Natural Language Processing with Deep Learning

IFT6289, Winter 2022

Lecture 3: Language Modeling and RNN

Bang Liu

2 Lecture outline

1. Language Modeling
2. Neural Language Modeling
3. Recurrent Neural Networks (RNN)
4. RNN Variants

3 Certain Slides Adapted From or Referred To…

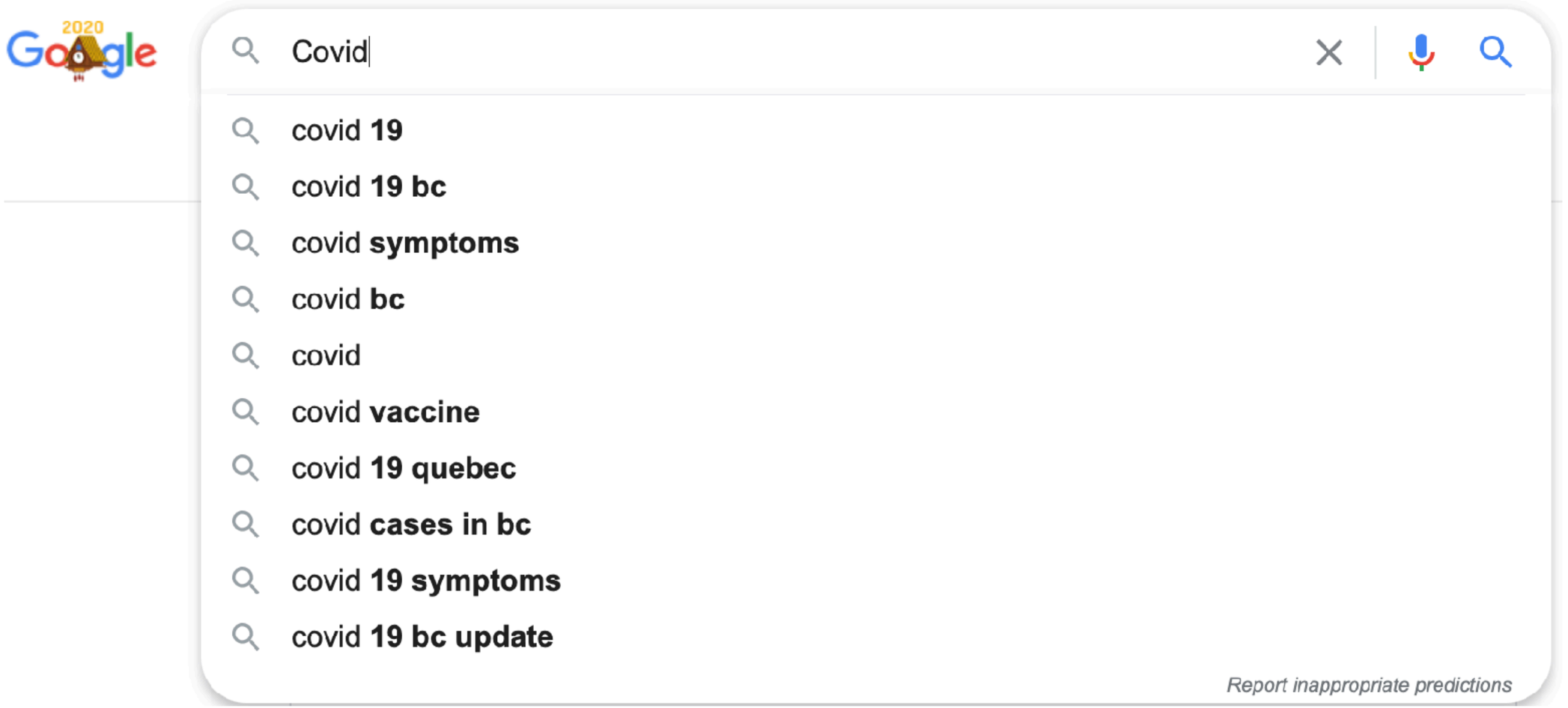
- **Stanford CS224n - Natural Language Processing with Deep Learning, Chris Manning**
  - Winter 2020: http://web.stanford.edu/class/cs224n/, lecture 6, 7

- https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn?source=sl_frs_nav_playlist_video_clicked

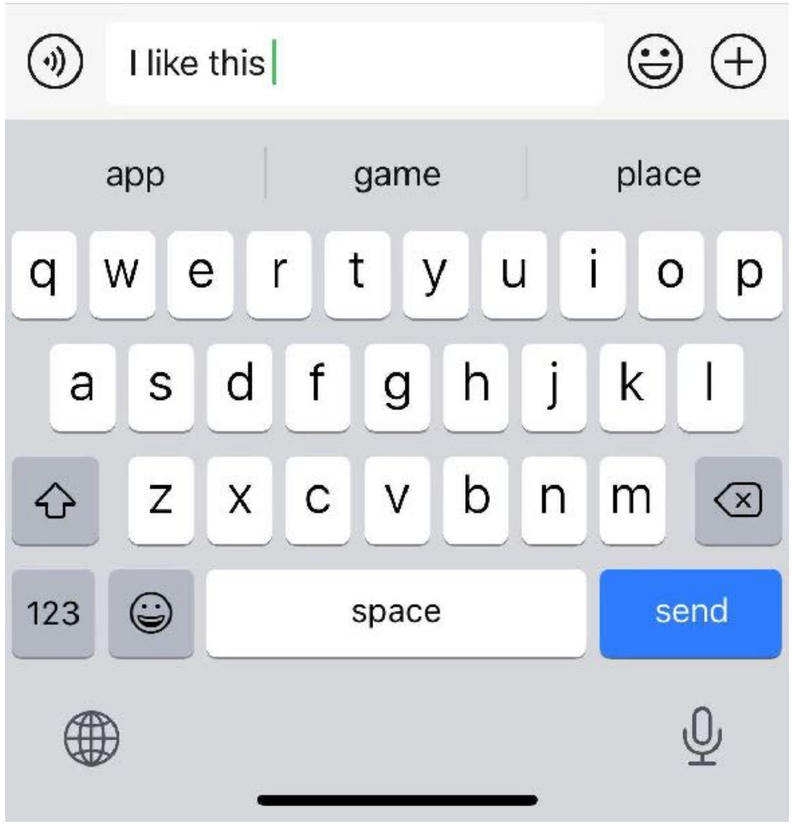
- https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Language Modeling

5 You Use Language Models Everyday

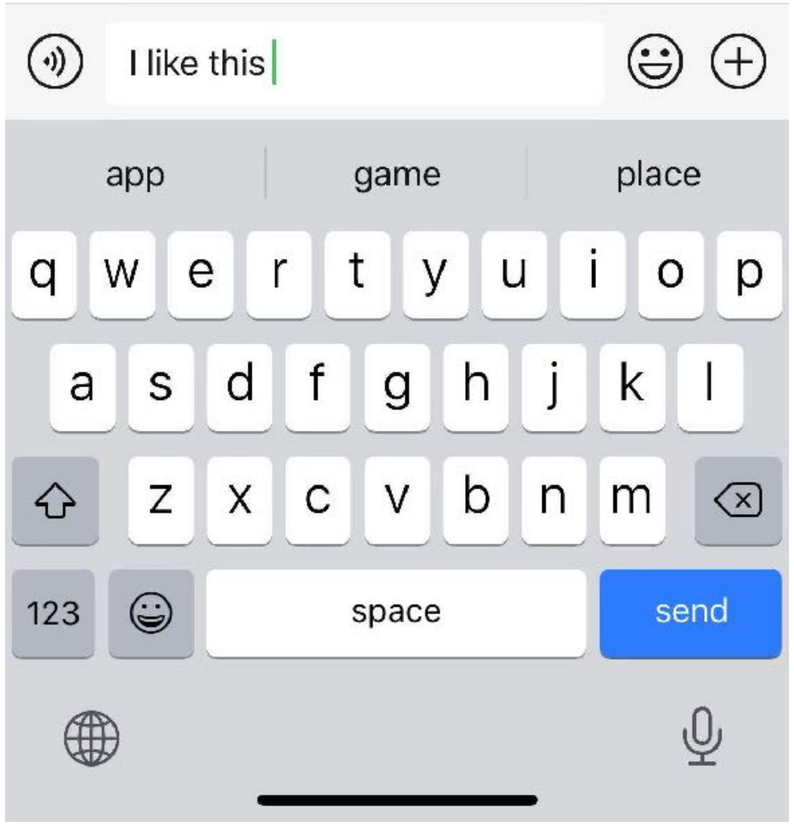


6 You Use Language Models Everyday



7 What is Language Modeling

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



What is Language Modeling

- **Language Model (LM)** is a system that estimates the probability of a piece of text (a word sequence)

Given a sequence of words:

$$x^{(1)}, x^{(2)}, \cdots, x^{(T)}$$

where $x^{(i)}$ can be any word in a vocabulary

$$V = \{w_1, \cdots, w_{|V|}\}$$

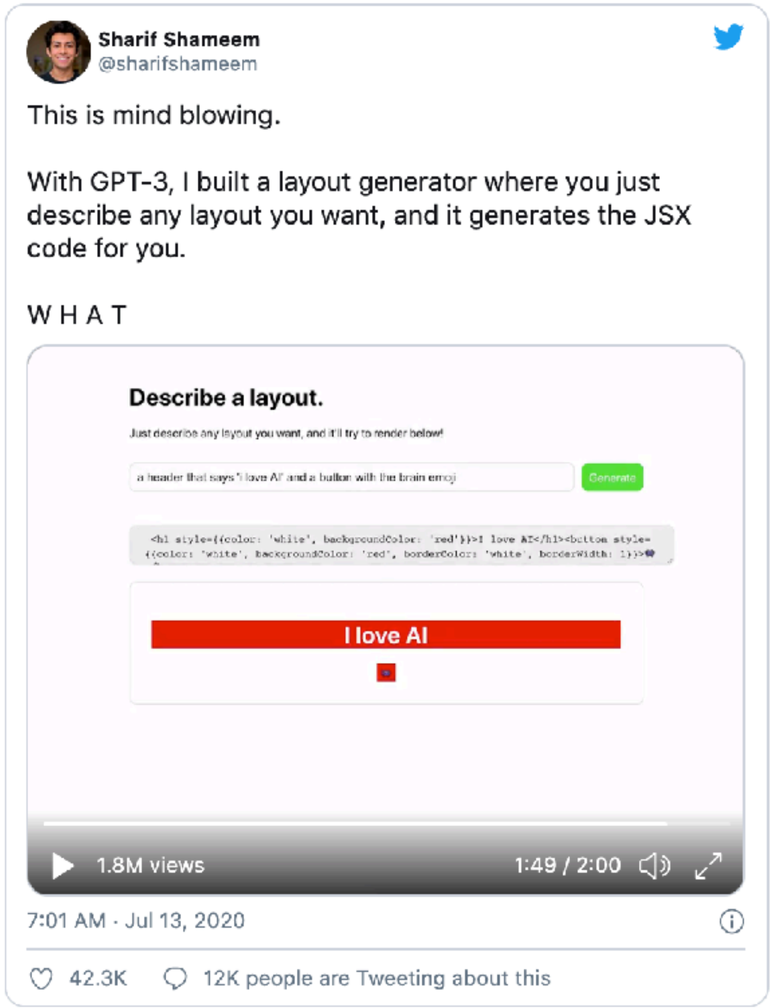
Estimate the probability:

$$P(x^{(1)}, x^{(2)}, \cdots, x^{(T)}) = P(x^{(1)}) \times P(x^{(2)}|x^{(1)}) \times \cdots \times P(x^{(T)}|x^{(T-1)}, \cdots, x^{(1)})$$

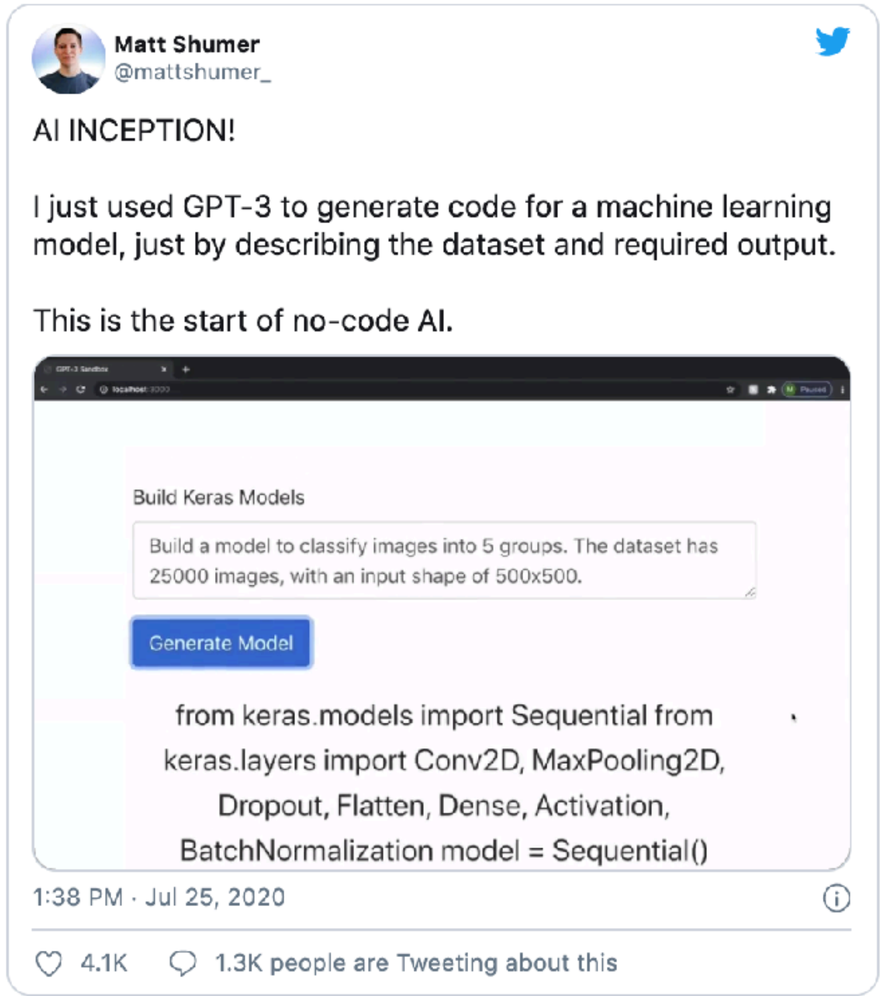
$$= \prod_{t=1}^{T} P(x^{(t)}|x^{(t-1)}, \cdots, x^{(1)})$$

This is what a LM provides

9 Language Models are Changing Everything



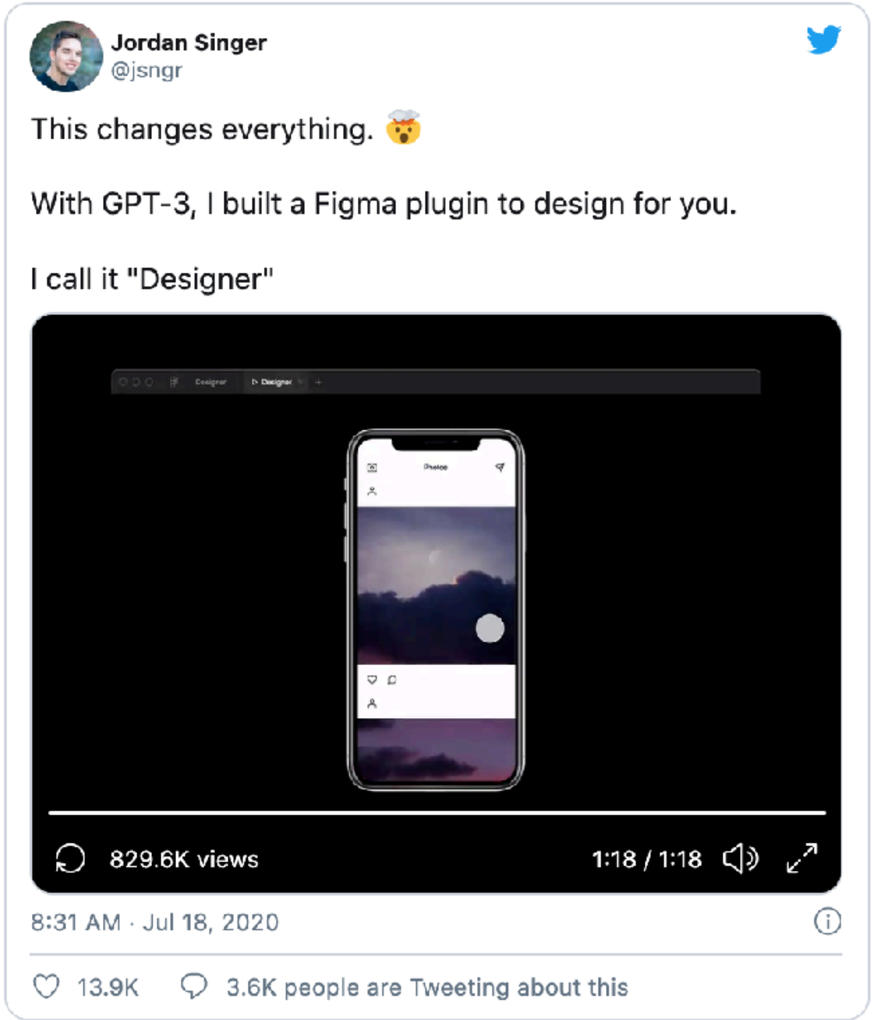
Code



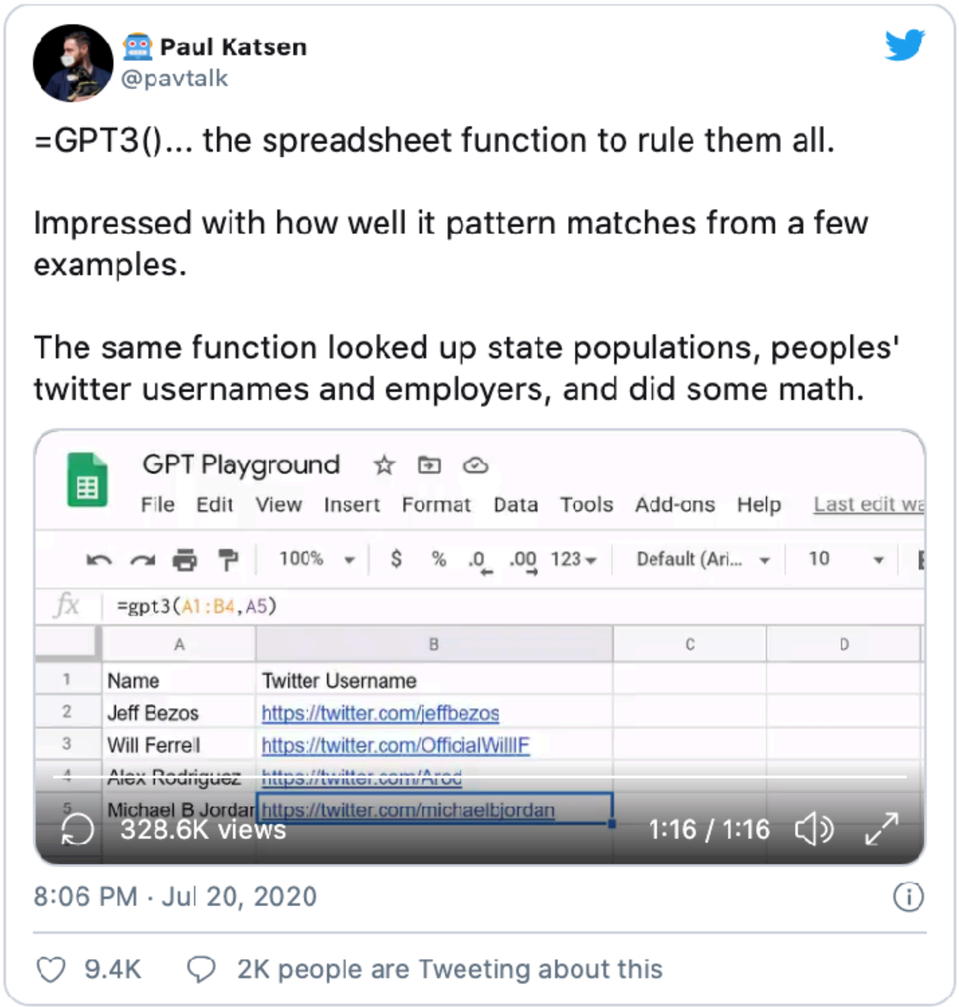
Machine Learning Code

https://machinelearningtokyo.com/2020/07/26/10-cool-gpt-3-demos/

10 Language Models are Changing Everything



Design



Spreedsheets

https://machinelearningtokyo.com/2020/07/26/10-cool-gpt-3-demos/

How to Learn a Language Model?

N-gram Language Models

N-gram Language Models

- A **n-gram** is a chunk of n consecutive words.

  Given a piece of text: "the students opened their …"
  - **uni**grams: "the", "students", "opened", "their"
  - **bi**grams: "the students", "students opened", "opened their"
  - **tri**grams: "the students opened", "students opened their"
  - **4**-grams: "the students opened their"
- **Idea (before deep learning)**: Collect statistics about how frequent different n-grams are, and use these to predict next word.

13 N-gram Language Models

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

n-1 words

$$
\begin{aligned}
P(x^{(t+1)}|x^{(t)},\cdots,x^{(1)}) &= P(x^{(t+1)}|x^{(t)},\cdots,x^{(t-n+2)}) && \text{(assumption)} \\
&= \frac{P(x^{(t+1)},x^{(t)},\cdots,x^{(t-n+2)})}{P(x^{(t)},\cdots,x^{(t-n+2)})} && \text{(conditional probability)} \\
&\approx \frac{\text{count}(x^{(t+1)},x^{(t)},\cdots,x^{(t-n+2)})}{\text{count}(x^{(t)},\cdots,x^{(t-n+2)})} && \text{(statistical approximation)}
\end{aligned}
$$

- We get these n-gram and (n-1)-gram probabilities by **counting** them in some large corpus of text!

N-gram Language Models: Example

- Suppose we are learning a 4-gram Language Model

~~as the proctor started the clock, the stu~~dents opened their ____

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

- For example, suppose that in the corpus:
  - "students opened their" occurred 1000 times
  - If "students opened their books" occurred 400 times

$$P(\text{books}|\text{students opened their}) = 0.4$$

  - "students opened their exams" occurred 100 times

$$P(\text{exams}|\text{students opened their}) = 0.1$$

Shall we discard the context "proctor"?

Sparsity Problem with N-gram Language Models

Sparsity Problem 1

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

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

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

Sparsity Problem 2

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

(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 Problem with N-gram Language Models

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

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

Increasing *n* or increasing corpus increases model size!

N-gram Language Models in Practice

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

Business and financial news

today the _____

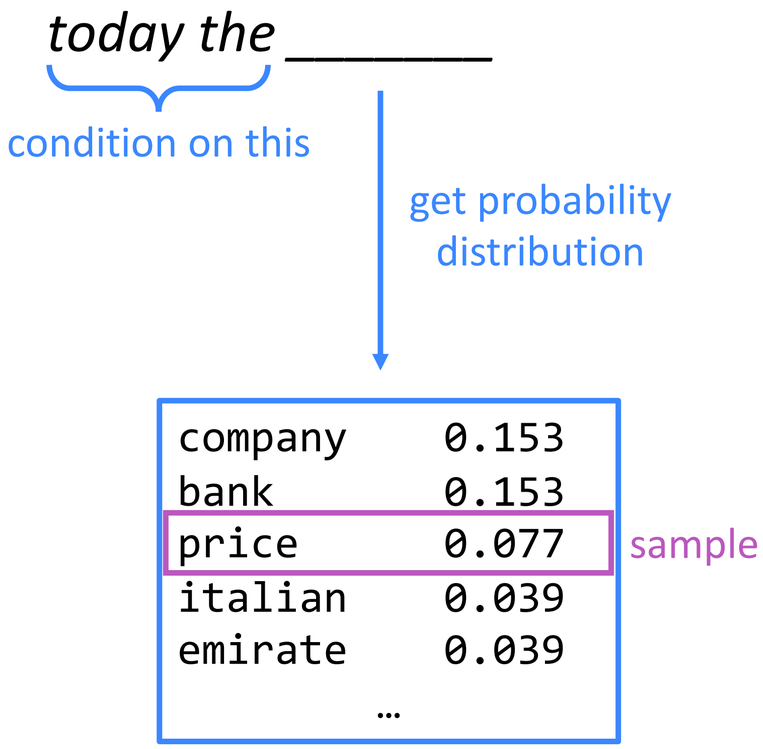
get probability distribution

company	0.153
bank	0.153
price	0.077
italian	0.039
emirate	0.039
…	

Sparsity problem:
not much granularity
in the probability
distribution

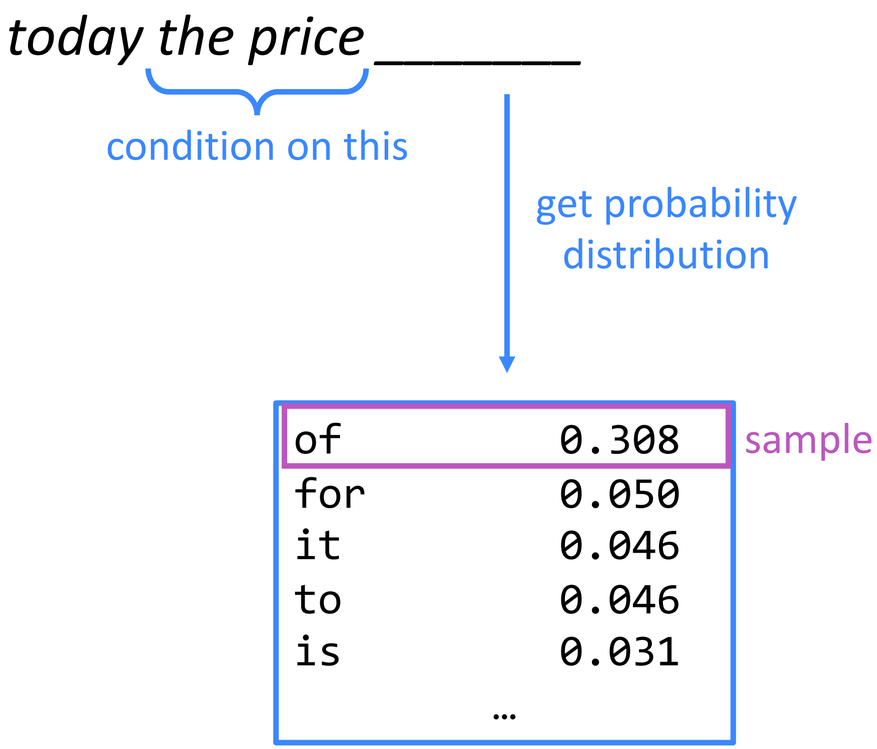
Generating Text with a N-gram Language Model

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



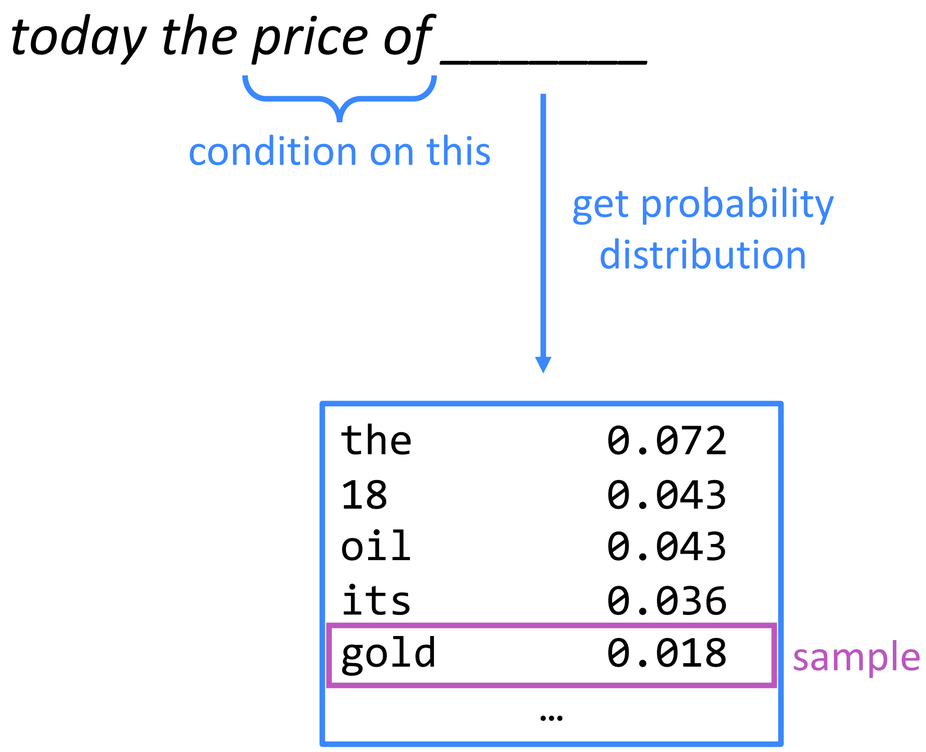
Generating Text with a N-gram Language Model

You can also use a Language Model to generate text.



Generating Text with a N-gram Language Model

You can also use a Language Model to generate text.



Generating Text with a N-gram Language Model

You can also use a Language Model to generate text.

today the price of gold _____

Generating Text with a N-gram Language Model

You can also use a Language Model to generate text.

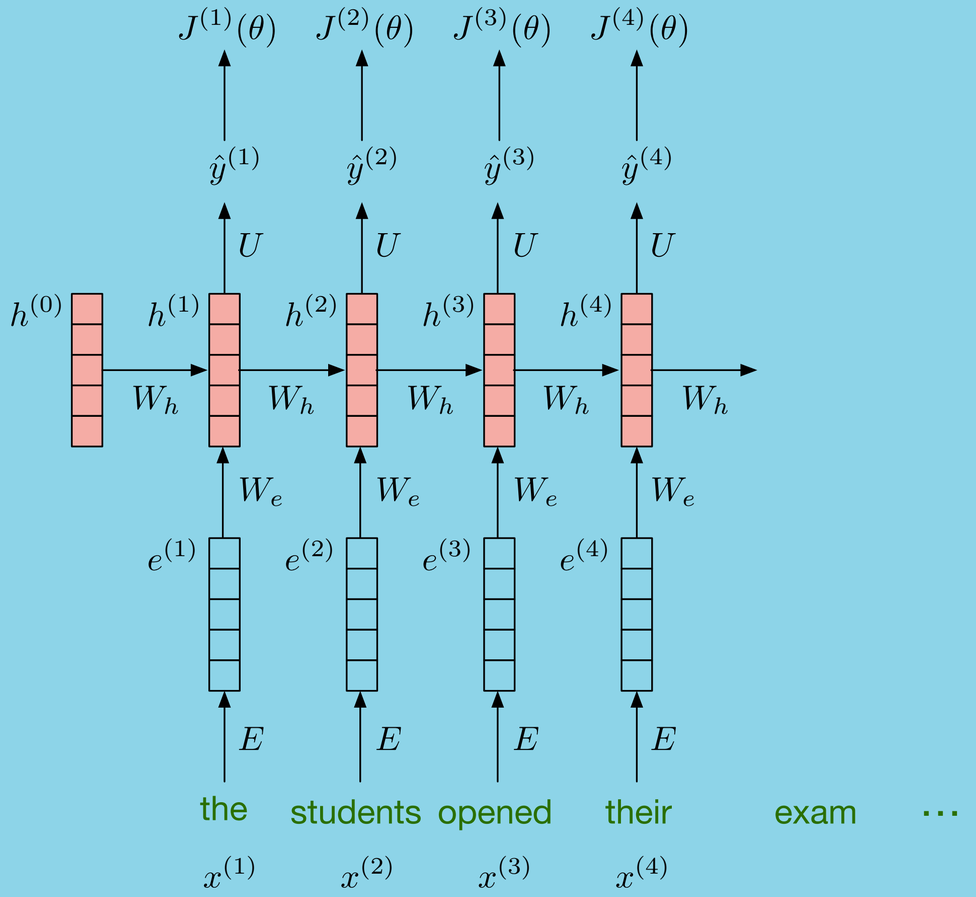
today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share .

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…

Neural Language Modeling



A Fixed-Window Neural Language Model

- Estimate $P(x^{(t+1)}|x^{(t)},\cdots,x^{(t-n+2)})$ not from **count**, but from **neural network prediction**

output distribution

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

hidden layer

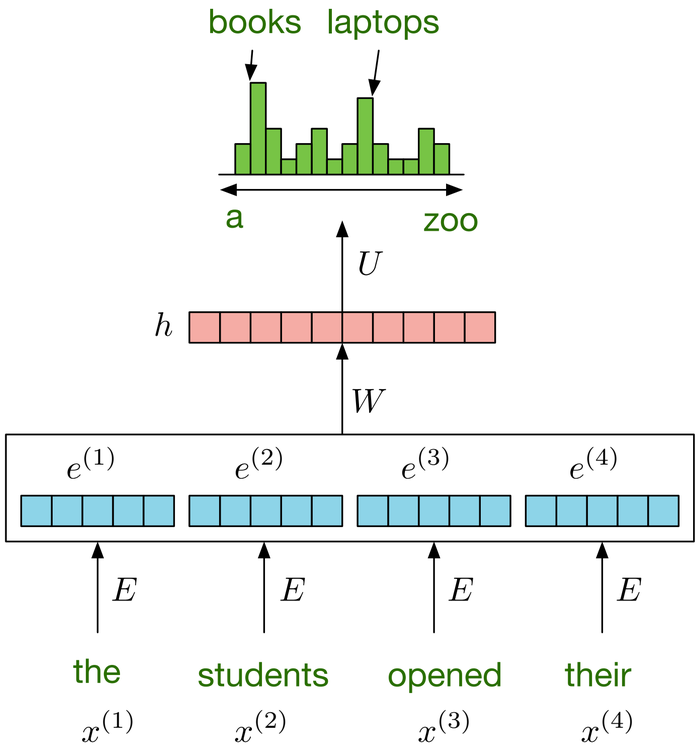
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

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

words / one-hot vectors

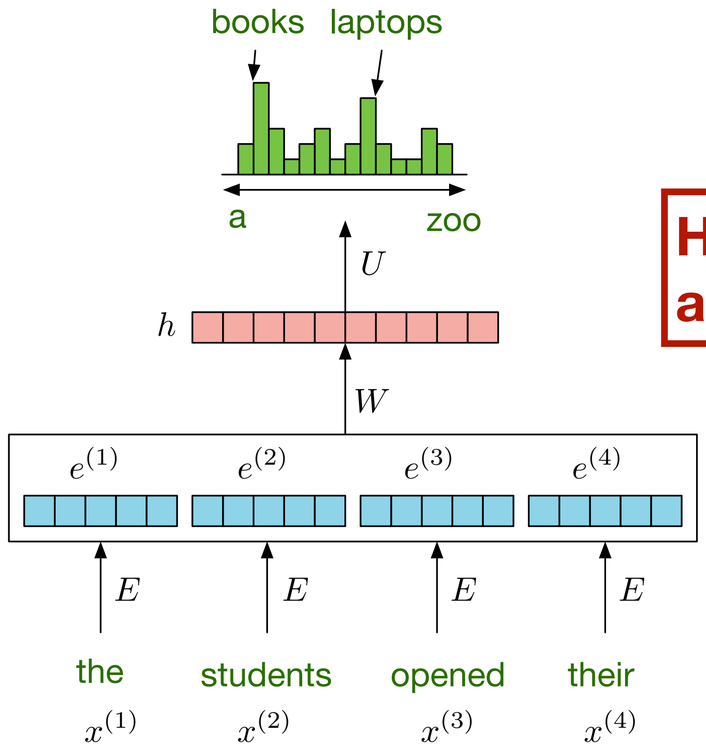
$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$



A Fixed-Window Neural Language Model

- **Improvements** to n-gram LM
  - No sparsity problem
  - Don't need to store all observed n-grams

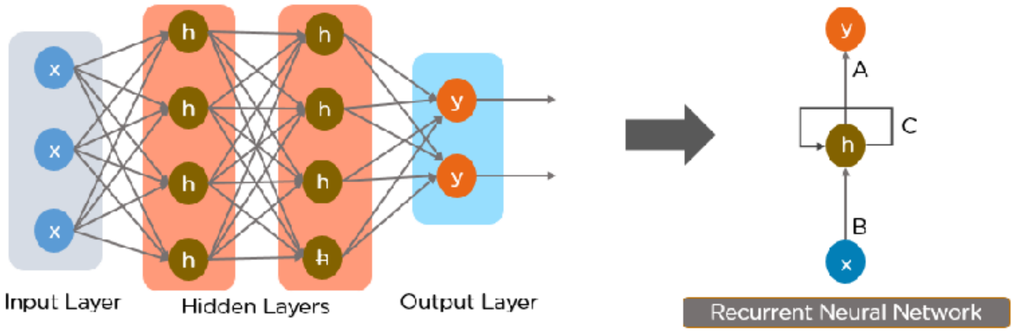
- **Remaining problems**
  - Fixed window is too small
  - Enlarging window enlarges W
  - Words in different positions are multiplied by different W, no symmetry in how the inputs are processed.

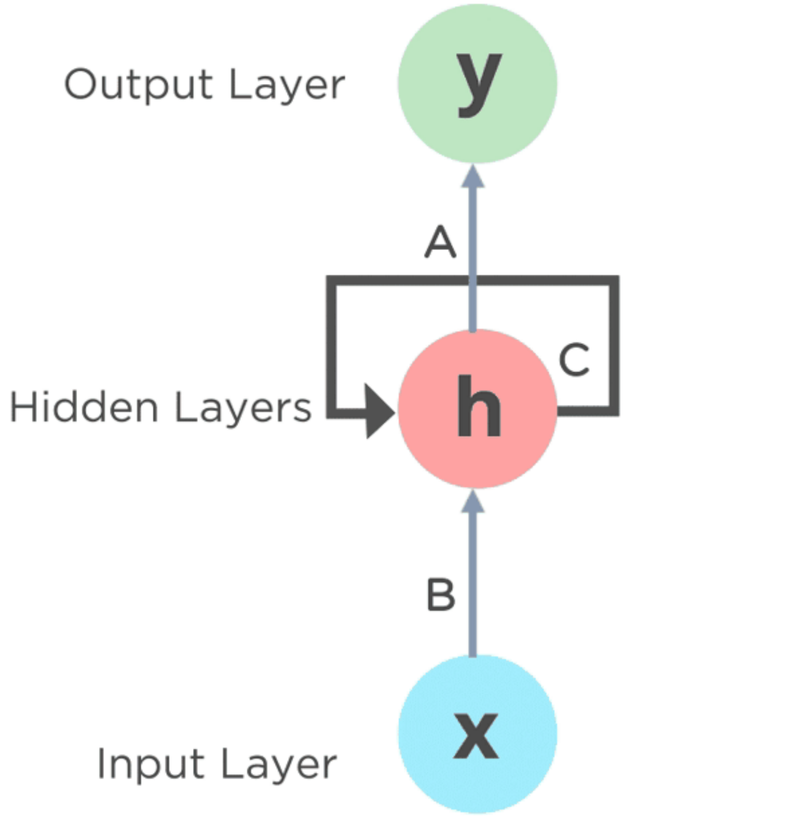


How to process any length input?

Recurrent Neural Networks

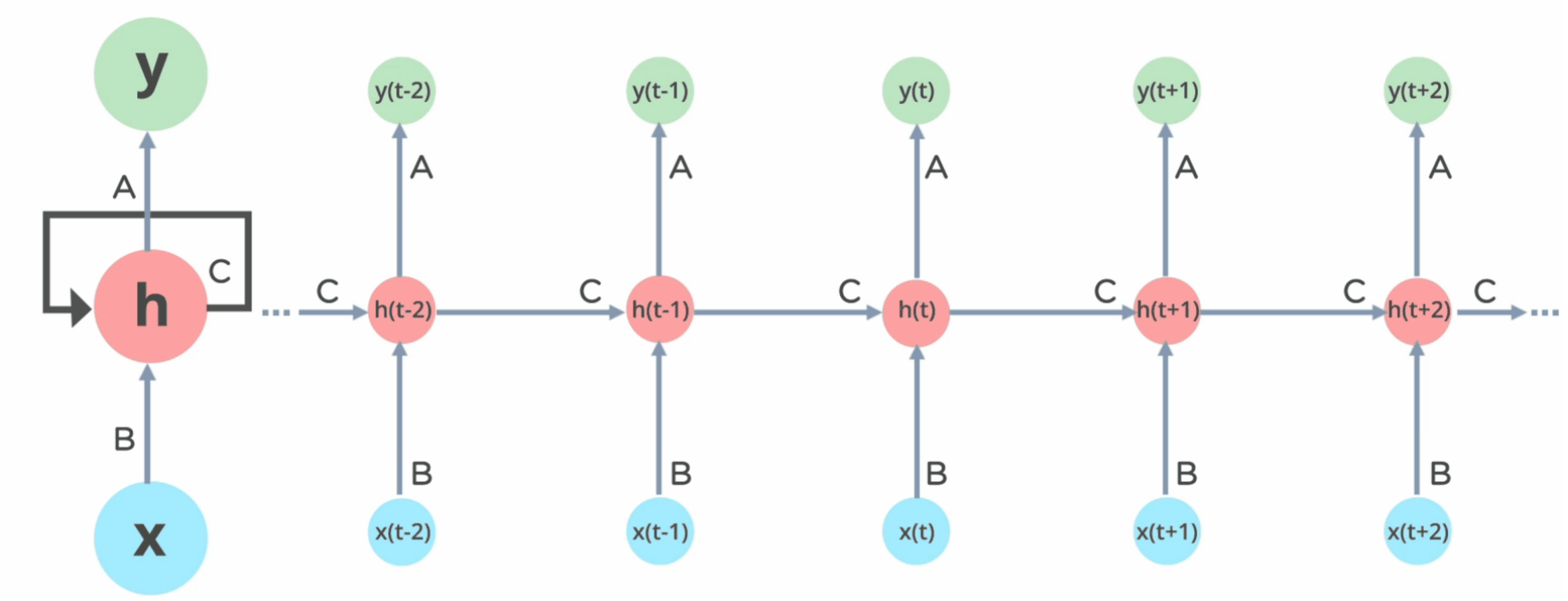
27 Recurrent Neural Networks (RNN)





https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn?source=sl_frs_nav_playlist_video_clicked

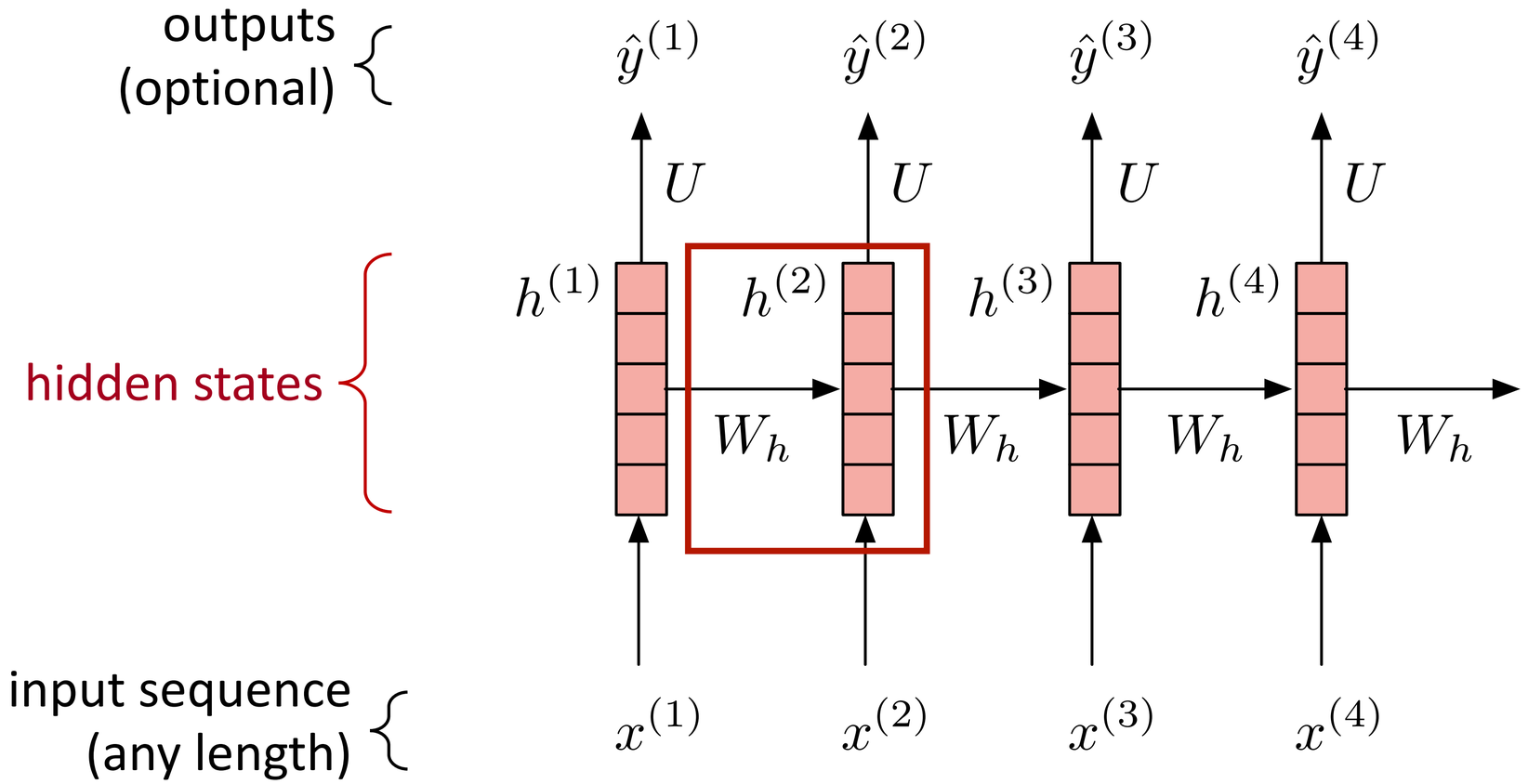
Recurrent Neural Networks (RNN)



https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn?source=sl_frs_nav_playlist_video_clicked

Recurrent Neural Networks (RNN)

- RNN is a family of neural architectures



Core idea: Apply the same weights W repeatedly

A Simple RNN Language Model

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

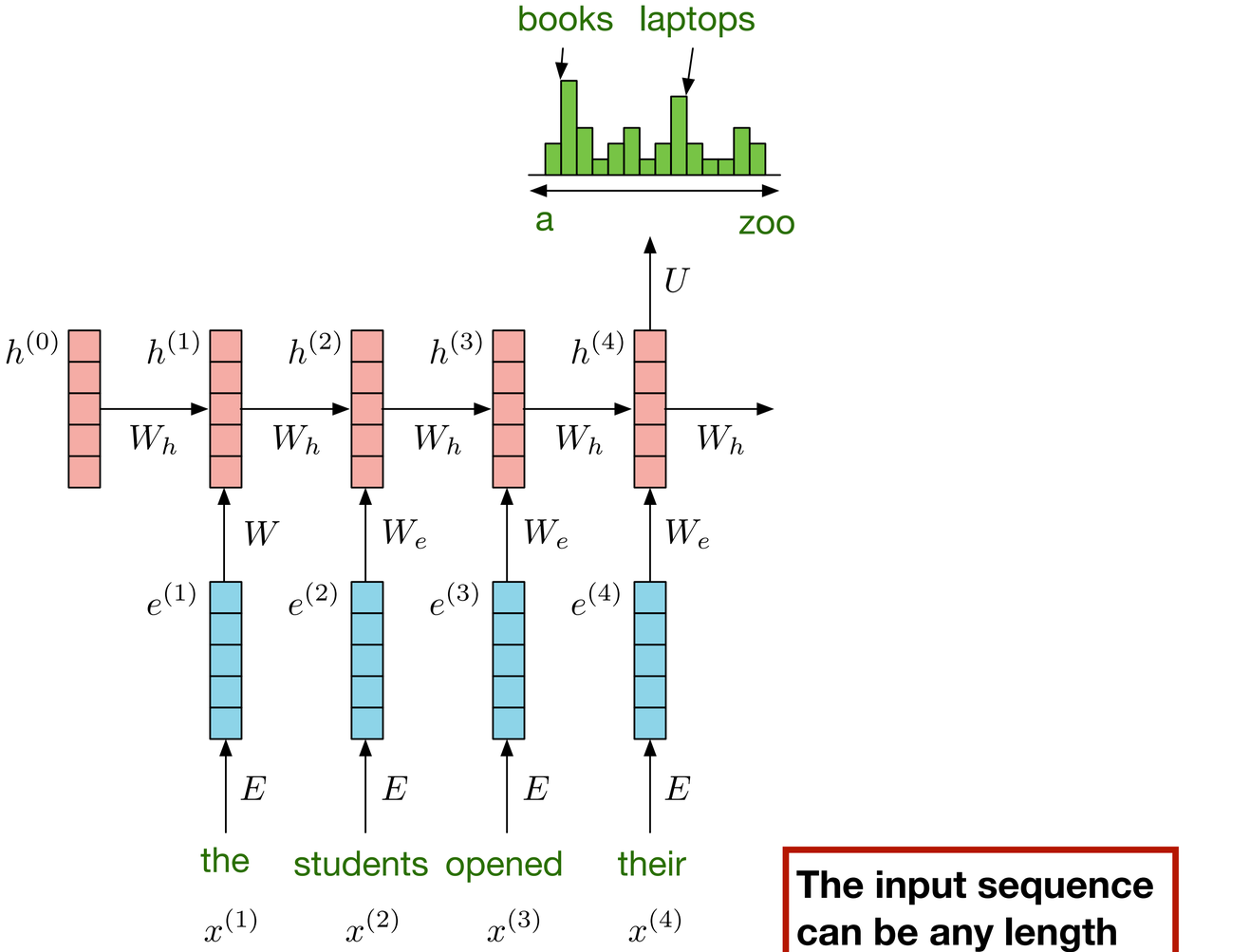
word embeddings

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

words / one-hot vectors

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

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

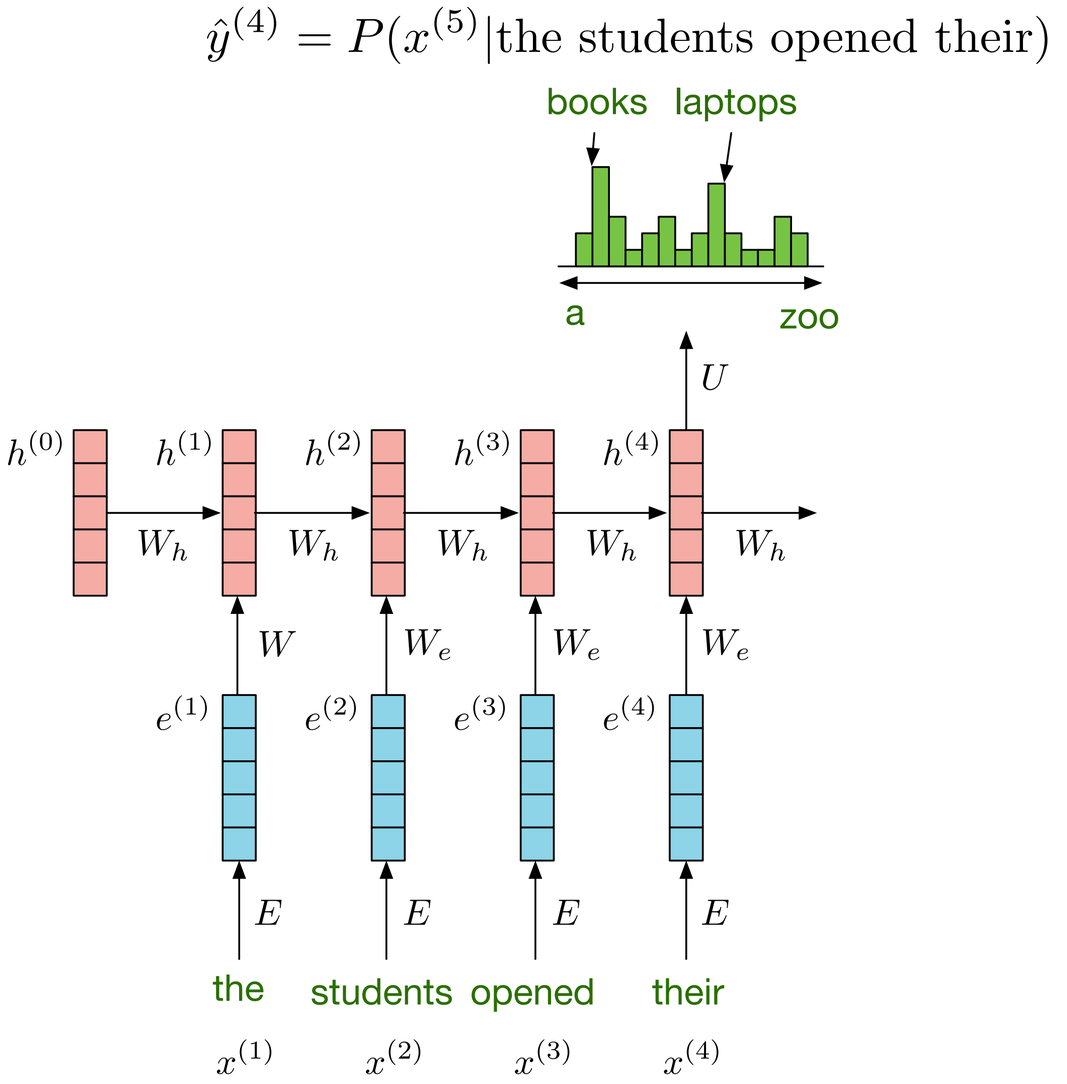


The input sequence can be any length

RNN Language Models

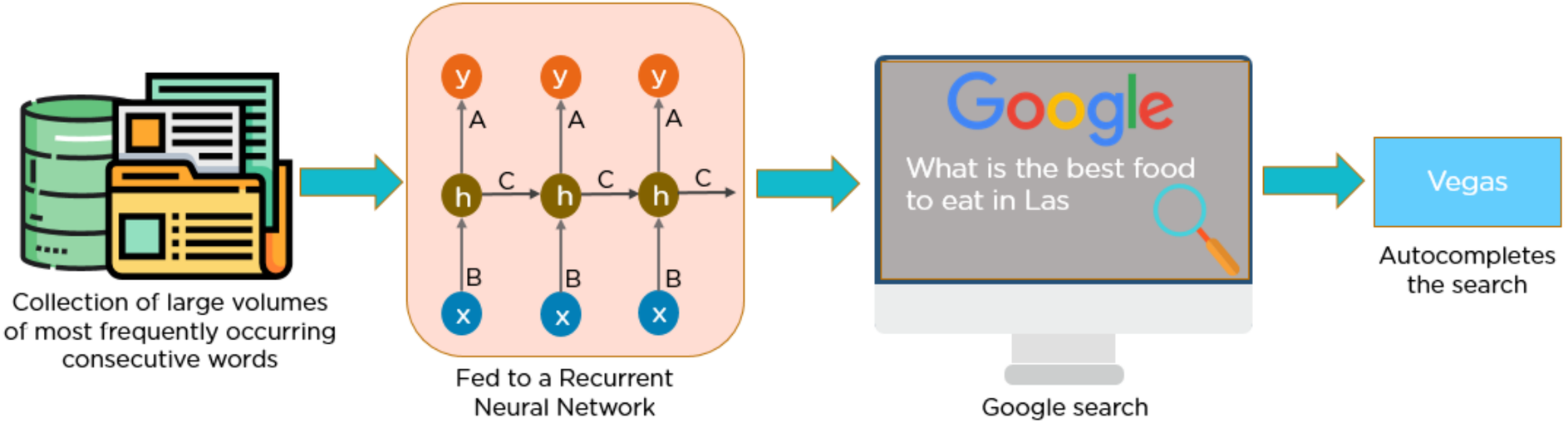
- **RNN advantages**
  - Can process any length input
  - Computation in step *t* can (in theory) use information from many steps back
  - Model size doesn't increase for longer inputs
  - Same weights applied on each time step, so there is symmetry on how inputs are processed

- **RNN problems**
  - RNN computations are slow
  - In practice, it is difficult to access information from many steps back



Training Recurrent Neural Networks

33 RNN Language Models



https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn?source=sl_frs_nav_playlist_video_clicked

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

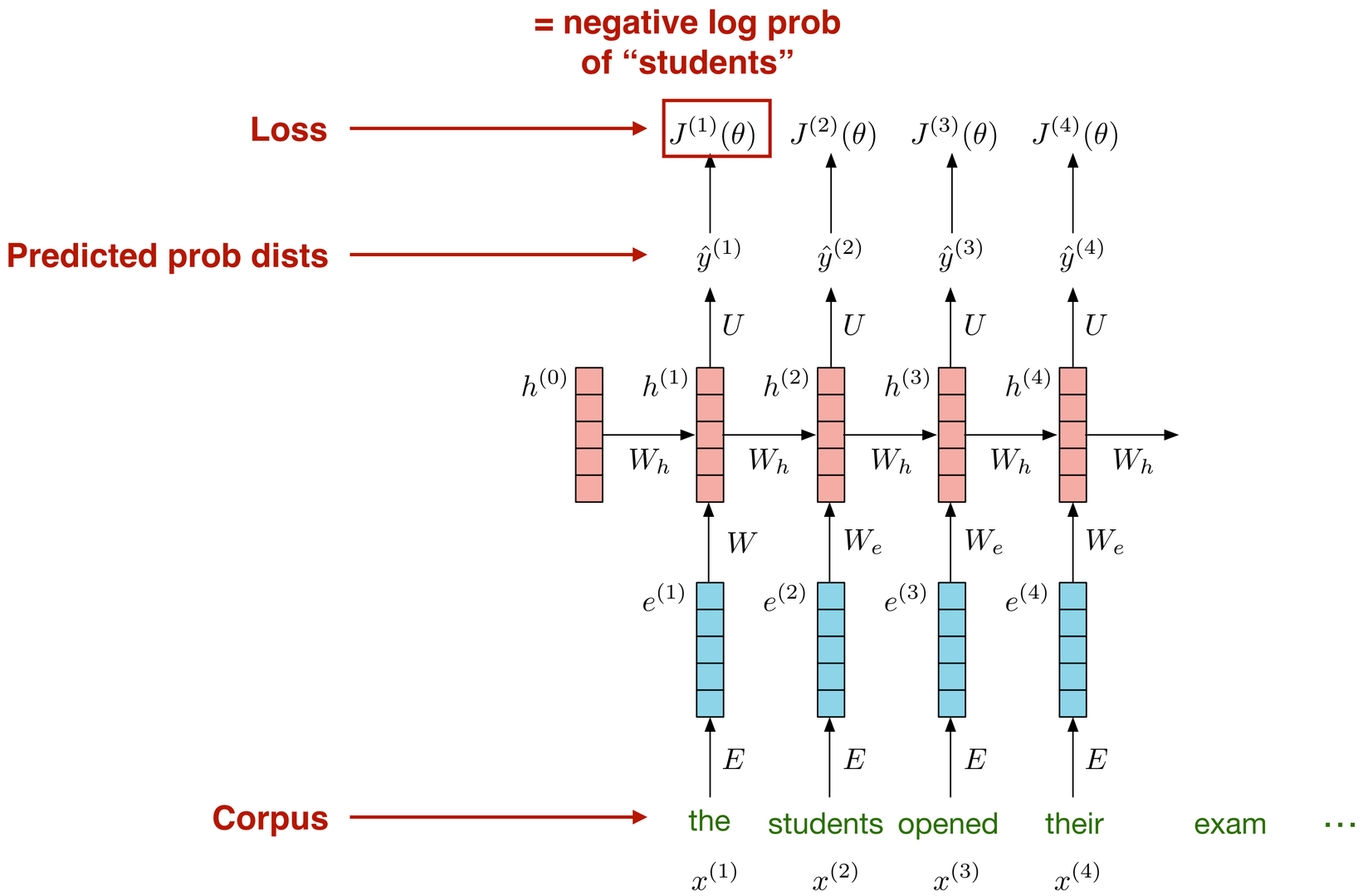
Loss function on step *t* is cross-entropy between predicted probability distribution $\hat{\boldsymbol{y}}^{(t)}$, and the true next word $\boldsymbol{y}^{(t)}$ (one-hot for $\boldsymbol{x}^{(t+1)}$):

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

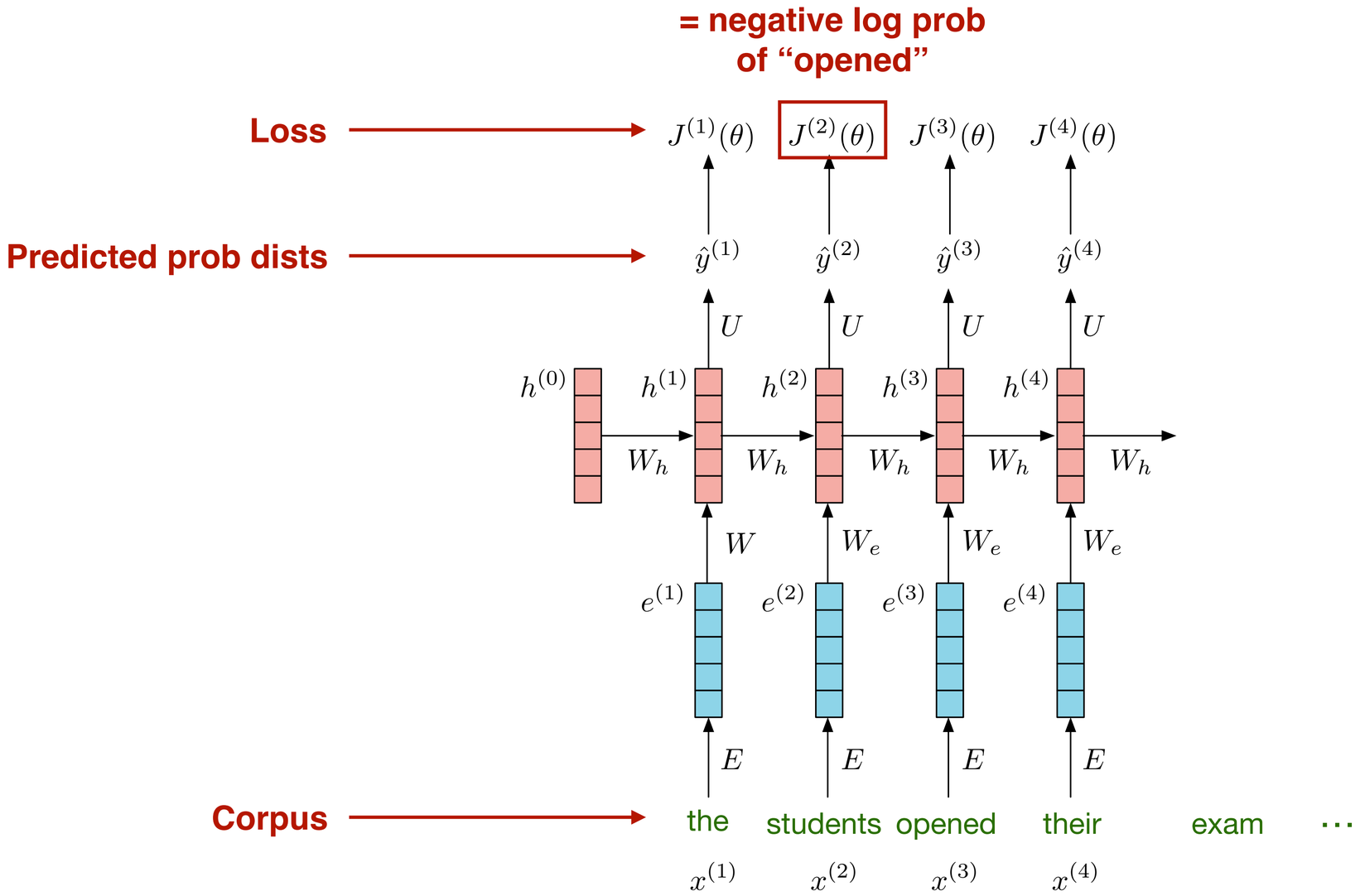
Average this to get overall loss for entire training set:

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

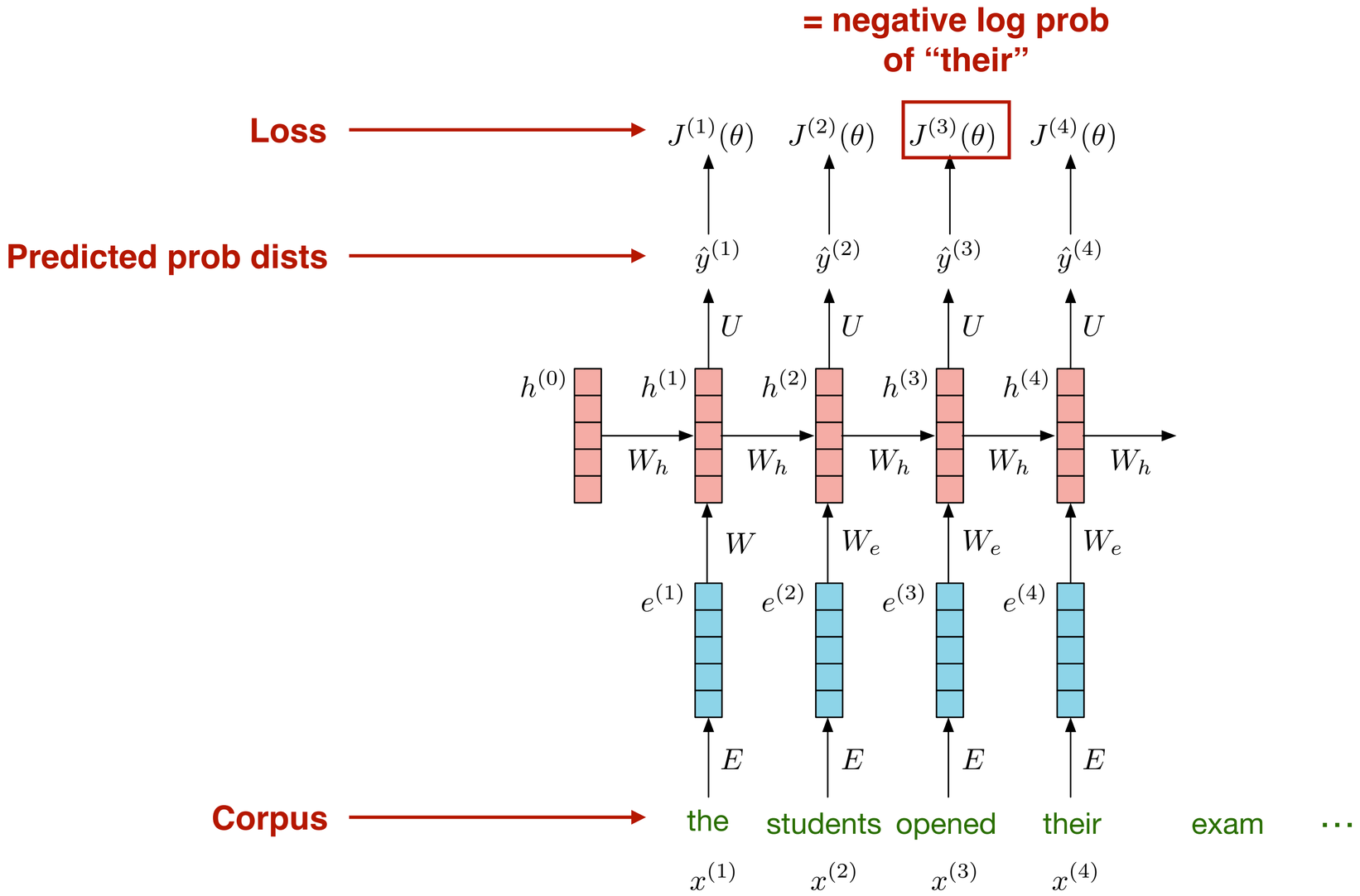
Training an RNN Language Model



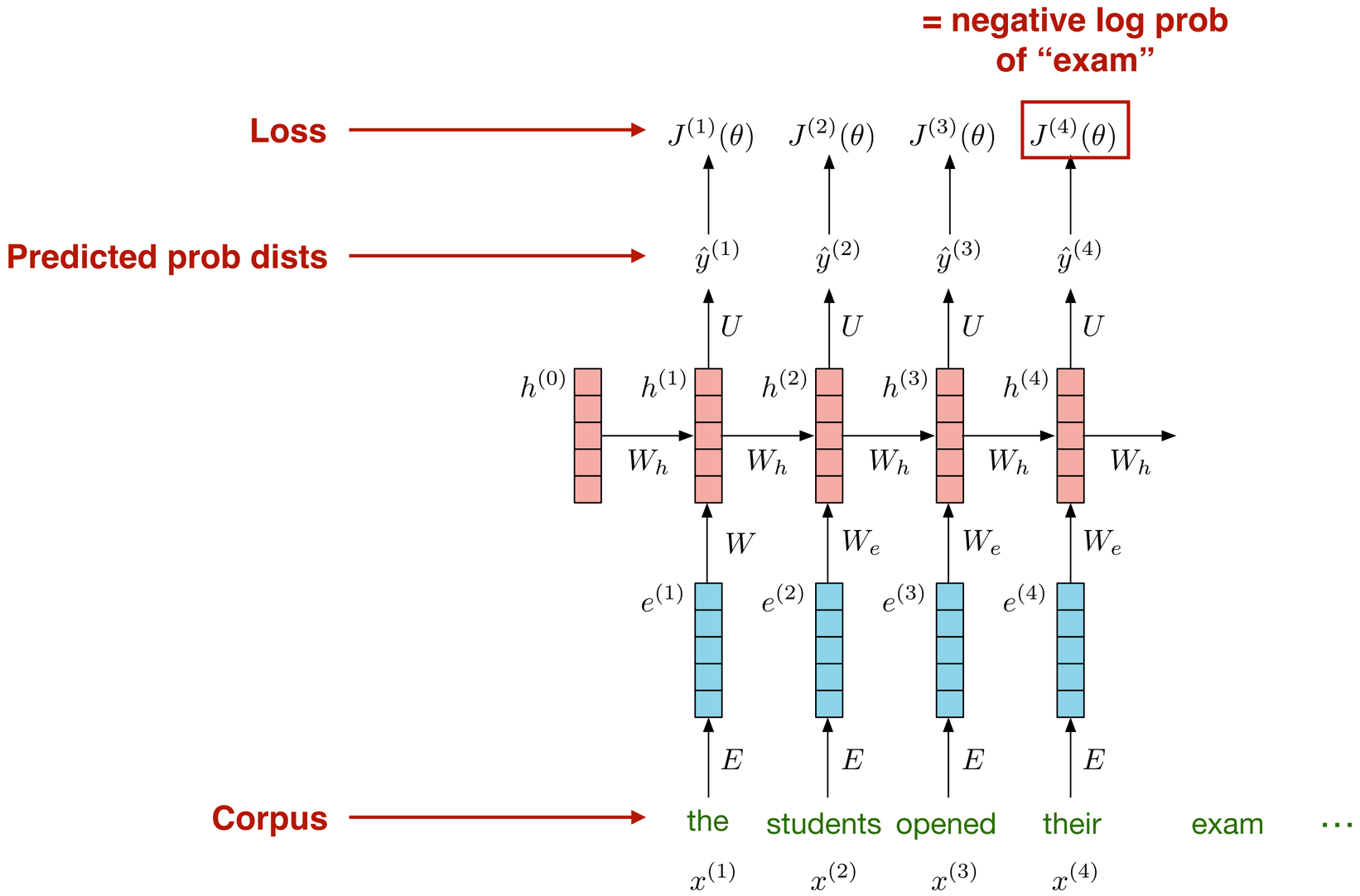
Training an RNN Language Model



Training an RNN Language Model



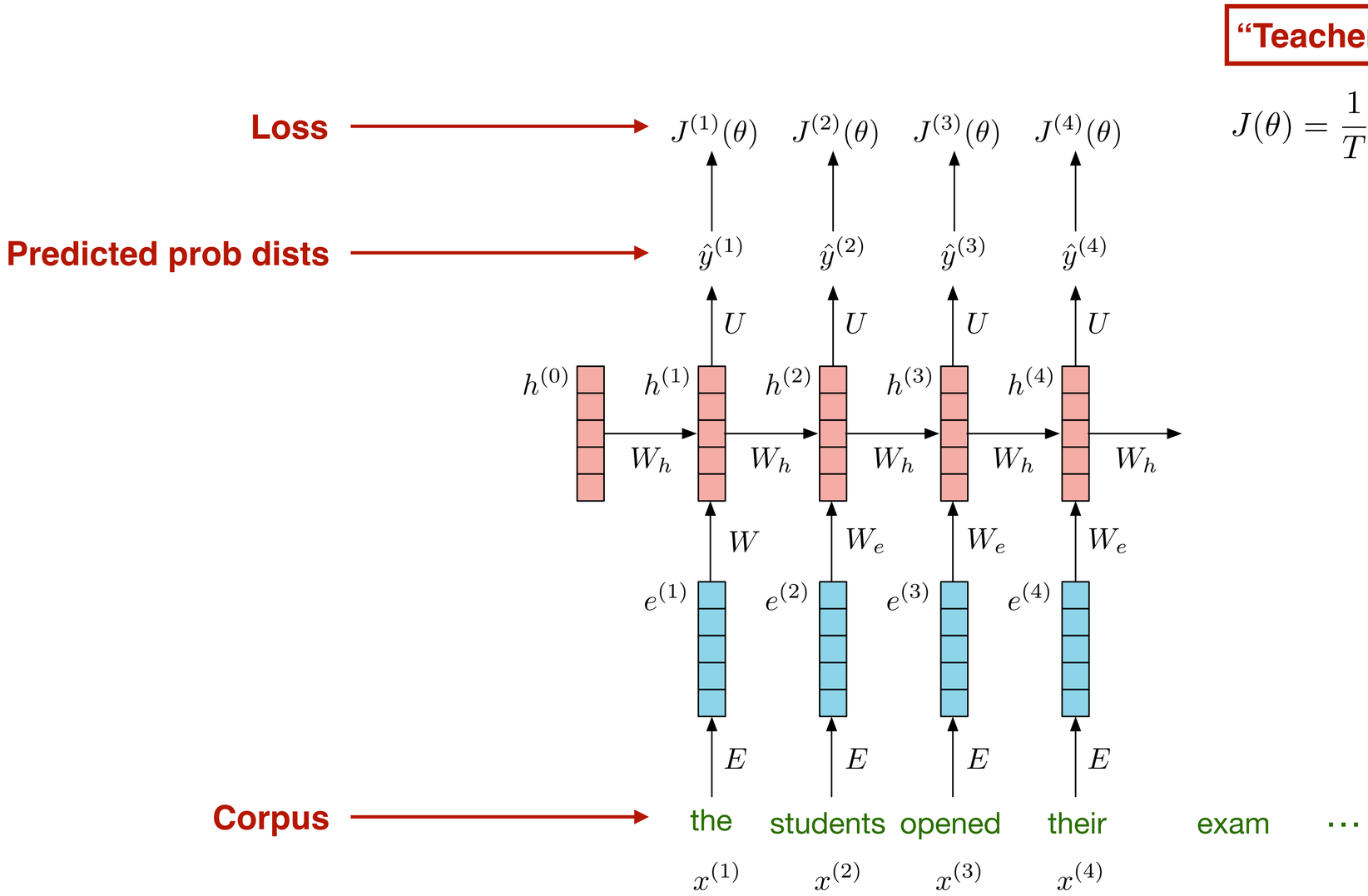
Training an RNN Language Model



Training an RNN Language Model

"Teacher forcing"

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



Training an RNN Language Model

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

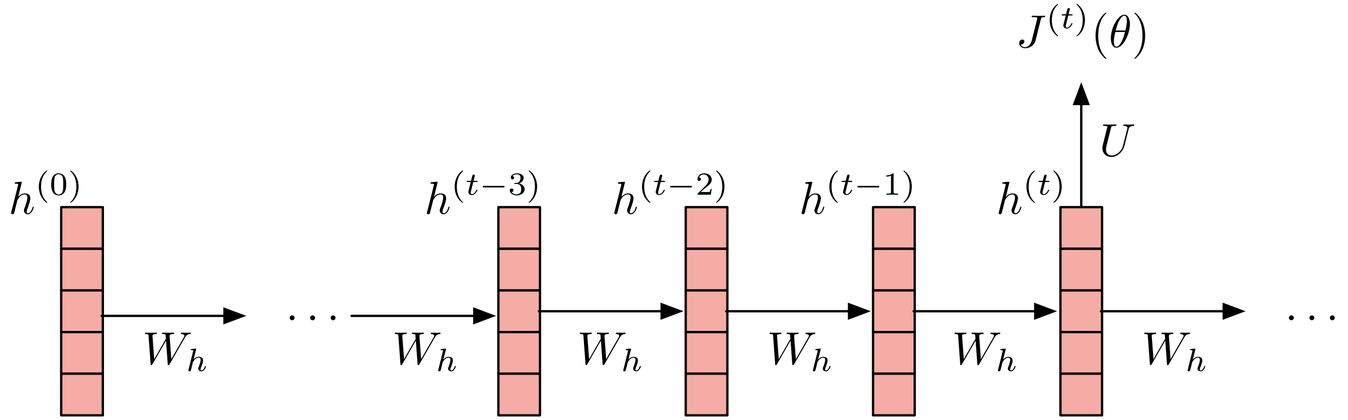
$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J^{(t)}(\theta)$$

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.

Backpropagation for RNNs



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

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

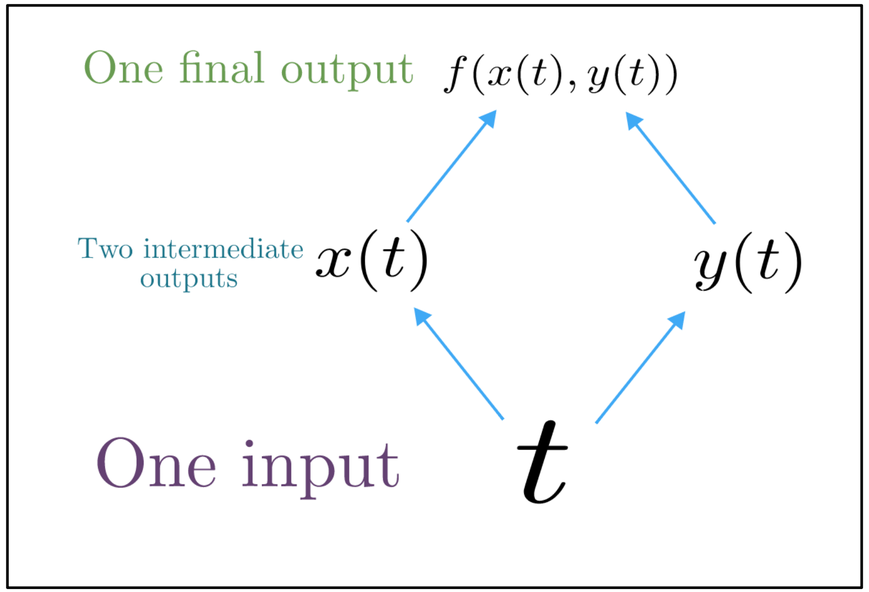
"The gradient w.r.t. a repeated weight is the sum of the gradient w.r.t. each time it appears"

Why?

42 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:

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



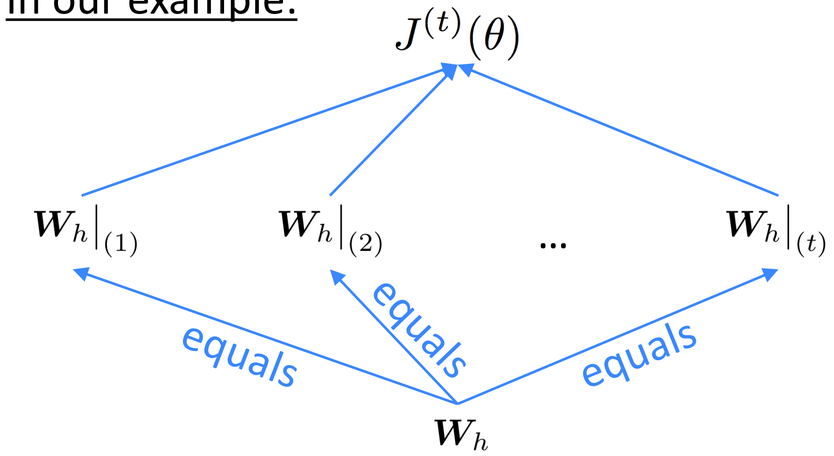
Source:
https://www.khanacademy.org/math/multivariable-calculus/multivariable-derivatives/differentiating-vector-valued-functions/a/multivariable-chain-rule-simple-version

Backpropagation for RNNs

- 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:



Apply the multivariable chain rule:

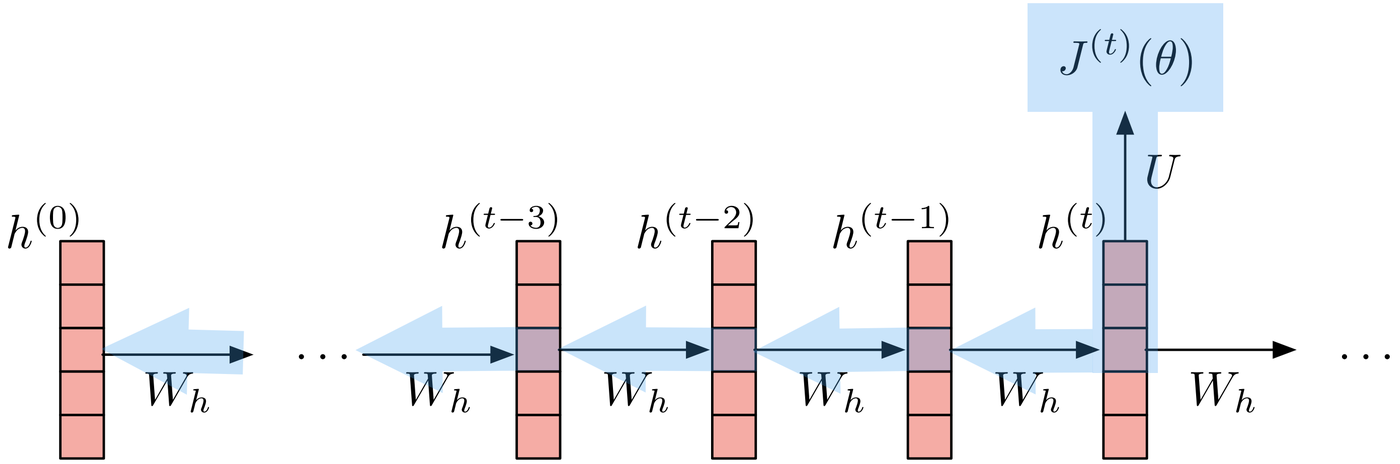
$$\frac{\partial J^{(t)}}{\partial \boldsymbol{W}_h} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \boldsymbol{W}_h}\bigg|_{(i)} \frac{\partial \boldsymbol{W}_h|_{(i)}}{\partial \boldsymbol{W}_h}$$

$= 1$

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

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

Backpropagation for RNNs



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

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]

RNN Applications

Generating Text with a RNN Language Model

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

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.

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

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!

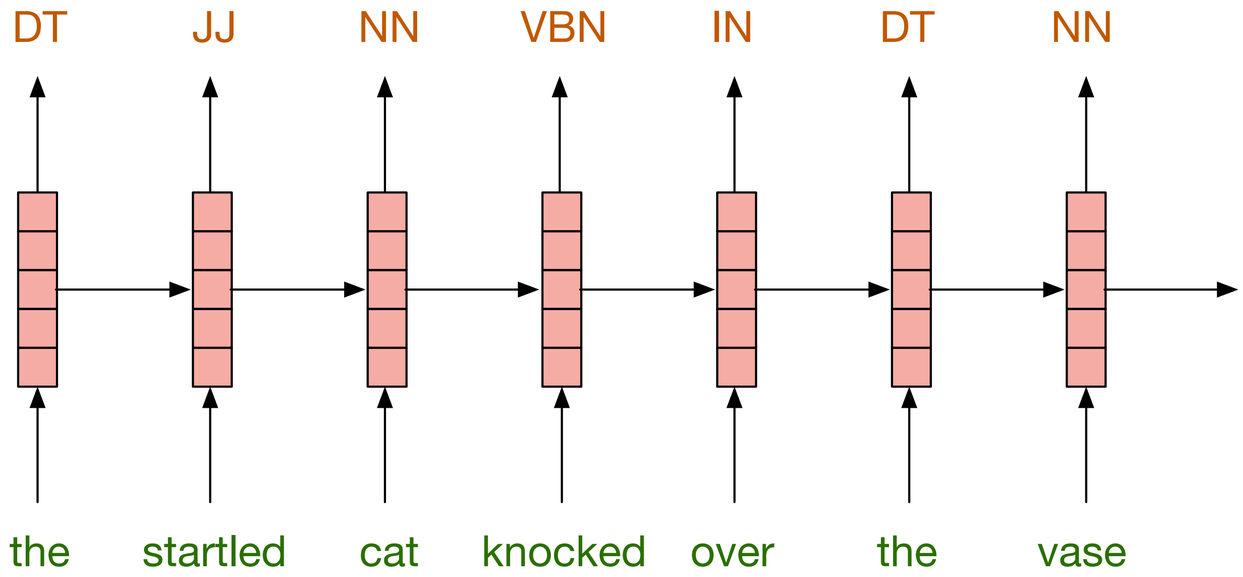
Why Should We Care about Language Modeling

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

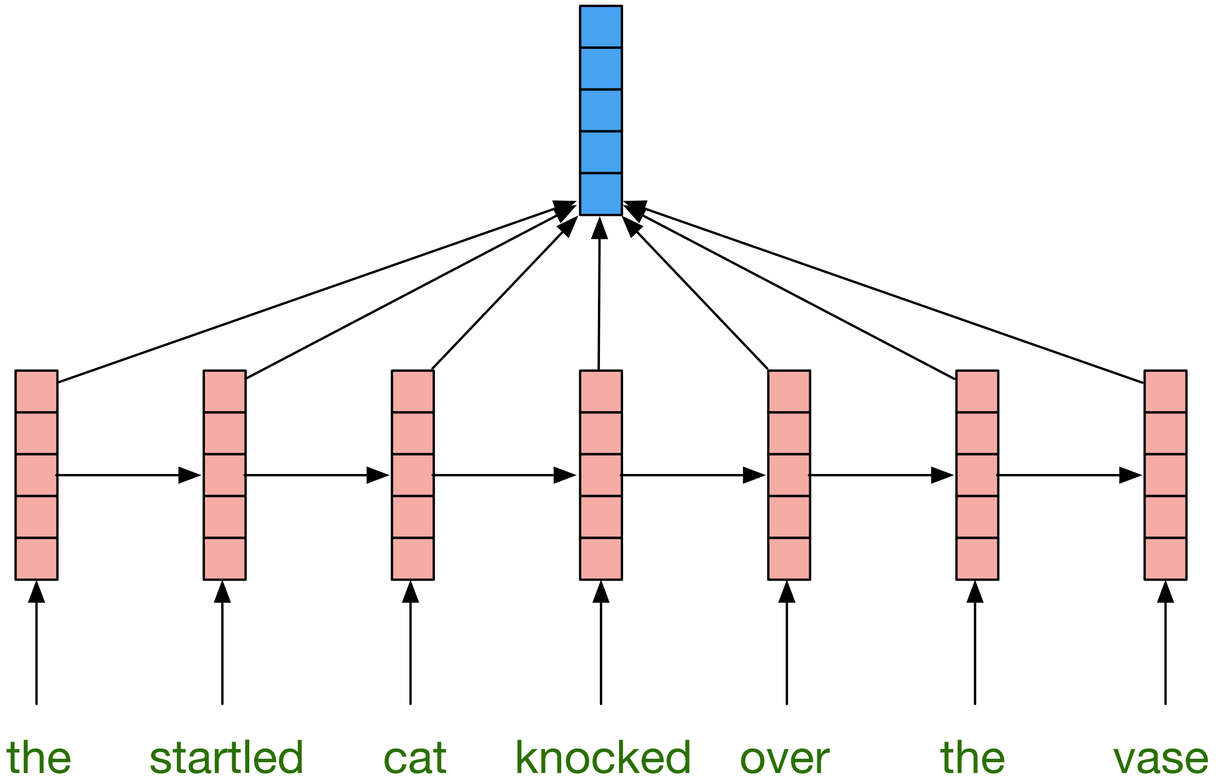
Language Modeling is a subcomponent of many NLP tasks, especially those involving generating text or estimating the probability of text:

- Predictive typing
- Speech recognition
- Handwriting recognition
- Spelling/grammar correction
- Authorship identification
- Machine translation
- Summarization
- Dialogue
- etc.

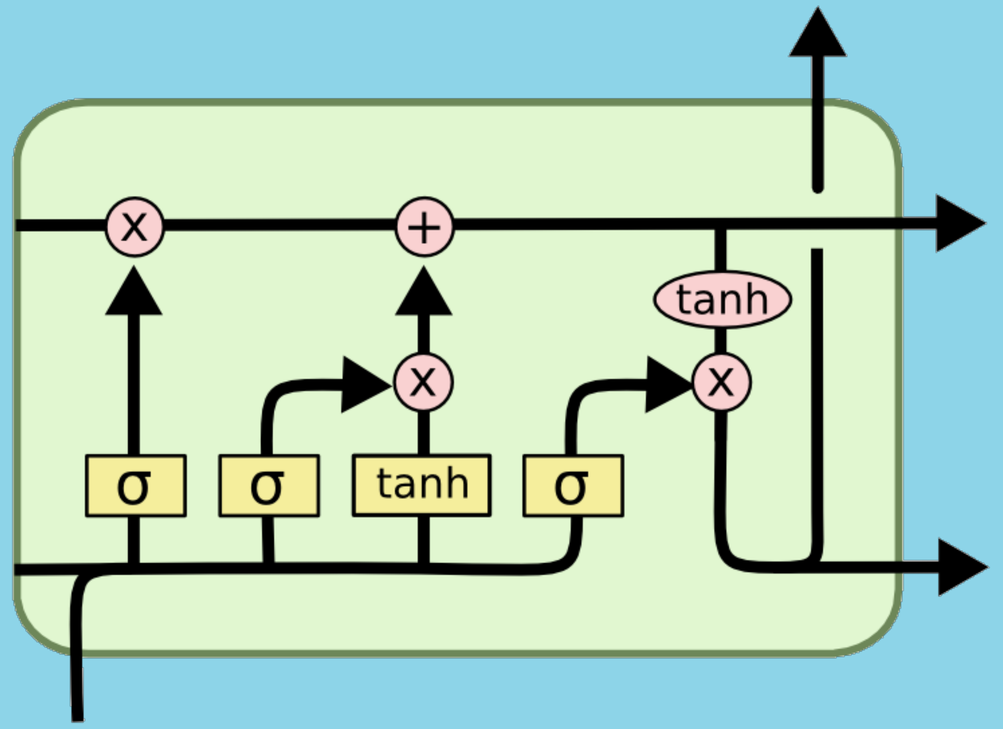
RNNs Can Be Used for Tagging



RNNs Can Be Used as an Encoder Module



RNN Variants



Get RNNs to Work

- Vanishing gradient problem

motivates

- Two new types of RNN: LSTM and GRU

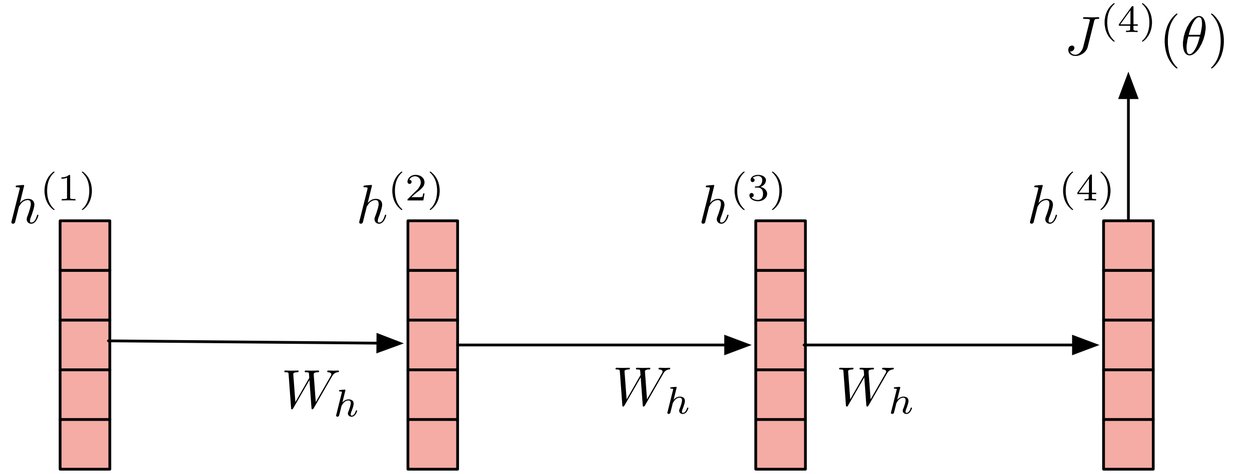
- Other fixes for vanishing (or exploding) gradient:
  - Gradient clipping
  - Skip connections

- More fancy RNN variants:
  - Bidirectional RNNs
  - Multi-layer RNNs

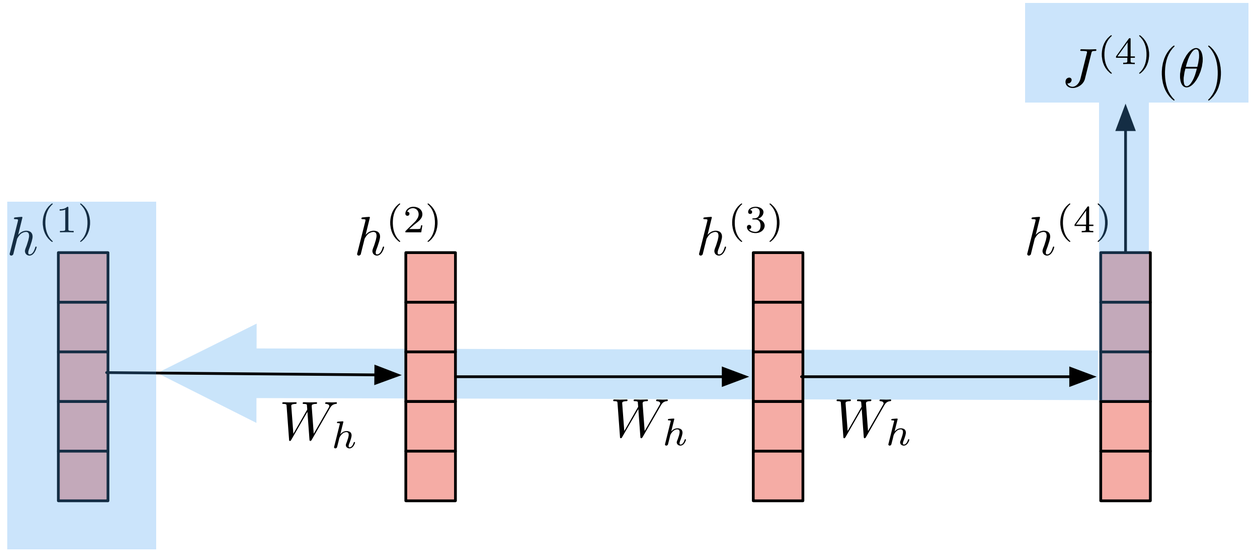
Lots of important definitions today!

Problems with RNNs: Vanishing Gradient and Exploding Gradient Problem

Vanishing Gradient Intuition

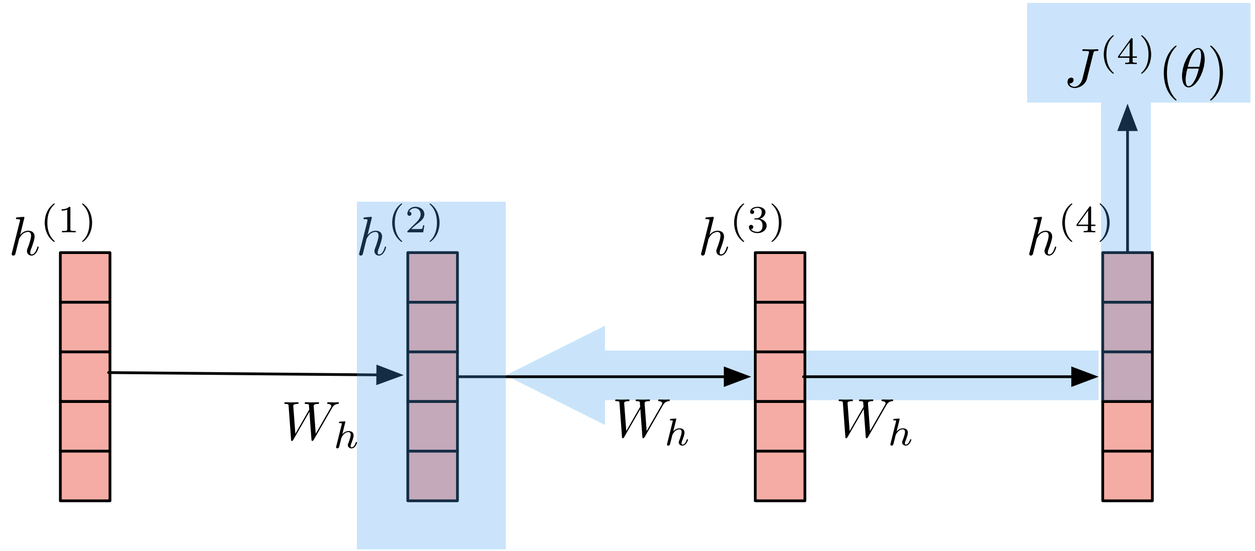


Vanishing Gradient Intuition



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

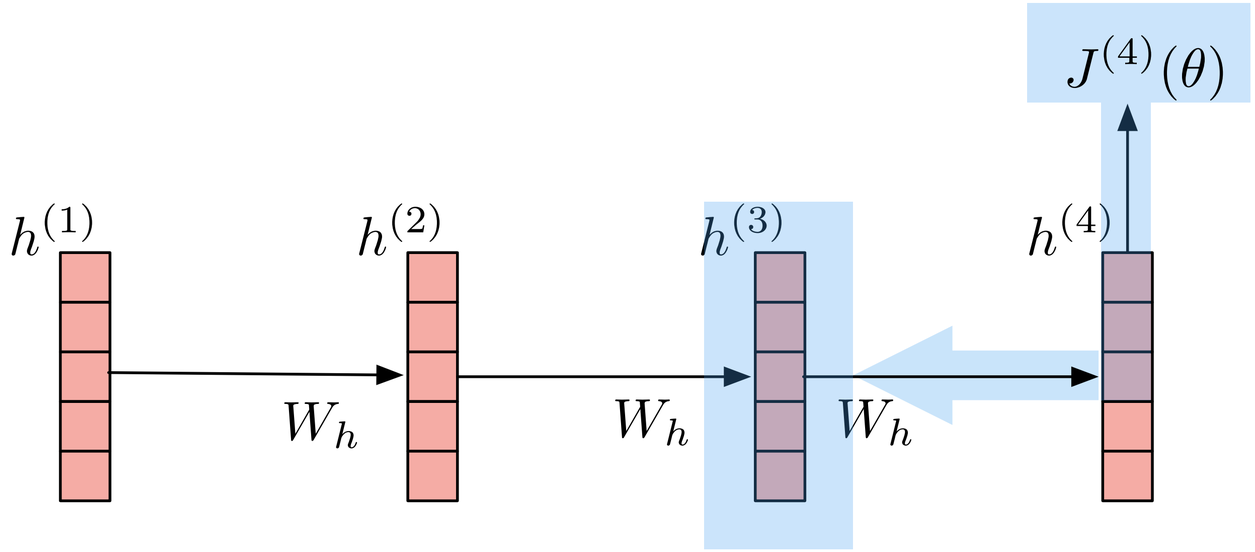
Vanishing Gradient Intuition



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

Chain Rule!

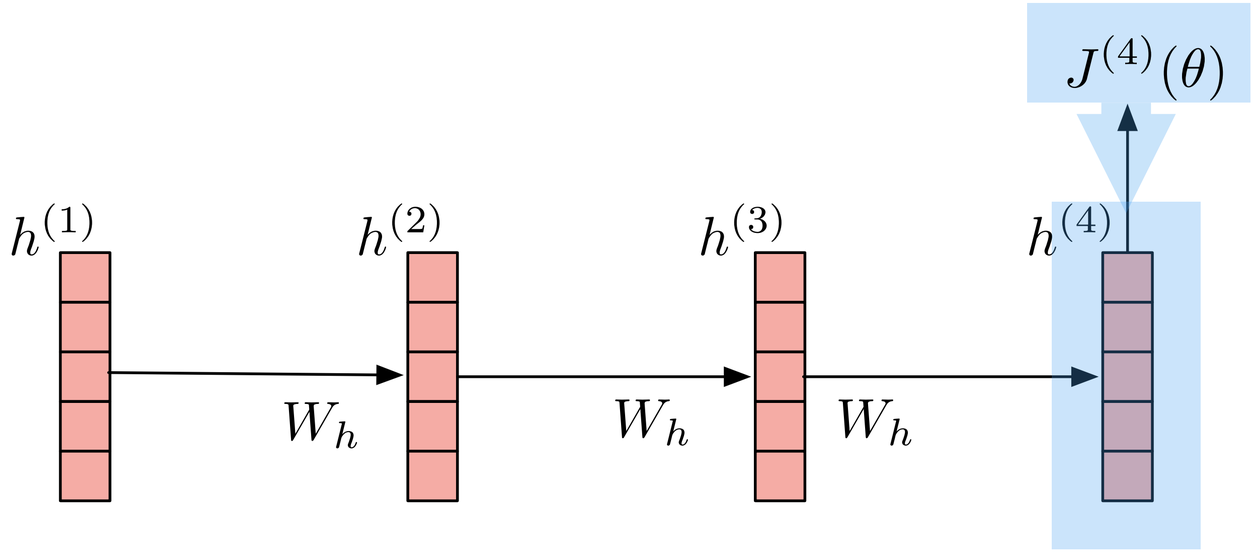
Vanishing Gradient Intuition



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

Chain Rule!

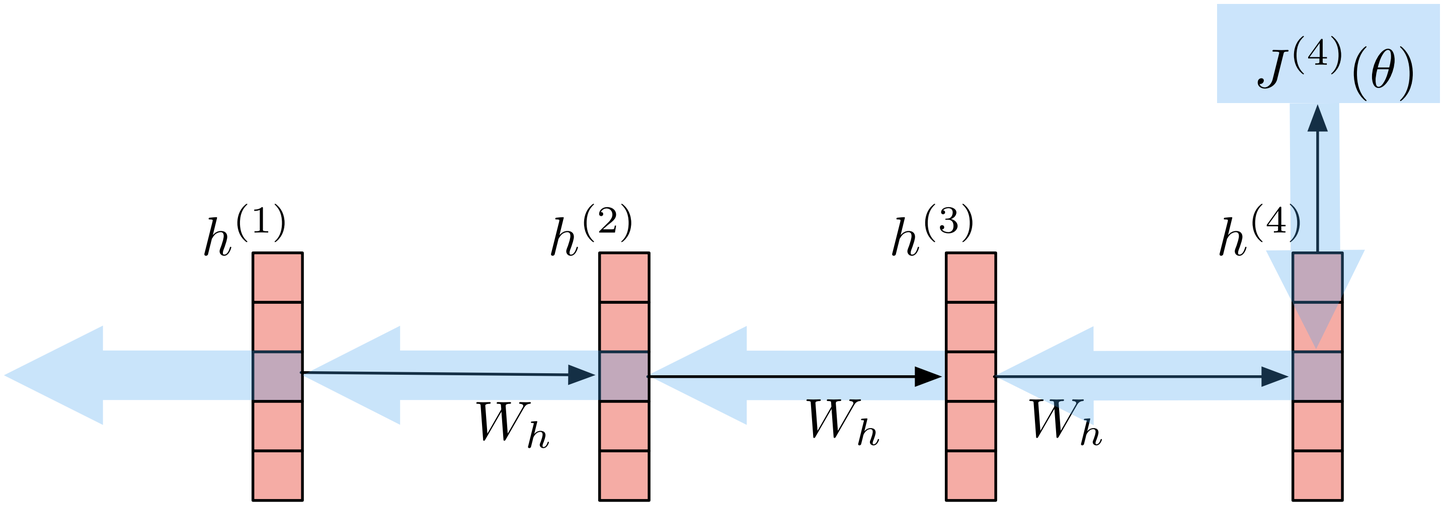
Vanishing Gradient Intuition



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

Chain Rule!

Vanishing Gradient Intuition



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

What happens if these are small?

Vanishing gradient problem:
the gradient signal gets smaller and smaller as it backpropagates further

Vanishing Gradient Proof Sketch (Linear Case)

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

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

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

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

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* $\ell = i - j$

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

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

If W_h is "small", then this term gets exponentially problematic as ℓ becomes large

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

Vanishing Gradient Proof Sketch (Linear Case)

What's wrong with $\boldsymbol{W}_h^\ell$?

Consider if the eigenvalues of $\boldsymbol{W}_h$ are all less than 1: (sufficient but not necessary)

$$\lambda_1, \lambda_2, \ldots, \lambda_n < 1$$
$$\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_n \text{ (eigenvectors)}$$

We can write $\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \boldsymbol{W}_h^\ell$ using the eigenvectors of $\boldsymbol{W}_h$ as a basis:

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

Approaches 0 as ℓ grows so gradient vanishes

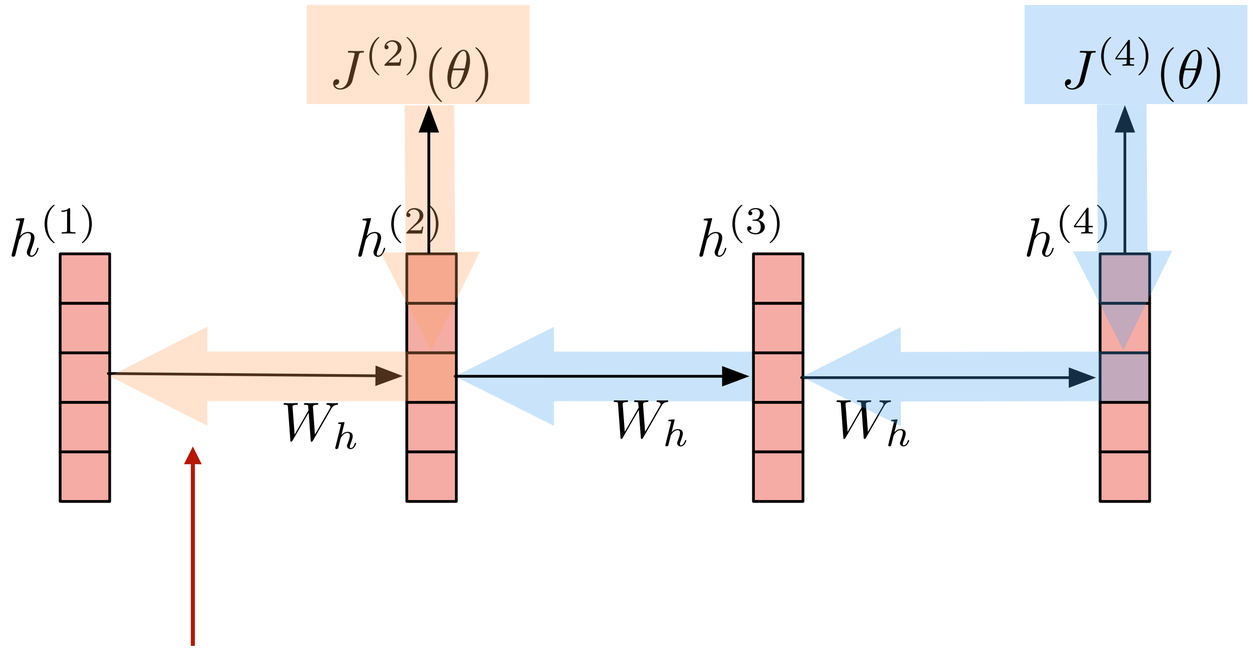
What about nonlinear activations σ (i.e., what we use?)

- Pretty much the same thing, except the proof requires $\lambda_i < \gamma$

for some γ dependent on dimensionality and σ

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 faraway is lost because it's much smaller than gradient signals from close-by.

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

Why is Vanishing Gradient a Problem?

- Another explanation: Gradient can be viewed as a measure of *the effect of the past on the future*

- If the gradient becomes vanishingly small over longer distances (step *t* to step *t+n*), then we can't tell whether:
  1. There's no dependency between step *t* and *t+n* in the data
  2. We have wrong parameters to capture the true dependency between *t* and *t+n*

Effect of Vanishing Gradient on RNN-LM

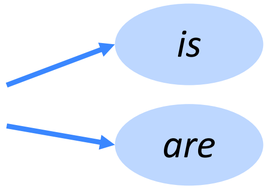
- **LM task:** *When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her _____*

- 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 gradient is small, the model can't learn this dependency
  - So the model is unable to predict similar long-distance dependencies at test time

Effect of Vanishing Gradient on RNN-LM

- **LM task:** *The writer of the books ___*



- **Correct answer**: *The writer of the books <u>is</u> planning a sequel*
- **Syntactic recency:** *The <u>writer</u> of the books <u>is</u>* (correct)
- **Sequential recency:** *The writer of the <u>books</u> <u>are</u>* (incorrect)
- Vanishing gradient problems may bias RNN-LMs towards learning from sequential recency, so they make this type of error more often than we'd like. [Linzen et al 2016]

"Assessing the Ability of LSTMs to Learn Syntax-Sensitive Dependencies", Linzen et al, 2016. https://arxiv.org/pdf/1611.01368.pdf

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 bad parameter configuration (with large loss)

- 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

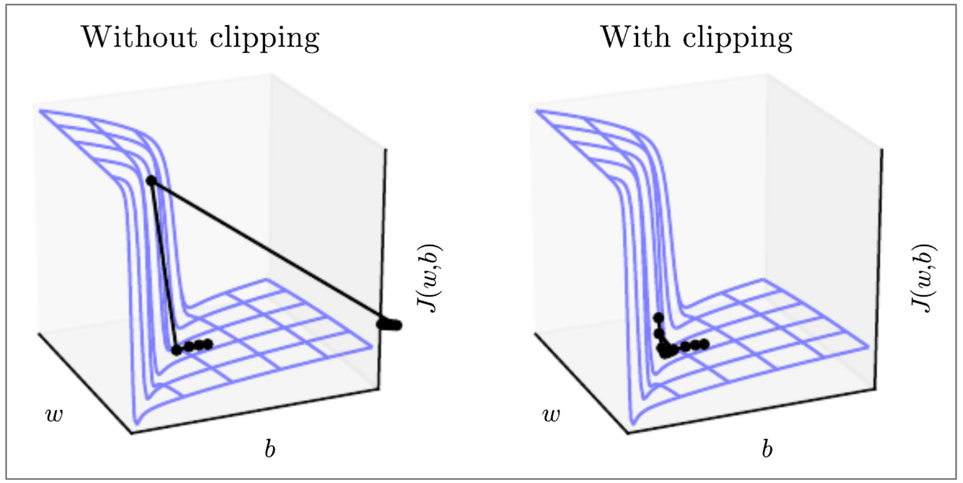
Algorithm 1 Pseudo-code for norm clipping

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

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

Gradient Clipping - Solution for Exploding Gradient



- This shows the loss surface of a simple RNN (hidden state is a scalar not a vector)
- The "cliff" is dangerous because it has steep gradient
- On the left, gradient descent takes two very big steps due to steep gradient, resulting in climbing the cliff then shooting off to the right (both bad updates)
- On the right, gradient clipping reduces the size of those steps, so effect is less drastic

Source: "Deep Learning", Goodfellow, Bengio and Courville, 2016. Chapter 10.11.1. https://www.deeplearningbook.org/contents/rnn.html

How to Fix 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?

LSTM and GRU

Long Short-Term Memory (LSTM)

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

- 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 erase, write and read information from the cell

- 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

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

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

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

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

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

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

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

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

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

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

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

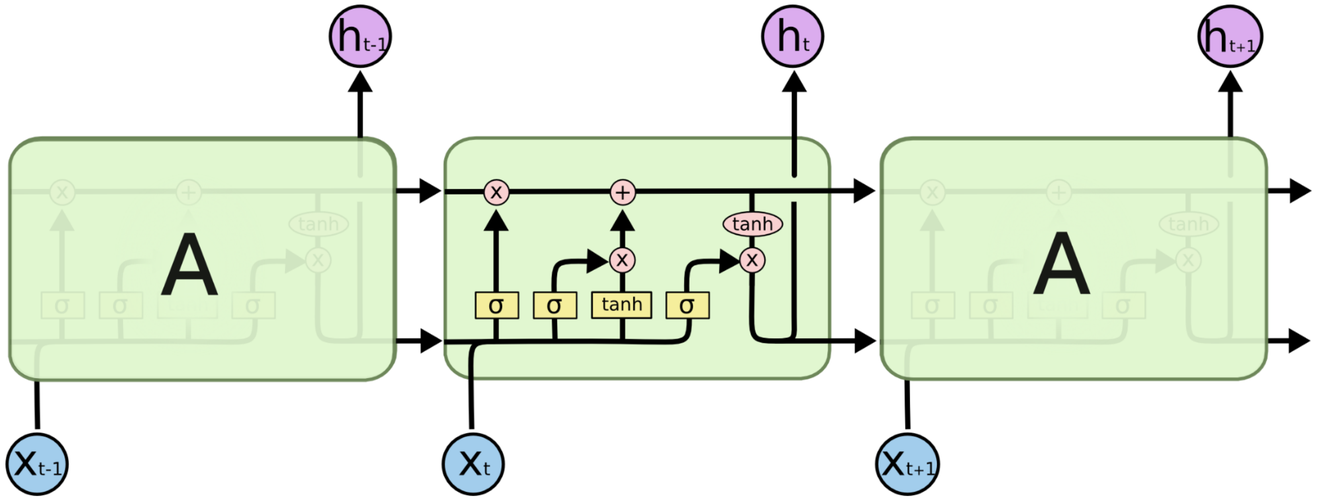
$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$$

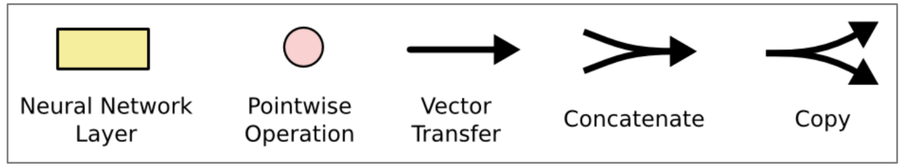
Gates are applied using element-wise product

All these are vectors of same length n

Long Short-Term Memory (LSTM)

You can think of the LSTM equations visually like this:

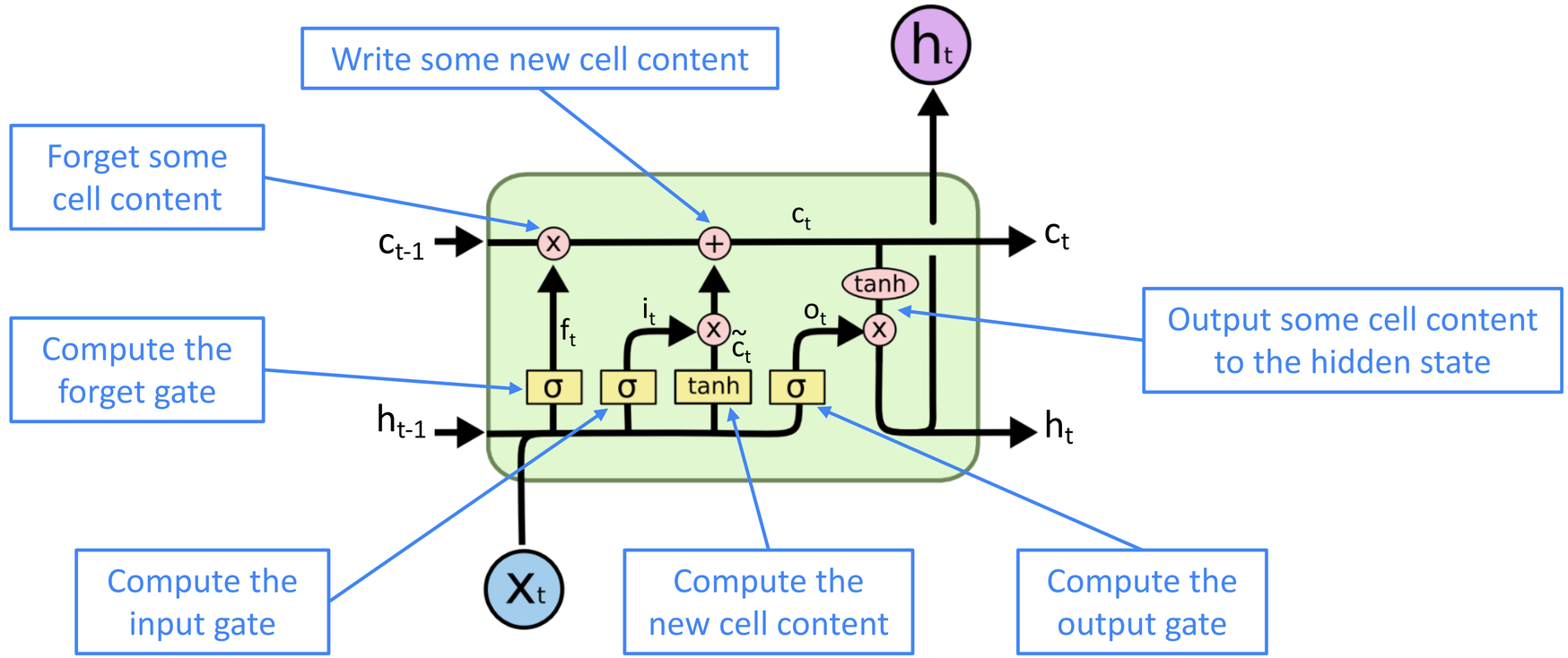




Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short-Term Memory (LSTM)

You can think of the LSTM equations visually like this:



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

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

- 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, image captioning
  - LSTM became the dominant approach

- Now (2020), other approaches (e.g. Transformers) have become more dominant for certain tasks.
  - For example in **WMT** (a MT conference + competition):
  - In WMT 2016, the summary report contains ”RNN” 44 times
  - In WMT 2018, the report contains “RNN” 9 times and “Transformer” 63 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

Gated Recurrent Unites (GRU)

- Proposed by Cho et al. in 2014 as a simpler alternative to the LSTM.
- On each timestep *t* we have input $\boldsymbol{x}^{(t)}$ and hidden state $\boldsymbol{h}^{(t)}$ (no cell state).

Update gate: controls what parts of hidden state are updated vs preserved

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

Reset gate: controls what parts of previous hidden state are used to compute new content

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

New hidden state content: reset gate selects useful parts of prev hidden state. Use this and current input to compute new hidden content.

$$\tilde{\boldsymbol{h}}^{(t)} = \tanh\left(\boldsymbol{W}_h(\boldsymbol{r}^{(t)} \circ \boldsymbol{h}^{(t-1)}) + \boldsymbol{U}_h\boldsymbol{x}^{(t)} + \boldsymbol{b}_h\right)$$

Hidden state: update gate simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

$$\boldsymbol{h}^{(t)} = (1 - \boldsymbol{u}^{(t)}) \circ \boldsymbol{h}^{(t-1)} + \boldsymbol{u}^{(t)} \circ \tilde{\boldsymbol{h}}^{(t)}$$

How does this solve vanishing gradient?
Like LSTM, GRU makes it easier to retain info long-term (e.g. by setting update gate to 0)

"Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation", Cho et al. 2014, https://arxiv.org/pdf/1406.1078v3.pdf

LSTM v.s. GRU

- Researchers have proposed many gated RNN variants, but LSTM and GRU are the most widely-used
- Rule of thumb: LSTM is a good default choice (especially if your data has particularly long dependencies, or you have lots of training data); Switch to GRUs for speed and fewer parameters.

Is Vanishing/Exploding Gradient just a RNN problem?

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

Residual Connections

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

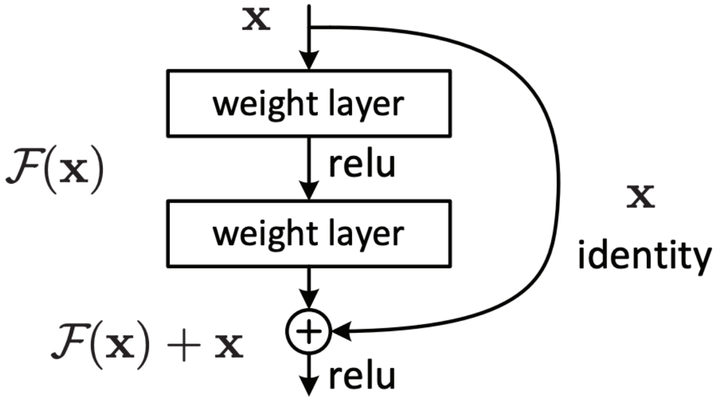


Figure 2. Residual learning: a building block.

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

Dense Connections

For example:

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

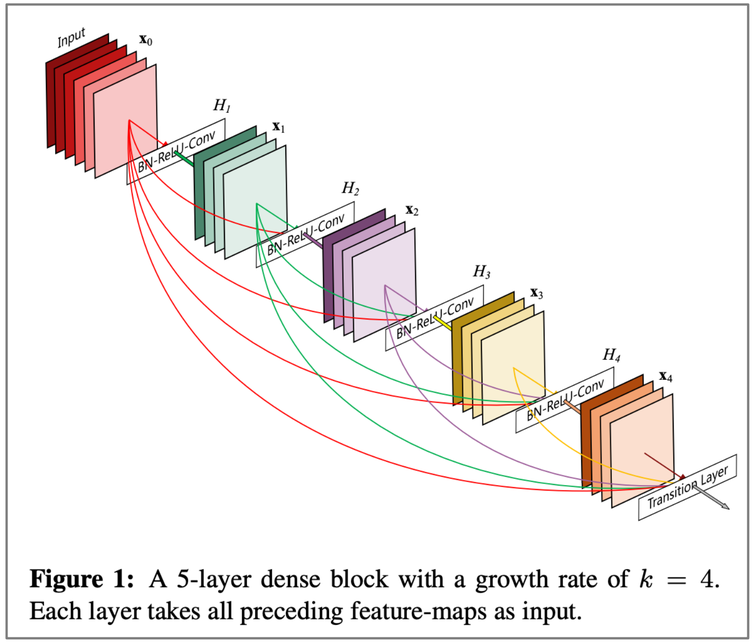


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

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

Highway Connections

For example:

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

Is Vanishing/Exploding Gradient just a RNN problem?

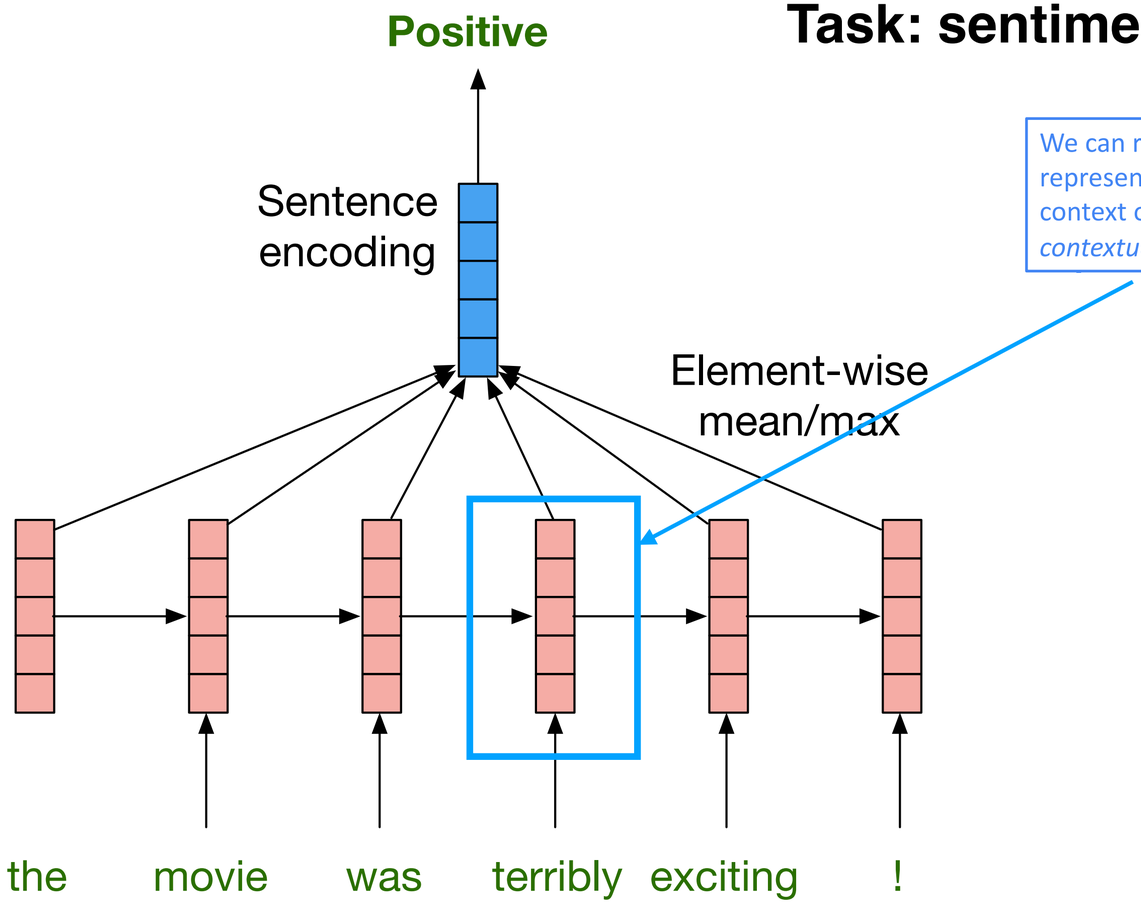
- No! It can be a problem for all neural architectures (including feed-forward and convolutional), especially deep ones.
  - Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small as it backpropagates
  - Thus lower layers are learnt very slowly (hard to train)
  - 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

Bi-RNN

Bidirectional RNNs: Motivation



Task: sentiment classification

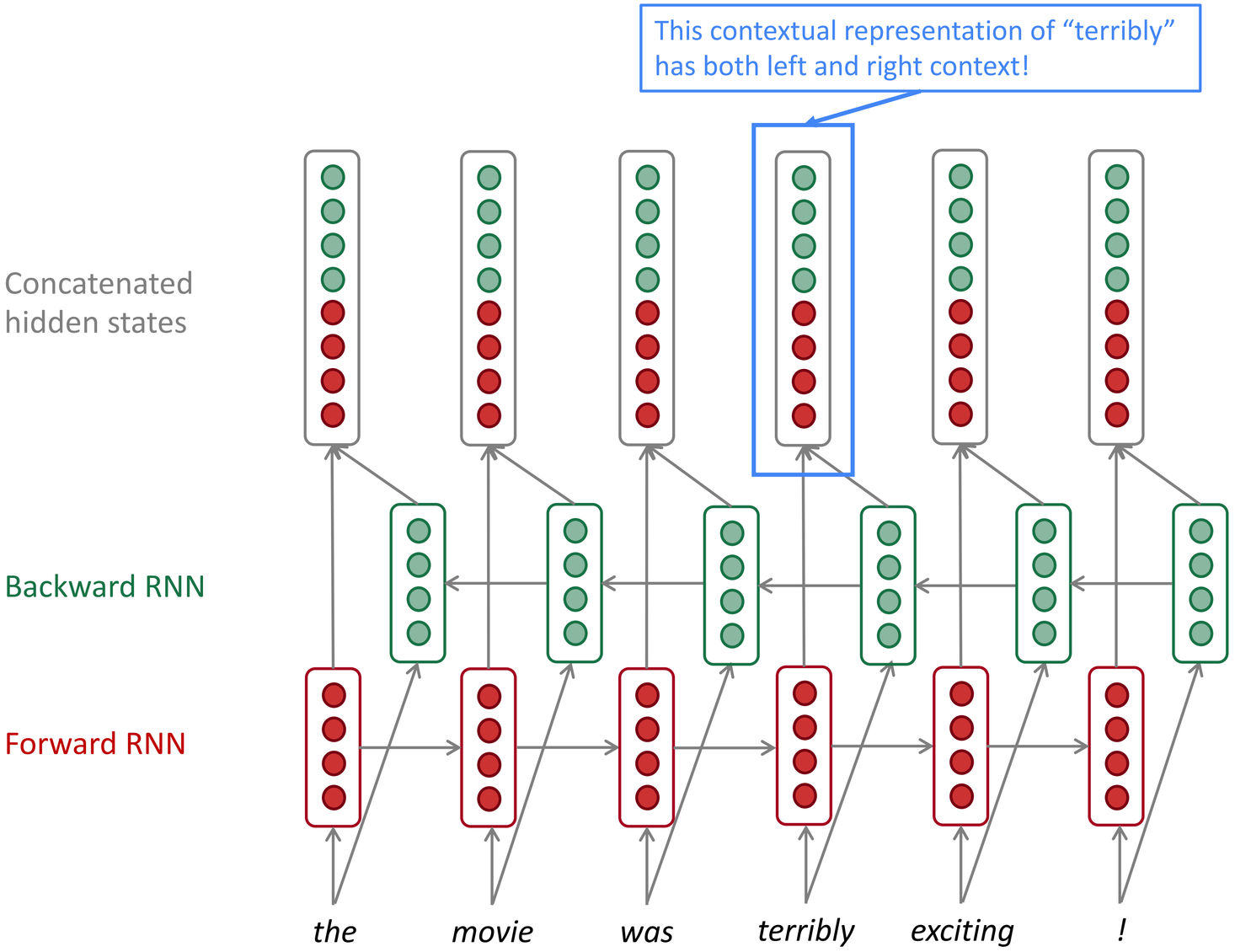
We can regard this hidden state as a representation of the word *"terribly"* in the context of this sentence. We call this a *contextual representation.*

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



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 $\overrightarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{FW}}(\overrightarrow{\boldsymbol{h}}^{(t-1)}, \boldsymbol{x}^{(t)})$

Backward RNN $\overleftarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{BW}}(\overleftarrow{\boldsymbol{h}}^{(t+1)}, \boldsymbol{x}^{(t)})$

Generally, these two RNNs have separate weights

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

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

- 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 BERT later in the course!

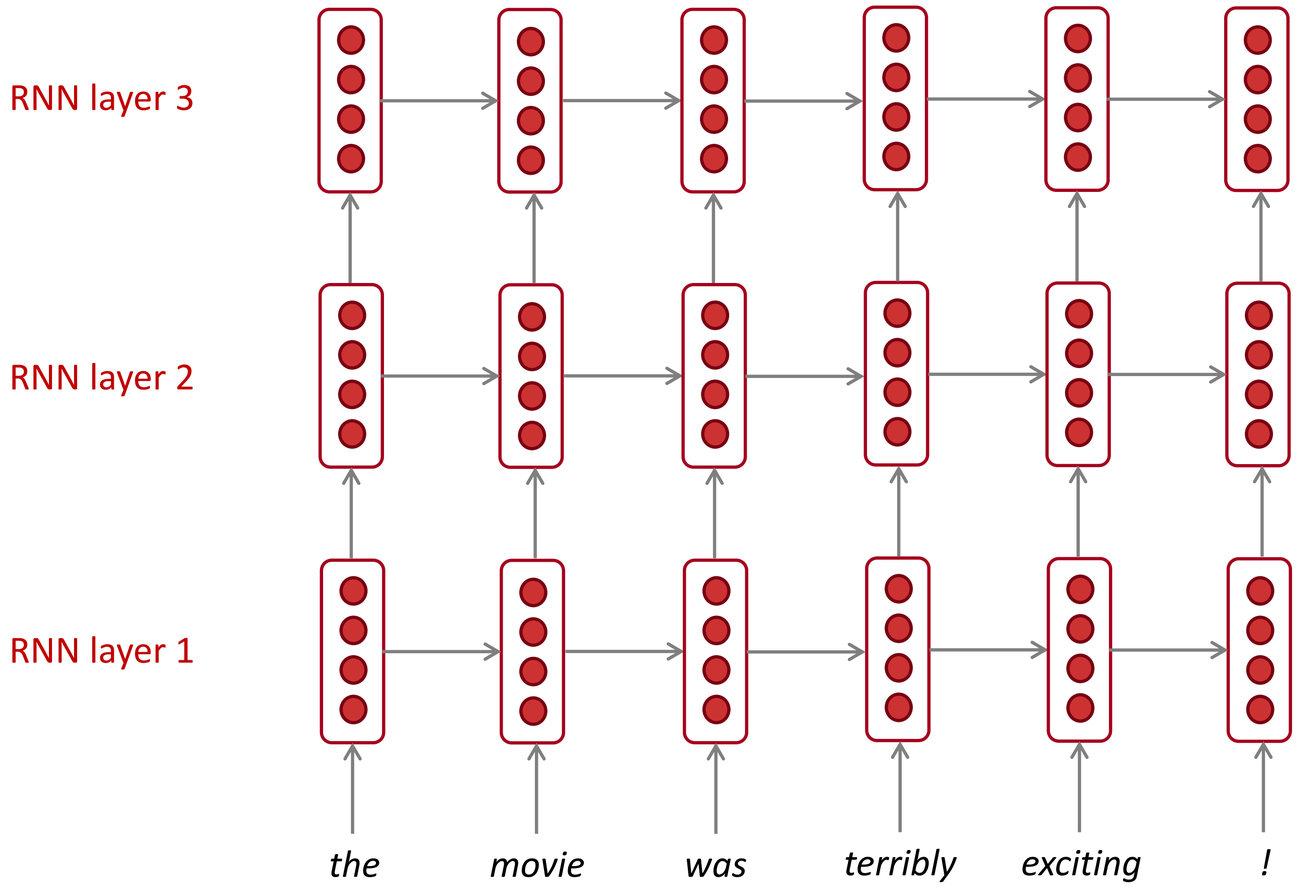
Deep-RNN

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
  - However, skip-connections/dense-connections are needed to train deeper RNNs (e.g. 8 layers)

- Transformer-based networks (e.g. BERT) are frequently 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

Todo

- **Suggested readings**
  - https://web.stanford.edu/~jurafsky/slp3/3.pdf
  - http://karpathy.github.io/2015/05/21/rnn-effectiveness/
  - https://www.deeplearningbook.org/contents/rnn.html
  - http://norvig.com/chomsky.html
  - https://arxiv.org/pdf/1211.5063.pdf
  - http://colah.github.io/posts/2015-08-Understanding-LSTMs/
  - https://arxiv.org/pdf/1503.04069.pdf

Next lecture: Word Meaning and Word Embedding

Thanks! Q&A

Bang Liu
Email: bang.liu@umontreal.ca
Homepage: http://www-labs.iro.umontreal.ca/~liubang/