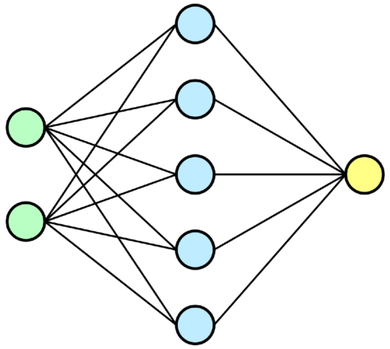
LLMs

# Recurrent networks and attention

Janik Euskirchen

## Neural networks



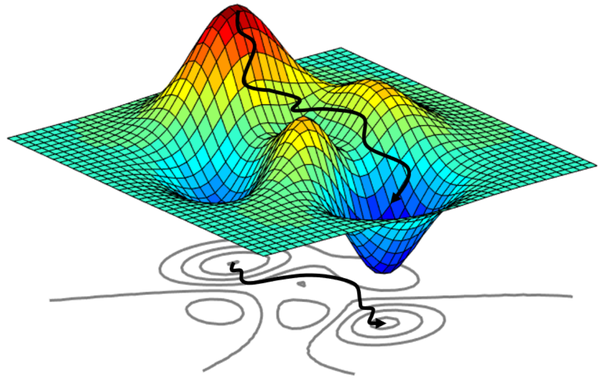
$$\mathbf{h}_0 = \mathbf{x}$$

$$\mathbf{h}_{l+1} = g(\mathbf{W}_{l+1}\mathbf{h}_l + \mathbf{b}_{l+1})$$

ReLU: $g(a) = \max(0, a)$

$$\mathbf{y} = \mathbf{h}_L$$

## Gradient descent

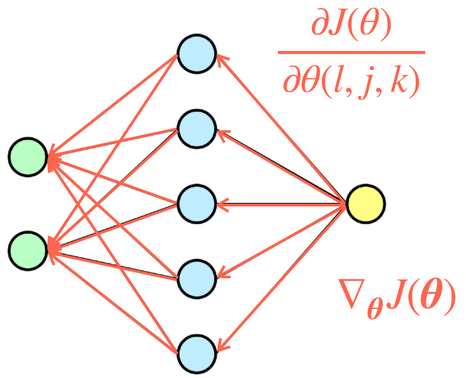


Solve $f_\theta(\mathbf{X}) = \mathbf{Y}$ for $\theta$

$$\theta^* = \arg\min_\theta J(\theta)$$

$$\theta \leftarrow \theta - \alpha \nabla_\theta J(\theta)$$

## Backpropagation

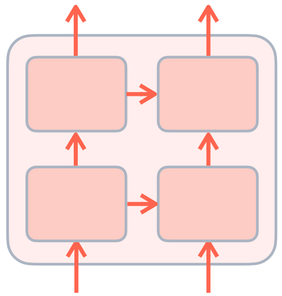



Loss: $\mathcal{L}(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)})$

Objective: $J(\theta) = \mathbb{E}_i[\mathcal{L}(f_\theta(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})]$

Part I

## Recurrent Networks, Seq2Seq and Attention



Part II

## Transformers



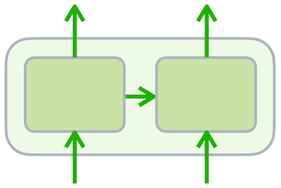
Part III

## State Space Models

???

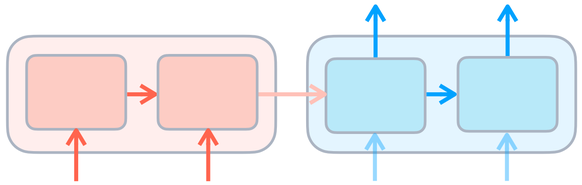
1

## RNN and LSTM



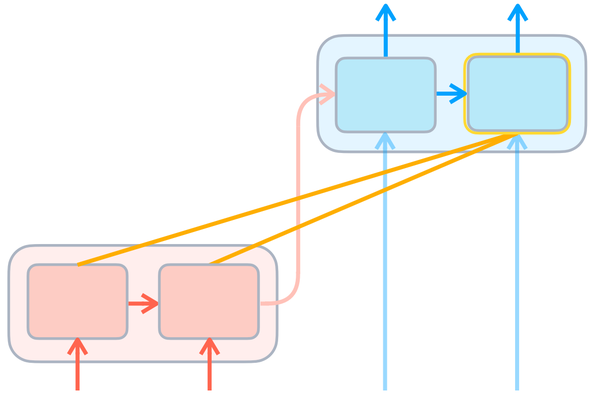
2

## Seq2Seq



3

## Seq2Seq + Attention

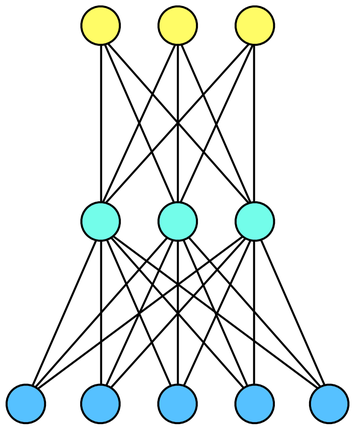


Part I

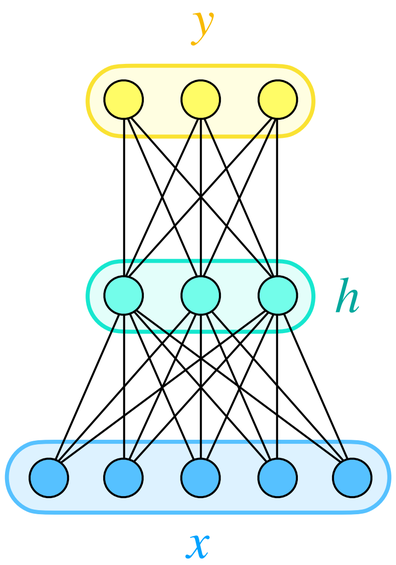
1

# RNN and LSTM

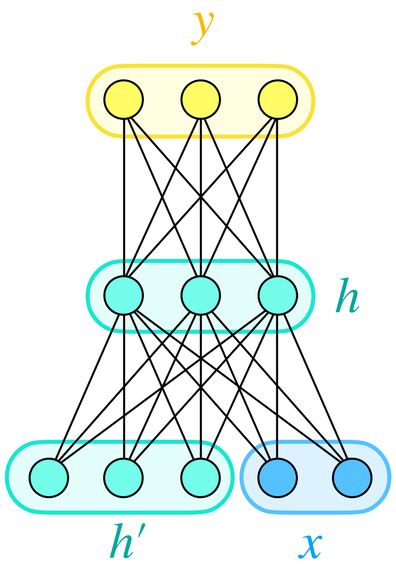
# Recurrent Neural Network



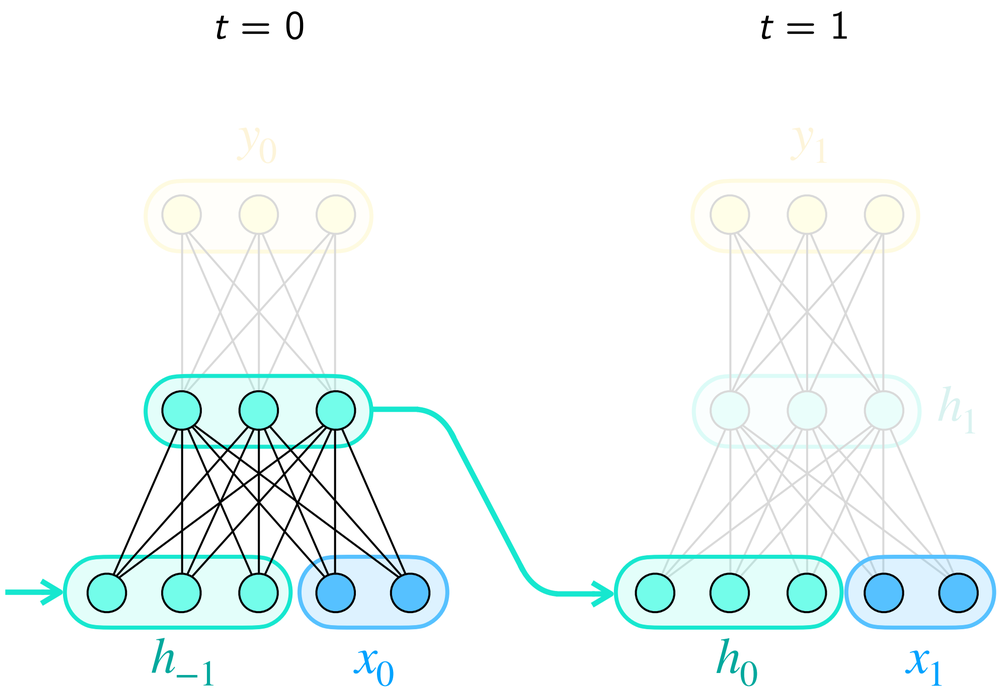
# Recurrent Neural Network



# Recurrent Neural Network



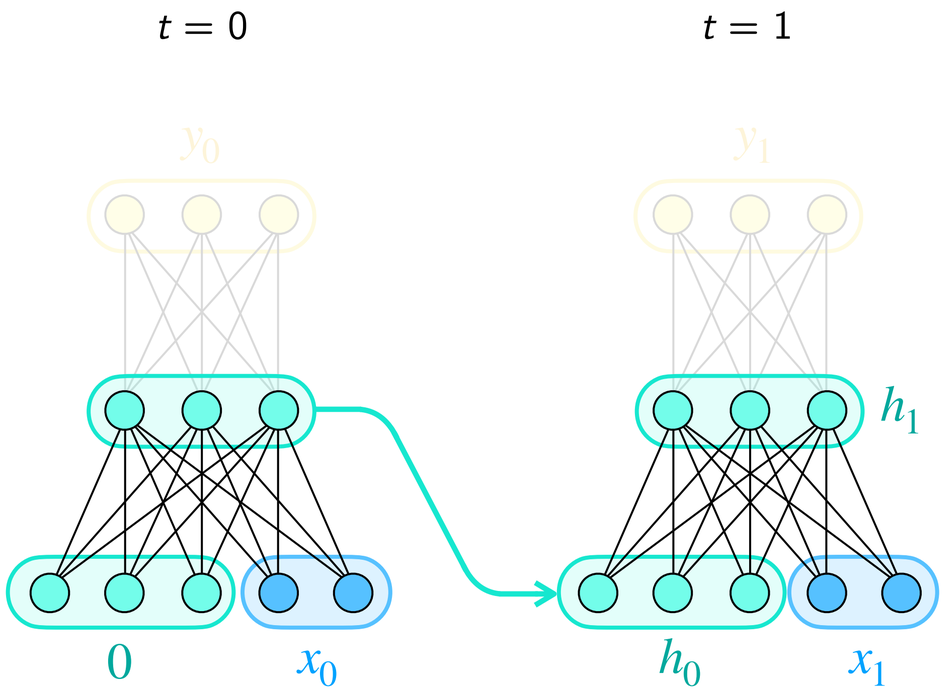
# Recurrent Neural Network




$$y_t = f_y(h_t)$$

$$h_t = f_h(h_{t-1}, x_t)$$

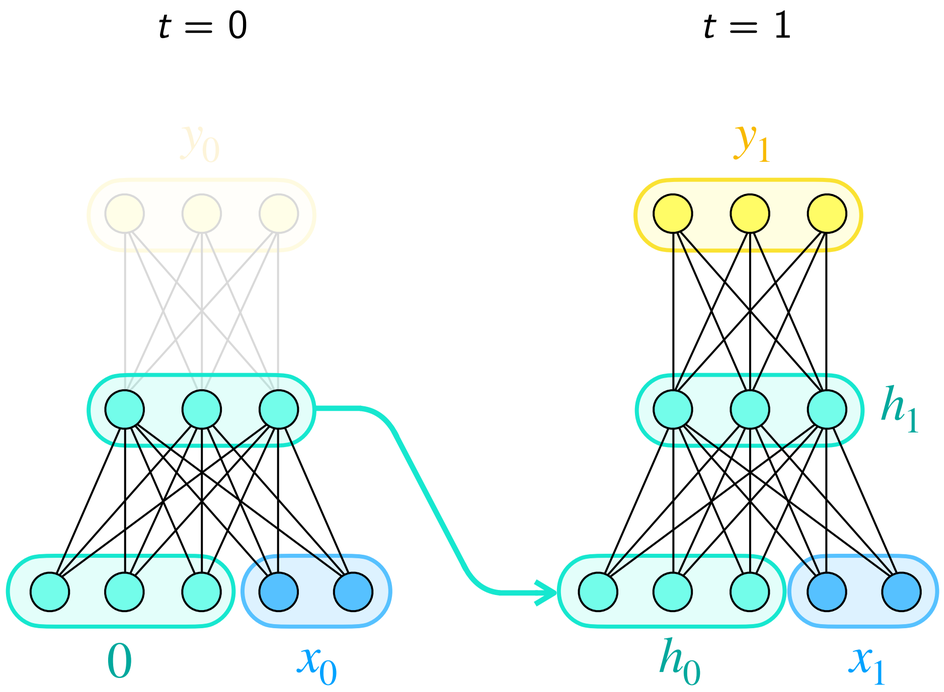
# Recurrent Neural Network




$$y_t = f_y(h_t)$$

$$\begin{aligned} h_t &= f_h(h_{t-1}, x_t) \\ &= g(W_h\,[h_{t-1}, x_t]) \end{aligned}$$

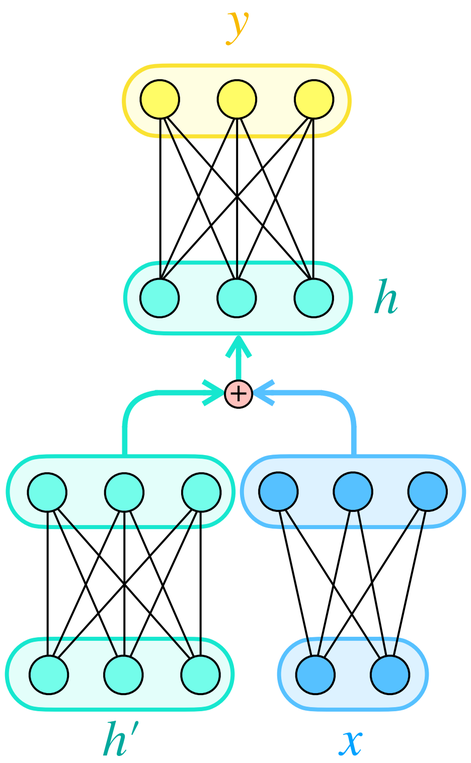
# Recurrent Neural Network




$$y_t = f_y(h_t)$$

$$h_t = f_h(h_{t-1}, x_t)$$
$$= g(W_h\,[h_{t-1}, x_t])$$

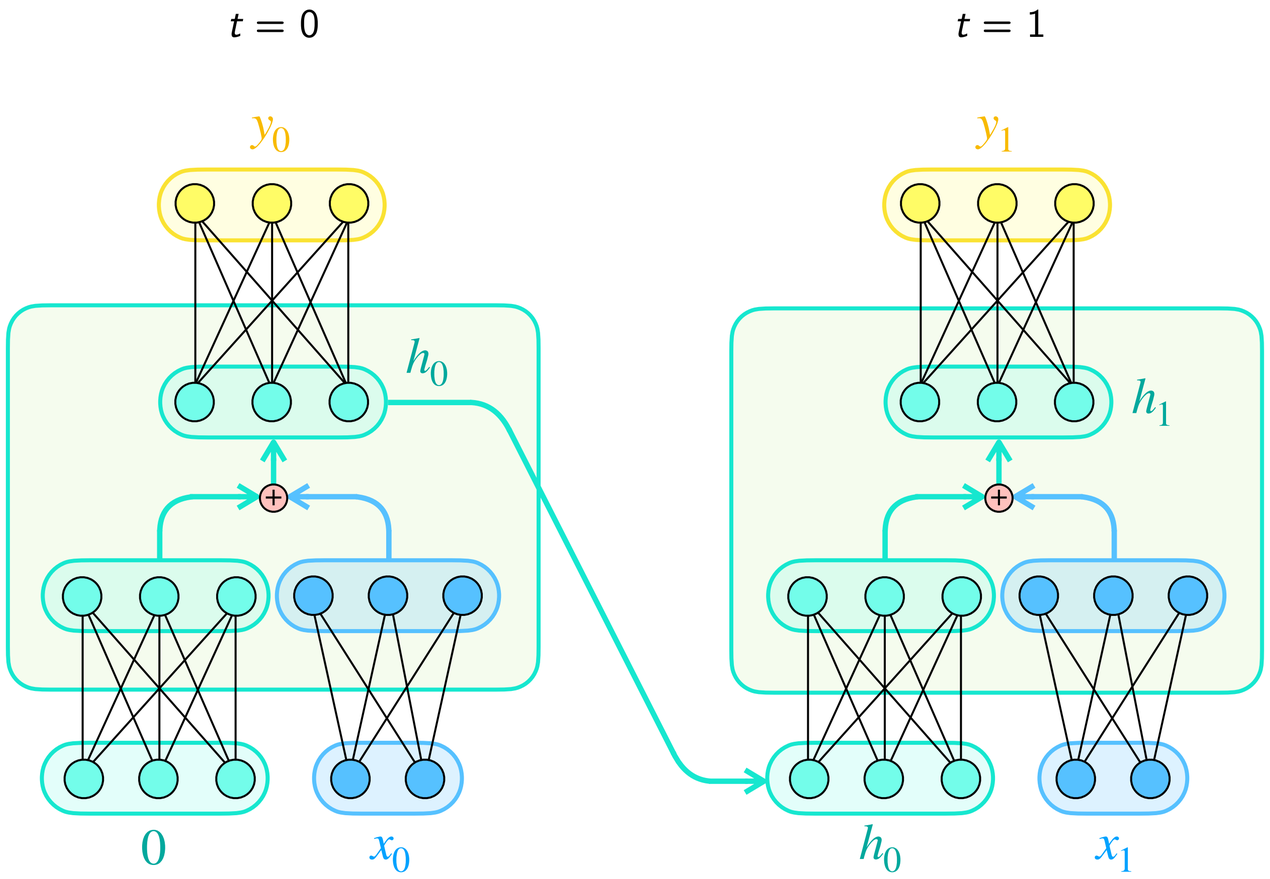
## Vanilla RNN




$$y = f_y(h)$$
$$= W_o h$$

$$h = f_h(h', x)$$
$$= \tanh(W_h h' + W_i x)$$

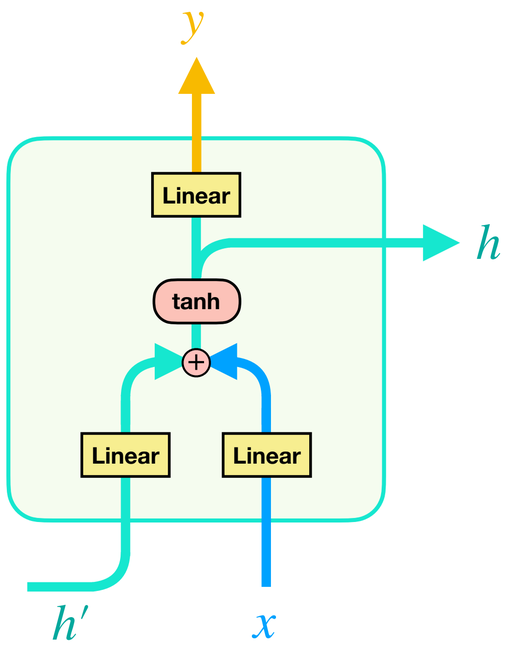
## Vanilla RNN




$$y_t = f_y(h_t)$$
$$= W_o h_t$$

$$h_t = f_h(h_{t-1}, x_t)$$
$$= \tanh(W_h h_{t-1} + W_i x_t)$$

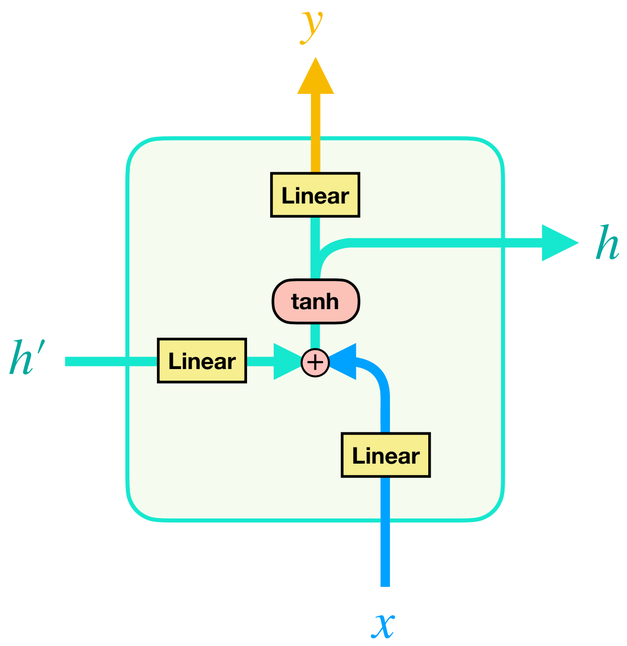
## Vanilla RNN




$$y = f_y(h)$$
$$= W_o h$$

$$h = f_h(h', x)$$
$$= \tanh(W_h h' + W_i x)$$

## Vanilla RNN

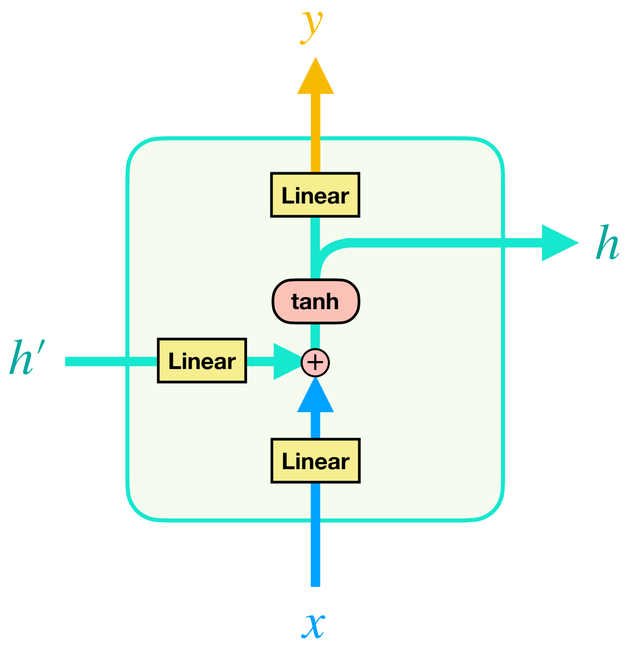



$$y = f_y(h)$$
$$= W_o h$$

$$h = f_h(h', x)$$
$$= \tanh(W_h h' + W_i x)$$

Inspired by Chris Olah's "Understanding LSTM Networks"

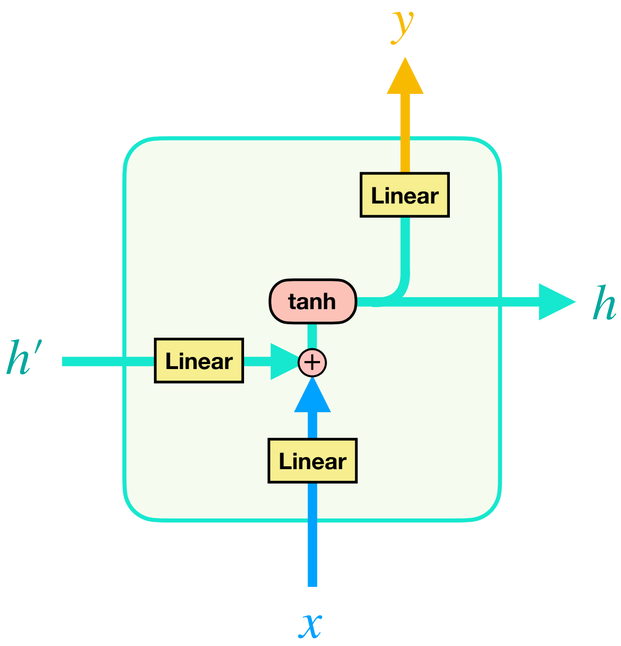
## Vanilla RNN




$$y = f_y(h)$$
$$= W_o h$$

$$h = f_h(h', x)$$
$$= \tanh(W_h h' + W_i x)$$

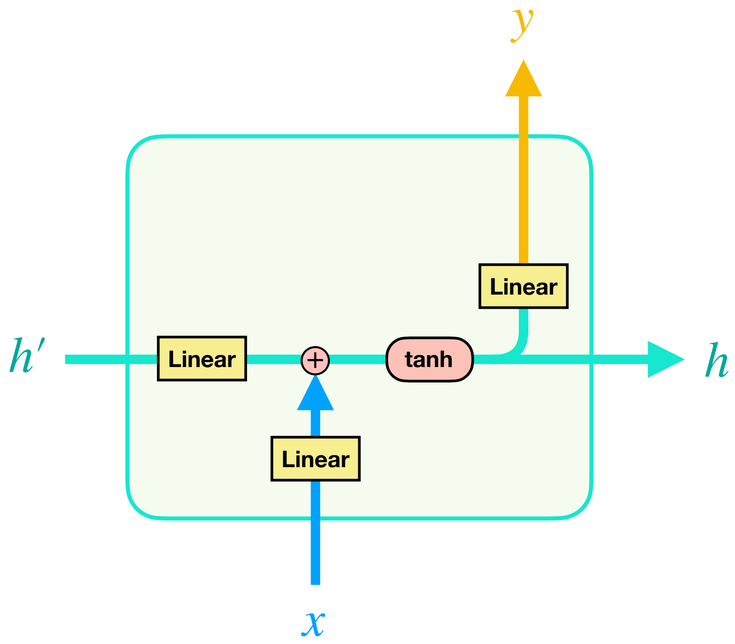
## Vanilla RNN




$$
\begin{aligned}
y &= f_y(h) \\
&= W_o\, h
\end{aligned}
$$

$$
\begin{aligned}
h &= f_h(h', x) \\
&= \tanh(W_h\, h' + W_i\, x)
\end{aligned}
$$

## Vanilla RNN

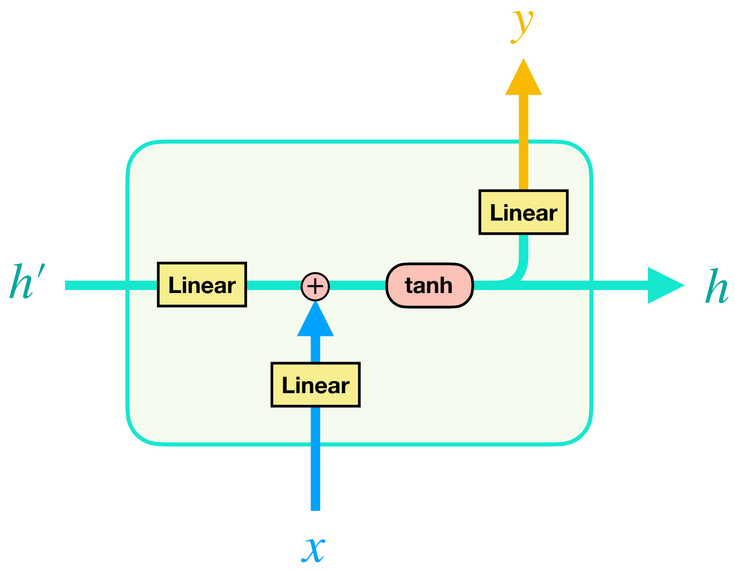



$$y = f_y(h)$$
$$= W_o h$$

$$h = f_h(h', x)$$
$$= \tanh(W_h h' + W_i x)$$

Inspired by Chris Olah's "Understanding LSTM Networks"

## Vanilla RNN

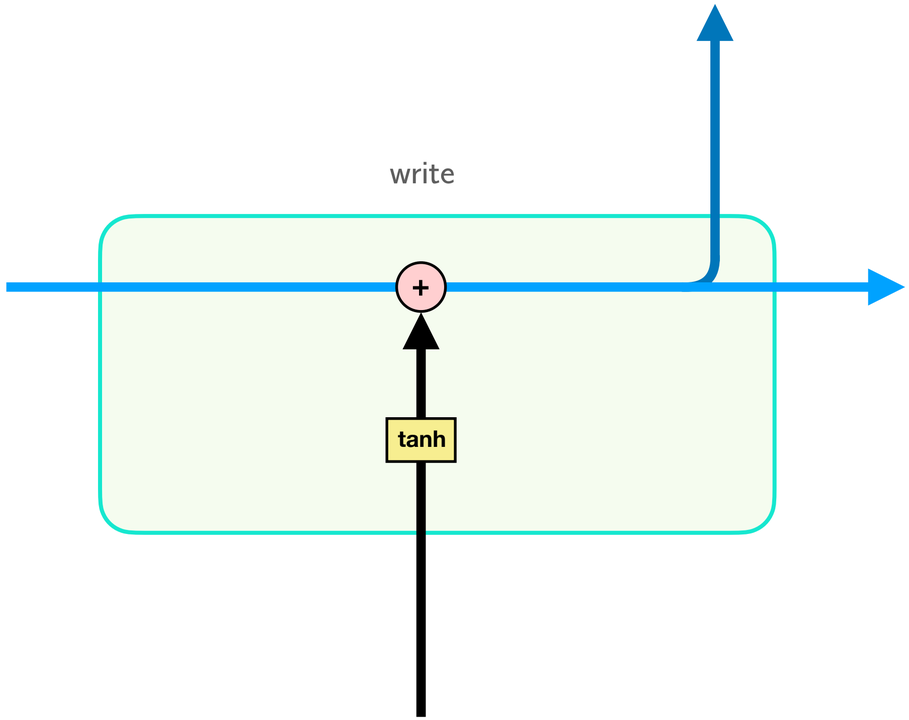



$$y = f_y(h)$$
$$= W_o h$$

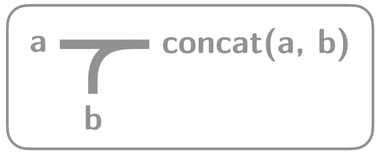
$$h = f_h(h', x)$$
$$= \tanh(W_h h' + W_i x)$$

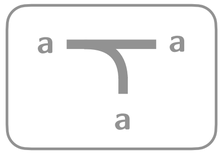
Inspired by Chris Olah's "Understanding LSTM Networks"

# LSTM (Long Short-Term Memory)

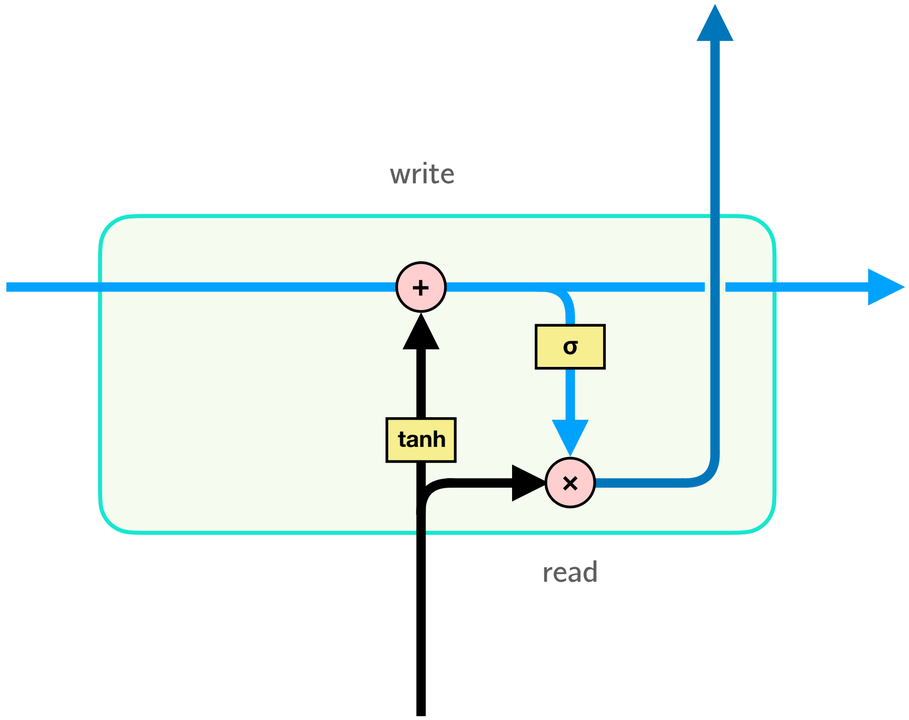



1. Update the state by writing input information into it

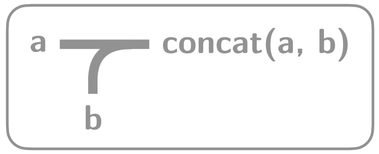


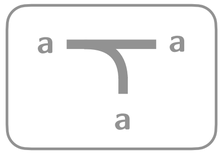



Distinguish between ***state*** and ***output***



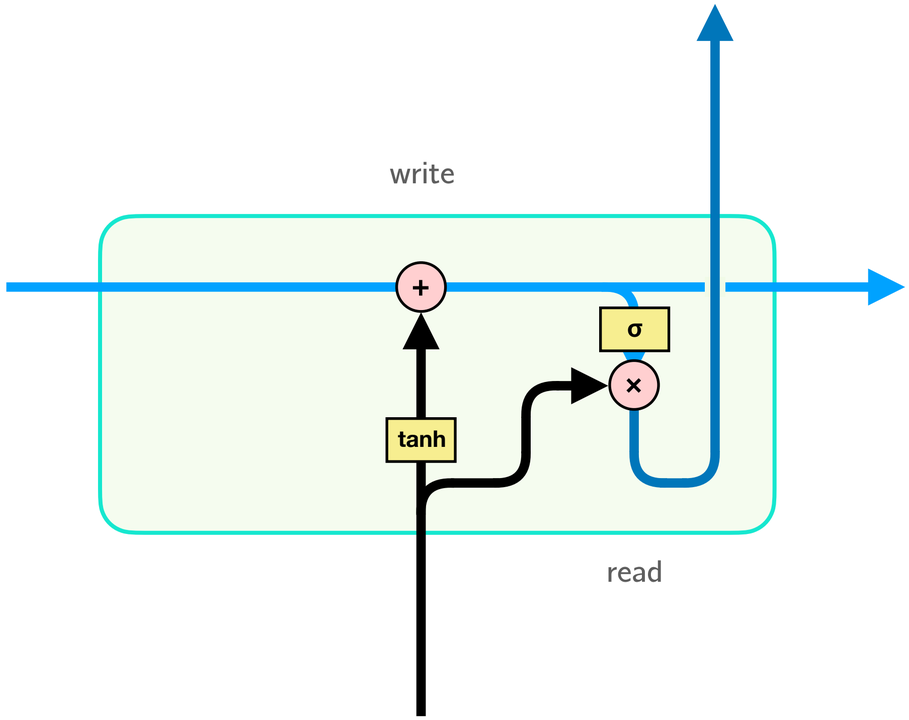

1. Update the state by writing input information into it
2. Build the output by taking the input, reading relevant state information and forgetting/retaining input information based on state



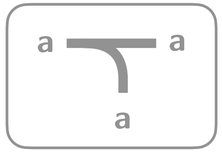




Inspired by Chris Olah's "Understanding LSTM Networks"

Distinguish between ***state*** and ***output***

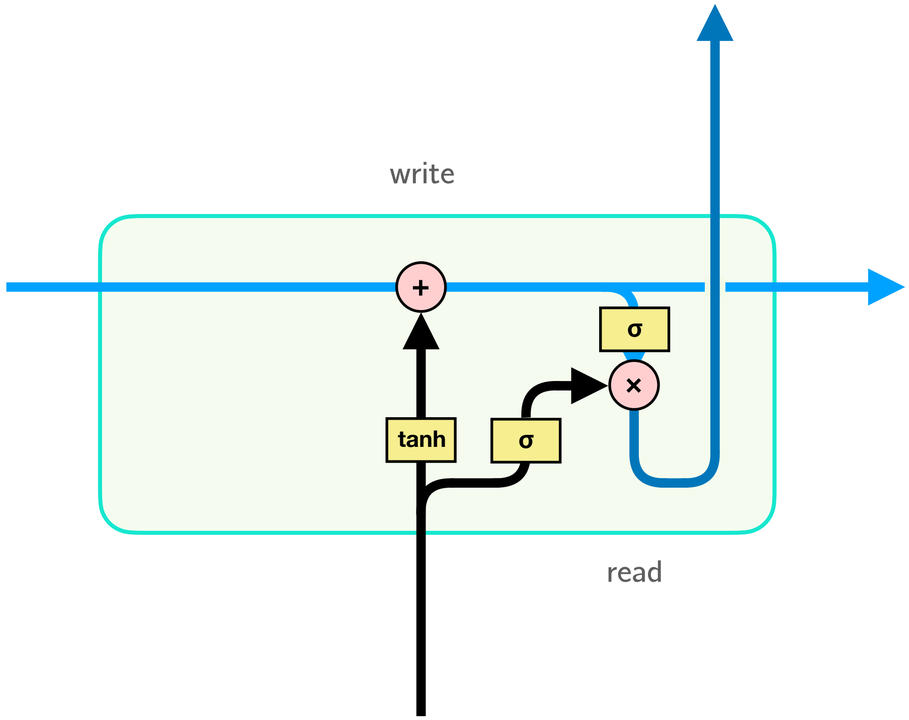



1. Update the state by writing input information into it
2. Build the output by taking the input, reading relevant state information and forgetting/retaining input information based on state

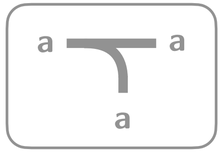




Inspired by Chris Olah's "Understanding LSTM Networks"

Distinguish between ***state*** and ***output***

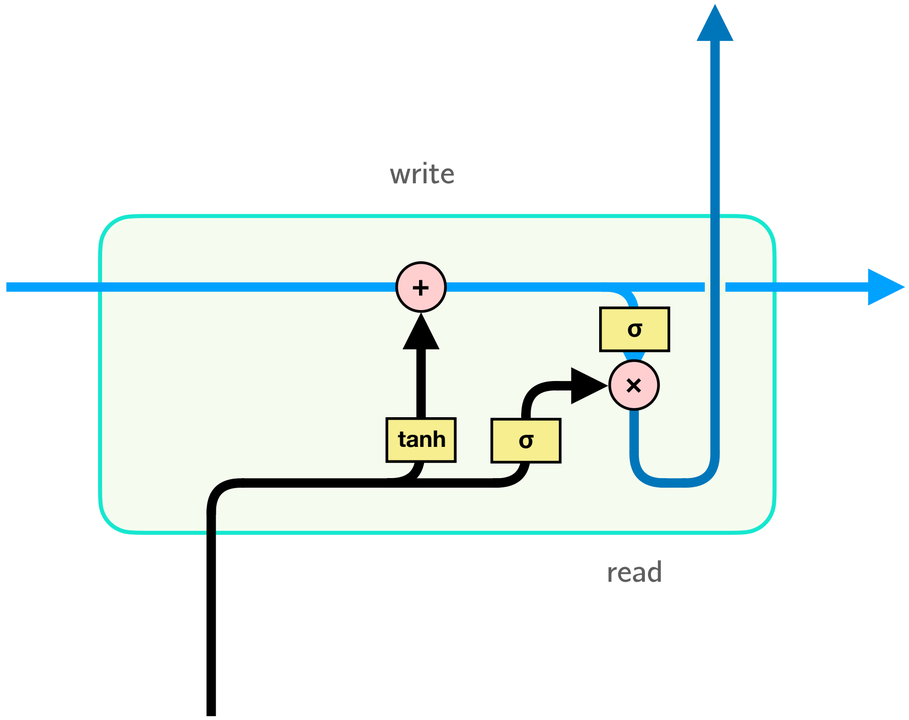



1. Update the state by writing input information into it
2. Build the output by taking the input, reading relevant state information and forgetting/retaining input information based on state

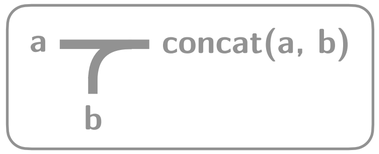



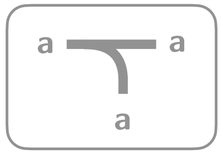

Inspired by Chris Olah's "Understanding LSTM Networks"

Distinguish between ***state*** and ***output***



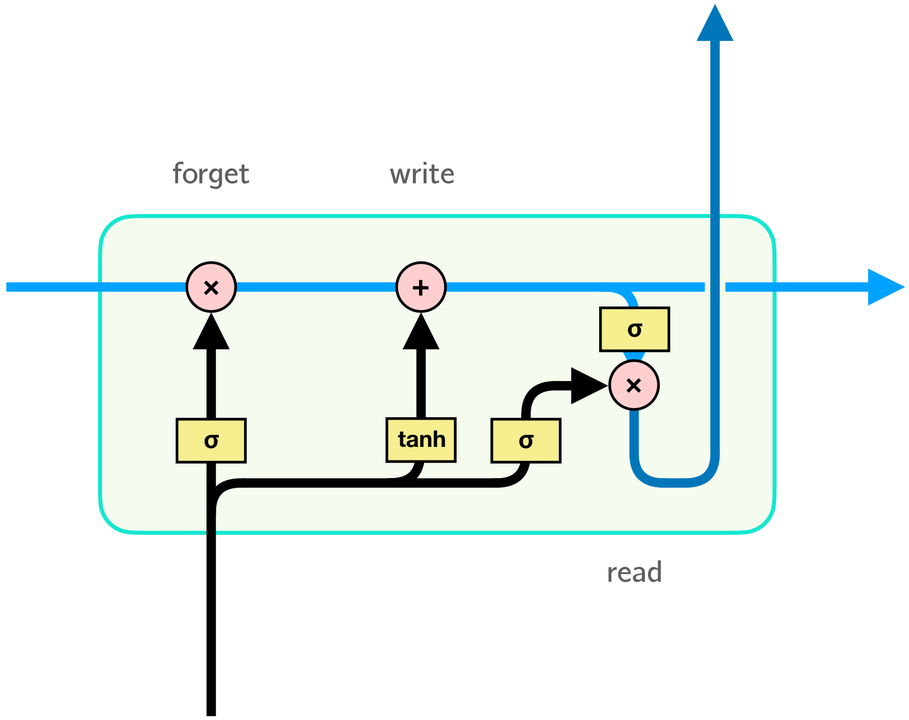

1. Update the state by writing input information into it
2. Build the output by taking the input, reading relevant state information and forgetting/retaining input information based on state
3. Build the output by taking the input, reading relevant state information and forgetting/retaining input information based on state



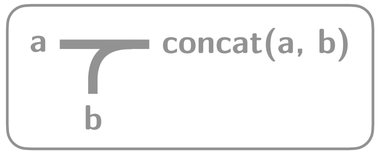



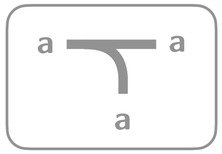

Inspired by Chris Olah's "Understanding LSTM Networks"

Distinguish between ***state*** and ***output***



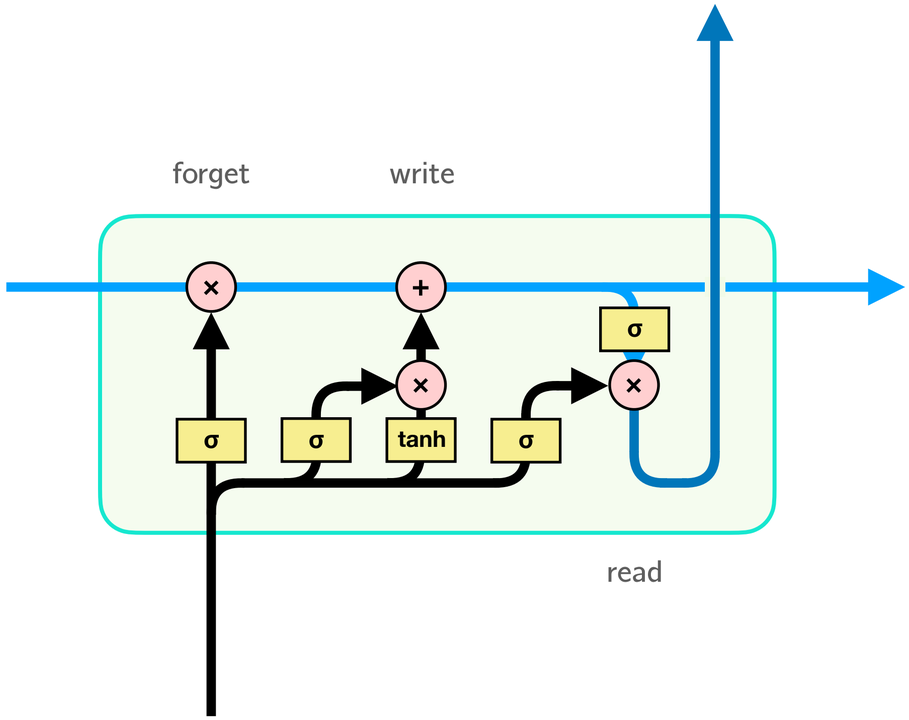

1. Update the state by writing input information into it
2. Build the output by taking the input, reading relevant state information and forgetting/retaining input information based on state
3. Build the output by taking the input, reading relevant state information and forgetting/retaining input information based on state



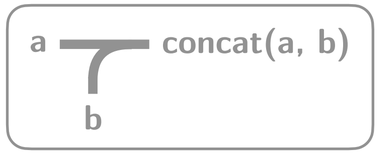



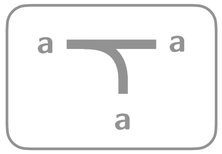

Inspired by Chris Olah's "Understanding LSTM Networks"

Distinguish between ***state*** and ***output***



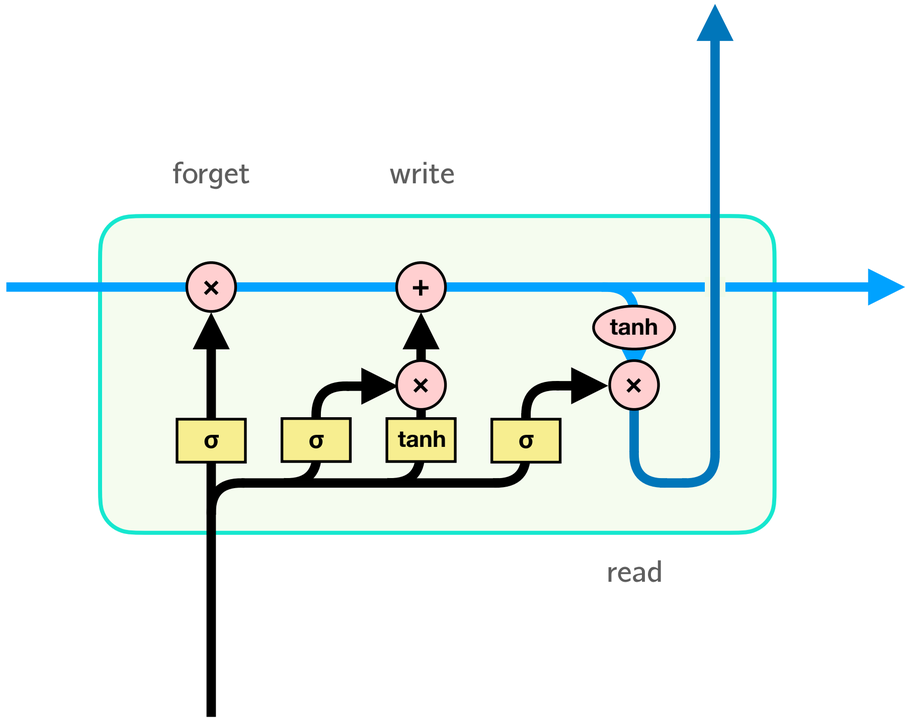

1. Update the state by writing input information into it
2. Build the output by taking the input, reading relevant state information and forgetting/retaining input information based on state
3. Build the output by taking the input, reading relevant state information and forgetting/retaining input information based on state



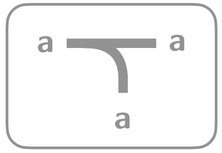




Inspired by Chris Olah's "Understanding LSTM Networks"

Distinguish between ***state*** and ***output***

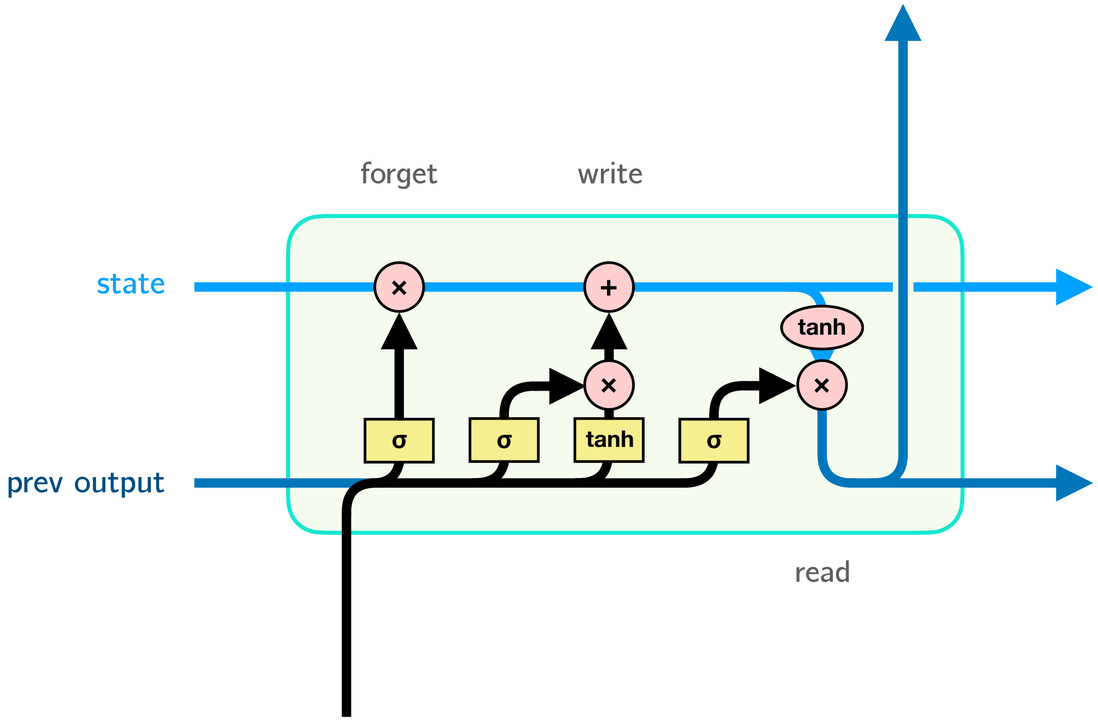



1. Update the state by writing input information into it
2. Build the output by taking the input, reading relevant state information and forgetting/retaining input information based on state
3. Build the output by taking the input, reading relevant state information and forgetting/retaining input information based on state

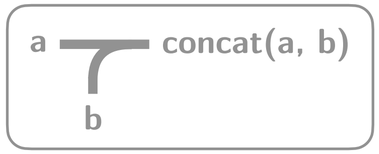



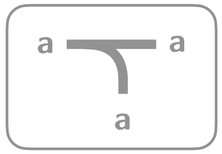

Inspired by Chris Olah's "Understanding LSTM Networks"

Distinguish between ***state*** and ***output***



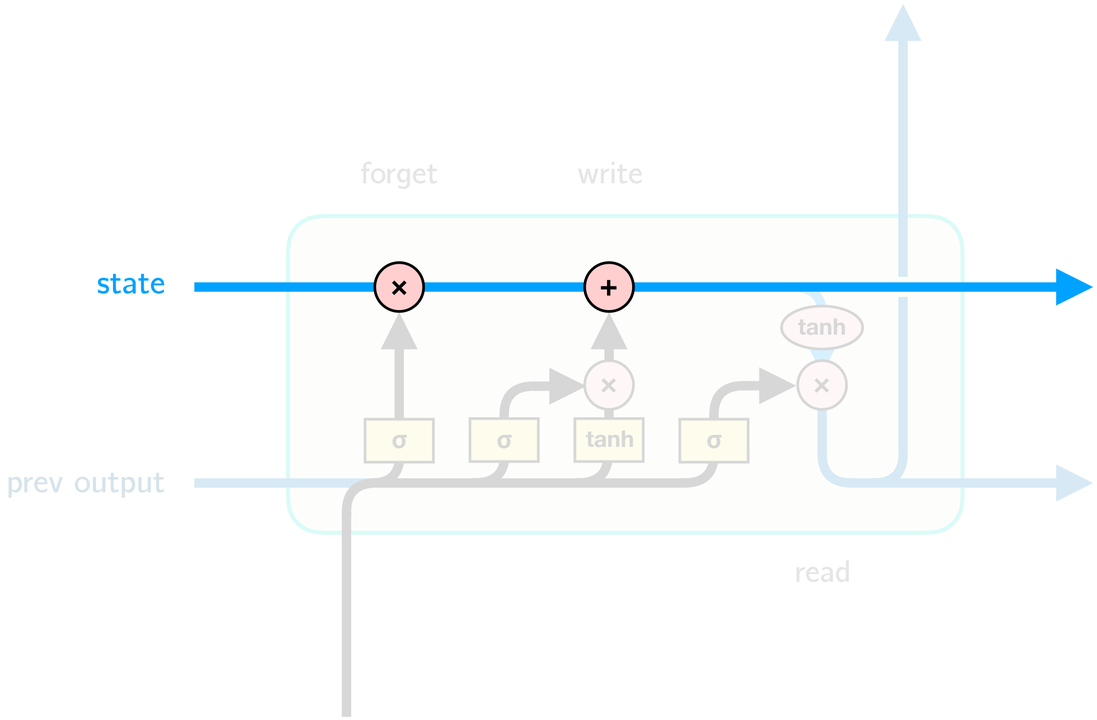

1. Update the state by writing input information into it
2. Build the output by taking the input, reading relevant state information and forgetting/retaining input information based on state
3. Build the output by taking the input, reading relevant state information and forgetting/retaining input information based on state



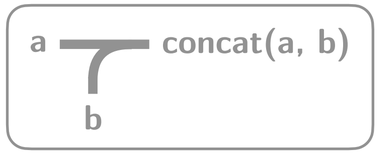



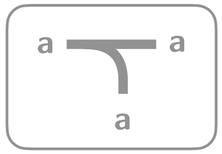

Inspired by Chris Olah's "Understanding LSTM Networks"

Distinguish between ***state*** and ***output***




1. Update the state by writing input information into it
2. Build the output by taking the input, reading relevant state information and forgetting/retaining input information based on state
3. Build the output by taking the input, reading relevant state information and forgetting/retaining input information based on state

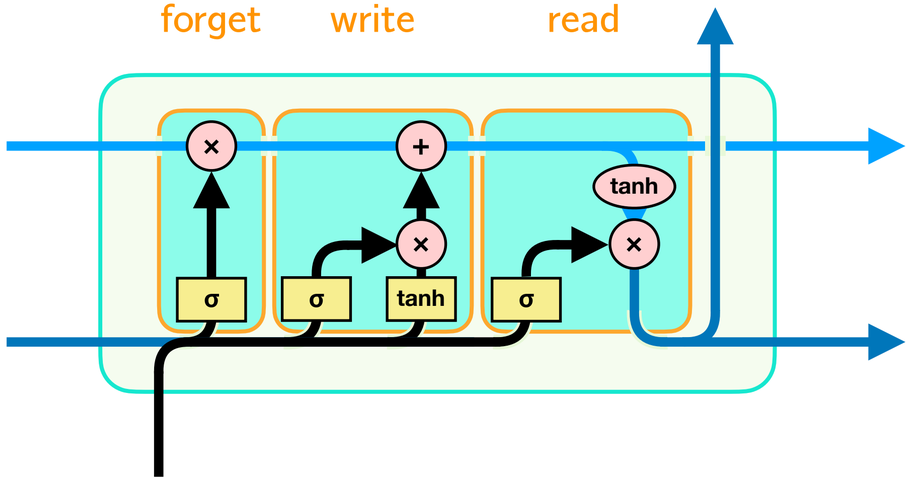


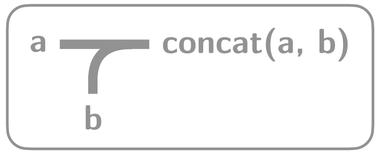



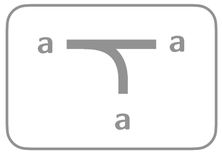

Inspired by Chris Olah's "Understanding LSTM Networks"

# LSTM

**LSTM** = long short-term memory



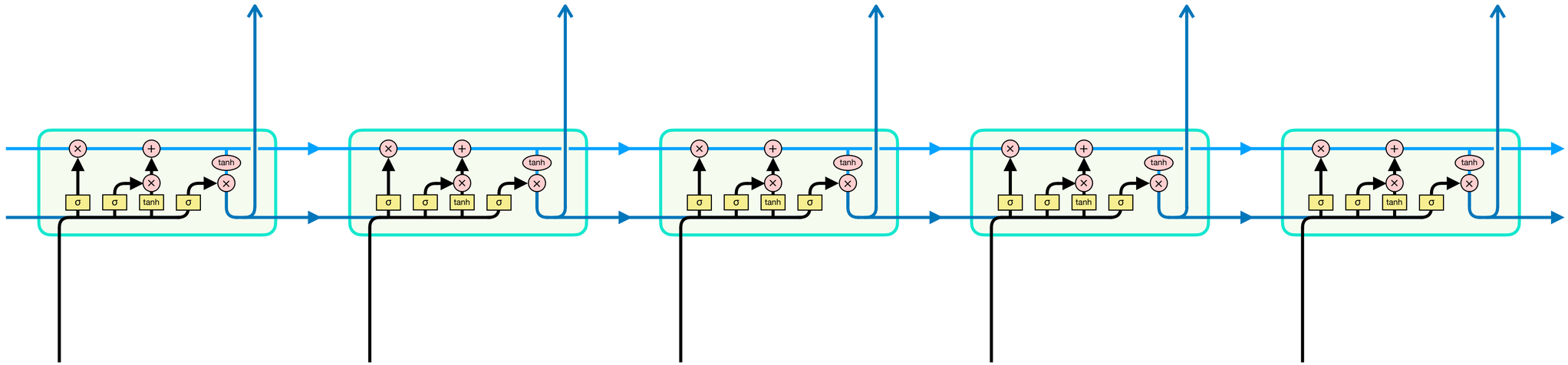



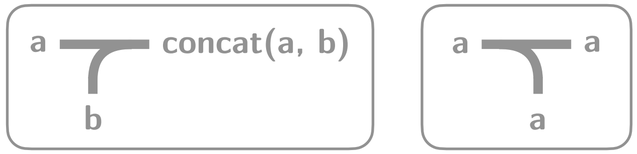




Inspired by Chris Olah's "Understanding LSTM Networks"

# LSTM

**LSTM** = long short-term memory







Inspired by Chris Olah's "Understanding LSTM Networks"

2

# Seq2Seq

(Refine sequence *S* to *S'*)

non-recurrent neural network

image captioning, text generation

text classification

new representation of text

text in language *B*

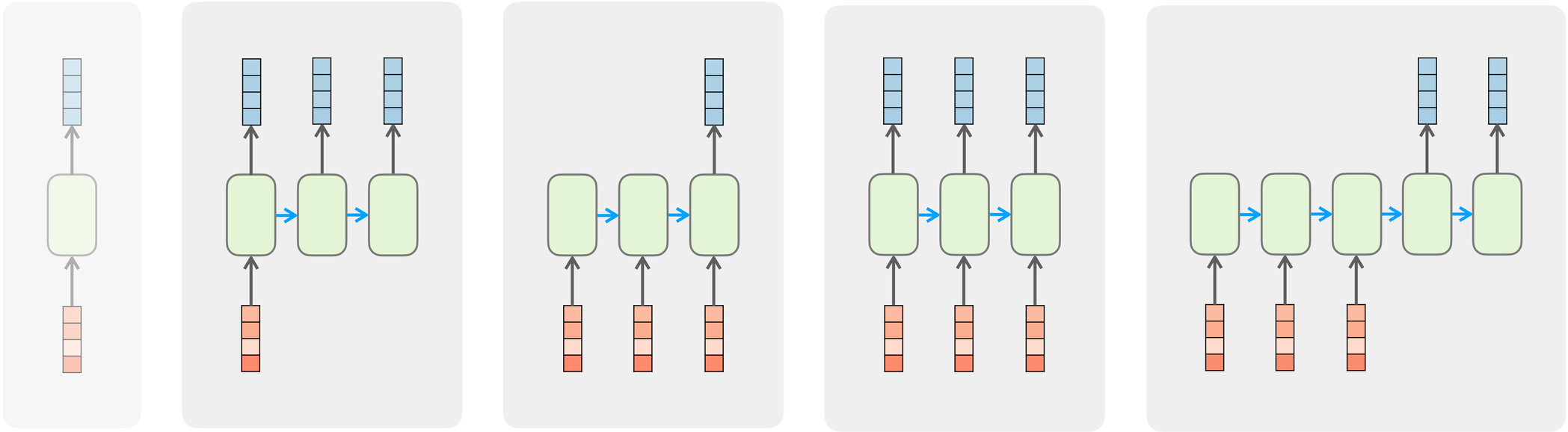
**1-to-1**

**1-to-*N***

***N*-to-1**

***N*-to-*N***

***N*-to-*M***



text

text

text in language *A*

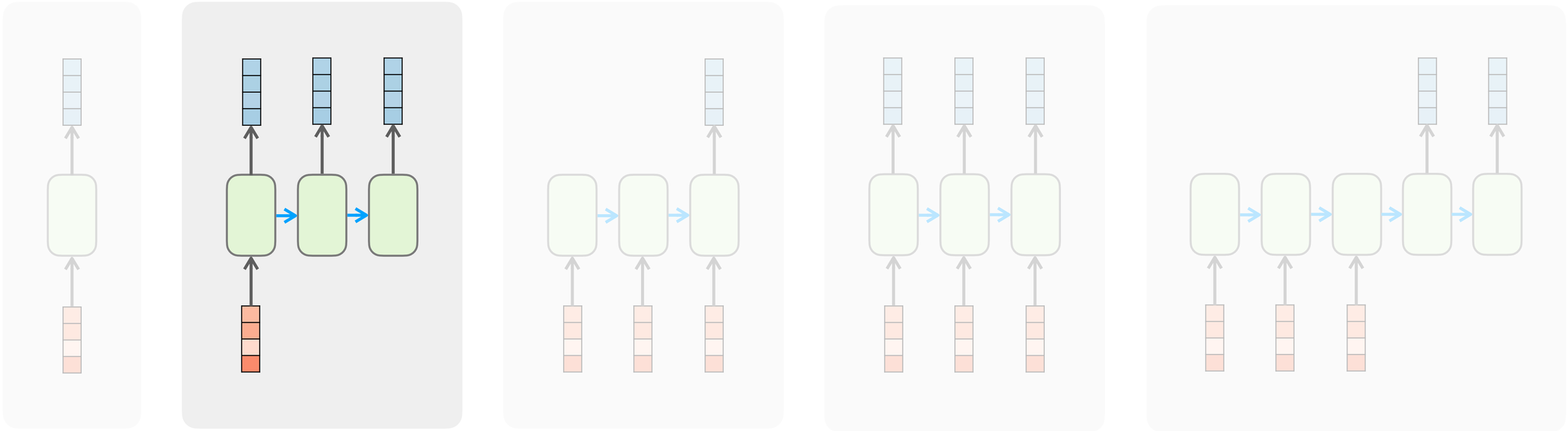
Language is just one type of sequential data. RNNs can be applied to *any* quantifiable sequences.

Diagram based on a slide from Stanford CS231n by Fei-Fei Li et al.

# Seq2Seq

text generation,
image captioning

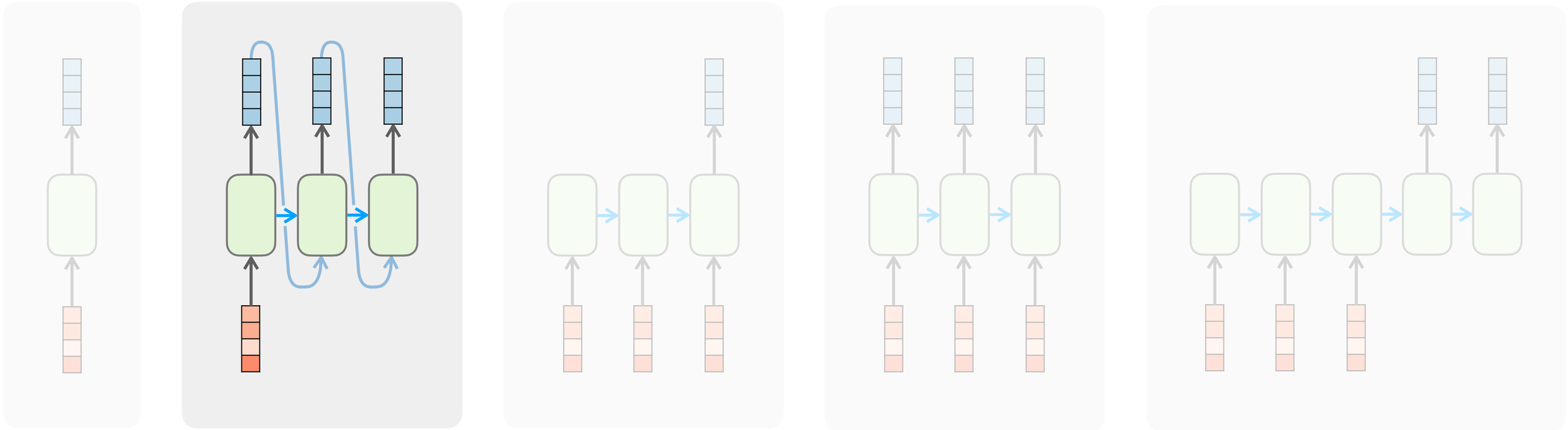
1-to-1 | **1-to-*N*** | *N*-to-1 | *N*-to-*N* | *N*-to-*M*



# Seq2Seq

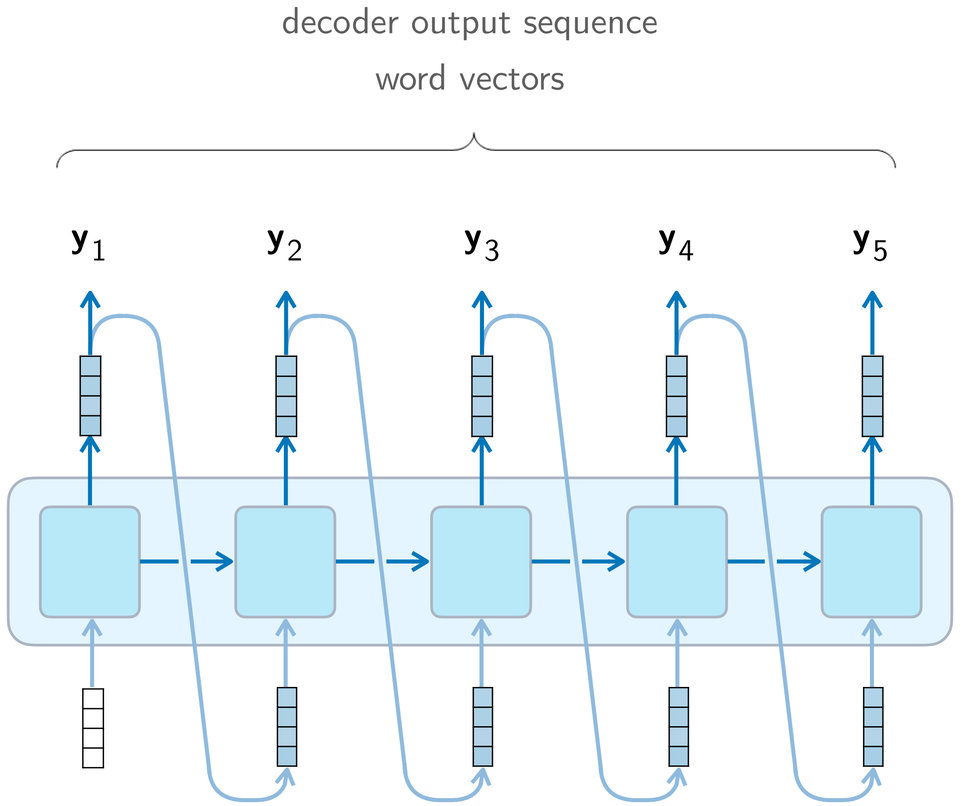
text generation,
image captioning

1-to-1 | 1-to-*N* | *N*-to-1 | *N*-to-*N* | *N*-to-*M*



## 1-to-*N*

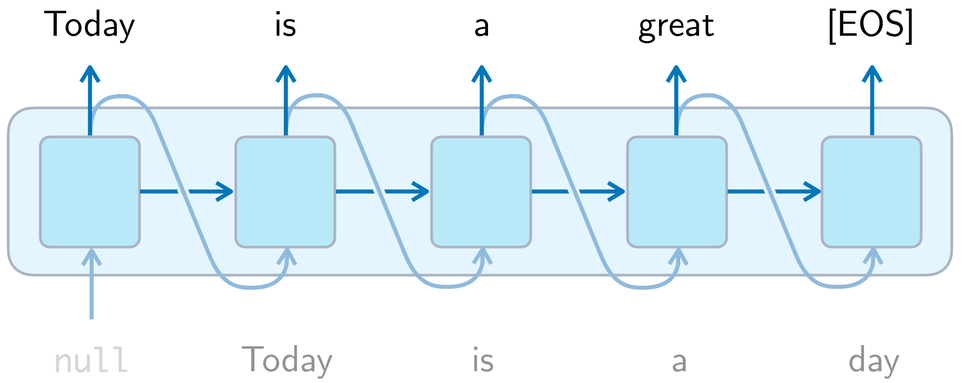
**Example: Text generation**



**1-to-*N***

**Example: Text generation**

(language modeling)

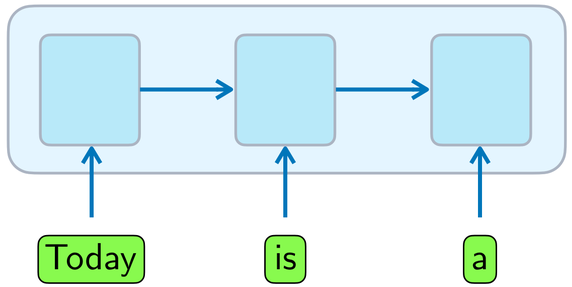


# Seq2Seq

## 1-to-*N*

**Example: Text generation**

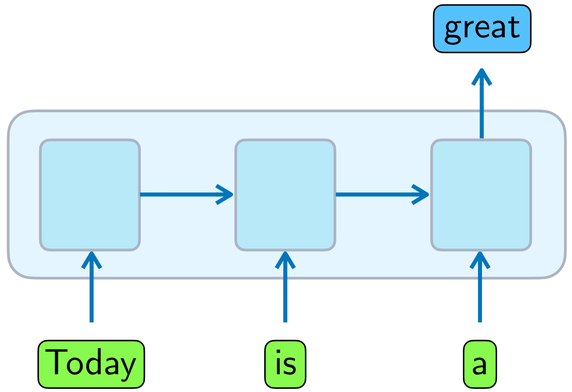
(language modeling)



## 1-to-*N*

**Example: Text generation**

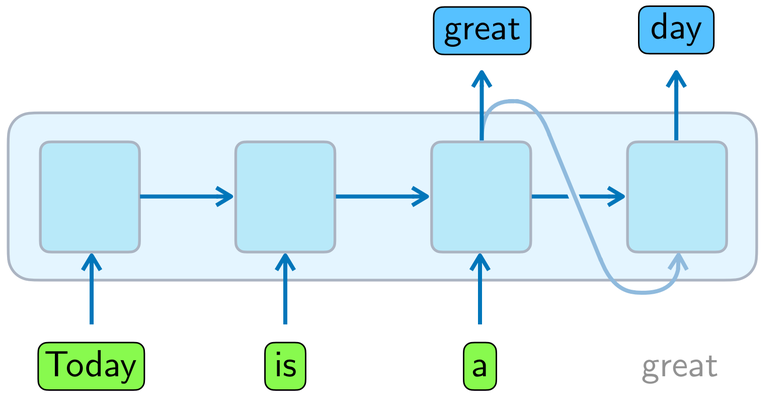
(language modeling)



## 1-to-*N*

**Example: Text generation**

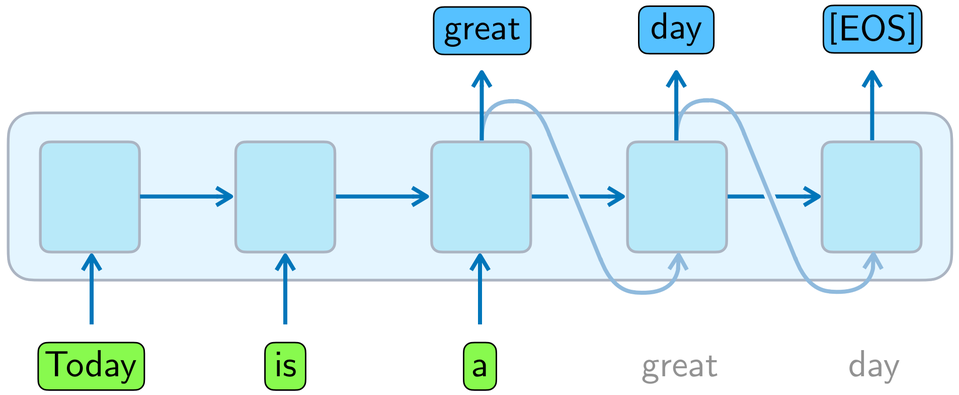
(language modeling)



## 1-to-*N*

**Example: Text generation**

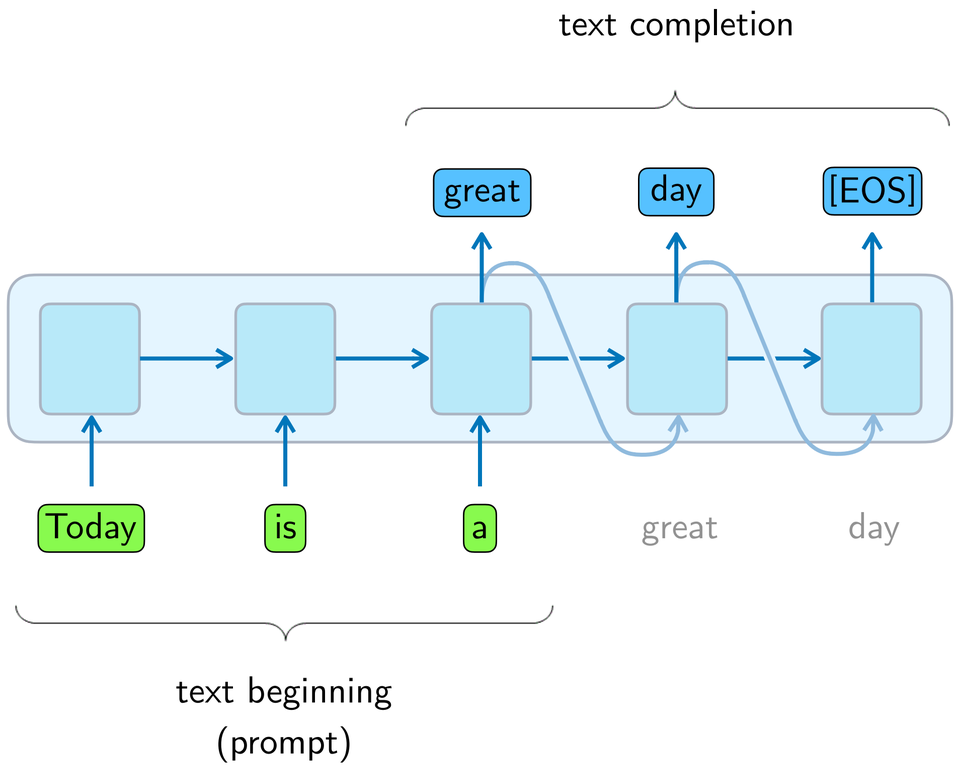
(language modeling)



## 1-to-*N*

**Example: Text generation**

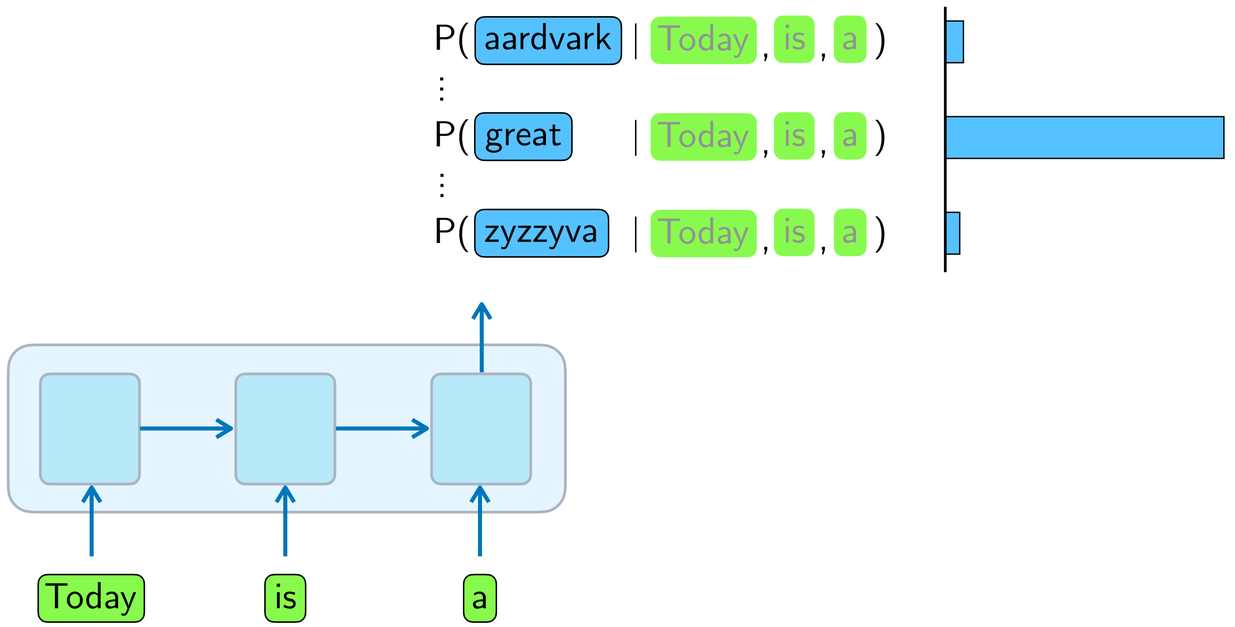
(language modeling)



## 1-to-*N*

**Example: Text generation**

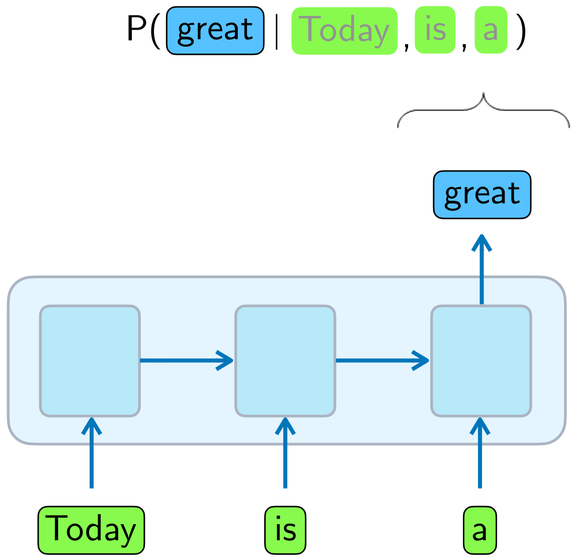
(language modeling)



## 1-to-*N*

**Example: Text generation**

(language modeling)



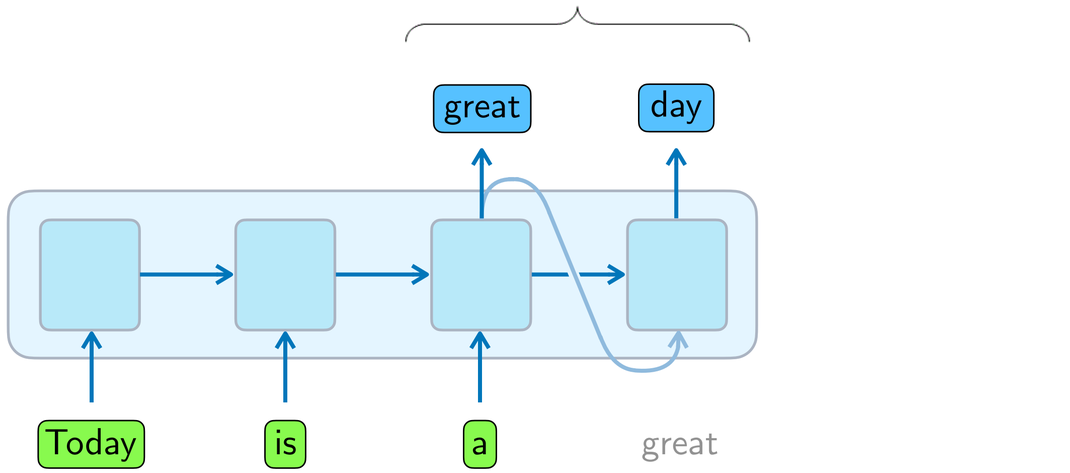
## 1-to-*N*

**Example: Text generation**

(language modeling)

$$P(\text{great}, \text{day} \mid \text{Today}, \text{is}, \text{a})$$

$$= P(\text{great} \mid \text{Today}, \text{is}, \text{a}) \cdot P(\text{day} \mid \text{Today}, \text{is}, \text{a}, \text{great})$$



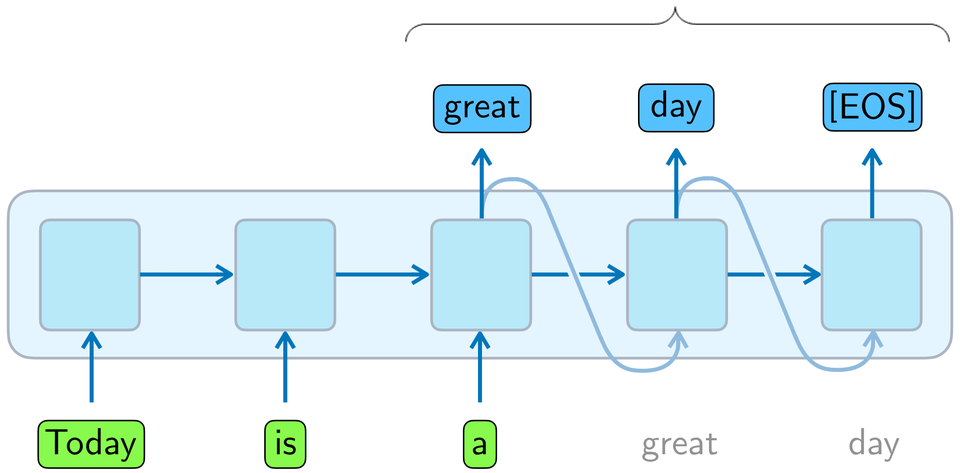
## 1-to-*N*

**Example: Text generation**

(language modeling)

$$P(\text{great}, \text{day}, \text{[EOS]} \mid \text{Today}, \text{is}, \text{a})$$

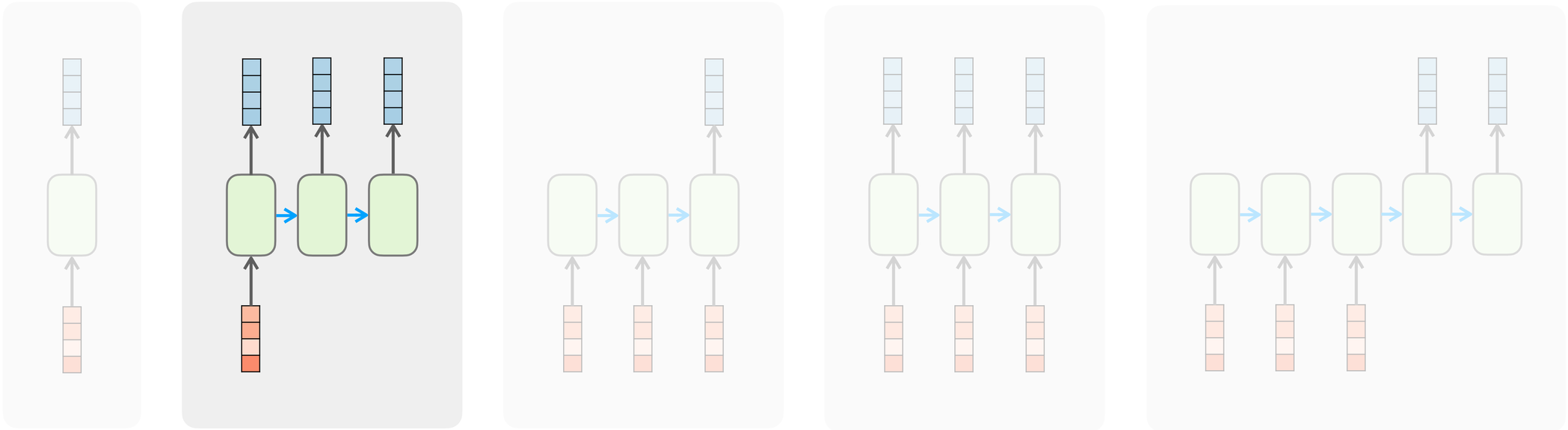
$$= P(\text{great} \mid \text{Today}, \text{is}, \text{a}) \cdot P(\text{day} \mid \text{Today}, \text{is}, \text{a}, \text{great}) \cdot P(\text{[EOS]} \mid \text{Today}, \text{is}, \text{a}, \text{great}, \text{day})$$



# Seq2Seq

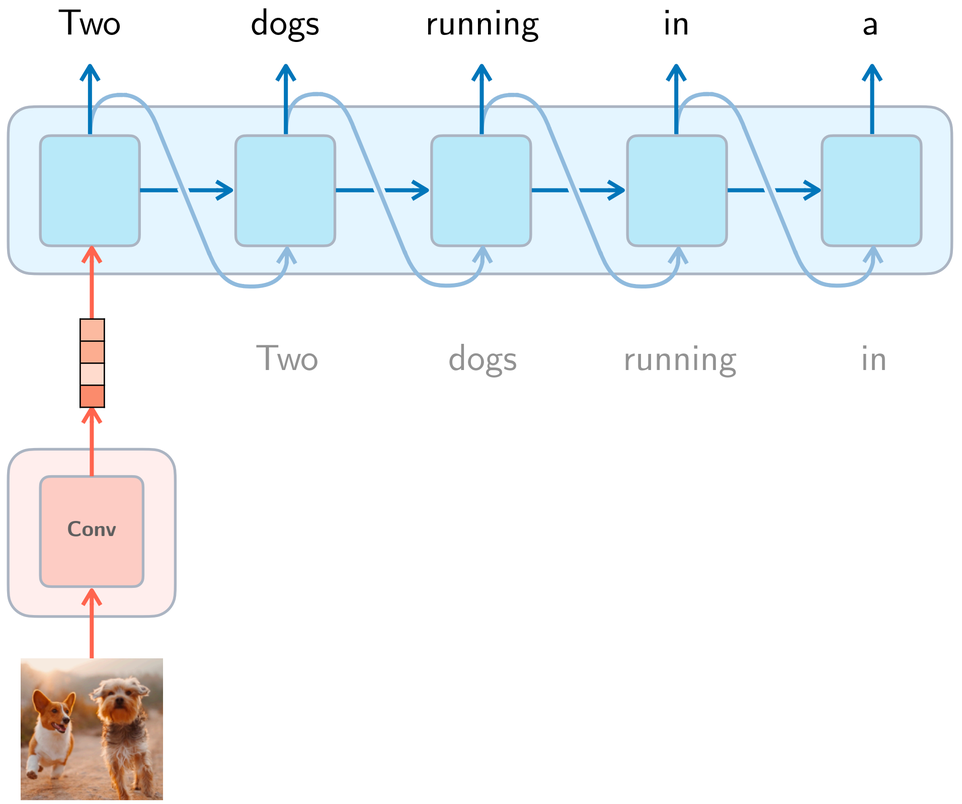
text generation,
image captioning

1-to-1 | **1-to-*N*** | *N*-to-1 | *N*-to-*N* | *N*-to-*M*



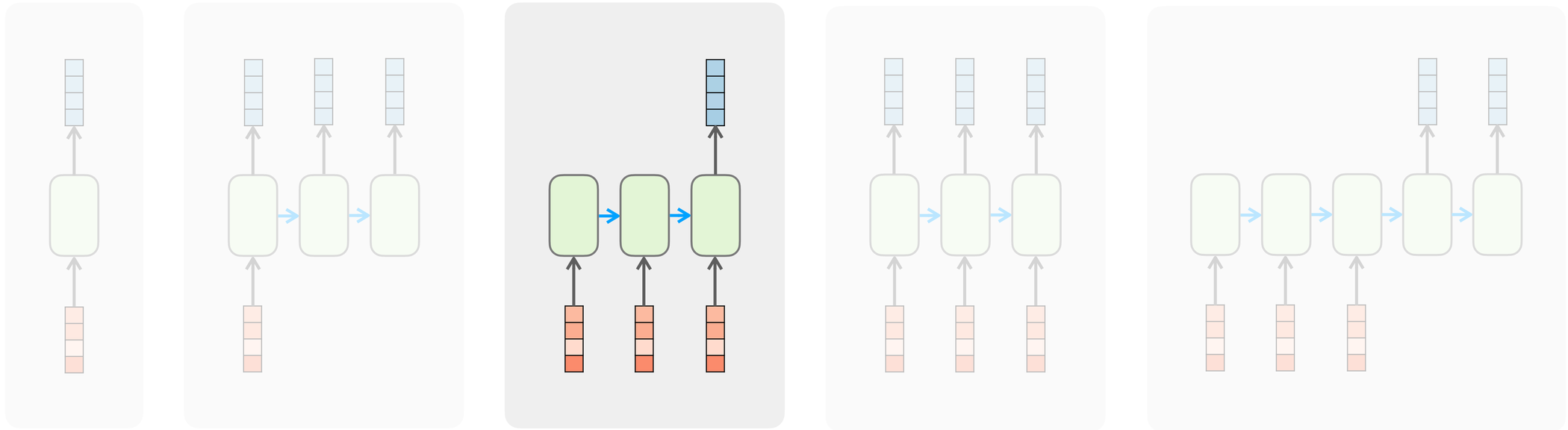
## 1-to-*N*

### Example: Image captioning



text
classification

1-to-1 1-to-*N* ***N*-to-1** *N*-to-*N* *N*-to-*M*



text

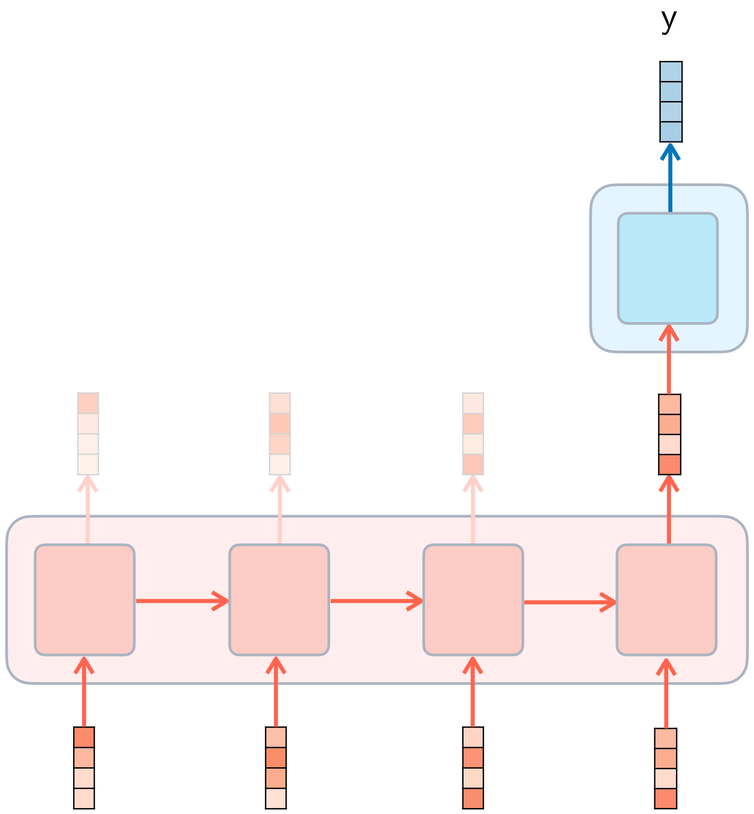
## *N*-to-1

Example: sentiment classification

A little bit naive: there's a better way of mapping the sequence to a single value.

Let's first look at *N*-to-*N* and *N*-to-*M*.

Probability that the text is *positive*.



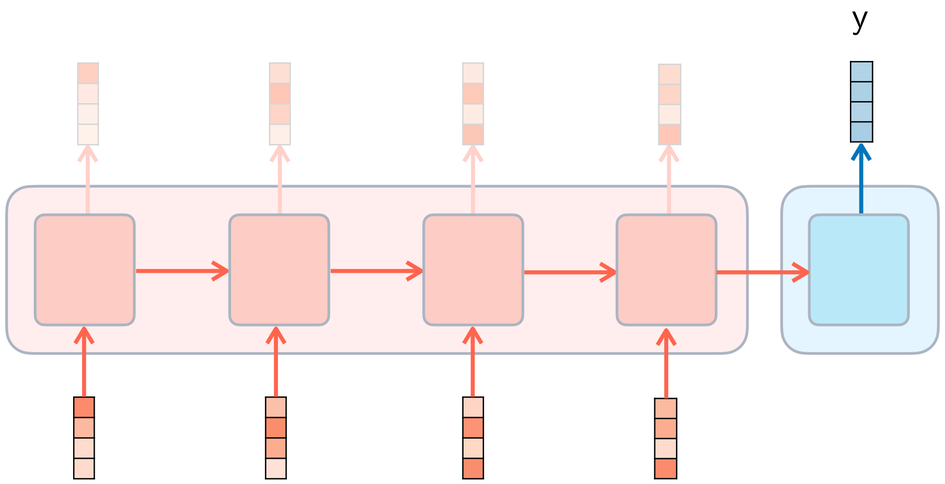
## *N*-to-1

Example: sentiment classification

A little bit naive: there's a better way of mapping the sequence to a single value.

Let's first look at *N*-to-*N* and *N*-to-*M*.

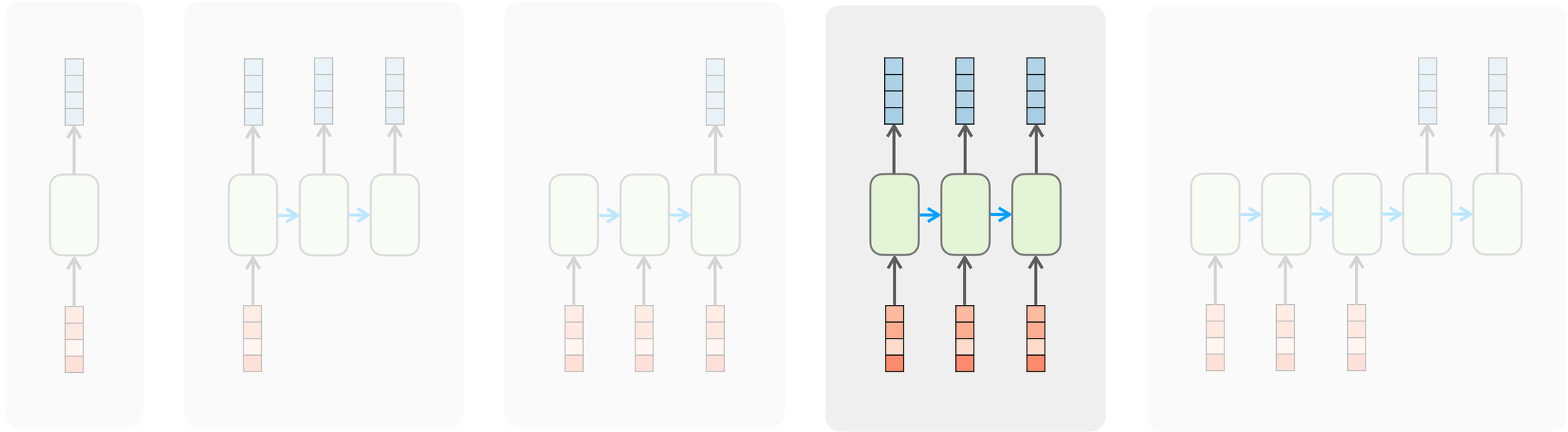
Probability that the text is *positive*.



(Refine sequence $S$ to $S'$)

new representation
of text

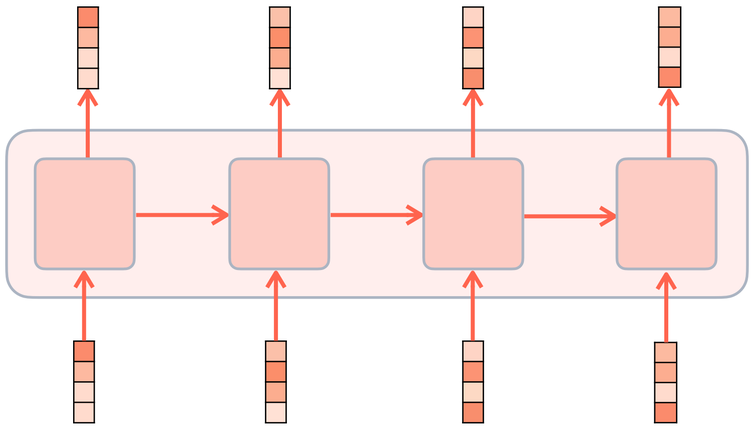
1-to-1 | 1-to-$N$ | $N$-to-1 | **$N$-to-$N$** | $N$-to-$M$



text

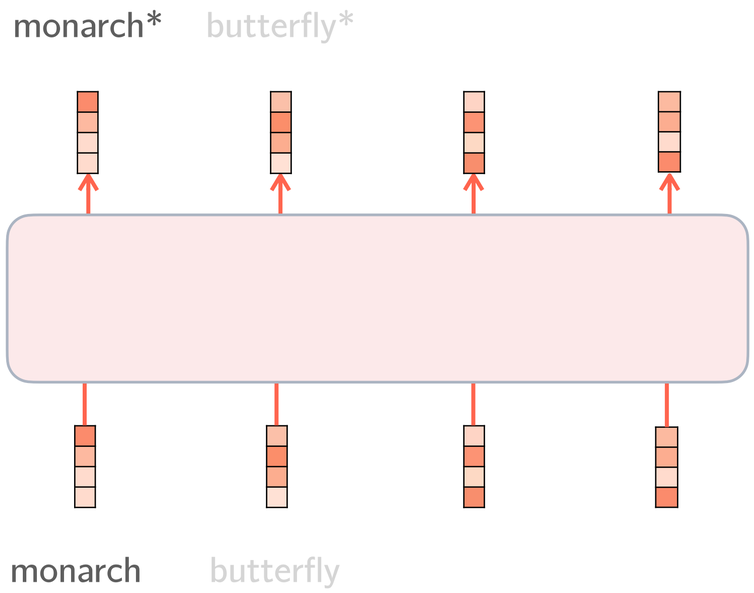
## *N*-to-*N*

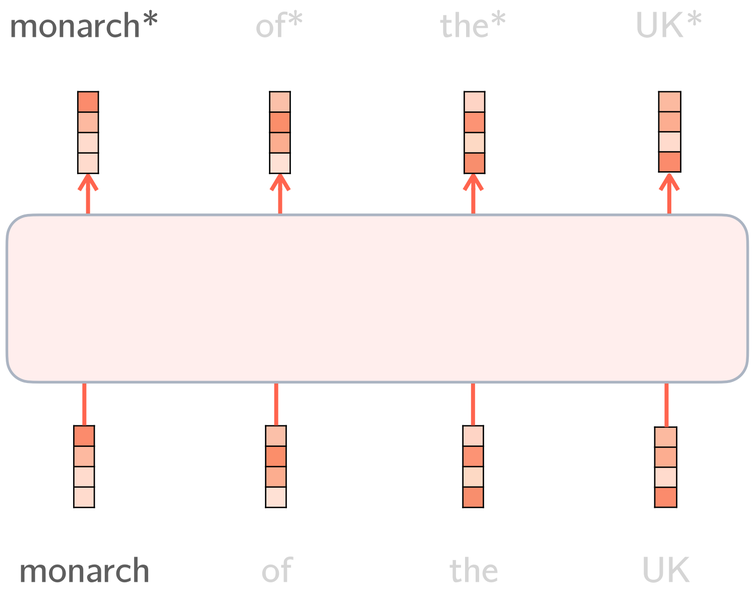
Why? e.g. to create a **contextualized representation** of our sequence



## *N*-to-*N*

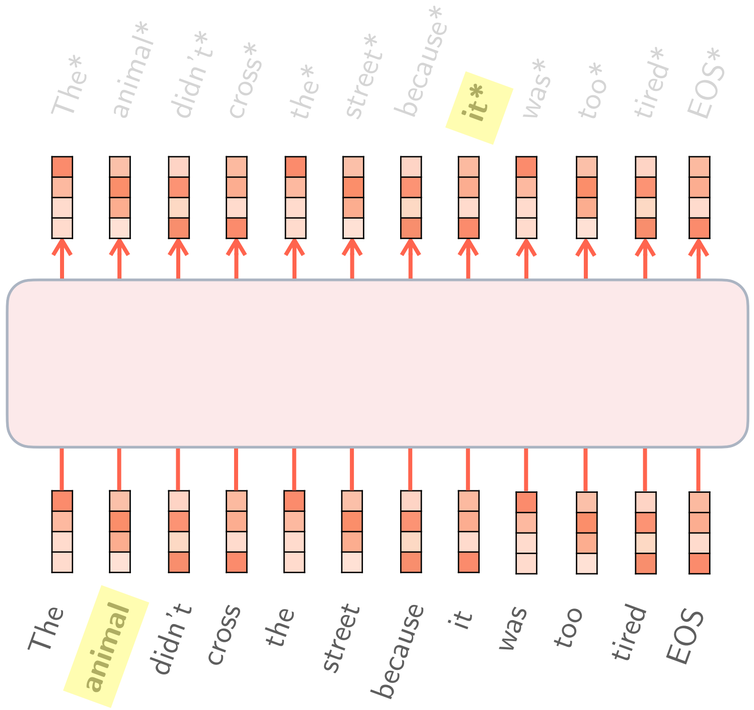
Why? e.g. to create a **contextualized representation** of our sequence

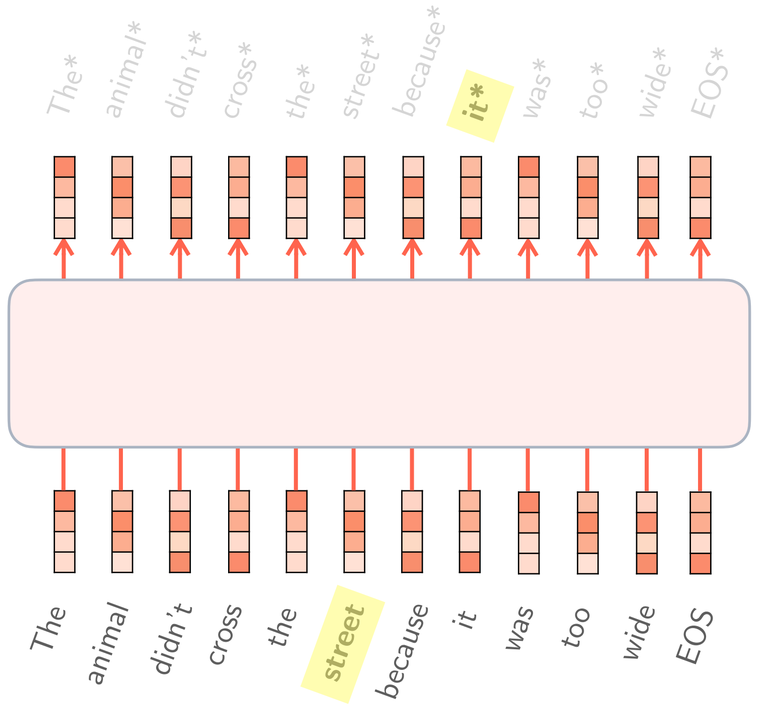


## *N*-to-*N*

Why? e.g. to create a **contextualized representation** of our sequence



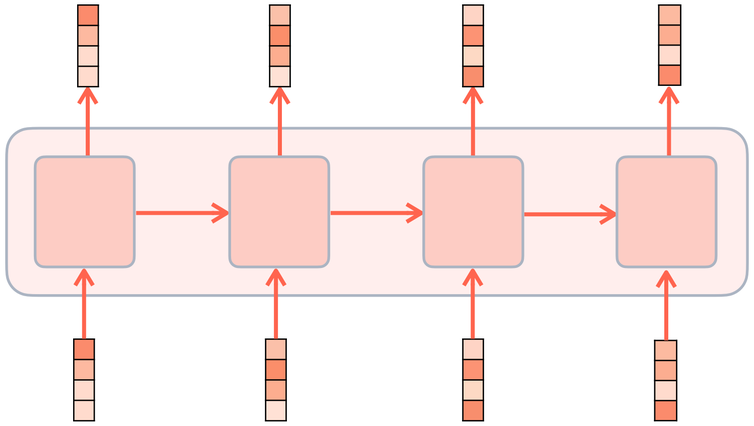




https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

### *N*-to-*N*

Why? e.g. to create a **contextualized representation** of our sequence

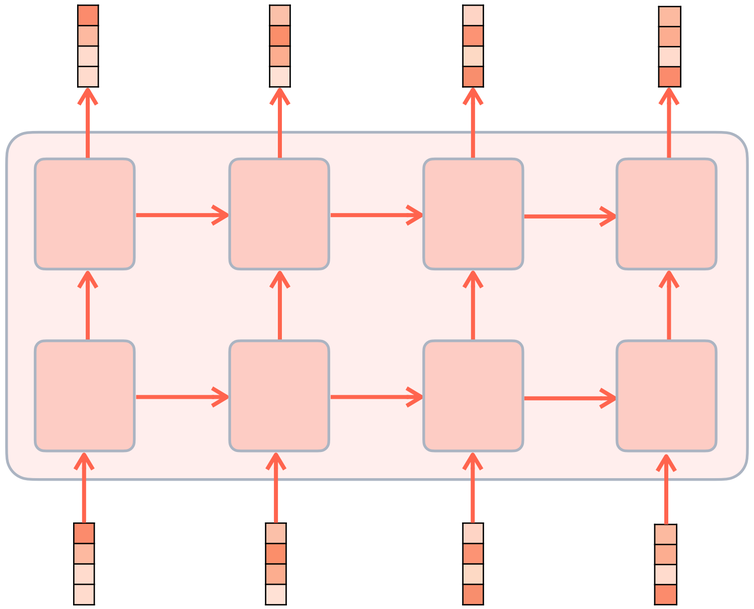
*Stacked RNN*



## *N*-to-*N*

Why? e.g. to create a **contextualized representation** of our sequence

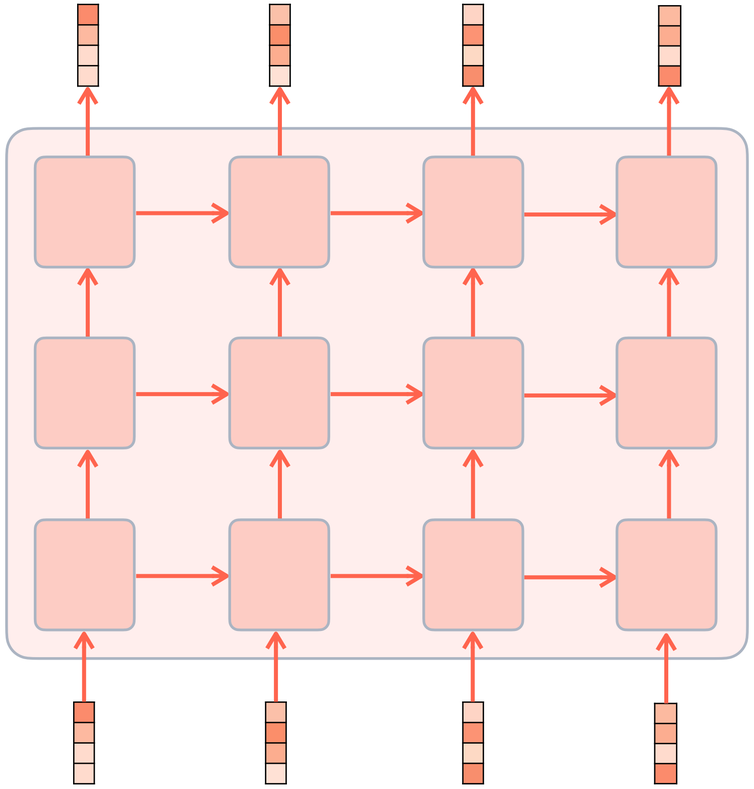
*Stacked RNN*



## *N*-to-*N*

Why? e.g. to create a **contextualized representation** of our sequence

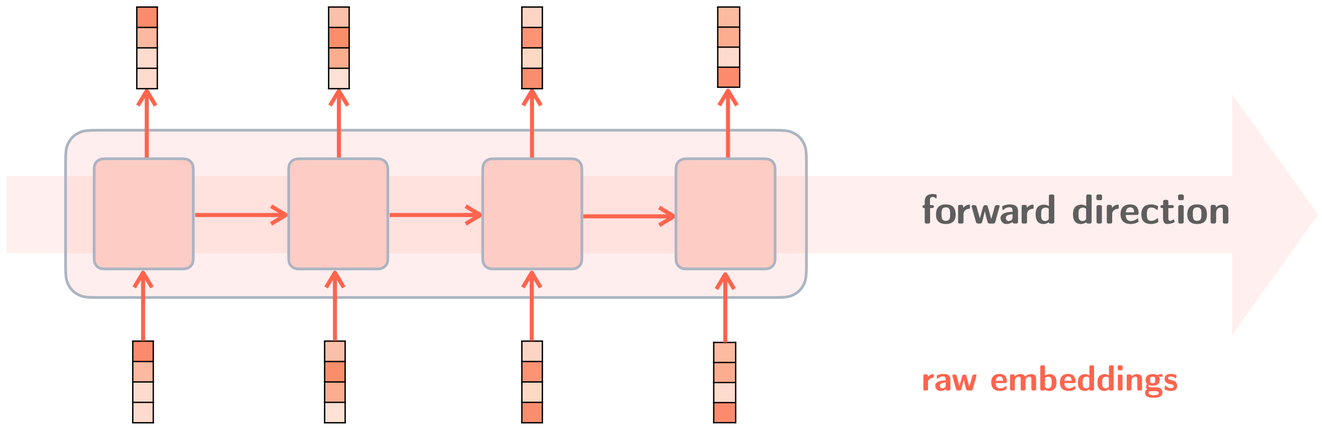
*Stacked RNN*



### *N*-to-*N*

Why? e.g. to create a **contextualized representation** of our sequence

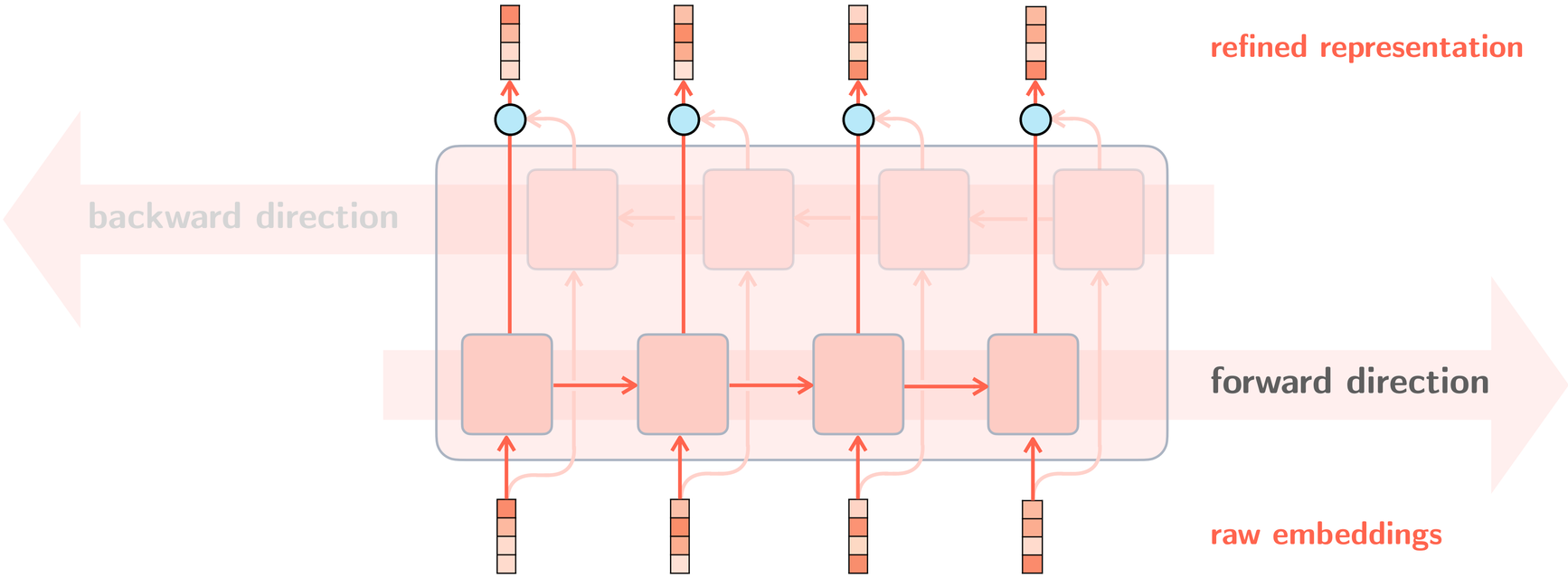
*Bidirectional RNN*



### *N*-to-*N*

Why? e.g. to create a **contextualized representation** of our sequence

*Bidirectional RNN*

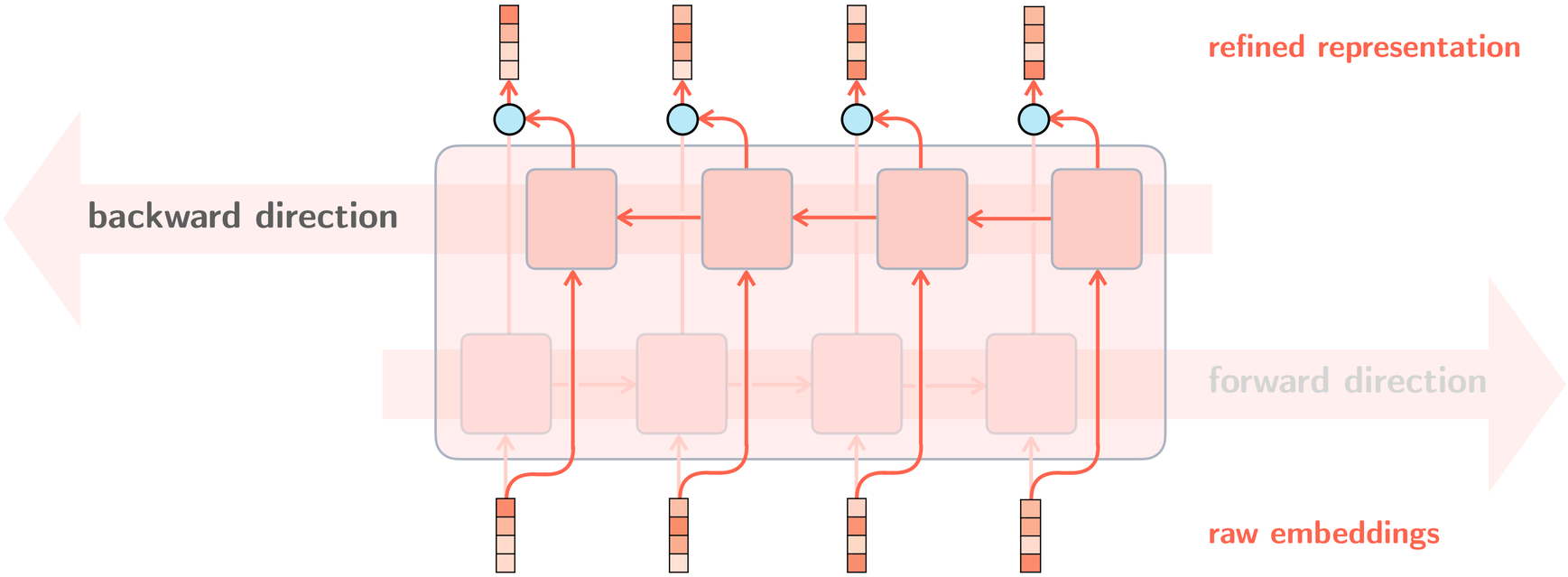



concat

### *N*-to-*N*

Why? e.g. to create a **contextualized representation** of our sequence

*Bidirectional RNN*

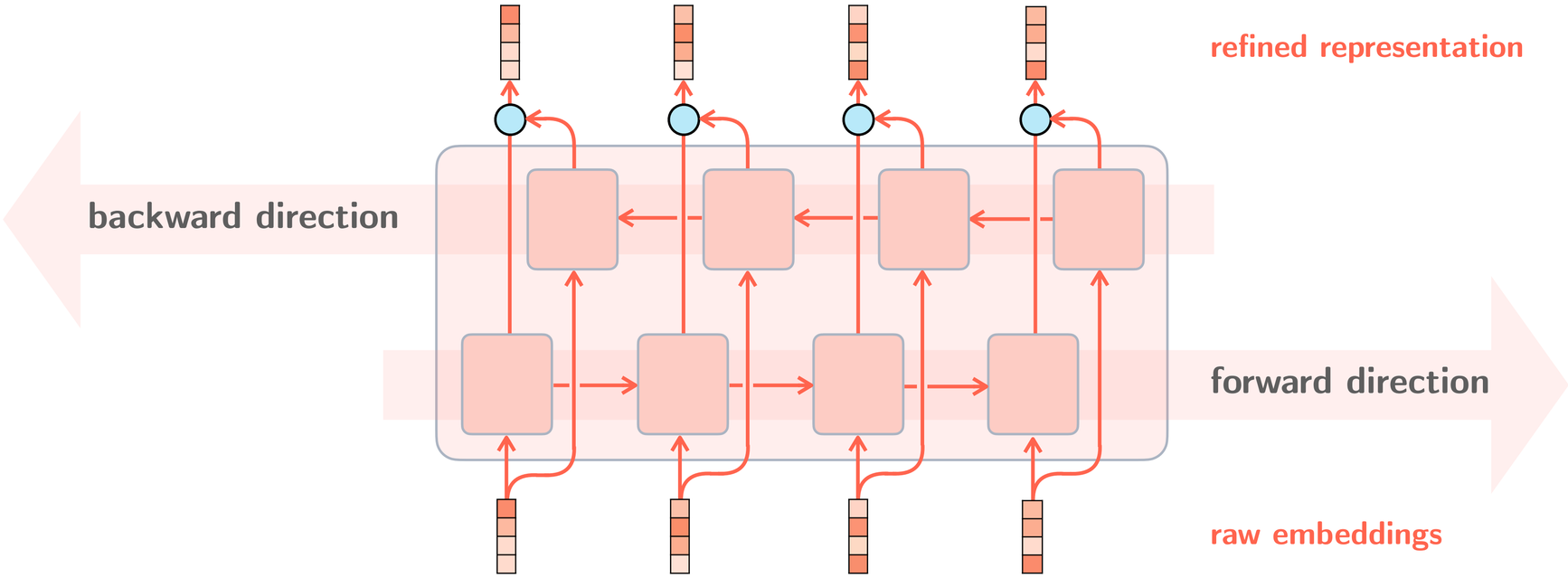



concat

### *N*-to-*N*

Why? e.g. to create a **contextualized representation** of our sequence

*Bidirectional RNN*

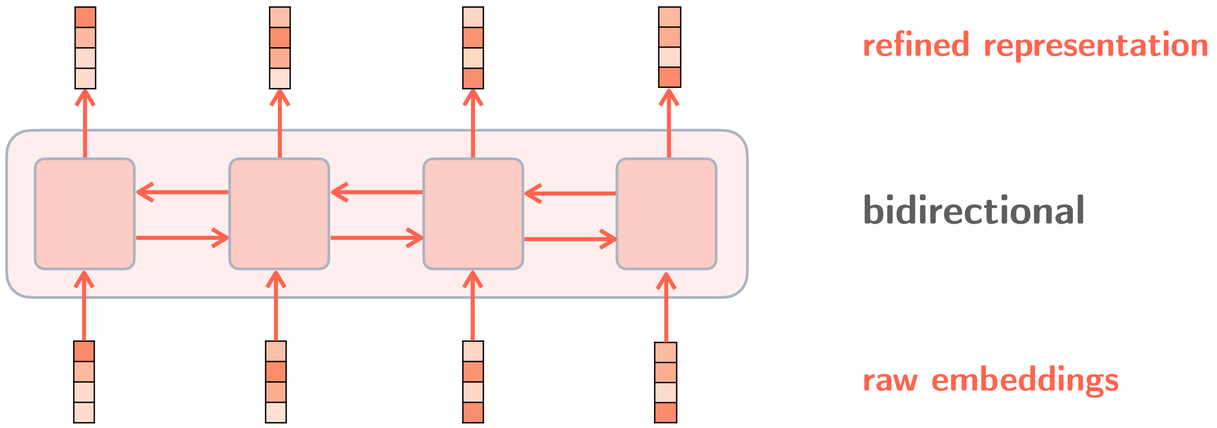



concat

### *N*-to-*N*

Why? e.g. to create a **contextualized representation** of our sequence

*Bidirectional RNN*



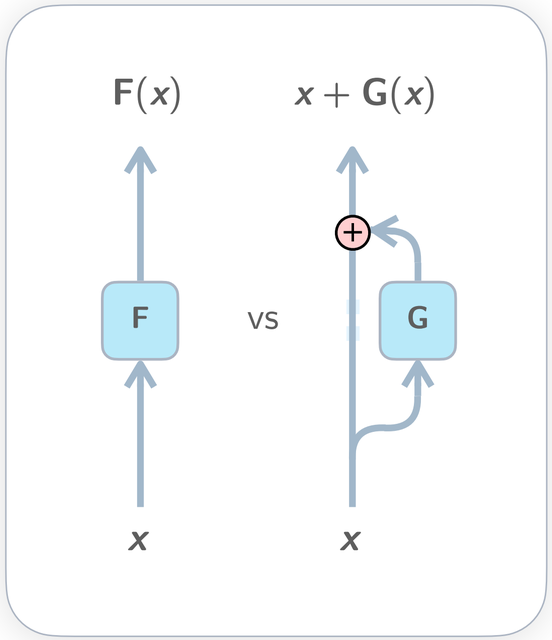
## *N*-to-*N*

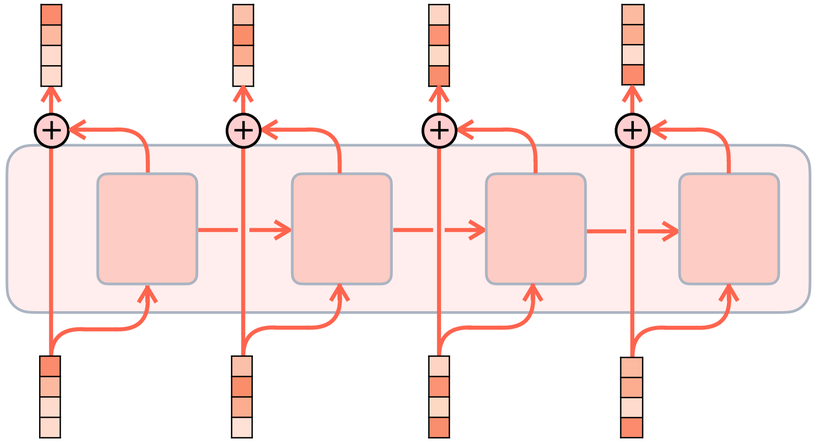
Why? e.g. to create a **contextualized representation** of our sequence

*Residual RNN*

RNN cell reads and writes information to the residual stream

Residual connections make sense if you ***refine*** a representation in some way, and if the network is deep
Not so much if you map an input to a very different output



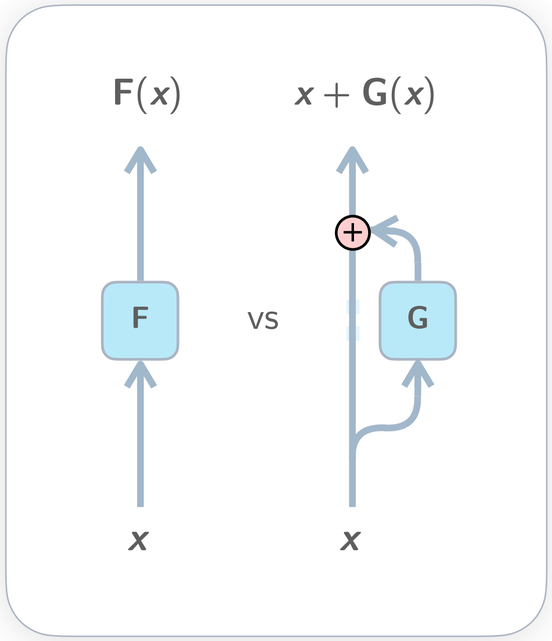



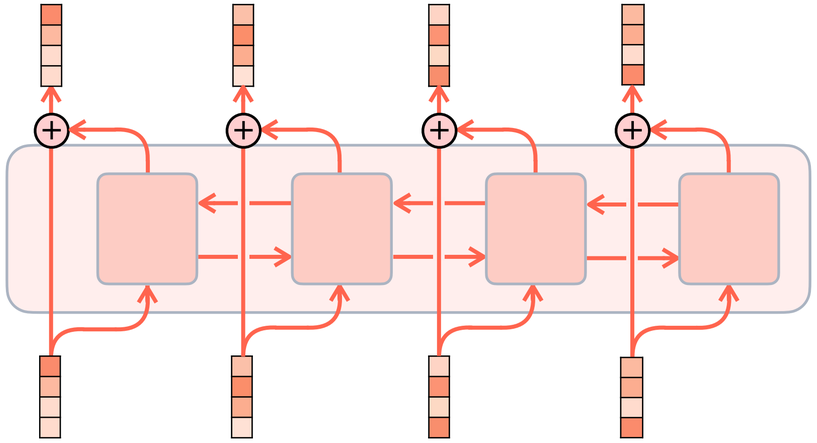
***N*-to-*N***

Why? e.g. to create a **contextualized representation** of our sequence

*Residual RNN*

RNN cell reads and writes information to the residual stream



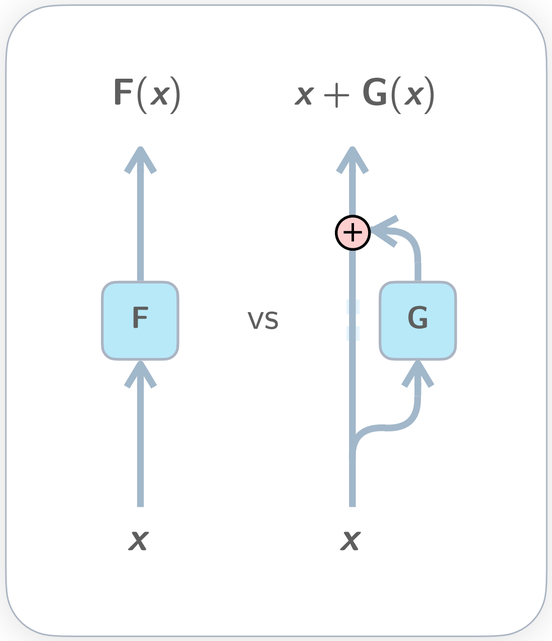



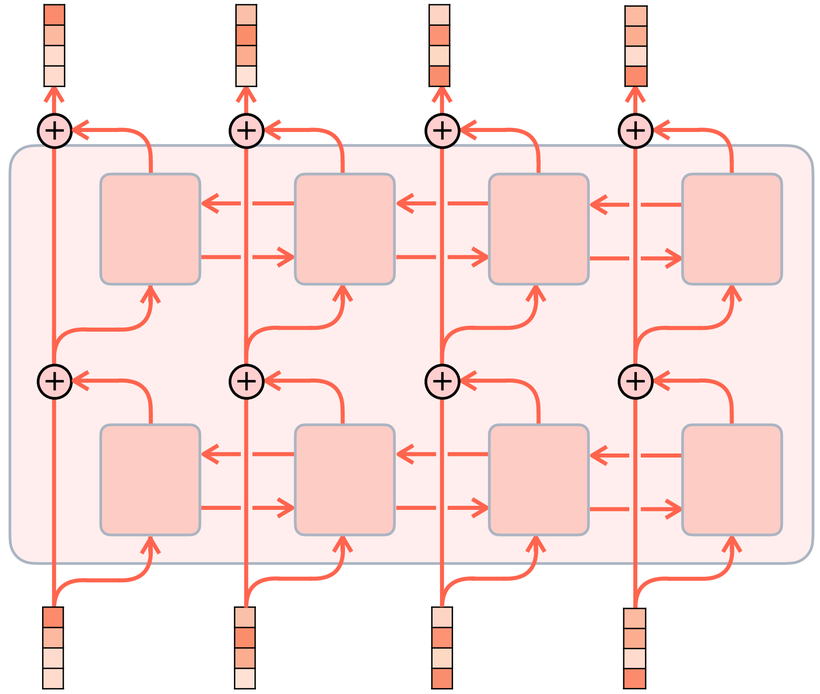
## *N*-to-*N*

Why? e.g. to create a **contextualized representation** of our sequence

*Residual RNN*

RNN cell reads and writes information to the residual stream



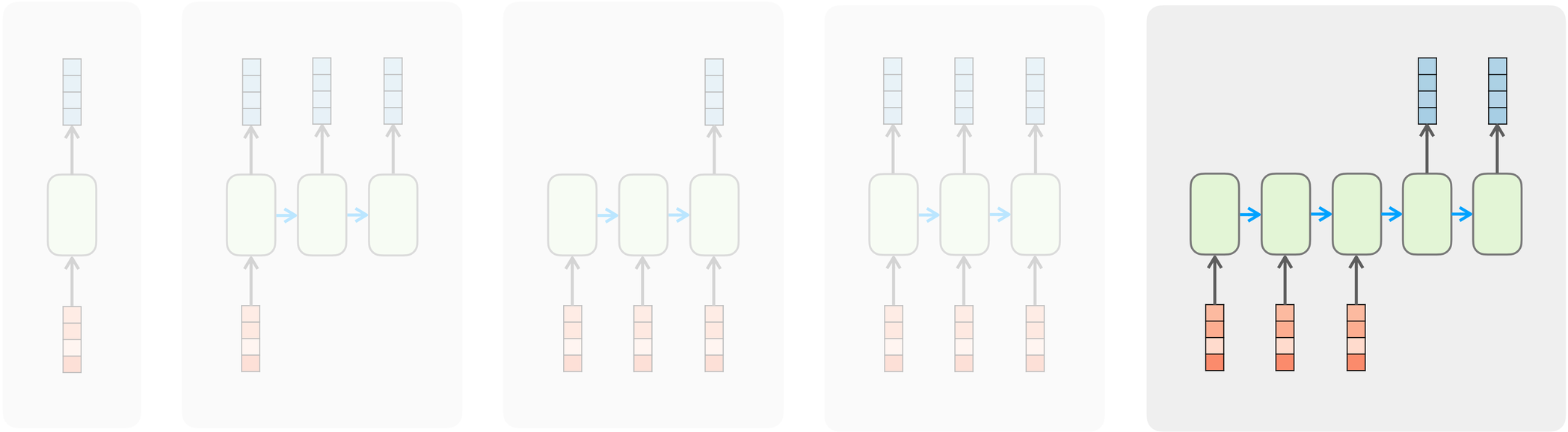



RNN encoder architecture of the 2016 Google Neural Machine Translation aka **GNMT** (Wu et al. 2016)

# Recurrent Neural Networks & Seq2Seq

text in language *B*

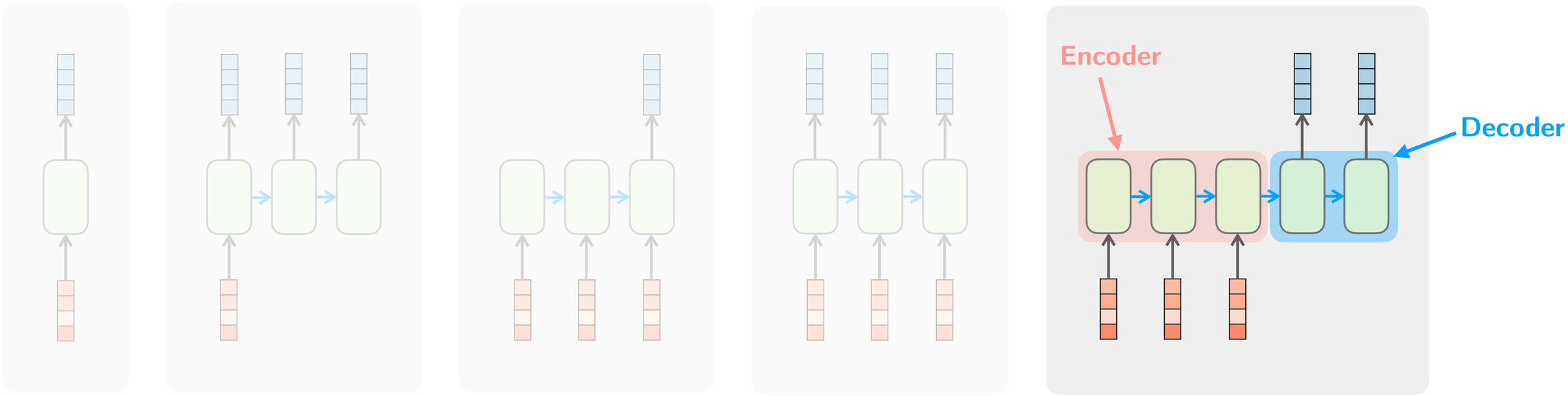
1-to-1 | 1-to-*N* | *N*-to-1 | *N*-to-*N* | ***N*-to-*M***



text in language *A*

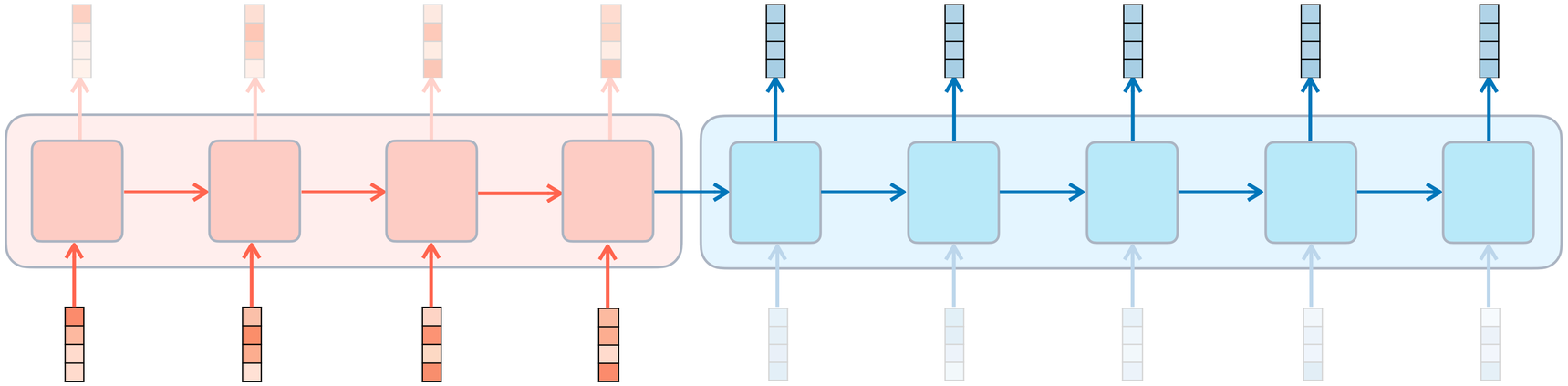
# Recurrent Neural Networks & Seq2Seq

1-to-1 | 1-to-*N* | *N*-to-1 | *N*-to-*N* | ***N*-to-*M***

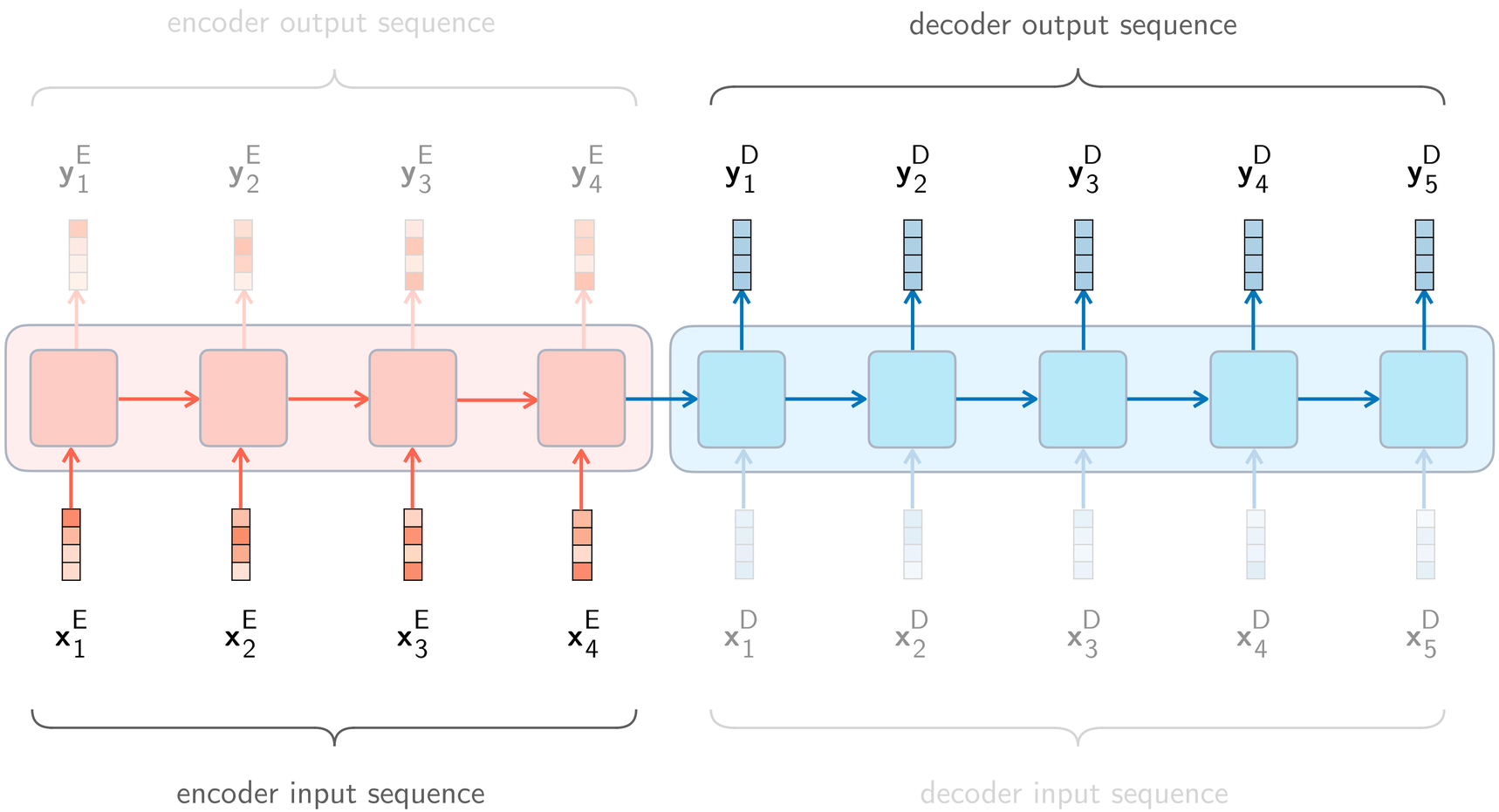


# Seq2Seq

## *N*-to-*M*

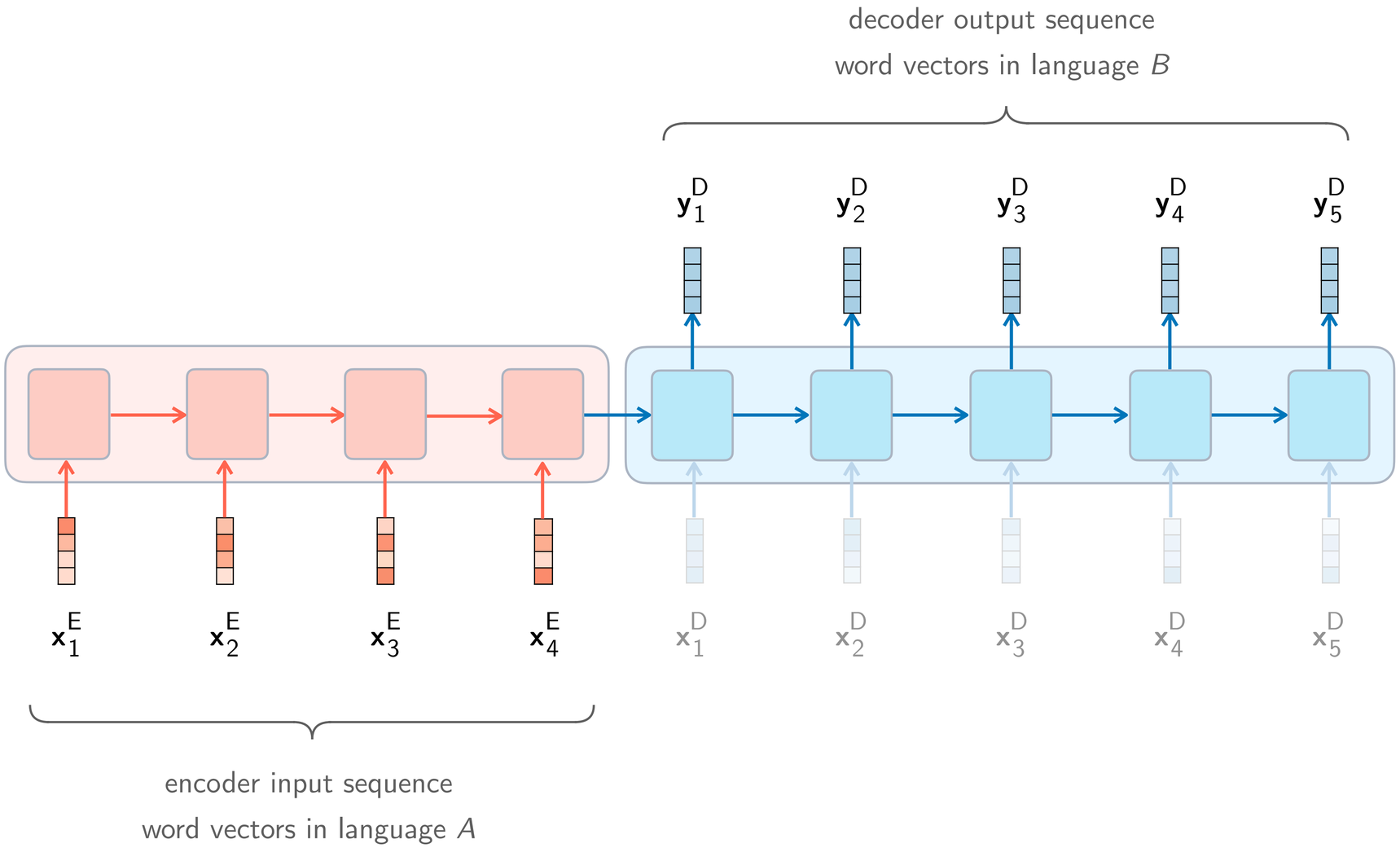


## *N*-to-*M*


encoder output sequence
decoder output sequence
$\mathbf{y}_1^E$ $\mathbf{y}_2^E$ $\mathbf{y}_3^E$ $\mathbf{y}_4^E$
$\mathbf{y}_1^D$ $\mathbf{y}_2^D$ $\mathbf{y}_3^D$ $\mathbf{y}_4^D$ $\mathbf{y}_5^D$
$\mathbf{x}_1^E$ $\mathbf{x}_2^E$ $\mathbf{x}_3^E$ $\mathbf{x}_4^E$
$\mathbf{x}_1^D$ $\mathbf{x}_2^D$ $\mathbf{x}_3^D$ $\mathbf{x}_4^D$ $\mathbf{x}_5^D$
encoder input sequence
decoder input sequence

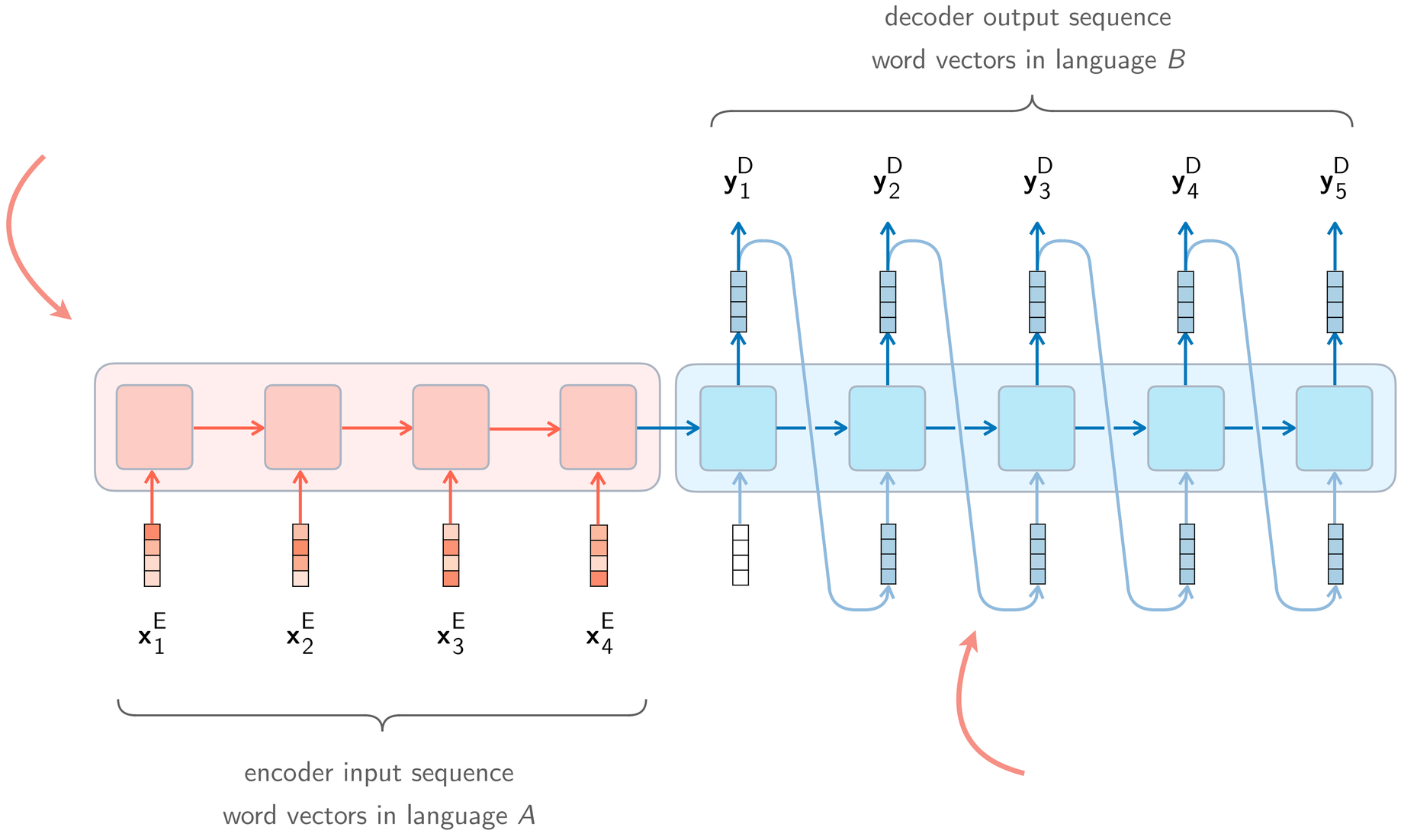
*N*-to-*M*

**Example: Translation**



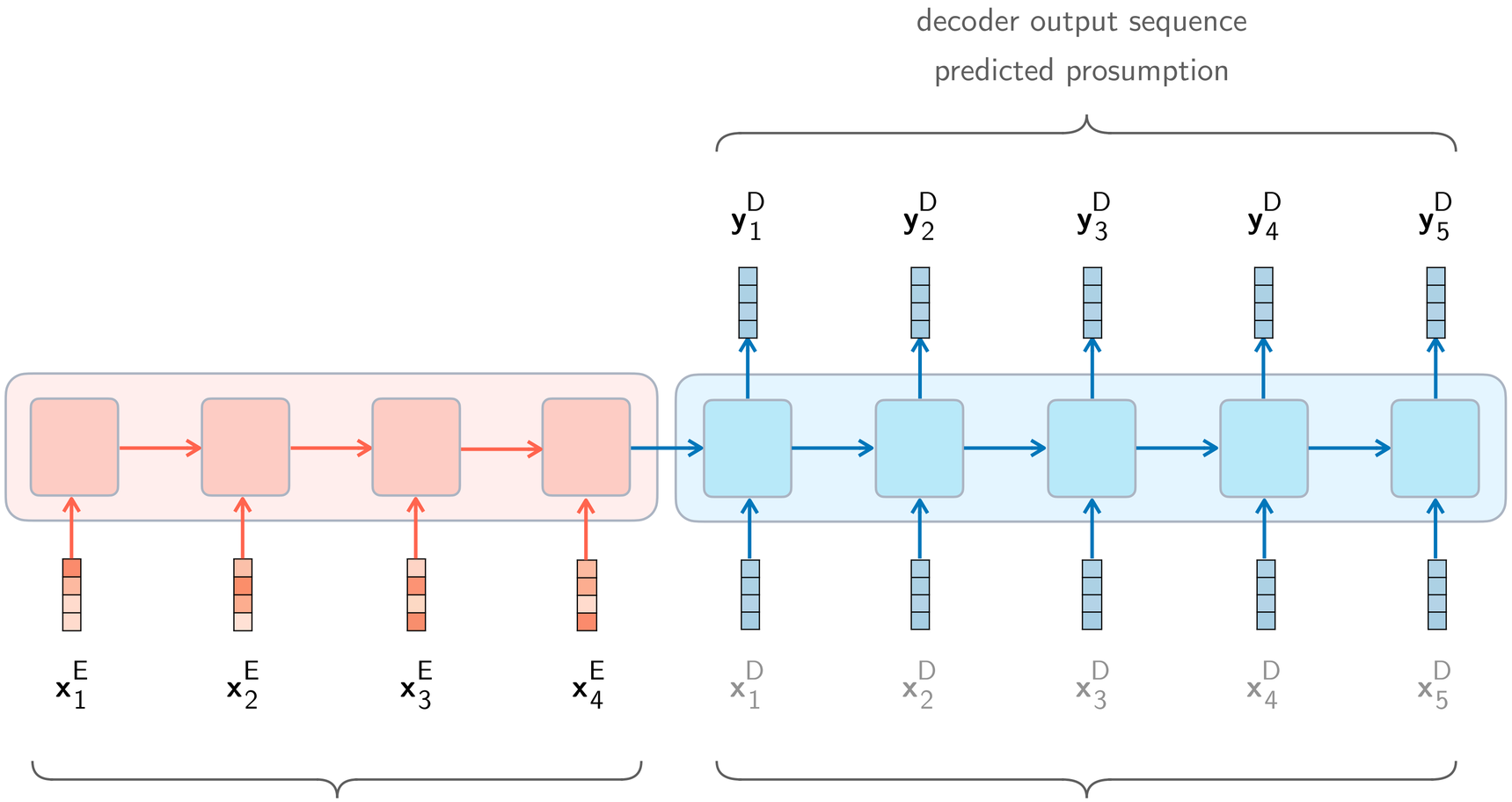
*N*-to-*M*

**Example: Translation**

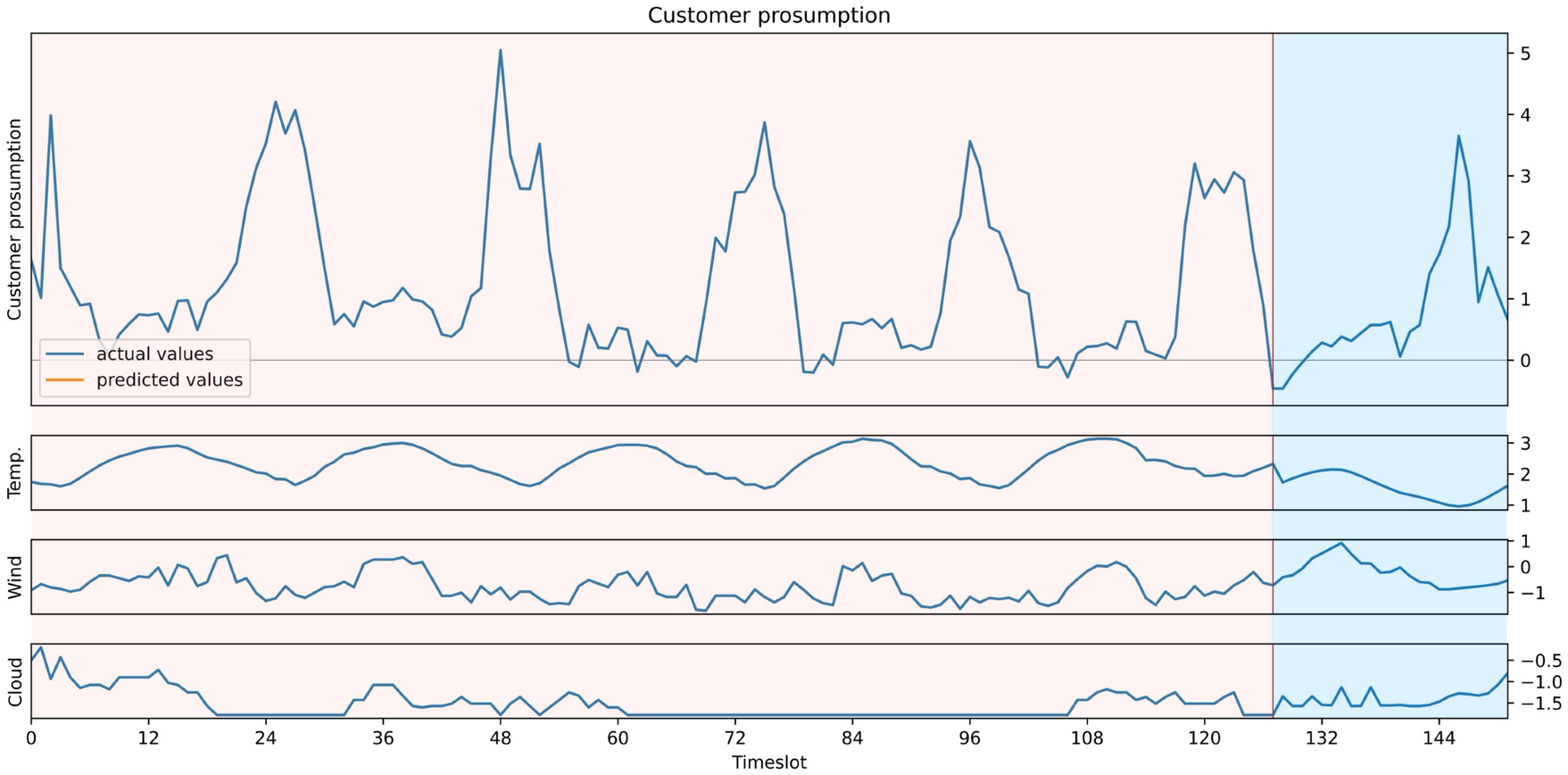


## *N*-to-*M*

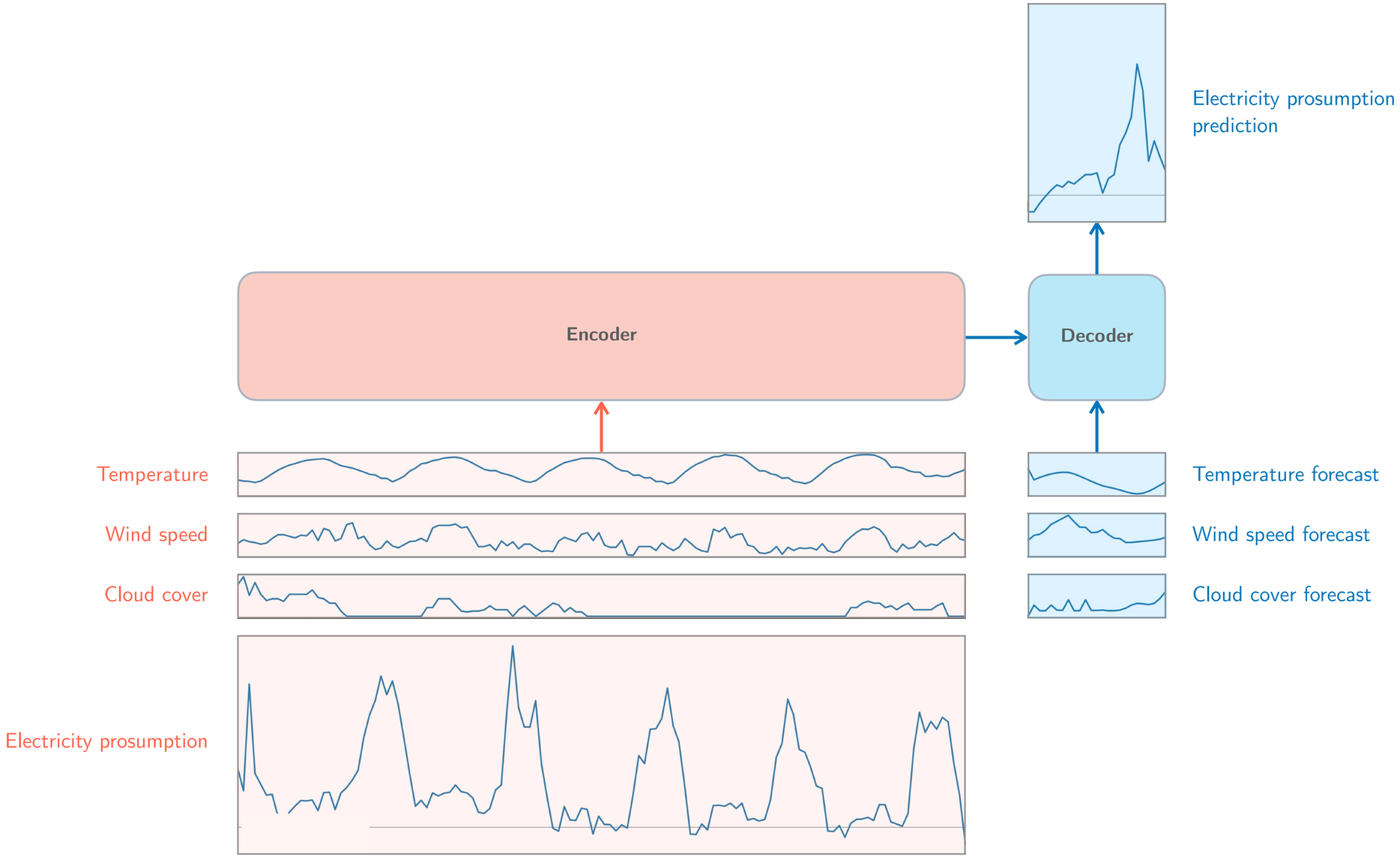
### Example: Predicting electricity markets



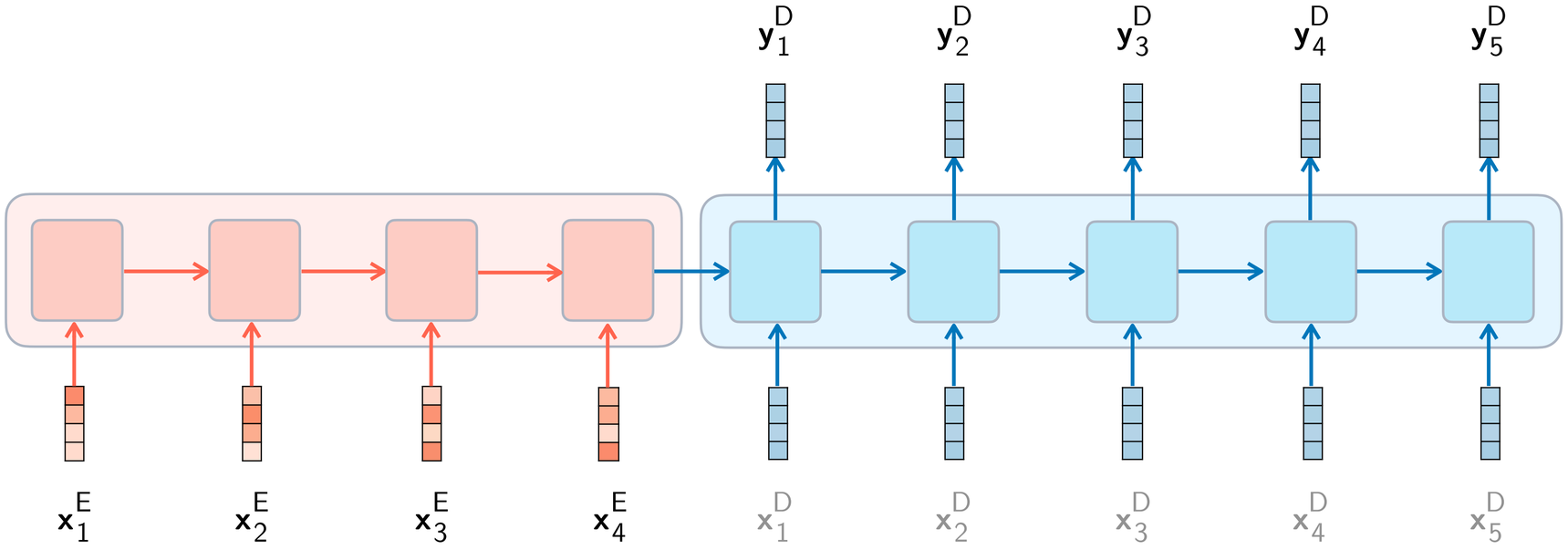
# Example: Electricity Market



# Example: Electricity Market



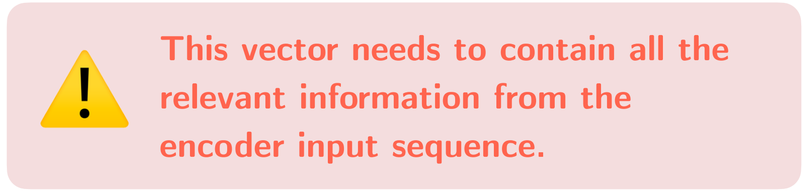
*N*-to-*M*


$\mathbf{y}_1^D$ $\mathbf{y}_2^D$ $\mathbf{y}_3^D$ $\mathbf{y}_4^D$ $\mathbf{y}_5^D$
$\mathbf{x}_1^E$ $\mathbf{x}_2^E$ $\mathbf{x}_3^E$ $\mathbf{x}_4^E$
$\mathbf{x}_1^D$ $\mathbf{x}_2^D$ $\mathbf{x}_3^D$ $\mathbf{x}_4^D$ $\mathbf{x}_5^D$

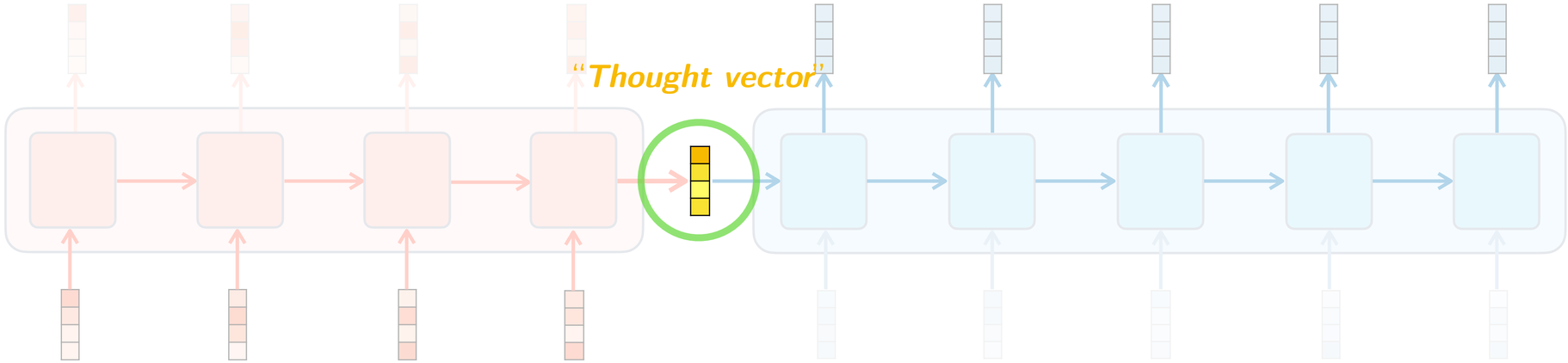
## *N*-to-*M*

How to best get encoder information into decoder?

### Bottleneck

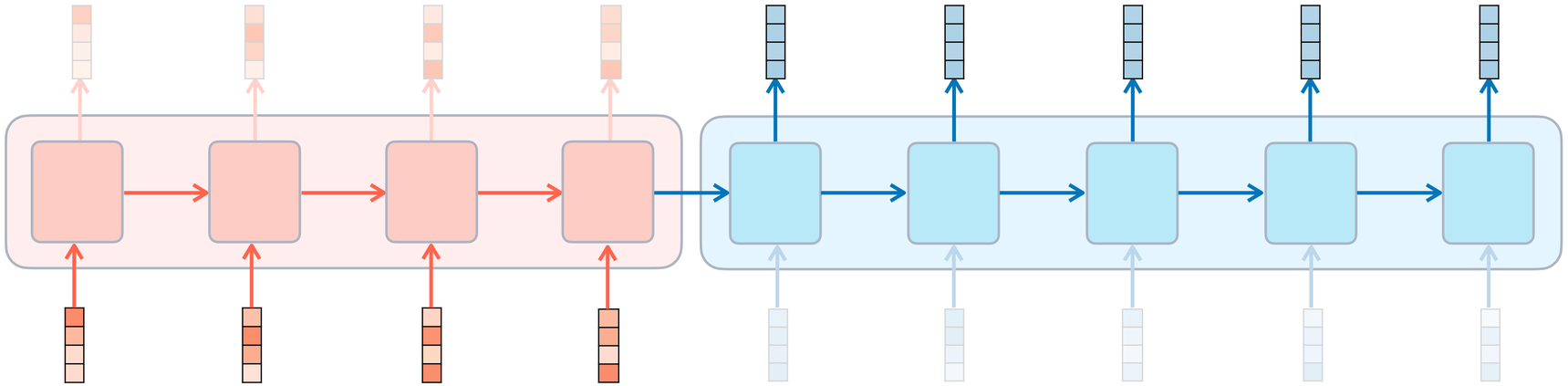


**This vector needs to contain all the relevant information from the encoder input sequence.**



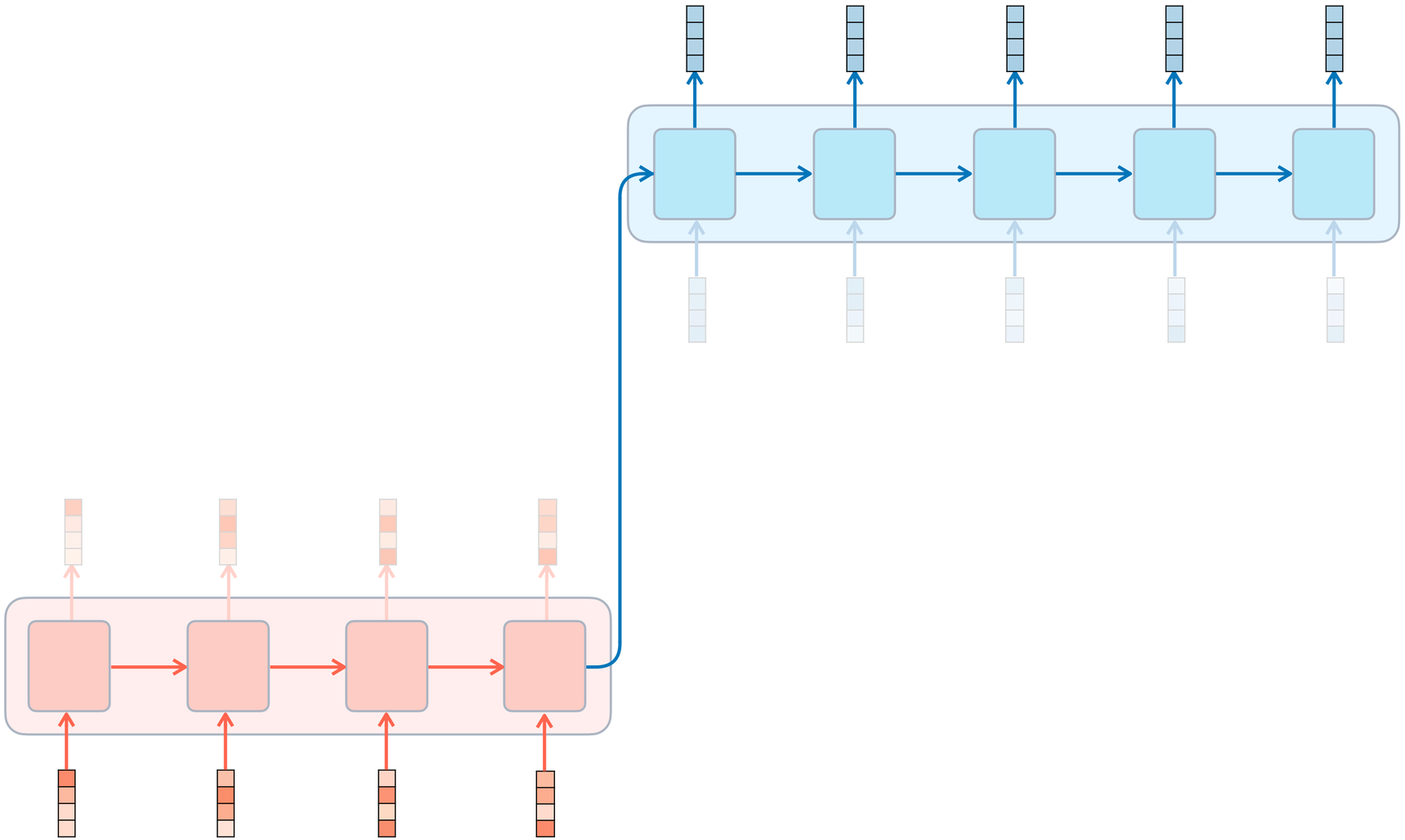
# Seq2Seq + Context Vector

*N*-to-*M*



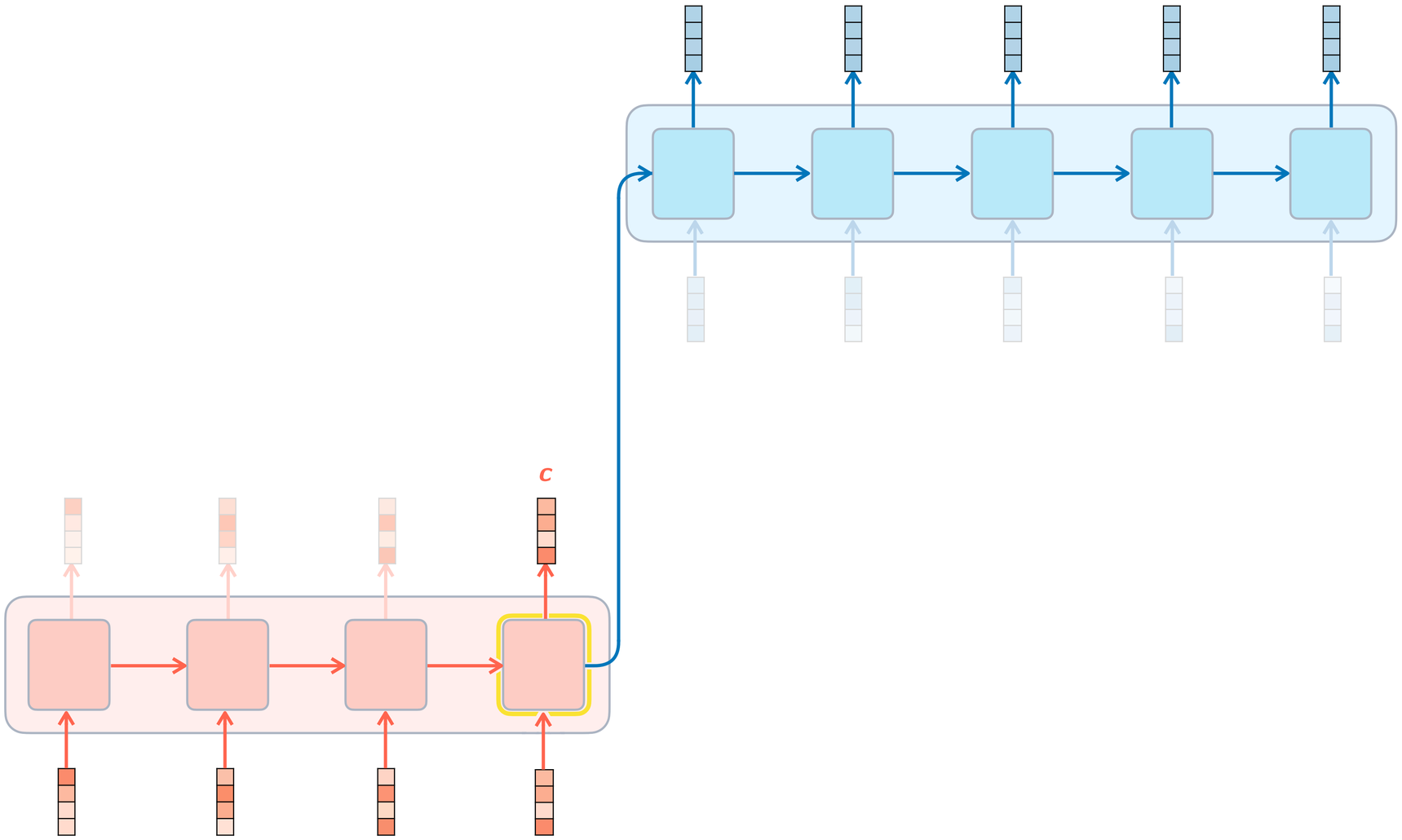
# Seq2Seq + Context Vector

*N*-to-*M*



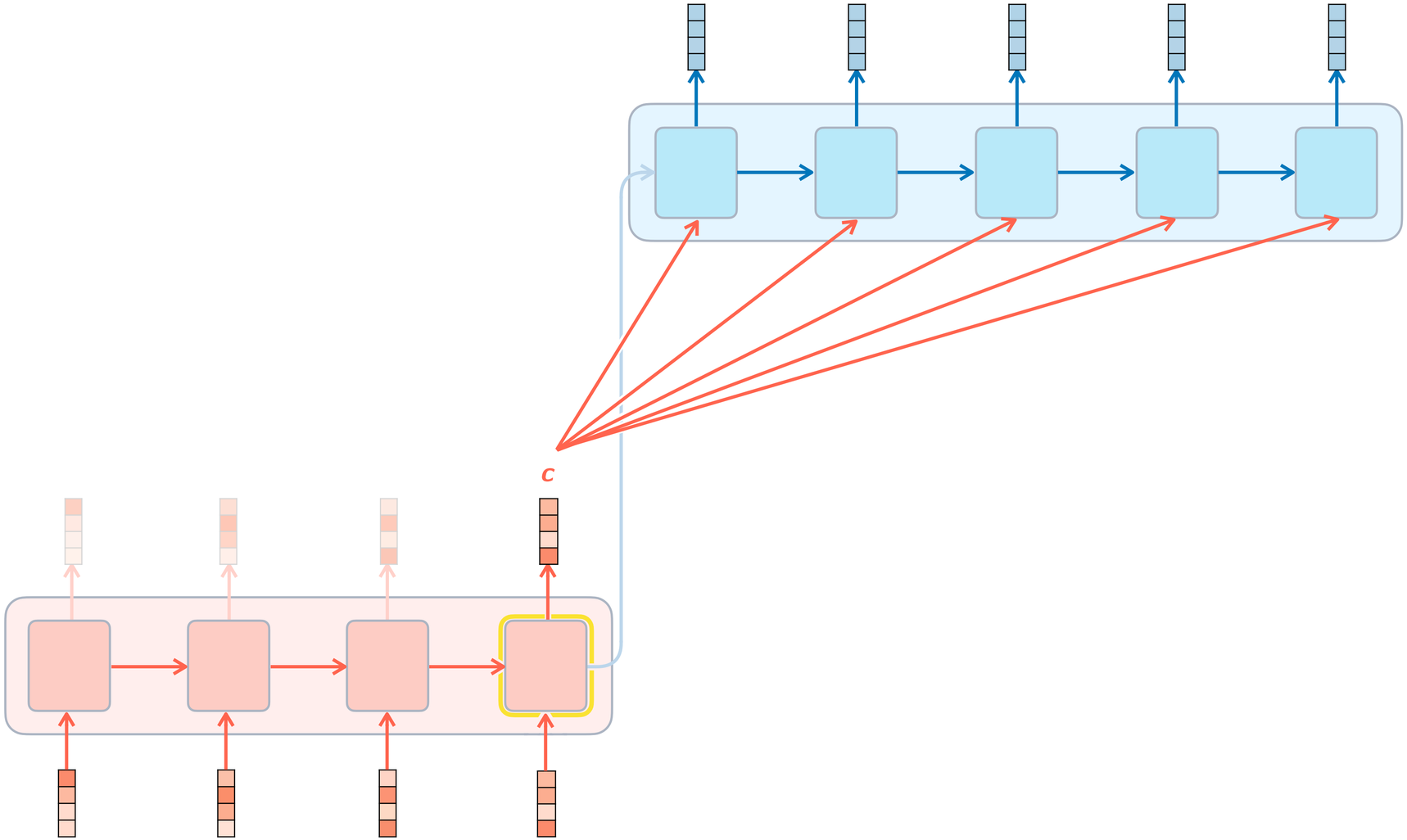
# Seq2Seq + Context Vector

*N*-to-*M*



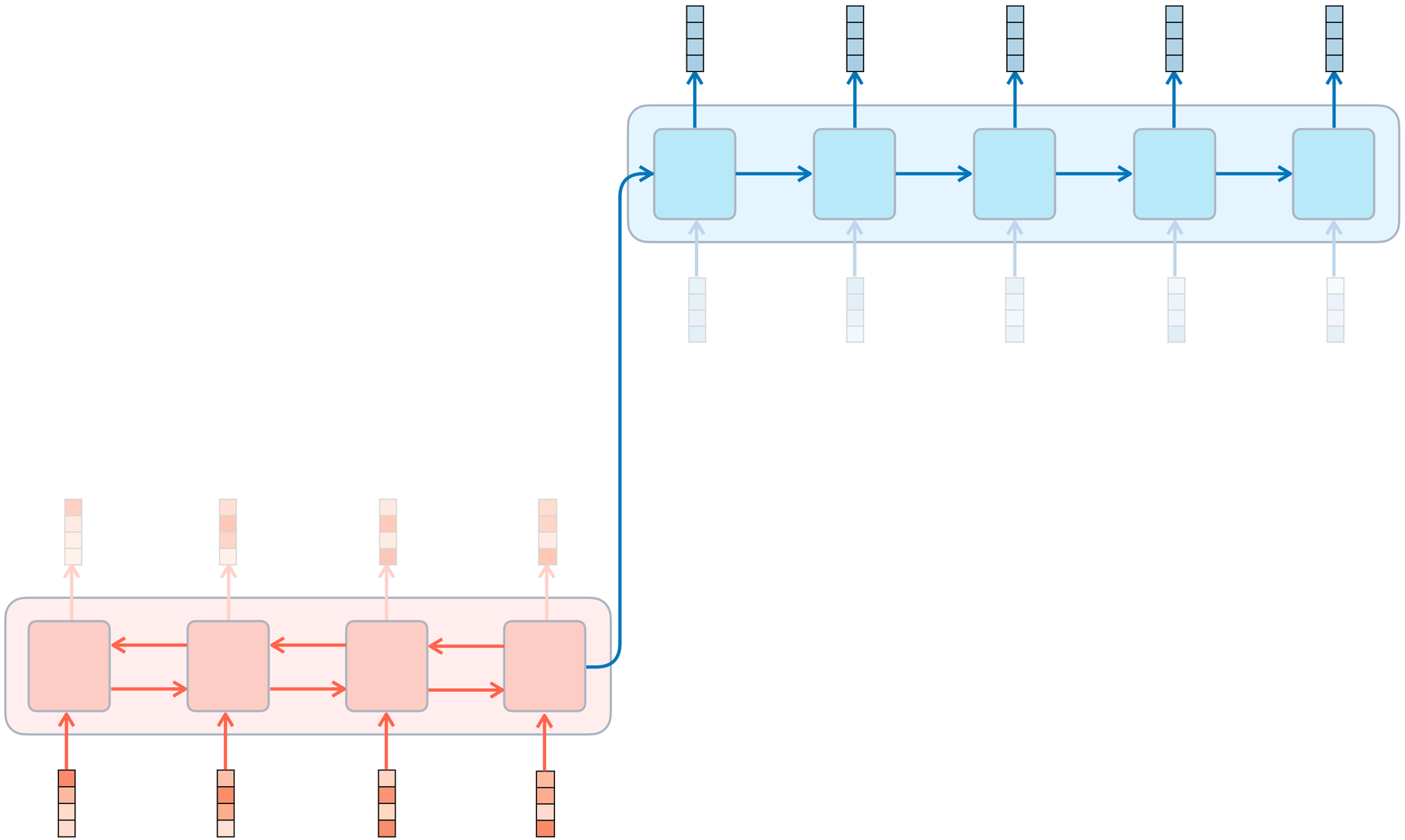
# Seq2Seq + Context Vector

*N*-to-*M*



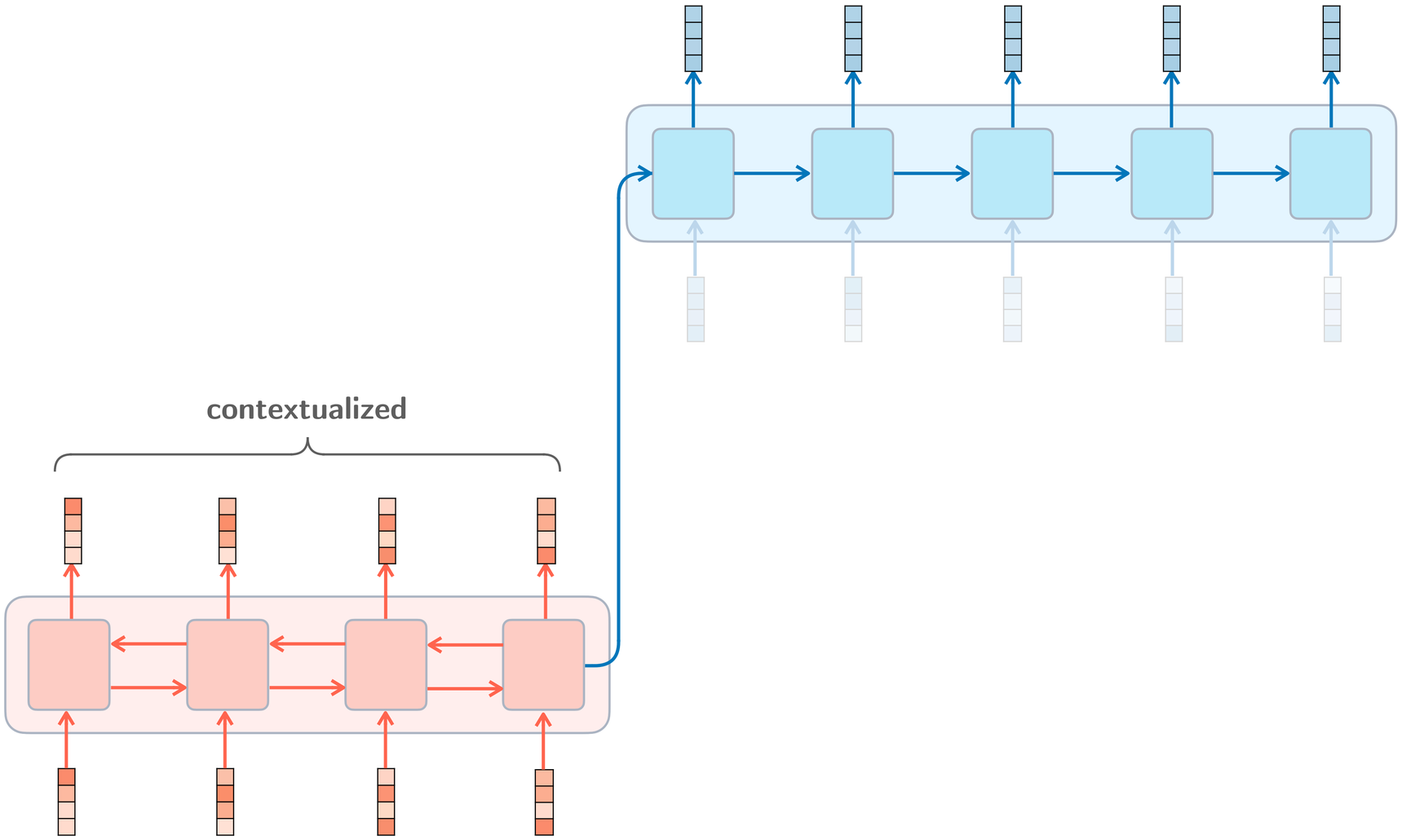
# Seq2Seq + Context Vector

*N*-to-*M*



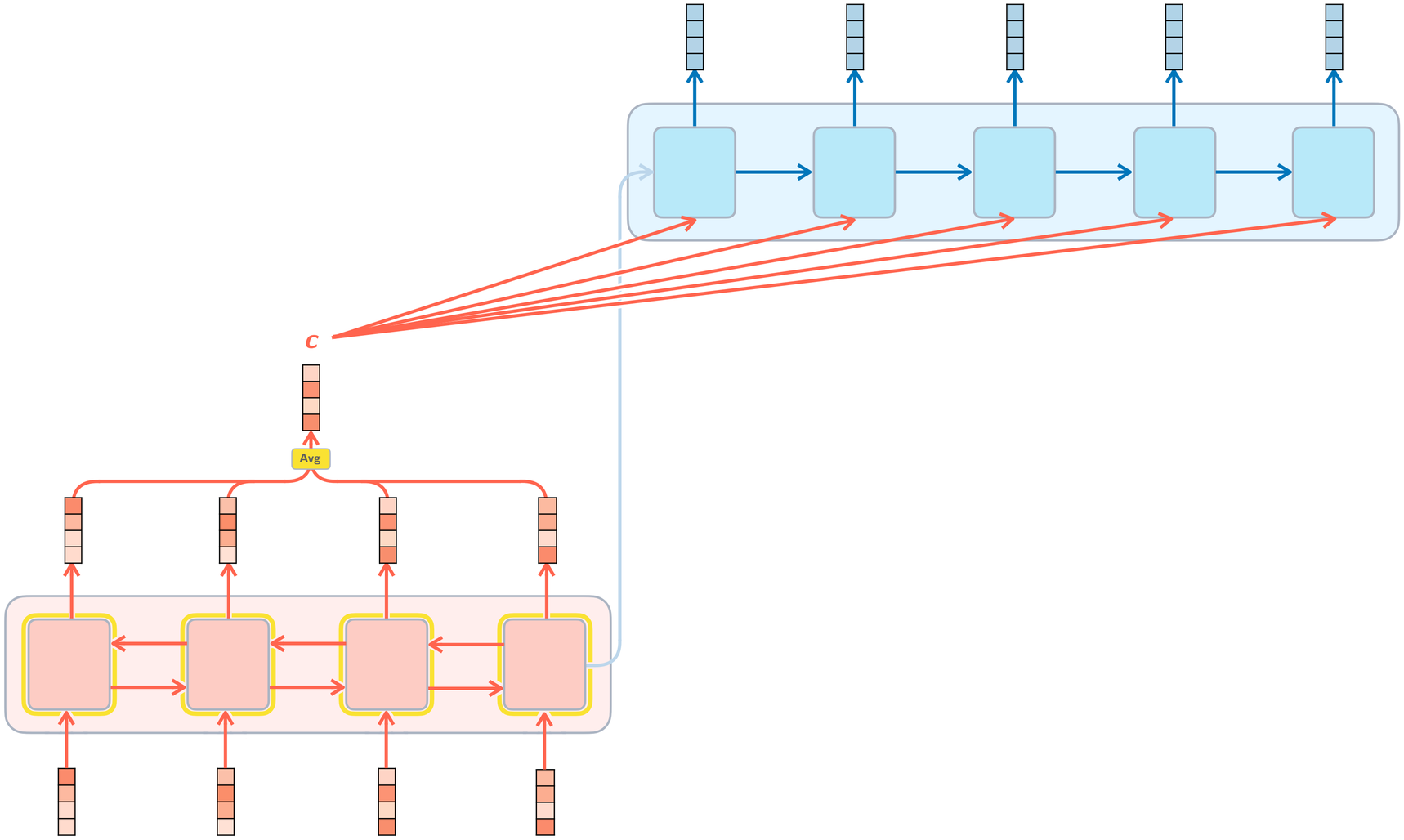
# Seq2Seq + Context Vector

*N*-to-*M*



# Seq2Seq + Context Vector

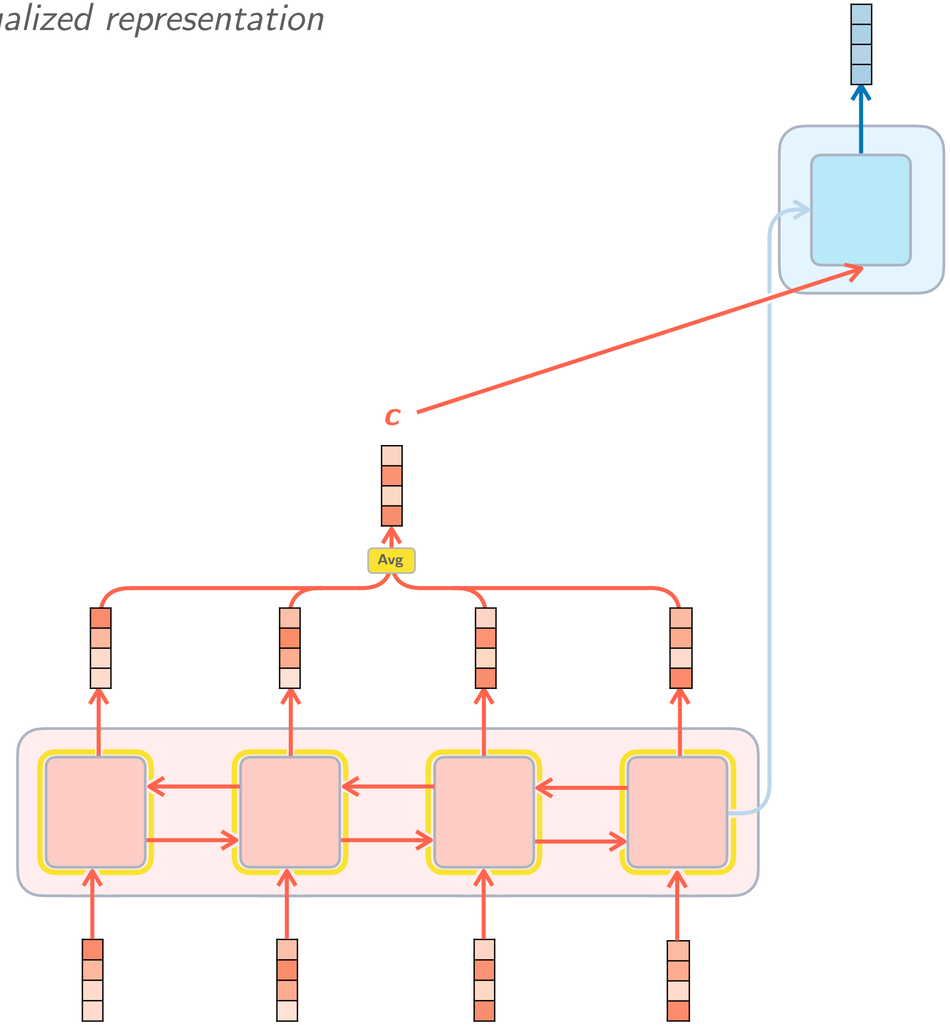
*N*-to-*M*



# Seq2Seq + Context Vector

## *N*-to-1

Making use of a *contextualized representation* and *context vector*.

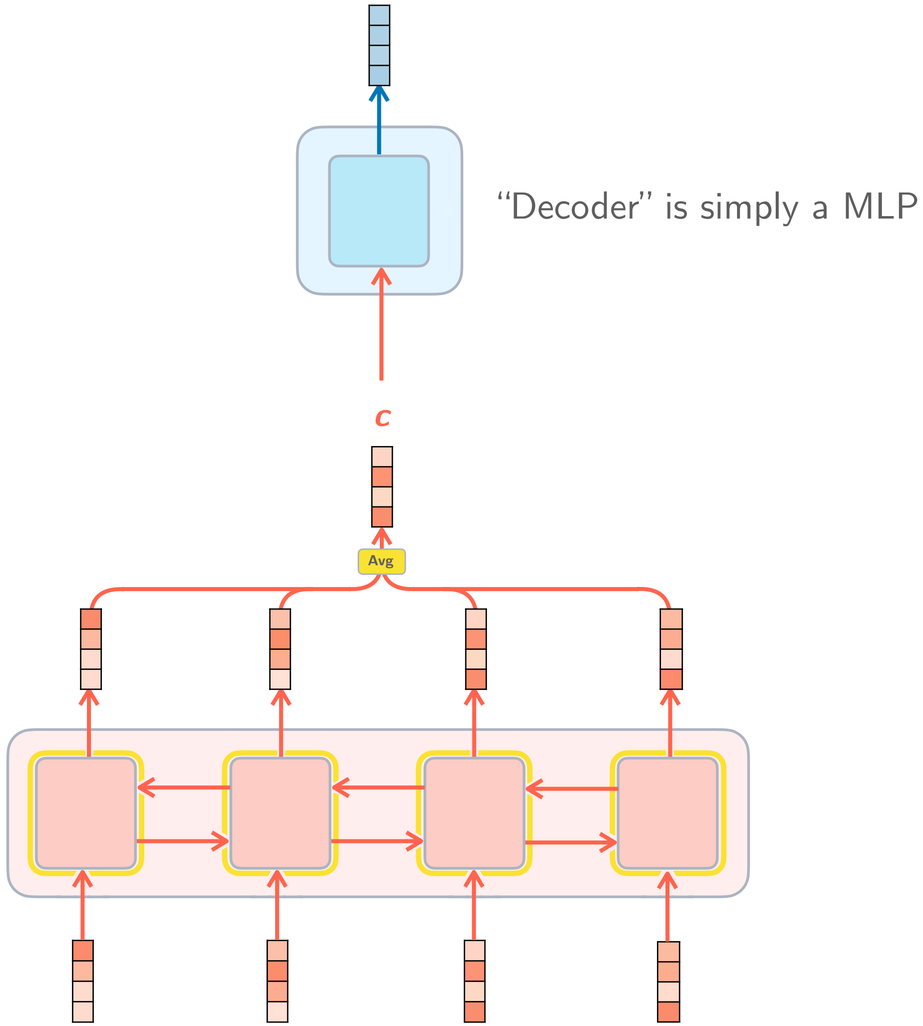



"Decoder" is simply a MLP

# Seq2Seq + Context Vector

## *N*-to-1

Making use of a *contextualized representation* and *context vector*.



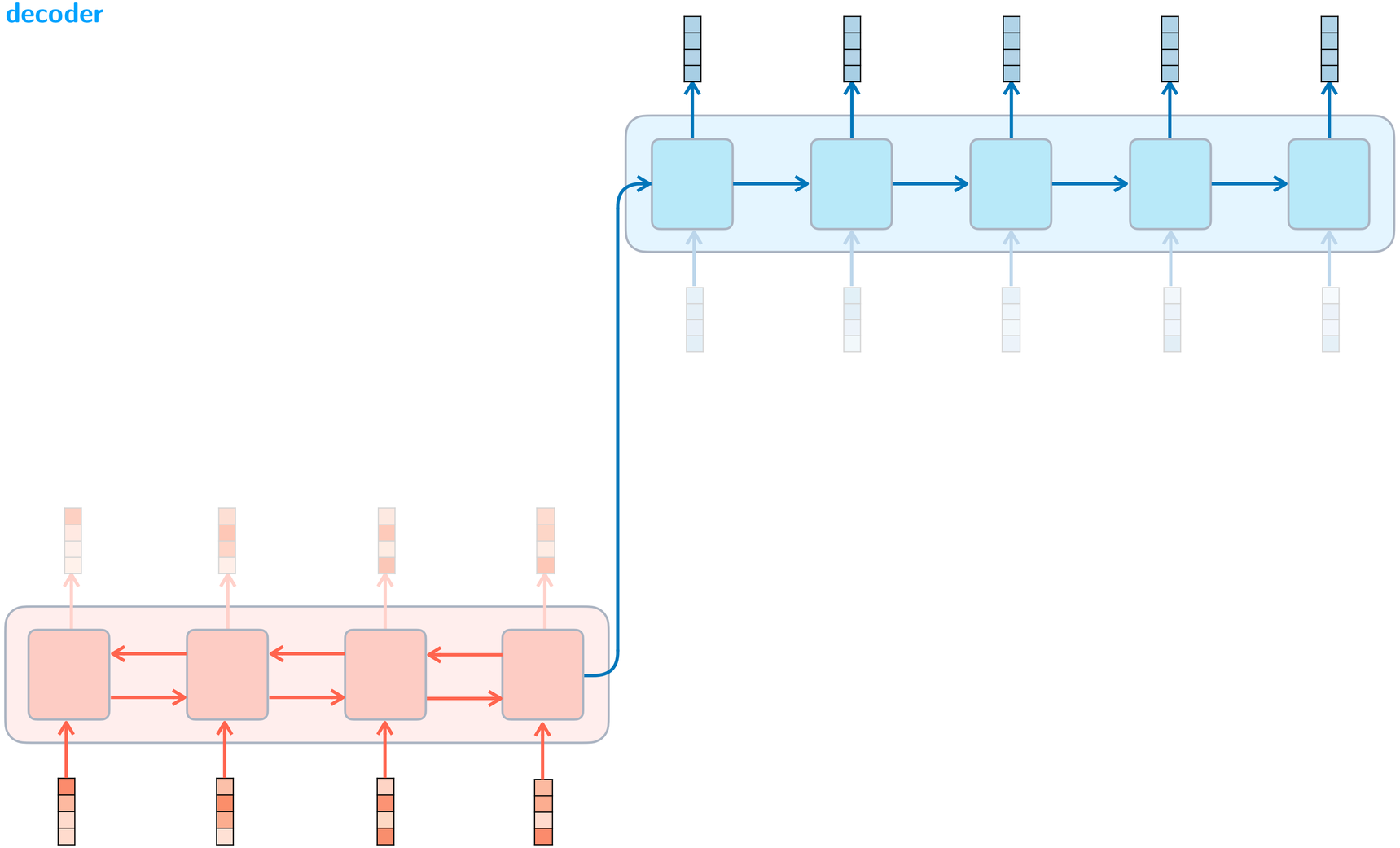
3

# Seq2Seq + Attention

# Seq2Seq + Attention

## *N*-to-*M*

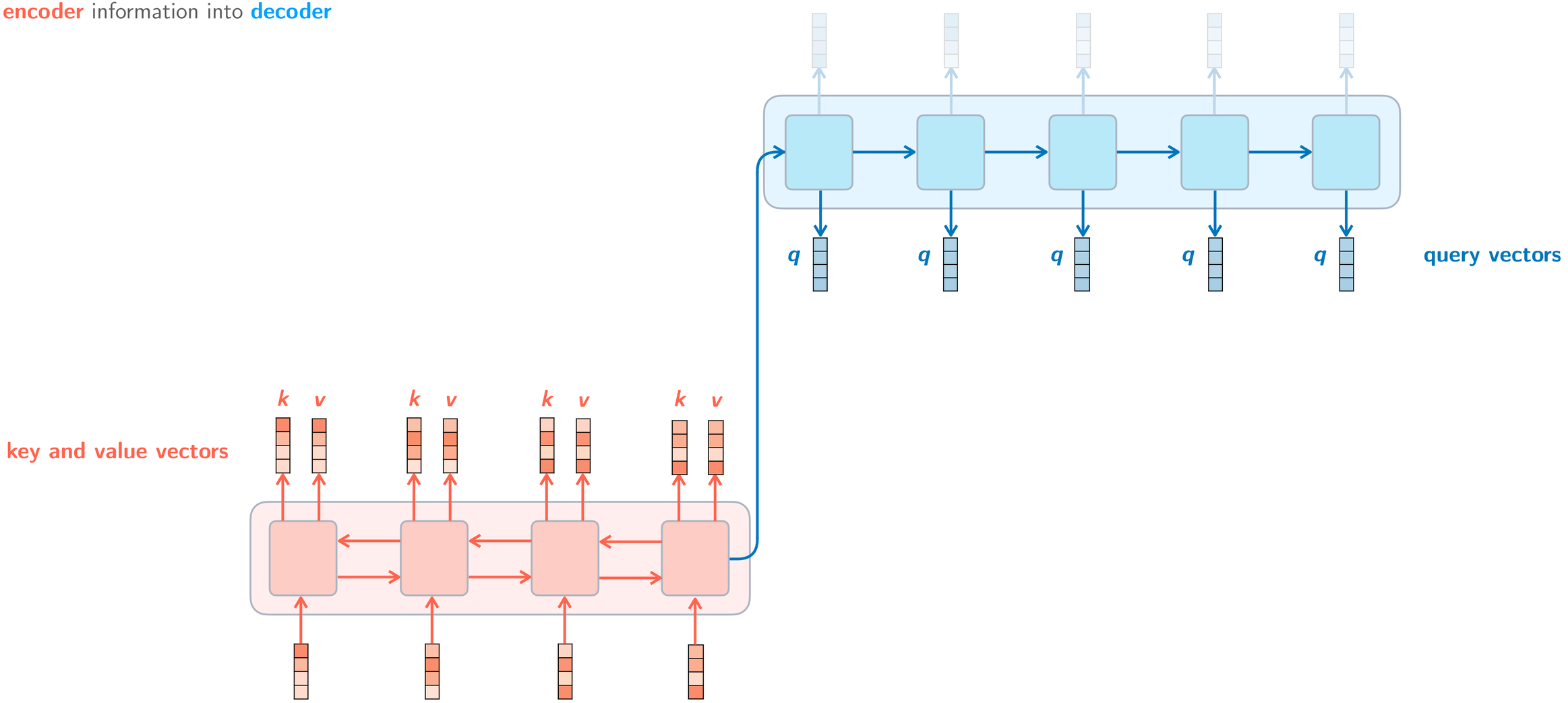
Feed **encoder** information into **decoder**



# Seq2Seq + Attention

## *N*-to-*M*

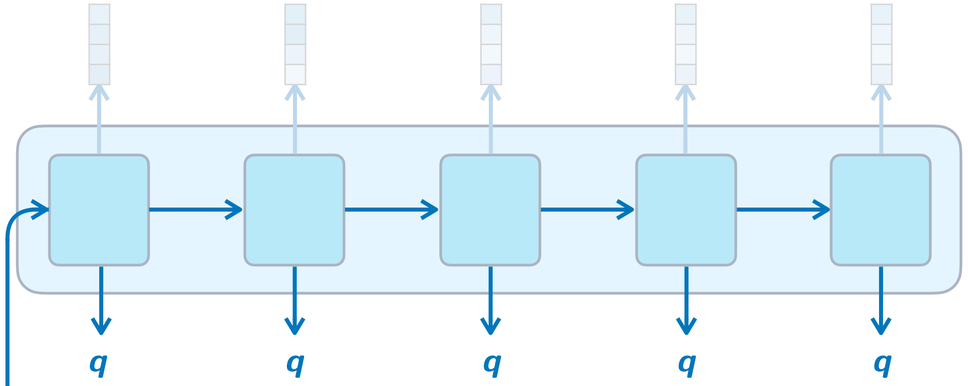
Feed **encoder** information into **decoder**



# Seq2Seq + Attention

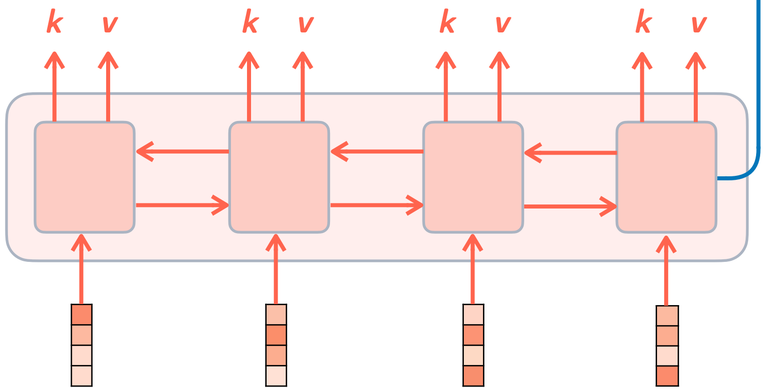
## *N*-to-*M*

come up with a **query**
for *this* **decoder** time step



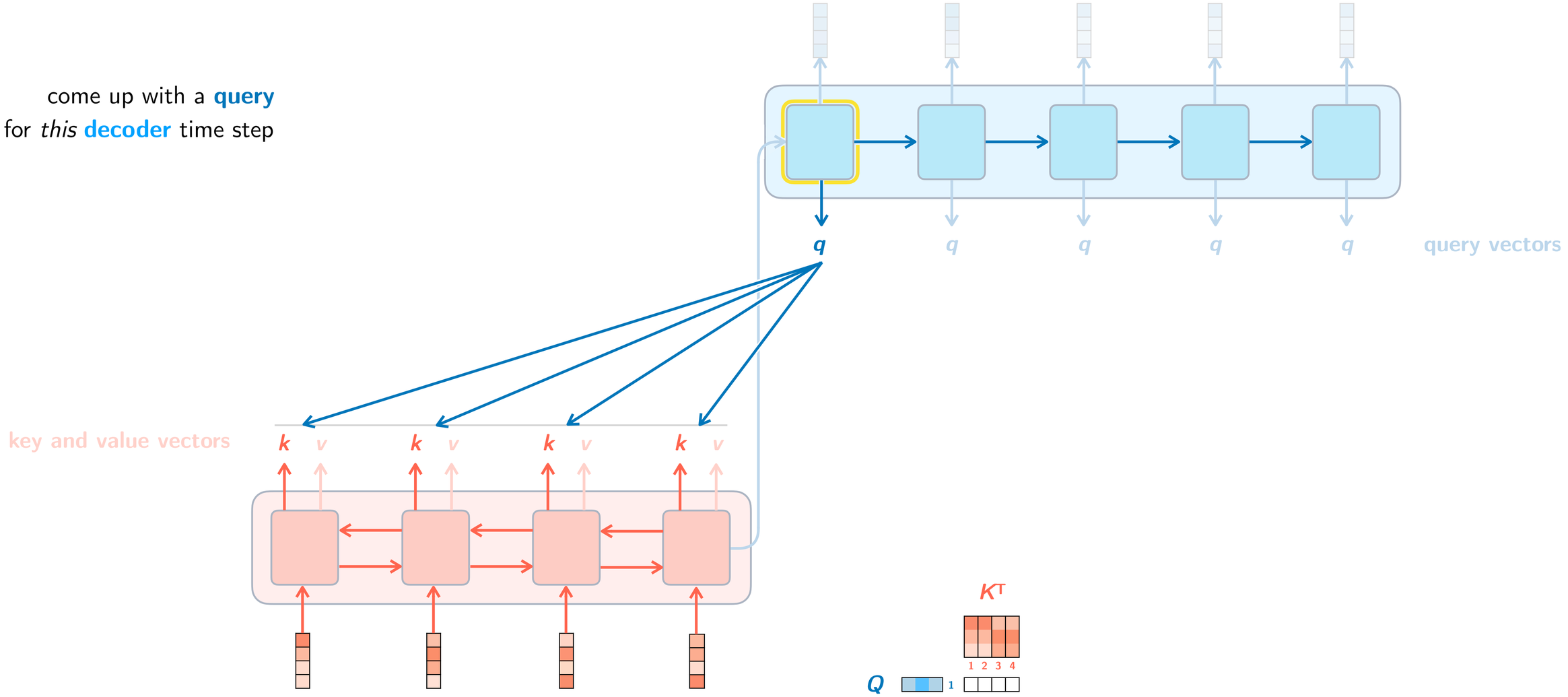

**query vectors**

**key and value vectors**



*N*-to-*M*

come up with a **query**
for *this* **decoder** time step



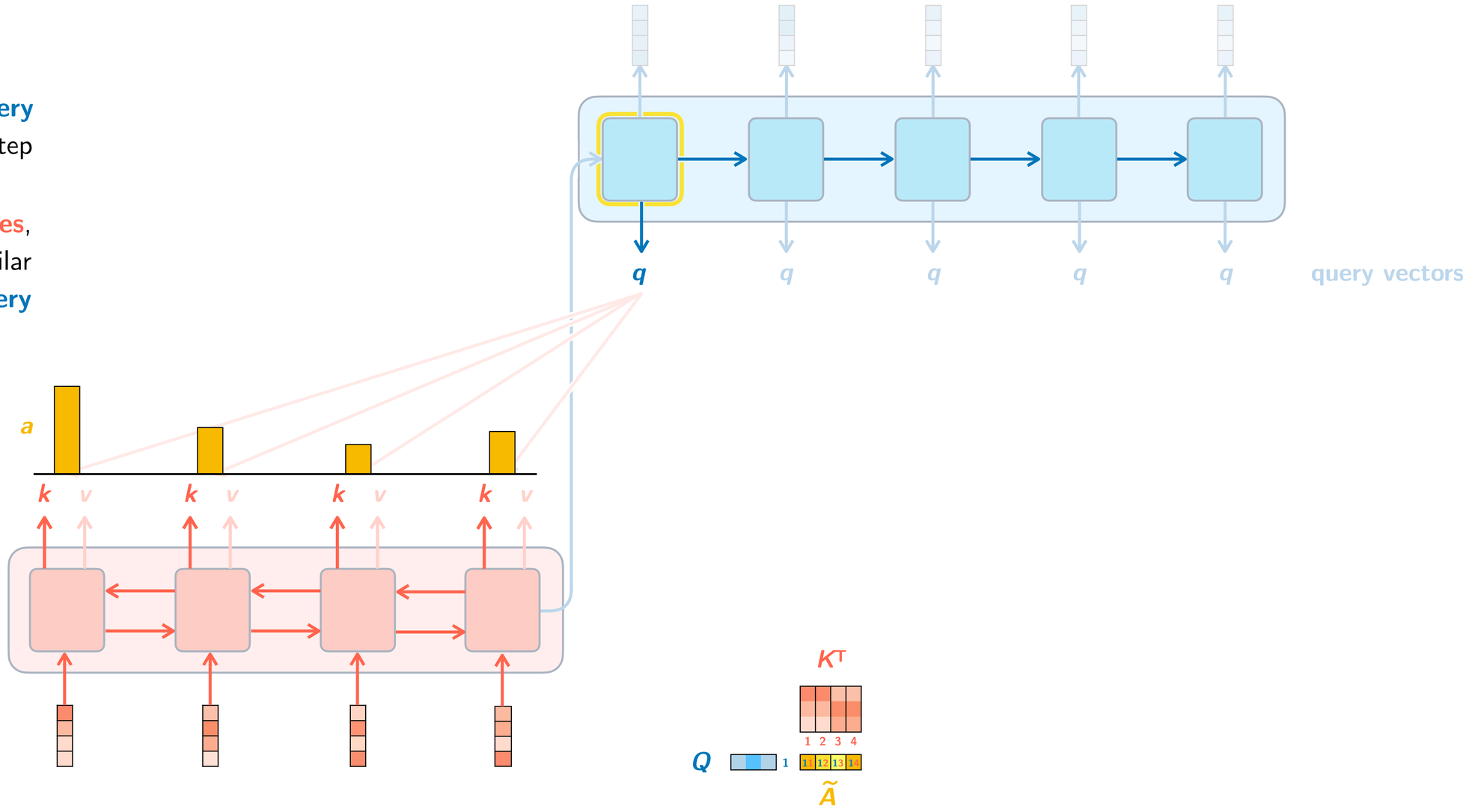
## *N*-to-*M*

come up with a **query** for *this* **decoder** time step

return the sum of **values**, **weighted** by how similar each **key** is to the **query**

$\tilde{a} = k^T q$

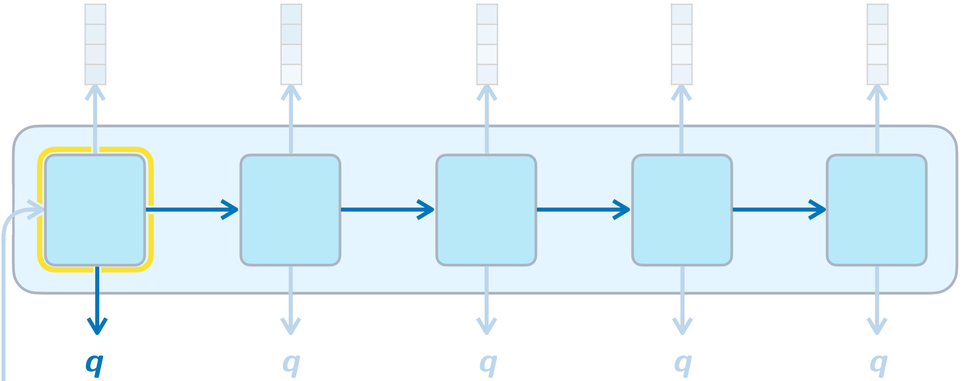
key and value vectors



*N*-to-*M*

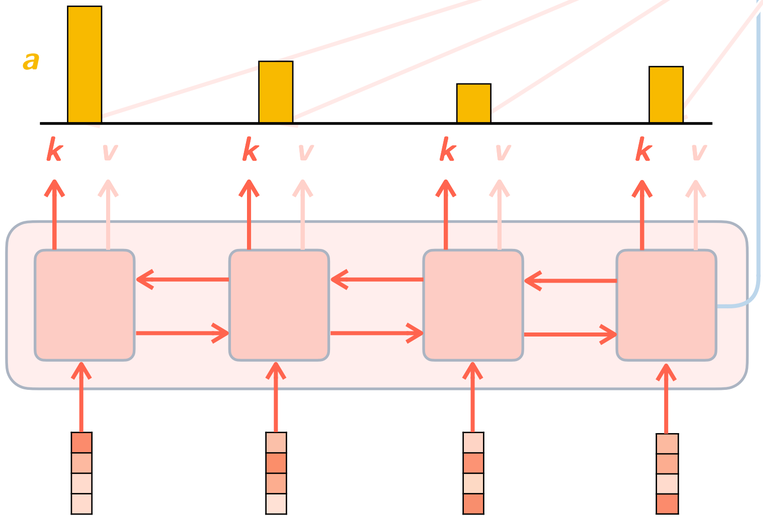
come up with a **query** for *this* **decoder** time step

return the sum of **values**, **weighted** by how similar each **key** is to the **query**



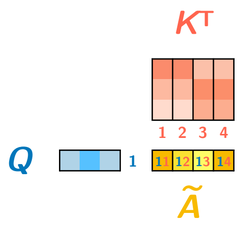

$\tilde{a} = k^T q$

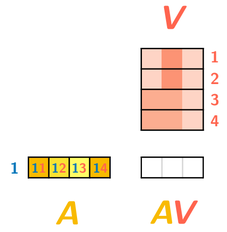
key and value vectors




return the sum of **values**, **weighted** by how similar a **value**'s **key** is to the **query**

$a = \text{softmax}([\tilde{a}_1, \ldots, \tilde{a}_N])$



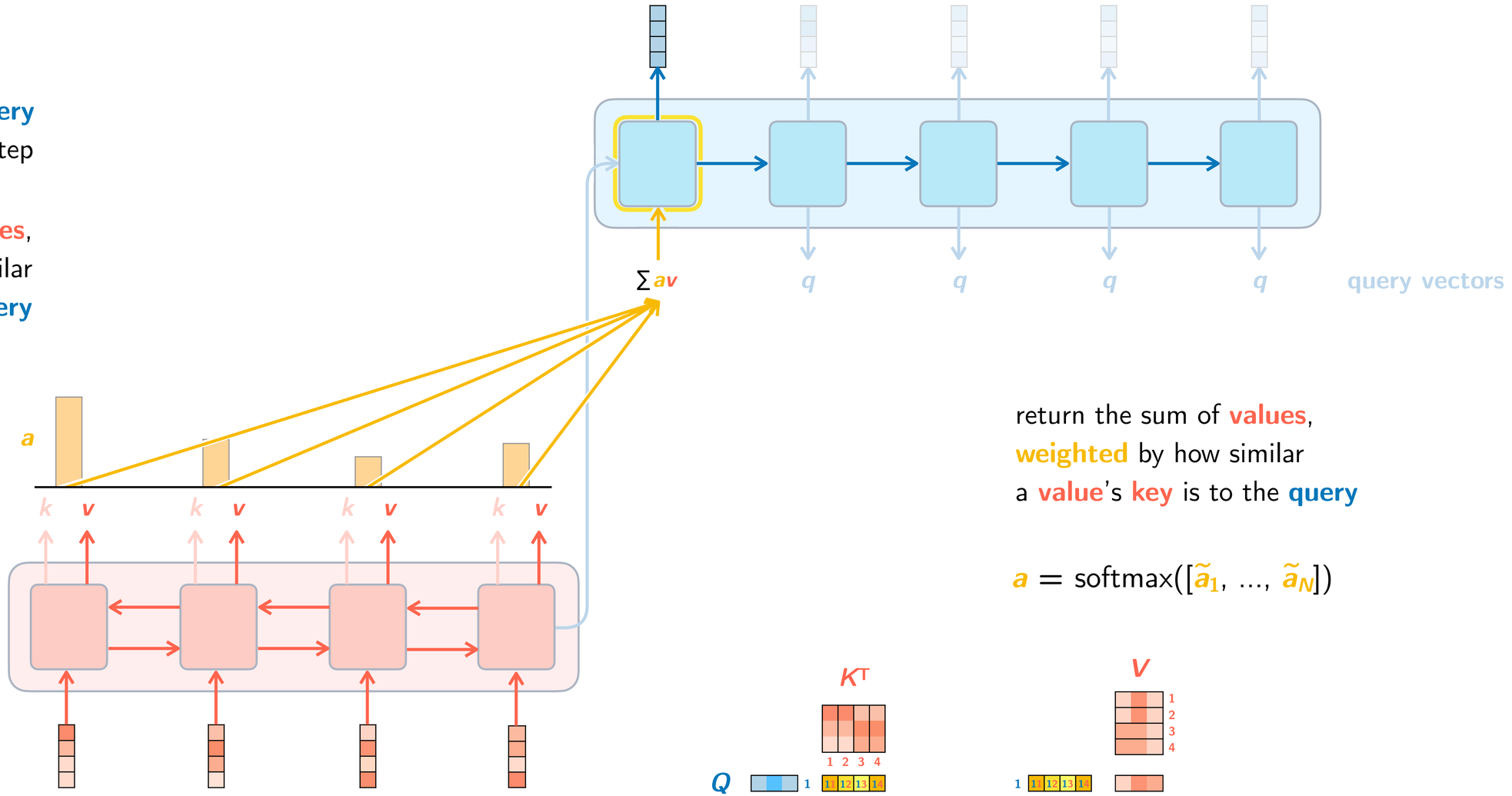
*N*-to-*M*

come up with a **query** for *this* **decoder** time step

return the sum of **values**, **weighted** by how similar each **key** is to the **query**

$\tilde{a} = k^T q$

key and value vectors



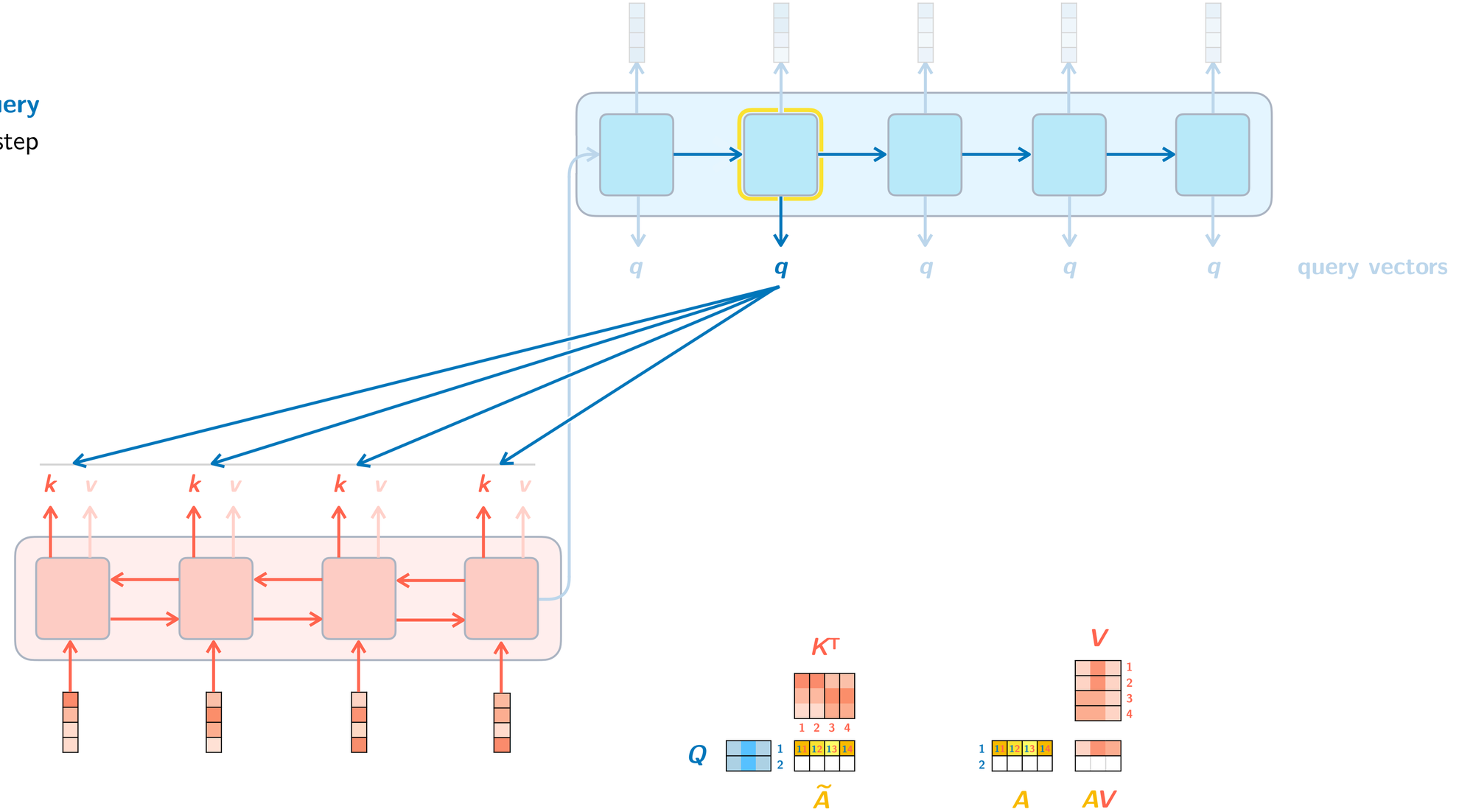

return the sum of **values**, **weighted** by how similar a **value**'s **key** is to the **query**

$a = \text{softmax}([\tilde{a}_1, \ldots, \tilde{a}_N])$

# Seq2Seq + Attention

## *N*-to-*M*

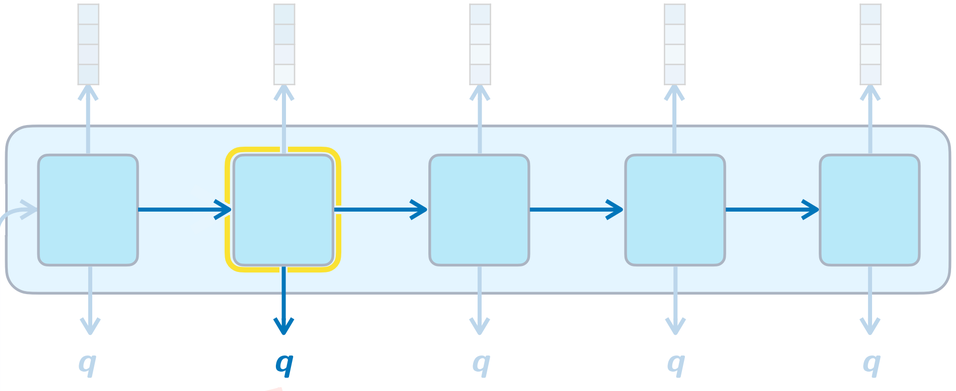
come up with a **query**
for *this* **decoder** time step



# Seq2Seq + Attention

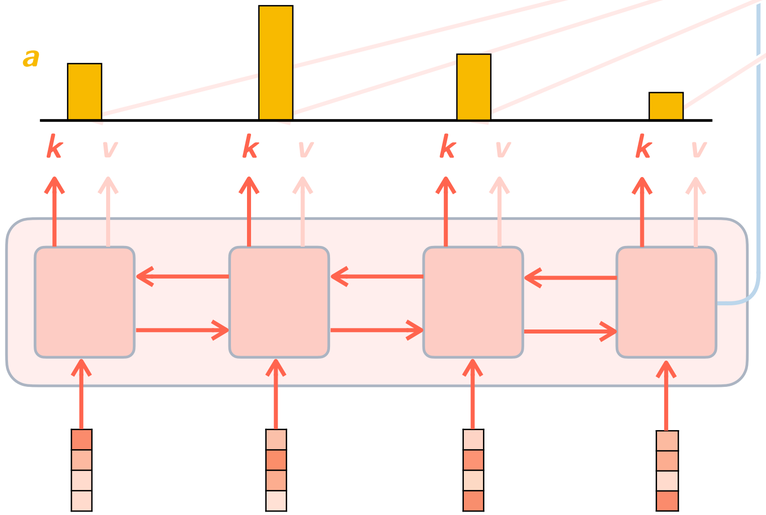
## *N*-to-*M*

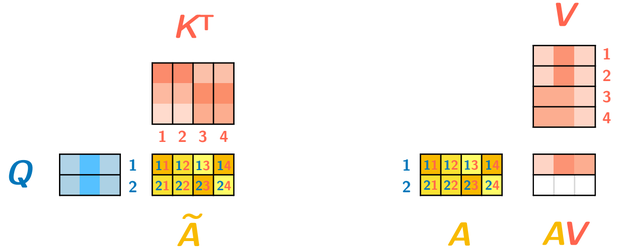
come up with a **query**
for *this* **decoder** time step




$\tilde{a} = k^\intercal q$

key and value vectors

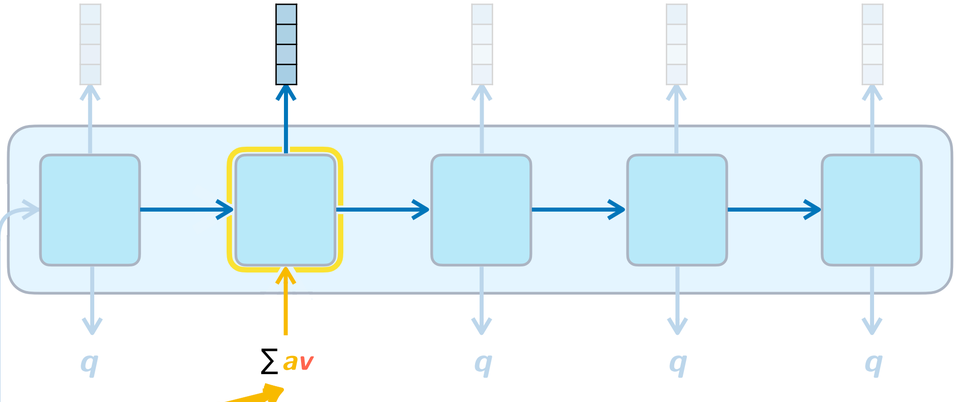


# Seq2Seq + Attention

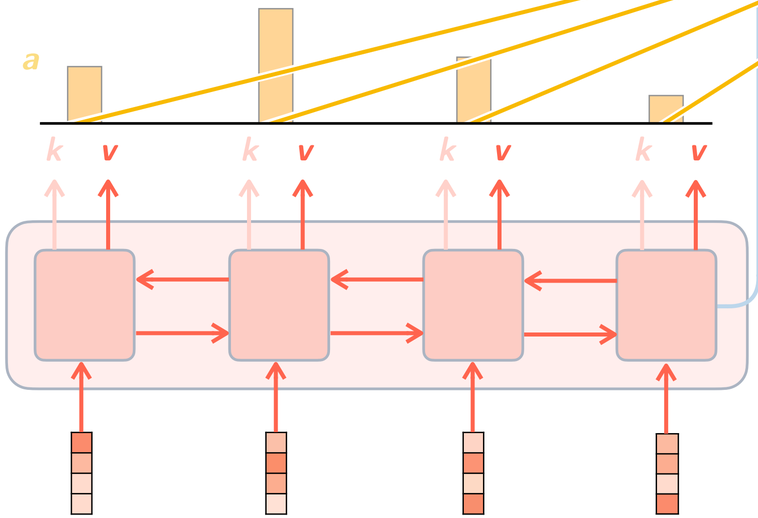
## *N*-to-*M*

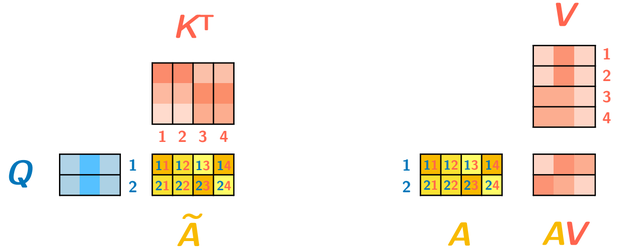
come up with a **query**
for *this* **decoder** time step




$\tilde{a} = k^\intercal q$

key and value vectors

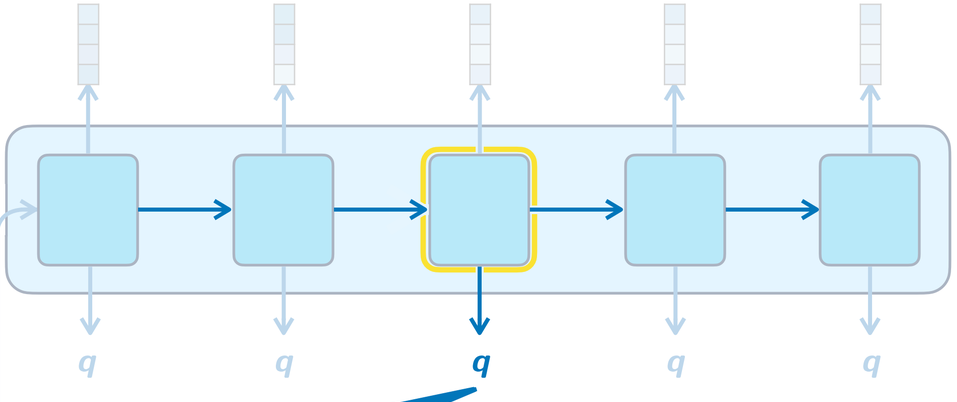


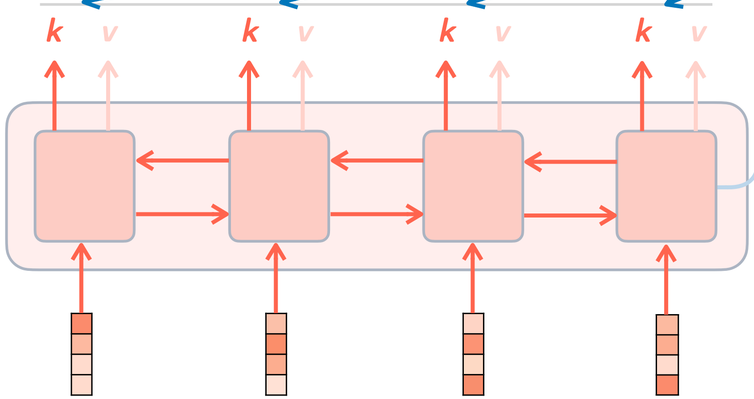
# Seq2Seq + Attention

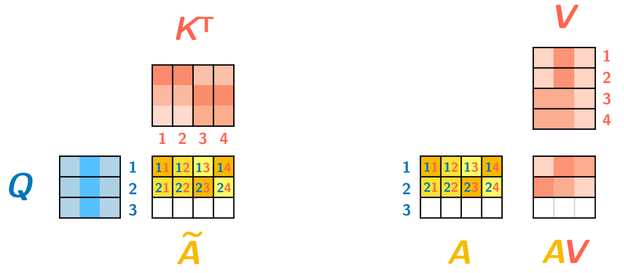
## *N*-to-*M*

come up with a **query**
for *this* **decoder** time step



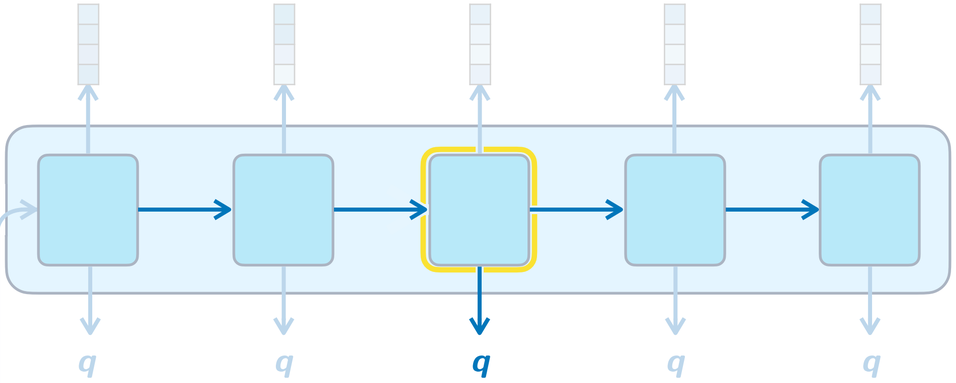

key and value vectors



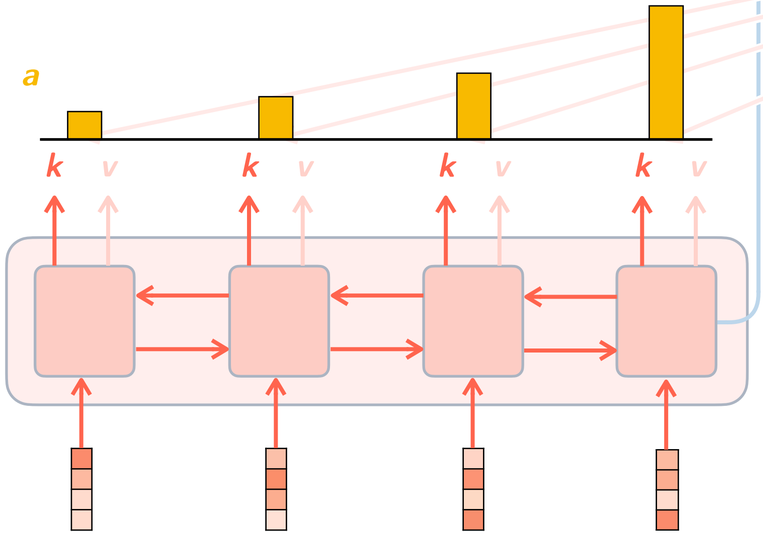
## *N*-to-*M*

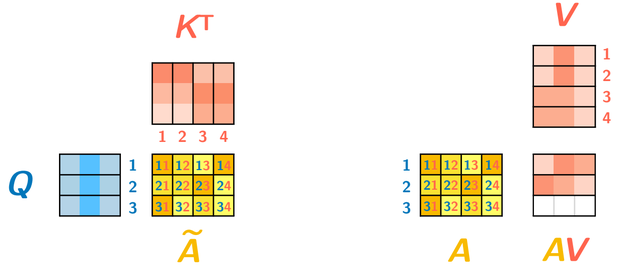
come up with a **query**
for *this* **decoder** time step




$\tilde{a} = k^\intercal q$

key and value vectors

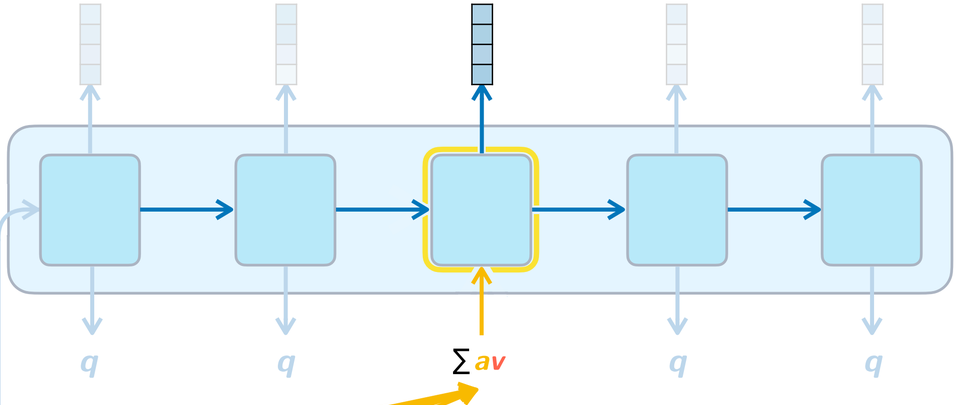


# Seq2Seq + Attention

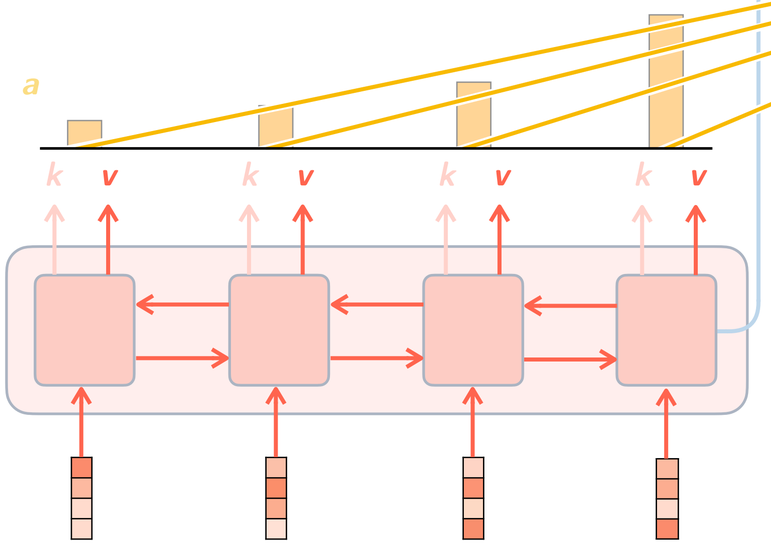
## *N*-to-*M*

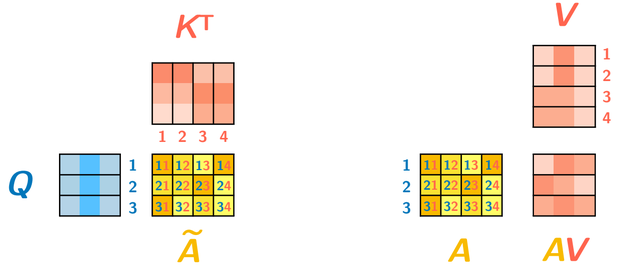
come up with a **query**
for *this* **decoder** time step




$\tilde{a} = k^T q$

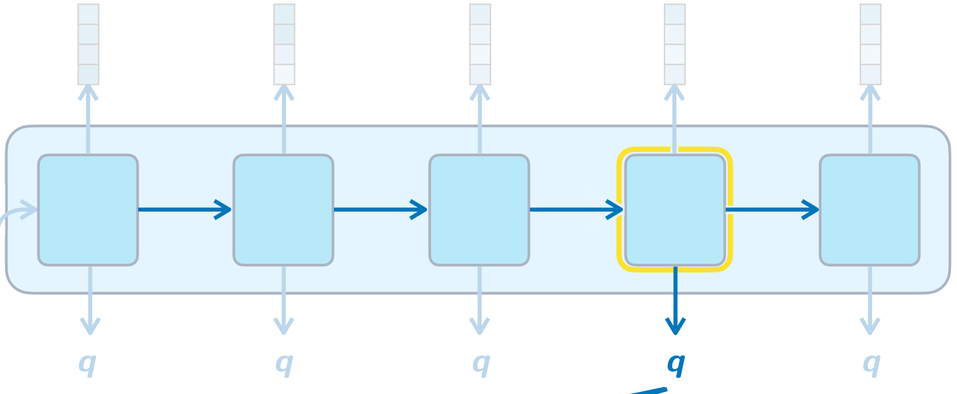
key and value vectors



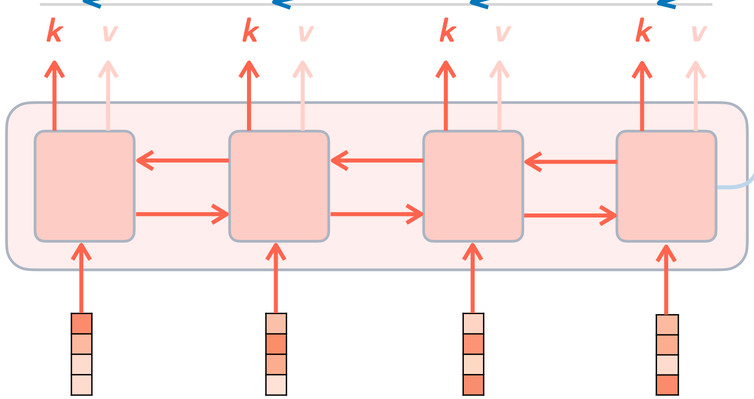



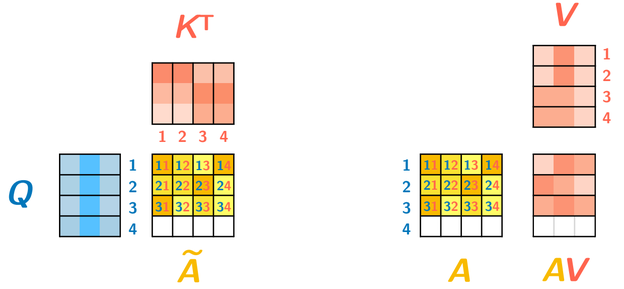
## *N*-to-*M*

come up with a **query**
for *this* **decoder** time step



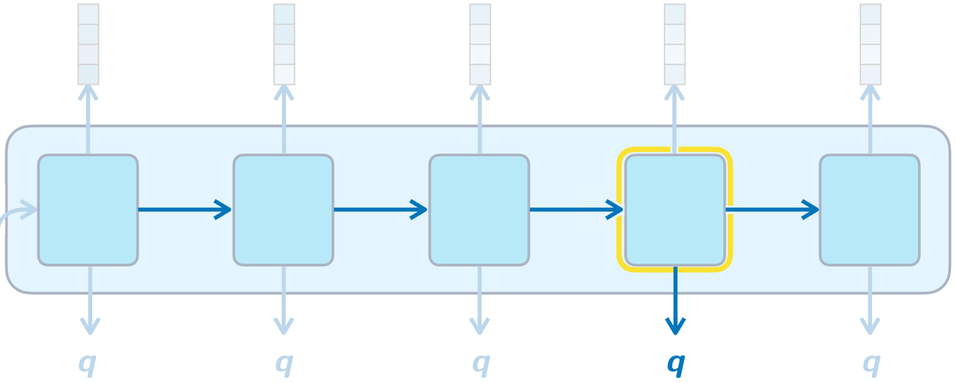

key and value vectors



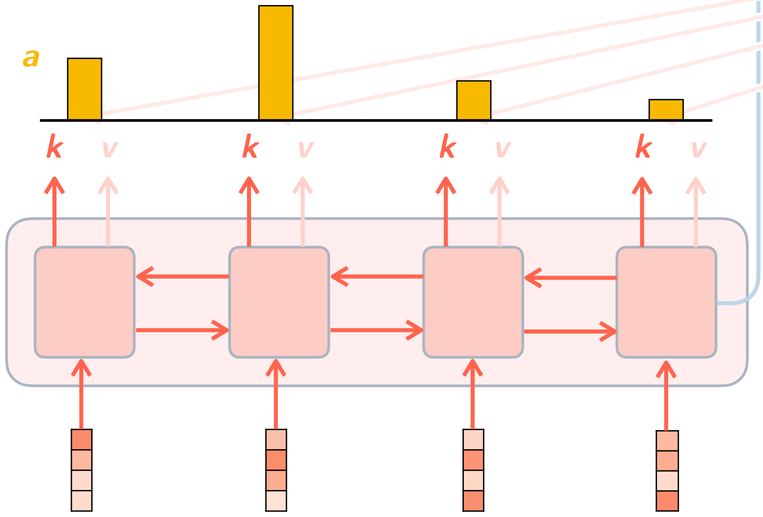



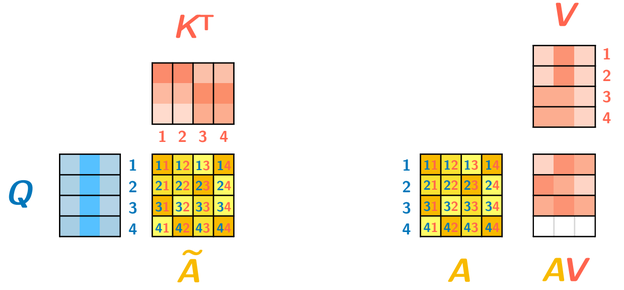
## *N*-to-*M*

come up with a **query**
for *this* **decoder** time step




key and value vectors

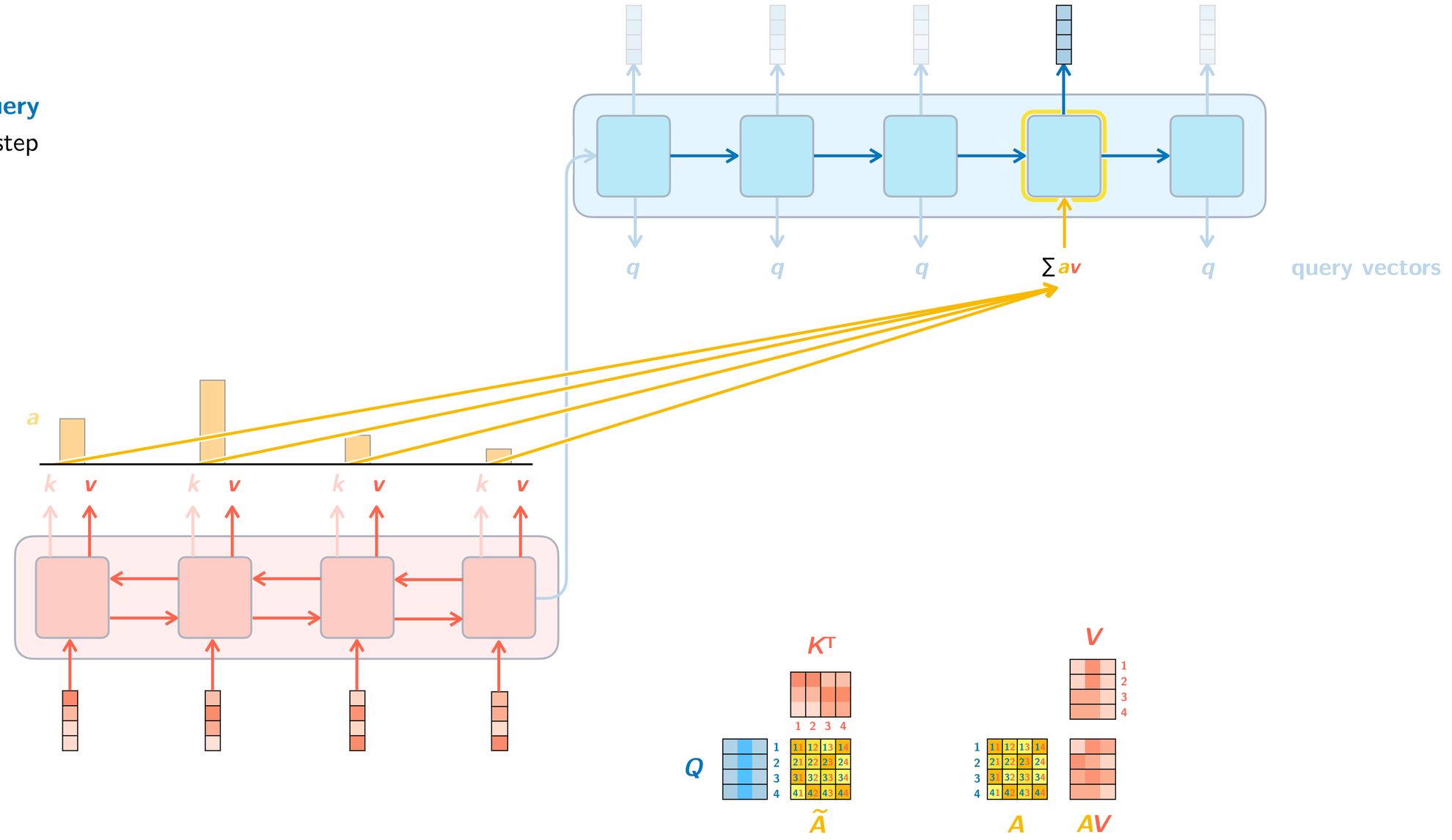


# Seq2Seq + Attention

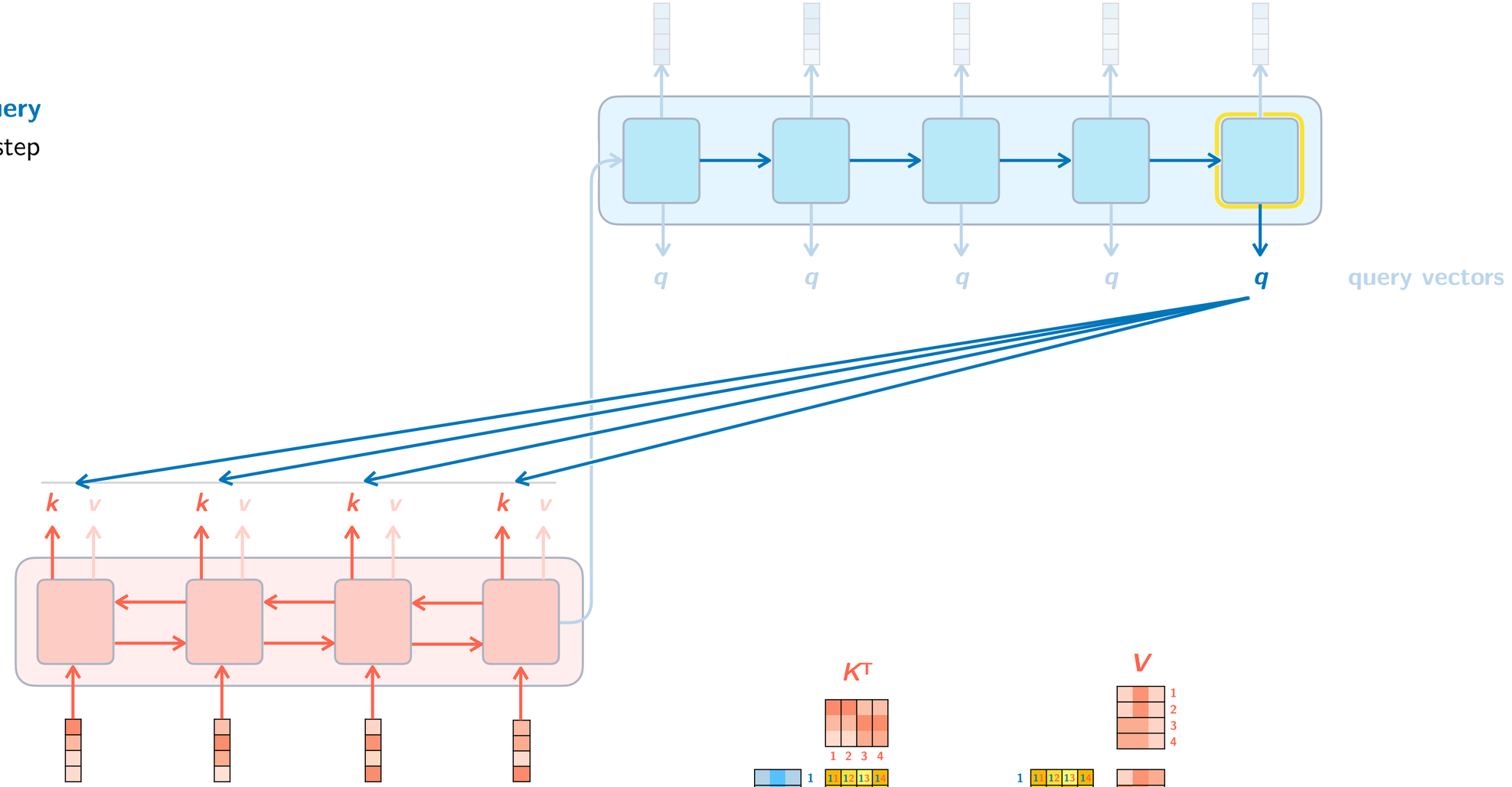
## *N*-to-*M*

come up with a **query**
for *this* **decoder** time step



## *N*-to-*M*

come up with a **query**
for *this* **decoder** time step

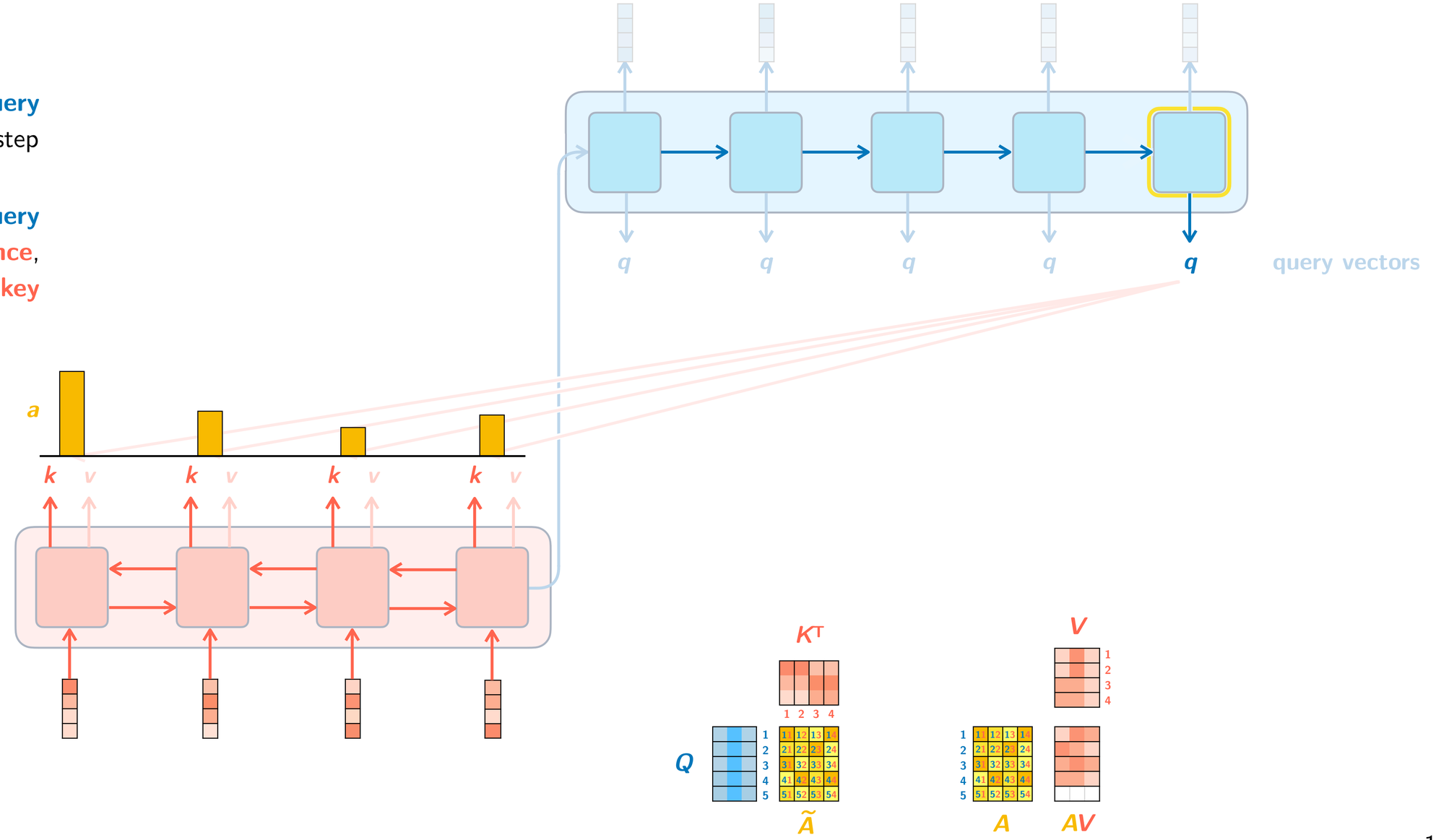


# Seq2Seq + Attention

## *N*-to-*M*

come up with a **query**
for *this* **decoder** time step

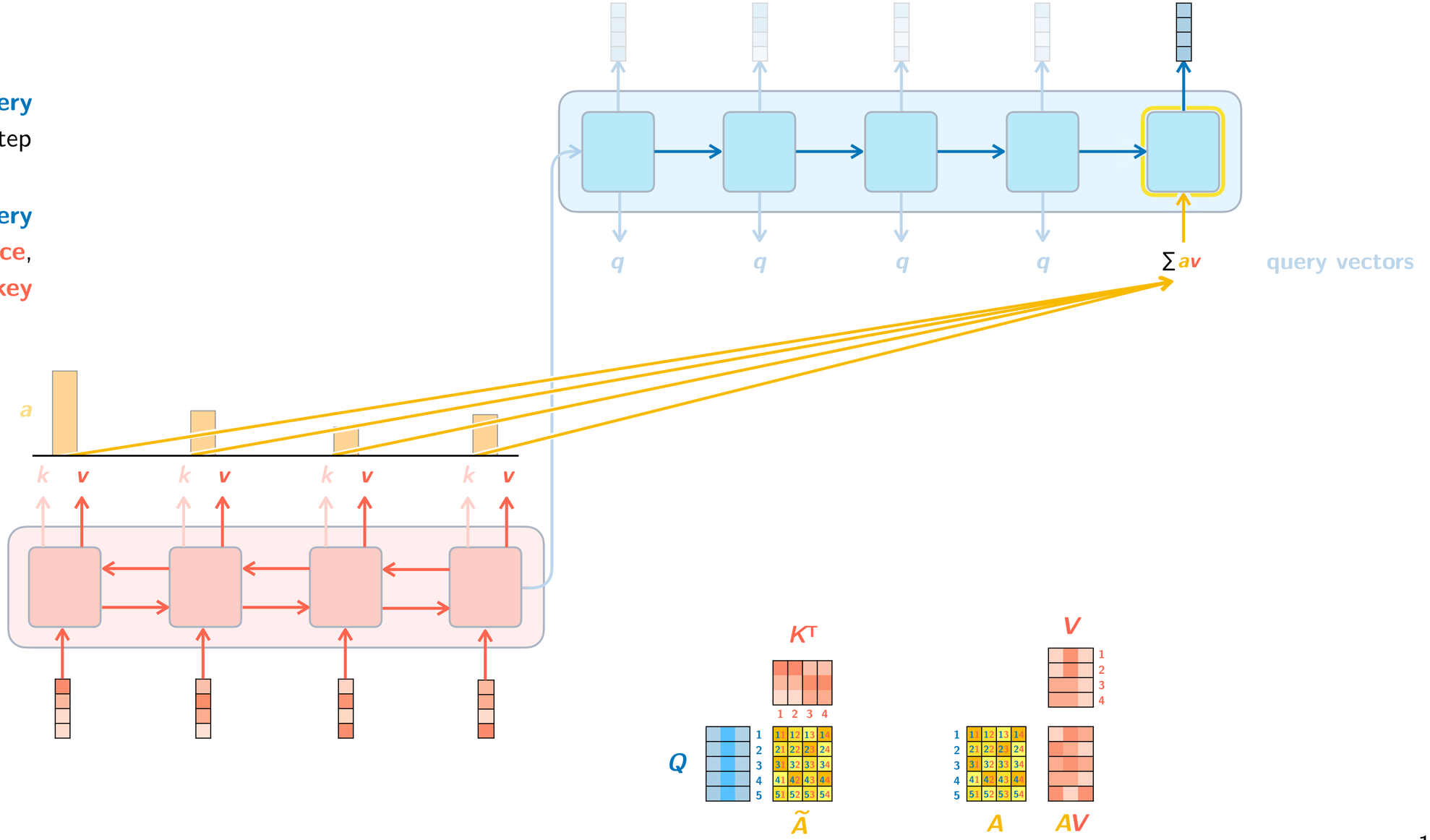
search for this **query**
in the **encoder sequence**,
by comparing it to each **key**



## *N*-to-*M*

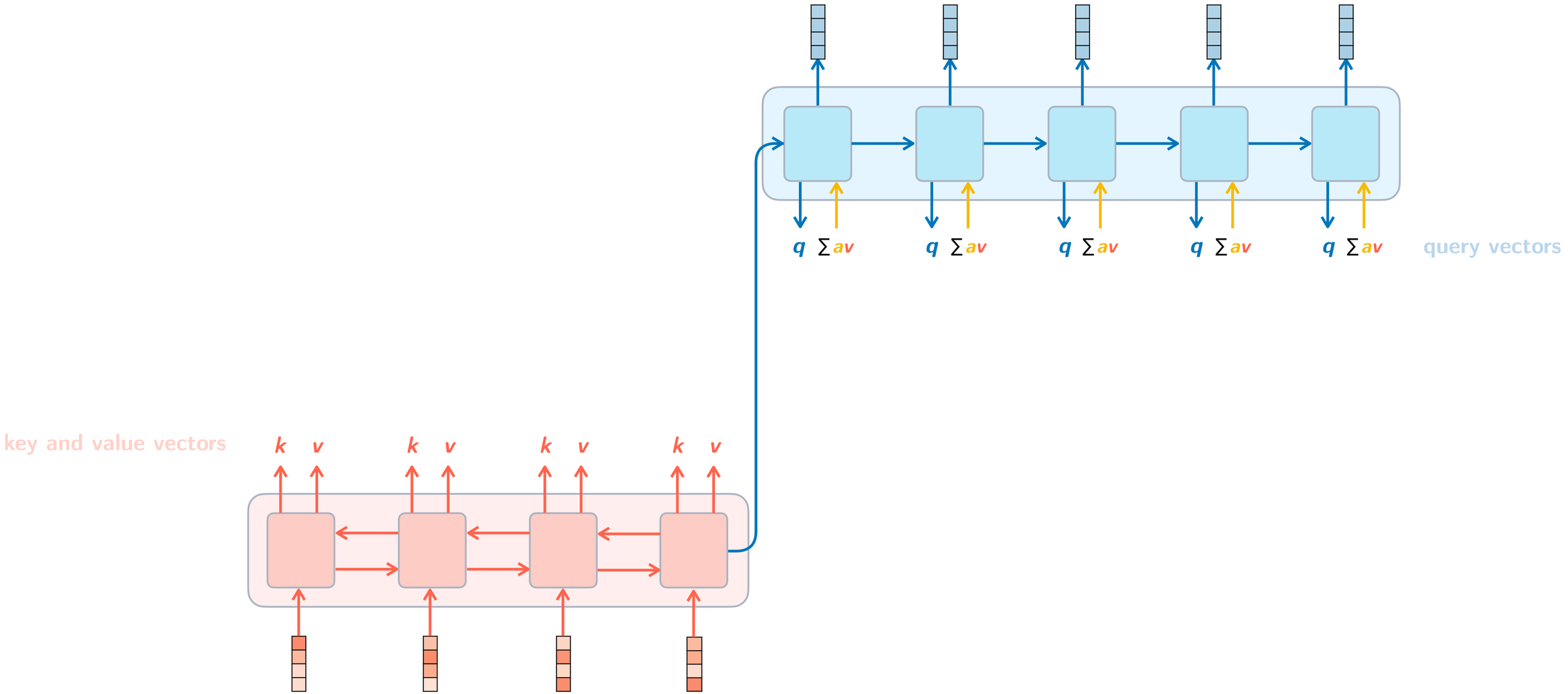
come up with a **query**
for *this* **decoder** time step

search for this **query**
in the **encoder sequence**,
by comparing it to each **key**



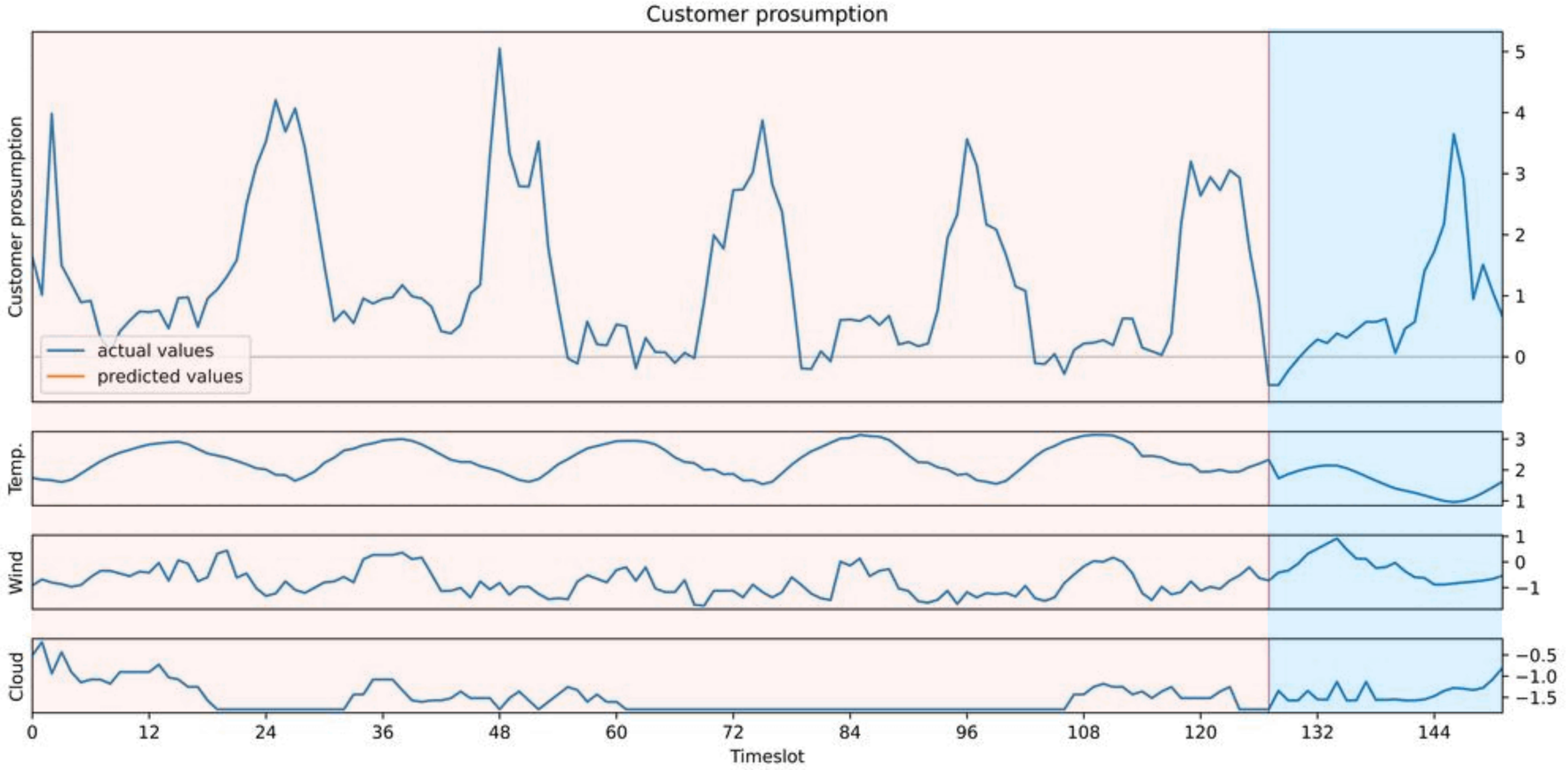
# Seq2Seq + Attention

*N*-to-*M*

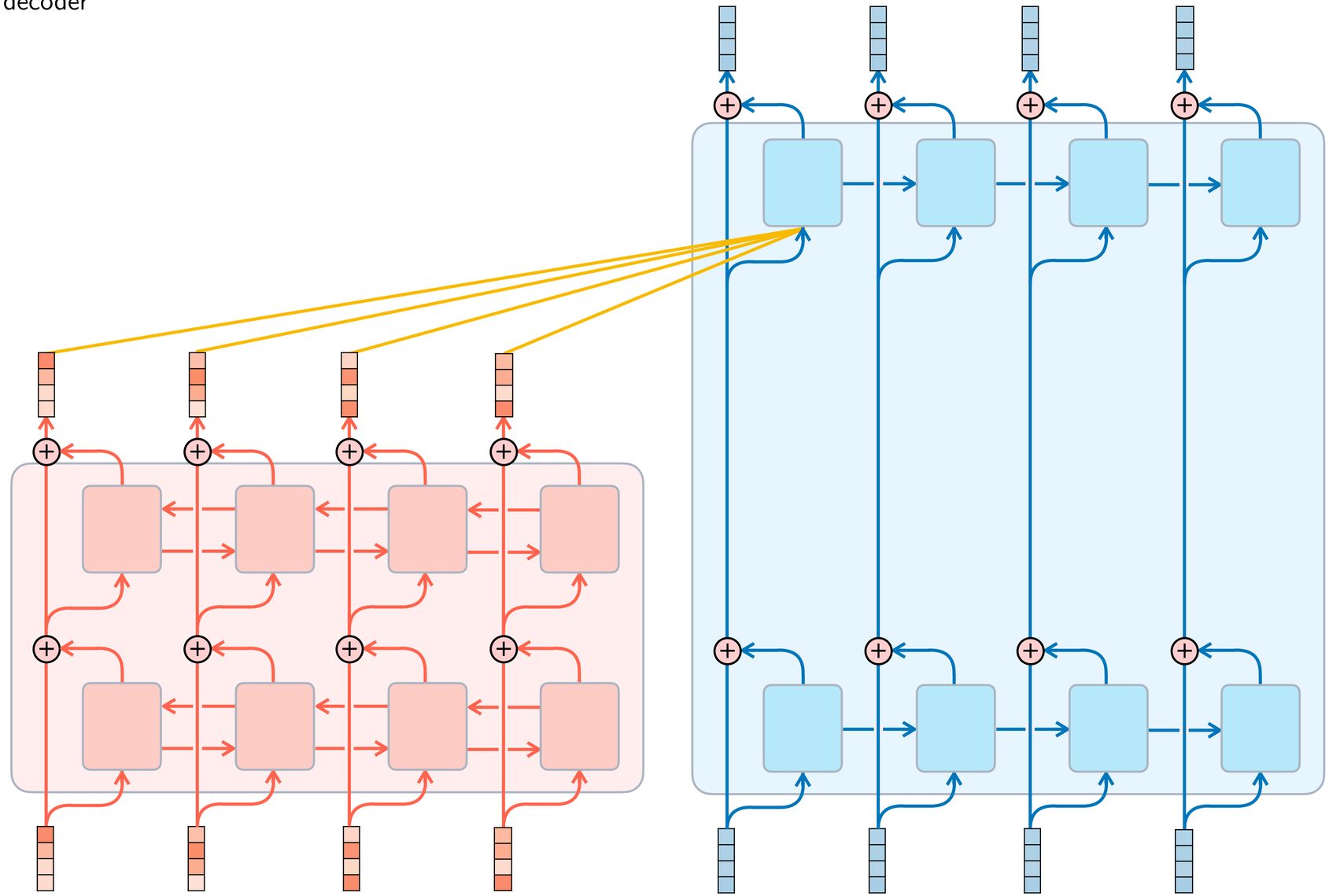


# Example: Electricity Market

(one particular attention head)



GNMT (2016), encoder and decoder



**Thanks for your *attention*.**

**Blog posts:**

- https://colah.github.io/posts/2015-08-Understanding-LSTMs
- https://jalammar.github.io/illustrated-transformer
- https://jalammar.github.io/illustrated-bert
- https://jalammar.github.io/illustrated-gpt2
- https://dugas.ch/artificial_curiosity/GPT_architecture.html

**Papers:**

- *Sequence-to-Sequence Learning with Neural Networks*
- *Neural Machine Translation by Jointly Learning to Align and Translate* (2016, Bahdanau, Cho & Bengio)
- *Attention Is All You Need* (2017, Vaswani et al.)
- *A Primer in BERTology* [...] (2020, Rogers et al.)
- *Google's Neural Machine Translation System* [...] (2016, Wu et al.)
- *Improving Language Understanding by Generative Pre-Training* (2017, Radford et al.)
- *Language Models are Unsupervised Multitask Learners* (2018, Radford et al.)
- *Language Models are Few-Shot Learners* (2020, Brown et al.)

**Videos:**

- *Illustrated Guide to Transformers Neural Network: A step by step explanation* (by Michael Phi), https://www.youtube.com/watch?v=4Bdc55j80l8
- *What are Transformer Neural Networks?* (by Ari Seff), https://www.youtube.com/watch?v=XSSTuhyAmnI
- *Attention Is All You Need* (by Yannic Kilcher), https://www.youtube.com/watch?v=iDulhoQ2pro
- *The Transformer neural network architecture EXPLAINED. "Attention is all you need" (NLP)* (by AI Coffee Break with Letitia) https://www.youtube.com/watch?v=FWFA4DGuzSc