

10-301/601: Introduction to Machine Learning

Lecture 20 – Recurrent Neural Networks

Henry Chai

6/2/25

Recurrent Neural Networks

- Neural networks are frequently applied to inputs with some inherent temporal or sequential structure (e.g., text or video) of variable length
- Idea: use the information from previous parts of the input to inform subsequent predictions
- Insight: the hidden layers learn a useful representation (relative to the task)
- Approach: incorporate the output from earlier hidden layers into later ones.

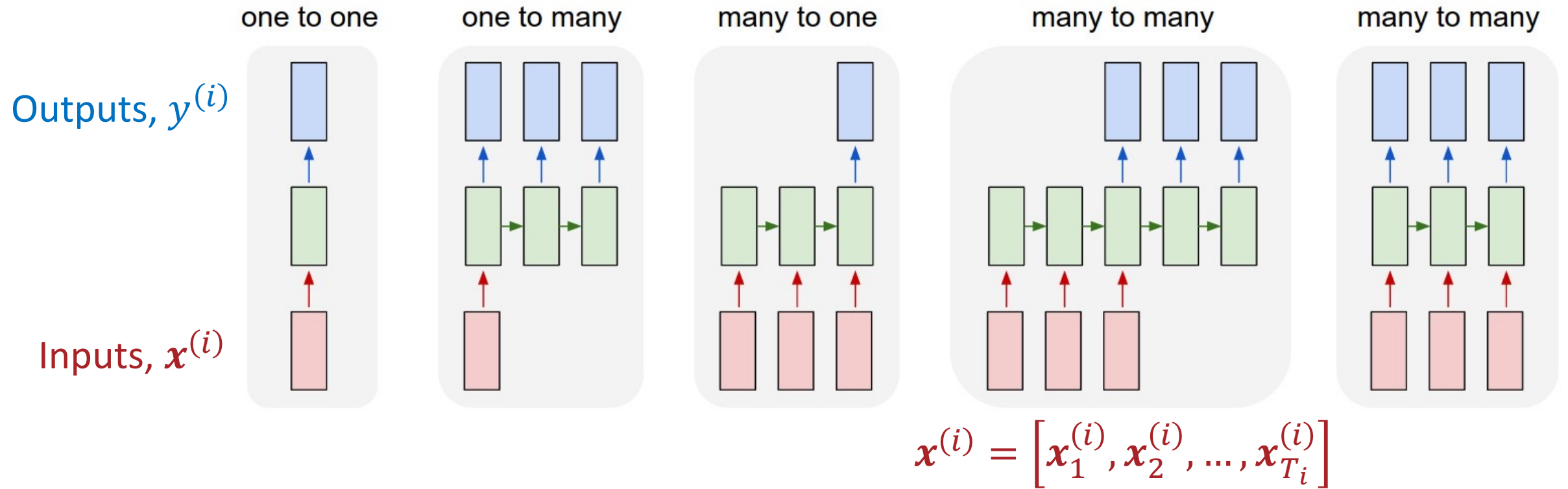
Example: Handwriting Recognition

U N E X P E C T E D

V O L C A N I C

E M B R A C E S

$$\mathbf{y}^{(i)} = [\mathbf{y}_1^{(i)}, \mathbf{y}_2^{(i)}, \dots, \mathbf{y}_{T_i}^{(i)}]$$



Sequential Data

Select all that apply: Which of the following are ways that you could reasonably formulate the handwritten character recognition task?

One-to-one

One-to-many

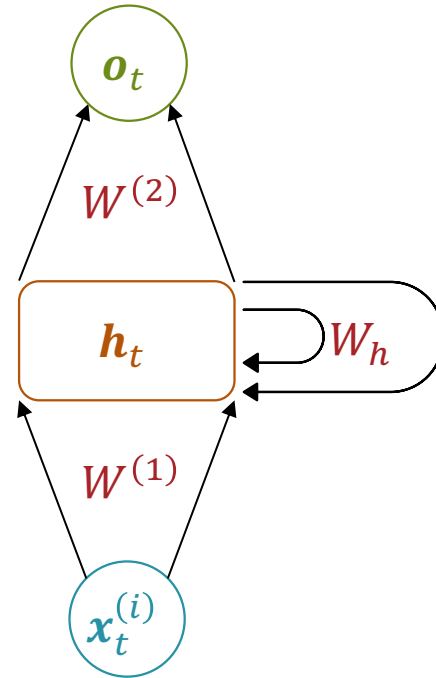
Many-to-one

Many-to-many (offset)

Many-to-many (syncd)

Recurrent Neural Networks

$$\mathbf{h}_t = \left[1, \theta \left(W^{(1)} \mathbf{x}_t^{(n)} + W_h \mathbf{h}_{t-1} \right) \right]^T \text{ and } \mathbf{o}_t = \hat{y}_t^{(n)} = \theta(W^{(2)} \mathbf{h}_t)$$



- Training dataset consists of (input **sequence**, label **sequence**) pairs, potentially of varying lengths

$$\mathcal{D} = \{(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}_{n=1}^N$$

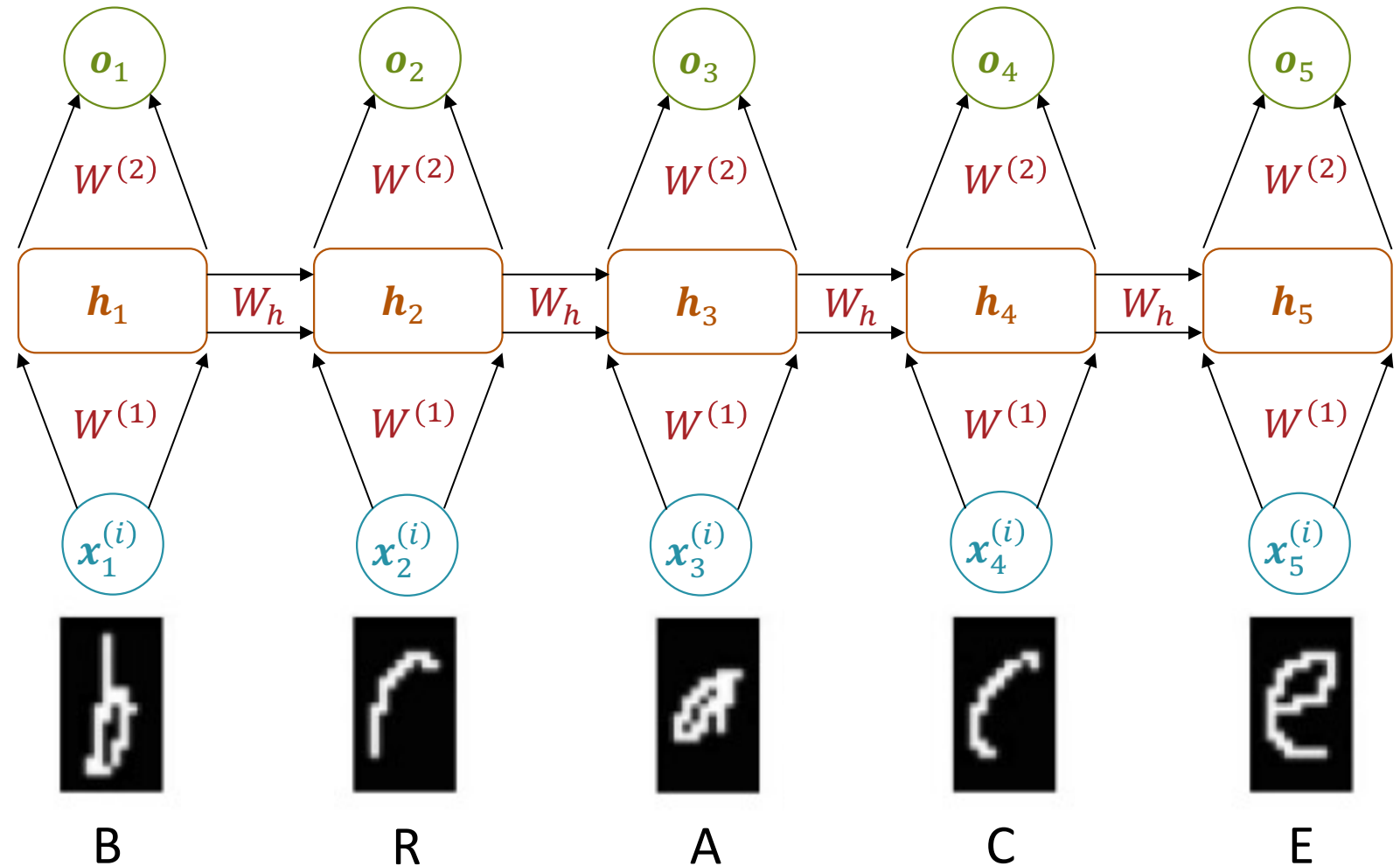
$$\mathbf{x}^{(n)} = [\mathbf{x}_1^{(n)}, \dots, \mathbf{x}_{T_n}^{(n)}]$$

$$\mathbf{y}^{(n)} = [\mathbf{y}_1^{(n)}, \dots, \mathbf{y}_{T_n}^{(n)}]$$

- This model requires an initial value for the hidden representation, $\mathbf{h}_0$, typically a vector of all zeros

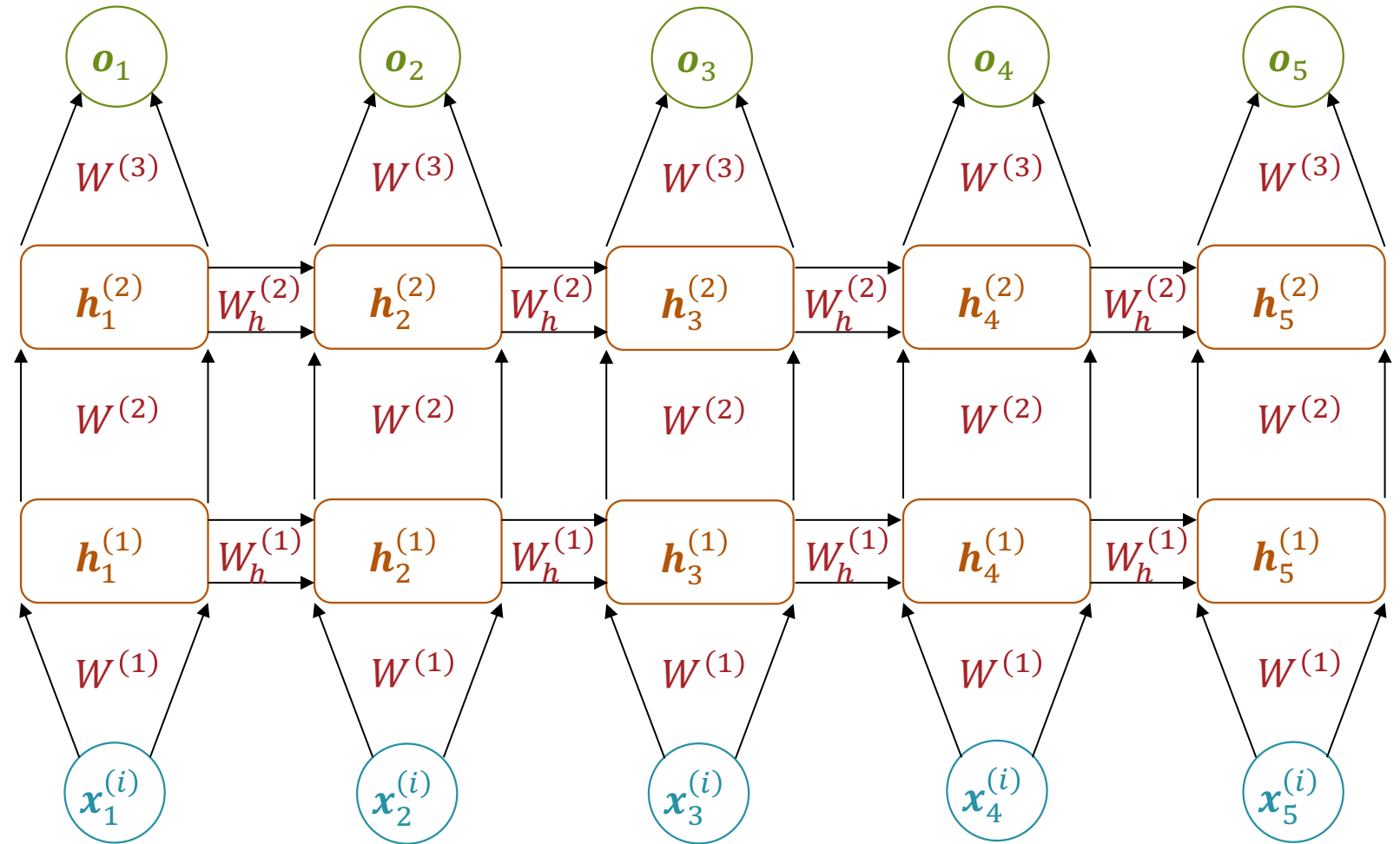
Unrolling Recurrent Neural Networks

$$\mathbf{h}_t = \left[1, \theta \left(W^{(1)} \mathbf{x}_t^{(n)} + W_h \mathbf{h}_{t-1} \right) \right]^T \text{ and } \mathbf{o}_t = \hat{y}_t^{(n)} = \theta(W^{(2)} \mathbf{h}_t)$$



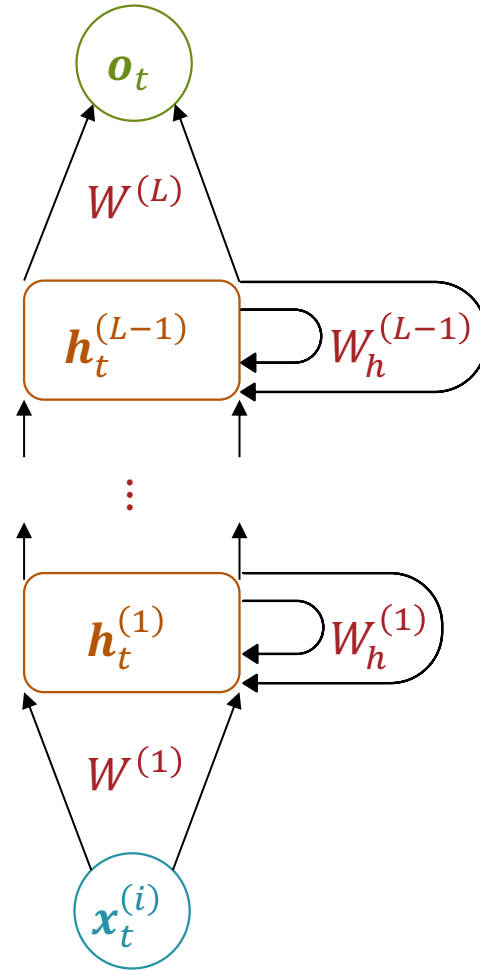
Deep Recurrent Neural Networks

$$\mathbf{h}_t^{(l)} = \left[1, \theta \left(W^{(l)} \mathbf{h}_t^{(l-1)} + W_h^{(l)} \mathbf{h}_{t-1}^{(l)} \right) \right]^T \text{ and } \mathbf{o}_t = \hat{\mathbf{y}}_t^{(n)} = \theta \left(W^{(L)} \mathbf{h}_t^{(L-1)} \right)$$



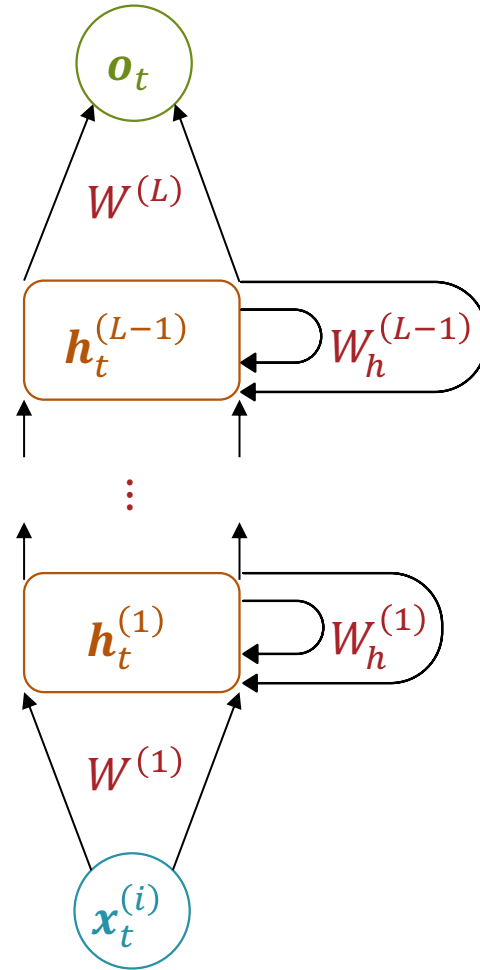
Deep Recurrent Neural Networks

$$\mathbf{h}_t^{(l)} = \left[1, \theta \left(W^{(l)} \mathbf{h}_t^{(l-1)} + W_h^{(l)} \mathbf{h}_{t-1}^{(l)} \right) \right]^T \text{ and } \mathbf{o}_t = \hat{y}_t^{(n)} = \theta \left(W^{(L)} \mathbf{h}_t^{(L-1)} \right)$$



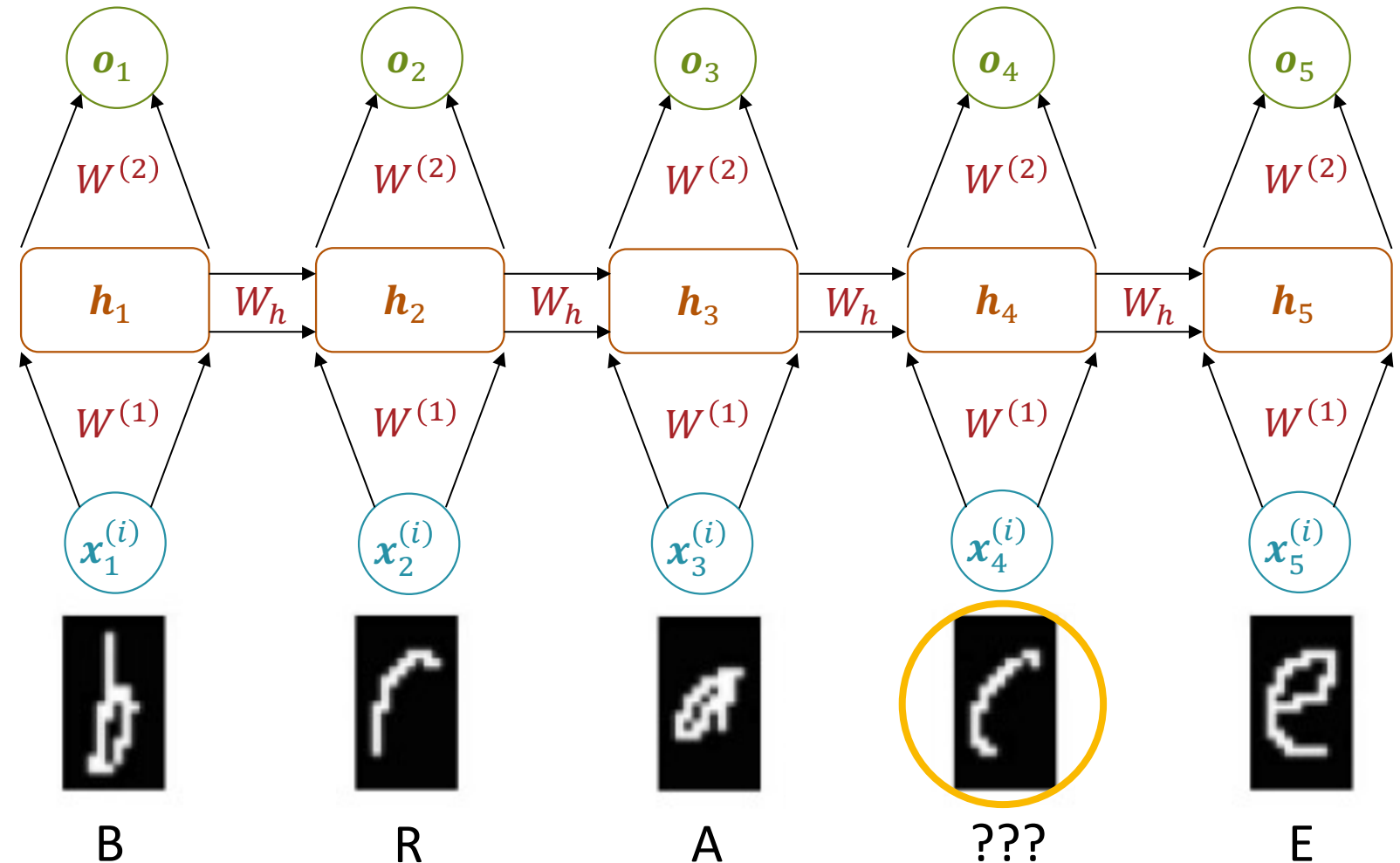
But why do we only pass information forward?
What if later time steps have useful information as well?

$$\mathbf{h}_t^{(l)} = \left[1, \theta \left(W^{(l)} \mathbf{h}_t^{(l-1)} + W_h^{(l)} \mathbf{h}_{t-1}^{(l)} \right) \right]^T \text{ and } \mathbf{o}_t = \hat{y}_t^{(n)} = \theta \left(W^{(L)} \mathbf{h}_t^{(L-1)} \right)$$



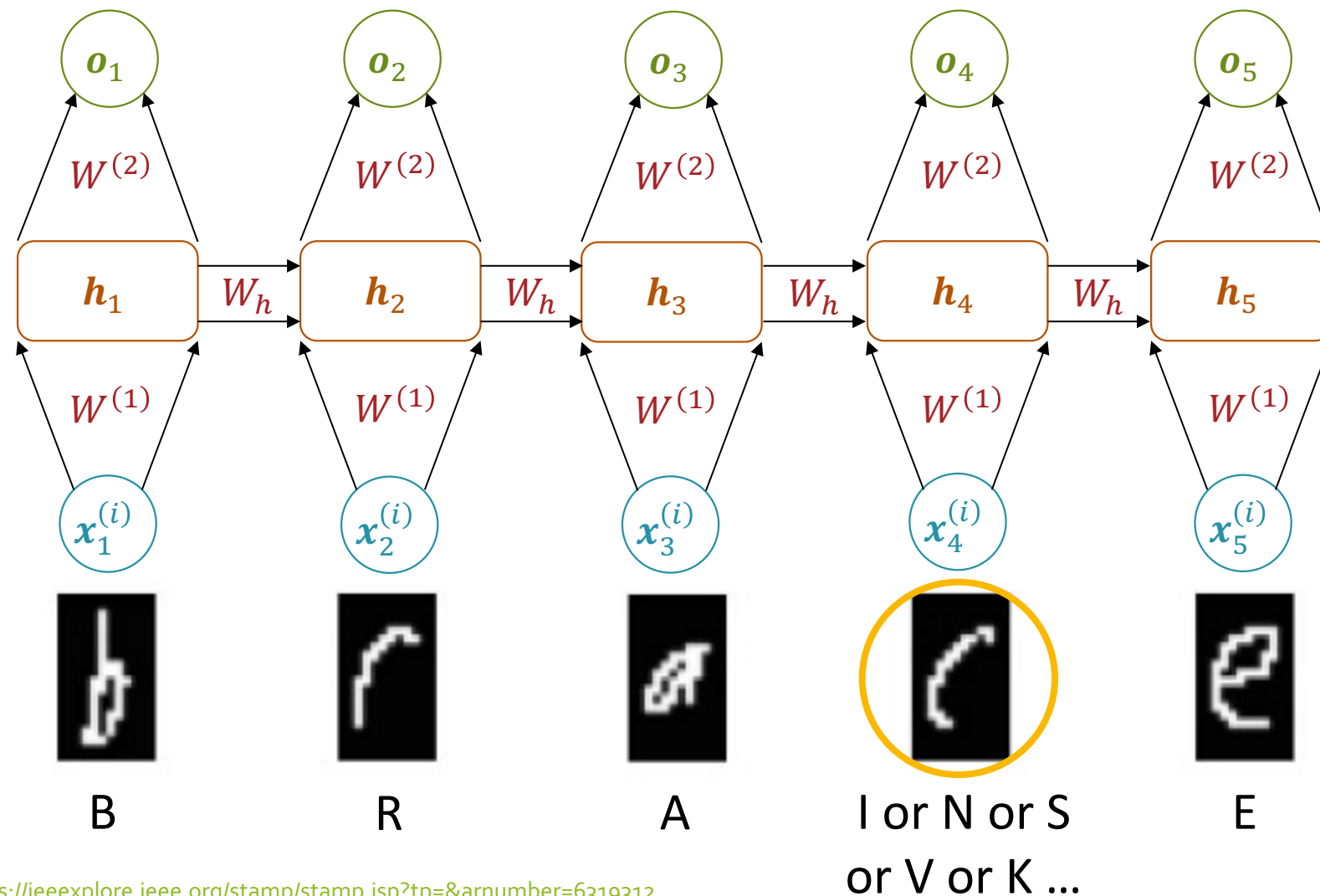
But why do we only pass information forward?
What if later time steps have useful information as well?

$$\mathbf{h}_t = \left[1, \theta \left(W^{(1)} \mathbf{x}_t^{(n)} + W_h \mathbf{h}_{t-1} \right) \right]^T \text{ and } \mathbf{o}_t = \hat{y}_t^{(n)} = \theta(W^{(2)} \mathbf{h}_t)$$



But why do we only pass information forward?
What if later time steps have useful information as well?

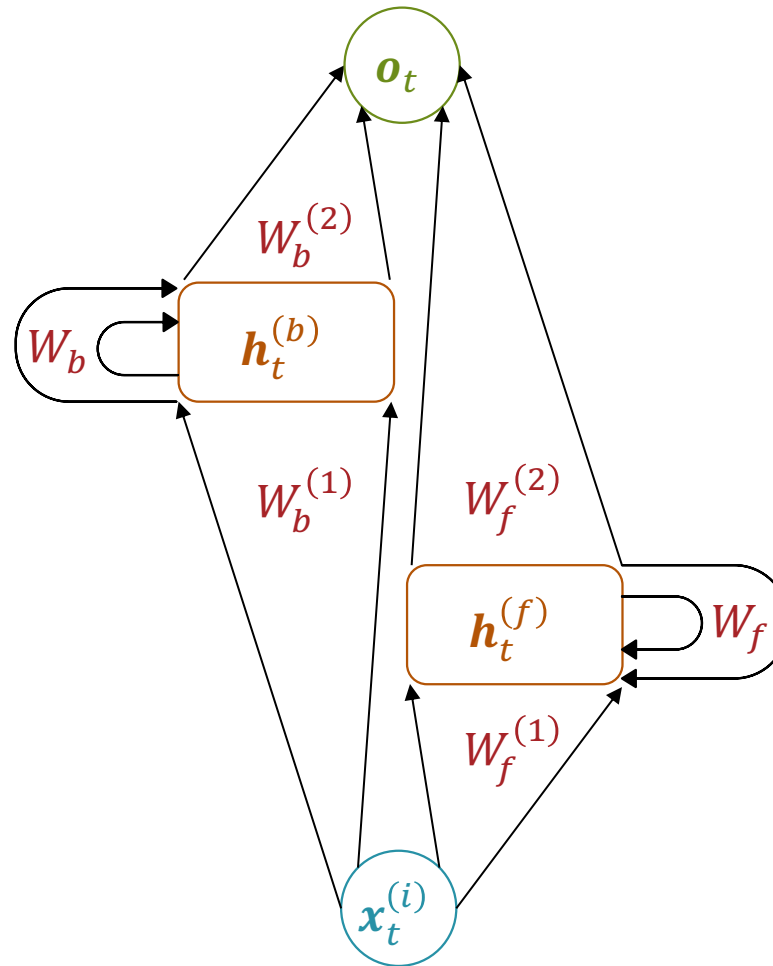
$$\mathbf{h}_t = \left[1, \theta \left(W^{(1)} \mathbf{x}_t^{(n)} + W_h \mathbf{h}_{t-1} \right) \right]^T \text{ and } \mathbf{o}_t = \hat{y}_t^{(n)} = \theta(W^{(2)} \mathbf{h}_t)$$



Bidirectional Recurrent Neural Networks

$$\mathbf{h}_t^{(f)} = \left[1, \theta \left(W_f^{(1)} \mathbf{x}_t^{(n)} + W_f \mathbf{h}_{t-1} \right) \right]^T \text{ and } \mathbf{h}_t^{(b)} = \left[1, \theta \left(W_b^{(1)} \mathbf{x}_t^{(n)} + W_b \mathbf{h}_{t+1} \right) \right]^T$$

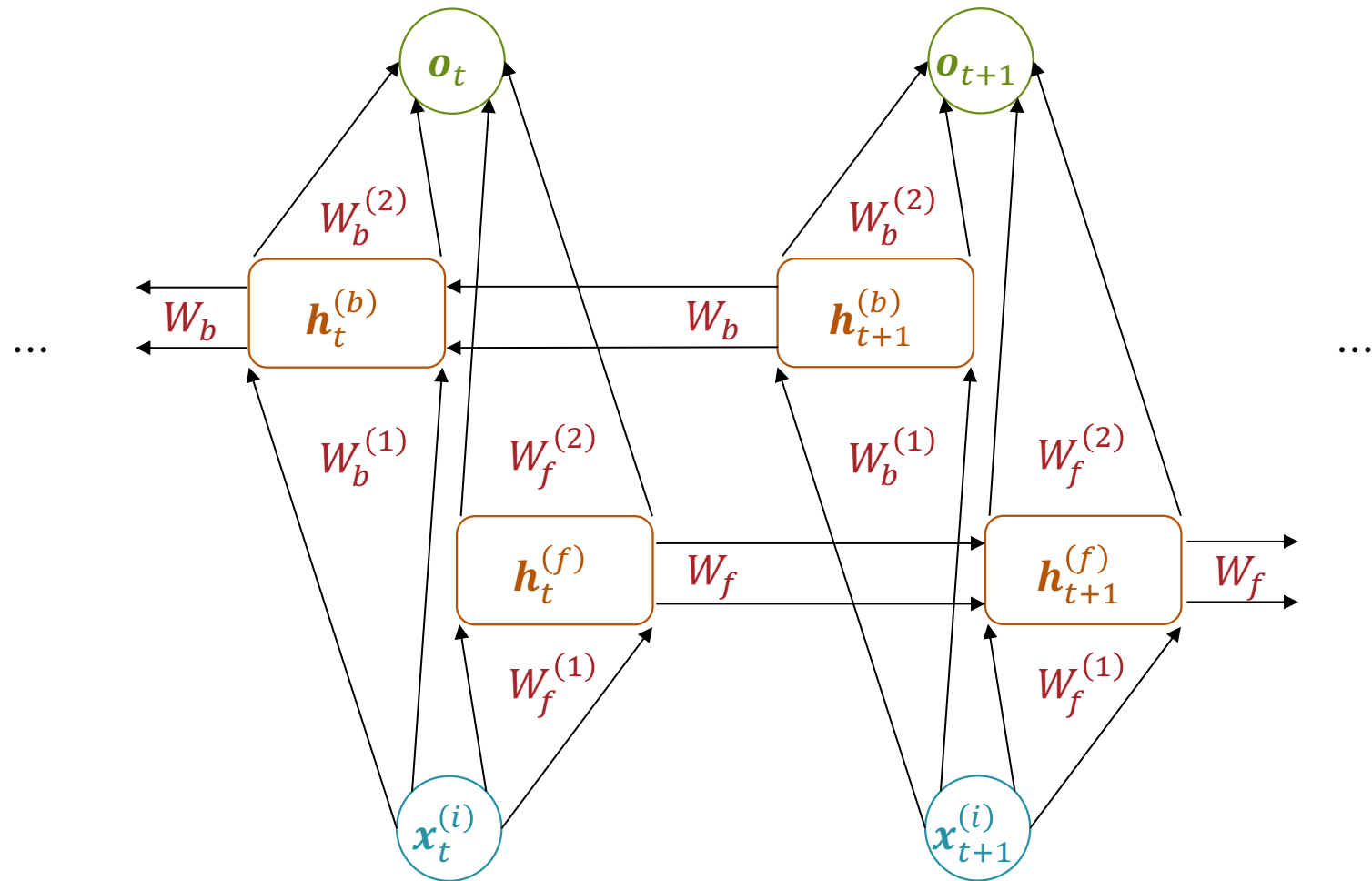
$$\mathbf{o}_t = \hat{y}_t^{(n)} = \theta \left(W_f^{(2)} \mathbf{h}_t^{(f)} + W_b^{(2)} \mathbf{h}_t^{(b)} \right)$$



Unrolling Bidirectional Recurrent Neural Networks

$$\mathbf{o}_t = \hat{\mathbf{y}}_t^{(n)} = \theta \left(W_f^{(2)} \mathbf{h}_t^{(f)} + W_b^{(2)} \mathbf{h}_t^{(b)} \right)$$

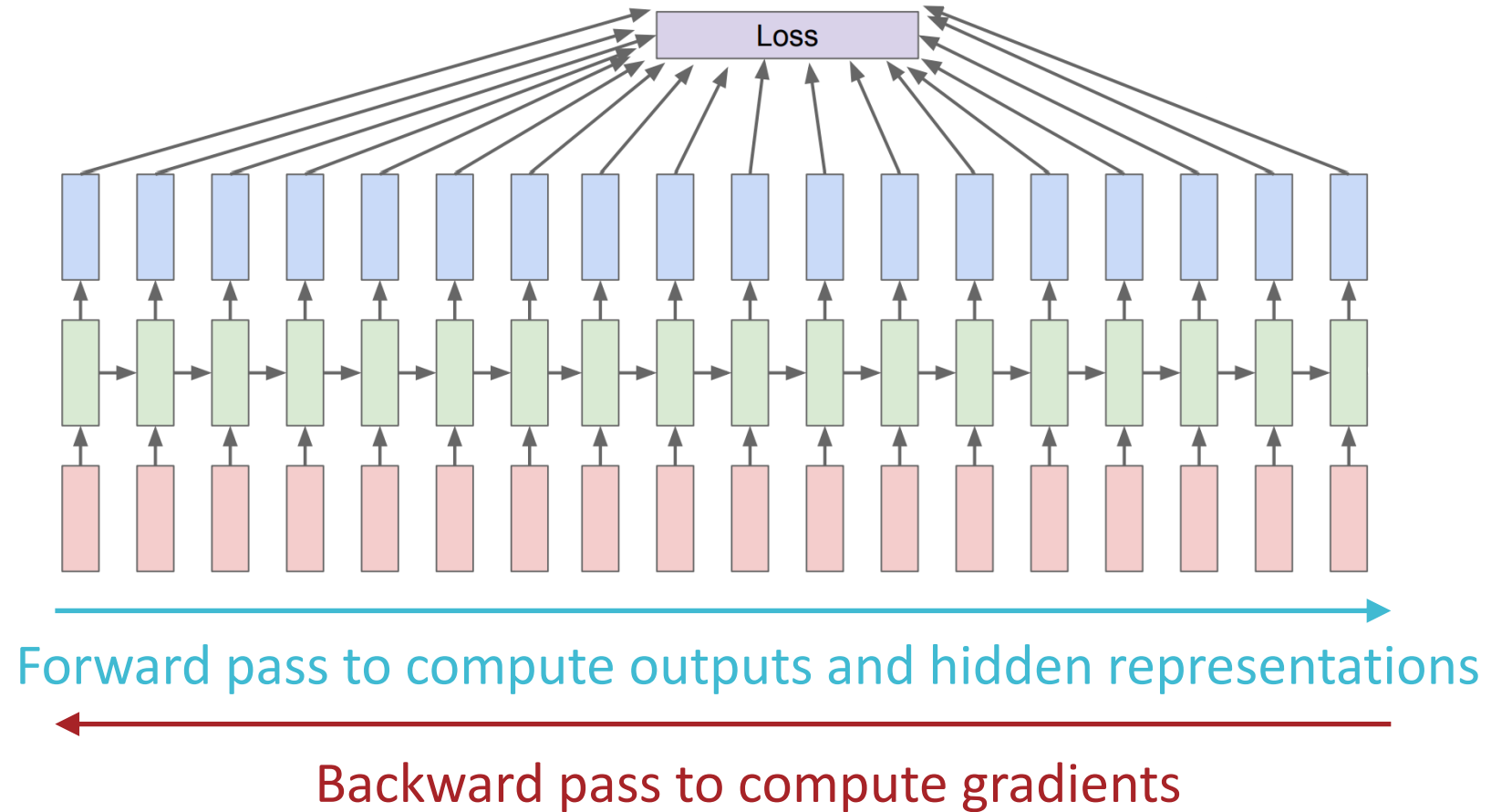
$$\mathbf{h}_t^{(f)} = \left[1, \theta \left(W_f^{(1)} \mathbf{x}_t^{(n)} + W_f \mathbf{h}_{t-1} \right) \right]^T \text{ and } \mathbf{h}_t^{(b)} = \left[1, \theta \left(W_b^{(1)} \mathbf{x}_t^{(n)} + W_b \mathbf{h}_{t+1} \right) \right]^T$$



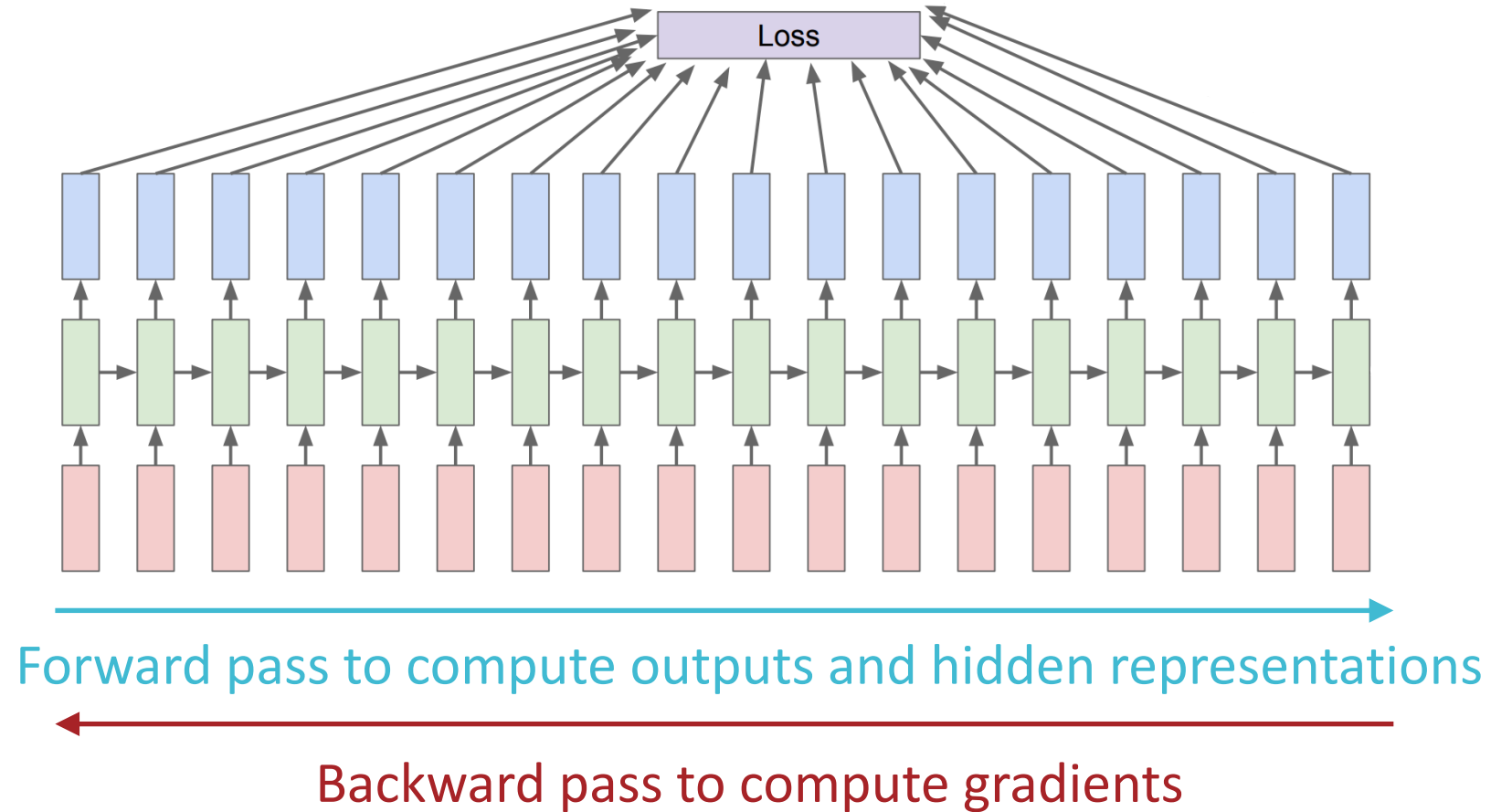
Training RNNs

- A (deep/bidirectional) RNN simply represents a (somewhat complicated) computation graph
 - Weights are shared between different timesteps, significantly reducing the number of parameters to be learned!
- Can be trained using (stochastic) gradient descent/backpropagation → “backpropagation through time”

Training RNNs



Training RNNs: Challenges

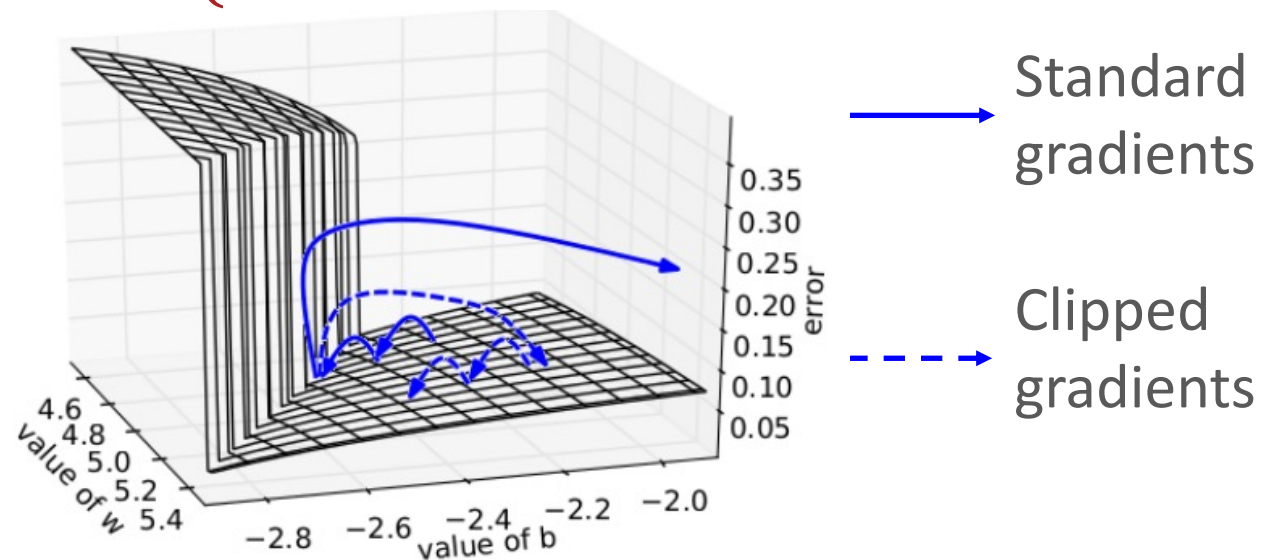


- Issue: as the sequence length grows, the gradient is more likely to explode or vanish

Gradient Clipping (Pascanu et al., 2013)

- Common strategy to deal with exploding gradients: if the magnitude of the gradient ever exceeds some threshold, simply scale it down to the threshold

$$G = \begin{cases} \nabla_W \ell^{(n)} & \text{if } \|\nabla_W \ell^{(n)}\|_2 \leq \tau \\ \left(\frac{\tau}{\|\nabla_W \ell^{(n)}\|_2} \right) \nabla_W \ell^{(n)} & \text{otherwise} \end{cases}$$



Recall: Computing Gradients

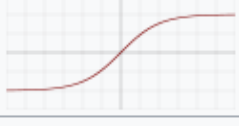
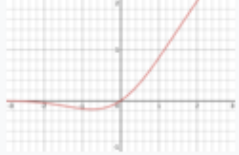
Insight: $s_b^{(l)}$ *only* affects $\ell^{(n)}$ via $o_b^{(l)}$

$$\text{Chain rule: } \delta_b^{(l)} = \frac{\partial \ell^{(n)}}{\partial o_b^{(l)}} \left(\frac{\partial o_b^{(l)}}{\partial s_b^{(l)}} \right)$$

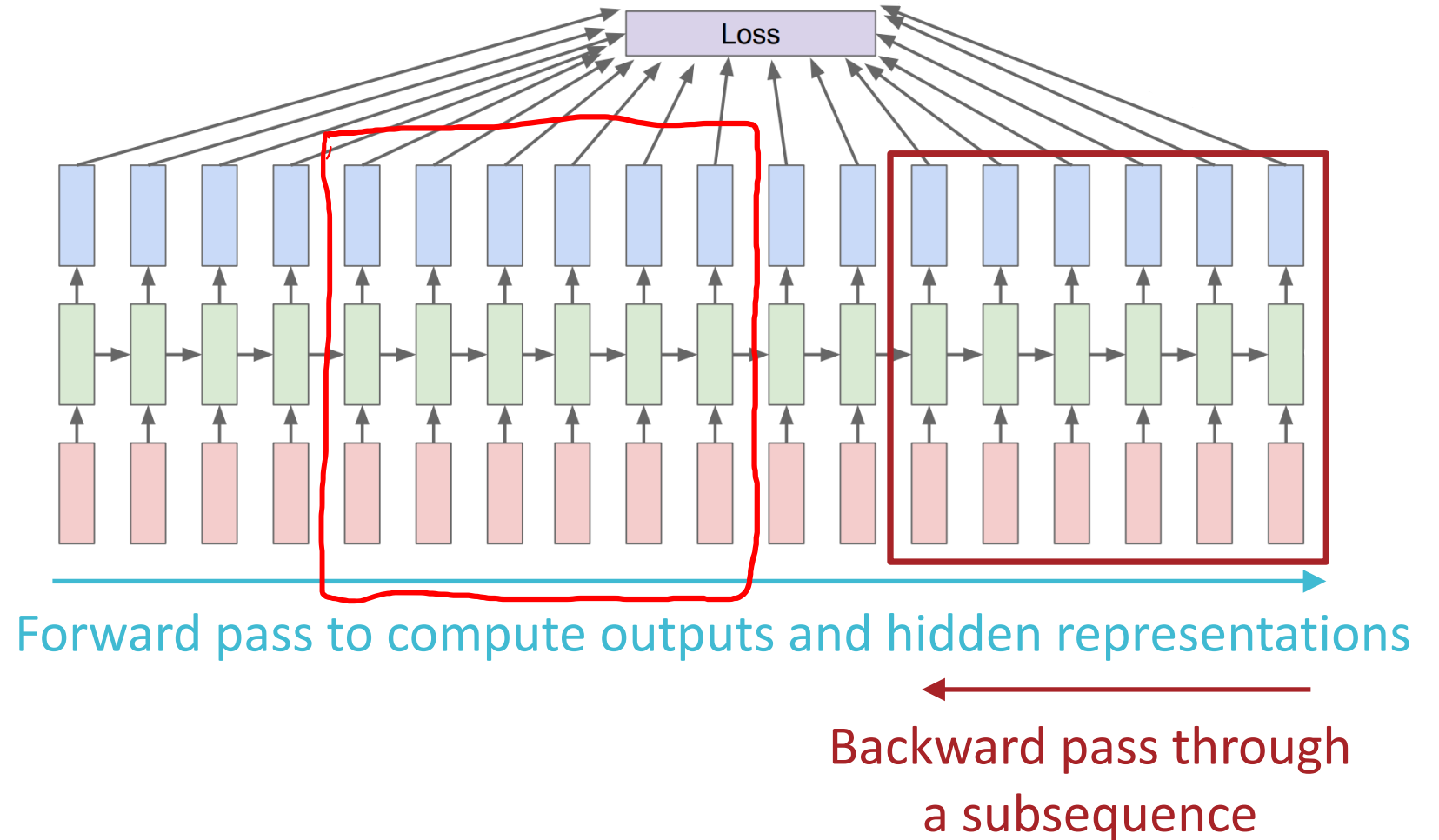
$$\begin{aligned} o_b^{(l)} = \theta(s_b^{(l)}) &\rightarrow \frac{\partial o_b^{(l)}}{\partial s_b^{(l)}} = \frac{\partial \theta(s_b^{(l)})}{\partial s_b^{(l)}} \\ &= 1 - \left(\tanh(s_b^{(l)}) \right)^2 \end{aligned}$$

when $\theta(\cdot) = \tanh(\cdot)$

Recall: Activation Functions

Logistic, sigmoid, or soft step		$\sigma(x) = \frac{1}{1 + e^{-x}}$
Hyperbolic tangent (tanh)		$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
Rectified linear unit (ReLU) ^[7]		$\begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ $= \max\{0, x\} = x \mathbf{1}_{x>0}$
Gaussian Error Linear Unit (GELU) ^[4]		$\frac{1}{2}x \left(1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right) \right)$ $= x\Phi(x)$
Softplus ^[8]		$\ln(1 + e^x)$
Exponential linear unit (ELU) ^[9]		$\begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ with parameter α
Leaky rectified linear unit (Leaky ReLU) ^[11]		$\begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$
Parametric rectified linear unit (PReLU) ^[12]		$\begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$ with parameter α

Truncated Backpropagation Through Time



- Idea: limit the number of time steps to backprop through

Key Takeaways

- Recurrent neural networks use contextual information to reason about sequential data
 - Can still be learned using backpropagation → backpropagation through time
 - Susceptible to exploding/vanishing gradients for long training sequences