



Università degli
Studi di Pavia

Deep Learning

09-Attention and Transformers

Marco Piastra

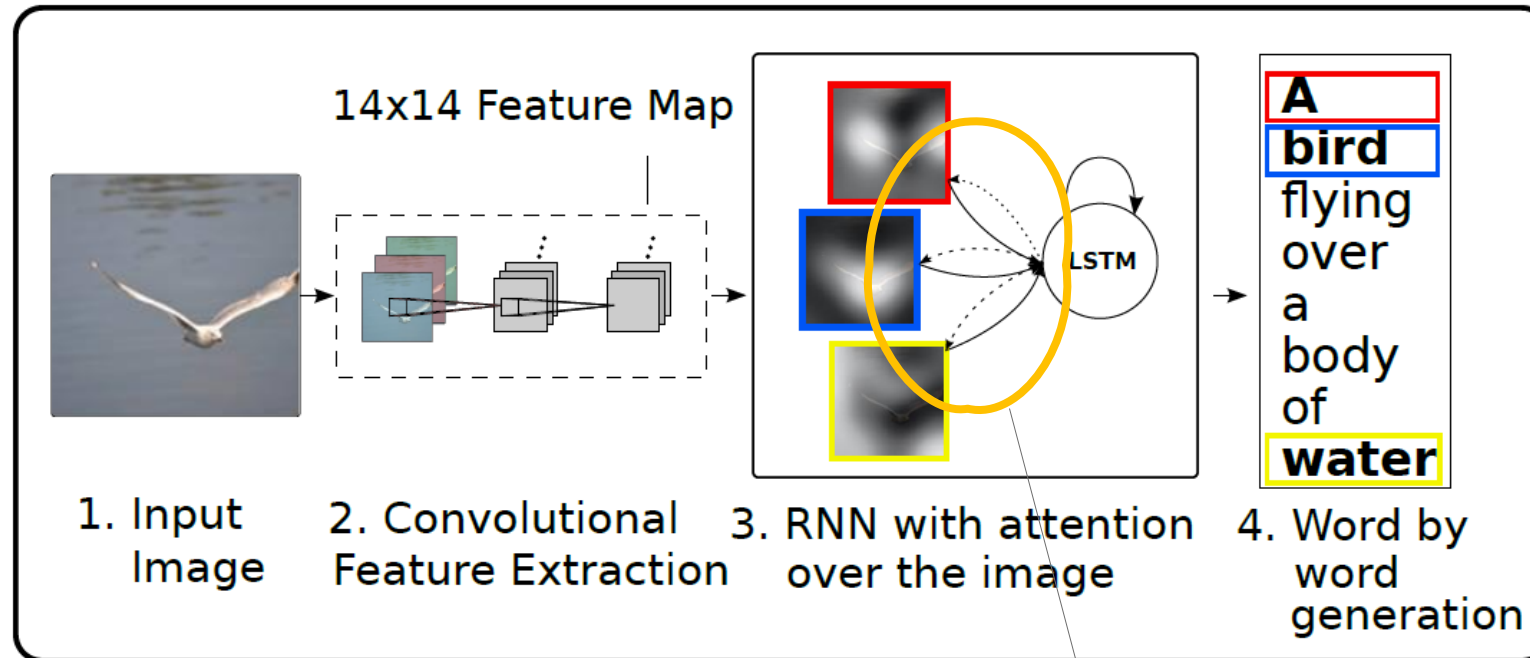
This presentation can be downloaded at:
<http://vision.unipv.it/DL>

*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]

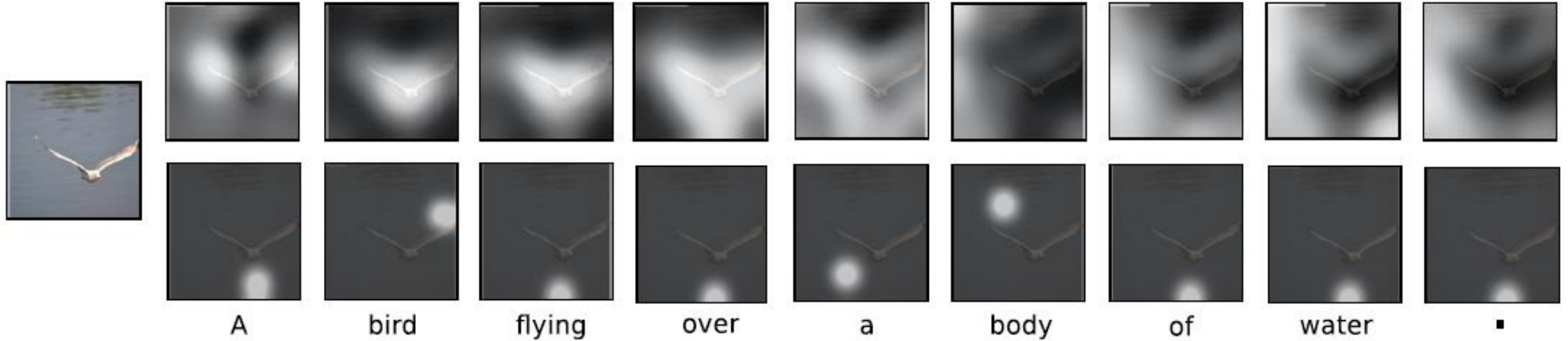


*The 'trick' is here:
when generating each word
the LSTM focuses on a specific
region in the image*

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.



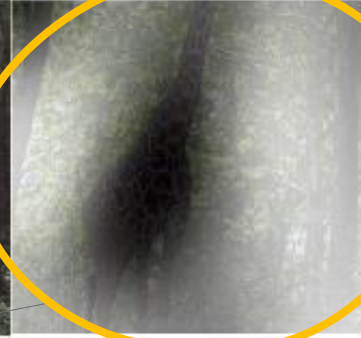
A stop sign is on a road with a mountain in the background.



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]

The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .

Current word being read

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

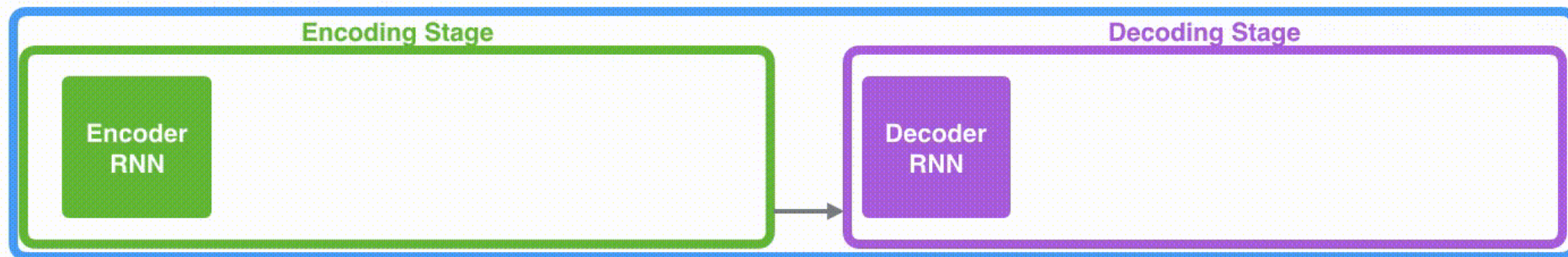
Attention with RNNs?

- **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

Neural Machine Translation SEQUENCE TO SEQUENCE MODEL

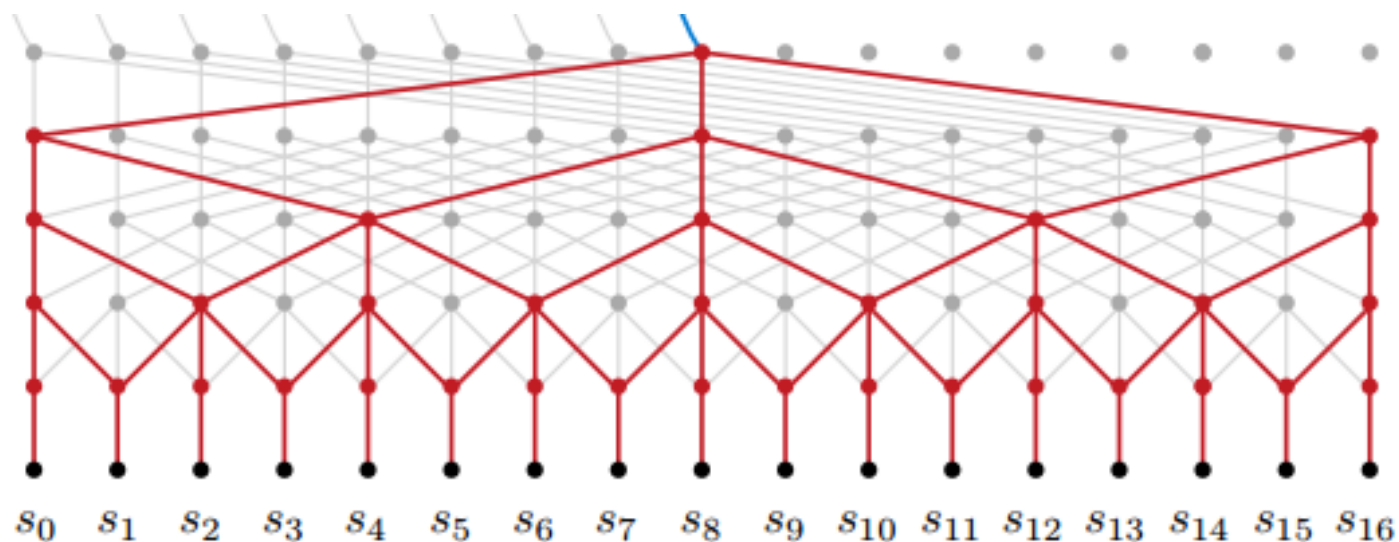


*Problem: the farther away is the focus of attention,
the more difficult it gets for the system to keep track of it*

Attention with Deep Convolution?

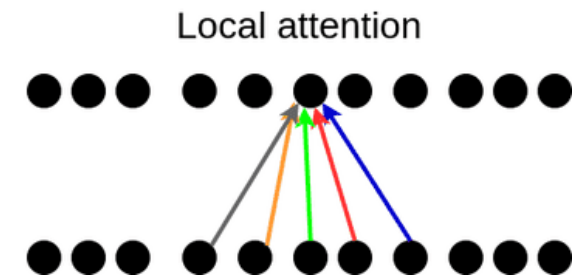
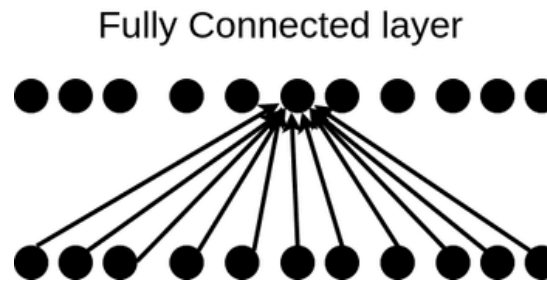
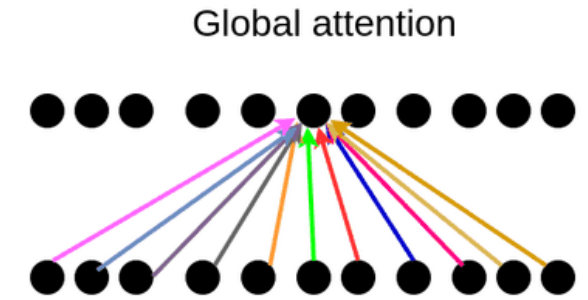
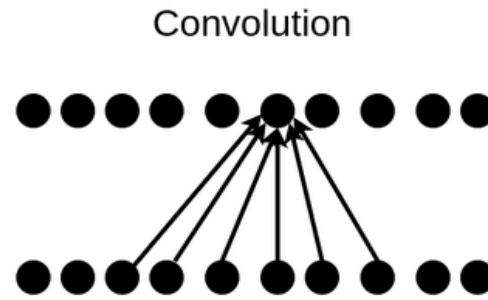
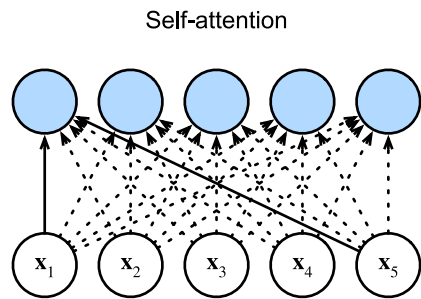
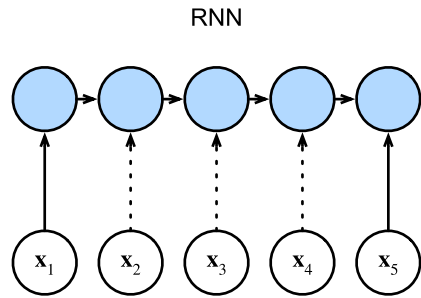
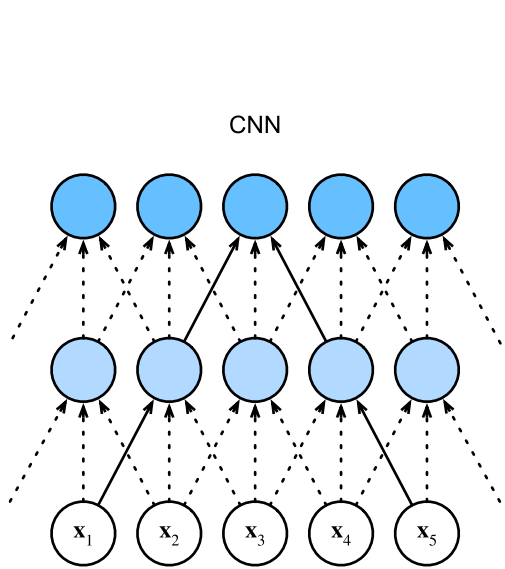
- **Progressively widening receptive field**

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 vs Convolution vs RNN



Attention as a Kernel

Attention as Kernel

■ Attention Pooling

Consider an input-output relation and a dataset

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

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

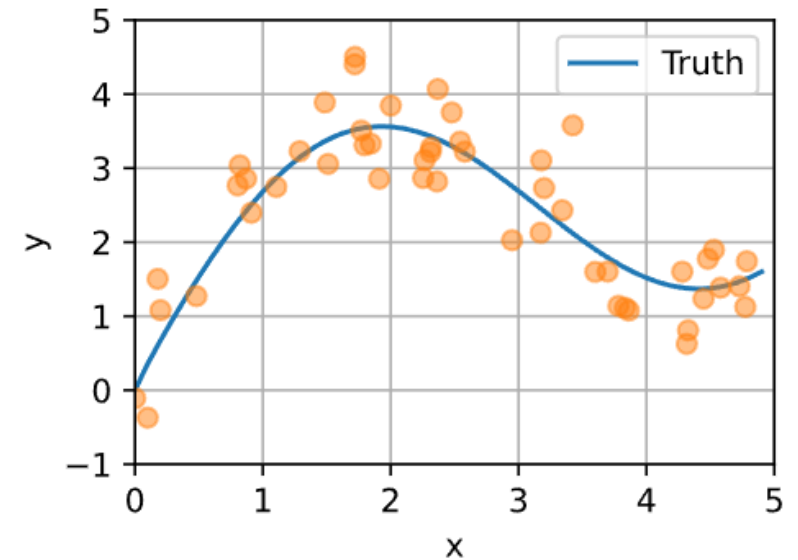
$$\tilde{\mathbf{y}} := \sum_{i=1}^N \alpha(\mathbf{x}, \mathbf{x}^{(i)}) \mathbf{y}^{(i)},$$

■ Example:

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

$$D := \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^N \quad \text{Dataset is noisy}$$

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



Attention as Kernel

■ Attention Pooling

Consider an input-output relation

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

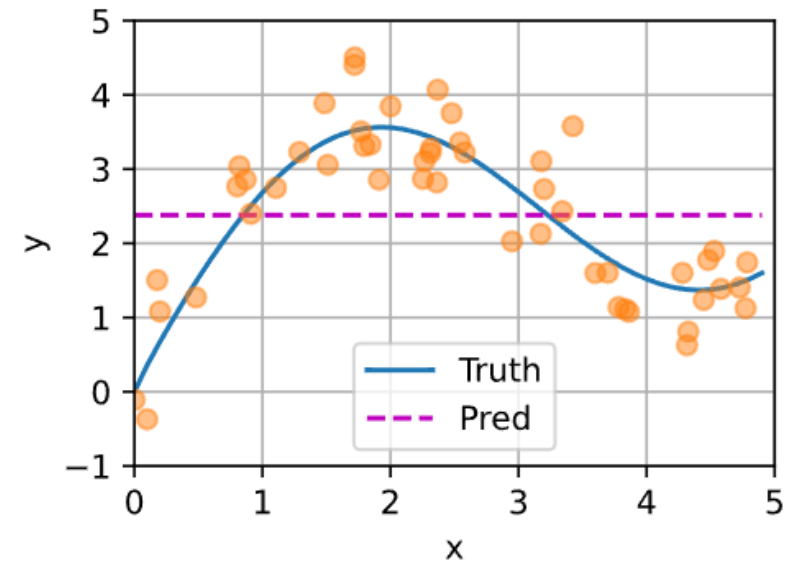
Attention Pooling is defined as a function on each input component

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

■ Example:

Global average (i.e., no attention)

$$\alpha(x, x_i) = \frac{1}{d} \sum_{i=1}^d y_i$$



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

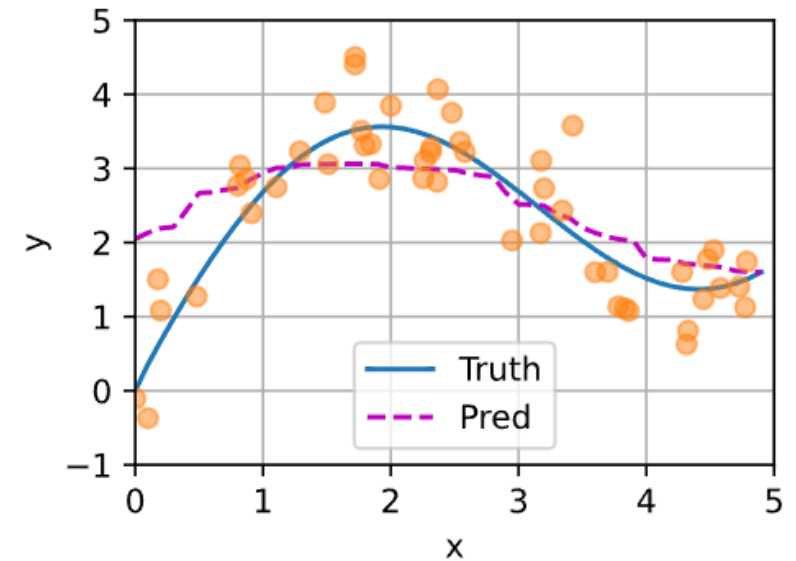
$$\tilde{\mathbf{y}} := \sum_{i=1}^N \alpha(\mathbf{x}, \mathbf{x}^{(i)}) \mathbf{y}^{(i)},$$

■ Example:

Gaussian Kernel [Nadaraya & Watson, 1964]

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

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

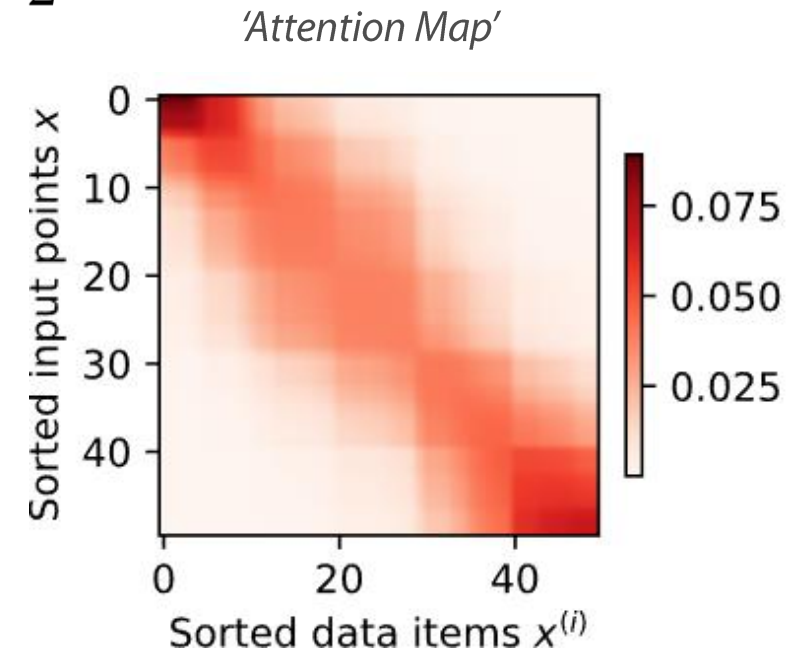


Attention as Kernel

■ Gaussian Kernel and softmax

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

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



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

Attention as Kernel

▪ (Simple) Parametric Attention Pooling

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

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*

Attention as Kernel

▪ (Simple) Parametric Attention Pooling

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

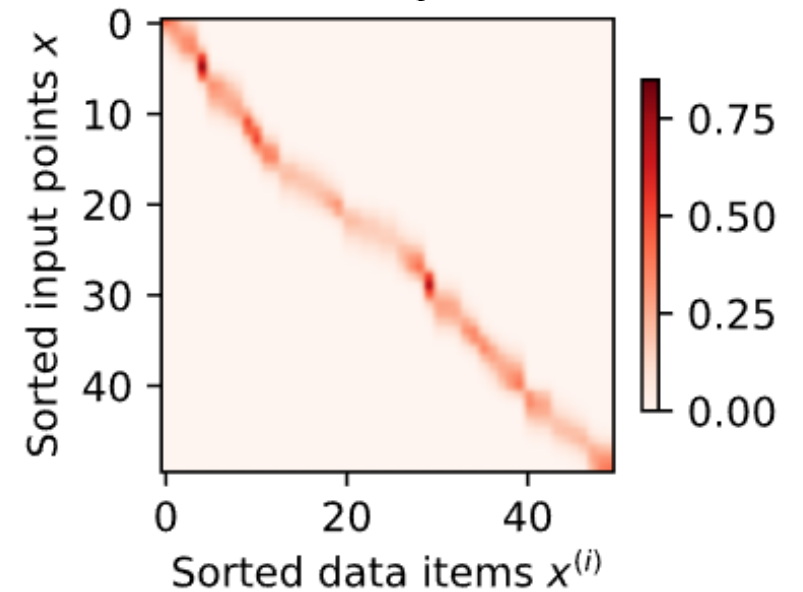
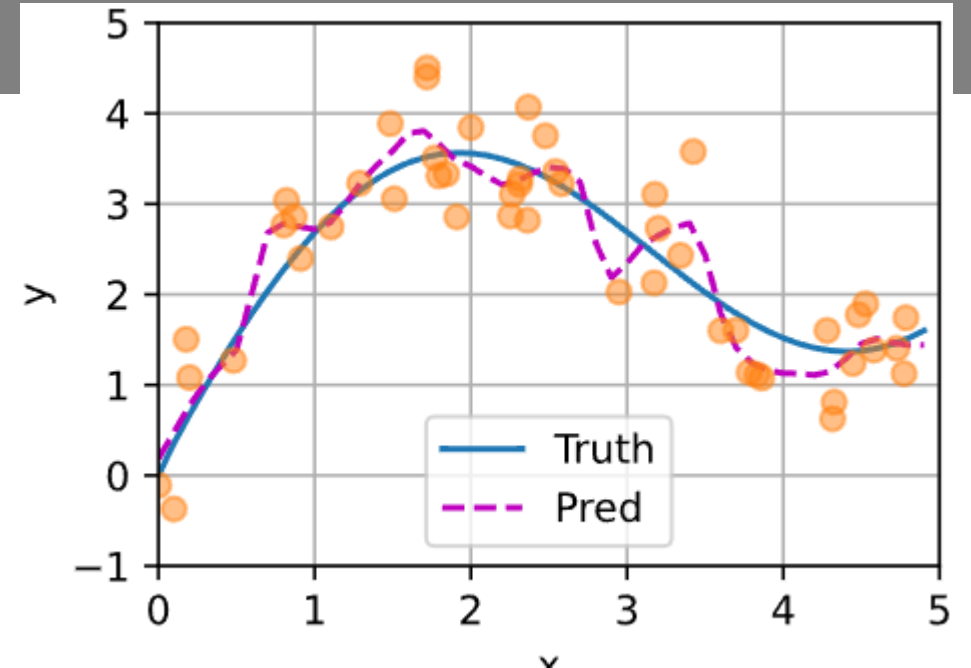
This requires training of the (unique) parameter w

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'



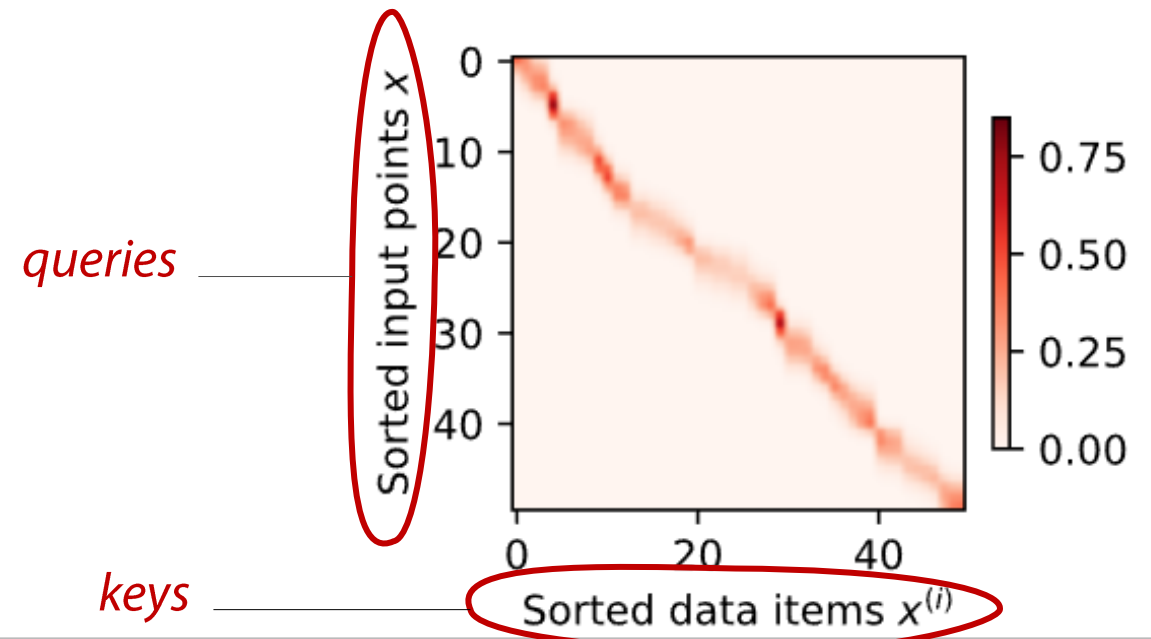
Attention as Kernel

■ 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



Attention: Queries, Key and Values

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 such Attention Pooling:

- *queries* and *keys* could come from different spaces
- the *attention map* α 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 map* 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

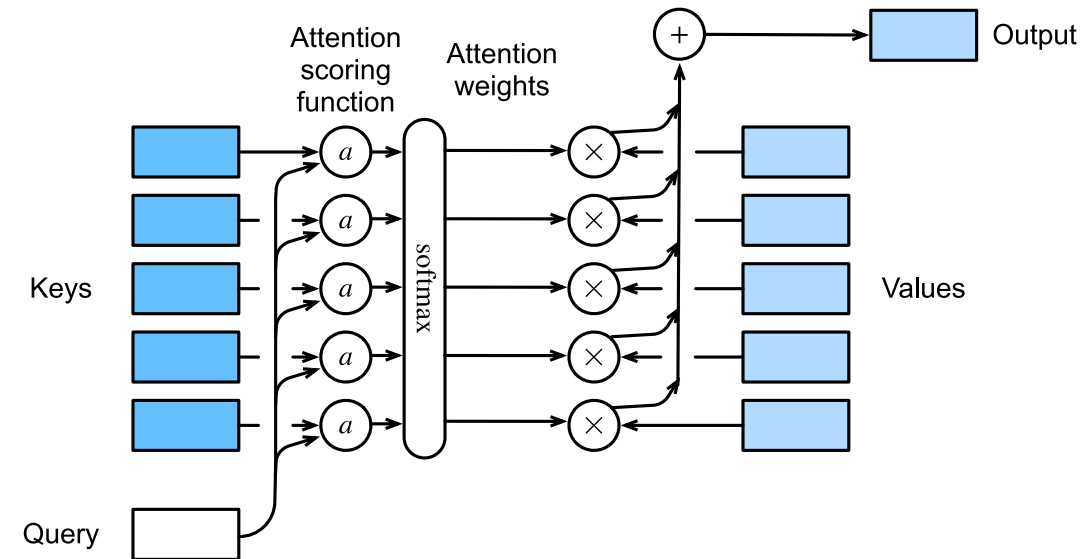
▪ Scaled Dot-Product Attention

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**

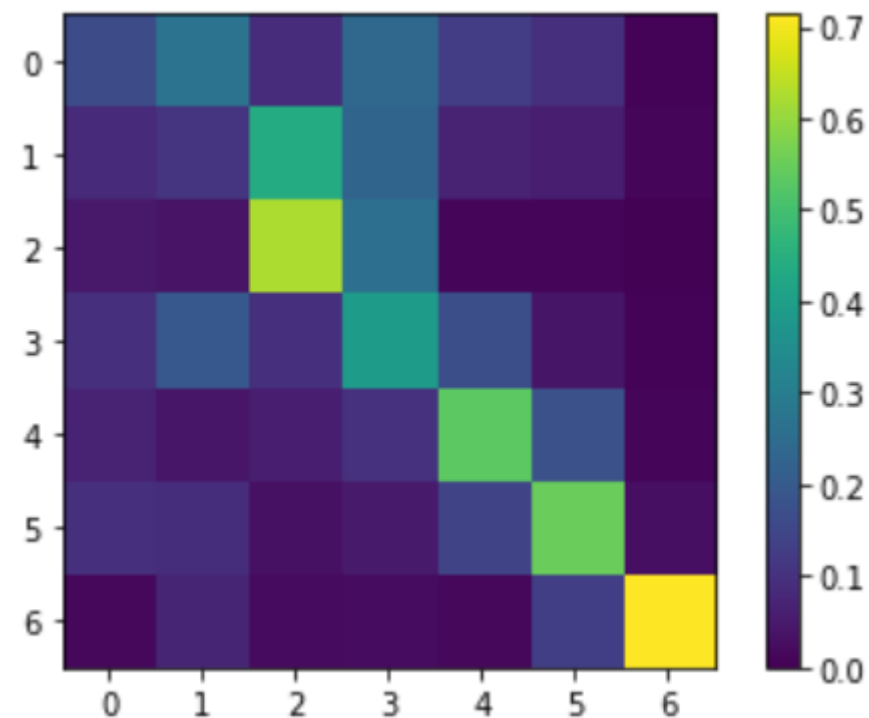
When both *queries* and *keys* come from the same source (e.g., the same *sequence*).

Namely:

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

Then:

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



Attention Scoring Function

▪ Scaled Dot-Product Attention

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

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

Attention Pooling becomes:

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

▪ Maximal Generality

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

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

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

Positional Encoding

■ Using sine and cosine

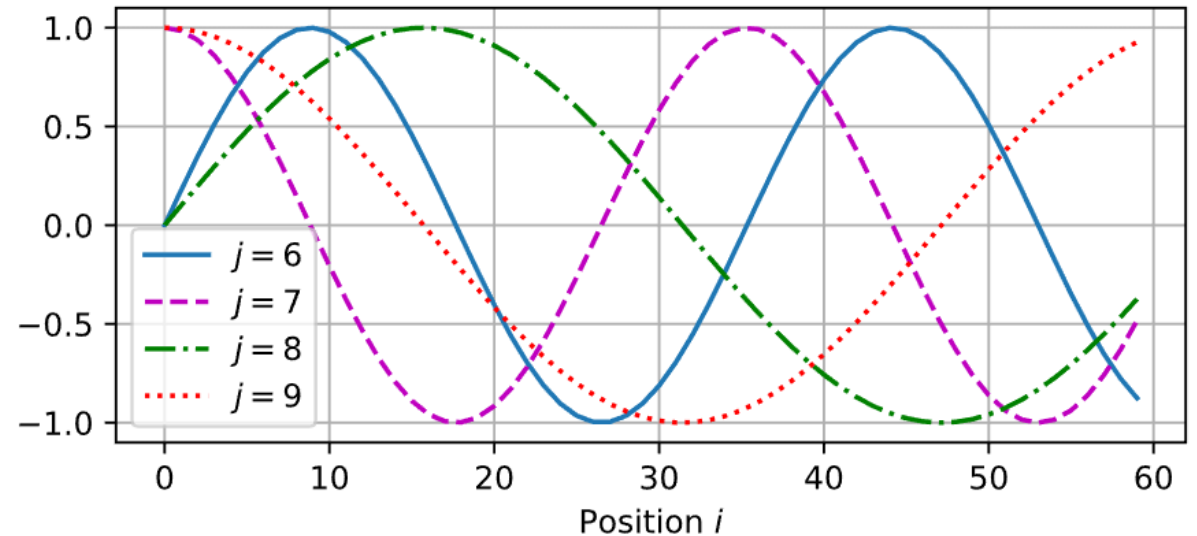
Assume we want to sum a *positional encoding* vector \mathbf{x} to a data vector \mathbf{p} ($\mathbf{x}, \mathbf{p} \in \mathbb{R}^d$)

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

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{x} to a data vector \mathbf{p} ($\mathbf{x}, \mathbf{p} \in \mathbb{R}^d$)

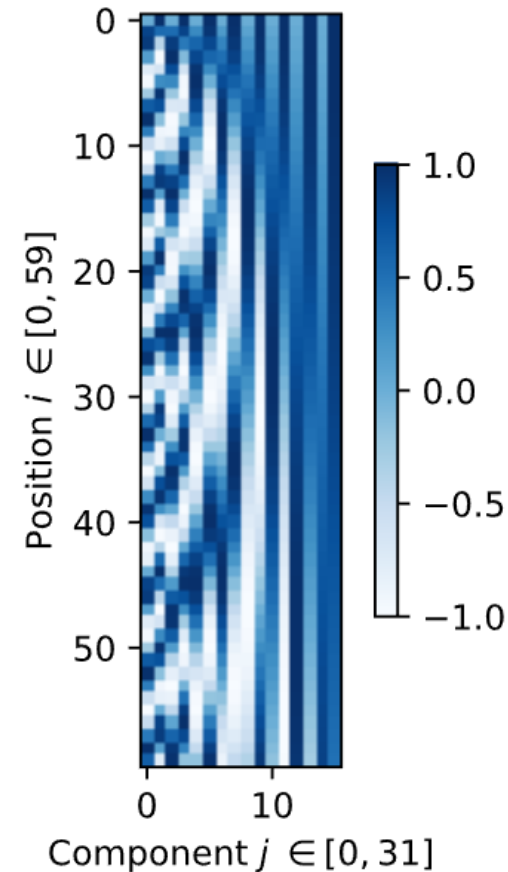
$$\mathbf{x} + \mathbf{p}$$

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



*Transformer:
a network architecture*

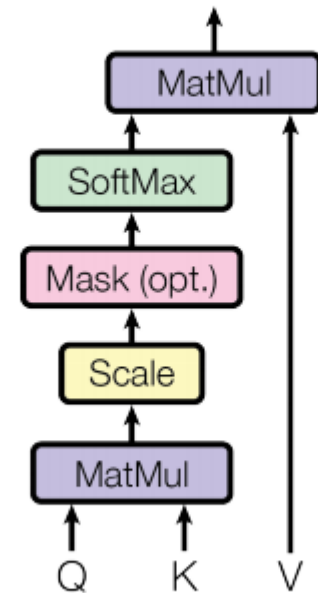
Scaled Dot-Product Attention

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

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

This is the basic building block

Scaled Dot-Product Attention



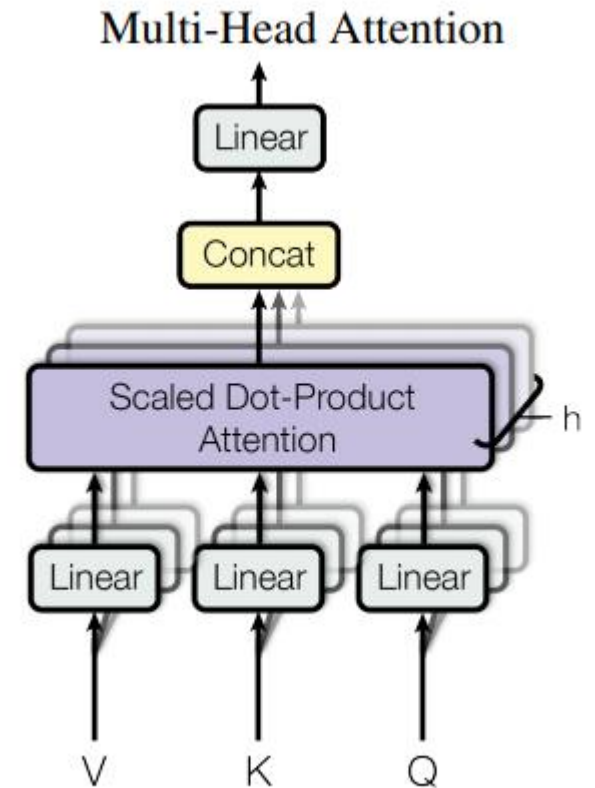
Multiple Attention Heads

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

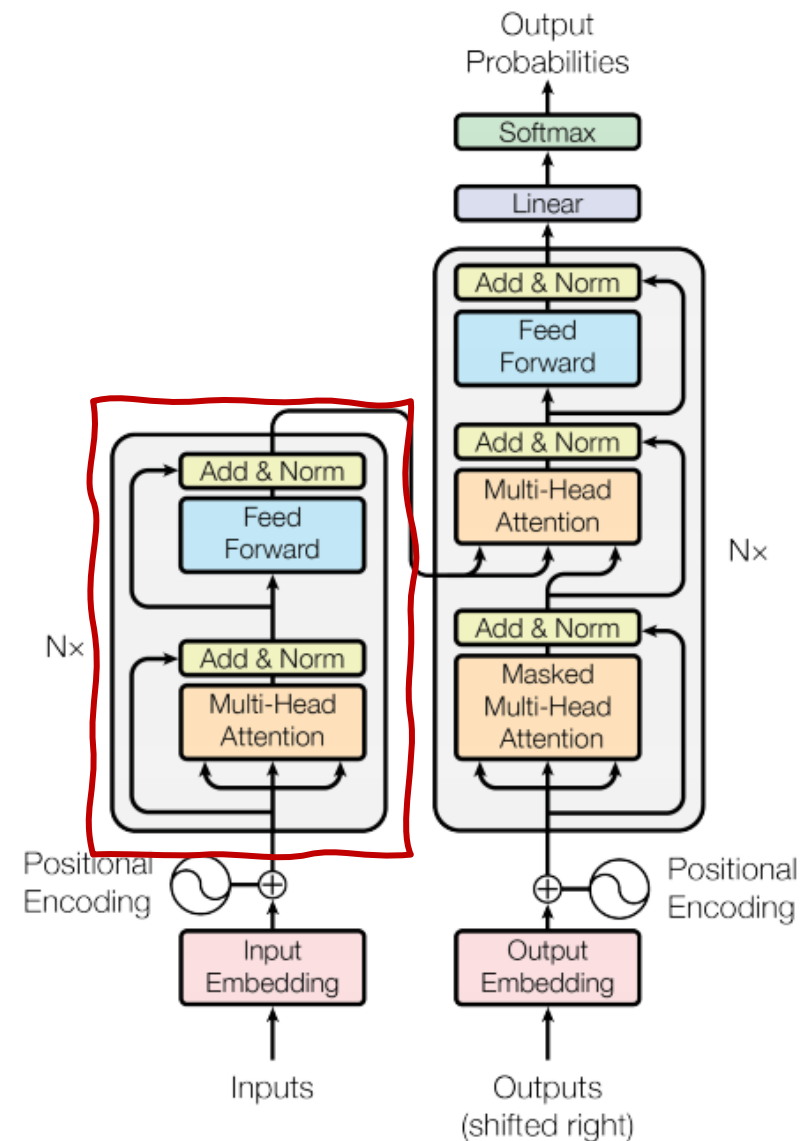


Encoder Layer

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

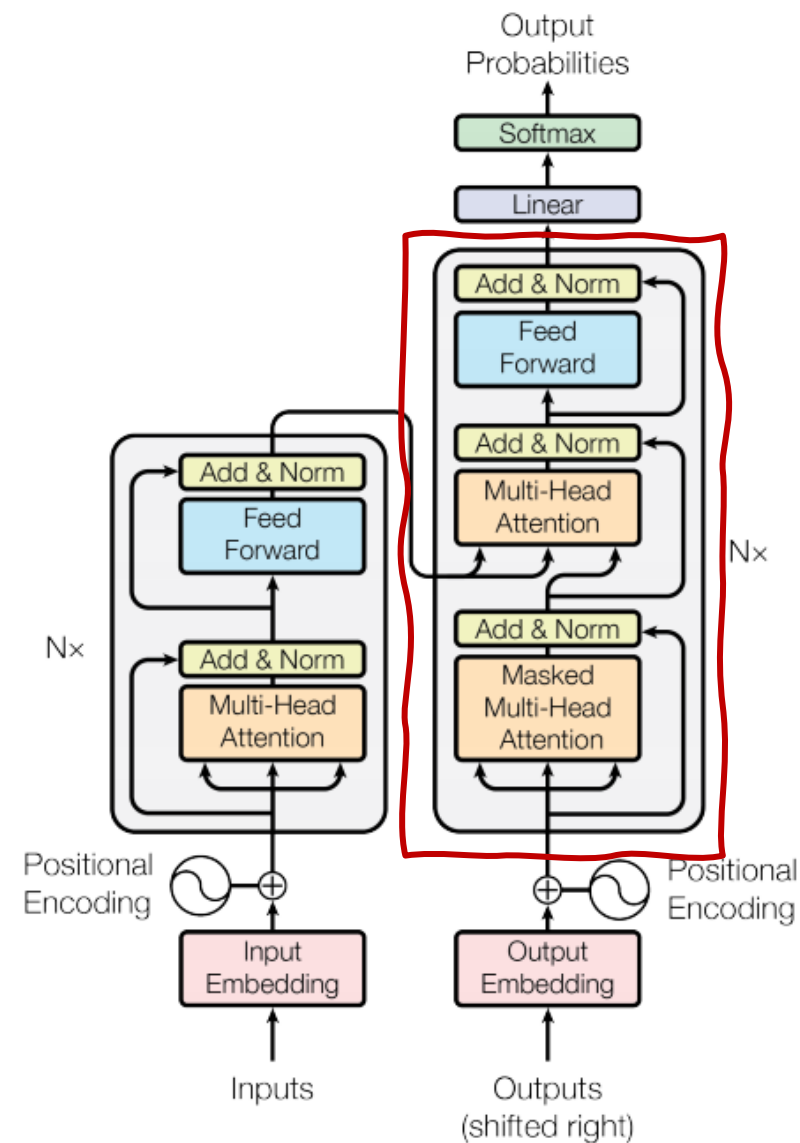


Decoder Layer

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

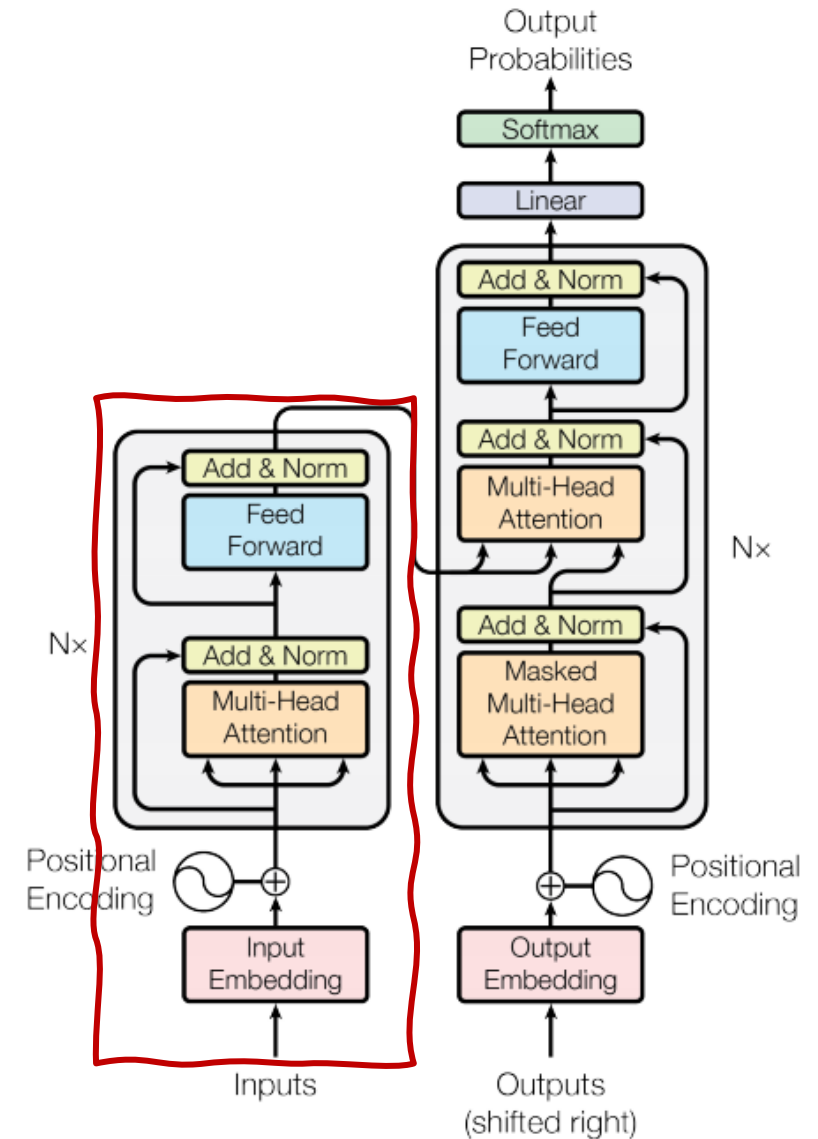
There could be many decoder layers in the overall architecture



Encoder

The encoder block includes:

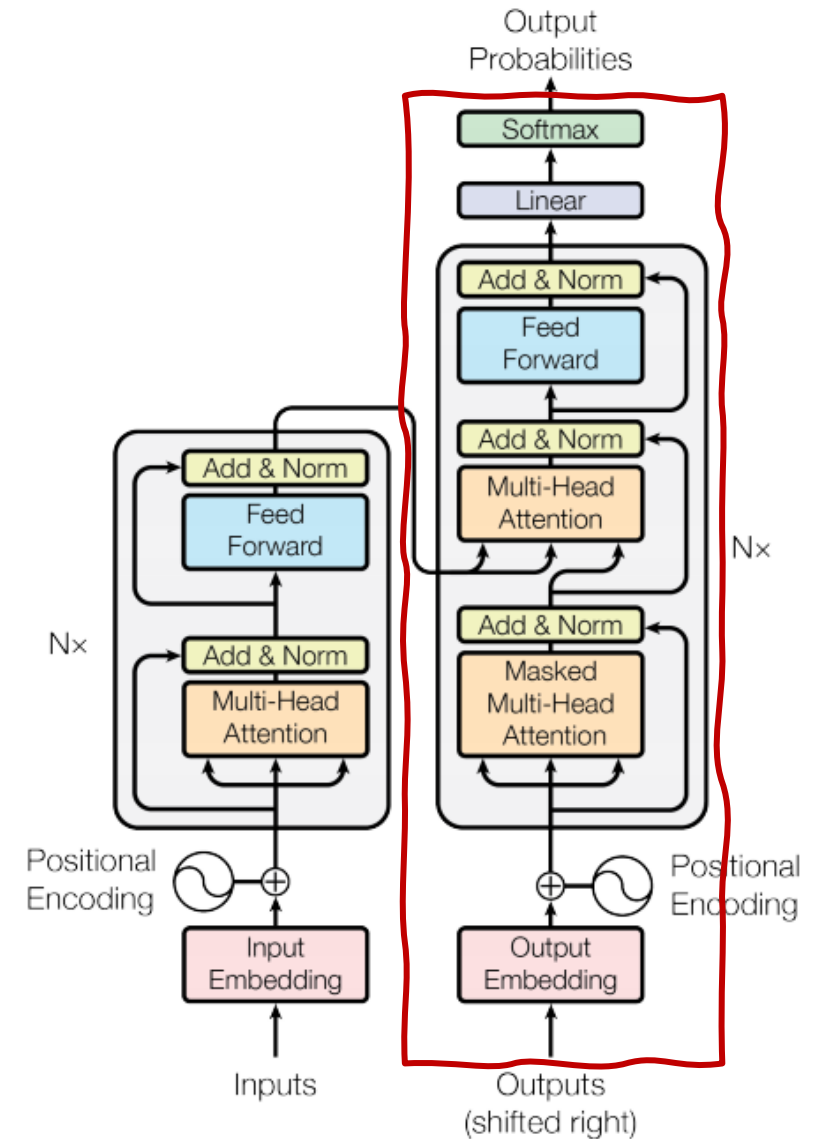
1. Input embedding (*word2vec* style)
2. Positional encoding
3. Addition
4. N encoder layers



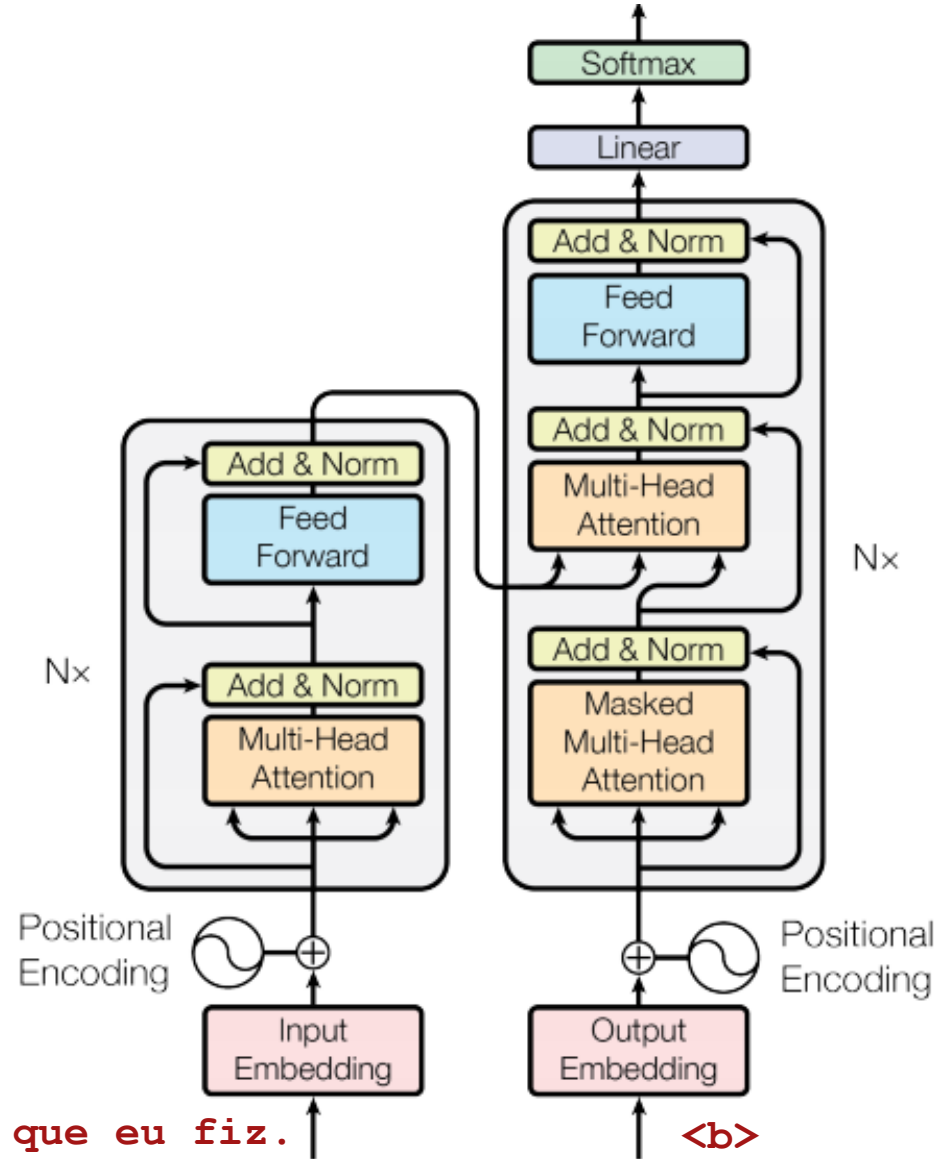
Decoder

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



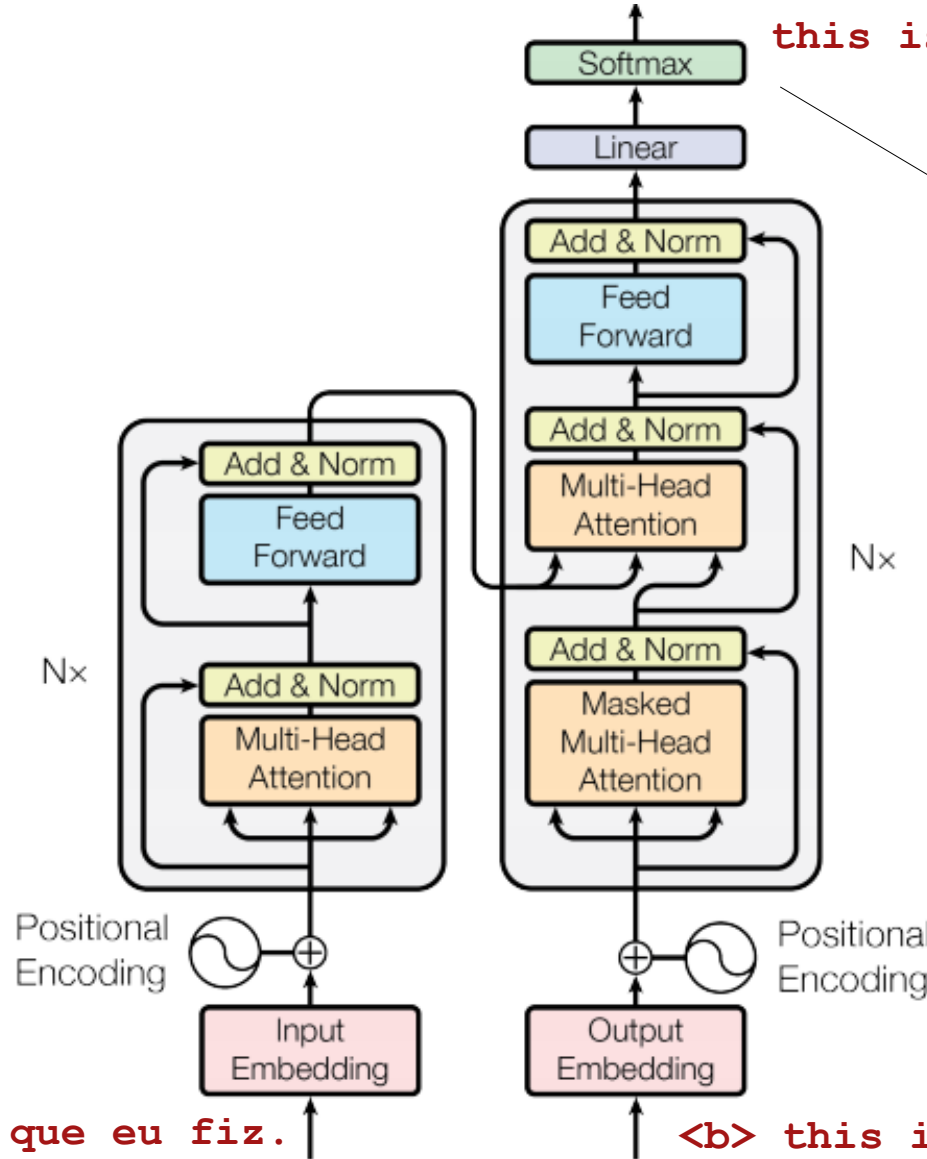
Translator



este é o primeiro livro que eu fiz.

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)

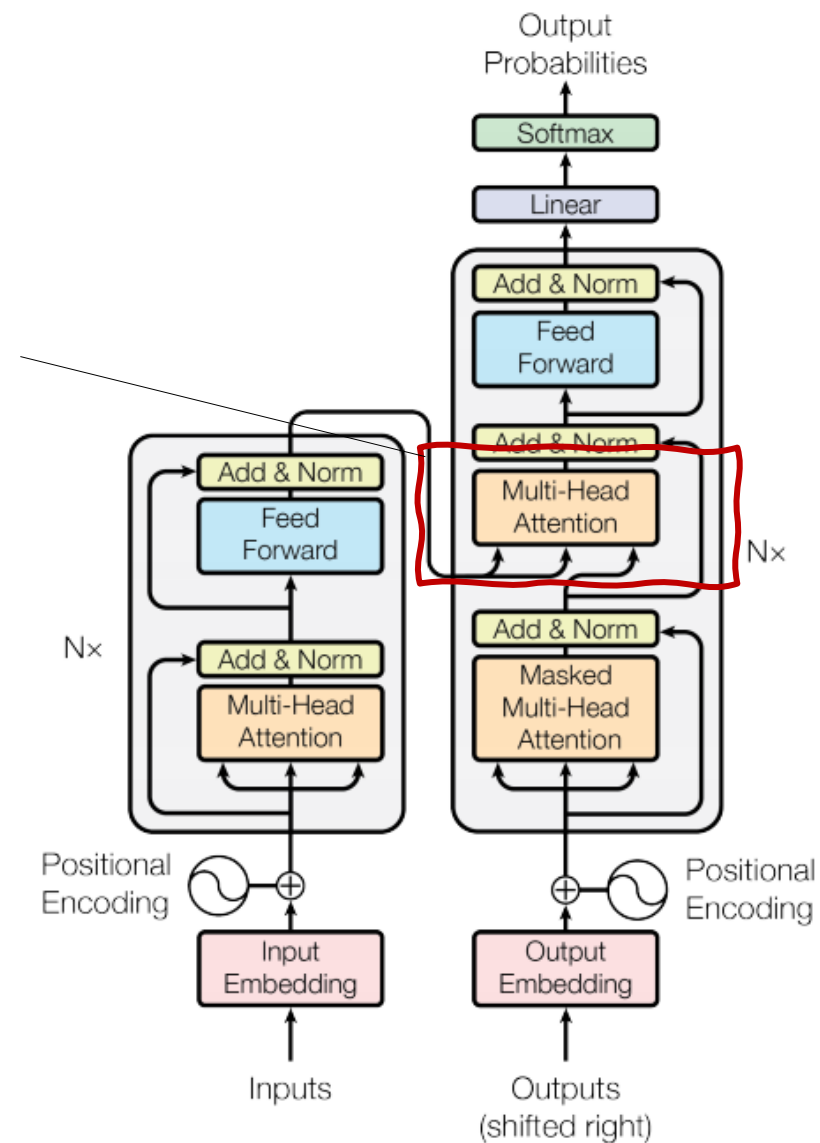
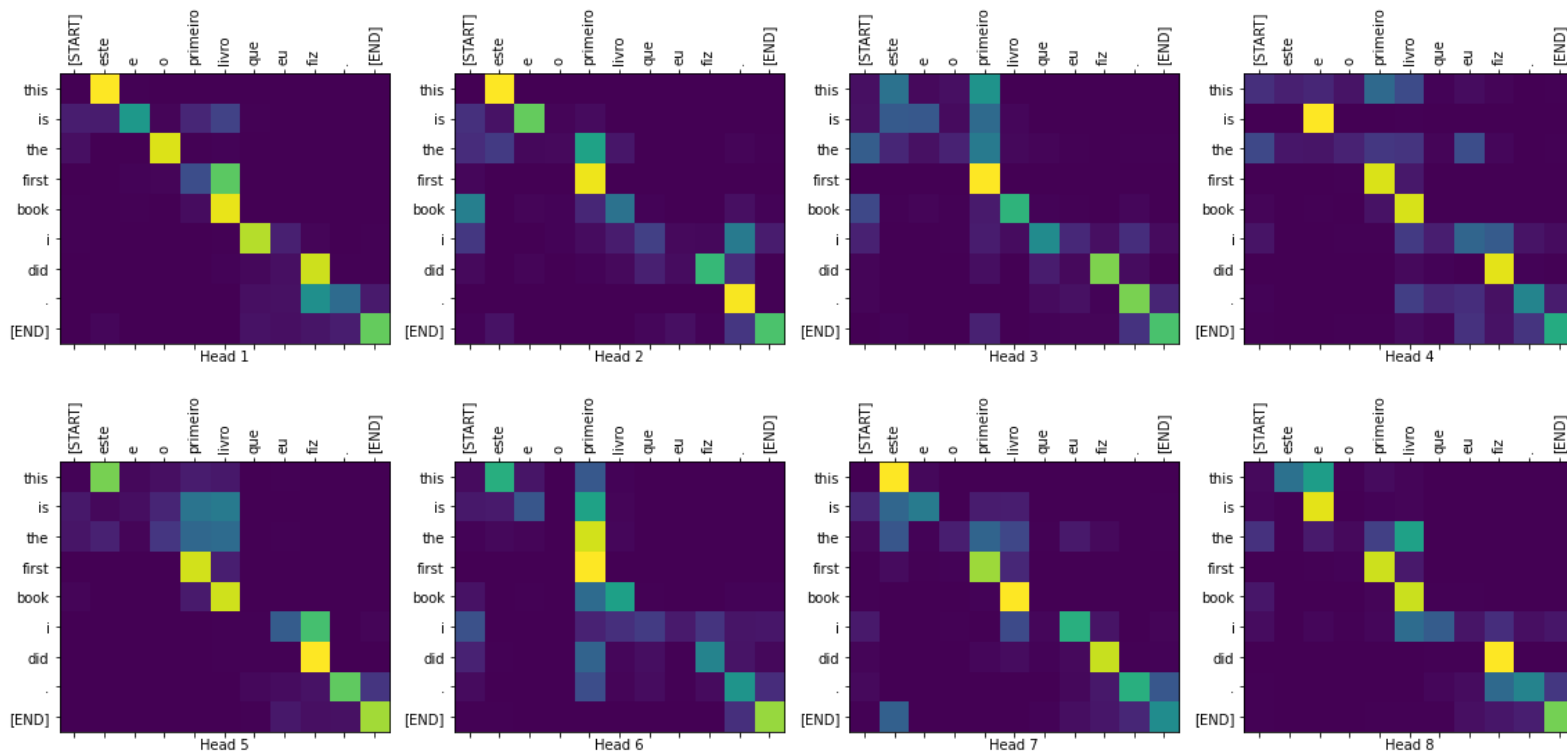
This the true output
(teacher forcing)

este é o primeiro livro que eu fiz.

 this is the first book i've ever done.

Attention Maps

Multiple heads, mixing layer, topmost decoder block



Google Colab

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