UNIVERSITÀ DI PAVIA

Deep Learning

09- Attention and Transformers

Marco Piastra

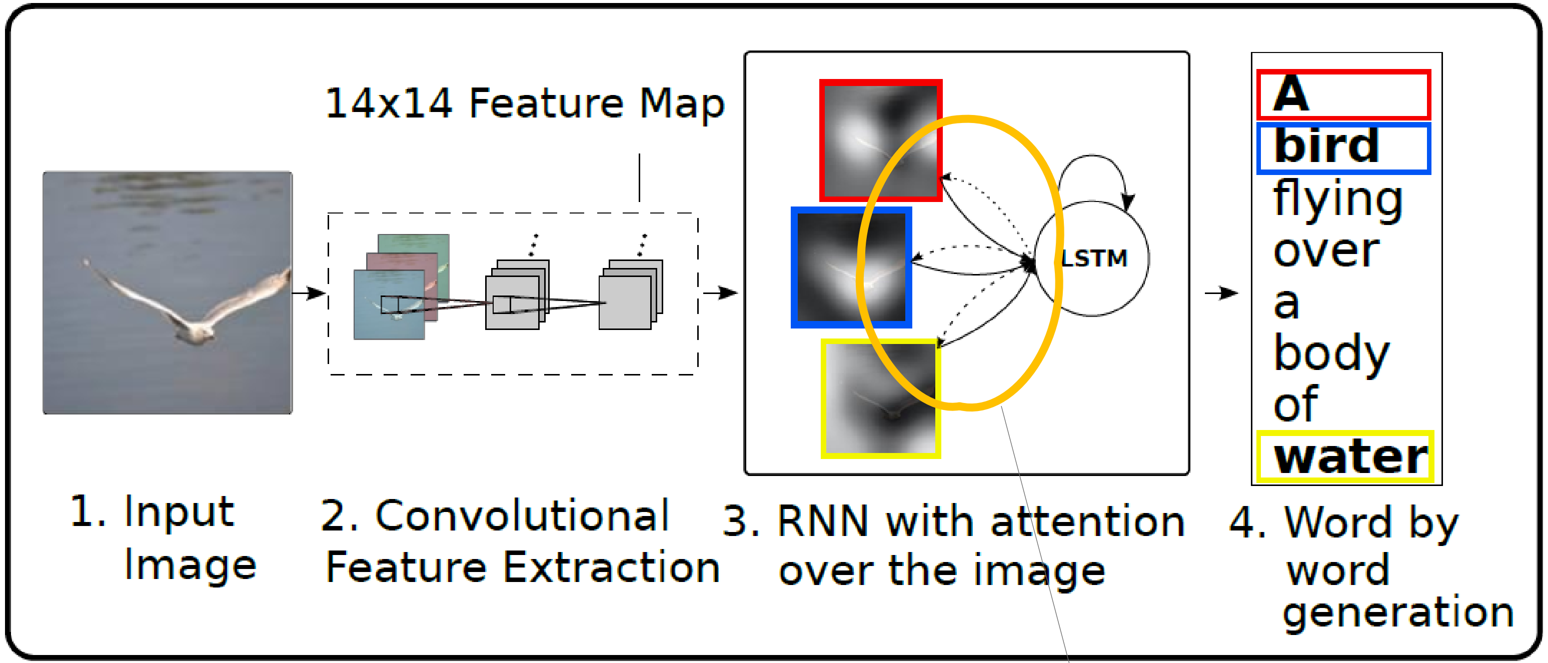
This presentation can be downloaded at:
http://vision.unipv.it/DL

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]

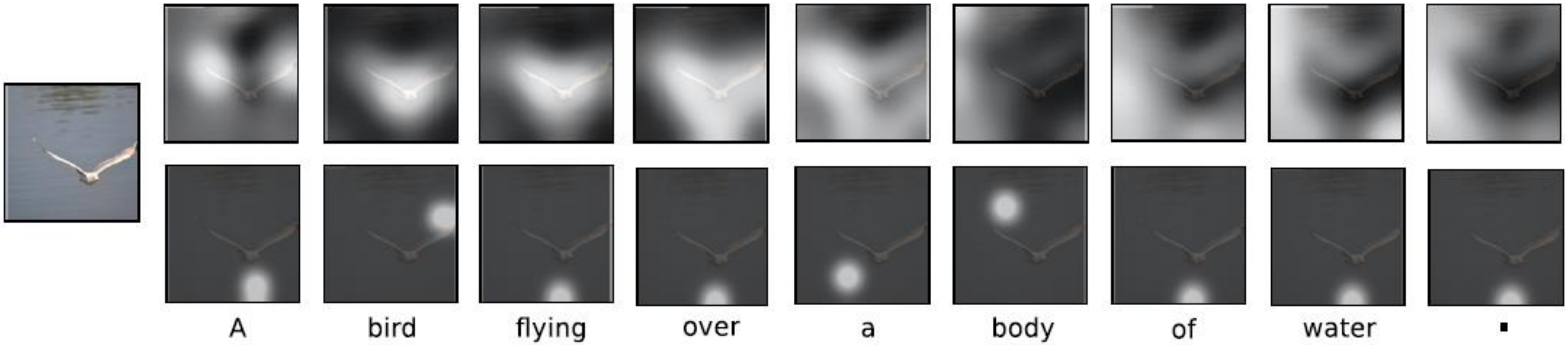


The 'trick' is here:
when generating each word
the LSTM focuses on a specific
region in the image

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]

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.

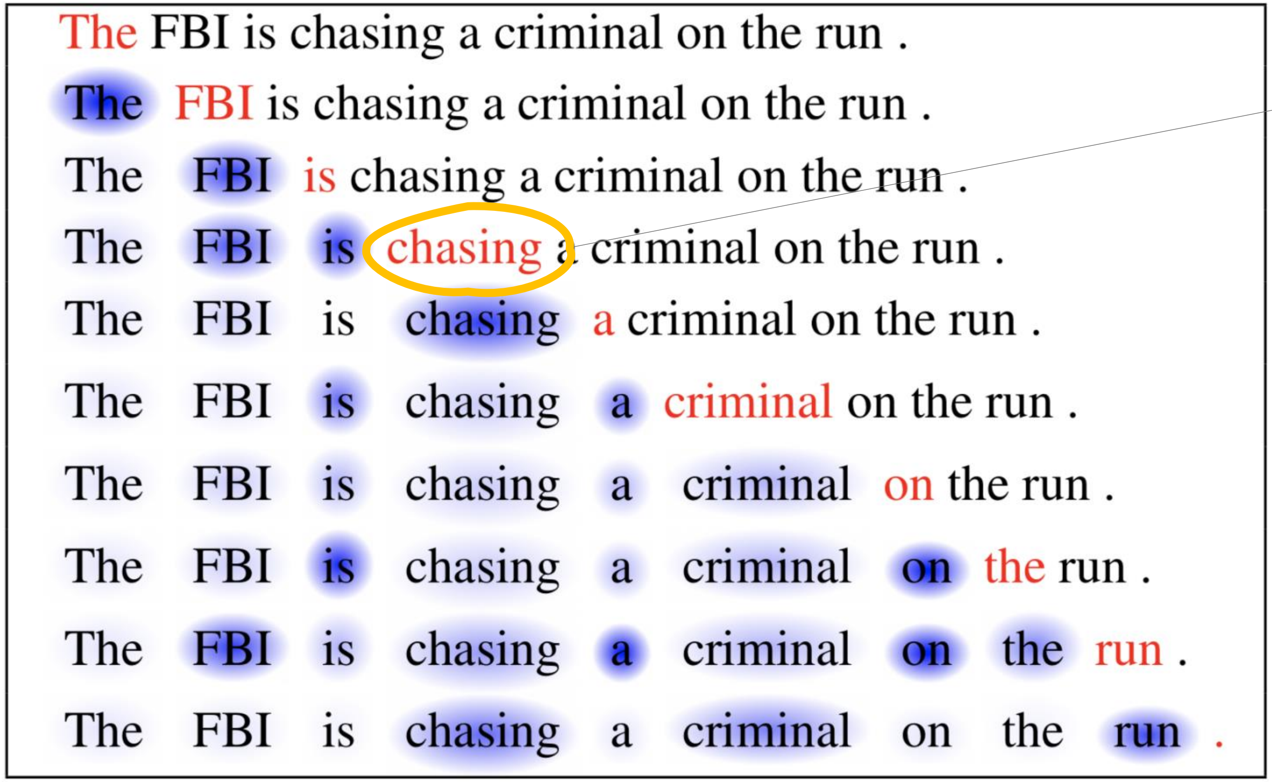
A giraffe standing in a forest with trees in the background.

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]

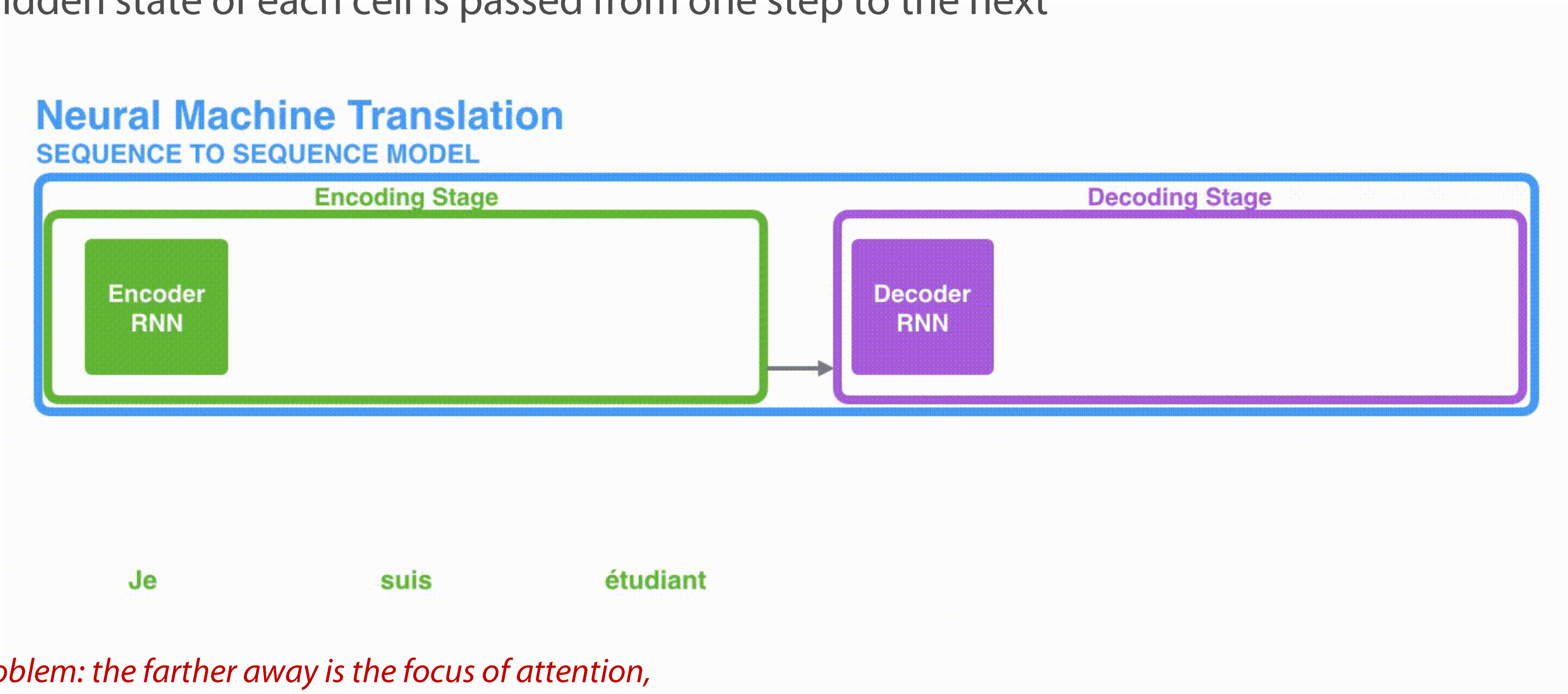


Current word being read

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

- ***seq2seq model for machine translation***

  Each RNN cell could be either a LSTM or a GRU
  The hidden state of each cell is passed from one step to the next

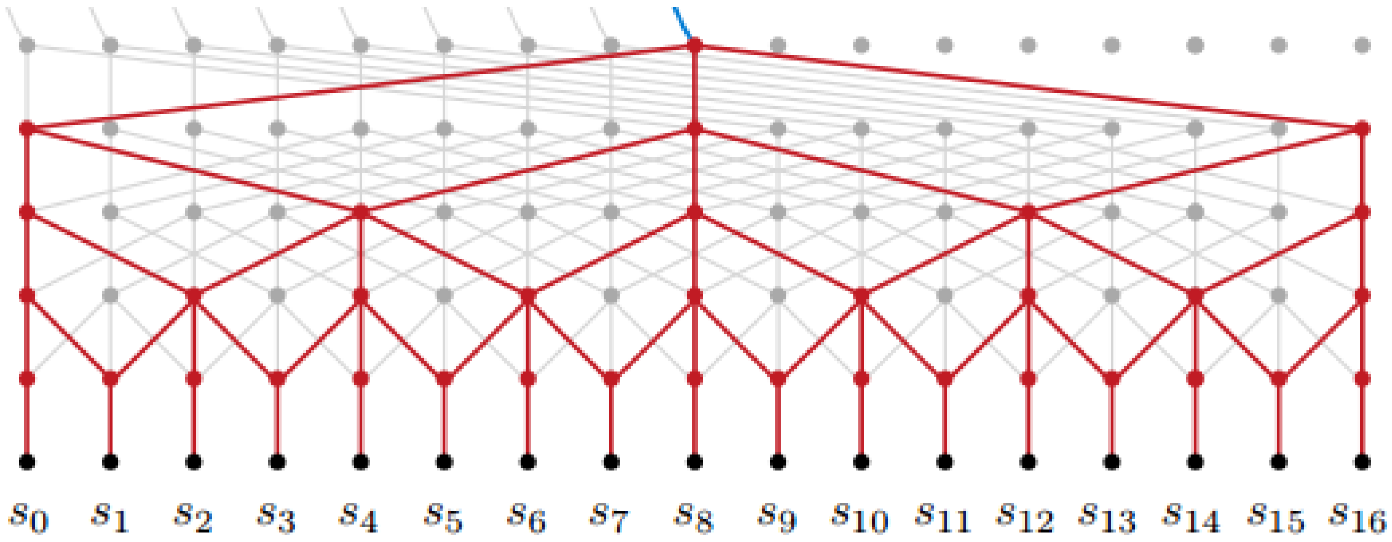


Problem: the farther away is the focus of attention,
the more difficult it gets for the system to keep track of it

[image from https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/]

- ***Progressively widening receptive field***

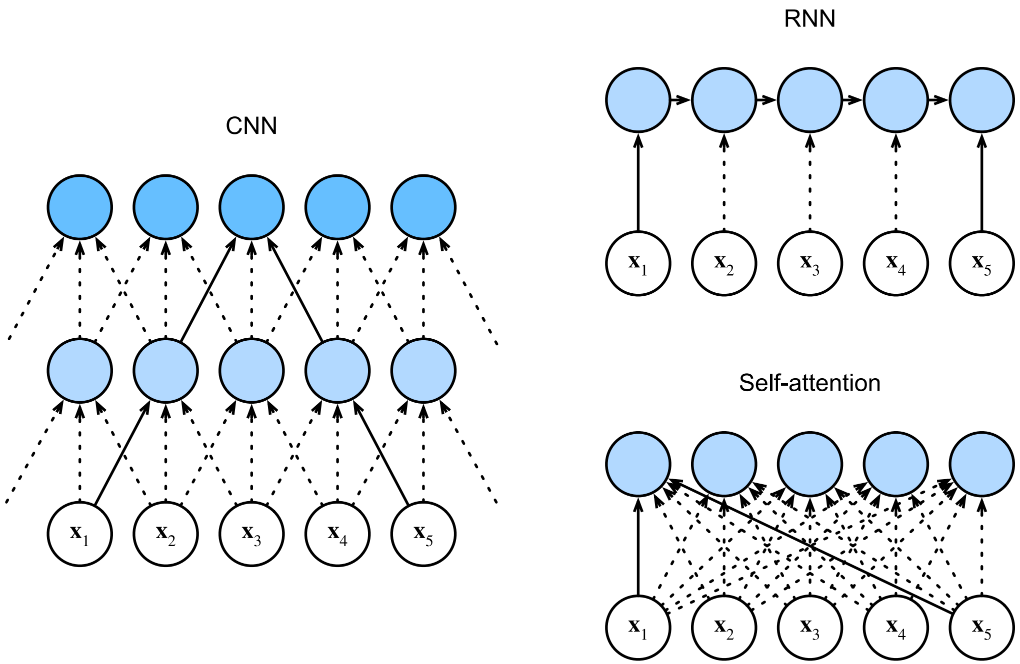
  Consider 1D convolution, size 3:
  the receptive field of each filter grows progressively

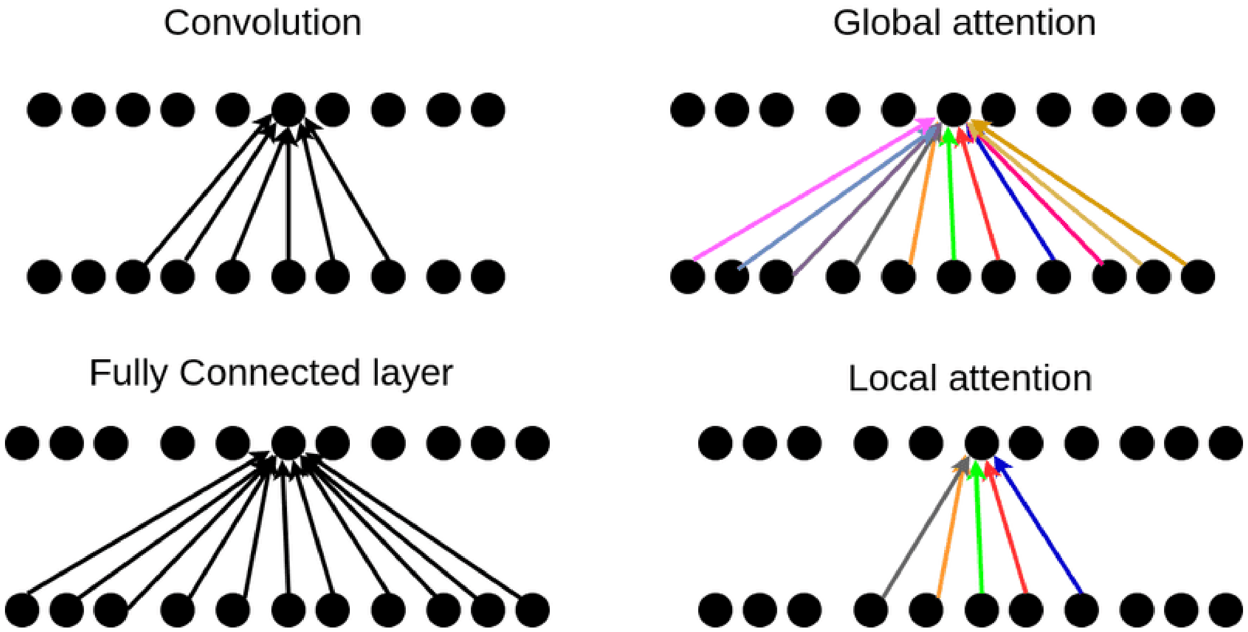


Problem: four layers are required in this case to have a receptive field of 16

 [image from https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/]

Attention vs Convolution vs RNN





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

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

Attention as a Kernel

Attention Pooling

Consider an input-output relation and a dataset

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

In general *Attention Pooling* is defined as a function on data items

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

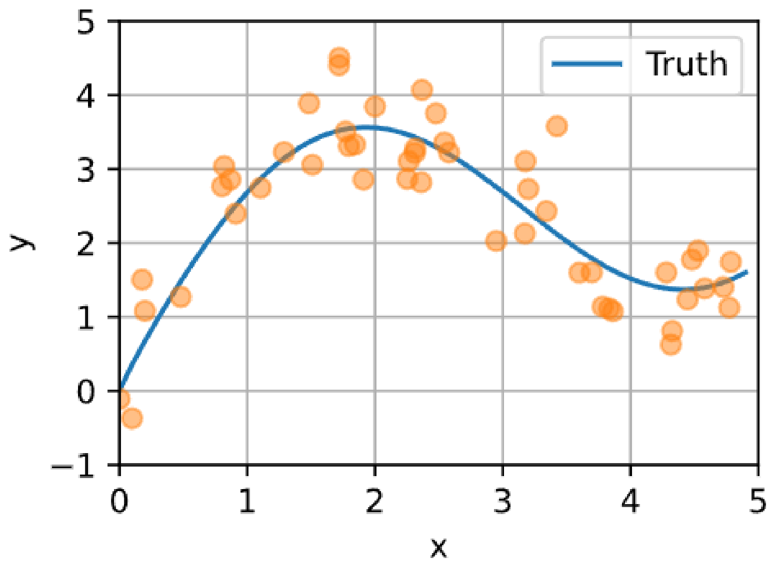
Example:

$$f^*(x) = 2\sin(x_i) + x_i^{0.8}$$

$$D := \{(x^{(i)}, y^{(i)})\}_{i=1}^N$$

Dataset is noisy

$$y^{(i)} = f^*(x^{(i)}) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, 0.5)$$



[image from http://d2l.ai/chapter_attention-mechanisms/]

Attention as Kernel

- ***Attention Pooling***

Consider an input-output relation

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

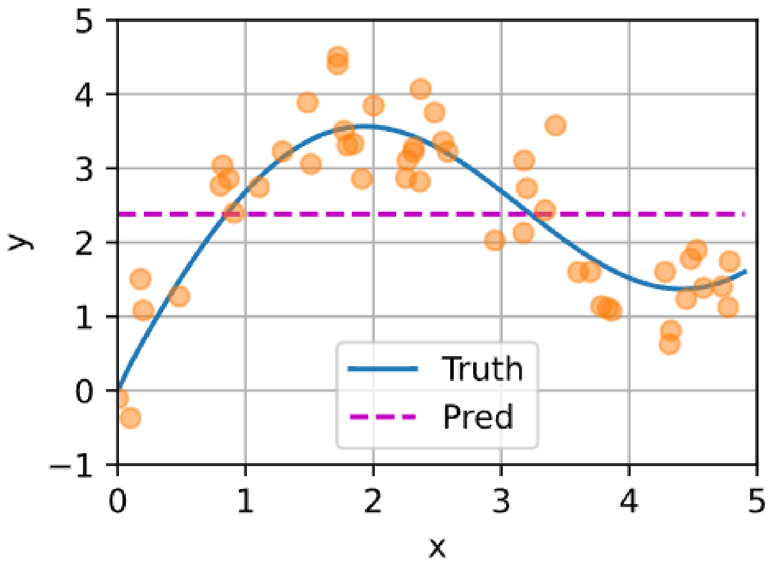
Attention Pooling is defined as a function on each input component

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

Example:

Global average (i.e., *no attention*)

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



[image from http://d2l.ai/chapter_attention-mechanisms/]

Attention as Kernel

- ***Attention Pooling***

Consider an input-output relation

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

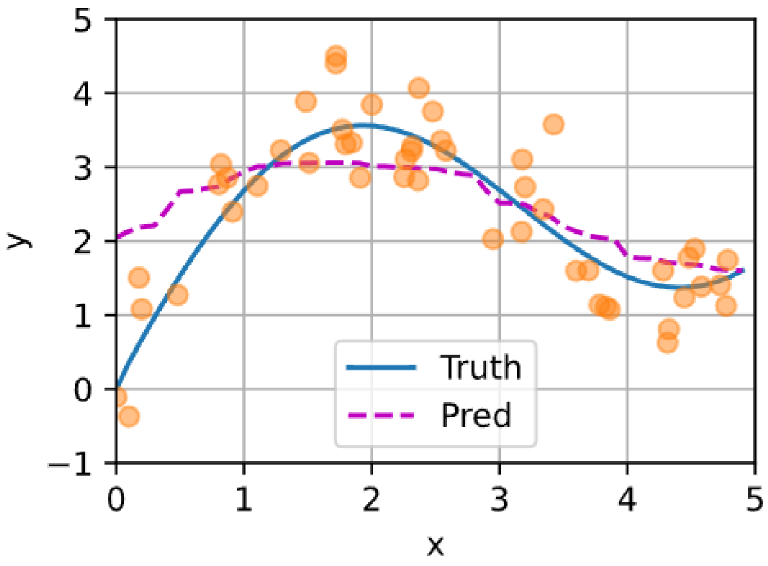
Attention Pooling is defined as a function on each input component

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

Example:

Gaussian Kernel [Nadaraya & Watson, 1964]

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



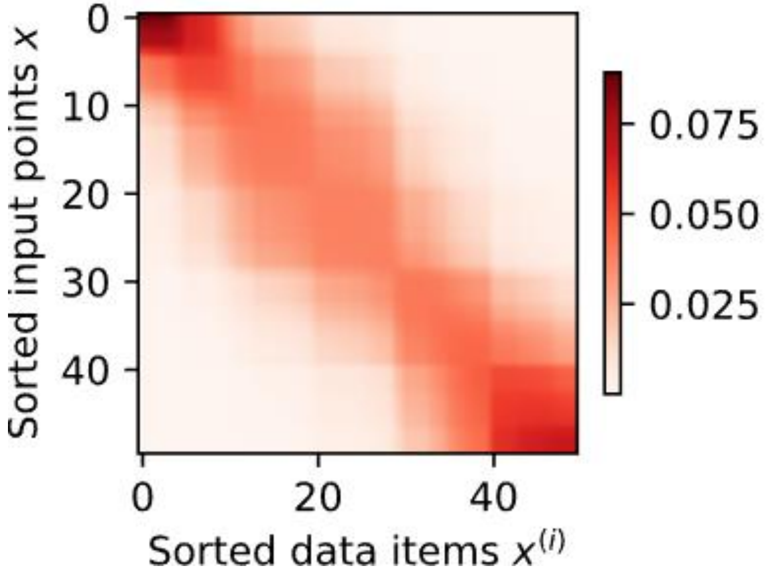
[image from http://d2l.ai/chapter_attention-mechanisms/]

Attention as Kernel

- ***Gaussian Kernel and Softmax***

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

'Attention Map'



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

$$= \mathrm{softmax}\left(-\frac{1}{2}(x - x^{(i)})^2\right)$$

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/]

Attention as Kernel

- ***(Simple) Parametric Attention Pooling***

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

$$= \operatorname{softmax}\left(-\frac{1}{2}(x - x_i)^2 w\right)$$

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

$$\Rightarrow\ \sigma^2 = \frac{1}{w}$$

This requires training of the (unique) parameter w

Consider an MSE *loss* function:

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

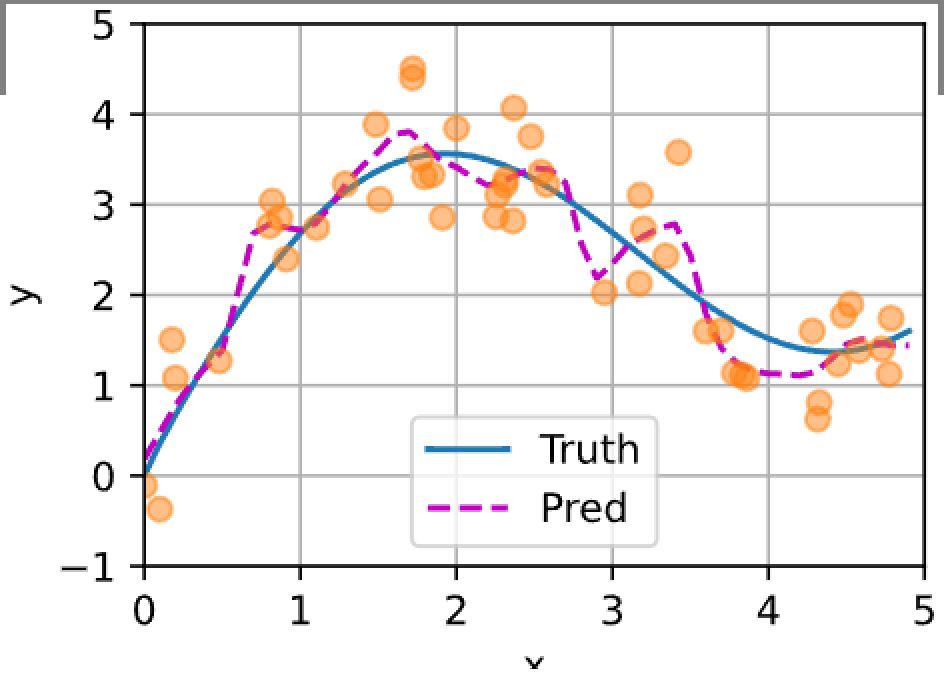
and perform *gradient descent*

Attention as Kernel

- ***(Simple) Parametric Attention Pooling***

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

$$= \mathrm{softmax}\left(-\frac{1}{2}(x - x_i)^2 w\right)$$



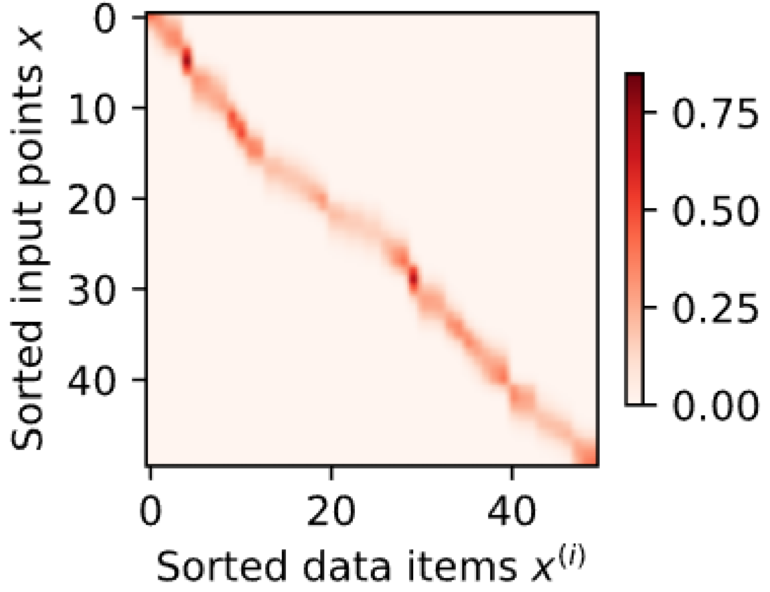
This requires training of the (unique) parameter w

Consider an MSE *loss* function:

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

and perform *gradient descent*

The (Gaussian) attention field becomes 'sharper'



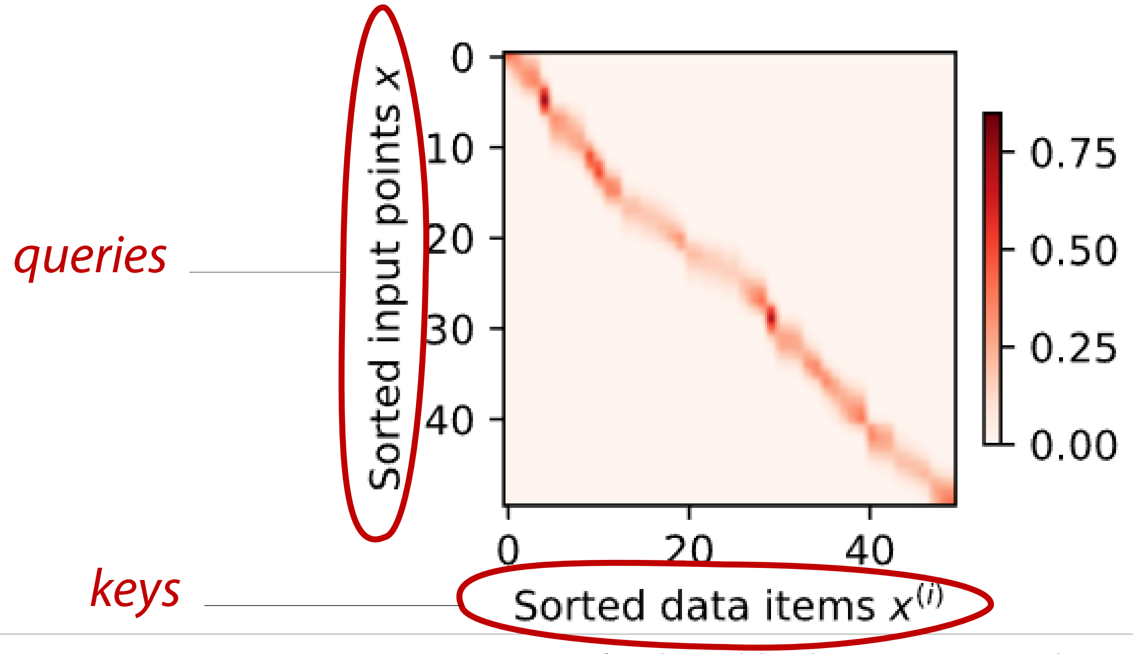
[image from http://d2l.ai/chapter_attention-mechanisms/]

- ***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/]

Attention: Queries, Key and Values

Attention Pooling: Queries, Keys and Values

- ***Generalized model***

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

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

In such Attention Pooling:

- *queries* and *keys* could come from different spaces
- the *attention map* α is <u>normalized</u>: 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***

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

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

The *attention map* is defined as:

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

where a is the *attention scoring function* of choice

[image from http://d2l.ai/chapter_attention-mechanisms/]

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:

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

  In the line of principle, $\boldsymbol{q}$ and $\boldsymbol{k}$ 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

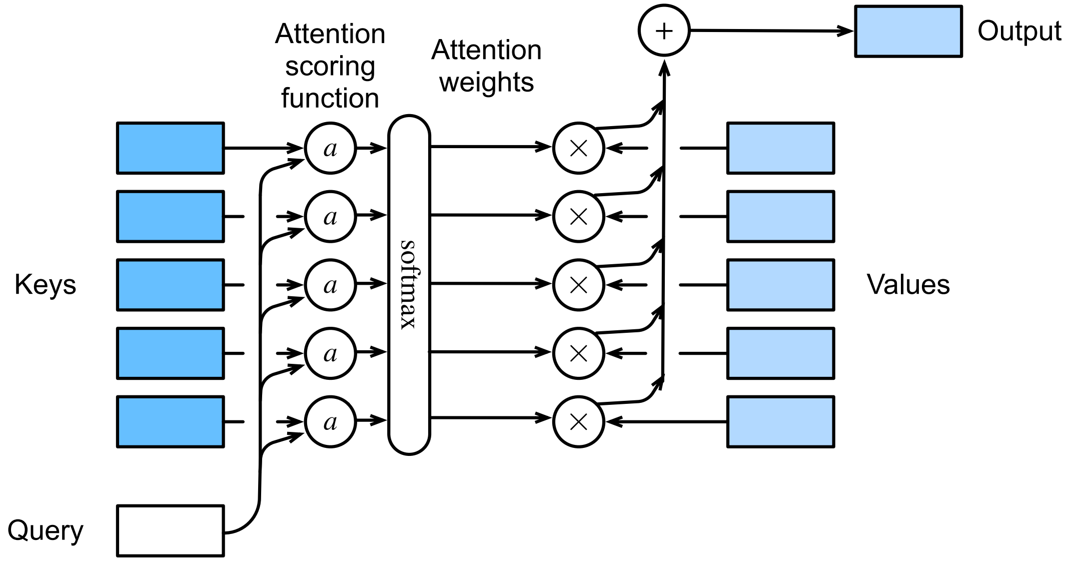
- ***Scaled Dot-Product Attention***

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

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

Attention Pooling becomes:

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



 [image from http://d2l.ai/chapter_attention-mechanisms/]

Attention Map

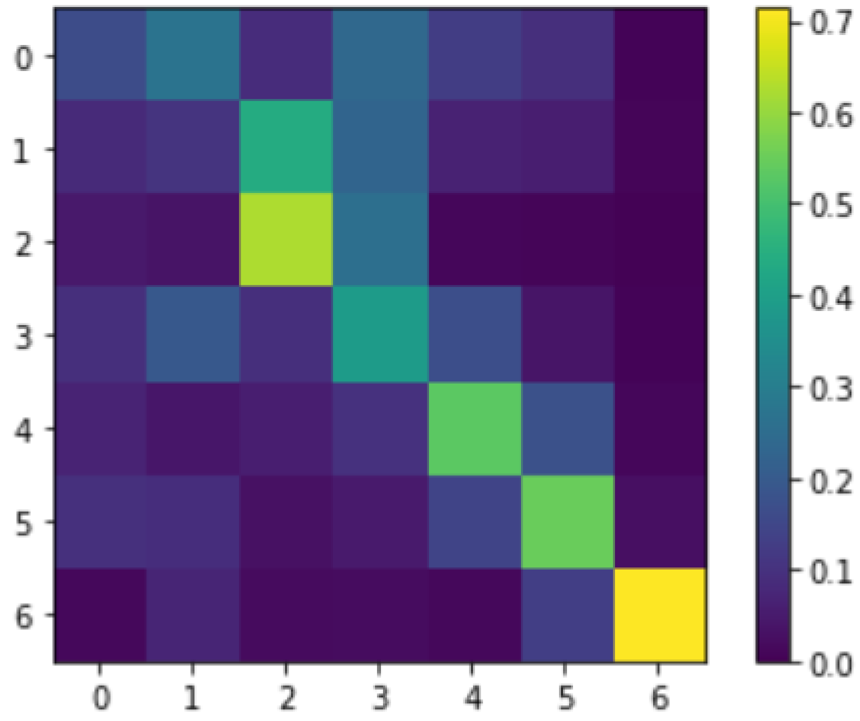
- ***Example: a square matrix***

When both *queries* and *keys* come from the same source (i.e., *self-attention*)
Namely:

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

Then:

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



Attention Scoring Function

- ***Scaled Dot-Product Attention***

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

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

Attention Pooling becomes:

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

- **Maximal Generality**

Provided the above shape constraints are respected, the above definition works fine

Queries and *keys* need <u>not</u> be in the same number *(i.e., the attention matrix needs not be square)*

Is it necessary to rely on tensor arrangements for <u>positional dependencies</u> among queries and keys?

- ***Using sine and cosine***

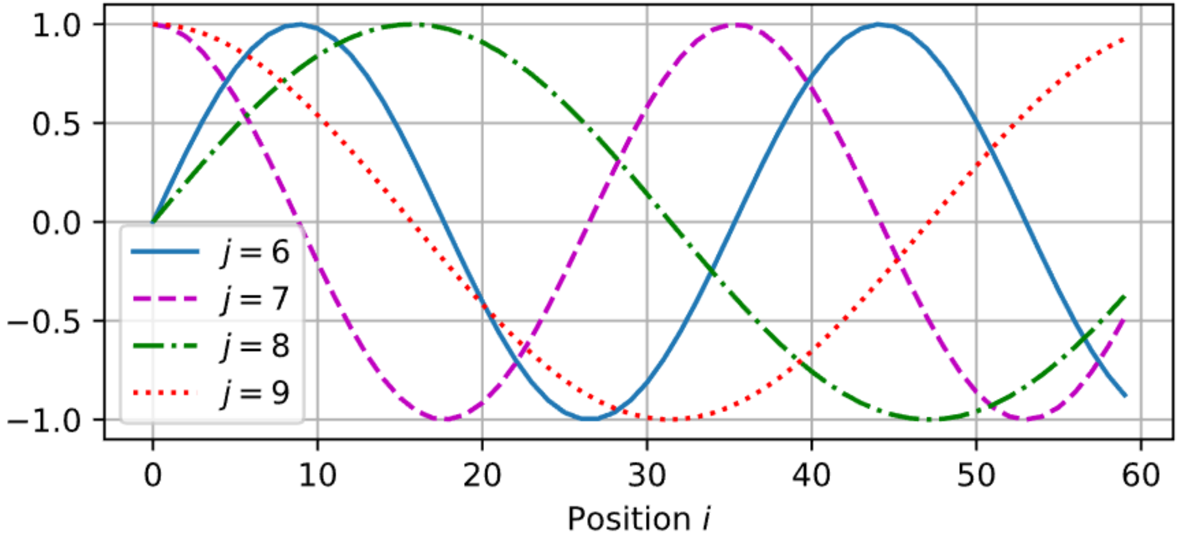
Assume we want to sum a *positional encoding* vector $\boldsymbol{p}$ to a data vector $\boldsymbol{x}$ ($\boldsymbol{x}, \boldsymbol{p} \in \mathbb{R}^d$)

$$\boldsymbol{x} + \boldsymbol{p}$$

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

Positional Encoding

- ***Using sine and cosine***

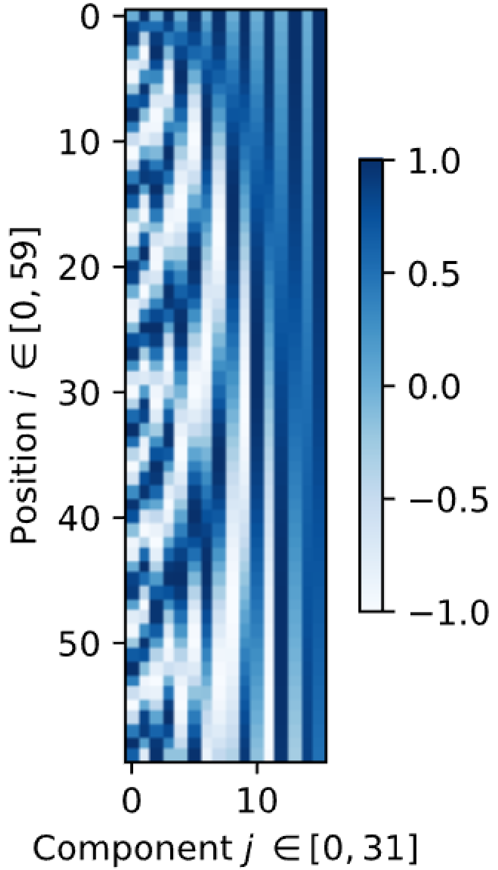
Assume we want to sum a *positional encoding* vector $\boldsymbol{p}$ to a data vector $\boldsymbol{x}$ ($\boldsymbol{x}, \boldsymbol{p} \in \mathbb{R}^d$)

$$\boldsymbol{x} + \boldsymbol{p}$$

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

Positional Encoding

- ***Relative displacements***

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

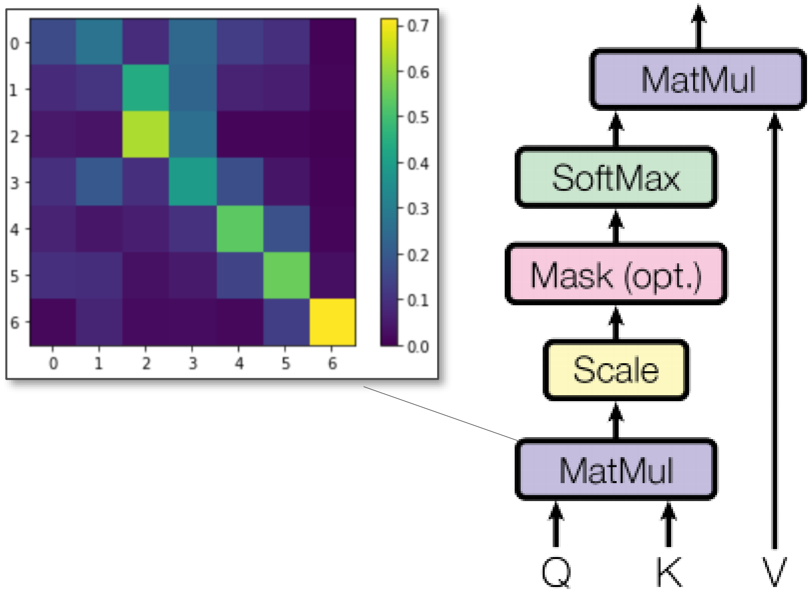
Displacements can be represented via a linear transformation.
This means that relative positions can be <u>learnt</u>

Transformer:
a network architecture

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

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

Scaled Dot-Product Attention

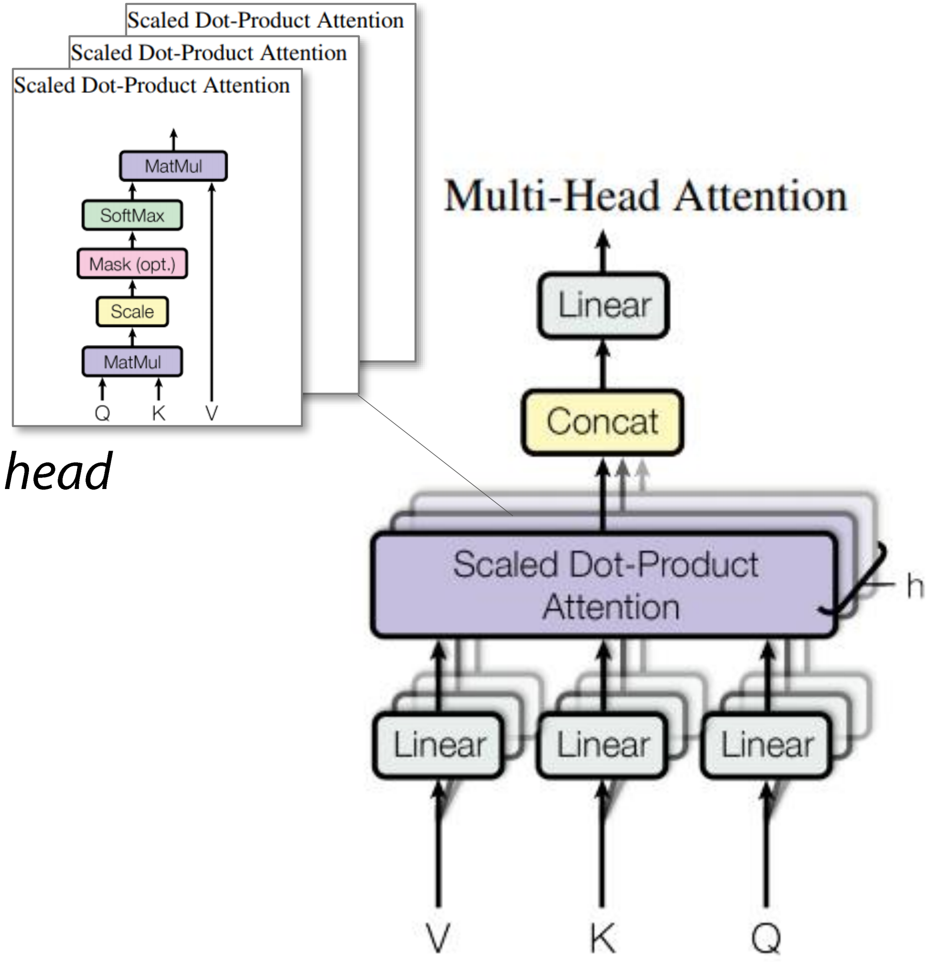


This is the basic building block

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

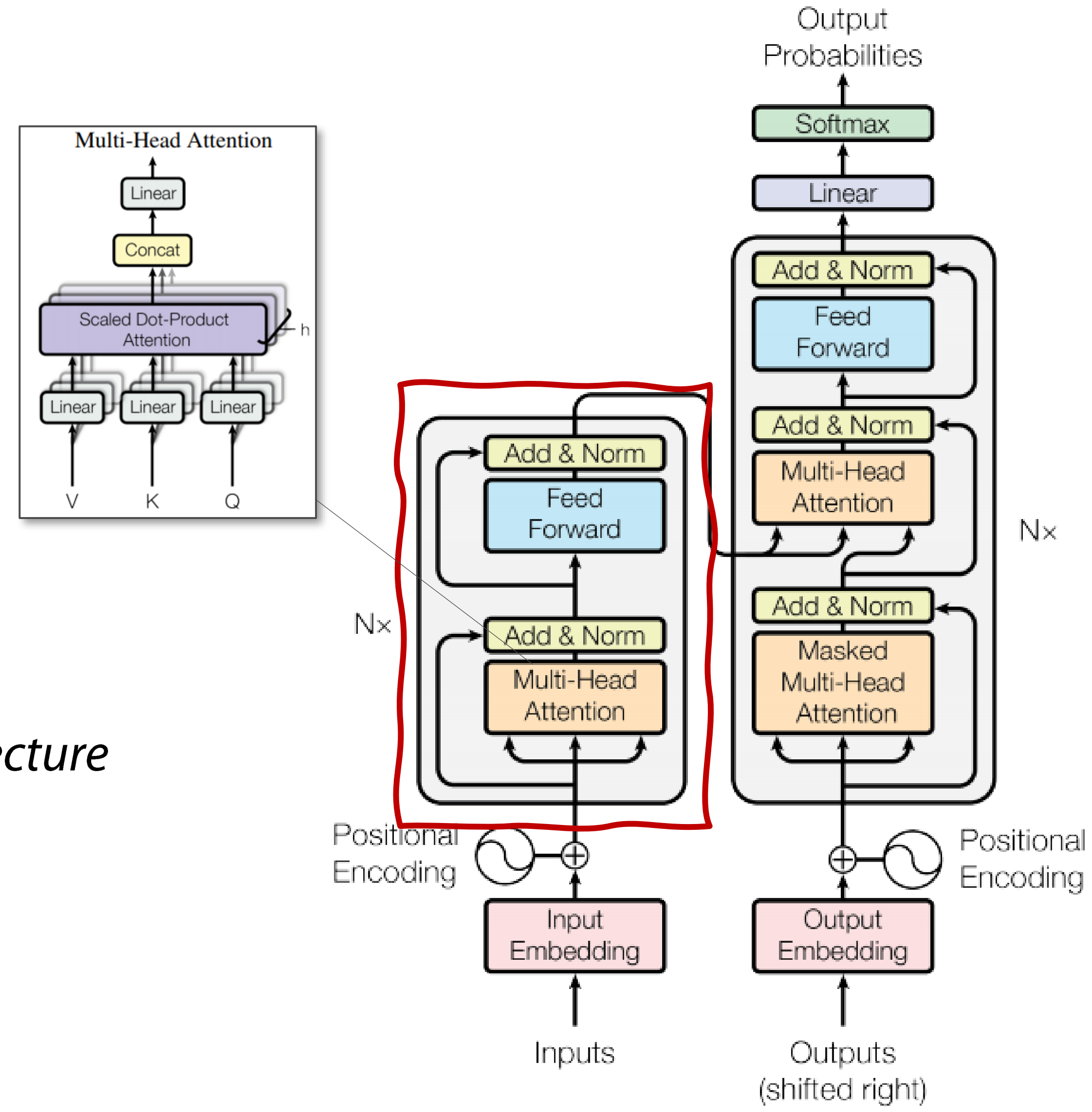
[image from https://arxiv.org/pdf/1706.03762.pdf]

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



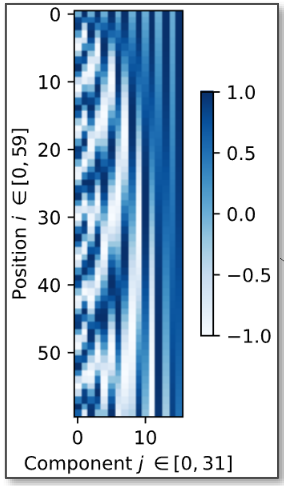
[image from https://arxiv.org/pdf/1706.03762.pdf]

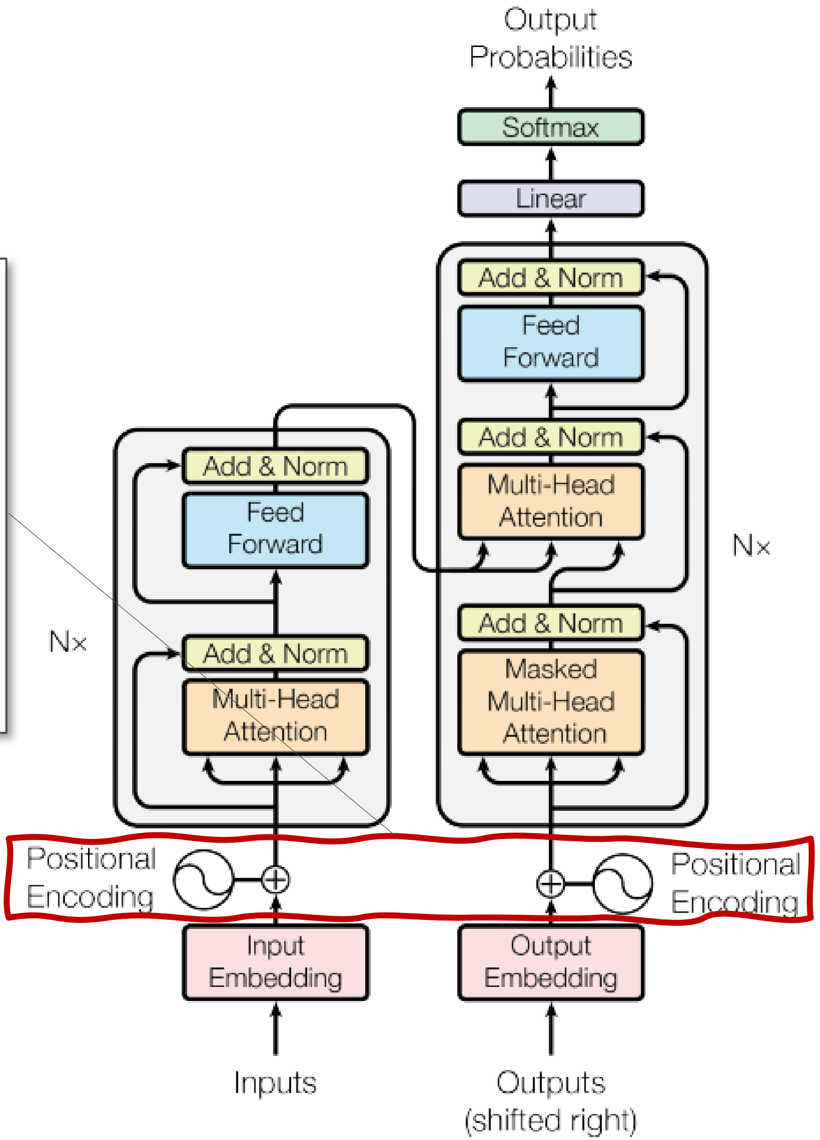
Encoder Layer

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





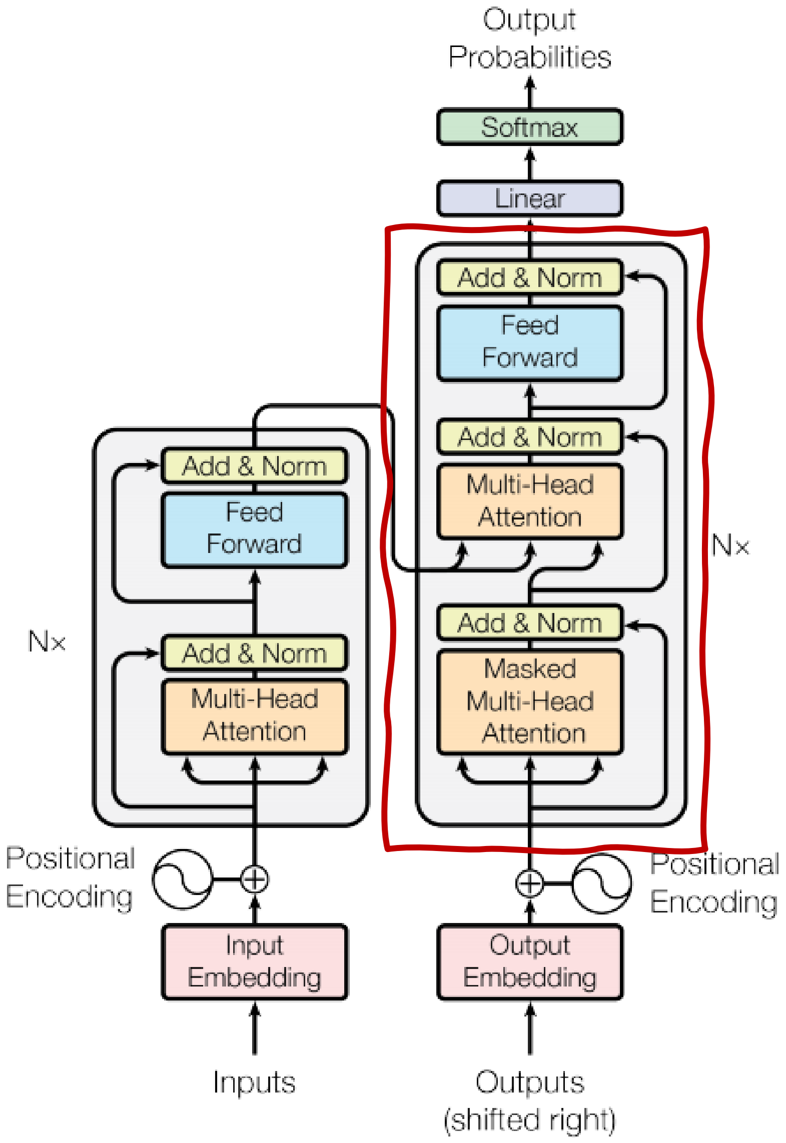
[image from https://arxiv.org/pdf/1706.03762.pdf]

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



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

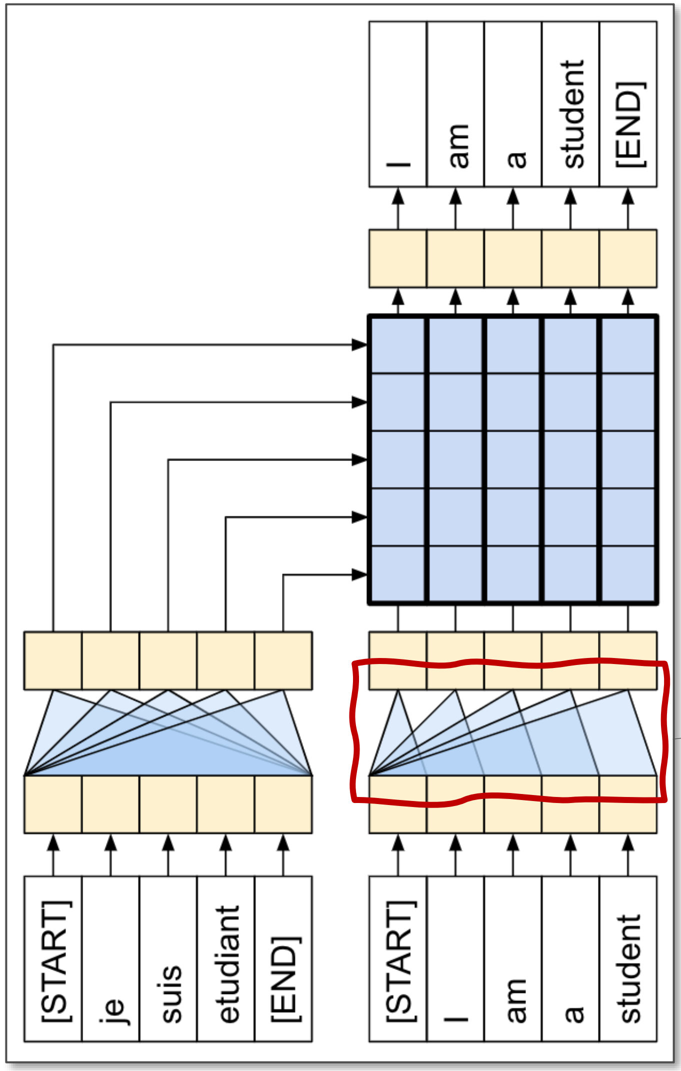
Decoder Layer

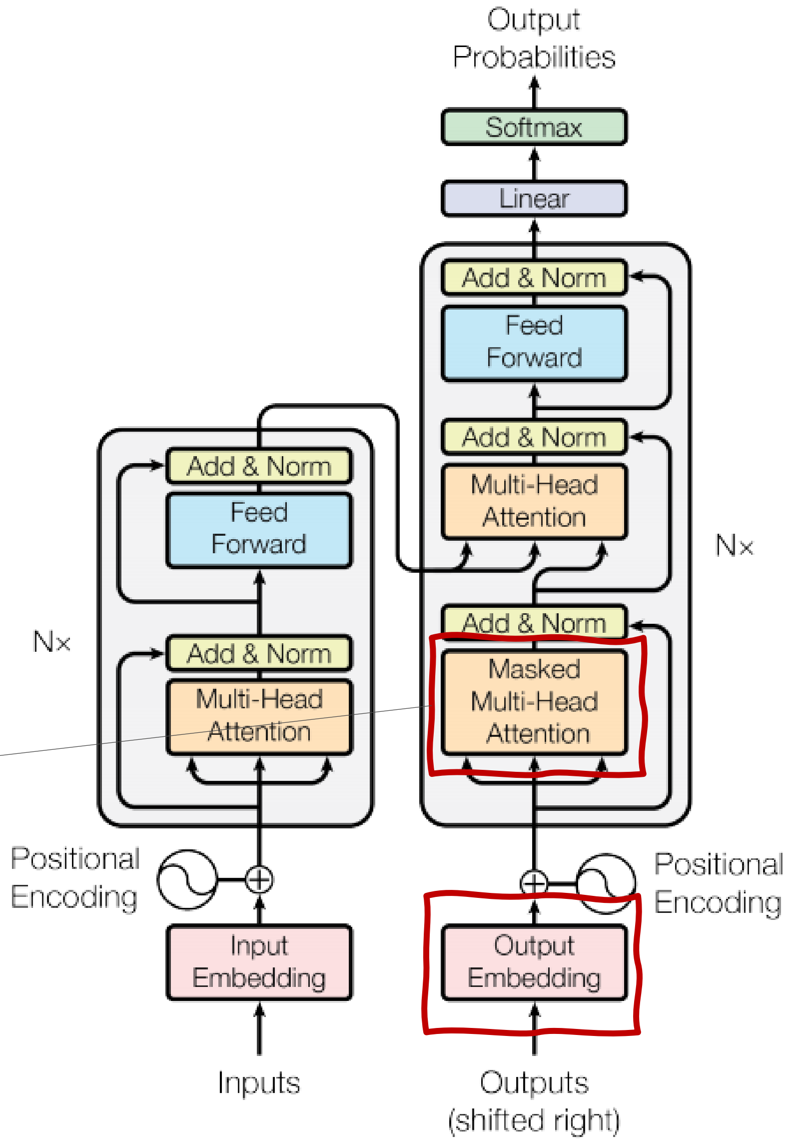
Why <u>masked</u> multi-head attention in the decoder layer?

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



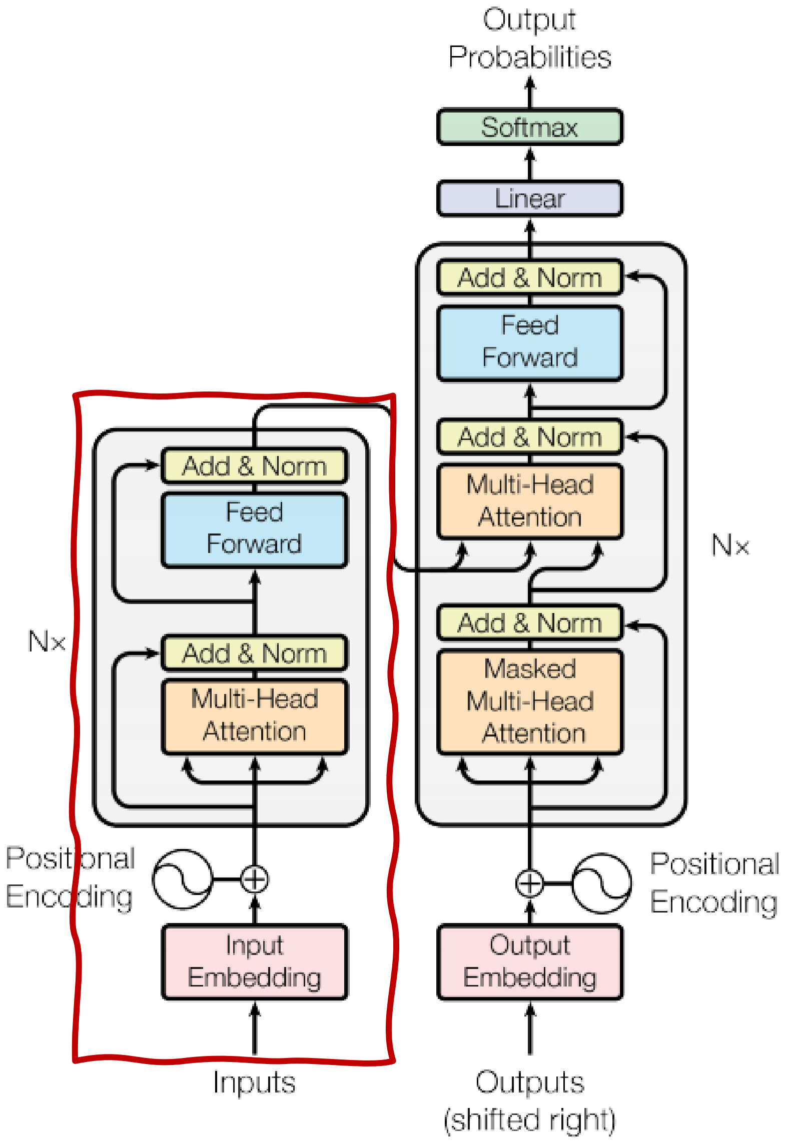


[image from https://arxiv.org/pdf/1706.03762.pdf]

Encoder

The encoder block includes:

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

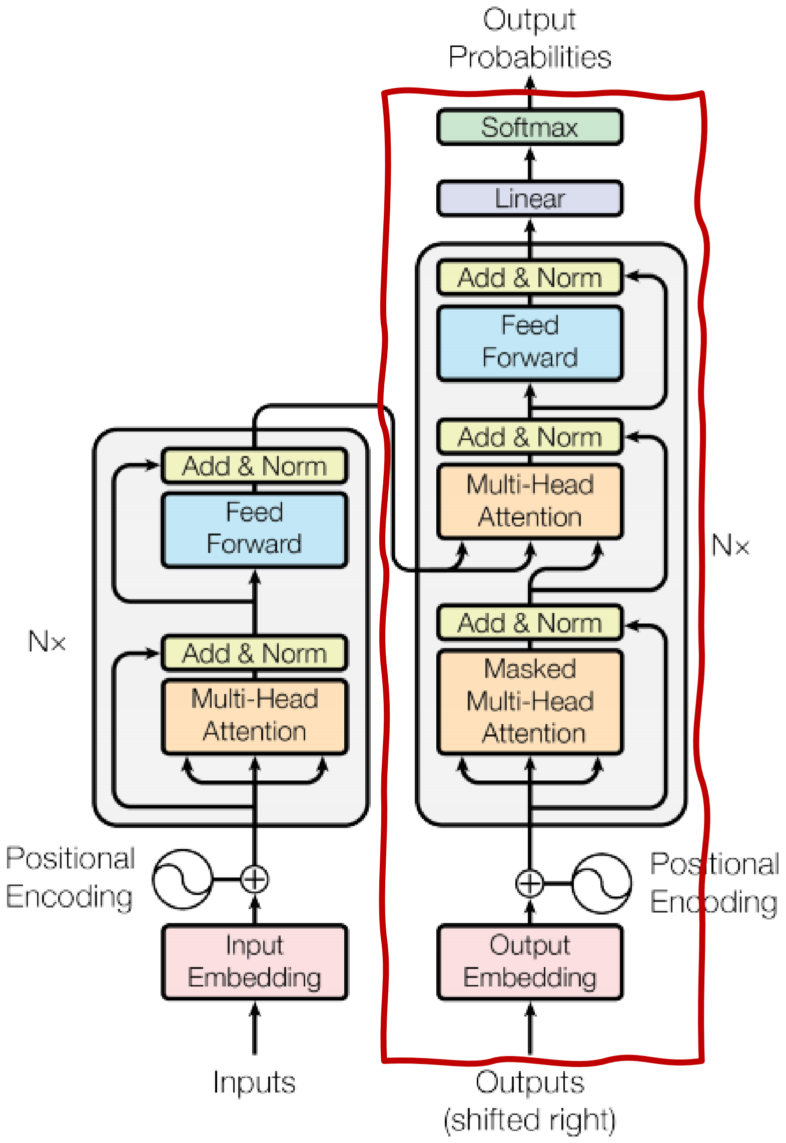


[image from https://arxiv.org/pdf/1706.03762.pdf]

Decoder

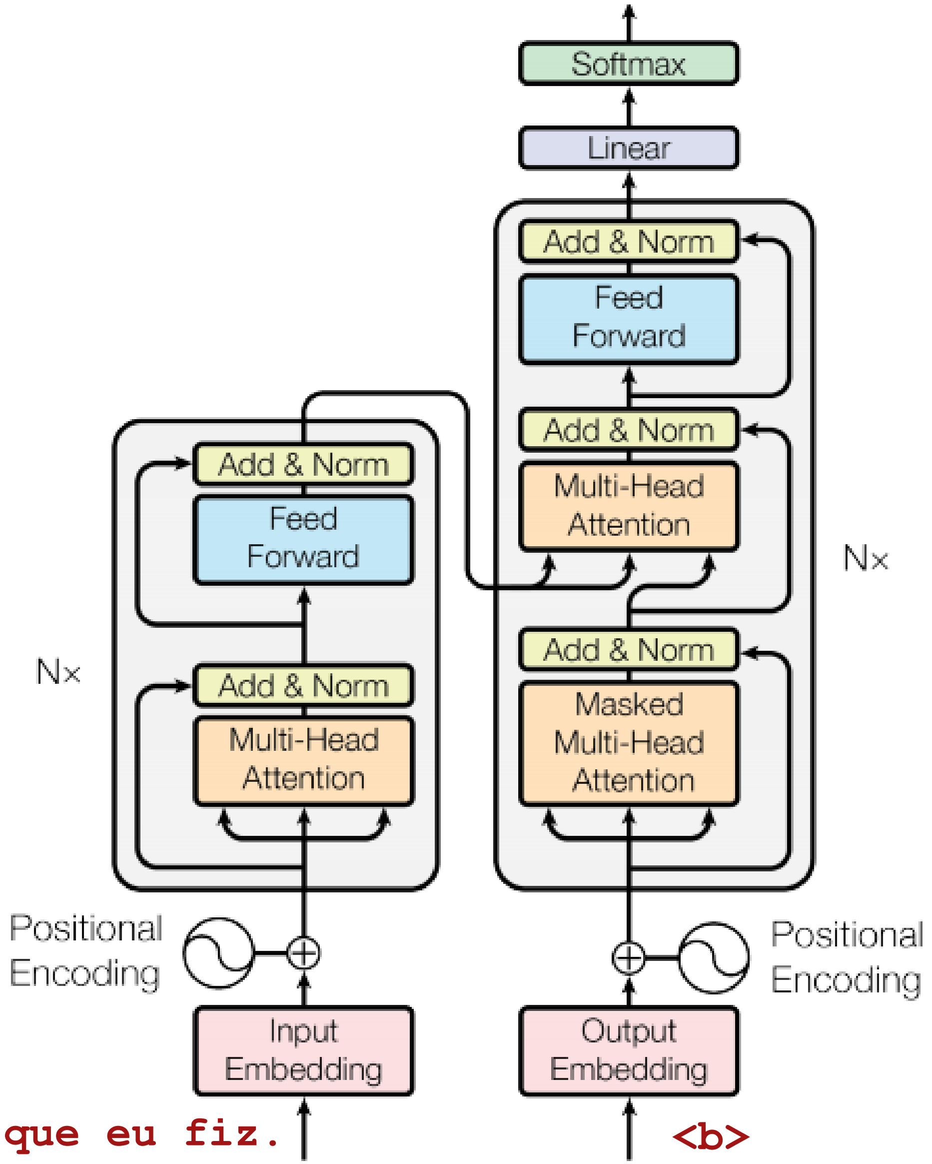
The decoder block includes:

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



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

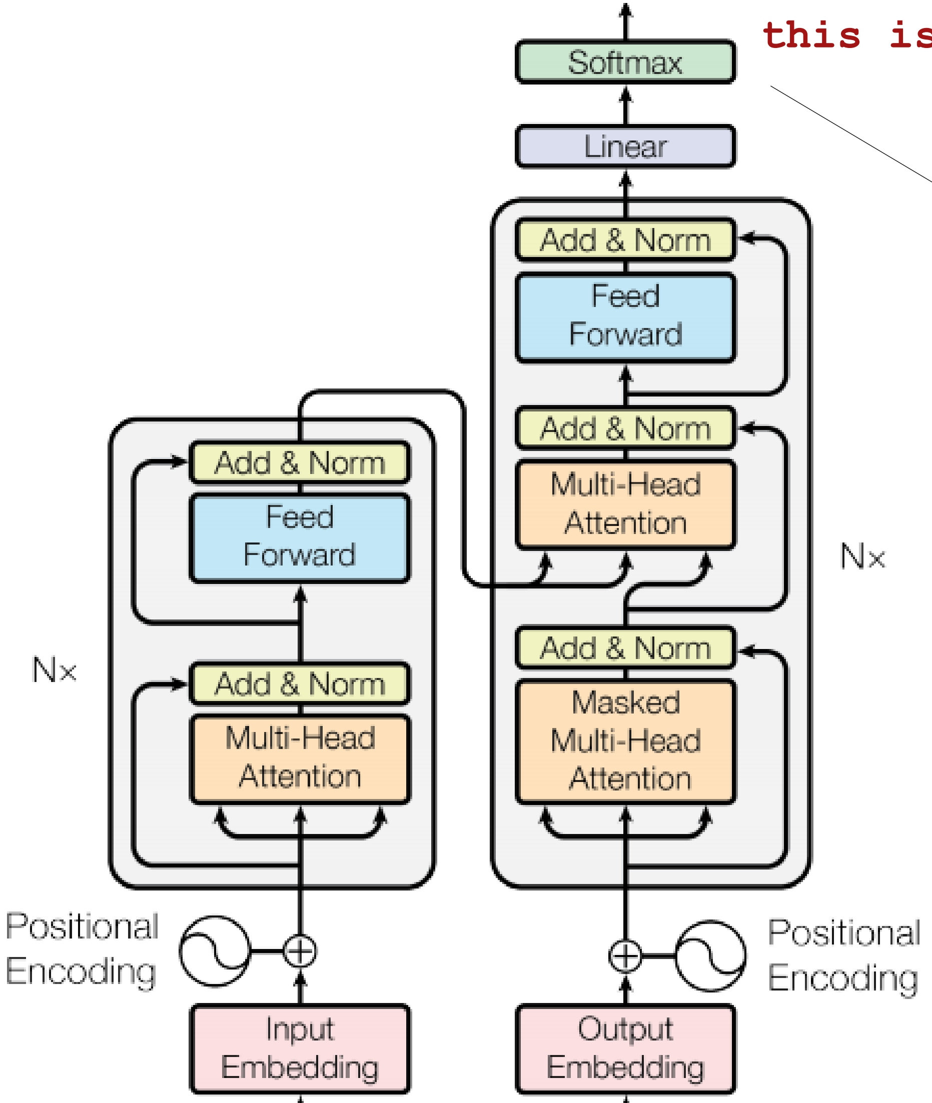
Translator



este é o primeiro livro que eu fiz. <b>

Translator

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)

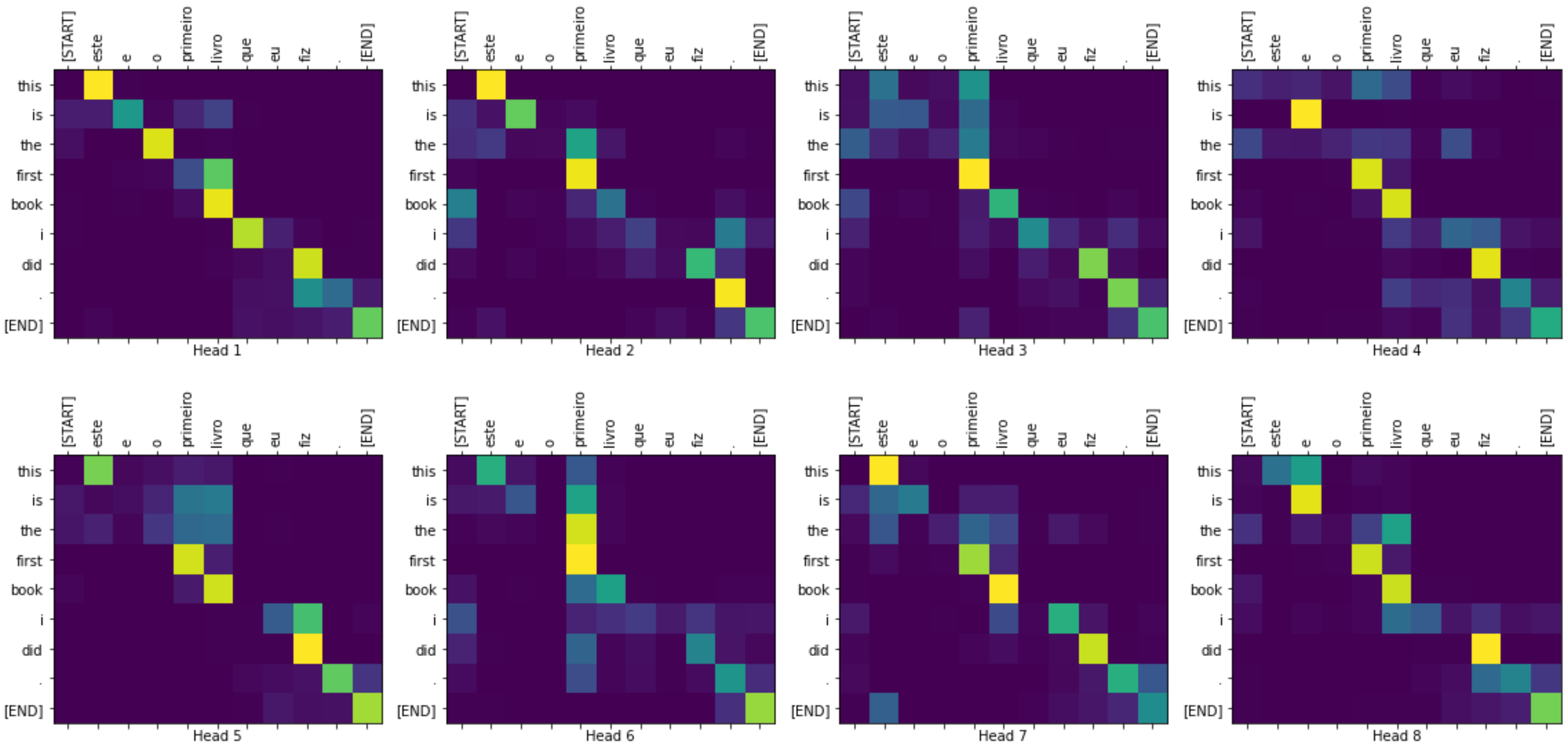
este é o primeiro livro que eu fiz.

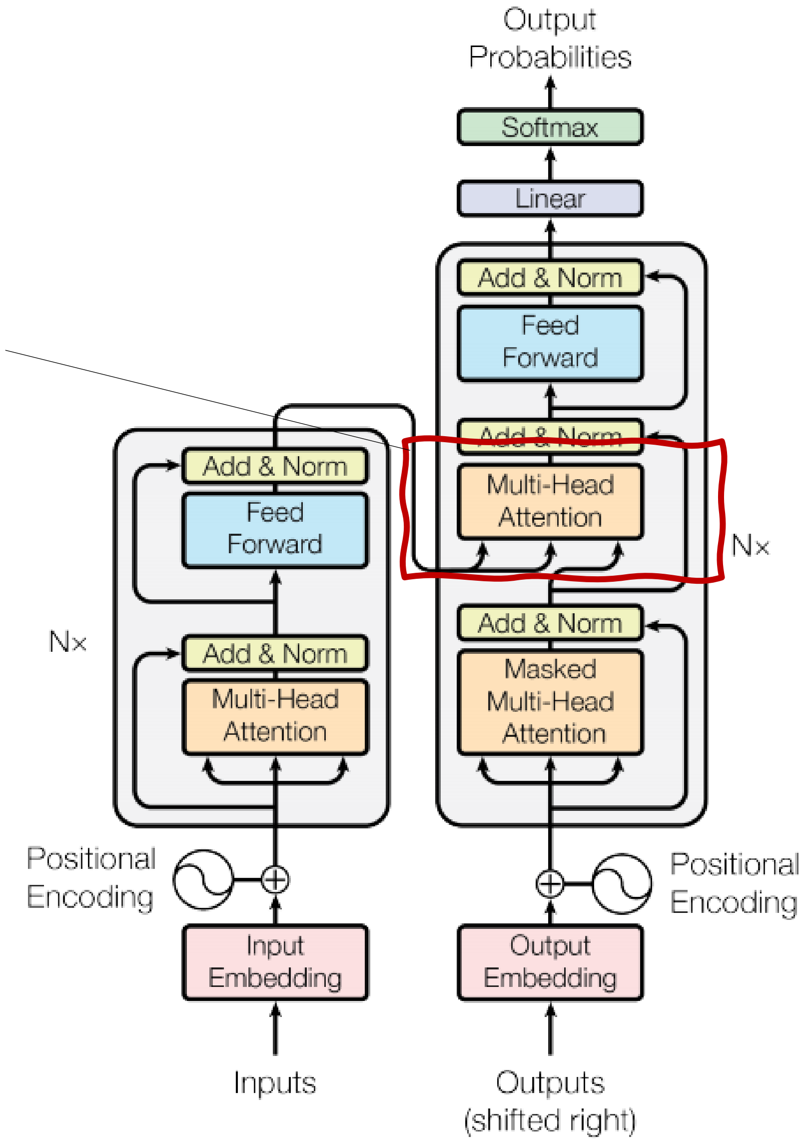
<b> this is the first book i've ever done.

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

Attention Maps

Multiple heads, mixing layer, topmost decoder block





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

Google Colab

https://www.tensorflow.org/text/tutorials/transformer