

# Deep Learning for Natural Language Processing

Cornelia Caragea

Computer Science  
University of Illinois at Chicago

Credits for slides: Manning, Socher, See

Language Models and Recurrent Neural Networks

# Today

## Lecture Structure (Part 1):

- Homework 1 is out
- Final Projects Discussion
- Language Models - NLP task
- Recurrent Neural Networks - a family of neural networks

## Lecture Structure (Part 2):

- Training an RNN Language Model
- Recurrent Neural Network  $\neq$  Language Model
- Problems with RNNs - the vanishing gradient problem

# Final Project

- Start early and clearly define your task and dataset
- Project types:
  - Apply existing neural network model to a new task
  - Implement a complex neural architecture
  - Come up with a new neural network model

# Class Project: Apply Existing NNets to Tasks

- Define Task:
  - Example: Summarization
- Define Dataset
  - Search for academic datasets
    - They already have baselines
    - E.g.: Document Understanding Conference (DUC)
  - Define your own (harder, need more new baselines)
    - Connect to your research
    - Summarization, Wikipedia: Intro paragraph and rest of large article
    - Be creative: Twitter, Blogs, News

# Class Project: Apply Existing NNets to Tasks

- Define your metric
  - Search online for well established metrics on this task
  - Summarization: Rouge (Recall-Oriented Understudy for Gisting Evaluation) which defines n-gram overlap to human summaries
- Split your dataset!
  - Train/Dev/Test
  - Academic dataset often come pre-split
  - Don't look at the test split until 1 week before deadline!

# Class Project: Apply Existing NNets to Tasks

- Establish a baseline
  - Implement the simplest model (often logistic regression on unigrams and bigrams) first
  - Compute metrics on train AND dev
  - Analyze errors
  - If metrics are amazing and no errors: done, problem was too easy, restart :)
- Implement existing neural net model
  - Compute metric on train and dev
  - Analyze output and errors
  - Minimum bar for this class

# Class Project: Apply Existing NNets to Tasks

- Always be close to your data!
  - Visualize the dataset
  - Collect summary statistics
  - Look at errors
  - Analyze how different hyperparameters affect performance
- Try out different model variants - Soon you will have more options
  - Fixed window neural model
  - Recurrent neural network
  - Convolutional neural network

# Class Project: A New Model – Advanced Option

- Start early!
- Gain intuition of why existing models are flawed
- Set up efficient experimental framework
- Build simpler new models first
- Implement new models and iterate quickly over ideas

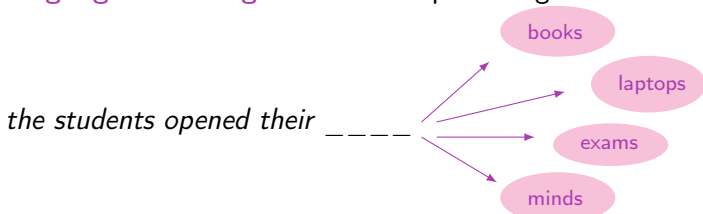


# Project Ideas

- Summarization
- Information extraction / NER
- Simple question answering
- Image to text mapping or generation
- Entity level sentiment
- Emotion detection

# 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  $x^{(t+1)}$ :

$$P(x^{(t+1)} | x^{(t)}, \dots, x^{(2)}, x^{(1)})$$

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

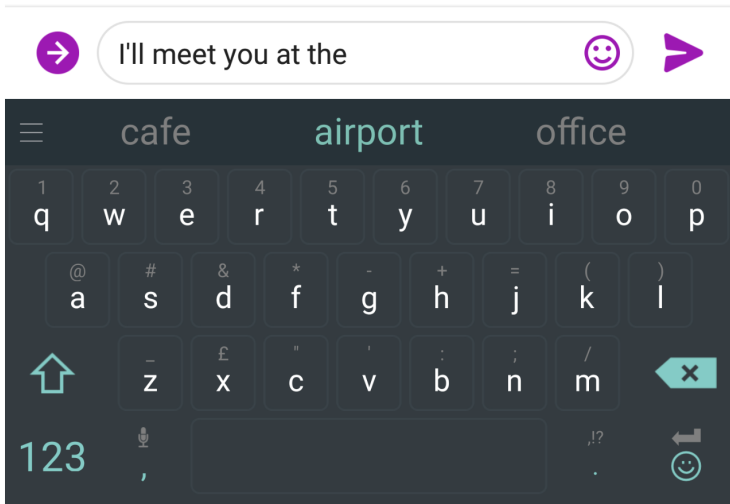
- A system that does this is called a **Language Model**.

# 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  $x^{(1)}, x^{(2)}, \dots, x^{(T)}$ , then the probability of this text (according to the Language Model) is:

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

# You use Language Models every day!



# You use Language Models every day!



what is the |



what is the **weather**

what is the **meaning of life**

what is the **dark web**

what is the **xfl**

what is the **doomsday clock**

what is the **weather today**

what is the **keto diet**

what is the **american dream**

what is the **speed of light**

what is the **bill of rights**

Google Search


I'm Feeling Lucky


# You use Language Models every day!


## Cut through the clutter


Find peer-reviewed research from the world's most trusted sources


All Fields


 keyword




 Keywan Riahi

 Khandan Keyomarsi

 K. Eric Wommack

 Key words for use in RFCs to Indicate Requirement Levels Bradner, 1997

 Keywords: a Vocabulary of Culture and Society Dawson, 1976

[See all results for "keywo"](#)

Supplement your research with Semantic Scholar

# 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”
  - *4-grams*: “the students opened their”
- Idea: Collect statistics about how frequent different  $n$ -grams are, and use these to predict next word.

# n-gram Language Models

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

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

$$= \frac{P(x^{(t+1)}, x^{(t)}, \dots, x^{(t-n+2)})}{P(x^{(t)}, \dots, x^{(t-n+2)})}$$

← prob of an  $n$ -gram

← prob of an  $(n-1)$ -gram

(definition of conditional prob)

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

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

(statistical approximation)



## n-gram Language Models: Example

Suppose we are learning a 4-gram Language Model.

*as the proctor started the clock, the students opened their \_\_\_\_\_*

*as the proctor started the clock, the* *students opened their* \_\_\_\_\_  
*discard* *condition on this*

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

For example, suppose that in the corpus:

- “students opened their” occurred 1,000 times
- “students opened their **books**” occurred 400 times
  - $\rightarrow P(\text{books} \mid \text{students opened their}) = 0.4$
- “students opened their **exams**” occurred 100 times
  - $\rightarrow P(\text{exams} \mid \text{students opened their}) = 0.1$

← Should we  
have discarded the  
“proctor” context?

# Sparsity Problems with n-gram LM

## Sparsity Problem 1

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

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

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

$\text{count}(\text{students opened their } w)$

$\text{count}(\text{students opened their})$

## Sparsity Problem 2

**Problem:** What if “students opened their” never occurred in data? Then we cannot 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 cannot have  $n$  bigger than 5.

# Storage Problems with $n$ -gram LM

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

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

Increasing  $n$  or increasing corpus increases model size!

# n-gram Language Models in practice

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

today the \_\_\_\_\_

get probability  
distribution

company	0.153
bank	0.153
price	0.077
italian	0.039
emirate	0.039

...

\*Try for yourself:

<https://nlpforhackers.io/language-models/>

# 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 \_\_\_\_\_

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 distri-  
bution

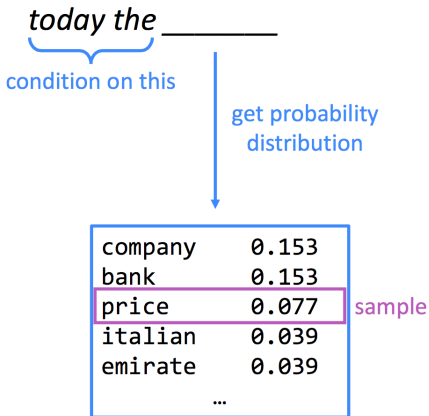
Otherwise, seems reasonable!

\*Try for yourself:

<https://nlpforhackers.io/language-models/>

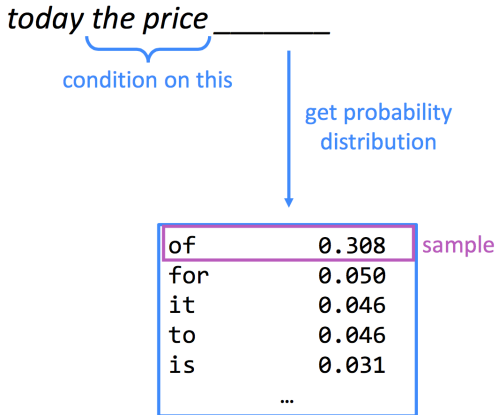
# Generating text with an n-gram Language Model

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



# Generating text with an n-gram Language Model

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



# Generating text with an n-gram Language Model

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

*today the price of* \_\_\_\_\_

condition on this

get probability  
distribution

the	0.072
18	0.043
oil	0.043
its	0.036
gold	0.018
...	

sample



# Generating text with an n-gram Language Model

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

*today the price of gold* \_ \_ \_ \_ \_

# Generating text with an 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 .*

# Generating text with an n-gram Language Model

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

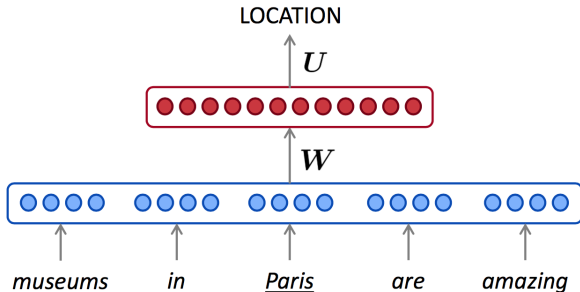
*today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share .*

- Surprisingly grammatical!.
- ...but incoherent. We need to consider more than three words at a time if we want to model language well.
- However, increasing  $n$  worsens sparsity problem, and increases model size...

Note: punctuation is treated like any other word.

# How to build a *neural* Language Model?

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



# A fixed-window neural Language Model

*as the proctor started the clock the students opened their \_\_\_\_\_*

# A fixed-window neural Language Model

~~as the proctor started the clock~~    *the students opened their* \_\_\_\_\_  
discard                                  fixed window

# A fixed-window neural Language Model

output distribution

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

hidden layer

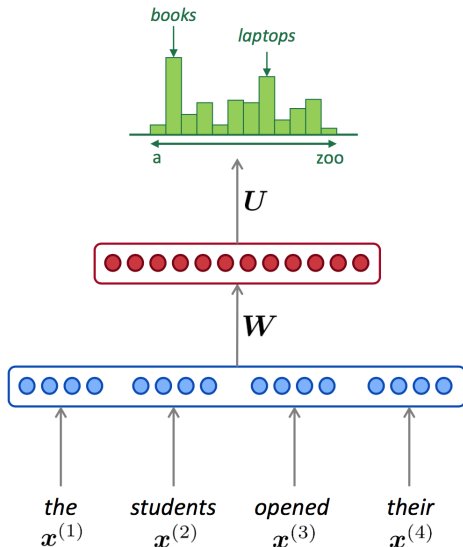
$$h = f(We + b_1)$$

concatenated word embeddings

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

words / one-hot vectors

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



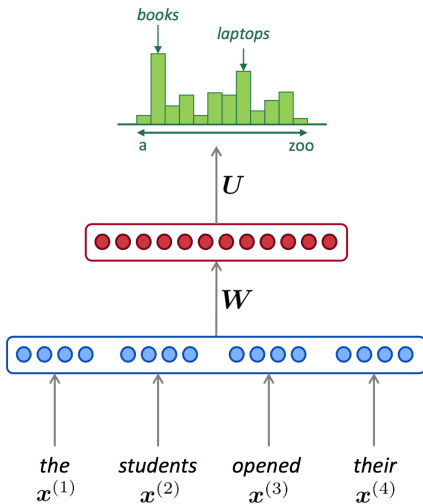
# A fixed-window neural Language Model

Improvements over  $n$ -gram LM:

- No sparsity problem.
- Do not 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.





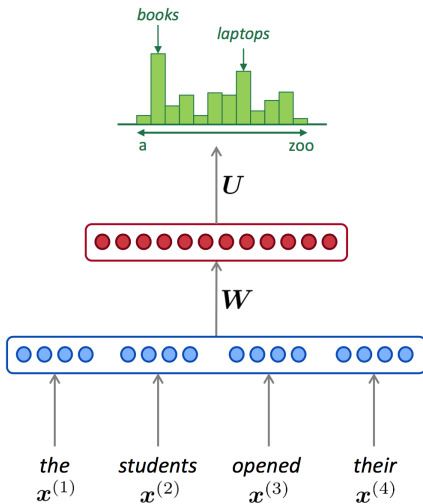
# A fixed-window neural Language Model

Improvements over  $n$ -gram LM:

- No sparsity problem.
- Do not need to store all observed  $n$ -grams.

Remaining **problems**:

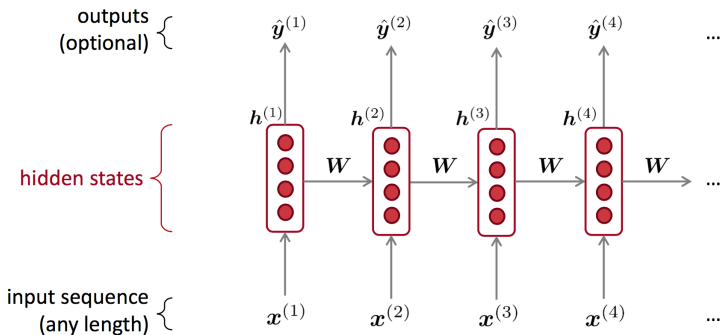
- Fixed window is **too small**.
- Enlarging window enlarges  $W$ .
- Window can never be large enough!
- $x^{(1)}$  and  $x^{(2)}$  are multiplied by completely different weights in  $W$ .  
**No symmetry** in how the inputs are processed.



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

# Recurrent Neural Networks (RNN)

A family of neural architectures



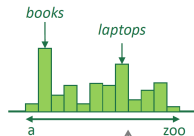
**Core idea:** Apply the same weights  $W$  repeatedly!

# An RNN Language Model

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

output distribution

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



hidden states

$$h^{(t)} = \sigma(W_h h^{(t-1)} + W_e e^{(t)} + b_1)$$

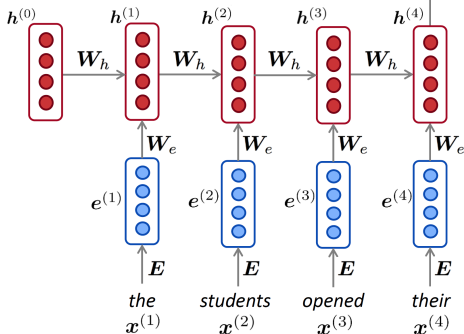
$h^{(0)}$  is the initial hidden state

word embeddings

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

words / one-hot vectors

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



Note: the input sequence could be much longer.

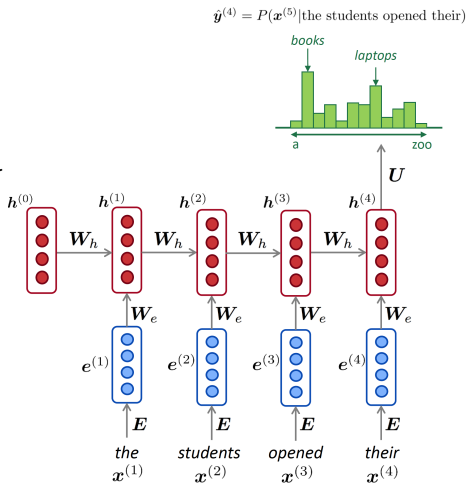
# An RNN Language Model

## RNN Advantages:

- Can process **any length** input
- Computation for step  $t$  can (in theory) use information from **many steps back**
- **Model size does not increase** for longer input
- Same weights applied on every time step, 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**.



# Next

- Training an RNN Language Model
- RNNs  $\neq$  LMs
- Problems with RNNs!
  - Vanishing gradients
  - Exploding gradients