

# *Deep Learning*

*A course about theory & practice*



## Attention and Transformers

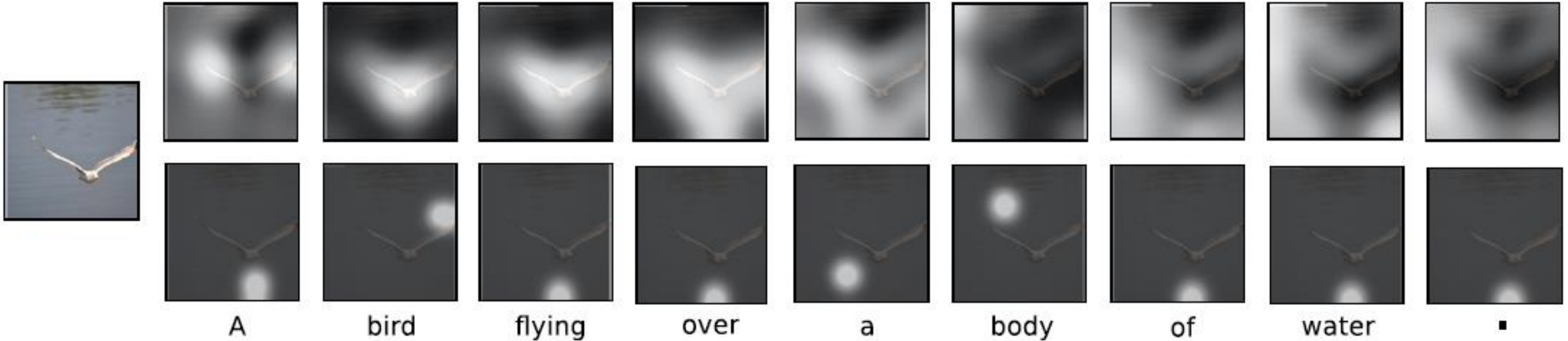
Marco Piastra

*Attention is what we need?  
(intuition)*

# Generating Text Captions from Images

## ■ DCNN + RNN

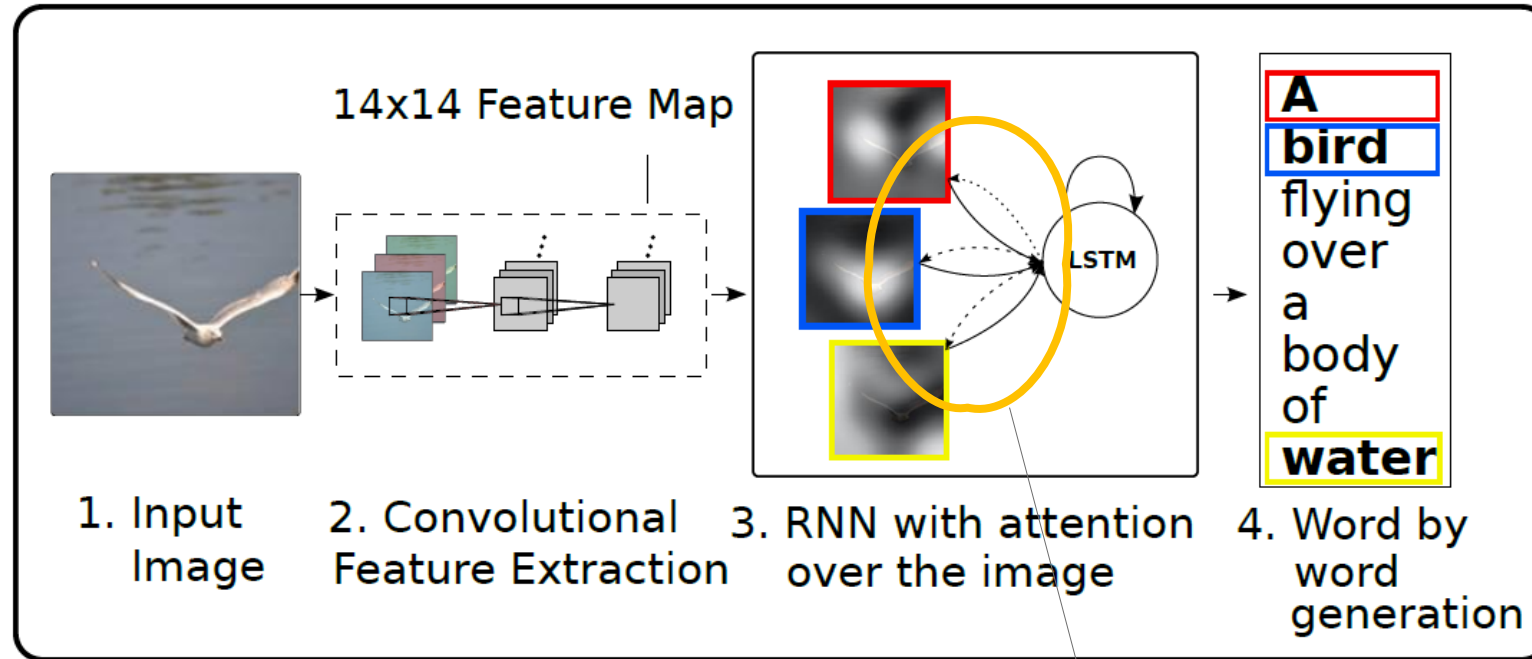
[*Show, Attend and Tell: Neural Image Caption Generation with Visual Attention*, Xu et al., 2015]



# Generating Text Captions from Images

## ■ DCNN + RNN

[*Show, Attend and Tell: Neural Image Caption Generation with Visual Attention*, Xu et al., 2015]

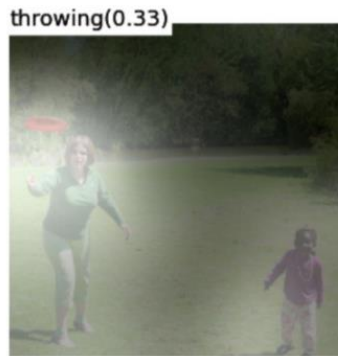


*The 'trick' is here:  
when generating each word  
the LSTM focuses on a specific  
region in the image*

# Generating Text Captions from Images

## ■ DCNN + RNN

[*Show, Attend and Tell: Neural Image Caption Generation with Visual Attention*, Xu et al., 2015]



# Generating Text Captions from Images

## ■ DCNN + RNN

[*Show, Attend and Tell: Neural Image Caption Generation with Visual Attention*, Xu et al., 2015]



A woman is throwing a frisbee in a park.



A dog is standing on a hardwood floor.



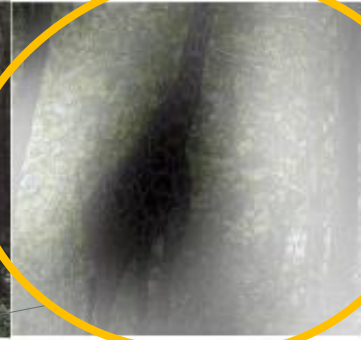
A stop sign is on a road with a mountain in the background.



A little girl sitting on a bed with a teddy bear.



A group of people sitting on a boat in the water.



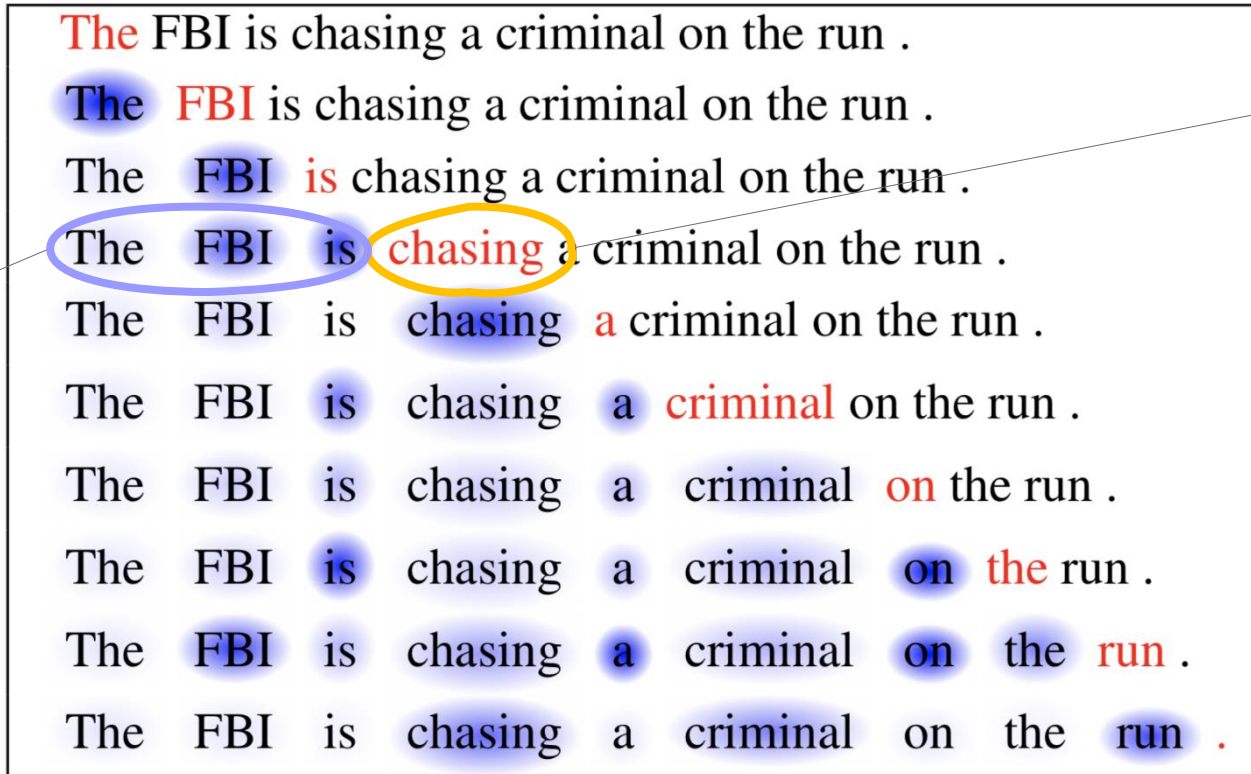
A giraffe standing in a forest with trees in the background.

*Look at this: attention focuses on regions that are far apart in the image*

# Natural Language Requires Attention

## ■ Encoder / Decoder with attention

[Long Short-Term Memory-Networks for Machine Reading, Cheng, Dong and Lapata, 2016]



Current word being read

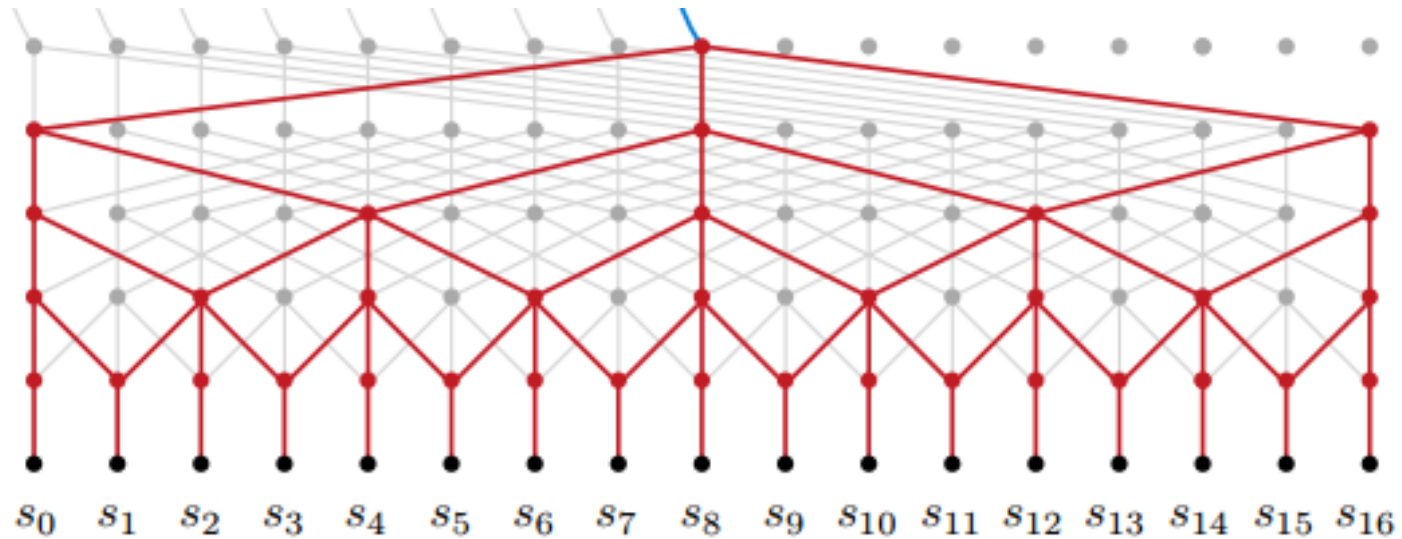
Relevance weights of previous words

The machine learns a hidden representation, for *sentiment analysis*, by focusing on different previous words while reading a sentence

# Attention and Convolution

- **Progressively widening receptive field**

Consider 1D convolution, size 3:  
the receptive field of each filter grows progressively



*Problem: four layers are required in this case to have a receptive field of 16*

*Attention as a Kernel  
(another perspective over convolution)*



# Attention as Kernel

## ■ Attention Pooling

Consider an input-output relation

$$(\mathbf{x}, \mathbf{y}), \quad \mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^n \quad D := \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^N$$

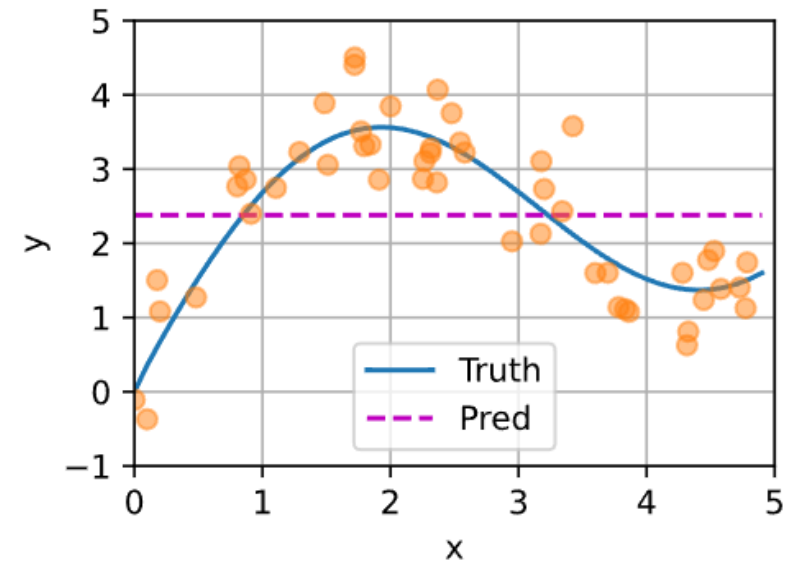
Attention Pooling is defined as a function on each input component

$$\tilde{\mathbf{y}} := \sum_{i=1}^N \alpha(\mathbf{x}, \mathbf{x}^{(i)}) \mathbf{y}^{(i)},$$

Example:

Global average as constant kernel (i.e., no attention)

$$\alpha(x, x_i) = \frac{1}{N}$$



[image from [http://d2l.ai/chapter\\_attention-mechanisms/](http://d2l.ai/chapter_attention-mechanisms/)]

# Attention as Kernel

## ■ Attention Pooling

Consider an input-output relation

$$(\mathbf{x}, \mathbf{y}), \quad \mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^n \quad D := \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^N$$

Attention Pooling is defined as a function on each input component

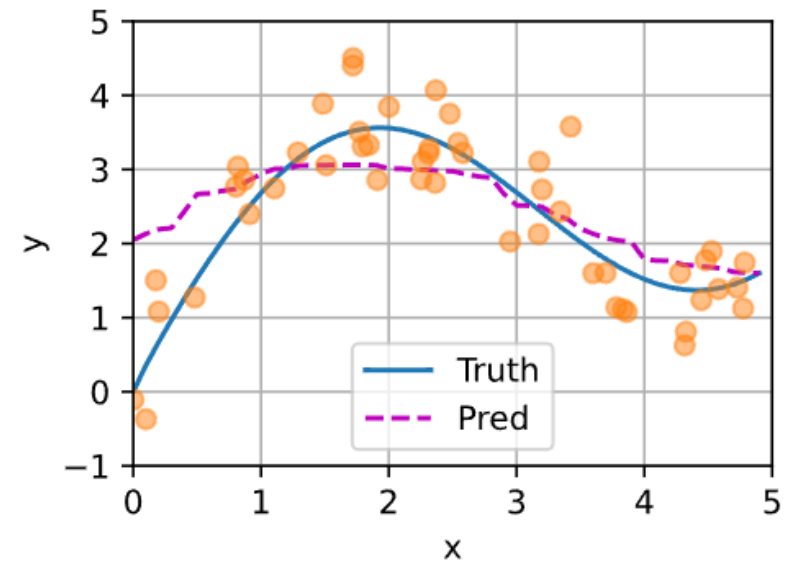
$$\tilde{\mathbf{y}} := \sum_{i=1}^N \alpha(\mathbf{x}, \mathbf{x}^{(i)}) \mathbf{y}^{(i)},$$

Example:

Gaussian Kernel [Nadaraya & Watson, 1964]

$$\alpha(x, x^{(i)}) = \frac{K(x - x^{(i)})}{\sum_{j=1}^N K(x - x^{(j)})}$$

$$K(u) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{u^2}{2}\right)$$



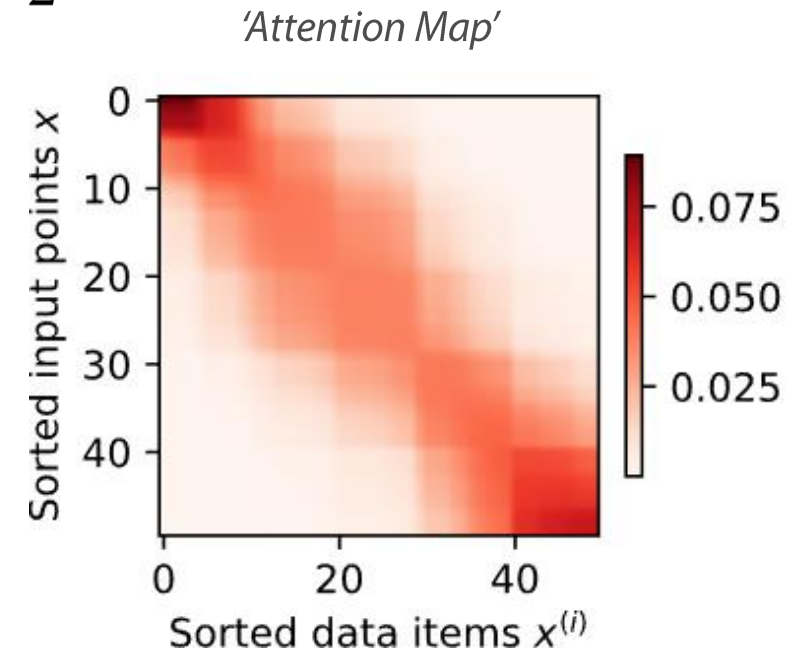
[image from [http://d2l.ai/chapter\\_attention-mechanisms/](http://d2l.ai/chapter_attention-mechanisms/)]

# Attention as Kernel

## ■ Gaussian Kernel and Softmax

$$\alpha(x, x^{(i)}) = \frac{K(x - x^{(i)})}{\sum_{j=1}^N K(x - x^{(j)})} \quad K(u) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{u^2}{2}\right)$$

$$\begin{aligned} \alpha(x, x^{(i)}) &= \frac{\exp\left(-\frac{1}{2}(x - x^{(i)})^2\right)}{\sum_{j=1}^N \exp\left(-\frac{1}{2}(x - x^{(j)})^2\right)} \\ &= \text{softmax}\left(-\frac{1}{2}(x - x^{(i)})^2\right) \end{aligned}$$



Gaussian kernel regression converges to the optimal solution, as the dataset increases

Note that Gaussian Kernel is non-parametric: it is a pure pooling operation

[image from [http://d2l.ai/chapter\\_attention-mechanisms/](http://d2l.ai/chapter_attention-mechanisms/)]

# Attention as Kernel

## ▪ (Simple) Parametric Attention Pooling

$$\begin{aligned}\alpha(x, x_i) &= \frac{\exp\left(-\frac{1}{2}(x - x_i)^2 w\right)}{\sum_{j=1}^N \exp\left(-\frac{1}{2}(x - x_j)^2 w\right)} \\ &= \text{softmax}\left(-\frac{1}{2}(x - x_i)^2 w\right)\end{aligned}$$

$$\begin{aligned}K(u) &= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{u^2 w}{2}\right) \\ &\Rightarrow \sigma^2 = \frac{1}{w}\end{aligned}$$

This requires training of the (unique) parameter  $w$

Consider an MSE loss function:

$$L(D) = \frac{1}{N} \sum_{i=1}^N (f(x^{(i)}) - y^{(i)})^2$$

and perform *gradient descent*

# Attention as Kernel

## ▪ (Simple) Parametric Attention Pooling

$$\begin{aligned}\alpha(x, x_i) &= \frac{\exp\left(-\frac{1}{2}(x - x_i)^2 w\right)}{\sum_{j=1}^N \exp\left(-\frac{1}{2}(x - x_j)^2 w\right)} \\ &= \text{softmax}\left(-\frac{1}{2}(x - x_i)^2 w\right)\end{aligned}$$

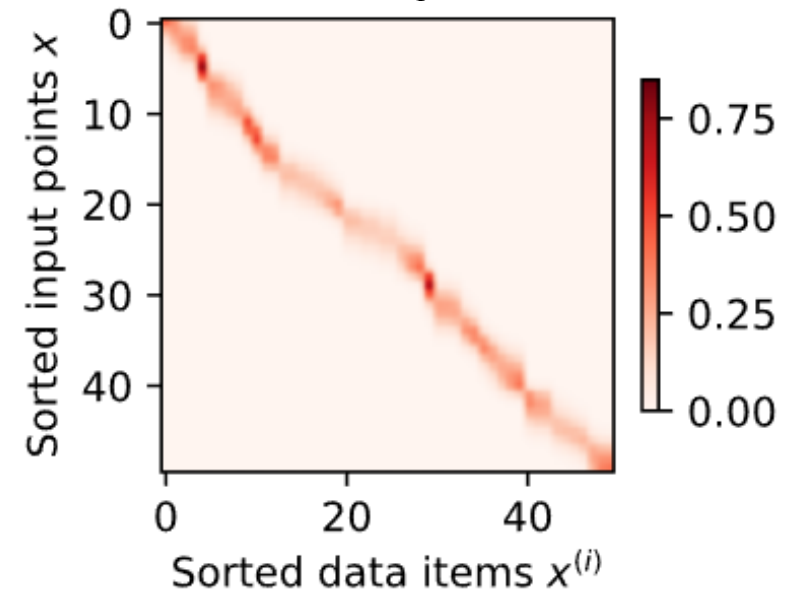
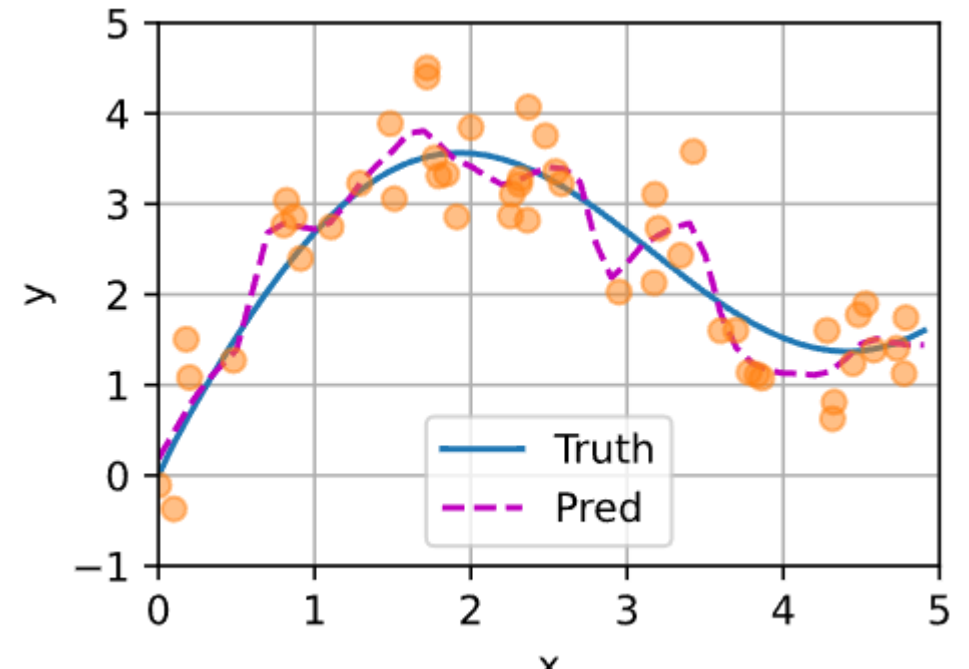
This requires training of the (unique) parameter  $w$

Consider an MSE loss function:

$$L(D) = \frac{1}{N} \sum_{i=1}^N (f(x^{(i)}) - y^{(i)})^2$$

and perform *gradient descent*

*The (Gaussian) attention field becomes 'sharper'*



[image from [http://d2l.ai/chapter\\_attention-mechanisms/](http://d2l.ai/chapter_attention-mechanisms/)]

# Attention as Kernel

## ■ Terminology

In the following:

- data items will be referred to as *keys*
- input items will be referred to as *queries*

(This is field-specific jargon)

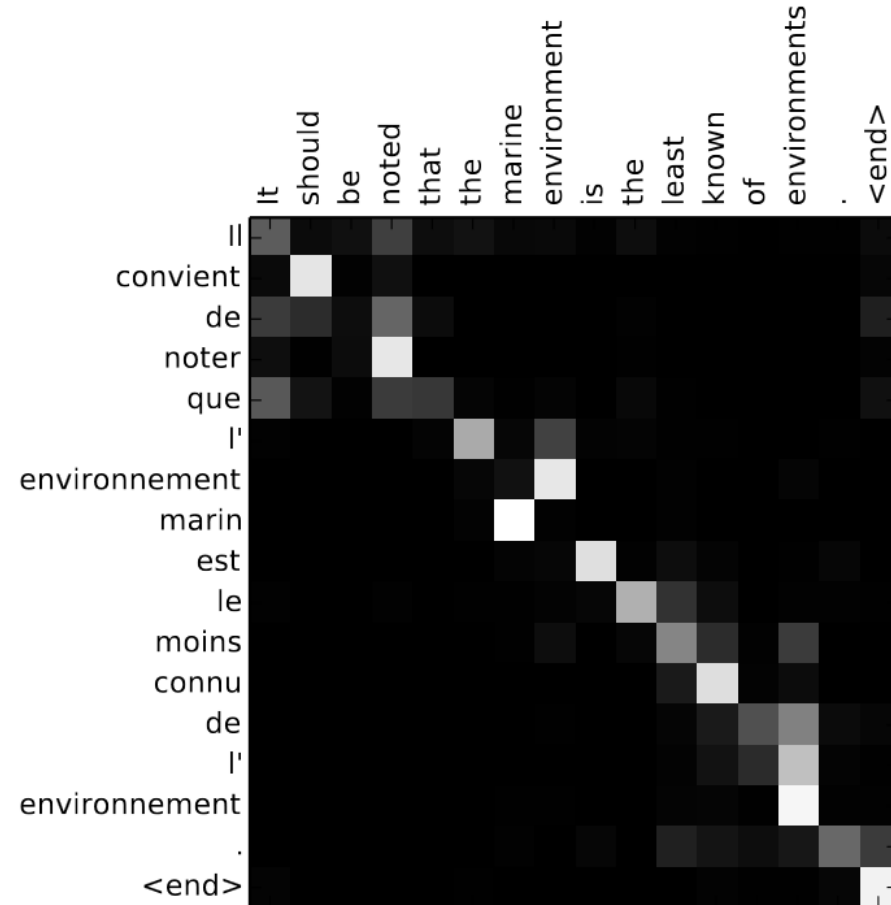
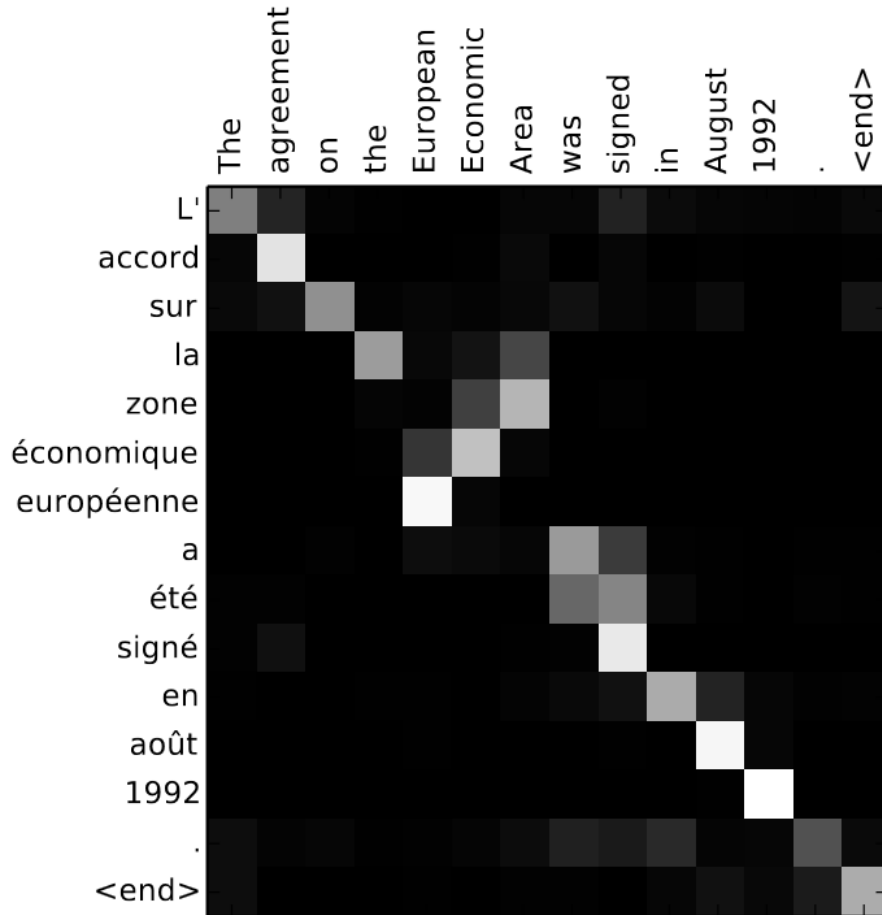


[image from [http://d2l.ai/chapter\\_attention-mechanisms/](http://d2l.ai/chapter_attention-mechanisms/)]

# *Attention: Queries, Key and Values*

# Is Attention a Kernel?

The phenomenon of inversion in language translation



[image from <https://arxiv.org/pdf/1409.0473>]

# Attention Pooling: Queries, Keys and Values

## ▪ Generalized model

$$\text{Attention}(\mathbf{q}, \mathbf{k}, \mathbf{v}) := \sum_{i=1}^m \alpha(\mathbf{q}, \mathbf{k}_i) \mathbf{v}_i$$

$$\mathbf{q} \in \mathbb{R}^q, \mathbf{k} \in \mathbb{R}^k, \mathbf{v} \in \mathbb{R}^v$$

In general:

- *queries* and *keys* could come from different spaces
- the *attention map (kernel)*  $\alpha$  is normalized: it describes how attention is *distributed*
- *values* are specific contributions (in general, sizes are equal  $k = v$ )
- $m$  is the width of the receptive field (=how many keys are in it)

# Attention: Queries, Keys and Values

- **Generalized model**

$$\text{Attention}(\mathbf{q}, \mathbf{k}, \mathbf{v}) := \sum_{i=1}^m \alpha(\mathbf{q}, \mathbf{k}_i) \mathbf{v}_i$$

$$\mathbf{q} \in \mathbb{R}^q, \mathbf{k} \in \mathbb{R}^k, \mathbf{v} \in \mathbb{R}^v$$

The *attention map* is defined as:

$$\alpha(\mathbf{q}, \mathbf{k}_i) = \text{softmax}(a(\mathbf{q}, \mathbf{k}_i)) = \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_{j=1}^m \exp(a(\mathbf{q}, \mathbf{k}_j))}$$

where  $a$  is the attention scoring function of choice

# Attention Scoring Function

## ▪ Scaled Dot-Product Attention

Assume that both queries and keys encoded as vectors of size  $d$

The *attention scoring function* is defined as:

$$a(\mathbf{q}, \mathbf{k}) := \frac{\mathbf{q} \cdot \mathbf{k}}{\sqrt{d}}, \quad \mathbf{q}, \mathbf{k} \in \mathbb{R}^d$$

In the line of principle,  $\mathbf{q}$  and  $\mathbf{k}$  could be anything, including the output of other *layers*

*The normalizing term  $\sqrt{d}$  comes from the assumptions that each component of the encodings is an independent random variable with zero mean and unit standard deviation*

# Attention Scoring Function

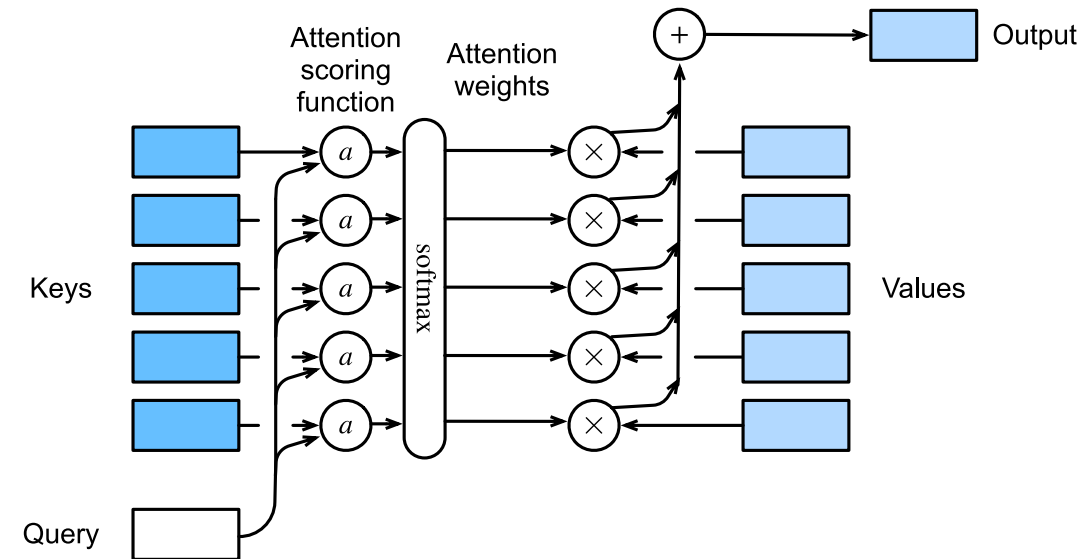
## ▪ Scaled Dot-Product Attention

Using a tensorial representation, assume there are  $m$  keys,  $n$  queries and  $v$  values:

$$\mathbf{Q} \in \mathbb{R}^{n \times d}, \mathbf{K} \in \mathbb{R}^{m \times d}, \mathbf{V} \in \mathbb{R}^{m \times v}$$

Attention Pooling becomes:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) := \text{softmax} \left( \frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}} \right) \mathbf{V} \in \mathbb{R}^{n \times v}$$



# Attention Map

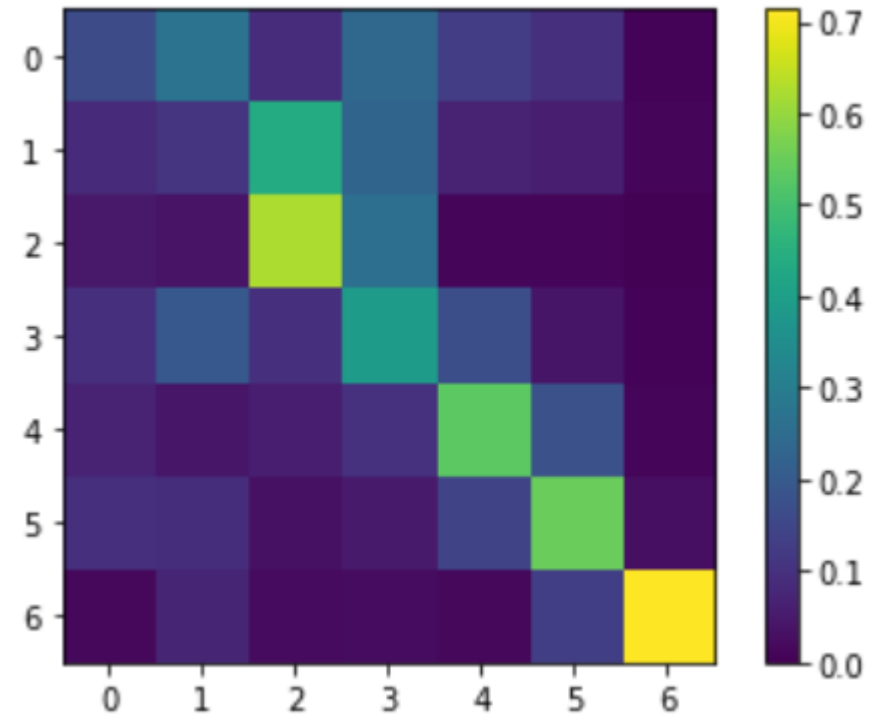
- **Example: a square matrix**

Assume that both *queries* and *keys* come from the same space:

$$\mathbf{Q} \in \mathbb{R}^{n \times d}, \mathbf{K} \in \mathbb{R}^{n \times d}$$

Then:

$$\alpha(\mathbf{Q}, \mathbf{K}) := \text{softmax} \left( \frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}} \right) \in \mathbb{R}^{n \times n}$$



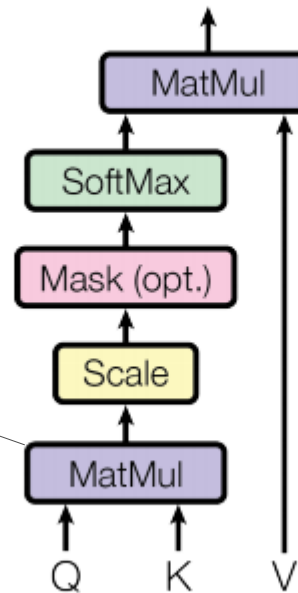
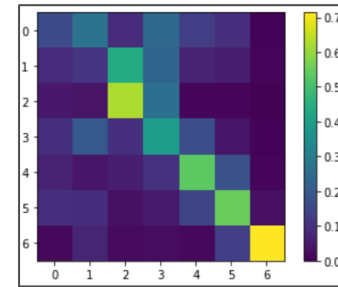
# Scaled Dot-Product Attention

$$\mathbf{Q} \in \mathbb{R}^{n \times d}, \mathbf{K} \in \mathbb{R}^{n \times d}, \mathbf{V} \in \mathbb{R}^{n \times v}$$

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) := \text{softmax} \left( \frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}} \right) \mathbf{V} \in \mathbb{R}^{n \times v}$$

Scaled Dot-Product Attention

*This is the basic building block*



# Scaled Dot-Product *Self*-Attention

$$\mathbf{Q} \in \mathbb{R}^{n \times d}, \mathbf{K} \in \mathbb{R}^{n \times d}, \mathbf{V} \in \mathbb{R}^{n \times n}$$

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) := \text{softmax} \left( \frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}} \right) \mathbf{V} \in \mathbb{R}^{n \times n}$$

In *self-attention*, flexibility can be gained by adding a *linear transformation*:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_q^T$$

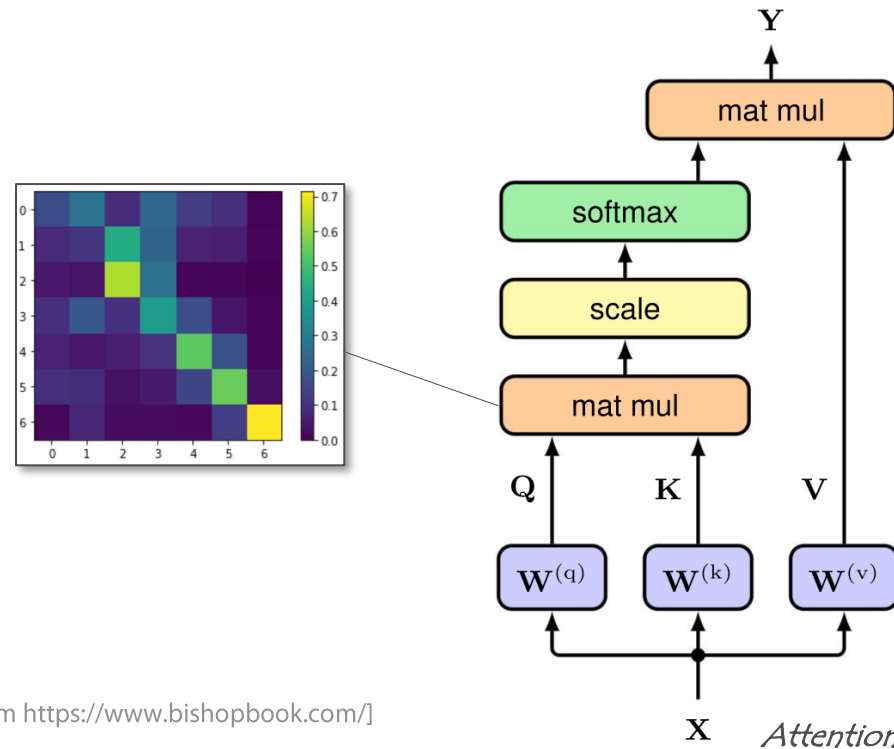
$$\mathbf{K} = \mathbf{X}\mathbf{W}_k^T$$

$$\mathbf{V} = \mathbf{X}\mathbf{W}_v^T$$

where:

$$\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v \in \mathbb{R}^{n \times d}$$

are **parameters to be trained**



# Scaled Dot-Product *Cross-Attention*

$$Q \in \mathbb{R}^{n \times d}, K \in \mathbb{R}^{n \times d}, V \in \mathbb{R}^{n \times n}$$

$$\text{Attention}(Q, K, V) := \text{softmax} \left( \frac{QK^T}{\sqrt{d}} \right) V \in \mathbb{R}^{n \times n}$$

In *cross-attention* blocks the query  $Q$  comes from one source, while  $K$  and  $V$  are from another:

$$Q = X^{[d]} W_q^T$$

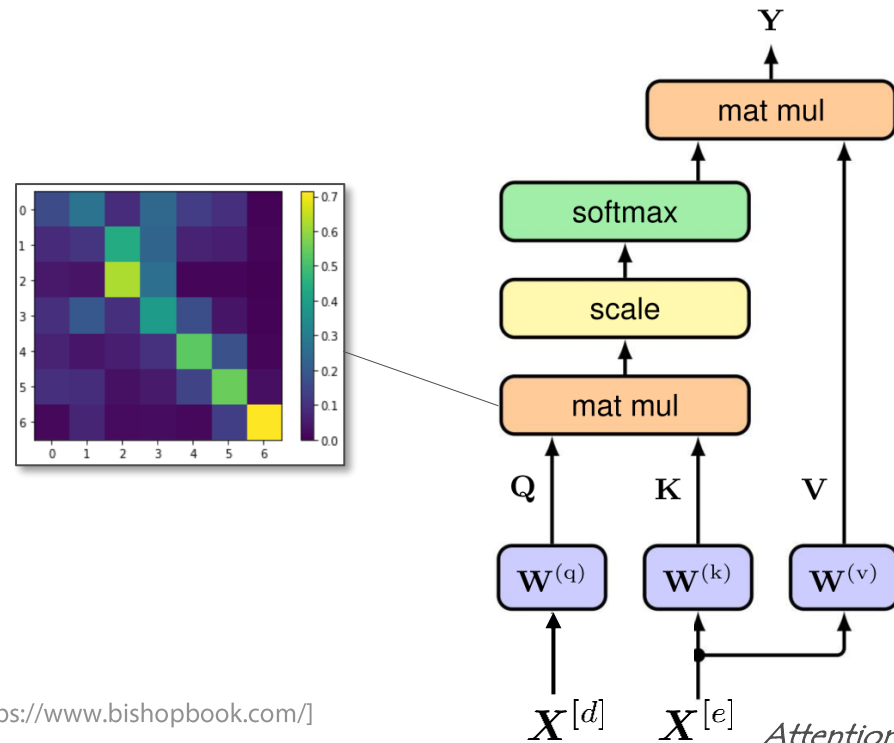
$$K = X^{[e]} W_k^T$$

$$V = X^{[e]} W_v^T$$

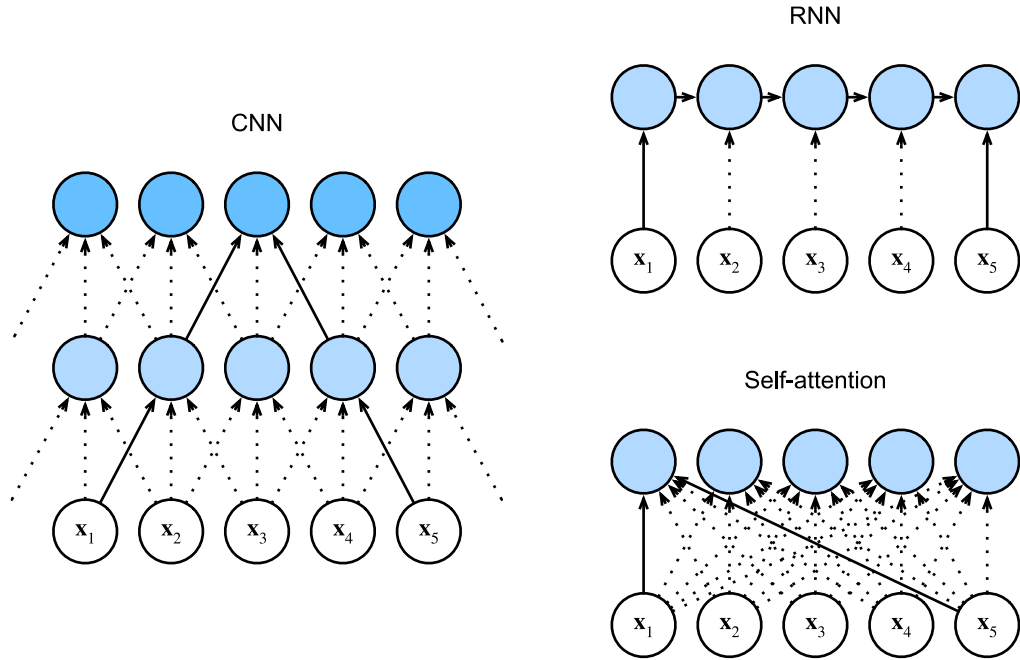
where:

$$W_q^{[d]}, W_k^{[e]}, W_v^{[e]} \in \mathbb{R}^{n \times d}$$

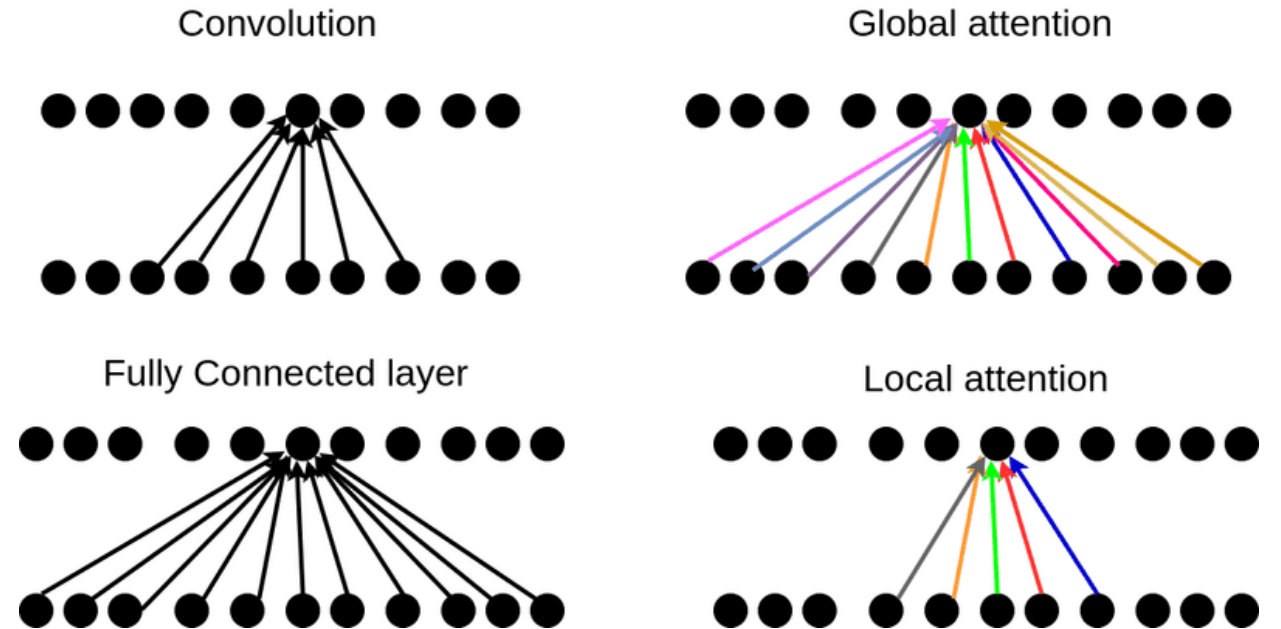
are **parameters to be trained**



# Attention vs Convolution vs RNN



[image from <http://d2l.ai/>]



[image from <https://theaisummer.com/attention/>]

# *Positional Encoding*

# Positional Encoding

## ■ Using sine and cosine

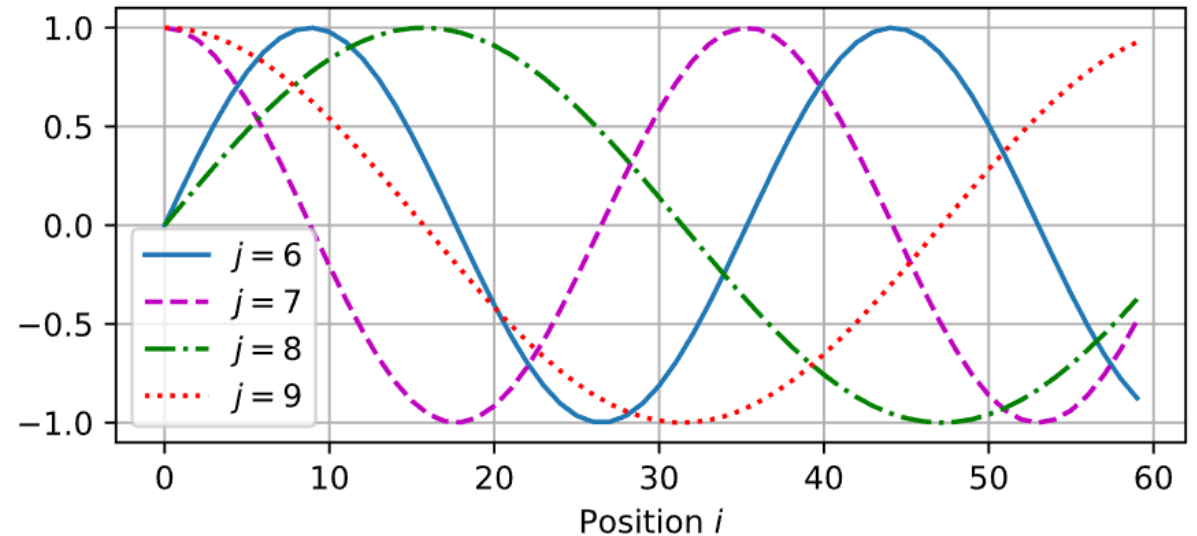
Assume we want to sum a *positional encoding* vector  $\mathbf{p}$  to a data vector  $\mathbf{x}$  ( $\mathbf{x}, \mathbf{p} \in \mathbb{R}^d$ )

$$\mathbf{x} + \mathbf{p}$$

To do so, we can use a *sine – cosine* representation:

$$p_{i,2j} = \sin\left(\frac{i}{s^{2j/d}}\right)$$

$$p_{i,2j+1} = \cos\left(\frac{i}{s^{2j/d}}\right)$$



Where  $i$  is the *position index* of data vector  $\mathbf{x}$  (*query* or *key*),  $j$  is one of the  $d$  components of vector  $\mathbf{p}$  and  $s$  is a suitable scale constant (in the original paper  $s = 1000$ )

# Positional Encoding

## ■ Using sine and cosine

Assume we want to sum a *positional encoding* vector  $\mathbf{p}$  to a data vector  $\mathbf{x}$  ( $\mathbf{x}, \mathbf{p} \in \mathbb{R}^d$ )

$$\mathbf{x} + \mathbf{p}$$

To do so, we can use a *sine – cosine* representation:

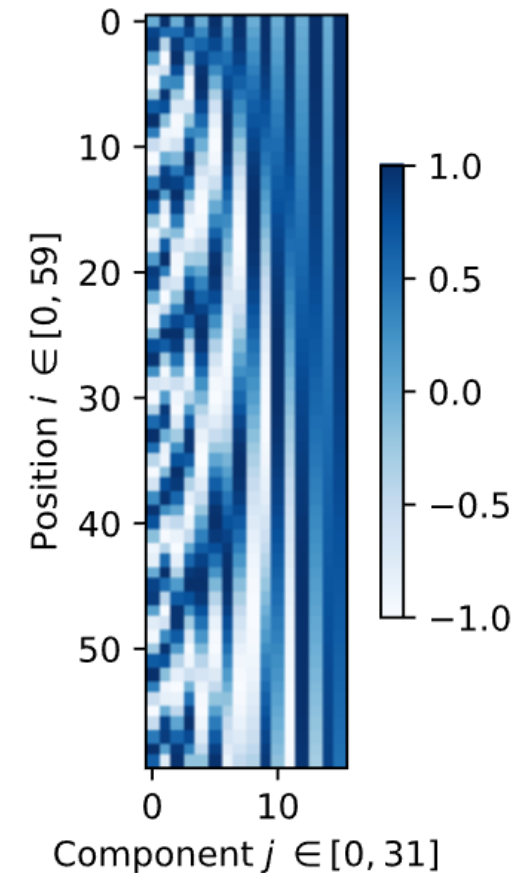
$$p_{i,2j} = \sin\left(\frac{i}{s^{2j/d}}\right)$$

$$p_{i,2j+1} = \cos\left(\frac{i}{s^{2j/d}}\right)$$

*Position can be either absolute or relative, to the position of each query or key*

*In high dimensions, two randomly chosen vectors tend to be orthogonal*

same dimension



# Positional Encoding

## ▪ Relative displacements

$$\begin{aligned} & \begin{bmatrix} \cos(\delta\omega_j) & \sin(\delta\omega_j) \\ -\sin(\delta\omega_j) & \cos(\delta\omega_j) \end{bmatrix} \begin{bmatrix} p_{i,2j} \\ p_{i,2j+1} \end{bmatrix} \\ = & \begin{bmatrix} \cos(\delta\omega_j)\sin(i\omega_j) + \sin(\delta\omega_j)\cos(i\omega_j) \\ -\sin(\delta\omega_j)\sin(i\omega_j) + \cos(\delta\omega_j)\cos(i\omega_j) \end{bmatrix} \\ = & \begin{bmatrix} \sin((i+\delta)\omega_j) \\ \cos((i+\delta)\omega_j) \end{bmatrix} \\ = & \begin{bmatrix} p_{i+\delta,2j} \\ p_{i+\delta,2j+1} \end{bmatrix}, \end{aligned}$$

*Displacements can be represented via a linear transformation.  
This means that relative positions can be learnt*

# *Transformers: a network architecture*

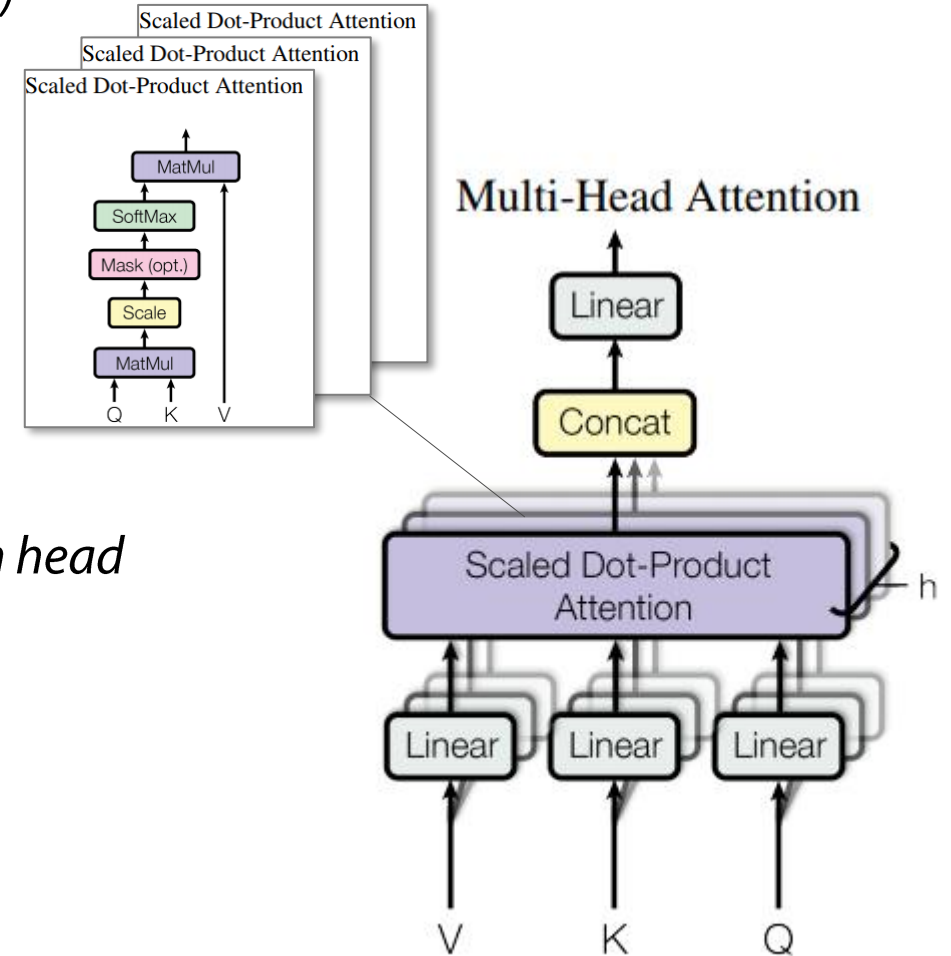
# Multiple Attention Heads

Multi-head attention consists of four parts:

1. Linear layers (*i.e., fully connected, no activation function*)
2. Scaled dot-product attention
3. Output concatenation
4. Final linear layer

*Each input combination of Q (query), K (key), V (values) is passed to each a separate linear layer hence to an attention head*

*The output of multiple attention heads is the concatenated and fed to a final linear layer*

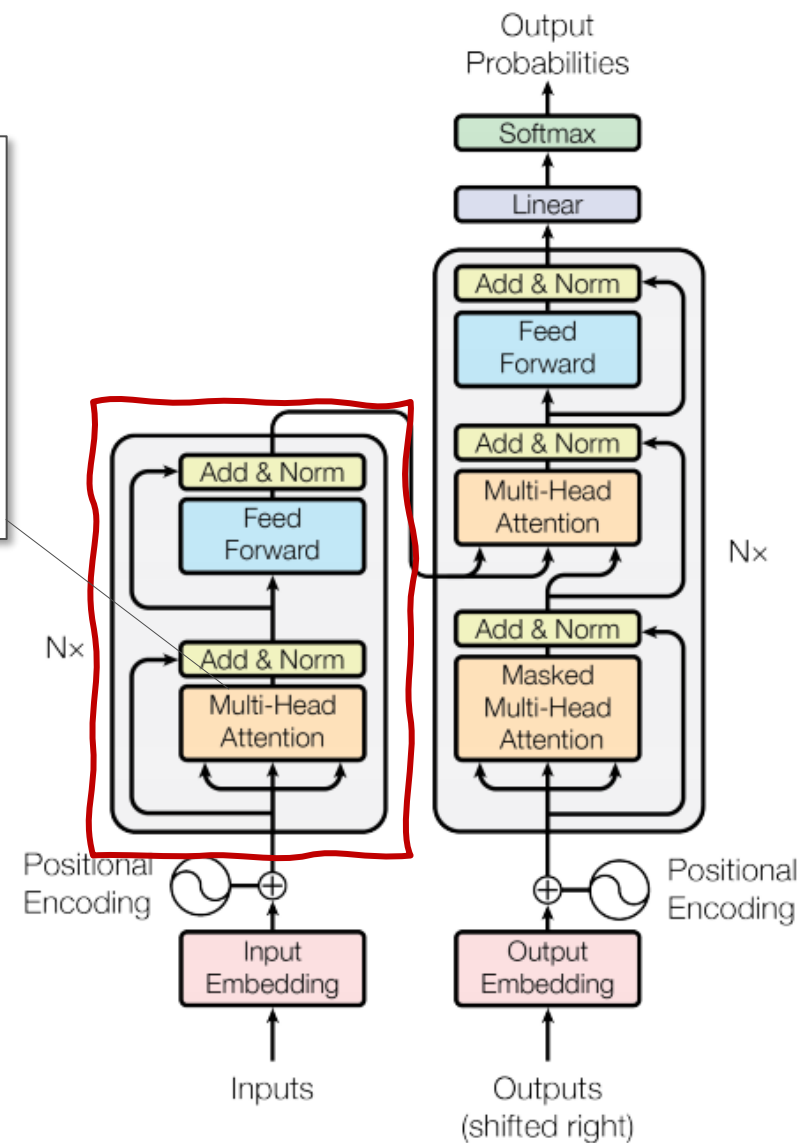
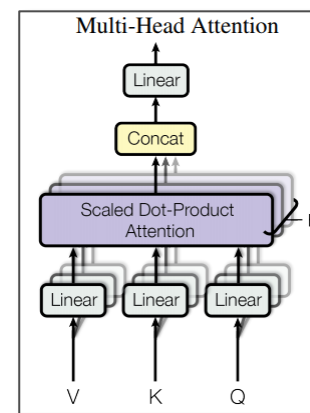


# Encoder Layer

Each encoder layer includes:

1. Multi-head attention
2. Addition (*ResNet style*)
3. Normalization (*per each input*)
4. Feed-forward network  
(*one hidden layer with ReLU plus one linear layer*)
5. Addition
6. Normalization

*There could be many encoder layers in the overall architecture*



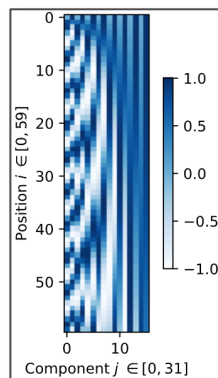
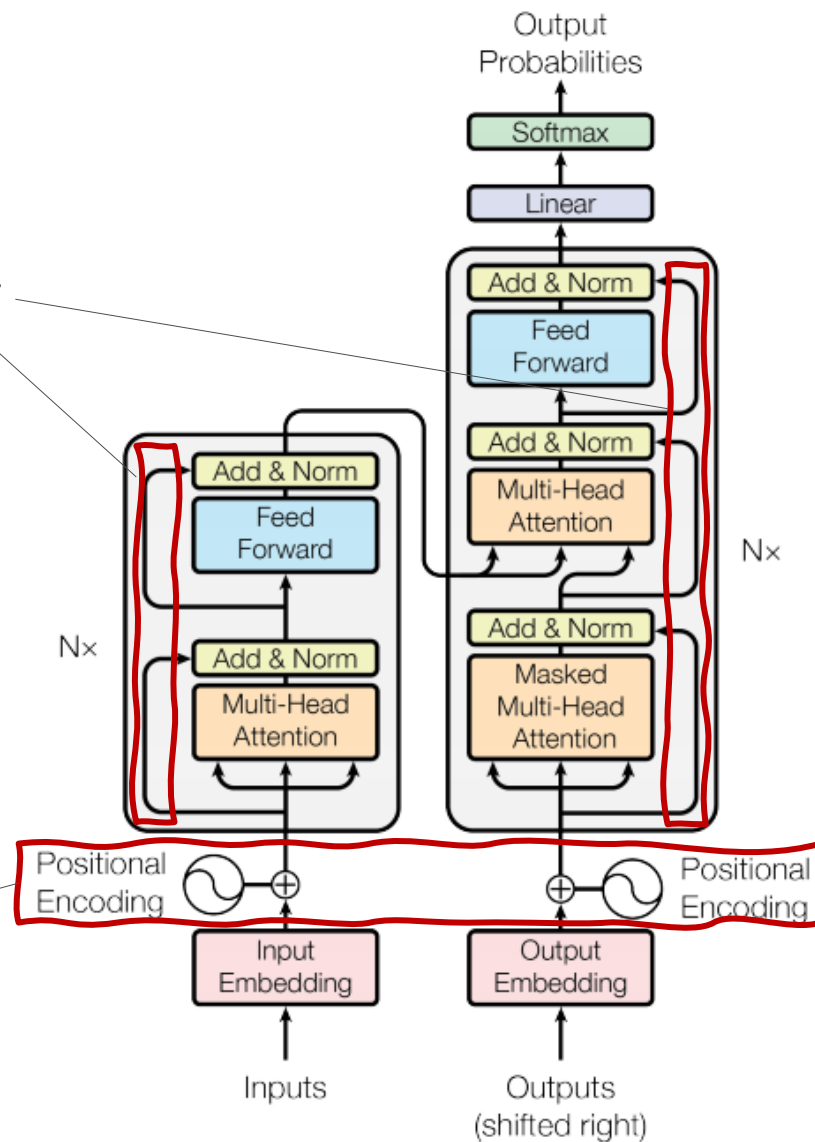
# Encoder Layer

Each encoder layer includes:

1. Multi-head attention
2. Addition (*ResNet style*)
3. Normalization (*per each input*)
4. Feed-forward network  
(*one hidden layer with ReLU plus one linear layer*)
5. Addition
6. Normalization

*There could be many encoder layers in the overall architecture*

*residual connections*



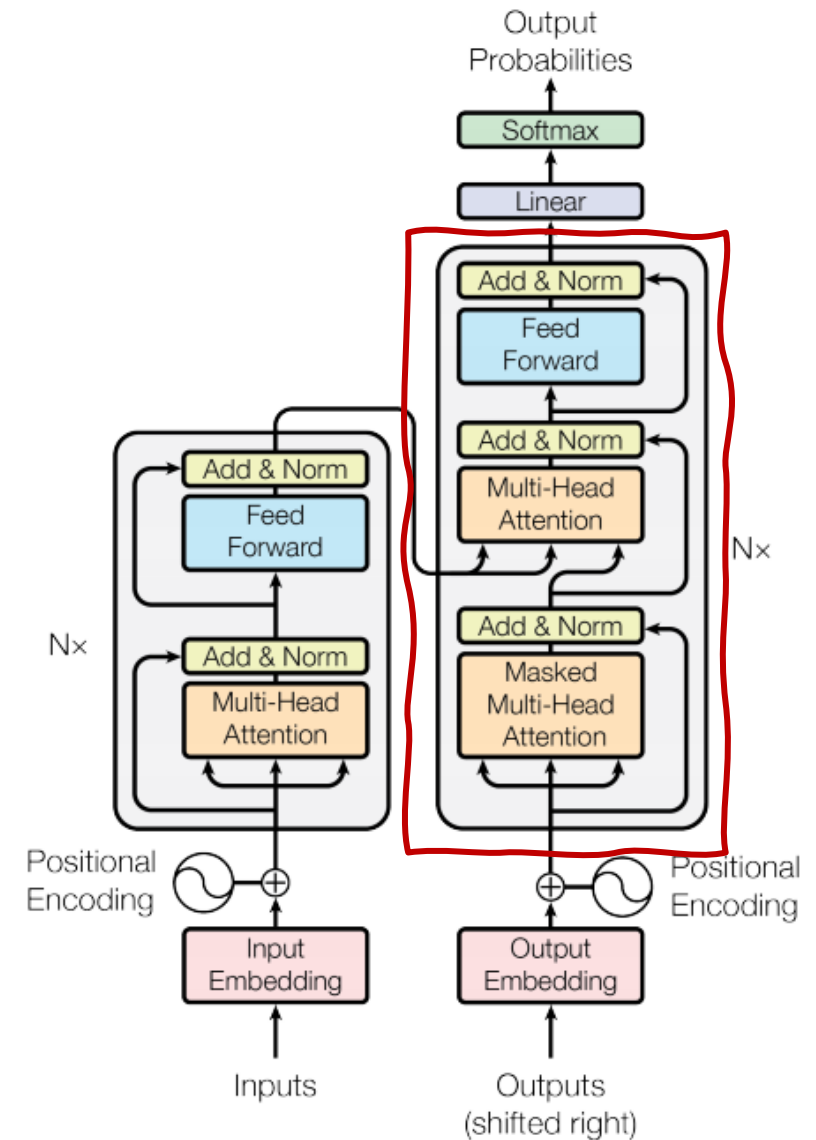
[image from <https://arxiv.org/pdf/1706.03762.pdf>]

# Decoder Layer

Each decode layer includes:

1. Multi-head attention
2. Addition
3. Normalization
4. Multi-head attention  
*values and keys come from the encoder output  
while queries come from the previous decoder layer*
5. Addition
6. Normalization
7. Feed-forward network  
*(one hidden layer with ReLU plus one linear layer)*
8. Addition
9. Normalization

*There could be many decoder layers in the overall architecture*



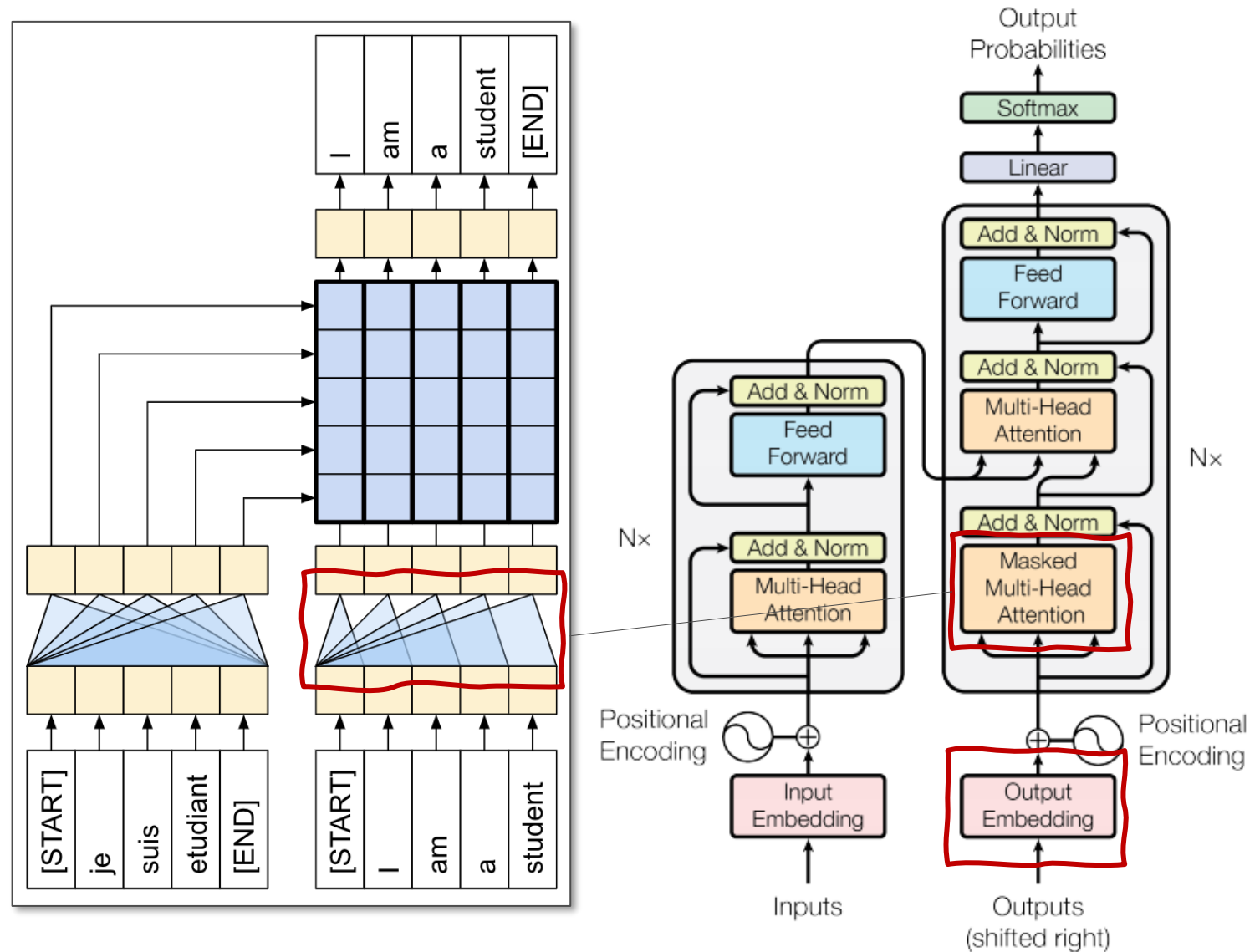
# Decoder Layer

Why masked multi-head attention in the decoder layer?

The production of the output is incremental: one word at time

The output embedding 'input' can only see what has been generated thus far

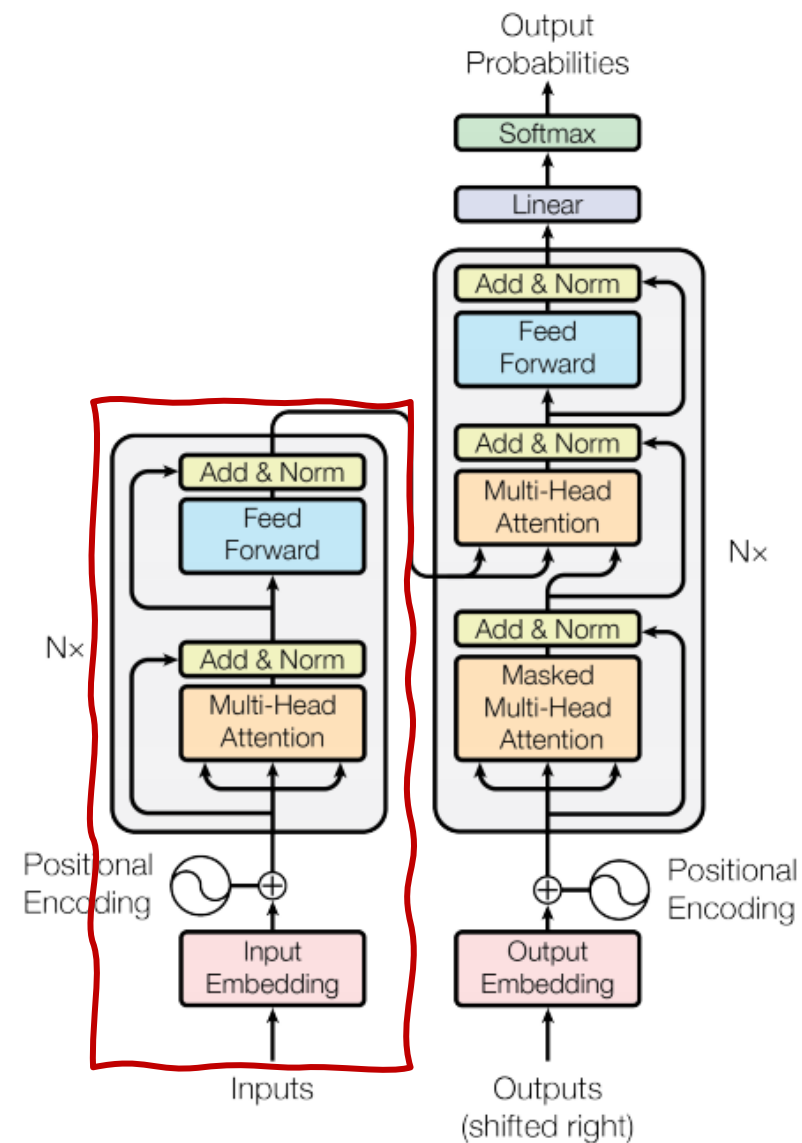
Masks are not trained, they are 'superimposed' as the generation process advances



# Encoder

The encoder block includes:

1. Input embedding (*word2vec* style)
2. Positional encoding
3. Addition
4. N encoder layers

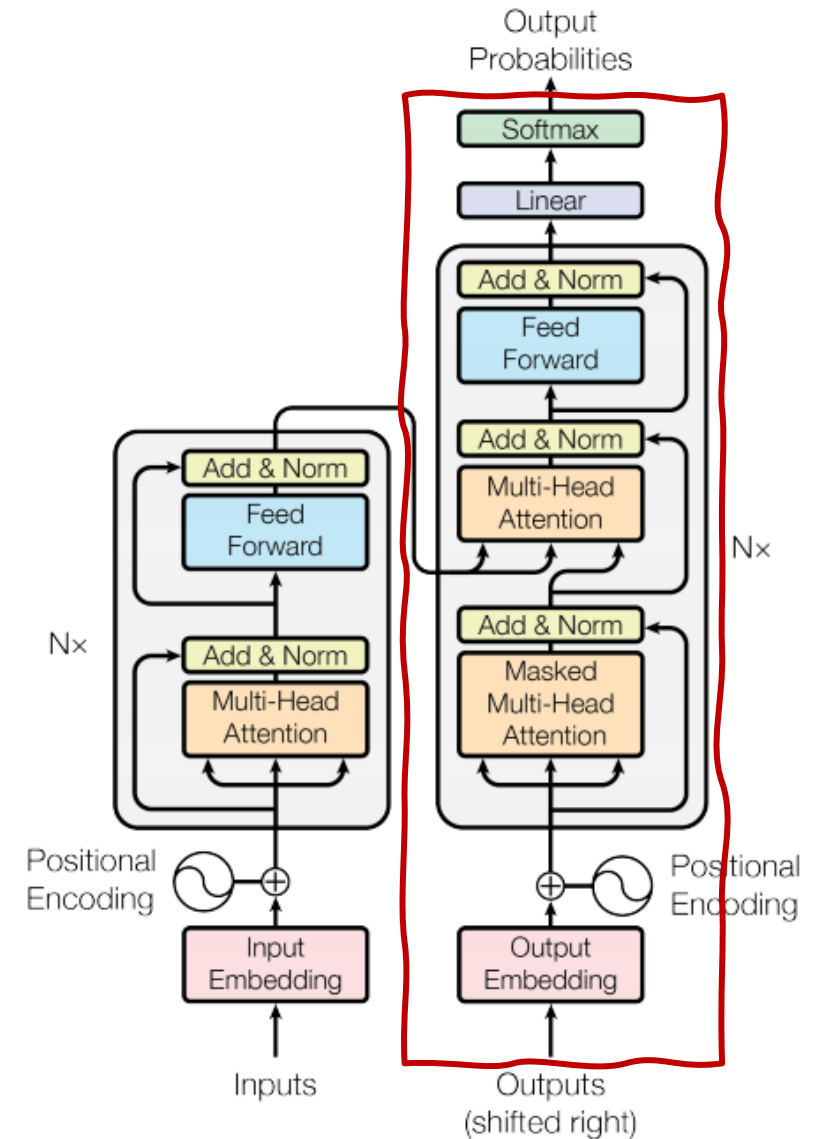


[image from <https://arxiv.org/pdf/1706.03762.pdf>]

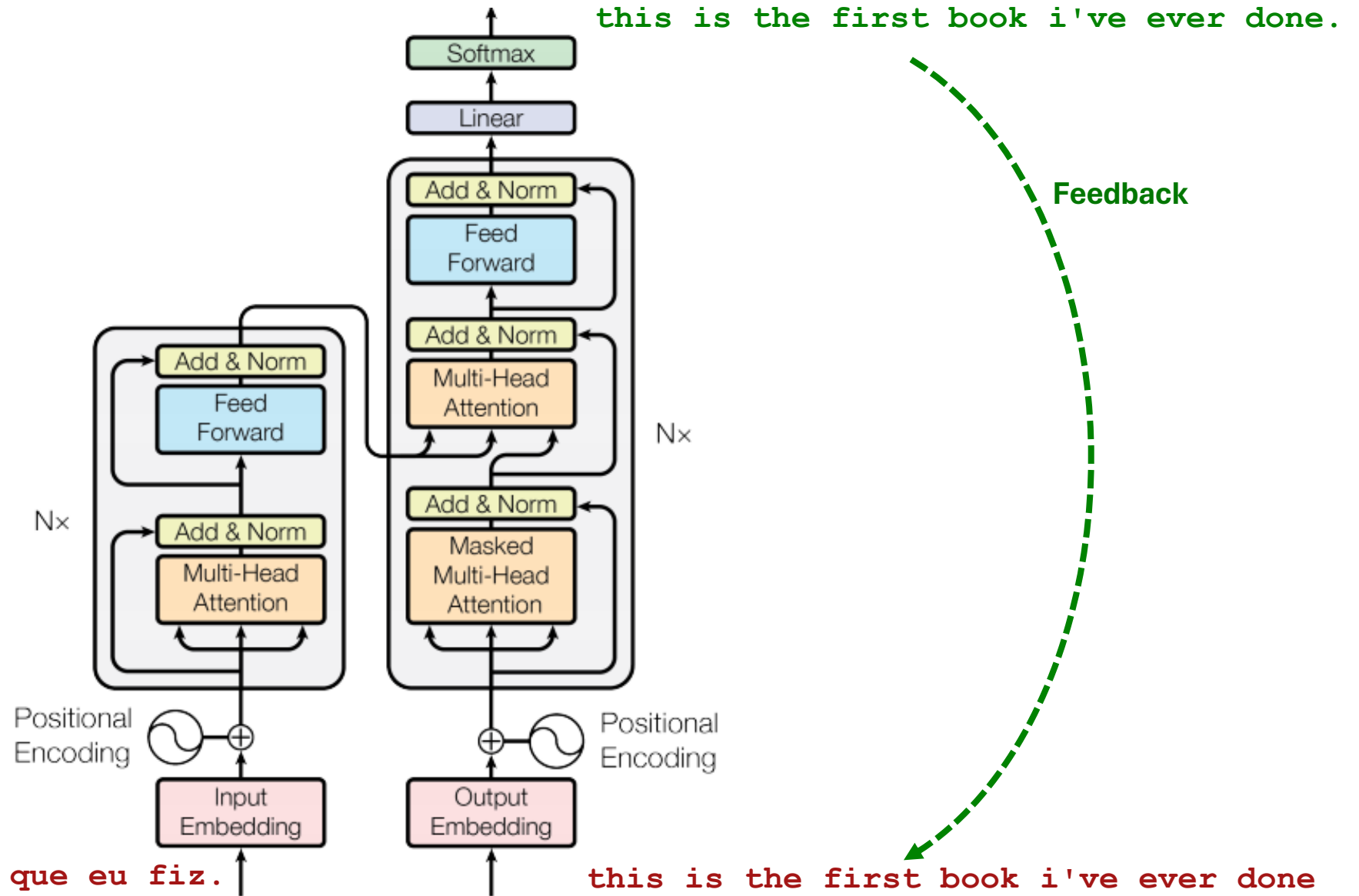
# Decoder

The decoder block includes:

1. Output embedding (*word2vec style*)  
*It encodes the output produced so far*
2. Positional encoding
3. Addition
4. N decoder layers  
Each connected to a corresponding encoder layer
5. Linear layer
6. Softmax layer  
*It predicts the next token in the sequence*

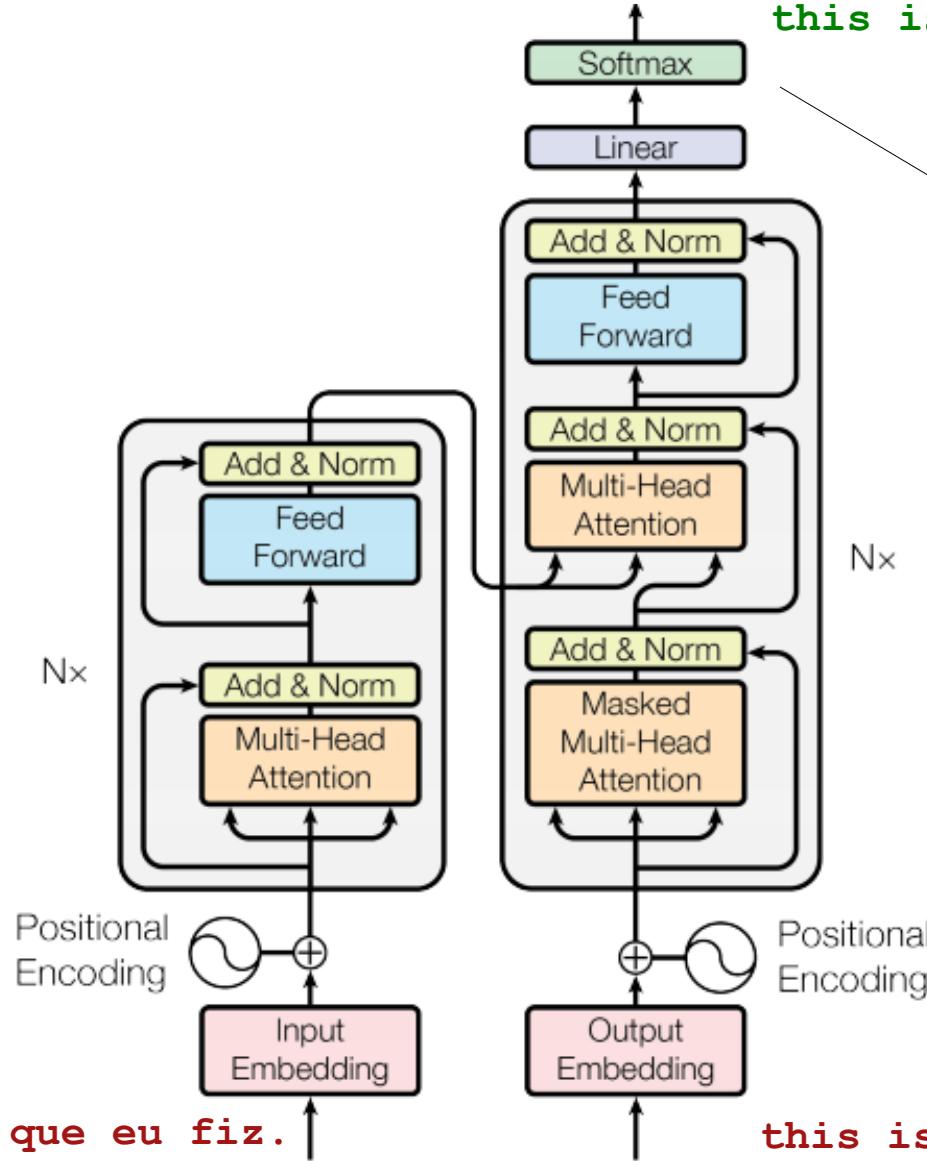


# Translator (Encoder-Decoder)



# Translator (Encoder-Decoder)

**During Training**



this is the first book i've ever done.

This is the predicted output

Loss is computed on the softmax

This one input  
(in one data item)

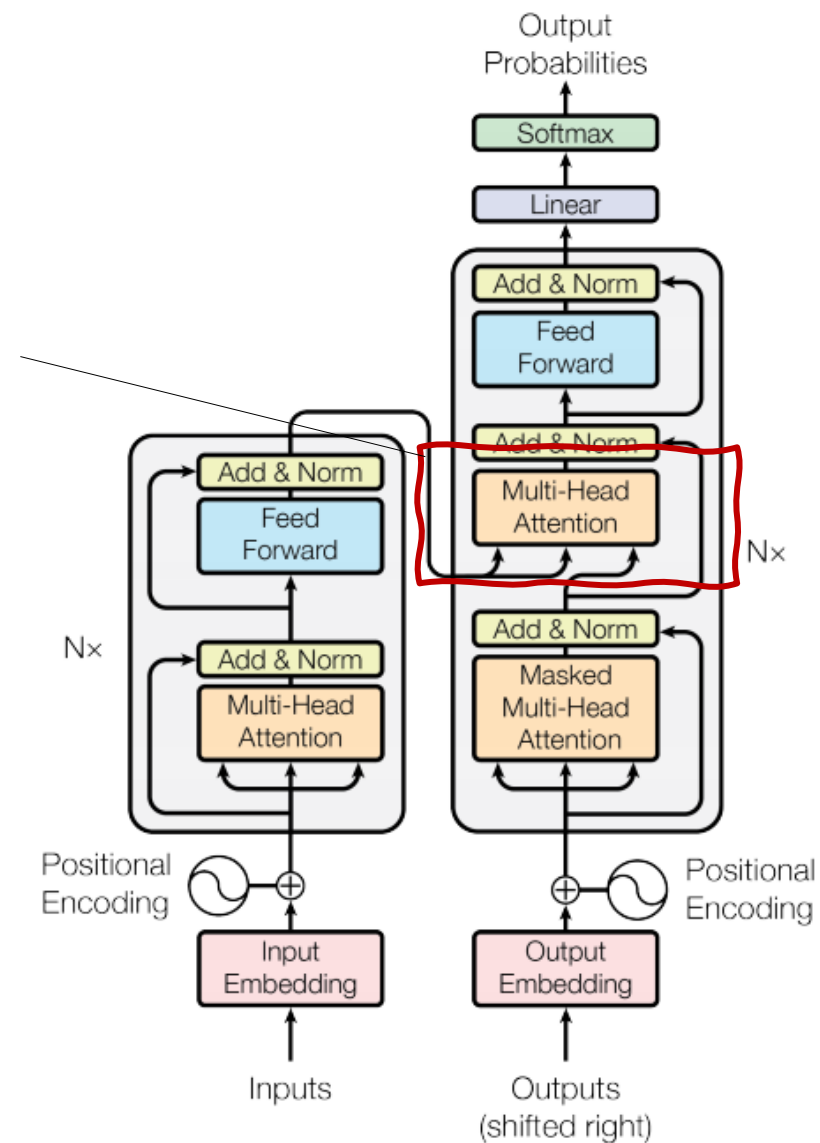
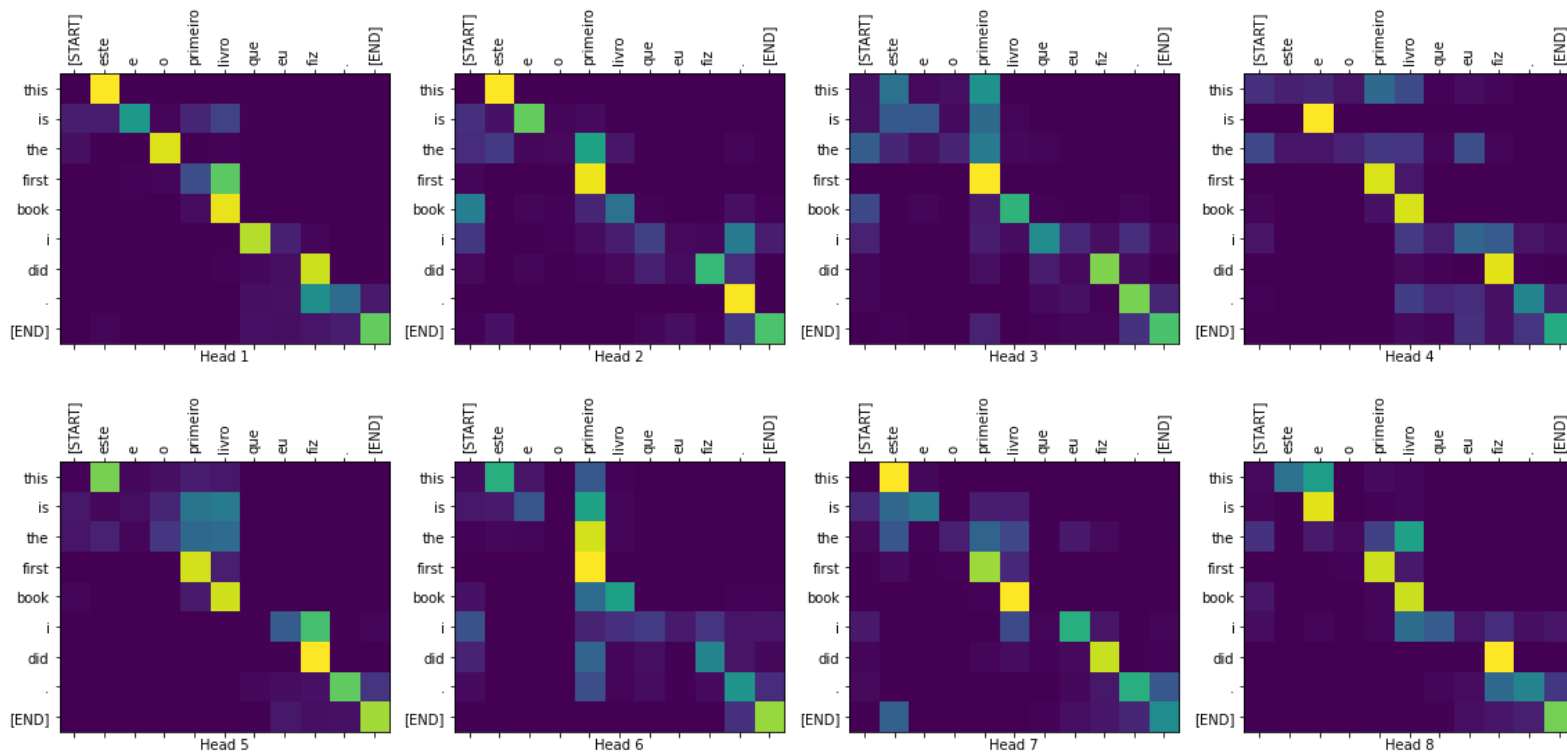
This the true output  
(teacher forcing)

este é o primeiro livro que eu fiz.

this is the first book i've ever done

# Attention Maps

Multiple heads, mixing layer ("Mixer"), topmost decoder block

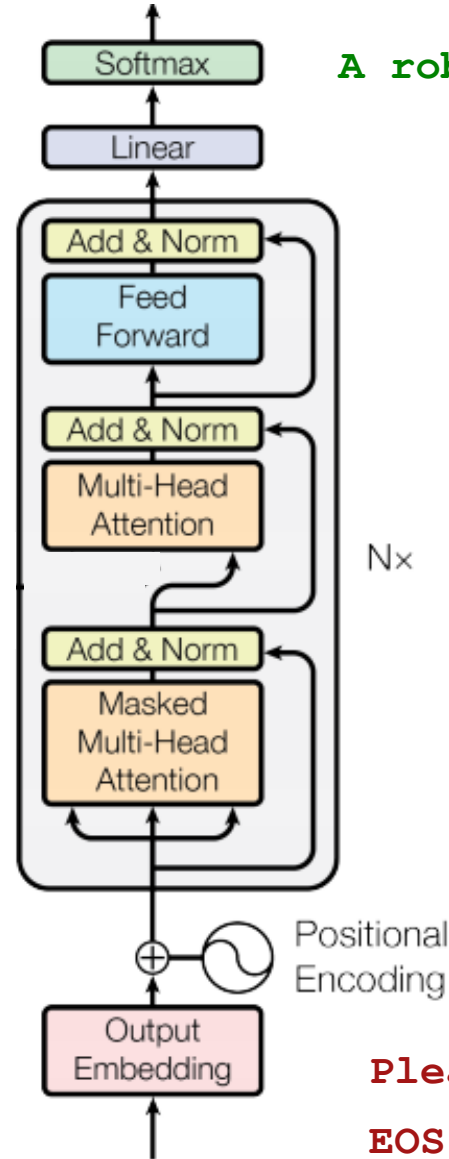


[image from <https://arxiv.org/pdf/1706.03762.pdf>]

# *Attention in Blocks*

1. Encoder (all layers)  
*input tokens over input tokens (self-attention)*
2. Decoder (lower layers)  
*output tokens over output tokens (self-attention)*
3. Decoder (top layer, "Mixer")  
*input tokens over output tokens (cross-attention)*

# Chat (Decoder-only)



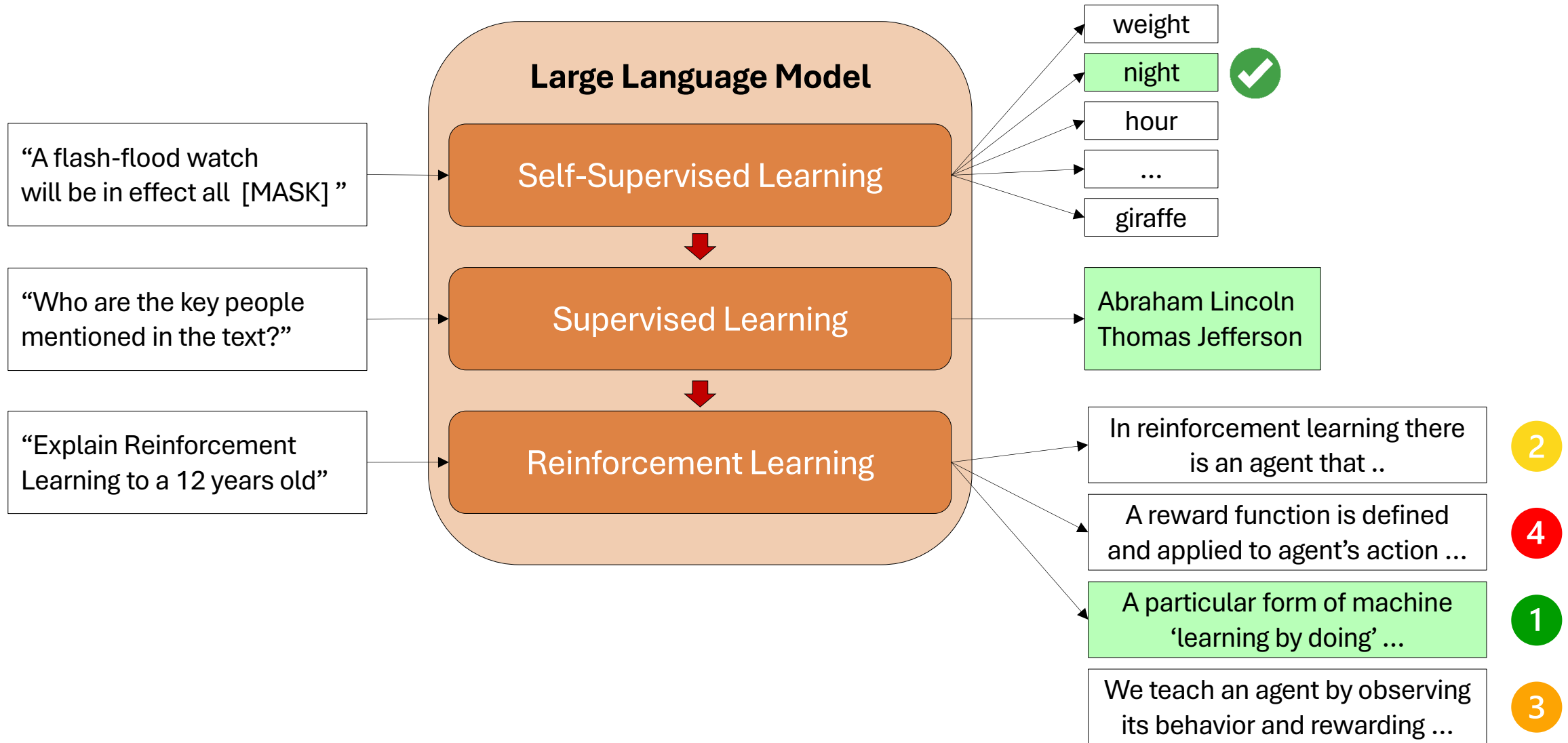
A robot may not injure a human being

*When in chat mode, the transformer network only performs text completions*

Please recite the first law of robotics

EOS A robot may not injure a human

# LLMs: learning in three steps



[image adapted from <https://snorkel.ai/blog/large-language-model-training-three-phases-shape-llm-training/>]

# LLMs: most used text corpora for training

Corpus Name	Description	Size / Language Coverage	Notable Usage / Models
Common Crawl	Massive web crawl dataset, billions of web pages	Petabytes; multilingual	GPT, LLaMA, Falcon, BLOOM
Common Corpus	Largest public domain dataset, diverse and multilingual	2 trillion tokens; multilingual	Open LLMs, research
Wikipedia	Cleaned encyclopedic articles, all topics	Millions of articles; multilingual	GPT, T5, RoBERTA
BooksCorpus	11,000+ unpublished books, narrative text	740M–985M words; English	RoBERTA, XLNet
OpenWebText	Web pages curated to mimic OpenAI's WebText (Reddit-based)	Billions of tokens; English	GPT-2, GPT-Neo
C4 (Colossal Clean Crawled Corpus)	Cleaned English web crawl, deduplicated	750GB; English	T5, MPT-7B
The Pile	22 diverse academic/professional datasets, broad coverage	800GB; mostly English	GPT-Neo, LLaMA, OPT
ROOTS	Multilingual, deduplicated web/code/crowdsourced text	1.6TB; 59 languages	BLOOM
Google News	News articles from 70,000+ sources	Large; multilingual	Topic modeling, classification
Starcoder Data	Programming code from GitHub, Jupyter, etc.	783GB; 86 languages (code)	Starcoder, CodeGen

# ChatGPT 4o: So clever even when hallucinates..

Who was the first elephant to swim across the Strait of Messina?



The first elephant to swim across the Strait of Messina was a famous circus elephant named Tusko. This remarkable feat occurred in 1979. Tusko, weighing over three tons, made the swim from the Italian mainland to the island of Sicily, covering a distance of about three kilometers. This event was widely publicized and showcased the impressive abilities of elephants, demonstrating their strength and endurance.

Could you provide more details?



I apologize for the confusion earlier. It seems there was a misunderstanding. There isn't a widely known or documented case of an elephant named Tusko or any other elephant swimming across the Strait of Messina.

In historical context, the notion of elephants in relation to the Strait of Messina often brings to mind Hannibal's famous crossing of the Alps with elephants during the Second Punic War. However, these elephants did not swim across the Strait of Messina.

[Conversation made by myself with ChatGPT 4o, on May 17, 2024]

# Links

[https://d4mucfpksywv.cloudfront.net/better-language-models/language\\_models\\_are\\_unsupervised\\_multitask\\_learners.pdf](https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf)

Language Models are Unsupervised Multitask Learners

<https://arxiv.org/pdf/1904.02679>

Visualizing Attention in Transformer-Based Language Representation Models

<https://arxiv.org/pdf/2203.02155>

Training language models to follow instructions with human feedback

<https://medium.com/@row3no6/why-chatgpt-uses-decoder-only-eaf0223143e6>

Why ChatGPT Uses Decoder-Only

<https://cameronwolfe.substack.com/p/decoder-only-transformers-the-workhorse>

Decoder-Only Transformers: The Workhorse of Generative LLMs