10-301/601: Introduction to Machine Learning Lecture 20 – Recurrent Neural Networks

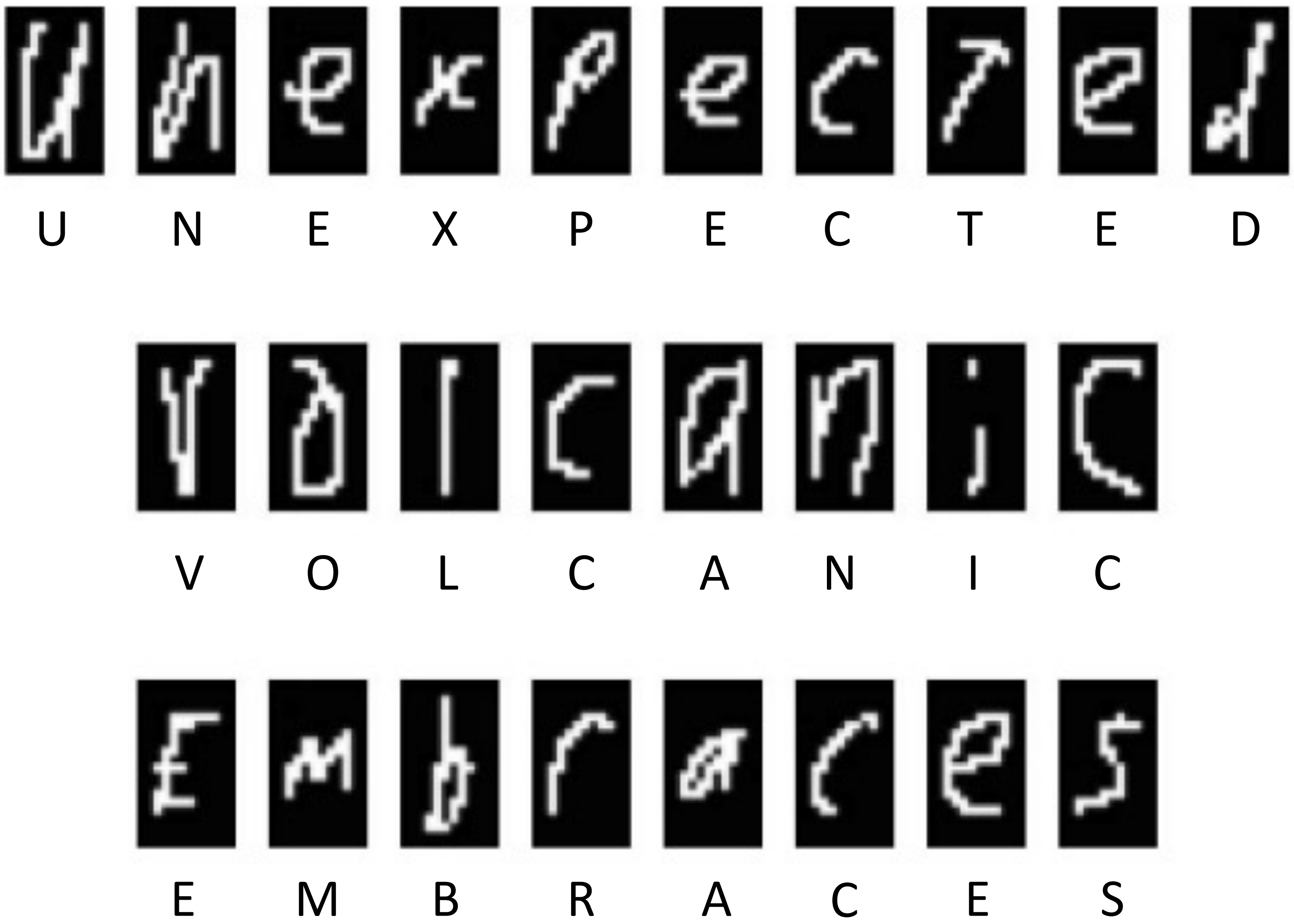
Henry Chai

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.

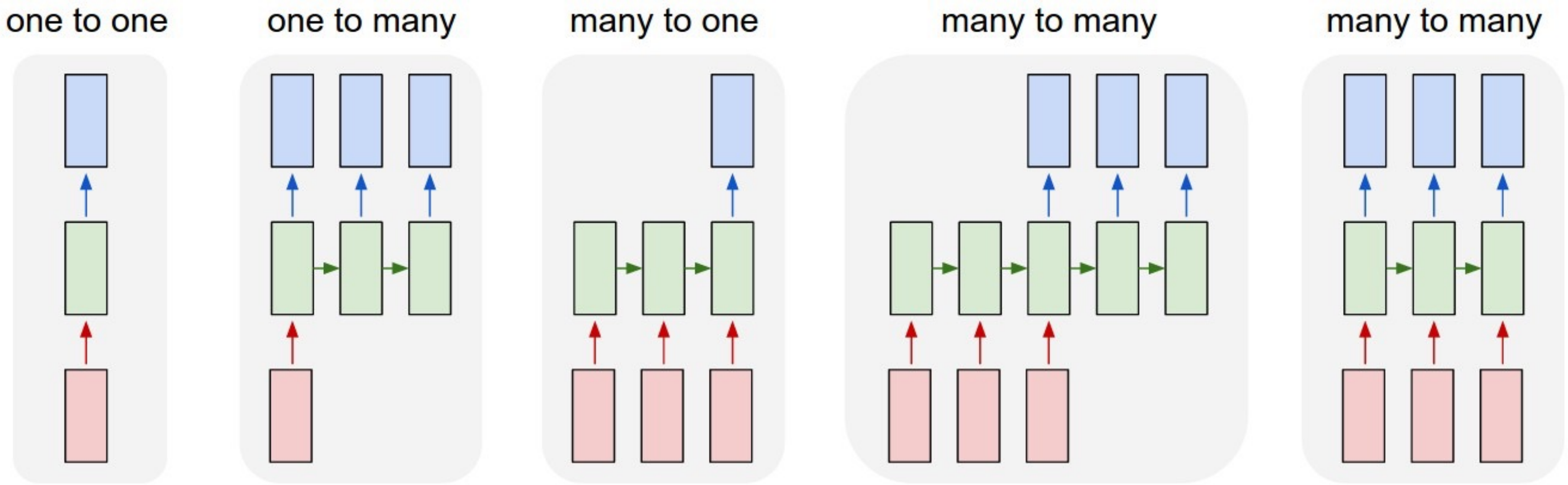
Example: Handwriting Recognition



Source: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6319312

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

Outputs, $y^{(i)}$



Inputs, $\boldsymbol{x}^{(i)}$

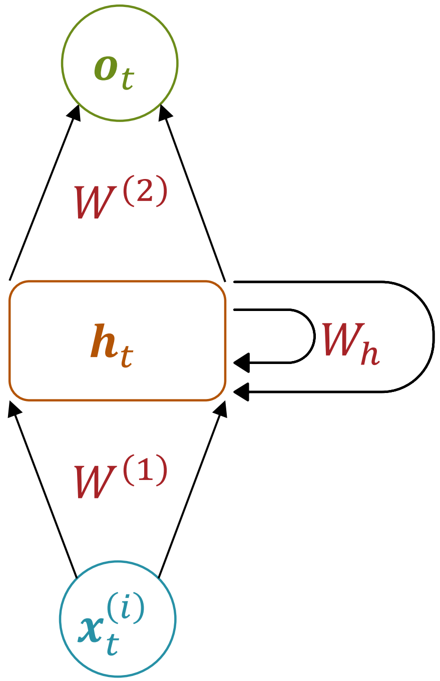
$$\boldsymbol{x}^{(i)} = \left[\boldsymbol{x}_1^{(i)}, \boldsymbol{x}_2^{(i)}, \dots, \boldsymbol{x}_{T_i}^{(i)}\right]$$

Sequential Data

Source: https://karpathy.github.io/2015/05/21/rnn-effectiveness/

Recurrent Neural Networks

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



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

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

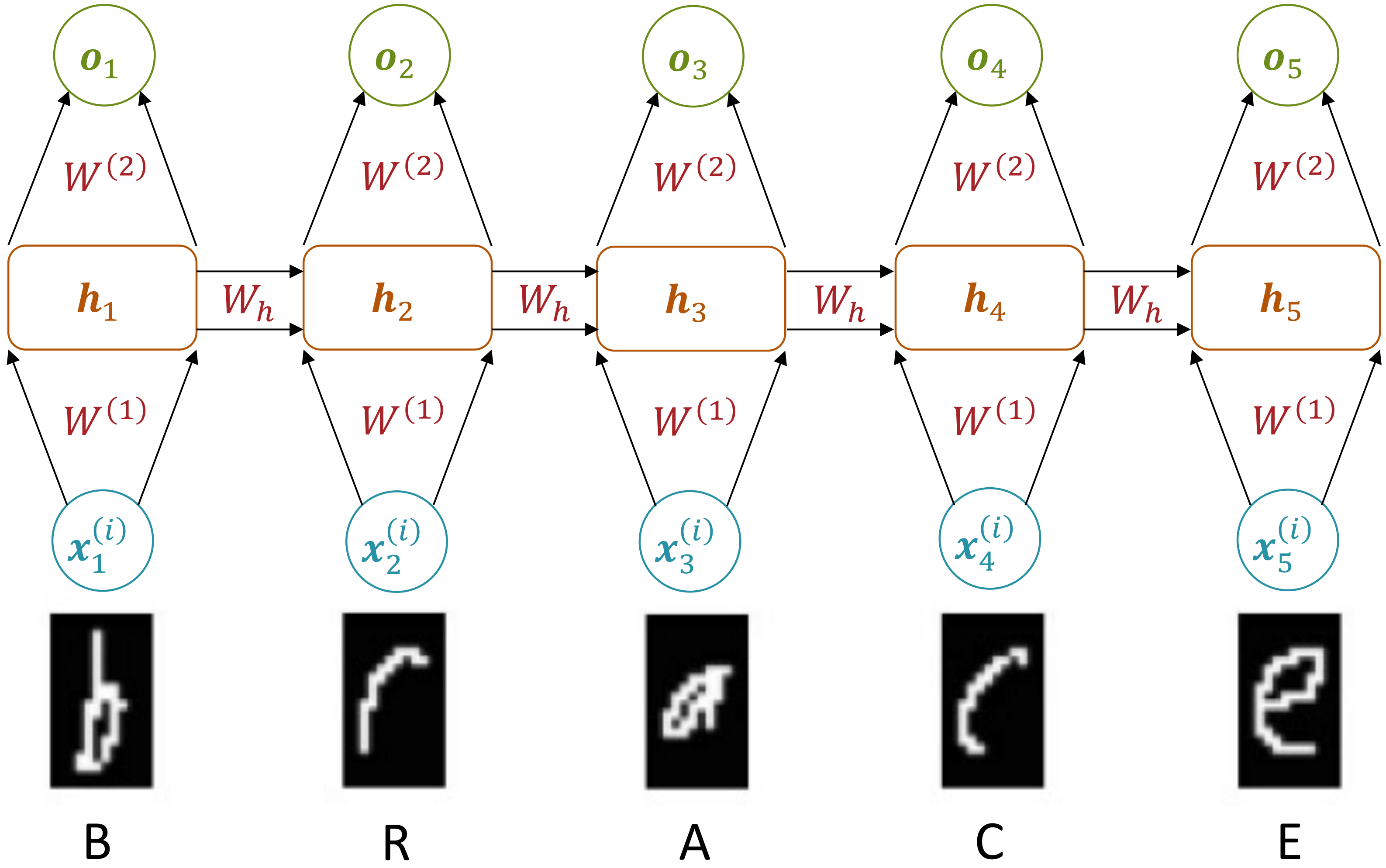
$$\boldsymbol{x}^{(n)} = \left[\boldsymbol{x}_1^{(n)}, \dots, \boldsymbol{x}_{T_n}^{(n)}\right]$$

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

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

Unrolling Recurrent Neural Networks

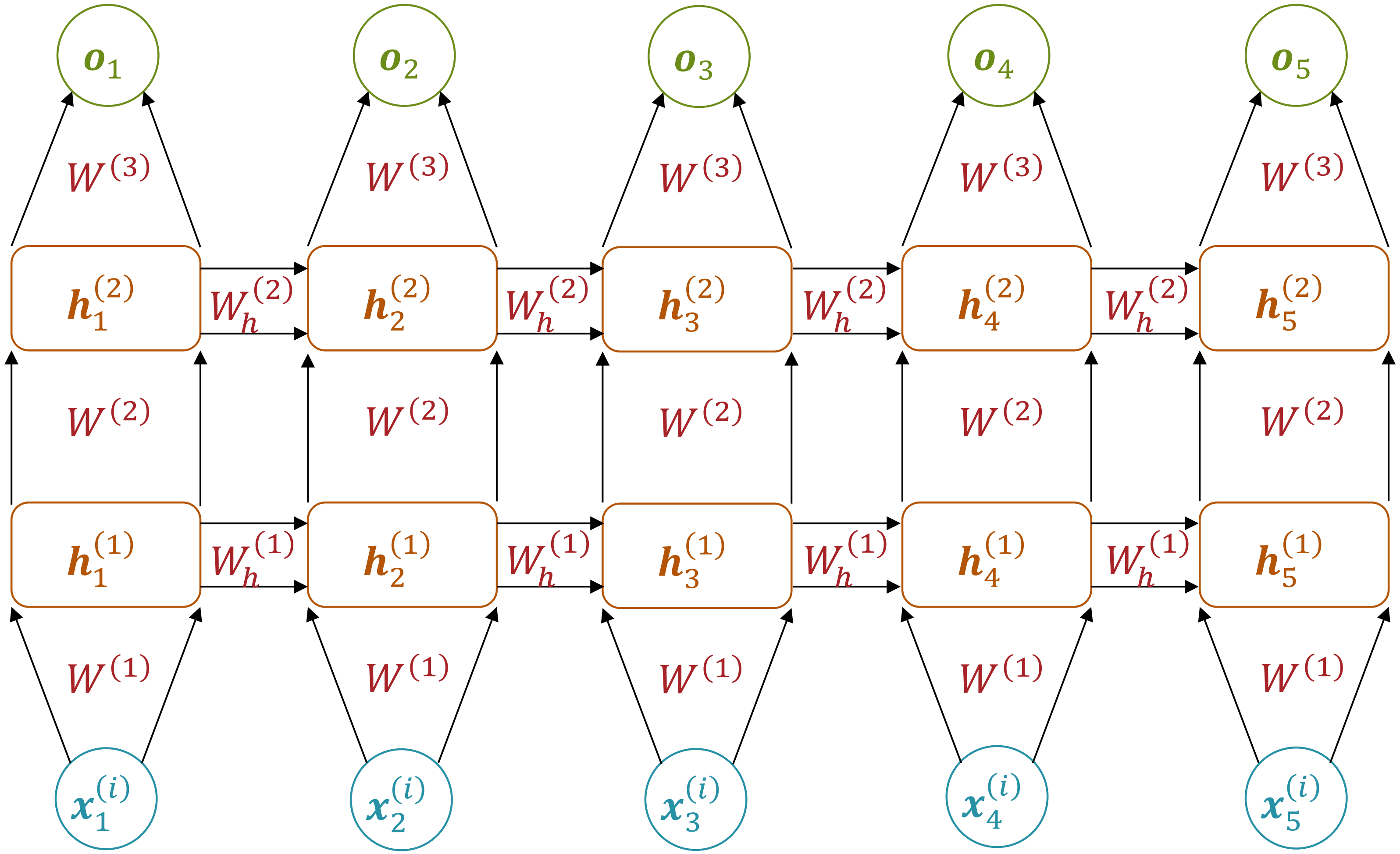
$$\boldsymbol{h}_t = \left[1, \theta\left(W^{(1)}\boldsymbol{x}_t^{(n)} + W_h \boldsymbol{h}_{t-1}\right)\right]^T \text{ and } \boldsymbol{o}_t = \hat{y}_t^{(n)} = \theta\left(W^{(2)}\boldsymbol{h}_t\right)$$



Source: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6319312

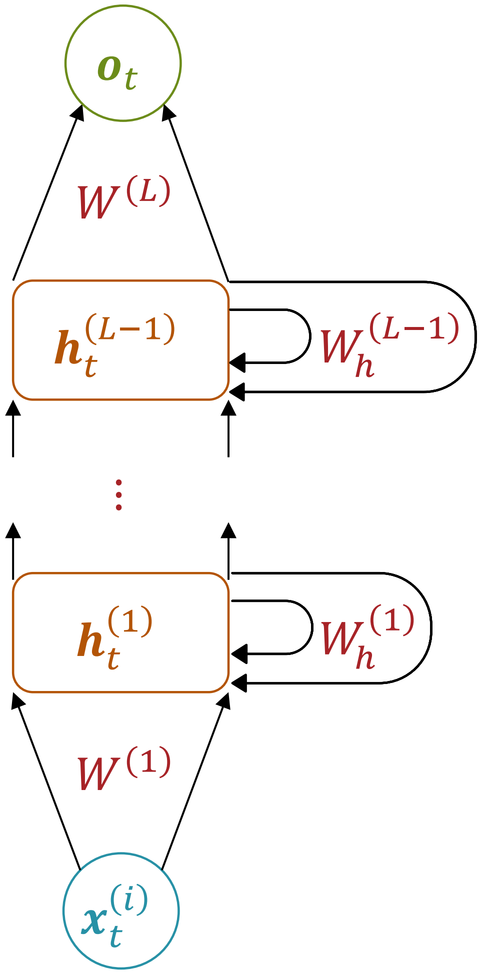
Deep Recurrent Neural Networks

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



Deep Recurrent Neural Networks

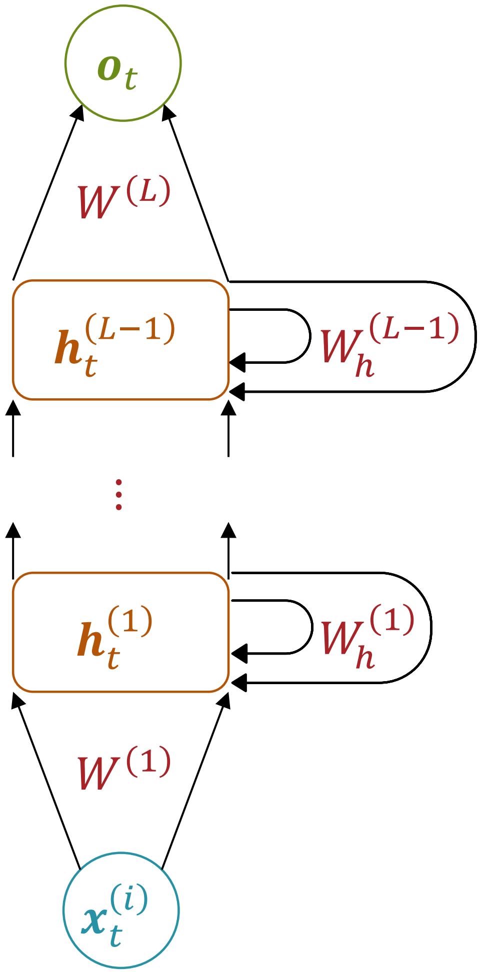
$\boldsymbol{h}_t^{(l)} = \left[1, \theta\left(W^{(l)}\boldsymbol{h}_t^{(l-1)} + W_h^{(l)}\boldsymbol{h}_{t-1}^{(l)}\right)\right]^T$ and $\boldsymbol{o}_t = \hat{y}_t^{(n)} = \theta\left(W^{(L)}\boldsymbol{h}_t^{(L-1)}\right)$



Source: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6319312

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

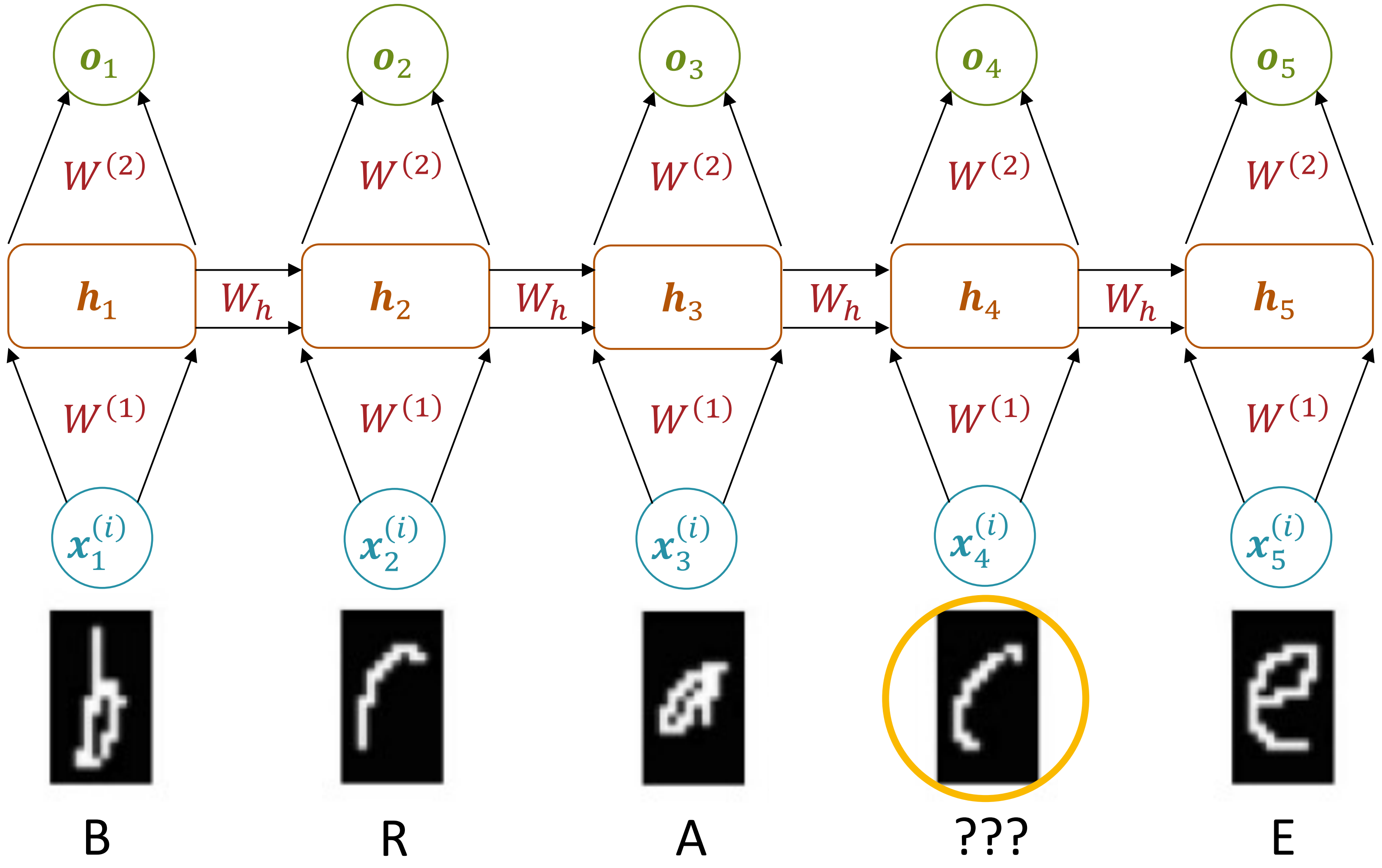
But why do we only pass information forward? What if later time steps have useful information as well?



Source: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6319312

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

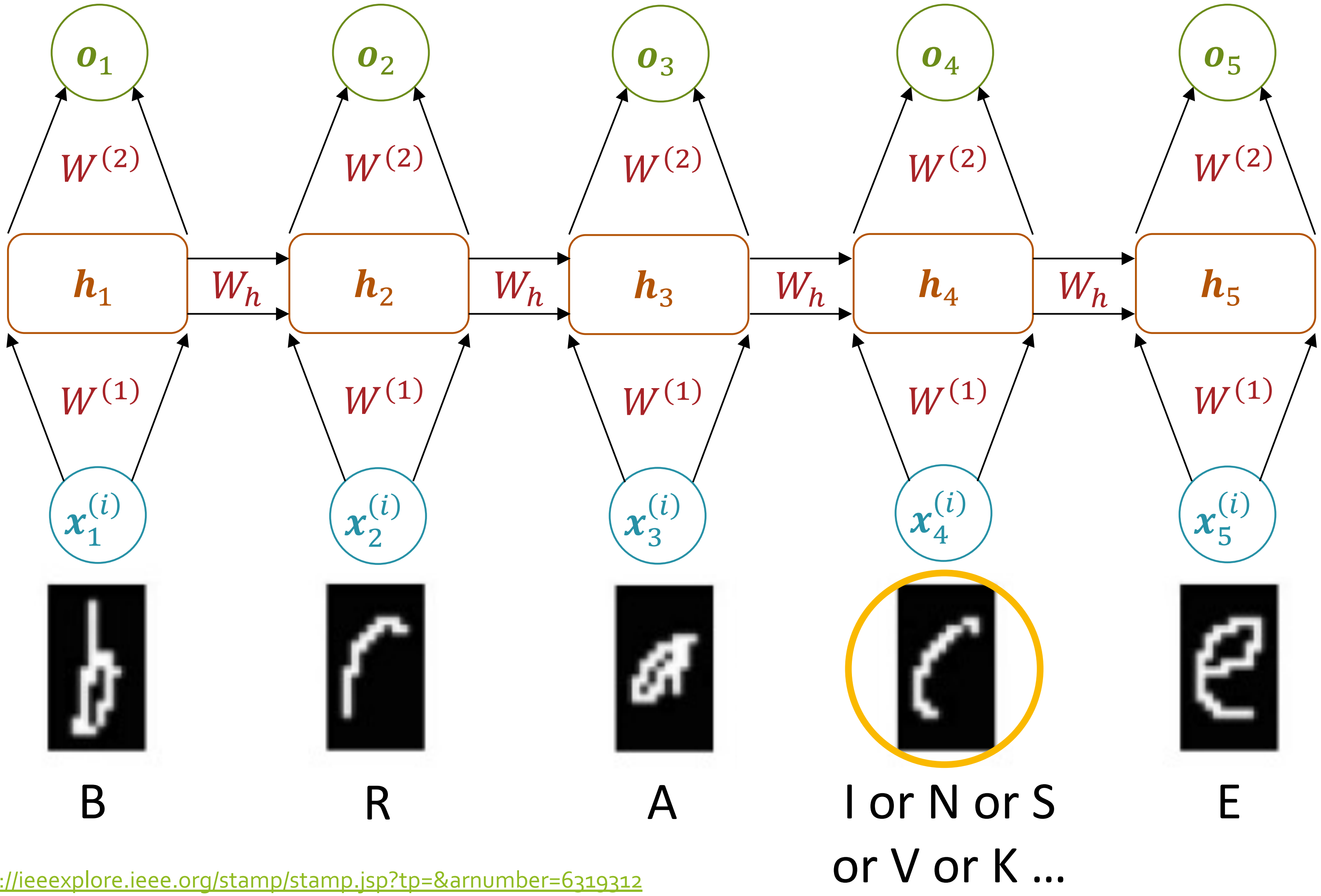
But why do we only pass information forward? What if later time steps have useful information as well?



Source: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6319312

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

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

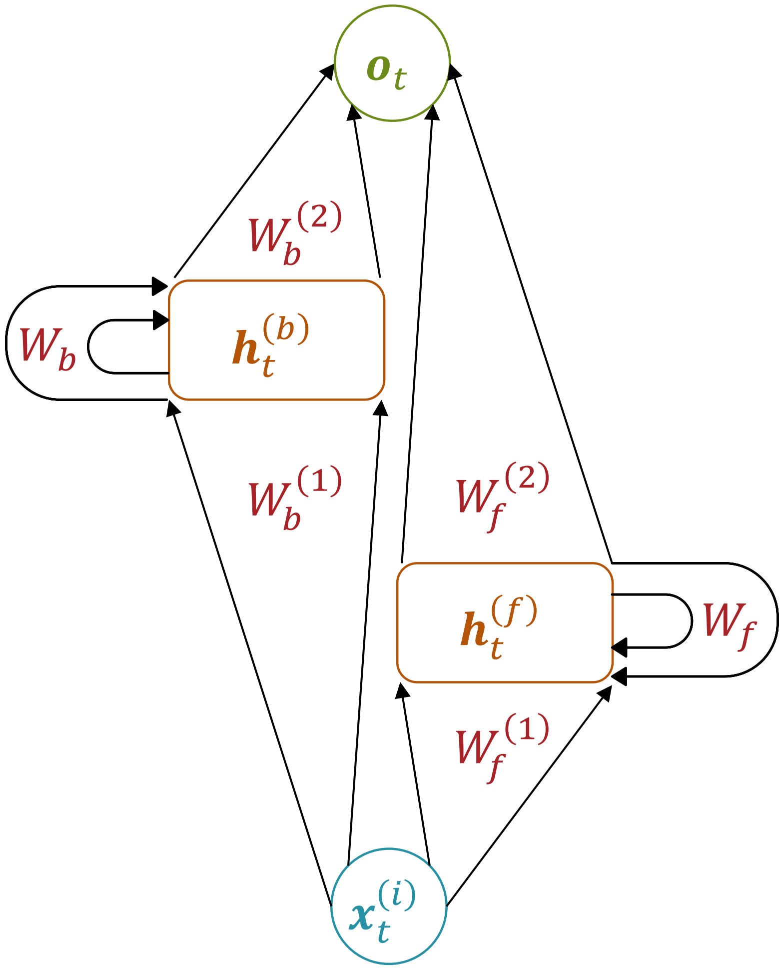


Source: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6319312

Bidirectional Recurrent Neural Networks

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

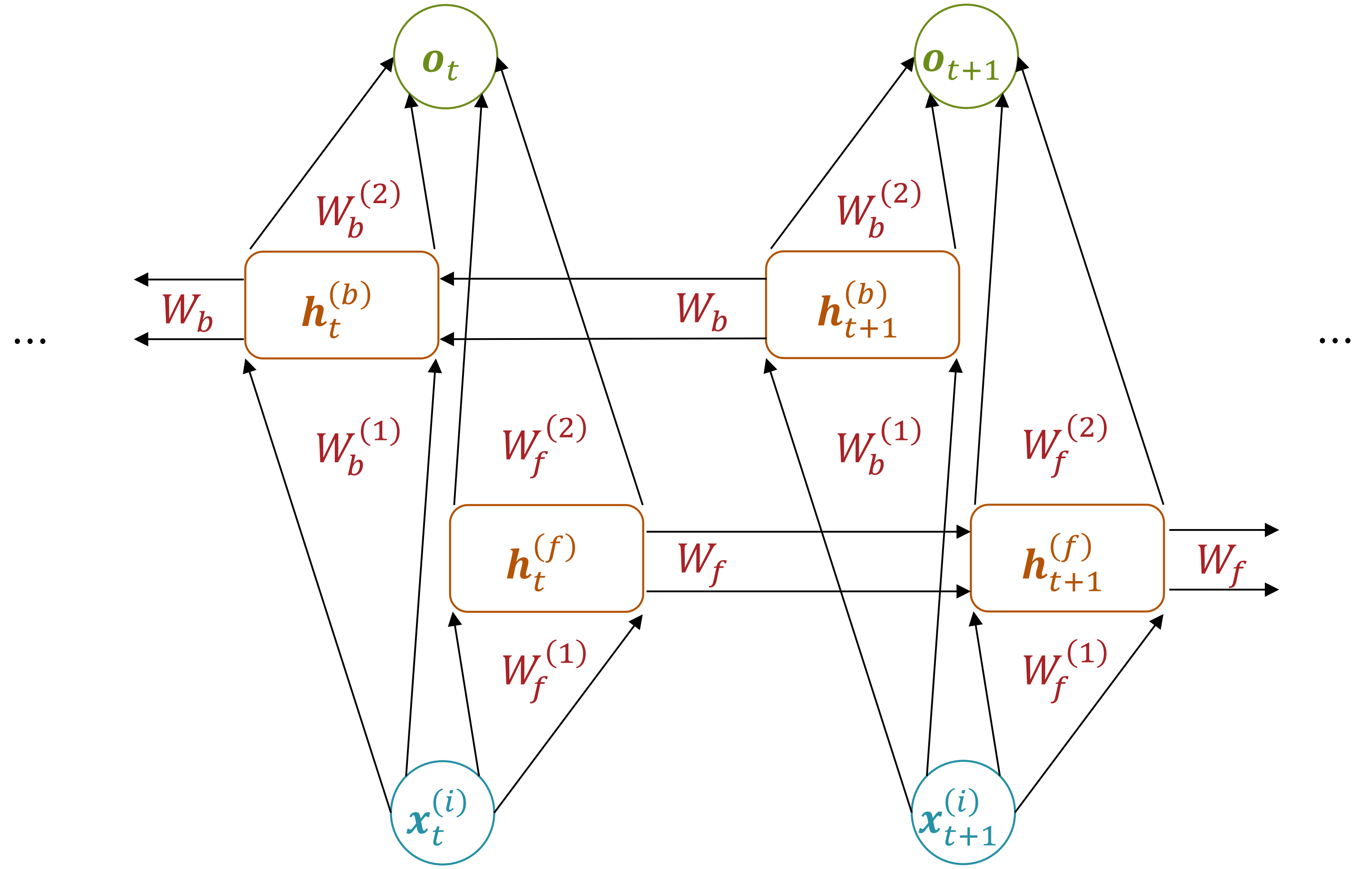
$\boldsymbol{o}_t = \hat{y}_t^{(n)} = \theta\left(W_f^{(2)} \boldsymbol{h}_t^{(f)} + W_b^{(2)} \boldsymbol{h}_t^{(b)}\right)$



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

Unrolling Bidirectional Recurrent Neural Networks

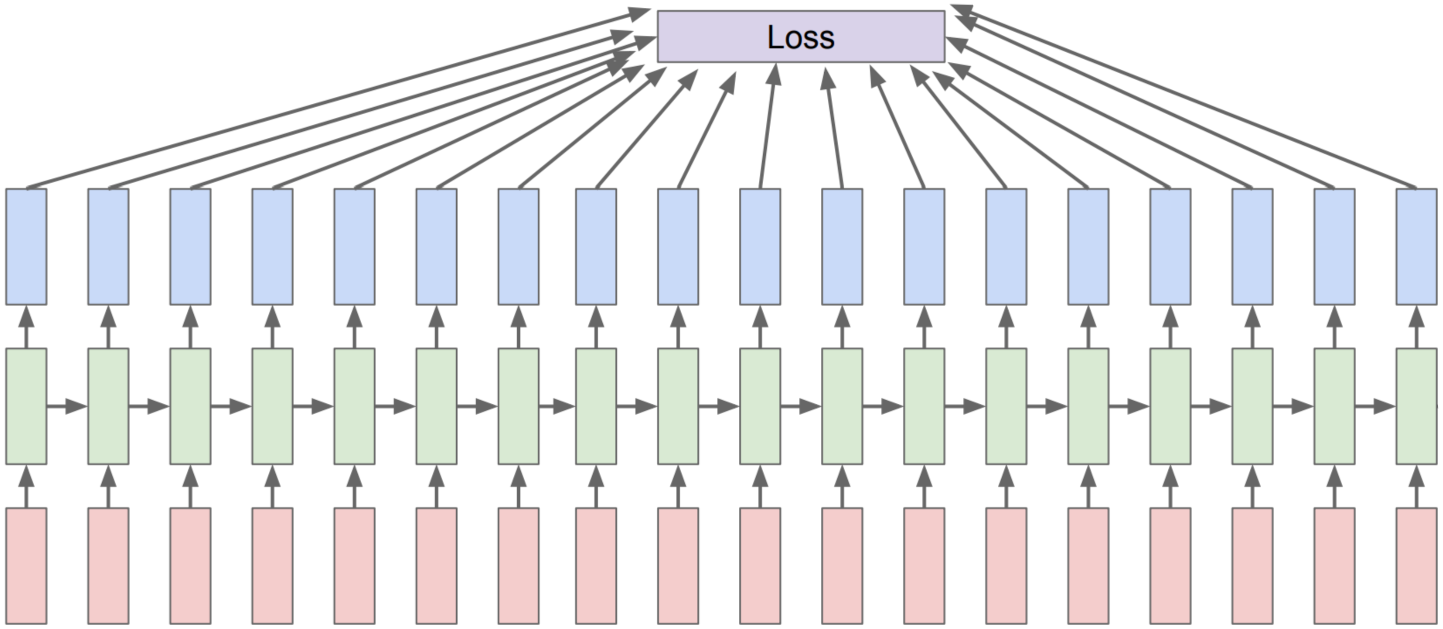
$$\boldsymbol{h}_t^{(f)} = \left[1, \theta\left(W_f^{(1)} \boldsymbol{x}_t^{(n)} + W_f \boldsymbol{h}_{t-1}\right)\right]^T \text{ and } \boldsymbol{h}_t^{(b)} = \left[1, \theta\left(W_b^{(1)} \boldsymbol{x}_t^{(n)} + W_b \boldsymbol{h}_{t+1}\right)\right]^T$$



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

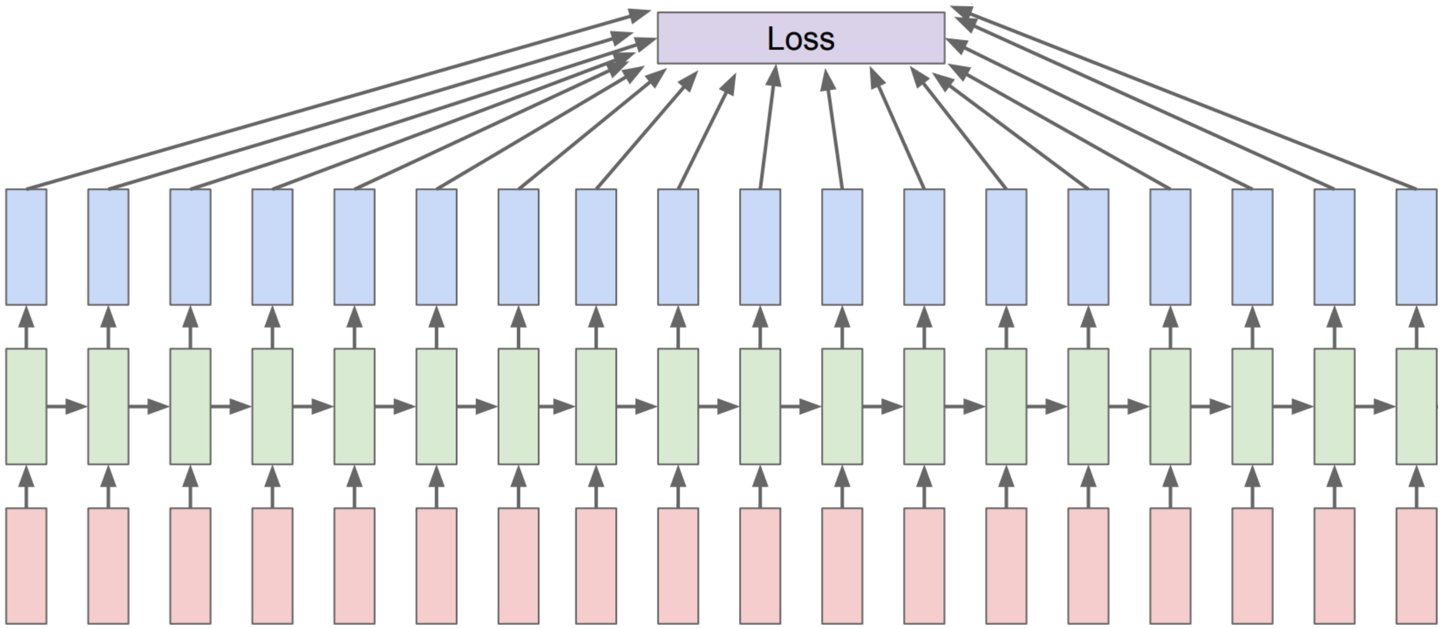


Forward pass to compute outputs and hidden representations

Backward pass to compute gradients

Source: http://cs231n.stanford.edu/slides/2023/lecture_8.pdf

Training RNNs: Challenges



Forward pass to compute outputs and hidden representations

Backward pass to compute gradients

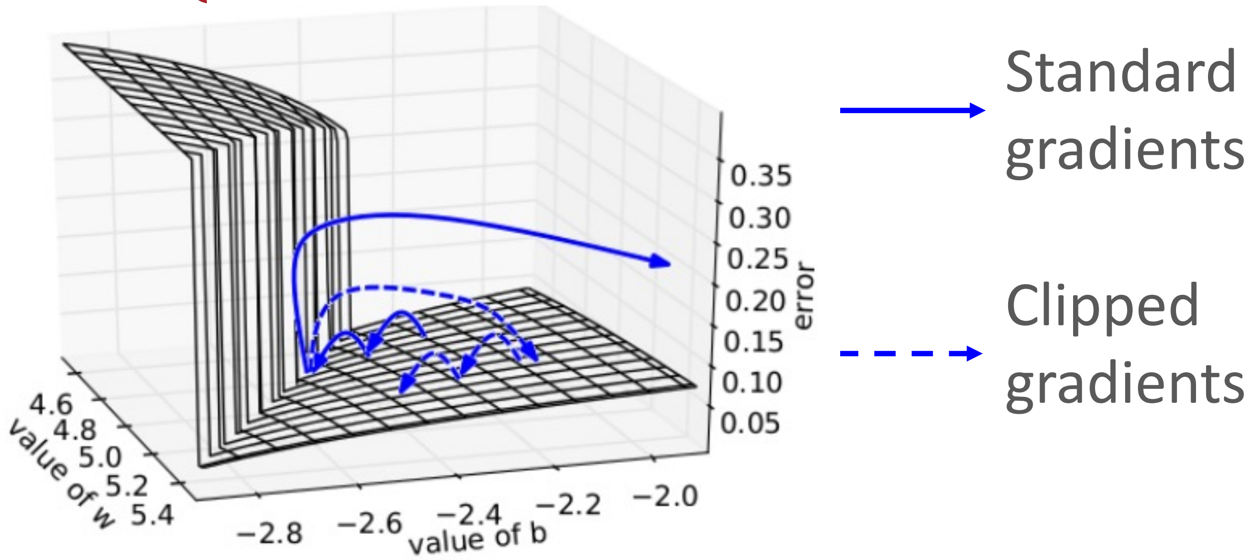
- Issue: as the sequence length grows, the gradient is more likely to explode or vanish

Source: http://cs231n.stanford.edu/slides/2023/lecture_8.pdf

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

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



Source: https://arxiv.org/pdf/1211.5063.pdf

Recall: Computing Gradients

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

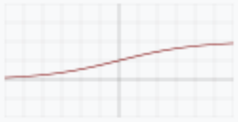
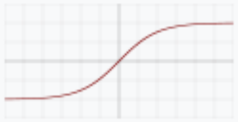
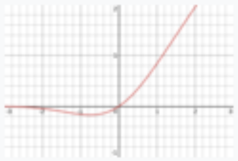
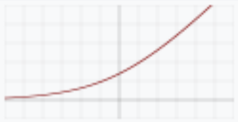
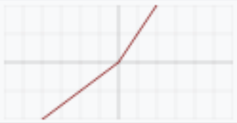
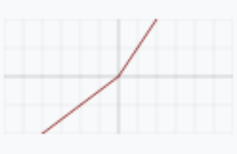
Chain rule: $\delta_b^{(l)} = \frac{\partial \ell^{(n)}}{\partial o_b^{(l)}} \left(\frac{\partial o_b^{(l)}}{\partial s_b^{(l)}} \right)$

$$o_b^{(l)} = \theta\left(s_b^{(l)}\right) \rightarrow \frac{\partial o_b^{(l)}}{\partial s_b^{(l)}} = \frac{\partial \theta\left(s_b^{(l)}\right)}{\partial s_b^{(l)}}$$

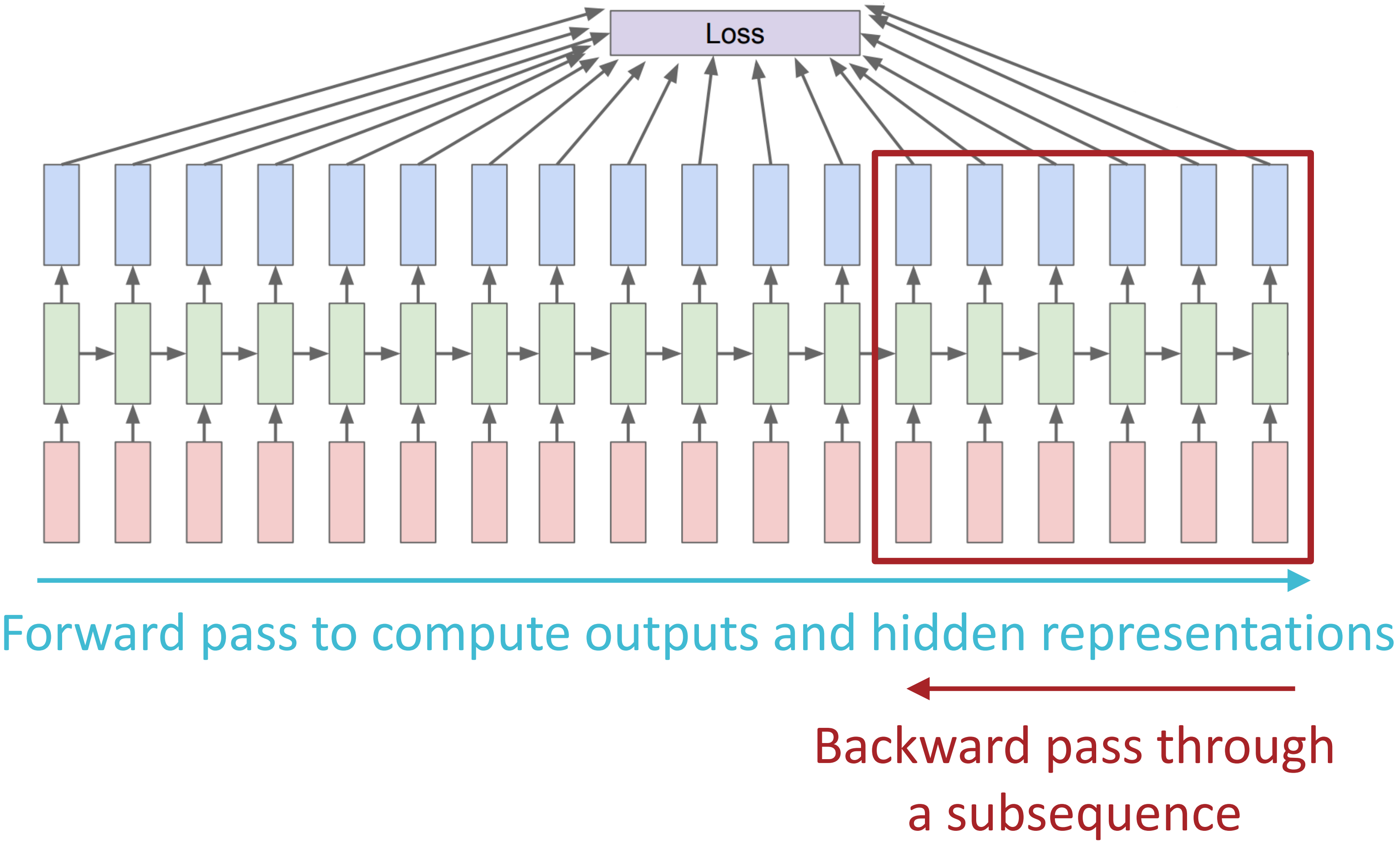
$$= 1 - \left(\tanh\left(s_b^{(l)}\right)\right)^2 \leq 1$$

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]		$\frac{1}{2}x\left(1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$ $= x\Phi(x)$
Softplus[8]		$\ln(1+e^x)$
Exponential linear unit (ELU)[9]		$\begin{cases} \alpha\left(e^x - 1\right) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ 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

Source: http://cs231n.stanford.edu/slides/2023/lecture_8.pdf

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
- Language models fit joint probability distributions to sequences of tokens
  - Tokenization and embedding to generate dense vector representations of texts
  - Can be sampled from to generate text