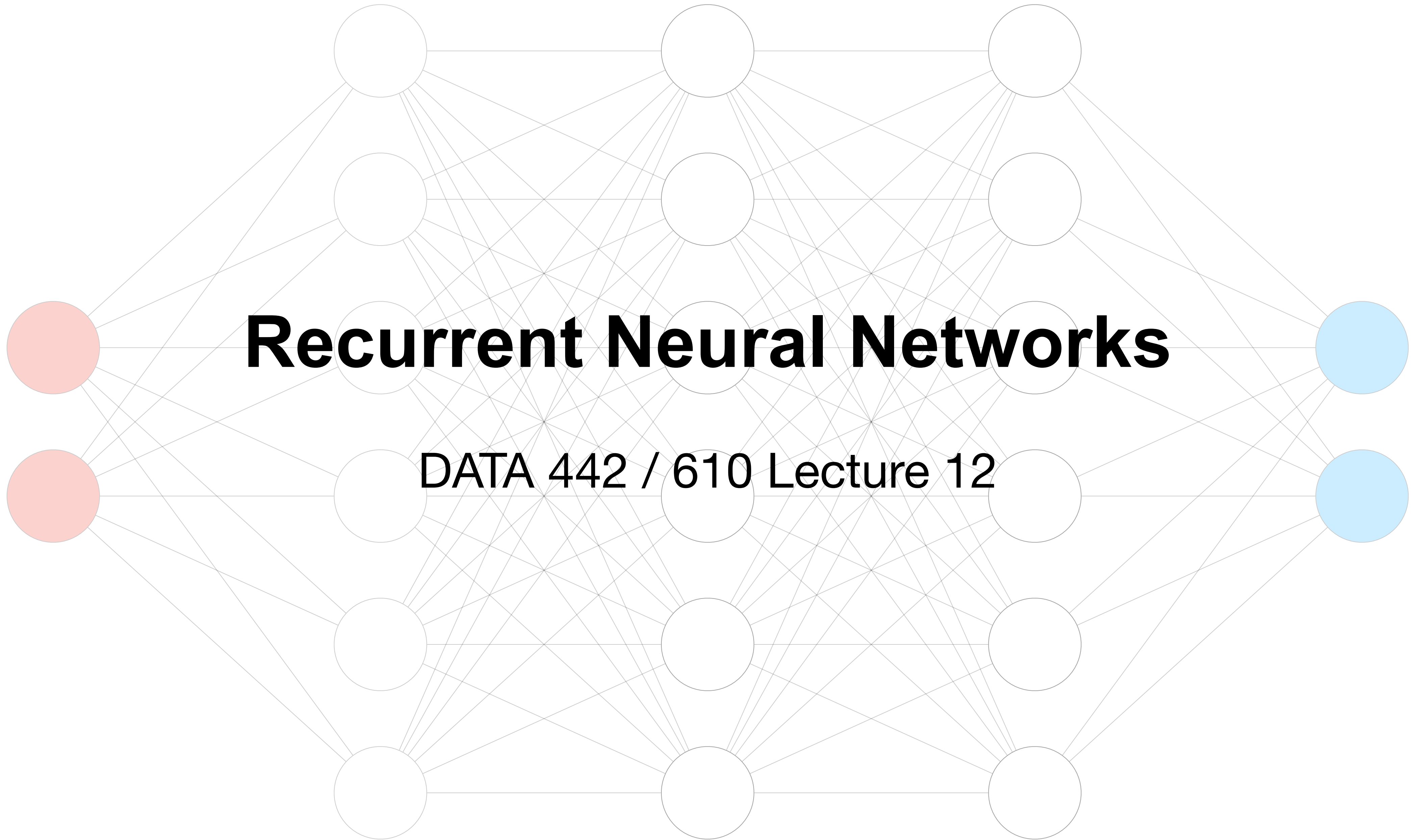


Recurrent Neural Networks

DATA 442 / 610 Lecture 12



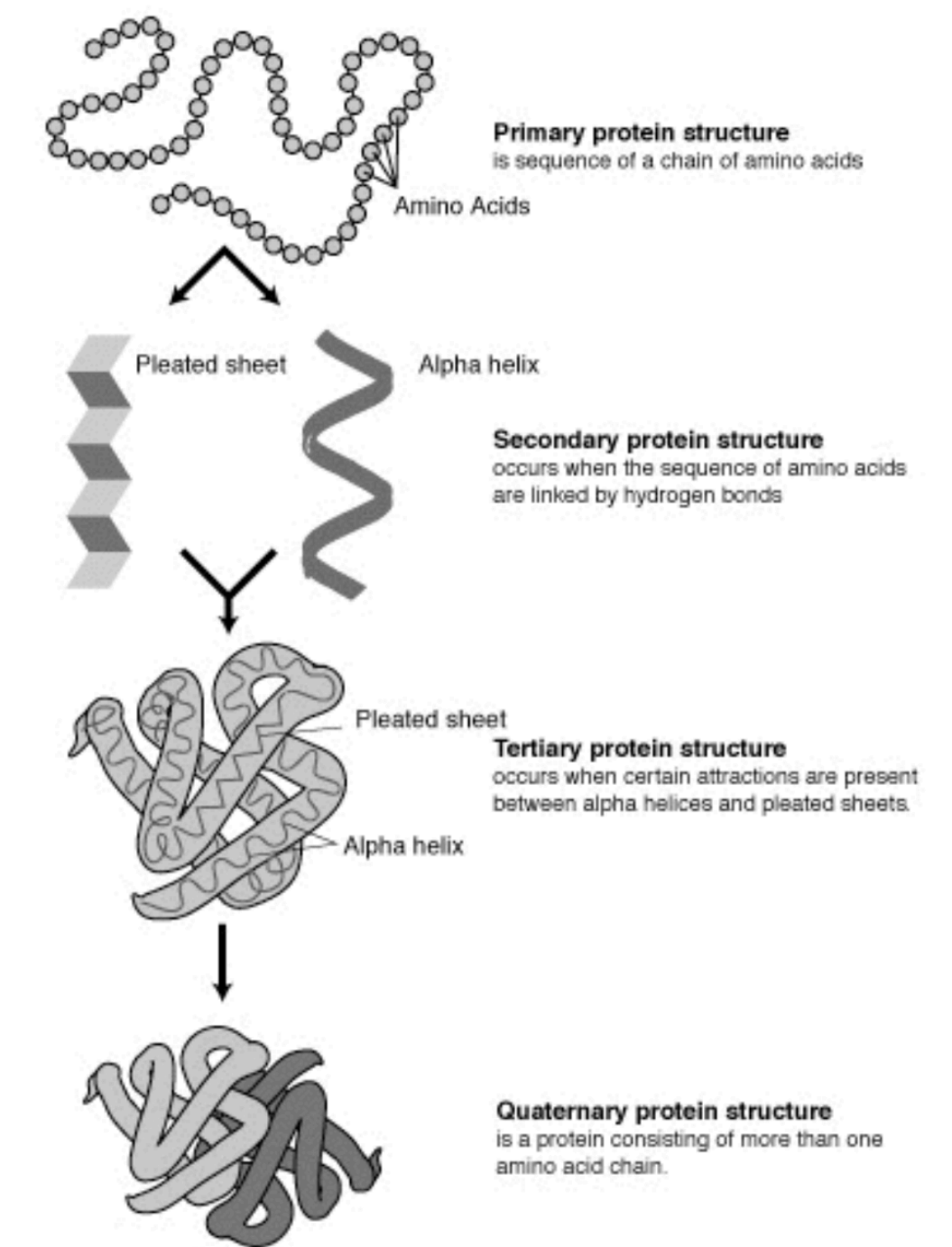
Sequential data

Sequential data appears in a particular order and can have arbitrary length

$$\{x^{(1)}, x^{(2)}, x^{(3)}, \dots, x^{(t)}\} \quad t > 1$$

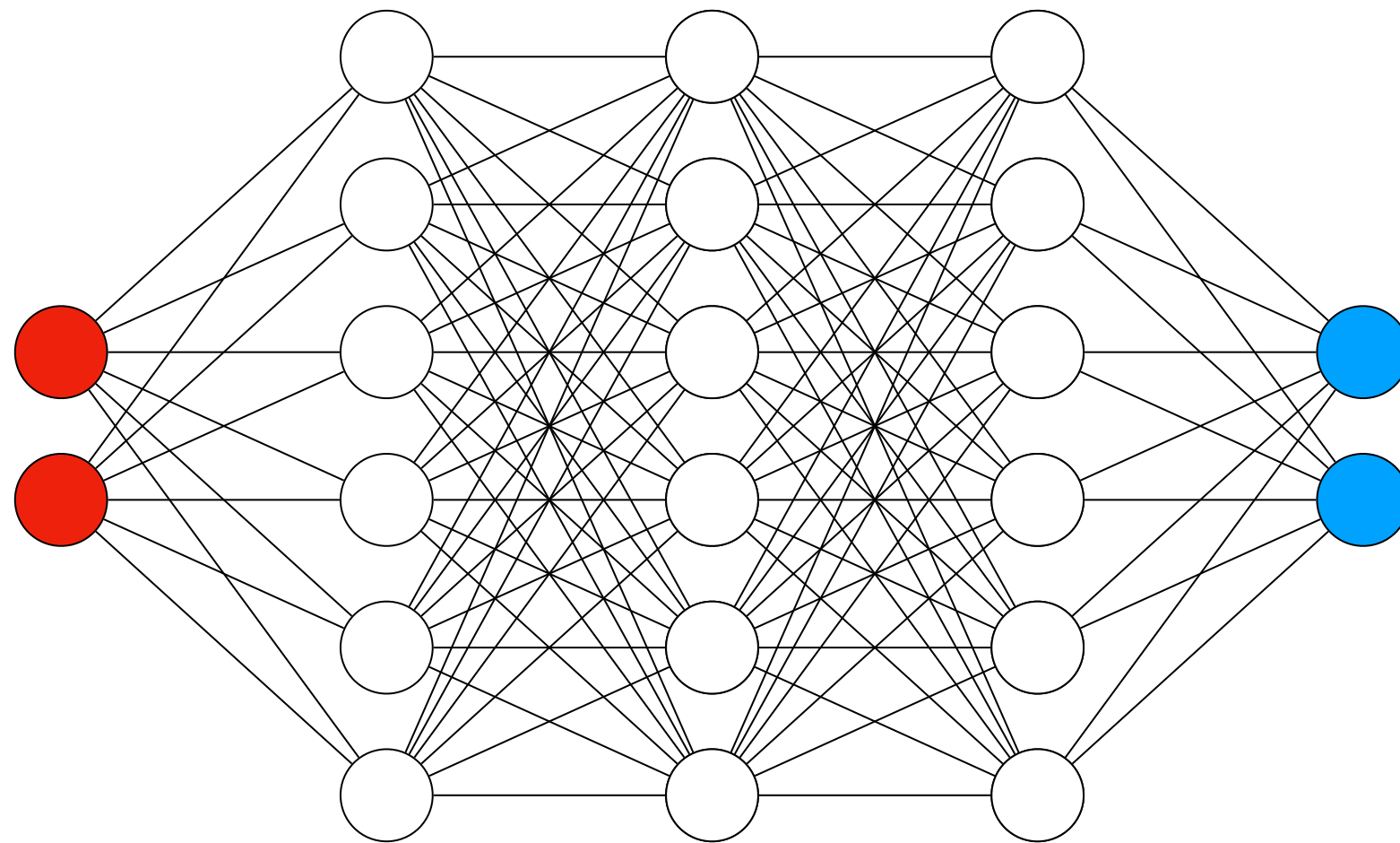
Examples:

- Time series data, e.g. stock prices each day, hour, minute
- Text, e.g. “Once upon a time there was a ...”
- DNA sequences, e.g. ACCTGCTGATTACA
- Amino acid chains (AlphaFold)



How to process sequential data?

Fully-connected NN

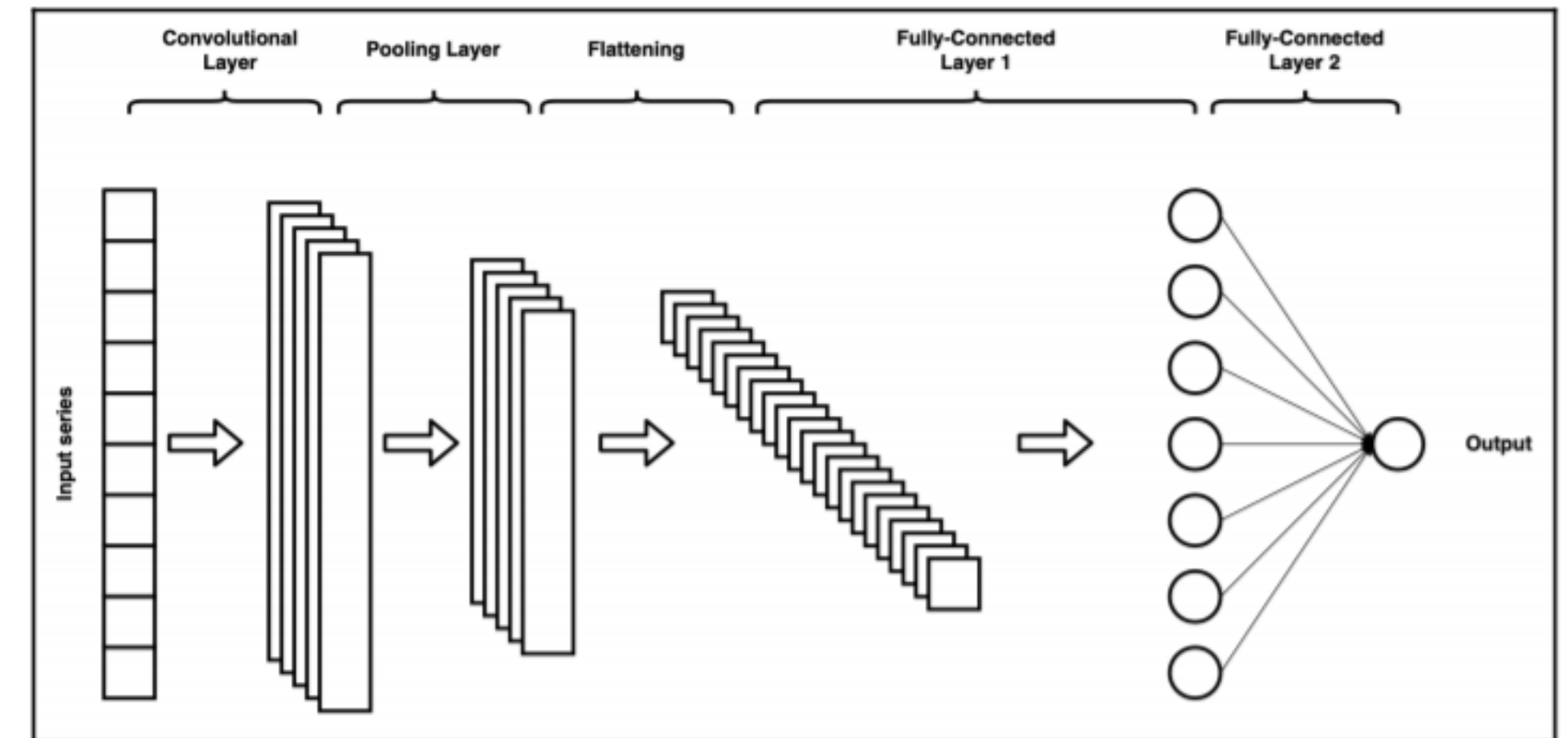


Requires fixed-length inputs

Cannot generalize to longer sequences

Struggles with variable ordering for some problems

1-dimensional CNN



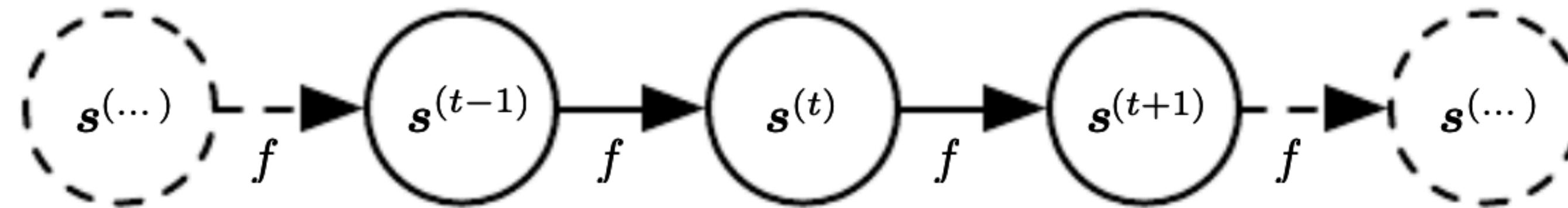
Accepts arbitrary sequence lengths due to **parameter sharing in time**

Shallow - only a few inputs contribute to predictions at each step

Fixed-size computational graph

Recursive structure in dynamical (sequential) systems

$$s^{(t)} = f(s^{(t-1)})$$



$$s^{(1)} = f(s^{(0)})$$

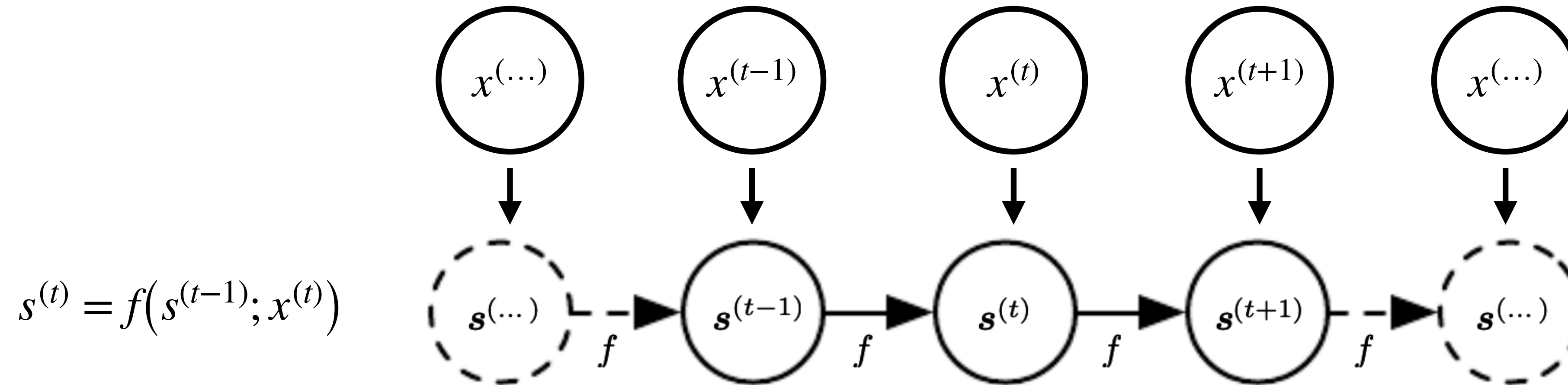
$$s^{(2)} = f(s^{(1)}) \quad s^{(2)} = f(f(s^{(0)}))$$

$$s^{(3)} = f(s^{(2)}) \quad s^{(3)} = f(f(s^{(1)})) \quad s^{(3)} = f(f(f(s^{(0)})))$$

$$s^{(4)} = f(s^{(3)}) \quad s^{(4)} = f(f(s^{(2)})) \quad s^{(4)} = f(f(f(s^{(1)}))) \quad s^{(4)} = f(f(f(f(s^{(0)}))))$$

Recursive structure can be “unfolded” into explicit computations

Dynamical systems with external driving



$$s^{(1)} = f(s^{(0)}; x^{(1)})$$

$$s^{(2)} = f(s^{(1)}; x^{(2)})$$

$$s^{(2)} = f(f(s^{(0)}; x^{(1)}); x^{(2)})$$

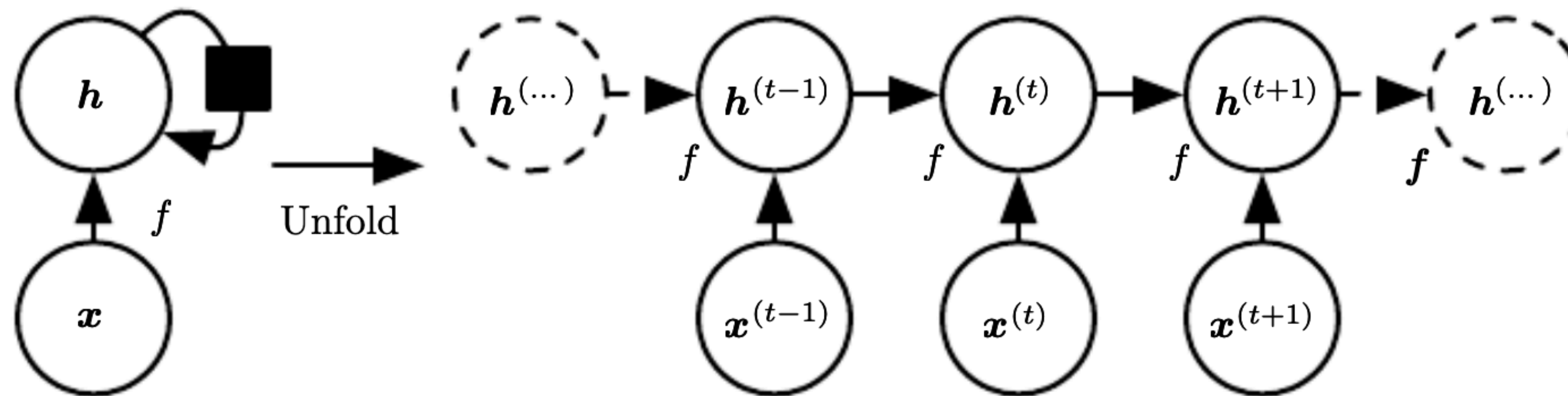
$$s^{(3)} = f(s^{(2)}; x^{(3)})$$

$$s^{(3)} = f(f(f(s^{(0)}; x^{(1)}); x^{(2)}); x^{(3)})$$

$$s^{(3)} = f(f(f(s^{(0)}; x^{(1)}); x^{(2)}); x^{(3)})$$

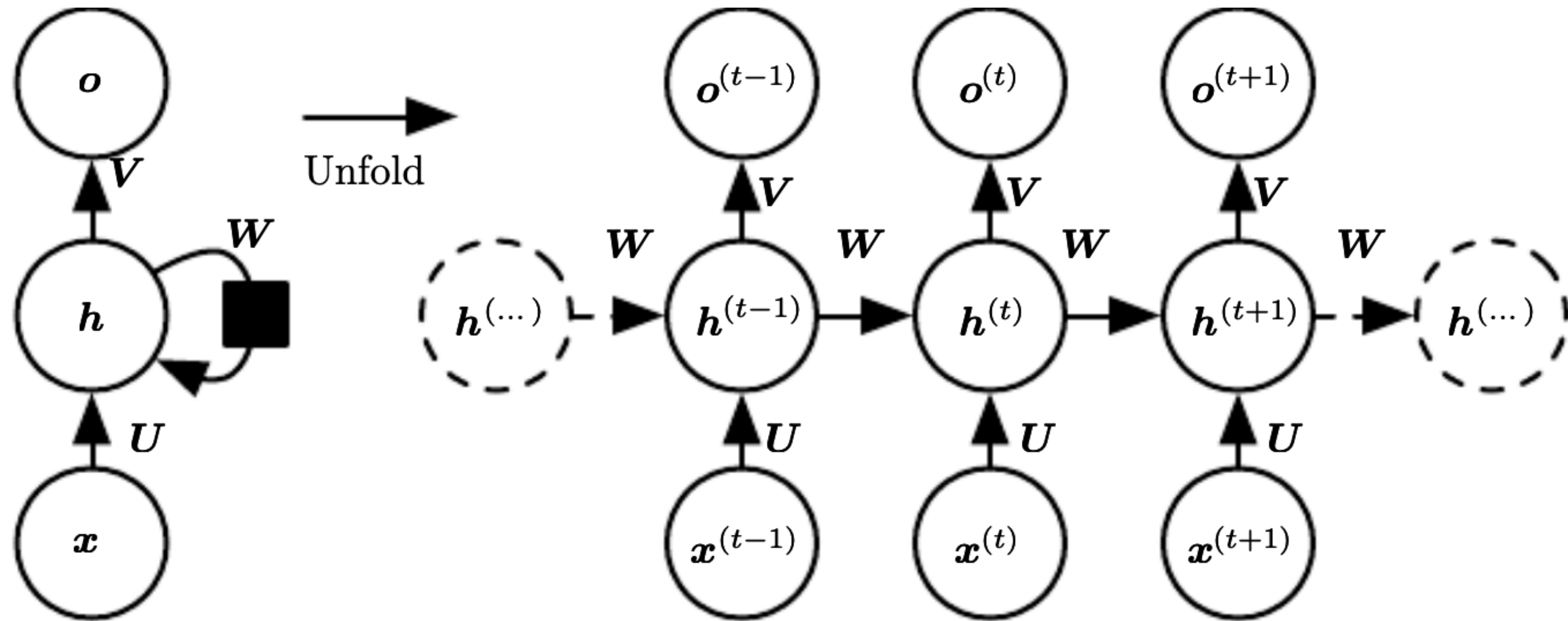
Recurrent neural networks

$$h^{(t)} = f(h^{(t-1)}; x^{(t)}; \theta)$$



RNN models evolution of a **hidden state** influenced by an input sequence

Recurrent neural networks



Recurrent neural networks share weights applied to both input sequence elements and the previous **hidden state**

Typical RNN formulations

Sentiment analysis

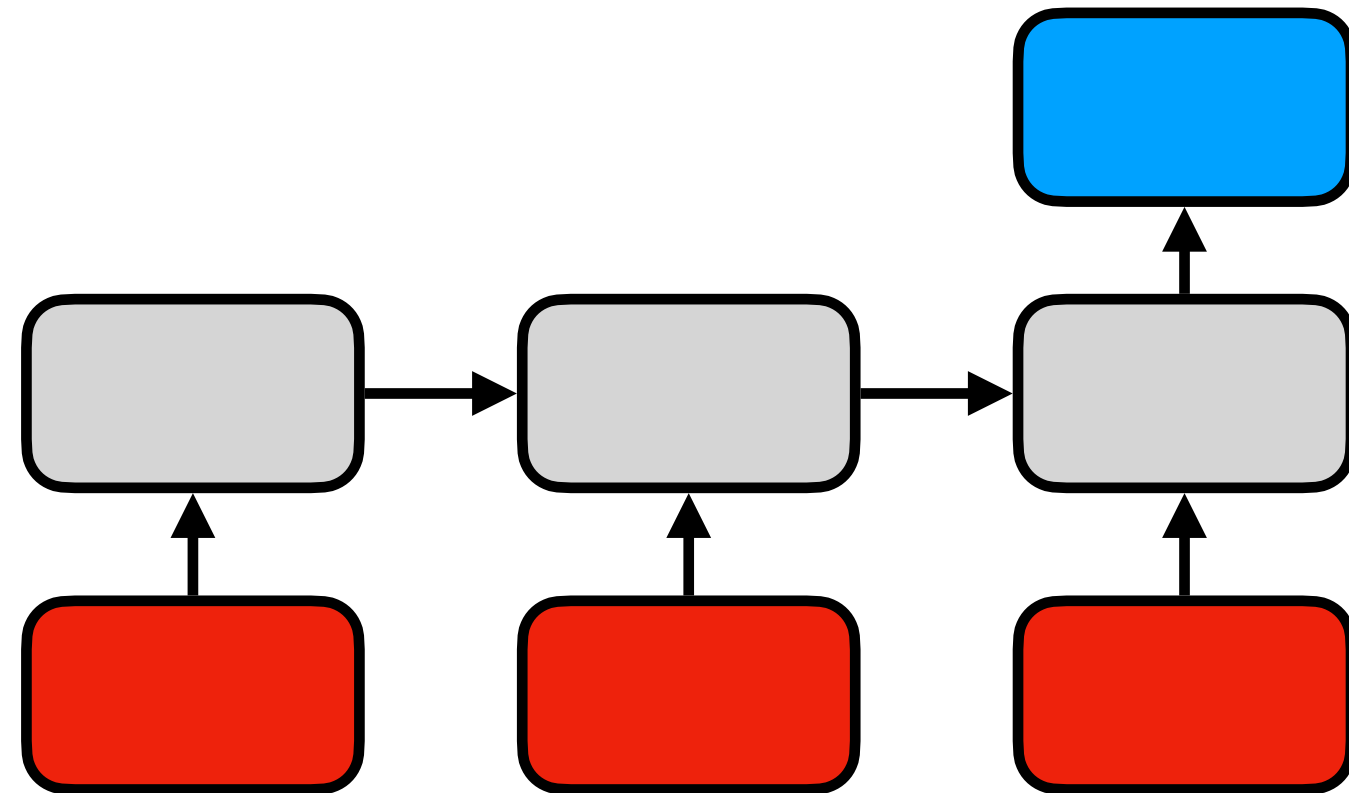
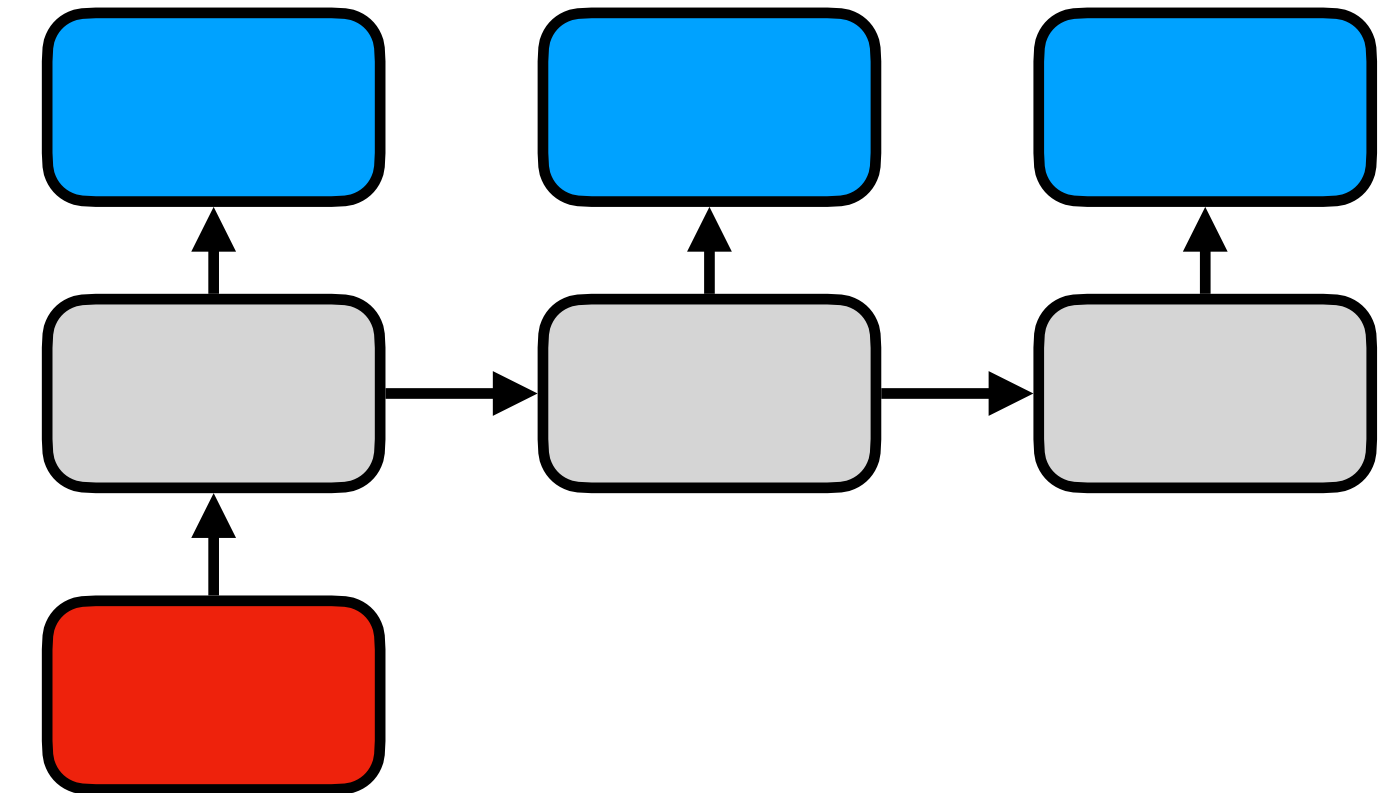
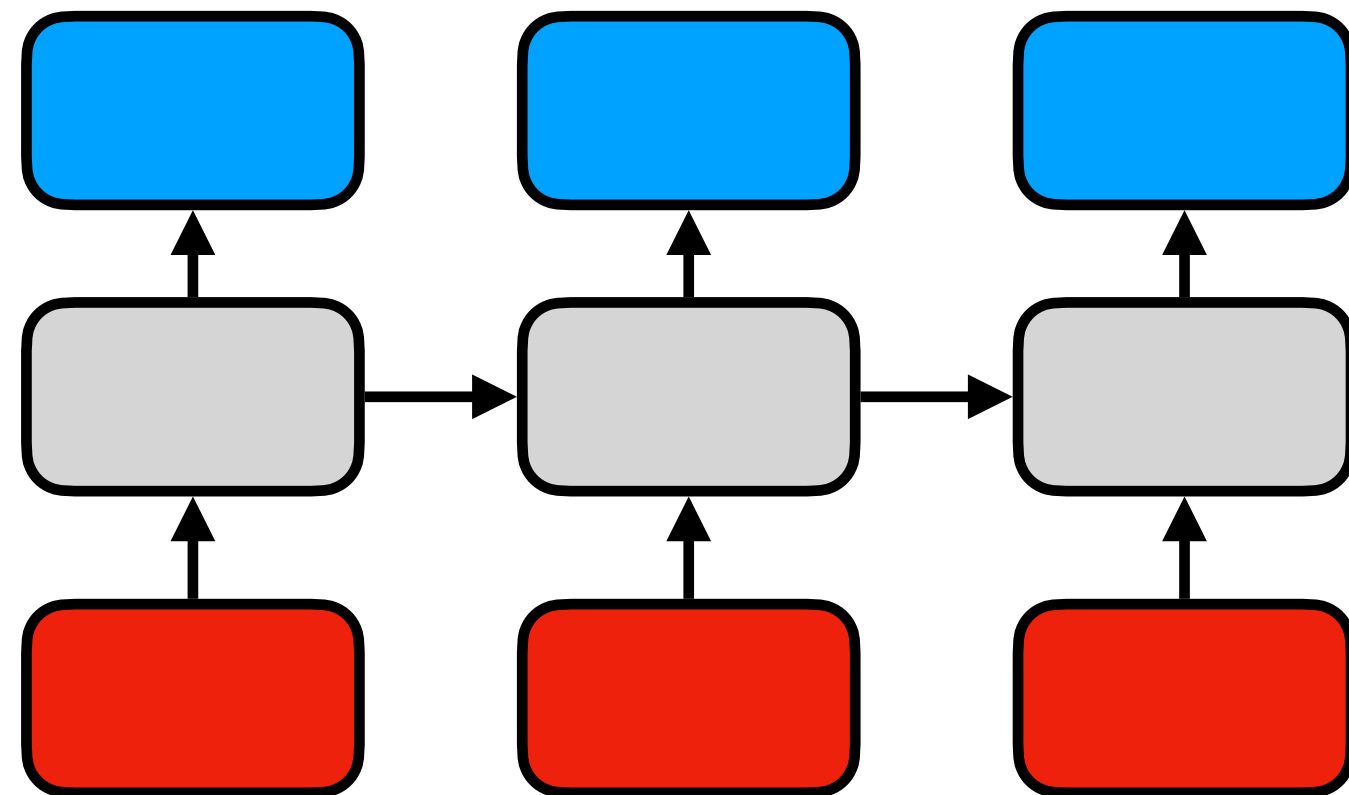


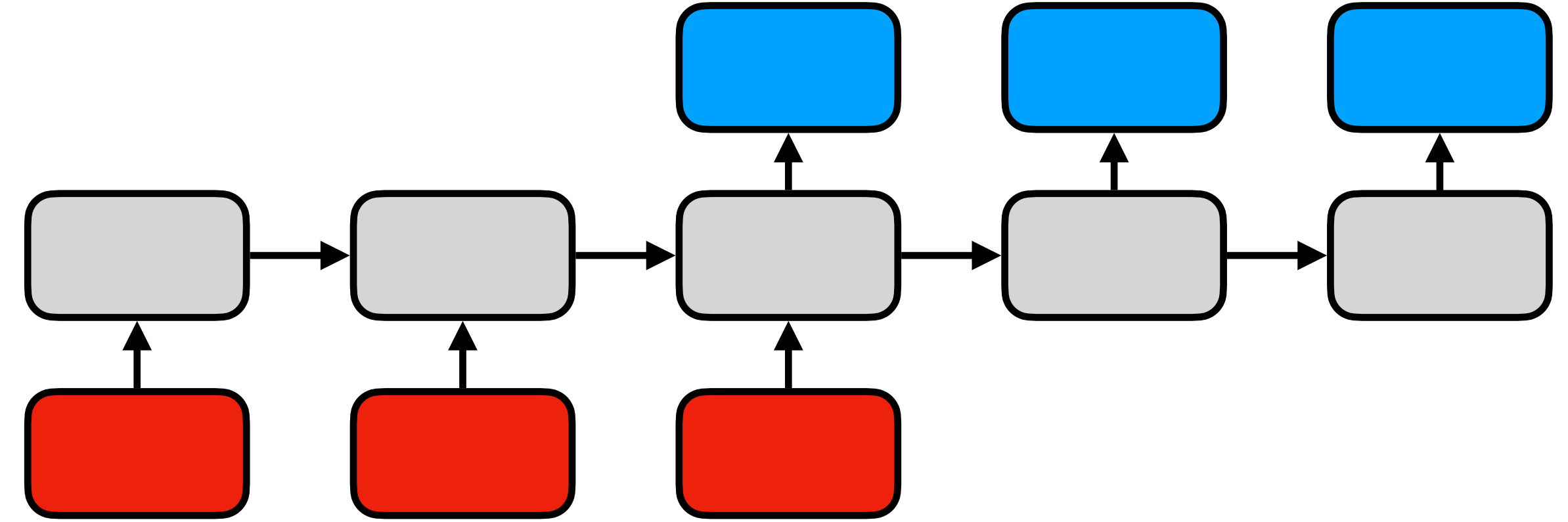
Image captioning



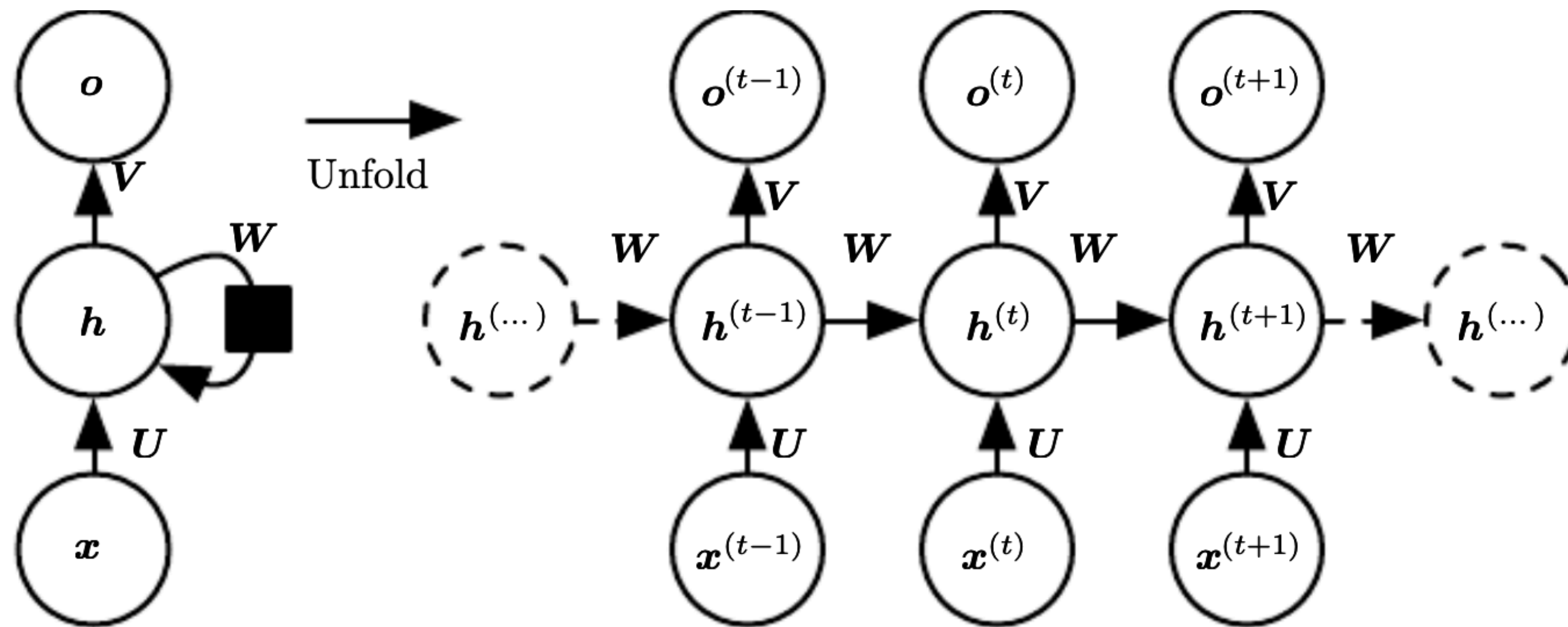
Video classification



Machine translation, text continuation



RNN forward pass



(Ignore W , V , U I do not like this notation)

$W_{ab} = W_{a \rightarrow b}$ = weight matrix for element a influencing element b

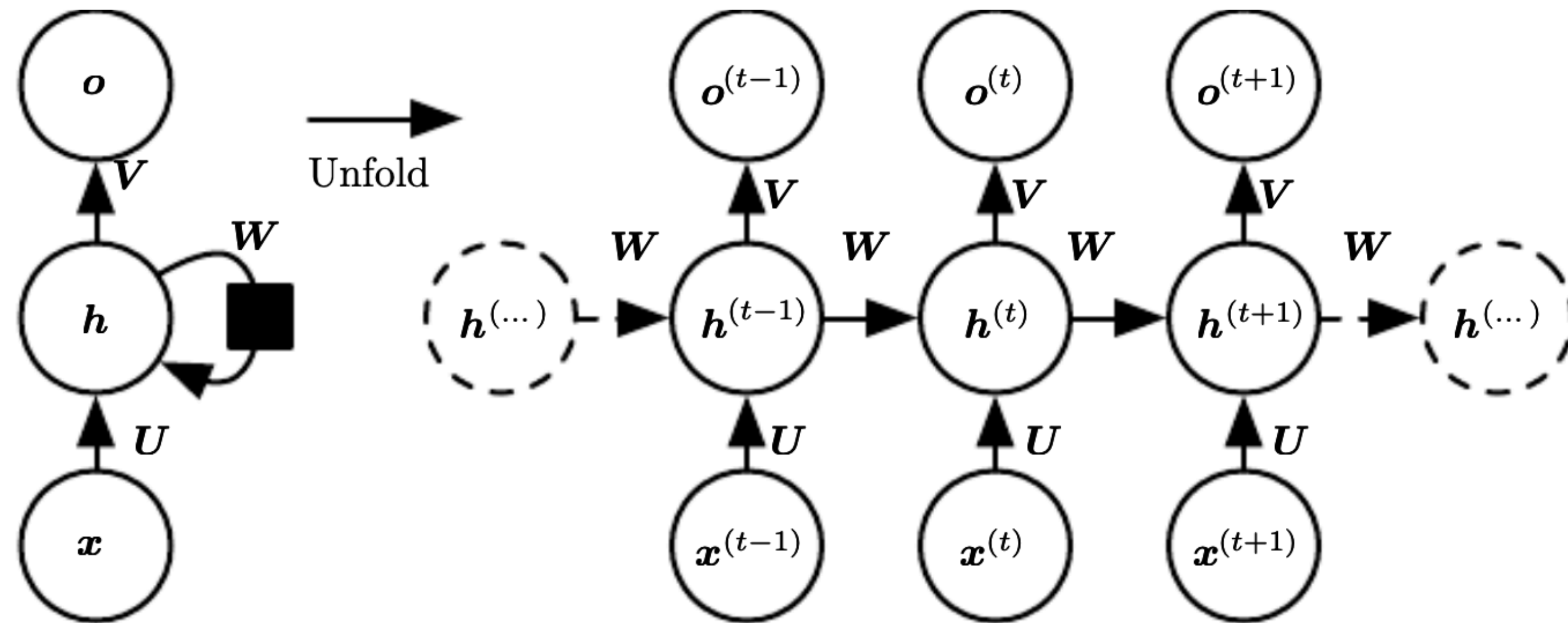
Preactivation: $z^{(t)} = b_h + W_{hh} z^{(t-1)} + W_{xh} x^{(t)}$

Hidden state: $h^{(t)} = \tanh z^{(t)}$

Output: $b_o + W_{ho} h^{(t)}$

(Optional) $y^{(t)} = \sigma(o^{(t)})$ or other activation, or softmax

RNN backward pass



Just propagate backward through time, right?

Because parameters are shared across time steps, gradients receive many contributions

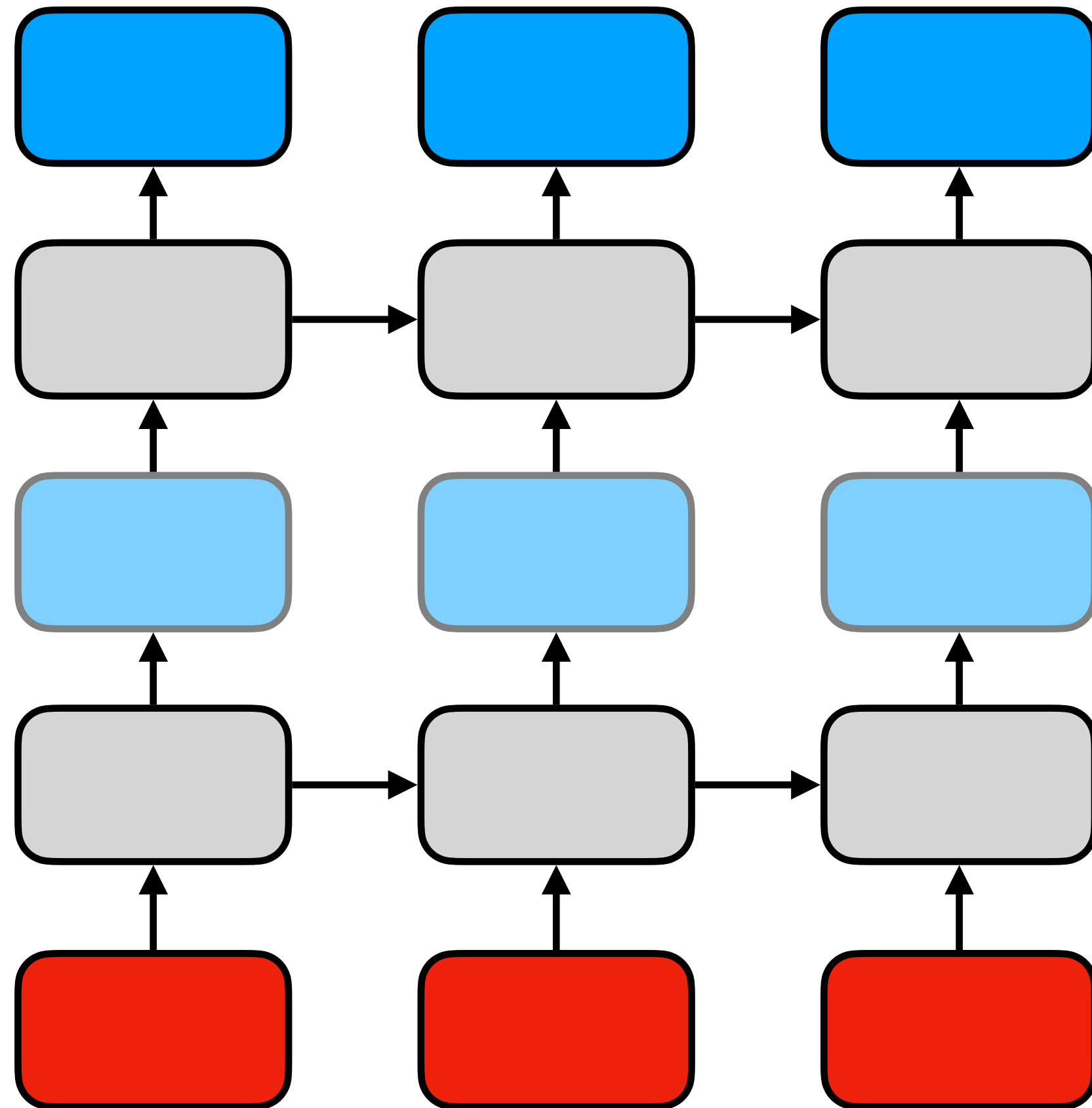
$$\frac{\partial L}{\partial W_{hh}} = \sum_t \frac{\partial L}{\partial h^{(t)}} \frac{\partial h^{(t)}}{\partial W_{hh}}$$

$$\frac{\partial L}{\partial h^{(t)}} = \frac{\partial L}{\partial h^{(t+1)}} \frac{\partial h^{(t+1)}}{\partial h^{(t)}} + \frac{\partial L}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial h^{(t)}}$$

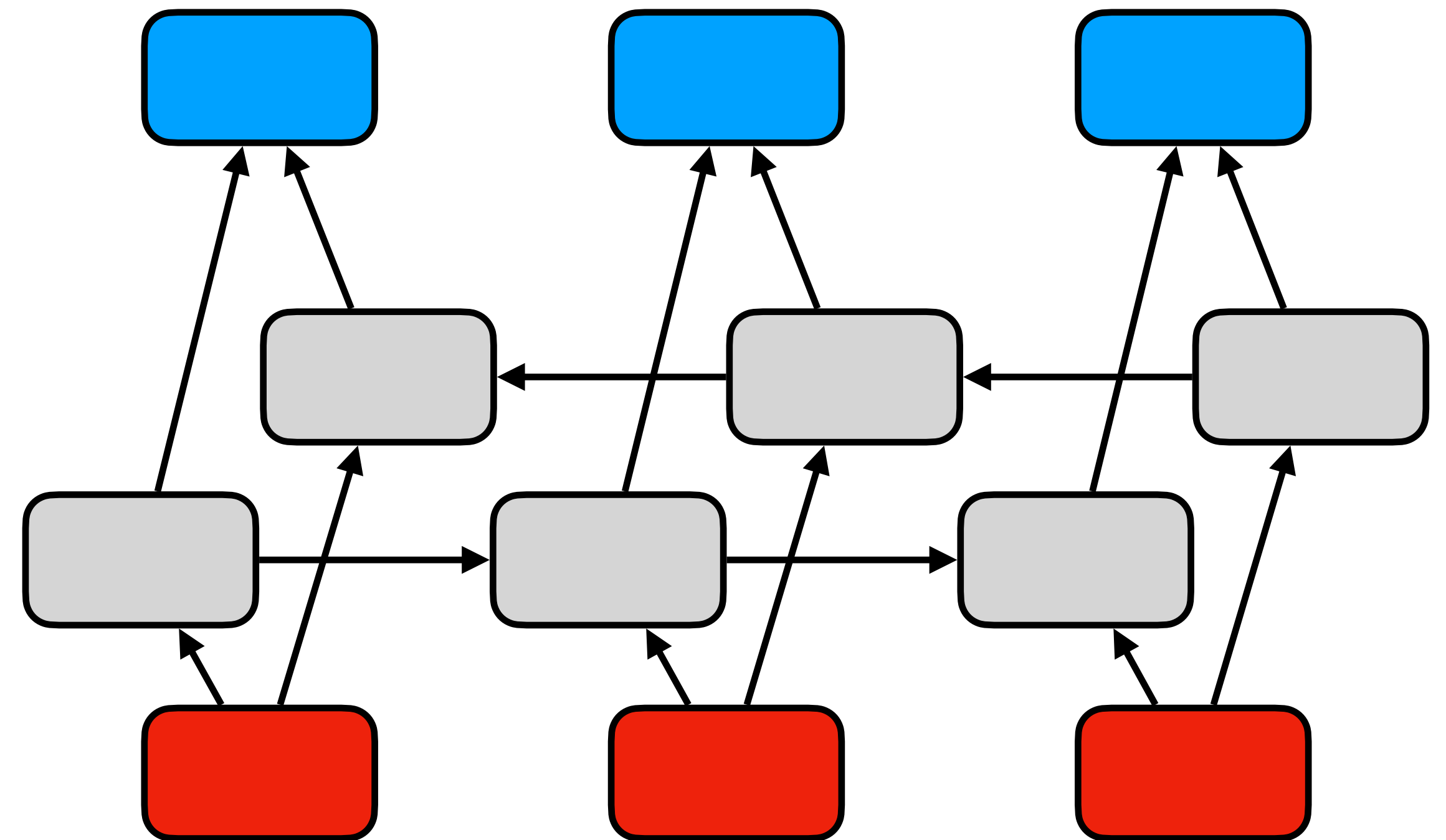
RNN training is **very** expensive...

More RNN formulations

Multi-layer RNN

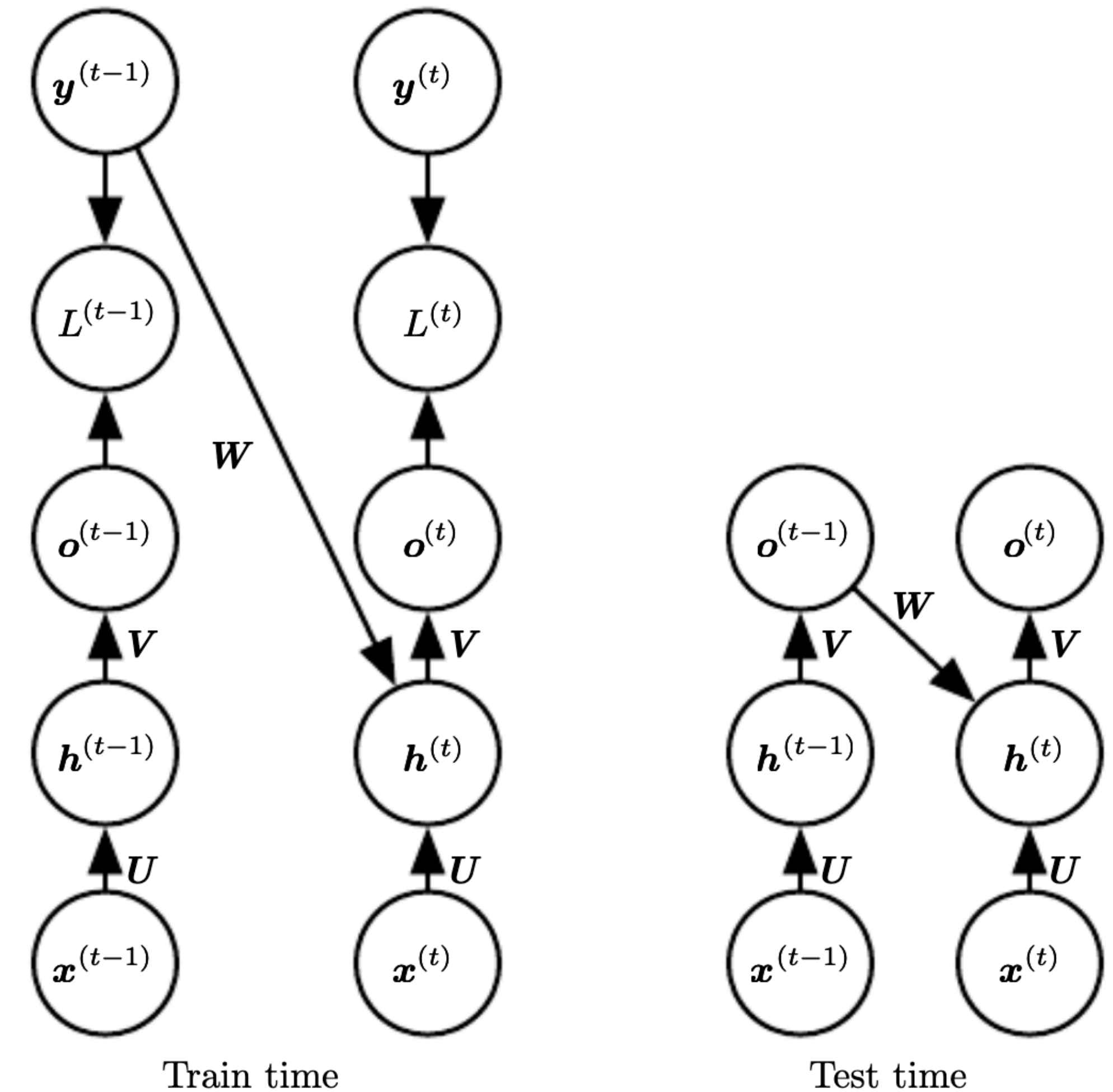


Bi-directional RNN



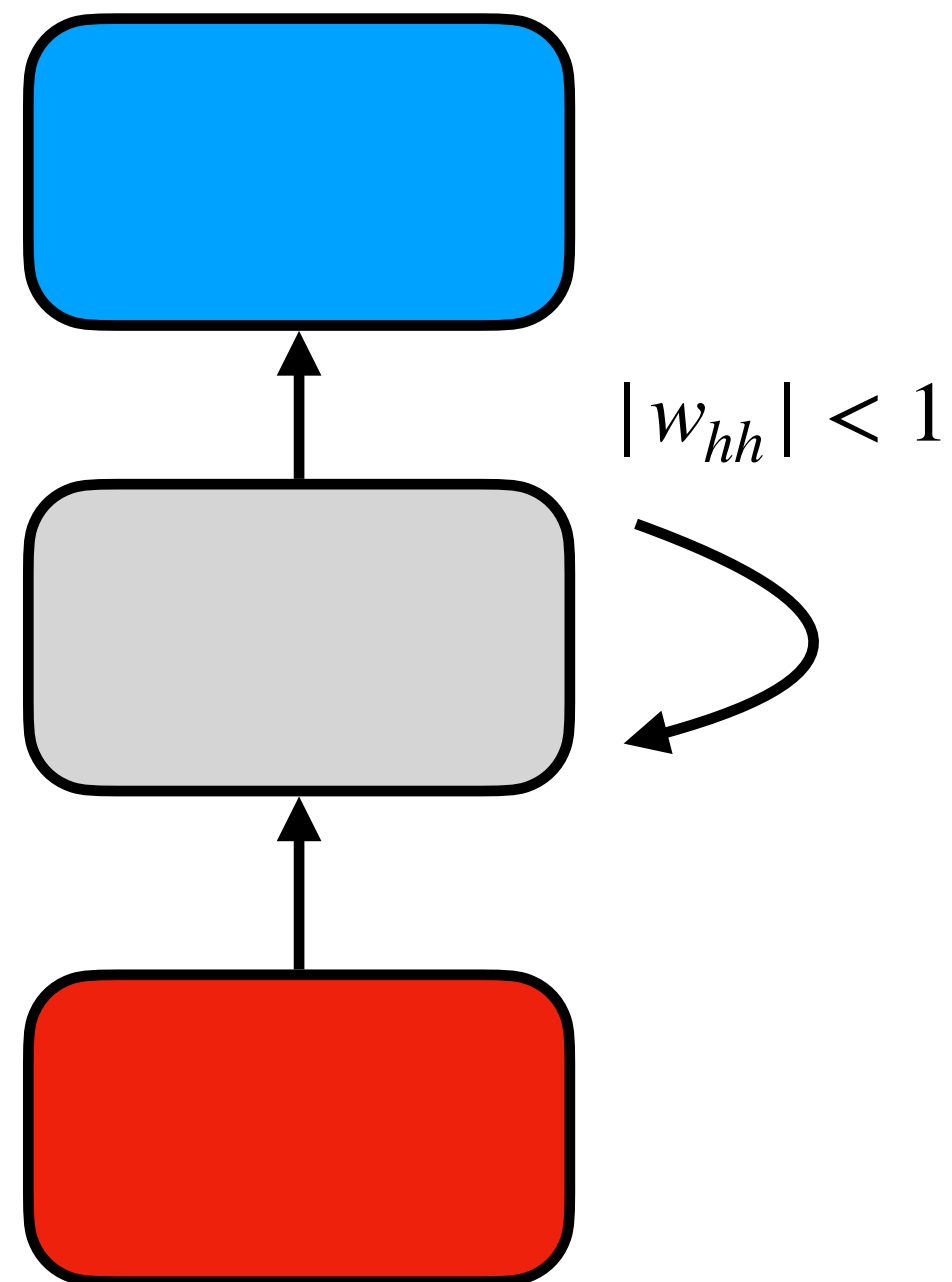
Teacher forcing: output-to-hidden recurrence

- Hidden-to-hidden training is slow because hidden states must be computed sequentially
- Output-to-hidden uses ground-truth values during training and model outputs during inference
- Gradients at each time step can be computed in parallel

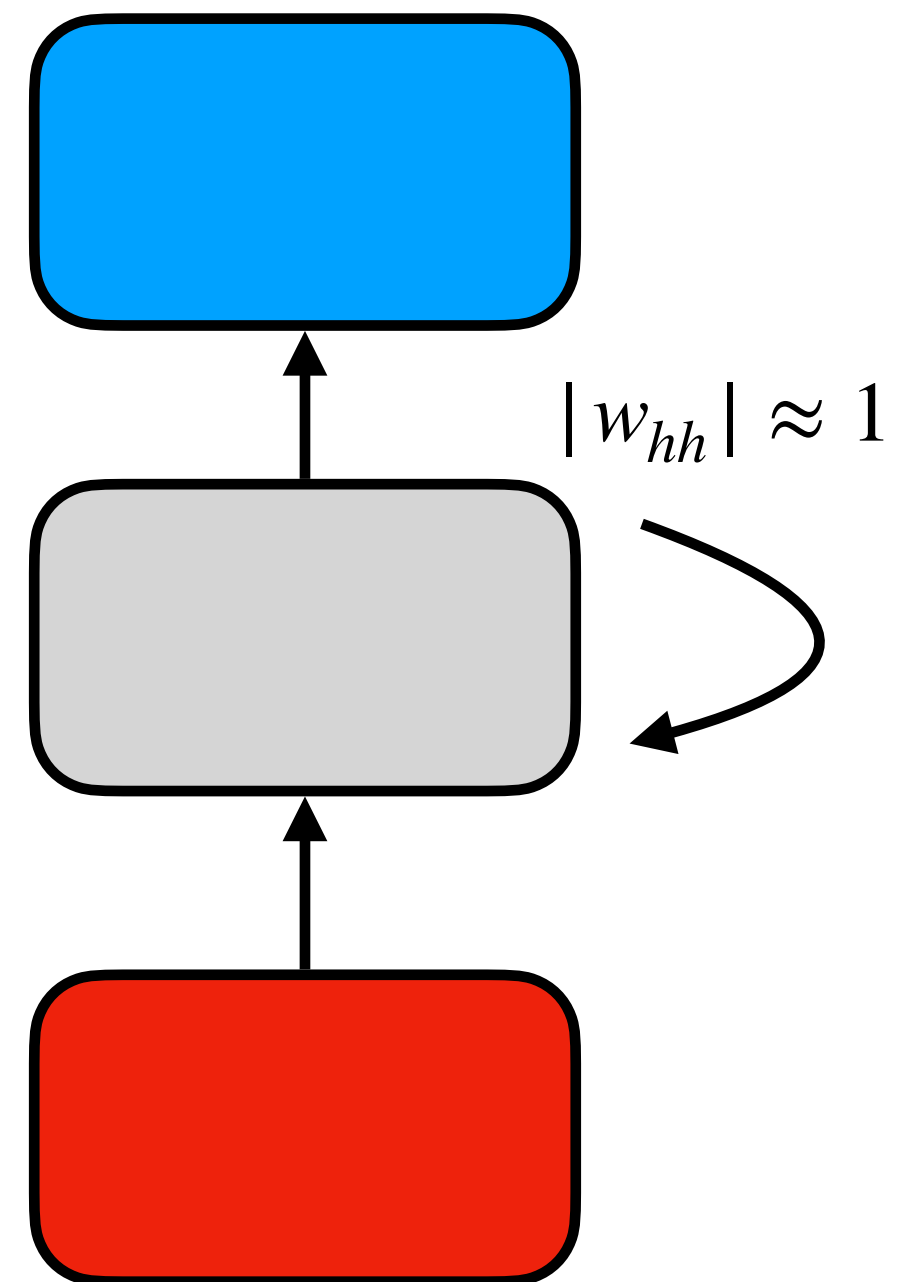


Challenges with long sequences

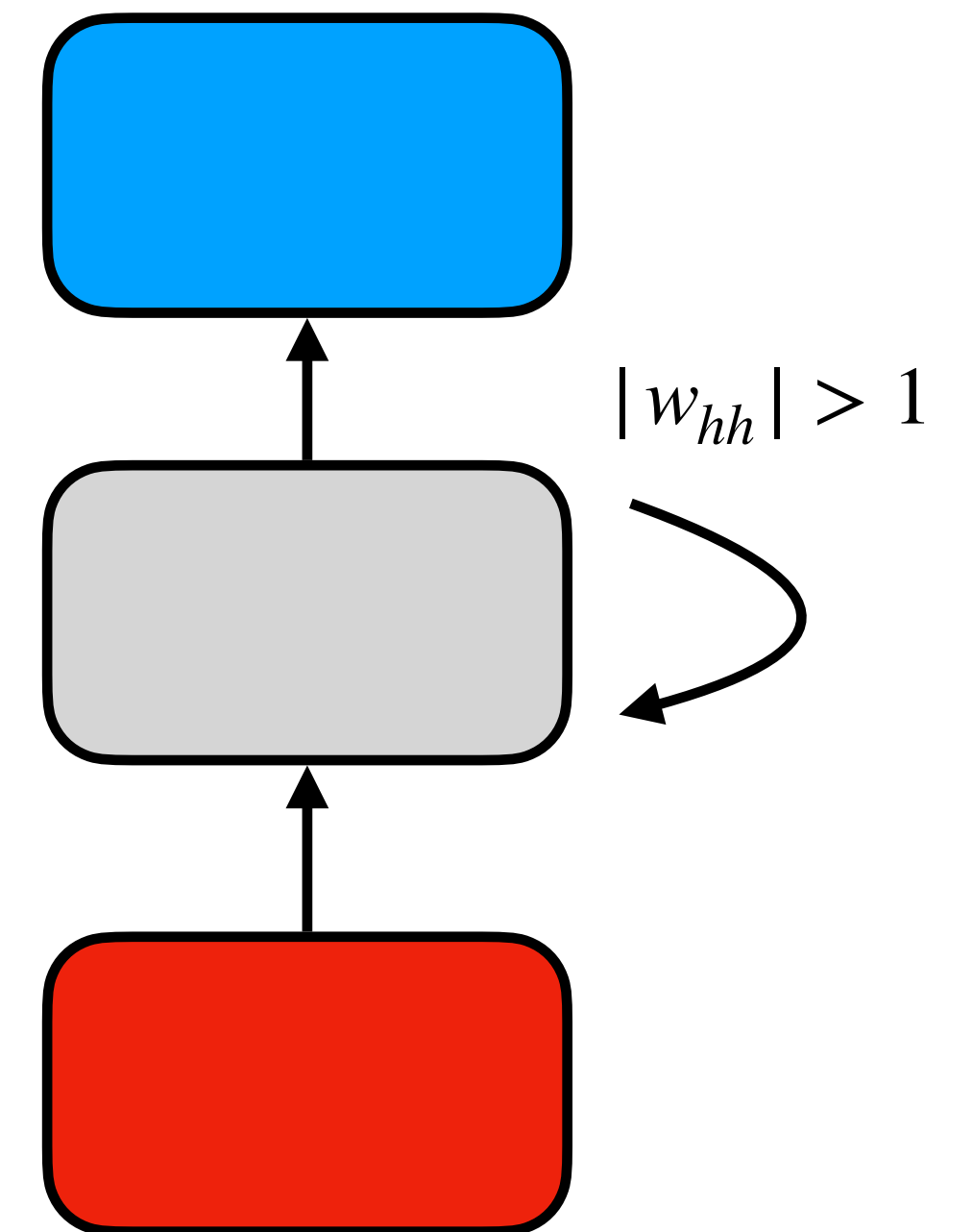
Vanishing gradient



Goldilocks zone



Exploding gradient



Strategies to handle vanishing/exploding gradients

Gradient clipping: `torch.nn.utils.clip_grad_norm_(model.parameters(), max_gradient_value)`

Truncated Backpropagation through time: don't backprop through the whole sequence

Long short-term memory (LSTM): gated connections allow the *model* to select whether information is propagated through time

Long Short Term Memory

Gating functions

$$i^{(t)} = \sigma \left(W_{ii}x^{(t)} + b_{ii} + W_{hi}h^{(t-1)} + b_{hi} \right)$$

$$f^{(t)} = \sigma \left(W_{if}x^{(t)} + b_{if} + W_{hf}h^{(t-1)} + b_{hf} \right)$$

$$o^{(t)} = \sigma \left(W_{io}x^{(t)} + b_{io} + W_{ho}h^{(t-1)} + b_{ho} \right)$$

Memory cell input

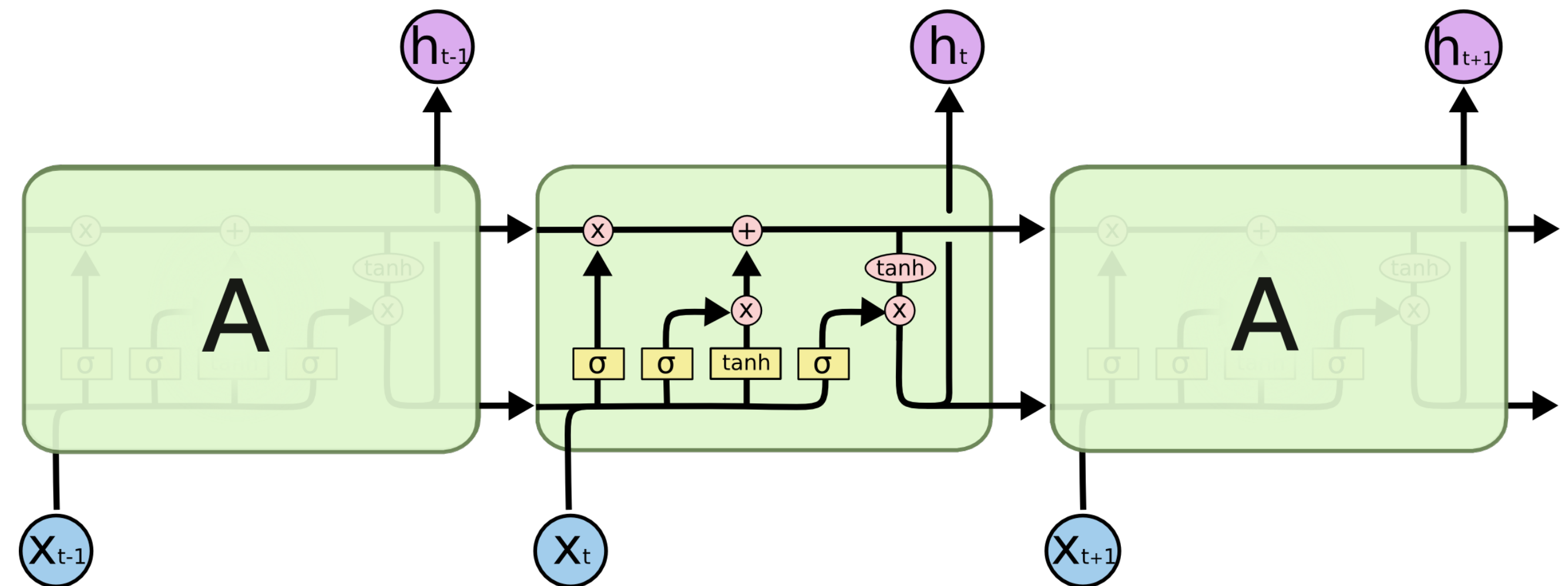
$$g^{(t)} = \tanh \left(W_{ig}x^{(t)} + b_{ig} + W_{hg}h^{(t-1)} + b_{hg} \right)$$

Memory cell update

$$c^{(t)} = f^{(t)}c^{(t-1)} + i^{(t)}g^{(t)}$$

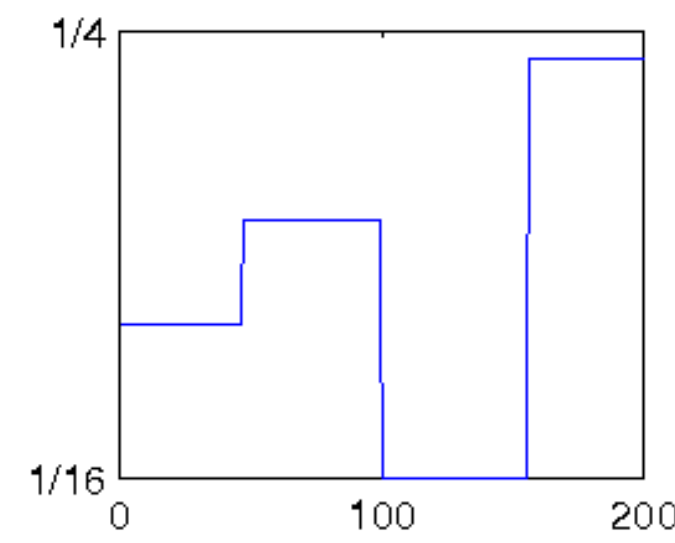
Hidden layer update

$$h^{(t)} = o^{(t)} \tanh c^{(t)}$$

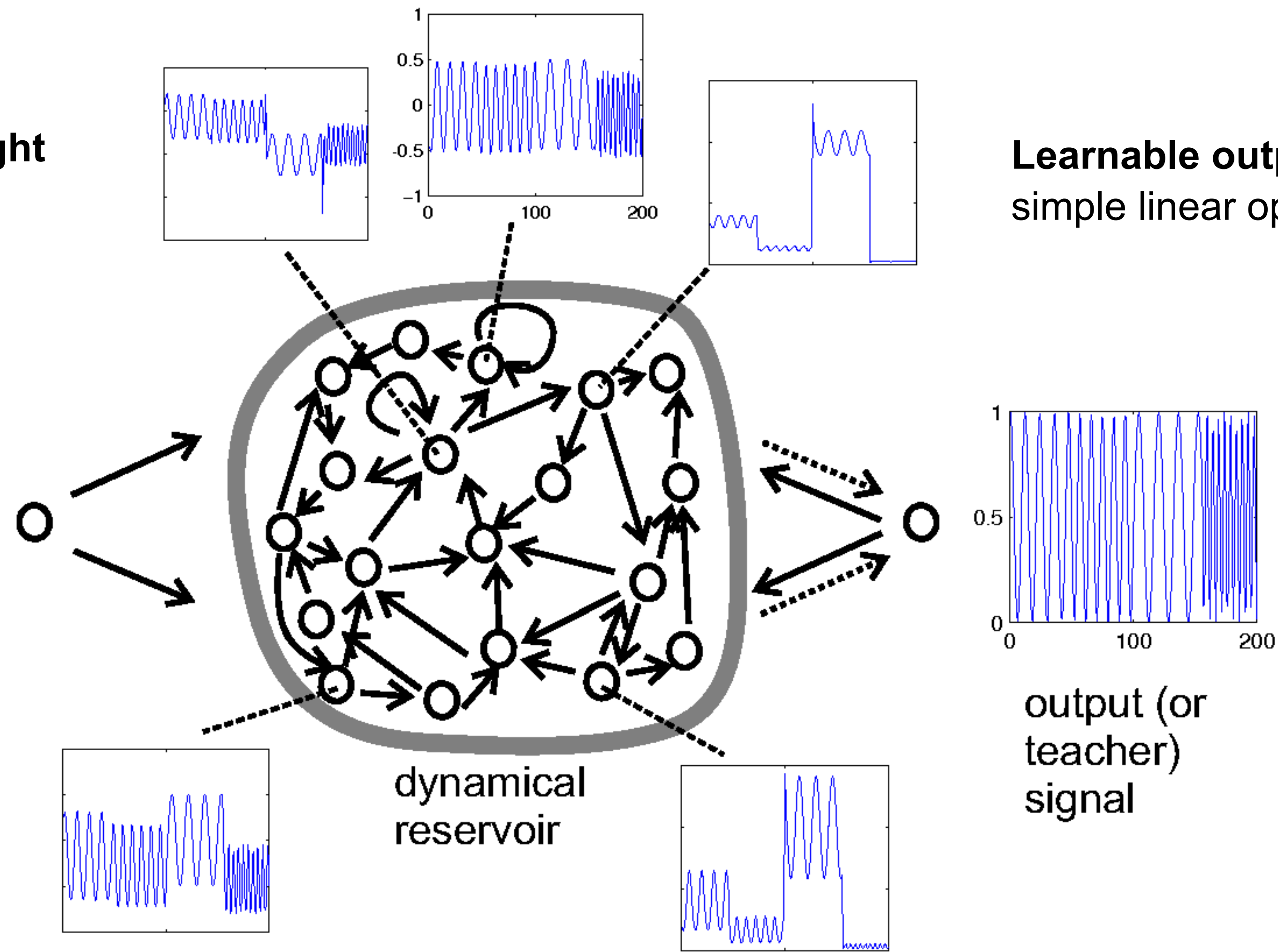


Reservoir computing and Echo State Networks

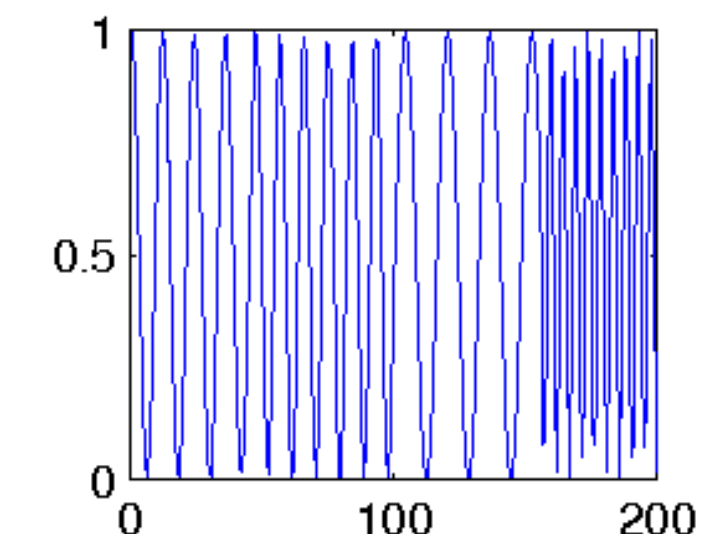
Fixed, sparsely-connected weight matrix W_{hh} with some desirable spectral properties



input signal



Learnable output matrix W_{oh} via simple linear optimization



output (or teacher) signal

https://en.wikipedia.org/wiki/Echo_state_network

That's all!

- Slides will be posted on the course website: <https://colen-teaching.github.io/DATA442-Fall-2026>
- Lab demos will also be posted
- **PS2** due October 23
- **REMEMBER:** TA Office Hours Tuesday 12-1pm, Student Success Center