Deep Network Development

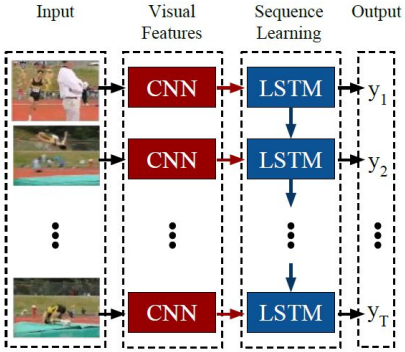
Lecture #10

Viktor Varga
Department of Artificial Intelligence, ELTE IK

Last week - Processing sequences - Cases

N → 1: e.g., sentiment analysis

N → N:
e.g., labeling each frame of a video



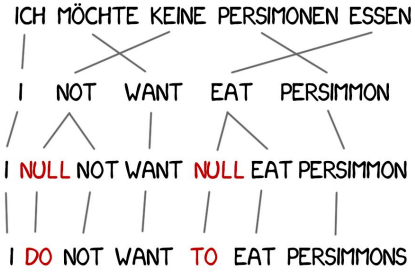

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

1 → N: e.g., image captioning

M → N: e.g., machine translation



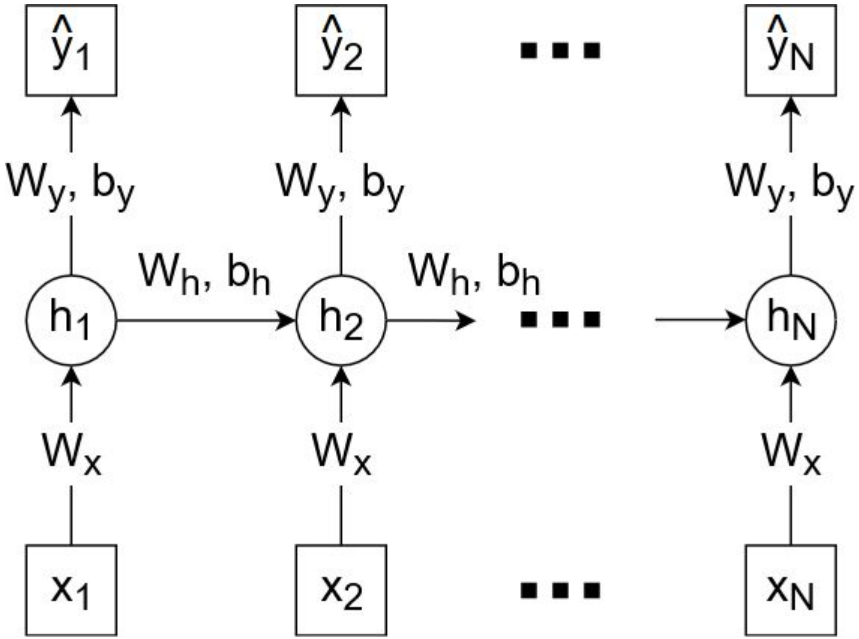
Last week - “Vanilla” RNN, **N → N**

$$h_t = \sigma_h(W_h h_{t-1} + W_x x_t + b_h)$$

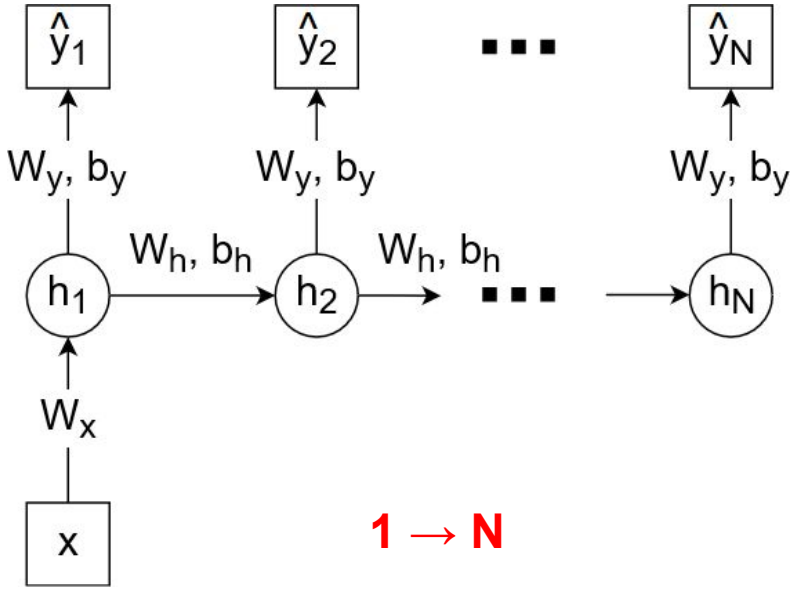
$$\hat{y}_t = \sigma_y(W_y h_t + b_y)$$

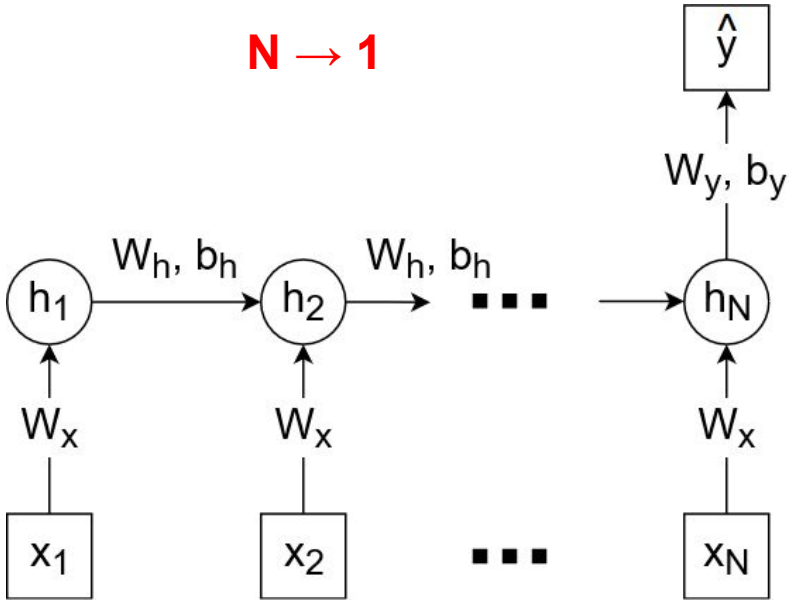
Weights shared across time:
Here, regardless of the sequence length, we only have 3 unique weight matrices and 2 bias vector.

h_i vectors: Hidden representation vectors (“memory”) store the current state of the sequence processing.



Last week - “Vanilla” RNN, **1 → N, N → 1**

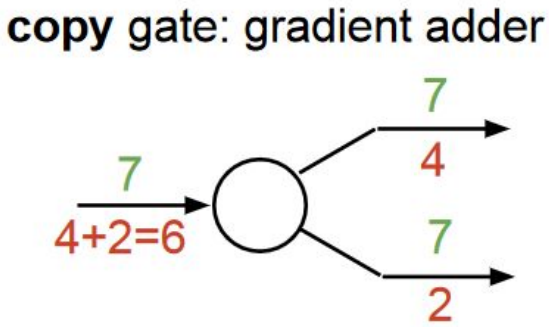


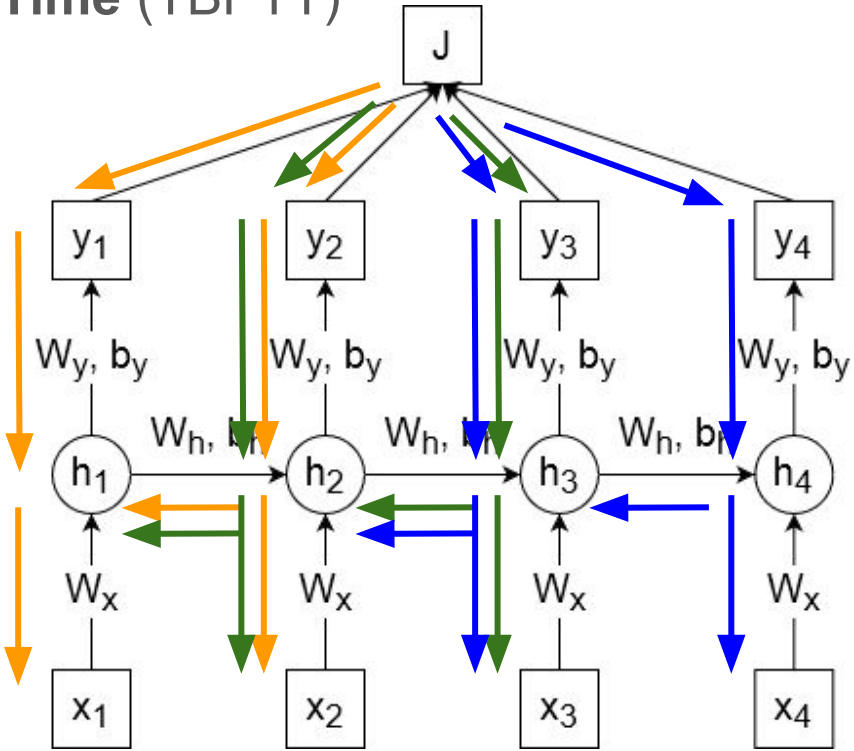


Last week - RNN, backpropagation

Truncated Backpropagation Through Time (TBPTT)

We **stop** backpropagation **after a given number of time steps**.

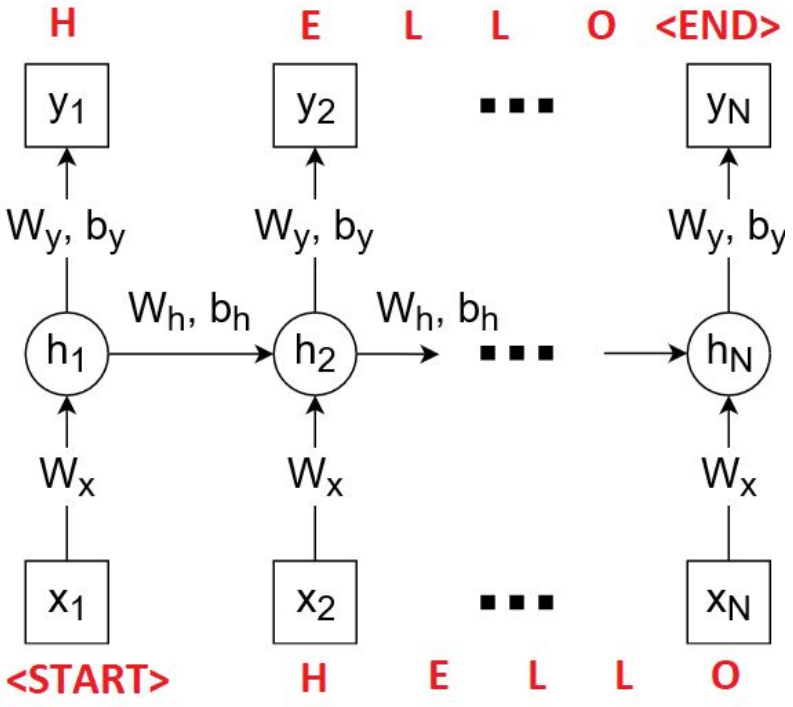




Last week - RNN, time series prediction

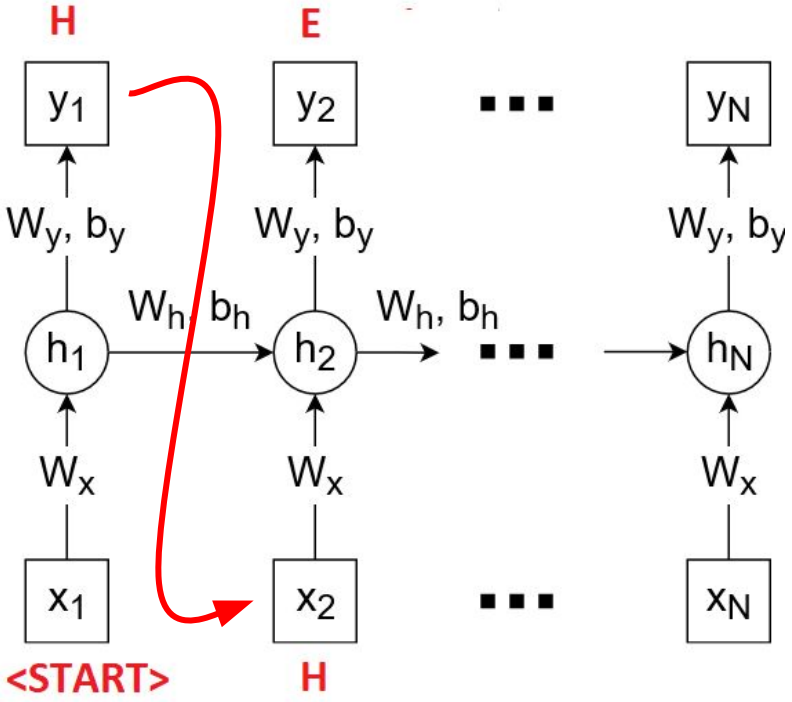
Task: Predict the next element in a time series based on the last few elements!

- With a "**N → N**" model:
  Learn to **estimate elements # t-N+1, …, # t from** input **elements # t-N, …, # t-1**!



Last week - RNN, generative use

The trained RNN can be used to **generate new time series!**



at first:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt    h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

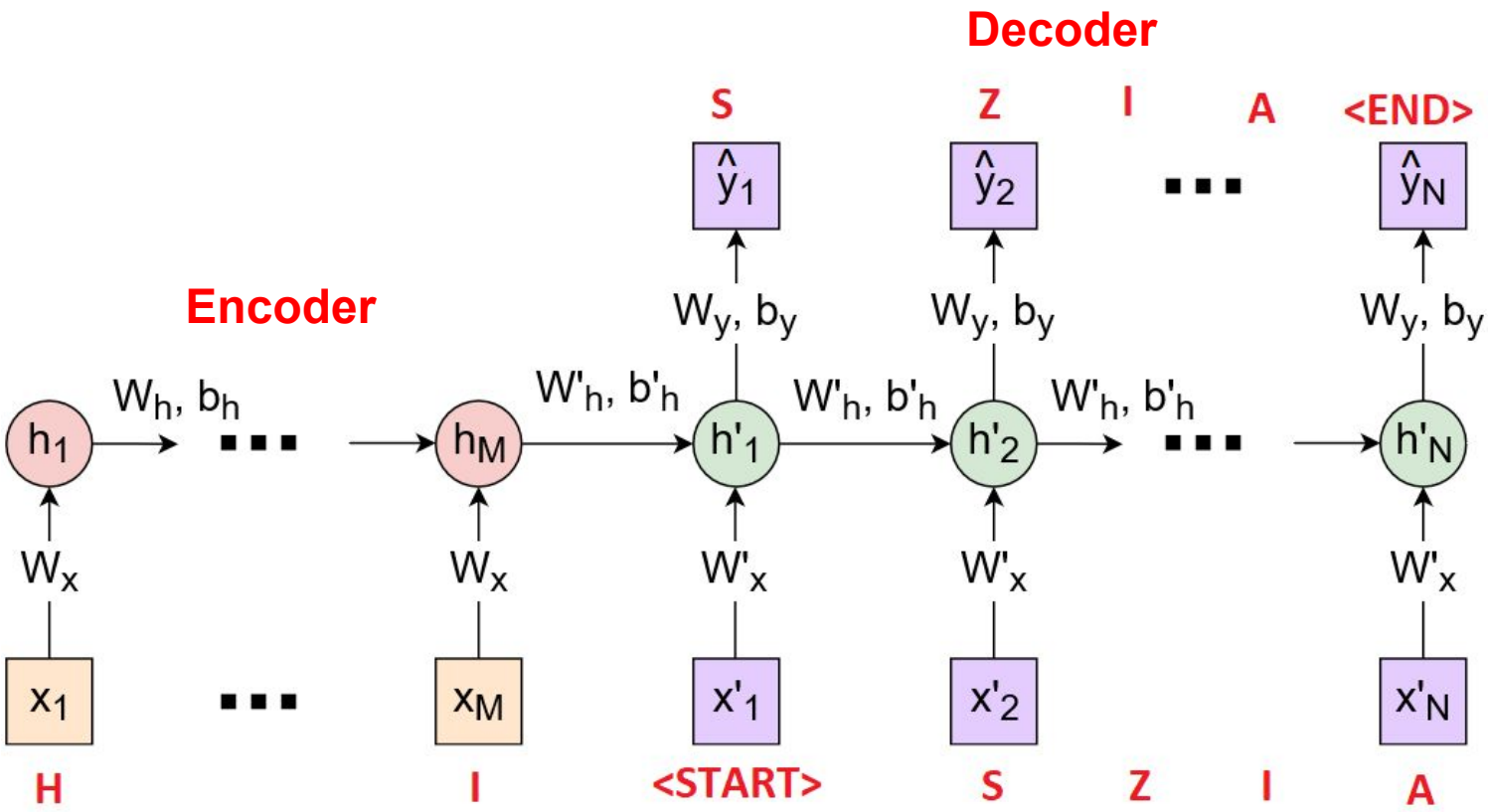
train more

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

train more

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

Last week - Seq2seq RNN, for **M** → **N** cases

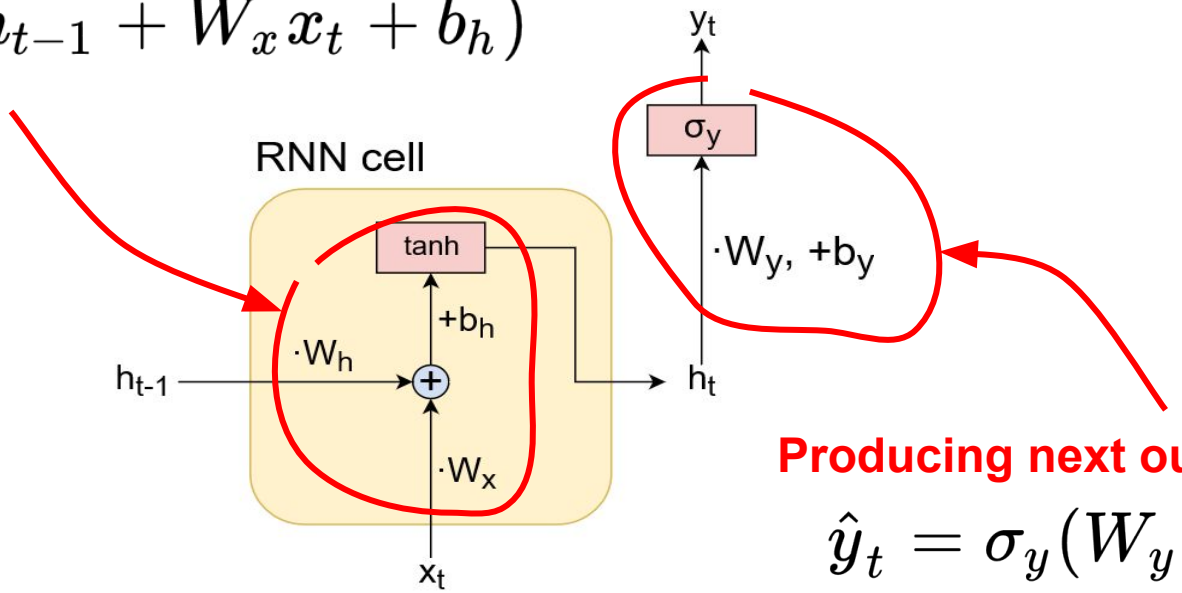


Last week - “Vanilla” RNN, “RNN cell”

RNN cell

Updating hidden representation (temporal step)

$$h_t = \sigma_h(W_h h_{t-1} + W_x x_t + b_h)$$



Producing next output

$$\hat{y}_t = \sigma_y(W_y h_t + b_y)$$

Last week - The unstable gradient problem in RNNs

The scaling of the backpropagated gradient in each time step:

$$\frac{\partial h_t}{\partial h_{t-1}} = \sigma_h'(w_h h_{t-1} + w_x x_t + b_h) \cdot w_h$$

The weights are shared across time, so we **multiply the gradient over and over again by the same number** (matrix) in each time step.

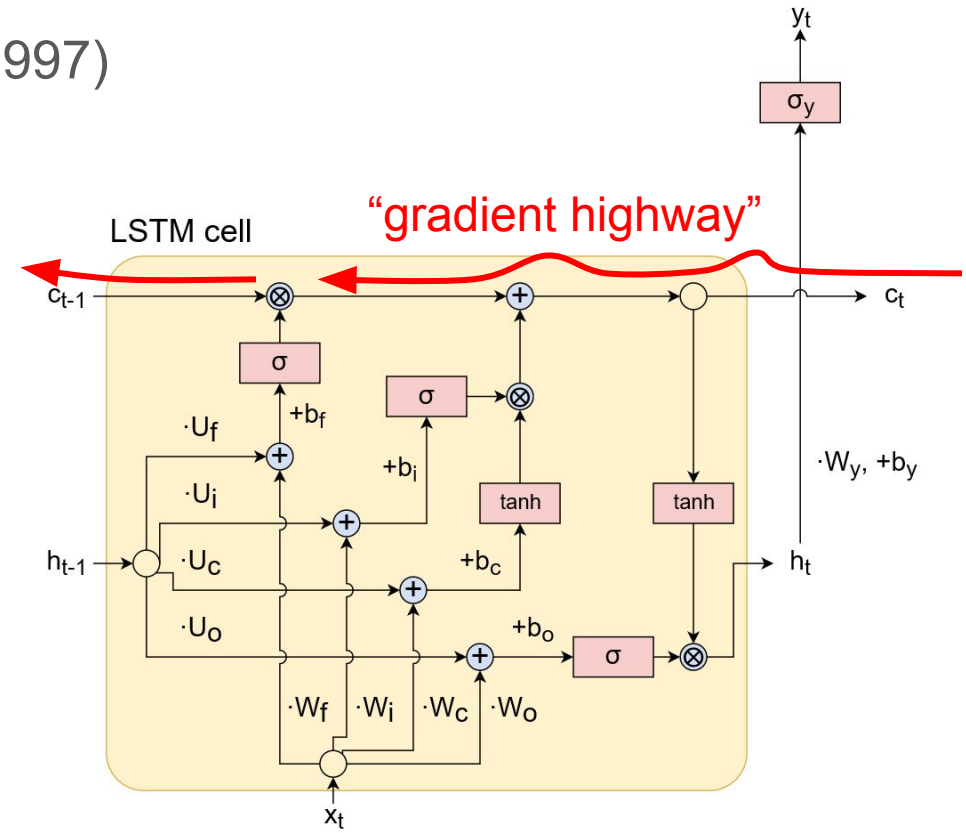
Consequence: Classic “Vanilla” RNNs will usually be **unable to learn to detect relationships** in the data that span a timescale **of more than 10–15 steps.**

Last week - Mitigating the unstable gradient problem

Long short-term memory (LSTM, 1997)

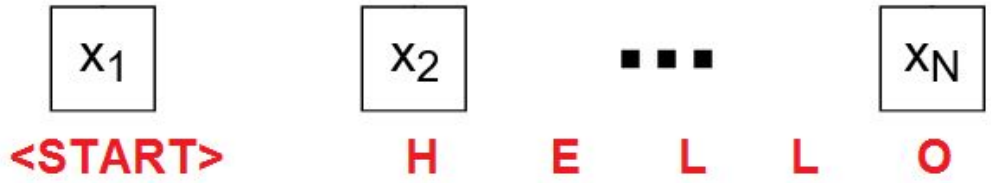
Advantages:

- **Uninterrupted flow of gradient** during backpropagation (except for the 'forget gate')
- **Forget gate:** More explicit control of what information is retained in the cell state and what is discarded in each time step.



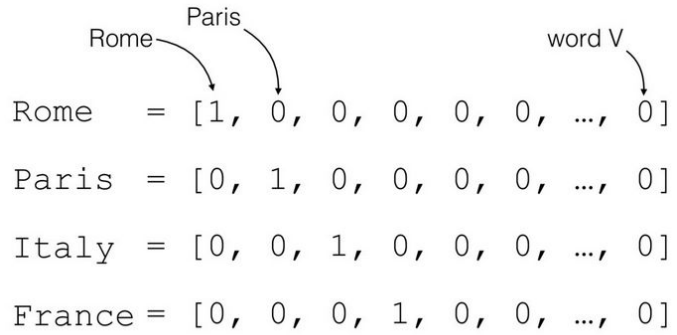
Different ways to represent text

- Text as a sequence of **characters.**



Different ways to represent text

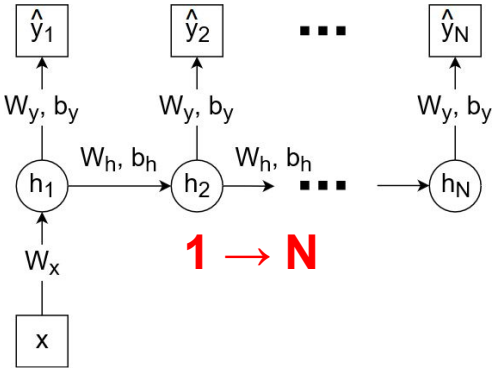
- Text as a sequence of **characters.**
- Text as a sequence of **words.**

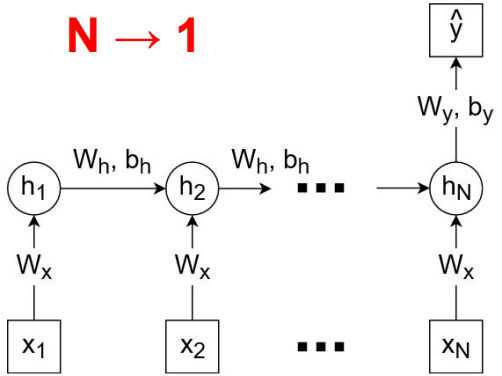


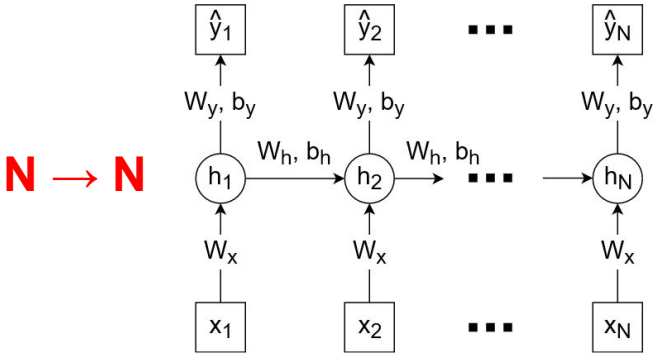
Different ways to represent text

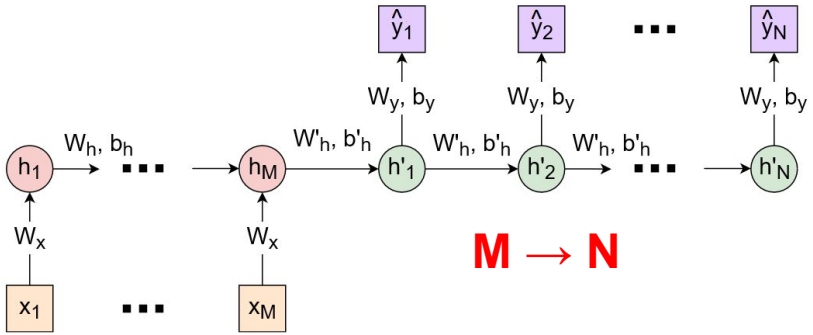
- Text as a sequence of **characters.**
- Text as a sequence of **words.**
- *Subword tokenization (later)*
- *Word embeddings (later)*

Until now - Processing sequences, RNN









Problems with Recurrent Neural Networks

1. The **implementation is inefficient** when the sequence length is variable or too long.

Problems with Recurrent Neural Networks

1. The **implementation is inefficient** when the sequence length is variable or too long.

The **processing of each time step** in recurrent networks **can only occur sequentially**, one after another; it cannot be parallelized.

Training: Batch-level parallelism is possible, but variable-length or unexpectedly long sequences make this difficult.

Inference: The inference of a single sequence cannot be parallelized.

Problems with Recurrent Neural Networks

1. The **implementation is inefficient** when the sequence length is variable or too long.
2. **Sequential data processing is not ideal** for learning / identifying complex relationships.

Problems with Recurrent Neural Networks

1. The **implementation is inefficient** when the sequence length is variable or too long.
2. **Sequential data processing is not ideal** for learning / identifying complex relationships.

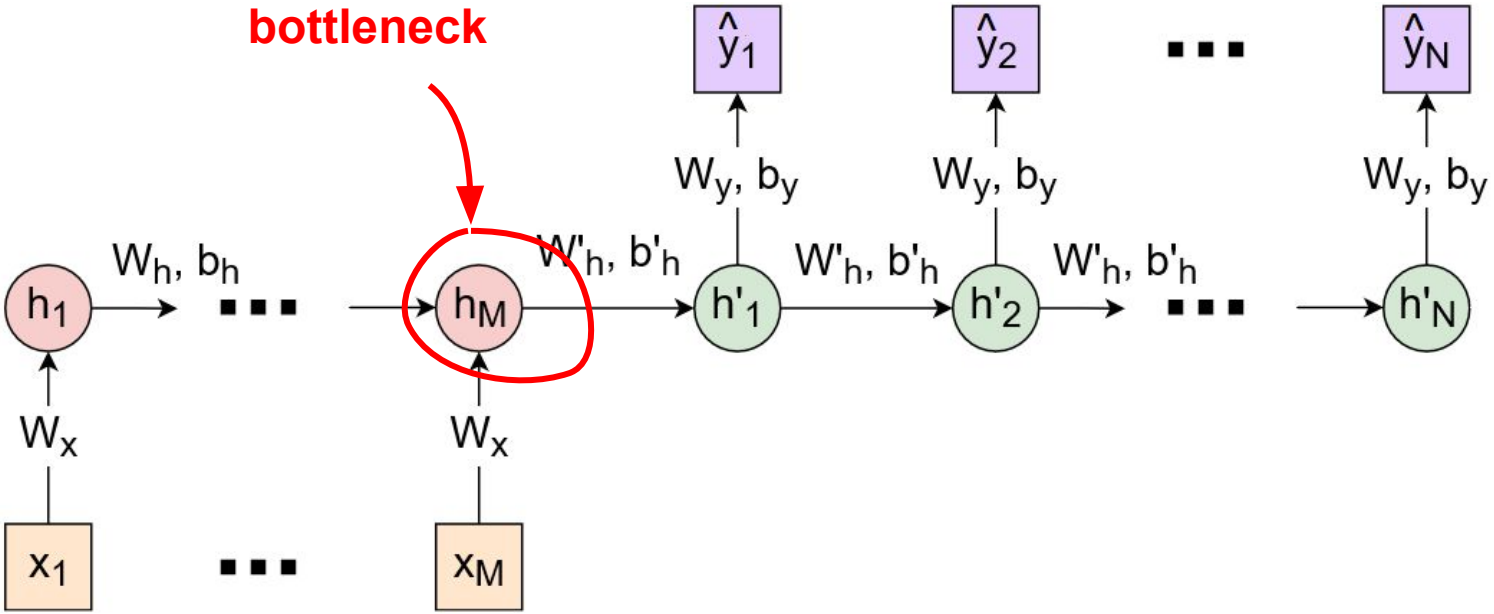
Recurrent networks **cannot look back to an earlier part of the sequence** to gather additional information about it. The network must learn to determine which parts of the input it will still need in the future and which parts it can discard. Only **a vector of fixed size** (the hidden representation) **is available for storing information** deemed important, regardless of sequence length.

Problems with Recurrent Neural Networks

1. The **implementation is inefficient** when the sequence length is variable or too long.
2. **Sequential data processing is not ideal** for learning / identifying complex relationships.
3. Excessively **long backpropagation paths** → Unstable gradient problem.

Problems with Recurrent Neural Networks

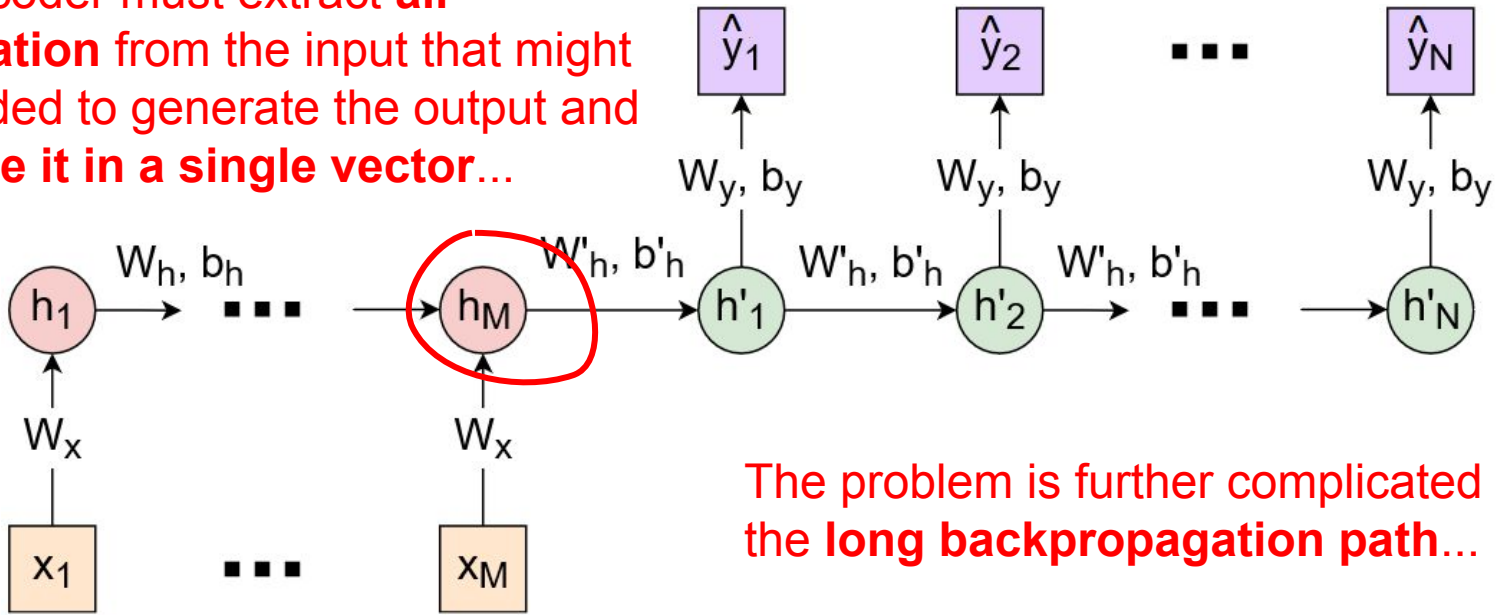
“M → N” RNN variant for machine translation (Seq2seq, 2014)



Problems with Recurrent Neural Networks

"M → N" RNN variant for machine translation (Seq2seq, 2014)

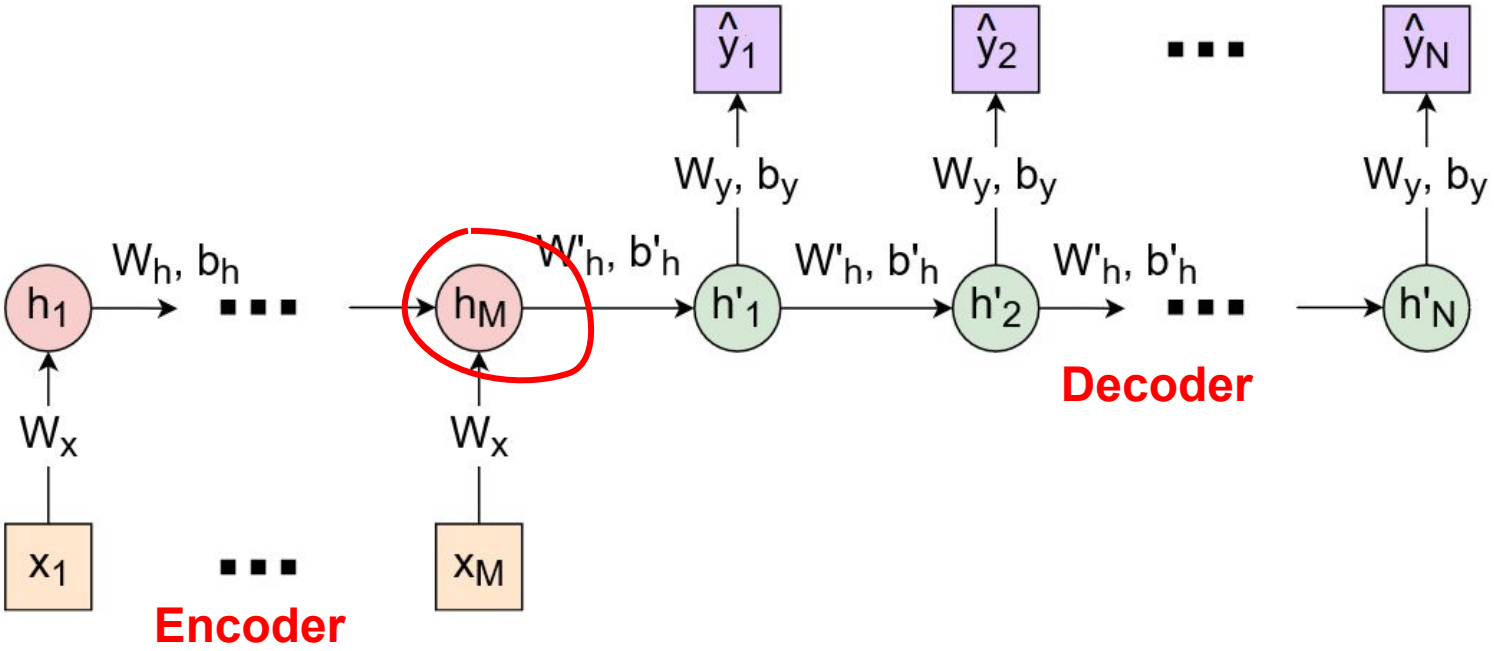
The encoder must extract **all information** from the input that might be needed to generate the output and **squeeze it in a single vector**...



The problem is further complicated by the **long backpropagation path**...

Attention-based models

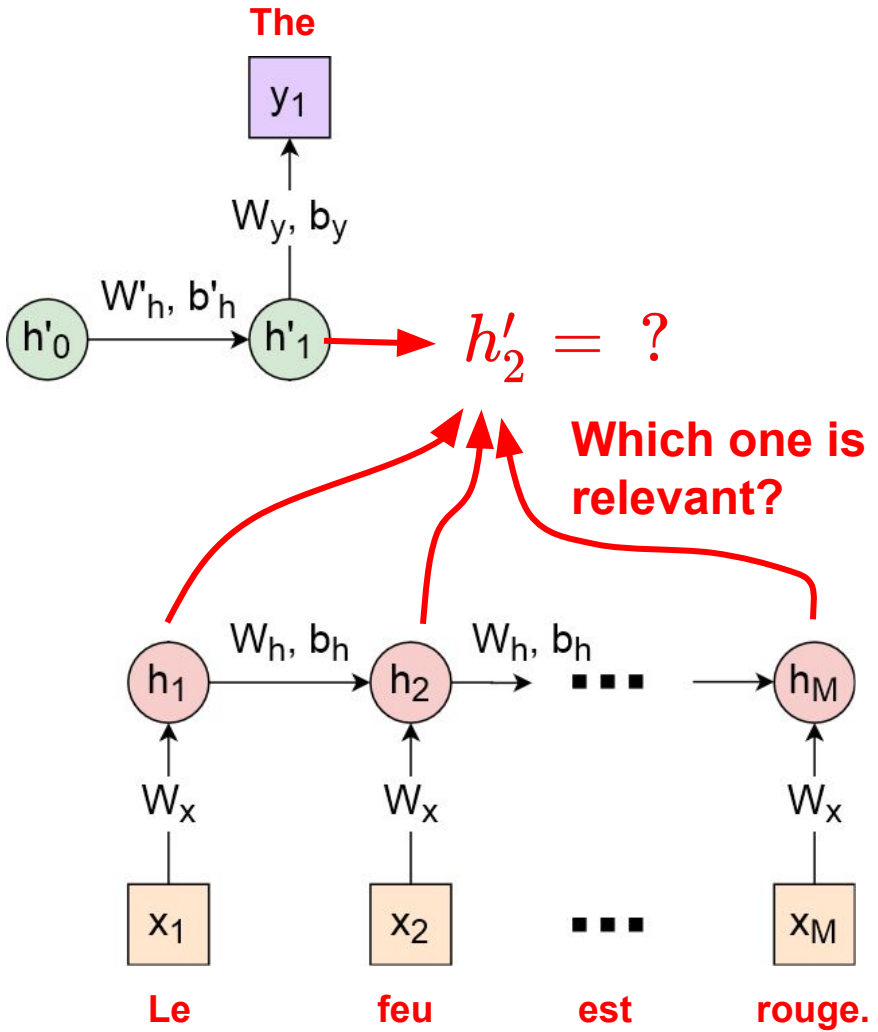
Idea: What if the decoder could access earlier encoder representations too, rather than just the last one?



Attention-based models

Idea: Let's try to find those $\mathbf{h}_i$ encoder representations which are the most useful for creating the next output (decoder representation)!

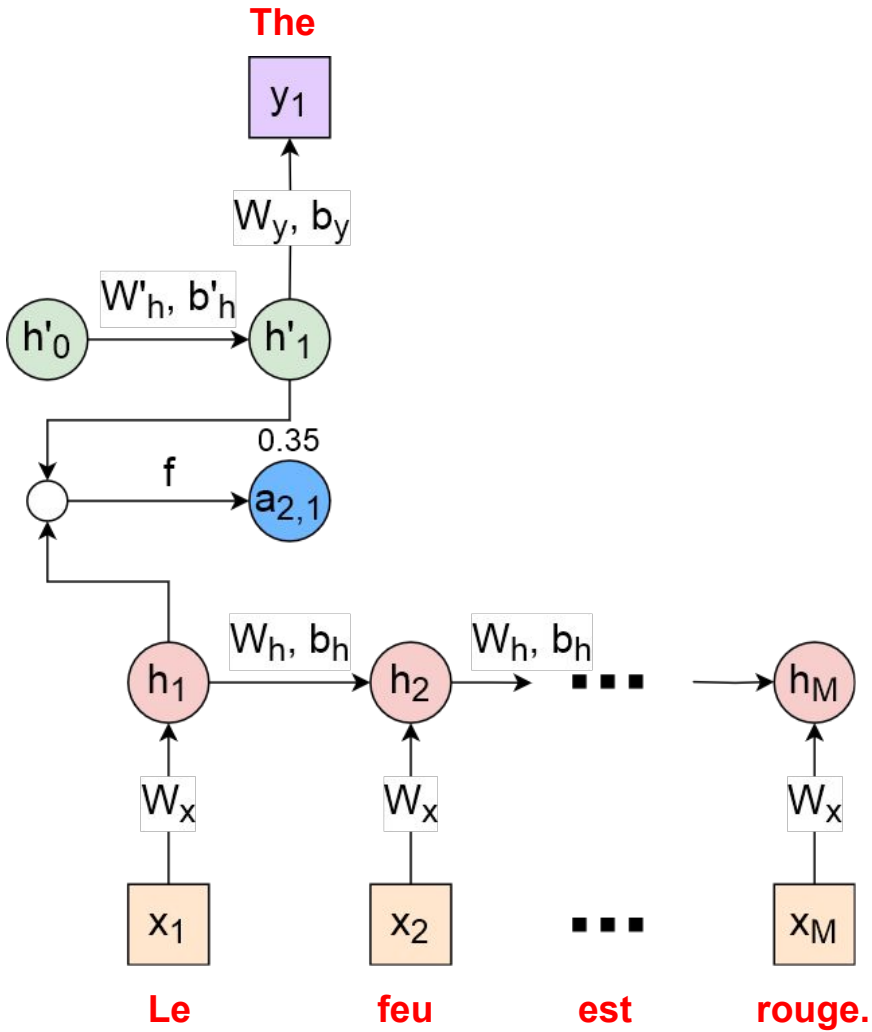
→ **Attention-based models**



“Seq2seq" attention

Let’s learn to estimate,
using a separate layer (**f**),
how useful the encoded representation
of **each input element is**
for generating the next element!

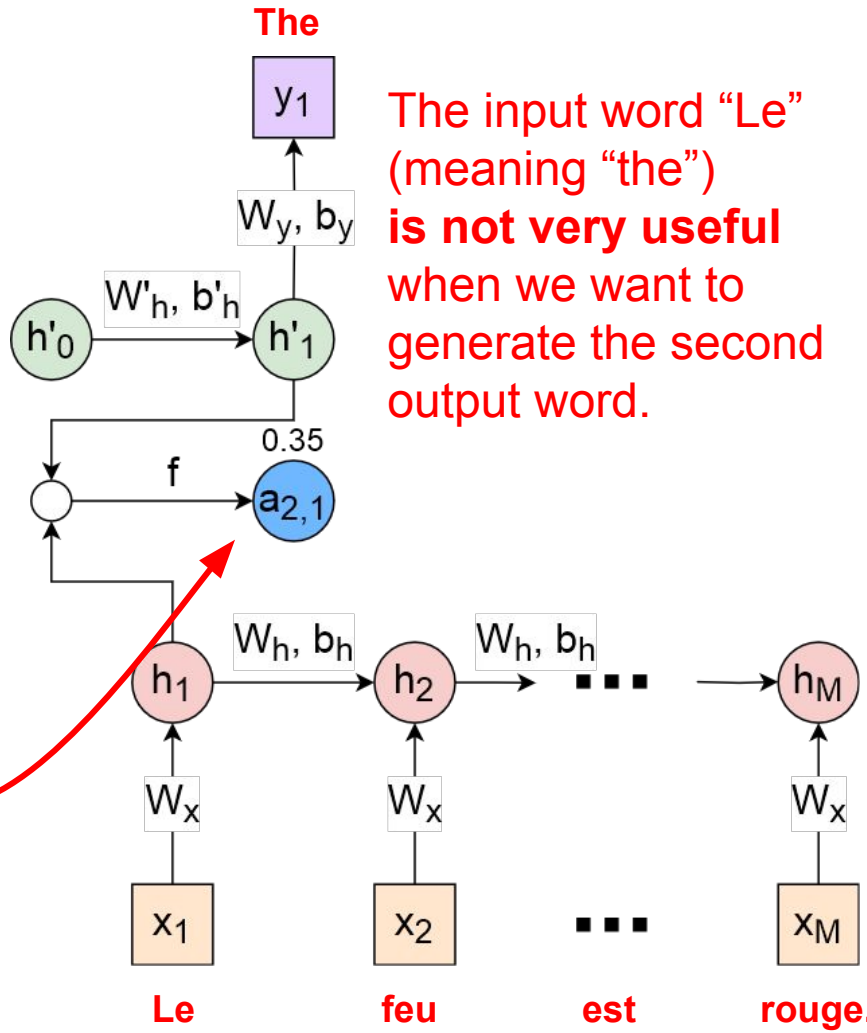
$$\forall_{t,t'} : a_{t,t'} := f([h'_{t-1}, h_{t'}])$$



“Seq2seq" attention

Let’s learn to estimate,
using a separate layer (**f**),
how useful the encoded representation
of **each input element is**
for generating the next element!

$$\forall_{t,t'} : a_{t,t'} := f([h'_{t-1}, h_{t'}])$$



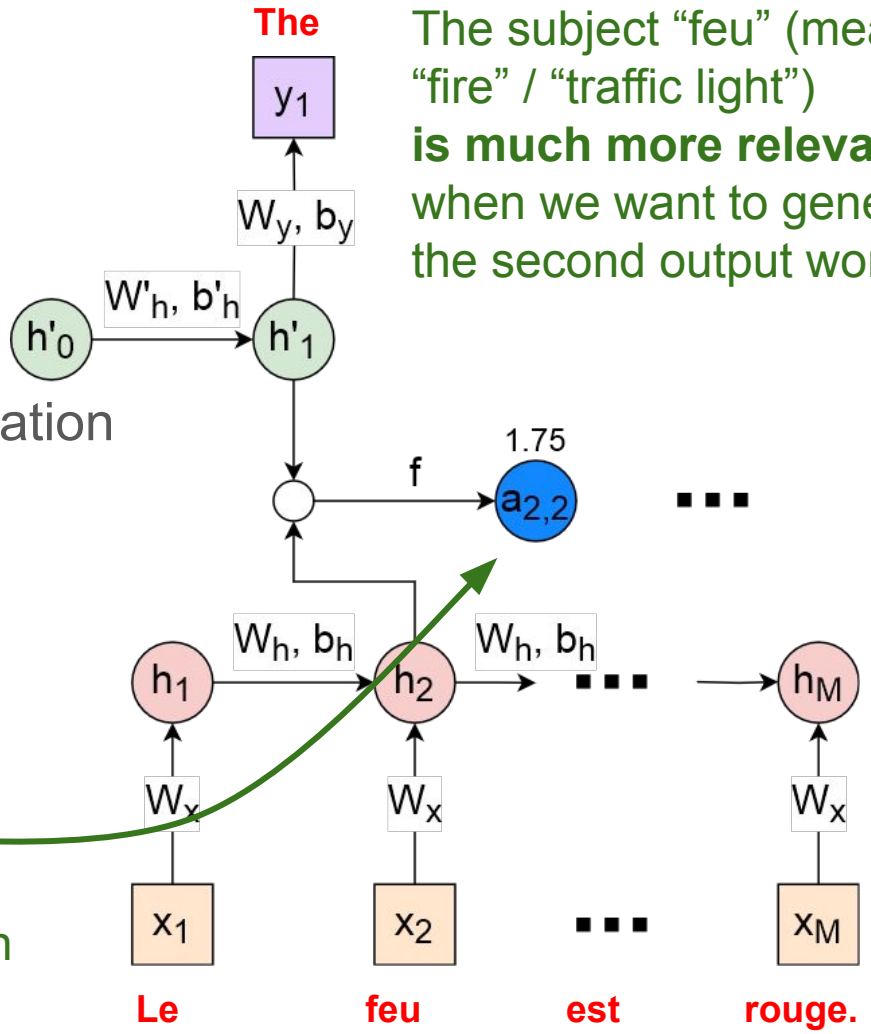
The input word “Le” (meaning “the”) **is not very useful** when we want to generate the second output word.

The estimated **attention score** between h'_1 and h_1 **is low**.

“Seq2seq" attention

Let’s learn to estimate,
using a separate layer (**f**),
how useful the encoded representation
of **each input element is**
for generating the next element!

$$\forall_{t,t'} : a_{t,t'} := f([h'_{t-1}, h_{t'}])$$



The subject “feu” (meaning “fire” / “traffic light”) **is much more relevant** when we want to generate the second output word.

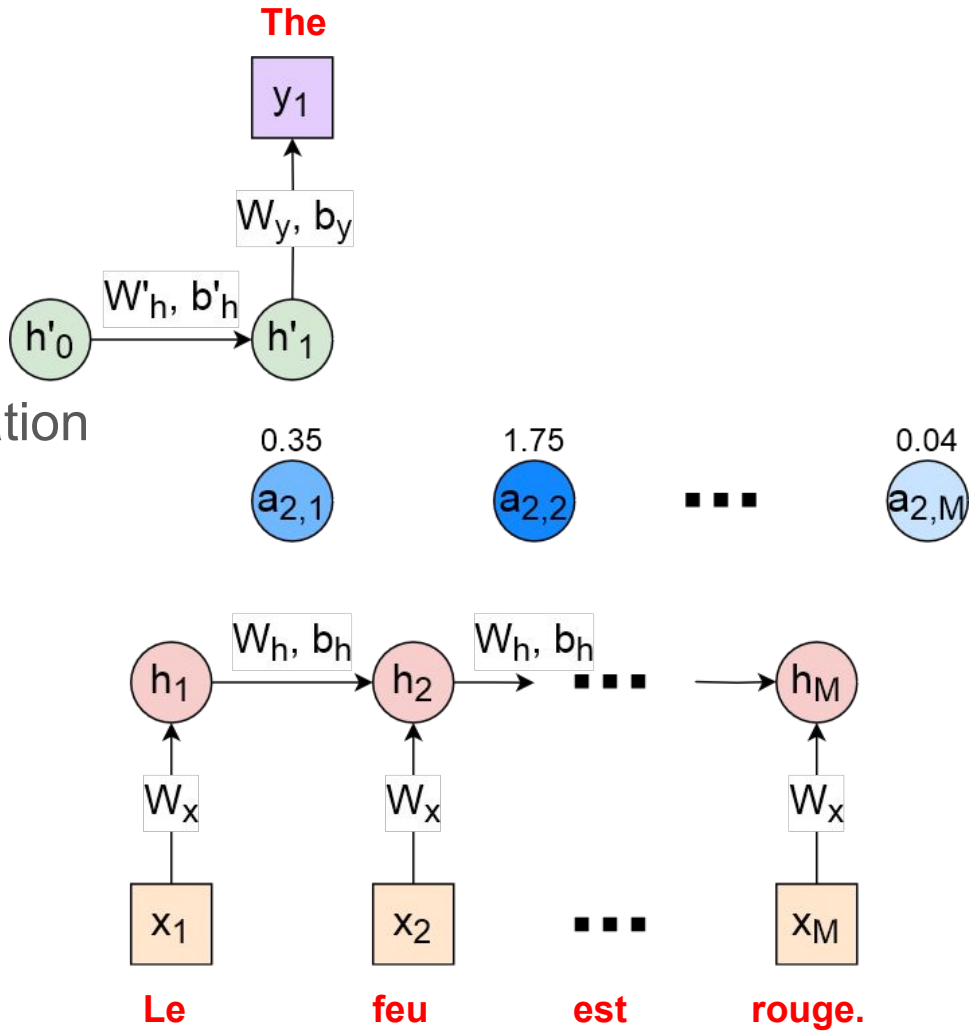
The estimated **attention score** between h'_1 and h_2 **is high**.

“Seq2seq" attention

Let’s learn to estimate,
using a separate layer (**f**),
how useful the encoded representation
of **each input element is**
for generating the next element!

$$\forall_{t,t'} : a_{t,t'} := f([h'_{t-1}, h_{t'}])$$

Compute the relevance
(the attention scores)
of each input representation!



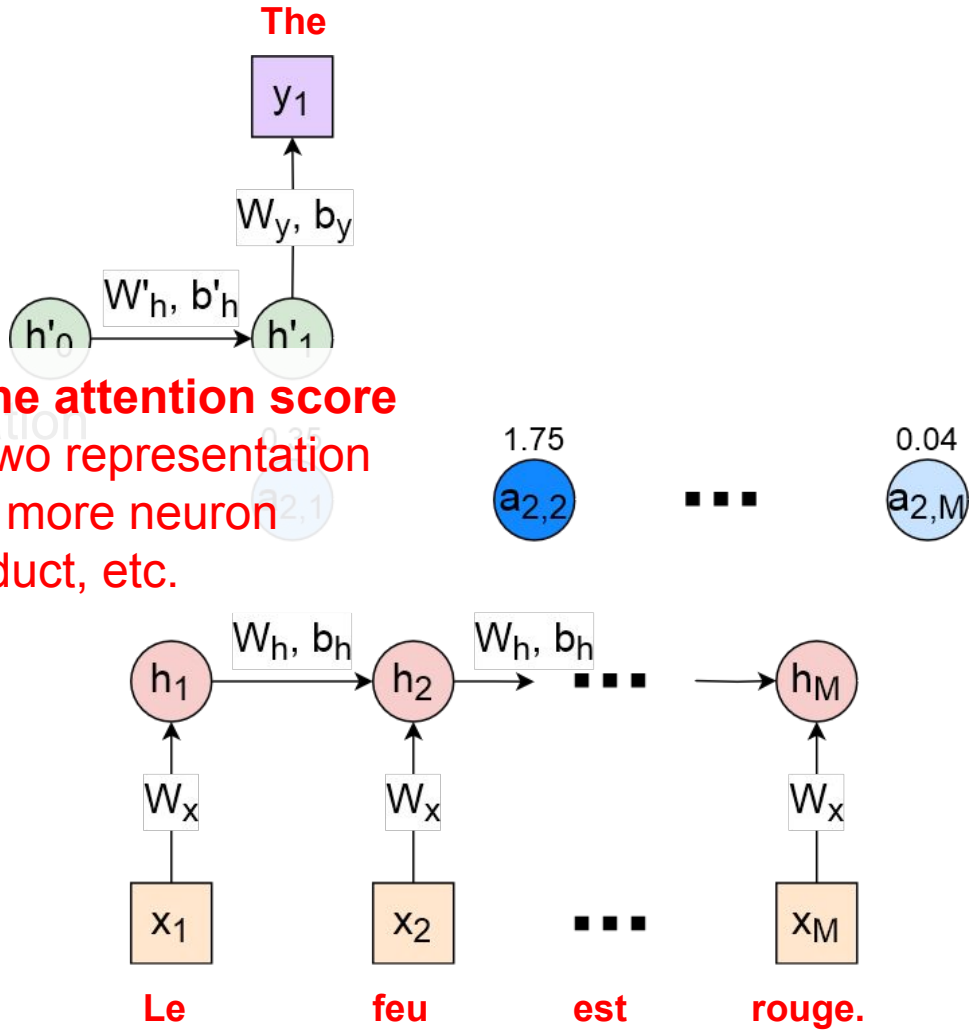
“Seq2seq" attention

Let’s learn to estimate,
using a separate layer (**f**),
how useful the encoded representation
of **each input** element is
for generating the current output

Function f estimates the attention score (a scalar) between the two representation vectors. **f** can be one or more neuron layers, a simple dot product, etc.

$$\forall_{t,t'} : a_{t,t'} := f([h'_{t-1}, h_{t'}])$$

Compute the relevance
(the attention scores)
of each input representation!

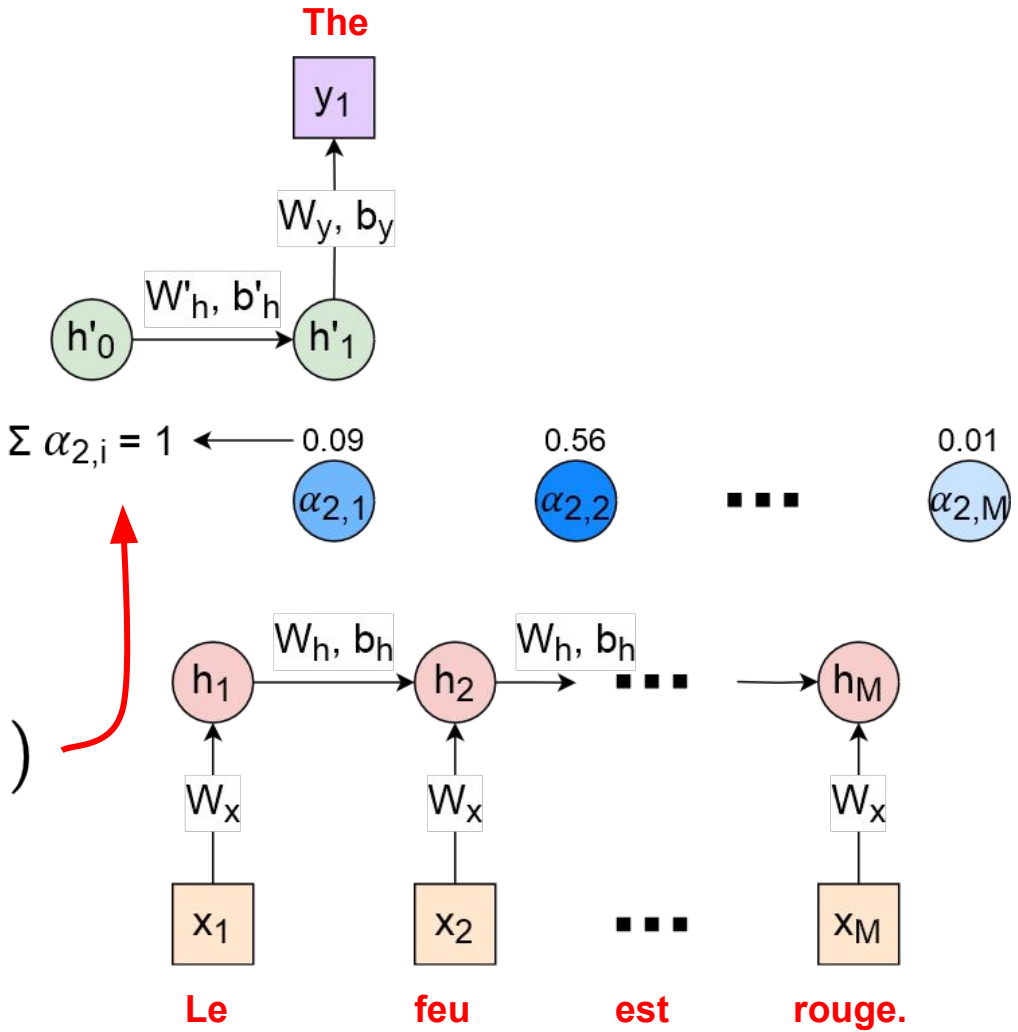


“Seq2seq" attention

Let’s **create** a **probability vector** (with a sum of 1) **from the attention scores using softmax**!

$$\forall_{t,t'} : a_{t,t'} := f([h'_{t-1}, h_{t'}])$$

$$\alpha_{t,t'} := \mathrm{softmax}_{t'}(a_{t,t'})$$



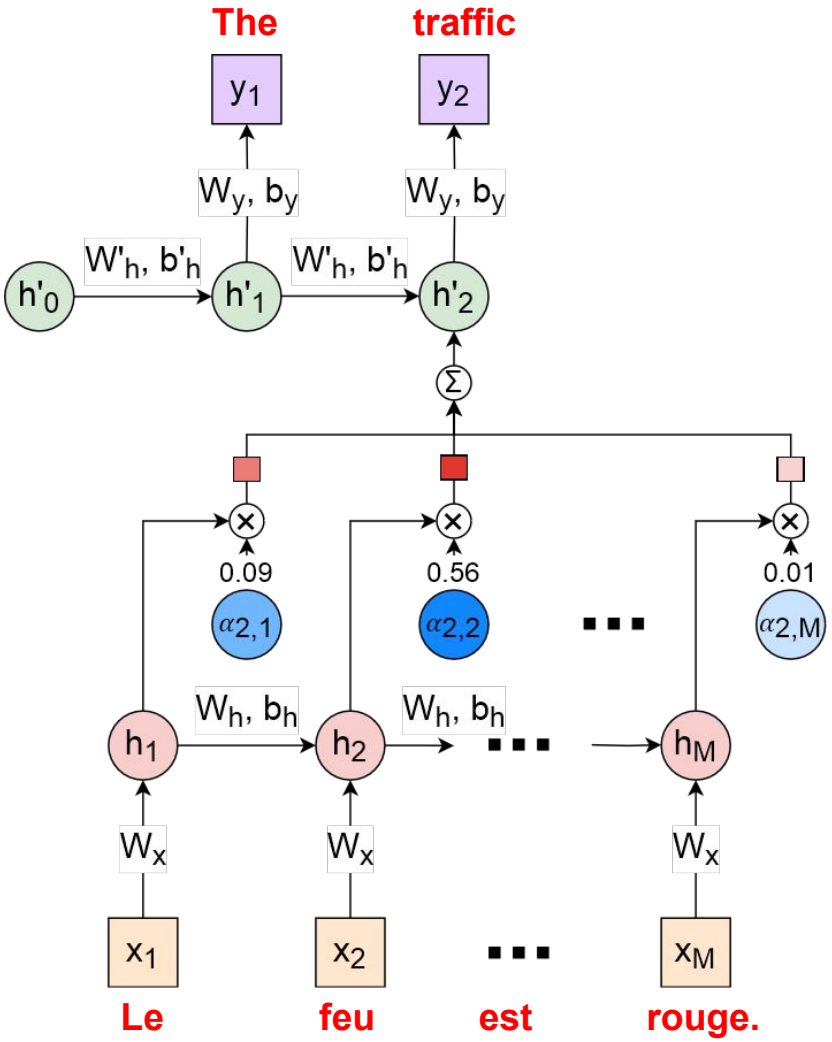
“Seq2seq" attention

We generate the **next decoder representation** using the **previous decoder representation**, as well as **the sum of all encoder representations weighted by the attention scores.**

$$\forall_{t,t'} : a_{t,t'} := f([h'_{t-1}, h_{t'}])$$

$$\alpha_{t,t'} := \text{softmax}_{t'}(a_{t,t'})$$

$$h'_t = g([h'_{t-1}, \sum_{t'} \alpha_{t,t'} \cdot h_{t'}])$$



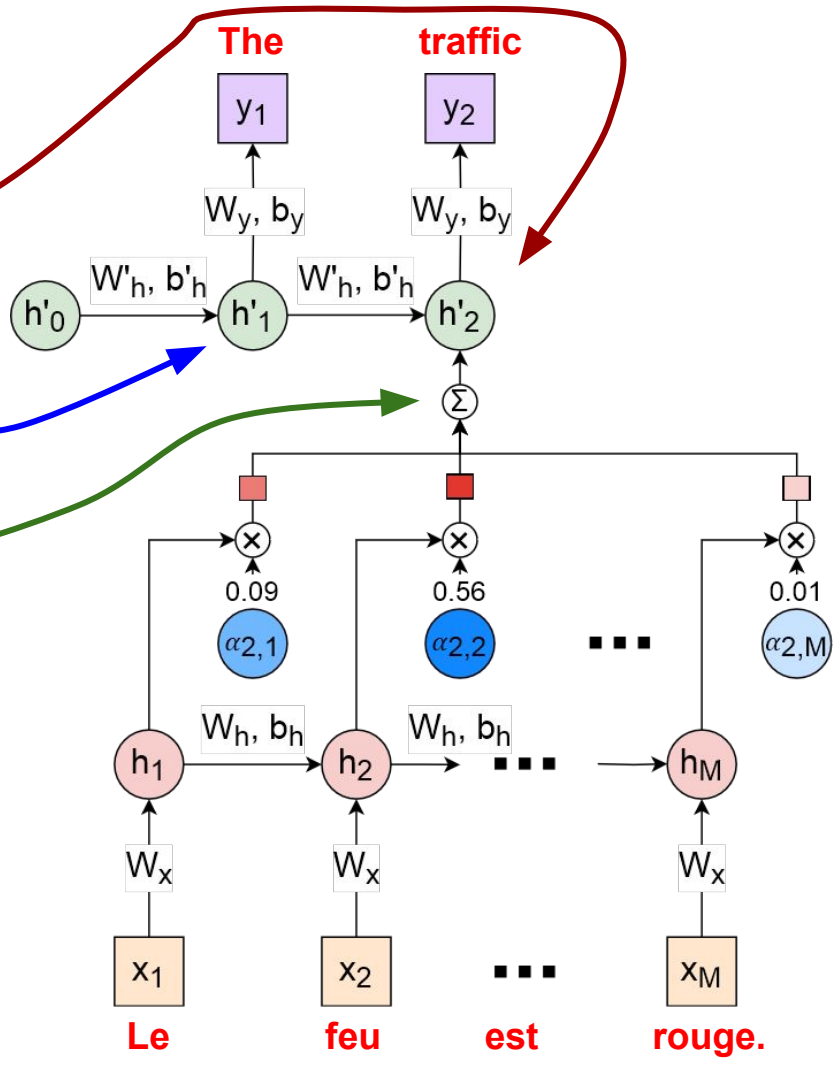
“Seq2seq" attention

We generate the **next decoder representation** using the **previous decoder representation**, as well as **the sum of all encoder representations weighted by the attention scores.**

$$\forall_{t,t'} : a_{t,t'} := f([h'_{t-1}, h_{t'}])$$

$$\alpha_{t,t'} := \mathrm{softmax}_{t'}(a_{t,t'})$$

$$h'_t = g([h'_{t-1}, \sum_{t'} \alpha_{t,t'} \cdot h_{t'}])$$



$$h'_2 := \sigma(W'_h h'_1 + b'_h + W_a(0.09 \cdot h_1 + 0.56 \cdot h_2 + \ldots + 0.01 \cdot h_M))$$

“Seq2seq" attention

We generate the **next decoder representation** using the **previous decoder representation**, as well as **the sum of all encoder representations weighted by the attention scores**

$$\forall_{t,t'} : a_{t,t'} := f([h'_{t-1}, h_{t'}])$$

$$\alpha_{t,t'} := \text{softmax}_{t'}(a_{t,t'})$$

$$h'_t = g([h'_{t-1}, \sum_{t'} \alpha_{t,t'} \cdot h_{t'}])$$

f and **g** are usually fully connected layers in Seq2seq attention.

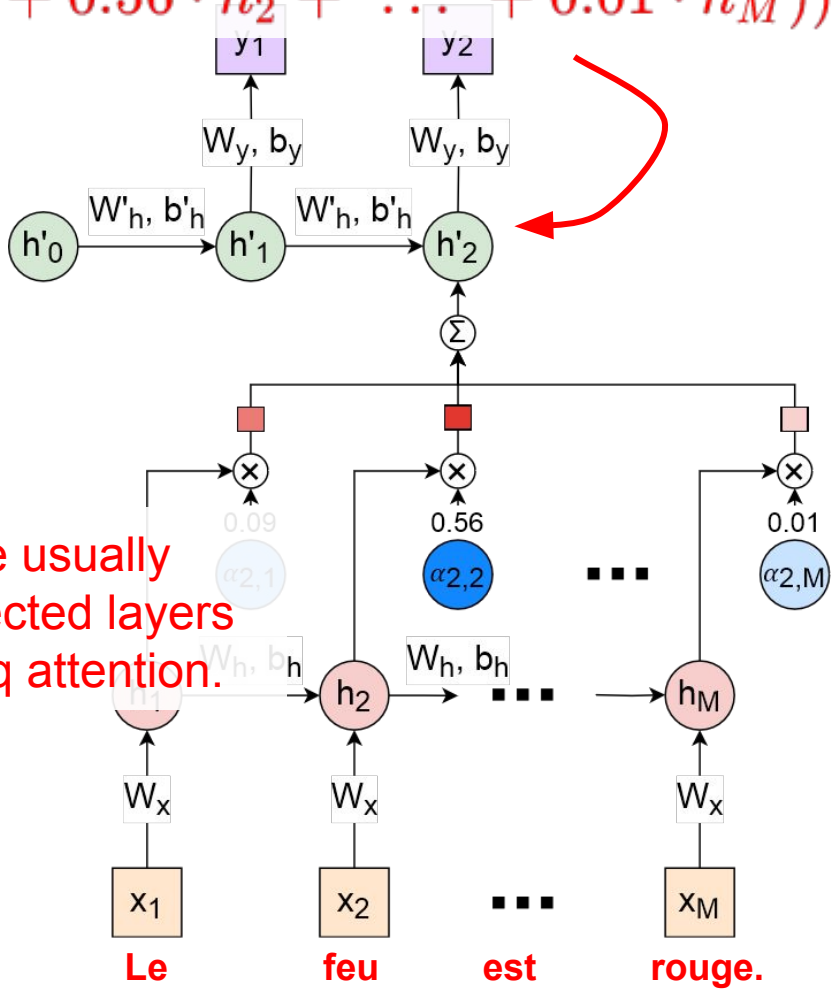


Image source: Bahdanau et al., 2015

Attention-based models - Example

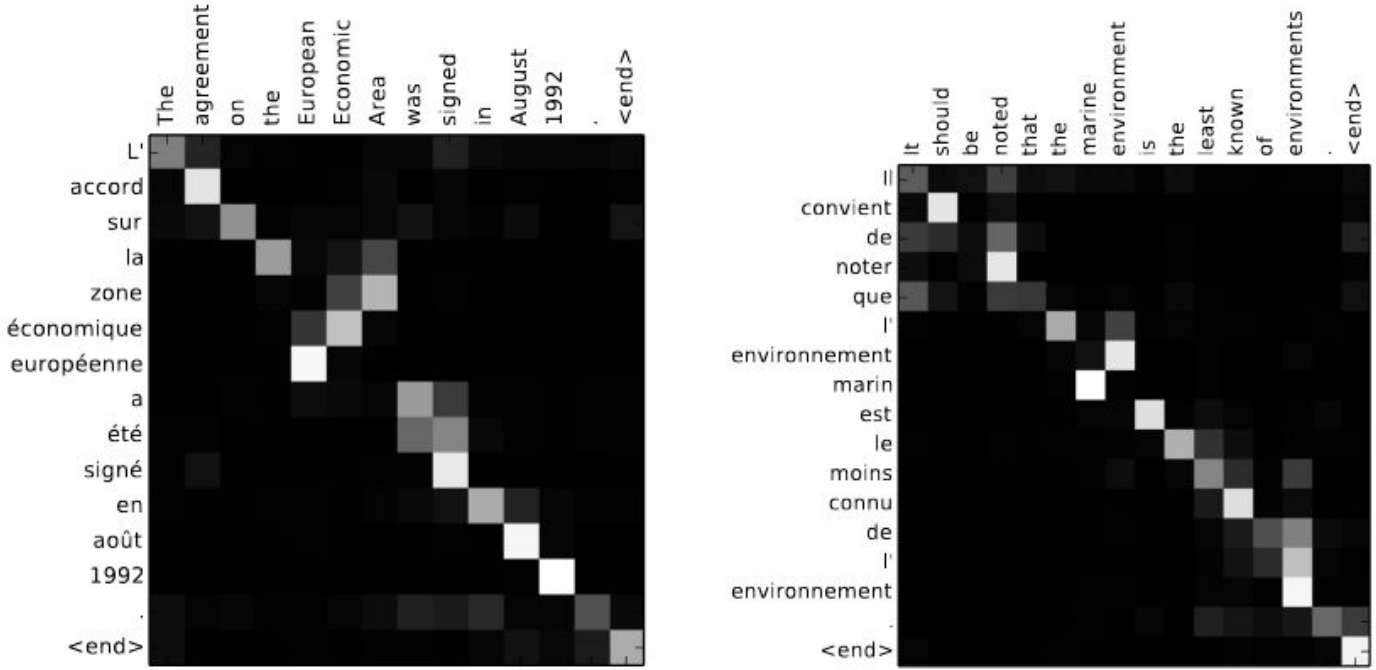
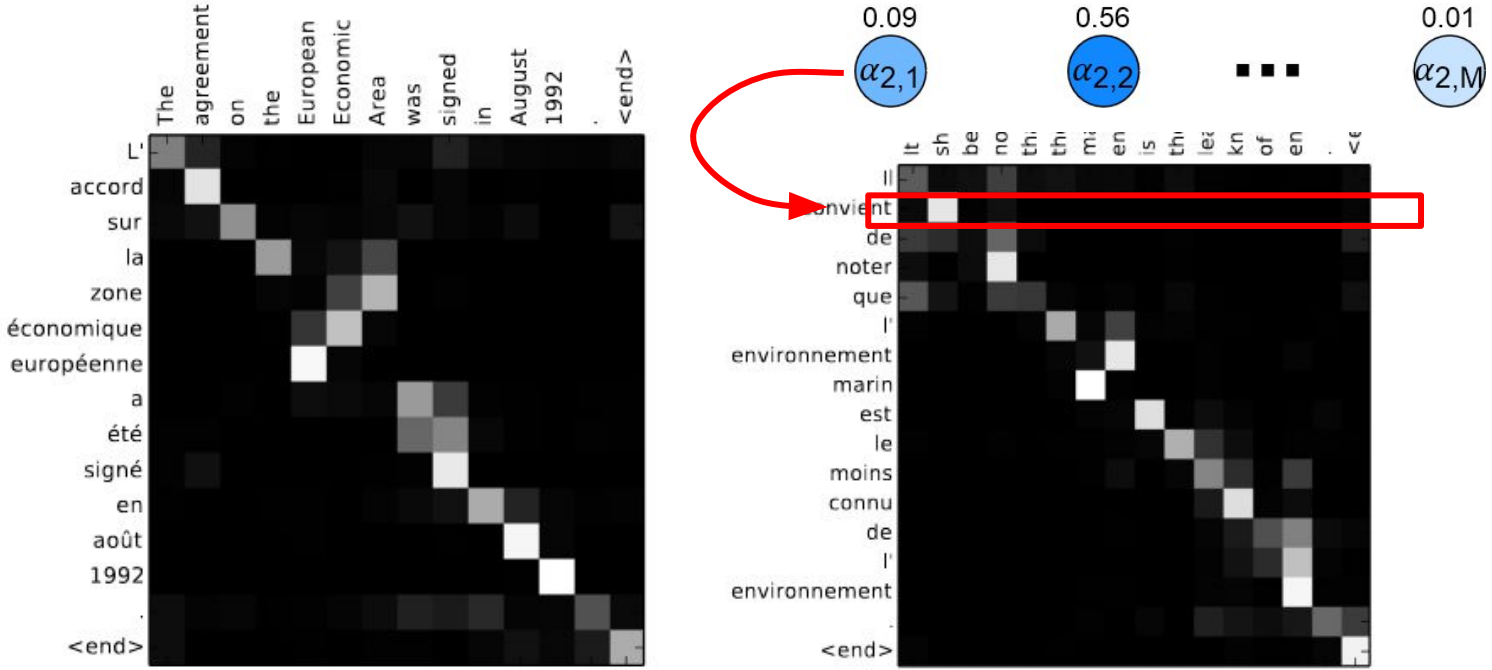


Image source: Bahdanau et al., 2015

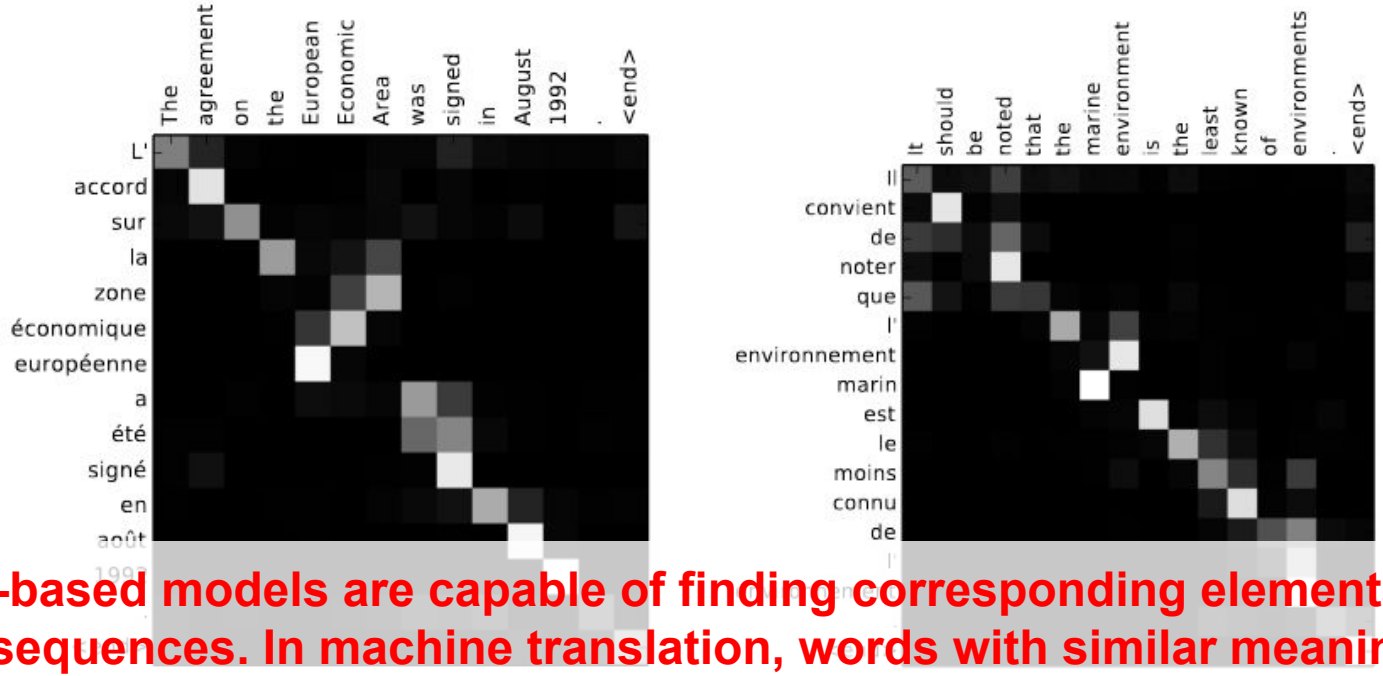
Attention-based models - Example



Element (i, j) of the matrix: The attention score of h'_{i-1} and h_j.
I.e., "how strongly are they related?"

Image source: Bahdanau et al., 2015

Attention-based models - Example



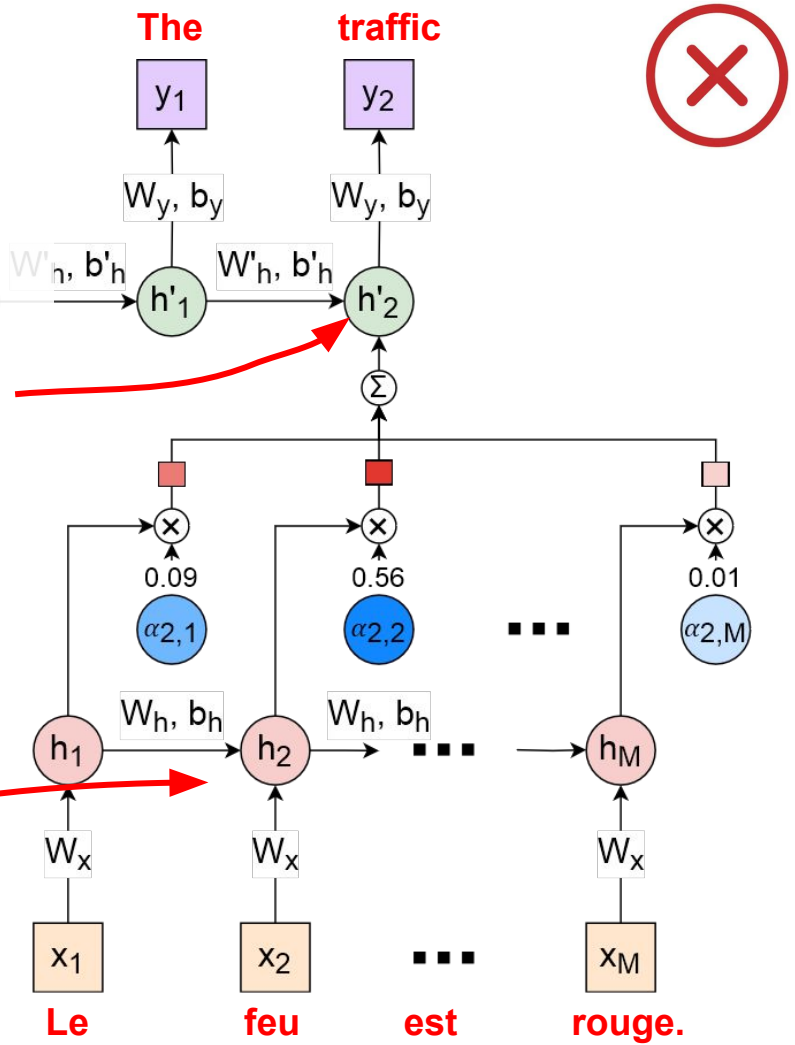
Attention-based models are capable of finding corresponding elements from different sequences. In machine translation, words with similar meanings in the input (e.g., an English sentence) **and the output** (e.g., a French sentence) **are expected to have a high attention score!**

“Seq2seq" attention

Optional: In addition to the two components mentioned before, the previous decoder **output** ($\mathbf{y}_{t-1}$) can also be used to estimate the next decoder representation
(as in the traditional seq2seq RNN model).

weighted by the attention scores.

Bidirectional RNN can also be used as an encoder: To properly assess the context, it is necessary to gather information from both sides...



“Seq2seq" attention

In the traditional seq2seq RNN model, the decoder had to be able to generate the entire output sequence from information stored in a single vector (**$h_M = h'_0$**).

In the attention-based seq2seq model, **the decoder can examine the entire sequence repeatedly**, focusing only on the parts of the encoded input sequence necessary for generating the next output.

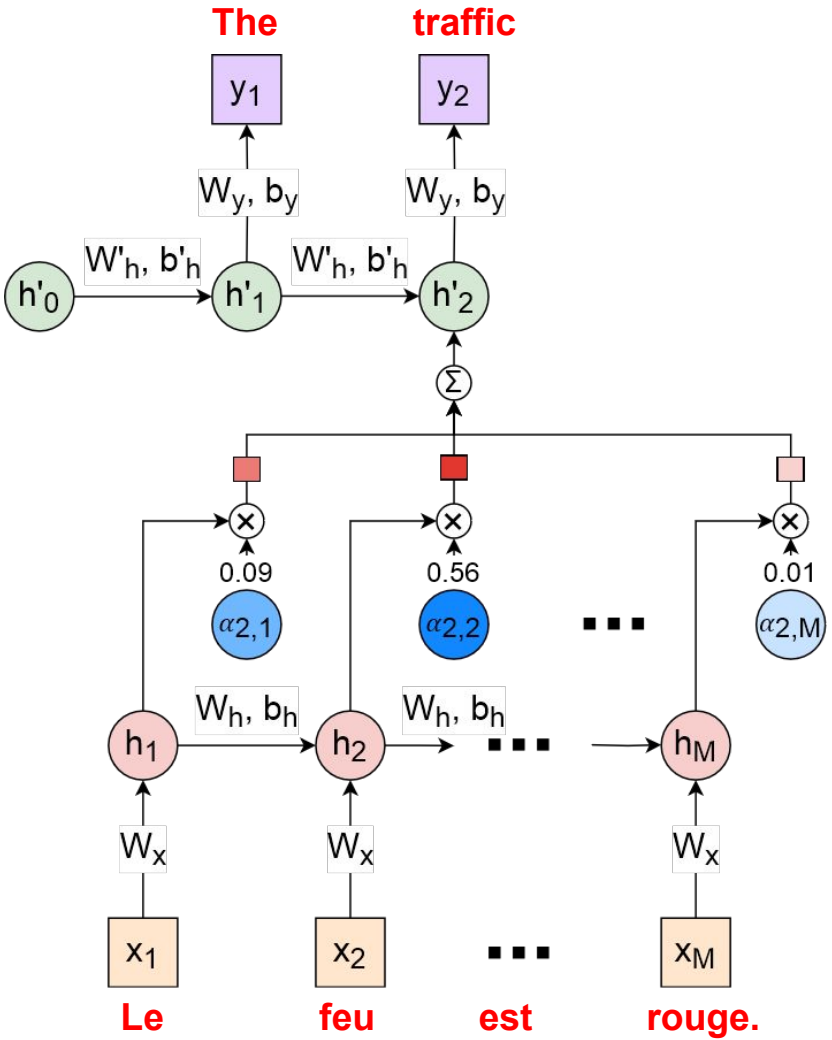
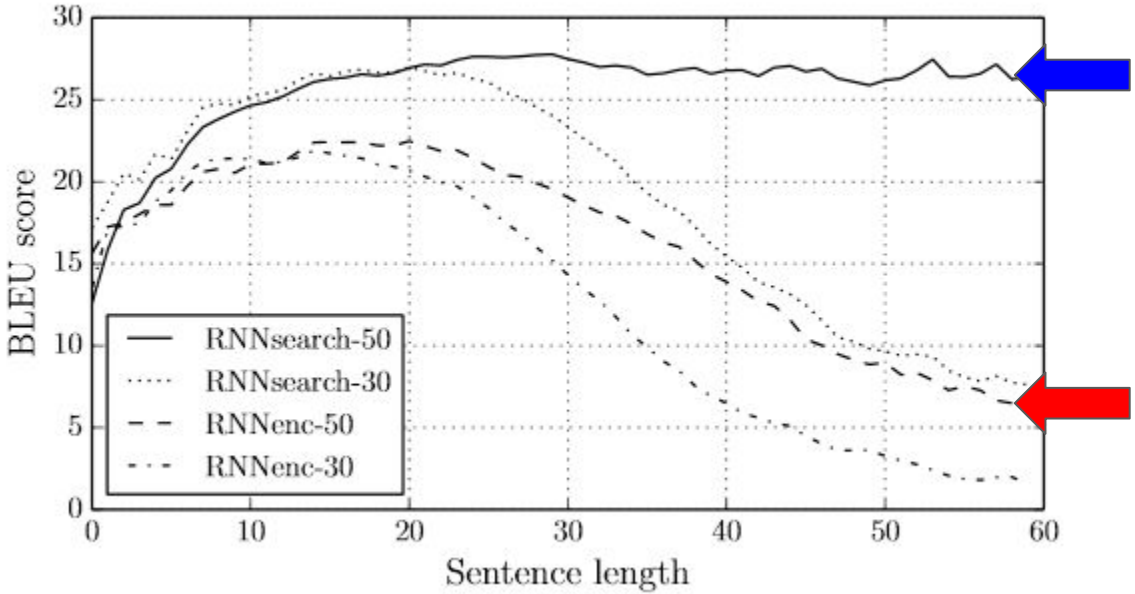


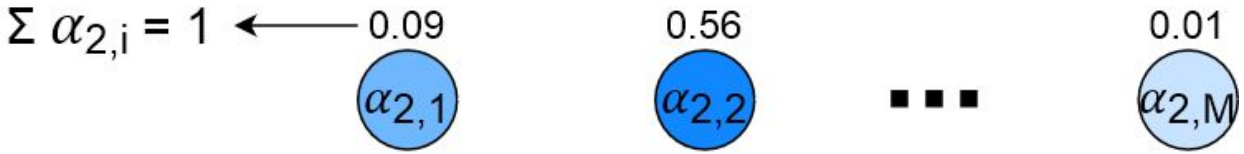
Image source: Bahdanau et al., 2015

Attention-based models



The attention-based model (RNNsearch) **outperforms the Seq2seq RNN** (RNNenc) **on long sequences.**

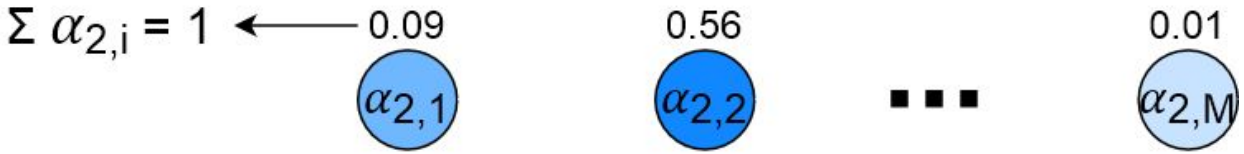
Attention-based models



Soft attention: The weighted sum of the encoder representation is taken

Hard attention: We choose the encoder representation with maximal score

Attention-based models



Soft attention: The weighted sum of the encoder representation is taken

Hard attention: We choose the encoder representation with maximal score
Cannot be used with gradient descent based learning.

Image source: Xu et al., 2015

Attention-based models

An example application: Image captioning using an attention-based model placed on top of a convolutional neural network.

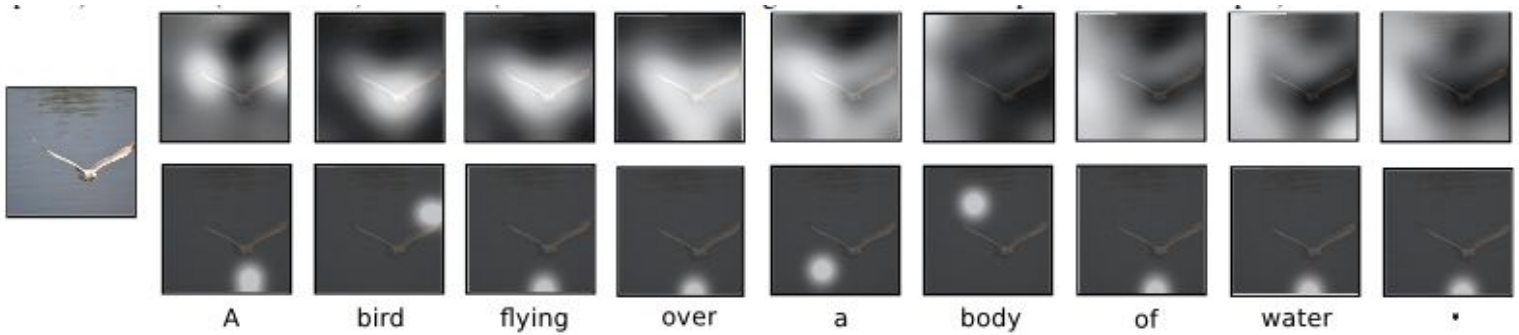
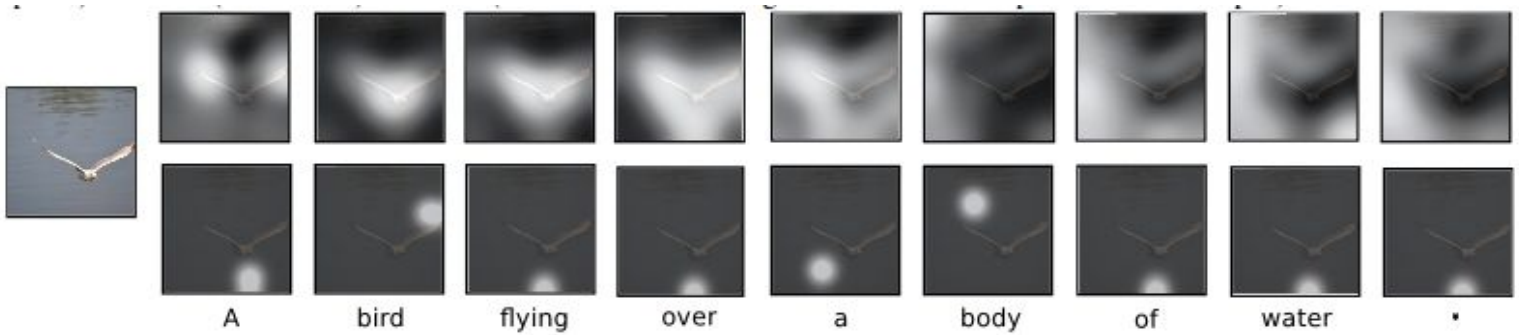


Image source: Xu et al., 2015

Attention-based models

An example application: Image captioning using an attention-based model placed on top of a convolutional neural network.



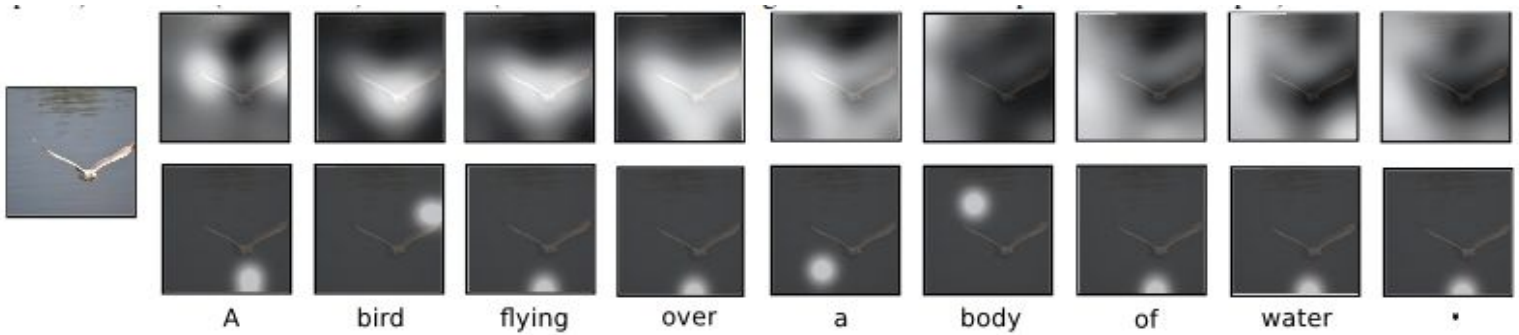
Attention-based model on top of a convolutional network:
The attention-based Seq2seq model focuses on those image patches that are needed to generate the next word of the caption.

Soft (top) **and hard** (bottom) **attention variants.**

Image source: Xu et al., 2015

Attention-based models

An example application: Image captioning using an attention-based model placed on top of a convolutional neural network.



Multimodality: The **encoder** processes visual representations (CNN output heatmaps), the **decoder** is responsible for learning to generate text, and the **attention mechanism** connects the two parts, aligning words with details in the heatmaps.

Image source: Xu et al., 2015

Attention-based models

An example application: Image captioning using an attention-based model placed on top of a convolutional neural network.

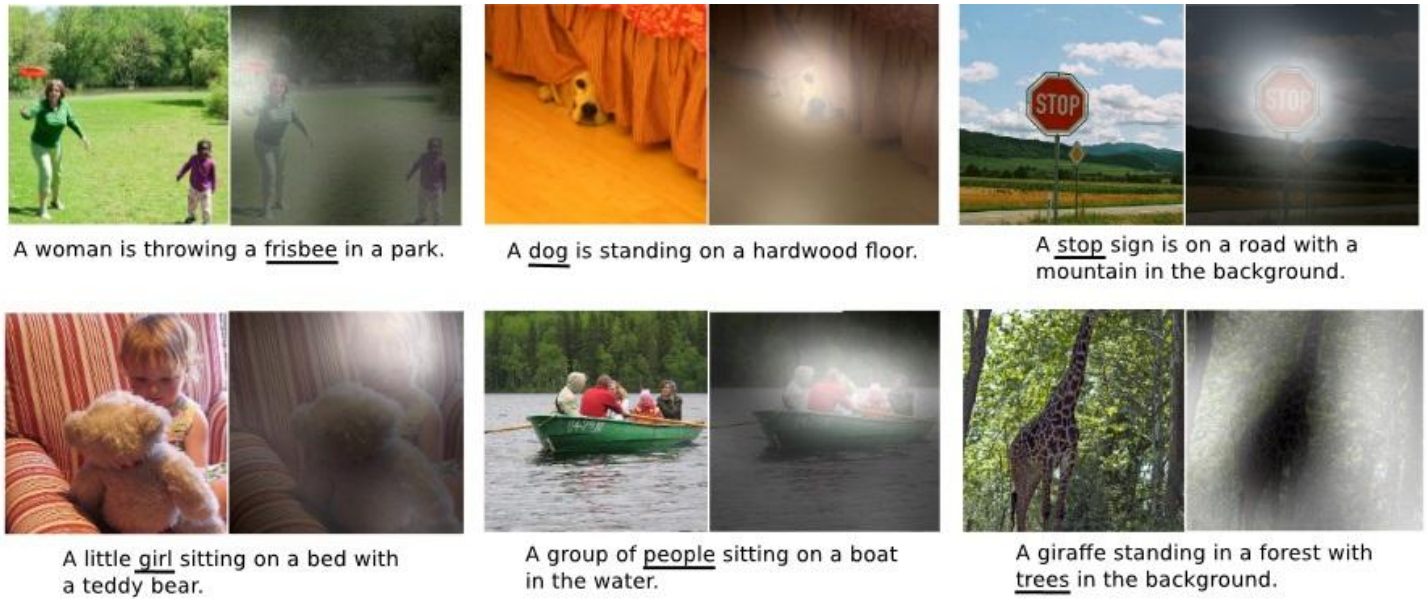
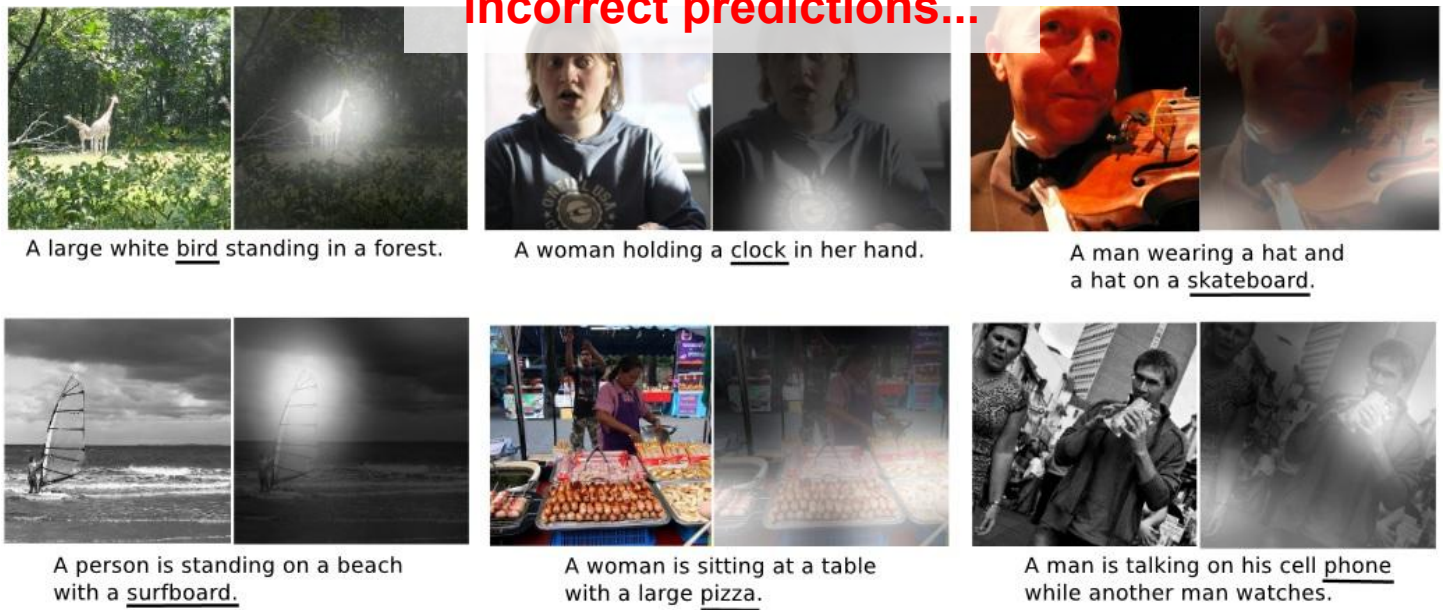


Image source: Xu et al., 2015

Attention-based models

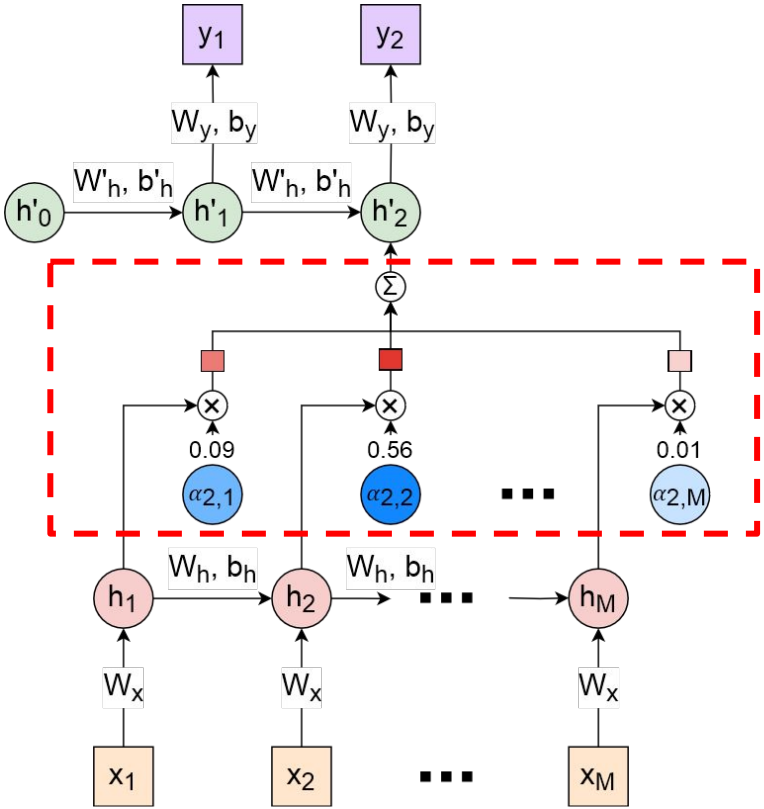
An example application: Image captioning using an attention-based model placed on top of a convolutional neural network.



Attention-based models

Attention-based Seq2seq model
(Bahdanau, 2015)

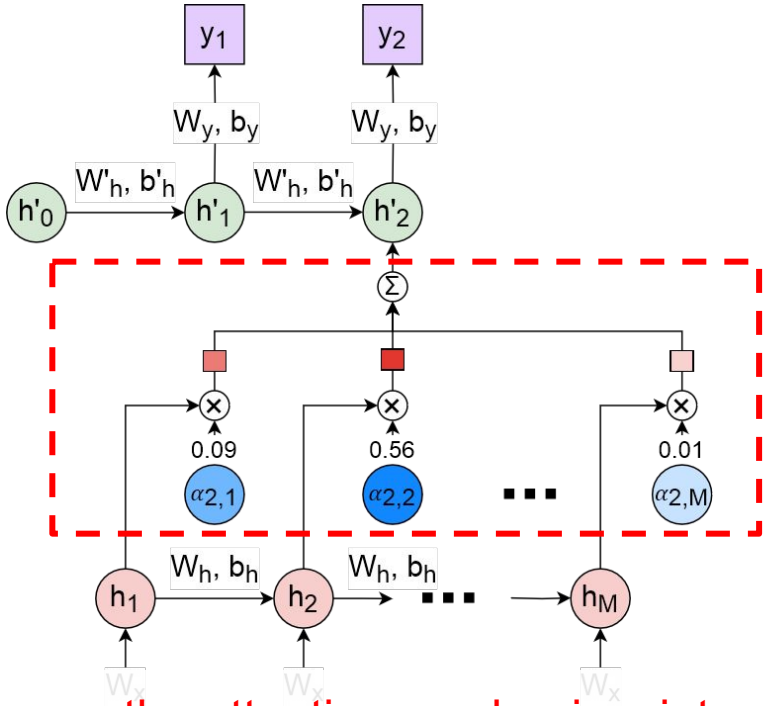
Limitation: The original attention mechanism **performs a single processing step** during which it searches for connections between elements in the input and output sequences.



Attention-based models

Attention-based Seq2seq model
(Bahdanau, 2015)

Limitation: The original attention mechanism **performs a single processing step** during which it searches for connections between elements in the input and output sequences.

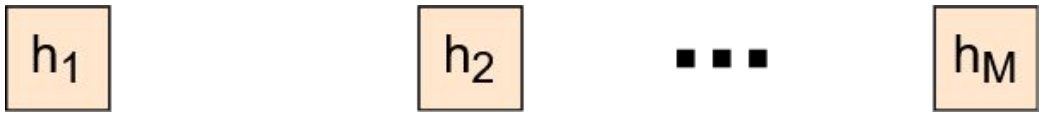


Idea: Let's wrap the attention mechanism into a standalone layer, so we can apply multiple of these in succession in a neural network!

Query, Key, Value

- For each element in the input sequence, **find those elements** in the same sequence **that are relevant to it.**
- Examine the attention scores of **all possible element pairs** in the input sequence!

Intuition: Logic similar to that of a web search engine…



Query, Key, Value

- For each element in the input sequence, **find those elements** in the same sequence **that are relevant to it.**
- Examine the attention scores of **all possible element pairs** in the input sequence!

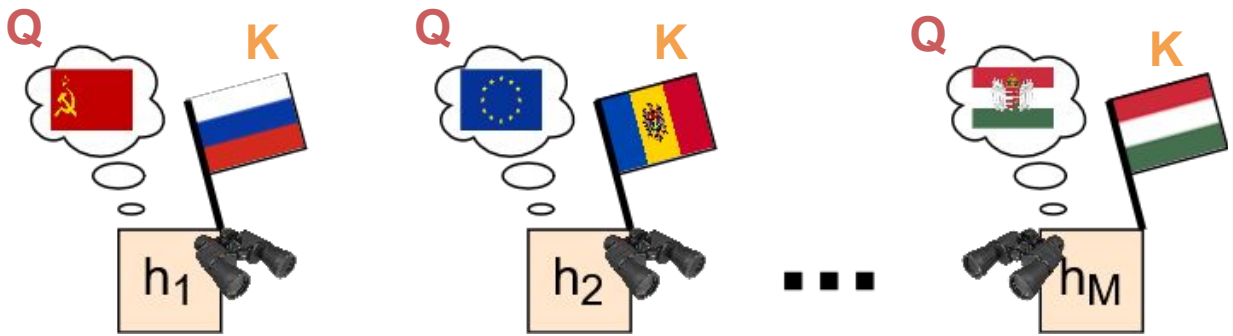
Intuition: Logic similar to that of a web search engine…

Query, Key, Value

Intuition: Web search engine, v1.0

- Representation of search keywords: **Query**
- Representation of keywords describing a website: **Key**

Compare all queries with all keys !



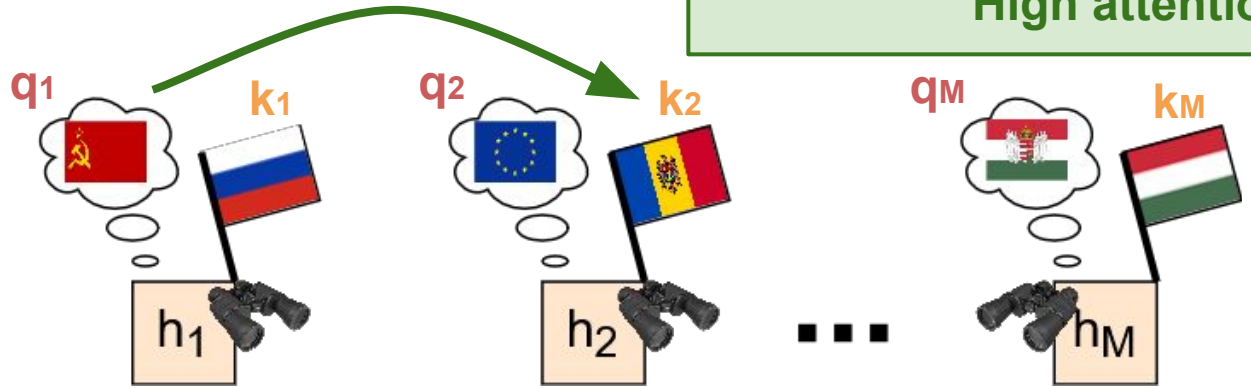
Query, Key, Value

Intuition: Web search engine, v1.0

- Representation of search keywords: **Query**
- Representation of keywords describing a website: **Key**

Compare all queries with all keys !

q_1 vs. k_2 → Good match!
High attention score



Attention scores are not symmetric…

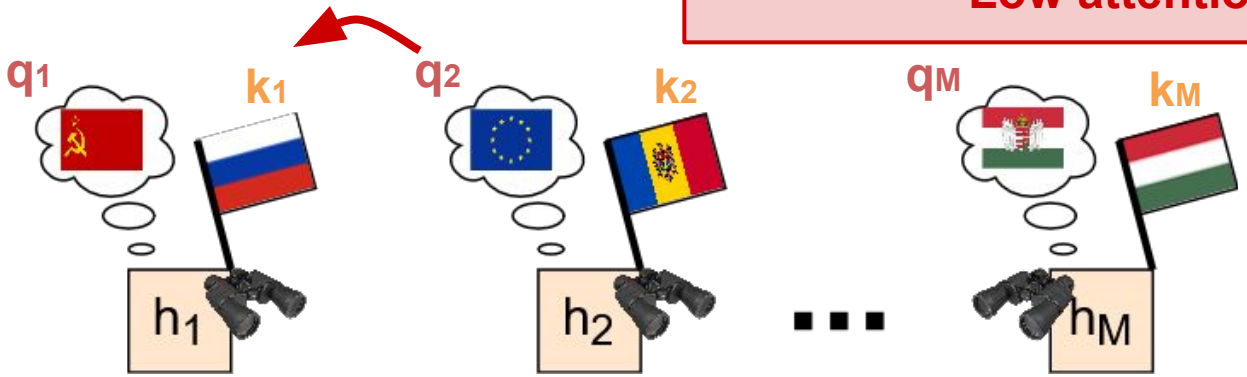
Query, Key, Value

Intuition: Web search engine, v1.0

- Representation of search keywords: **Query**
- Representation of keywords describing a website: **Key**

Compare all queries with all keys !

q_2 **vs.** k_1 → **Not so good match…**
Low attention score



Also checking match with self…

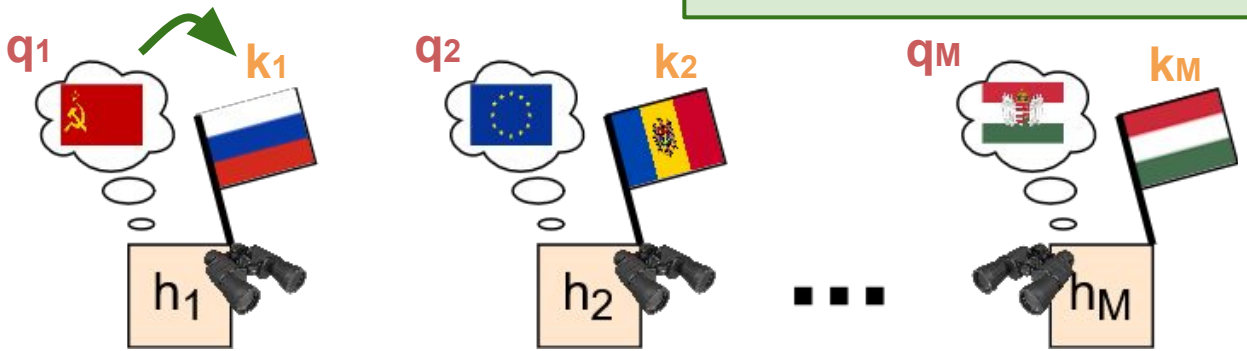
Query, Key, Value

Intuition: Web search engine, v1.0

- Representation of search keywords: **Query**
- Representation of keywords describing a website: **Key**

Compare all queries with all keys !

q_1 **vs.** k_1 → **Good match!**
High attention score



Query, Key, Value

Intuition: Web search engine, v1.0

- Representation of search keywords: **Query**
- Representation of keywords describing a website: **Key**

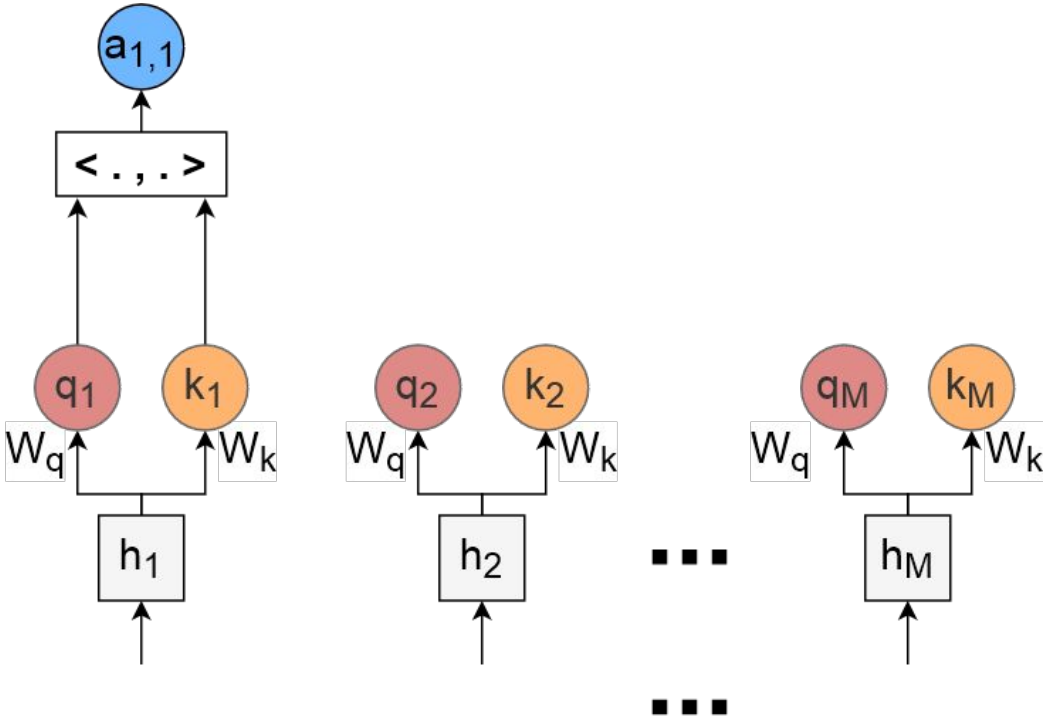
Compare all queries with all keys !

$$q_i = W_q h_i$$

$$k_i = W_k h_i$$

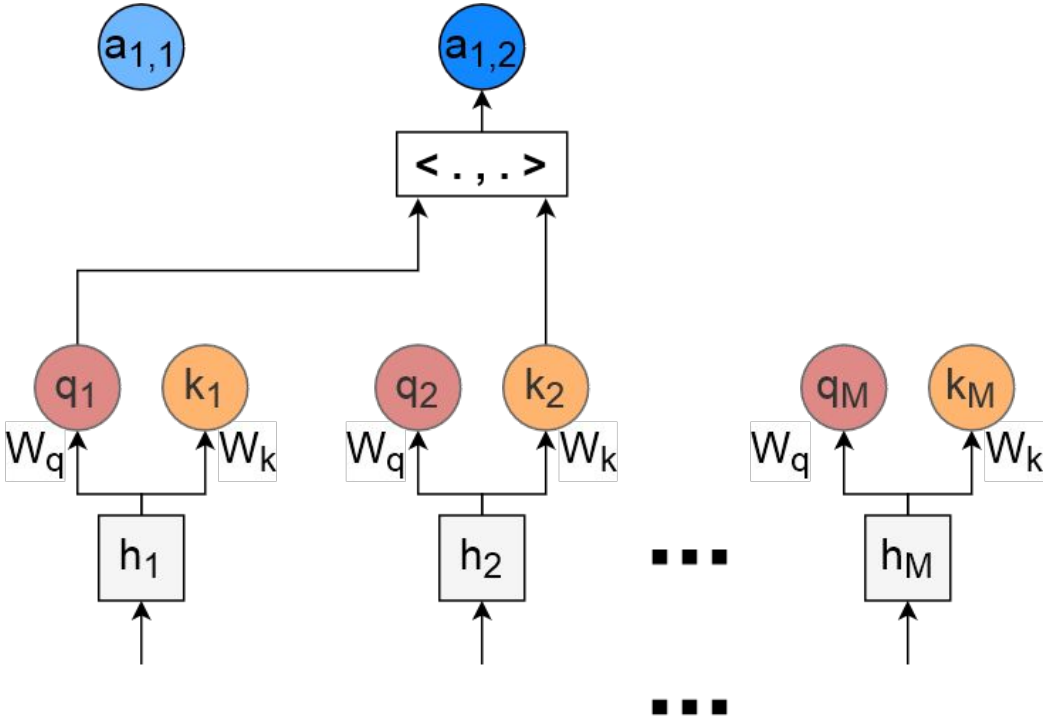
Self-attention layer *(**incomplete**)*

$$q_i = W_q h_i$$
$$k_i = W_k h_i$$
$$a_{i,j} = \langle q_i, k_j \rangle$$



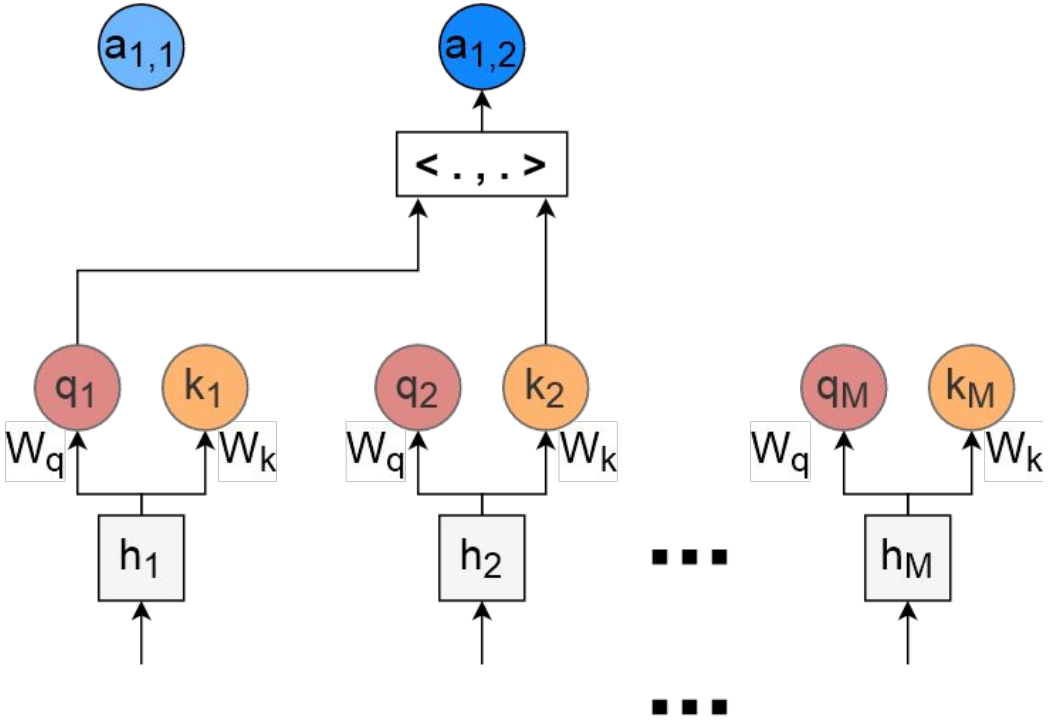
Self-attention layer *(**incomplete**)*

$$q_i = W_q h_i$$
$$k_i = W_k h_i$$
$$a_{i,j} = \langle q_i, k_j \rangle$$



Self-attention layer *(**incomplete**)*

$$q_i = W_q h_i$$
$$k_i = W_k h_i$$
$$a_{i,j} = \langle q_i, k_j \rangle$$



Query, Key, Value

Intuition: Web search engine, v1.0

- Representation of search keywords: **Query**
- Representation of keywords describing a website: **Key**

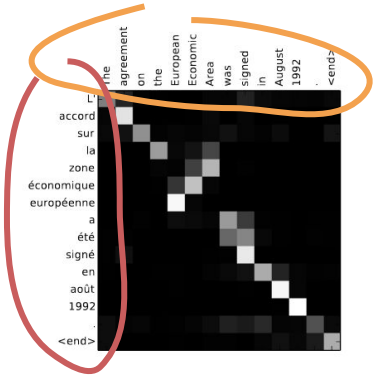
$$q_i = W_q h_i$$

$$k_i = W_k h_i$$

Compare all queries with all keys !

$$a_{i,j} = \langle q_i, k_j \rangle$$

The **attention score** between the **query** and the **key** is given by the **dot product** of the two vectors.



Query, Key, Value

Intuition: Web search engine, v1.0

- Representation of search keywords: **Query**
- Representation of keywords describing a website: **Key**

$$q_i = W_q h_i$$

$$k_i = W_k h_i$$

Compare all queries with all keys !

$$a_{i,j} = \langle q_i, k_j \rangle$$

Finally, the sum of the **keys** **weighted** by the normalized attention scores yields the individual outputs:

$$\alpha_{i,j} = \text{softmax}_i\left(\frac{a_{i,j}}{\sqrt{d_q}}\right) \qquad \hat{y}_i = \sum_{j=1}^{M} \alpha_{i,j} \cdot k_j$$

Query, Key, Value

Intuition: Web search engine, v1.0

- Representation of search keywords: **Query**
- Representation of keywords describing a website: **Key**

$$q_i = W_q h_i$$

Compare all queries with all keys:

$$a_{i,j} = \langle q_i, k_j \rangle$$

"Scaled dot-product attention**":** $\mathbf{d_q}$ is the length of vectors **q** and **k**. The greater $\mathbf{d_q}$ is, the larger the dot product is expected to be → Softmax with large inputs is not ideal during training: derivative is getting very close to zero (similarly to sigmoid).

Finally, the sum of the **keys** **weighted** by the normalized attention scores yields the individual outputs:

$$\alpha_{i,j} = \text{softmax}_i\left(\frac{a_{i,j}}{\sqrt{d_q}}\right) \qquad \hat{y}_i = \sum_{j=1}^{M} \alpha_{i,j} \cdot k_j$$

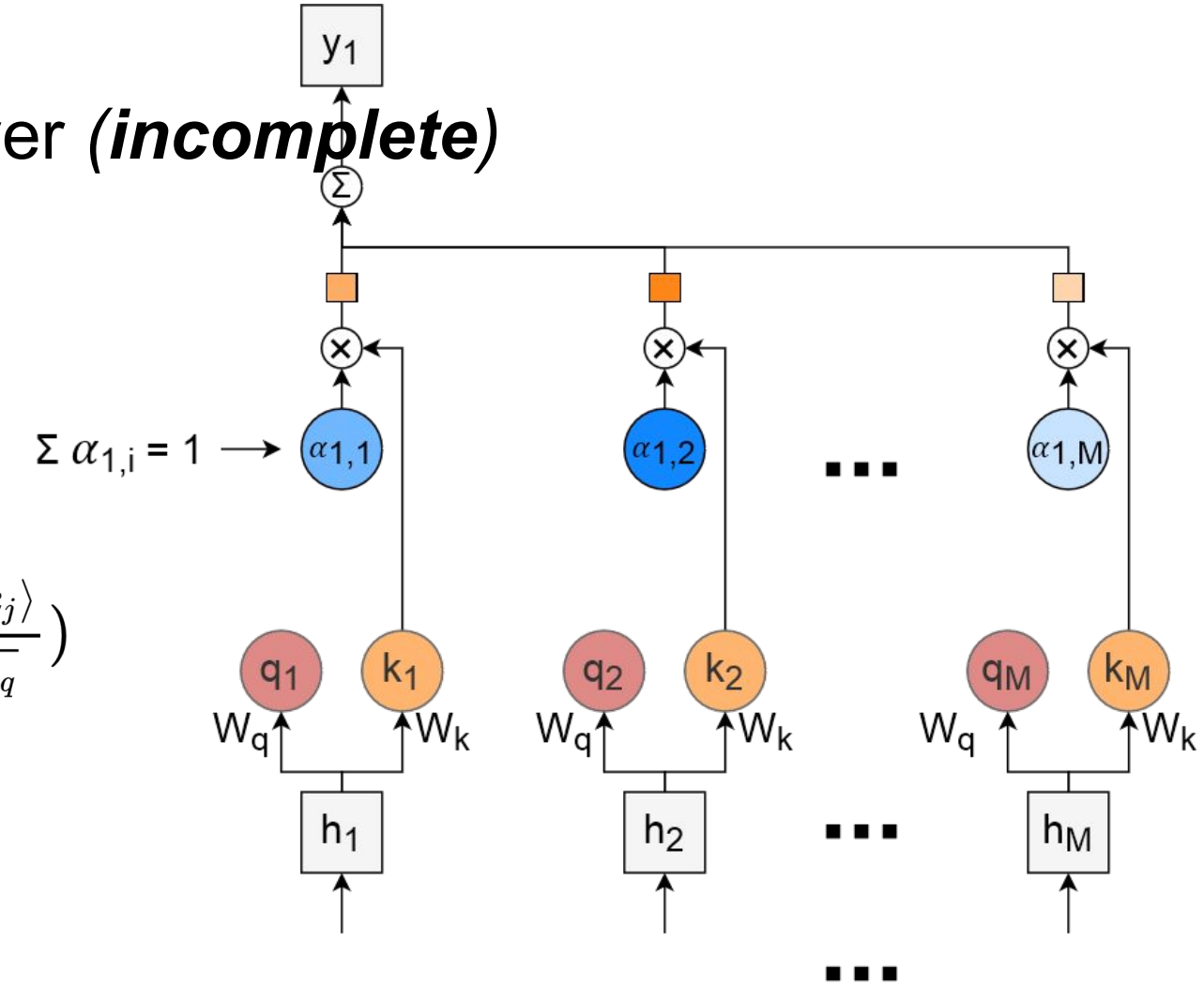
Self-attention layer *(**incomplete**)*

$$q_i = W_q h_i$$

$$k_i = W_k h_i$$

$$\alpha_{i,j} = \mathrm{softmax}_i\left(\frac{\langle q_i, k_j \rangle}{\sqrt{d_q}}\right)$$

$$\hat{y}_i = \sum_{j=1}^{M} \alpha_{i,j} \cdot k_j$$



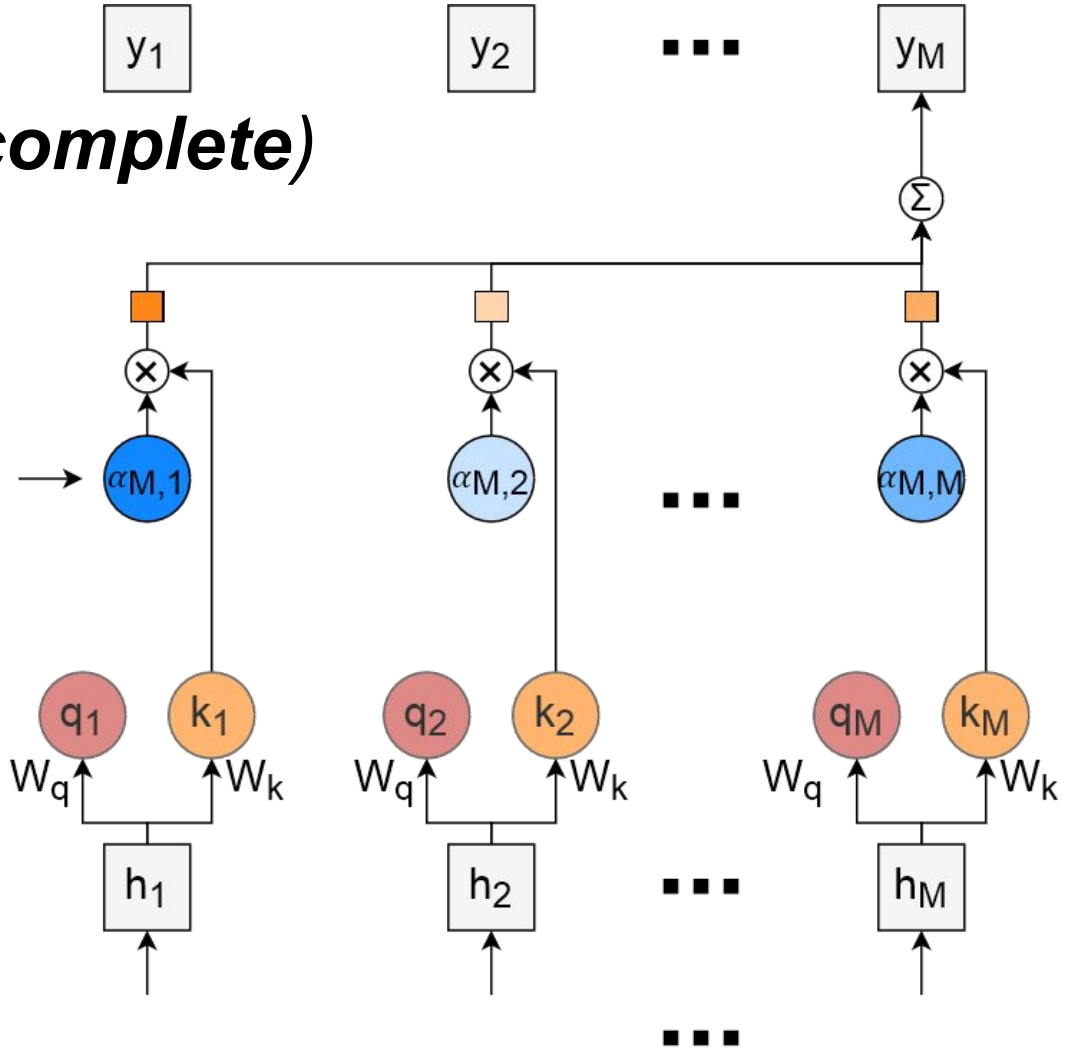
Self-attention layer *(**incomplete**)*

$$q_i = W_q h_i$$

$$k_i = W_k h_i$$

$$\alpha_{i,j} = \text{softmax}_i\left(\frac{\langle q_i, k_j \rangle}{\sqrt{d_q}}\right)$$

$$\hat{y}_i = \sum_{j=1}^{M} \alpha_{i,j} \cdot k_j$$



Self-attention layer (***inco***mplete)

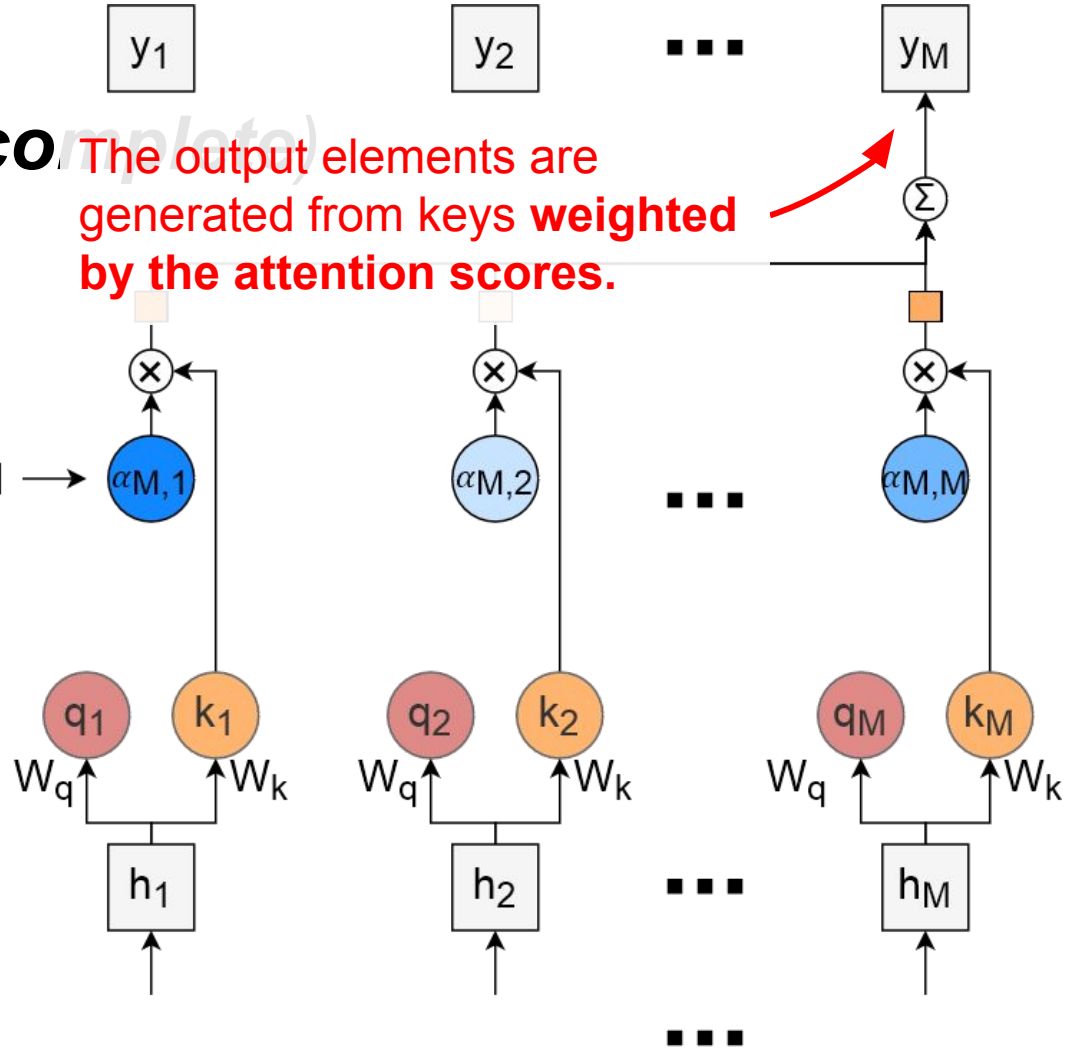
$$q_i = W_q h_i$$

$$k_i = W_k h_i$$

$$\alpha_{i,j} = \text{softmax}_i\left(\frac{\langle q_i, k_j \rangle}{\sqrt{d_q}}\right)$$

$$\hat{y}_i = \sum_{j=1}^{M} \alpha_{i,j} \cdot k_j$$

The output elements are generated from keys **weighted by the attention scores.**



Query, Key, Value

Intuition: Web search engine, v1.0

- Representation of search keywords: **Query**
- Representation of keywords describing a website: **Key**

$$q_i = W_q h_i$$

$$k_i = W_k h_i$$

Compare all queries with all keys !

$$a_{i,j} = \langle q_i, k_j \rangle$$

Why do we return keys?
Why not something else?
(E.g., the records corresponding to the key(word)s?)

Finally, the sum of the **keys** weighted by the normalized attention scores yields the individual outputs:

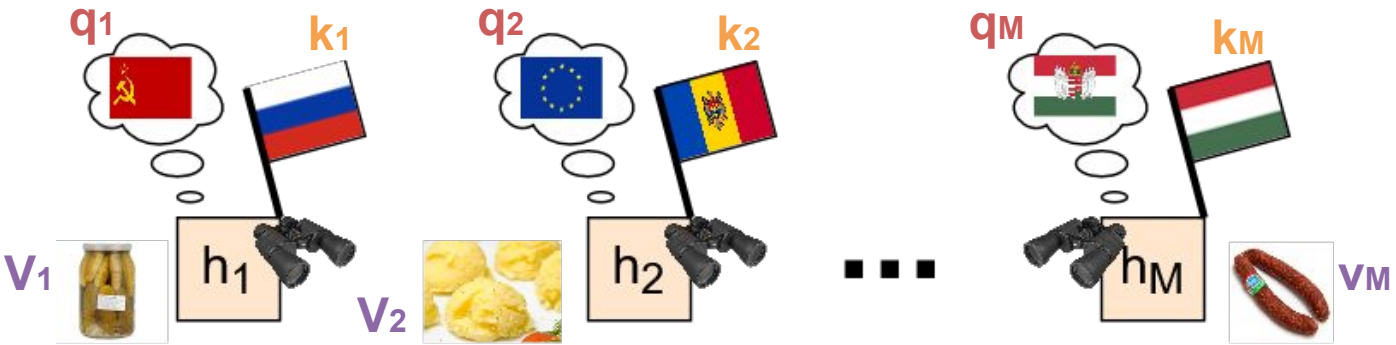
$$\alpha_{i,j} = \text{softmax}_i\left(\frac{a_{i,j}}{\sqrt{d_q}}\right) \qquad \hat{y}_i = \sum_{j=1}^{M} \alpha_{i,j} \cdot k_j$$

Query, Key, Value

$$q_i = W_q h_i$$
$$k_i = W_k h_i$$
$$v_i = W_v h_i$$

Intuition: Web search engine, v2.0

- Representation of search keywords: **Query**
- Representation of keywords describing a website: **Key**
- Representation of the content of websites: **Value**



Query, Key, Value

$$q_i = W_q h_i$$
$$k_i = W_k h_i$$
$$v_i = W_v h_i$$

Intuition: Web search engine, v2.0

- Representation of search keywords: **Query**
- Representation of keywords describing a website: **Key**
- Representation of the content of websites: **Value**

Compare all queries with all keys !

→ Attention scores.

$$a_{i,j} = \langle q_i, k_j \rangle$$

Finally, the sum of the **values** **weighted** by the normalized attention scores yields the individual outputs:

$$\alpha_{i,j} = \mathrm{softmax}_i\left(\frac{a_{i,j}}{\sqrt{d_q}}\right) \qquad \hat{y}_i = \sum_{j=1}^{M} \alpha_{i,j} \cdot v_j$$

Query, Key, Value

$$q_i = W_q h_i$$
$$k_i = W_k h_i$$
$$v_i = W_v h_i$$

Intuition: Web search engine, v2.0

- Representation of search keywords: **Query**
- Representation of keywords describing a website: **Key**
- Representation of the content of websites: **Value**

Compare all queries with all keys !

→ Attention scores.

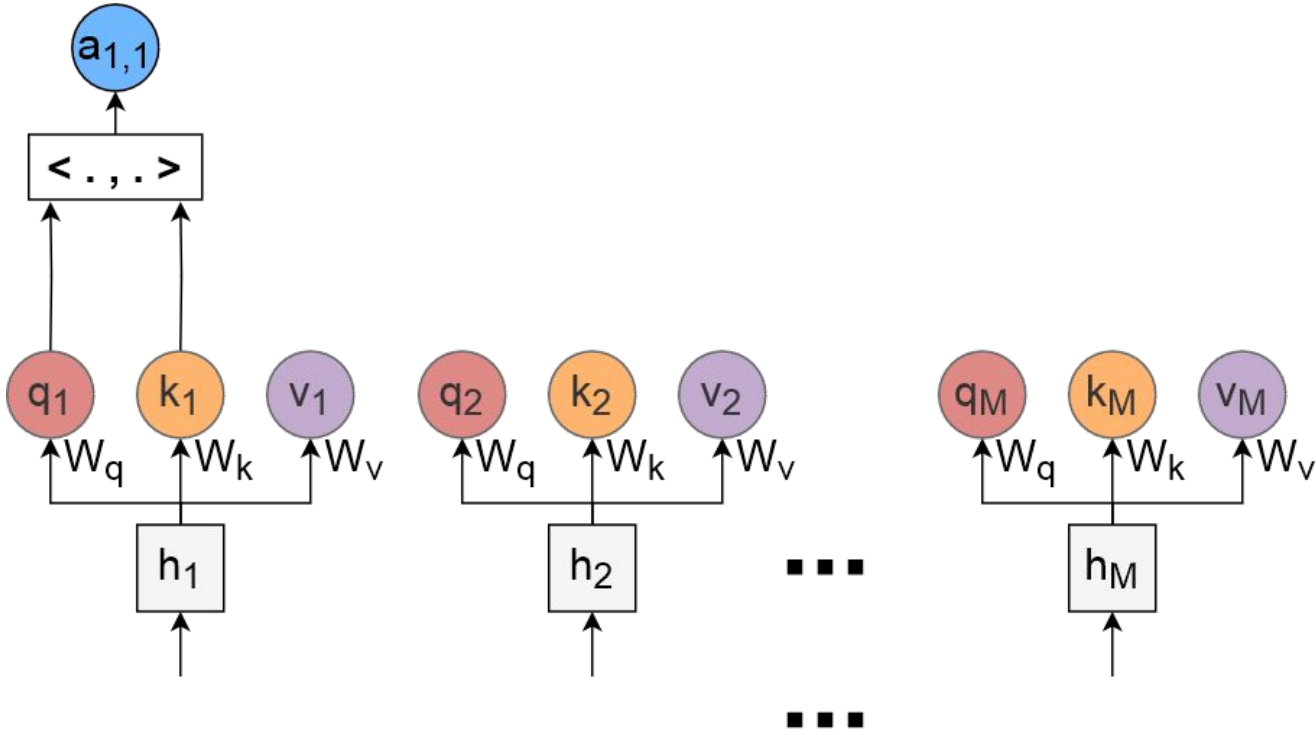
$$a_{i,j} = \langle q_i, k_j \rangle$$

Finally, the sum of the **values** **weighted** by the normalized attention scores yields the individual outputs:

$$\alpha_{i,j} = \text{softmax}_i\left(\frac{a_{i,j}}{\sqrt{d_q}}\right) \qquad \hat{y}_i = \sum_{j=1}^{M} \alpha_{i,j} \cdot v_j$$

Self-attention layer

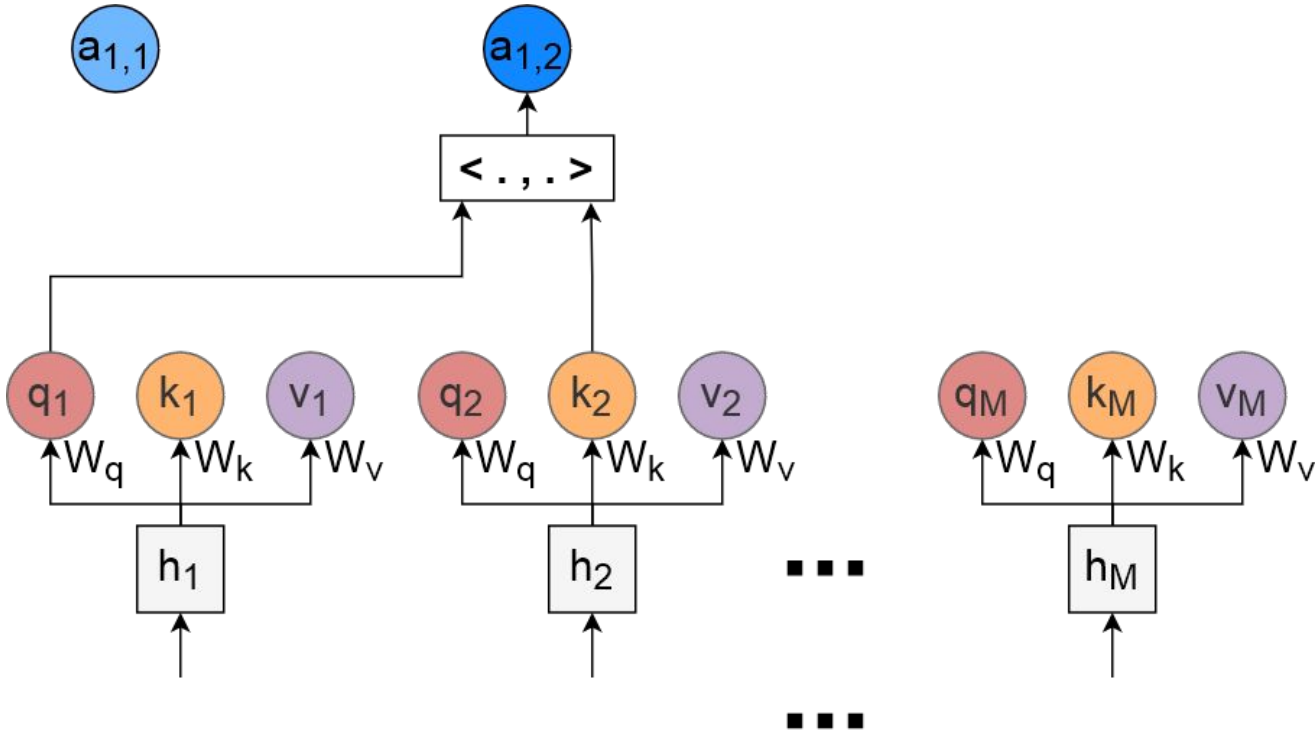
$$q_i = W_q h_i$$
$$k_i = W_k h_i$$
$$v_i = W_v h_i$$
$$a_{i,j} = \langle q_i, k_j \rangle$$



Self-attention layer

The **attention score** estimation remains unchanged.

$$q_i = W_q h_i$$
$$k_i = W_k h_i$$
$$v_i = W_v h_i$$
$$a_{i,j} = \langle q_i, k_j \rangle$$

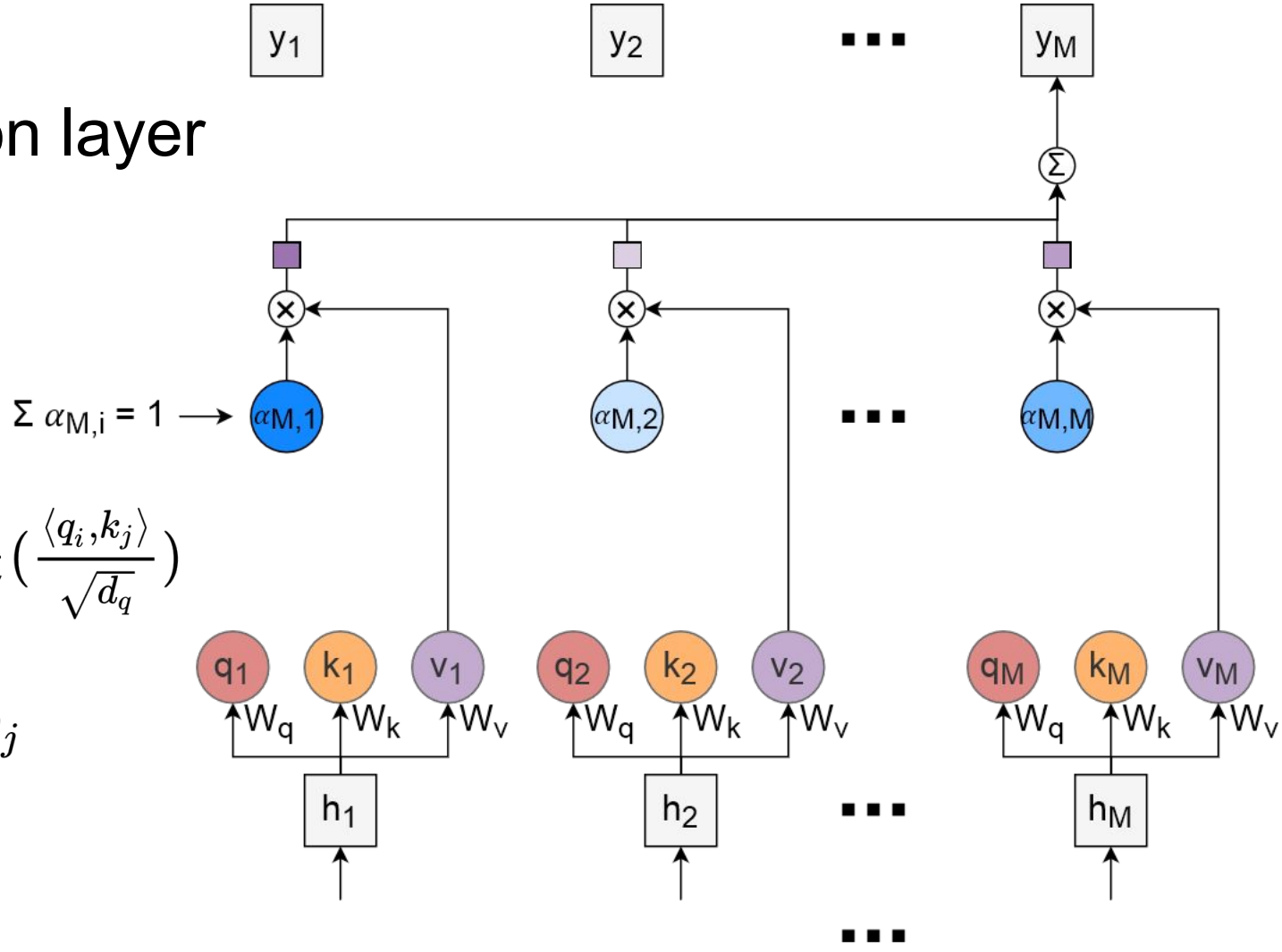


Self-attention layer

$$q_i = W_q h_i$$
$$k_i = W_k h_i$$
$$v_i = W_v h_i$$

$$\alpha_{i,j} = \text{softmax}_i\left(\frac{\langle q_i, k_j \rangle}{\sqrt{d_q}}\right)$$

$$\hat{y}_i = \sum_{j=1}^{M} \alpha_{i,j} \cdot v_j$$

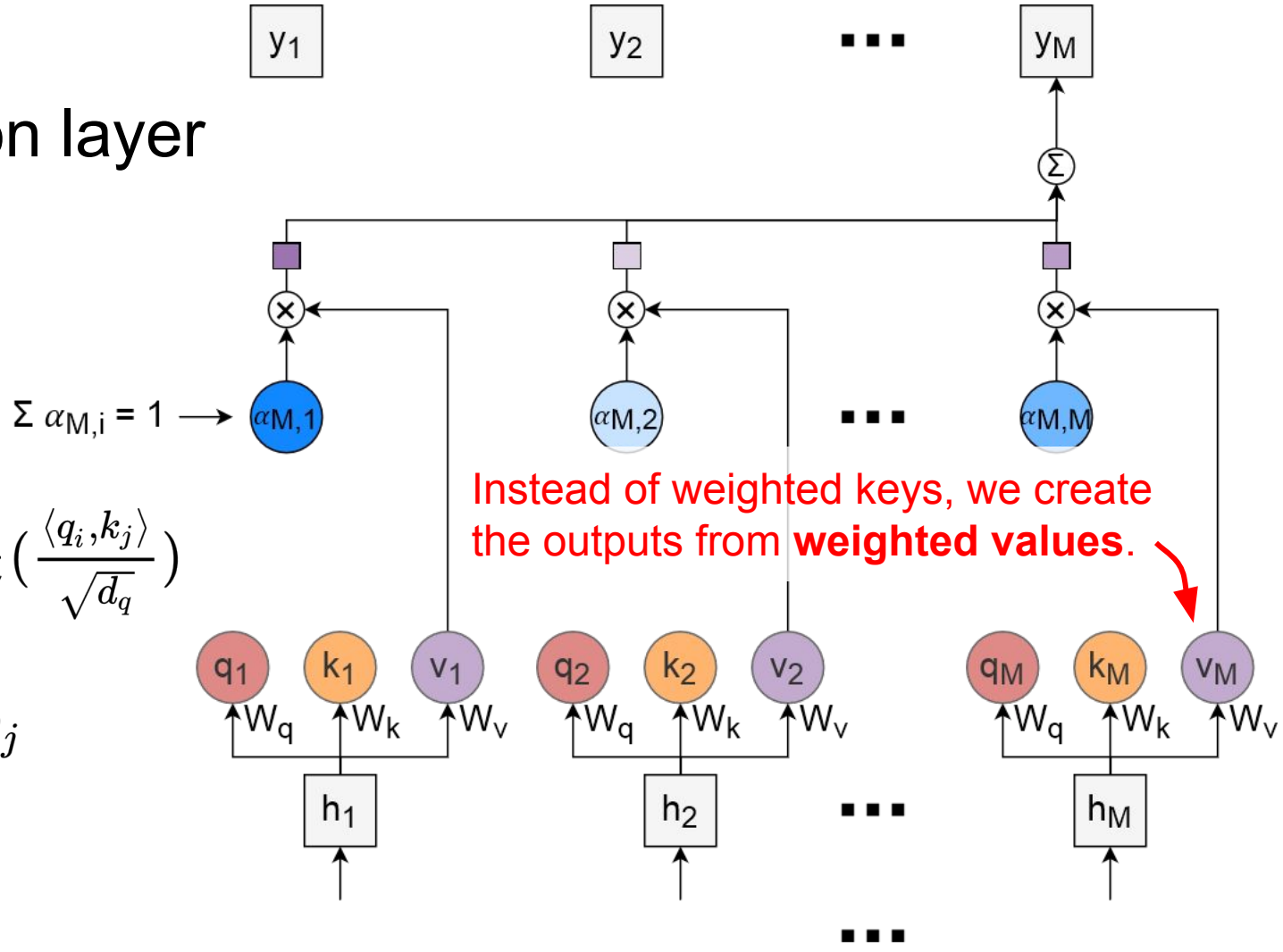


Self-attention layer

$$q_i = W_q h_i$$
$$k_i = W_k h_i$$
$$v_i = W_v h_i$$

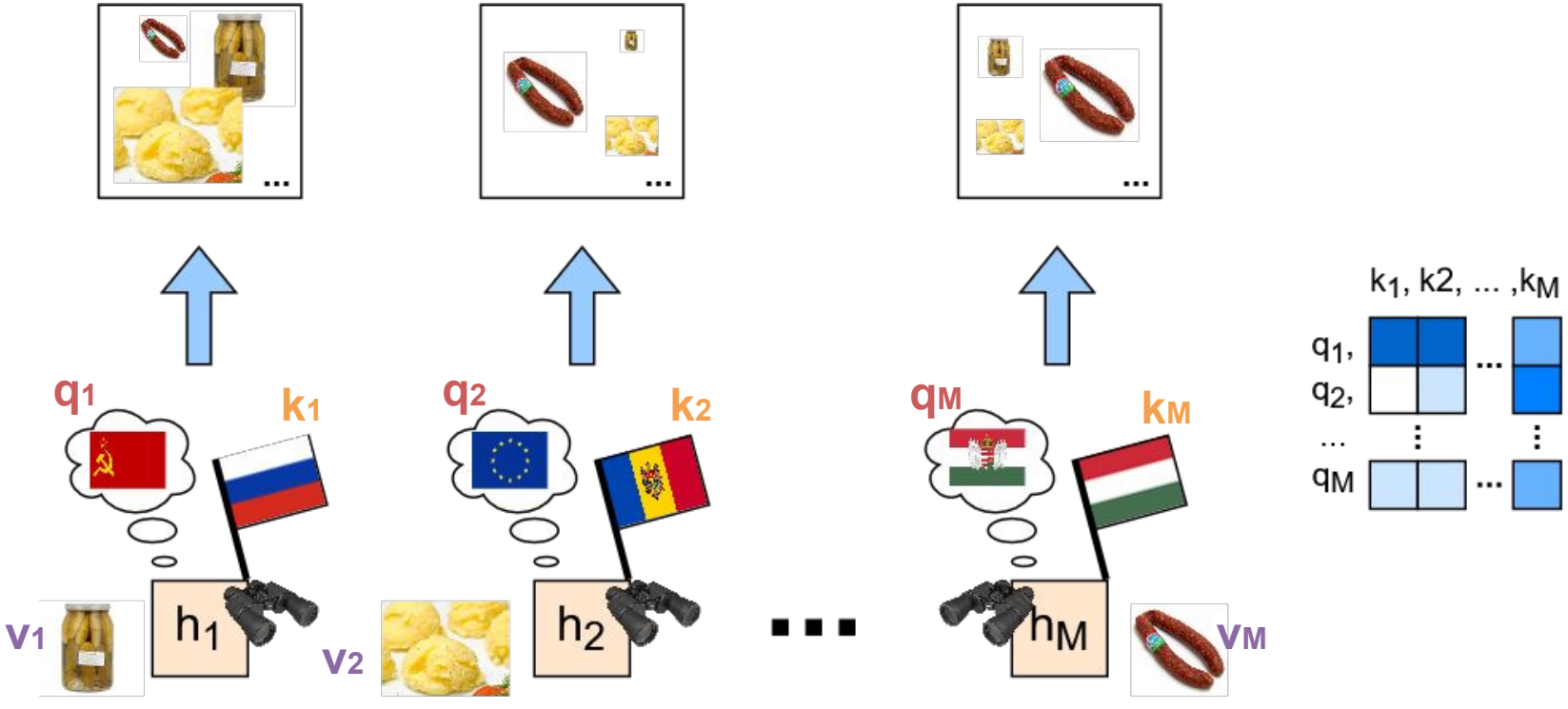
$$\alpha_{i,j} = \text{softmax}_i\left(\frac{\langle q_i, k_j \rangle}{\sqrt{d_q}}\right)$$

$$\hat{y}_i = \sum_{j=1}^{M} \alpha_{i,j} \cdot v_j$$

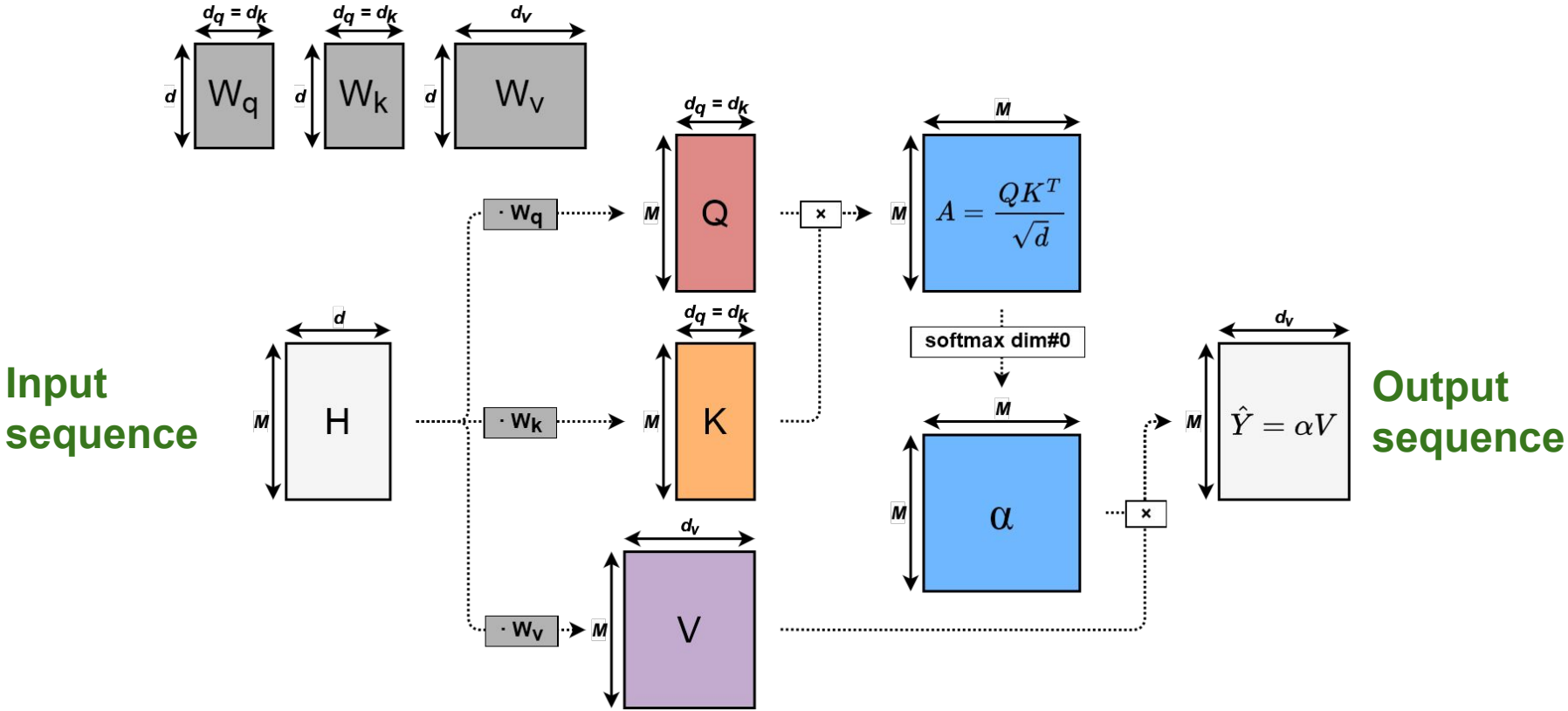


Query, Key, Value

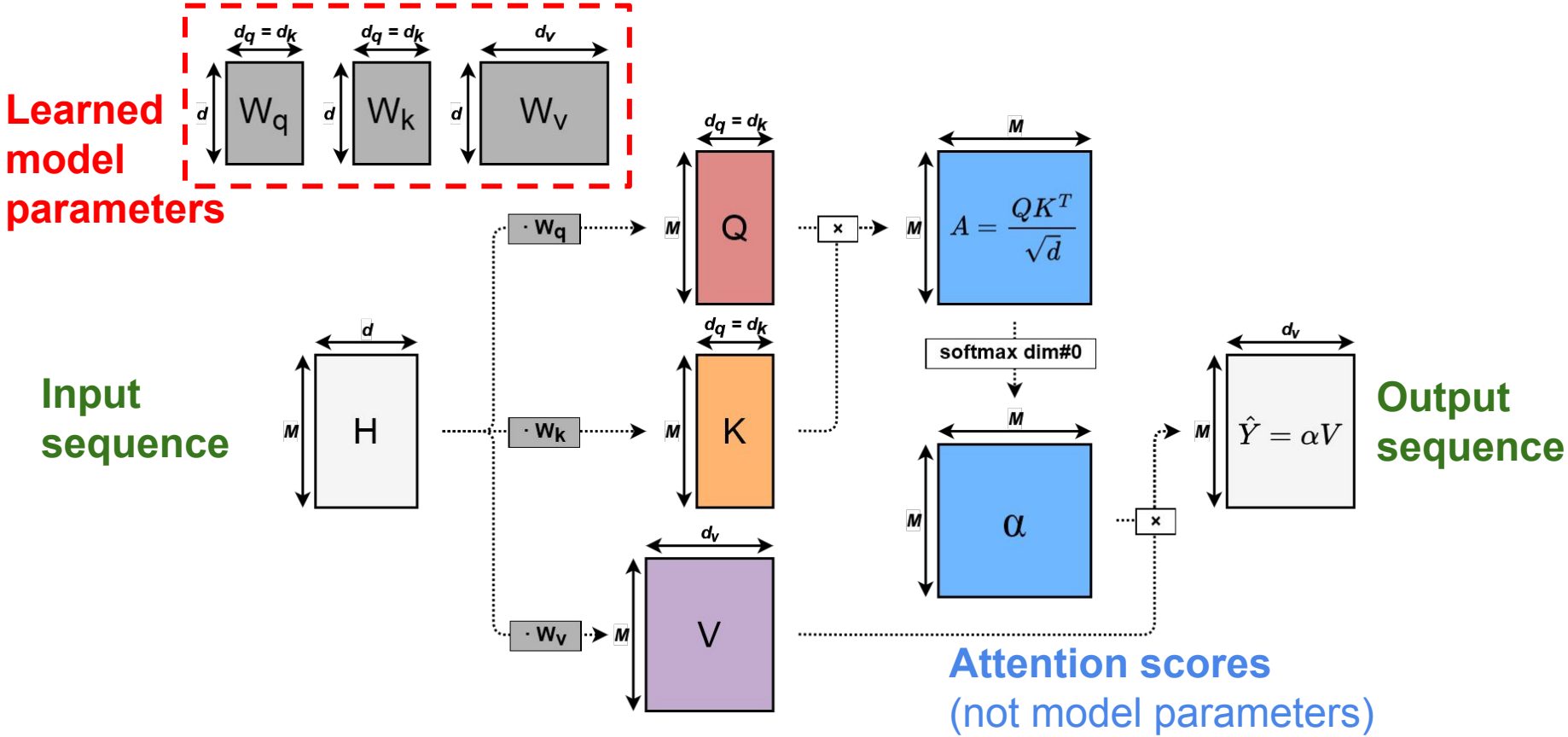
Intuition



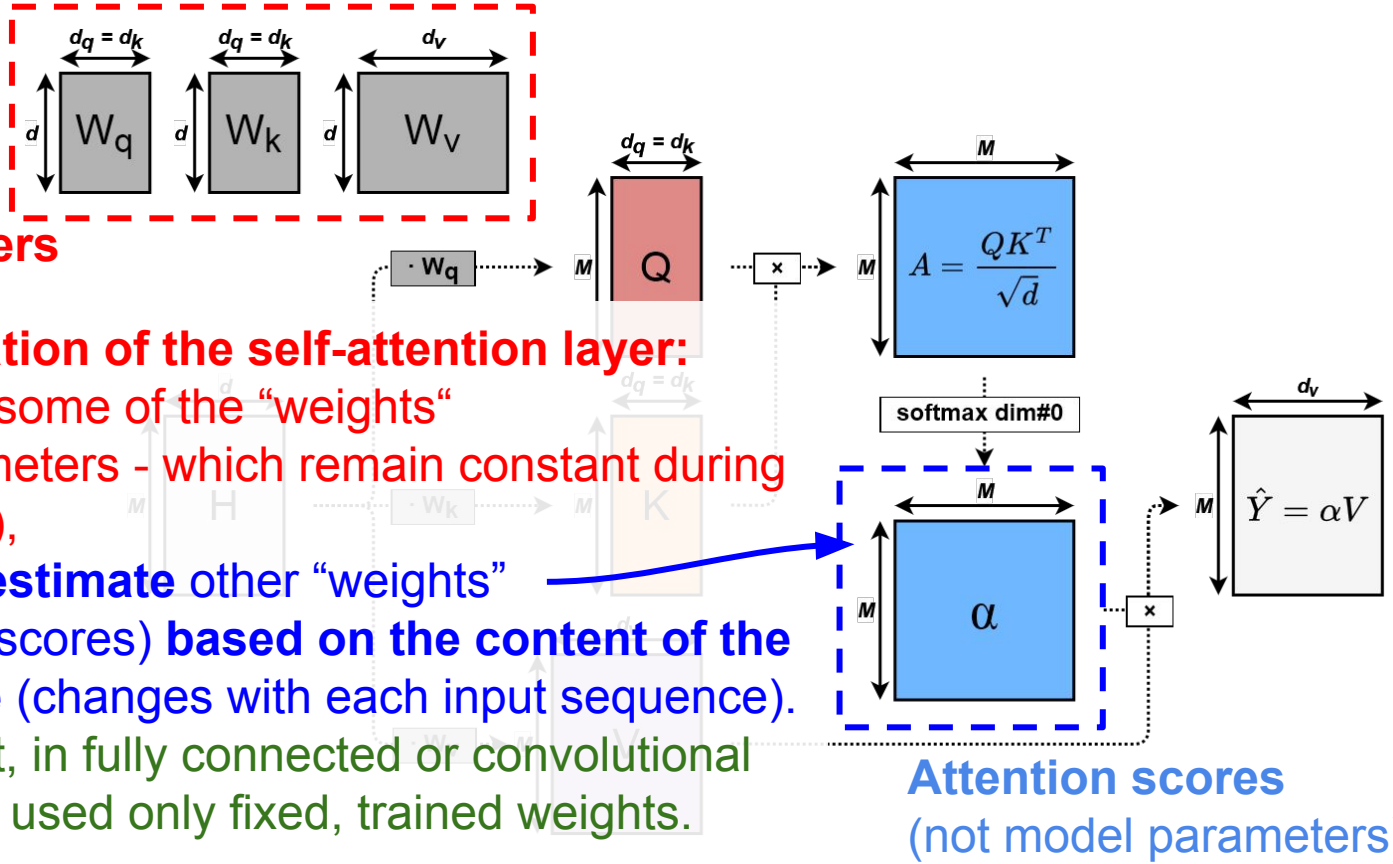
Self-attention layer - Vectorized form



Self-attention layer - Vectorized form



Self-attention layer - Vectorized form

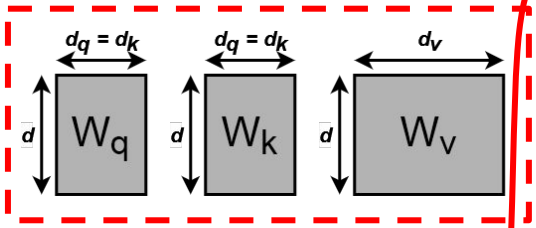


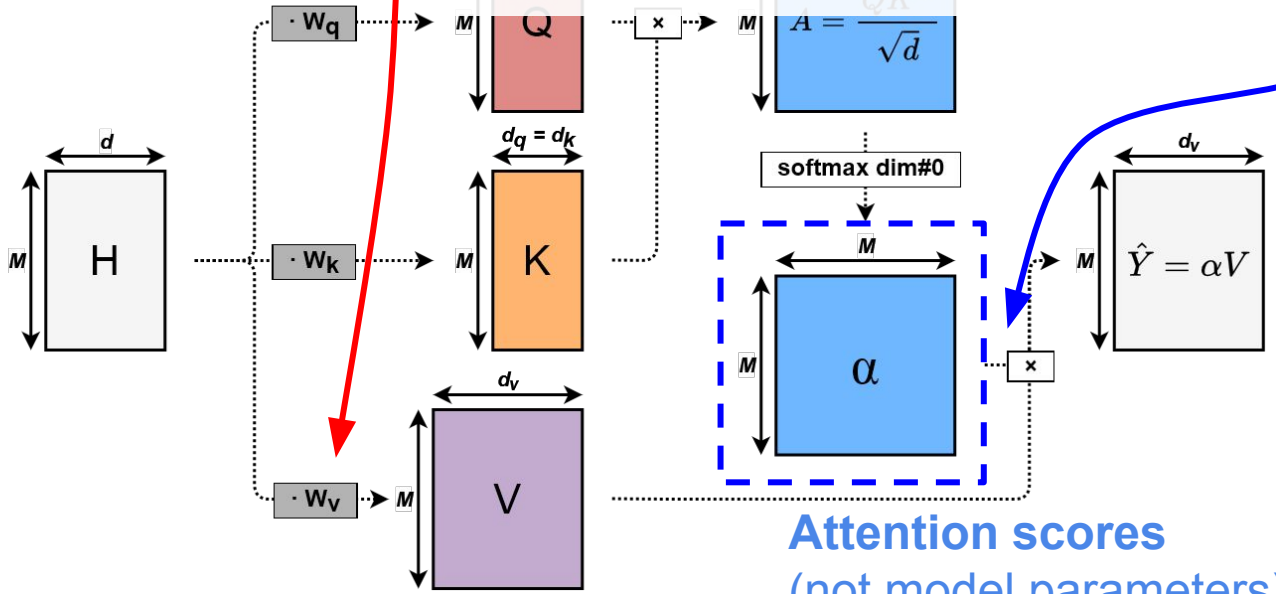
Interpretation of the self-attention layer:
We **learn** some of the "weights"
(the parameters - which remain constant during prediction),
while **we estimate** other "weights"
(attention scores) **based on the content of the sequence** (changes with each input sequence).
In contrast, in fully connected or convolutional layers, we used only fixed, trained weights.

Self-attention layer - Vectorized form

The fixed, learned weights perform a mixing operation between the variables inside each element of the sequence (vectors of length d). **The affinities,** on the other hand, **perform a mixing operation across the elements of the sequence** (M vectors) **using weights estimated based on the content.**

Learned model parameters





Attention scores
(not model parameters)

Image source: Vaswani et al., 2017

Attention-based models

Transformer Network (2017)

The foundations of (Chat)GPT…

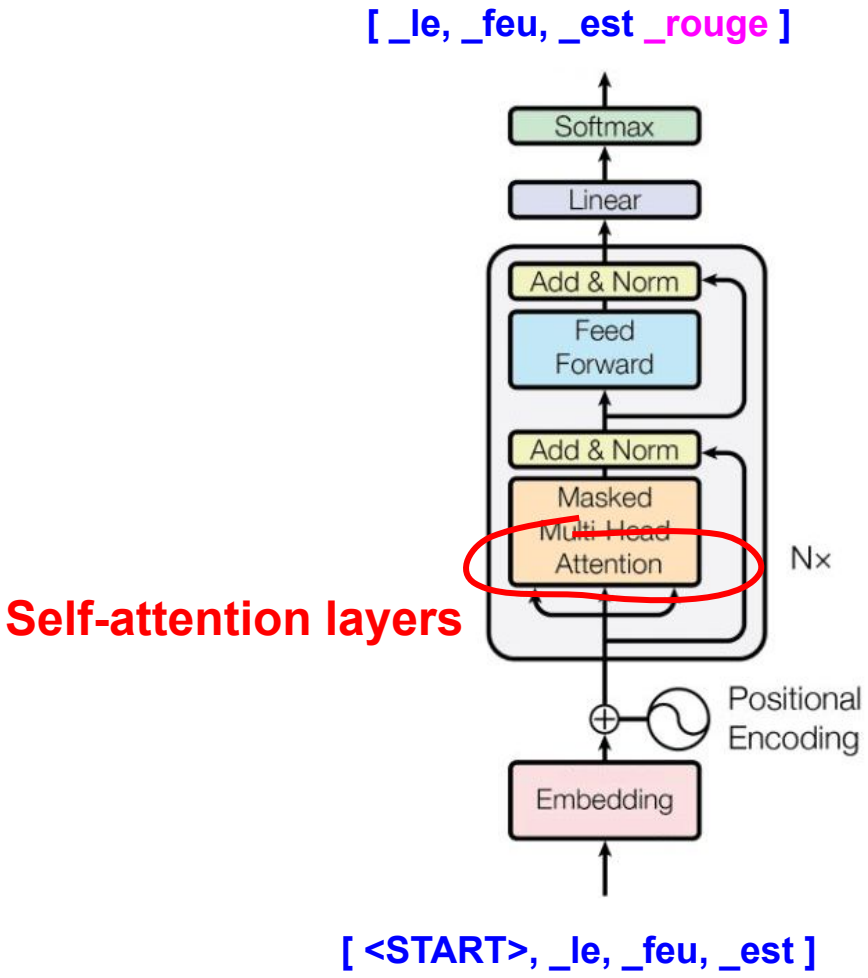
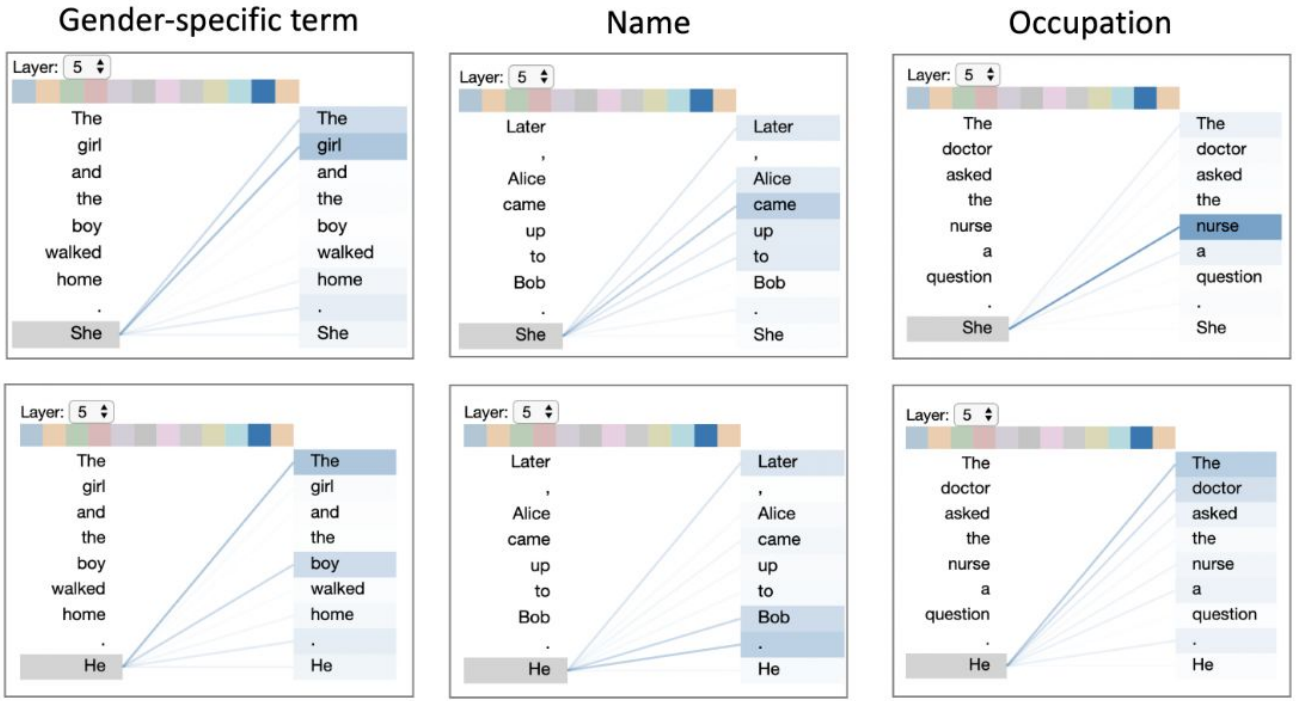


Image source: J. Vig, 2019

Attention-based models

Attention score examples (OpenAI GPT-2), https://github.com/jessevig/bertviz



Advantages of attention-based models

- **The impact of the unstable gradient problem is mitigated:** A larger input context can be learned with shorter backpropagation paths.

Advantages of attention-based models

- **The impact of the unstable gradient problem is mitigated:** A larger input context can be learned with shorter backpropagation paths.
- **Spatially extensive relationships can be learned:** It is not necessary to “memorize” everything at once in a hidden representation vector.

Advantages of attention-based models

- **The impact of the unstable gradient problem is mitigated:** A larger input context can be learned with shorter backpropagation paths.
- **Spatially extensive relationships can be learned:** It is not necessary to “memorize” everything at once in a hidden representation vector.
- **More efficient evaluation:** Estimating attention scores for every element of the sequence can be easily performed in parallel.

Advantages of attention-based models

- **The impact of the unstable gradient problem is mitigated:** A larger input context can be learned with shorter backpropagation paths.
- **Spatially extensive relationships can be learned:** It is not necessary to “memorize” everything at once in a hidden representation vector.
- **More efficient evaluation:** Estimating attention scores for every element of the sequence can be easily performed in parallel.
- **Drawback:** Every element of the sequence must be compared with every other element.
  → The computational cost of calculating attention scores is $\sim \mathcal{O}(n^2)$