Lecture 10: Memory and sequence modeling

Speaker: Sara Beery

11. Memory and sequence modeling

- CNNs for sequences
- RNNs
- LSTMs
- Sequence models and long memory


kindergarden classroom


television
person
chair

“What color is the chair?”


"What color is the chair?"
red

“What will the girl do next?”

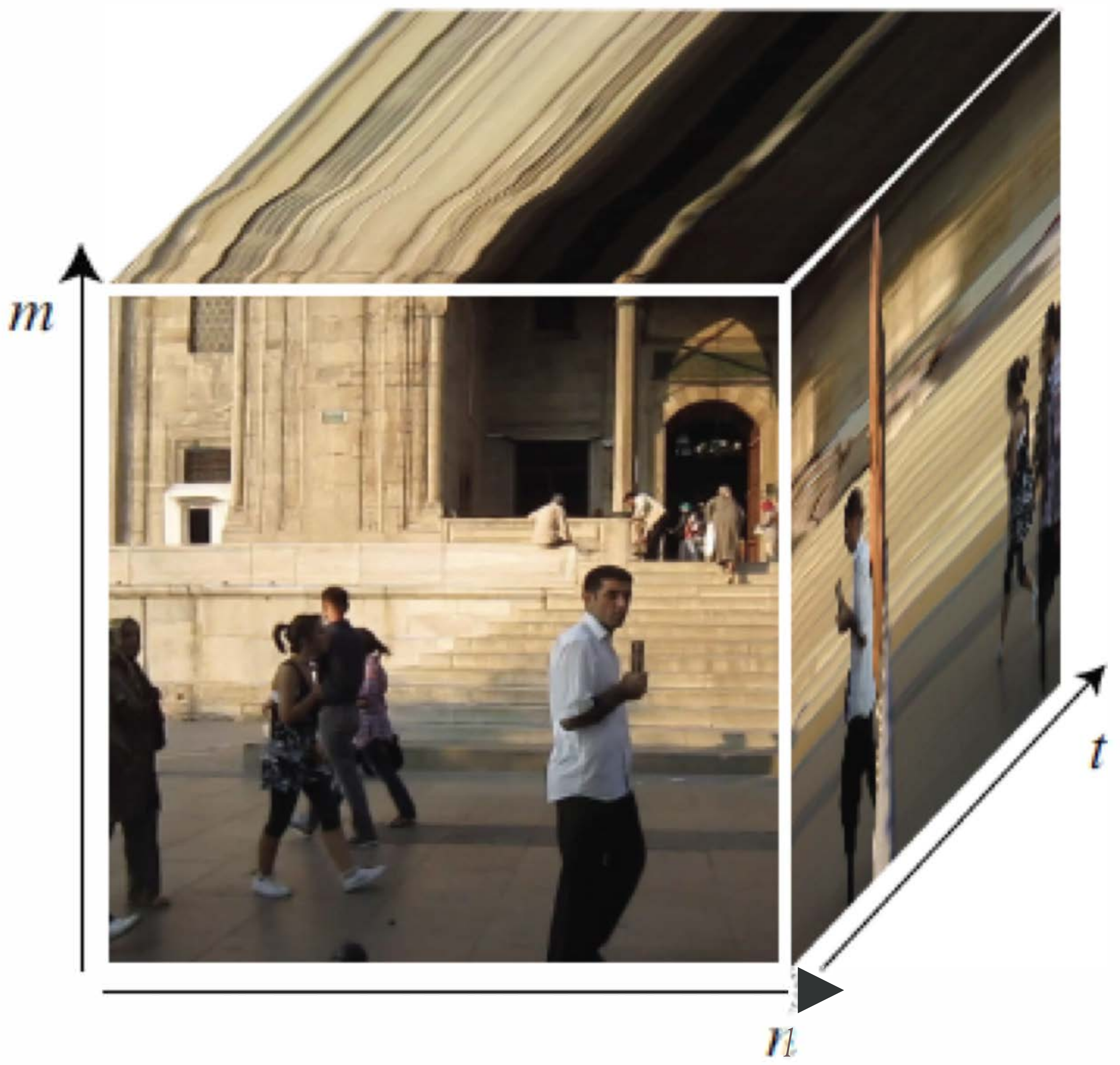
Sequences

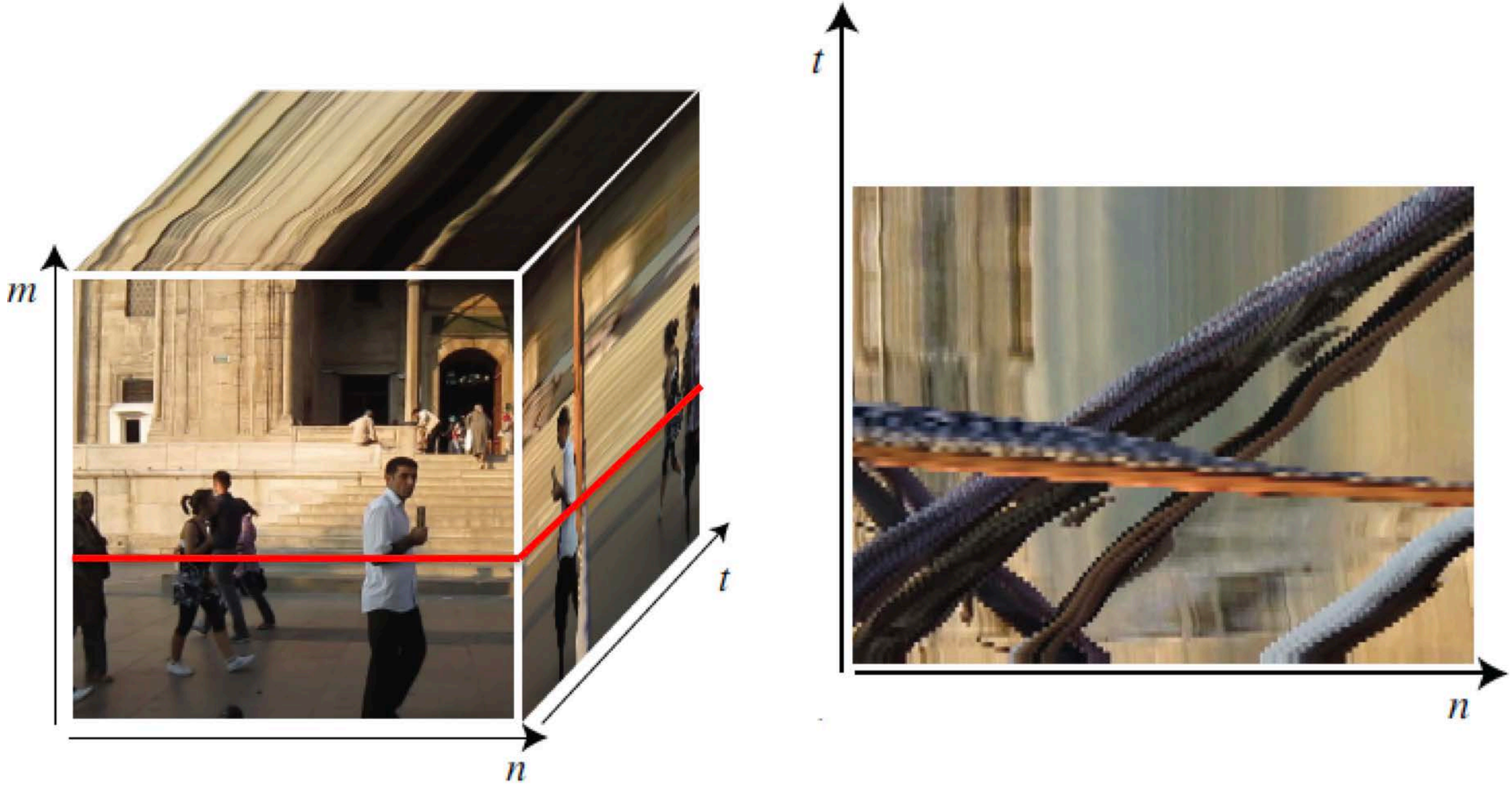
time

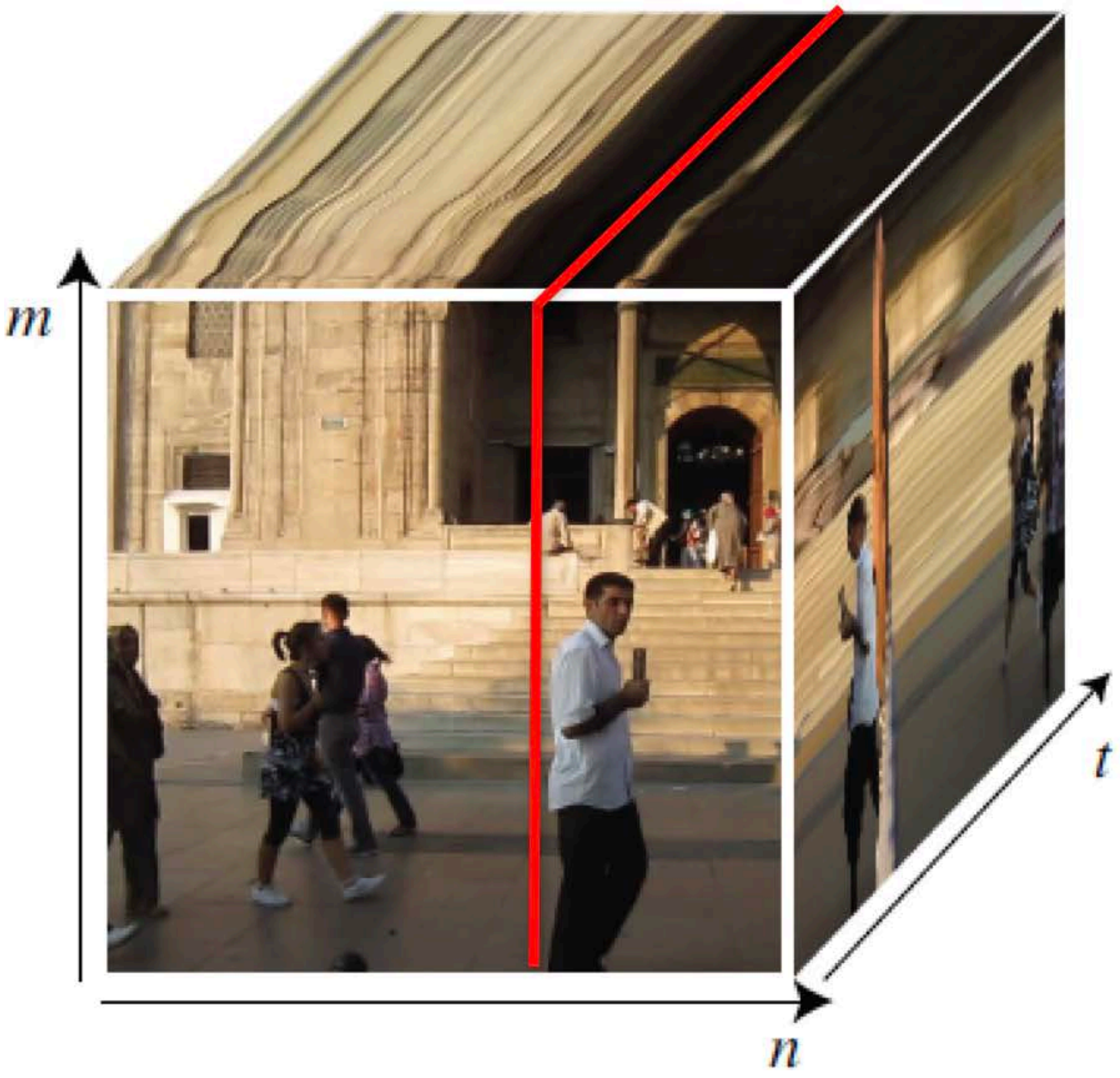
“An”, “evening”, “stroll”, “through”, “a”, “city”, “square”

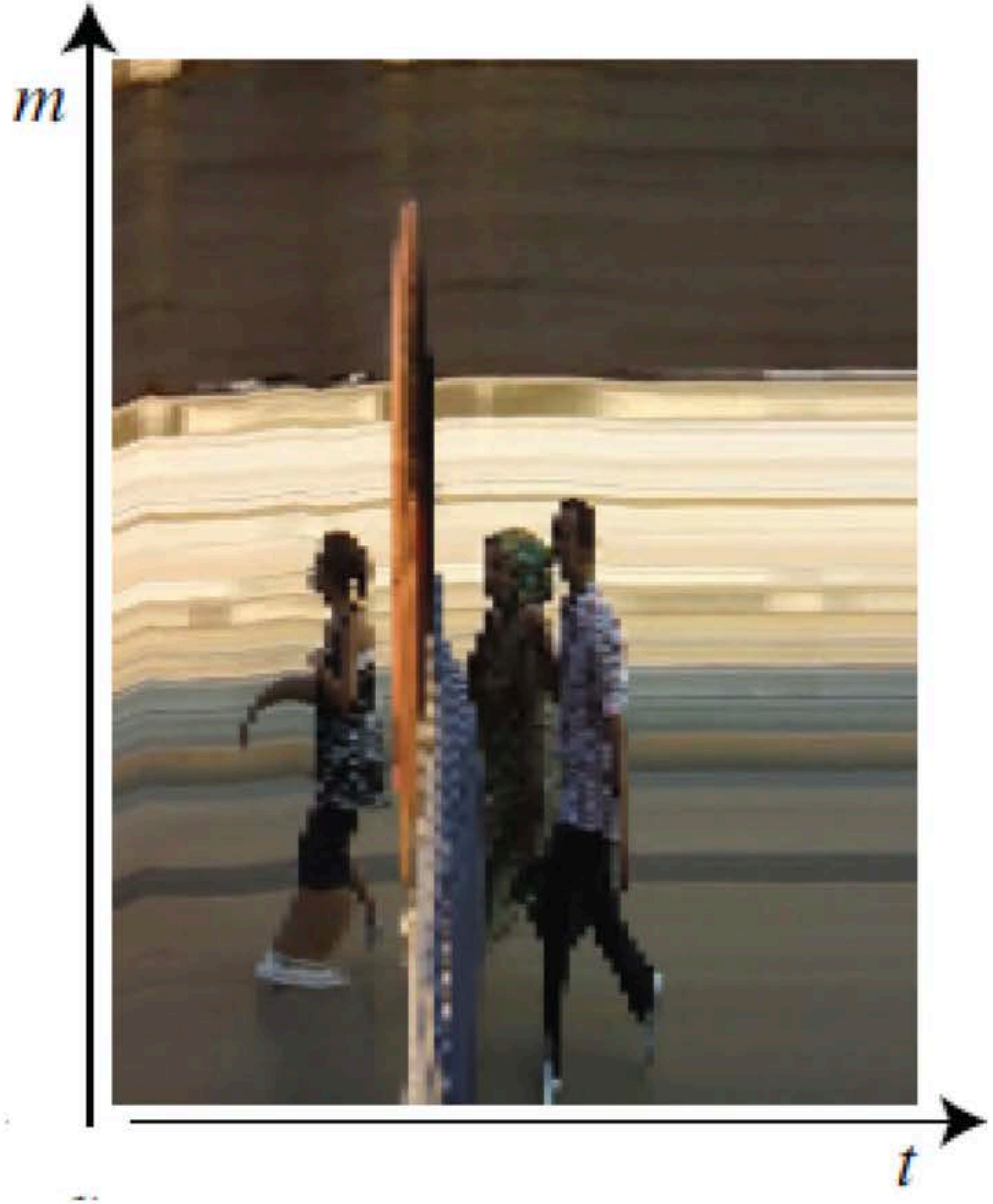
time

time

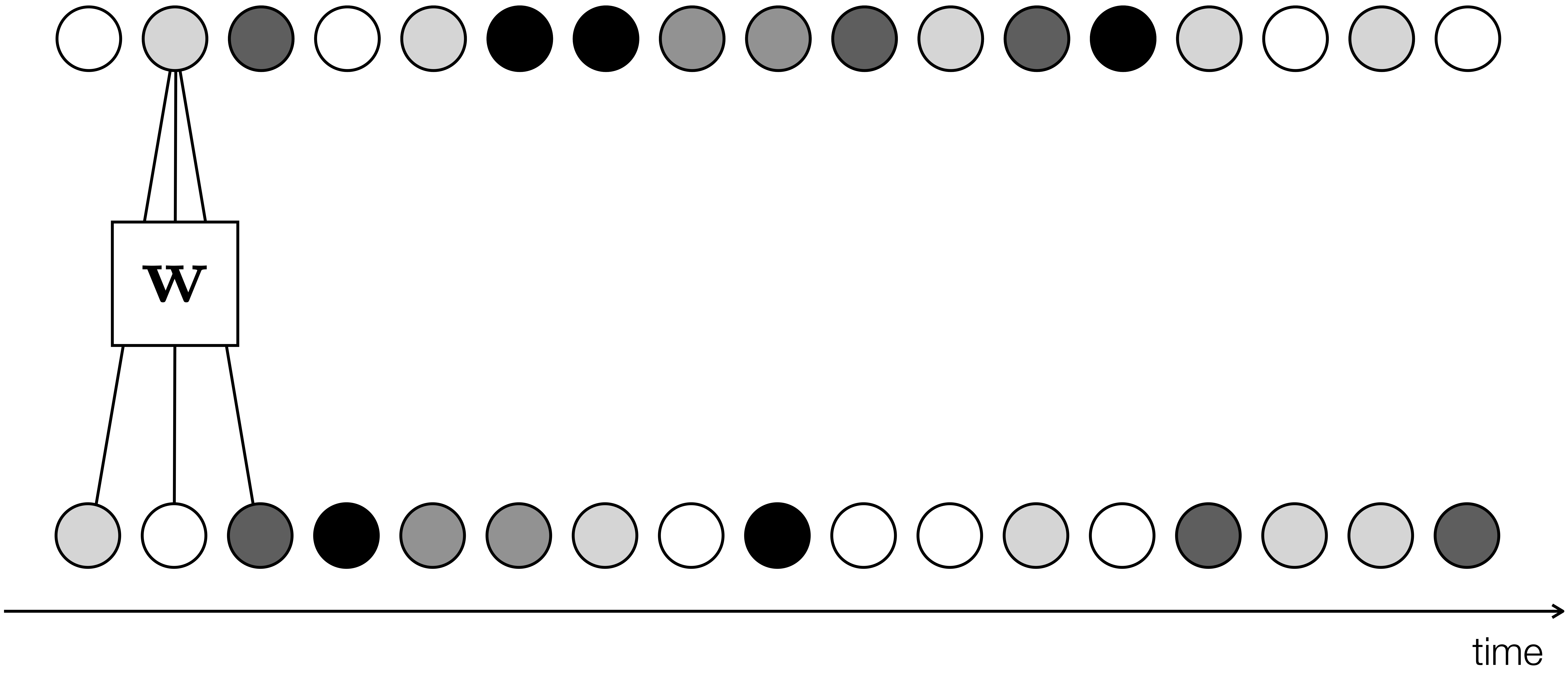

m
t
n

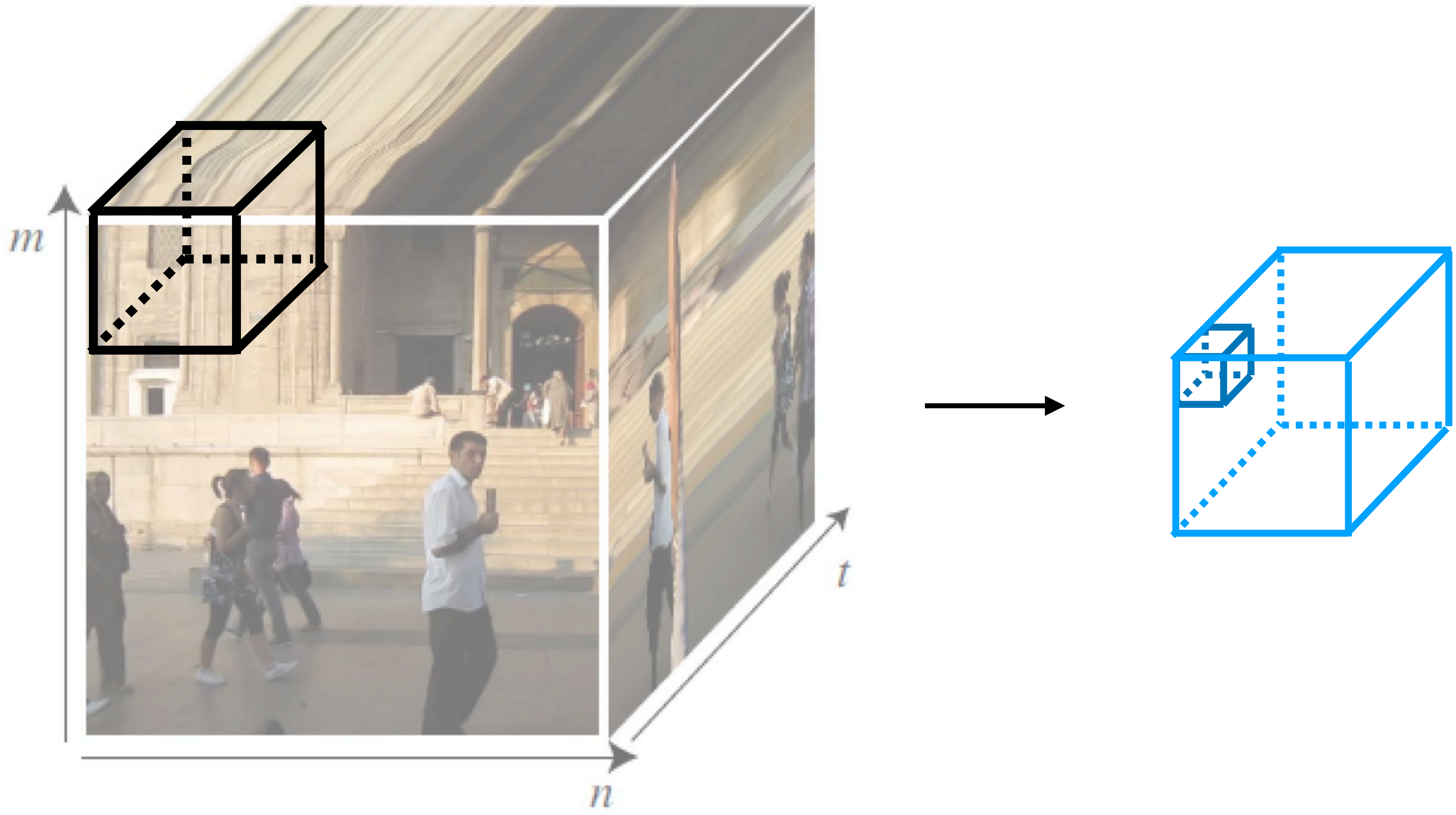

m
n
t
t
n


m
n
t

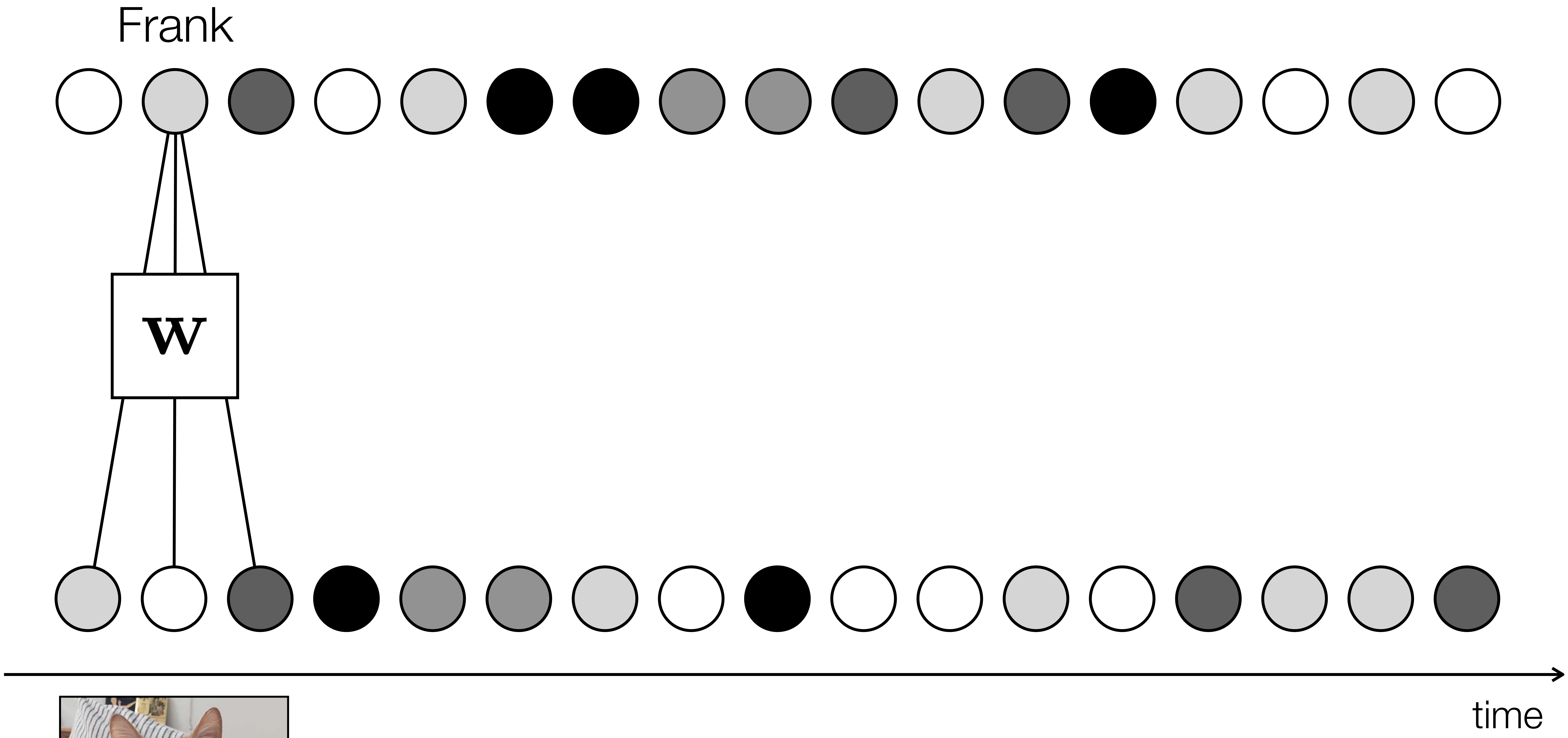

m
t

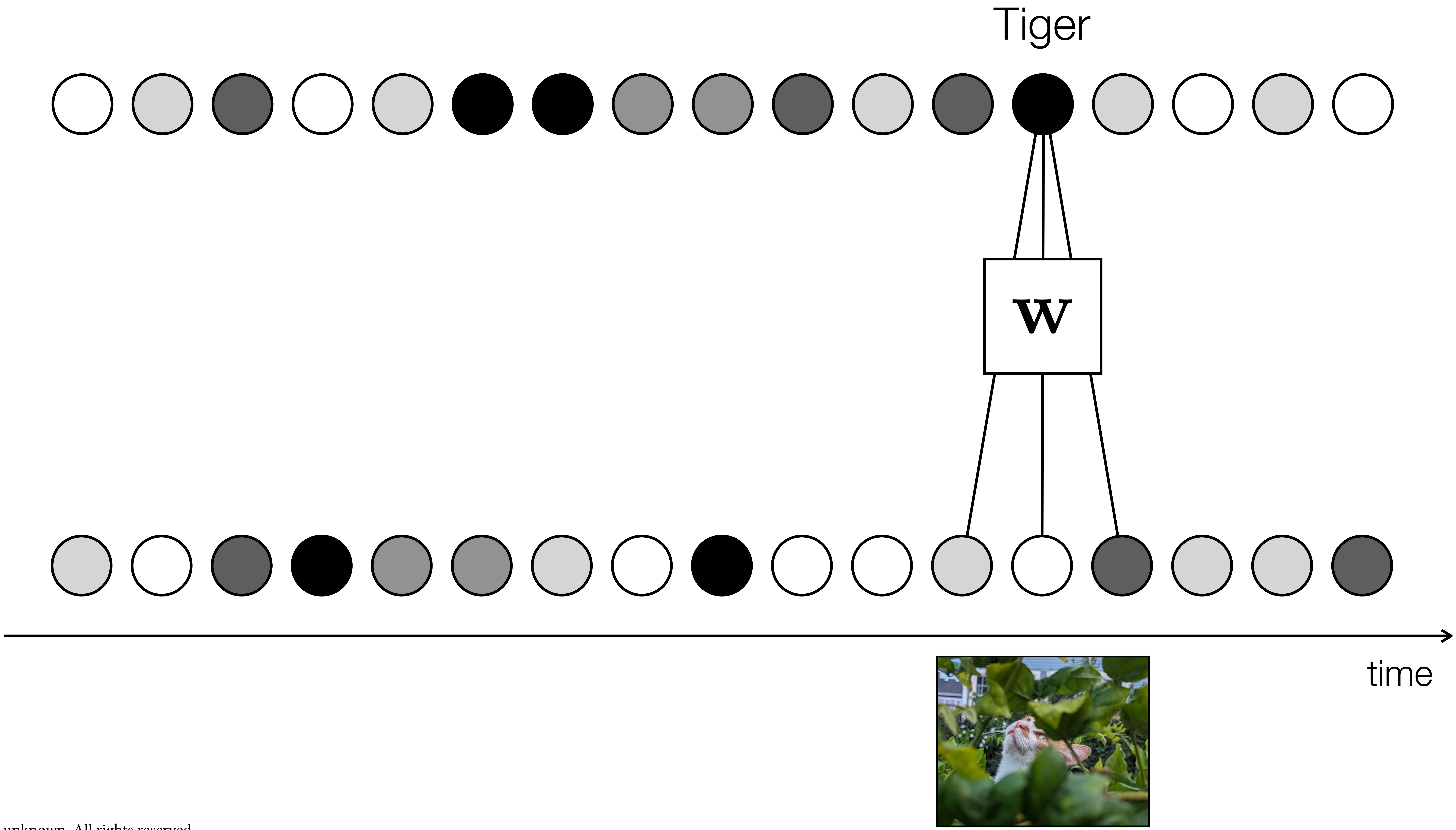
Convolutions in time

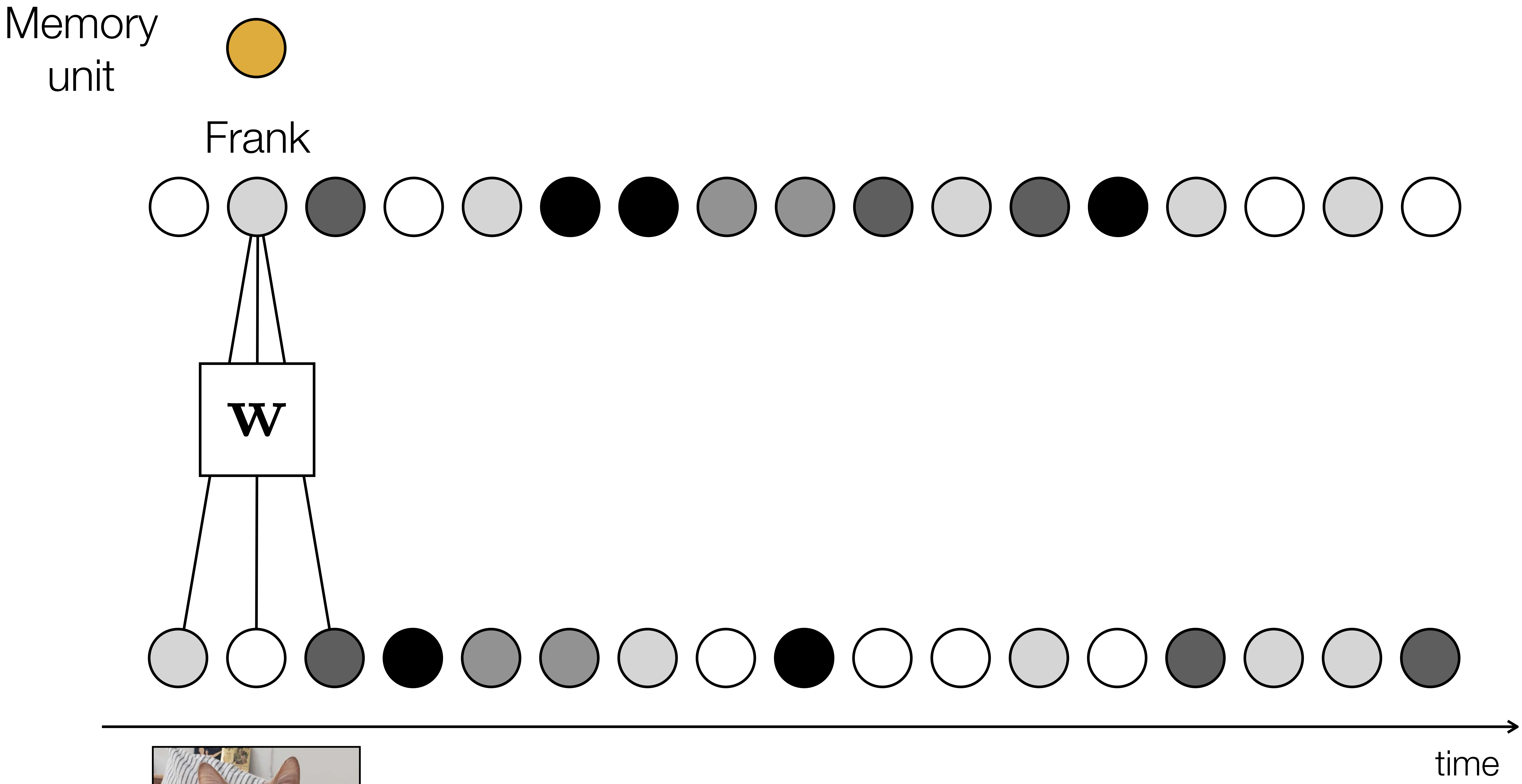


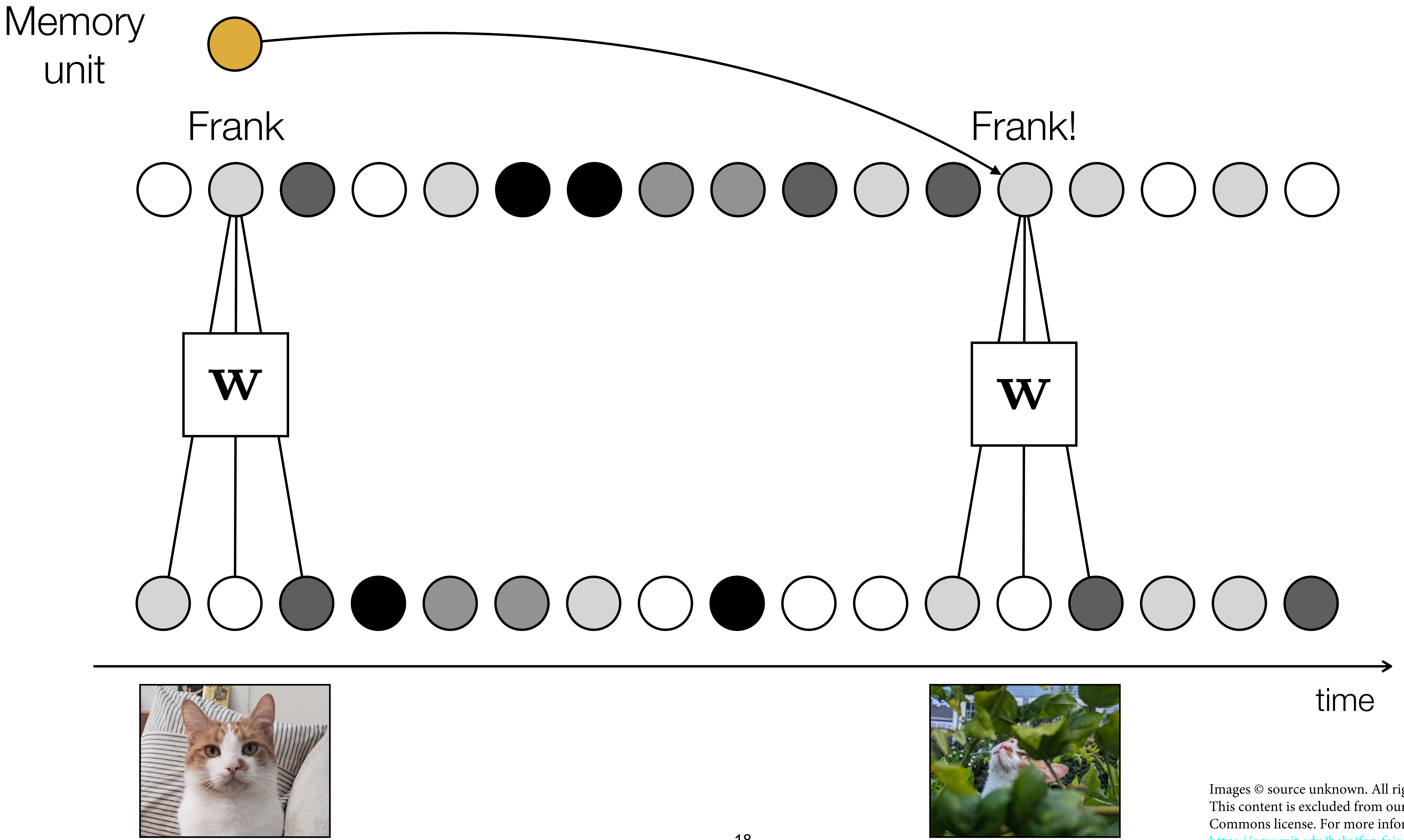


m
t
n

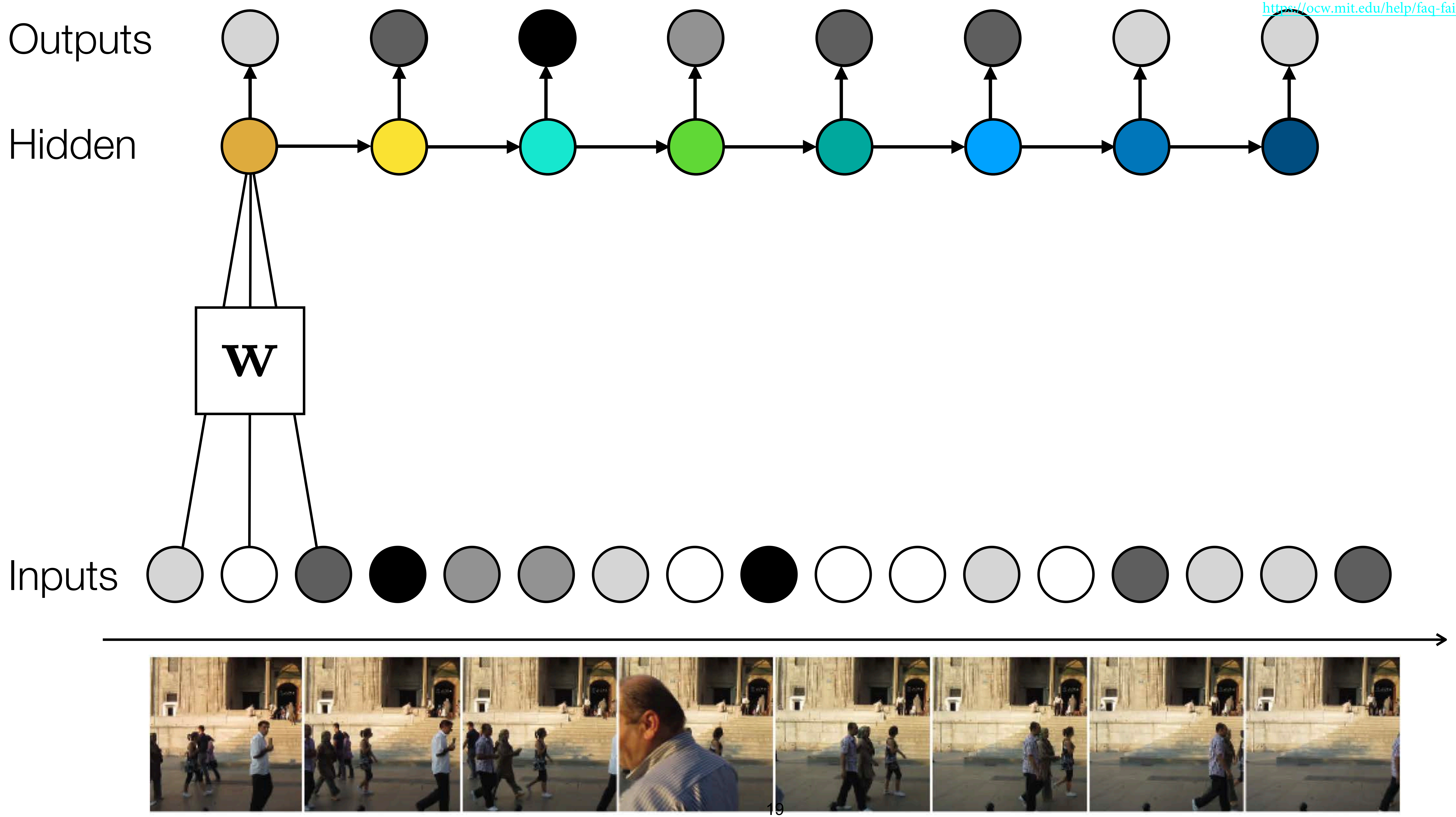

Frank
W
time


Tiger
w
time

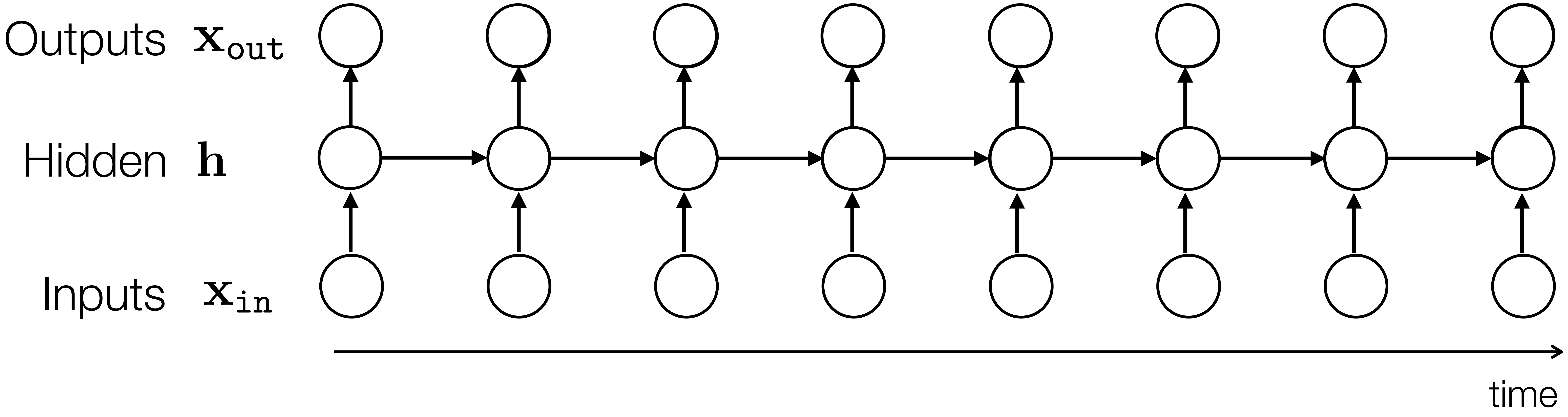

Memory unit
Frank
W
time


Memory unit
Frank
Frank!
w
w
time

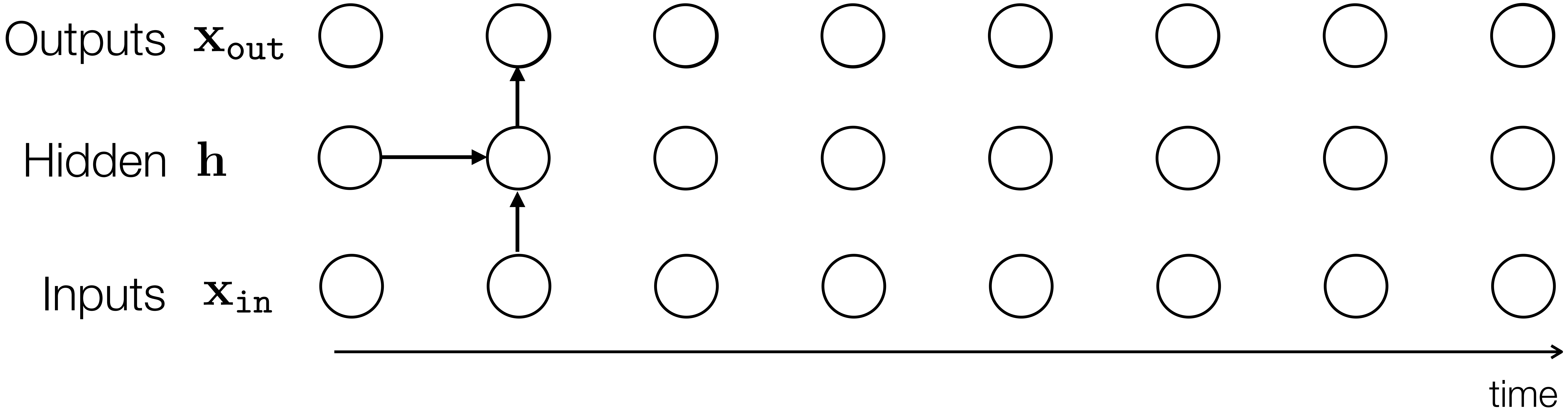
Recurrent Neural Networks (RNNs)



Recurrent Neural Networks (RNNs)



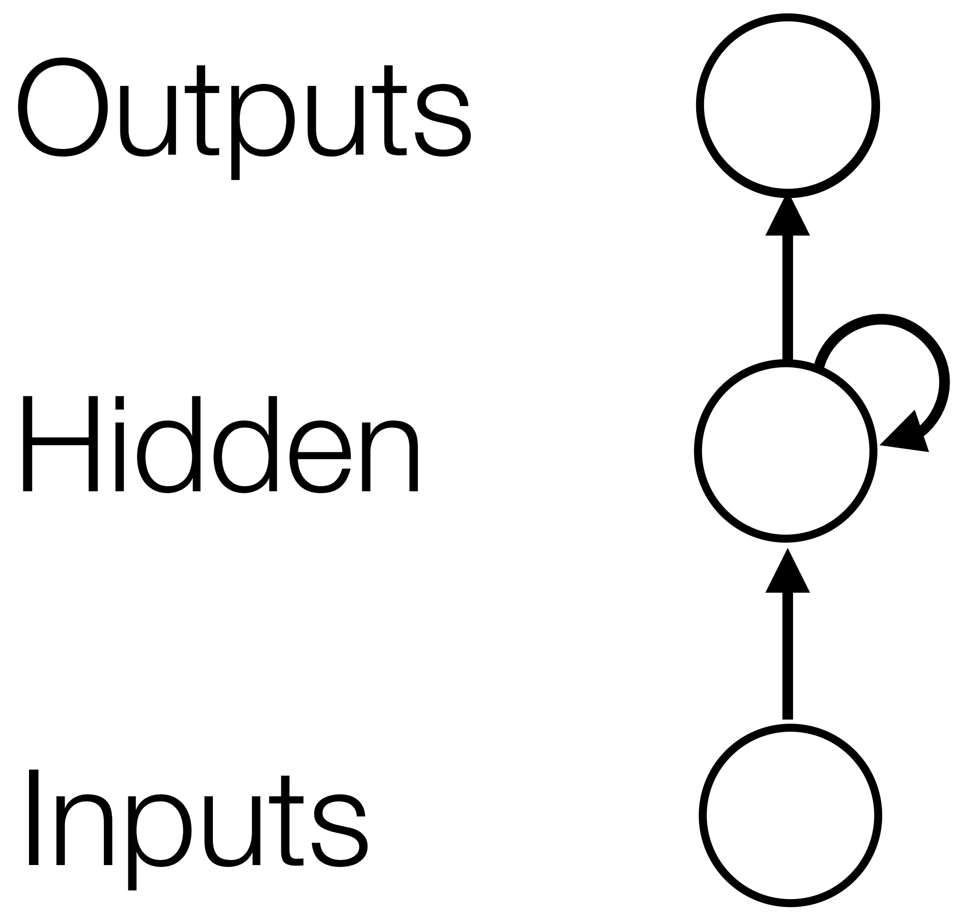
Recurrent Neural Networks (RNNs)



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

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

Recurrent Neural Networks (RNNs)

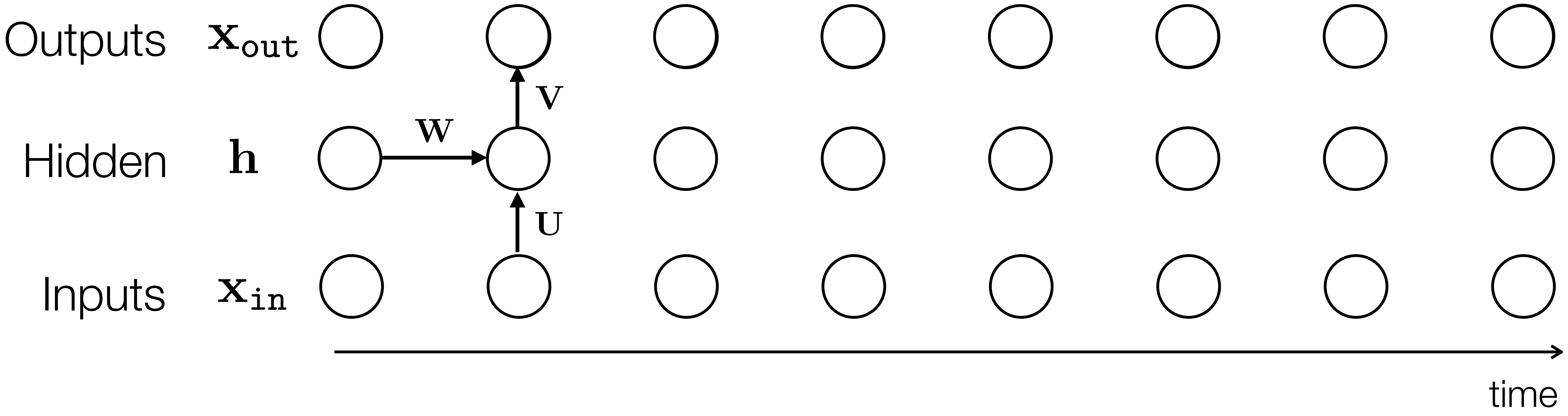


Recurrent!

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

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

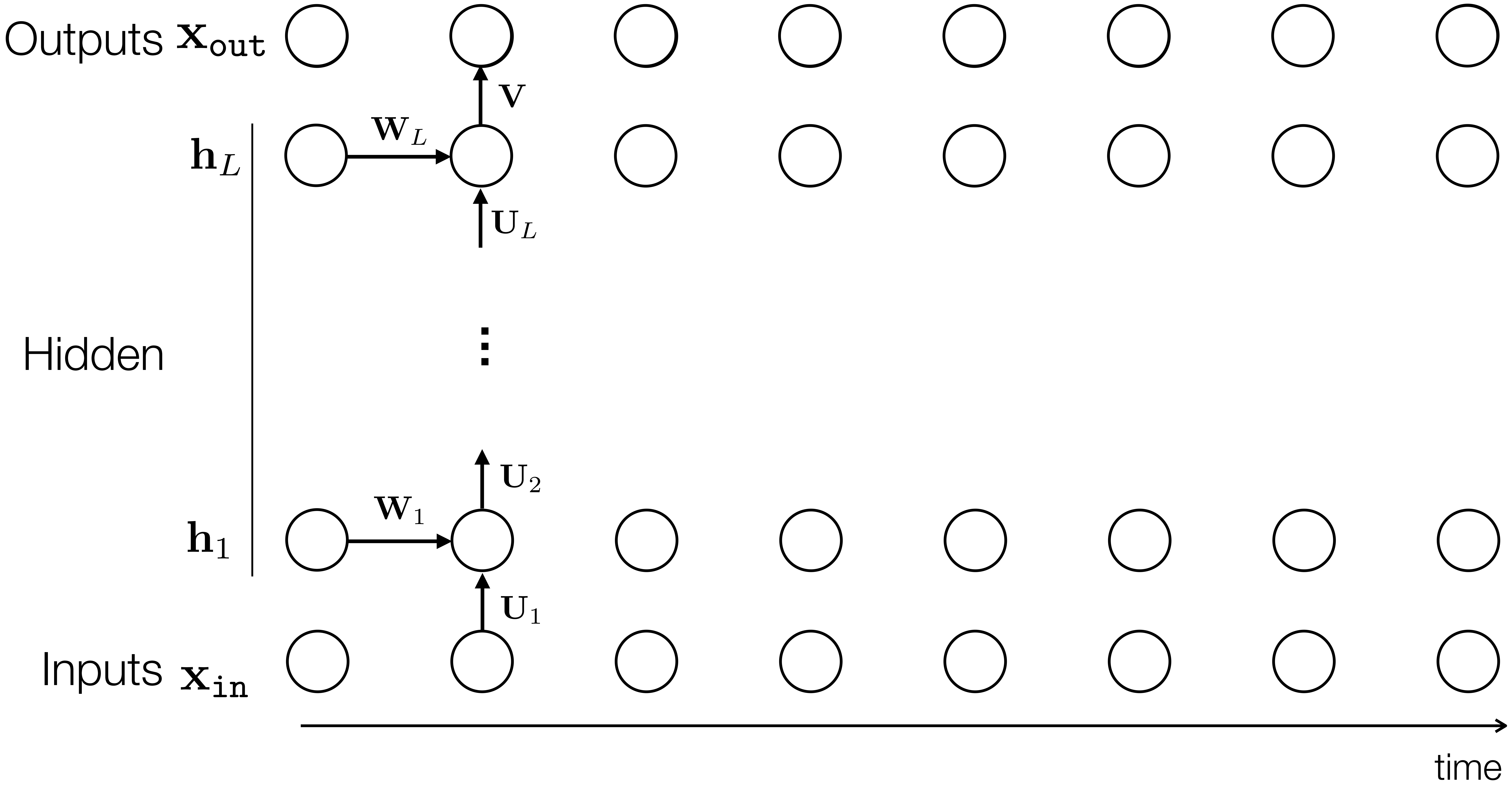
Recurrent Neural Networks (RNNs)



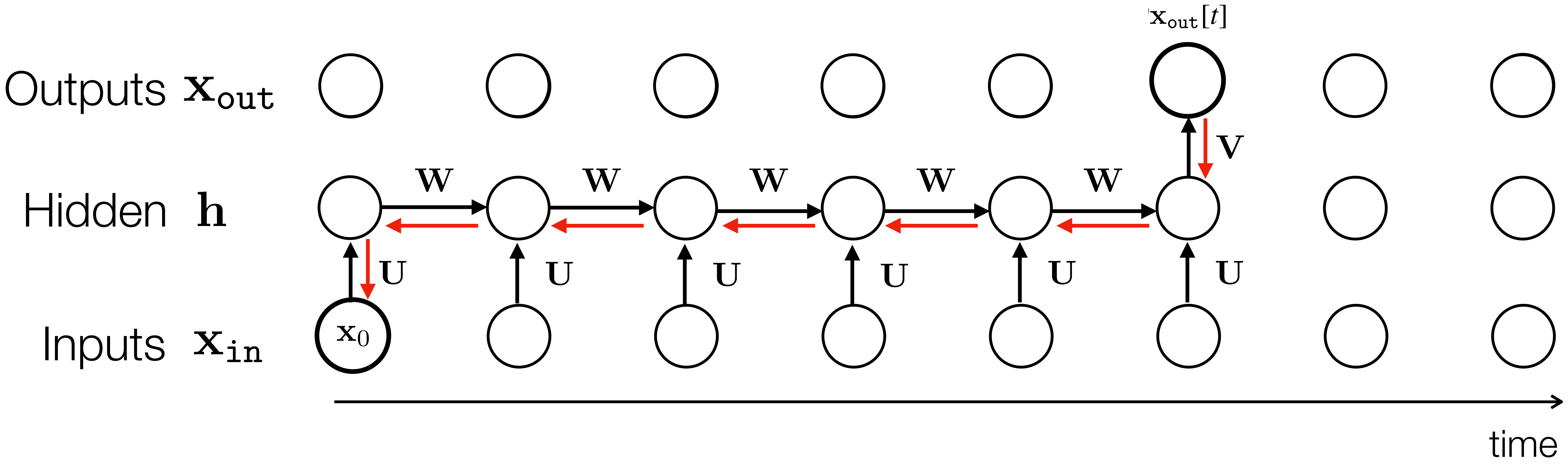
$$\mathbf{h}_t = \sigma_1(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_{\mathtt{in}}[t] + \mathbf{b})$$

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

Deep Recurrent Neural Networks (RNNs)

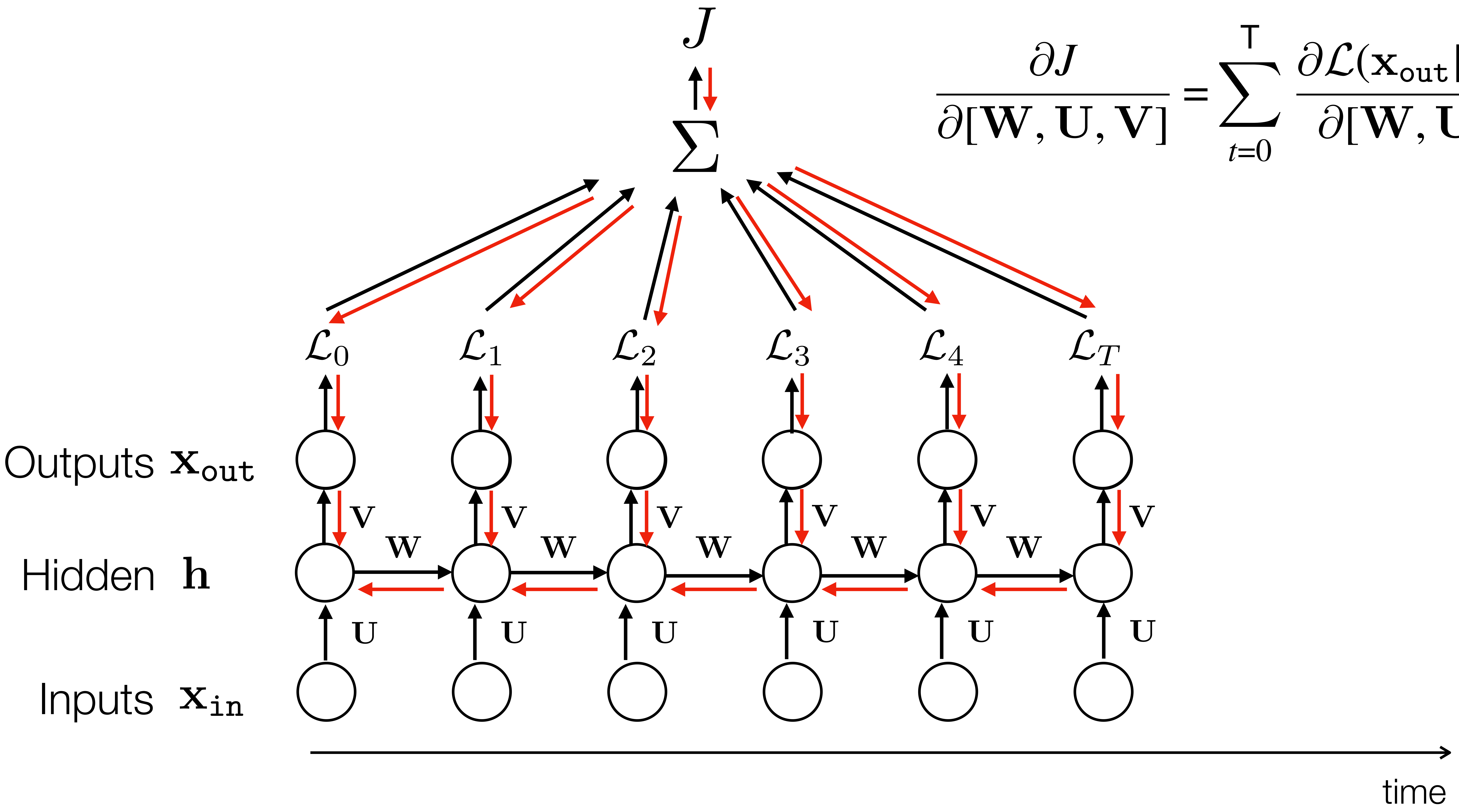


Backprop through time

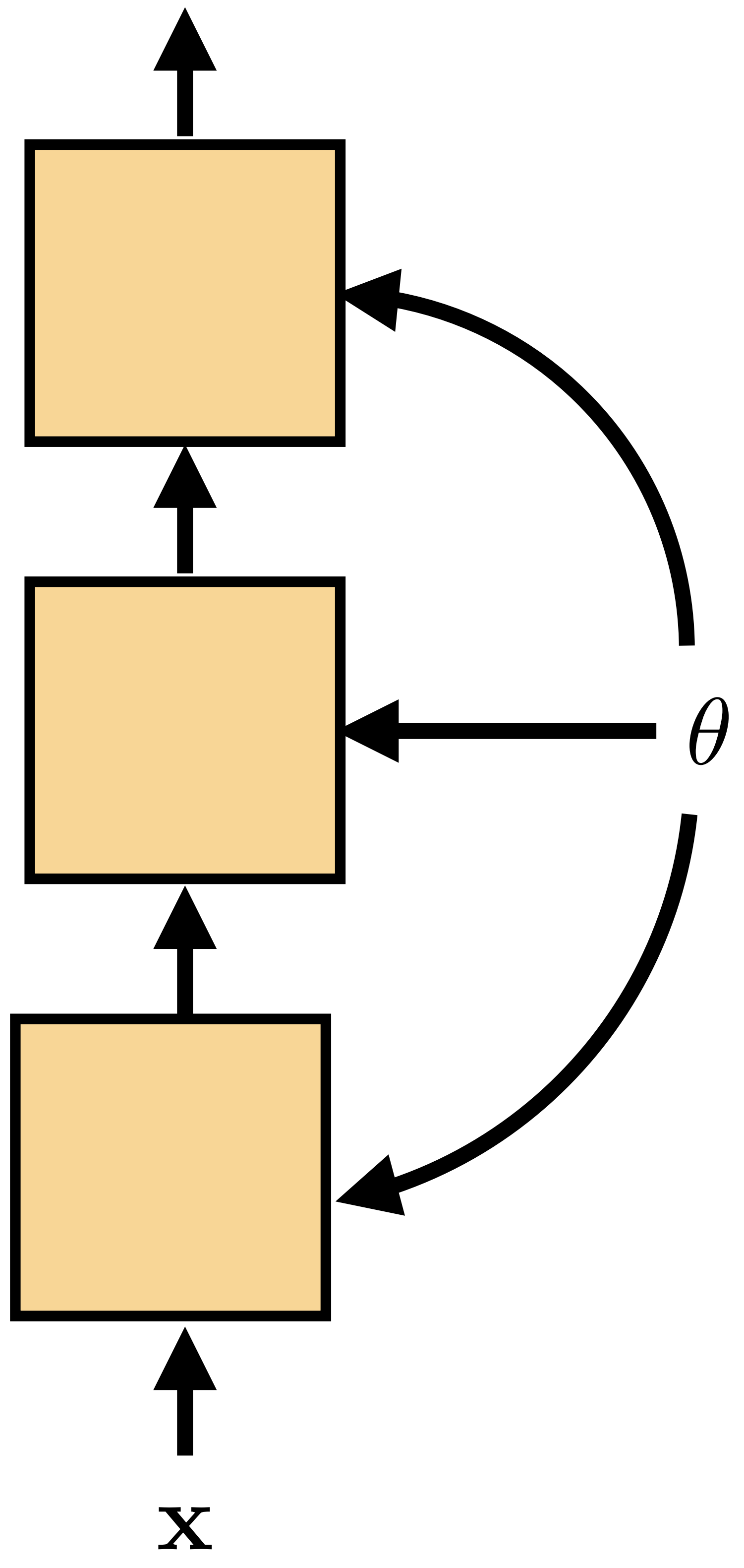


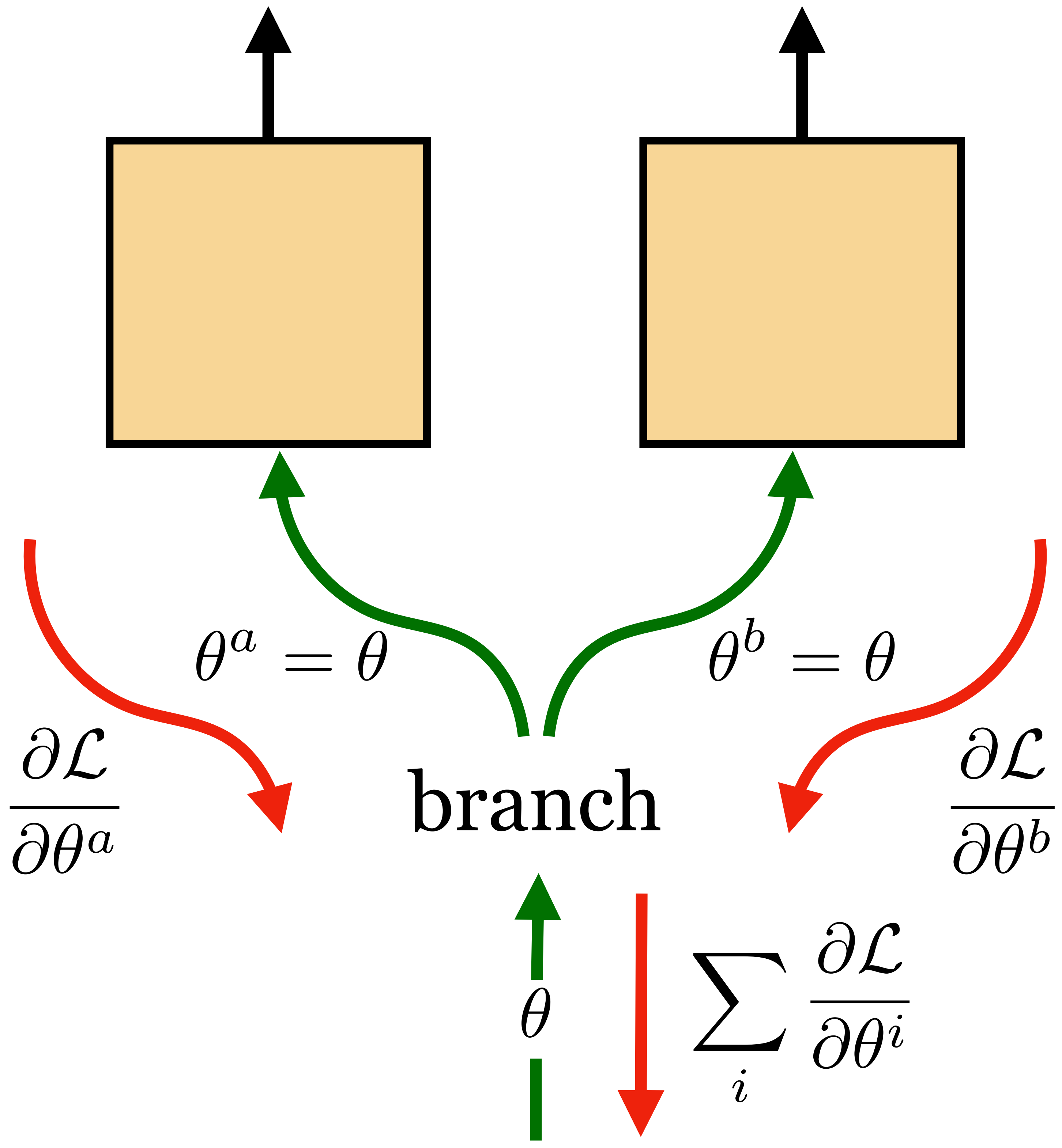
$$\frac{\partial \mathbf{x}_{\mathtt{out}}[t]}{\partial \mathbf{x}_{\mathtt{in}}[0]} = \frac{\partial \mathbf{x}_{\mathtt{out}}[t]}{\partial \mathbf{h}_T} \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_{T-1}} \cdots \frac{\partial \mathbf{h}_1}{\partial \mathbf{h}_0} \frac{\partial \mathbf{h}_0}{\partial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

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



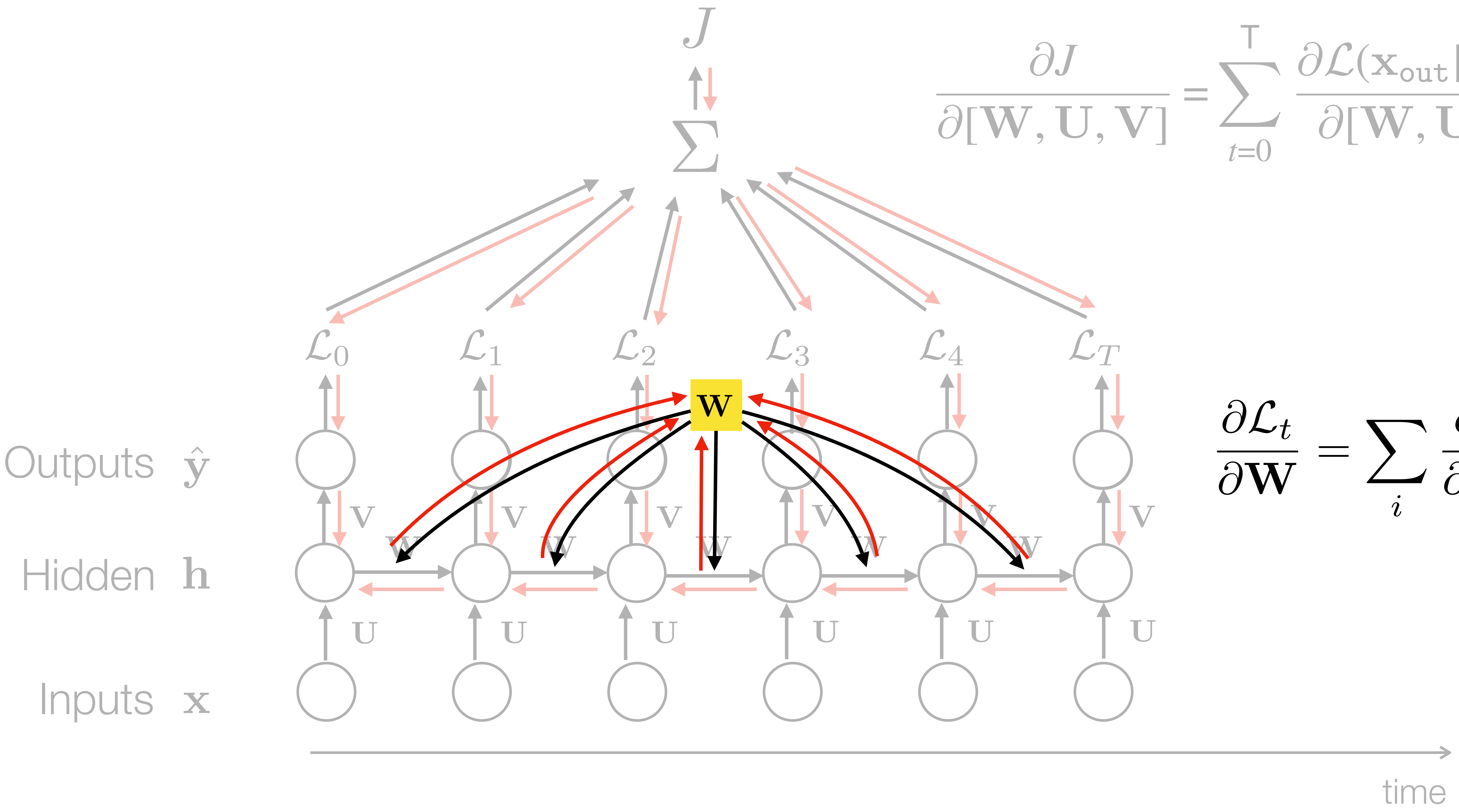
Parameter sharing





Parameter sharing —> sum gradients

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



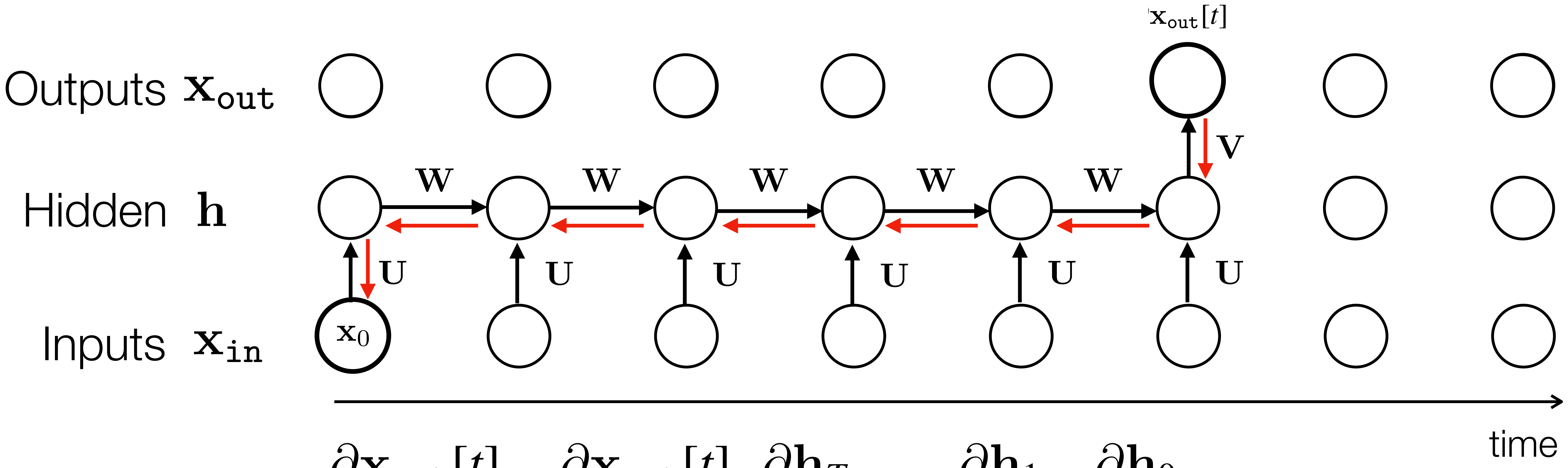
$$\frac{\partial \mathcal{L}_t}{\partial \mathbf{W}} = \sum_i \frac{\partial \mathcal{L}_t}{\partial \mathbf{W}^i}$$

The problem of long-range 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 depedences that are arbitrarily far apart

The problem of long-range dependences



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

- 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**

Optional reading: more detailed discussion of stability analysis in recursion from a control theory perspective from Bhiksha Raj @ CMU

https://www.cs.cmu.edu/~bhiksha/courses/deeplearning/Spring.2019/archive-f19/www-bak11-22-2019/document/lecture/lec13.recurrent2.pdf

LSTMs

Long Short Term Memory

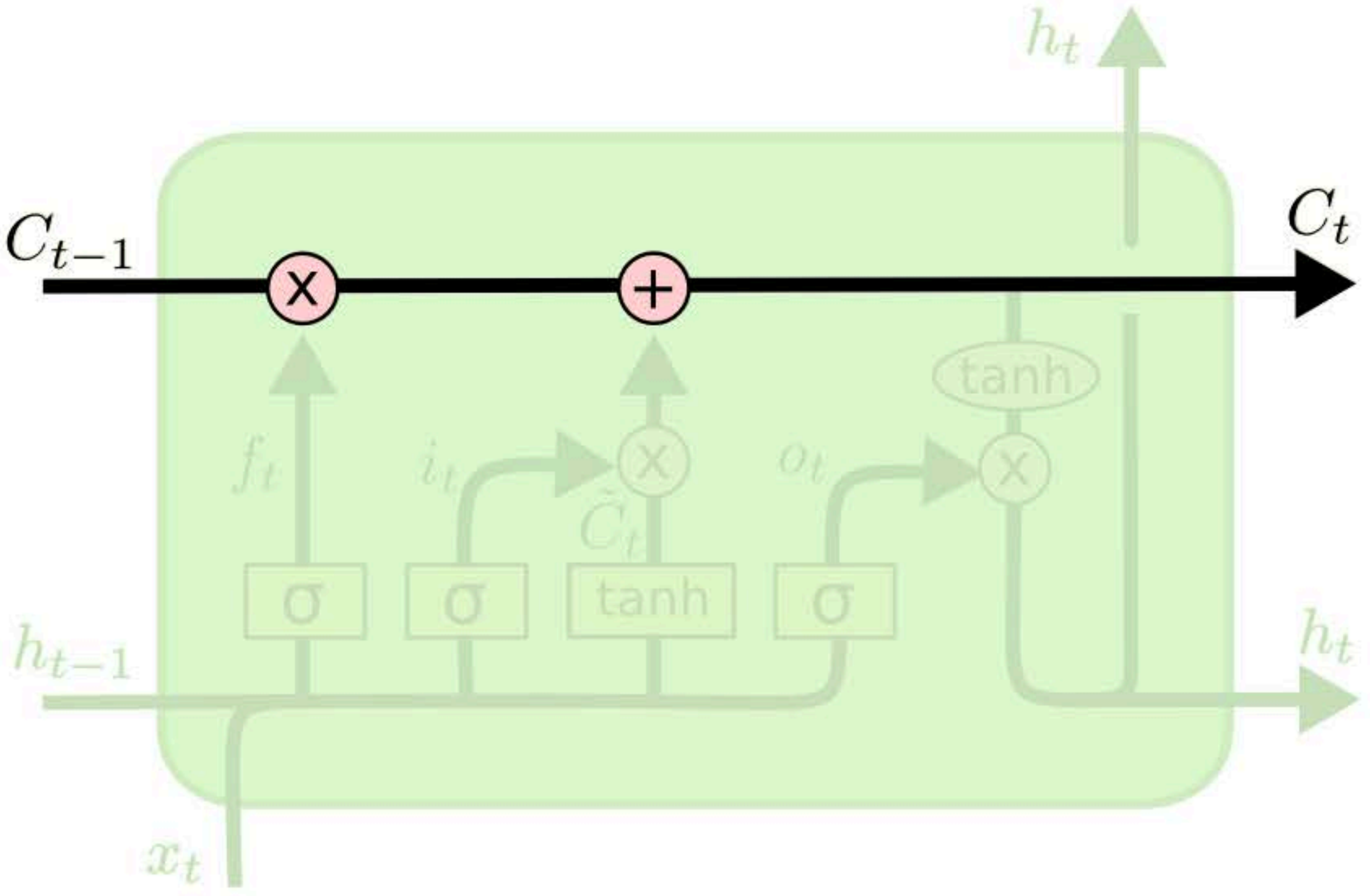
[Hochreiter & Schmidhuber, 1997]

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*.

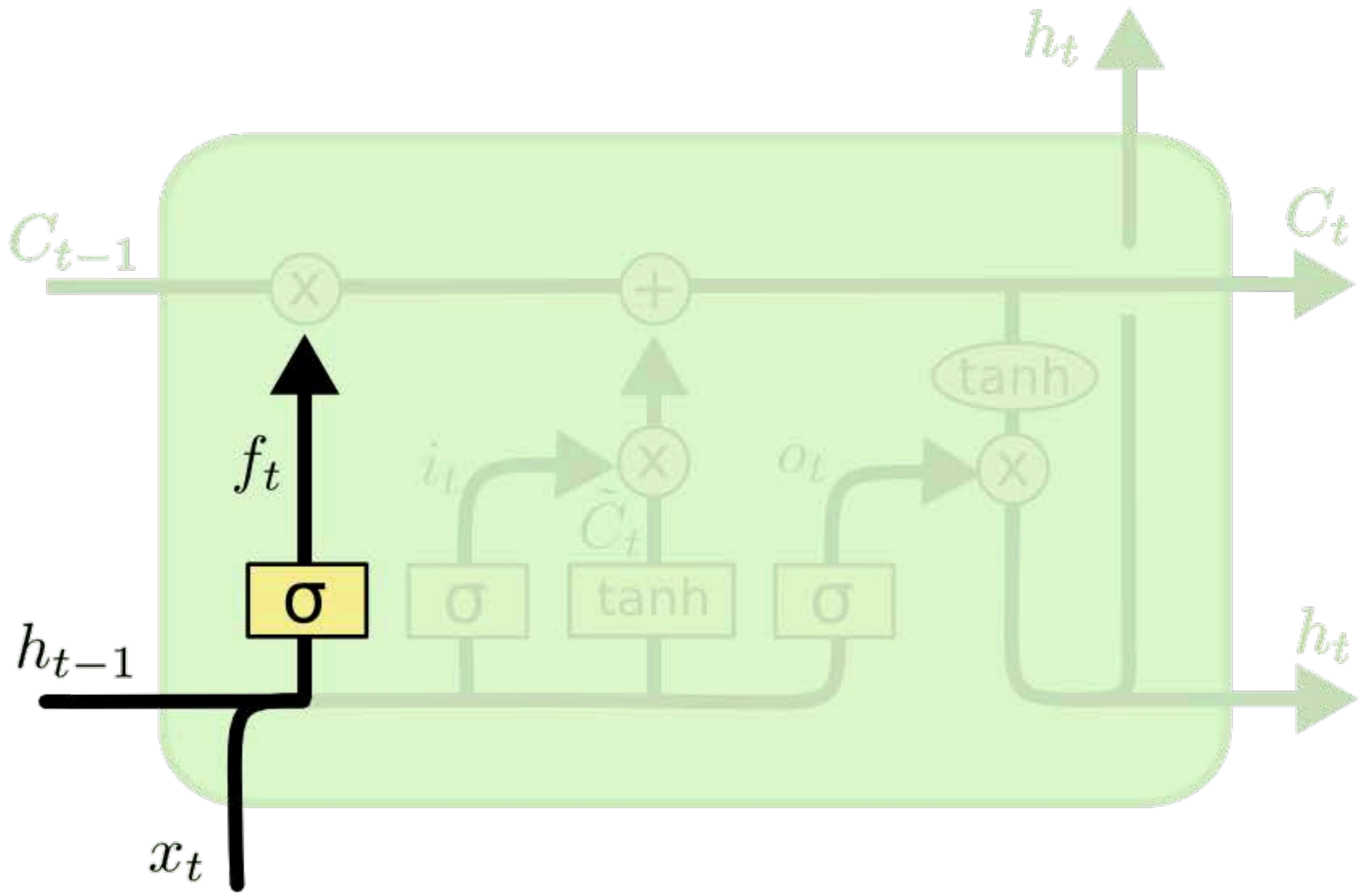
[Slide derived from Chris Olah: http://colah.github.io/posts/2015-08-Understanding-LSTMs/]

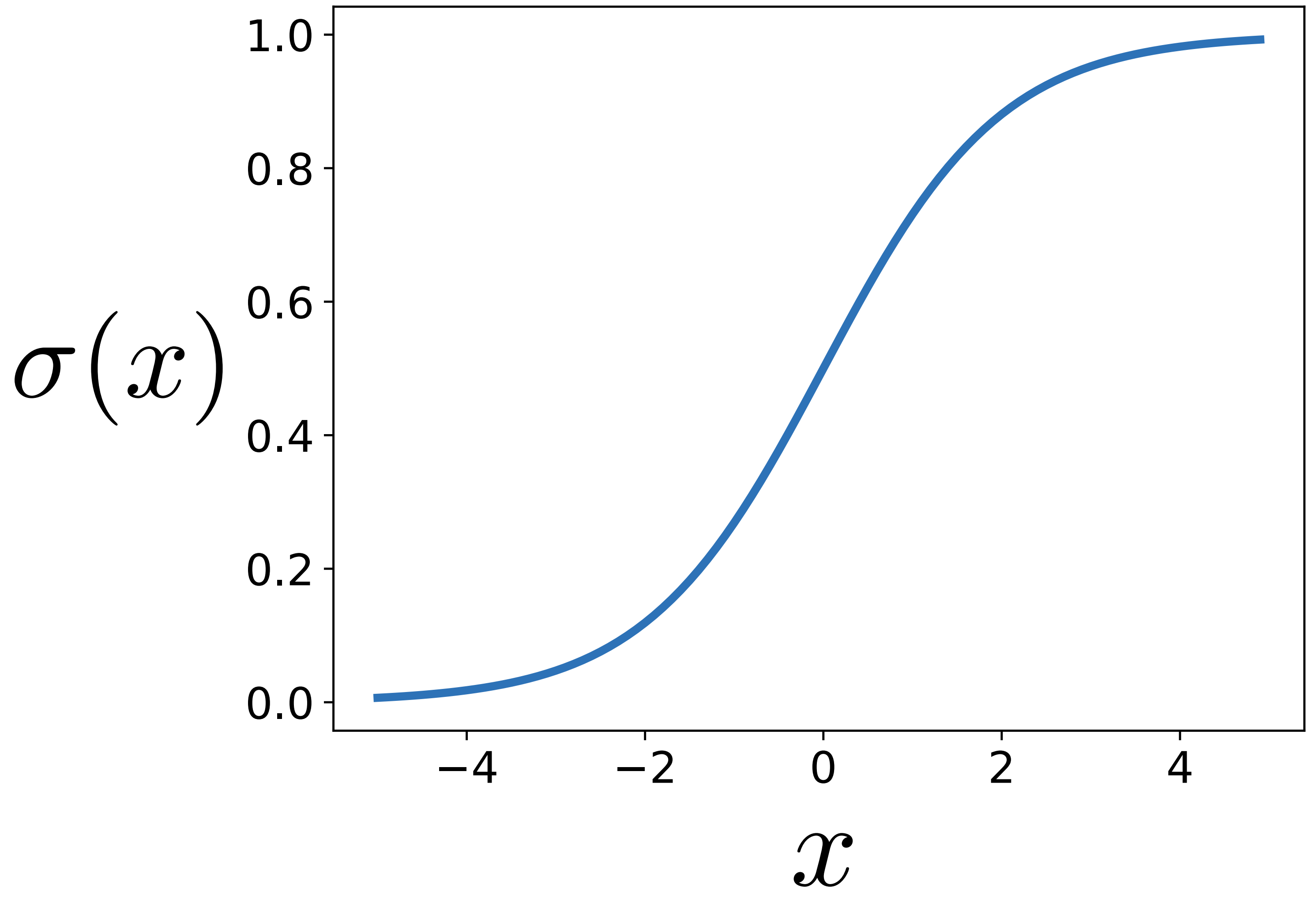
[Slide derived from Chris Olah: http://colah.github.io/posts/2015-08-Understanding-LSTMs/]



C_t = Cell state

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



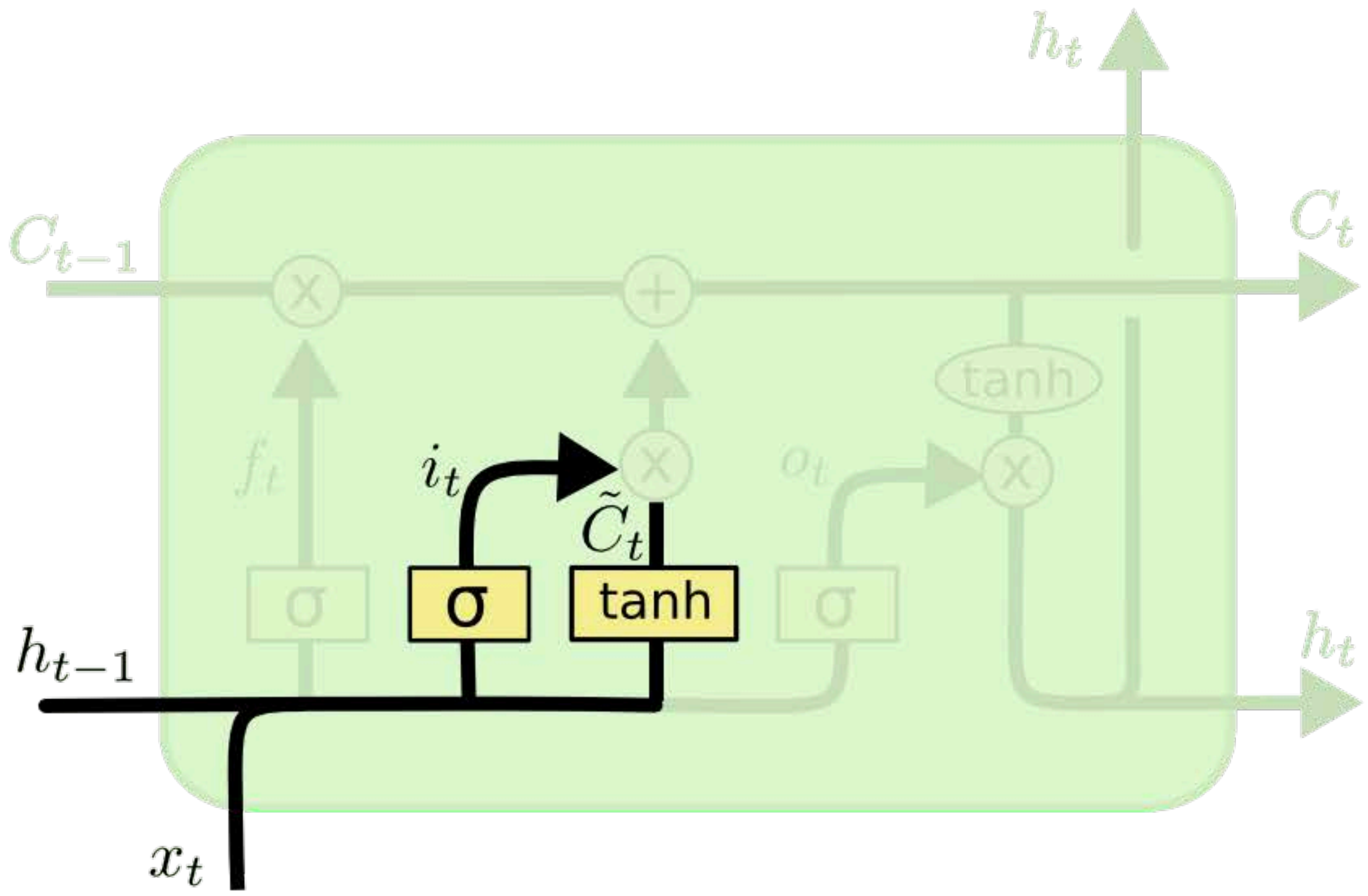


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Decide what information to throw away from the cell state.

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

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



which indices to write to

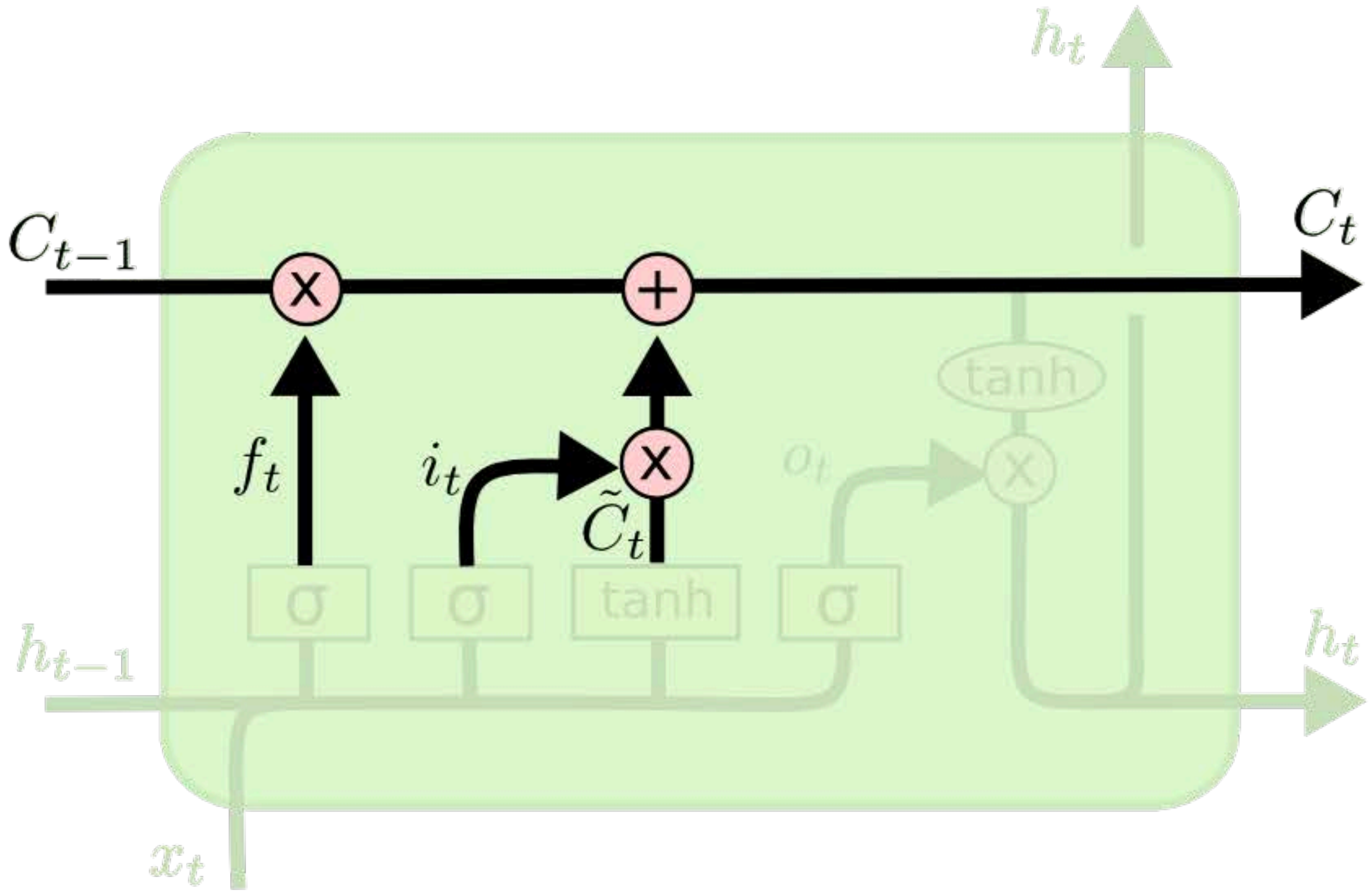
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

what to write to those indices

Decide what new information to add to the cell state.

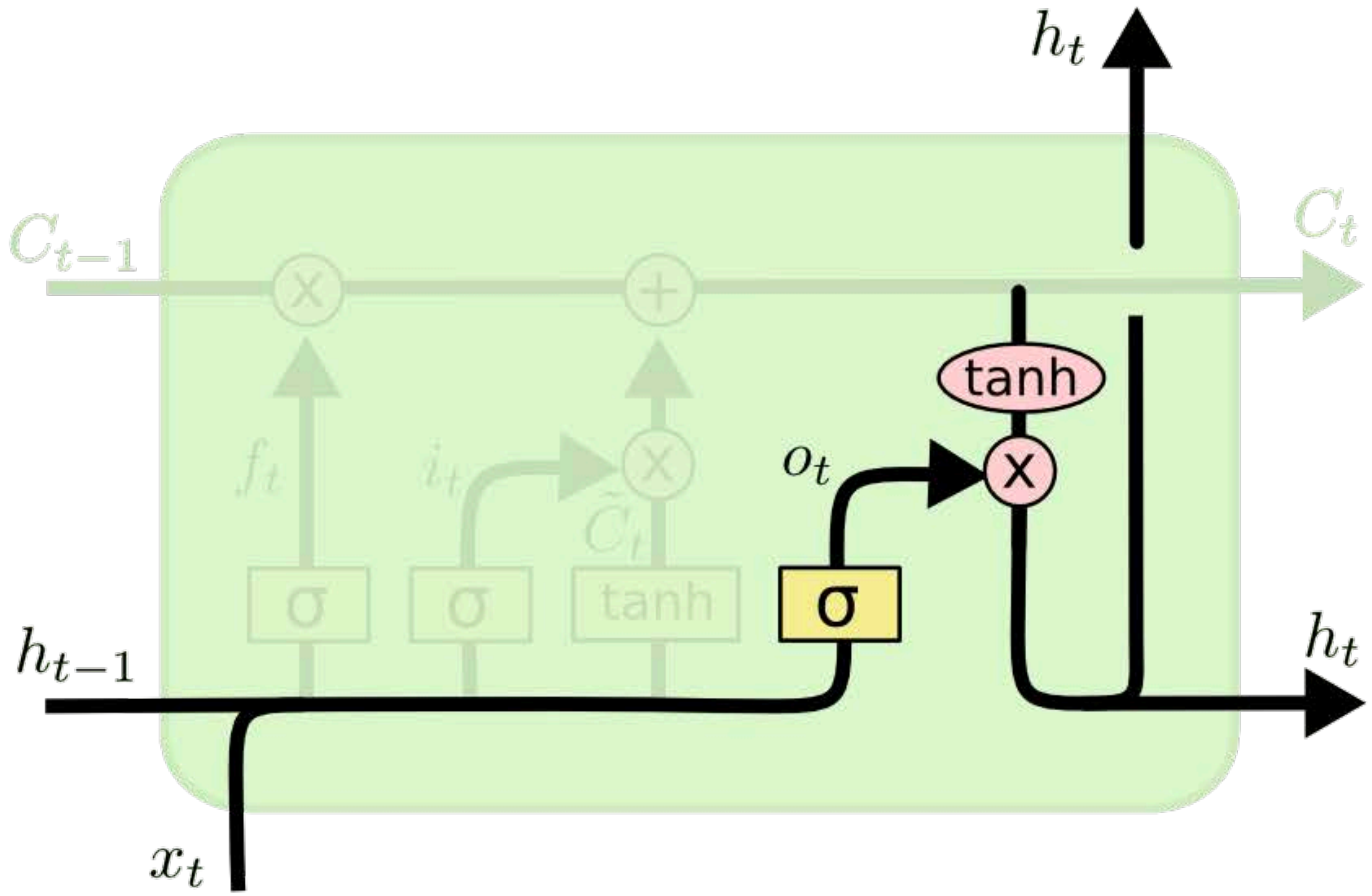
[Slide derived from Chris Olah: http://colah.github.io/posts/2015-08-Understanding-LSTMs/]



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

Forget selected old information, write selected new information.

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



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

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

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

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

Sequence models

Autoregressive models

`Once upon ___` → Predictor → `time` / `a`

`Once ___ a time` → Predictor → `Upon`


Training
$\mathbf{x}_1, \ldots, \mathbf{x}_{n-1}$
$\mathbf{x}_n$
Once upon a , time
There and back , again
The slow brown , fox
To be or not to, be
Learner
Predictor
Sampling
$\mathbf{x}_1, \ldots, \mathbf{x}_{n-1}$
Colorless green ideas sleep
Predictor
$\hat{\mathbf{x}}_n$
furiously

Autoregressive probability model

$$p(\mathbf{X}) = p(\mathbf{x_n}|\mathbf{x_1}, \ldots, \mathbf{x}_{n-1})p(\mathbf{x}_{n-1}|\mathbf{x}_1, \ldots, \mathbf{x}_{n-2}) \quad \ldots \quad p(\mathbf{x}_2|\mathbf{x}_1)p(\mathbf{x}_1)$$

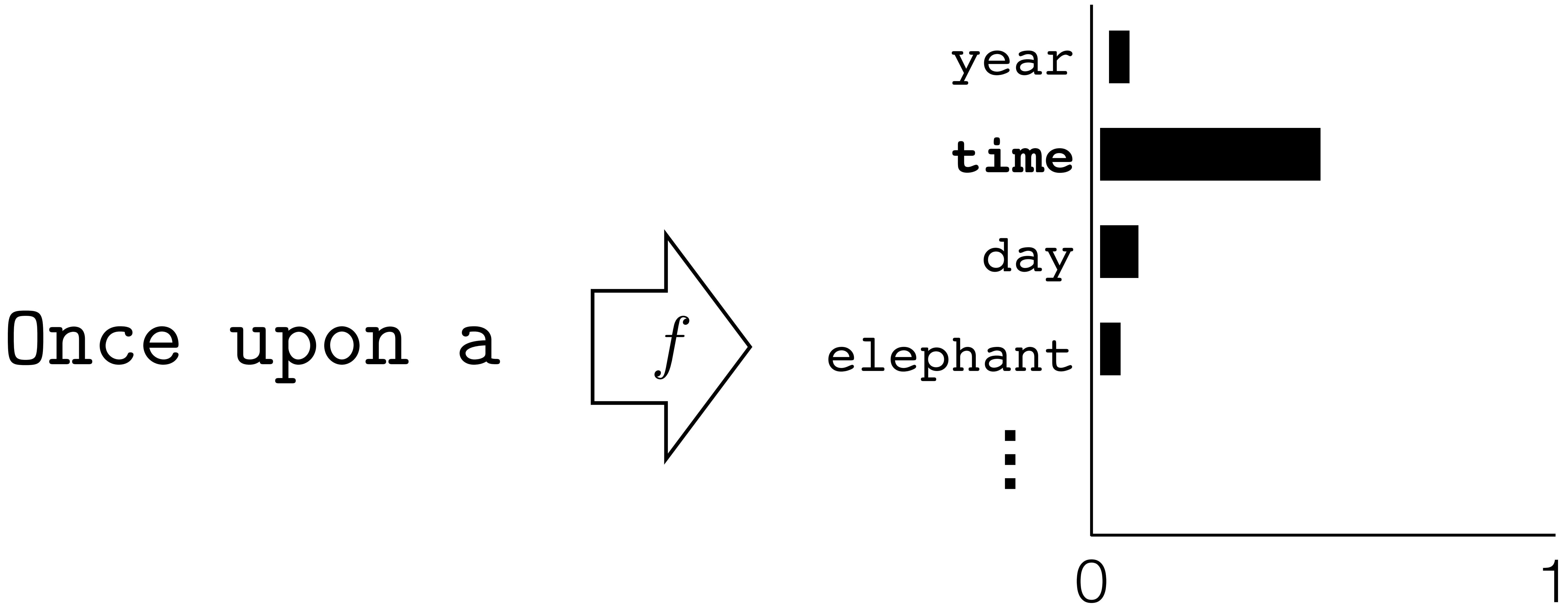
$$p(\mathbf{X}) = \prod_{i=1}^{n} p(\mathbf{x}_i|\mathbf{x}_1, \ldots, \mathbf{x}_{i-1})$$

$$p(\overbrace{\overbrace{\underbrace{\underbrace{\texttt{Once}}_{p(\texttt{Once})}\ \texttt{upon}}_{p(\texttt{upon}|\texttt{Once})}\ \texttt{a}}^{p(\texttt{a}|\texttt{Once},\texttt{upon})}\ \texttt{time}}^{p(\texttt{time}|\texttt{Once},\texttt{upon},\texttt{a})})$$

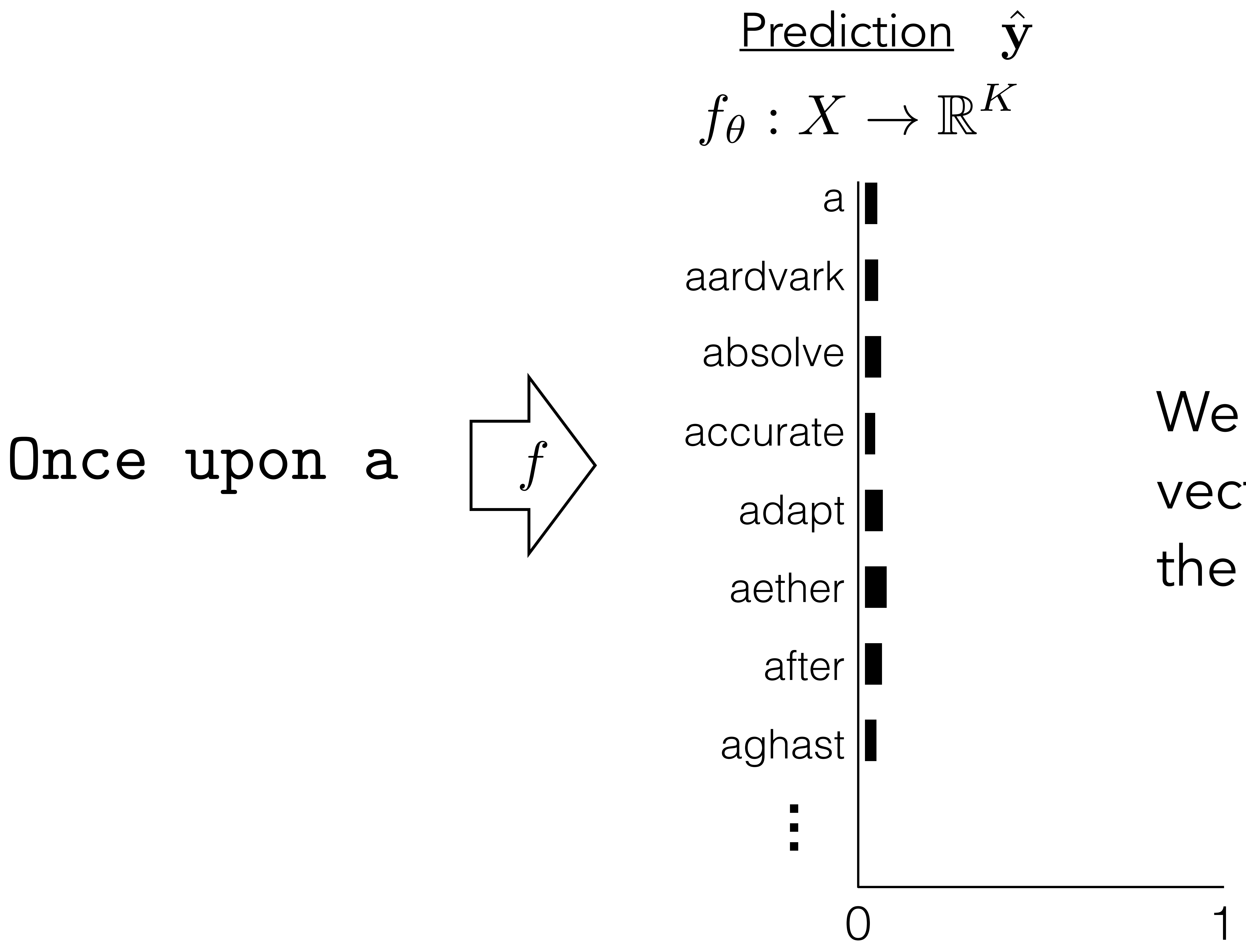
Modeling a sequence of words

How to model $p(\texttt{time}|\texttt{Once}, \texttt{upon}, \texttt{a})$?

Just treat it as a next word classifier!

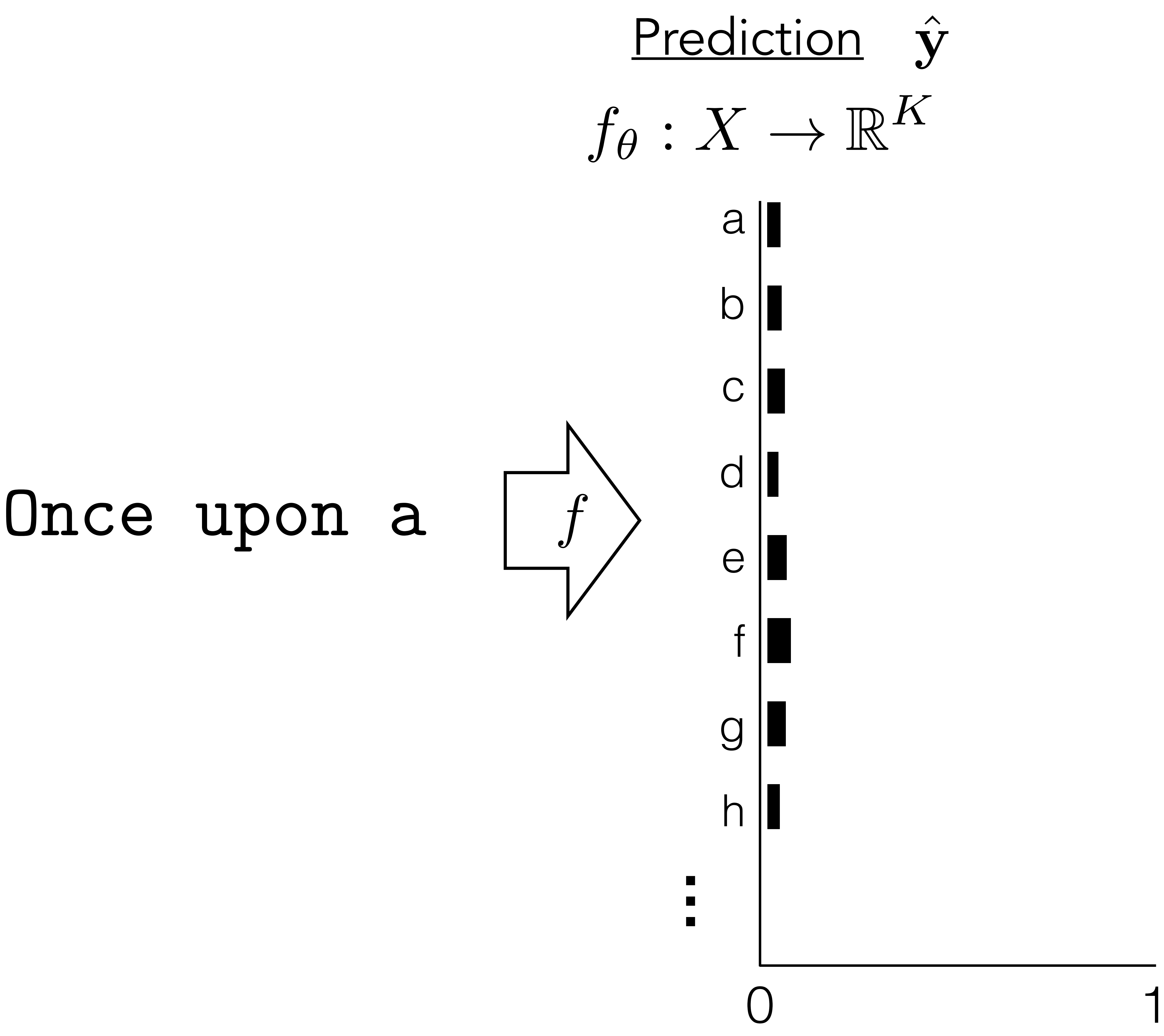


How to represent words as numbers?



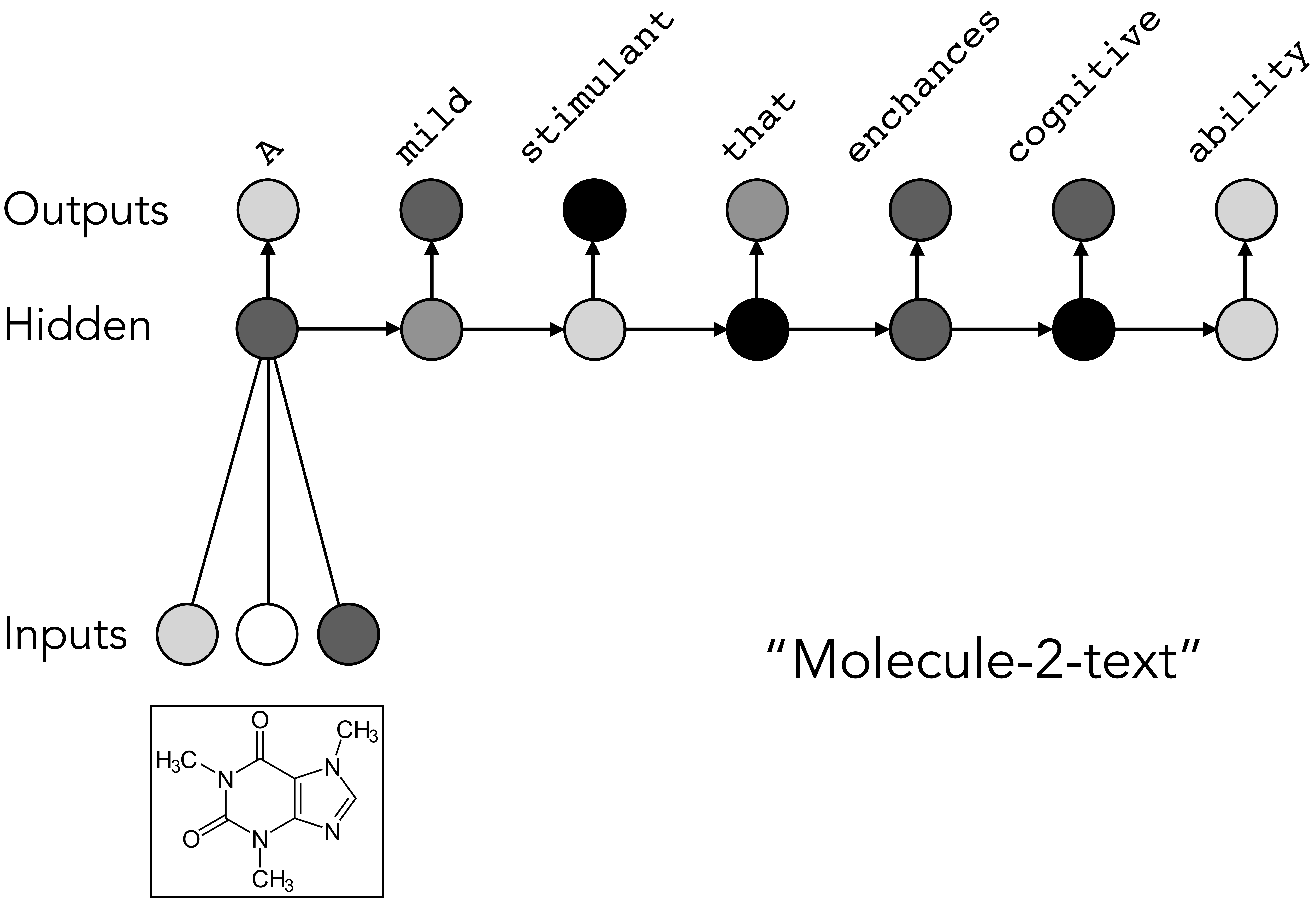
We can represent words as 1-hot-vectors of size K, where K is the size the vocabulary (e.g., K=100,000).

How to represent words as numbers?

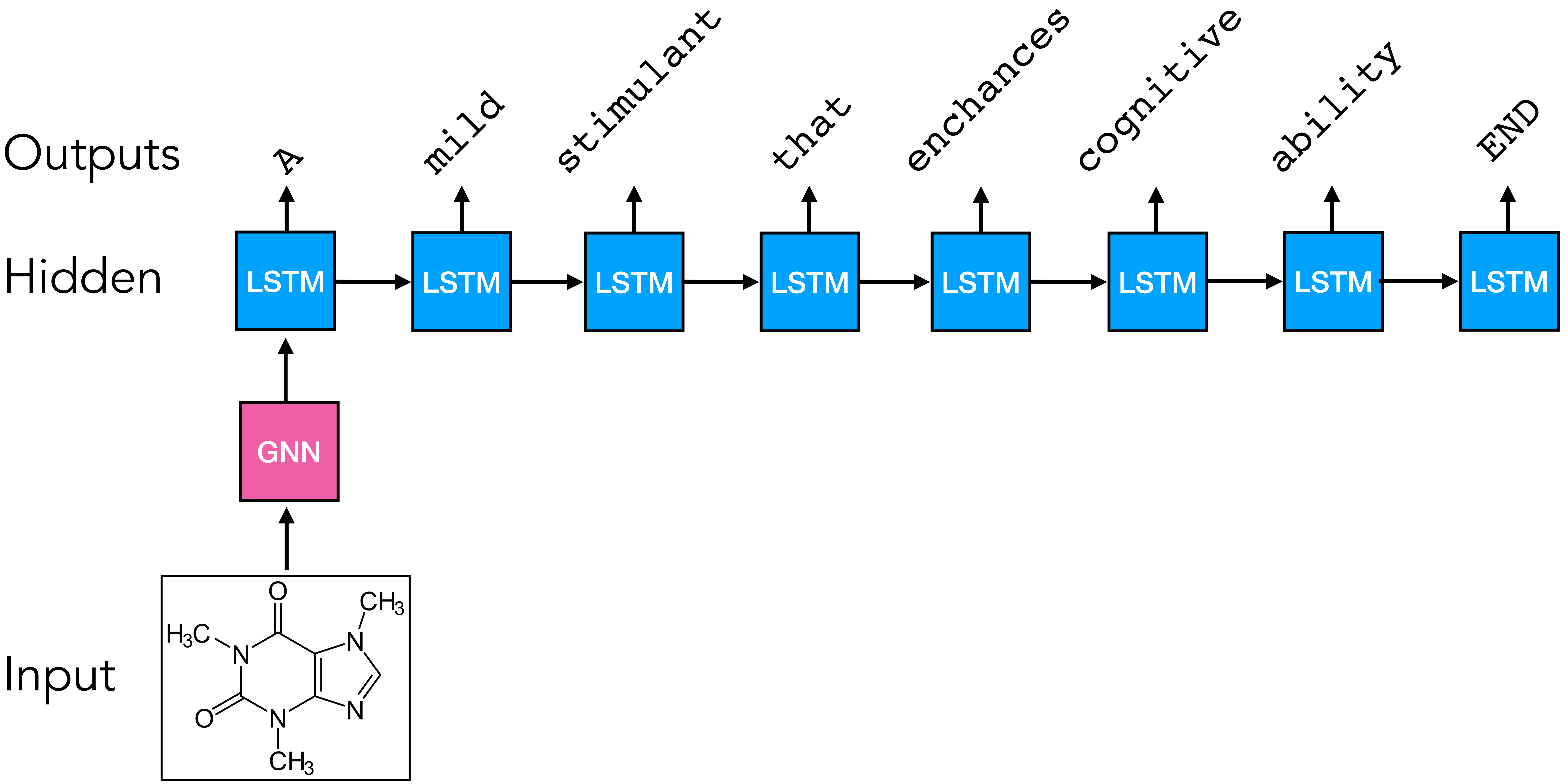


Or, represent each character as a class (e.g., K=26 for English letters),

and represent words as a sequence of characters.

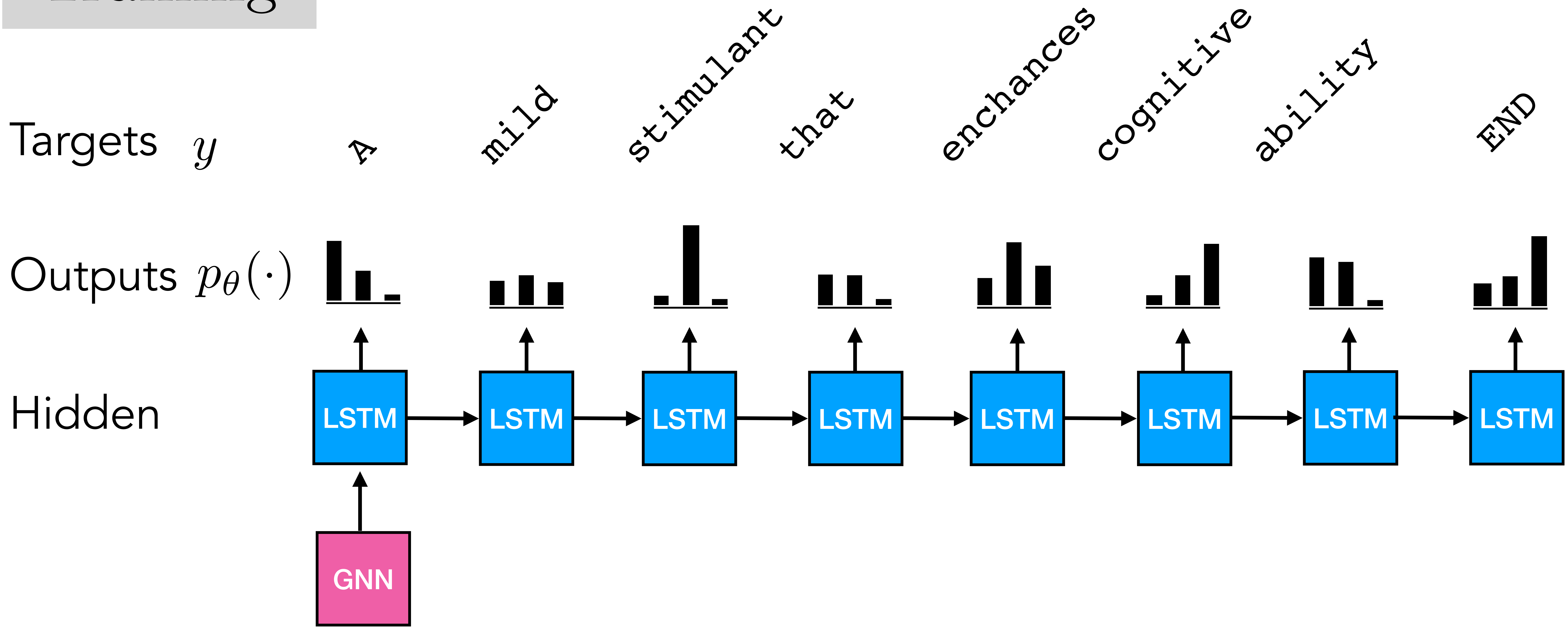


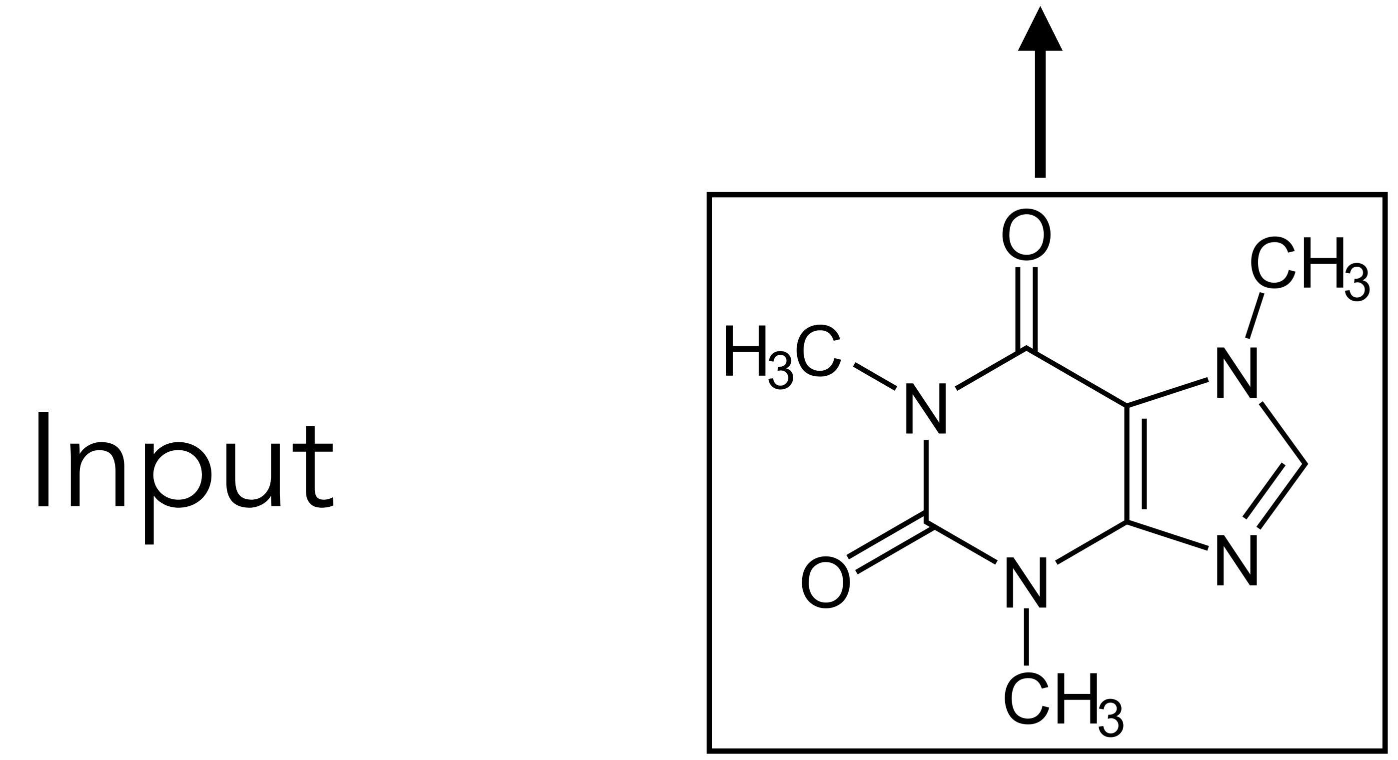
“Molecule-2-text”



Outputs
A
mild
stimulant
that
enchances
cognitive
ability
END
Hidden
LSTM
LSTM
LSTM
LSTM
LSTM
LSTM
LSTM
LSTM
GNN
Input
O
CH_3
H_3C
N
N
N
O
N
CH_3

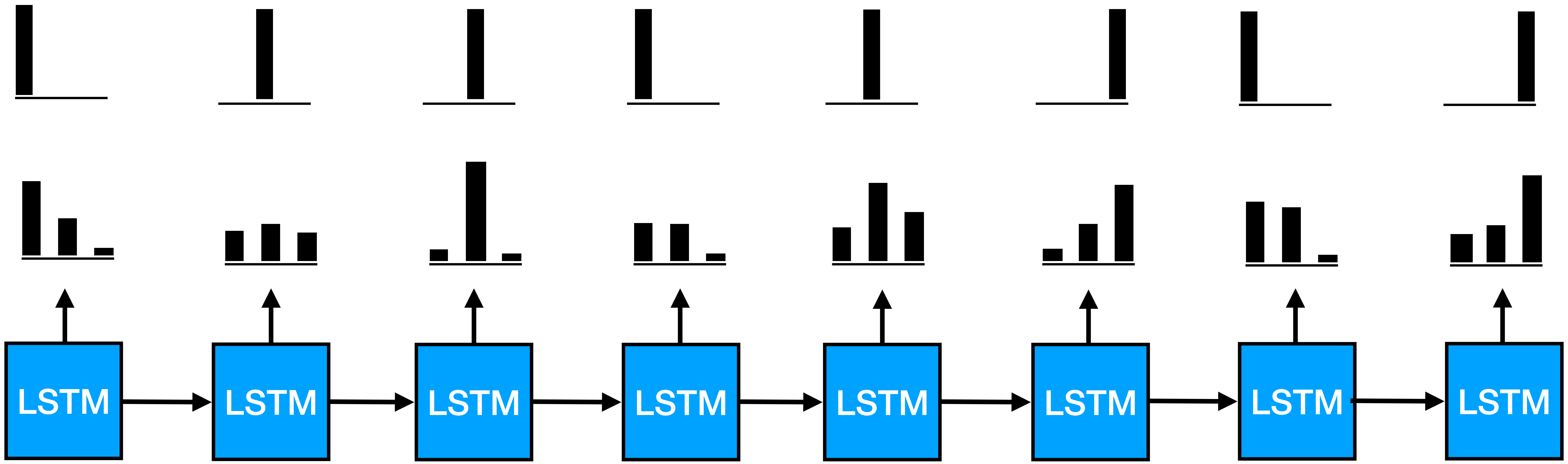
Training

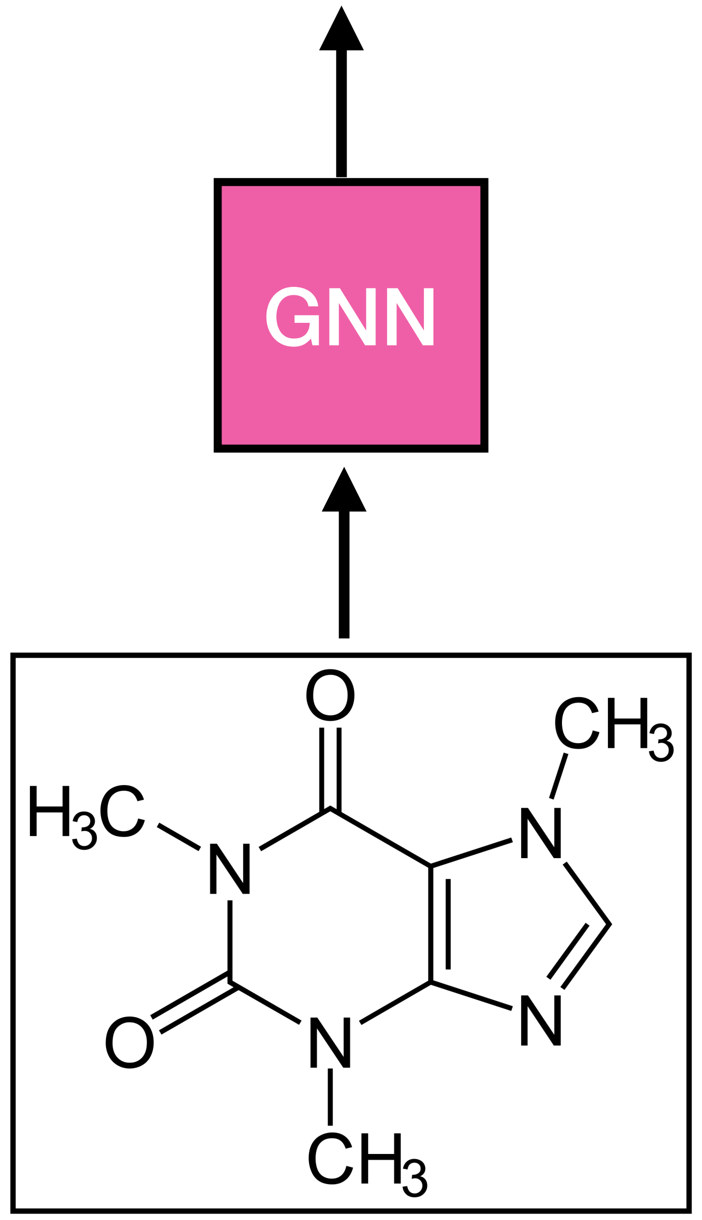




Max-likelihood objective: maximize probability the model assigns to each target word: $\arg\max_\theta \log p_\theta(y)$

Training



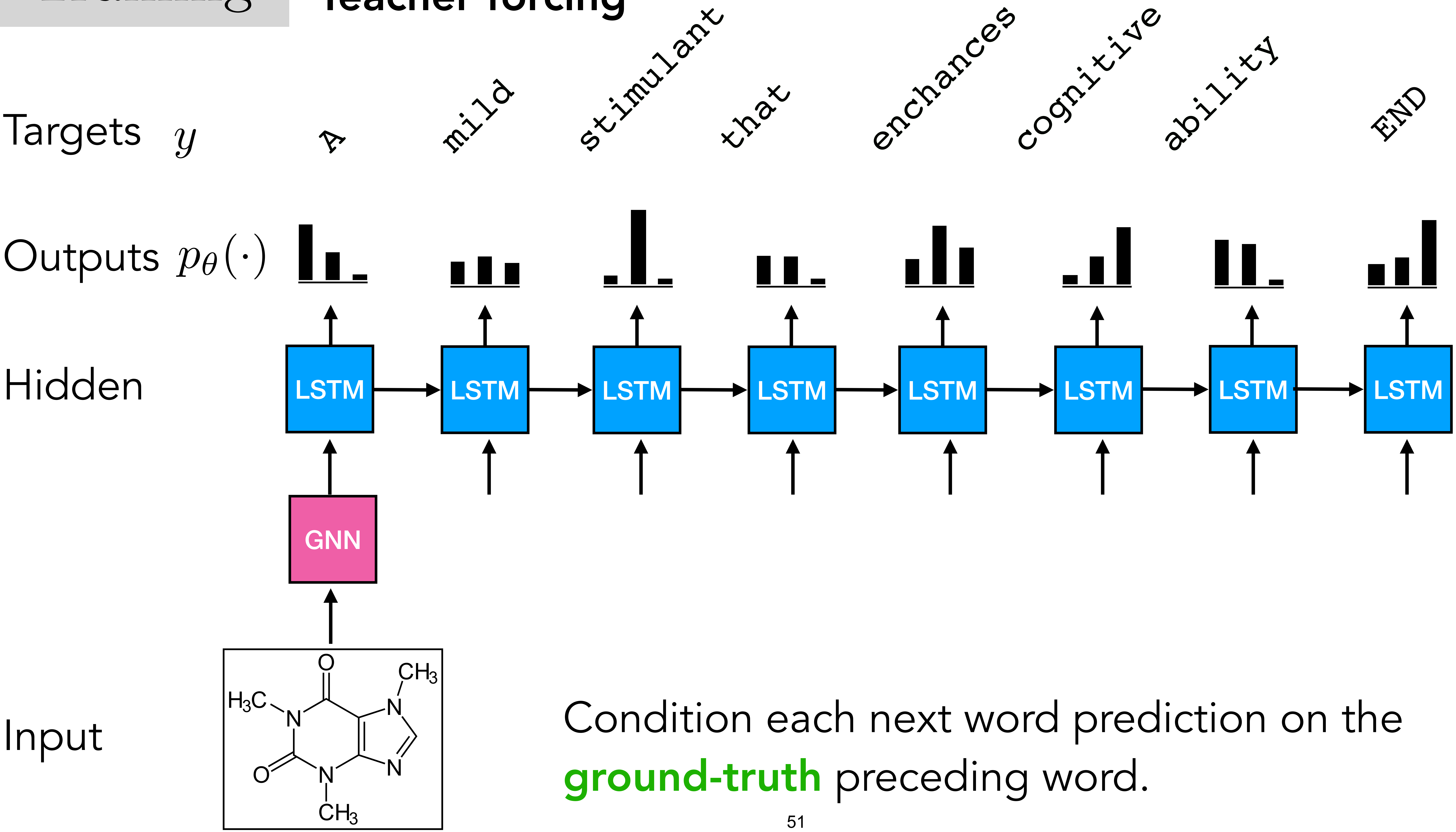


Max-likelihood objective:
minimize cross-entropy between model outputs and one-hot encoded targets.

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

Training

Teacher forcing



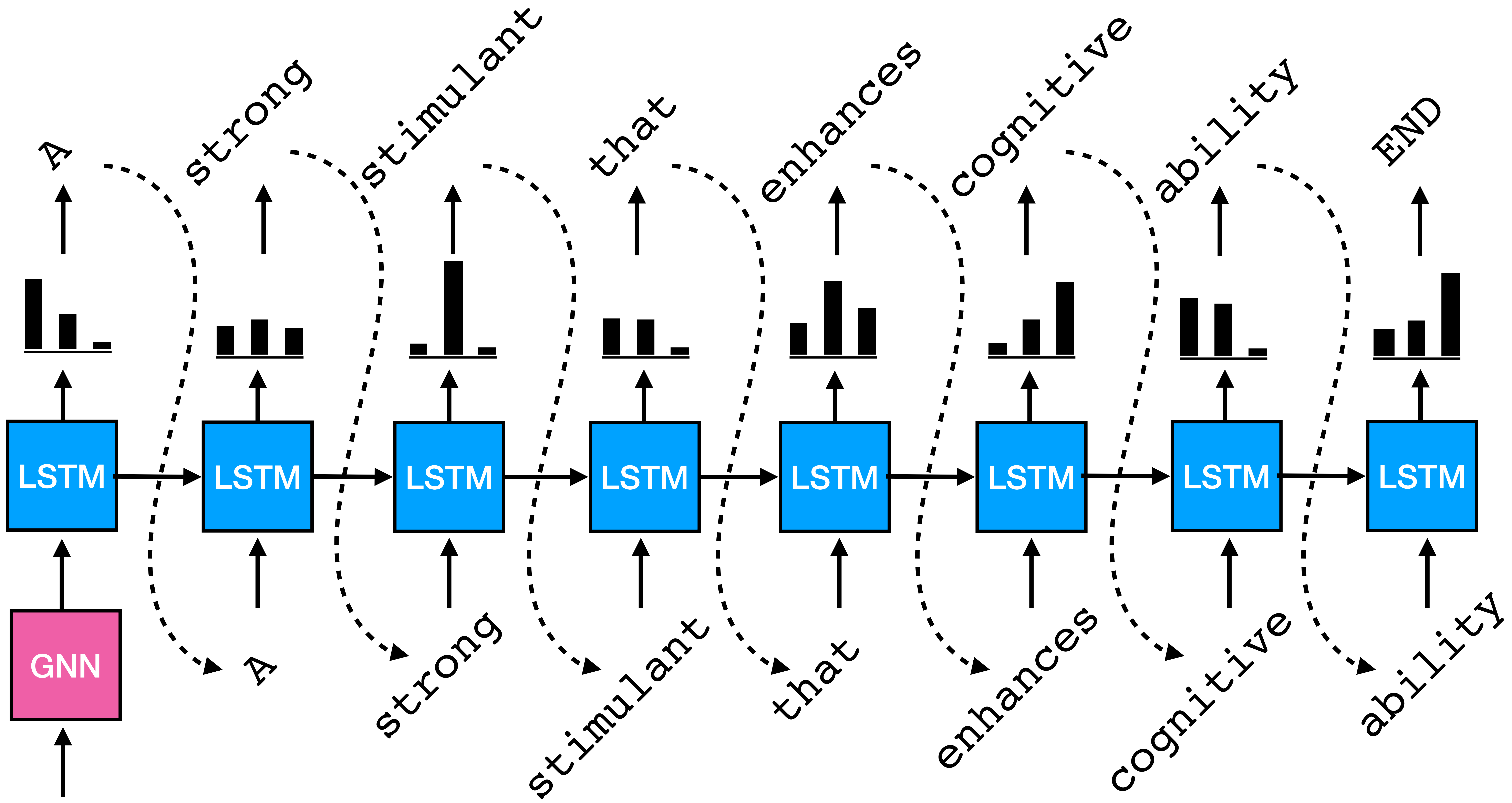
Condition each next word prediction on the **ground-truth** preceding word.

Testing

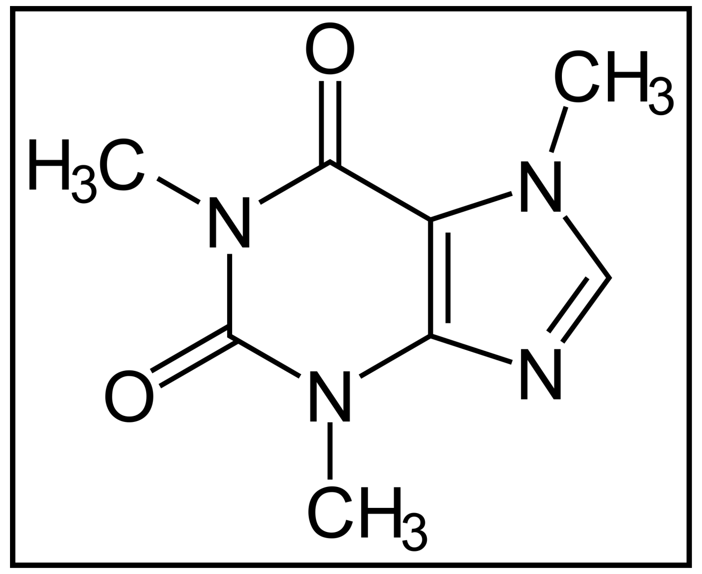
Samples

Outputs $p_\theta(\cdot)$

Hidden



Input



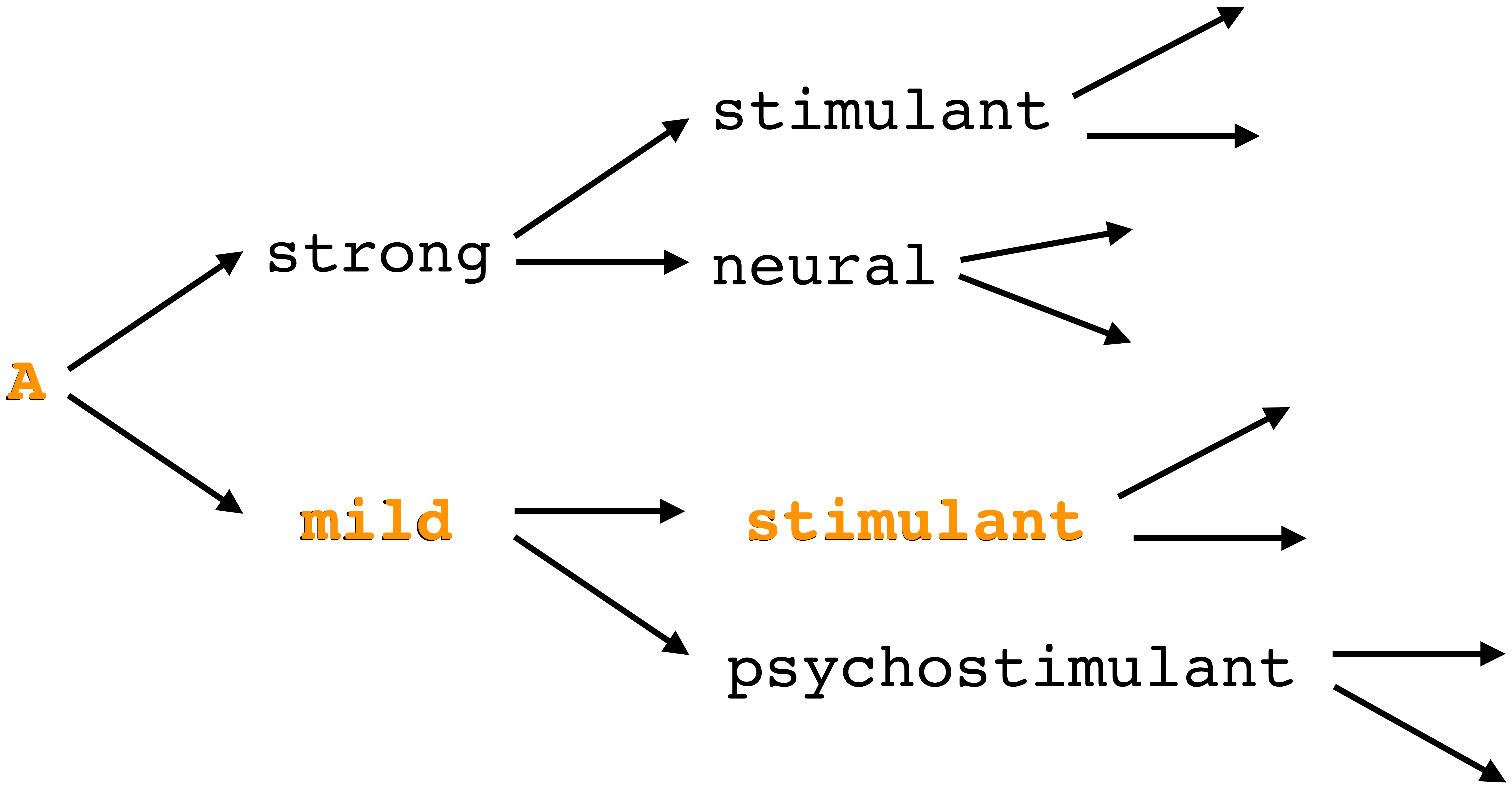
Sample from predicted distribution over words.

Alternatively, sample most likely word.

Testing

Beam search

Tree of samples



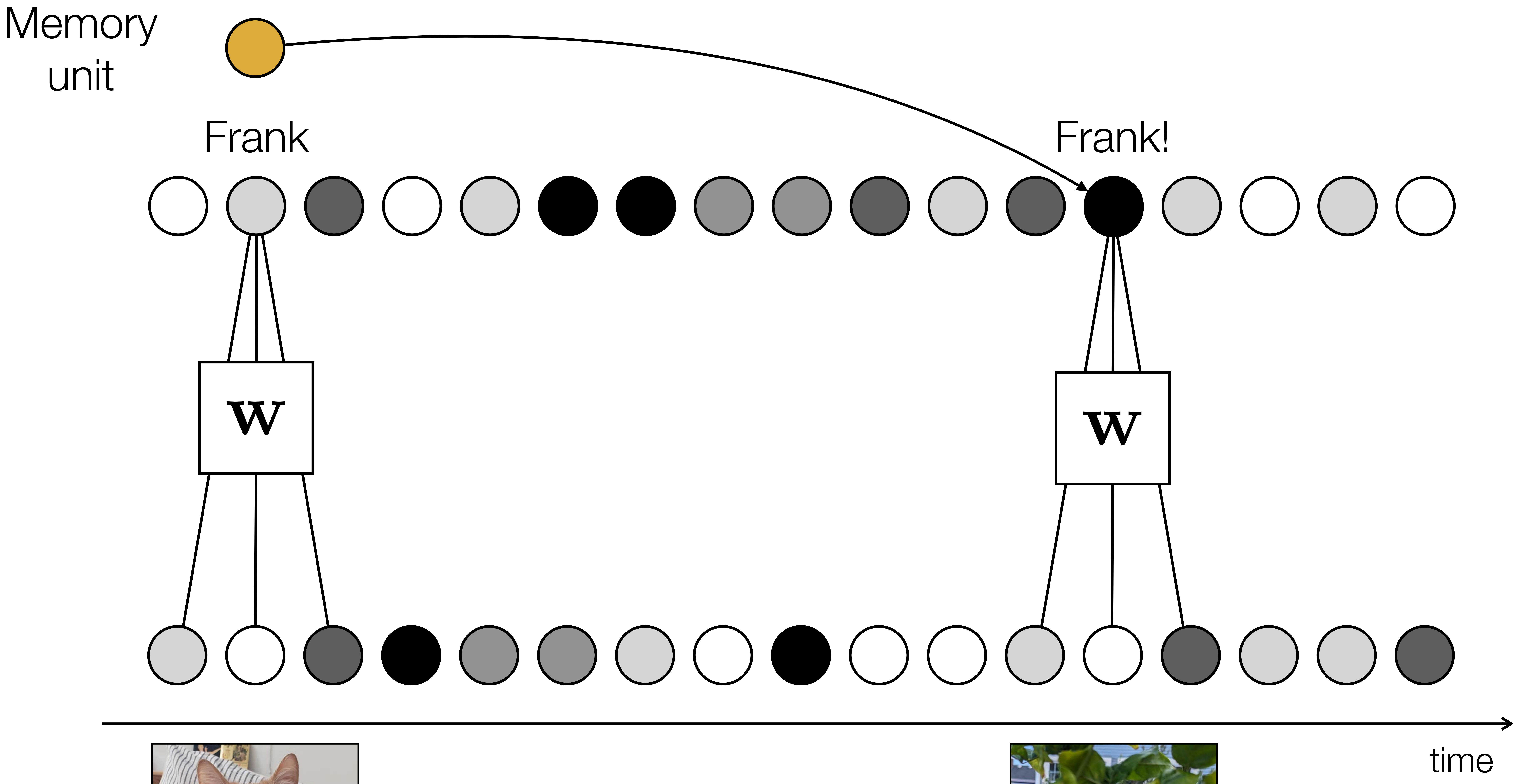
Sample multiple sequences (top-k greedy completions on each step), then pick the sequence with highest score.

Score could be model's confidence: $p_\theta(\mathbf{y}_1, \ldots, \mathbf{y}_T | \mathbf{x})$

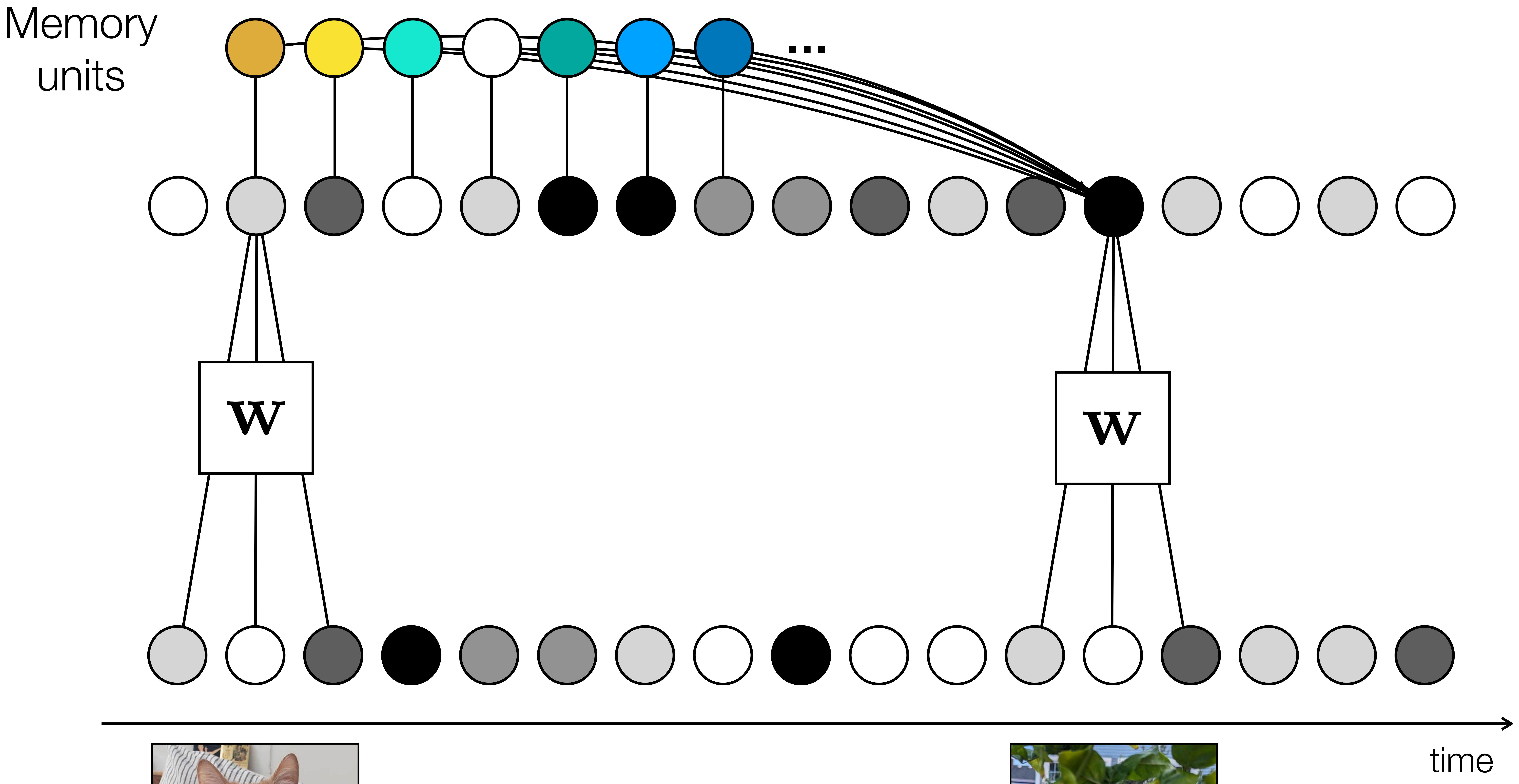
The problem of long-range dependences

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 depedencies that are arbitrarily far apart



Memory unit
Frank
Frank!
W
W
time


Memory units
...
W
W
time

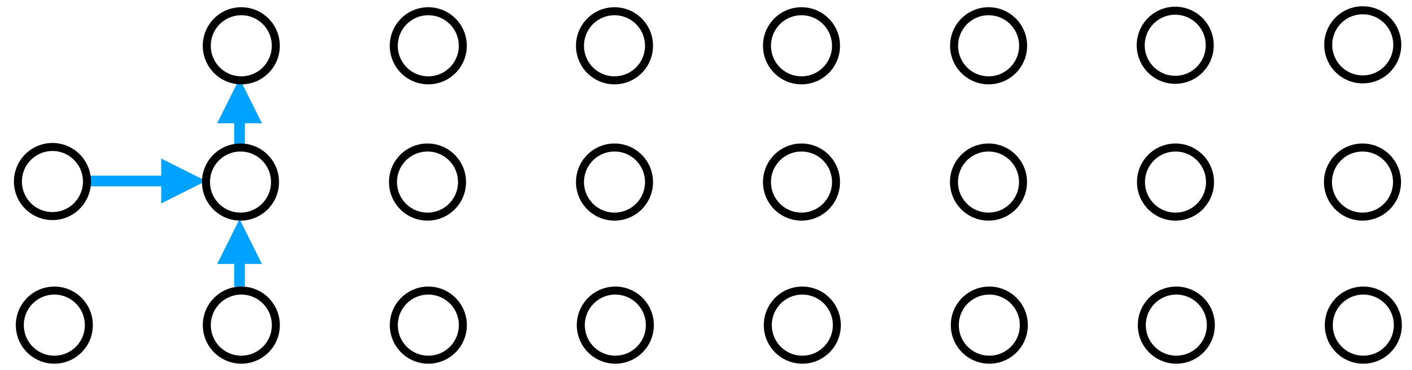
The problem of long-range dependences

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

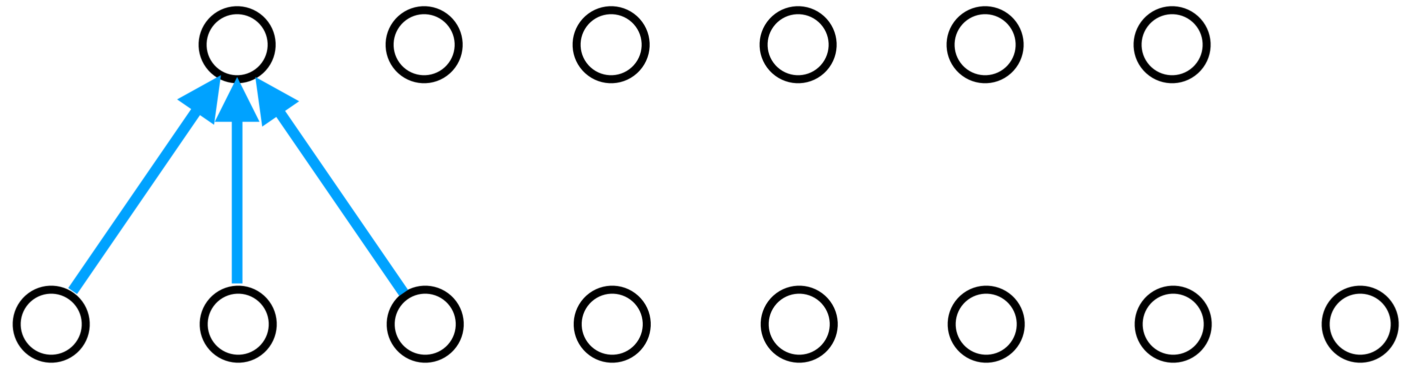
- Temporal convolutions
- Attention / Transformers (see https://arxiv.org/abs/1706.03762)
- Memory networks (see https://arxiv.org/abs/1410.3916)

Modeling arbitrarily long sequences

- **Recurrence** — recurrent weights are shared across time



- **Convolution** — conv weights are shared across time



- **Attention** — weights are dynamically determined as a function of the data (conv kernel with attention weights is shown on the right)

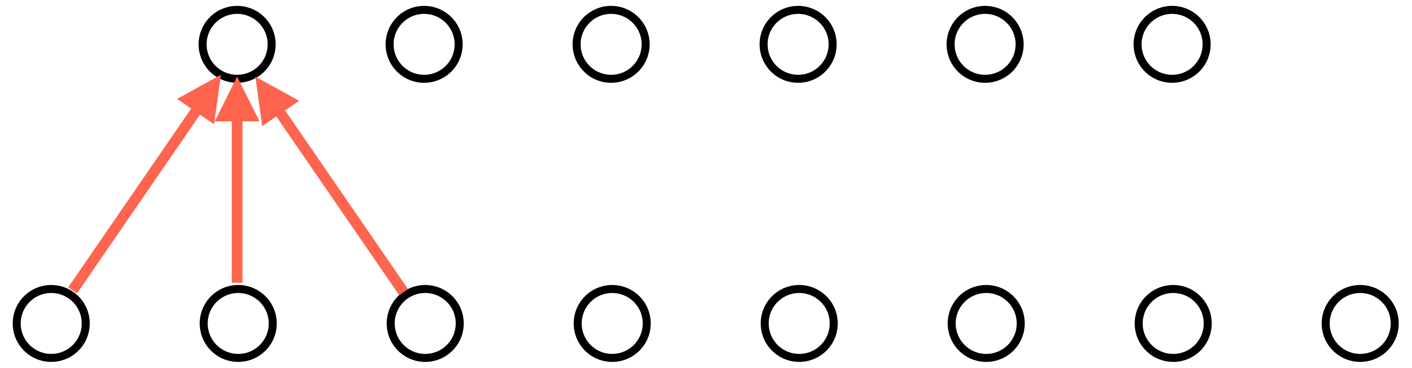


Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

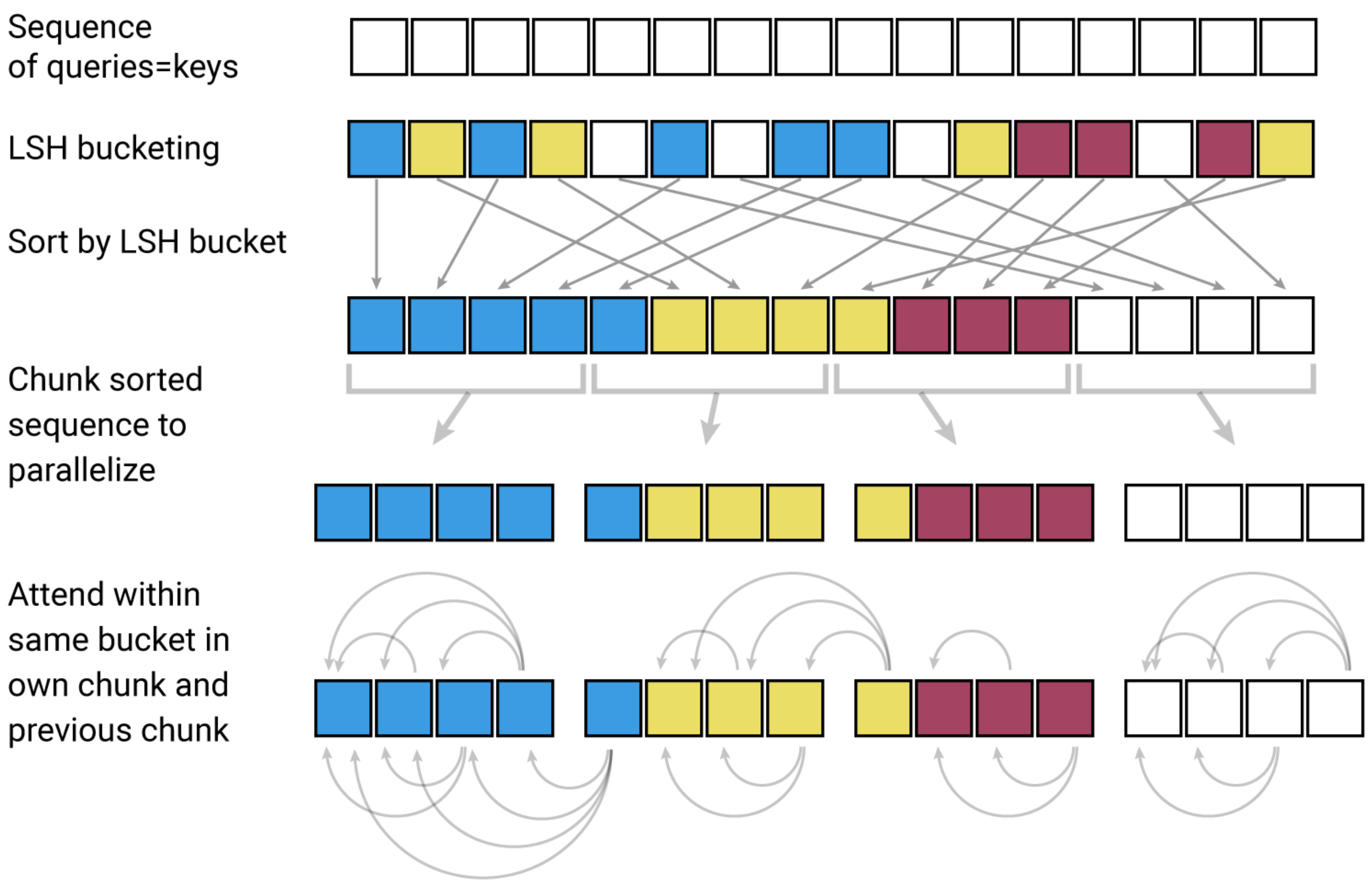
Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

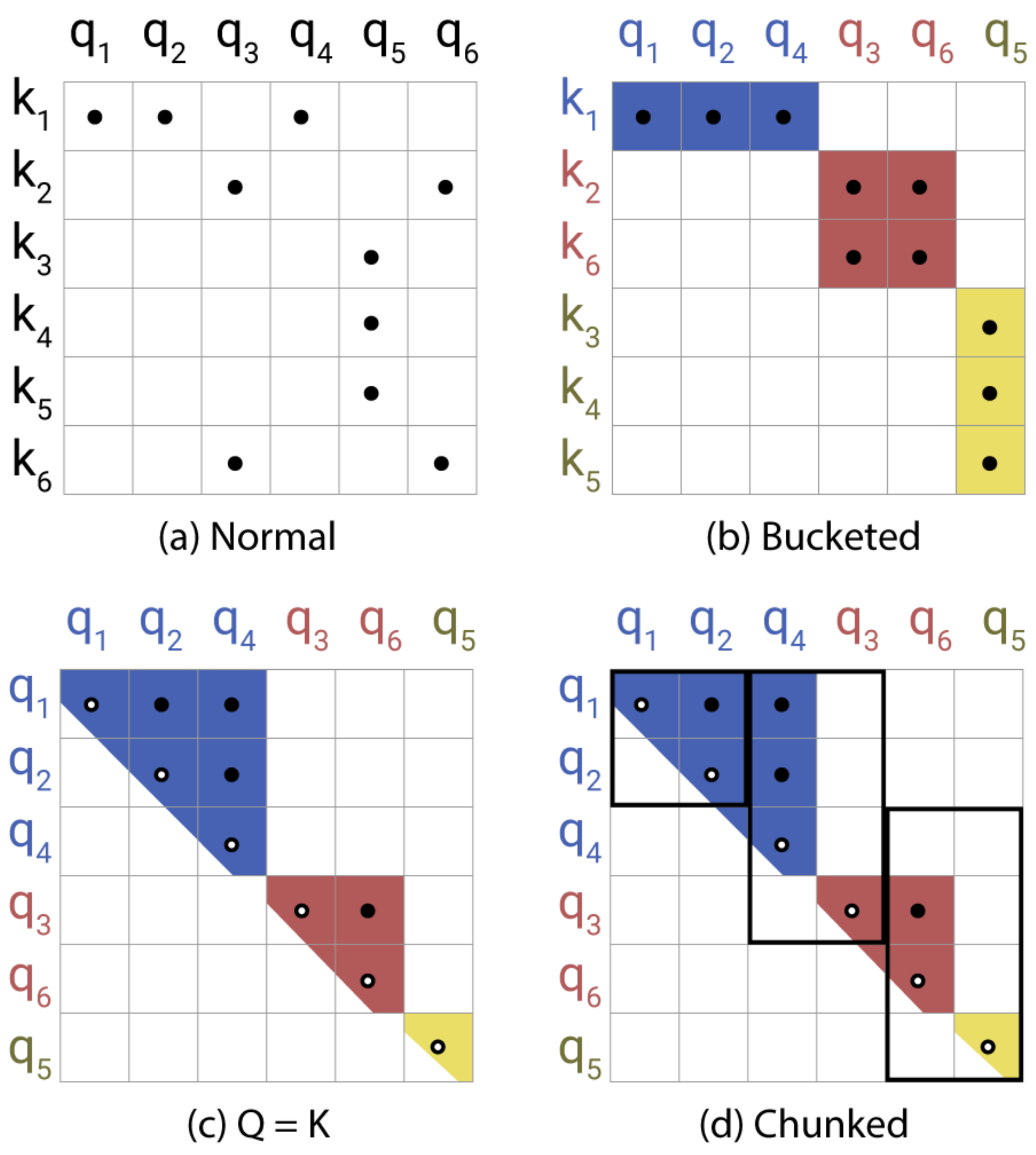
[“Attention is All you Need”, https://arxiv.org/abs/1706.03762]

Even-larger-context transformers

Efficiency from sparsification

- **Reformer** replaces quadratic dot product attention with a mechanism that uses local hashing to get to O(n log n) (https://arxiv.org/abs/2001.04451)
- **Performers** introduce the use of positive orthogonal random features within attention to get to O(n) (https://arxiv.org/pdf/2009.14794)
- **Linformers** use low-rank matrix approximation to get O(n) in time and space (https://arxiv.org/pdf/2006.04768)

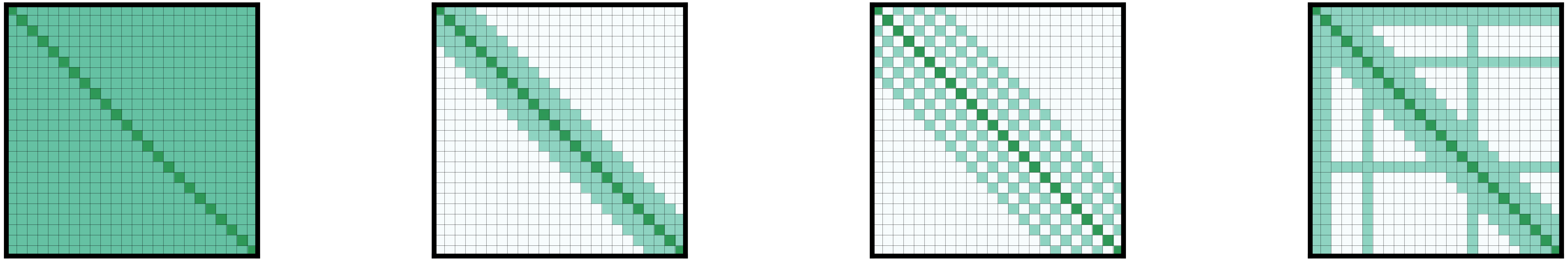




Even-larger-context transformers

Local + global

- **Transformer XL** uses segment-level recurrence and fancy positional encoding to increase context (https://arxiv.org/abs/1901.02860)
- **Longformer** scales self-attention linearly with sequence length as opposed to quadratically, using deconstructed local + global attention (https://arxiv.org/abs/2004.05150)
- **Big Bird** uses a combo of random, dense sliding window, and global token attention to get sparsity, also O(n) (https://arxiv.org/pdf/2007.14062)



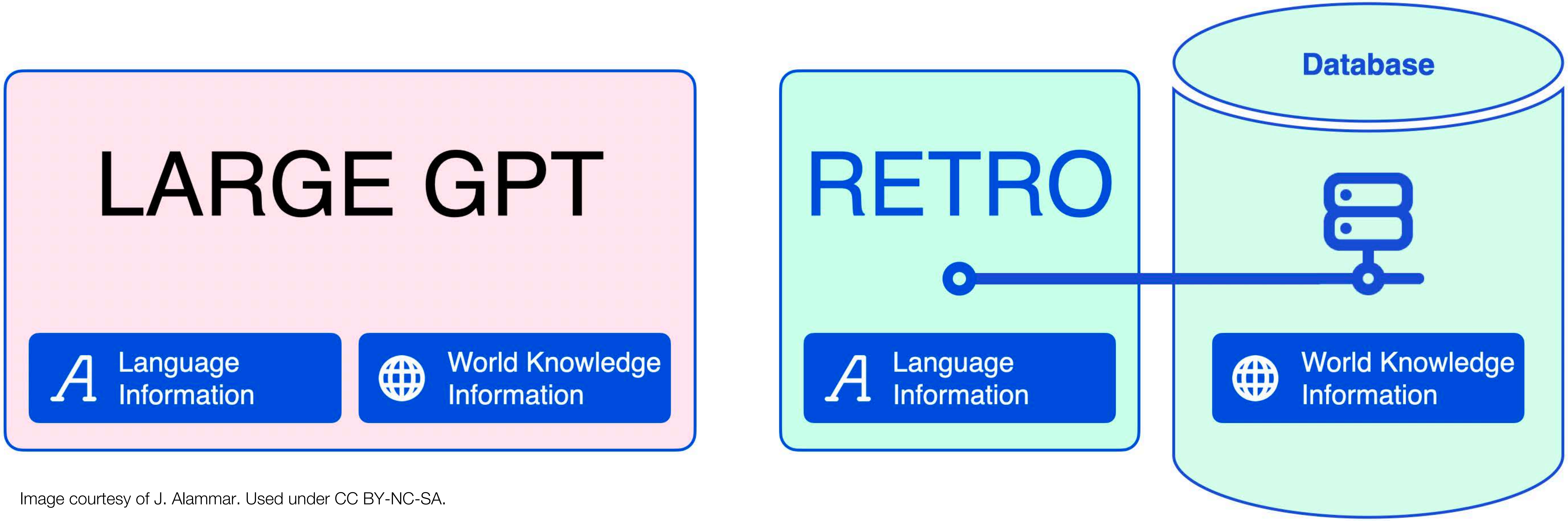
(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

Figure 2: Comparing the full self-attention pattern and the configuration of attention patterns in our Longformer.

Even-larger-context transformers

Retrieval-enhanced

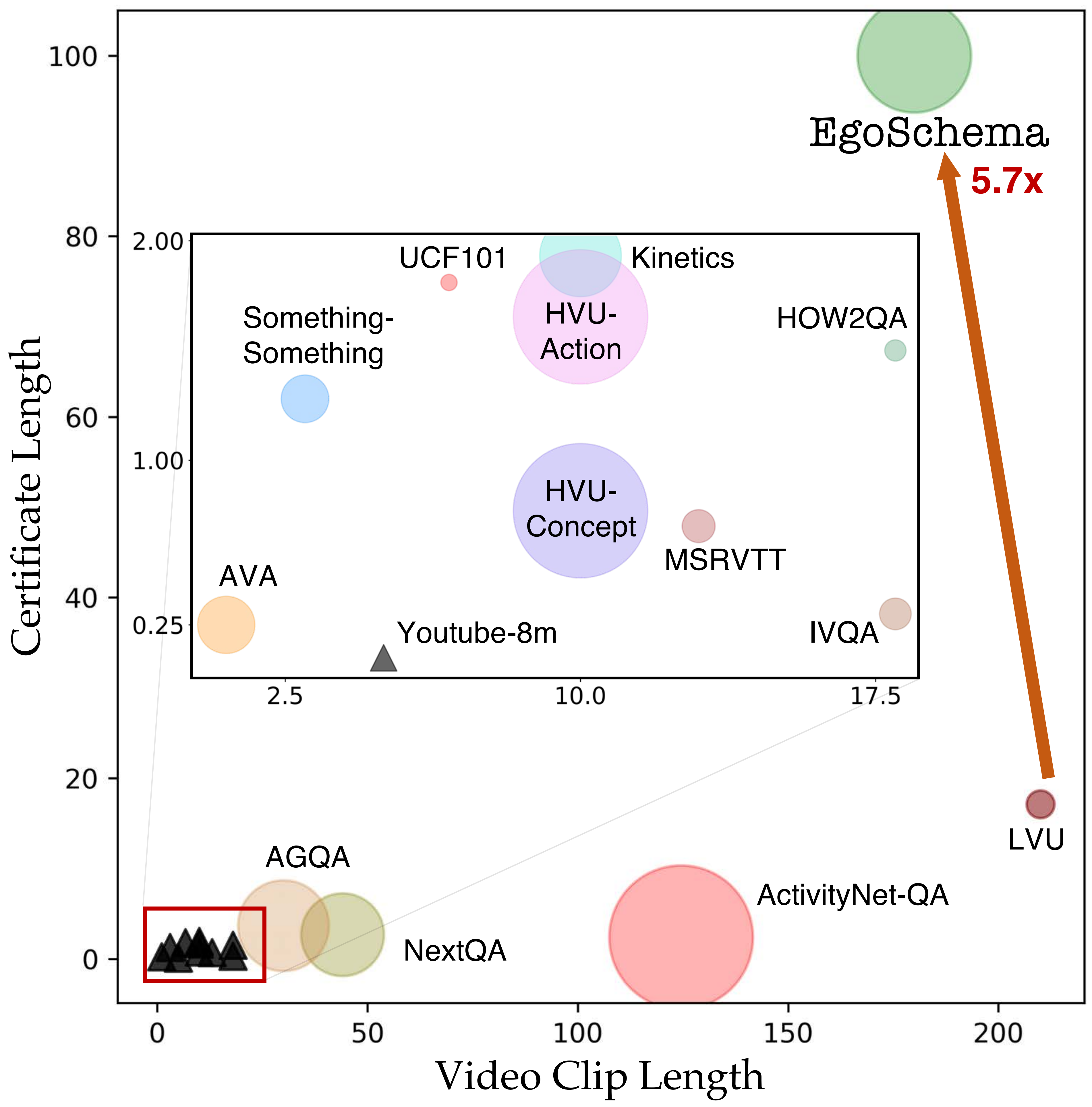
- **RETRO** enables retrieval from trillion-token databases based on local similarity, swapping model parameters for direct lookup (helps separate language modeling from fact lookup) (https://arxiv.org/abs/2112.04426)

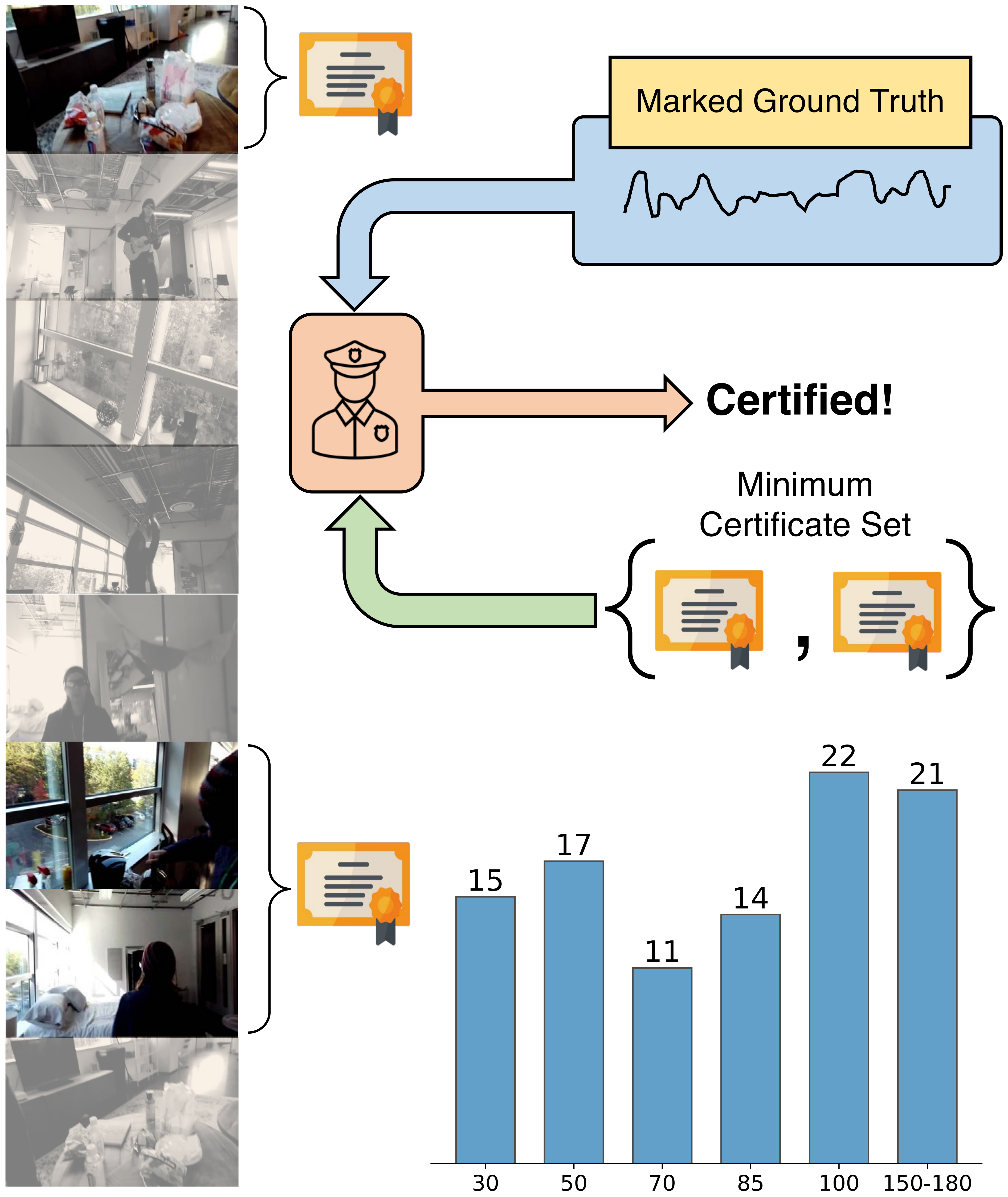


How far back can we go with attention?

- BERT: 512 tokens
- GPT-2: 1024 tokens
- GPT-3: 2048 tokens
- GPT-4: 8,000 tokens, with a souped up 32K token version available
- Anthropic apparently has a model with a 100K token window (about 75K words)

When do we actually need long-term context?





https://arxiv.org/abs/2308.09126

Memory fast and slow

parameters are "slow memory"

activations are "fast memory"

Fast weights? Slow activations?

- **Hypernets** are nets that output weights of another net — these weights are a “fast memory” of the input to the hypernet.
- **Code books** use tensors of activations that are learned (backprop to activations). These activations are “slow memory” of the dataset you are learning.

11. Memory and sequence modeling

- CNNs for sequences
- RNNs
- LSTMs
- Sequence models and long memory

9. Memory and sequence modeling

- CNNs for sequences
- RNNs
- LSTMs
- Sequence models