10-301/601: Introduction to Machine Learning Lecture 19 – Recurrent Neural Networks

Henry Chai

7/1/24

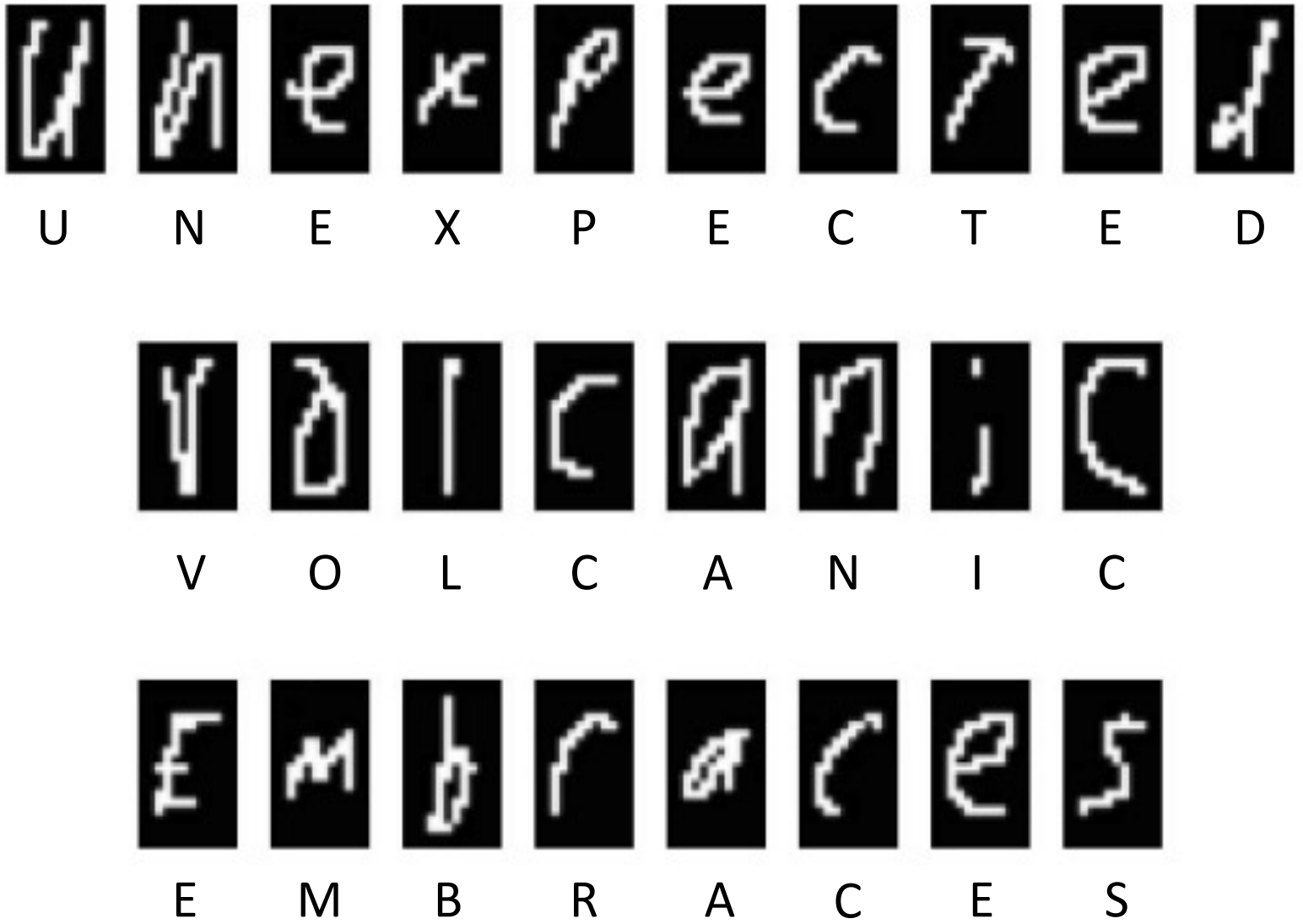
Front Matter

- Announcements
  - HW6 released 6/25, due 7/2 (tomorrow) at 11:59 PM
  - HW7 released 7/2 (tomorrow), due **7/11** at 11:59 PM
  - **Recitation on Wednesday (7/3)**
  - **No class on Thursday (7/4)**
- Recommended Readings
  - Zhang, Lipton, Li & Smola, Chapters 9 & 10

Recurrent Neural Networks

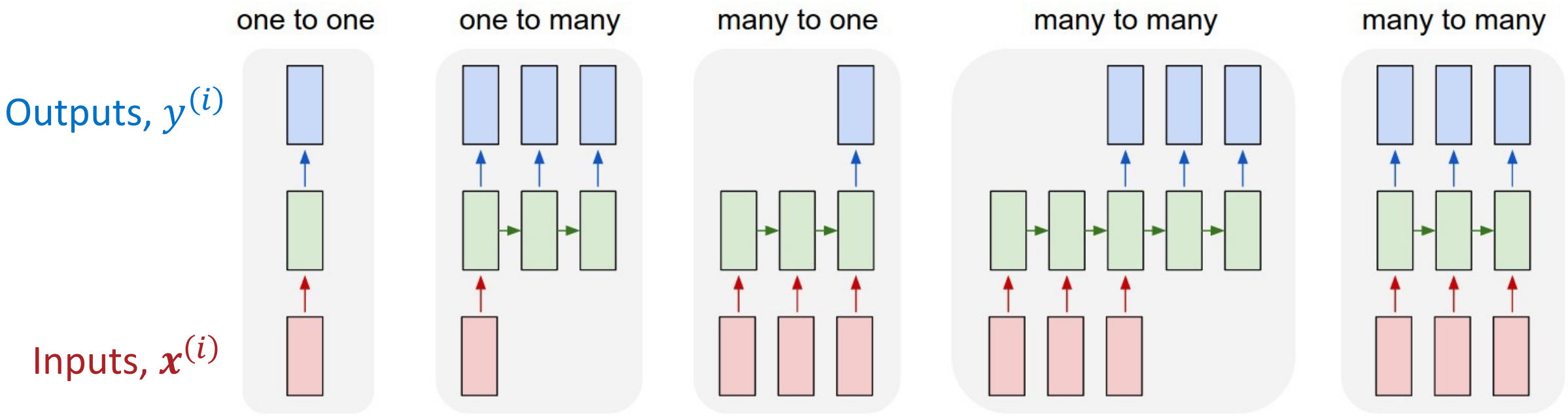
- Neural networks are frequently applied to inputs with some inherent temporal or sequential structure (e.g., text or video) of variable length
- Idea: use the information from previous parts of the input to inform subsequent predictions
- Insight: the hidden layers learn a useful representation (relative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

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



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

Sequential Data

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

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

One-to-one 0%

One-to-many 0%

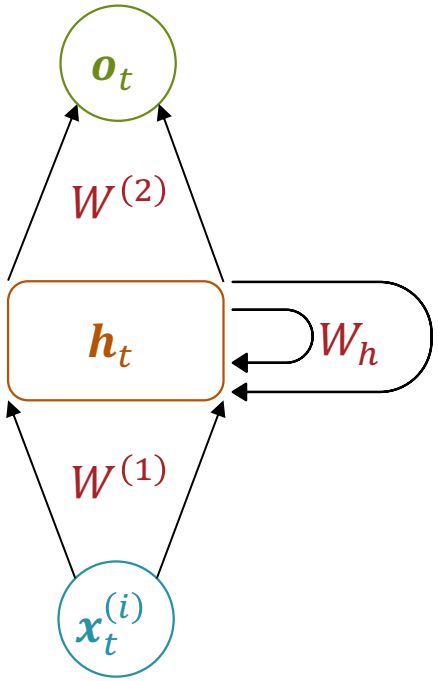
Many-to-one 0%

Many-to-many (offset) 0%

Many-to-many (synced) 0%

Recurrent Neural Networks

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



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

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

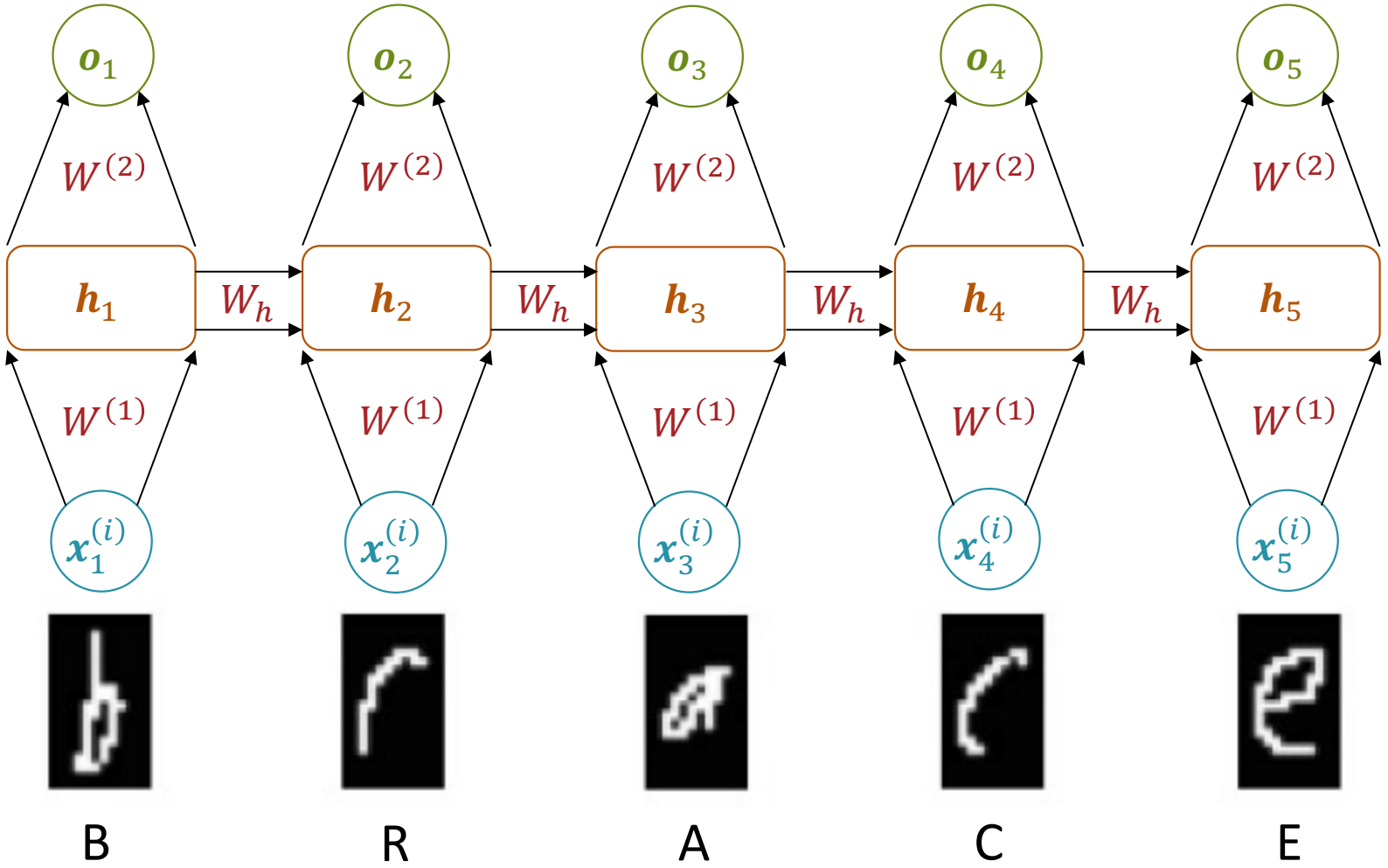
$$\boldsymbol{x}^{(n)} = \left[\boldsymbol{x}_1^{(n)}, \ldots, \boldsymbol{x}_{T_n}^{(n)}\right]$$

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

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

Unrolling Recurrent Neural Networks

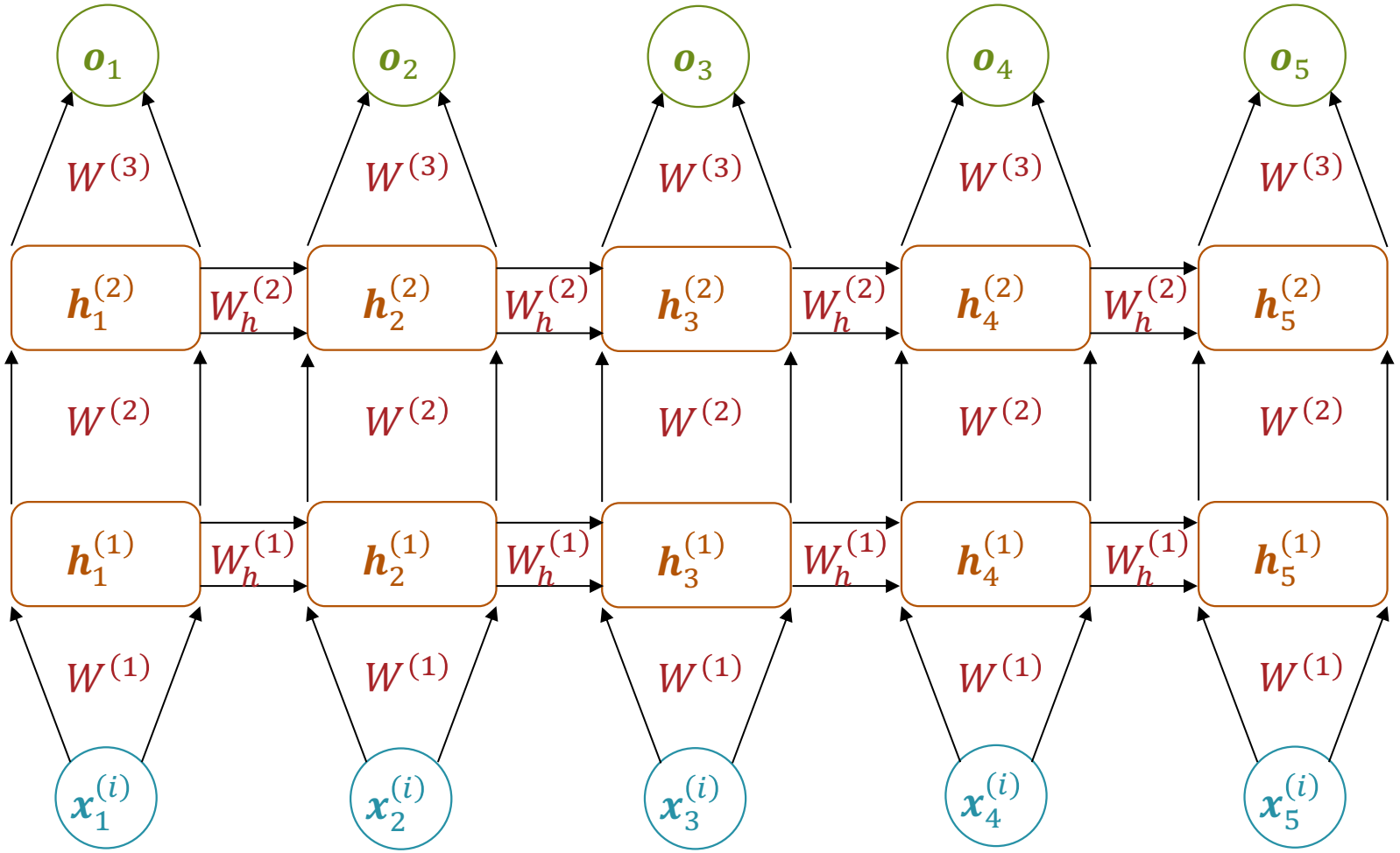
$$\boldsymbol{h}_t = \left[1, \theta\left(W^{(1)}\boldsymbol{x}_t^{(i)} + W_h\boldsymbol{h}_{t-1}\right)\right]^T \text{ and } \boldsymbol{o}_t = \hat{y}_t^{(i)} = \theta\left(W^{(2)}\boldsymbol{h}_t\right)$$



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

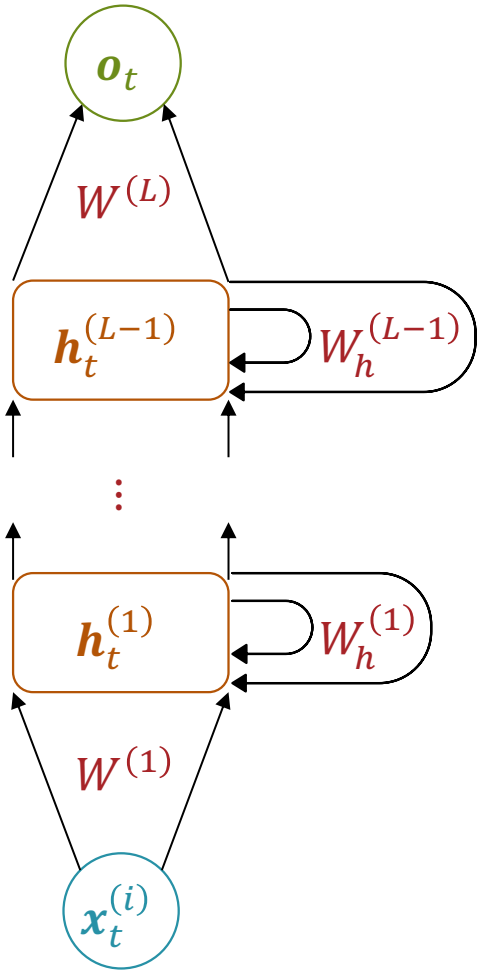
Deep Recurrent Neural Networks

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



Deep Recurrent Neural Networks

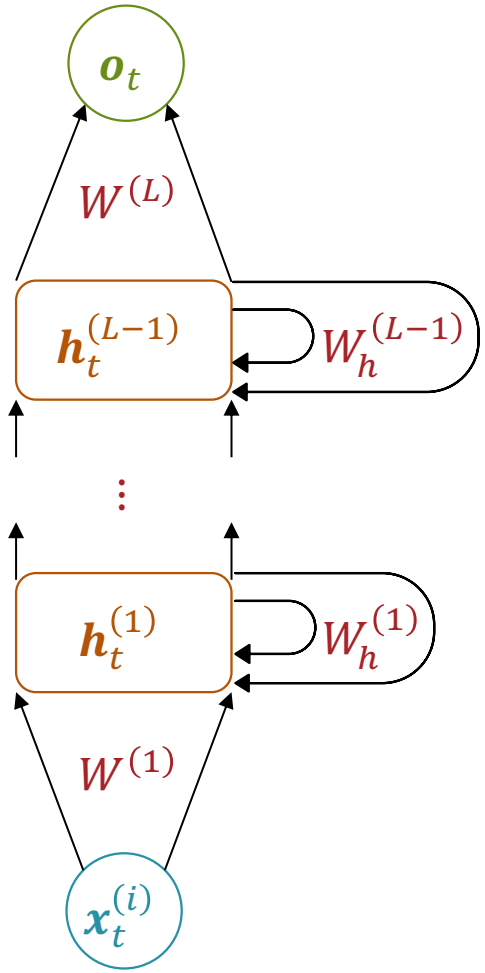
$$\boldsymbol{h}_t^{(l)} = \left[1, \theta\left(W^{(l)}\boldsymbol{h}_t^{(l-1)} + W_h^{(l)}\boldsymbol{h}_{t-1}^{(l)}\right)\right]^T \text{ and } \boldsymbol{o}_t = \hat{y}_t^{(i)} = \theta\left(W^{(L)}\boldsymbol{h}_t^{(L-1)}\right)$$



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?

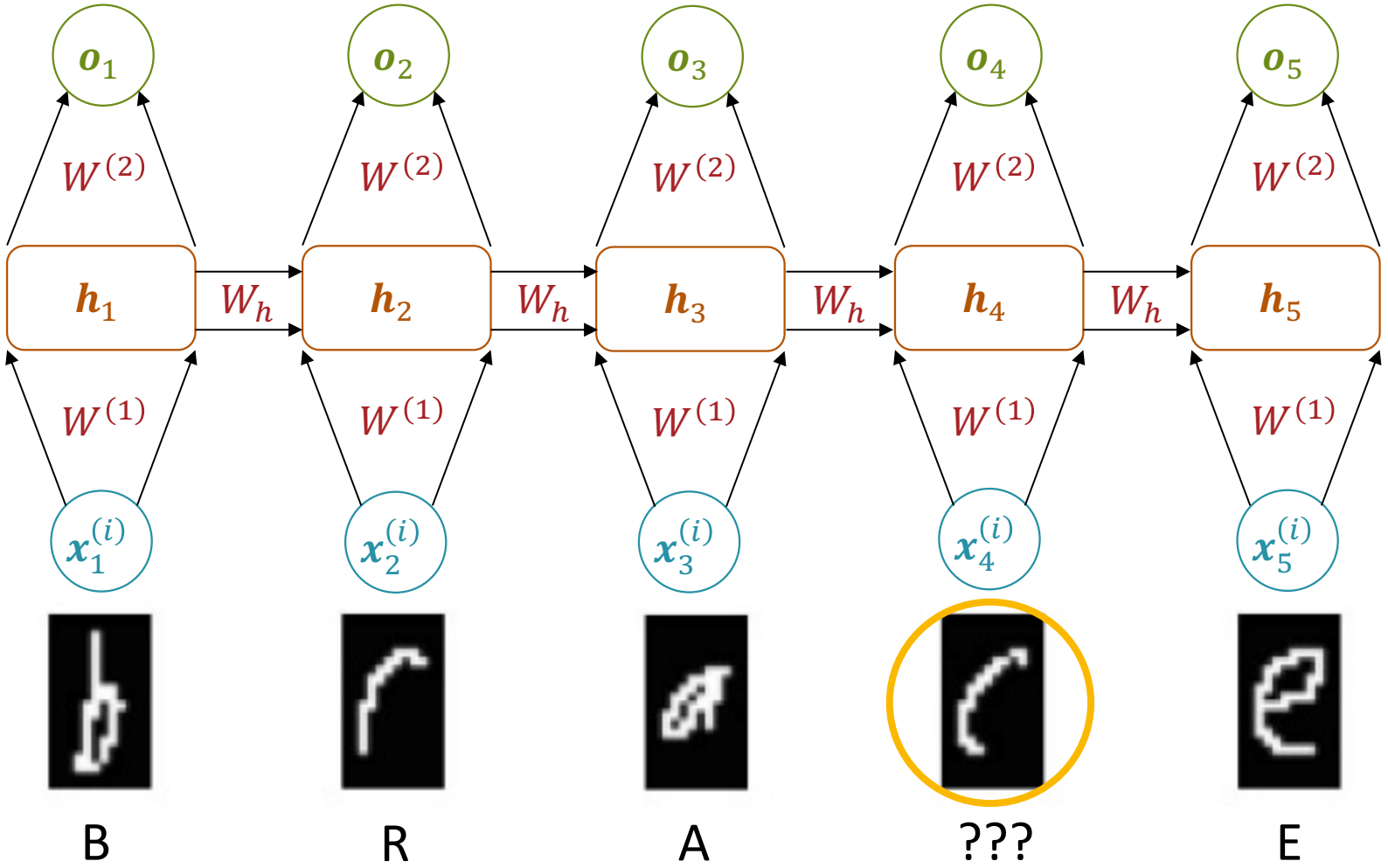
$$\boldsymbol{h}_t^{(l)} = \left[1, \theta\left(W^{(l)}\boldsymbol{h}_t^{(l-1)} + W_h^{(l)}\boldsymbol{h}_{t-1}^{(l)}\right)\right]^T \text{ and } \boldsymbol{o}_t = \hat{y}_t^{(i)} = \theta\left(W^{(L)}\boldsymbol{h}_t^{(L-1)}\right)$$



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?

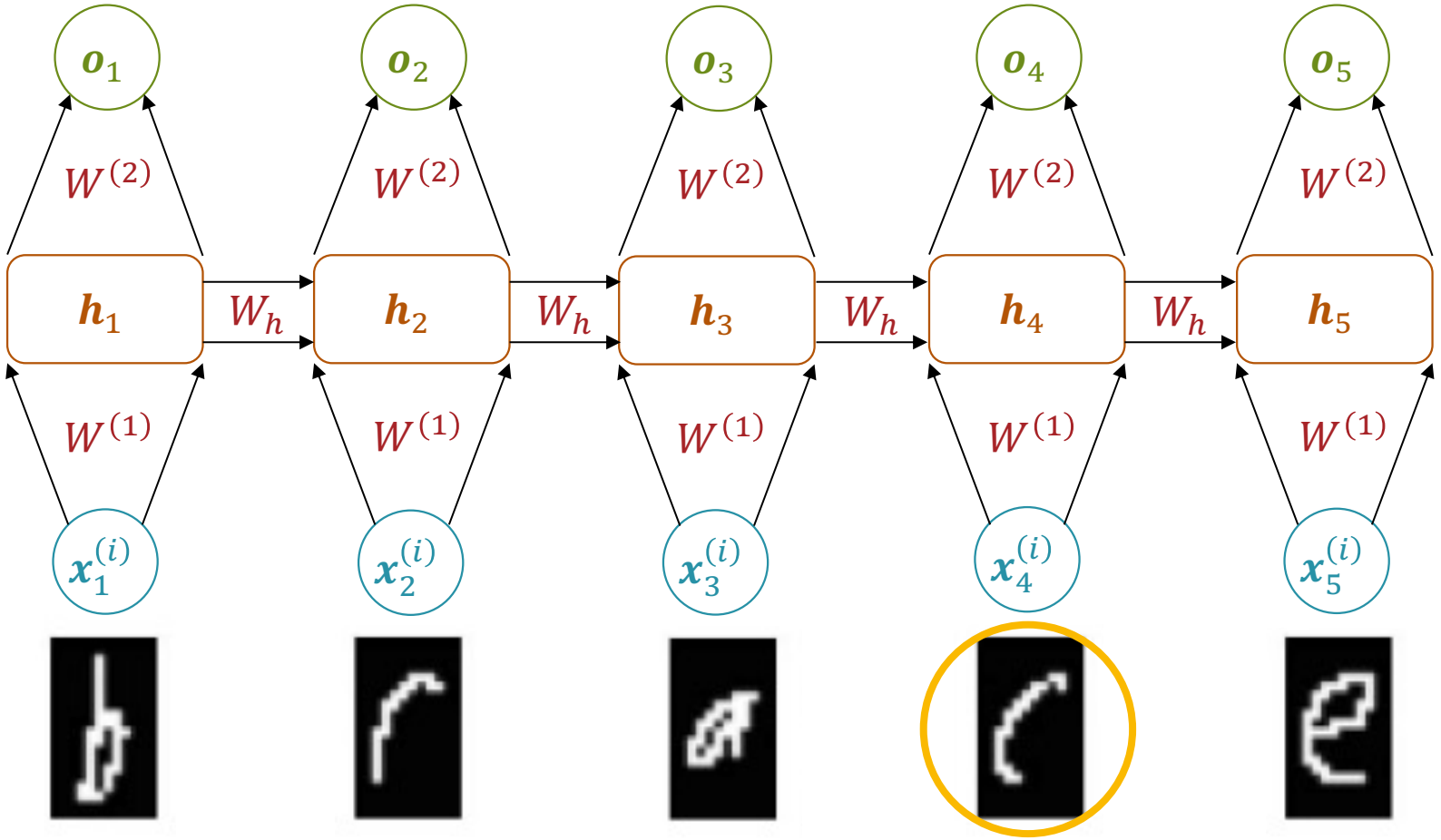
$$\boldsymbol{h}_t = \left[1, \theta\left(W^{(1)}\boldsymbol{x}_t^{(i)} + W_h\boldsymbol{h}_{t-1}\right)\right]^T \text{ and } \boldsymbol{o}_t = \hat{y}_t^{(i)} = \theta\left(W^{(2)}\boldsymbol{h}_t\right)$$



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?

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

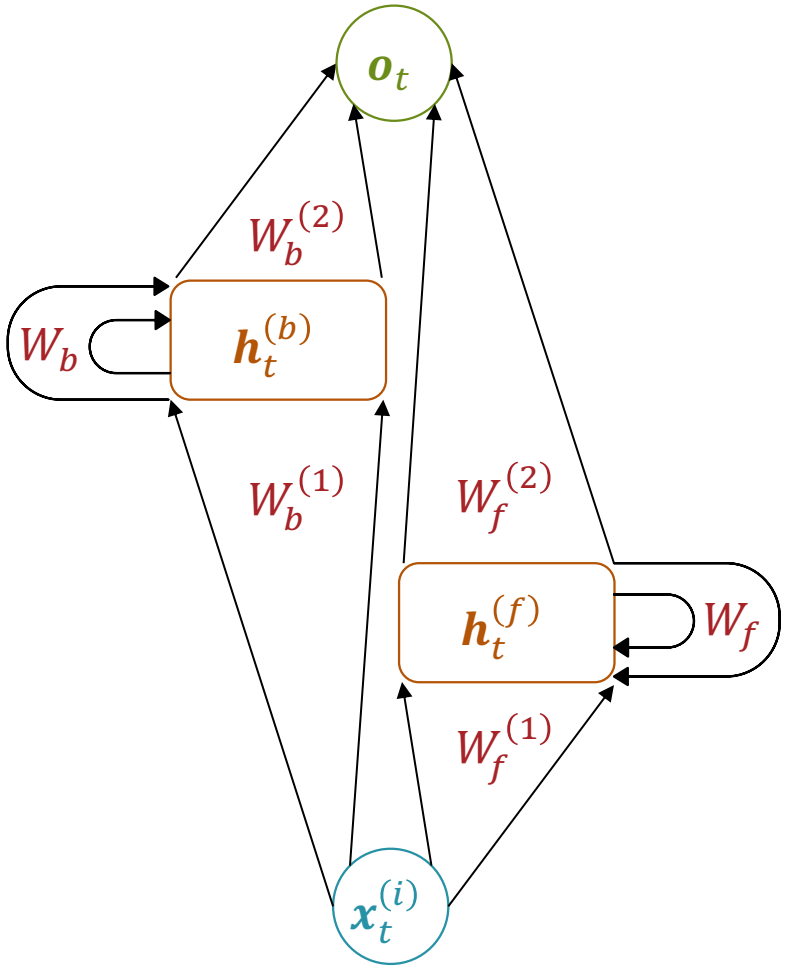


Source: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6319312

Bidirectional Recurrent Neural Networks

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

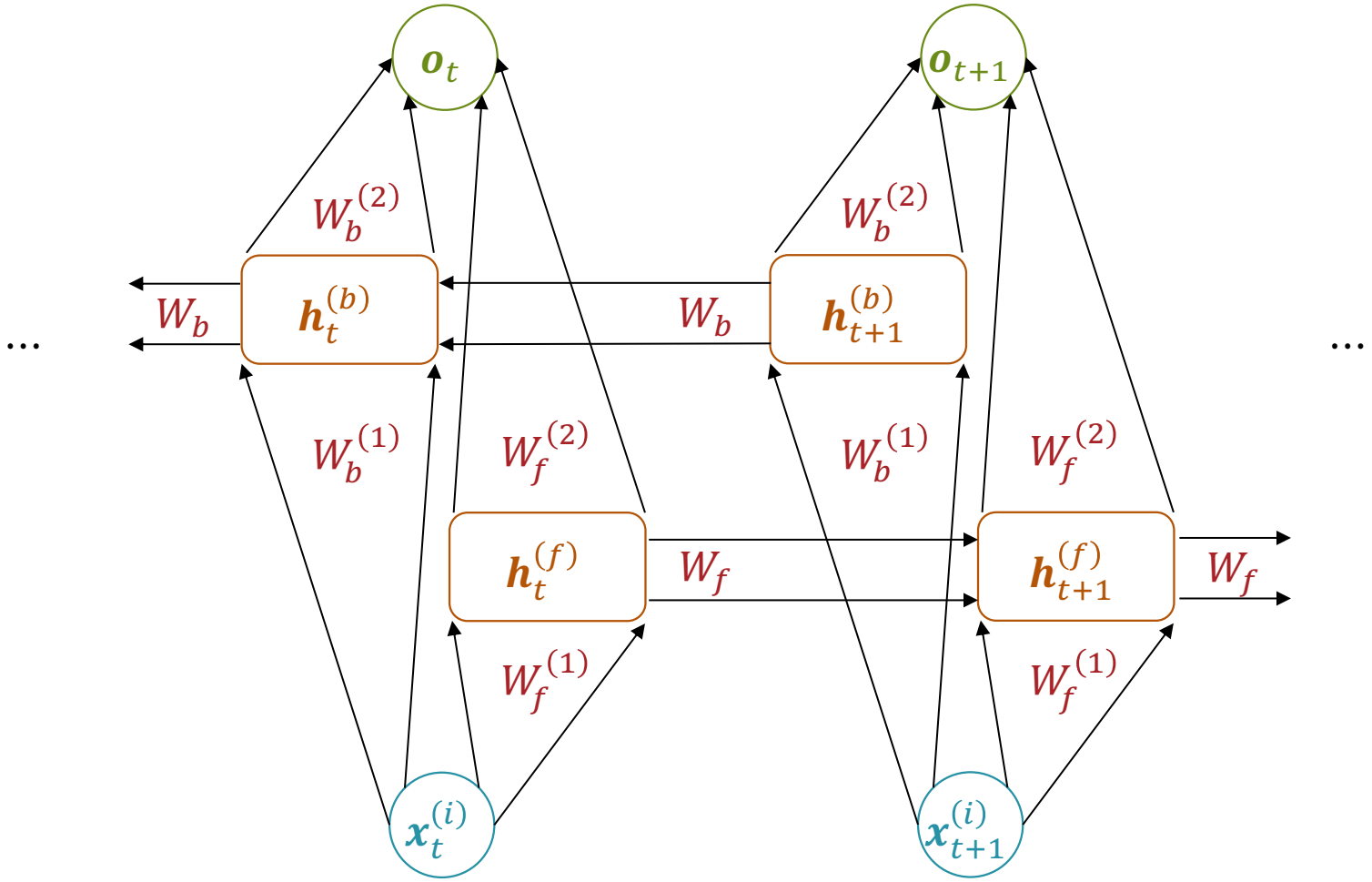
$$\boldsymbol{o}_t = \hat{y}_t^{(i)} = \theta\left(W_f^{(2)}\boldsymbol{h}_t^{(f)} + W_b^{(2)}\boldsymbol{h}_t^{(b)}\right)$$



Unrolling Bidirectional Recurrent Neural Networks

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

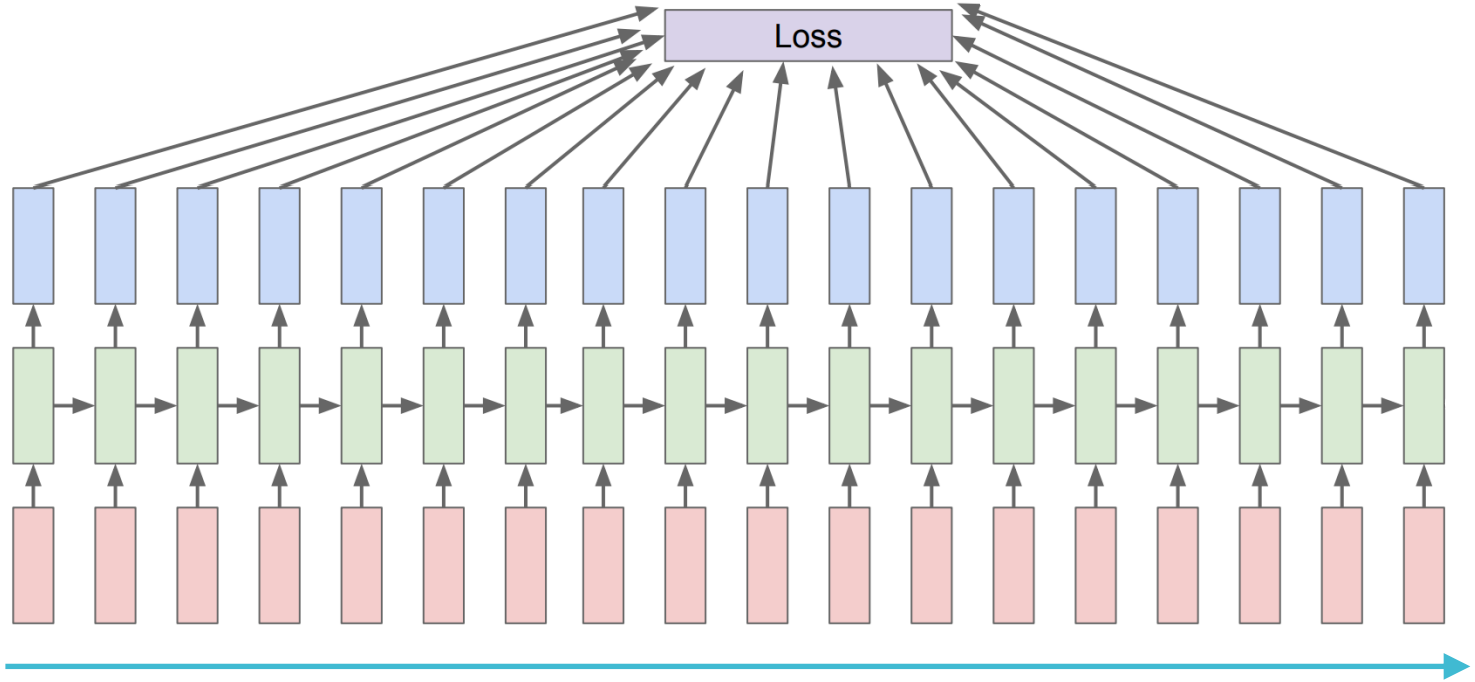
$$\boldsymbol{h}_t^{(f)} = \left[1, \theta\left(W_f^{(1)}\boldsymbol{x}_t^{(i)} + W_f\boldsymbol{h}_{t-1}\right)\right]^T \text{ and } \boldsymbol{h}_t^{(b)} = \left[1, \theta\left(W_b^{(1)}\boldsymbol{x}_t^{(i)} + W_b\boldsymbol{h}_{t+1}\right)\right]^T$$



Training RNNs

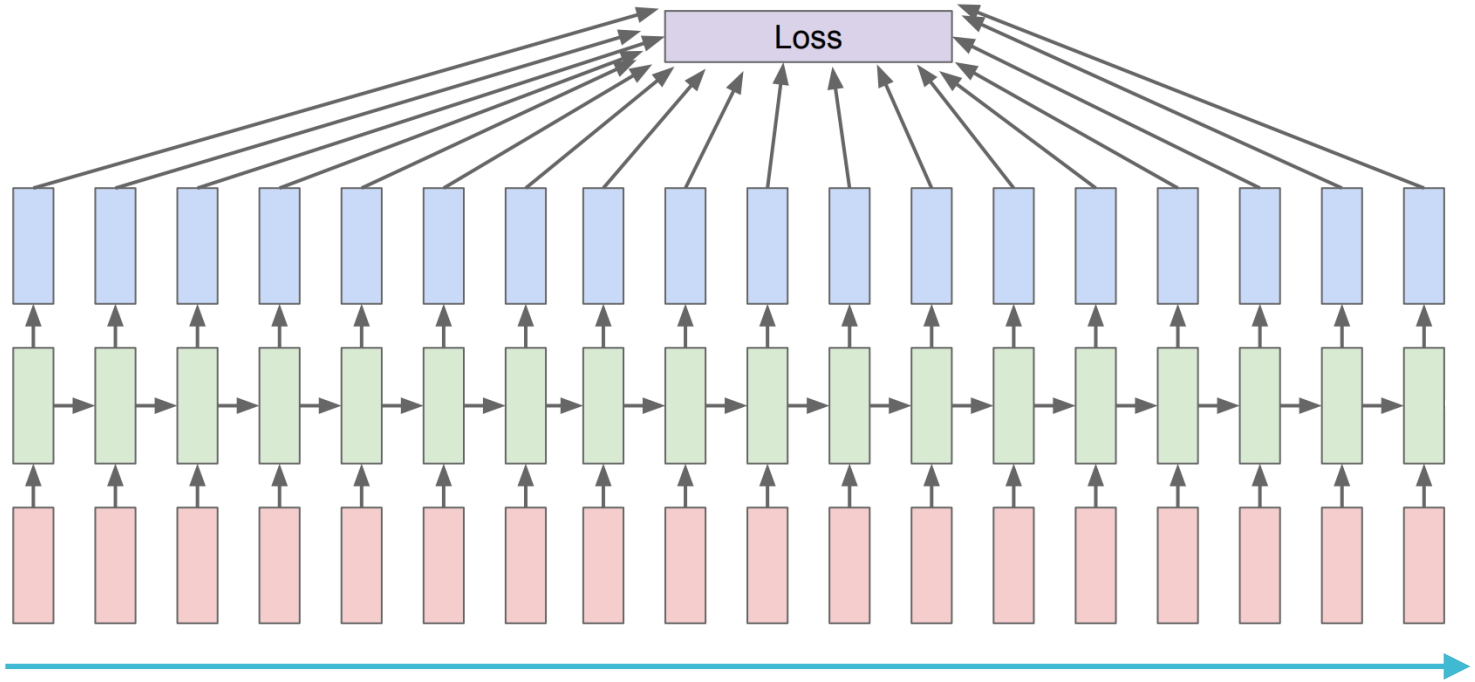
- A (deep/bidirectional) RNN simply represents a (somewhat complicated) computation graph
  - Weights are shared between different timesteps, significantly reducing the number of parameters to be learned!
- Can be trained using (stochastic) gradient descent/ backpropagation → “backpropagation through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

Training RNNs



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

Training RNNs: Challenges



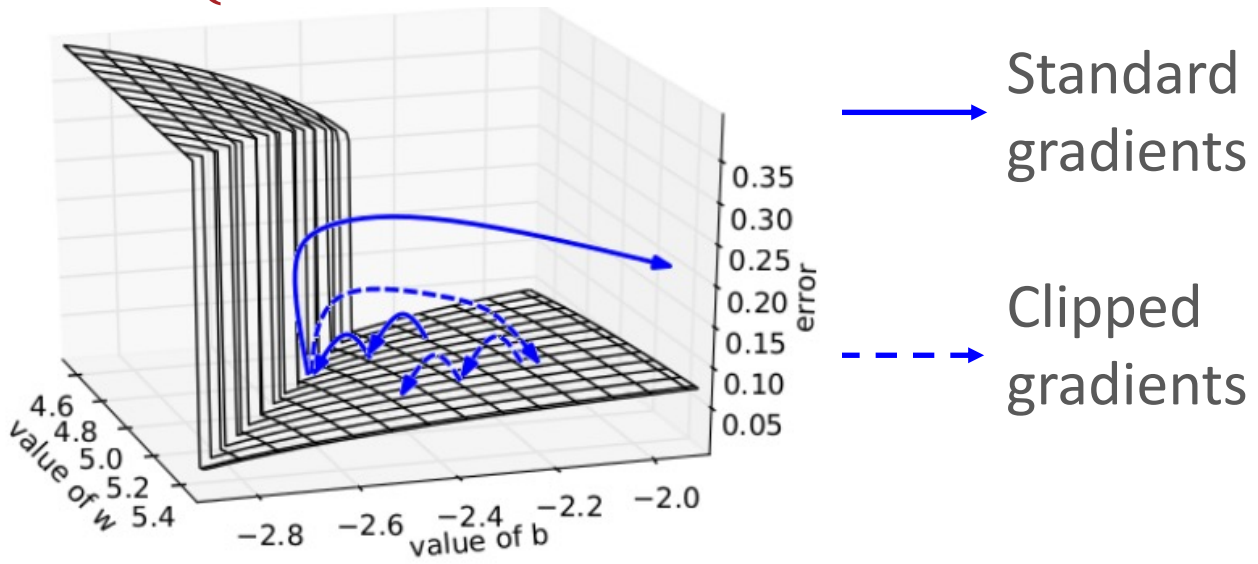
- Issue: as the sequence length grows, the gradient is more likely to explode or vanish

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

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



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

Recall: Computing Gradients

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

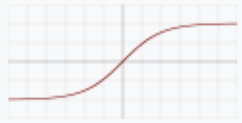
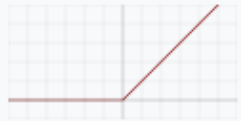
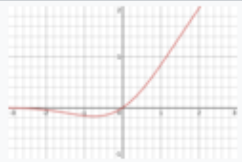
Chain rule: $\delta_b^{(l)} = \frac{\partial \ell^{(i)}}{\partial o_b^{(l)}} \left(\frac{\partial o_b^{(l)}}{\partial s_b^{(l)}} \right)$

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

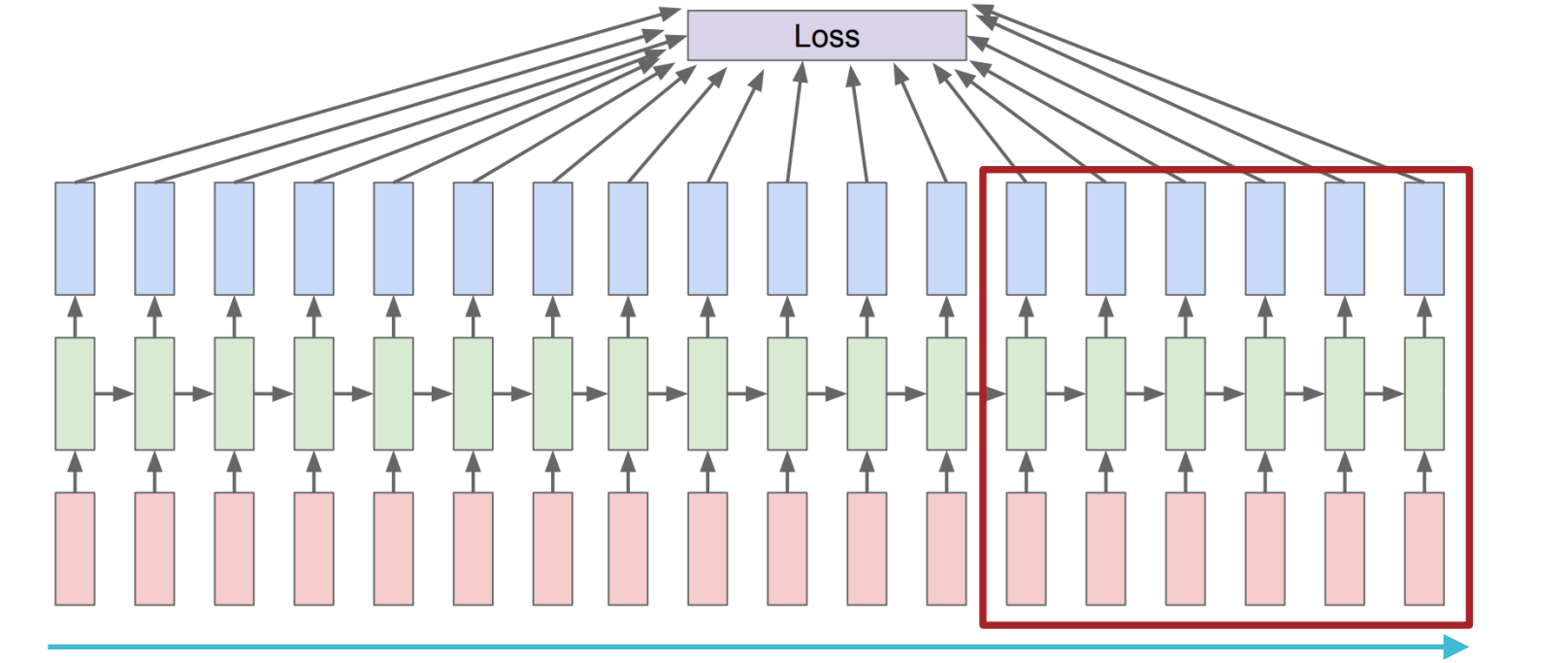
$$= 1 - \left(\tanh\left(s_b^{(l)}\right)\right)^2 \leq 1$$

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

Language Models

1. Convert raw text into sequence data

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

2. Learn or approximate a joint probability distribution over sequences

$$P\left(\boldsymbol{x}^{(i)}\right) = P\left(\boldsymbol{x}_1^{(i)}, \ldots, \boldsymbol{x}_{T_i}^{(i)}\right)$$

3. Sample from the implied conditional distribution to generate new sequences

$$P\left(\boldsymbol{x}_{T_i+1} \mid \boldsymbol{x}_1^{(i)}, \ldots, \boldsymbol{x}_{T_i}^{(i)}\right) = \frac{P\left(\boldsymbol{x}_1^{(i)}, \ldots, \boldsymbol{x}_{T_i}^{(i)}, \boldsymbol{x}_{T_i+1}\right)}{P\left(\boldsymbol{x}_1^{(i)}, \ldots, \boldsymbol{x}_{T_i}^{(i)}\right)}$$

Tokenization and Embedding

1. Convert raw text into sequence data

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

- High-level approach: split raw text into smaller units ("tokens"), then learn a dense, numerical vector representation ("embedding") for each token

Tokenization

- Example: “Henry is giving a lecture on language models”
- Idea: word-based tokenization

[“henry”, “is”, “giving”, “a”, “lecture”, “on”, “language”, “models”]

Tokenization

- Example: “Henry is givin’ a lectrue on LMs”
- Idea: word-based tokenization?

  [“henry”, “is”, ???, “a”, ???, “on”, “???”]

  - Can have difficulty trading off between vocabulary size and computational tractability
  - Similar words e.g., “model” and “models” can get mapped to completely disparate representations
  - Typos or acronyms will likely be out-of-vocabulary (OOV)

Tokenization

- Example: “Henry is givin’ a lectrue on LMs”
- Idea: character-based tokenization:

[“h”, “e”, “n”, “r”, “y”, “i”, “s”, “g”, “i”, “v”, “i”, “n”, “ ’ ”, …]

  - Much smaller vocabularies but a lot of semantic meaning is lost…
  - Sequences will be much longer than word-based tokenization, potentially causing computational issues
  - Can do well on logographic languages e.g., Kanji 漢字

Tokenization

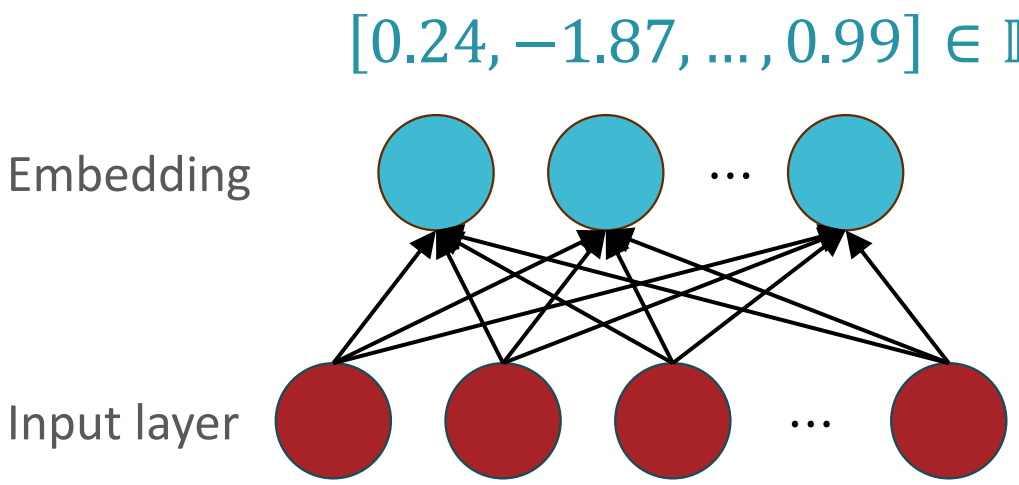
- Example: “Henry is givin’ a lectrue on LMs”
- Common practice: subword tokenization

[“henry”, “is”, “giv”, “##in”, “ ‘ ”, “a”, “lect” “##re”, “on”, “language”, “model”, “#s”]

  - Split long or rare words into smaller, semantically meaningful components or *subwords*

Embedding

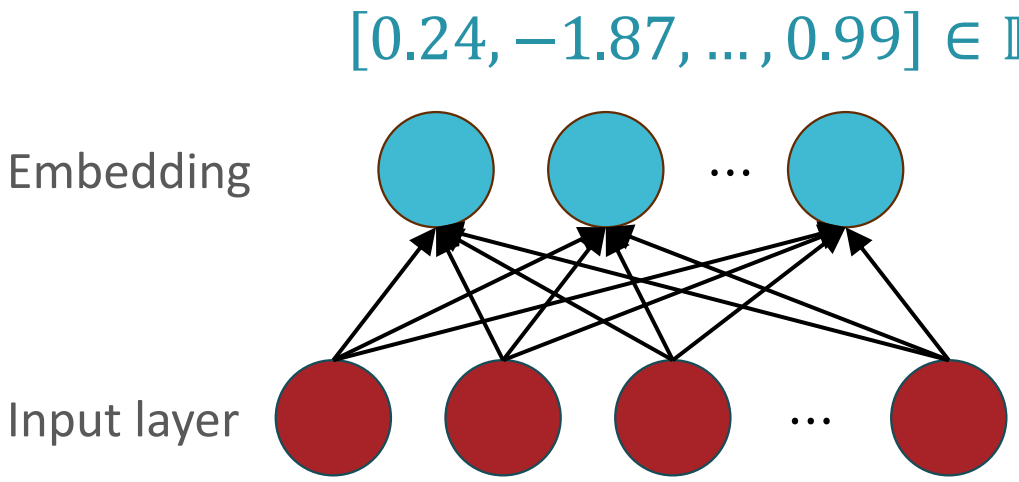
- Given a vocabulary V with $|V|$ tokens, learn an embedding by training a 1-layer, fully-connected feed-forward NN that takes one-hot encoded vectors as input



- Example: "is" $\rightarrow$ $[0, \quad 0, \quad 1, \quad \ldots, \quad 0] \in \mathbb{R}^{|V|}$

Okay but how do I go about training this neural network?

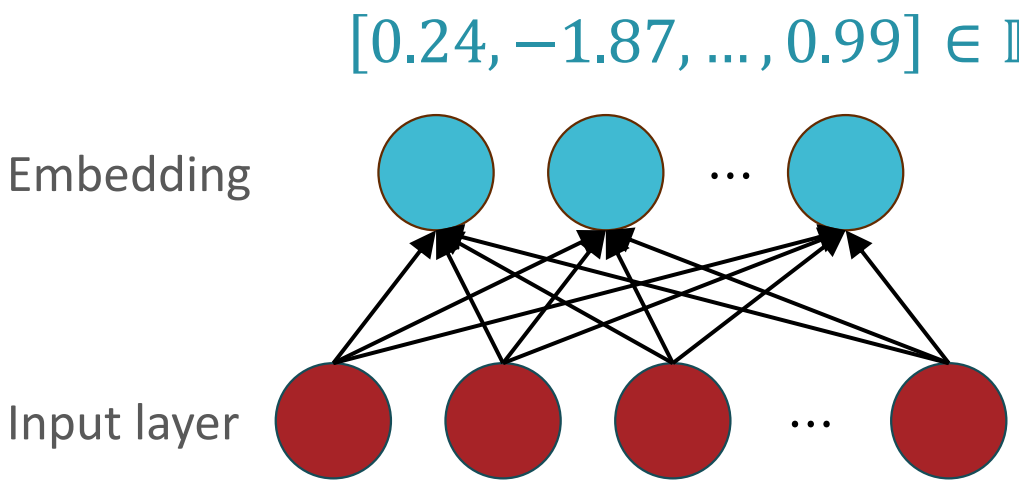
- Given a vocabulary V with $|V|$ tokens, learn an embedding by training a 1-layer, fully-connected feed-forward NN that takes one-hot encoded vectors as input



- Example: "is" $\rightarrow$ $[0, \quad 0, \quad 1, \quad \ldots, \quad 0] \in \mathbb{R}^{|V|}$

Embedding

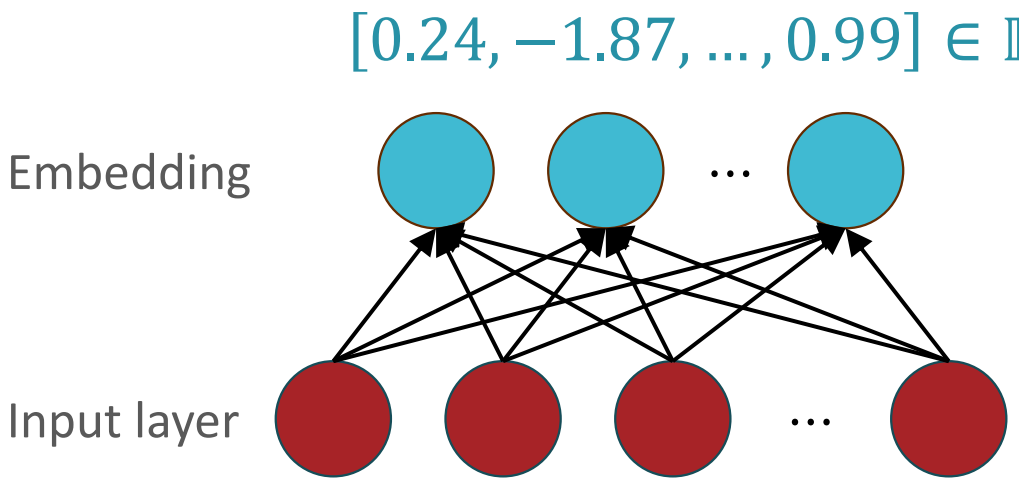
- Given a vocabulary V with $|V|$ tokens, learn an embedding by training a 1-layer, fully-connected feed-forward NN that takes one-hot encoded vectors as input



- Example: "is" $\rightarrow$ $[0, \quad 0, \quad 1, \quad \ldots, \quad 0] \in \mathbb{R}^{|V|}$
- Idea: use a pretrained embedding e.g., word2vec or GloVe
  - Requires you to use the same vocabulary/tokenization

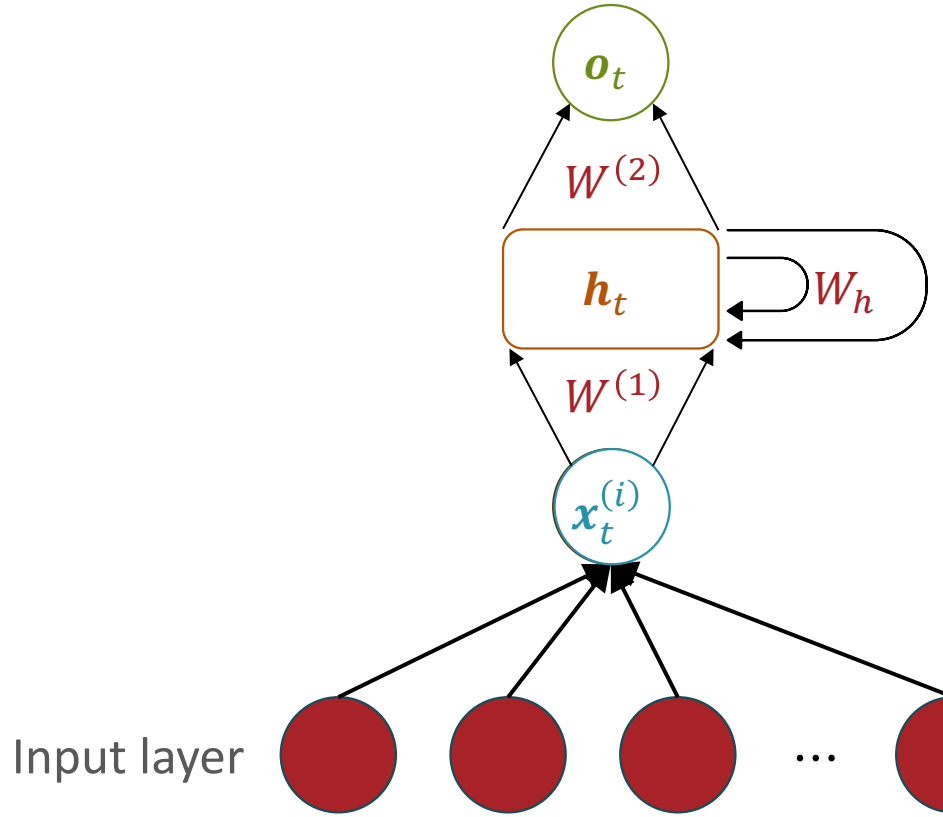
Embedding Layer

- Given a vocabulary V with $|V|$ tokens, learn an embedding by training a 1-layer, fully-connected feed-forward NN that takes one-hot encoded vectors as input



- Example: "is" $\rightarrow$ $[0, \quad 0, \quad 1, \quad \ldots, \quad 0] \in \mathbb{R}^{|V|}$
- Common practice: add this 1-layer NN to whatever architecture you're using and fit it to the task

Embedding Layer



- Example: "is" → $[0, \quad 0, \quad 1, \quad \ldots, \quad 0] \in \mathbb{R}^{|V|}$
- Common practice: add this 1-layer NN to whatever architecture you're using and fit it to the task

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