Natural Language Processing with Deep Learning
CS224N/Ling284

Lecture 6:
Language Models and
Recurrent Neural Networks

Abigail See

Overview

Today we will:

- Introduce a new NLP task
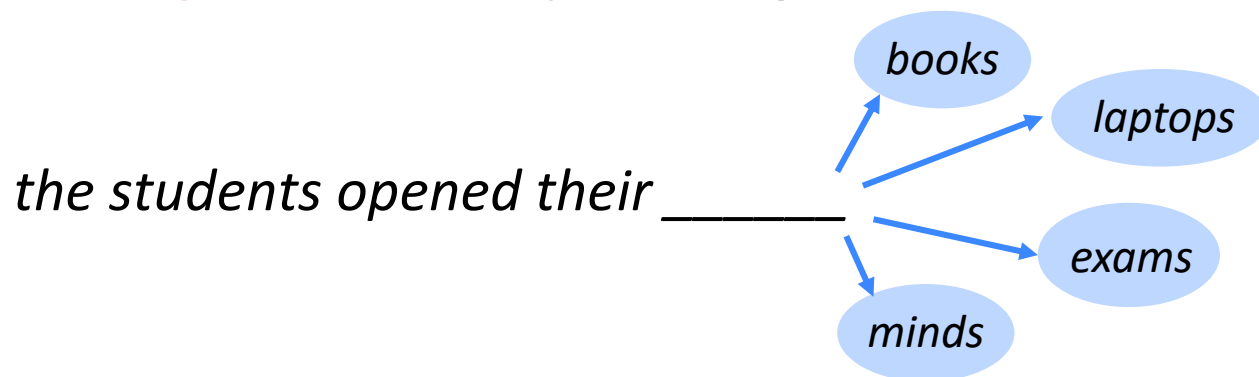 - **Language Modeling**

motivates

- Introduce a new family of neural networks
  - **Recurrent Neural Networks (RNNs)**

These are two of the most important ideas for the rest of the class!

Language Modeling

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

the students opened their _____

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

$$P(\boldsymbol{x}^{(t+1)} |\ \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

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

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

3

Language Modeling

- You can also think of a Language Model as a system that assigns probability to a piece of text.

- For example, if we have some text $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$, then the probability of this text (according to the Language Model) is:
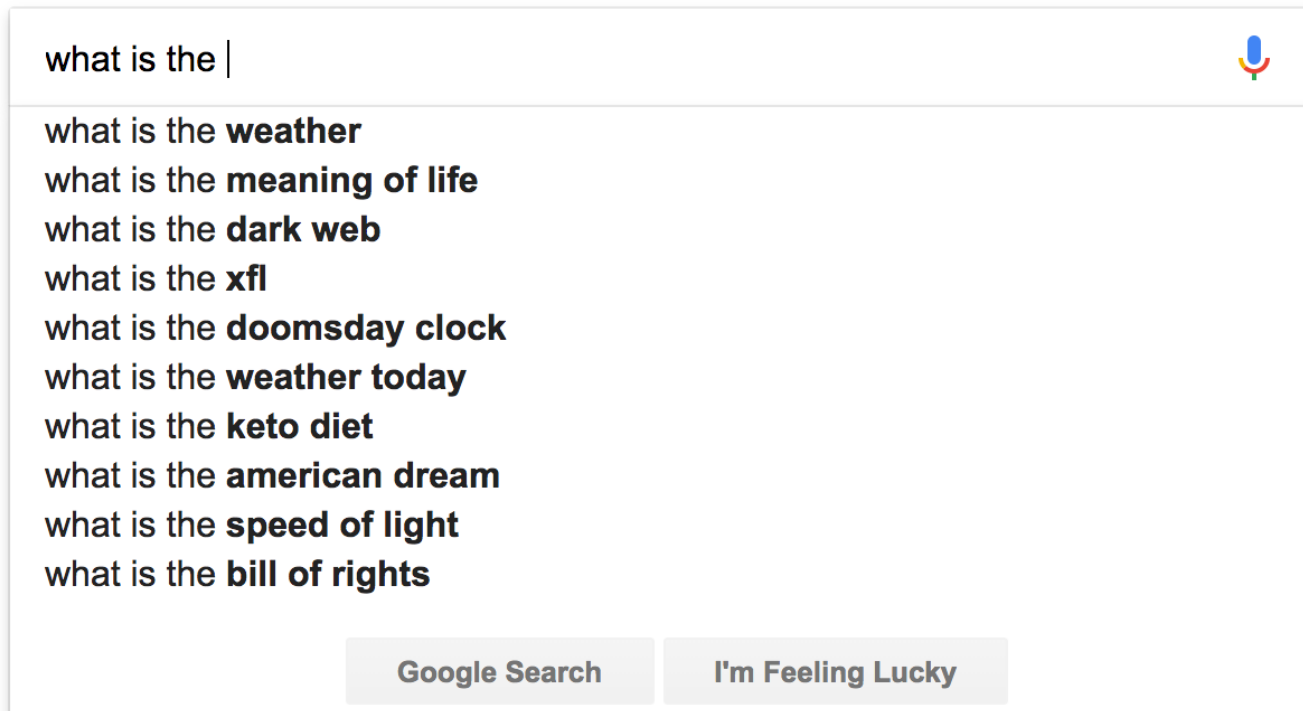
$$P(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}) = P(\boldsymbol{x}^{(1)}) \times P(\boldsymbol{x}^{(2)} | \ \boldsymbol{x}^{(1)}) \times \cdots \times P(\boldsymbol{x}^{(T)} | \ \boldsymbol{x}^{(T-1)}, \ldots, \boldsymbol{x}^{(1)})$$

$$= \prod_{t=1}^{T} P(\boldsymbol{x}^{(t)} | \ \boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(1)})$$

This is what our LM provides

You use Language Models every day!
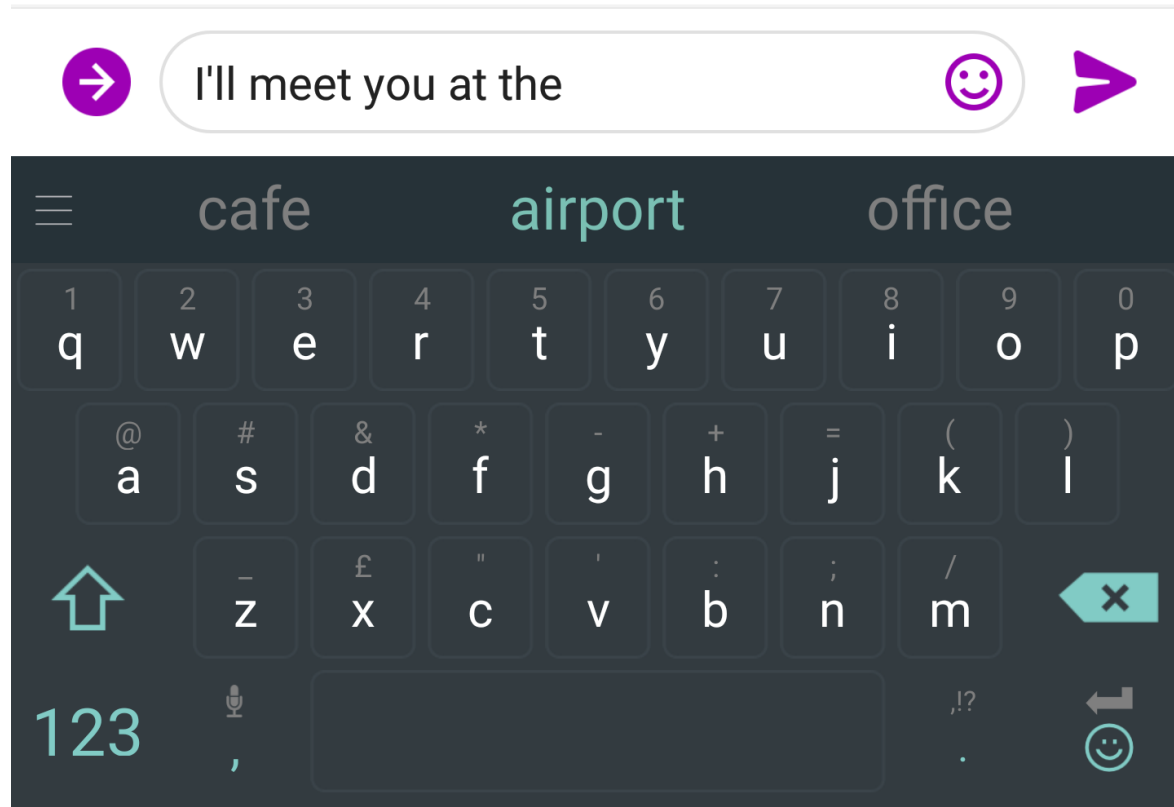
You use Language Models every day!

n-gram Language Models

the students opened their _____

- **Question**: How to learn a Language Model?
- **Answer** (pre- Deep Learning): learn a *n*-gram Language Model!

- Definition: A *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"
  - 4-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 simplifying assumption: $\boldsymbol{x}^{(t+1)}$ depends only on the preceding *n-1* words.

n-1 words

$$P(\boldsymbol{x}^{(t+1)}|\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(1)}) = P(\boldsymbol{x}^{(t+1)}|\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)}) \qquad \text{(assumption)}$$

prob of a n-gram

prob of a (n-1)-gram

$$= \frac{P(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{P(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})} \qquad \begin{array}{l}\text{(definition of} \\ \text{conditional prob)}\end{array}$$

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

$$\approx \frac{\text{count}(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{\text{count}(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})} \qquad \begin{array}{l}\text{(statistical} \\ \text{approximation)}\end{array}$$

8

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

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

For example, suppose that in the corpus:

- "students opened their" occurred 1000 times
- "students opened their books" occurred 400 times
  - → P(books | students opened their) = 0.4
- "students opened their exams" occurred 100 times
  - → P(exams | students opened their) = 0.1

Should we have discarded the "proctor" context?

9

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 δ to the count for every $w \in V$. This is called *smoothing*.

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

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.

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

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
…	

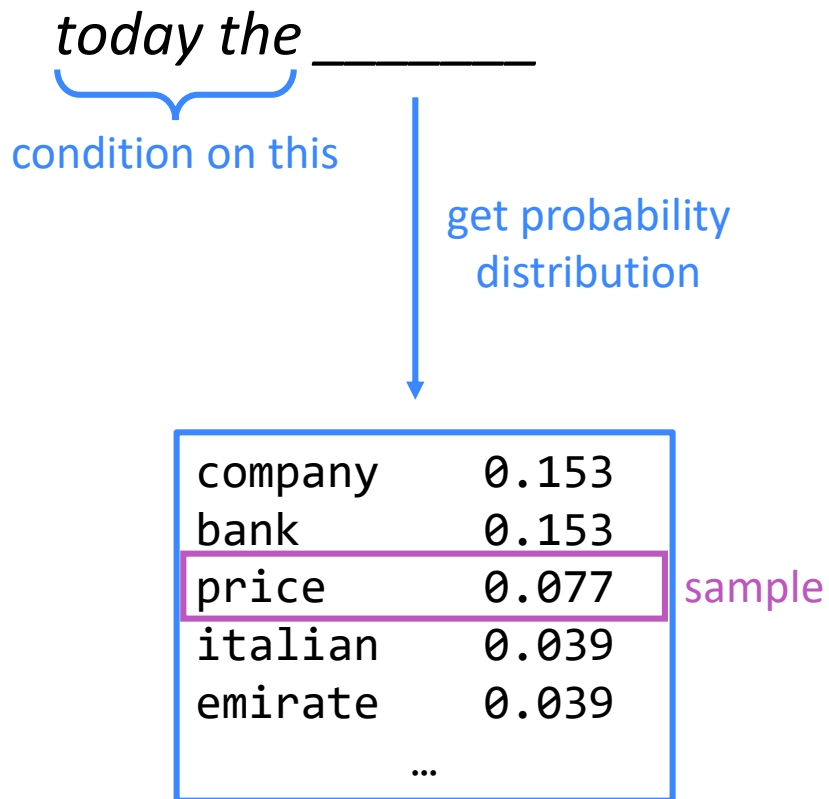**Sparsity problem**:
not much granularity
in the probability
distribution
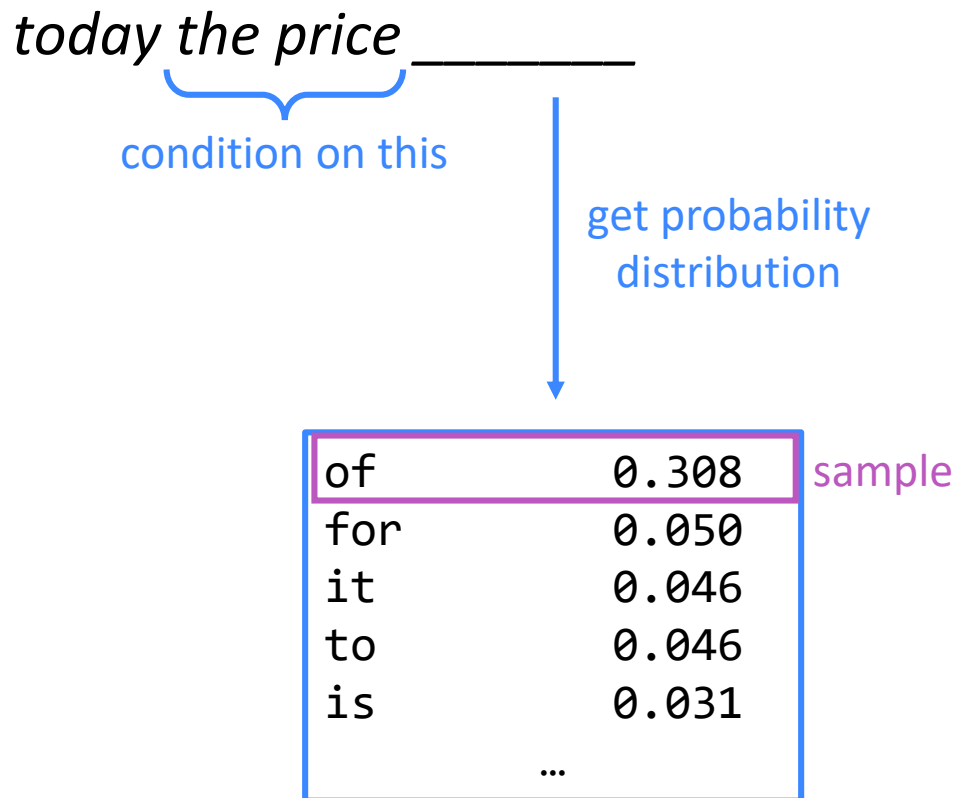
Otherwise, seems reasonable!

* **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.

today the _____

condition on this

get probability
distribution

company	0.153
bank	0.153
price	0.077
italian	0.039
emirate	0.039
	…

sample

Generating text with a n-gram Language Model

- You can also use a Language Model to generate text.

today the price _____

condition on this

get probability
distribution

of	0.308
for	0.050
it	0.046
to	0.046
is	0.031
…	

sample

Generating text with a n-gram Language Model

- You can also use a Language Model to generate text.

today the price of _____

condition on this

get probability
distribution

the	0.072
18	0.043
oil	0.043
its	0.036
gold	0.018
...	

sample
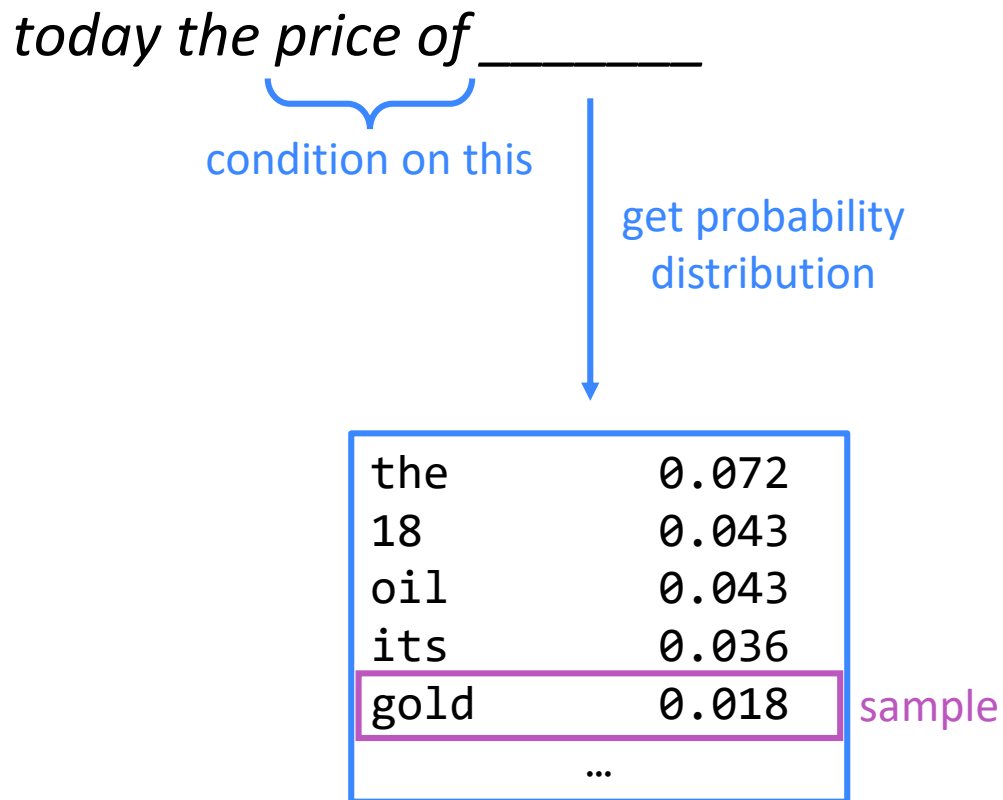
Generating text with a n-gram Language Model

- You can also use a Language Model to generate text.

today the price of gold _____

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…

How to build a *neural* Language Model?

- Recall the Language Modeling task:
  - Input: sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$
  - Output: prob dist of the next word $P(\boldsymbol{x}^{(t+1)} | \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$

- How about a window-based neural model?
  - We saw this applied to Named Entity Recognition in Lecture 3:

LOCATION

U

W

museums   *in*   *Paris*   *are*   *amazing*

18
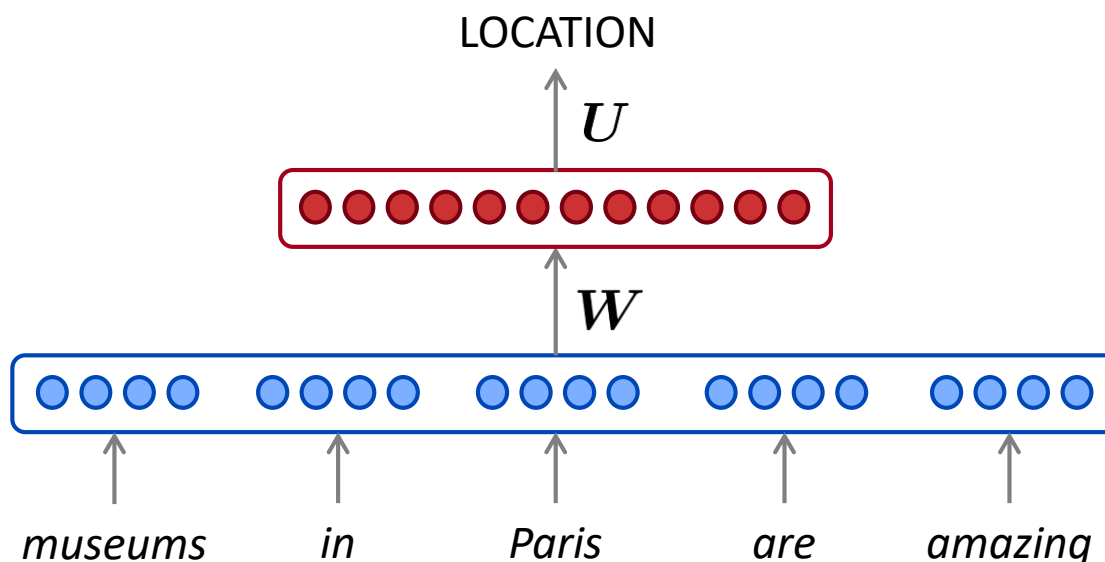
A fixed-window neural Language Model

as   the   proctor   started   the   clock   *the*   *students*   *opened*   *their*   _____

discard

fixed window

A fixed-window neural Language Model

output distribution

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

hidden layer

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

concatenated word embeddings

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

words / one-hot vectors
$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$

books

laptops

a   zoo

$\boldsymbol{U}$

$\boldsymbol{W}$

the
$\boldsymbol{x}^{(1)}$

students
$\boldsymbol{x}^{(2)}$

opened
$\boldsymbol{x}^{(3)}$

their
$\boldsymbol{x}^{(4)}$

20
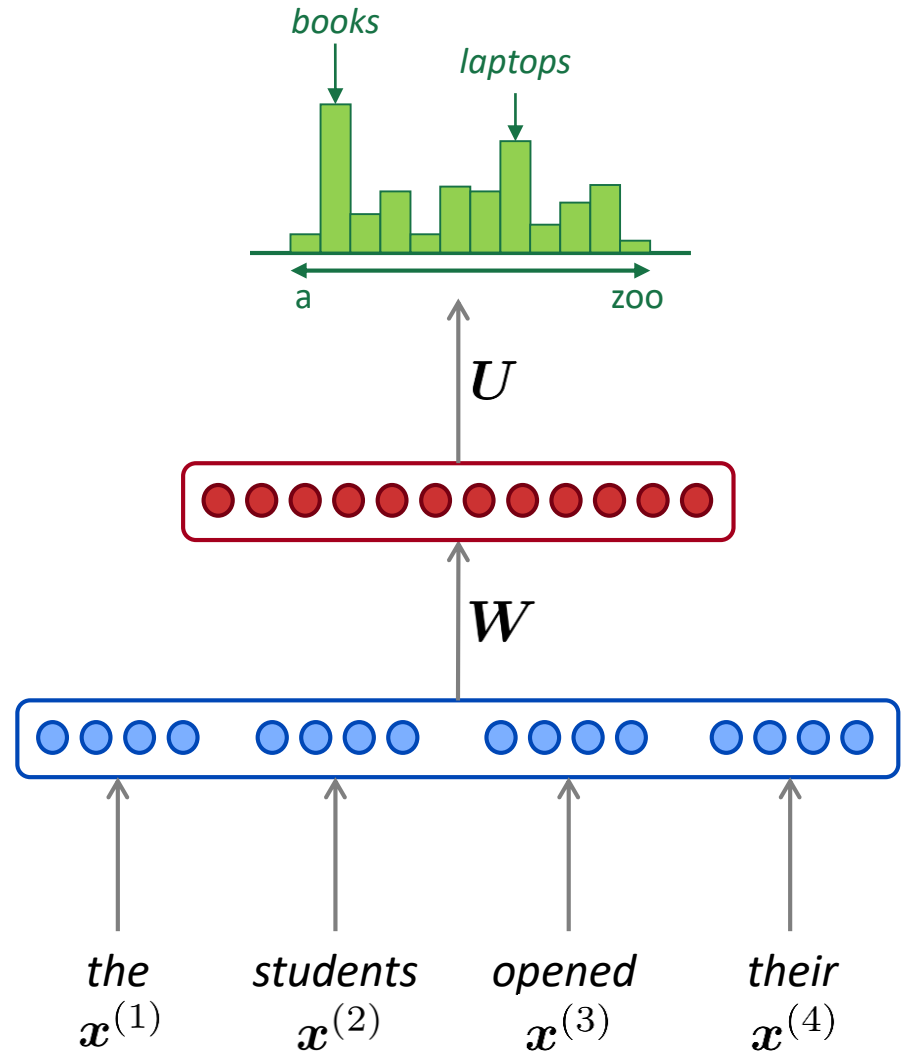
# A fixed-window neural Language Model

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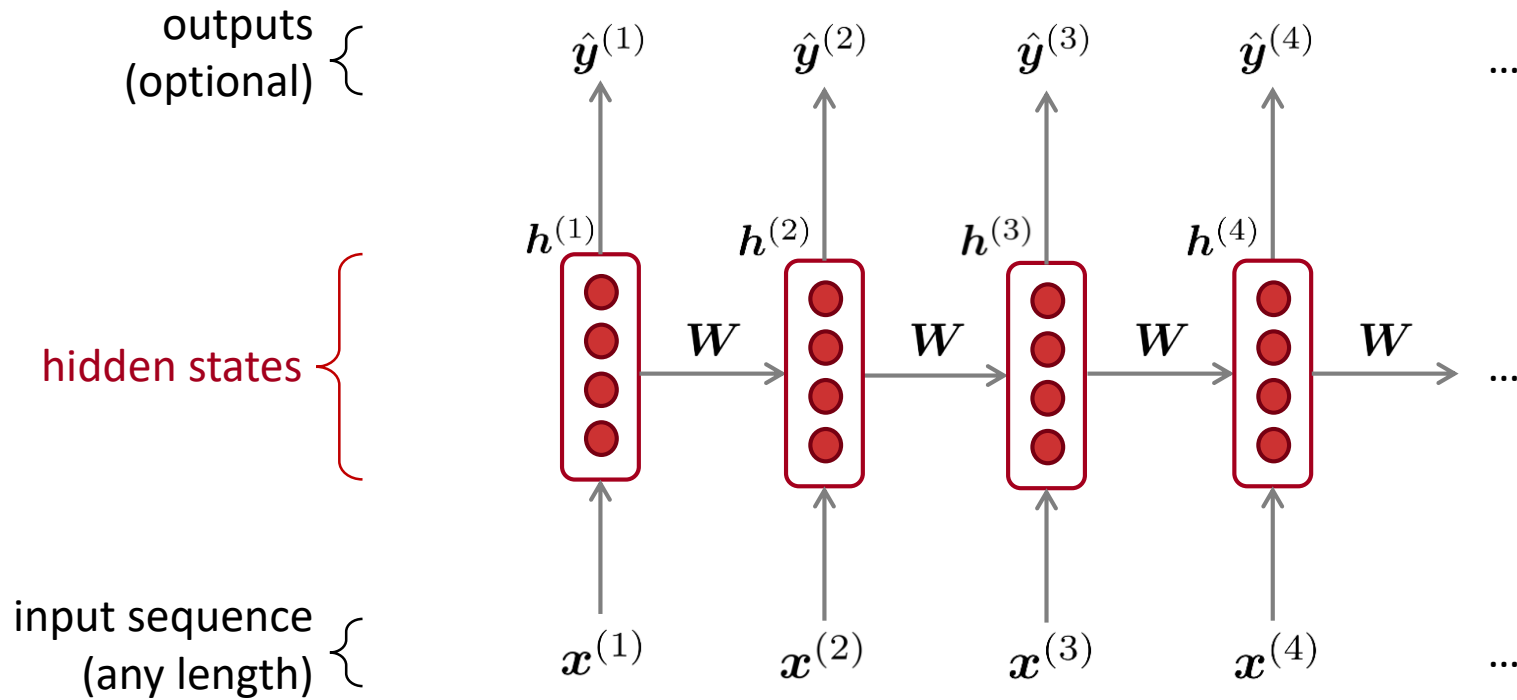
We need a neural architecture that can process *any length input*

books

laptops

a   zoo

U

W

the $x^{(1)}$ *students* $x^{(2)}$ *opened* $x^{(3)}$ *their* $x^{(4)}$
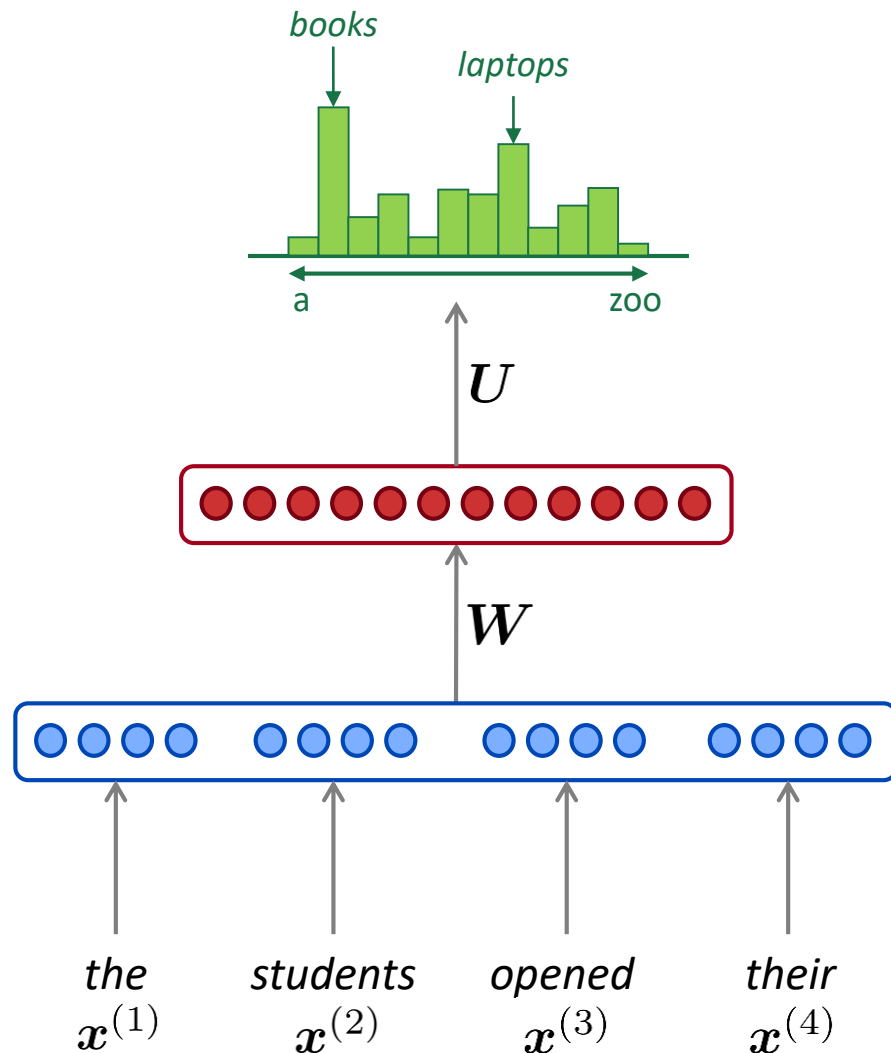
Recurrent Neural Networks (RNN)

A family of neural architectures

Core idea: Apply the same weights W *repeatedly*

outputs
(optional)

$\hat{y}^{(1)}$ $\hat{y}^{(2)}$ $\hat{y}^{(3)}$ $\hat{y}^{(4)}$ ...

$h^{(1)}$ $h^{(2)}$ $h^{(3)}$ $h^{(4)}$

hidden states

W W W W ...

input sequence
(any length)

$x^{(1)}$ $x^{(2)}$ $x^{(3)}$ $x^{(4)}$ ...

A RNN Language Model

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

books

laptops

a   zoo

U

output distribution

$$\hat{y}^{(t)} = \text{softmax}\left(U h^{(t)} + b_2\right) \in \mathbb{R}^{|V|}$$

$h^{(0)}$ $h^{(1)}$ $h^{(2)}$ $h^{(3)}$ $h^{(4)}$
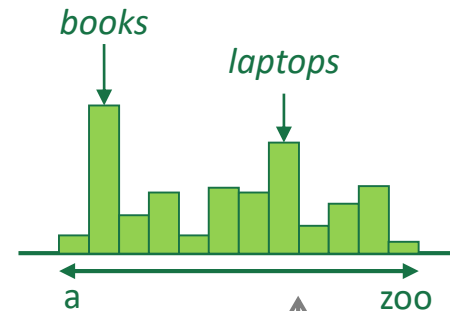
hidden states

$$h^{(t)} = \sigma\left(W_h h^{(t-1)} + W_e e^{(t)} + b_1\right)$$

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

W_h W_h W_h W_h

W_e W_e W_e W_e

word embeddings

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

$e^{(1)}$ $e^{(2)}$ $e^{(3)}$ $e^{(4)}$

E E E E

words / one-hot vectors

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

the   *students*   *opened*   *their*
$x^{(1)}$ $x^{(2)}$ $x^{(3)}$ $x^{(4)}$
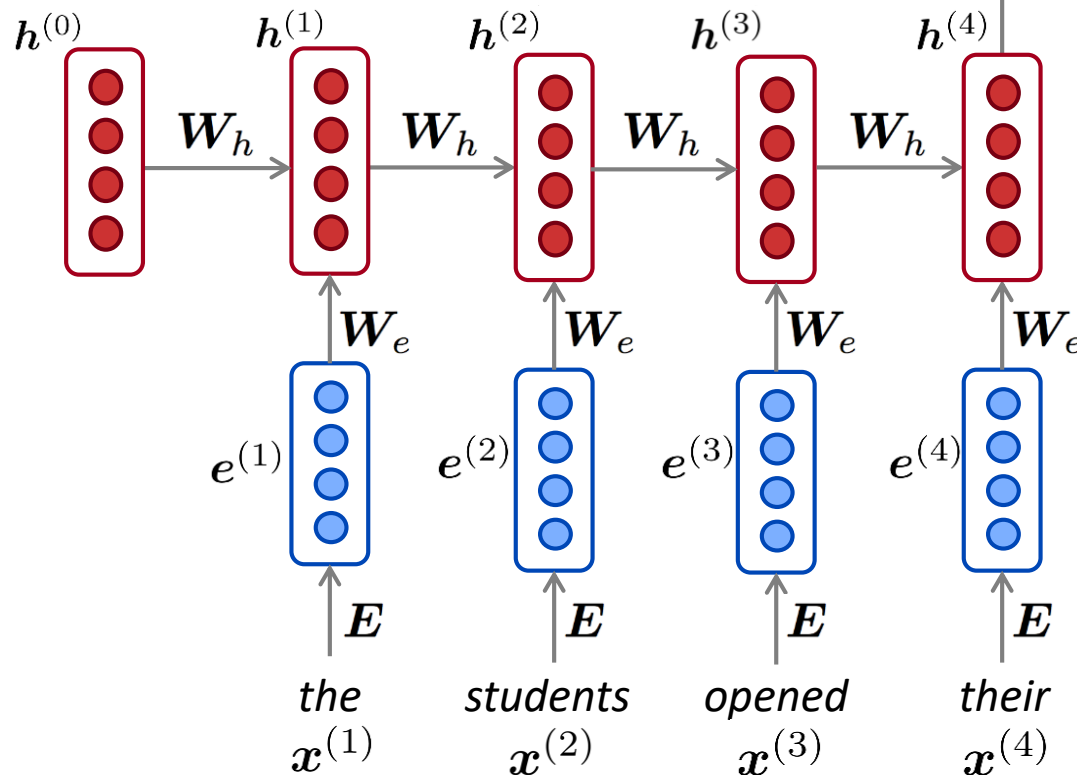
__Note__: this input sequence could be much longer, but this slide doesn't have space!
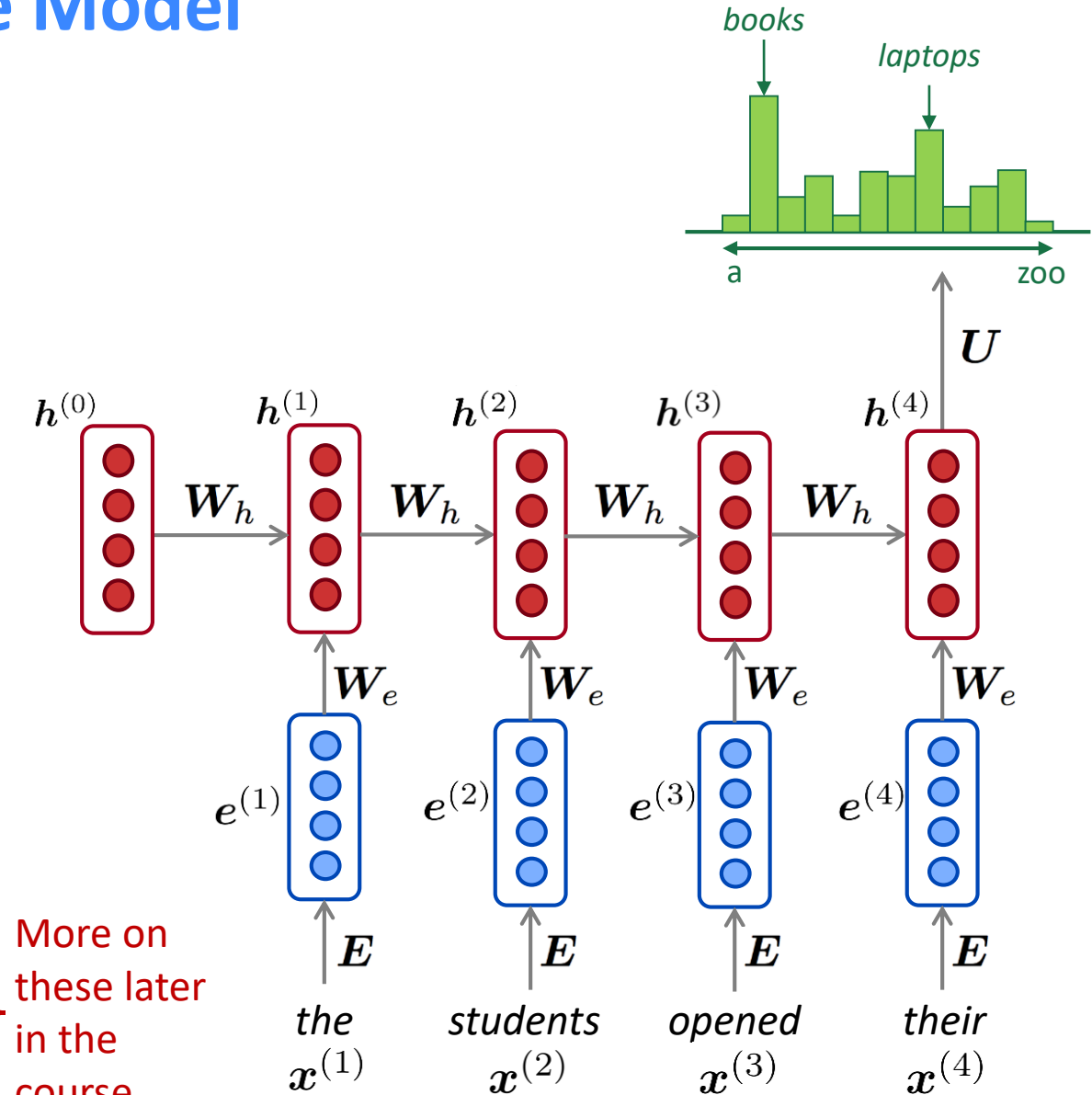
A RNN Language Model

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

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
- 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

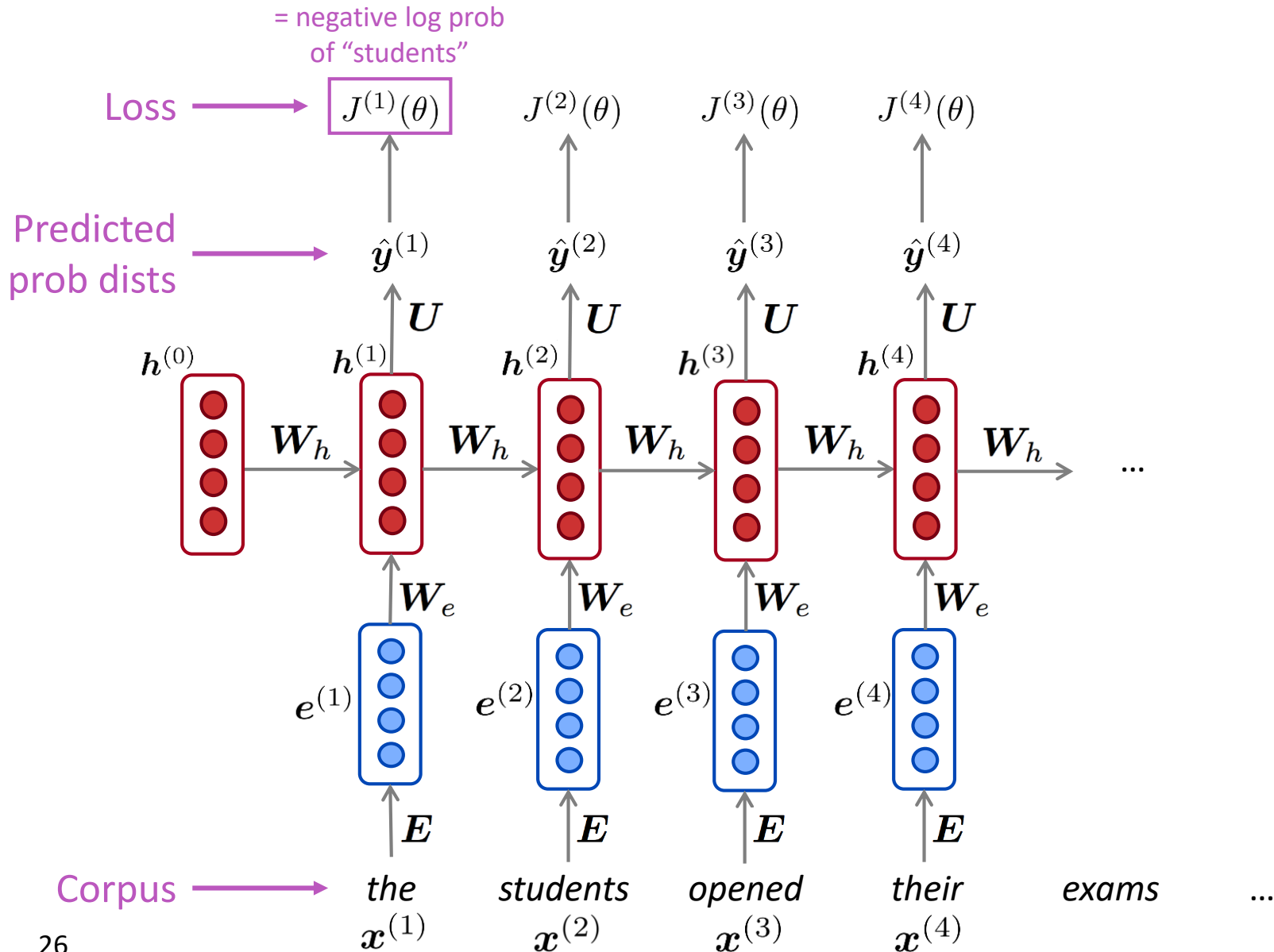More on these later in the course

24

Training a RNN Language Model

- Get a big corpus of text which is a sequence of words $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$
- Feed into RNN-LM; compute output distribution $\hat{\boldsymbol{y}}^{(t)}$ 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 $\hat{\boldsymbol{y}}^{(t)}$, and the true next word $\boldsymbol{y}^{(t)}$ (one-hot for $\boldsymbol{x}^{(t+1)}$):

$$J^{(t)}(\theta) = CE(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) = -\sum_{w \in V} \boldsymbol{y}_w^{(t)} \log \hat{\boldsymbol{y}}_w^{(t)} = -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$
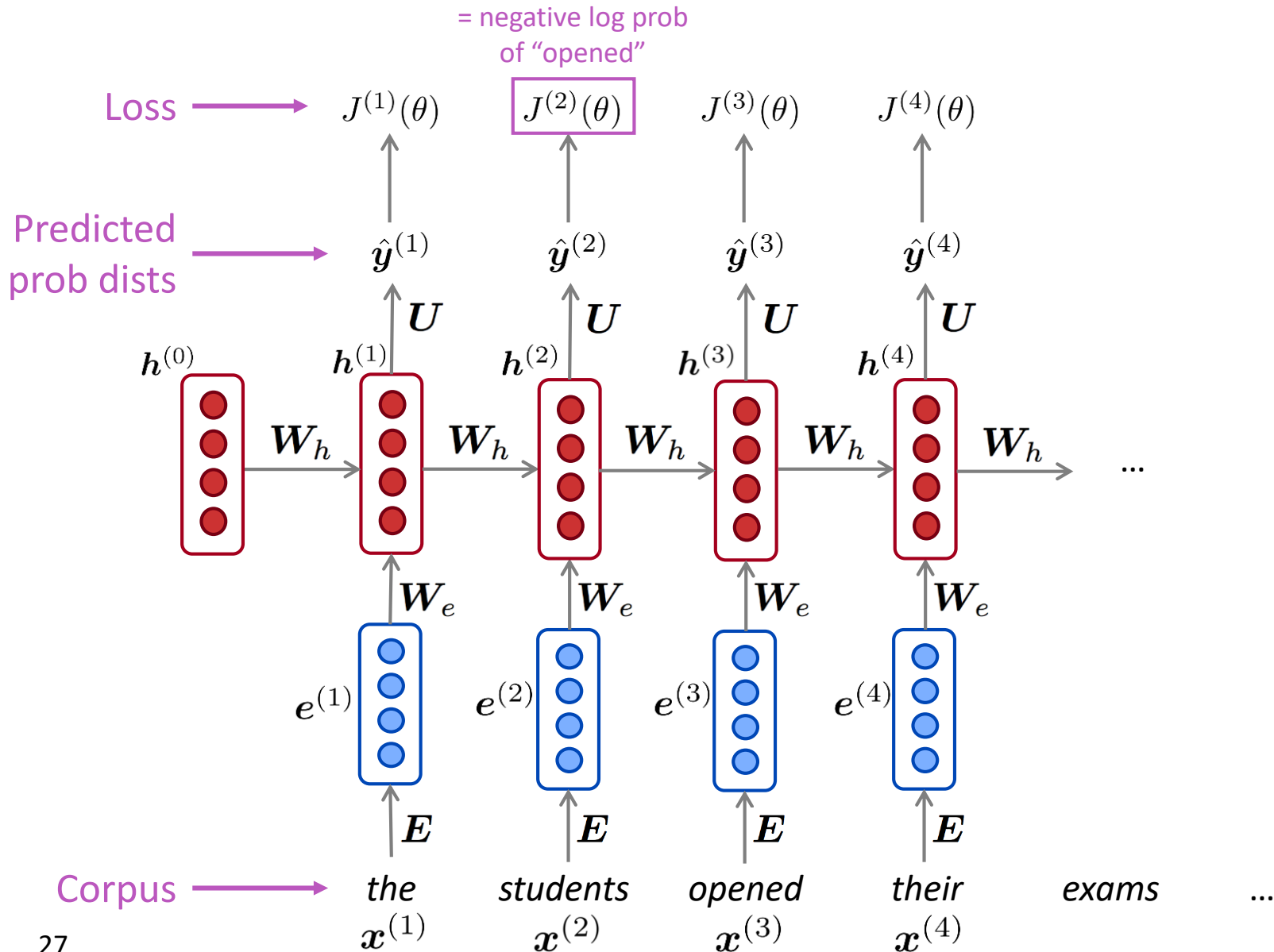
- Average this to get overall loss for entire training set:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$
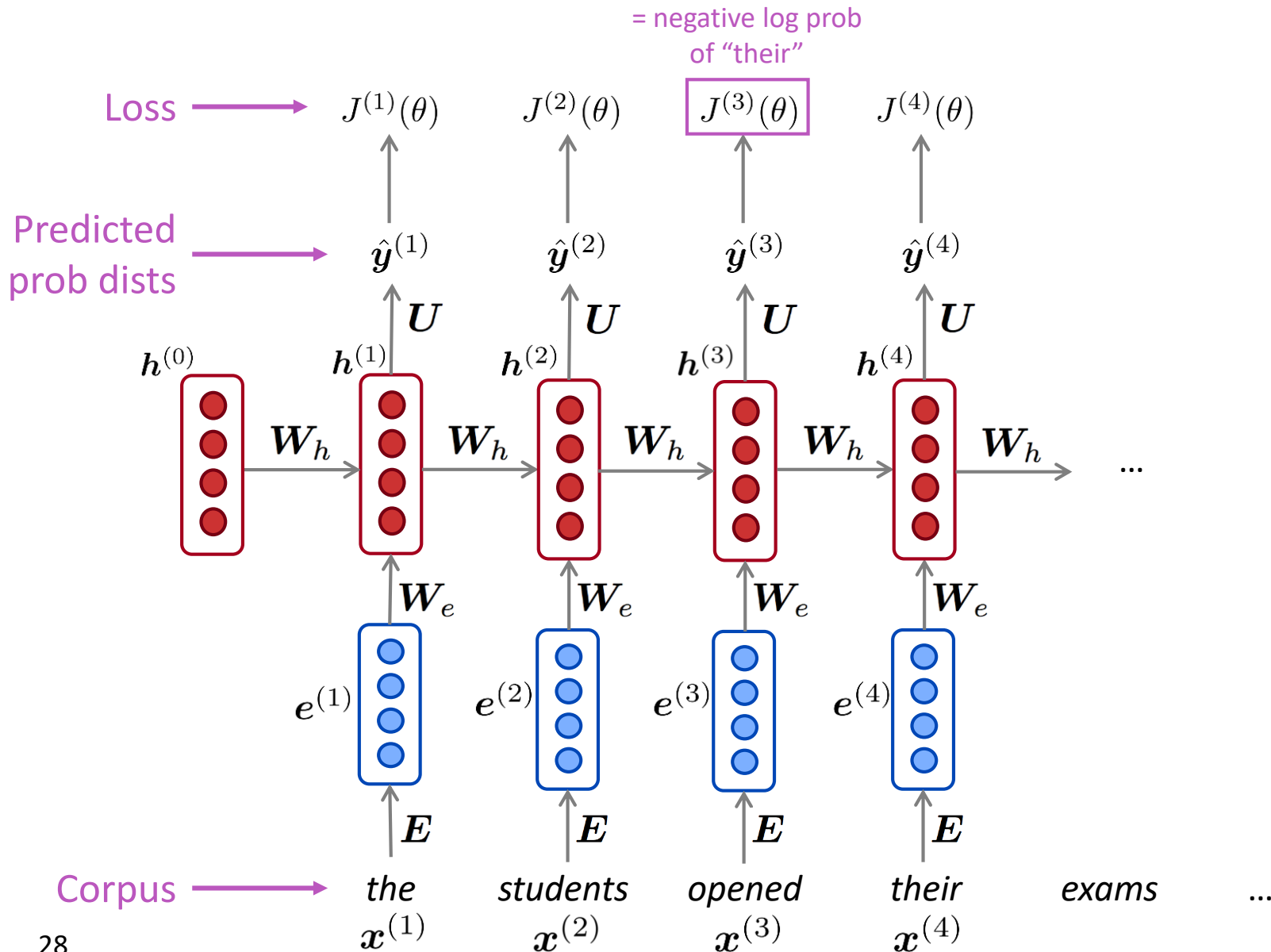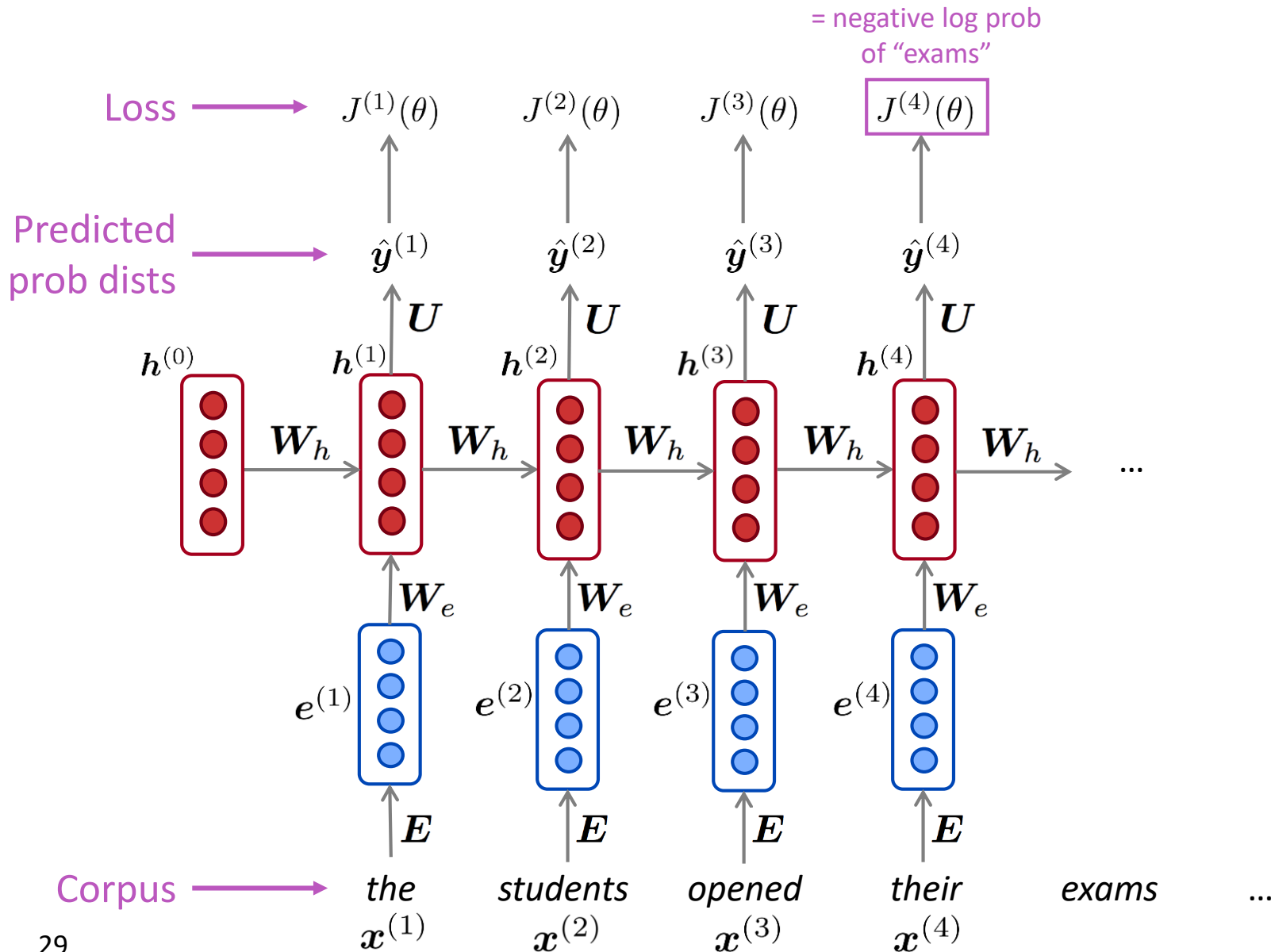
Training a RNN Language Model

= negative log prob
of "students"

Loss $\longrightarrow$ $J^{(1)}(\theta)$ $\qquad$ $J^{(2)}(\theta)$ $\qquad$ $J^{(3)}(\theta)$ $\qquad$ $J^{(4)}(\theta)$

Predicted
prob dists $\longrightarrow$ $\hat{\boldsymbol{y}}^{(1)}$ $\qquad$ $\hat{\boldsymbol{y}}^{(2)}$ $\qquad$ $\hat{\boldsymbol{y}}^{(3)}$ $\qquad$ $\hat{\boldsymbol{y}}^{(4)}$

$\boldsymbol{U}$ $\qquad$ $\boldsymbol{U}$ $\qquad$ $\boldsymbol{U}$ $\qquad$ $\boldsymbol{U}$

$\boldsymbol{h}^{(0)}$ $\quad$ $\boldsymbol{h}^{(1)}$ $\quad$ $\boldsymbol{h}^{(2)}$ $\quad$ $\boldsymbol{h}^{(3)}$ $\quad$ $\boldsymbol{h}^{(4)}$

$\boldsymbol{W}_h$ $\qquad$ $\boldsymbol{W}_h$ $\qquad$ $\boldsymbol{W}_h$ $\qquad$ $\boldsymbol{W}_h$ $\qquad$ $\boldsymbol{W}_h$ ...

$\boldsymbol{W}_e$ $\qquad$ $\boldsymbol{W}_e$ $\qquad$ $\boldsymbol{W}_e$ $\qquad$ $\boldsymbol{W}_e$

$\boldsymbol{e}^{(1)}$ $\qquad$ $\boldsymbol{e}^{(2)}$ $\qquad$ $\boldsymbol{e}^{(3)}$ $\qquad$ $\boldsymbol{e}^{(4)}$

$\boldsymbol{E}$ $\qquad$ $\boldsymbol{E}$ $\qquad$ $\boldsymbol{E}$ $\qquad$ $\boldsymbol{E}$

Corpus $\longrightarrow$ *the* $\qquad$ *students* $\qquad$ *opened* $\qquad$ *their* $\qquad$ *exams* $\qquad$ ...

$\boldsymbol{x}^{(1)}$ $\qquad$ $\boldsymbol{x}^{(2)}$ $\qquad$ $\boldsymbol{x}^{(3)}$ $\qquad$ $\boldsymbol{x}^{(4)}$

Training a RNN Language Model

= negative log prob
of "opened"

Loss $\longrightarrow$ $J^{(1)}(\theta)$ $\boxed{J^{(2)}(\theta)}$ $J^{(3)}(\theta)$ $J^{(4)}(\theta)$

Predicted $\longrightarrow$ $\hat{y}^{(1)}$ $\hat{y}^{(2)}$ $\hat{y}^{(3)}$ $\hat{y}^{(4)}$
prob dists

U U U U

$h^{(0)}$ $h^{(1)}$ $h^{(2)}$ $h^{(3)}$ $h^{(4)}$

W_h W_h W_h W_h W_h ...

W_e W_e W_e W_e

$e^{(1)}$ $e^{(2)}$ $e^{(3)}$ $e^{(4)}$

E E E E

Corpus $\longrightarrow$ *the* *students* *opened* *their* *exams* ...
$x^{(1)}$ $x^{(2)}$ $x^{(3)}$ $x^{(4)}$

27

Training a RNN Language Model

= negative log prob of "their"

Loss ⟶ $J^{(1)}(\theta)$ $J^{(2)}(\theta)$ $\boxed{J^{(3)}(\theta)}$ $J^{(4)}(\theta)$

Predicted prob dists ⟶ $\hat{\boldsymbol{y}}^{(1)}$ $\hat{\boldsymbol{y}}^{(2)}$ $\hat{\boldsymbol{y}}^{(3)}$ $\hat{\boldsymbol{y}}^{(4)}$

$\boldsymbol{U}$ $\boldsymbol{U}$ $\boldsymbol{U}$ $\boldsymbol{U}$

$\boldsymbol{h}^{(0)}$ $\boldsymbol{h}^{(1)}$ $\boldsymbol{h}^{(2)}$ $\boldsymbol{h}^{(3)}$ $\boldsymbol{h}^{(4)}$

$\boldsymbol{W}_h$ $\boldsymbol{W}_h$ $\boldsymbol{W}_h$ $\boldsymbol{W}_h$ $\boldsymbol{W}_h$ ...

$\boldsymbol{W}_e$ $\boldsymbol{W}_e$ $\boldsymbol{W}_e$ $\boldsymbol{W}_e$

$\boldsymbol{e}^{(1)}$ $\boldsymbol{e}^{(2)}$ $\boldsymbol{e}^{(3)}$ $\boldsymbol{e}^{(4)}$

$\boldsymbol{E}$ $\boldsymbol{E}$ $\boldsymbol{E}$ $\boldsymbol{E}$

Corpus ⟶   *the*   *students*   *opened*   *their*   *exams*   ...

28   $\boldsymbol{x}^{(1)}$ $\boldsymbol{x}^{(2)}$ $\boldsymbol{x}^{(3)}$ $\boldsymbol{x}^{(4)}$

Training a RNN Language Model

= negative log prob of "exams"

Loss $\longrightarrow$ $J^{(1)}(\theta)$ $J^{(2)}(\theta)$ $J^{(3)}(\theta)$ $J^{(4)}(\theta)$

Predicted prob dists $\longrightarrow$ $\hat{\boldsymbol{y}}^{(1)}$ $\hat{\boldsymbol{y}}^{(2)}$ $\hat{\boldsymbol{y}}^{(3)}$ $\hat{\boldsymbol{y}}^{(4)}$

$\boldsymbol{U}$ $\boldsymbol{U}$ $\boldsymbol{U}$ $\boldsymbol{U}$

$\boldsymbol{h}^{(0)}$ $\boldsymbol{h}^{(1)}$ $\boldsymbol{h}^{(2)}$ $\boldsymbol{h}^{(3)}$ $\boldsymbol{h}^{(4)}$

$\boldsymbol{W}_h$ $\boldsymbol{W}_h$ $\boldsymbol{W}_h$ $\boldsymbol{W}_h$ $\boldsymbol{W}_h$ ...

$\boldsymbol{W}_e$ $\boldsymbol{W}_e$ $\boldsymbol{W}_e$ $\boldsymbol{W}_e$

$\boldsymbol{e}^{(1)}$ $\boldsymbol{e}^{(2)}$ $\boldsymbol{e}^{(3)}$ $\boldsymbol{e}^{(4)}$

$\boldsymbol{E}$ $\boldsymbol{E}$ $\boldsymbol{E}$ $\boldsymbol{E}$

Corpus $\longrightarrow$ *the* *students* *opened* *their* *exams* *...*

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

29

Training a RNN Language Model

Loss ➞ $J^{(1)}(\theta)$ + $J^{(2)}(\theta)$ + $J^{(3)}(\theta)$ + $J^{(4)}(\theta)$ + ... = $J(\theta) = \dfrac{1}{T}\displaystyle\sum_{t=1}^{T} J^{(t)}(\theta)$

Predicted prob dists ➞ $\hat{\boldsymbol{y}}^{(1)}$ $\hat{\boldsymbol{y}}^{(2)}$ $\hat{\boldsymbol{y}}^{(3)}$ $\hat{\boldsymbol{y}}^{(4)}$

$\boldsymbol{U}$ $\boldsymbol{U}$ $\boldsymbol{U}$ $\boldsymbol{U}$

$\boldsymbol{h}^{(0)}$ $\boldsymbol{h}^{(1)}$ $\boldsymbol{h}^{(2)}$ $\boldsymbol{h}^{(3)}$ $\boldsymbol{h}^{(4)}$

$\boldsymbol{W}_h$ $\boldsymbol{W}_h$ $\boldsymbol{W}_h$ $\boldsymbol{W}_h$ $\boldsymbol{W}_h$ ...

$\boldsymbol{W}_e$ $\boldsymbol{W}_e$ $\boldsymbol{W}_e$ $\boldsymbol{W}_e$

$\boldsymbol{e}^{(1)}$ $\boldsymbol{e}^{(2)}$ $\boldsymbol{e}^{(3)}$ $\boldsymbol{e}^{(4)}$

$\boldsymbol{E}$ $\boldsymbol{E}$ $\boldsymbol{E}$ $\boldsymbol{E}$

Corpus ➞ *the* *students* *opened* *their* *exams* ...

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

Training a RNN Language Model

- However: Computing loss and gradients across entire corpus $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$ is too expensive!

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

- In practice, consider $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$ 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.

31

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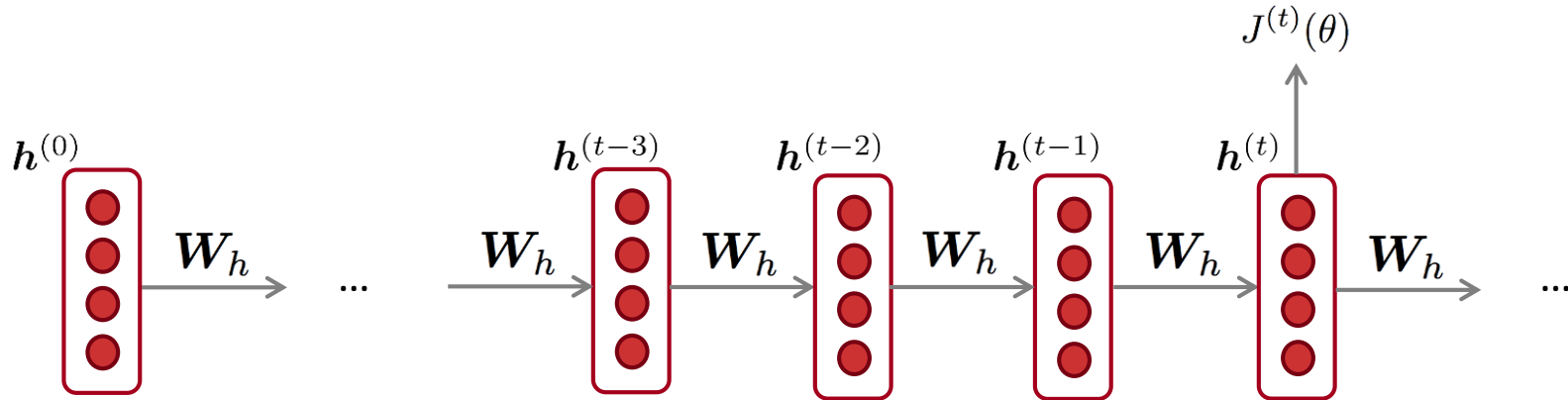
Question: What's the derivative of $J^{(t)}(\theta)$ w.r.t. the repeated weight matrix $\boldsymbol{W_h}$?

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

"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:

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

One final output $f(x(t), y(t))$

Two intermediate outputs $x(t)$ $\qquad$ $y(t)$

One input t

33
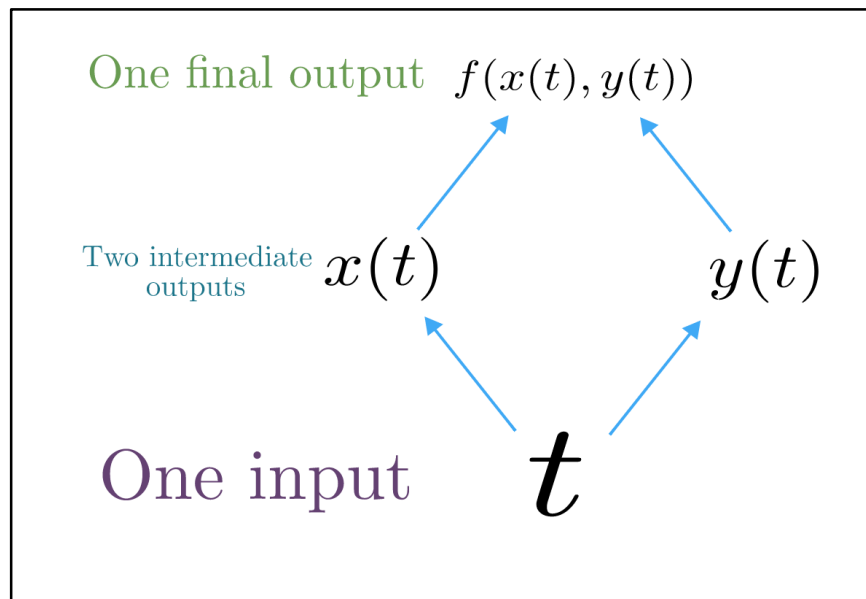
Backpropagation for RNNs: Proof sketch

- 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:

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

In our example:

$$J^{(t)}(\theta)$$

$$W_h\big|_{(1)} \quad W_h\big|_{(2)} \quad \ldots \quad W_h\big|_{(t)}$$

equals   equals   equals

$$W_h$$
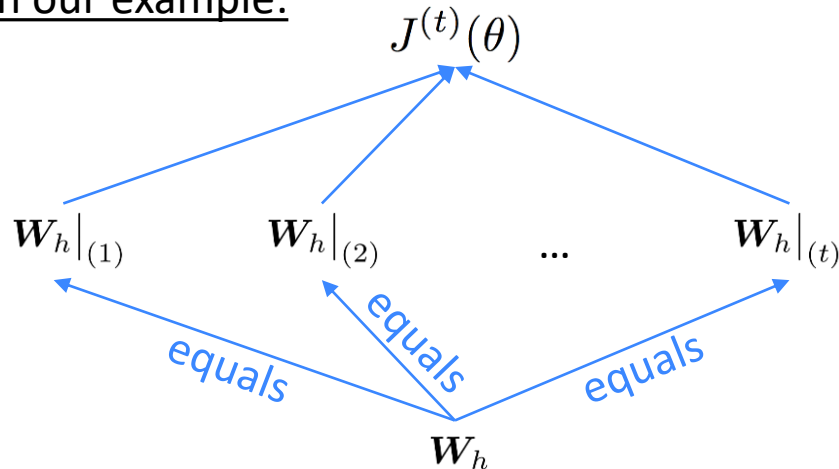
Apply the multivariable chain rule:

$$= 1$$

$$\frac{\partial J^{(t)}}{\partial \boldsymbol{W}_h} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \boldsymbol{W}_h}\bigg|_{(i)} \boxed{\frac{\partial \boldsymbol{W}_h\big|_{(i)}}{\partial \boldsymbol{W}_h}}$$

$$= \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \boldsymbol{W}_h}\bigg|_{(i)}$$

34

Backpropagation for RNNs
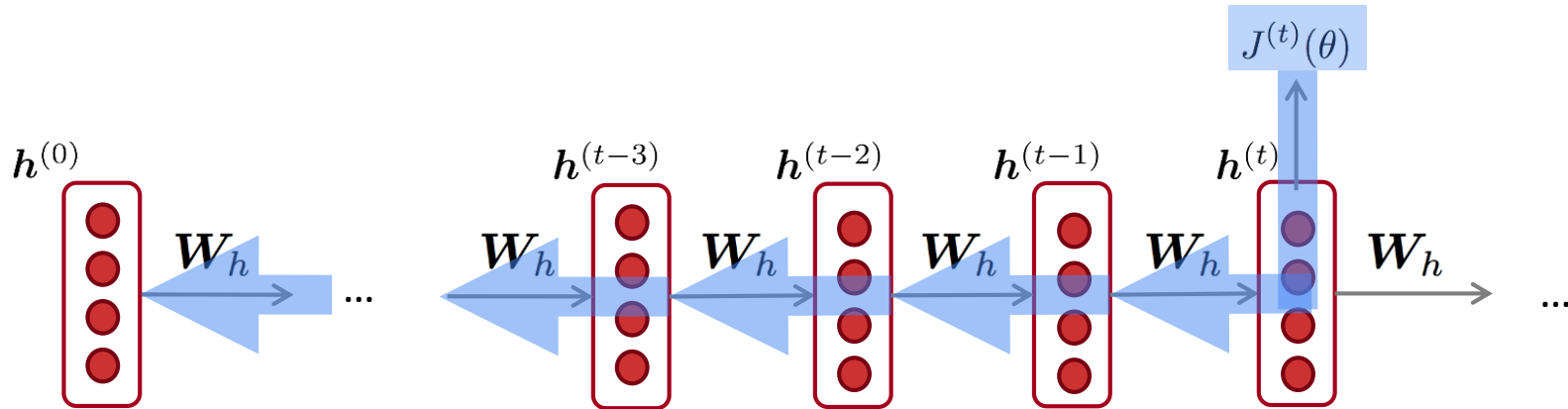
$$\frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} = \boxed{\sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}}\bigg|_{(i)}}$$

Question: How do we calculate this?

Answer: Backpropagate over timesteps $i=t,\ldots,0$, summing gradients as you go.
This algorithm is called **"backpropagation through time"**

Generating text with a RNN Language Model

Just like a n-gram Language Model, you can use a RNN Language Model to generate text by repeated sampling. Sampled output is next step's input.
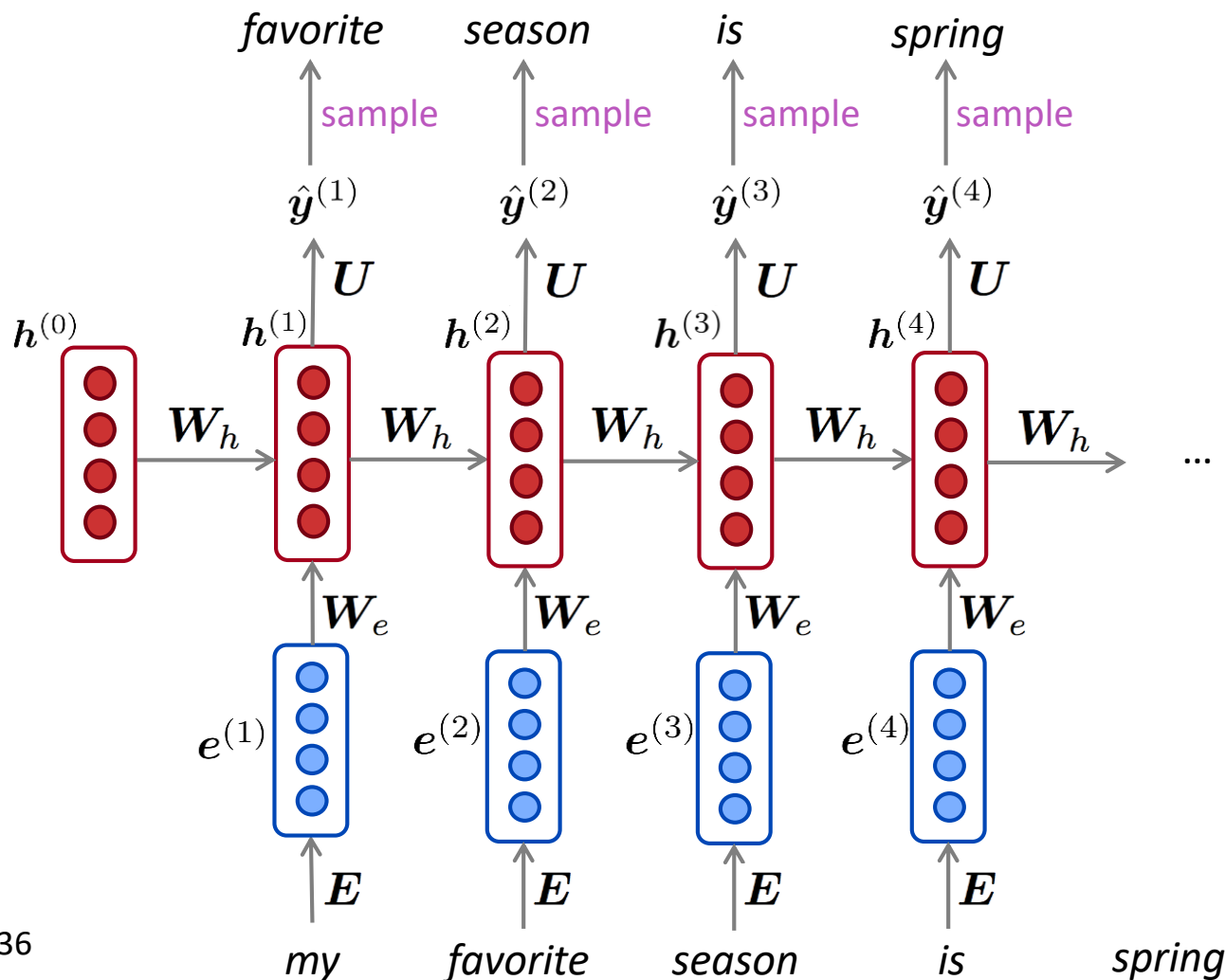
Generating text with a RNN Language Model

- Let's have some fun!
- You can train a RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on Obama speeches:

The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done.

Source: https://medium.com/@samim/obama-rnn-machine-generated-political-speeches-c8abd18a2ea0

Generating text with a RNN Language Model

- Let's have some fun!
- You can train a 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.

Generating text with a RNN Language Model

- Let's have some fun!
- You can train a RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on recipes:


```
        Title: CHOCOLATE RANCH BARBECUE
Categories: Game, Casseroles, Cookies, Cookies
        Yield: 6 Servings

    2 tb Parmesan cheese -- chopped
    1 c  Coconut milk
    3     Eggs, beaten
```

Place each pasta over layers of lumps. Shape mixture into the moderate oven and simmer until firm. Serve hot in bodied fresh, mustard, orange and cheese.

Combine the cheese and salt together the dough in a large skillet; add the ingredients and stir in the chocolate and pepper.

Source: https://gist.github.com/nylki/1efbaa36635956d35bcc

39

Generating text with a RNN Language Model

- Let's have some fun!

- You can train a RNN-LM on any kind of text, then generate text in that style.

- RNN-LM trained on paint color names:

Ghasty Pink 231 137 165		Sand Dan 201 172 143	
Power Gray 151 124 112		Grade Bat 48 94 83	
Navel Tan 199 173 140		Light Of Blast 175 150 147	
Bock Coe White 221 215 236		Grass Bat 176 99 108	
Horble Gray 178 181 196		Sindis Poop 204 205 194	
Homestar Brown 133 104 85		Dope 219 209 179	
Snader Brown 144 106 74		Testing 156 101 106	
Golder Craam 237 217 177		Stoner Blue 152 165 159	
Hurky White 232 223 215		Burble Simp 226 181 132	
Burf Pink 223 173 179		Stanky Bean 197 162 171	
Rose Hork 230 215 198		Turdly 190 164 116	

This is an example of a character-level RNN-LM (predicts what character comes next)

Source: http://aiweirdness.com/post/160776374467/new-paint-colors-invented-by-neural-network

Evaluating Language Models

- The standard evaluation metric for Language Models is perplexity.

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

Normalized by number of words

Inverse probability of corpus, according to Language Model

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

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

Lower perplexity is better!

RNNs have greatly improved perplexity

n-gram model →

Increasingly complex RNNs

Model	Perplexity
Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
LSTM-2048 (Jozefowicz et al., 2016)	43.7
2-layer LSTM-8192 (Jozefowicz et al., 2016)	30
Ours small (LSTM-2048)	43.9
Ours large (2-layer LSTM-2048)	39.8

Perplexity improves (lower is better)

Source: https://research.fb.com/building-an-efficient-neural-language-model-over-a-billion-words/

42

Why should we care about Language Modeling?
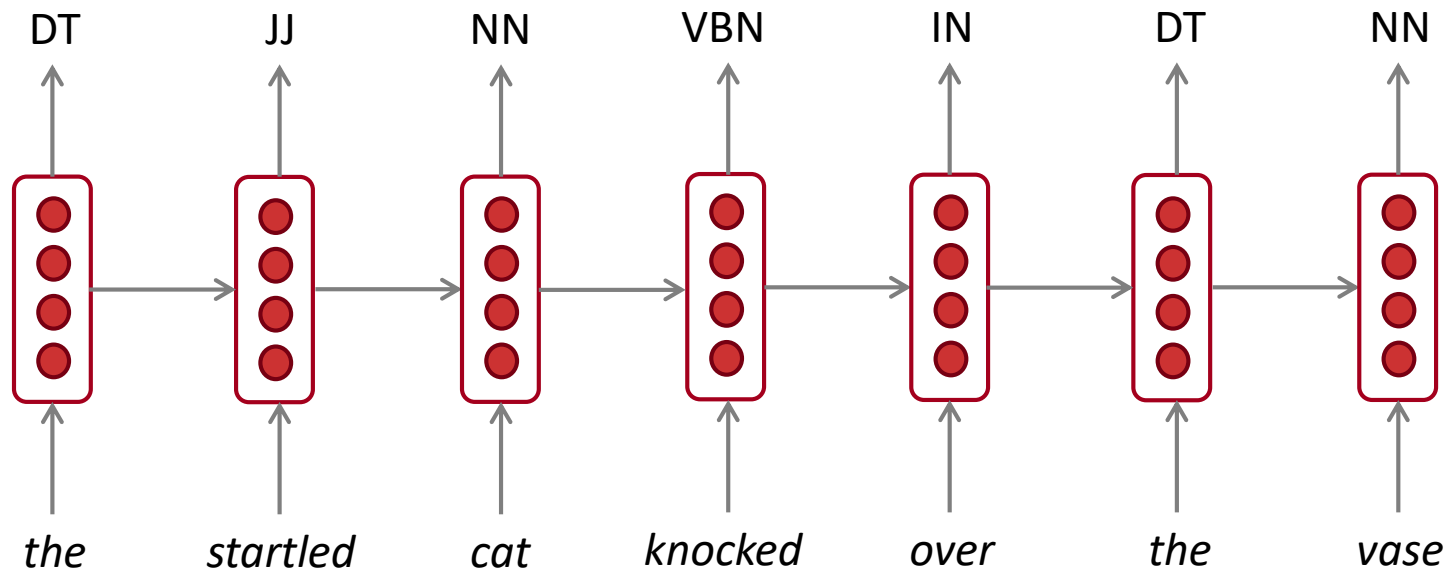
- Language Modeling is a benchmark task that helps us measure our progress on understanding language

- 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.

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 ≠ Language Model

- We've shown that RNNs are a great way to build a LM.
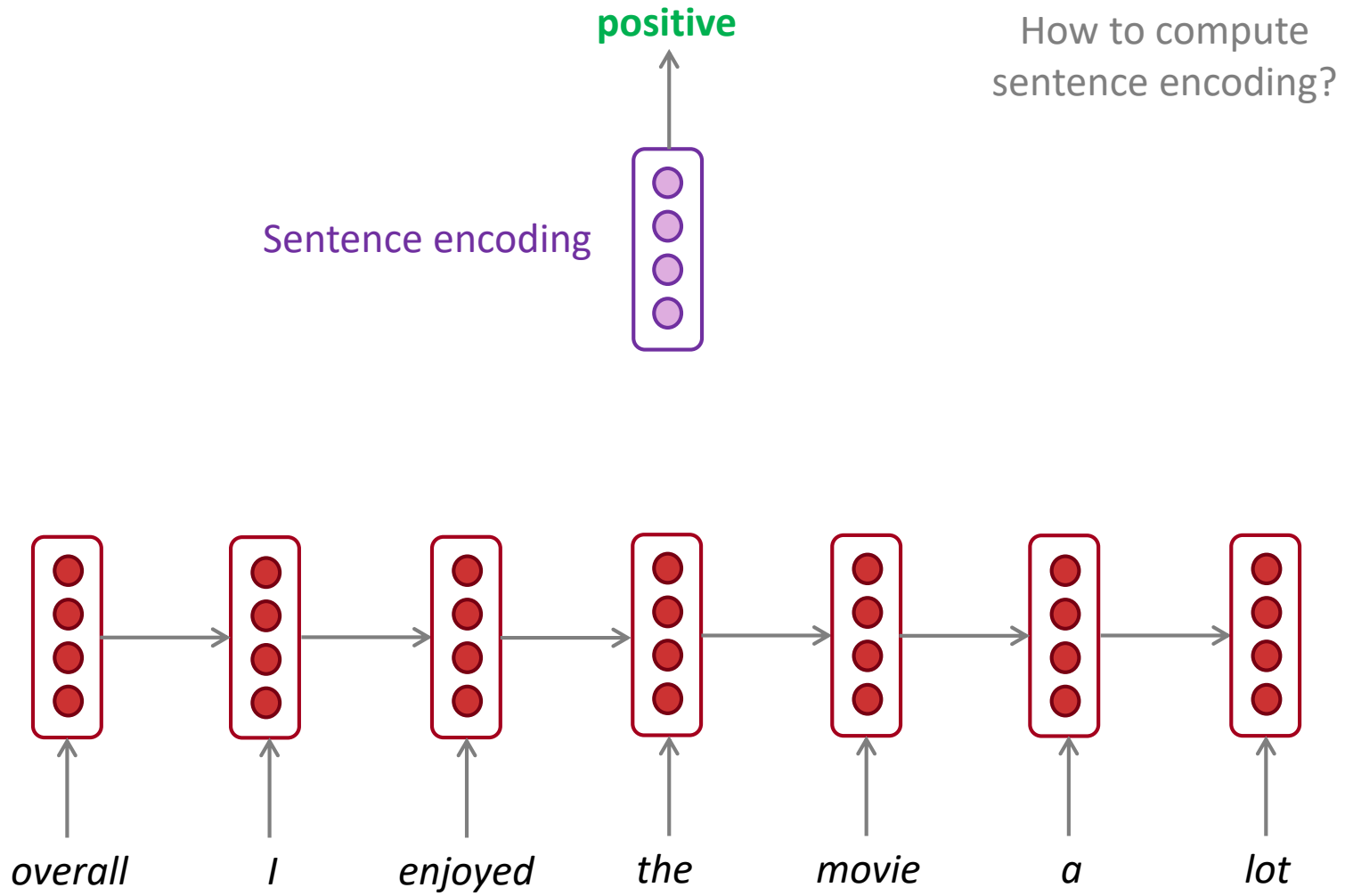
- But RNNs are useful for much more!

RNNs can be used for tagging
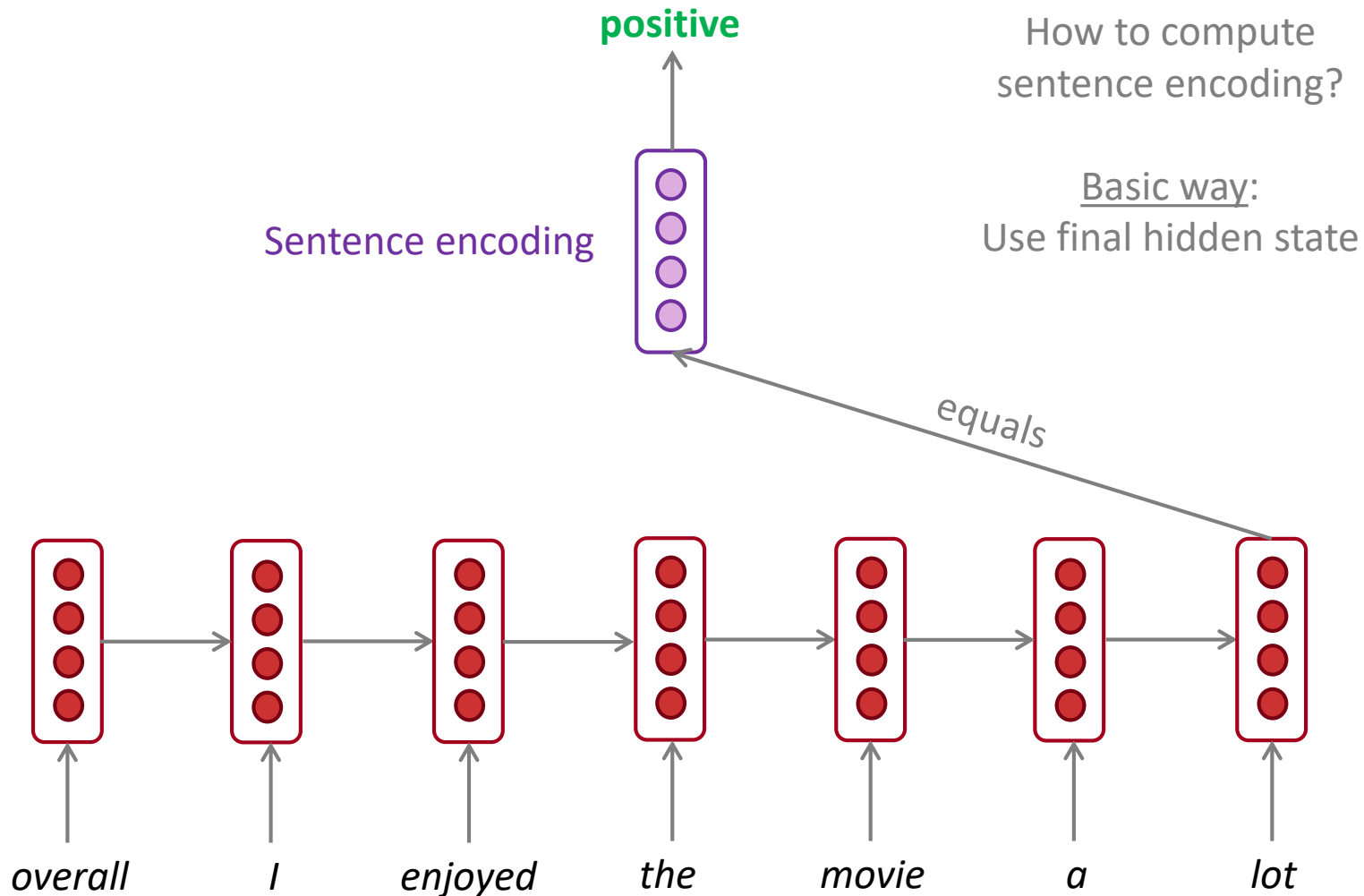
e.g. part-of-speech tagging, named entity recognition

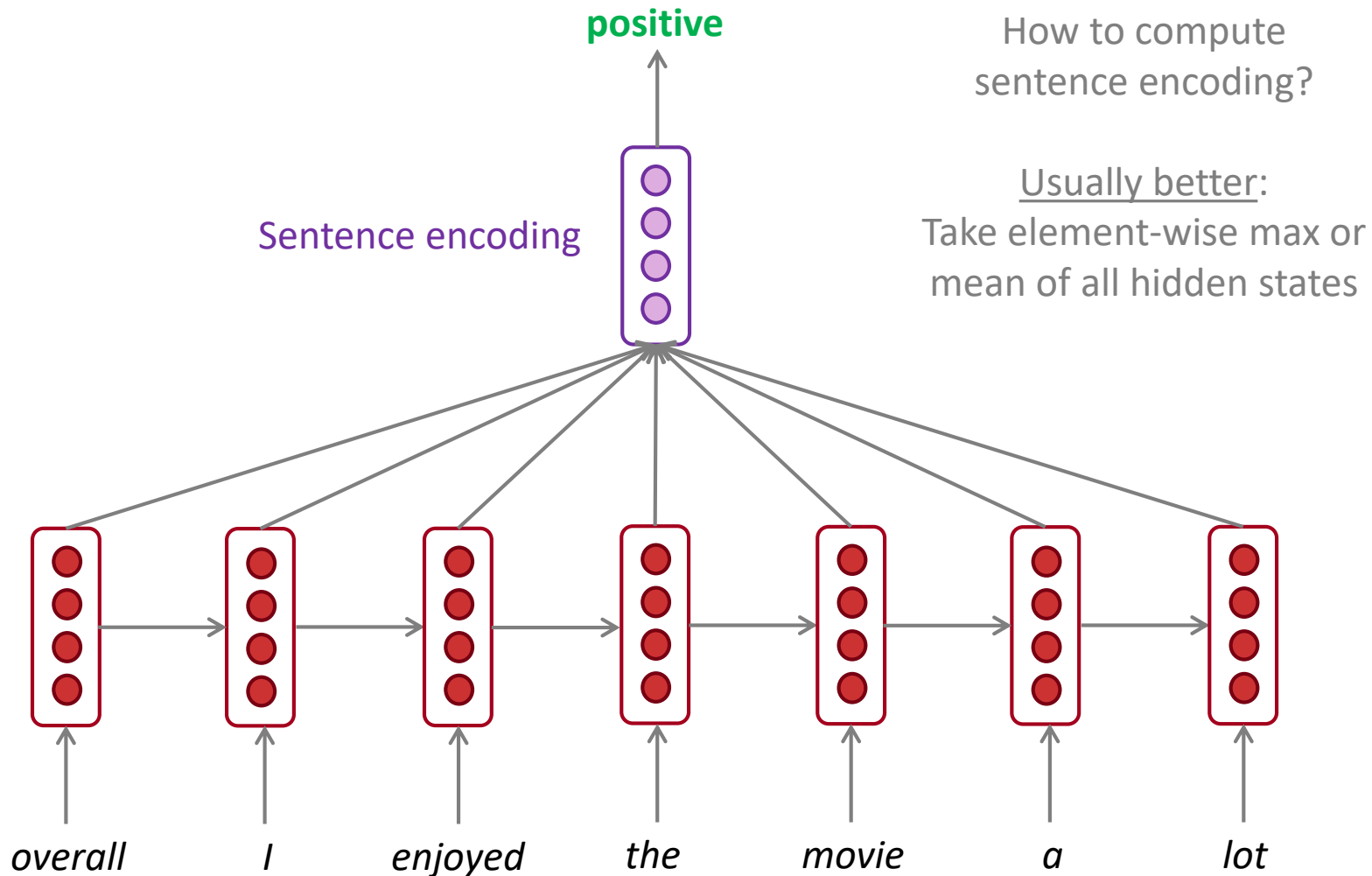# RNNs can be used for sentence classification

e.g. sentiment classification

positive

How to compute sentence encoding?

Sentence encoding

overall   I   enjoyed   the   movie   a   lot

RNNs can be used for sentence classification

e.g. sentiment classification

positive

How to compute
sentence encoding?

Sentence encoding

Basic way:
Use final hidden state

equals

overall   I   enjoyed   the   movie   a   lot

RNNs can be used for sentence classification

e.g. sentiment classification

positive

Sentence encoding

How to compute sentence encoding?

Usually better:
Take element-wise max or mean of all hidden states

overall   I   enjoyed   the   movie   a   lot

RNNs can be used as an encoder module

e.g. question answering, machine translation, *many other tasks!*

Answer: German

Here the RNN acts as an **encoder** for the Question (the hidden states represent the Question). The encoder is part of a larger neural system.
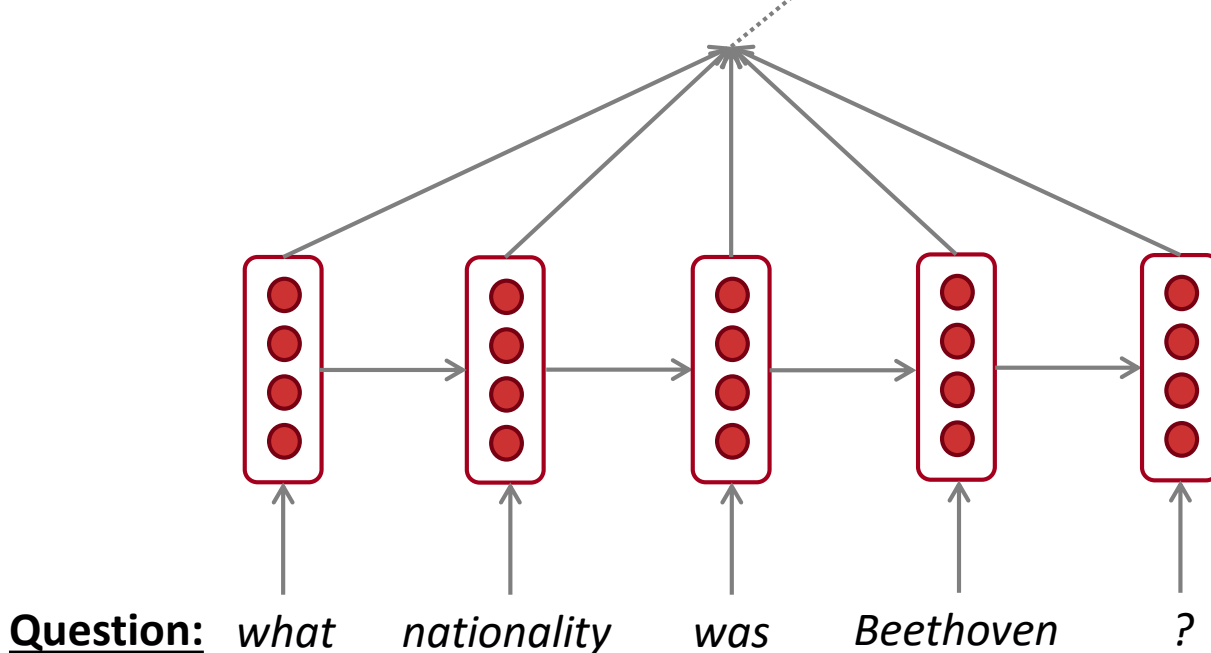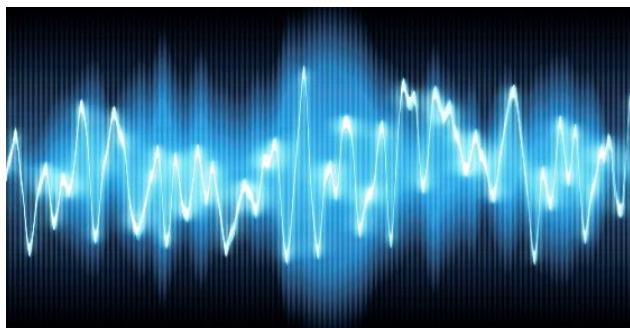
lots of neural architecture

lots of neural architecture

Context: *Ludwig van Beethoven was a German composer and pianist. A crucial figure …*
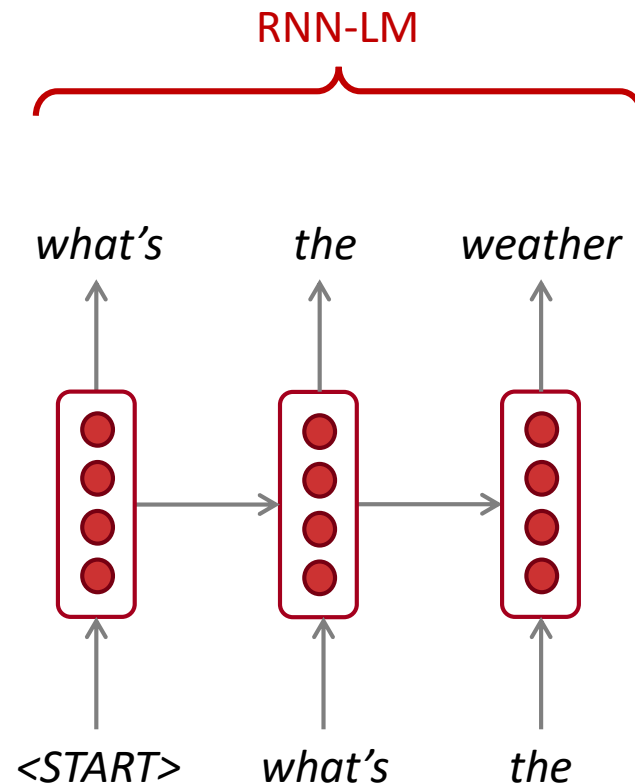

Question: *what   nationality   was   Beethoven   ?*

RNN-LMs can be used to generate text

e.g. speech recognition, machine translation, summarization

RNN-LM

Input (audio)

conditioning

what's   the   weather

<START>   what's   the

This is an example of a *conditional language model*.
We'll see Machine Translation in much more detail later.

A note on terminology

RNN described in this lecture = "vanilla RNN"

Next lecture: You will learn about other RNN flavors

like GRU   and LSTM   and multi-layer RNNs

By the end of the course: You will understand phrases like
"stacked bidirectional LSTM with residual connections and self-attention"

Next time

- Problems with RNNs!
  - Vanishing gradients

    motivates

- Fancy RNN variants!
  - LSTM
  - GRU
  - multi-layer
  - bidirectional