Learning in log-linear language models

CS 585, Fall 2019 Introduction to Natural Language Processing http://people.cs.umass.edu/~miyyer/cs585/

Mohit lyyer College of Information and Computer Sciences University of Massachusetts Amherst

[some slides adapted from Richard Socher]

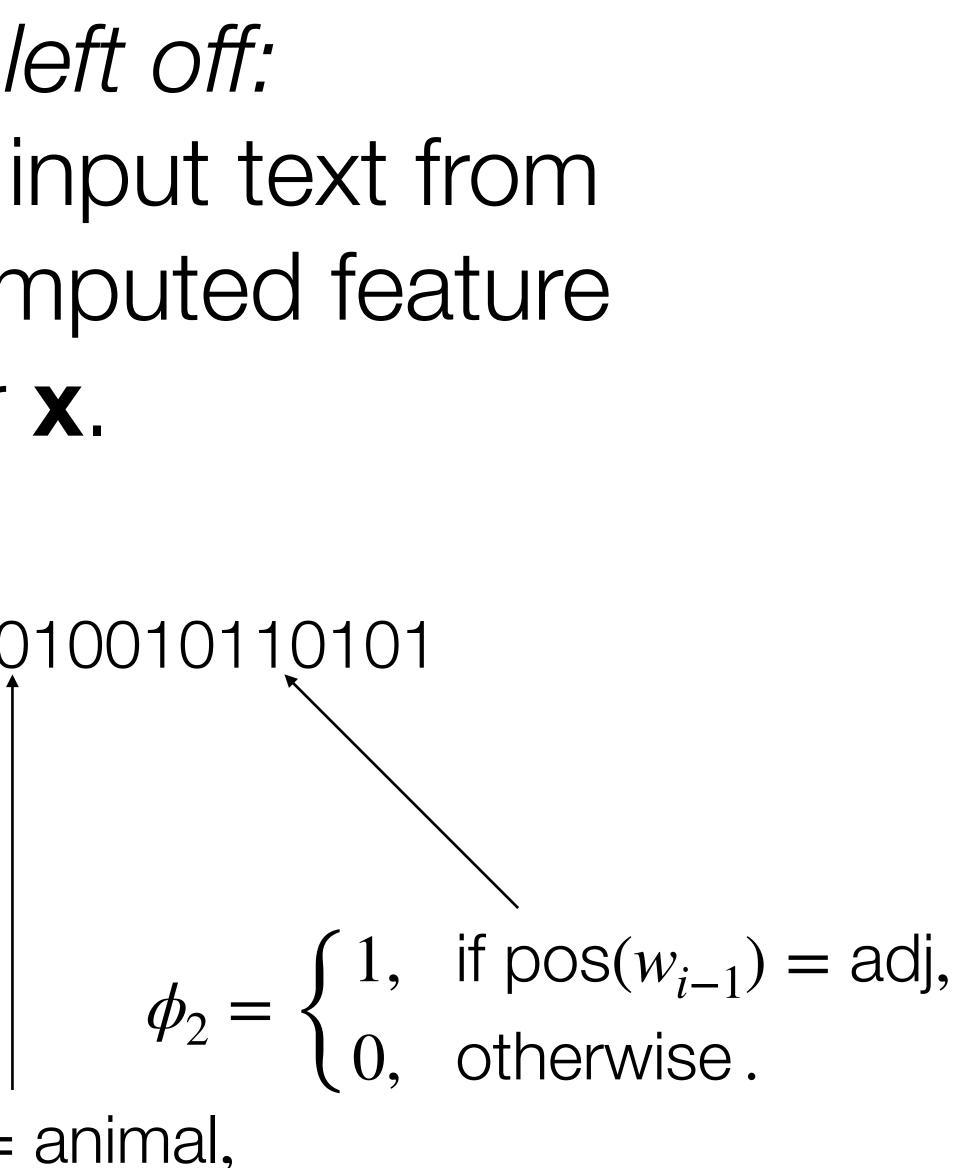
stuff from last class....

- group assignments due by this Thursday Sep 19th at the end of the day, otherwise you'll be randomly assigned!
- HW1 bug fixed, recopy the notebook for those who already started!
- more readings that are research papers? ok
- talk about state-of-the-art models? later
- code libraries for project?

where we left off: so we have some input text from which we have computed feature vector **x**.

x (...Hence, in any statistical) = 10010010110101 $\phi_0 = \begin{cases} 1, & \text{if } w_{i-1} = \text{statistical,} \\ 0, & \text{otherwise.} \end{cases}$ $\phi_2 = \begin{cases} 1, \\ \phi_2 = \begin{cases} 1, \\ 0 \end{cases}$

$$\phi_1 = \begin{cases} 1, & \text{if } w_{i-1} = \\ 0, & \text{otherwise} \end{cases}$$

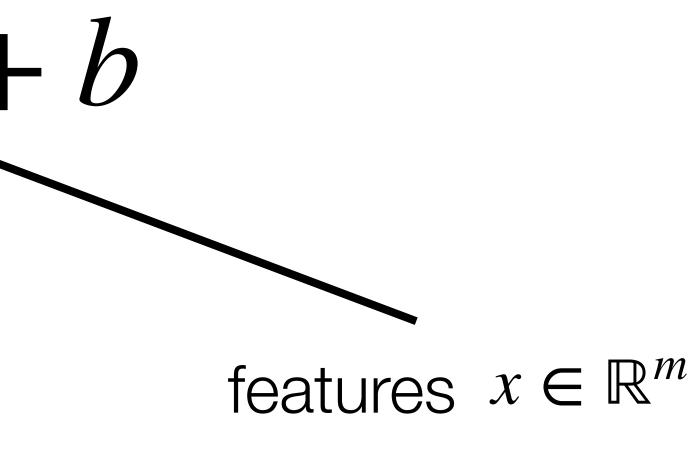


se.

given features x, how do we predict the next word y? s = Wx + bscore vector $s \in \mathbb{R}^{|V|}$ features $x \in \mathbb{R}^m$

weight matrix $W \in \mathbb{R}^{|V| \times m}$

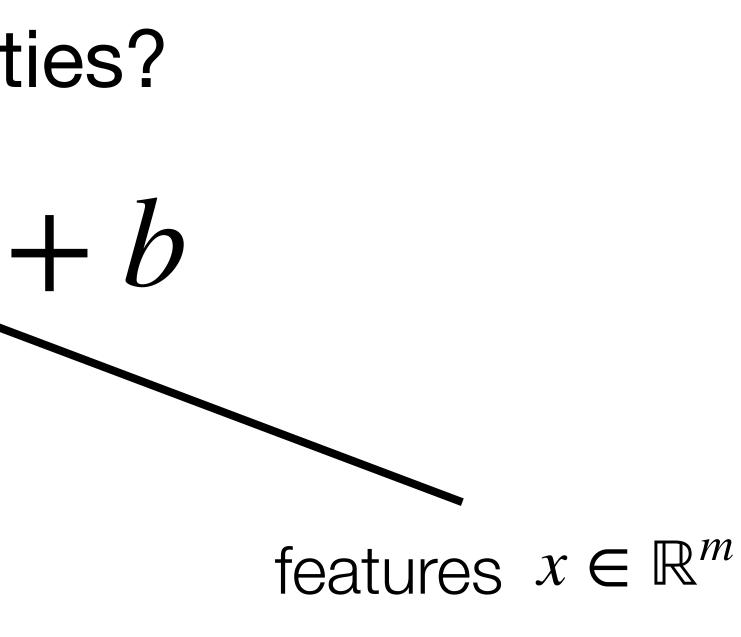
each row of W contains weights for a (word y, \mathbf{x}) pair



how do we obtain probabilities? s = Wx + bscore vector $s \in \mathbb{R}^{|V|}$

weight matrix $W \in \mathbb{R}^{|V| \times m}$

 e^{S_i} $= \frac{1}{\sum_{i} e^{s_j}}; p = \text{softmax}(s)$



colab demo!

what do we have left?

- how do we find the optimal values of W and b for our language modeling problem?
- gradient descent! this involves computing:
 - 1. a loss function, which tells us how good the current values of W and b are on our training data
 - 2. the partial derivatives $\frac{\partial L}{\partial W}$ and $\frac{\partial L}{\partial h}$



what do we have left?

- how do we find the optimal values of W and b for our language modeling problem?
- gradient descent! this involves computing:
 - 1. a loss function, which tells us how good the current values of W and b are on our training data
 - 2. the partial derivatives $\frac{\partial L}{\partial W}$ and $\frac{\partial L}{\partial h}$ kinda like we did in HWO!



first, an aside: what is the bias **b**?

- Let's say we have a feature that is always set to 1 regardless of what the input text is.
- This is clearly not an informative feature. However, let's say it was the only one I had...

first, an aside: what is the bias **b**?

- Let's say we have a feature that is always set to 1 regardless of what the input text is.
- This is clearly not an informative feature. However, let's say it was the only one I had...

first, how many weights do l need to learn for this feature?

first, an aside: what is the bias **b**?

- Let's say we have a feature that is always set to 1 regardless of what the input text is.
- This is clearly not an informative feature. However, let's say it was the only one I had...

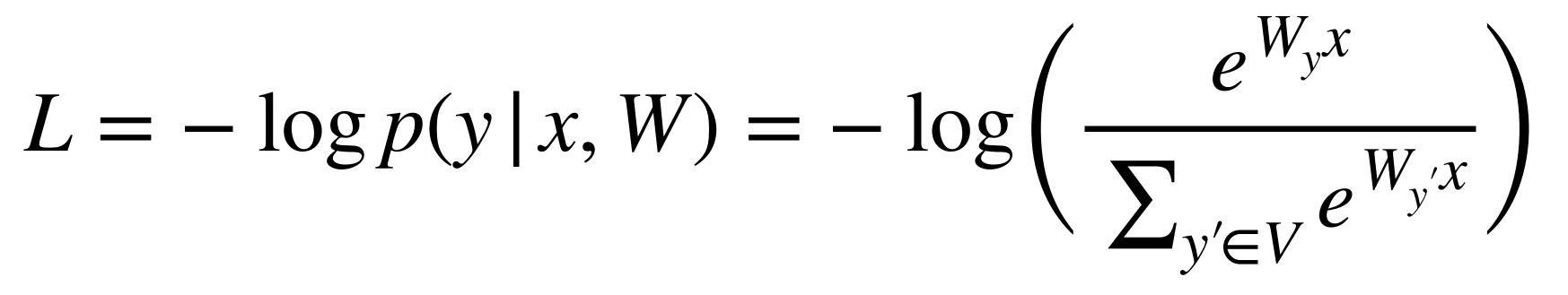
first, how many weights do l need to learn for this feature?

okay... what is the best set of weights for it?

Training with softmax and cross-entropy error

For each training example {x,y}, our objective is to maximize the probability of the correct class y

Hence, we minimize the negative log probability of that class:

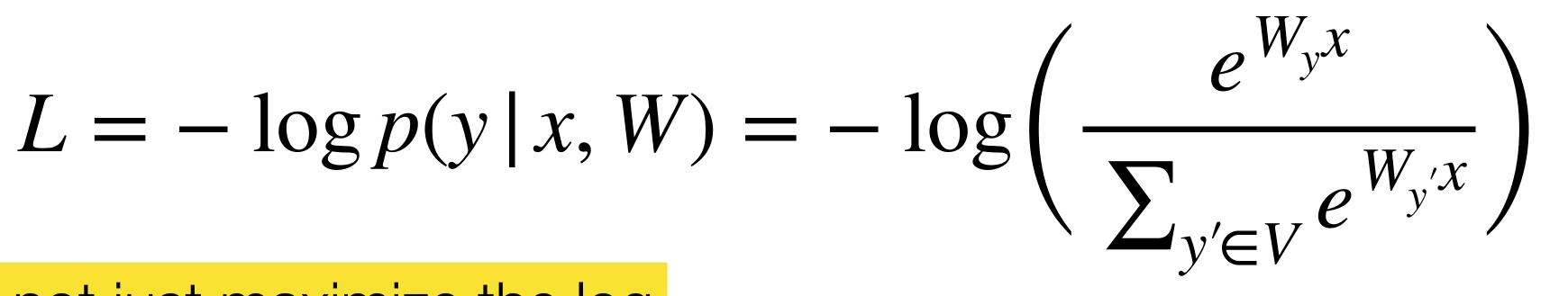


Training with softmax and cross-entropy error

For each training example {x,y}, our objective is to maximize the probability of the correct class y

Hence, we minimize the negative log probability of that class:

why not just maximize the log probability?



Background: Why "Cross entropy" error

Assuming a ground truth (or gold or target) probability distribution that is 1 at the right class and 0 everywhere else: p = [0, ..., 0, 1, 0, ..., 0] and our computed probability is q, then the cross entropy is:

$$H(p,q) = -\sum_{w \in V} I$$

Because of one-hot p, the only term left is the negative log probability of the true class

$p(w)\log q(w)$

let's say I also have the derivatives ∂L ∂L ∂W дh

- the partial derivatives tell us how the loss changes given a change in the corresponding parameter
- we can thus take steps in the *negative* direction of the gradient to *minimize* the loss function

draw on paper



derivation on paper