Using RNNs to generate Super Mario Maker levels, Adam Geitgey

COMP547

DEEP UNSUPERVISED LEARNING

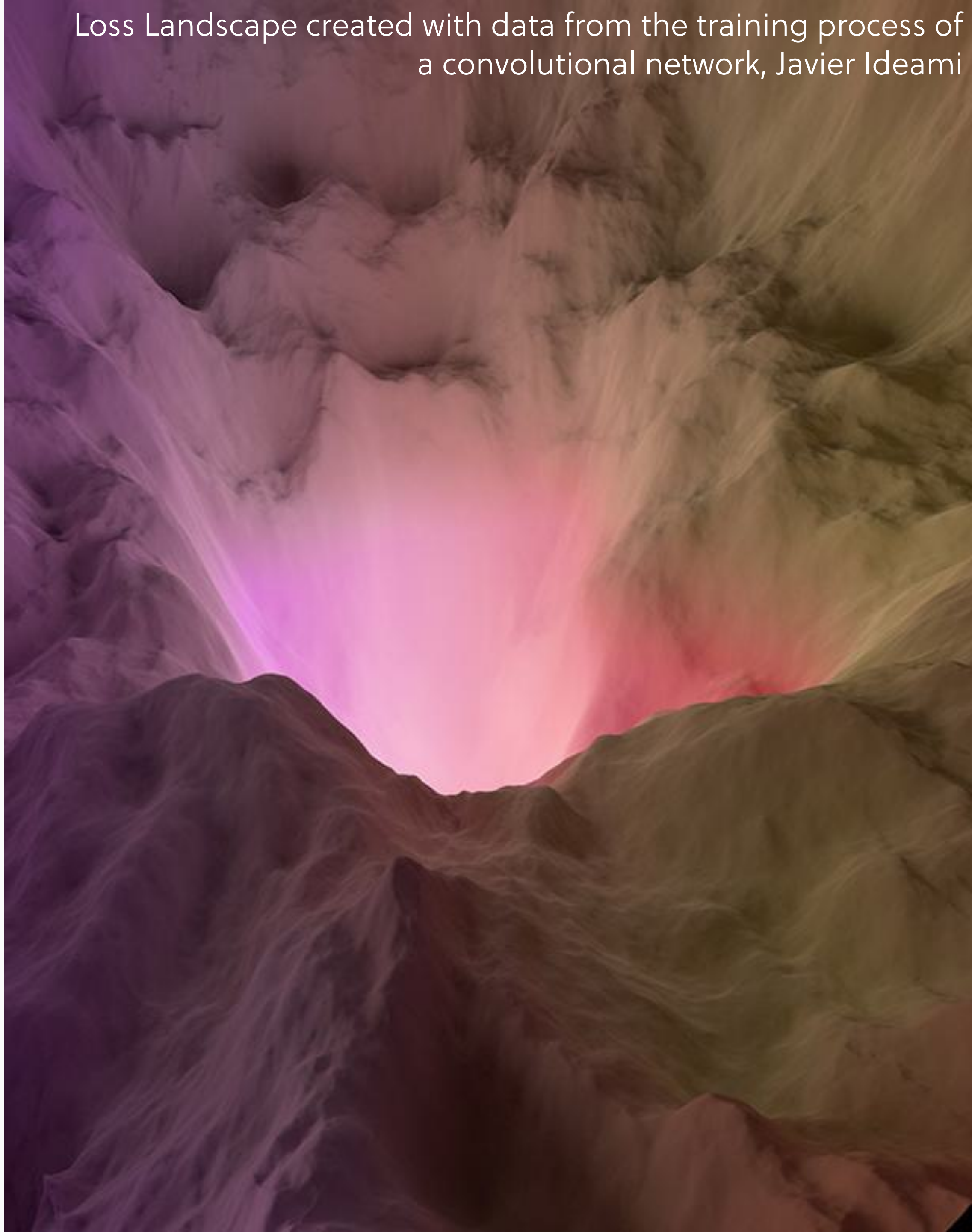
Lecture #3 – Neural Networks Basics II: Sequential Processing with NNs

KOÇ UNIVERSITY

Aykut Erdem // Koç University // Spring 2025

Previously on COMP547

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



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

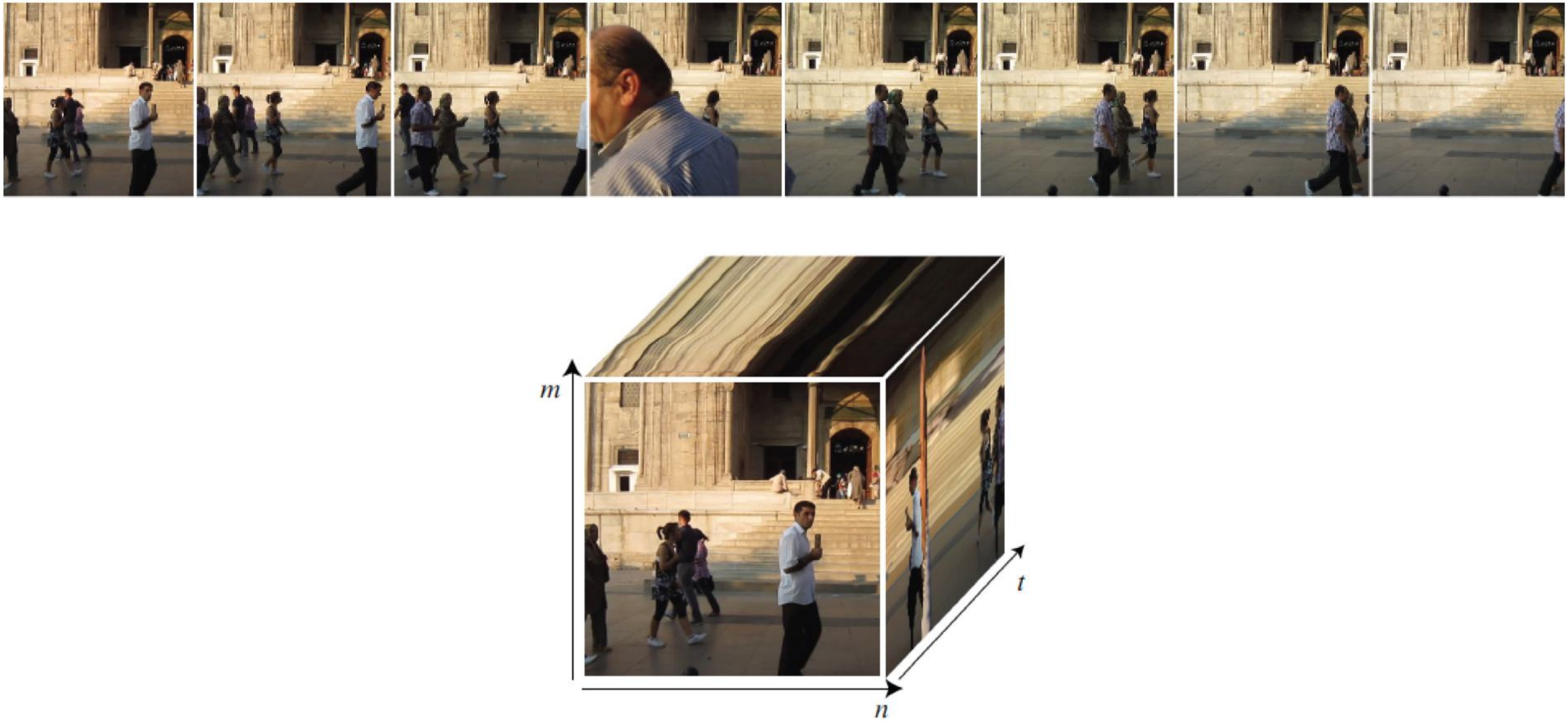
Sequences

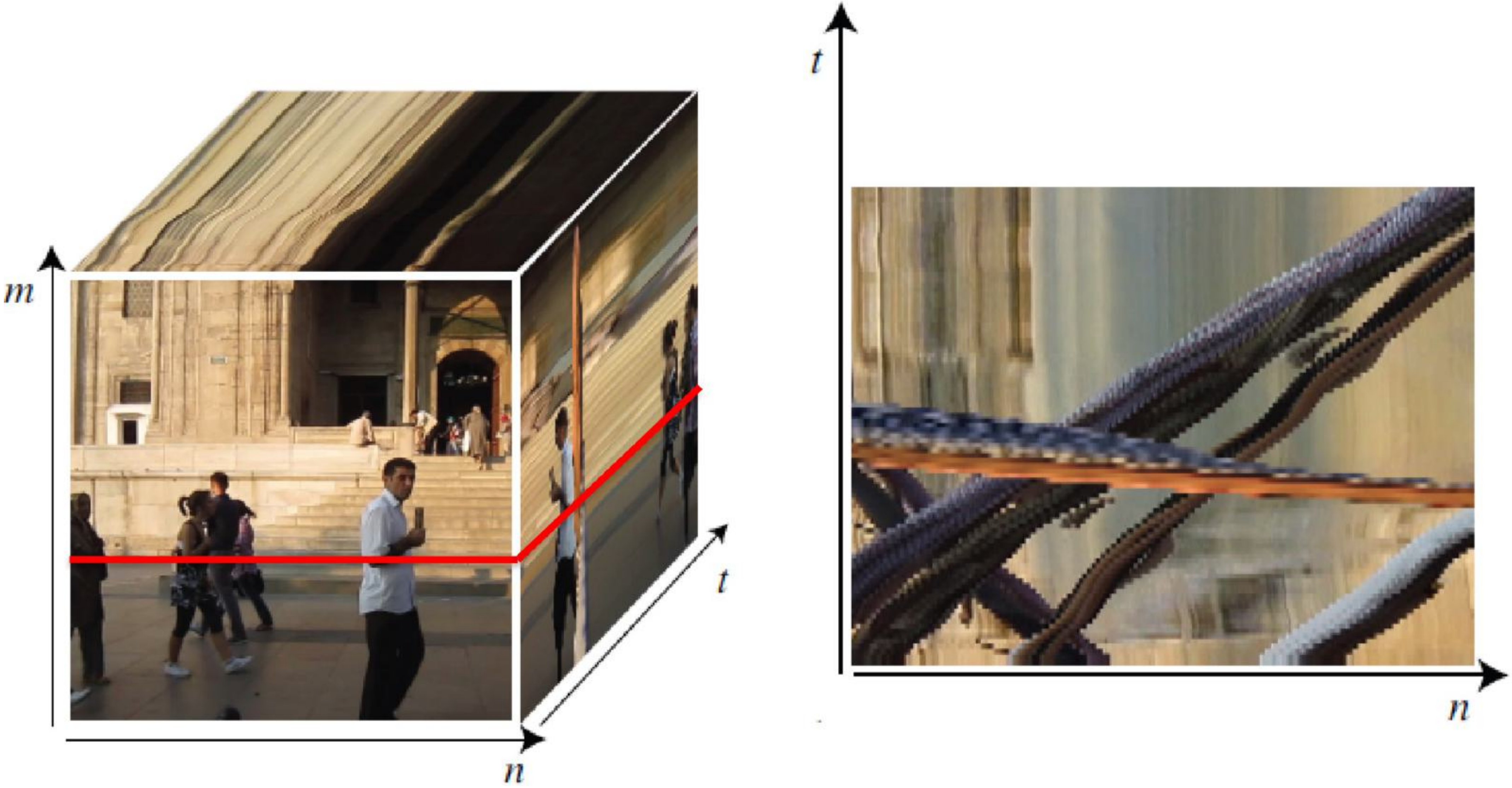
time

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

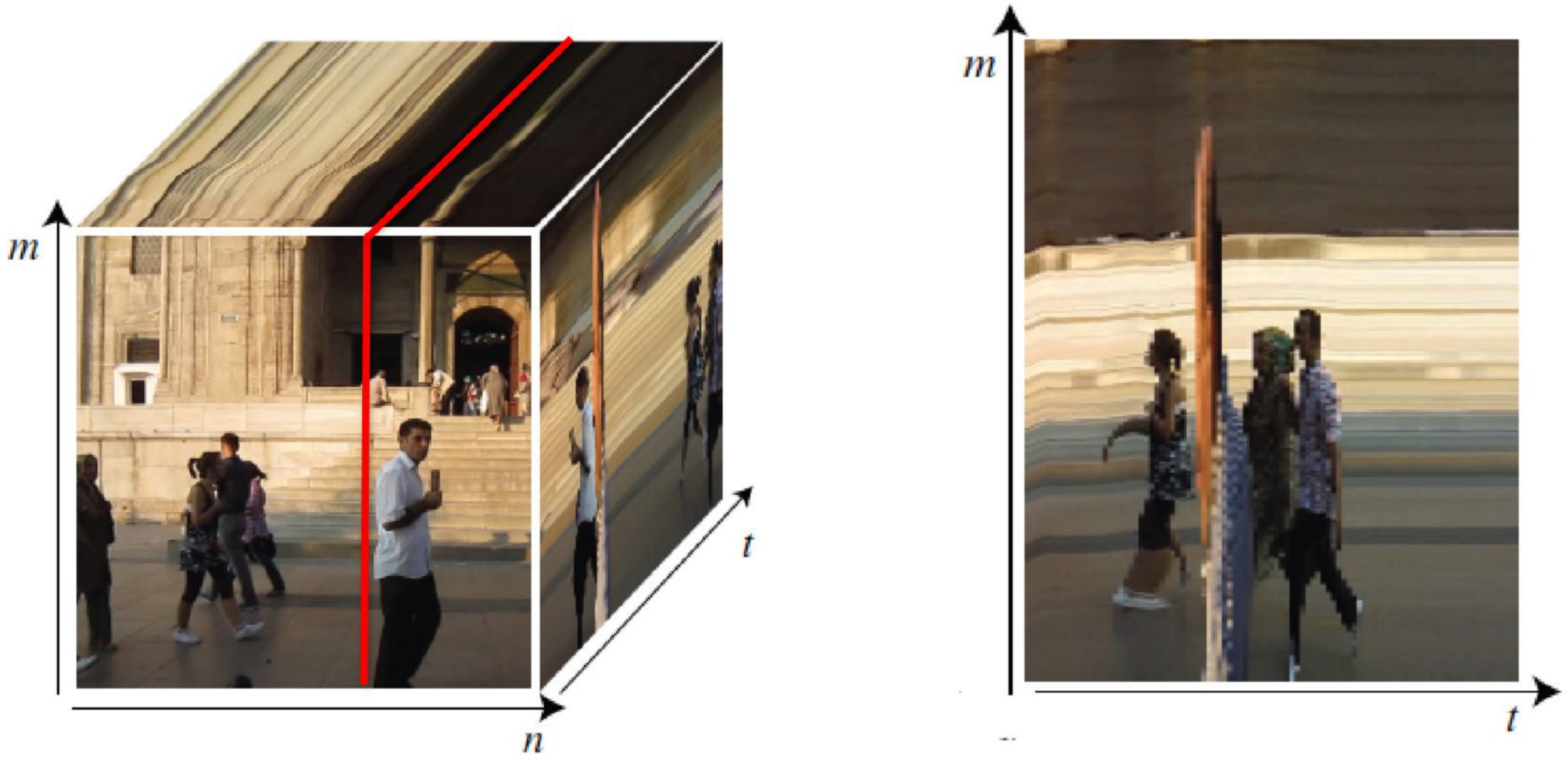
time

time

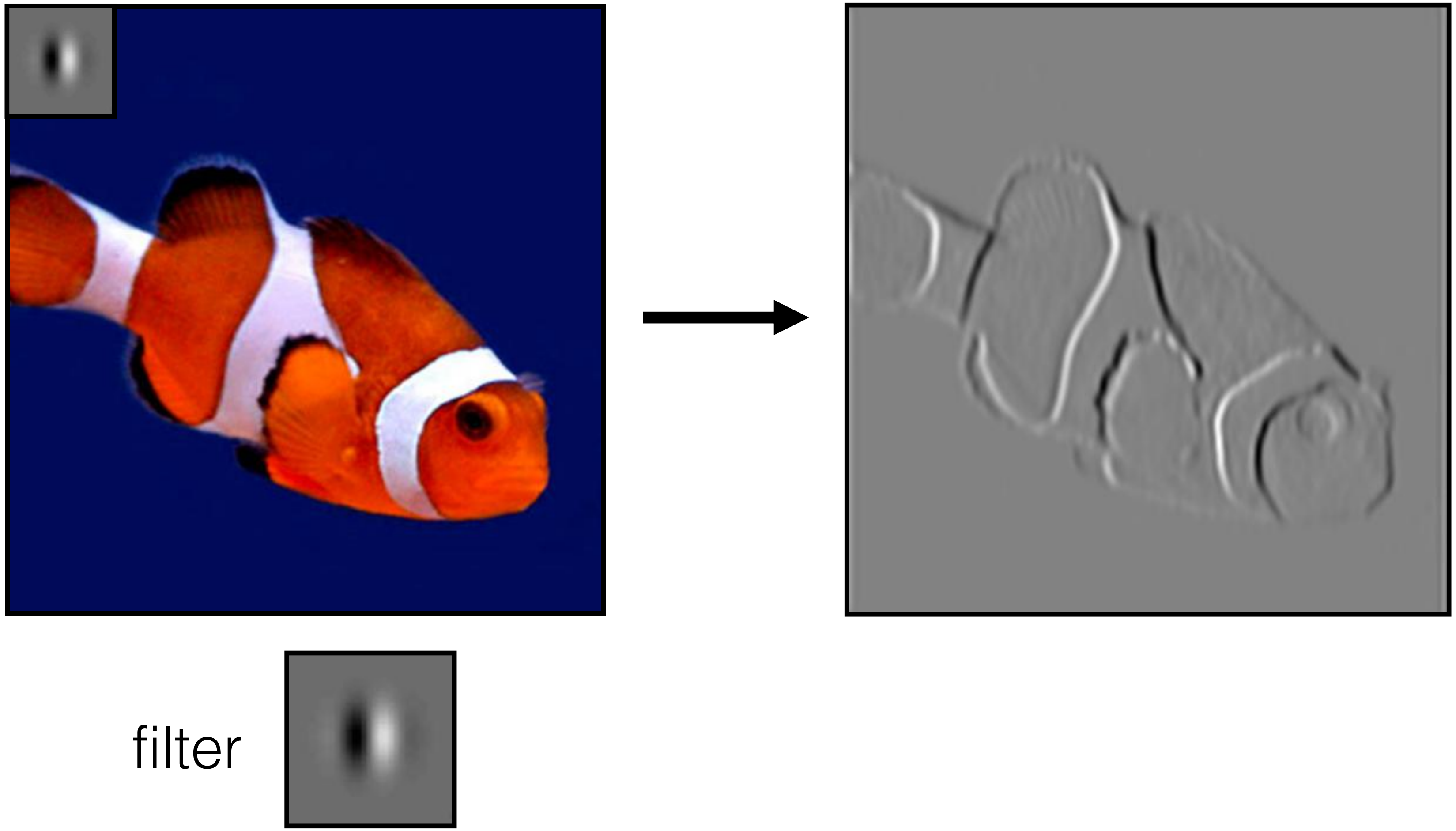

m
n
t



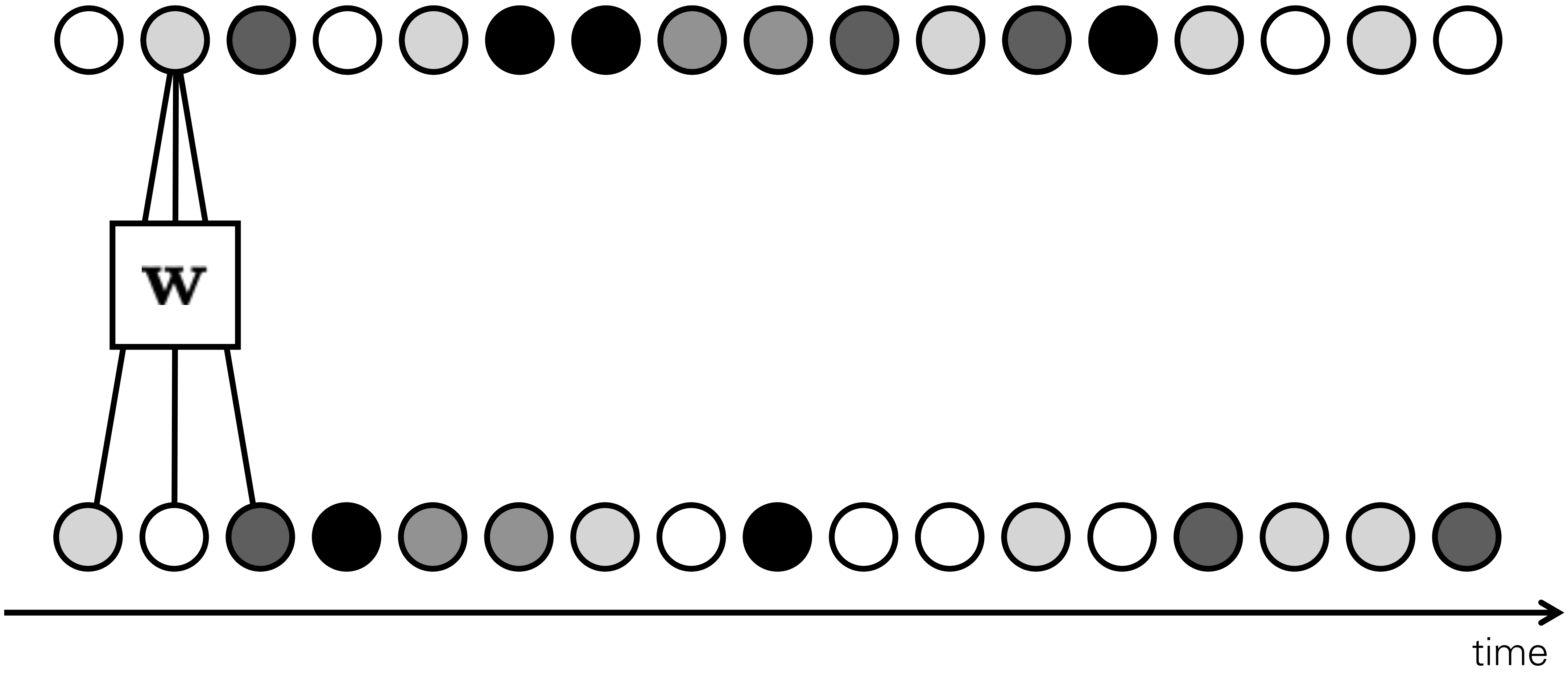
m
n
t
t
n

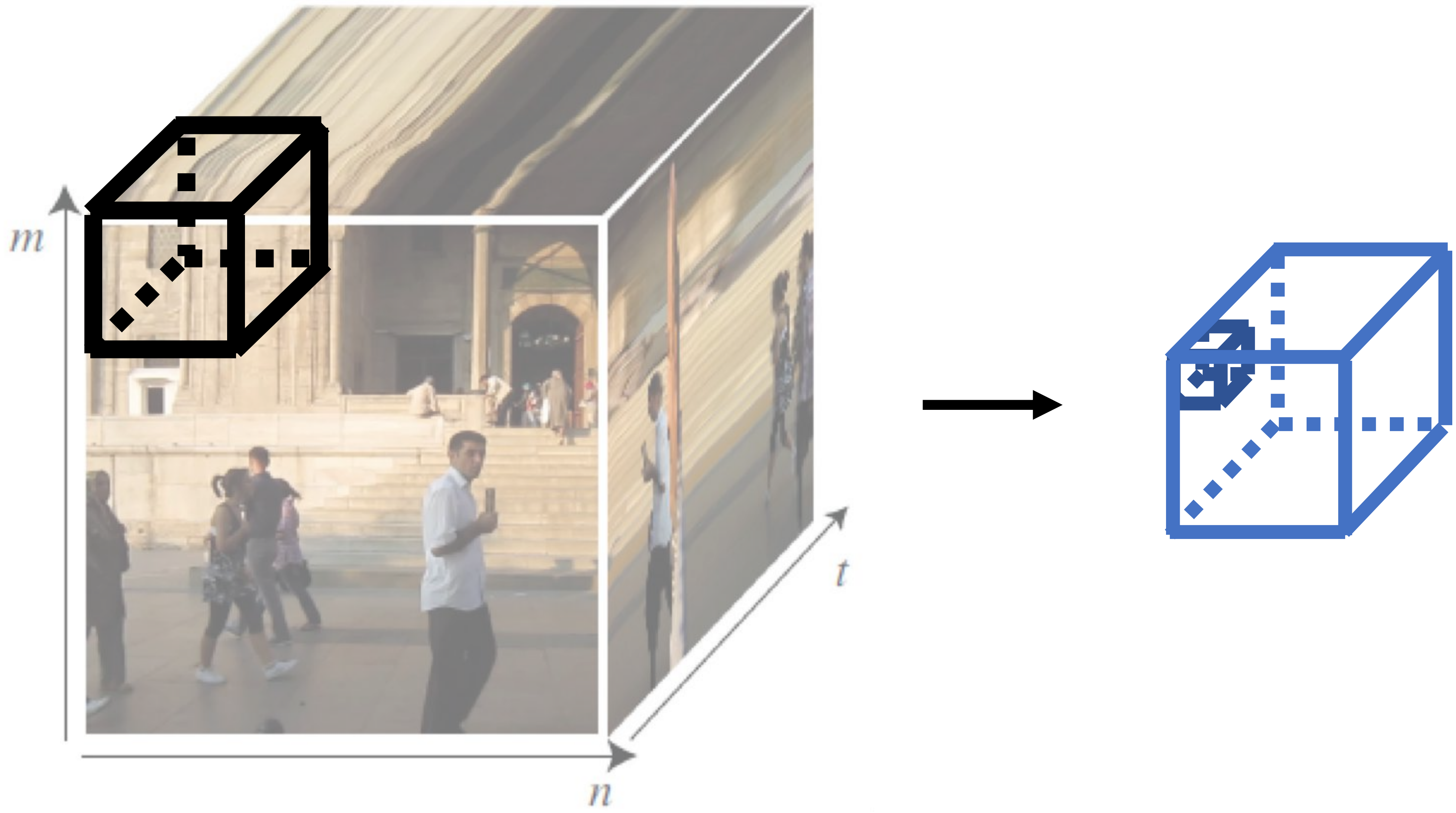


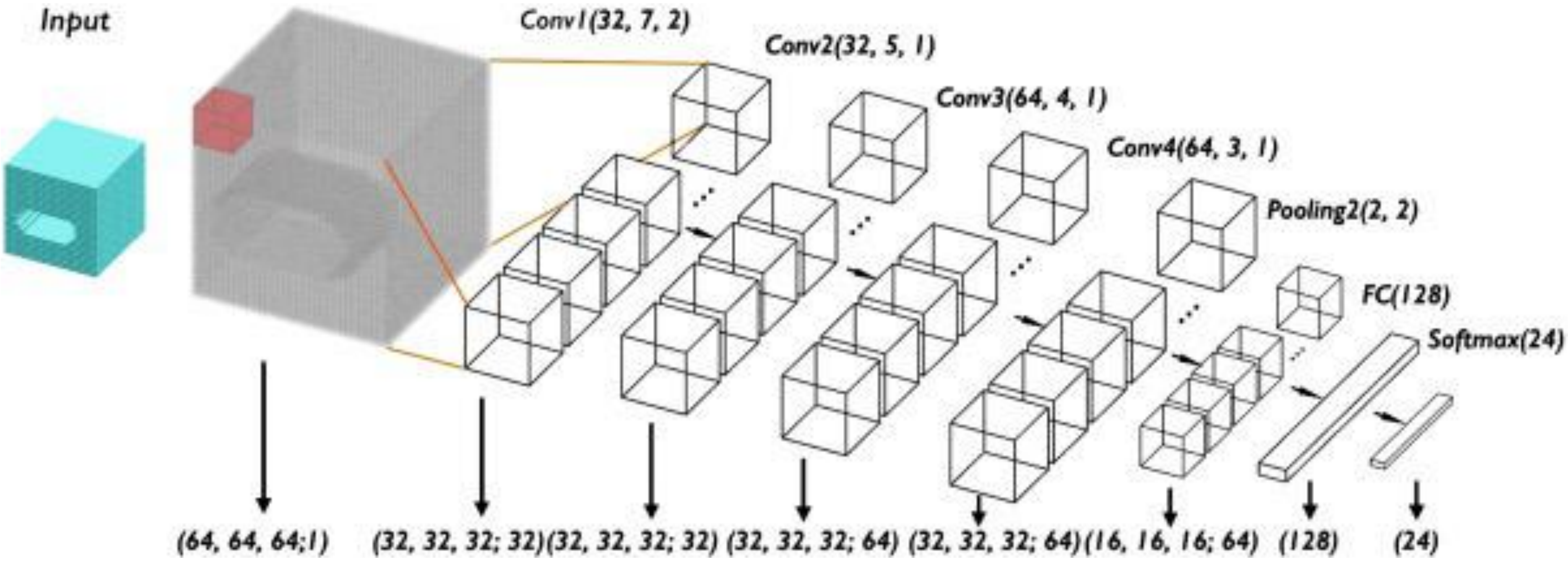
m
n
t
m
t


filter

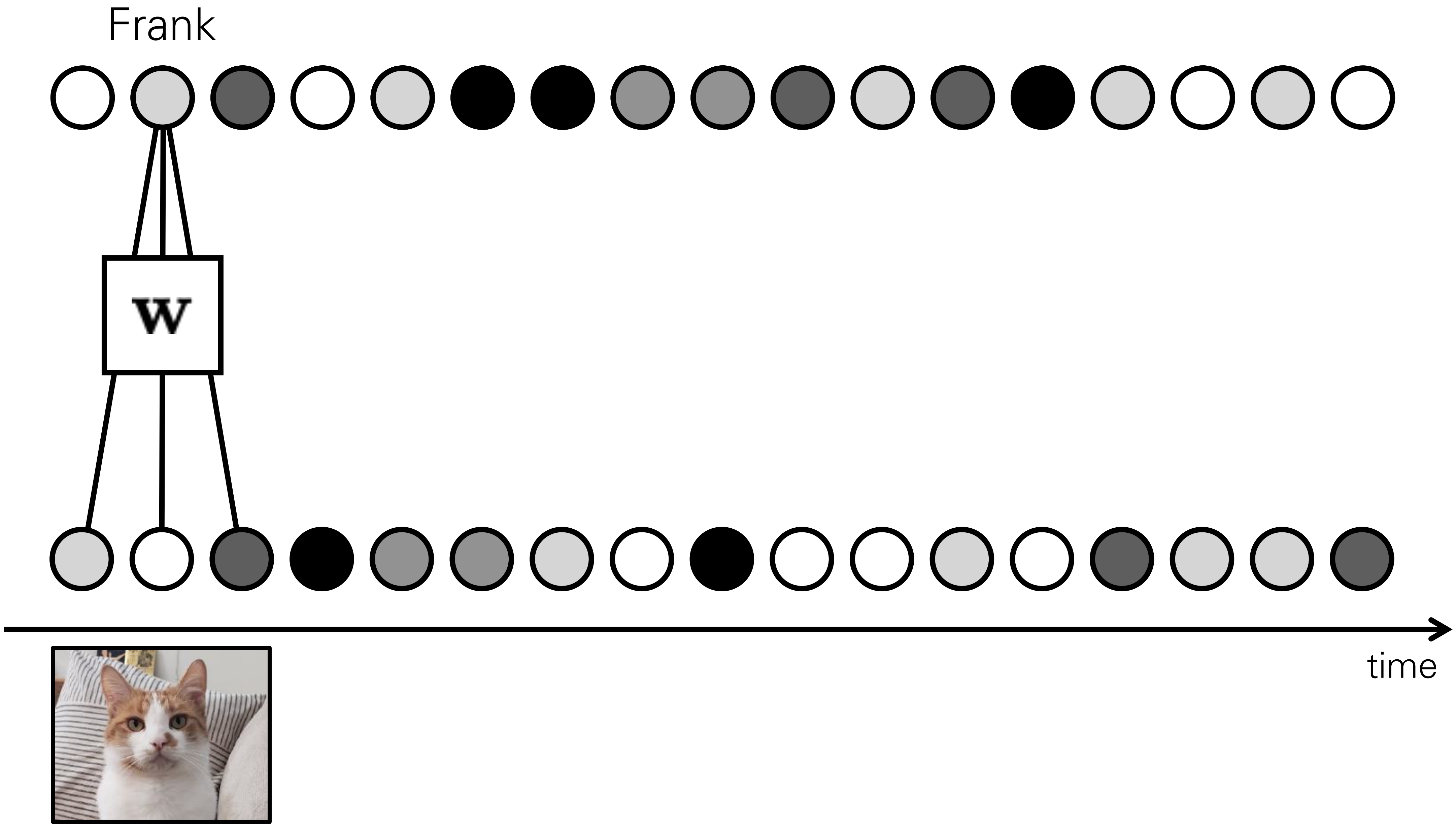
Convolutions in time

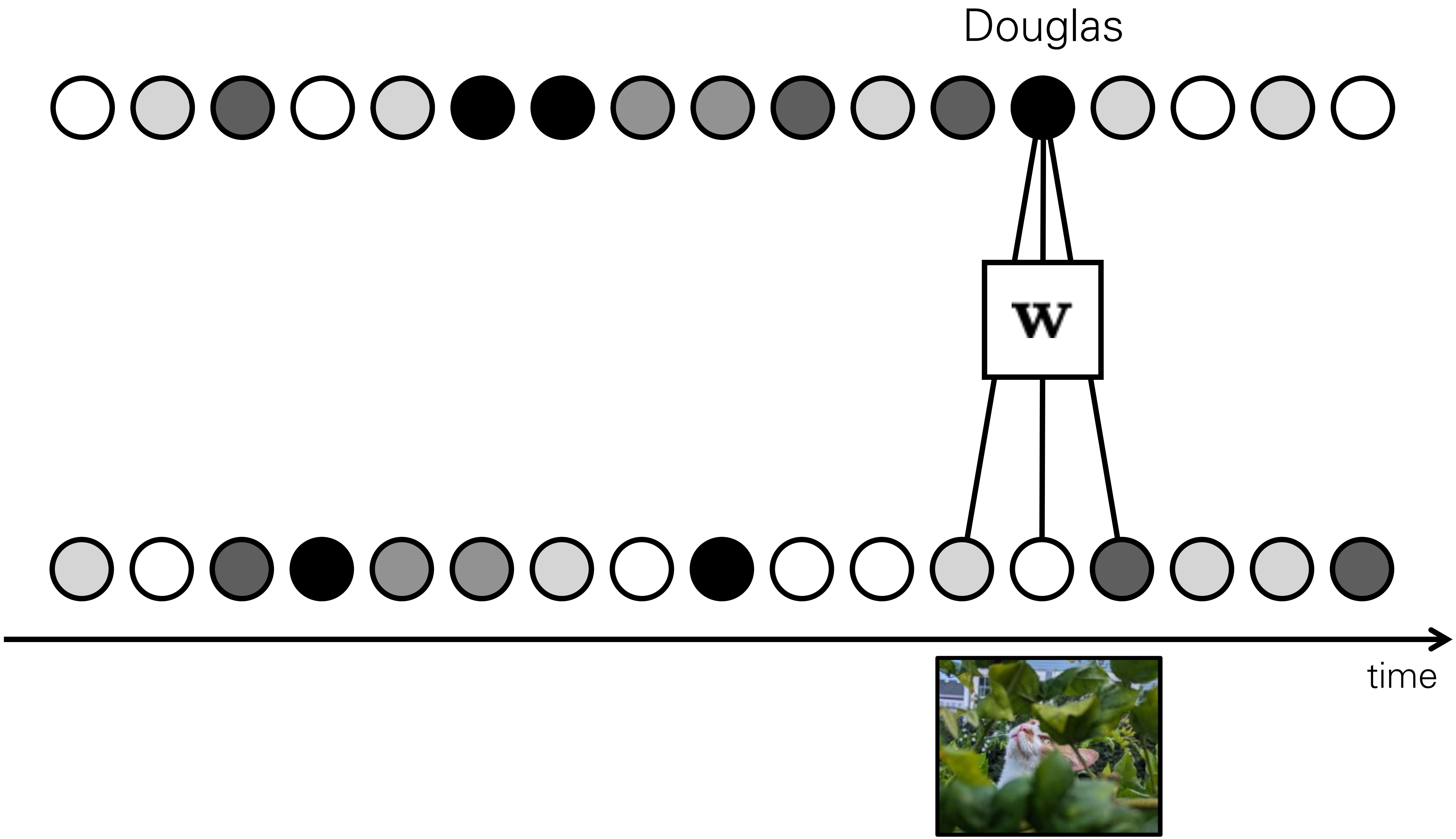


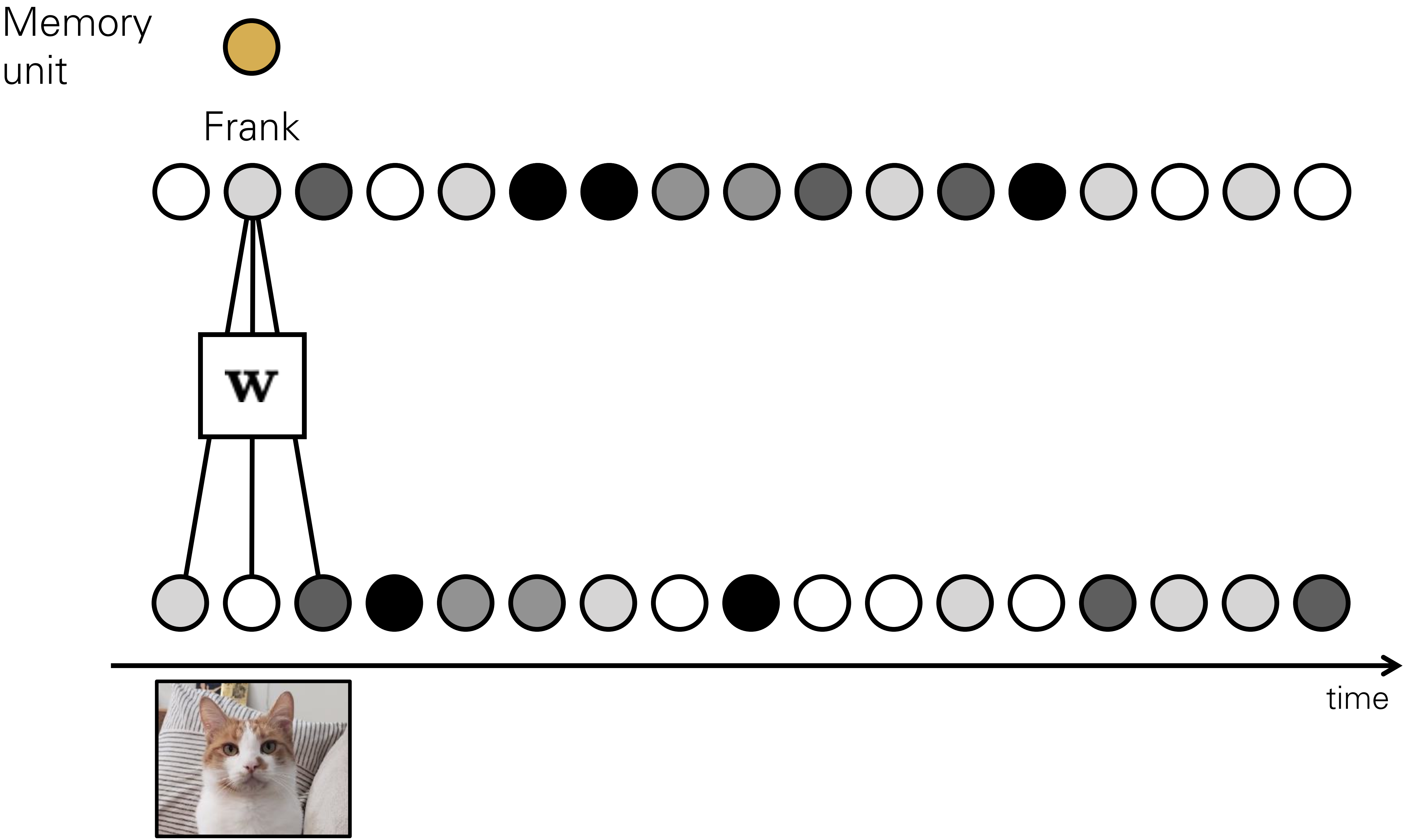

m
n
t

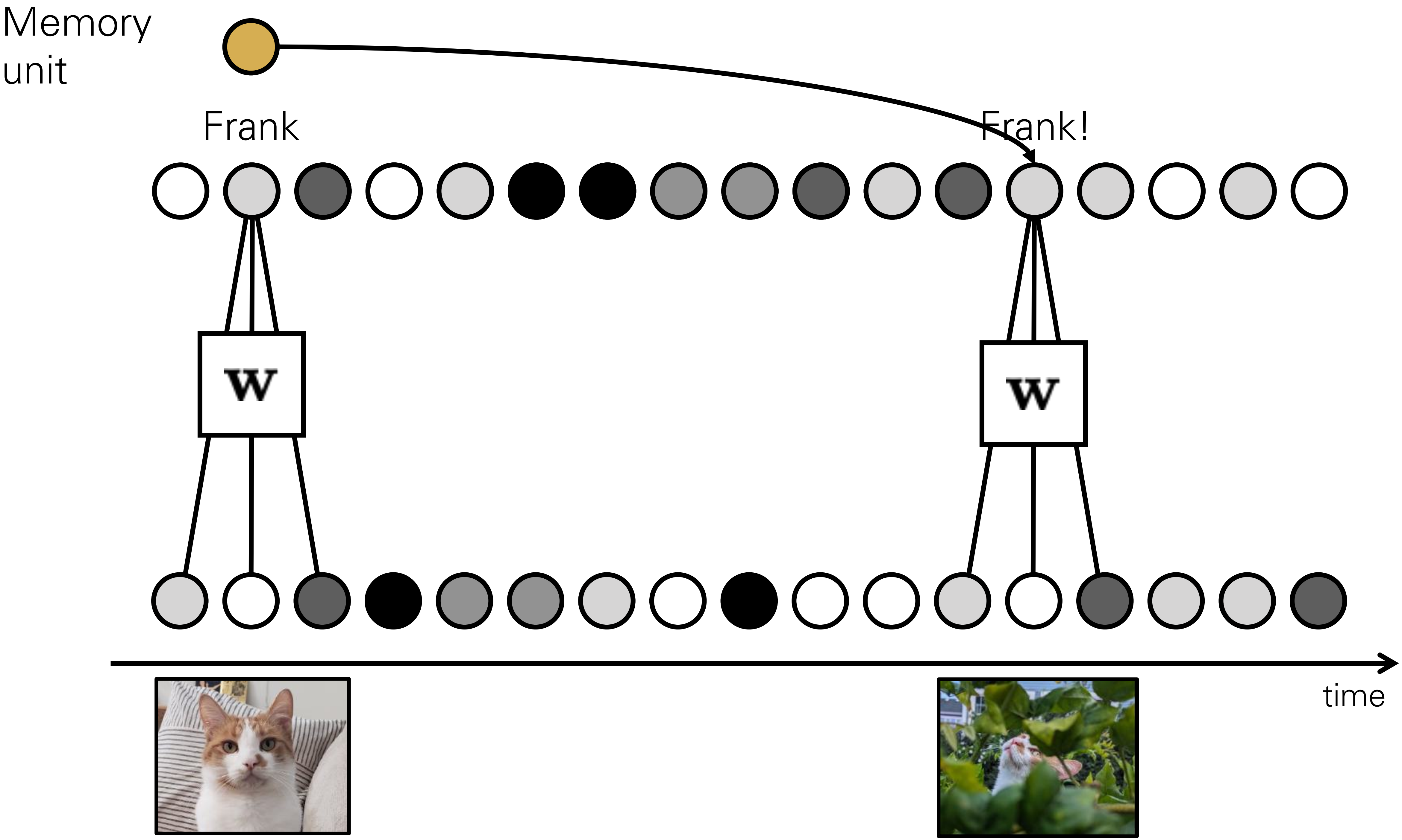


[fig from FeatureNet: Machining feature recognition based on 3D Convolution Neural Network]

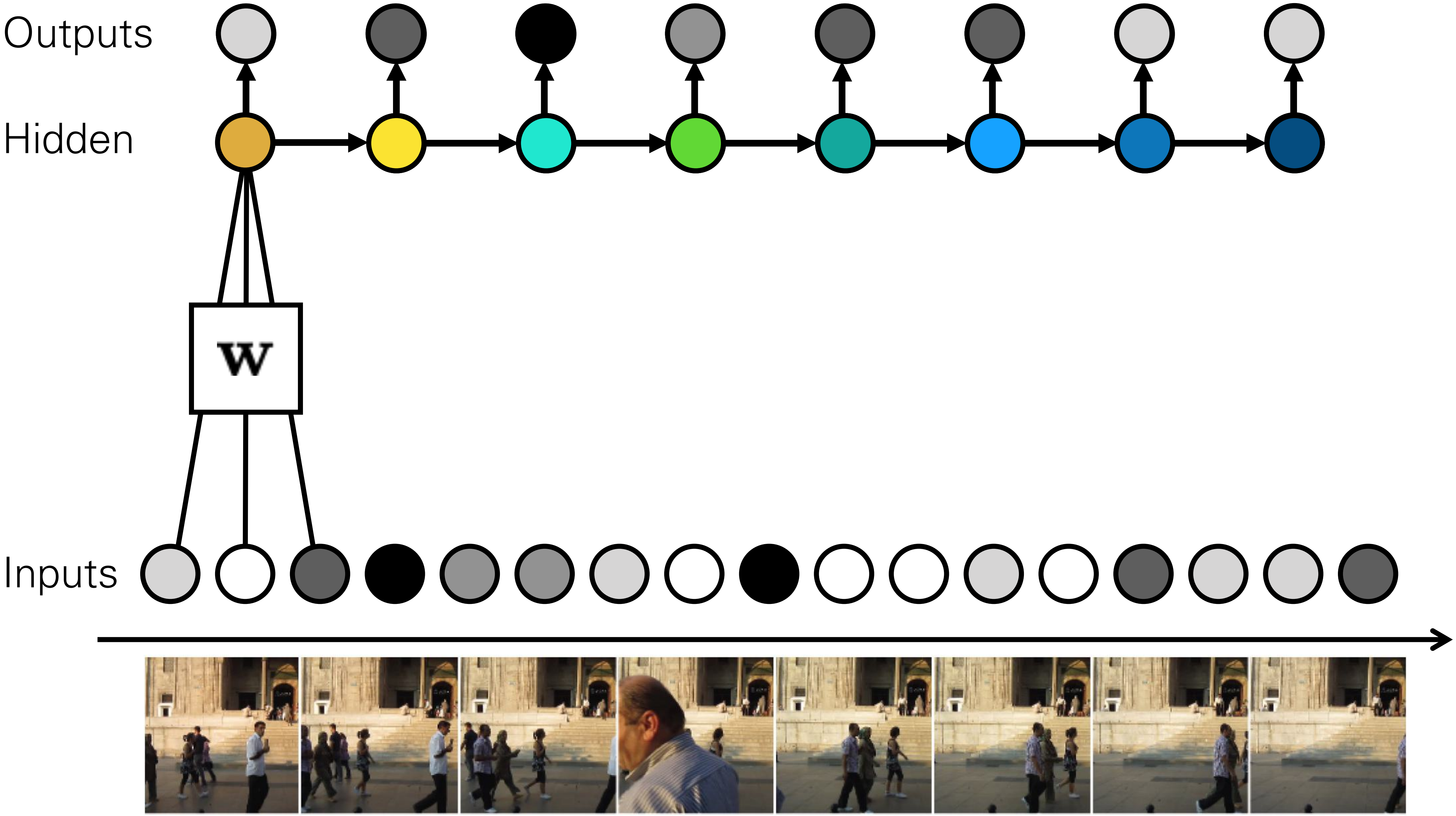

Frank
w
time


Douglas
w
time


Memory unit
Frank
w
time


Memory
unit
Frank
Frank!
w
w
time

Recurrent Neural Networks (RNNs)

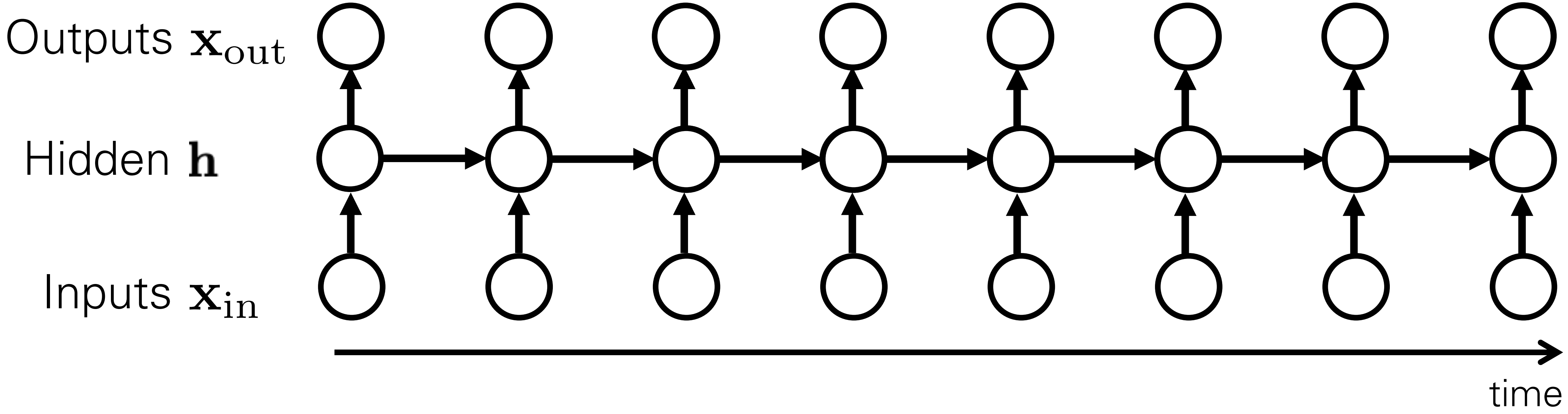


To model sequences, we need

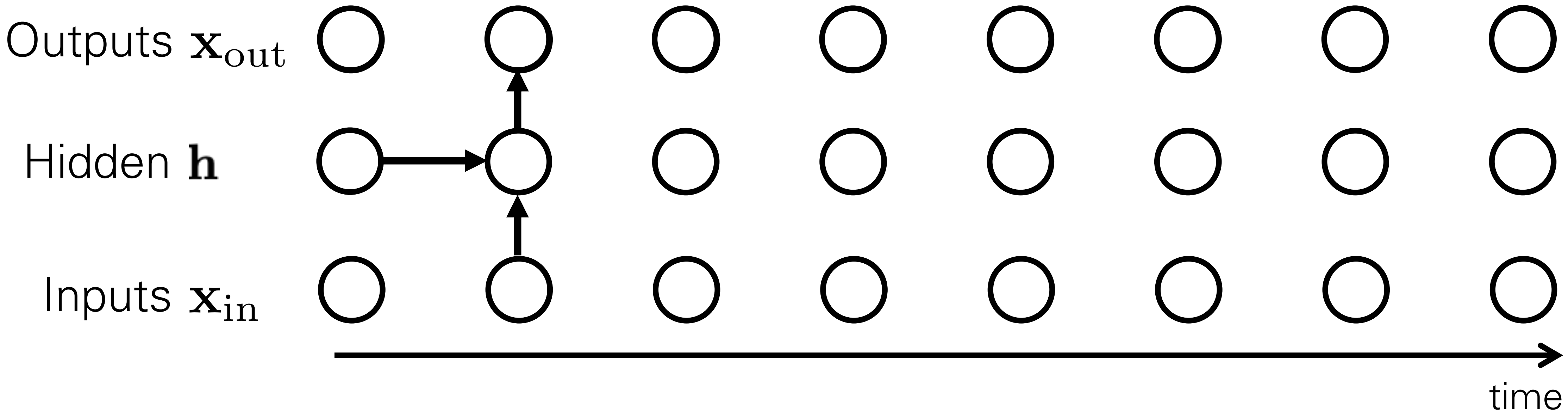
1. to deal with variable length sequences
2. to maintain sequence order
3. to keep track of long-term dependencies
4. to share parameters across the sequence

Recurrent Neural Networks

Recurrent Neural Networks (RNNs)



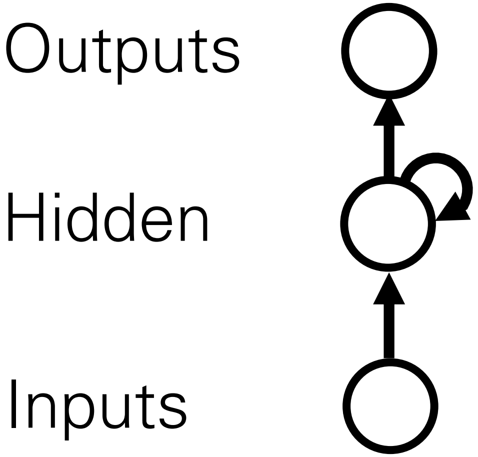
Recurrent Neural Networks (RNNs)



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

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

Recurrent Neural Networks (RNNs)

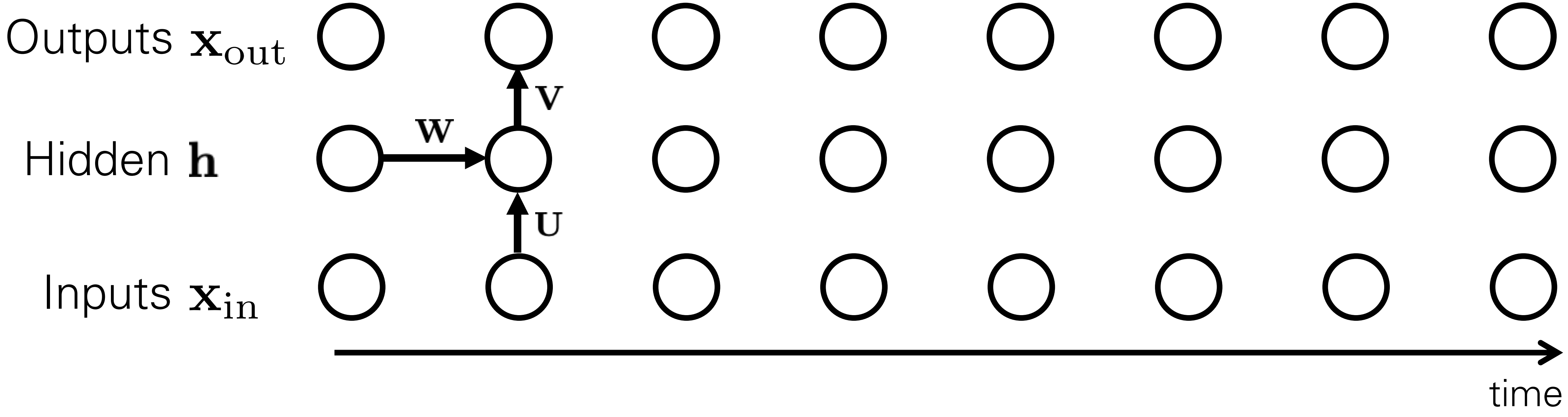


Recurrent!

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

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

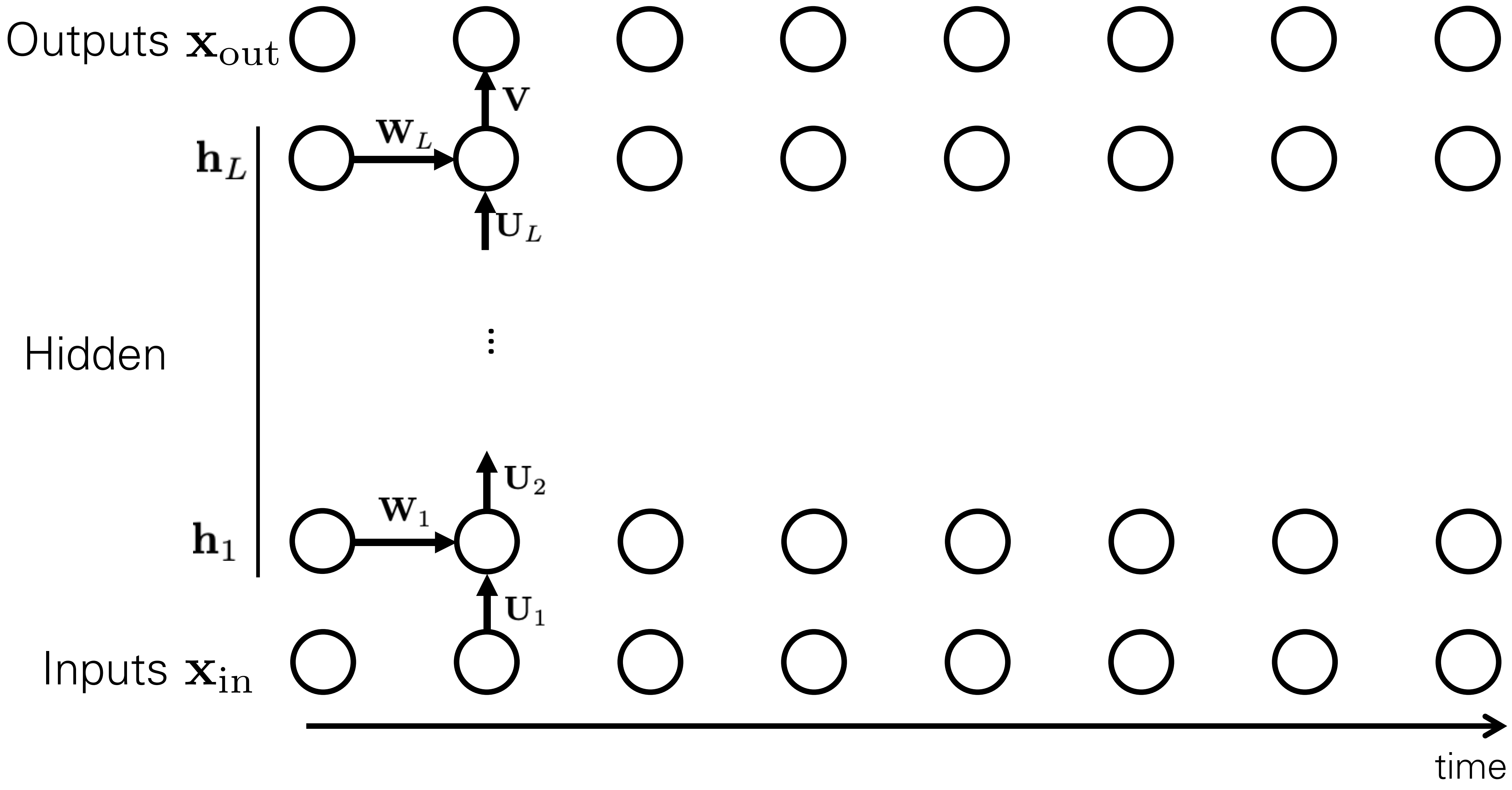
Recurrent Neural Networks (RNNs)



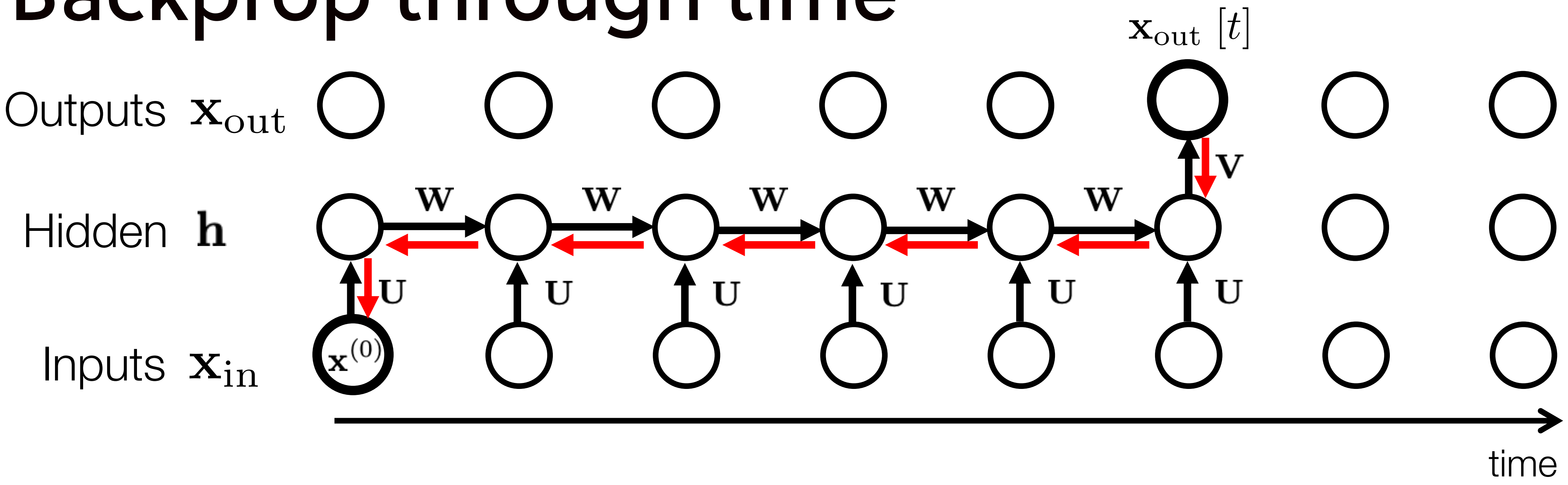
$$\mathbf{h}_t = \sigma_1 \left(\mathbf{W} \mathbf{h}_{t-1} + \mathbf{U} \mathbf{x}_{\text{in}} \left[t \right] + \mathbf{b} \right)$$

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

Deep Recurrent Neural Networks (RNNs)

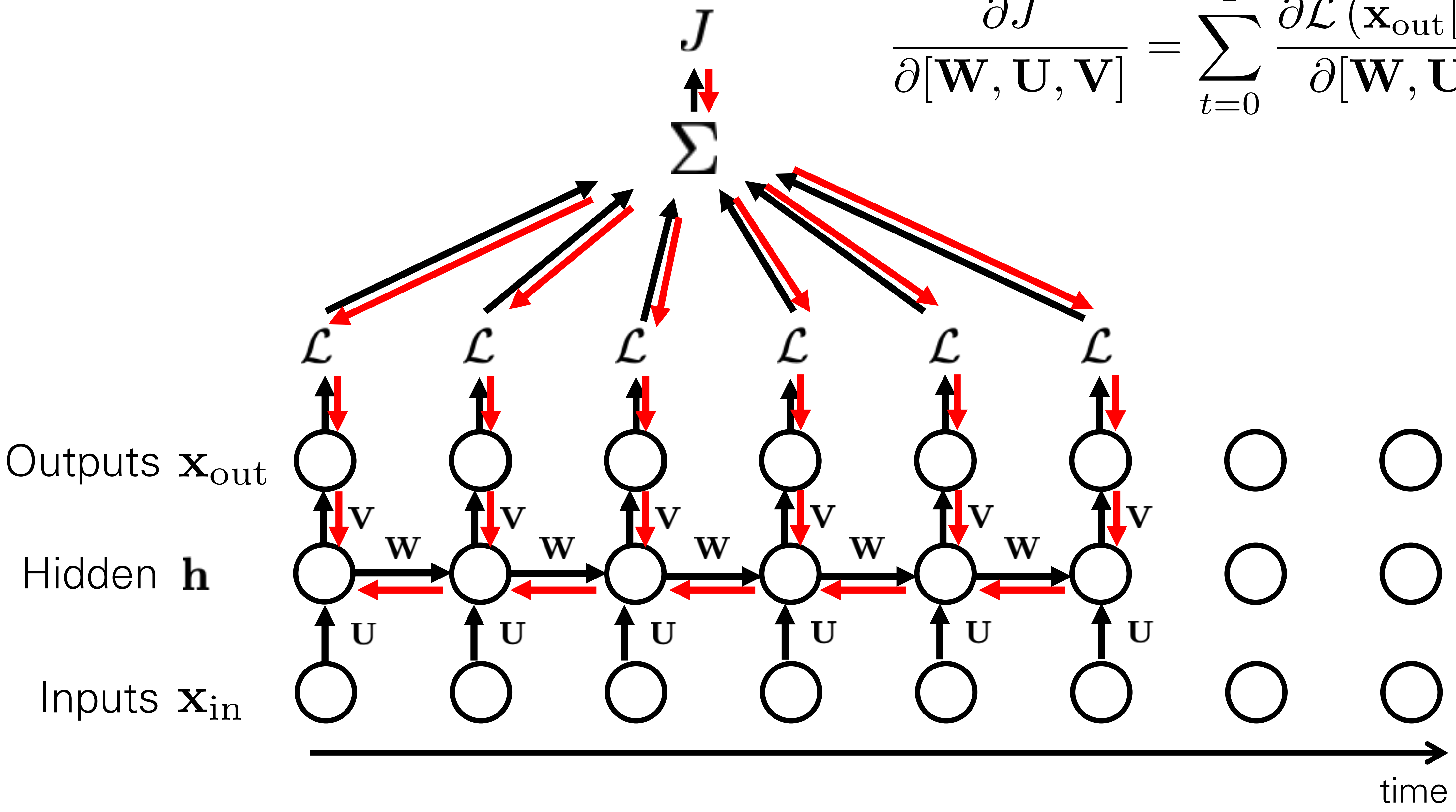


Backprop through time

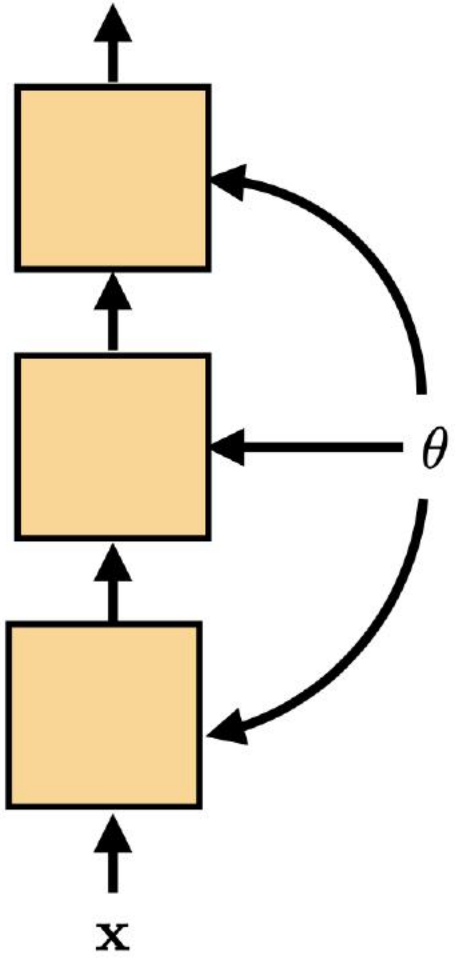


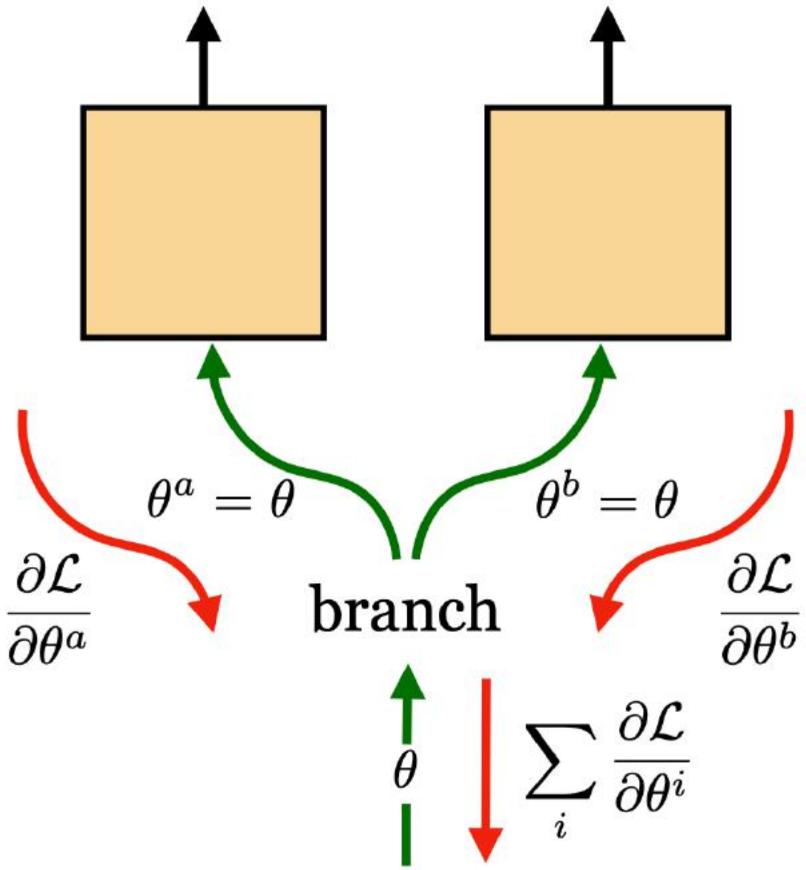
$$\frac{\partial \mathbf{x}_{\text{out}}[t]}{\partial \mathbf{x}_{\text{in}}[0]} = \frac{\partial \mathbf{x}_{\text{out}}[t]}{\partial \mathbf{h}_T} \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_{T-1}} \cdots \frac{\partial \mathbf{h}_1}{\partial \mathbf{h}_0} \frac{\partial \mathbf{h}_0}{\partial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

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

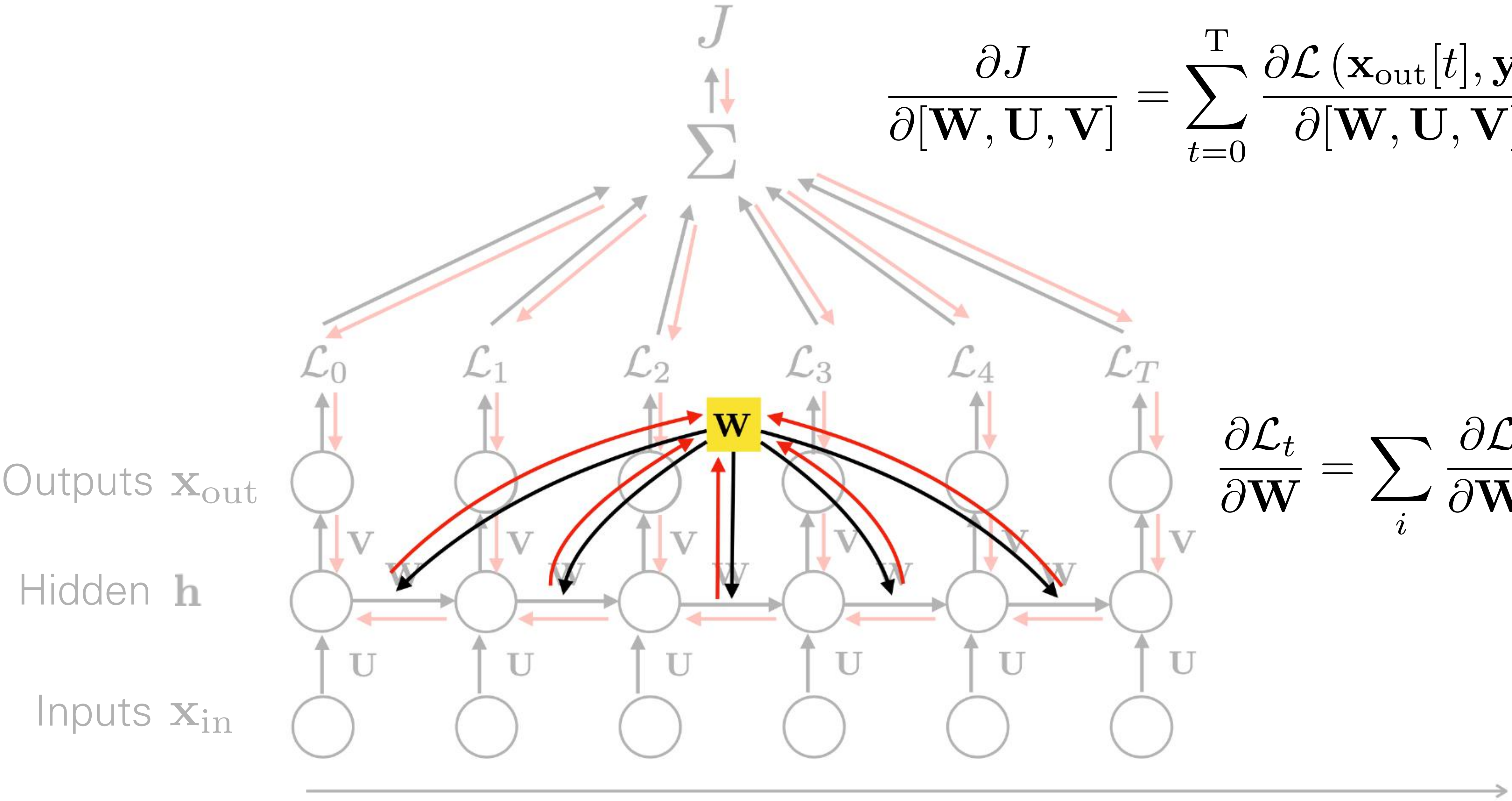


Parameter Sharing





Parameter sharing —> sum gradients



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

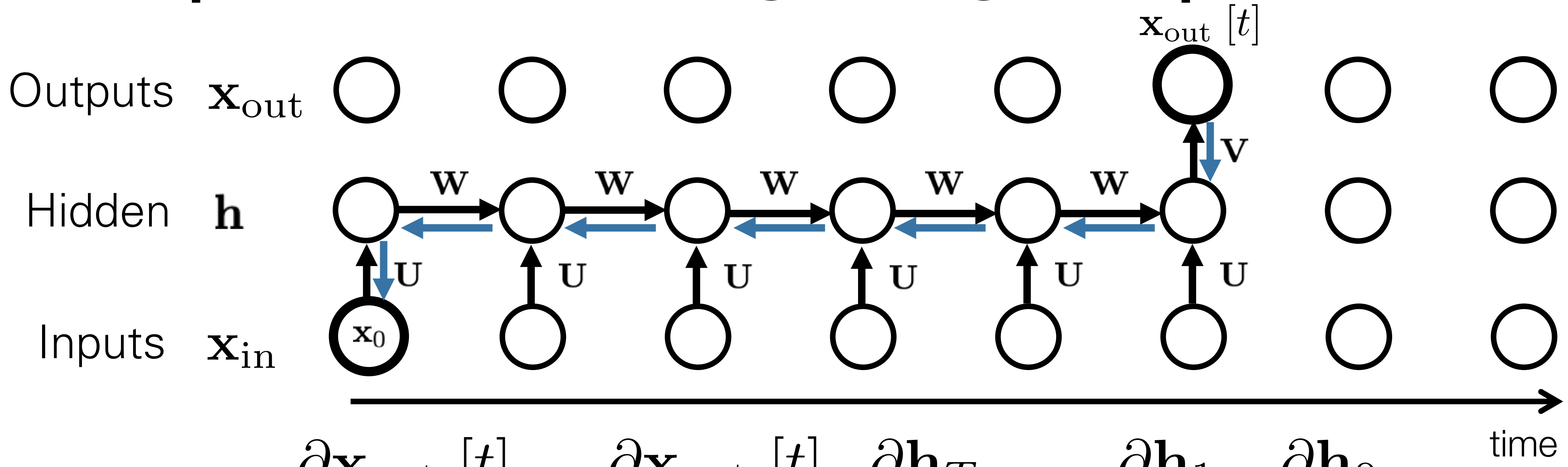
$$\frac{\partial\mathcal{L}_t}{\partial\mathbf{W}}=\sum_i\frac{\partial\mathcal{L}_t}{\partial\mathbf{W}^i}$$

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

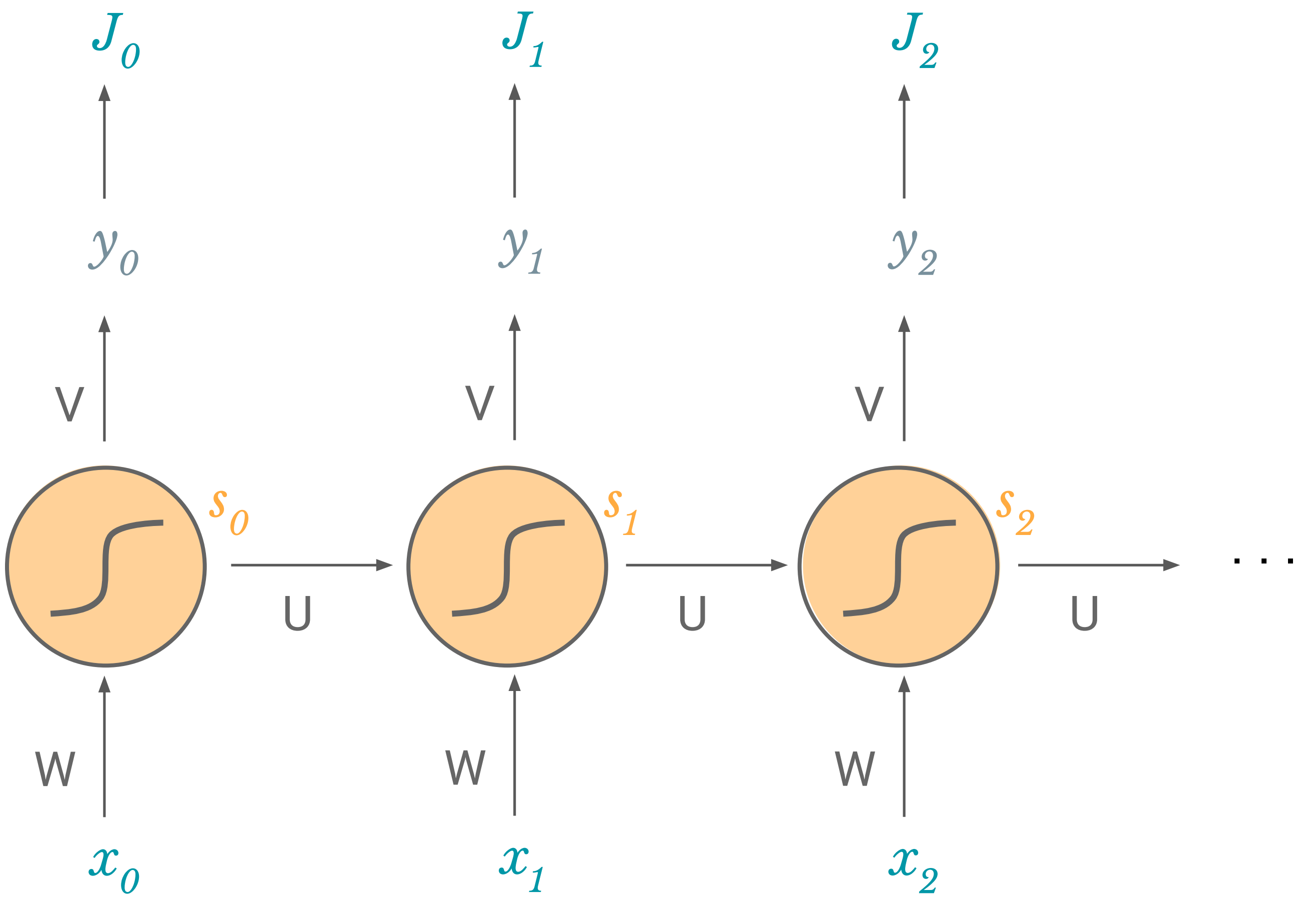
The problem of long-range dependencies



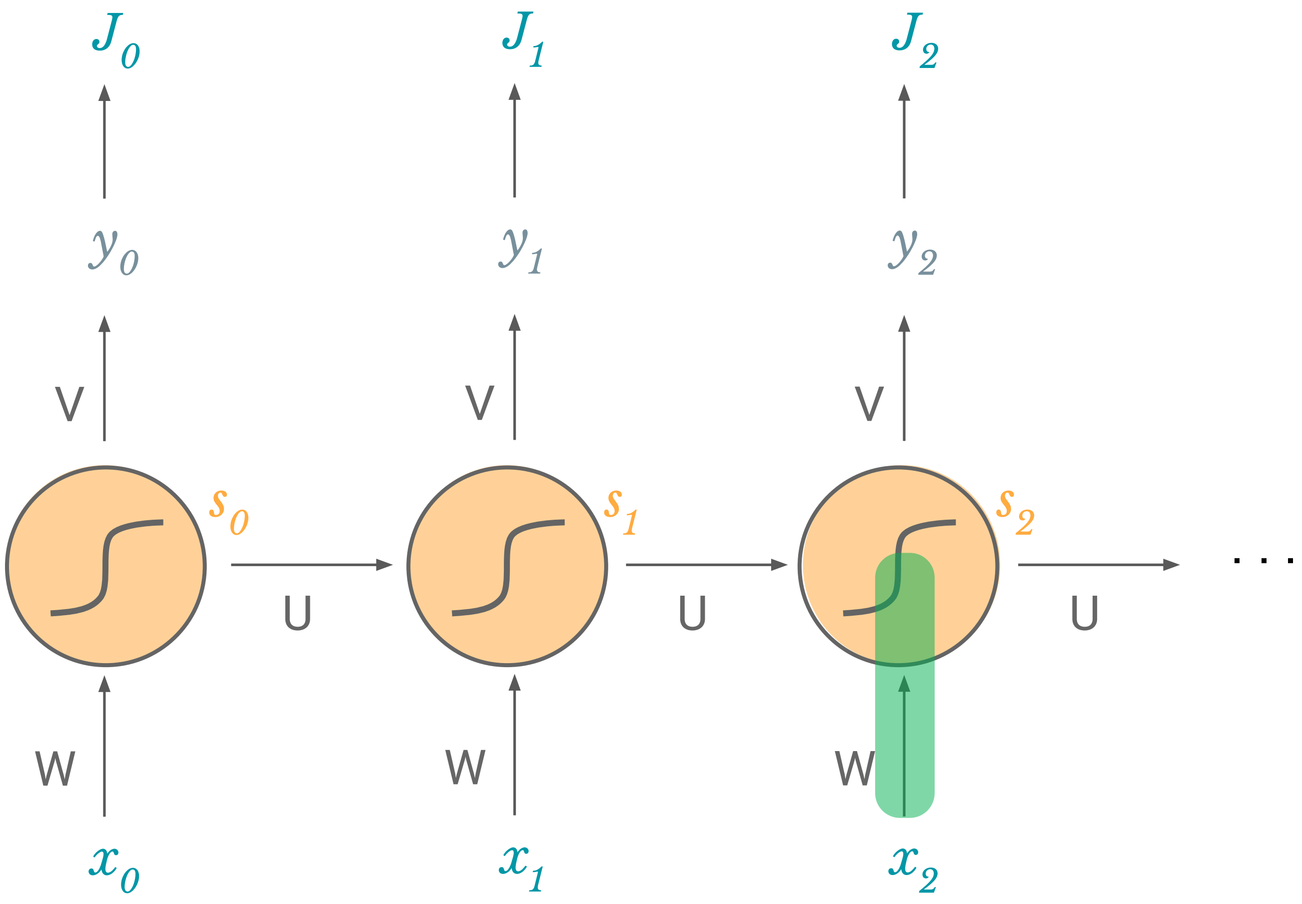
$$\frac{\partial \mathbf{x}_{\text{out}}[t]}{\partial \mathbf{x}_{\text{in}}[0]} = \frac{\partial \mathbf{x}_{\text{out}}[t]}{\partial \mathbf{h}_T} \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_{T-1}} \cdots \frac{\partial \mathbf{h}_1}{\partial \mathbf{h}_0} \frac{\partial \mathbf{h}_0}{\partial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

- 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 our for W with the **chain rule:**

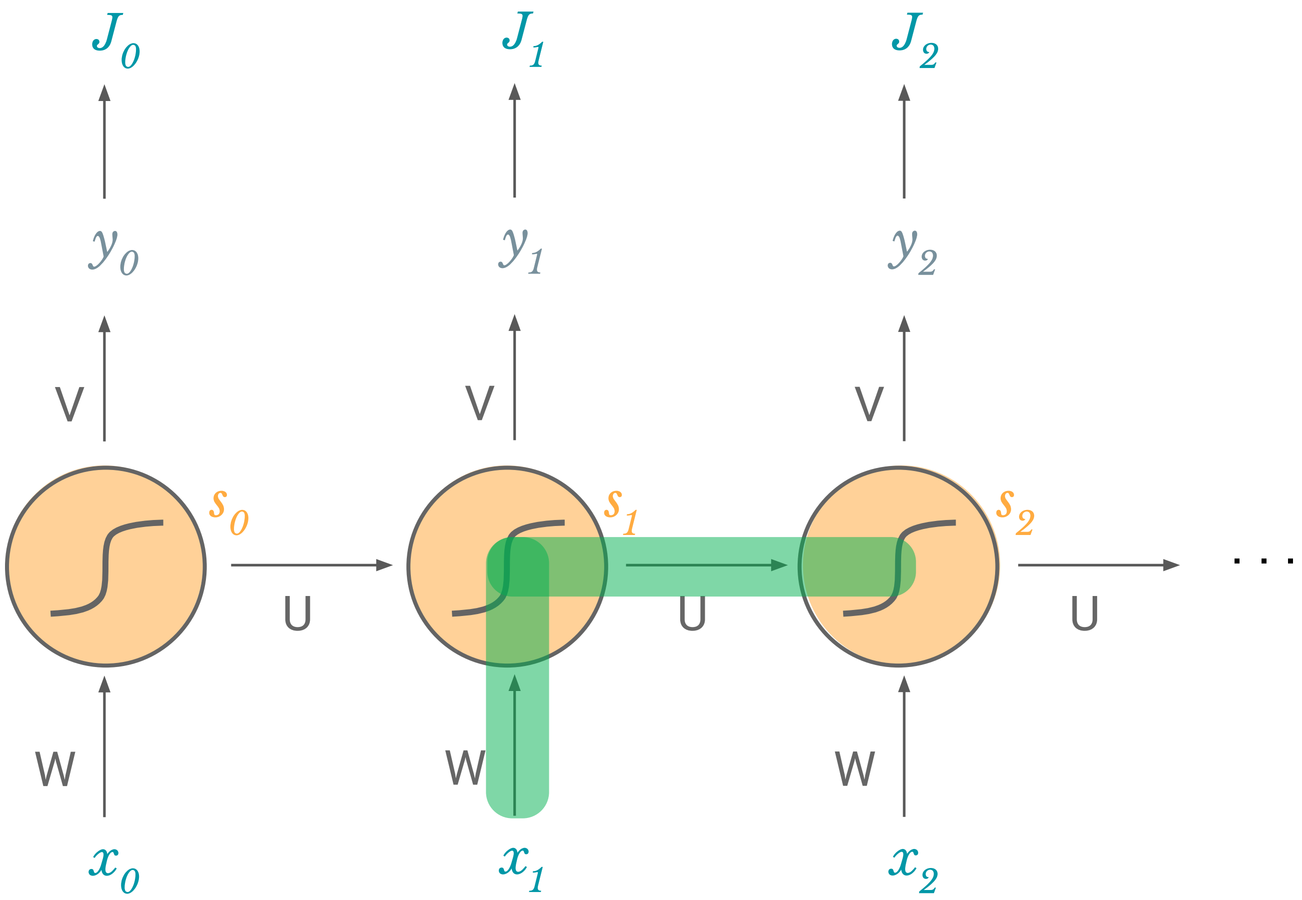


Let's try it our for W with the **chain rule:**



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

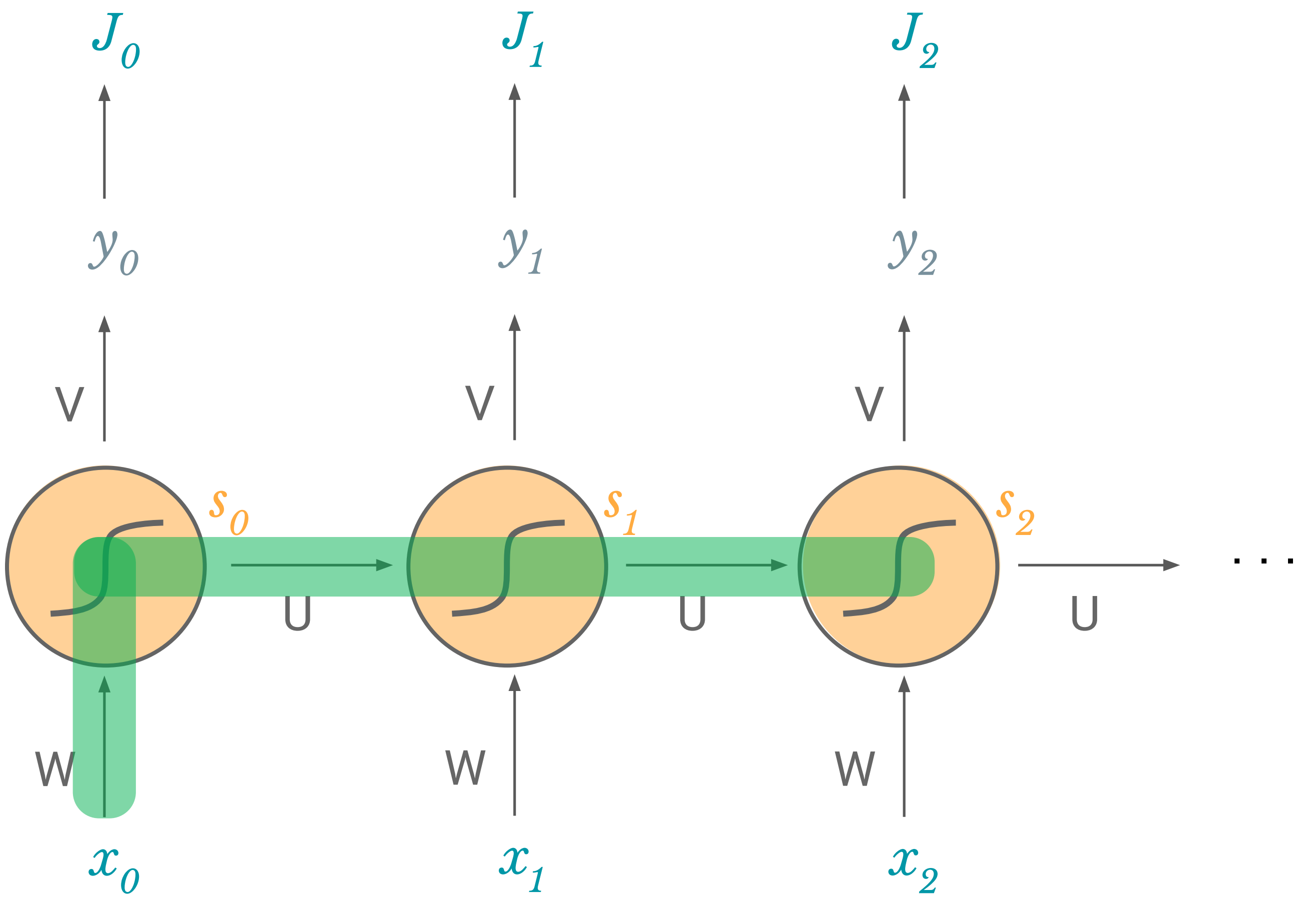
Let's try it our for W with the **chain rule:**



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

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

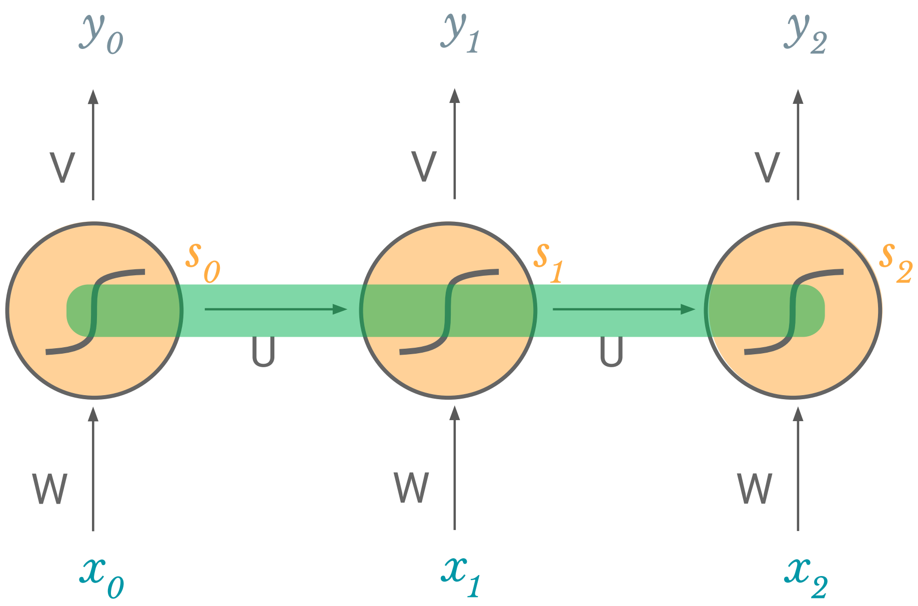
Let's try it our for W with the **chain rule:**



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

Vanishing Gradient Problem

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



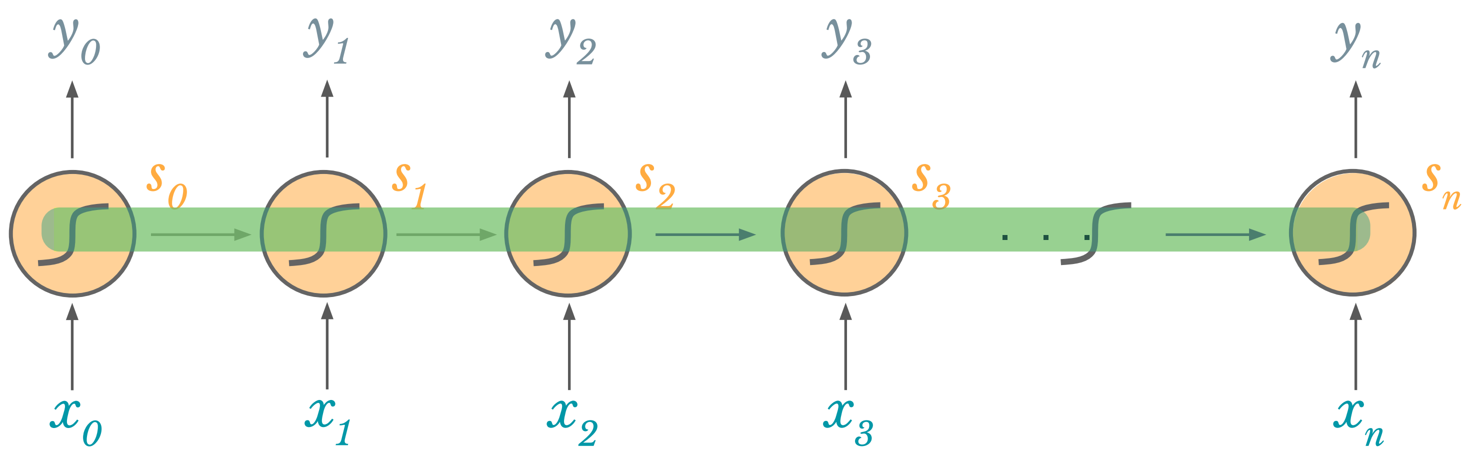
at k=0: $$\boxed{\frac{\partial s_2}{\partial s_0} = \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}}$$

Vanishing Gradient Problem

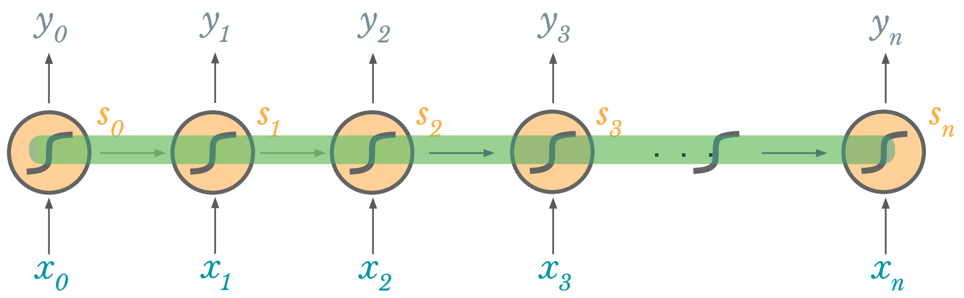
$$\frac{\partial J_n}{\partial W} = \sum_{k=0}^{n} \frac{\partial J_n}{\partial y_n} \frac{\partial y_n}{\partial s_n} \boxed{\frac{\partial s_n}{\partial s_k}} \frac{\partial s_k}{\partial W}$$

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

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



Vanishing Gradient Problem



what are each of these terms?

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

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

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?

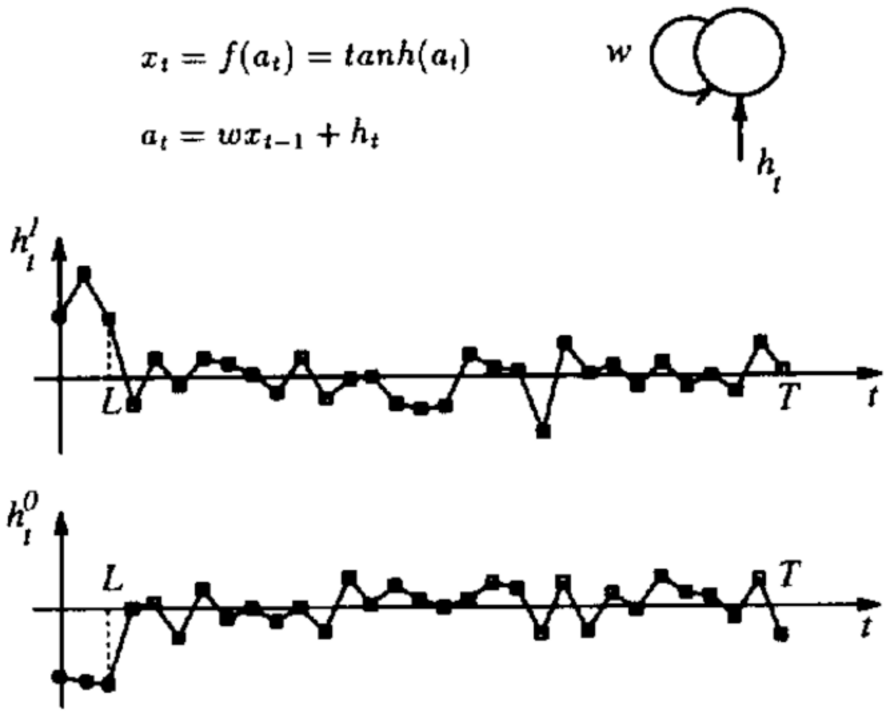
errors due to further back timesteps have increasingly **smaller gradients**.

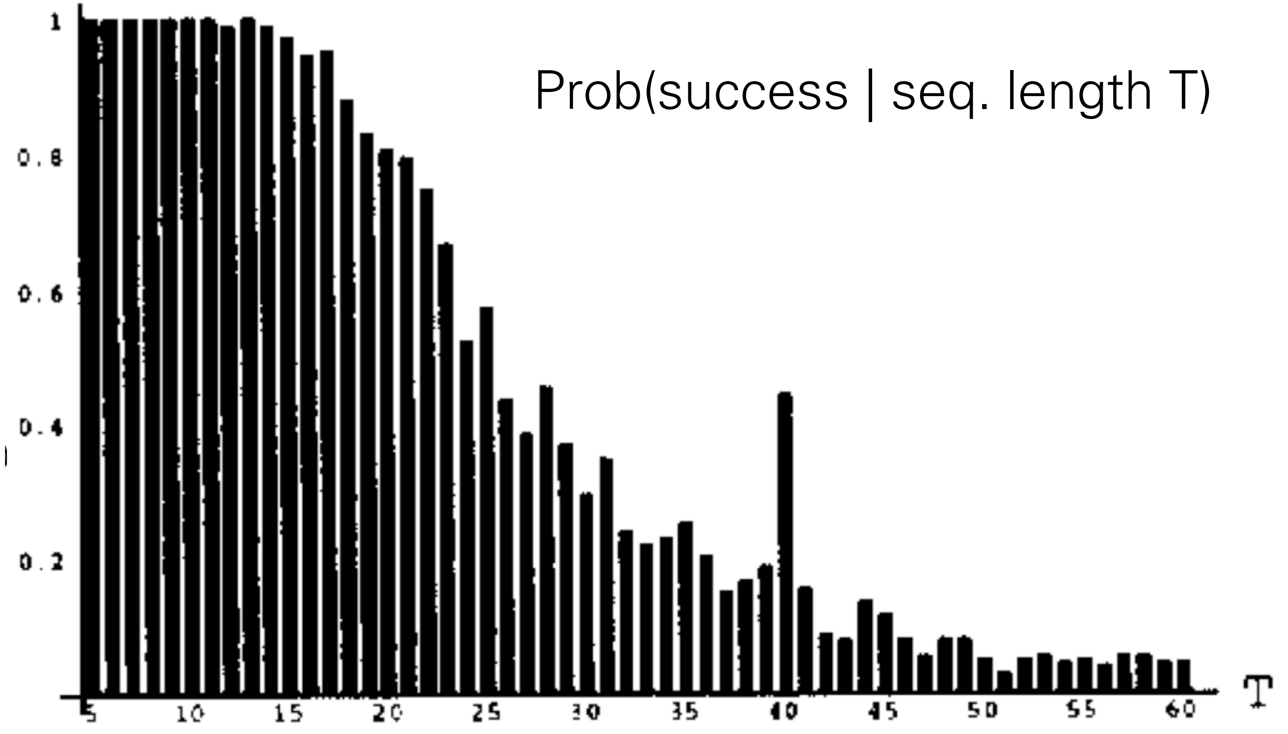
so what?

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?





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

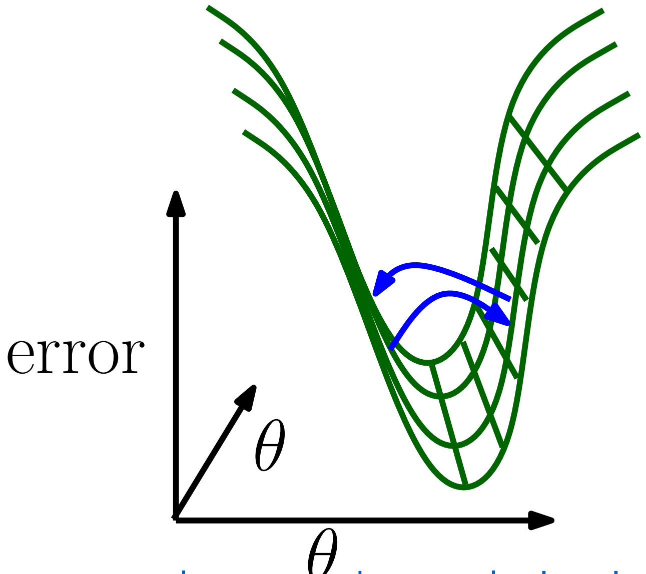
Gradient Norm Clipping

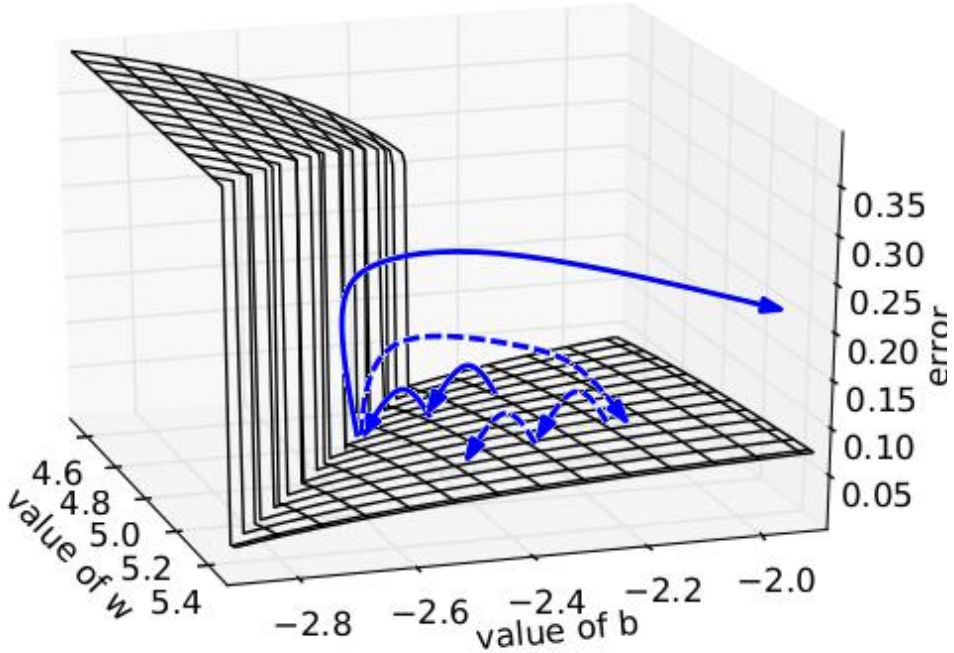
$$\hat{\mathbf{g}} \leftarrow \frac{\partial error}{\partial \theta}$$

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

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

$$\textbf{end if}$$





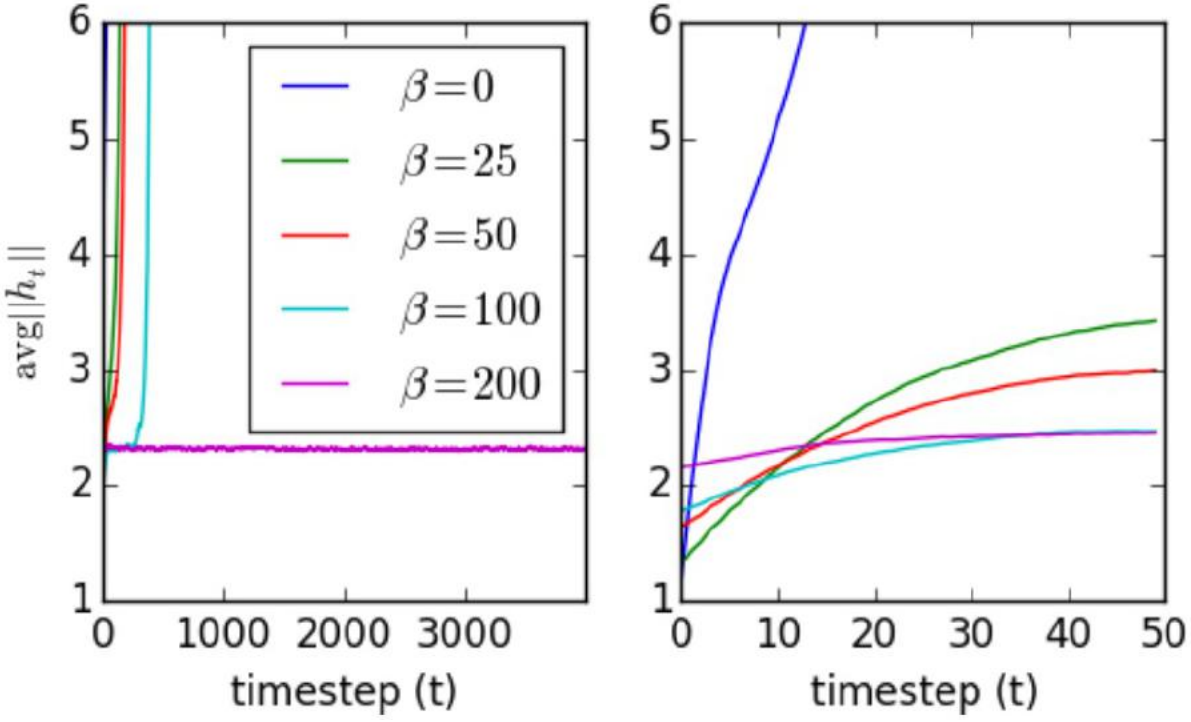
Recurrent neural network regularization. Zaremba et al., arXiv 2014.

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

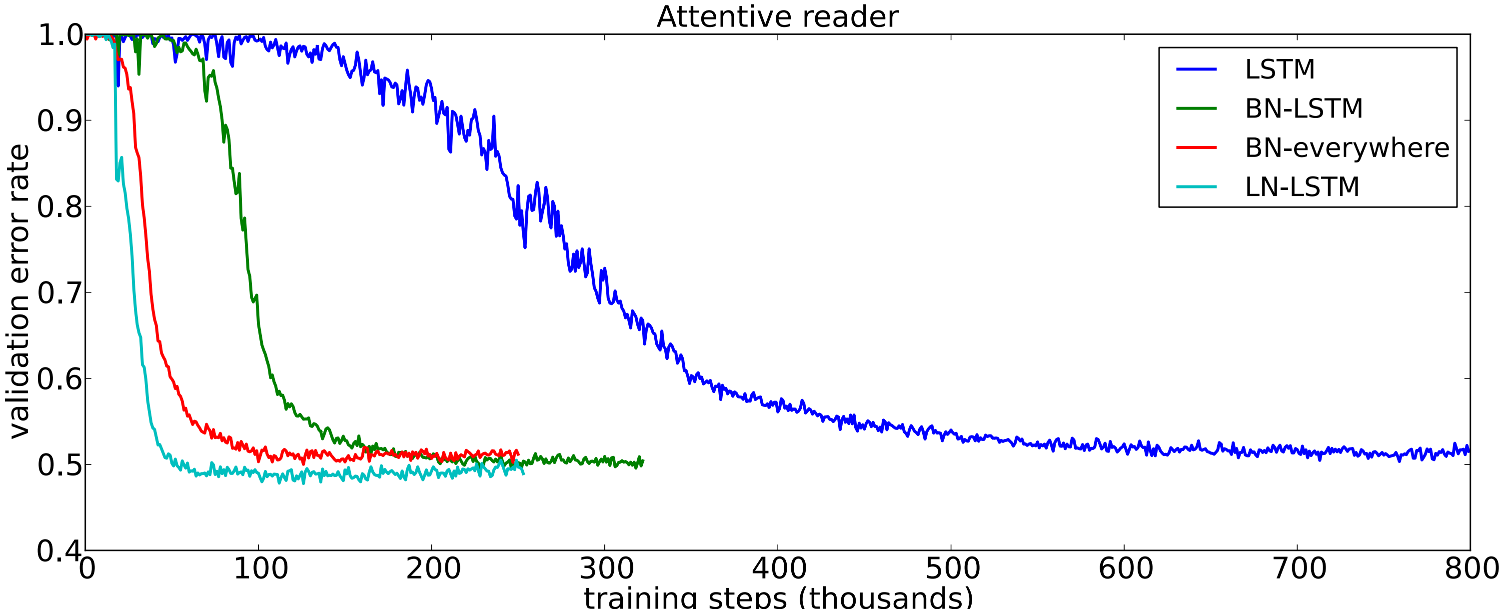
Regularization: Layer Normalization

- 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

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

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

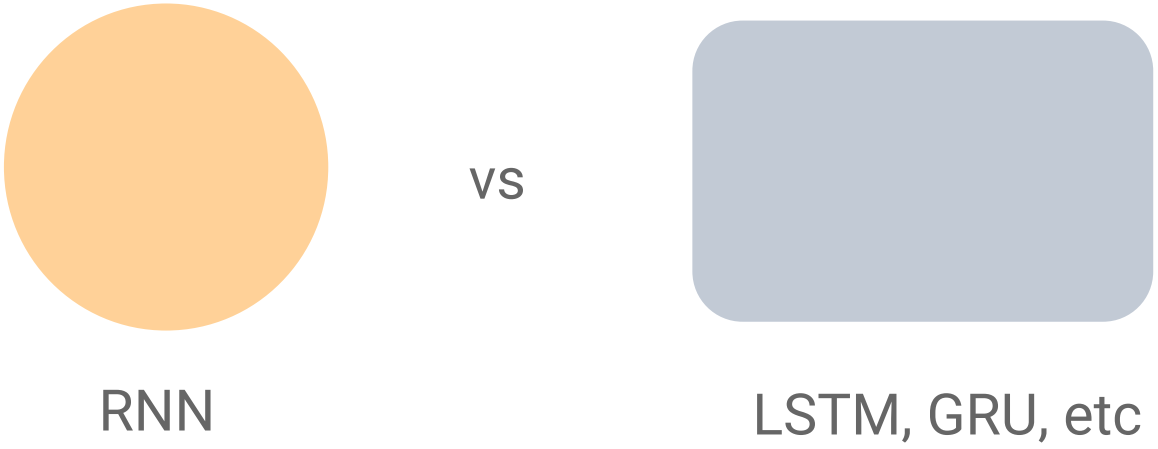
$$\sigma^t = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^t - \mu^t\right)^2}$$



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.

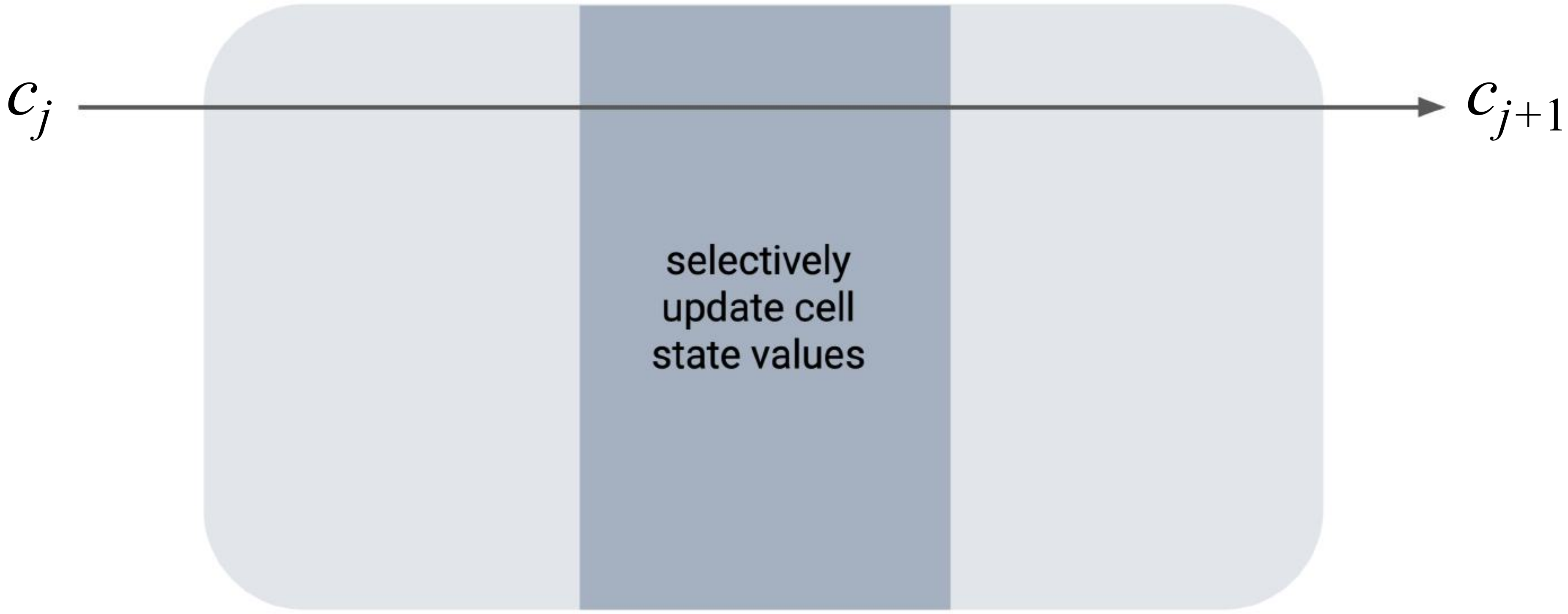
Long Short-Term Memory (LSTM)

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

Long Short-Term Memory (LSTM)

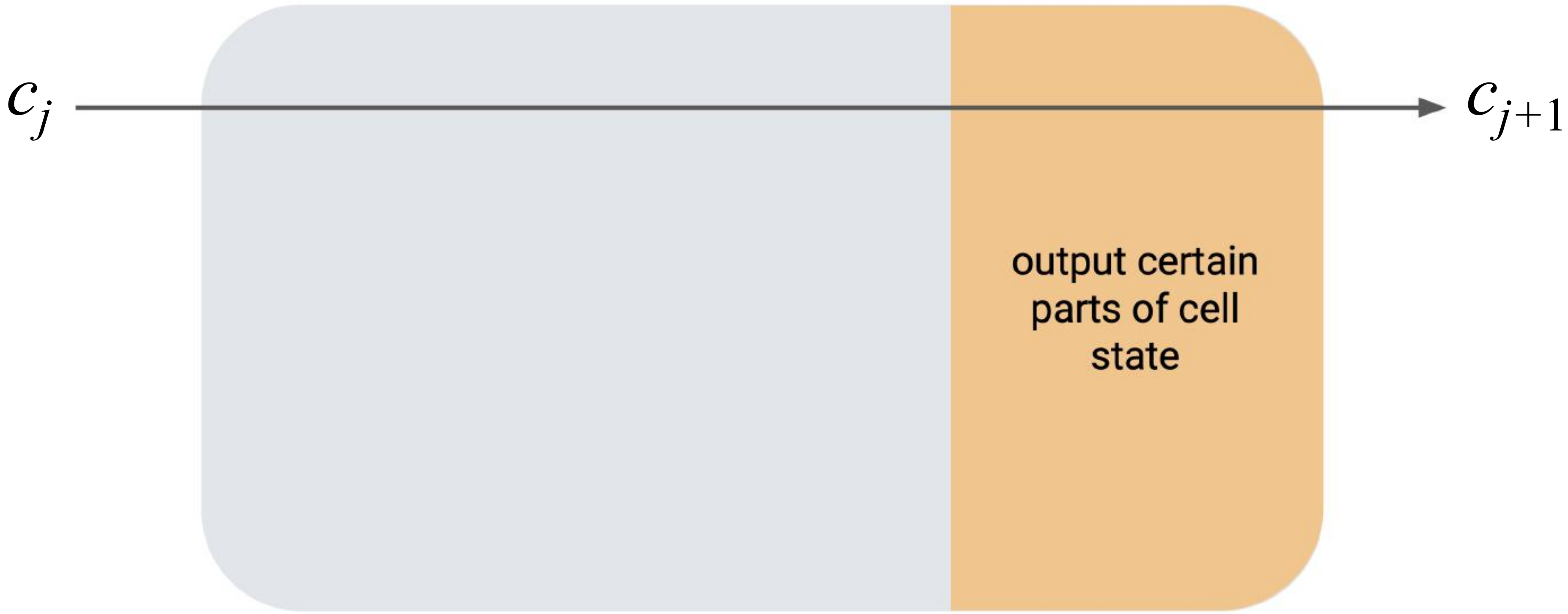
Long Short-Term Memory [Hochreiter et al., 1997]

Long Short-Term Memory (LSTM)



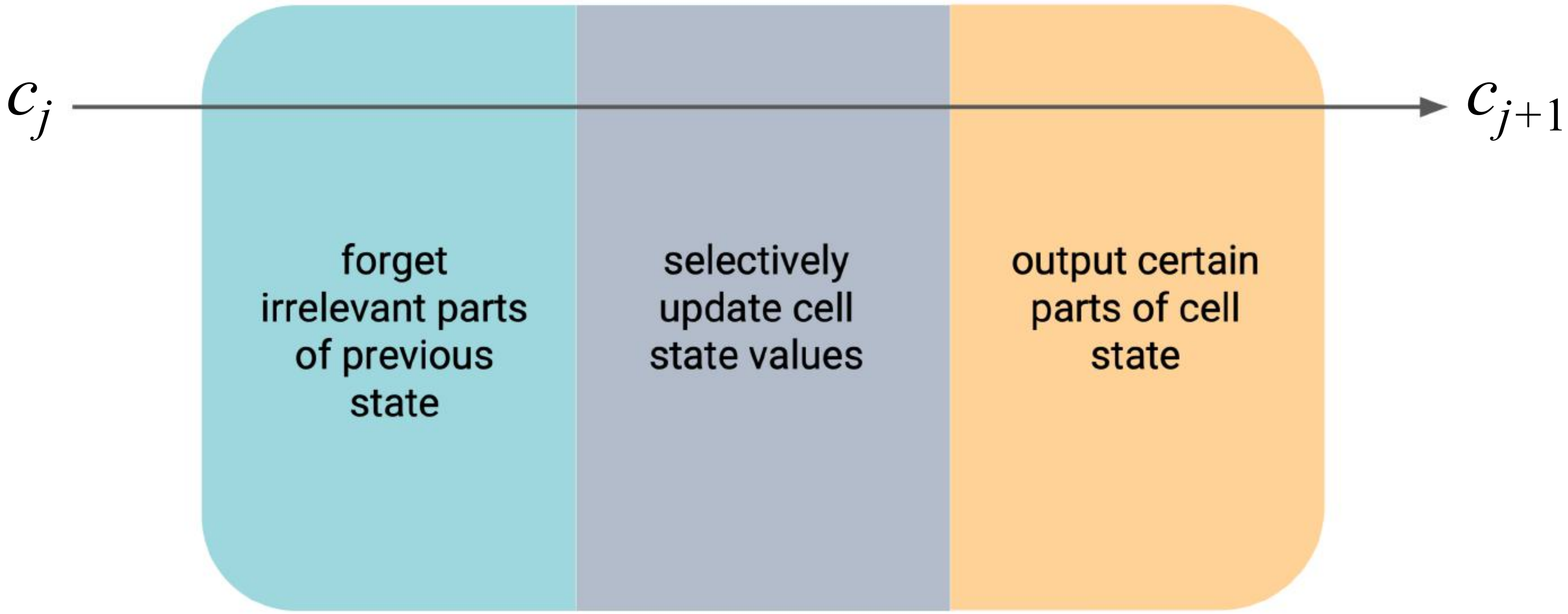
Long Short-Term Memory [Hochreiter et al., 1997]

Long Short-Term Memory (LSTM)



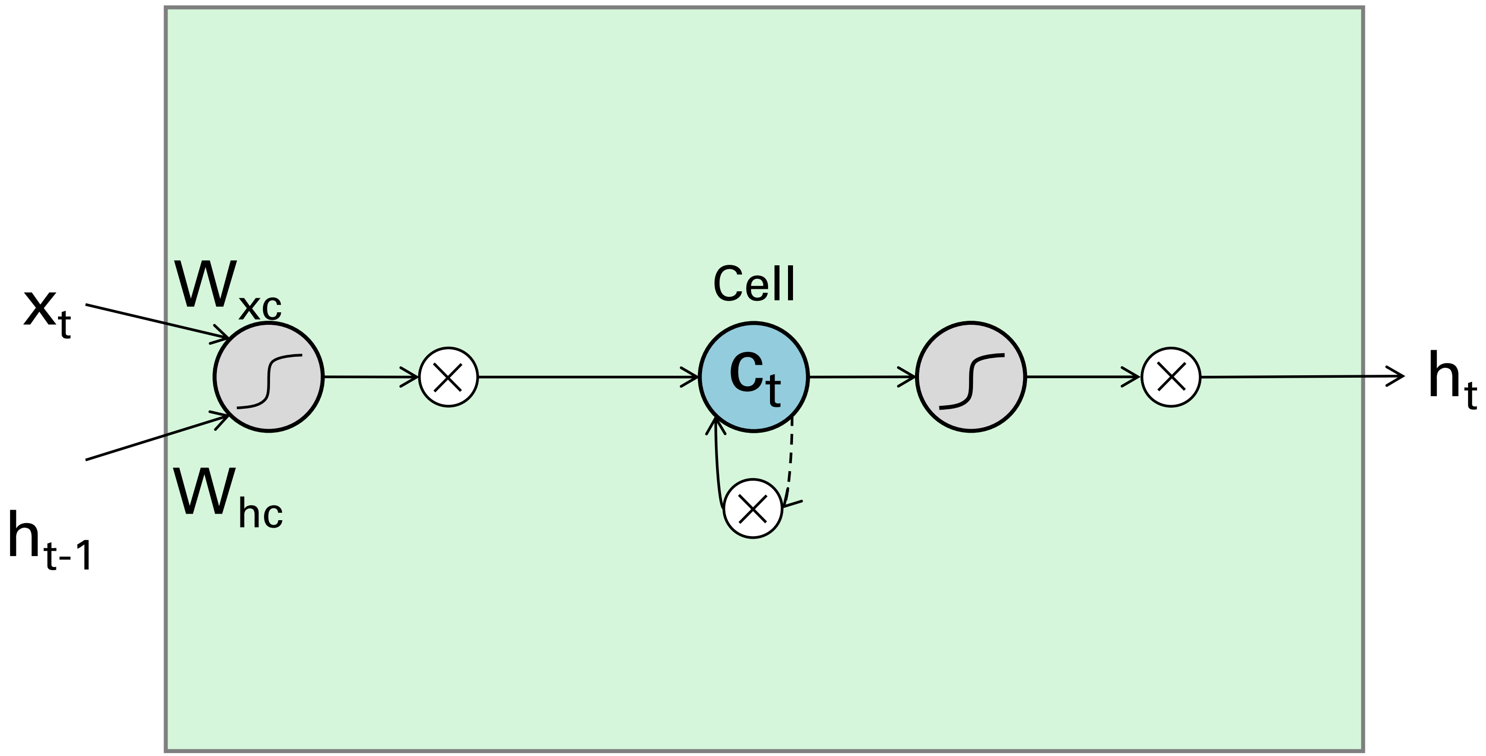
[Long Short-Term Memory [Hochreiter et al., 1997]]

Long Short-Term Memory (LSTM)



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

The LSTM Idea

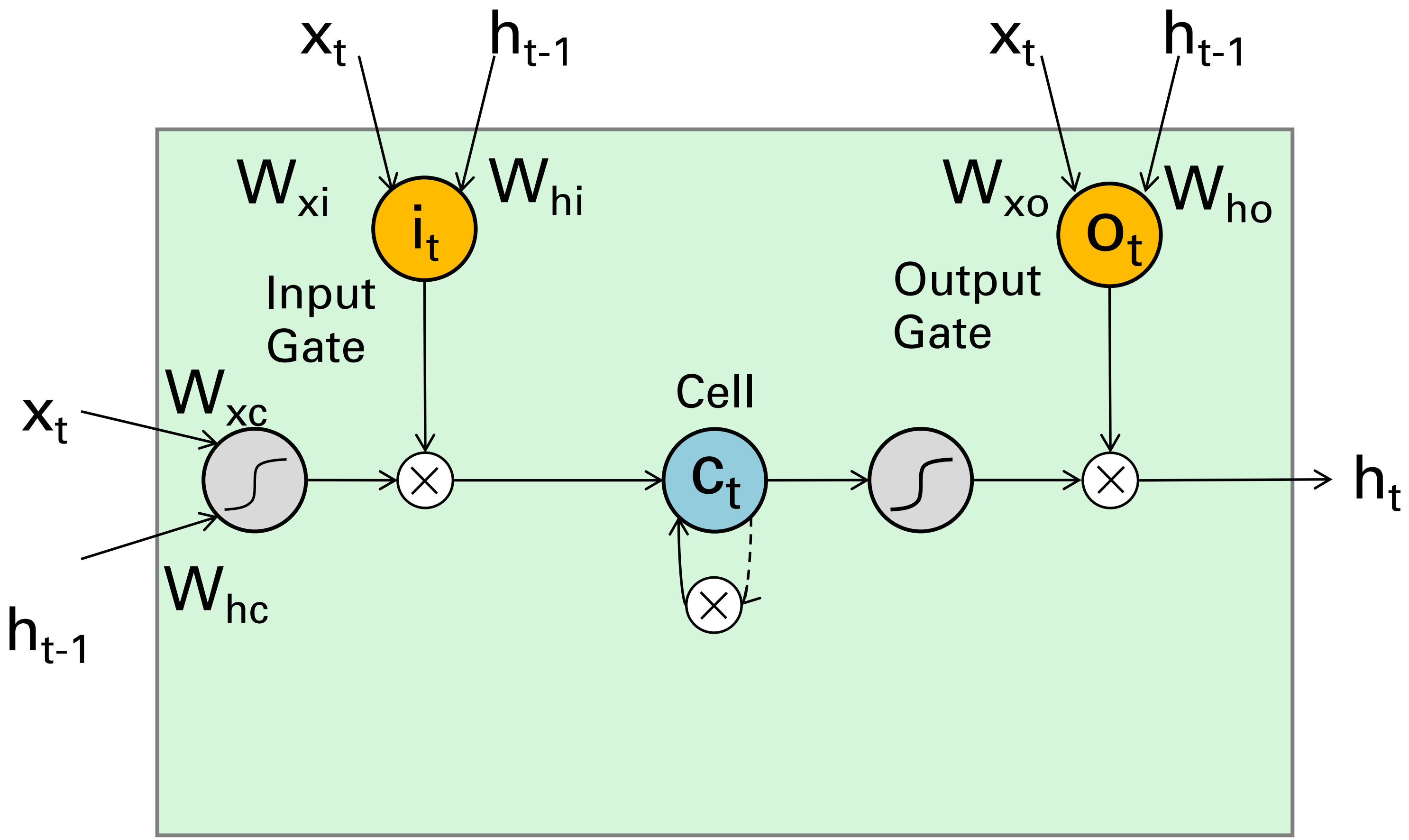


* Dashed line indicates time-lag

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

$$h_t = \tanh c_t$$

The Original LSTM Cell



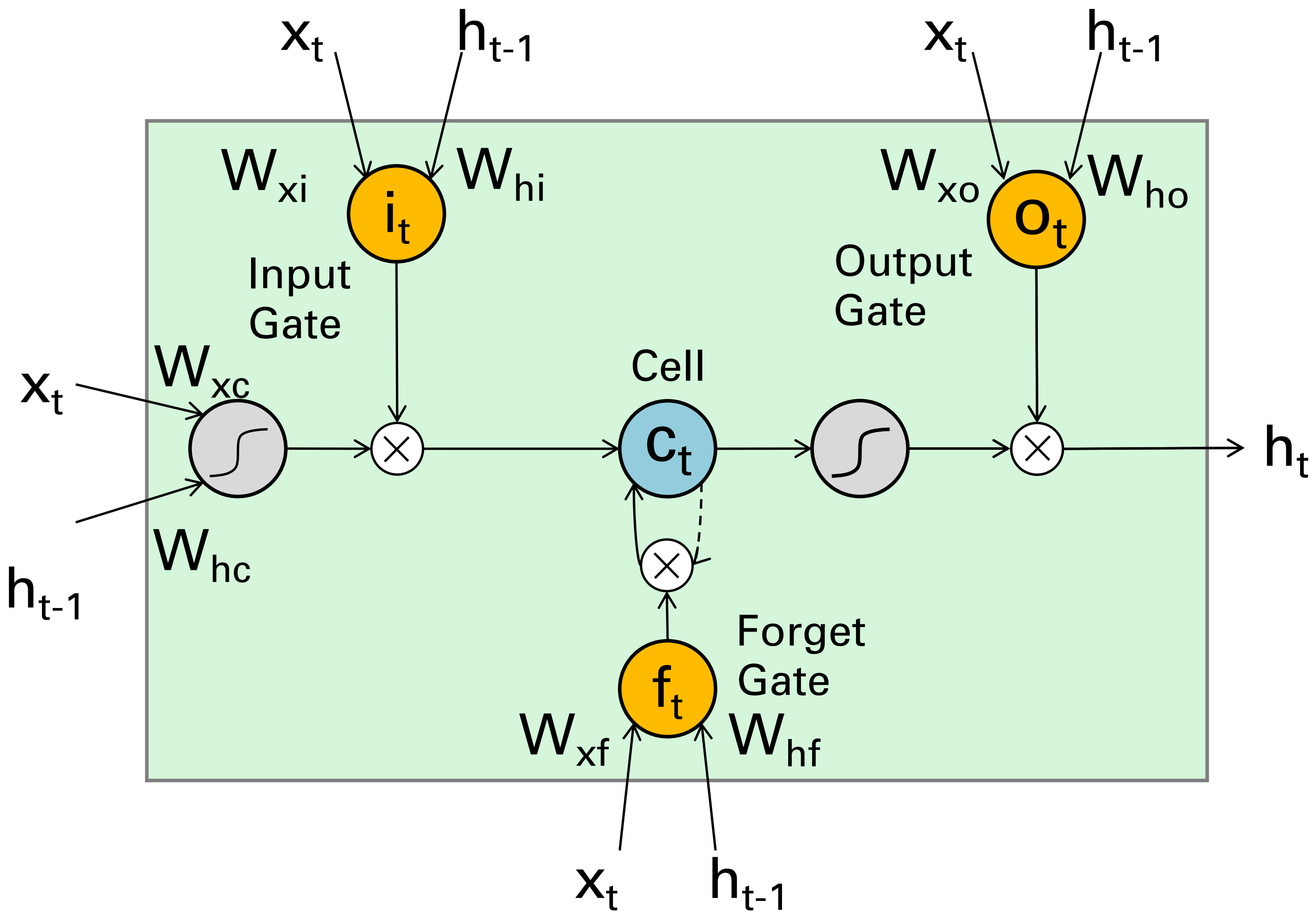
$$c_t = c_{t-1} + i_t \otimes \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

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

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

Similarly for o_t

The Popular LSTM Cell



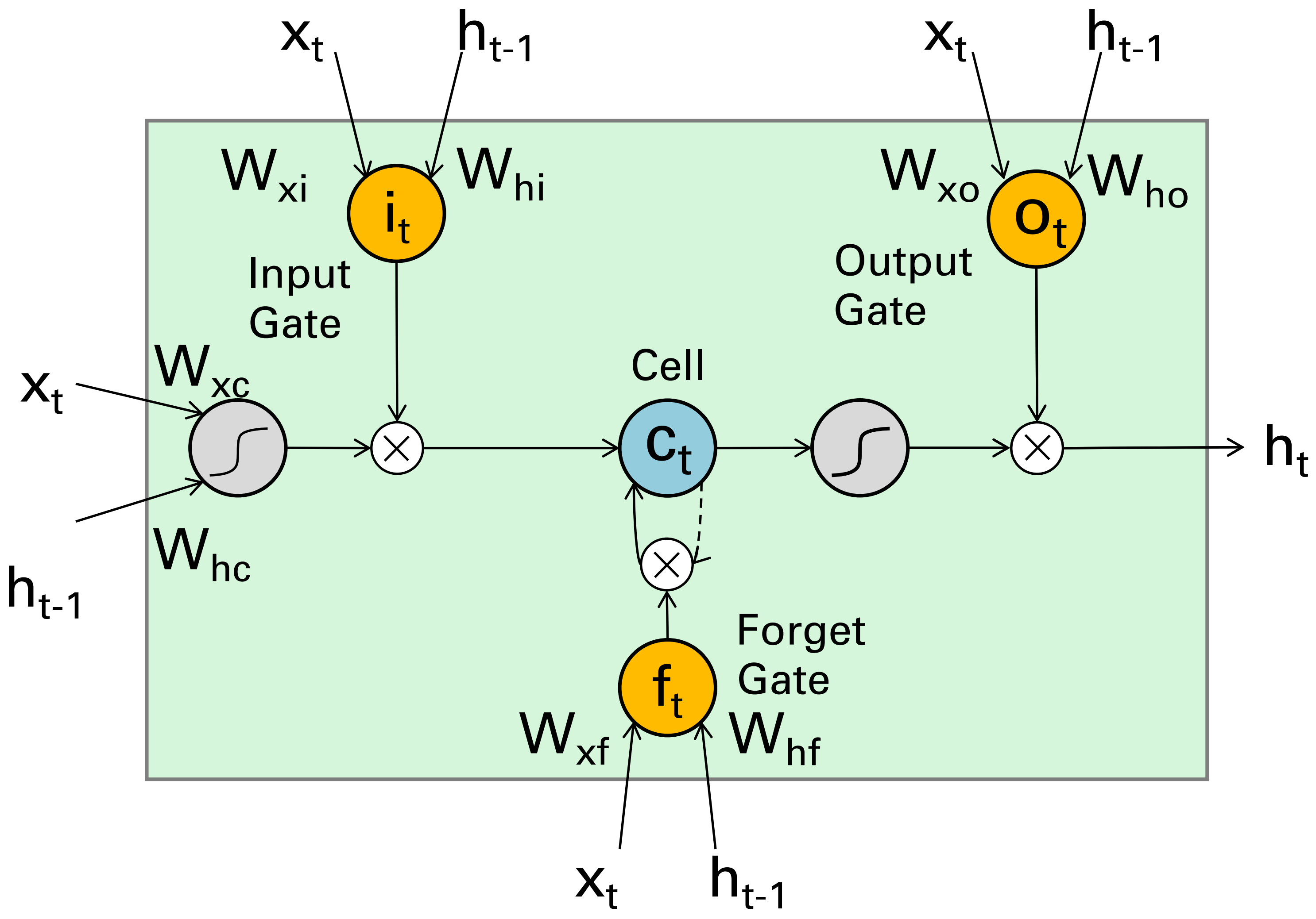
$$i_t = \sigma\left(W_i\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix}+b_i\right)$$

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

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

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

The Popular LSTM Cell



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

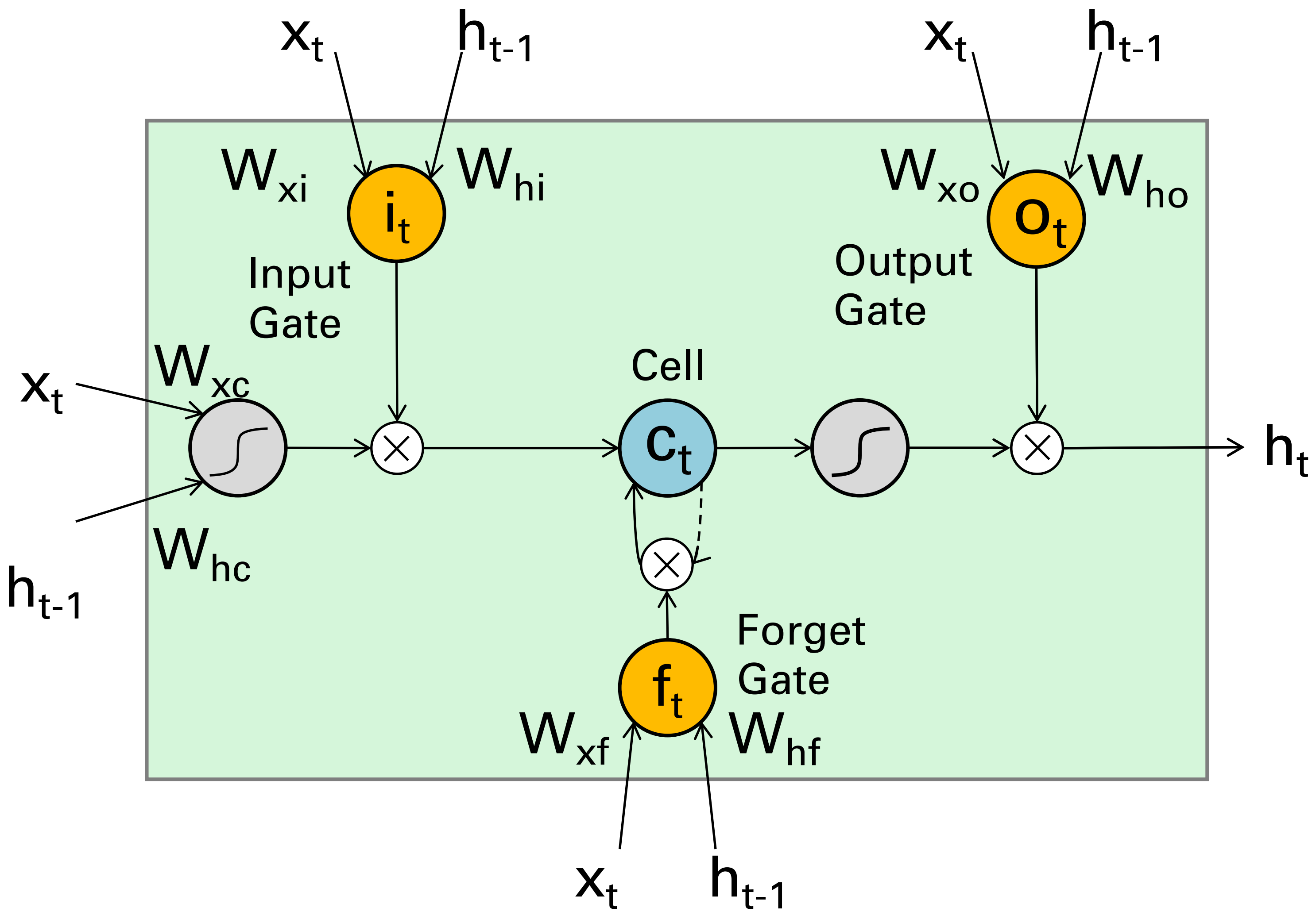
$$c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

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

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

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

The Popular LSTM Cell



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

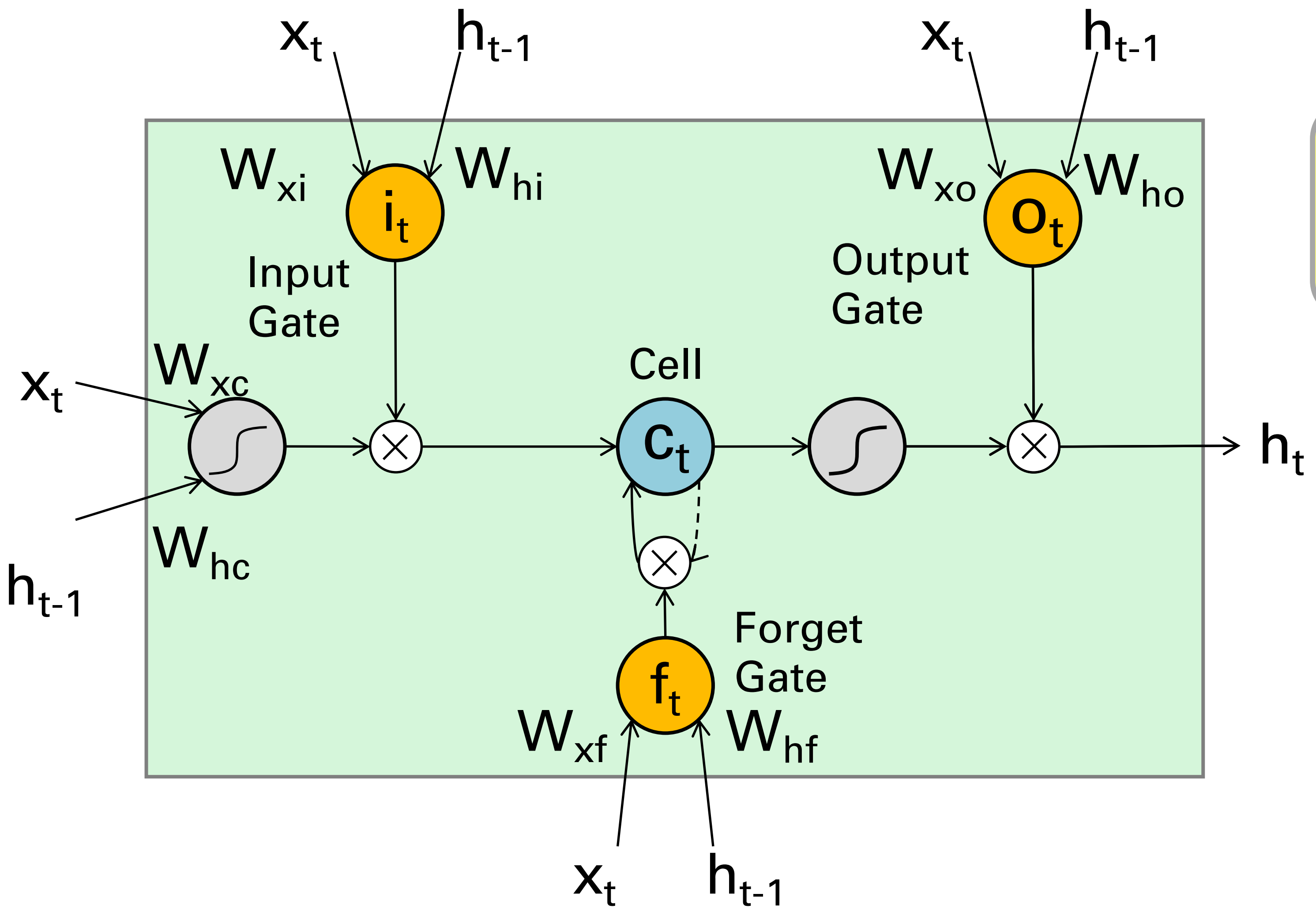
$$c_t = f_t \otimes c_{t-1} + \boxed{i_t \otimes \tanh W\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix}}$$

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

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

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

The Popular LSTM Cell



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

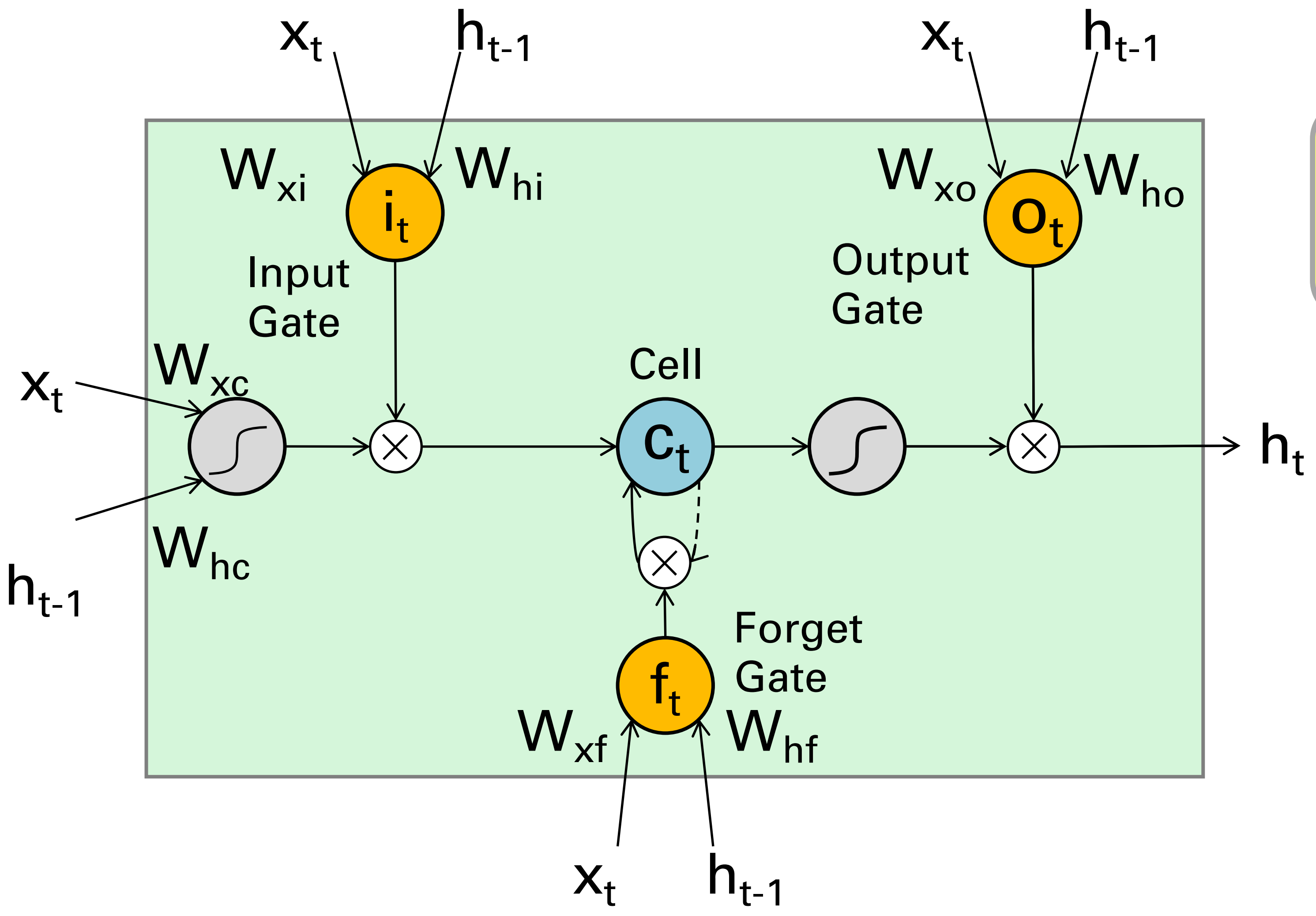
$$\boxed{c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh W\begin{pmatrix}x_t\\ h_{t-1}\end{pmatrix}}$$

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

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

Update the old cell state with the new one.

The Popular LSTM Cell

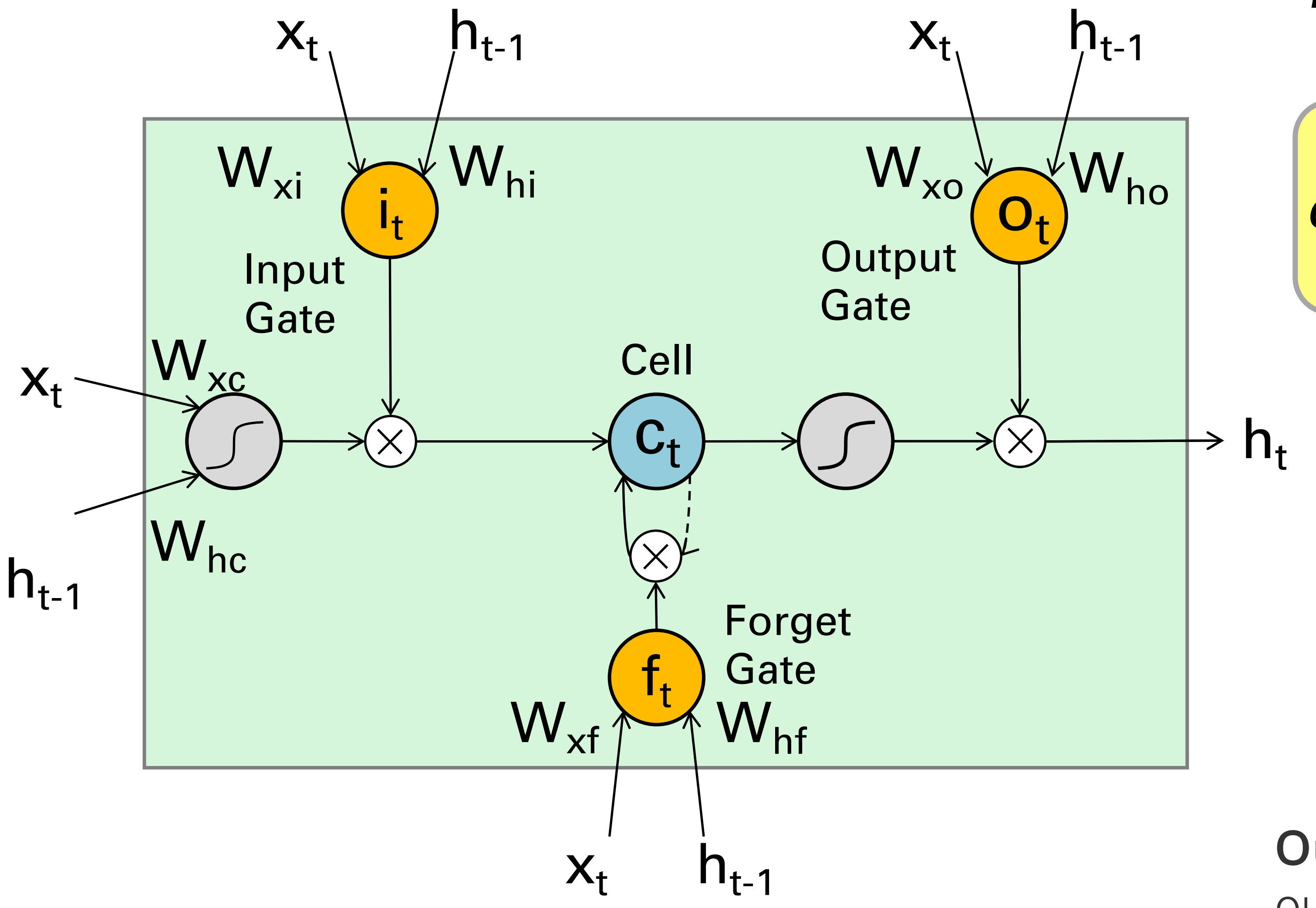


$$i_t = \sigma\left(W_i \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_i \right)$$

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

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.

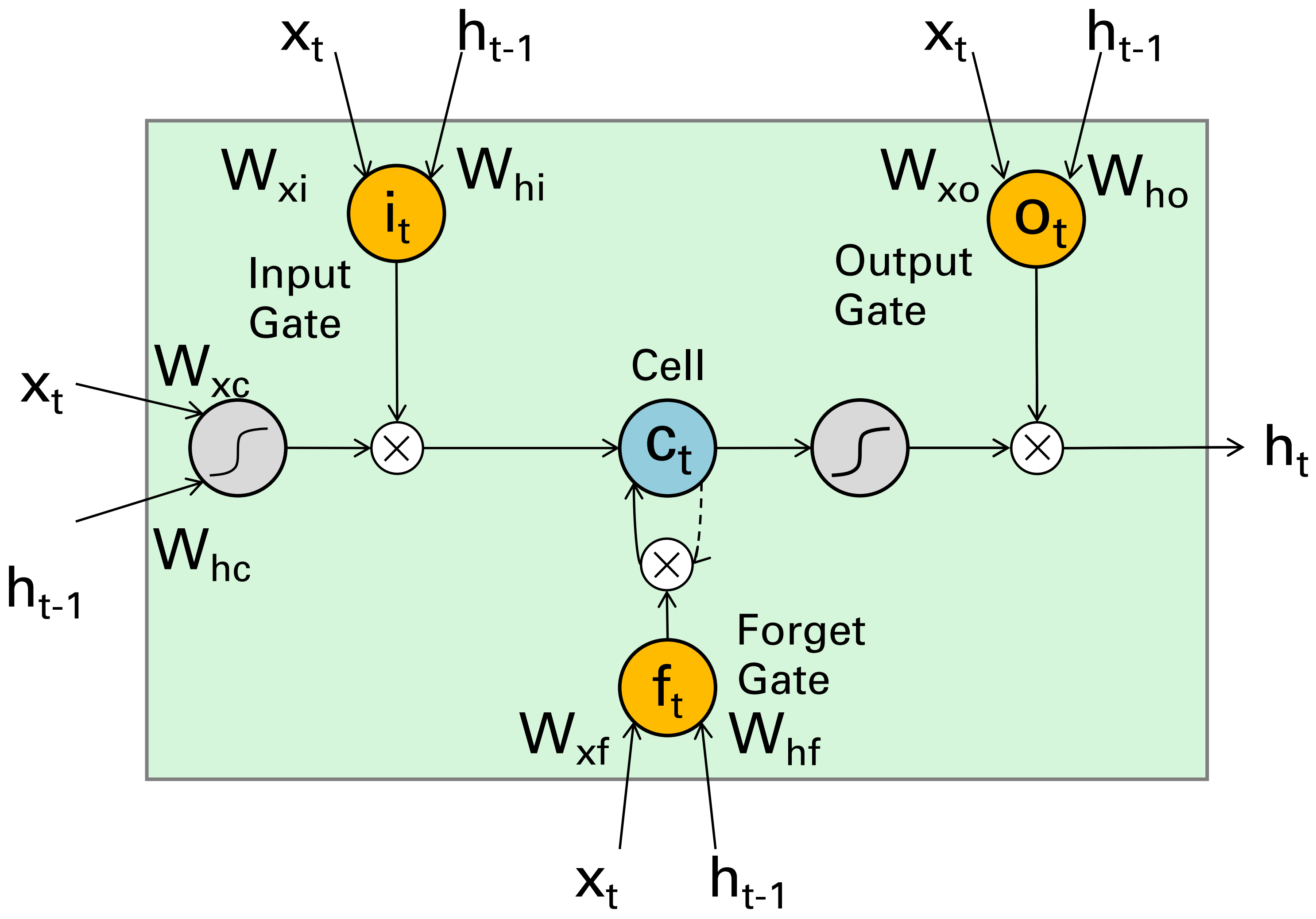
LSTM – Forward/Backward

[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



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

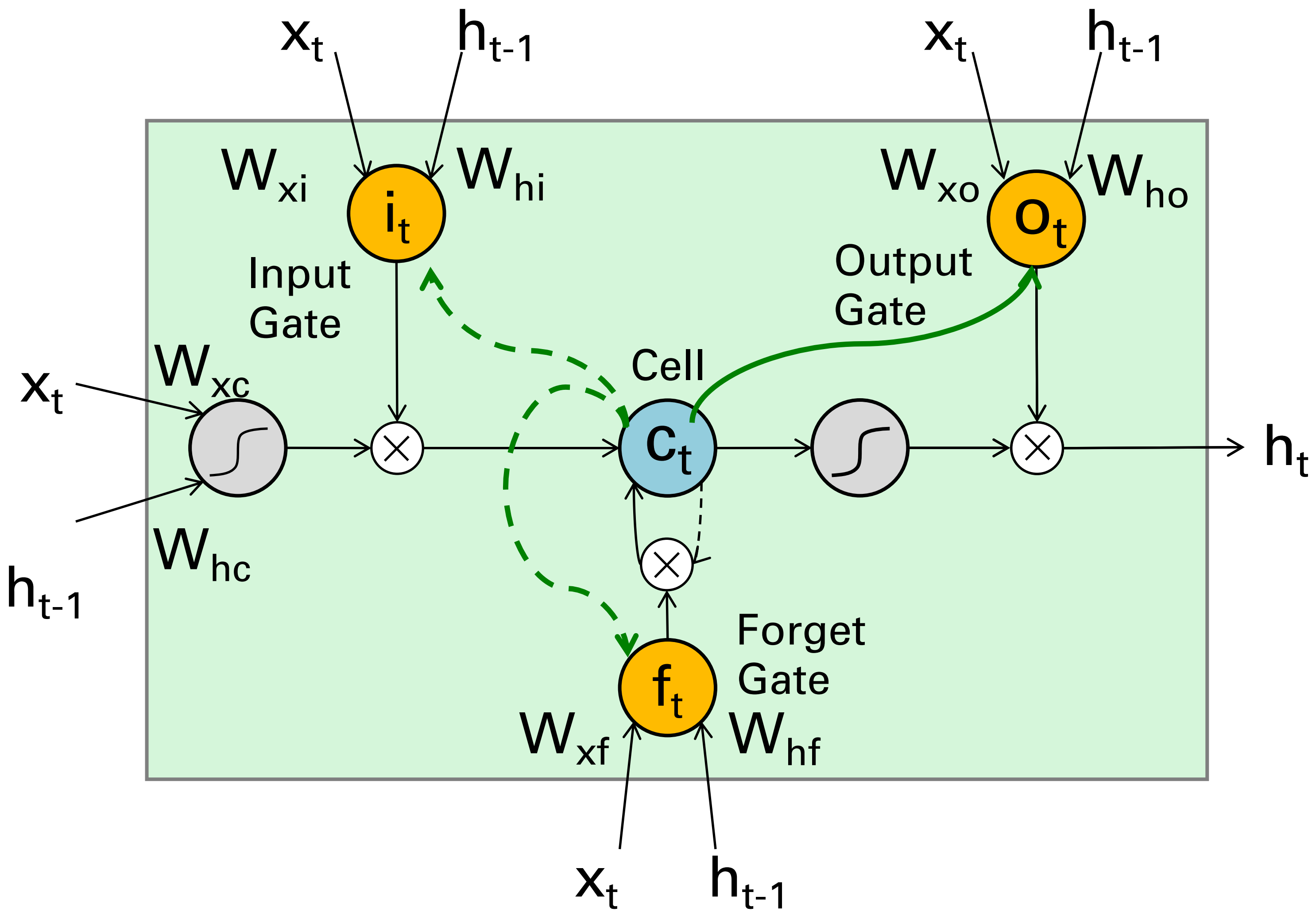
Similarly for i_t, o_t

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

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

* 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

Other minor variants

- Coupled Input and Forget Gate $f_t = 1 - i_t$

- Full Gate Recurrence

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

LSTM: A Search Space Odyssey

- Tested the following variants, using Peephole LSTM as standard:
  1. No Input Gate (NIG)
  2. No Forget Gate (NFG)
  3. No Output Gate (NOG)
  4. No Input Activation Function (NIAF)
  5. No Output Activation Function (NOAF)
  6. No Peepholes (NP)
  7. Coupled Input and Forget Gate (CIFG)
  8. Full Gate Recurrence (FGR)

- On the tasks of:
  – Timit Speech Recognition: Audio frame to 1 of 61 phonemes
  – IAM Online Handwriting Recognition: Sketch to characters
  – JSB Chorales: Next-step music frame prediction

[LSTM: A Search Space Odyssey [Greff et al., 2015]]

LSTM: A Search Space Odyssey

- The standard LSTM performed reasonably well on multiple datasets and none of the modifications significantly improved the performance
- Coupling gates and removing peephole connections simplified the LSTM without hurting performance much
- The forget gate and output activation are crucial

- Found interaction between learning rate and network size to be minimal – indicates calibration can be done using a small network first

LSTM: A Search Space Odyssey [Greff et al., 2015]

Gated Recurrent Unit

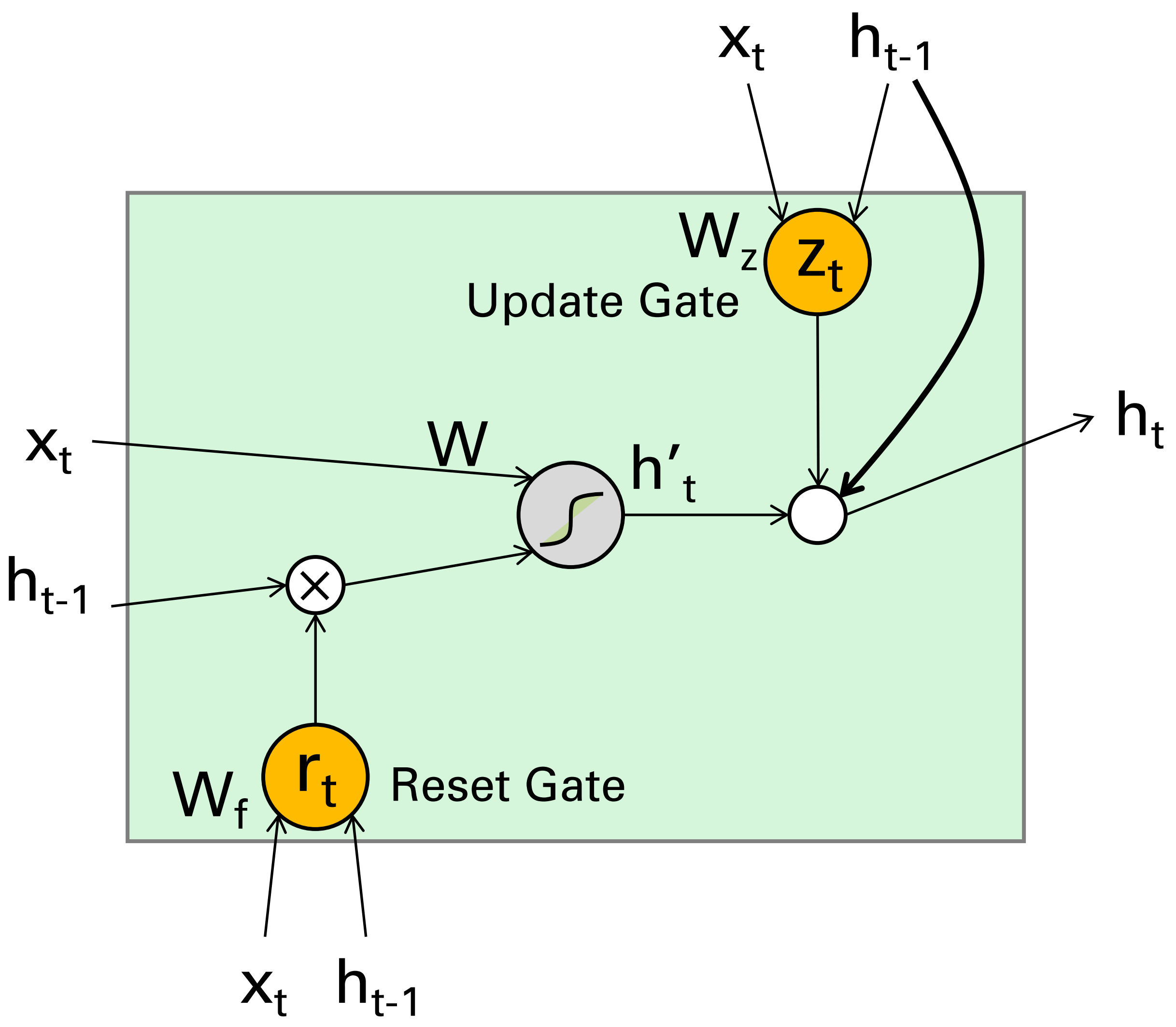
Gated Recurrent Unit (GRU)

- 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



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

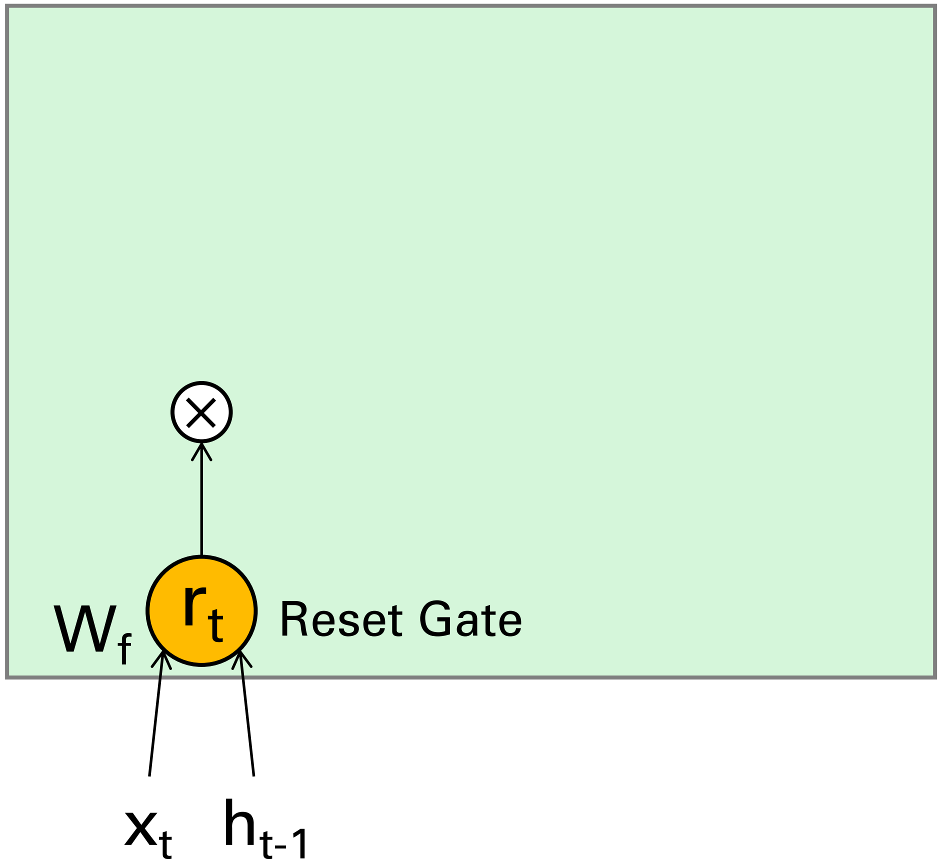
$$h'_t = \tanh W \begin{pmatrix} x_t \\ r_t \otimes h_{t-1} \end{pmatrix}$$

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

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

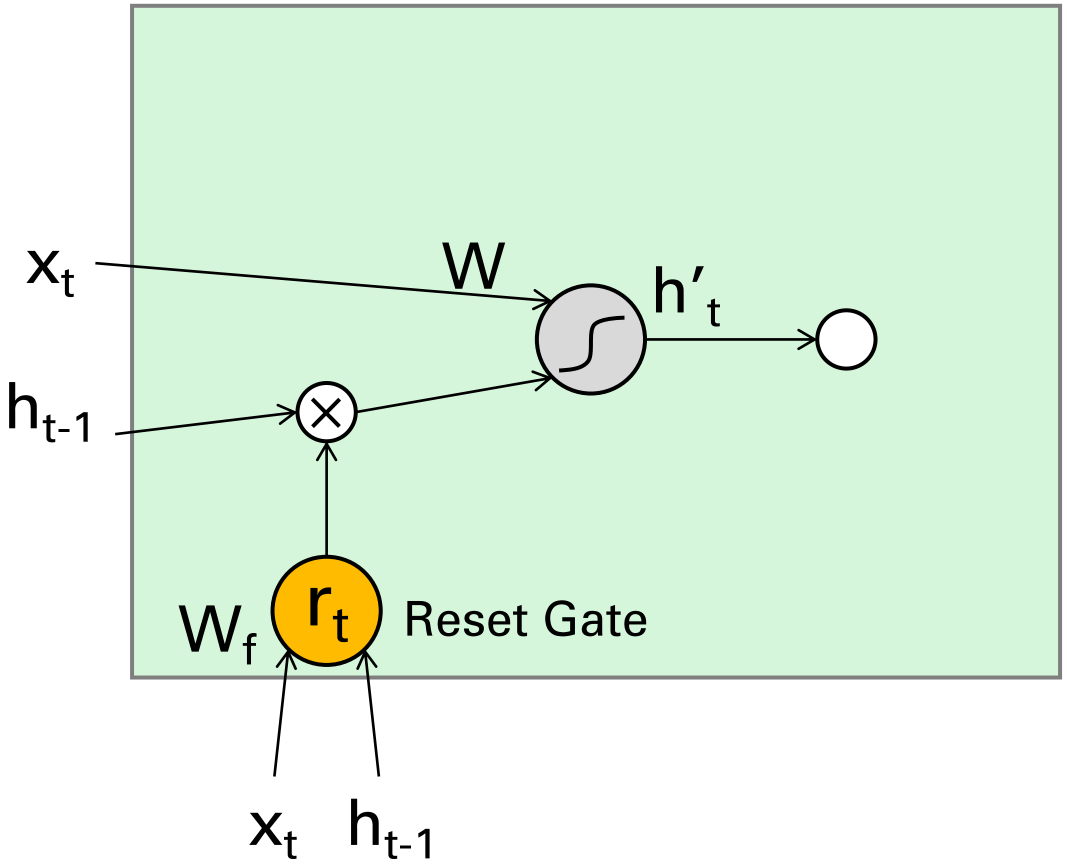
GRU

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



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

GRU



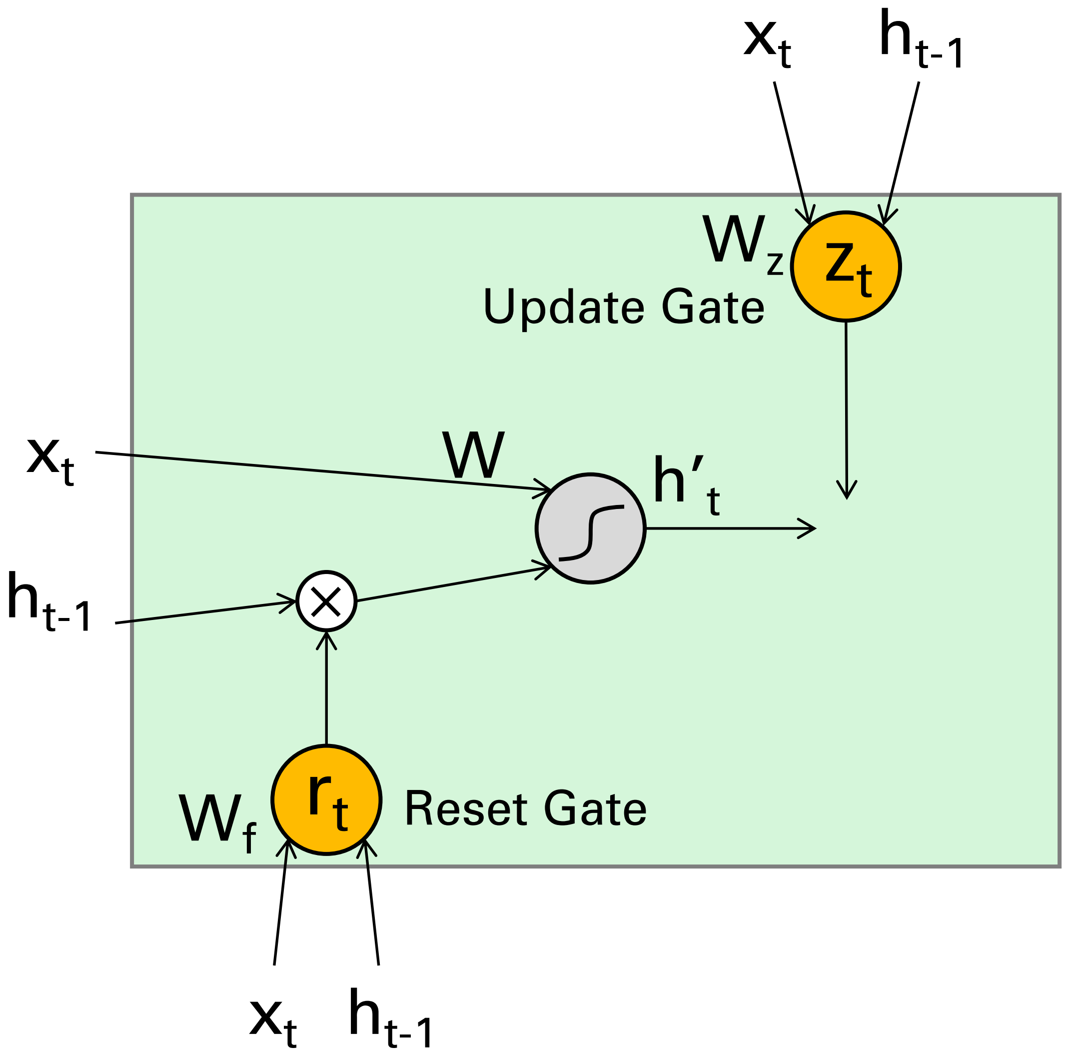
$$r_t = \sigma\left(W_r\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_f\right)$$

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

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



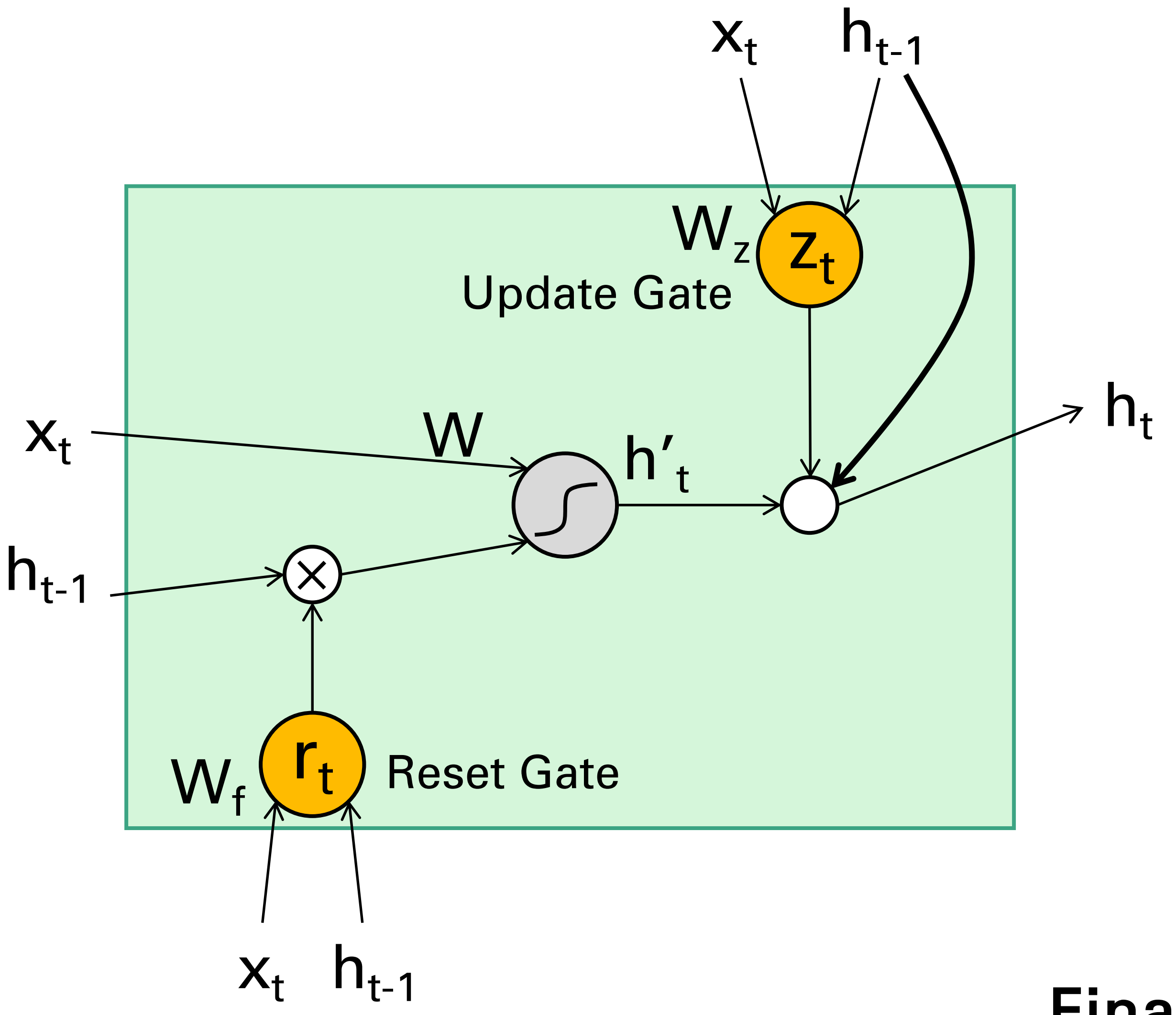
$$r_t = \sigma\left(W_r \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f \right)$$

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

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

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

GRU



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

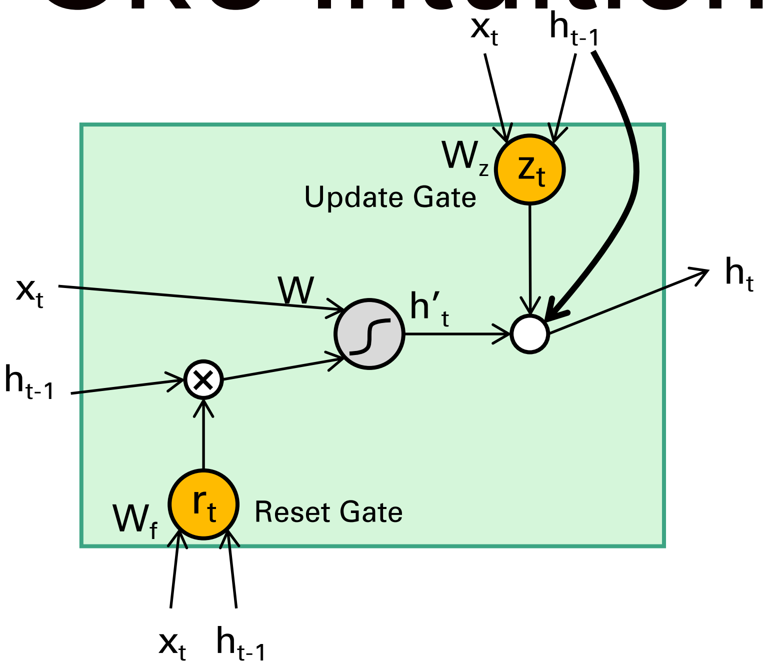
$$h'_t = \tanh W\begin{pmatrix} x_t \\ r_t \otimes h_{t-1}\end{pmatrix}$$

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

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

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

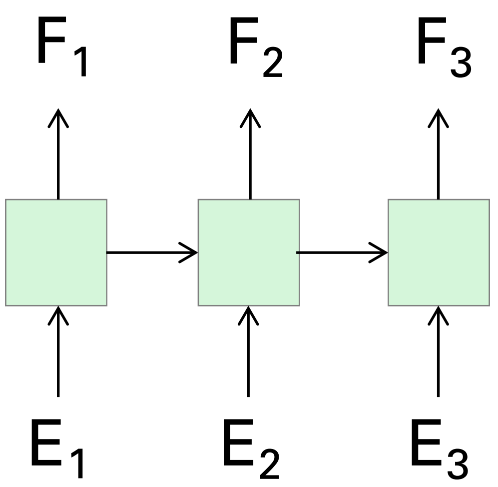
- Careful initialization and optimization of vanilla RNNs can enable them to learn long(ish) dependencies, but gated additive cells, like the LSTM and GRU, often just work.

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*

Is RNNs enough?

- Consider the problem of translation of English to French
- E.g. What is your name $\longrightarrow$ Comment tu t'appelle
- Is the below architecture suitable for this problem?

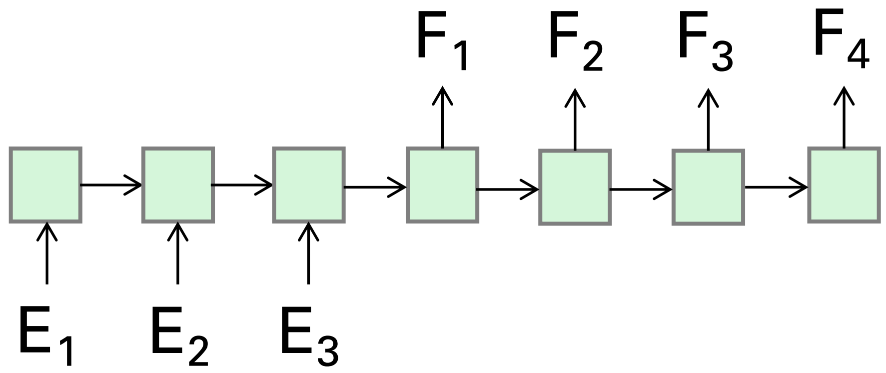


- No, sentences might be of different length and words might not align. Need to see entire sentence before translating

Adapted from http://www.cs.toronto.edu/~rgrosse/csc321/lec10.pdf

Encoder-decoder seq2seq model

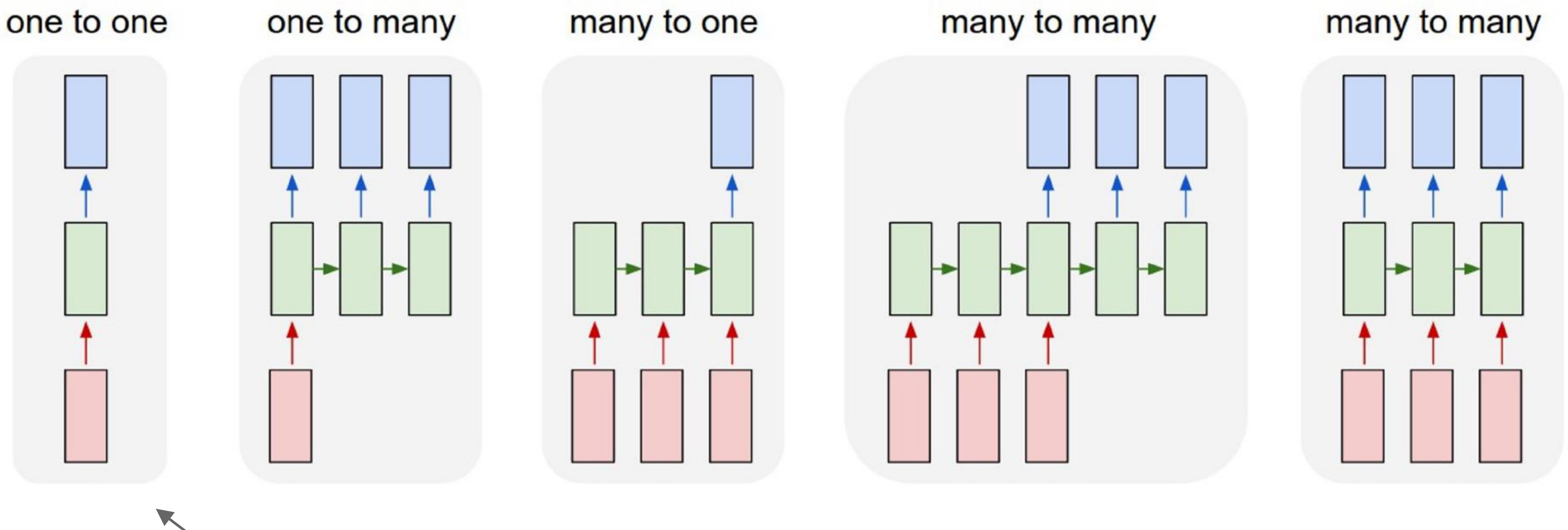
- Consider the problem of translation of English to French
- E.g. What is your name $\longrightarrow$ Comment tu t'appelle
- Sentences might be of different length and words might not align. Need to see entire sentence before translating



- Input-Output nature depends on the structure of the problem at hand

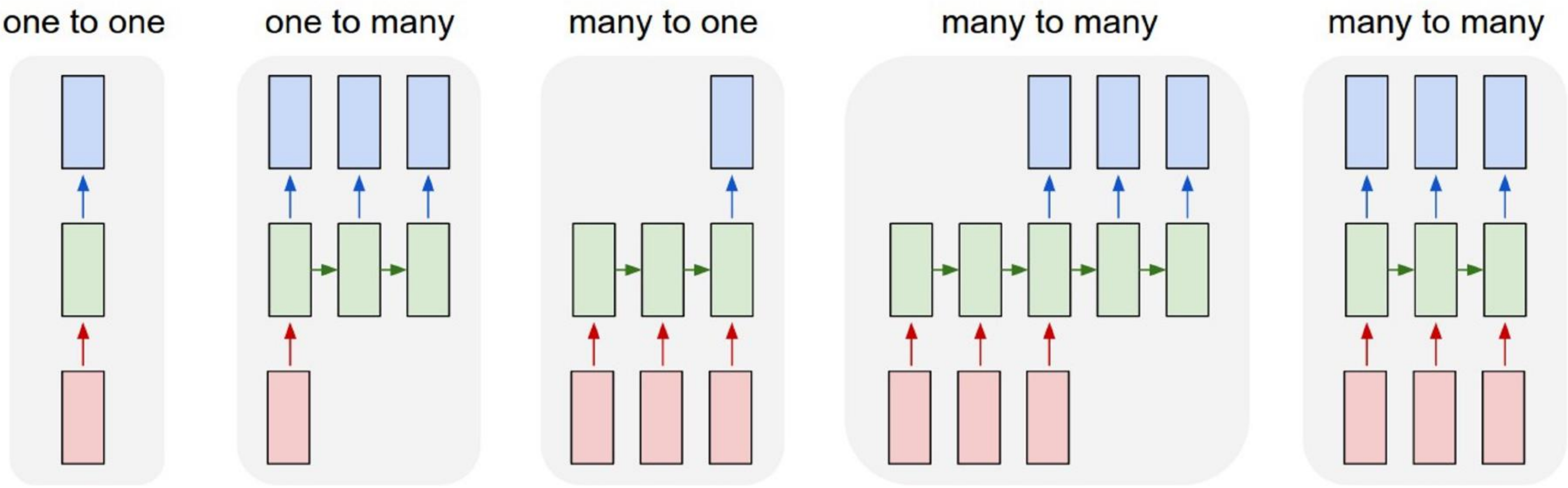
[Seq2Seq Learning with Neural Networks. Sutskever et al., NIPS 2014](#)

Recurrent Networks offer a lot of flexibility:



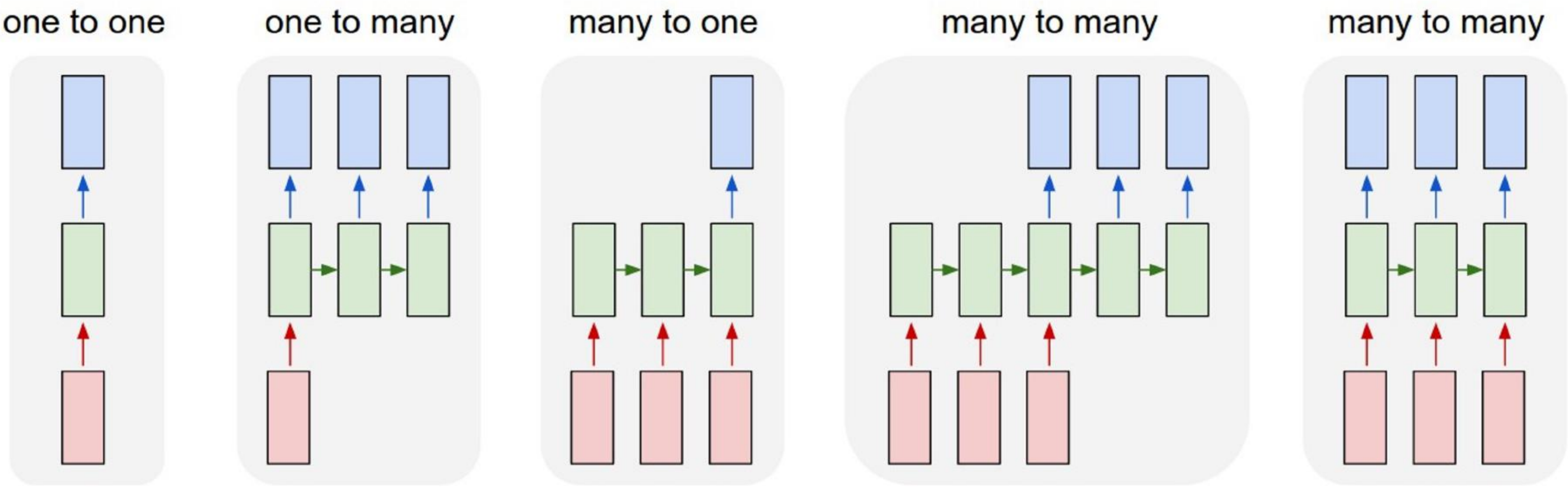
Vanilla Neural Networks

Recurrent Networks offer a lot of flexibility:



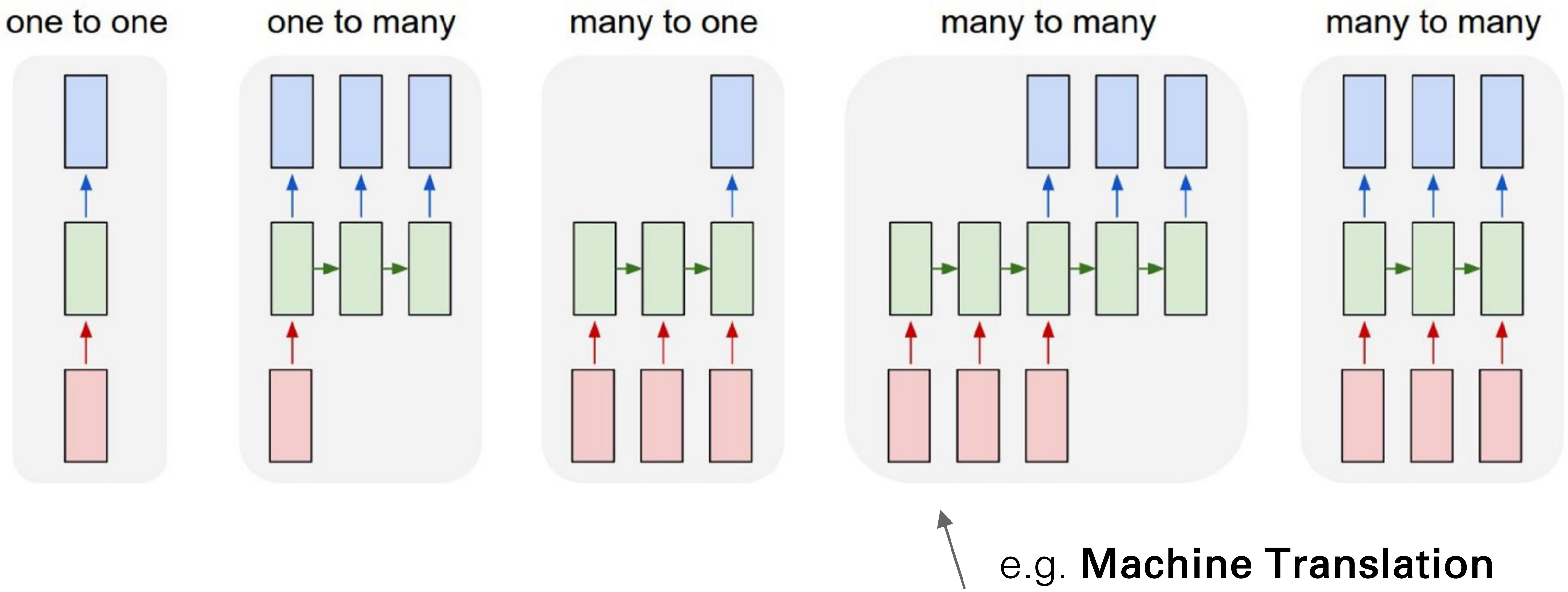
e.g. **Image Captioning**
image → sequence of words

Recurrent Networks offer a lot of flexibility:



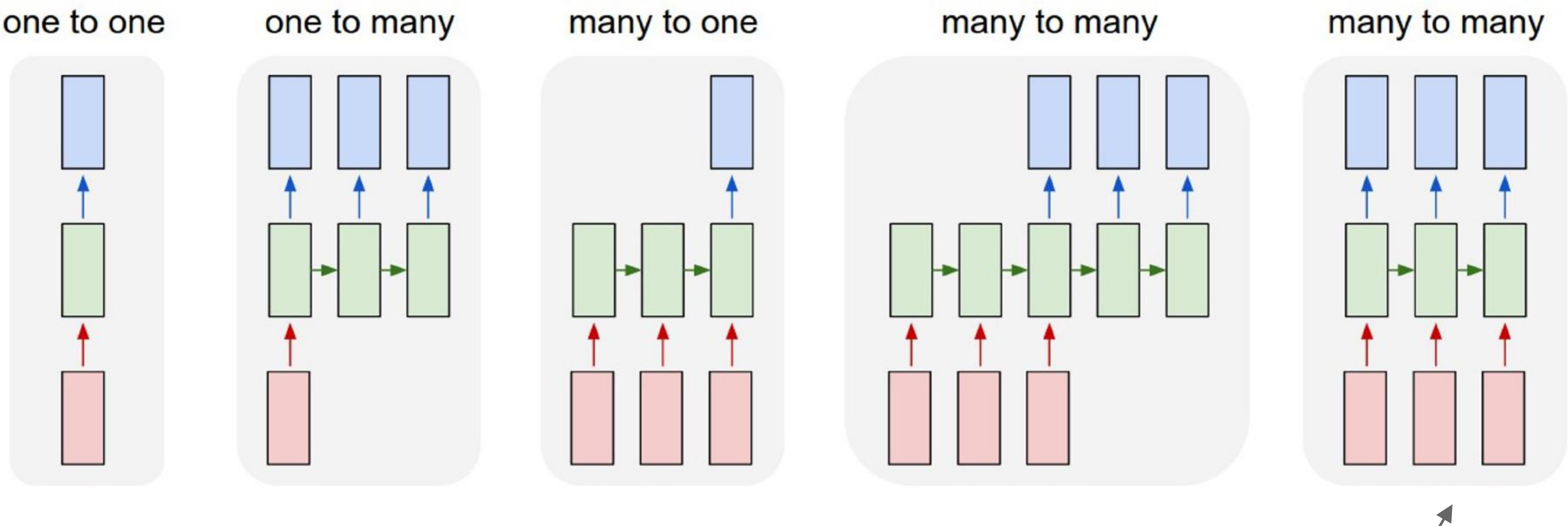
e.g. **Sentiment Classification**
sequence of words → sentiment

Recurrent Networks offer a lot of flexibility:



e.g. **Machine Translation**
seq of words → seq of words

Recurrent Networks offer a lot of flexibility:



e.g. **Video classification on frame level**

Next Lecture:
Attention and Transformers