# Natural Language Processing with Deep Learning CS224N/Ling284

Christopher Manning

Lecture 6: Simple and LSTM Recurrent Neural Networks

# Lecture Plan

1. RNN Language Models (25 mins)
2. Other uses of RNNs (8 mins)
3. Exploding and vanishing gradients (15 mins)
4. LSTMs (20 mins)
5. Bidirectional and multi-layer RNNs (12 mins)

- Projects
    - Next Thursday: a lecture about choosing final projects
    - It's fine to delay thinking about projects until next week
    - But if you're already thinking about projects, you can view some info/inspiration on the website. It's still last year's information at present!
    - It's great if you can line up your own mentor; we also lining up some mentors

# Overview

- Last lecture we learned:
  - Language models, n-gram language models, and Recurrent Neural Networks (RNNs)

- Today we'll learn how to get RNNs to work for you
  - Training RNNs
  - Uses of RNNs
  - Problems with RNNs (exploding and vanishing gradients) and how to fix them
  - These problems motivate a more sophisticated RNN architecture: LSTMs
  - And other more complex RNN options: bidirectional RNNs and multi-layer RNNs

- Next lecture we'll learn:
  - How we can do Neural Machine Translation (NMT) using an RNN-based architecture called sequence-to-sequence with attention

3

# 1. The Simple RNN Language Model

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

**output distribution**

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

**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
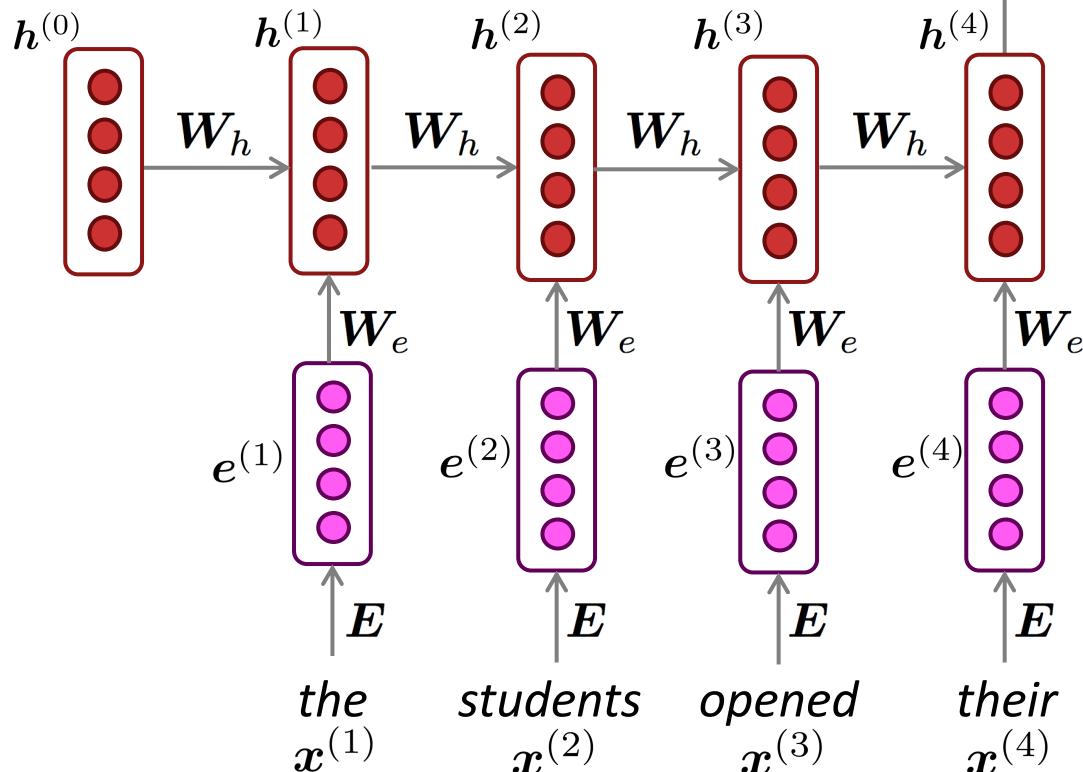
**word embeddings**

$$e^{(t)} = Ex^{(t)}$$
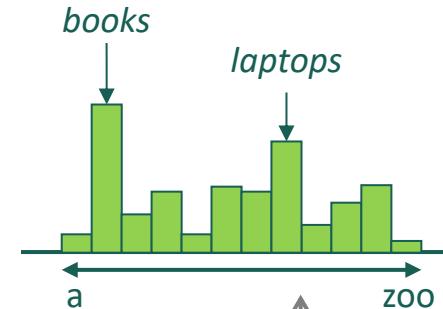
**words / one-hot vectors**

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



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

4

# Training an RNN Language Model
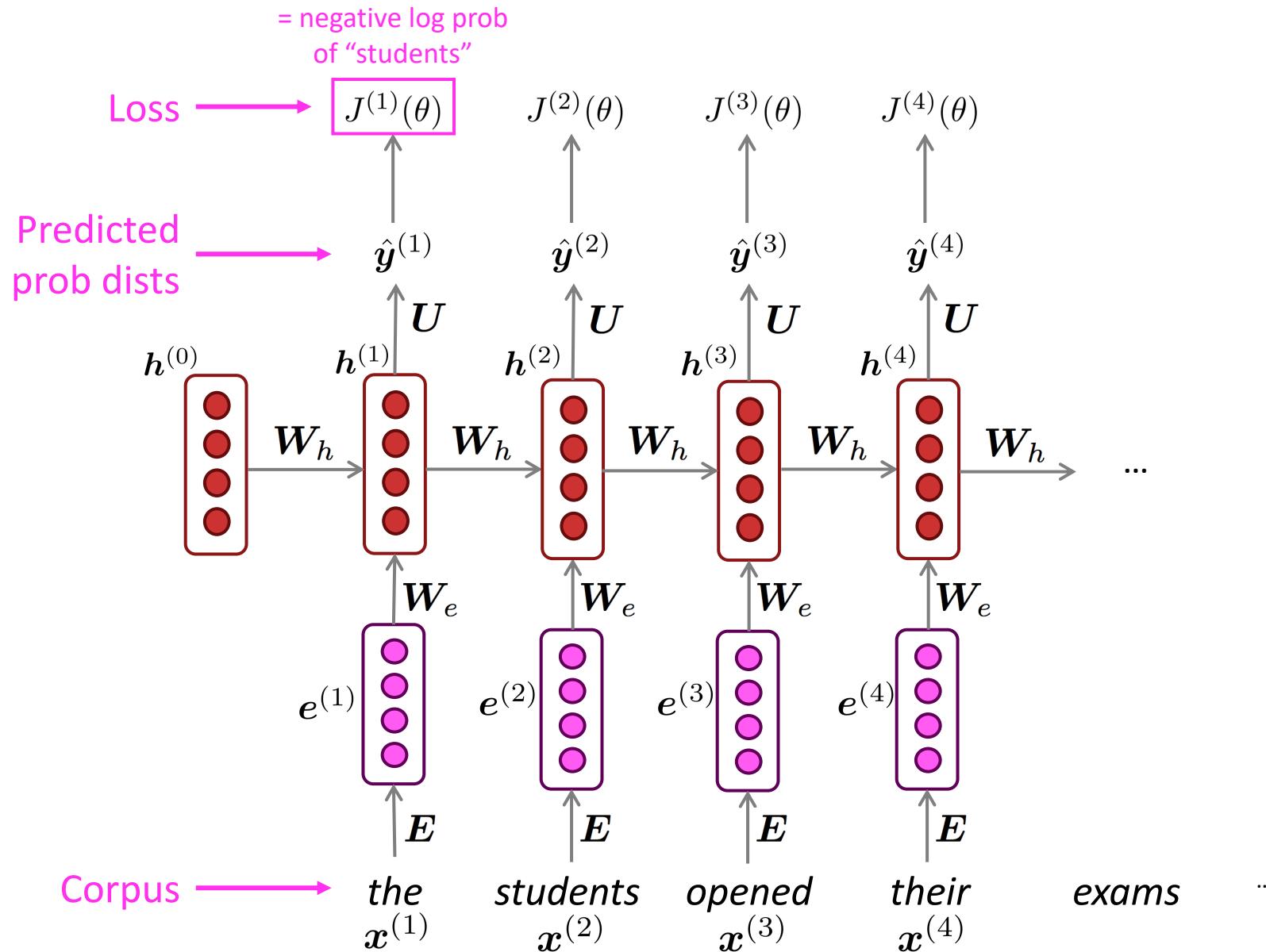
- Get a big corpus of text which is a sequence of words $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$
- Feed into RNN-LM; compute output distribution $\hat{\boldsymbol{y}}^{(t)}$ for *every step t*.
  - i.e., predict probability dist of *every word*, given words so far

- 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 $\boldsymbol{x}^{(t+1)}$):

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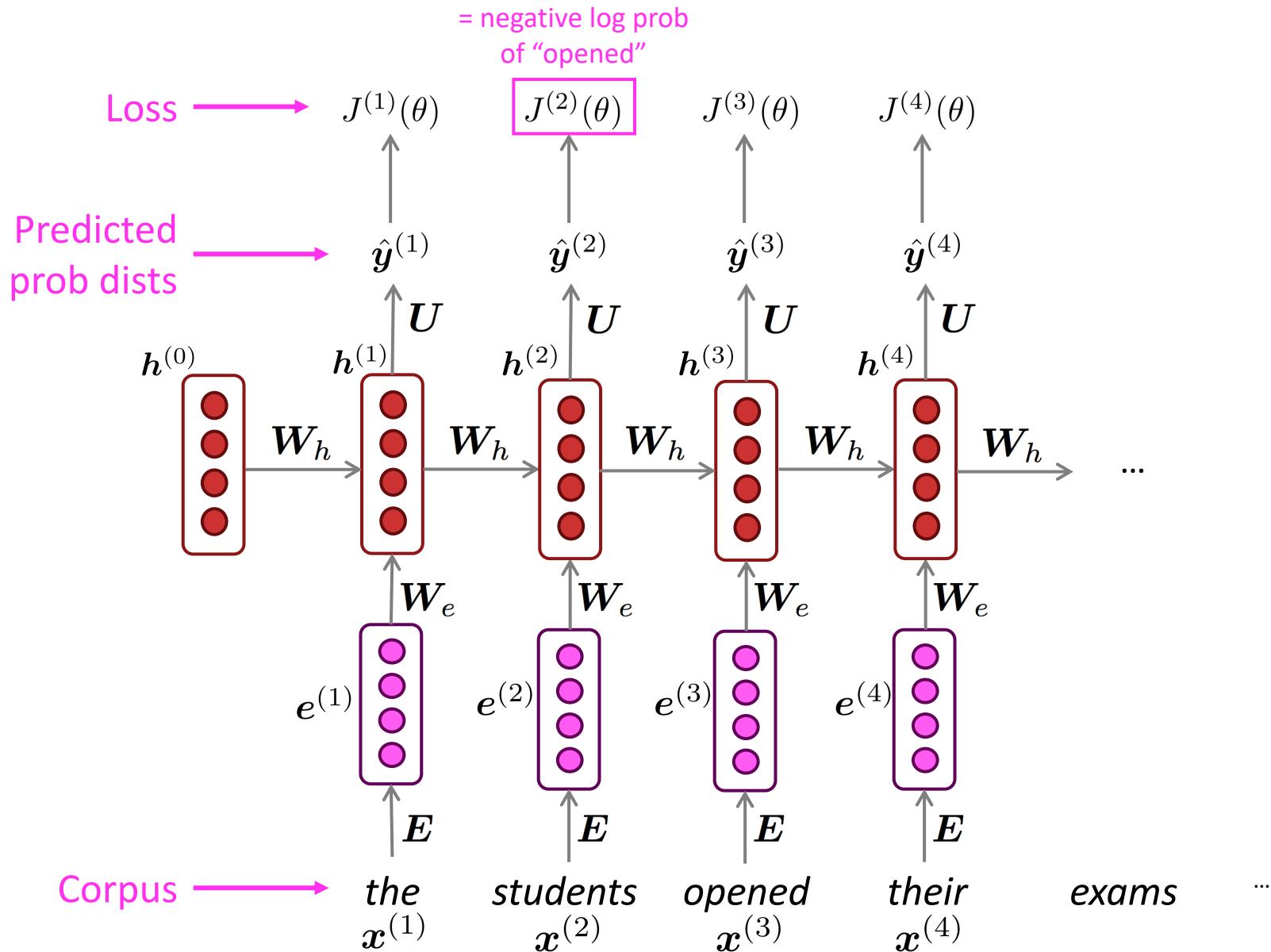
- Average this to get overall loss for entire training set:

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

5

# Training an RNN Language Model



= negative log prob
of "students"

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

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

6

# Training an RNN Language Model



= negative log prob of "opened"

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

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

# Training an RNN Language Model



= negative log prob
of "their"

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

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

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

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

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

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

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

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

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

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

# Training an RNN Language Model

= negative log prob of "exams"

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

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

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

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

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

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

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

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

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 an RNN Language Model

"Teacher forcing"

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

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* ...
$\boldsymbol{x}^{(1)}$ $\boldsymbol{x}^{(2)}$ $\boldsymbol{x}^{(3)}$ $\boldsymbol{x}^{(4)}$
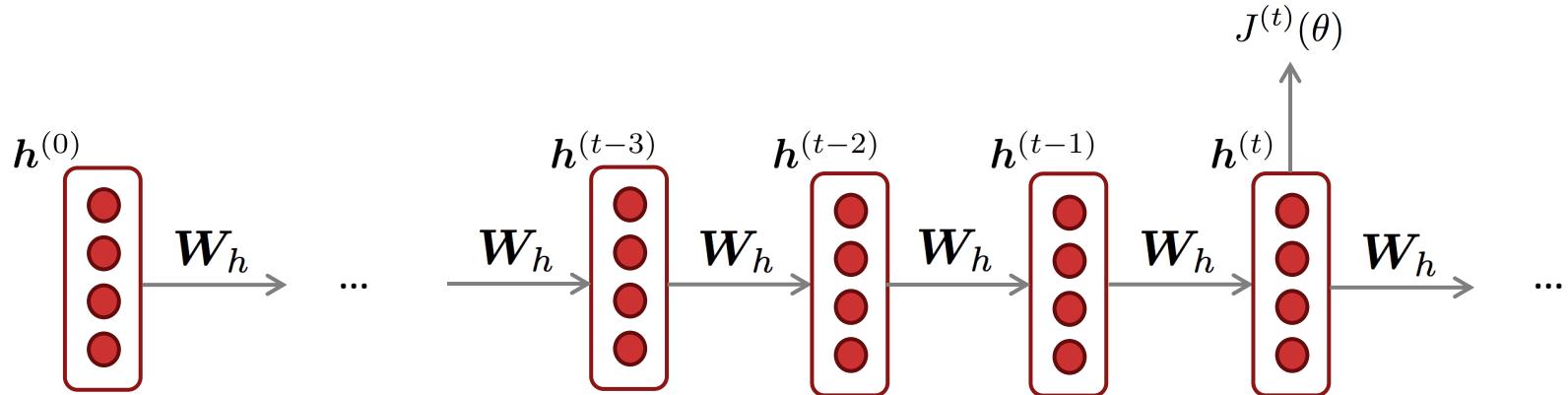
# Training a RNN Language Model

- 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.

# Training the parameters of RNNs: Backpropagation for RNNs



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

**Answer:** $\dfrac{\partial J^{(t)}}{\partial W_h} = \displaystyle\sum_{i=1}^{t} \dfrac{\partial J^{(t)}}{\partial W_h}\bigg|_{(i)}$
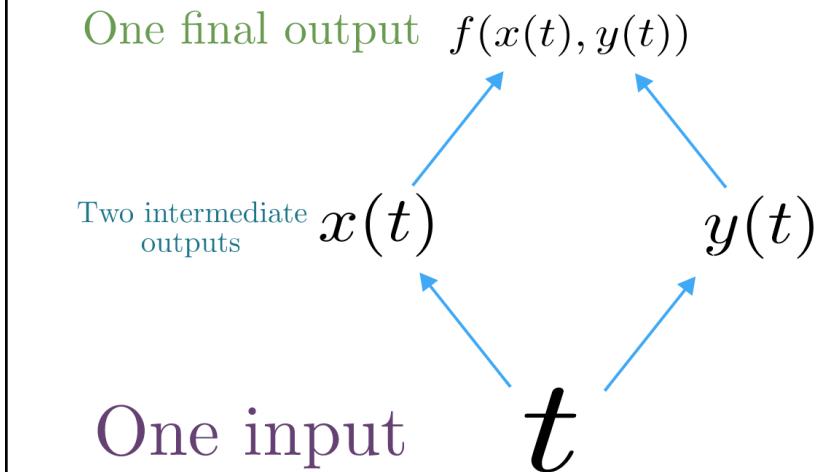
"The gradient w.r.t. a repeated weight is the sum of the gradient w.r.t. each time it appears"

**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 $t$

Gradients sum at outward branches! (lecture 3)

13
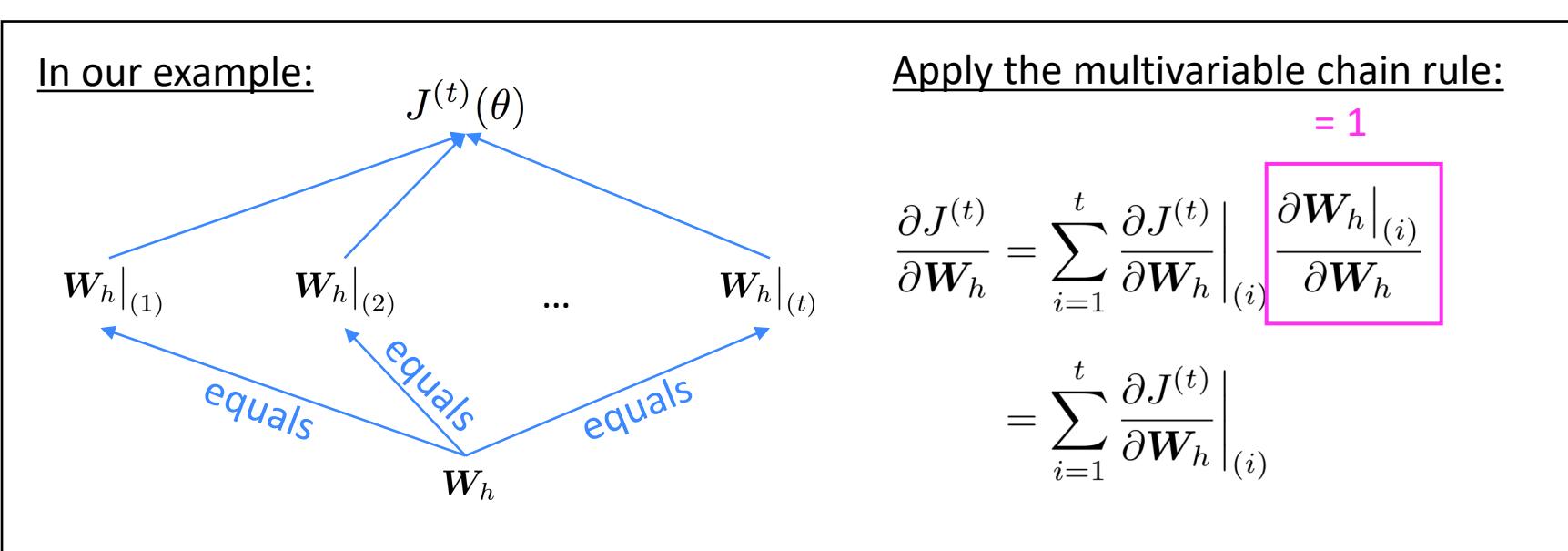
# Backpropagation for RNNs: Proof sketch

- 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:

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

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

equals   equals   equals

$W_h$

Apply the multivariable chain rule:

= 1

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

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

14

# Backpropagation for RNNs
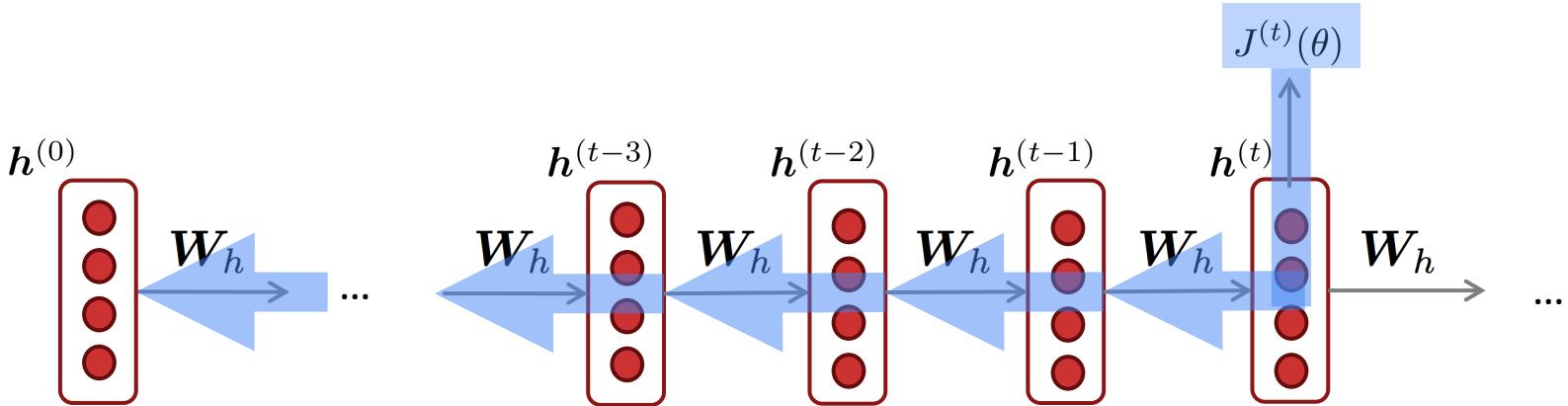


**Answer:** Backpropagate over timesteps *i=t*,...,0, summing gradients as you go. This algorithm is called **"backpropagation through time"** [Werbos, P.G., 1988, *Neural Networks* **1**, and others]
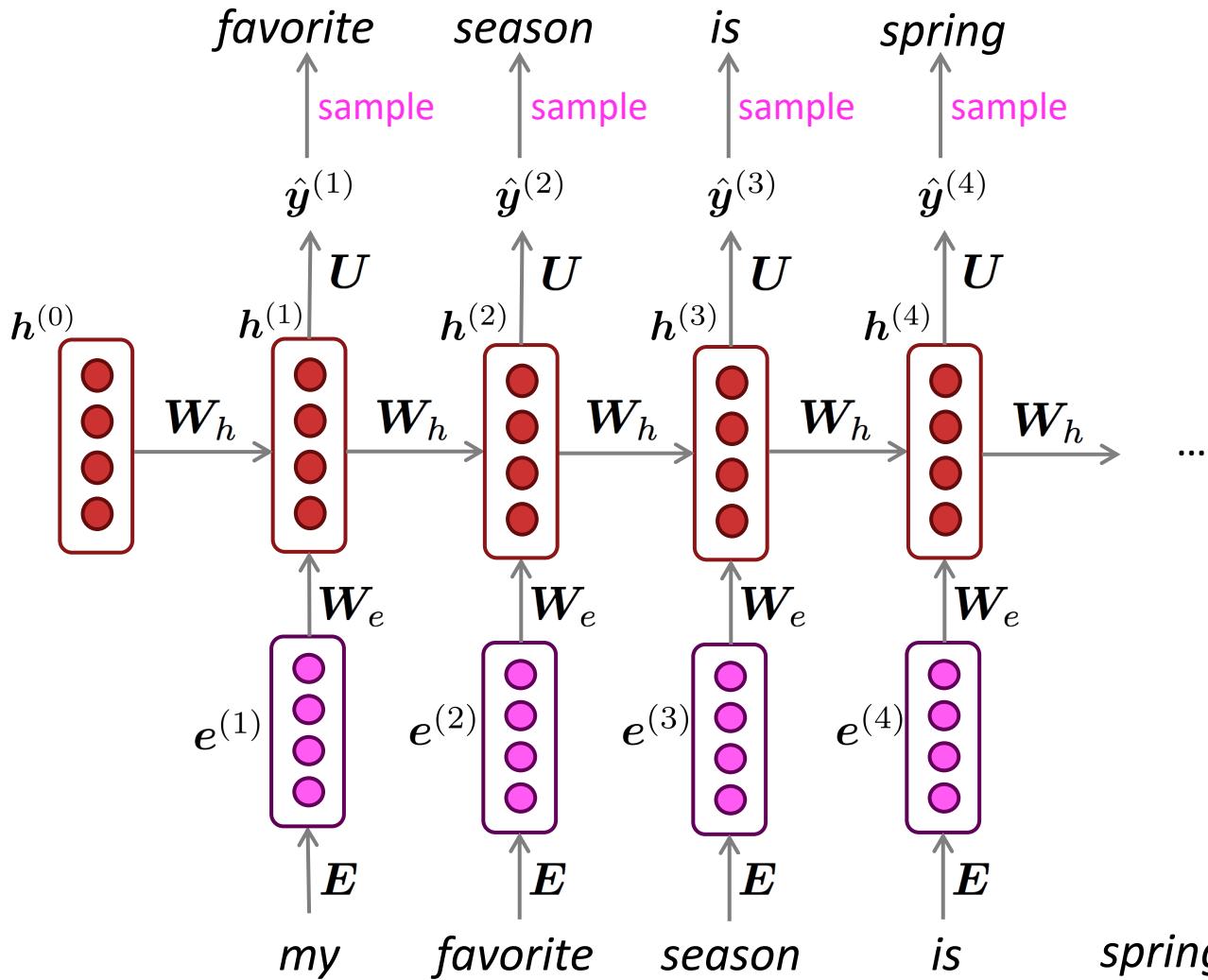
$$\frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} = \boxed{\sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}}\bigg|_{(i)}}$$

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

# Generating text with a RNN Language Model

Just like a n-gram Language Model, you can use an 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.
```

18

# 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 $J(\theta)$ :

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

**Lower** perplexity is better!

# 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/

20
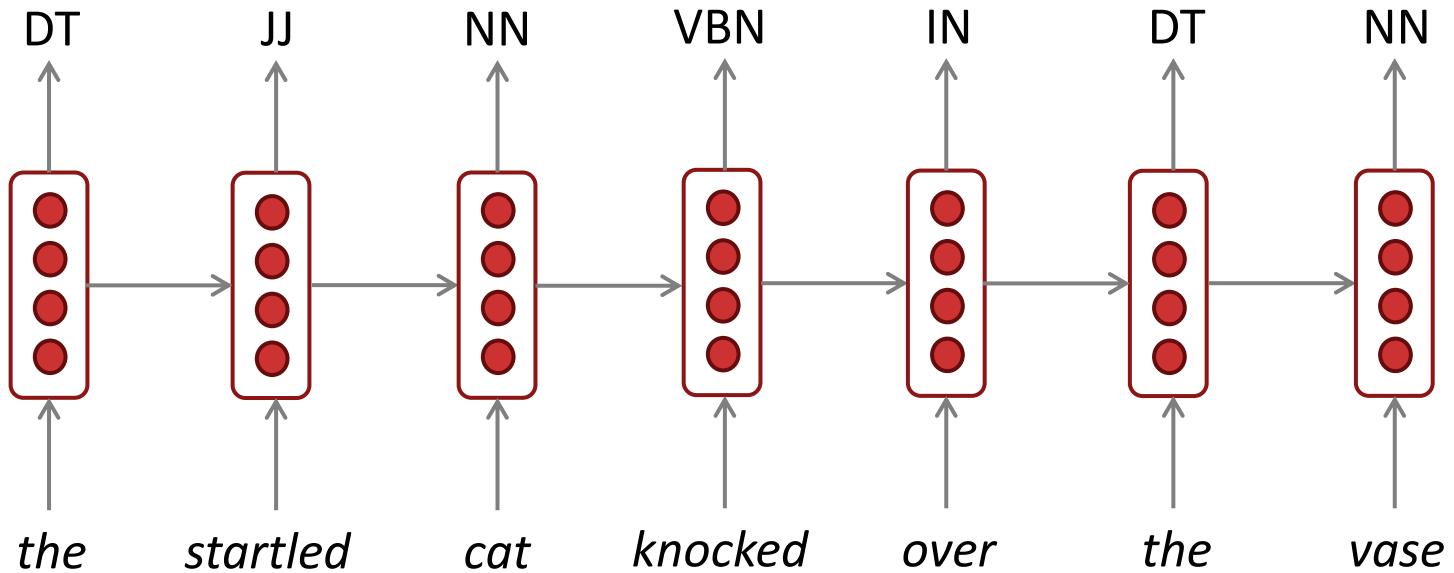
# Why should we care about Language Modeling?

- 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
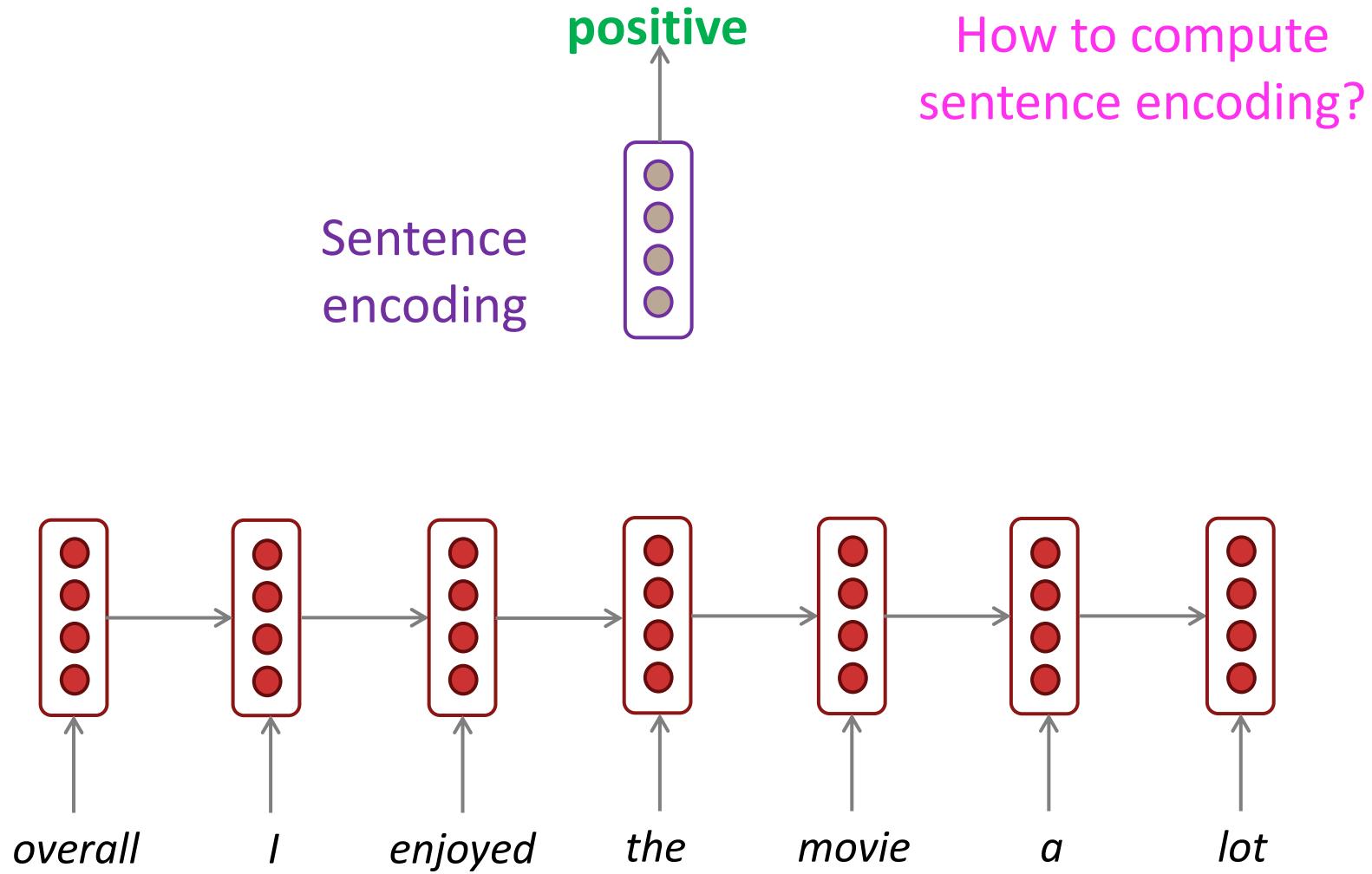
- **Language Model**: A system that predicts the next word

- **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!

# 2. Other RNN uses: RNNs can be used for sequence tagging
e.g., **part-of-speech tagging,** named entity recognition
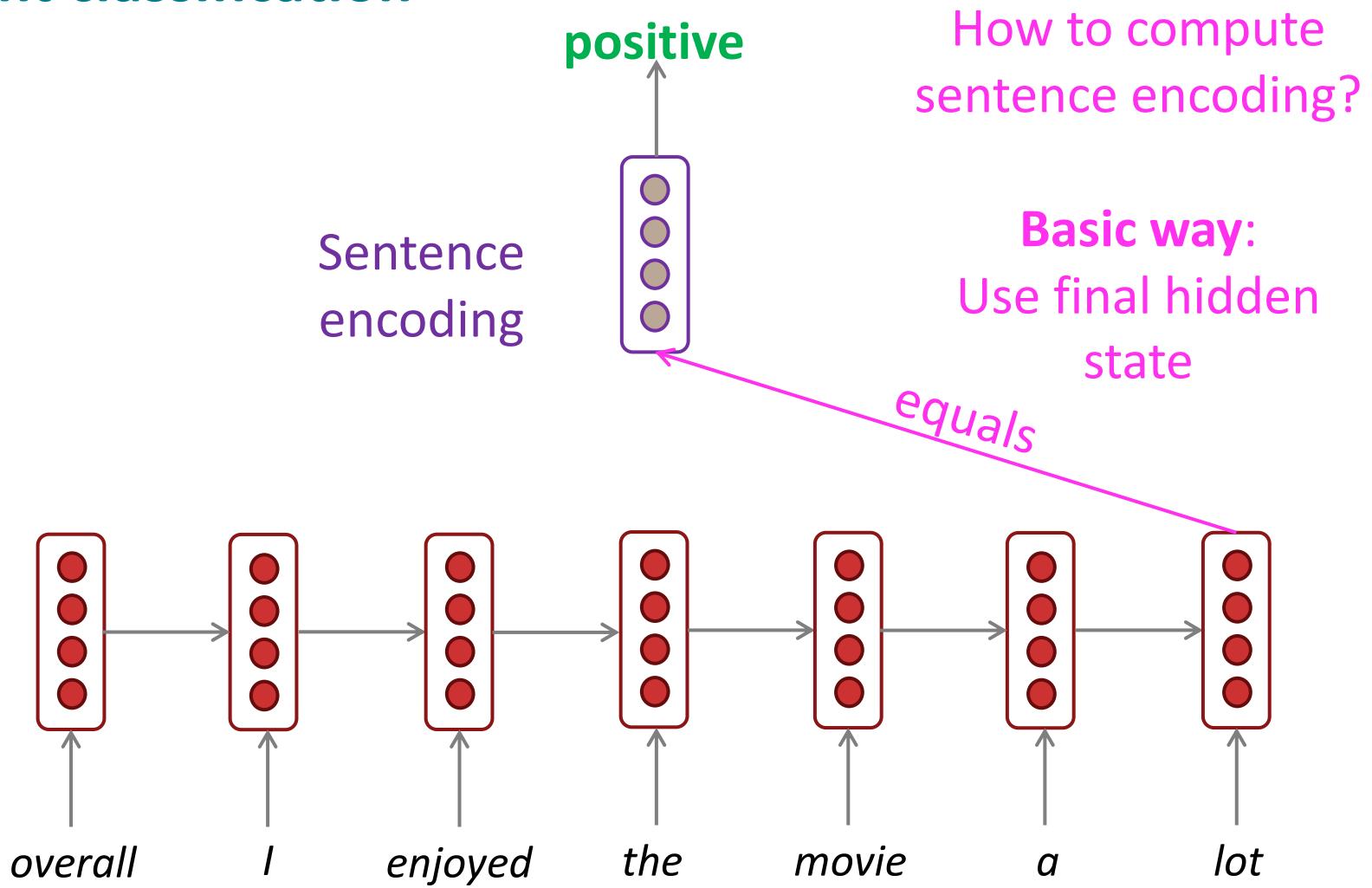
# RNNs can be used for sentence classification
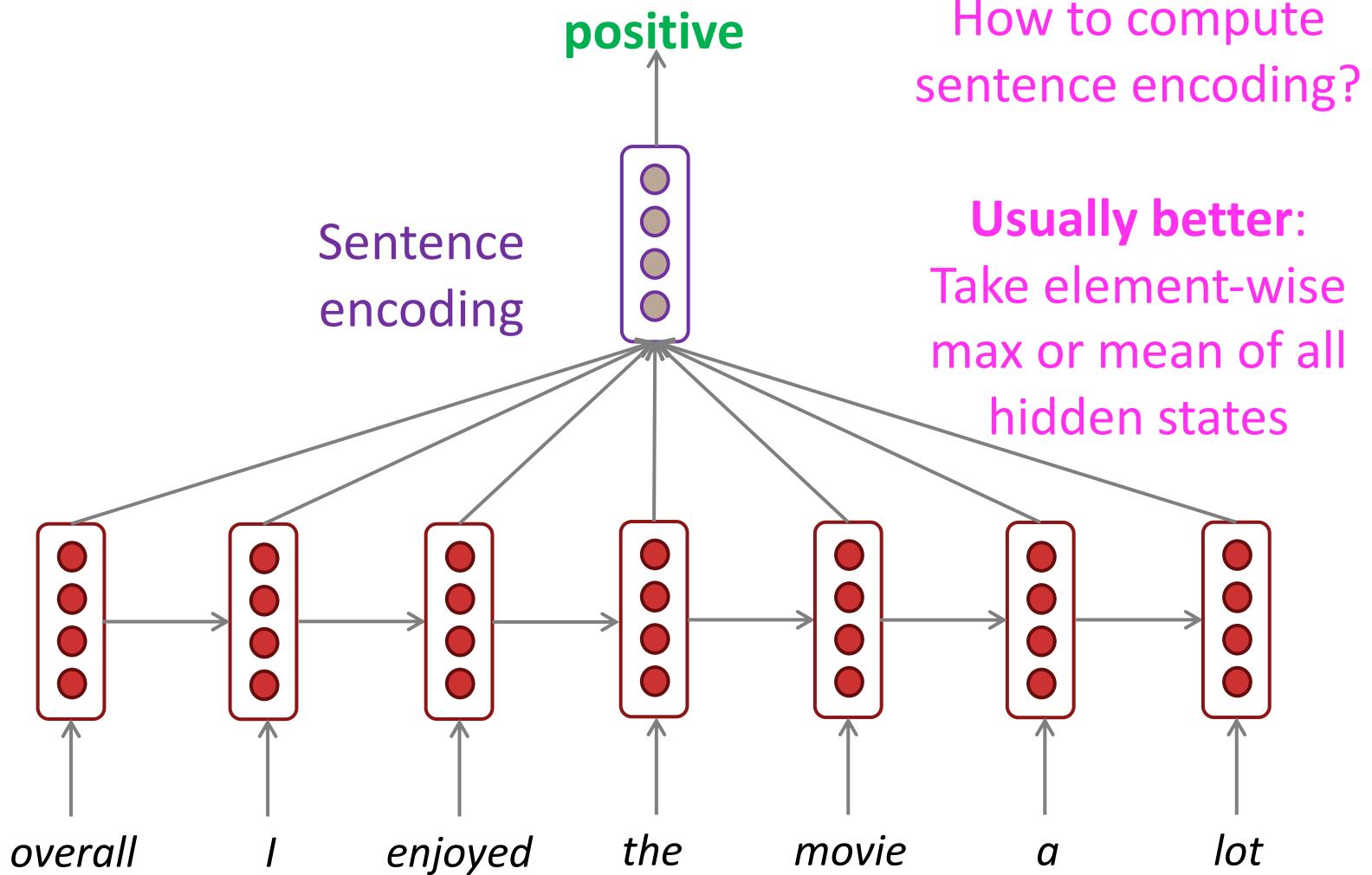
## e.g., **sentiment classification**

**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
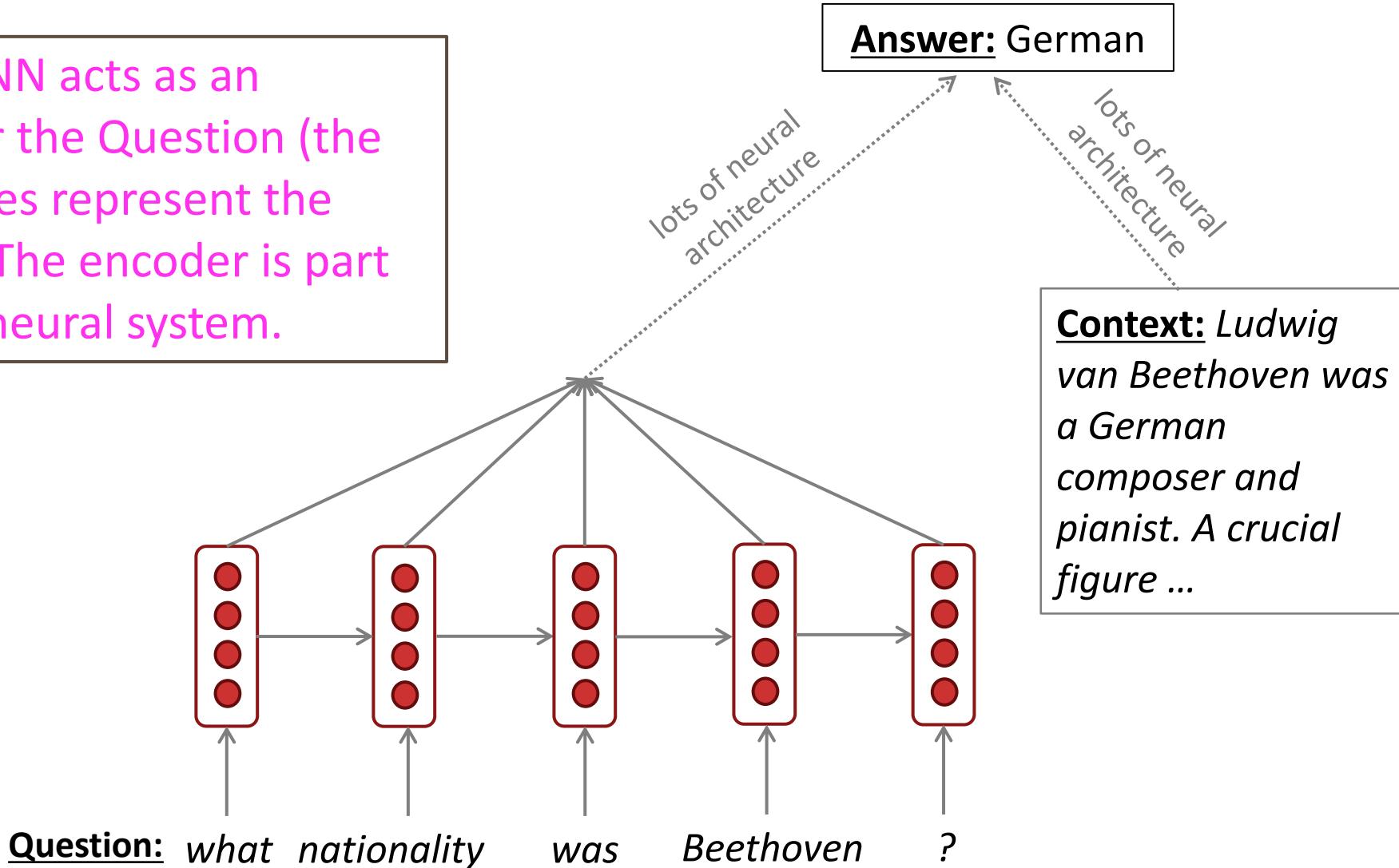
# RNNs can be used as a language encoder module
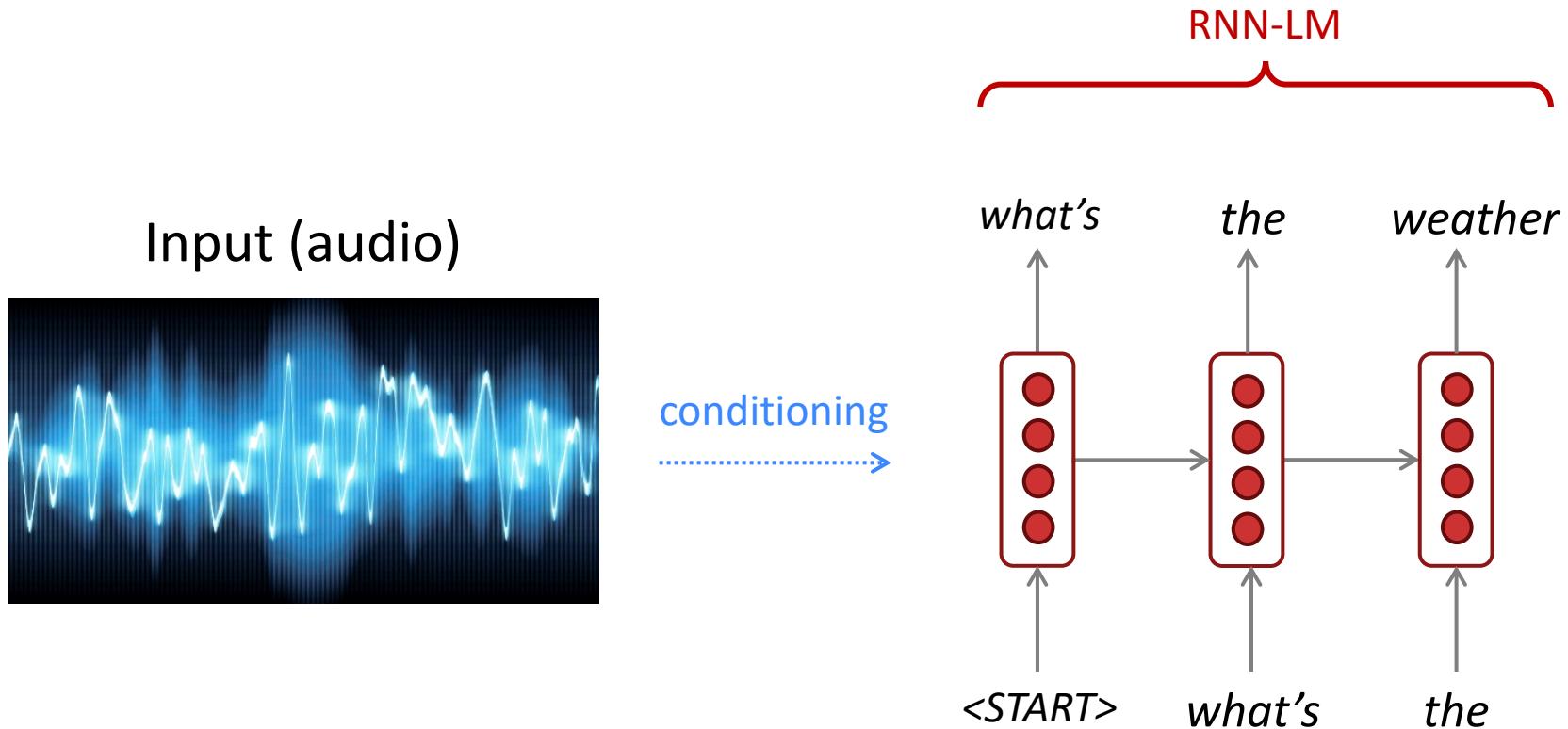
e.g., **question answering**, machine translation, *many other tasks!*

**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   ?*
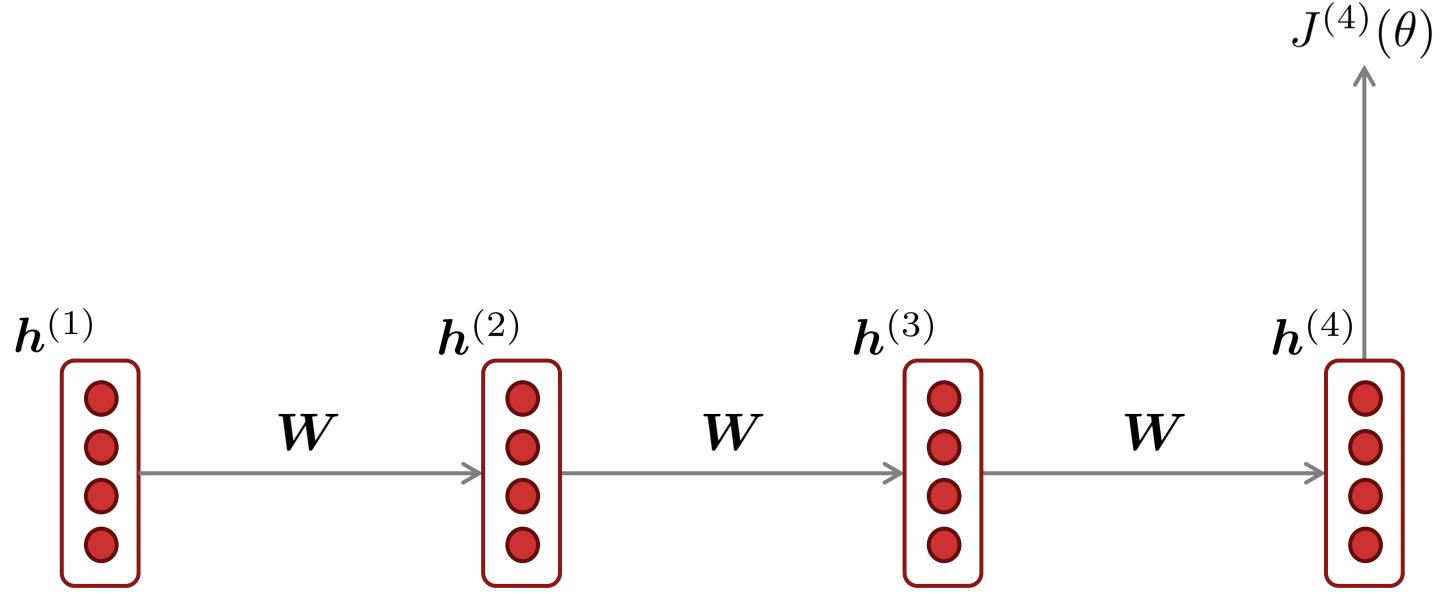
# RNN-LMs can be used to generate text

e.g., **speech recognition**, machine translation, summarization

RNN-LM

what's        the        weather

Input (audio)

conditioning

<START>        what's        the
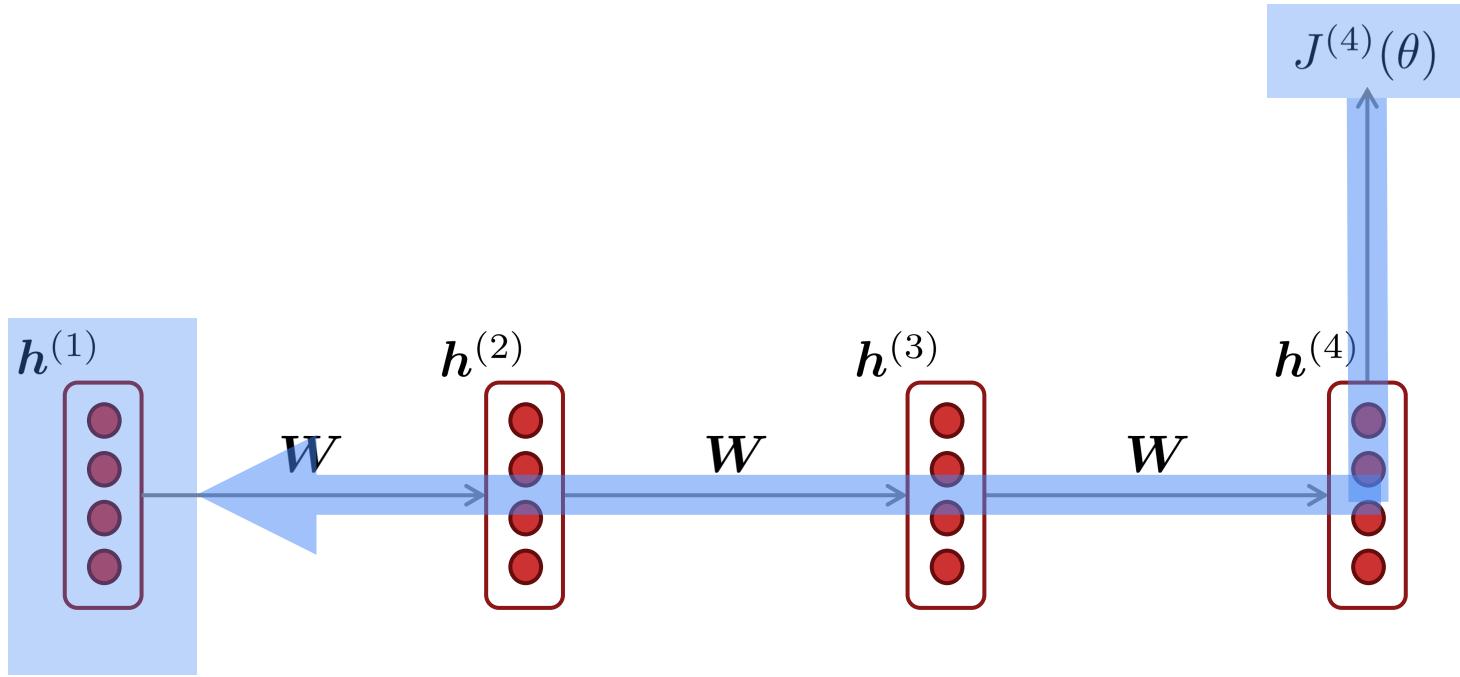
This is an example of a *conditional language model*.
We'll see Machine Translation in much more detail next class.

# 3. Problems with Vanishing and Exploding Gradients
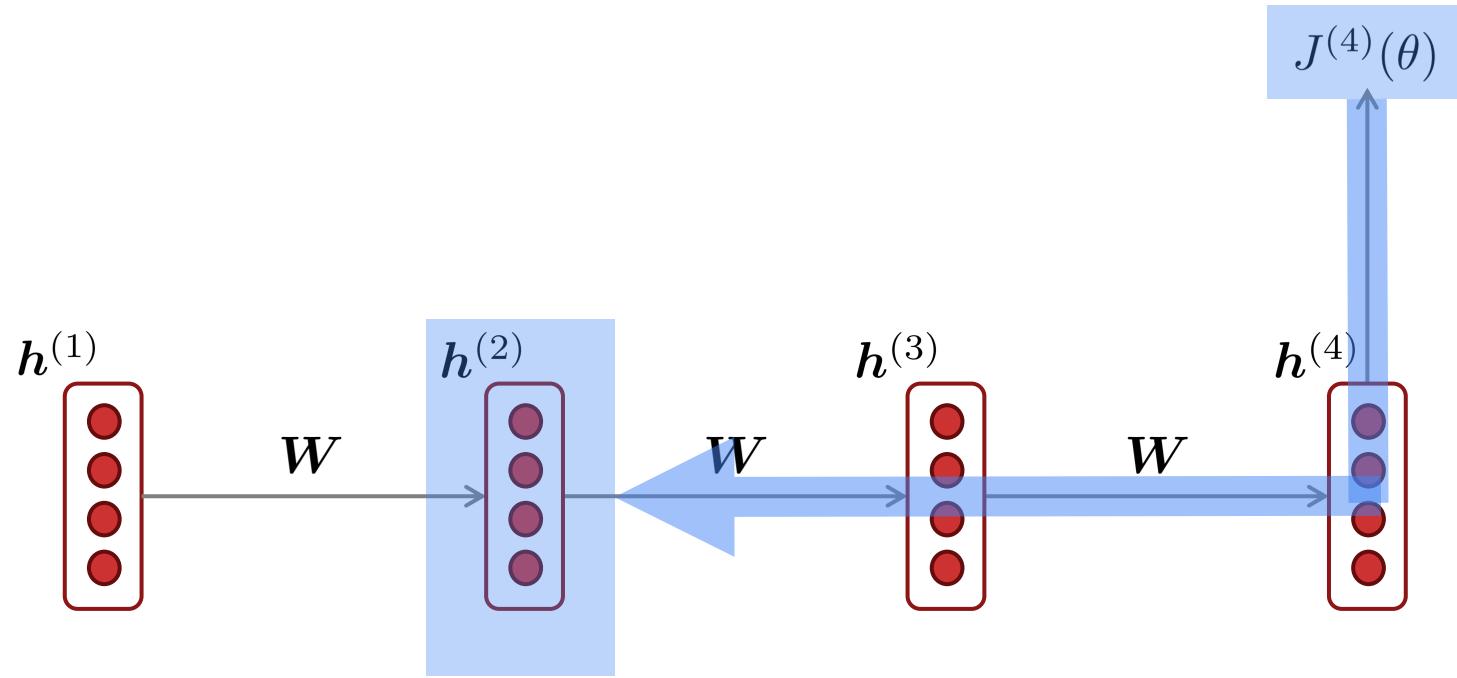
# Vanishing gradient intuition



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

# Vanishing gradient intuition

$J^{(4)}(\theta)$

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

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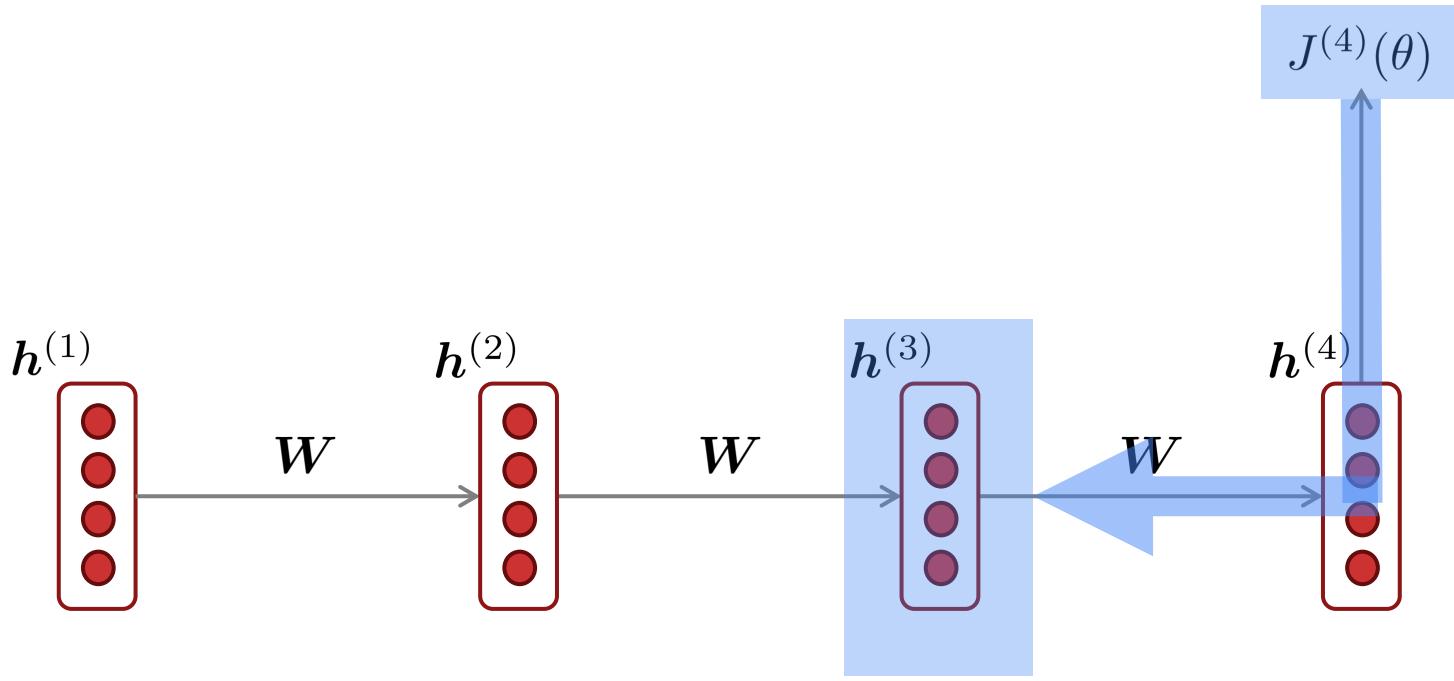
$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial J^{(4)}}{\partial h^{(2)}}$$

chain rule!

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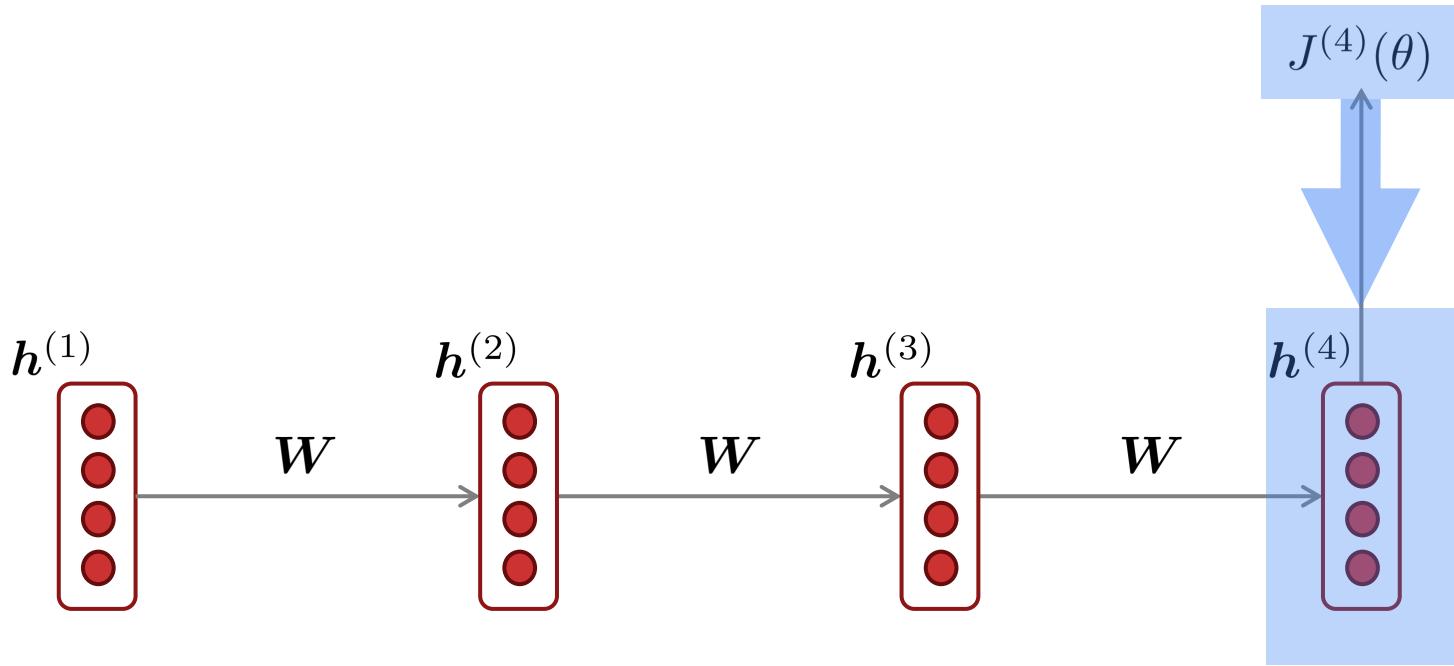
$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \times \qquad \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(3)}}$$

chain rule!

# Vanishing gradient intuition

$J^{(4)}(\theta)$

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

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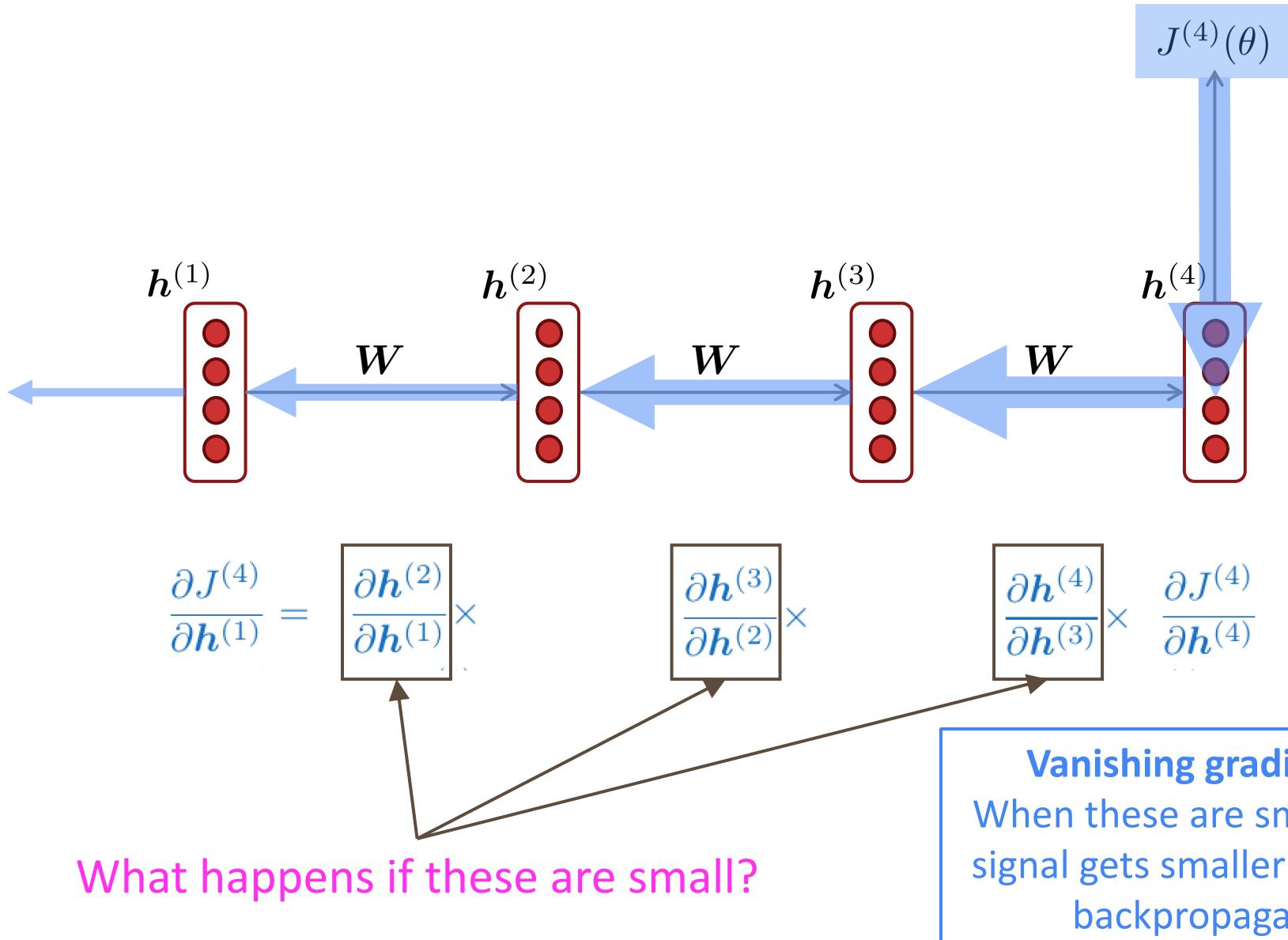
$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \quad \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \times \qquad\qquad \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \times \qquad\qquad \frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}}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

chain rule!

# Vanishing gradient intuition

$$J^{(4)}(\theta)$$

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

$$W \qquad W \qquad W$$

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

What happens if these are small?

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

# Vanishing gradient proof sketch (linear case)

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

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

$$\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} = \mathrm{diag}\left(\sigma'\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)\right) \boldsymbol{W}_h \qquad \text{(chain rule)}$$

$$= \boldsymbol{I} \; \boldsymbol{W}_h = \boldsymbol{W}_h$$

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

$$\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}} = \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \prod_{j < t \le i} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} \qquad \text{(chain rule)}$$

$$= \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \prod_{j < t \le i} \boldsymbol{W}_h = \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \boxed{\boldsymbol{W}_h^{\ell}} \qquad \left(\text{value of } \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}\right)$$

If $W_h$ 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: <span style="color:magenta">sufficient but not necessary</span>

$$\lambda_1, \lambda_2, \ldots, \lambda_n < 1$$
$$q_1, q_2, \cdots, q_n \text{ (eigenvectors)}$$
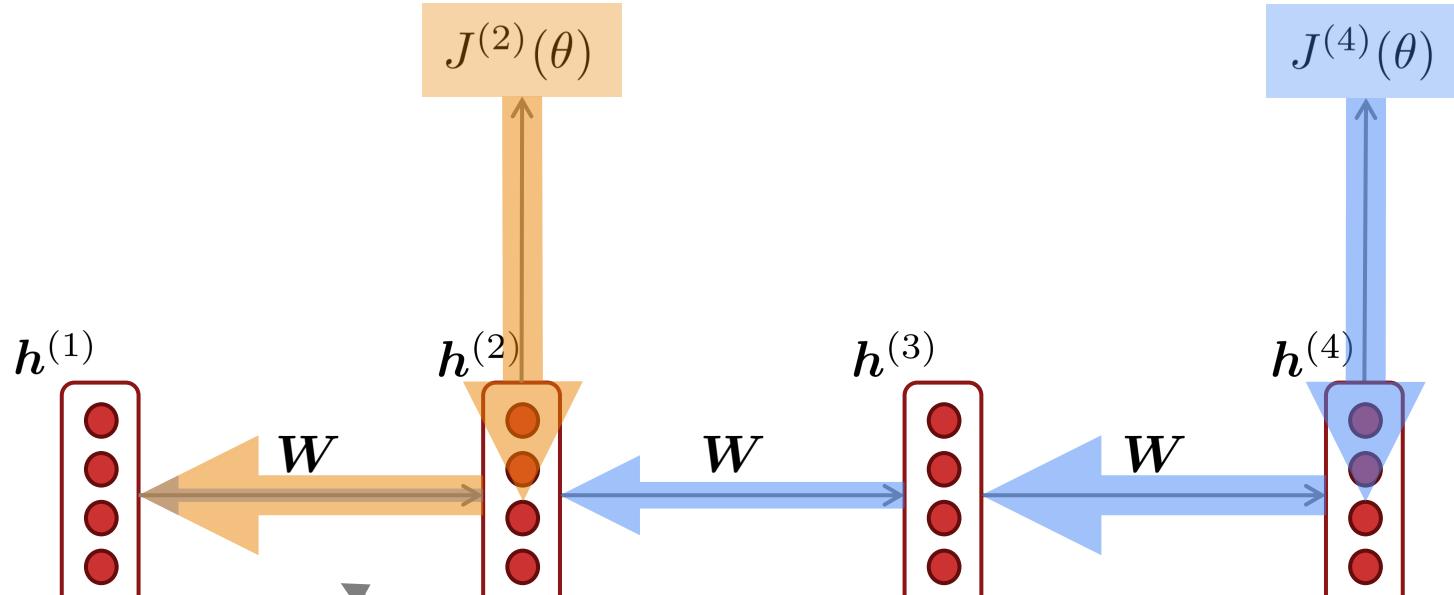
- We can write $\frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} W_h^\ell$ using the eigenvectors of $W_h$ as a basis:

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

<span style="color:magenta">Approaches 0 as $\ell$ grows, so gradient vanishes</span>

- 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 $\sigma$

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?



$J^{(2)}(\theta)$

$J^{(4)}(\theta)$

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

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

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 <span style="color:magenta">model the dependency</span> between *"tickets"* on the 7th step and the target word *"tickets"* at the end.

- But if gradient is small, the model <span style="color:magenta">can't learn this dependency</span>
  - So, the model is <span style="color:magenta">unable to predict similar long-distance dependencies</span> at test time

38

# 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

$$\hat{g} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$$

$$\textbf{if } \|\hat{g}\| \geq threshold \textbf{ then}$$

$$\hat{g} \leftarrow \frac{threshold}{\|\hat{g}\|} \hat{g}$$

$$\textbf{end if}$$

- **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

40

**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

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

- How about a RNN with separate memory?

# 4. Long Short-Term Memory RNNs (LSTMs)

- A type of RNN proposed by Hochreiter and Schmidhuber in 1997 as a solution to the vanishing gradients problem.

  - Everyone cites that paper but really a crucial part of the modern LSTM is from Gers et al. (2000) 💜

- On step $t$, there is a hidden state $\boldsymbol{h}^{(t)}$ *and* a cell state $\boldsymbol{c}^{(t)}$

  - Both are vectors length $n$

  - The cell stores long-term information

  - The LSTM can read, erase, and write information from the cell

    - The cell becomes conceptually rather like RAM in a computer

- The selection of which information is erased/written/read is controlled by three corresponding gates

  - The gates are also vectors length $n$

  - On each timestep, each element of the gates can be open (1), closed (0), or somewhere in-between

  - The gates are dynamic: their value is computed based on the current context

"Long short-term memory", Hochreiter and Schmidhuber, 1997. https://www.bioinf.jku.at/publications/older/2604.pdf
"Learning to Forget: Continual Prediction with LSTM", Gers, Schmidhuber, and Cummins, 2000. https://dl.acm.org/doi/10.1162/089976600300015015

# Long Short-Term Memory (LSTM)

We have a sequence of inputs $x^{(t)}$, and we will compute a sequence of hidden states $h^{(t)}$ and cell states $c^{(t)}$. On timestep $t$:

**Sigmoid function**: all gate values are between 0 and 1

**Forget gate:** controls what is kept vs forgotten, from previous cell state

**Input gate:** controls what parts of the new cell content are written to cell

**Output gate:** controls what parts of cell are output to hidden state

$$f^{(t)} = \sigma\left(W_f h^{(t-1)} + U_f x^{(t)} + b_f\right)$$

$$i^{(t)} = \sigma\left(W_i h^{(t-1)} + U_i x^{(t)} + b_i\right)$$

$$o^{(t)} = \sigma\left(W_o h^{(t-1)} + U_o x^{(t)} + b_o\right)$$

**New cell content:** this is the new content to be written to the cell

**Cell state**: erase ("forget") some content from last cell state, and write ("input") some new cell content

**Hidden state**: read ("output") some content from the cell

$$\tilde{c}^{(t)} = \tanh\left(W_c h^{(t-1)} + U_c x^{(t)} + b_c\right)$$

$$c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \tilde{c}^{(t)}$$
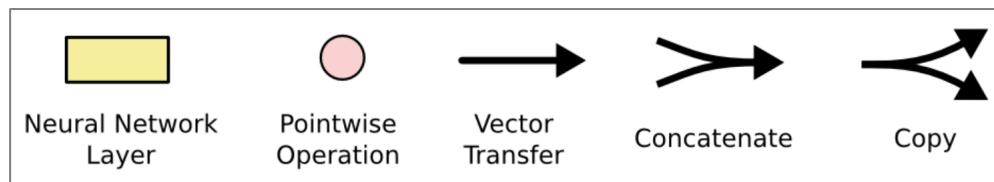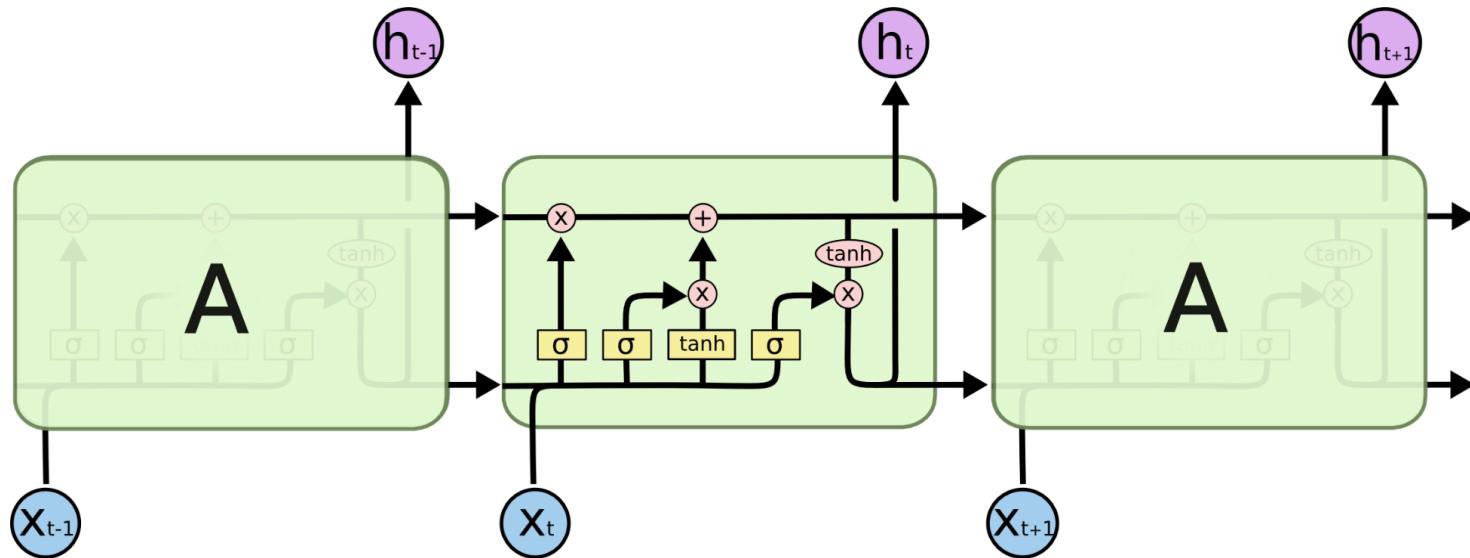
$$h^{(t)} = o^{(t)} \circ \tanh c^{(t)}$$

All these are vectors of same length $n$

Gates are applied using element-wise (or Hadamard) product: $\odot$
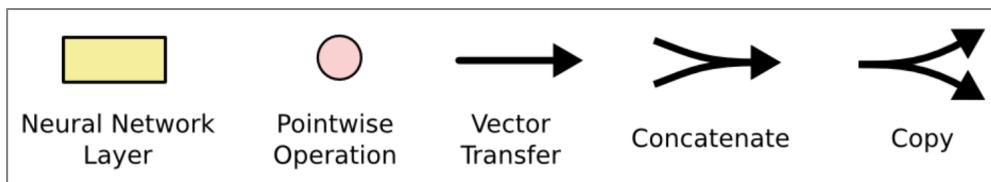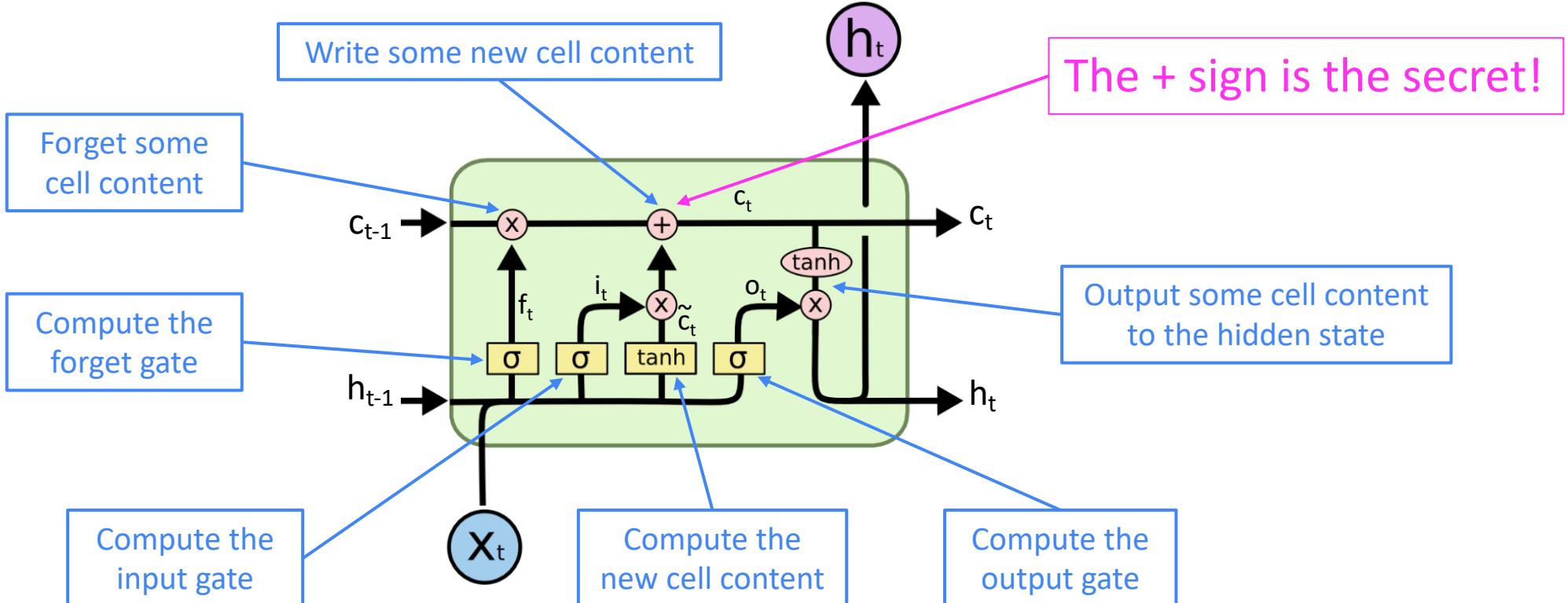
43

# Long Short-Term Memory (LSTM)

You can think of the LSTM equations visually like this:

**Source:** http://colah.github.io/posts/2015-08-Understanding-LSTMs/

# Long Short-Term Memory (LSTM)

You can think of the LSTM equations visually like this:



Write some new cell content

Forget some cell content

The + sign is the secret!

Compute the forget gate

Output some cell content to the hidden state

Compute the input gate

Compute the new cell content

Compute the output gate

Neural Network Layer • Pointwise Operation → Vector Transfer ⤳ Concatenate ⤳ Copy

# How does LSTM solve vanishing gradients?

- The LSTM architecture makes it easier for the RNN to preserve information over many timesteps

  - e.g., if the forget gate is set to 1 for a cell dimension and the input gate set to 0, then the information of that cell is preserved indefinitely.

  - In contrast, it's harder for a vanilla RNN to learn a recurrent weight matrix $W_h$ that preserves info in the hidden state

  - In practice, you get about 100 timesteps rather than about 7

- LSTM doesn't *guarantee* that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies

# LSTMs: real-world success

- In 2013–2015, LSTMs started achieving state-of-the-art results
  - Successful tasks include handwriting recognition, speech recognition, machine translation, parsing, and image captioning, as well as language models
  - LSTMs became the dominant approach for most NLP tasks

- Now (2021), other approaches (e.g., Transformers) have become dominant for many tasks
  - For example, in **WMT** (a Machine Translation conference + competition):
  - In WMT 2016, the summary report contains "RNN" 44 times
  - In WMT 2019: "RNN" 7 times, "Transformer" 105 times

**Source:** "Findings of the 2016 Conference on Machine Translation (WMT16)", Bojar et al. 2016, http://www.statmt.org/wmt16/pdf/W16-2301.pdf
**Source:** "Findings of the 2018 Conference on Machine Translation (WMT18)", Bojar et al. 2018, http://www.statmt.org/wmt18/pdf/WMT028.pdf
**Source:** "Findings of the 2019Conference on Machine Translation (WMT19)", Barrault et al. 2019, http://www.statmt.org/wmt18/pdf/WMT028.pdf

# Is vanishing/exploding gradient just a RNN problem?

- No! It can be a problem for all neural architectures (including feed-forward and convolutional), especially very deep ones.
  - Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small as it backpropagates
  - Thus, lower layers are learned very slowly (hard to train)
- Solution: lots of new deep feedforward/convolutional architectures that add more direct connections (thus allowing the gradient to flow)

For example:

- Residual connections aka "ResNet"
- Also known as skip-connections
- The identity connection preserves information by default
- This makes deep networks much easier to train
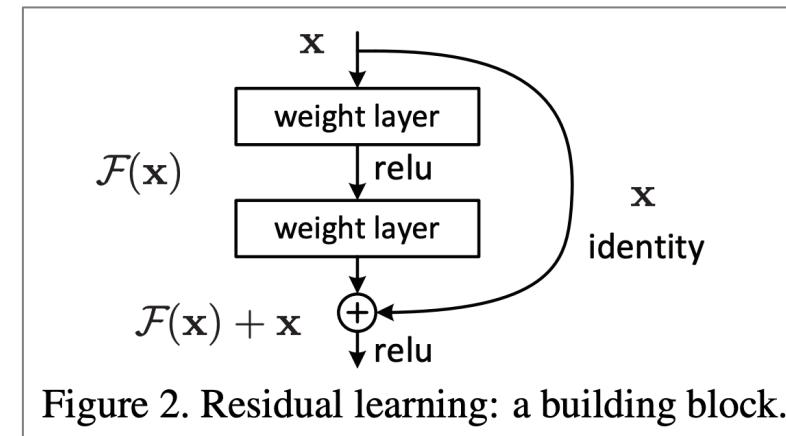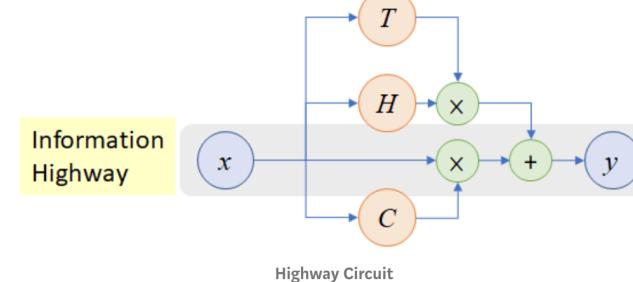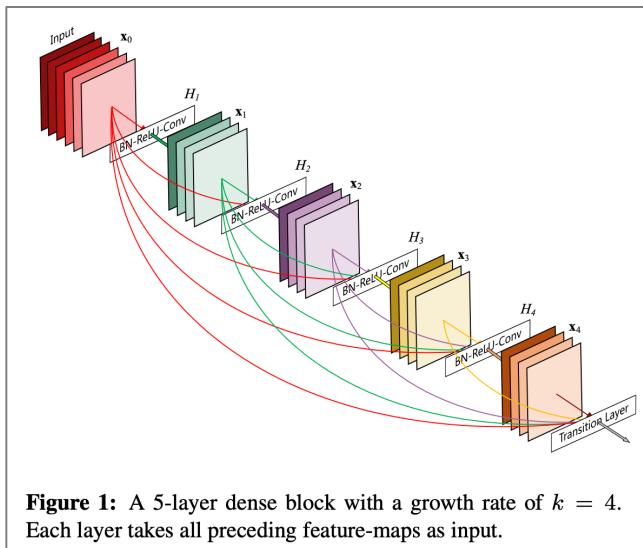


Figure 2. Residual learning: a building block.

"Deep Residual Learning for Image Recognition", He et al, 2015. https://arxiv.org/pdf/1512.03385.pdf

48

# Is vanishing/exploding gradient just a RNN problem?

- Solution: lots of new deep feedforward/convolutional architectures that add more direct connections (thus allowing the gradient to flow)

Other methods:

- Dense connections aka "DenseNet"
- Directly connect each layer to all future layers!



**Figure 1:** A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

- Highway connections aka "HighwayNet"
- Similar to residual connections, but the identity connection vs the transformation layer is controlled by a dynamic gate
- Inspired by LSTMs, but applied to deep feedforward/convolutional networks
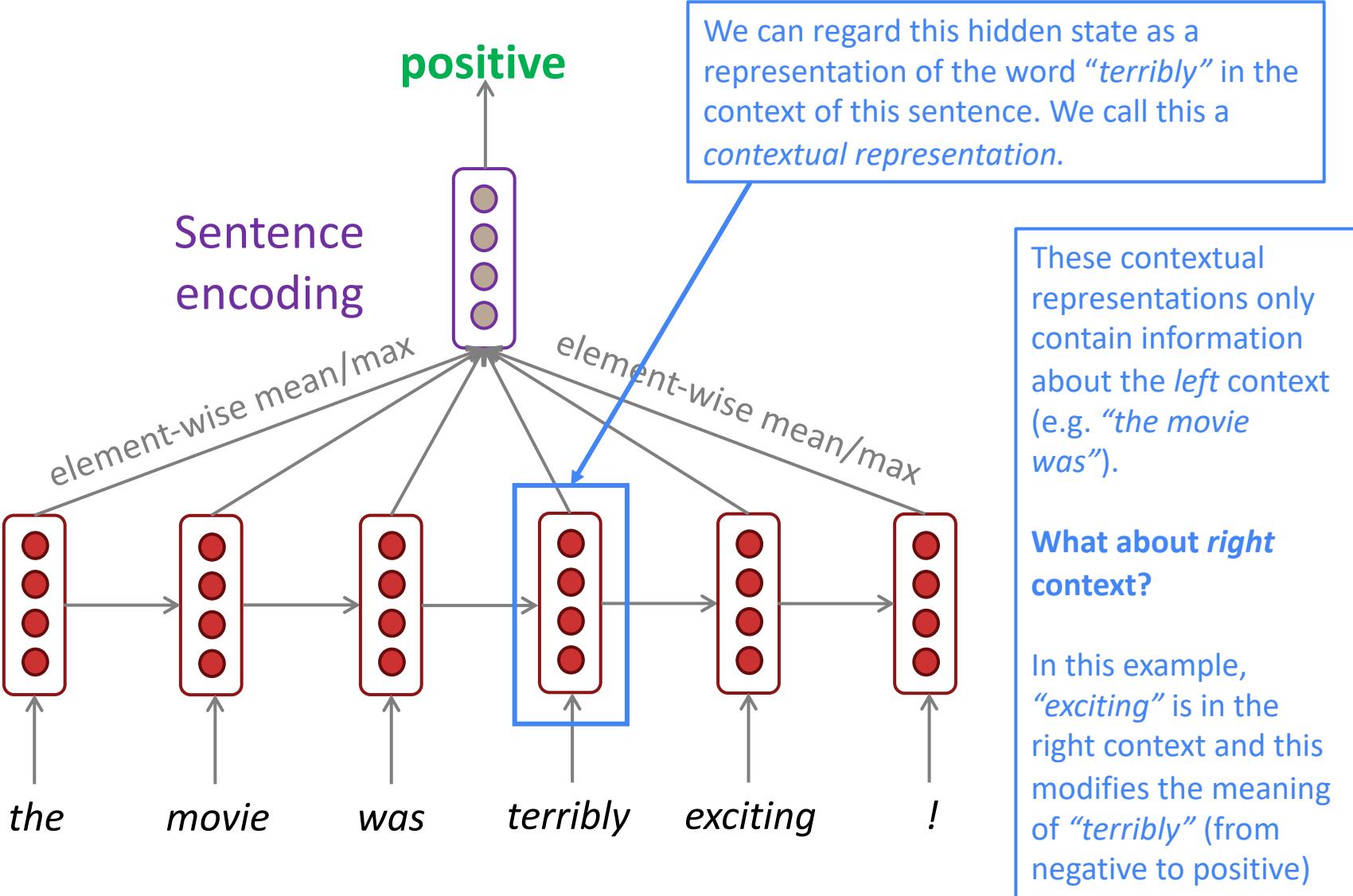


Highway Circuit

"Densely Connected Convolutional Networks", Huang et al, 2017. https://arxiv.org/pdf/1608.06993.pdf

"Highway Networks", Srivastava et al, 2015. https://arxiv.org/pdf/1505.00387.pdf

# Is vanishing/exploding gradient just a RNN problem?

- No! It can be a problem for all neural architectures (including feed-forward and convolutional), especially very deep ones.
  - Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small as it backpropagates
  - Thus, lower layers are learned very slowly (hard to train)
- Solution: lots of new deep feedforward/convolutional architectures that add more direct connections (thus allowing the gradient to flow)

- **Conclusion**: Though vanishing/exploding gradients are a general problem, RNNs are particularly unstable due to the repeated multiplication by the same weight matrix [Bengio et al, 1994]

”Learning Long-Term Dependencies with Gradient Descent is Difficult", Bengio et al. 1994, http://ai.dinfo.unifi.it/paolo//ps/tnn-94-gradient.pdf

# 5. Bidirectional and Multi-layer RNNs: motivation

Task: Sentiment Classification

**positive**

Sentence encoding

We can regard this hidden state as a representation of the word *"terribly"* in the context of this sentence. We call this a *contextual representation.*

element-wise mean/max

element-wise mean/max

the        movie        was        terribly        exciting        !

These contextual representations only contain information about the *left* context (e.g. *"the movie was"*).

**What about *right* context?**

In this example, *"exciting"* is in the right context and this modifies the meaning of *"terribly"* (from negative to positive)

# Bidirectional RNNs

This contextual representation of "terribly" has both left and right context!

Concatenated hidden states

Backward RNN

Forward RNN

*the*  *movie*  *was*  *terribly*  *exciting*  *!*

# Bidirectional RNNs

On timestep $t$:

This is a general notation to mean "compute one forward step of the RNN" – it could be a vanilla, LSTM or GRU computation.

Forward RNN $\qquad \overrightarrow{\boldsymbol{h}}^{(t)} = \boxed{\mathrm{RNN}_{\mathrm{FW}}}(\overrightarrow{\boldsymbol{h}}^{(t-1)}, \boldsymbol{x}^{(t)})$
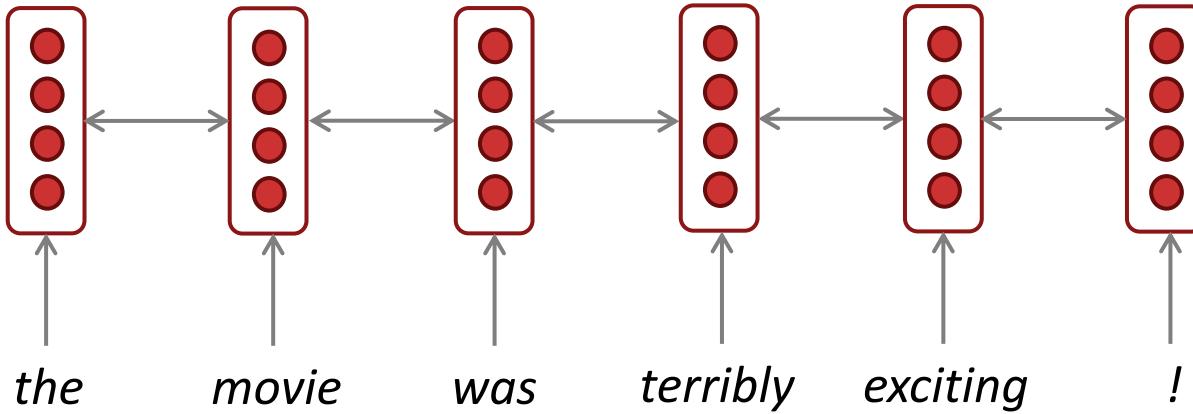
Backward RNN $\qquad \overleftarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{BW}}(\overleftarrow{\boldsymbol{h}}^{(t+1)}, \boldsymbol{x}^{(t)})$

Generally, these two RNNs have separate weights

Concatenated hidden states $\qquad \boxed{\boldsymbol{h}^{(t)}} = [\overrightarrow{\boldsymbol{h}}^{(t)}; \overleftarrow{\boldsymbol{h}}^{(t)}]$

We regard this as "the hidden state" of a bidirectional RNN. This is what we pass on to the next parts of the network.

# Bidirectional RNNs: simplified diagram



The two-way arrows indicate bidirectionality and the depicted hidden states are assumed to be the concatenated forwards+backwards states

# Bidirectional RNNs
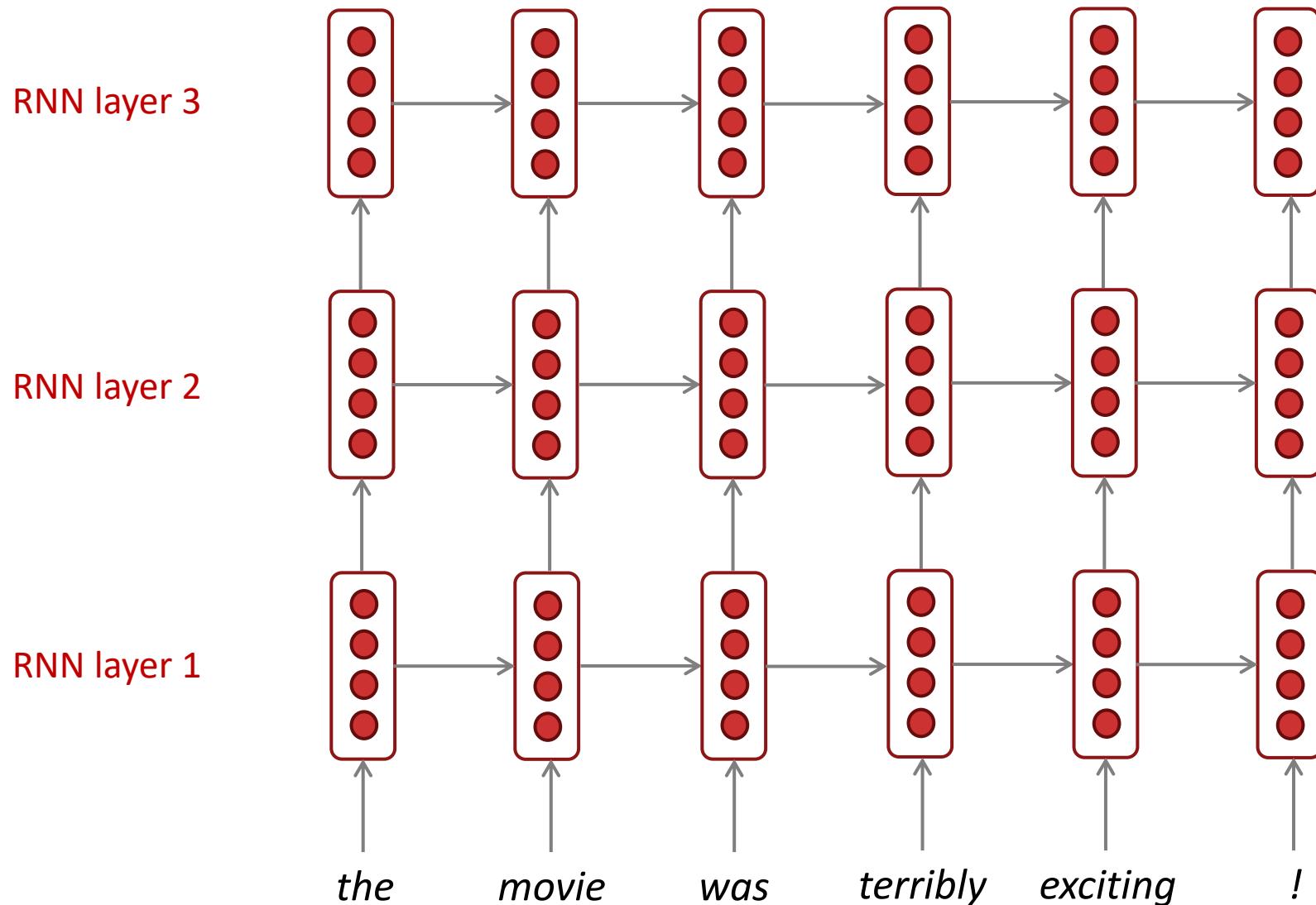
- Note: bidirectional RNNs are only applicable if you have access to the entire input sequence
  - They are **not** applicable to Language Modeling, because in LM you *only* have left context available.

- If you do have entire input sequence (e.g., any kind of encoding), bidirectionality is powerful (you should use it by default).

- For example, BERT (**Bidirectional** Encoder Representations from Transformers) is a powerful pretrained contextual representation system built on bidirectionality.
  - You will learn more about transformers include BERT in a couple of weeks!

# Multi-layer RNNs

- RNNs are already "deep" on one dimension (they unroll over many timesteps)

- We can also make them "deep" in another dimension by applying multiple RNNs – this is a multi-layer RNN.

- This allows the network to compute more complex representations
  - The lower RNNs should compute lower-level features and the higher RNNs should compute higher-level features.

- Multi-layer RNNs are also called *stacked RNNs*.

# Multi-layer RNNs

The hidden states from RNN layer *i* are the inputs to RNN layer *i*+1
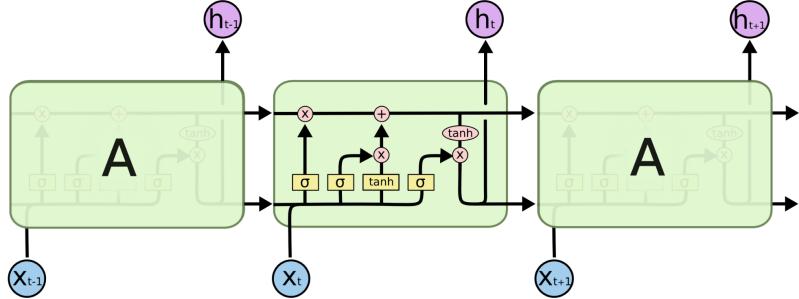
# Multi-layer RNNs in practice
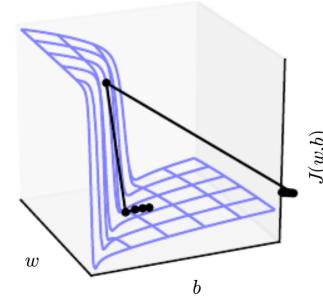
- High-performing RNNs are often multi-layer (but aren't as deep as convolutional or feed-forward networks)

- For example: In a 2017 paper, Britz et al find that for Neural Machine Translation, 2 to 4 layers is best for the encoder RNN, and 4 layers is best for the decoder RNN
  - Usually, skip-connections/dense-connections are needed to train deeper RNNs (e.g., 8 layers)

- Transformer-based networks (e.g., BERT) are usually deeper, like 12 or 24 layers.
  - You will learn about Transformers later; they have a lot of skipping-like connections

"Massive Exploration of Neural Machine Translation Architecutres", Britz et al, 2017. https://arxiv.org/pdf/1703.03906.pdf

# In summary
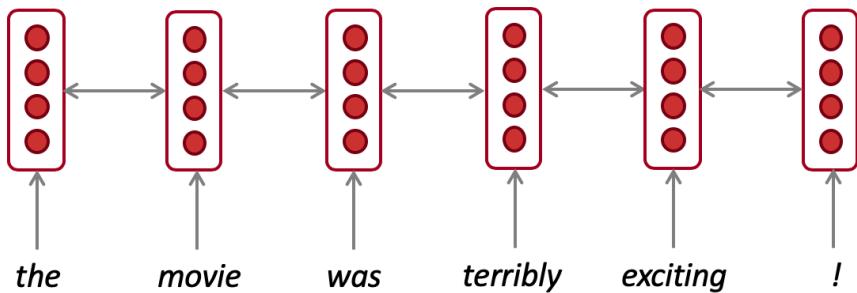
Lots of new information today! What are some of the practical takeaways?
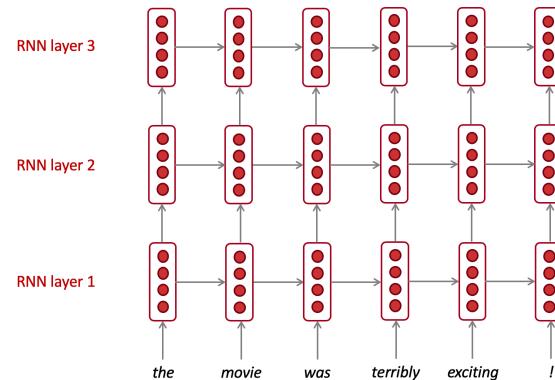


## 1. LSTMs are powerful



## 2. Clip your gradients



## 3. Use bidirectionality when possible



## 4. Multi-layer RNNs are more powerful, but you might need skip connections if it's deep