

Natural Language Processing with Deep Learning

CS224N/Ling284



Diyi Yang / Tatsunori Hashimoto

Lecture 1: Introduction and Word Vectors

Representing words as discrete symbols

In traditional NLP, we regard words as discrete symbols:

hotel, conference, motel – a **localist** representation

Means one 1, the rest 0s

Such symbols for words can be represented by **one-hot** vectors:

motel = [0 0 0 0 0 0 0 0 0 1 0 0 0 0]

hotel = [0 0 0 0 0 0 1 0 0 0 0 0 0 0]

Vector dimension = number of words in vocabulary (e.g., 500,000+)

19

Problem with words as discrete symbols

Example: in web search, if a user searches for “Seattle motel”, we would like to match documents containing “Seattle hotel”

But:

motel = [0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 1 0 0 0 0 0 0 0]

These two vectors are **orthogonal**

There is no natural notion of **similarity** for one-hot vectors!

Solution:

- Could try to rely on WordNet’s list of synonyms to get similarity?
 - But it is well-known to fail badly: incompleteness, etc.
- **Instead: learn to encode similarity in the vectors themselves**

20



Representing words by their context

- **Distributional semantics:** A word’s meaning is given by the words that frequently appear close-by
 - *You shall know a word by the company it keeps* (J. R. Firth 1957: 11)
 - One of the most successful ideas of modern statistical NLP!
- When a word w appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).
- We use the many contexts of w to build up a representation of w

...government debt problems turning into **banking** crises as happened in 2009...
...saying that Europe needs unified **banking** regulation to replace the hodgepodge...
...India has just given its **banking** system a shot in the arm...

21

These **context words** will represent **banking**

Word vectors

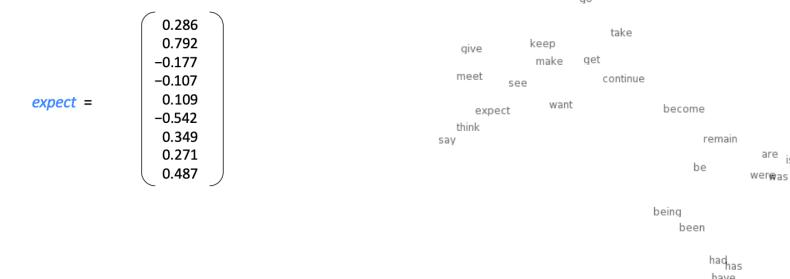
We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts, measuring similarity as the vector **dot** (scalar) **product**

$$\text{banking} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix} \quad \text{monetary} = \begin{pmatrix} 0.413 \\ 0.582 \\ -0.007 \\ 0.247 \\ 0.216 \\ -0.718 \\ 0.147 \\ 0.051 \end{pmatrix}$$

Note: word vectors are also called (word) embeddings or (neural) word representations
They are a **distributed** representation

22

Word meaning as a neural word vector – visualization



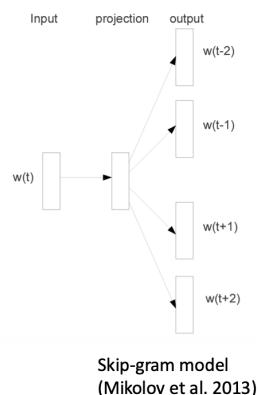
23

3. Word2vec: Overview

Word2vec is a framework for learning word vectors
(Mikolov et al. 2013)

Idea:

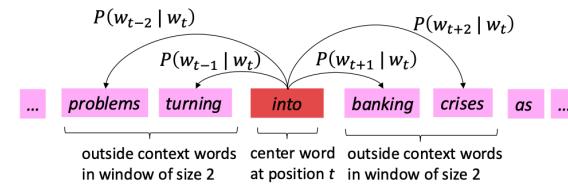
- We have a large corpus ("body") of text: a long list of words
- Every word in a fixed vocabulary is represented by a **vector**
- Go through each position t in the text, which has a center word c and context ("outside") words o
- Use the **similarity of the word vectors** for c and o to **calculate the probability** of o given c (or vice versa)
- **Keep adjusting the word vectors** to maximize this probability



24

Word2Vec Overview

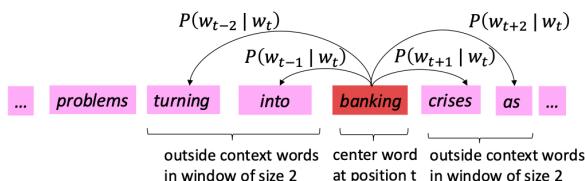
Example windows and process for computing $P(w_{t+j} | w_t)$



25

Word2Vec Overview

Example windows and process for computing $P(w_{t+j} | w_t)$



26

Word2Vec: objective function

For each position $t = 1, \dots, T$, predict context words within a window of fixed size m , given center word w_t . Data likelihood:

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^T \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j} | w_t; \theta)$$

θ is all variables to be optimized

sometimes called a *cost* or *loss* function

The **objective function** $J(\theta)$ is the **(average) negative log likelihood**:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} | w_t; \theta)$$

Minimizing objective function \Leftrightarrow Maximizing predictive accuracy

27

Word2Vec: objective function

- We want to minimize the objective function:

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} | w_t; \theta)$$

- Question:** How to calculate $P(w_{t+j} | w_t; \theta)$?

- Answer:** We will use two vectors per word w :

- v_w when w is a center word
- u_w when w is a context word

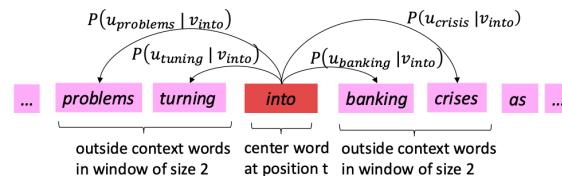
- Then for a center word c and a context word o :

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

28

Word2Vec with Vectors

- Example windows and process for computing $P(w_{t+j} | w_t)$
- $P(u_{problems} | v_{into})$ short for $P(problems | into ; u_{problems}, v_{into}, \theta)$



29

Word2Vec: prediction function

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

② Exponentiation makes anything positive
 ① Dot product compares similarity of o and c .
 $u^T v = u \cdot v = \sum_{i=1}^n u_i v_i$
 Larger dot product = larger probability
 ③ Normalize over entire vocabulary to give probability distribution

- This is an example of the **softmax function** $\mathbb{R}^n \rightarrow (0,1)^n$
 $\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^n \exp(x_j)} = p_i$
- The softmax function maps arbitrary values x_i to a probability distribution p_i
 - "max" because amplifies probability of largest x_i
 - "soft" because still assigns some probability to smaller x_i
- Frequently used in Deep Learning

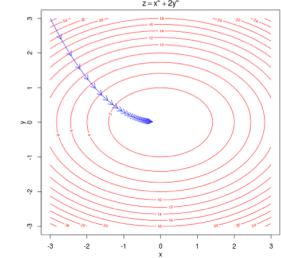
30

To train the model: Optimize value of parameters to minimize loss

To train a model, we gradually adjust parameters to minimize a loss

- Recall: θ represents **all** the model parameters, in one long vector
- In our case, with d -dimensional vectors and V -many words, we have $\theta \in \mathbb{R}^{2dV}$
- Remember: every word has two vectors

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$



- We optimize these parameters by walking down the gradient (see right figure)
- We compute **all** vector gradients!

31

Natural Language Processing with Deep Learning

CS224N/Ling284

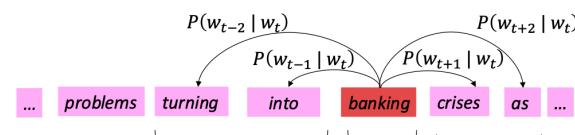


Diyi Yang

Lecture 2: Word Vectors, Word Senses, and Neural Classifiers

2. Review: Main idea of word2vec

- Start with random word vectors
- Iterate through each word position in the whole corpus
- Try to predict surrounding words using word vectors: $P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$



- Learning:** Update vectors so they can predict actual surrounding words better
- Doing no more than this, this algorithm learns word vectors that capture well word similarity and meaningful directions in a word space!



5

Word2vec parameters

$$\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \quad \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$$

U
outside V
center

... and computations

$$\begin{array}{ll} \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} & \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} \\ U \cdot v_4^T & \text{softmax}(U \cdot v_4^T) \\ \text{dot product} & \text{probabilities} \end{array}$$

"Bag of words" model!

The model makes the same predictions at each position

We want a model that gives a reasonably high probability estimate to *all* words that occur in the context (at all often)

9

Word2vec algorithm family (Mikolov et al. 2013): More details

Why two vectors? → Easier optimization. Average both at the end

- But can implement the algorithm with just one vector per word ... and it helps a bit

Two model variants:

1. Skip-grams (SG)

Predict context ("outside") words (position independent) given center word

2. Continuous Bag of Words (CBOW)

Predict center word from (bag of) context words

We presented: Skip-gram model

Loss functions for training:

1. Naïve softmax (simple but expensive loss function, when many output classes)
2. More optimized variants like hierarchical softmax
3. Negative sampling

So far, we explained naïve softmax

11

The skip-gram model with negative sampling

- The normalization term is computationally expensive (when many output classes):
- $$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$
 ← A big sum over words
- Hence, standard word2vec implements the skip-gram model with **negative sampling**
- Main idea: train binary logistic regressions to differentiate a true pair (center word and a word in its context window) versus several "noise" pairs (the center word paired with a random word)

12

4. How to evaluate word vectors?

- Related to general evaluation in NLP: Intrinsic vs. extrinsic
- Intrinsic:
 - Evaluation on a specific/intermediate subtask
 - Fast to compute
 - Helps to understand that system
 - Not clear if it's helpful unless correlation to real task is established
- Extrinsic:
 - Evaluation on a real task
 - Can take a long time to compute accuracy
 - Unclear if the subsystem is the problem or its interaction or other subsystems
 - If replacing exactly one subsystem with another improves accuracy → Winning!

24

6. Deep Learning Classification: Named Entity Recognition (NER)

- The task: find and classify names in text, by labeling word tokens, for example:

Last night , Paris Hilton wowed in a sequin gown .

PER PER

Samuel Quinn was arrested in the Hilton Hotel in Paris in April 1989 .

PER PER LOC LOC LOC DATE DATE

- Possible uses:
 - Tracking mentions of particular entities in documents
 - For question answering, answers are usually named entities
 - Relating sentiment analysis to the entity under discussion
- Often followed by Entity Linking/Canonicalization into a Knowledge Base such as Wikidata

34

Simple NER: Window classification using binary logistic classifier

- Idea: classify each word in its context window of neighboring words
- Train logistic classifier on hand-labeled data to classify center word {yes/no} for each class based on a concatenation of word vectors in a window
 - Really, we usually use multi-class softmax, but we're trying to keep it simple ☺
- Example: Classify "Paris" as +/– location in context of sentence with window length 2:

the museums in Paris are amazing to see .

$$X_{\text{window}} = [x_{\text{museums}} \ x_{\text{in}} \ x_{\text{Paris}} \ x_{\text{are}} \ x_{\text{amazing}}]^T$$

- Resulting vector $x_{\text{window}} = x \in \mathbb{R}^{5d}$
- To classify all words: run classifier for each class on the vector centered on each word in the sentence

35

NER: Binary classification for center word being location

- We do supervised training and want high score if it's a location

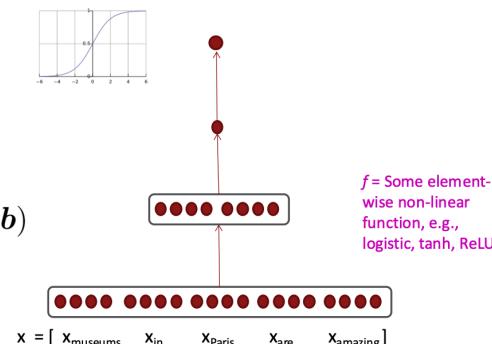
$$J_t(\theta) = \sigma(s) = \frac{1}{1 + e^{-s}}$$

predicted model probability of class

$$s = \mathbf{u}^T \mathbf{h}$$

$$\mathbf{h} = f(\mathbf{W}\mathbf{x} + \mathbf{b})$$

\mathbf{x} (input)



39

Natural Language Processing with Deep Learning CS224N/Ling284

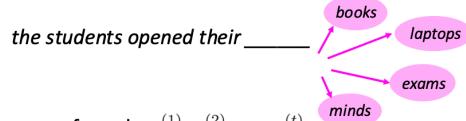


Tatsunori Hashimoto

Lecture 5: Language Models and Recurrent Neural Networks

1. Language Modeling

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



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

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

where $x^{(t+1)}$ can be any word in the vocabulary $V = \{w_1, \dots, w_{|V|}\}$

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

3

Language Modeling

- You can also think of a Language Model as a system that assigns a probability to a piece of text
- For example, if we have some text $x^{(1)}, \dots, x^{(T)}$, then the probability of this text (according to the Language Model) is:

$$\begin{aligned} P(x^{(1)}, \dots, x^{(T)}) &= P(x^{(1)}) \times P(x^{(2)} | x^{(1)}) \times \dots \times P(x^{(T)} | x^{(T-1)}, \dots, x^{(1)}) \\ &= \prod_{t=1}^T P(x^{(t)} | x^{(t-1)}, \dots, x^{(1)}) \end{aligned}$$

This is what our LM provides

4

Why should we care about Language Modeling?

- Language Modeling is a **benchmark task** that helps us measure our progress on predicting language use
- Language Modeling is a **subcomponent** of many NLP tasks, especially those involving **generating text** or **estimating the probability of text**:
 - Predictive typing
 - Speech recognition
 - Handwriting recognition
 - Spelling/grammar correction
 - Authorship identification
 - Machine translation
 - Summarization
 - Dialogue
 - etc.
- Everything else in NLP has been rebuilt upon Language Modeling: ChatGPT is an LM!

7

What can you do with next-word prediction?

*A sufficiently strong (!) language model can do many, many things

Stanford University is located in _____ California. [Trivia]

I put ____ fork down on the table. [syntax]

The woman walked across the street, checking for traffic over ____ shoulder. [coreference]

I went to the ocean to see the fish, turtles, seals, and _____. [lexical semantics/topic]

Overall, the value I got from the two hours watching it was the sum total of the popcorn and the drink. The movie was _____. [sentiment]

Iroh went into the kitchen to make some tea. Standing next to Iroh, Zuko pondered his destiny. Zuko left the _____. [some reasoning – this is harder]

I was thinking about the sequence that goes 1, 1, 2, 3, 5, 8, 13, 21, _____. [some basic arithmetic]

n-gram Language Models

the students opened their _____

- **Question:** How to learn a Language Model?
- **Answer (pre- Deep Learning):** learn an *n*-gram Language Model!
- **Definition:** An *n*-gram is a chunk of *n* consecutive words.
 - unigrams: "the", "students", "opened", "their"
 - bigrams: "the students", "students opened", "opened their"
 - trigrams: "the students opened", "students opened their"
 - four-grams: "the students opened their"
- **Idea:** Collect statistics about how frequent different n-grams are and use these to predict next word.

11

n-gram Language Models

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

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

$$\begin{aligned} \text{prob of a n-gram} &\rightarrow P(x^{(t+1)}, x^{(t)}, \dots, x^{(t-n+2)}) \\ \text{prob of a (n-1)-gram} &\rightarrow P(x^{(t)}, \dots, x^{(t-n+2)}) \end{aligned} \quad (\text{definition of conditional prob})$$

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

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

12

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

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

For example, suppose that in the corpus:

- "students opened their" occurred 1000 times
 - "students opened their books" occurred 400 times
 - $\rightarrow P(\text{books} | \text{students opened their}) = 0.4$
 - "students opened their exams" occurred 100 times
 - $\rightarrow P(\text{exams} | \text{students opened their}) = 0.1$
- Should we have discarded the "proctor" context?

13

Sparsity Problems with n-gram Language Models

Sparsity Problem 1

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

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

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

Sparsity Problem 2

Problem: What if "students opened their" never occurred in data? Then we can't calculate probability for any *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.

14

Storage Problems with n-gram Language Models

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

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

Increasing n or increasing corpus increases model size!

15

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*

today the _____

Business and financial news

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

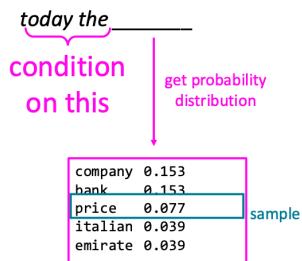
Otherwise, seems reasonable!

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

16

Generating text with a n-gram Language Model

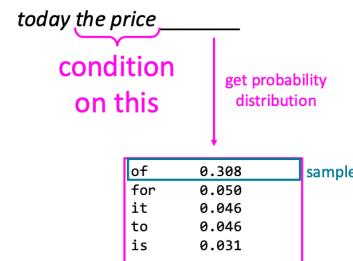
You can also use a Language Model to generate text



17

Generating text with a n-gram Language Model

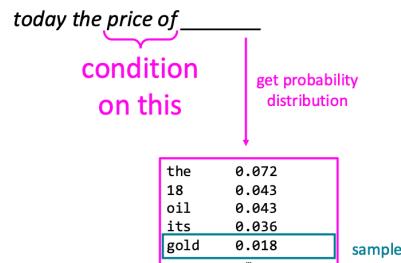
You can also use a Language Model to generate text



18

Generating text with a n-gram Language Model

You can also use a Language Model to generate text



19

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...

20

Evaluating Language Models

- The standard **evaluation metric** for Language Models is **perplexity**.

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

Inverse probability of corpus, according to Language Model

Normalized by number of words

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

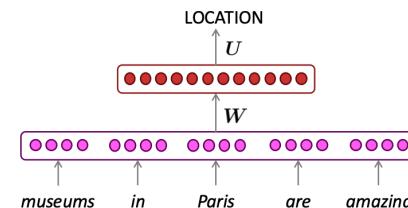
$$= \prod_{t=1}^T \left(\frac{1}{\hat{y}_{\mathbf{x}^{t+1}}^{(t)}} \right)^{1/T} = \exp \left(\frac{1}{T} \sum_{t=1}^T -\log \hat{y}_{\mathbf{x}^{t+1}}^{(t)} \right) = \exp(J(\theta))$$

Lower perplexity is better!

21

How to build a *neural* language model?

- Recall the Language Modeling task:
 - Input: sequence of words $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(t)}$
 - Output: prob. dist. of the next word $P(\mathbf{x}^{(t+1)} | \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(1)})$
- How about a **window-based neural model**?
 - We saw this applied to Named Entity Recognition in Lecture 2:



22

A fixed-window neural Language Model

as the proctor started the clock
 discard the students opened their _____
 fixed window

23

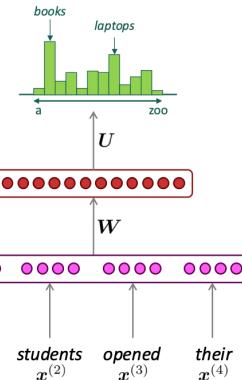
A fixed-window neural Language Model

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

hidden layer
 $h = f(We + b_1)$

concatenated word embeddings
 $e = [e^{(1)}, e^{(2)}; e^{(3)}; e^{(4)}]$

words / one-hot vectors
 $x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}$



24

A fixed-window neural Language Model

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

Improvements over n-gram LM:

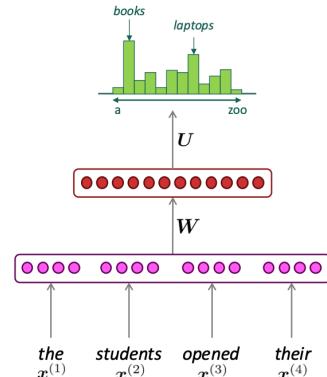
- No sparsity problem
- Don't need to store all observed n-grams

Remaining problems:

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

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

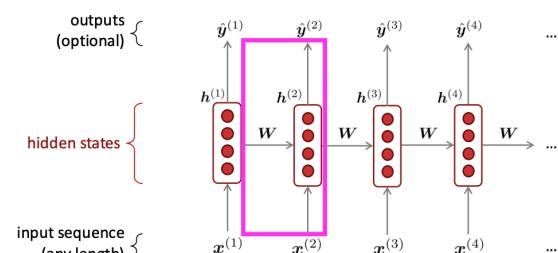
25



2. Recurrent Neural Networks (RNN)

A family of neural architectures

Core idea: Apply the same weights W repeatedly

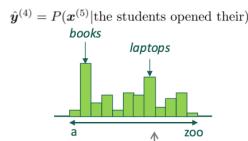


26

A Simple RNN Language Model

output distribution

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



hidden states

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

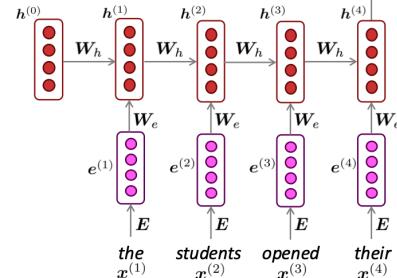
$h^{(0)}$ is the initial hidden state

word embeddings

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

words / one-hot vectors

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



Note: this input sequence could be much longer now!

27

RNN Language Models

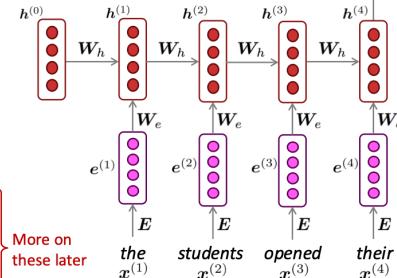
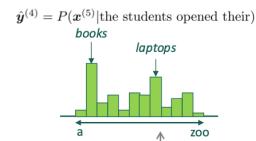
RNN Advantages:

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

RNN Disadvantages:

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

More on these later



28

Training an RNN Language Model

- Get a **big corpus of text** which is a sequence of words $x^{(1)}, \dots, x^{(T)}$
- Feed into RNN-LM; compute output distribution $\hat{y}^{(t)}$ 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 $\hat{y}^{(t)}$, and the true next word $y^{(t)}$ (one-hot for $x^{(t+1)}$):

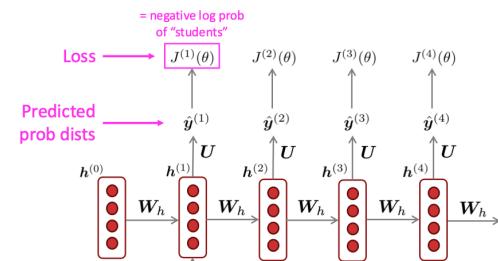
$$J^{(t)}(\theta) = CE(y^{(t)}, \hat{y}^{(t)}) = - \sum_{w \in V} y_w^{(t)} \log \hat{y}_w^{(t)} = -\log \hat{y}_{x_{t+1}}^{(t)}$$

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

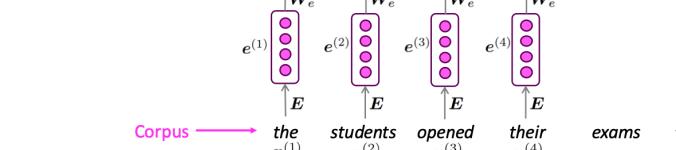
$$J(\theta) = \frac{1}{T} \sum_{t=1}^T J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^T -\log \hat{y}_{x_{t+1}}^{(t)}$$

29

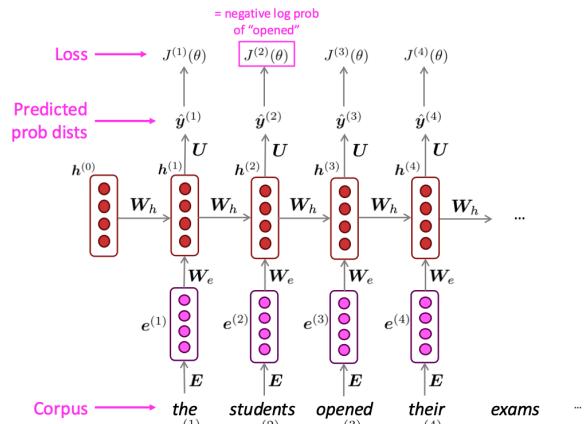
Training an RNN Language Model



30

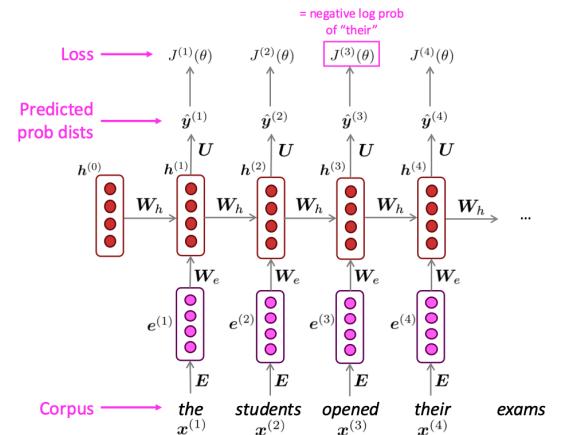


Training an RNN Language Model



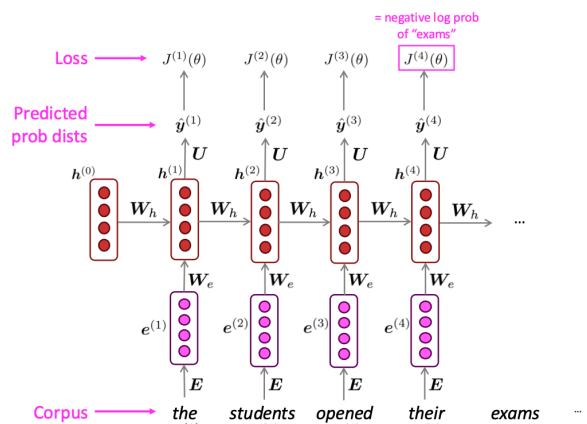
31

Training an RNN Language Model



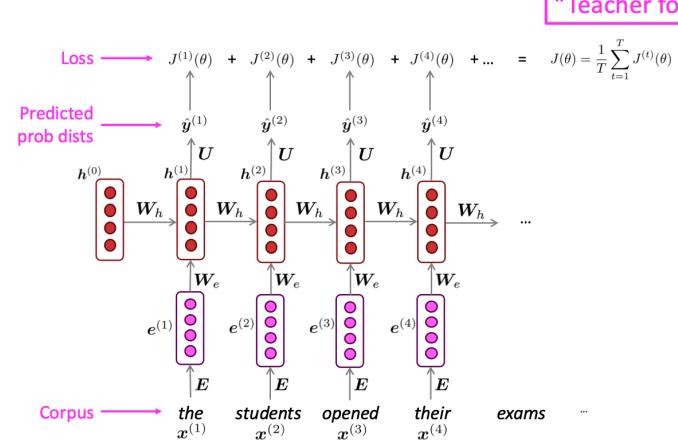
32

Training an RNN Language Model



33

Training an RNN Language Model



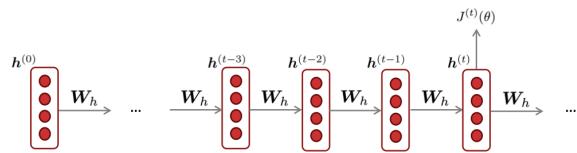
34

Training a RNN Language Model

- However: Computing loss and gradients across entire corpus $x^{(1)}, \dots, x^{(T)}$ at once is too expensive (memory-wise)!
- In practice, consider $x^{(1)}, \dots, x^{(T)}$ as a sentence (or a document)
- Recall: Stochastic Gradient Descent allows us to compute loss and gradients for small chunk of data, and update.
- Compute loss $J(\theta)$ for a sentence (actually, a batch of sentences), compute gradients and update weights. Repeat on a new batch of sentences.

35

Backpropagation for RNNs



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

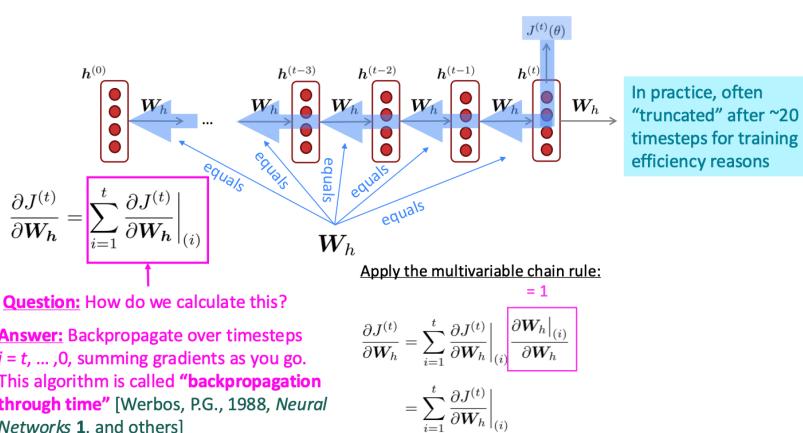
$$\text{Answer: } \frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1}^t \frac{\partial J^{(t)}}{\partial W_h} \Big|_{(i)}$$

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

Why?

36

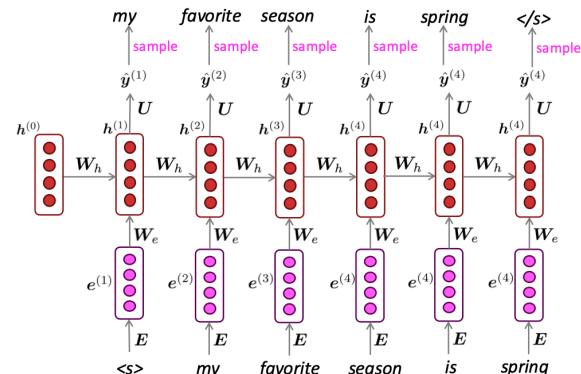
Training the parameters of RNNs: Backpropagation for RNNs



38

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

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



39

Generating text with an RNN Language Model

Let's have some fun!

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



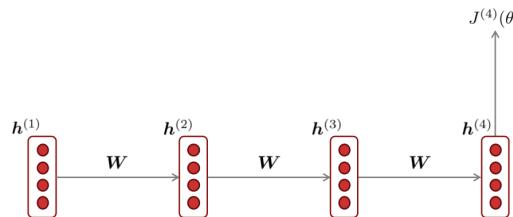
"Sorry," Harry shouted, panicking—"I'll leave those brooms in London, are they?"

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

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

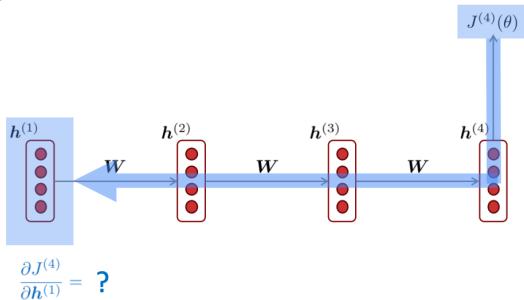
41

3. Problems with RNNs: Vanishing and Exploding Gradients



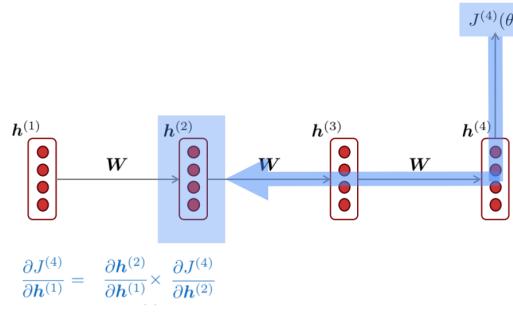
44

Vanishing gradient intuition



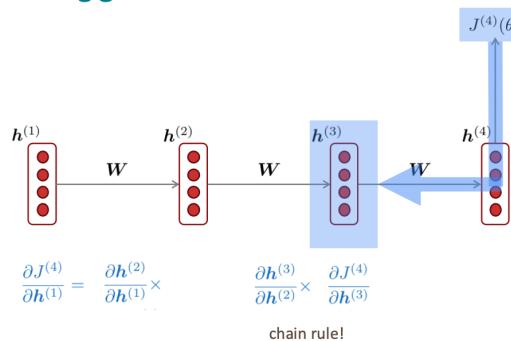
45

Vanishing gradient intuition



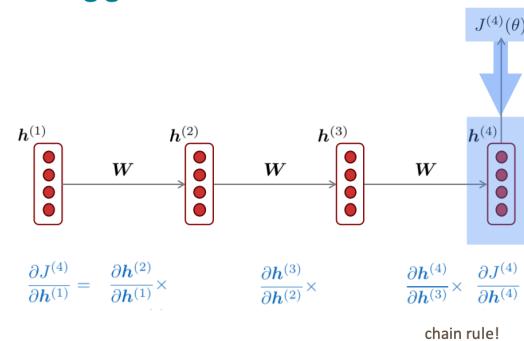
46

Vanishing gradient intuition



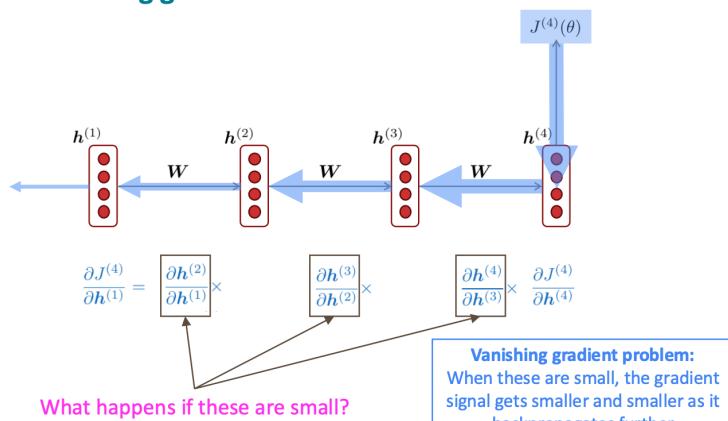
47

Vanishing gradient intuition



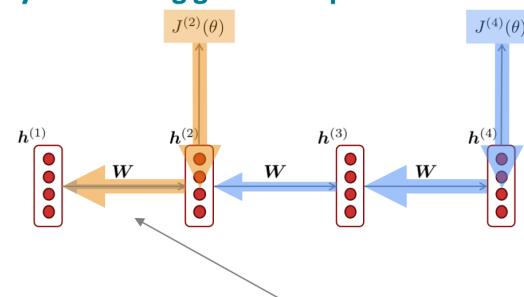
48

Vanishing gradient intuition



49

Why is vanishing gradient a problem?



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

52

Effect of vanishing gradient on RNN-LM

- LM task: When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her _____
- To learn from this training example, the RNN-LM needs to model the dependency between "tickets" on the 7th step and the target word "tickets" at the end.
- But if the gradient is small, the model can't learn this dependency
 - So, the model is unable to predict similar long-distance dependencies at test time

53

Why is exploding gradient a problem?

- If the gradient becomes too big, then the SGD update step becomes too big:

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

- This can cause bad updates: we take too large a step and reach a weird and bad parameter configuration (with large loss)
 - You think you've found a hill to climb, but suddenly you're in Iowa
- In the worst case, this will result in Inf or NaN in your network (then you have to restart training from an earlier checkpoint)

54

Gradient clipping: solution for exploding gradient

- Gradient clipping: if the norm of the gradient is greater than some threshold, scale it down before applying SGD update

Algorithm 1 Pseudo-code for norm clipping

```
 $\hat{g} \leftarrow \frac{\partial E}{\partial \theta}$ 
if  $\|\hat{g}\| \geq \text{threshold}$  then
     $\hat{g} \leftarrow \frac{\text{threshold}}{\|\hat{g}\|} \hat{g}$ 
end if
```

- Intuition: take a step in the same direction, but a smaller step
- In practice, remembering to clip gradients is important, but exploding gradients are an easy problem to solve

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

55

How to fix the vanishing gradient problem?

- The main problem is that it's too difficult for the RNN to learn to preserve information over many timesteps.
- In a vanilla RNN, the hidden state is constantly being rewritten
$$h^{(t)} = \sigma(W_h h^{(t-1)} + W_x x^{(t)} + b)$$
- First off next time: How about an RNN with separate memory which is added to?
 - LSTMs
- And then: Creating more direct and linear pass-through connections in model
 - Attention, residual connections, etc.

56