CS 2731 Introduction to Natural Language Processing

Session 11: N-gram language models, part 2

Michael Miller Yoder

October 4, 2023

University of Pittsburgh School of Computing and Information

Course logistics

- Homework 2 is **due this Thu 10-05, 11:59pm**
  - There is a Canvas discussion forum for asking questions (feel free to offer answers, too)
  - We will run hw2_{your pitt email id}_test.py test_data.csv (held-out test set)
    - This script should run both your logistic regression and neural network classifiers and print accuracy on the test dataset passed to it
    - This script can either load your trained model (which also needs to be submitted) or train in a reasonable amount of time with the politeness_data.csv assumed to be in the current working directory
    - Word embedding files are often very big, so if it’s >400 MB just point give us a URL, name, and version to download

Course logistics

- Projects
    - If you haven't yet met with Pantho or me about your project, please do!
    - Come to our office hours
        - Wed 1:30-2:30pm with Michael in Sennott Square 6505
        - Thu 2:45-3:45pm with Pantho in Sennott Square 5106
    - Proposal and literature review is **due Thu 10-12, 11:59pm**
        - Instructions are on the project webpage
    - Look for NLP papers in ACL Anthology, Semantic Scholar, and Google Scholar

Lecture overview: N-gram language models, part 2

- 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 (<s>, w) according to its probability
- Now choose a random bigram (w, x) according to its probability
- And so on until we choose </s>
- 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
```


Slide adapted from Jurafsky & Martin

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.

Slide adapted from Jurafsky & Martin

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

Slide adapted from Jurafsky & Martin

Can you guess the training set author of the LM that generated these random 3-gram sentences?

- 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 gram Brazil on market conditions
- This shall forbid it should be branded, if renown made it empty.
- “You are uniformly charming!” cried he, with a smile of associating and now and then I bowed and they perceived a chaise and four to wish for.

Slide adapted from Jurafsky & Martin

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

Slide credit: David Mortensen

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

$$P(\text{one}|\text{whereof}) = \frac{c(\text{whereof}, \text{one})}{c(\text{whereof})} = \frac{0}{20} = 0 \tag{19}$$

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

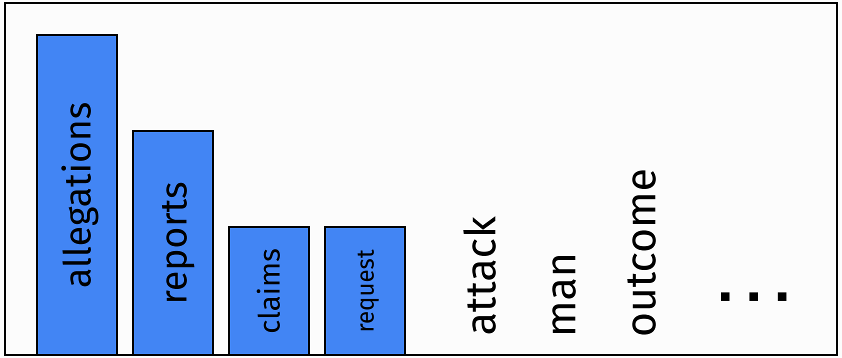
The intuition of smoothing

When we have sparse statistics:

P(w | denied the)

3 allegations
2 reports
1 claims
1 request

7 total

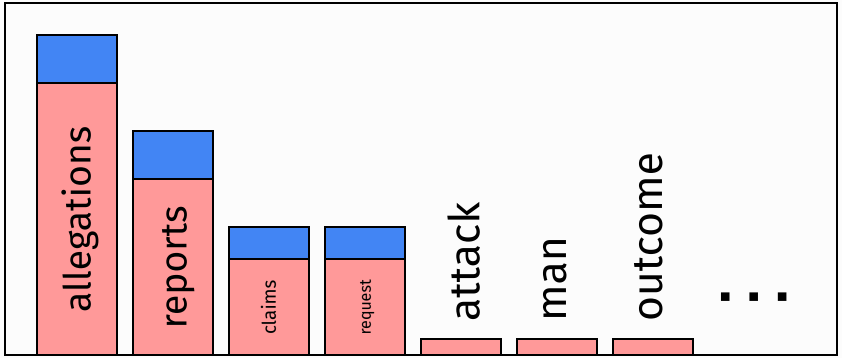


Steal probability mass to generalize better

P(w | denied the)

2.5 allegations
1.5 reports
0.5 claims
0.5 request
2 other

7 total



Slide adapted from Jurafsky & Martin, Dan Klein

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.

Slide credit: David Mortensen

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 is does work:

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

Slide credit: David Mortensen

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.

Slide credit: David Mortensen

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:

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

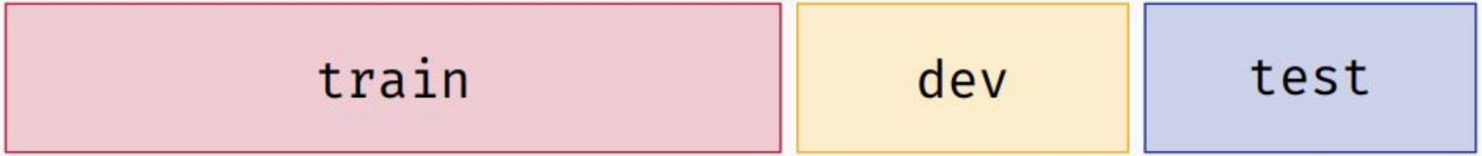
where

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

That is, the lambdas must sum to one.

Slide credit: David Mortensen

Lambdas Are Tuned Using a Held-Out `dev` Set



Choose λs to maximize the probability of held-out data (**`dev`**):

- Fix the ngram probabilities (on **`train`**)
- Then search for λs that give the largest probability to **`dev`**:

Slide credit: David Mortensen

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)

Slide credit: David Mortensen

Stupid Backoff is Stupid but Efficient

No discounting, just use relative frequencies

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

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

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 ___

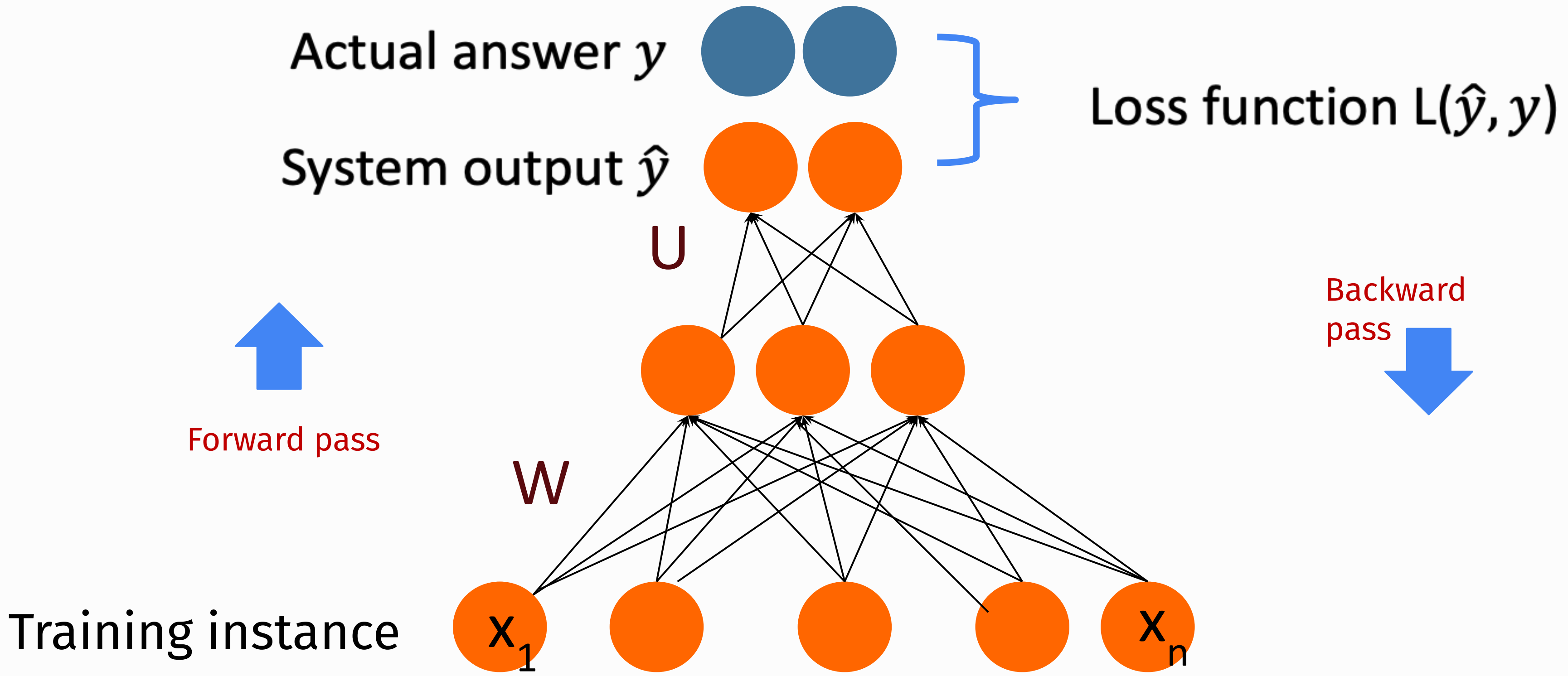
N-gram LM can't predict "fed"!

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

Slide adapted from Jurafsky & Martin

Language modeling with recurrent neural networks (RNNs)

Reminder: training a 2-layer network



Slide adapted from Jurafsky & Martin

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 **E** that can be used to represent words in other tasks (assuming that you did not freeze the embedding layer)

Slide credit: David Mortensen

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

Slide credit: David Mortensen

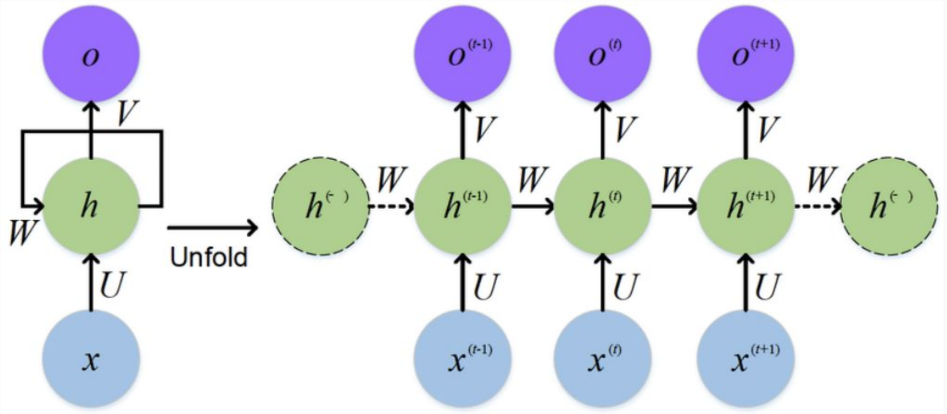
The number tokens in a text—even a sentence—can be **arbitrarily large** (or short)

Slide credit: David Mortensen

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 form the input layer, **but also from the hidden layer at a preceding timestep**
- RNNs can "remember" information from earlier on



Slide credit: David Mortensen

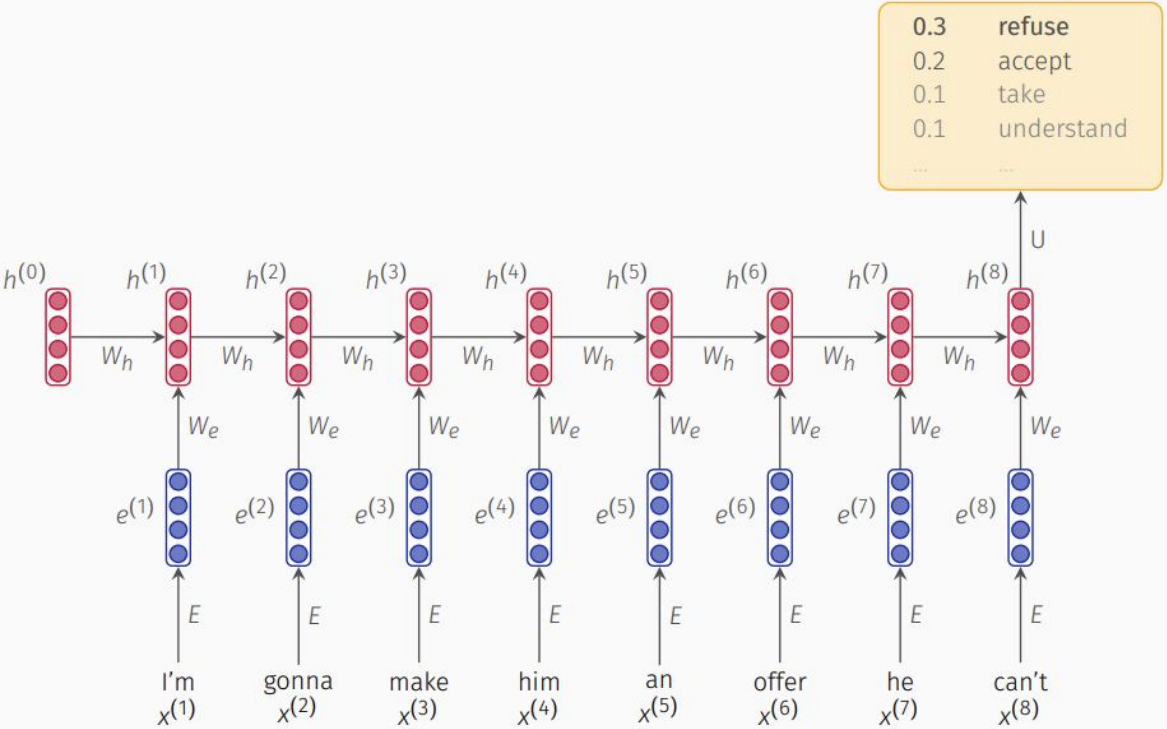
An RNN Language Model

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



Slide credit: David Mortensen

Training an RNN Language Model

- Get a big corpus of text, which is a sequence of words $x^{(1)}, \ldots, x^{(T)}$
- Feed it into the RNN-LM, computing output distribution $^{(t)}$ for every step t.
- Loss function on step t is **cross-entropy** between the predicted probability distribution $\hat{\mathbf{y}}^{(t)}$ and the true next word $y^{(t)}$ (one-hot for $x^{(t+1)}$):

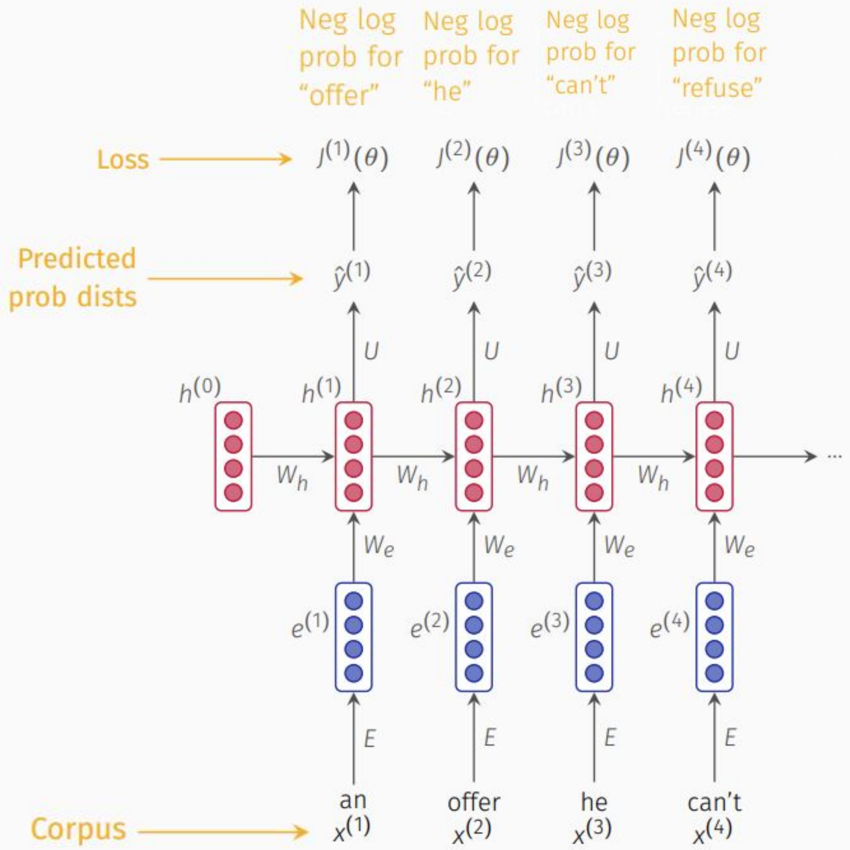
$$J^{(t)}(\theta) = CE(\mathbf{y}^{(t)}, \hat{\mathbf{y}}^{(t)}) = -\sum_{w \in V} \mathbf{y}_w^{(t)} \log \hat{\mathbf{y}}_w^{(t)} = -\log \hat{\mathbf{y}}_{\mathbf{x}_{t+1}}^{(t)}$$

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

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

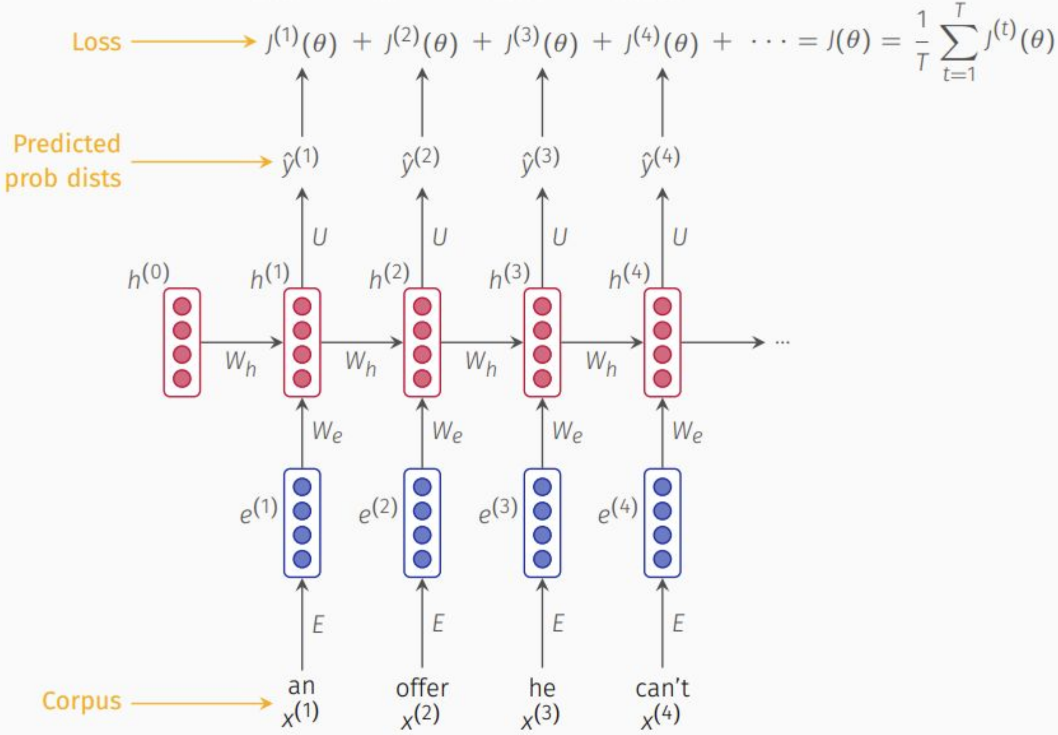
Slide credit: David Mortensen

Training an RNN Language Model



Slide credit: David Mortensen

Training an RNN Language Model



Slide credit: David Mortensen

Computing Loss and Gradients in Practice

- In principle, we could compute loss and gradients across the whole corpus $(x^{(1)}, \ldots, x^{(T)})$ but that would be incredibly expensive!

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

- Instead, we usually treat $x^{(1)}, \ldots, x^{(T)}$ as a document, or even a sentence
- This works much better with **Stochastic Gradient Descent**, which lets us compute loss and gradients for little chunks and update as we go.
- Actually, we do this in batches: compute $J(\theta)$ for a batch of sentences; update weights; repeat.

Slide credit: David Mortensen

We Will Skip the Details of Backpropogation in RNNs for Now

- The fact that training RNNs involves backpropagation over timesteps, summing as you go, means that it (the **backpropagation through time** algorithm) is a bit more complicated than backpropagation in feedforward neural networks.
- We will skip these details for now, but you will want to learn them if you are doing serious work with RNNs.

Slide credit: David Mortensen

Advantages of RNN Language Models

- Input can be of an arbitrary length
- Computation can use information from many steps back (in principle)
- Longer inputs do not mean larger model sizes
- Same weights applied at every time step—**symmetry**

Slide credit: David Mortensen

Disadvantages of RNN LMs

- Recurrent computation is slow
    - Computing $h^{(t)}$ requires computing $h^{(t-1)}$ which requires computing $h^{(t-2)}$
    - Cannot be parallelized
- In practice, it is difficult to access information from many steps back (cf. the VANISHING GRADIENT PROBLEM)

Slide credit: David Mortensen

LSTMs Address (but Do not Solve) the Vanishing and Exploding Gradient Problems

- LSTMs: Long Short Term Memory
- Process data sequentially, but keep hidden state through time
- Still subject, at some level, to vanishing gradients, but to a lesser degree that traditional RNNs
- Widely used in language modeling

Slide credit: David Mortensen

Questions?