Recurrent Neural Networks

Yu Meng
University of Virginia
yumeng5@virginia.edu

Sep 25, 2024

UNIVERSITY of VIRGINIA

Join at
slido.com
#1931 885

Reminders

- Assignment 2 is due today!
- Assignment 3 has been released

UNIVERSITY of VIRGINIA

Join at
slido.com
#1931 885

Overview of Course Contents

- Week 1: Logistics & Overview
- Week 2: N-gram Language Models
- Week 3: Word Senses, Semantics & Classic Word Representations
- Week 4: Word Embeddings
- Week 5: Sequence Modeling and Neural Language Models
- Week 6-7: Language Modeling with Transformers (Pretraining + Fine-tuning)
- Week 8: Large Language Models (LLMs) & In-context Learning
- Week 9-10: Knowledge in LLMs and Retrieval-Augmented Generation (RAG)
- Week 11: LLM Alignment
- Week 12: Language Agents
- Week 13: Recap + Future of NLP
- Week 15 (after Thanksgiving): Project Presentations

(Recap) Word Embedding Limitations

- **Static representations (context independence)**: A word is always assigned a single vector representation regardless of its context
  - Words can have multiple meanings (polysemy)
  - Example: “bank” can mean a financial institution or the side of a river
- **Shallow representations**: Word embedding learning only focus on local context (a fixed window size of nearby words)
  - Cannot capture complex syntactic or long-range dependencies
  - Example: “The book that the president, who everyone admires, recommended is fascinating.” – distant subject (“book”) and adjective (“fascinating”)
- **Single-word representations**: Can only represent single words rather than larger linguistic units (phrases, sentences, paragraphs)
  - Many tasks require modeling relationships & compositionality between larger text chunks
  - Example: “They sell delicious hot dogs.” – “hot dogs” should be understood as an entire unit

(Recap) Sequence Modeling: Overview

- Use deep learning methods to understand, process, and generate **text sequences**
- Goals:
  - Learn context-dependent representations
  - Capture long-range dependencies
  - Handle complex relationships among large text units
- Sequence modeling architectures are based on deep neural networks (DNNs)!
  - Language exhibits hierarchical structures (e.g., letters form words, words form phrases, phrases form sentences)
  - DNNs learn multiple levels of abstraction across layers, allowing them to capture low-level patterns (e.g., word relations) in lower layers and high-level patterns (e.g., sentence meanings) in higher layers
  - Each layer in DNNs refines the word representations by considering contexts at different granularities (shorter & longer-range contexts), allowing for contextualized understanding of words and sequences

UNIVERSITY of VIRGINIA

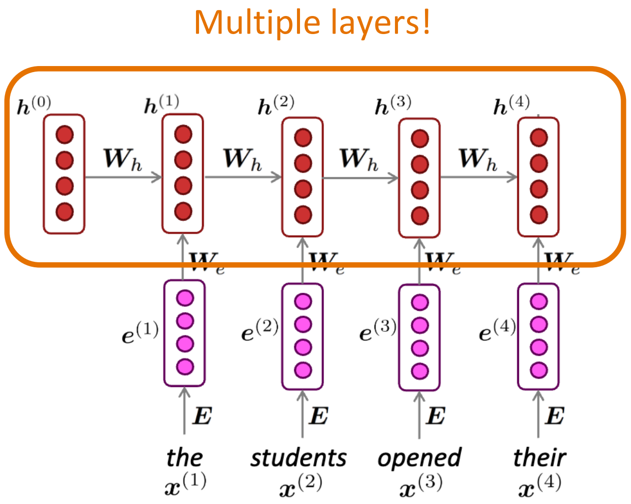
Join at
slido.com
#1931 885

(Recap) Sequence Modeling Architectures

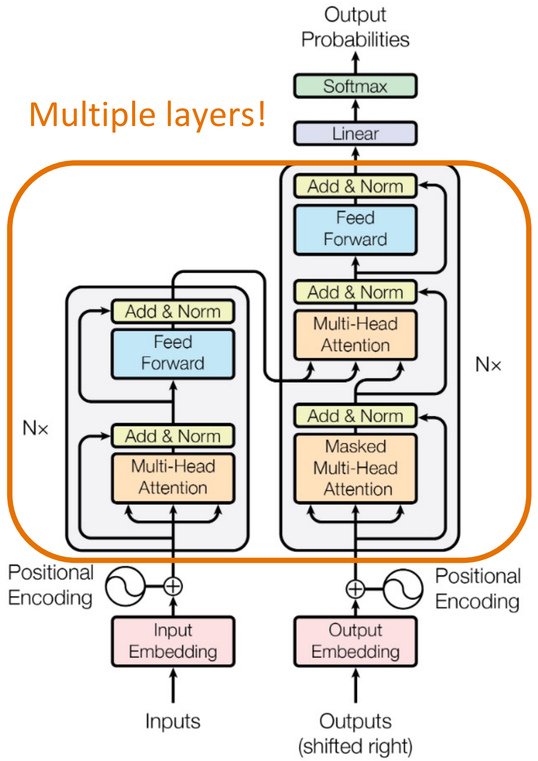
hidden states
$h^{(t)} = \sigma\left(W_h h^{(t-1)} + W_e e^{(t)} + b_1\right)$
$h^{(0)}$ is the initial hidden state

word embeddings
$e^{(t)} = E x^{(t)}$

words / one-hot vectors
$x^{(t)} \in \mathbb{R}^{|V|}$



RNN neural networks:
https://web.stanford.edu/class/cs224n/slides/cs224n-spr2024-lecture05-rnnlm.pdf



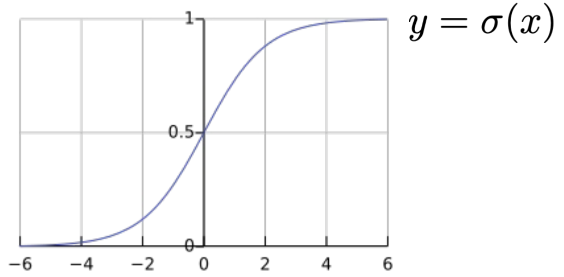
Transformer: https://arxiv.org/pdf/1706.03762

UNIVERSITY of VIRGINIA

Join at
slido.com
#1931 885

(Recap) Basic Neural Network Unit (Perceptron)

- Input: $\boldsymbol{x} = [x_1, x_2, x_3]$
- Model parameters (weights & bias): $\boldsymbol{w} = [w_1, w_2, w_3]$ & b
- Linear computation: $z = \boldsymbol{w} \cdot \boldsymbol{x} + b$
- Nonlinear activation: $a = \sigma(z)$



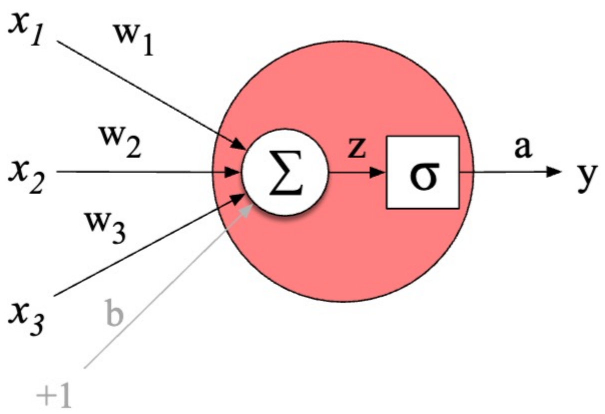
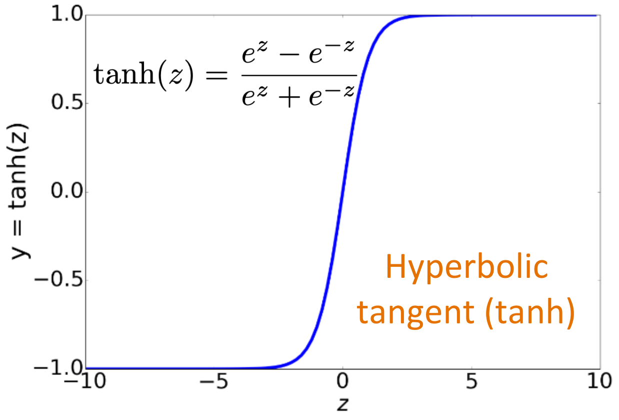


Figure source: https://web.stanford.edu/~jurafsky/slp3/7.pdf

Join at
slido.com
#1931 885

(Recap) Common Non-linear Activations

- Why non-linear activations?
- Stacking linear operations will only result in another linear operation
- We wish our network to model complex, non-linear relationships between inputs and outputs



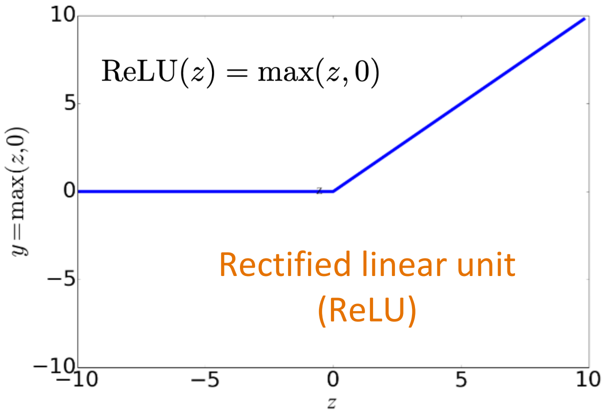


Figure source: https://web.stanford.edu/~jurafsky/slp3/7.pdf

Agenda

- Feedforward Network (FFN)
- Simple Neural Language Model
- Recurrent Neural Network (RNN)
- RNN Limitations
- Advanced RNNs

Feedforward Network (FFN)

- Feedforward network (FFN) = multilayer network where the outputs from units in each layer are passed to units in the next higher layer
- FFNs are also called multi-layer perceptrons (MLPs)
- Model parameters in each layer in FFNs: a weight matrix $\boldsymbol{W}$ and a bias vector $\boldsymbol{b}$
  - Each layer has multiple hidden units
  - Recall: a single hidden unit has as a weight vector and a bias parameters
  - Weight matrix: combining the weight vector for each unit
  - Bias vector: combining the bias for each unit

UNIVERSITY of VIRGINIA

Join at
slido.com
#1931 885

Example: 2-layer FFN

- Input: $\boldsymbol{x} = [x_1, x_2, \ldots, x_{n_0}]$
- Model parameters (weights & bias): $\boldsymbol{W} \in \mathbb{R}^{n_1 \times n_0}$, $\boldsymbol{U} \in \mathbb{R}^{n_2 \times n_1}$ & $\boldsymbol{b} \in \mathbb{R}^{n_1}$
- Forward computation:

First layer: $\boldsymbol{h} = \sigma(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$

Non-linear function (element-wise)

Second layer: $\boldsymbol{z} = \boldsymbol{U}\boldsymbol{h}$

Output: $\boldsymbol{y} = \text{softmax}(\boldsymbol{z})$

Convert to probability distribution

$$= \left[\frac{\exp(z_1)}{\sum_{j=1}^{n_2} \exp(z_j)}, \ldots, \frac{\exp(z_{n_2})}{\sum_{j=1}^{n_2} \exp(z_j)} \right]$$

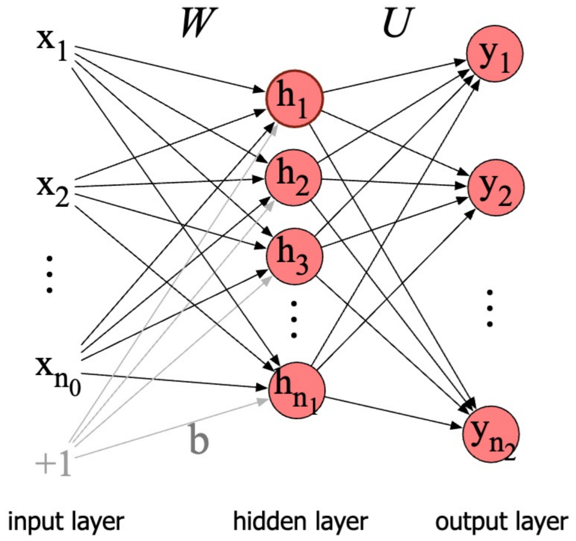
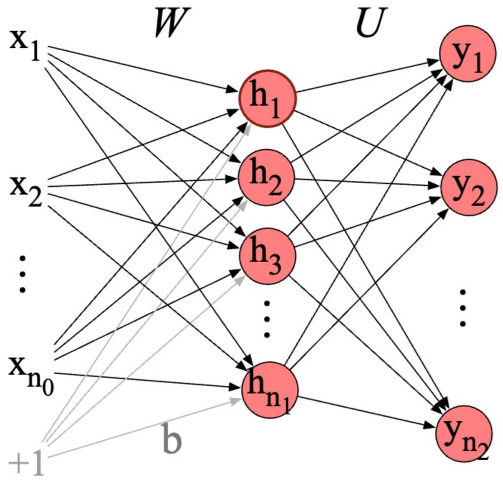
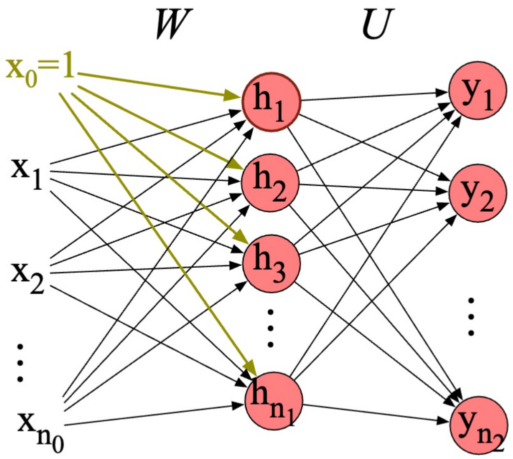


Figure source: https://web.stanford.edu/~jurafsky/slp3/7.pdf

Replacing the Bias Term

- In neural network computations, we often use a slightly simplified notation that represents exactly the same function without an explicit bias node
- We assume the input will always have a dummy node $x_0 = 1$





$$\boldsymbol{h} = \sigma(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$

$$\boldsymbol{x} = [x_1, x_2, \ldots, x_{n_0}] \quad \boldsymbol{W} \in \mathbb{R}^{n_1 \times n_0}$$

$$\boldsymbol{h} = \sigma(\boldsymbol{W}'\boldsymbol{x}')$$

$$\boldsymbol{x}' = [1, x_1, x_2, \ldots, x_{n_0}] \quad \boldsymbol{W}' \in \mathbb{R}^{n_1 \times (n_0+1)}$$

Training Objective

- We'll need a **loss function** that models the distance between the model output and the gold/desired output
- The common loss function for classification tasks is **cross-entropy** (CE) loss

K-way classification (K classes): $\mathcal{L}_{\mathrm{CE}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\sum_{k=1}^{K} y_k \log \hat{y}_k$

Model output probability

Ground-truth probability

Usually a one-hot vector (one dimension is 1; others are 0): $\boldsymbol{y} = [0, \ldots, 1, \ldots, 0]$

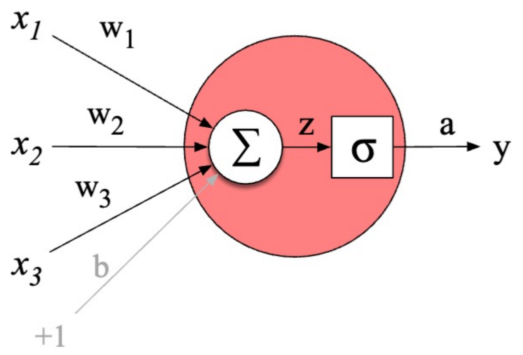
$$\mathcal{L}_{\mathrm{CE}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\log \hat{y}_c = -\log \frac{\exp(z_c)}{\sum_{j=1}^{K} \exp(z_j)}$$

Also called "negative log likelihood (NLL) loss"

c is the ground-truth class

Model Training (Forward Pass)

- Most optimization methods for DNNs are based on gradient descent
- First, randomly initialize model parameters
- In each optimization step, run two passes
    - **Forward pass**: evaluate the loss function given the input and current model parameters



$$z = \boldsymbol{w} \cdot \boldsymbol{x} + b$$
$$y = \sigma(z)$$

UNIVERSITY of VIRGINIA

Join at
slido.com
#1931 885

Model Training (Backward Pass)

- Most optimization methods for DNNs are based on gradient descent
- First, randomly initialize model parameters
- In each optimization step, run two passes
  - **Forward pass**: evaluate the loss function given the input and current model parameters
  - **Backward pass:** update the parameters following the opposite direction of the gradient

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \nabla_{\boldsymbol{w}} \mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y})$$

- Gradient computed via the chain rule $\nabla_{\boldsymbol{w}} \mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}} \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{w}}$

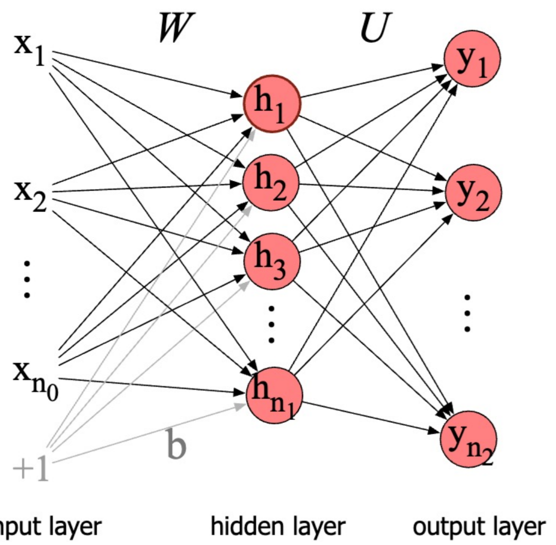
Gradient computation taken care of by deep learning libraries
(e.g., PyTorch)

Agenda

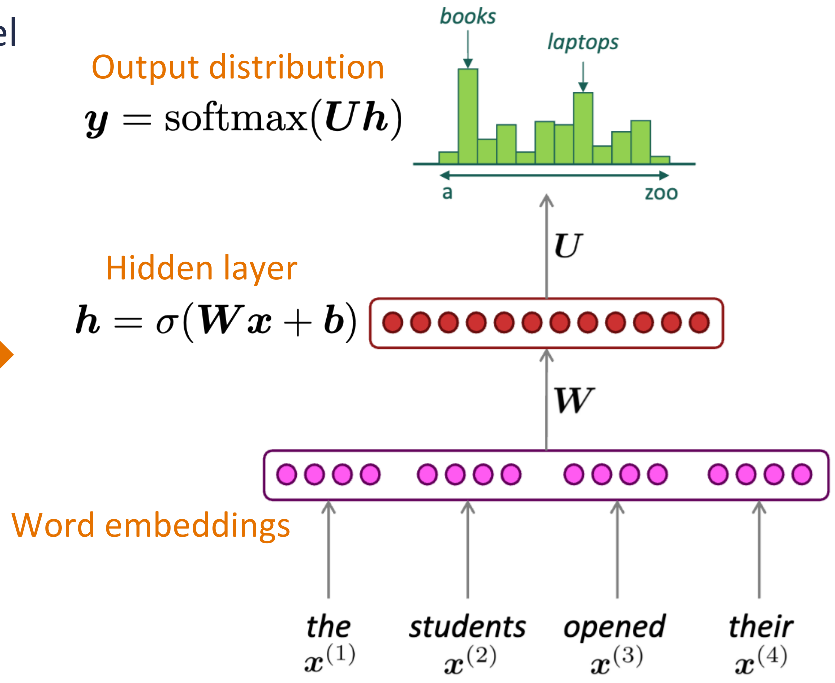
- Feedforward Network (FFN)
- Simple Neural Language Model
- Recurrent Neural Network (RNN)
- RNN Limitations
- Advanced RNNs

Simple Neural Language Model

Instantiate FFN as a neural language model



2-layer FFN



2-layer neural language model

Figure source: https://web.stanford.edu/class/cs224n/slides/cs224n-spr2024-lecture05-rnnlm.pdf

UNIVERSITY of VIRGINIA

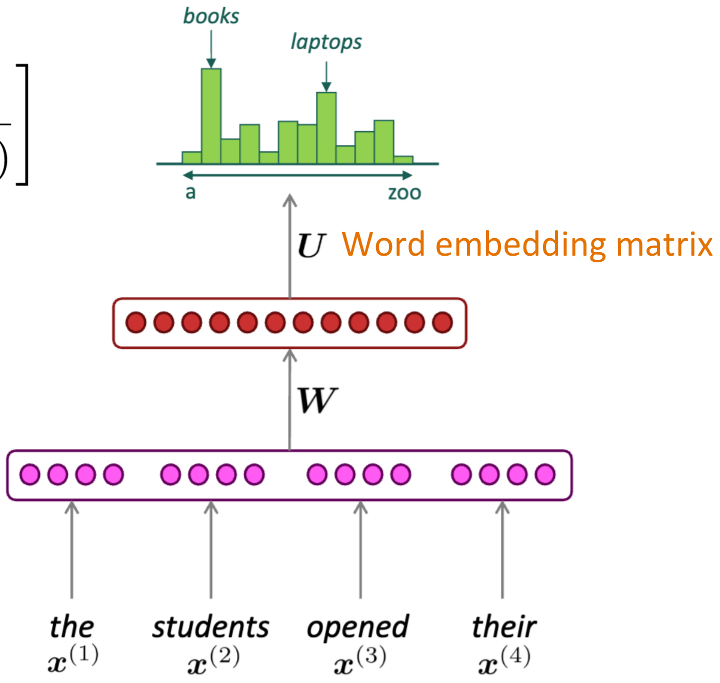
Join at slido.com #1931 885

Benefits of Neural Language Models

Output distribution

$$\boldsymbol{y} = \mathrm{softmax}(\boldsymbol{U}\boldsymbol{h}) = \underbrace{\left[\frac{\exp(\boldsymbol{u}_1 \cdot \boldsymbol{h})}{\sum_{j=1}^{|\mathcal{V}|} \exp(z_j)}, \dots, \frac{\exp(\boldsymbol{u}_{|\mathcal{V}|} \cdot \boldsymbol{h})}{\sum_{j=1}^{\mathcal{V}} \exp(z_j)}\right]}_{|\mathcal{V}|\text{-dimensions}}$$

- Address sparsity issue:
  - Strictly positive probability on every token in the vocabulary
  - Semantically similar words tend to have similar probabilities



books
laptops
a
zoo
$\boldsymbol{U}$ Word embedding matrix
$\boldsymbol{W}$
the $\boldsymbol{x}^{(1)}$
students $\boldsymbol{x}^{(2)}$
opened $\boldsymbol{x}^{(3)}$
their $\boldsymbol{x}^{(4)}$

Join at
slido.com
#1931 885

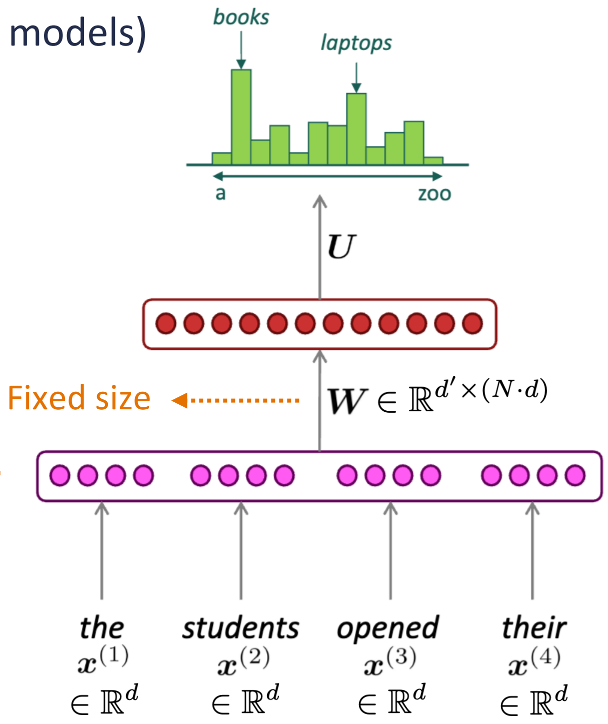
Limitations of (Simple) Neural Language Models

- Context window is fixed (same as N-gram language models)
- Increasing N will enlarge $\boldsymbol{W}$

Concatenated word embeddings

$\boldsymbol{x} = \boldsymbol{x}^{(1)} \oplus \boldsymbol{x}^{(2)} \oplus \cdots \oplus \boldsymbol{x}^{(N)} \in \mathbb{R}^{N \cdot d}$

Fixed size



books
laptops
a
zoo
$\boldsymbol{U}$
$\boldsymbol{W} \in \mathbb{R}^{d' \times (N \cdot d)}$
the $\boldsymbol{x}^{(1)} \in \mathbb{R}^d$
students $\boldsymbol{x}^{(2)} \in \mathbb{R}^d$
opened $\boldsymbol{x}^{(3)} \in \mathbb{R}^d$
their $\boldsymbol{x}^{(4)} \in \mathbb{R}^d$

Agenda

- Feedforward Network (FFN)
- Simple Neural Language Model
- Recurrent Neural Network (RNN)
- RNN Limitations
- Advanced RNNs

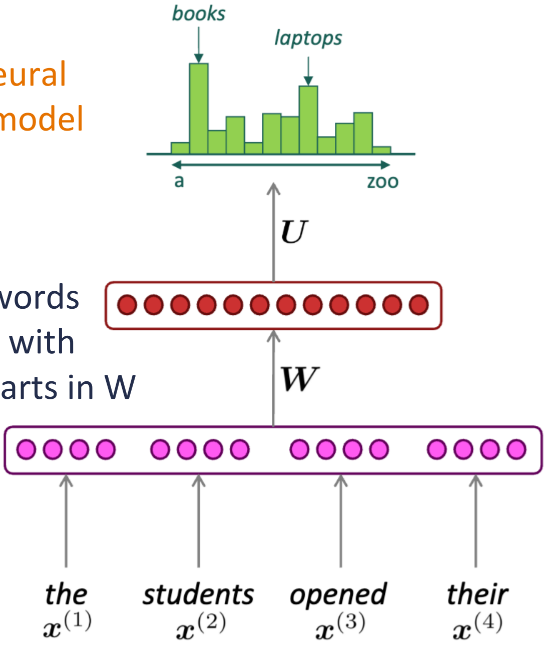
Join at
slido.com
#1931 885

Recurrent Neural Network (RNN) Overview

A neural language model that can process inputs of arbitrary lengths

Simple neural language model

Different words multiplied with different subparts in W



Recurrent neural language model

Reuse the same weights for all words

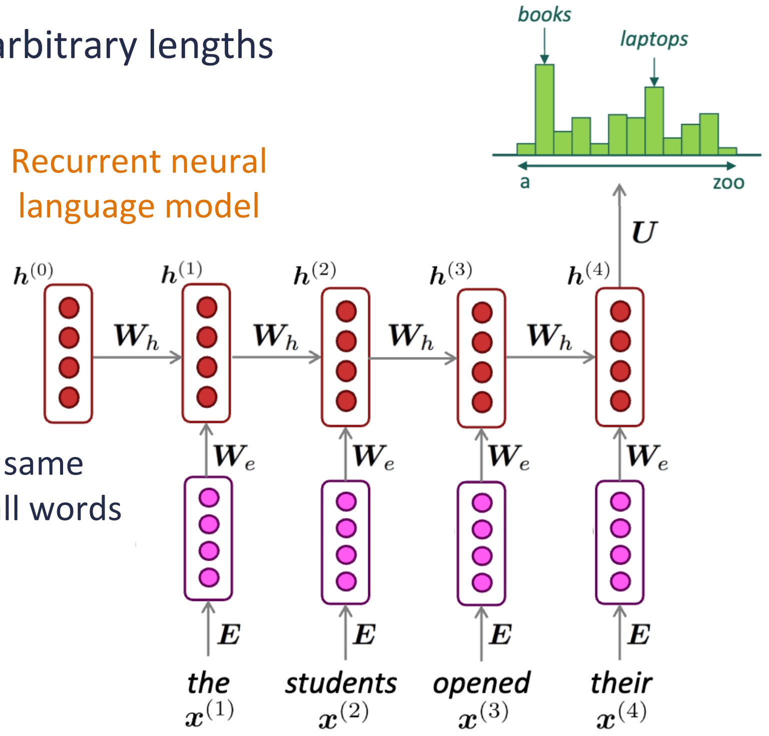


Figure source: https://web.stanford.edu/class/cs224n/slides/cs224n-spr2024-lecture05-rnnlm.pdf

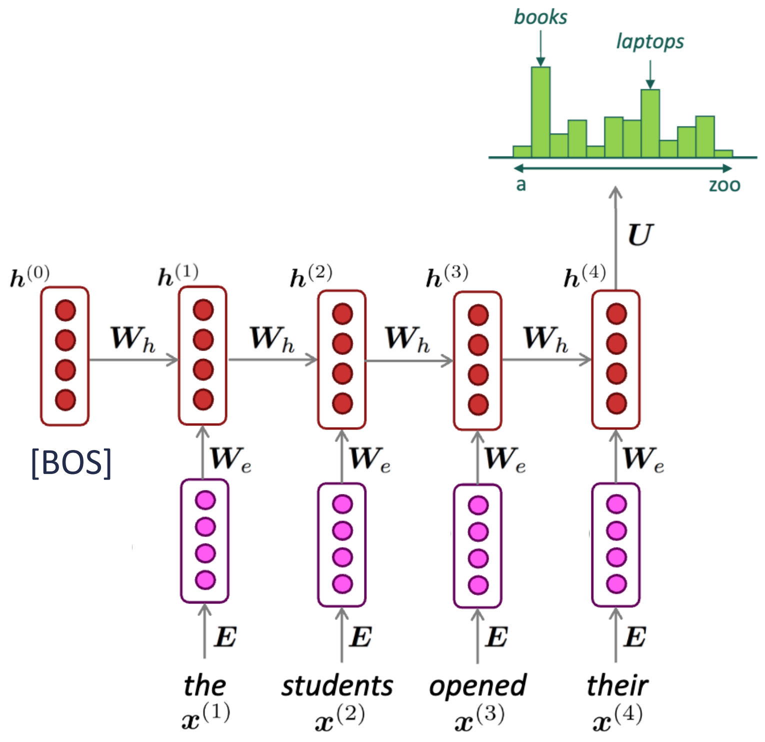
RNN Computation

- Hidden states in RNNs are computed based on
  - The hidden state at the previous step (memory)
  - The word embedding at the current step
- Parameters:
  - $\boldsymbol{W}_h$: weight matrix for the recurrent connection
  - $\boldsymbol{W}_e$: weight matrix for the input connection

$$\boldsymbol{h}^{(t)} = \sigma\left(\underbrace{\boldsymbol{W}_h\boldsymbol{h}^{(t-1)}}_{\text{Hidden states at the previous word (time step)}} + \underbrace{\boldsymbol{W}_e\boldsymbol{x}^{(t)}}_{\text{Word embedding of the current word (time step)}}\right)$$

Hidden states at the previous word (time step)

Word embedding of the current word (time step)



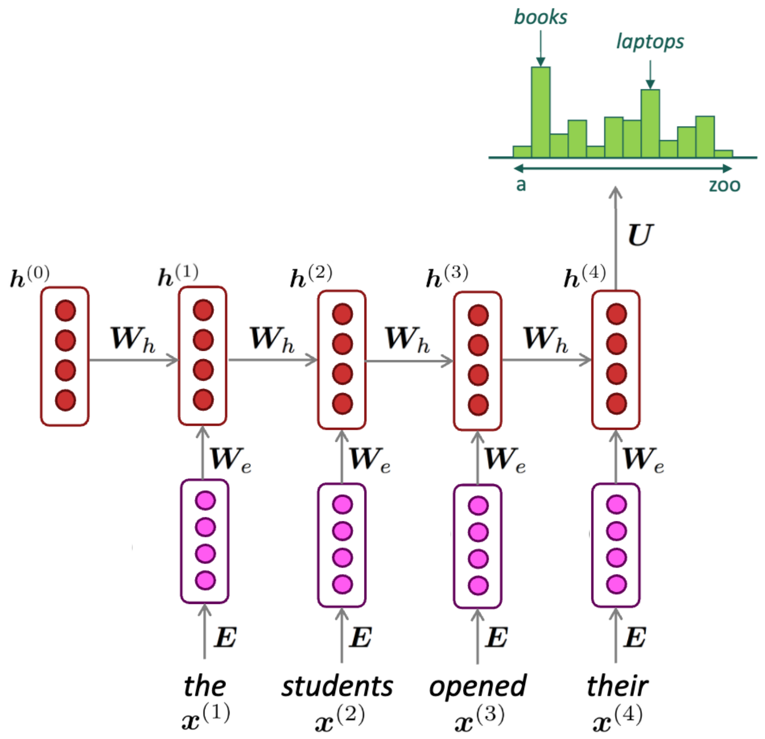
RNN Computation

- Input: $\boldsymbol{x} = [\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \cdots, \boldsymbol{x}^{(N)}]$
- Initialize $\boldsymbol{h}^{(0)}$
- For each time step (word) in the input:
  - Compute hidden states:

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e \boldsymbol{x}^{(t)}\right)$$

  - Compute output:

$$\boldsymbol{y}^{(t)} = \mathrm{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)}\right)$$



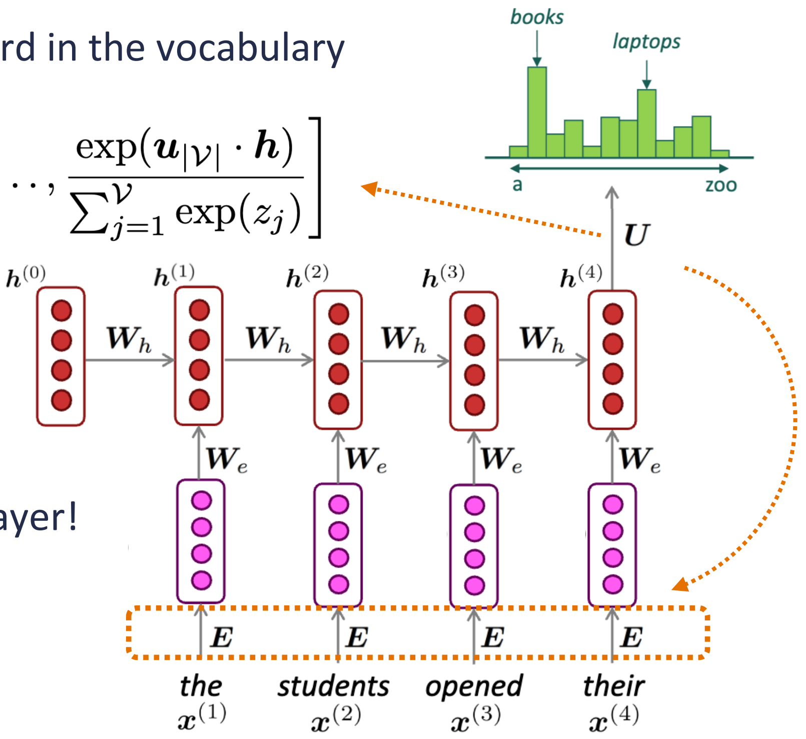
RNN Weight Tying

- Role of matrix $\boldsymbol{U}$: score the likelihood of each word in the vocabulary

$$\boldsymbol{y} = \text{softmax}(\boldsymbol{U}\boldsymbol{h}) = \left[\frac{\exp(\boldsymbol{u}_1 \cdot \boldsymbol{h})}{\sum_{j=1}^{|\mathcal{V}|} \exp(z_j)}, \ldots, \frac{\exp(\boldsymbol{u}_{|\mathcal{V}|} \cdot \boldsymbol{h})}{\sum_{j=1}^{\mathcal{V}} \exp(z_j)}\right]$$

$$\boldsymbol{U} \in \mathbb{R}^{|\mathcal{V}| \times d}$$

Same dimensionality of the word embedding matrix!



- Use the same input embeddings in the softmax layer!
- Weight tying benefits:
  - Improve learning efficiency & effectiveness
  - Reduce the number of parameters in the model

RNN for Language Modeling

- Recall that language modeling predict the next word given previous words

$$p(\boldsymbol{x}) = p\left(x^{(1)}\right) p\left(x^{(2)} | x^{(1)}\right) \cdots p\left(x^{(n)} | x^{(1)}, \ldots, x^{(n-1)}\right) = \prod_{t=1}^{n} p\left(x^{(t)} | x^{(1)}, \ldots, x^{(t-1)}\right)$$

- How to use RNNs to represent $p\left(x^{(t)} | x^{(1)}, \ldots, x^{(t-1)}\right)$?

Output probability at (t-1) step: $\boldsymbol{y}^{(t-1)} = \mathrm{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t-1)}\right) := f\left(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t-2)}, \boldsymbol{x}^{(t-1)}\right)$

$\boldsymbol{h}^{(t-1)}$ is a function of $\boldsymbol{h}^{(t-2)}$ and $\boldsymbol{x}^{(t-1)}$: $\boldsymbol{h}^{(t-1)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-2)} + \boldsymbol{W}_e \boldsymbol{x}^{(t-1)}\right) := g\left(\boldsymbol{h}^{(t-2)}, \boldsymbol{x}^{(t-1)}\right)$

$\boldsymbol{h}^{(t-2)}$ is a function of $\boldsymbol{h}^{(t-3)}$ and $\boldsymbol{x}^{(t-2)}$: $\boldsymbol{h}^{(t-2)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-3)} + \boldsymbol{W}_e \boldsymbol{x}^{(t-2)}\right) := g\left(\boldsymbol{h}^{(t-3)}, \boldsymbol{x}^{(t-2)}\right)$

$\vdots \qquad \vdots$

$\boldsymbol{h}^{(1)}$ is a function of $\boldsymbol{h}^{(0)}$ and $\boldsymbol{x}^{(1)}$: $\boldsymbol{h}^{(1)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(0)} + \boldsymbol{W}_e \boldsymbol{x}^{(1)}\right) := g\left(\boldsymbol{h}^{(0)}, \boldsymbol{x}^{(1)}\right)$

RNN Language Model Training

Train the output probability at each time step to predict the next word

$$\mathcal{L}_{\mathrm{LM}}(\boldsymbol{x}) = \frac{1}{n}\sum_{t=1}^{n}\mathcal{L}_{\mathrm{CE}}\left(\hat{\boldsymbol{y}}^{(t)}, \boldsymbol{y}^{(t)}\right) = \frac{1}{n}\sum_{t=1}^{n} -\log \hat{y}^{(t)}_{x^{(t)}} = \frac{1}{n}\sum_{t=1}^{n} -\log \frac{\exp\left(x^{(t)}\right)}{\sum_{w' \in \mathcal{V}} \exp(w')}$$

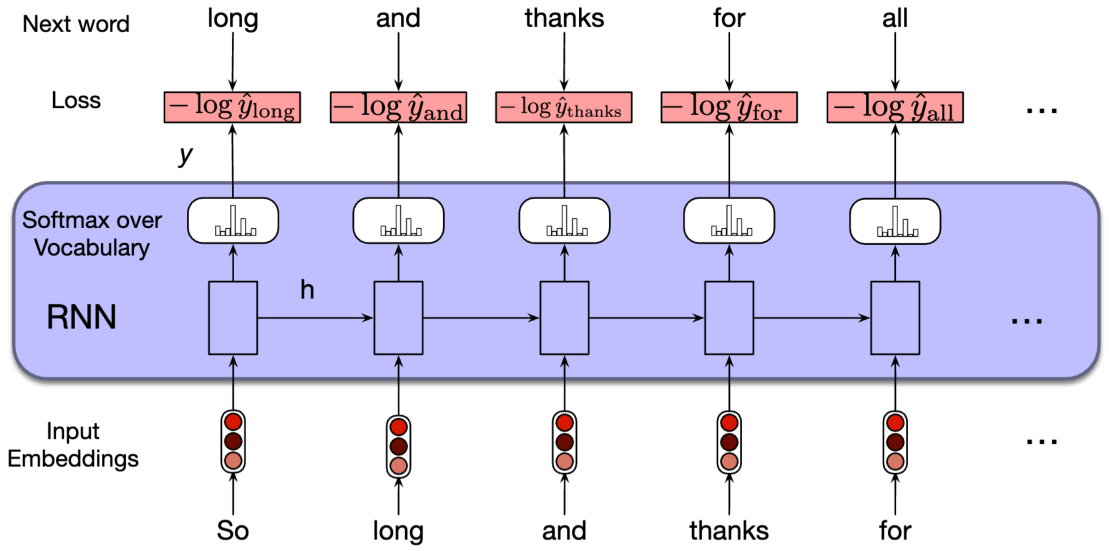

Next word
long
and
thanks
for
all
Loss
$-\log \hat{y}_{\mathrm{long}}$
$-\log \hat{y}_{\mathrm{and}}$
$-\log \hat{y}_{\mathrm{thanks}}$
$-\log \hat{y}_{\mathrm{for}}$
$-\log \hat{y}_{\mathrm{all}}$
...
y
Softmax over Vocabulary
RNN
h
...
Input Embeddings
...
So
long
and
thanks
for

Figure source: https://web.stanford.edu/~jurafsky/slp3/8.pdf

RNN for Text Generation

- Input [BOS] (beginning-of-sequence) token to the model
- Sample a word from the softmax distribution at the first time step
- Use the word embedding of that first word as the input at the next time step
- Repeat until the [EOS] (end-of-sequence) token is generated

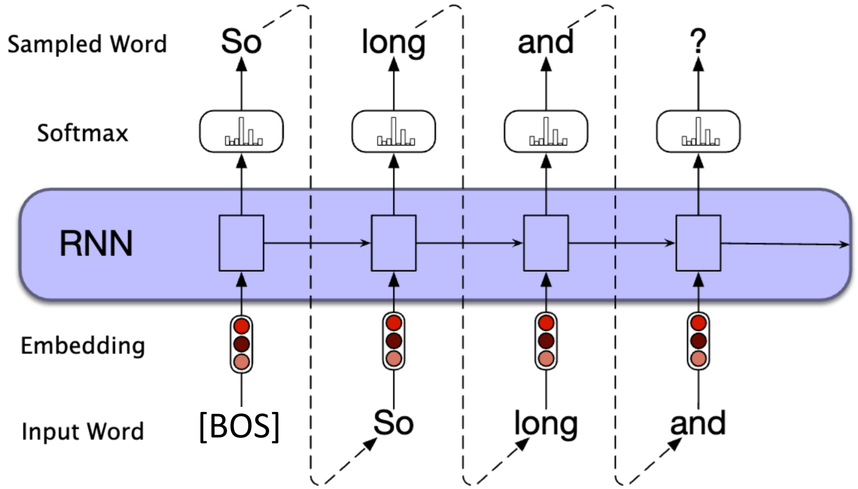


Figure source: https://web.stanford.edu/~jurafsky/slp3/8.pdf

UNIVERSITY of VIRGINIA

Join at
slido.com
#1931 885

Agenda

- Feedforward Network (FFN)
- Simple Neural Language Model
- Recurrent Neural Network (RNN)
- RNN Limitations
- Advanced RNNs

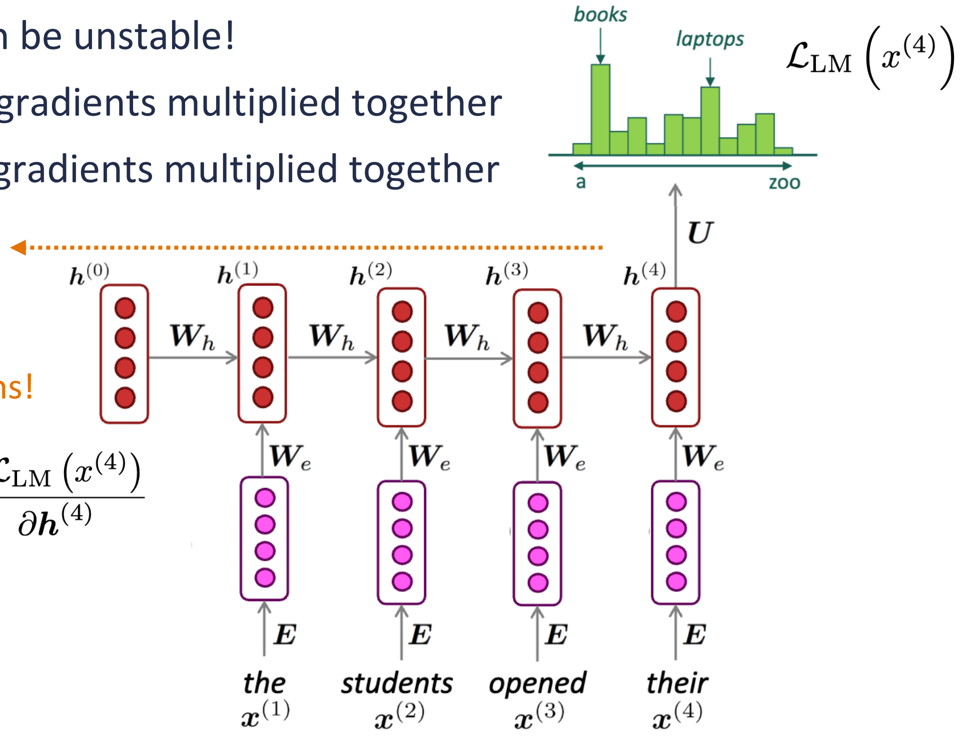
Vanishing & Exploding Gradient

- Gradient signal from far away can be unstable!
- Vanishing gradient = many small gradients multiplied together
- Exploding gradient = many large gradients multiplied together

Gradient backpropagation

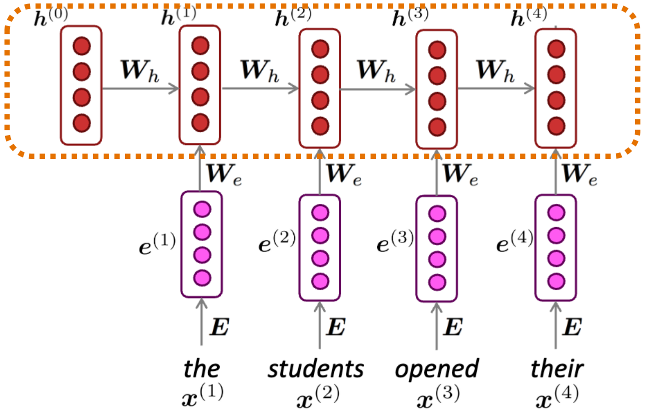
Lots of gradient multiplications!

$$\frac{\partial \mathcal{L}_{\text{LM}}\left(x^{(4)}\right)}{\partial \boldsymbol{h}^{(0)}} = \boxed{\frac{\partial \boldsymbol{h}^{(1)}}{\partial \boldsymbol{h}^{(0)}} \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}}} \frac{\partial \mathcal{L}_{\text{LM}}\left(x^{(4)}\right)}{\partial \boldsymbol{h}^{(4)}}$$



Difficulty in Capturing Long-Term Dependencies

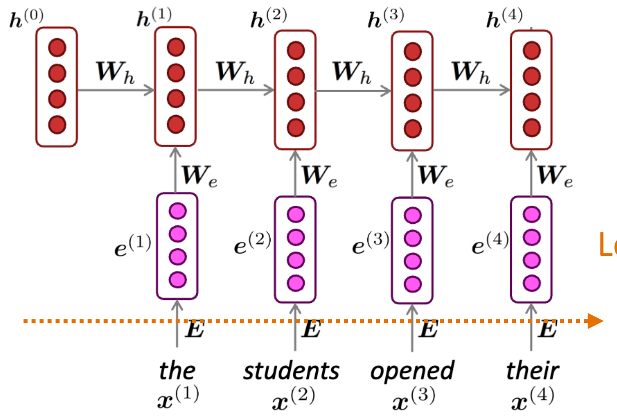
- RNNs are theoretically capable of remembering information over arbitrary lengths of input, but they struggle in practice with long-term dependencies
- RNNs use a fixed-size hidden state to encode an entire sequence of variable length; the hidden state is required to compress a lot of information
- RNNs might give more weight to the most recent inputs and may ignore or "forget" important information at the beginning of the sentence while processing the end



Fixed size hidden states!

Lack of Bidirectionality

- RNNs process the input sequence step by step from the beginning to the end (left to right for English)
- At each time step, the hidden state only has access to the information from the past without being able to leverage future contexts
- Example: “The bank is on the river” the word “bank” can be correctly disambiguated only if the model has access to the word “river” later in the sentence



Left to right processing

Thank You!

Yu Meng
University of Virginia
yumeng5@virginia.edu