

TTIC 31190: Natural Language Processing

Kevin Gimpel
Spring 2018

Lecture 7:
Language Modeling, Smoothing,
Neural Language Models

- if you submitted assignment 1, I responded to your email (let me know if you didn't get a response)

- assignment 2 has been posted, due in two weeks from today
- I emailed the class Friday night after posting it (let me know if you didn't get the email)

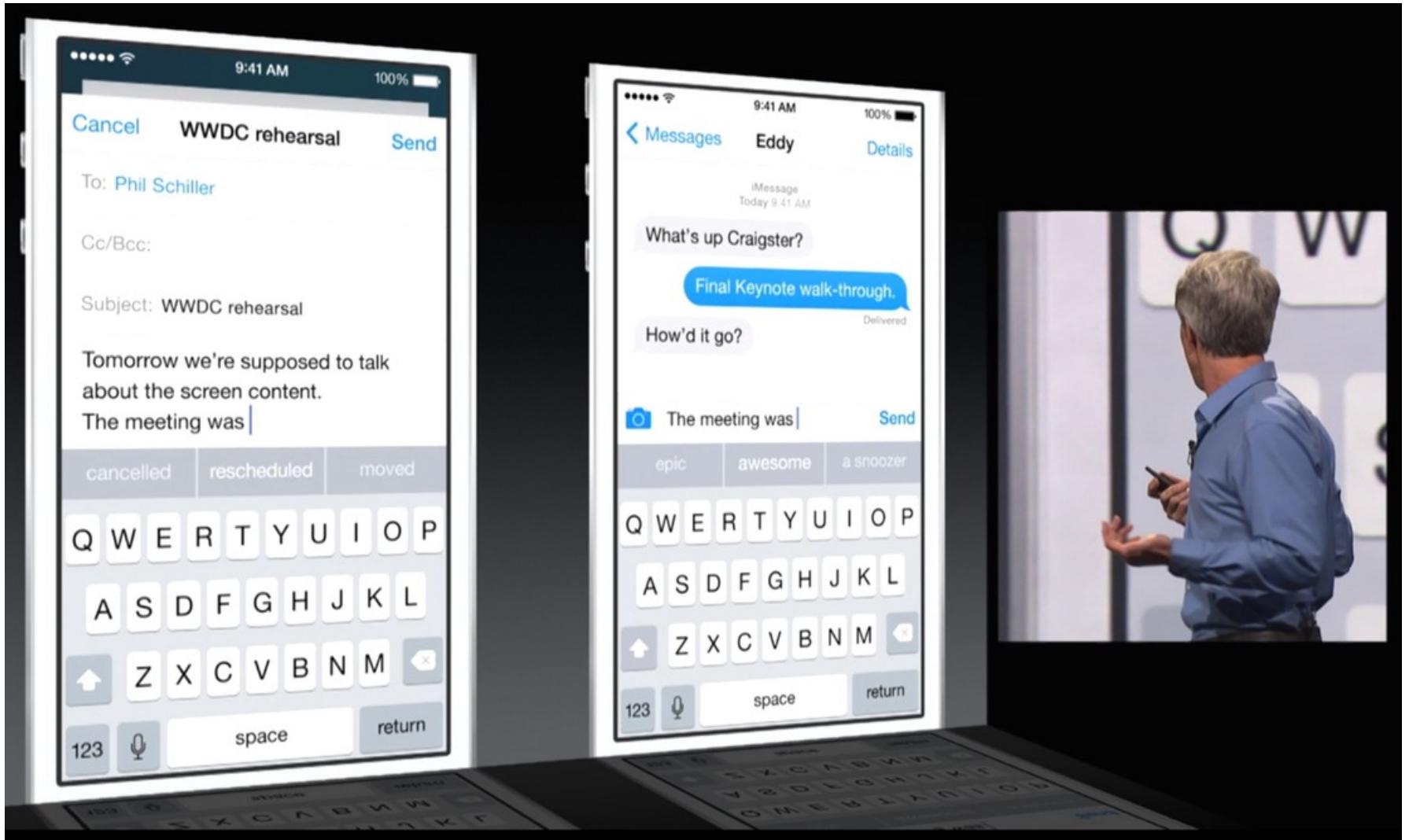
- start thinking about your project, who you might want to work with, etc.

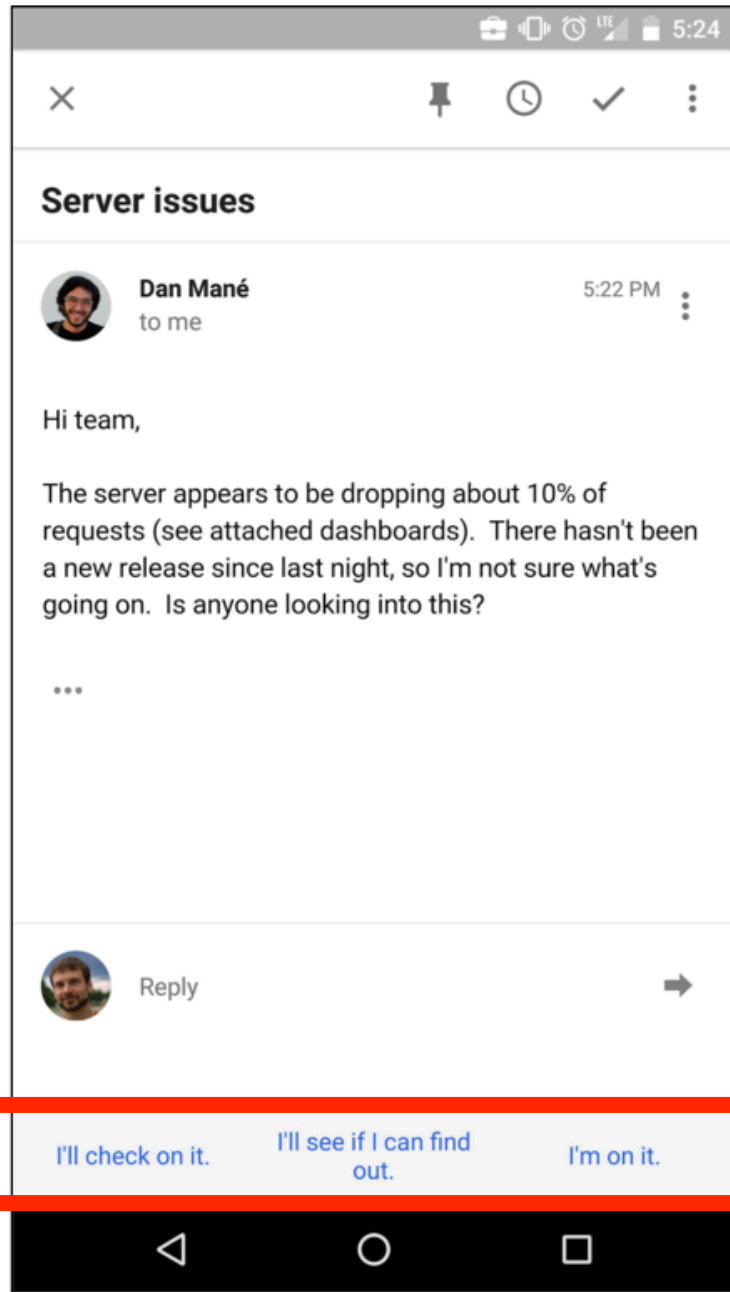
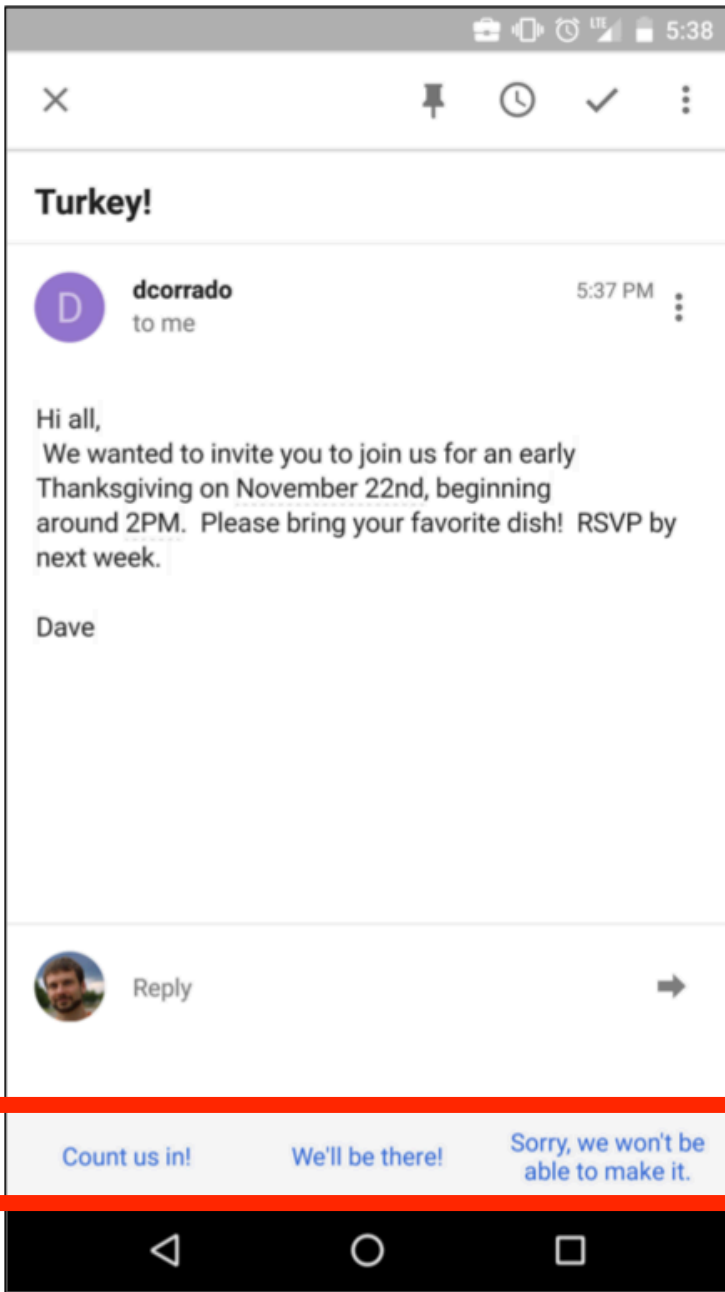
- short quiz at start of class Wed., April 18th
- covering material up to and including Mon., April 9th
- don't stress about it
- grading will be check-minus/check/check-plus

Roadmap

- words, morphology, lexical semantics
- text classification
- language modeling
- word embeddings
- recurrent/recursive/convolutional networks in NLP
- sequence labeling, HMMs, dynamic programming
- syntax and syntactic parsing
- semantics, compositionality, semantic parsing
- machine translation and other NLP tasks

Automatic Completion





Probabilistic Language Modeling

- goal: compute the probability of a sequence of words:

$$P(\mathbf{w}) = P(w_1, w_2, \dots, w_n)$$

- related task: probability of next word:

$$P(w_4 \mid w_1, w_2, w_3)$$

- a model that computes either of these:

$$P(\mathbf{w}) \quad \text{or} \quad P(w_k \mid w_1, w_2, \dots, w_{k-1})$$

is called a **language model (LM)**

Markov Assumption



Andrei Markov

- simplifying assumption:

$P(\text{the } l \text{ its water is so transparent that}) \approx P(\text{the } l \text{ that})$

- or maybe:

$P(\text{the } l \text{ its water is so transparent that}) \approx P(\text{the } l \text{ transparent that})$

Markov Assumption

- i.e., we approximate each component in the product:

$$P(w_i \mid w_1, \dots, w_{i-2}, w_{i-1}) \approx P(w_i \mid w_{i-k}, \dots, w_{i-2}, w_{i-1})$$

Simplest case: Unigram model

$$P(w_1, w_2, \dots, w_n) = \prod_i P(w_i)$$

automatically generated sentences from a unigram model:

fifth an of futures the an incorporated a a the
inflation most dollars quarter in is mass

thrift did eighty said hard 'm july bullish

that or limited the

Bigram model

condition on the previous word:

$$P(w_1, w_2, \dots, w_n) = \prod_i P(w_i | w_{i-1})$$

automatically generated sentences from a bigram model:

texaco rose one in this issue is pursuing growth in a boiler
house said mr. gurria mexico 's motion control proposal
without permission from five hundred fifty five yen

outside new car parking lot of the agreement reached

this would be a record november

Estimating bigram probabilities

- the maximum likelihood estimate (MLE)

$$P(w_i \mid w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$$

An example

$$P(w_i | w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$$

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

$$P(\text{I} | \text{<s>}) = \frac{2}{3} = .67$$

$$P(\text{Sam} | \text{<s>}) = \frac{1}{3} = .33$$

$$P(\text{am} | \text{I}) = \frac{2}{3} = .67$$

$$P(\text{</s>} | \text{Sam}) = \frac{1}{2} = 0.5$$

$$P(\text{Sam} | \text{am}) = \frac{1}{2} = .5$$

$$P(\text{do} | \text{I}) = \frac{1}{3} = .33$$

More examples:

Berkeley Restaurant Project sentences

- can you tell me about any good cantonese restaurants close by
- mid priced thai food is what i'm looking for
- tell me about chez panisse
- can you give me a listing of the kinds of food that are available
- i'm looking for a good place to eat breakfast
- when is caffe venezia open during the day

Raw bigram counts

- counts from 9,222 sentences
- e.g., “*i want*” occurs 827 times

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Raw bigram probabilities

- normalize by unigram counts:

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

- bigram probabilities:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Bigram estimates of sentence probabilities

$$P(\langle s \rangle \text{ I want english food } \langle /s \rangle) =$$

$$P(\text{I} \mid \langle s \rangle)$$

$$\times P(\text{want} \mid \text{I})$$

$$\times P(\text{english} \mid \text{want})$$

$$\times P(\text{food} \mid \text{english})$$

$$\times P(\langle /s \rangle \mid \text{food})$$

$$= .000031$$

Practical Issues

- we do everything in log space
 - avoid underflow
 - (also adding is faster than multiplying)

$$\log(p_1 \times p_2 \times p_3) = \log p_1 + \log p_2 + \log p_3$$

Language Modeling Toolkits

- SRILM
 - <http://www.speech.sri.com/projects/srilm/>
- KenLM
 - <https://kheafield.com/code/kenlm/>

Google N-Gram Release, August 2006

AUG

3

All Our N-gram are Belong to You

Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word [n-gram models](#) for a variety of R&D projects,

...

That's why we decided to share this enormous dataset with everyone. We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than 200 times.

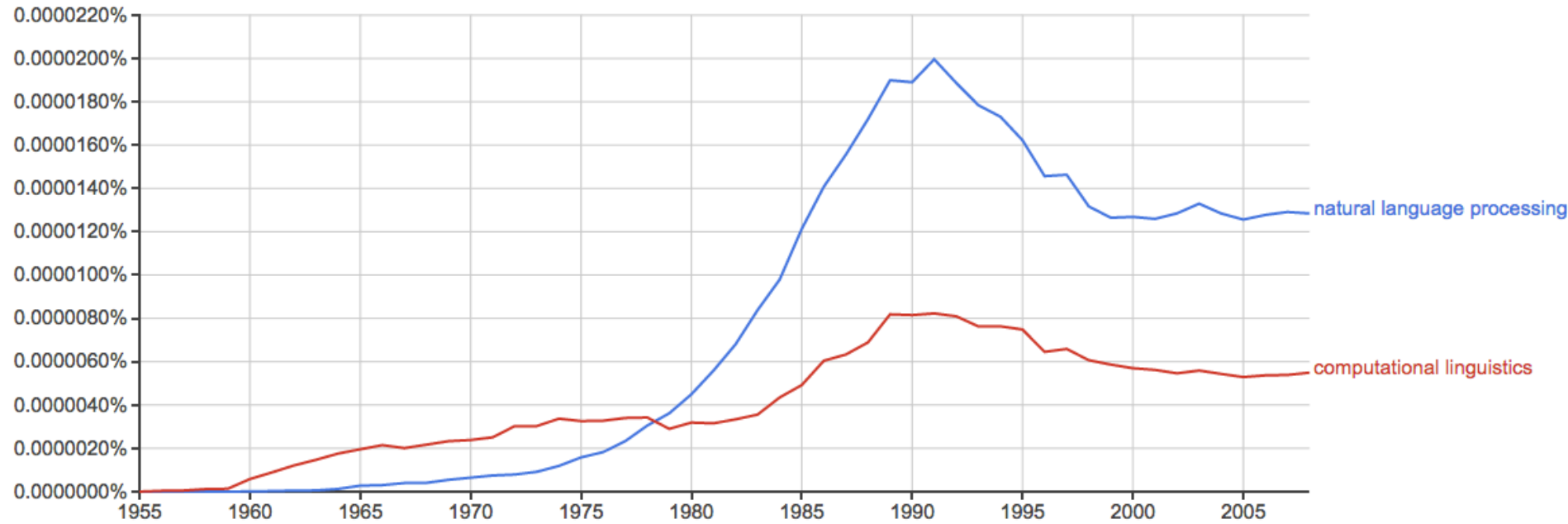
Google N-Gram Release

- serve as the incoming 92
- serve as the incubator 99
- serve as the independent 794
- serve as the index 223
- serve as the indication 72
- serve as the indicator 120
- serve as the indicators 45
- serve as the indispensable 111
- serve as the indispensable 40
- serve as the individual 234

Google Books Ngram Viewer

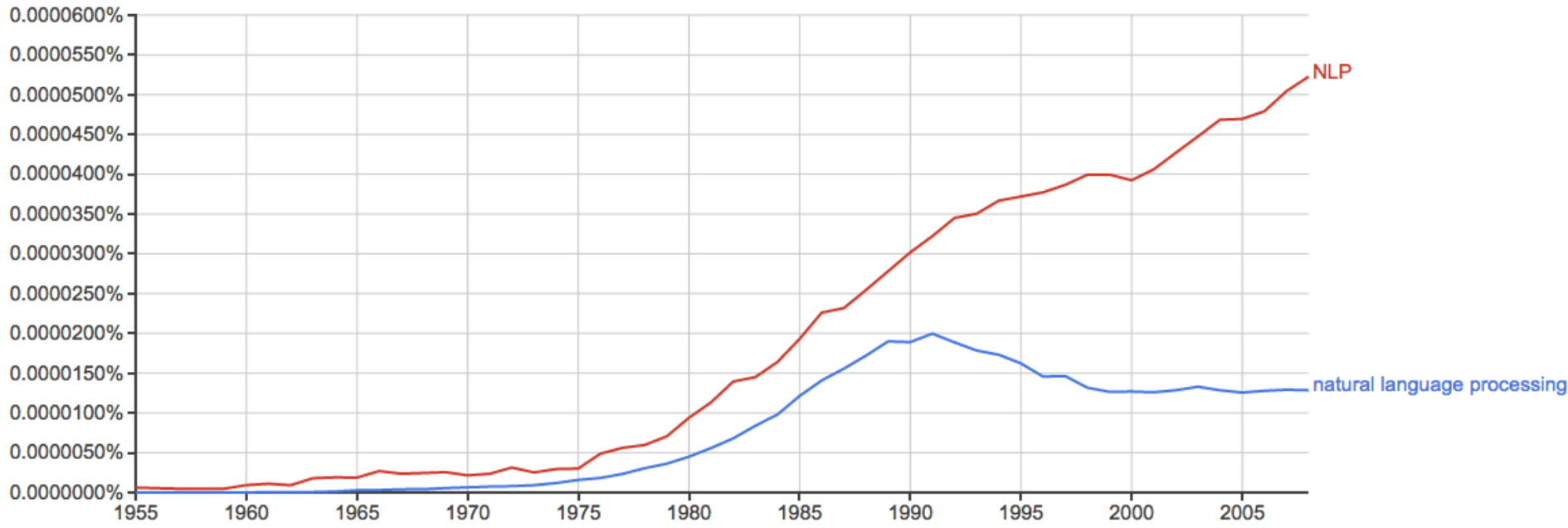
Graph these comma-separated phrases: case-insensitive

between and from the corpus with smoothing of [Search lots of books](#)



Google Books Ngram Viewer

Graph these comma-separated phrases: case-insensitive
between and from the corpus with smoothing of [Search lots of books](#)



Evaluation: How good is our model?

- does our language model prefer good sentences to bad ones?
 - assign higher probability to “real” or “frequently observed” sentences
 - than “ungrammatical” or “rarely observed” sentences?

Extrinsic evaluation of N-gram models

- best evaluation for comparing models A and B
 - put each model in a task
 - spelling corrector, speech recognizer, MT system
 - run the task, get an accuracy for A and for B
 - how many misspelled words corrected properly
 - how many words translated correctly
 - compare accuracy for A and B

Difficulty of extrinsic evaluation of N-gram models

- extrinsic evaluation is time-consuming
 - days or weeks depending on system
- so, sometimes use intrinsic evaluation: **perplexity**
 - bad approximation
 - unless the test data looks **just** like the training data
 - so **generally only useful in pilot experiments**
 - but is helpful to think about

Intuition of Perplexity

- the Shannon Game:
 - how well can we predict the next word?

I always order pizza with cheese and _____

The 33rd President of the US was _____

I saw a _____

mushrooms 0.1

pepperoni 0.1

anchovies 0.01

...

fried rice 0.0001

...

and 1e-100

- unigrams are terrible at this game (why?)
- a better model of a text is one which assigns a higher probability to the word that actually occurs

Probability of Held-out Data

- probability of held-out sentences:

$$\prod_i P(\mathbf{w}^{(i)})$$

- let's work with log-probabilities:

$$\log_2 \prod_i P(\mathbf{w}^{(i)}) = \sum_i \log_2 P(\mathbf{w}^{(i)})$$

- divide by number of words M in held-out sentences:

$$\frac{1}{M} \sum_i \log_2 P(\mathbf{w}^{(i)})$$

Probability -> Perplexity

- average log-probability of held-out words:

$$\ell = \frac{1}{M} \sum_i \log_2 P(\mathbf{w}^{(i)})$$

- perplexity:

$$PP = 2^{-\ell}$$

Perplexity as branching factor

- given a sentence consisting of random digits
- perplexity of this sentence under a model that gives probability $1/10$ to each digit?

$$\begin{aligned}\ell &= \frac{1}{M} \log_2 P(w_1, w_2, \dots, w_M) \\ &= \frac{1}{M} \log_2 \prod_{i=1}^M \frac{1}{10}\end{aligned}$$

Perplexity as branching factor

- given a sentence consisting of random digits
- perplexity of this sentence under a model that gives probability $1/10$ to each digit?

$$\ell = \frac{1}{M} \log_2 P(w_1, w_2, \dots, w_M)$$

$$= \frac{1}{M} \log_2 \prod_{i=1}^M \frac{1}{10}$$

$$PP = 2^{-\ell} = 10$$

$$= \frac{1}{M} \log_2 \left(\frac{1}{10} \right)^M$$

$$= \frac{1}{M} M \log_2 \frac{1}{10}$$

Lower perplexity = better model

- train: 38 million words
- test: 1.5 million words

n-gram order:	unigram	bigram	trigram
perplexity:	962	170	109

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.

Shakespeare as corpus

- 884,647 tokens, 29,066 types
- Shakespeare produced 300,000 bigram types out of 844 million possible bigrams
 - 99.96% of possible bigrams were never seen (have zero entries in the table)
- 4-grams worse: what's coming out looks like Shakespeare because it *is* Shakespeare

Wall Street Journal

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 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

Zeros

training set:

... denied the allegations

... denied the reports

... denied the claims

... denied the request

test set:

... denied the offer

... denied the loan

$$P(\textit{offer} \mid \textit{denied the}) = 0$$

Zero probability bigrams

- test set bigrams with zero probability → assign 0 probability to entire test set!
- cannot compute perplexity (can't divide by 0)!

Intuition of smoothing (from Dan Klein)

- When we have sparse statistics:

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

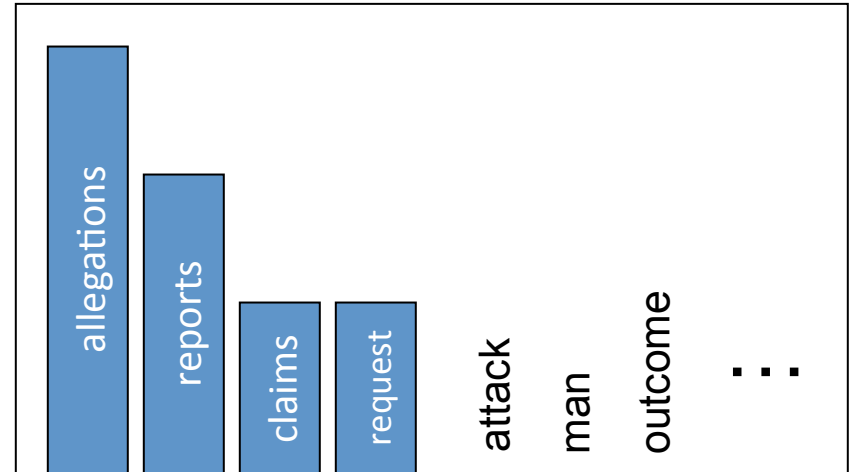
3 *allegations*

2 *reports*

1 *claims*

1 *request*

7 total



- Steal probability mass to generalize better:

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

2.5 *allegations*

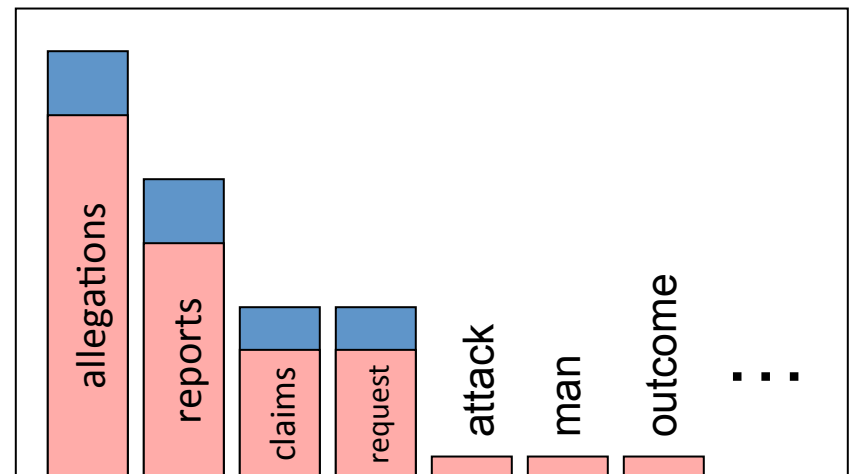
1.5 *reports*

0.5 *claims*

0.5 *request*

2 **other**

7 total



“Add-1” estimation

- also called Laplace smoothing
- pretend we saw each word 1 more time than we did
- just add 1 to all counts!
- MLE estimate:

$$P_{\text{MLE}}(w_i \mid w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$$

- Add-1 estimate:

$$P_{\text{add-1}}(w_i \mid w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i) + 1}{\text{count}(w_{i-1}) + |\mathcal{V}|}$$

Berkeley Restaurant Corpus: Laplace smoothed bigram counts

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Laplace-smoothed bigrams

$$P_{\text{add-1}}(w_i \mid w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i) + 1}{\text{count}(w_{i-1}) + |\mathcal{V}|}$$

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Reconstituted counts

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Compare with raw bigram counts

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Add-1 smoothing is a blunt instrument

- so add-1 isn't used for N-grams:
 - we'll see better methods
- but add-1 is used to smooth other NLP models
 - text classification
 - domains where the number of zeros isn't so huge

Backoff and Interpolation

- sometimes it helps to use **less** context
 - condition on less context for contexts you haven't learned much about
- **backoff:**
 - use trigram if you have good evidence, otherwise bigram, otherwise unigram
- **interpolation:**
 - mixture of unigram, bigram, trigram (etc.) models
- interpolation works better

Linear Interpolation

- simple interpolation:

$$\hat{P}(w_n | w_{n-2} w_{n-1}) = \lambda_1 P(w_n | w_{n-2} w_{n-1}) + \lambda_2 P(w_n | w_{n-1}) + \lambda_3 P(w_n) \quad \sum_i \lambda_i = 1$$

- lambdas are functions of context:

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

How to set the lambdas?

- use a **held-out** corpus:



- choose lambdas to maximize probability of held-out data:
 - fix N-gram probabilities (on the training data)
 - then search for λ s that give largest probability to held-out set:

$$\log P(w_1 \dots w_n \mid M(\lambda_1 \dots \lambda_k)) = \sum_i \log P_{M(\lambda_1 \dots \lambda_k)}(w_i \mid w_{i-1})$$

- subtlety: what happens if we use training data to learn λ s?

Unknown words: open vs. closed vocabulary tasks

- if we know all the words in advance:
 - vocabulary V is fixed
 - “closed vocabulary” task
- often we don’t know this
 - **out-of-vocabulary (OOV)** words
 - “open vocabulary” task
- so, create an unknown word token <UNK>
 - at training time:
 - randomly change some instances of rare words to <UNK>
 - then estimate its probabilities like a normal word
 - at test time:
 - replace OOV words with <UNK>

Huge web-scale n-grams

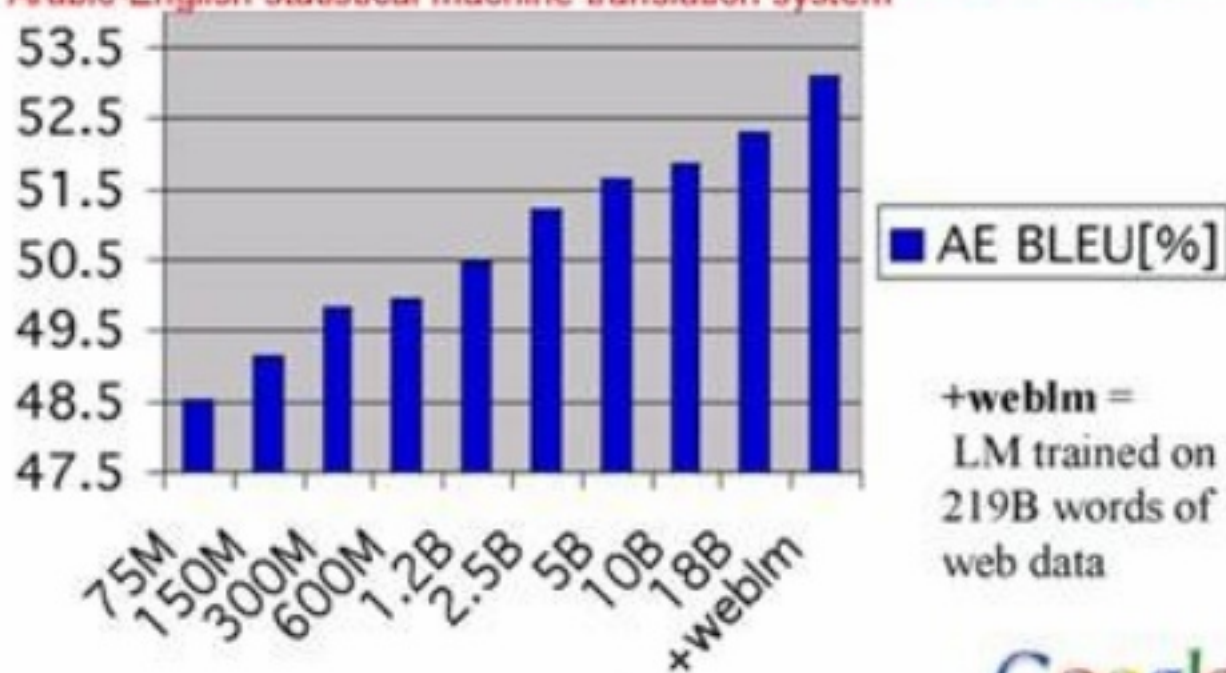
- how to deal with, e.g., Google N-gram corpus?
- pruning:
 - only store N-grams with count $>$ threshold.
 - remove singletons of higher-order n-grams
 - entropy-based pruning
- efficiency
 - efficient data structures like tries
 - bloom filters: approximate language models
 - quantize probabilities (4-8 bits instead of 8-byte float)

Google trillion word language model



More data is better data...

Impact on size of language model training data (in words) on quality of Arabic-English statistical machine translation system



Smoothing for Web-scale N-grams

- “Stupid backoff” (Brants et al., 2007)
- no discounting, just use relative frequencies

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

$$S(w_i) = \frac{\text{count}(w_i)}{N}$$

N-gram Smoothing Summary

- Add-1 estimation:
 - OK for text categorization, not for language modeling
- most commonly used method:
 - modified interpolated Kneser-Ney
- for very large N-gram collections like the Web:
 - stupid backoff

Advanced Language Modeling

- discriminative models:
 - choose n-gram weights to improve a task, not to fit the training set
- syntactic language models
- caching models
 - recently used words are more likely to appear

$$P_{CACHE}(w | history) = \lambda P(w_i | w_{i-2} w_{i-1}) + (1 - \lambda) \frac{c(w \in history)}{|history|}$$

- these perform very poorly for speech recognition (why?)

A Neural Probabilistic Language Model

Yoshua Bengio

Réjean Ducharme

Pascal Vincent

Christian Jauvin

Département d'Informatique et Recherche Opérationnelle

Centre de Recherche Mathématiques

Université de Montréal, Montréal, Québec, Canada

BENGIOY@IRO.UMONTREAL.CA

DUCHARME@IRO.UMONTREAL.CA

VINCENTP@IRO.UMONTREAL.CA

JAUVINC@IRO.UMONTREAL.CA

- idea: use a neural network for n -gram language modeling:

$$P_{\theta}(w_t \mid w_{t-n+1}, \dots, w_{t-2}, w_{t-1})$$

A Neural Probabilistic Language Model

Yoshua Bengio
Réjean Ducharme
Pascal Vincent
Christian Jauvin

Département d'Informatique et Recherche Opérationnelle
Centre de Recherche Mathématiques
Université de Montréal, Montréal, Québec, Canada

BENGIOY@IRO.UMONTREAL.CA
DUCHARME@IRO.UMONTREAL.CA
VINCENTP@IRO.UMONTREAL.CA
JAUVINC@IRO.UMONTREAL.CA

- this is not the earliest paper on using neural networks for n -gram language models, but it's the most well-known and first to scale up
- see paper for citations of earlier work

Neural Probabilistic Language Models

(Bengio et al., 2003)

1.1 Fighting the Curse of Dimensionality with Distributed Representations

In a nutshell, the idea of the proposed approach can be summarized as follows:

1. associate with each word in the vocabulary a distributed *word feature vector* (a real-valued vector in \mathbb{R}^m),
2. express the joint *probability function* of word sequences in terms of the feature vectors of these words in the sequence, and
3. learn simultaneously the *word feature vectors* and the parameters of that *probability function*.

What is a neural network?

- just think of a neural network as a function
- it has inputs and outputs
- “neural” typically means one type of functional building block (“neural layers”), but the term has broadened
- neural modeling is now better thought of as a modeling strategy (leveraging “distributed representations” or “representation learning”), or a family of related methods

Classifier Framework

$$\text{classify}(\mathbf{x}, \mathbf{w}) = \underset{y}{\operatorname{argmax}} \text{ score}(\mathbf{x}, y, \mathbf{w})$$

- linear model score function:

$$\text{score}(\mathbf{x}, y, \mathbf{w}) = \mathbf{w}^\top \mathbf{f}(\mathbf{x}, y) = \sum_i w_i f_i(\mathbf{x}, y)$$

- we can also use a neural network for the score function!

Notation

\mathbf{u} = a vector

u_i = entry i in the vector

\mathbf{W} = a matrix

w_{ij} = entry (i,j) in the matrix

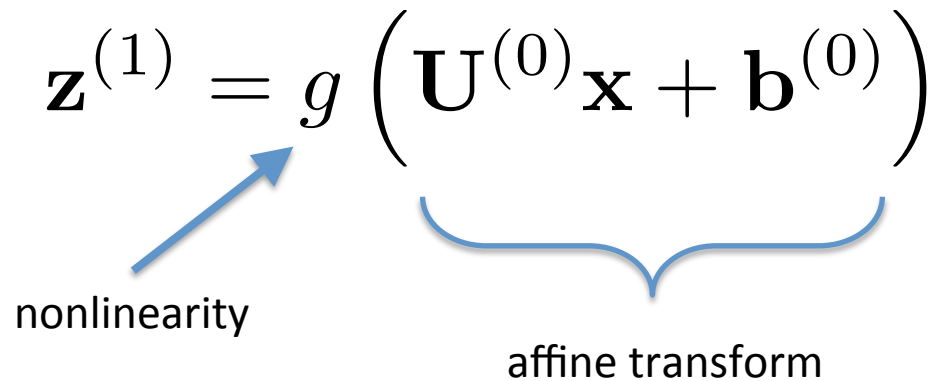
\mathcal{x} = a structured object

x_i = entry i in the structured object

neural layer = affine transform + nonlinearity

$$\mathbf{z}^{(1)} = g \left(\underbrace{\mathbf{U}^{(0)} \mathbf{x} + \mathbf{b}^{(0)}}_{\text{affine transform}} \right)$$

nonlinearity



- this is a single “layer” of a neural network
- input vector is \mathbf{X}

neural layer = affine transform + nonlinearity

$$\mathbf{z}^{(1)} = g \left(\underbrace{\mathbf{U}^{(0)} \mathbf{x} + \mathbf{b}^{(0)}}_{\text{affine transform}} \right)$$

nonlinearity

- this is a single “layer” of a neural network
- input vector is \mathbf{X}
- $\mathbf{U}^{(0)}$ and $\mathbf{b}^{(0)}$ are parameters

neural layer = affine transform + nonlinearity

$$\mathbf{z}^{(1)} = g \left(\mathbf{U}^{(0)} \mathbf{x} + \mathbf{b}^{(0)} \right)$$

“hidden units”

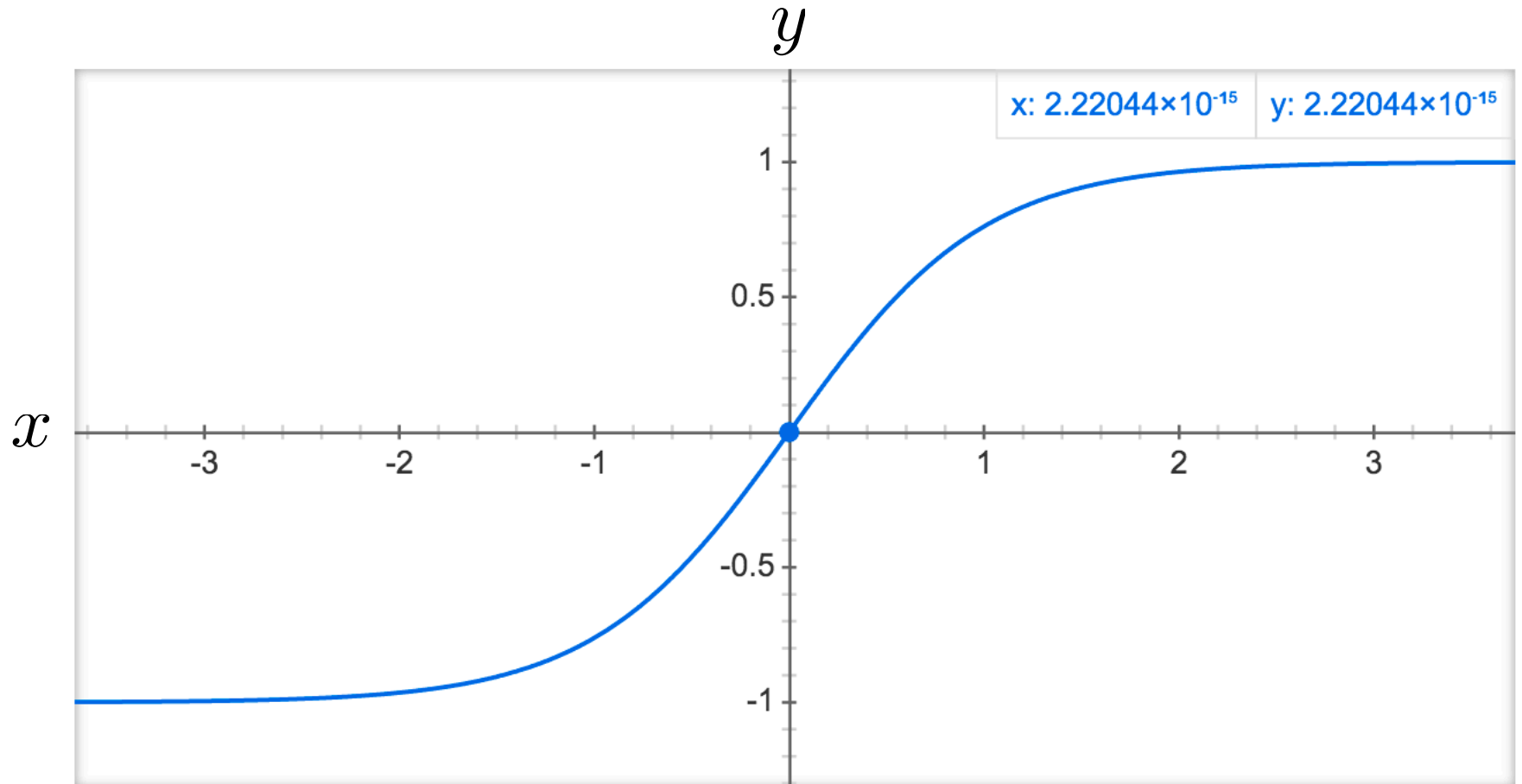
- vector of “hidden units” is $\mathbf{z}^{(1)}$
- think of these as features computed from \mathbf{x}

Nonlinearities

$$\mathbf{z}^{(1)} = \boxed{g}\left(\mathbf{U}^{(0)}\mathbf{x} + \mathbf{b}^{(0)}\right)$$

- most common: elementwise application of g function to each entry in vector
- examples...

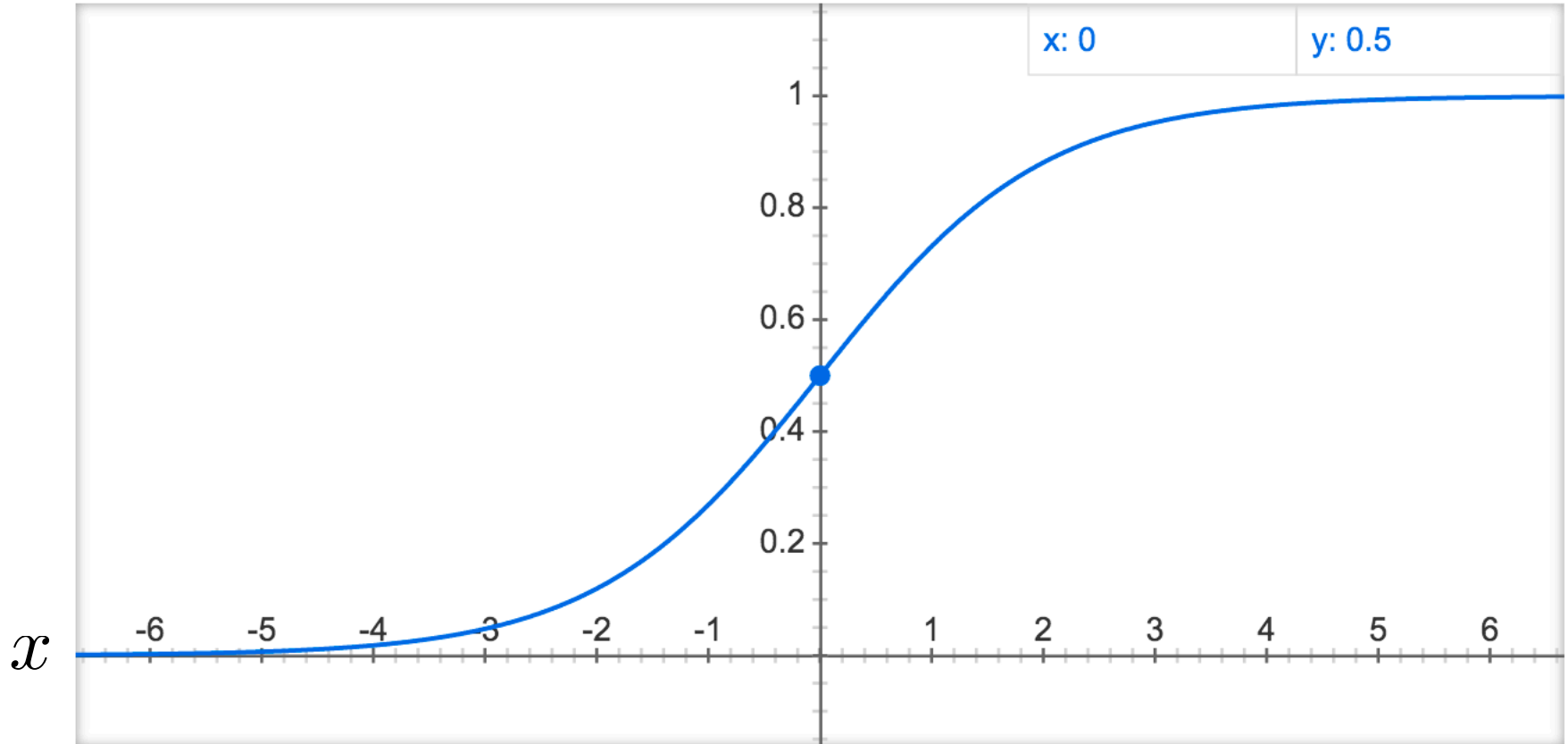
tanh: $y = \tanh(x)$



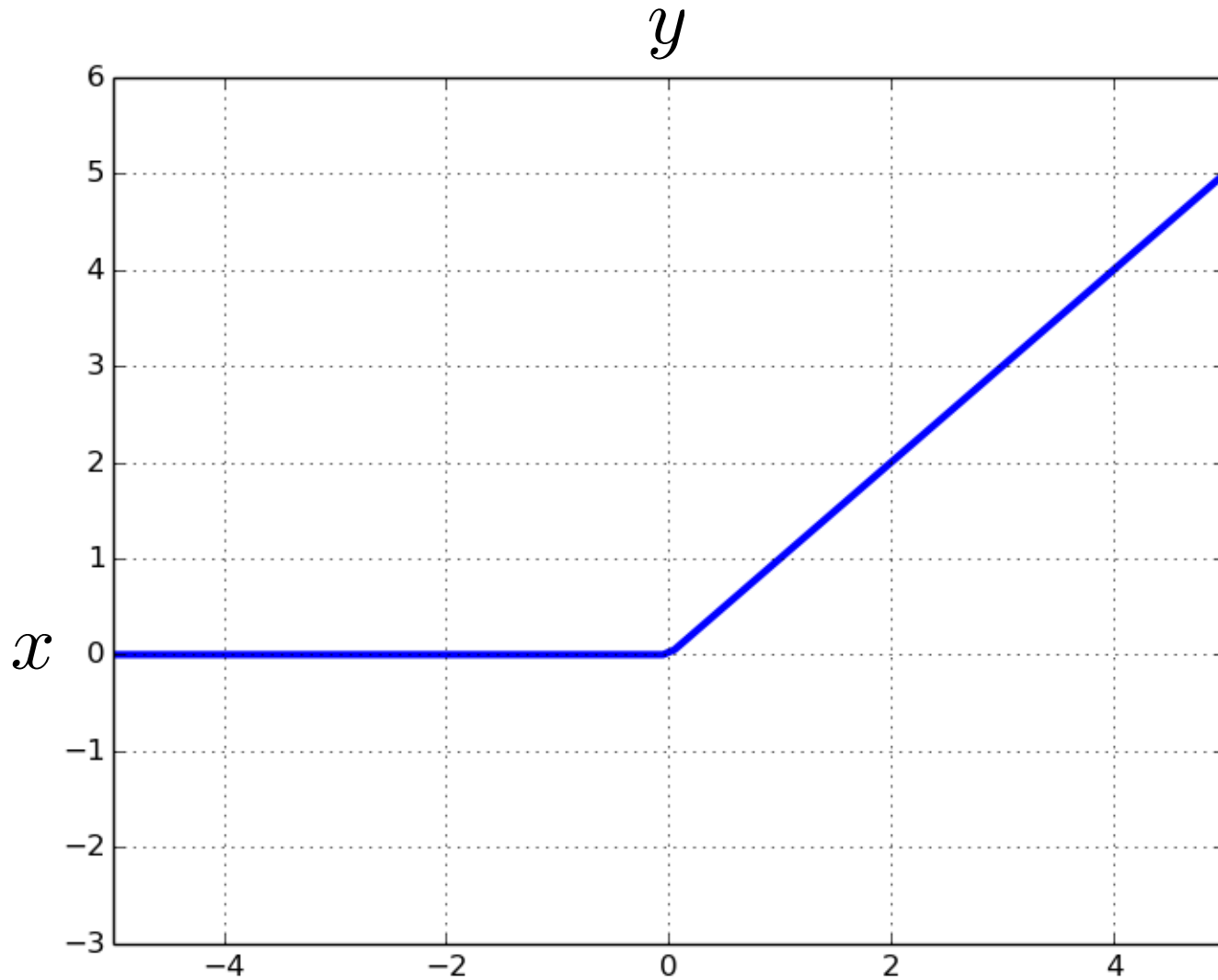
(logistic) sigmoid:

$$y = \frac{1}{1 + \exp\{-x\}}$$

y



rectified linear unit (ReLU): $y = \max(0, x)$



Adding layers...

$$\mathbf{z}^{(1)} = g \left(\mathbf{U}^{(0)} \mathbf{x} + \mathbf{b}^{(0)} \right)$$

$$\mathbf{z}^{(2)} = g \left(\mathbf{U}^{(1)} \mathbf{z}^{(1)} + \mathbf{b}^{(1)} \right)$$

...

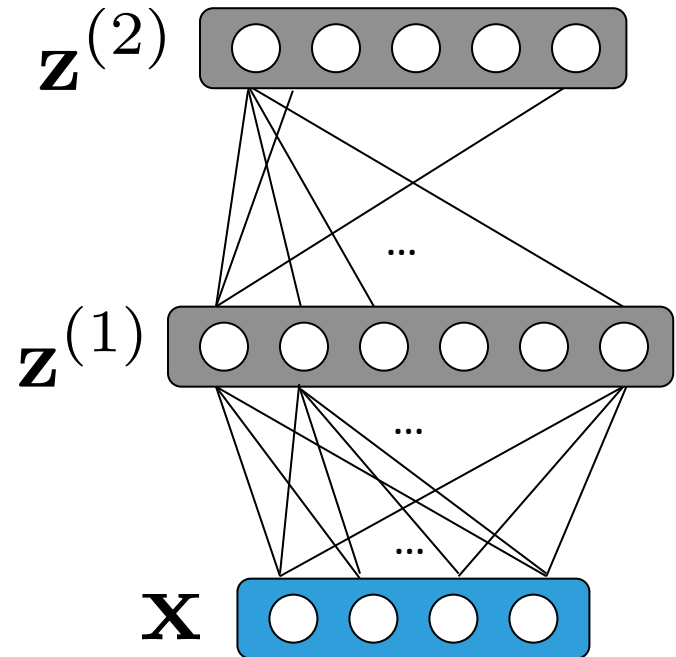
- use output of one layer as input to next

Adding layers...

$$\mathbf{z}^{(1)} = g \left(\mathbf{U}^{(0)} \mathbf{x} + \mathbf{b}^{(0)} \right)$$

$$\mathbf{z}^{(2)} = g \left(\mathbf{U}^{(1)} \mathbf{z}^{(1)} + \mathbf{b}^{(1)} \right)$$

...



- use output of one layer as input to next
- “feed-forward” and/or “fully-connected” layers

Neural Network for Sentiment Classification

$$\mathbf{z}^{(1)} = g \left(\mathbf{U}^{(0)} \mathbf{x} + \mathbf{b}^{(0)} \right)$$

$$\mathbf{s} = \mathbf{U}^{(1)} \mathbf{z}^{(1)} + \mathbf{b}^{(1)}$$



vector of label scores

Neural Network for Sentiment Classification

$$\mathbf{z}^{(1)} = g \left(\mathbf{U}^{(0)} \mathbf{x} + \mathbf{b}^{(0)} \right)$$

$$\mathbf{s} = \mathbf{U}^{(1)} \mathbf{z}^{(1)} + \mathbf{b}^{(1)}$$



$$\mathbf{s} = \begin{bmatrix} \text{score}(\mathbf{x}, \text{positive}, \mathbf{w}) \\ \text{score}(\mathbf{x}, \text{negative}, \mathbf{w}) \end{bmatrix}$$

Use softmax function to convert scores into probabilities

$$\mathbf{s} = \begin{bmatrix} \text{score}(\mathbf{x}, \text{positive}, \mathbf{w}) \\ \text{score}(\mathbf{x}, \text{negative}, \mathbf{w}) \end{bmatrix}$$

$$\mathbf{p} = \text{softmax}(\mathbf{s}) = \begin{bmatrix} \frac{\exp\{\text{score}(\mathbf{x}, \text{positive}, \mathbf{w})\}}{Z} \\ \frac{\exp\{\text{score}(\mathbf{x}, \text{negative}, \mathbf{w})\}}{Z} \end{bmatrix}$$

$$Z = \exp\{\text{score}(\mathbf{x}, \text{positive}, \mathbf{w})\} + \exp\{\text{score}(\mathbf{x}, \text{negative}, \mathbf{w})\}$$

Why nonlinearities?

network with
1 hidden layer:

$$\mathbf{z}^{(1)} = g \left(\mathbf{U}^{(0)} \mathbf{x} + \mathbf{b}^{(0)} \right)$$


$$\mathbf{s} = \mathbf{U}^{(1)} \mathbf{z}^{(1)} + \mathbf{b}^{(1)}$$

- if g is linear, then we can rewrite the above as a single affine transform
- can you prove this? (use distributivity of matrix multiplication)

Learning with Neural Networks

$$\mathbf{z}^{(1)} = g\left(\mathbf{U}^{(0)}\mathbf{x} + \mathbf{b}^{(0)}\right)$$

$$\mathbf{s} = \mathbf{U}^{(1)}\mathbf{z}^{(1)} + \mathbf{b}^{(1)}$$


$$\mathbf{s} = \begin{bmatrix} \text{score}(\mathbf{x}, \text{positive}, \mathbf{w}) \\ \text{score}(\mathbf{x}, \text{negative}, \mathbf{w}) \end{bmatrix}$$

$$\text{classify}(\mathbf{x}, \mathbf{w}) = \underset{y}{\operatorname{argmax}} \text{score}(\mathbf{x}, y, \mathbf{w})$$

- we can use any of our loss functions from before, as long as we can compute (sub)gradients
- algorithm for doing this efficiently: **backpropagation**
- basically just the chain rule of derivatives

Computation Graphs

- a useful way to represent the computations performed by a neural model (or any model!)
- why useful? makes it easy to implement automatic differentiation (backpropagation)
- many neural net toolkits let you define your model in terms of computation graphs (PyTorch, TensorFlow, DyNet, Theano, etc.)