Recurrent Networks

Tripp Deep Learning F23

With figures from Goodfellow et al., *Deep Learning*

UNIVERSITY OF WATERLOO

TODAY'S GOAL

By the end of the class, you will be familiar with recurrent neural networks as nonlinear discrete-time dynamical systems, how gradients are calculated in these networks, challenges in retaining long-term context, and the role of gating.

Summary

1. Recurrent networks are stateful
2. Recurrent networks can produce an output at each step or a summary output at the end of a sequence
3. Recurrent network gradients are found by backpropagation through time
4. The number of sequential steps normally scales with sequence length
5. Gains and nonlinearities compound with repeated application
6. Gating helps to retain long-term context

RECURRENT NETWORKS ARE STATEFUL

State

- Recall from dynamical systems theory that a system's "state" is the information which, along with the system's future inputs, determines its future outputs

- For example, in the continuous-time linear system,

  $$\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u}$$

  $$\boldsymbol{y} = C\boldsymbol{x} + D\boldsymbol{u}$$

  with input $\boldsymbol{u}$ and output $\boldsymbol{y}$, $\boldsymbol{x}$ is the state

- Note that $\boldsymbol{x}$ now affects $\boldsymbol{x}$ in the future, and that future $\boldsymbol{x}$ and $\boldsymbol{u}$ determine future $\boldsymbol{y}$

- In contrast, in the networks we have seen in previous lectures, future outputs are *entirely* determined by future inputs; those networks have no state

Recurrent networks are discrete-time nonlinear systems

A lumped time-invariant discrete-time nonlinear system can be written in state-space form as,

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{f}\left(\boldsymbol{x}^{(t)}, \boldsymbol{u}^{(t)}\right)$$

$$\boldsymbol{y}^{(t)} = \boldsymbol{g}\left(\boldsymbol{x}^{(t)}, \boldsymbol{u}^{(t)}\right)$$

where $\boldsymbol{u}^{(t)}$, $\boldsymbol{x}^{(t)}$ and $\boldsymbol{y}^{(t)}$ are the input, state, and output at time t, and $\boldsymbol{f}$ and $\boldsymbol{g}$ are nonlinear functions.

Deep networks are lumped, i.e., their state has finite dimension, corresponding to a finite number of hidden neurons. Deep networks are not time-invariant (their weights change slowly during learning), but for some purposes we can approximate them as such, e.g., their weights don’t change while processing a single sequence.

Recurrent networks are discrete-time nonlinear systems

- Dynamical systems theory and deep networks tend to use different variable names. In dynamical systems the input, state, and output are called $\boldsymbol{u}$, $\boldsymbol{x}$, and $\boldsymbol{y}$.
- In deep networks the input is often called $\boldsymbol{x}$, the state may be called $\boldsymbol{s}$ or $\boldsymbol{h}$ (for *state* or *hidden*), and the output may be called $\boldsymbol{o}$ or $\hat{\boldsymbol{y}}$.
- We may also consider a single recurrent block within a deep network as a dynamical system and call its inputs $\boldsymbol{h}_{i-1}$ and its outputs $\boldsymbol{h}_i$, or similar.

Recurrent networks are discrete-time nonlinear systems

Let’s rewrite the standard state equations using variables that make sense for a network with a single recurrent block that acts on the network input. We will call the input $\boldsymbol{x}$, the state $\boldsymbol{h}$, and the output $\boldsymbol{o}$:

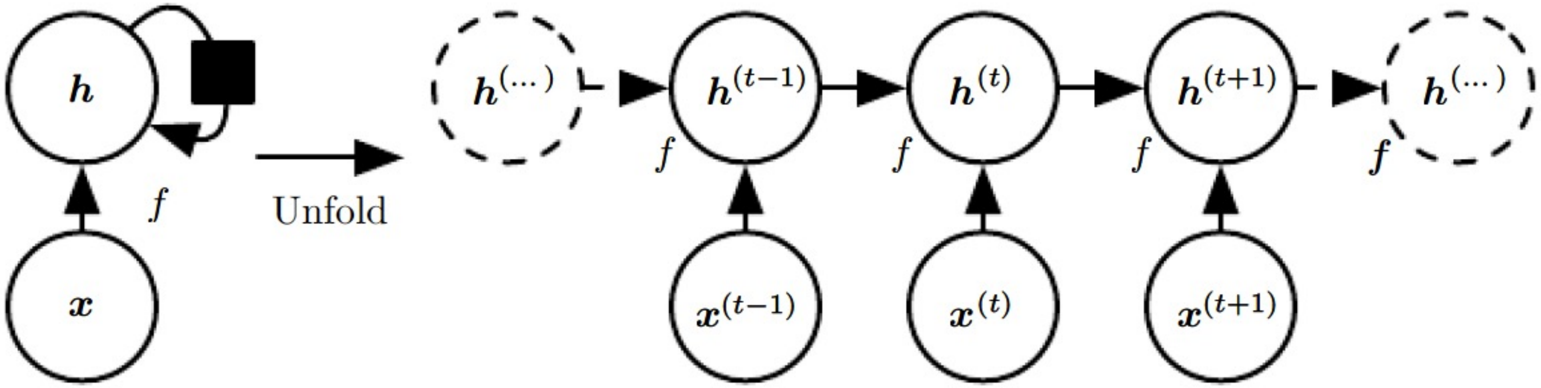
$$\boldsymbol{h}^{(t+1)} = \boldsymbol{f}\left(\boldsymbol{h}^{(t)}, \boldsymbol{x}^{(t)}\right)$$

$$\boldsymbol{o}^{(t)} = \boldsymbol{g}\left(\boldsymbol{h}^{(t)}, \boldsymbol{x}^{(t)}\right)$$

Whereas a continuous-time dynamical system reacts to an input signal $\boldsymbol{u}(t)$, this network processes a sequence $\boldsymbol{x}^{(1)} \dots \boldsymbol{x}^{(\tau)}$, where τ is the sequence length.

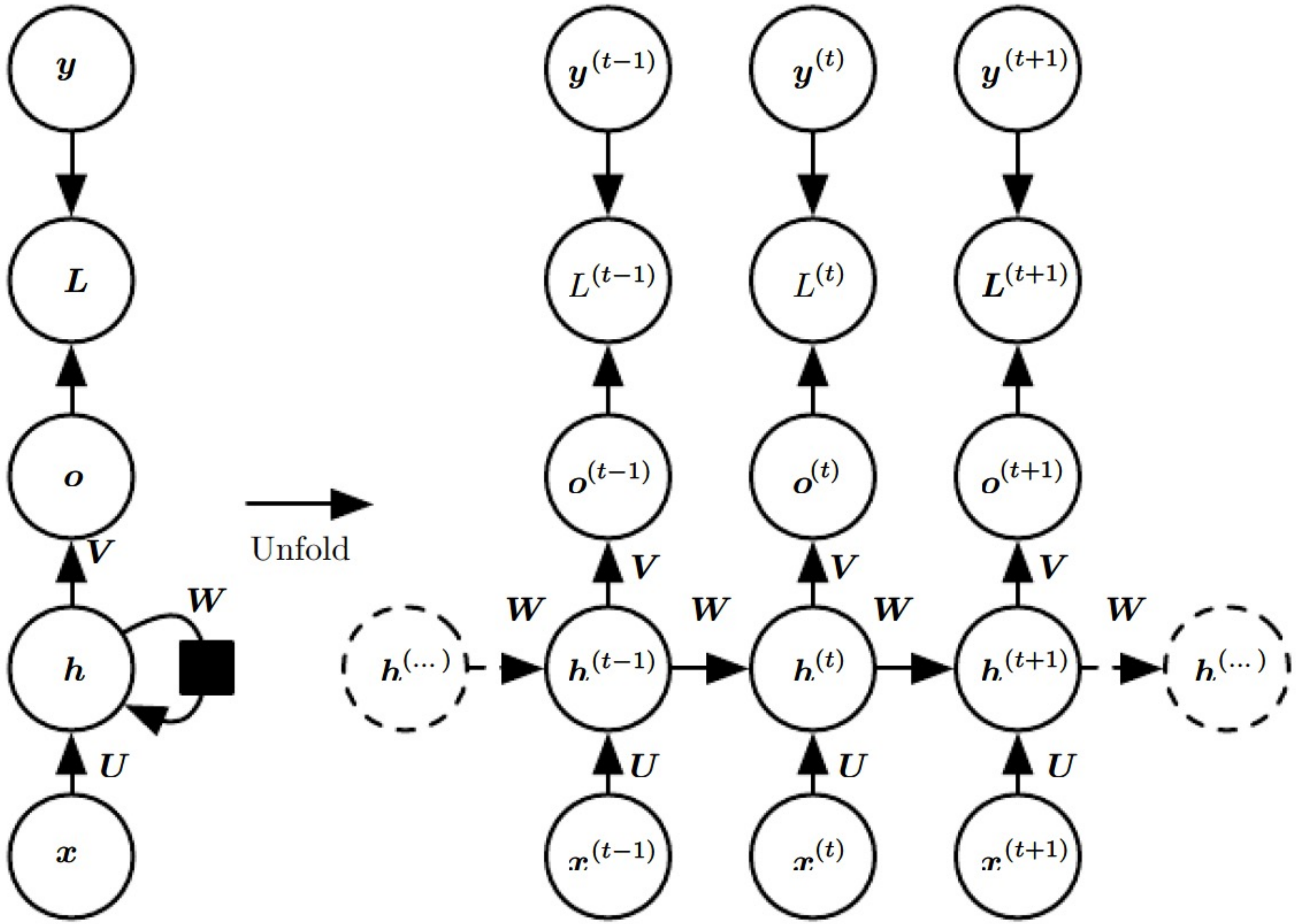
Different ways to draw recurrent neural networks

- Recurrent networks are usually diagrammed in one of two ways, both shown below
  - With a loop (left); here the square means a delay of one timestep, i.e., $\boldsymbol{h}^{(t)} = \boldsymbol{f}\left(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}\right)$
  - Unfolded in time, with the input, state, etc. drawn separately for each timestep (right)



RECURRENT NETWORKS CAN PRODUCE AN OUTPUT AT EACH STEP OR A SUMMARY OUTPUT AT THE END OF A SEQUENCE

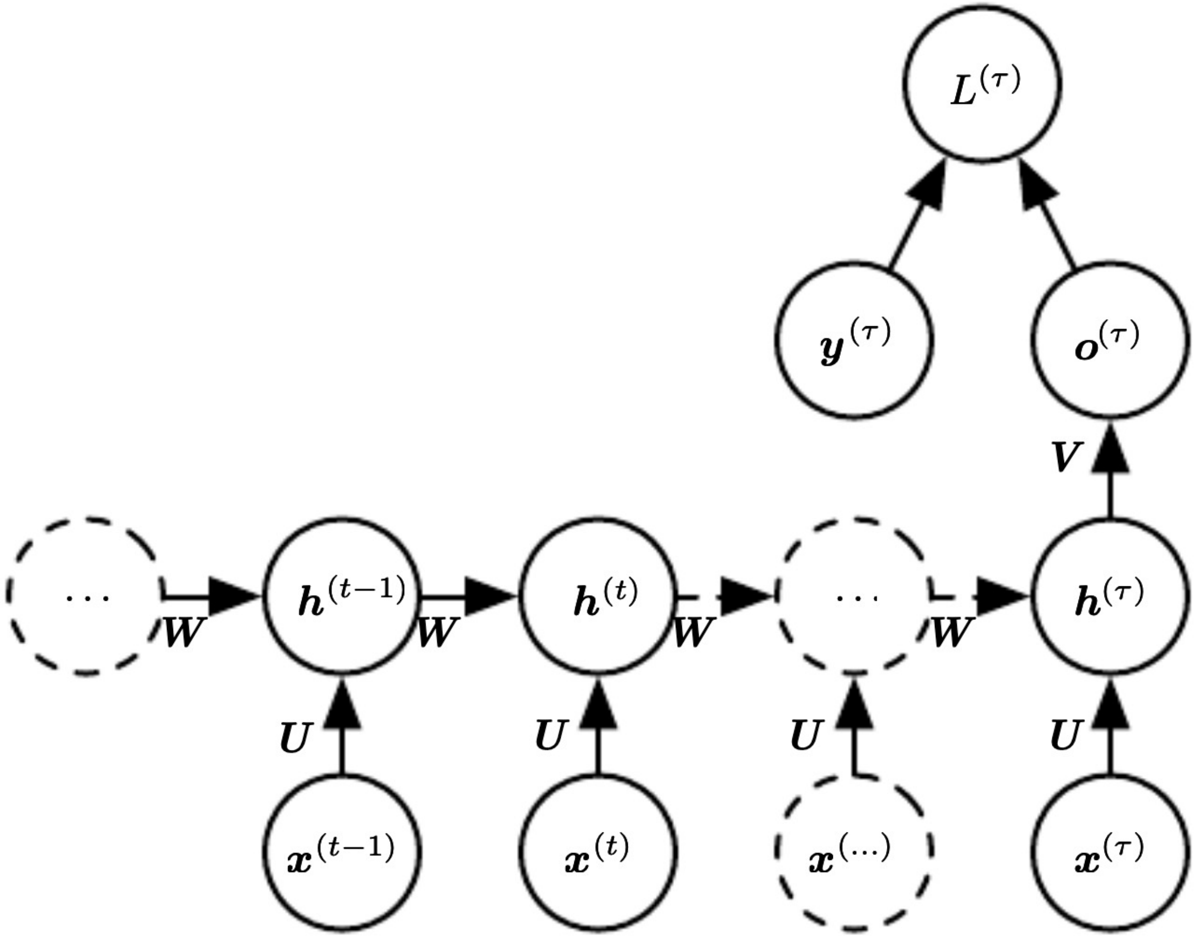
Network with output at each step



Network with output at end of sequence

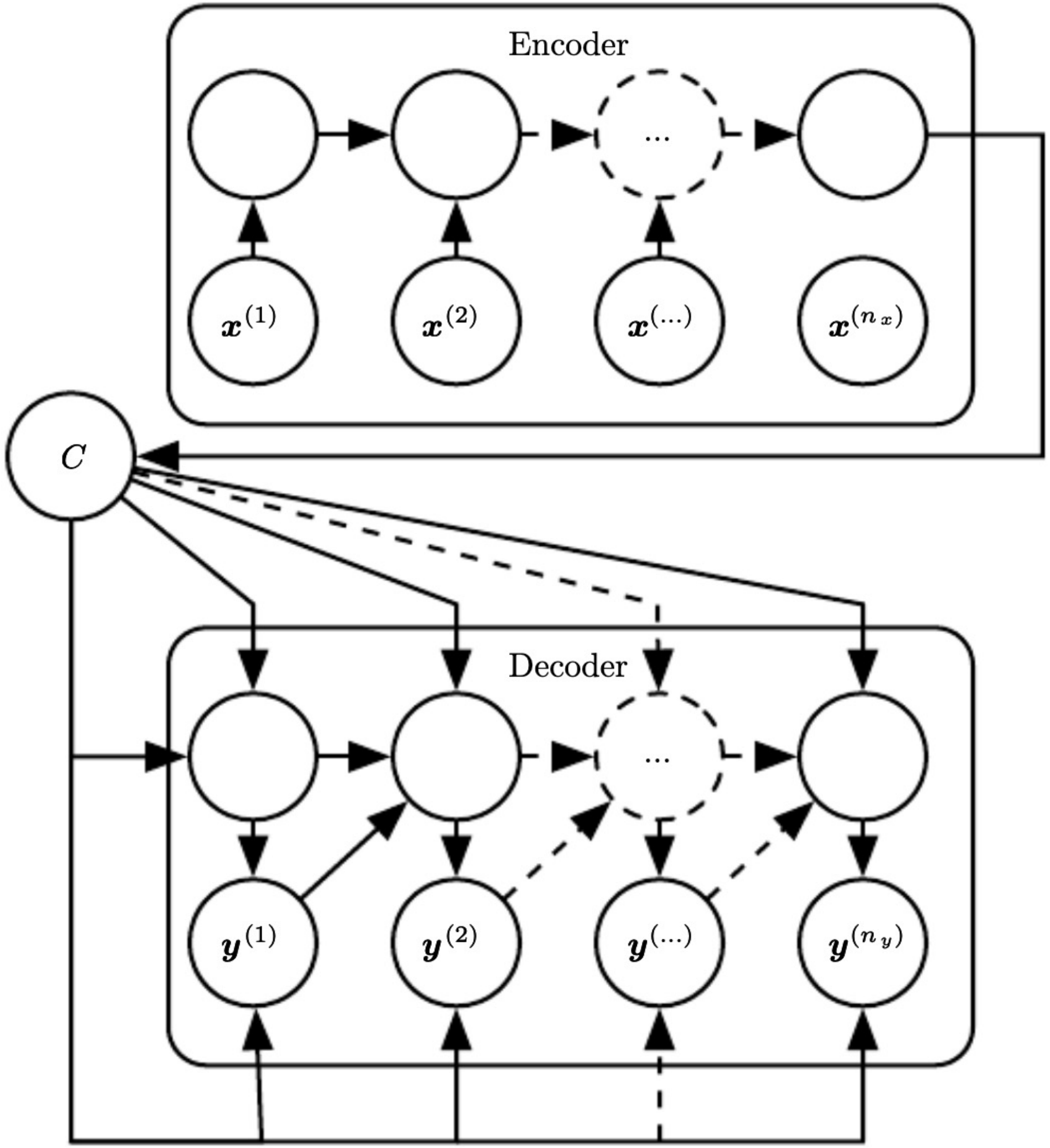
- In this design, information about the sequence is compressed into a single output vector $\boldsymbol{o}^{(\tau)}$

- Unless the state and the output have much higher dimensions than the input elements, $\boldsymbol{o}^{(\tau)}$ is a compressed representation of the sequence



Encoder-decoder architectures

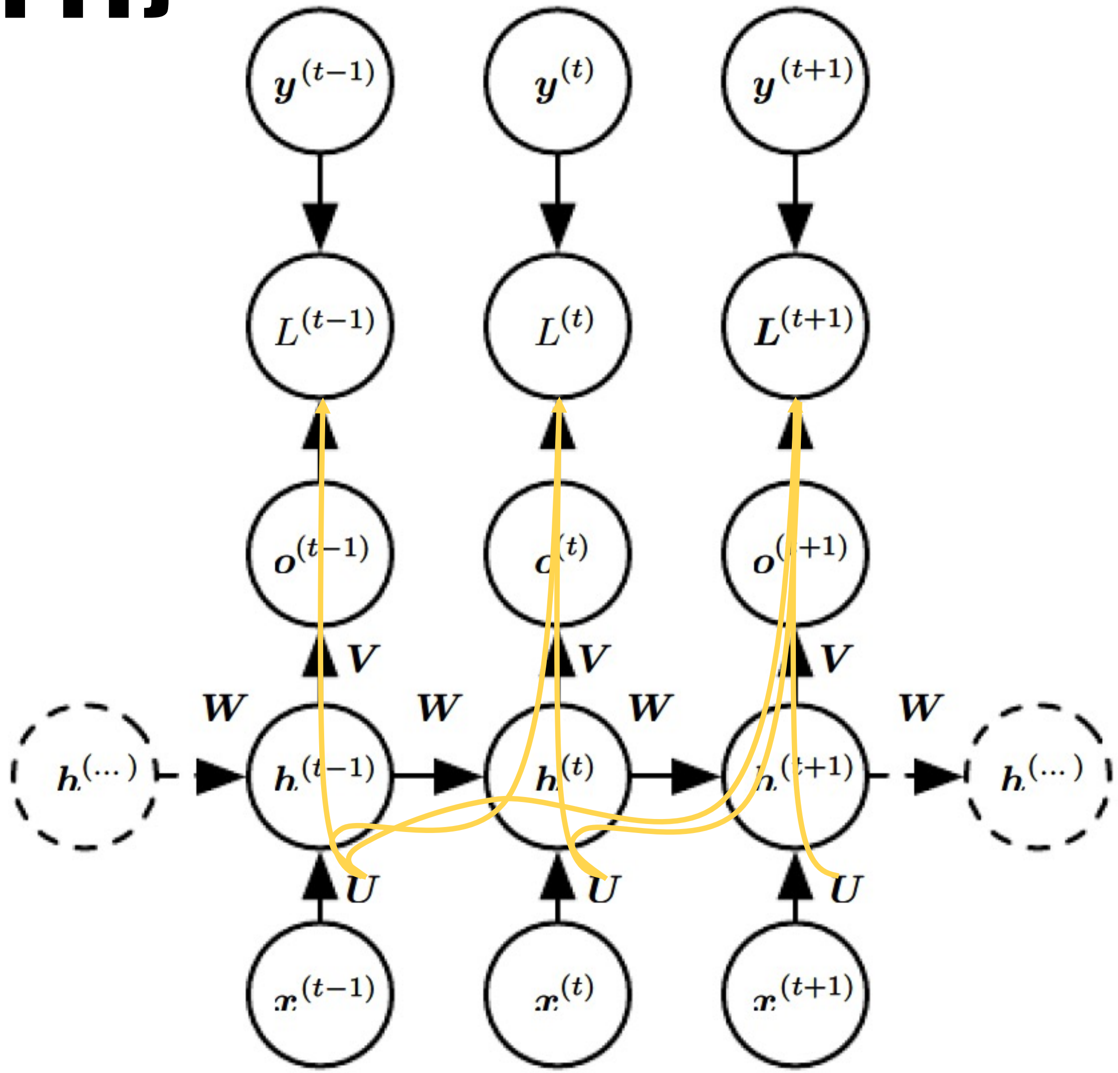
- In an encoder-decoder RNN
  - The encoder produces a vector based on the input sequence (here called C rather than $\boldsymbol{o}^{(\tau)}$, since it it not used as an output)
  - The decoder produces a new sequence based on C
  - C can be provided to the decoder as an initial state, or as constant input to the state at each step, or both (as shown here)
- Note that the lengths of the input and output sequences can be different



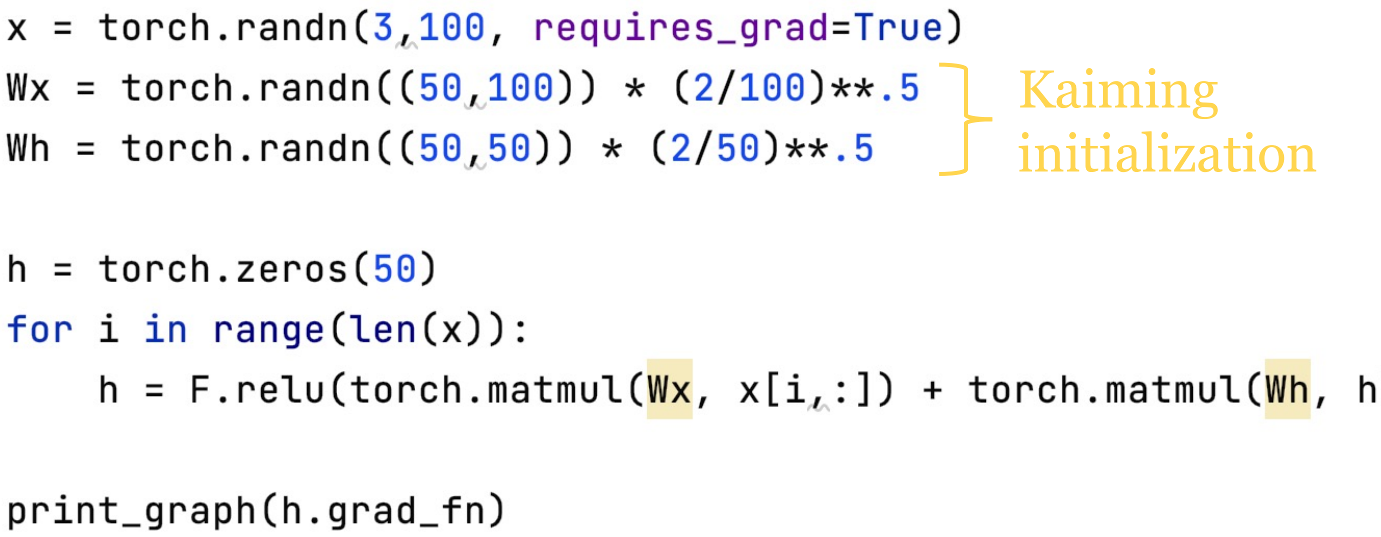
RECURRENT NETWORK GRADIENTS ARE FOUND BY BACKPROPAGATION THROUGH TIME

Backpropagation through time (BPTT)

- This is the same as regular backpropagation
- In a recurrent network, some inputs to a node may come from previous time steps
- This doesn't change anything, but the name sounds cool
- Be aware that a parameter may affect the loss through many paths because:
    - It is re-used in each timestep, and
    - It affects the loss in future timesteps



A recurrent graph in PyTorch

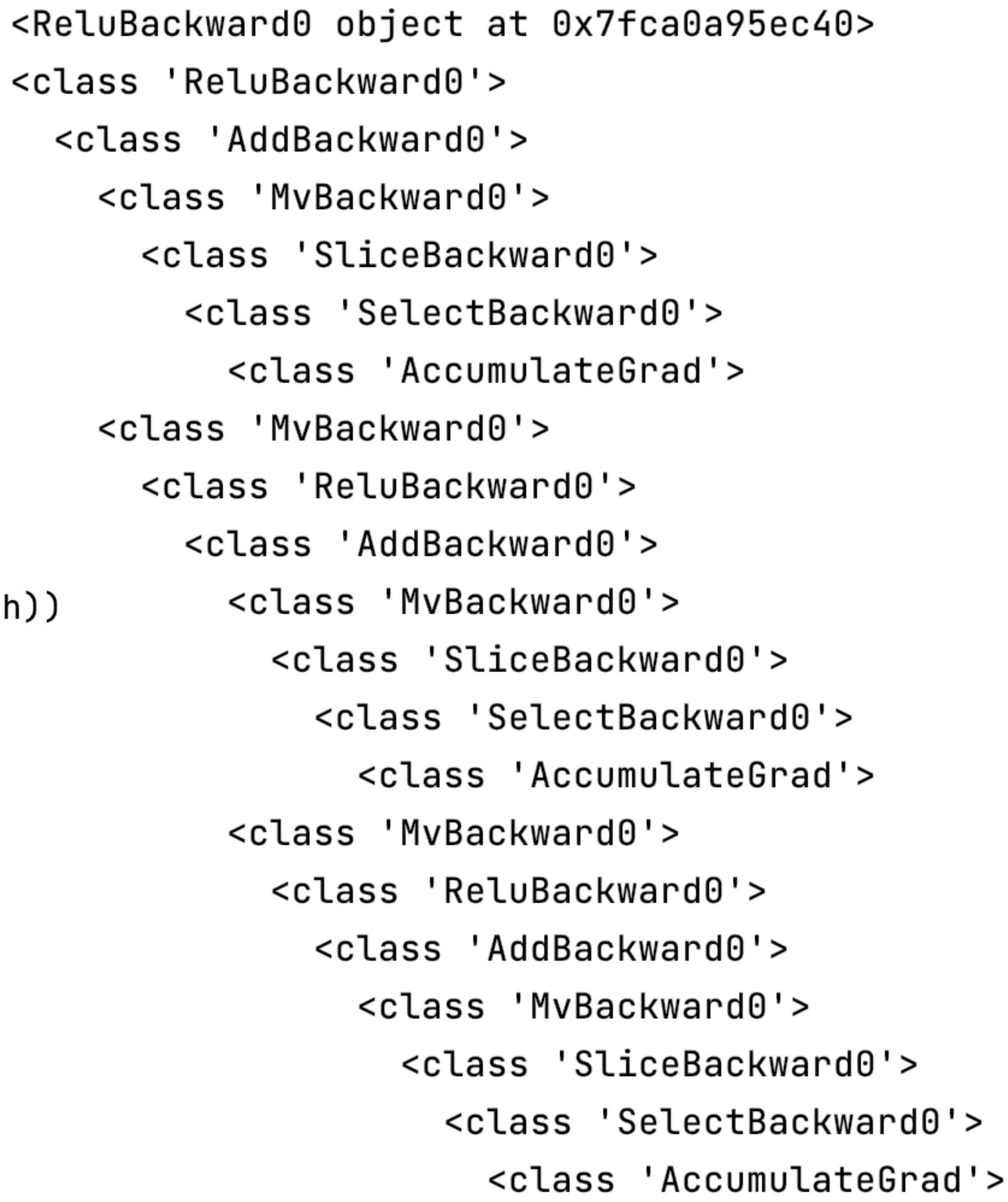


```
x = torch.randn(3,100, requires_grad=True)
Wx = torch.randn((50,100)) * (2/100)**.5
Wh = torch.randn((50,50)) * (2/50)**.5

h = torch.zeros(50)
for i in range(len(x)):
    h = F.relu(torch.matmul(Wx, x[i,:]) + torch.matmul(Wh, h))

print_graph(h.grad_fn)
```

Kaiming initialization

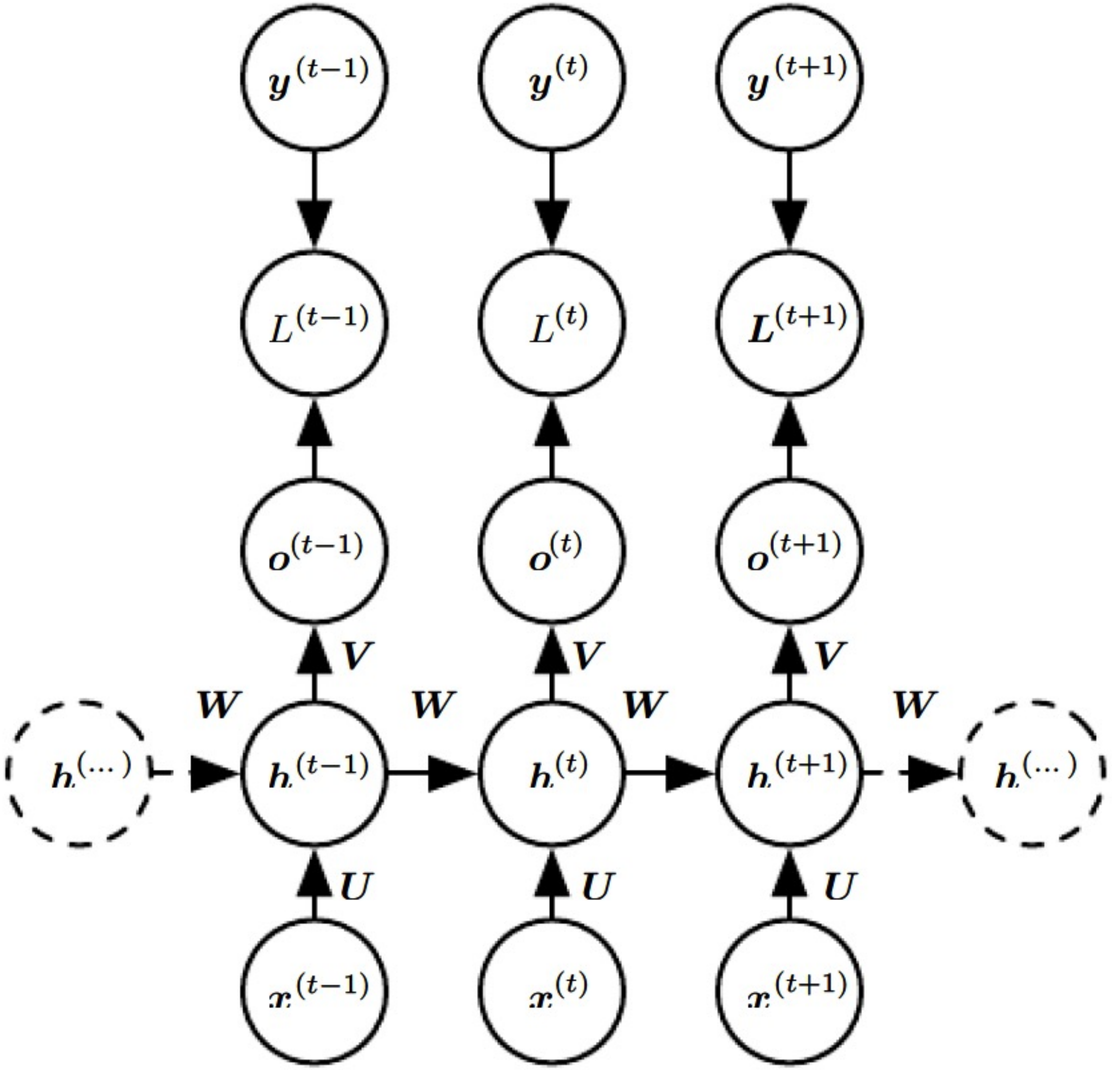


```
<ReluBackward0 object at 0x7fca0a95ec40>
<class 'ReluBackward0'>
  <class 'AddBackward0'>
    <class 'MvBackward0'>
      <class 'SliceBackward0'>
        <class 'SelectBackward0'>
          <class 'AccumulateGrad'>
    <class 'MvBackward0'>
      <class 'ReluBackward0'>
        <class 'AddBackward0'>
          <class 'MvBackward0'>
            <class 'SliceBackward0'>
              <class 'SelectBackward0'>
                <class 'AccumulateGrad'>
          <class 'MvBackward0'>
            <class 'ReluBackward0'>
              <class 'AddBackward0'>
                <class 'MvBackward0'>
                  <class 'SliceBackward0'>
                    <class 'SelectBackward0'>
                      <class 'AccumulateGrad'>
```

THE NUMBER OF SEQUENTIAL STEPS NORMALLY SCALES WITH SEQUENCE LENGTH

Serial processing

- The state at time t depends on the state at time $t\ -1$ etc., back to the beginning of the sequence, and on the corresponding inputs
- So, the network must run for each input before it runs for the next input
- In contrast with convolutional networks and transformers, the minimum number of sequential computational steps grows with the sequence length

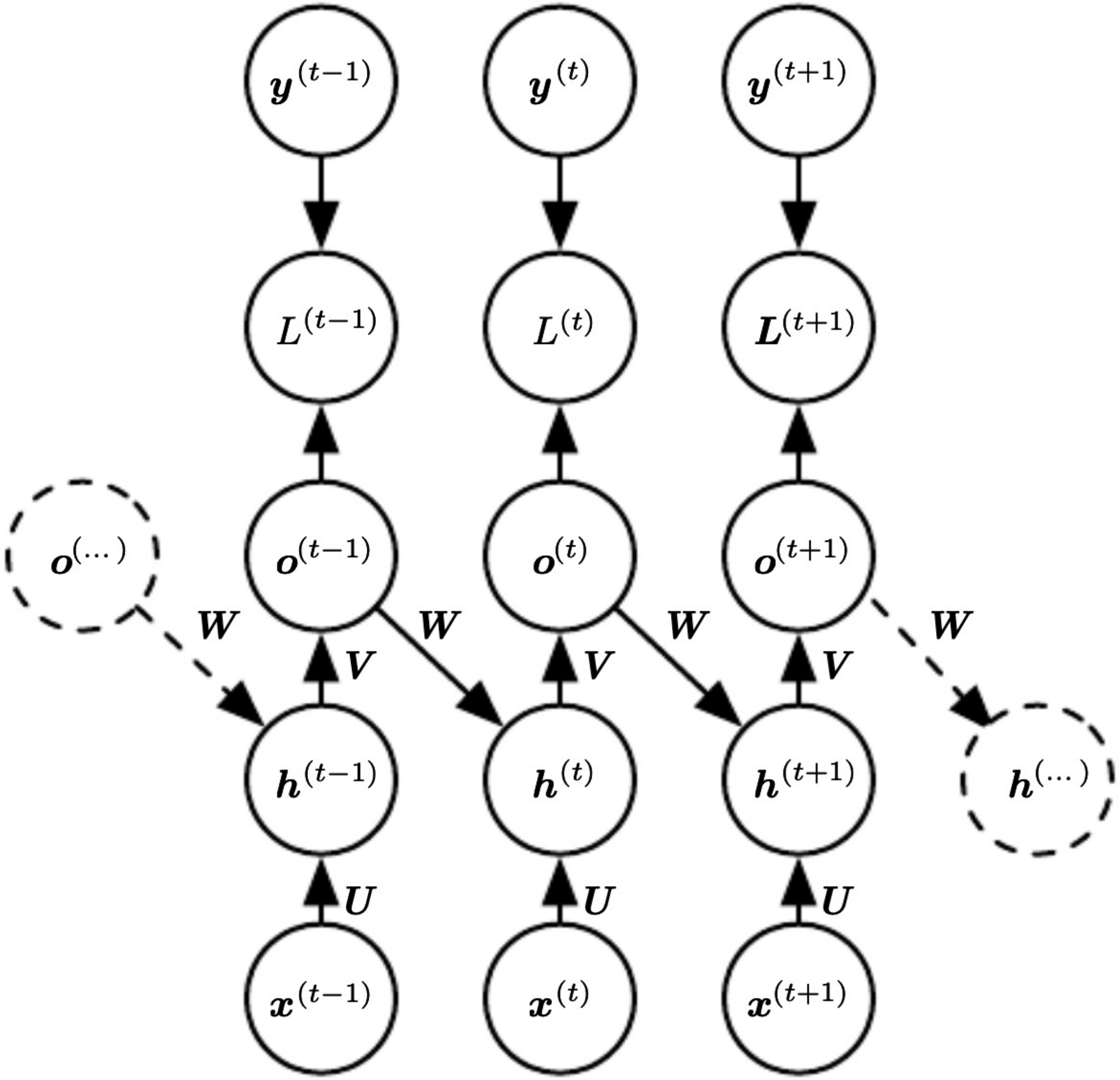


The exception to this rule

- A network can process different sequence elements in parallel during *training* if:
  - It uses teacher forcing, and
  - It only has feedback through its outputs
- *Inference* with such a network still requires sequential processing

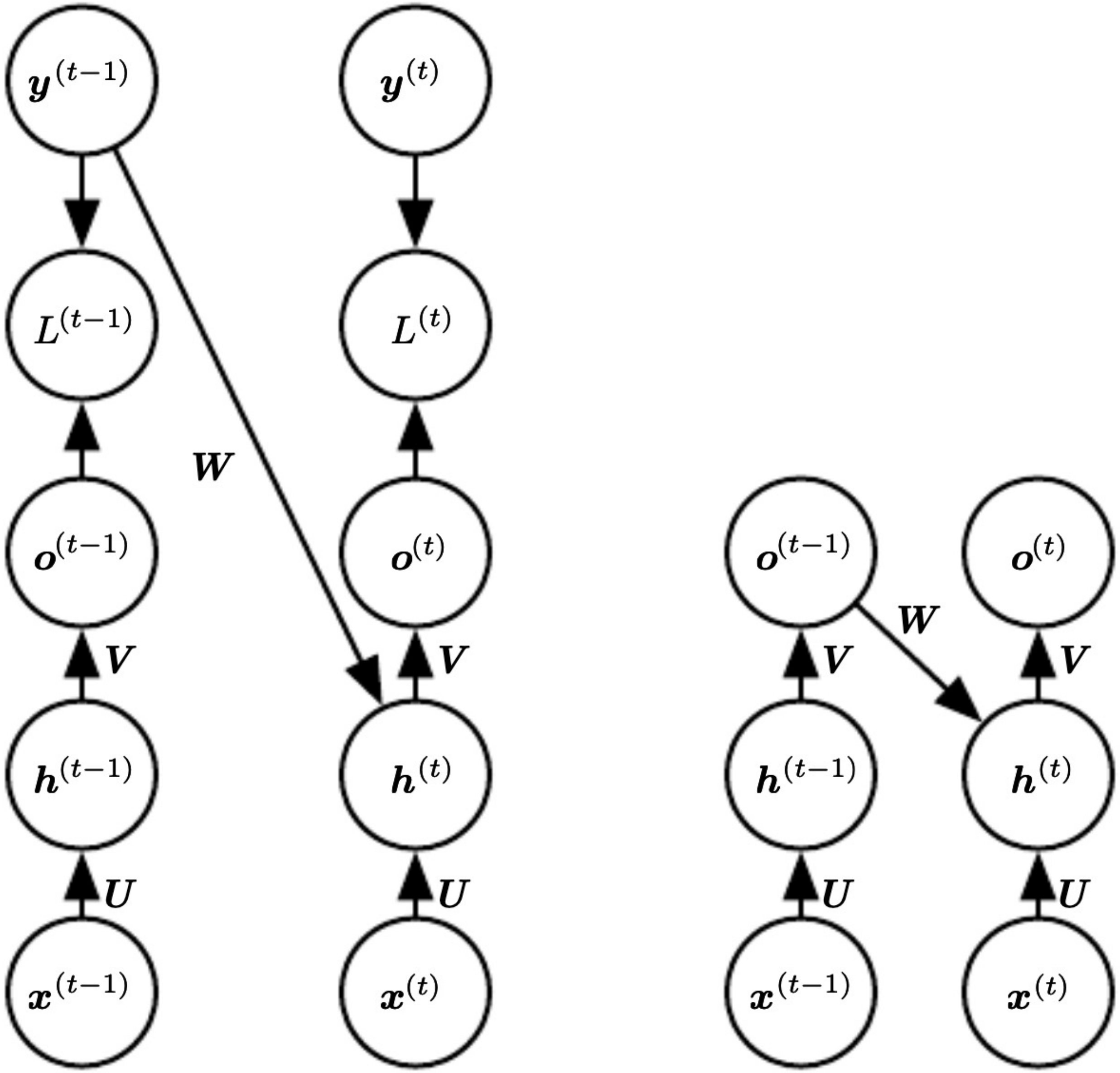
Feedback only through the output

- Some recurrent networks do not connect state directly to state in the next step
- Instead, the network only receives its own output from the previous step
- This design is less powerful than the standard one
    - To the extent that the output ***o*** approaches the target ***y***, it may not contain all the information needed for future outputs



Teacher forcing

- Networks with output feedback can be trained with teacher forcing
- This involves giving the network the *correct* output of the previous step, rather than its own output
- This simplifies learning, but it can also introduce a distribution shift between training and inference
- This can be mitigated by giving the network its own output more and more often as training progresses



Train time

Test time

GAINS AND NONLINEARITIES COMPOUND WITH REPEATED APPLICATION

Repeated nonlinearities

- A recurrent layer that runs for several timesteps can result in a highly nonlinear function of the input

- E.g., suppose there is a constant input $\boldsymbol{x}$, an element-wise nonlinearity $\boldsymbol{g}(\cdot)$, and the state $\boldsymbol{h}$ is,

$$\begin{aligned}\boldsymbol{h}^{(t)} &= \boldsymbol{g}\left(A\boldsymbol{h}^{(t-1)} + B\boldsymbol{x}^{t} + \boldsymbol{b}\right)\\ &= \boldsymbol{g}(A(\boldsymbol{g}(A\boldsymbol{h}^{(t-2)} + B\boldsymbol{x}^{t-1} + \boldsymbol{b})) + B\boldsymbol{x}^{t} + \boldsymbol{b})\\ &= \cdots\end{aligned}$$

- Repeated application of the nonlinearity can make $\boldsymbol{h}^{(t)}$ a complex and highly nonlinear function of $\boldsymbol{x}^{0}$ after a few steps

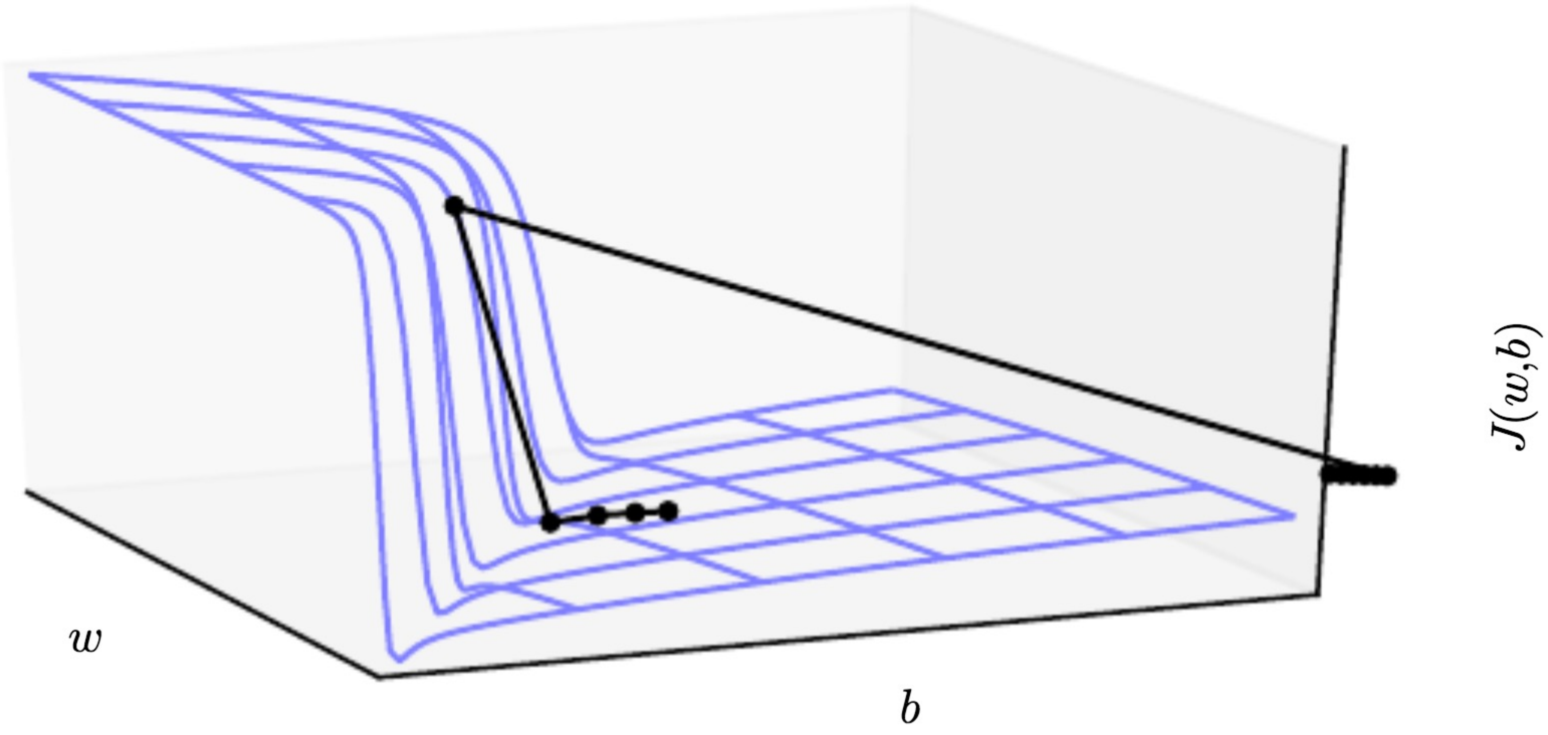
Repeated application of large gains

- The state might also grow over time due to large recurrent weights
  - Analogous to an unstable dynamical system
  - Maybe not indefinitely (e.g., it could approach a fixed point or a limit cycle)

Gradient clipping

- Because recurrent networks can exhibit severe nonlinearities and large gains, gradient cliffs are an issue

- As we discussed previously, gradient clipping can mitigate this issue

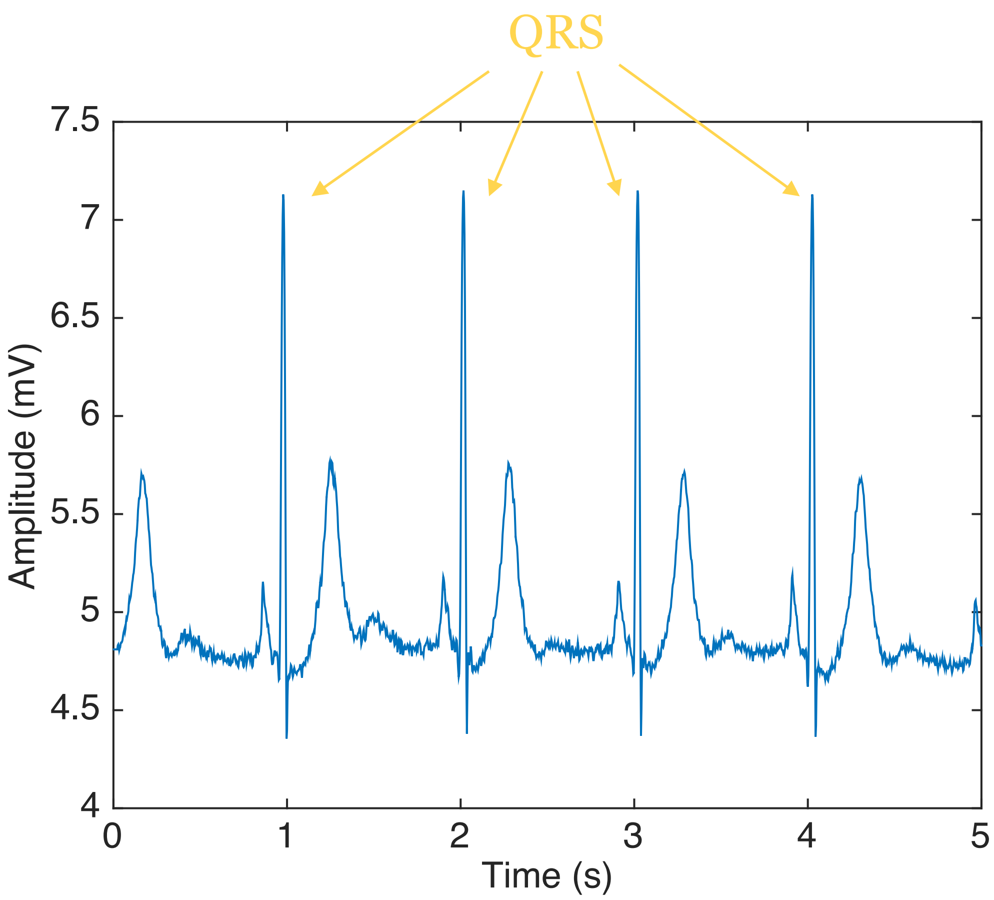


GATING HELPS TO RETAIN LONG-TERM CONTEXT

Good performance may require long-term context

- Example: Estimating heart rate from ECG
  - The network would need short-term context (maybe within 50ms) to reliably identify QRS complex peaks
  - The network would need longer-term context (perhaps >1s) to calculate the time between QRS complexes
- Example: Understanding language
  - "The high temperature at which 300-series steel transitions to plastic deformation would eliminate the need for a heat shield on **Starship**'s leeward side, while the much hotter windward side would be cooled by allowing fuel or water to bleed through micropores in a double-wall stainless steel skin, removing heat by evaporation. However, in July 2019, Musk indicated on Twitter that this would probably not be pursued, instead, **it** would use reusable heat shield tiles … "

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



There are several ways to retain long-term context

- Wide range of time constants
    - It can be arranged that some units' activities decay quickly, emphasizing short-term context, while other units' activities decay slowly, integrating information from a longer period
- Skip connections in time
    - Instead of only having recurrent connections that add state from the immediately previous time step, additional connections can add state from a few time steps back, improving longer-term retention of information

Gating can adapt time constants as needed

- Although it may be necessary to store information for many time steps, the information may suddenly become useless once it has been used
- A large time constant will cause the information to be retained regardless of whether it is still needed, perhaps interfering with storage of new information
- In a "gated" recurrent connection, the time constant is controllable:

$$s^{(t)} = \sigma(c)s^{(t-1)} + x^{(t)},$$

where $\sigma(c) \in [0,1]$ is a logistic sigmoid function of some contextual signal c

- As $\sigma(c) \to 1$, the state is retained indefinitely; as $\sigma(c) \to 0$, it is discarded
- A network with learnable c can learn to retain and forget as needed for the task

Long short-term memory networks (LSTMs)

- LSTM networks were introduced by Hochreiter & Schmidhuber (1997) and remain effective for many problems
- An LSTM block has multiple sigmoid gates per element of state s_i
  - Forget gate (f_i): Controls time constant (as in the last slide)
  - Input gate (g_i): Controls entry of inputs into the state
  - Output gate (o_i): Controls passage of state to output
- Each gate is a logistic sigmoid function of affine-transformed inputs and outputs (and sometimes states are included as well), e.g., the i^{th} forget gate output at time t is,

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right)$$

LSTM block

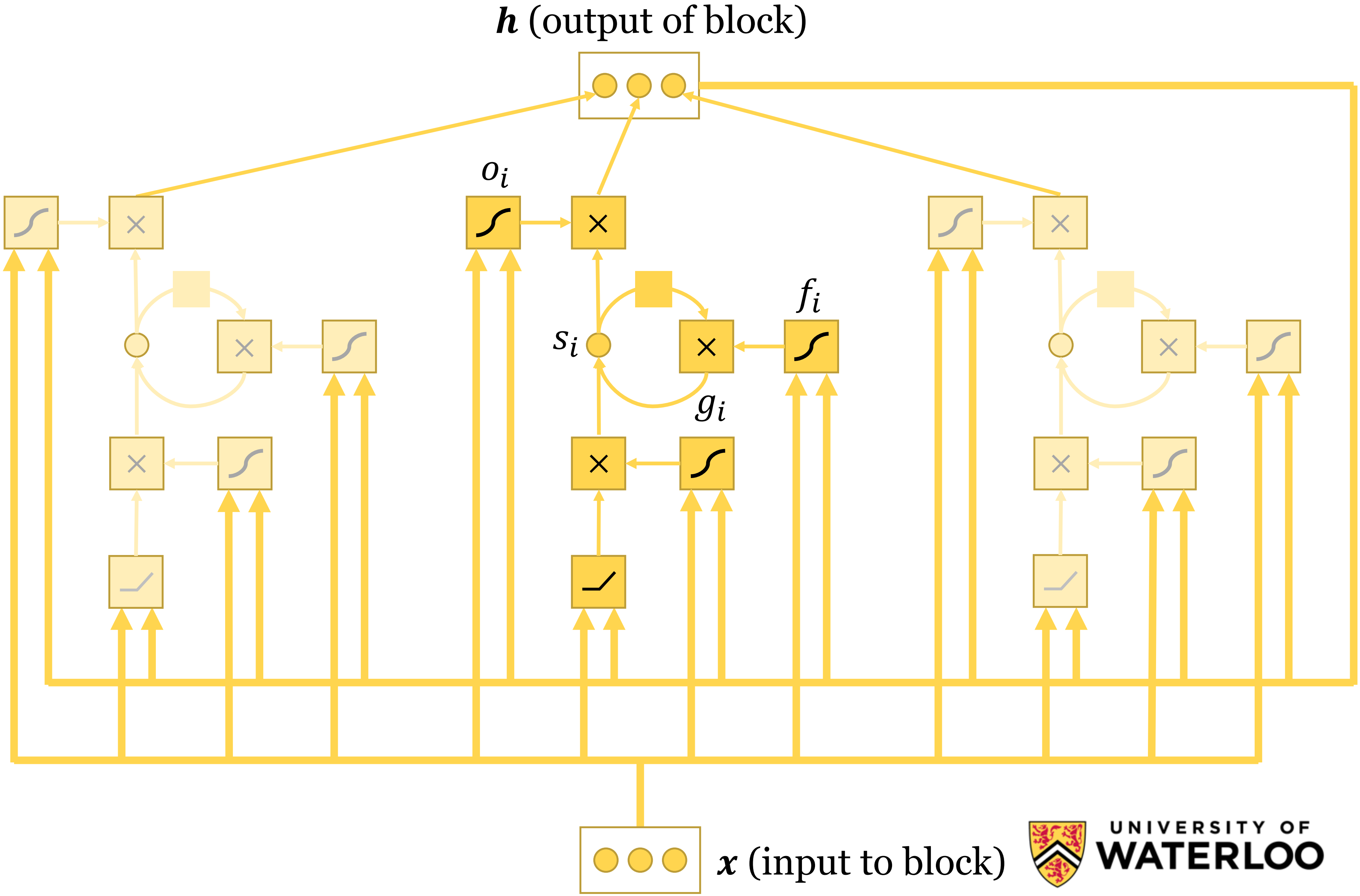
━ n-D signal

— 1-D signal

× scalar product

ʃ $\sigma(U_i \boldsymbol{x} + W_i \boldsymbol{h} + b_i)$ where U_i and W_i are the i^{th} rows of these matrices and b_i is a bias

⟋ $\gamma(B_i \boldsymbol{x} + A_i \boldsymbol{h} + b_i)$, where γ is a scalar nonlinearity


$\boldsymbol{h}$ (output of block)
o_i
s_i
f_i
g_i
$\boldsymbol{x}$ (input to block)

Summary

1. Recurrent networks are stateful
2. Recurrent networks can produce an output at each step or a summary output at the end of a sequence
3. Recurrent network gradients are found by backpropagation through time
4. The number of sequential steps normally scales with sequence length
5. Gains and nonlinearities compound with repeated application
6. Gating helps to retain long-term context

References

- Goodfellow, Bengio & Courville (2016) *Deep Learning*, MIT Press
- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735-1780.