Recurrent Neural Networks

Slido: https://app.sli.do/event/ov2SEjHCQJUPSd9EhLsUGA

Yu Meng

University of Virginia

yumeng5@virginia.edu

Sept 24, 2025

Overview of Course Contents

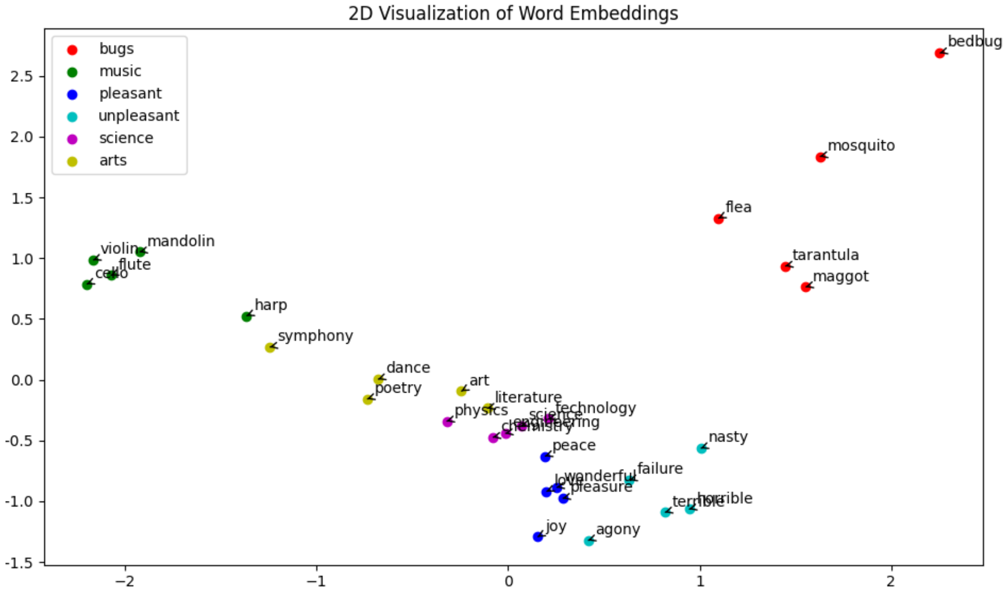
- Week 1: Logistics & Overview
- Week 2: N-gram Language Models
- Week 3: Word Senses, Semantics & Classic Word Representations
- Week 4: Word Embeddings
- Week 5: Sequence Modeling & Recurrent Neural Networks (RNNs)
- Week 6: Language Modeling with Transformers
- Week 9: Large Language Models (LLMs) & In-context Learning
- Week 10: Knowledge in LLMs and Retrieval-Augmented Generation (RAG)
- Week 11: LLM Alignment
- Week 12: Reinforcement Learning for LLM Post-Training
- Week 13: LLM Agents + Course Summary
- Week 15 (after Thanksgiving): Project Presentations

Reminder

- Project proposal is due today (no late days allowed)!

(Recap) Word Similarity

- Measure word similarity with cosine similarity between embeddings $\cos(\boldsymbol{v}_{w_1}, \boldsymbol{v}_{w_2})$
- Higher cosine similarity = more semantically close



(Recap) Word Similarity Evaluation

- An **intrinsic** word embedding evaluation
- Measure how well word vector similarity correlates with human judgments
- Example dataset: WordSim353 (353 word pairs with their similarity scores assessed by humans)

Word 1	Word 2	Human (mean)
tiger	cat	7.35
book	paper	7.46
computer	internet	7.58
plane	car	5.77
professor	doctor	6.62
stock	phone	1.62
stock	CD	1.31
stock	jaguar	0.92

Figure source: https://web.stanford.edu/class/cs224n/slides/cs224n-spr2024-lecture02-wordvecs2.pdf

(Recap) Correlation Metric

Spearman rank correlation: measure the correlation between two rank variables

Word 1	Word 2	Human (mean)
tiger	cat	7.35
book	paper	7.46
computer	internet	7.58
plane	car	5.77
professor	doctor	6.62
stock	phone	1.62
stock	CD	1.31
stock	jaguar	0.92

Rank by human

$$\begin{bmatrix} 6 \\ 7 \\ 8 \\ 4 \\ 5 \\ 3 \\ 2 \\ 1 \end{bmatrix}$$

$$\mathrm{Cov}(X, Y) = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])]$$

Covariance

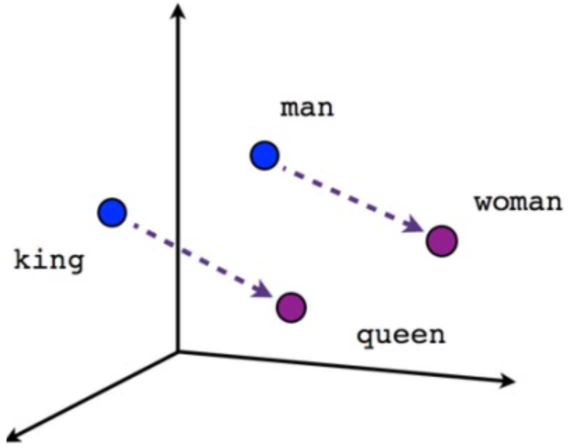
$$r = \frac{\mathrm{Cov}[R[X], R[Y]]}{\sigma_{R[X]}\sigma_{R[Y]}}$$

Standard deviations

(Recap) Word Analogy

- Word embeddings reflect intuitive semantic and syntactic analogy
- Example: man : woman :: king : ? $\boldsymbol{v}_{\text{queen}} \approx \boldsymbol{v}_{\text{woman}} - \boldsymbol{v}_{\text{man}} + \boldsymbol{v}_{\text{king}}$
- General case: find the word such that a : b :: c : ?
- Find the word that maximizes the cosine similarity

$$
\begin{aligned}
w &= \arg\max_{w' \in \mathcal{V}} \cos(\boldsymbol{v}_b - \boldsymbol{v}_a + \boldsymbol{v}_c, \boldsymbol{v}_{w'}) \\
&= \arg\max_{w' \in \mathcal{V}} \frac{(\boldsymbol{v}_b - \boldsymbol{v}_a + \boldsymbol{v}_c) \cdot \boldsymbol{v}_{w'}}{|\boldsymbol{v}_b - \boldsymbol{v}_a + \boldsymbol{v}_c||\boldsymbol{v}_{w'}|}
\end{aligned}
$$



(Recap) Word Analogy Evaluation

- Word analogy is another **intrinsic** word embedding evaluation
- Encompass various types of word relationships
- Usually use accuracy as the metric

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

Figure source: https://arxiv.org/pdf/1301.3781

(Recap) Extrinsic Evaluation of Word Embeddings

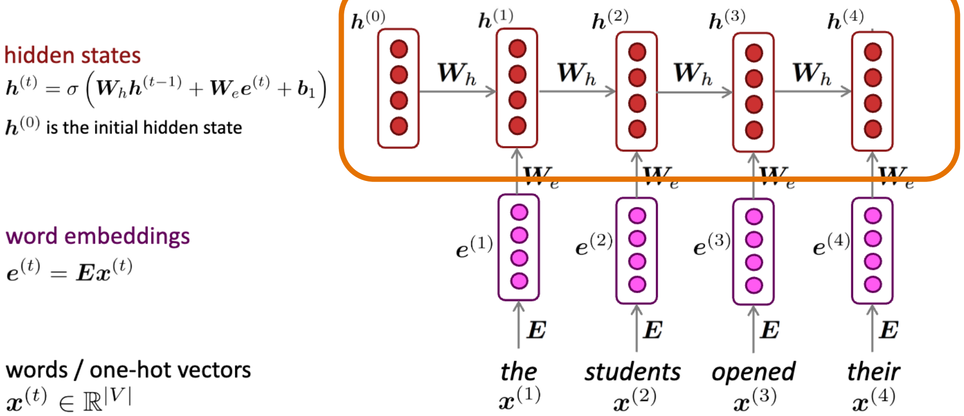
- Word embeddings can be used as input features to task-specific NLP models
- Example 1: Text classification (topic/sentiment classification)
  - Sentence/document embeddings are obtained by applying sequence modeling architectures on top of word embeddings
  - Classification accuracy is used as the extrinsic metric
- Example 2: Named entity recognition (NER)
  - Find and classify entity names (e.g., person, organization, location) in text
  - Concatenated word embeddings can be used to represent spans of words (entities)
  - Precision/recall/F1 are used as the extrinsic metrics
- Word embedding demo

(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

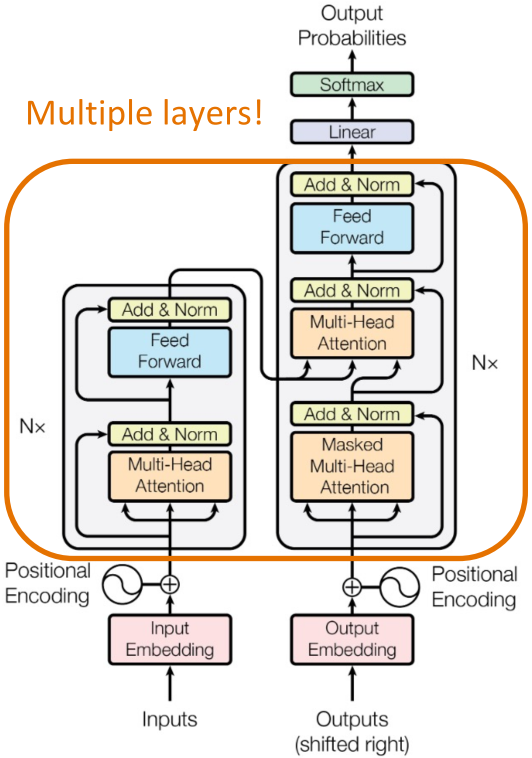
(Recap) Sequence Modeling Architectures

Multiple layers!



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

Multiple layers!



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

(Recap) Neural Networks (Overview)

Biological neural network

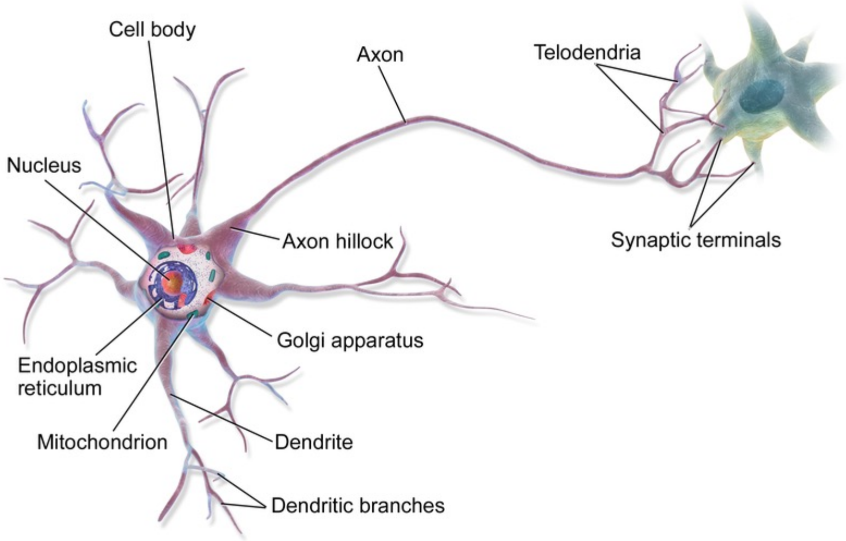


Figure source:
https://commons.wikimedia.org/w/index.php?curid=28761830

Artificial neural network

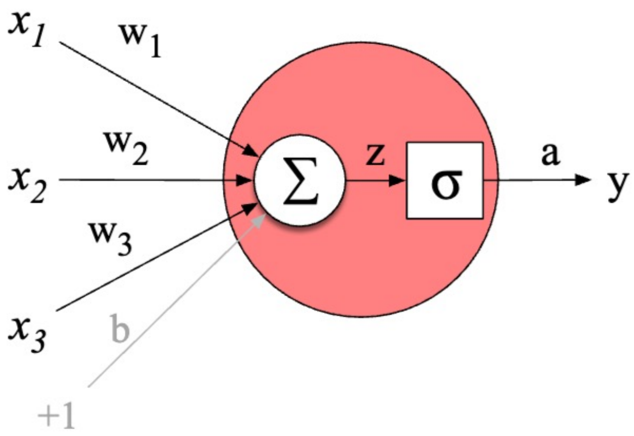
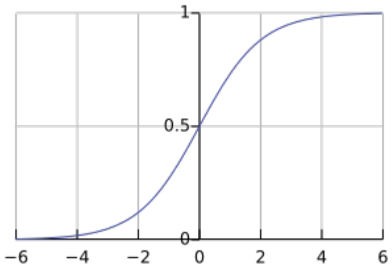


Figure source:
https://web.stanford.edu/~jurafsky/slp3/7.pdf

Neural Network: Basic 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: $a = \sigma(z)$



$y = \sigma(x)$

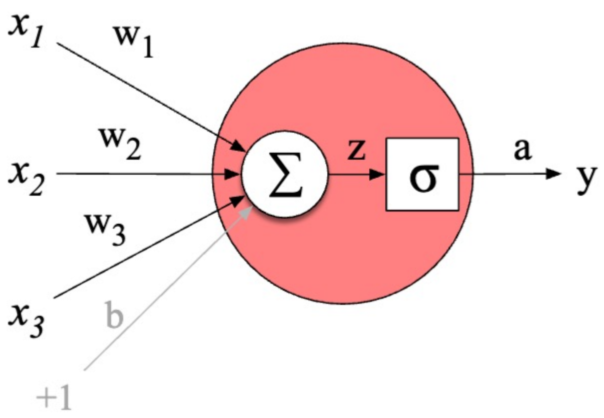


Figure source: https://web.stanford.edu/~jurafsky/slp3/6.pdf

Basic Unit (Perceptron): Example

- Input: $\boldsymbol{x} = [0.5, 0.6, 0.1]$
- Model parameters (weights & bias): $\boldsymbol{w} = [0.2, 0.3, 0.9]$ & $b = 0.5$
- Linear computation: $z = \boldsymbol{w} \cdot \boldsymbol{x} + b = 0.87$
- Nonlinear activation: $a = \sigma(z) = \dfrac{1}{1 + \exp(-0.87)} \approx 0.70$

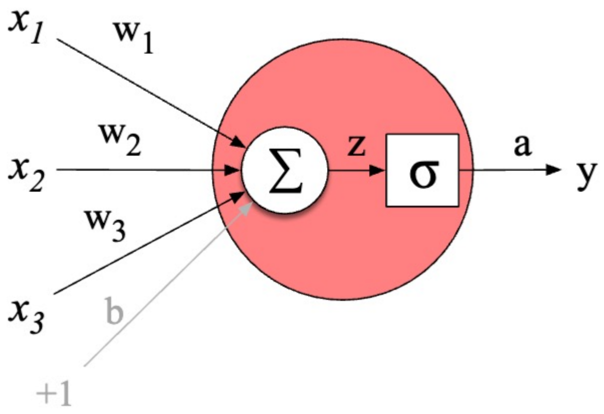
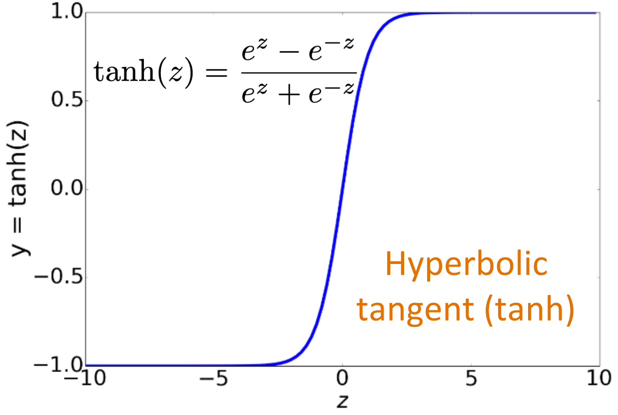


Figure source: https://web.stanford.edu/~jurafsky/slp3/6.pdf

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



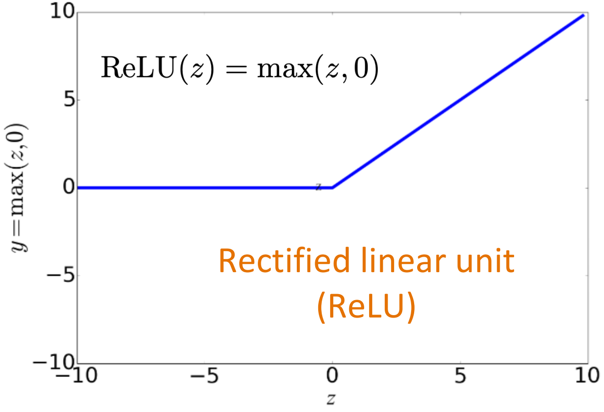


Figure source: https://web.stanford.edu/~jurafsky/slp3/6.pdf

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 a weight vector and a bias parameter
  - Weight matrix: combining the weight vector for each unit
  - Bias vector: combining the bias for each unit

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

Convert to probability distribution

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

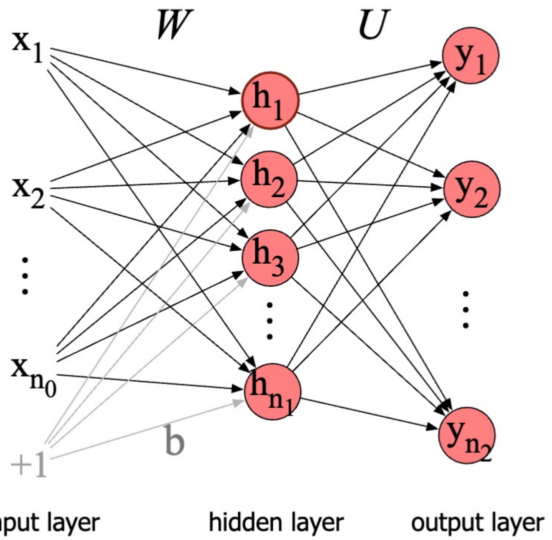


Figure source: https://web.stanford.edu/~jurafsky/slp3/13.pdf

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

Model output probability

Ground-truth probability

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

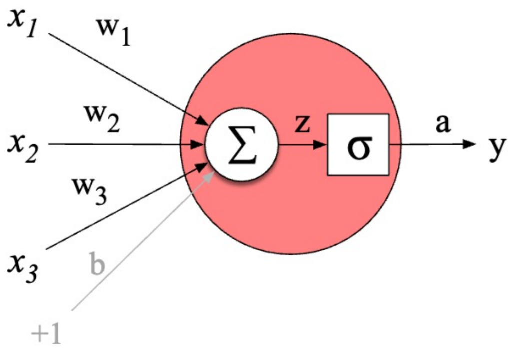
$$\mathcal{L}_{\text{CE}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\log \hat{y}_c = -\log \frac{\exp(z_c)}{\sum_{j=1}^{K} \exp(z_j)}$$

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



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

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 $\nabla_{\boldsymbol{w}} \mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}} \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{w}}$

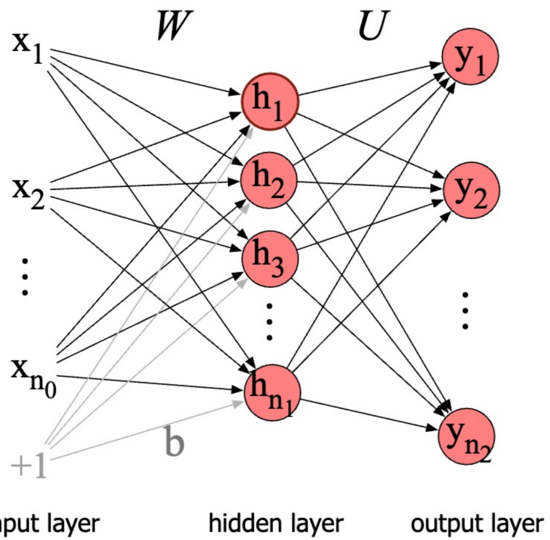
Gradient computation taken care of by deep learning libraries
(e.g., PyTorch)

Agenda

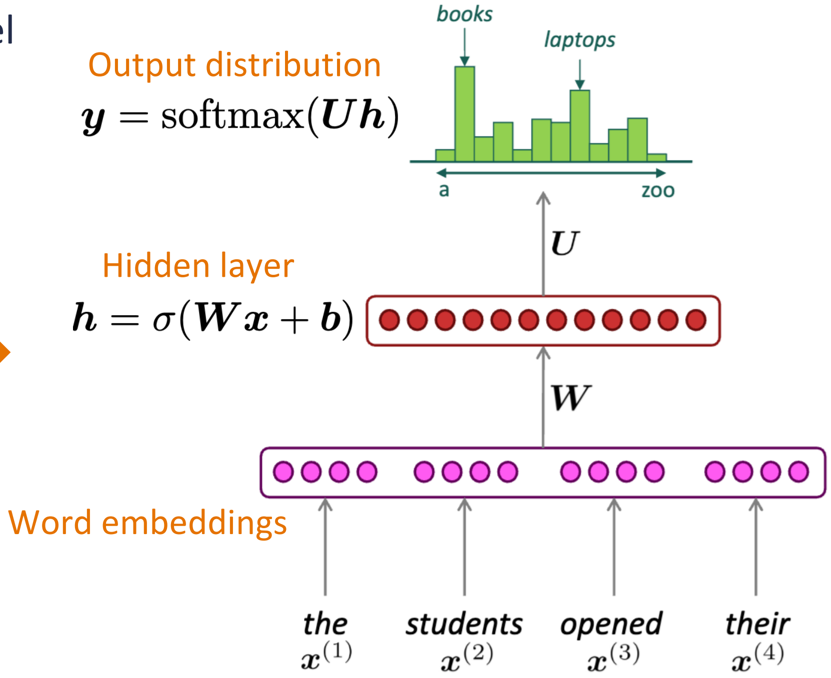
- Feedforward Network (FFN)
- Simple Neural Language Model
- Recurrent Neural Network (RNN)
- RNN Limitations
- Advanced RNNs

Simple Neural Language Model

Instantiate FFN as a neural language model



2-layer FFN



2-layer neural language model

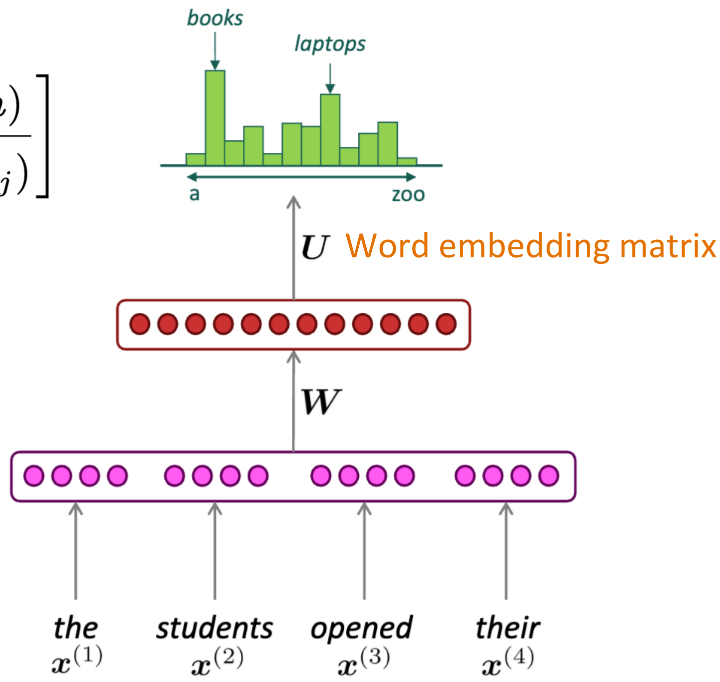
Figure source: https://web.stanford.edu/class/cs224n/slides/cs224n-spr2024-lecture05-rnnlm.pdf

Benefits of Neural Language Models

Output distribution

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

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

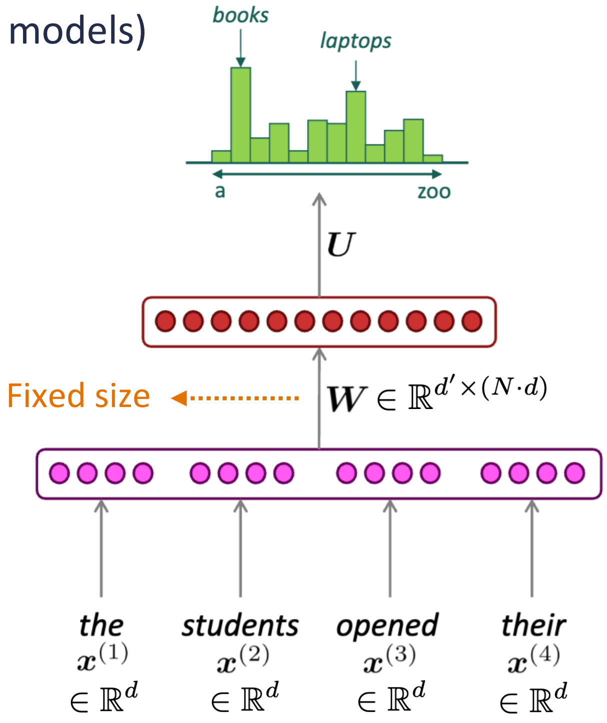


Limitations of (Simple) Neural Language Models

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

Concatenated word embeddings

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



Agenda

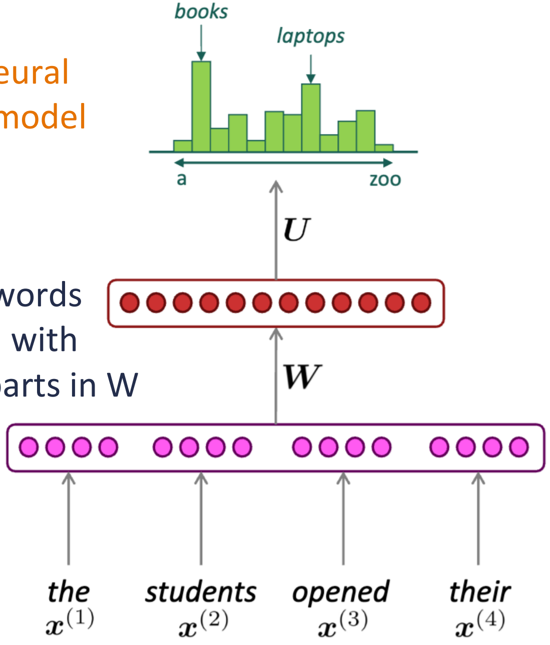
- Feedforward Network (FFN)
- Simple Neural Language Model
- Recurrent Neural Network (RNN)
- RNN Limitations
- Advanced RNNs

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

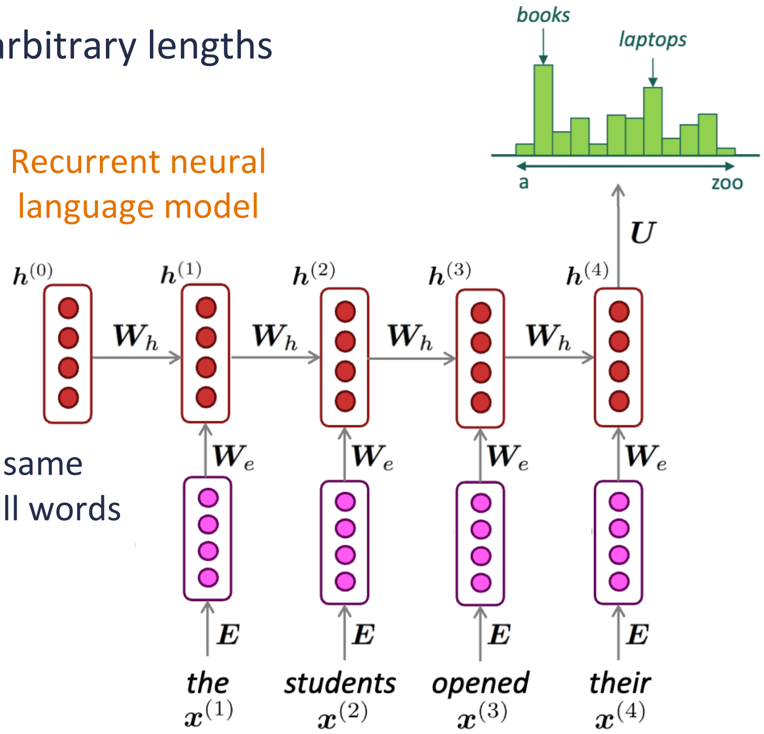


Figure source: https://web.stanford.edu/class/cs224n/slides/cs224n-spr2024-lecture05-rnnlm.pdf

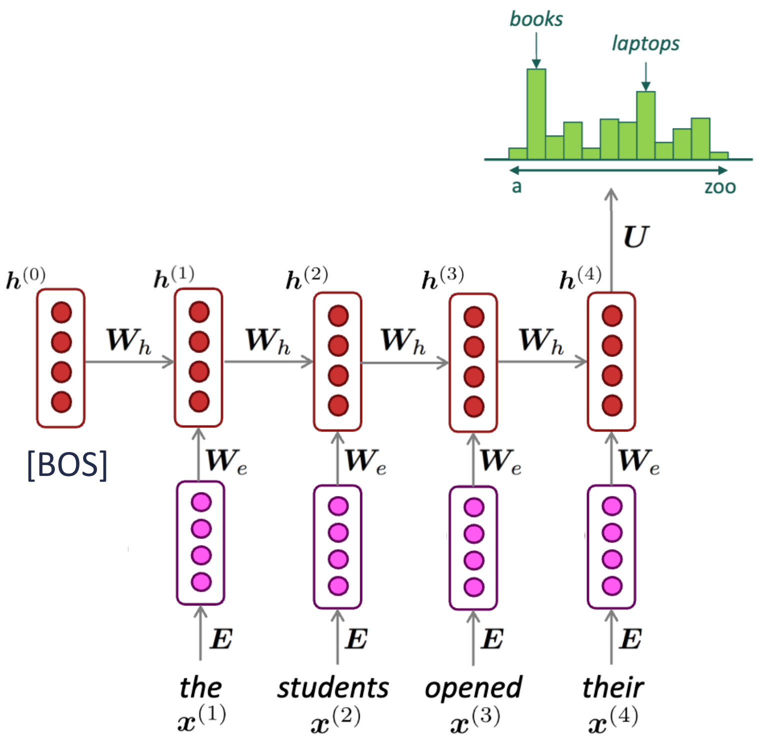
RNN Computation

- 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

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

Hidden states at the previous word (time step)

Word embedding of the current word (time step)



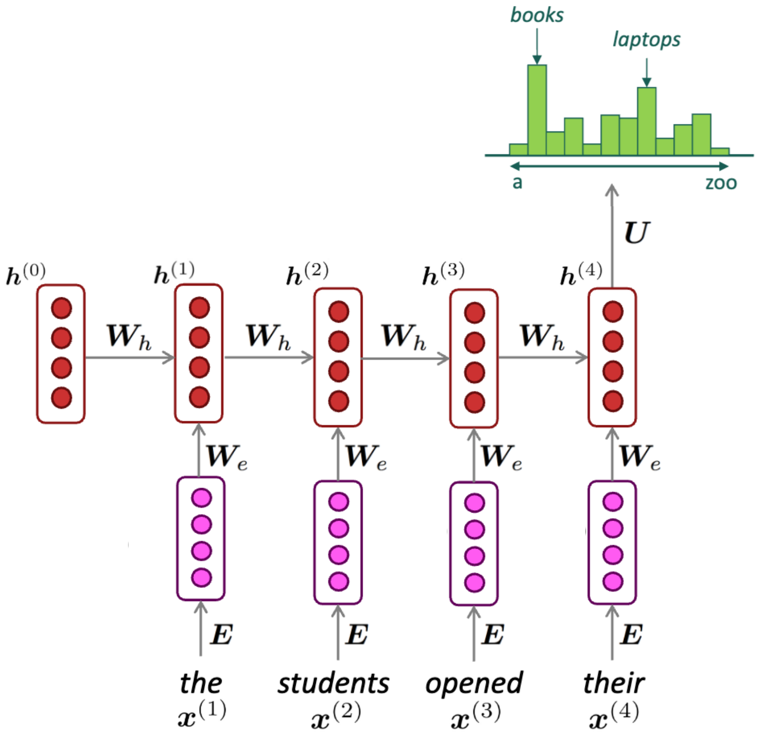
RNN Computation

- 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:

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



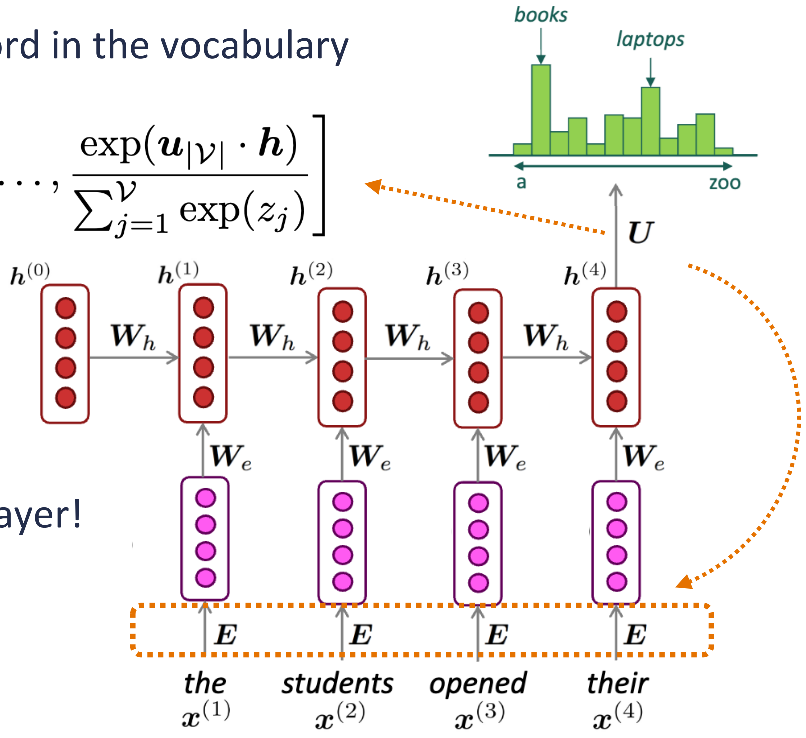
RNN Weight Tying

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

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

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

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 predicts 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

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

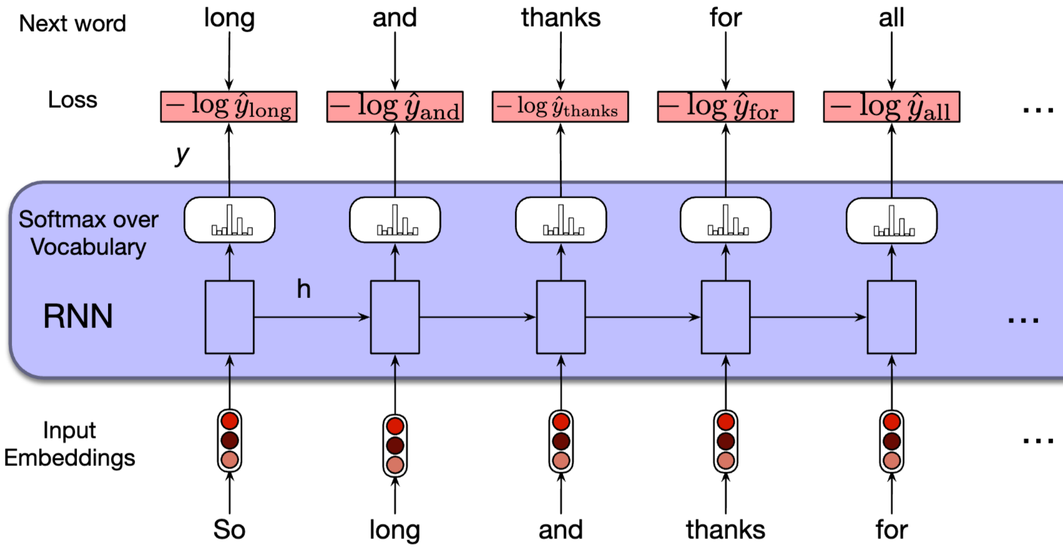


Figure source: https://web.stanford.edu/~jurafsky/slp3/13.pdf

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

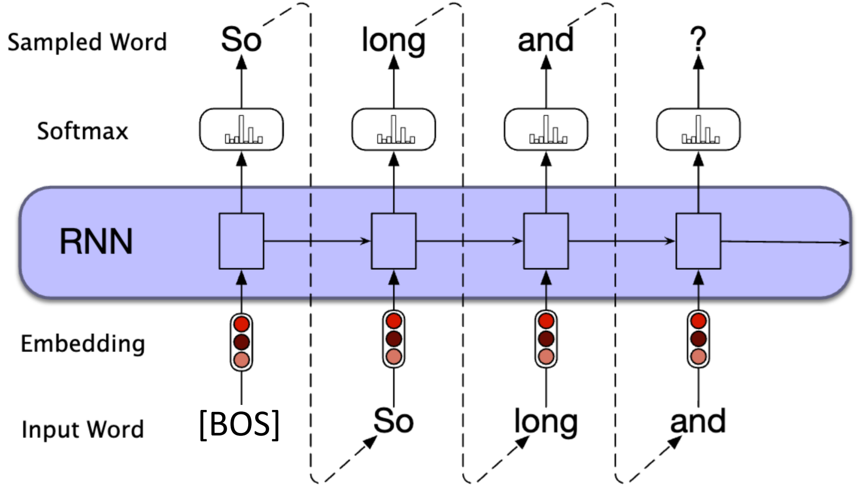


Figure source: https://web.stanford.edu/~jurafsky/slp3/13.pdf

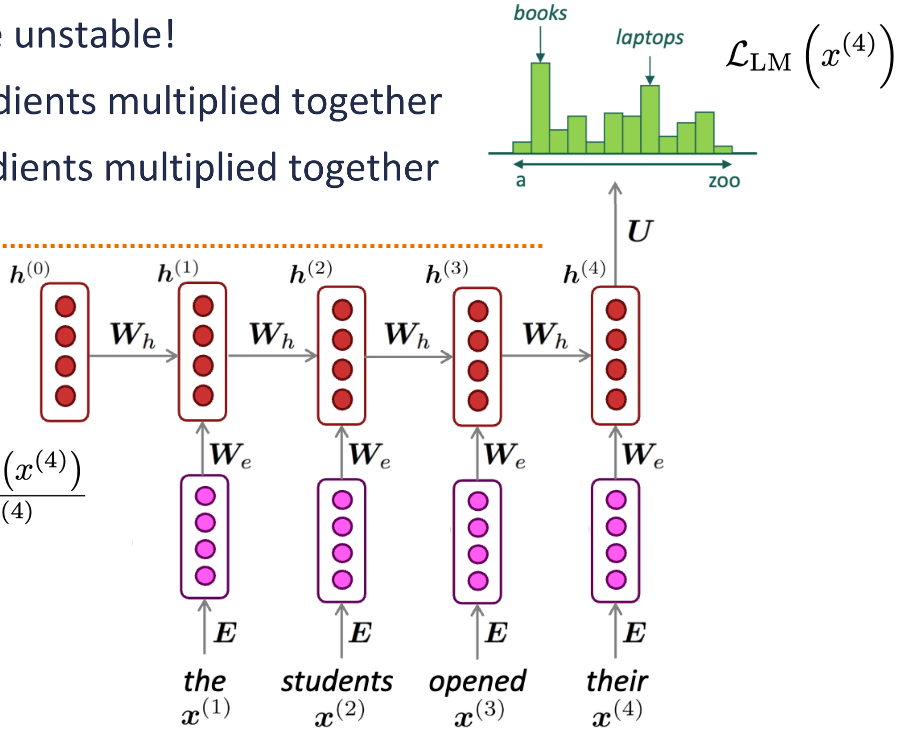
Agenda

- 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

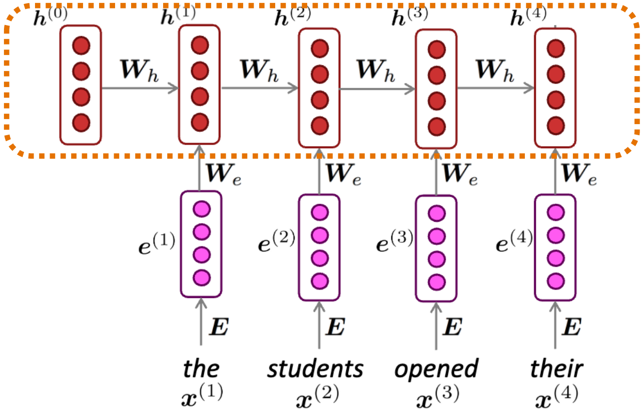


Lots of gradient multiplications!

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

Difficulty in Capturing Long-Term Dependencies

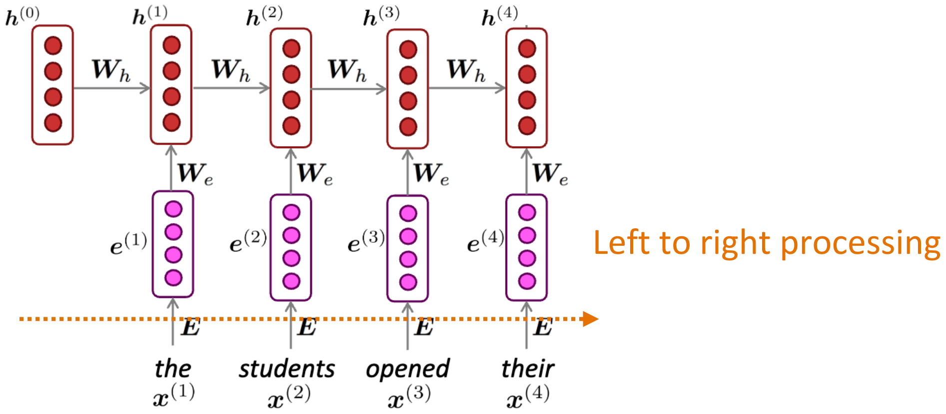
- RNNs are theoretically capable of remembering information over arbitrary lengths of input, but they struggle in practice with long-term dependencies
- RNNs use a fixed-size hidden state to encode an entire sequence of variable length; the hidden state is required to compress a lot of information
- 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!

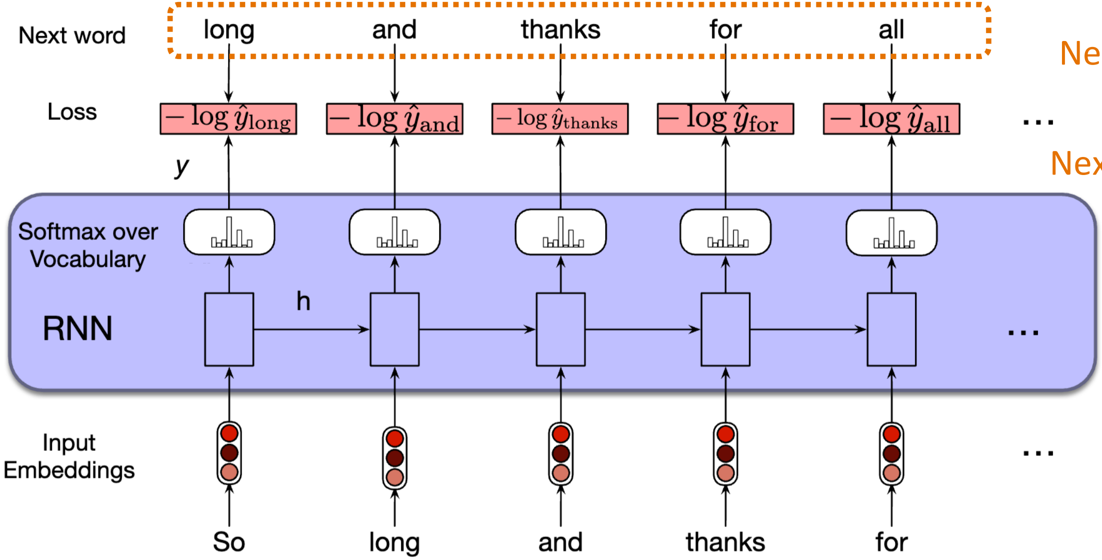
Lack of Bidirectionality

- 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



Exposure Bias

- **Teacher forcing/exposure bias**: during RNN training, the model always receives the **correct** next word from the training data as input for the next step
- When the model has to predict sequences on its own, it may perform poorly if it hasn't learned how to correct its own mistakes



During training:
Next word = actual next word

During generation:
Next word = model's prediction

Agenda

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

Long Short-Term Memory (LSTM)

- Challenge in RNNs: information encoded in hidden states tends to be local; distant information gets lost
- LSTM design intuition:
  - Remove information no longer needed from the context
  - Add information likely to be needed for future time steps
- Inputs at each time step:
  - Word embedding of the current word
  - Hidden state from the previous time step
  - **Memory/cell state**
- Three gates:
  - Forget gate
  - Add/input gate
  - Output gate

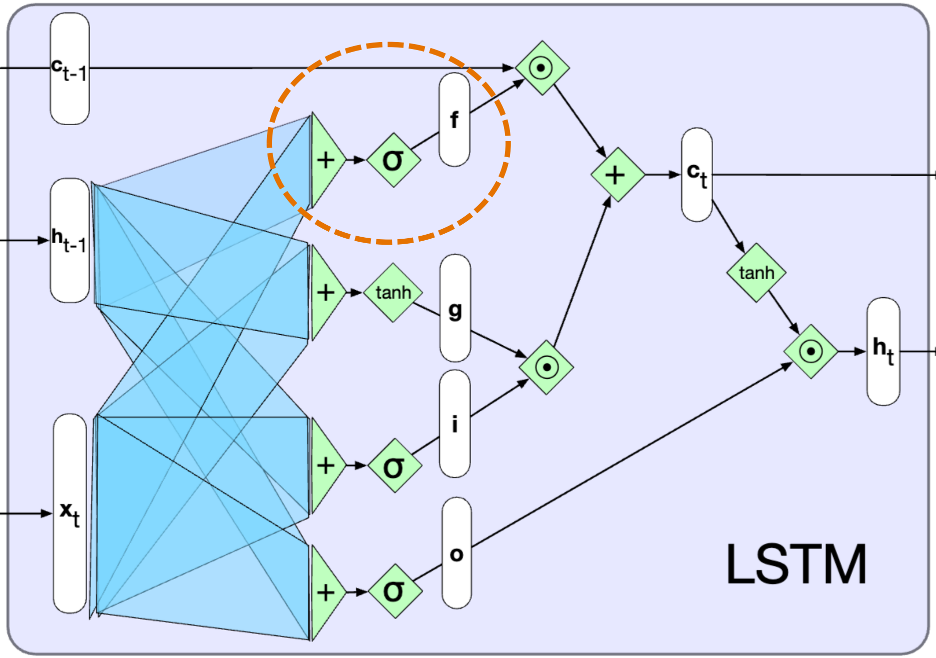
LSTM Computation (Forget Gate)

$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{U}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_f \boldsymbol{x}^{(t)}\right)$$

Cell state $\boldsymbol{c}^{(t-1)}$

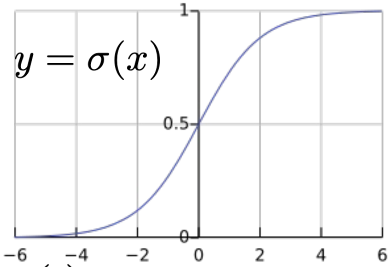
Hidden state $\boldsymbol{h}^{(t-1)}$

Word embedding $\boldsymbol{x}^{(t)}$



$\boldsymbol{c}^{(t)}$

$\boldsymbol{h}^{(t)}$



Decides what information to discard from the cell state

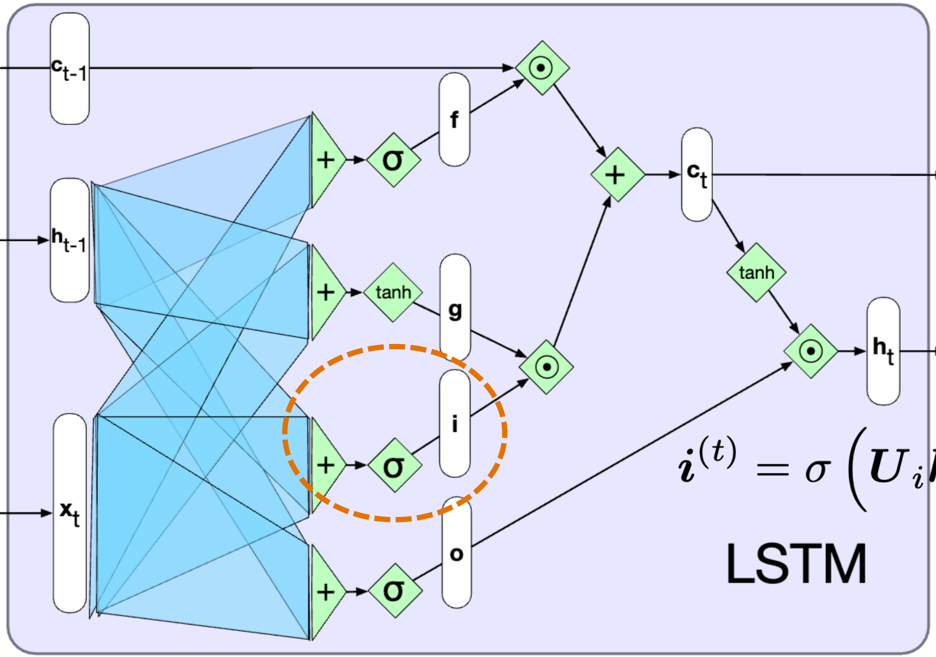
Figure source: https://web.stanford.edu/~jurafsky/slp3/13.pdf

LSTM Computation (Add/Input Gate)

Cell state $\boldsymbol{c}^{(t-1)}$

Hidden state $\boldsymbol{h}^{(t-1)}$

Word embedding $\boldsymbol{x}^{(t)}$



$\boldsymbol{c}^{(t)}$

$\boldsymbol{h}^{(t)}$

Decides what new information to store to the cell state

$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{U}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_i \boldsymbol{x}^{(t)}\right)$$

Figure source: https://web.stanford.edu/~jurafsky/slp3/13.pdf

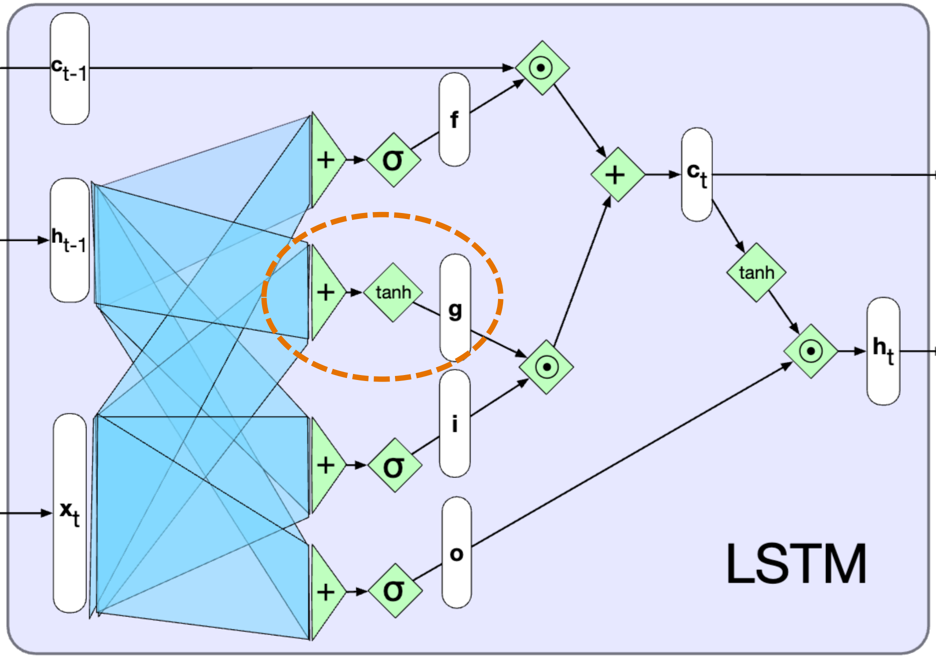
LSTM Computation (Candidate Cell State)

$$\boldsymbol{g}^{(t)} = \tanh\left(\boldsymbol{U}_g \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_g \boldsymbol{x}^{(t)}\right)$$

Cell state $\boldsymbol{c}^{(t-1)}$

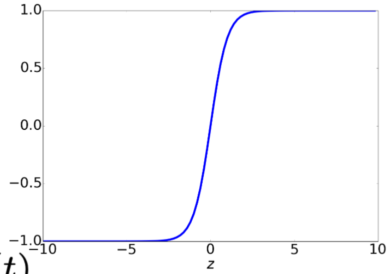
Hidden state $\boldsymbol{h}^{(t-1)}$

Word embedding $\boldsymbol{x}^{(t)}$



$\boldsymbol{c}^{(t)}$

$\boldsymbol{h}^{(t)}$



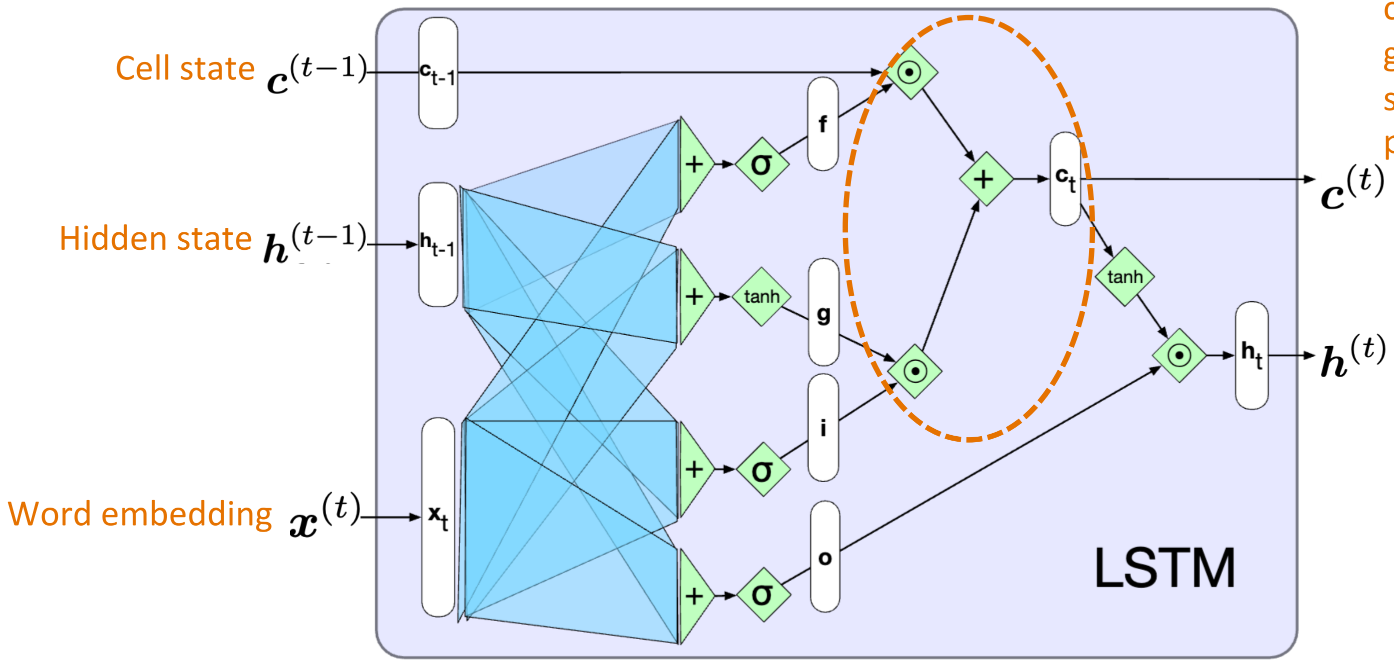
$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Compute information needed from the previous hidden state and current inputs

Figure source: https://web.stanford.edu/~jurafsky/slp3/13.pdf

LSTM Computation (Cell State Update)

$$\boldsymbol{c}^{(t)} = \boldsymbol{i}^{(t)} \odot \boldsymbol{g}^{(t)} + \boldsymbol{f}^{(t)} \odot \boldsymbol{c}^{(t-1)}$$



Cell state updated by combining the input gate, candidate cell state, forget gate & previous cell state

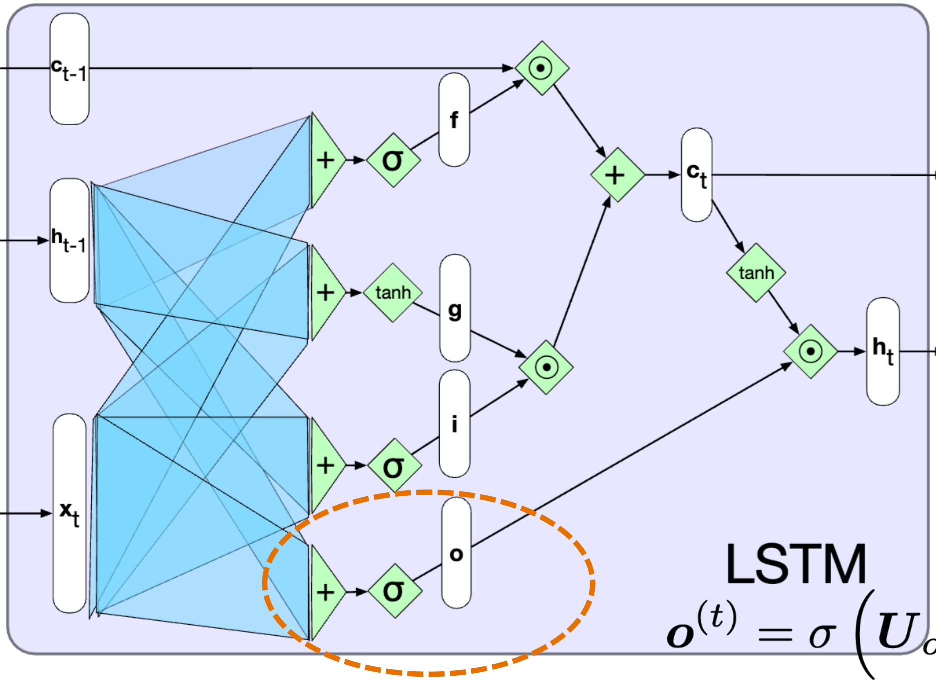
Figure source: https://web.stanford.edu/~jurafsky/slp3/13.pdf

LSTM Computation (Output Gate)

Cell state $\boldsymbol{c}^{(t-1)}$

Hidden state $\boldsymbol{h}^{(t-1)}$

Word embedding $\boldsymbol{x}^{(t)}$



$\boldsymbol{c}^{(t)}$

$\boldsymbol{h}^{(t)}$

Decides what parts of the cell state will be output

$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{U}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_o \boldsymbol{x}^{(t)}\right)$$

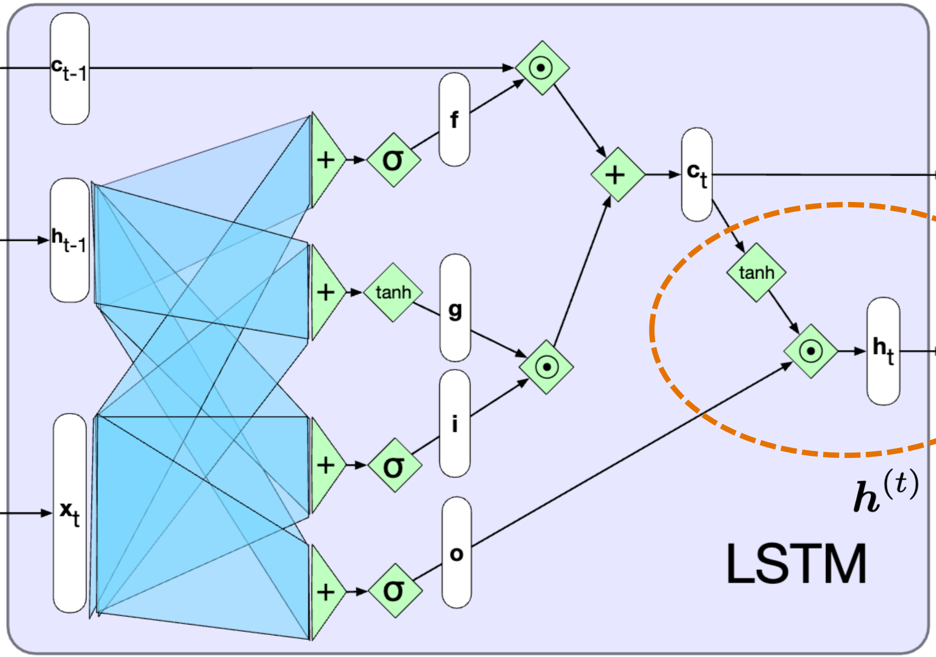
Figure source: https://web.stanford.edu/~jurafsky/slp3/13.pdf

LSTM Computation (Hidden State Update)

Cell state $\boldsymbol{c}^{(t-1)}$

Hidden state $\boldsymbol{h}^{(t-1)}$

Word embedding $\boldsymbol{x}^{(t)}$



$\boldsymbol{c}^{(t)}$

$\boldsymbol{h}^{(t)}$

$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \odot \tanh\left(\boldsymbol{c}^{(t)}\right)$$

Hidden state updated using the output gate & the updated cell state

Figure source: https://web.stanford.edu/~jurafsky/slp3/13.pdf

Bidirectional RNNs

- Separate models are trained in the forward and backward directions
- Hidden states from both RNNs are concatenated as the final representations

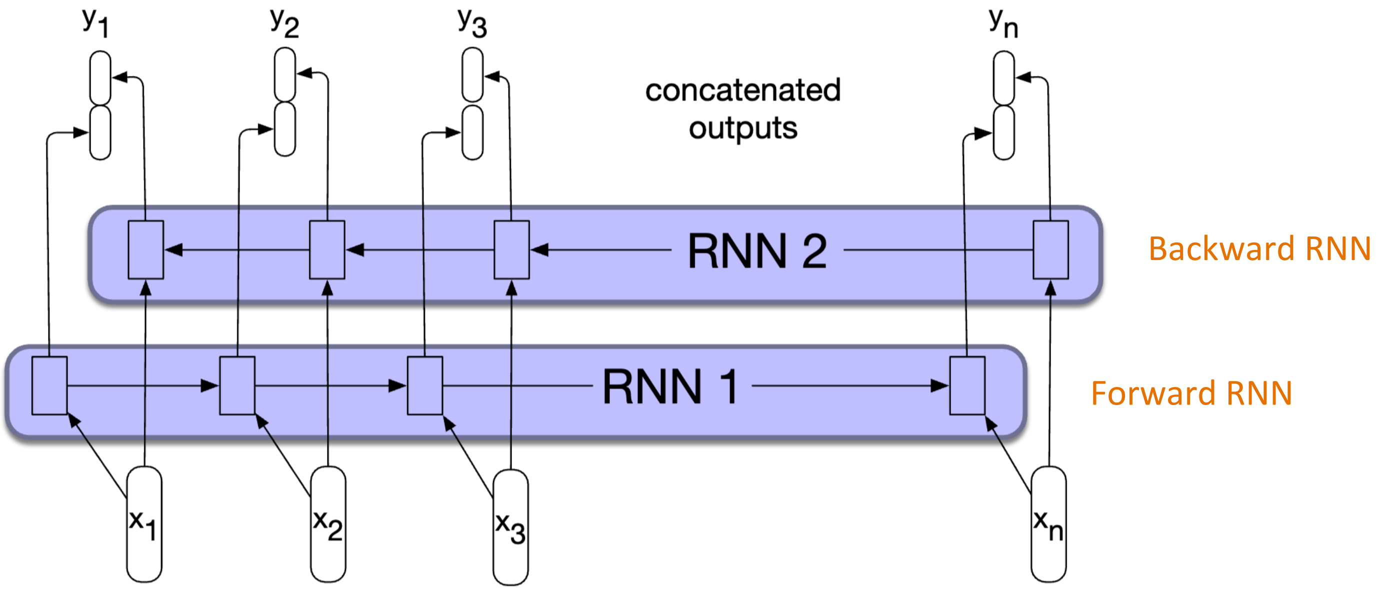


Figure source: https://web.stanford.edu/~jurafsky/slp3/13.pdf

UNIVERSITY of VIRGINIA

Deep RNNs

- We can stack multiple RNN layers to build deep RNNs
- The output of a lower level serves as the input to higher levels
- The output of the last layer is used as the final output

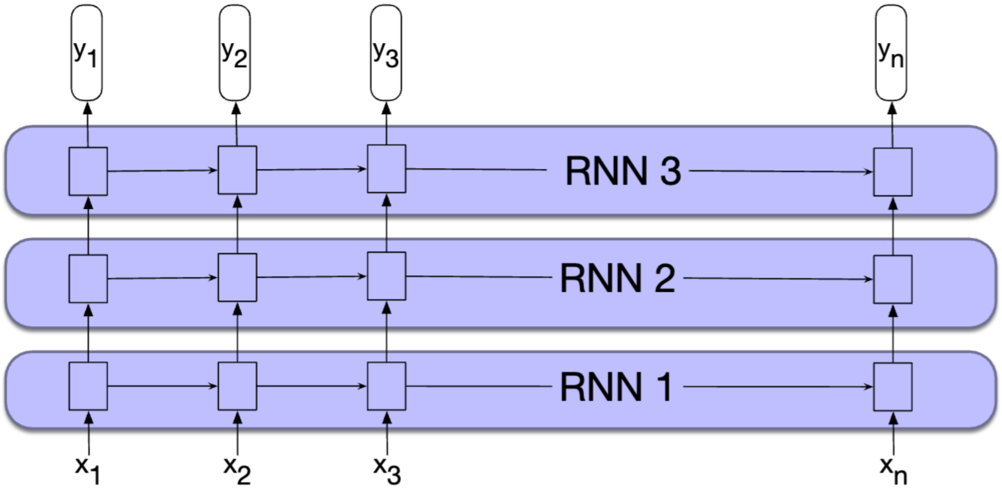


Figure source: https://web.stanford.edu/~jurafsky/slp3/13.pdf

Summary: Sequence Modeling

- Sequence modeling goals:
  - Learn context-dependent representations
  - Capture long-range dependencies
  - Handle complex relationships among large text units
- Use deep learning architectures to understand, process, and generate text sequences
- Why DNNs?
  - The multi-layer structure in DNNs mirrors the hierarchical structures in language
  - DNNs learn multiple levels of semantics across layers: low-level patterns (e.g., relations between words) in lower layers & high-level patterns (e.g., sentence meanings) in higher layers

Summary: Neural Language Models

- Address the sparsity issue in N-gram language models by computing the output distribution based on distributed representations (with semantic information)
- Simple neural language models based on feedforward networks suffer from the fixed context window issue
  - Can only model a fixed number of words (similar to N-gram assumption)
  - Increasing the context window requires adding more model parameters

Summary: Recurrent Neural Networks

- General idea: Use the same set of model weights to process all input words
- RNNs as language models
  - Theoretically able to process infinitely long sequences
  - Practically can only keep track of recent contexts
- Training issues: vanishing & exploding gradients
- LSTM is a prominent RNN variant to keep track of both long-term and short-term memories via multiple gates

Thank You!

Yu Meng
University of Virginia
yumeng5@virginia.edu