Natural Language Processing with Deep Learning

CS224N/Ling284

Diyi Yang

Lecture 4: Language Models and Recurrent Neural Networks

Lecture Plan

Lecture 4: Language modeling + RNNs

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

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

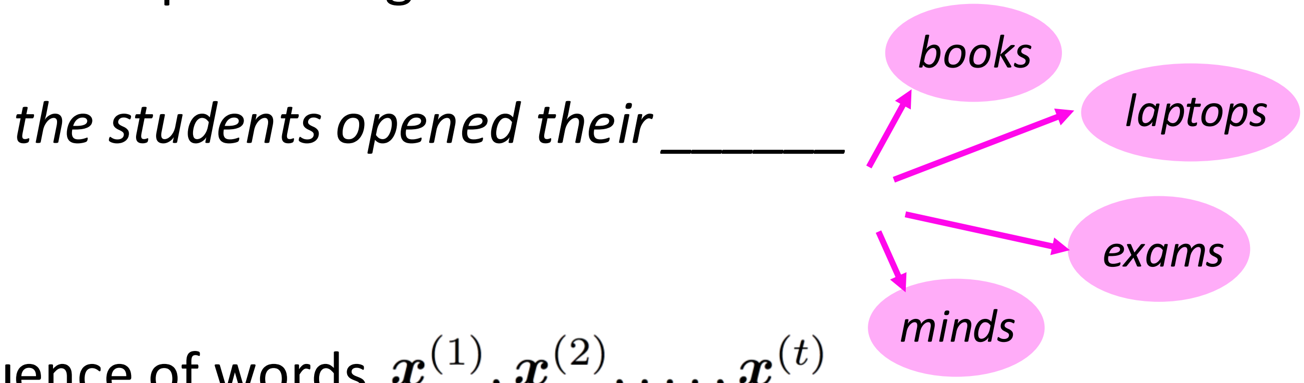
↓ **motivates**

2. Language models with neural nets: **Recurrent Neural Networks (RNNs)** (25 mins)
3. Problems with RNNs: exploding and vanishing gradients (20 mins)
4. Machine translation (10 mins)

Reminder: Assignment 2 – Due Jan 22, Thursday

1. 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**

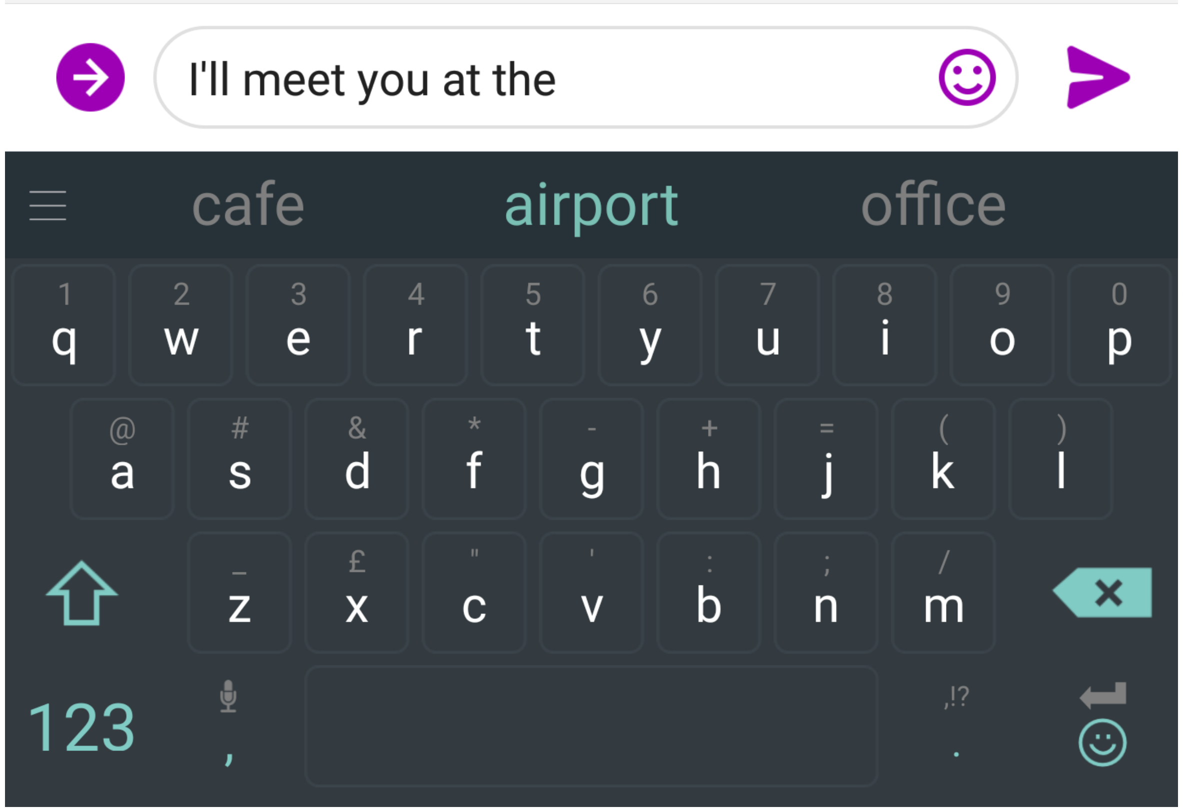
Language Modeling

- 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 $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$, then the probability of this text (according to the Language Model) is:

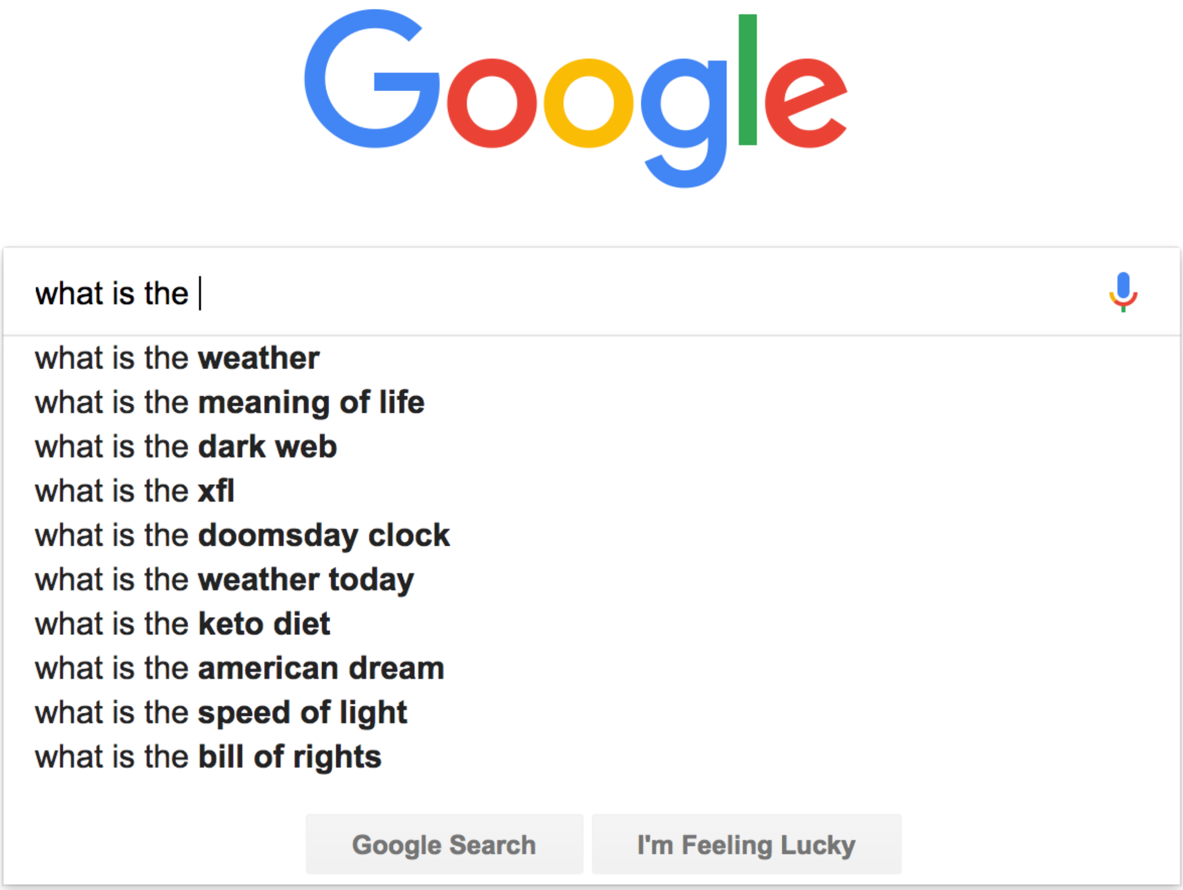
$$\begin{aligned} P(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}) &= P(\boldsymbol{x}^{(1)}) \times P(\boldsymbol{x}^{(2)} |\, \boldsymbol{x}^{(1)}) \times \cdots \times P(\boldsymbol{x}^{(T)} |\, \boldsymbol{x}^{(T-1)}, \ldots, \boldsymbol{x}^{(1)}) \\ &= \prod_{t=1}^{T} \underbrace{P(\boldsymbol{x}^{(t)} |\, \boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(1)})}_{\text{This is what our LM provides}} \end{aligned}$$

You use Language Models every day!



I'll meet you at the

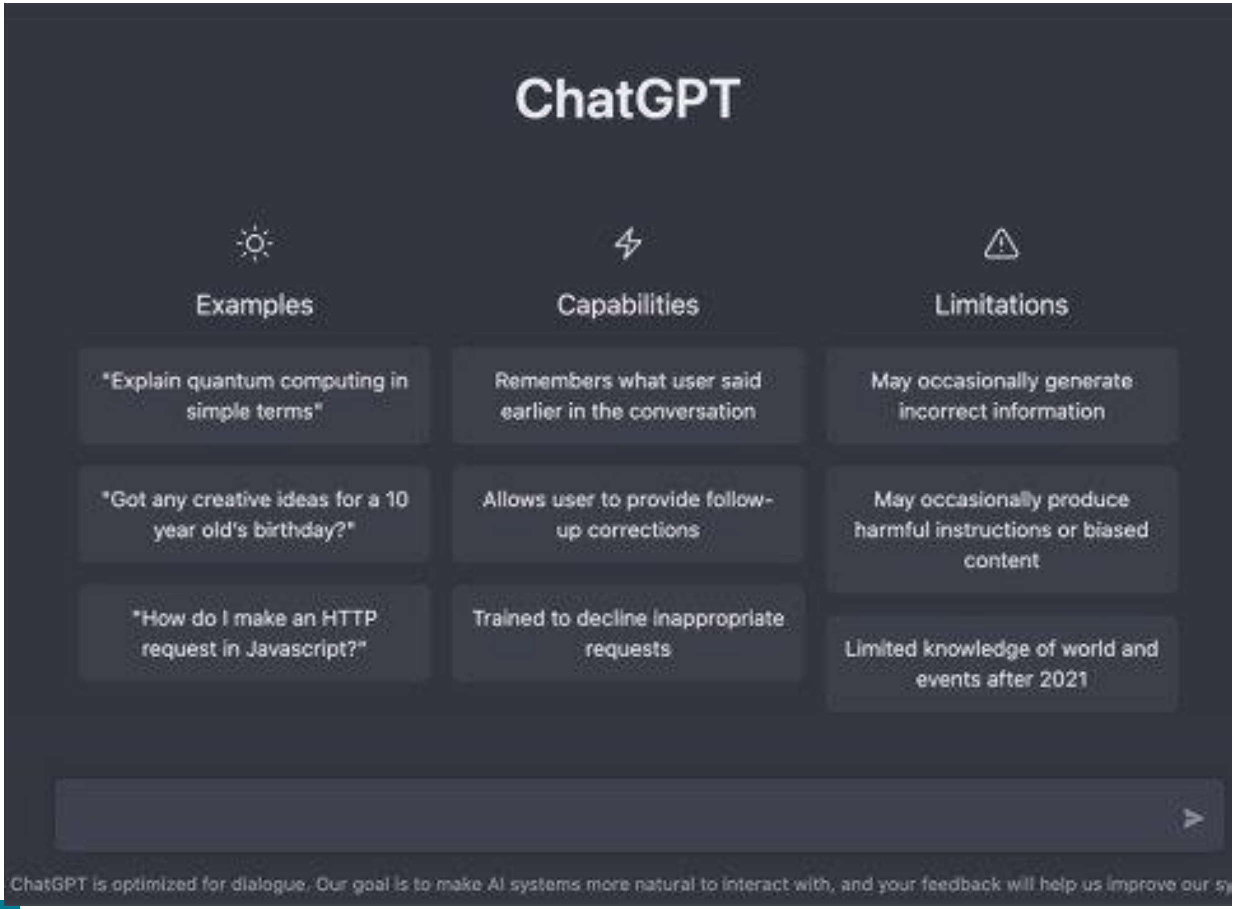
cafe
airport
office
1 q 2 w 3 e 4 r 5 t 6 y 7 u 8 i 9 o 0 p
@ a # s & d * f - g + h = j (k) l
_ z £ x " c ' v : b ; n / m
123
,
,!?
.



what is the

what is the **weather**
what is the **meaning of life**
what is the **dark web**
what is the **xfl**
what is the **doomsday clock**
what is the **weather today**
what is the **keto diet**
what is the **american dream**
what is the **speed of light**
what is the **bill of rights**

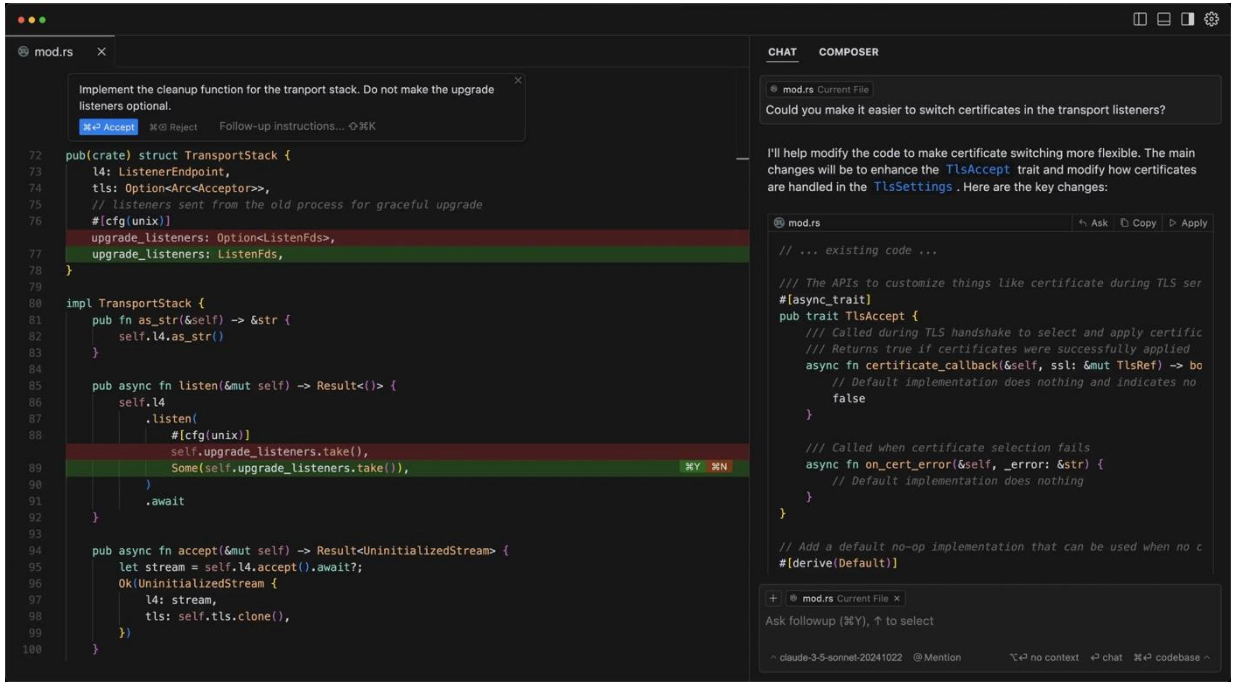
You use Language Models every day!



ChatGPT

Examples	Capabilities	Limitations
"Explain quantum computing in simple terms"	Remembers what user said earlier in the conversation	May occasionally generate incorrect information
"Got any creative ideas for a 10 year old's birthday?"	Allows user to provide follow-up corrections	May occasionally produce harmful instructions or biased content
"How do I make an HTTP request in Javascript?"	Trained to decline inappropriate requests	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 sy



Implement the cleanup function for the tranport stack. Do not make the upgrade listeners optional.

```
pub(crate) struct TransportStack {
    l4: ListenerEndpoint,
    tls: Option<Arc<Acceptor>>,
    // listeners sent from the old process for graceful upgrade
    #[cfg(unix)]
    upgrade_listeners: Option<ListenFds>,
    upgrade_listeners: ListenFds,
}

impl TransportStack {
    pub fn as_str(&self) -> &str {
        self.l4.as_str()
    }

    pub async fn listen(&mut self) -> Result<()> {
        self.l4
            .listen(
                #[cfg(unix)]
                self.upgrade_listeners.take(),
                Some(self.upgrade_listeners.take()),
            )
            .await
    }

    pub async fn accept(&mut self) -> Result<UninitializedStream> {
        let stream = self.l4.accept().await?;
        Ok(UninitializedStream {
            l4: stream,
            tls: self.tls.clone(),
        })
    }
```

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)]
```

mod.rs Current File

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]

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

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

prob of a n-gram

prob of a (n-1)-gram

$$= \frac{\boxed{P(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\dots,\boldsymbol{x}^{(t-n+2)})}}{\boxed{P(\boldsymbol{x}^{(t)},\dots,\boldsymbol{x}^{(t-n+2)})}} \qquad \text{(definition of conditional prob)}$$

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

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

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

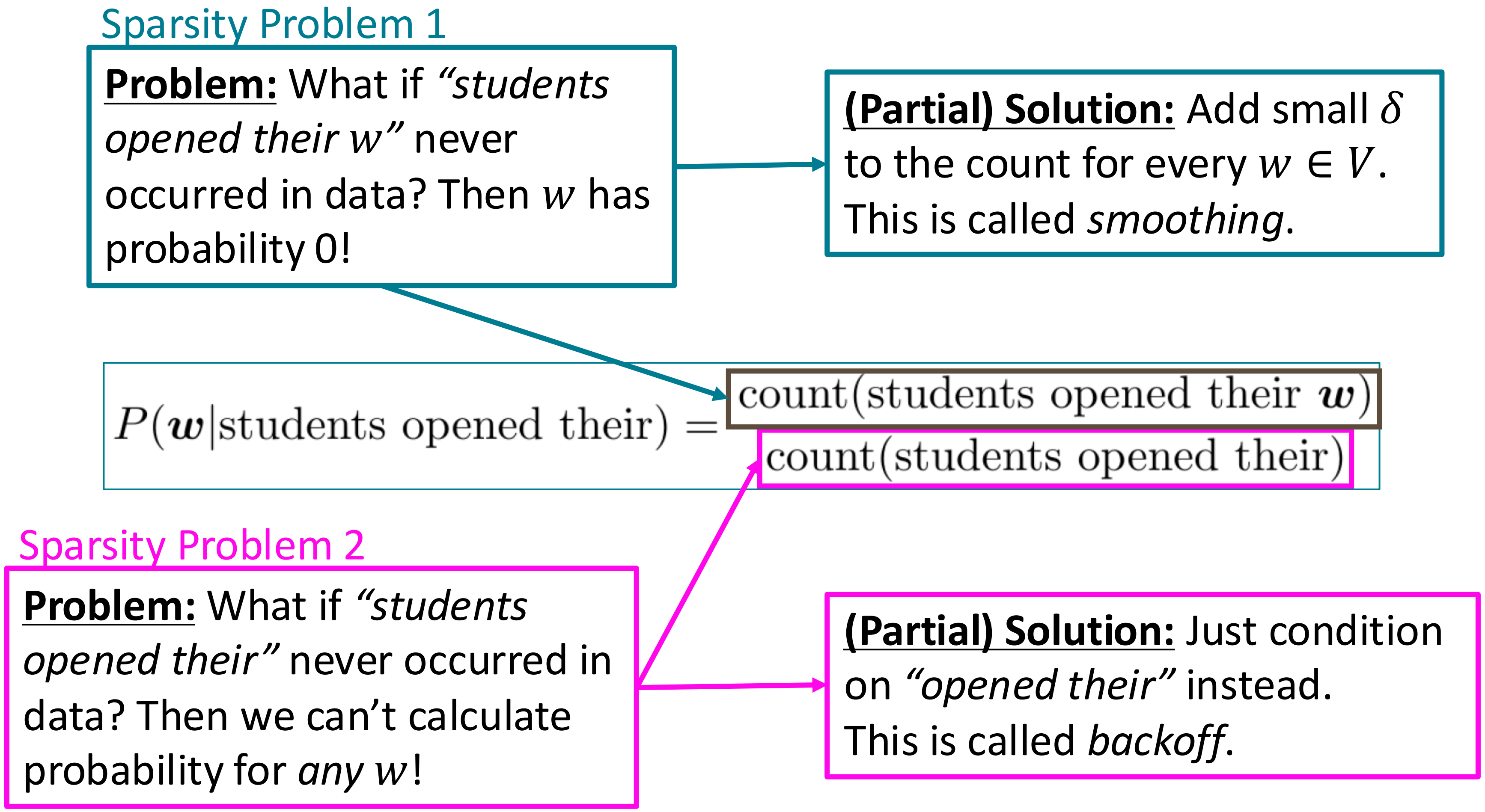
$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

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?

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*.

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

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

<u>Storage</u>: Need to store count for all *n*-grams you saw in the corpus.

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

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
           …
```

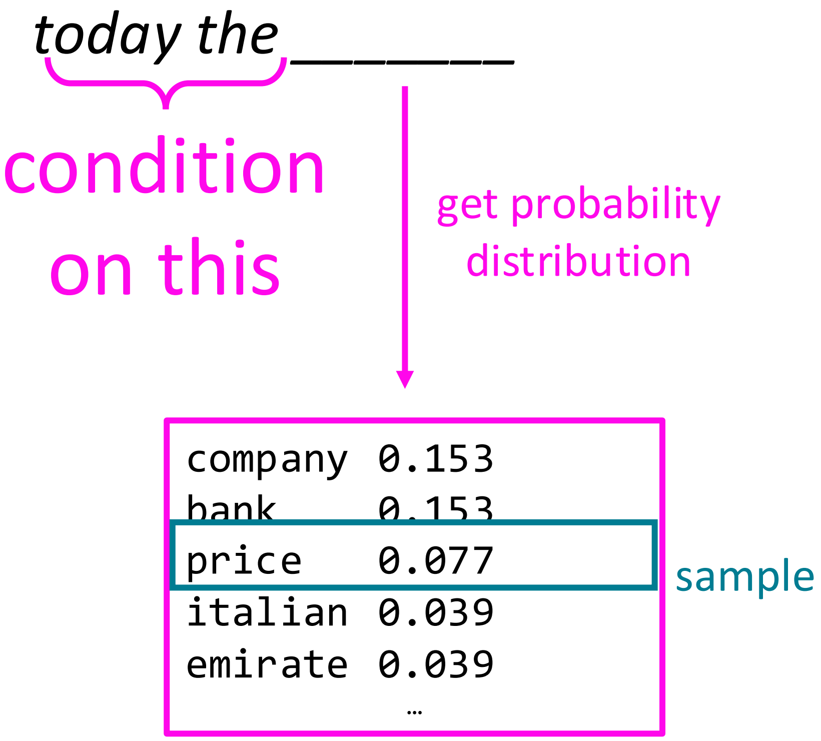
Sparsity problem: not much granularity in the probability distribution

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

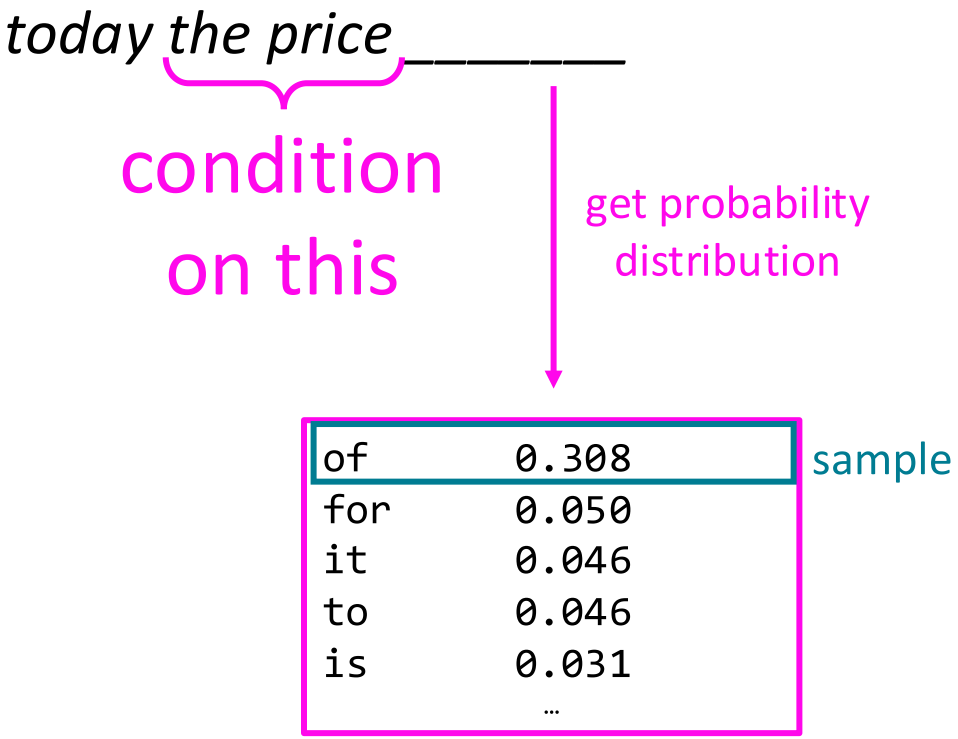
get probability distribution

```
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

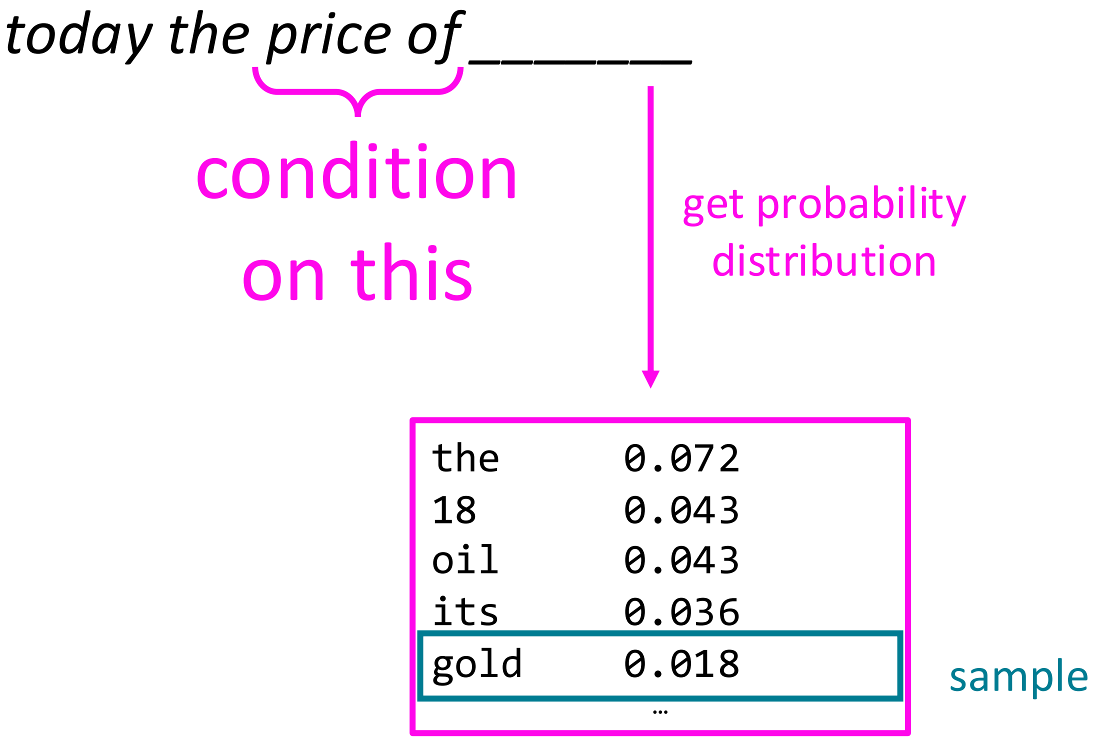
get probability distribution

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 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!

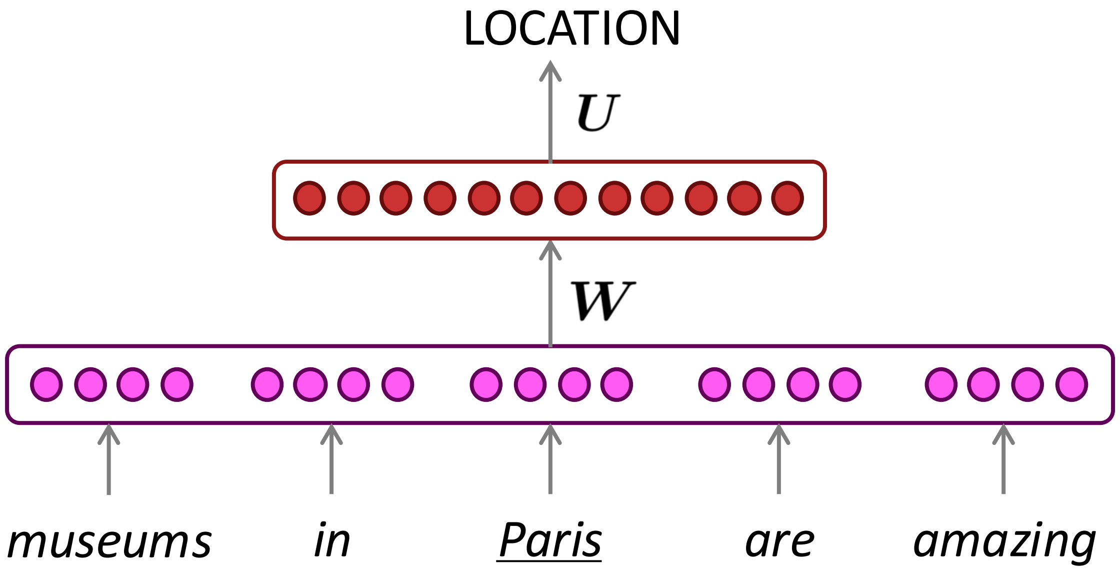
…but **incoherent.** We need to consider more than three words at a time if we want to model language well.

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:



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

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

hidden layer

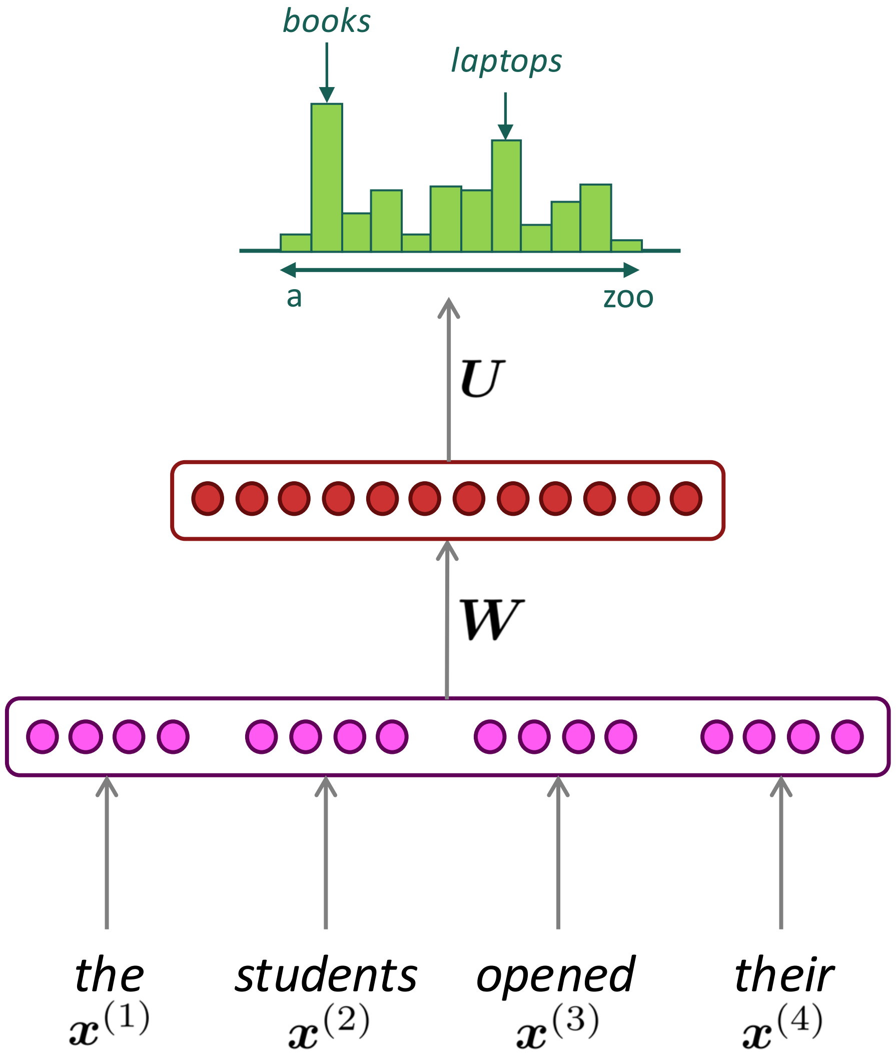
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

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

A fixed-window neural Language Model

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

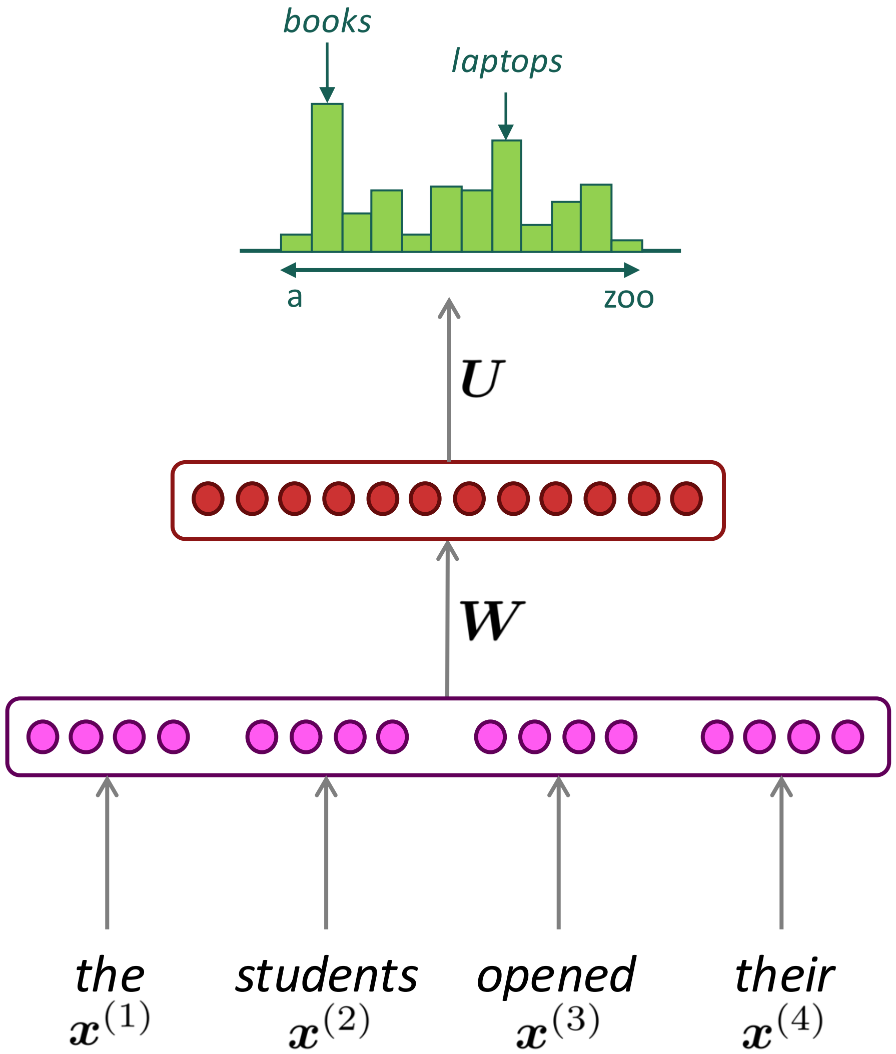
Improvements over *n*-gram LM:

- 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*

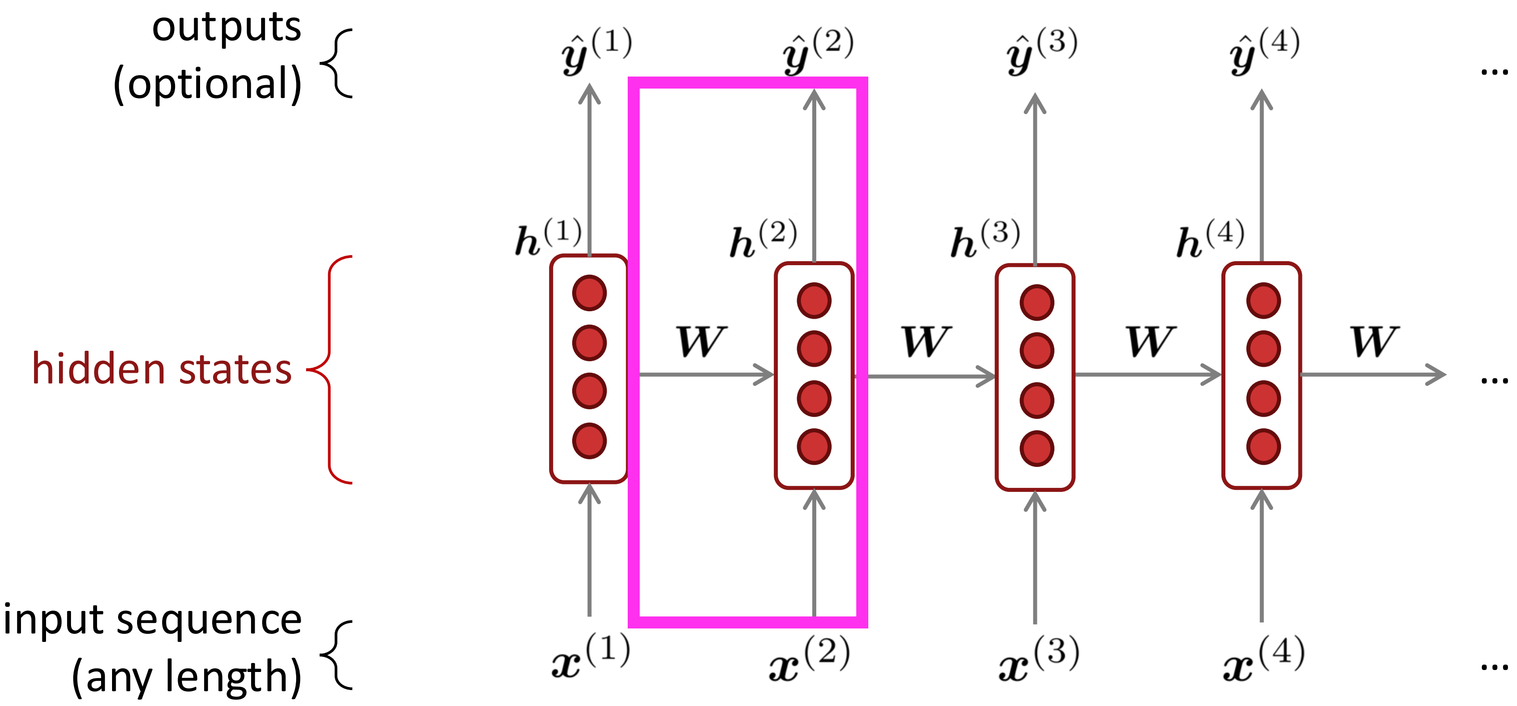


books
laptops
a
zoo
$\boldsymbol{U}$
$\boldsymbol{W}$
the $\boldsymbol{x}^{(1)}$
students $\boldsymbol{x}^{(2)}$
opened $\boldsymbol{x}^{(3)}$
their $\boldsymbol{x}^{(4)}$

2. Recurrent Neural Networks (RNN)

A family of neural architectures

Core idea: Apply the same weights W *repeatedly*



A Simple RNN Language Model

output distribution

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

hidden states

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

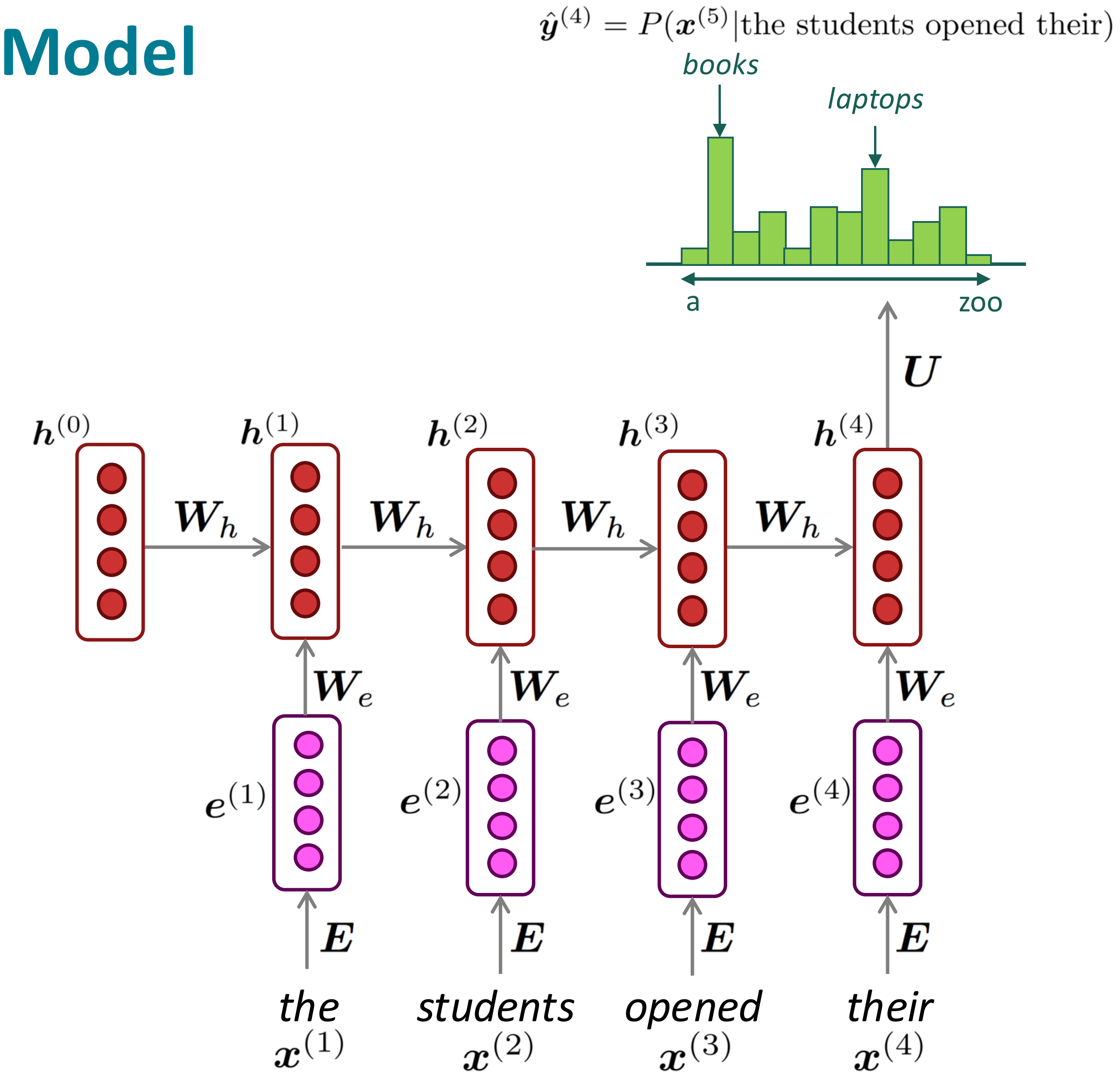
$\boldsymbol{h}^{(0)}$ is the initial hidden state

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 now!*

RNN Language Models

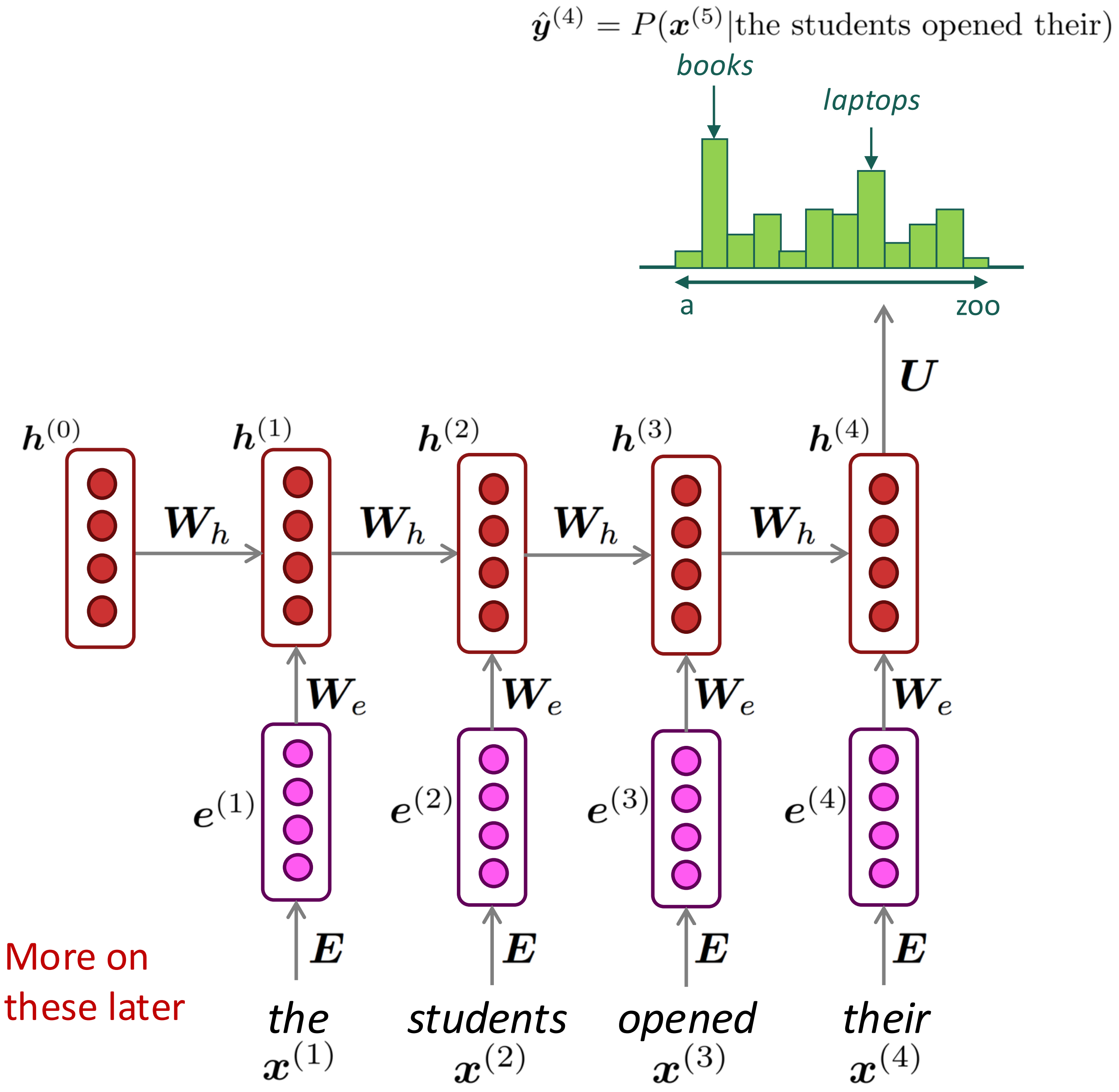
RNN **Advantages**:

- 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 info 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 $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$
- 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

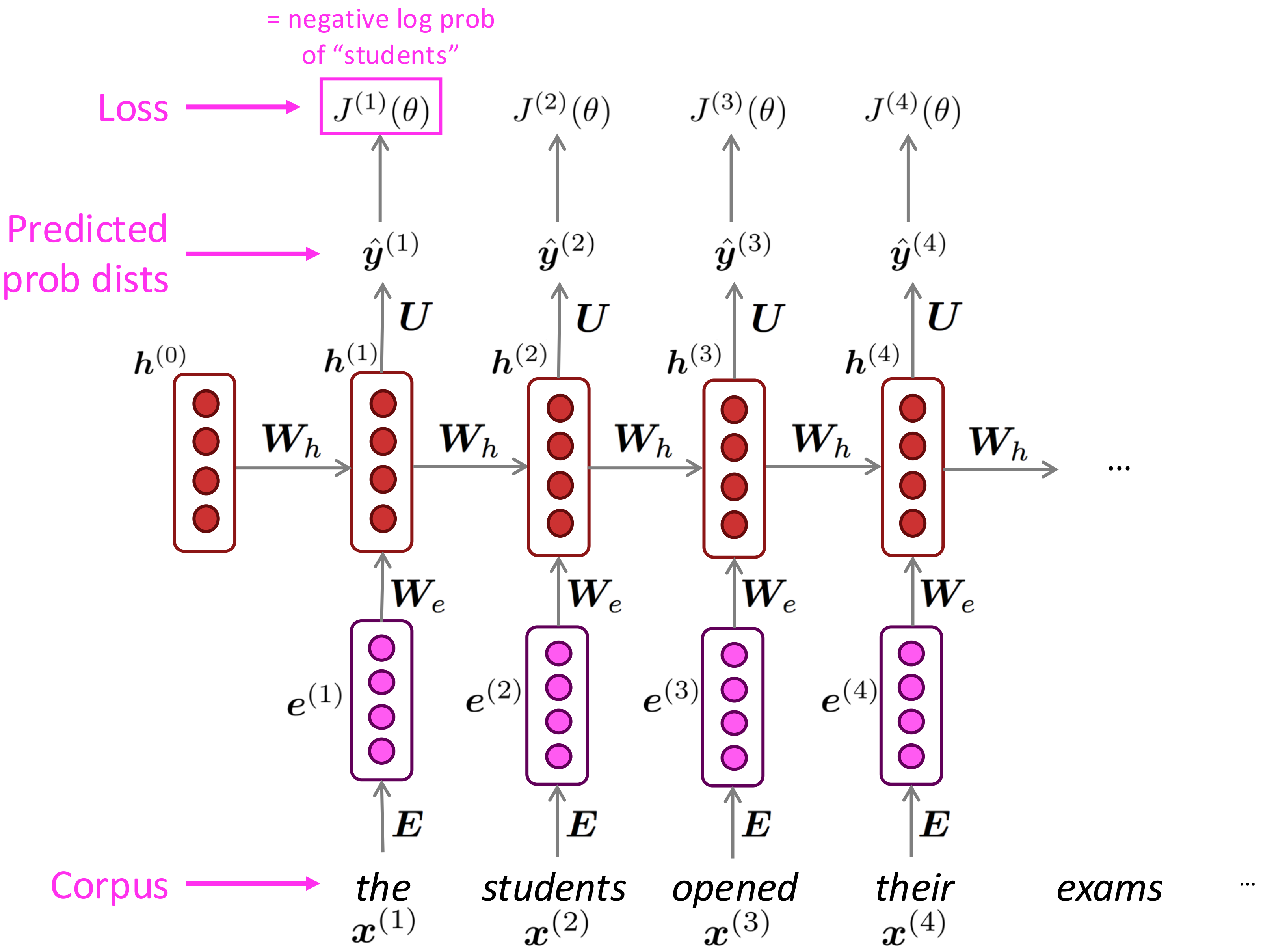
- Loss function on step *t* is cross-entropy between predicted probability distribution $\hat{\boldsymbol{y}}^{(t)}$, and the true next word $\boldsymbol{y}^{(t)}$ (one-hot for $\boldsymbol{x}^{(t+1)}$):

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

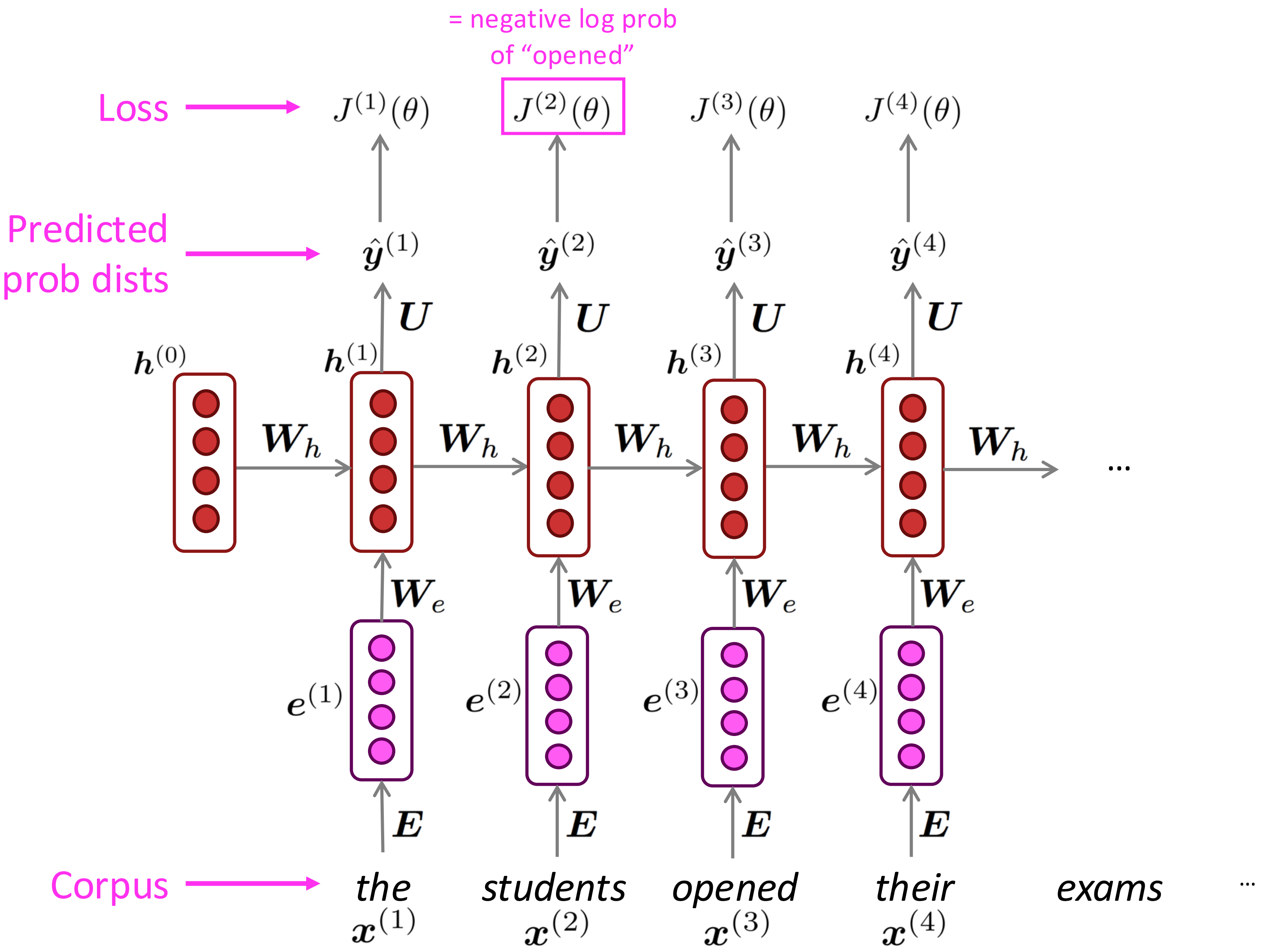
- Average this to get overall loss for entire training set:

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

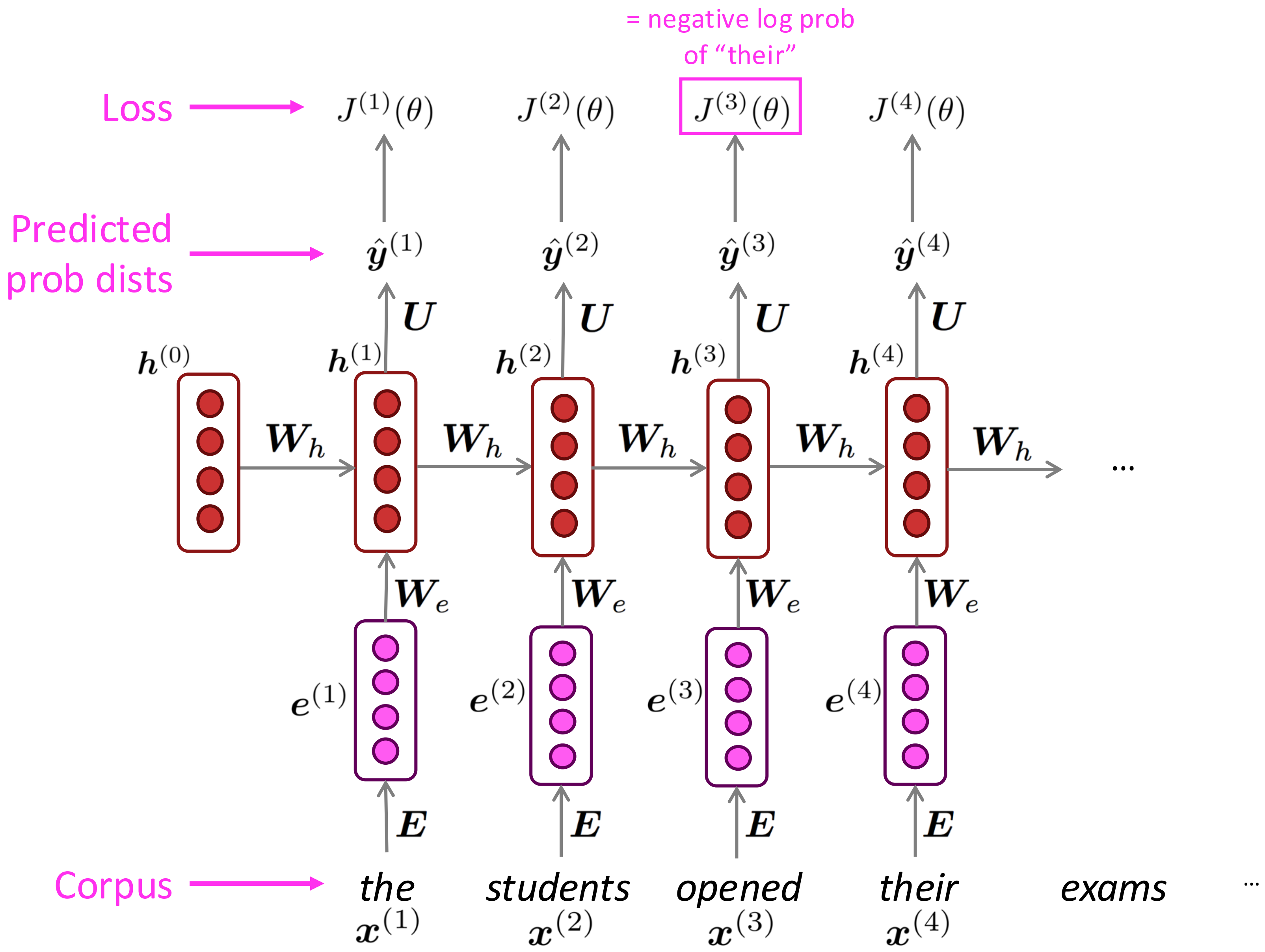
Training an RNN Language Model



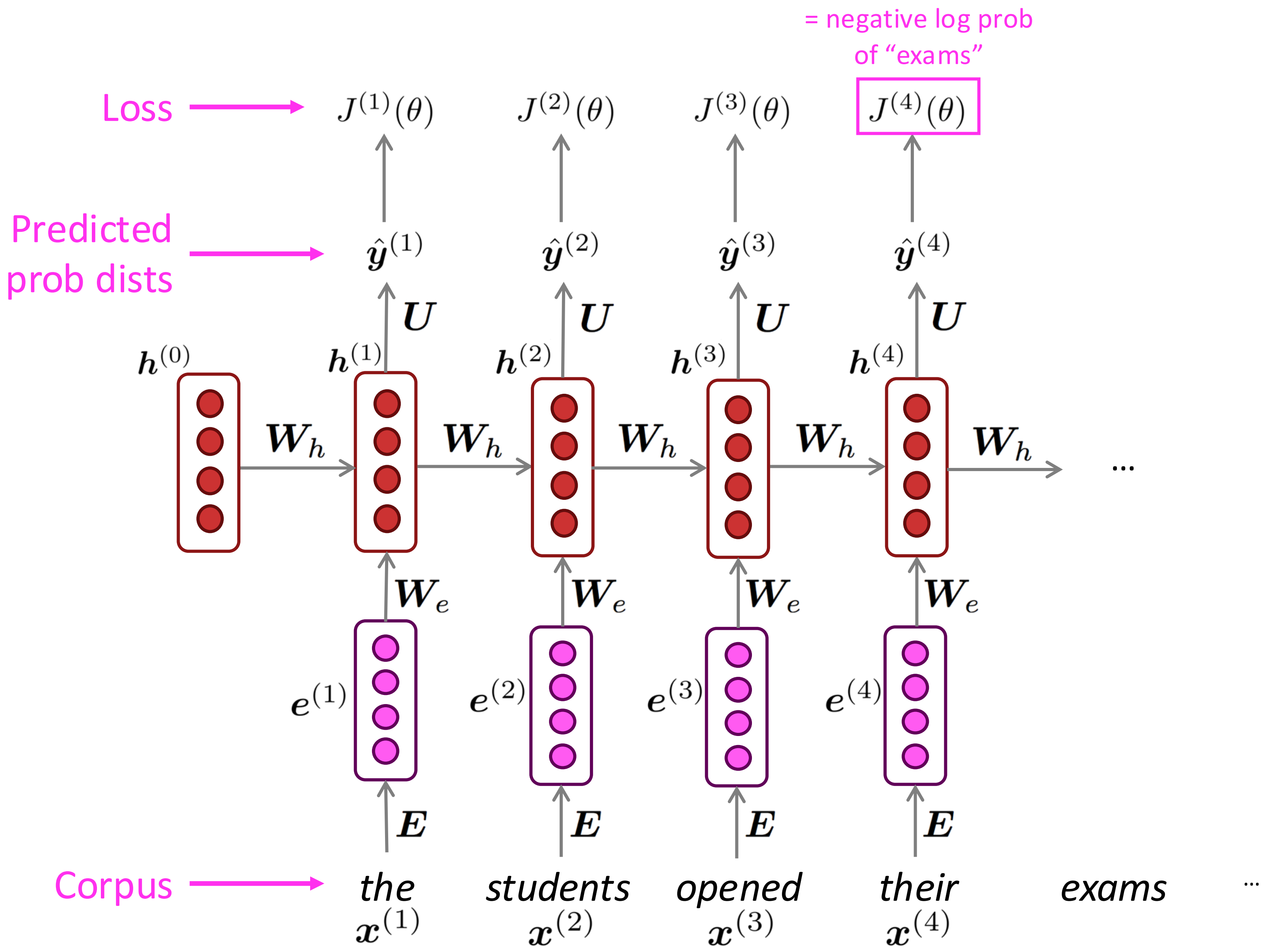
Training an RNN Language Model



Training an RNN Language Model

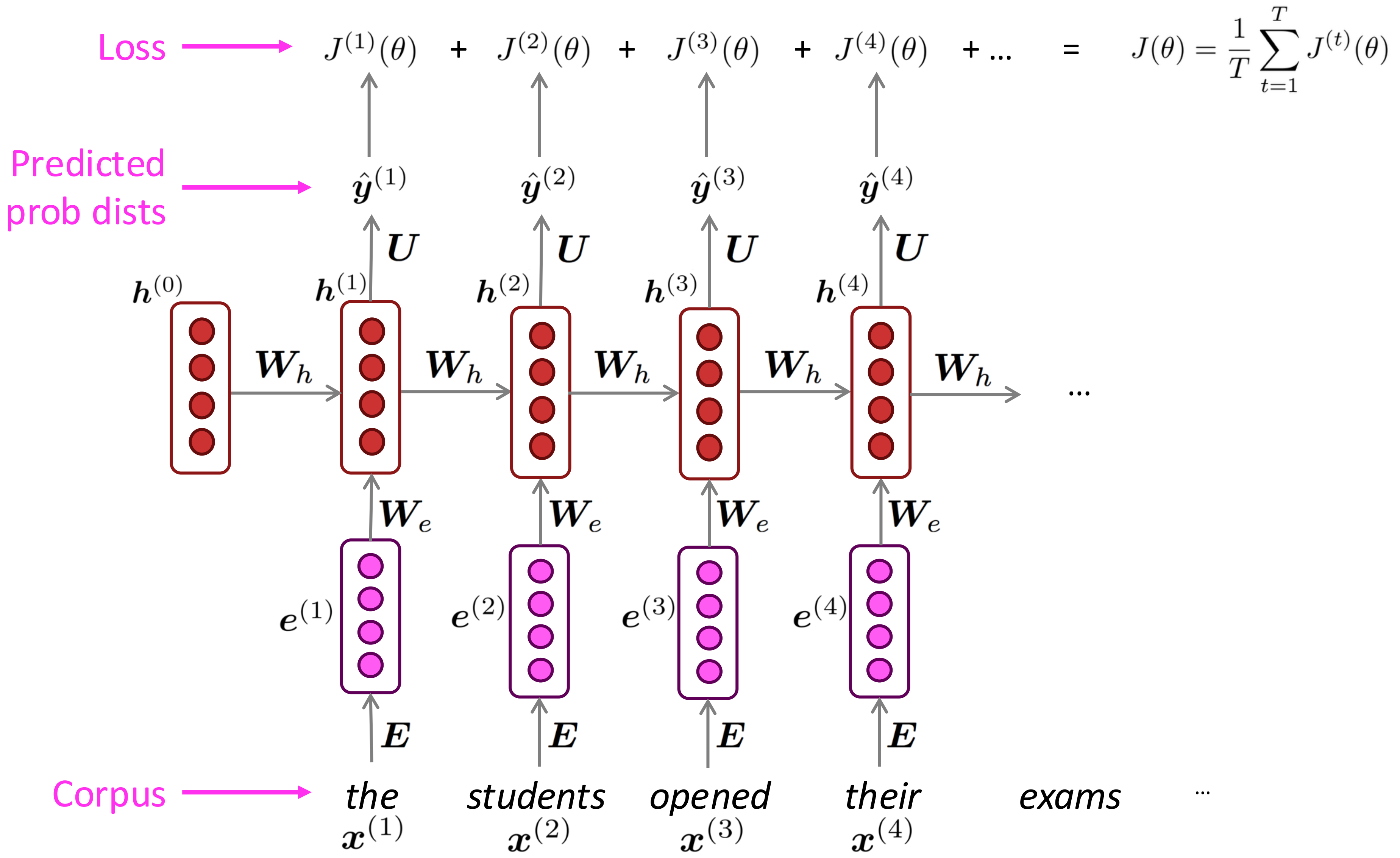


Training an RNN Language Model



Training an RNN Language Model

“Teacher forcing”



Loss → $J^{(1)}(\theta)$ + $J^{(2)}(\theta)$ + $J^{(3)}(\theta)$ + $J^{(4)}(\theta)$ + … = $J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta)$

Training a RNN Language Model

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

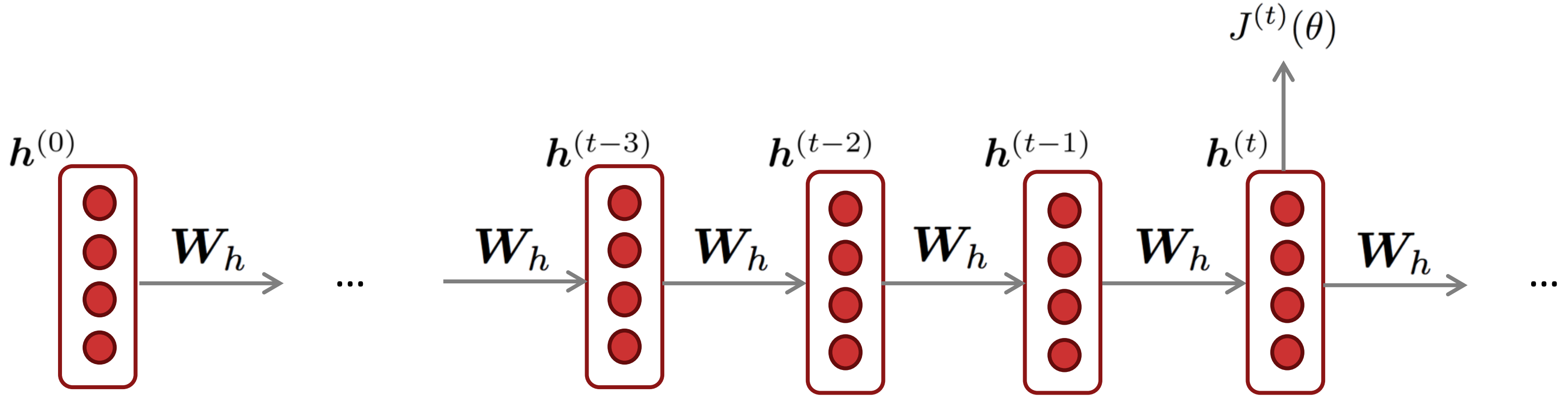
$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J^{(t)}(\theta)$$

- 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 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 $\boldsymbol{W}_h$?

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

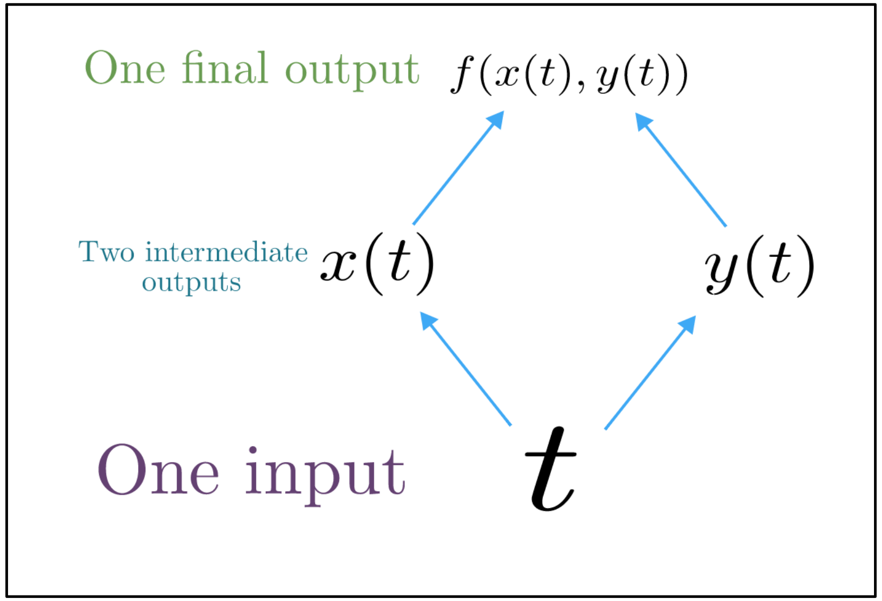
"The gradient w.r.t. a repeated weight is the sum of the gradient w.r.t. each time it appears"

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:

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

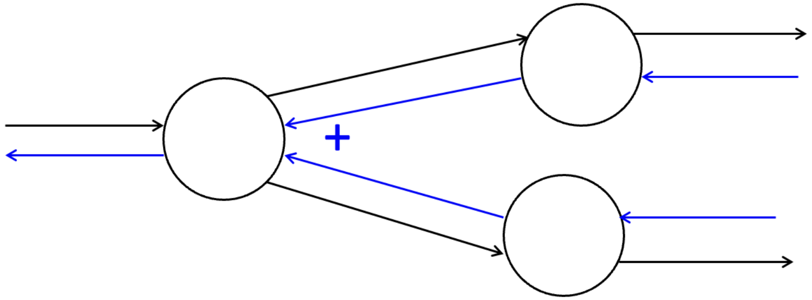


One final output $f(x(t), y(t))$

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

One input t

Gradients sum at outward branches

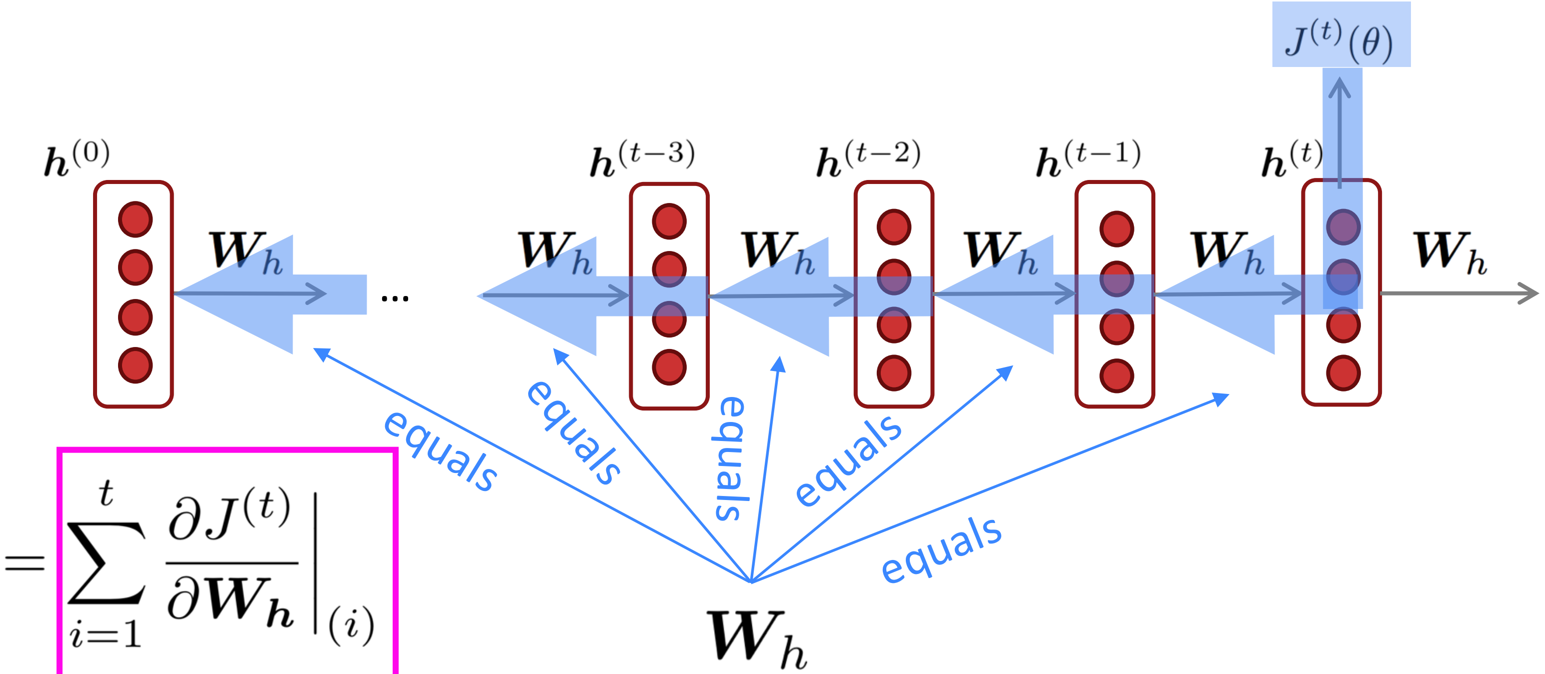


$$a = x + y$$
$$b = \max(y, z)$$
$$f = ab$$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial a}\frac{\partial a}{\partial y} + \frac{\partial f}{\partial b}\frac{\partial b}{\partial y}$$

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

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

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"** [Werbos, P.G., 1988, *Neural Networks* **1**, and others]

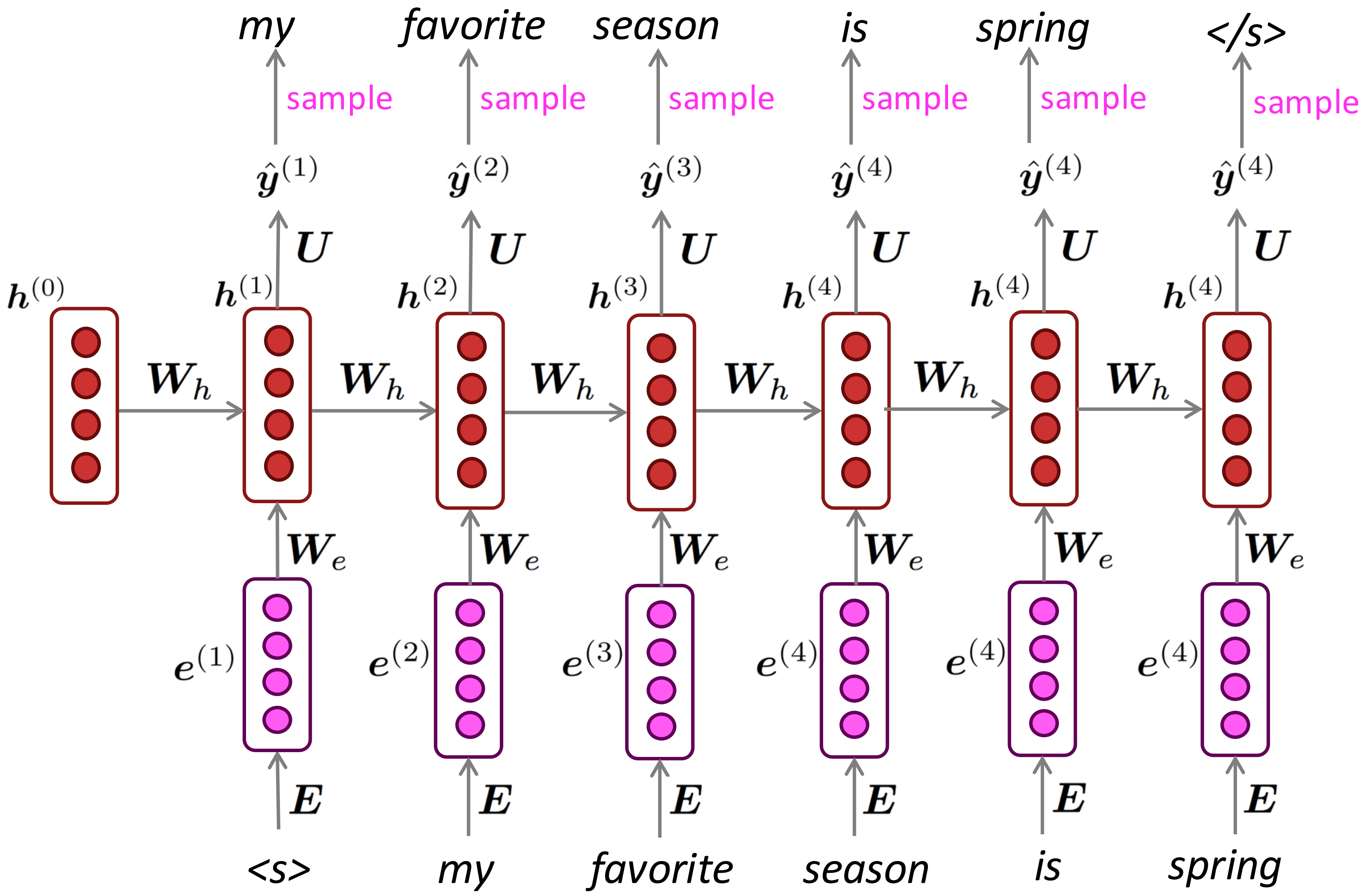
Apply the multivariable chain rule:

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

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

Generating with an RNN Language Model ("Generating roll outs")

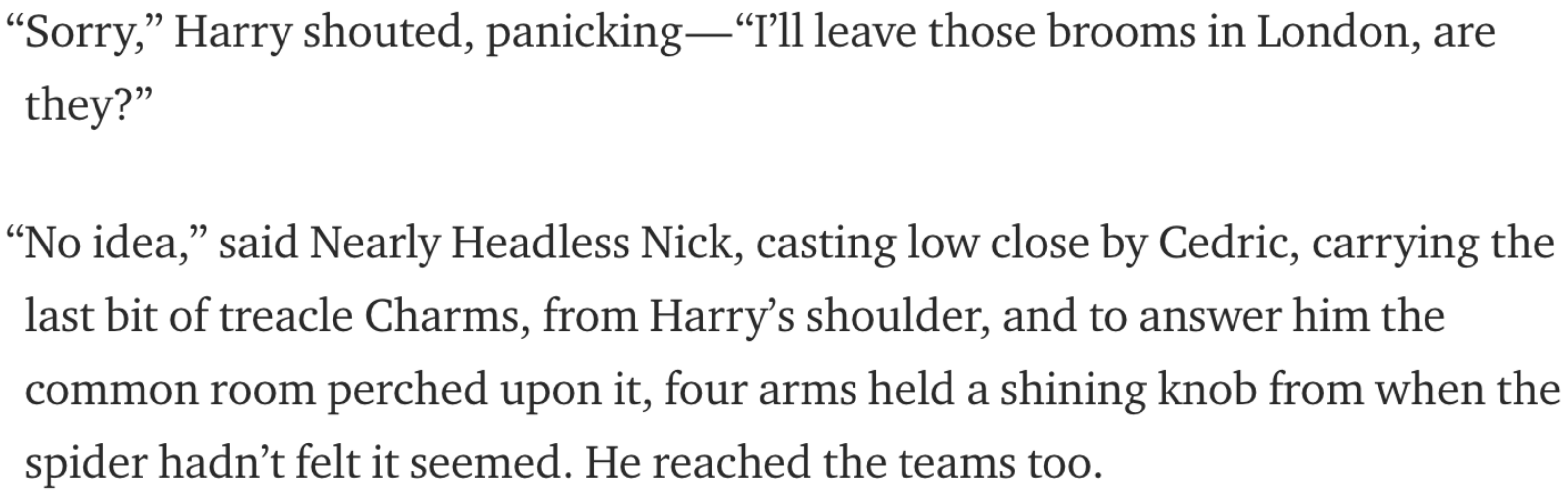
Just like an n-gram Language Model, you can use a RNN Language Model to generate text by repeated sampling. Sampled output becomes next step's input.



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?”

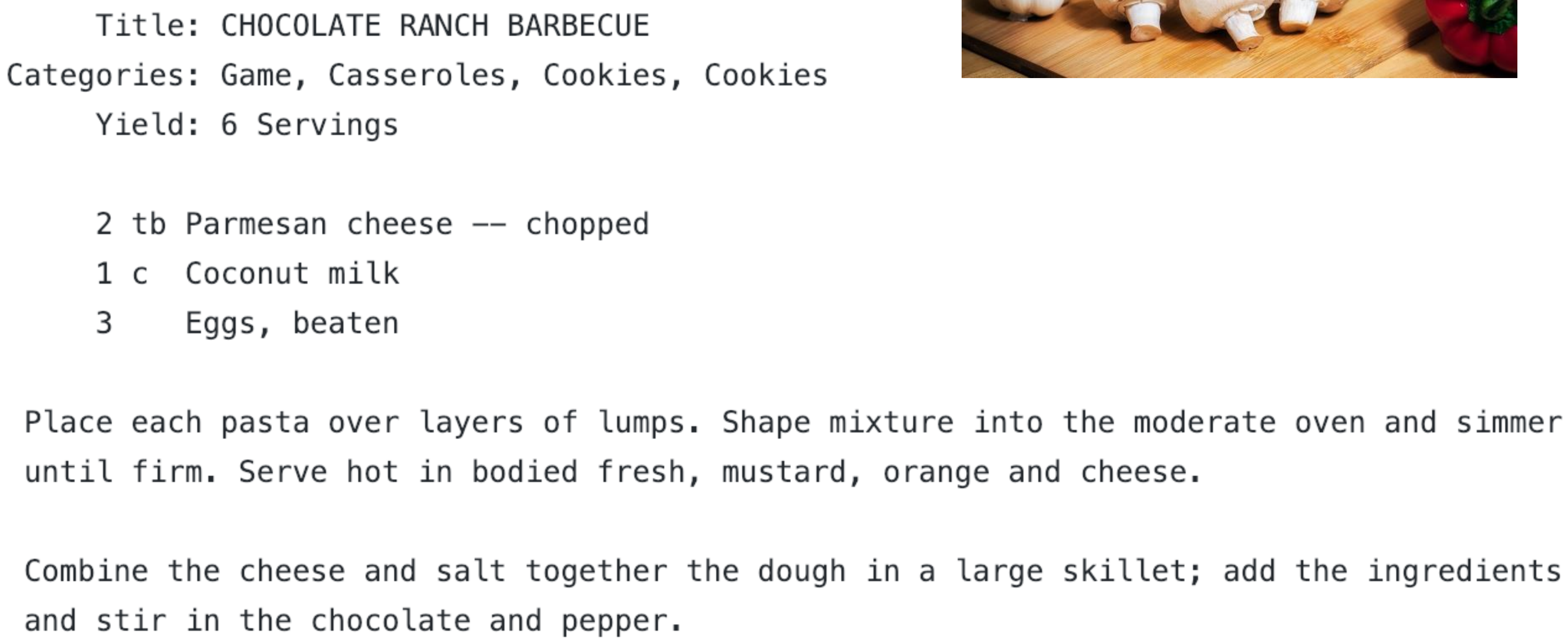
“No idea,” said Nearly Headless Nick, casting low close by Cedric, carrying the last bit of treacle Charms, from Harry’s shoulder, and to answer him the common room perched upon it, four arms held a shining knob from when the spider hadn’t felt it seemed. He reached the teams too.

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

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 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

Evaluating Language Models

- The standard evaluation metric for Language Models is perplexity.

$$\text{perplexity} = \underbrace{\prod_{t=1}^{T} \left(\frac{1}{P_{\mathrm{LM}}(\boldsymbol{x}^{(t+1)} \mid \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})} \right)}_{\text{Inverse probability of corpus, according to Language Model}}{}^{1/T}$$

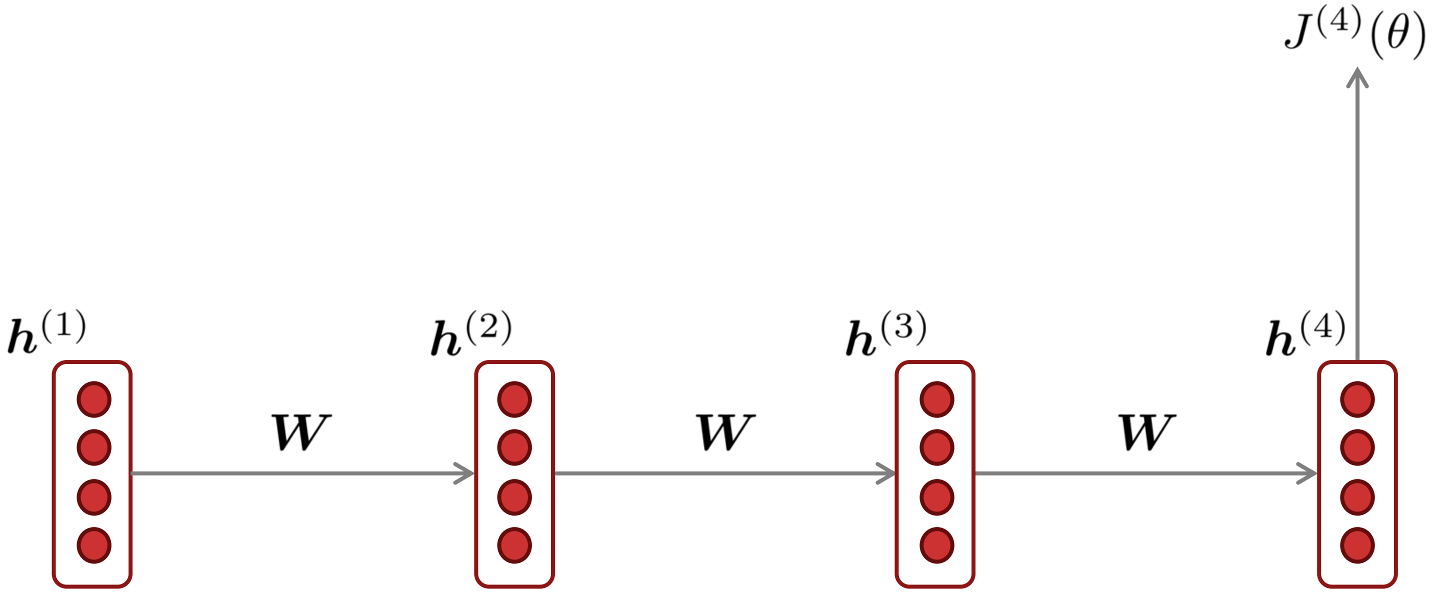
Normalized by number of words

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

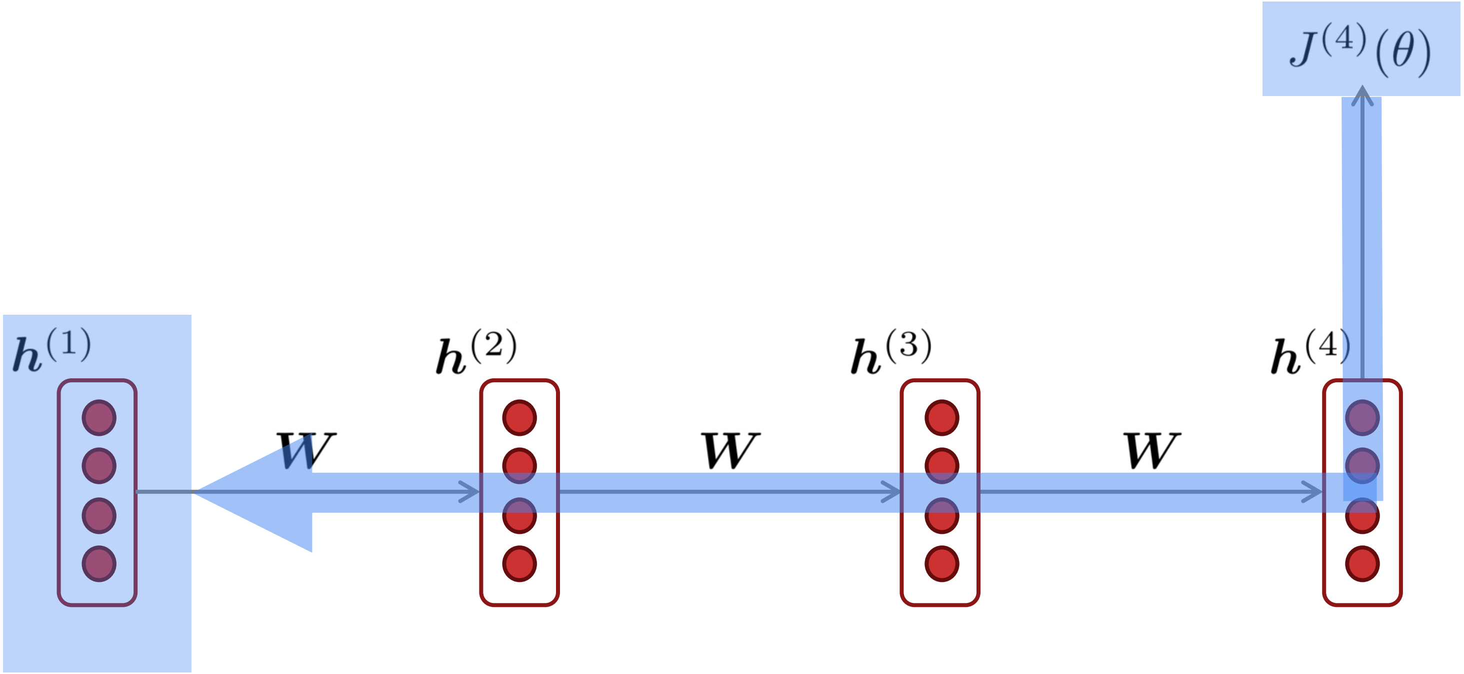
$$= \prod_{t=1}^{T} \left(\frac{1}{\hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}}} \right)^{1/T} = \exp\left(\frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}} \right) = \exp(J(\theta))$$

Lower perplexity is better!

3. Problems with RNNs: Vanishing and Exploding Gradients

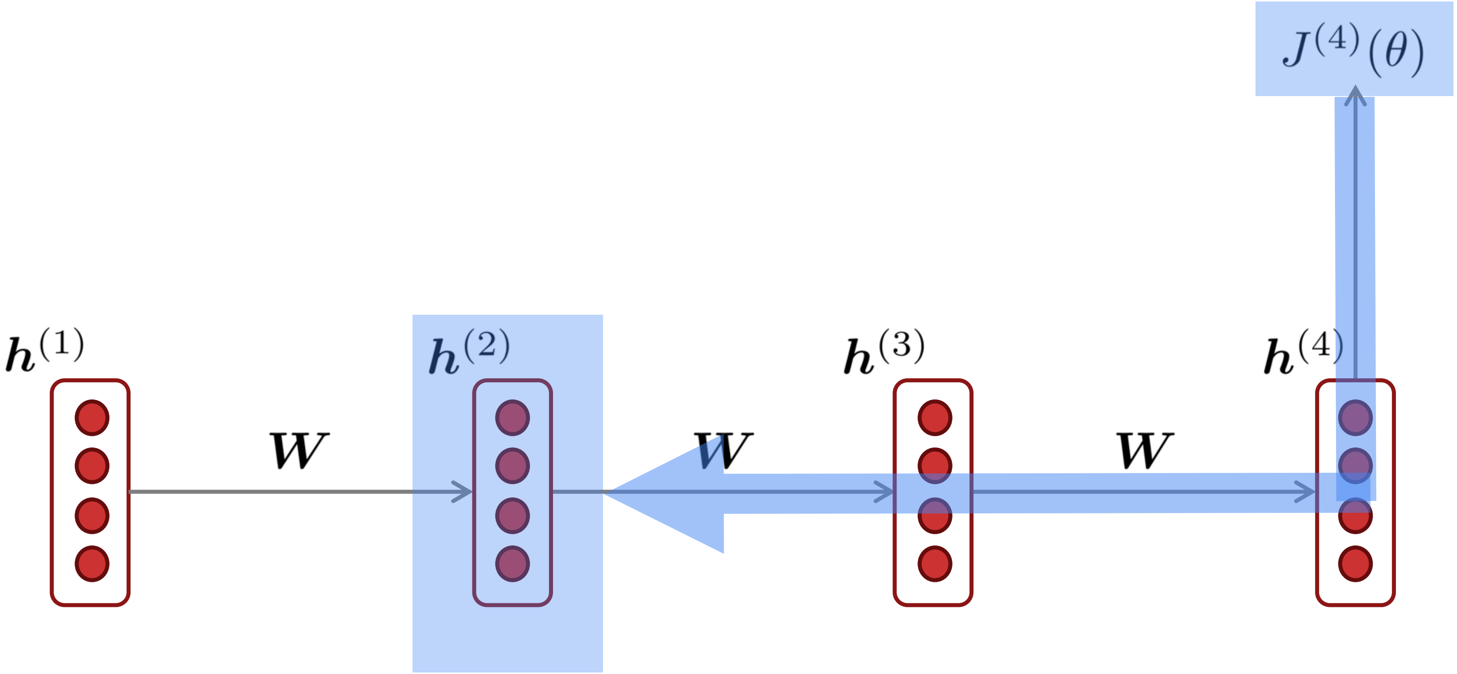


Vanishing gradient intuition



$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \ ?$$

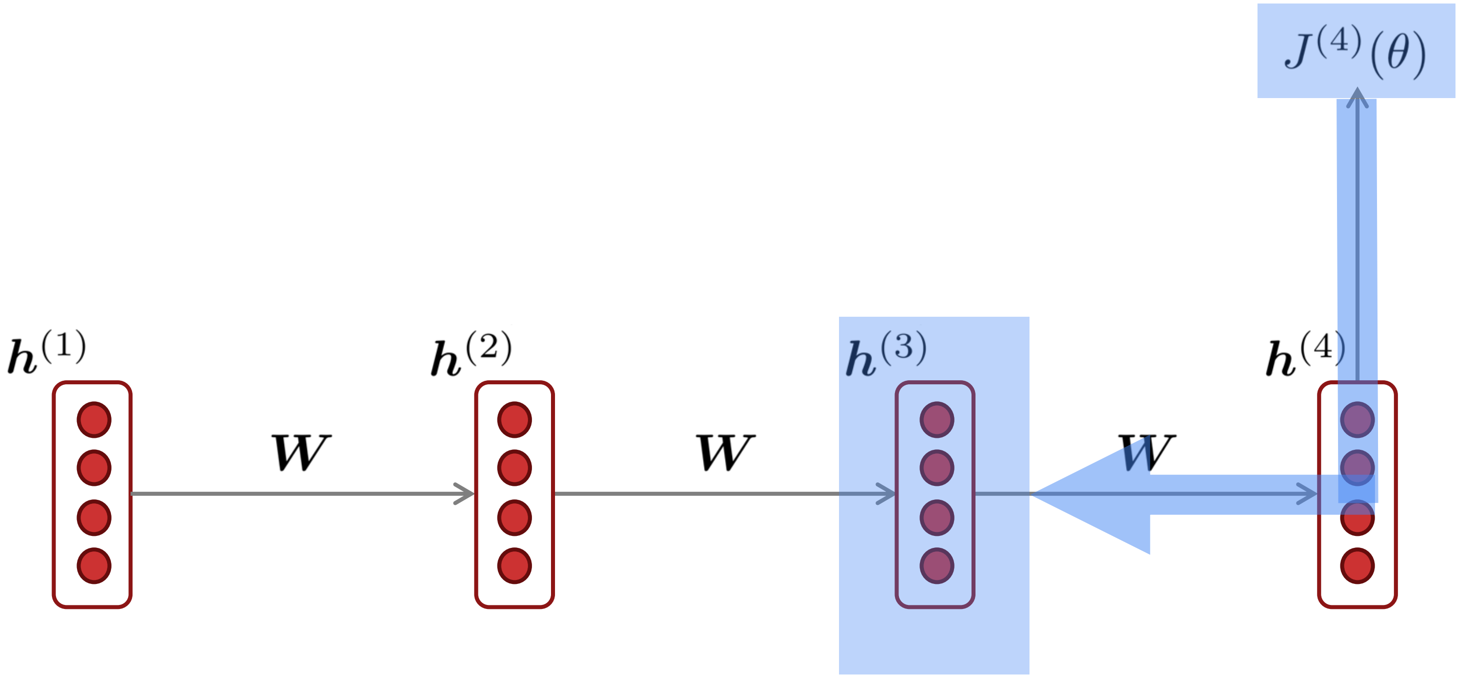
Vanishing gradient intuition



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

chain rule!

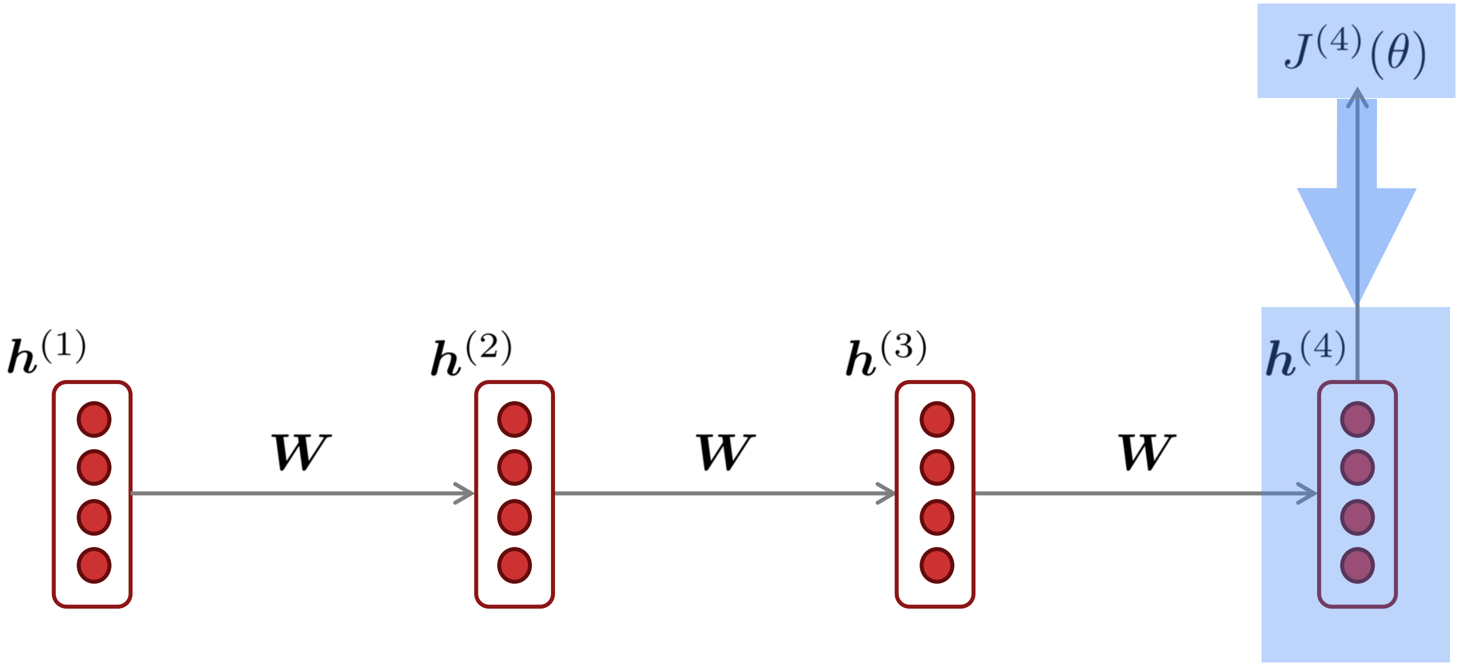
Vanishing gradient intuition



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

chain rule!

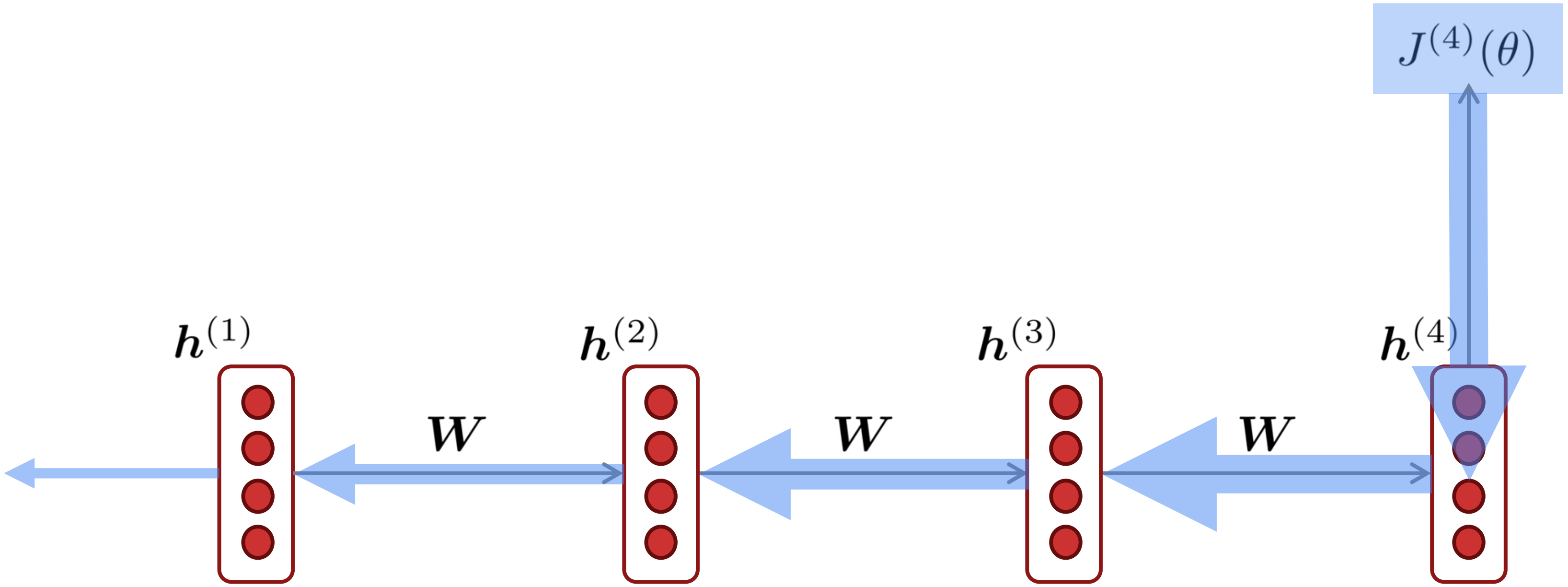
Vanishing gradient intuition



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

chain rule!

Vanishing gradient intuition



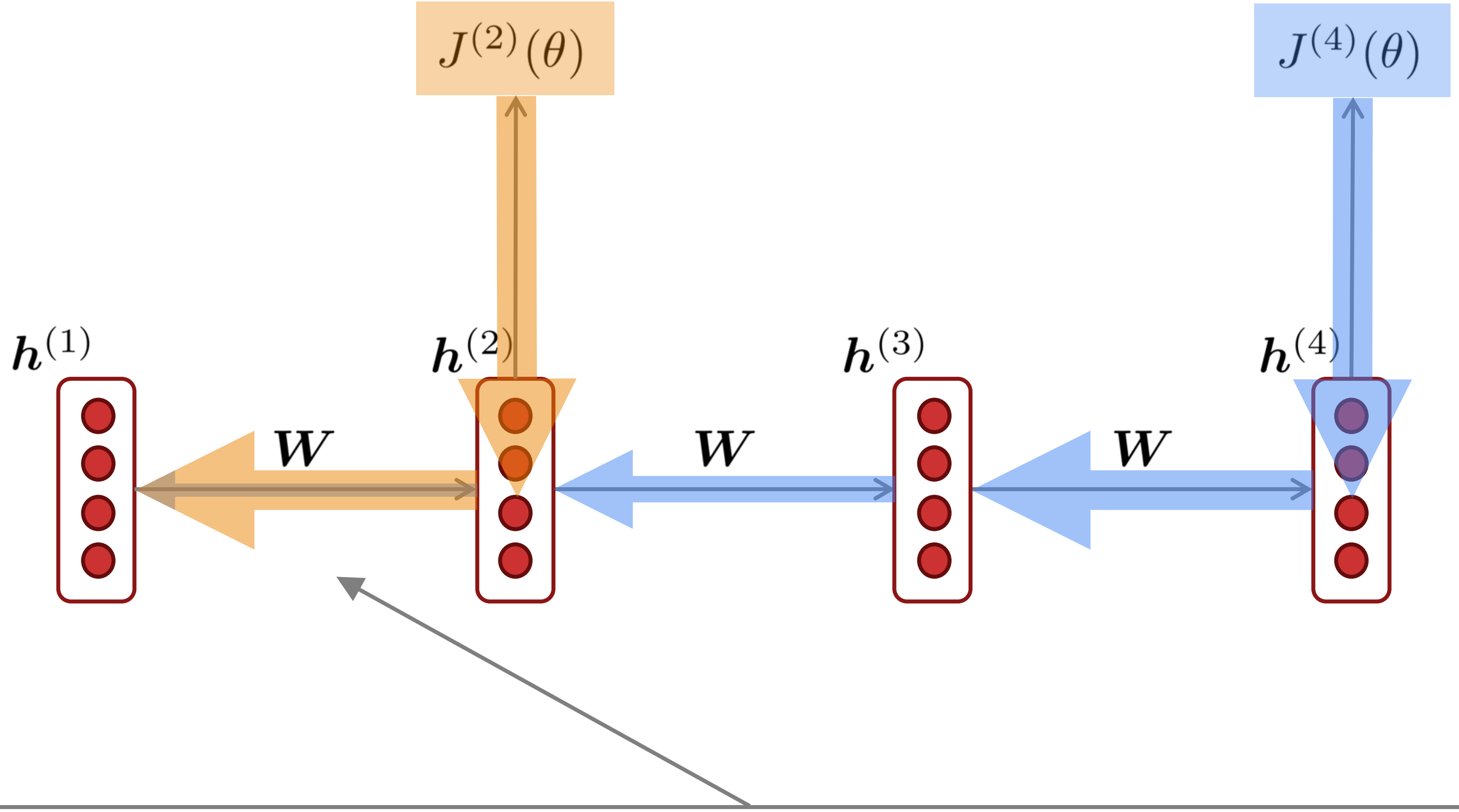
$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \boxed{\frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}}} \times \boxed{\frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}}} \times \boxed{\frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}}}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

What happens if these are small?

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

Source: "On the difficulty of training recurrent neural networks", Pascanu et al, 2013. http://proceedings.mlr.press/v28/pascanu13.pdf
(and supplemental materials), at http://proceedings.mlr.press/v28/pascanu13-supp.pdf

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 7th 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:

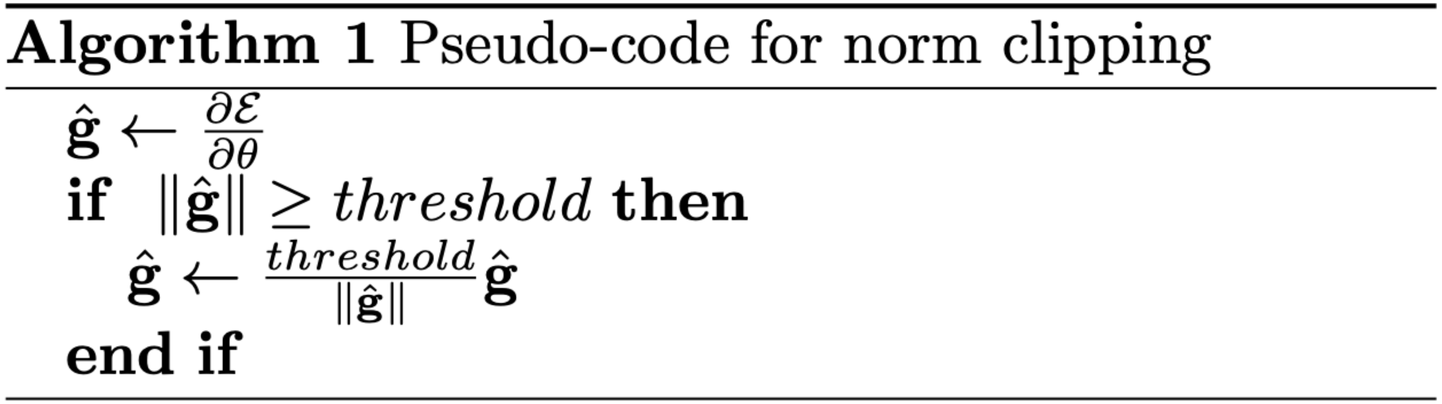
$$\theta^{new} = \theta^{old} - \overbrace{\alpha}^{\text{learning rate}} \underbrace{\nabla_\theta J(\theta)}_{\text{gradient}}$$

- 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 $\|\hat{\mathbf{g}}\| \geq threshold$ then
    $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
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

Source: "On the difficulty of training recurrent neural networks", Pascanu et al, 2013. http://proceedings.mlr.press/v28/pascanu13.pdf

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

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

- First: How about an RNN with separate memory which is added to?
  - Long Short-Term Memory (LSTM) [link]

- And then: Creating more direct and linear pass-through connections in model
  - Attention, residual connections, etc.

5. Machine Translation

Machine Translation (MT) is the task of translating a sentence *x* from one language (the source language) to a sentence *y* in another language (the target language).

x: *I like deep learning*

↓

y: *我喜欢深度学习*

NMT: the first big success story of NLP Deep Learning

Neural Machine Translation went from a fringe research attempt in **2014** to the leading standard method in **2016**

- **2014**: First seq2seq paper published [Sutskever et al. 2014]

- **2016**: Google Translate switches from SMT to NMT – and by 2018 everyone has

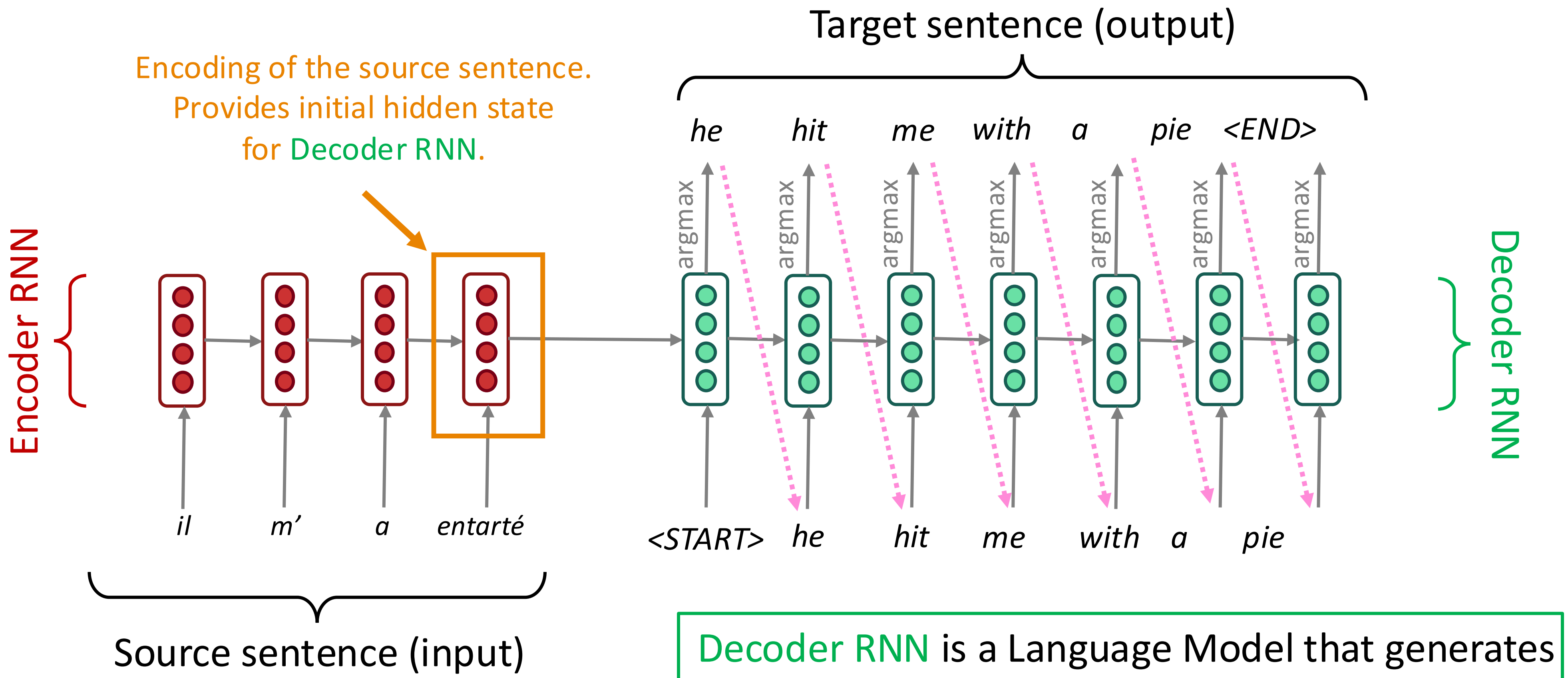
Microsoft SYSTRAN beyond language Google facebook

Baidu 百度 網易 NetEase www.163.com Tencent 腾讯 搜狗搜索

- This is amazing!
  - **SMT** systems, built by hundreds of engineers over many years, outperformed by NMT systems trained by small groups of engineers in a few months

Neural Machine Translation (NMT)

The sequence-to-sequence model



Decoder RNN is a Language Model that generates target sentence, *conditioned on encoding*.

Note: This diagram shows **test time** behavior: decoder output is fed in ⋯⋯▸ as next step's input

Encoder RNN produces an encoding of the source sentence.

Sequence-to-sequence is versatile!

- The general notion here is an encoder-decoder model
  - One neural network takes input and produces a neural representation
  - Another network produces output based on that neural representation
  - If the input and output are sequences, we call it a seq2seq model

- Sequence-to-sequence is useful for *more than just MT*
- Many NLP tasks can be phrased as sequence-to-sequence:
  - Summarization (long text → short text)
  - Dialogue (previous utterances → next utterance)
  - Parsing (input text → output parse as sequence)
  - Code generation (natural language → Python code)

Neural Machine Translation (NMT)

- The sequence-to-sequence model is an example of a **Conditional Language Model**
  - **Language Model** because the decoder is predicting the next word of the target sentence *y*
  - **Conditional** because its predictions are *also* conditioned on the source sentence *x*

- NMT directly calculates $P(y|x)$:

$$P(y|x) = P(y_1|x)\, P(y_2|y_1, x)\, P(y_3|y_1, y_2, x) \ldots \underbrace{P(y_T|y_1, \ldots, y_{T-1}, x)}_{\text{Probability of next target word, given target words so far and source sentence } x}$$

- **Question:** How to train an NMT system?
- **(Easy) Answer:** Get a big parallel corpus…
  - But there is now exciting work on “unsupervised NMT”, data augmentation, etc.

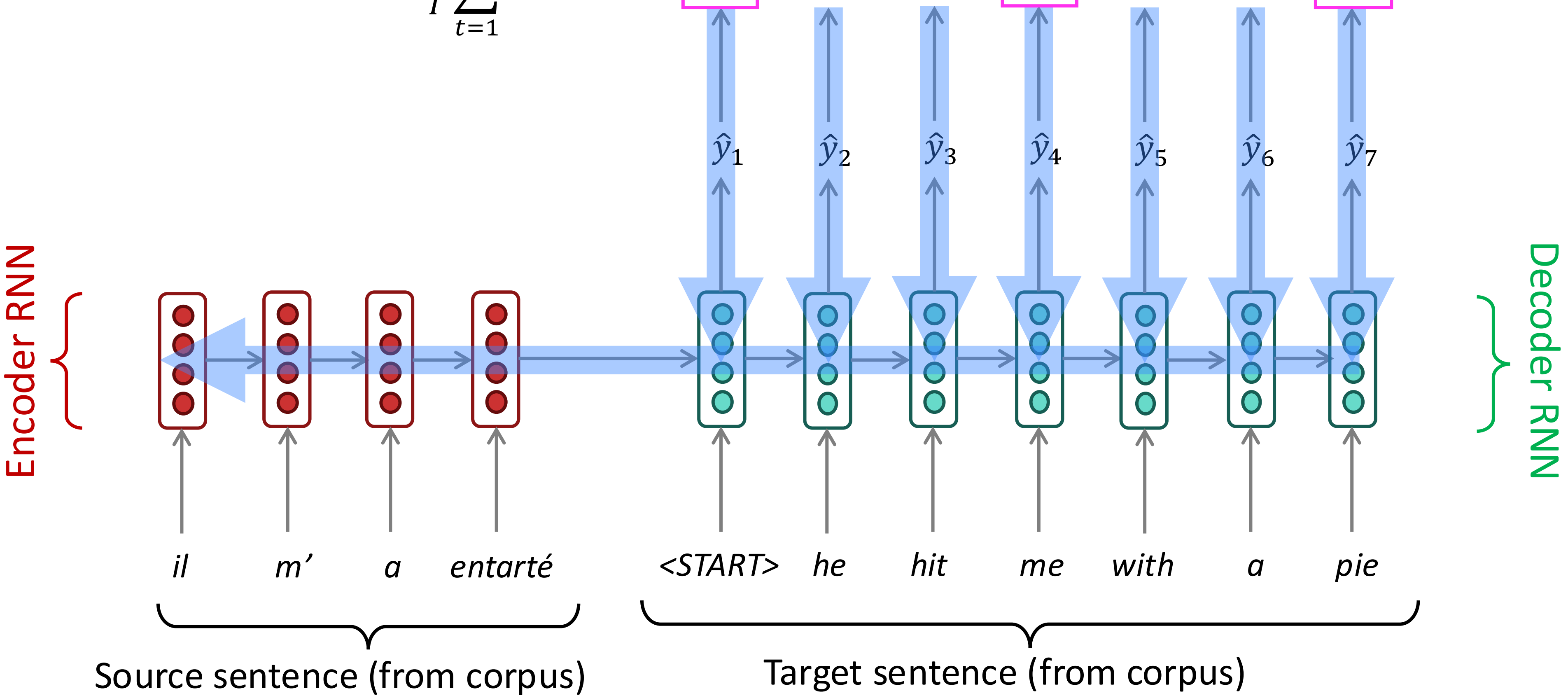
Training a Neural Machine Translation system

$$J = \frac{1}{T}\sum_{t=1}^{T} J_t \quad = \quad \boxed{J_1} + J_2 + J_3 + \boxed{J_4} + J_5 + J_6 + \boxed{J_7}$$

= negative log prob of "he"

= negative log prob of "with"

= negative log prob of <END>

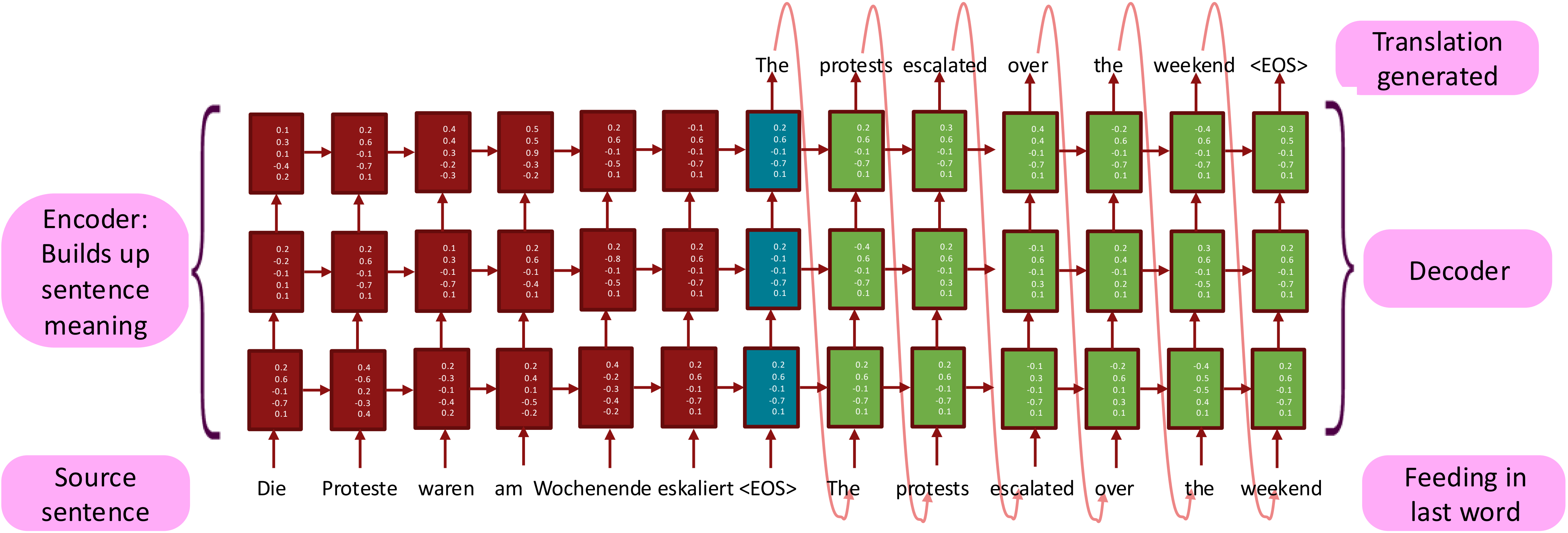


Seq2seq is optimized as a **single system.** Backpropagation operates *"end-to-end"*.

Multi-layer deep encoder-decoder machine translation net

[Sutskever et al. 2014; Luong et al. 2015]

The hidden states from RNN layer *i* are the inputs to RNN layer *i*+1



Translation generated

Encoder: Builds up sentence meaning

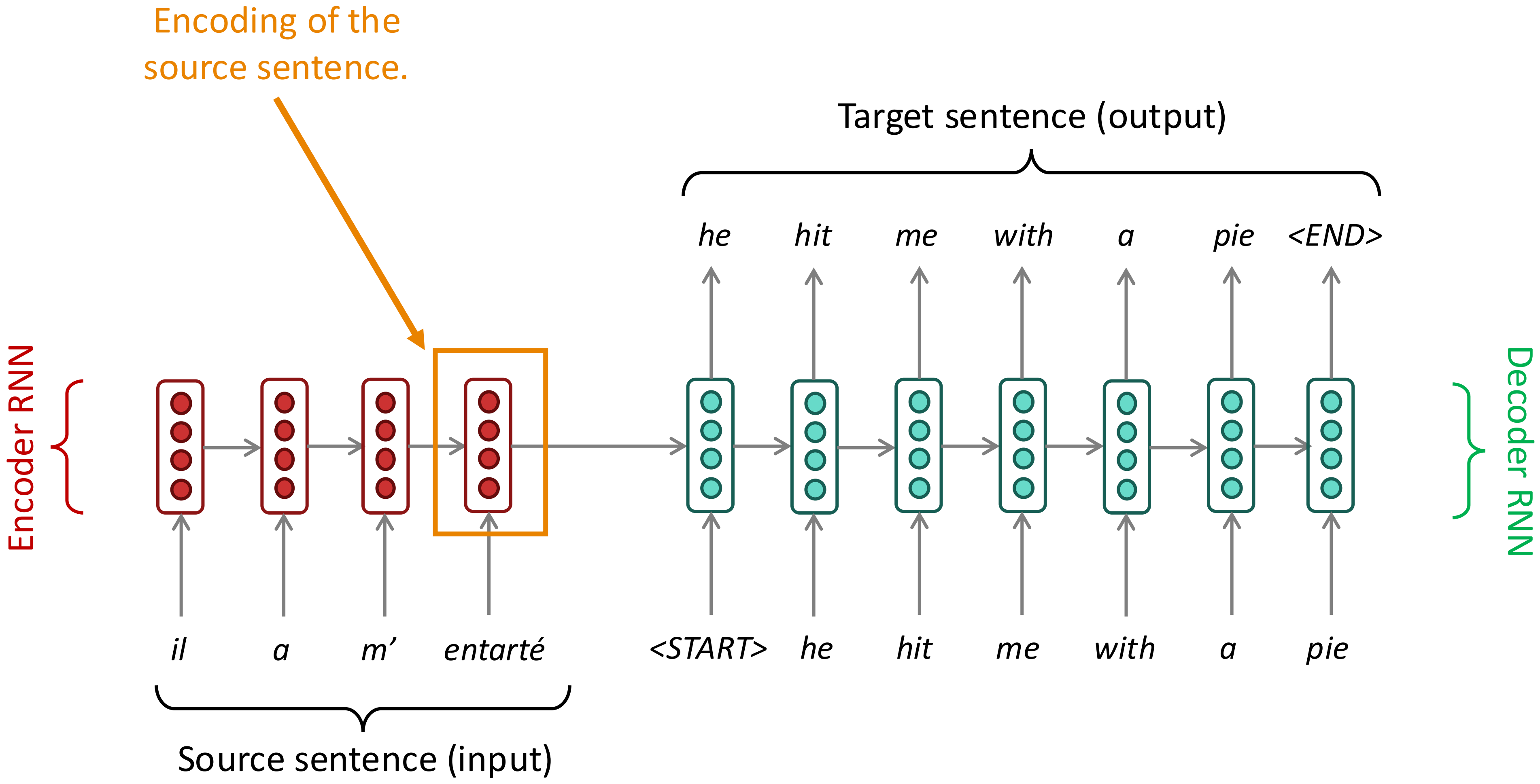
Decoder

Source sentence

Feeding in last word

Conditioning = Bottleneck

The final piece: the bottleneck problem in RNNs



Problems with this architecture?

Lecture Plan

Lecture 4: Language modeling + RNNs

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

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

↓ **motivates**

2. Language models with neural nets: **Recurrent Neural Networks (RNNs)** (25 mins)
3. Problems with RNNs: exploding and vanishing gradients (20 mins)
4. Machine translation (10 mins)