



# CS 2731 Introduction to Natural Language Processing

## Session 11: N-gram language models part 2, RNNs part 1

---

Michael Miller Yoder

February 14, 2024



School of Computing and Information

# Course logistics

- [Homework 2](#) is **due tomorrow, Thu Feb 15 at 11:59pm**
  - There is a Canvas discussion forum for asking questions (feel free to offer answers, too)
  - Please put `_NN` or `_LR` after your team name in the Kaggle competition
  - Please provide Python scripts
    - If using Jupyter Notebooks, export to Python scripts
- Homework 1 grades are out
- Projects
  - Proposal and literature review is **due Thu Feb 22**
    - Instructions are on the [project webpage](#)
  - It's good to start the literature review early
  - Look for NLP papers in [ACL Anthology](#), [Semantic Scholar](#), and [Google Scholar](#)

# Midterm course evaluation (OMETs)

- CS 2731:  
<https://go.blueja.io/3ydJumSMRk23LZULkOgifQ>
- ISSP 2230:  
<https://go.blueja.io/U9GquEwBj0WkSuaDGWNx5g>
- All types of feedback are welcome  
(critical and positive)
- **Completely anonymous, will not affect grades**
- Let me know what's working and what to  
improve on while the course is still running!
- Please be as specific as possible
- Available until **Mon Feb 19**



# Lecture overview: N-gram language models part 2, RNNs part 1

- Sampling sentences from language models
- The problem of zeros
- Laplace smoothing
- Interpolation and backoff
- Neural language models
- RNN language modeling

# Sampling sentences from language models

---

# The Shannon Visualization Method

- Choose a random bigram ( $\langle s \rangle, w$ ) according to its probability
- Now choose a random bigram ( $w, x$ ) according to its probability
- And so on until we choose  $\langle /s \rangle$
- Then string the words together

```
<s> I
    I want
      want to
        to eat
          eat Chinese
            Chinese food
              food </s>

I want to eat Chinese food
```

# Approximating Shakespeare

1

gram

–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have

–Hill he late speaks; or! a more to leg less first you enter

2

gram

–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow.

–What means, sir. I confess she? then all sorts, he is trim, captain.

3

gram

–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.

–This shall forbid it should be branded, if renown made it empty.

4

gram

–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;

–It cannot be but so.



# The Wall Street Journal is not Shakespeare (no offense)

1  
gram

Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives

2  
gram

Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her

3  
gram

They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions



# The problem of zeros

---

# The Perils of Overfitting

N-grams only work well for word prediction if the test corpus looks like the training corpus

- In real life, it often doesn't
- We need to train robust models that generalize!
  - One kind of generalization: Zeros!
  - Things that don't ever occur in the training set but occur in the test set

# N-grams in the test set that weren't in the training set

Suppose our bigram LM, trained on Twitter, reads a document by the philosopher Wittgenstein:

*Whereof one cannot speak, thereof one must be silent.*

This contains the bigrams: whereof one, one cannot, cannot speak, speak [comma], [comma] thereof, thereof one, one must, must be, be silent.

Suppose “whereof one” never occurs in the training corpus (**train**) but whereof occurs 20 times. According to MLE, it's probability is

$$P(\text{one}|\text{whereof}) = \frac{c(\text{whereof, one})}{c(\text{whereof})} = \frac{0}{20} = 0$$

The probability of the sentence is the **product** of the probabilities of the bigrams. What happens if one of the probabilities is zero?

Compare, again, to Naïve Bayes.

# Two kinds of “zeros”

1. Completely unseen words in the test set
2. Words in unseen contexts in the test set

# Unknown Words

If we know all the words in advanced

- Vocabulary  $V$  is fixed
- Closed vocabulary task

Often we don't know this

- Out Of Vocabulary = OOV words
- Open vocabulary task

Instead: create an unknown word token <UNK>

- Training of <UNK> probabilities
- Create a fixed lexicon  $L$  of size  $V$
- At text normalization phase, any training word not in  $L$  changed to <UNK>
- Now we train its probabilities like a normal word
- At decoding time
- If text input: Use UNK probabilities for any word not in training

# Laplace smoothing

---

# The intuition of smoothing

When we have sparse statistics:

$P(w \mid \text{denied the})$

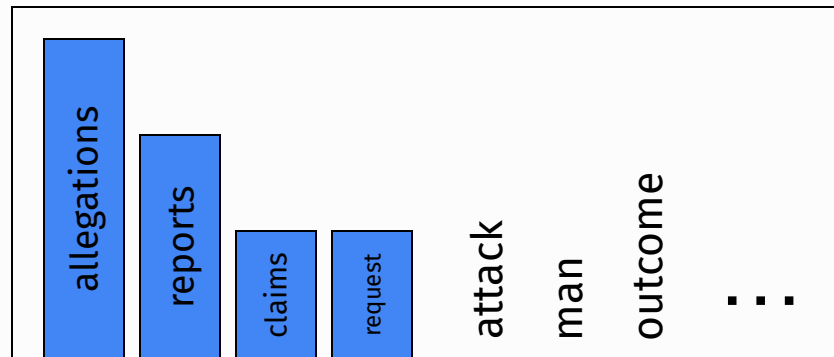
3 allegations

2 reports

1 claims

1 request

7 total



Steal probability mass to generalize better

$P(w \mid \text{denied the})$

2.5 allegations

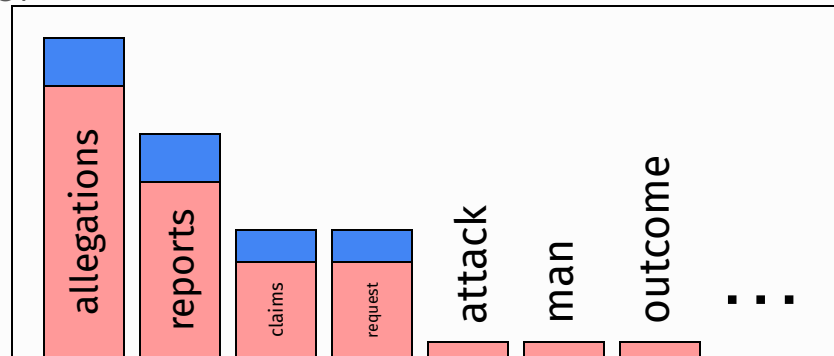
1.5 reports

0.5 claims

0.5 request

2 other

7 total





# Laplace smoothing: Pretending that we saw each word once more

This should already be familiar from Naive Bayes

$$\text{MLE estimate } P_{MLE}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

$$\text{Add-1 estimate } P_{Add-1}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + |V|}$$

Where  $V$  is the vocabulary of the corpus.

# Laplace Smooth Is too Blunt

It shifts too much probability mass away from attested ngrams and onto unattested ngrams, so it isn't used much for ngram LMs any more (there are better methods).

But remember that it does work:

- For text classification
- In domains where the number of zeros isn't as large as with ngrams

# Interpolation and backoff

---

# Backoff and Interpolation Let You Use Less Context

Suppose you have a context you don't know much about (because you have seen few or no relevant ngrams). You can condition your probabilities for these contexts on shorter contexts you know more about.

**Backoff** Use trigram if you have good evidence, otherwise bigram, otherwise unigram.

**Interpolation** Mix unigrams, bigrams, and trigrams together in one (weighted) probability soup.

Interpolation works better; backoff is sometimes cheaper.

# Linear interpolation takes into account different n-grams

The simplest way to do this is to **not** take context into account. The lambdas, in the following formula, are weighting factors:

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1 P(w_n|w_{n-2}w_{n-1}) \\ &\quad + \lambda_2 P(w_n|w_{n-1}) \\ &\quad + \lambda_3 P(w_n)\end{aligned}$$

where

$$\forall i \lambda_i \geq 0 \wedge \sum_i^n \lambda_i = 1$$

That is, the lambdas must sum to one.

# Lambdas Are Tuned Using a Held-Out dev Set



Choose  $\lambda$ s to maximize the probability of held-out data (**dev**):

- Fix the ngram probabilities (on **train**)
- Then search for  $\lambda$ s that give the largest probability to **dev**:

# Web-Scale Ngrams

How to deal with, e.g., Google N-gram corpus Pruning

- Only store N-grams with count  $>$  threshold.
- Remove singletons of higher-order n-grams
- Entropy-based pruning

Efficiency

- Efficient data structures like tries
- Bloom filters: approximate language models
- Store words as indexes, not strings
- Use Huffman coding to fit large numbers of words into two bytes
- Quantize probabilities (4-8 bits instead of 8-byte float)

# Stupid Backoff is Stupid but Efficient

No discounting, just use relative frequencies

$$S(w_i | w_{i-k+1}^{j-1}) = \begin{cases} \frac{c(w_{i-k+1}^j)}{c(w_{i-k+1}^{j-1})} & \text{if } c(w_{i-k+1}^j) > 0 \\ 0.4S(w_i | w_{i-k+2}^{j-1}) & \text{otherwise} \end{cases}$$

$$S(w_i) = \frac{c(w_i)}{N}$$



# Neural language models

---

# Why Neural LMs work better than N-gram LMs

Training data:

We've seen: I have to make sure that the cat gets fed.

Never seen: dog gets fed

Test data:

I forgot to make sure that the dog gets \_\_\_

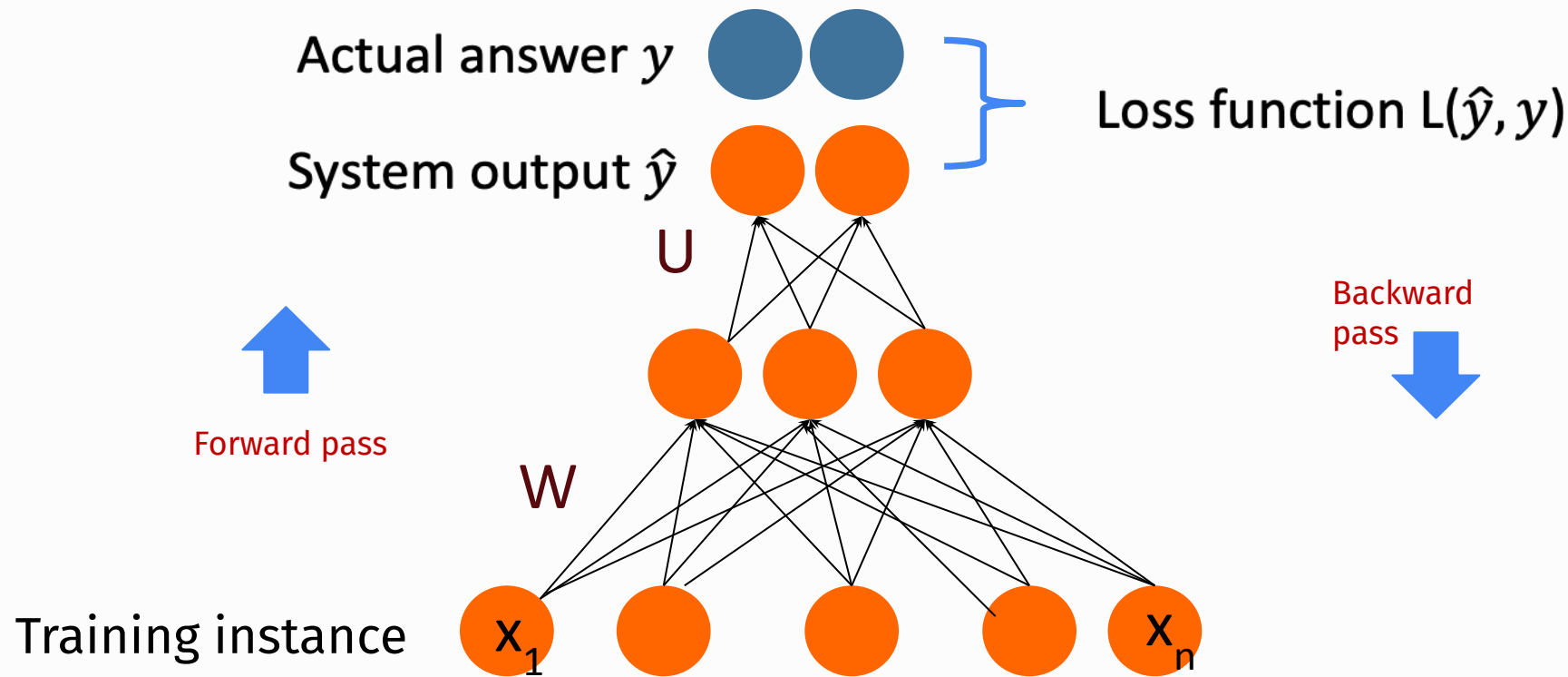
N-gram LM can't predict "fed"!

Neural LM can use similarity of "cat" and "dog" embeddings to generalize and predict "fed" after dog

# Language modeling with recurrent neural networks (RNNs)

---

# Reminder: training a 2-layer network



# There are Two Benefits to Training a Neural Language Model

If you train a neural language model, you get two things:

1. An algorithm that will allow you to predict the next word in a sequence
2. A set of embeddings  $\mathbf{E}$  that can be used to represent words in other tasks (assuming that you did not freeze the embedding layer)

**FFNNs take an input of fixed dimensions**—a fixed number of features, a fixed number of tokens

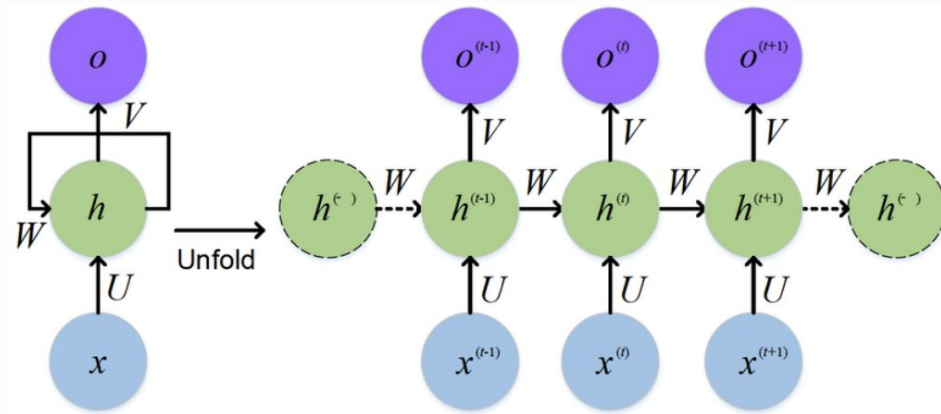
The number tokens in a text—even a sentence—can be **arbitrarily large** (or short)

RNNs help us address this issue



# The architecture of an RNN

- Special kind of multilayer neural network for modeling sequences
- Hidden layers between the input and output receive input not just from the input layer, **but also from the hidden layer at a preceding timestep**
- RNNs can “remember” information from earlier on



# An RNN Language Model

## output distribution

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

## 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

$$e^{(t)} = Ex^{(t)}$$

## one-hot vectors

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

