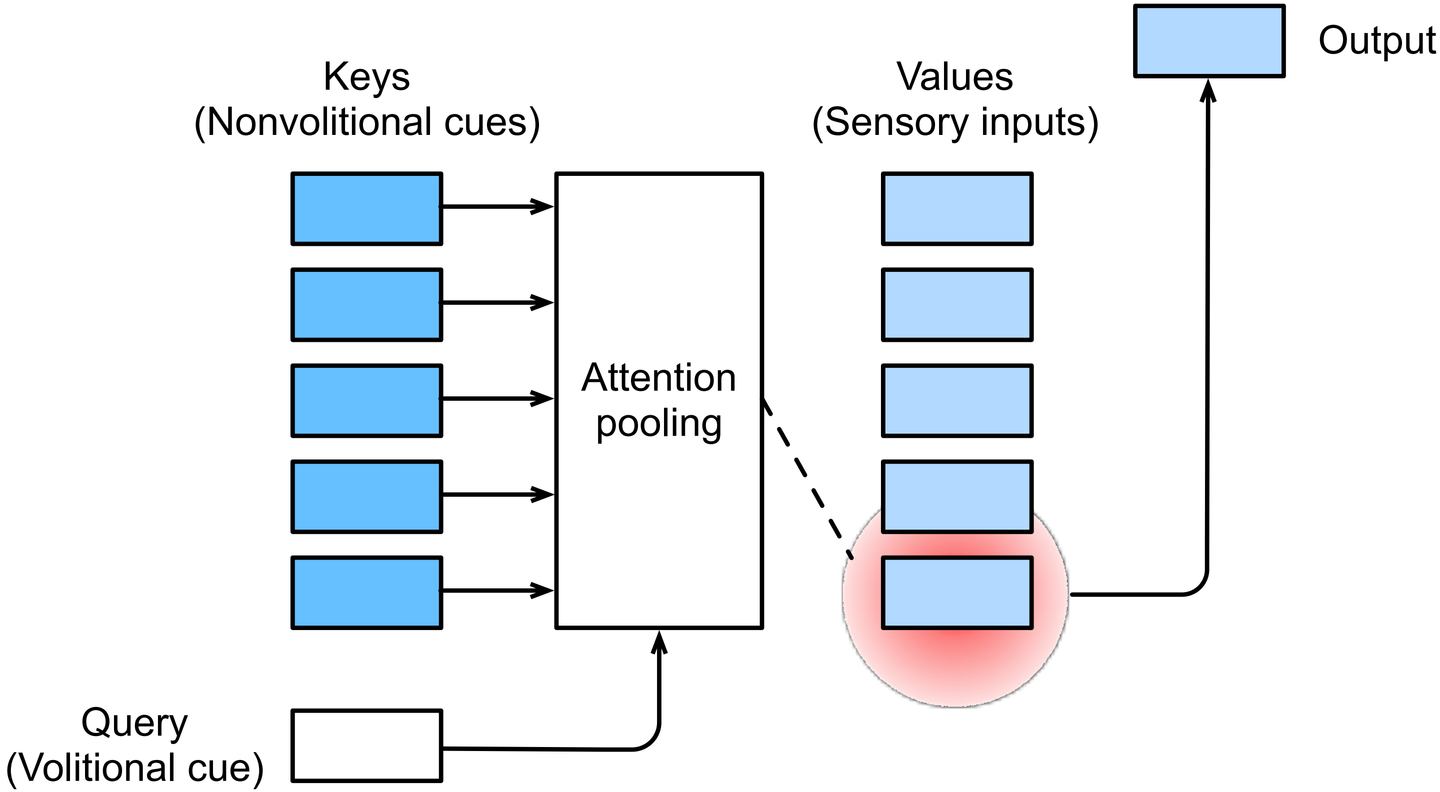
Using the nonvolitional cue based on saliency (red cup, non-paper), attention is involuntarily directed to the coffee.

—
Credits: Dive Into Deep Learning, 2023.



Using the volitional cue (want to read a book) that is task-dependent, attention is directed to the book under volitional control.

—
Credits: Dive Into Deep Learning, 2023.

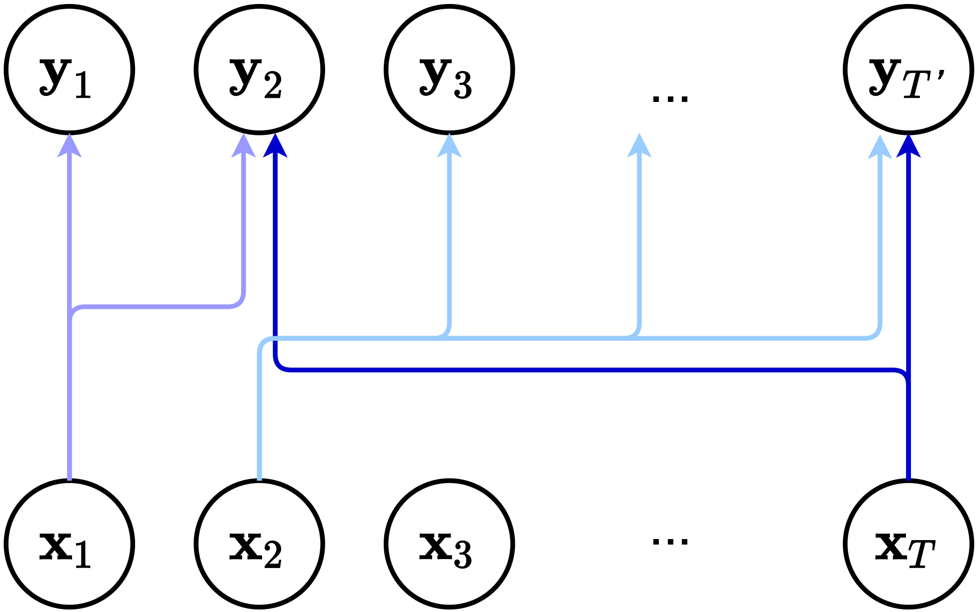


—

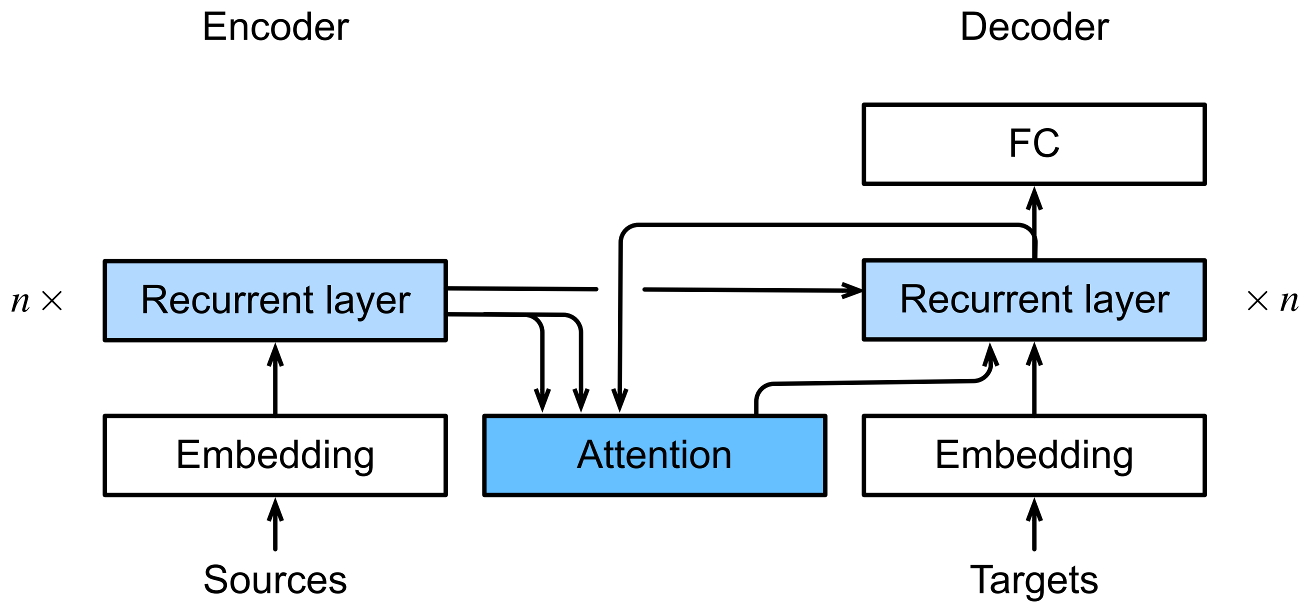
Credits: Dive Into Deep Learning, 2023.

Attention mechanisms can transport information from parts of the input signal to parts of the output **specified dynamically**.

Under the assumption that each output token comes from one or a handful of input tokens, the decoder should attend to only those tokens that are relevant for producing the next output token.



Attention-based machine translation



—

Following Bahdanau et al. (2014), the encoder is specified as a bidirectional recurrent neural network (RNN) that computes an annotation vector for each input token,

$$\mathbf{h}_j = (\overrightarrow{\mathbf{h}}_j, \overleftarrow{\mathbf{h}}_j)$$

for $j = 1, \ldots, T$, where $\overrightarrow{\mathbf{h}}_j$ and $\overleftarrow{\mathbf{h}}_j$ respectively denote the forward and backward hidden recurrent states of the bidirectional RNN.

From this, they compute a new process $\mathbf{s}_i$, $i = 1, \ldots, T'$, which looks at weighted averages of the $\mathbf{h}_j$ where the **weights are functions of the signal**.

Given $\mathbf{y}_1, \ldots, \mathbf{y}_{i-1}$ and $\mathbf{s}_1, \ldots, \mathbf{s}_{i-1}$, first compute an attention vector

$$\alpha_{i,j} = \mathrm{softmax}_j(a(\mathbf{s}_{i-1}, \mathbf{h}_j))$$

for $j = 1, \ldots, T$, whered a is an **attention scoring function**, here specified as a one hidden layer $\tanh$ MLP.

Then, compute the context vector from the weighted $\mathbf{h}_j$'s,

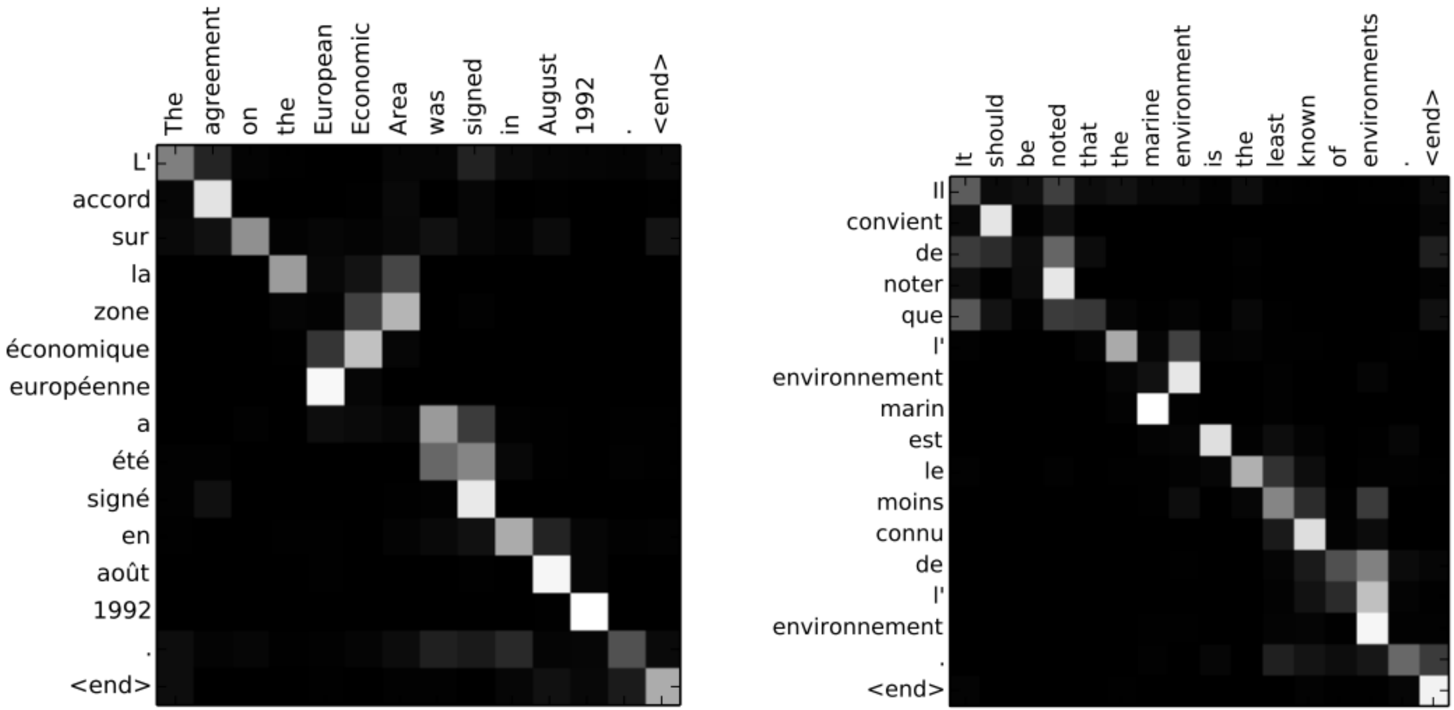
$$\mathbf{c}_i = \sum_{j=1}^{T} \alpha_{i,j} \mathbf{h}_j .$$

The model can now make the prediction $\mathbf{y}_i$ as

$$\begin{aligned}\mathbf{s}_i &= f(\mathbf{s}_{i-1}, y_{i-1}, c_i) \\ \mathbf{y}_i &\sim g(\mathbf{y}_{i-1}, \mathbf{s}_i, \mathbf{c}_i),\end{aligned}$$

where f is a GRU.

This is context attention, where $\mathbf{s}_{i-1}$ modulates what to look in $\mathbf{h}_1, \ldots, \mathbf{h}_T$ to compute $\mathbf{s}_i$ and sample $\mathbf{y}_i$.


The
agreement
on
the
European
Economic
Area
was
signed
in
August
1992
.
<end>
L'
accord
sur
la
zone
économique
européenne
a
été
signé
en
août
1992
.
<end>
It
should
be
noted
that
the
marine
environment
is
the
least
known
of
environments
.
<end>
Il
convient
de
noter
que
l'
environnement
marin
est
le
moins
connu
de
l'
environnement
.
<end>

(a) (b)

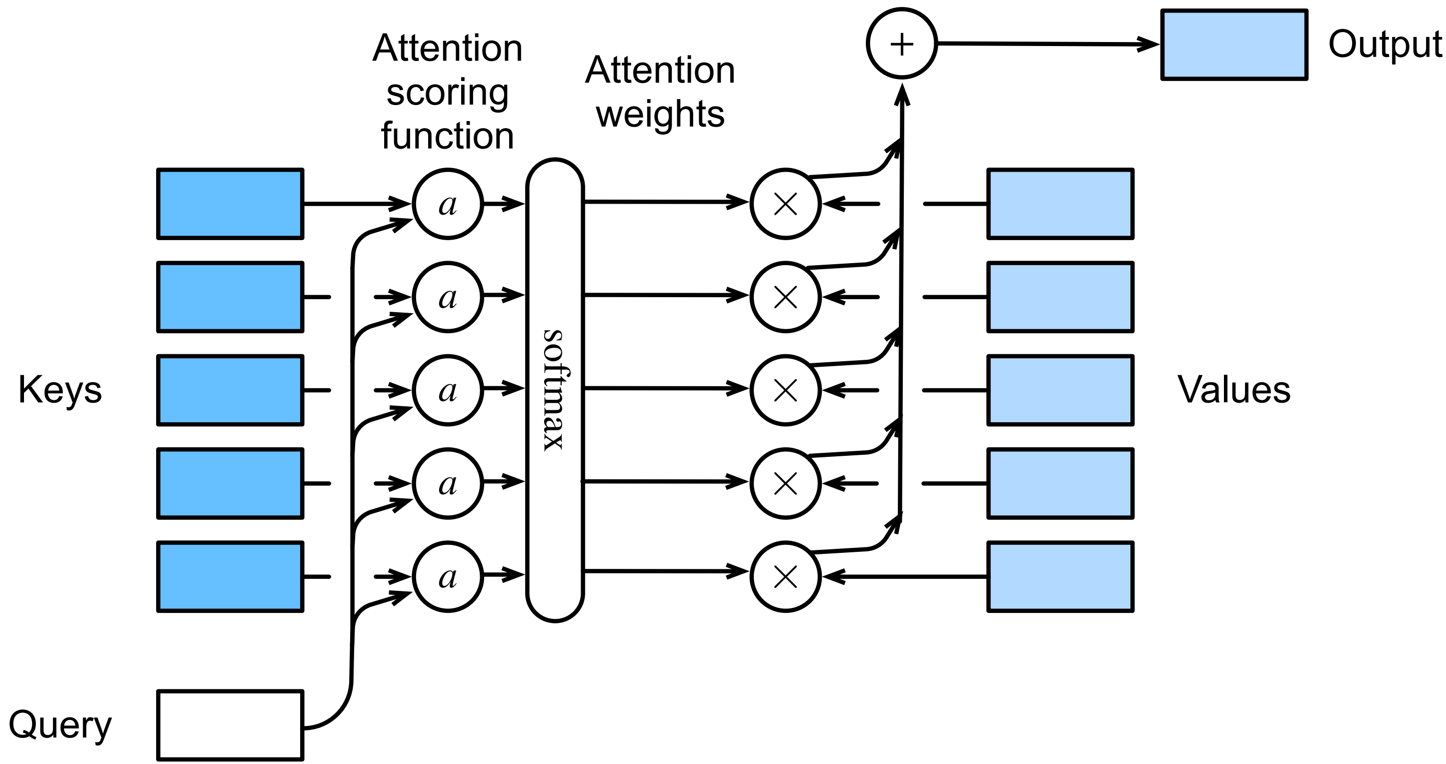
Attention layers

The attention mechanisms can be defined generically as follows.

Given a context or query vector $\mathbf{q} \in \mathbb{R}^q$, a key tensor $\mathbf{K} \in \mathbb{R}^{m \times k}$, and a value tensor $\mathbf{V} \in \mathbb{R}^{m \times v}$, an attention layer computes an output vector $\mathbf{y} \in \mathbb{R}^v$ with

$$\mathbf{y} = \sum_{i=1}^{m} \mathrm{softmax}_i(a(\mathbf{q}, \mathbf{K}_i; \theta))\mathbf{V}_i,$$

where $a : \mathbb{R}^q \times \mathbb{R}^k \to \mathbb{R}$ is a scalar attention scoring function.


Attention
scoring
function
Attention
weights
Output
Keys
softmax
Values
Query

Additive attention

When queries and keys are vectors of different lengths, we can use an additive attention as the scoring function.

Given $\mathbf{q} \in \mathbb{R}^q$ and $\mathbf{k} \in \mathbb{R}^k$, the additive attention scoring function is

$$a(\mathbf{q}, \mathbf{k}) = \mathbf{w}_v^T \tanh(\mathbf{W}_q^T \mathbf{q} + \mathbf{W}_k^T \mathbf{k})$$

where $\mathbf{w}_v \in \mathbb{R}^h$, $\mathbf{W}_q \in \mathbb{R}^{q \times h}$ and $\mathbf{W}_k \in \mathbb{R}^{k \times h}$ are learnable parameters.

Scaled dot-product attention

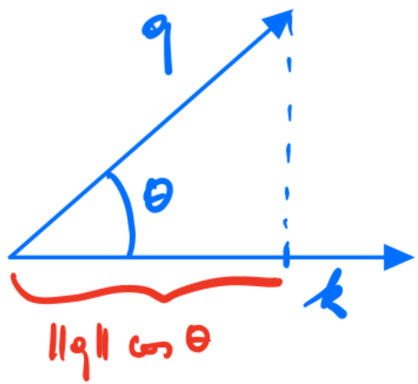
When queries and keys are vectors of the same length d, we can use a scaled dot-product attention as the scoring function.

Given $\mathbf{q} \in \mathbb{R}^d$ and $\mathbf{k} \in \mathbb{R}^d$, the scaled dot-product attention scoring function is

$$a(\mathbf{q}, \mathbf{k}) = \frac{\mathbf{q}^T \mathbf{k}}{\sqrt{d}}.$$

For $\boldsymbol{n}$ queries $\mathbf{Q} \in \mathbb{R}^{n \times d}$, keys $\mathbf{K} \in \mathbb{R}^{m \times d}$ and values $\mathbf{V} \in \mathbb{R}^{m \times v}$, the scaled dot-product attention layer computes an output tensor

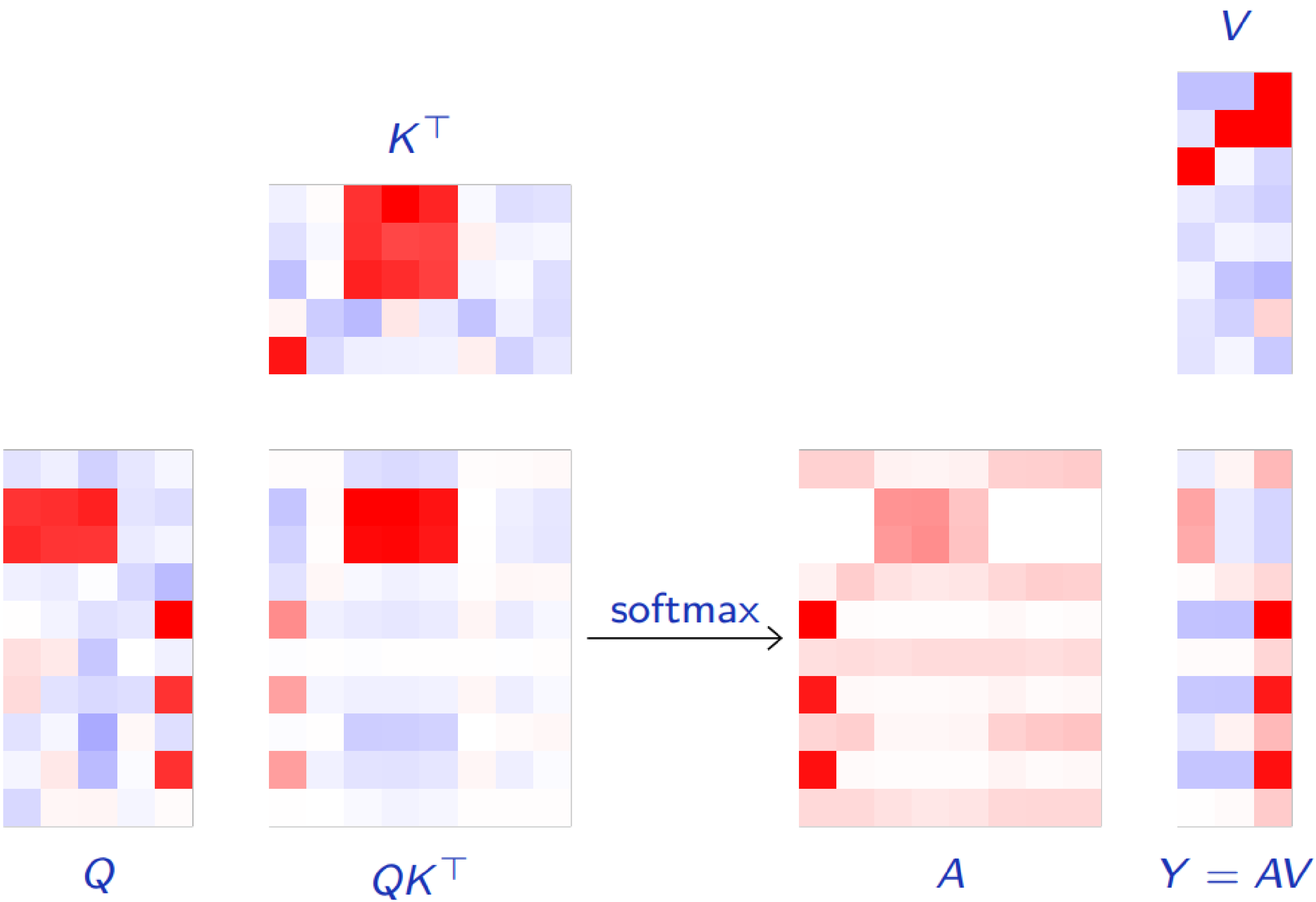
$$\mathbf{Y} = \underbrace{\text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right)}_{\text{attention matrix } \mathbf{A}} \mathbf{V} \in \mathbb{R}^{n \times v}.$$



$$q^T k = \|q\| \, \|k\| \cos\theta$$

Recall that the dot product is simply a un-normalised cosine similarity, which tells us about the alignment of two vectors.

Therefore, the $\mathbf{QK}^T$ matrix is a similarity matrix between queries and keys.



—

Credits: Francois Fleuret, Deep Learning, UNIGE/EPFL.

In the currently standard models for sequences, the queries, keys and values are linear functions of the inputs.

Given the learnable matrices $\mathbf{W}_q \in \mathbb{R}^{d \times x}$, $\mathbf{W}_k \in \mathbb{R}^{d \times x'}$, and $\mathbf{W}_v \in \mathbb{R}^{v \times x'}$, and two input sequences $\mathbf{X} \in \mathbb{R}^{n \times x}$ and $\mathbf{X}' \in \mathbb{R}^{m \times x'}$, we have

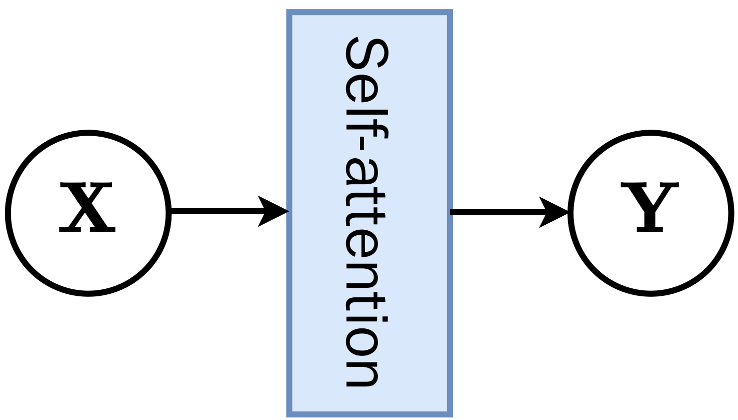
$$\mathbf{Q} = \mathbf{X}\mathbf{W}_q^T \in \mathbb{R}^{n \times d}$$
$$\mathbf{K} = \mathbf{X}'\mathbf{W}_k^T \in \mathbb{R}^{m \times d}$$
$$\mathbf{V} = \mathbf{X}'\mathbf{W}_v^T \in \mathbb{R}^{m \times v}.$$

Self-attention

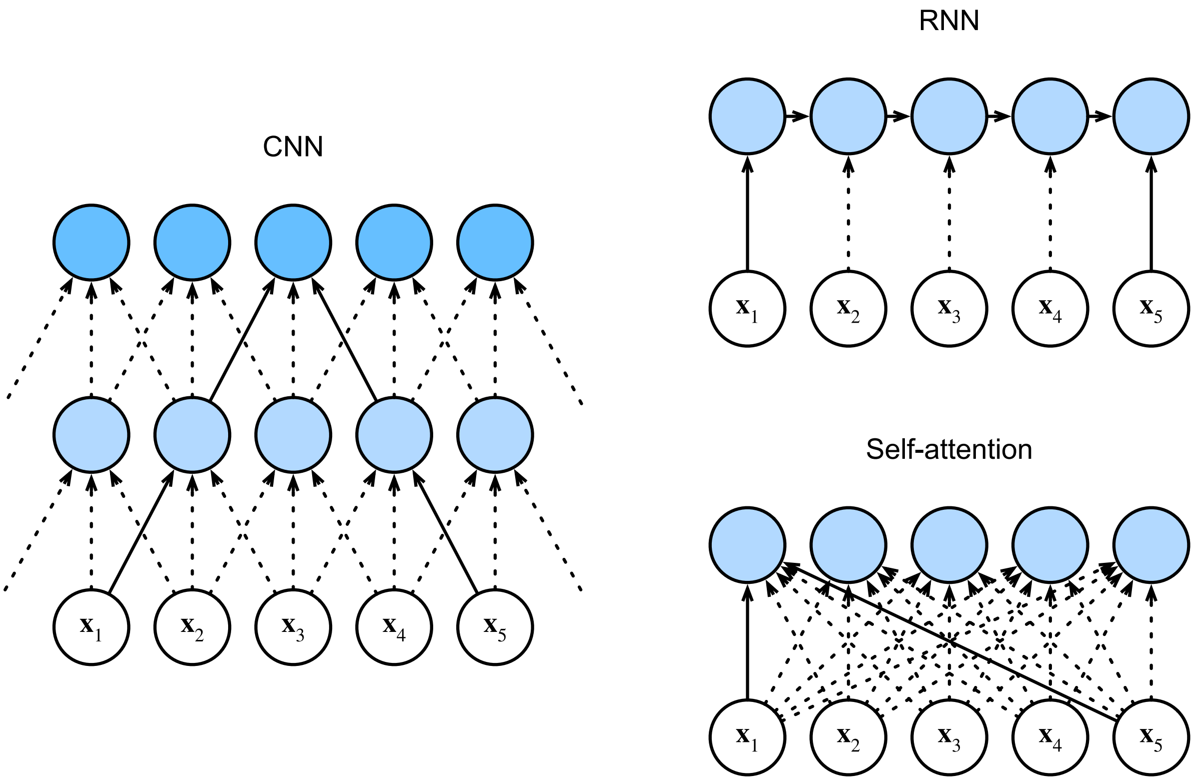
When the queries, keys and values are derived from the same inputs, the attention mechanism is called self-attention.

For the scaled dot-product attention, the self-attention layer is obtained when $\mathbf{X} = \mathbf{X}'$.

Therefore, self-attention can be used as a regular feedforward-kind of layer, similarly to fully-connected or convolutional layers.



CNNs vs. RNNs vs. self-attention

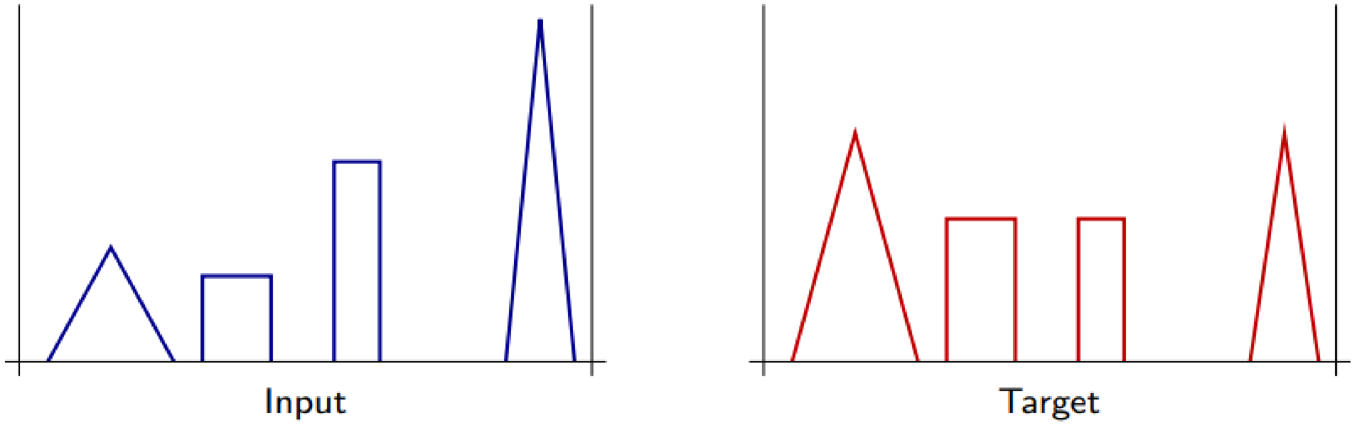


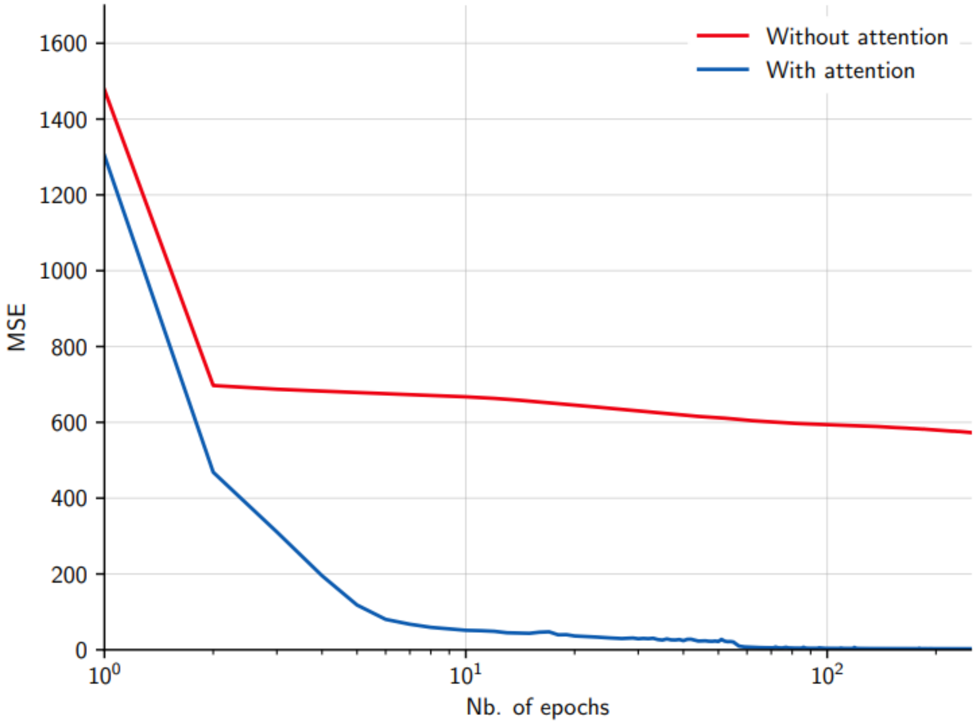
Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$

where n is the sequence length, d is the embedding dimension, and k is the kernel size of convolutions.

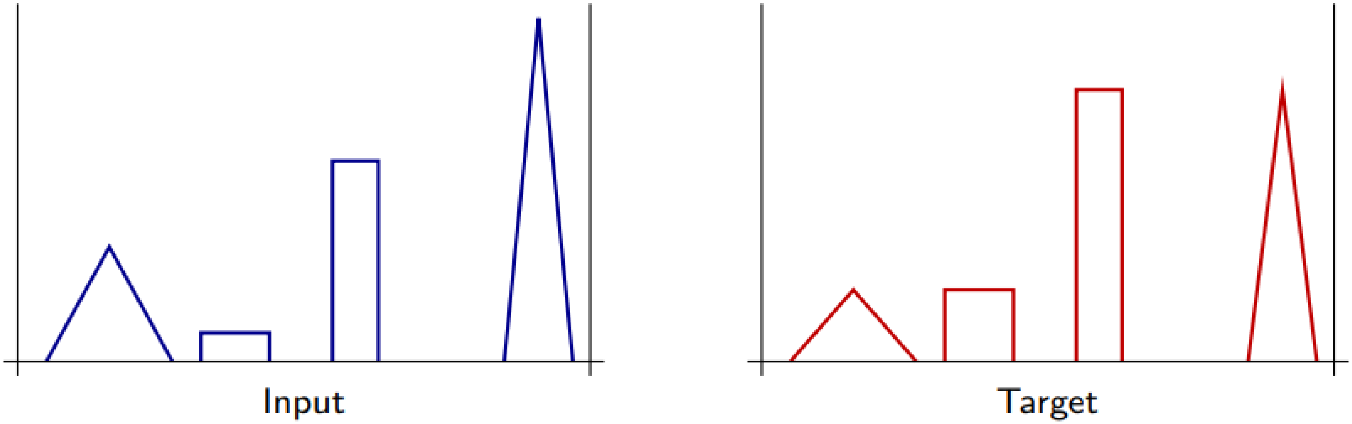
A toy example

To illustrate the behavior of the attention mechanism, we consider a toy problem with 1d sequences composed of two triangular and two rectangular patterns. The target sequence averages the heights in each pair of shapes.




Without attention
With attention
1600
1400
1200
1000
800
600
400
200
0
MSE
10^0
10^1
10^2
Nb. of epochs

We can modify the toy problem to consider targets where the pairs to average are the two right and leftmost shapes.

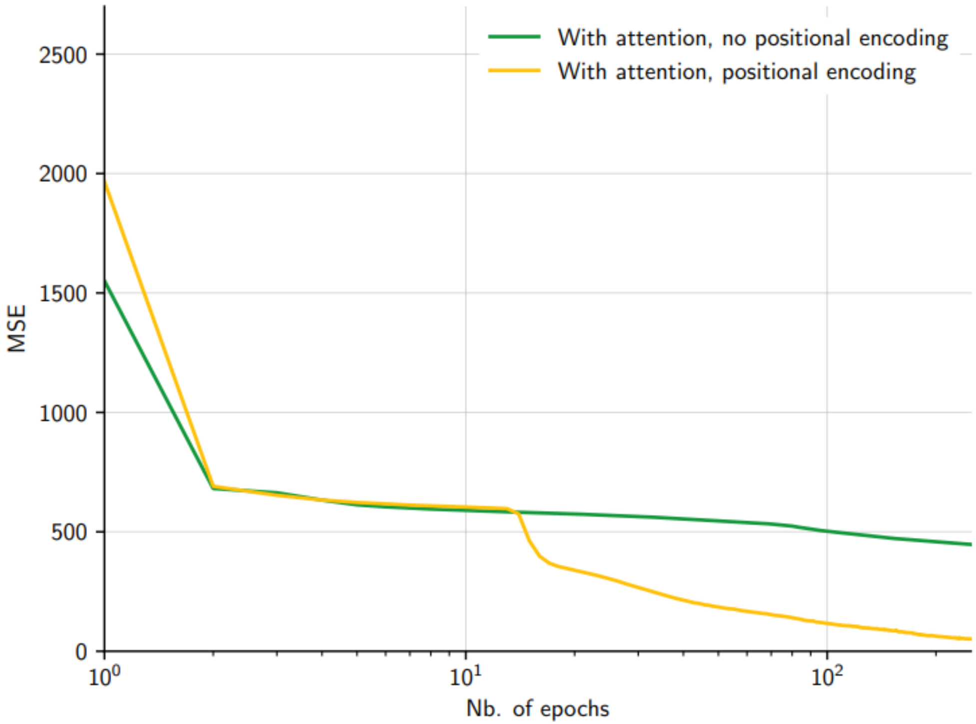


The performance is expected to be poor given the inability of the self-attention layer to take into account absolute or relative positions. Indeed, self-attention is permutation-invariant:

$$\begin{aligned}\mathbf{y} &= \sum_{i=1}^{m} \text{softmax}_i \left(\frac{\mathbf{q}^T \mathbf{K}_i^T}{\sqrt{d}} \right) \mathbf{V}_i \\ &= \sum_{i=1}^{m} \text{softmax}_i \left(\frac{\mathbf{q}^T \mathbf{K}_{\sigma(i)}^T}{\sqrt{d}} \right) \mathbf{V}_{\sigma(i)}\end{aligned}$$

for any permutation σ of the key-value pairs.

(It is also permutation-equivariant with permutation σ of the queries.)



However, this problem can be fixed by providing positional encodings explicitly to the attention layer.

Transformers

Vaswani et al. (2017) proposed to go one step further: instead of using attention mechanisms as a supplement to standard convolutional and recurrent layers, they designed a model, the **transformer**, combining only attention layers.

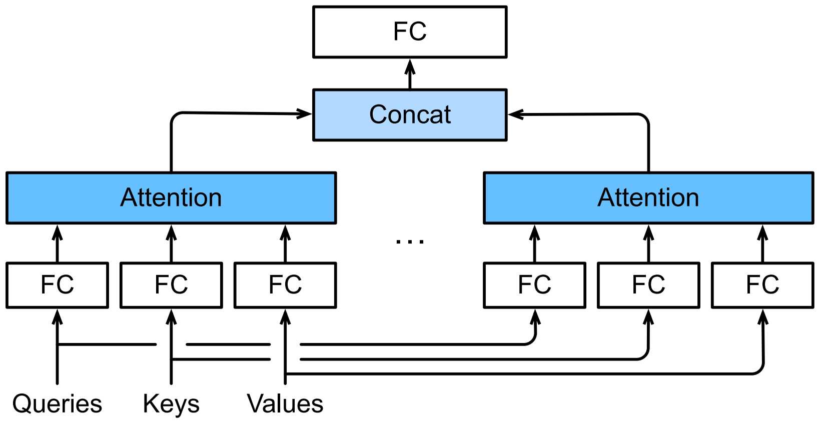
The transformer was designed for a sequence-to-sequence translation task, but it is currently key to state-of-the-art approaches for most tasks involving sets or sequences.

Scaled dot-product attention

The first building block of the transformer architecture is a scaled dot-production attention module

$$\text{attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$

where the $1/\sqrt{d_k}$ scaling is used to keep the (softmax's) temperature constant across different choices of the query/key dimension d_k.



Multi-head attention

The transformer projects the queries, keys and values $h = 8$ times with distinct linear projections to $d_k = 64$, $d_k = 64$ and $d_v = 64$ dimensions respectively.

$$\begin{aligned} \text{multihead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) &= \text{concat}\left(\mathbf{H}_1, \ldots, \mathbf{H}_h\right) \mathbf{W}^O \\ \mathbf{H}_i &= \text{attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V) \end{aligned}$$

with

$$\mathbf{W}_i^Q \in \mathbb{R}^{d_\text{model} \times d_k}, \mathbf{W}_i^K \in \mathbb{R}^{d_\text{model} \times d_k}, \mathbf{W}_i^V \in \mathbb{R}^{d_\text{model} \times d_v}, \mathbf{W}_i^O \in \mathbb{R}^{hd_v \times d_\text{model}}$$

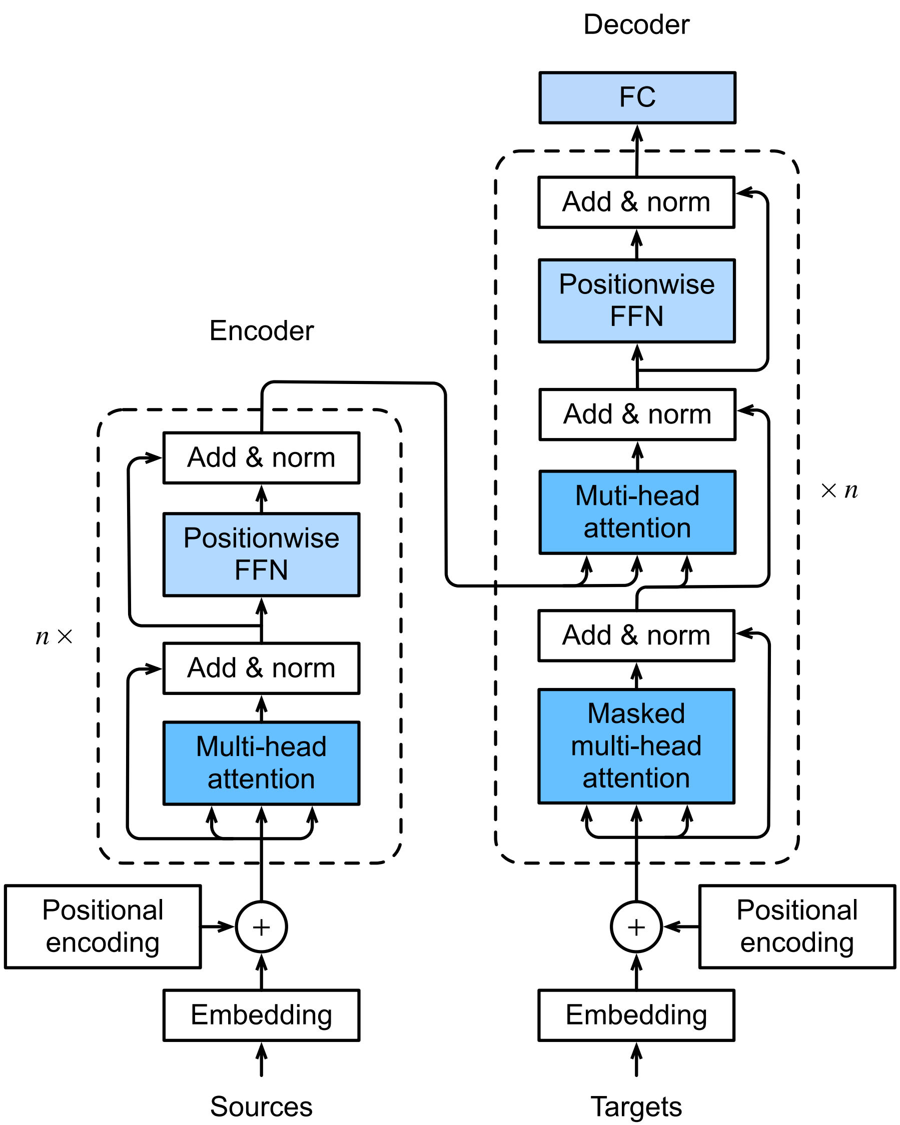
—

Credits: Dive Into Deep Learning, 2023.

Encoder-decoder architecture

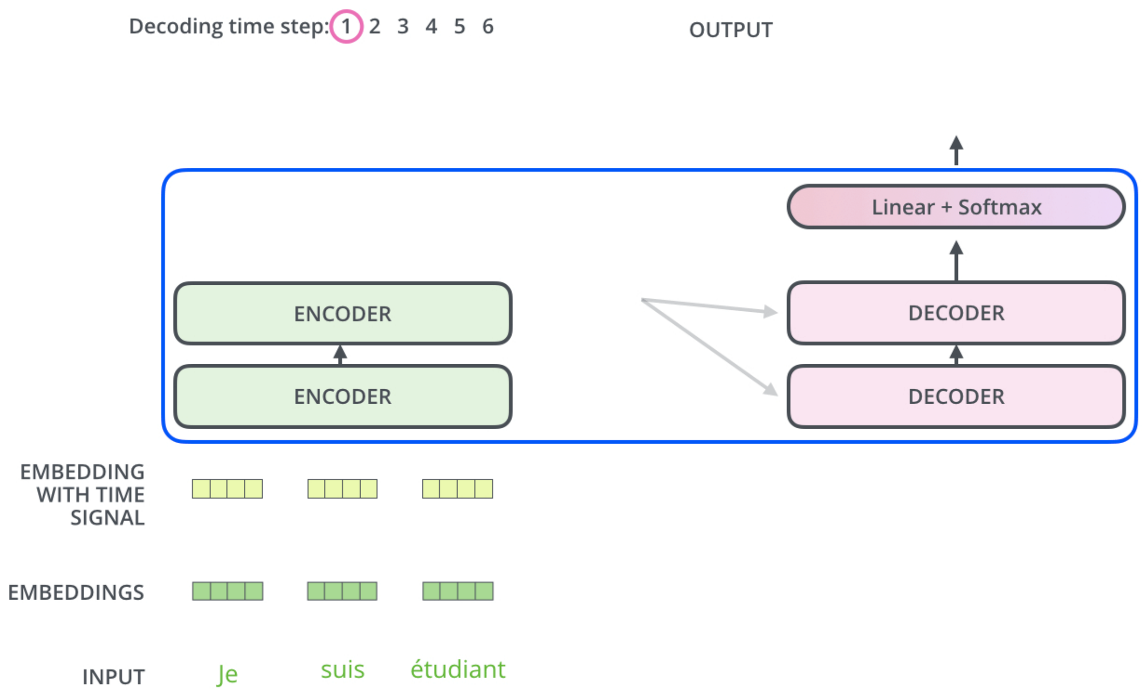
The transformer model is composed of:

- An encoder that combines $N = 6$ modules, each composed of a multi-head attention sub-module, and a (per-component) one-hidden-layer MLP, with residual pass-through and layer normalization. All sub-modules and embedding layers produce outputs of dimension $d_{\mathrm{model}} = 512$.
- A decoder that combines $N = 6$ modules similar to the encoder, but using masked self-attention to prevent positions from attending to subsequent positions. In addition, the decoder inserts a third sub-module which performs multi-head attention over the output of the encoder stack.



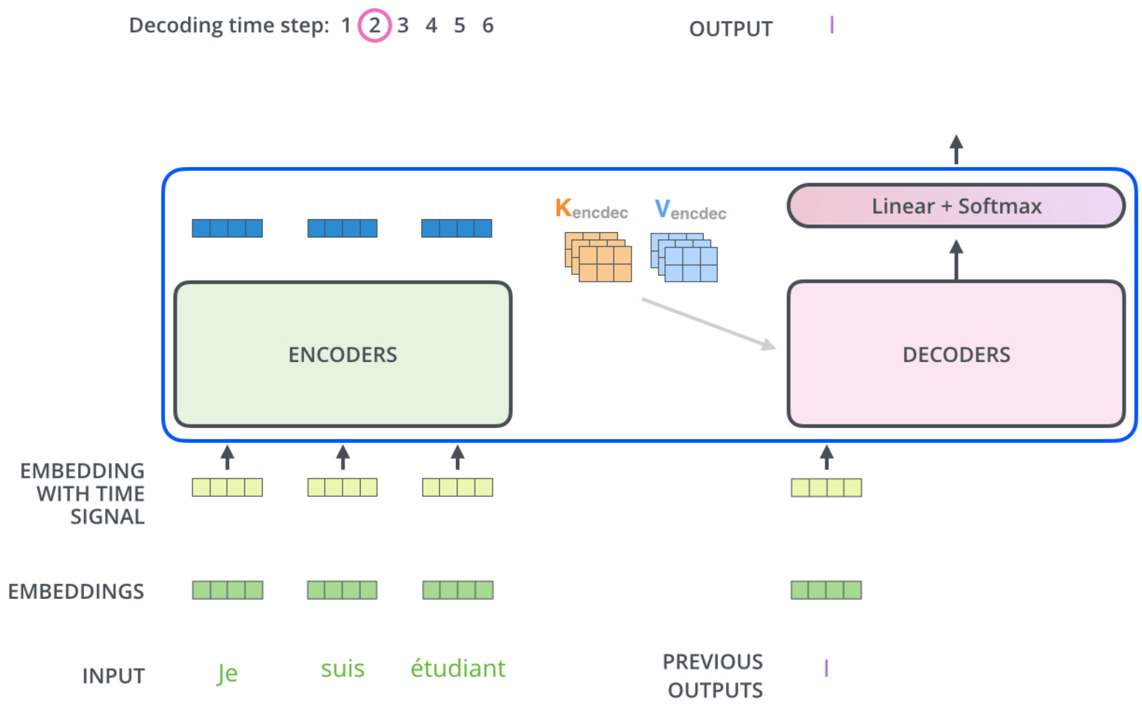
—

Credits: Dive Into Deep Learning, 2023.



The encoders start by processing the input sequence. The output of the top encoder is then transformed into a set of attention vectors $\mathbf{K}$ and $\mathbf{V}$ passed to the decoders.

—
Credits: Jay Alammar, The Illustrated Transformer.



Each step in the decoding phase produces an output token, until a special symbol is reached indicating the completion of the transformer decoder's output.

The output of each step is fed to the bottom decoder in the next time step, and the decoders bubble up their decoding results just like the encoders did.

—

Credits: Jay Alammar, The Illustrated Transformer.

In the decoder:

- The first masked self-attention sub-module is only allowed to attend to earlier positions in the output sequence. This is done by masking future positions.
- The second multi-head attention sub-module works just like multi-head self-attention, except it creates its query matrix from the layer below it, and takes the keys and values matrices from the output of the encoder stack.

—
Credits: Jay Alammar, The Illustrated Transformer.

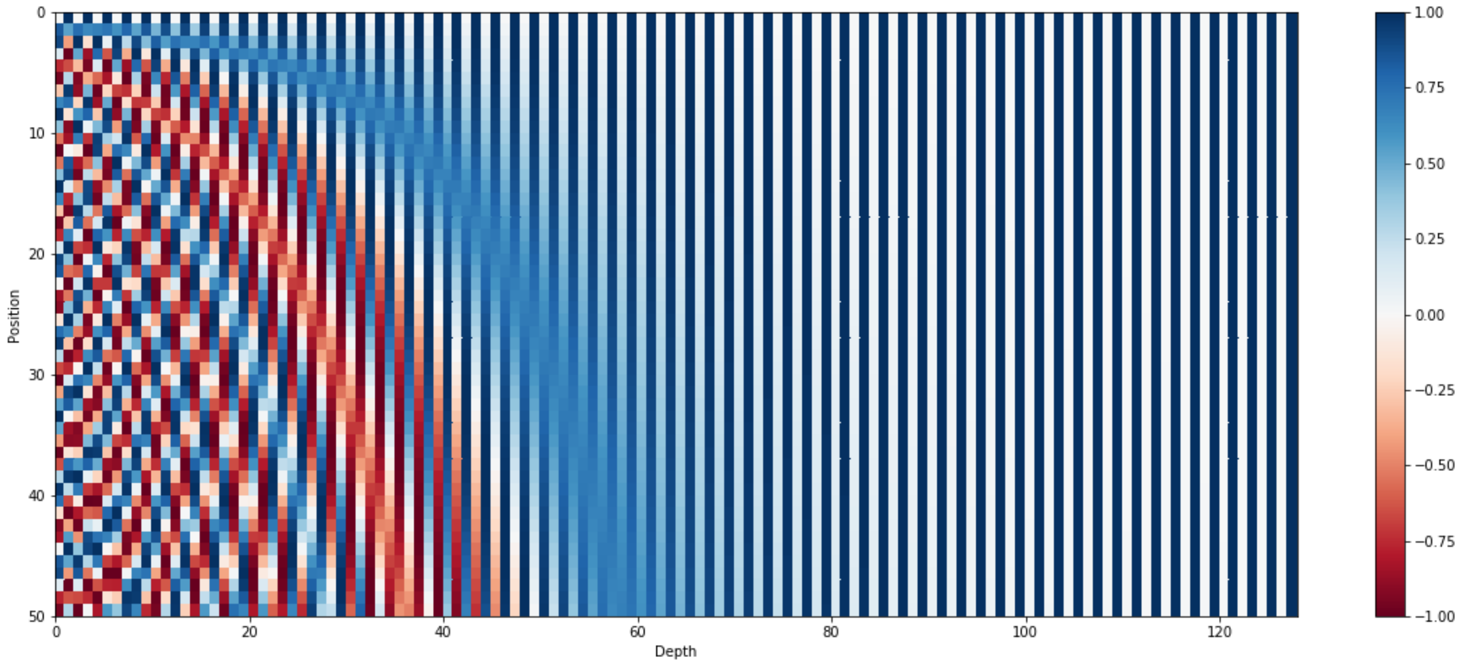
Positional encoding

Positional information is provided through an additive positional encoding of the same dimension d_{model} as the internal representation and is of the form

$$\begin{aligned} \text{PE}_{t,2i} &= \sin\left(\frac{t}{10000^{\frac{2i}{d_{\text{model}}}}}\right) \\ \text{PE}_{t,2i+1} &= \cos\left(\frac{t}{10000^{\frac{2i}{d_{\text{model}}}}}\right). \end{aligned}$$

After adding the positional encoding, words will be closer to each other based on the similarity of their meaning and their relative position in the sentence, in the d_{model}-dimensional space.

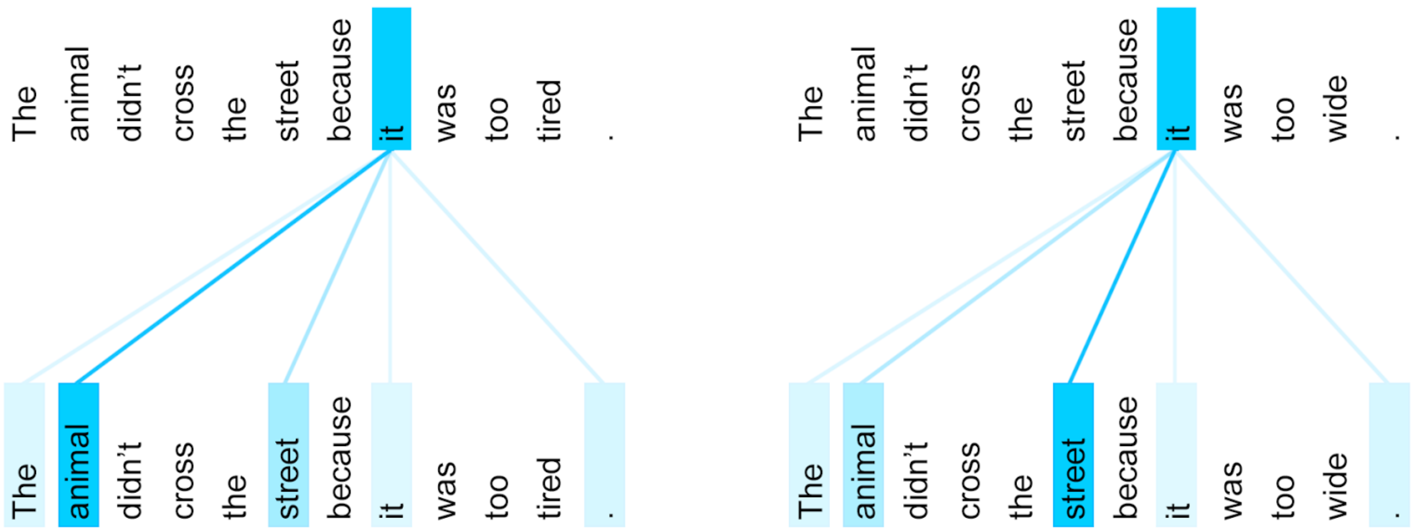
Alternatively, the model can also learn the positional encoding.



128-dimensional positonal encoding for a sentence with the maximum length of 50. Each row represents the embedding vector.

Machine translation

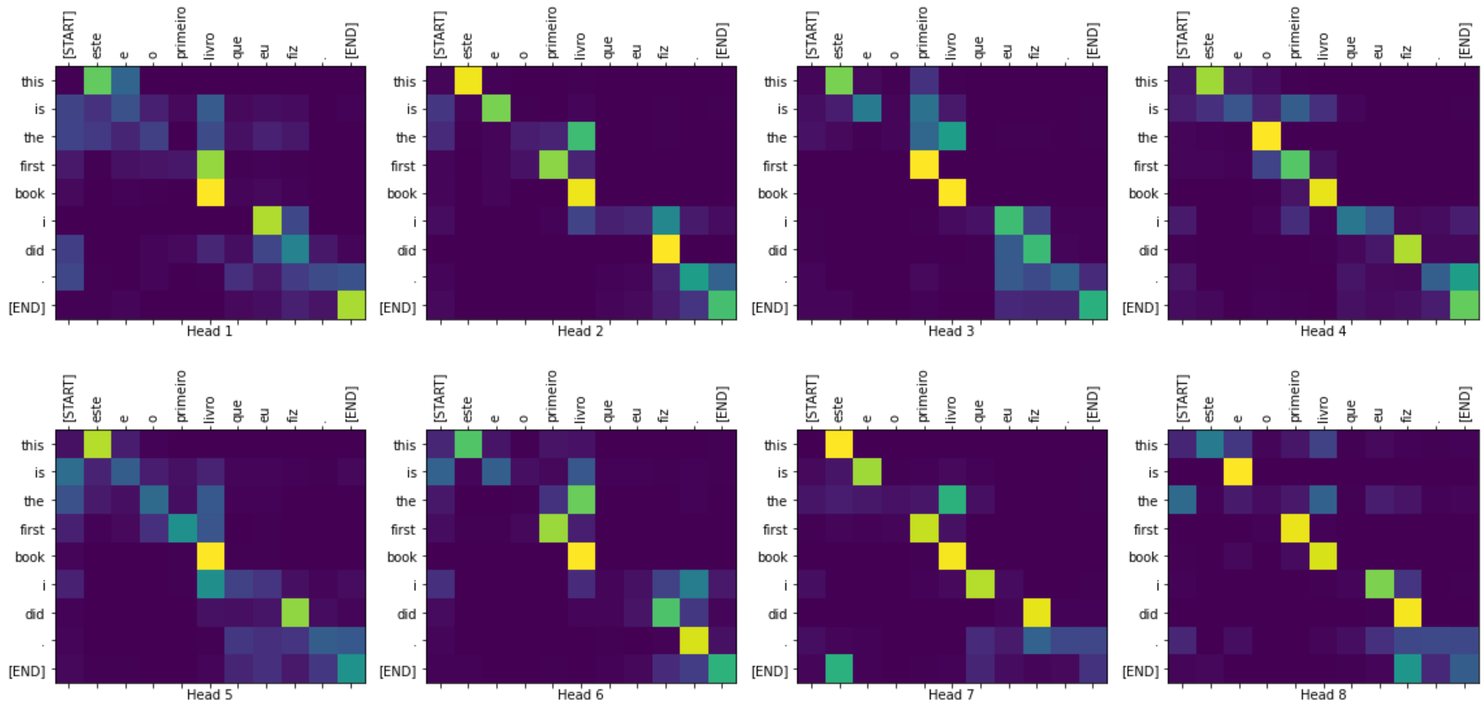
The transformer architecture was first designed for machine translation and tested on English-to-German and English-to-French translation tasks.



Self-attention layers learned that "it" could refer to different entities, in different contexts.

—

Credits: Transformer: A Novel Neural Network Architecture for Language Understanding, 2017.



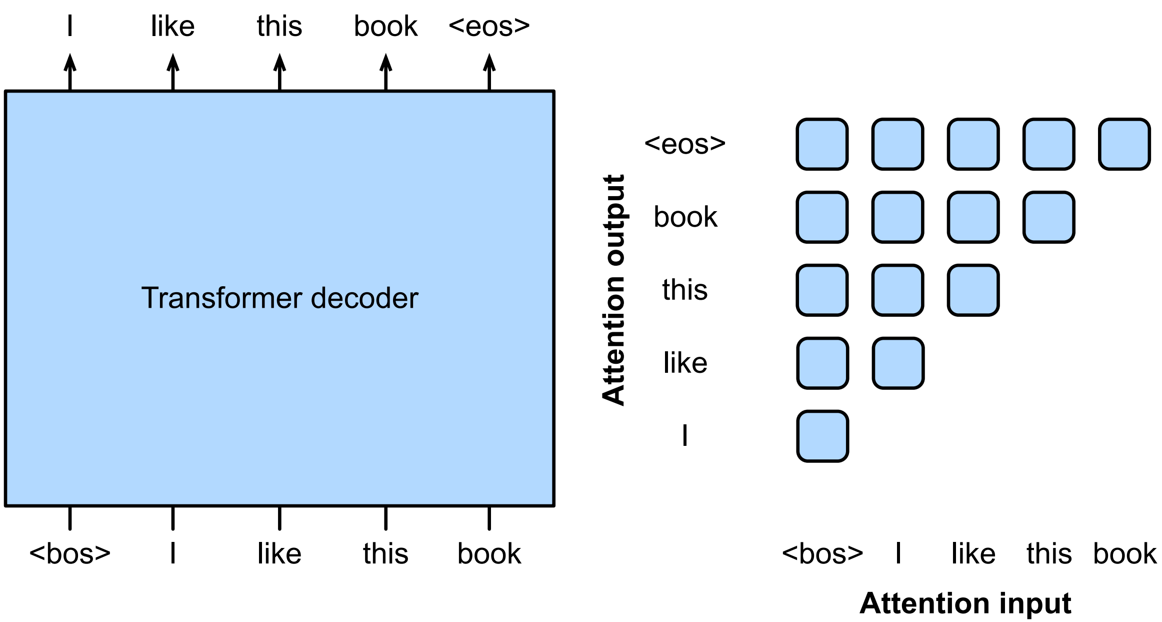
Attention maps extracted from the multi-head attention modules show how input tokens relate to output tokens.

—
Credits: Transformer model for language understanding.

Decoder-only transformers

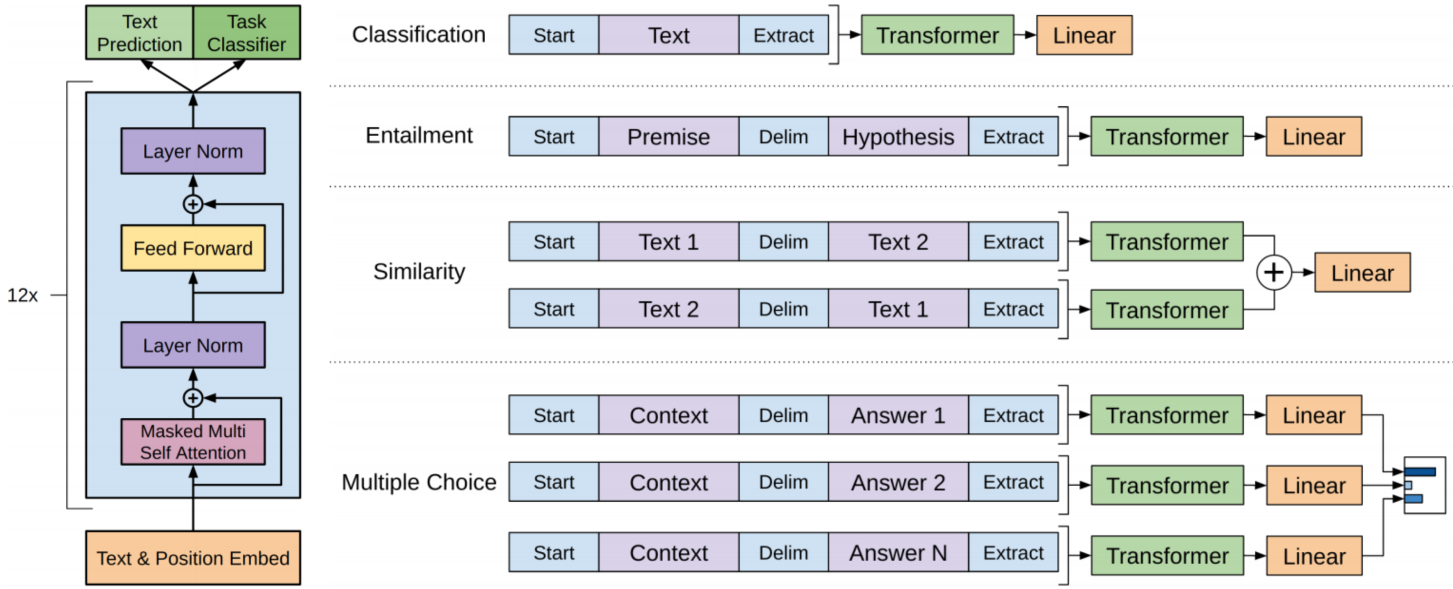
The decoder-only transformer has become the de facto architecture for large language models $p(\mathbf{x}_t | \mathbf{x}_{1:t-1})$.

These models are trained with self-supervised learning, where the target sequence is the same as the input sequence, but shifted by one token to the right.



(demo)

Historically, GPT-1 was first pre-trained and then fine-tuned on downstream tasks.



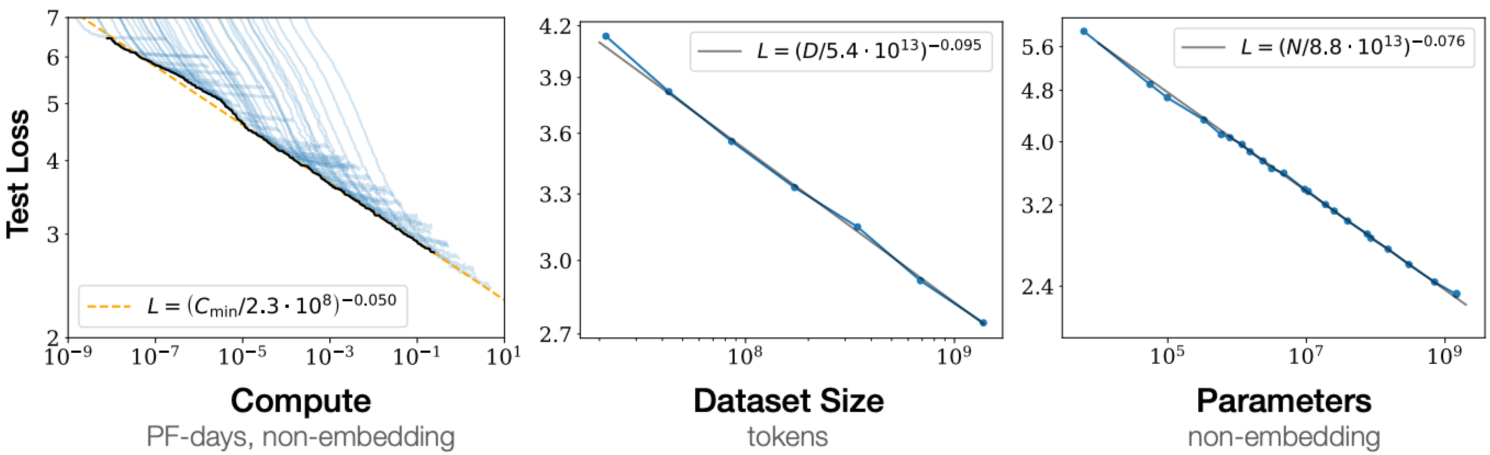
—

Credits: Radford et al., Improving Language Understanding by Generative Pre-Training, 2018.

Scaling laws

Transformer language model performance improves smoothly as we increase the model size, the dataset size, and amount of compute used for training.

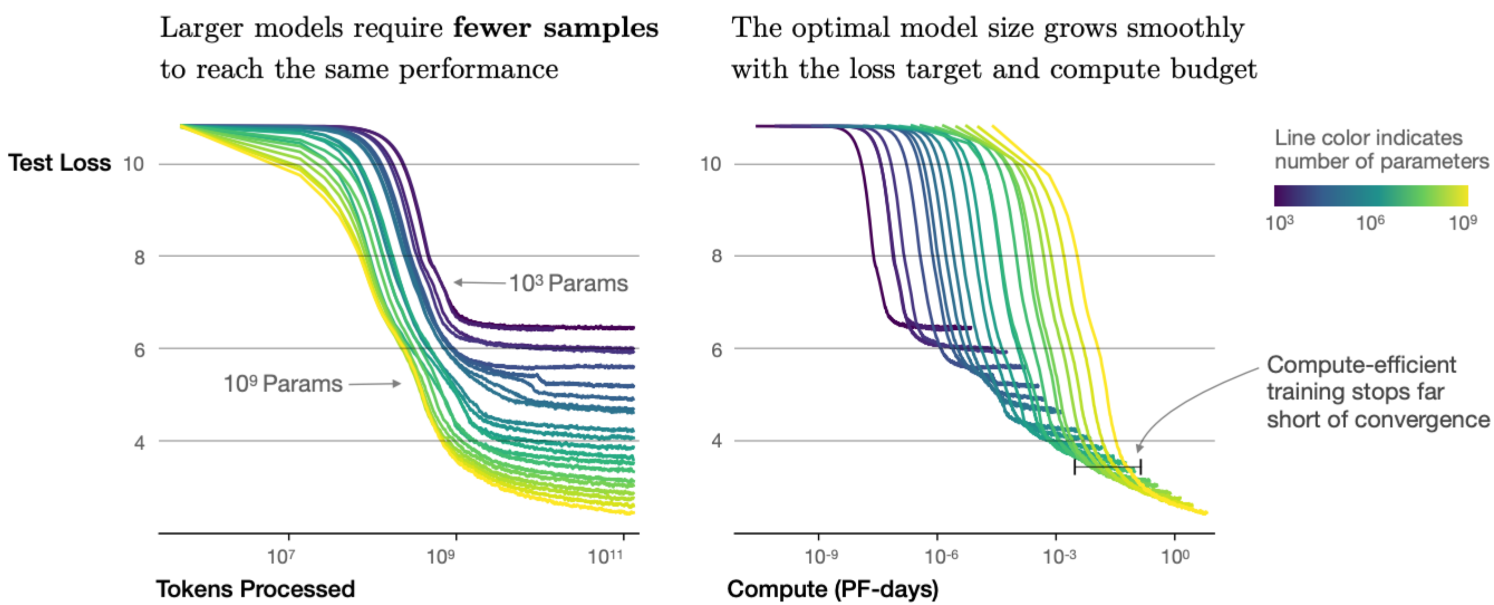
For optimal performance, all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.



—
Credits: Kaplan et al, 2020.

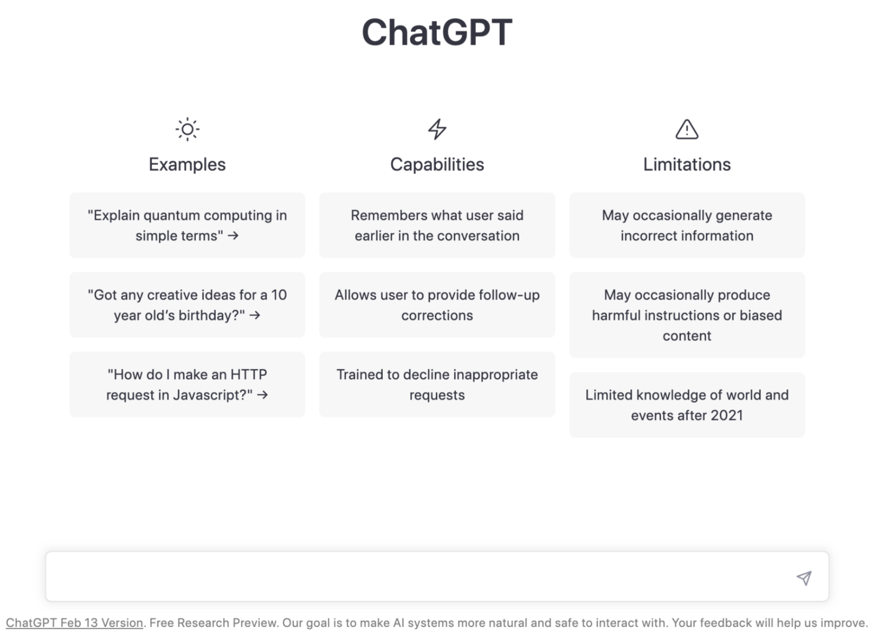
Large models also enjoy better sample efficiency than small models.

- Larger models require less data to achieve the same performance.
- The optimal model size shows to grow smoothly with the amount of compute available for training.



—
Credits: Kaplan et al, 2020.

Conversational agents

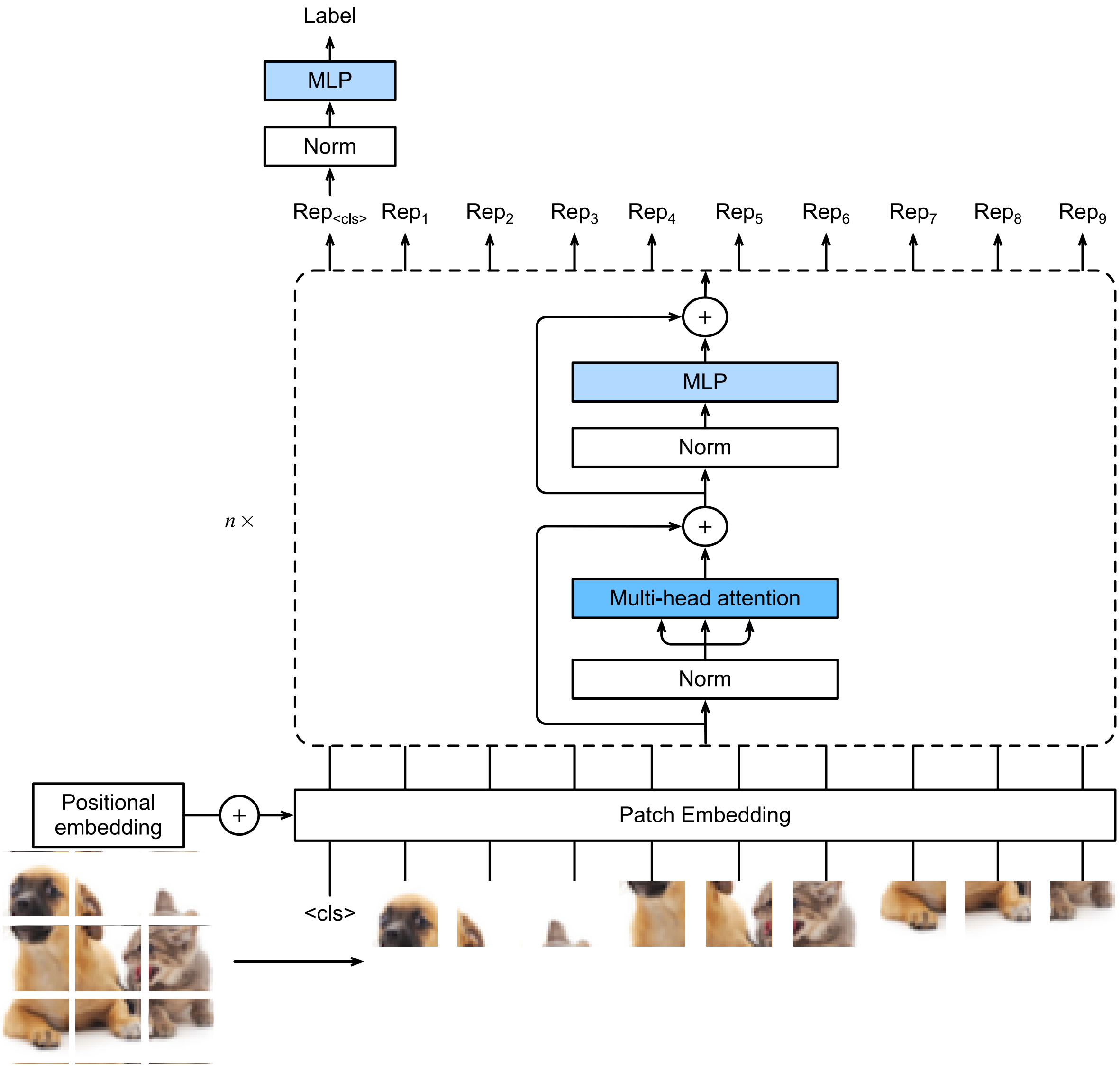


All modern conversational agents are based on the same transformer models, scaled up to billions of parameters, trillions of training tokens, and thousands of petaflop/s-days of compute.

Transformers for images

The transformer architecture was first designed for sequences, but it can be adapted to process images.

The key idea is to reshape the input image into a sequence of patches, which are then processed by a transformer encoder. This architecture is known as the **vision transformer** (ViT).



—
Credits: Dive Into Deep Learning, 2023.

- The input image is divided into non-overlapping patches, which are then linearly embedded into a sequence of vectors.
- The sequence of vectors is then processed by a transformer encoder, which outputs a sequence of vectors.
- Training the vision transformer can be done with supervised or self-supervised learning.

The end.