Deep Learning

A course about theory & practice

UNIVERSITÀ DI PAVIA

Attention and Transformers

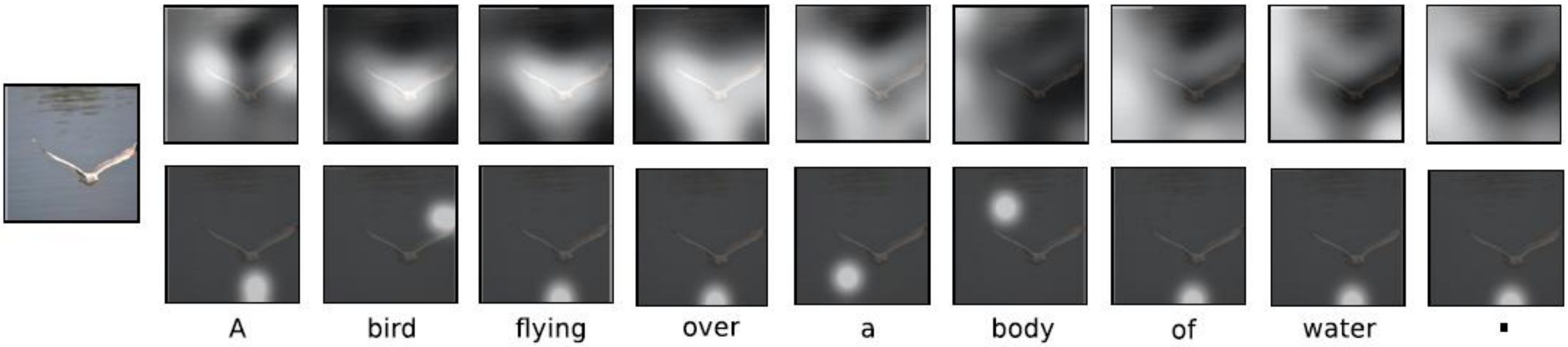
Marco Piastra

Attention is what we need? (intuition)

Generating Text Captions from Images

- ***DCNN + RNN***

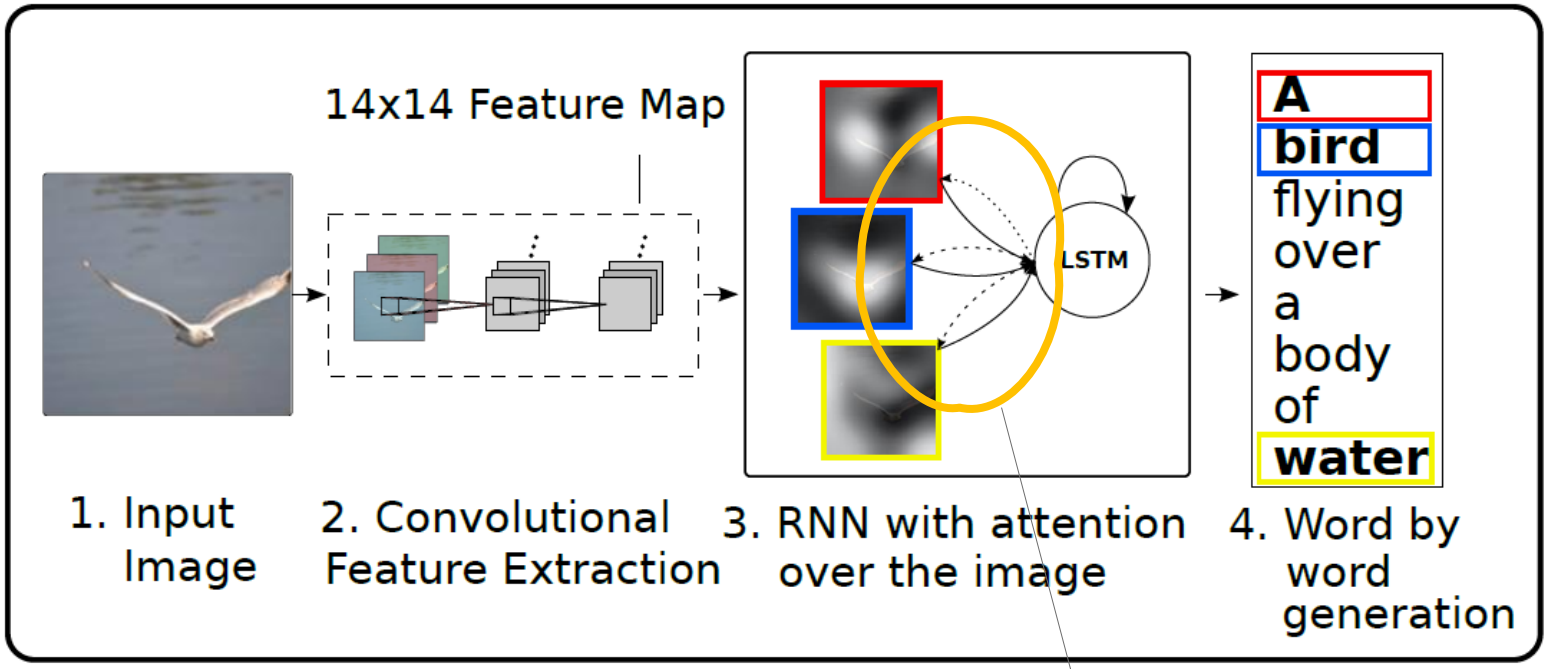
  [*Show, Attend and Tell: Neural Image Caption Generation with Visual Attention*, Xu et al., 2015]



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.

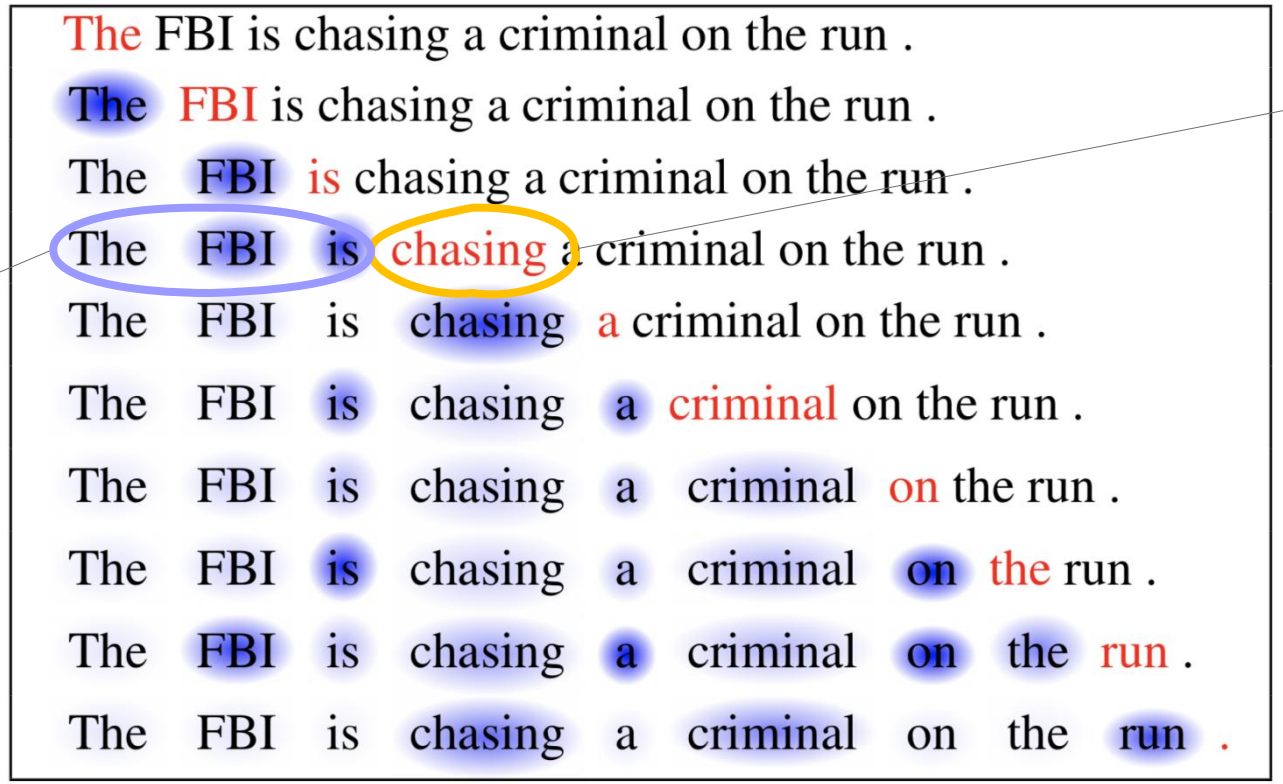
A giraffe standing in a forest with trees in the background.

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

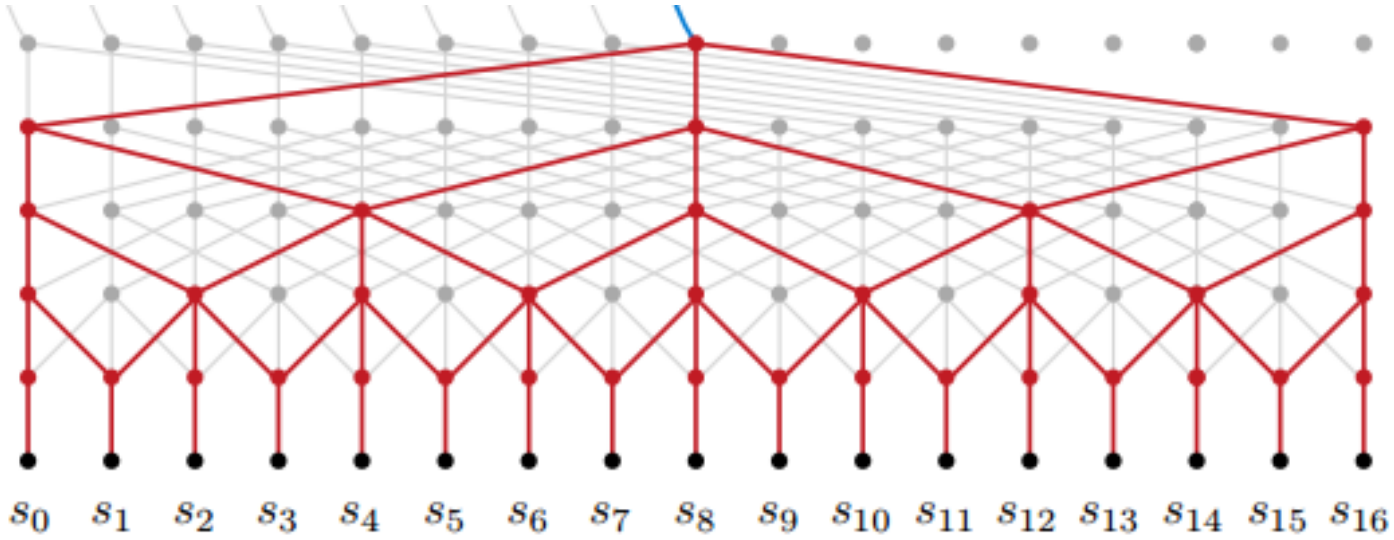
Relevance weights of previous words

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

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

Attention as a Kernel
(another perspective over convolution)

Attention as Kernel

- ***Attention Pooling***

  Consider an input-output relation and a dataset

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

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

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

  'kernel'

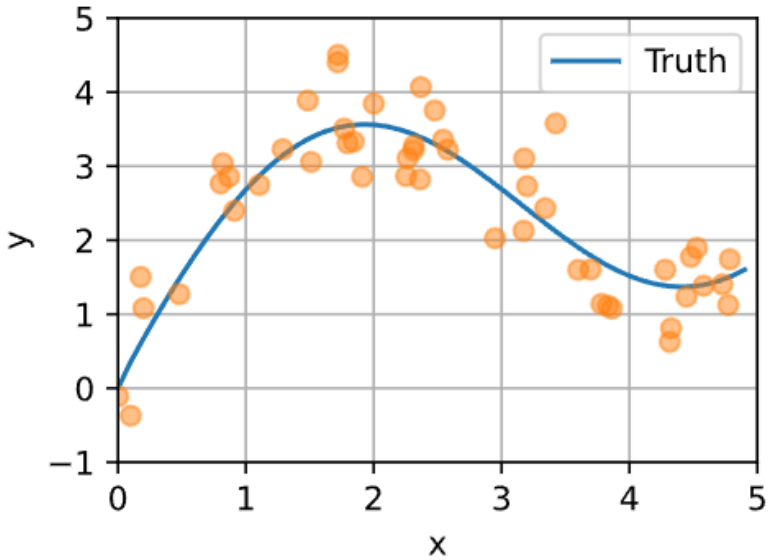
  Example:

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

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

  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

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

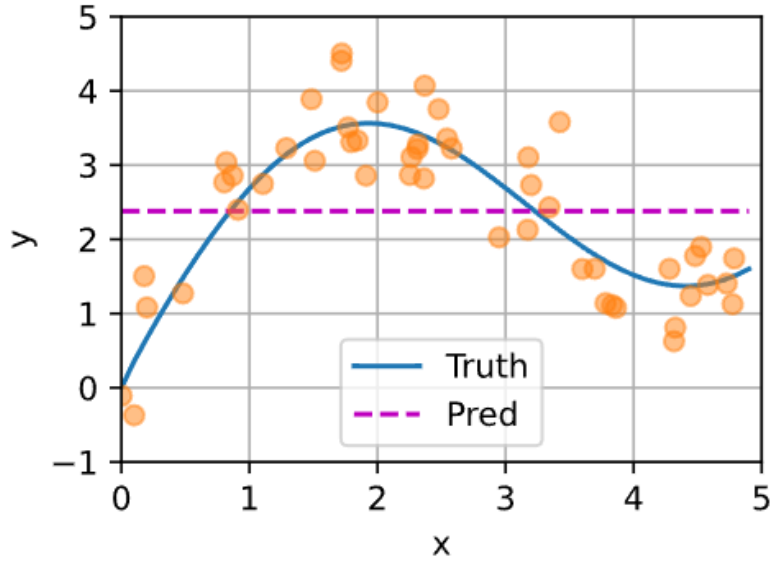
  Attention Pooling is defined as a function on each input component

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

  Example:

  Global average as constant kernel (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

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

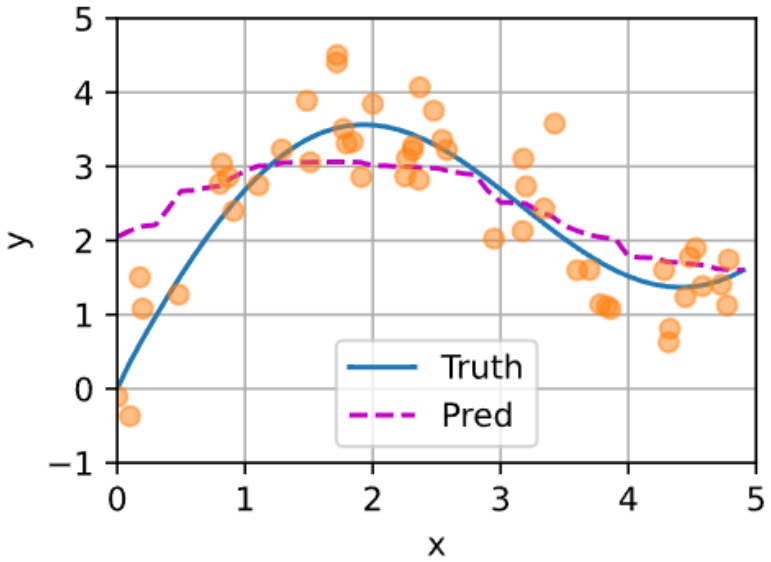
  Attention Pooling is defined as a function on each input component

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

  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

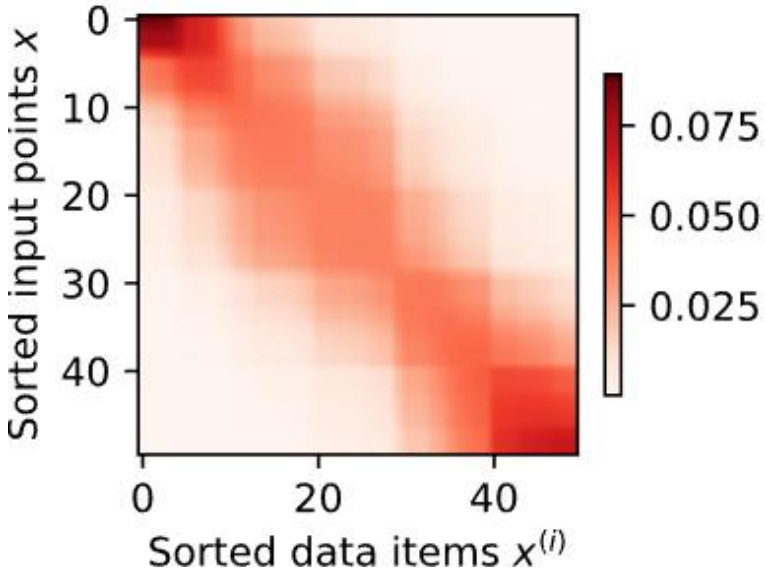
- ***Gaussian Kernel and Softmax***

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

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

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

'Attention Map'



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***

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

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

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***

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

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

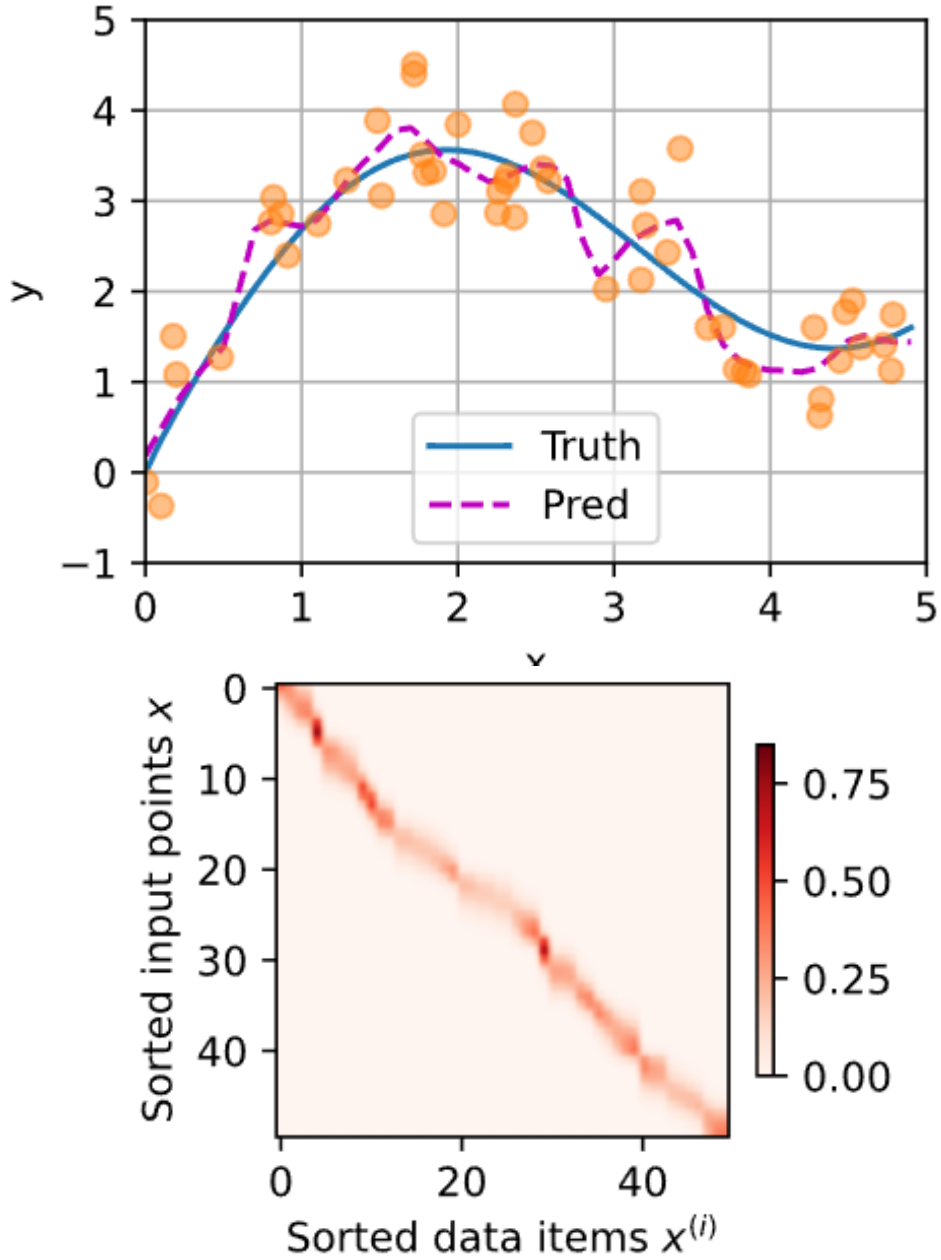
This requires training of the (unique) parameter w

Consider an MSE *loss* function:

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

and perform *gradient descent*

The (Gaussian) attention field becomes 'sharper'



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

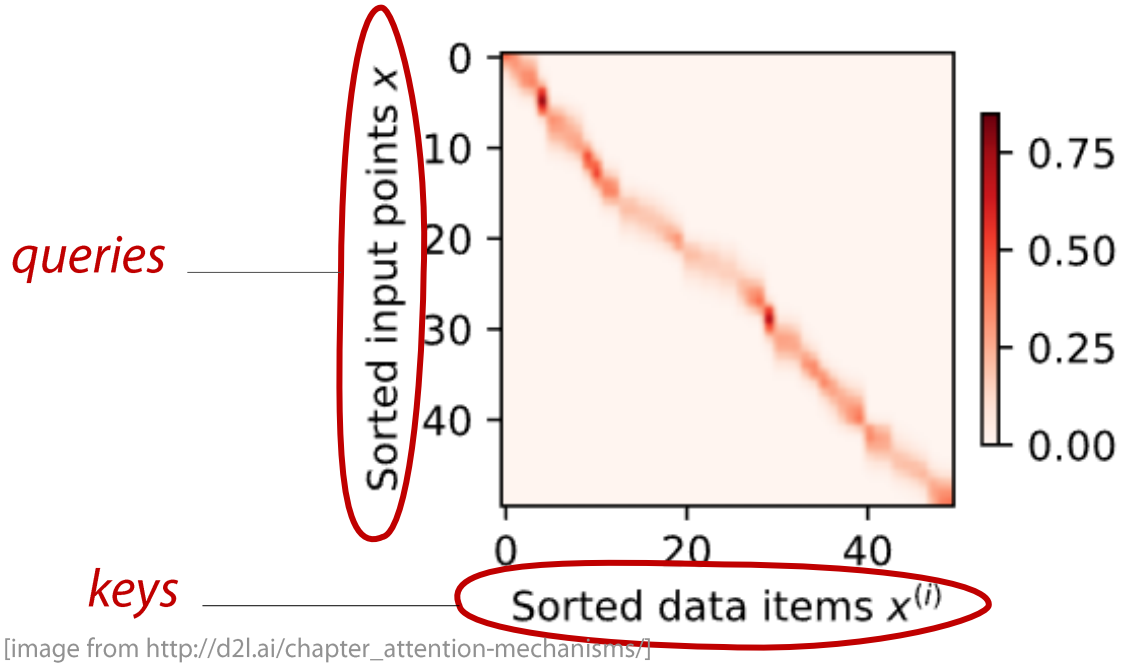
Attention as Kernel

- ***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)

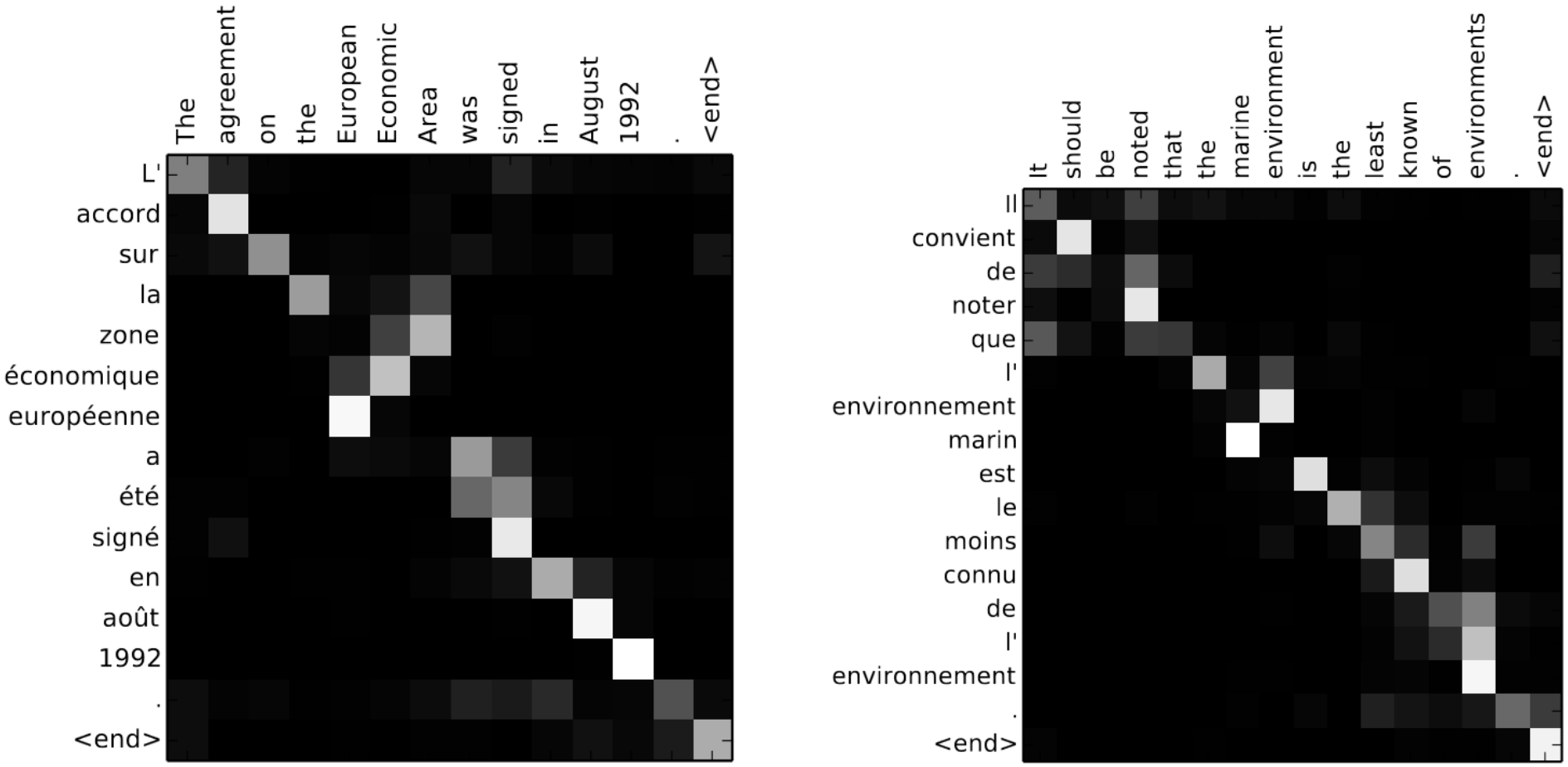


[image from http://d2l.ai/chapter_attention-mechanisms/]

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***

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

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

In general:

- *queries* and *keys* could come from different spaces
- the *attention map (kernel)* 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***

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

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

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

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 assumption that each component of the two vectors is an independent, standard-normal random variable (zero mean and unit variance)

  Under such assumption, each component has $\mu = 0$ and $\sigma^2 = 1$, therefore

$$\mathrm{Var}(\boldsymbol{q} \cdot \boldsymbol{k}) = \sigma^2 = d \quad \Rightarrow \quad \sigma = \sqrt{d}$$

Attention Scoring Function

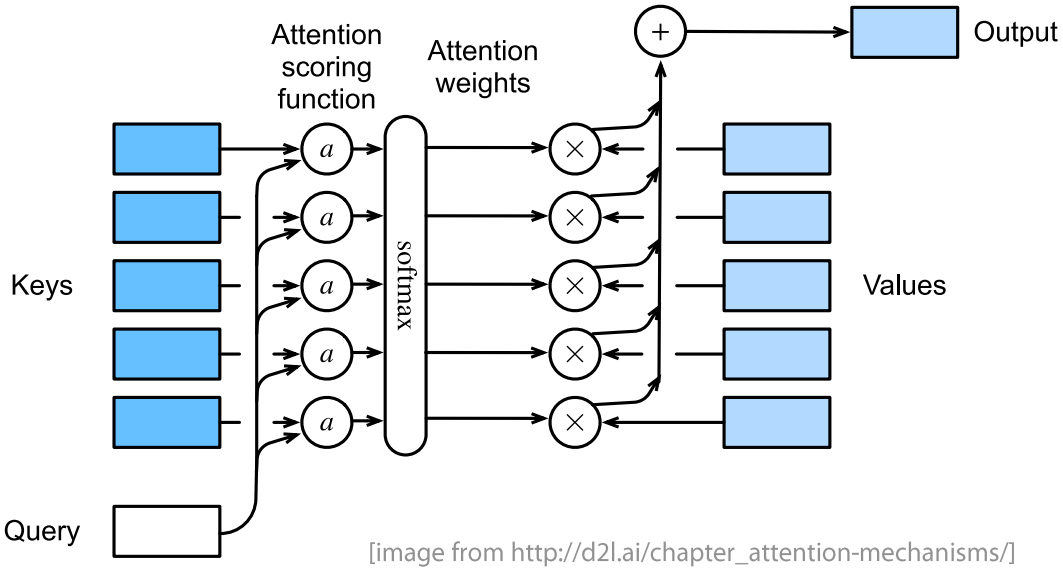
- ***Scaled Dot-Product Attention***

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:

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



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

Attention Map

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

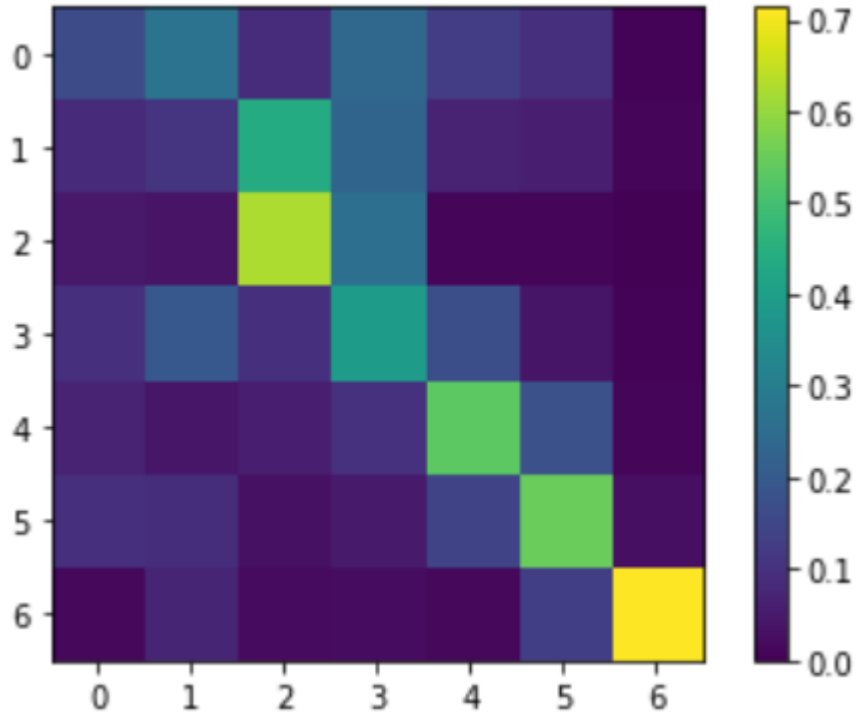
  Assume that both *queries* and *keys* come from the same space:

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

Then:

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

applied row-wise

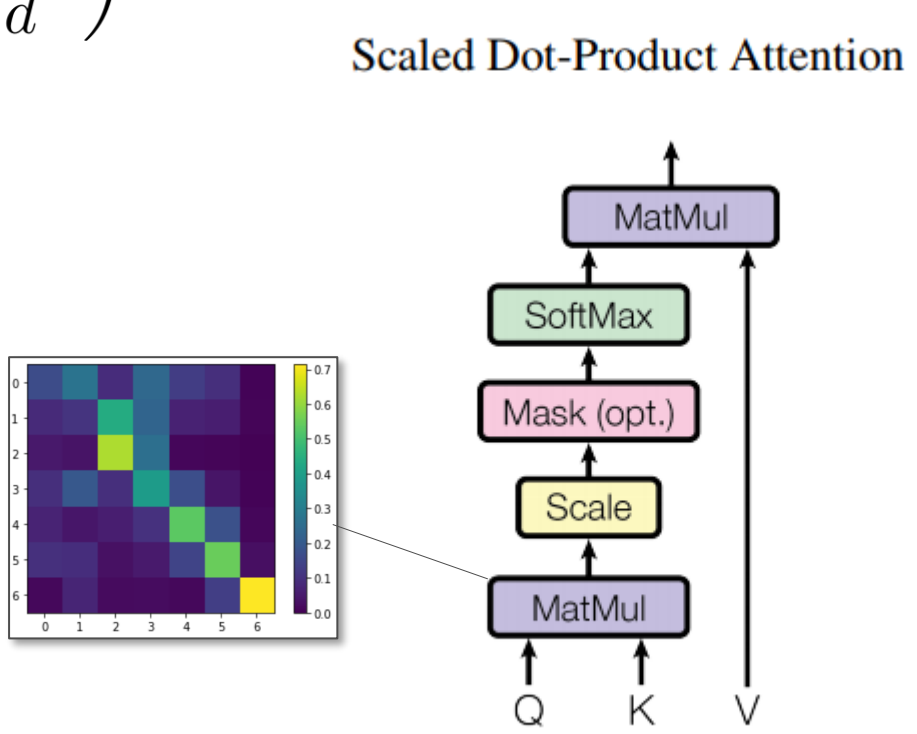


Scaled Dot-Product Attention

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

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

This is the basic building block



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

Scaled Dot-Product Self-Attention

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

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

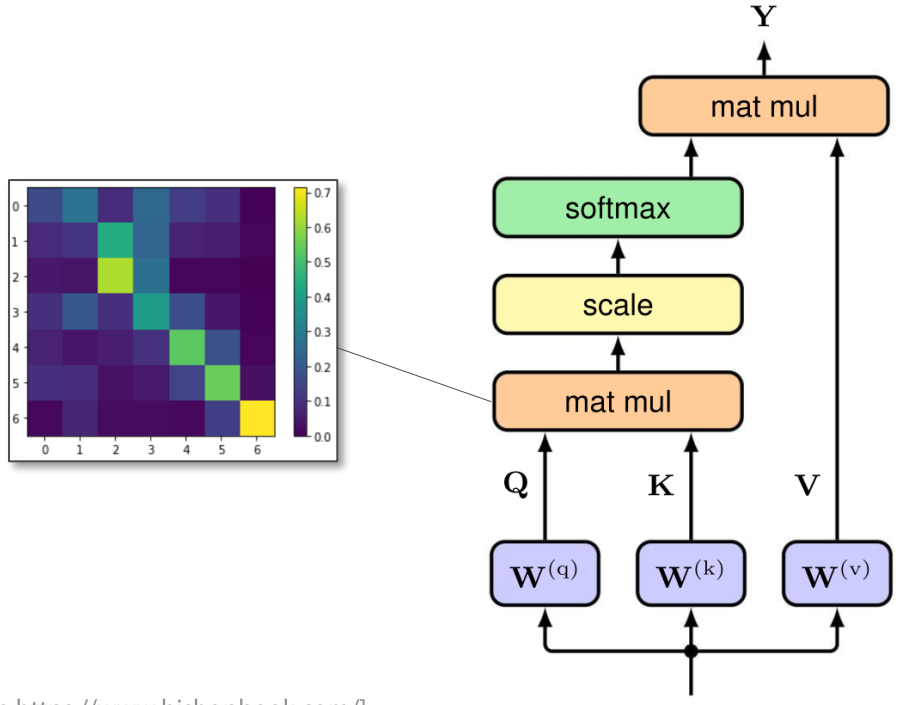
In *self-attention*, flexibility can be gained by adding a *linear transformation*:

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_q^T$$
$$\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}_k^T$$
$$\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_v^T$$

where:

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

are parameters to be trained



[image from https://www.bishopbook.com/]

Scaled Dot-Product Cross-Attention

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

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

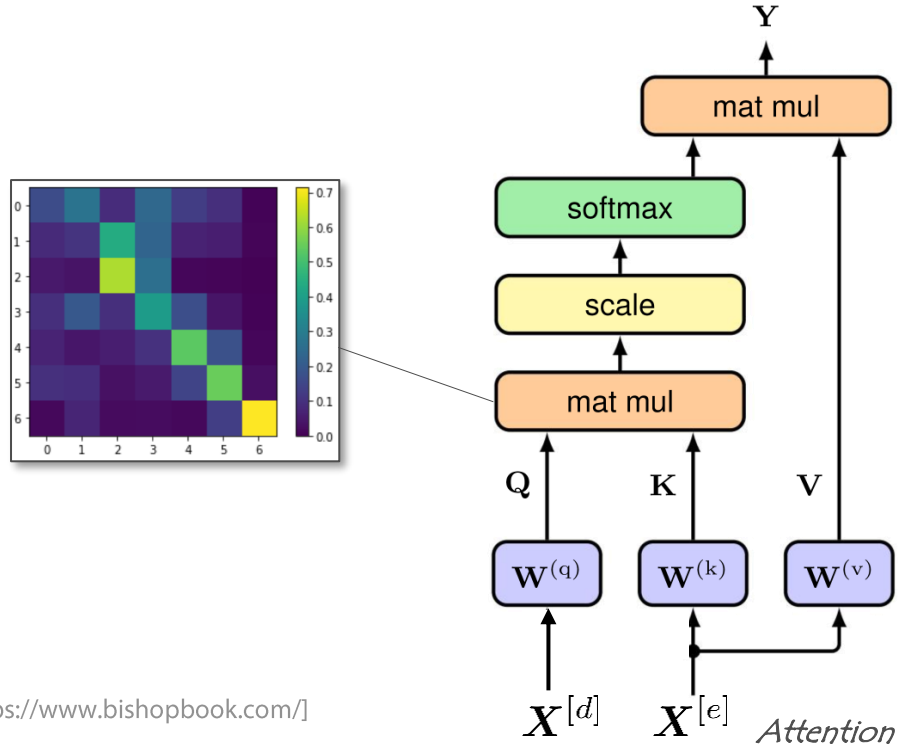
In *cross-attention* blocks the query $\boldsymbol{Q}$ comes from one source, while $\boldsymbol{K}$ and $\boldsymbol{V}$ are from another:

$$\boldsymbol{Q} = \boldsymbol{X}^{[d]}\boldsymbol{W}_q^T$$
$$\boldsymbol{K} = \boldsymbol{X}^{[e]}\boldsymbol{W}_k^T$$
$$\boldsymbol{V} = \boldsymbol{X}^{[e]}\boldsymbol{W}_v^T$$

where:

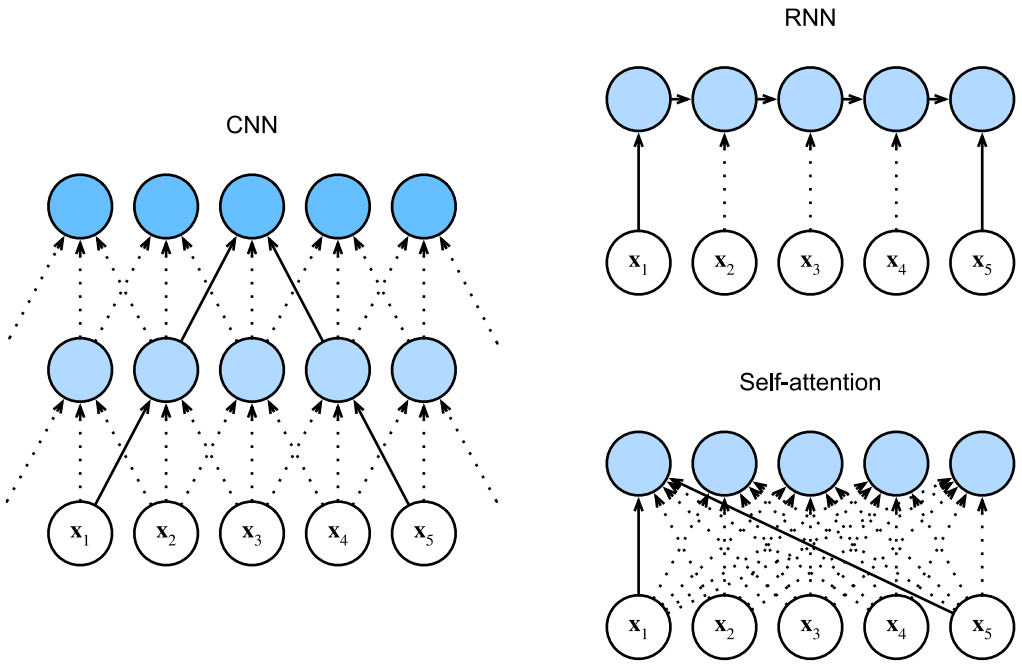
$$\boldsymbol{W}_q^{[d]},\ \boldsymbol{W}_k^{[e]},\ \boldsymbol{W}_v^{[e]} \in \mathbb{R}^{n \times d}$$

are parameters to be trained

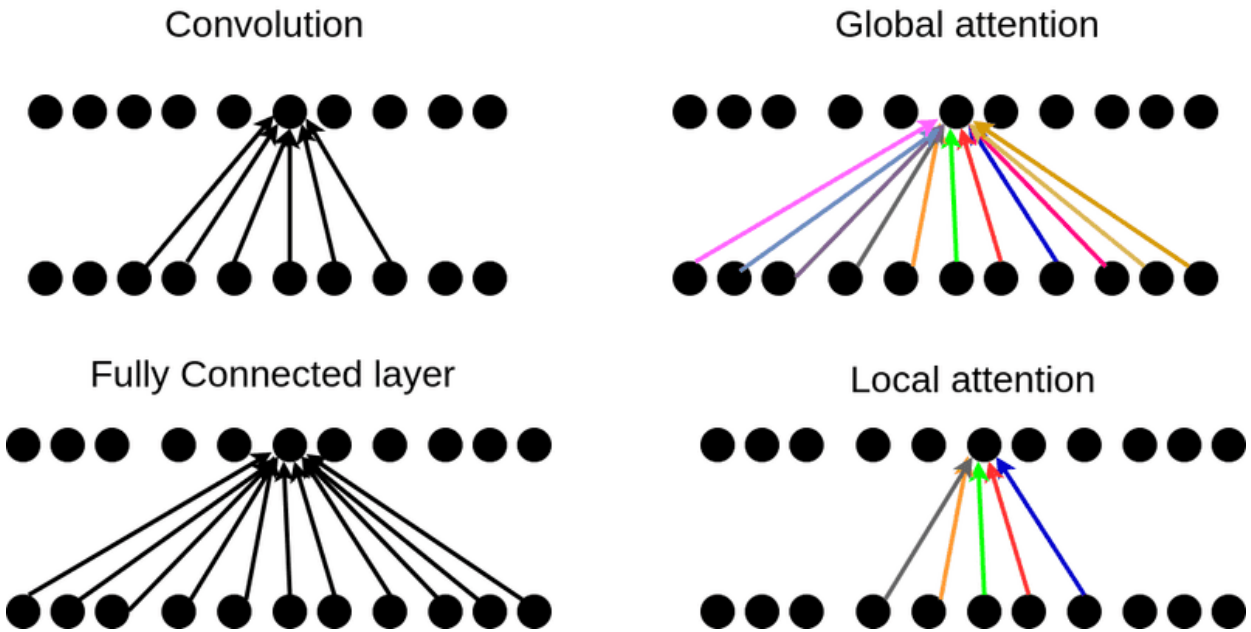


[image from https://www.bishopbook.com/]

Attention vs Convolution vs RNN



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



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

Positional Encoding

Positional Encoding

- ***In attention blocks, input topology may be lost***

Consider the main attention transformation:

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

$$= \boldsymbol{A}\,\boldsymbol{V}$$

Attention matrix

$$Y_{i,j} = \sum_k A_{i,k} V_{k,j}$$

- *Summation is commutative: any permutation over the index k will produce the same result*
- *In other words, the ordering of keys indexes becomes irrelevant*

Positional Encoding

- ***Using sine and cosine***

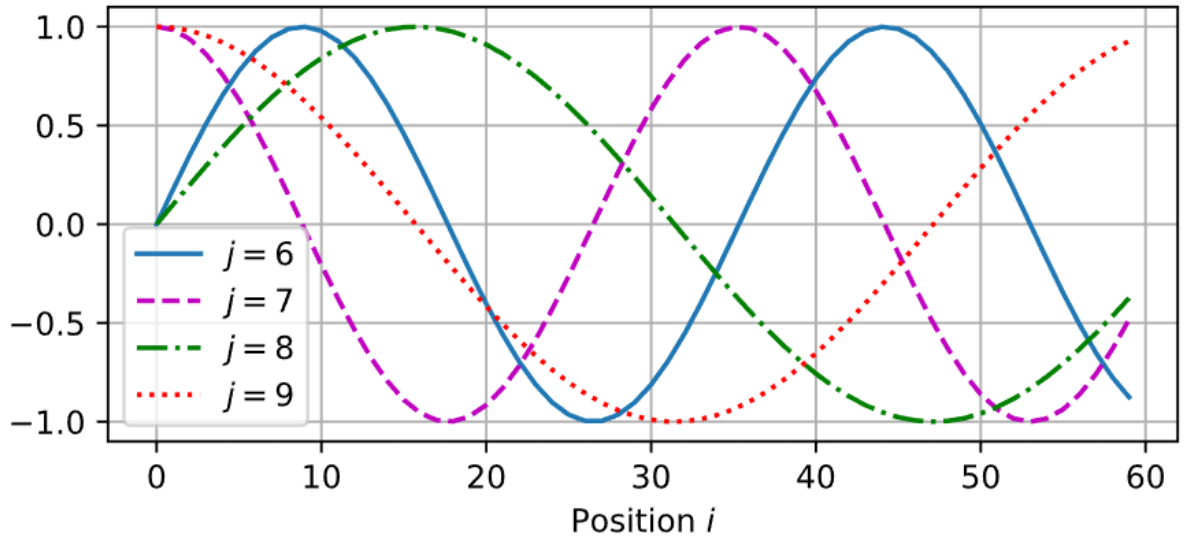
  Assume we want to sum a *positional encoding* vector $\boldsymbol{p}$ to a data vector $\boldsymbol{x}$ ($\boldsymbol{x}, \boldsymbol{p} \in \mathbb{R}^d$)

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

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

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

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



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**

  same dimension

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

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

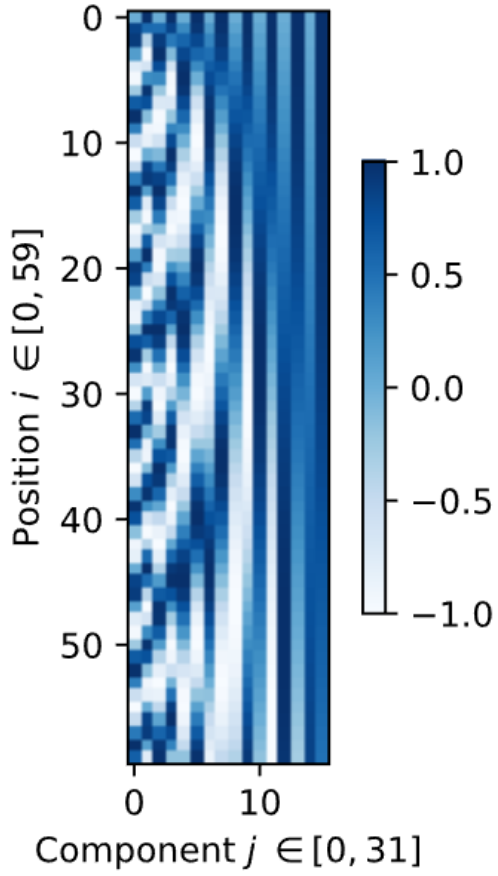
  To do so, we can use a *sine – cosine* representation:

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

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

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

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



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 learnt