

The background of the slide is a reproduction of Salvador Dalí's painting 'The Persistence of Memory'. It depicts a desolate, brown landscape under a pale sky. In the foreground, a pocket watch is melting and draped over a curved surface. To the left, a tree with a gnarled trunk stands on a small patch of ground. In the upper right, a large, melting pocket watch is visible. The overall scene is surreal and dreamlike, with distorted perspectives and melting objects.

# Lecture 13

## Temporal Processing, RNNs, Attention and Transformers

# 13. Temporal Processing and RNNs

- Sequence problems
- Temporal convnets
- Recurrent Neural Networks (RNNs)
- LSTMs
- Attention
- Example problems:
  - Image captioning
  - Sound prediction
- Transformers





**kindergarden classroom**



**television**

**person**



**chair**

**“What color is the chair?”**



**“What color is the chair?”**  
**red**



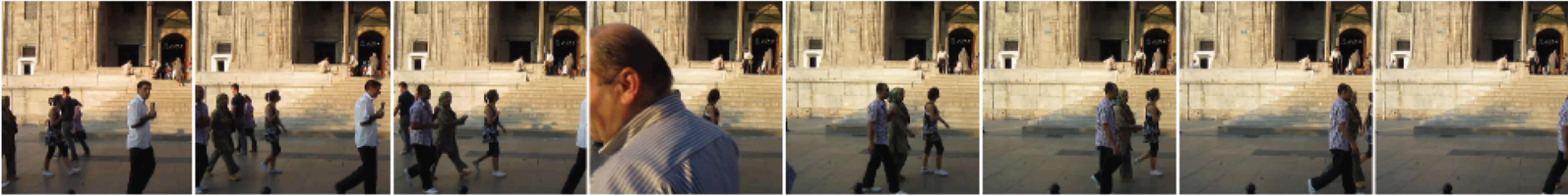
**“What will the girl do next?”**







# Sequences



time

“An”, “evening”, “stroll”, “through”, “a”, “city”, “square”

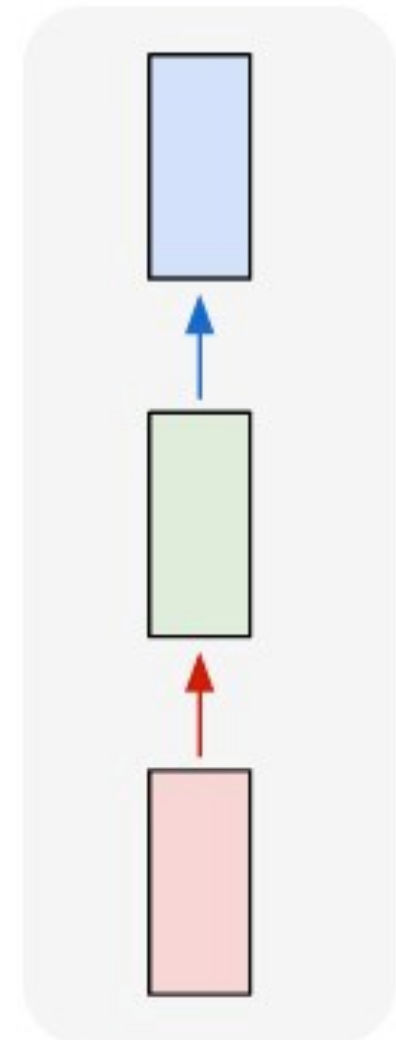
time



time

# How do we model sequences?

one to one

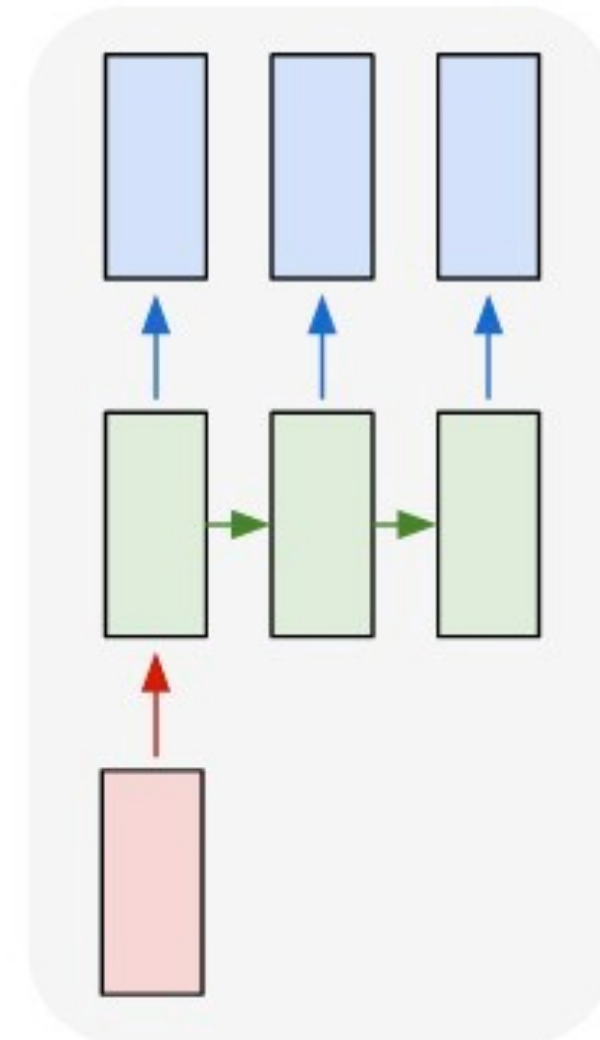


Input: No sequence

Output: No sequence

Example: "standard" classification / regression problems

one to many

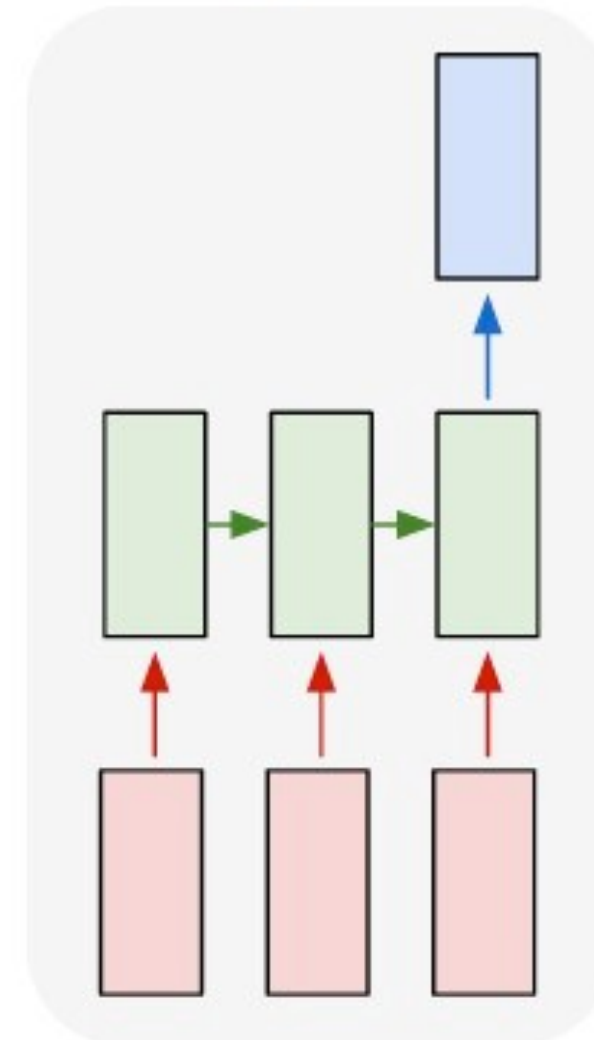


Input: No sequence

Output: Sequence

Example: Im2Caption

many to one

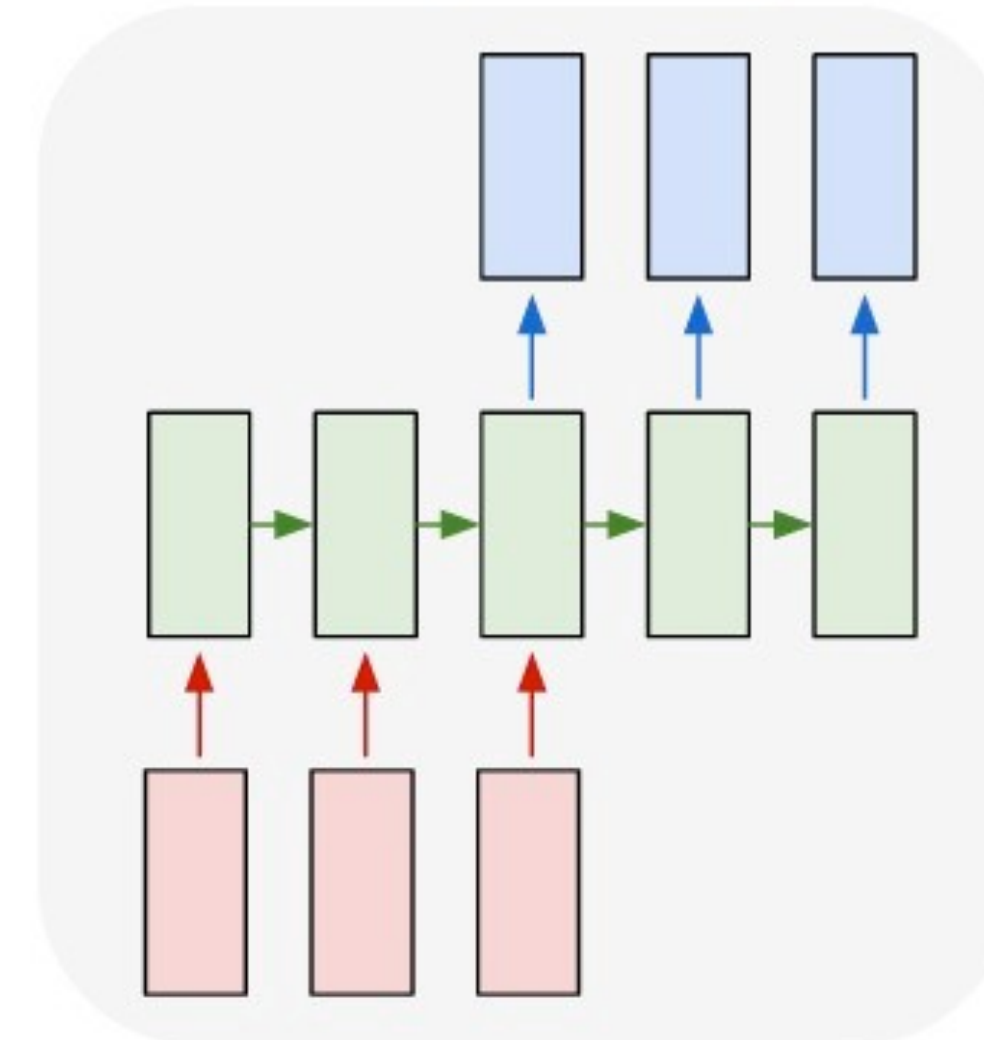


Input: Sequence

Output: No sequence

Example: sentence classification, multiple-choice question answering

many to many

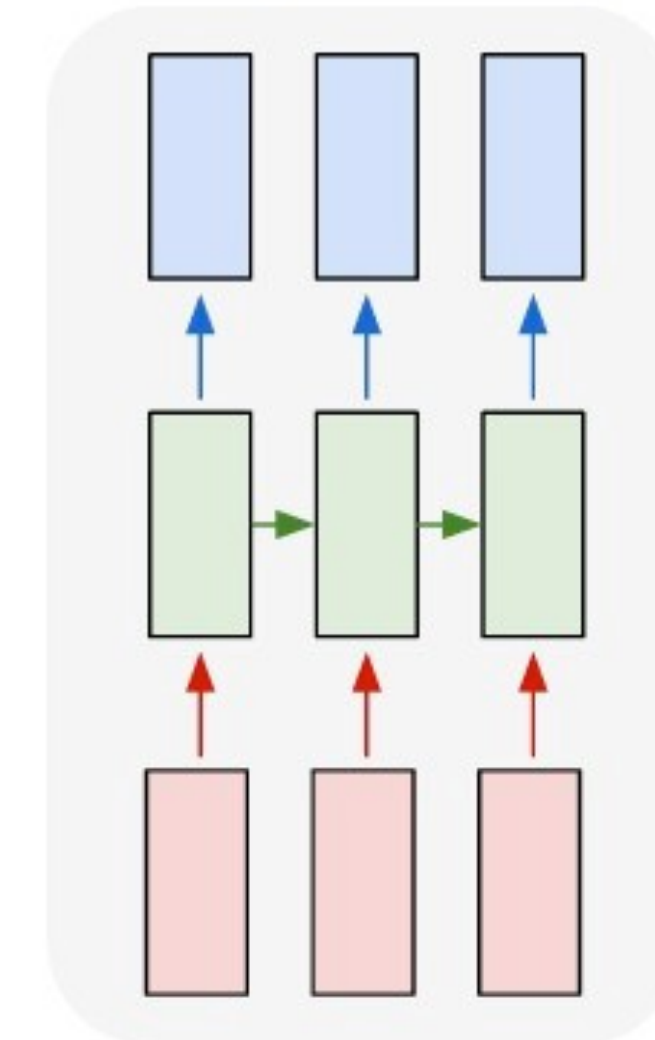


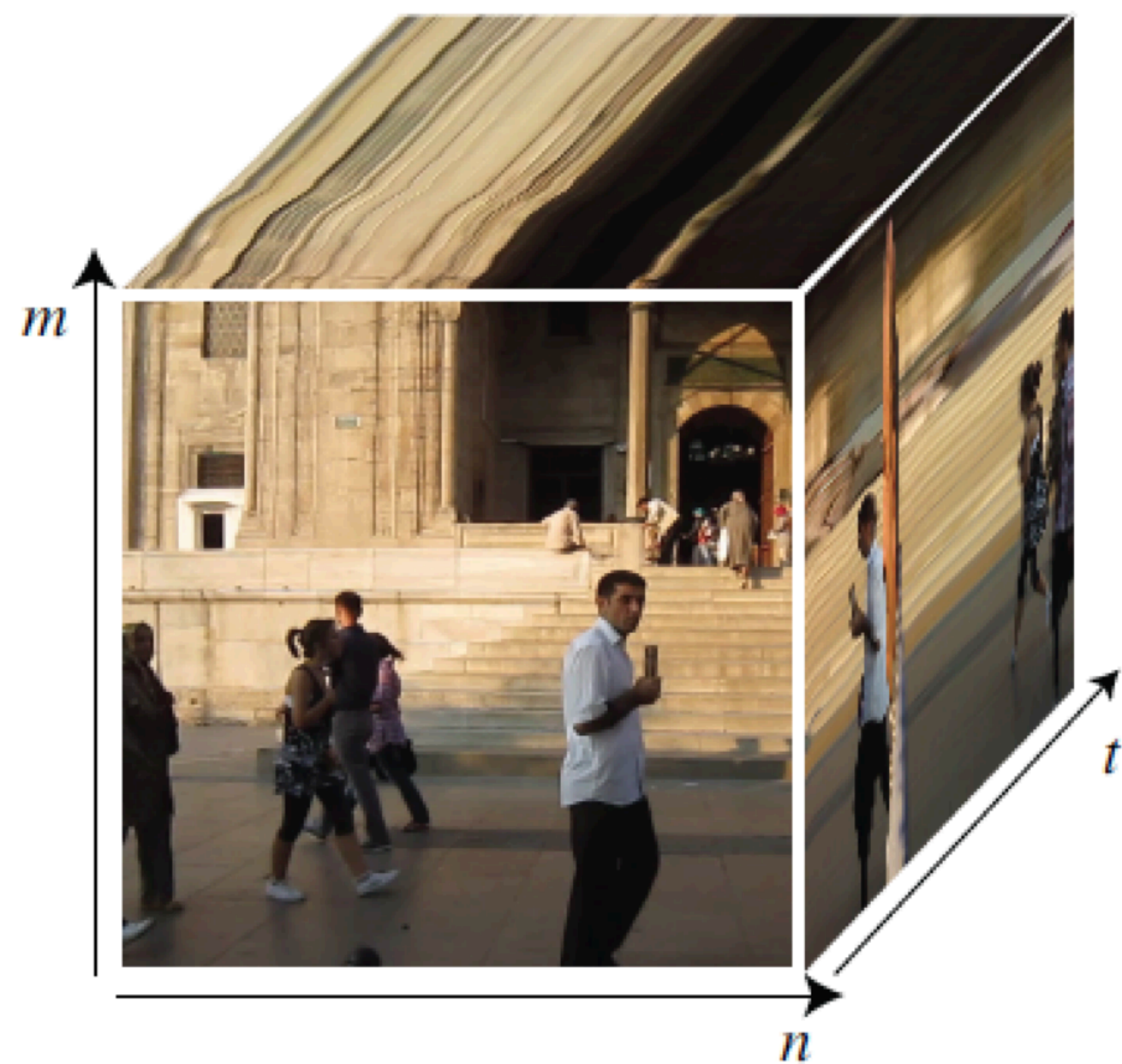
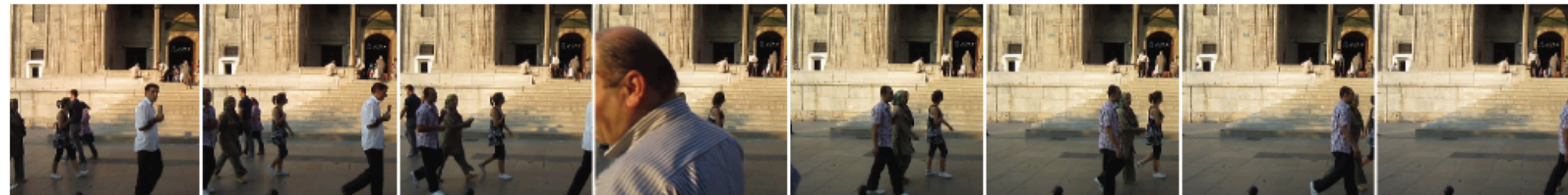
Input: Sequence

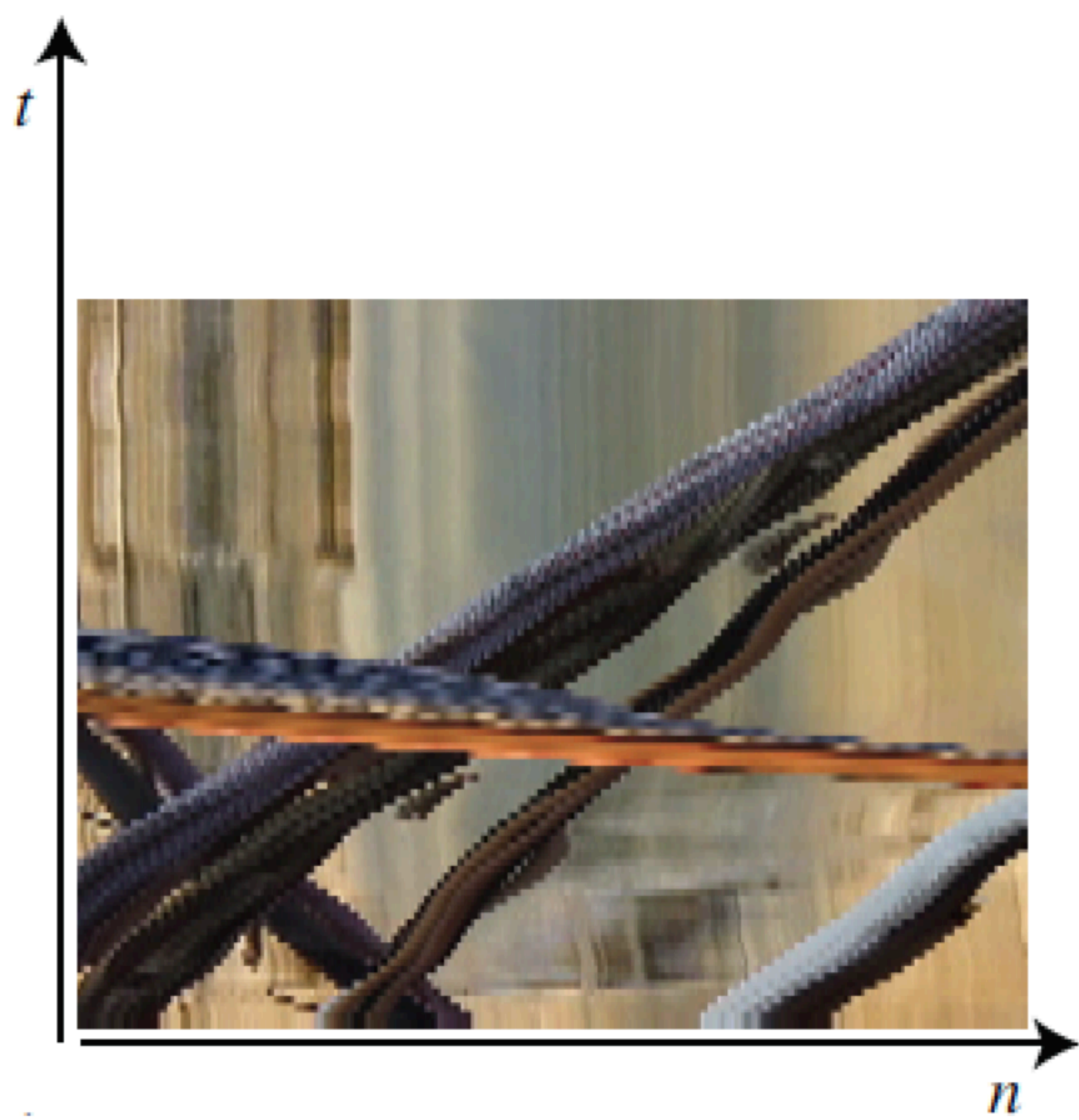
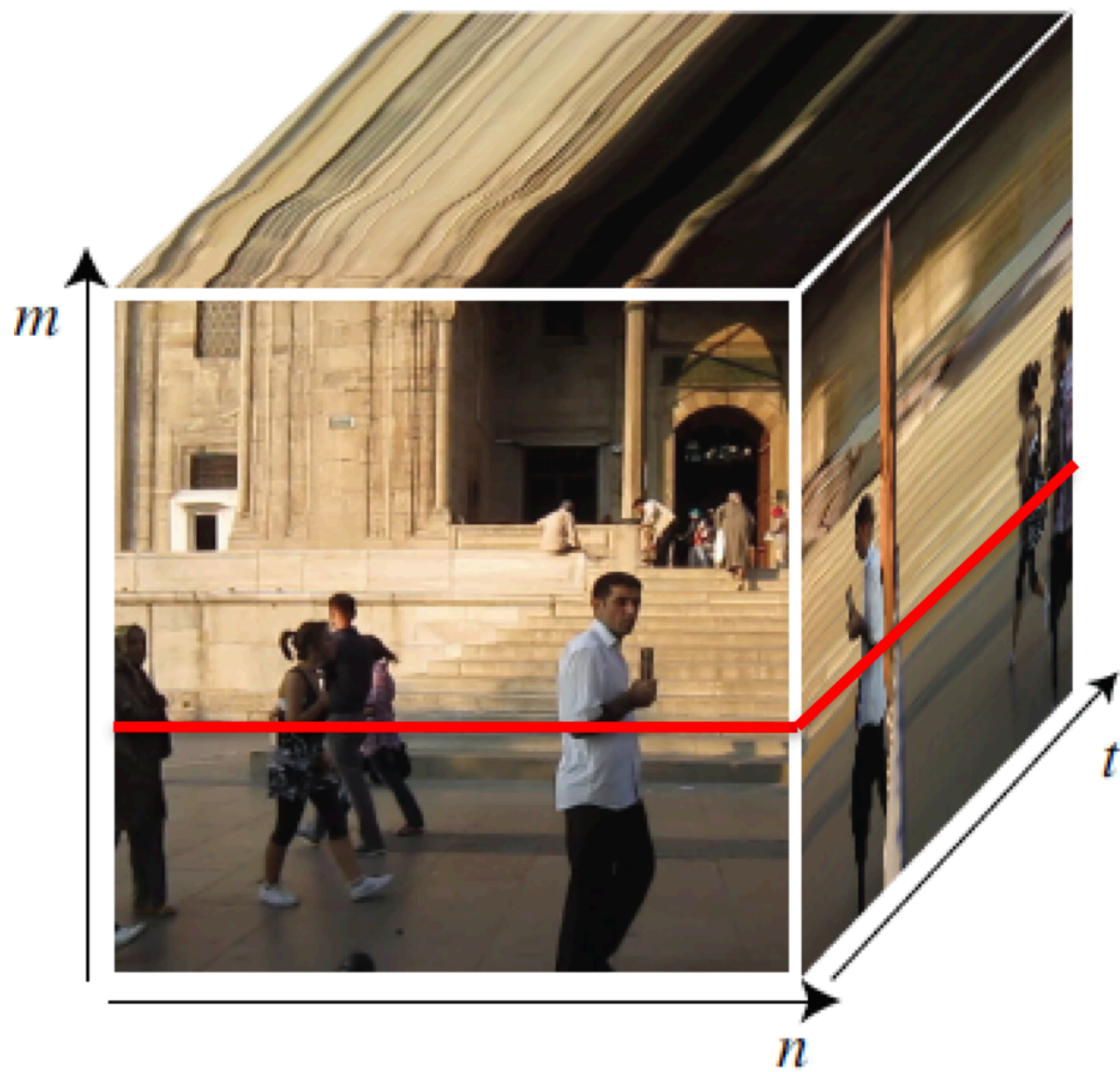
Output: Sequence

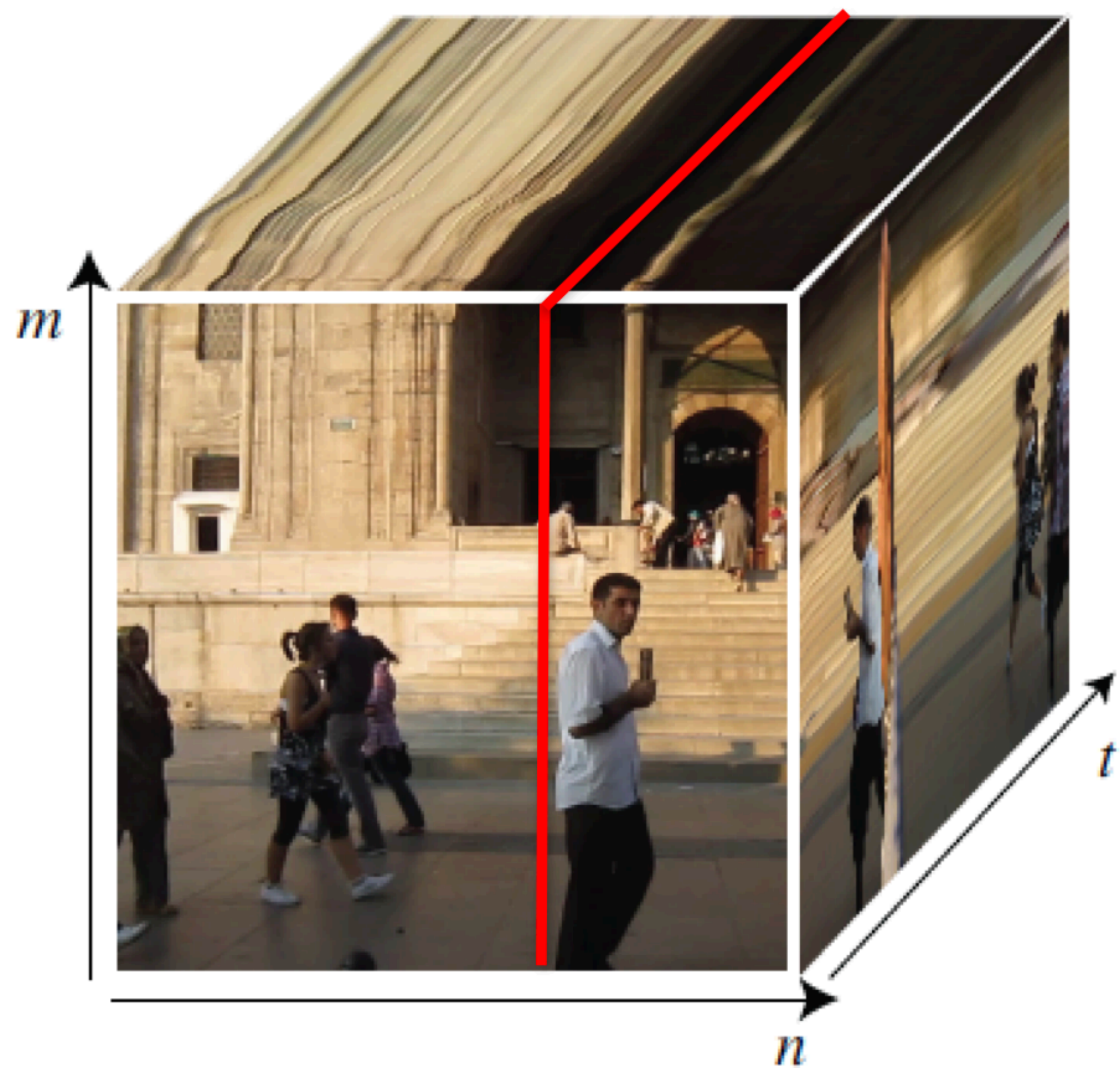
Example: machine translation, video captioning, open-ended question answering, video question answering

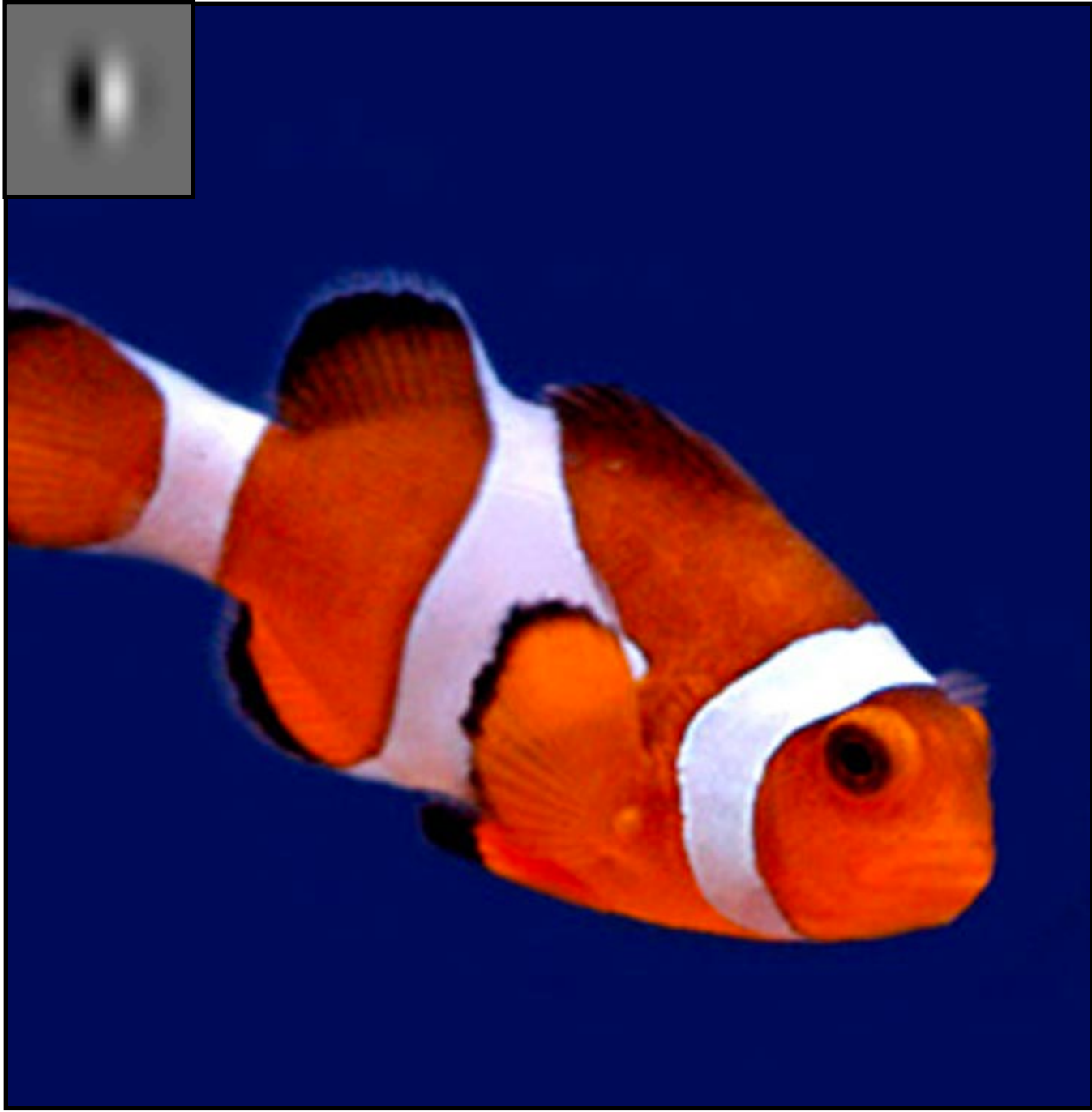
many to many



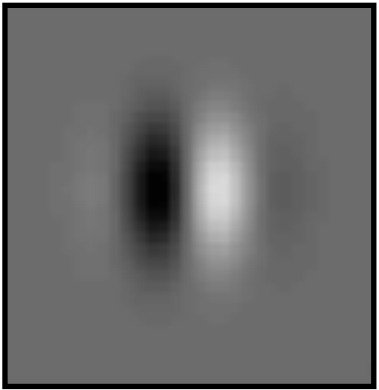




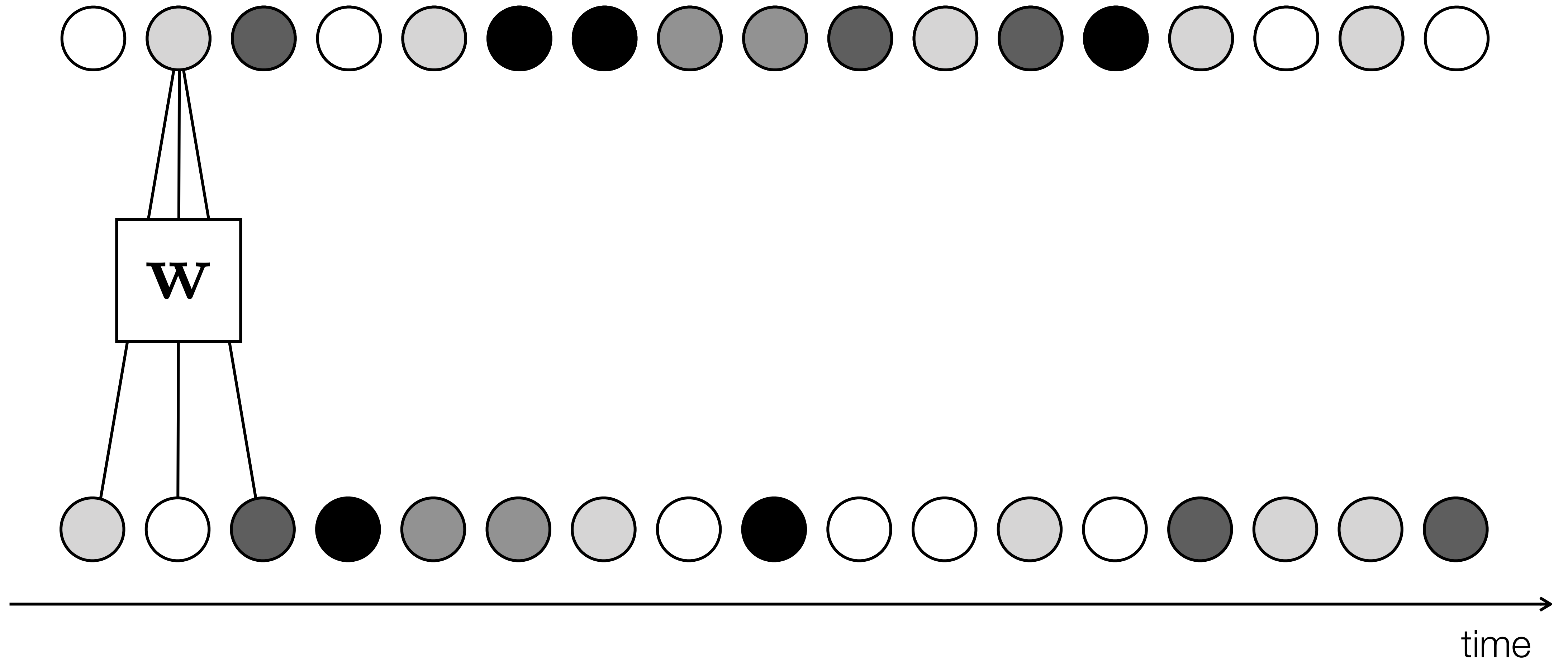




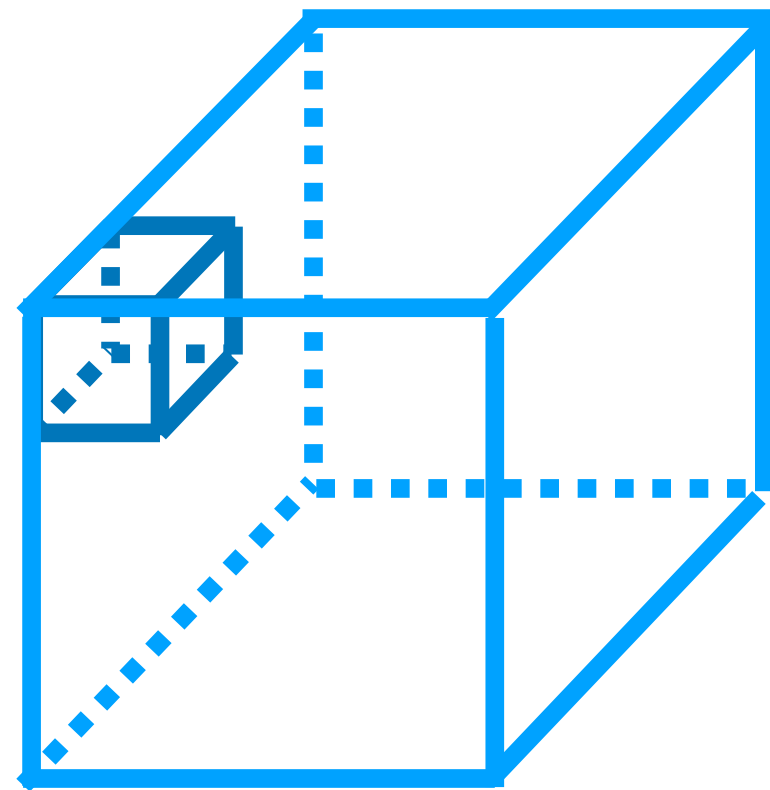
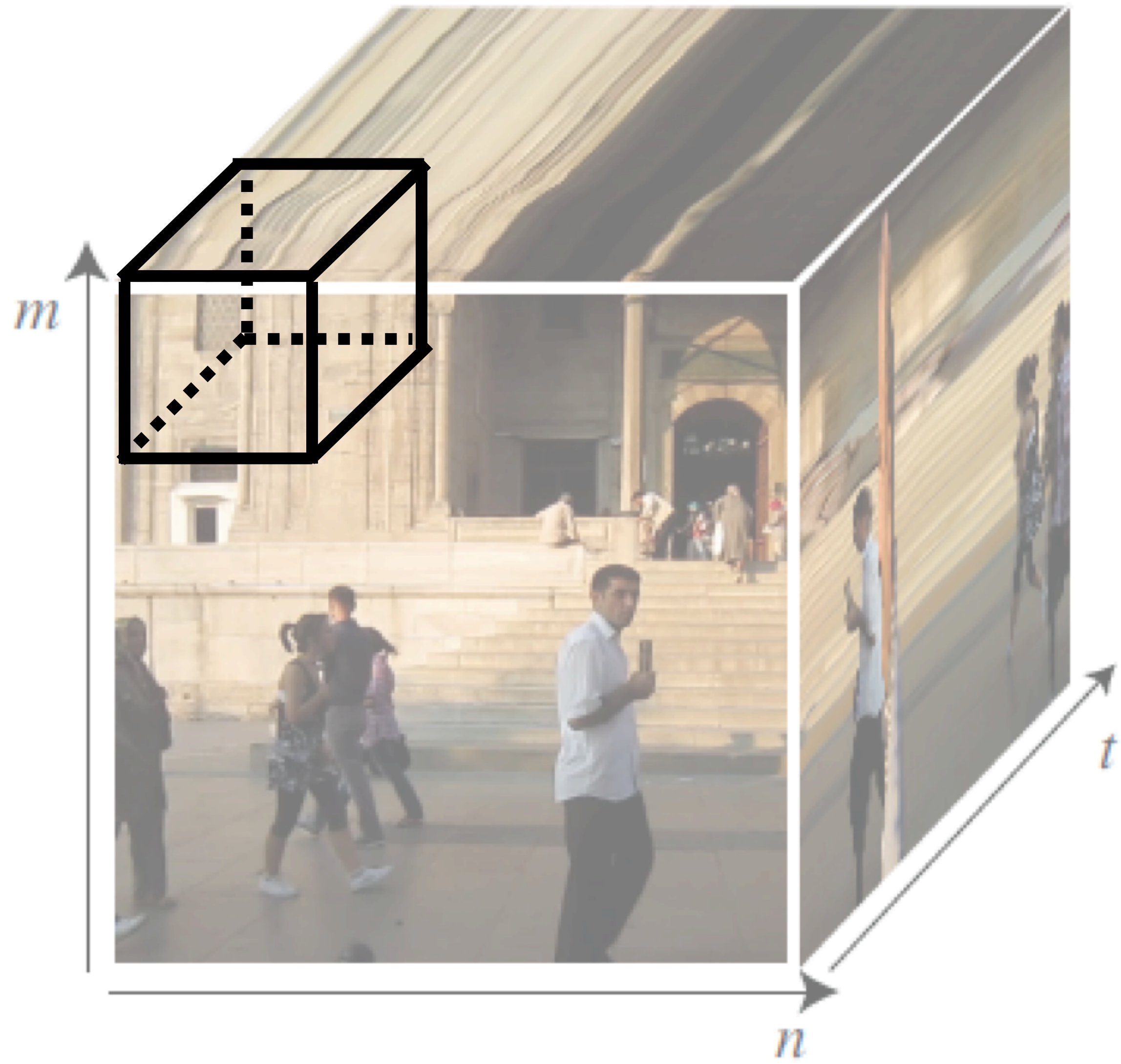
filter

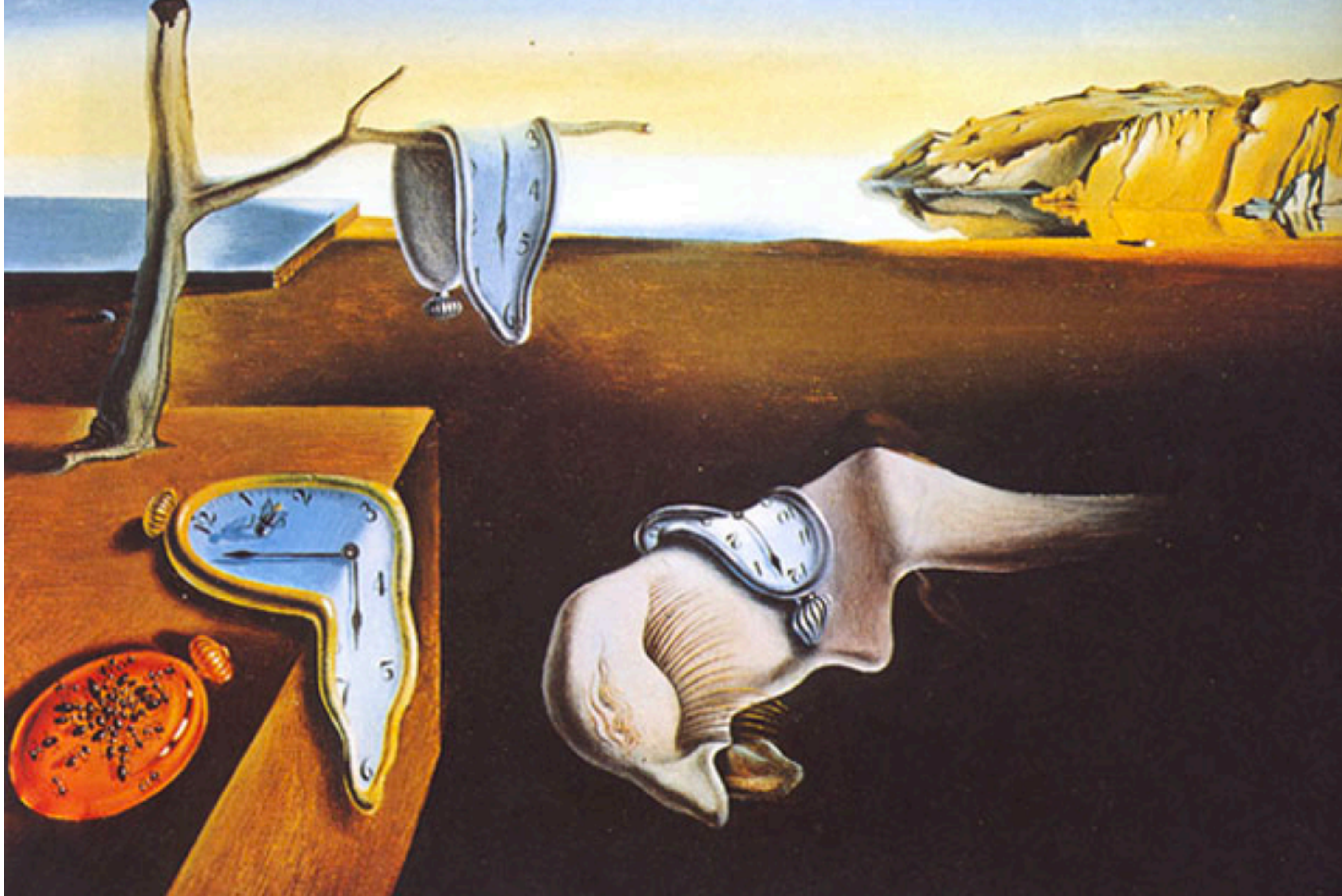


# Convolutions in time







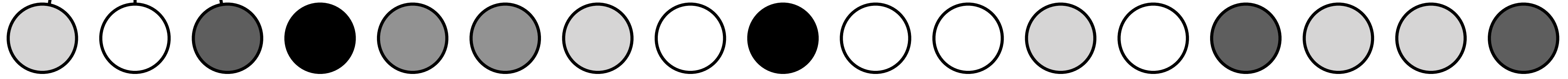
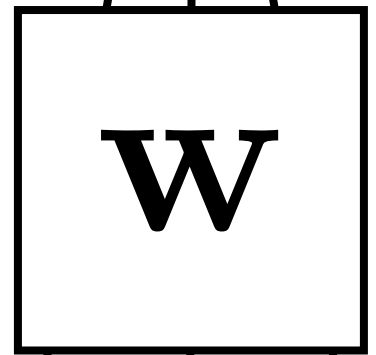
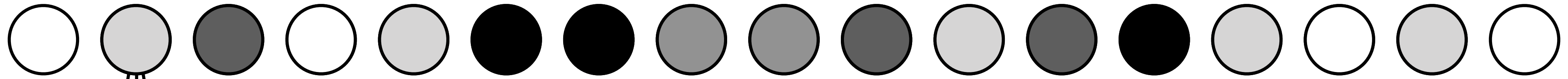


“The Persistence of Memory”,  
Dali 1931

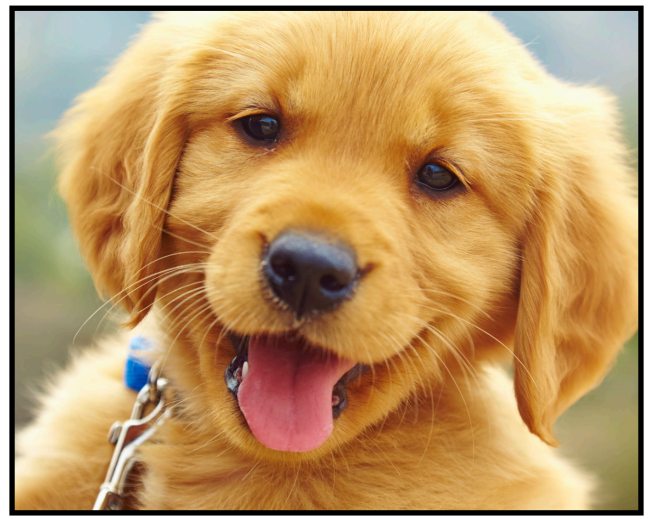
It bothered him that the dog at three fourteen (seen from the side) should have the same name as the dog at three fifteen (seen from the front).

— “Funes the Memorius”, Borges 1962

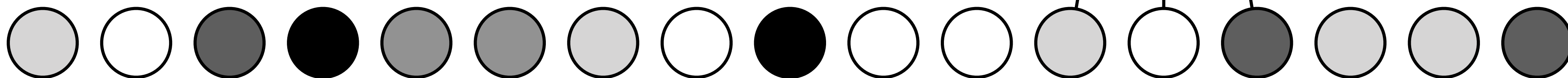
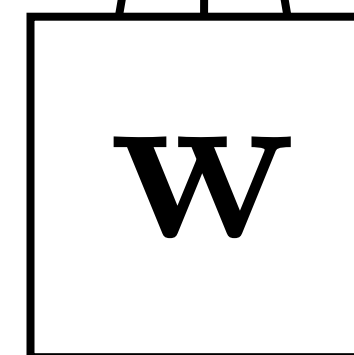
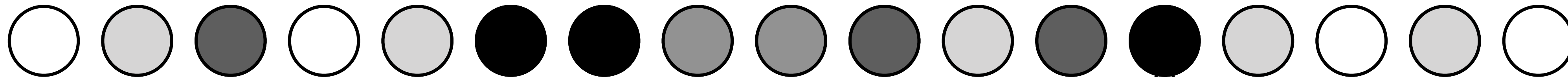
Rufus



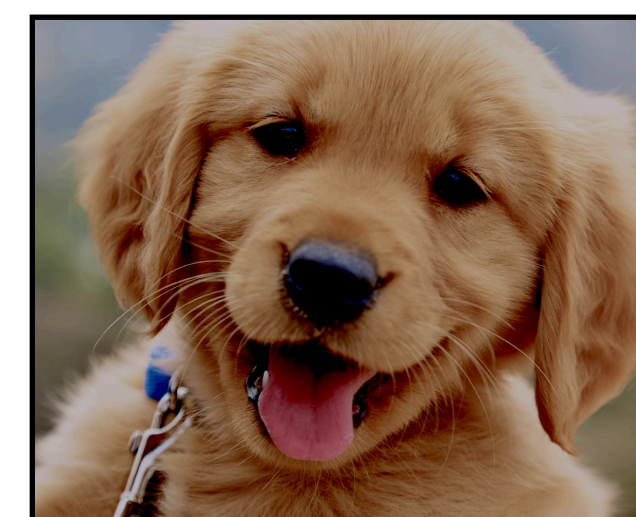
time



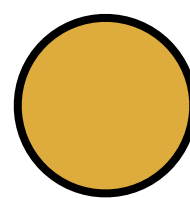
Douglas



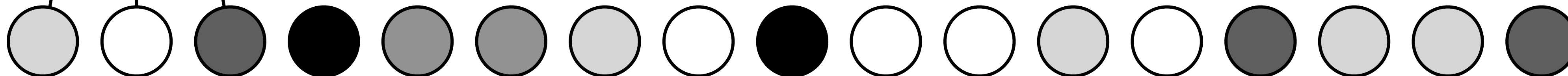
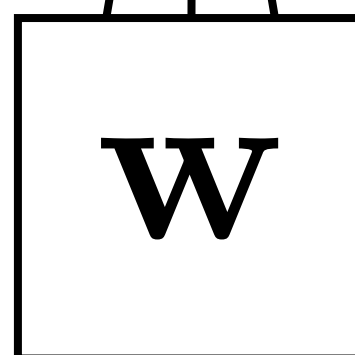
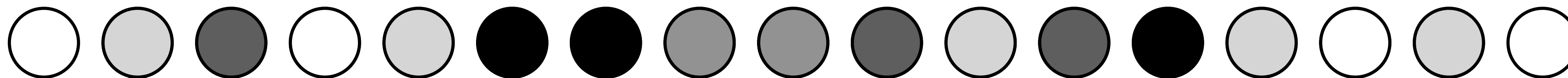
time



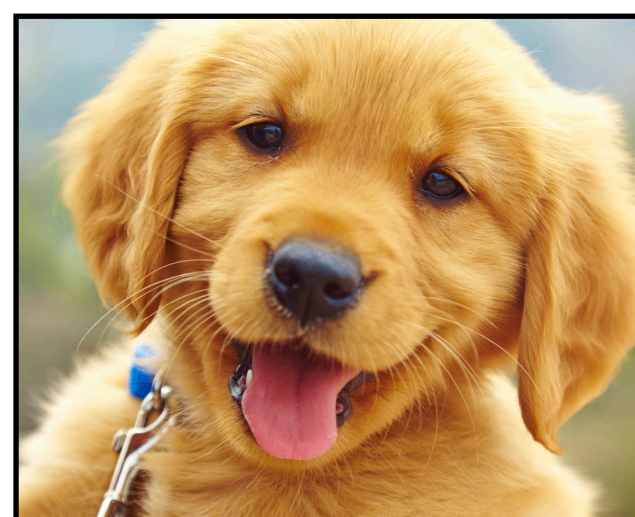
Memory  
unit



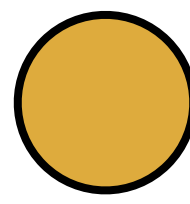
Rufus



time

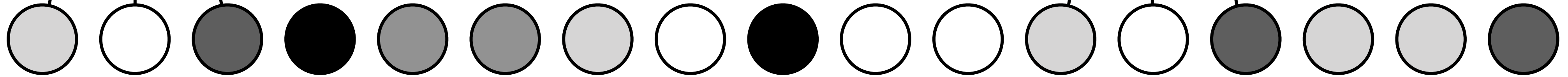
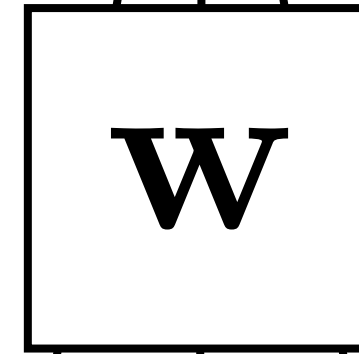
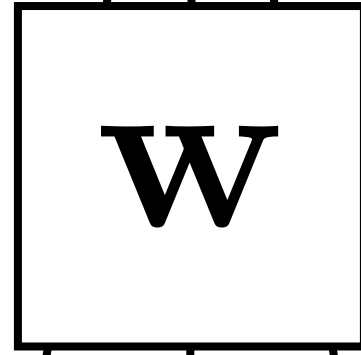
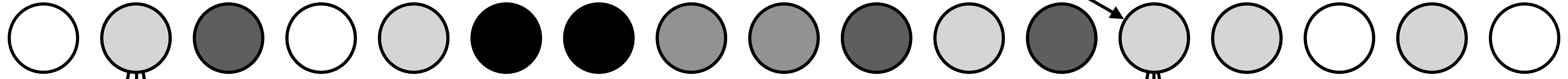


Memory unit



Rufus

Rufus!



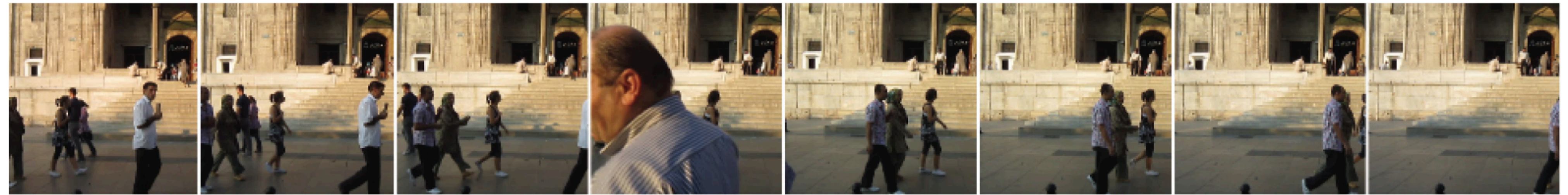
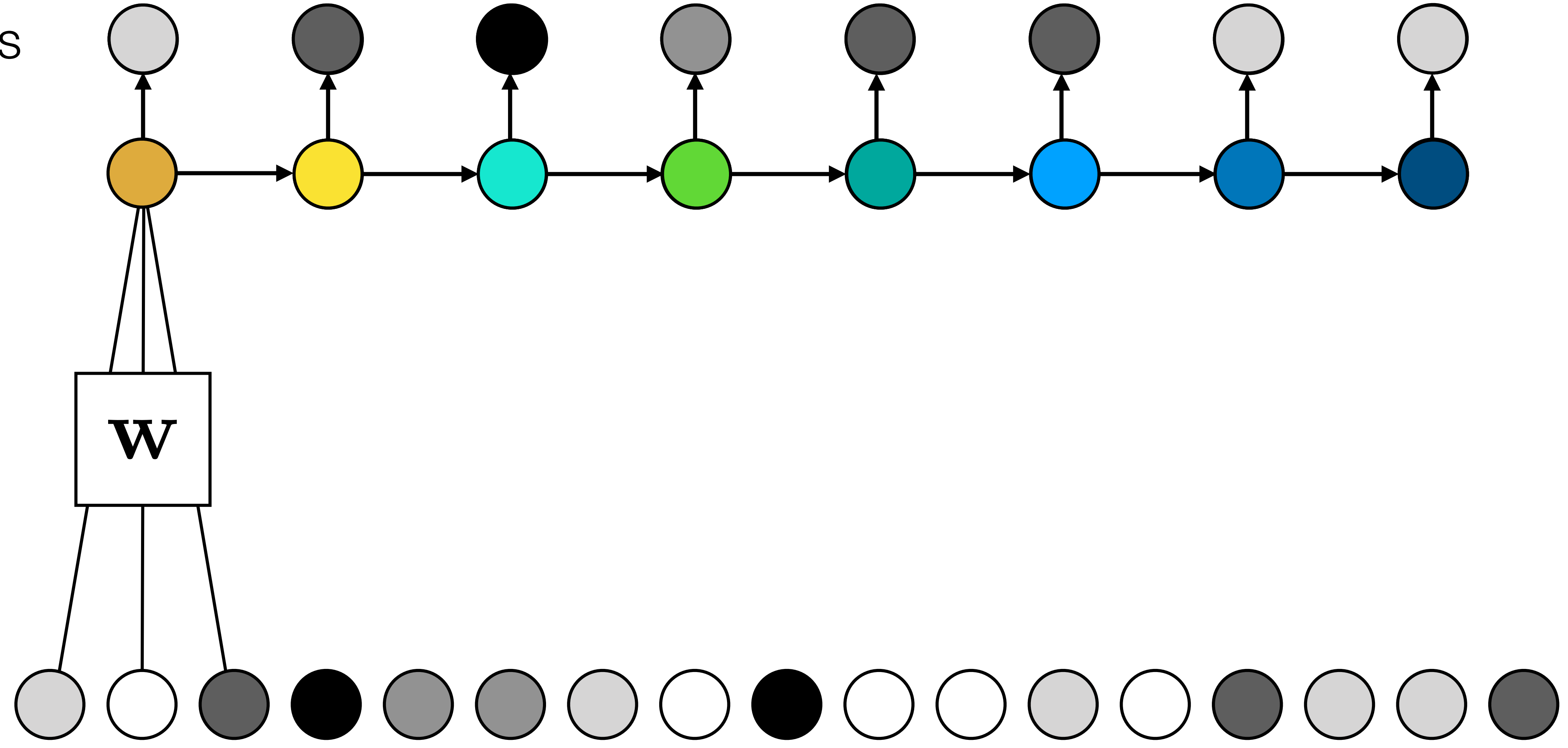
time

# Recurrent Neural Networks (RNNs)

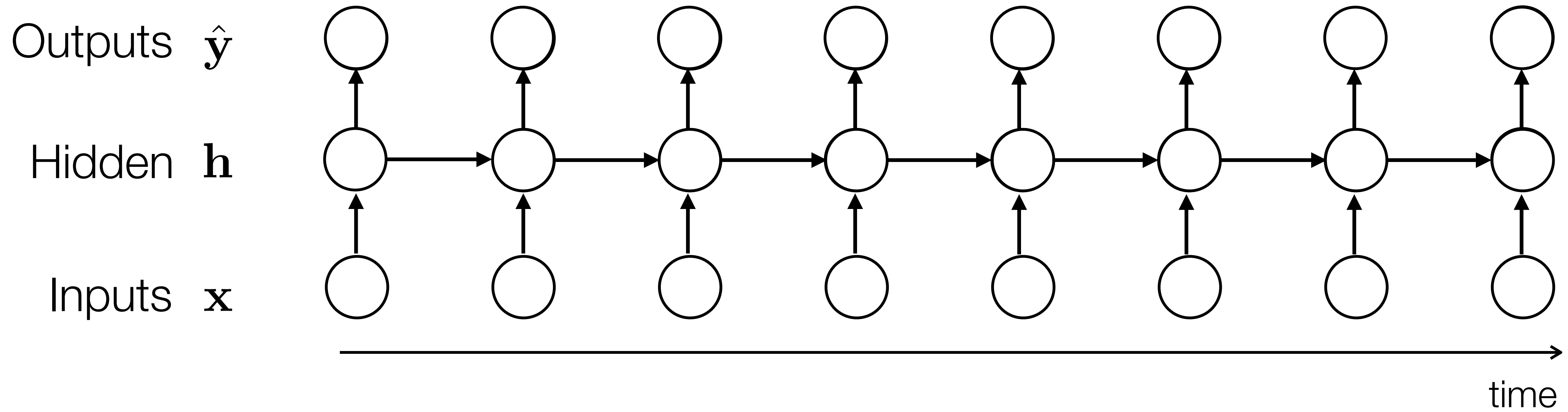
Outputs

Hidden

Inputs

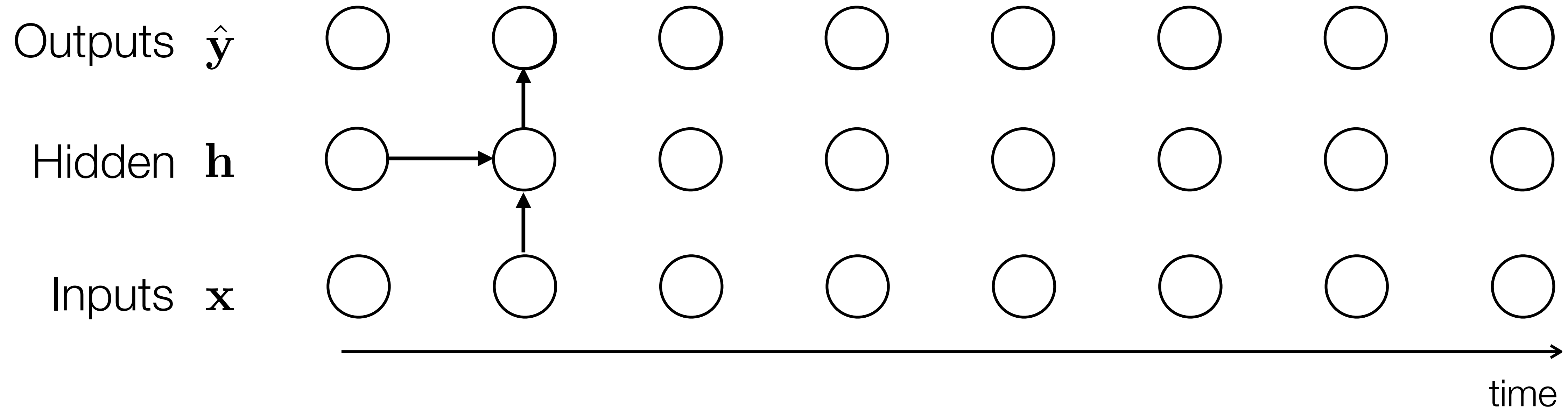


# Recurrent Neural Networks (RNNs)





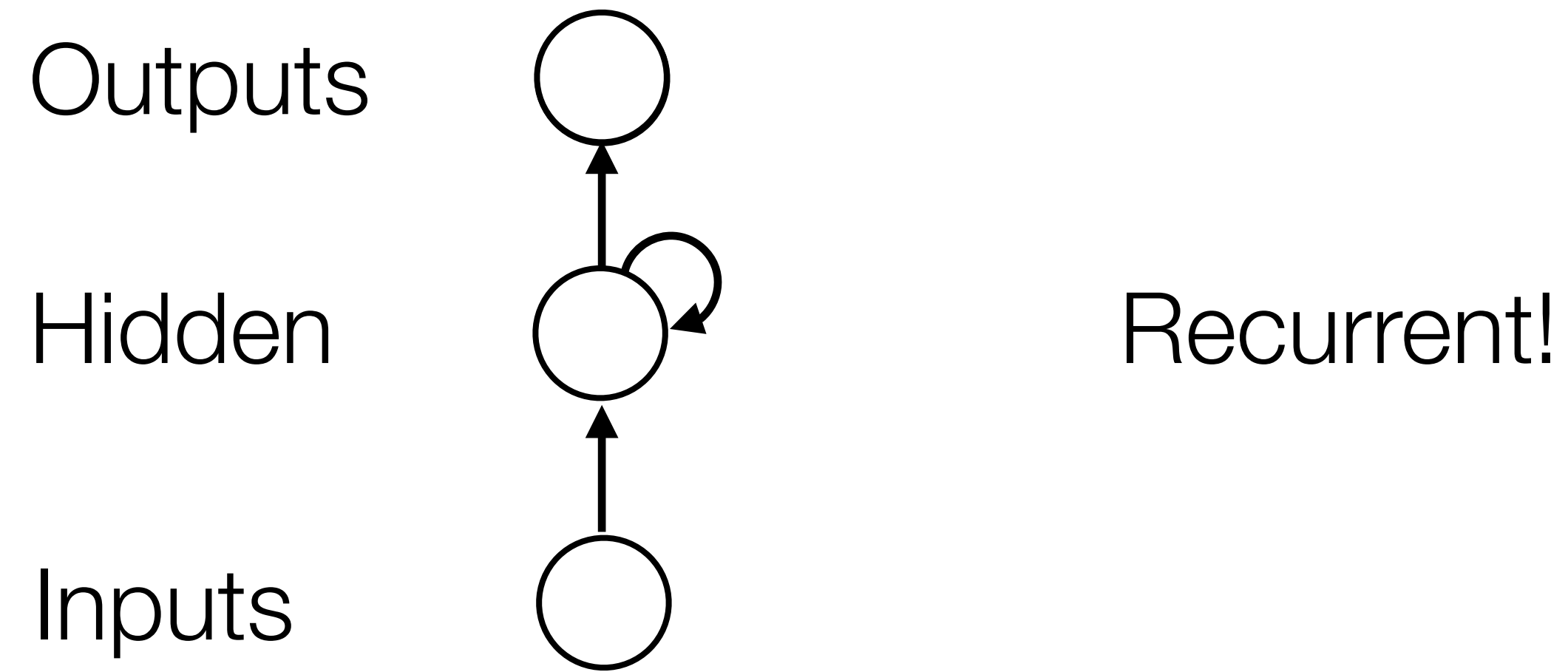
# Recurrent Neural Networks (RNNs)



$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

$$\mathbf{y}_t = g(\mathbf{h}_t)$$

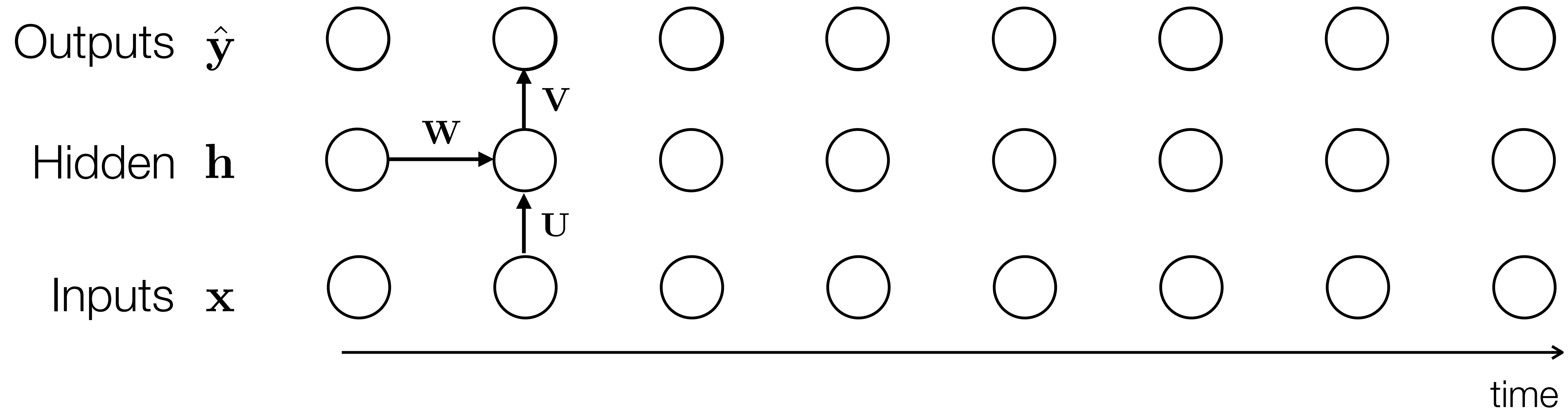
# Recurrent Neural Networks (RNNs)



$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

$$\mathbf{y}_t = g(\mathbf{h}_t)$$

# Recurrent Neural Networks (RNNs)



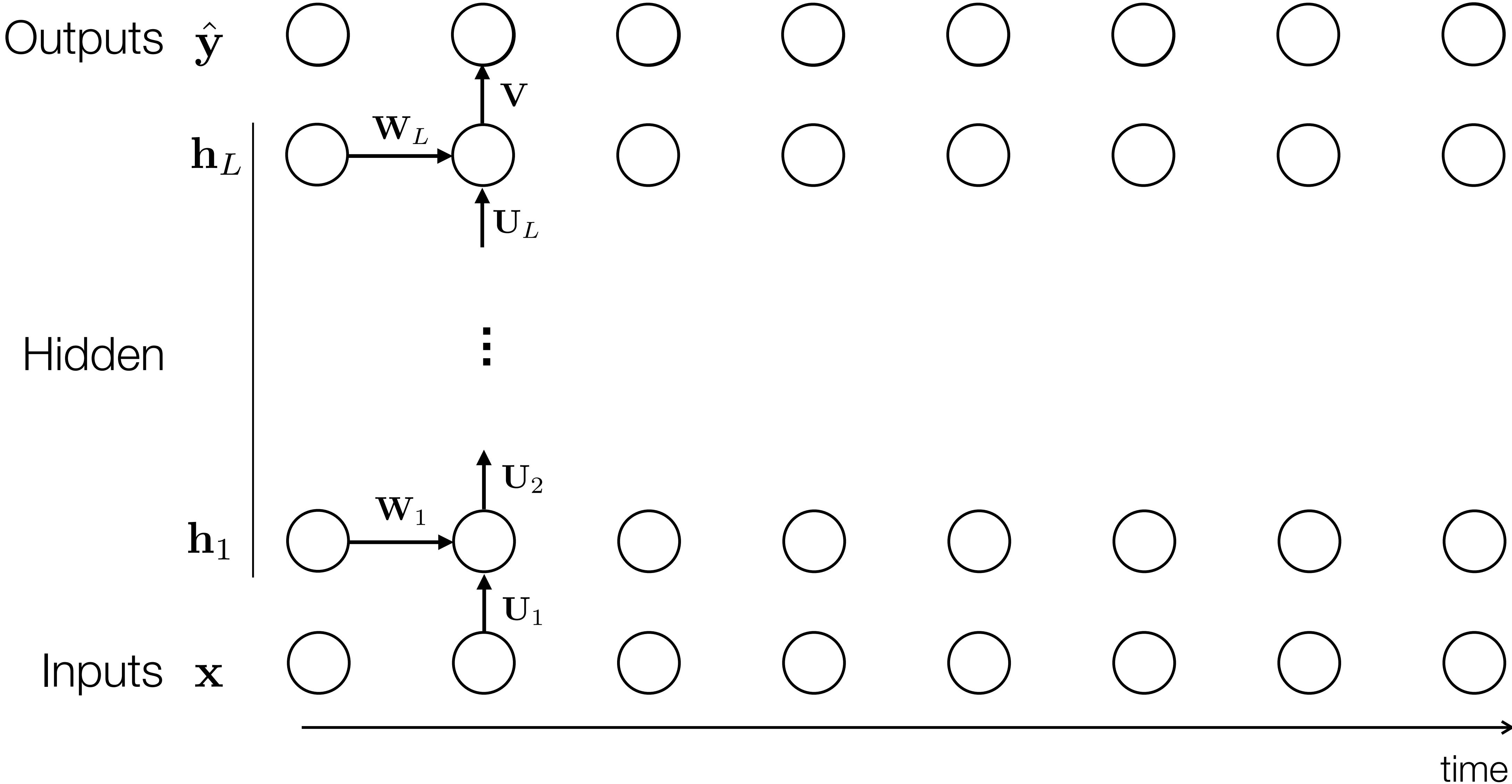
$$\mathbf{a}_t = \mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}$$

$$\mathbf{h}_t = \tanh(\mathbf{a}_t)$$

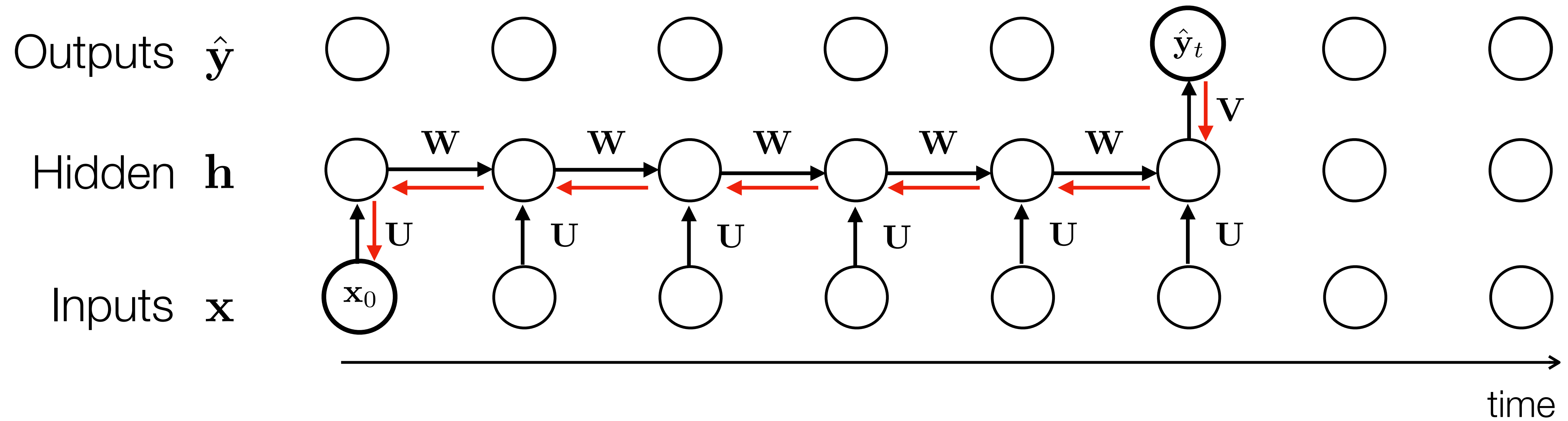
$$\mathbf{o}_t = \mathbf{V}\mathbf{h}_t + \mathbf{c}$$

$$\hat{\mathbf{y}}_t = \text{softmax}(\mathbf{o}_t)$$

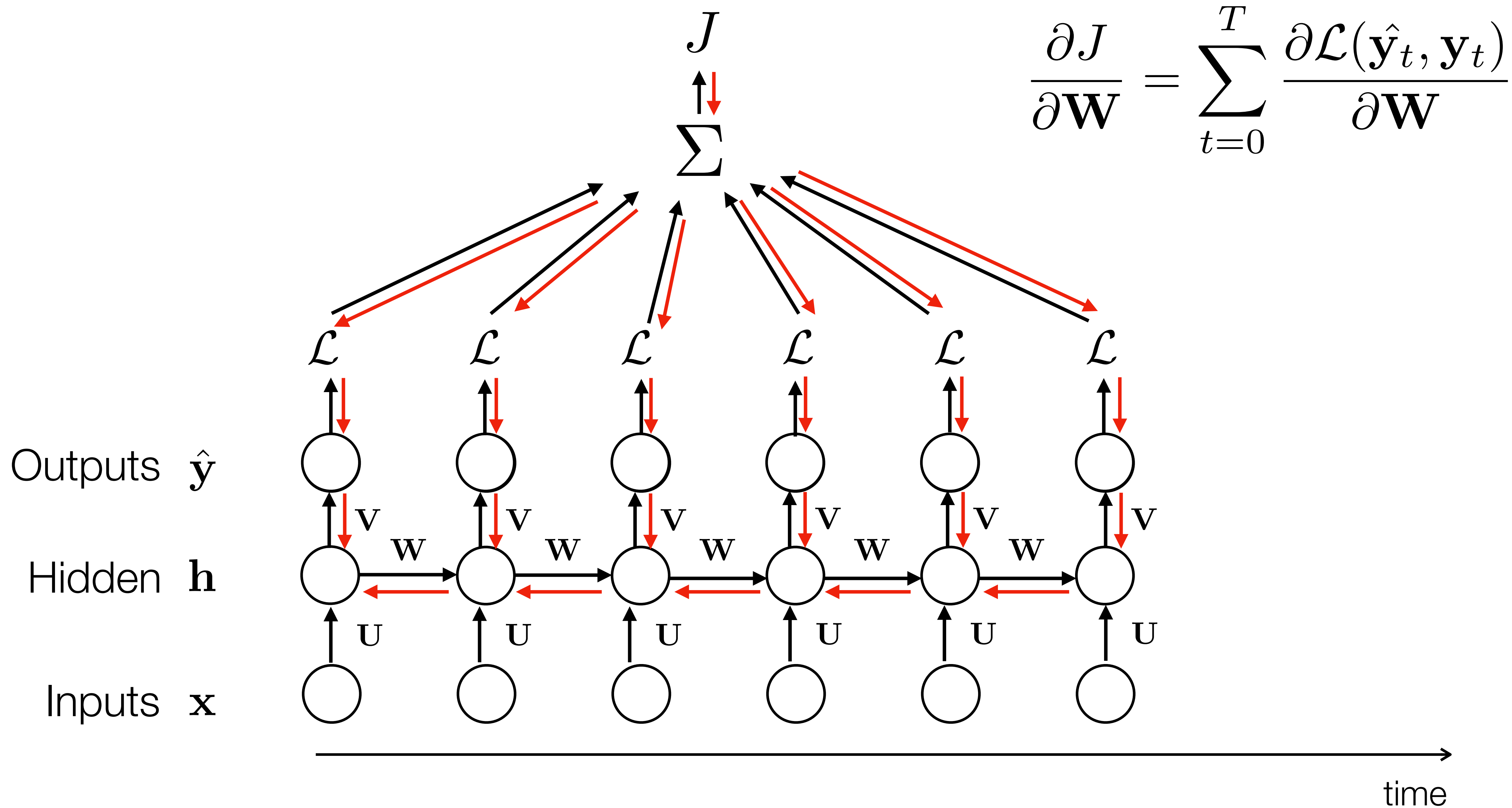
# Deep Recurrent Neural Networks (RNNs)



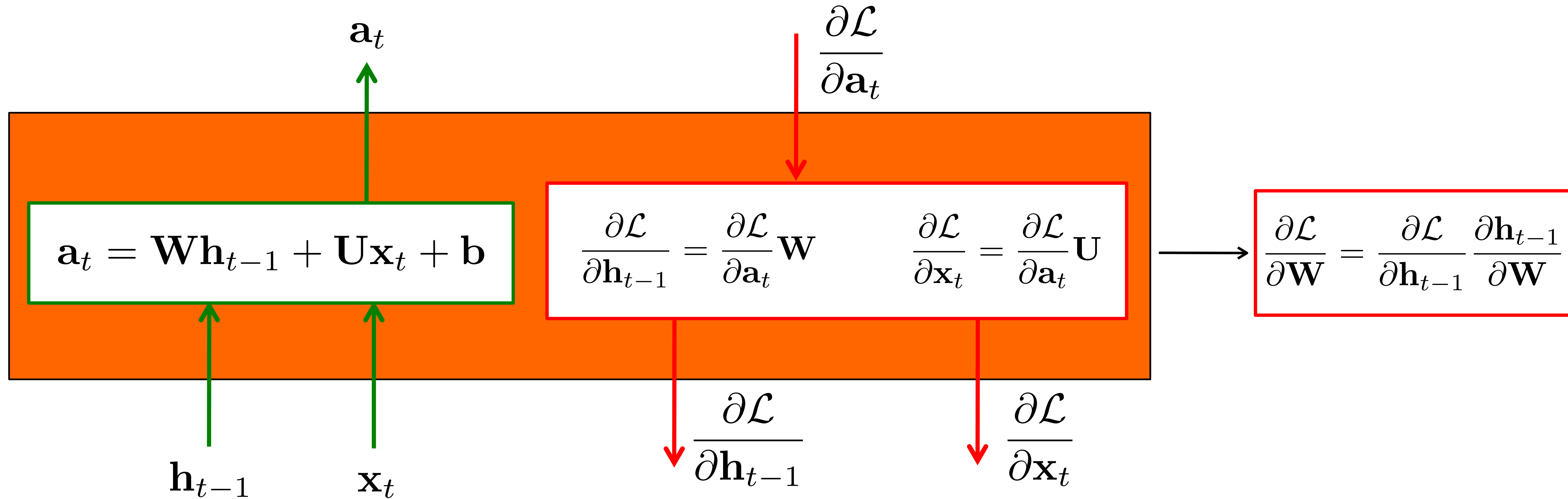
# Backprop through time



$$\frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{x}_0} = \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \dots \frac{\partial \mathbf{h}_1}{\partial \mathbf{h}_0} \frac{\partial \mathbf{h}_0}{\partial \mathbf{x}_0}$$



# Recurrent linear layer



$$\frac{\partial J}{\partial \mathbf{W}} = \sum_{t=0}^T \frac{\partial \mathcal{L}(\hat{\mathbf{y}}_t, \mathbf{y}_t)}{\partial \mathbf{W}}$$

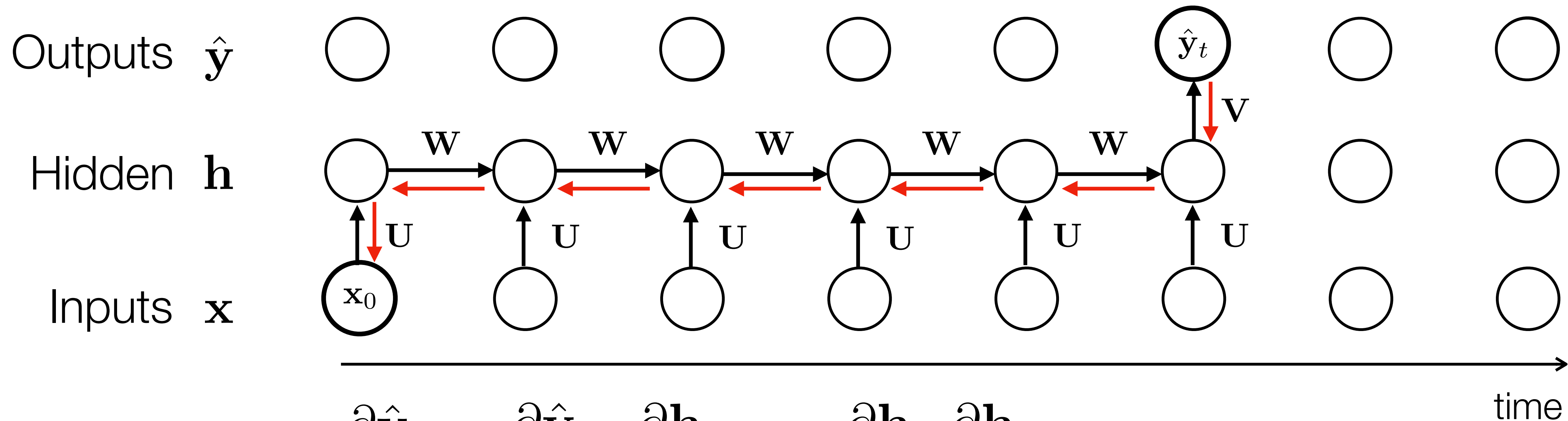
# The problem of long-range dependences

Why not remember everything?

- Memory size grows with  $t$
- This kind of memory is **nonparametric**: there is no finite set of parameters we can use to model it
- RNNs make a Markov assumption — the future hidden state only depends on the immediately preceding hidden state
- By putting the right info in to the hidden state, RNNs can model dependences that are arbitrarily far apart



# The problem of long-range dependences



$$\frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{x}_0} = \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \dots \frac{\partial \mathbf{h}_1}{\partial \mathbf{h}_0} \frac{\partial \mathbf{h}_0}{\partial \mathbf{x}_0}$$

- Capturing long-range dependences requires propagating information through a long chain of dependences.
- Old observations are forgotten
- Stochastic gradients become high variance (noisy), and gradients may **vanish** or **explode**

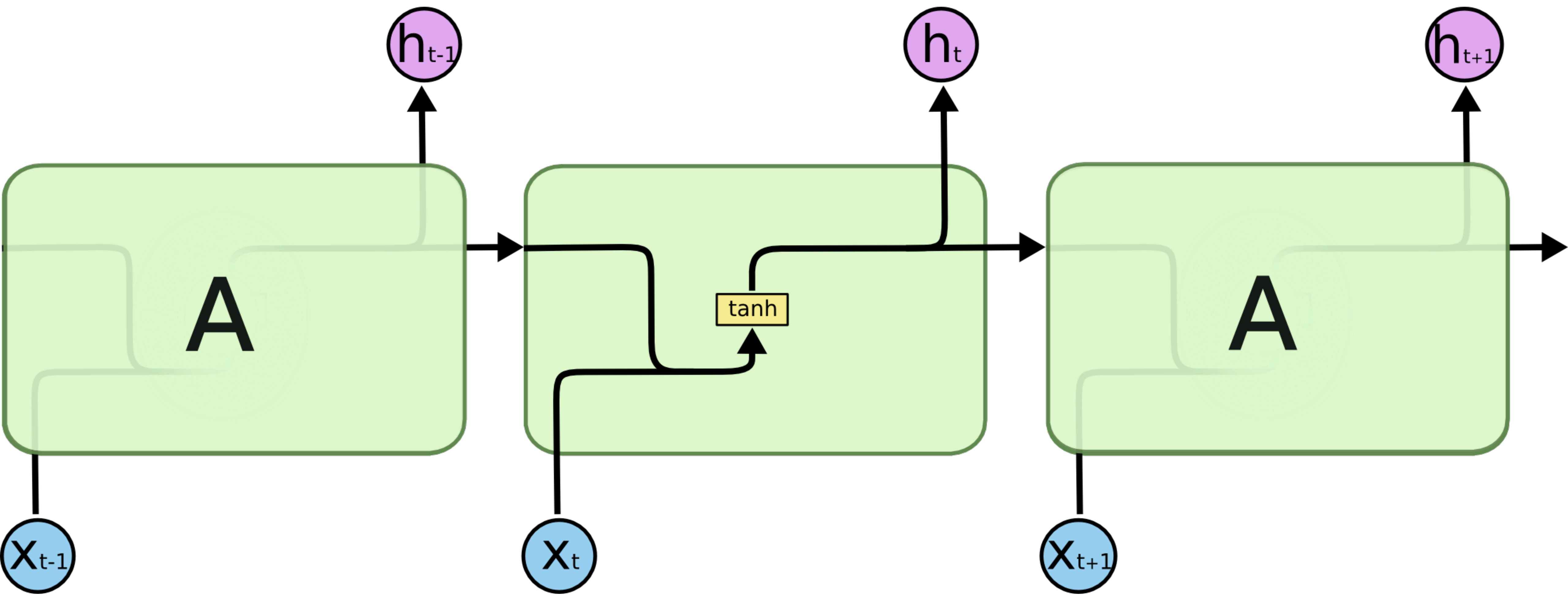
# LSTMs

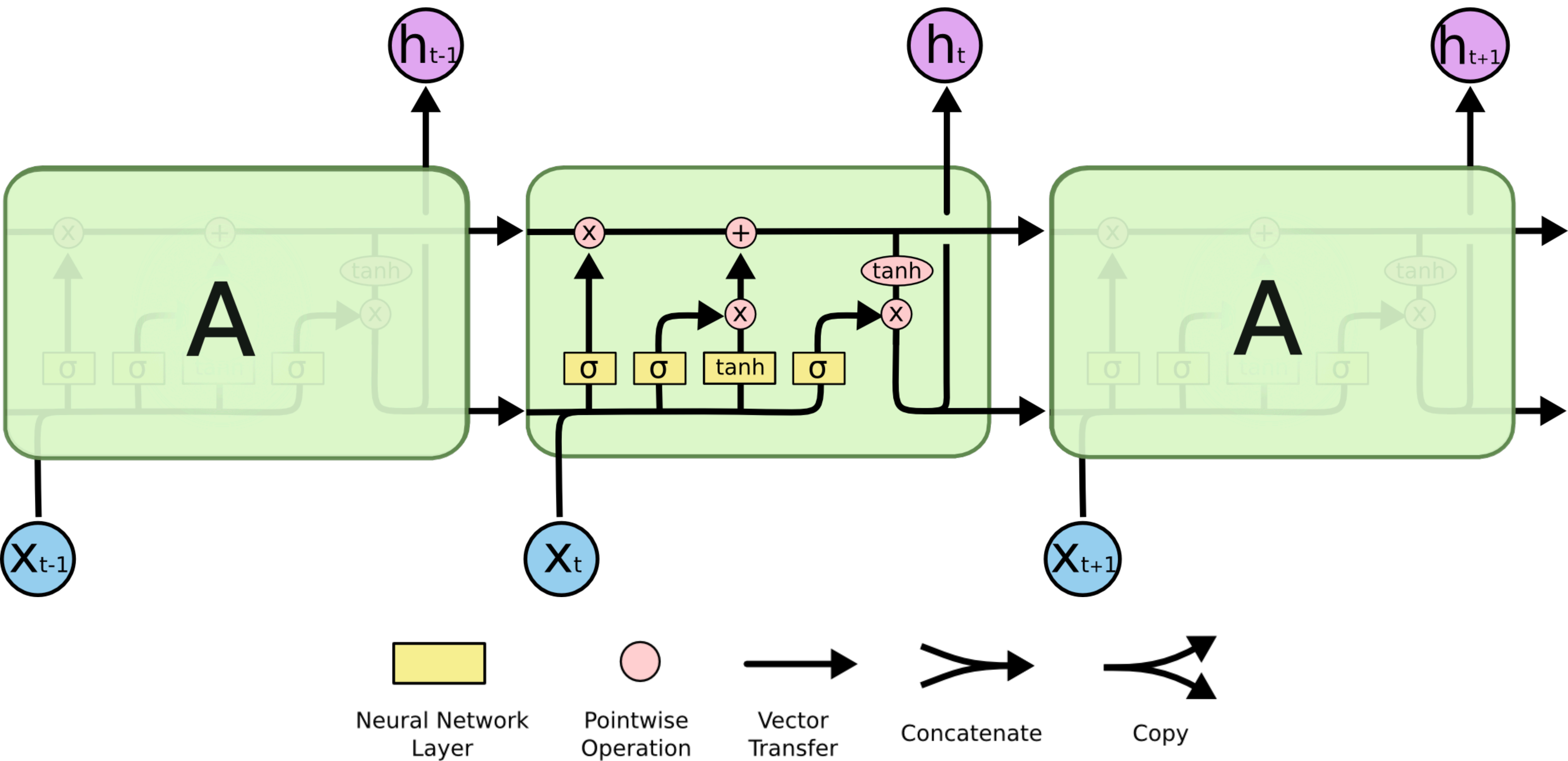
Long Short Term Memory

[Hochreiter & Schmidhuber, 1997]

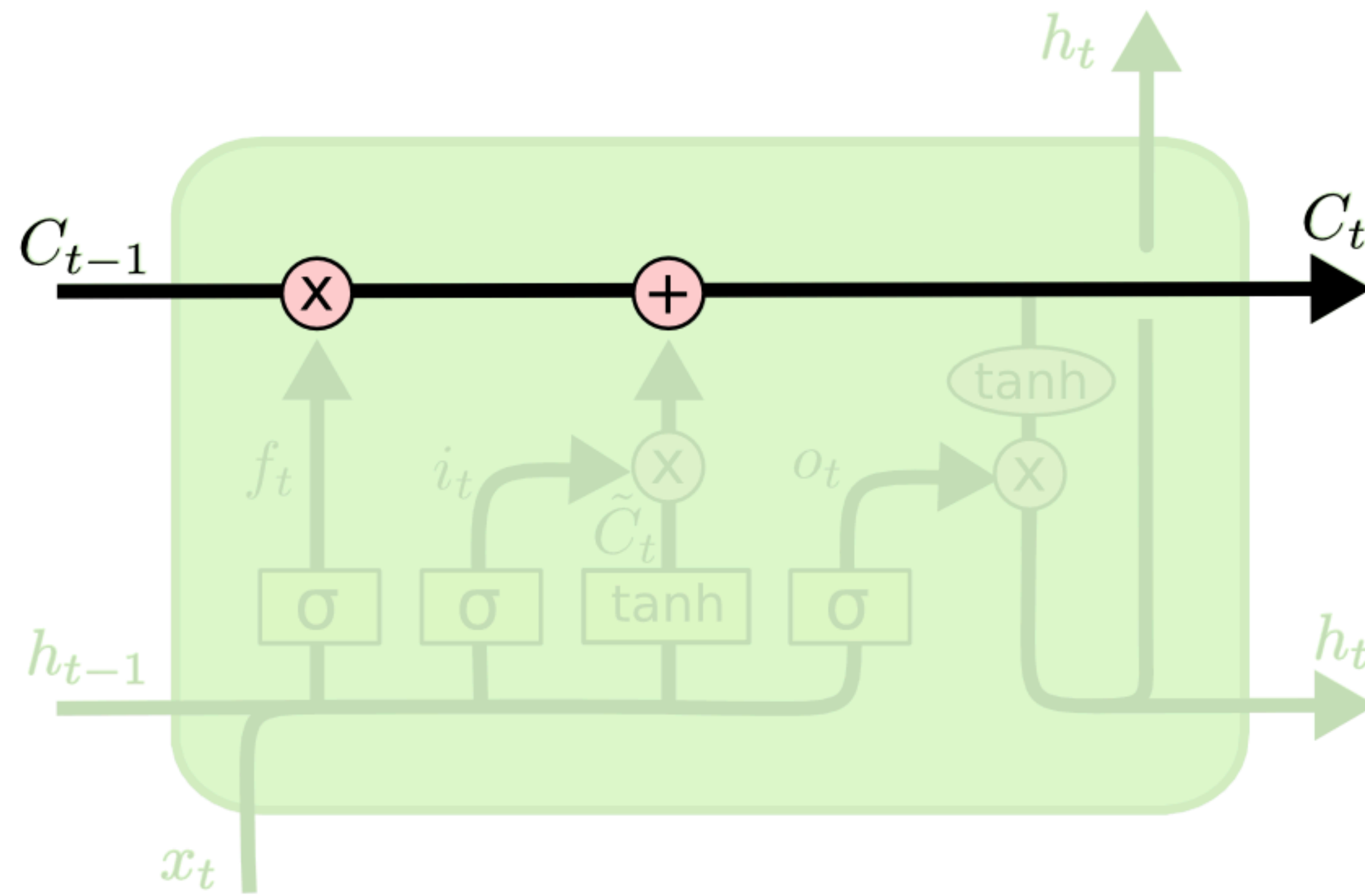
A special kind of RNN designed to avoid forgetting.

This way the default behavior is not to forget an old state. Instead of forgetting by default, the network has to *learn to forget*.

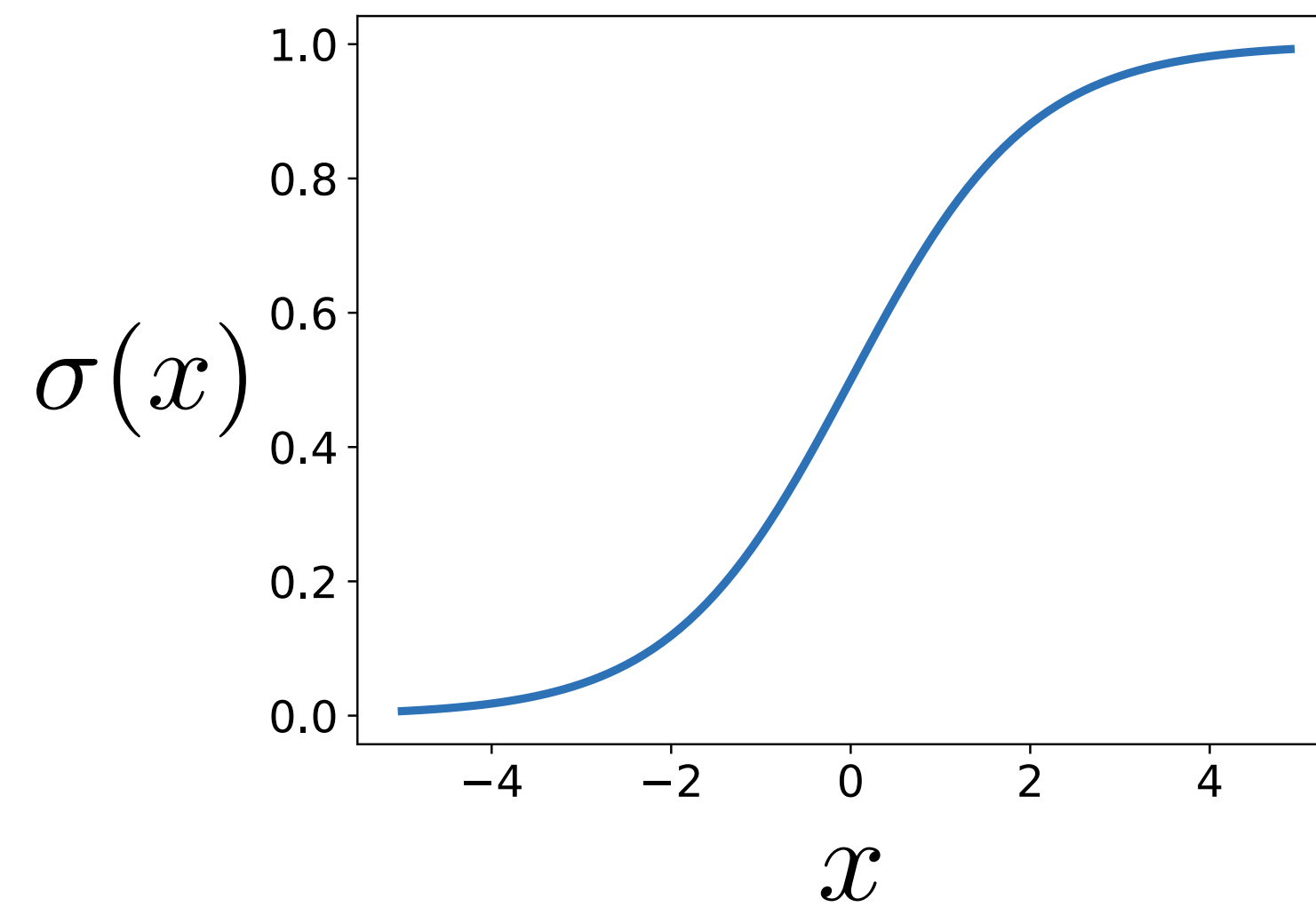
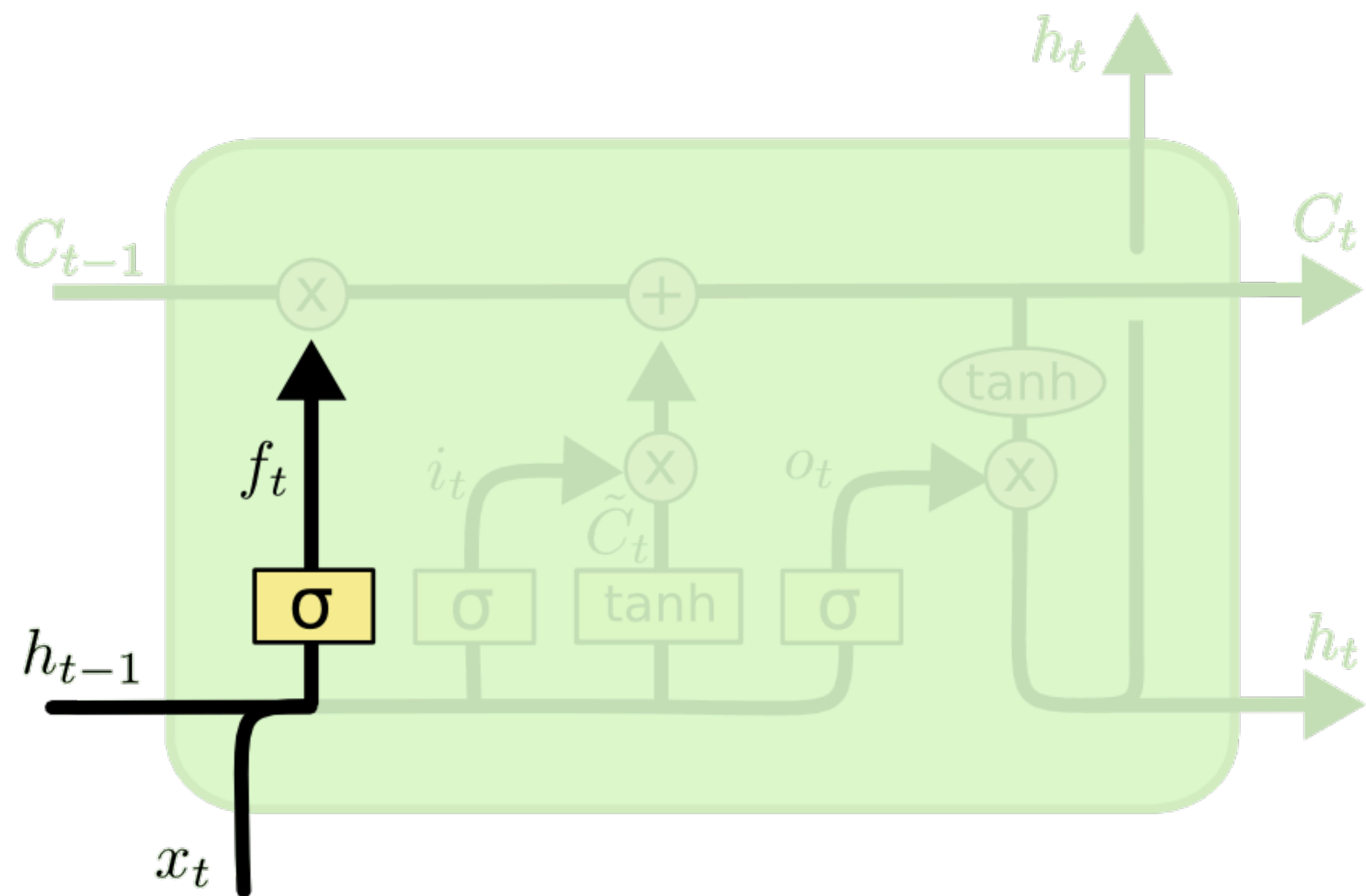




[Slide derived from Chris Olah: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>]



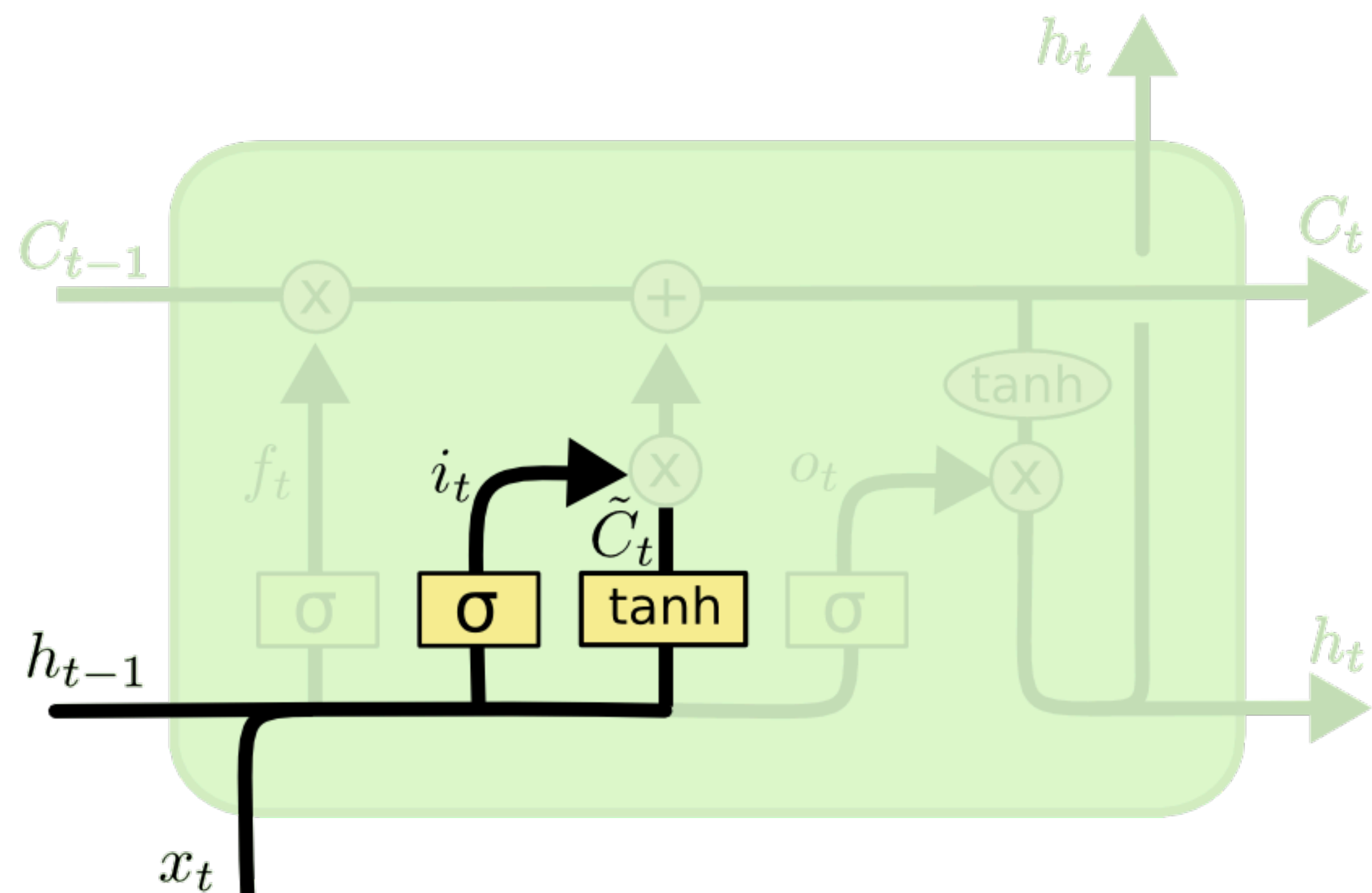
**$C_t$  = Cell state**



$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

Decide what information to throw away from the cell state.

Each element of cell state is multiplied by  $\sim 1$  (remember) or  $\sim 0$  (forget).



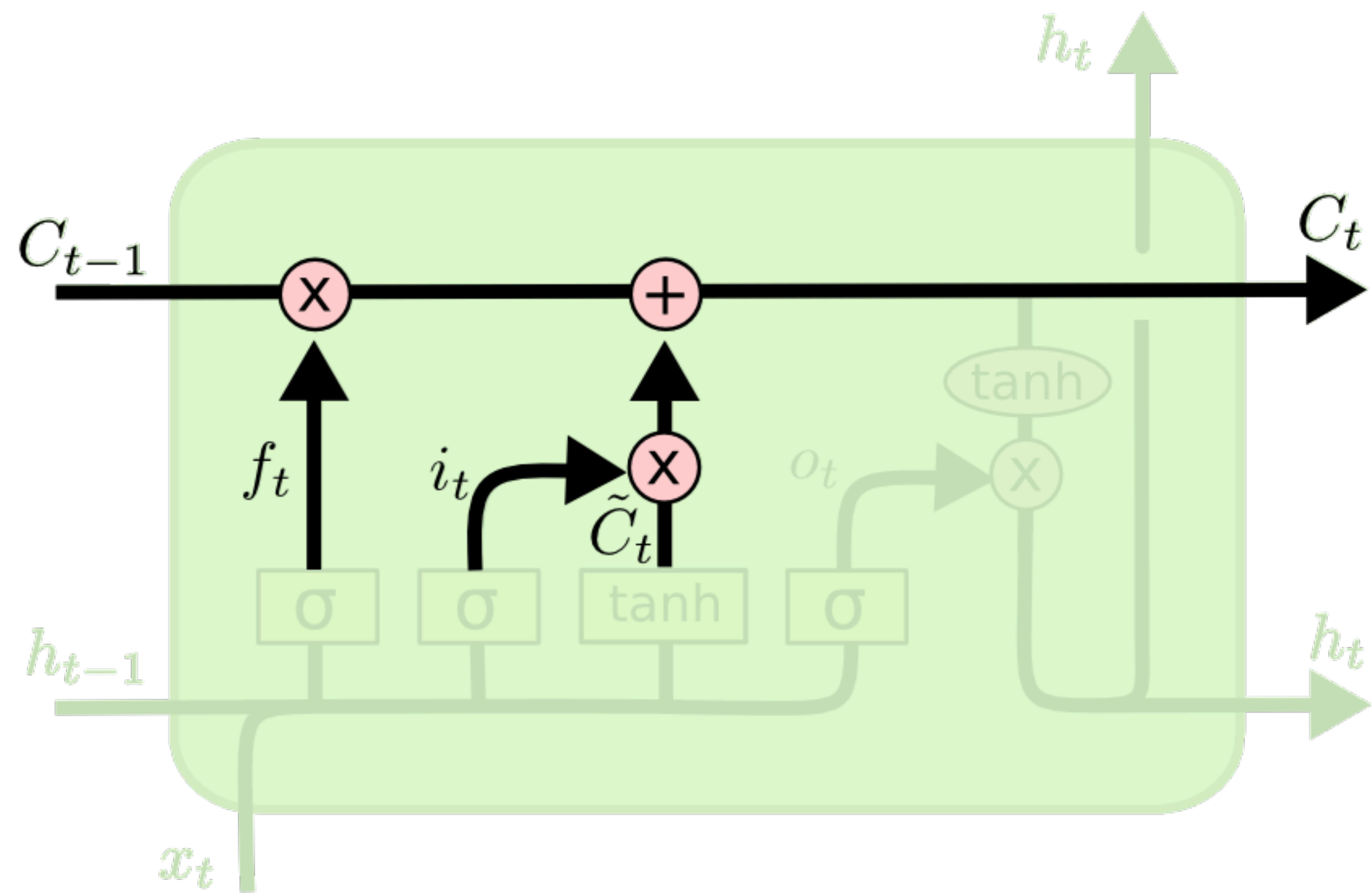
which indices to write to

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

what to write to those indices

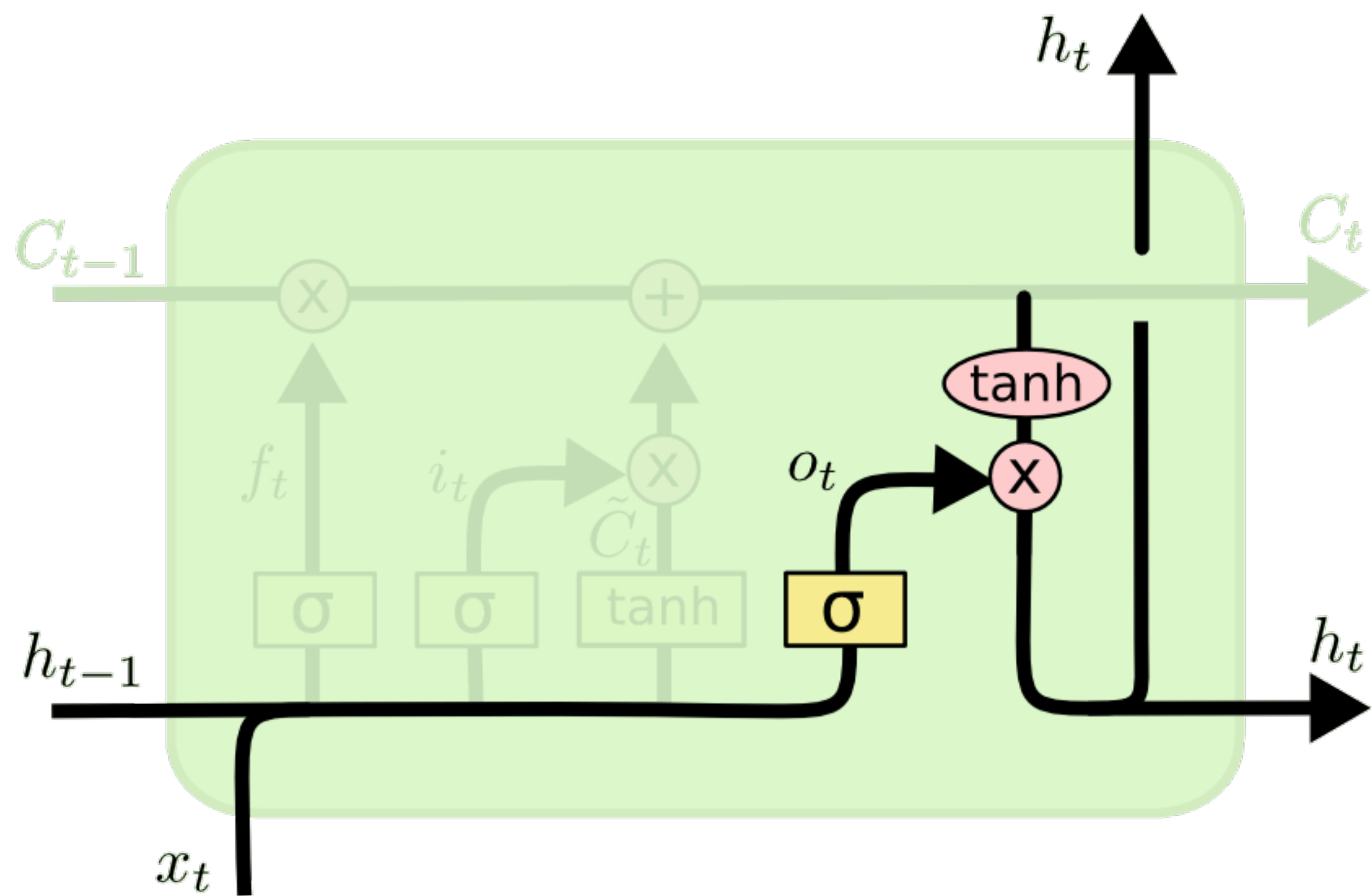
Decide what new information to add to the cell state.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Forget selected old information, write selected new information.



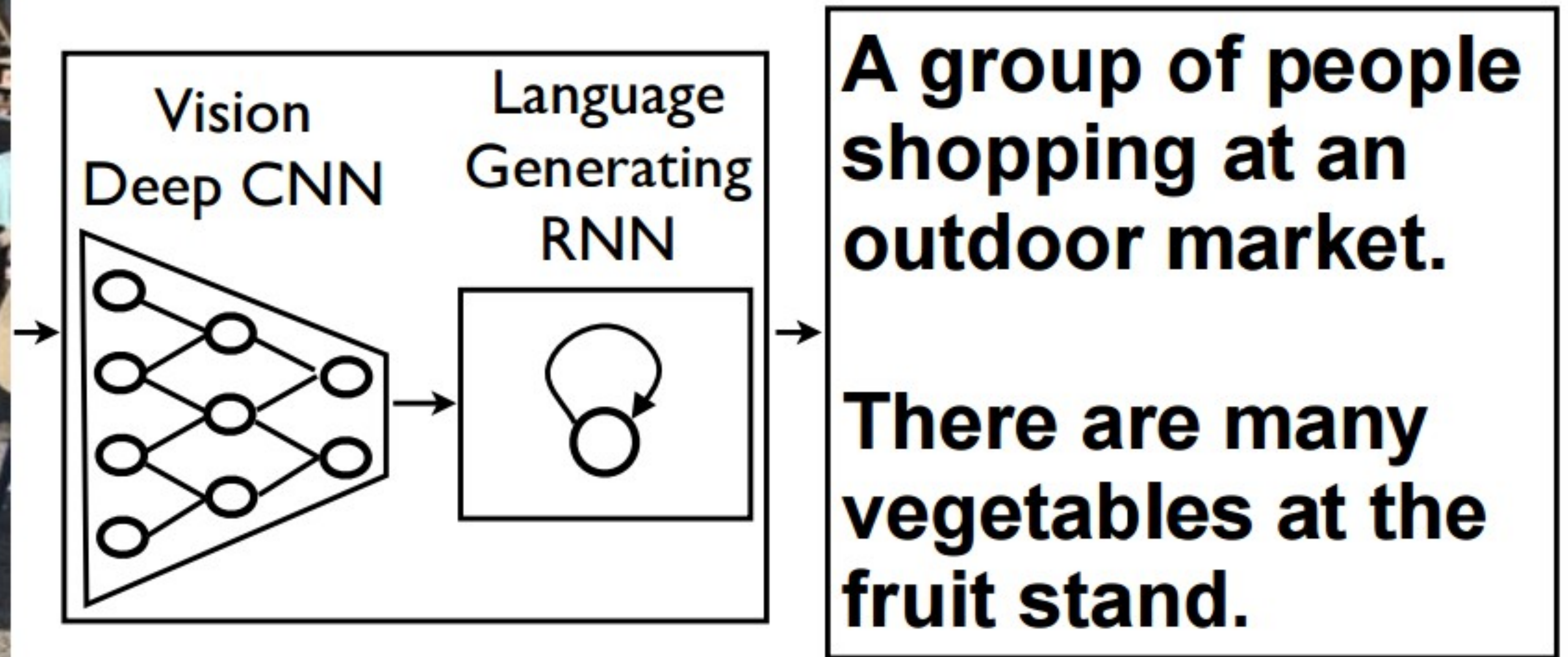


$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

After having updated the cell state's information, decide what to output.

# Image Captioning



# Recipe for deep learning in a new domain

1. Transform your data into numbers (e.g., a vector)
2. Transform your goal into a numerical measure (objective function)
3. #1 and #2 specify the “learning problem”
4. Use a generic optimizer (SGD) and an appropriate architecture (e.g., CNN or RNN) to solve the learning problem

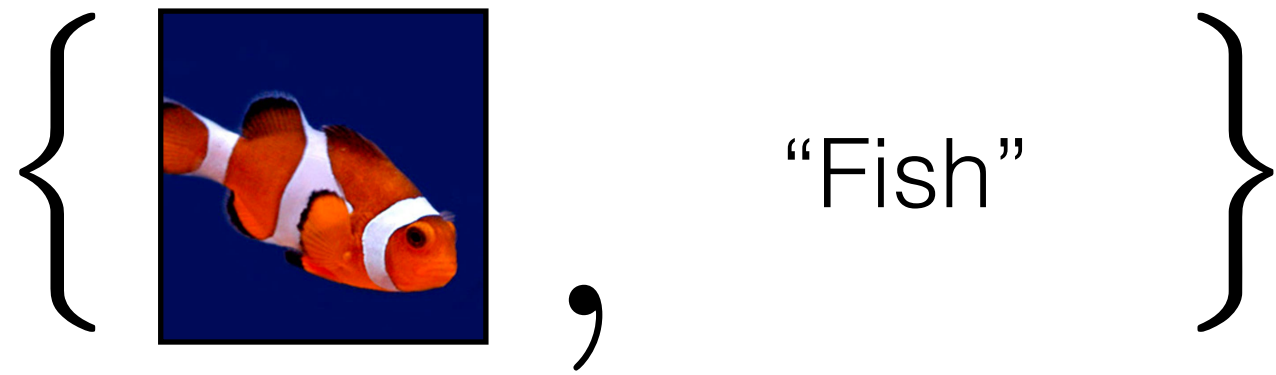
# How to represent words as numbers?

## One-hot vector

*Training data*

$x$

$y$



⋮

*Training data*

$x$

$y$

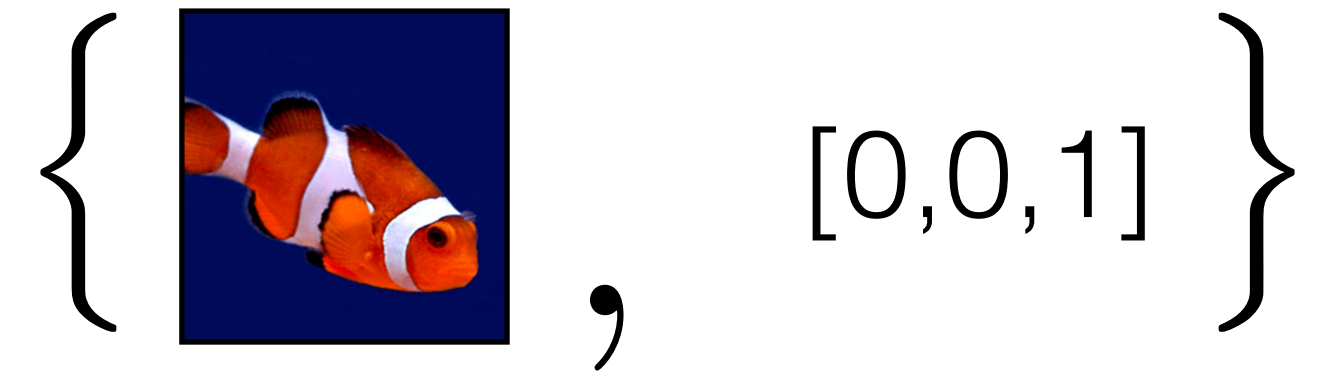


⋮

*Training data*

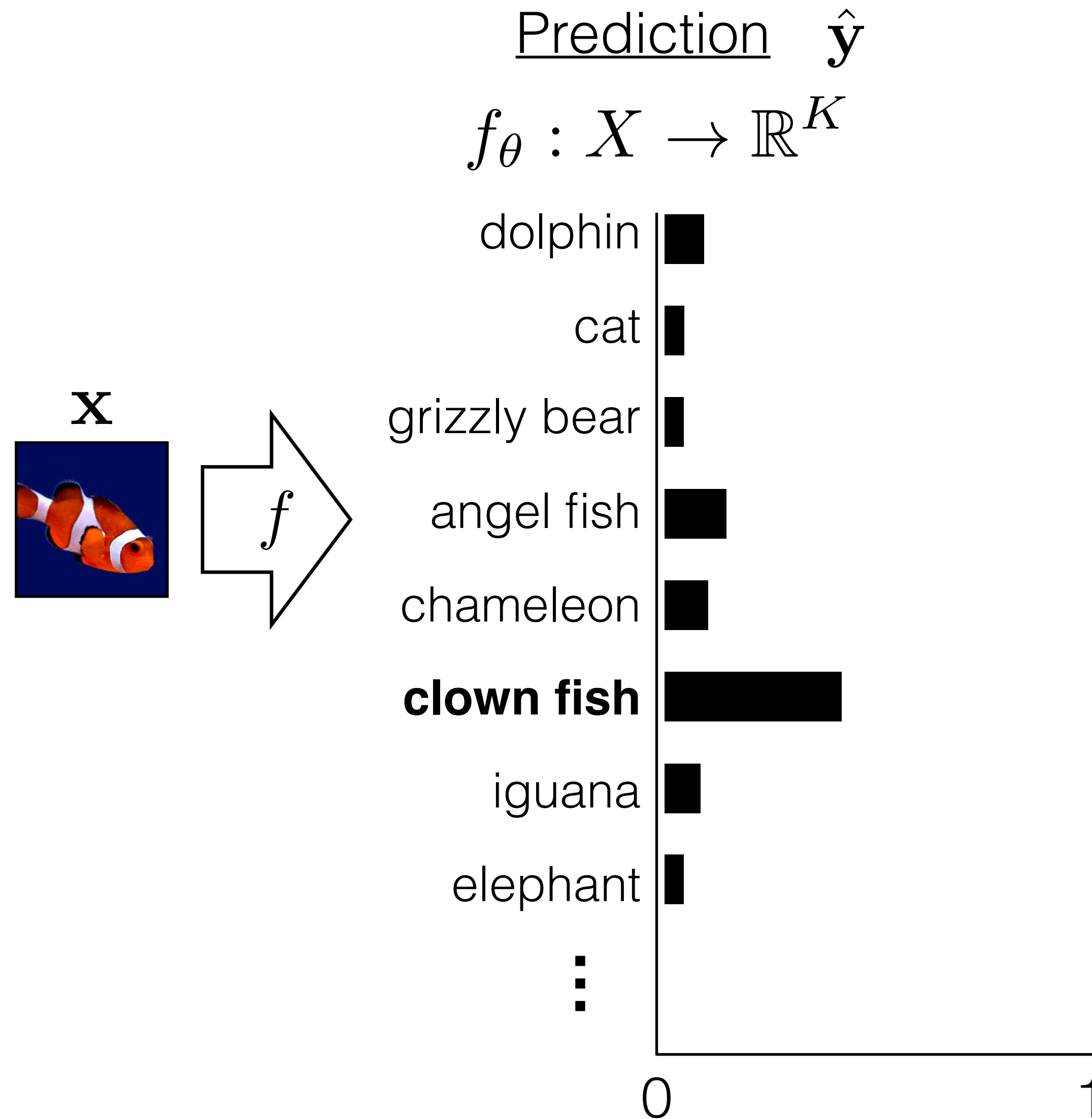
$x$

$y$

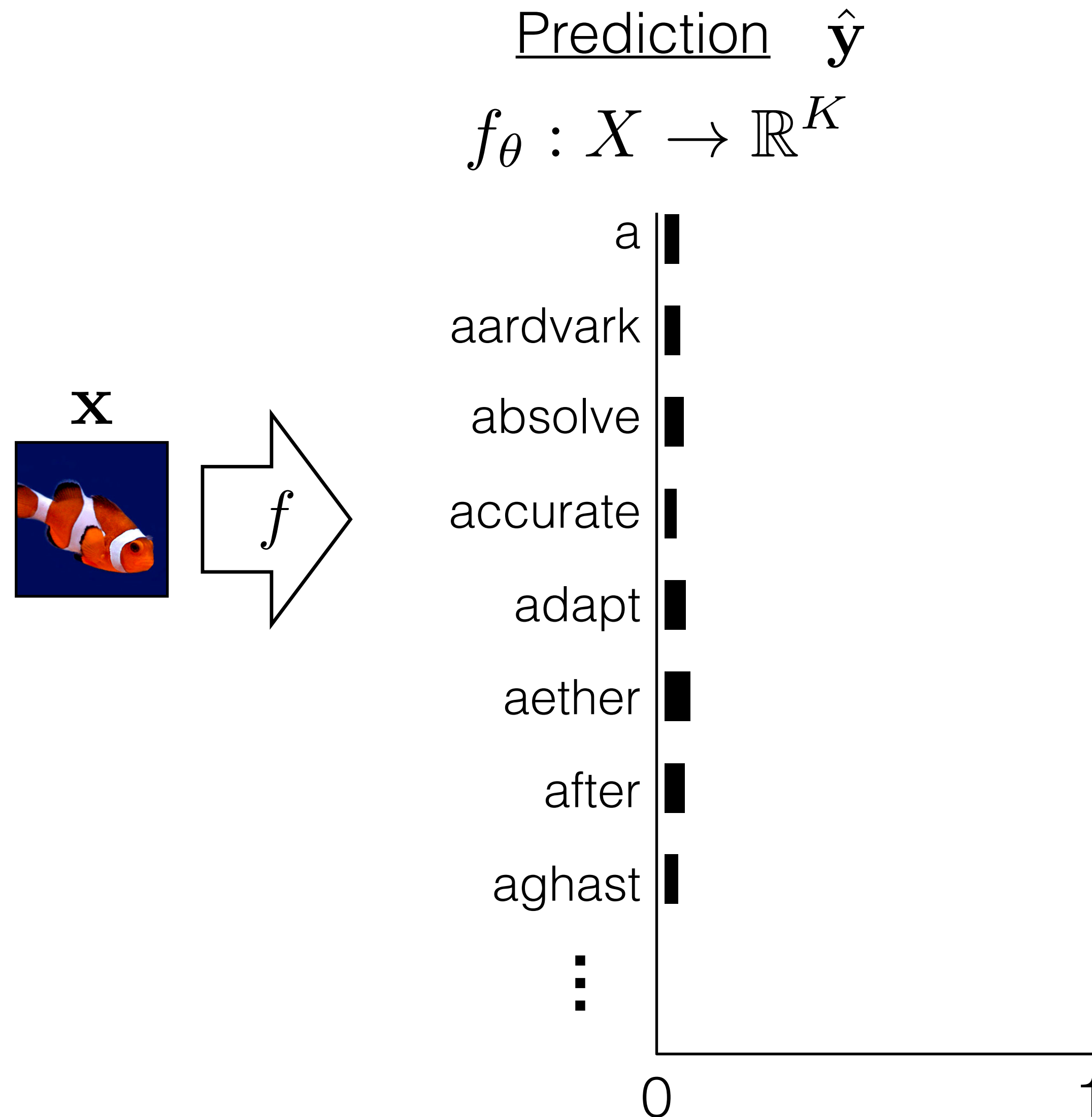


⋮

# How to represent words as numbers?



# How to represent words as numbers?

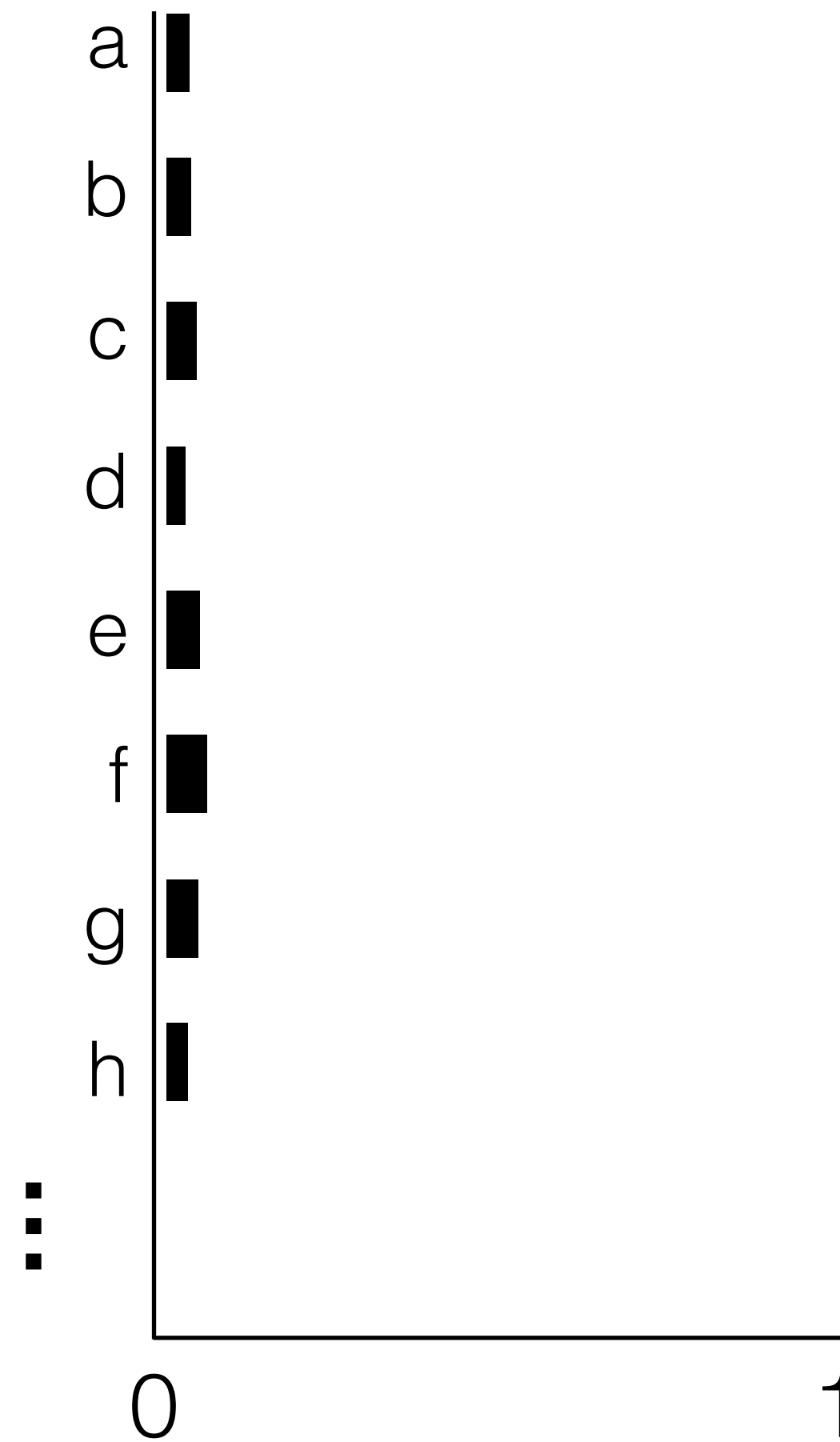
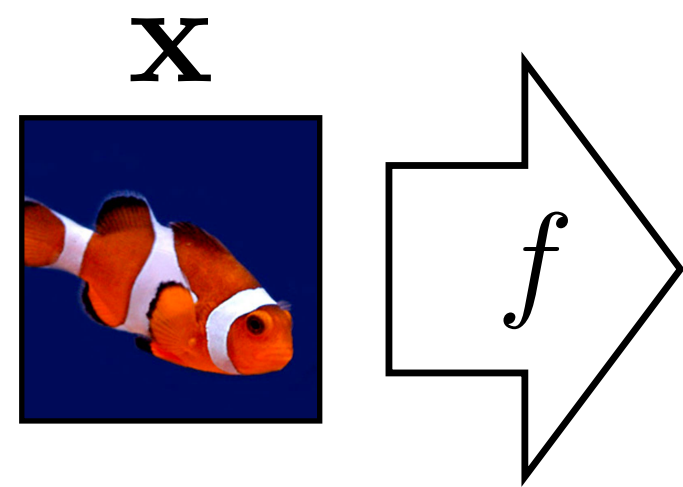


Rather than having just a handful of possible object classes, we can represent all words in a large vocabulary using a very large  $K$  (e.g.,  $K=100,000$ ).

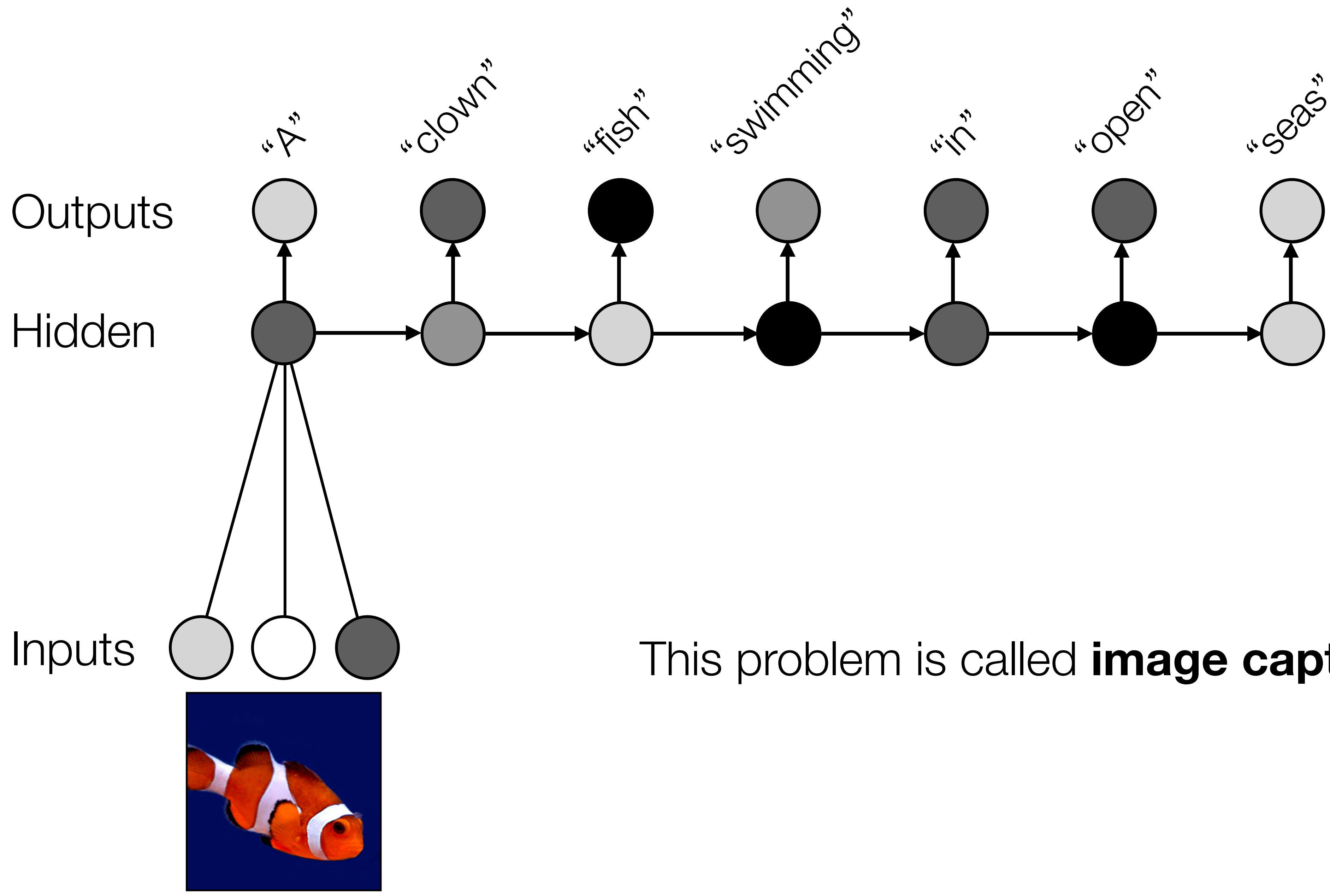
# How to represent words as numbers?

Prediction  $\hat{y}$

$$f_{\theta} : X \rightarrow \mathbb{R}^K$$



Or, represent each character as a class (e.g.,  $K=26$  for English letters), and represent words as a sequence of characters.



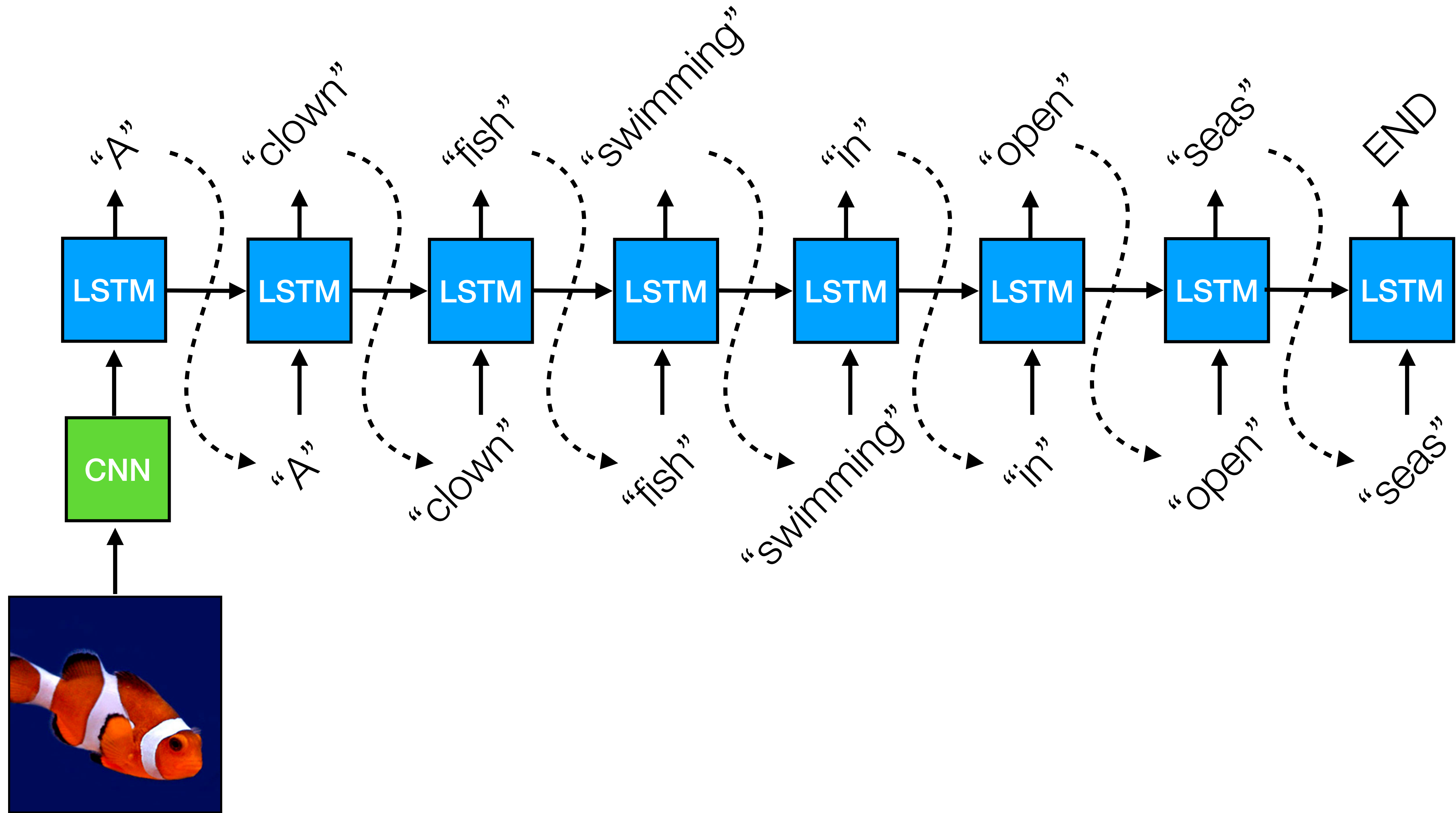
This problem is called **image captioning**



Outputs

Hidden

Input



# Training

Targets  $y$

“A”

“clown”

“fish”

“swimming”

“in”

“open”

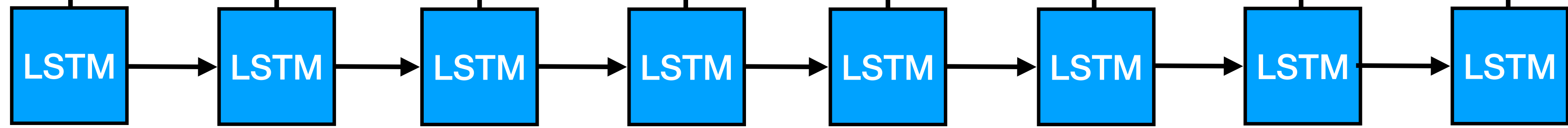
“seas”

END

Outputs  $p_{\theta}(\cdot)$



Hidden



“A”

“clown”

“fish”

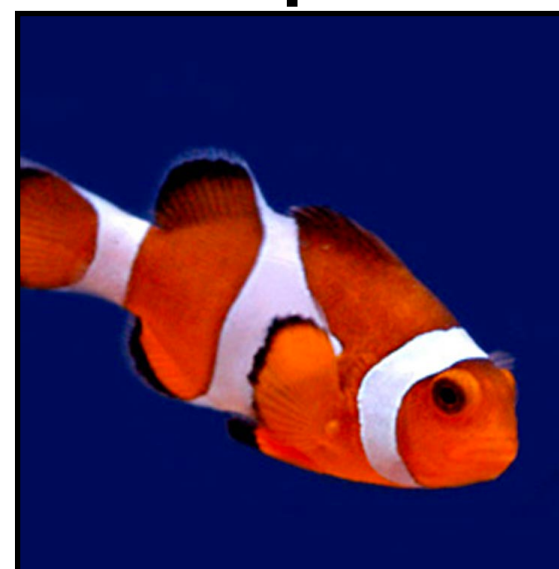
“swimming”

“in”

“open”

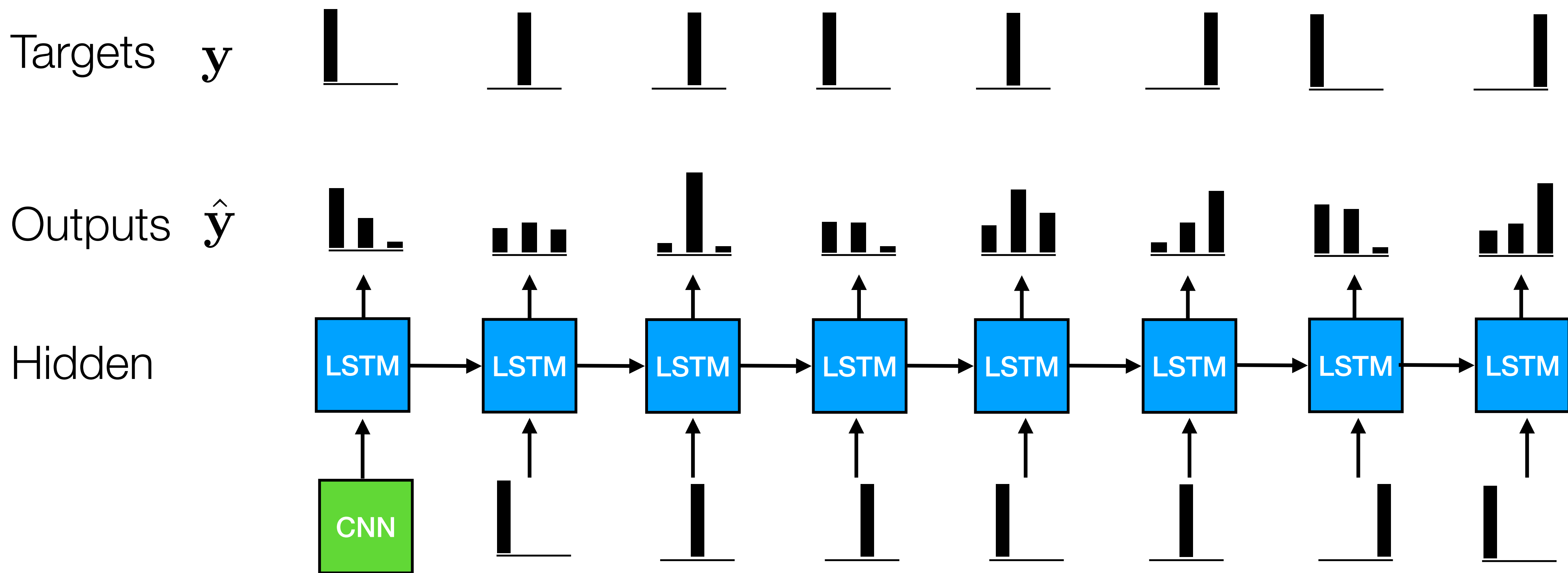
“seas”

Input

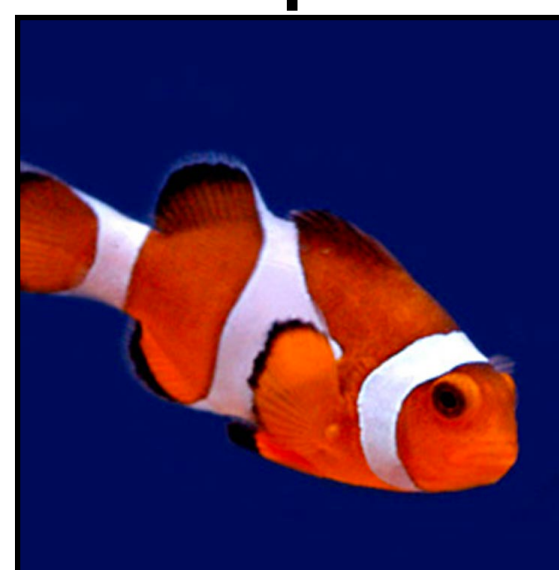


**Max-likelihood objective:** maximize probability the model assigns to each target word:  $\arg \max_{\theta} \log p_{\theta}(y)$

# Training



Input



**Max-likelihood objective:**  
minimize cross-entropy between  
model outputs and one-hot  
encoded targets.

$$f^* = \arg \min_{f \in \mathcal{F}} \sum_{i=1}^N H(\mathbf{y}_i, \hat{\mathbf{y}}_i)$$

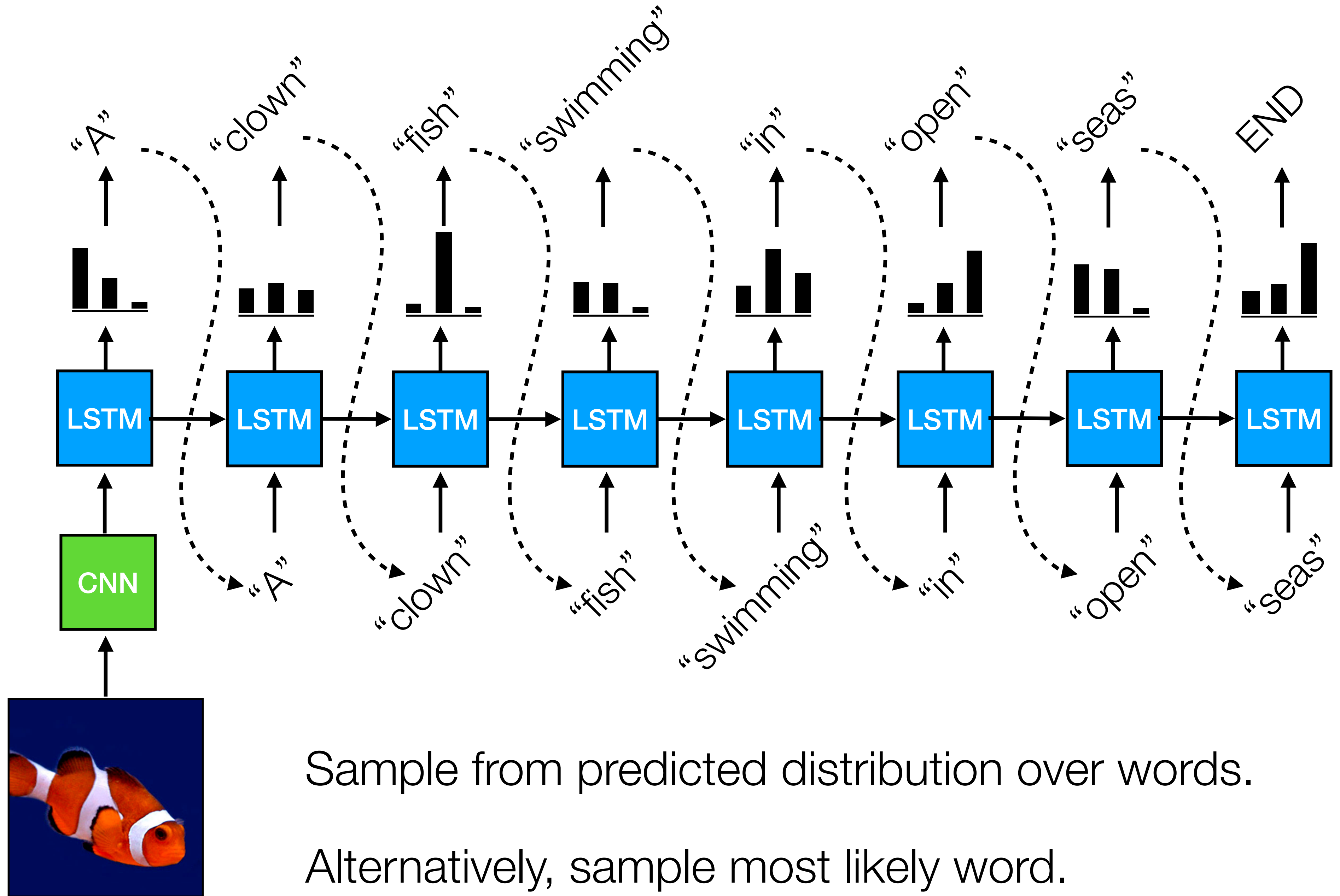
# Testing

Samples

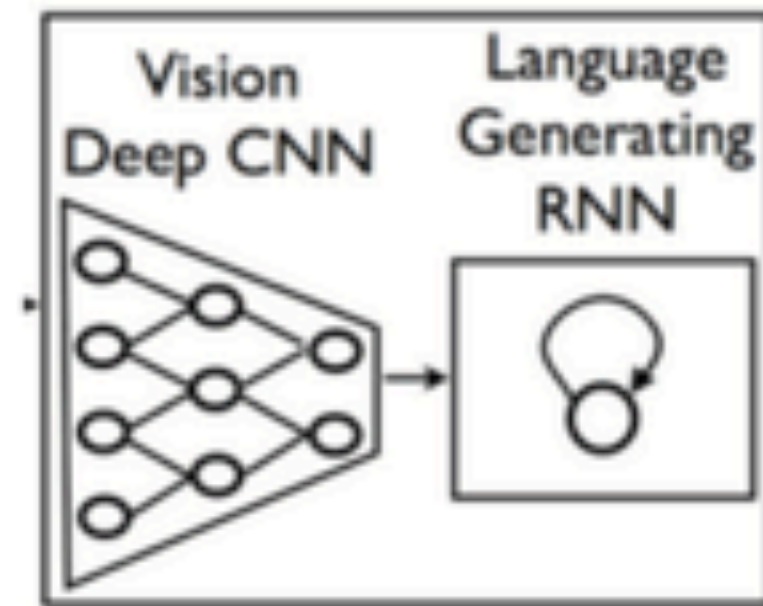
Outputs  $p_{\theta}(\cdot)$

Hidden

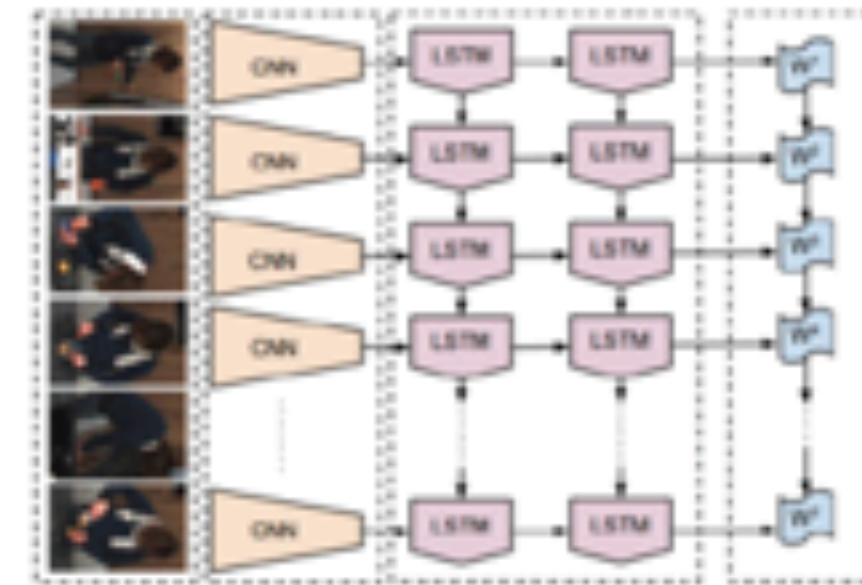
Input



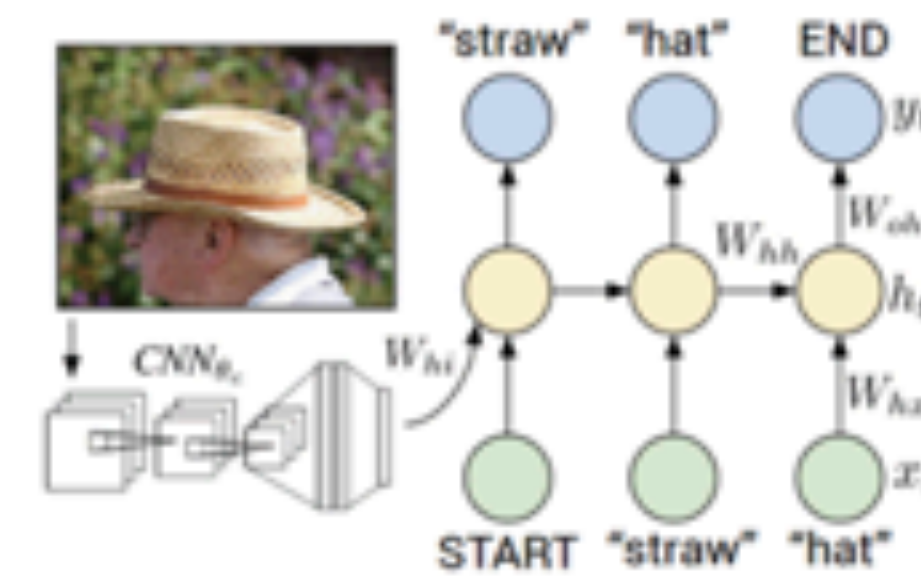
# It was very popular a few years ago



Vinyals et al., 2015



Donahue et al., 2015



Karpathy and Fei-Fei, 2015



Hodosh et al., 2013



Fang et al., 2015



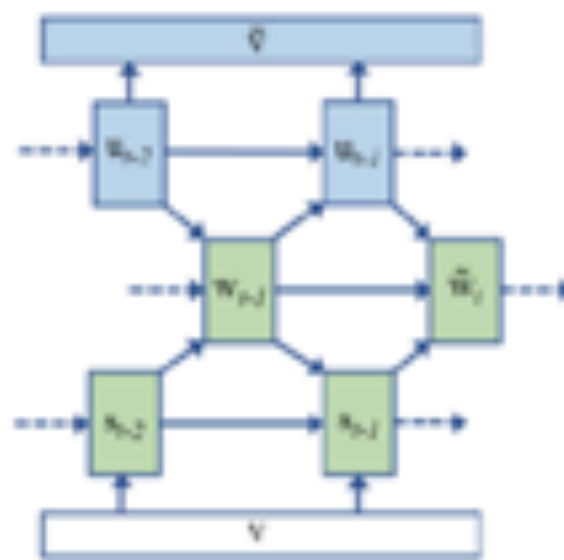
Mao et al., 2015



Ordonez et al., 2011



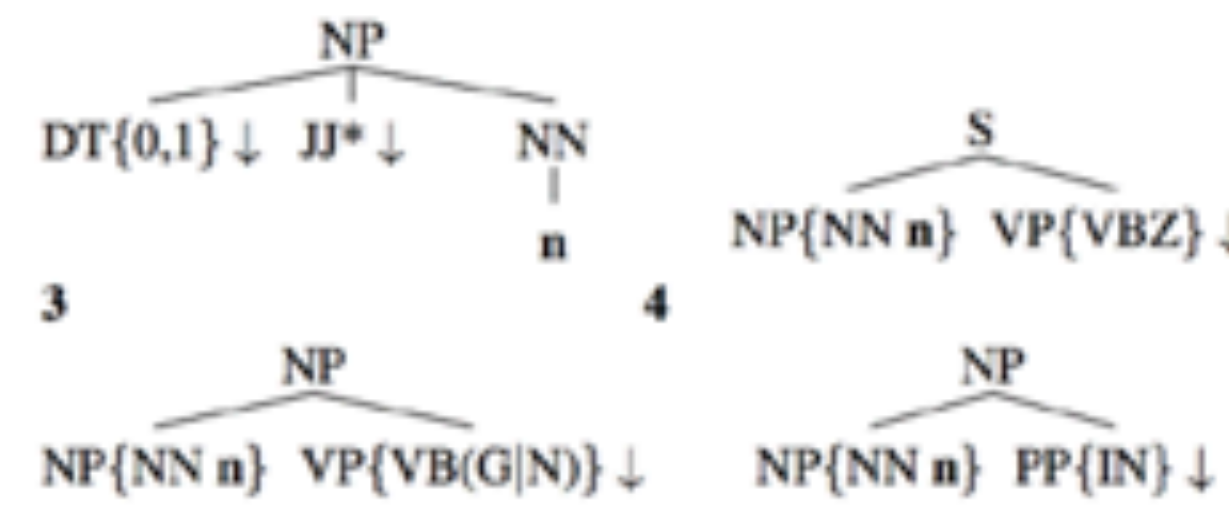
Kulkarni et al., 2011



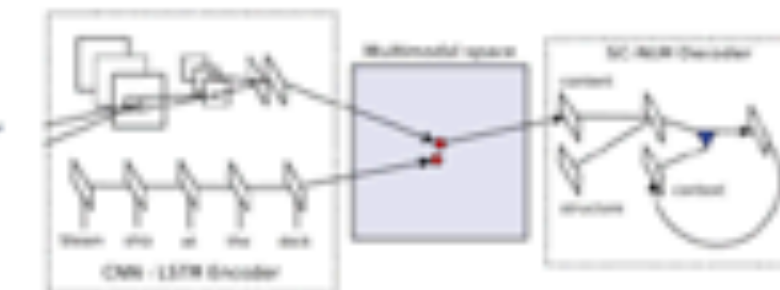
Chen and Zitnick, 2015



Farhadi et al., 2010



Mitchell et al., 2012



Kiros et al., 2015

**A person riding a motorcycle on a dirt road.**



**Two dogs play in the grass.**



**A skateboarder does a trick on a ramp.**



**A dog is jumping to catch a frisbee.**



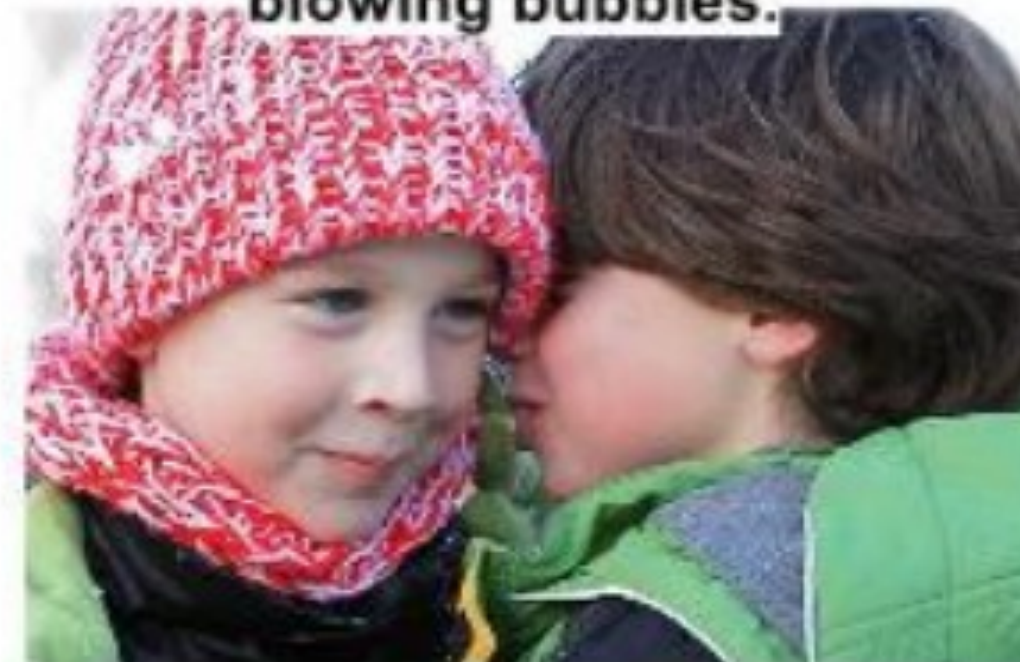
**A group of young people playing a game of frisbee.**



**Two hockey players are fighting over the puck.**



**A little girl in a pink hat is blowing bubbles.**



**A refrigerator filled with lots of food and drinks.**



**A herd of elephants walking across a dry grass field.**



**A close up of a cat laying on a couch.**



**A red motorcycle parked on the side of the road.**



**A yellow school bus parked in a parking lot.**



**Describes without errors**

**Describes with minor errors**

**Somewhat related to the image**

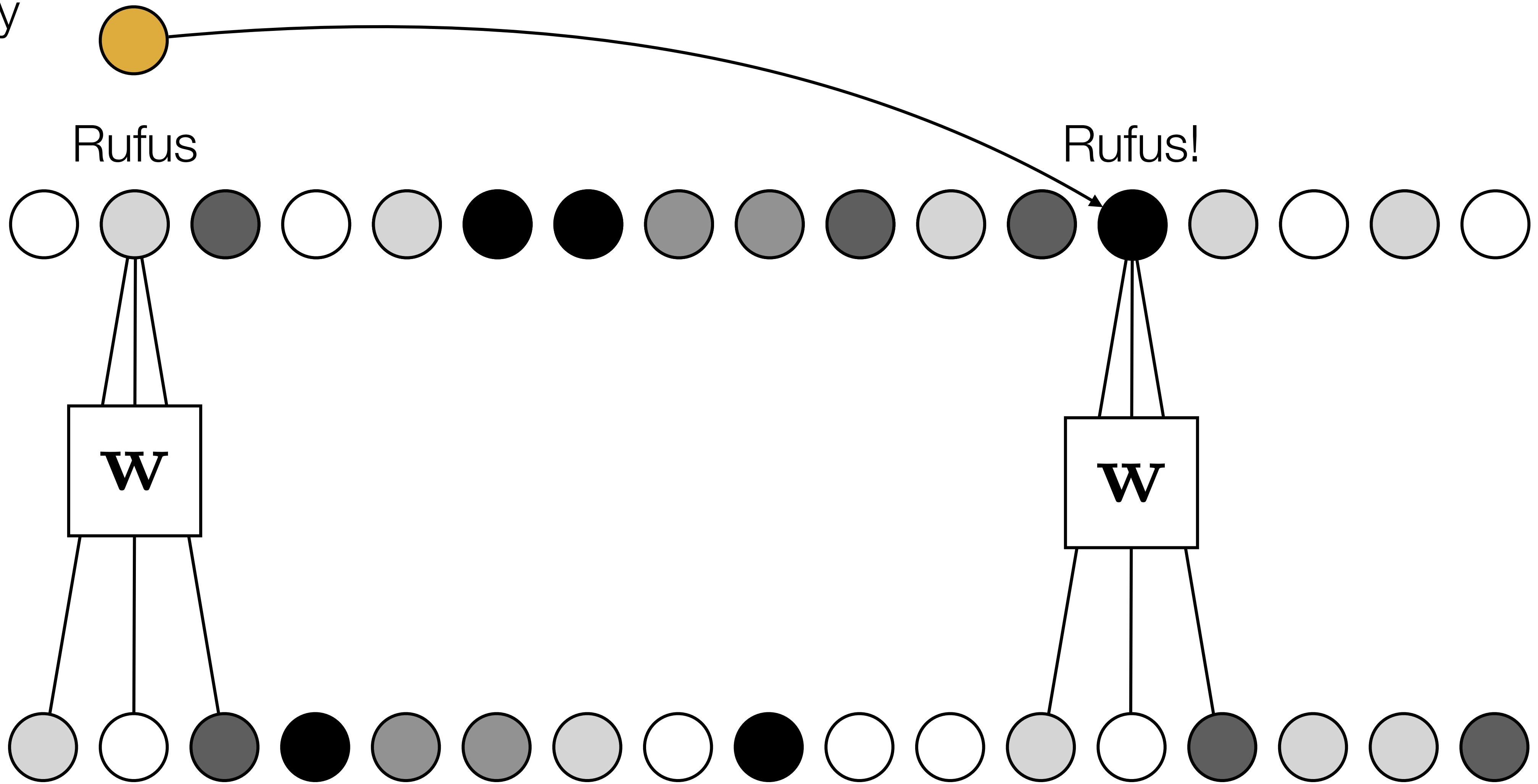
**Unrelated to the image**

# The problem of long-range dependences

Why not remember everything?

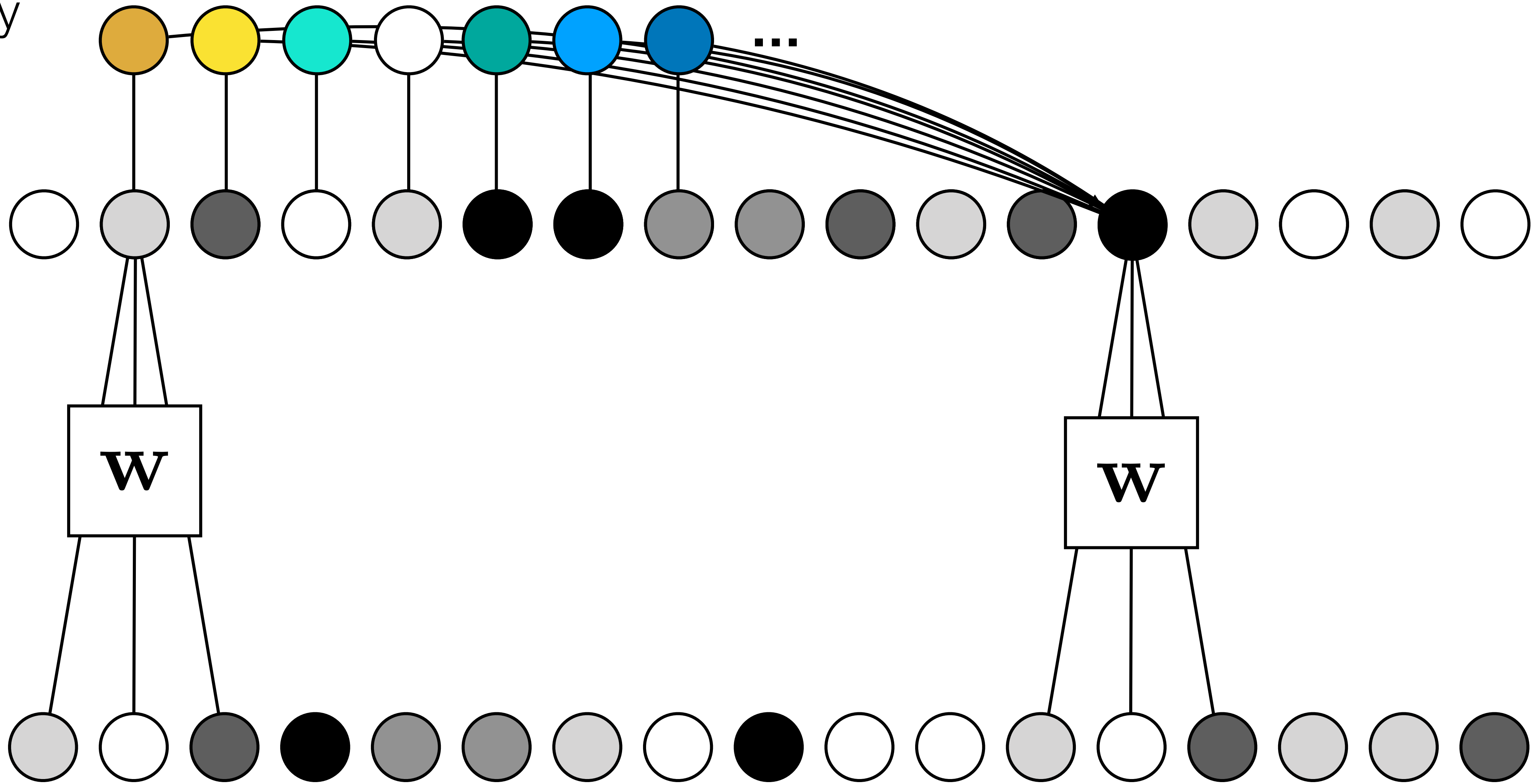
- Memory size grows with  $t$
- This kind of memory is **nonparametric**: there is no finite set of parameters we can use to model it
- RNNs make a Markov assumption — the future hidden state only depends on the immediately preceding hidden state
- By putting the right info in to the hidden state, RNNs can model dependences that are arbitrarily far apart

Memory unit





Memory units



time



# The problem of long-range dependences

Other methods exist that do directly link old “memories” (observations or hidden states) to future predictions:

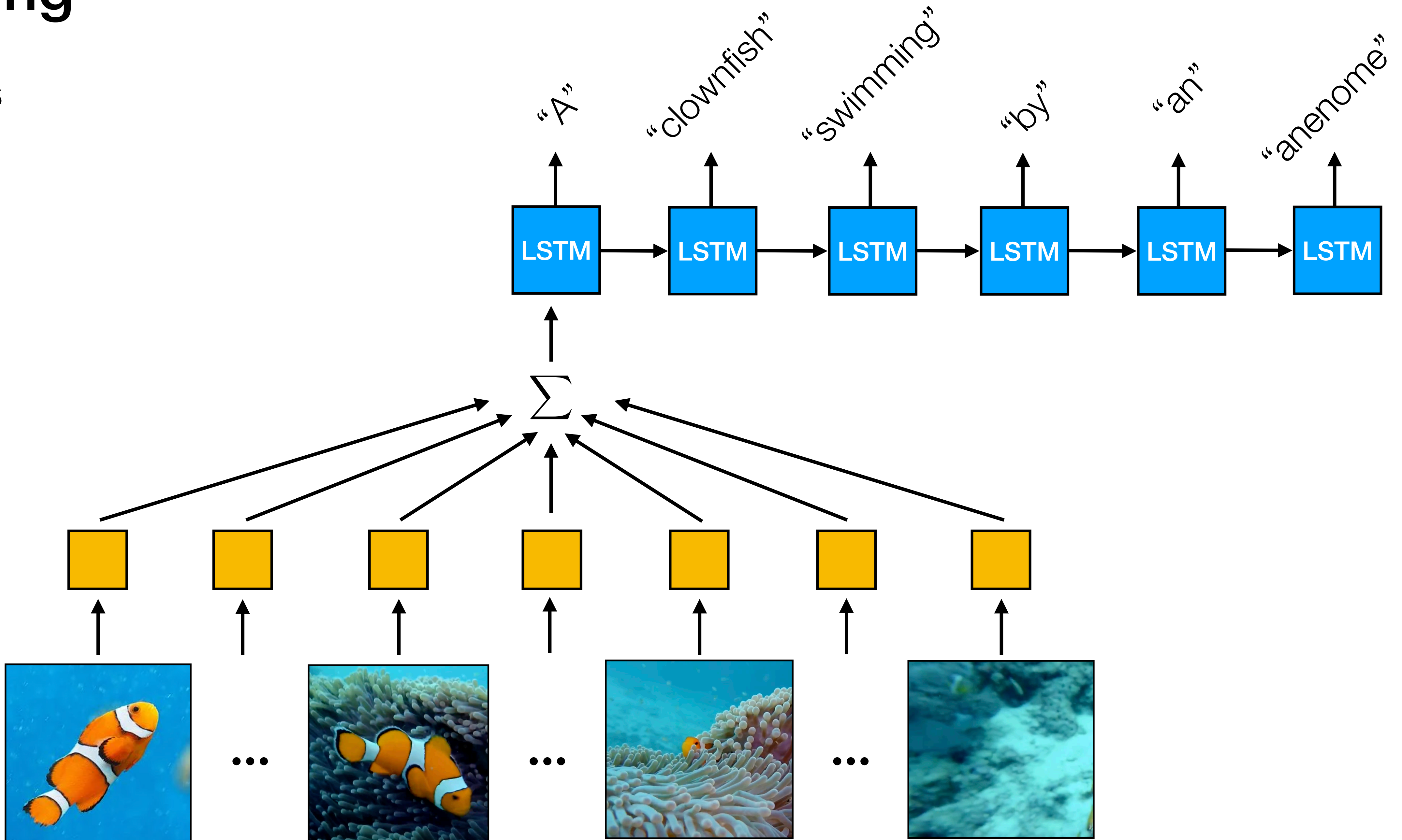
- Temporal convolutions
- Attention / Transformers (see <https://arxiv.org/abs/1706.03762>)
- Memory networks (see <https://arxiv.org/abs/1410.3916>)

# Pooling

Outputs

Hidden

Input

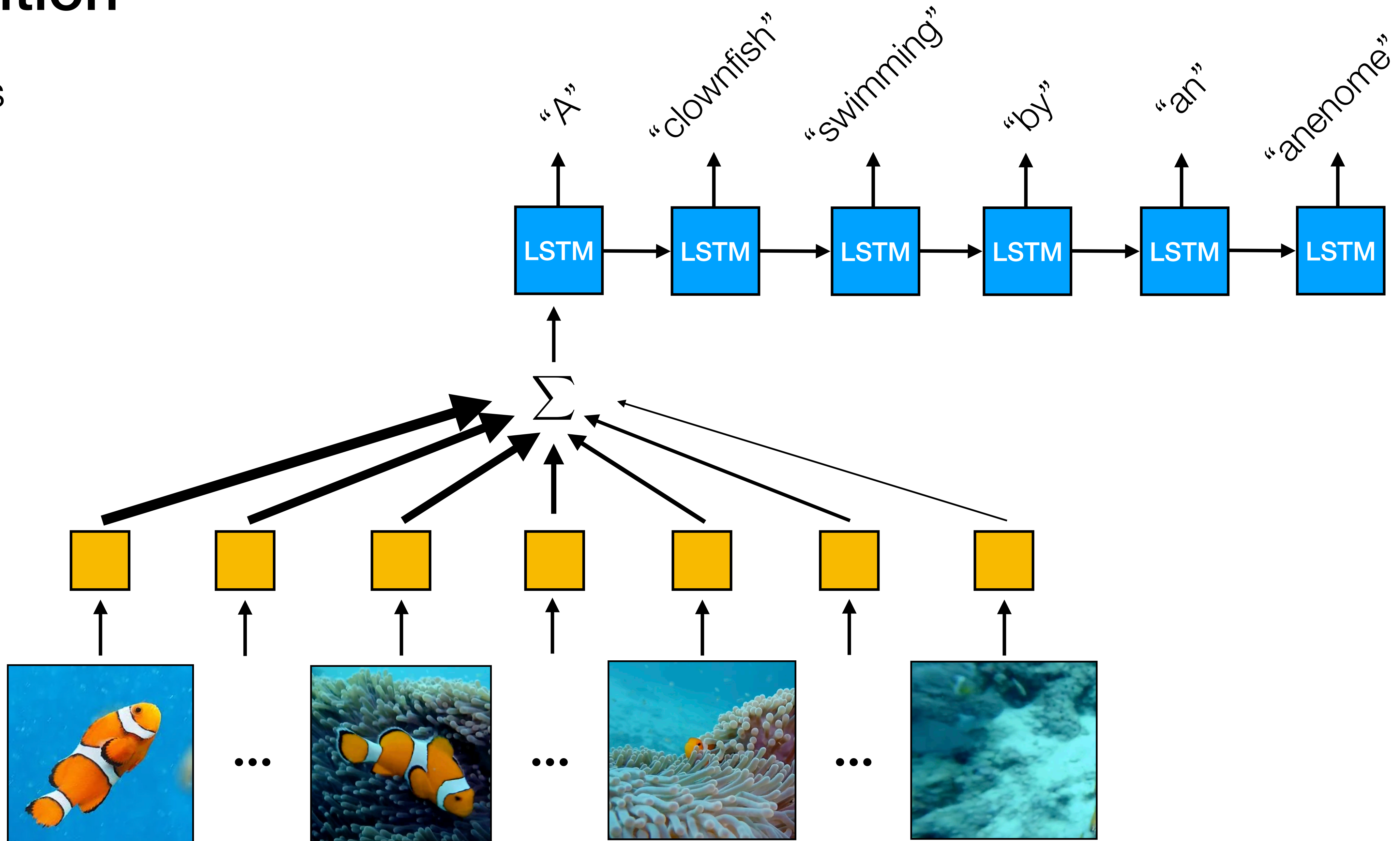


# Attention

Outputs

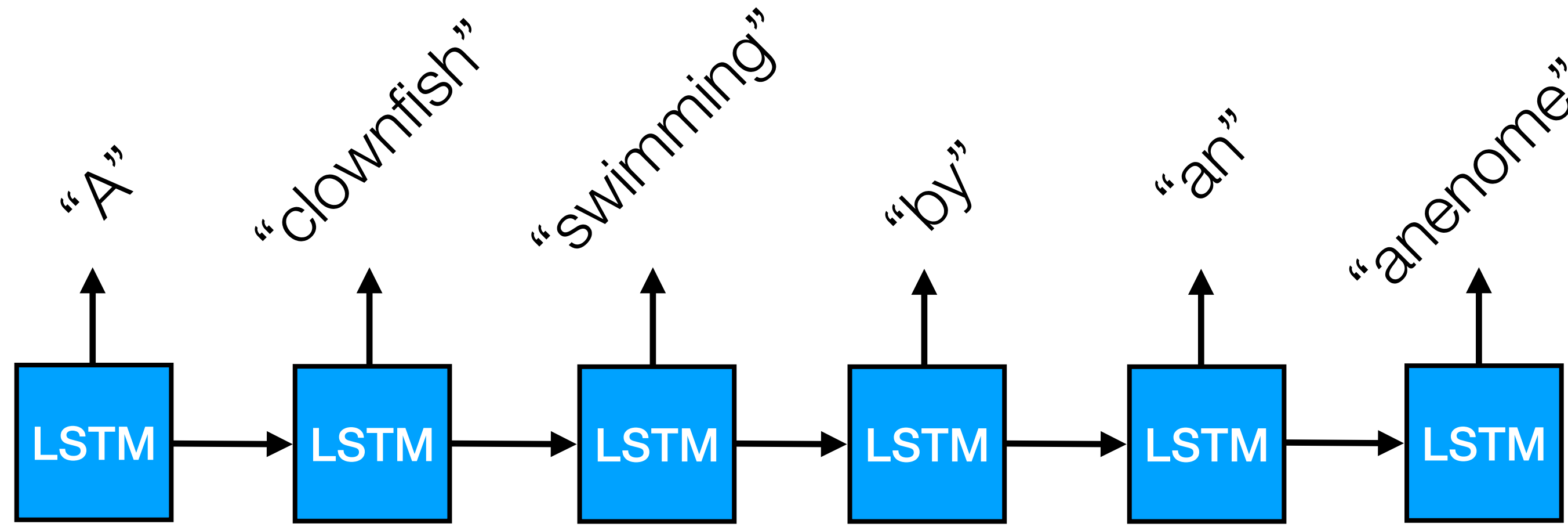
Hidden

Input

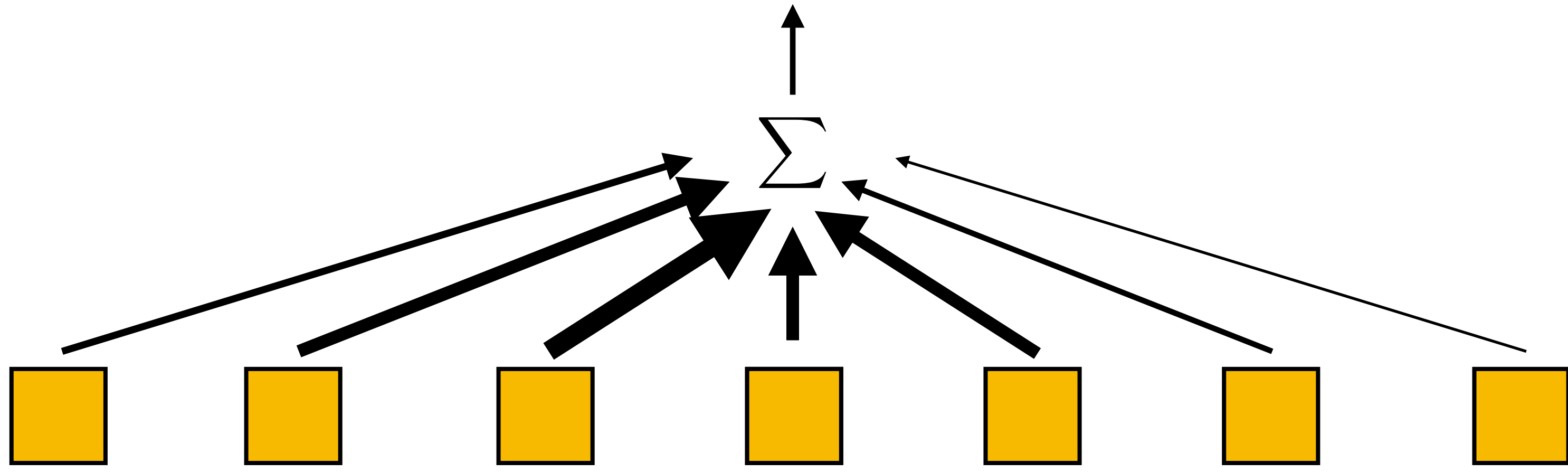


# Attention

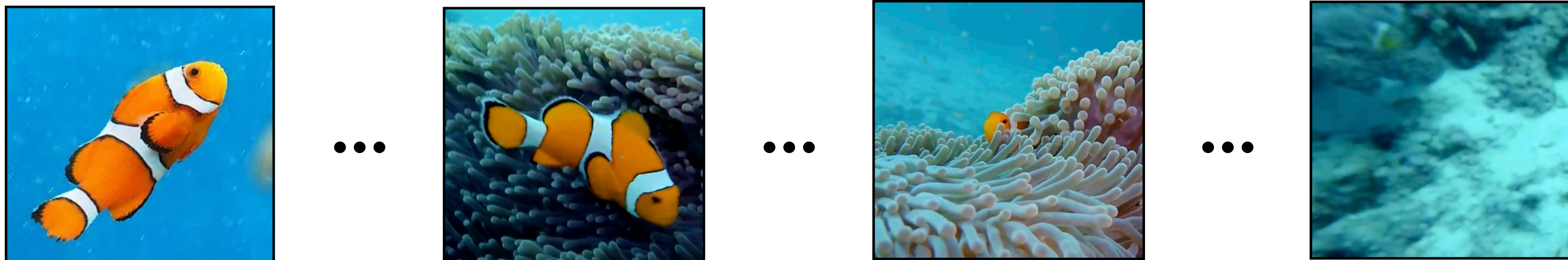
Outputs



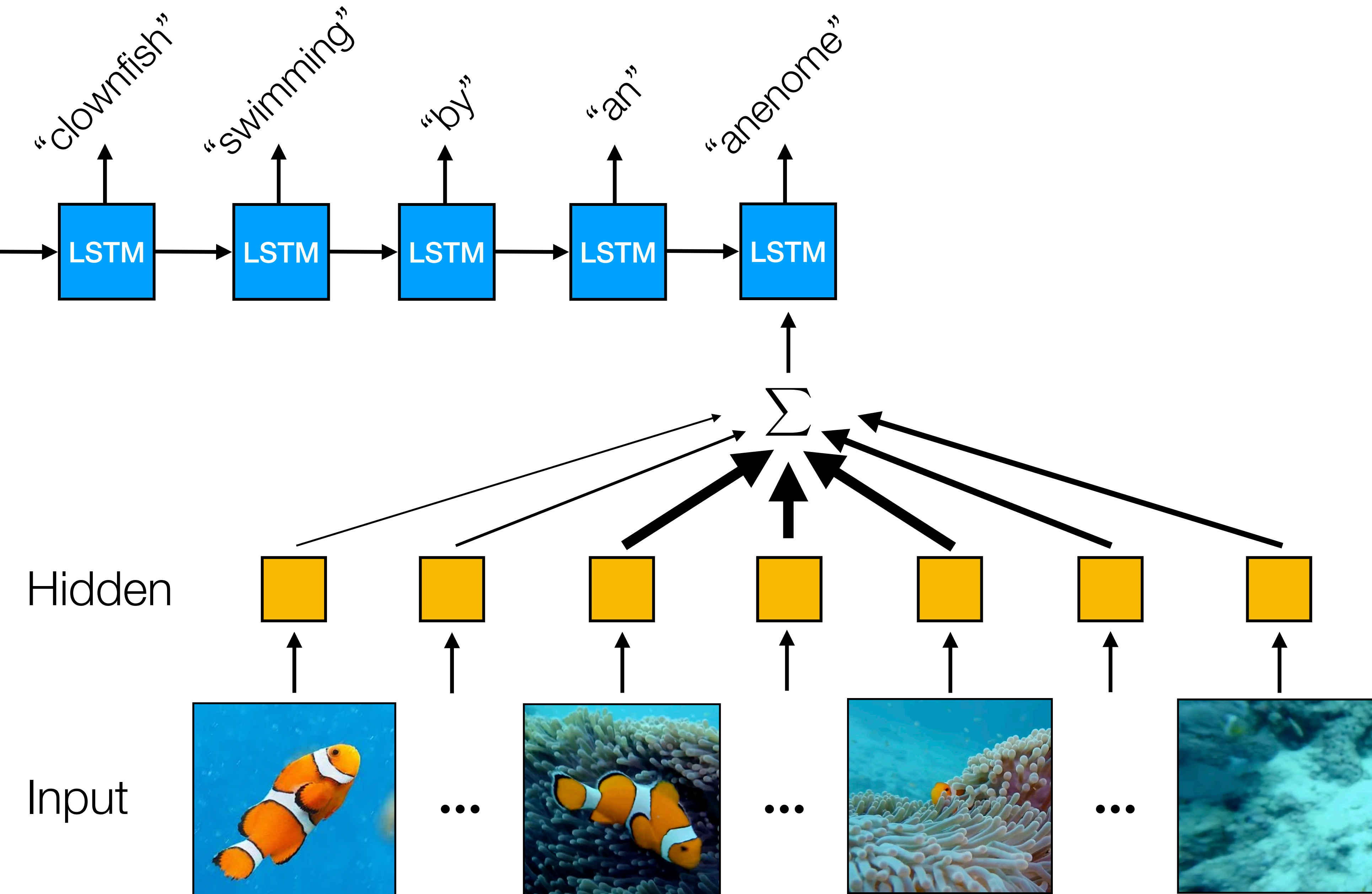
Hidden



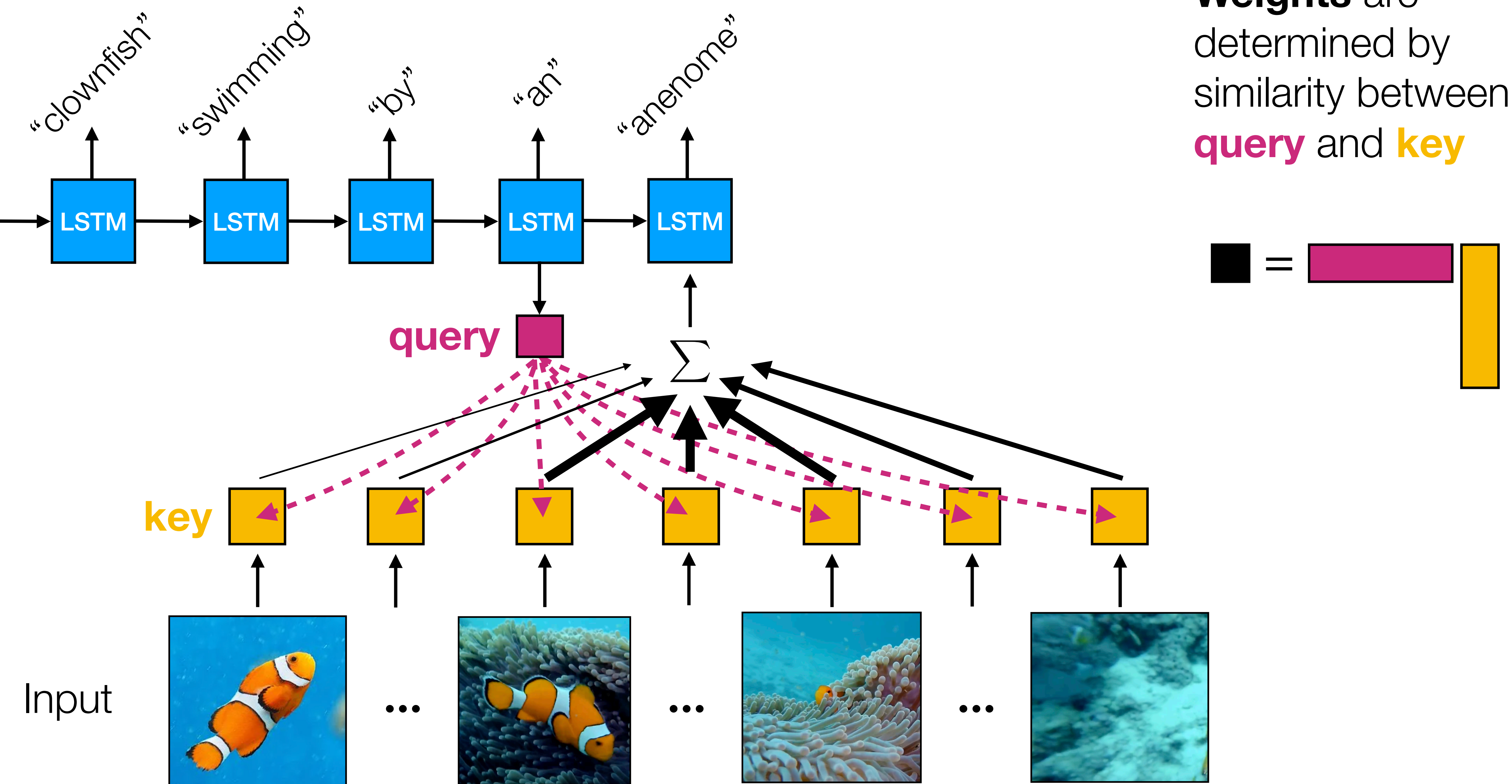
Input



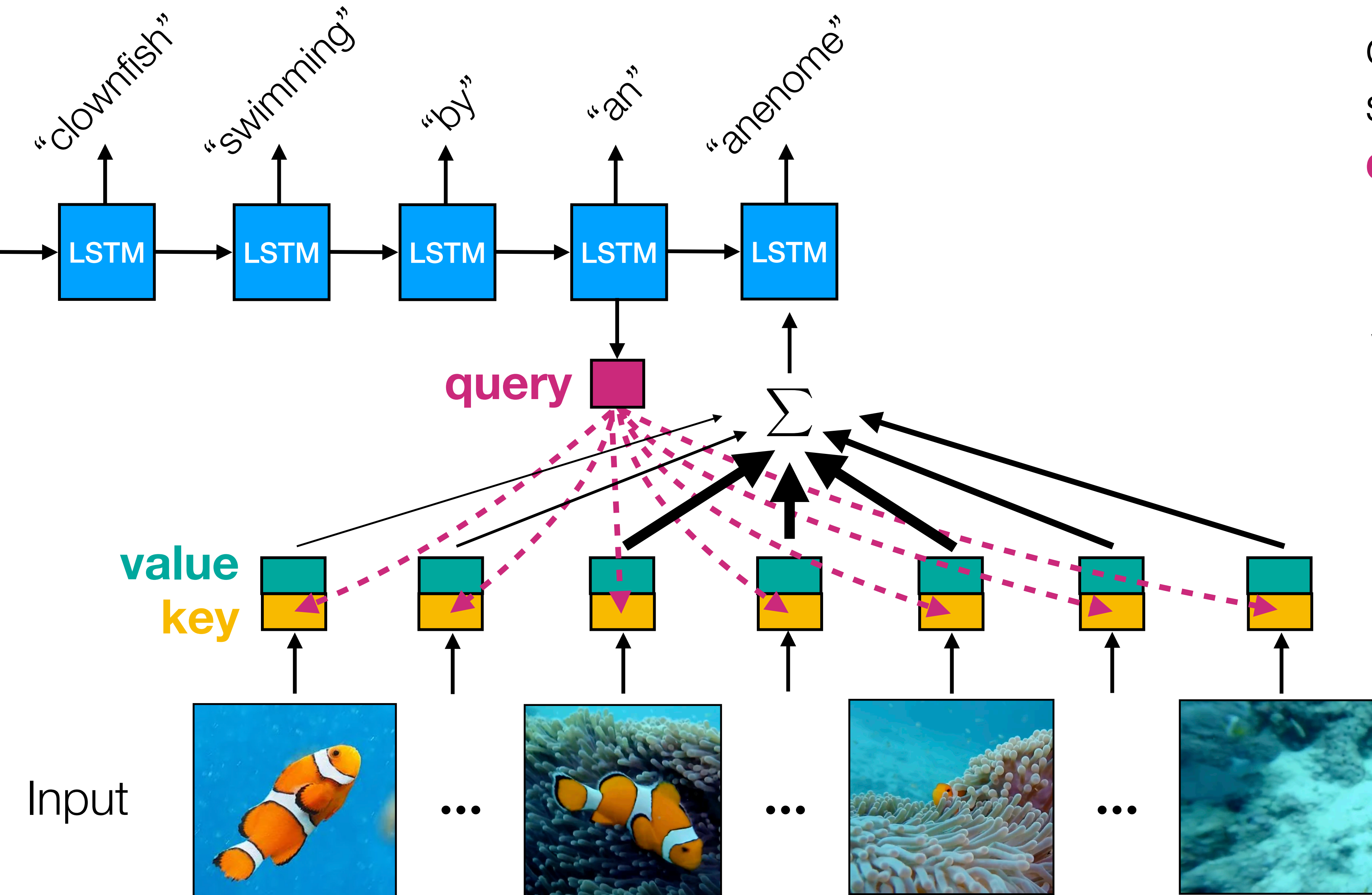
# Attention



# Attention



# Attention



**Weights** are determined by similarity between **query** and **key**

summation is over **weights** \* **value**



# Problems with Recurrent Architectures

- Long sequences still tricky, exploding / vanishing gradients
- Inherently sequential: Need to generate word after word...
- Regularly *do not* remember long-ago events.

---

## Attention Is All You Need

---

**Ashish Vaswani\***  
Google Brain  
avaswani@google.com

**Noam Shazeer\***  
Google Brain  
noam@google.com

**Niki Parmar\***  
Google Research  
nikip@google.com

**Jakob Uszkoreit\***  
Google Research  
usz@google.com

**Llion Jones\***  
Google Research  
llion@google.com

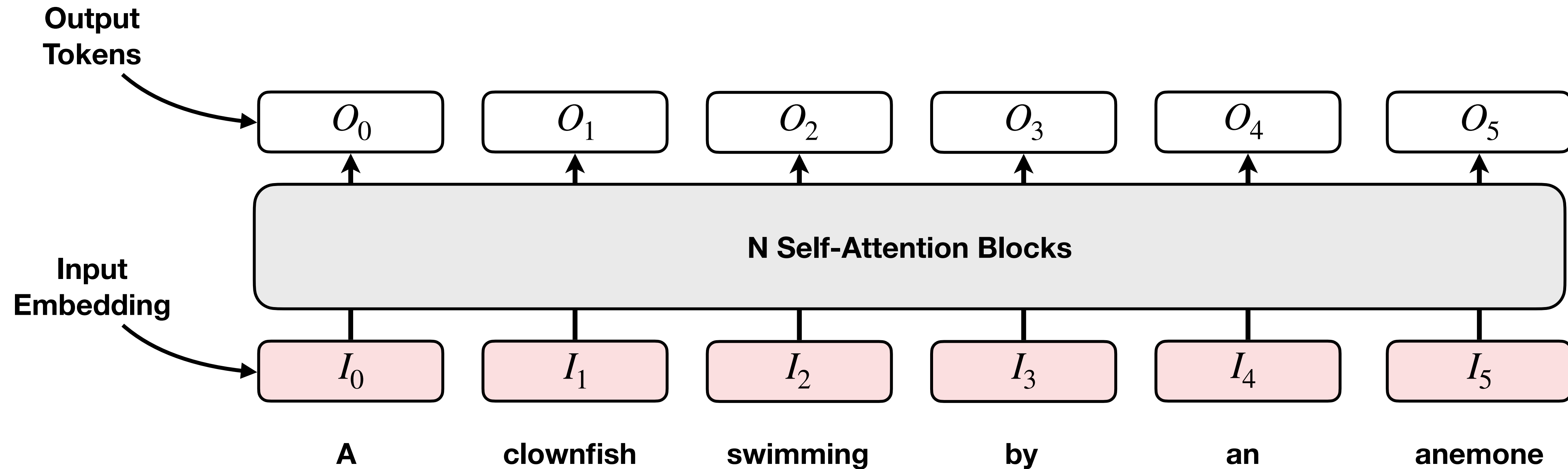
**Aidan N. Gomez\* †**  
University of Toronto  
aidan@cs.toronto.edu

**Lukasz Kaiser\***  
Google Brain  
lukaszkaizer@google.com

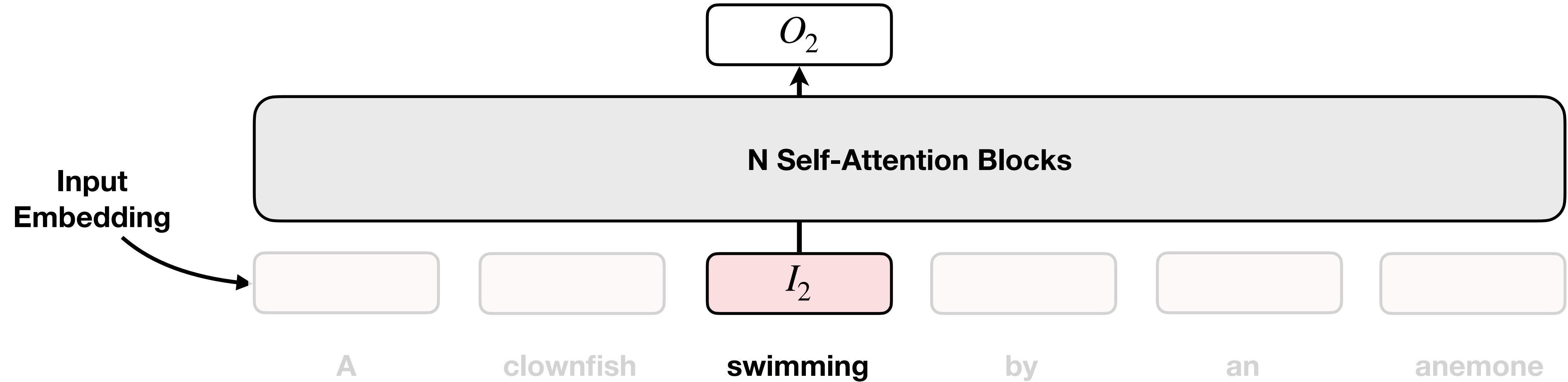
**Illia Polosukhin\* ‡**  
illia.polosukhin@gmail.com

6 Dec 2017

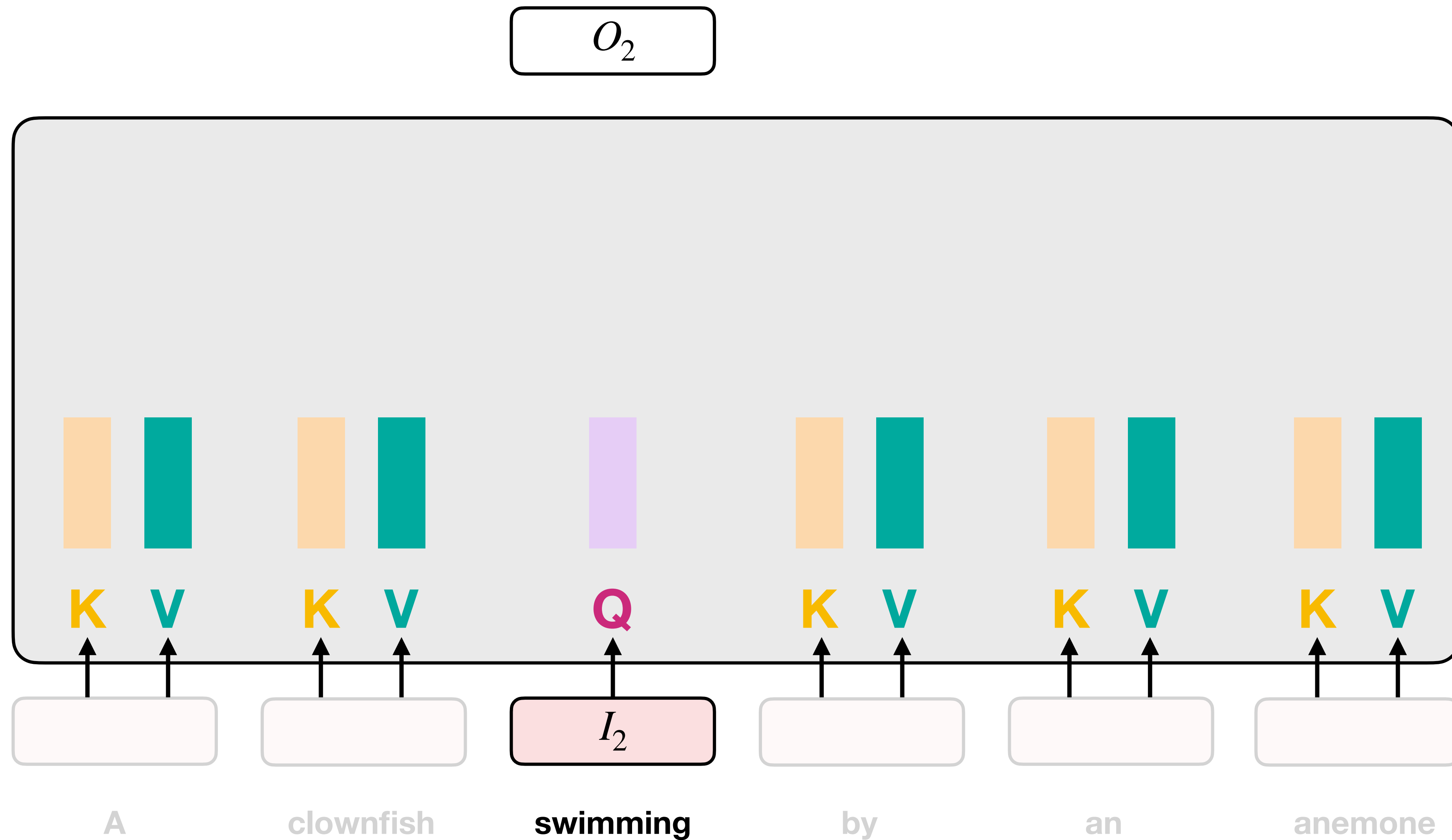
# Encoding Sequences with Attention



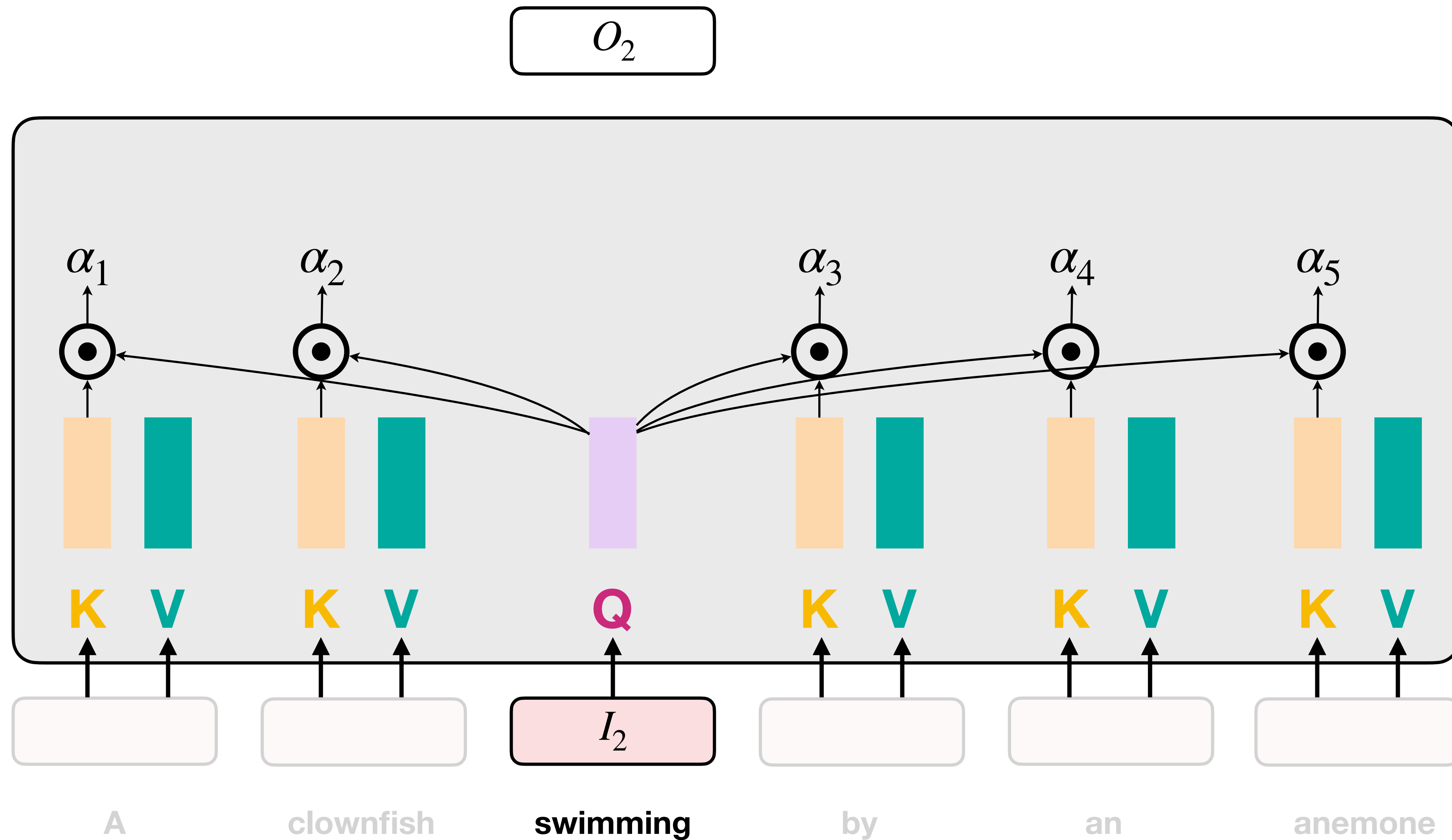
# Encoding Sequences with Attention



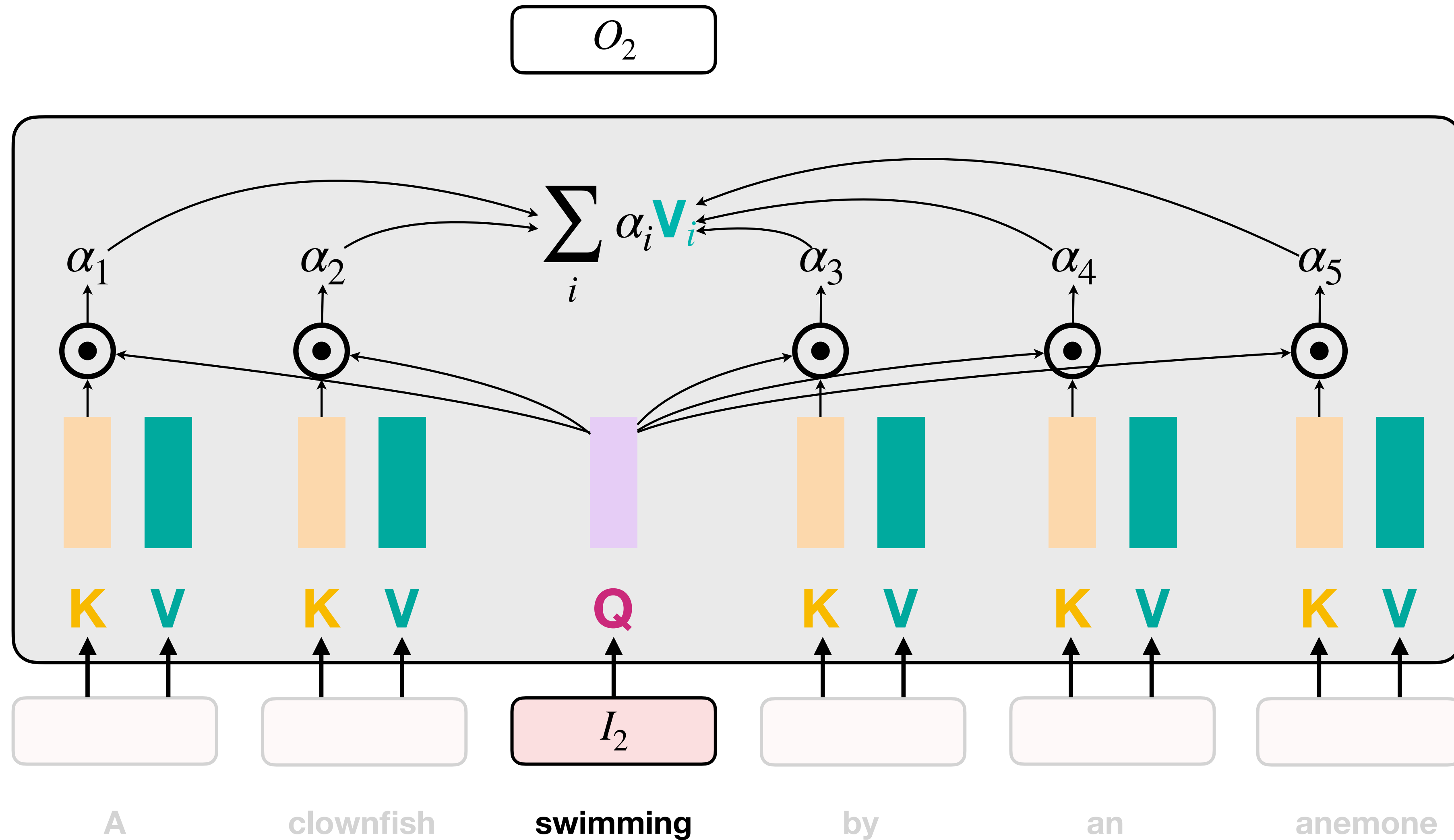
# Core component: Dot-product Attention (drastically simplified)



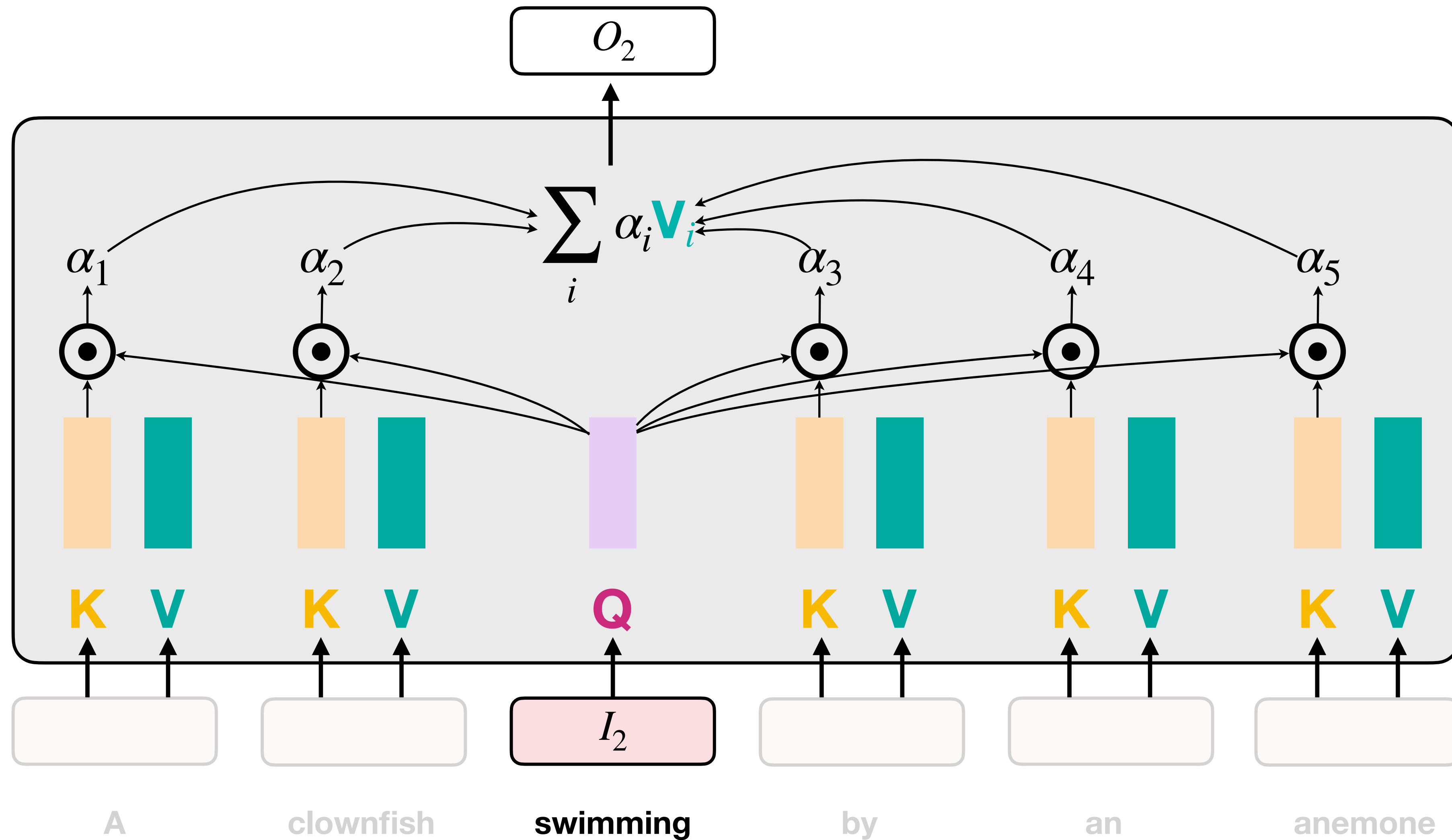
# Core component: Dot-product Attention (drastically simplified)



# Core component: Dot-product Attention (drastically simplified)



# Core component: Dot-product Attention (drastically simplified)



# Scaled Dot-Product Attention

$$\mathbf{K}_i, \mathbf{V}_i, \mathbf{Q}_i = \text{linear}(I_i), \quad \mathbf{K}_i, \mathbf{V}_i, \mathbf{Q}_i \in \mathbb{R}^k$$



# Scaled Dot-Product Attention

$$\mathbf{K}_i, \mathbf{V}_i, \mathbf{Q}_i = \text{linear}(I_i), \quad \mathbf{K}_i, \mathbf{V}_i, \mathbf{Q}_i \in \mathbb{R}^k$$

$$\alpha_i^j = \text{softmax} \left( \frac{\begin{array}{c} \mathbf{Q}_j \\ \text{ } \end{array} \odot \begin{array}{c} \mathbf{K}_i^T \\ \text{ } \end{array}}{\sqrt{d_k}} \right)$$

# Scaled Dot-Product Attention

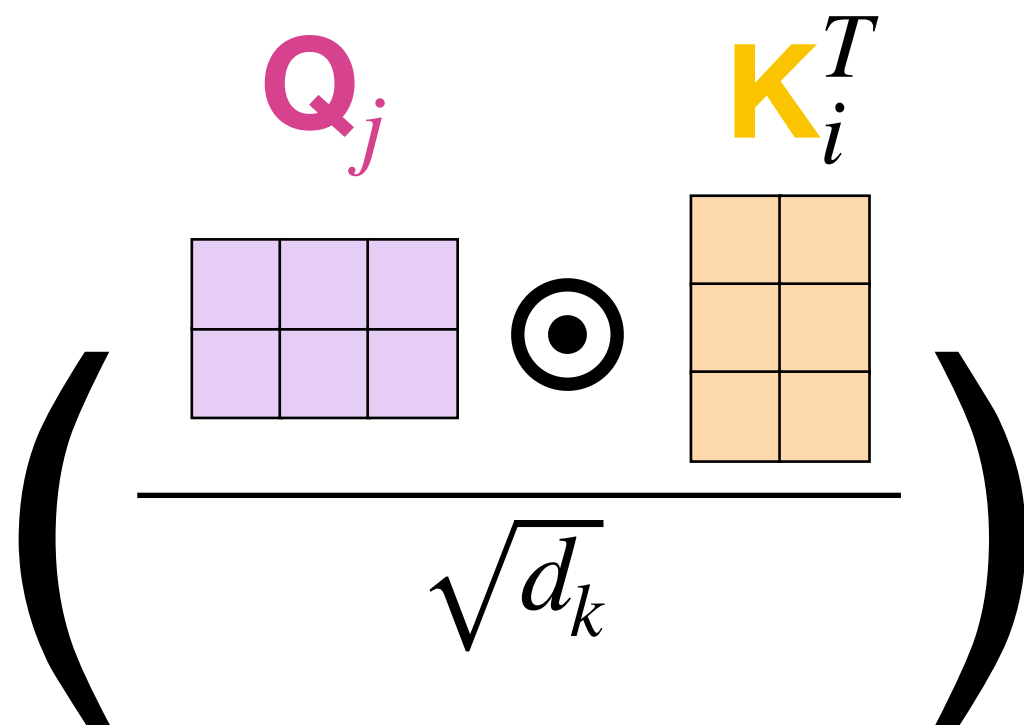
$$\mathbf{K}_i, \mathbf{V}_i, \mathbf{Q}_i = \text{linear}(I_i), \quad \mathbf{K}_i, \mathbf{V}_i, \mathbf{Q}_i \in \mathbb{R}^k$$

$$\alpha_i^j = \text{softmax} \left( \frac{\begin{array}{c} \mathbf{Q}_j \\ \text{---} \odot \text{---} \\ \mathbf{K}_i^T \end{array}}{\sqrt{d_k}} \right)$$

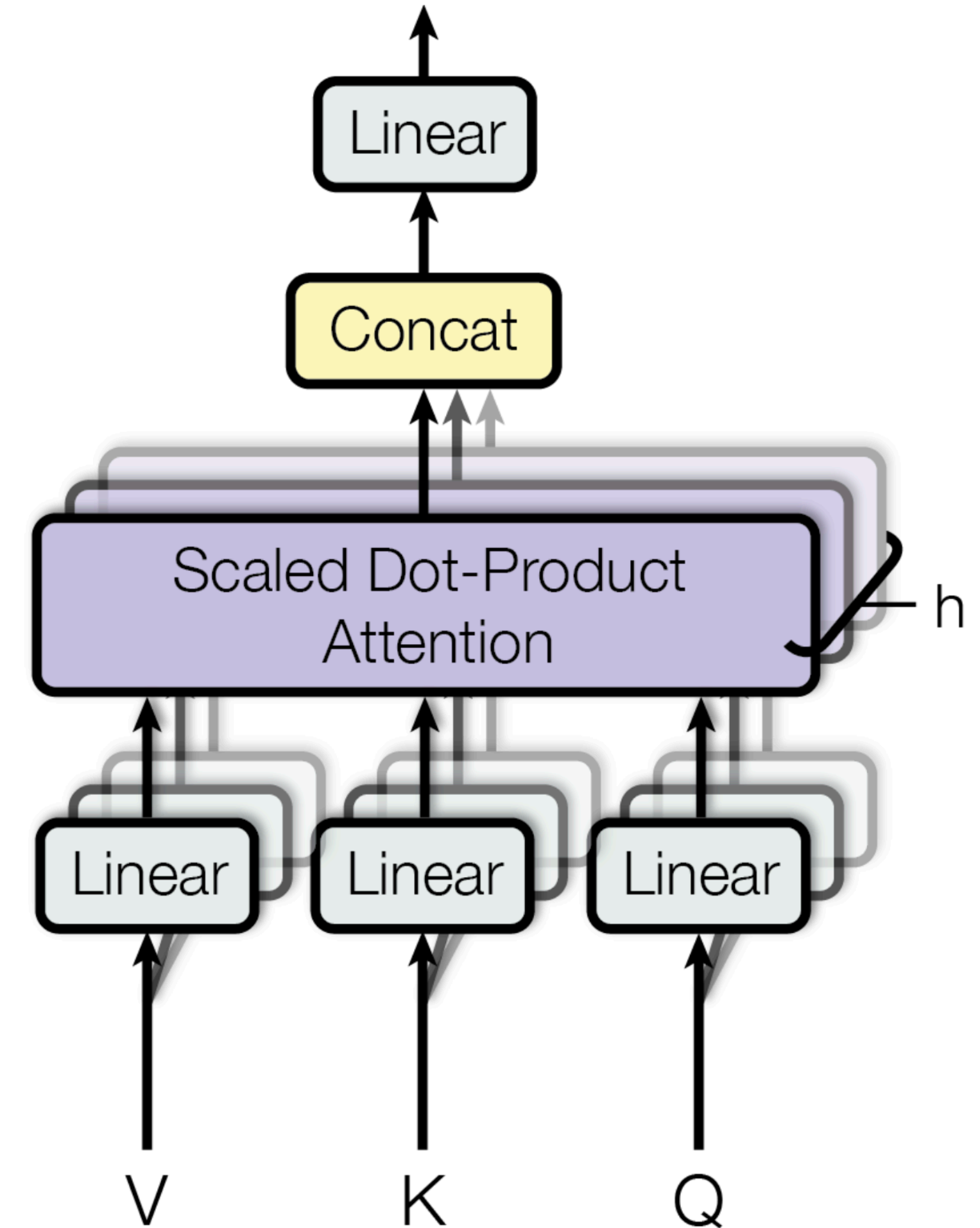
$$O_j = \sum_i \alpha_i^j \mathbf{V}_i$$

# Multi-Head Attention

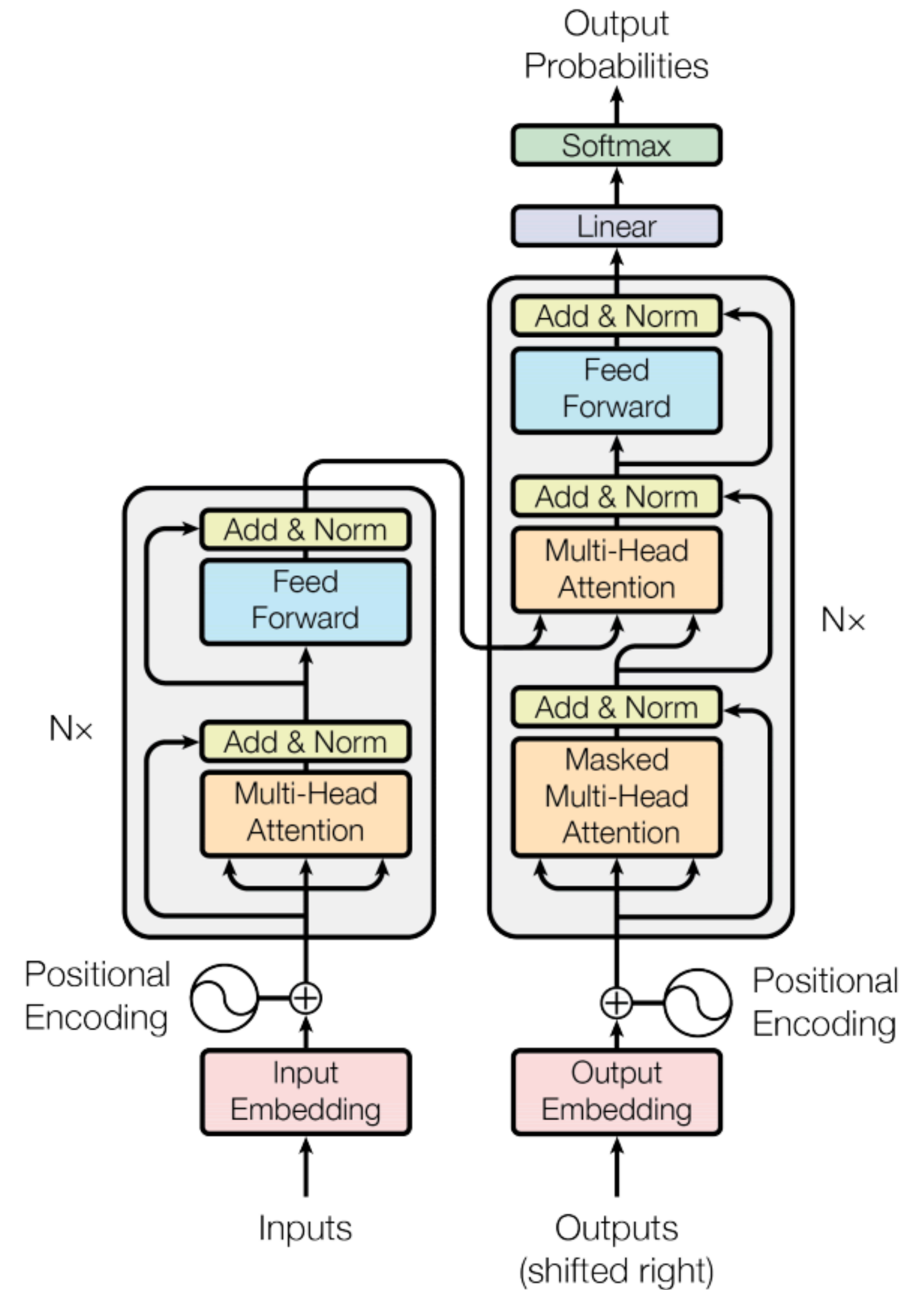
$$\mathbf{K}_i, \mathbf{V}_i, \mathbf{Q}_i = \text{linear}(I_i), \quad \mathbf{K}_i, \mathbf{V}_i, \mathbf{Q}_i \in \mathbb{R}^k$$

$$\alpha_i^j = \text{softmax} \left( \frac{\mathbf{Q}_j \odot \mathbf{K}_i^T}{\sqrt{d_k}} \right)$$


$$O_j = \sum_i \alpha_i^j \mathbf{V}_i$$

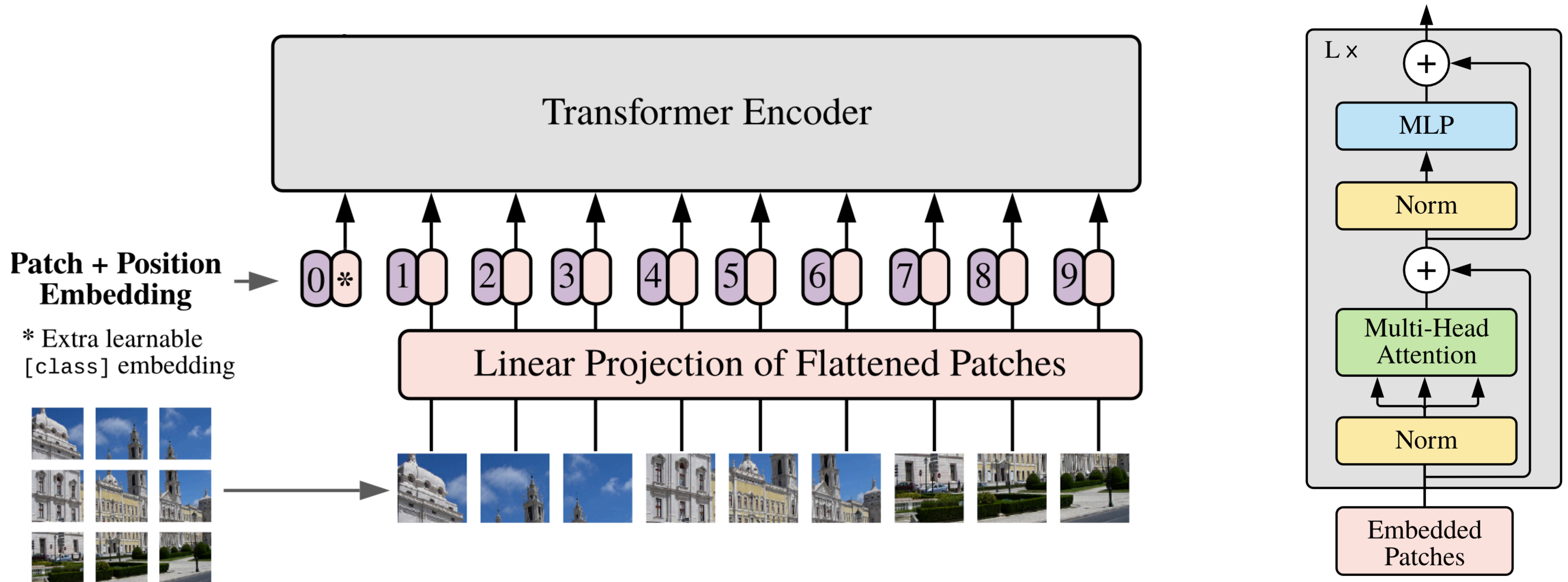


# The Transformer

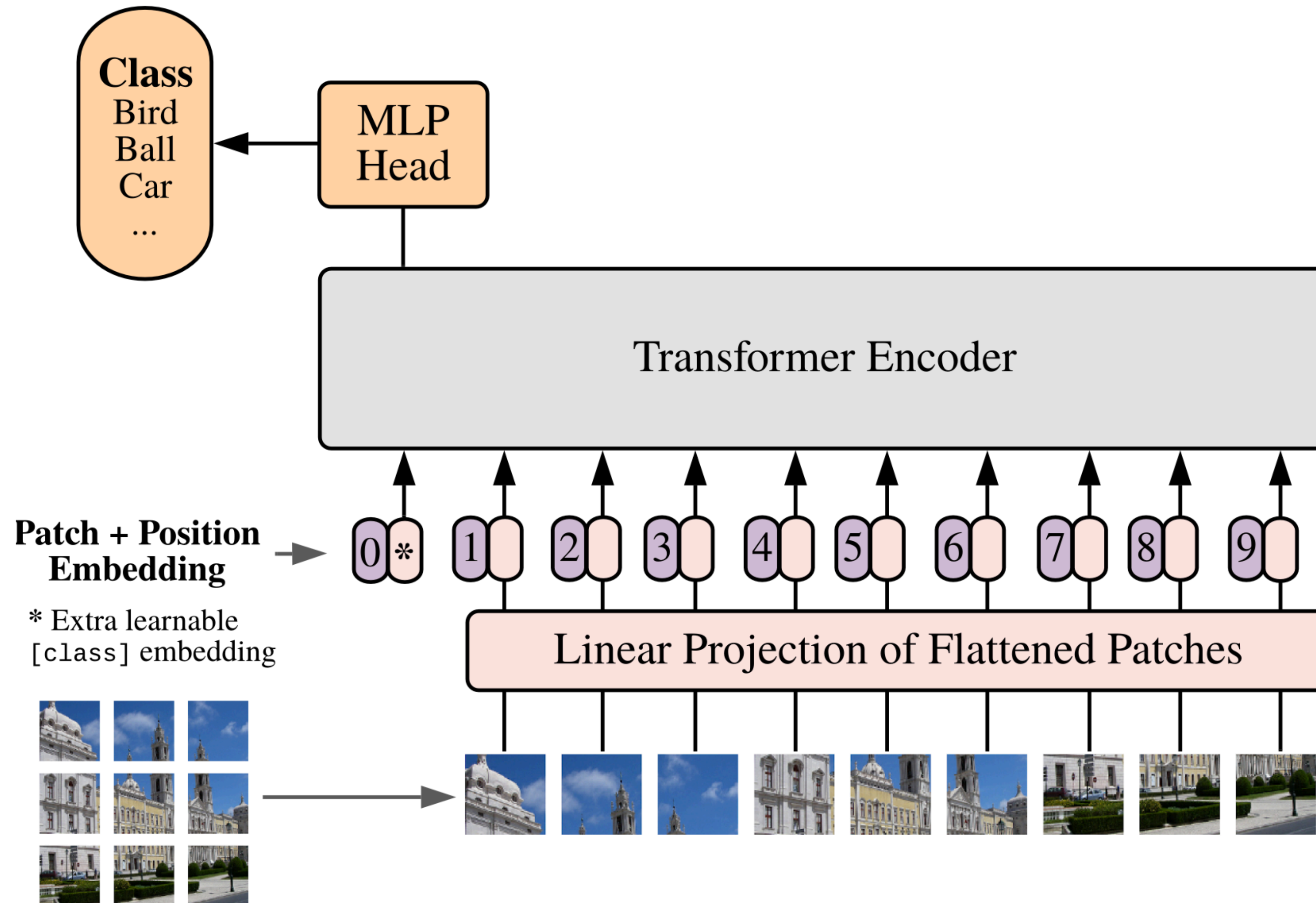


["Attention is all you need", Vaswani et al. 2017]

# Not only for Seq2Seq: Vision Transformers



# Not only for Seq2Seq: Vision Transformers



# Image Captioning... with Transformers

