DDA4220 · Fall 2026

Lecture 10

Deep Learning and Applications

Sequence models: RNNs and LSTMs

Zhen Liu

CUHK-Shenzhen

Lecture outline

01 Sequence tasks and recurrent state

02 Language modeling

03 Backpropagation through time (BPTT)

04 LSTM

05 Using recurrent layers

* denotes optional slides

Sequences

Video frames

Time

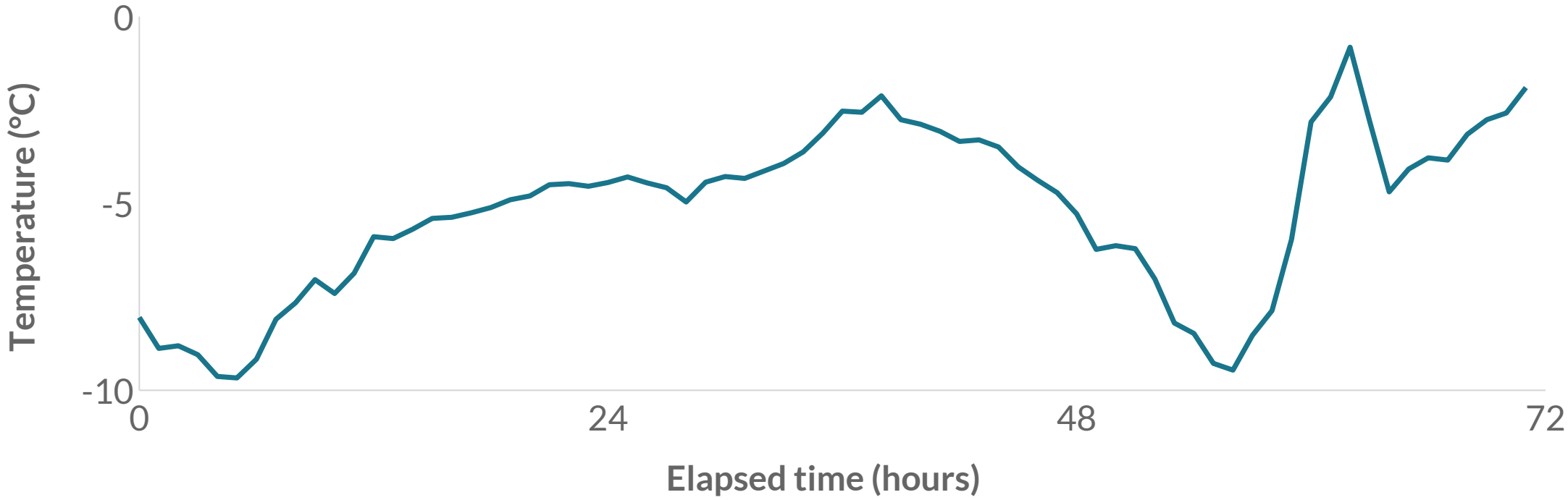
Words in a sentence

The person pours milk into the bowl.

Source: CS7643

Measurements over time

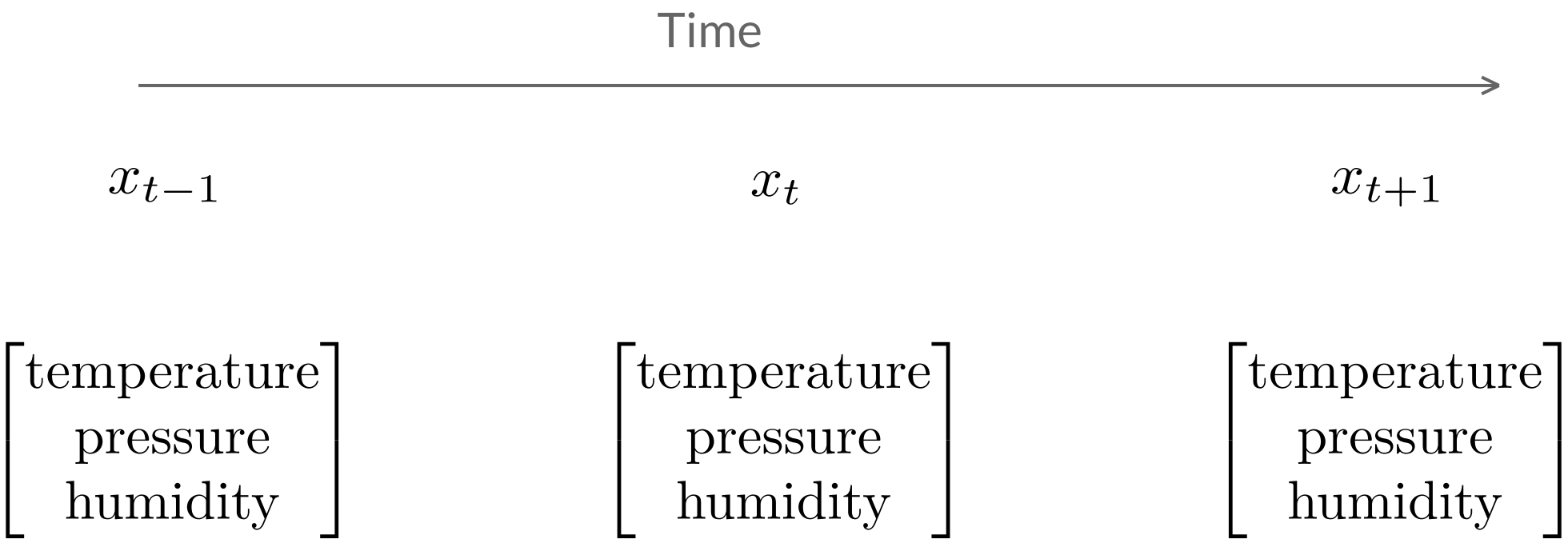
Hourly observations at one weather station



Jena, January 2009

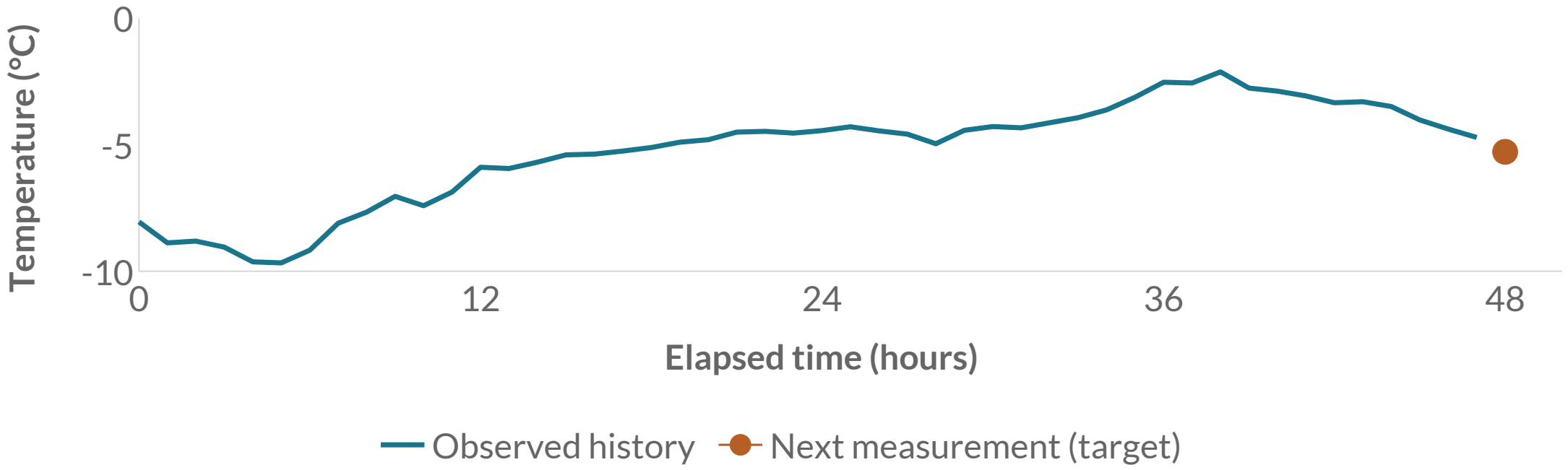
Source: Jena weather data

One time step, several measurements



One column = one observation time

Predicting the next measurement



$x_1, \ldots, x_t$ → $\widehat{T}_{t+1}$

Available measurements

Predicted next-hour temperature

Source: Jena weather data

Many-to-one: video classification

Video frames

Preparing breakfast

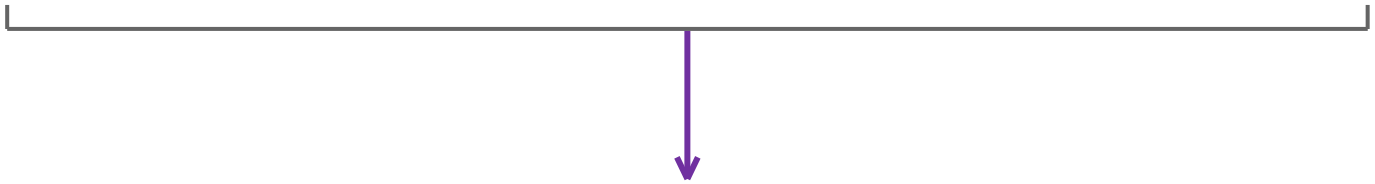
One category for the whole video

Source: CS7643

Many-to-one: sentence classification

Movie review

"I expected to hate it, but I loved it."



Positive

One sentiment label for the whole sentence

Labeling each time step

Aligned many-to-many

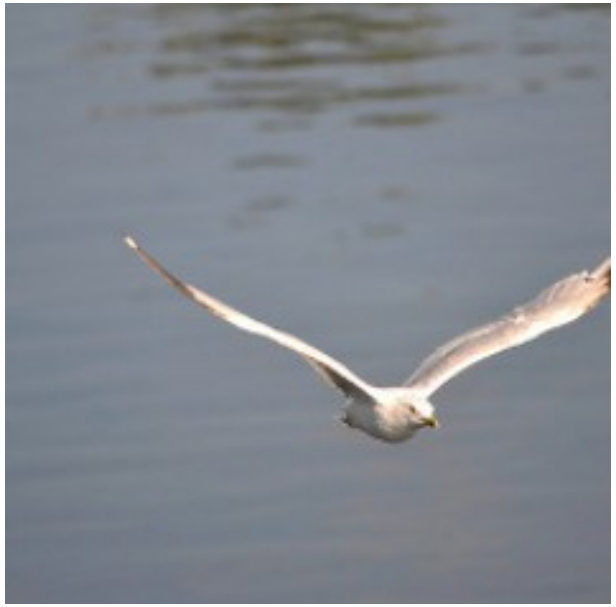
Green: background

Each frame has its own target label

Source: CS7643

One-to-many: image captioning

One image



A sequence of words

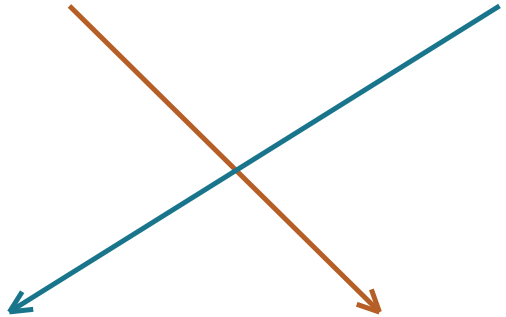
A bird is flying over water.

Source: Show, Attend and Tell (2015)

Many-to-many: machine translation

Unaligned many-to-many

English: The **red** **car** stopped.



French: La **voiture** **rouge** s'est arrêtée.

Output length and word order can differ.

Order matters

The **dog** chased the **cat.**

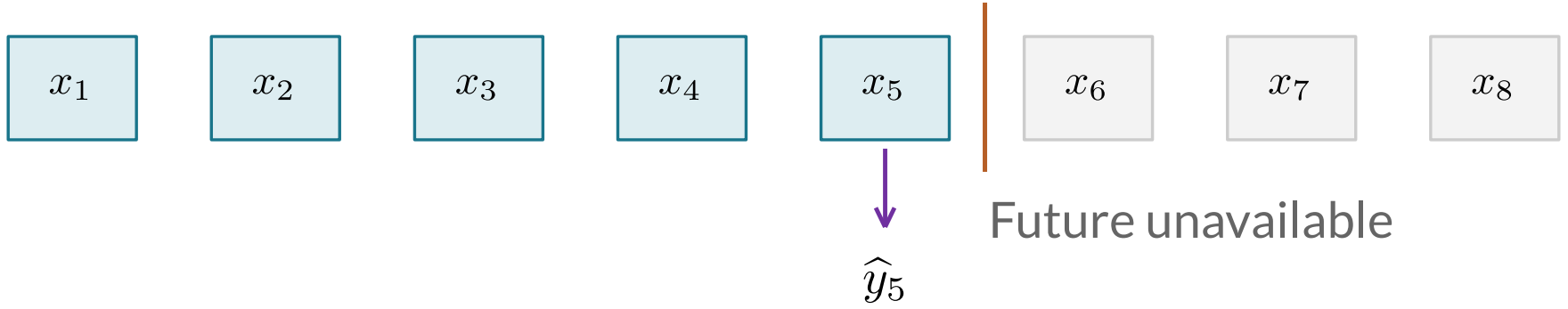
The **cat** chased the **dog.**

The same words can describe different events.

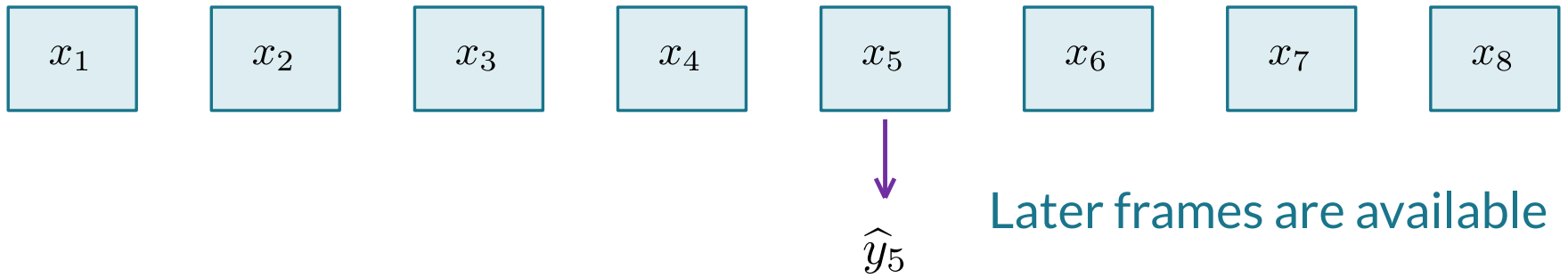
Available context

Label frame 5

Live

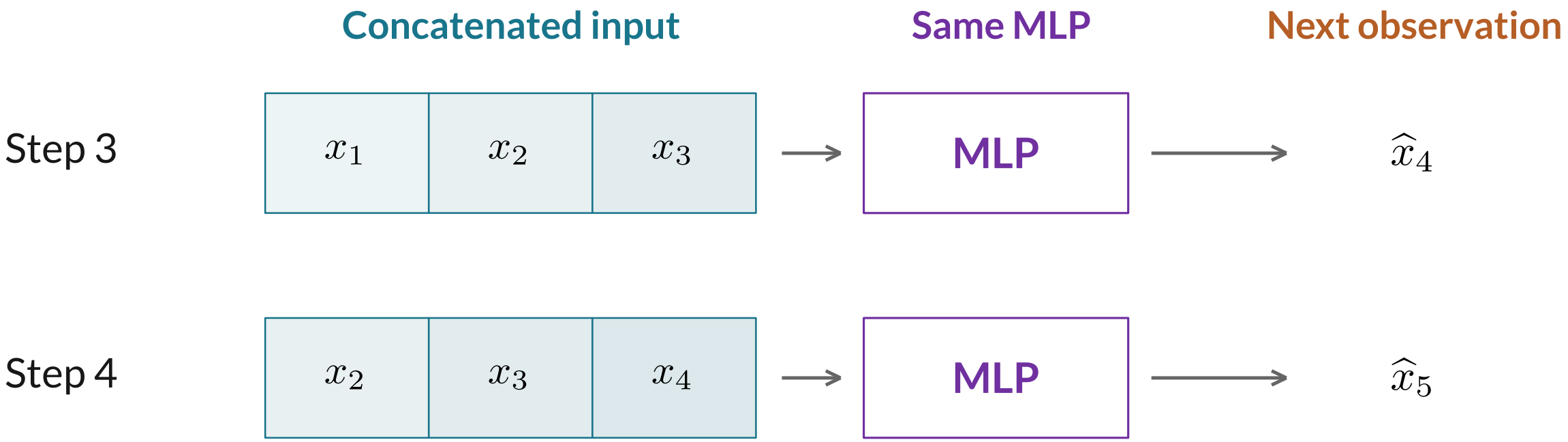


Completed recording



MLP for sequence prediction

Fixed window: use the same number of recent observations.



Slide the window forward and predict again.

Context outside the window

Predict the next word: “is” or “are”

	Last three words	Required word
The **key** beside the windows	in the hall	→ **is**
The **keys** beside the windows	in the hall	→ **are**

An MLP given only this window cannot distinguish the two cases.

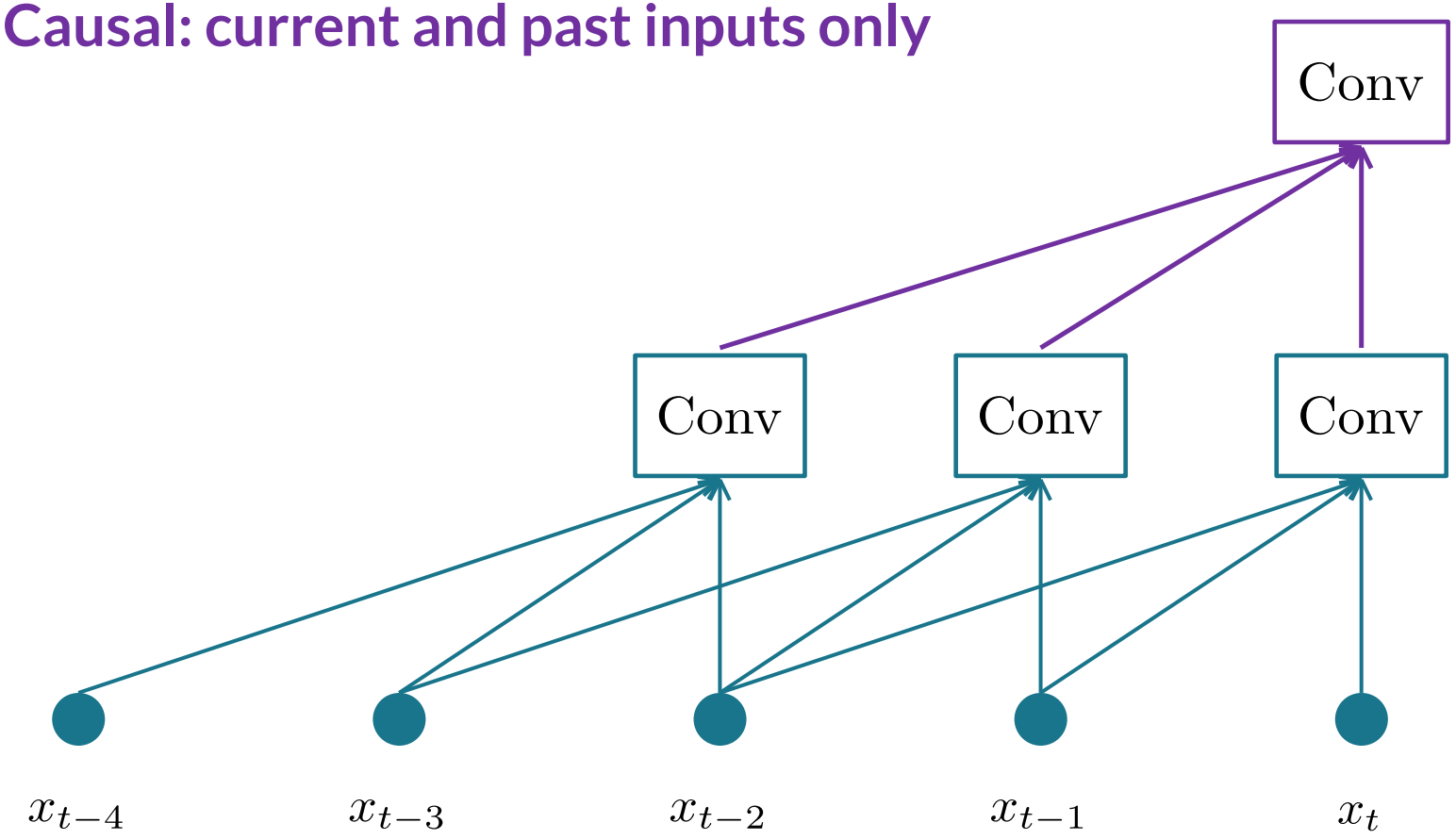
Convolution along time

Causal: current and past inputs only

Second layer

First layer

Observations



At fixed depth, this CNN has a fixed receptive field

Representing the history

Keep useful information about earlier inputs in a fixed-size feature vector.

Observed words

Hidden state

The **keys**

The **keys** beside the windows in the hall

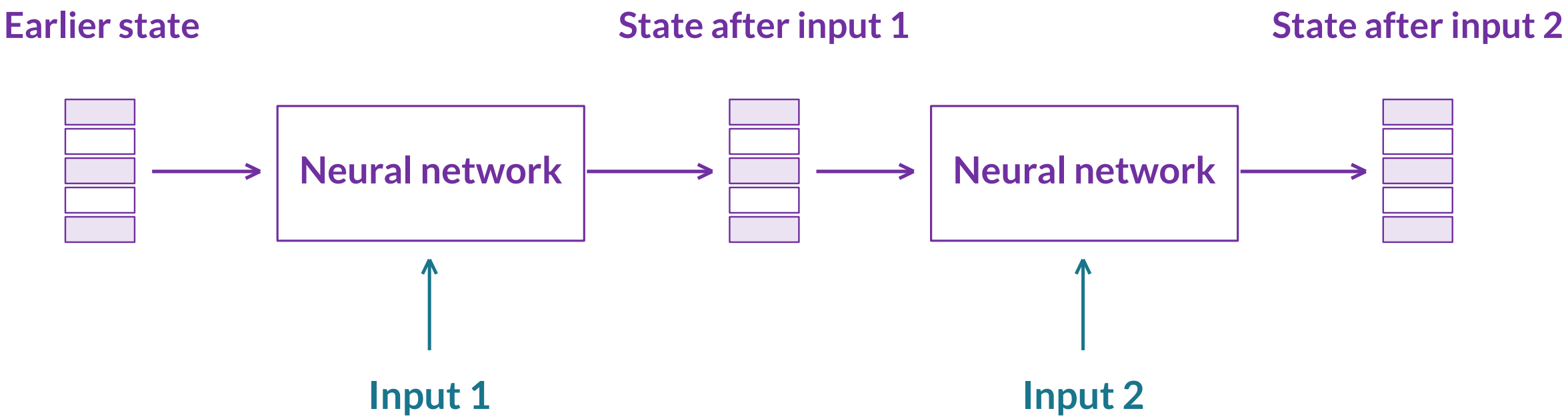
Useful here: a plural subject

Same vector size

Learned from the prediction task. Some details may be lost.

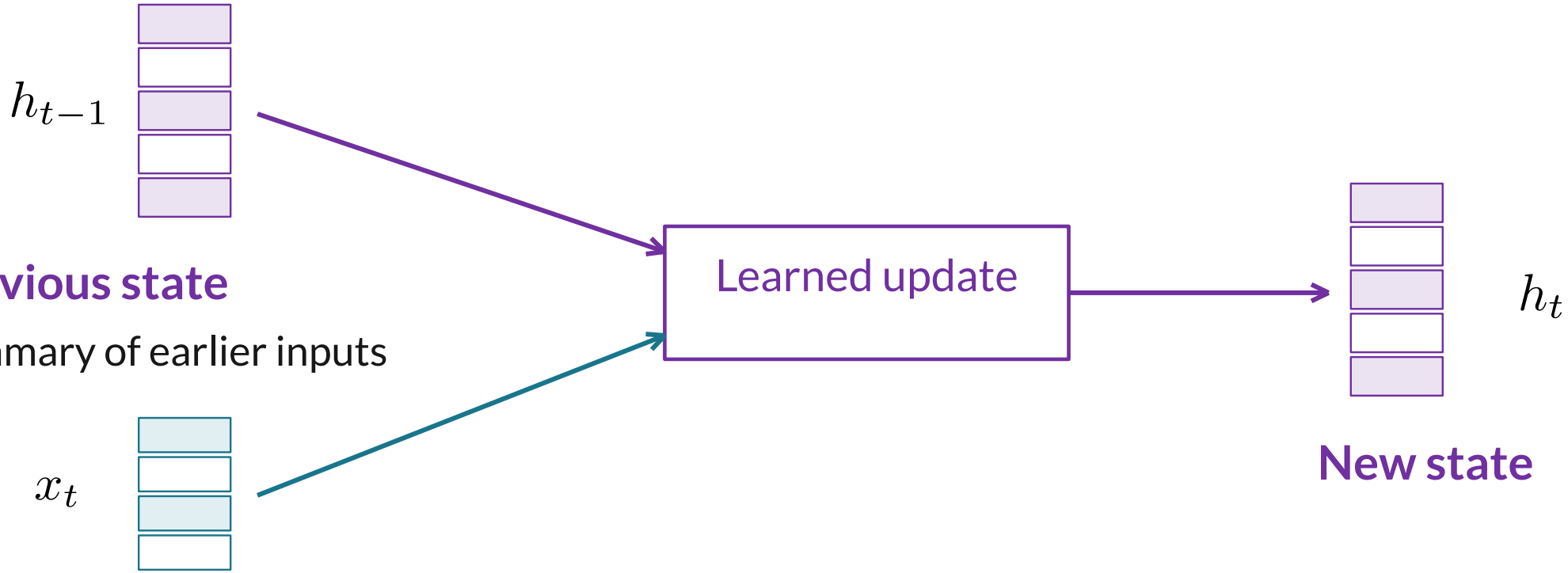
Updating the state

Reuse the previous hidden features when a new observation arrives.



Reuse the same network parameters at every step.

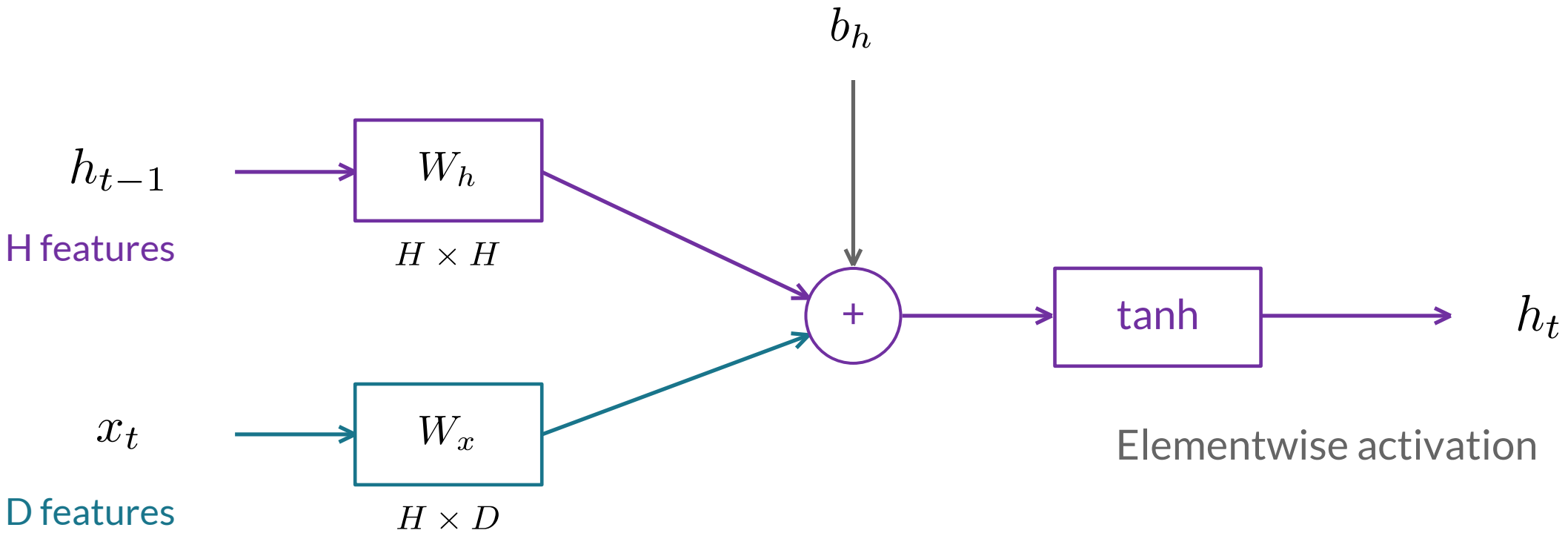
Recurrent neural network (RNN)



$$h_t = f_\theta(h_{t-1}, x_t)$$

Source: Adapted from CS7643

Vanilla RNN cell

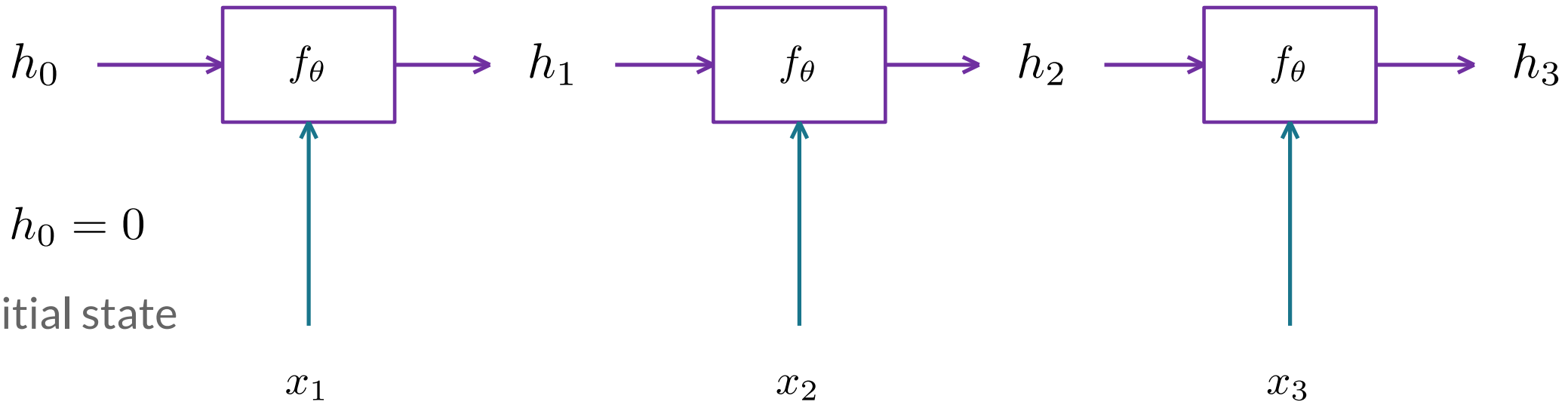


$$h_t = \tanh(W_x x_t + W_h h_{t-1} + b_h)$$

Unrolling through time

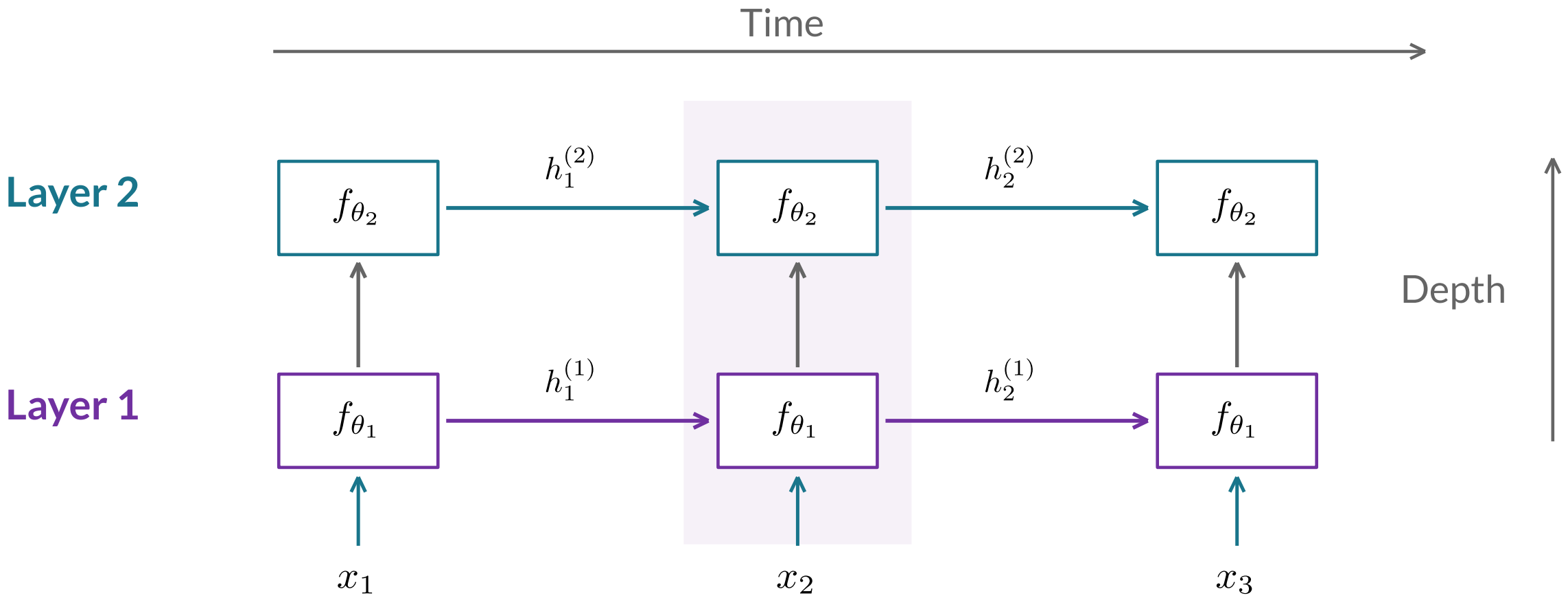
Step 1 Step 2 Step 3

Each new state depends on the preceding state



Source: Adapted from CS7643

Deep RNNs: stacked recurrent layers



Given previous states, a time slice is feedforward

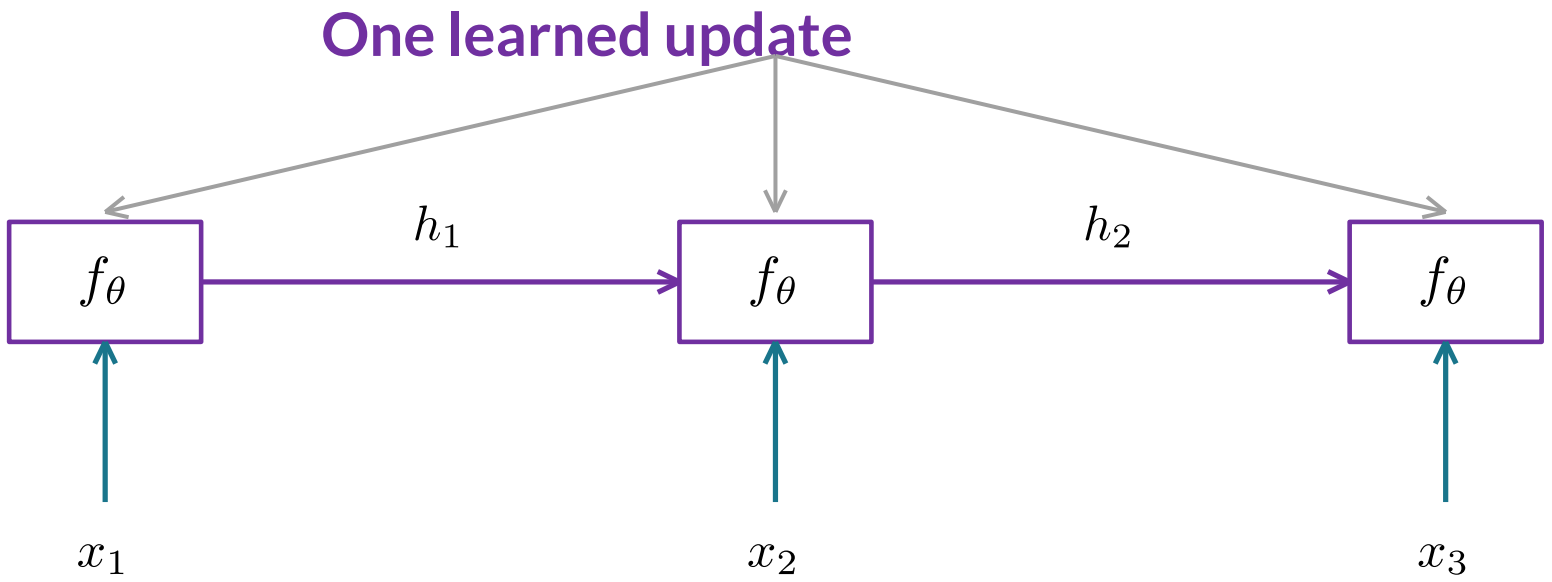
For clarity, we usually draw only one recurrent layer.

Source: Adapted from CMU Deep Learning

Shared parameters across time

One recurrent layer shown

$$\theta = \{W_x, W_h, b_h\}$$

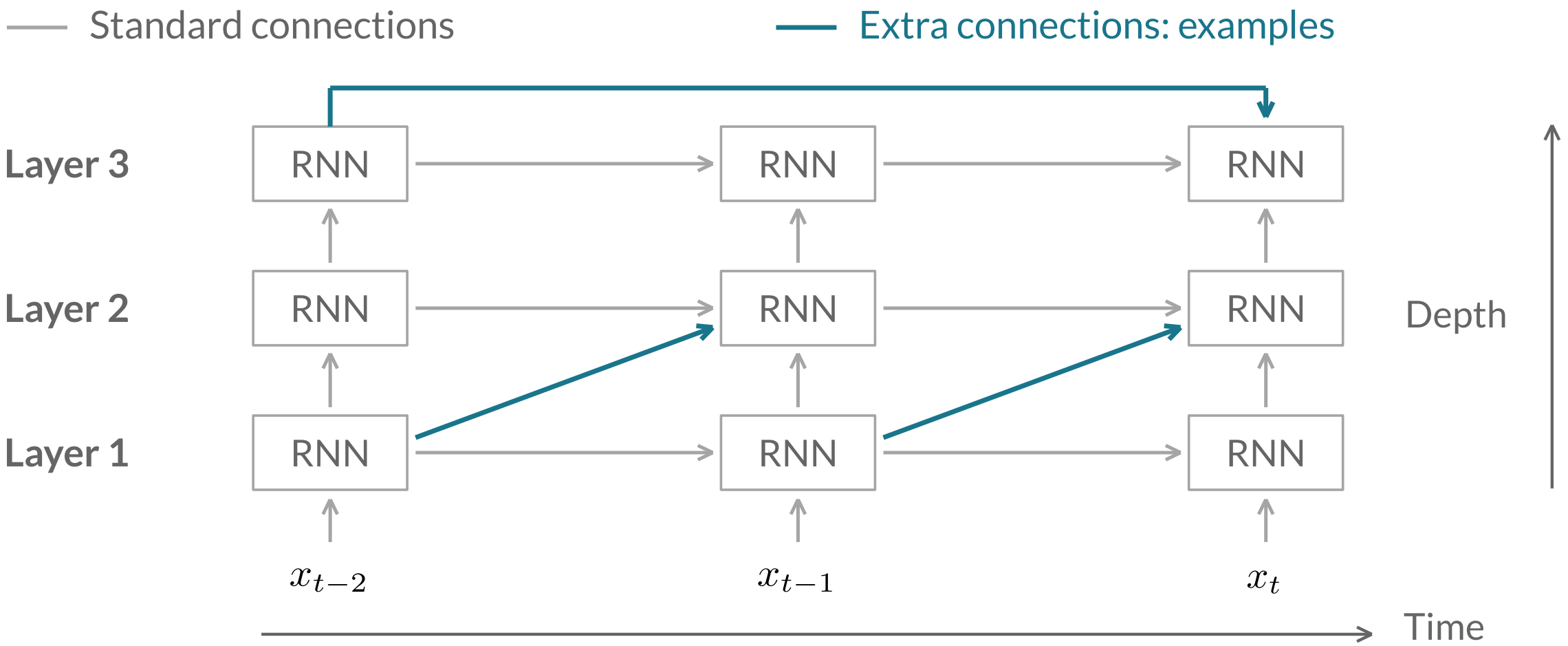


More time steps: more computation

Same parameter set

Source: Adapted from CS7643

Other recurrent topologies



Default: gray connections

Patterns and weights repeat over time

Source: Adapted from CMU Deep Learning

States and parameters

Parameters

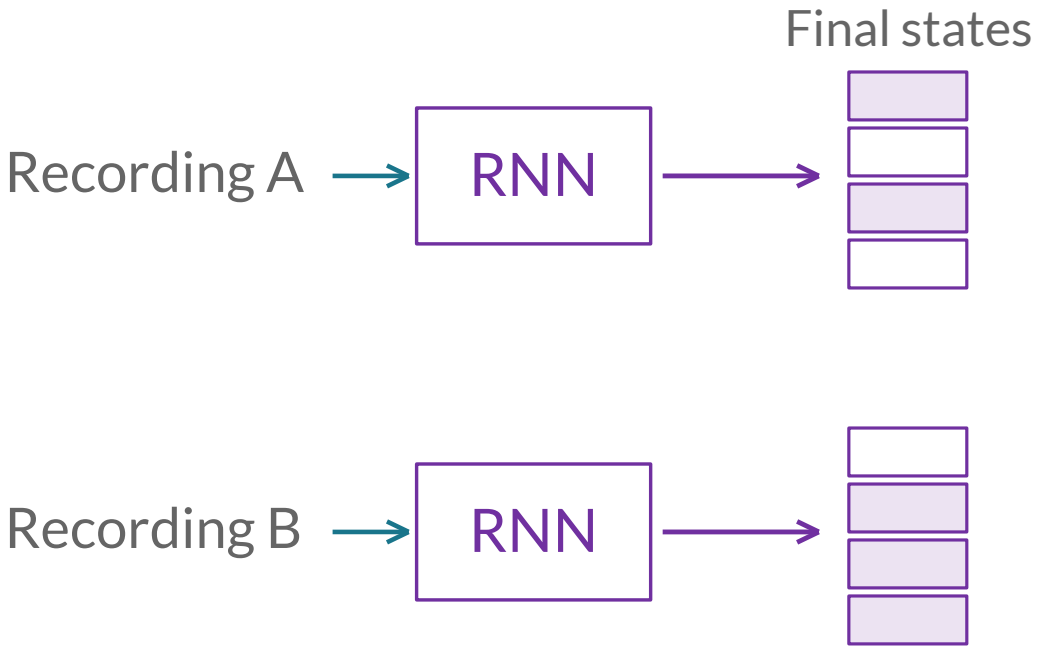
Learned across the dataset

$$\theta = \{W_x, W_h, b_h\}$$

Fixed during inference

States

Computed for each example



Same learned parameters

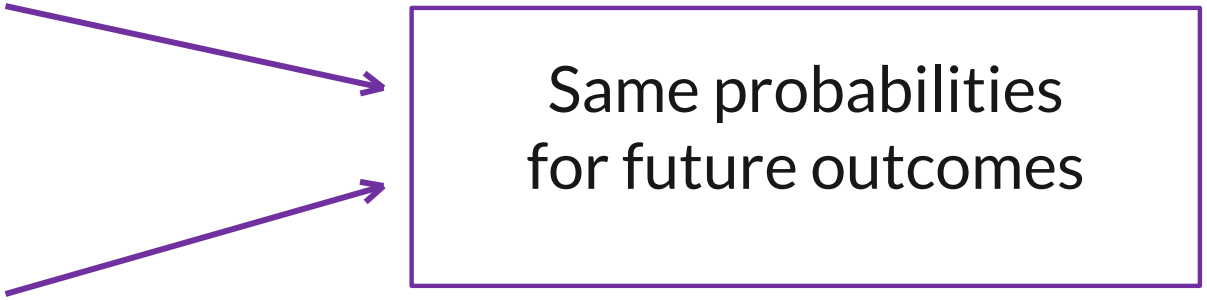
Markovian state

Ideal sufficient state

Preserve all past information relevant to predicting the future

Full history $x_1, \ldots, x_t$

State h_t



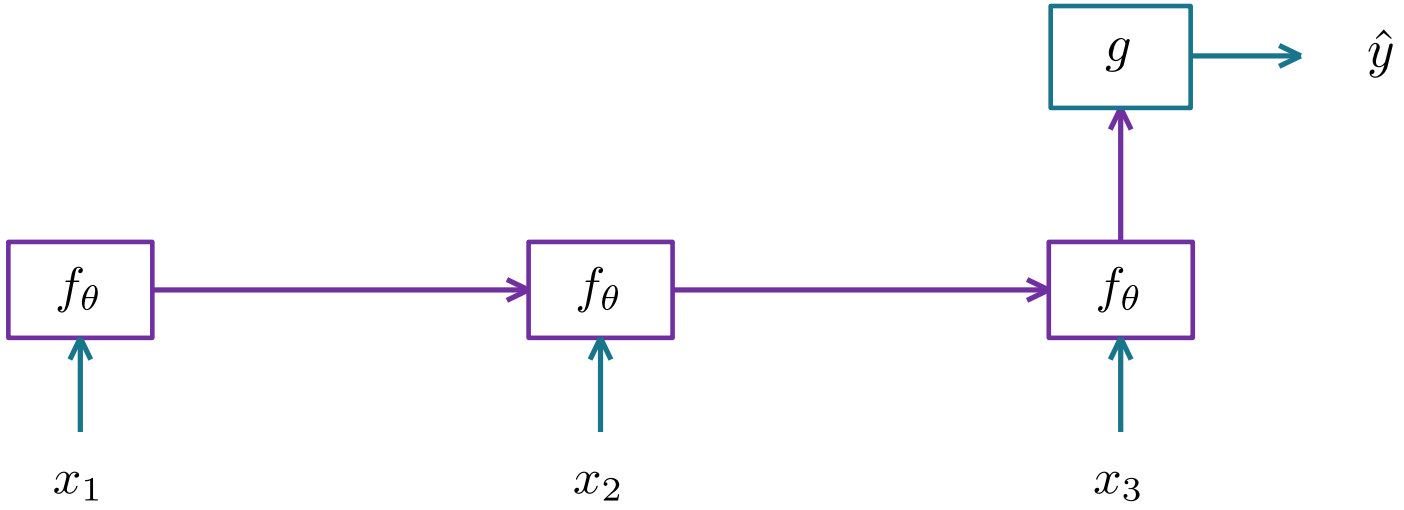
$$P(\text{future} \mid \text{past}) = P(\text{future} \mid \text{state})$$

An RNN aims to learn this state, but can lose relevant information.

Where predictions are made

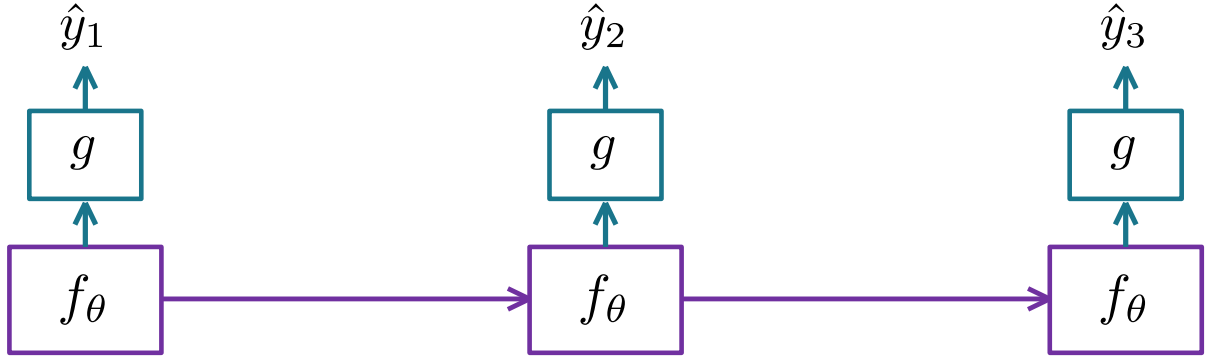
One label for the sequence

g : prediction layer



One label at each step

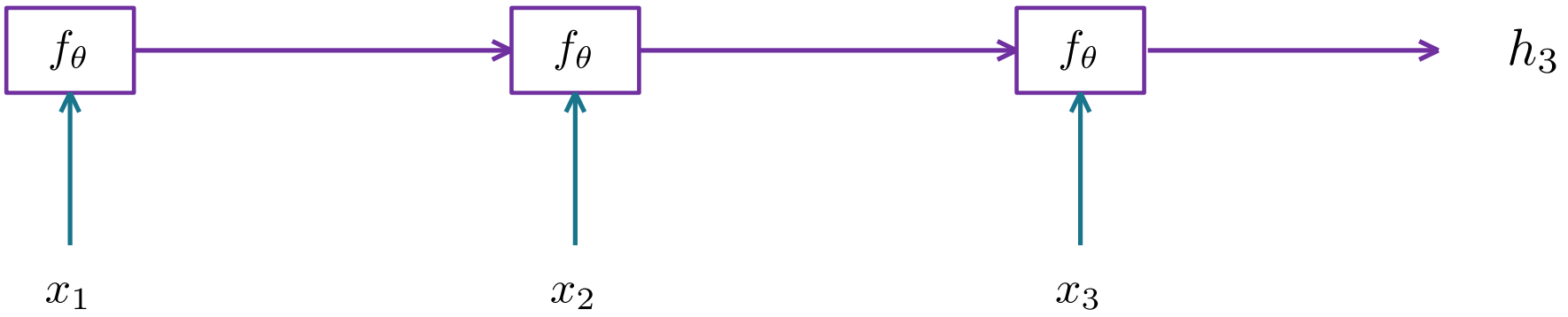
Same inputs as above



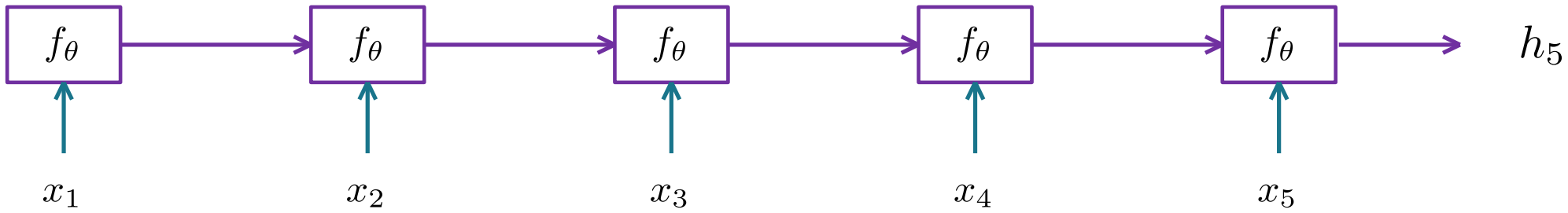
Source: Adapted from CS231n

Variable sequence lengths

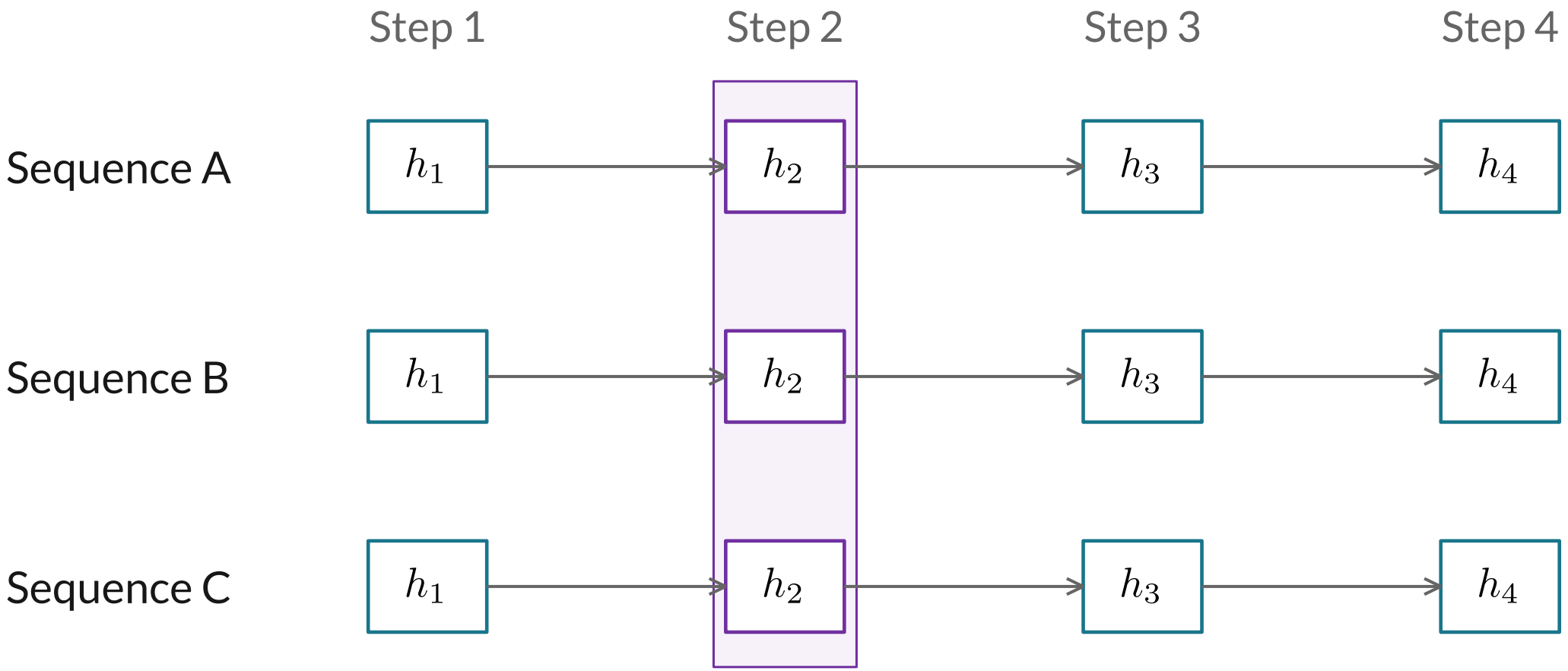
Short sequence



Longer sequence



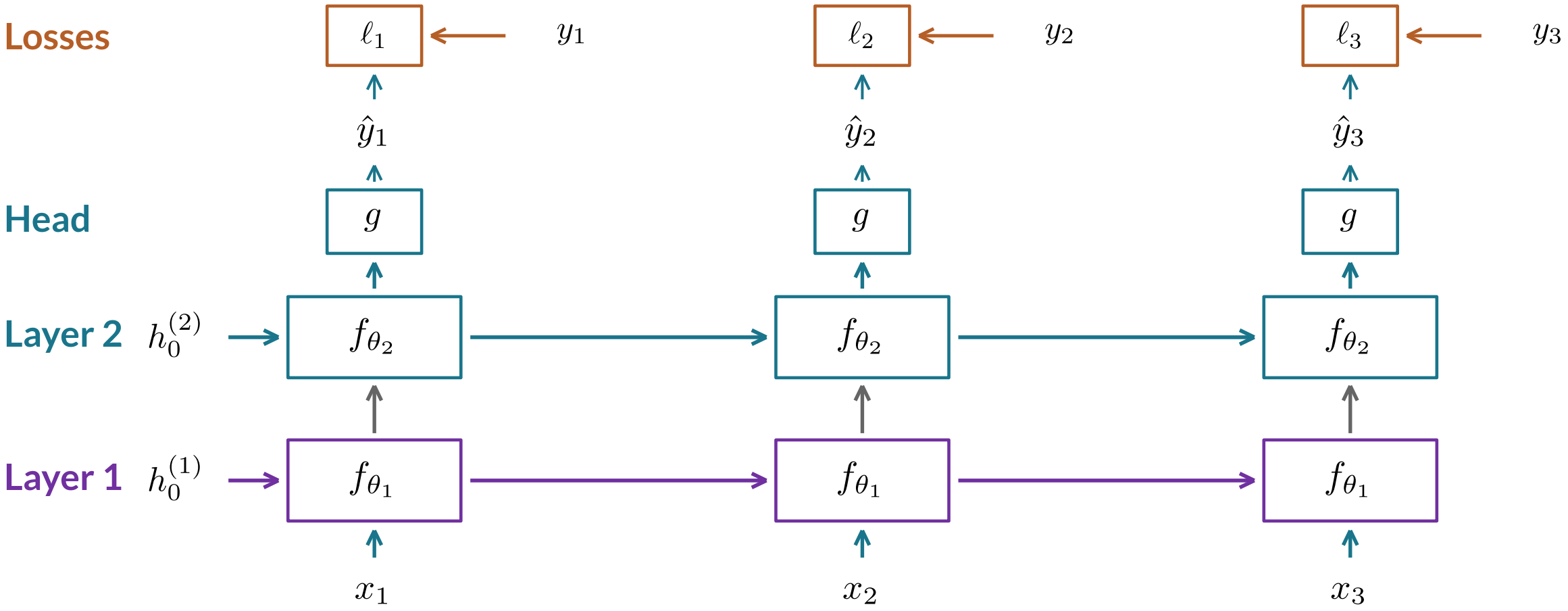
Sequential computation



Batch: compute together

Training a recurrent predictor

$$\mathcal{L} = (\ell_1 + \cdots + \ell_T)/T$$



Source: Adapted from CS7643 · CMU Deep Learning

Lecture outline

01 Sequence tasks and recurrent state

02 Language modeling

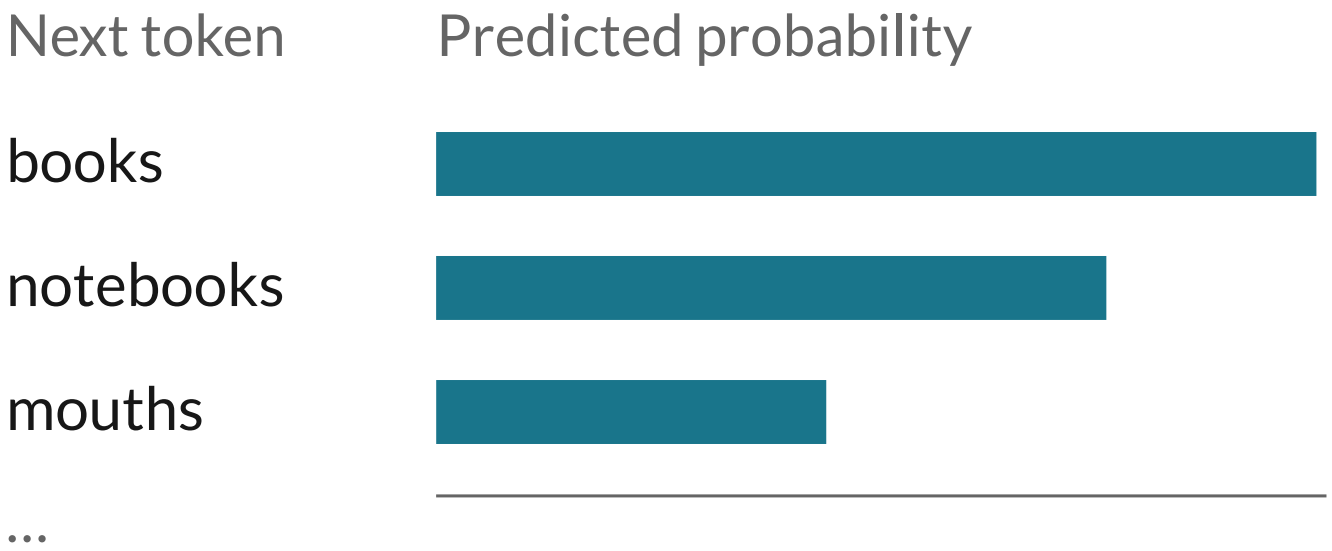
03 Backpropagation through time (BPTT)

04 LSTM

05 Using recurrent layers

Language modeling

The students opened their …

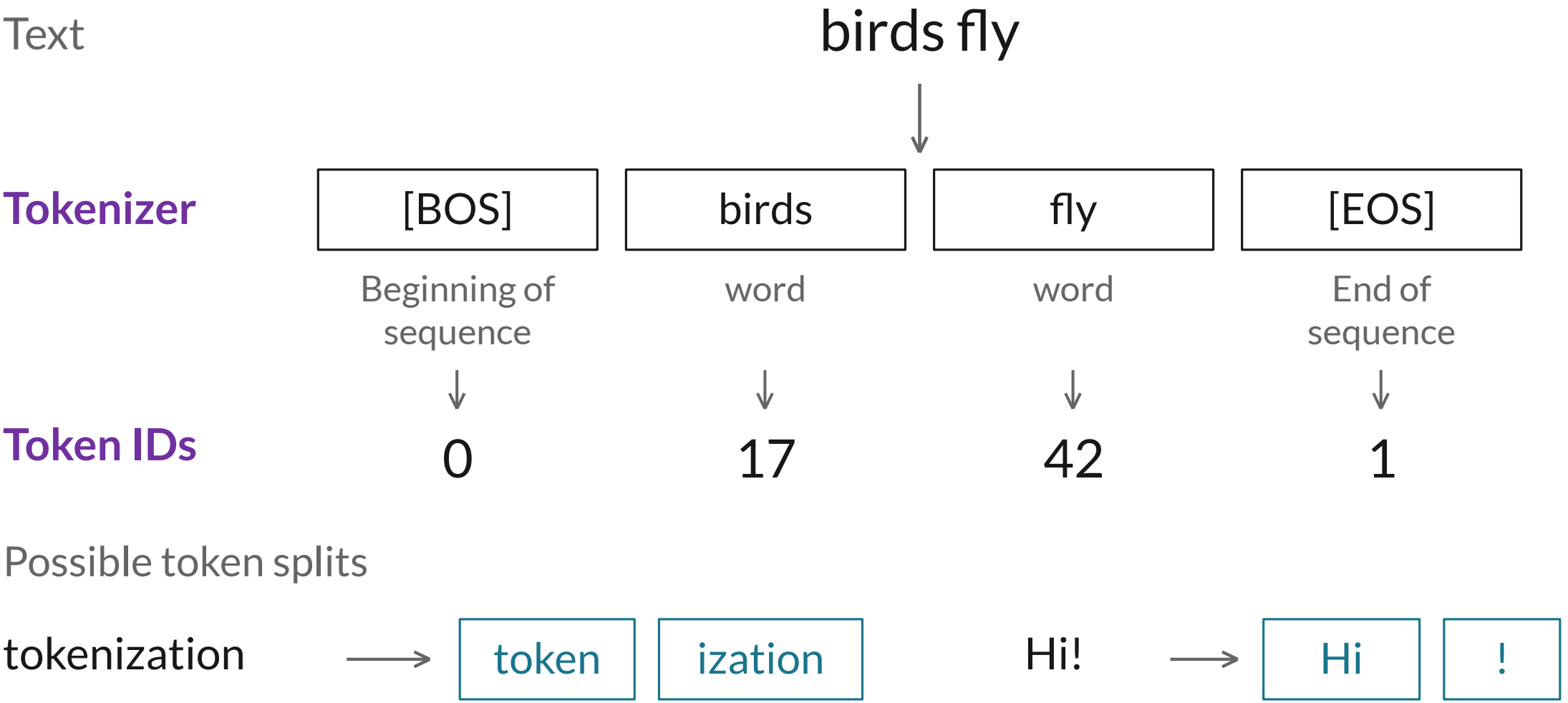


Classification over the vocabulary

Earlier context: an examination, or a choir rehearsal?

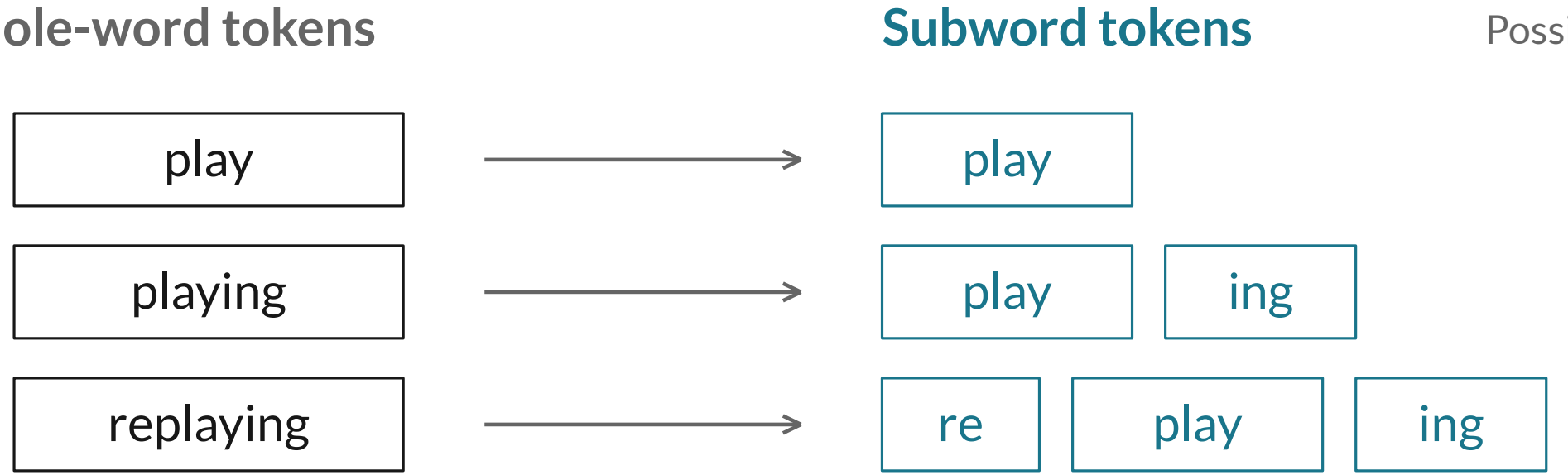
Source: Adapted from Stanford CS224n

Tokens, vocabulary and tokenization



Subword tokenization

Whole-word vocabularies grow with word forms, names and rare words.



A missing word becomes an unknown token.

Known pieces can represent many unseen words.

Smaller vocabulary usually means longer token sequences.

Word embeddings

A compact learned vector for each token

Token ID

birds

$u = 17$

One-hot (vocabulary size V)

$$\begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

Learned projection

Embedding

$$x = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_d \end{bmatrix}$$

Token identity

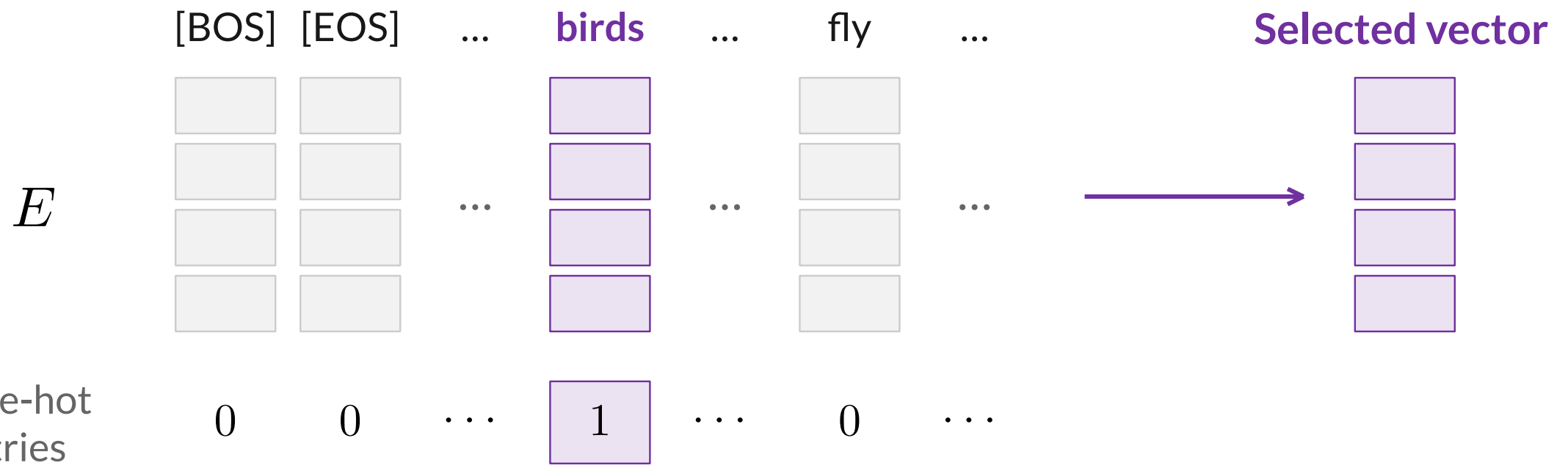
cat–dog and cat–car are equally far apart

d learned features

Typically $d \ll V$

Embedding lookup

One learned vector per token, reused across occurrences

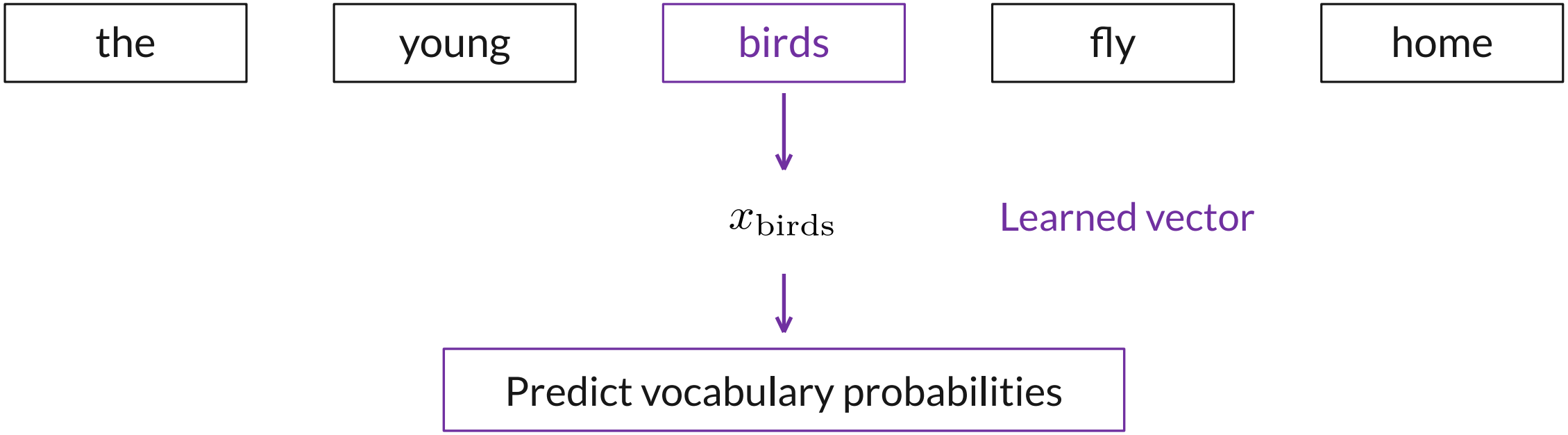


1 selects birds. Every other entry is 0.

$$x = E \ \text{onehot}(u)$$

Word2vec (2013)

Skip-gram: predict surrounding words from the center word



Training targets: the, young, fly, home

Related contexts can produce related vectors.

Source: Adapted from arXiv:1301.3781

Embeddings in PyTorch

```
embedding = nn.Embedding(vocab_size, embed_dim)
ids = torch.tensor([[0, 17, 42]], dtype=torch.long)
x = embedding(ids)  # retrieve learned vectors
```

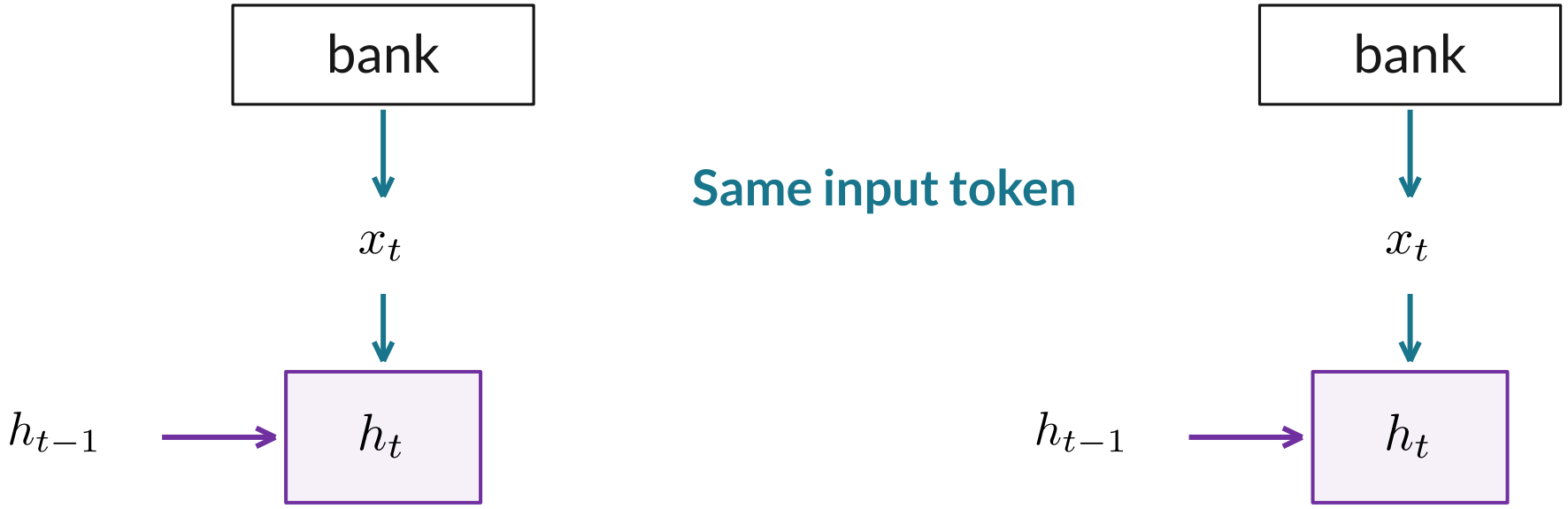
ids: batch × time (integer token IDs)

x: batch × time × embedding dimension

Input and context

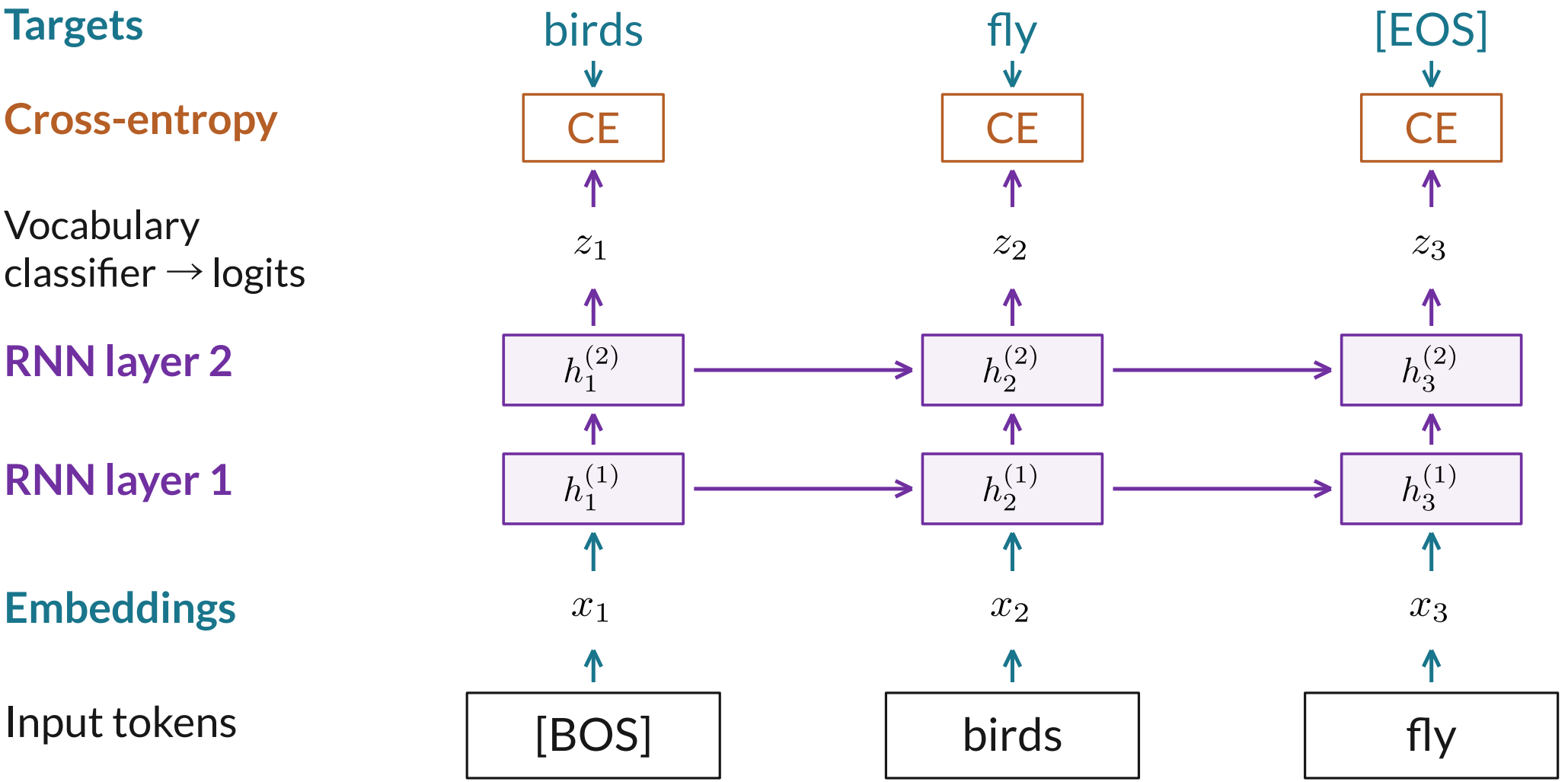
"… beside the river bank"

"… deposit cash at the bank"



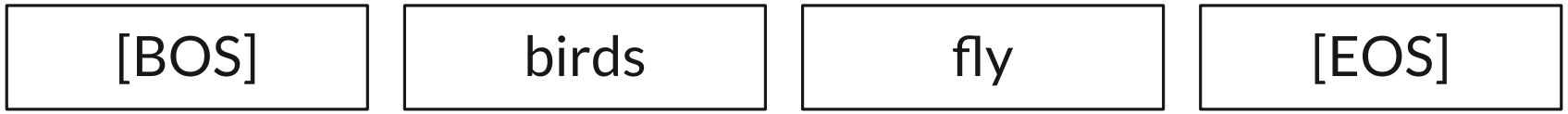
Different context can produce different hidden representations.

The recurrent language model



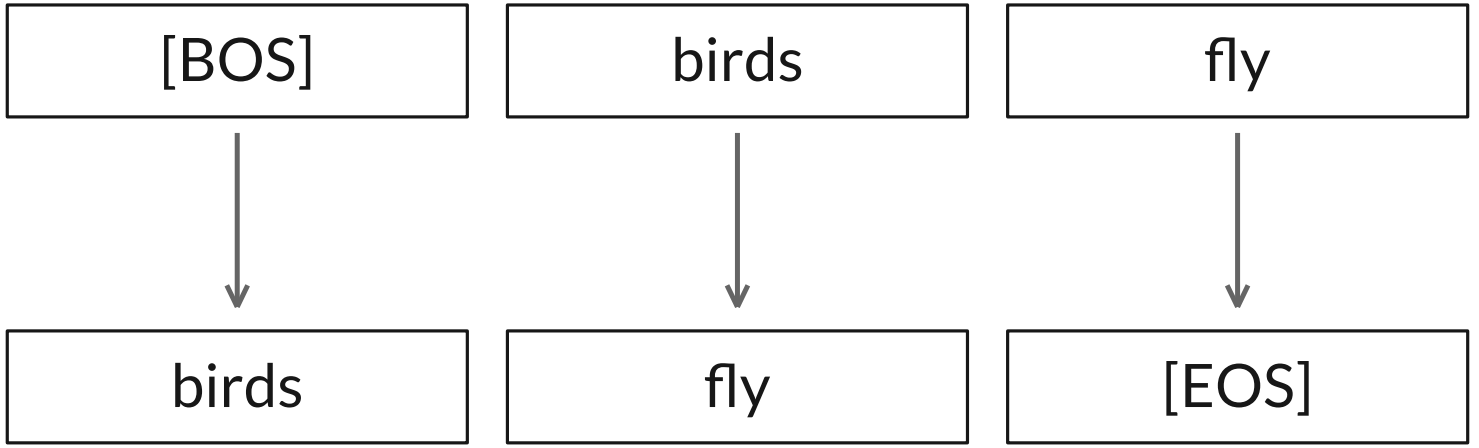
Inputs and targets

Observed sequence



Inputs

Targets



Predict the next token at every position.

Autoregressive language modeling

Autoregressive: predict the next token from the sequence's own past

$$p(\text{birds}, \text{fly}, [\text{EOS}] \mid [\text{BOS}])$$

$$= p(\text{birds} \mid [\text{BOS}])$$
$$\times\ p(\text{fly} \mid [\text{BOS}], \text{birds})$$
$$\times\ p([\text{EOS}] \mid [\text{BOS}], \text{birds}, \text{fly})$$

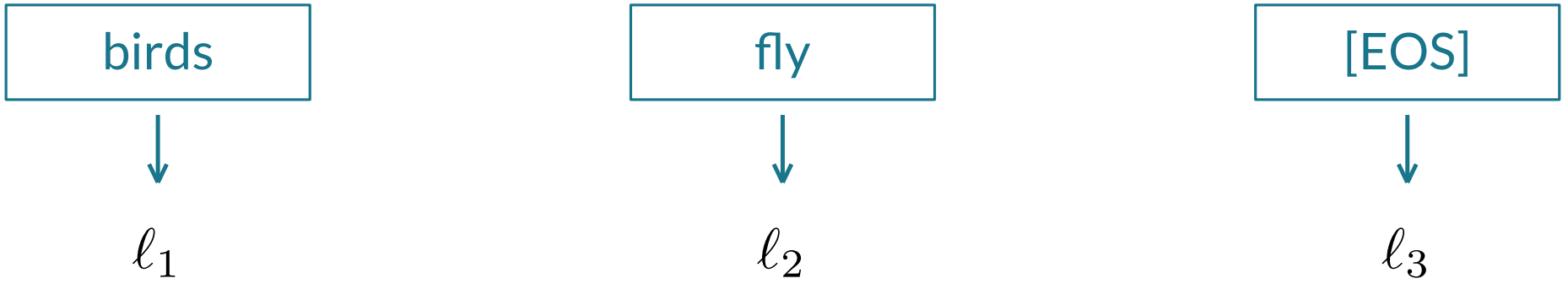
Token sequence:

$$u_0 = [\text{BOS}], \quad u_{1:T} = (u_1, \ldots, u_T), \quad u_T = [\text{EOS}]$$

$$p_\theta(u_{1:T} \mid u_0) = \prod_{t=1}^{T} p_\theta(u_t \mid u_{0:t-1})$$

Multiply the next-token probabilities

Cross-entropy across time



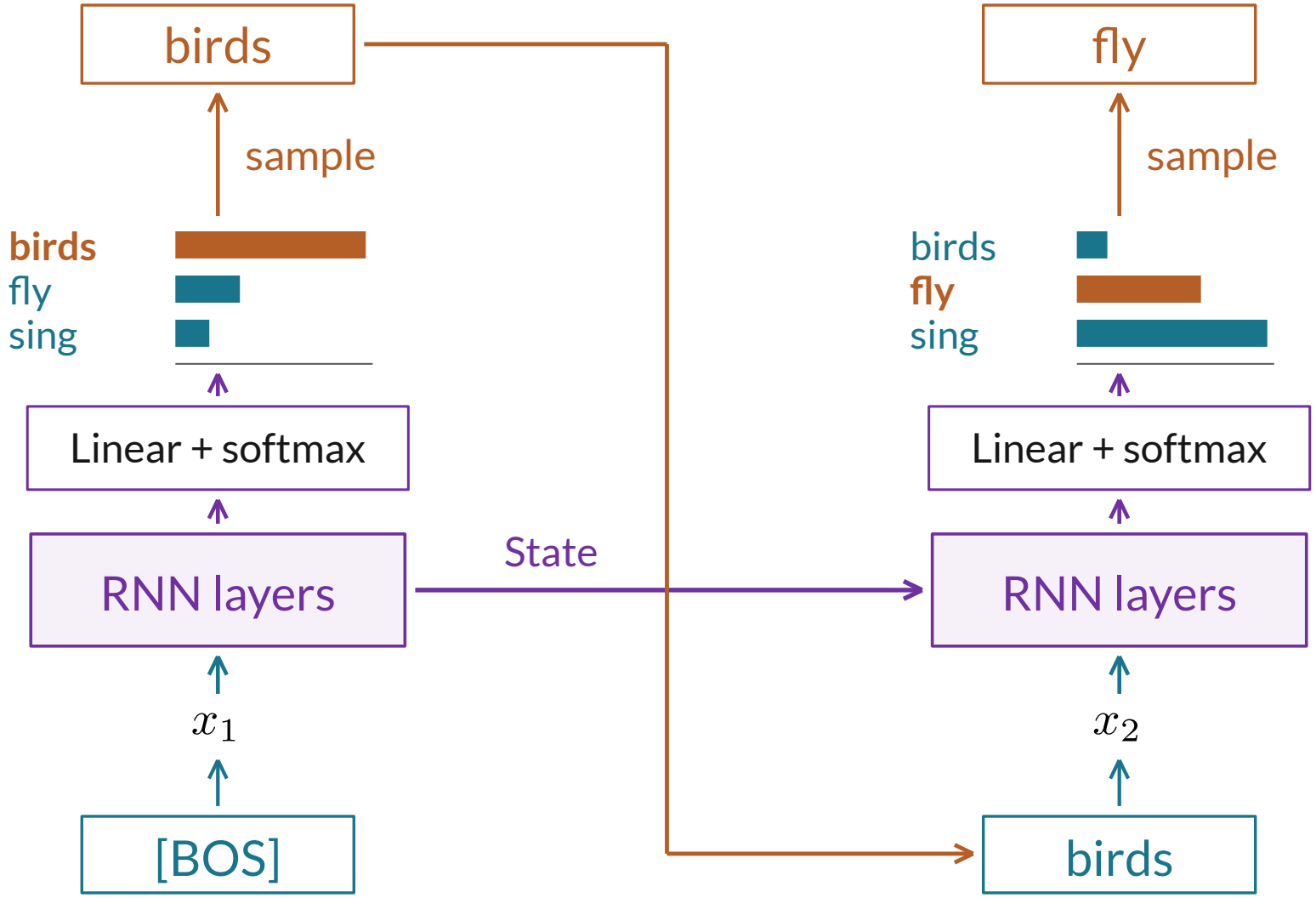
Loss for the observed token

$$\ell_t = -\log p_\theta(u_t \mid u_{0:t-1})$$

Mean over valid target positions

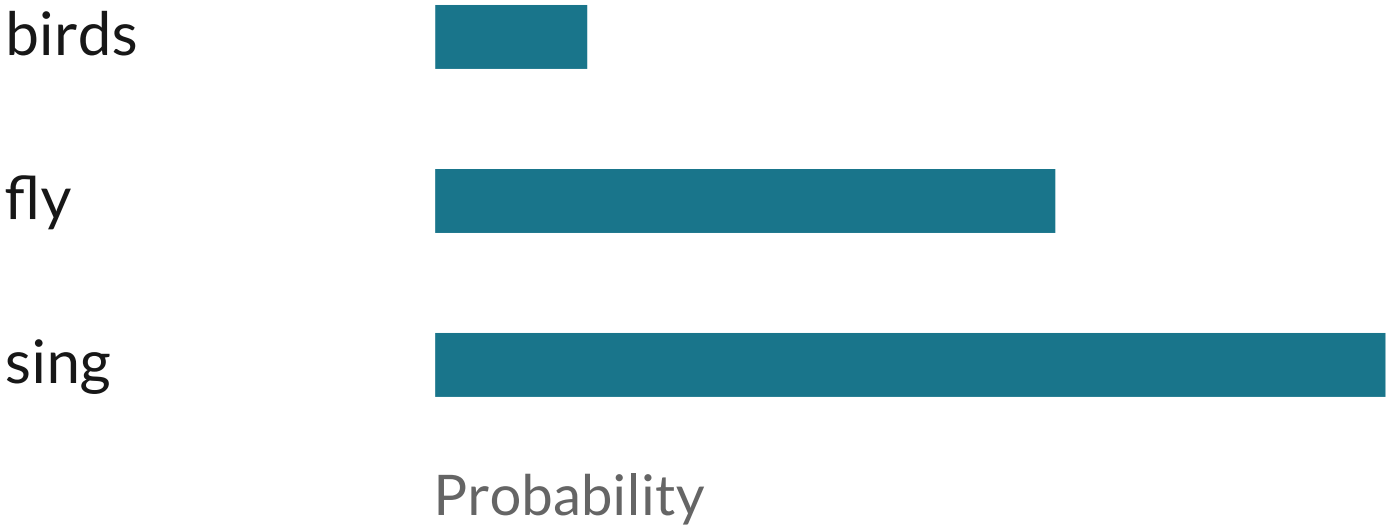
$$\mathcal{L} = \frac{1}{T}\sum_{t=1}^{T} \ell_t$$

Autoregressive generation



Choosing the next token

Prefix: birds



Greedy

Choose the largest probability

Sampling

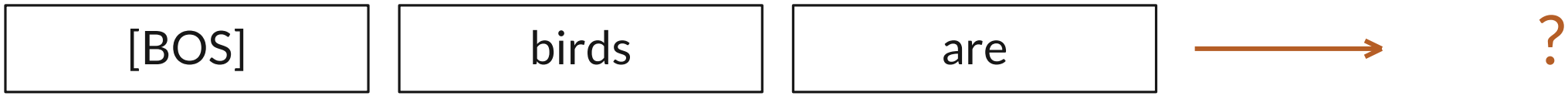
Draw according to the distribution

The selection changes the prefix used at the next step.

Generated prefixes during training

Recorded prefix

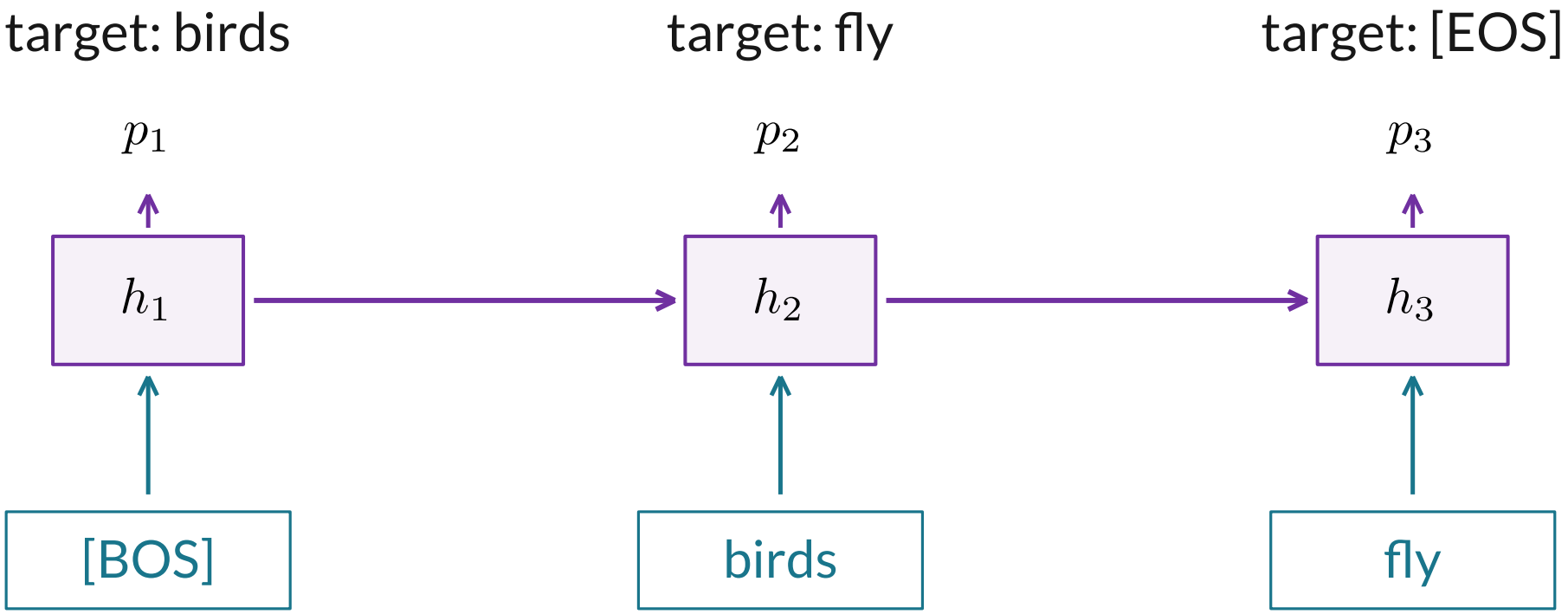
If we feed generated tokens



After “birds are”, [EOS] would end the sentence too early.

Teacher forcing

Train on observed prefixes



Training uses observed tokens, even after an incorrect prediction.
During generation, a wrong token becomes a later input.

Source: Adapted from Stanford CS224n

Lecture outline

01 Sequence tasks and recurrent state

02 Language modeling

03 Backpropagation through time (BPTT)

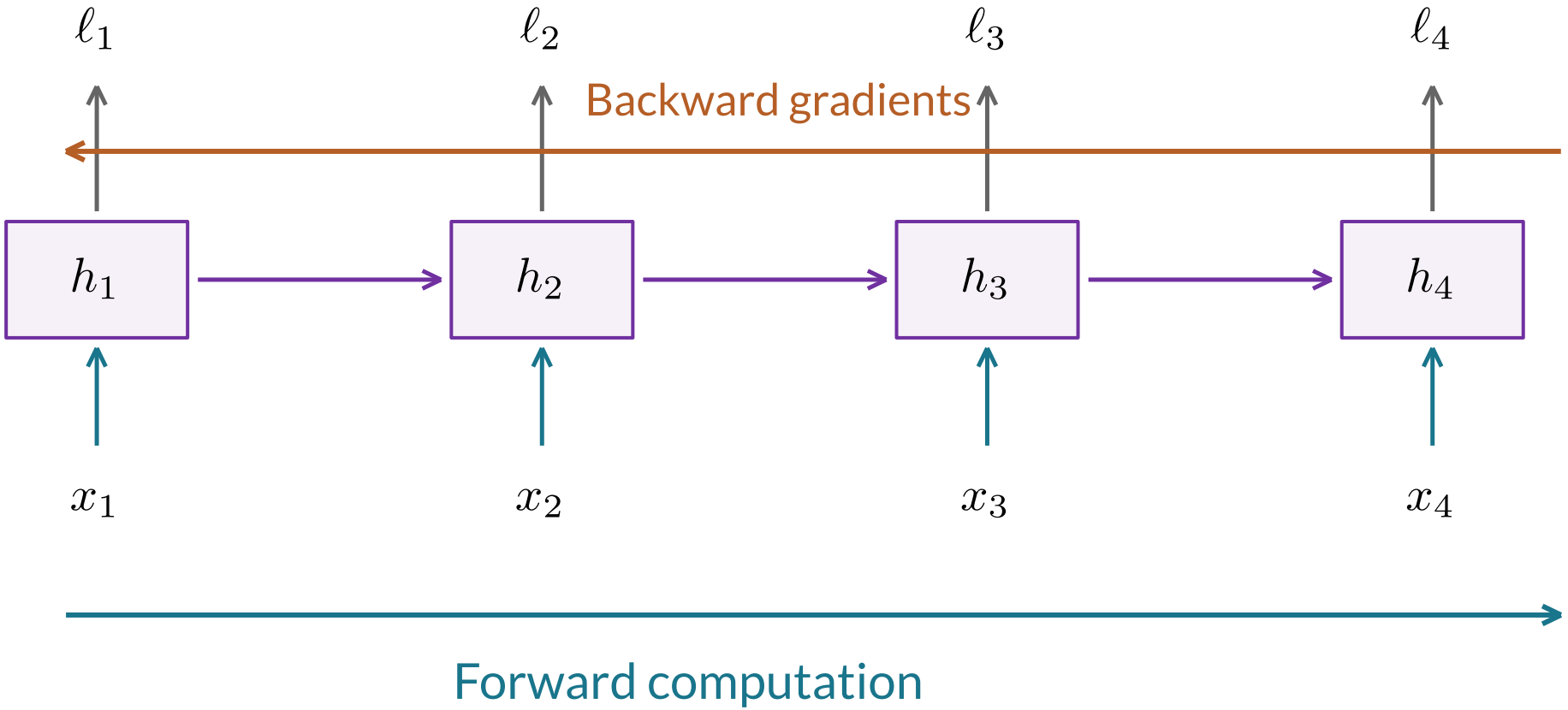
04 LSTM

05 Using recurrent layers

Backpropagation through time (BPTT)

Summed step losses

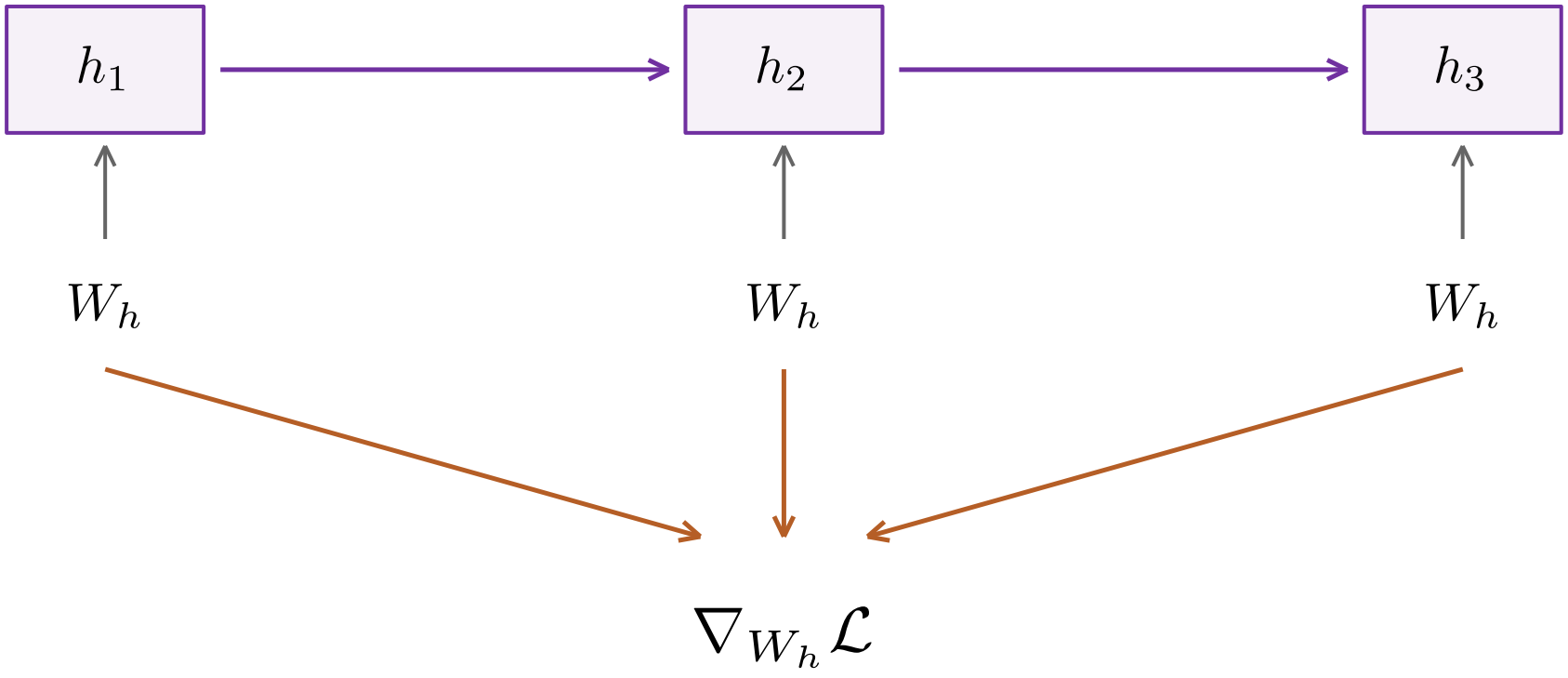
$$\mathcal{L} = \sum_t \ell_t$$



Source: Adapted from Georgia Tech CS 7643

Gradients of shared parameters

Same parameter tensor at every step



Add the contributions from every use.

Source: Adapted from Berkeley CS 182

Local and future losses

From current loss

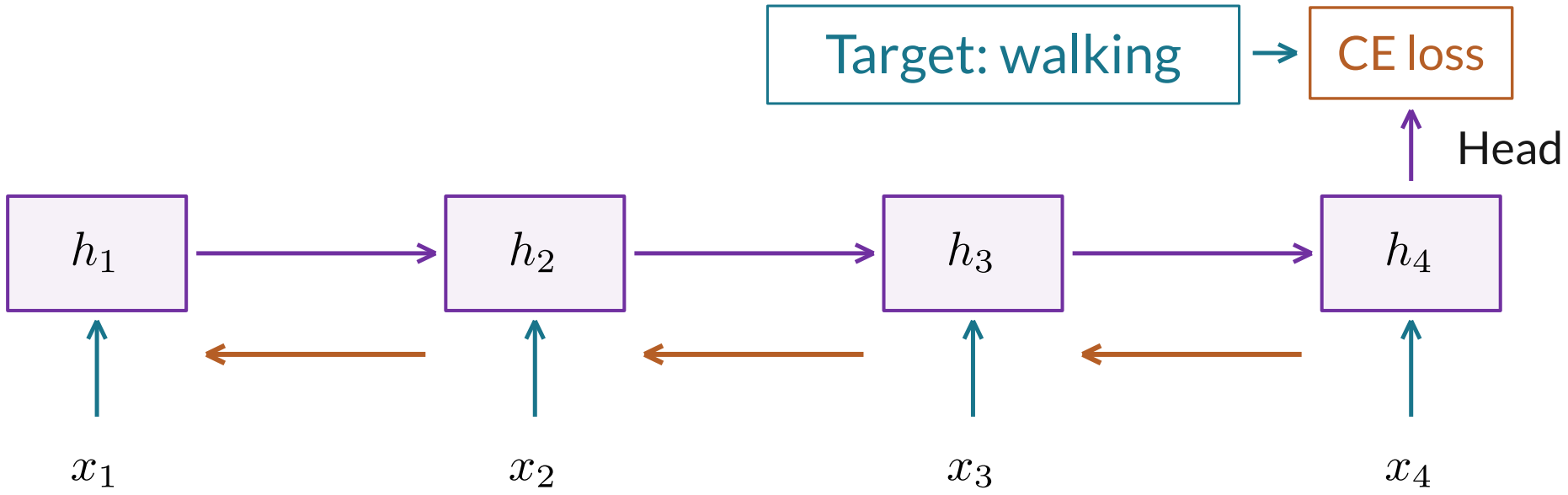
Future losses

$$\nabla_{h_t} \mathcal{L} = \nabla_{h_t} \ell_t + \left(\frac{\partial h_{t+1}}{\partial h_t} \right)^T \nabla_{h_{t+1}} \mathcal{L}$$

Total gradient | Current loss | Future losses through next state

Learning from a final prediction

One label for the recording



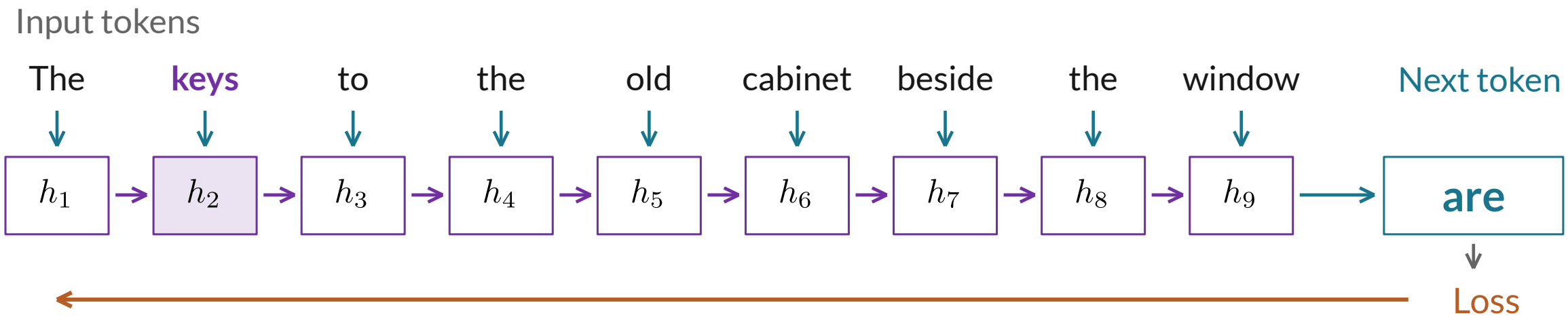
The final loss sends gradients through earlier state updates.

Long-term dependencies

The **keys** to the old cabinet beside the window

are missing.

Plural subject → plural verb

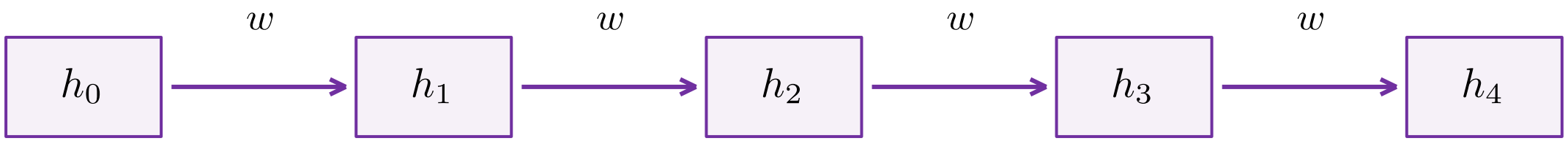


Learning from this prediction reaches back to “keys”.

Sensitivity through repeated updates

Scalar linear example

$$h_t = w\, h_{t-1}$$

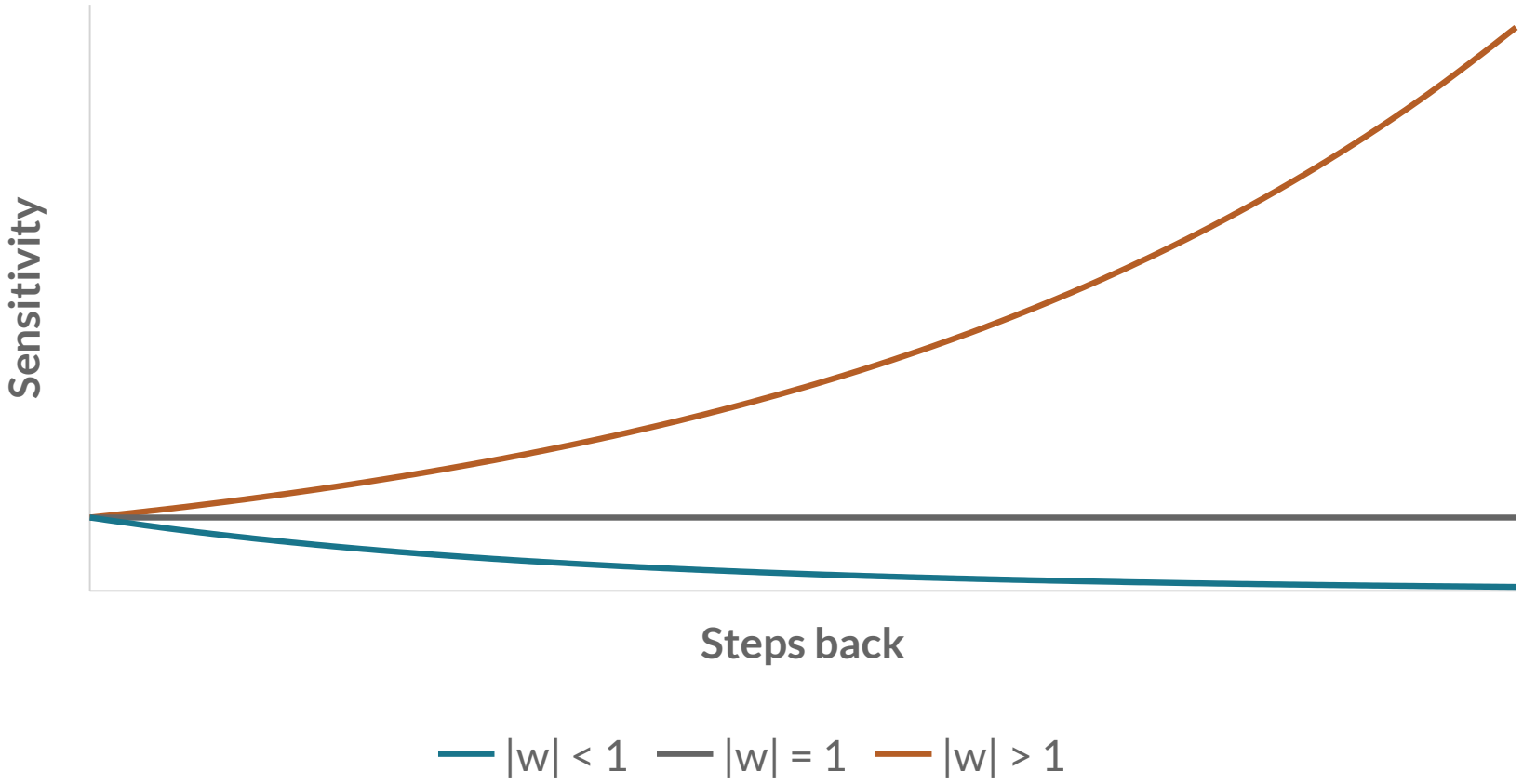


Sensitivity to an earlier state

$$\frac{\partial h_t}{\partial h_{t-k}} = w^k$$

k recurrent steps multiply k copies of w

Vanishing and exploding gradients

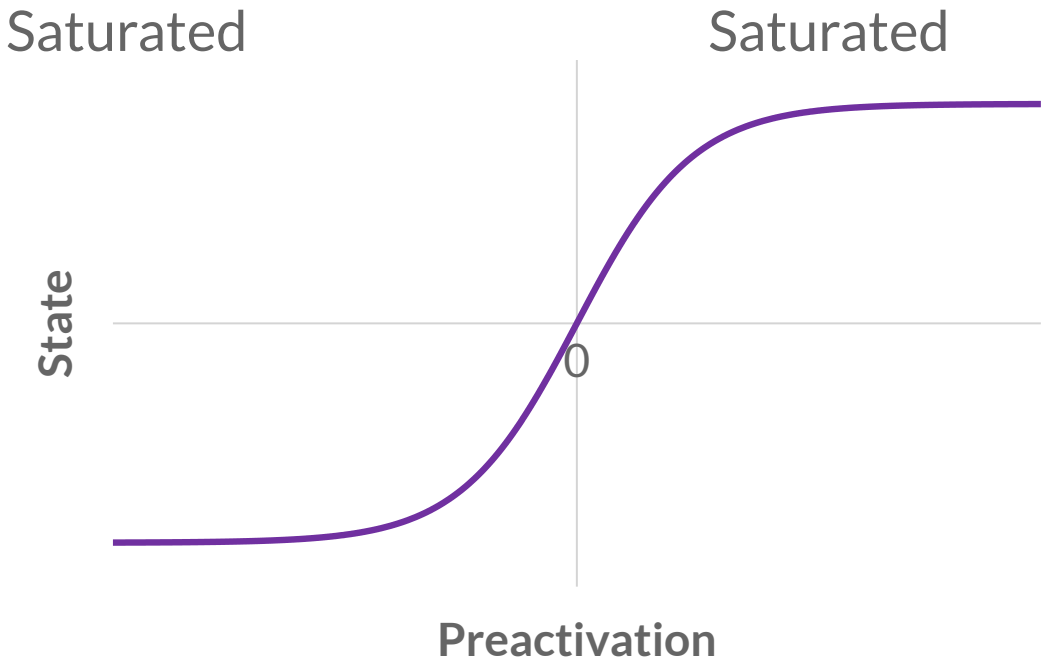


Linear recurrence

$$\left|\frac{\partial h_t}{\partial h_{t-k}}\right| = |w|^k$$

Repeated contraction or expansion

The effect of the activation



Responsive

$$h_t = \tanh(a_t)$$

$$a_t = wh_{t-1} + \text{input term}$$

$$\frac{\partial h_t}{\partial h_{t-1}} = w \tanh'(a_t)$$

Weight × activation slope

Gradient clipping

$$g \leftarrow g \min\left(1, \frac{c}{\|g\|}\right)$$

g: accumulated gradient c: clipping threshold

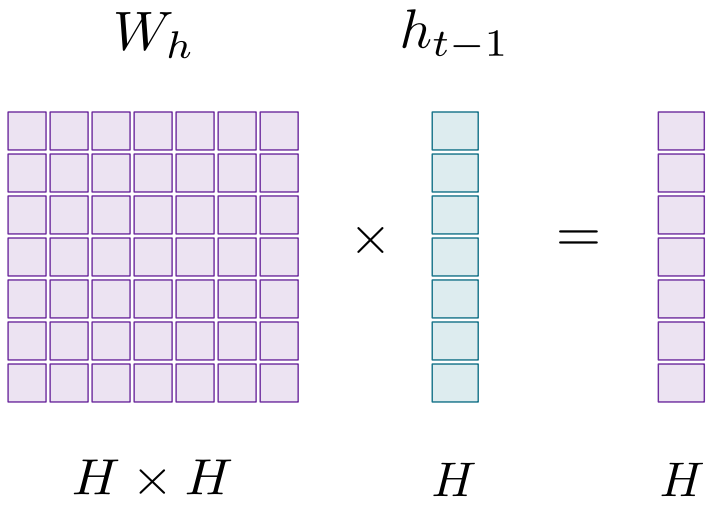
Large gradient

Clipped gradient

Clipping does not restore a vanishing gradient.

Forward computation cost

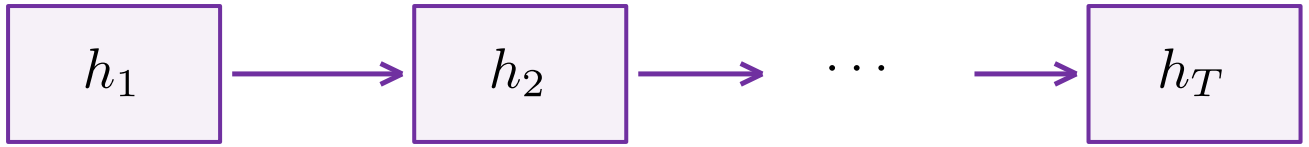
H: state width T: time steps L: recurrent layers



Work per state update $O(H^2)$

Dense input and recurrent products

T time steps through L layers



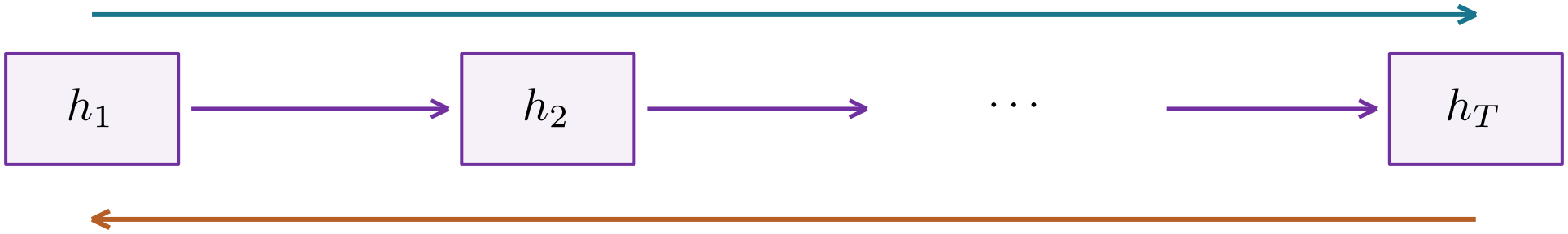
$O(TLH^2)$

One sequence. Input width comparable to H. Output head excluded.

Backward computation cost

One traversal in each direction

Forward $O(TLH^2)$



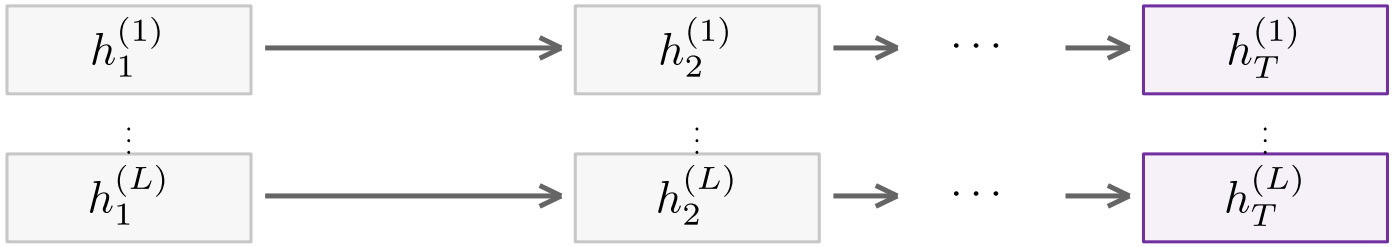
Backward $O(TLH^2)$

Future-loss gradients accumulate before each backward step.

The recurrent state chain still limits time-step parallelism.

Training on long sequences

Streaming inference

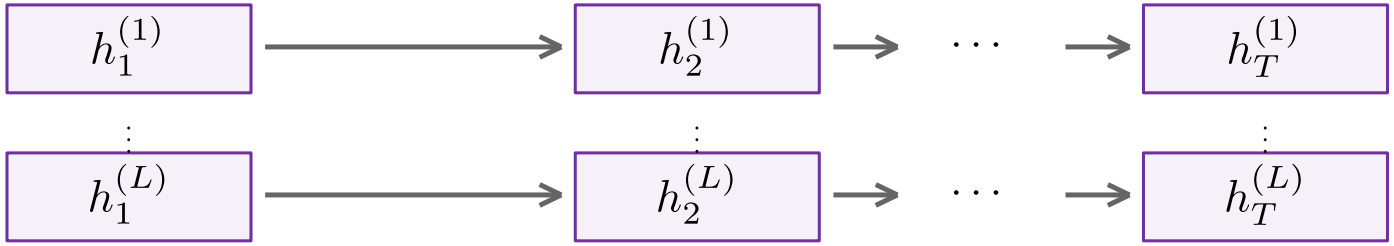


Discard earlier states after use

Current layer states

$$O(LH)$$

Full BPTT



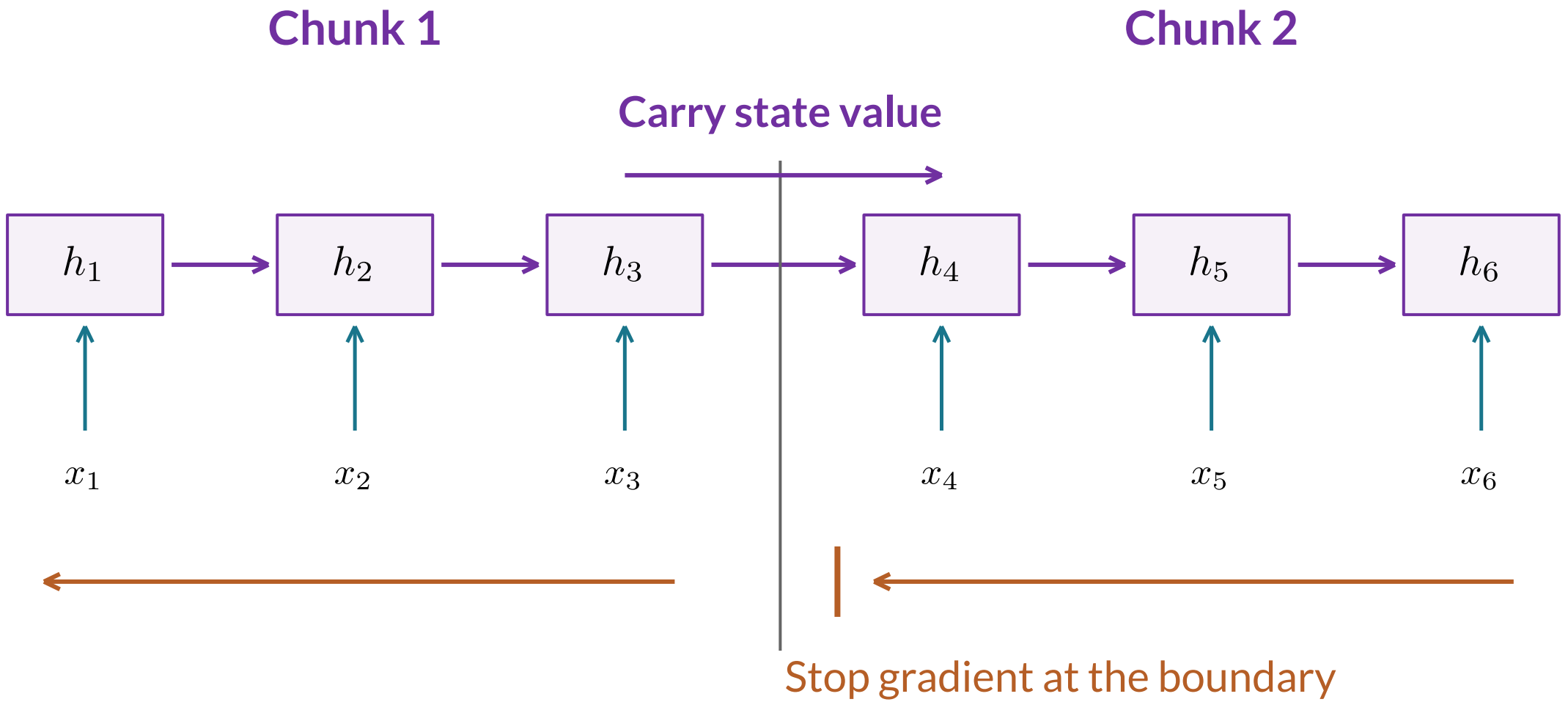
Save states and intermediate values for backward

Saved activations

$$O(TLH)$$

Truncated BPTT

Shorter chunks save memory but shorten the gradient path.



Source: Adapted from Georgia Tech CS 7643

Detaching the recurrent state

```
# After training on the previous chunk:
h = h.detach()                  # keep values, cut graph
out, h = rnn(next_chunk, h)  # continue the same stream
```

Same hidden-state values

No gradient path into the previous chunk

Lecture outline

01 Sequence tasks and recurrent state

02 Language modeling

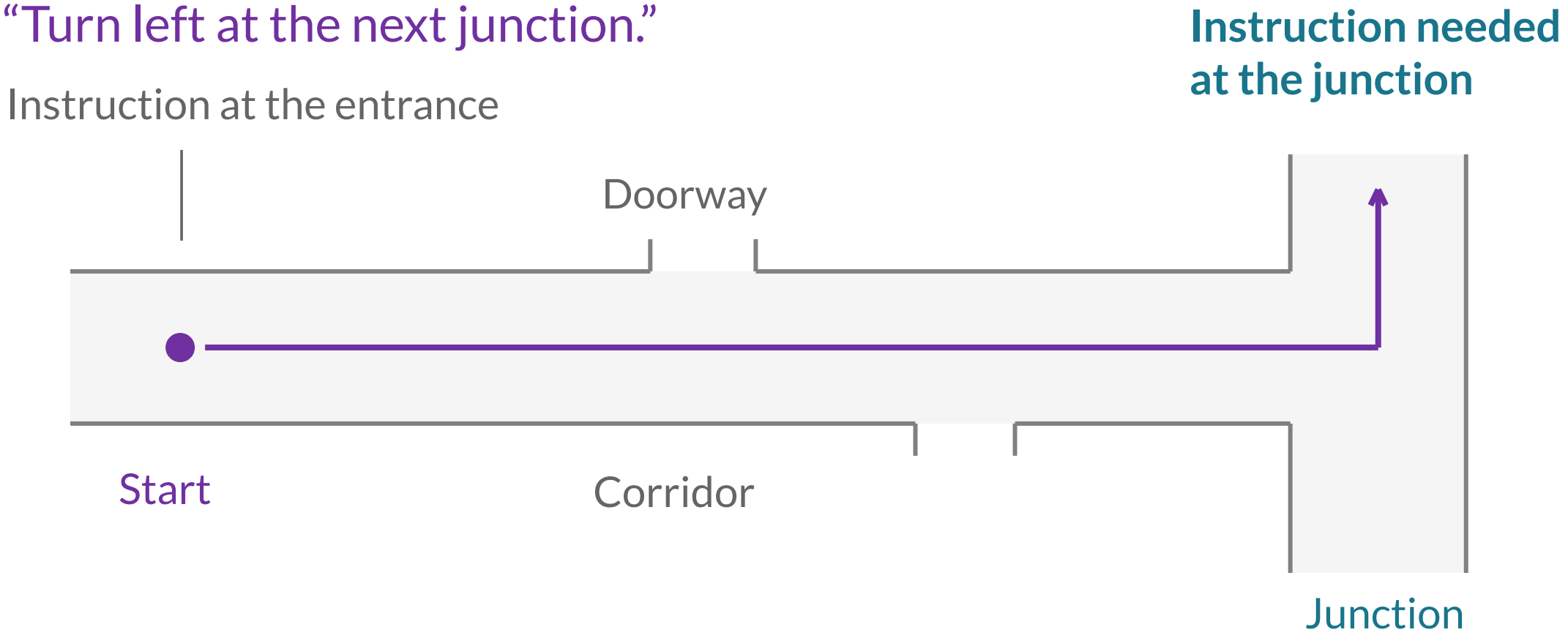
03 Backpropagation through time (BPTT)

04 LSTM

05 Using recurrent layers

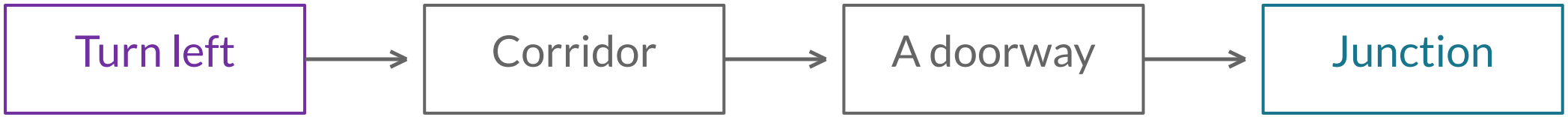
Selective memory

"Turn left at the next junction."

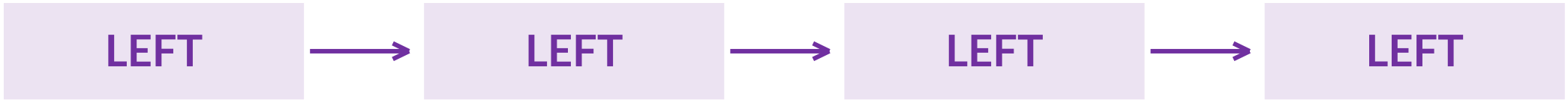


Keep the instruction while new observations arrive

Retaining an instruction



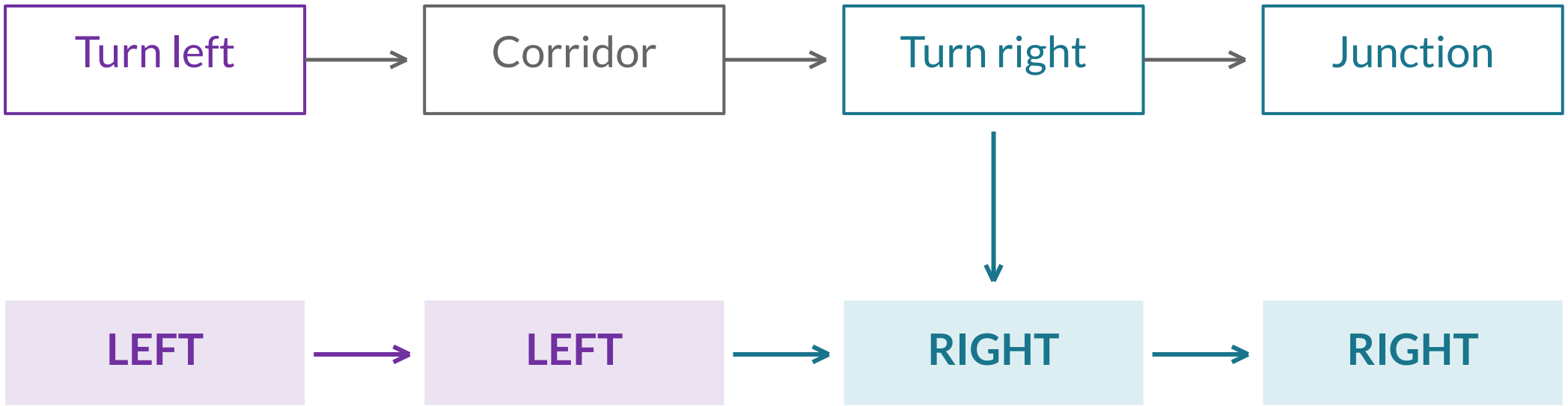
Relevant memory components



Retain the old memory

Write no new content

Replacing obsolete memory



Discard obsolete memory

Write the new instruction

Long short-term memory (LSTM)

LONG SHORT-TERM MEMORY

NEURAL COMPUTATION 9(8):1735–1780, 1997

Sepp Hochreiter and Jürgen Schmidhuber

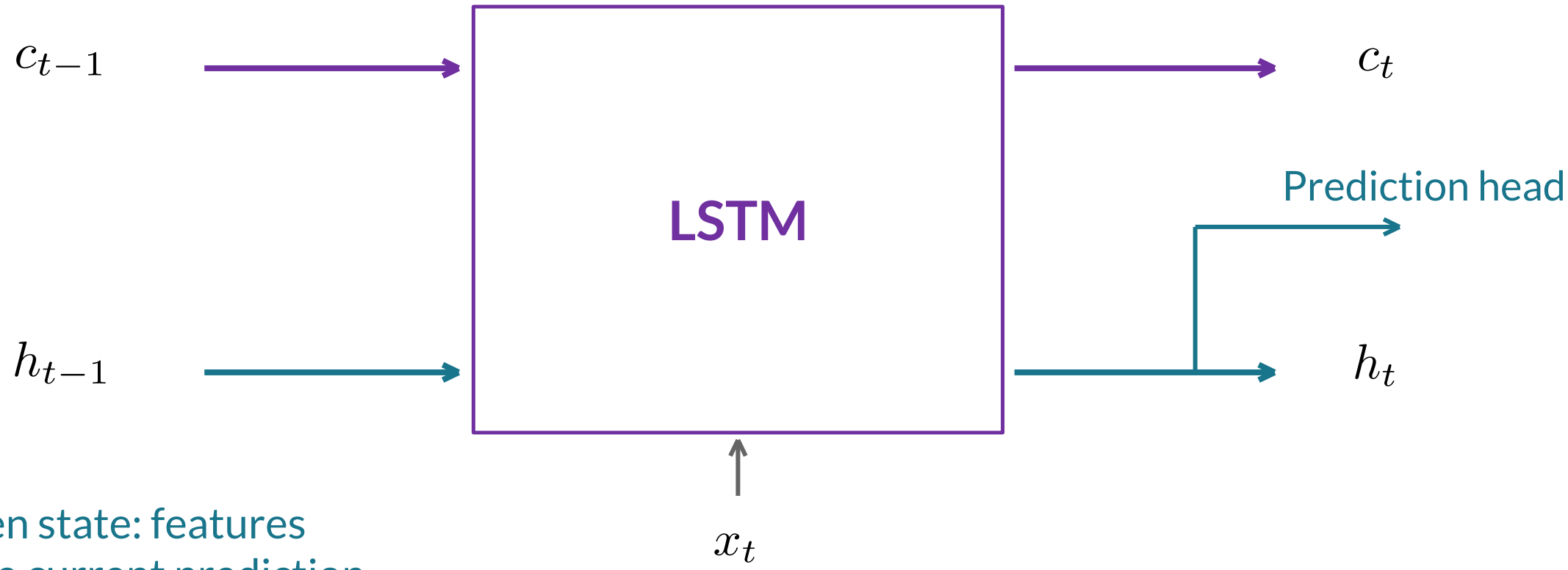
1997 Gated memory for long dependencies

1999–2000 Forget gates allow obsolete memory to be reset

Source: Long Short-Term Memory, 1997

Cell state and hidden state

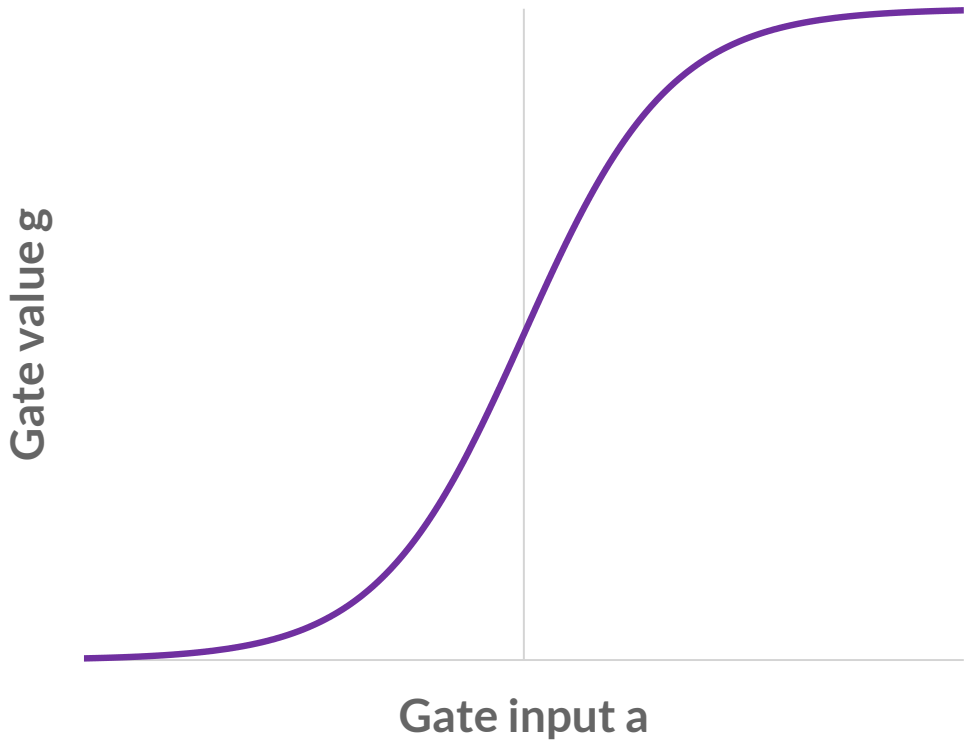
Cell state: information retained for later



Hidden state: features
for the current prediction

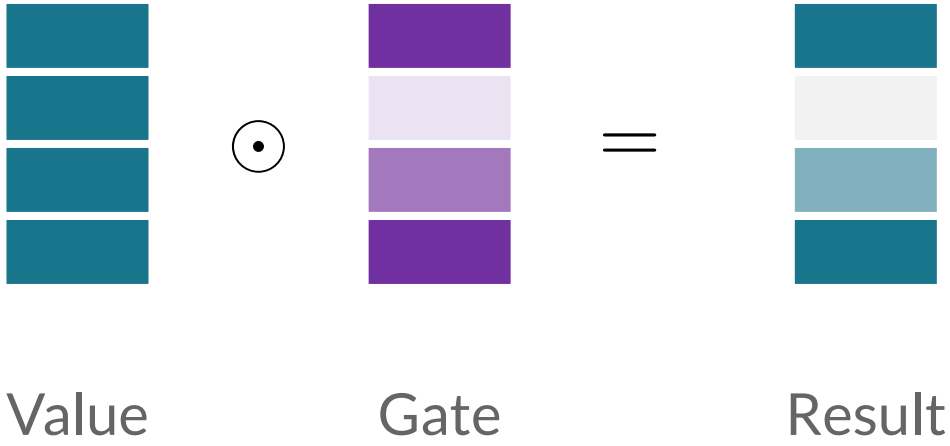
Gates

$$g = \sigma(a)$$



Gate value near 0: suppress

Gate value near 1: preserve



One control per component

Forget gate

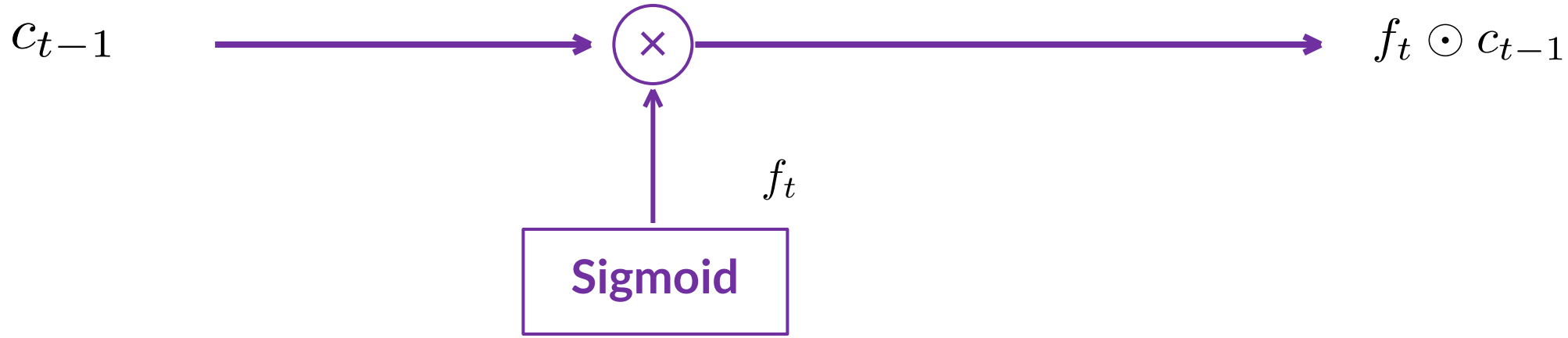
$$f_t = \sigma(W_f[h_{t-1}; x_t] + b_f)$$

f_t forget gate

W_f, b_f learned weights and bias

near 1: keep

near 0: discard



$[h_{t-1}; x_t]$ concatenate previous hidden state and current input

Candidate memory

$$\widetilde{c}_t = \tanh(W_c[h_{t-1}; x_t] + b_c)$$

Candidate content

What could be added

Gate value

How much passes through

$$-1 < \widetilde{c}_{t,j} < 1$$

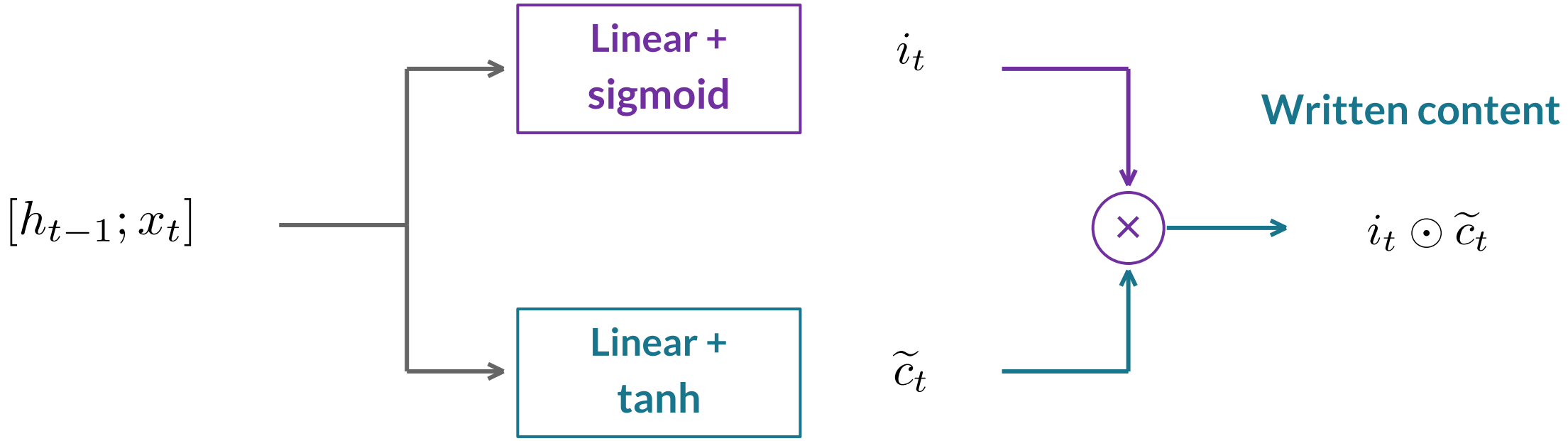
$$0 < g_{t,j} < 1$$

j: component index

Input gate

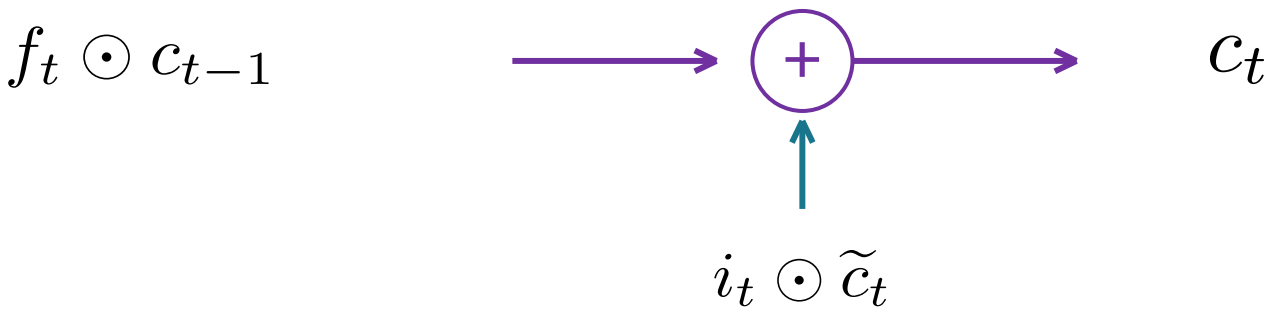
$$i_t = \sigma(W_i[h_{t-1}; x_t] + b_i)$$

i_t input gate: controls what is written



Updating the cell state

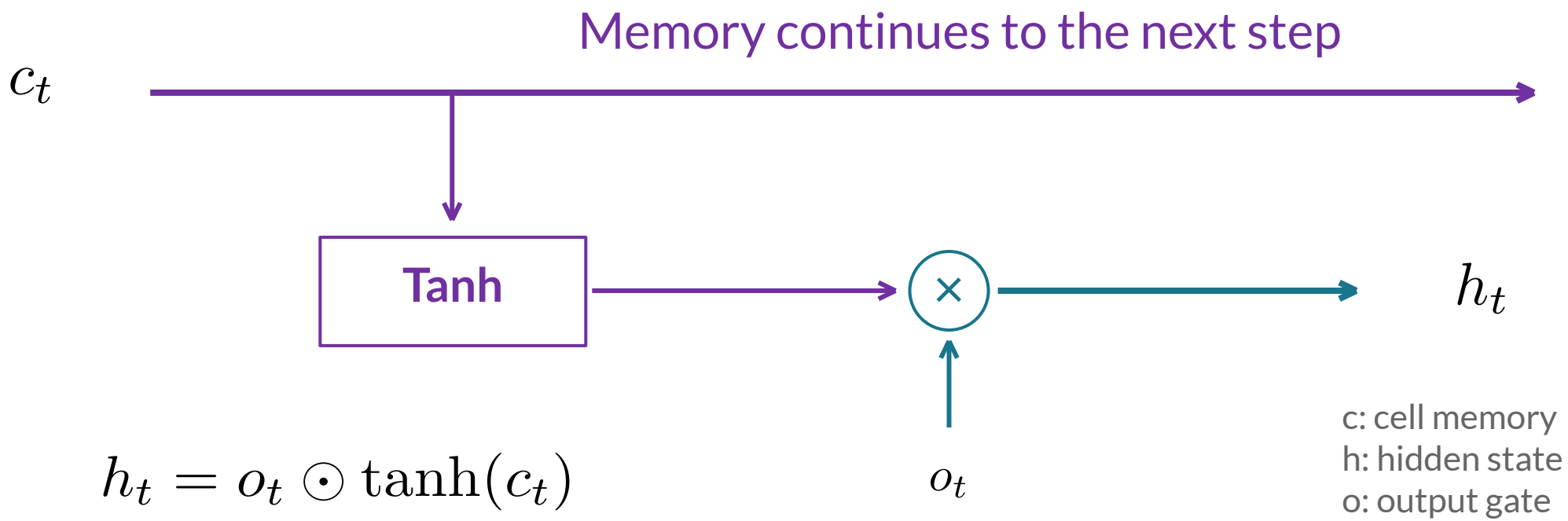
$$c_t = \underbrace{f_t \odot c_{t-1}}_{\text{Retained memory}} + \underbrace{i_t \odot \widetilde{c}_t}_{\text{Written content}}$$



Output gate

Control what the current prediction reads from memory

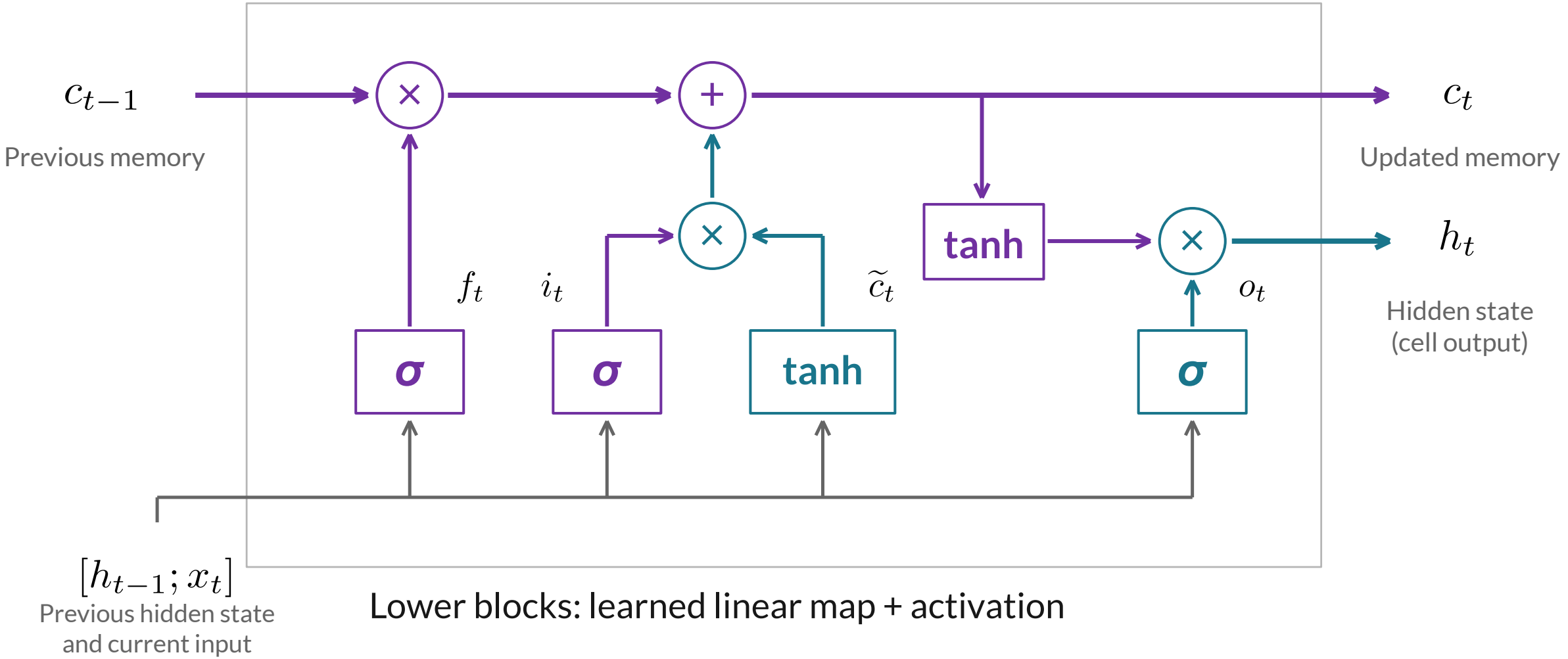
$$o_t = \sigma(W_o[h_{t-1}; x_t] + b_o)$$



$$h_t = o_t \odot \tanh(c_t)$$

c: cell memory
h: hidden state
o: output gate

LSTM cell



Lower blocks: learned linear map + activation

LSTM equations

Gates and proposed content

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

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

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

Candidate $$\widetilde{c}_t = \tanh(W_c[h_{t-1}; x_t] + b_c)$$

State updates

Stored memory

$$c_t = f_t \odot c_{t-1} + i_t \odot \widetilde{c}_t$$

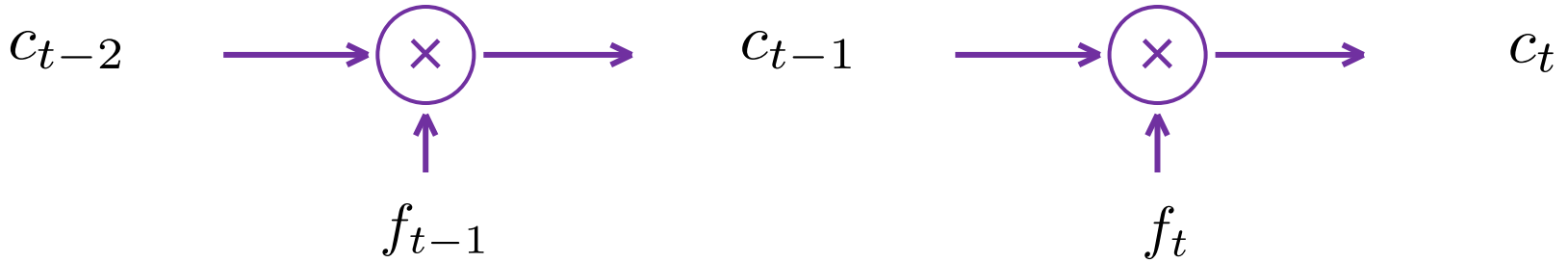
Hidden state

$$h_t = o_t \odot \tanh(c_t)$$

W, b: learned parameters x: input h: hidden state c: cell memory

The memory path

Forget gates near 1 preserve the gradient along this path



Gradient along the direct memory route

Two-step multiplier
(one memory component)

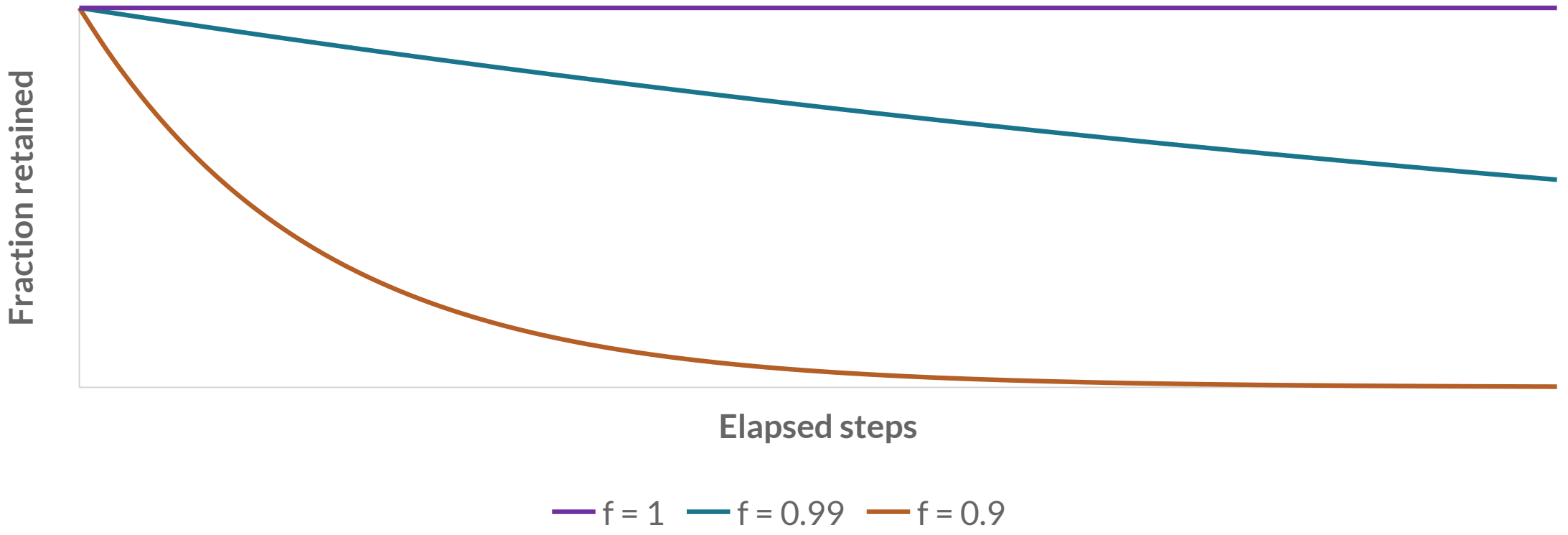
$f_{t-1} f_t$

Other gradient paths through
the gates also contribute

Memory retention over time

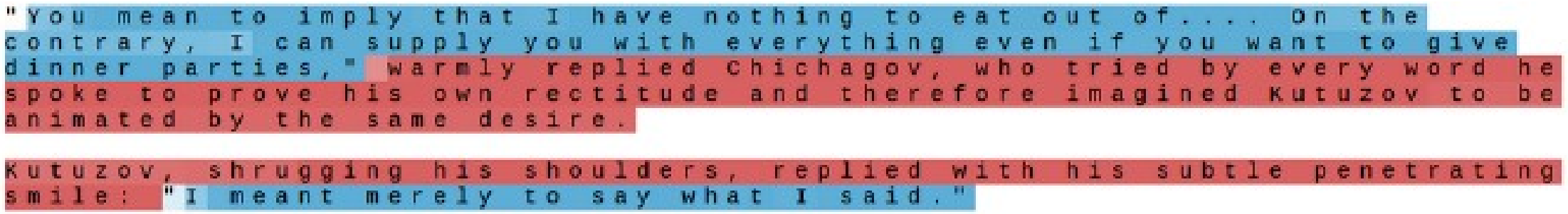
$$c_{t+k} = f^k c_t$$

One component, fixed f, no new content



Learned memory in text

Trained to predict the next character



Positive cell activation

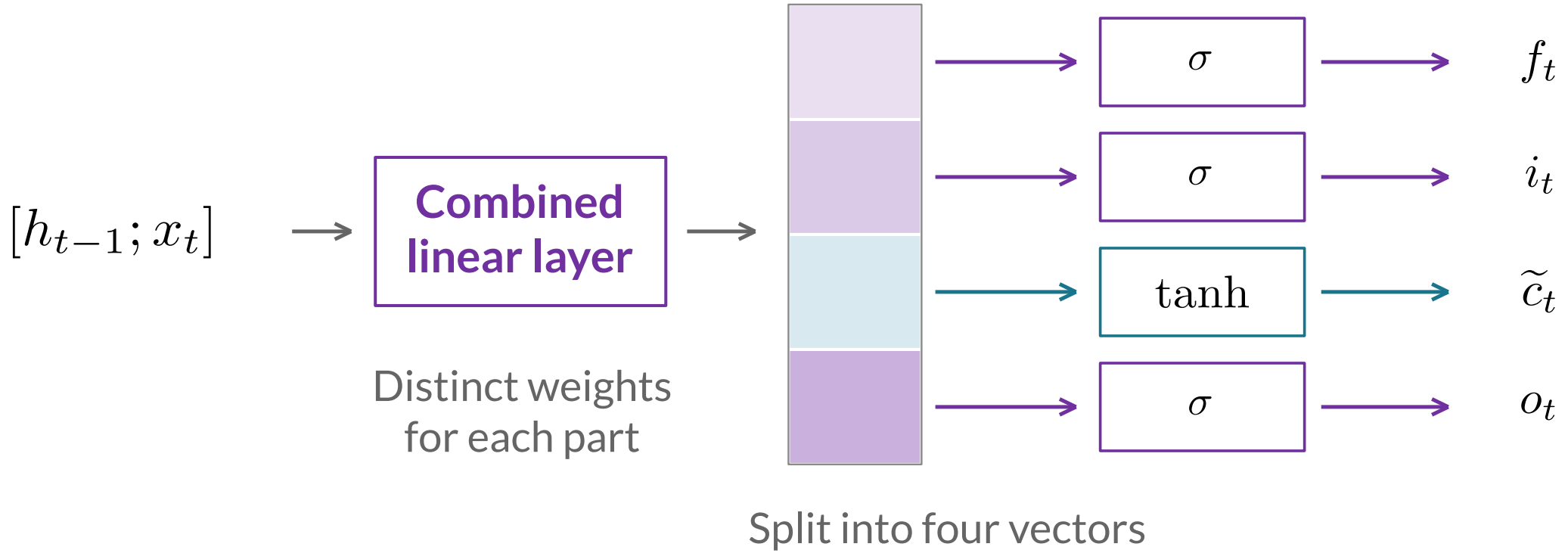
Negative cell activation

One learned cell tracks whether text is inside quotes

Source: Visualizing and Understanding Recurrent Networks

LSTM computation

All four transforms receive the same input



Time steps still depend on the previous states

Lecture outline

01 Sequence tasks and recurrent state

02 Language modeling

03 Backpropagation through time (BPTT)

04 LSTM

05 Using recurrent layers

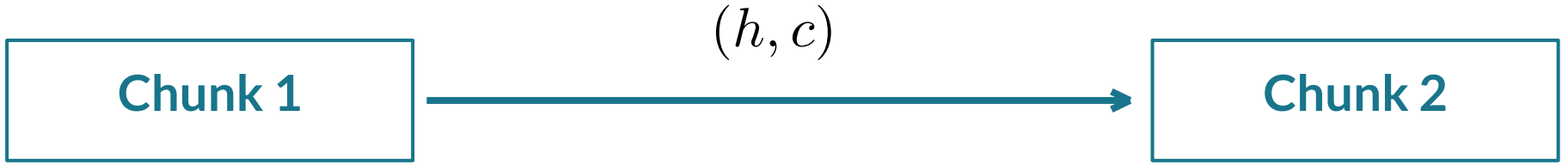
Resetting and carrying state

LSTM memory belongs to one sequence

Independent recordings

Reset: one recording must not affect another

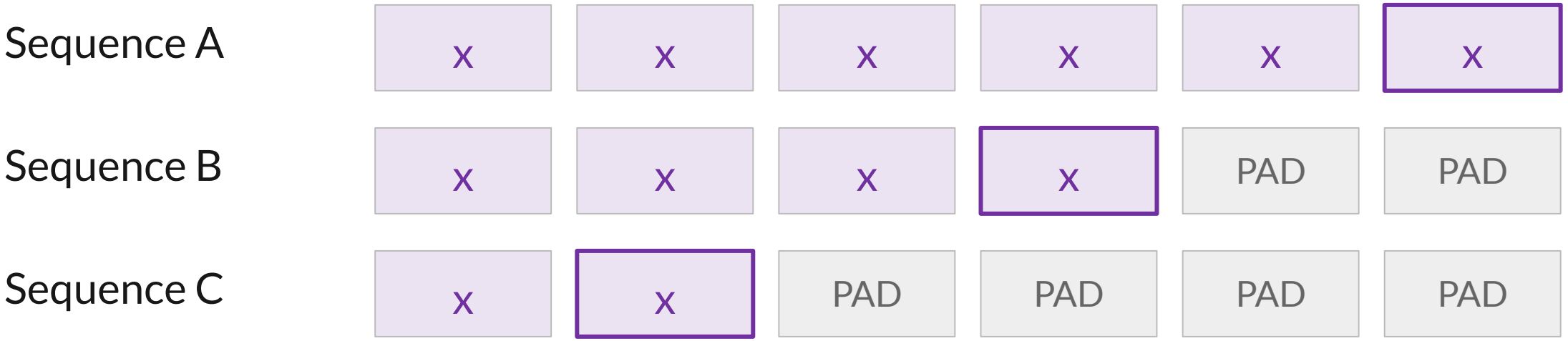
Consecutive chunks of one recording



Carry: later chunks need the earlier context

Variable-length sequences

Padding makes different lengths fit a rectangular batch



Exclude padding from the loss

Forward RNN: use the last valid state for a sequence summary

Group similar lengths to reduce padding

Batched token IDs

3 rows (examples), 6 columns (time steps)

Token IDs

Here, PAD has token ID 2

A	5	8	17	42	9	1
B	17	42	9	1	2	2
C	17	1	2	2	2	2

Valid-token mask

1: real token 0: padding

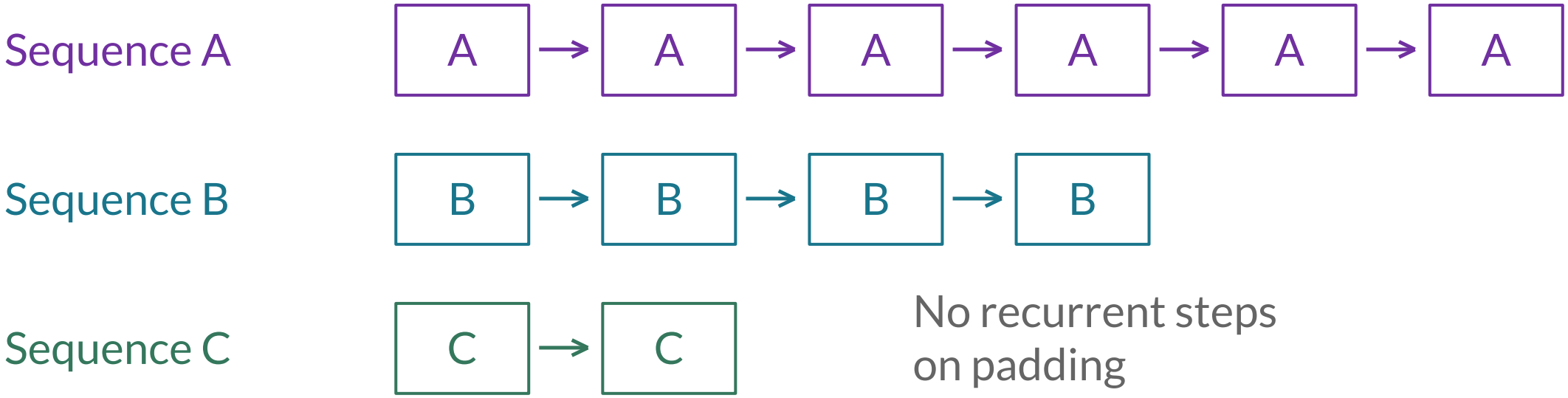
A	1	1	1	1	1	1
B	1	1	1	1	0	0
C	1	1	0	0	0	0

$$\text{lengths} = [6,\ 4,\ 2]$$

Ignore padded target positions in the loss

Packed sequences

Process only sequences that have not ended

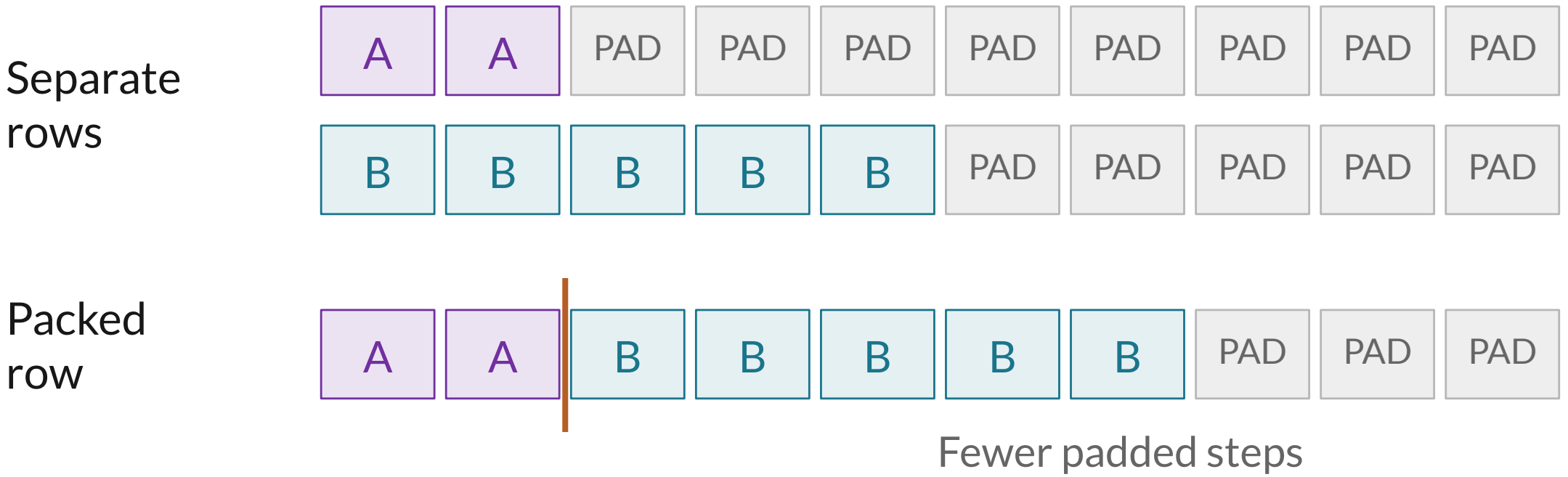


Each row keeps its own recurrent state

PyTorch: pack_padded_sequence

Packing short sequences

Fill fixed-length rows with several short examples



Reset h and c at each boundary (custom recurrent loop)

Exclude next-token targets across the boundary

LSTM in PyTorch

```
# x: (B, T, D), equal-length input sequences
rnn = nn.LSTM(D, H, num_layers=2, batch_first=True)
head = nn.Linear(H, C)        # C classes

output, (h_n, c_n) = rnn(x)  # zero initial states
logits = head(h_n[-1])        # final top-layer state

# output: top-layer hidden states at all steps (B, T, H)
# h_n, c_n: final states by layer (2, B, H)
```

B: batch size T: steps D: input features H: hidden width

Next: sequence-to-sequence models

Input: English

The red car stopped.

Output: French

La voiture rouge s’est arrêtée.

Different lengths and word order

Also: observed measurements to a future trajectory