Using RNNs to generate Super Mario Maker levels, Adam Geitgey

COMP547

DEEP UNSUPERVISED LEARNING

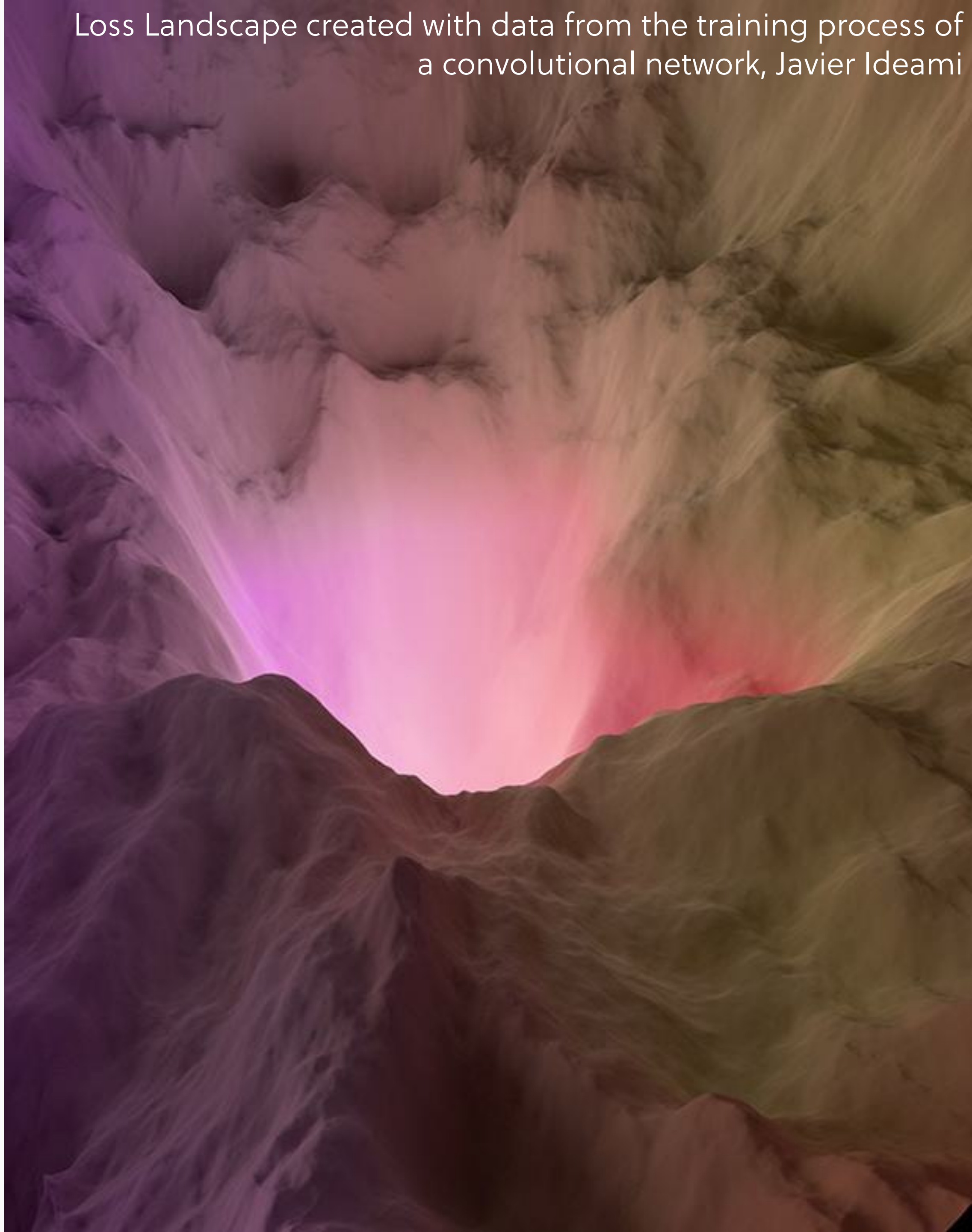
Lecture #3 – Neural Networks Basics II: Sequential Processing with NNs

KOÇ UNIVERSITY

Aykut Erdem // Koç University // Spring 2026

Previously on COMP547

- deep learning
- computation in a neural net
- optimization
- backpropagation
- training tricks
- convolutional neural networks


Loss Landscape created with data from the training process of a convolutional network, Javier Ideami

Lecture overview

- sequence modeling
- recurrent neural networks (RNNs)
- how to train RNNs
- long short-term memory (LSTM)
- gated recurrent unit (GRU)
- sequence to sequence modeling

Disclaimer: Much of the material and slides for this lecture were borrowed from

—Bill Freeman, Antonio Torralba and Phillip Isola's MIT 6.869 class
—Fei-Fei Li, Andrej Karpathy and Justin Johnson's CS231n class
—Arun Mallya's tutorial on Recurrent Neural Networks

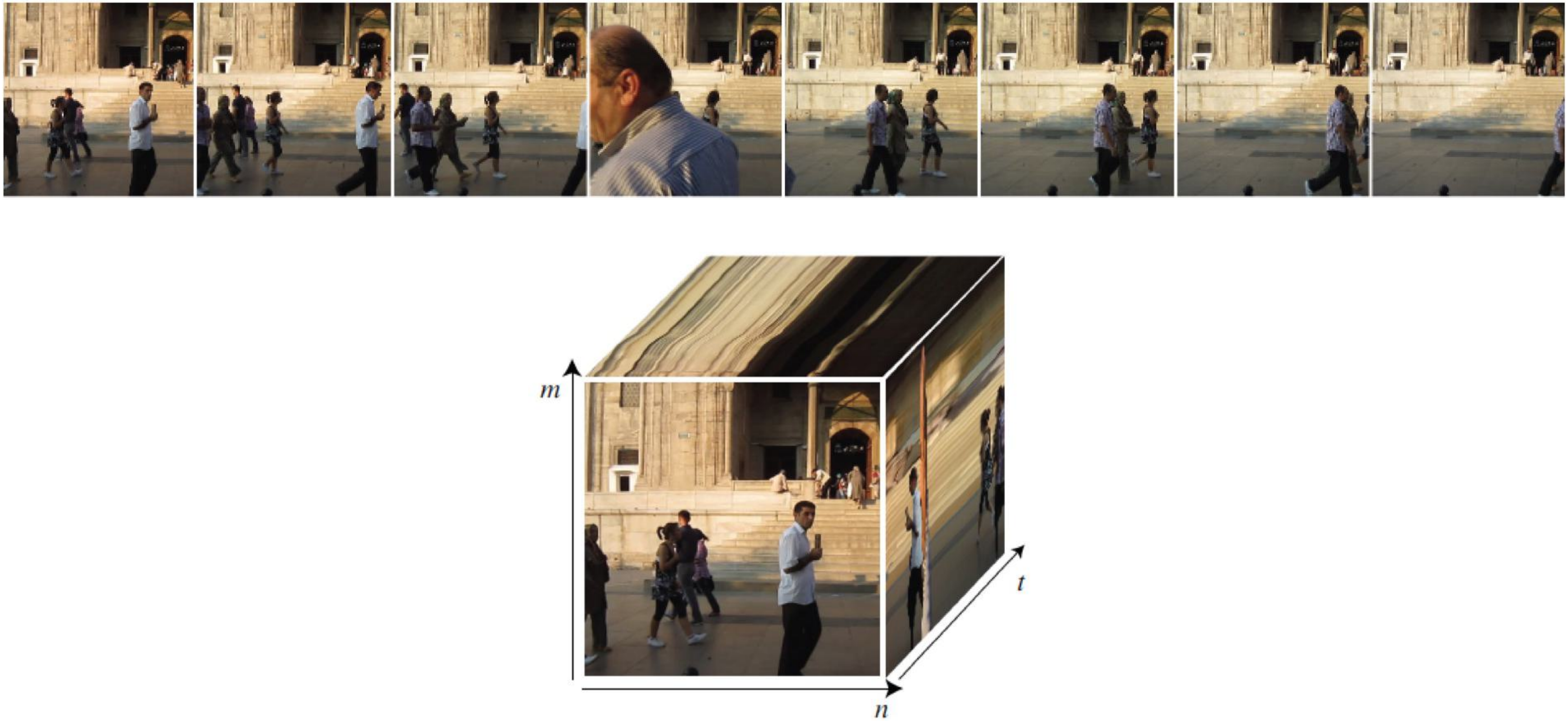
Sequences

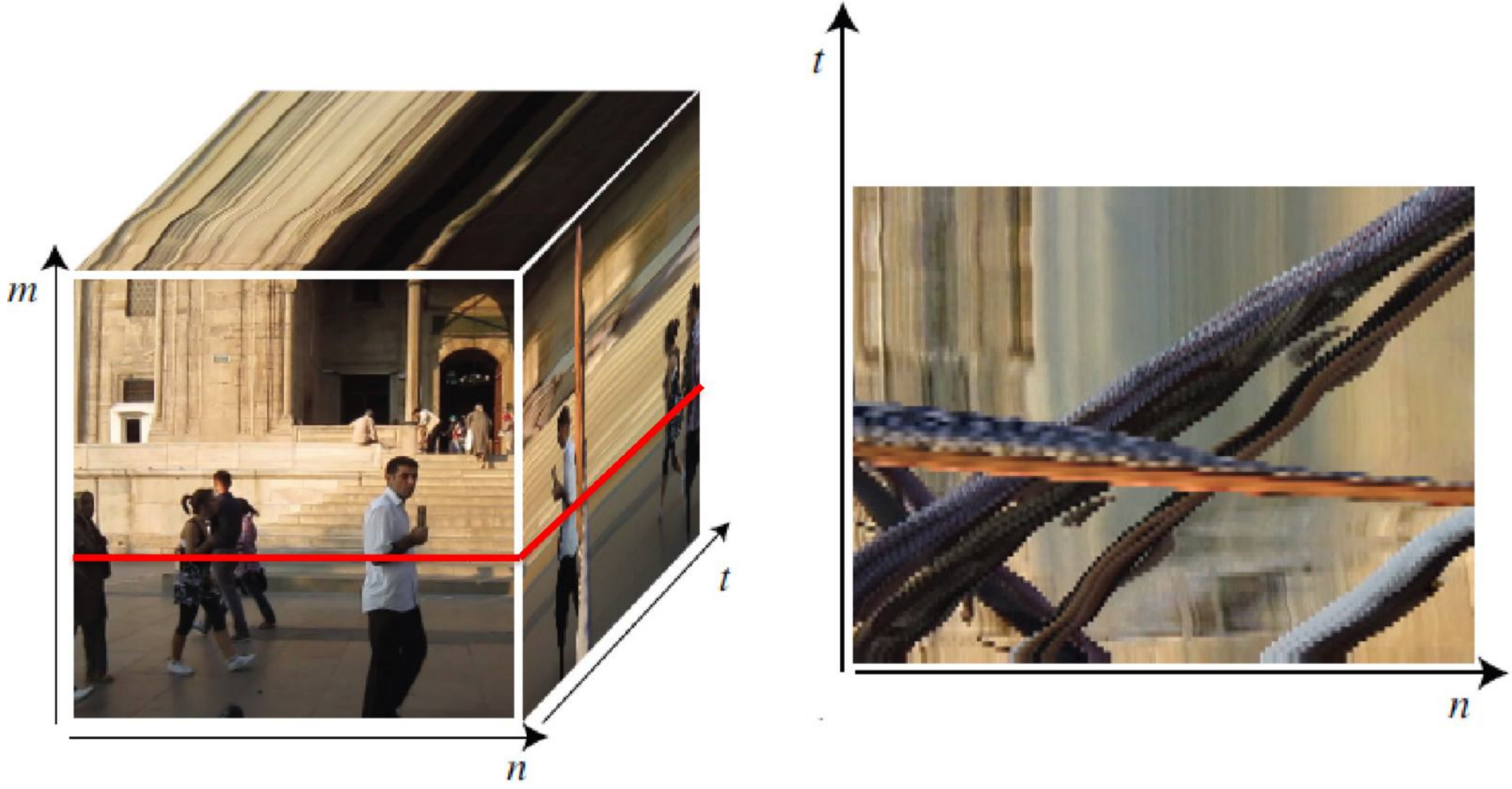
time

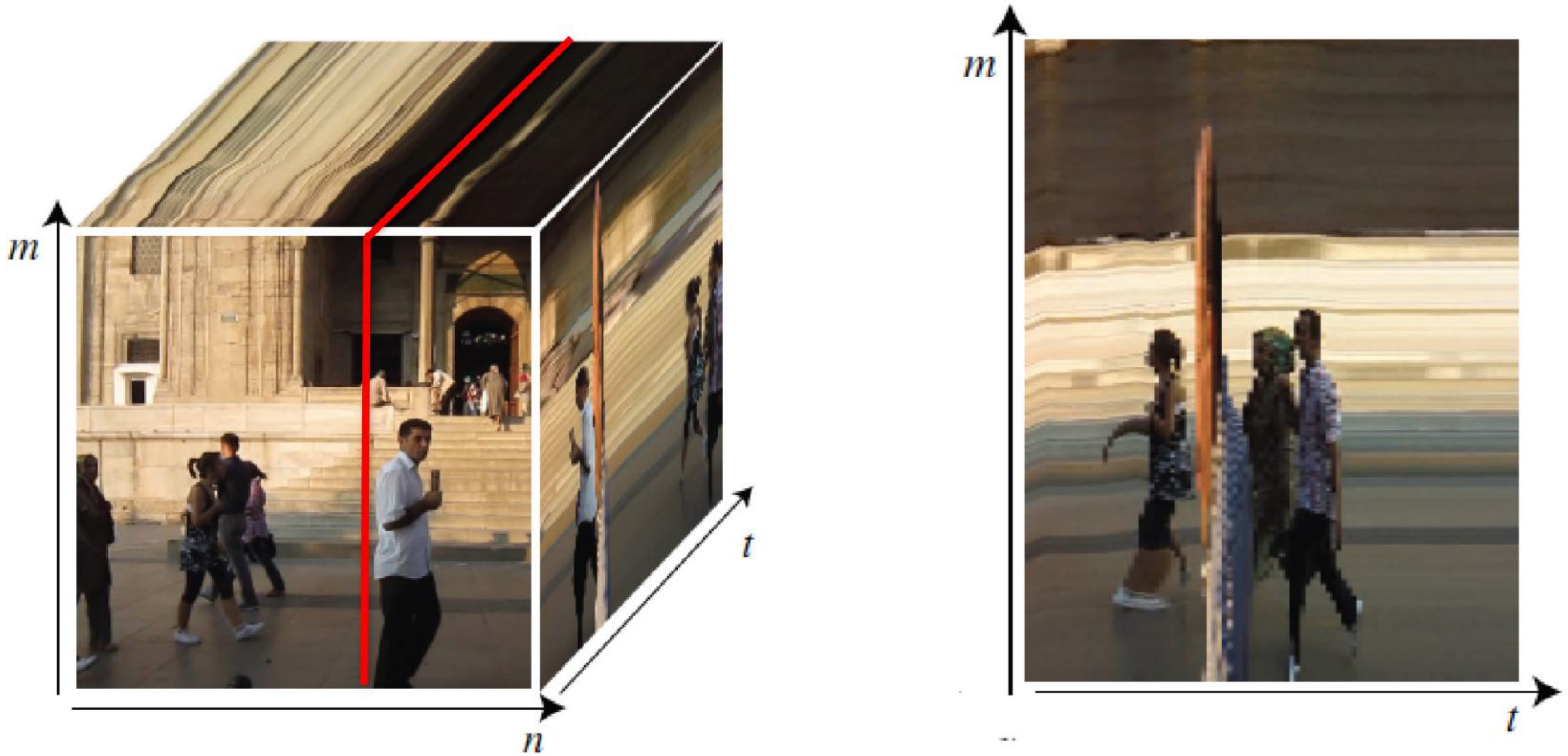
“An”, “evening”, “stroll”, “through”, “a”, “city”, “square”

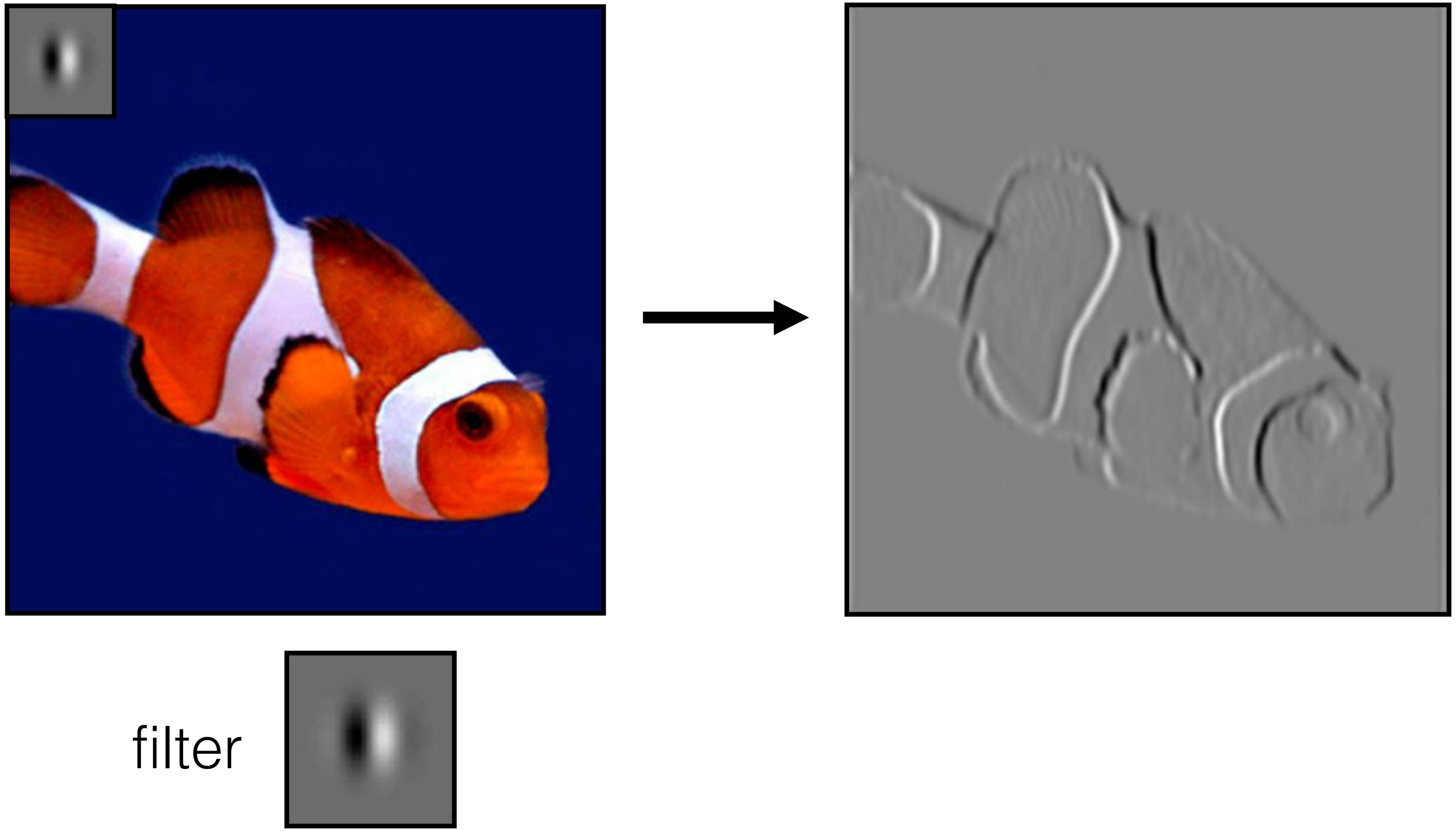
time

time

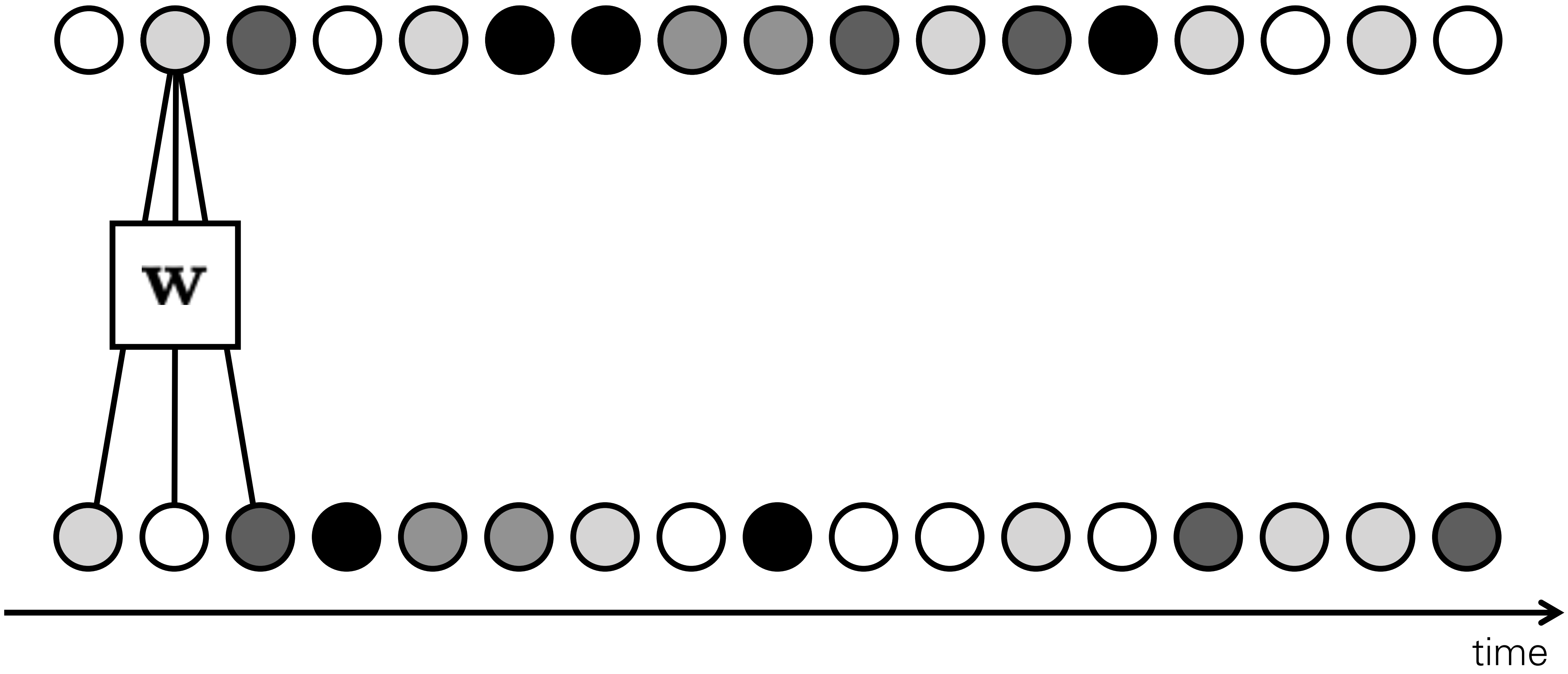

m
n
t

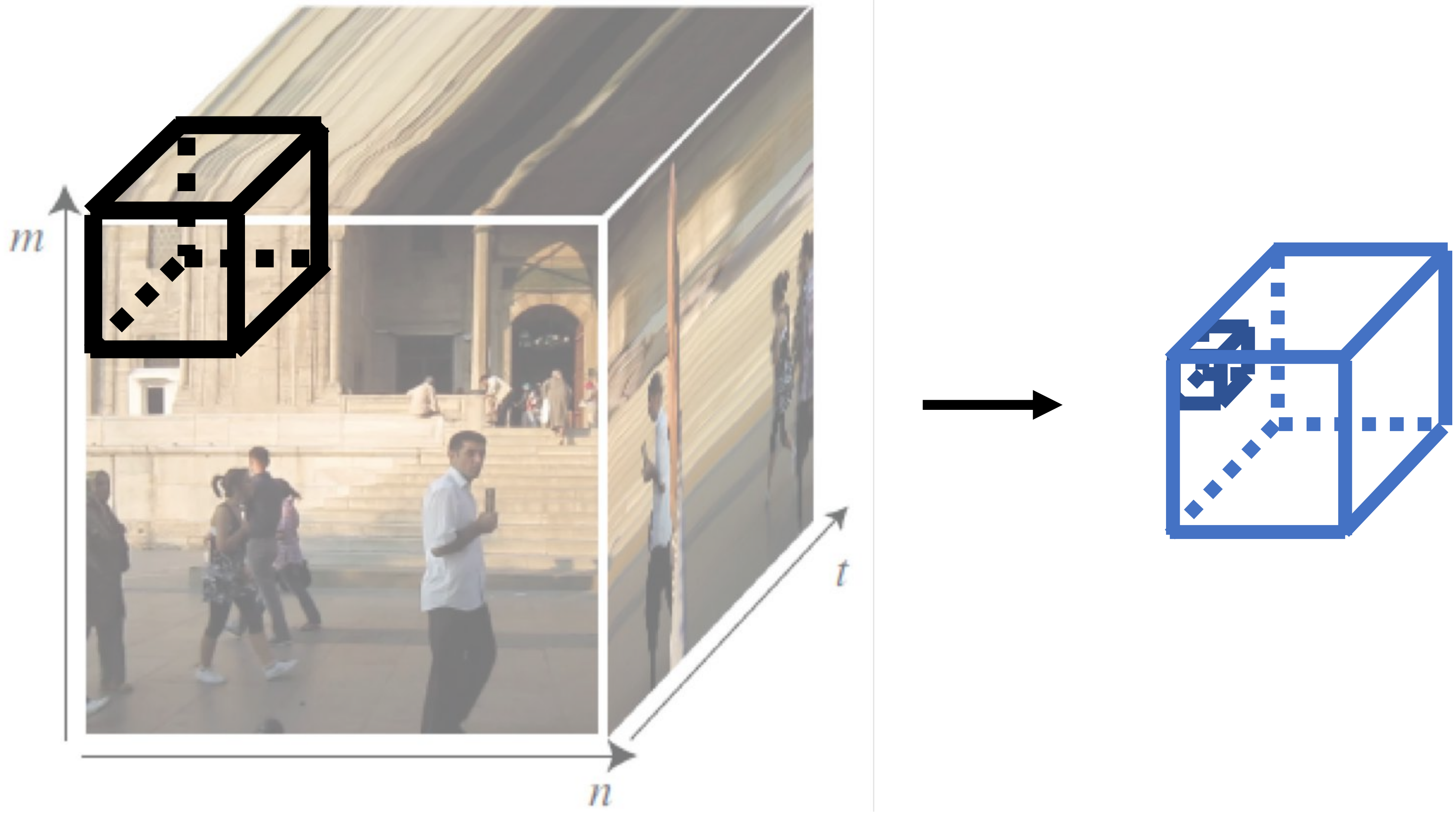

m
n
t
t
n

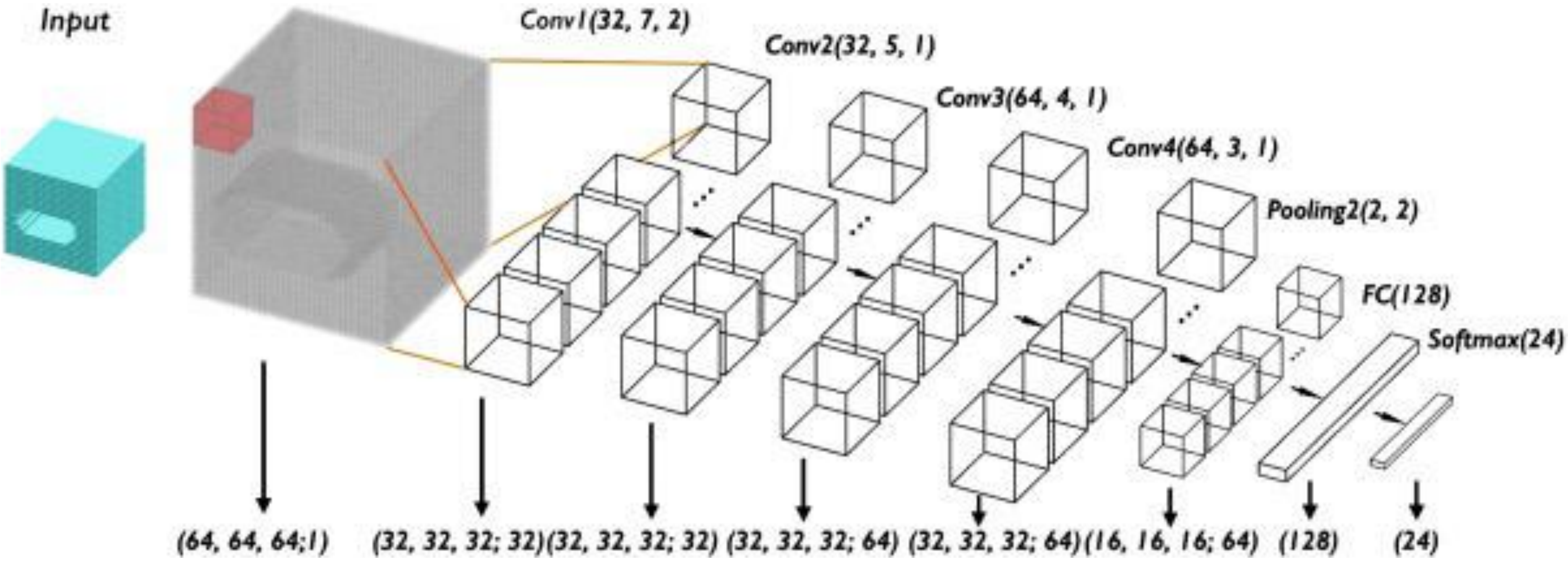

m
n
t
m
t


filter

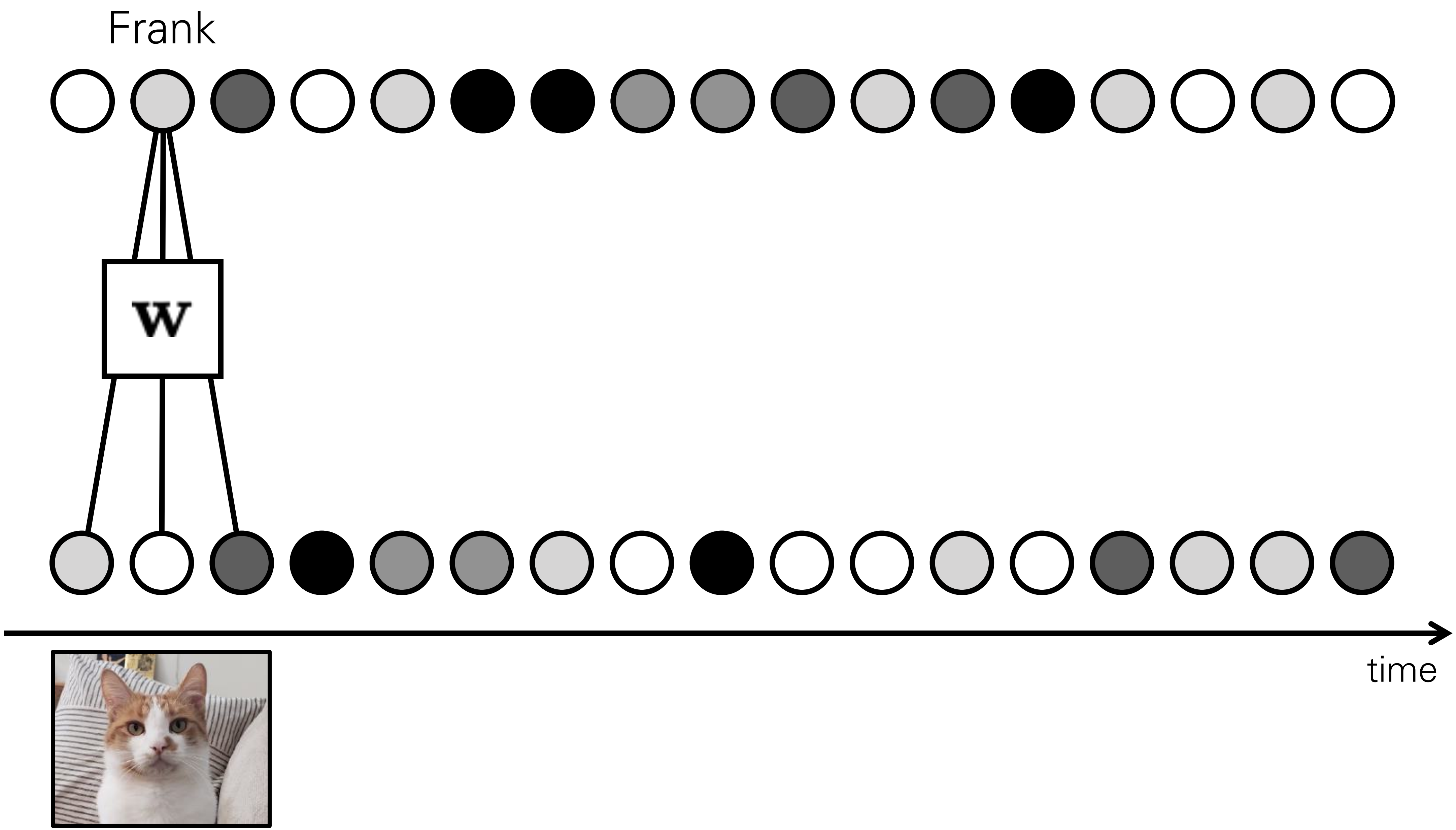
Convolutions in time

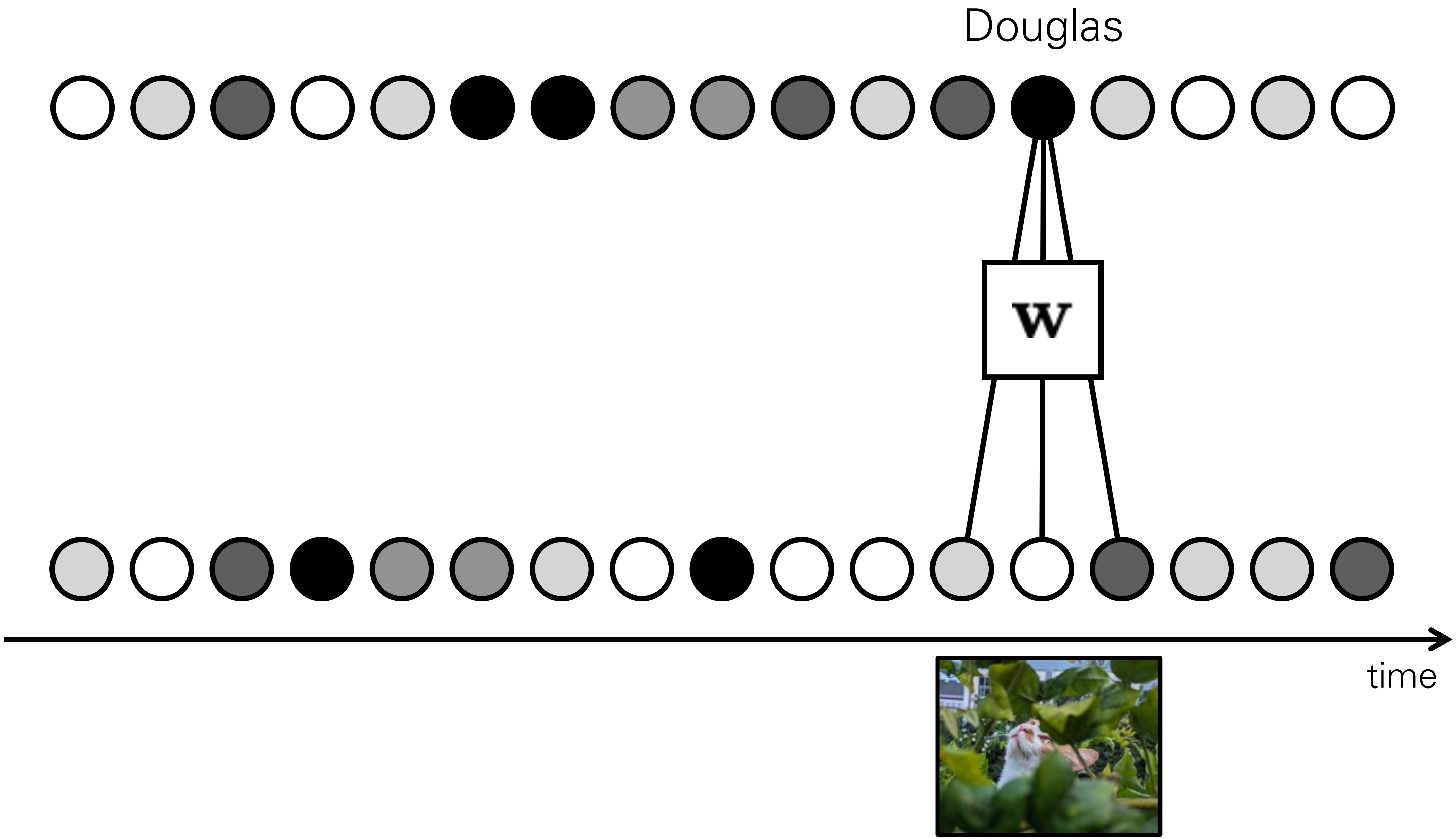


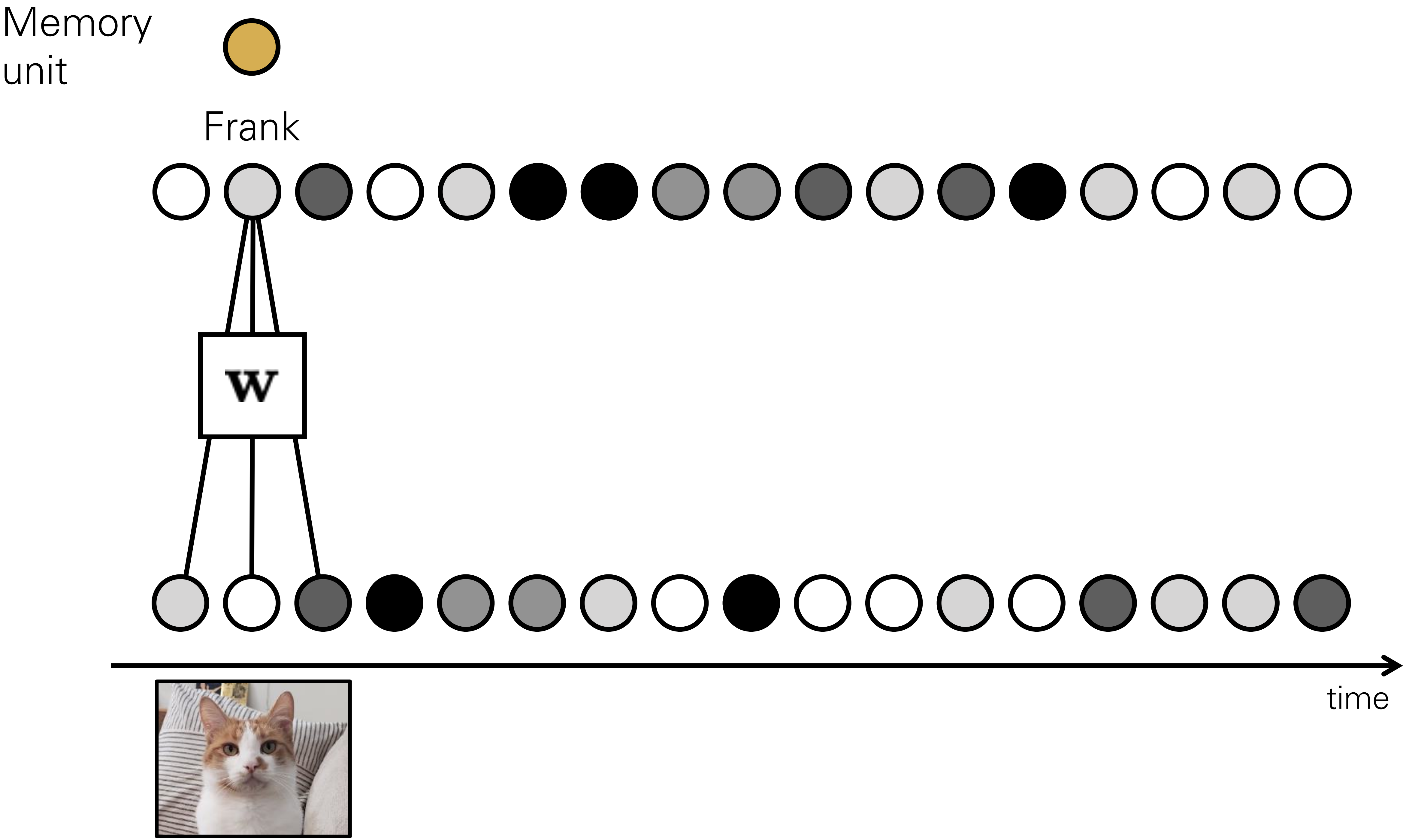

m
n
t

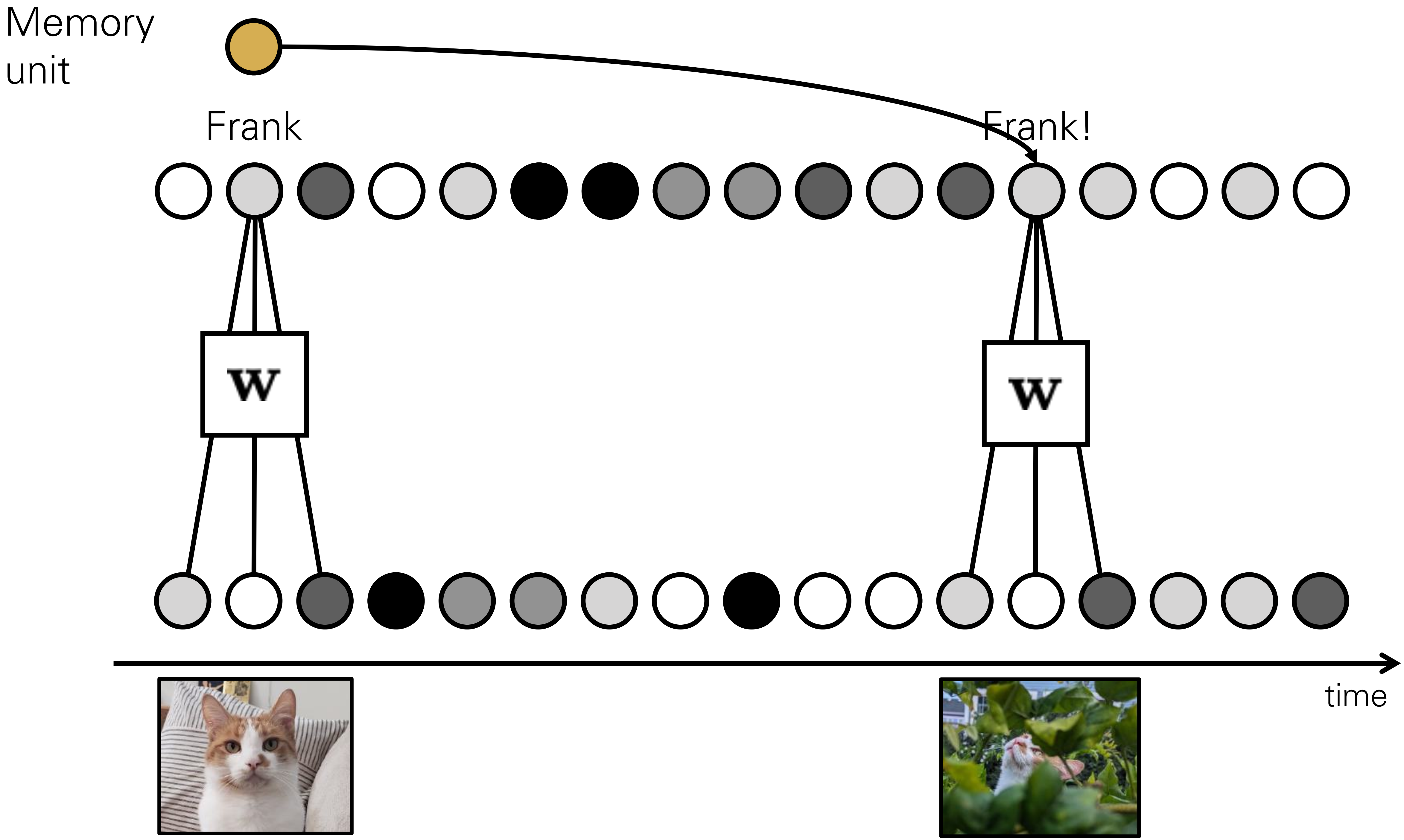


[fig from FeatureNet: Machining feature recognition based on 3D Convolution Neural Network]

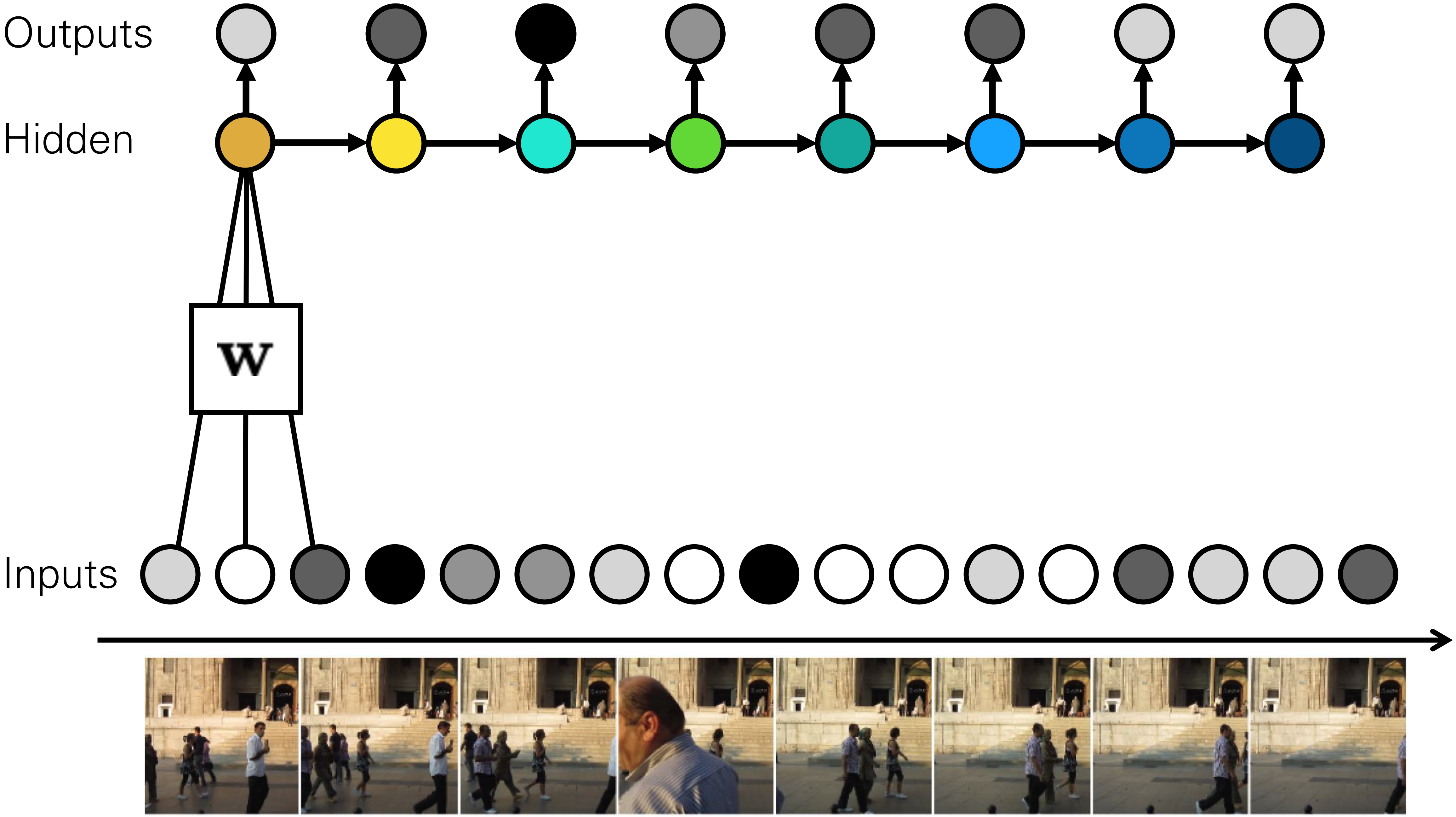

Frank
w
time


Douglas
w
time


Memory
unit
Frank
w
time


Memory
unit
Frank
Frank!
w
w
time

Recurrent Neural Networks (RNNs)

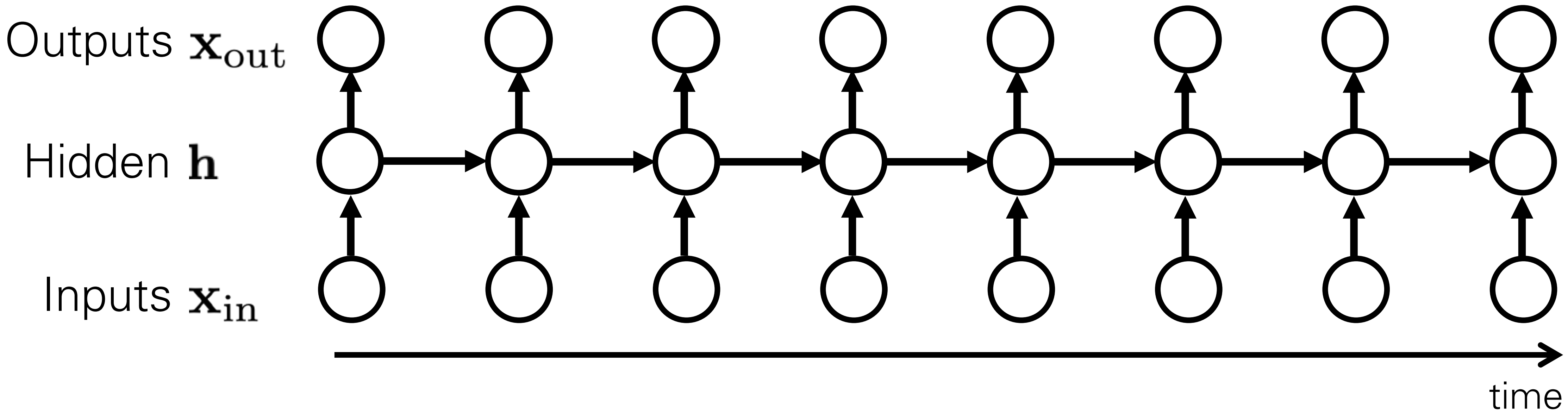


To model sequences, we need

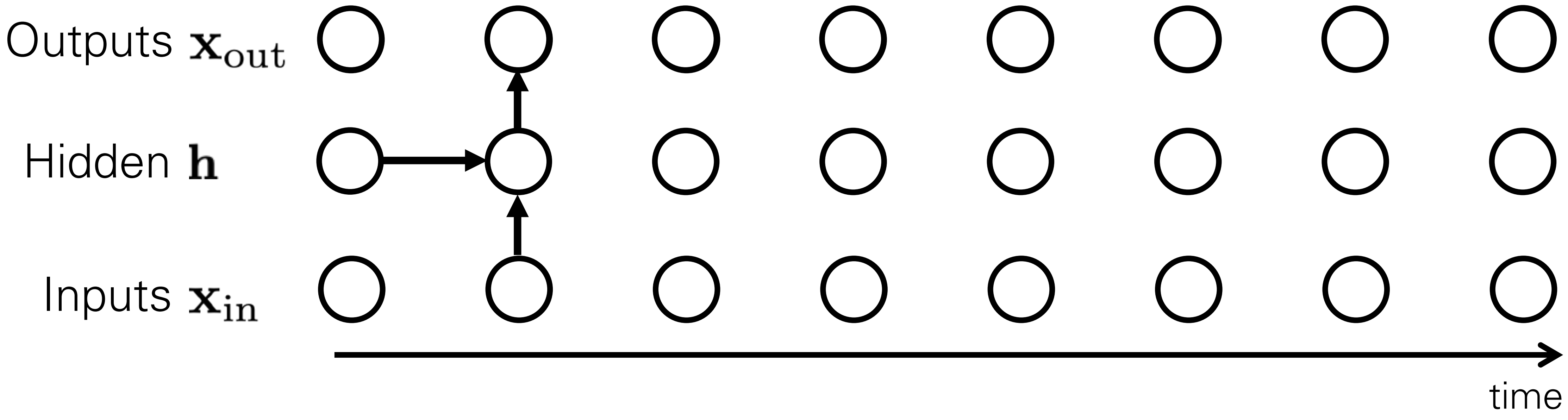
1. to deal with variable length sequences
2. to maintain sequence order
3. to keep track of long-term dependencies
4. to share parameters across the sequence

Recurrent Neural Networks

Recurrent Neural Networks (RNNs)



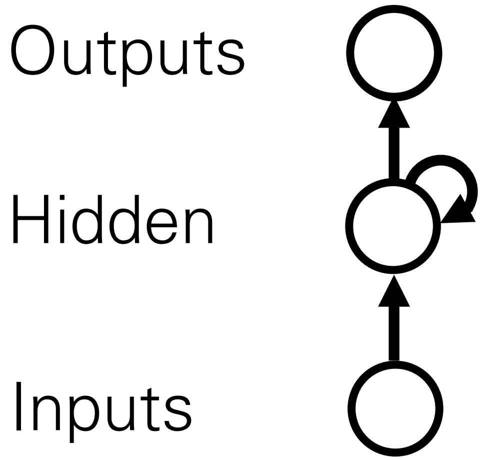
Recurrent Neural Networks (RNNs)



$$\mathbf{h}_t = f\left(\mathbf{h}_{t-1}, \mathbf{x}_{\text{in}}[t]\right)$$

$$\mathbf{x}_{\text{out}}\left[t\right] = g\left(\mathbf{h}_t\right)$$

Recurrent Neural Networks (RNNs)

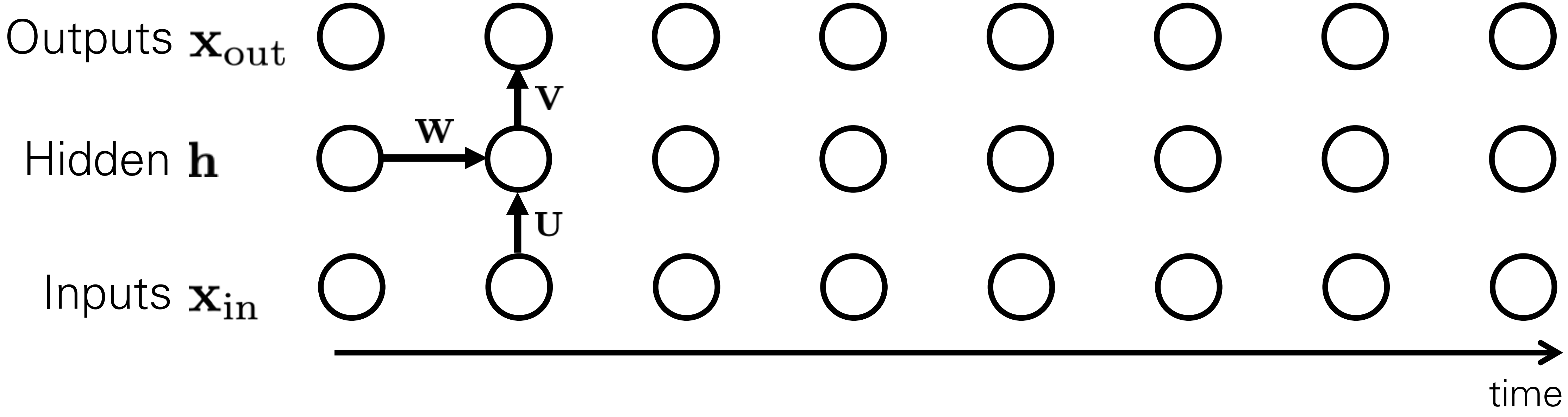


Recurrent!

$$\mathbf{h}_t = f\left(\mathbf{h}_{t-1}, \mathbf{x}_{\text{in}}[t]\right)$$

$$\mathbf{x}_{\text{out}}\left[t\right] = g\left(\mathbf{h}_t\right)$$

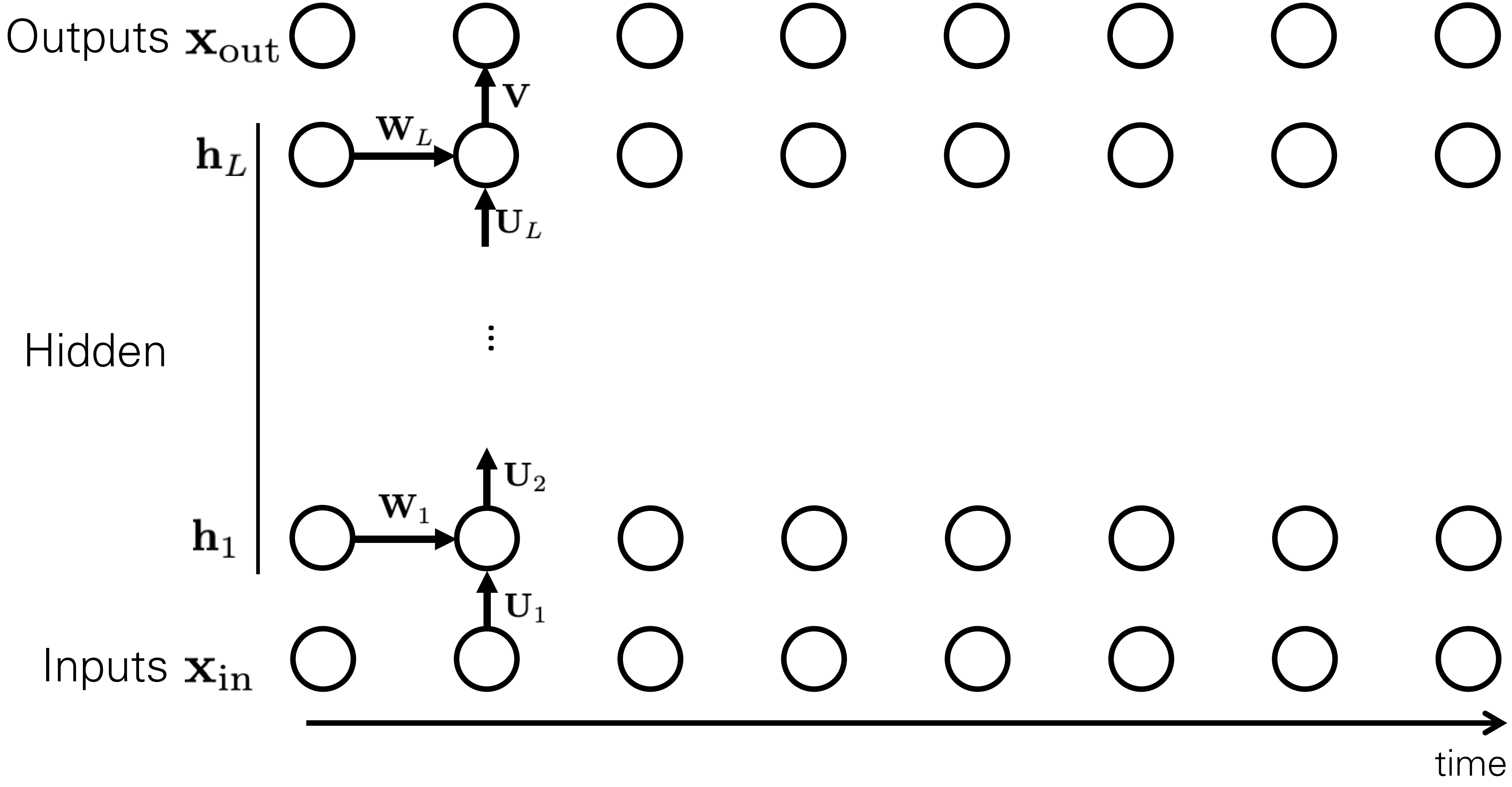
Recurrent Neural Networks (RNNs)



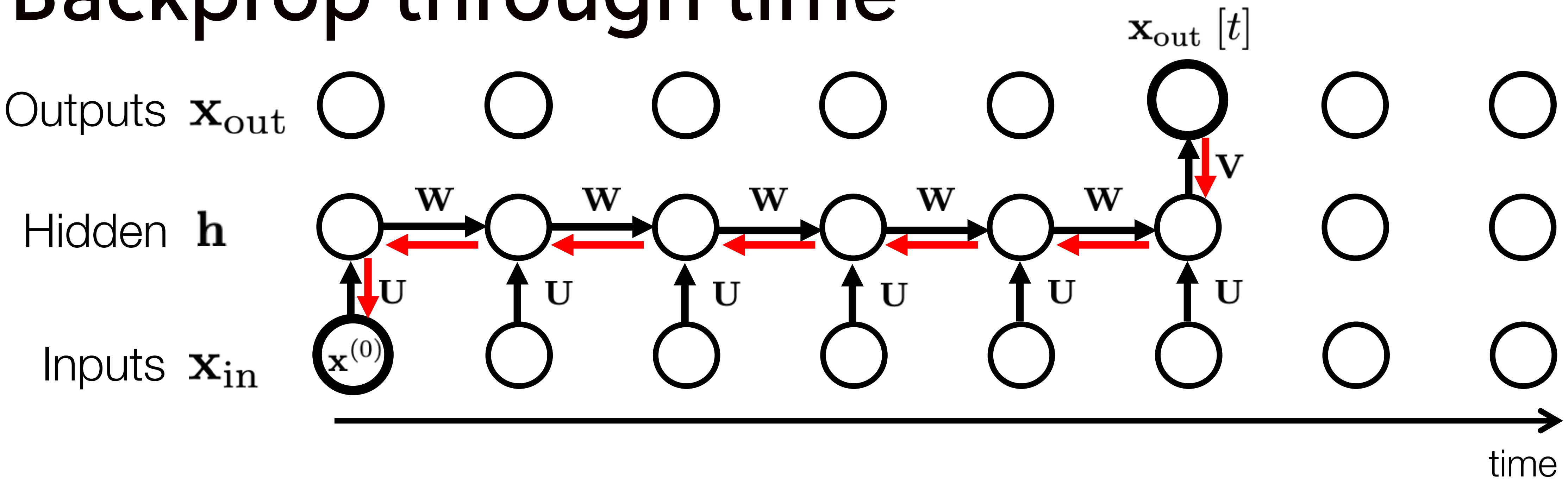
$$\mathbf{h}_t = \sigma_1 \left(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_{\text{in}} \left[t \right] + \mathbf{b} \right)$$

$$\mathbf{x}_{\text{out}} \left[t \right] = \sigma_2 \left(\mathbf{V}\mathbf{h}_t + \mathbf{c} \right)$$

Deep Recurrent Neural Networks (RNNs)

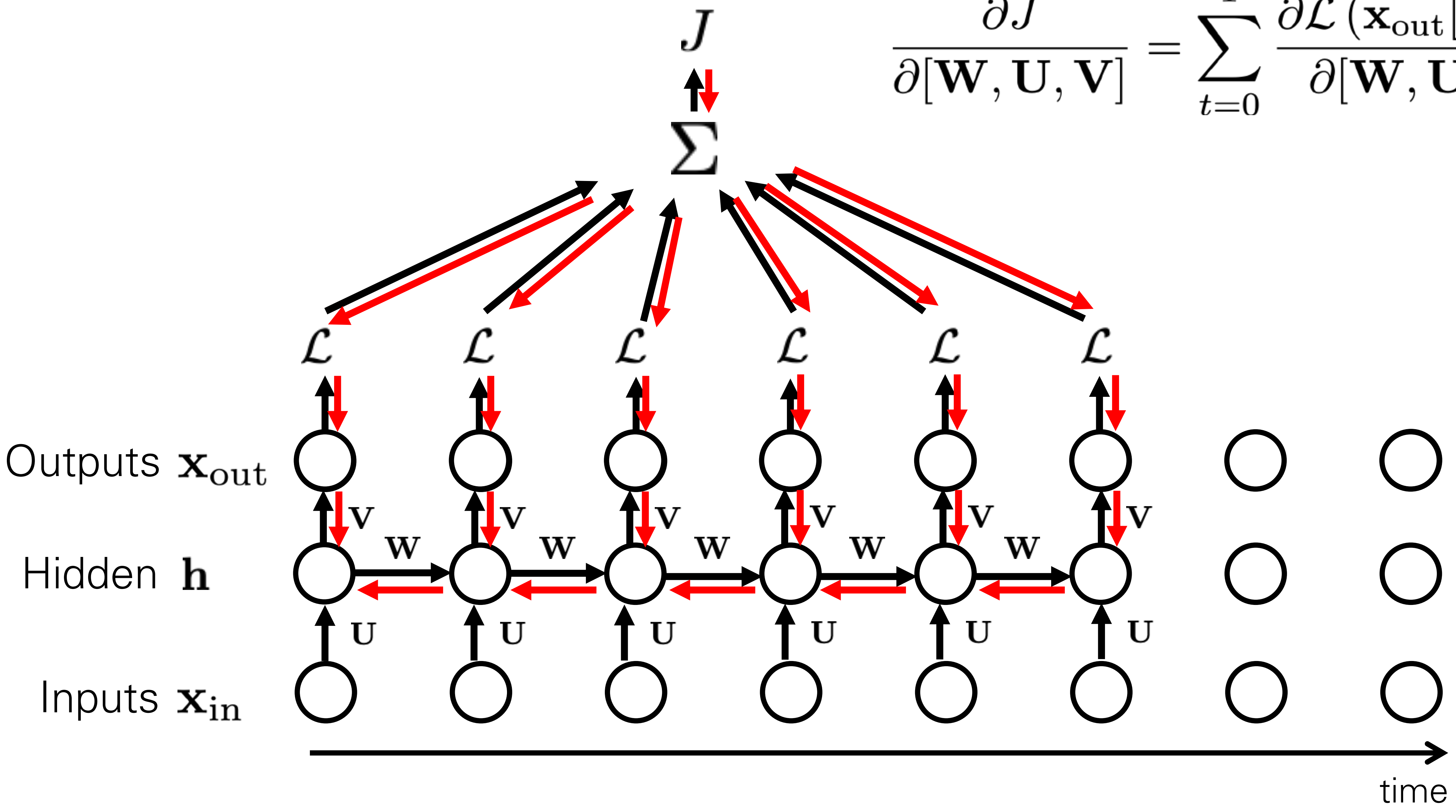


Backprop through time

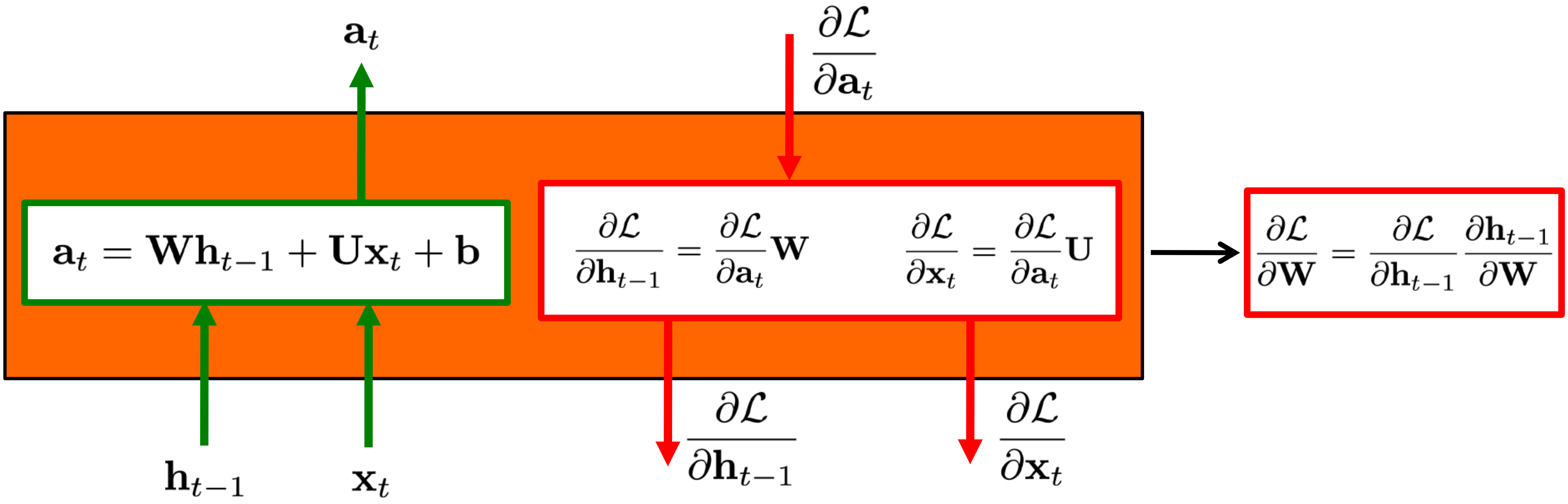


$$\frac{\partial \mathbf{x}_{\text{out}}[t]}{\partial \mathbf{x}_{\text{in}}[0]} = \frac{\partial \mathbf{x}_{\text{out}}[t]}{\partial \mathbf{h}_T} \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_{T-1}} \cdots \frac{\partial \mathbf{h}_1}{\partial \mathbf{h}_0} \frac{\partial \mathbf{h}_0}{\partial \mathbf{x}_{\text{in}}[0]}$$

$$\frac{\partial J}{\partial [\mathbf{W}, \mathbf{U}, \mathbf{V}]} = \sum_{t=0}^{T} \frac{\partial \mathcal{L}\left(\mathbf{x}_{\text{out}}[t], \mathbf{y}_t\right)}{\partial [\mathbf{W}, \mathbf{U}, \mathbf{V}]}$$

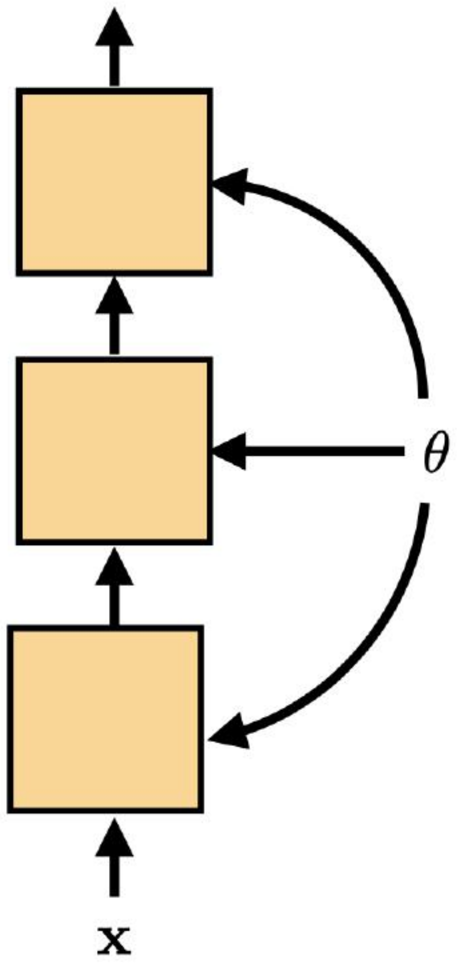


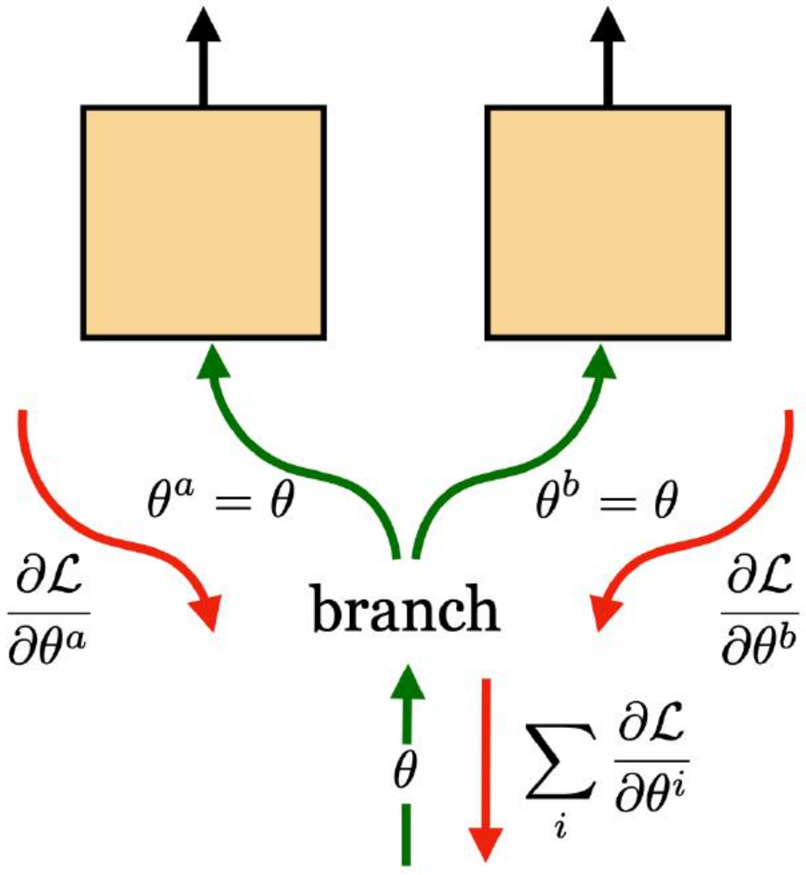
Recurrent Linear Layer



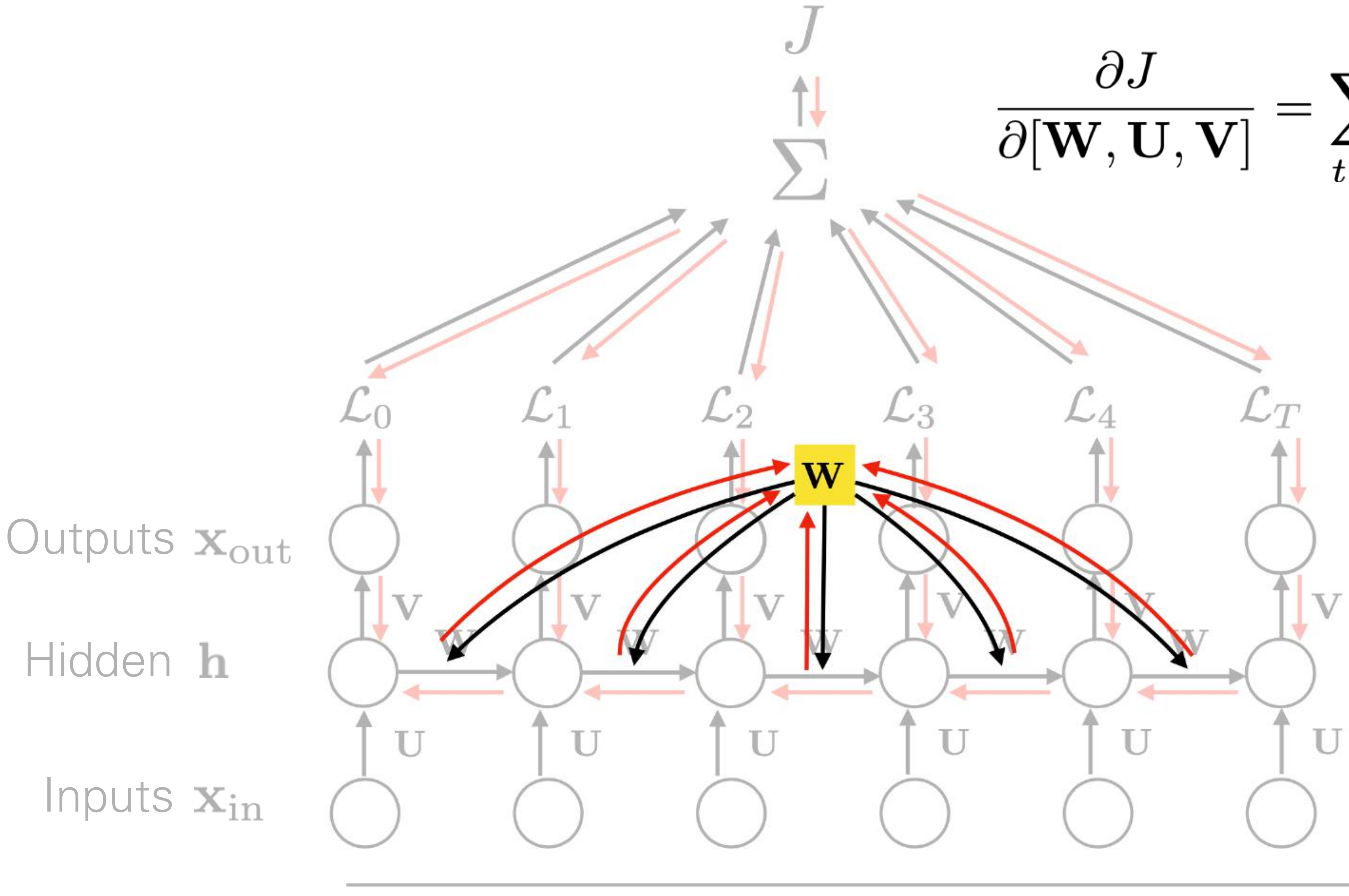
$$\frac{\partial J}{\partial \mathbf{W}} = \sum_{t=0}^{T} \frac{\partial \mathcal{L}(\hat{\mathbf{y}}_t, \mathbf{y}_t)}{\partial \mathbf{W}}$$

Parameter Sharing





Parameter sharing —> sum gradients



$$\frac{\partial J}{\partial[\mathbf{W}, \mathbf{U}, \mathbf{V}]} = \sum_{t=0}^{\mathrm{T}} \frac{\partial \mathcal{L}\left(\mathbf{x}_{\text{out}}[t], \mathbf{y}_t\right)}{\partial[\mathbf{W}, \mathbf{U}, \mathbf{V}]}$$

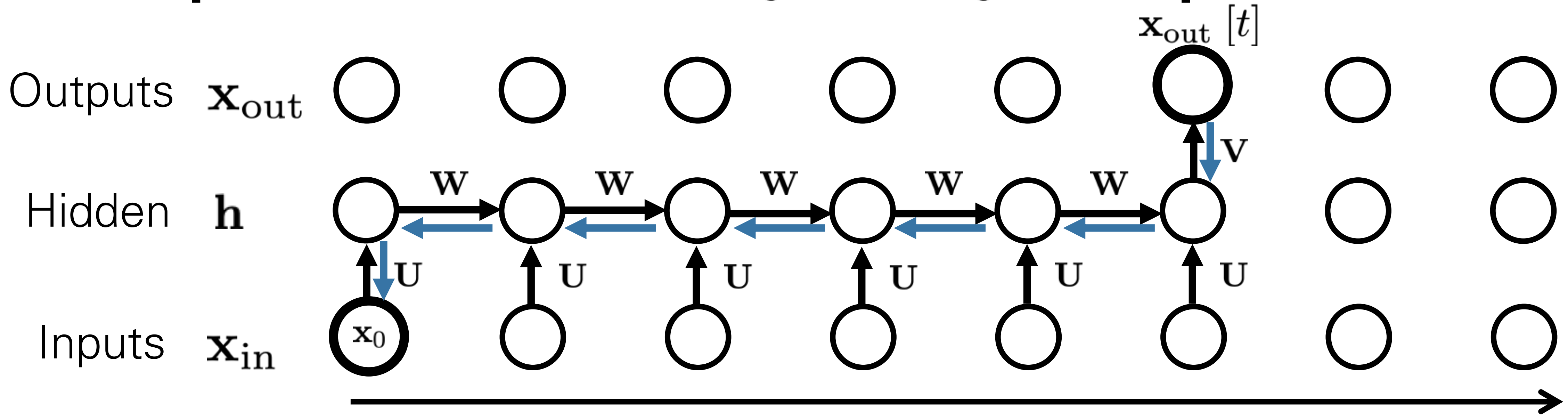
$$\frac{\partial \mathcal{L}_t}{\partial \mathbf{W}} = \sum_i \frac{\partial \mathcal{L}_t}{\partial \mathbf{W}^i}$$

The problem of long-range dependencies

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 into the hidden state, RNNs can model dependencies that are arbitrarily far apart

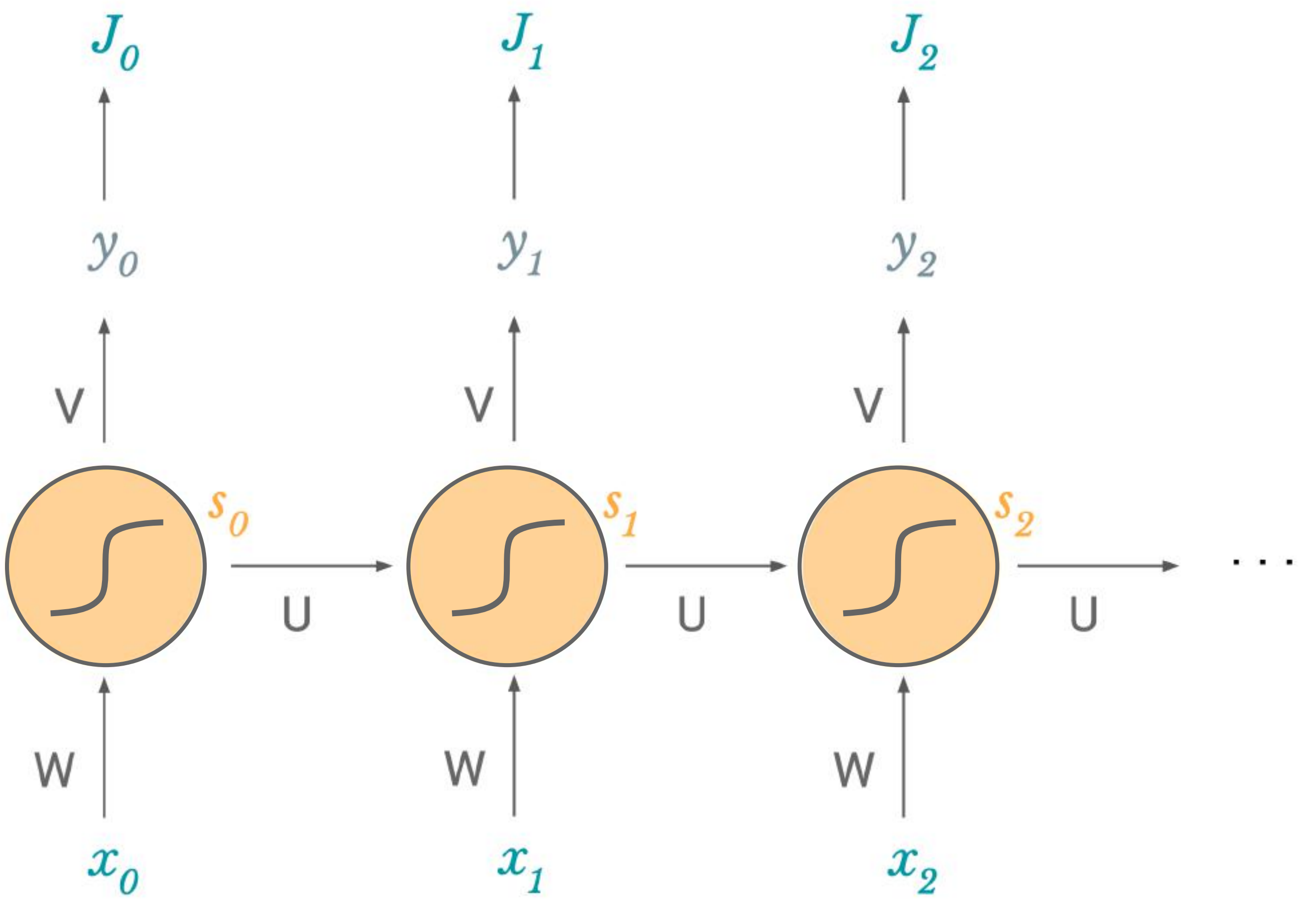
The problem of long-range dependencies



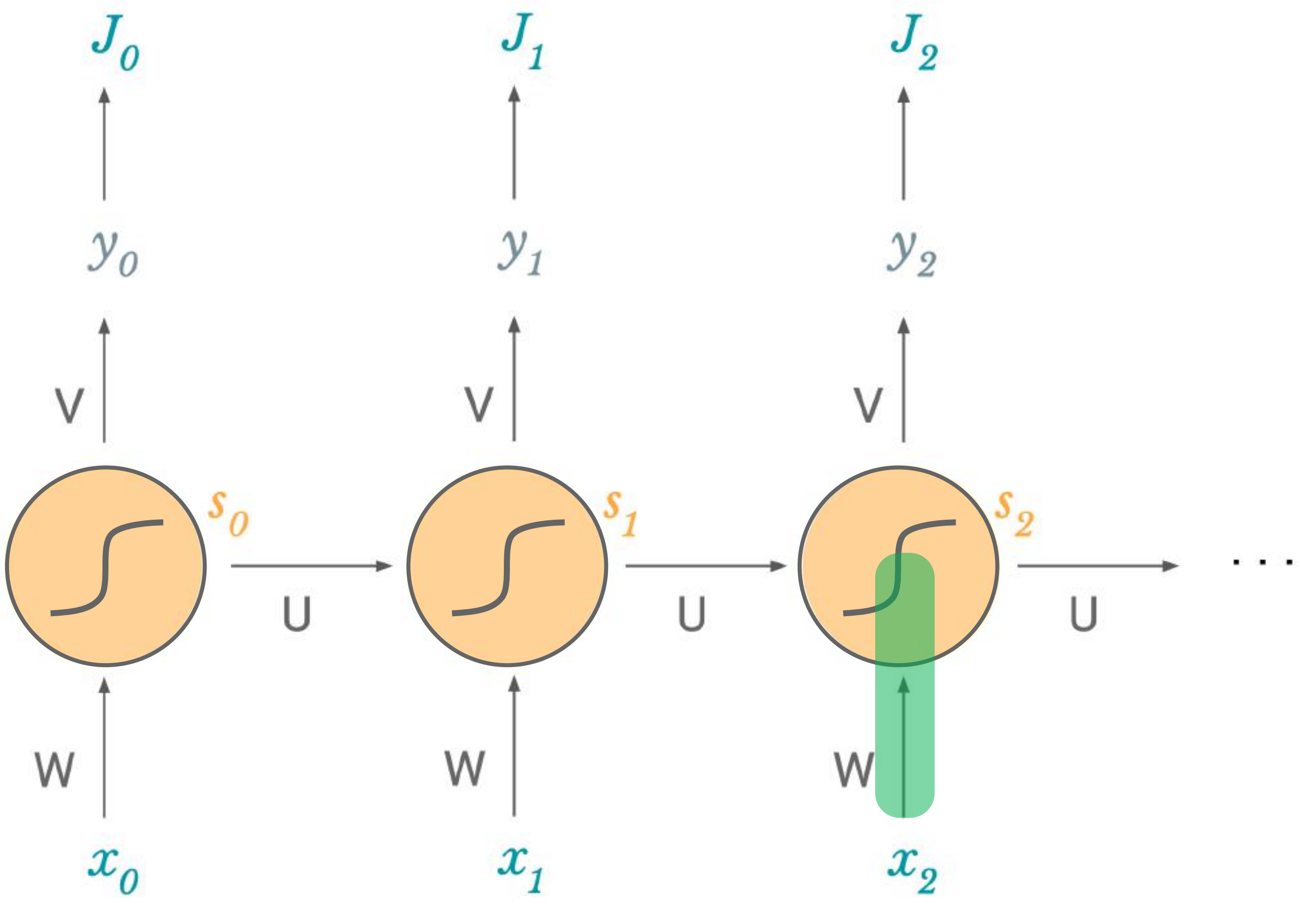
$$\frac{\partial \mathbf{x}_{\text{out}}[t]}{\partial \mathbf{x}_{\text{in}}[0]} = \frac{\partial \mathbf{x}_{\text{out}}[t]}{\partial \mathbf{h}_T} \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_{T-1}} \cdots \frac{\partial \mathbf{h}_1}{\partial \mathbf{h}_0} \frac{\partial \mathbf{h}_0}{\partial \mathbf{x}_{\text{in}}[0]}$$

- 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**

Let's try it out for W with the **chain rule:**

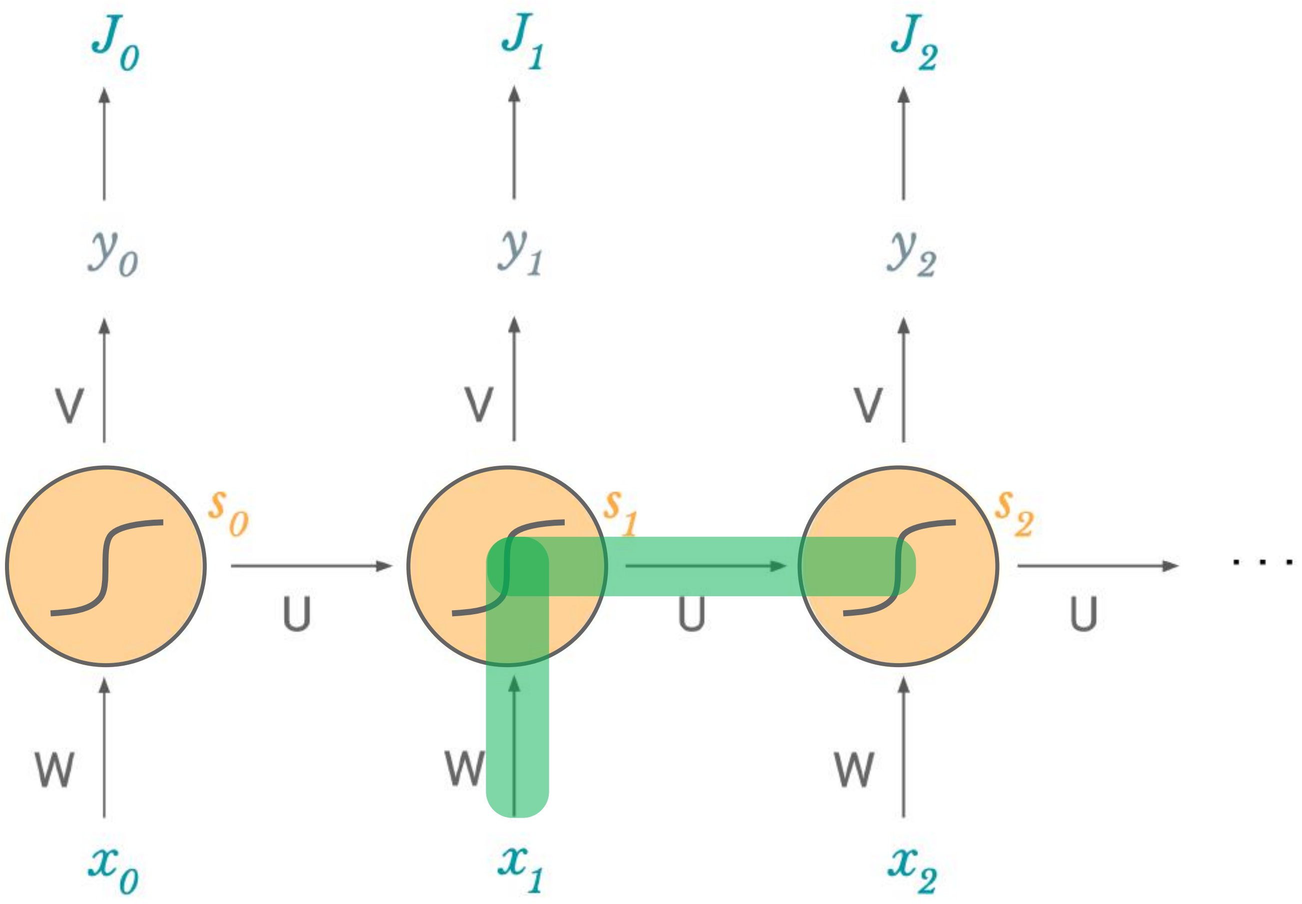


Let's try it out for W with the **chain rule:**



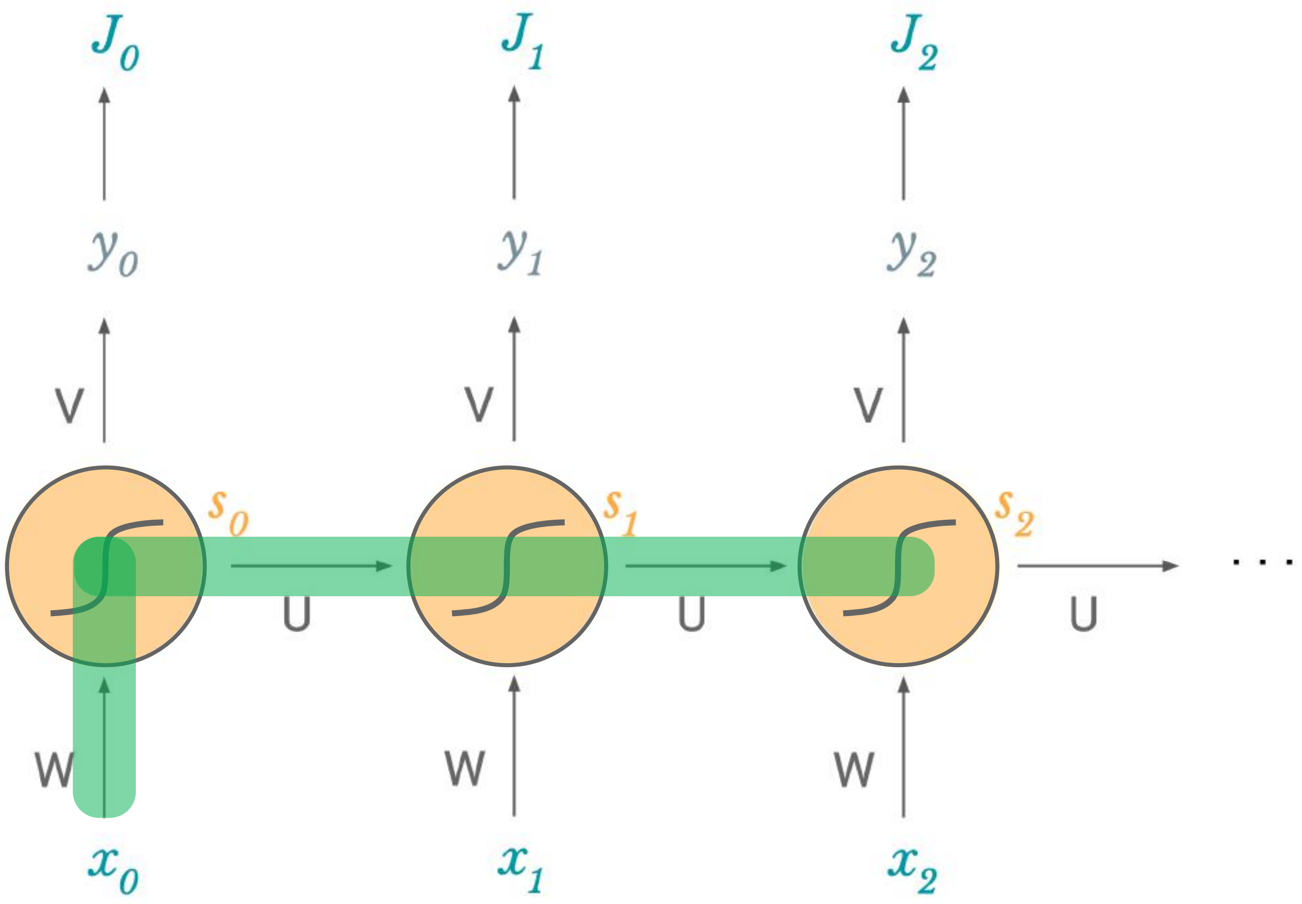
$$\frac{\partial s_2}{\partial W}$$

Let's try it out for W with the **chain rule:**



$$\frac{\partial s_2}{\partial W} + \frac{\partial s_2}{\partial s_1}\frac{\partial s_1}{\partial W}$$

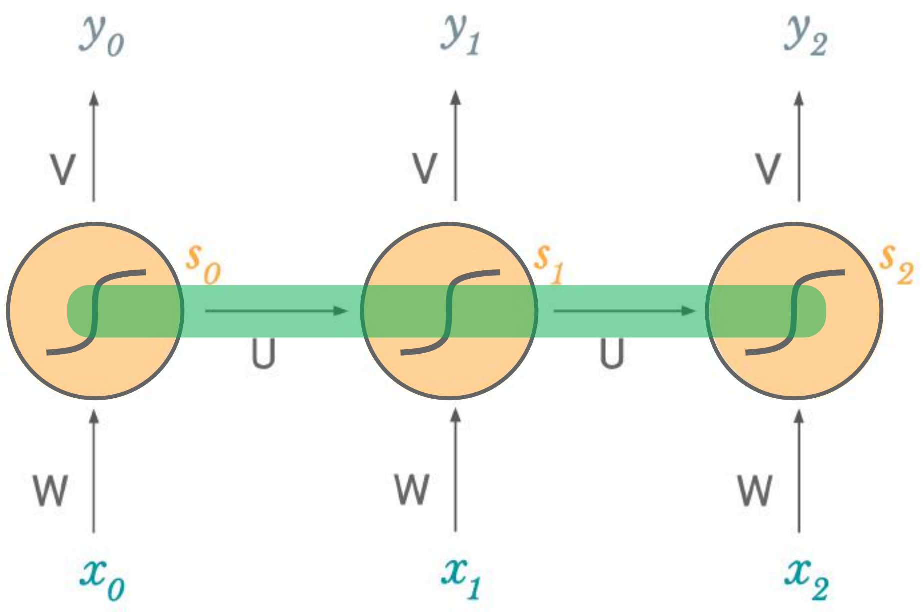
Let's try it out for W with the **chain rule:**



$$\frac{\partial s_2}{\partial W} + \frac{\partial s_2}{\partial s_1}\frac{\partial s_1}{\partial W} + \frac{\partial s_2}{\partial s_0}\frac{\partial s_0}{\partial W}$$

Vanishing Gradient Problem

$$\frac{\partial J_2}{\partial W} = \sum_{k=0}^{2} \frac{\partial J_2}{\partial y_2} \frac{\partial y_2}{\partial s_2} \boxed{\frac{\partial s_2}{\partial s_k}} \frac{\partial s_k}{\partial W}$$



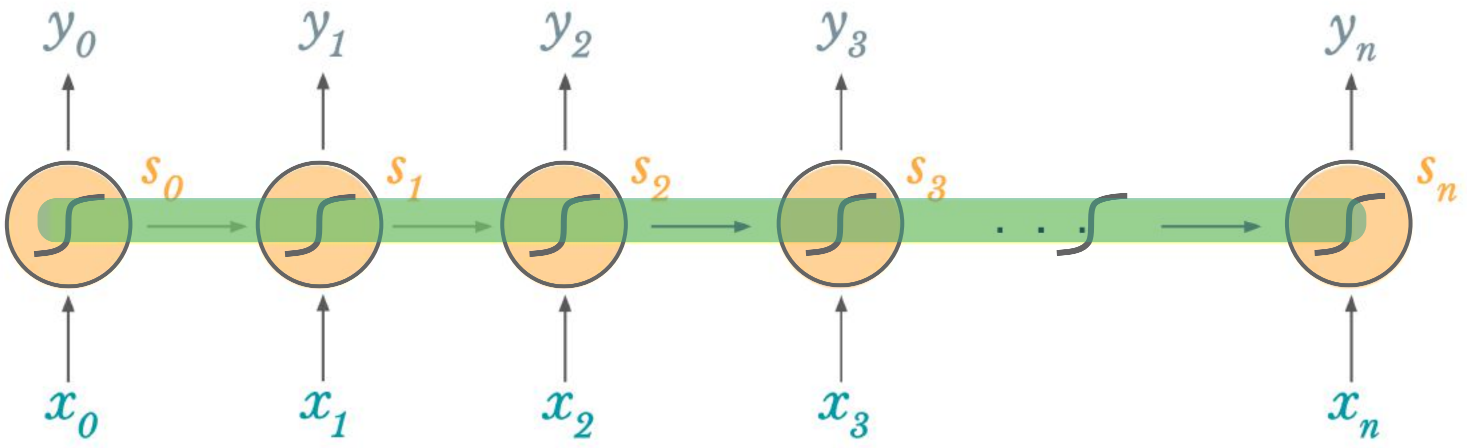
at k=0: $\boxed{\frac{\partial s_2}{\partial s_0} = \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}}$

Vanishing Gradient Problem

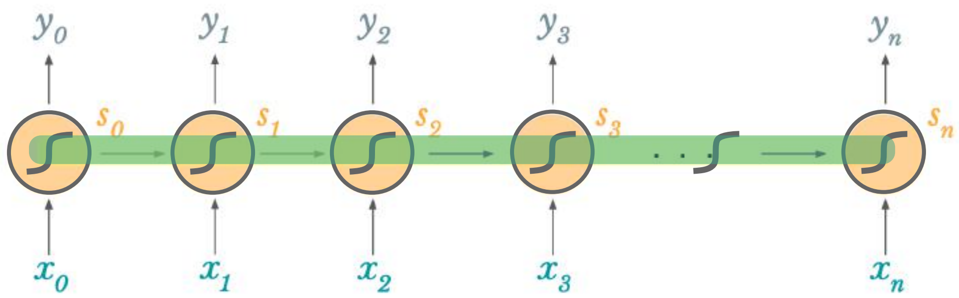
$$\frac{\partial J_n}{\partial W} = \sum_{k=0}^{n} \frac{\partial J_n}{\partial y_n} \frac{\partial y_n}{\partial s_n} \boxed{\frac{\partial s_n}{\partial s_k}} \frac{\partial s_k}{\partial W}$$

$$\boxed{\frac{\partial s_n}{\partial s_{n-1}} \frac{\partial s_{n-1}}{\partial s_{n-2}} \cdots \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}}$$

as the gap between timesteps gets bigger, this product gets longer and longer!



Vanishing Gradient Problem



what are each of these terms?

$$\frac{\partial s_n}{\partial s_{n-1}} \frac{\partial s_{n-1}}{\partial s_{n-2}} \cdots \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}$$

$$\frac{\partial s_n}{\partial s_{n-1}} = W^T diag\left[f'(W_{s_{j-1}+Ux_j})\right]$$

W = sampled from standard normal distribution = mostly < 1

f = tanh or sigmoid so $f' < 1$

we're multiplying a lot of **small numbers** together.

Vanishing Gradient Problem

we’re multiplying a lot of **small numbers** together.

so what?

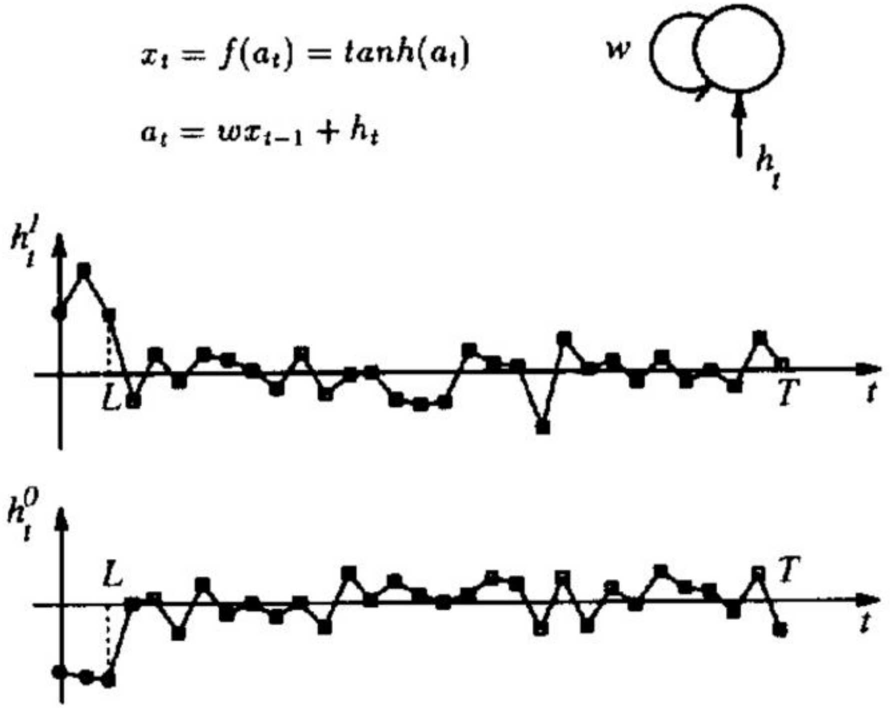
errors due to further back timesteps have increasingly **smaller gradients**.

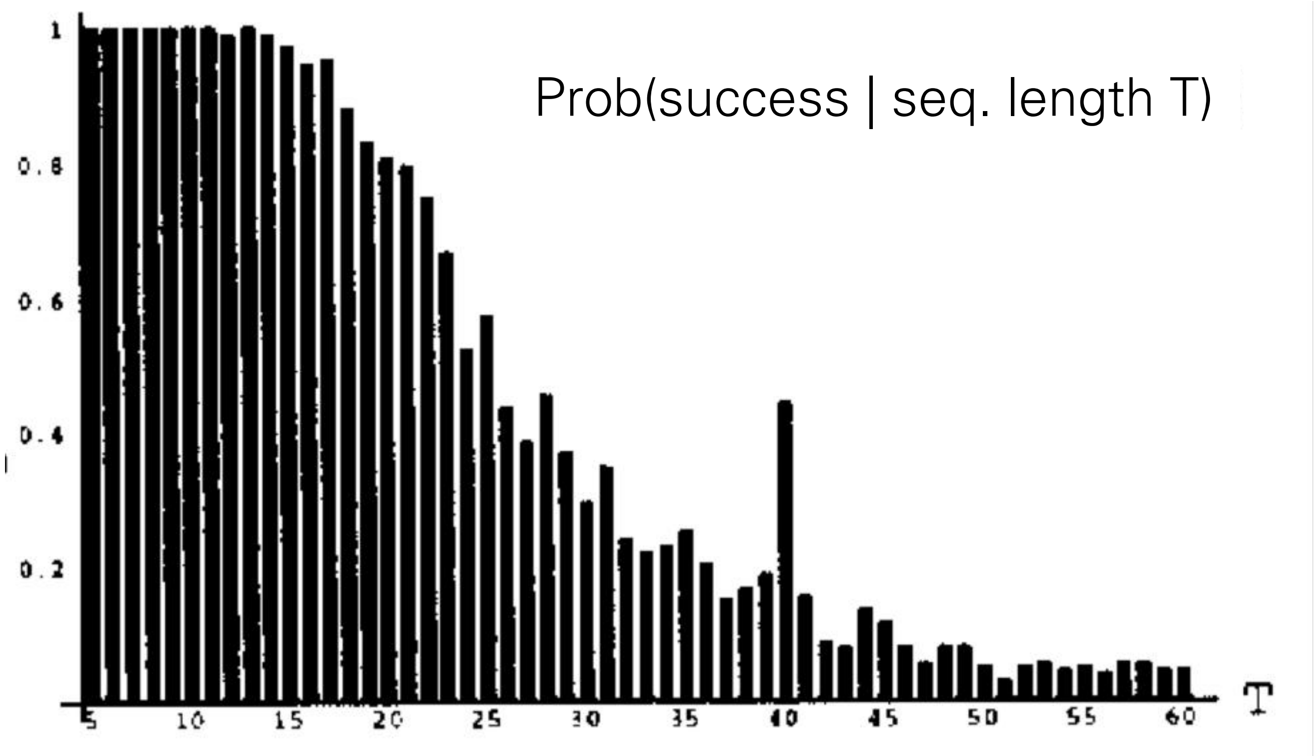
so what?

parameters become biased to **capture shorter-term** dependencies.

A Toy Example

- 2 categories of sequences
- Can the single tanh unit learn to store for T time steps 1 bit of information given by the sign of initial input?





Slide credit: Yoshua Bengio

Vanishing Gradient Problem

"In France, I had a great time and I learnt some of the _____ language."

our parameters are not trained to capture long-term dependencies, so the word we predict will mostly depend on the previous few words, not much earlier ones

Long-Term Dependencies

- The RNN gradient is a product of Jacobian matrices, each associated with a step in the forward computation. To store information robustly in a finite-dimensional state, the dynamics must be contractive [Bengio et al 1994].

$$L = L(s_T(s_{T-1}(\ldots s_{t+1}(s_t, \ldots))))$$
$$\frac{\partial L}{\partial s_t} = \frac{\partial L}{\partial s_T} \frac{\partial s_T}{\partial s_{T-1}} \cdots \frac{\partial s_{t+1}}{\partial s_t}$$

- Problems:
  - sing. values of Jacobians > 1 → **gradients explode**
  - or sing. values < → **gradients shrink & vanish**
  - or random → **variance grows exponentially**

Slide credit: Yoshua Bengio

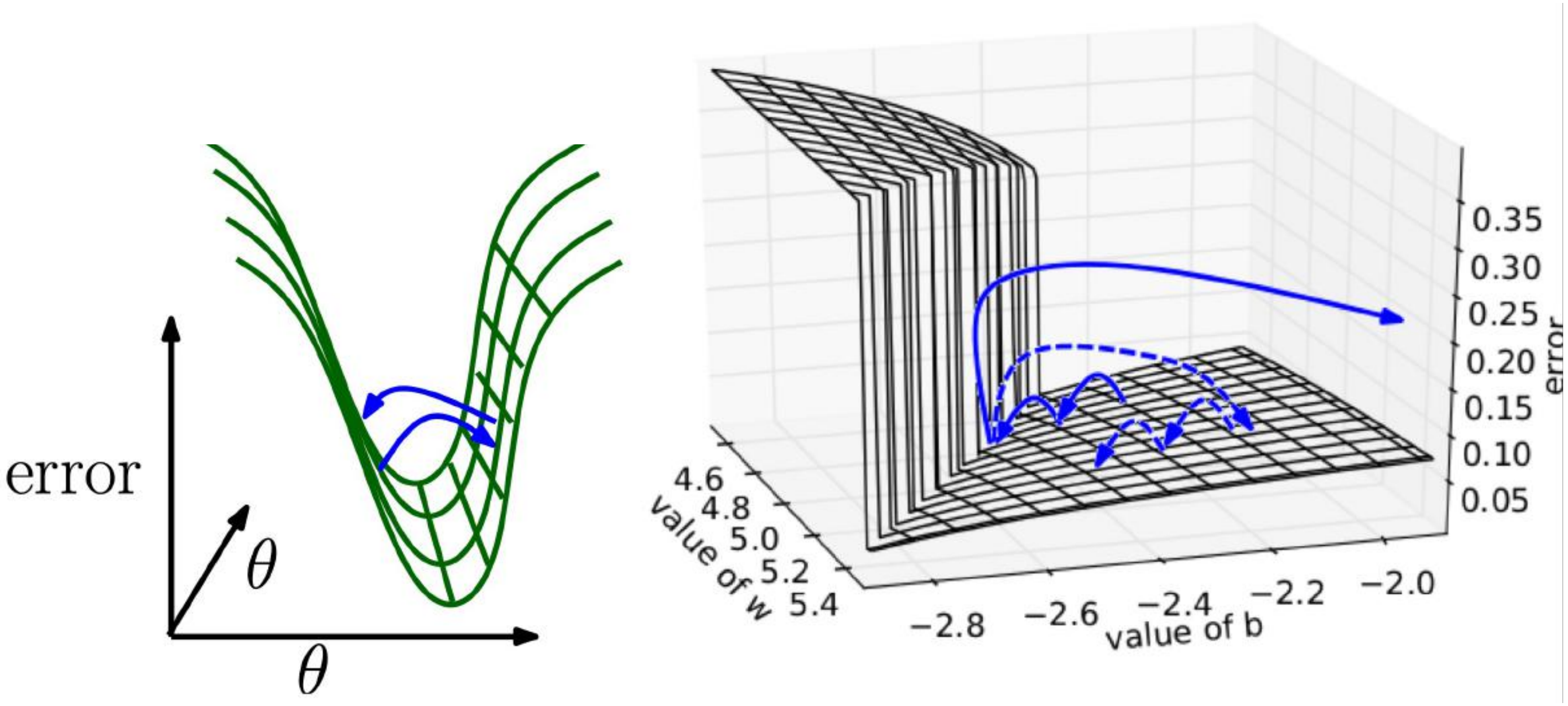
Gradient Norm Clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial error}{\partial \theta}$

if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**

$\quad \hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$

end if



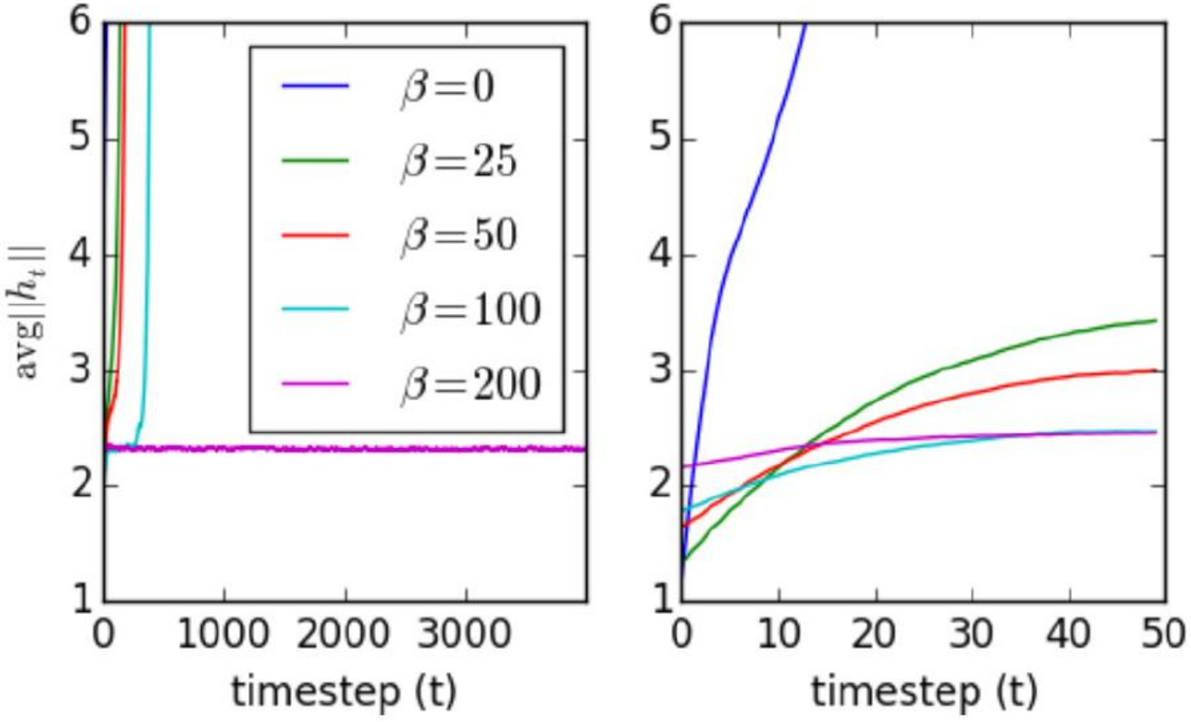
Recurrent neural network regularization. Zaremba et al., arXiv 2014.

Regularization: Norm-stabilizer

- Stabilize the activations of RNNs by penalizing the squared distance between successive hidden states' norms

$$\beta \frac{1}{T} \sum_{t=1}^{T} (\|h_t\|_2 - \|h_{t-1}\|_2)^2$$

- Enforce the norms of the hidden layer activations approximately constant across time



Regularizing RNNs by Stabilizing Activations. Krueger and Memisevic, ICLR 2016

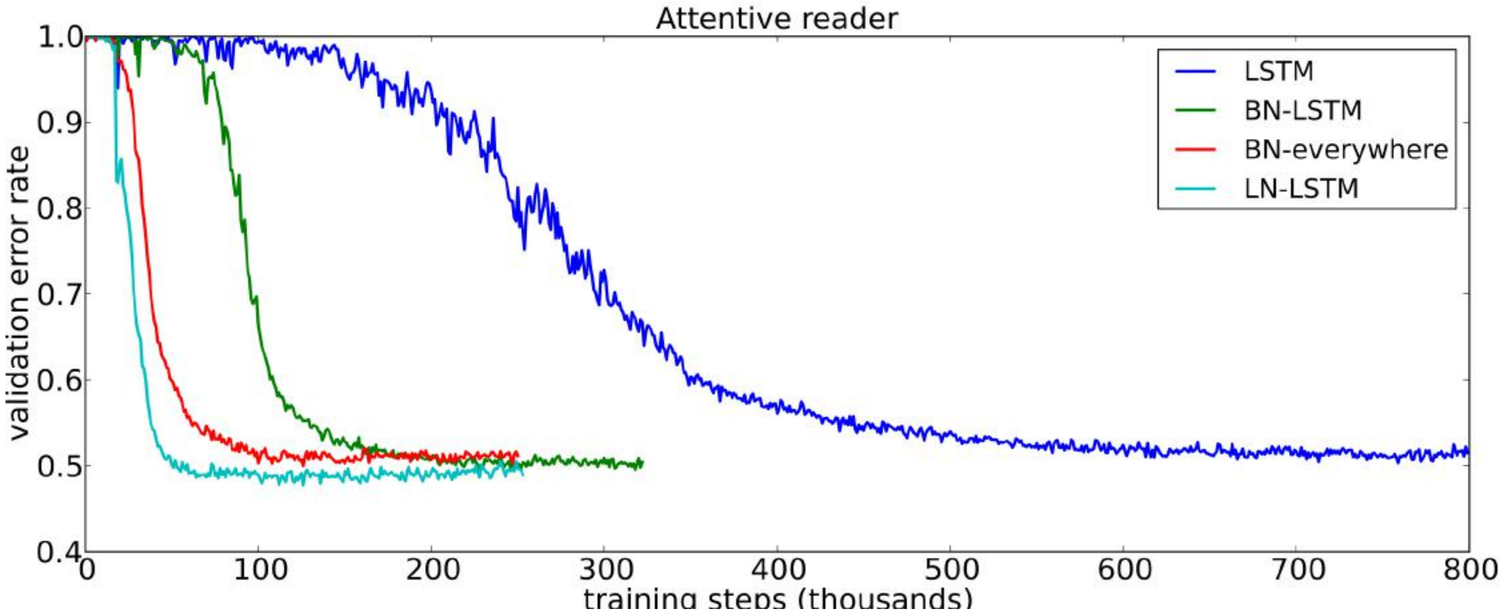
Regularization: Layer Normalization

- Similar to batch normalization
- Computes the normalization statistics separately at each time step
- Effective for stabilizing the hidden state dynamics in RNNs
- Reduces training time

$$\mathbf{h}^t = f\left[\frac{\mathbf{g}}{\sigma^t} \odot \left(\mathbf{a}^t - \mu^t\right) + \mathbf{b}\right]$$

$$\mu^t = \frac{1}{H}\sum_{i=1}^{H} a_i^t$$

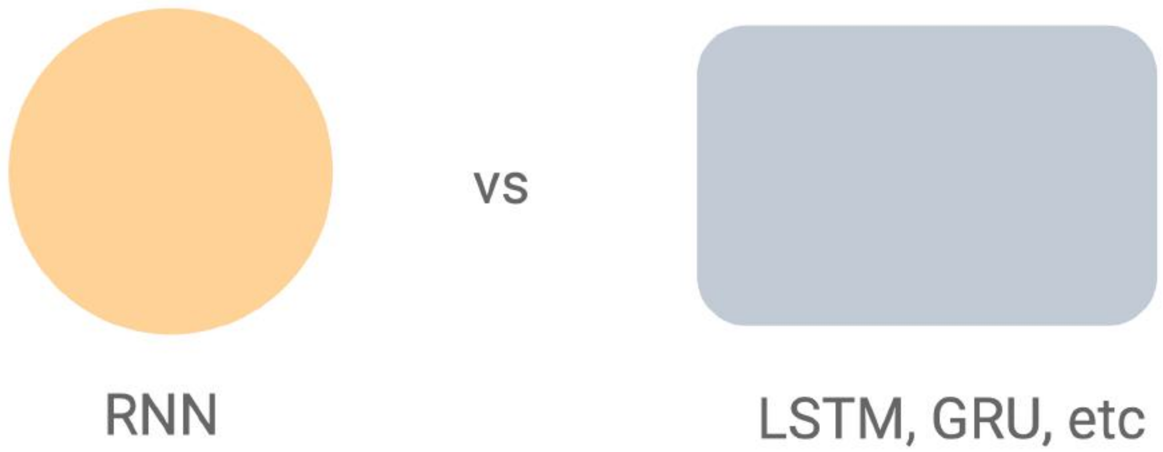
$$\sigma^t = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^t - \mu^t\right)^2}$$



Layer Normalization [Ba, Kiros & Hinton, 2016]

Gated Cells

- rather each node being just a simple RNN cell, make each node a more **complex unit with gates** controlling what information is passed through

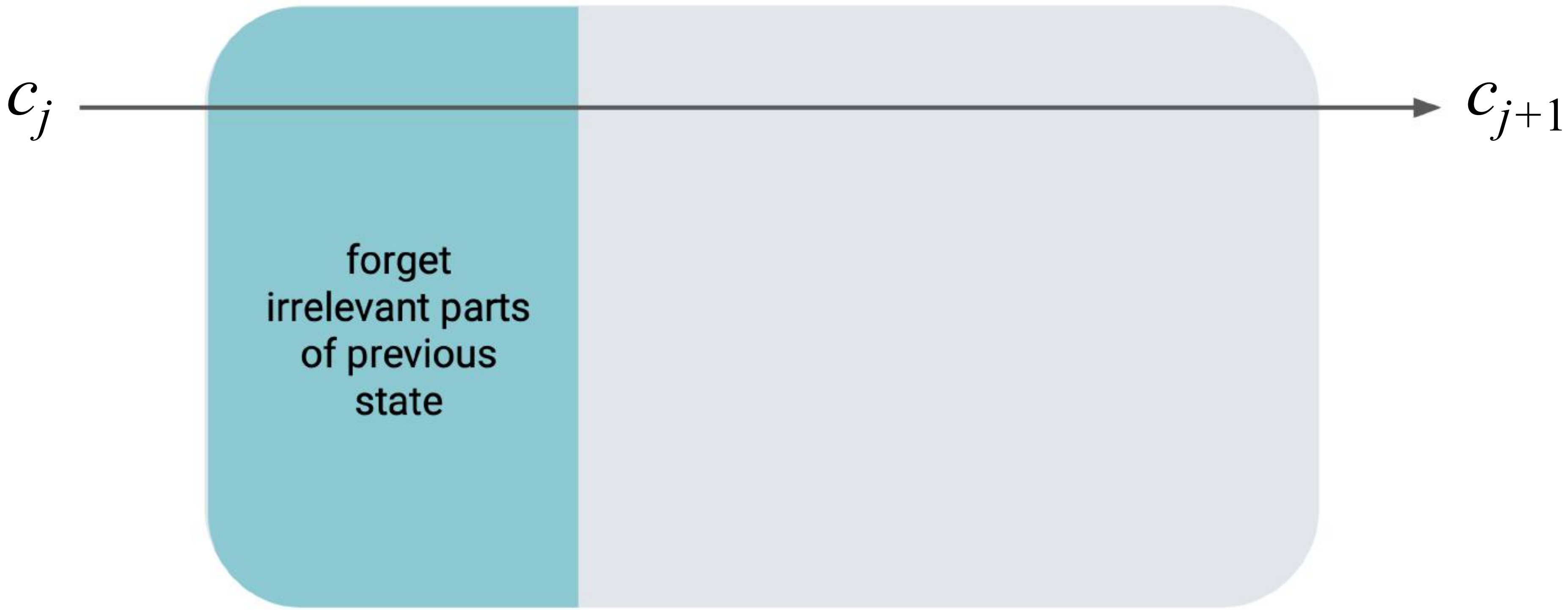


Long short term memory cells are able to keep track of information throughout many timesteps.

Long Short-Term Memory (LSTM)

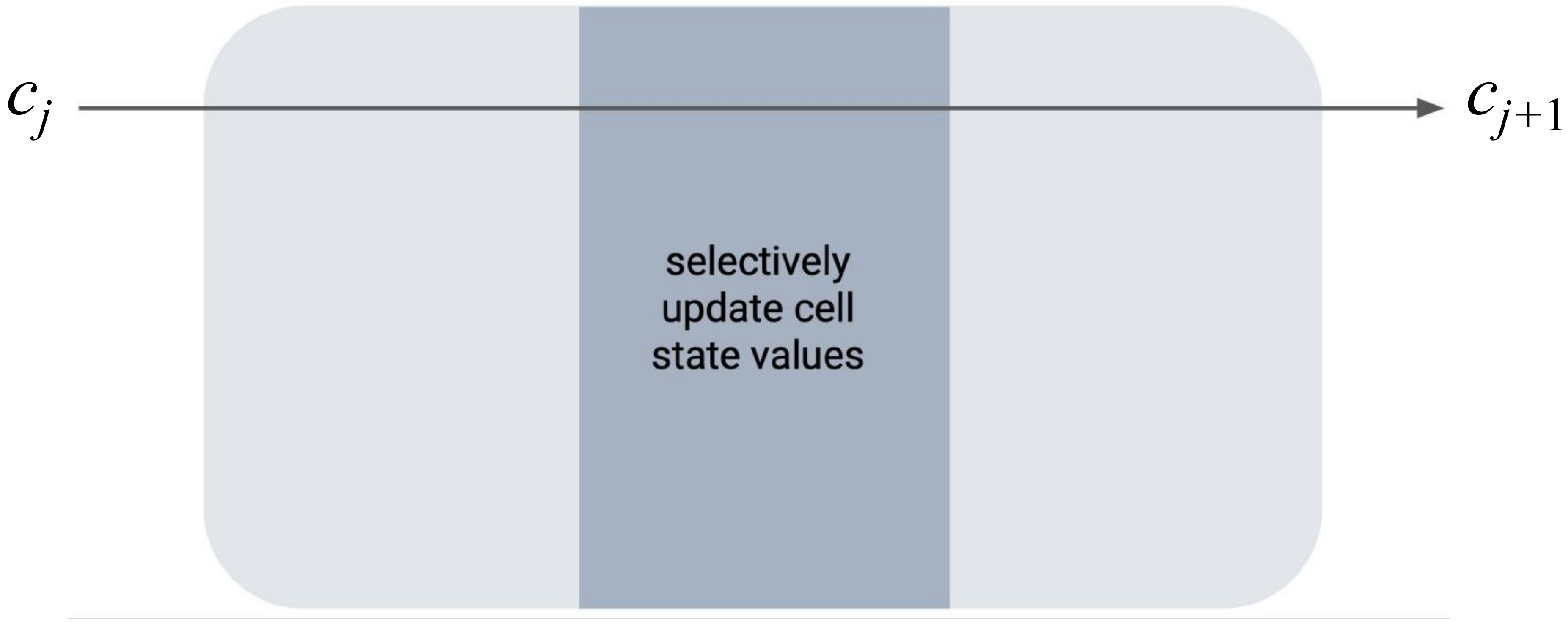
Long Short-Term Memory [Hochreiter et al., 1997]

Long Short-Term Memory (LSTM)



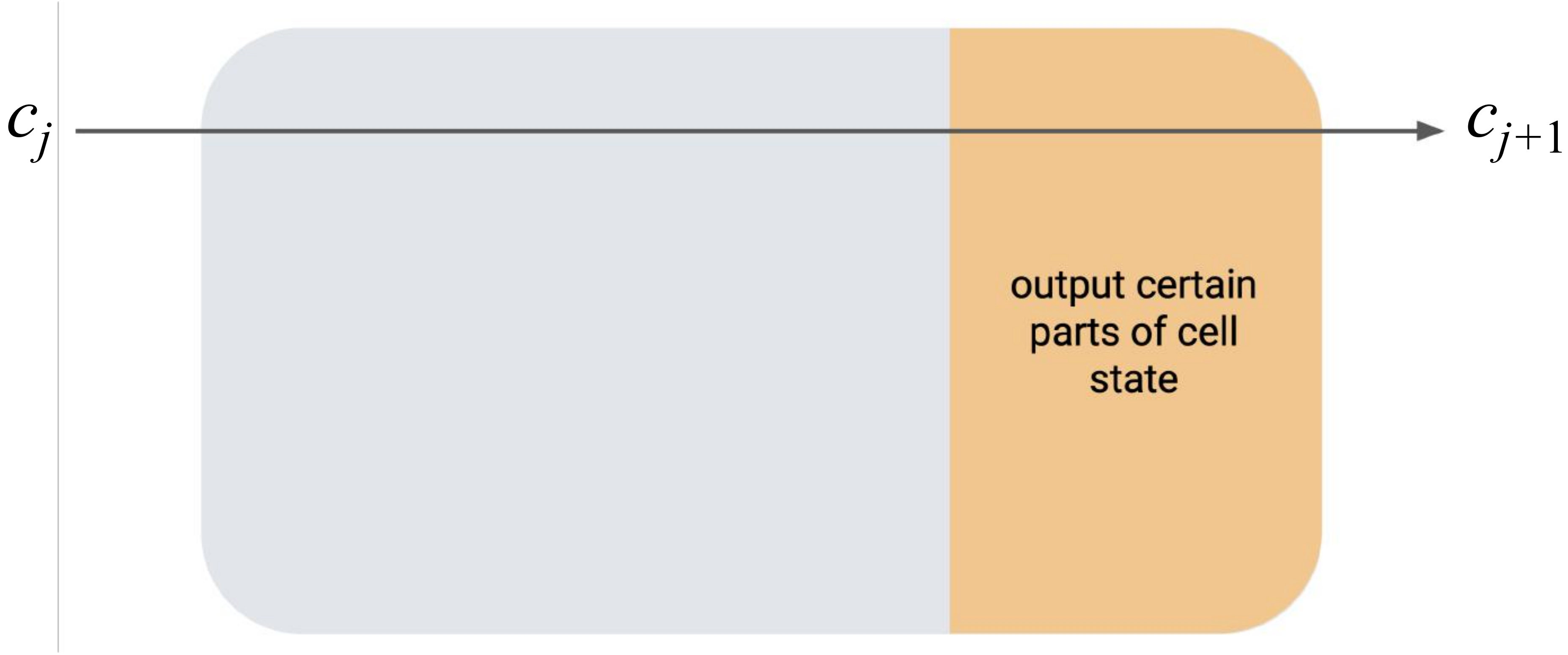
[Long Short-Term Memory [Hochreiter et al., 1997]]

Long Short-Term Memory (LSTM)



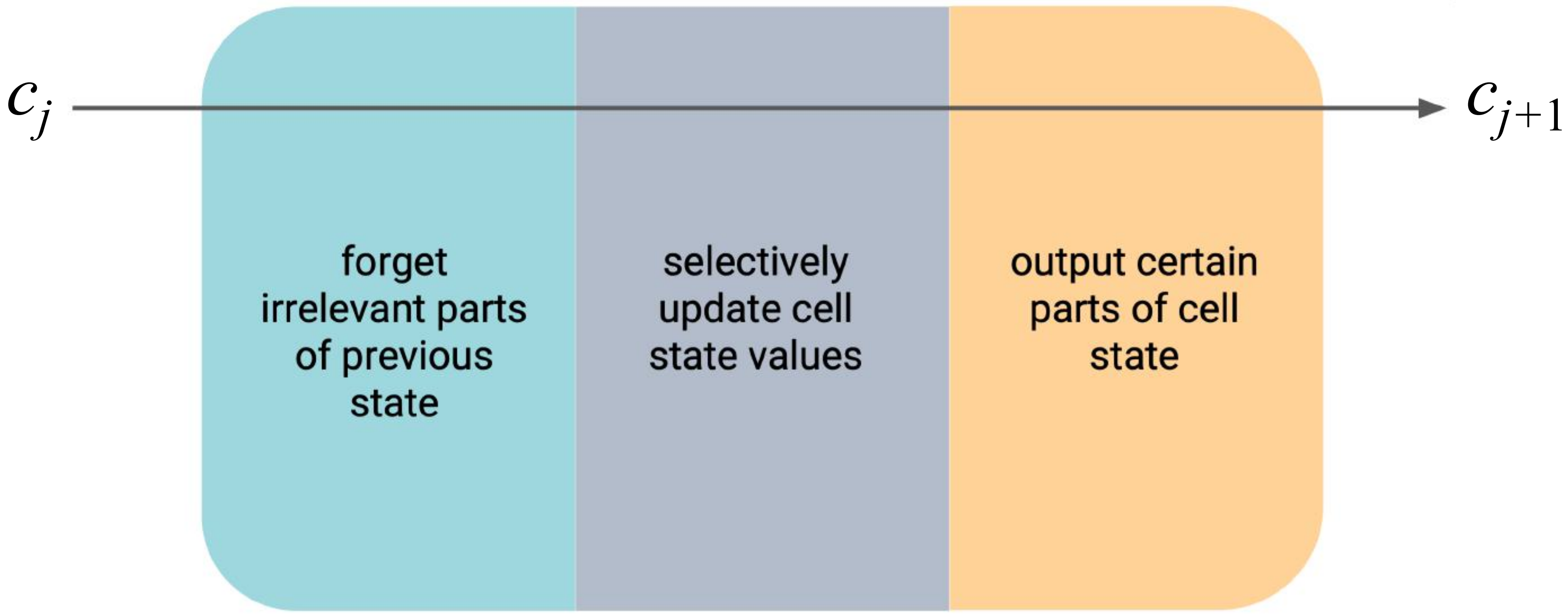
Long Short-Term Memory [Hochreiter et al., 1997]

Long Short-Term Memory (LSTM)



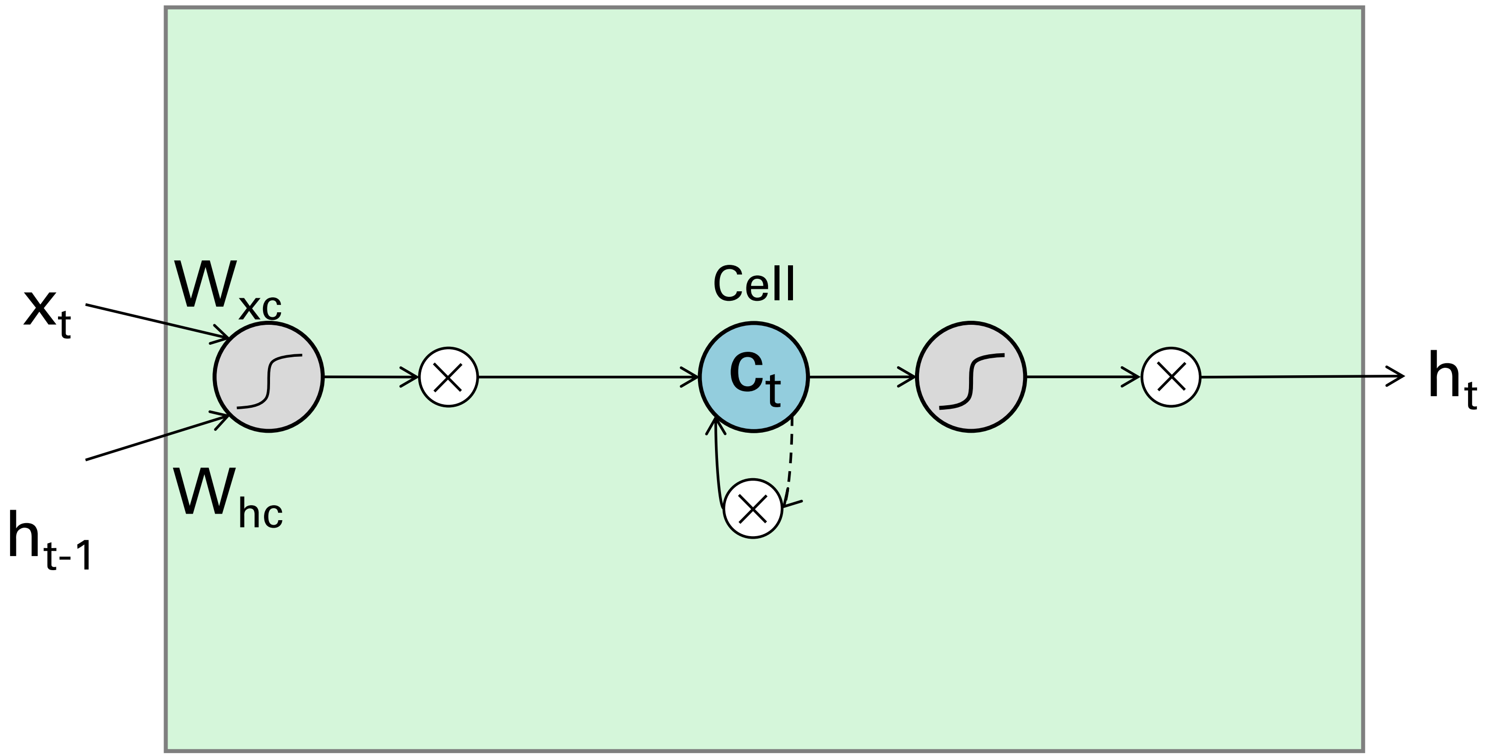
Long Short-Term Memory [Hochreiter et al., 1997]

Long Short-Term Memory (LSTM)



Long Short-Term Memory [Hochreiter et al., 1997]

The LSTM Idea

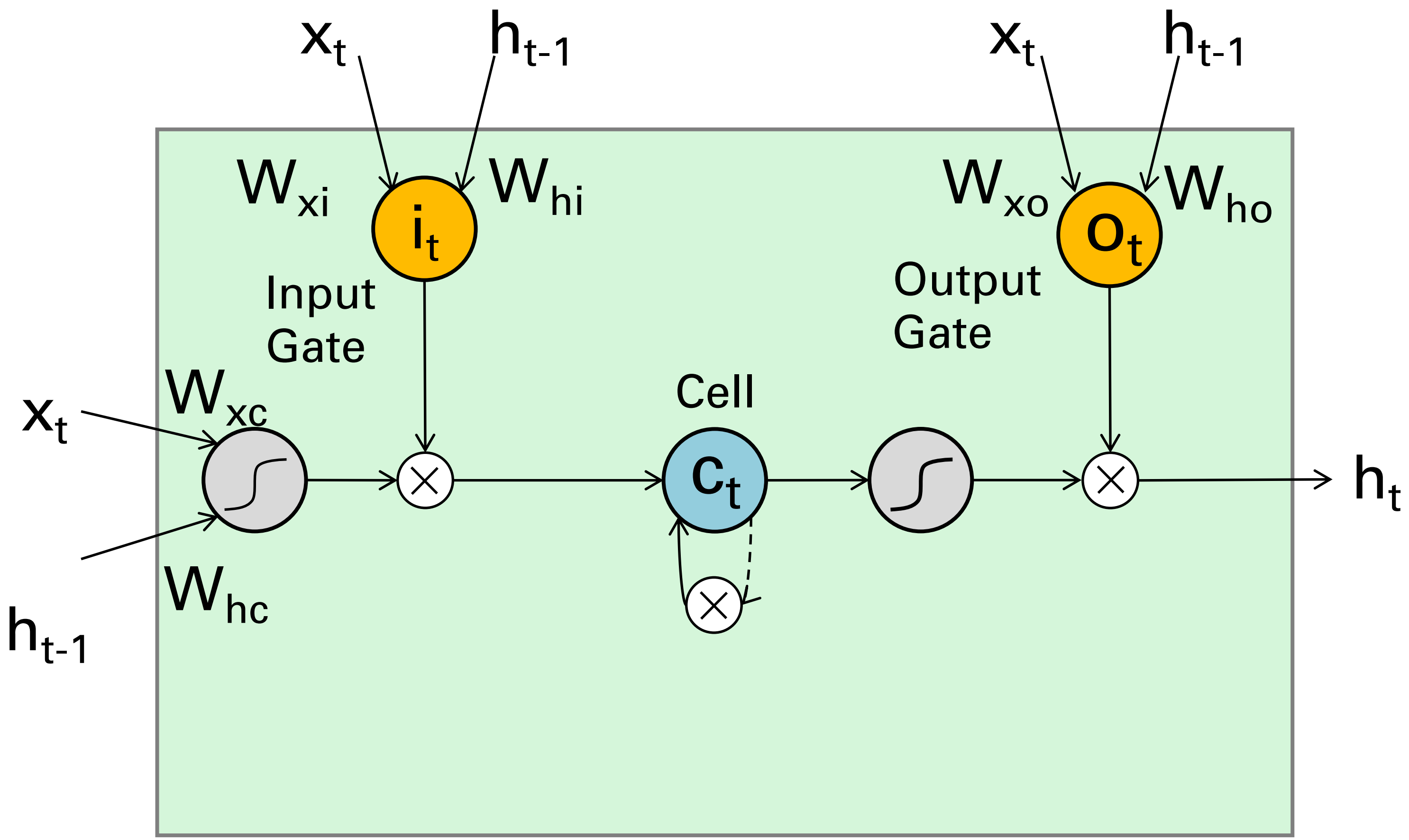


* Dashed line indicates time-lag

$$c_t = c_{t-1} + \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$h_t = \tanh c_t$$

The Original LSTM Cell



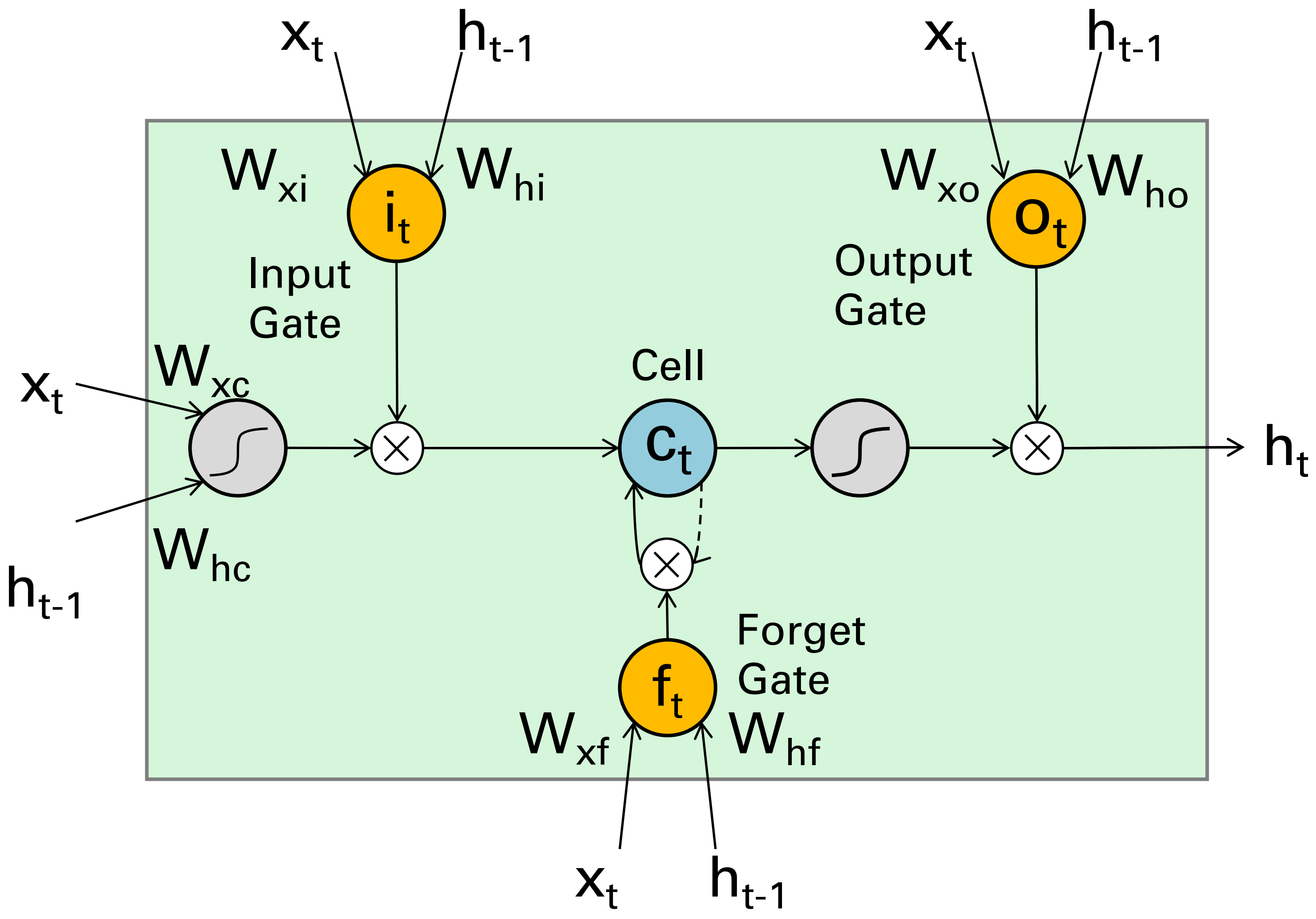
$$c_t = c_{t-1} + i_t \otimes \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$h_t = o_t \otimes \tanh c_t$$

$$i_t = \sigma \left(W_i \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_i \right)$$

Similarly for o_t

The Popular LSTM Cell



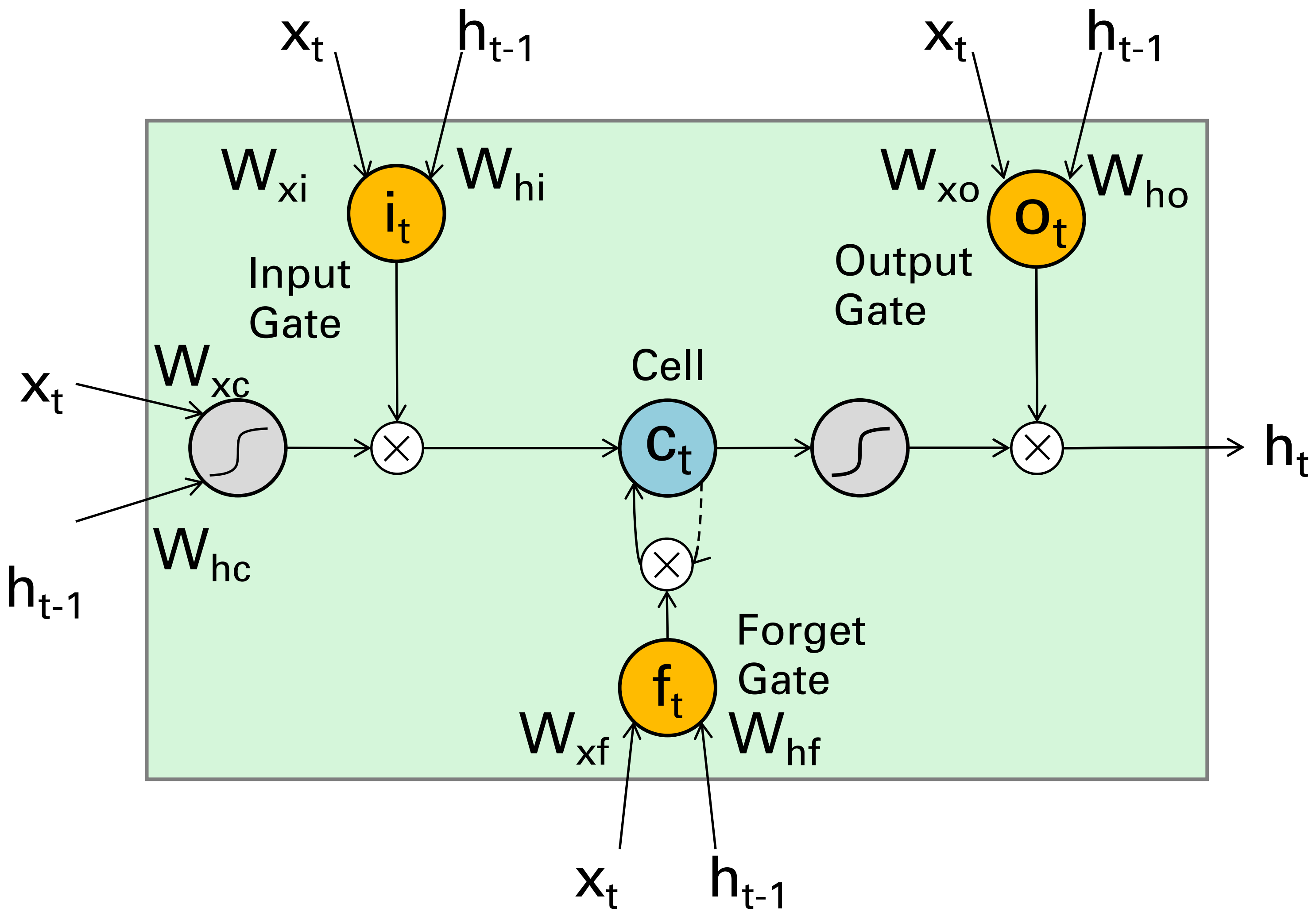
$$i_t = \sigma\left(W_i \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_i\right)$$

$$c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$f_t = \sigma\left(W_f \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f\right)$$

$$h_t = o_t \otimes \tanh c_t$$

The Popular LSTM Cell



$$i_t = \sigma\left(W_i \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_i \right)$$

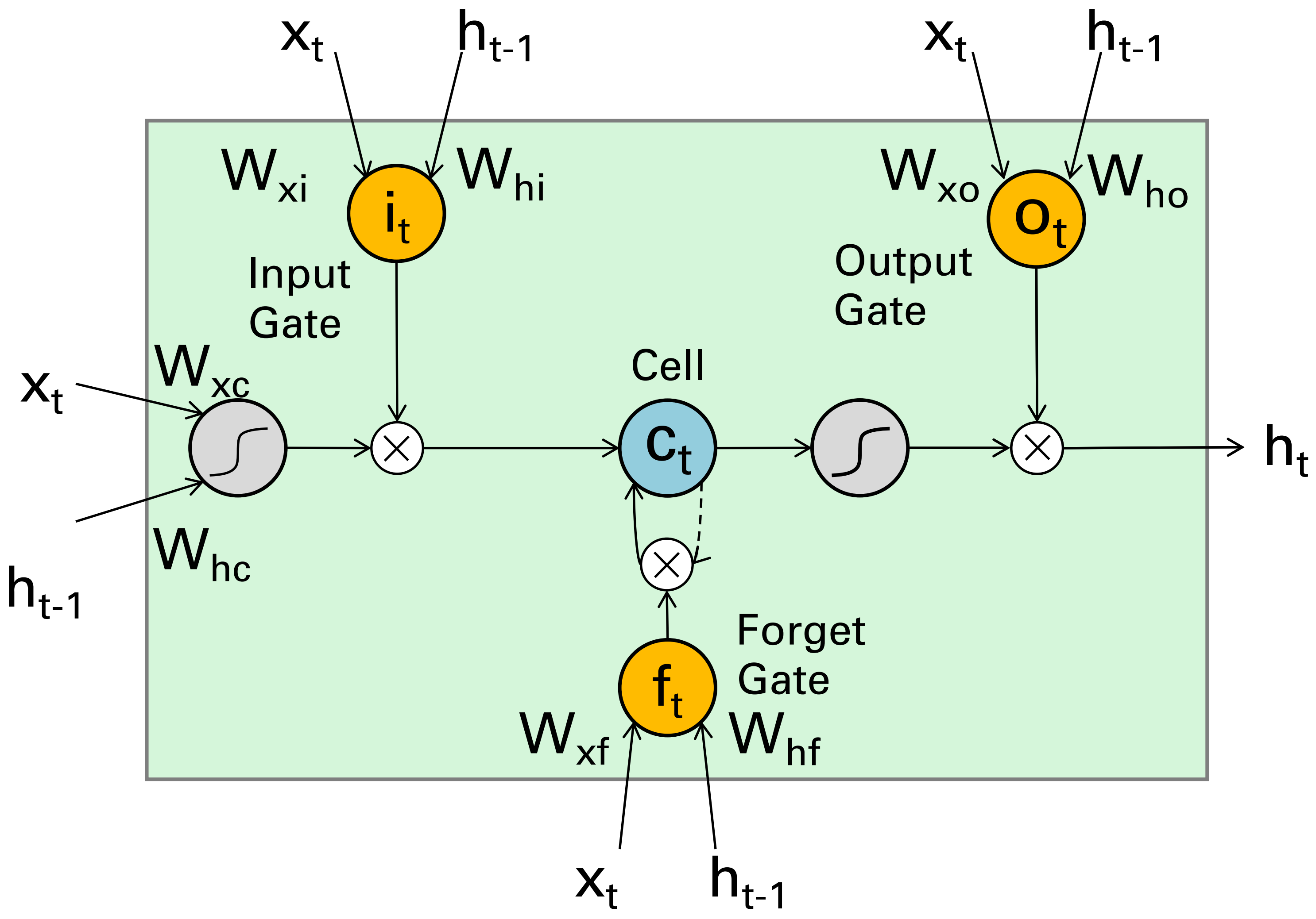
$$c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$\boxed{f_t = \sigma\left(W_f \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f \right)}$$

$$h_t = o_t \otimes \tanh c_t$$

forget gate decides what information is going to be thrown away from the cell state

The Popular LSTM Cell



$$i_t = \sigma\left(W_i\begin{pmatrix}x_t\\ h_{t-1}\end{pmatrix}+b_i\right)$$

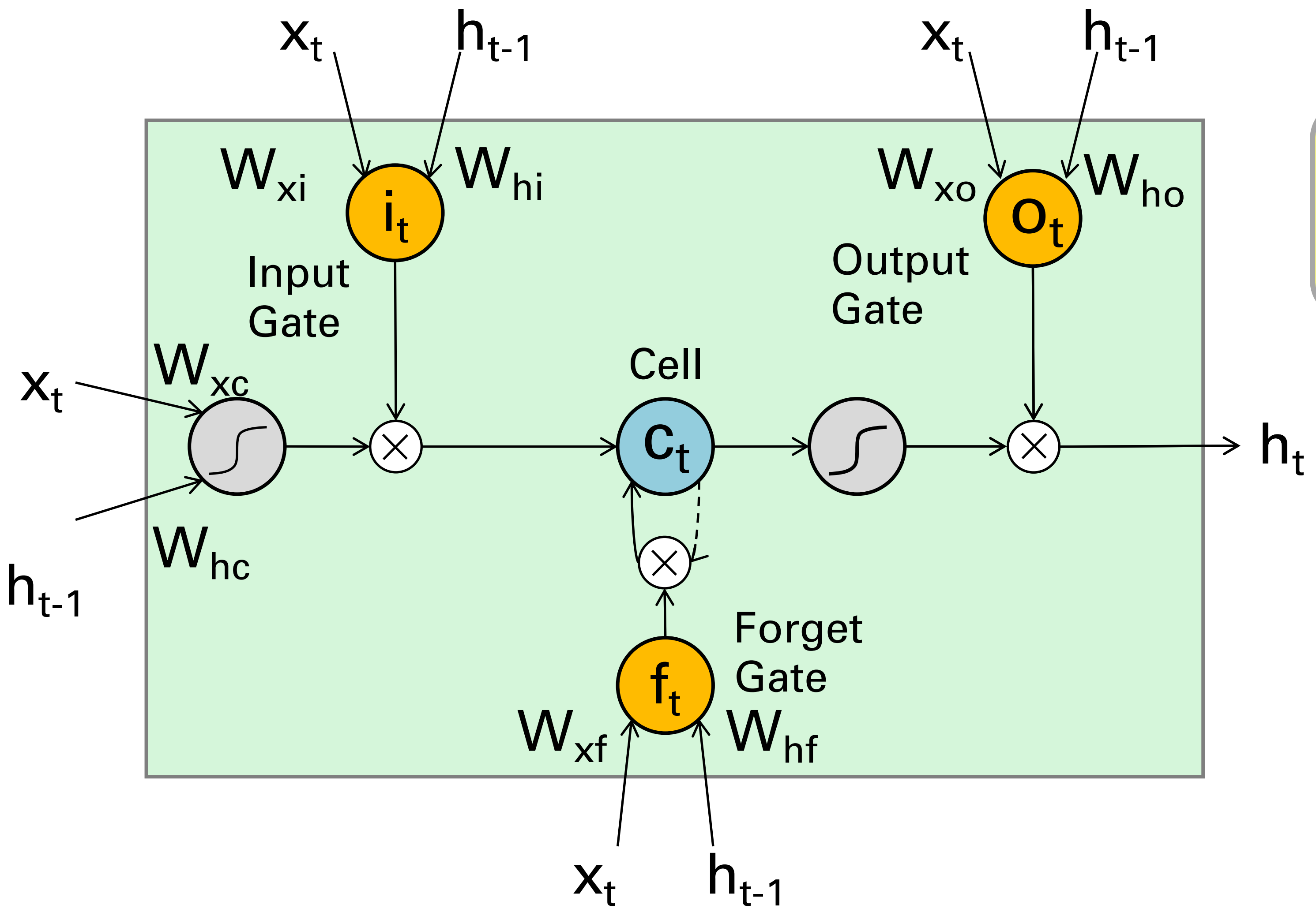
$$c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh W\begin{pmatrix}x_t\\ h_{t-1}\end{pmatrix}$$

$$f_t = \sigma\left(W_f\begin{pmatrix}x_t\\ h_{t-1}\end{pmatrix}+b_f\right)$$

$$h_t = o_t \otimes \tanh c_t$$

input gate and **a tanh layer** decides what information is going to be stored in the cell state

The Popular LSTM Cell



$$i_t = \sigma\left(W_i\begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_i\right)$$

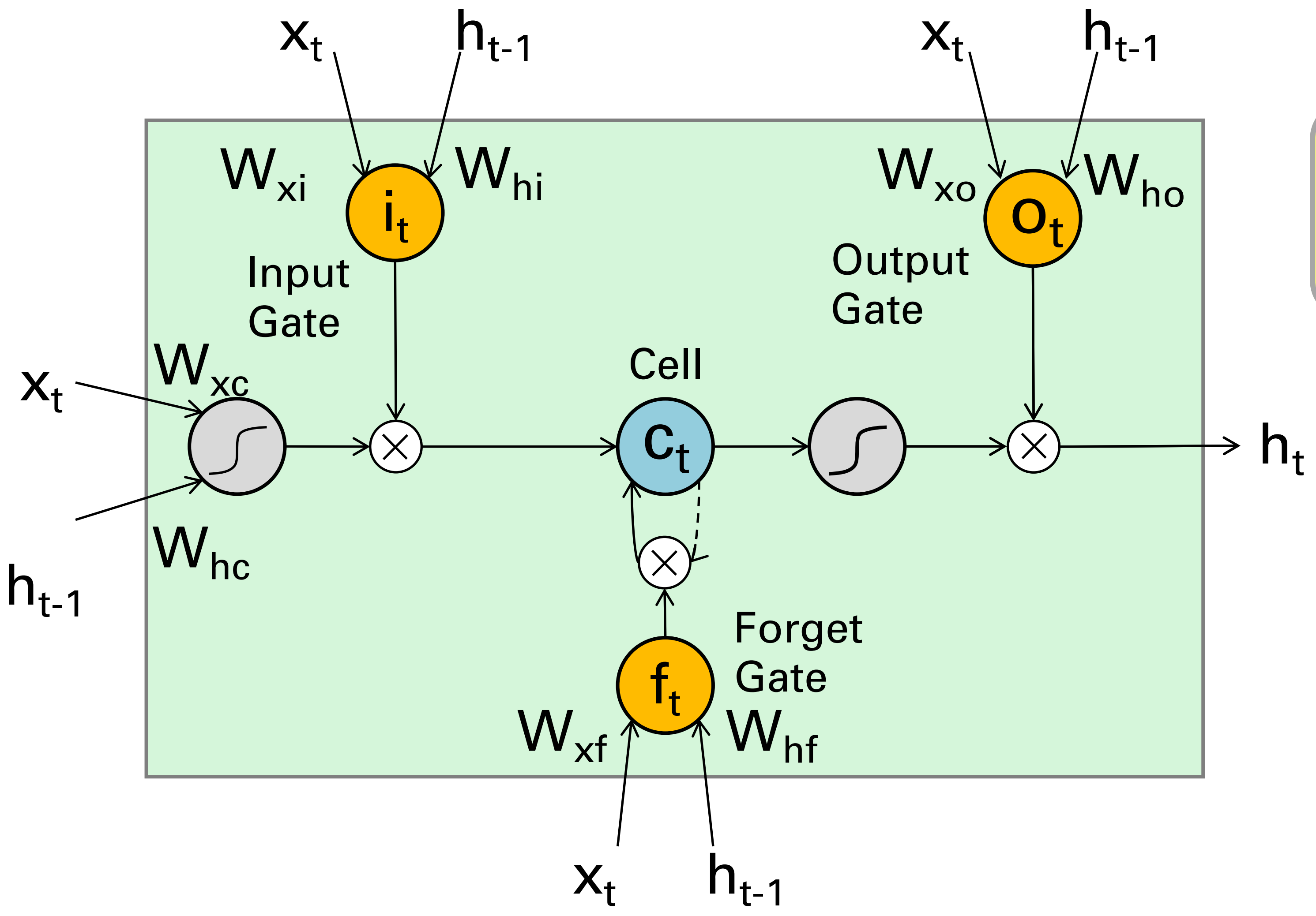
$$c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh W\begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$f_t = \sigma\left(W_f\begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f\right)$$

$$h_t = o_t \otimes \tanh c_t$$

Update the old cell state with the new one.

The Popular LSTM Cell

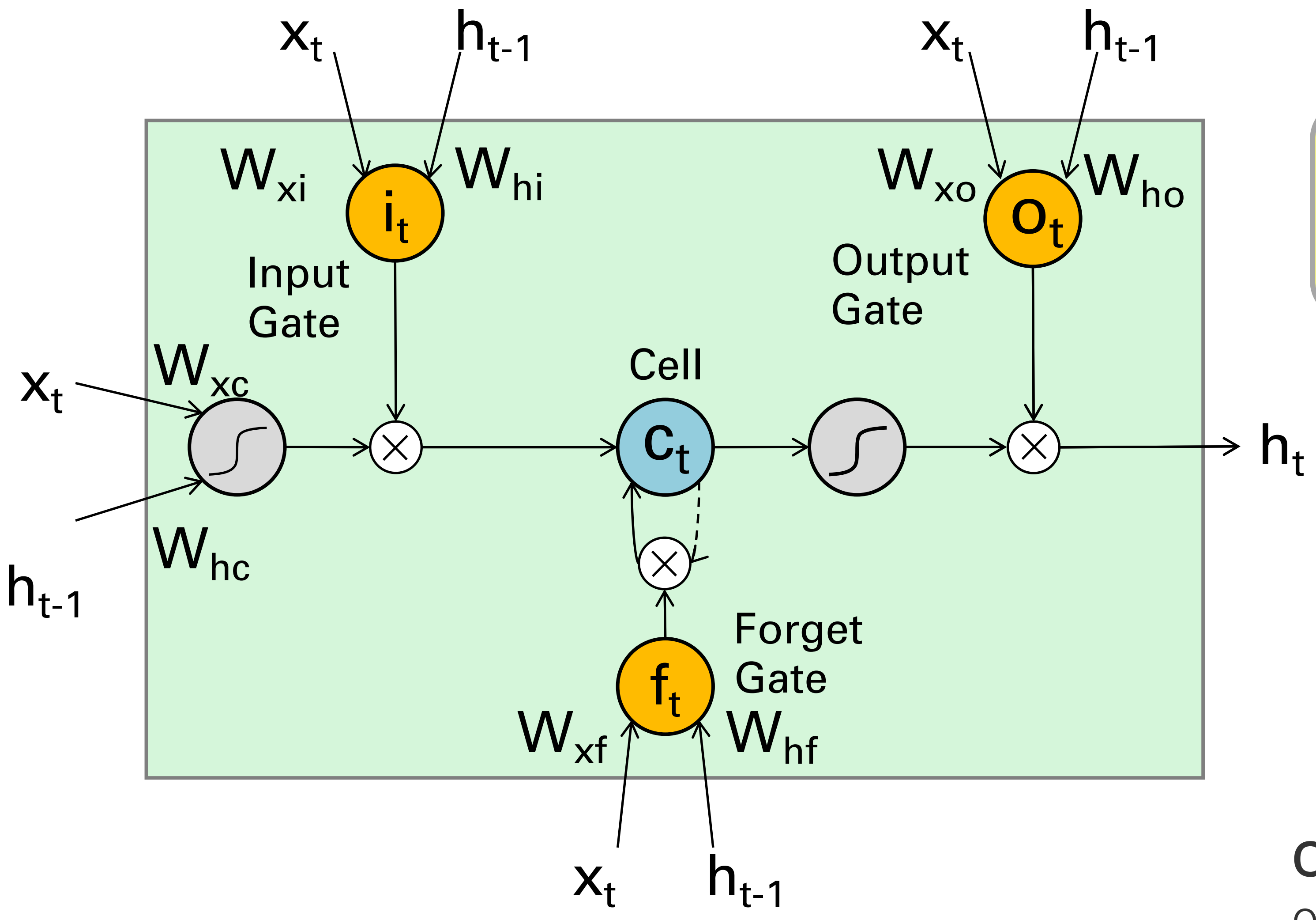


$$i_t = \sigma\left(W_i\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix}+b_i\right)$$

$$c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh W\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix}$$

input gate	forget gate	behavior
0	1	remember the previous value
1	1	add to the previous value
0	0	erase the value
1	0	overwrite the value

The Popular LSTM Cell



$$i_t = \sigma\left(W_i \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_i \right)$$

$$\boxed{c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}}$$

$$f_t = \sigma\left(W_f \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f \right)$$

$$h_t = o_t \otimes \tanh c_t$$

$$o_i = \sigma\left(W_o \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_o \right)$$

Output gate decides what is going to be outputted. The final output is based on cell state and output of sigmoid gate.

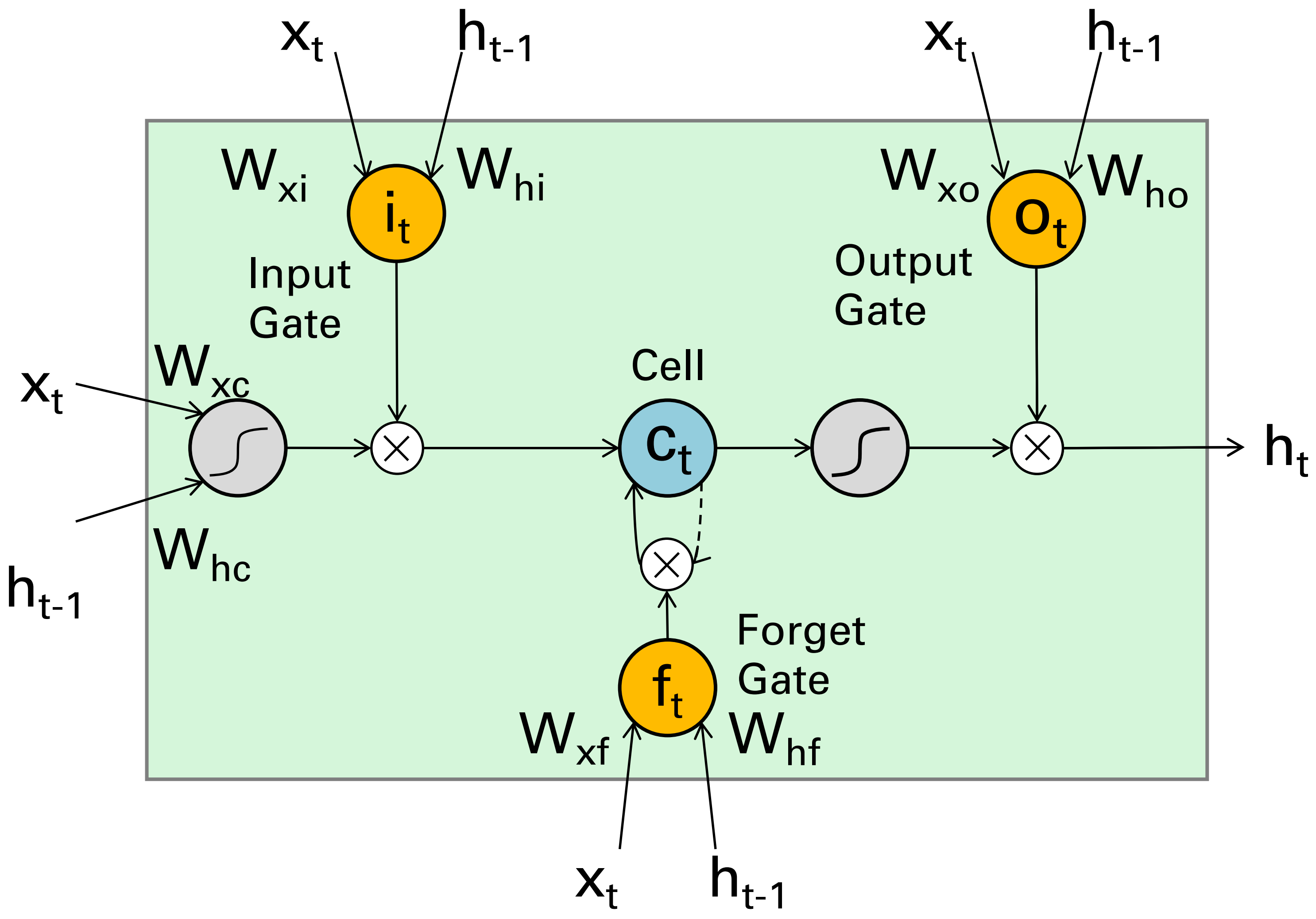
LSTM – Forward/Backward

[Illustrated LSTM Forward and Backward Pass](http://arunmallya.github.io/writeups/nn/lstm/index.html)

http://arunmallya.github.io/writeups/nn/lstm/index.html

LSTM variants

The Popular LSTM Cell



$$f_t = \sigma\left(W_f \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f \right)$$

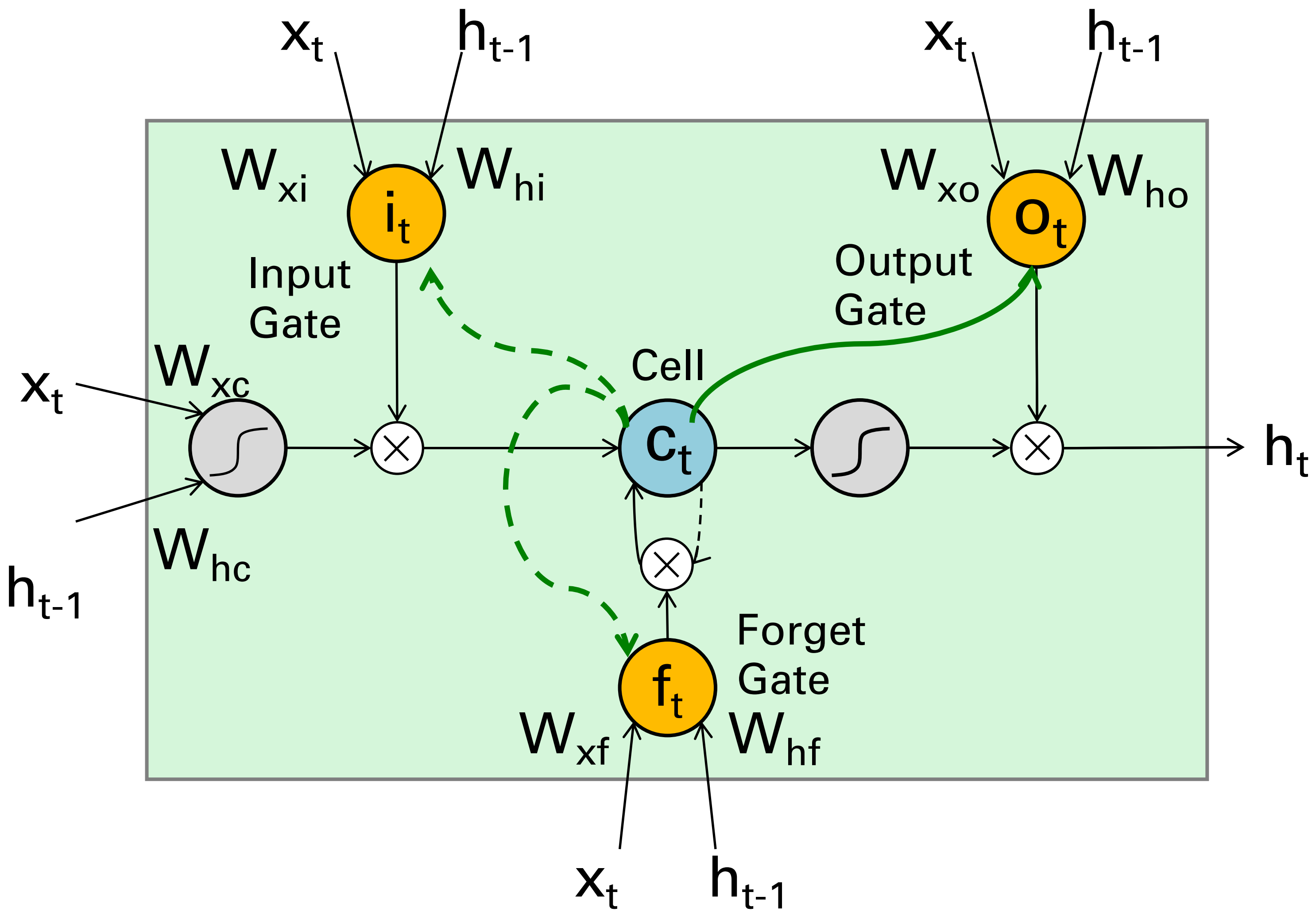
Similarly for i_t, o_t

$$c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$h_t = o_t \otimes \tanh c_t$$

* Dashed line indicates time-lag

Extension I: Peephole LSTM



$$f_t = \sigma\left(W_f \begin{pmatrix} x_t \\ h_{t-1} \\ c_{t-1} \end{pmatrix} + b_f \right)$$

Similarly for i_t, o_t (uses c_t)

$$c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$h_t = o_t \otimes \tanh c_t$$

- Add **peephole connections**.
- All gate layers look at the cell state!

* Dashed line indicates time-lag

Gated Recurrent Unit

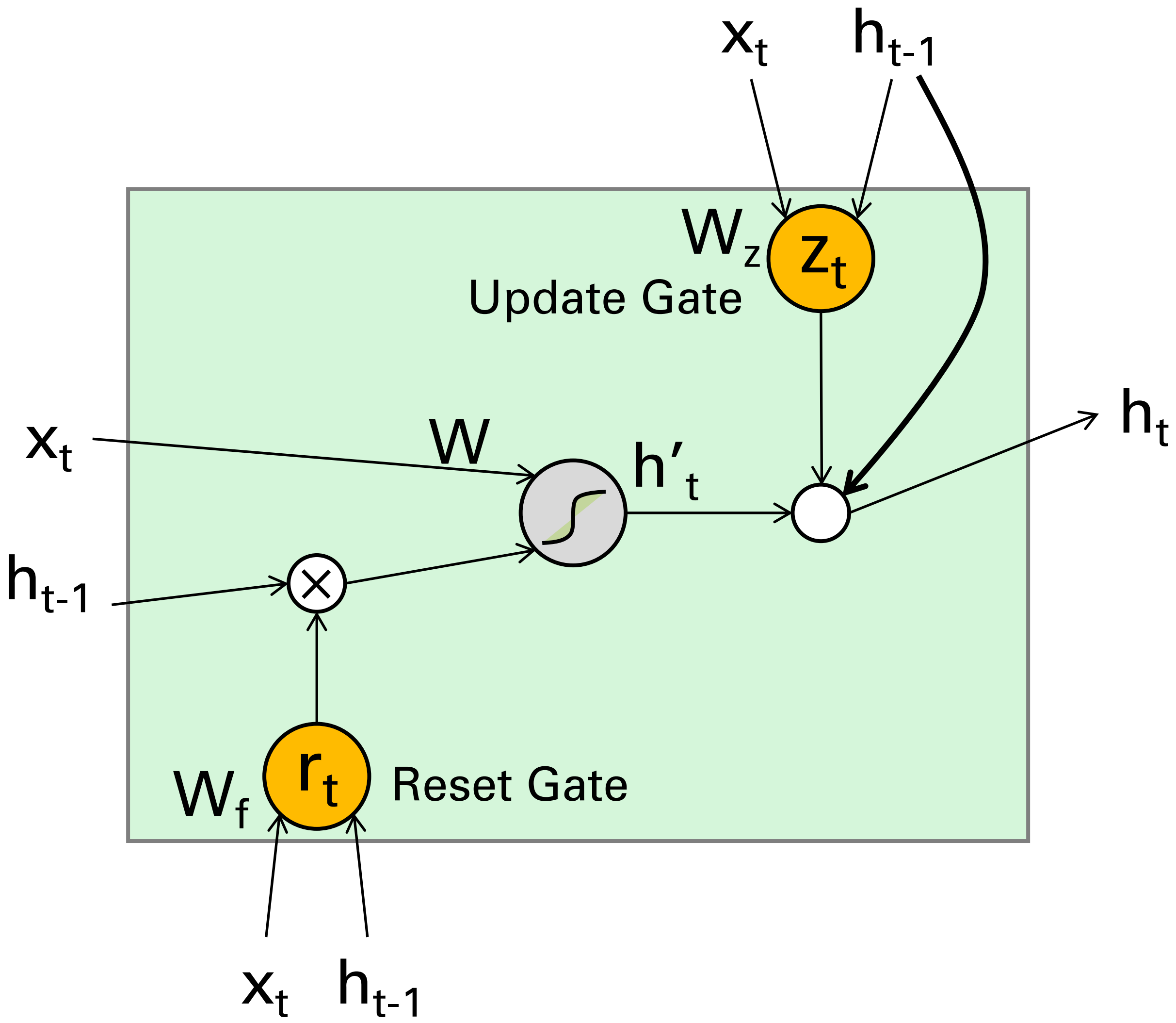
Gated Recurrent Unit (GRU)

- A very simplified version of the LSTM
  - Merges forget and input gate into a single 'update' gate
  - Merges cell and hidden state

- Has fewer parameters than an LSTM and has been shown to outperform LSTM on some tasks

Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation [Cho et al.,14]

GRU



$$r_t = \sigma\left(W_r\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix} + b_f\right)$$

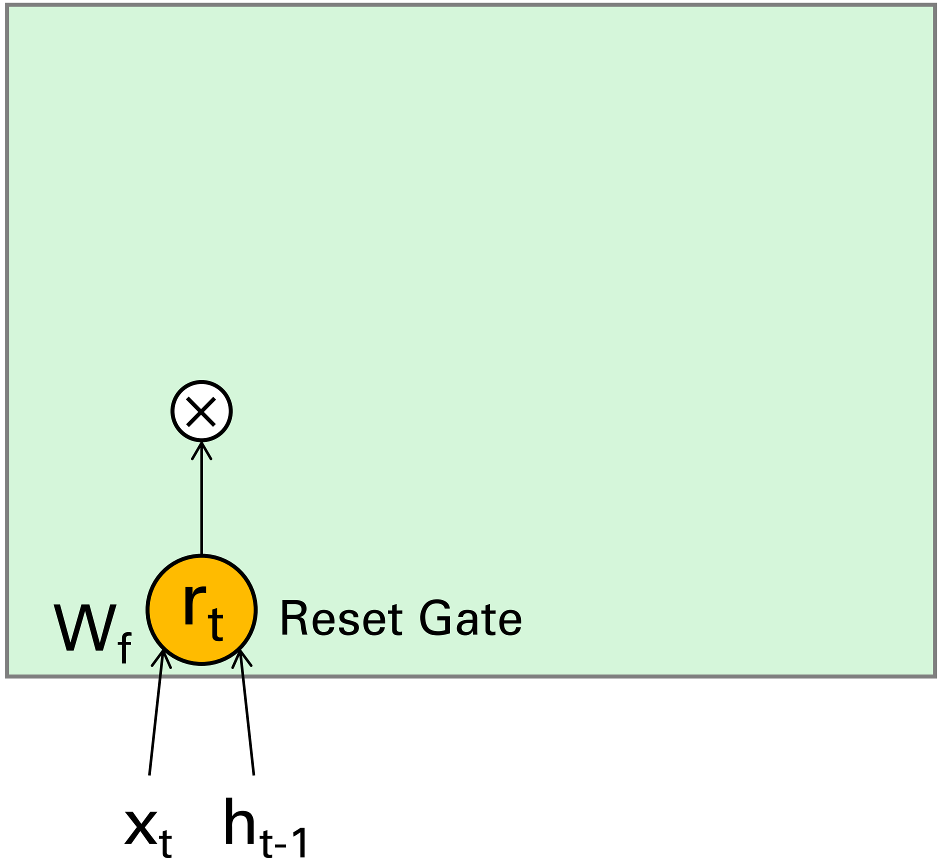
$$h'_t = \tanh W\begin{pmatrix}x_t\\r_t\otimes h_{t-1}\end{pmatrix}$$

$$z_t = \sigma\left(W_z\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix} + b_z\right)$$

$$h_t = (1 - z_t)\otimes h_{t-1} + z_t \otimes h'_t$$

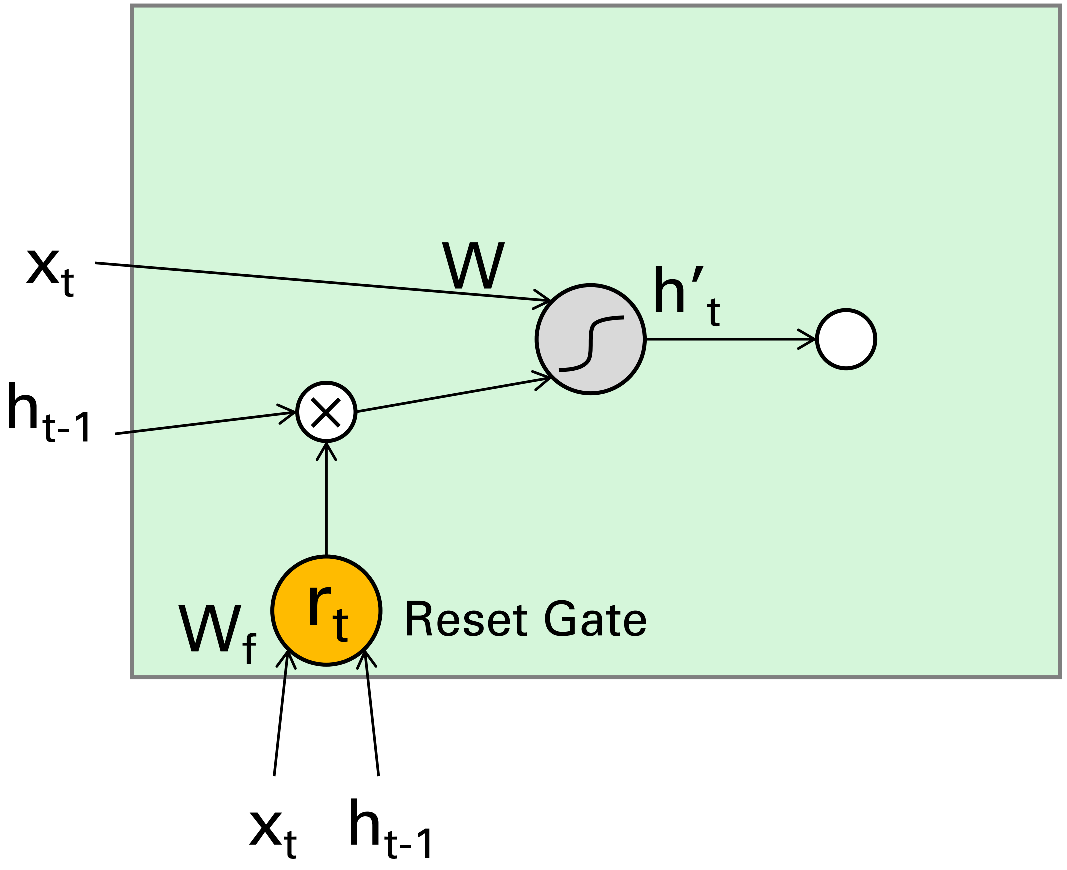
GRU

$$r_t = \sigma\left(W_r\begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f\right)$$



computes a **reset gate** based on current input and hidden state

GRU



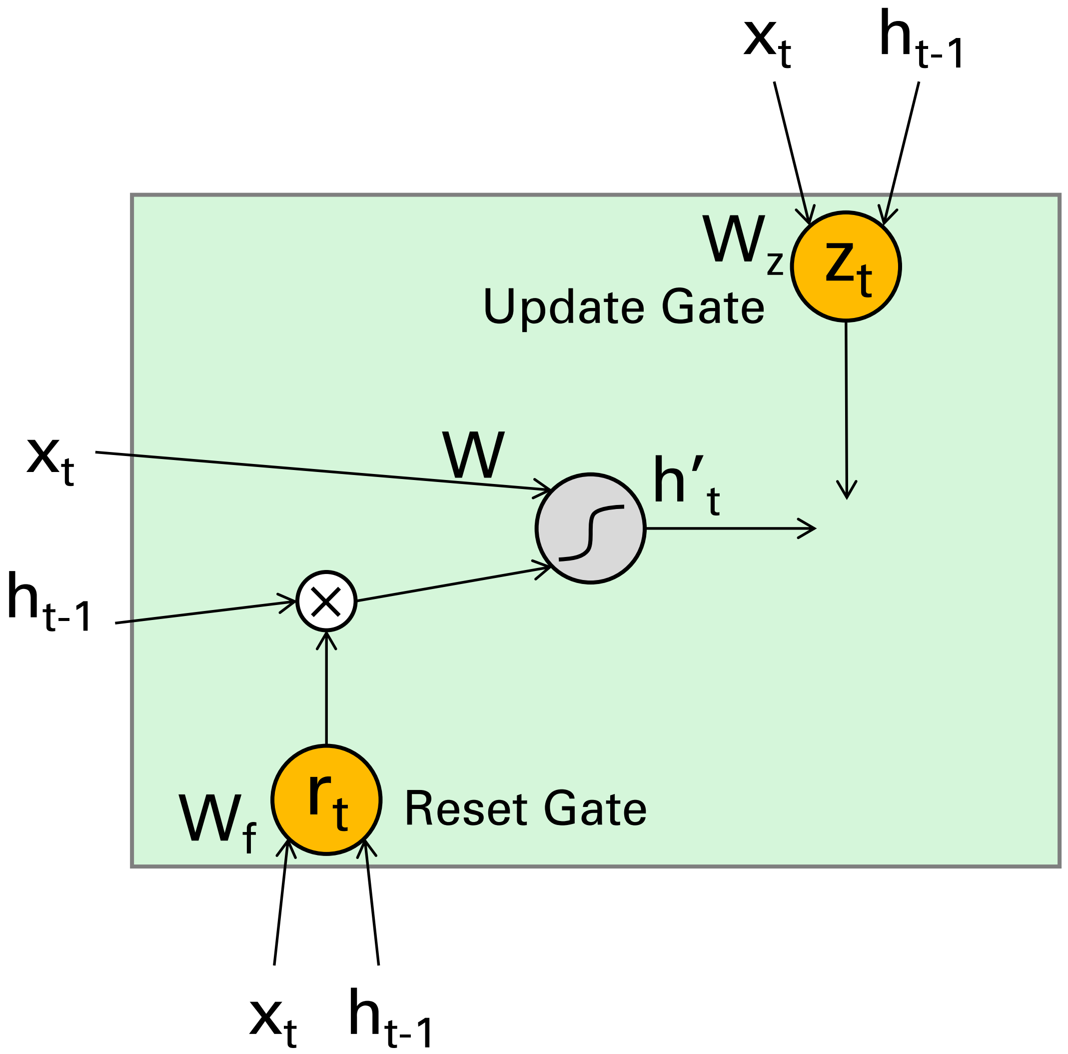
$$r_t = \sigma\left(W_r\begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f\right)$$

$$h'_t = \tanh W\begin{pmatrix} x_t \\ r_t \otimes h_{t-1} \end{pmatrix}$$

computes the **hidden state** based on current input and hidden state

if reset gate unit is ~0, then this ignores previous memory and only stores the new input information

GRU



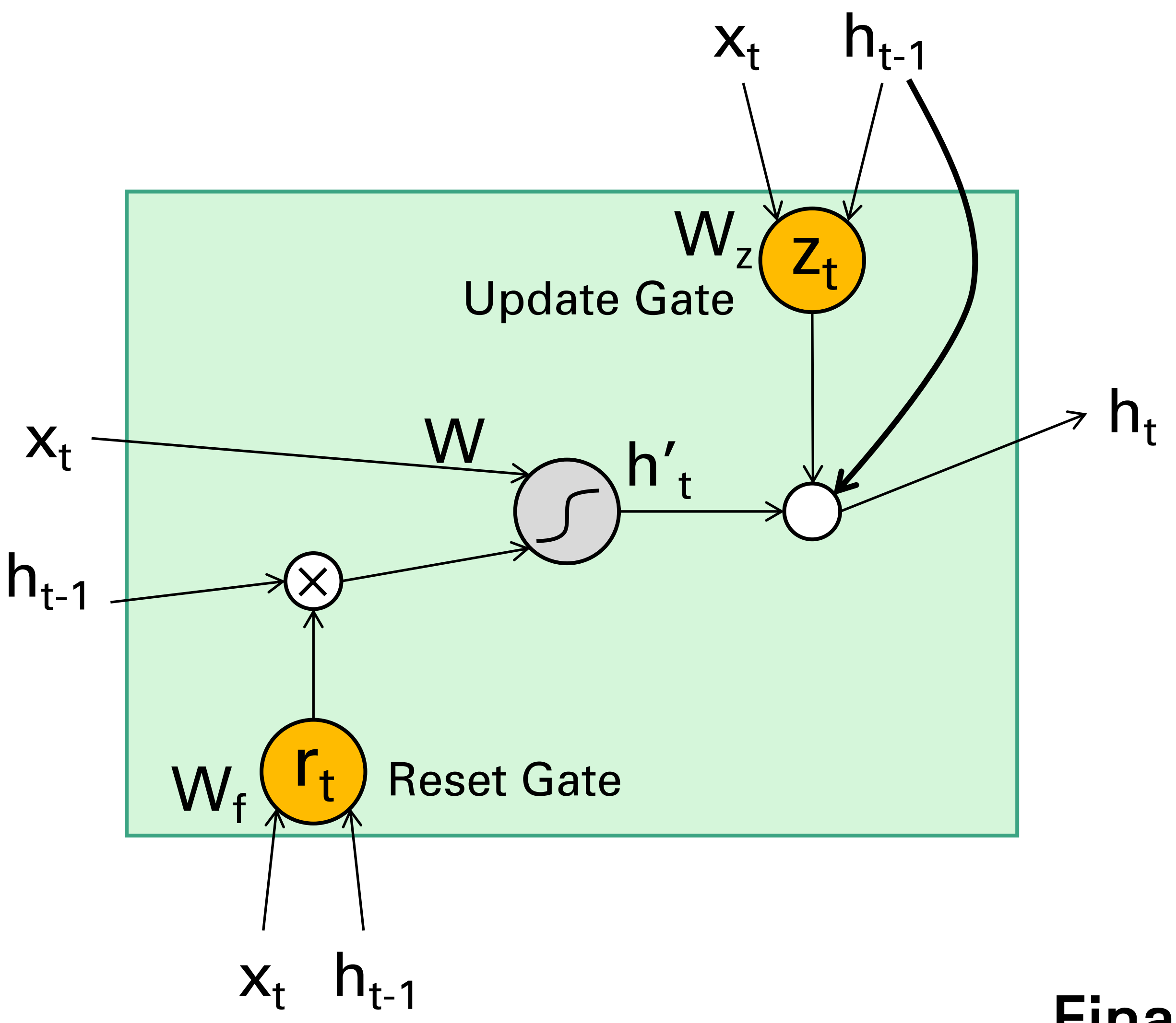
$$r_t = \sigma\left(W_r \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f \right)$$

$$h'_t = \tanh W \begin{pmatrix} x_t \\ r_t \otimes h_{t-1} \end{pmatrix}$$

$$z_t = \sigma\left(W_z \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_z \right)$$

computes an **update gate** again based on current input and hidden state

GRU



$$r_t = \sigma\left(W_r \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f\right)$$

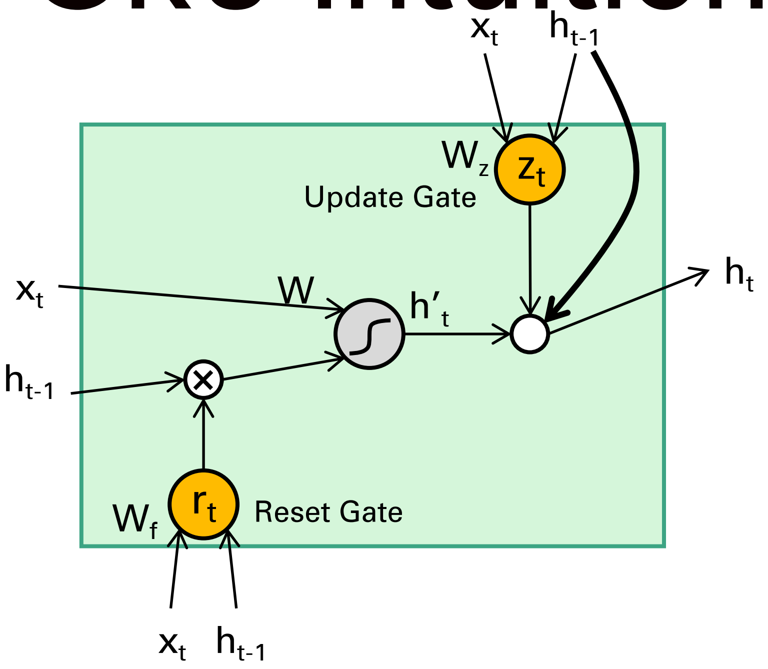
$$h'_t = \tanh W \begin{pmatrix} x_t \\ r_t \otimes h_{t-1} \end{pmatrix}$$

$$z_t = \sigma\left(W_z \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_z\right)$$

$$h_t = (1 - z_t) \otimes h_{t-1} + z_t \otimes h'_t$$

Final memory at timestep t combines both current and previous timesteps

GRU Intuition



$$r_t = \sigma\left(W_r\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_f\right)$$

$$h'_t = \tanh W\begin{pmatrix} x_t \\ r_t \otimes h_{t-1}\end{pmatrix}$$

$$z_t = \sigma\left(W_z\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_f\right)$$

$$h_t = (1 - z_t) \otimes h_{t-1} + z_t \otimes h'_t$$

- If reset is close to 0, ignore previous hidden state
  - Allows model to drop information that is irrelevant in the future
- Update gate z controls how much of past state should matter now.
  - If z close to 1, then we can copy information in that unit through many time steps! **Less vanishing gradient!**
- Units with short-term dependencies often have reset gates very active

Slide credit: Richard Socher

LSTMs and GRUs

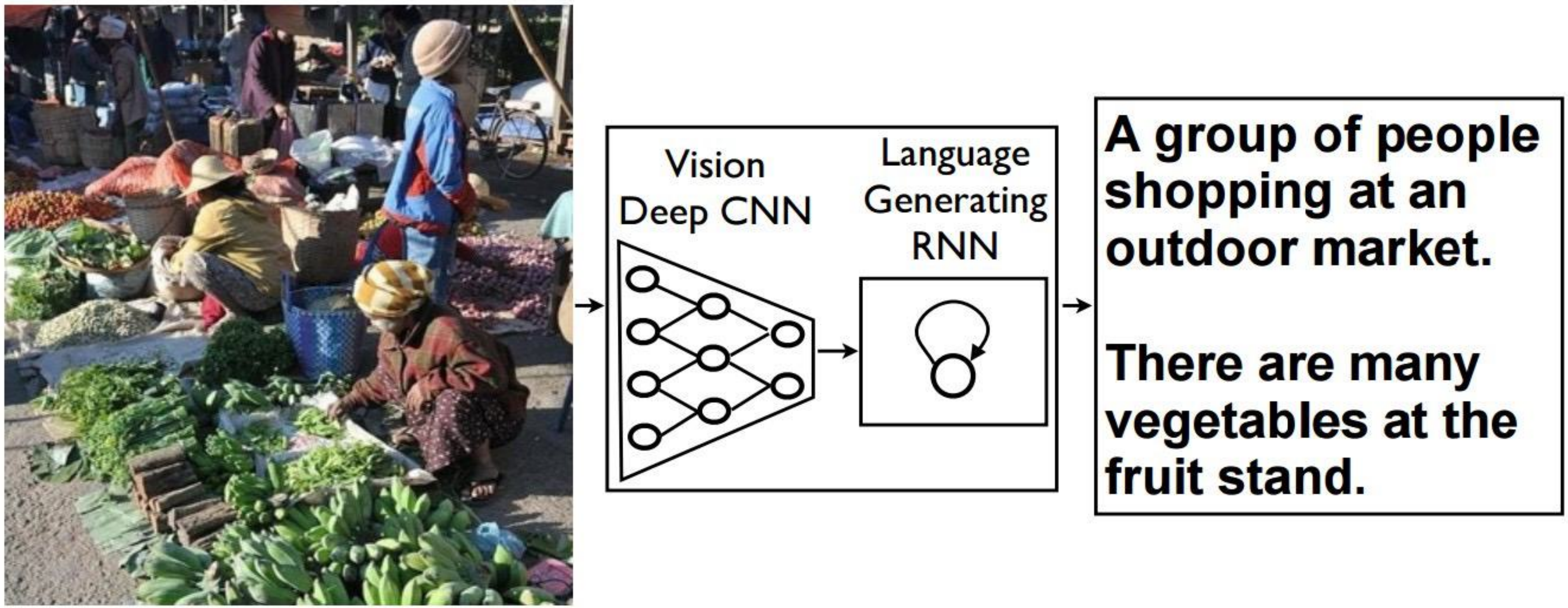
Good

- Careful initialization and optimization of vanilla RNNs can enable them to learn long(ish) dependencies, but gated additive cells, like the LSTM and GRU, often just work.

Bad

- LSTMs and GRUs have considerably more parameters and computation per memory cell than a vanilla RNN, as such they have less memory capacity per parameter*

Image Captioning



[Example above from: Vinyals, Toshev, Bengio, Erhan, CVPR 2015, https://arxiv.org/abs/1411.4555]

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?

Training data

$\mathbf{x}$ y

$\{$  , "Fish" $\}$

$\{$  , "Grizzly" $\}$

$\{$  , "Chameleon" $\}$

$\vdots$

→

Training data

$\mathbf{x}$ y

$\{$  , 1 $\}$

$\{$  , 2 $\}$

$\{$  , 3 $\}$

$\vdots$

→

One-hot vector

Training data

$\mathbf{x}$ $\mathbf{y}$

$\{$  , [0,0,1] $\}$

$\{$  , [0,1,0] $\}$

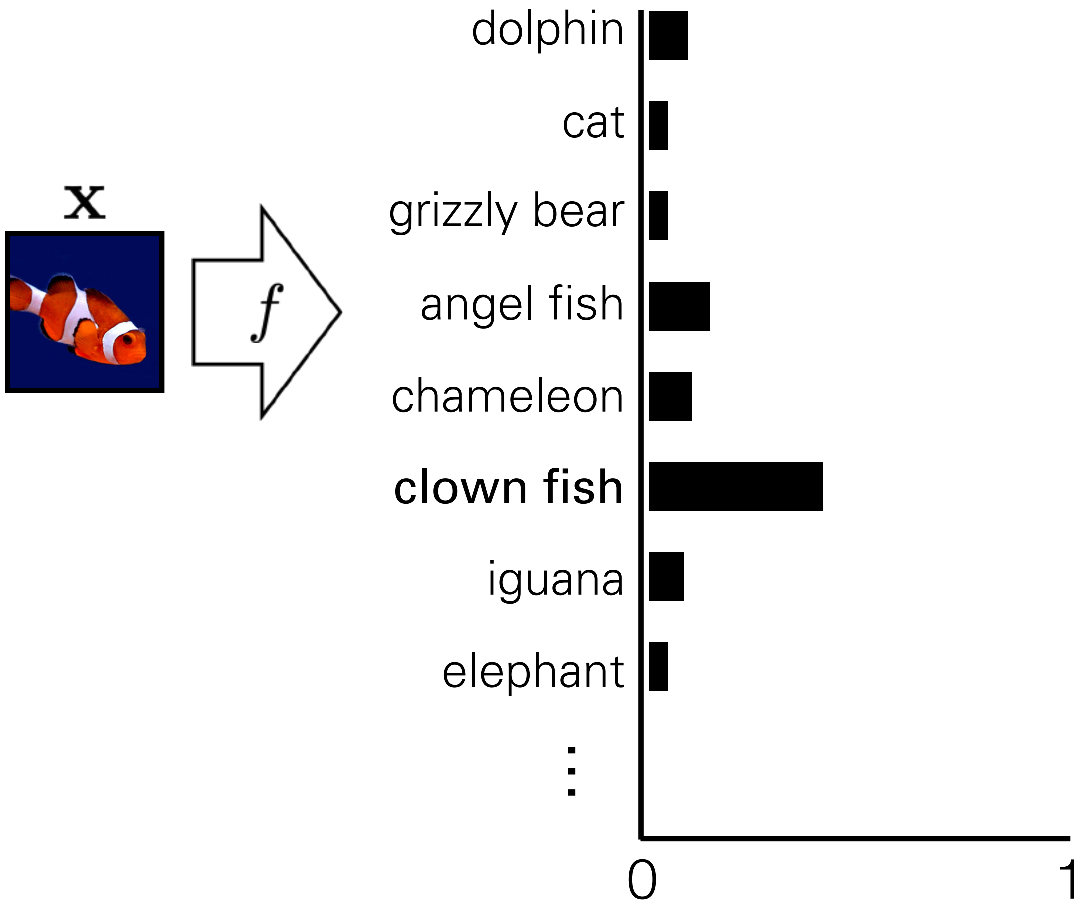
$\{$  , [1,0,0] $\}$

$\vdots$

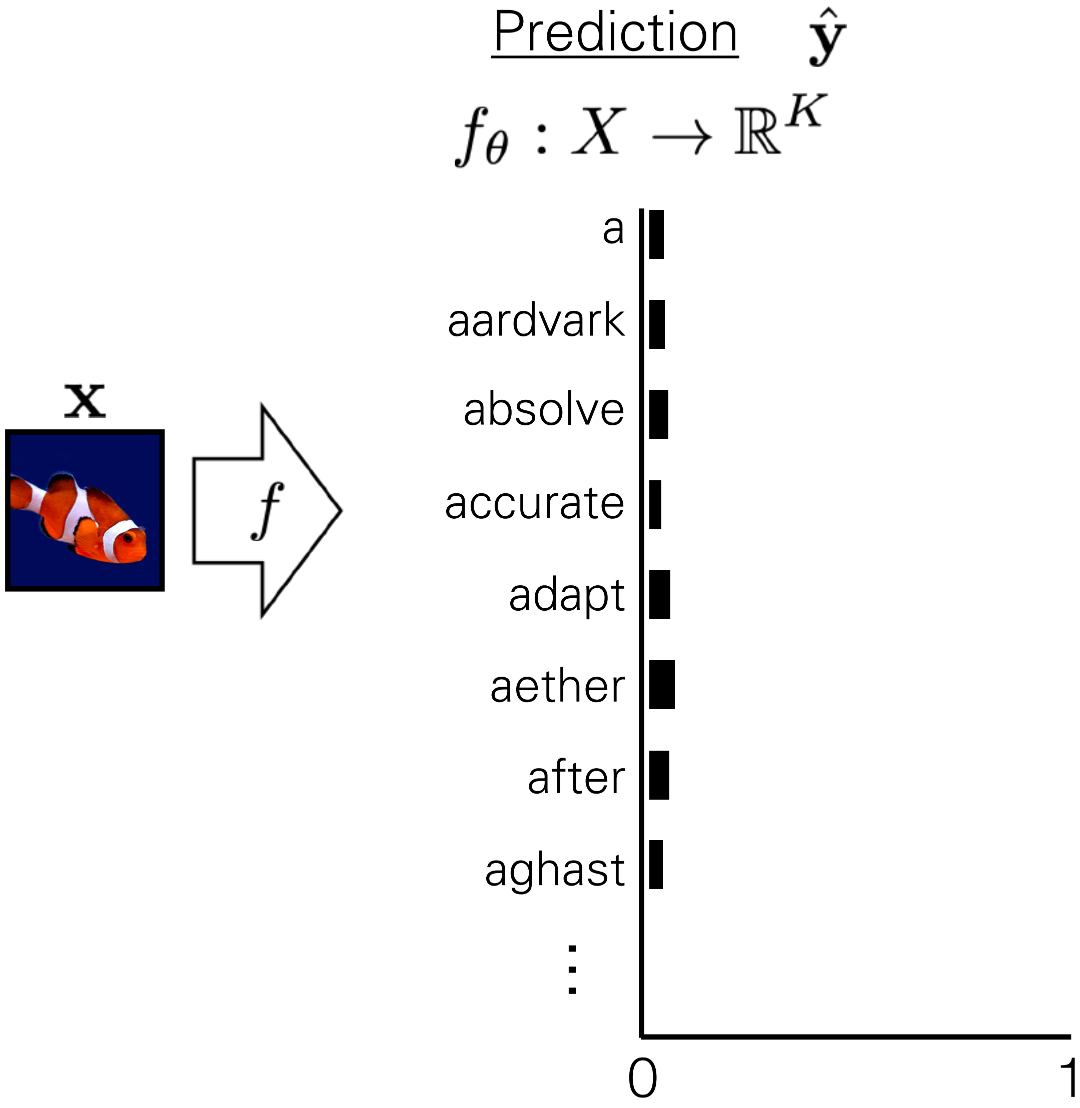
How to represent words as numbers?

Prediction $\hat{\mathbf{y}}$

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

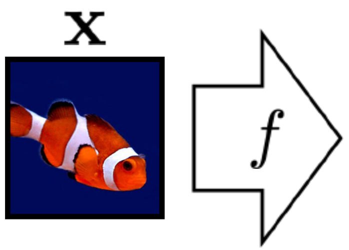


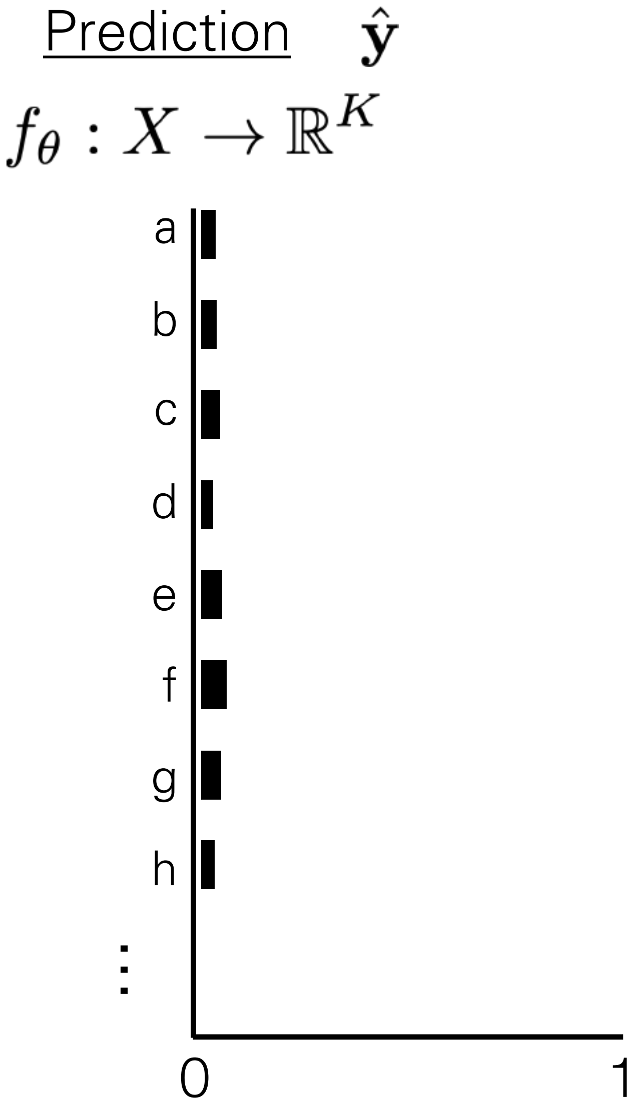
How to represent words as numbers?



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).

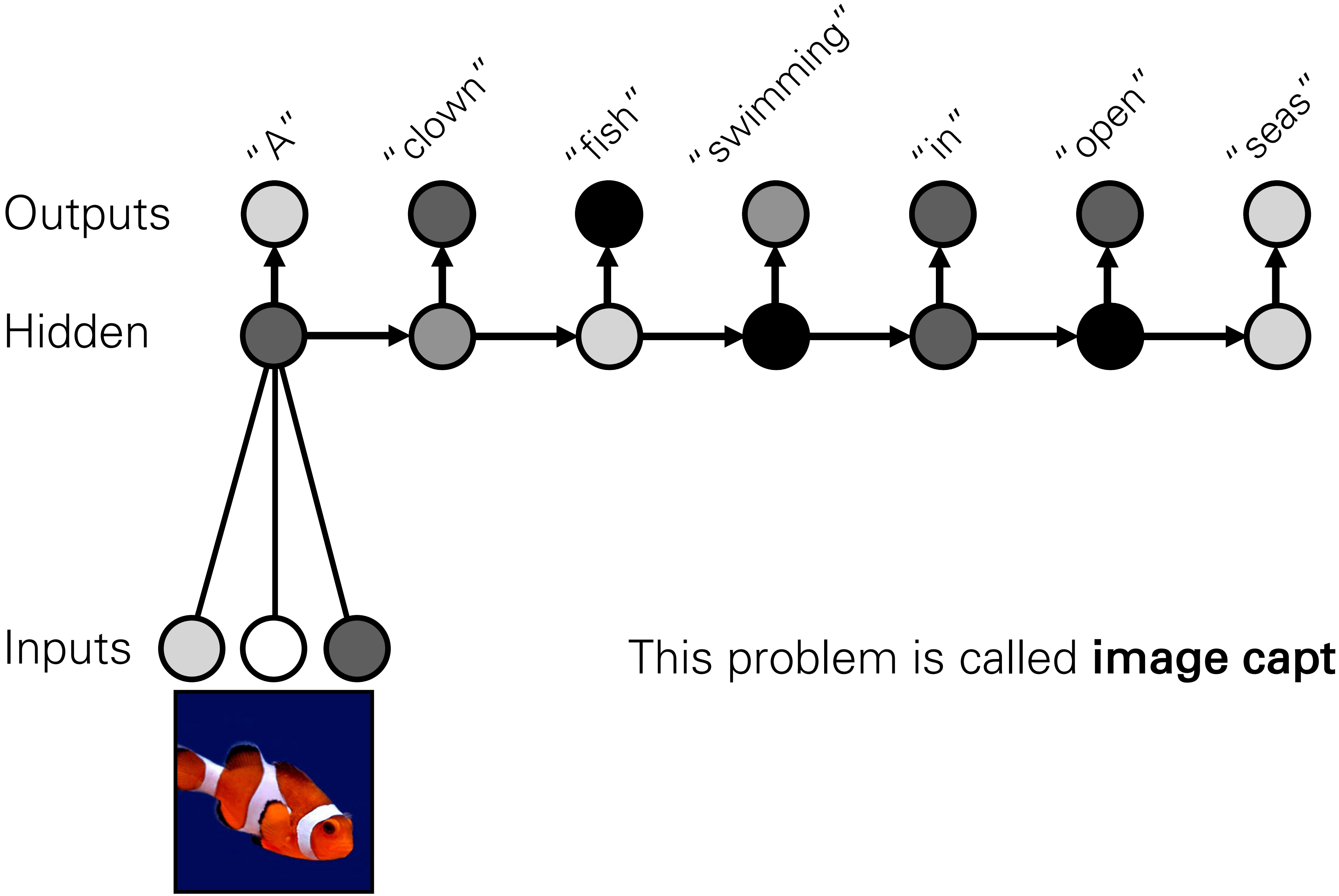
How to represent words as numbers?



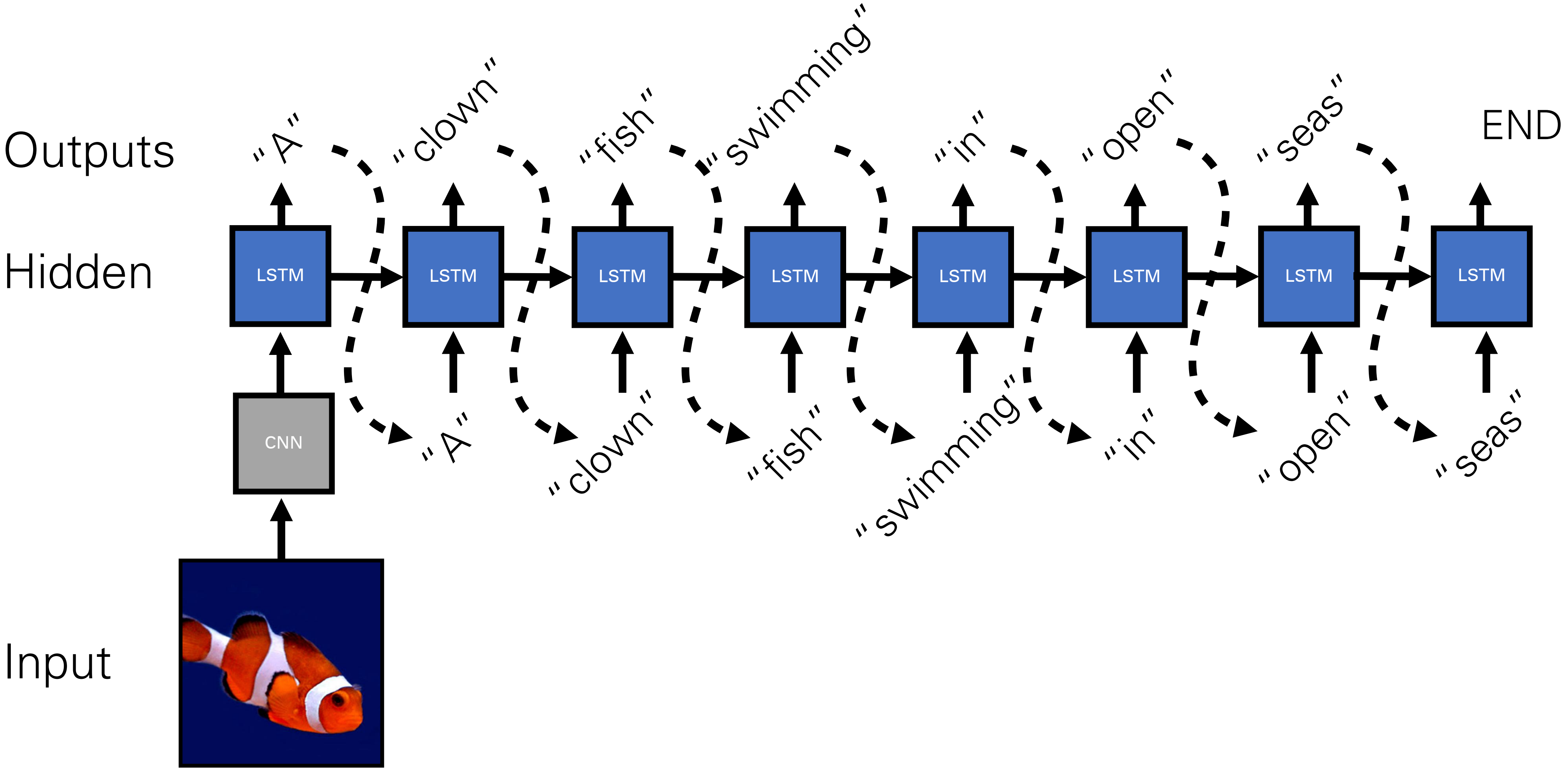


Or, represent each character as a class (e.g., K=26 for English letters),

and represent words as a sequence of characters.

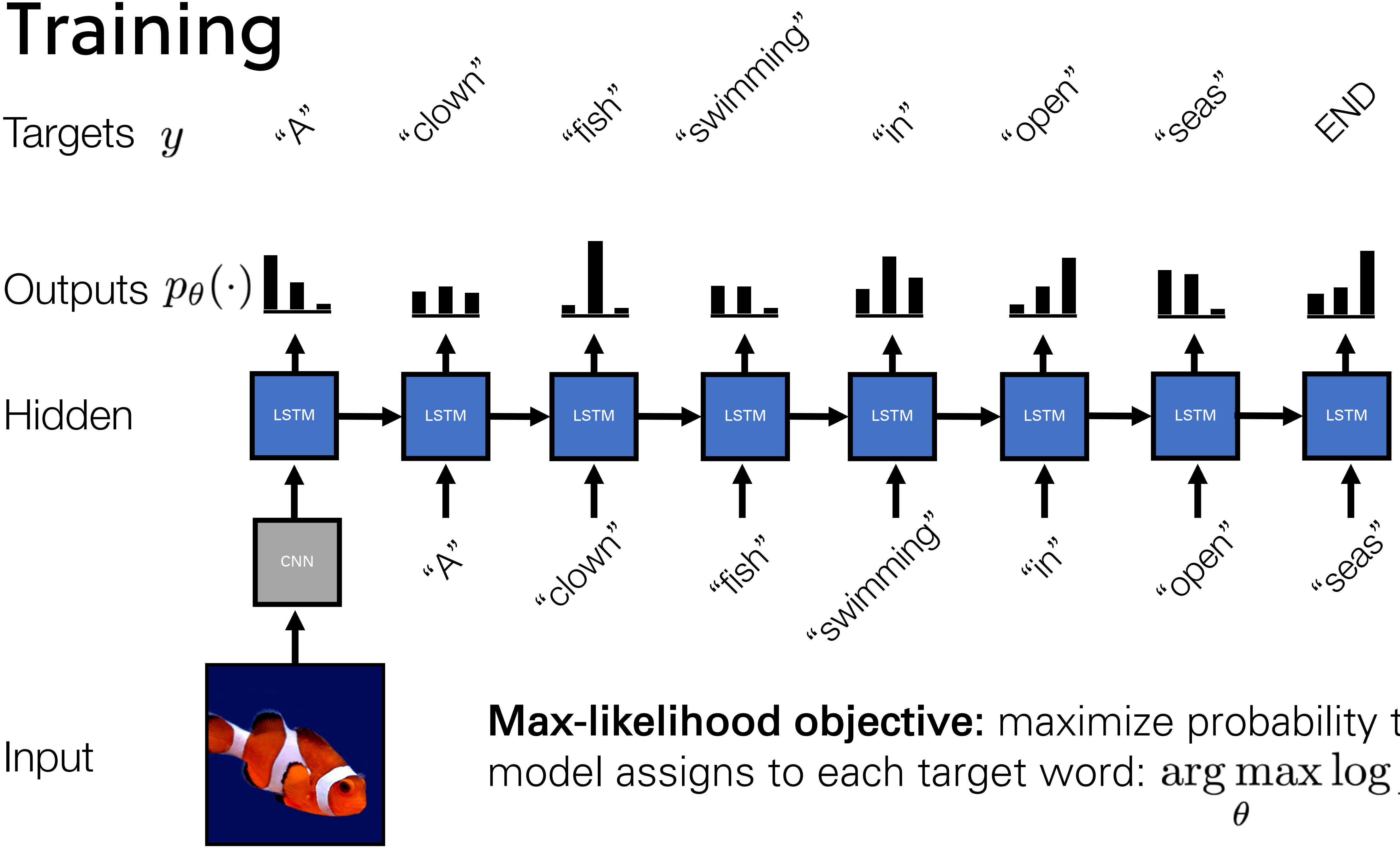


This problem is called **image captioning**



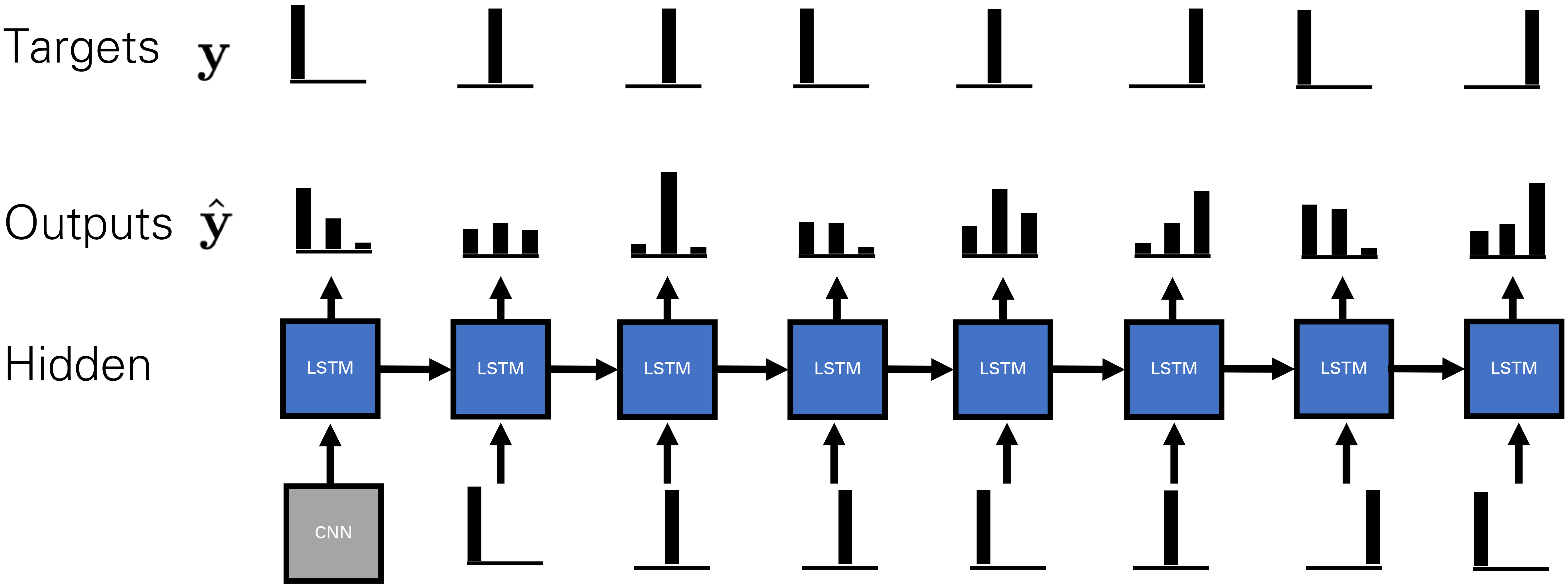
Outputs
Hidden
Input
"A"
"clown"
"fish"
"swimming"
"in"
"open"
"seas"
END
LSTM
LSTM
LSTM
LSTM
LSTM
LSTM
LSTM
LSTM
CNN
"A"
"clown"
"fish"
"swimming"
"in"
"open"
"seas"

Training

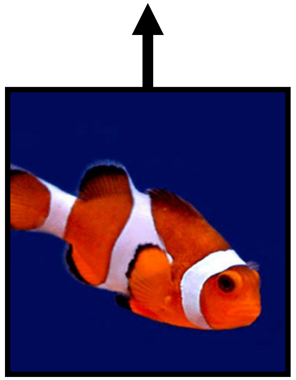


Max-likelihood objective: maximize probability the model assigns to each target word: $\arg\max_\theta \log p_\theta(y)$

Training



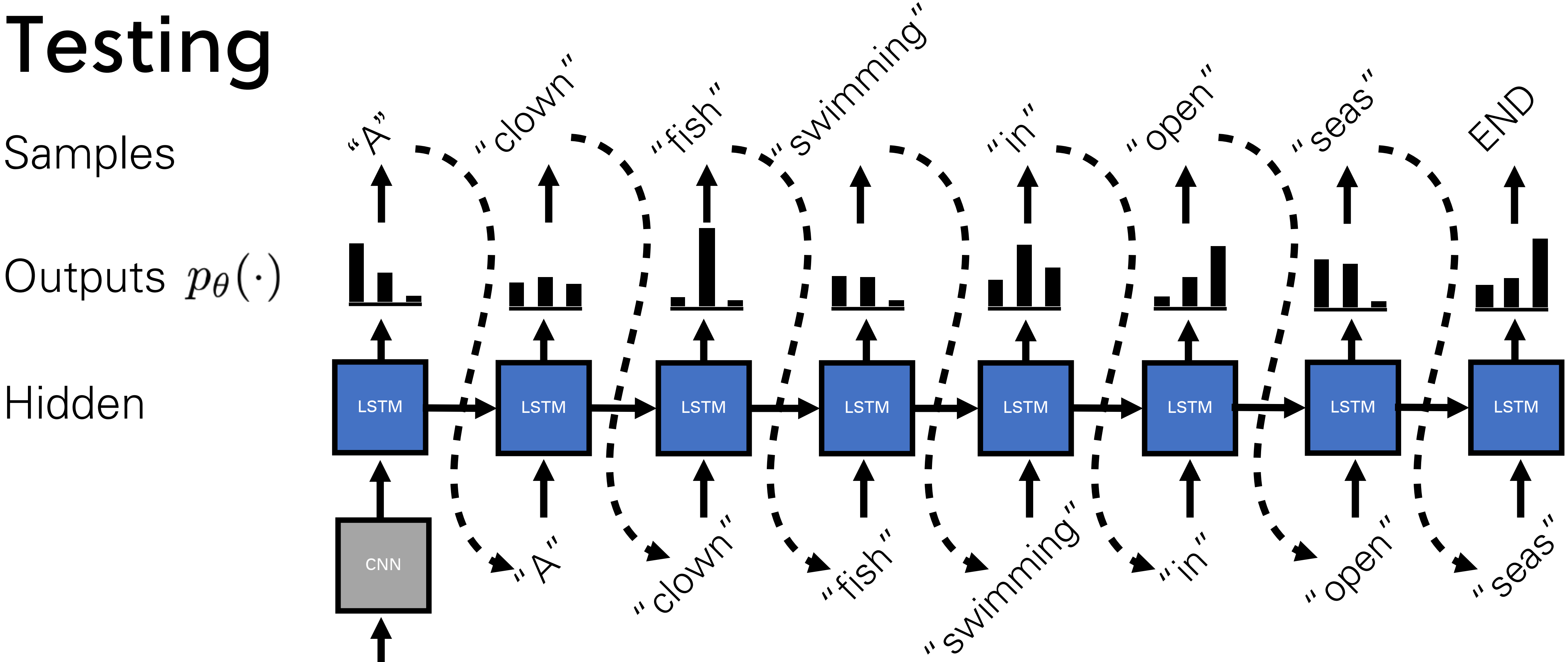
Input

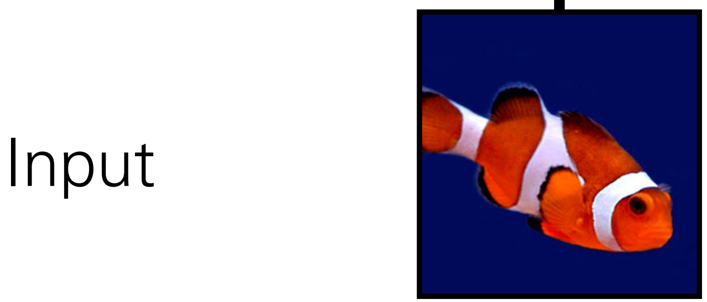


Max-likelihood objective:
minimize cross-entropy between model outputs and one-hot encoded targets.

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

Testing





Sample from predicted distribution over words.

Alternatively, sample most likely word.

A person riding a motorcycle on a dirt road.

Two dogs play in the grass.

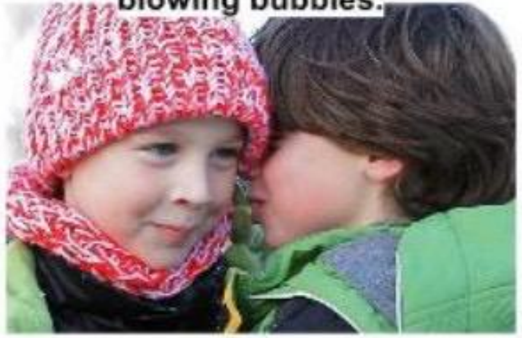
A skateboarder does a trick on a ramp.

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

Next Lecture:
Attention and Transformers