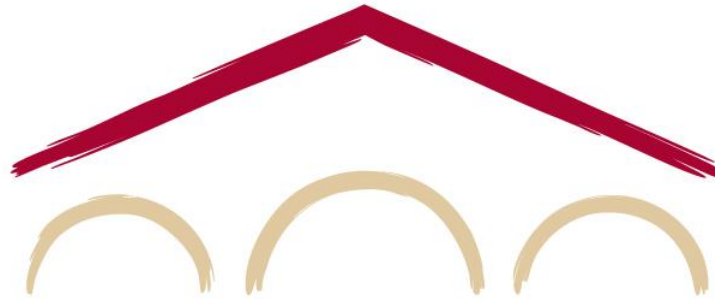


# Natural Language Processing with Deep Learning

## CS224N/Ling284



Tatsunori Hashimoto

Lecture 5: Language Models and Recurrent Neural Networks

# Lecture Plan

## Language modeling + RNNs

- 1. A new NLP task: **Language Modeling** (20 mins)

motivates

This is the most important concept in the class! Leads to most of modern NLP

- 2. Language models with neural nets: **Recurrent Neural Networks (RNNs)** (25 mins)
- 4. Problems with RNNs (15 mins)
- 5. Recap on RNNs/LMs (10 mins)

Important and used in Ass3, but not the only way to build LMs

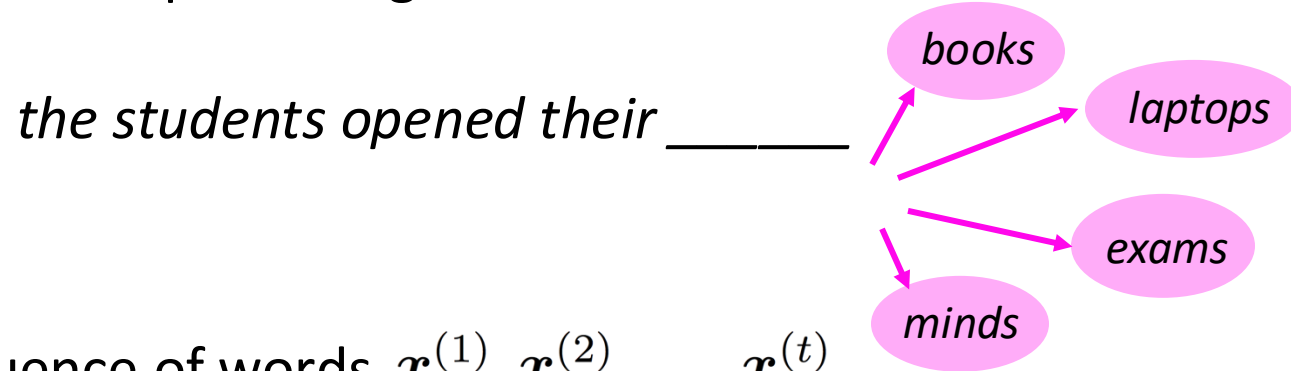
## Reminders:

Assignment 2 – Due Jan 23, Thursday

In Assignment 3, out Thursday – Neural MT with RNNs!

# 1. Language Modeling

- **Language Modeling** is the task of predicting what word comes next



- More formally: given a sequence of words  $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(t)}$ , compute the probability distribution of the next word  $\mathbf{x}^{(t+1)}$ :

$$P(\mathbf{x}^{(t+1)} \mid \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(1)})$$

where  $\mathbf{x}^{(t+1)}$  can be any word in the vocabulary  $V = \{\mathbf{w}_1, \dots, \mathbf{w}_{|V|}\}$

- A system that does this is called a **Language Model**

# Language Modeling

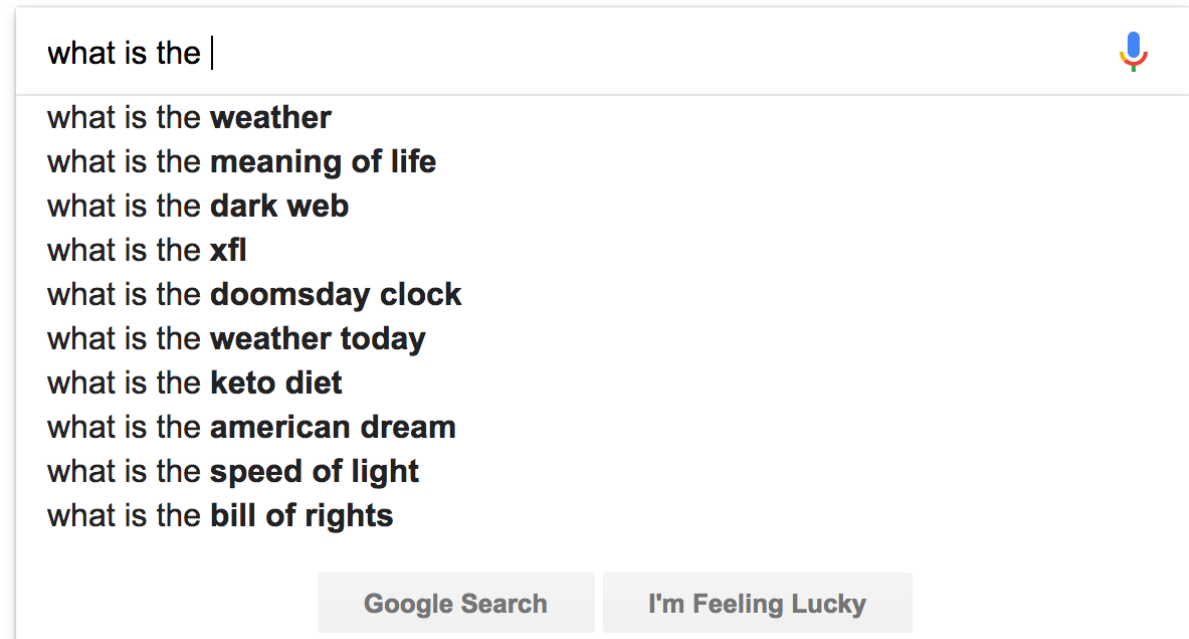
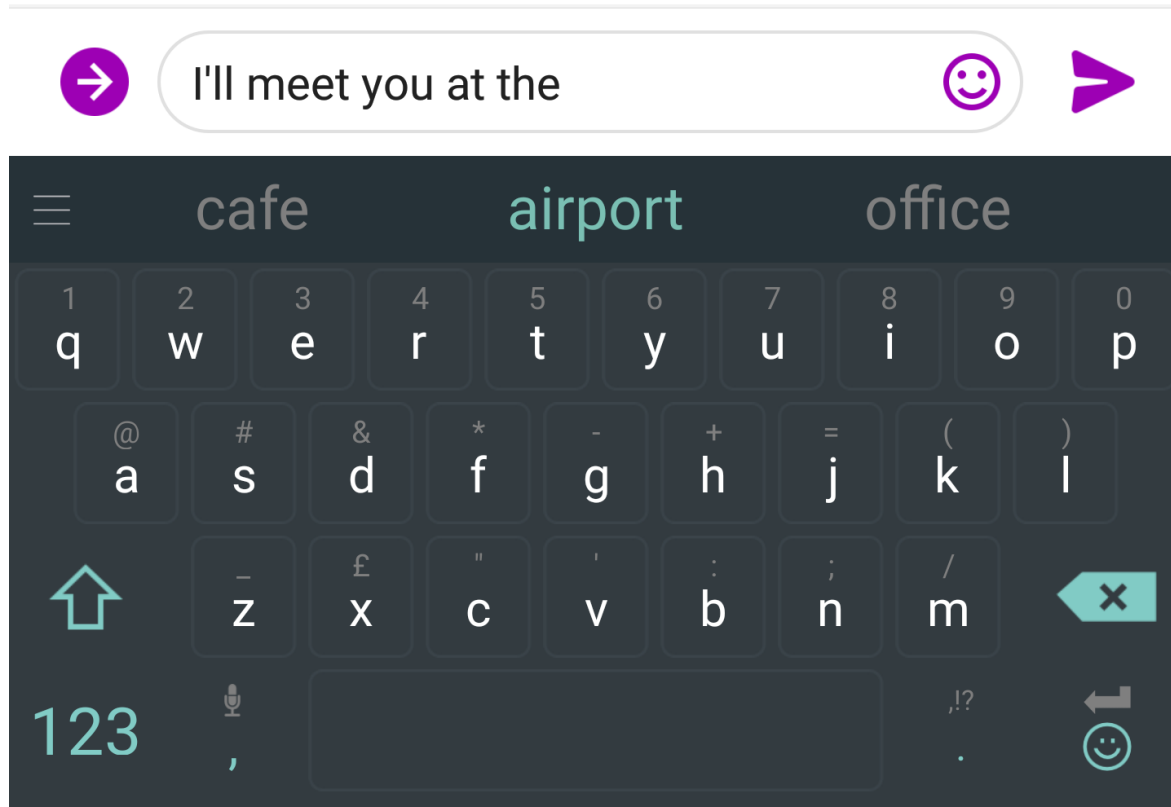
- You can also think of a Language Model as a system that **assigns a probability to a piece of text**
- For example, if we have some text  $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(T)}$ , then the probability of this text (according to the Language Model) is:

$$\begin{aligned} P(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(T)}) &= P(\mathbf{x}^{(1)}) \times P(\mathbf{x}^{(2)} | \mathbf{x}^{(1)}) \times \dots \times P(\mathbf{x}^{(T)} | \mathbf{x}^{(T-1)}, \dots, \mathbf{x}^{(1)}) \\ &= \prod_{t=1}^T P(\mathbf{x}^{(t)} | \mathbf{x}^{(t-1)}, \dots, \mathbf{x}^{(1)}) \end{aligned}$$



This is what our LM provides

# You use Language Models every day!



# You use Language Models every day!

## ChatGPT

### Examples

"Explain quantum computing in simple terms"

"Got any creative ideas for a 10 year old's birthday?"

"How do I make an HTTP request in Javascript?"

### Capabilities

Remembers what user said earlier in the conversation

Allows user to provide follow-up corrections

Trained to decline inappropriate requests

### Limitations

May occasionally generate incorrect information

May occasionally produce harmful instructions or biased content

Limited knowledge of world and events after 2021

ChatGPT is optimized for dialogue. Our goal is to make AI systems more natural to interact with, and your feedback will help us improve our system.

The screenshot shows a code editor with a Rust file named `mod.rs`. The code defines a `TransportStack` struct and implements its `listen` and `accept` methods. A chat window on the right is titled "CHAT COMPOSER" and contains the following text:

mod.rs Current File  
Could you make it easier to switch certificates in the transport listeners?

I'll help modify the code to make certificate switching more flexible. The main changes will be to enhance the `TlsAccept` trait and modify how certificates are handled in the `TlsSettings`. Here are the key changes:

```
mod.rs
// ... existing code ...

// The APIs to customize things like certificate during TLS ser
#[async_trait]
pub trait TlsAccept {
    // Called during TLS handshake to select and apply certific
    // Returns true if certificates were successfully applied
    async fn certificate_callback(&self, ssl: &mut TlsRef) -> bo
    // Default implementation does nothing and indicates no
    false
}

// Called when certificate selection fails
async fn on_cert_error(&self, _error: &str) {
    // Default implementation does nothing
}

// Add a default no-op implementation that can be used when no c
#[derive(Default)]
```

At the bottom of the chat window, it says "Ask followup (#Y), ↑ to select" and "claude-3.5-sonnet-20241022 @ Mention no context chat codebase".

# Why should we care about Language Modeling?

- Language Modeling is a **benchmark task** that helps us **measure our progress** on predicting language use
- Language Modeling is a **subcomponent** of many NLP tasks, especially those involving **generating text** or **estimating the probability of text**:
  - Predictive typing
  - Speech recognition
  - Handwriting recognition
  - Spelling/grammar correction
  - Authorship identification
  - Machine translation
  - Summarization
  - Dialogue
  - etc.
- Everything else in NLP has been rebuilt upon Language Modeling: **ChatGPT is an LM!**

# What can you do with next-word prediction?

- A sufficiently strong (!) language model can do many, many things

*Stanford University is located in \_\_\_\_\_, California. [Trivia]*

*I put \_\_\_ fork down on the table. [syntax]*

*The woman walked across the street, checking for traffic over \_\_\_ shoulder. [coreference]*

*I went to the ocean to see the fish, turtles, seals, and \_\_\_\_\_. [lexical semantics/topic]*

*Overall, the value I got from the two hours watching it was the sum total of the popcorn and the drink. The movie was \_\_\_\_\_. [sentiment]*

*Iroh went into the kitchen to make some tea. Standing next to Iroh, Zuko pondered his destiny.*

*Zuko left the \_\_\_\_\_. [some reasoning – this is harder]*

*I was thinking about the sequence that goes 1, 1, 2, 3, 5, 8, 13, 21, \_\_\_\_\_. [some basic arithmetic]*



# Next word prediction (2019 – GPT2)

- Fixing /.ssh/ Permissions in Cygwin
- In Cygwin on Windows I found that I could not set the permissions of my /.ssh/ folder to be 0600, as is required for ssh to allow you to use keys. The symptom I had was that I no matter what I did, it always modified the owner and the group

, making these files read-only. I tried just starting ssh and using a program like passwd which modifies the owner/group to 0644 permissions, but it didn't work. It was annoying but not nearly as bad as it could be, just annoying to have to fiddle with permissions. So I decided to do this as the reverse: For each .ssh folder in your Cygwin directory, set the permission of each file and directory to 2644. Then start ssh with: ssh-add /.ssh/ The first time you connect, your permissions of the /.ssh folder that you just had in Cygwin will be set to 0600. Then every time you connect again, your permissions will change to 0644.

Stylistically correct, content wise, questionable

# Next word prediction (2020 – GPT3)

Context → Title: The Blitz

Background: From the German point of view, March 1941 saw an improvement. The Luftwaffe flew 4,000 sorties that month, including 12 major and three heavy attacks. The electronic war intensified but the Luftwaffe flew major inland missions only on moonlit nights. Ports were easier to find and made better targets. To confuse the British, radio silence was observed until the bombs fell. X- and Y-Gerät beams were placed over false targets and switched only at the last minute. Rapid frequency changes were introduced for X-Gerät, whose wider band of frequencies and greater tactical flexibility ensured it remained effective at a time when British selective jamming was degrading the effectiveness of Y-Gerät.

Q: How many sorties were flown in March 1941?

A: 4,000

Q: When did the Luftwaffe fly inland missions?

A:

---

Target Completion → only on moonlit nights

---

The model can perform tasks when given a few examples ('in-context learning')

# n-gram Language Models

*the students opened their \_\_\_\_\_*

- **Question:** How to learn a Language Model?
- **Answer** (pre- Deep Learning): learn an *n*-gram Language Model!
- **Definition:** An *n*-gram is a chunk of *n* consecutive words.
  - **unigrams:** “the”, “students”, “opened”, “their”
  - **bigrams:** “the students”, “students opened”, “opened their”
  - **trigrams:** “the students opened”, “students opened their”
  - **four-grams:** “the students opened their”
- **Idea:** Collect statistics about how frequent different n-grams are and use these to predict next word.

# n-gram Language Models

- First we make a **Markov assumption**:  $x^{(t+1)}$  depends only on the preceding  $n-1$  words

$$P(\mathbf{x}^{(t+1)} | \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(1)}) = P(\mathbf{x}^{(t+1)} | \overbrace{\mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)}}^{n-1 \text{ words}}) \quad (\text{assumption})$$

prob of a n-gram  $\rightarrow$

$$= P(\mathbf{x}^{(t+1)}, \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)})$$

prob of a (n-1)-gram  $\rightarrow$

$$P(\mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)})$$

(definition of conditional prob)

- **Question:** How do we get these  $n$ -gram and  $(n-1)$ -gram probabilities?
- **Answer:** By **counting** them in some large corpus of text!

$$\approx \frac{\text{count}(\mathbf{x}^{(t+1)}, \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)})}{\text{count}(\mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)})} \quad (\text{statistical approximation})$$

# n-gram Language Models: Example

Suppose we are learning a 4-gram Language Model.

~~as the proctor started the clock, the~~ *students opened their* \_\_\_\_\_  
discard } condition on this

$$P(\mathbf{w} | \text{students opened their}) = \frac{\text{count}(\text{students opened their } \mathbf{w})}{\text{count}(\text{students opened their})}$$

For example, suppose that in the corpus:

- “students opened their” occurred 1000 times
- “students opened their *books*” occurred 400 times
  - $\rightarrow P(\text{books} | \text{students opened their}) = 0.4$
- “students opened their *exams*” occurred 100 times
  - $\rightarrow P(\text{exams} | \text{students opened their}) = 0.1$

Should we have discarded the “proctor” context?

# Sparsity Problems with n-gram Language Models

## Sparsity Problem 1

**Problem:** What if “students opened their  $w$ ” never occurred in data? Then  $w$  has probability 0!

**(Partial) Solution:** Add small  $\delta$  to the count for every  $w \in V$ . This is called *smoothing*.

$$P(w|\text{students opened their}) = \frac{\text{count}(\text{students opened their } w)}{\text{count}(\text{students opened their})}$$

## Sparsity Problem 2

**Problem:** What if “students opened their” never occurred in data? Then we can’t calculate probability for *any*  $w$ !

**(Partial) Solution:** Just condition on “opened their” instead. This is called *backoff*.

**Note:** Increasing  $n$  makes sparsity problems worse. Typically, we can’t have  $n$  bigger than 5.

# Storage Problems with $n$ -gram Language Models

**Storage:** Need to store count for all  $n$ -grams you saw in the corpus.

$$P(\mathbf{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \mathbf{w})}{\text{count}(\text{students opened their})}$$

Increasing  $n$  or increasing corpus increases model size!

# n-gram Language Models in practice

- You can build a simple trigram Language Model over a 1.7 million word corpus (Reuters) in a few seconds on your laptop\*

Business and financial news

today the \_\_\_\_\_

get probability distribution

company	0.153
bank	0.153
price	0.077
italian	0.039
emirate	0.039
...	

**Sparsity problem:**  
not much granularity  
in the probability  
distribution

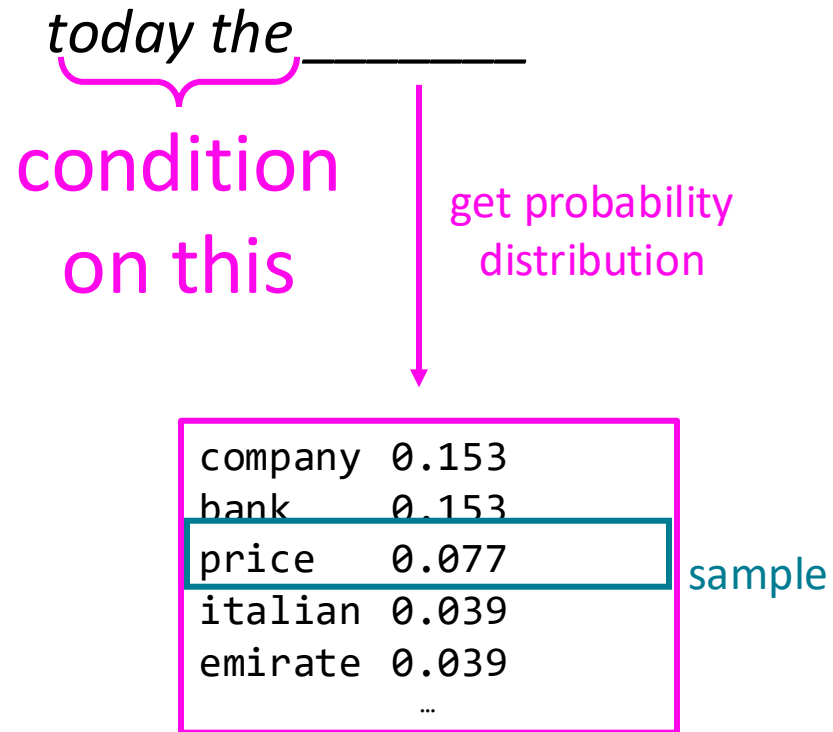
Otherwise, seems reasonable!

\* Try for yourself: <https://nlpforhackers.io/language-models/>



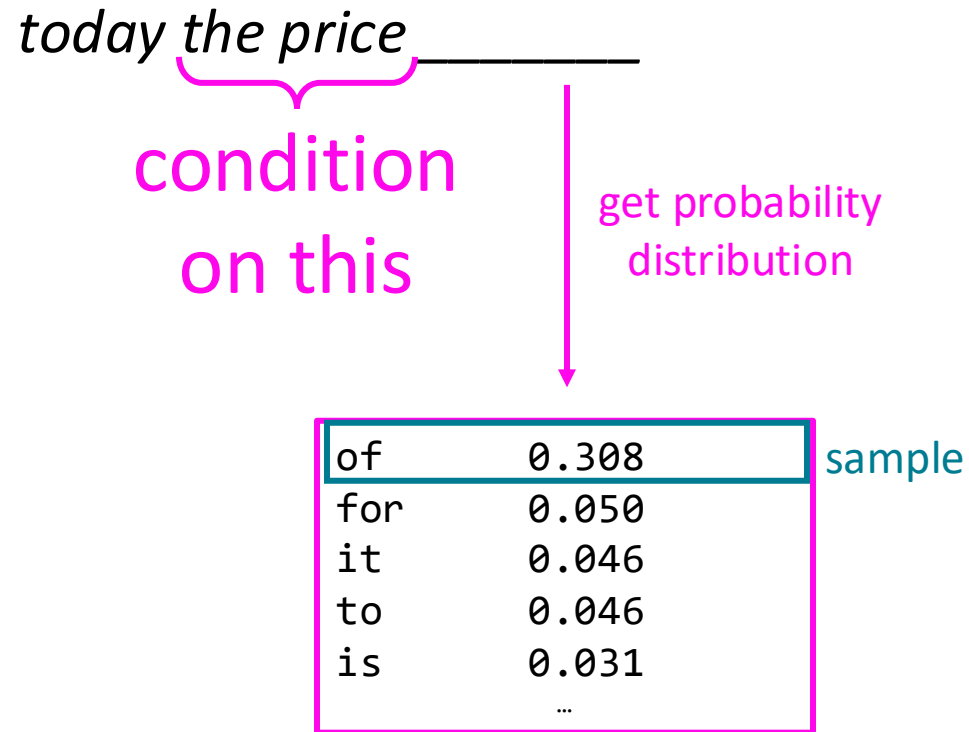
# Generating text with a n-gram Language Model

You can also use a Language Model to generate text



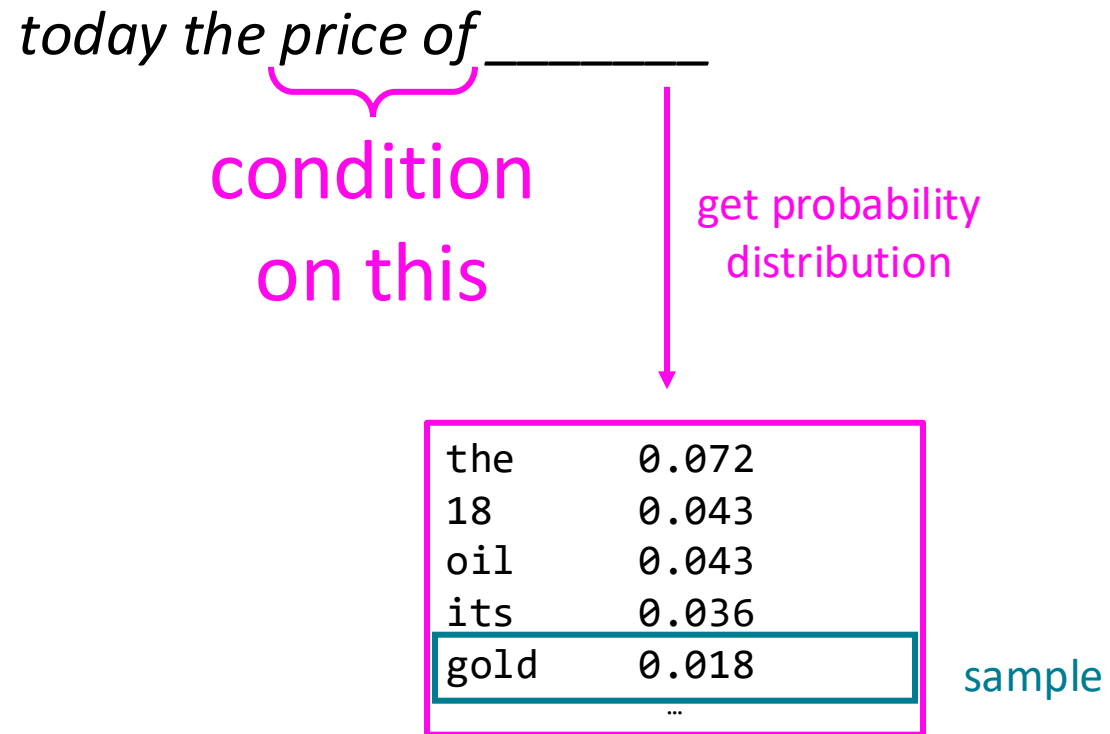
# Generating text with a n-gram Language Model

You can also use a Language Model to generate text



# Generating text with a n-gram Language Model

You can also use a Language Model to generate text



# Generating text with a n-gram Language Model

You can also use a Language Model to generate text

*today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share .*

Surprisingly grammatical!

...but **incoherent**. We need to consider more than three words at a time if we want to model language well.

But increasing  $n$  worsens sparsity problem,  
and increases model size...

# Evaluating Language Models

- The standard **evaluation metric** for Language Models is **perplexity**.

$$\text{perplexity} = \prod_{t=1}^T \left( \frac{1}{P_{\text{LM}}(\mathbf{x}^{(t+1)} | \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(1)})} \right)^{1/T}$$

Normalized by number of words

Inverse probability of corpus, according to Language Model

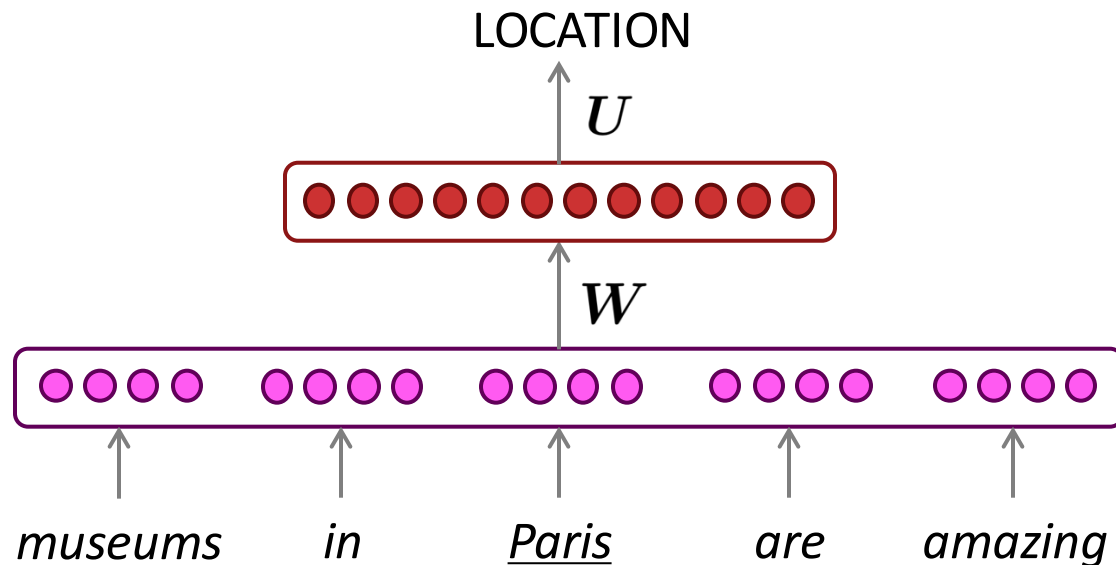
- This is equal to the exponential of the cross-entropy loss  $J(\theta)$  :

$$= \prod_{t=1}^T \left( \frac{1}{\hat{y}_{\mathbf{x}_{t+1}}^{(t)}} \right)^{1/T} = \exp \left( \frac{1}{T} \sum_{t=1}^T -\log \hat{y}_{\mathbf{x}_{t+1}}^{(t)} \right) = \exp(J(\theta))$$

**Lower perplexity is better!**

# How to build a *neural* language model?

- Recall the Language Modeling task:
  - Input: sequence of words  $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(t)}$
  - Output: prob. dist. of the next word  $P(\mathbf{x}^{(t+1)} | \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(1)})$
- How about a **window-based neural model**?
  - We saw this applied to Named Entity Recognition in Lecture 2:



# A fixed-window neural Language Model

~~us the proctor started the clock~~  
discard

the students opened their \_\_\_\_\_  
fixed window

# A fixed-window neural Language Model

output distribution

$$\hat{y} = \text{softmax}(U\mathbf{h} + \mathbf{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

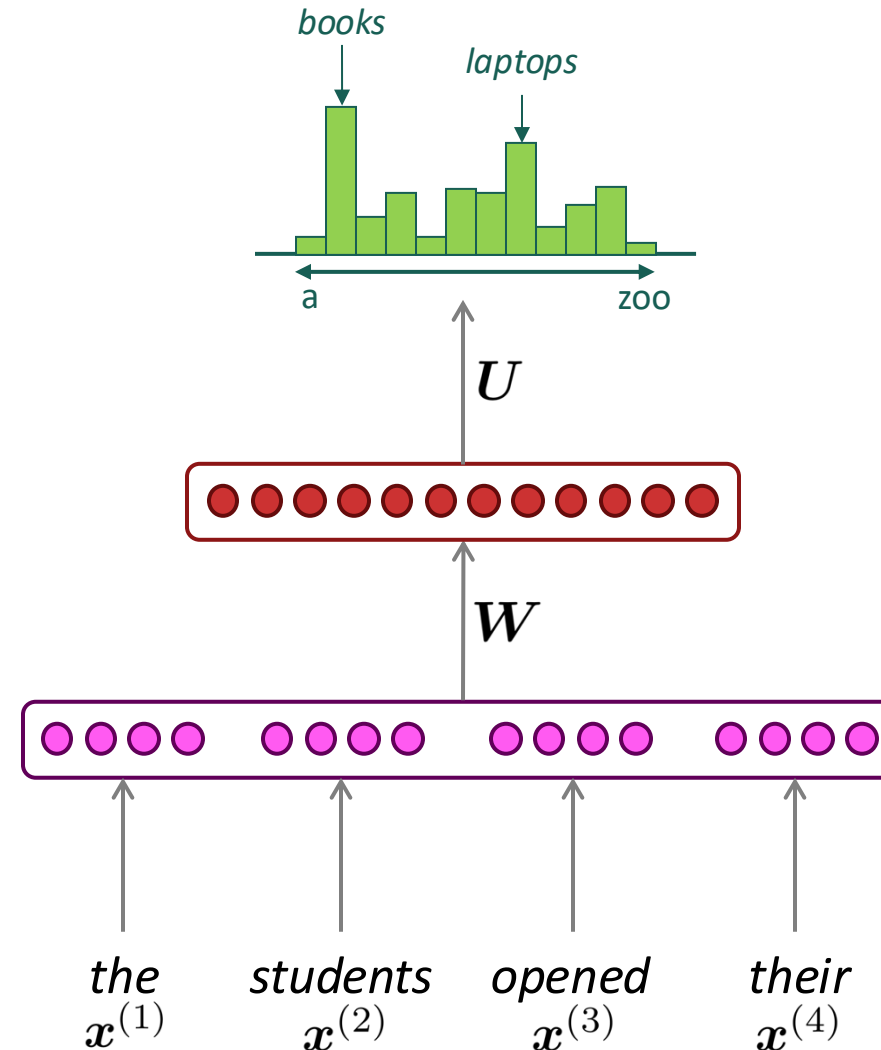
$$\mathbf{h} = f(\mathbf{W}\mathbf{e} + \mathbf{b}_1)$$

concatenated word embeddings

$$\mathbf{e} = [e^{(1)}; e^{(2)}; e^{(3)}; e^{(4)}]$$

words / one-hot vectors

$$\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}, \mathbf{x}^{(4)}$$





# A fixed-window neural Language Model

Approximately: Y. Bengio, et al. (2000/2003): A Neural Probabilistic Language Model

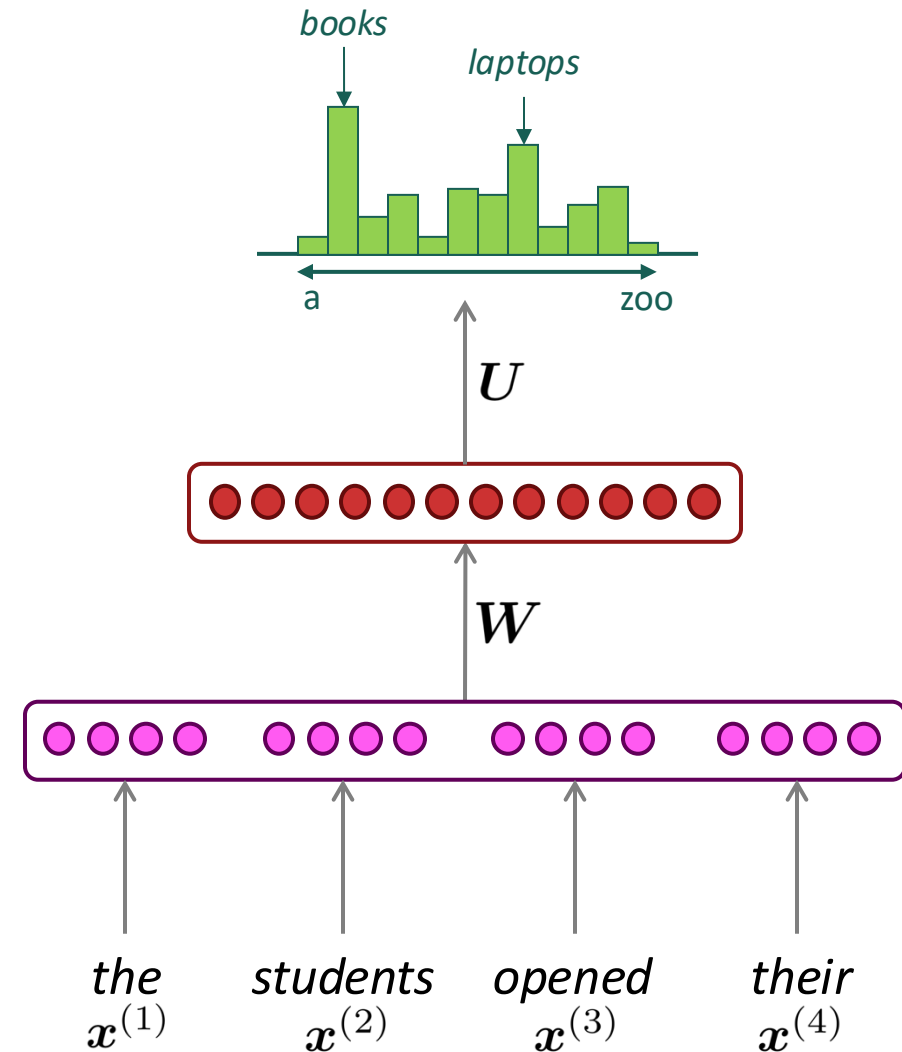
**Improvements** over  $n$ -gram LM:

- No sparsity problem
- Don't need to store all observed  $n$ -grams

Remaining **problems**:

- Fixed window is **too small**
  - Enlarging window enlarges  $W$
  - Window can never be large enough!
  - $x^{(1)}$  and  $x^{(2)}$  are multiplied by completely different weights in  $W$ .
- No symmetry** in how the inputs are processed.

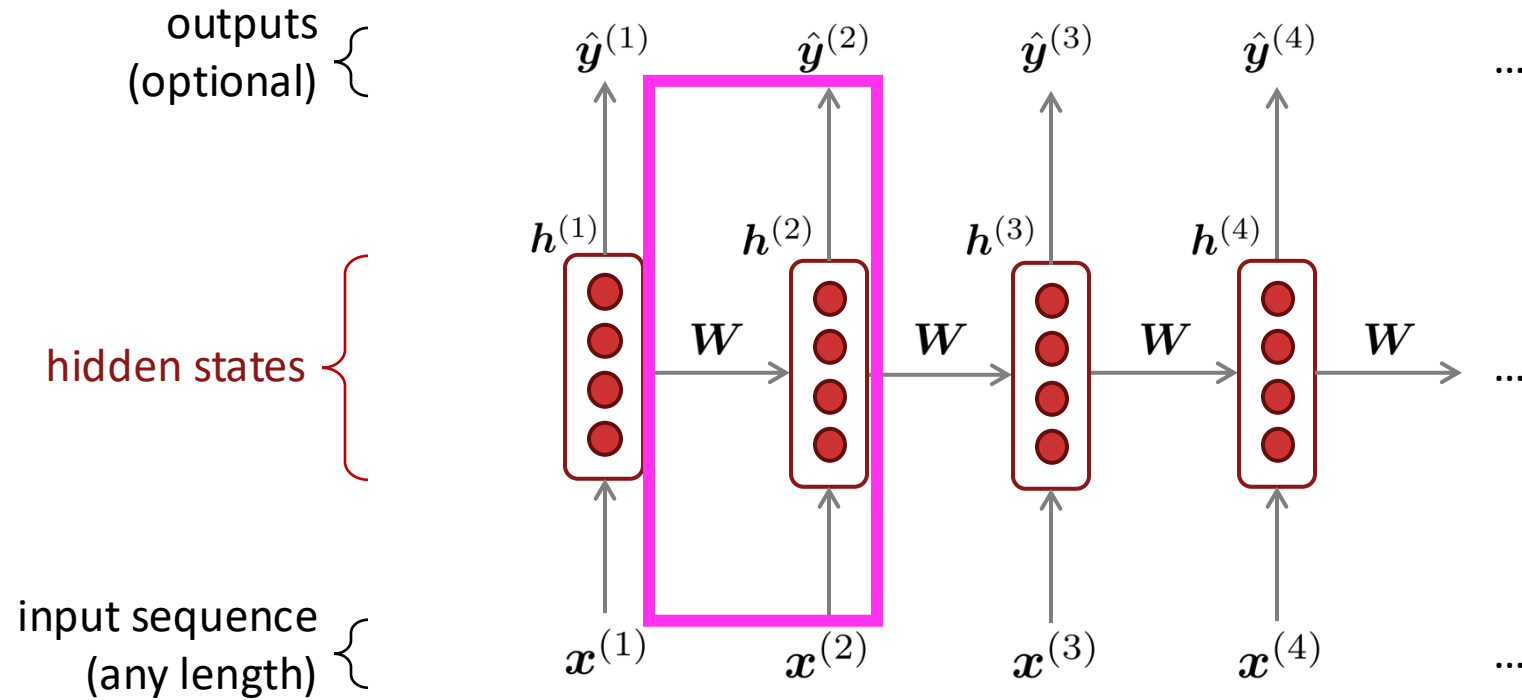
We need a neural architecture that can process *any length input*



## 2. Recurrent Neural Networks (RNN)

A family of neural architectures

**Core idea:** Apply the same weights  $W$  repeatedly



# A Simple RNN Language Model

$$\hat{y}^{(4)} = P(x^{(5)} | \text{the students opened their})$$

output distribution

$$\hat{y}^{(t)} = \text{softmax}(U\mathbf{h}^{(t)} + \mathbf{b}_2) \in \mathbb{R}^{|V|}$$

hidden states

$$\mathbf{h}^{(t)} = \sigma(\mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_e \mathbf{e}^{(t)} + \mathbf{b}_1)$$

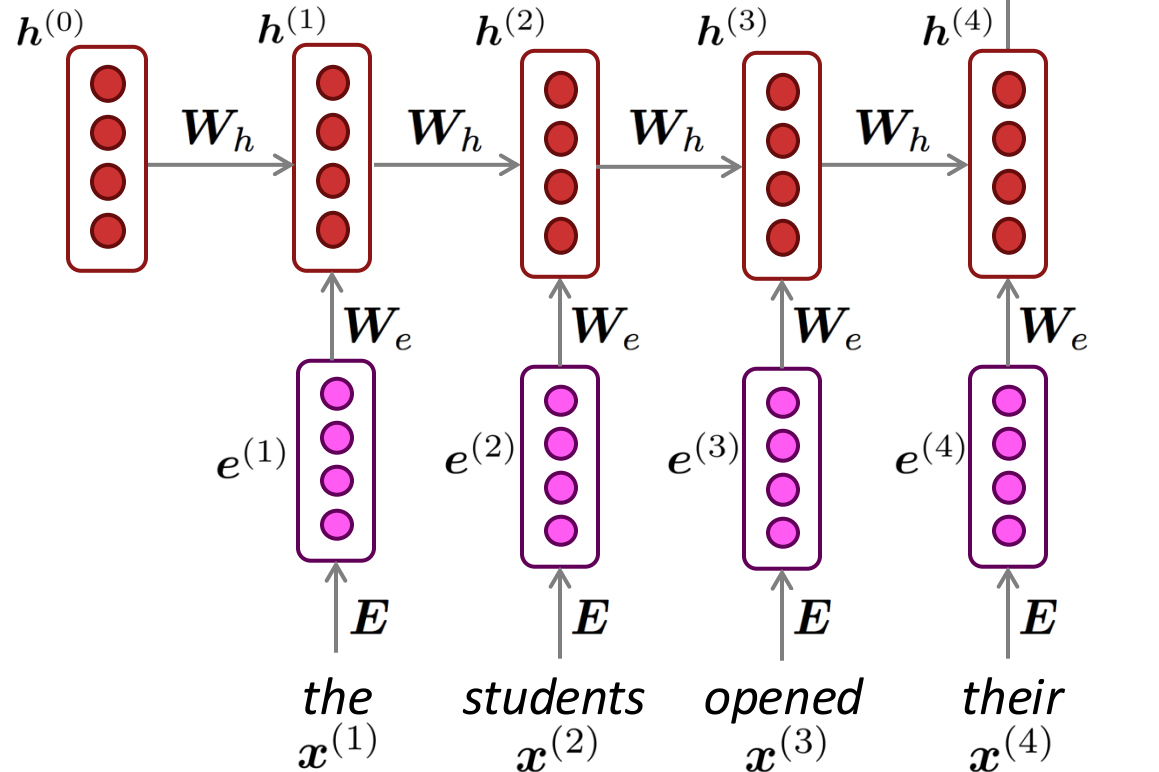
$\mathbf{h}^{(0)}$  is the initial hidden state

word embeddings

$$\mathbf{e}^{(t)} = \mathbf{E}x^{(t)}$$

words / one-hot vectors

$$\mathbf{x}^{(t)} \in \mathbb{R}^{|V|}$$



*Note: this input sequence could be much longer now!*

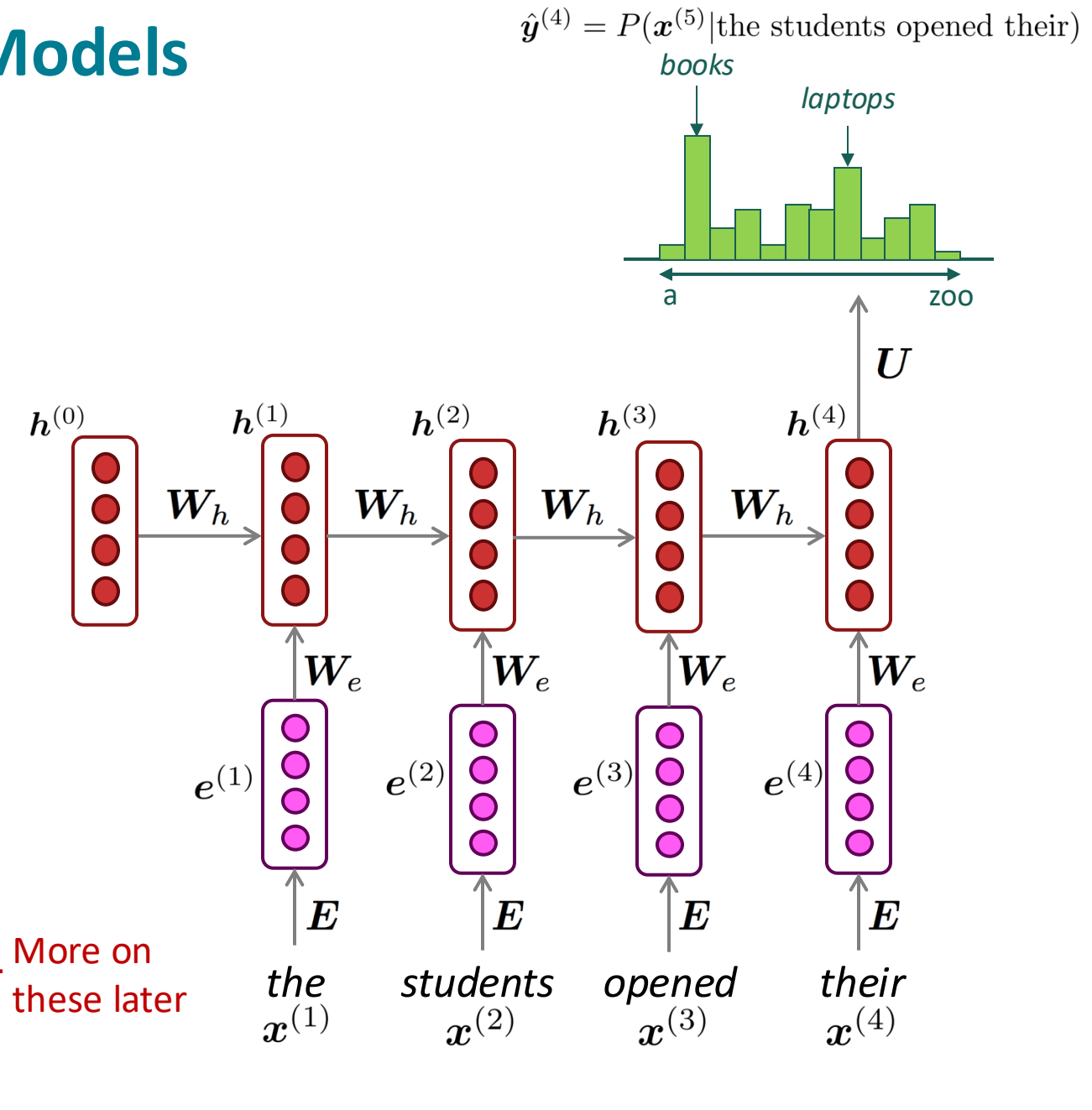
# RNN Language Models

## RNN Advantages:

- Can process **any length** input
- Computation for step  $t$  can (in theory) use information from **many steps back**
- **Model size doesn't increase** for longer input context
- Same weights applied on every timestep, so there is **symmetry** in how inputs are processed.

## RNN Disadvantages:

- Recurrent computation is **slow**
- In practice, difficult to access information from **many steps back**



More on these later

# Training an RNN Language Model

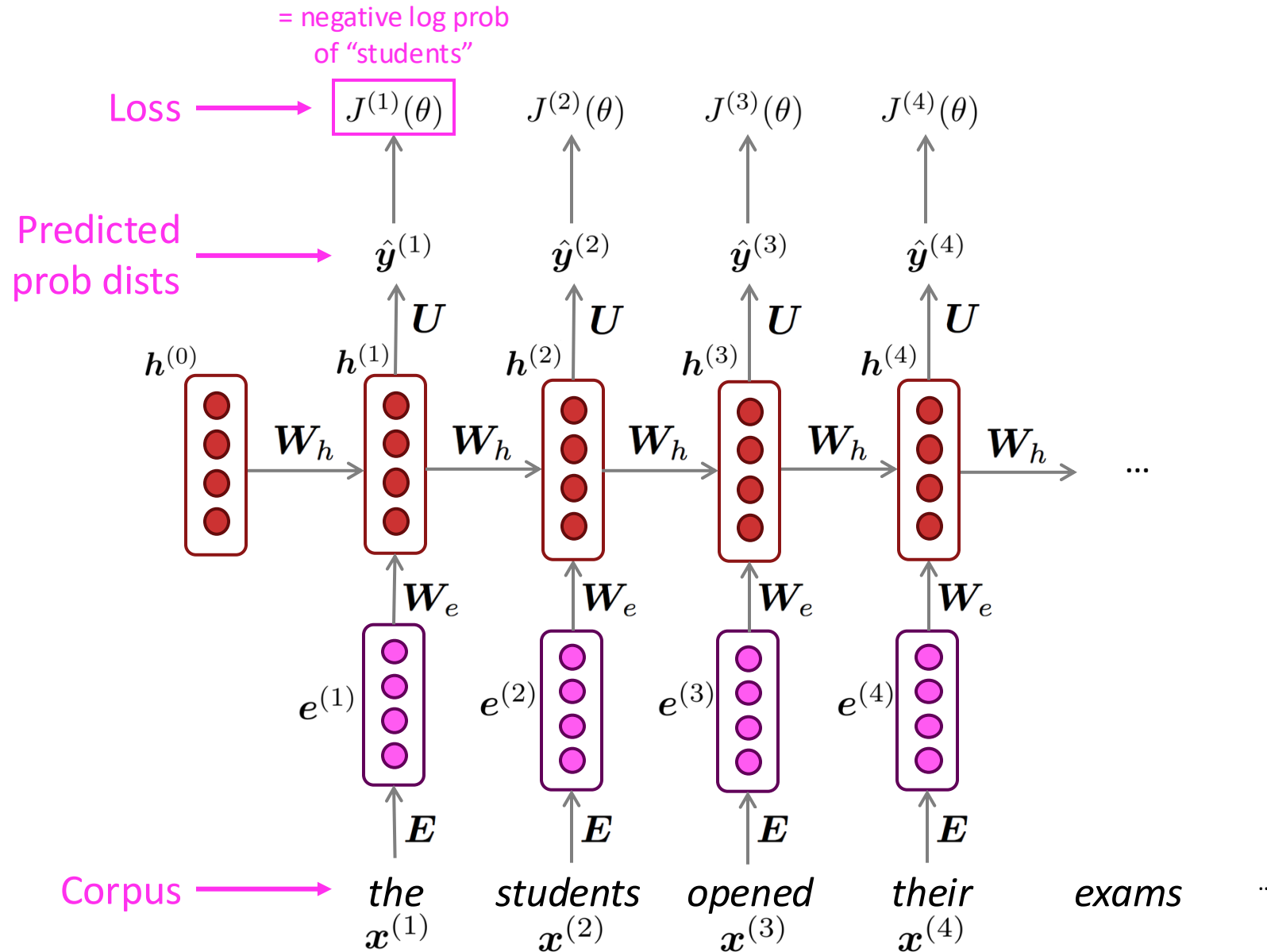
- Get a **big corpus of text** which is a sequence of words  $x^{(1)}, \dots, x^{(T)}$
- Feed into RNN-LM; compute output distribution  $\hat{\mathbf{y}}^{(t)}$  **for every step  $t$** .
  - i.e., predict probability dist of *every word*, given words so far
- **Loss function** on step  $t$  is **cross-entropy** between predicted probability distribution  $\hat{\mathbf{y}}^{(t)}$ , and the true next word  $\mathbf{y}^{(t)}$  (one-hot for  $x^{(t+1)}$ ):

$$J^{(t)}(\theta) = CE(\mathbf{y}^{(t)}, \hat{\mathbf{y}}^{(t)}) = - \sum_{w \in V} \mathbf{y}_w^{(t)} \log \hat{\mathbf{y}}_w^{(t)} = - \log \hat{\mathbf{y}}_{x_{t+1}}^{(t)}$$

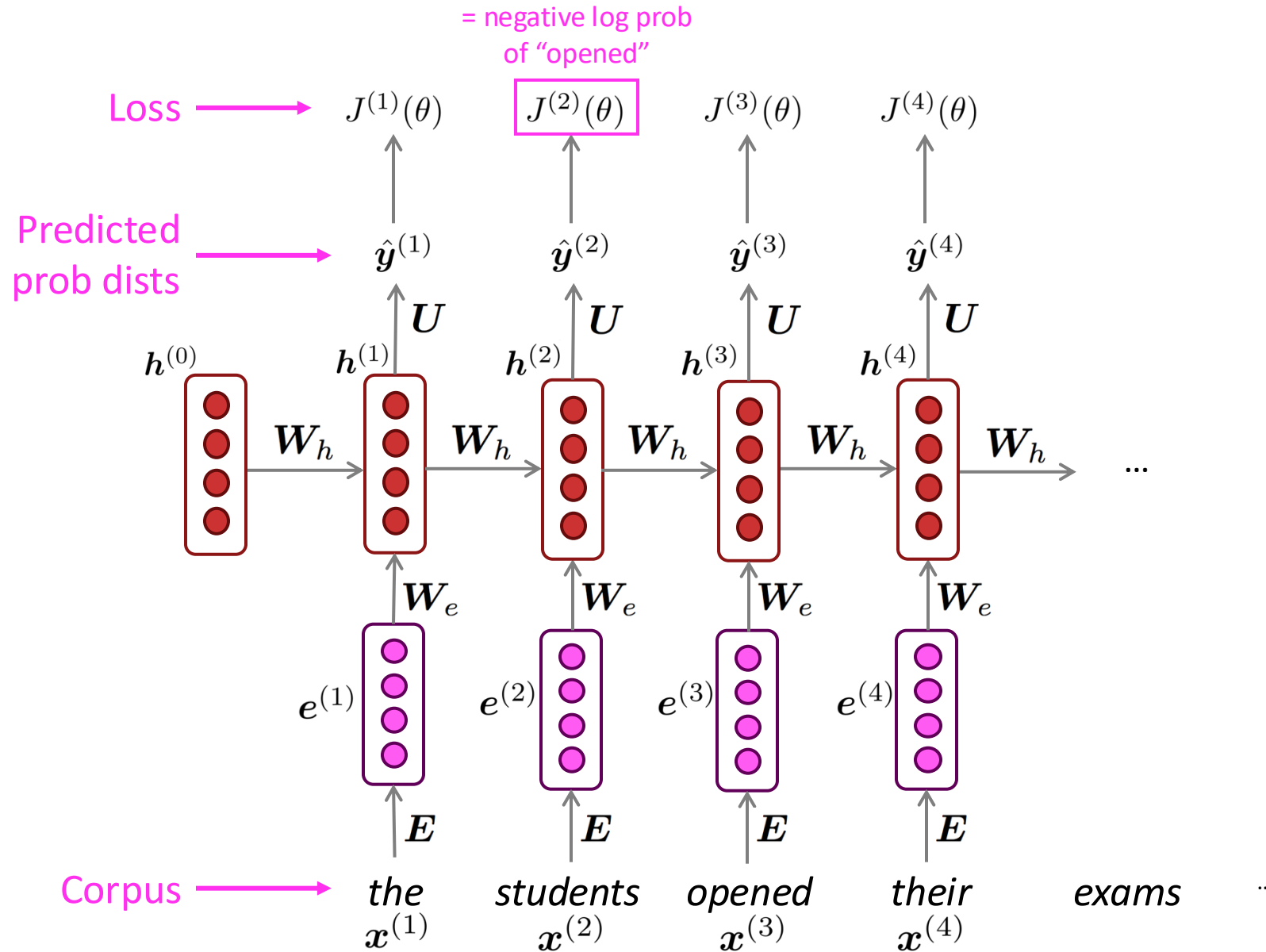
- Average this to get **overall loss** for entire training set:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^T J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^T - \log \hat{\mathbf{y}}_{x_{t+1}}^{(t)}$$

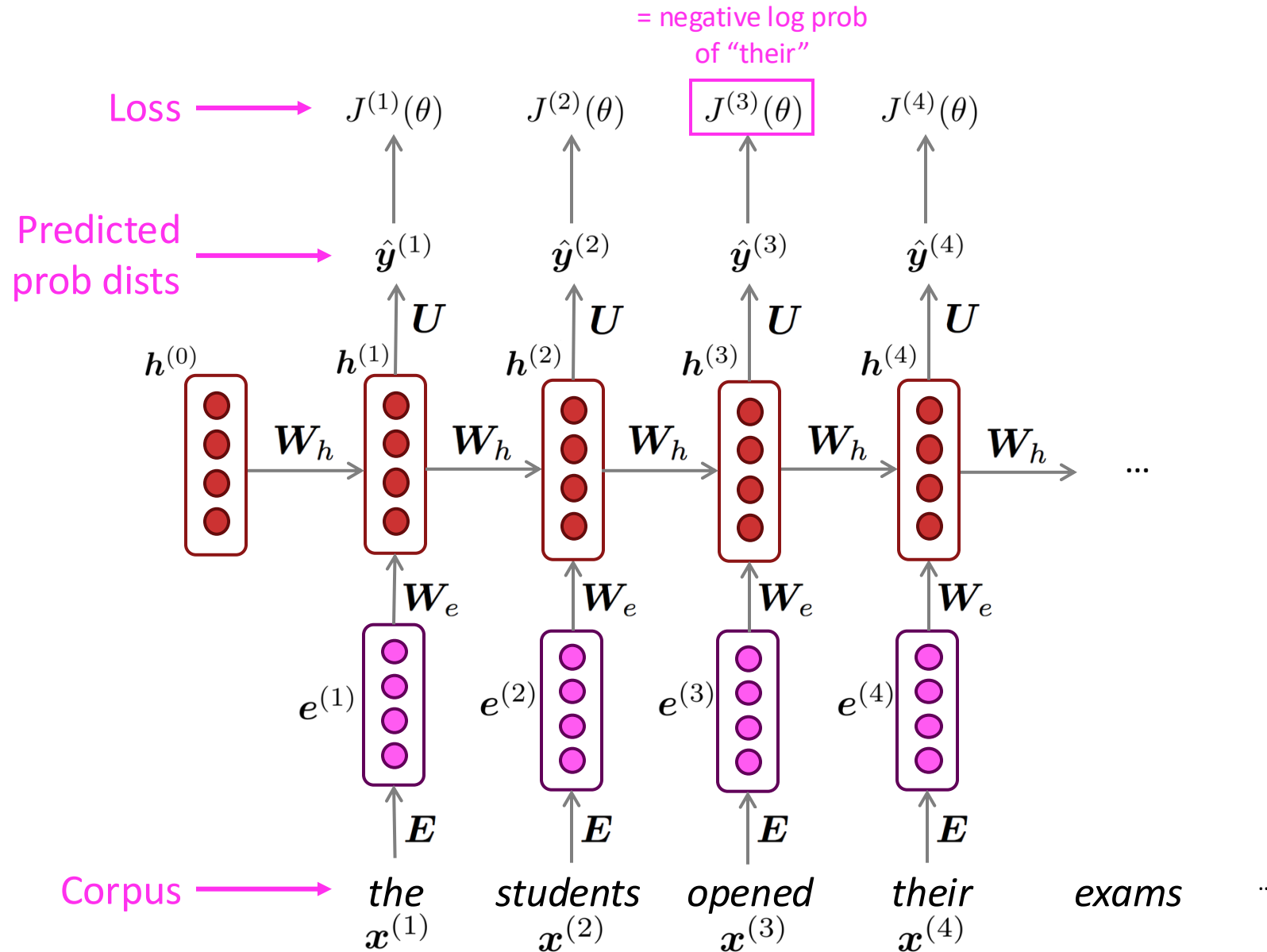
# Training an RNN Language Model



# Training an RNN Language Model

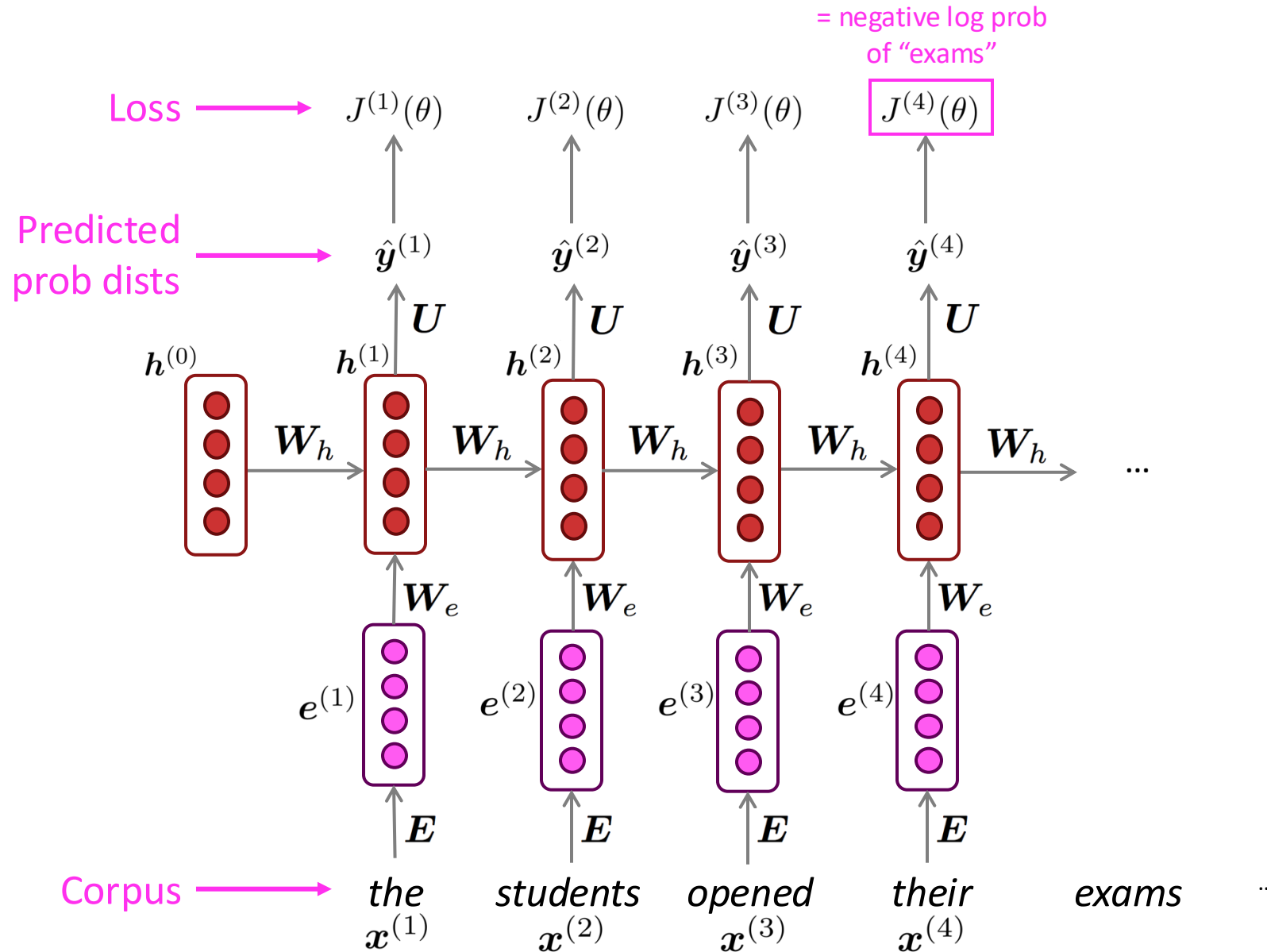


# Training an RNN Language Model



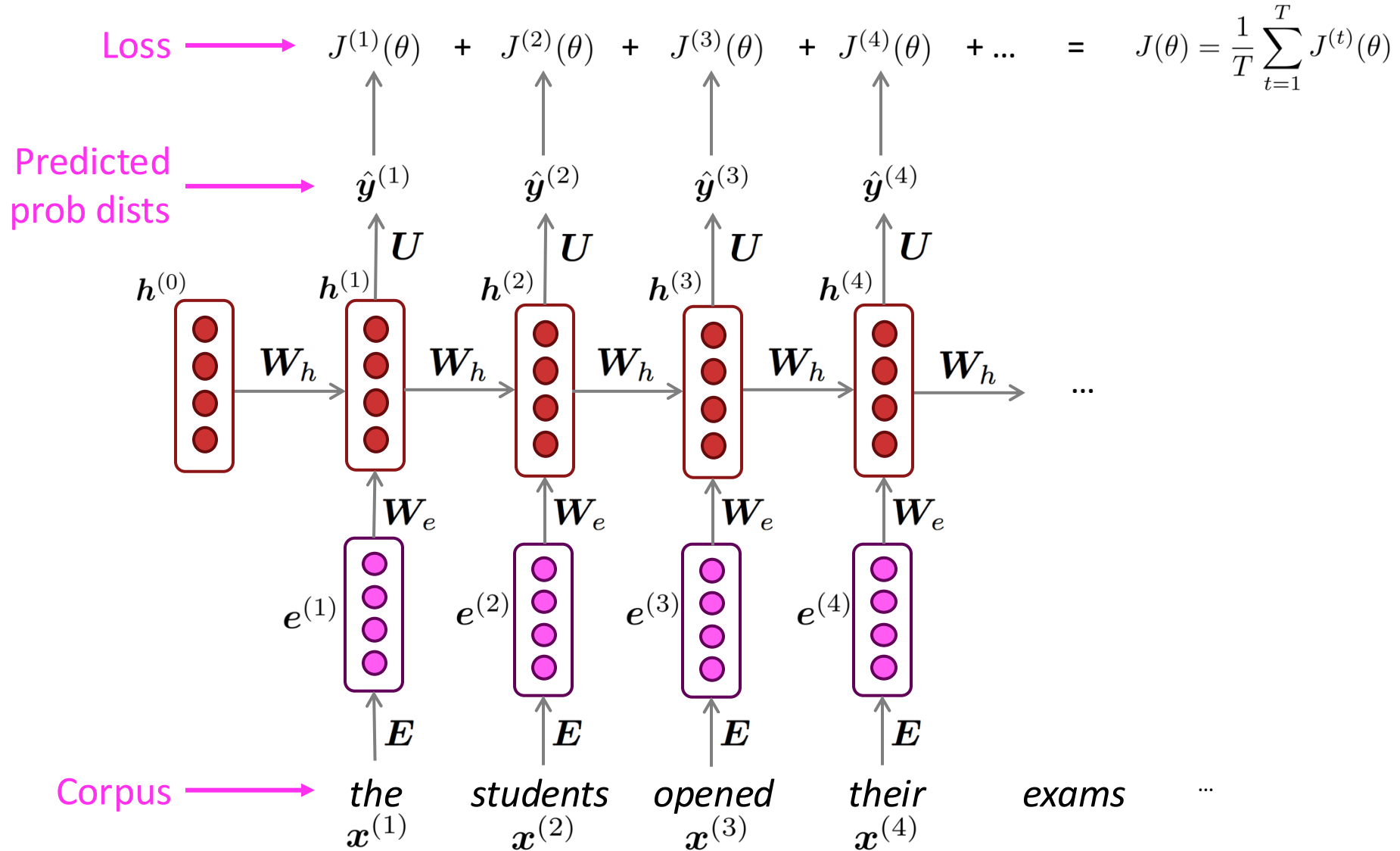


# Training an RNN Language Model



# Training an RNN Language Model

“Teacher forcing”



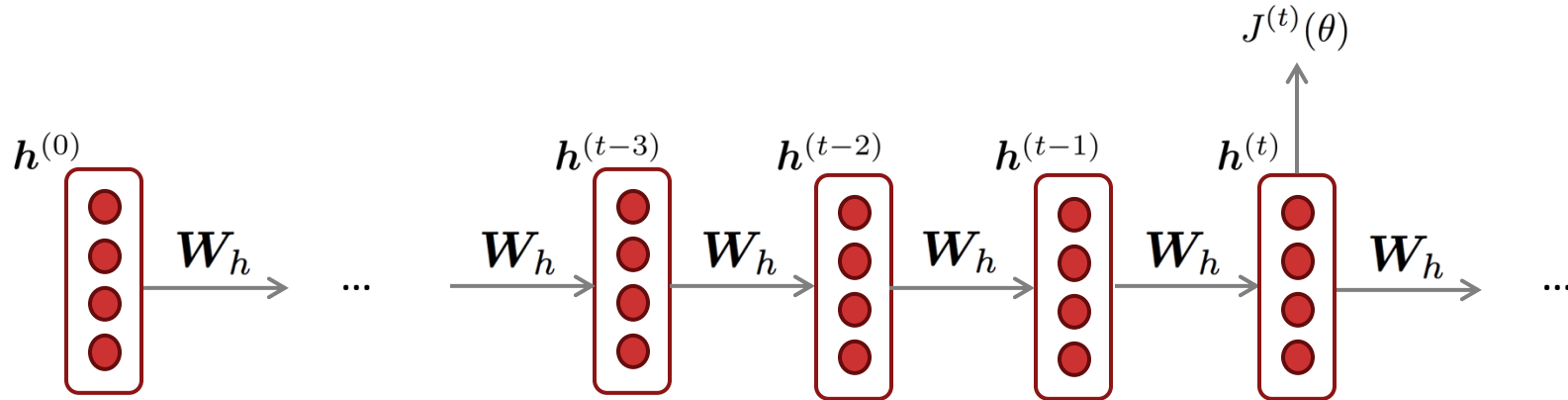
# Training a RNN Language Model

- However: Computing loss and gradients across **entire corpus**  $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(T)}$  at once is **too expensive** (memory-wise)!

$$J(\theta) = \frac{1}{T} \sum_{t=1}^T J^{(t)}(\theta)$$

- In practice, consider  $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(T)}$  as a **sentence** (or a **document**)
- Recall: **Stochastic Gradient Descent** allows us to compute loss and gradients for small chunk of data, and update.
- Compute loss  $J(\theta)$  for a sentence (actually, a batch of sentences), compute gradients and update weights. Repeat on a new batch of sentences.

# Backpropagation for RNNs



**Question:** What's the derivative of  $J^{(t)}(\theta)$  w.r.t. the **repeated** weight matrix  $W_h$  ?

**Answer:** 
$$\frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1}^t \frac{\partial J^{(t)}}{\partial W_h} \Big|_{(i)}$$

“The gradient w.r.t. a repeated weight is the sum of the gradient w.r.t. each time it appears”

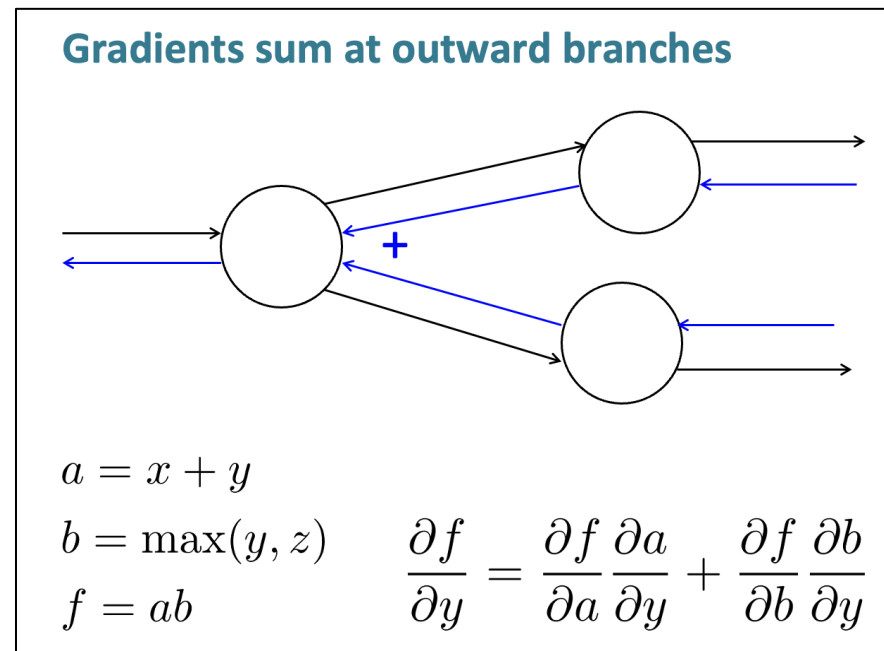
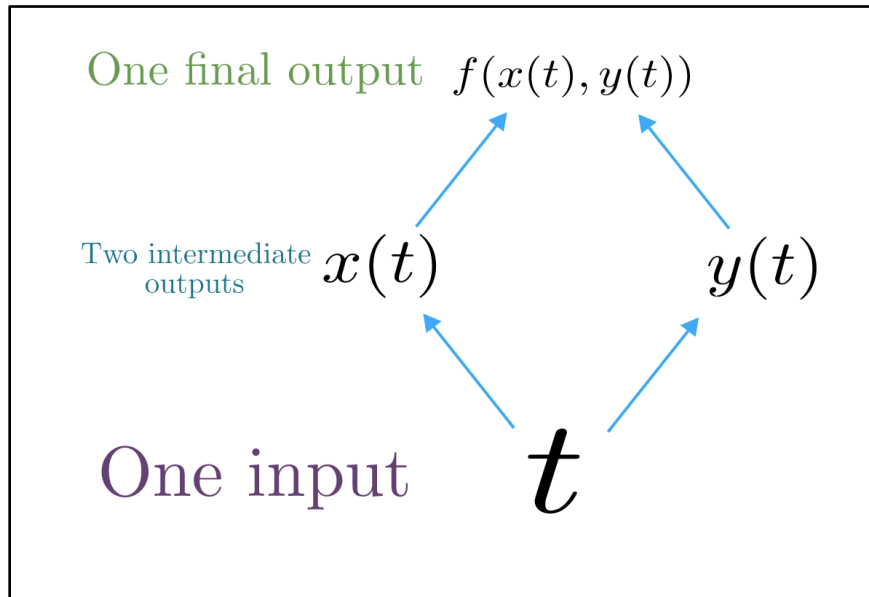
Why?

# Multivariable Chain Rule

- Given a multivariable function  $f(x, y)$ , and two single variable functions  $x(t)$  and  $y(t)$ , here's what the multivariable chain rule says:

$$\underbrace{\frac{d}{dt} f(x(t), y(t))}_{\text{Derivative of composition function}} = \frac{\partial f}{\partial x} \frac{dx}{dt} + \frac{\partial f}{\partial y} \frac{dy}{dt}$$

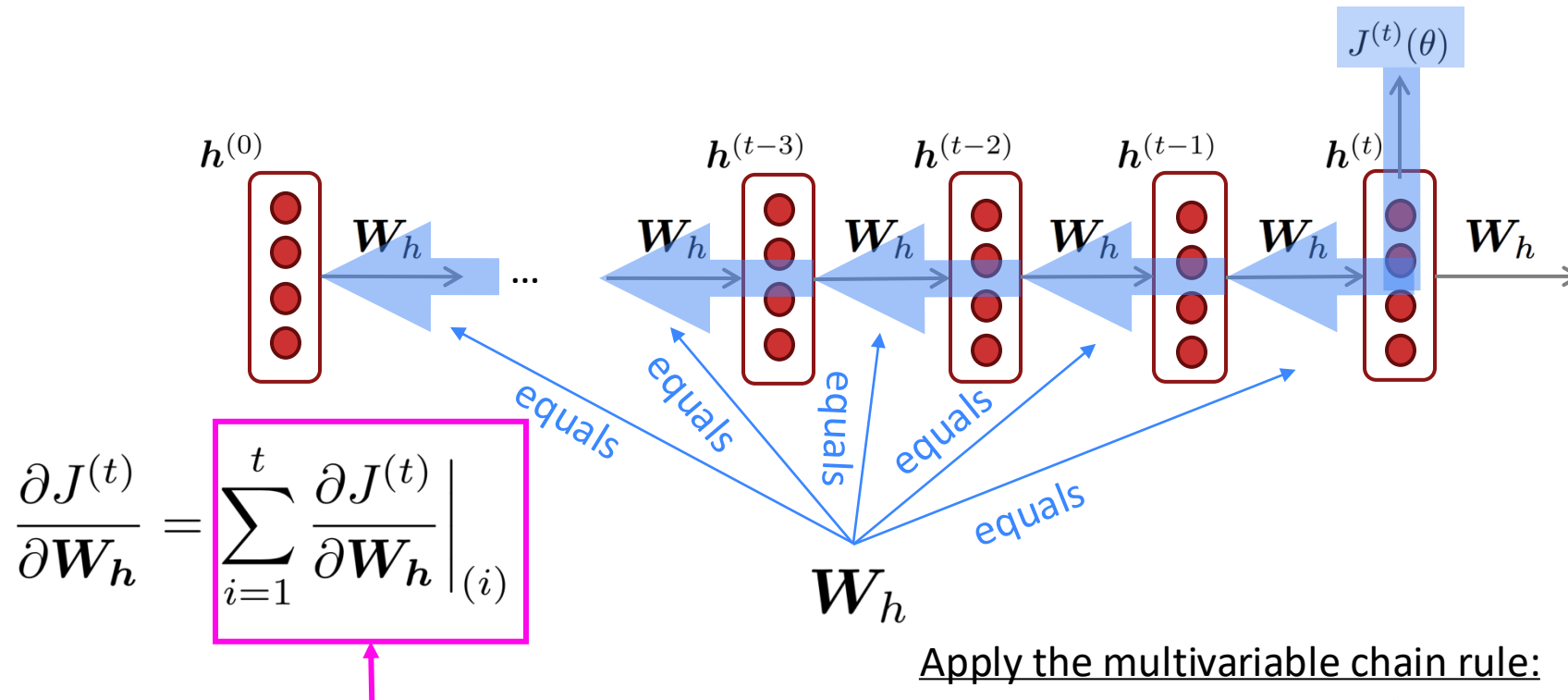
Derivative of composition function



Source:

<https://www.khanacademy.org/math/multivariable-calculus/multivariable-derivatives/differentiating-vector-valued-functions/a/multivariable-chain-rule-simple-version>

# Training the parameters of RNNs: Backpropagation for RNNs



In practice, often “truncated” after ~20 timesteps for training efficiency reasons

**Question:** How do we calculate this?

**Answer:** Backpropagate over timesteps  $i = t, \dots, 0$ , summing gradients as you go. This algorithm is called “**backpropagation through time**” [Werbos, P.G., 1988, *Neural Networks 1*, and others]

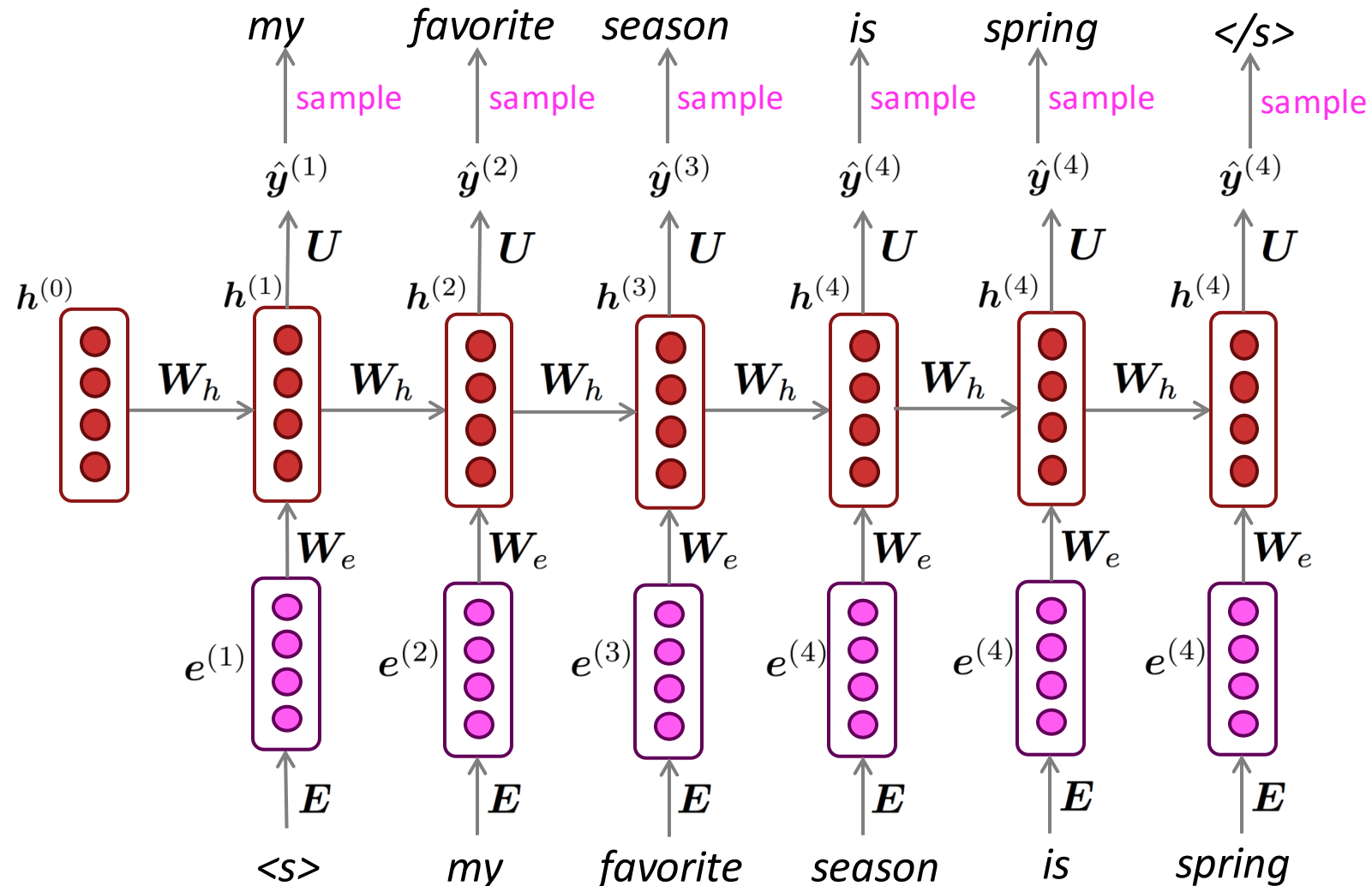
Apply the multivariable chain rule:

$$\begin{aligned} \frac{\partial J^{(t)}}{\partial \mathbf{W}_h} &= \sum_{i=1}^t \frac{\partial J^{(t)}}{\partial \mathbf{W}_h} \Big|_{(i)} \frac{\partial \mathbf{W}_h \Big|_{(i)}}{\partial \mathbf{W}_h} \\ &= \sum_{i=1}^t \frac{\partial J^{(t)}}{\partial \mathbf{W}_h} \Big|_{(i)} \end{aligned}$$

= 1

# Generating with an RNN Language Model (“Generating roll outs”)

Just like an n-gram Language Model, you can use a RNN Language Model to **generate text** by **repeated sampling**. Sampled output becomes next step’s input.



# Generating text with an RNN Language Model

Let's have some fun!

- You can train an RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on *Harry Potter*:



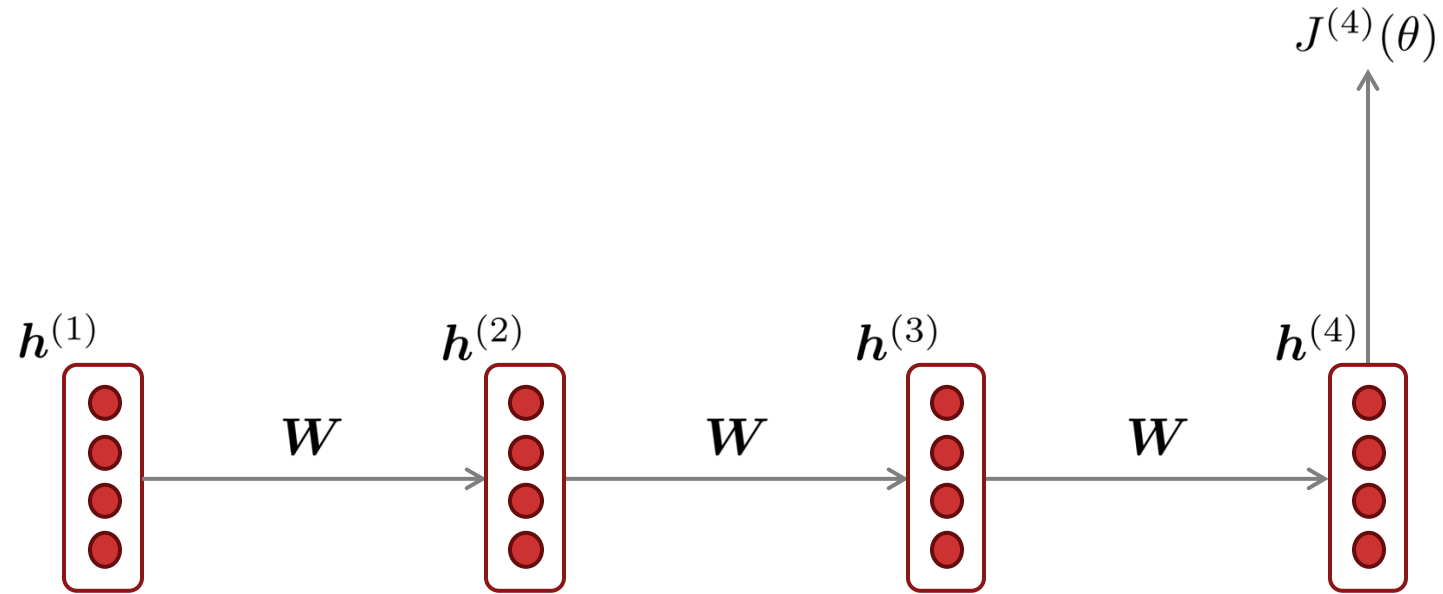
“Sorry,” Harry shouted, panicking—“I’ll leave those brooms in London, are they?”

“No idea,” said Nearly Headless Nick, casting low close by Cedric, carrying the last bit of treacle Charms, from Harry’s shoulder, and to answer him the common room perched upon it, four arms held a shining knob from when the spider hadn’t felt it seemed. He reached the teams too.

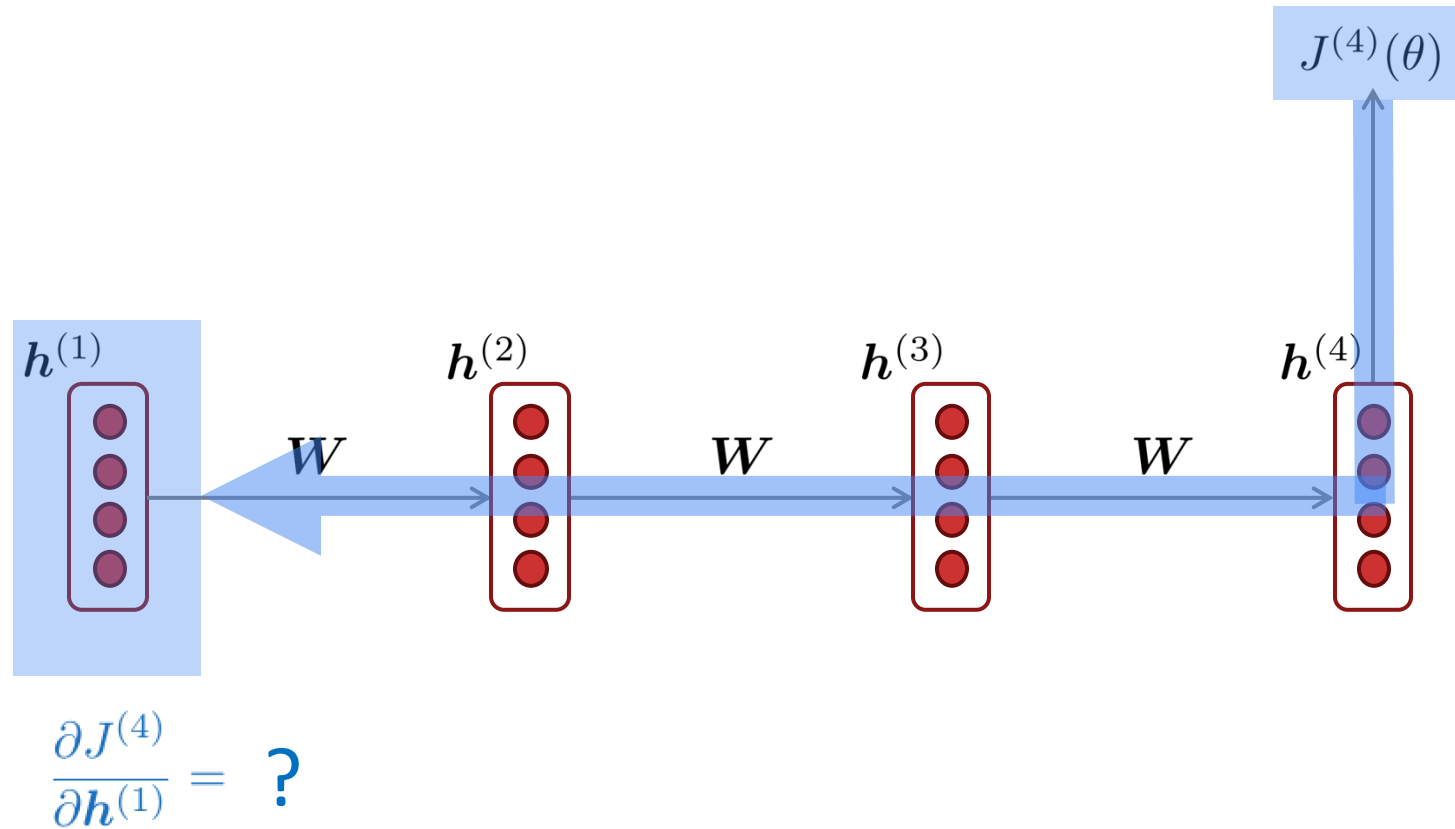
**Source:** <https://medium.com/deep-writing/harry-potter-written-by-artificial-intelligence-8a9431803da6>



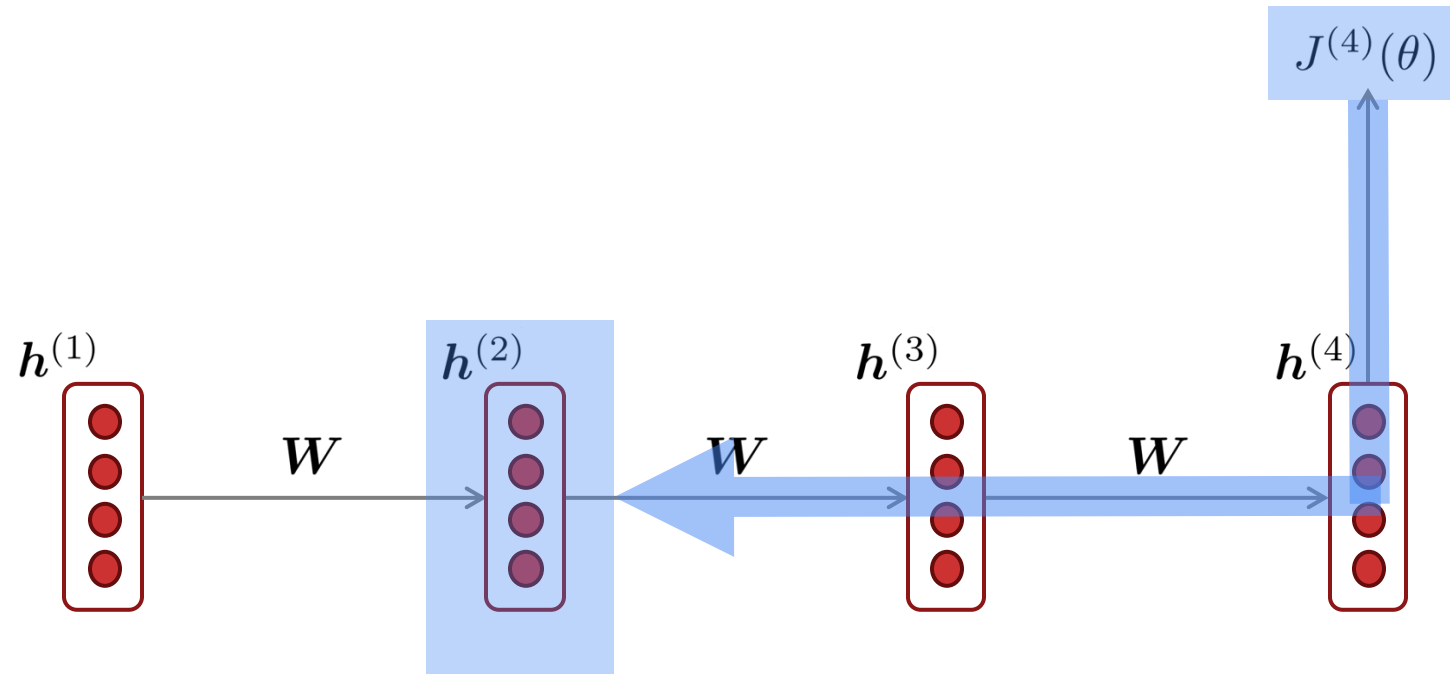
### 3. Problems with RNNs: Vanishing and Exploding Gradients



# Vanishing gradient intuition



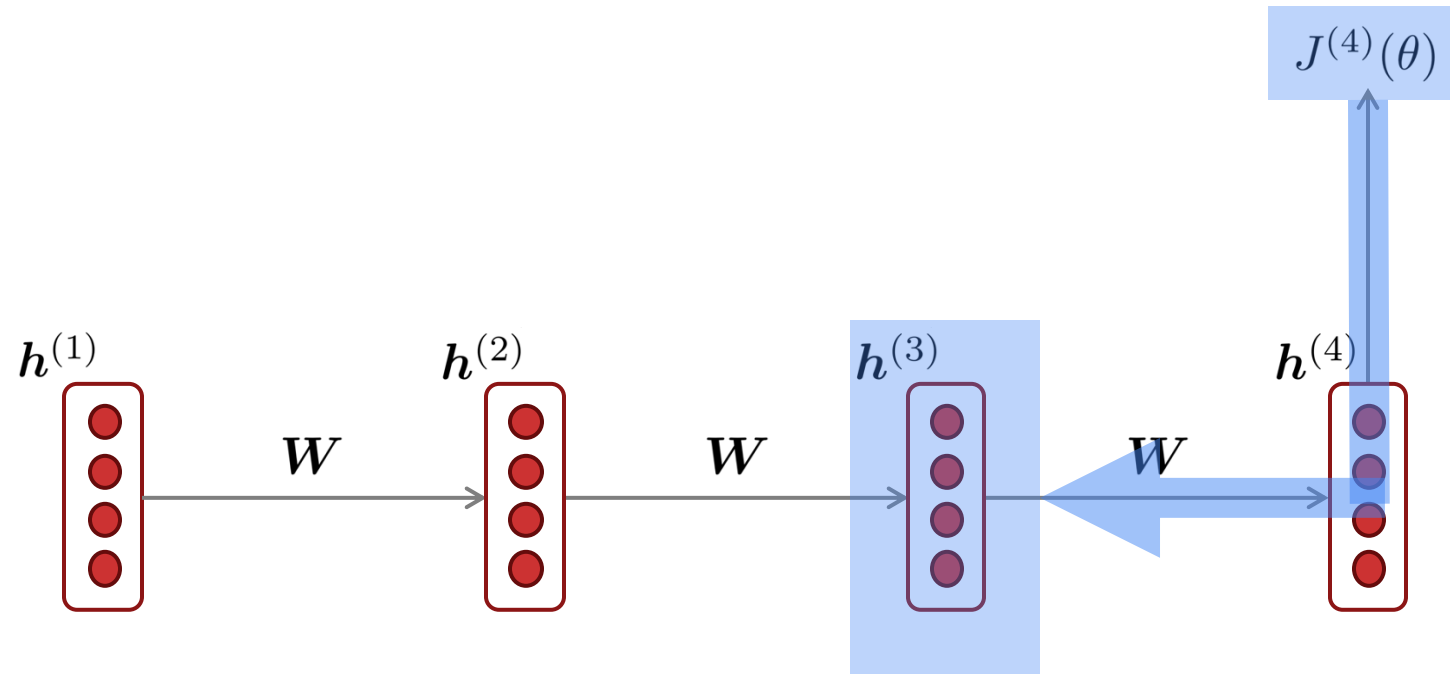
# Vanishing gradient intuition



$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial J^{(4)}}{\partial h^{(2)}}$$

chain rule!

# Vanishing gradient intuition

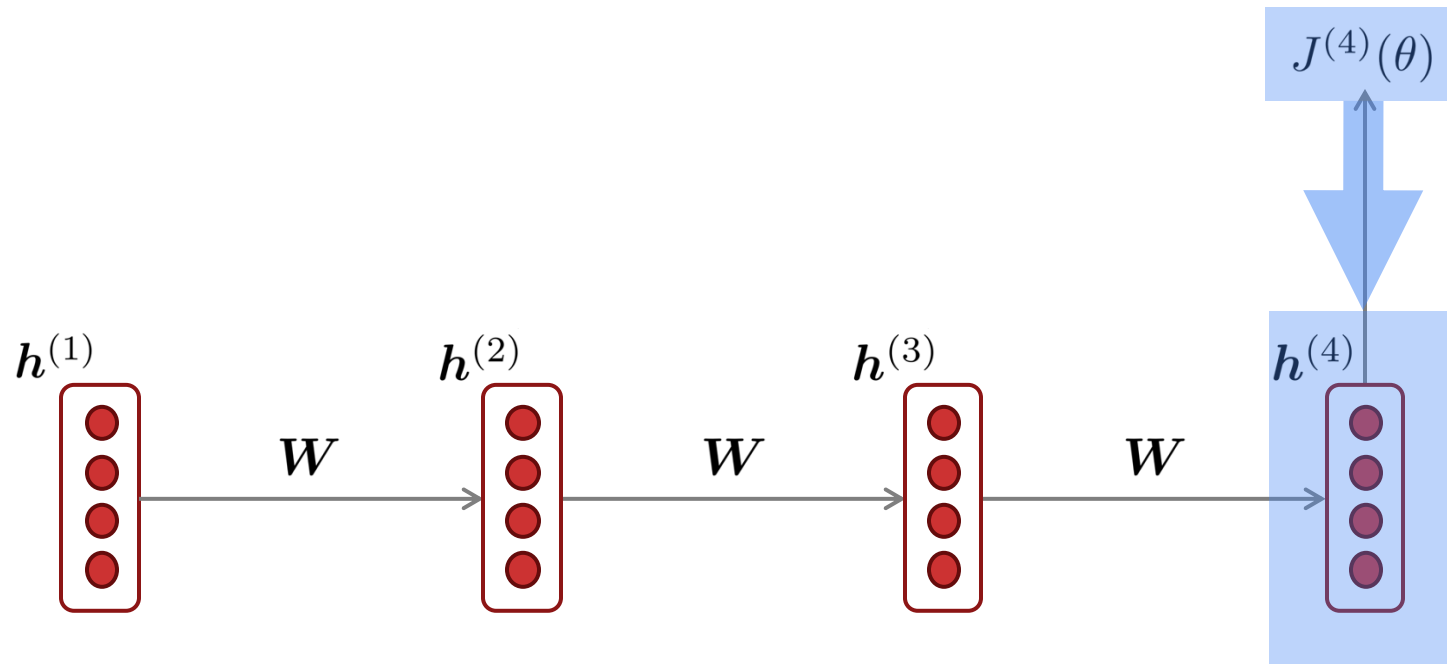


$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times$$

$$\frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial J^{(4)}}{\partial h^{(3)}}$$

chain rule!

# Vanishing gradient intuition



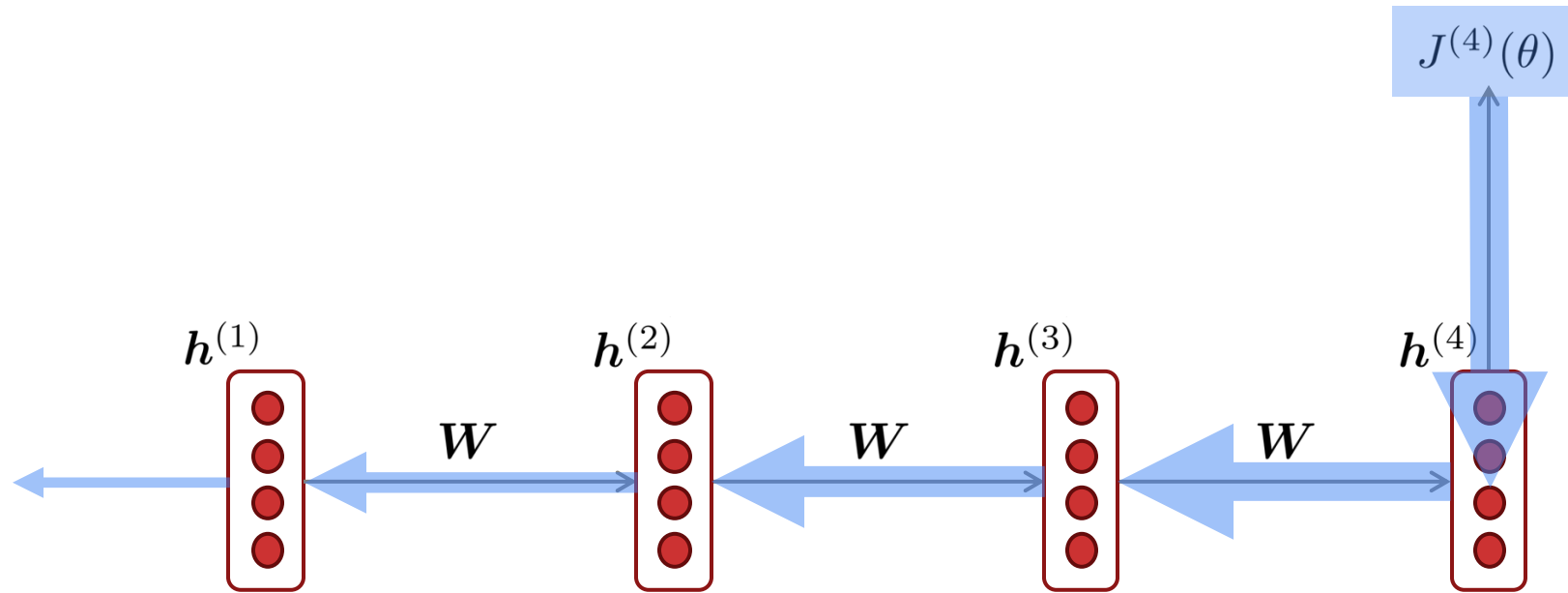
$$\frac{\partial J^{(4)}}{\partial \mathbf{h}^{(1)}} = \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{h}^{(1)}} \times$$

$$\frac{\partial \mathbf{h}^{(3)}}{\partial \mathbf{h}^{(2)}} \times$$

$$\frac{\partial \mathbf{h}^{(4)}}{\partial \mathbf{h}^{(3)}} \times \frac{\partial J^{(4)}}{\partial \mathbf{h}^{(4)}}$$

chain rule!

# Vanishing gradient intuition



$$\frac{\partial J^{(4)}}{\partial \mathbf{h}^{(1)}} = \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{h}^{(1)}} \times \frac{\partial \mathbf{h}^{(3)}}{\partial \mathbf{h}^{(2)}} \times \frac{\partial \mathbf{h}^{(4)}}{\partial \mathbf{h}^{(3)}} \times \frac{\partial J^{(4)}}{\partial \mathbf{h}^{(4)}}$$

What happens if these are small?

**Vanishing gradient problem:**  
When these are small, the gradient signal gets smaller and smaller as it backpropagates further


# Vanishing gradient proof sketch (linear case)

- Recall: 
$$\mathbf{h}^{(t)} = \sigma \left( \mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_x \mathbf{x}^{(t)} + \mathbf{b}_1 \right)$$

- What if  $\sigma$  were the identity function,  $\sigma(x) = x$  ?

$$\begin{aligned} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} &= \text{diag} \left( \sigma' \left( \mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_x \mathbf{x}^{(t)} + \mathbf{b}_1 \right) \right) \mathbf{W}_h && \text{(chain rule)} \\ &= \mathbf{I} \mathbf{W}_h = \mathbf{W}_h \end{aligned}$$

- Consider the gradient of the loss  $J^{(i)}(\theta)$  on step  $i$ , with respect to the hidden state  $\mathbf{h}^{(j)}$  on some previous step  $j$ . Let  $\ell = i - j$

$$\begin{aligned} \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(j)}} &= \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(i)}} \prod_{j < t \leq i} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} && \text{(chain rule)} \\ &= \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(i)}} \prod_{j < t \leq i} \mathbf{W}_h = \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(i)}} \boxed{\mathbf{W}_h^\ell} && \text{(value of } \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} \text{)} \end{aligned}$$


If  $\mathbf{W}_h$  is “small”, then this term gets exponentially problematic as  $\ell$  becomes large

# Vanishing gradient proof sketch (linear case)

- What's wrong with  $W_h^\ell$  ?
- Consider if the eigenvalues of  $W_h$  are all less than 1: sufficient but not necessary  
 $\lambda_1, \lambda_2, \dots, \lambda_n < 1$   
 $\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_n$  (eigenvectors)
- We can write  $\frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(i)}} W_h^\ell$  using the eigenvectors of  $W_h$  as a basis:

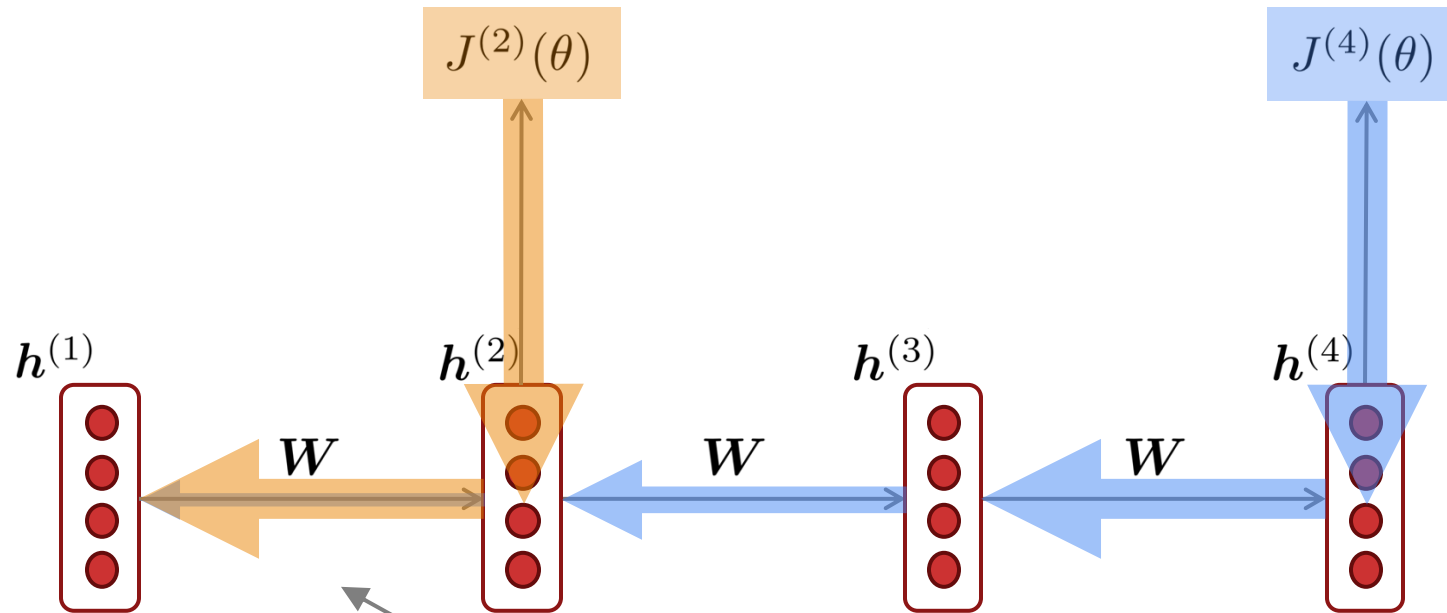
$$\frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(i)}} W_h^\ell = \sum_{i=1}^n c_i \lambda_i^\ell \mathbf{q}_i \approx \mathbf{0} \text{ (for large } \ell \text{)}$$

Approaches 0 as  $\ell$  grows, so gradient vanishes

- What about nonlinear activations  $\sigma$  (i.e., what we use?)
  - Pretty much the same thing, except the proof requires  $\lambda_i < \gamma$  for some  $\gamma$  dependent on dimensionality and  $\sigma$



# Why is vanishing gradient a problem?



Gradient signal from far away is lost because it's much smaller than gradient signal from close-by.

So, model weights are updated only with respect to near effects, not long-term effects.

# Effect of vanishing gradient on RNN-LM

- **LM task:** *When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her \_\_\_\_\_*
- To learn from this training example, the RNN-LM needs to **model the dependency** between “*tickets*” on the 7<sup>th</sup> step and the target word “*tickets*” at the end.
- But if the gradient is small, the model **can't learn this dependency**
  - So, the model is **unable to predict similar long-distance dependencies** at test time

# Why is exploding gradient a problem?

- If the gradient becomes too big, then the SGD update step becomes too big:

$$\theta^{new} = \theta^{old} - \overbrace{\alpha}^{\text{learning rate}} \underbrace{\nabla_{\theta} J(\theta)}_{\text{gradient}}$$

- This can cause **bad updates**: we take too large a step and reach a weird and bad parameter configuration (with large loss)
  - You think you've found a hill to climb, but suddenly you're in Iowa
- In the worst case, this will result in **Inf** or **NaN** in your network (then you have to restart training from an earlier checkpoint)

# Gradient clipping: solution for exploding gradient

- **Gradient clipping**: if the norm of the gradient is greater than some threshold, scale it down before applying SGD update

---

## Algorithm 1 Pseudo-code for norm clipping

---

```
 $\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$   
if  $\|\hat{\mathbf{g}}\| \geq \textit{threshold}$  then  
     $\hat{\mathbf{g}} \leftarrow \frac{\textit{threshold}}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$   
end if
```

---

- **Intuition**: take a step in the same direction, but a smaller step
- In practice, **remembering to clip gradients is important**, but exploding gradients are an easy problem to solve

# How to fix the vanishing gradient problem?

- The main problem is that *it's too difficult for the RNN to learn to preserve information over many timesteps.*

- In a vanilla RNN, the hidden state is constantly being **rewritten**

$$\mathbf{h}^{(t)} = \sigma \left( \mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_x \mathbf{x}^{(t)} + \mathbf{b} \right)$$

- First off next time: How about an RNN with separate **memory** which is added to?
  - LSTMs
- And then: Creating more direct and linear pass-through connections in model
  - Attention, residual connections, etc.

## 4. Recap

- **Language Model**: A system that predicts the next word
- **Recurrent Neural Network**: A family of neural networks that:
  - Take sequential input of any length
  - Apply the same weights on each step
  - Can optionally produce output on each step
- **Recurrent Neural Network  $\neq$  Language Model**
- We've shown that RNNs are a great way to build a LM (despite some problems)
- RNNs – still relevant today with the rise of *state space models*