Natural Language Processing
Lecture 06 Language Models; N-gram LMs; Recurrent Neural LMs

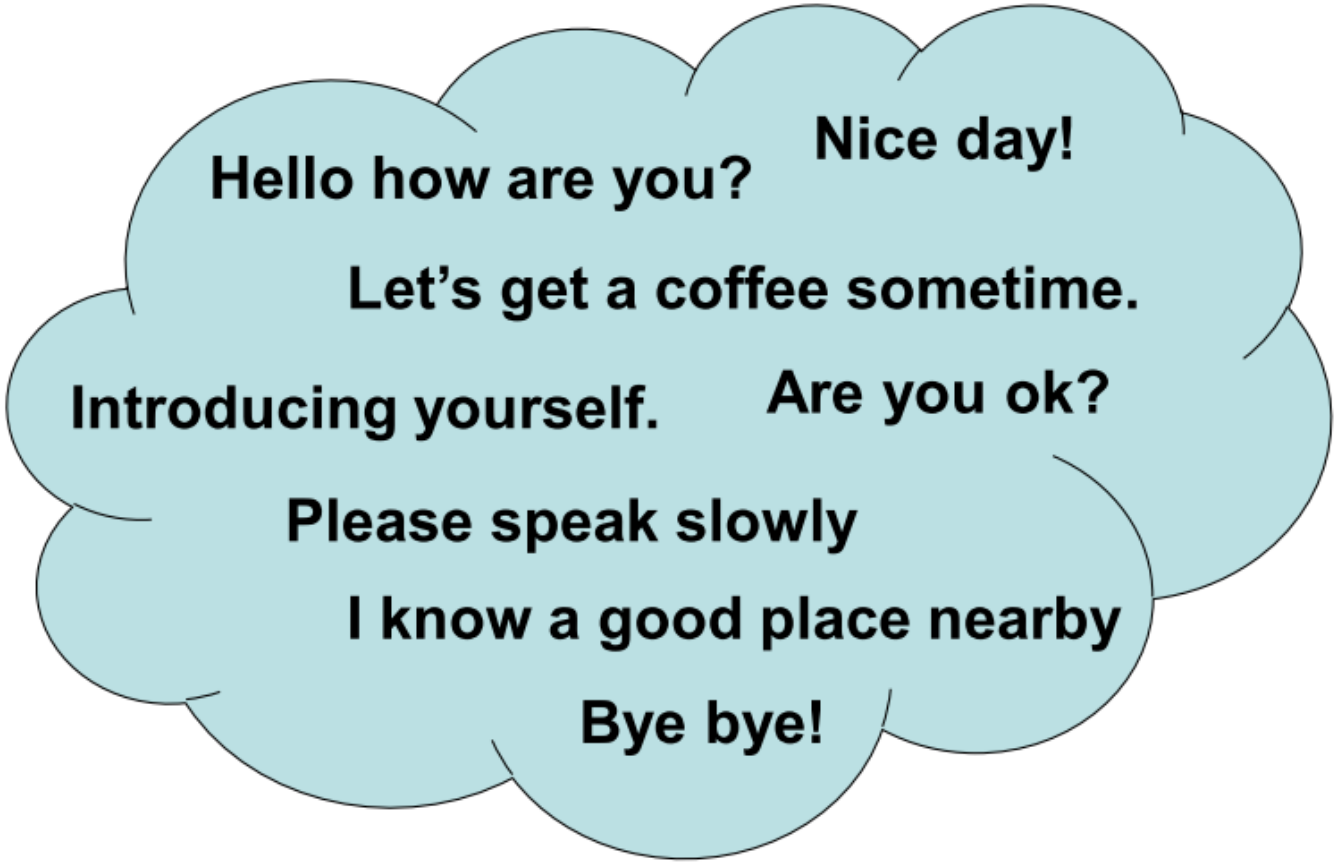
Qun Liu, Valentin Malykh
Huawei Noah's Ark Lab

Spring 2020
A course delivered at MIPT, Moscow

Content

Content

How can we define a language? (Recap)



How can we define a language? (Recap)

— The set theory approach

- A language can be defined as the set of sentences which can be accepted by the speakers of that language.
- It is not possible to define a natural language by enumerate all the sentences, because the number of sentences in a natural languages is infinite.
- Two feasible ways to define a language with infinite sentences:
  - By a Grammar
  - By an Automaton

How can we define a language?

— The probabilistic approach

- A language can also be defined as a probabilistic distribution over all the possible sentences.
- A statistical language model is a probability distribution over sequences of words (sentences) in a given language L:

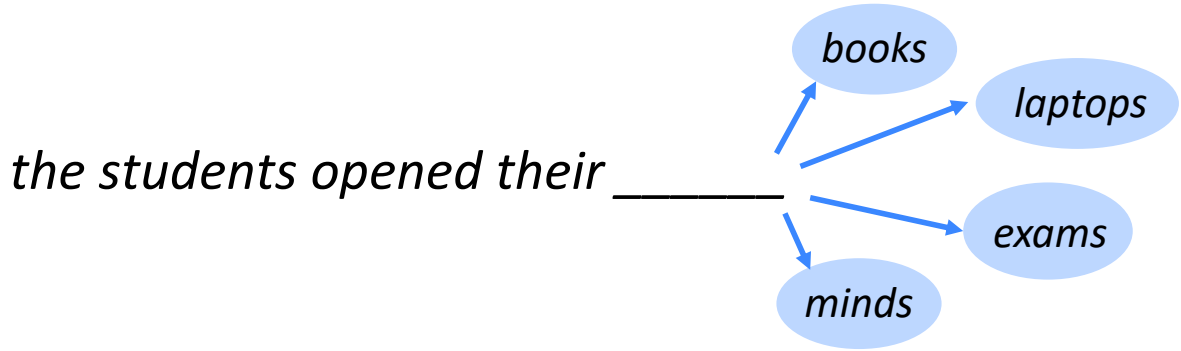
$$\sum_{s \in V^+} P_{LM}(s) = 1$$

- Or:

$$\sum_{\substack{s = w_1 w_2 \ldots w_n \\ w_i \in V, n > 0}} P_{LM}(s) = 1$$

Language Modeling

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



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

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

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

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

15

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Language Modeling

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

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

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

This is what our LM provides

16

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

LMs are extremely powerful tools

- The definition of LMs is simple but LMs are extremely powerful tools in NLP
- A number of NLP problems could be viewed as variations of language modelling problems, and can be solved by LMs.
  - Test generation / Data to text / Image captioning / Text summarization
  - Machine translation
  - Speech Recognition
  - Reading Comprehension
  - POS tagging / Entity recognition / Parsing
- Any task which could be transferred to a sequential problems is suitable for LMs

The bag-of-word problem

From a collection of 100 sentences, we considered the 38 sentences with fewer than 11 words each by simply using an n-gram LM.

Exact reconstruction (24 of 38)

Please give me your response as soon as possible.
⇒ Please give me your response as soon as possible.

Reconstruction preserving meaning (8 of 38)

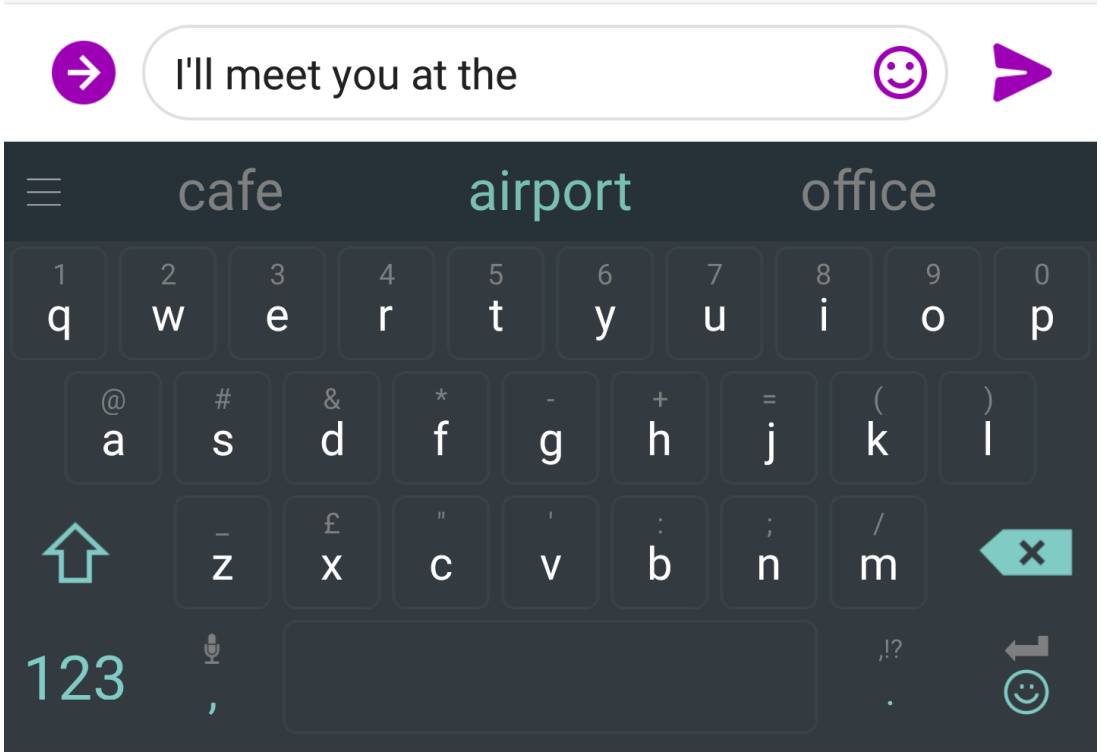
Now let me mention some of the disadvantages.
⇒ Let me mention some of the disadvantages now.

Garbage reconstruction (6 of 38)

In our organization research has two missions.
⇒ In our missions research organization has two.

Brown P F, Cocke J, Pietra S A D, et al., A statistical approach to machine translation. Computational linguistics, 1990, 16(2):79-85.

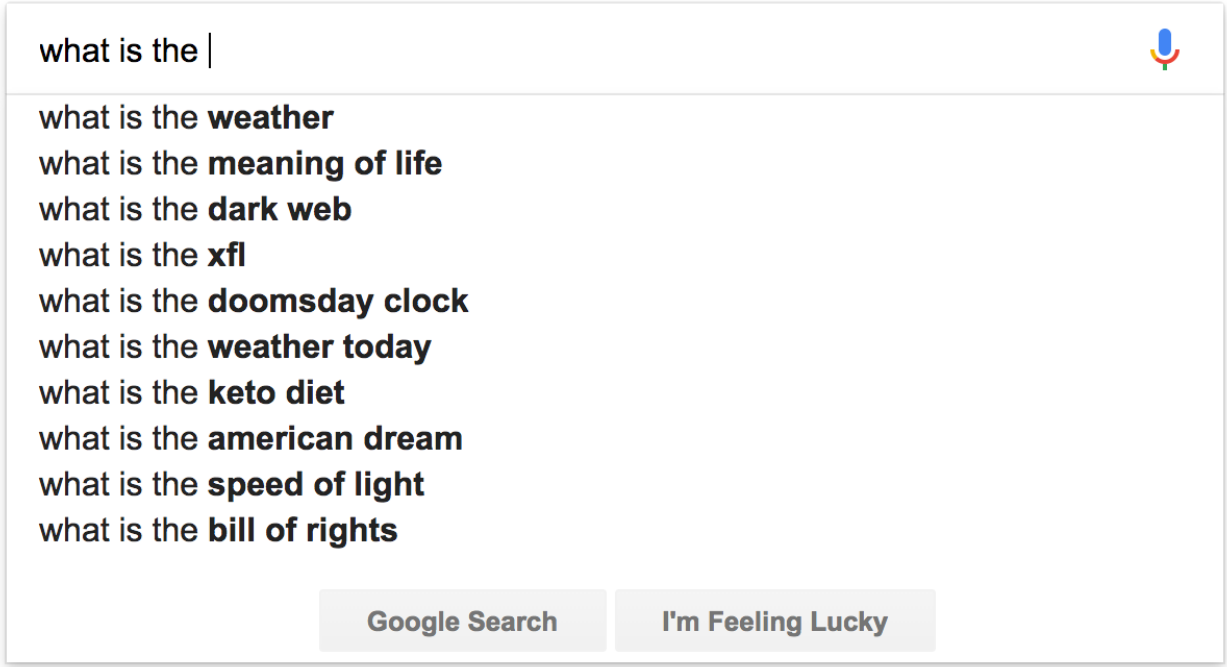
You use Language Models every day!



17

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

You use Language Models every day!



18

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

NLP tasks as Conditional LMs

x "input"	w "**text** output"
An author	A document written by that author
A topic label	An article about that topic
{SPAM, NOT_SPAM}	An email
A sentence in French	Its English translation
A sentence in English	Its French translation
A sentence in English	Its Chinese translation
An image	A text description of the image
A document	Its summary
A document	Its translation
Meterological measurements	A weather report
Acoustic signal	Transcription of speech
Conversational history + database	Dialogue system response
A question + a document	Its answer
A question + an image	Its answer

Chris Dyer, Conditional LMs (slides)

Content

n-gram Language Models

the students opened their _____

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

- Definition: A *n*-gram is a chunk of *n* consecutive words.
  - unigrams: “the”, “students”, “opened”, ”their”
  - bigrams: “the students”, “students opened”, “opened their”
  - trigrams: “the students opened”, “students opened their”
  - 4-grams: “the students opened their”

- Idea: Collect statistics about how frequent different n-grams are, and use these to predict next word.

19

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

n-gram Language Models

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

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

prob of a n-gram

prob of a (n-1)-gram

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

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

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

20

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Bigram LM Parameters

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

- Normalize by unigrams:

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

- Result:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Dan Jurafsky, Speech and Language Processing (slides)

n-gram Language Models: Example

Suppose we are learning a 4-gram Language Model.

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

discard

condition on this

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

For example, suppose that in the corpus:

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

Should we have discarded the "proctor" context?

21

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Sparsity Problems 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.

22

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Storage Problems with n-gram Language Models

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

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

Increasing *n* or increasing corpus increases model size!

23

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

n-gram Language Models in practice

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

Business and financial news

today the _____

get probability distribution

company	0.153
bank	0.153
price	0.077
italian	0.039
emirate	0.039
…	

Sparsity problem: not much granularity in the probability distribution

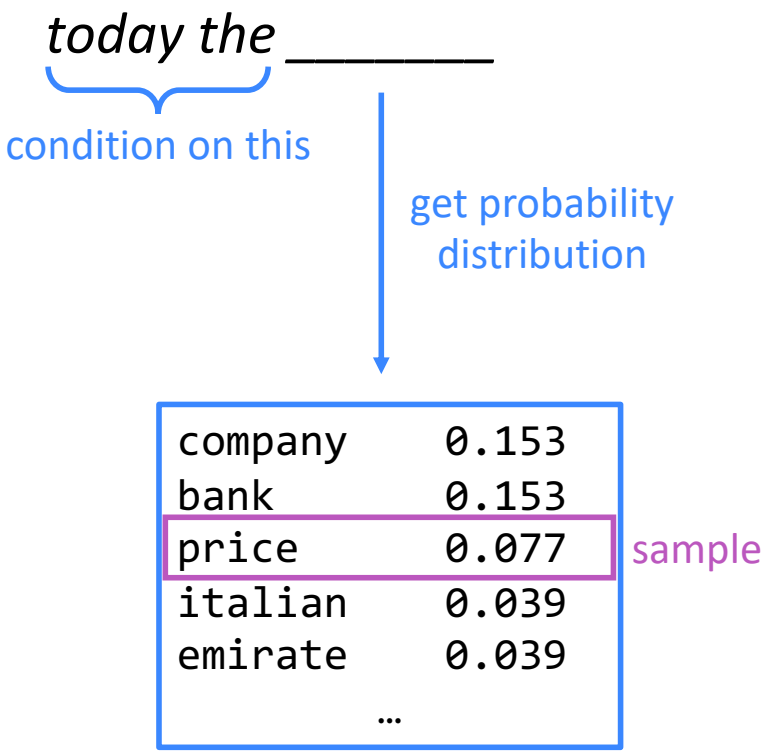
Otherwise, seems reasonable!

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

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Generating text with a n-gram Language Model

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

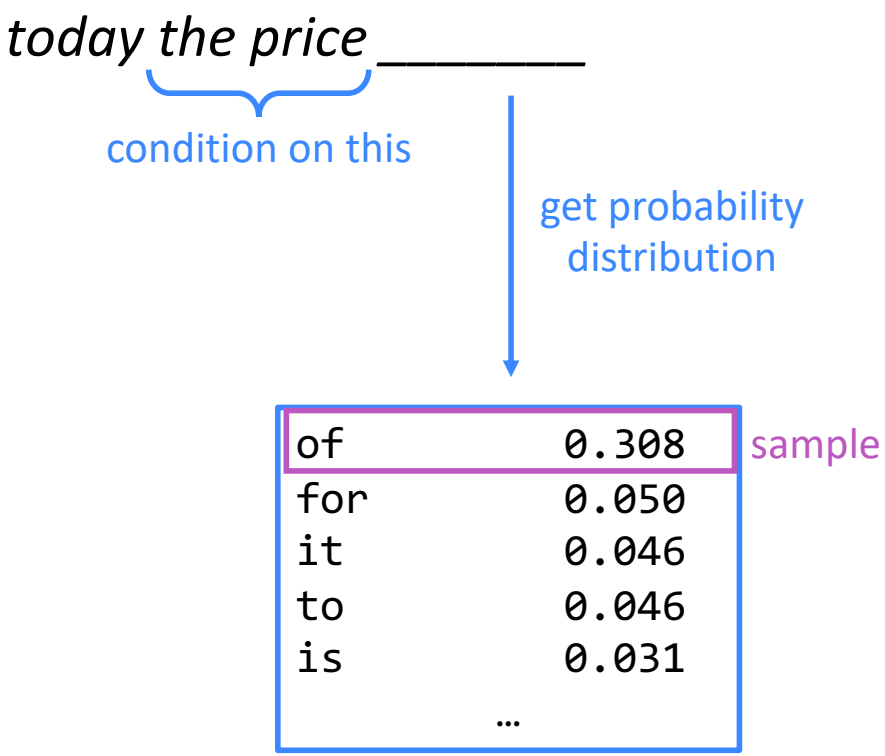


25

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Generating text with a n-gram Language Model

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

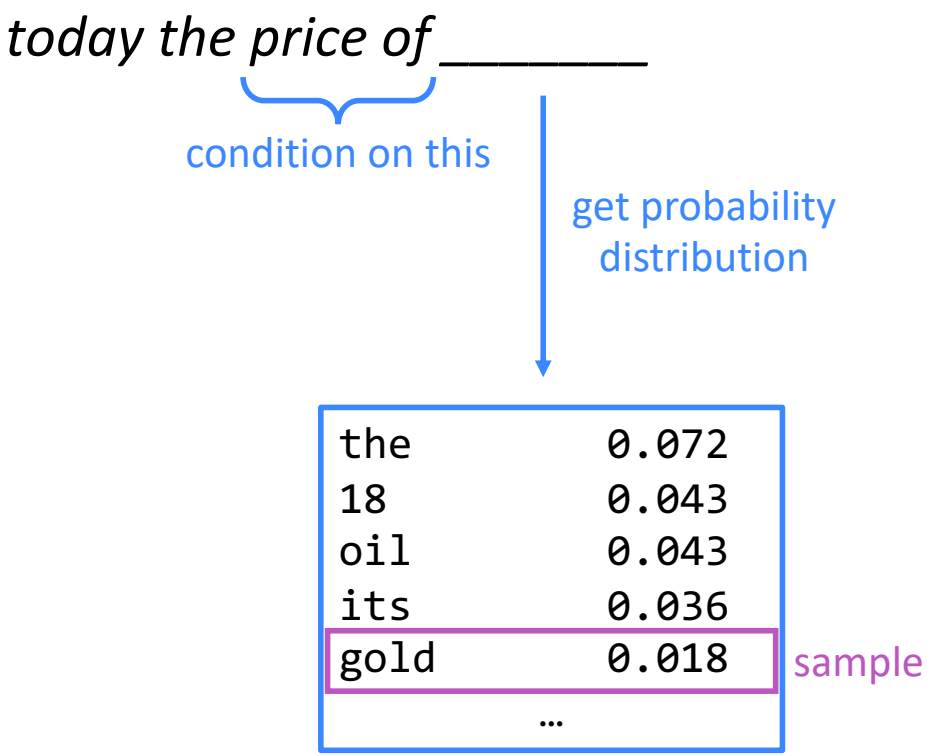


26

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Generating text with a n-gram Language Model

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



27

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Generating text with a n-gram Language Model

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

today the price of gold _____

28

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

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…

29

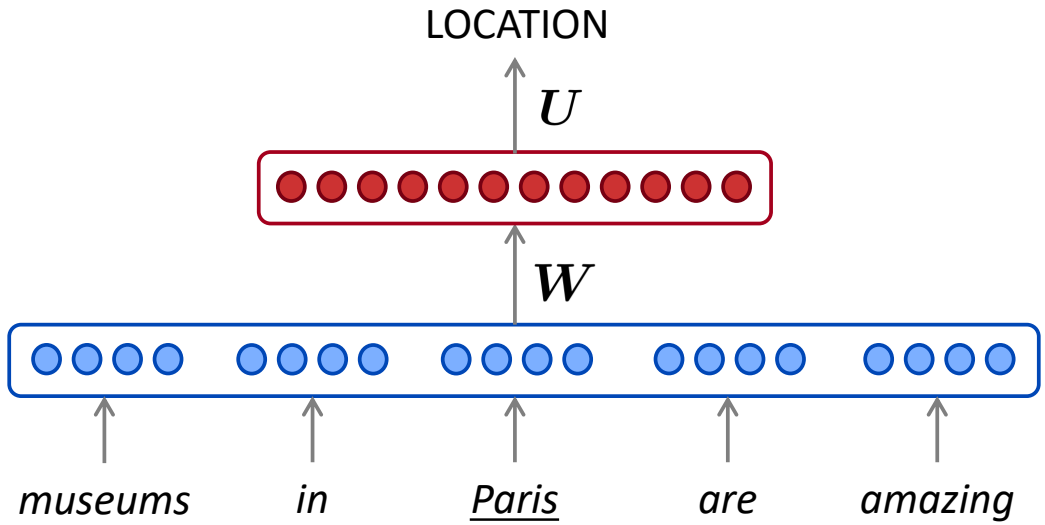
Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Content

How to build a *neural* Language Model?

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

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



Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

A fixed-window neural Language Model

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

discard

fixed window

31

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

A fixed-window neural Language Model

output distribution

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

hidden layer

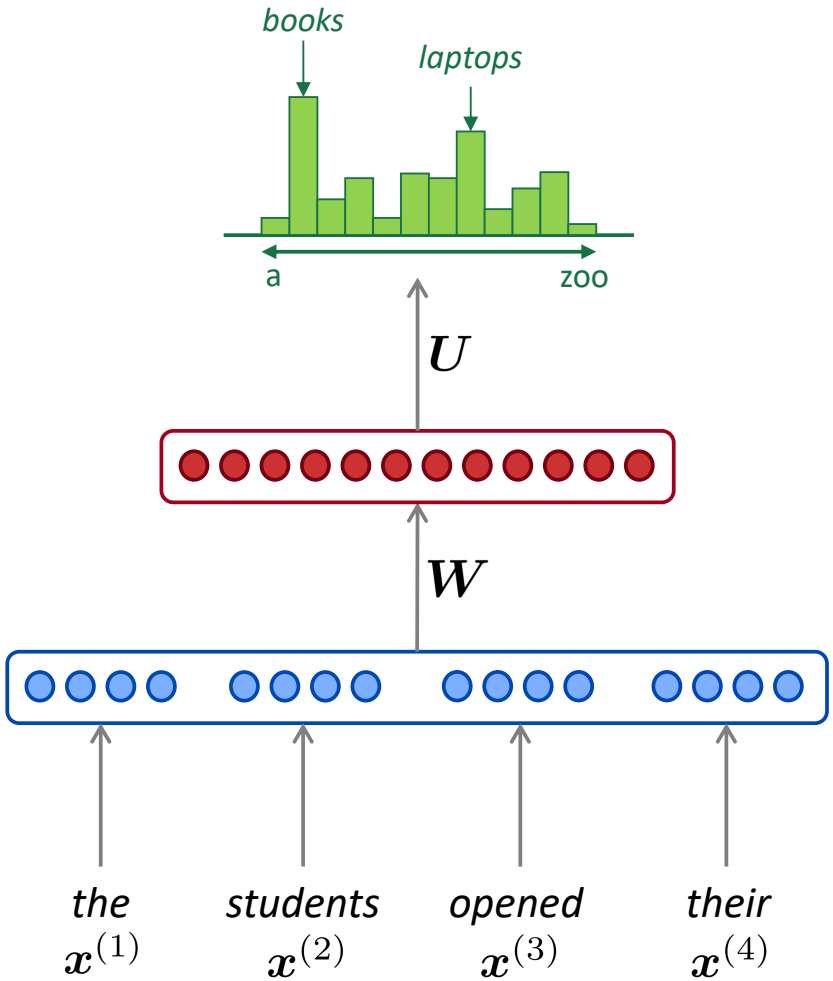
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

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

words / one-hot vectors

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



32

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

A fixed-window neural Language Model

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

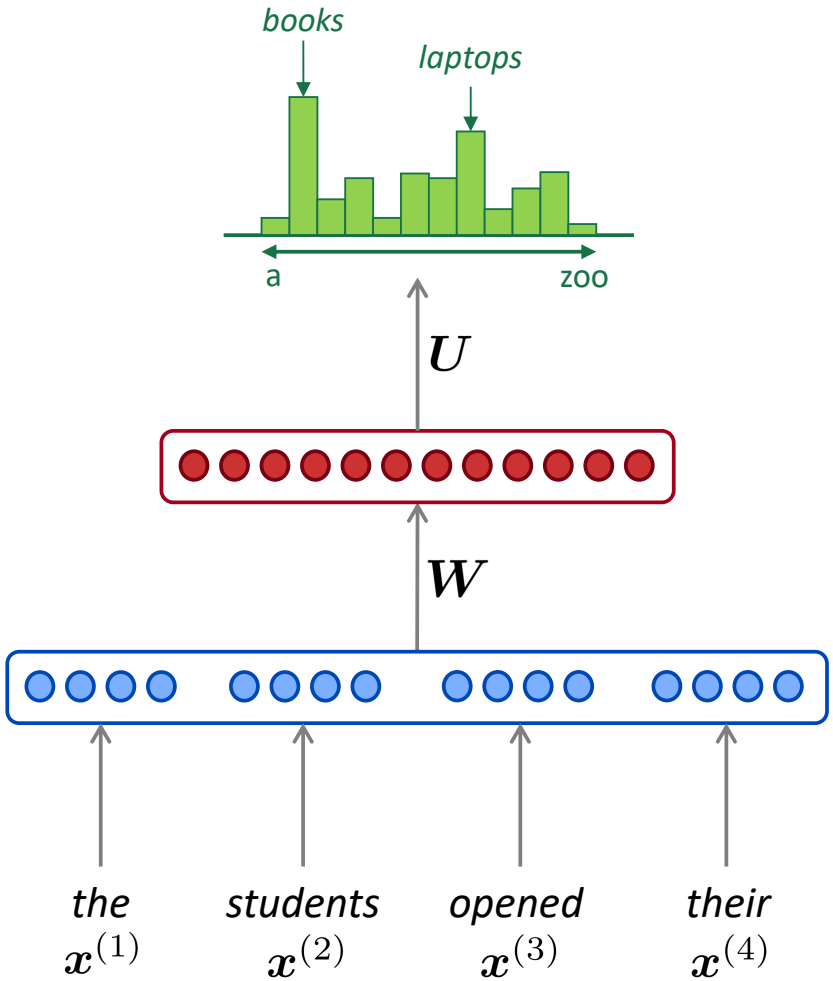
Improvements over *n*-gram LM:

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

Remaining **problems**:

- Fixed window is too small
- Enlarging window enlarges $\boldsymbol{W}$
- Window can never be large enough!
- $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$ are multiplied by completely different weights in $\boldsymbol{W}$. No symmetry in how the inputs are processed.

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



33

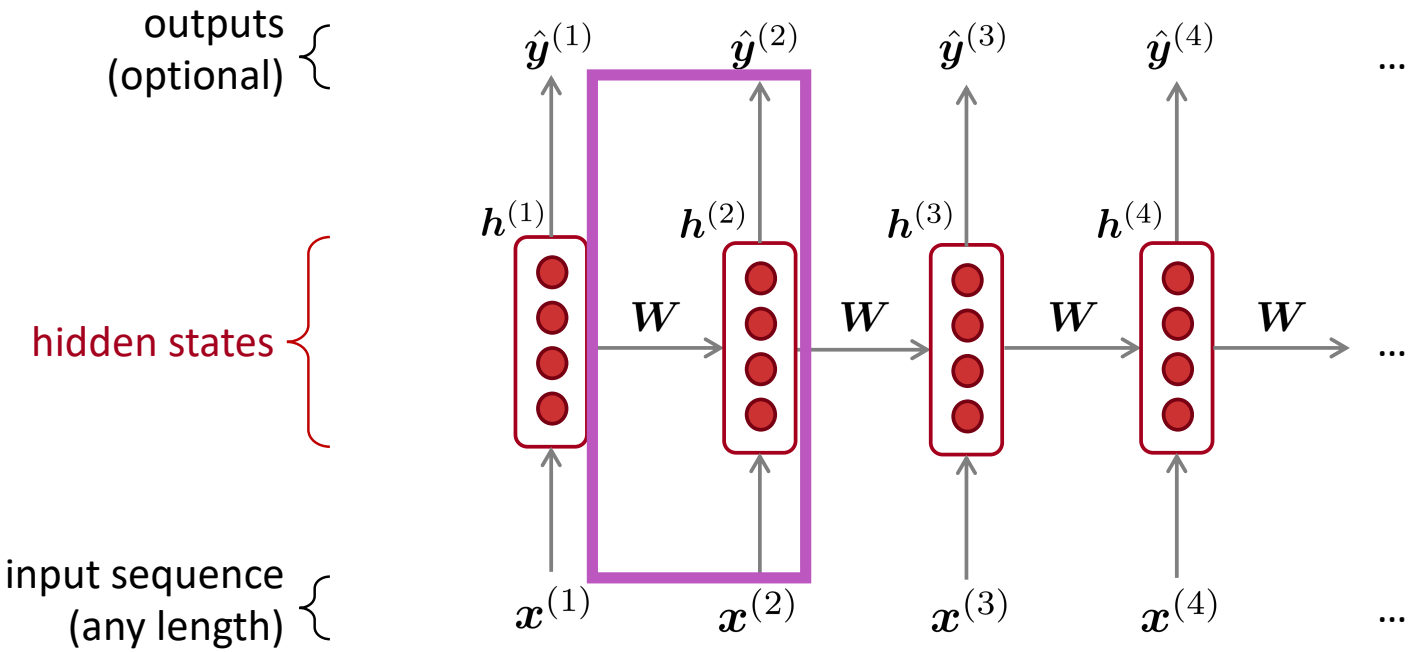
Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Content

Recurrent Neural Networks (RNN)

A family of neural architectures

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

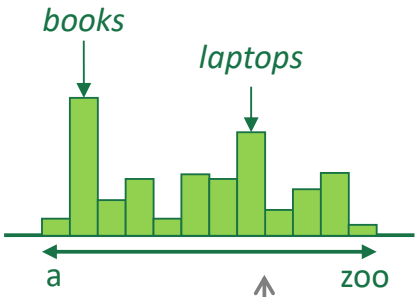


34

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

A Simple RNN Language Model

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



output distribution

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

hidden states

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

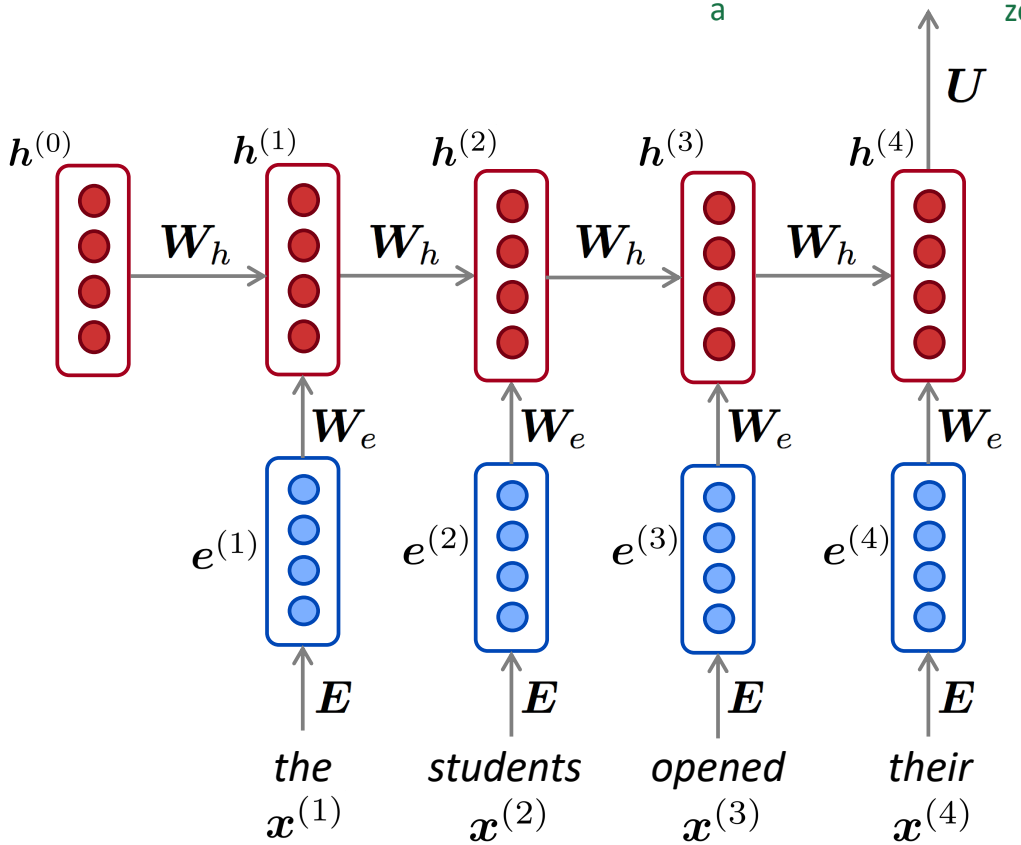
$\boldsymbol{h}^{(0)}$ is the initial hidden state

word embeddings

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

words / one-hot vectors

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



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

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

RNN Language Models

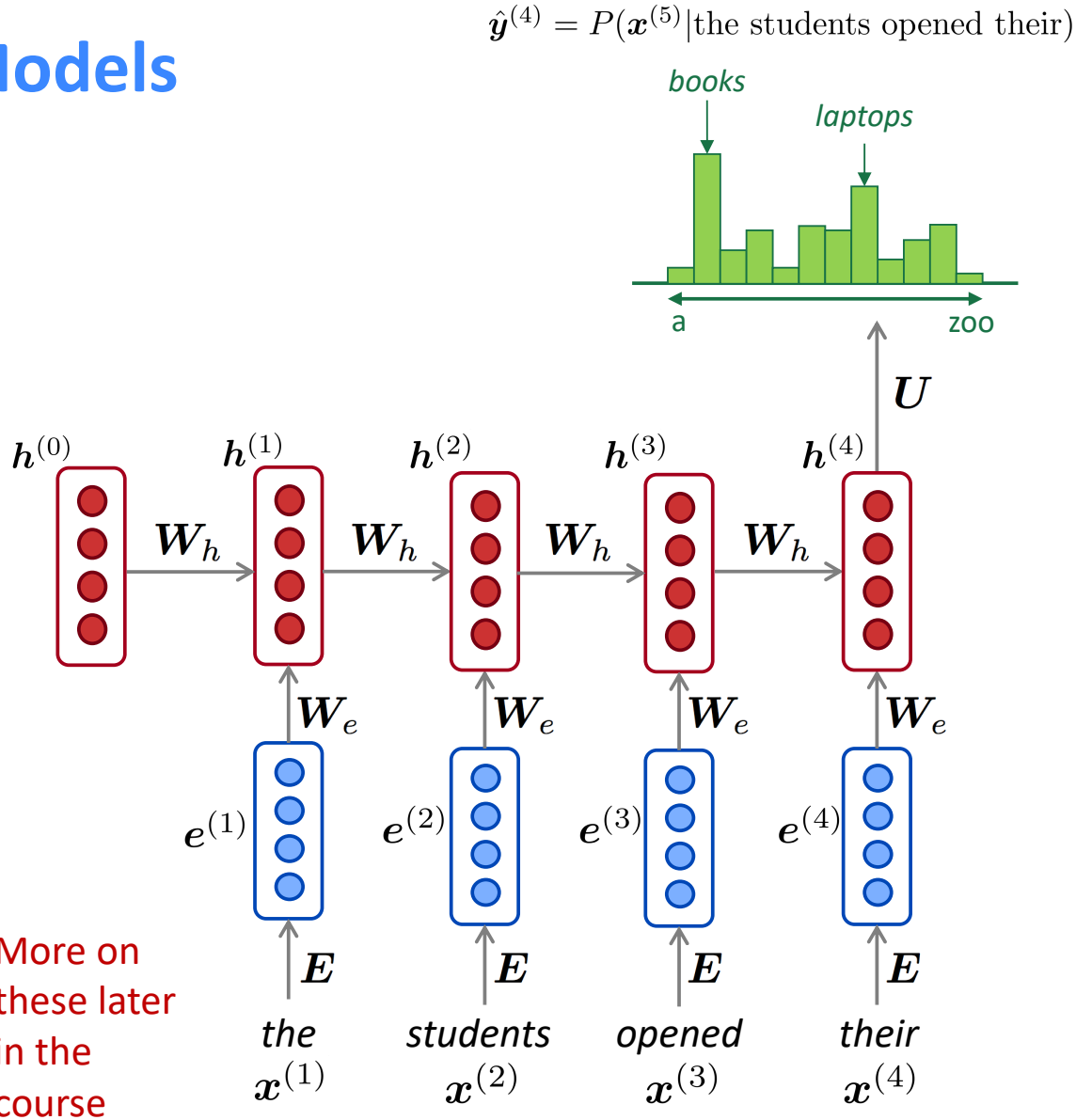
RNN **Advantages**:

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

RNN **Disadvantages**:

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

More on these later in the course



Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Content

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

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

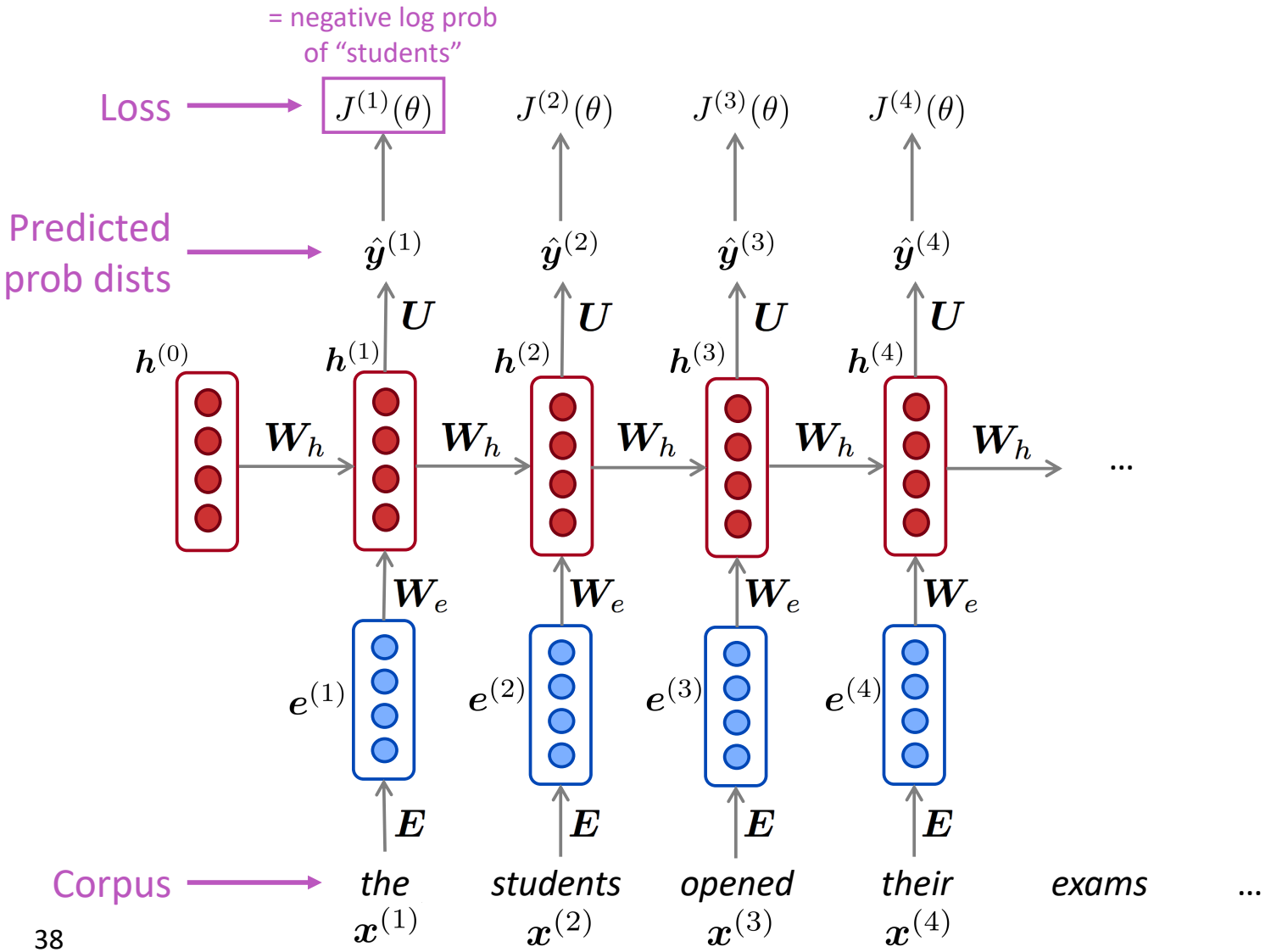
- Average this to get overall loss for entire training set:

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

37

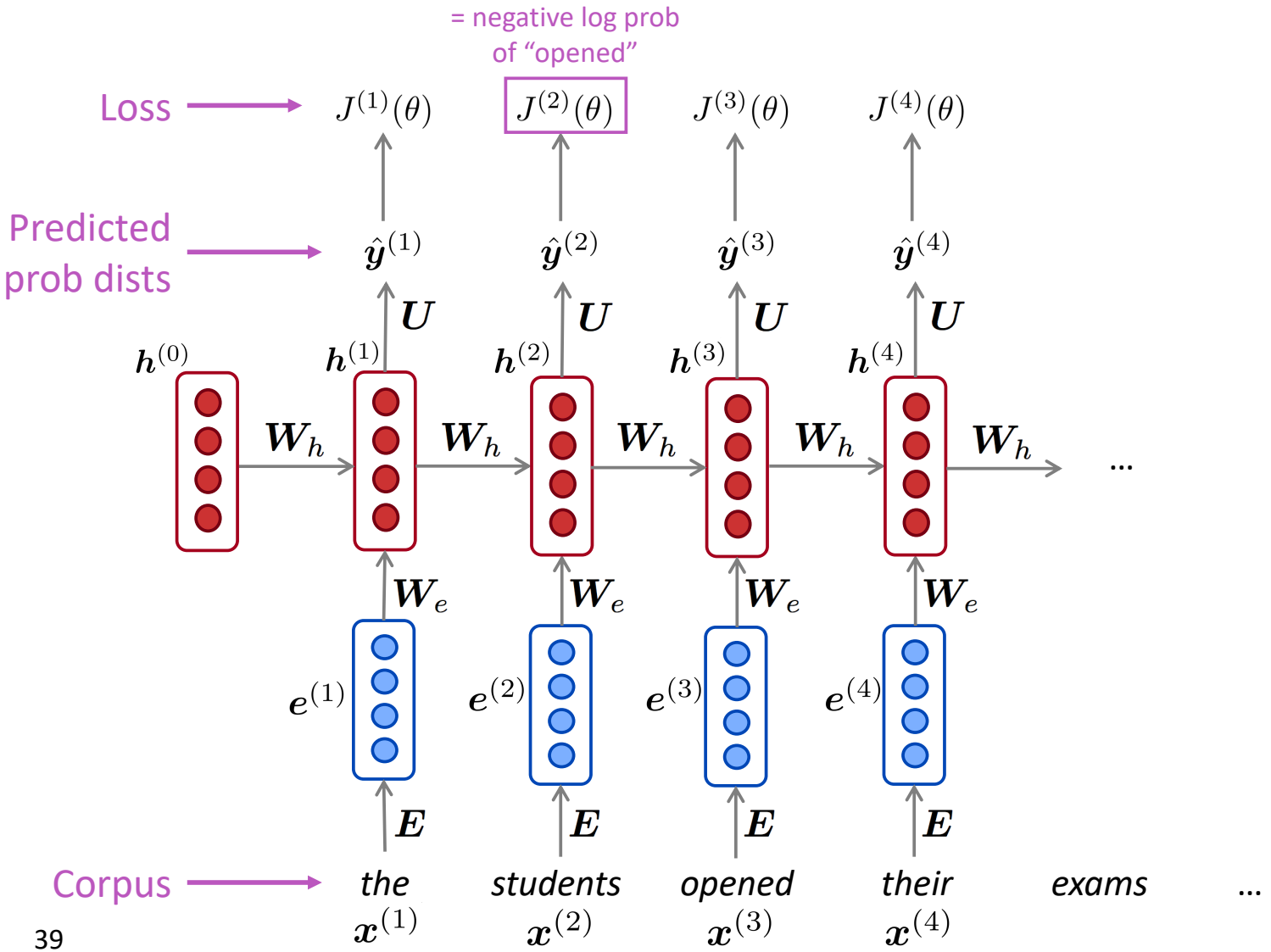
Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Training an RNN Language Model



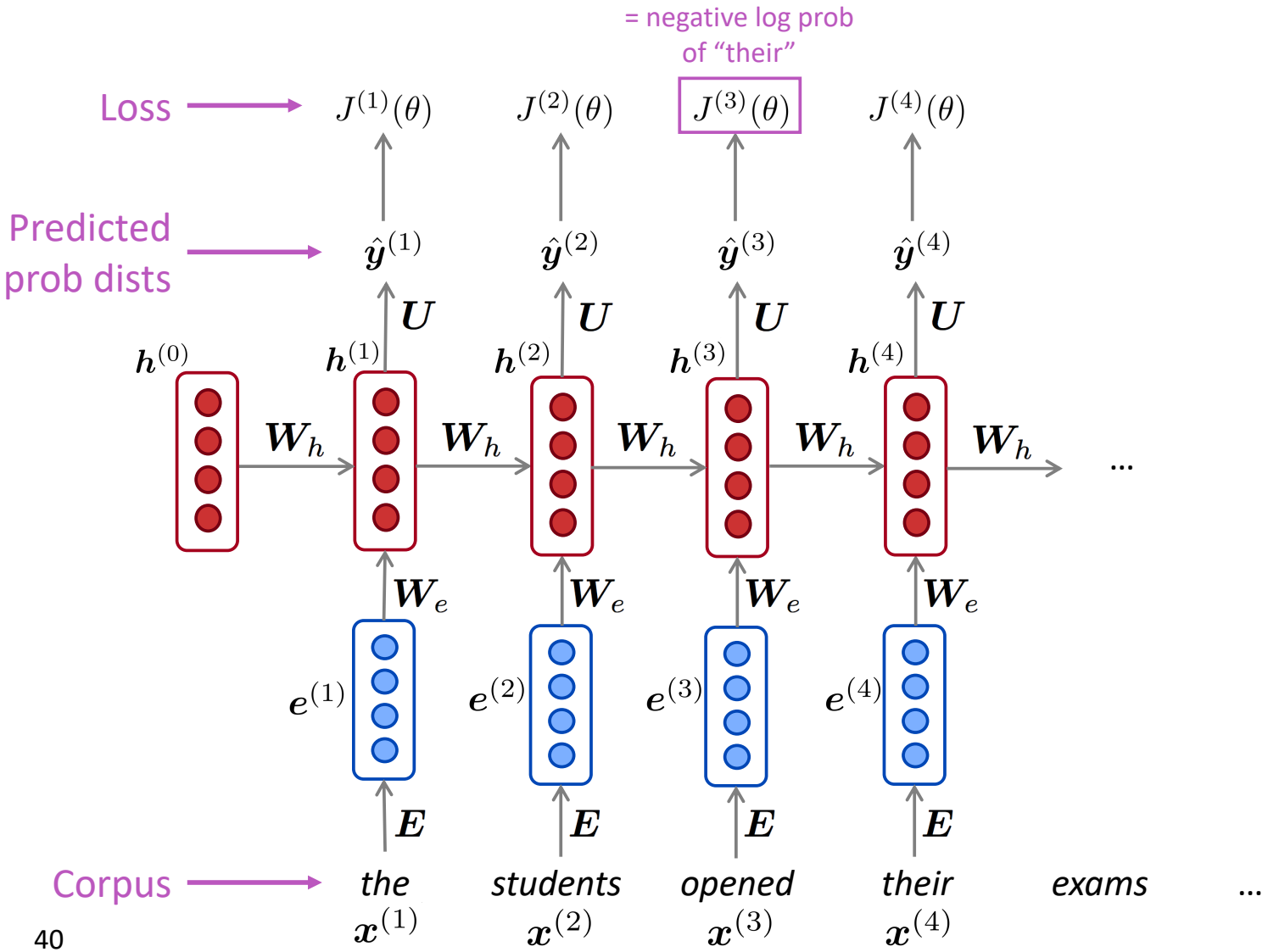
Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Training an RNN Language Model



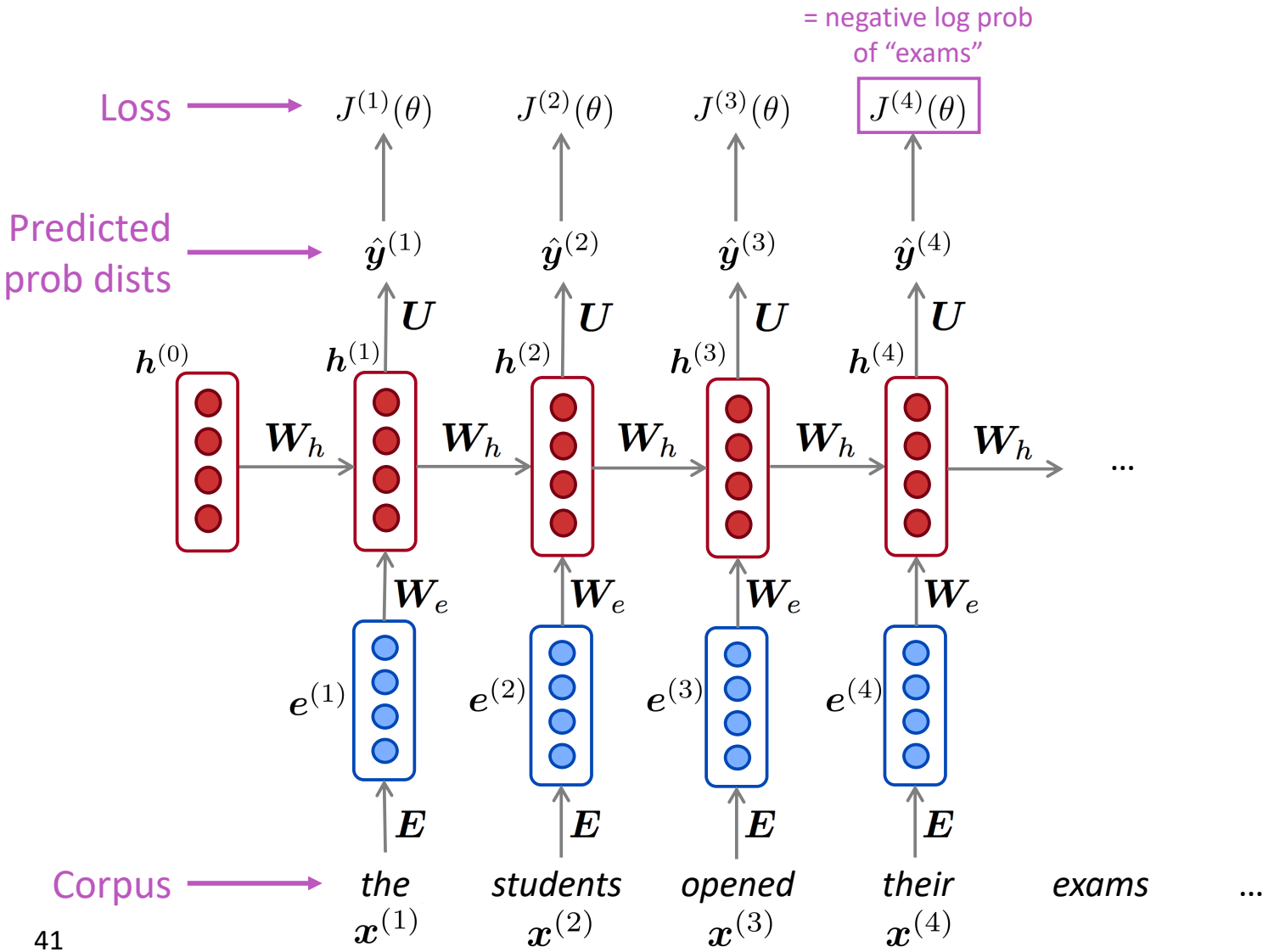
Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Training an RNN Language Model



Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Training an RNN Language Model

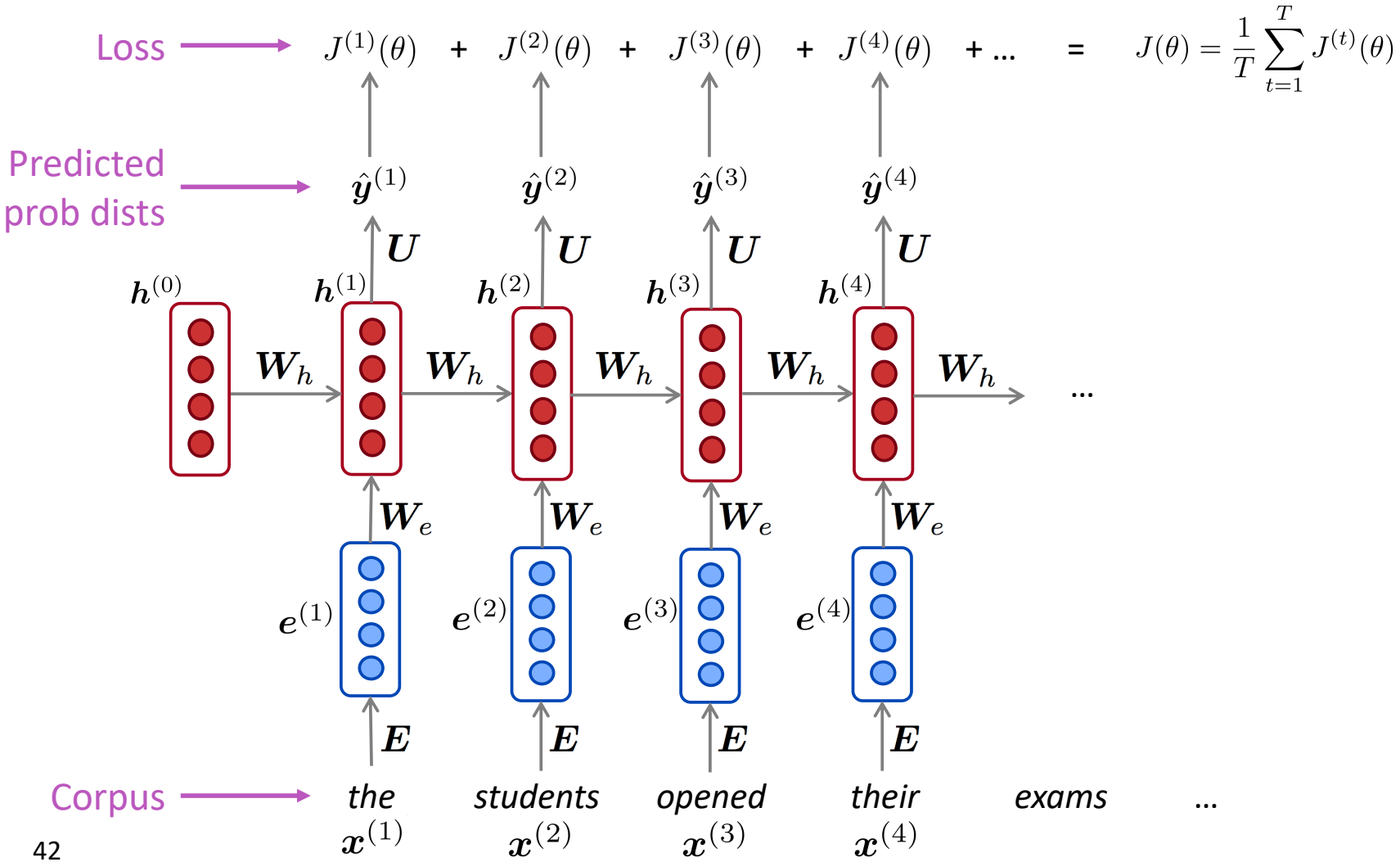


41

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Training an RNN Language Model

"Teacher forcing"



Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

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)

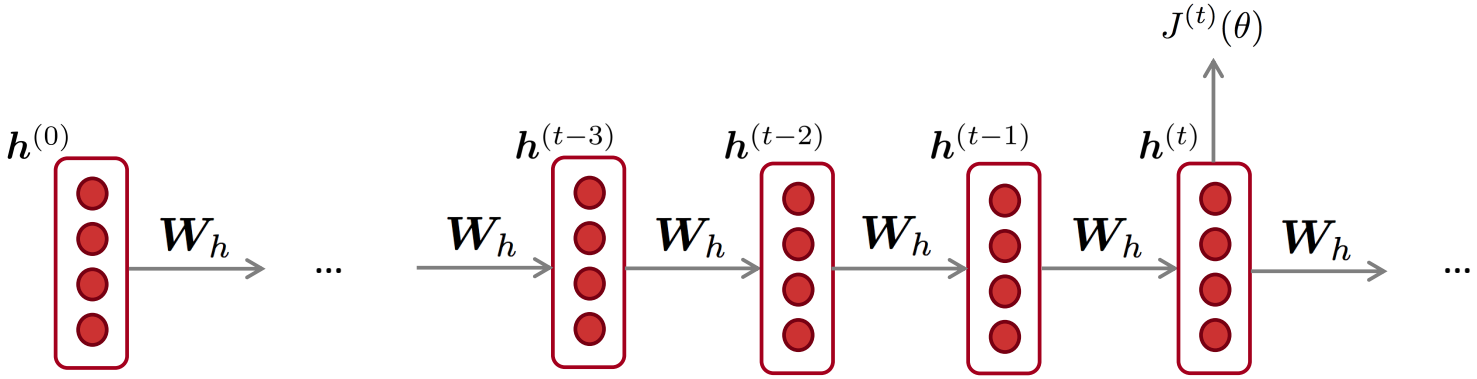
- Recall: Stochastic Gradient Descent allows us to compute loss and gradients for small chunk of data, and update.

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

43

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Backpropagation for RNNs



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

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

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

Why?

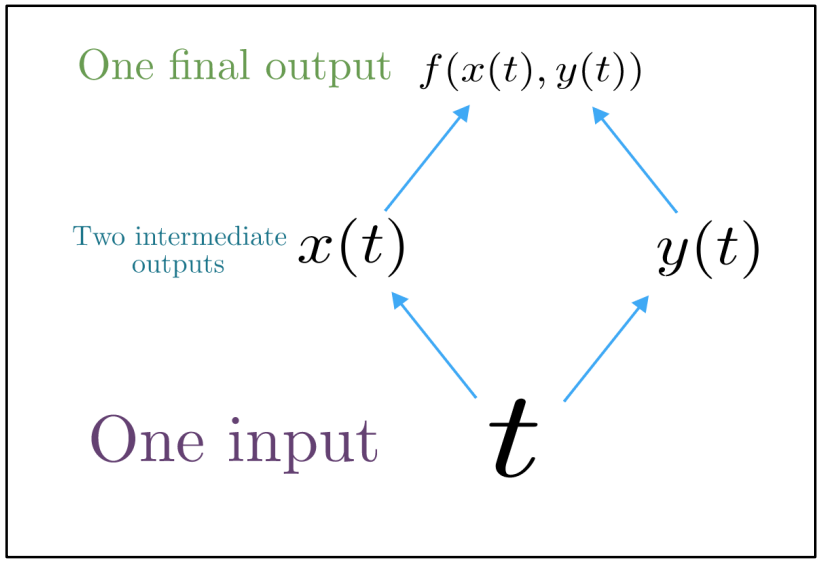
44

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

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

45

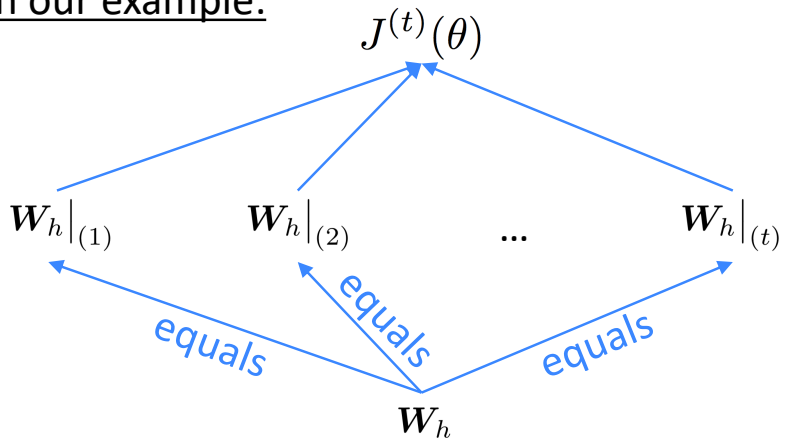
Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

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:



Apply the multivariable chain rule:

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

= 1

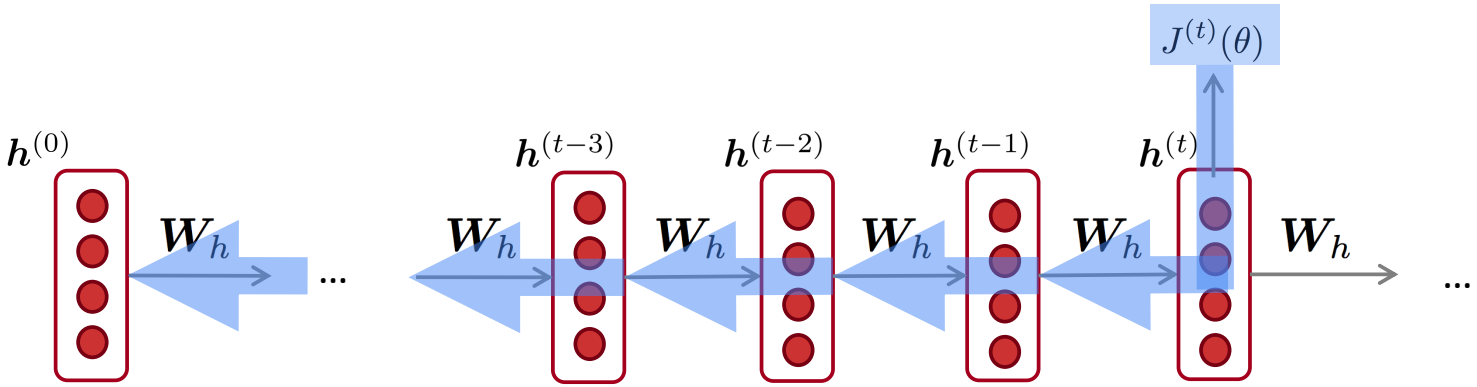
$$= \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \boldsymbol{W}_h}\Big|_{(i)}$$

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

46

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Backpropagation for RNNs



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

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]

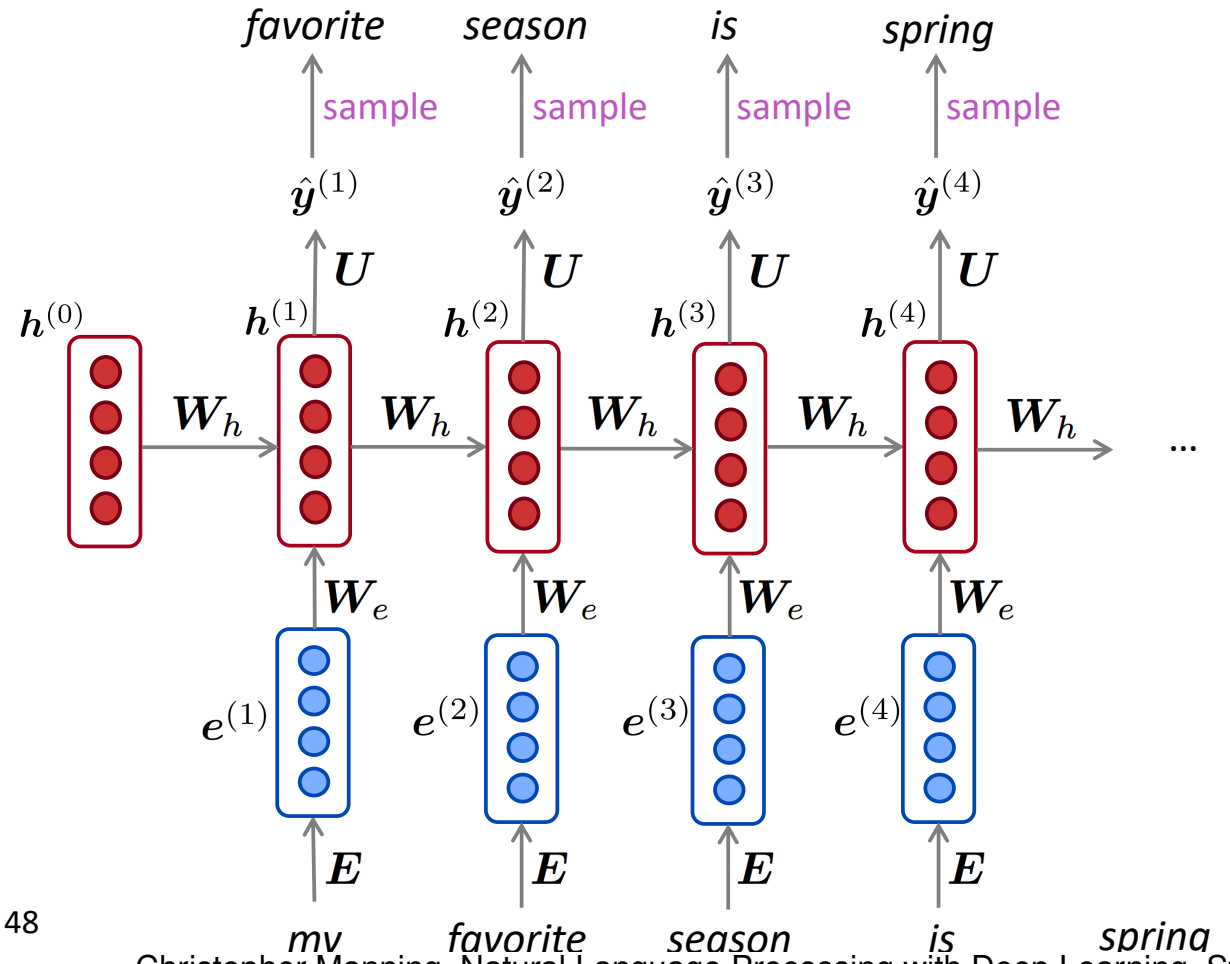
47

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Content

Generating text with a RNN Language Model

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



48

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Generating text with a RNN Language Model

Let's have some fun!

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

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

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

49

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Generating text with a RNN Language Model

Let's have some fun!

- You can train a RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on *Harry Potter*:

"Sorry," Harry shouted, panicking—"I'll leave those brooms in London, are they?"

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

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

50

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Generating text with a RNN Language Model

Let's have some fun!

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

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

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

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

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

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

51

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Generating text with a RNN Language Model

Let's have some fun!

- You can train a RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on paint color names:

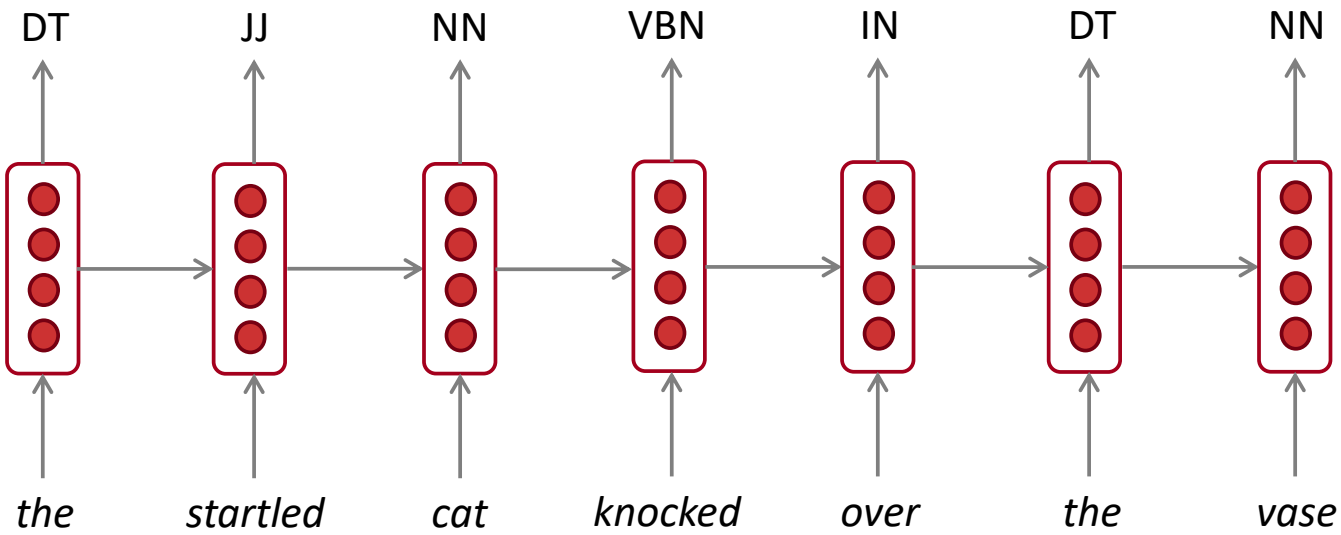
Ghasty Pink 231 137 165	Sand Dan 201 172 143
Power Gray 151 124 112	Grade Bat 48 94 83
Navel Tan 199 173 140	Light Of Blast 175 150 147
Bock Coe White 221 215 236	Grass Bat 176 99 108
Horble Gray 178 181 196	Sindis Poop 204 205 194
Homestar Brown 133 104 85	Dope 219 209 179
Snader Brown 144 106 74	Testing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

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

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

RNNs can be used for tagging

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

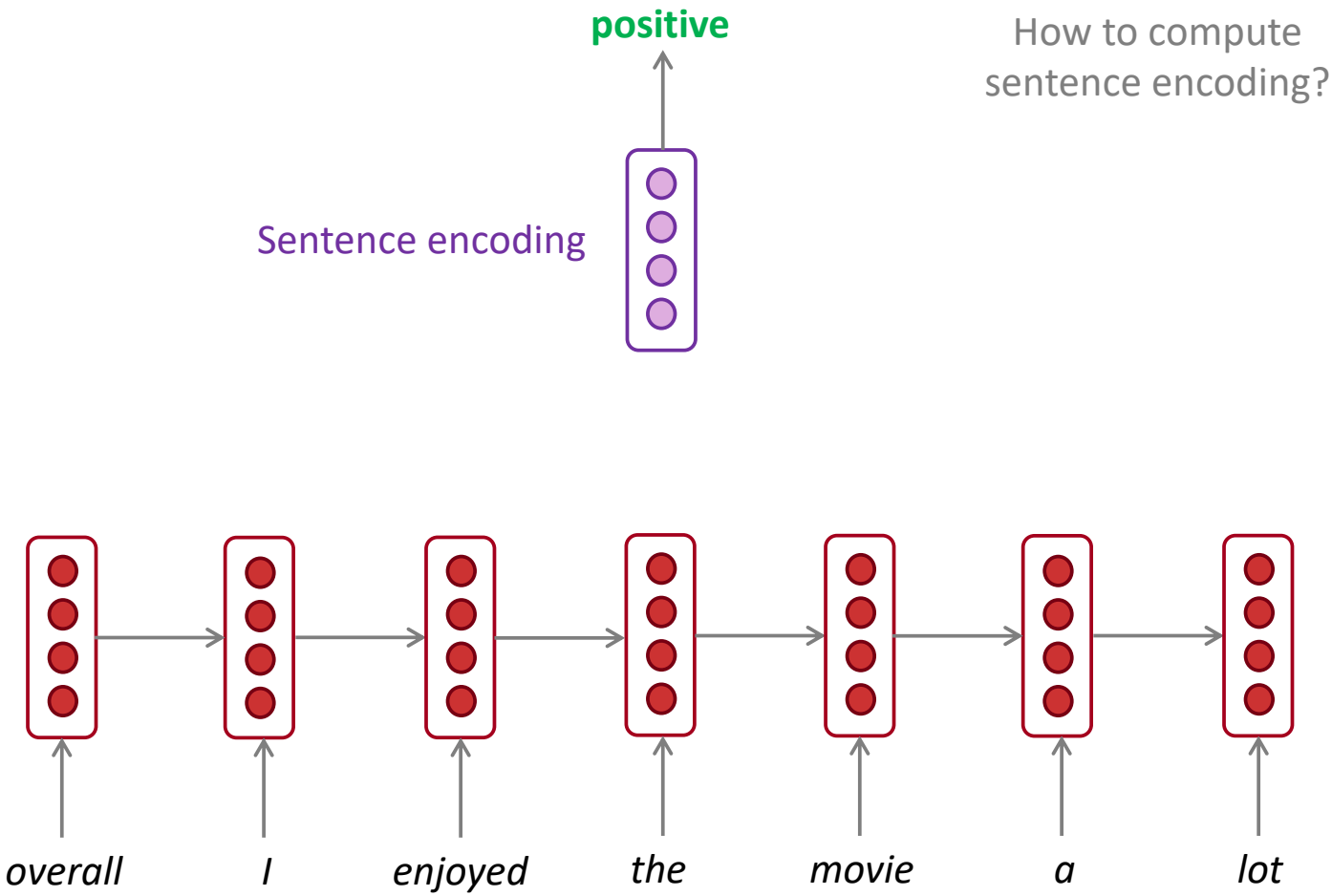


57

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

RNNs can be used for sentence classification

e.g. sentiment classification

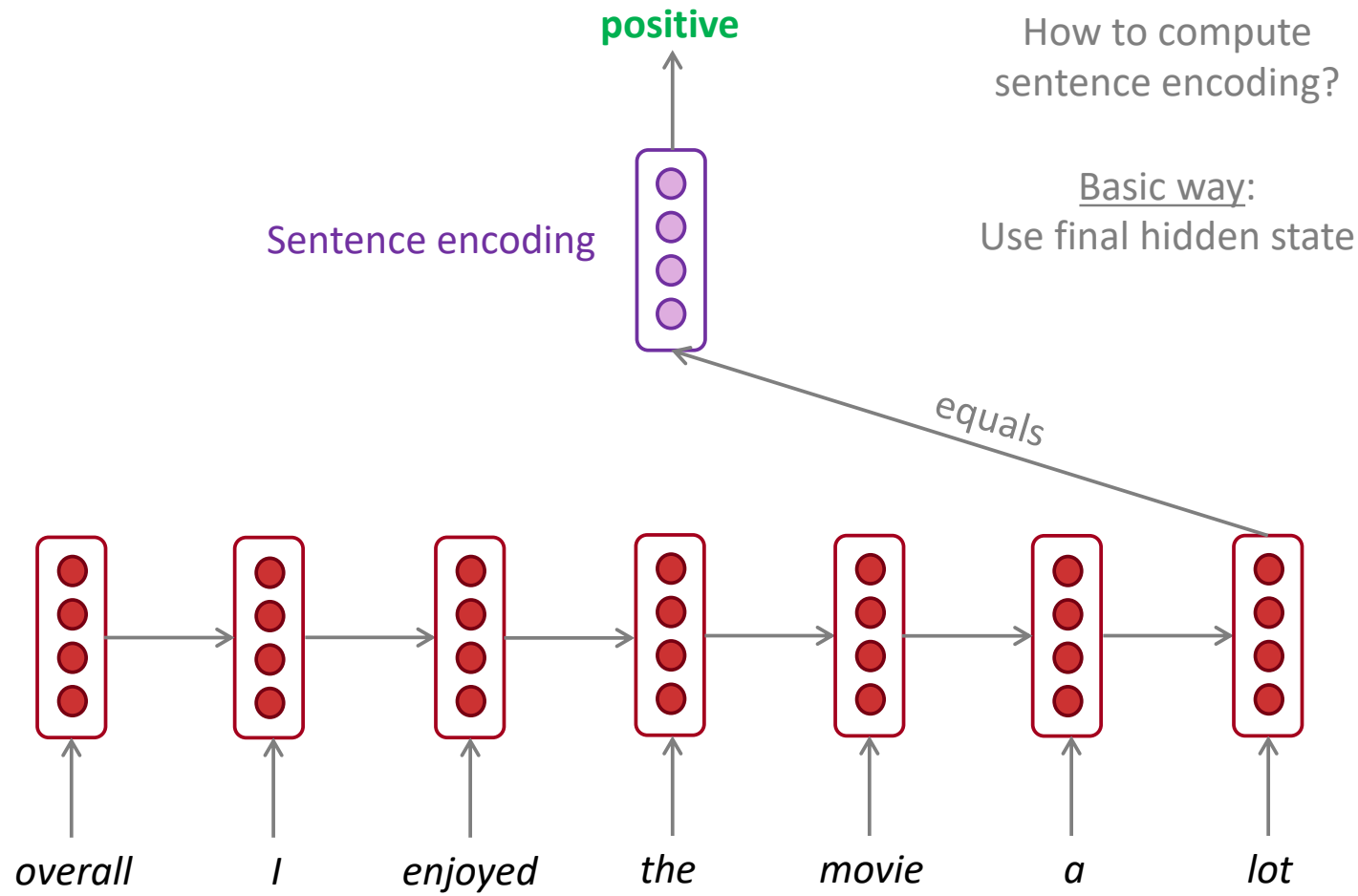


58

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

RNNs can be used for sentence classification

e.g. sentiment classification

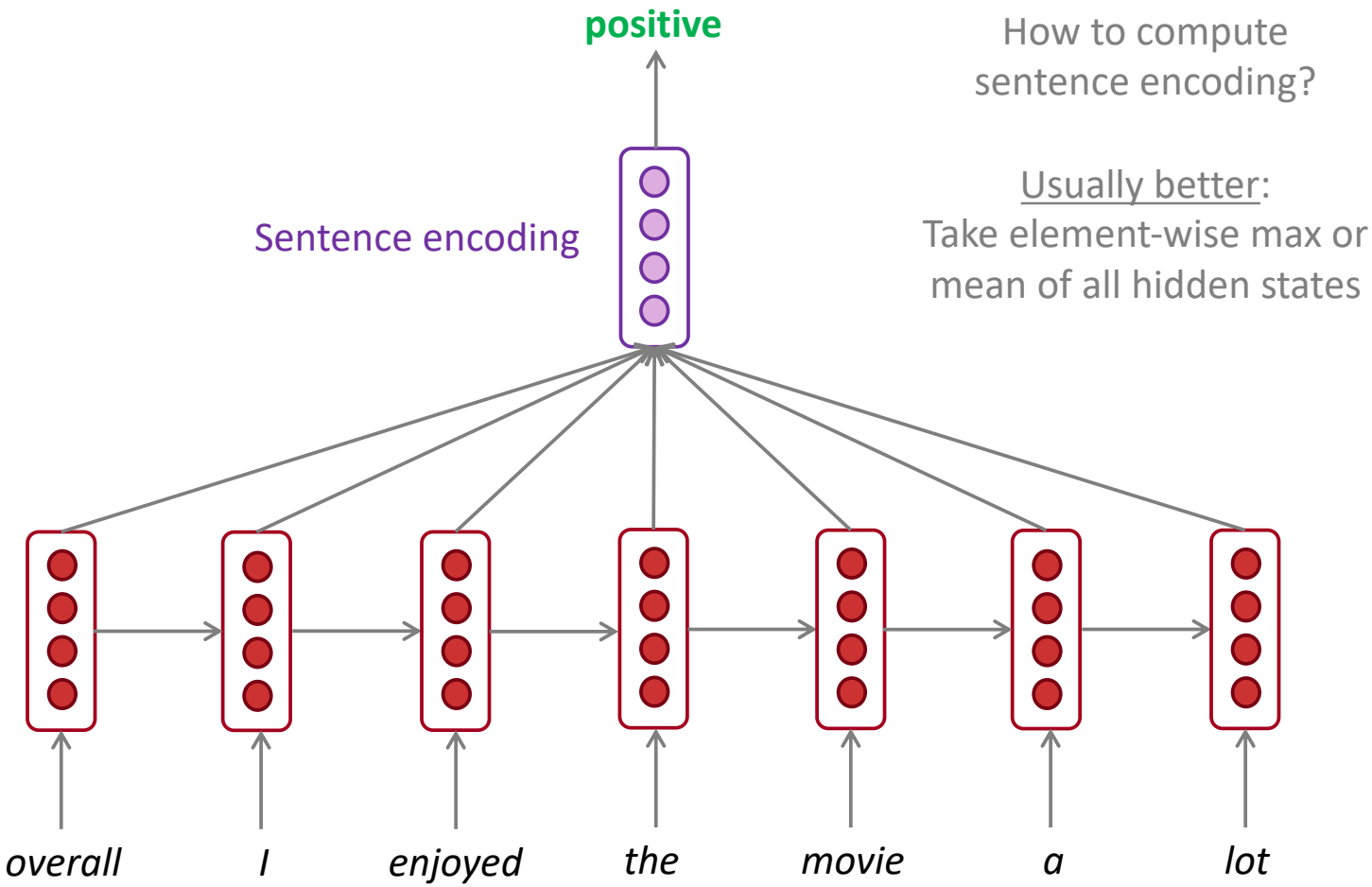


59

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

RNNs can be used for sentence classification

e.g. sentiment classification

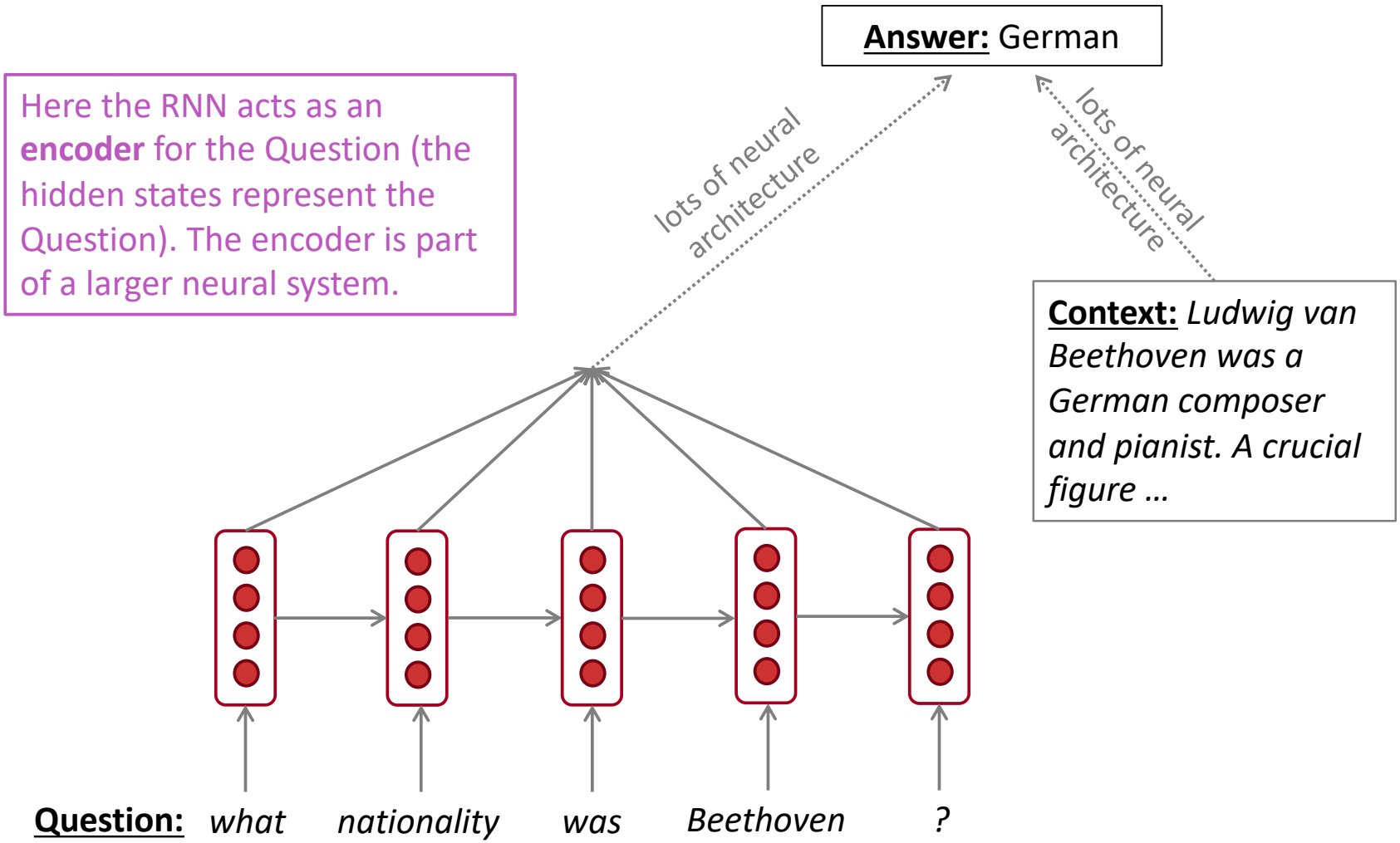


60

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

RNNs can be used as an encoder module

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

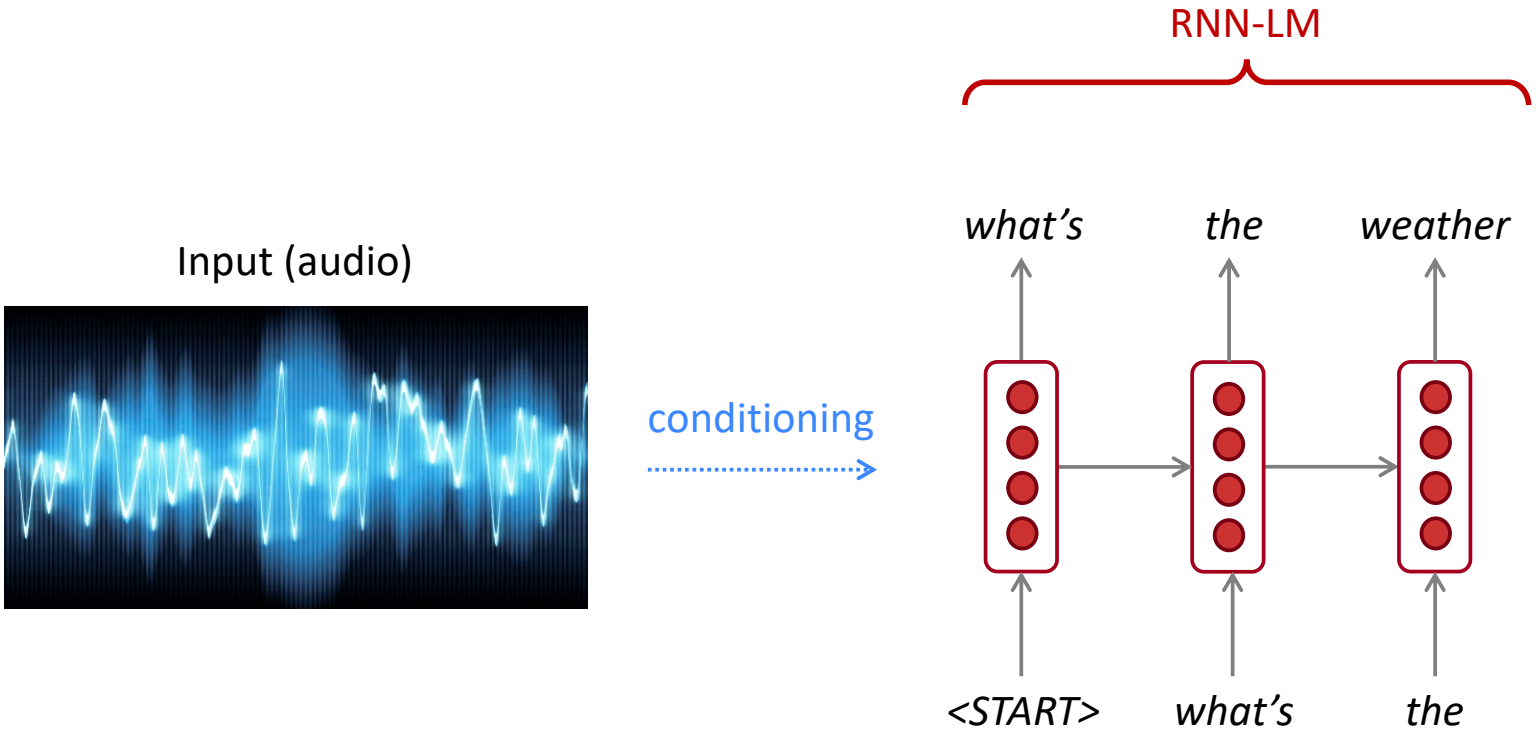


61

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

RNN-LMs can be used to generate text

e.g. speech recognition, machine translation, summarization



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

62

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

A note on terminology

The RNN described in this lecture = simple/vanilla/Elman RNN

Next lecture: You will learn about other RNN flavors

like GRU  and LSTM  and multi-layer RNNs 

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

63

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Content

Evaluating Language Models

- The standard evaluation metric for Language Models is perplexity.

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

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!

53

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

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

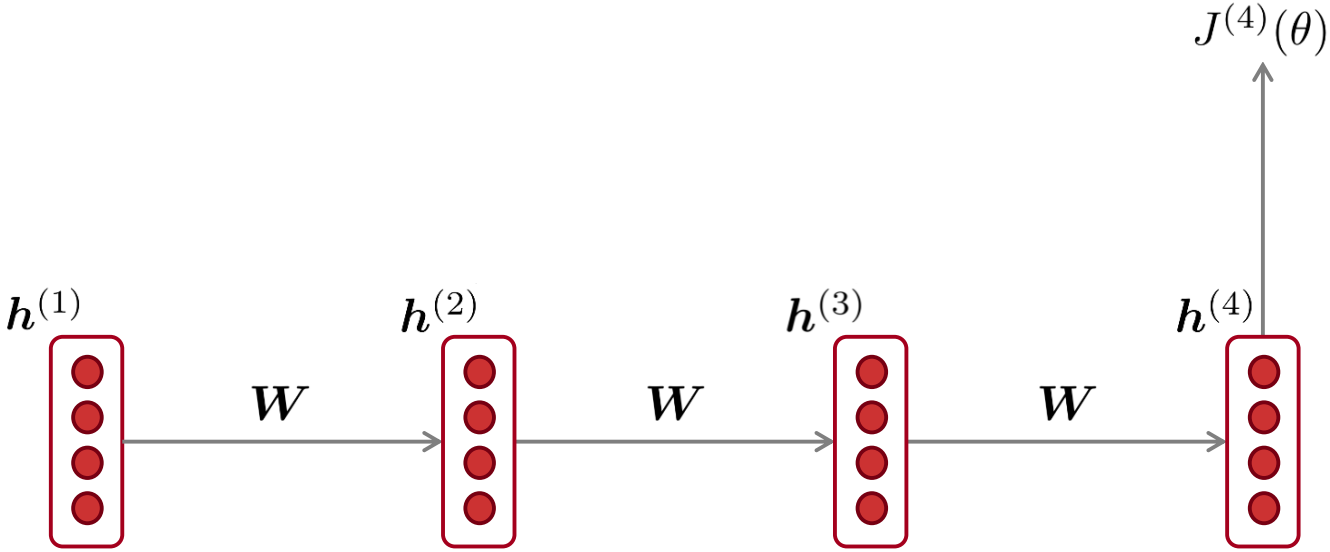
↓ Perplexity improves (lower is better)

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

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Content

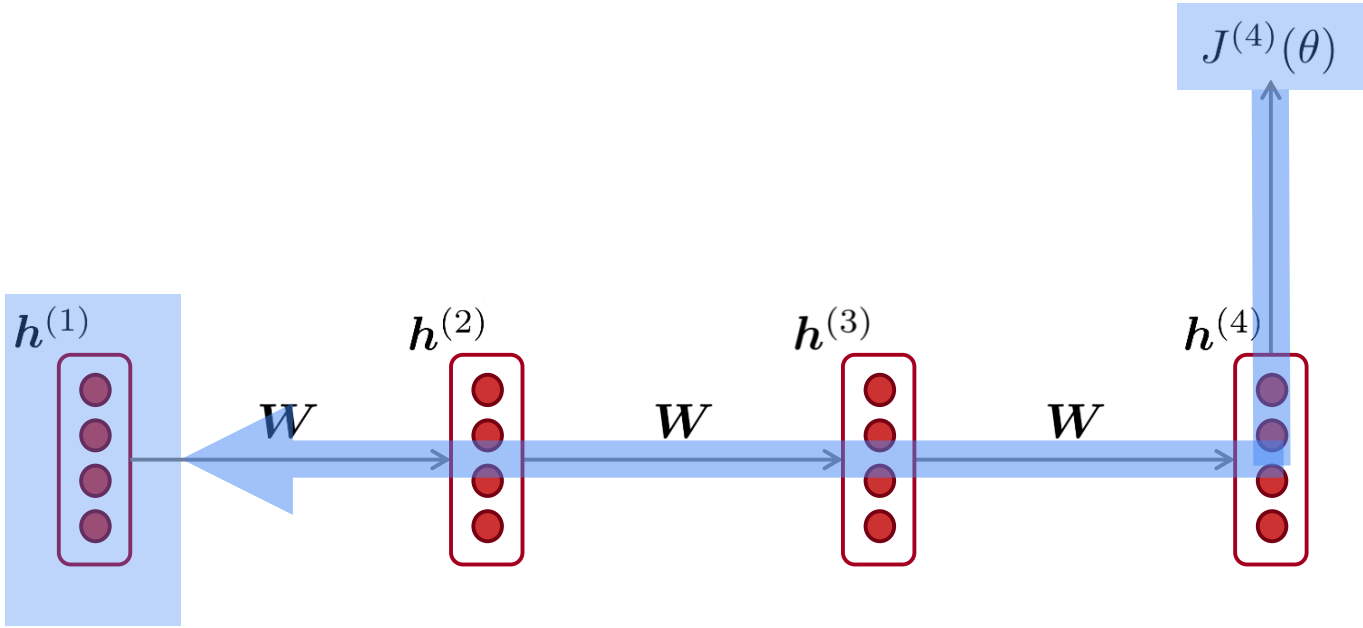
Vanishing gradient intuition



6

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Vanishing gradient intuition

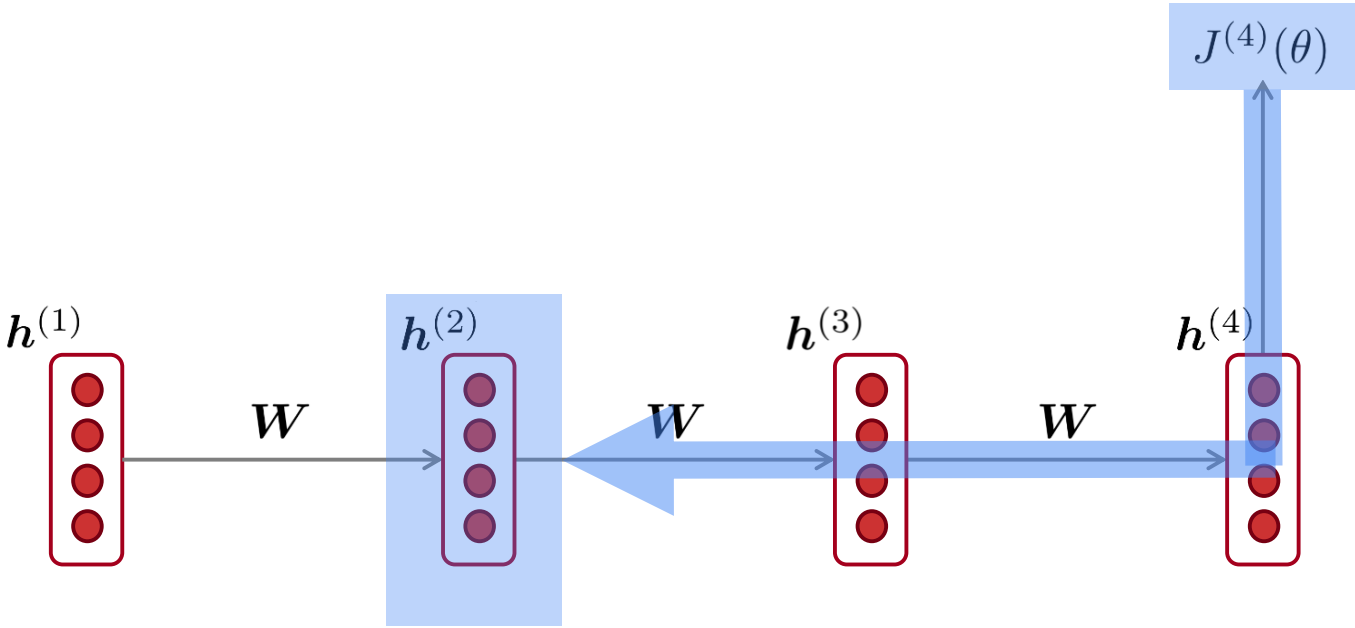


$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \ ?$$

7

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Vanishing gradient intuition



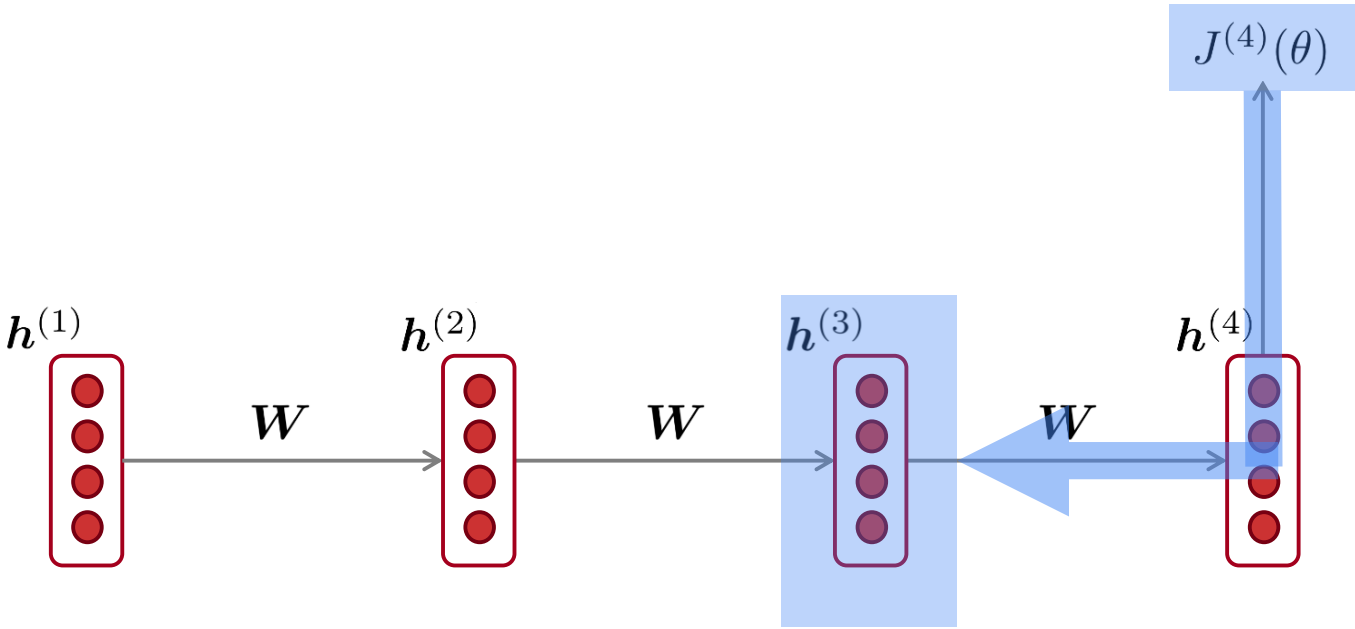
$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial J^{(4)}}{\partial h^{(2)}}$$

chain rule!

8

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Vanishing gradient intuition



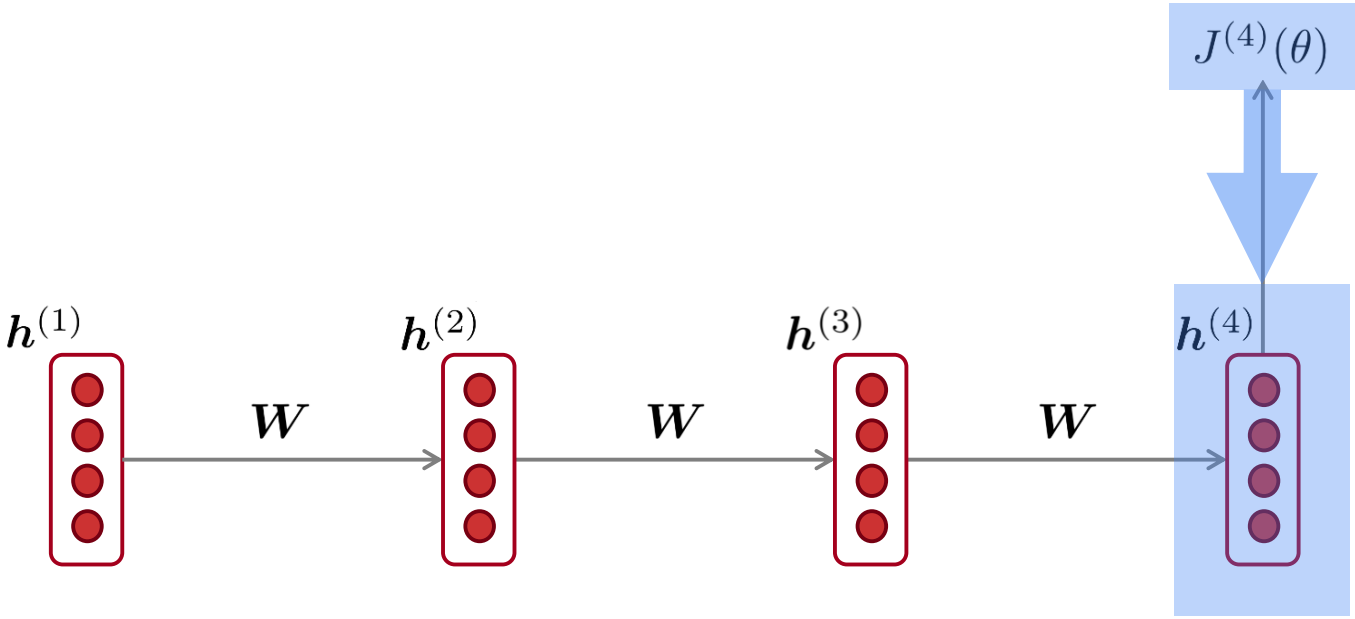
$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial J^{(4)}}{\partial h^{(3)}}$$

chain rule!

9

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Vanishing gradient intuition



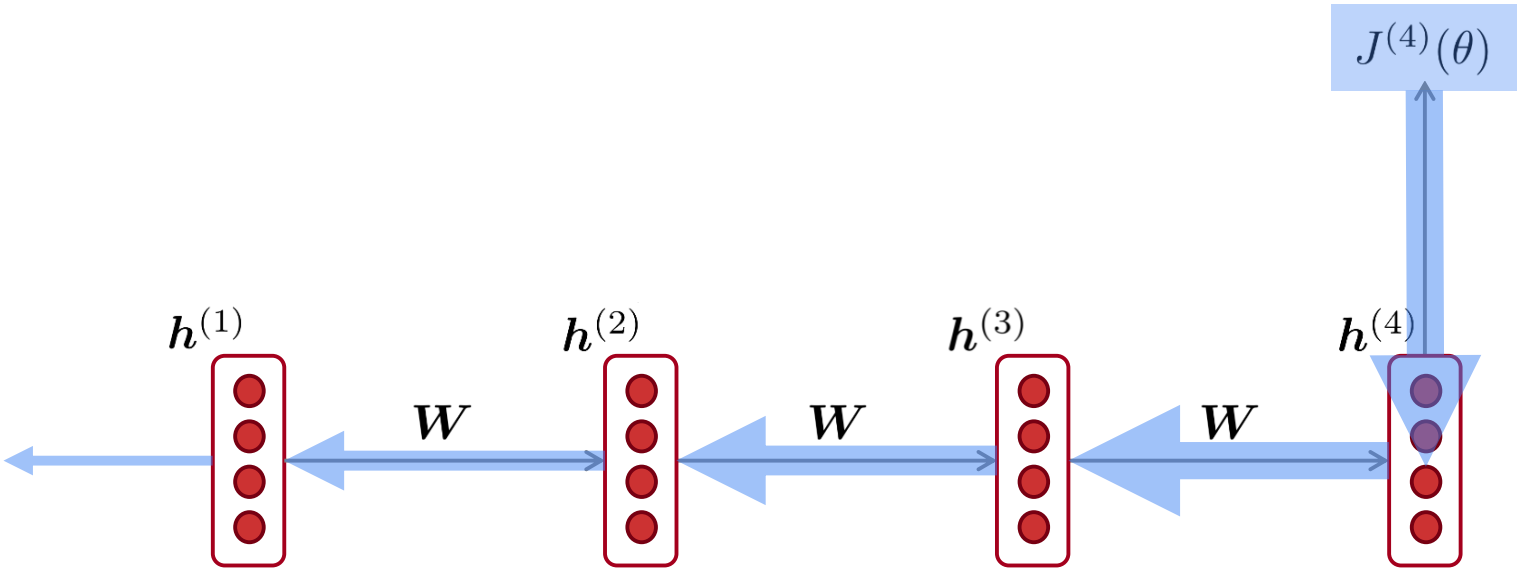
$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial J^{(4)}}{\partial h^{(4)}}$$

chain rule!

10

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

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

11

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Vanishing gradient proof sketch

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

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

- 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*.

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

$$= \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \boldsymbol{W}_h^{(i-j)} \prod_{j<t\leq i} \operatorname{diag}\left(\sigma'\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)\right)$$ (value of $\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}$)

If W_h is small, then this term gets vanishingly small as *i* and *j* get further apart

12

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

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Vanishing gradient proof sketch

- Consider matrix L2 norms:

$$\left\|\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}}\right\| \leq \left\|\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}}\right\| \|\boldsymbol{W}_h\|^{(i-j)} \prod_{j<t\leq i} \left\|\operatorname{diag}\left(\sigma'\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)\right)\right\|$$

- Pascanu et al showed that that if the largest eigenvalue of W_h is less than 1, then the gradient $\left\|\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}}\right\|$ will shrink exponentially
  - Here the bound is 1 because we have sigmoid nonlinearity

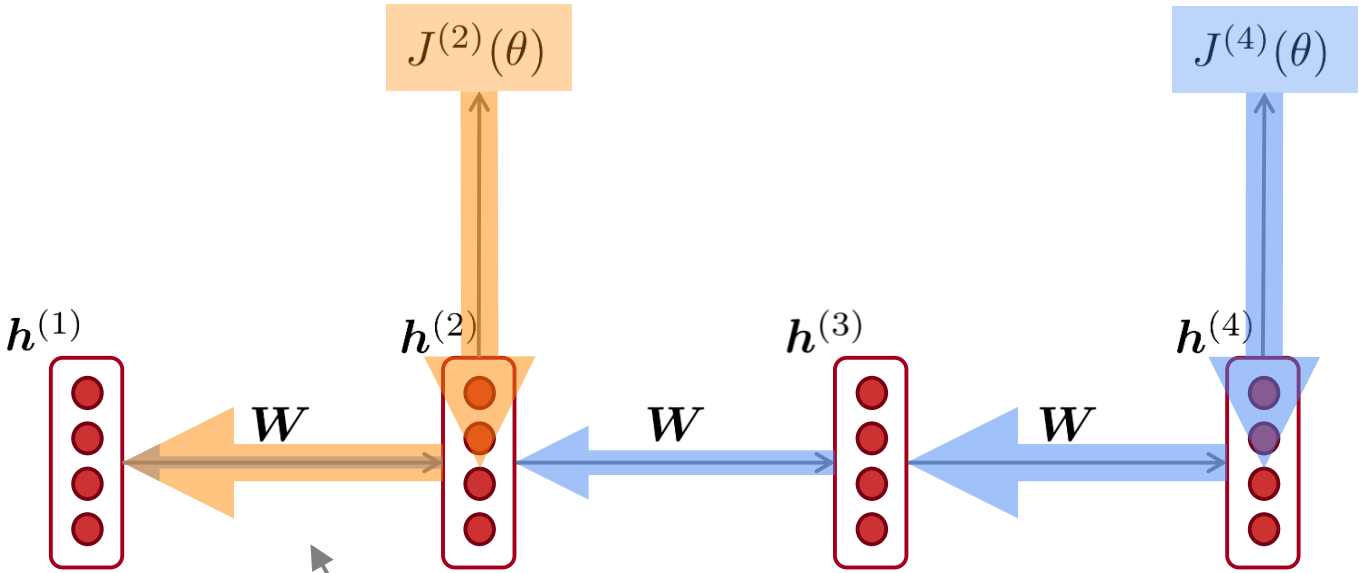
- There's a similar proof relating a largest eigenvalue >1 to exploding gradients

13

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

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Why is vanishing gradient a problem?



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

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

14

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

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*

15

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

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.

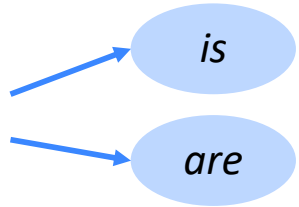
- But if gradient is small, the model can't learn this dependency
  - So the model is unable to predict similar long-distance dependencies at test time

16

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Effect of vanishing gradient on RNN-LM

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



- **Correct answer**: *The writer of the books is planning a sequel*

- **Syntactic recency:** *The writer of the books is* (correct)

- **Sequential recency:** *The writer of the books are* (incorrect)

- Due to vanishing gradient, RNN-LMs are better at learning from sequential recency than syntactic recency, so they make this type of error more often than we'd like [Linzen et al 2016]

17

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

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

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)

18

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

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

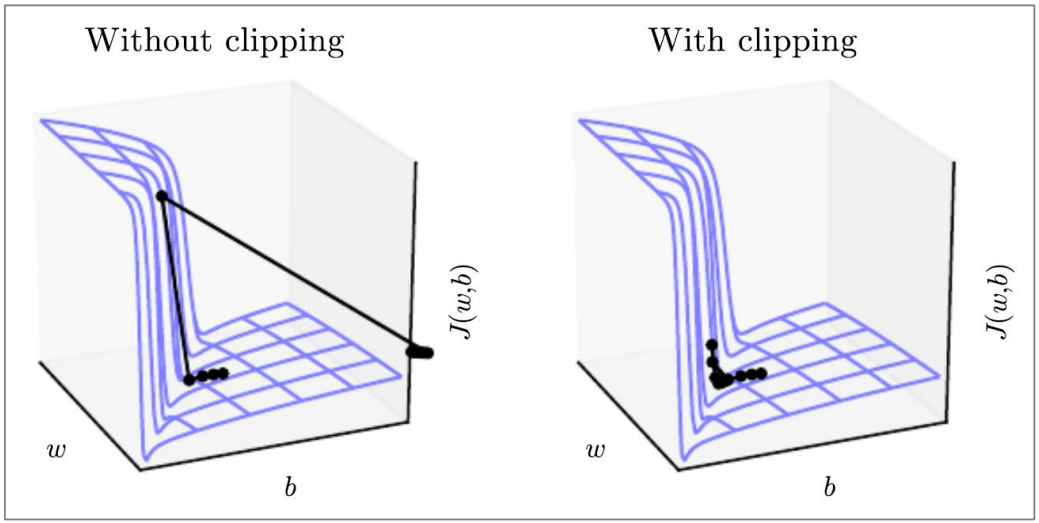
- Intuition: take a step in the same direction, but a smaller step

19

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

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

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

20

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

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

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?

21

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Content

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

22

"Long short-term memory", Hochreiter and Schmidhuber, 1997. https://www.bioinf.jku.at/publications/older/2604.pdf

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Long Short-Term Memory (LSTM)

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

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

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

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

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

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

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

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

All these are vectors of same length n

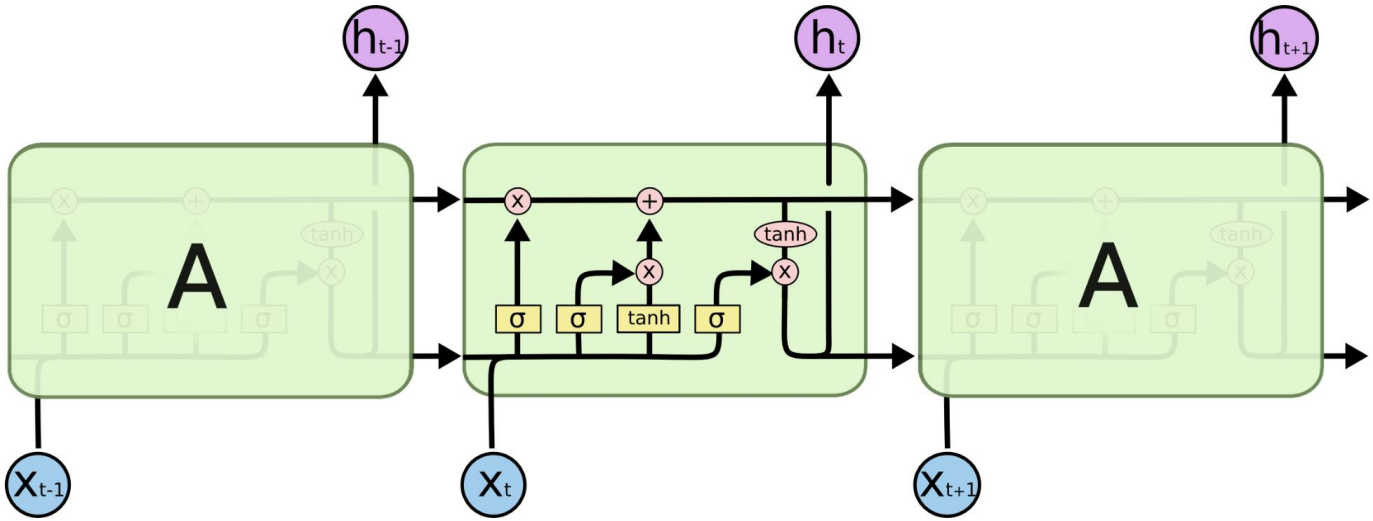
Gates are applied using element-wise product

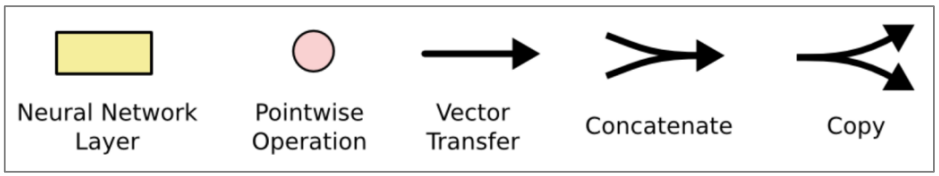
23

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Long Short-Term Memory (LSTM)

You can think of the LSTM equations visually like this:





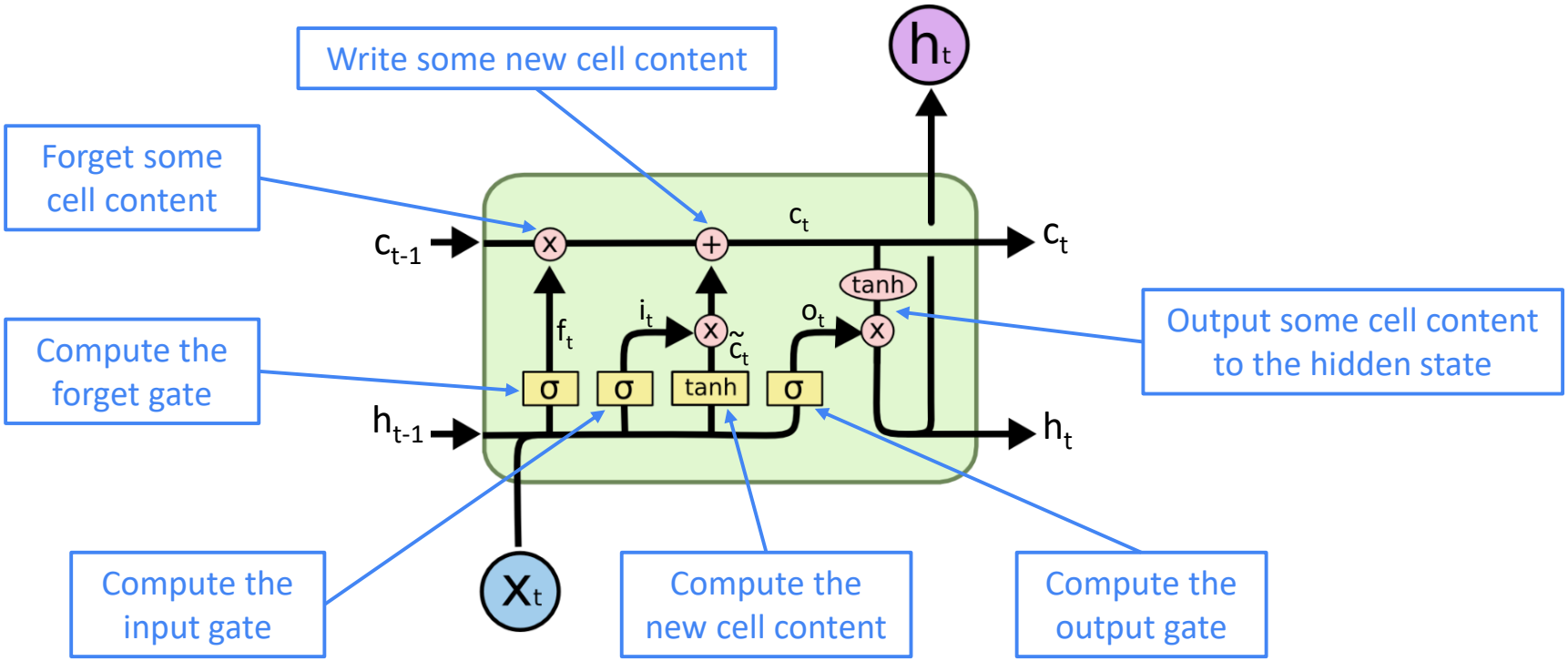
24

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

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Long Short-Term Memory (LSTM)

You can think of the LSTM equations visually like this:



25

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

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

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 remember everything on every timestep, then the info in the 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

26

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

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 (2019), 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

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

27

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Content

Gated Recurrent Units (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)

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

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

LSTM vs GRU

- Researchers have proposed many gated RNN variants, but LSTM and GRU are the most widely-used
- The biggest difference is that GRU is quicker to compute and has fewer parameters
- There is no conclusive evidence that one consistently performs better than the other
- LSTM is a good default choice (especially if your data has particularly long dependencies, or you have lots of training data)
- Rule of thumb: start with LSTM, but switch to GRU if you want something more efficient

29

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Content

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)

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

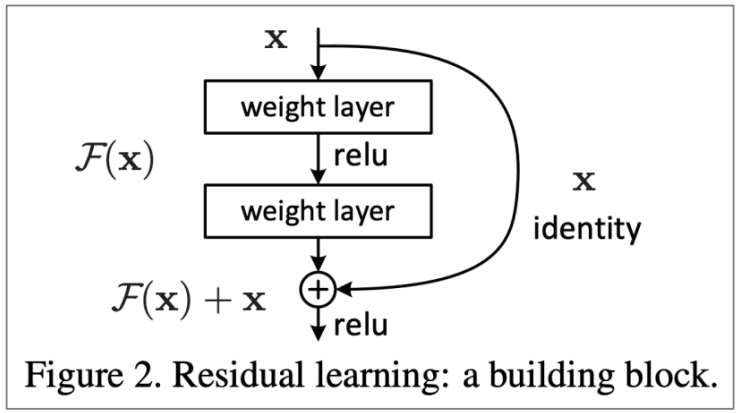


Figure 2. Residual learning: a building block.

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

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)

For example:

- Dense connections aka “DenseNet”
- Directly connect everything to everything!

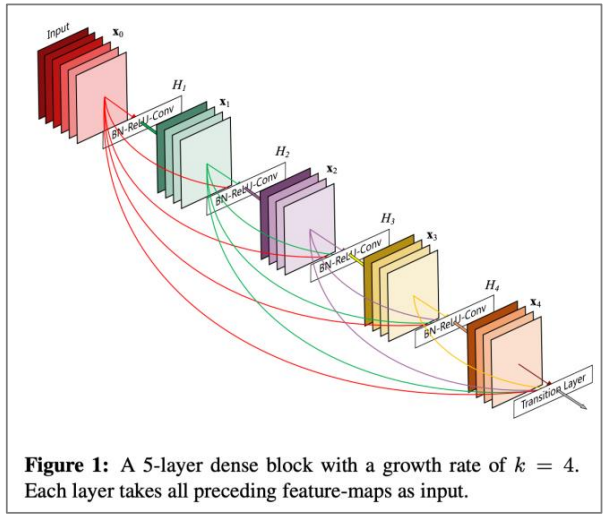


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

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

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

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)

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

32

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

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

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)

33

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

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]

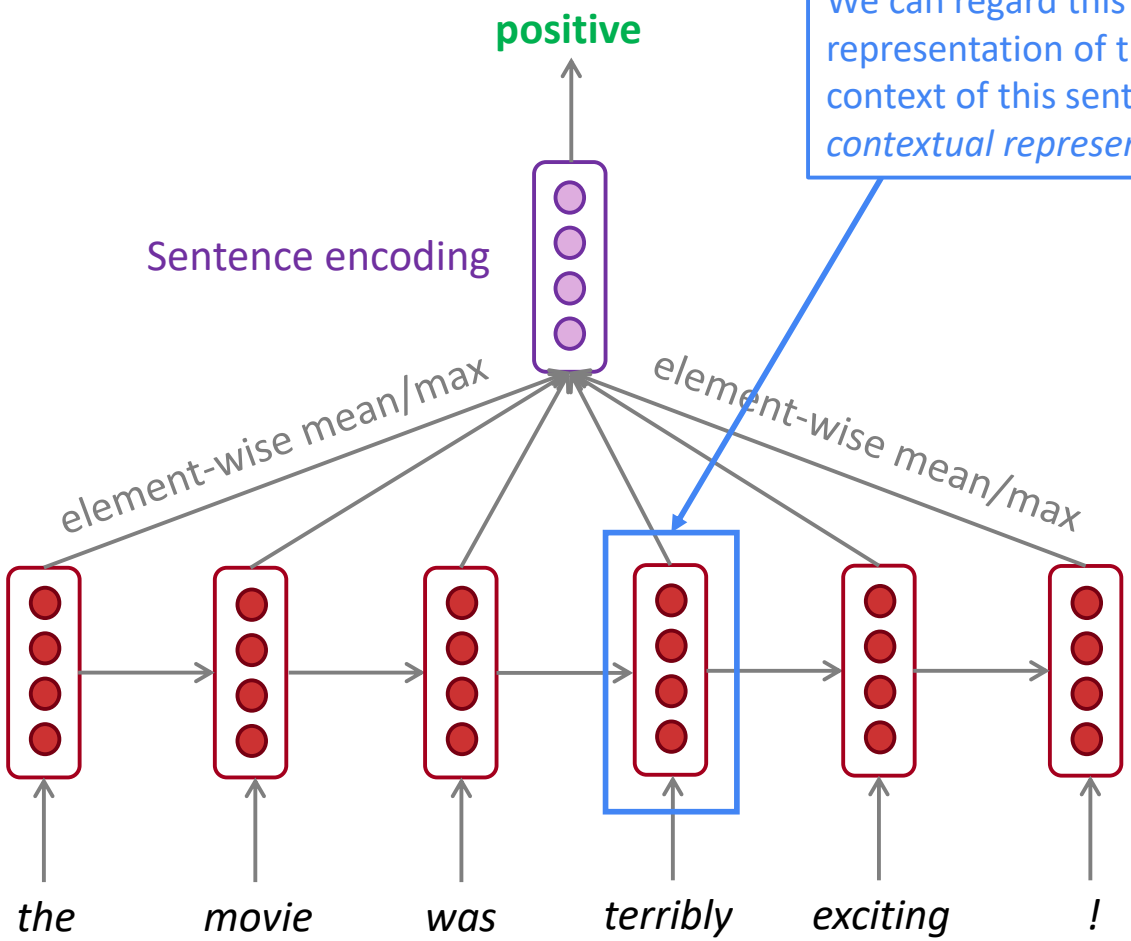
34 "Learning Long-Term Dependencies with Gradient Descent is Difficult", Bengio et al. 1994, http://ai.dinfo.unifi.it/paolo//ps/tnn-94-gradient.pdf

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Content

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.*

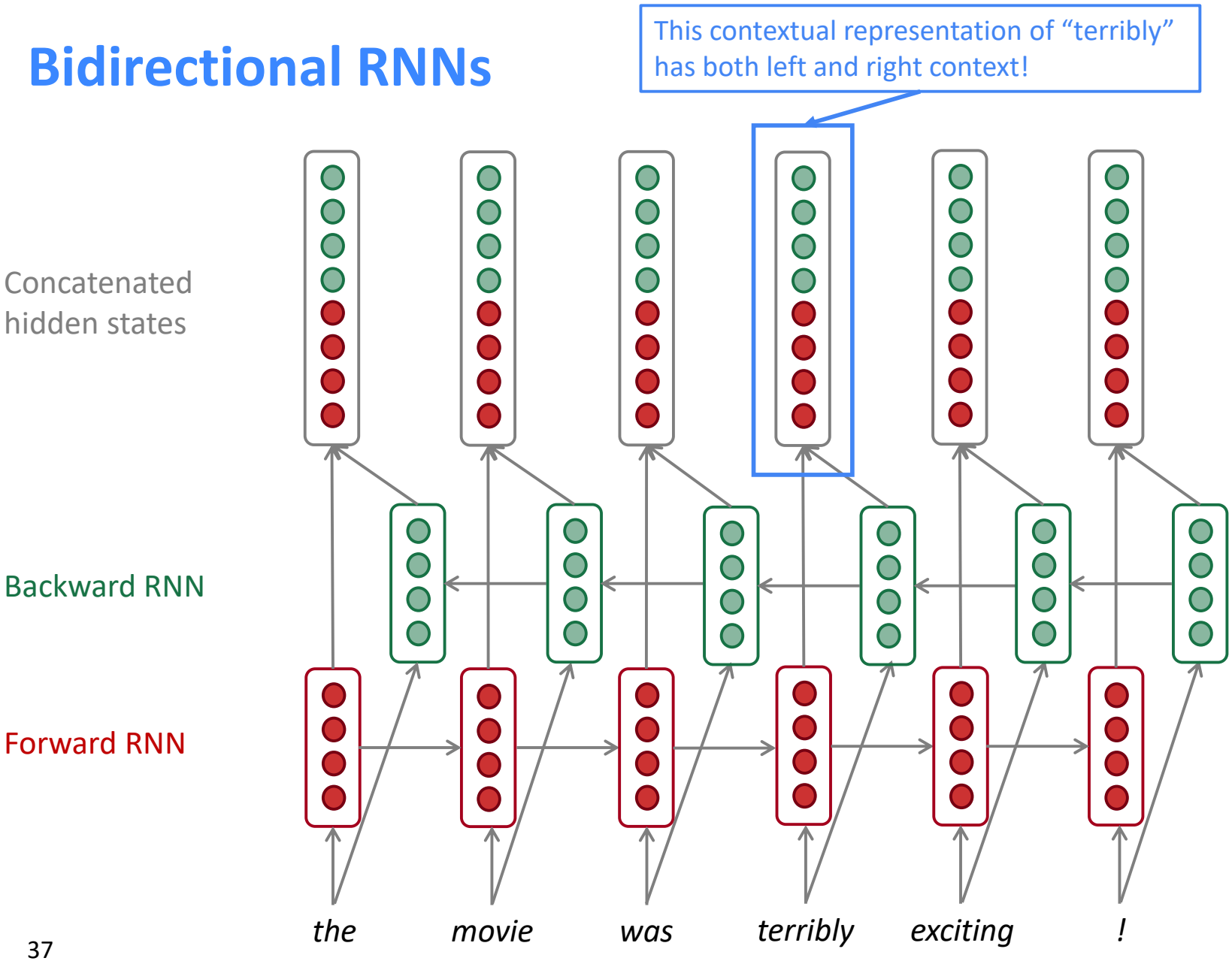
These contextual representations only contain information about the *left* context (e.g. *"the movie was"*).

What about *right* context?

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

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Bidirectional RNNs



Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

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

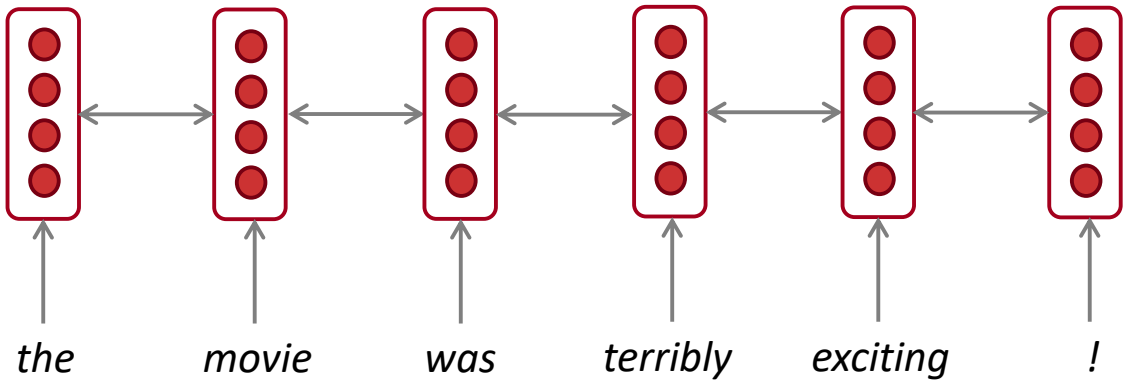
Concatenated hidden states $\boldsymbol{h}^{(t)} = [\overrightarrow{\boldsymbol{h}}^{(t)}; \overleftarrow{\boldsymbol{h}}^{(t)}]$

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

38

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Bidirectional RNNs: simplified diagram



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

39

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

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!

40

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Multi-layer RNNs

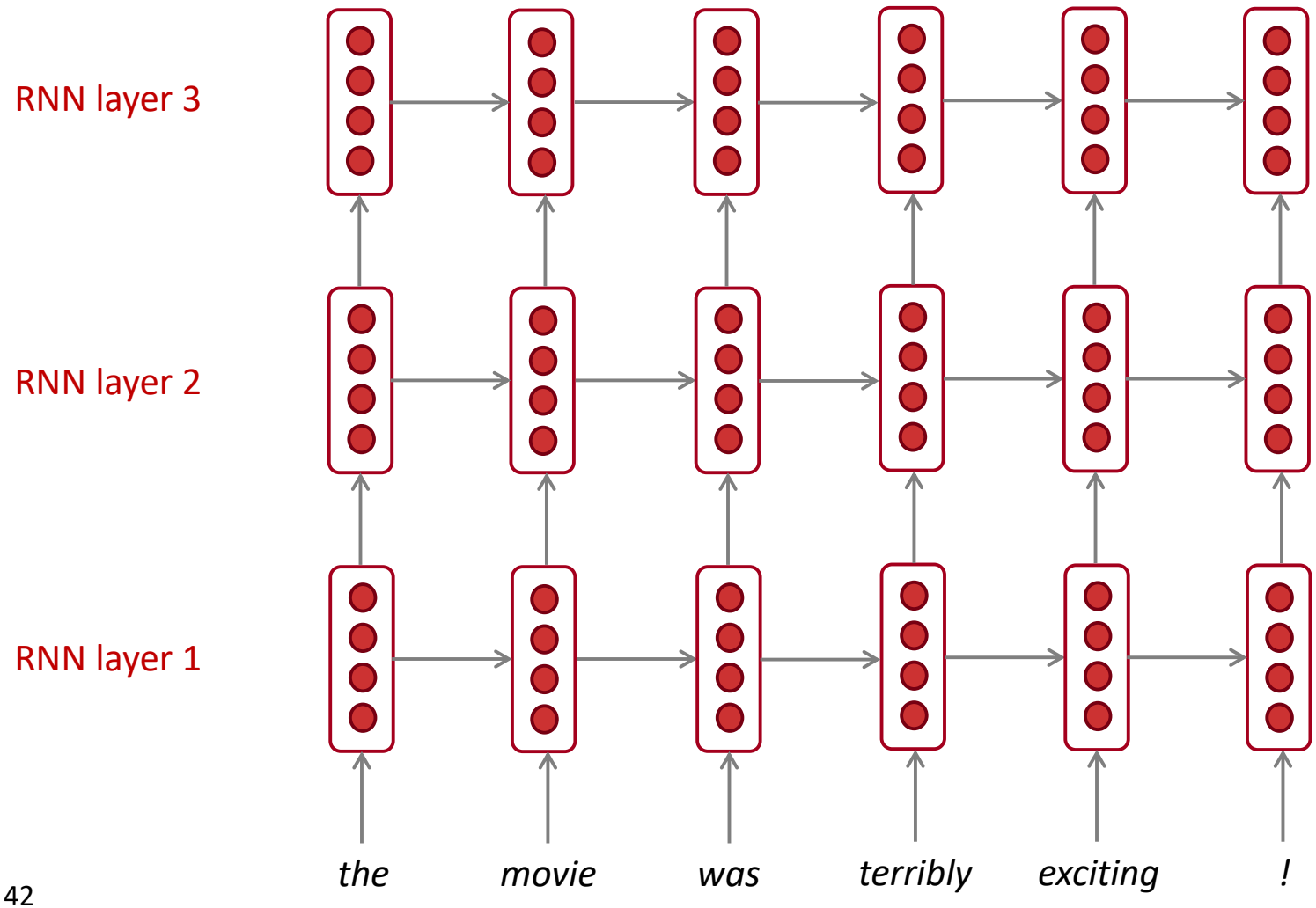
- RNNs are already "deep" on one dimension (they unroll over many timesteps)
- We can also make them "deep" in another dimension by applying multiple RNNs – this is a multi-layer RNN.
- This allows the network to compute more complex representations
  - The lower RNNs should compute lower-level features and the higher RNNs should compute higher-level features.
- Multi-layer RNNs are also called *stacked RNNs*.

41

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Multi-layer RNNs

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



42

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

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) can be up to 24 layers
  - You will learn about Transformers later; they have a lot of skipping-like connections

43

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

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

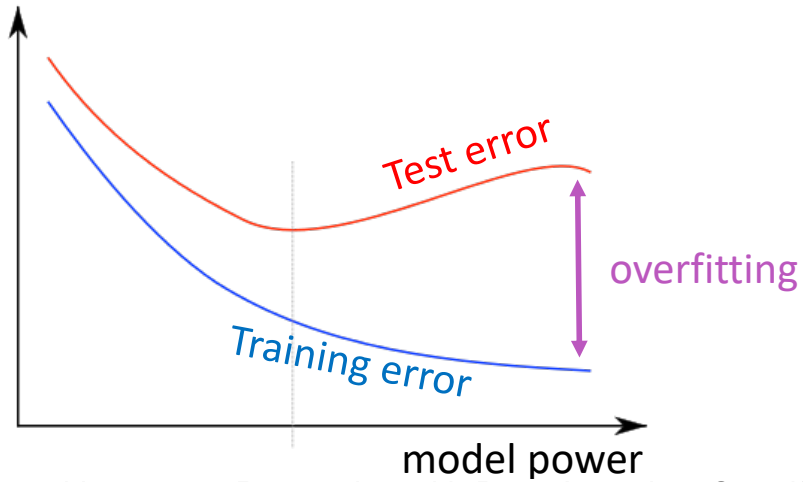
Content

We have models with many params! Regularization!

- Really a full loss function in practice includes **regularization** over all parameters θ, e.g., L2 regularization:

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} -\log\left(\frac{e^{f_{y_i}}}{\sum_{c=1}^{C} e^{f_c}}\right) + \lambda \sum_{k} \theta_k^2$$

- Regularization works to prevent overfitting when we have a lot of features (or later a very powerful/deep model, ++)



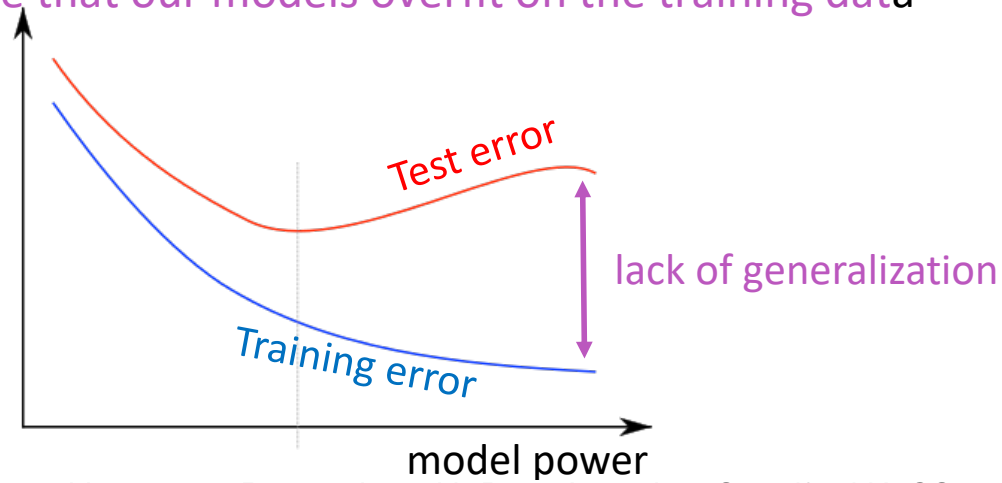
Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

We have models with many params! Regularization!

- Really a full loss function in practice includes **regularization** over all parameters θ, e.g., L2 regularization:

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} -\log\left(\frac{e^{f_{y_i}}}{\sum_{c=1}^{C} e^{f_c}}\right) + \lambda \sum_{k} \theta_k^2$$

- Regularization produces models that generalize well when we have a lot of features (or later a very powerful/deep model, ++)
  - We do not care that our models overfit on the training data



Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Dropout

(Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov 2012/JMLR 2014)

Preventing Feature Co-adaptation = Regularization

- Training time: at each instance of evaluation (in online SGD-training), randomly set 50% of the inputs to each neuron to 0
- Test time: halve the model weights (now twice as many)
- This prevents feature co-adaptation: A feature cannot only be useful in the presence of particular other features
- In a single layer: A kind of middle-ground between Naïve Bayes (where all feature weights are set independently) and logistic regression models (where weights are set in the context of all others)
- Can be thought of as a form of model bagging
- Nowadays usually thought of as strong, feature-dependent regularizer [Wager, Wang, & Liang 2013]

6

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

"Vectorization"

- E.g., looping over word vectors versus concatenating them all into one large matrix and then multiplying the softmax weights with that matrix

```
from numpy import random
N = 500 # number of windows to classify
d = 300 # dimensionality of each window
C = 5 # number of classes
W = random.rand(C,d)
wordvectors_list = [random.rand(d,1) for i in range(N)]
wordvectors_one_matrix = random.rand(d,N)

%timeit [W.dot(wordvectors_list[i]) for i in range(N)]
%timeit W.dot(wordvectors_one_matrix)
```

- 1000 loops, best of 3: **639 µs** per loop
  10000 loops, best of 3: **53.8 µs** per loop

7

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

"Vectorization"

```
from numpy import random
N = 500 # number of windows to classify
d = 300 # dimensionality of each window
C = 5 # number of classes
W = random.rand(C,d)
wordvectors_list = [random.rand(d,1) for i in range(N)]
wordvectors_one_matrix = random.rand(d,N)

%timeit [W.dot(wordvectors_list[i]) for i in range(N)]
%timeit W.dot(wordvectors_one_matrix)
```

- The (10x) faster method is using a C x N matrix
- Always try to use vectors and matrices rather than for loops!
- You should speed-test your code a lot too!!
- These differences go from 1 to 2 orders of magnitude with GPUs
- tl;dr: Matrices are awesome!!!

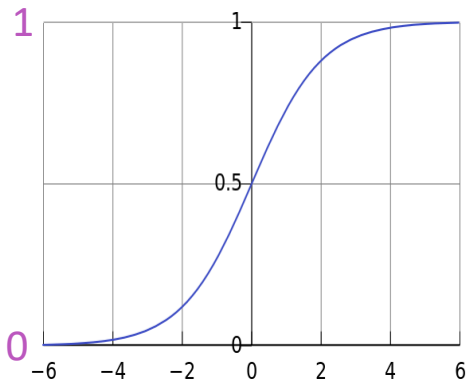
8

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Non-linearities: The starting points

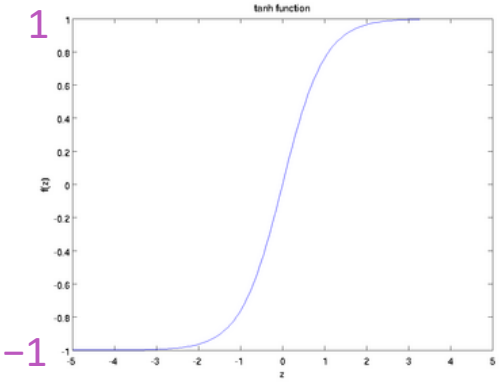
logistic ("sigmoid")

$$f(z) = \frac{1}{1 + \exp(-z)}.$$



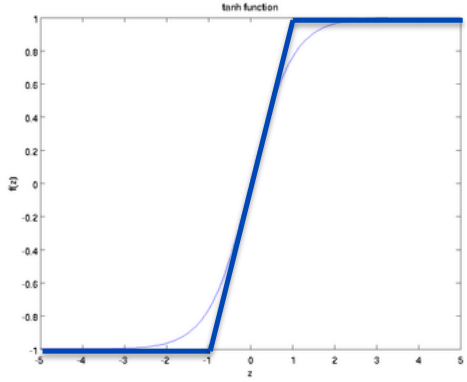
tanh

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}},$$



hard tanh

$$\text{HardTanh}(x) = \begin{cases} -1 & \text{if } x < -1 \\ x & \text{if } -1 <= x <= 1 \\ 1 & \text{if } x > 1 \end{cases}$$



tanh is just a rescaled and shifted sigmoid (2 × as steep, [−1,1]):

$$\tanh(z) = 2\text{logistic}(2z) - 1$$

Both logistic and tanh are still used in particular uses, but are no longer the defaults for making deep networks

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Non-linearities: The new world order

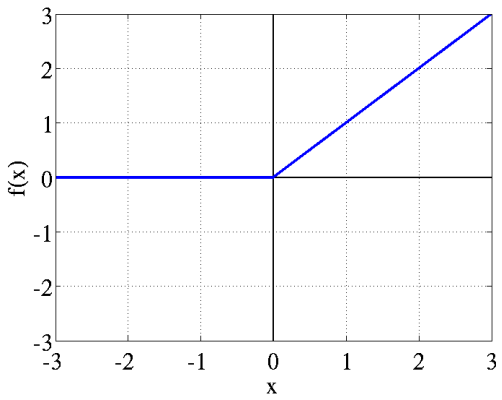
ReLU (rectified linear unit) hard tanh

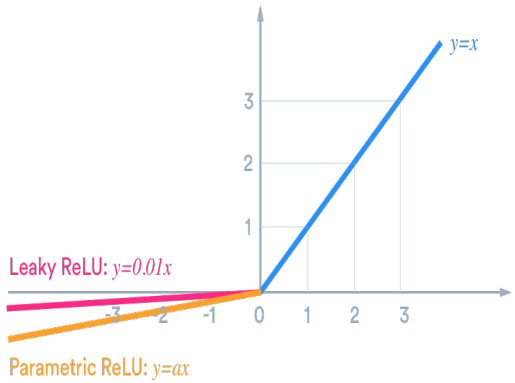
Leaky ReLU / Parametric ReLU

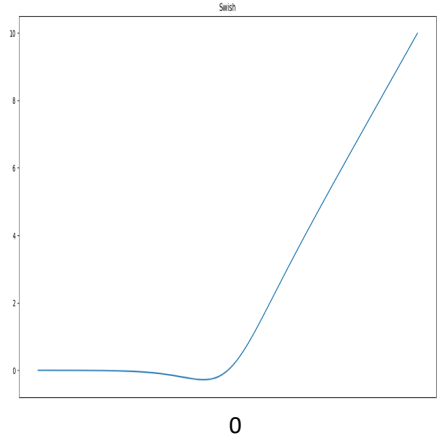
Swish

[Ramachandran, Zoph & Le 2017]

$$\text{rect}(z) = \max(z, 0)$$







- For building a deep feed-forward network, the first thing you should try is ReLU — it trains quickly and performs well due to good gradient backflow

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Parameter Initialization

- You normally must initialize weights to small random values
  - To avoid symmetries that prevent learning/specialization
- Initialize hidden layer biases to 0 and output (or reconstruction) biases to optimal value if weights were 0 (e.g., mean target or inverse sigmoid of mean target)
- Initialize **all other weights** ~ Uniform(–*r*, *r*), with *r* chosen so numbers get neither too big or too small
- Xavier initialization has variance inversely proportional to fan-in n_{in} (previous layer size) and fan-out n_{out} (next layer size):

$$\mathrm{Var}(W_i) = \frac{2}{n_{\mathrm{in}} + n_{\mathrm{out}}}$$

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Optimizers

- Usually, plain SGD will work just fine
  - However, getting good results will often require hand-tuning the learning rate (next slide)
- For more complex nets and situations, or just to avoid worry, you often do better with one of a family of more sophisticated "adaptive" optimizers that scale the parameter adjustment by an accumulated gradient.
  - These models give differentiak per-parameter learning rates
    - Adagrad
    - RMSprop
    - Adam ← A fairly good, safe place to begin in many cases
    - SparseAdam
    - …

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Learning Rates

- You can just use a constant learning rate. Start around *lr* = 0.001?
  - It must be order of magnitude right – try powers of 10
    - Too big: model may diverge or not converge
    - Too small: your model may not have trained by the deadline
- Better results can generally be obtained by allowing learning rates to decrease as you train
  - By hand: halve the learning rate every *k* epochs
    - An epoch = a pass through the data (shuffled or sampled)
  - By a formula: $lr = lr_0 e^{-kt}$, for epoch t
  - There are fancier methods like cyclic learning rates (q.v.)
- Fancier optimizers still use a learning rate but it may be an initial rate that the optimizer shrinks – so may need to start high

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Content