Natural Language Processing with Deep Learning

CS224N/Ling284

Lecture 8:
Recurrent Neural Networks and Language Models

Abigail See

Announcements

- Assignment 1: Grades will be released after class
- Assignment 2: Coding session next week on Monday; details on Piazza
- Midterm logistics: Fill out form on Piazza if you can't do main midterm, have special requirements, or other special case

Announcements

- Default Final Project (PA4) release late tonight
  - Read the handout, look at the code, decide which project you want to do
  - You may not understand all the technical parts, but you'll get an overview
  - You don't yet have the Azure resources you need to run the code

- Project proposal due next week (Thurs Feb 8)
  - Details released later today
  - Everyone submits their teams
  - Custom final project teams also describe their project

Call for participation

Overview

Today we will:

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

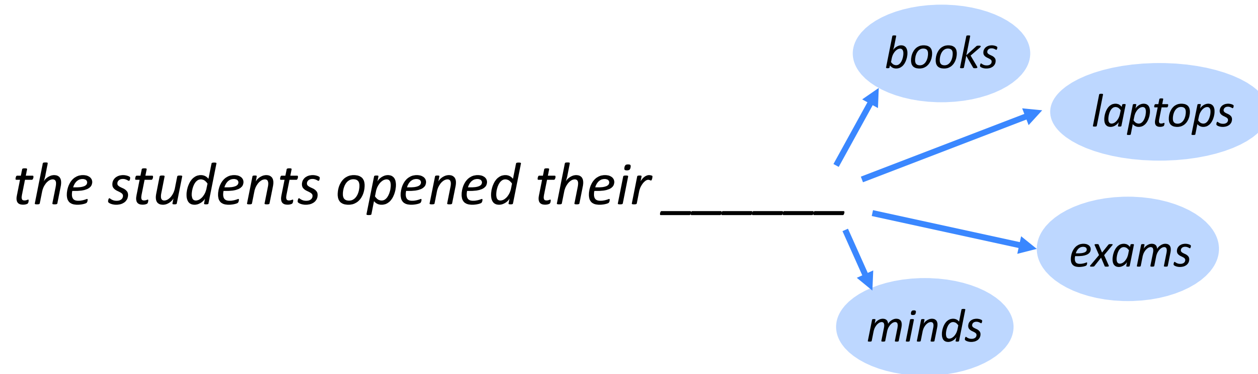
motivates

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

THE most important idea
for the rest of the class!

Language Modeling

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



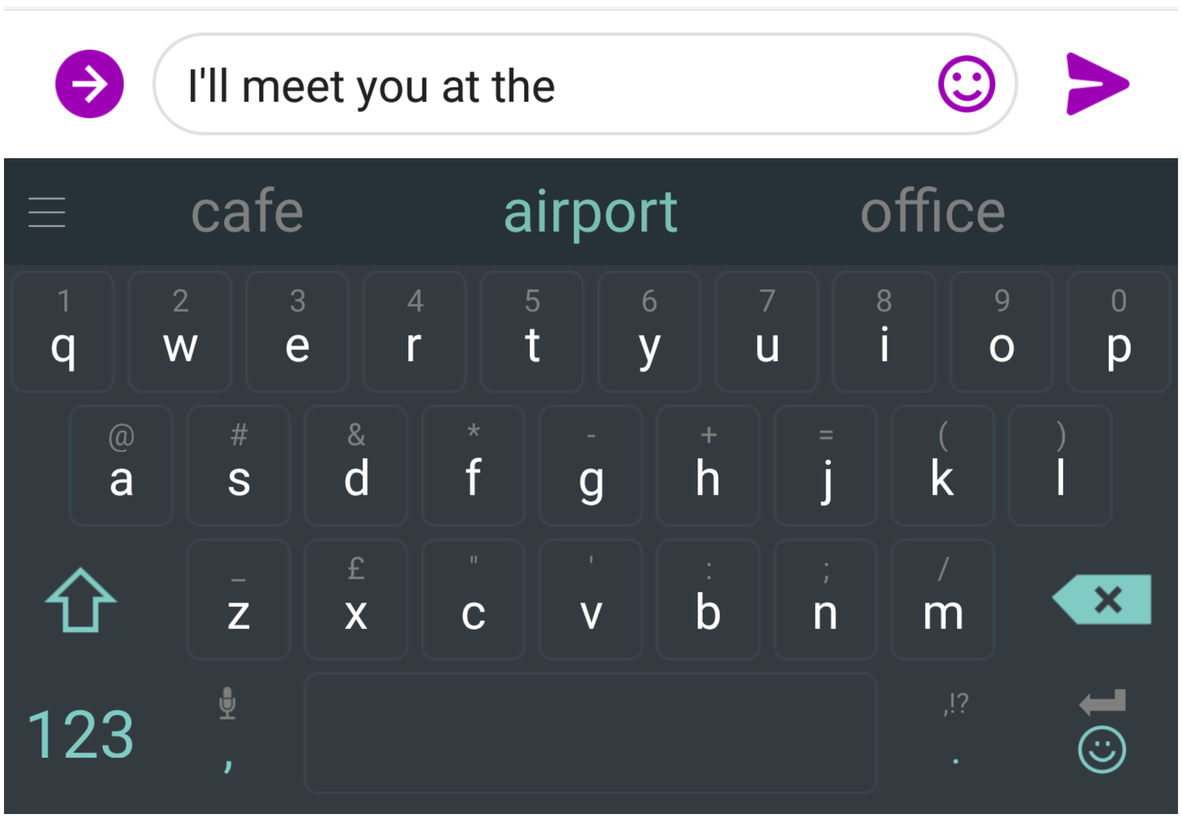
- More formally: given a sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$, compute the probability distribution of the next word $\boldsymbol{x}^{(t+1)}$:

$$P(\boldsymbol{x}^{(t+1)} = \boldsymbol{w}_j \mid \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

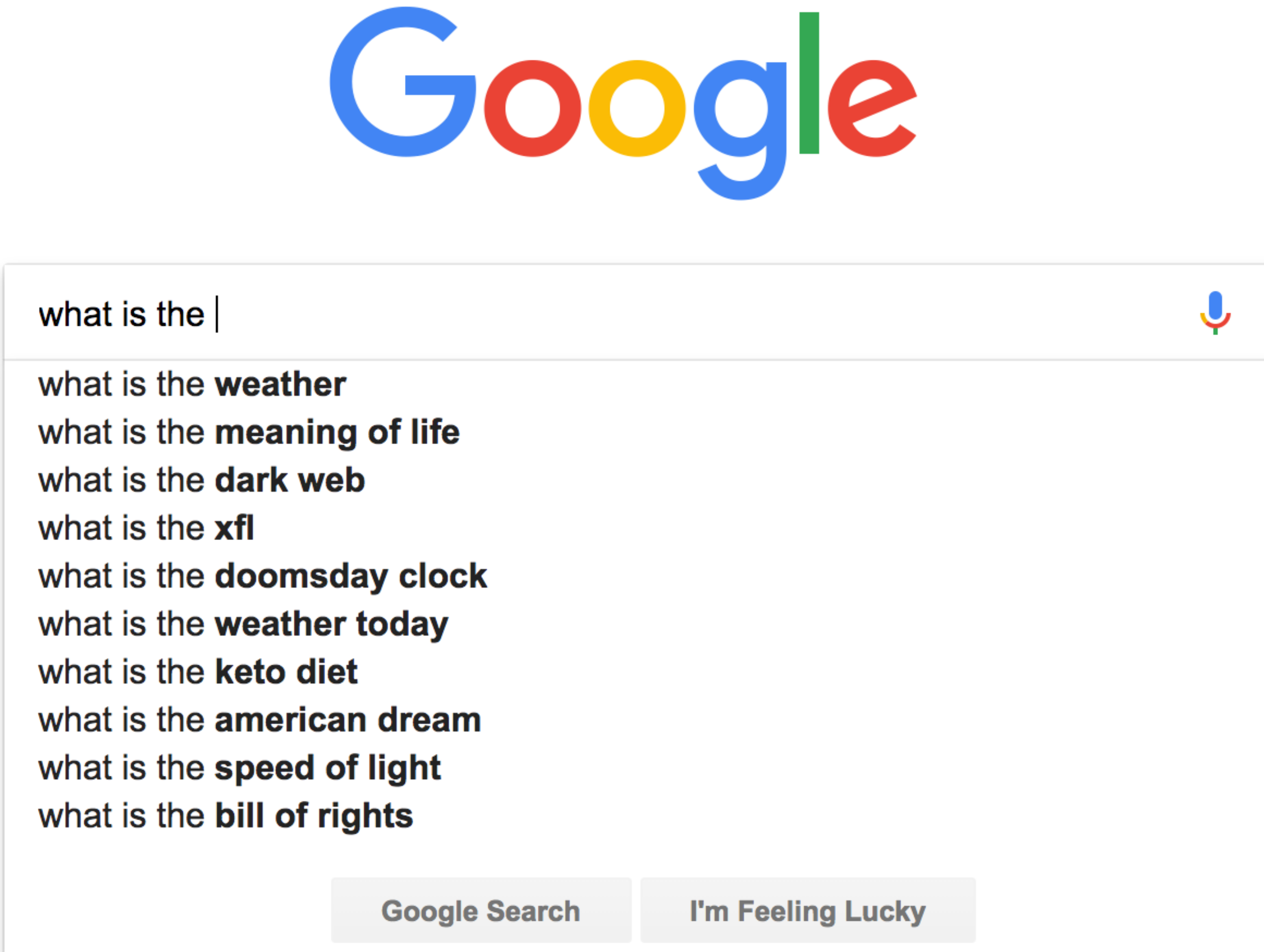
where $\boldsymbol{w}_j$ is a word in the vocabulary $V = \{\boldsymbol{w}_1, ..., \boldsymbol{w}_{|V|}\}$

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

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

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

prob of a n-gram

prob of a (n-1)-gram

$$= \frac{\boxed{P(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}}{\boxed{P(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}}$$ (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!

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

n-gram Language Models: Example

Suppose we are learning a 4-gram Language Model.

~~*as the proctor started the clock, the*~~ $\underbrace{\textit{students opened their}}_{\text{condition on this}}$ ______

discard

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

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?

Problems with n-gram Language Models

Sparsity Problem 1

Problem: What if *"students opened their* $\boldsymbol{w}_j$*"* never occurred in data? Then $\boldsymbol{w}_j$ has probability 0!

(Partial) Solution: Add small δ to count for every $\boldsymbol{w}_j \in V$. This is called *smoothing*.

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

Sparsity Problem 2

Problem: What if *"students opened their"* never occurred in data? Then we can't calculate probability for *any* $\boldsymbol{w}_j$!

(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.

Problems with n-gram Language Models

Storage: Need to store count for all possible *n*-grams. So model size is O(exp(*n*)).

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

Increasing *n* makes model size huge!

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

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

Otherwise, seems reasonable!

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 .

Incoherent! We need to consider more than 3 words at a time if we want to generate good text.

But increasing *n* worsens sparsity problem, and exponentially 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 $P(\boldsymbol{x}^{(t+1)} = \boldsymbol{w}_j \mid \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$

- How about a window-based neural model?
  - We saw this applied to Named Entity Recognition in Lecture 4

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

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

hidden layer

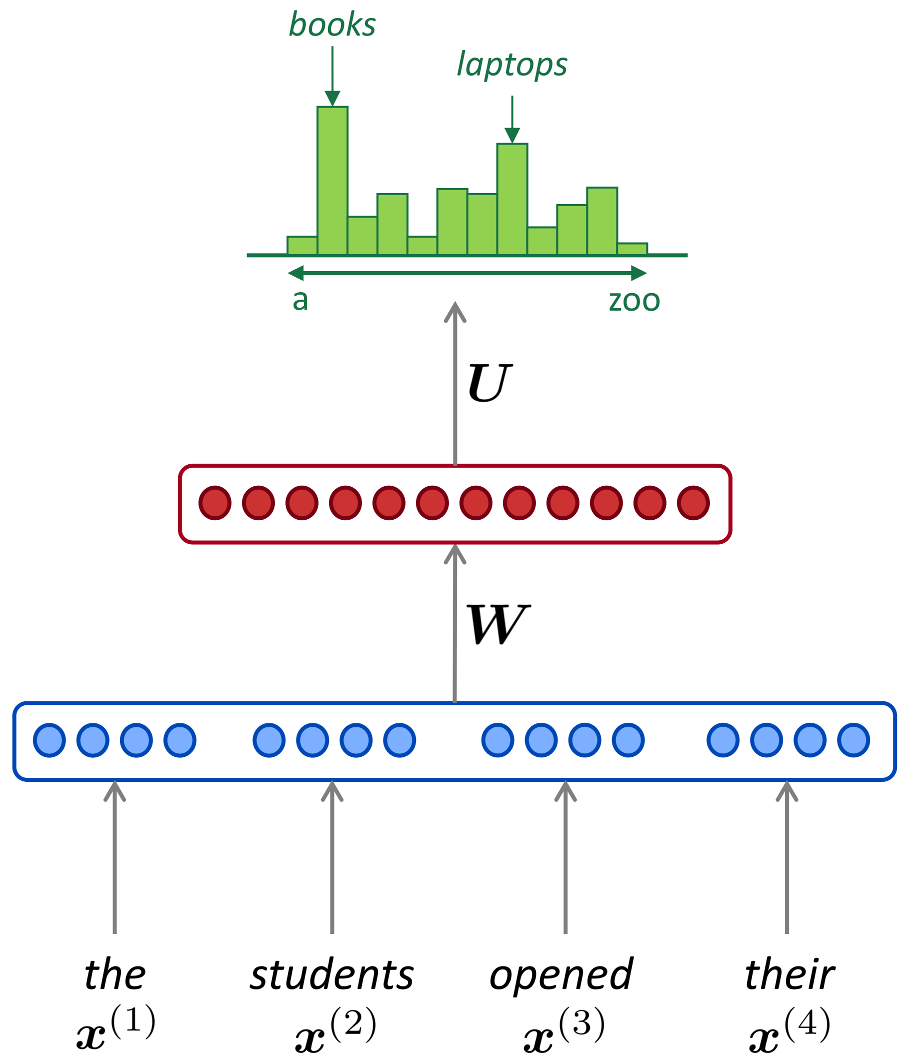
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

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

words / one-hot vectors

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



A fixed-window neural Language Model

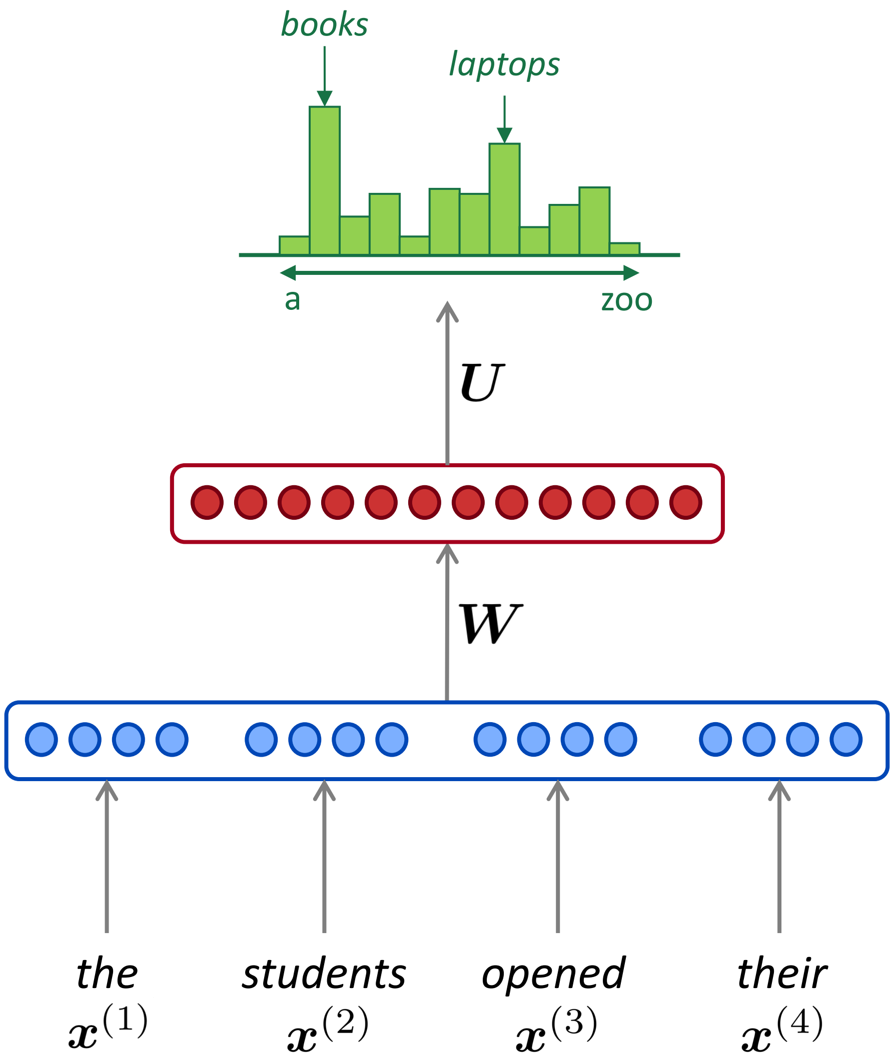
Improvements over *n*-gram LM:

- No sparsity problem
- Model size is O(*n*) not O(exp(*n*))

Remaining **problems**:

- Fixed window is too small
- Enlarging window enlarges $\boldsymbol{W}$
- Window can never be large enough!
- Each $\boldsymbol{x}^{(i)}$ uses different rows of $\boldsymbol{W}$. We don't share weights across the window.

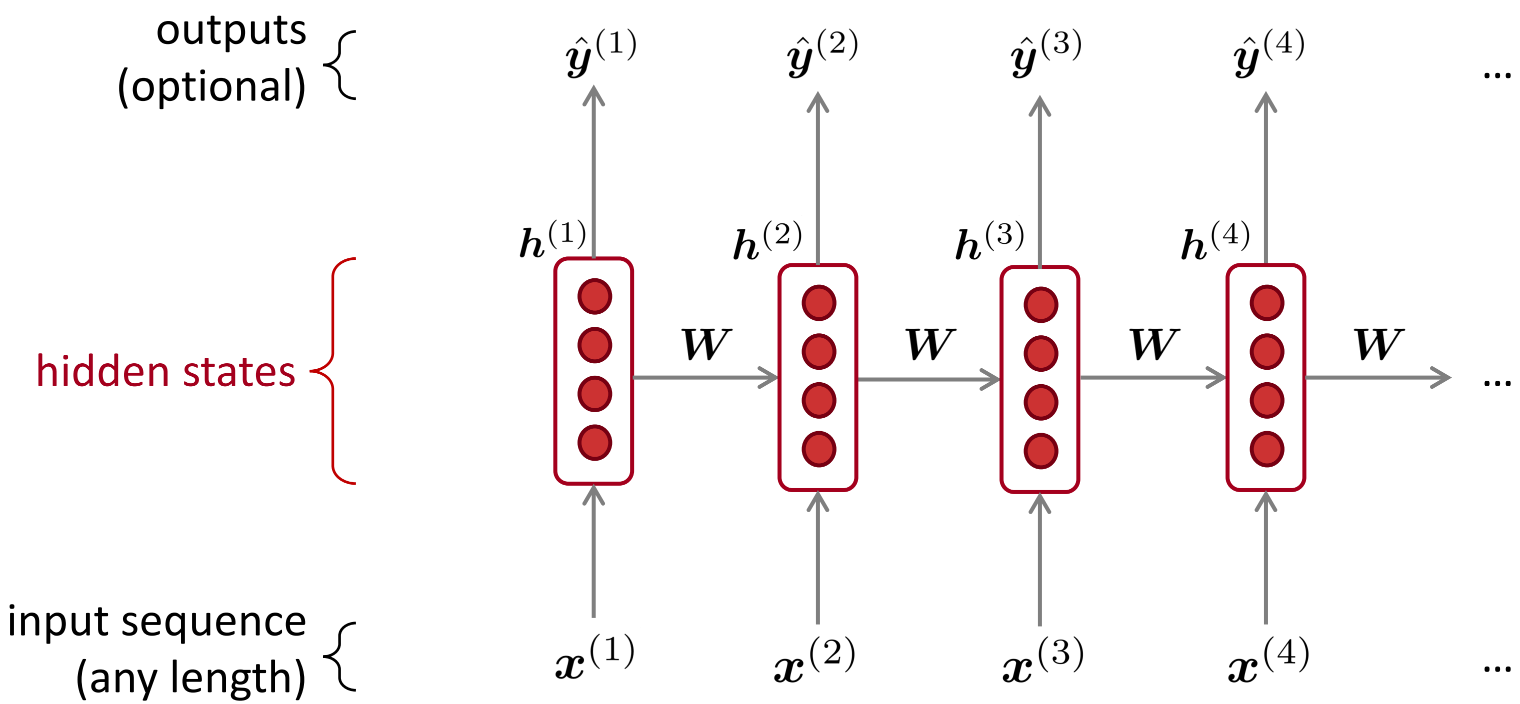
We need a neural architecture that can process *any length input*



Recurrent Neural Networks (RNN)

A family of neural architectures

Core idea: Apply the same weights $\boldsymbol{W}$ *repeatedly*



A RNN Language Model

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

output distribution

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

hidden states

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

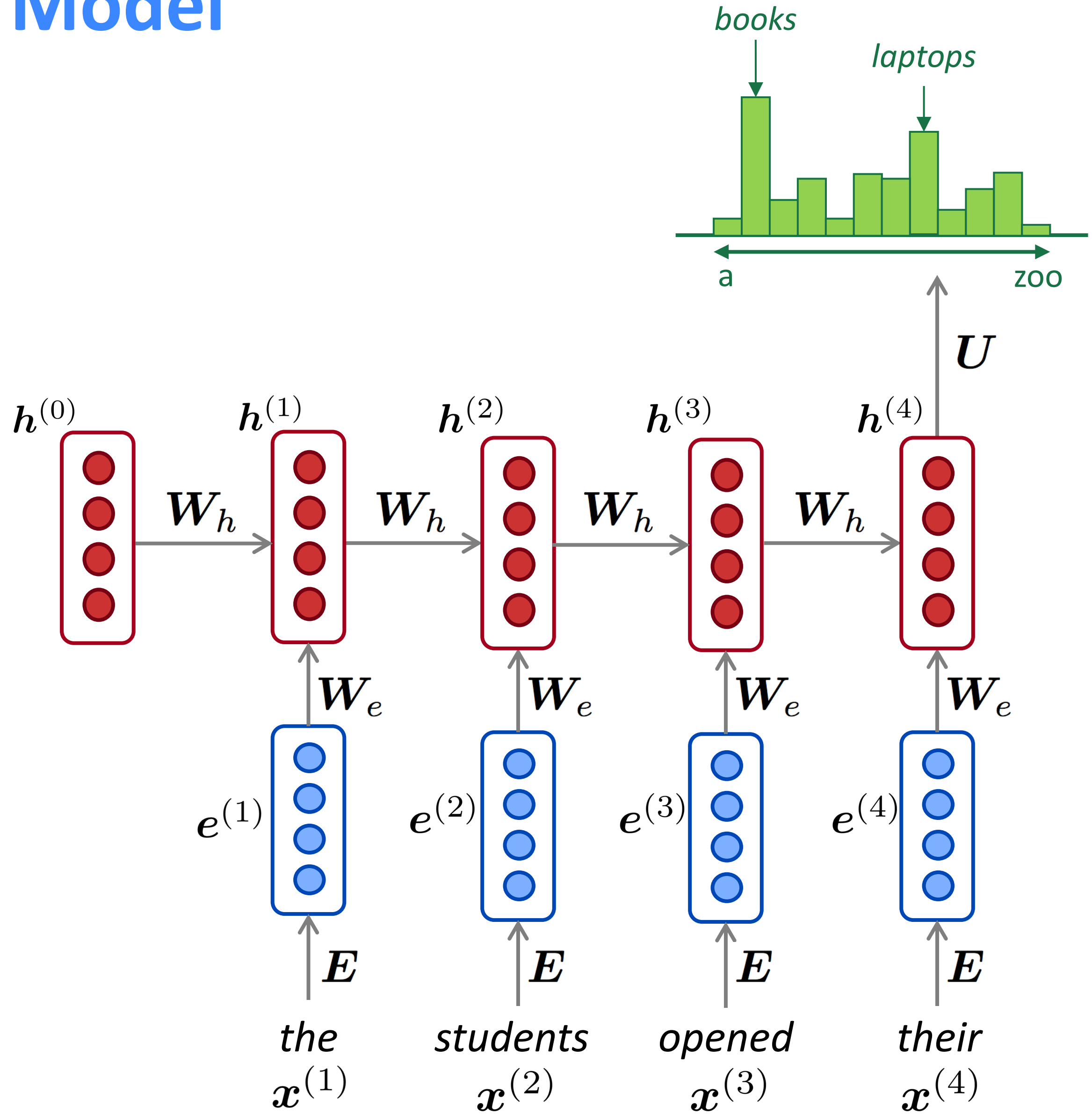
$\boldsymbol{h}^{(0)}$ is the initial hidden state

word embeddings

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

words / one-hot vectors

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



__Note__: this input sequence could be much longer, but this slide doesn't have space!

A RNN Language Model

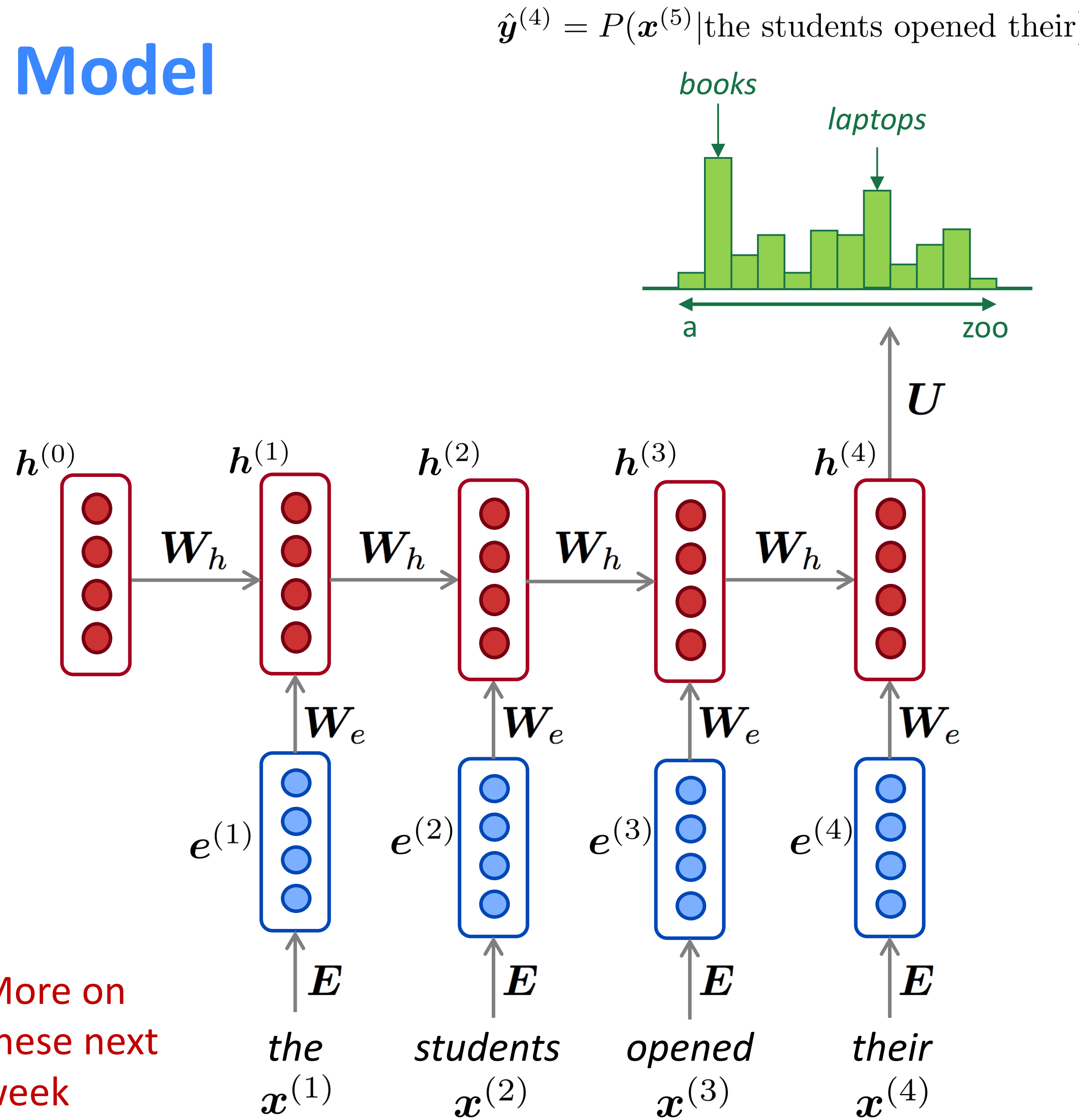
RNN **Advantages**:

- Can process any length input
- Model size doesn't increase for longer input
- Computation for step *t* can (in theory) use information from many steps back
- Weights are shared across timesteps → representations are shared

RNN **Disadvantages**:

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

More on these next week



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

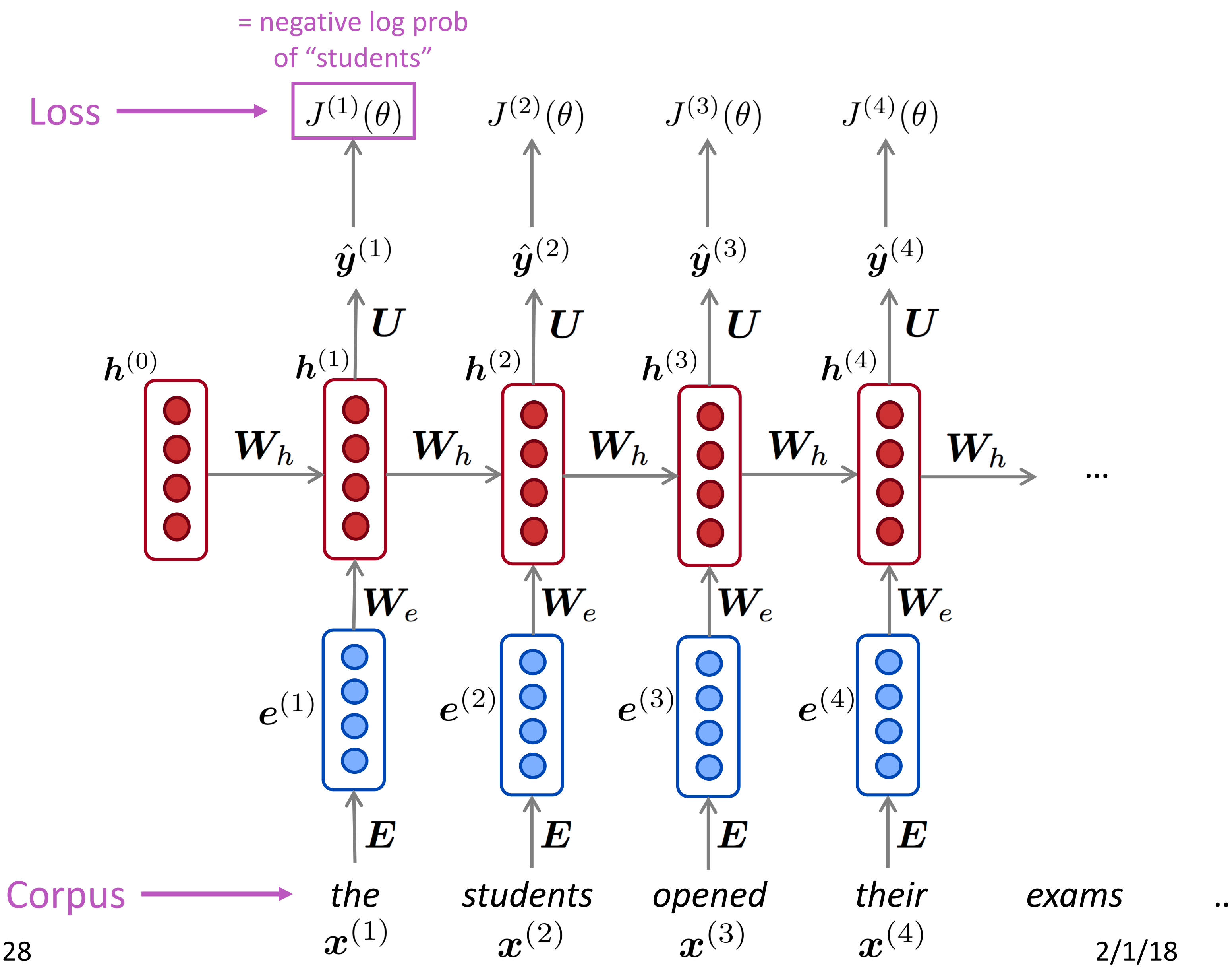
- Loss function on step *t* is usual cross-entropy between our predicted probability distribution $\hat{\boldsymbol{y}}^{(t)}$, and the true next word $\boldsymbol{y}^{(t)} = \boldsymbol{x}^{(t+1)}$:

$$J^{(t)}(\theta) = CE(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) = -\sum_{j=1}^{|V|} y_j^{(t)} \log \hat{y}_j^{(t)}$$

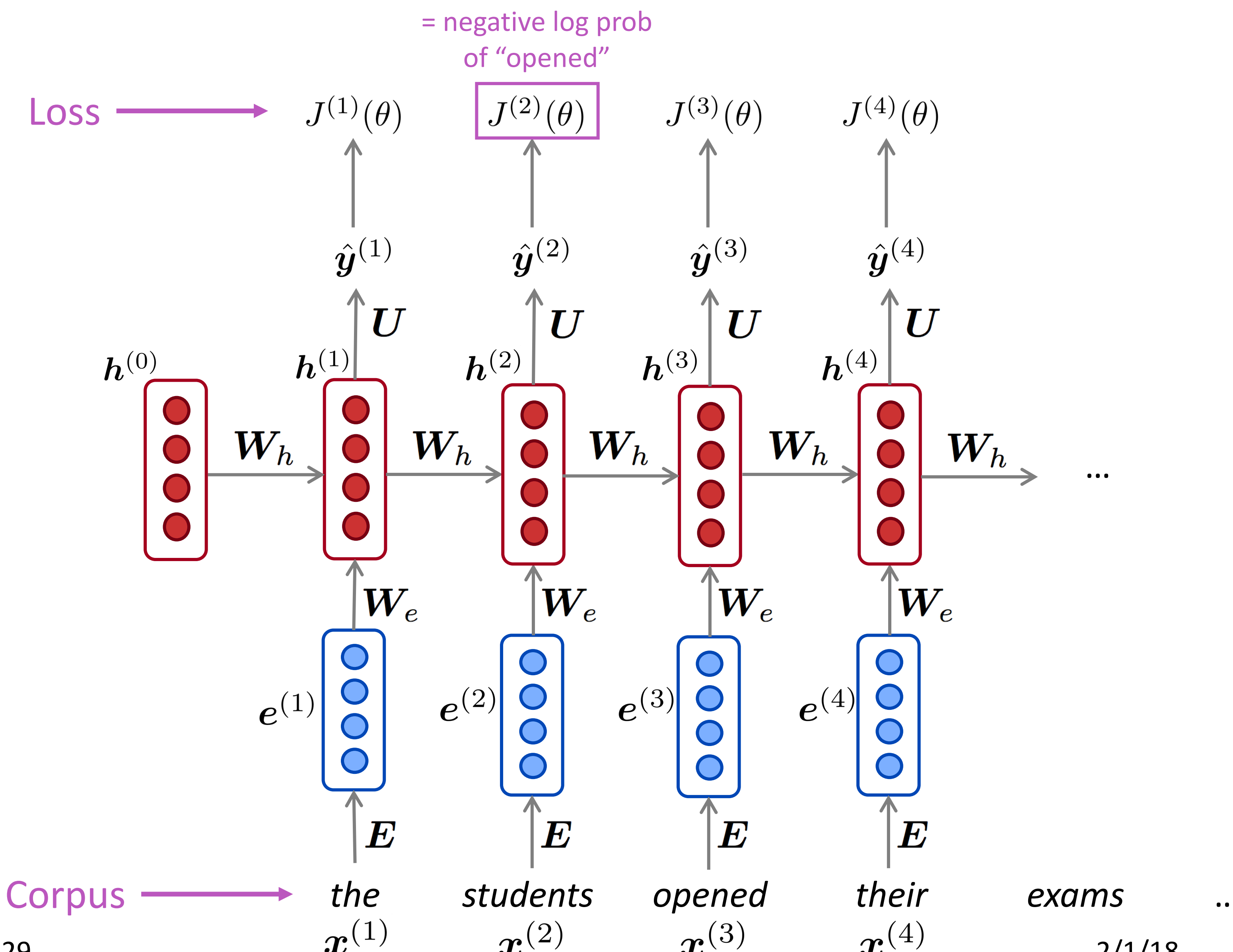
- Average this to get overall loss for entire training set:

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

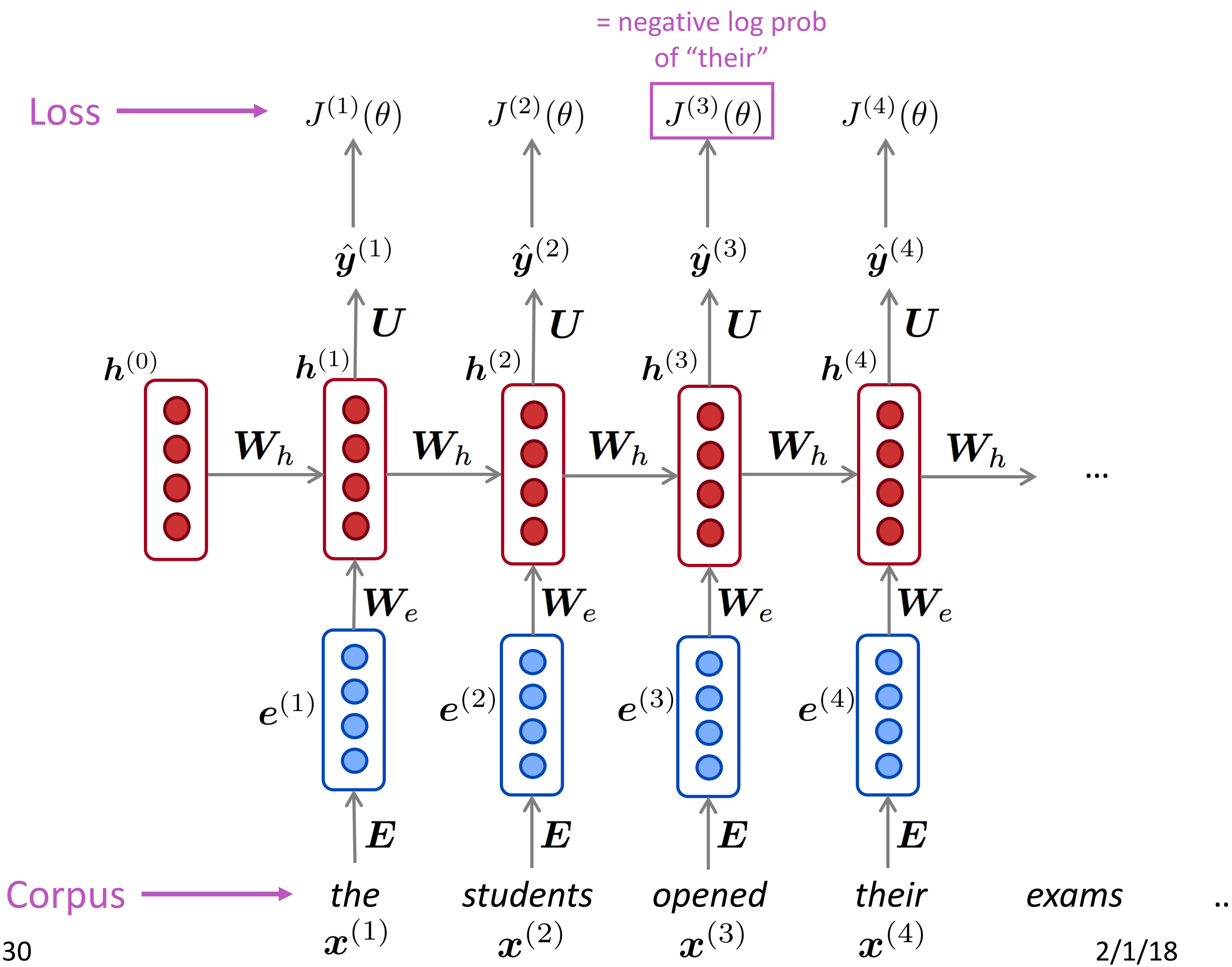
Training a RNN Language Model



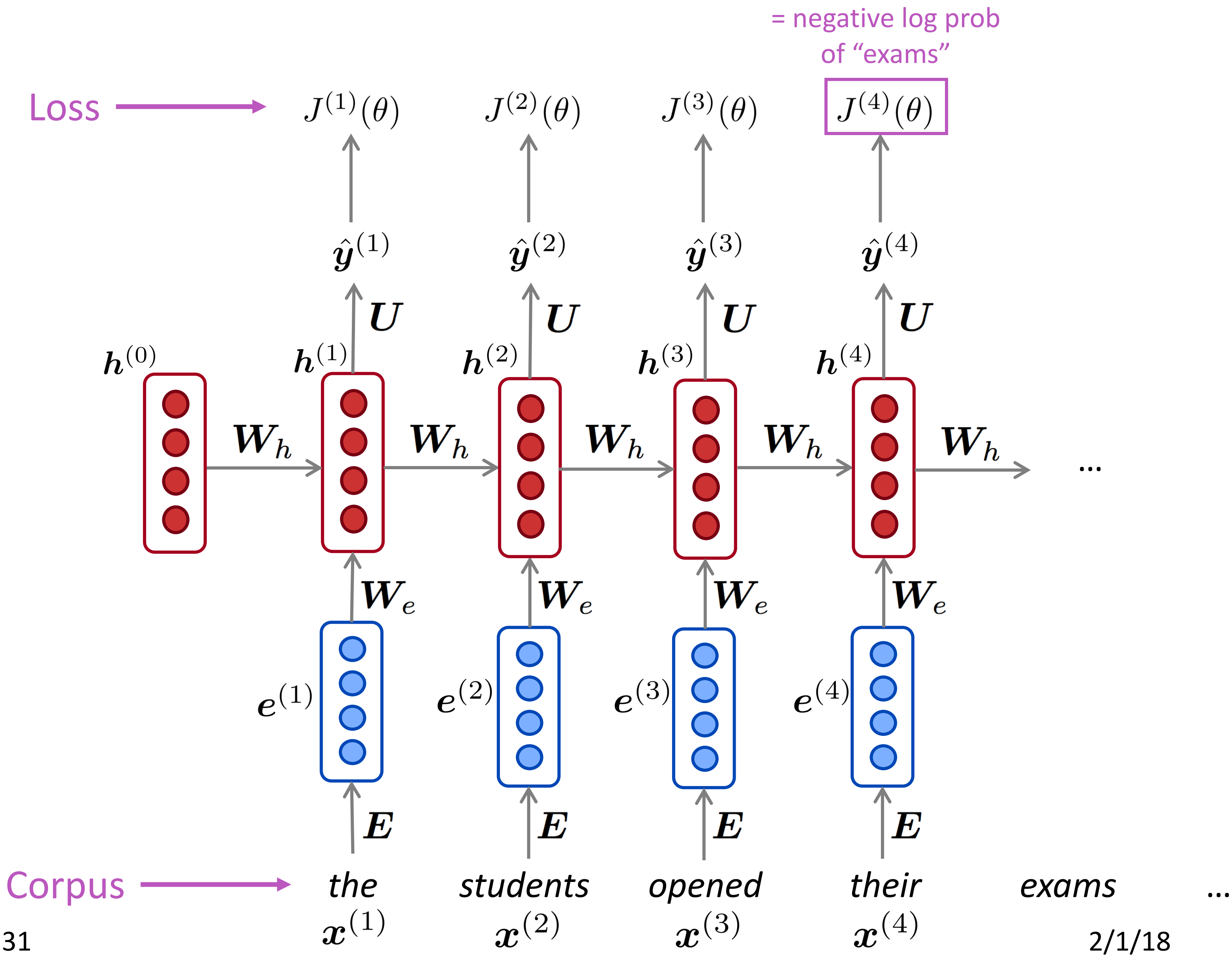
Training a RNN Language Model



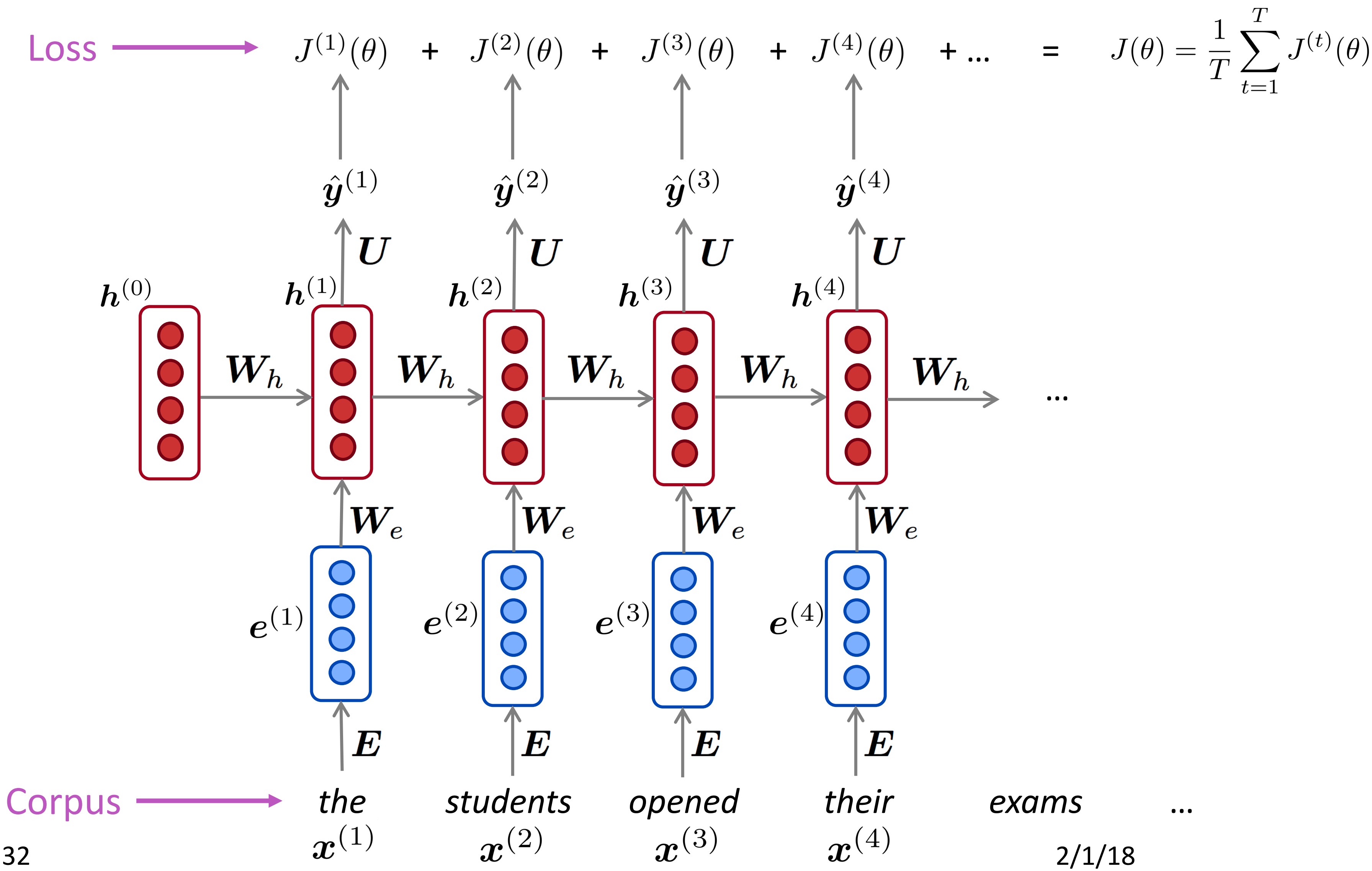
Training a RNN Language Model



Training a RNN Language Model



Training a RNN Language Model



Training a RNN Language Model

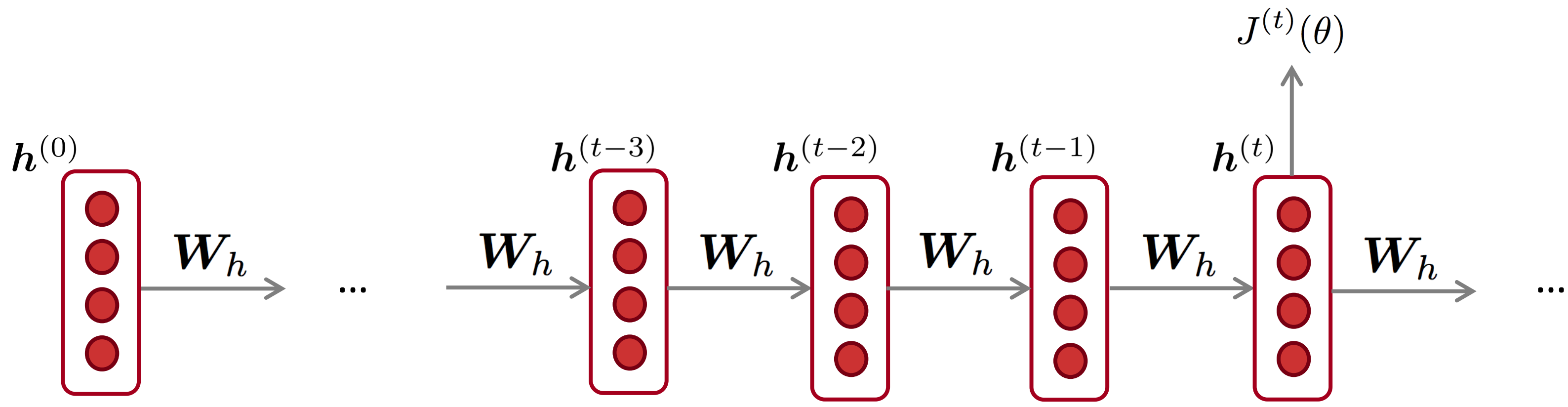
- However: Computing loss and gradients across entire corpus is too expensive!
- Recall: Stochastic Gradient Descent allows us to compute loss and gradients for small chunk of data, and update.

- → In practice, consider $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$ as a sentence

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

- Compute loss $J(\theta)$ for a sentence (actually usually a batch of sentences), compute gradients and update weights. Repeat.

Backpropagation for RNNs



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

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

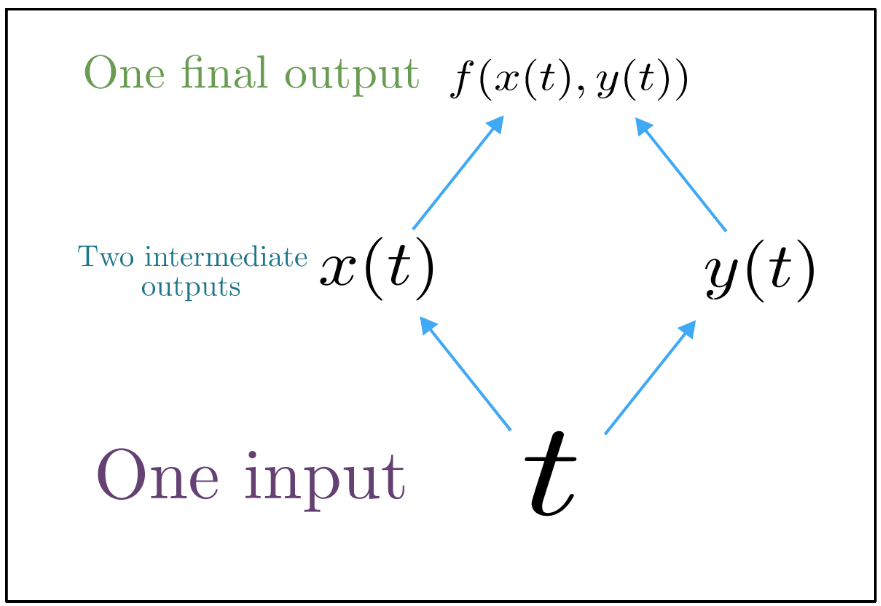
"The gradient w.r.t. a repeated weight is the sum of the gradient w.r.t. each time it appears"

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:

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



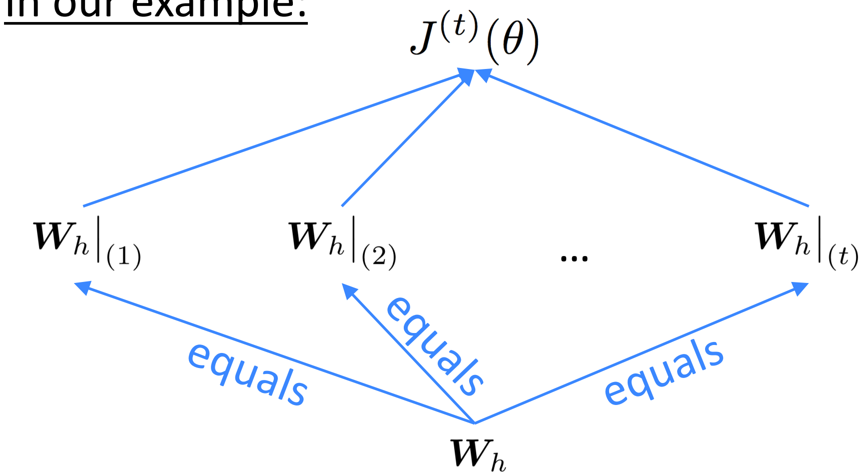
Source:
https://www.khanacademy.org/math/multivariable-calculus/multivariable-derivatives/differentiating-vector-valued-functions/a/multivariable-chain-rule-simple-version

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:

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

<u>In our example:</u>



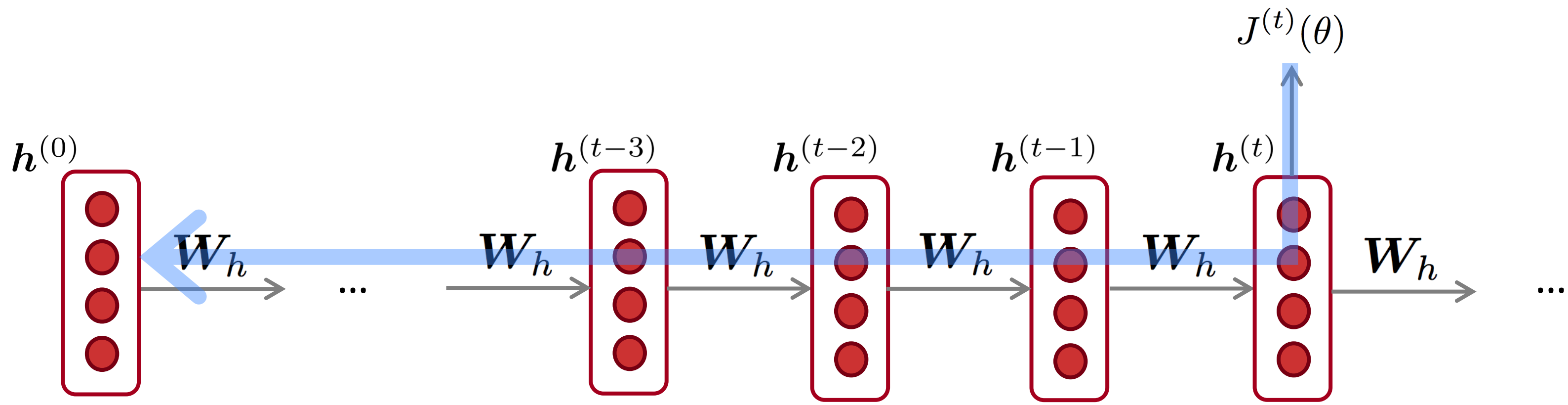
<u>Apply the multivariable chain rule:</u>

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

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

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

Backpropagation for RNNs



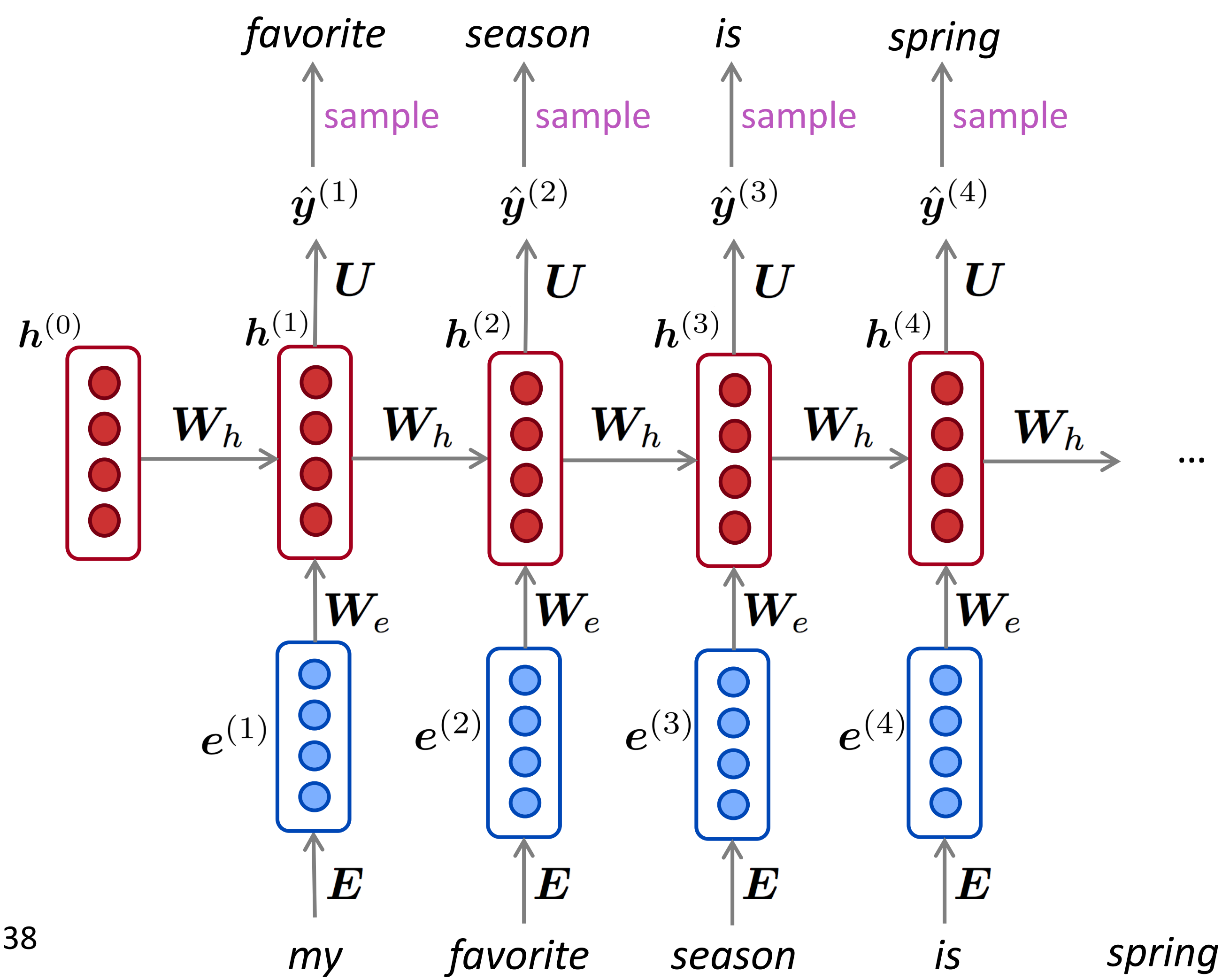
$$\frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} = \boxed{\sum_{i=1}^{t} \left. \frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} \right|_{(i)}}$$

Question: How do we calculate this?

Answer: Backpropagate over timesteps *i*=*t*,…,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.

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

Generating text with a RNN Language Model

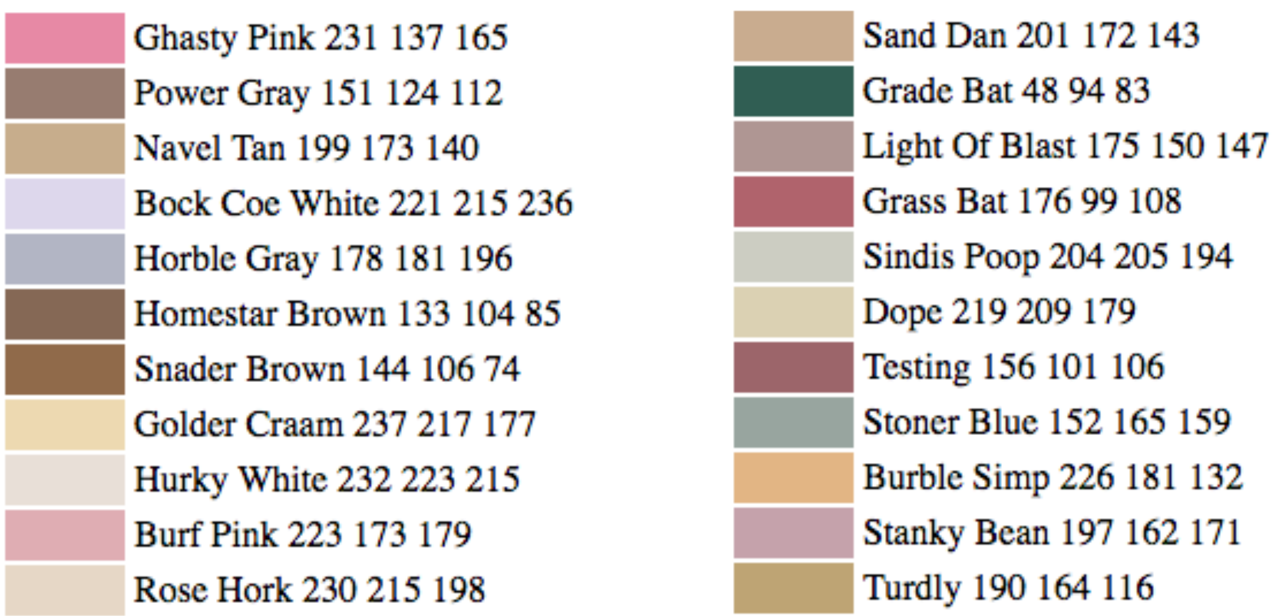
- Let's have some fun!
- You can train a RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on *Seinfeld* scripts:


```
He slams his hand on the door. KRAMER enters dancing with
garbage.

                    KRAMER
          Hey hey hey, great idea for a big
          sponge: Make it so large you think
          it's got a fat clock in the
          middle.

                    JERRY
              (takes off his bones)
          Kramer, do you have a fun
          flashback to do?
```

Source: https://www.avclub.com/a-bunch-of-comedy-writers-teamed-up-with-a-computer-to-1818633242

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.
- (character-level) RNN-LM trained on paint colors:



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

Evaluating Language Models

- The traditional evaluation metric for Language Models is perplexity.

$$\mathrm{PP} = \underbrace{\prod_{t=1}^{T} \left(\frac{1}{\sum_{j=1}^{|V|} y_j^{(t)} \cdot \hat{y}_j^{(t)}} \right)}_{\text{Inverse probability of dataset}}{}^{1/T}$$

Normalized by number of words

- Lower is better!
- In Assignment 2 you will show that minimizing perplexity and minimizing the loss function are equivalent.

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/

Why should we care about Language Modeling?

- Language Modeling is a subcomponent of other NLP systems:
  - Speech recognition
    - Use a LM to generate transcription, conditioned on audio
  - Machine Translation
    - Use a LM to generate translation, conditioned on original text
  - Summarization
    - Use a LM to generate summary, conditioned on original text

These systems are called *conditional Language Models*

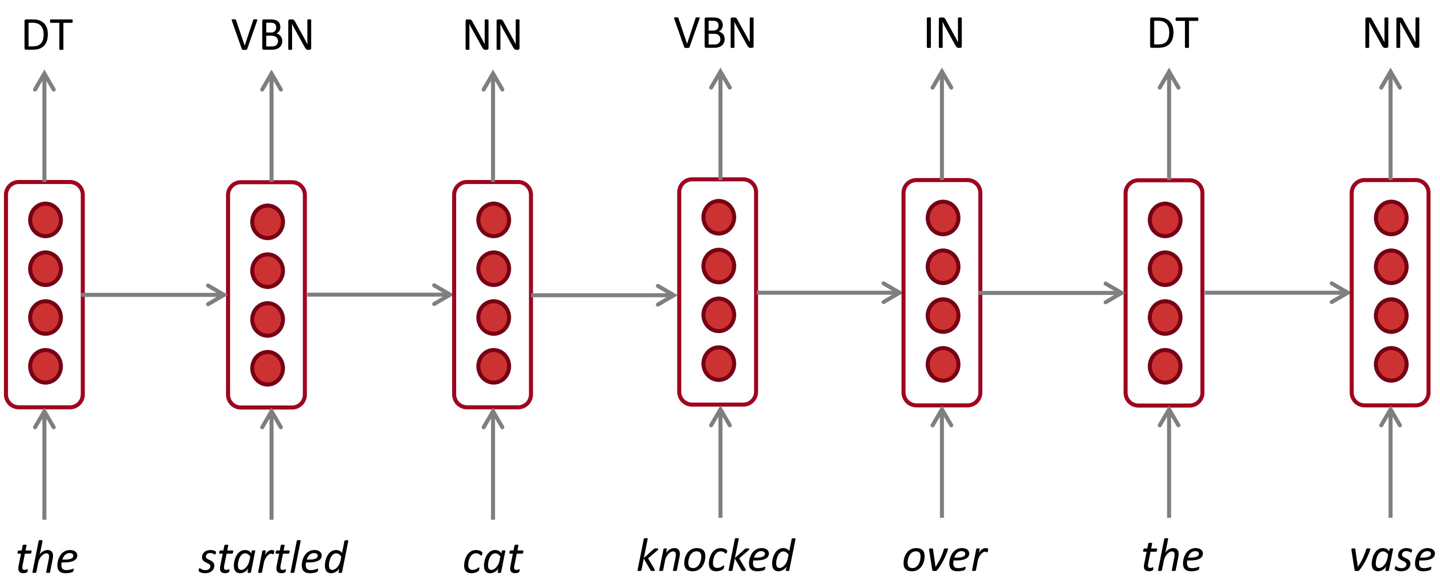
- Language Modeling is a benchmark task that helps us measure our progress on understanding language

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.
- But RNNs are useful for much more!

RNNs can be used for tagging

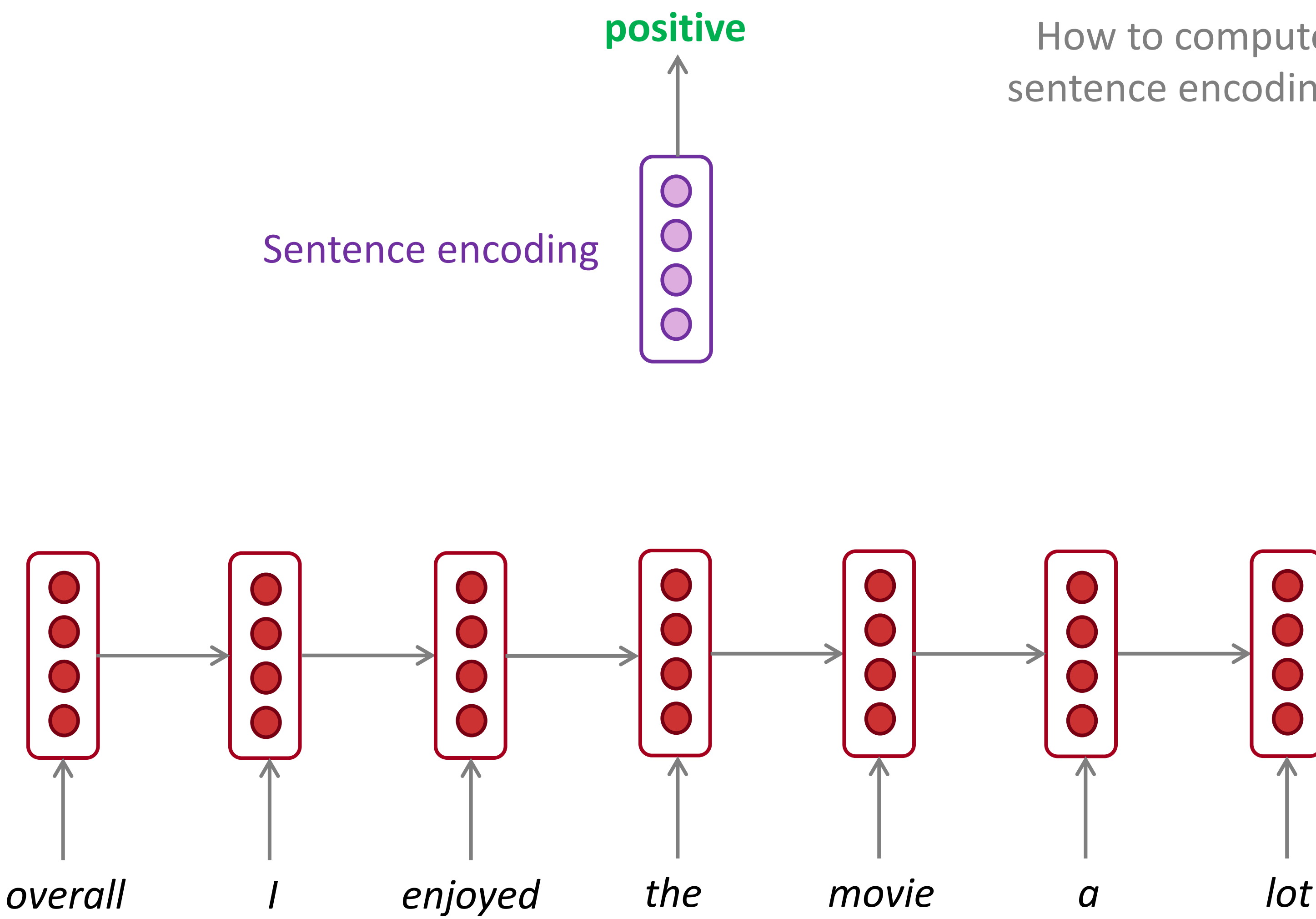
e.g. part-of-speech tagging, named entity recognition



RNNs can be used for sentence classification

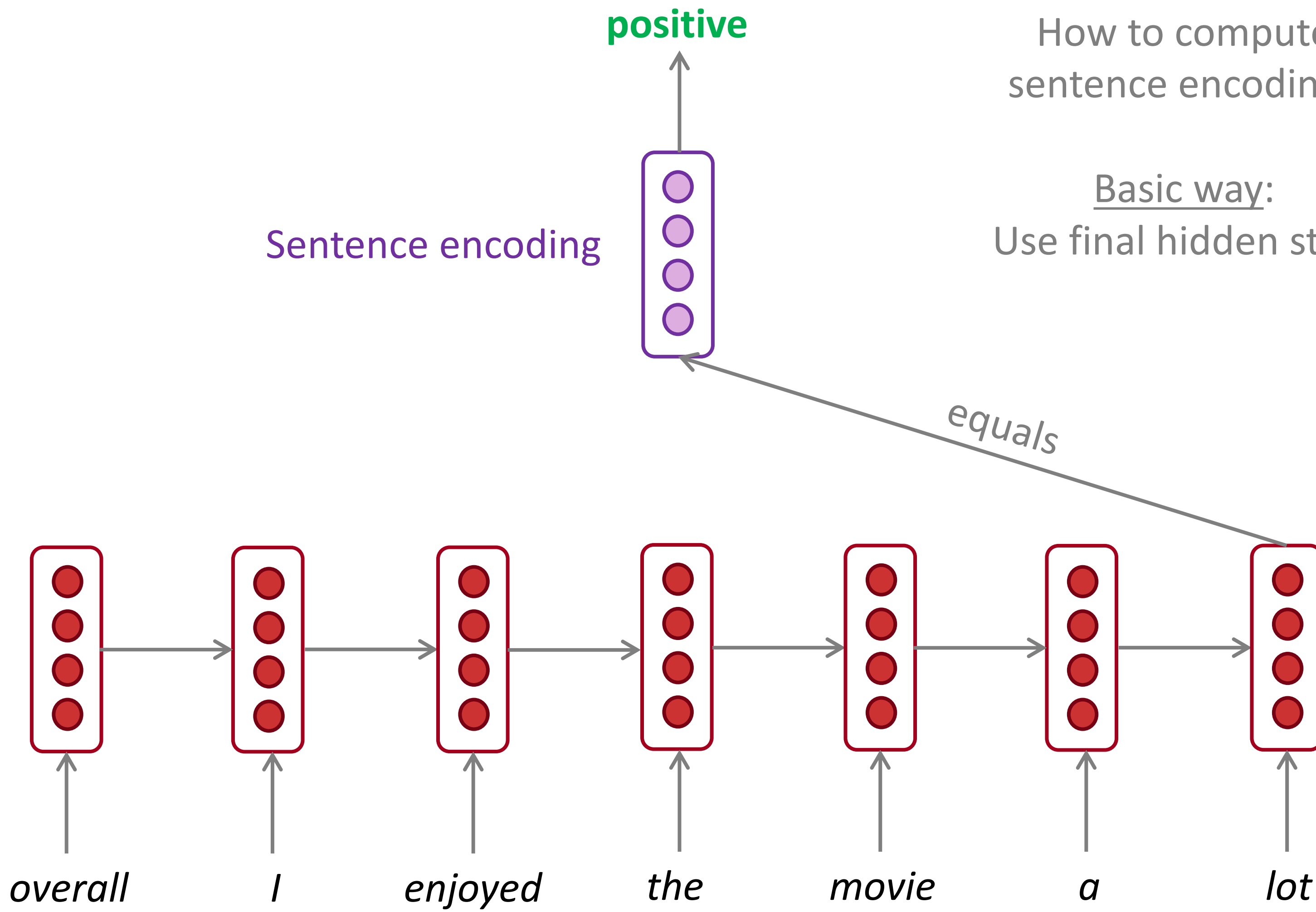
e.g. sentiment classification

How to compute sentence encoding?



RNNs can be used for sentence classification

e.g. sentiment classification

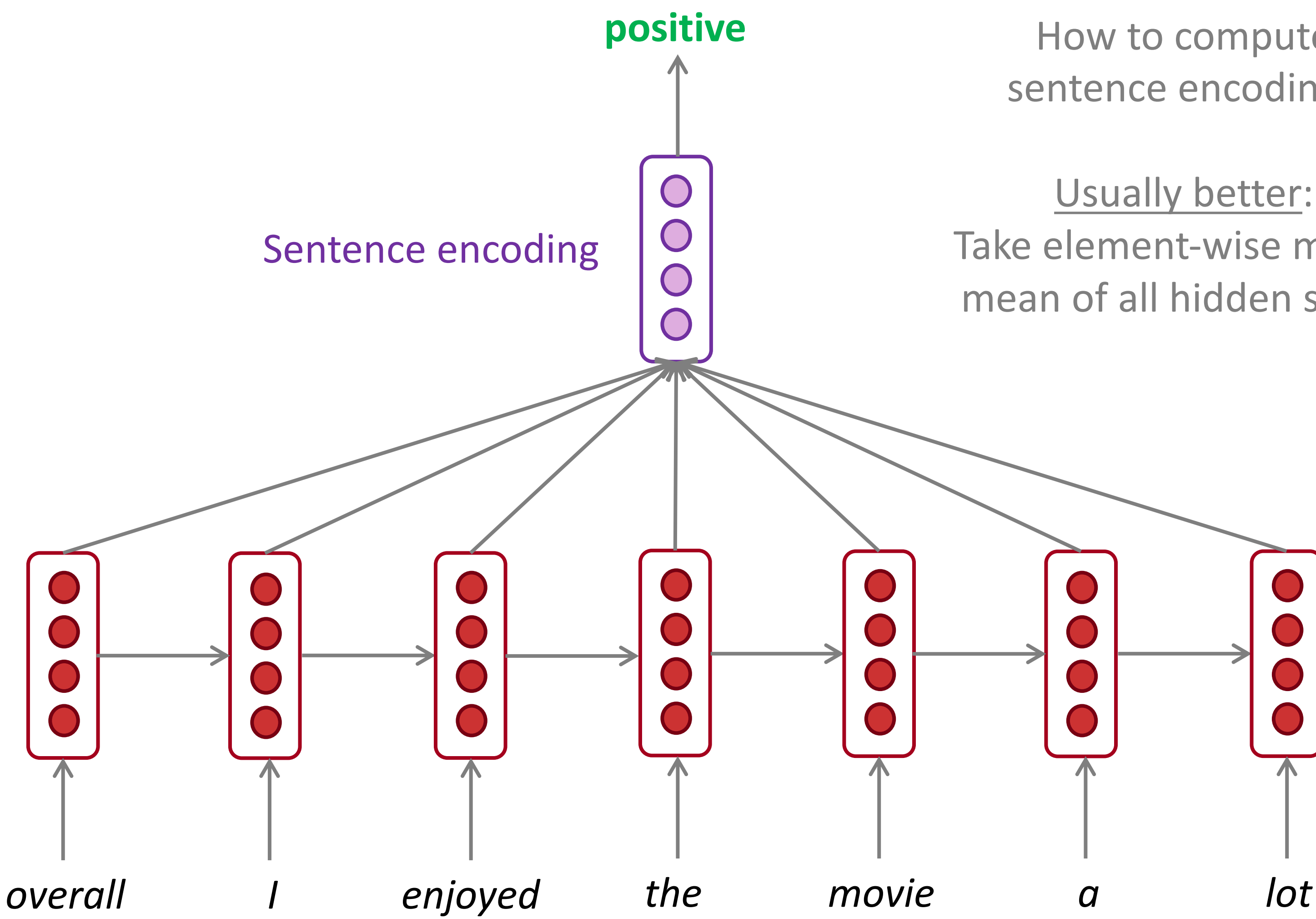


How to compute sentence encoding?

Basic way:
Use final hidden state

RNNs can be used for sentence classification

e.g. sentiment classification

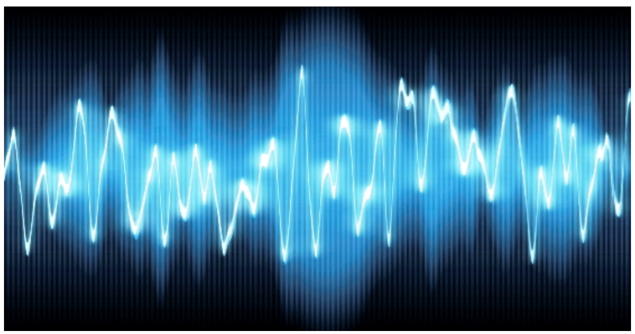


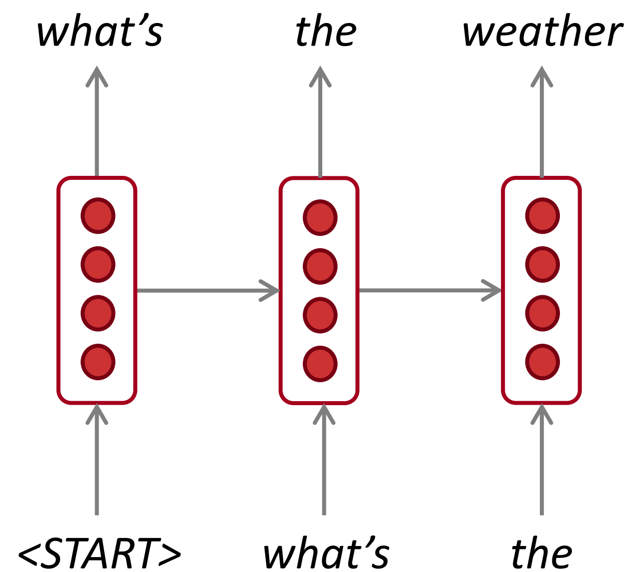
How to compute sentence encoding?

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

RNNs can be used to generate text

e.g. speech recognition, machine translation, summarization

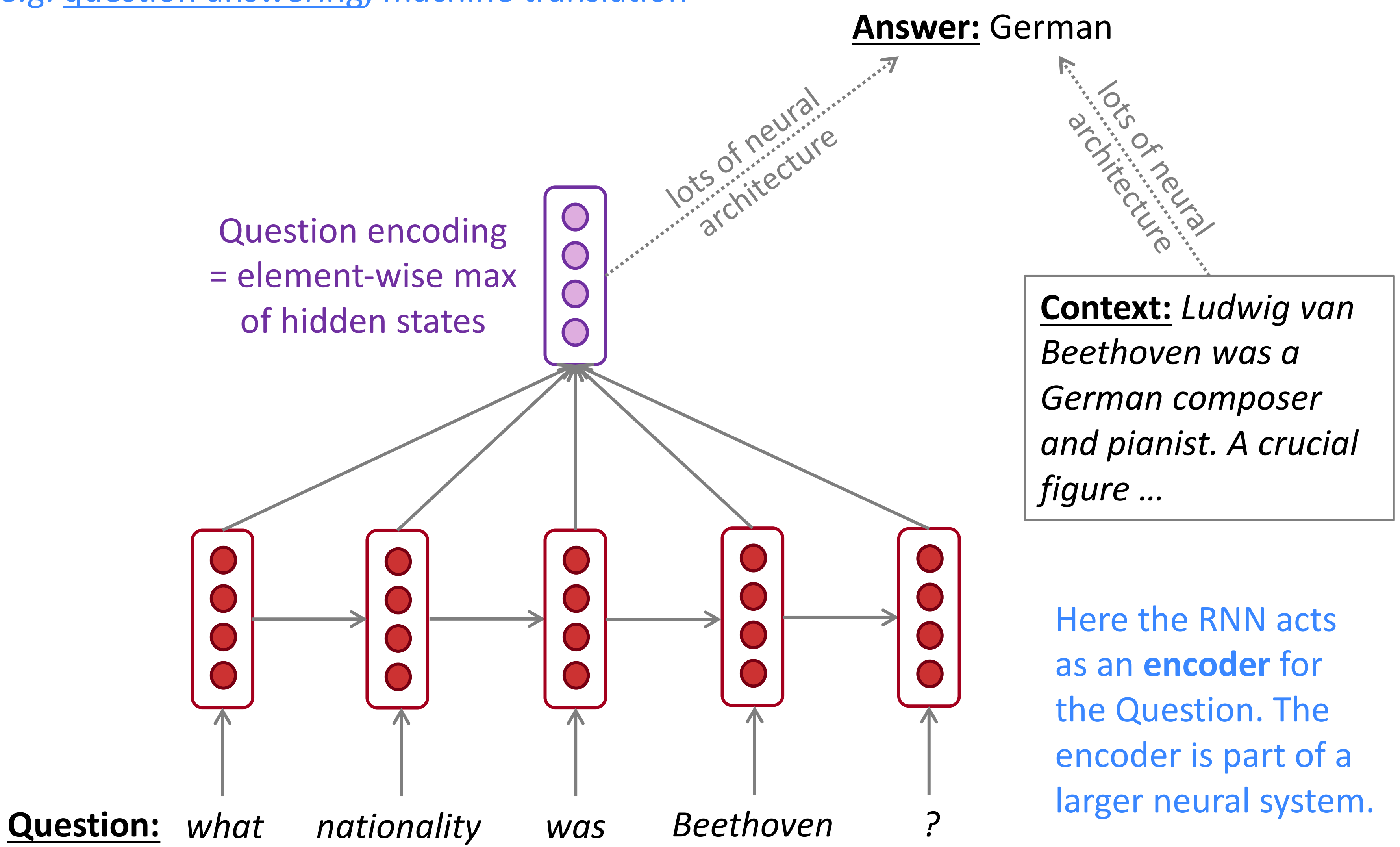




Remember: these are called "conditional language models".
We'll see Machine Translation in much more detail later.

RNNs can be used as an encoder module

e.g. question answering, machine translation



Here the RNN acts as an **encoder** for the Question. The encoder is part of a larger neural system.

A note on terminology

RNN described in this lecture = "vanilla RNN" 

<u>Next lecture:</u> You will learn about other RNN flavors

like GRU  and LSTM  and multi-layer RNNs 

<u>By the end of the course:</u> 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