



Natural Language Processing

Session-1



BITS Pilani
Pilani Campus

The instructor is gratefully acknowledging the authors who made their course materials freely available online.

Session Content

- Objective of course
 - Evaluation Plan
 - What is Natural Language Processing?
 - Application areas of Natural Language Processing
 - What will we learn in this course?
 - Introduction to Natural Language Processing
-

Objective of course

- To learn the fundamental concepts and techniques of natural language processing (NLP)
- To learn computational properties of natural languages and the commonly used algorithms for processing linguistic information
- To apply NLP techniques in state of art applications
- To learn implementation of NLP algorithms and techniques

Books

Text books and Reference book(s)

T1	Speech and Language processing: An introduction to Natural Language Processing, Computational Linguistics and speech Recognition by Daniel Jurafsky and James H. Martin[3rd edition]
T2	Natural language understanding[2nd edition] by James Allen
R1	Handbook of Natural Language Processing, Second Edition— NitinIndurkhy, Fred J. Damerau, Fred J. Damerau
R2	Natural Language Processing with Python by Steven Bird, Ewan Klein, Edward Lopper

Evaluation Plan

Name	Type	Duration	Weight
Assignment-I	Take Home		15%
Quiz	Online		5%
Mid-Semester Exam (after 7 sessions)	Closed Book 1.5 Hrs		30%
Comprehensive Exam	Open Book 2.5 Hrs		50%

What is Natural Language Processing?

- Natural Language Processing
 - Process information contained in natural language text.
 - Also known as Computational Linguistics (CL), Human Language Technology (HLT), Natural Language Engineering (NLE)

What is it..

- Analyze, understand and generate human languages just like humans do.
- Applying computational techniques to language domain..
- To explain linguistic theories, to use the theories to build systems that can be of social use..
- Started off as a branch of Artificial Intelligence..
- Borrows from Linguistics, Psycholinguistics, Cognitive Science & Statistics.
- Make computers learn our language rather than we learn theirs.

Why Study NLP?

- A hallmark of human intelligence.
 - Text is the largest repository of human knowledge and is growing quickly.
 - emails, news articles, web pages, IM, scientific articles, insurance claims, customer complaint letters, transcripts of phone calls, technical documents, government documents, patent portfolios, court decisions, contracts,
 - Are we reading any faster than before?
-

Why are language technologies needed?

- Many companies make a lot of money if they could use computer programmes that understood text or speech.
 - answering the phone, and replying to a question
 - understanding the text on a Web page to decide who it might be of interest to
 - translating a daily newspaper from Japanese to English
 - understanding text in journals / books and building an expert systems based on that understanding
-

Dreams or reality??

- Will my computer talk to me like another human ??
 - Will the search engine get me exactly what I am looking for??
 - Can my PC read the whole newspaper and tell me the important news only..??
 - Can my palmtop translate what that Japanese lady is telling me.. ??
 - Ahhh.. Can my PC do my NLP assignments ??
 - Do you know how our brain processes language ??
-

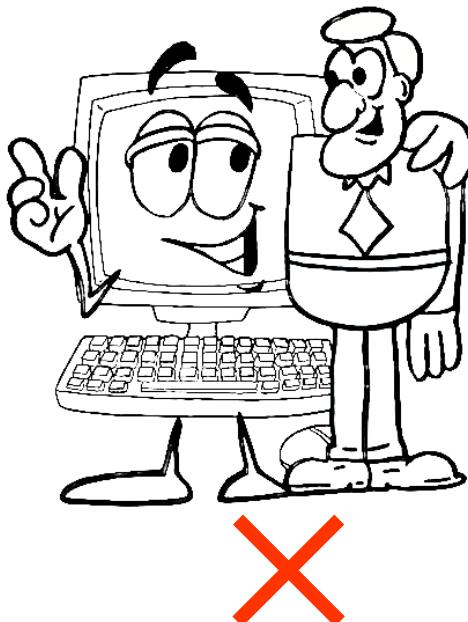
The Dream

- It'd be great if machines could
 - Process our email (usefully)
 - Translate languages accurately
 - Help us manage, summarize, and aggregate information
 - Talk to us / listen to us
- But they can't:
 - Language is complex, ambiguous, flexible, and subtle
 - Good solutions need linguistics and machine learning knowledge



The mystery

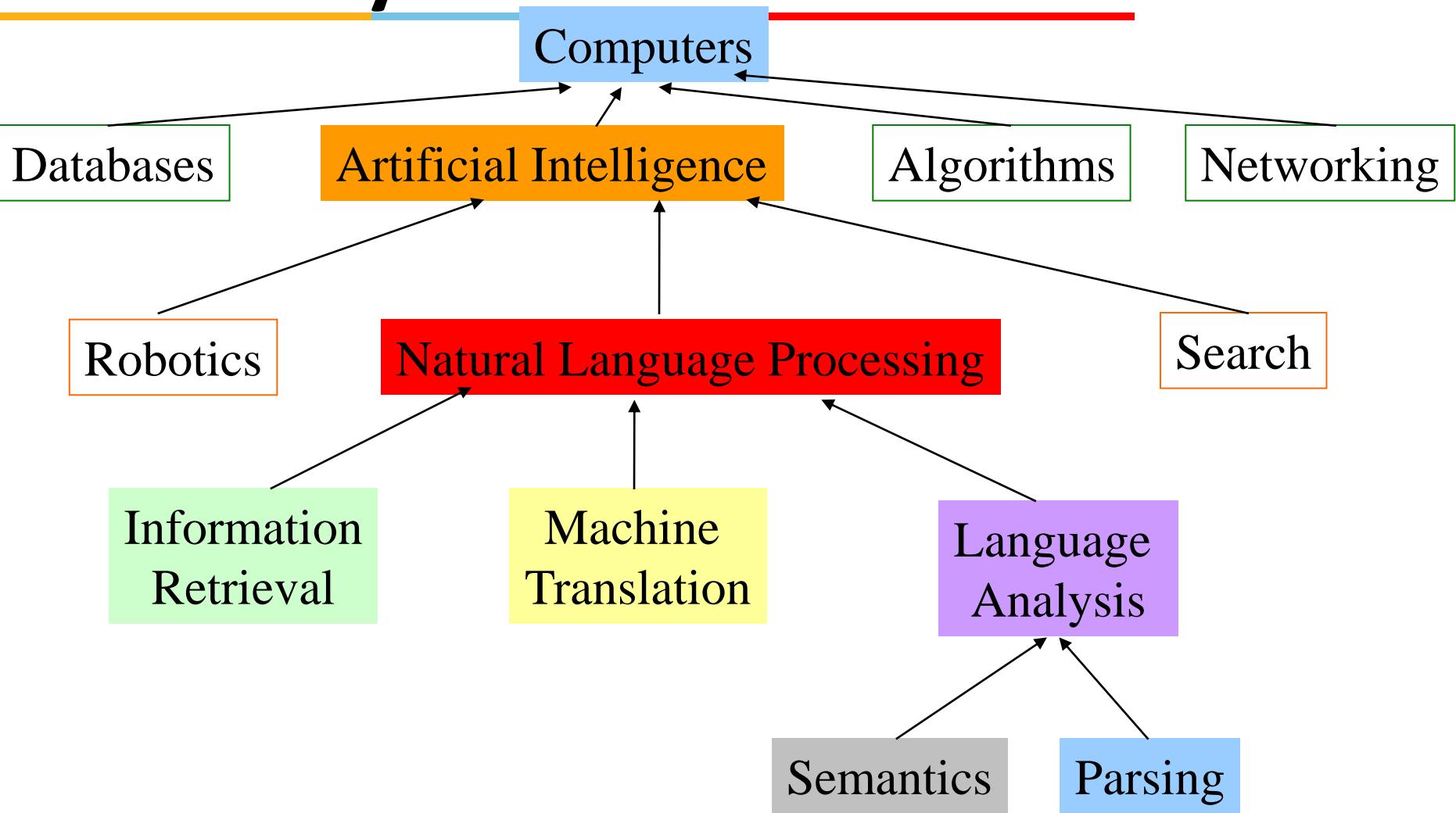
- What's now impossible for computers (and any other species) to do is effortless for humans



Dreams??



Where does it fit in the CS taxonomy?



Brief history of NLP

- 1966: Eliza
- 1988: Latent Semantic Analysis patent
- January 2011: IBM Watson beats Jeopardy! champions
- October 2011: Apple Siri launches in beta
- April 2014: Microsoft Cortana demoed
- November 2014: Amazon Alexa
- May 2016: Google Assistant

2020 – Conversational Agents

What We will learn in this Course

- Introduction to Natural Language Understanding
 - N-gram Language Models
 - Hidden Markov Models
 - Part-of-Speech Tagging
 - Grammars and Parsing
 - Statistical Constituency Parsing
 - Word sense and word net
 - Dependency Parsing
 - Statistical Machine translation
 - Semantic web ontology
 - Question Answering
 - Dialogue Systems and Chatbots
 - Sentiment analysis
 - Implementation using NLTK
-

Introduction to Natural Language Processing

- The Study of Language.
- Applications of Natural Language Understanding.
- Evaluating Language Understanding Systems.
- The Different Levels of Language Analysis.
- Representations and Understanding.
- The Organization of Natural Language Processing Systems.

Dave: Open the pod bay doors, HAL.

HAL: I am sorry, Dave. I am afraid I can't do that.

Dave: What's the problem.

HAL: I think you know what the problem is just as well as I do.

Dave: I don't know what you're talking about.

HAL: I know that you and Frank were planning to disconnect me, and I'm afraid that's something I cannot allow to happen.

General speech and language understanding and generation capabilities

Politeness: emotional intelligence

Self-awareness: a model of self, including goals and plans

Belief ascription: modeling others; reasoning about their goals and plans

Hal: I can tell from the tone of your voice, Dave, that you're upset.
Why don't you take a stress pill and get some rest.

[Dave has just drawn another sketch of Dr. Hunter].

HAL: Can you hold it a bit closer?

[Dave does so].

HAL: That's Dr. Hunter, isn't it?

Dave: Yes.

Recognition of emotion from speech

Vision capability including visual recognition of emotions and faces

Also: situational ambiguity

To attain the levels of performance we attribute to HAL, we need to be able to define, model, acquire and manipulate

- Knowledge of the world and of agents in it,
- Text meaning,
- Intention

NLP Applications

- Question answering
 - Who is the first Taiwanese president?
 - Text Categorization/Routing
 - e.g., customer e-mails.
 - Text Mining
 - Find everything that can be done with NLP
 - Machine (Assisted) Translation
 - Language Teaching/Learning
 - Usage checking
 - Spelling correction
 - Is that just dictionary lookup?
-

Application areas

- Text-to-Speech & Speech recognition
 - Healthcare
 - Natural Language Dialogue Interfaces to Databases
 - Information Retrieval
 - Information Extraction
 - Document Classification
 - Document Image Analysis
 - Automatic Summarization
 - Text Proofreading – Spelling & Grammar
 - Machine Translation
 - Story understanding systems
 - Plagiarism detection
 - Can u think of anything else ??
-

Some commercial tools

- [IBM Watson](#) | A pioneer AI platform for businesses
- [Google Cloud NLP API](#) | Google technology applied to NLP
- [Amazon Comprehend](#) | An AWS service to get insights from text

Open Source Tools

- [Stanford Core NLP](#) is a popular Java library built and maintained by Stanford University.
- **SpaCy** - One of the newest open-source Natural Language Processing with Python libraries
- [Gensim](#) is a highly specialized Python library that largely deals with topic modeling tasks using algorithms like Latent Dirichlet Allocation (LDA)
- [Natural Language Toolkit \(NLTK\)](#) is the most popular Python library

- The Natural Language Toolkit (NLTK) is a platform used for building programs for text analysis.

Open Anaconda terminal, run

pip install nltk.

Anaconda and Jupiter are best and popular data science tools

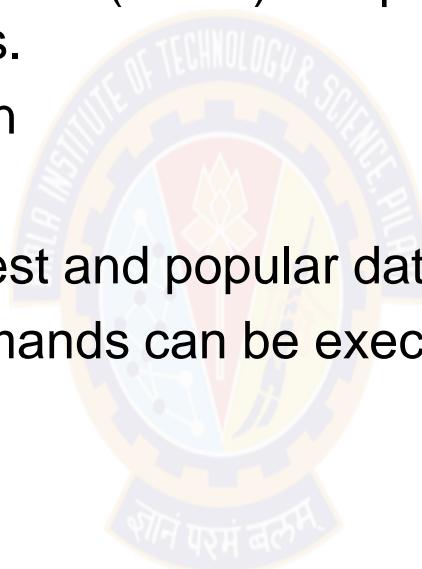
In Jupyter, the console commands can be executed by the ‘!’ sign before the command within the cell.

! pip install nltk

[NLTK book](#)

[NLTK discussion forum](#)

<https://www.nltk.org/install.html>



Why is NLP Big Deal

- L = Words + rules + exceptions..
- Ambiguity at all levels..
- We speak different languages..
- And language is a cultural entity..
- So they are not equivalent..
- Highly systematic but also complex..
- Keeps changing.. New words, New rules and New exceptions..
- Source : Electronic texts / Printed texts / Acoustic Speech Signal..
they are noisy..
- Language looks obvious to us.. But it is a Big Deal 😊!



Types of Ambiguities

I. Structural Ambiguities

- Namrata thinks she understands me.
- She thinks Namrata understands me.
- Visiting relatives can be nuisance. (two meanings)

II. Grammatical Ambiguities

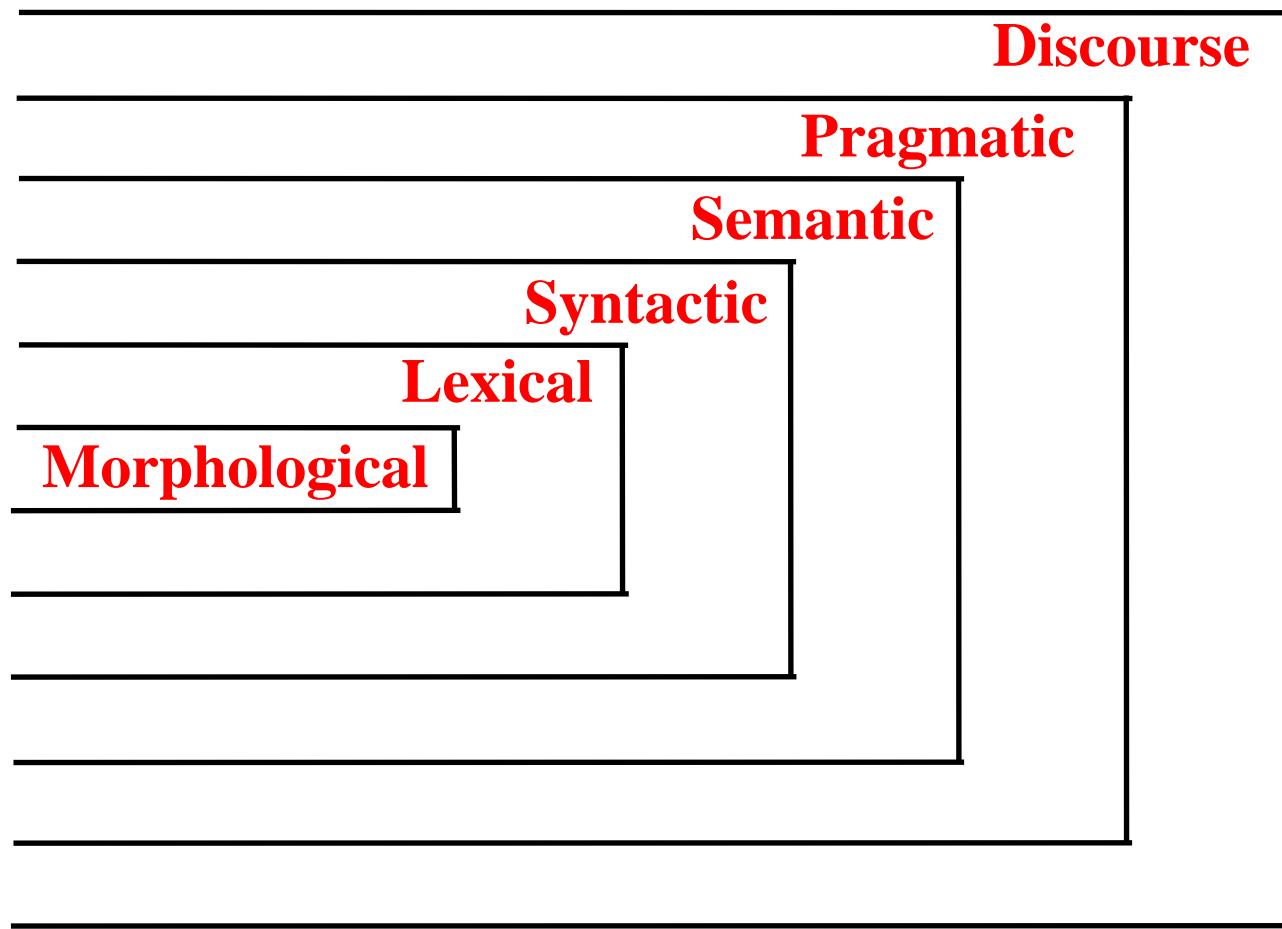
- I (feminine or masculine) go.
- Can- Noun = container, Can – Modal(auxiliary verb),
Can-verb = to can means to pack etc

III. Lexical Ambiguities:

Polysemy Ex: "understand" (I get it)

- Homonymy Ex: Bank= river, financial bank

Different Levels of Language Analysis



Levels of language understanding

Morphological Knowledge : Concerns how words are constructed from basic meaning units called morphemes. A morpheme is the primitive unit of meaning in a language (Ex: “*friendly*” is derived from the meaning of noun “*friend*” and suffix “-ly”, which transforms noun into adjective)

Lexical Knowledge : Concerns with listing of words and categorizing them Ex: friendful or beautyship is incorrect lexically. But friendship and beautiful is correct

Syntactic Knowledge : Concerns how words can be put together to form correct sentences and determines what structural role each word plays in the sentence and what phrases are subpart of other phrases Ex: “Large have green ideas nose” is lexically correct but syntactically incorrect.

Levels of language understanding

Semantic Knowledge :Concerns what words mean and how these meanings – combine in sentences to form sentence meanings. This is the study of context-independent meaning. Ex: “Green ideas have large noses” is syntactically correct but semantically incorrect.

Pragmatic Knowledge :Concerns how sentences are used in different situations how use affects the interpretation of sentence “She cuts banana with a pen” is semantically correct but pragmatically incorrect as it has no useful meaning.

Discourse Knowledge :Concerns how the immediately preceding sentences affect the interpretation of next sentence
Ex: Chetana is a PhD student at IIT bombay. She is also teacher at Cummins College of Engineering for Women.

Why is NLP difficult?



Where is Black Panther playing in Mountain View?

Black Panther is playing at the Century 16 Theater.

When is it playing there?

It's playing at 2pm, 5pm, and 8pm.



OK. I'd like 1 adult and 2 children for the first show.
How much would that cost?

Need domain knowledge, discourse knowledge, world knowledge

Context Free Grammar

-
- (1) $S \rightarrow NP VP$
 - (2) $NP \rightarrow ART ADJ N$
 - (3) $NP \rightarrow ART N$
 - (4) $NP \rightarrow ADJ N$
 - (5) $VP \rightarrow AUX VP$
 - (6) $VP \rightarrow V NP$
-

Representations and Understanding

Allen 1995: Natural Language Understanding - Introduction

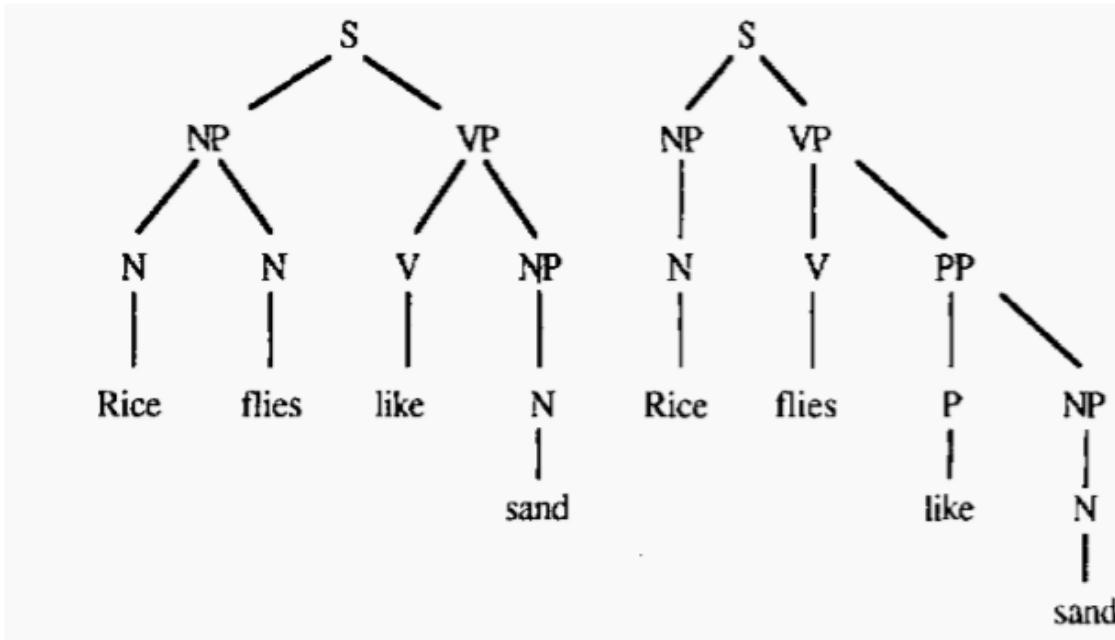


Figure 1.4 Two structural representations of *Rice flies like sand*.

CFG Rules

$S \rightarrow NP\ VP$
 $NP \rightarrow N\ N$
 $NP \rightarrow N$
 $VP \rightarrow V\ NP$
 $VP \rightarrow V\ PP$
 $PP \rightarrow P\ NP$

Lexicon

Rice : N
 Flies : N, V
 Like : V, P
 Sand : N

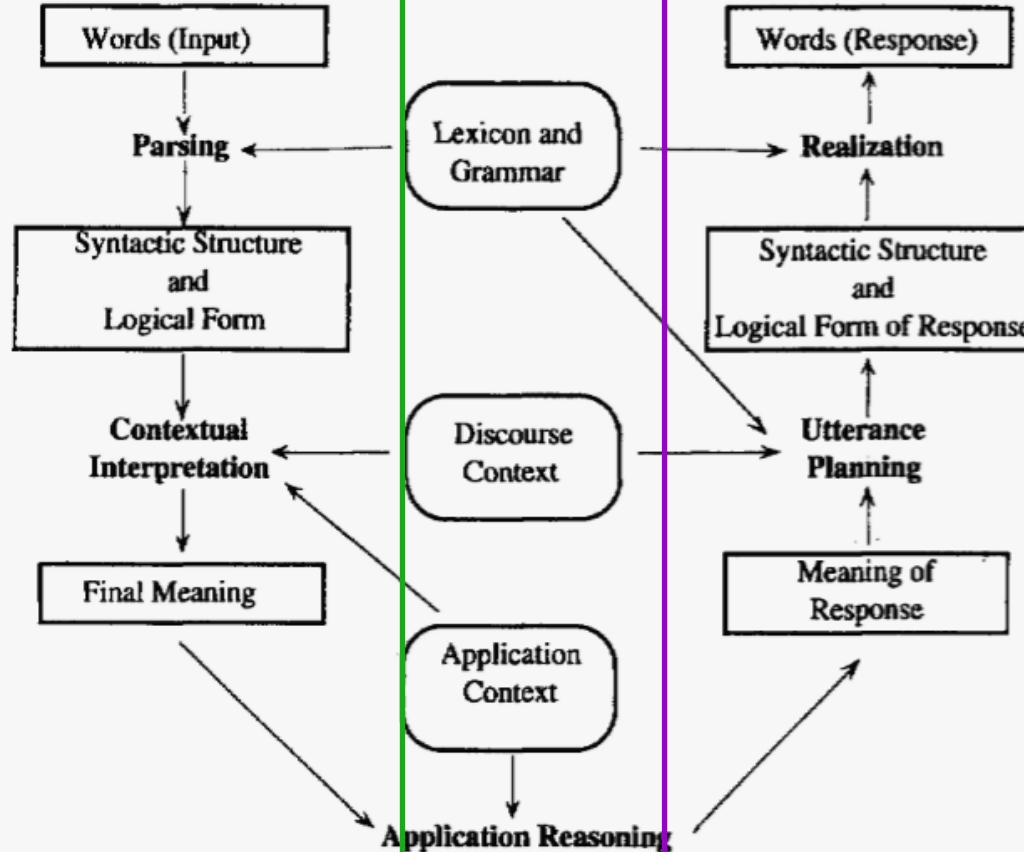
Figure 1.4 Two structural representations of "Rice flies like sand".

The Organization of Natural Language Processing Systems

Natural Language Understanding

Natural Language Generation

Allen 1995: Natural Language Understanding - Introduction



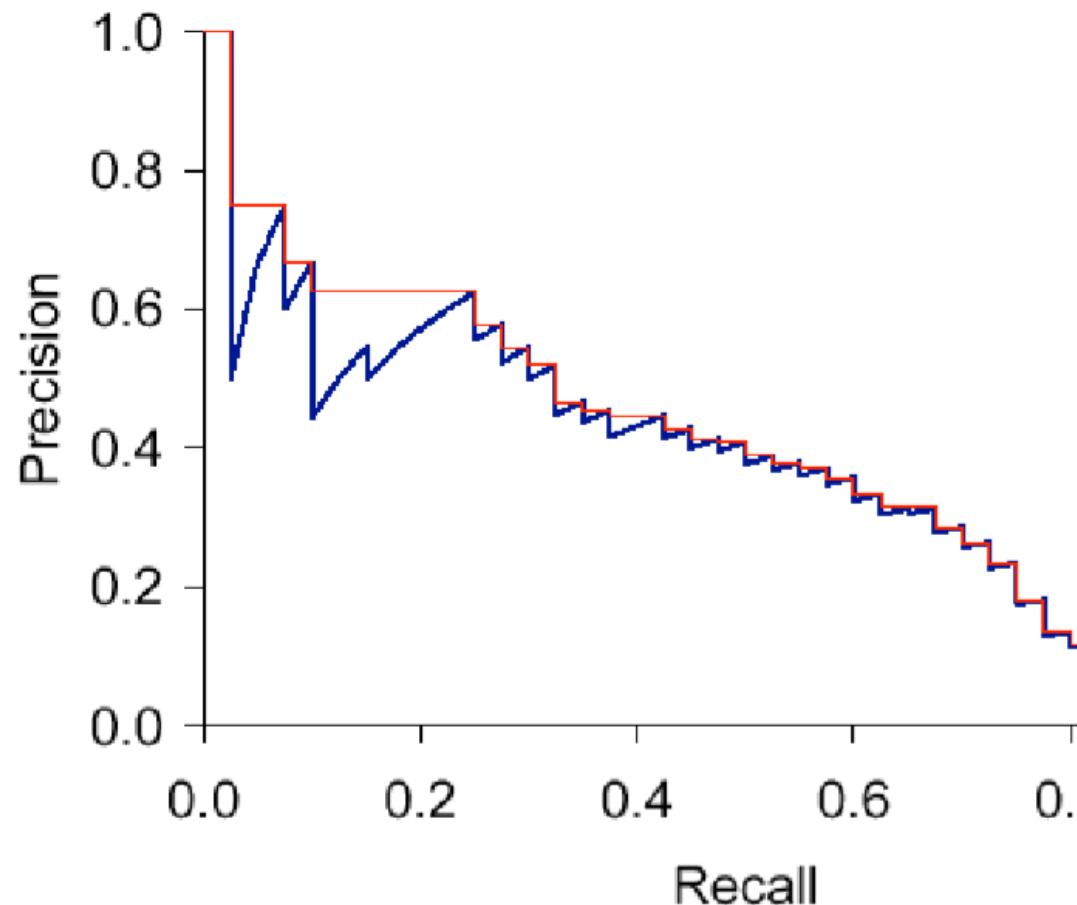
Evaluating Language Understanding Systems

- What metrics to use?
- How to deal with complex outputs like translations?
- Are the human judgments measuring something real? reliable?
- Is the sample of texts sufficiently representative?
- How reliable or certain are the results?

Contingency Table

		<i>gold standard labels</i>		
		gold positive	gold negative	
<i>system output labels</i>	system positive	true positive	false positive	precision = $\frac{tp}{tp+fp}$
	system negative	false negative	true negative	
		recall = $\frac{tp}{tp+fn}$		accuracy = $\frac{tp+tn}{tp+fp+tn+fn}$

Tradeoff between Precision and Recall



Some other evaluation measures

- Manual (the best!?):
 - SSER (subjective sentence error rate)
 - Correct/Incorrect
 - **Adequacy and Fluency** (5 or 7 point scales)
 - Error categorization
 - **Comparative ranking of translations**
- Testing in an application that uses MT as one sub-component
 - E.g., question answering from foreign language documents
 - May not test many aspects of the translation (e.g., cross-lingual IR)

Pre-processing text data



Cleaning (or pre-processing) the data typically consists of a number of steps:

- **Remove punctuation**
- **Tokenization**
- **Remove stop words**
- Lemmatize/Stem

Good References

<https://emerj.com/partner-content/nlp-current-applications-and-future-possibilities/>

<https://venturebeat.com/2019/04/05/why-nlp-will-be-big-in-2019/>

<https://www.nltk.org/book/>

Basic Text Processing

Regular Expressions

Regular expressions

- A formal language for specifying text strings
- How can we search for any of these?
 - woodchuck
 - woodchucks
 - Woodchuck
 - Woodchucks



Regular Expressions: Disjunctions

- Letters inside square brackets []

Pattern	Matches
[wW]oodchuck	Woodchuck, woodchuck
[1234567890]	Any digit

- Ranges [A-Z]

Pattern	Matches	
[A-Z]	An upper case letter	Drenched Blossoms
[a-z]	A lower case letter	my beans were impatient
[0-9]	A single digit	Chapter 1: Down the Rabbit Hole

Regular Expressions: Negation in Disjunction

- Negations [^Ss]
 - Carat means negation only when first in []

Pattern	Matches	
[^A-Z]	Not an upper case letter	O <u>y</u> fn pripetchik
[^Ss]	Neither 'S' nor 's'	<u>I</u> have no exquisite reason"
[^e^]	Neither e nor ^	Look <u>h</u> ere
a^b	The pattern a carat b	Look up <u>a^b</u> now

Regular Expressions: More Disjunction

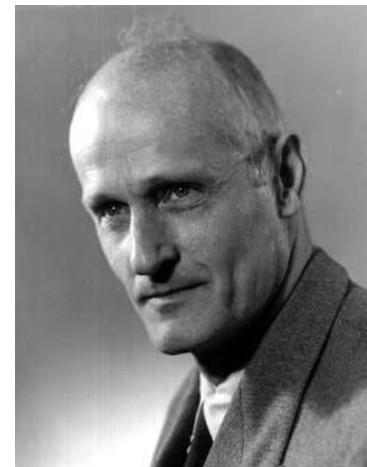
- Woodchucks is another name for groundhog!
- The pipe | for disjunction

Pattern	Matches
groundhog woodchuck	
yours mine	yours mine
a b c	= [abc]
[gG] roundhog [Ww]oodchuck	



Regular Expressions: ? * + .

Pattern	Matches
colou?r	Optional previous char
oo*h!	0 or more of previous char
o+h!	1 or more of previous char
baa+	
beg.n	



Stephen C Kleene

Kleene *, Kleene +

Regular Expressions: Anchors ^ \$

Pattern	Matches
<code>^ [A-Z]</code>	<u>P</u> alo Alto
<code>^ [^A-Za-z]</code>	<u>1</u> "Hello"
<code>\. \$</code>	The end <u>.</u>
<code>. \$</code>	The end <u>?</u> The end <u>!</u>

Example

- Find me all instances of the word “the” in a text.

the

Misses capitalized examples

[tT]he

Incorrectly returns other or theology

[^a-zA-Z] [tT]he [^a-zA-Z]

Errors

- The process we just went through was based on **fixing two kinds of errors**
 - Matching strings that we should not have matched (**there, then, other**)
 - **False positives (Type I)**
 - Not matching things that we should have matched (**The**)
 - **False negatives (Type II)**

Errors cont.

- In NLP we are always dealing with these kinds of errors.
- Reducing the error rate for an application often involves two antagonistic efforts:
 - Increasing accuracy or precision (minimizing false positives)
 - Increasing coverage or recall (minimizing false negatives).

Summary

- Regular expressions play a surprisingly large role
 - Sophisticated sequences of regular expressions are often the first model for any text processing task
- For many hard tasks, we use machine learning classifiers
 - But regular expressions are used as features in the classifiers
 - Can be very useful in capturing generalizations

Basic Text Processing

Regular Expressions

Basic Text Processing

Word tokenization

Text Normalization

- Every NLP task needs to do text normalization:
 1. Segmenting/tokenizing words in running text
 2. Normalizing word formats
 3. Segmenting sentences in running text

How many words?

- I do uh main-mainly business data processing
 - Fragments, filled pauses
- Seuss's **cat** in the hat is different from other **cats!**
 - **Lemma:** same stem, part of speech, rough word sense
 - **cat** and **cats** = same lemma
 - **Wordform:** the full inflected surface form
 - **cat** and **cats** = different wordforms

How many words?

they lay back on the San Francisco grass and looked at the stars and their

- **Type:** an element of the vocabulary.
- **Token:** an instance of that type in running text.
- How many?
 - 15 tokens (or 14)
 - 13 types (or 12) (or 11?)

How many words?

N = number of tokens

V = vocabulary = set of types

$|V|$ is the size of the vocabulary

Church and Gale (1990): $|V| > O(N^{1/2})$

	Tokens = N	Types = $ V $
Switchboard phone conversations	2.4 million	20 thousand
Shakespeare	884,000	31 thousand
Google N-grams	1 trillion	13 million

Simple Tokenization in UNIX

- (Inspired by Ken Church's UNIX for Poets.)
- Given a text file, output the word tokens and their frequencies

```
tr -sc 'A-Za-z' '\n' < shakes.txt      Change all non-alpha to newlines  
| sort          Sort in alphabetical order  
| uniq -c       Merge and count each type
```

1945 A	25 Aaron
72 AARON	6 Abate
19 ABBESS	1 Abates
5 ABBOT	5 Abbess
.... .	6 Abbey
	3 Abbot

The first step: tokenizing

```
tr -sc 'A-Za-z' '\n' < shakes.txt | head
```

THE
SONNETS
by
William
Shakespeare
From
fairest
creatures
We
...

The second step: sorting

```
tr -sc 'A-Za-z' '\n' < shakes.txt | sort | head
```

A

A

A

A

A

A

A

A

A

A

...

More counting

- Merging upper and lower case

```
tr 'A-Z' 'a-z' < shakes.txt | tr -sc 'A-Za-z' '\n' | sort | uniq -c
```

- Sorting the counts

```
tr 'A-Z' 'a-z' < shakes.txt | tr -sc 'A-Za-z' '\n' | sort | uniq -c | sort -n -r
```

23243 the

22225 i

18618 and

16339 to

15687 of

12780 a

12163 you

10839 my

10005 in

8954 d

What happened here?

Issues in Tokenization

- Finland's capital → Finland Finlands Finland's ?
- what're, I'm, isn't → What are, I am, is not
- Hewlett-Packard → Hewlett Packard ?
- state-of-the-art → state of the art ?
- Lowercase → lower-case lowercase lower case ?
- San Francisco → one token or two?
- m.p.h., PhD. → ??

Tokenization: language issues

- French
 - *L'ensemble* → one token or two?
 - *L* ? *L'* ? *Le* ?
 - Want *L'ensemble* to match with *un ensemble*
- German noun compounds are not segmented
 - *Lebensversicherungsgesellschaftsangestellter*
 - ‘life insurance company employee’
 - German information retrieval needs **compound splitter**

Tokenization: language issues

- Chinese and Japanese no spaces between words:
 - 莎拉波娃现在居住在美国东南部的佛罗里达。
 - 莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达
 - Sharapova now lives in US southeastern Florida
- Further complicated in Japanese, with multiple alphabets intermingled
 - Dates/amounts in multiple formats

フォーチュン500社は情報不足のため時間あた\$500K(約6,000万円)

The diagram shows a line of Japanese text with various characters highlighted by boxes and arrows pointing to labels below. The text is: "フォーチュン500社は情報不足のため時間あた\$500K(約6,000万円)".

- A box highlights the first four characters "フォーチュン". An arrow points from this box to a pink box labeled "Katakana".
- A box highlights the next three characters "500社". An arrow points from this box to a pink box labeled "Hiragana".
- A box highlights the characters "情報不足". An arrow points from this box to a pink box labeled "Kanji".
- A box highlights the characters "ため時間". An arrow points from this box to a pink box labeled "Romaji".
- A box highlights the characters "\$500K". An arrow points from this box to a pink box labeled "Romaji".
- A box highlights the characters "(約6,000万円)". An arrow points from this box to a pink box labeled "Romaji".

End-user can express query entirely in hiragana!

Word Tokenization in Chinese

- Also called **Word Segmentation**
- Chinese words are composed of characters
 - Characters are generally 1 syllable and 1 morpheme.
 - Average word is 2.4 characters long.
- Standard baseline segmentation algorithm:
 - Maximum Matching (also called Greedy)

Maximum Matching Word Segmentation Algorithm

- Given a wordlist of Chinese, and a string.
- 1) Start a pointer at the beginning of the string
 - 2) Find the longest word in dictionary that matches the string starting at pointer
 - 3) Move the pointer over the word in string
 - 4) Go to 2

Max-match segmentation illustration

- The cat in the hat the cat in the hat
- The table down there the table down there
- Doesn't generally work in English!
- But works astonishingly well in Chinese
 - 莎拉波娃现在居住在美国东南部的佛罗里达。
 - 莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达
- Modern probabilistic segmentation algorithms even better

Basic Text Processing

Word tokenization

Basic Text Processing

**Word Normalization and
Stemming**

Normalization

- Need to “normalize” terms
 - Information Retrieval: indexed text & query terms must have same form.
 - We want to match *U.S.A.* and *USA*
- We implicitly define equivalence classes of terms
 - e.g., deleting periods in a term
- Alternative: asymmetric expansion:
 - Enter: *window* Search: *window, windows*
 - Enter: *windows* Search: *Windows, windows, window*
 - Enter: *Windows* Search: *Windows*
- Potentially more powerful, but less efficient

Case folding

- Applications like IR: reduce all letters to lower case
 - Since users tend to use lower case
 - Possible exception: upper case in mid-sentence?
 - e.g., *General Motors*
 - *Fed* vs. *fed*
 - *SAIL* vs. *sail*
- For sentiment analysis, MT, Information extraction
 - Case is helpful (*US* versus *us* is important)

Lemmatization

- Reduce inflections or variant forms to base form
 - *am, are, is* → *be*
 - *car, cars, car's, cars'* → *car*
- *the boy's cars are different colors* → *the boy car be different color*
- Lemmatization: have to find correct dictionary headword form
- Machine translation
 - Spanish **quiero** ('I want'), **quieres** ('you want') same lemma as **querer** 'want'

Morphology

- **Morphemes:**
 - The small meaningful units that make up words
 - **Stems:** The core meaning-bearing units
 - **Affixes:** Bits and pieces that adhere to stems
 - Often with grammatical functions

Stemming

- Reduce terms to their stems in information retrieval
- *Stemming* is crude chopping of affixes
 - language dependent
 - e.g., *automate(s)*, *automatic*, *automation* all reduced to *automat.*

for example compressed
and compression are both
accepted as equivalent to
compress.



for exempl compress and
compress ar both accept
as equival to compress

Porter's algorithm

The most common English stemmer

Step 1a

sses	→ ss	caresses	→ caress
ies	→ i	ponies	→ poni
ss	→ ss	caress	→ caress
s	→ Ø	cats	→ cat

Step 1b

(*v*)ing	→ Ø	walking	→ walk
		sing	→ sing
(*v*)ed	→ Ø	plastered	→ plaster
...			

Step 2 (for long stems)

ational	→ ate	relational	→ relate
izer	→ ize	digitizer	→ digitize
ator	→ ate	operator	→ operate
...			

Step 3 (for longer stems)

al	→ Ø	revival	→ reviv
able	→ Ø	adjustable	→ adjust
ate	→ Ø	activate	→ activ
...			

Viewing morphology in a corpus

Why only strip –ing if there is a vowel?

(*v*) ing → Ø walking → walk
sing → sing

Viewing morphology in a corpus

Why only strip -ing if there is a vowel?

(*v*) ing → ø walking → walk
sing → sing

```
tr -sc 'A-Za-z' '\n' < shakes.txt | grep 'ing$' | sort | uniq -c | sort -nr
1312 King
548 being
541 nothing
388 king
375 bring
358 thing
307 ring
152 something
145 coming
130 morning
122 having
120 living
117 loving
116 Being
102 going
```



```
tr -sc 'A-Za-z' '\n' < shakes.txt | grep '[aeiou].*ing$' | sort | uniq -c | sort -nr
```

Dealing with complex morphology is sometimes necessary

- Some languages require complex morpheme segmentation
 - Turkish
 - **Uygarlastiramadiklarimizdanmissinizcasina**
 - `(behaving) as if you are among those whom we could not civilize'
 - **Uygar** `civilized' + **las** `become'
 - + **tir** `cause' + **ama** `not able'
 - + **dik** `past' + **lar** 'plural'
 - + **imiz** 'p1pl' + **dan** 'abl'
 - + **mis** 'past' + **siniz** '2pl' + **casina** 'as if'

Basic Text Processing

**Word Normalization and
Stemming**

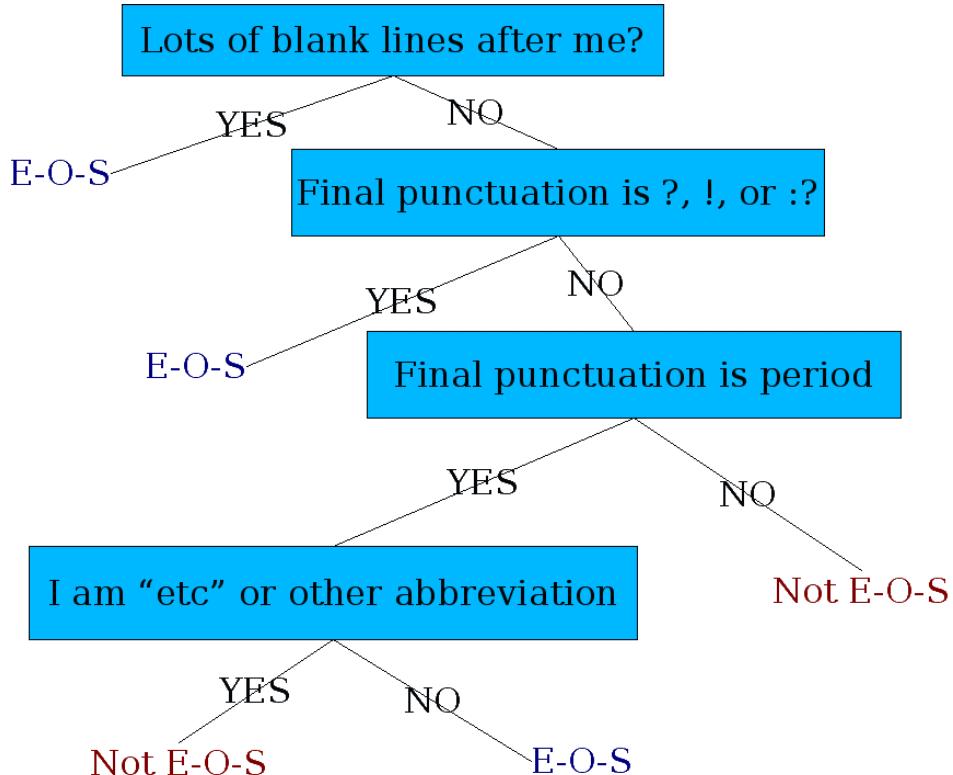
Basic Text Processing

Sentence Segmentation
and Decision Trees

Sentence Segmentation

- !, ? are relatively unambiguous
- Period “.” is quite ambiguous
 - Sentence boundary
 - Abbreviations like Inc. or Dr.
 - Numbers like .02% or 4.3
- Build a binary classifier
 - Looks at a “.”
 - Decides EndOfSentence/NotEndOfSentence
 - Classifiers: hand-written rules, regular expressions, or machine--learning

Determining if a word is end-of-sentence: a Decision Tree



More sophisticated decision tree features

- Case of word with “.”: Upper, Lower, Cap, Number
- Case of word after “.”: Upper, Lower, Cap, Number
- Numeric features
 - Length of word with “.”
 - Probability(word with “.” occurs at end-of-s)
 - Probability(word after “.” occurs at beginning-of-s)

Implementing Decision Trees

- A decision tree is just an if–then–else statement
- The interesting research is choosing the features
- Setting up the structure is often too hard to do by hand
 - Hand–building only possible for very simple features, domains
 - For numeric features, it’s too hard to pick each threshold
 - Instead, structure usually learned by machine learning from a training corpus

Decision Trees and other classifiers

- We can think of the questions in a decision tree
- As features that could be exploited by any kind of classifier
 - Logistic regression
 - SVM
 - Neural Nets
 - etc.

Basic Text Processing

Sentence Segmentation
and Decision Trees

Naïve Bayes for Text Classification

Text classification: Naïve Bayes Text Classification

- Today:
 - Introduction to Text Classification
 - Probabilistic Language Models →
 - Naïve Bayes text categorization →



Is this spam? ↗

From: "" <takworlld@hotmail.com>

Subject: real estate is the only way... gem oalvgkay

Text → Model

```
graph LR; A[Text] --> B[Model]; B --> C["prob(Spam)"]; B --> D["prob(Ham)"]
```

Anyone can buy real estate with no money down

Stop paying rent TODAY !

SPAM $\Leftarrow \text{prob(Spam)} > \text{prob(Ham)}$

There is no need to spend hundreds or even thousands for similar courses

I am 22 years old and I have already purchased 6 properties using the methods outlined in this truly INCREDIBLE ebook.

Change your life NOW !

=====
Click Below to order:

<http://www.wholesaledaily.com/sales/nmd.htm>

Categorization/Classification

- Given:

$$\begin{aligned} x &\rightarrow c_1 \\ x' &\rightarrow c_2 \end{aligned}$$

- A description of an instance, $x \in X$, where X is the *instance language* or *instance space*.

- Issue: how to represent text documents.

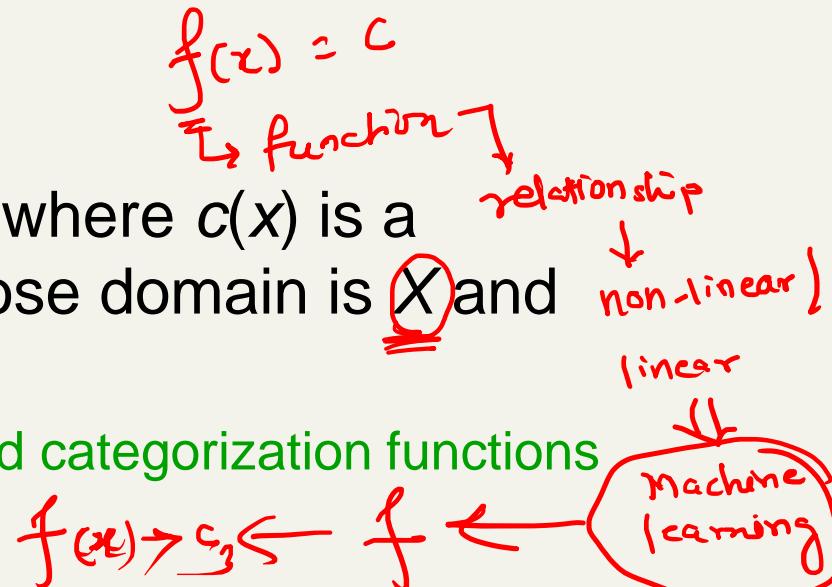
- A fixed set of categories: $\underline{\text{class}}$

$$\underline{C} = \{c_1, c_2, \dots, c_n\}$$

- Determine:

- The category of x : $c(x) \in C$, where $c(x)$ is a *categorization function* whose domain is \underline{X} and whose range is \underline{C} .

- We want to know how to build categorization functions ("classifiers").

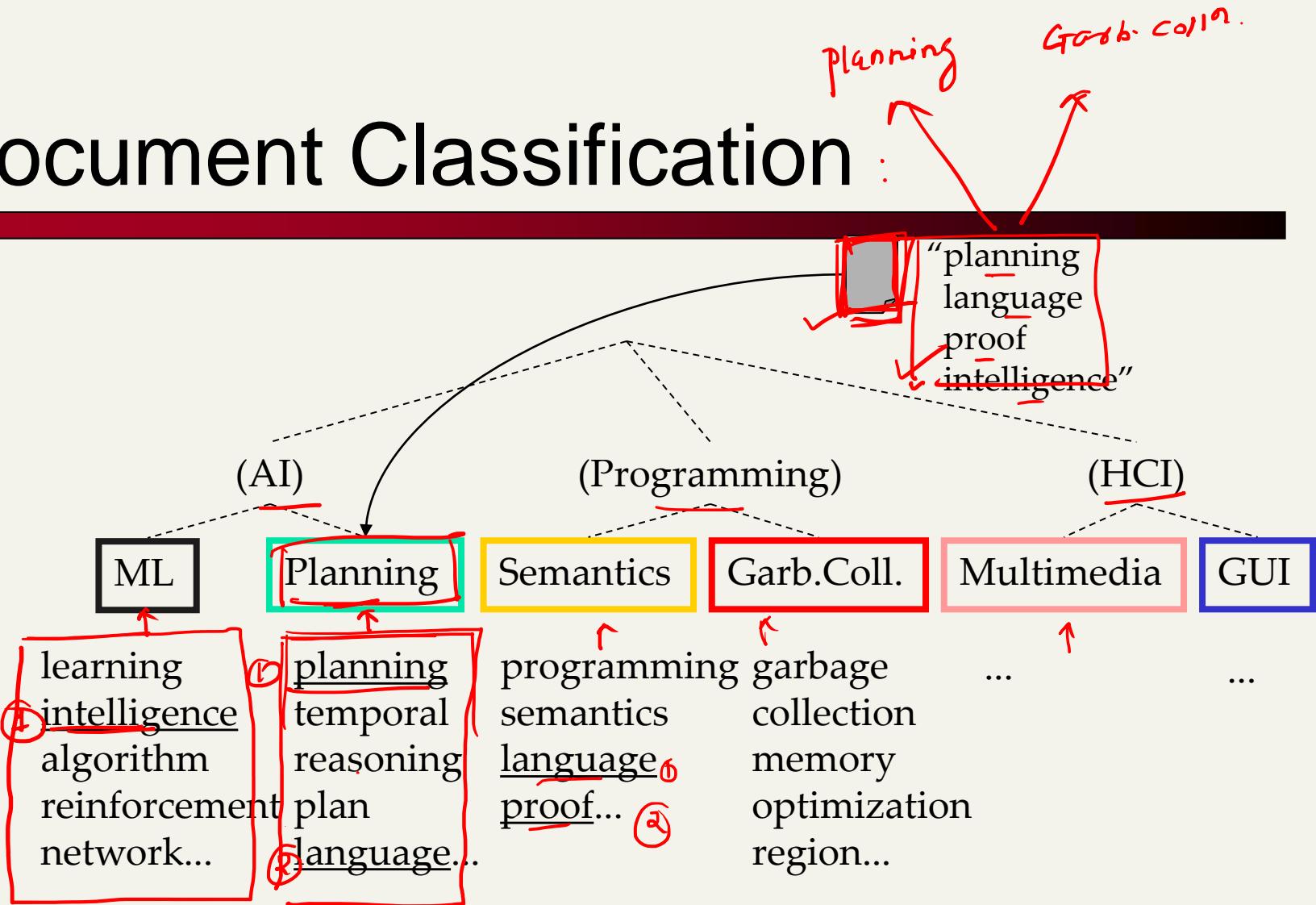


Document Classification

Test Data:

Classes:

Training Data:



(Note: in real life there is often a hierarchy, not present in the above problem statement; and you get papers on ML approaches to Garb. Coll.)

Text Categorization Examples

Assign labels to each document or web-page:

- Labels are most often topics such as Yahoo-categories
e.g., "finance," "sports," "news>world>asia>business"
- Labels may be genres
e.g., "editorials" "movie-reviews" "news"
- Labels may be opinion
e.g., "like," "hate," "neutral"
- Labels may be domain-specific binary
 - e.g., "interesting-to-me" : "not-interesting-to-me"
 - e.g., "spam" : "not-spam"
 - e.g., "contains adult language" : "doesn't"

Classification Methods (1)

- Manual classification →
 - Used by Yahoo!, Looksmart, about.com, ODP, Medline
 - Very accurate when job is done by experts
 - Consistent when the problem size and team is small
 - Difficult and expensive to scale

Classification Methods (2)

- Automatic document classification
 - Hand-coded rule-based systems
 - One technique used by CS dept's spam filter, Reuters, CIA, Verity, ...
 - E.g., assign category if document contains a given boolean combination of words
 - Standing queries: Commercial systems have complex query languages (everything in IR query languages + accumulators)
 - Accuracy is often very high if a rule has been carefully refined over time by a subject expert
 - Building and maintaining these rules is expensive

Classification Methods (3)

- Supervised learning of a document-label assignment function
 - Many systems partly rely on machine learning (Autonomy, MSN, Verity, Enkata, Yahoo!, ...)
 - k-Nearest Neighbors (simple, powerful)
 - Naive Bayes (simple, common method)
 - Support-vector machines (new, more powerful)
 - ... plus many other methods
 - No free lunch: requires hand-classified training data
 - But data can be built up (and refined) by amateurs
- Note that many commercial systems use a mixture of methods

(1) frequentist approach - count classification $\rightarrow C_1$
(2) probabilistic $\rightarrow p = \text{prob}$

Bayesian Methods

Generative \rightarrow how approx.
data is generated



frequentist
 \rightarrow neutral

- Our focus this lecture
- Learning and classification methods based on probability theory.
- Bayes theorem plays a critical role in probabilistic learning and classification.
- Build a *generative model* that approximates how data is produced
- Uses prior probability of each category given no information about an item.
- Categorization produces a *posterior* probability distribution over the possible categories given a description of an item.

$$\begin{aligned} 1000 &\xrightarrow{C_1=500} \\ \times &\xrightarrow{C_2=250} \\ &\xrightarrow{C_3=250} \end{aligned}$$

Bayes' Rule

$$P(\underline{C}, \underline{X}) = P(C | X)P(X) = P(X | C)P(C)$$

$$P(\underline{C} | \underline{X}) = \frac{P(\underline{X} | \underline{C})P(\underline{C})}{P(\underline{X})}$$

$$C = \{c_1, c_2, c_3, \dots, c_n\}$$

Maximum a posteriori Hypothesis

$$h_{MAP} \equiv \operatorname{argmax}_{h \in H} P(h | D)$$

$D \in c_1 \rightarrow P$

$D \in c_2 \rightarrow P$

\vdots

$D \in c_3 \rightarrow P$

$$= \operatorname{argmax}_{h \in H} \frac{P(D | h)P(h)}{P(D)}$$
$$= \operatorname{argmax}_{h \in H} P(D | h)P(h)$$

As $P(D)$ is constant

Maximum likelihood Hypothesis

If all hypotheses are a priori equally likely, we only need to consider the $P(D/h)$ term:



$$h_{ML} \equiv \operatorname{argmax}_{h \in H} P(D | h)$$

Naive Bayes Classifiers

Task: Classify a new instance \underline{D} based on a tuple of attribute values $\underline{D} = \langle \underline{x}_1, \underline{x}_2, \dots, \underline{x}_n \rangle$ into one of the classes $c_j \in \underline{C}$

$$\begin{aligned} c_{MAP} &= \underset{c_j \in C}{\operatorname{argmax}} P(\underline{c}_j \mid \underline{x}_1, \underline{x}_2, \dots, \underline{x}_n) \\ &= \underset{c_j \in C}{\operatorname{argmax}} \frac{P(\underline{x}_1, \underline{x}_2, \dots, \underline{x}_n \mid c_j) P(c_j)}{P(\underline{x}_1, \underline{x}_2, \dots, \underline{x}_n)} \\ &= \underset{c_j \in C}{\operatorname{argmax}} P(\underline{x}_1, \underline{x}_2, \dots, \underline{x}_n \mid c_j) P(c_j) \end{aligned}$$

Joint probability is equal to product of the independent prob.

Naïve Bayes Classifier:

$P(x_1, x_2, x_3, \dots, x_{1000}) = \frac{P(x_1)}{1000} \cdot \frac{P(x_2)}{1000} \cdot \frac{P(x_3)}{1000} \cdots \frac{P(x_{1000})}{1000}$

Naïve Bayes Assumption

- $P(c_j)$ 1000 documents $\begin{cases} c_1 = 500 \\ c_2 = 250 \\ c_3 = 250 \end{cases} \rightarrow P(c_1) = \frac{500}{1000} = \frac{1}{2} = 0.5$
 $P(c_2) = \frac{250}{1000} = 0.25$
 $P(c_3) = \frac{250}{1000} = 0.25$
 - Can be estimated from the frequency of classes in the training examples.
- $P(x_1, x_2, \dots, x_n | c_j)$ $x_1, x_2, \dots, x_n : D_i$ $x_2, x_1, \dots, x_n : D_2$ $c_j \in D = 1000 \text{ features/words}$ $P(x_1, \dots, x_{1000} | c_j)$
 - $O(|X|^n | C)$ parameters
 - Could only be estimated if a very, very large number of training examples was available.

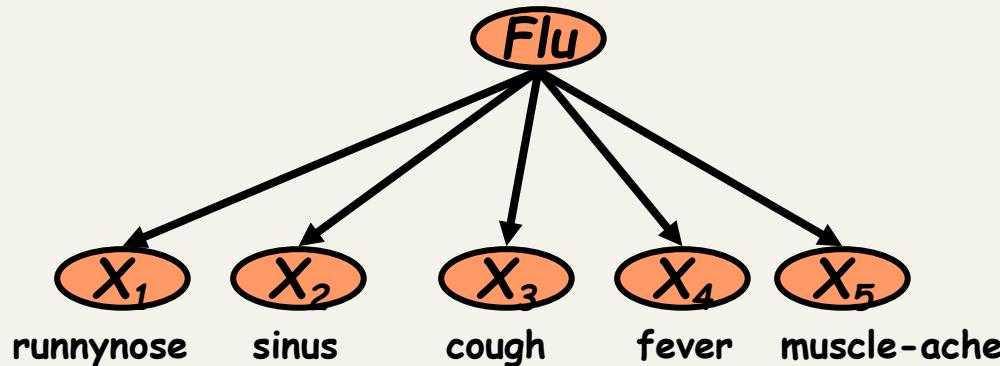
→ Naïve Bayes Conditional Independence Assumption:

- Assume that the probability of observing the conjunction of attributes is equal to the product of the individual probabilities $P(x_i | c_j)$.

very very huge- | (← (1000))

$\begin{cases} x_1, x_2, \dots, x_{1000} \\ x_1, x_3, x_2, \dots, x_{1000} \\ x_1, x_3, x_4, x_5, \dots, x_{1000} \\ x_2, x_1, x_3, x_4, \dots, x_{1000} \end{cases}$

The Naïve Bayes Classifier

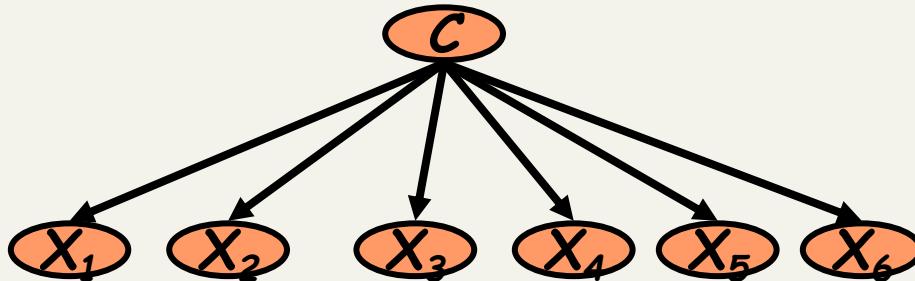


- **Conditional Independence Assumption:** features detect term presence and are independent of each other given the class:

$$P(\underline{X_1, \dots, X_5} | C) = P(\underline{X_1} | C) \cdot P(\underline{X_2} | C) \cdot \dots \cdot P(\underline{X_5} | C)$$

- This model is appropriate for binary variables
 - Multivariate binomial model

Learning the Model



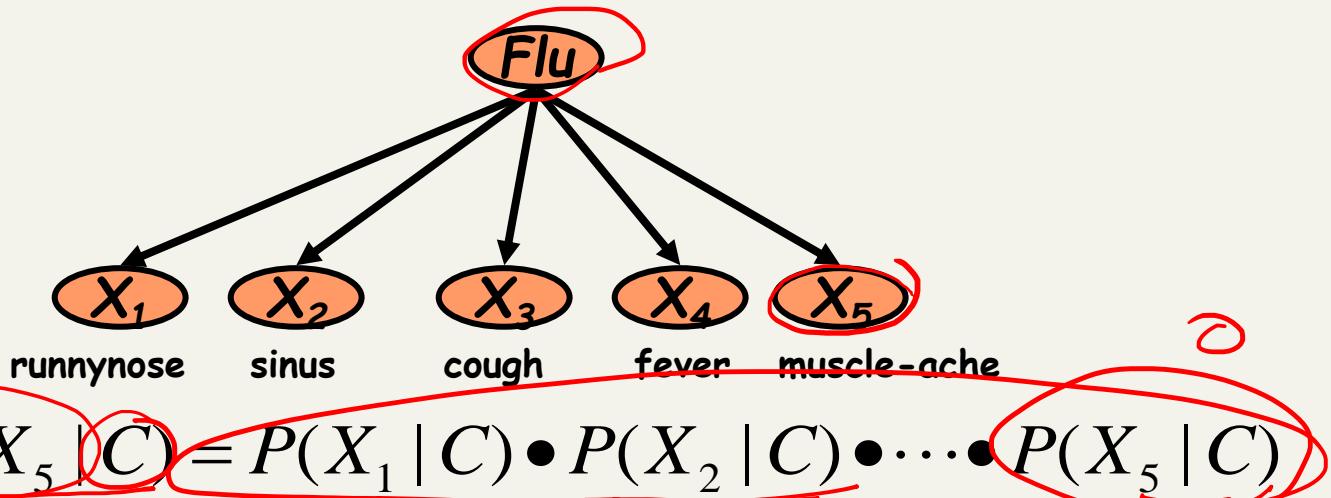
- First attempt: maximum likelihood estimates
 - simply use the frequencies in the data

$$\hat{P}(c_j) = \frac{\underline{N(C=c_j)}}{\underline{N}} =$$

$$\hat{P}(\underline{x_i | c_j}) = \frac{\underline{N(X_i=x_i, C=c_j)}}{\underline{N(C=c_j)}}$$

(x_i, c_j) comes together
c_j is present in the dataset

Problem with Max Likelihood



- What if we have seen no training cases where patient had no flu and muscle aches?

$$\hat{P}(X_5 = t | C = nf) = \frac{N(X_5 = t, C = nf)}{N(C = nf)} = 0$$

- Zero probabilities cannot be conditioned away, no matter the other evidence!

$$\ell = \arg \max_c \hat{P}(c) \prod_i \hat{P}(x_i | c)$$

Smoothing to Avoid Overfitting

$$\hat{P}(x_i | c_j) = \frac{N(X_i = \underline{x}_j, C = c_j) + 1}{N(C = c_j) + k}$$

of values of X_i

- Somewhat more subtle version

overall fraction in
data where $X_i=x_{i,k}$

$$\hat{P}(x_{i,k} | c_j) = \frac{N(X_i = x_{i,k}, C = c_j) + mp_{i,k}}{N(C = c_j) + m}$$

extent of
“smoothing”

unigram (1 word)
bigram (2 words)
n-gram (n words)

Stochastic Language Models

- Models *probability* of generating strings (each word in turn) in the language (commonly all strings over Σ). E.g., unigram model

Model M

0.2	the
0.1	a
0.01	man
0.01	woman
0.03	said
0.02	likes
...	

the man likes the woman

0.2 0.01 0.02 0.2 0.01

multiply

$$P(s | M) = \underline{\underline{0.00000008}}$$

Stochastic Language Models

- Model *probability* of generating any string

Model M1

0.2	the
0.01	class
0.0001	sayst
0.0001	pleaseth
0.0001	yon
0.0005	maiden
0.01	woman

Model M2

0.2	the
0.0001	class
0.03	sayst
0.02	pleaseth
0.1	yon
0.01	maiden
0.0001	woman

the class pleaseth yon maiden

0.2 0.01 0.0001 0.0001 0.0005
0.2 0.0001 0.02 0.1 0.01

$$\underline{P(s|M2)} > \underline{P(s|M1)}$$

Unigram and higher-order models

$$P(\text{what is the weather})$$

\Rightarrow language model \rightarrow language

$$= P(\text{ }) P(\text{ }|\text{ }) P(\text{ }|\text{ }) P(\text{ }|\text{ }) P(\text{ }|\text{ })$$

- Unigram Language Models

$$P(\text{ }) P(\text{ }) P(\text{ }) P(\text{ })$$

Easy.
Effective!

- Bigram (generally, n -gram) Language Models

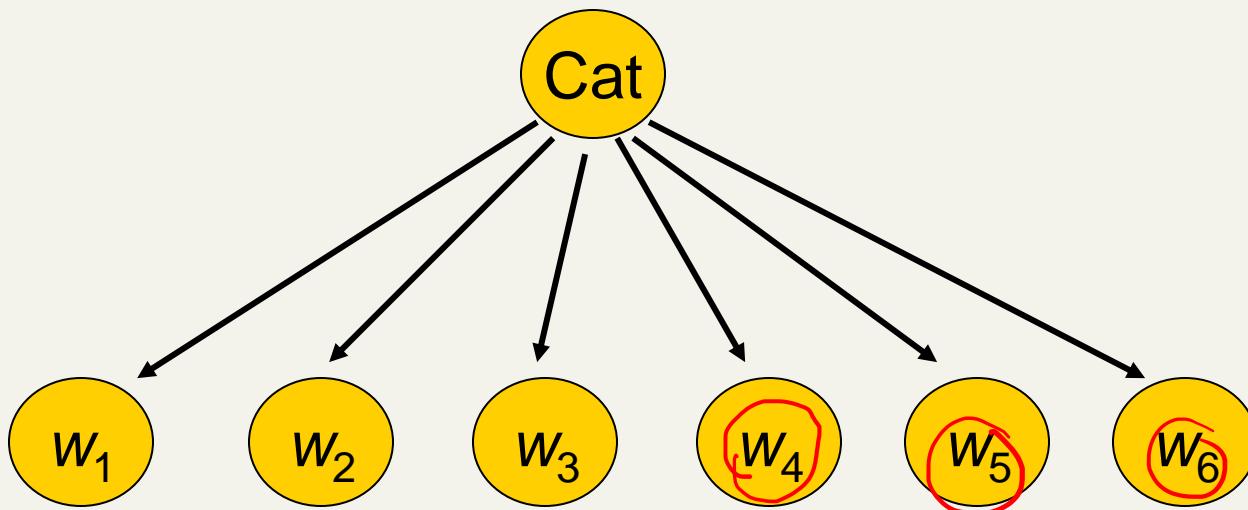
$$P(\text{ }) P(\text{ }|\text{ }) P(\text{ }|\text{ }) P(\text{ }|\text{ }) P(\text{ }|\text{ })$$

- Other Language Models

- Grammar-based models (PCFGs), etc.

- Probably not the first thing to try in IR

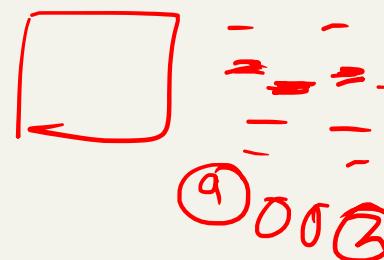
Naïve Bayes via a class conditional language model = multinomial NB



- Effectively, the probability of each class is done as a class-specific unigram language model

Using Multinomial Naive Bayes Classifiers to Classify Text: Basic method

- Attributes are text positions, values are words.

$$\begin{aligned} \underline{\underline{c_{NB}}} &= \operatorname{argmax}_{c_j \in C} P(c_j) \prod_i P(\underline{x_i} | \underline{c_j}) \\ &= \operatorname{argmax}_{c_j \in C} P(c_j) P(x_1 = "our" | c_j) \cdots P(x_n = "text" | c_j) \end{aligned}$$


- Still too many possibilities
- Assume that classification is *independent* of the positions of the words
 - Use same parameters for each position
 - Result is bag of words model (over tokens not types)

Naïve Bayes: Learning :-

- From training corpus, extract Vocabulary $\xrightarrow{\text{(set of unique words)}}$
- Calculate required $P(c_j)$ and $P(x_k | c_j)$ terms
 - For each c_j in C do
 - $docs_{c_j} \leftarrow$ subset of documents for which the target class is c_j
 - $P(c_j) \leftarrow \frac{|docs_{c_j}|}{\text{total # documents |}}$
 - $Text_j \leftarrow$ single document containing all $docs_{c_j}$
 - for each word x_k in Vocabulary
 - $n_k \leftarrow$ number of occurrences of x_k in $Text_j$
 - $P(x_k | c_j) \leftarrow \frac{n_k + \alpha}{n + \alpha |Vocabulary|}$

Naïve Bayes: Classifying

- positions \leftarrow all word positions in current document which contain tokens found in *Vocabulary*
- Return c_{NB} , where



$$c_{NB} = \operatorname{argmax}_{c_j \in C} P(c_j) \prod_{i \in positions} P(x_i | c_j)$$

Parameter estimation

- Binomial model:
 $\hat{P}(X_w = t | c_j) = \frac{P(x) \cdot P(x') \cdot P(x'')}$
 ~~$\hat{P}(X_w = t | c_j)$~~ fraction of documents of topic c_j
in which word w appears
 $D = \{x_1, x_2, \dots, x_n\}$
- Multinomial model:
 $\hat{P}(X_i = w | c_j) = \frac{P(x_1, x_2, \dots, x_n | c)}$
 ~~$\hat{P}(X_i = w | c_j)$~~ fraction of times in which
word w appears
across all documents of topic c_j
 - Can create a mega-document for topic j by concatenating all documents in this topic
 - Use frequency of w in mega-document

NB example

- Given: 4 documents
 - D1 (sports): China soccer
 - D2 (sports): Japan baseball
 - D3 (politics): China trade
 - D4 (politics): Japan Japan exports
- Classify:
 - D5: soccer → soccer sport
 - D6: Japan → polity
- Use
 - Add-one smoothing
 - Multinomial model
 - Multivariate binomial model

Feature Selection: Why?

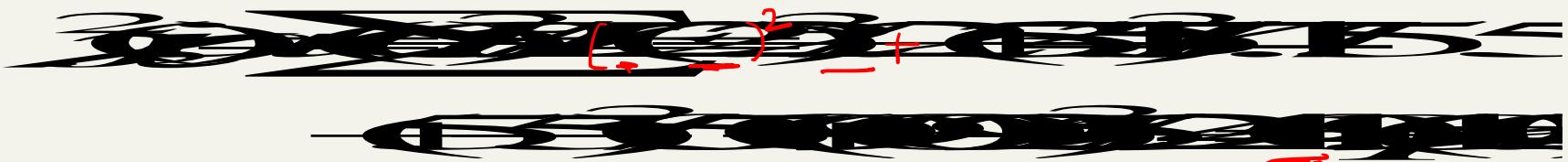
- Text collections have a large number of features
 - 10,000 – 1,000,000 unique words ... and more
- May make using a particular classifier feasible
 - Some classifiers can't deal with 100,000 of features
- Reduces training time
 - Training time for some methods is quadratic or worse in the number of features
- Can improve generalization (performance)
 - Eliminates noise features
 - Avoids overfitting

Feature selection: how?

- Two idea:
 - Hypothesis testing statistics: ✓
 - Are we confident that the value of one categorical variable is associated with the value of another
 - Chi-square test
 - Information theory: ✓
 - How much information does the value of one categorical variable give you about the value of another
 - Mutual information
- They're similar, but χ^2 measures confidence in association, (based on available statistics), while MI measures extent of association (assuming perfect knowledge of probabilities)

χ^2 statistic (CHI)

- χ^2 is interested in $(f_o - f_e)^2/f_e$ summed over all table entries: is the observed number what you'd expect given the marginals?



- The null hypothesis is rejected with confidence .999,
- since 12.9 > 10.83 (the value for .999 confidence).

	$Term = jaguar$	$Term \neq jaguar$
$Class = auto$	2 (0.25)	500 (502)
$Class \neq auto$	3 (4.75)	9500 (9498)

expected: f_e

observed: f_o

Feature selection via Mutual Information

$$\frac{\frac{300}{(50)(50)}}{300} = \frac{100}{(200)+(200)} = 0.5$$

- In training set, choose k words which best discriminate (give most info on) the categories.
- The Mutual Information between a word, class is:

$$I(w, c) = \sum_{e_w \in \{0,1\}} \sum_{e_c \in \{0,1\}} p(e_w, e_c) \log \frac{p(e_w, e_c)}{p(e_w)p(e_c)}$$

- For each word w and each category c



$MI = \text{high}$

Information entropy = $-\log P(x)$

Entropy = $-\sum P(x) \log P(x)$

$\text{MI} = \text{no mutual information}$

Feature Selection

- Mutual Information
 - Clear information-theoretic interpretation
 - May select rare uninformative terms
- Chi-square
 - Statistical foundation
 - May select very slightly informative frequent terms that are not very useful for classification
- Just use the commonest terms?
 - No particular foundation
 - In practice, this is often 90% as good

Feature selection for NB

- In general feature selection is *necessary* for binomial NB.
- Otherwise you suffer from noise, multi-counting
- “Feature selection” really means something different for multinomial NB. It means dictionary truncation
 - The multinomial NB model only has 1 feature
- This “feature selection” normally isn’t needed for multinomial NB, but may help a fraction with quantities that are badly estimated

SpamAssassin

- Naïve Bayes has found a home for spam filtering
 - Graham's *A Plan for Spam*
 - And its mutant offspring...
 - Naive Bayes-like classifier with weird parameter estimation
 - Widely used in spam filters
 - Classic Naive Bayes superior when appropriately used
 - According to David D. Lewis
- Many email filters use NB classifiers
 - But also many other things: black hole lists, etc.

Violation of NB Assumptions

- Conditional independence
- “Positional independence”



Questions If Any?

Naïve Bayes Posterior Probabilities

- Classification results of naïve Bayes (the class with maximum posterior probability) are usually fairly accurate.
- However, due to the inadequacy of the conditional independence assumption, the actual posterior-probability numerical estimates are not.
 - Output probabilities are generally very close to 0 or 1.

Logistic Regression

Background: Generative and
Discriminative Classifiers

Logistic Regression

Important analytic tool in natural and social sciences

Baseline supervised machine learning tool for classification

Is also the foundation of neural networks

Generative and Discriminative Classifiers

Naive Bayes is a **generative** classifier

by contrast:

Logistic regression is a **discriminative** classifier

Generative and Discriminative Classifiers

Suppose we're distinguishing cat from dog images



imagenet



imagenet

Generative Classifier:

- Build a model of what's in a cat image
 - Knows about whiskers, ears, eyes
 - Assigns a probability to any image:
 - how cat-y is this image?



Also build a model for dog images

Now given a new image:

Run both models and see which one fits better

Discriminative Classifier

Just try to distinguish dogs from cats



Oh look, dogs have collars!
Let's ignore everything else

Finding the correct class c from a document d in Generative vs Discriminative Classifiers

Naive Bayes

$$\hat{c} = \operatorname{argmax}_{c \in C} \underbrace{P(d|c)}_{\text{likelihood}} \underbrace{P(c)}_{\text{prior}}$$

Logistic Regression

$$\hat{c} = \operatorname{argmax}_{c \in C} P(c/d) \quad \text{posterior}$$

Components of a probabilistic machine learning classifier

Given m input/output pairs $(x^{(i)}, y^{(i)})$:

1. A **feature representation** of the input. For each input observation $x^{(i)}$, a vector of features $[x_1, x_2, \dots, x_n]$. Feature j for input $x^{(i)}$ is x_j , more completely $x_j^{(i)}$, or sometimes $f_j(x)$.
2. A **classification function** that computes \hat{y} , the estimated class, via $p(y|x)$, like the **sigmoid** or **softmax** functions.
3. An objective function for learning, like **cross-entropy loss**.
4. An algorithm for optimizing the objective function: **stochastic gradient descent**.

The two phases of logistic regression

Training: we learn weights w and b using **stochastic gradient descent** and **cross-entropy loss**.

Test: Given a test example x we compute $p(y|x)$ using learned weights w and b , and return whichever label ($y = 1$ or $y = 0$) is higher probability

Logistic Regression

Background: Generative and
Discriminative Classifiers

Classification in Logistic Regression

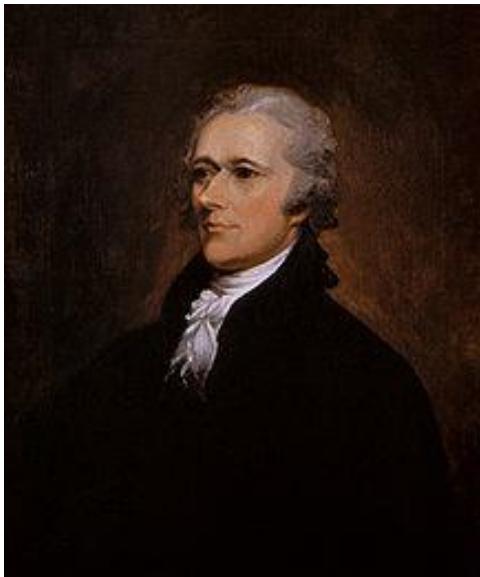
Logistic
Regression

Classification Reminder

Positive/negative sentiment

Spam/not spam

Authorship attribution
(Hamilton or Madison?)



Alexander Hamilton

Text Classification: definition

Input:

- a document x
- a fixed set of classes $C = \{c_1, c_2, \dots, c_J\}$

Output: a predicted class $\hat{y} \in C$

Binary Classification in Logistic Regression

Given a series of input/output pairs:

- $(x^{(i)}, y^{(i)})$

For each observation $x^{(i)}$

- We represent $x^{(i)}$ by a **feature vector** $[x_1, x_2, \dots, x_n]$
- We compute an output: a predicted class $\hat{y}^{(i)} \in \{0,1\}$

Features in logistic regression

- For feature x_i , weight w_i tells us how important is x_i
 - $x_i = \text{"review contains 'awesome'"}: w_i = +10$
 - $x_j = \text{"review contains 'abysmal'"}: w_j = -10$
 - $x_k = \text{"review contains 'mediocre'"}: w_k = -2$

Logistic Regression for one observation x

Input observation: vector $x = [x_1, x_2, \dots, x_n]$

Weights: one per feature: $W = [w_1, w_2, \dots, w_n]$

- Sometimes we call the weights $\theta = [\theta_1, \theta_2, \dots, \theta_n]$

Output: a predicted class $\hat{y} \in \{0, 1\}$

(multinomial logistic regression: $\hat{y} \in \{0, 1, 2, 3, 4\}$)

How to do classification

For each feature x_i , weight w_i tells us importance of x_i

- (Plus we'll have a bias b)

We'll sum up all the weighted features and the bias

$$Z = w_i x_i + b$$

$$Z = \sum_{i=1}^n w_i x_i + b$$

If this sum is high, we say $y=1$; if low, then $y=0$

But we want a probabilistic classifier

We need to formalize “sum is high”.

We'd like a principled classifier that gives us a probability, just like Naive Bayes did

We want a model that can tell us:

$$p(y=1|x; \theta)$$

$$p(y=0|x; \theta)$$

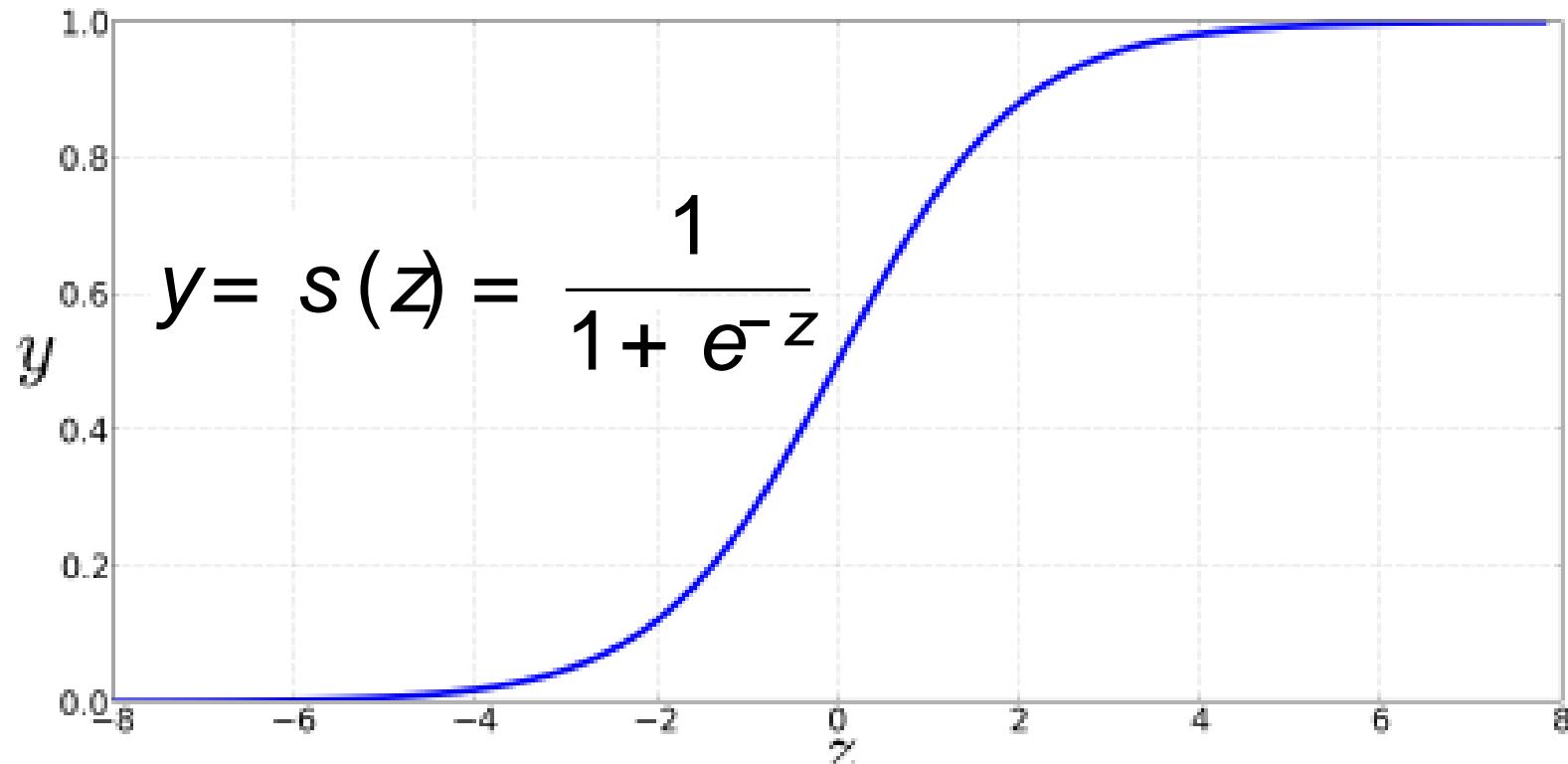
The problem: z isn't a probability, it's just a number!

$$z = w \cdot x + b$$

Solution: use a function of z that goes from 0 to 1

$$y = s(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + \exp(-z)}$$

The very useful sigmoid or logistic function



Idea of logistic regression

We'll compute $w \cdot x + b$

And then we'll pass it through the sigmoid function:

$$\sigma(w \cdot x + b)$$

And we'll just treat it as a probability

Making probabilities with sigmoids

$$\begin{aligned} P(y = 1) &= \sigma(w \cdot x + b) \\ &= \frac{1}{1 + \exp(-(w \cdot x + b))} \end{aligned}$$

$$\begin{aligned} P(y = 0) &= 1 - \sigma(w \cdot x + b) \\ &= 1 - \frac{1}{1 + \exp(-(w \cdot x + b))} \\ &= \frac{\exp(-(w \cdot x + b))}{1 + \exp(-(w \cdot x + b))} \end{aligned}$$

By the way:

$$\begin{aligned} P(y=0) &= 1 - \sigma(w \cdot x + b) & = \sigma(-(w \cdot x + b)) \\ &= 1 - \frac{1}{1 + \exp(-(w \cdot x + b))} & \text{Because} \\ &= \frac{\exp(-(w \cdot x + b))}{1 + \exp(-(w \cdot x + b))} & 1 - \sigma(x) = \sigma(-x) \end{aligned}$$

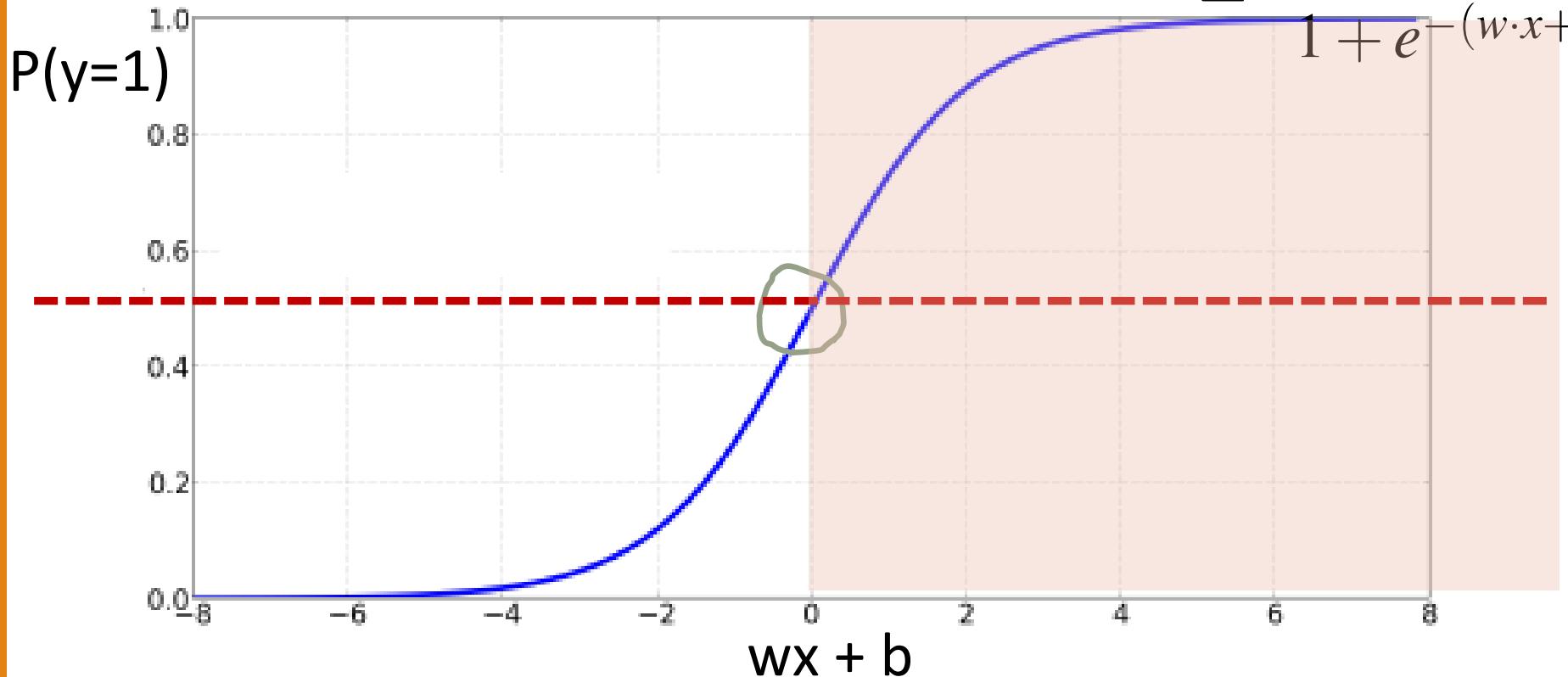
Turning a probability into a classifier

$$\hat{y} = \begin{cases} 1 & \text{if } P(y=1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

0.5 here is called the **decision boundary**

The probabilistic classifier $P(y = 1) = \sigma(w \cdot x + b)$

$$= \frac{1}{1 + e^{-(w \cdot x + b)}}$$



Turning a probability into a classifier

$$\hat{y} = \begin{cases} 1 & \text{if } P(y=1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Classification in Logistic Regression

Logistic
Regression

Logistic Regression

Logistic Regression: a text example
on sentiment classification

Sentiment example: does $y=1$ or $y=0$?

It's hokey . There are virtually no surprises , and the writing is second-rate . So why was it so enjoyable ? For one thing , the cast is great . Another nice touch is the music . I was overcome with the urge to get off the couch and start dancing . It sucked me in , and it'll do the same to you .

It's hokey. There are virtually no surprises , and the writing is second-rate.
 So why was it so enjoyable ? For one thing , the cast is
great . Another nice touch is the music I was overcome with the urge to get off
 the couch and start dancing . It sucked me in, and it'll do the same to you .

$$x_1 = 3$$

$$x_5 = 0$$

$$x_6 = 4.19$$

$$x_4 = 3$$

$$x_2 = 2$$

$$x_3 = 1$$

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) \in doc)	3
x_2	count(negative lexicon) \in doc)	2
x_3	$\begin{cases} 1 & \text{if “no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns \in doc)	3
x_5	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	$\log(\text{word count of doc})$	$\ln(66) = 4.19$

Classifying sentiment for input x

Var	Definition	Val	5.2
x_1	$\text{count}(\text{positive lexicon}) \in \text{doc}$)	3	
x_2	$\text{count}(\text{negative lexicon}) \in \text{doc})$	2	
x_3	$\begin{cases} 1 & \text{if “no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1	
x_4	$\text{count}(1\text{st and 2nd pronouns} \in \text{doc})$	3	
x_5	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0	
x_6	$\log(\text{word count of doc})$	$\ln(66) = 4.19$	

Suppose $w = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7]$

$$b = 0.1$$

Classifying sentiment for input x

$$p(+|x) = P(Y=1|x) = s(w \cdot x + b)$$

$$= s([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.19] + 0.1)$$

$$= s(.833)$$

$$= 0.70$$

$$p(-|x) = P(Y=0|x) = 1 - s(w \cdot x + b)$$

$$= 0.30$$

We can build features for logistic regression for any classification task: period disambiguation

This ends in a period.

The house at 465 Main St. is new.

End of sentence
Not end

$$\begin{aligned}x_1 &= \begin{cases} \rightarrow 1 & \text{if "Case}(w_i) = \text{Lower"} \\ 0 & \text{otherwise} \end{cases} \\x_2 &= \begin{cases} \rightarrow 1 & \text{if "w}_i \in \text{AcronymDict"} \\ 0 & \text{otherwise} \end{cases} \\x_3 &= \begin{cases} \rightarrow 1 & \text{if "w}_i = \text{St. \& Case}(w_{i-1}) = \text{Cap"} \\ 0 & \text{otherwise} \end{cases}\end{aligned}$$

Classification in (binary) logistic regression: summary

Given:

- a set of classes: (+ sentiment, - sentiment)
- a vector \mathbf{x} of features $[x_1, x_2, \dots, x_n]$
 - $x_1 = \text{count}(\text{"awesome"})$
 - $x_2 = \log(\text{number of words in review})$
- A vector \mathbf{w} of weights $[w_1, w_2, \dots, w_n]$
 - w_i for each feature f_i

$$\begin{aligned} P(y=1) &= \sigma(w \cdot x + b) \\ &= \frac{1}{1 + e^{-(w \cdot x + b)}} \end{aligned}$$

Logistic Regression

Logistic Regression: a text example
on sentiment classification

Learning: Cross-Entropy Loss

Logistic
Regression

Wait, where did the W's come from?

Supervised classification:

- We know the correct label y (either 0 or 1) for each x .
- But what the system produces is an estimate, \hat{y}

We want to set w and b to minimize the **distance** between our estimate $\hat{y}^{(i)}$ and the true $y^{(i)}$.

- We need a distance estimator: a **loss function** or a **cost function**
- We need an optimization algorithm to update w and b to minimize the loss.

Learning components

A loss function:

- **cross-entropy loss**

An optimization algorithm:

- **stochastic gradient descent**

The distance between \hat{y} and y

We want to know how far is the classifier output:

$$\hat{y} = \sigma(w \cdot x + b)$$

from the true output:

$$y \quad [= \text{either 0 or 1}]$$

We'll call this difference:

$$L(\hat{y}, y) = \text{how much } \hat{y} \text{ differs from the true } y$$

Intuition of negative log likelihood loss = cross-entropy loss

A case of conditional maximum likelihood estimation

We choose the parameters w, b that maximize

- the log probability
- of the true y labels in the training data
- given the observations x

Deriving cross-entropy loss for a single observation x

Goal: maximize probability of the correct label $p(y|x)$

Since there are only 2 discrete outcomes (0 or 1) we can express the probability $p(y|x)$ from our classifier (the thing we want to maximize) as

$$p(y|x) = \hat{y}^y (1 - \hat{y})^{1-y}$$

noting:

if $y=1$, this simplifies to \hat{y}

if $y=0$, this simplifies to $1 - \hat{y}$

Deriving cross-entropy loss for a single observation x

Goal: maximize probability of the correct label $p(y|x)$

Maximize: $p(y|x) = \hat{y}^y (1 - \hat{y})^{1-y}$

Now take the log of both sides (mathematically handy)

Maximize: $\log p(y|x) = \log [\hat{y}^y (1 - \hat{y})^{1-y}]$
= $y \log \hat{y} + (1 - y) \log (1 - \hat{y})$

Whatever values maximize $\log p(y|x)$ will also maximize $p(y|x)$

Deriving cross-entropy loss for a single observation x

Goal: maximize probability of the correct label $p(y|x)$

Maximize:

$$\begin{aligned}\log p(y|x) &= \log [\hat{y}^y (1 - \hat{y})^{1-y}] \\ &= y \log \hat{y} + (1 - y) \log (1 - \hat{y})\end{aligned}$$

Now flip sign to turn this into a loss: something to minimize

Cross-entropy loss (because is formula for $\text{cross-entropy}(y, \hat{y})$)

Minimize: $L_{\text{CE}}(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1 - y) \log (1 - \hat{y})]$

Or, plugging in definition of \hat{y} :

$$L_{\text{CE}}(\hat{y}, y) = -[y \log \sigma(w \cdot x + b) + (1 - y) \log (1 - \sigma(w \cdot x + b))]$$

Let's see if this works for our sentiment example

We want loss to be:

- smaller if the model estimate is close to correct
- bigger if model is confused

Let's first suppose the true label of this is $y=1$ (positive)

It's hokey . There are virtually no surprises , and the writing is second-rate . So why was it so enjoyable ? For one thing , the cast is great . Another nice touch is the music . I was overcome with the urge to get off the couch and start dancing . It sucked me in , and it'll do the same to you .

Let's see if this works for our sentiment example

True value is $y=1$. How well is our model doing?

$$\begin{aligned} p(+|x) = P(Y=1|x) &= s(w \cdot x + b) \\ &= s([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.19] + 0.1) \\ &= s(.833) \\ &= 0.70 \end{aligned} \tag{5.6}$$

Pretty well! What's the loss?

$$\begin{aligned} L_{\text{CE}}(\hat{y}, y) &= -[y \log \sigma(w \cdot x + b) + (1 - y) \log(1 - \sigma(w \cdot x + b))] \\ &= -[\log \sigma(w \cdot x + b)] \\ &= -\log(.70) \\ &= .36 \end{aligned}$$

Let's see if this works for our sentiment example

Suppose true value instead was $y=0$.

$$\begin{aligned} p(-|x) = P(Y=0|x) &= 1 - s(w \cdot x + b) \\ &= 0.30 \end{aligned}$$

What's the loss?

$$\begin{aligned} L_{\text{CE}}(\hat{y}, y) &= -[y \log \sigma(w \cdot x + b) + (1 - y) \log(1 - \sigma(w \cdot x + b))] \\ &= -[\log(1 - \sigma(w \cdot x + b))] \\ &= -\log(.30) \\ &= 1.2 \end{aligned}$$

Let's see if this works for our sentiment example

The loss when model was right (if true $y=1$)

$$\begin{aligned} L_{\text{CE}}(\hat{y}, y) &= -[y \log \sigma(w \cdot x + b) + (1 - y) \log(1 - \sigma(w \cdot x + b))] \\ &= -[\log \sigma(w \cdot x + b)] \\ &= -\log(.70) \\ &= .36 \end{aligned}$$

Is lower than the loss when model was wrong (if true $y=0$):

$$\begin{aligned} L_{\text{CE}}(\hat{y}, y) &= -[y \log \sigma(w \cdot x + b) + (1 - y) \log(1 - \sigma(w \cdot x + b))] \\ &= -[\log(1 - \sigma(w \cdot x + b))] \\ &= -\log(.30) \\ &= 1.2 \end{aligned}$$

Sure enough, loss was bigger when model was wrong!

Cross-Entropy Loss

Logistic
Regression

Stochastic Gradient Descent

Logistic Regression

Our goal: minimize the loss

Let's make explicit that the loss function is parameterized by weights $\theta=(w,b)$

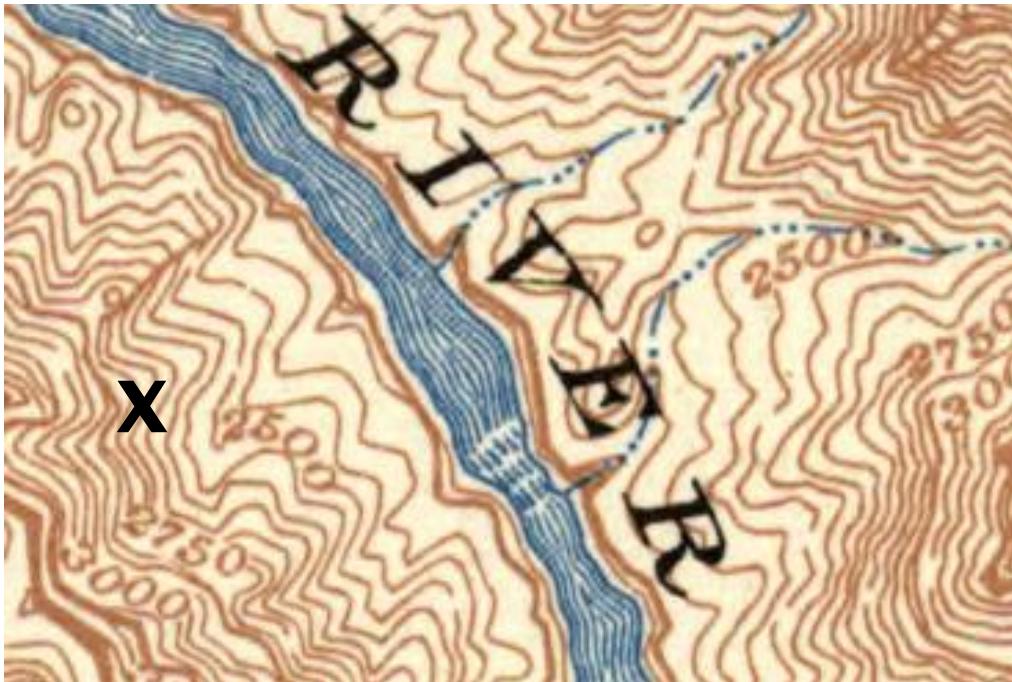
- And we'll represent \hat{y} as $f(x; \theta)$ to make the dependence on θ more obvious

We want the weights that minimize the loss, averaged over all examples:

$$\hat{\theta} = \operatorname{argmin}_{\theta} \frac{1}{m} \sum_{i=1}^m L_{\text{CE}}(f(x^{(i)}; \theta), y^{(i)})$$

Intuition of gradient descent

How do I get to the bottom of this river canyon?



Look around me 360°
Find the direction of
steepest slope down
Go that way

Our goal: minimize the loss

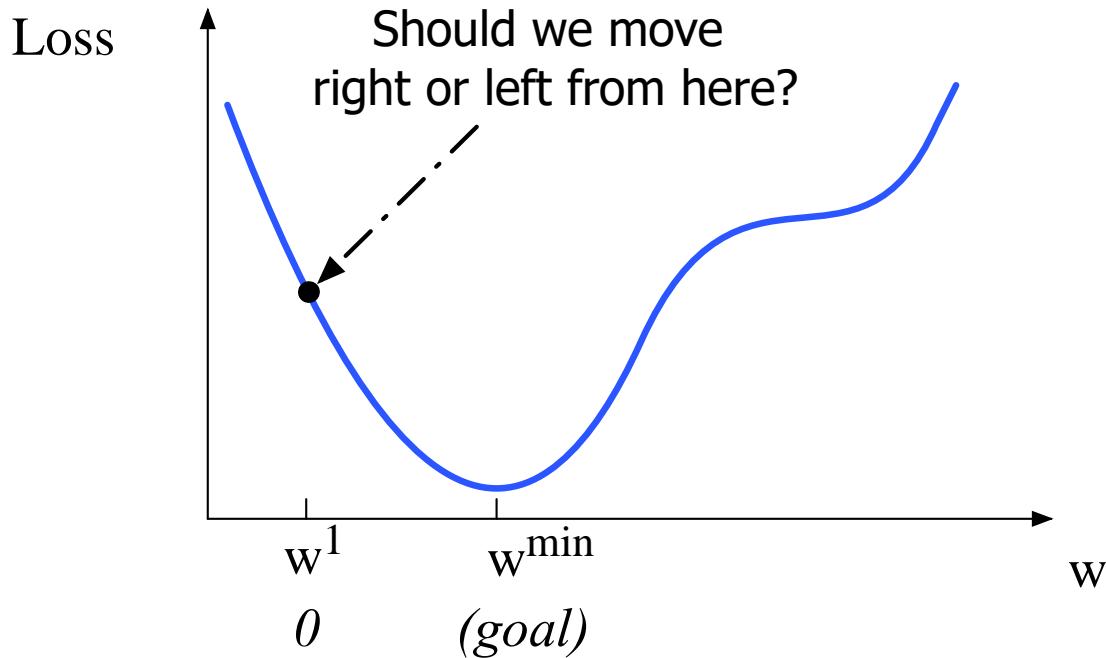
For logistic regression, loss function is **convex**

- A convex function has just one minimum
- Gradient descent starting from any point is guaranteed to find the minimum
 - (Loss for neural networks is non-convex)

Let's first visualize for a single scalar w

Q: Given current w , should we make it bigger or smaller?

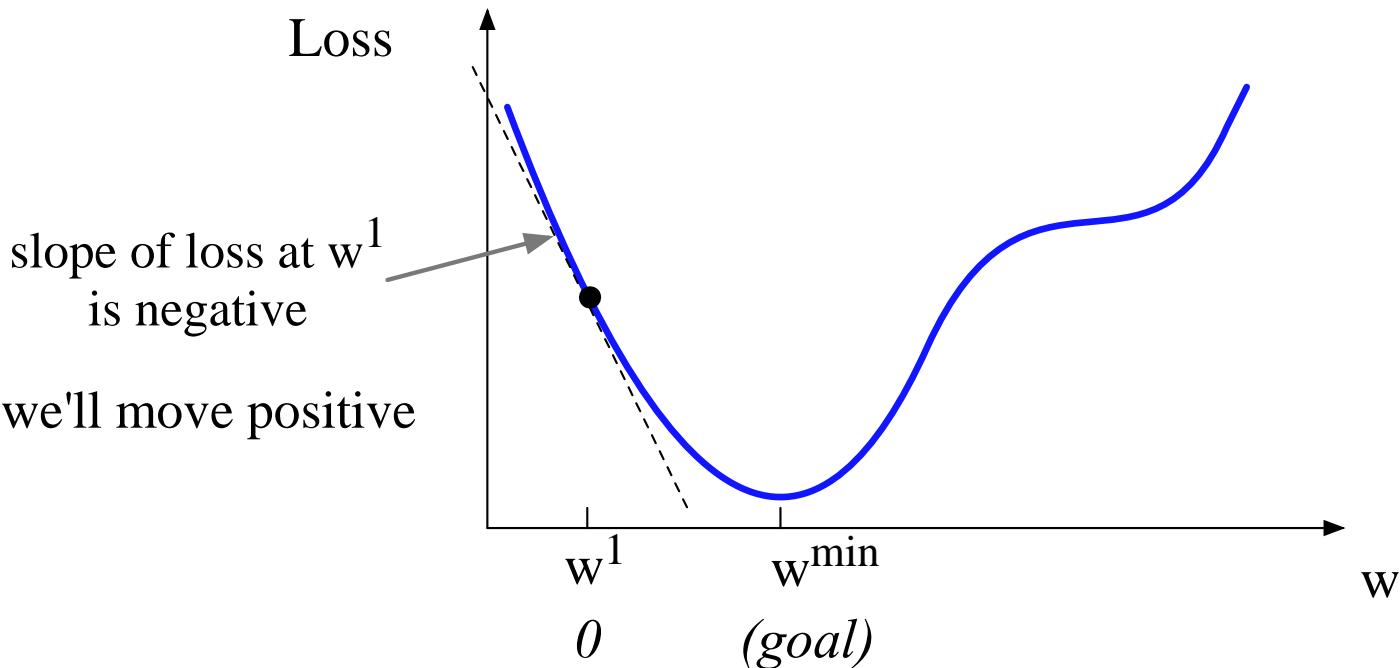
A: Move w in the reverse direction from the slope of the function



Let's first visualize for a single scalar w

Q: Given current w , should we make it bigger or smaller?

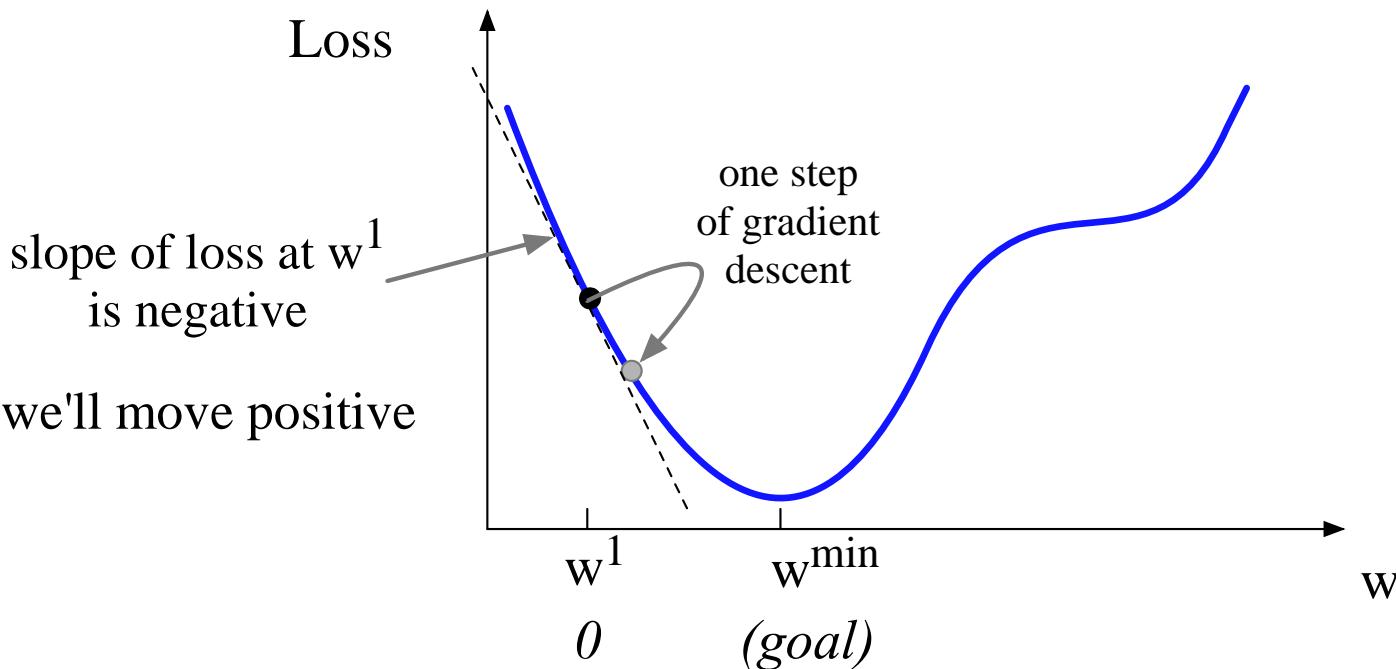
A: Move w in the reverse direction from the slope of the function



Let's first visualize for a single scalar w

Q: Given current w , should we make it bigger or smaller?

A: Move w in the reverse direction from the slope of the function



Gradients

The **gradient** of a function of many variables is a vector pointing in the direction of the greatest increase in a function.

Gradient Descent: Find the gradient of the loss function at the current point and move in the **opposite** direction.

How much do we move in that direction ?

- The value of the gradient (slope in our example) $\frac{d}{dw} L(f(x; w), y)$ weighted by a **learning rate** η
- Higher learning rate means move w faster

$$w^{t+1} = w^t - h \frac{d}{dw} L(f(x; w), y)$$

Now let's consider N dimensions

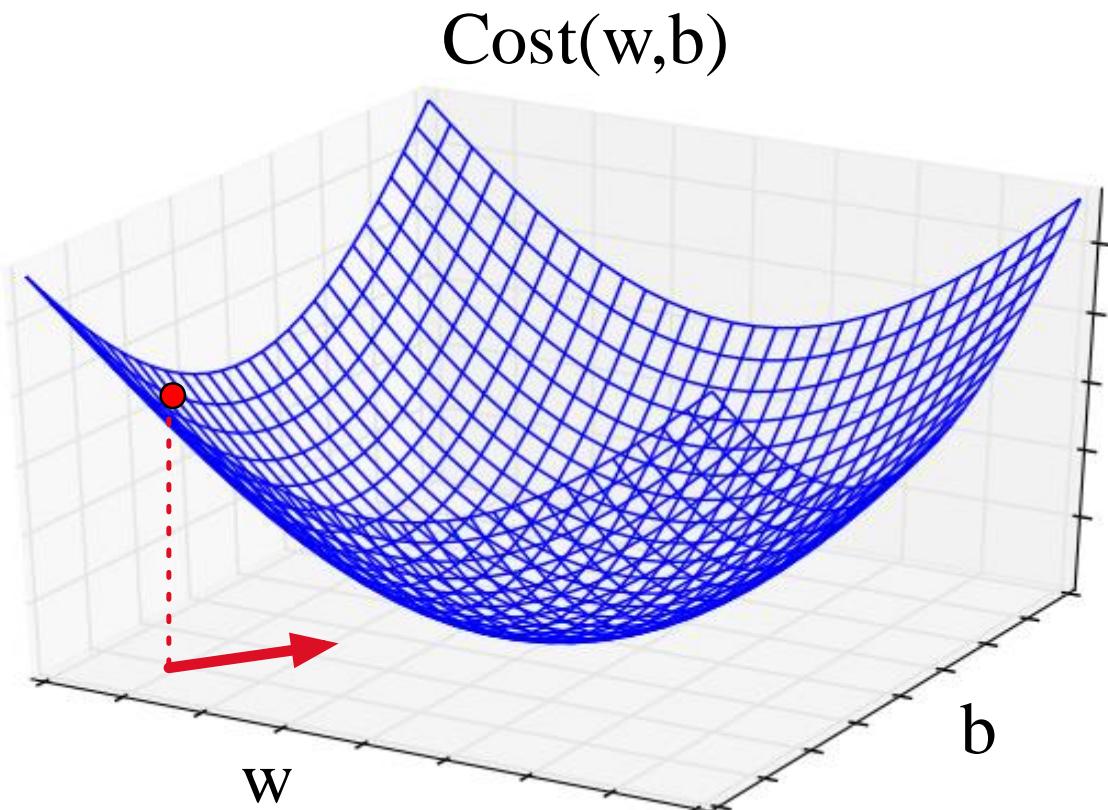
We want to know where in the N -dimensional space (of the N parameters that make up θ) we should move.

The gradient is just such a vector; it expresses the directional components of the sharpest slope along each of the N dimensions.

Imagine 2 dimensions, w and b

Visualizing the
gradient vector at
the red point

It has two
dimensions shown
in the x-y plane



Real gradients

Are much longer; lots and lots of weights

For each dimension w_i , the gradient component i tells us the slope with respect to that variable.

- “How much would a small change in w_i influence the total loss function L ? ”
- We express the slope as a partial derivative ∂ of the loss ∂w_i

The gradient is then defined as a vector of these partials.

The gradient

We'll represent \hat{y} as $f(x; \theta)$ to make the dependence on θ more obvious:

$$-\nabla_q L(f(x, q), y) = \begin{pmatrix} \frac{\partial}{\partial w_1} L(f(x, q), y) \\ \frac{\partial}{\partial w_2} L(f(x, q), y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x, q), y) \end{pmatrix}$$

The final equation for updating θ based on the gradient is thus

$$q_{t+1} = q_t - h \nabla L(f(x, q), y)$$

What are these partial derivatives for logistic regression?

The loss function

$$L_{\text{CE}}(\hat{y}, y) = -[y \log \sigma(w \cdot x + b) + (1 - y) \log (1 - \sigma(w \cdot x + b))]$$

The elegant derivative of this function (see textbook 5.8 for derivation)

$$\frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$$

```

function STOCHASTIC GRADIENT DESCENT( $L()$ ,  $f()$ ,  $x$ ,  $y$ ) returns  $\theta$ 
    # where: L is the loss function
    #      f is a function parameterized by  $\theta$ 
    #      x is the set of training inputs  $x^{(1)}$ ,  $x^{(2)}$ , ...,  $x^{(m)}$ 
    #      y is the set of training outputs (labels)  $y^{(1)}$ ,  $y^{(2)}$ , ...,  $y^{(m)}$ 

```

$\theta \leftarrow 0$

repeat til done

For each training tuple $(x^{(i)}, y^{(i)})$ (in random order)

1. Optional (for reporting): # How are we doing on this tuple?
 Compute $\hat{y}^{(i)} = f(x^{(i)}; \theta)$ # What is our estimated output \hat{y} ?
 Compute the loss $L(\hat{y}^{(i)}, y^{(i)})$ # How far off is $\hat{y}^{(i)}$ from the true output $y^{(i)}$?
2. $g \leftarrow \nabla_{\theta} L(f(x^{(i)}; \theta), y^{(i)})$ # How should we move θ to maximize loss?
 3. $\theta \leftarrow \theta - \eta g$ # Go the other way instead

return θ

Hyperparameters

The learning rate η is a **hyperparameter**

- too high: the learner will take big steps and overshoot
- too low: the learner will take too long

Hyperparameters:

- Briefly, a special kind of parameter for an ML model
- Instead of being learned by algorithm from supervision (like regular parameters), they are chosen by algorithm designer.

Stochastic Gradient Descent

Logistic Regression

Logistic Regression

Stochastic Gradient Descent: An example and more details

Working through an example

One step of gradient descent

A mini-sentiment example, where the true $y=1$ (positive)

Two features:

$x_1 = 3$ (count of positive lexicon words)

$x_2 = 2$ (count of negative lexicon words)

Assume 3 parameters (2 weights and 1 bias) in Θ^0 are zero:

$$w_1 = w_2 = b = 0$$

$$\eta = 0.1$$

Example of gradient descent

Update step for update θ is:

$$w_1 = w_2 = b = 0; \\ x_1 = 3; \quad x_2 = 2$$

$$q_{t+1} = q_t - h - L(f(x, q), y)$$

where
$$\frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$$

Gradient vector has 3 dimensions:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_1} \\ \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_2} \\ \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial b} \end{bmatrix}$$

Example of gradient descent

Update step for update θ is:

$$w_1 = w_2 = b = 0; \\ x_1 = 3; \quad x_2 = 2$$

$$q_{t+1} = q_t - h - L(f(x, q), y)$$

where $\frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$

Gradient vector has 3 dimensions:

$$\nabla_{w,b} = \left[\begin{array}{c} \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_1} \\ \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_2} \\ \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial b} \end{array} \right] = \left[\quad \right]$$

Example of gradient descent

Update step for update θ is:

$$w_1 = w_2 = b = 0; \\ x_1 = 3; \quad x_2 = 2$$

$$q_{t+1} = q_t - h - L(f(x, q), y)$$

where $\frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$

Gradient vector has 3 dimensions:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_1} \\ \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_2} \\ \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix}$$

Example of gradient descent

$$w_1 = w_2 = b = 0; \\ x_1 = 3; \quad x_2 = 2$$

Update step for update θ is:

$$q_{t+1} = q_t - h - L(f(x, q), y)$$

where $\frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$

Gradient vector has 3 dimensions:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_1} \\ \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_2} \\ \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} =$$

Example of gradient descent

$$w_1 = w_2 = b = 0; \\ x_1 = 3; \quad x_2 = 2$$

Update step for update θ is:

$$q_{t+1} = q_t - h - L(f(x, q), y)$$

where
$$\frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$$

Gradient vector has 3 dimensions:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_1} \\ \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_2} \\ \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Example of gradient descent

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_1} \\ \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_2} \\ \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Now that we have a gradient, we compute the new parameter vector θ^1 by moving θ^0 in the opposite direction from the gradient:

$$q_{t+1} = q_t - h \nabla L(f(x, q), y) \quad \eta = 0.1;$$

$$\theta^1 =$$

Example of gradient descent

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{CE}(\hat{y}, y)}{\partial w_1} \\ \frac{\partial L_{CE}(\hat{y}, y)}{\partial w_2} \\ \frac{\partial L_{CE}(\hat{y}, y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Now that we have a gradient, we compute the new parameter vector θ^1 by moving θ^0 in the opposite direction from the gradient:

$$q_{t+1} = q_t - h \nabla L(f(x, q), y) \quad \eta = 0.1;$$

$$\theta^1 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Example of gradient descent

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_1} \\ \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_2} \\ \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Now that we have a gradient, we compute the new parameter vector θ^1 by moving θ^0 in the opposite direction from the gradient:

$$q_{t+1} = q_t - h \nabla L(f(x, q), y) \quad \eta = 0.1;$$

$$\theta^1 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} .15 \\ .1 \\ .05 \end{bmatrix}$$

Example of gradient descent

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{CE}(\hat{y}, y)}{\partial w_1} \\ \frac{\partial L_{CE}(\hat{y}, y)}{\partial w_2} \\ \frac{\partial L_{CE}(\hat{y}, y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Now that we have a gradient, we compute the new parameter vector θ^1 by moving θ^0 in the opposite direction from the gradient:

$$q_{t+1} = q_t - h \nabla L(f(x, q), y) \quad \eta = 0.1;$$

$$\theta^1 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} .15 \\ .1 \\ .05 \end{bmatrix}$$

Note that enough negative examples would eventually make w_2 negative

Mini-batch training

Stochastic gradient descent chooses a single random example at a time.

That can result in choppy movements

More common to compute gradient over batches of training instances.

Batch training: entire dataset

Mini-batch training: m examples (512, or 1024)

Logistic Regression

Stochastic Gradient Descent: An example and more details

Regularization

Logistic
Regression

Overfitting

A model that perfectly match the training data has a problem.

It will also **overfit** to the data, modeling noise

- A random word that perfectly predicts y (it happens to only occur in one class) will get a very high weight.
- Failing to generalize to a test set without this word.

A good model should be able to **generalize**

Overfitting

+

This movie drew me in, and it'll do the same to you.

-

I can't tell you how much I hated this movie. It sucked.

Useful or harmless features

X1 = "this"

X2 = "movie"

X3 = "hated"

X4 = "drew me in"

4gram features that just "memorize" training set and might cause problems

X5 = "the same to you"

X7 = "tell you how much"

Overfitting

4-gram model on tiny data will just memorize the data

- 100% accuracy on the training set

But it will be surprised by the novel 4-grams in the test data

- Low accuracy on test set

Models that are too powerful can **overfit** the data

- Fitting the details of the training data so exactly that the model doesn't generalize well to the test set
- How to avoid overfitting?
 - Regularization in logistic regression
 - Dropout in neural networks

Regularization

A solution for overfitting

Add a regularization term $R(\theta)$ to the loss function
(for now written as maximizing logprob rather than minimizing loss)

$$\hat{\theta} = \operatorname{argmax}_{\theta} \sum_{i=1}^m \log P(y^{(i)} | x^{(i)}) - \alpha R(\theta)$$

Idea: choose an $R(\theta)$ that penalizes large weights

- fitting the data well with lots of big weights not as good as fitting the data a little less well, with small weights

L2 Regularization (= ridge regression)

The sum of the squares of the weights

The name is because this is the (square of the)
L2 norm $\|\theta\|_2$, = **Euclidean distance** of θ to the origin.

$$R(\theta) = \|\theta\|_2^2 = \sum_{j=1}^n \theta_j^2$$

L2 regularized objective function:

$$\hat{\theta} = \operatorname{argmax}_{\theta} \left[\sum_{i=1}^m \log P(y^{(i)} | x^{(i)}) \right] - \alpha \sum_{j=1}^n \theta_j^2$$

L1 Regularization (= lasso regression)

The sum of the (absolute value of the) weights

Named after the **L1 norm** $\|W\|_1$, = sum of the absolute values of the weights, = **Manhattan distance**

$$R(\theta) = \|\theta\|_1 = \sum_{i=1}^n |\theta_i|$$

L1 regularized objective function:

$$\hat{\theta} = \operatorname{argmax}_{\theta} \left[\sum_{i=1}^m \log P(y^{(i)} | x^{(i)}) \right] - \alpha \sum_{j=1}^n |\theta_j|$$

Regularization

Logistic
Regression

Logistic Regression

Multinomial Logistic Regression

Multinomial Logistic Regression

Often we need more than 2 classes

- Positive/negative/neutral
- Parts of speech (noun, verb, adjective, adverb, preposition, etc.)
- Classify emergency SMSs into different actionable classes

If >2 classes we use **multinomial logistic regression**

- = Softmax regression
- = Multinomial logit
- = (defunct names : Maximum entropy modeling or MaxEnt

So "logistic regression" will just mean binary (2 output classes)

Multinomial Logistic Regression

The probability of everything must still sum to 1

$$P(\text{positive} \mid \text{doc}) + P(\text{negative} \mid \text{doc}) + P(\text{neutral} \mid \text{doc}) = 1$$

Need a generalization of the sigmoid called the **softmax**

- Takes a vector $z = [z_1, z_2, \dots, z_k]$ of k arbitrary values
- Outputs a probability distribution
 - each value in the range $[0,1]$
 - all the values summing to 1

The softmax function

Turns a vector $z = [z_1, z_2, \dots, z_k]$ of k arbitrary values into probabilities

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^k \exp(z_j)} \quad 1 \leq i \leq k$$

The denominator $\sum_{i=1}^k e^{z_i}$ is used to normalize all the values into probabilities.

$$\text{softmax}(z) = \left[\frac{\exp(z_1)}{\sum_{i=1}^k \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^k \exp(z_i)}, \dots, \frac{\exp(z_k)}{\sum_{i=1}^k \exp(z_i)} \right]$$

The softmax function

- Turns a vector $z = [z_1, z_2, \dots, z_k]$ of k arbitrary values into probabilities

$$z = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]$$

$$\text{softmax}(z) = \left[\frac{\exp(z_1)}{\sum_{i=1}^k \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^k \exp(z_i)}, \dots, \frac{\exp(z_k)}{\sum_{i=1}^k \exp(z_i)} \right]$$

$$[0.055, 0.090, 0.0067, 0.10, 0.74, 0.010]$$

Softmax in multinomial logistic regression

$$p(y = c|x) = \frac{\exp(w_c \cdot x + b_c)}{\sum_{j=1}^k \exp(w_j \cdot x + b_j)}$$

Input is still the dot product between weight vector w and input vector x

But now we'll need separate weight vectors for each of the K classes.

Features in binary versus multinomial logistic regression

Binary: positive weight $\rightarrow y=1$ neg weight $\rightarrow y=0$

$$x_5 = \begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases} \quad w_5 = 3.0$$

Multinominal: separate weights for each class:

Feature	Definition	$w_{5,+}$	$w_{5,-}$	$w_{5,0}$
$f_5(x)$	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	3.5	3.1	-5.3

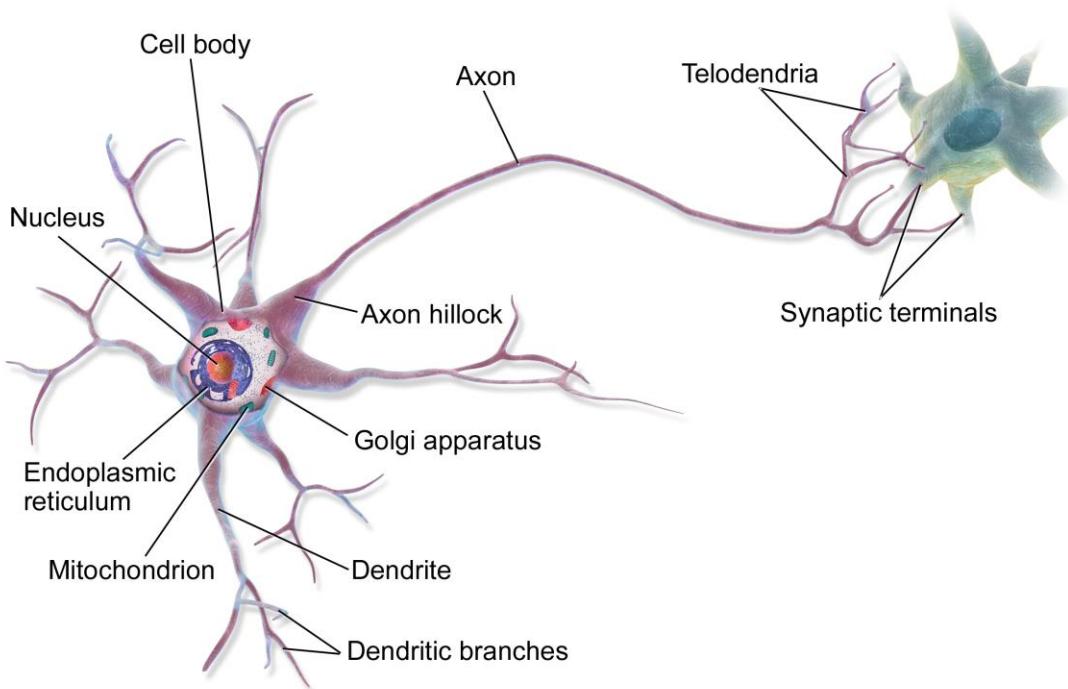
Logistic Regression

Multinomial Logistic Regression

Simple Neural Networks and Neural Language Models

Units in Neural Networks

This is in your brain



By BruceBlaus - Own work, CC BY 3.0,
<https://commons.wikimedia.org/w/index.php?curid=28761830>

Neural Network Unit

This is not in your brain

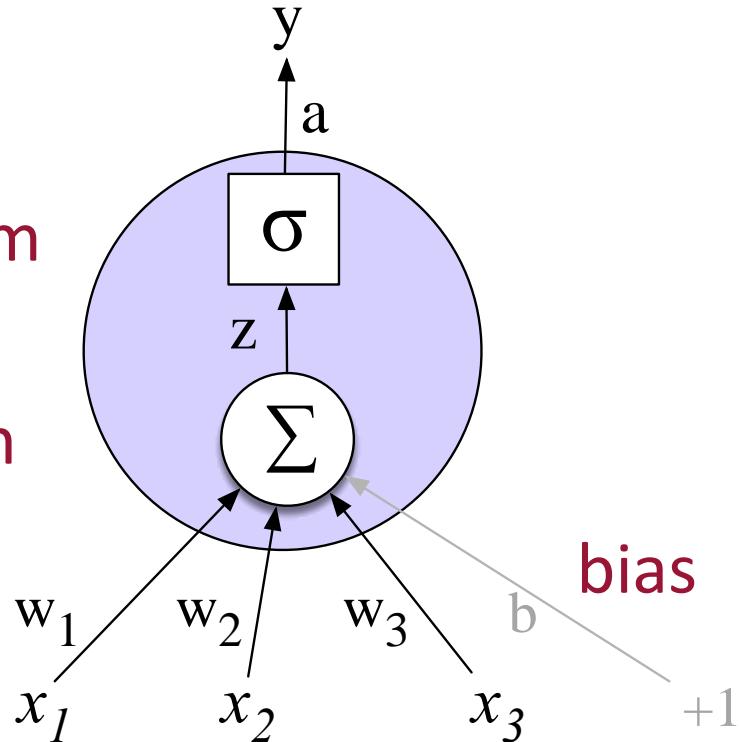
Output value

Non-linear transform

Weighted sum

Weights

Input layer



Neural unit

Take weighted sum of inputs, plus a bias

$$z = b + \sum_i w_i x_i$$

$$z = w \cdot x + b$$

Instead of just using z , we'll apply a nonlinear activation function f :

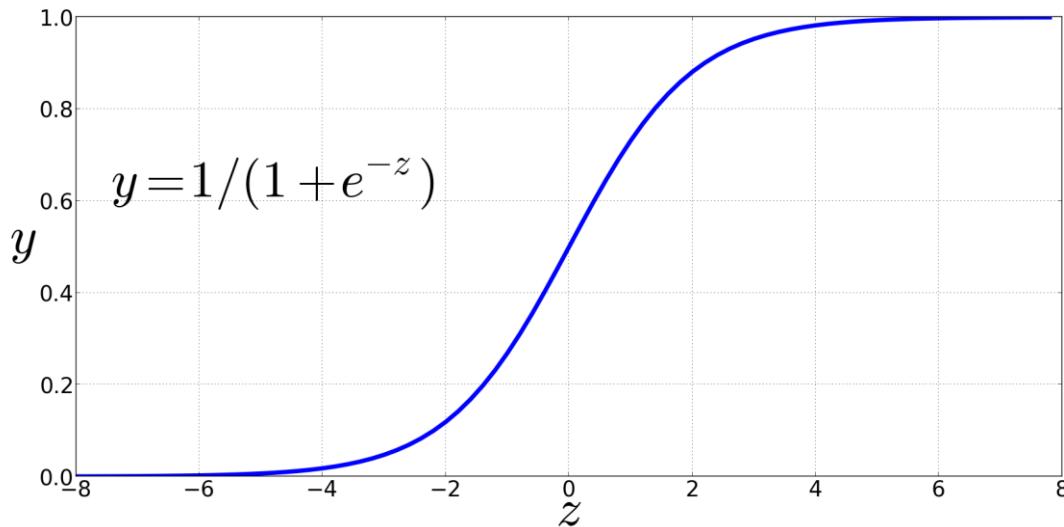
$$y = a = f(z)$$

Non-Linear Activation Functions

We're already seen the sigmoid for logistic regression:

Sigmoid

$$y = s(z) = \frac{1}{1 + e^{-z}}$$



Final function the unit is computing

$$y = s(w \cdot x + b) = \frac{1}{1 + \exp(-(w \cdot x + b))}$$

Final unit again

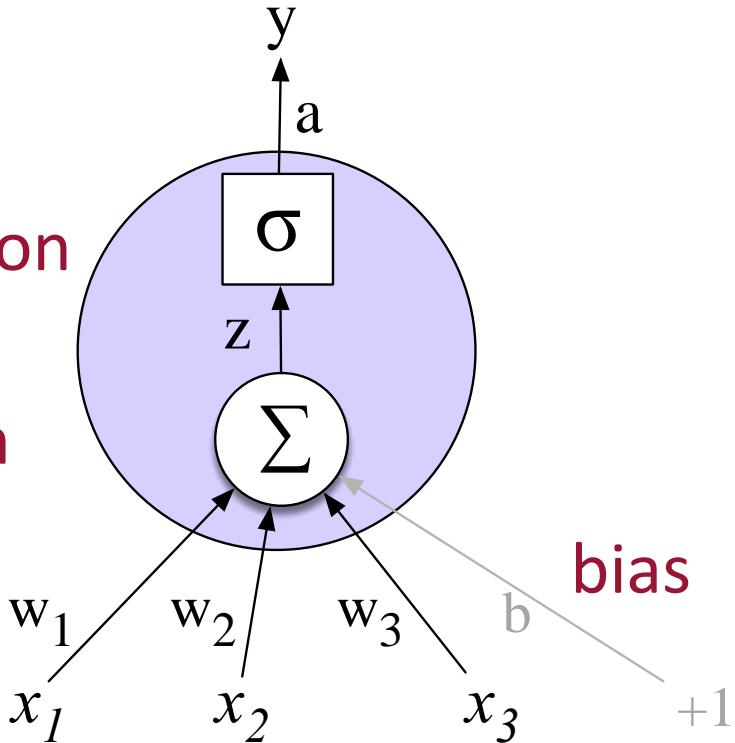
Output value

Non-linear activation function

Weighted sum

Weights

Input layer



An example

Suppose a unit has:

$$w = [0.2, 0.3, 0.9]$$

$$b = 0.5$$

What happens with input x:

$$x = [0.5, 0.6, 0.1]$$

$$y = s(w \cdot x + b) =$$

An example

Suppose a unit has:

$$w = [0.2, 0.3, 0.9]$$

$$b = 0.5$$

What happens with the following input x ?

$$x = [0.5, 0.6, 0.1]$$

$$1$$

$$y = s(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}} =$$

An example

Suppose a unit has:

$$w = [0.2, 0.3, 0.9]$$

$$b = 0.5$$

What happens with input x :

$$x = [0.5, 0.6, 0.1]$$

1

$$y = s(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}} =$$
$$\frac{1}{1 + e^{-(.5 \cdot 2 + .6 \cdot 3 + .1 \cdot 9 + .5)}} =$$

An example

Suppose a unit has:

$$w = [0.2, 0.3, 0.9]$$

$$b = 0.5$$

What happens with input x:

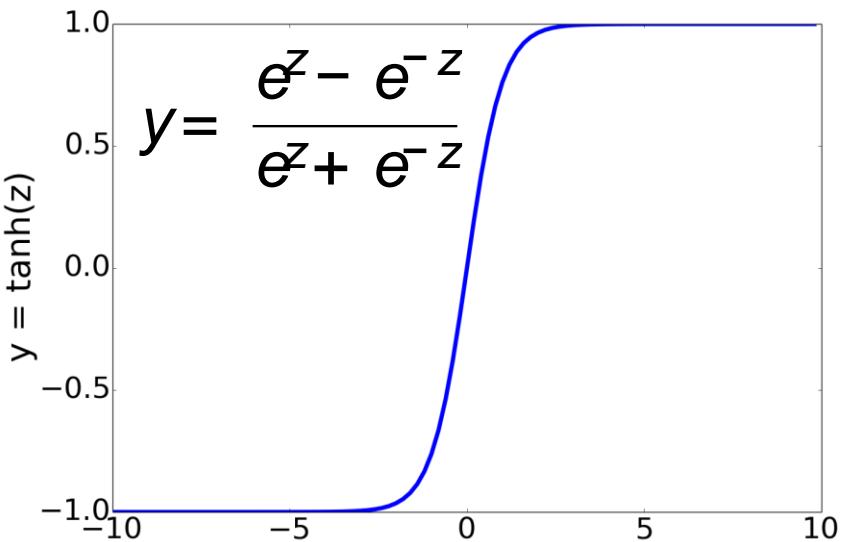
$$x = [0.5, 0.6, 0.1]$$

$$\frac{1}{1 + e^{-(w \cdot x + b)}}$$

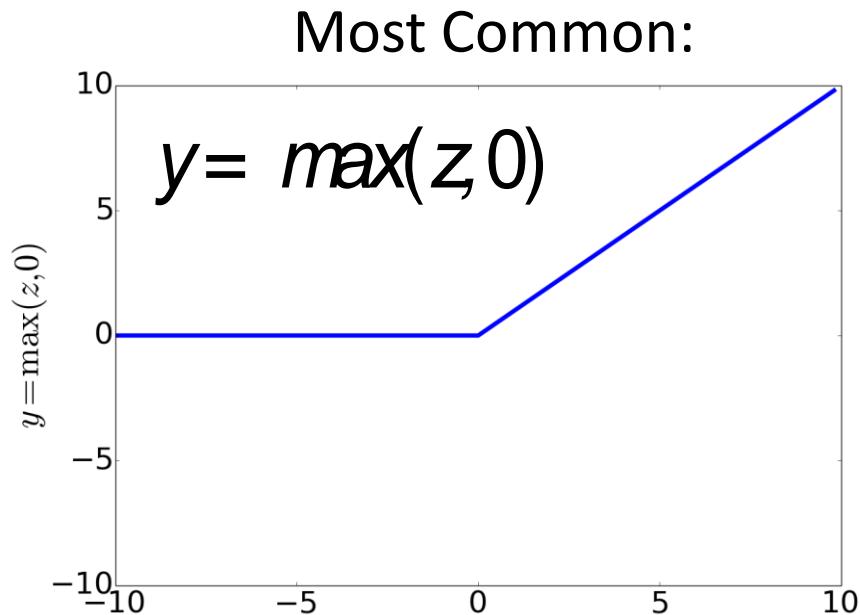
$$y = s(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}} =$$

$$\frac{1}{1 + e^{-(.5 \cdot 2 + .6 \cdot 3 + .1 \cdot 9 + .5)}} = \frac{1}{1 + e^{-0.87}} = .70$$

Non-Linear Activation Functions besides sigmoid



tanh



ReLU
Rectified Linear Unit

Simple Neural Networks and Neural Language Models

Units in Neural Networks

Simple Neural Networks and Neural Language Models

The XOR problem

The XOR problem

Minsky and Papert (1969)

Can neural units compute simple functions of input?

AND		OR		XOR	
x1	x2	x1	x2	x1	x2
0	0	0	0	0	0
0	1	0	1	1	1
1	0	1	0	1	0
1	1	1	1	1	0

Perceptrons

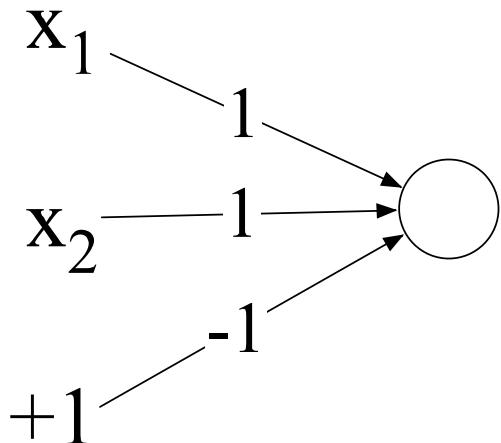
A very simple neural unit

- Binary output (0 or 1)
- No non-linear activation function

$$y = \begin{cases} 0, & \text{if } w \cdot x + b \leq 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases}$$

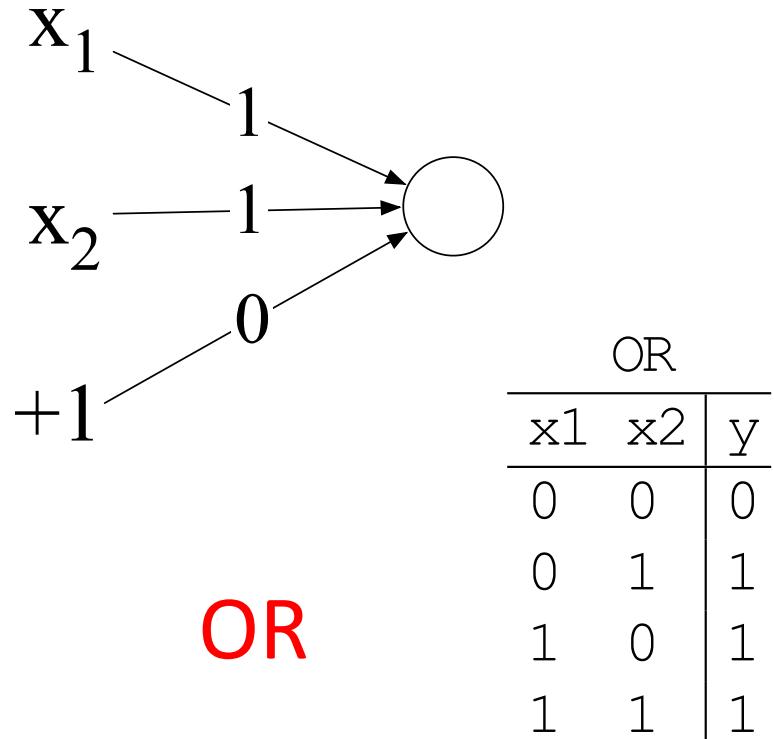
Easy to build AND or OR with perceptrons

$$y = \begin{cases} 0, & \text{if } w \cdot x + b \leq 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases}$$



AND

		AND
x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

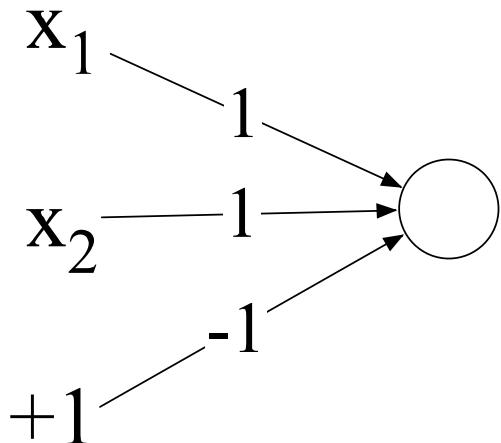


OR

		OR
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

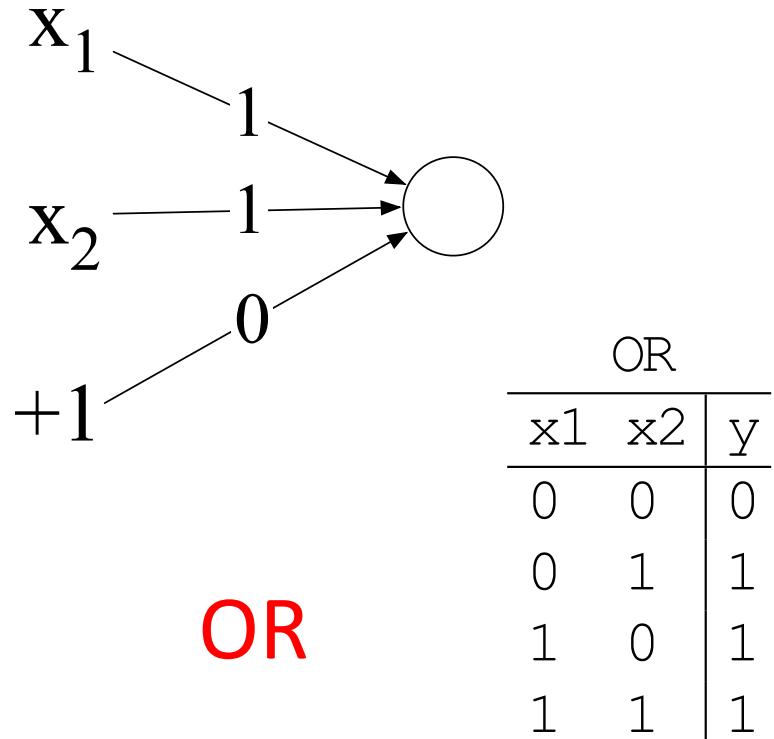
Easy to build AND or OR with perceptrons

$$y = \begin{cases} 0, & \text{if } w \cdot x + b \leq 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases}$$



AND

		AND
x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

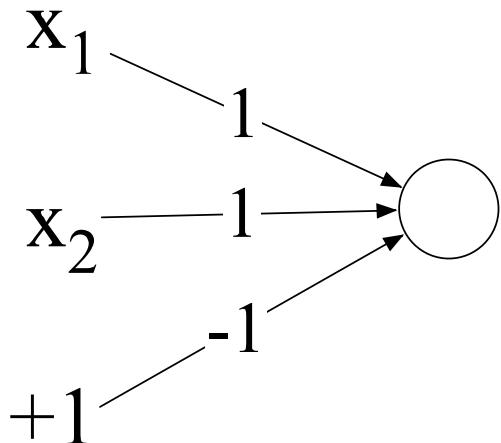


OR

		OR
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

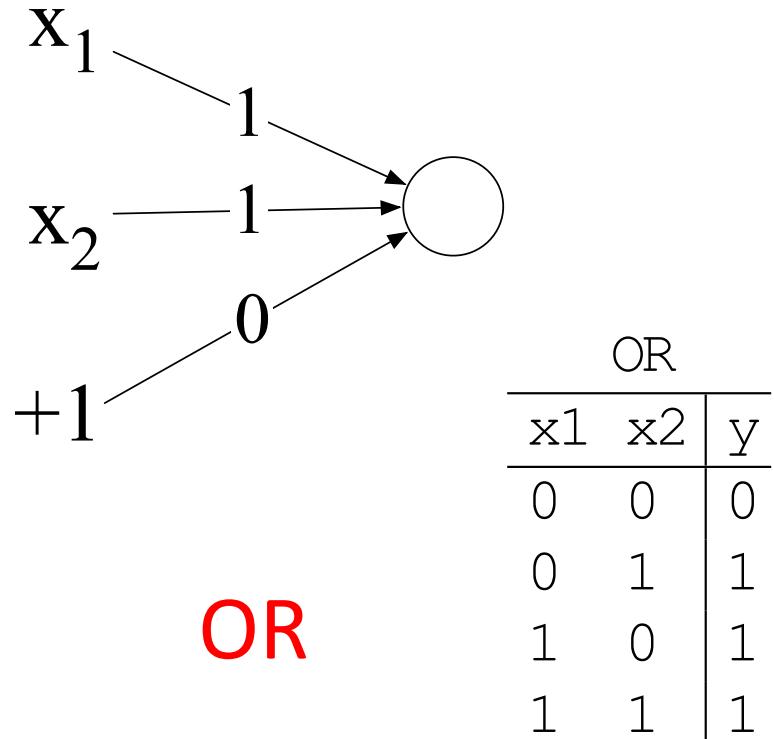
Easy to build AND or OR with perceptrons

$$y = \begin{cases} 0, & \text{if } w \cdot x + b \leq 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases}$$



AND

		AND
x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1



OR

		OR
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

Not possible to capture XOR with perceptrons

Pause the lecture and try for yourself!

Why? Perceptrons are linear classifiers

Perceptron equation given x_1 and x_2 , is the equation of a line

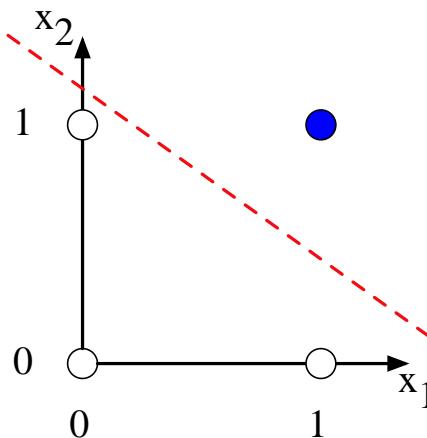
$$w_1x_1 + w_2x_2 + b = 0$$

(in standard linear format: $x_2 = (-w_1/w_2)x_1 + (-b/w_2)$)

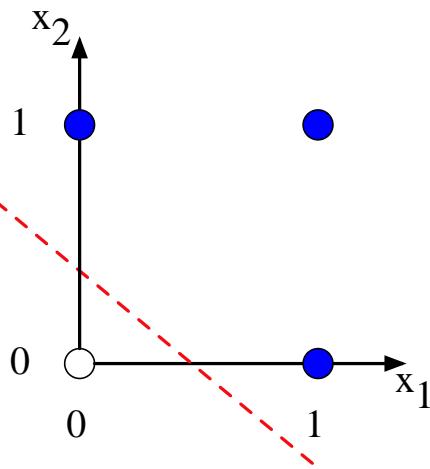
This line acts as a **decision boundary**

- 0 if input is on one side of the line
- 1 if on the other side of the line

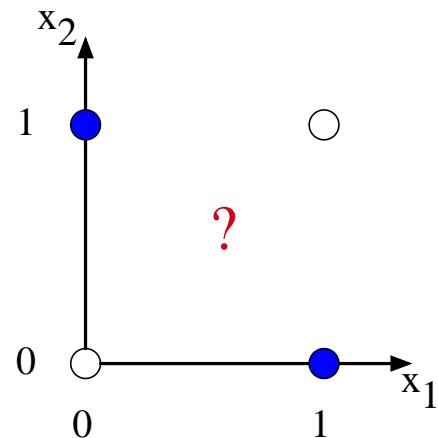
Decision boundaries



a) x_1 AND x_2



b) x_1 OR x_2



c) x_1 XOR x_2

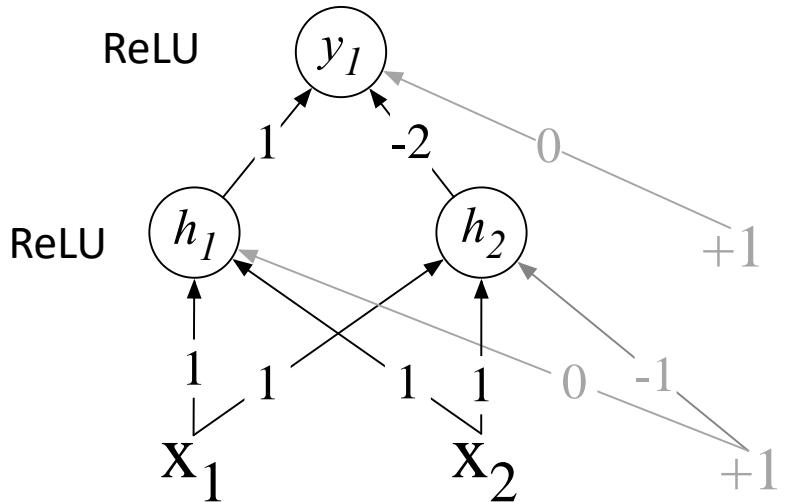
XOR is not a **linearly separable** function!

Solution to the XOR problem

XOR **can't** be calculated by a single perceptron

XOR **can** be calculated by a layered network of units.

XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

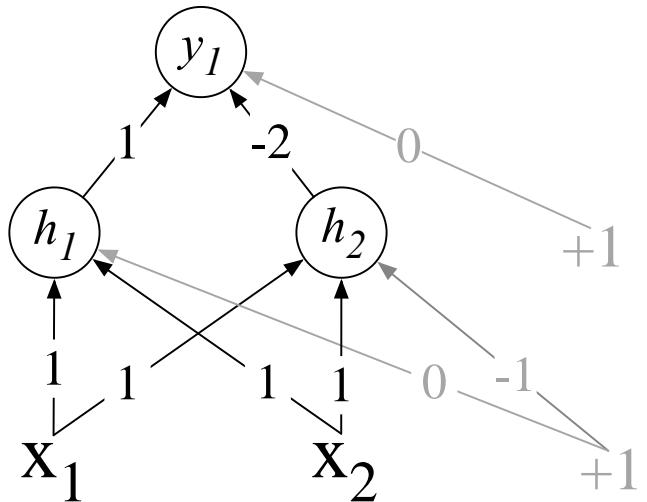


Solution to the XOR problem

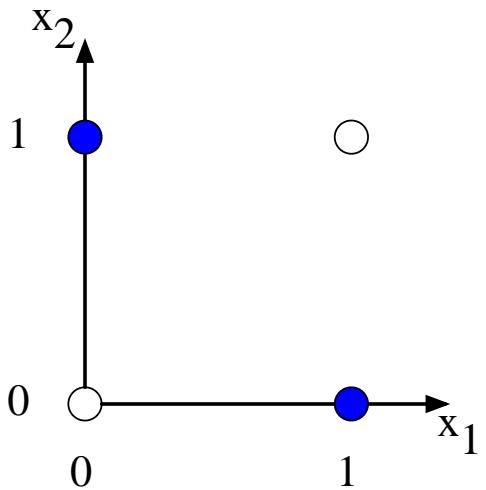
XOR **can't** be calculated by a single perceptron

XOR **can** be calculated by a layered network of units.

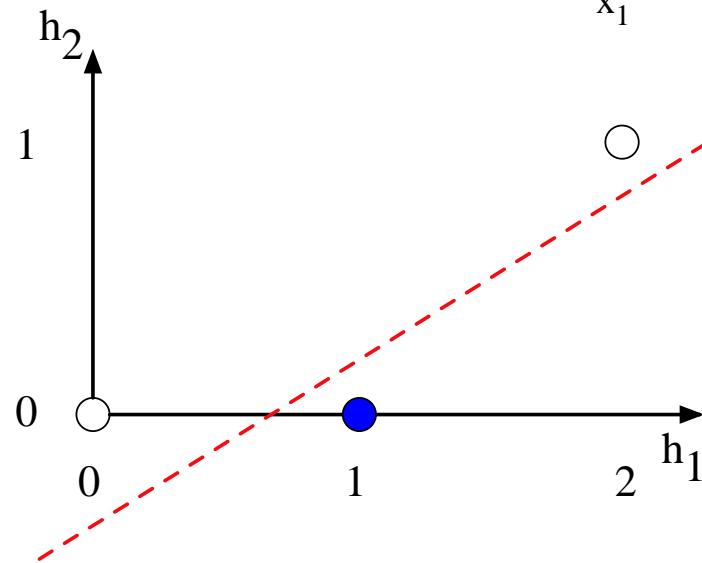
XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0



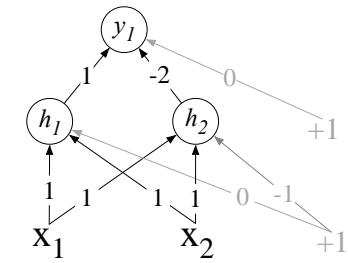
The hidden representation h



a) The original x space



b) The new (linearly separable) h space



(With learning: hidden layers will learn to form useful representations)

Simple Neural Networks and Neural Language Models

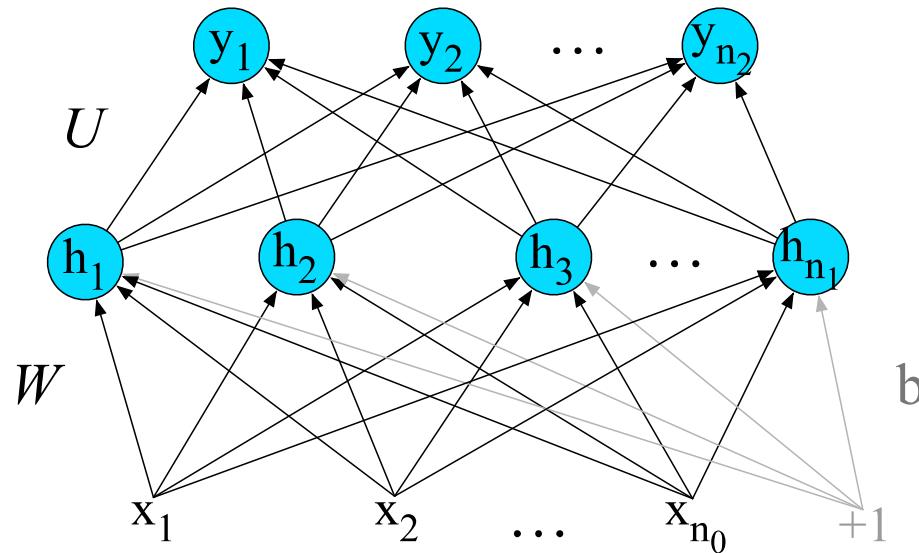
The XOR problem

Simple Neural Networks and Neural Language Models

Feedforward Neural Networks

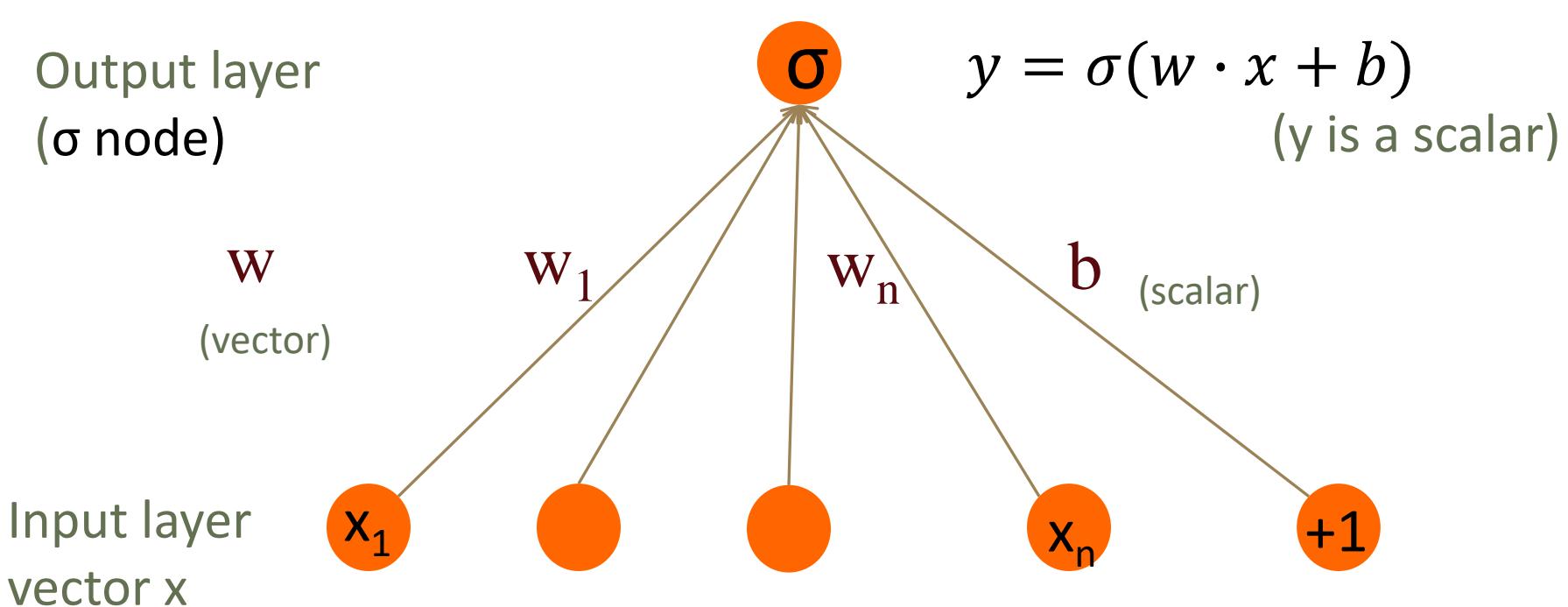
Feedforward Neural Networks

Can also be called **multi-layer perceptrons** (or **MLPs**) for historical reasons



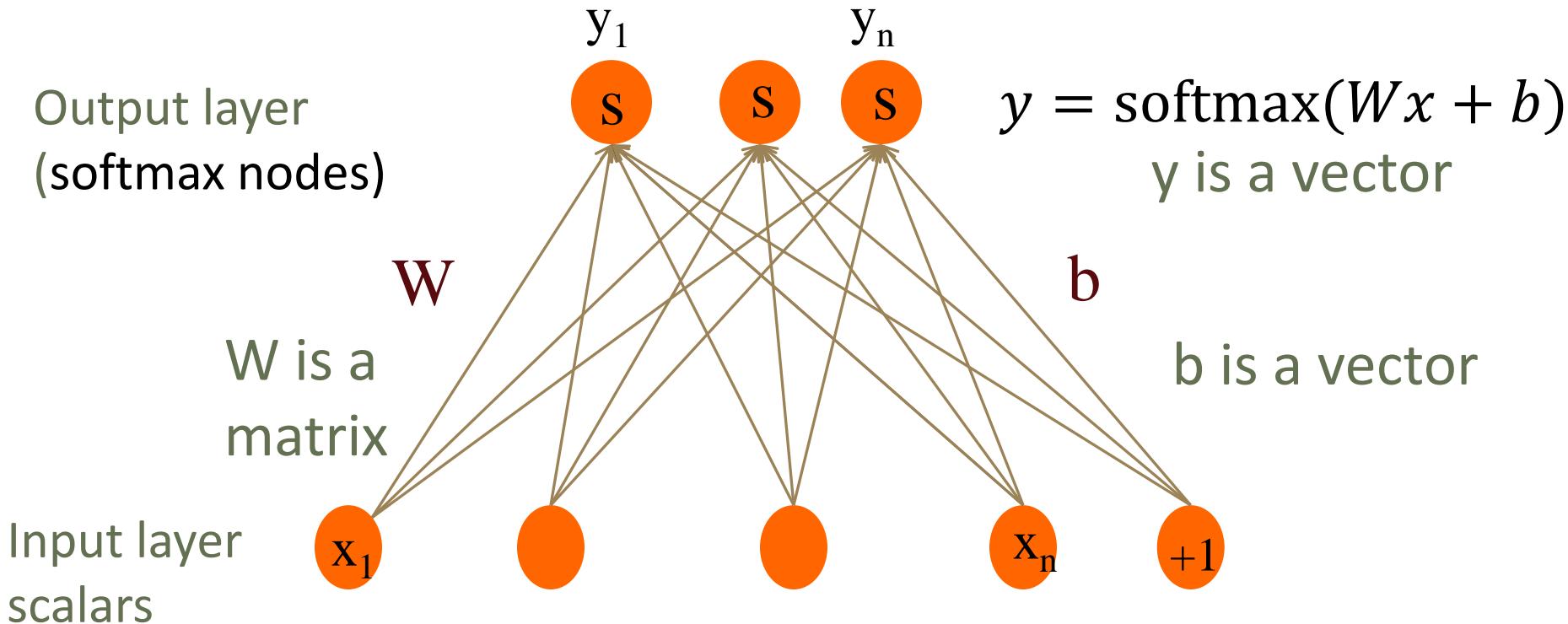
Binary Logistic Regression as a 1-layer Network

(we don't count the input layer in counting layers!)



Multinomial Logistic Regression as a 1-layer Network

Fully connected single layer network



Reminder: softmax: a generalization of sigmoid

For a vector z of dimensionality k , the softmax is:

$$\text{softmax}(z) = \left[\frac{\exp(z_1)}{\sum_{i=1}^k \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^k \exp(z_i)}, \dots, \frac{\exp(z_k)}{\sum_{i=1}^k \exp(z_i)} \right]$$

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^k \exp(z_j)} \quad 1 \leq i \leq k$$

Example:

$$z = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]$$

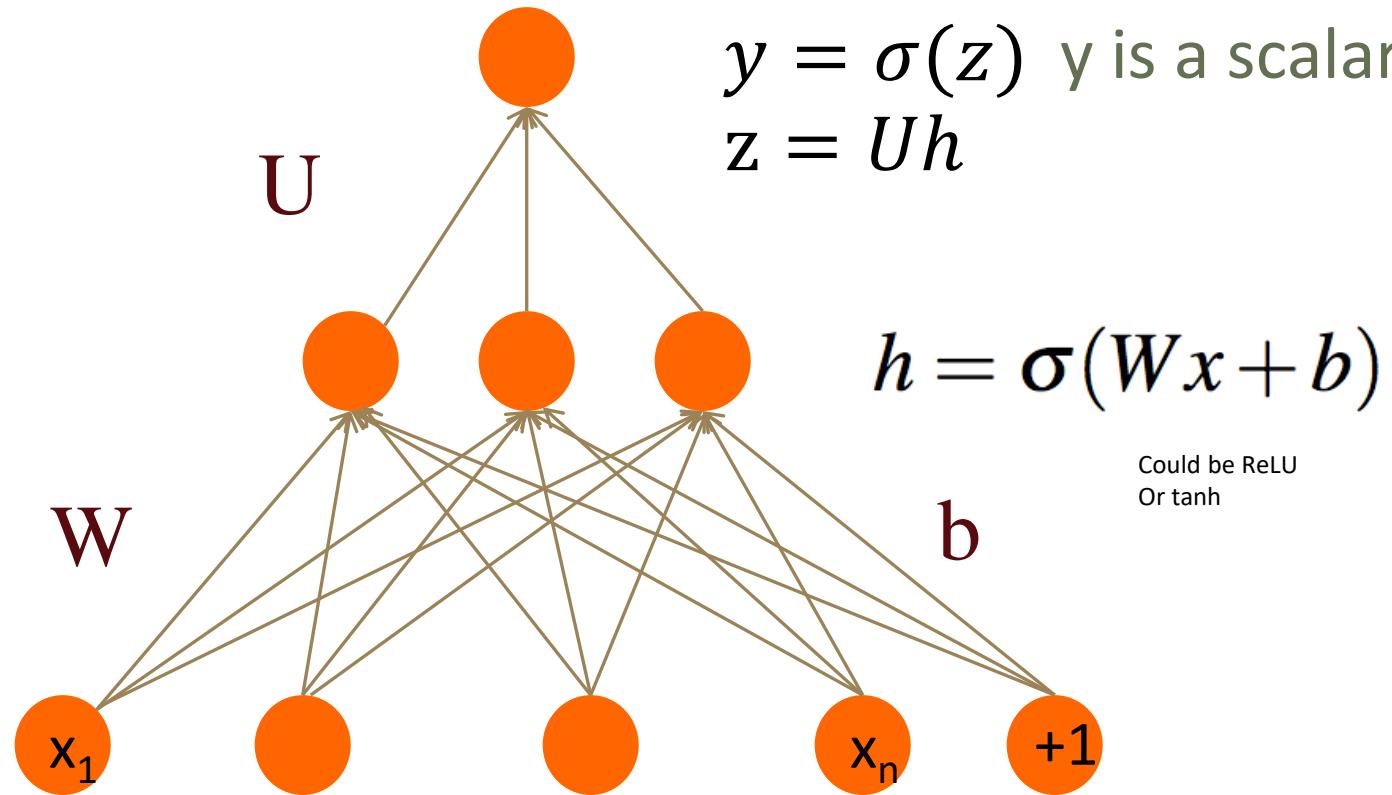
$$\text{softmax}(z) = [0.055, 0.090, 0.006, 0.099, 0.74, 0.010]$$

Two-Layer Network with scalar output

Output layer
(σ node)

hidden units
(σ node)

Input layer
(vector)

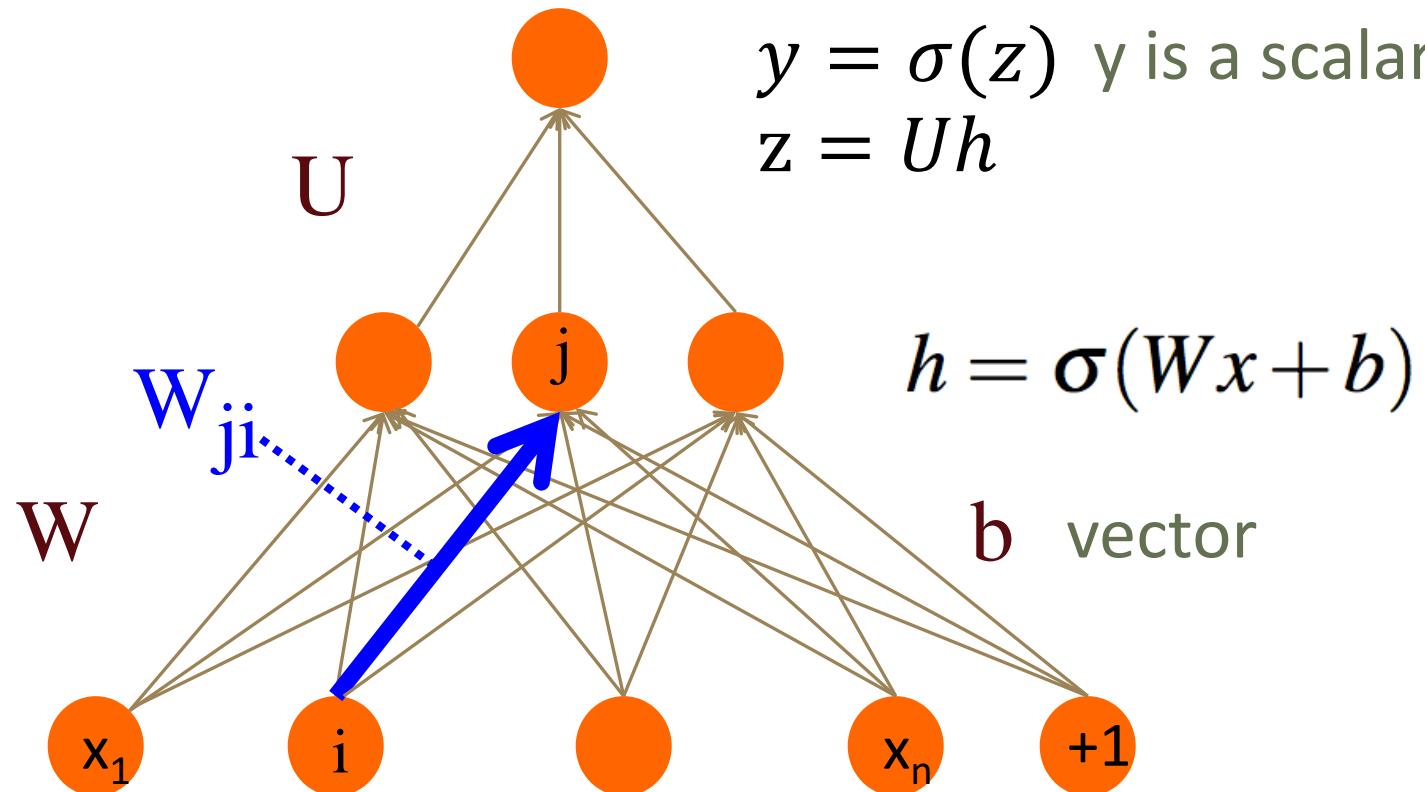


Two-Layer Network with scalar output

Output layer
(σ node)

hidden units
(σ node)

Input layer
(vector)

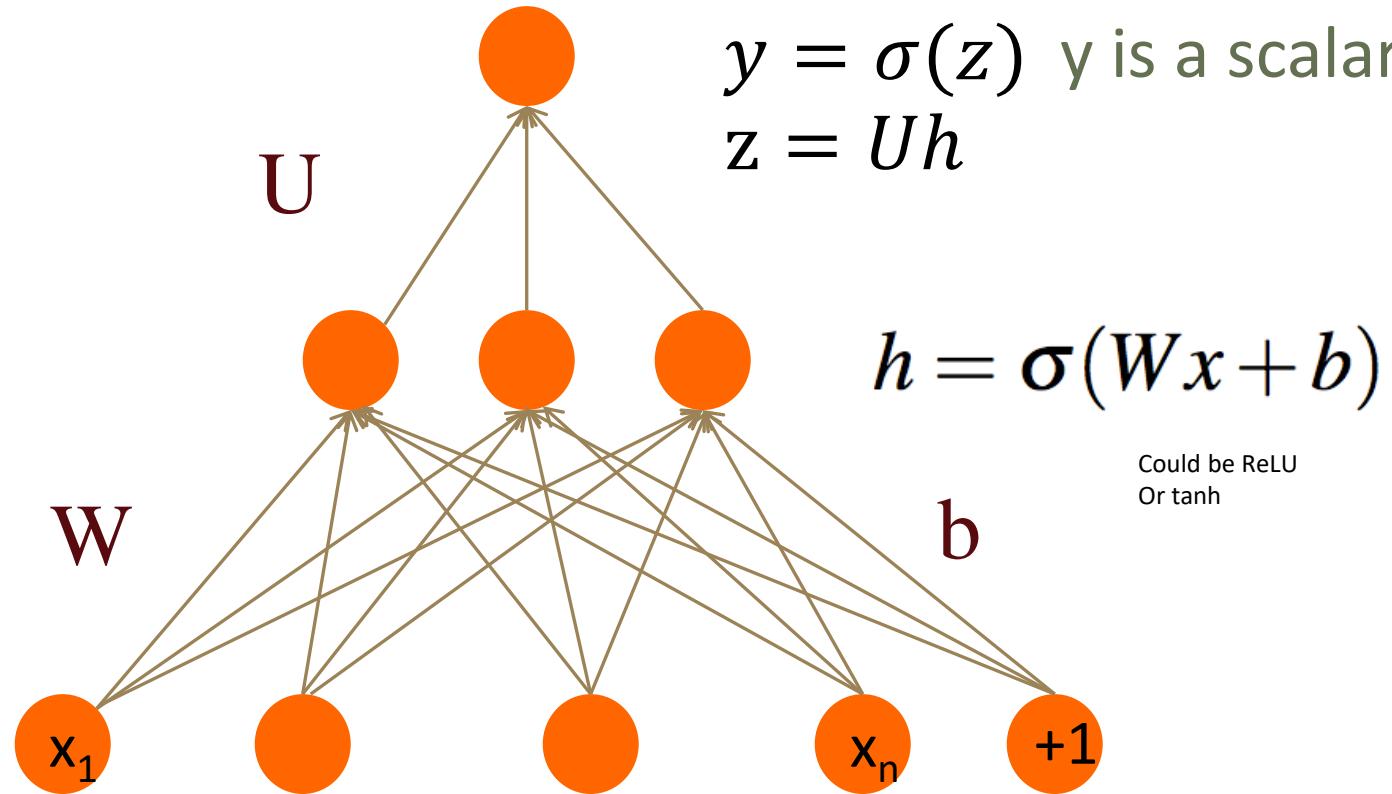


Two-Layer Network with scalar output

Output layer
(σ node)

hidden units
(σ node)

Input layer
(vector)

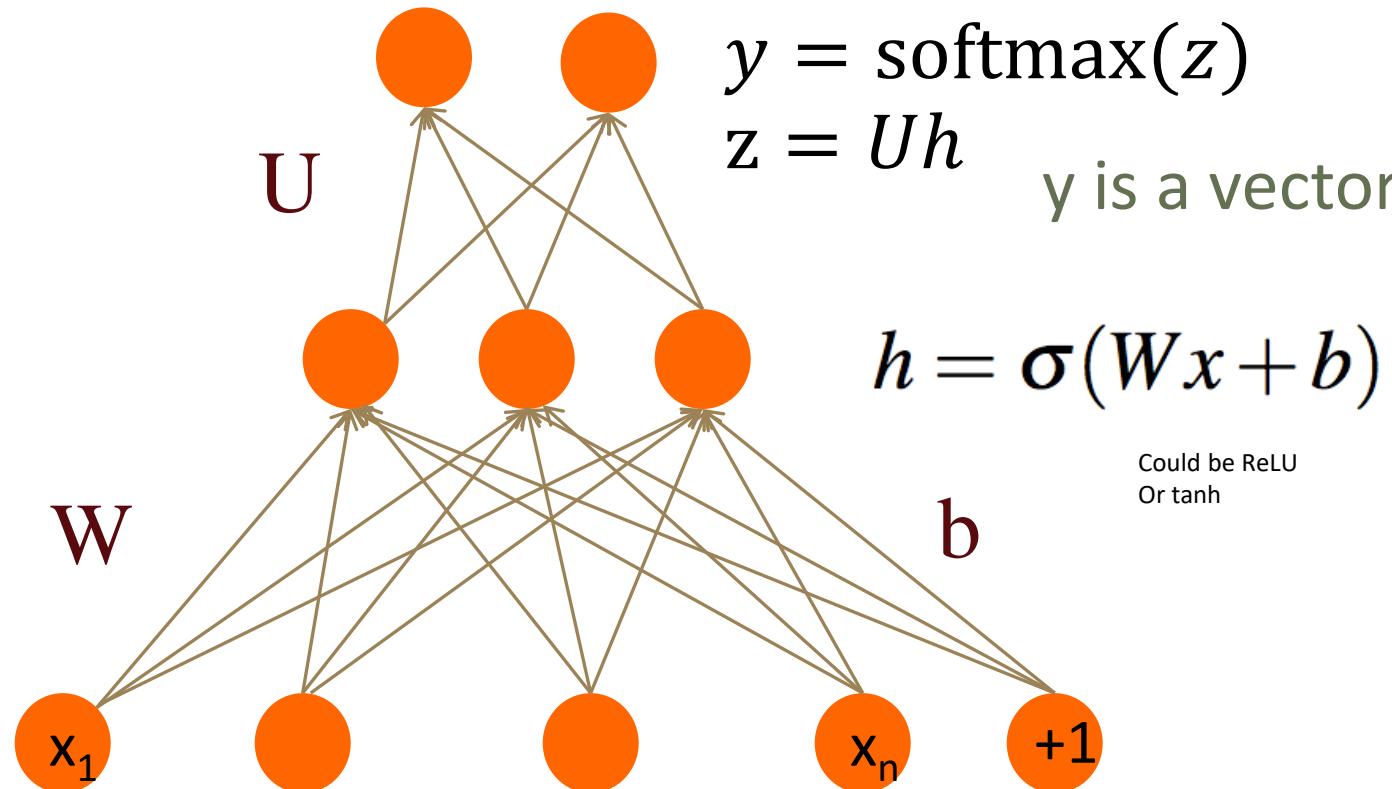


Two-Layer Network with softmax output

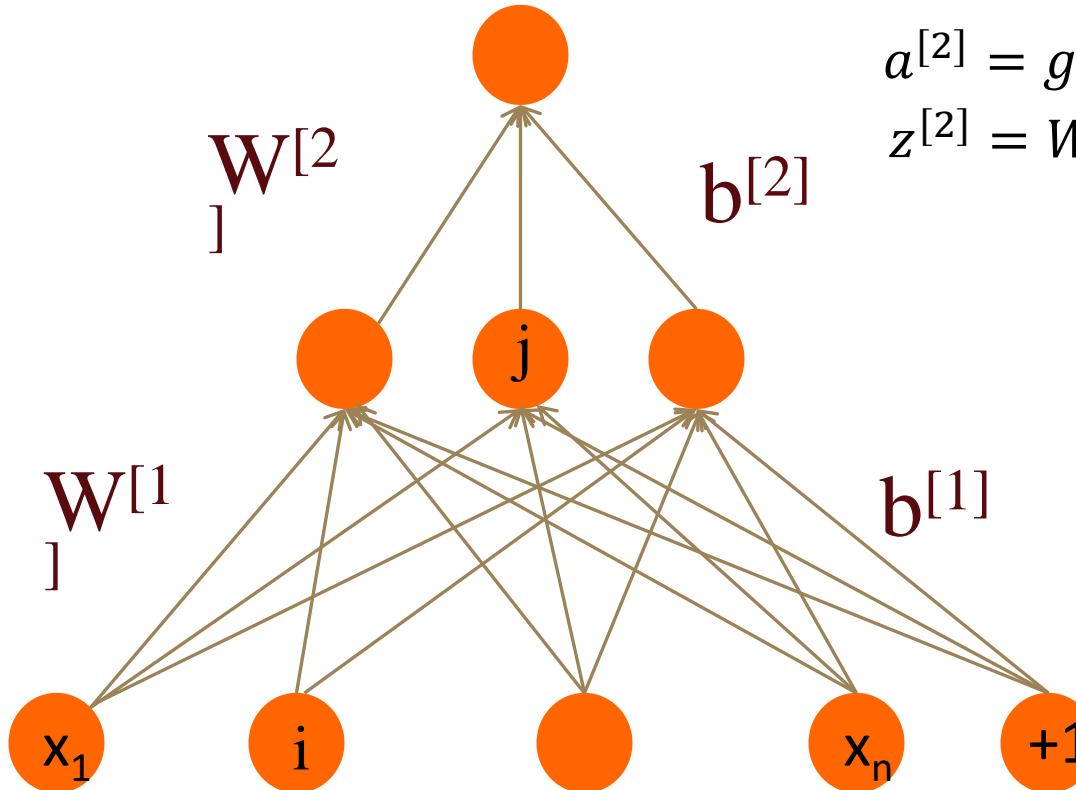
Output layer
(σ node)

hidden units
(σ node)

Input layer
(vector)



Multi-layer Notation



$$y = a^{[2]}$$

$$a^{[2]} = g^{[2]}(z^{[2]}) \quad \text{sigmoid or softmax}$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[1]} = g^{[1]}(z^{[1]}) \quad \text{ReLU}$$

$$z^{[1]} = W^{[1]}a^{[0]} + b^{[1]}$$

$$a^{[0]}$$

Multi Layer Notation

$$z^{[1]} = W^{[1]}a^{[0]} + b^{[1]}$$

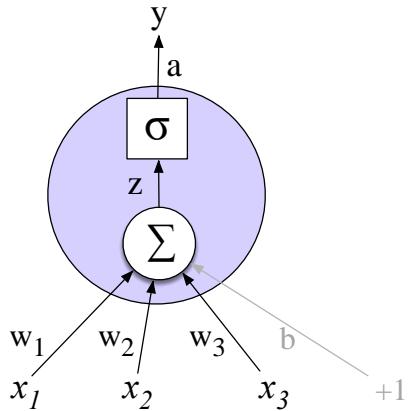
$$a^{[1]} = g^{[1]}(z^{[1]})$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = g^{[2]}(z^{[2]})$$

$$\hat{y} = a^{[2]}$$

for i in 1..n

$$z^{[i]} = W^{[i]} a^{[i-1]} + b^{[i]}$$
$$a^{[i]} = g^{[i]}(z^{[i]})$$
$$\hat{y} = a^{[n]}$$


Replacing the bias unit

Let's switch to a notation without the bias unit

Just a notational change

1. Add a dummy node $a_0=1$ to each layer
2. Its weight w_0 will be the bias
3. So input layer $a^{[0]}_0=1$,
 - And $a^{[1]}_0=1, a^{[2]}_0=1, \dots$

Replacing the bias unit

Instead of:

$$x = x_1, x_2, \dots, x_{n0}$$

$$h = \sigma(Wx + b)$$

$$h_j = \sigma \left(\sum_{i=1}^{n_0} W_{ji} x_i + b_j \right)$$

We'll do this:

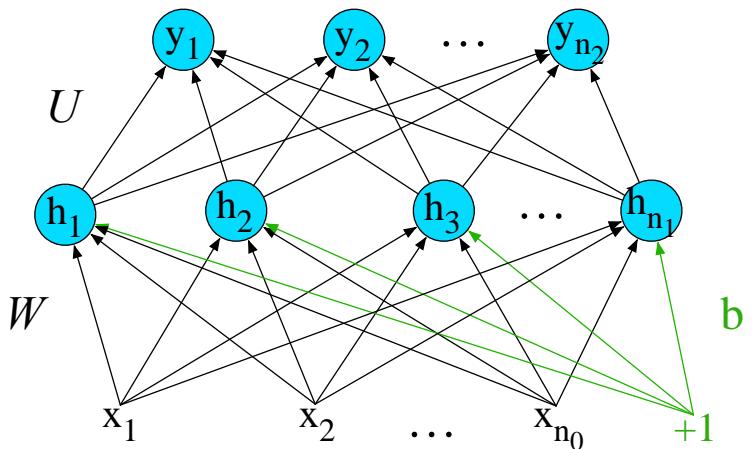
$$x = x_0, x_1, x_2, \dots, x_{n0}$$

$$h = \sigma(Wx)$$

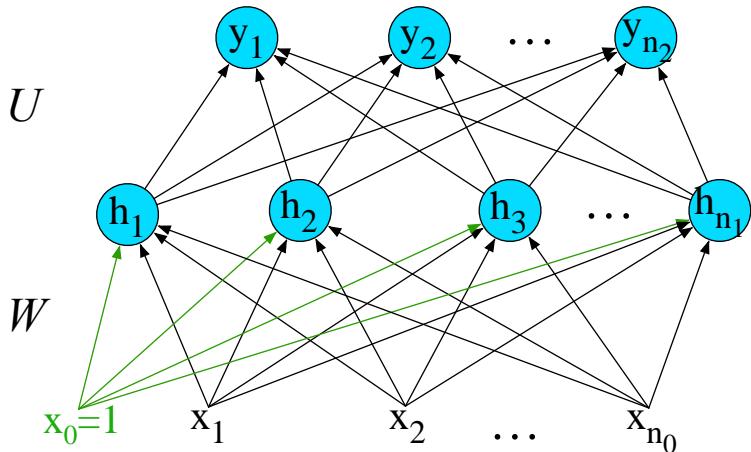
$$\sigma \left(\sum_{i=0}^{n_0} W_{ji} x_i \right)$$

Replacing the bias unit

Instead of:



We'll do this:



Simple Neural Networks and Neural Language Models

Feedforward Neural Networks

Simple Neural Networks and Neural Language Models

Applying feedforward networks to NLP tasks

Use cases for feedforward networks

Let's consider 2 (simplified) sample tasks:

1. Text classification
2. Language modeling

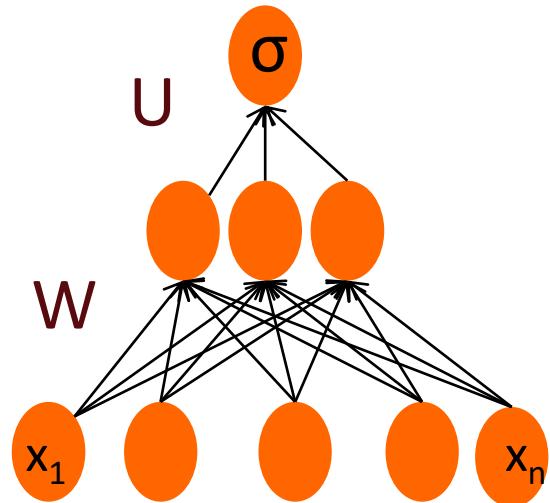
State of the art systems use more powerful neural architectures, but simple models are useful to consider!

Classification: Sentiment Analysis

We could do exactly what we did with logistic regression

Input layer are binary features as before

Output layer is 0 or 1

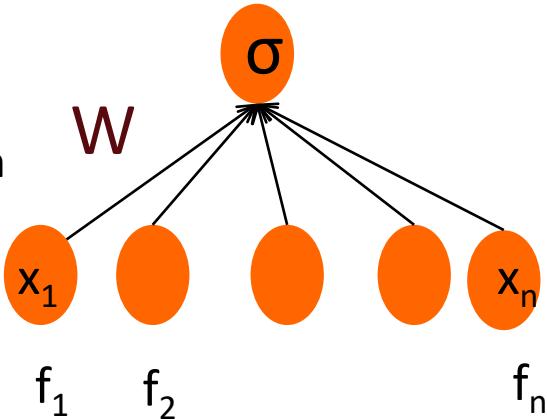


Sentiment Features

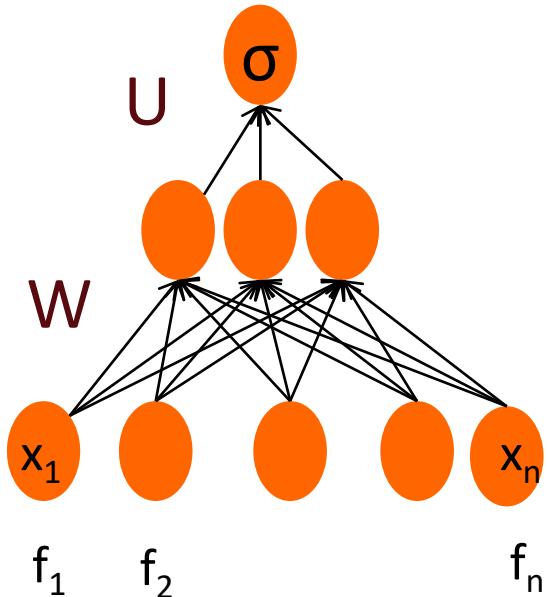
Var	Definition
x_1	count(positive lexicon) \in doc)
x_2	count(negative lexicon) \in doc)
x_3	$\begin{cases} 1 & \text{if “no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_4	count(1st and 2nd pronouns \in doc)
x_5	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_6	log(word count of doc)

Feedforward nets for simple classification

Logistic
Regression



2-layer
feedforward
network



Just adding a hidden layer to logistic regression

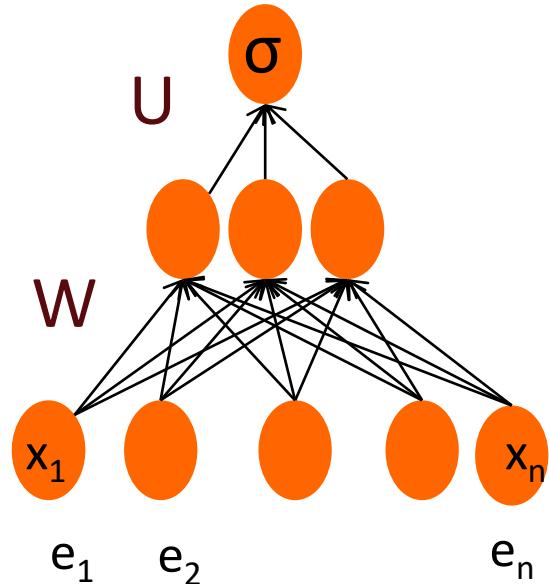
- allows the network to use non-linear interactions between features
- which may (or may not) improve performance.

Even better: representation learning

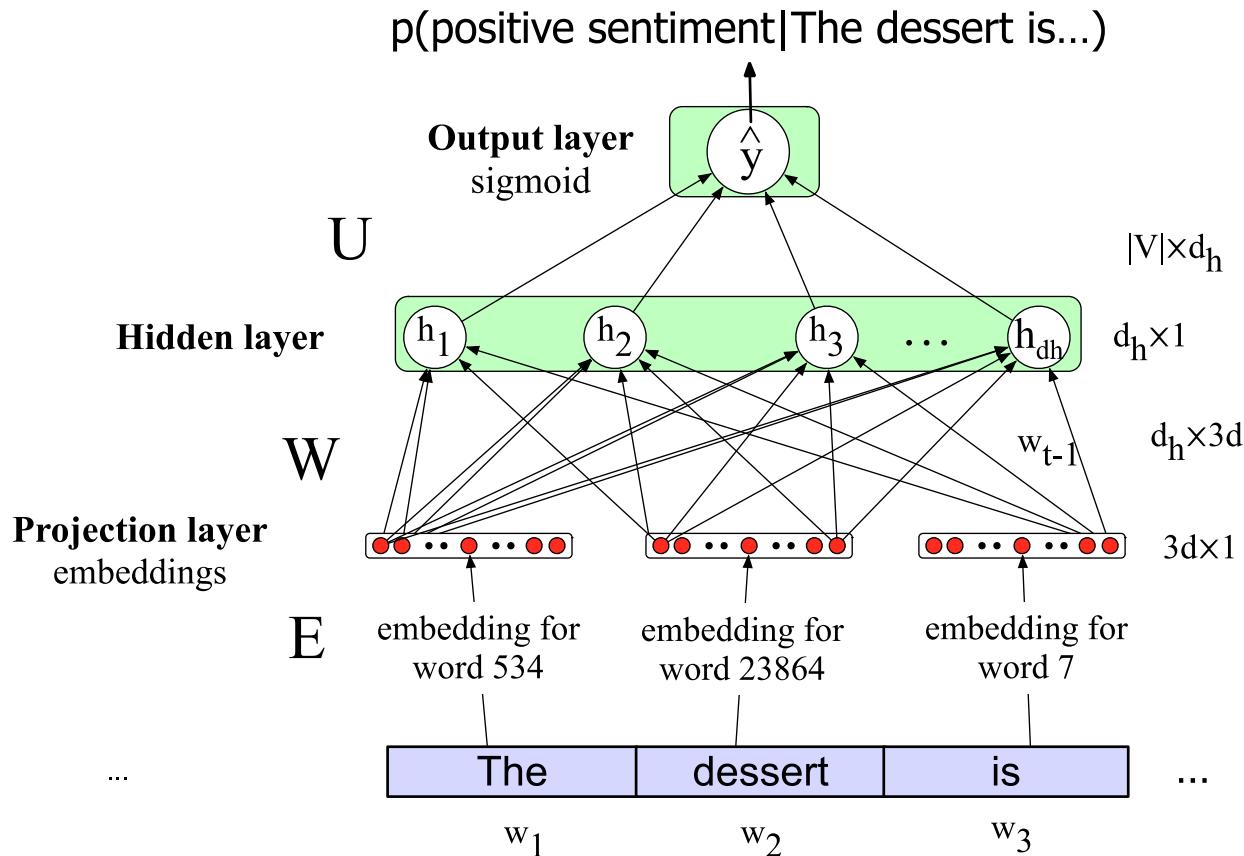
The real power of deep learning comes from the ability to **learn** features from the data

Instead of using hand-built human-engineered features for classification

Use learned representations like embeddings!



Neural Net Classification with embeddings as input features!



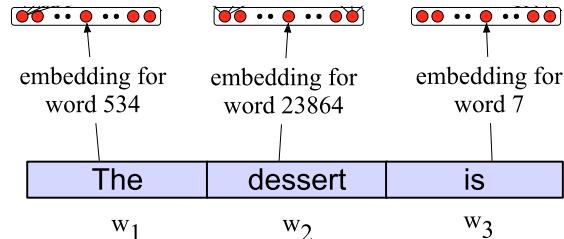
Issue: texts come in different sizes

This assumes a fixed size length (3)!

Kind of unrealistic.

Some simple solutions (more sophisticated solutions later)

1. Make the input the length of the longest review
 - If shorter then pad with zero embeddings
 - Truncate if you get longer reviews at test time
2. Create a single "sentence embedding" (the same dimensionality as a word) to represent all the words
 - Take the mean of all the word embeddings
 - Take the element-wise max of all the word embeddings
 - For each dimension, pick the max value from all words

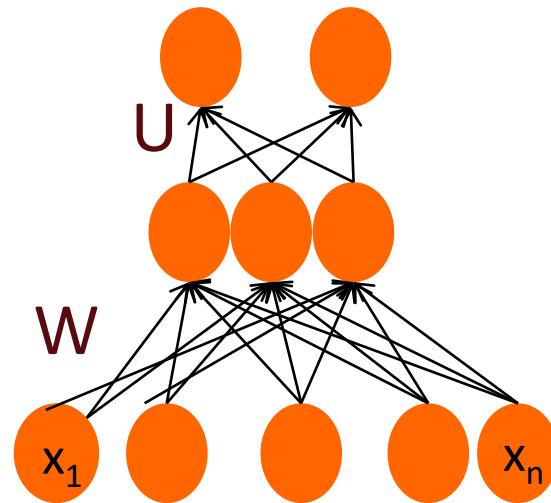


Reminder: Multiclass Outputs

What if you have more than two output classes?

- Add more output units (one for each class)
- And use a “softmax layer”

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^k e^{z_j}} \quad 1 \leq i \leq D$$



Neural Language Models (LMs)

Language Modeling: Calculating the probability of the next word in a sequence given some history.

- We've seen N-gram based LMs
- But neural network LMs far outperform n-gram language models

State-of-the-art neural LMs are based on more powerful neural network technology like Transformers

But **simple feedforward LMs** can do almost as well!

Simple feedforward Neural Language Models

Task: predict next word w_t

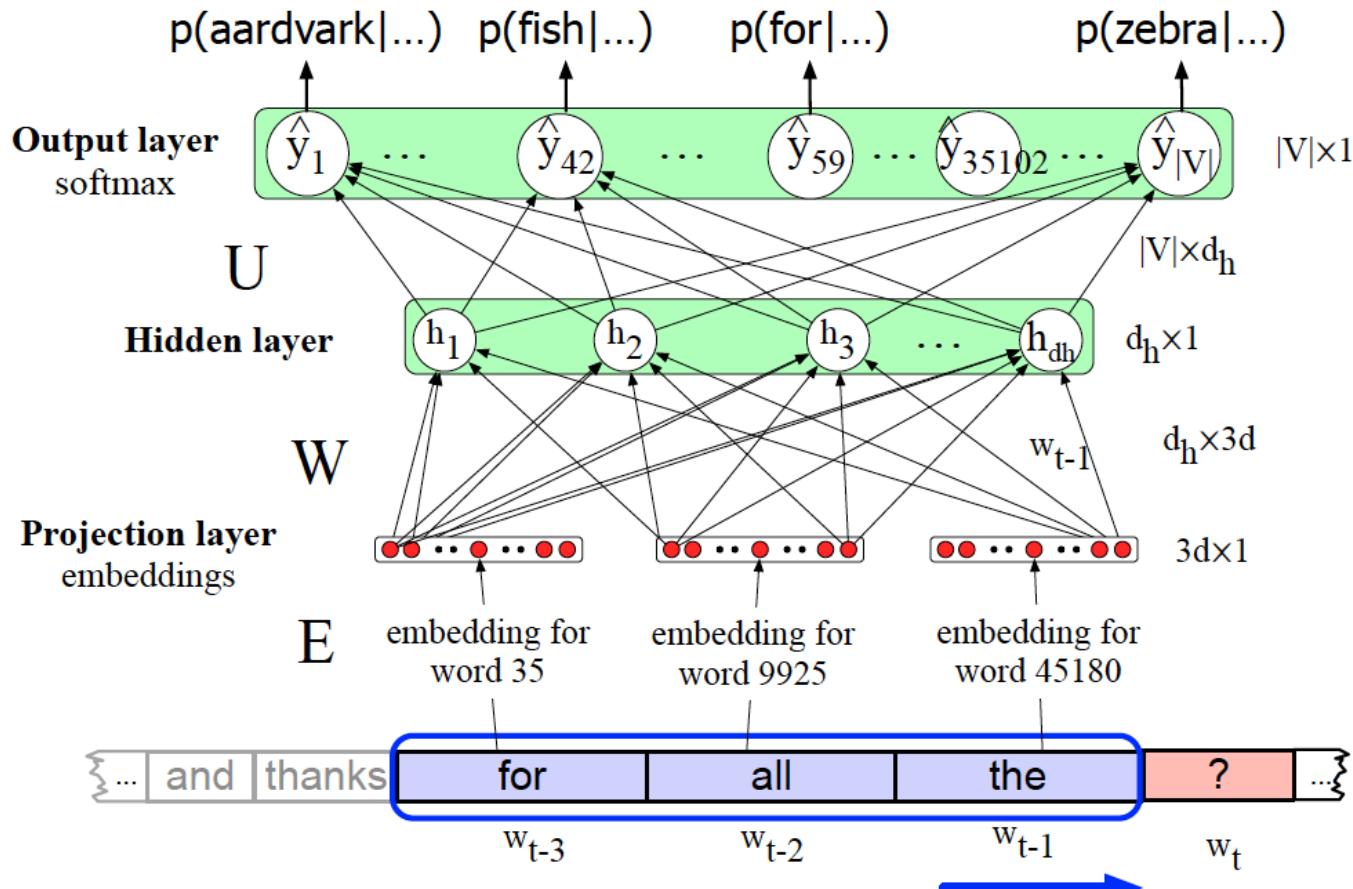
given prior words $w_{t-1}, w_{t-2}, w_{t-3}, \dots$

Problem: Now we're dealing with sequences of arbitrary length.

Solution: Sliding windows (of fixed length)

$$P(w_t | w_1^{t-1}) \approx P(w_t | w_{t-N+1}^{t-1})$$

Neural Language Model



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

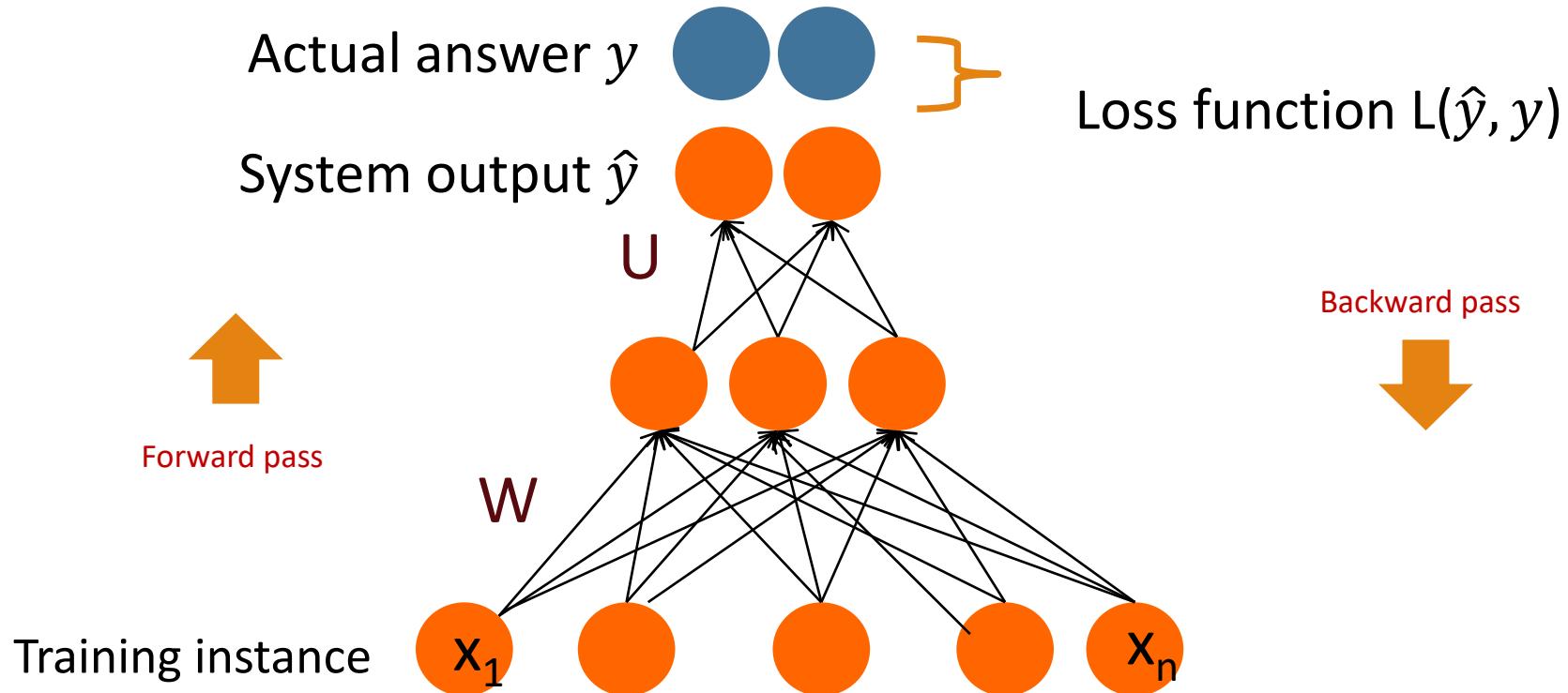
Simple Neural Networks and Neural Language Models

Applying feedforward networks to NLP tasks

Simple Neural Networks and Neural Language Models

Training Neural Nets: Overview

Intuition: training a 2-layer Network



Intuition: Training a 2-layer network

For every training tuple (x, y)

- Run *forward* computation to find our estimate \hat{y}
- Run *backward* computation to update weights:
 - For every output node
 - Compute loss L between true y and the estimated \hat{y}
 - For every weight w from hidden layer to the output layer
 - Update the weight
 - For every hidden node
 - Assess how much blame it deserves for the current answer
 - For every weight w from input layer to the hidden layer
 - Update the weight

Reminder: Loss Function for binary logistic regression

A measure for how far off the current answer is to the right answer

Cross entropy loss for logistic regression:

$$\begin{aligned} L_{\text{CE}}(\hat{y}, y) &= -\log p(y|x) = -[y \log \hat{y} + (1-y) \log(1-\hat{y})] \\ &= -[y \log \sigma(w \cdot x + b) + (1-y) \log(1 - \sigma(w \cdot x + b))] \end{aligned}$$

Reminder: gradient descent for weight updates

Use the derivative of the loss function with respect to weights $\frac{d}{dw} L(f(x; w), y)$

To tell us how to adjust weights for each training item

- Move them in the opposite direction of the gradient

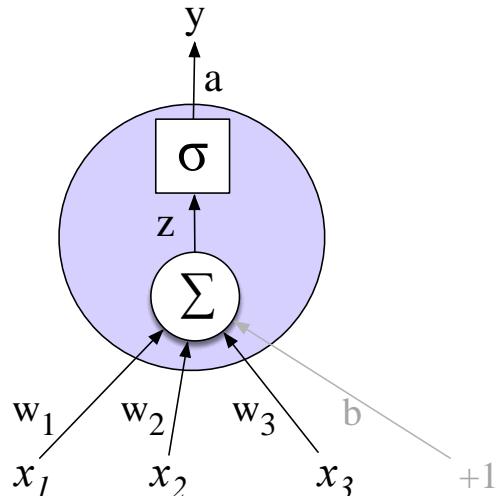
$$w^{t+1} = w^t - h \frac{d}{dw} L(f(x; w), y)$$

- For logistic regression

$$\frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$$

Where did that derivative come from?

Using the chain rule! $f(x) = u(v(x))$ $\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx}$
Intuition (see the text for details)



Derivative of the weighted sum

Derivative of the Activation

Derivative of the Loss

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial z} \frac{\partial z}{\partial w_i}$$

How can I find that gradient for every weight in the network?

These derivatives on the prior slide only give the updates for one weight layer: the last one!

What about deeper networks?

- Lots of layers, different activation functions?

Solution in the next lecture:

- Even more use of the chain rule!!
- Computation graphs and backward differentiation!

Simple Neural Networks and Neural Language Models

Training Neural Nets: Overview

Simple Neural
Networks and
Neural
Language
Models

Computation Graphs and Backward Differentiation

Why Computation Graphs

For training, we need the derivative of the loss with respect to each weight in every layer of the network

- But the loss is computed only at the very end of the network!

Solution: **error backpropagation** (Rumelhart, Hinton, Williams, 1986)

- **Backprop** is a special case of **backward differentiation**
- Which relies on **computation graphs**.

Computation Graphs

A computation graph represents the process of computing a mathematical expression

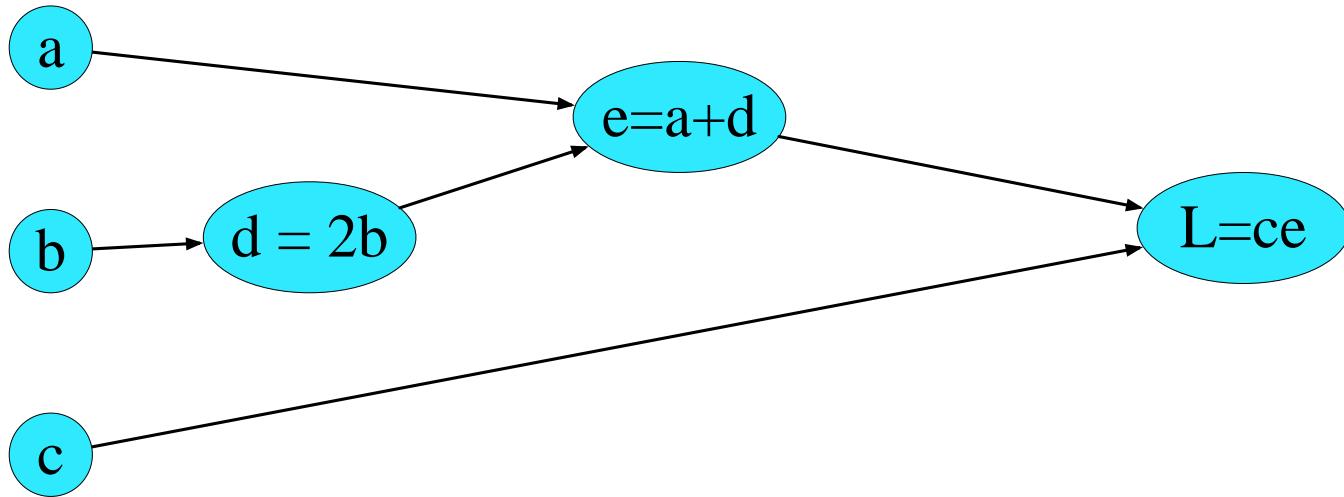
Example: $L(a,b,c) = c(a + 2b)$

$$d = 2 * b$$

Computations:

$$e = a + d$$

$$L = c * e$$



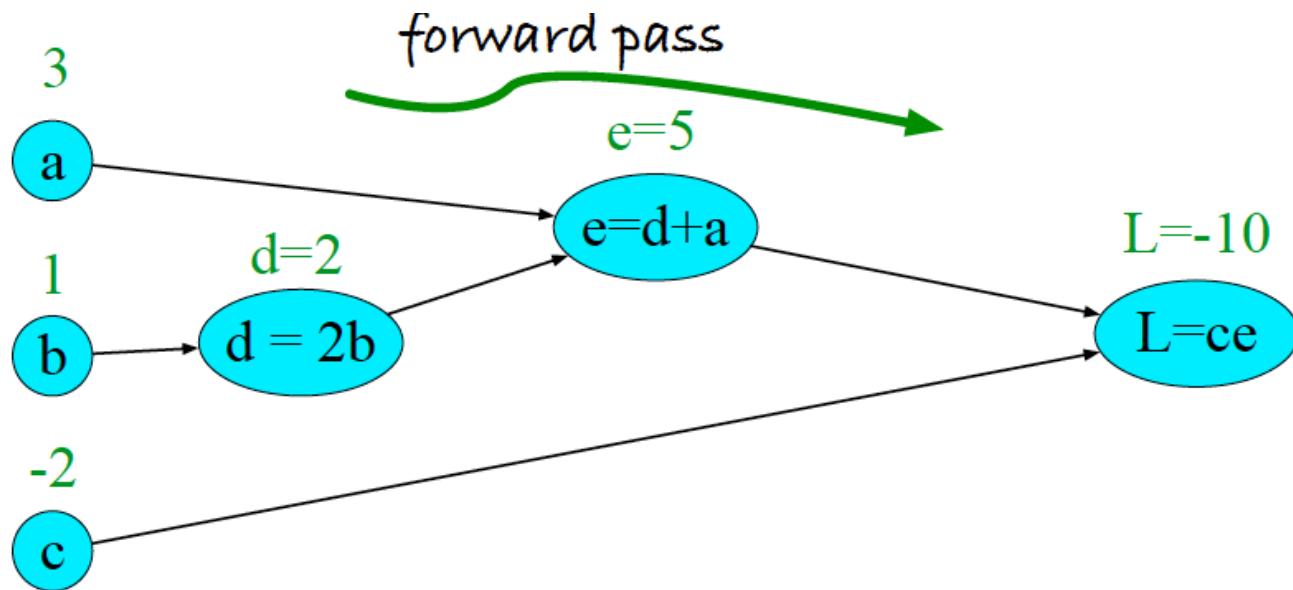
Example: $L(a, b, c) = c(a + 2b)$

$$d = 2 * b$$

Computations:

$$e = a + d$$

$$L = c * e$$



Backwards differentiation in computation graphs

The importance of the computation graph comes from the backward pass

This is used to compute the derivatives that we'll need for the weight update.

Example $L(a, b, c) = c(a + 2b)$

$$d = 2 * b$$

$$e = a + d$$

$$L = c * e$$

We want: $\frac{\partial L}{\partial a}$, $\frac{\partial L}{\partial b}$, and $\frac{\partial L}{\partial c}$

The derivative $\frac{\partial L}{\partial a}$, tells us how much a small change in a affects L .

The chain rule

Computing the derivative of a composite function:

$$f(x) = u(v(x))$$

$$\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx}$$

$$f(x) = u(v(w(x)))$$

$$\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dw} \cdot \frac{dw}{dx}$$

Example $L(a, b, c) = c(a + 2b)$

$$d = 2 * b$$

$$e = a + d$$

$$L = c * e$$

$$\frac{\partial L}{\partial c} = e$$

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e} \frac{\partial e}{\partial a}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e} \frac{\partial e}{\partial d} \frac{\partial d}{\partial b}$$

Example

$$L(a, b, c) = c(a + 2b)$$

$$d = 2 * b$$

$$e = a + d$$

$$L = c * e$$

$$\begin{aligned}\frac{\partial L}{\partial a} &= \frac{\partial L}{\partial e} \frac{\partial e}{\partial a} \\ \frac{\partial L}{\partial b} &= \frac{\partial L}{\partial e} \frac{\partial e}{\partial d} \frac{\partial d}{\partial b}\end{aligned}$$

$$L = ce : \quad \frac{\partial L}{\partial e} = c, \frac{\partial L}{\partial c} = e$$

$$e = a + d : \quad \frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial d} = 1$$

$$d = 2b : \quad \frac{\partial d}{\partial b} = 2$$

Example

$$a=3$$



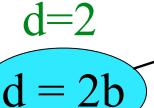
$$b=1$$



$$c=-2$$

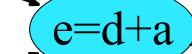


$$d = 2b$$

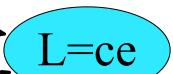


$$e=5$$

$$e=d+a$$



$$L=ce$$



$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e} \frac{\partial e}{\partial a}$$

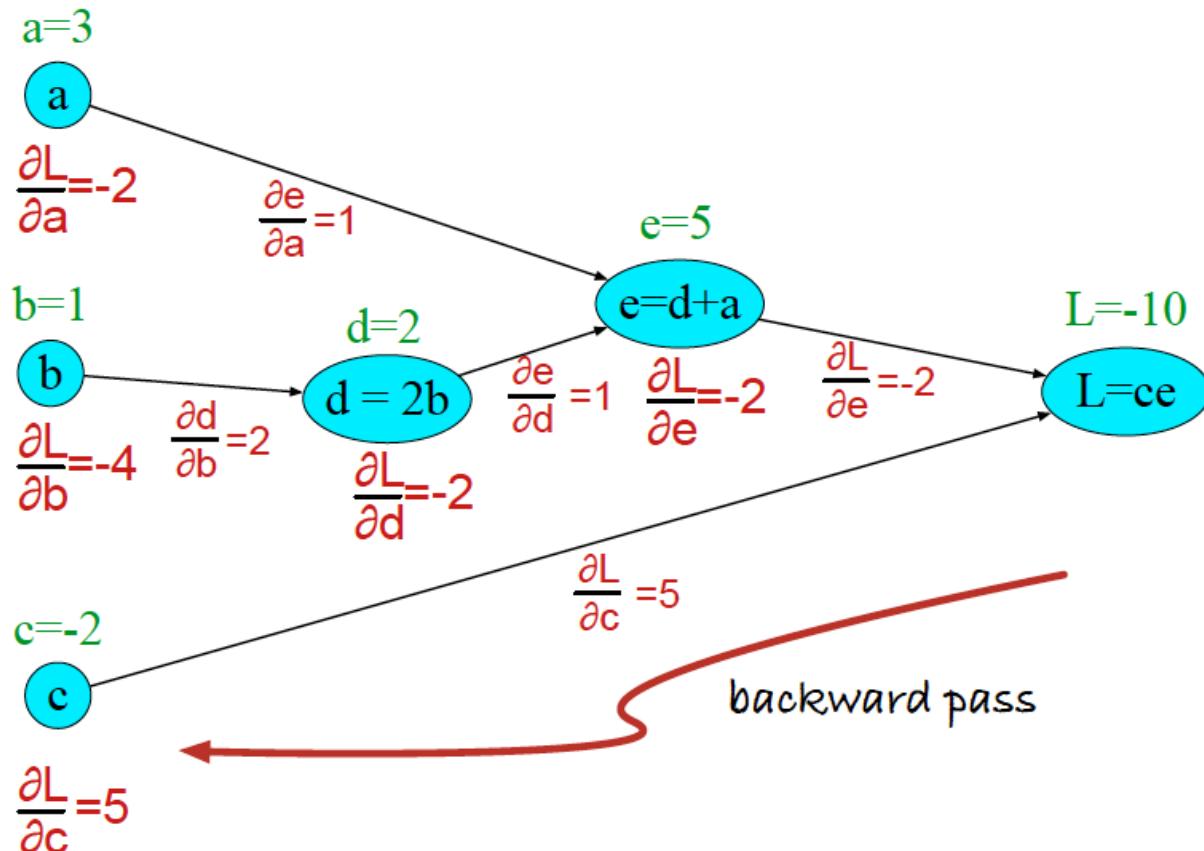
$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e} \frac{\partial e}{\partial d} \frac{\partial d}{\partial b}$$

$$L=ce : \quad \frac{\partial L}{\partial e} = c, \frac{\partial L}{\partial c} = e$$

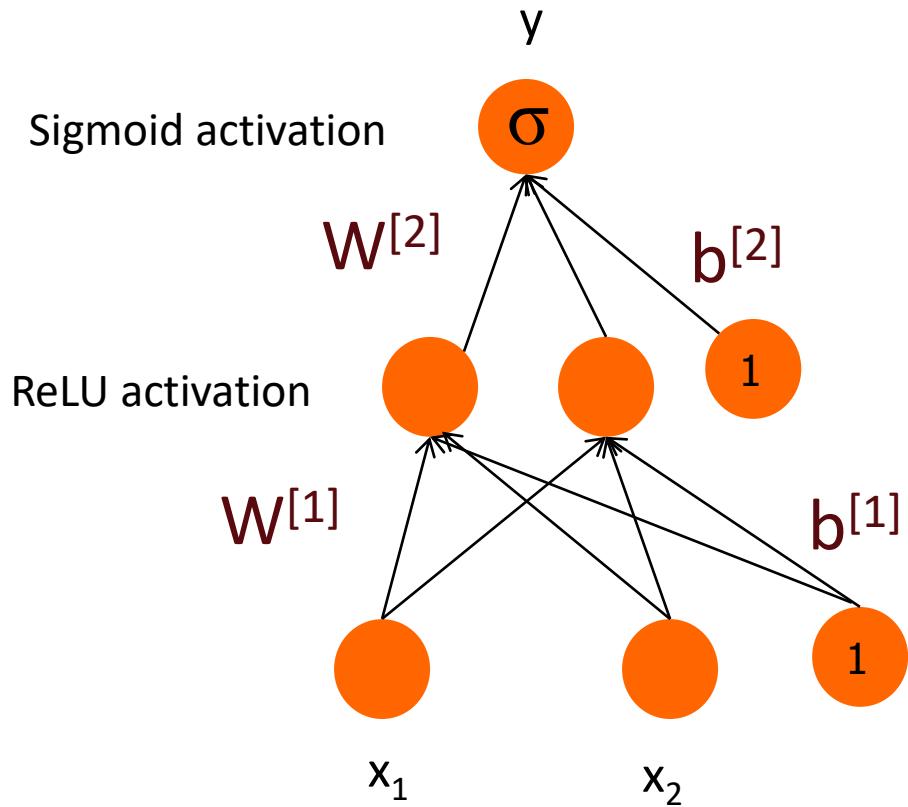
$$e=a+d : \quad \frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial d} = 1$$

$$d=2b : \quad \frac{\partial d}{\partial b} = 2$$

Example



Backward differentiation on a two layer network



$$z^{[1]} = W^{[1]} \mathbf{x} + b^{[1]}$$

$$a^{[1]} = \text{ReLU}(z^{[1]})$$

$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$

$$a^{[2]} = \sigma(z^{[2]})$$

$$\hat{y} = a^{[2]}$$

Backward differentiation on a two layer network

$$z^{[1]} = W^{[1]} \mathbf{x} + b^{[1]}$$

$$a^{[1]} = \text{ReLU}(z^{[1]})$$

$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$

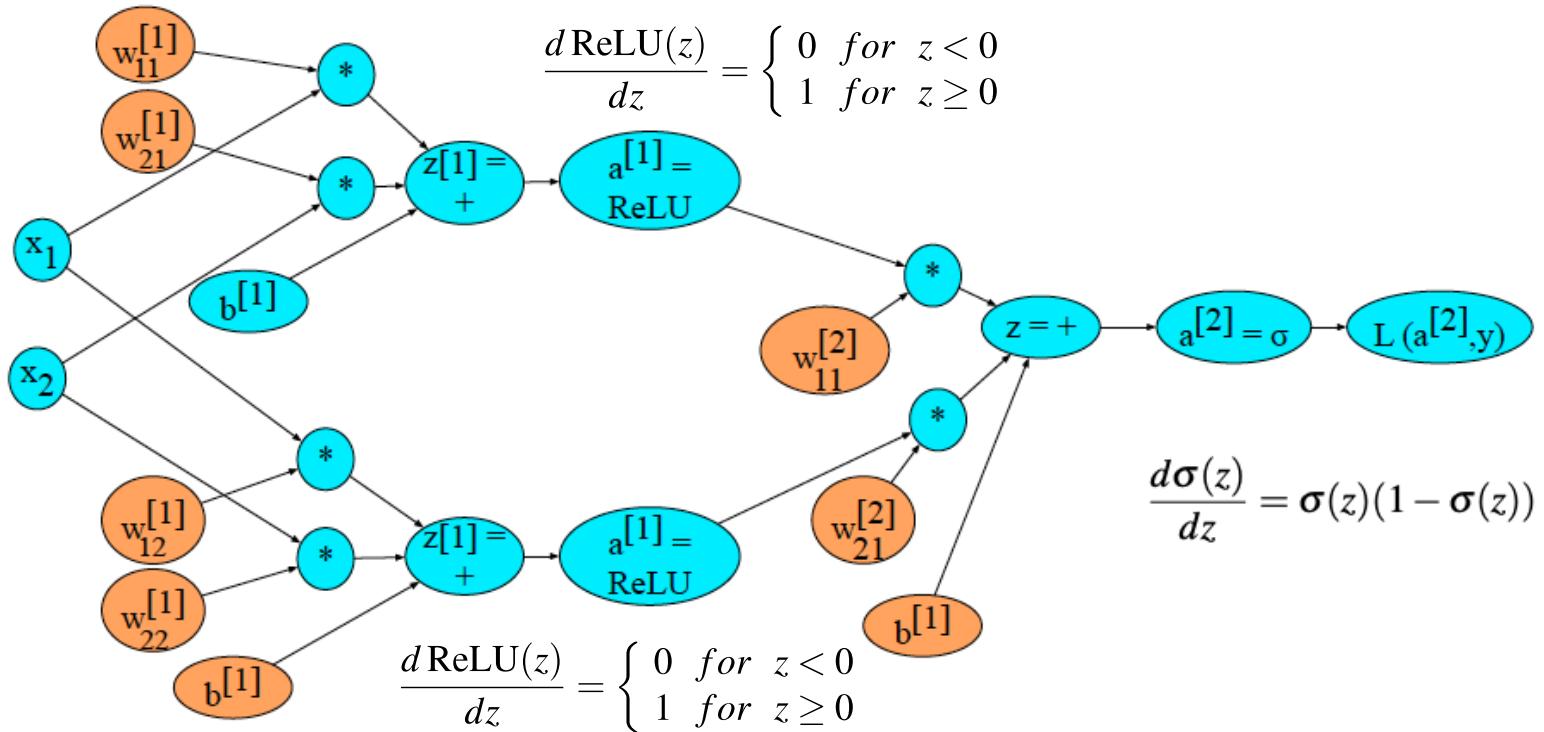
$$a^{[2]} = \sigma(z^{[2]})$$

$$\hat{y} = a^{[2]}$$

$$\frac{d \text{ReLU}(z)}{dz} = \begin{cases} 0 & \text{for } z < 0 \\ 1 & \text{for } z \geq 0 \end{cases}$$

$$\frac{d\sigma(z)}{dz} = \sigma(z)(1 - \sigma(z))$$

Backward differentiation on a 2-layer network



Starting off the backward pass: $\frac{\partial L}{\partial z}$

(I'll write a for $a^{[2]}$ and z for $z^{[2]}$)

$$\begin{aligned} z^{[1]} &= W^{[1]} \mathbf{x} + b^{[1]} \\ a^{[1]} &= \text{ReLU}(z^{[1]}) \\ z^{[2]} &= W^{[2]} a^{[1]} + b^{[2]} \\ a^{[2]} &= \sigma(z^{[2]}) \\ \hat{y} &= a^{[2]} \end{aligned}$$

$$L(\hat{y}, y) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

$$L(a, y) = -(y \log a + (1 - y) \log(1 - a))$$

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial a} \frac{\partial a}{\partial z}$$

$$\begin{aligned} \frac{\partial L}{\partial a} &= - \left(\left(y \frac{\partial \log(a)}{\partial a} \right) + (1 - y) \frac{\partial \log(1 - a)}{\partial a} \right) \\ &= - \left(\left(y \frac{1}{a} \right) + (1 - y) \frac{1}{1 - a} (-1) \right) = - \left(\frac{y}{a} + \frac{y - 1}{1 - a} \right) \end{aligned}$$

$$\frac{\partial a}{\partial z} = a(1 - a) \quad \frac{\partial L}{\partial z} = - \left(\frac{y}{a} + \frac{y - 1}{1 - a} \right) a(1 - a) = a - y$$

Summary

For training, we need the derivative of the loss with respect to weights in early layers of the network

- But loss is computed only at the very end of the network!

Solution: **backward differentiation**

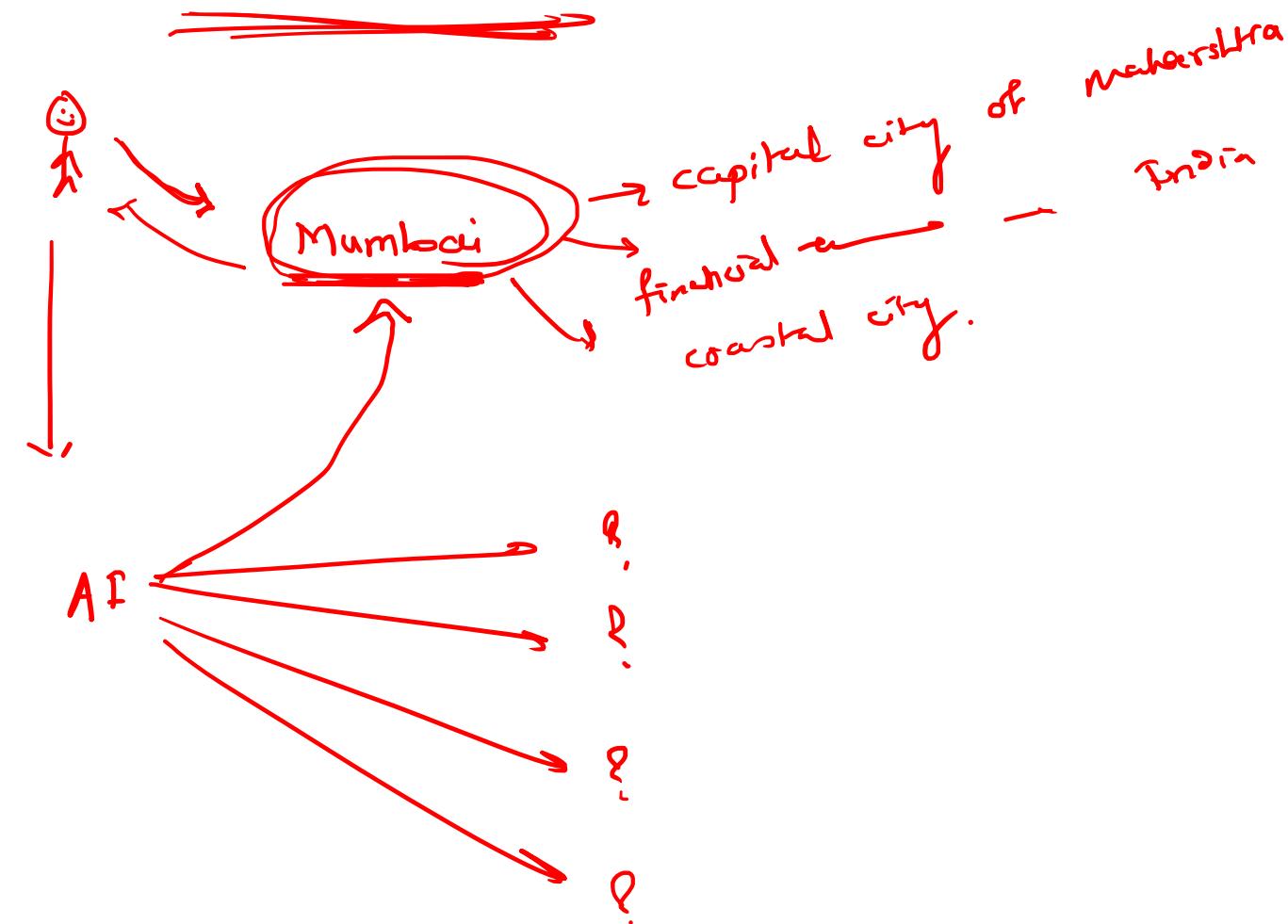
Given a computation graph and the derivatives of all the functions in it we can automatically compute the derivative of the loss with respect to these early weights.

Simple Neural
Networks and
Neural
Language
Models

Computation Graphs and
Backward Differentiation

Vector Semantics & Embeddings

Word Meaning



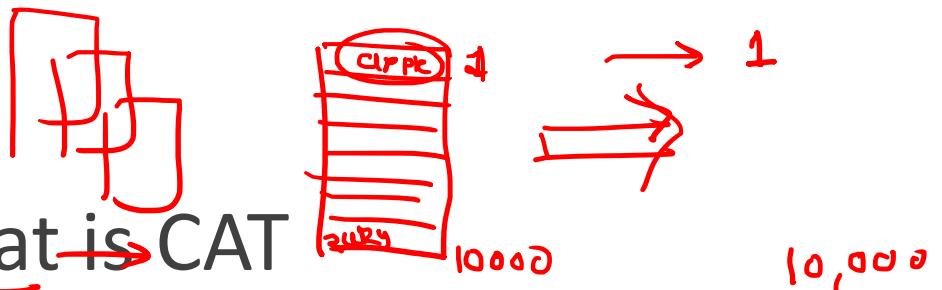
What do words mean?

N-gram or text classification methods we've seen so far

- Words are just strings (or indices w_i in a vocabulary list)
- That's not very satisfactory!

Introductory logic classes:

- The meaning of "dog" is DOG; cat is CAT
 $\forall x \text{ DOG}(x) \rightarrow \text{MAMMAL}(x)$



Old linguistics joke by Barbara Partee in 1967:

- Q: What's the meaning of life?
◦ A: LIFE

That seems hardly better!

Desiderata

What should a theory of word meaning do for us?

Let's look at some desiderata

From lexical semantics, the linguistic study of word meaning

Lemmas and senses

lemma

mouse (N)

1. any of numerous small rodents...

sense

2. a hand-operated device that controls
a cursor...

Modified from the online thesaurus WordNet

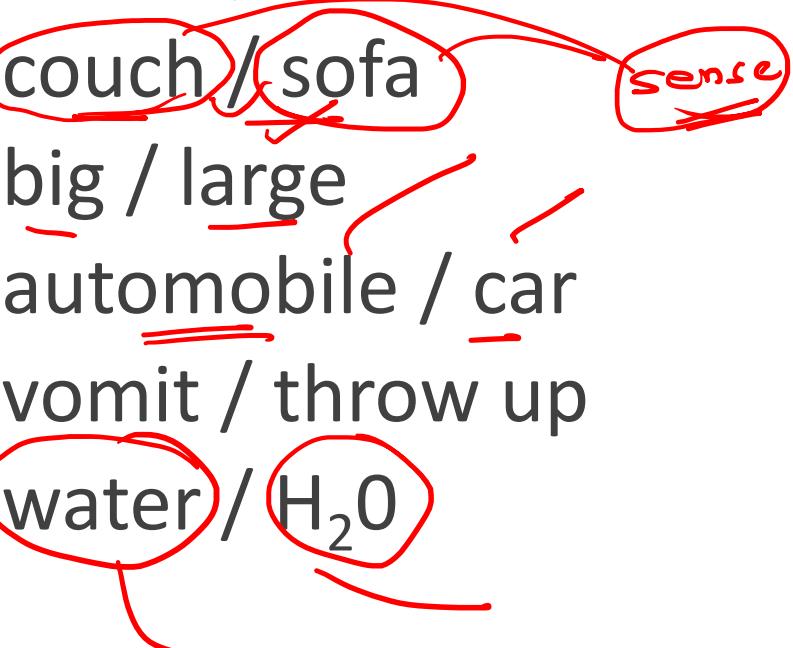
A **sense** or “**concept**” is the meaning component of a word
Lemmas can be **polysemous** (have multiple senses)

Relations between senses: Synonymy

Synonyms have the same meaning in some or all contexts.

- filbert / hazelnut
- couch / sofa
- big / large
- automobile / car
- vomit / throw up
- water / H_2O

sense



Relations between senses: Synonymy

Note that there are probably no examples of perfect synonymy.

- Even if many aspects of meaning are identical
- Still may differ based on politeness, slang, register, genre, etc.

Relation: Synonymy?

water/H₂O

"H₂O" in a surfing guide?

big/large

my big sister != my large sister

The Linguistic Principle of Contrast

Difference in form → difference in meaning



Abbé Gabriel Girard 1718

Re: "exact" synonyms

"je ne crois pas qu'il y ait de mot synonyme dans aucune Langue."

[I do not believe that there is a synonymous word in any language]

LA JUSTESSE
DE LA
LANGUE FRANÇOISE,
ou
LES DIFFERENTES SIGNIFICATIONS
DES MOTS QUI PASSENT
POUR
SYNONIMES.

Par M. l'Abbé GIRARD C. D. M. D. D. B.



A PARIS,
Chez LAURENT d'HOURY, Imprimeur-
Libraire, au bas de la rue de la Harpe, vis-
à vis la rue S. Severin, au Saint-Esprit.

M. DCC. XVIII.
Avec Approbation & Privilegs du Roy.

Relation: Similarity

Words with similar meanings. Not synonyms, but sharing some element of meaning



cow, horse

— —

Ask humans how similar 2 words are

word1	word2	similarity
vanish ↛	disappear →	<u>9.8</u>
behave ↛	obey →	7.3
belief	impression →	5.95
<u>muscle</u> ↛	<u>bone</u> →	<u>3.65</u>
modest	flexible →	0.98
hole ↛	agreement →	<u>0.3</u>

SimLex-999 dataset (Hill et al., 2015)

Relation: Word relatedness

Also called "word association"

Words can be related in any way, perhaps via a semantic frame or field

- coffee ↔ tea: **similar**
- coffee ↔ cup: **related**, not similar

Semantic field

Words that

- cover a particular semantic domain
- bear structured relations with each other.

hospitals

{surgeon, scalpel, nurse, anaesthetic, hospital}

restaurants

waiter, menu, plate, food, menu, chef

houses

{door, roof, kitchen, family, bed}

Relation: Antonymy

Senses that are opposites with respect to only one feature of meaning

Otherwise, they are very similar!

dark/light

short/long

fast/slow

rise/fall

hot/cold

up/down

in/out

More formally: antonyms can

- define a binary opposition or be at opposite ends of a scale
 - long/short, fast/slow
- Be *reversives*:
 - rise/fall, up/down

Connotation (sentiment)

- Words have **affective** meanings
 - Positive connotations (happy)
 - Negative connotations (sad)
- Connotations can be subtle:
 - Positive connotation: *copy, replica, reproduction*
 - Negative connotation: *fake, knockoff, forgery*
- Evaluation (sentiment!)
 - Positive evaluation (great, love)
 - Negative evaluation (terrible, hate)

Connotation

Osgood et al. (1957)

Words seem to vary along 3 affective dimensions:

- **valence**: the pleasantness of the stimulus
- **arousal**: the intensity of emotion provoked by the stimulus
- **dominance**: the degree of control exerted by the stimulus

	Word	Score		Word	Score
Valence	love	1.000		toxic	0.008
	happy	1.000		nightmare	0.005
Arousal	elated	0.960		mellow	0.069
	frenzy	0.965		napping	0.046
Dominance	powerful	0.991		weak	0.045
	leadership	0.983		empty	0.081

So far

Concepts or word senses

- Have a complex many-to-many association with **words** (homonymy, multiple senses)

Have relations with each other

- Synonymy
- Antonymy
- Similarity
- Relatedness
- Connotation

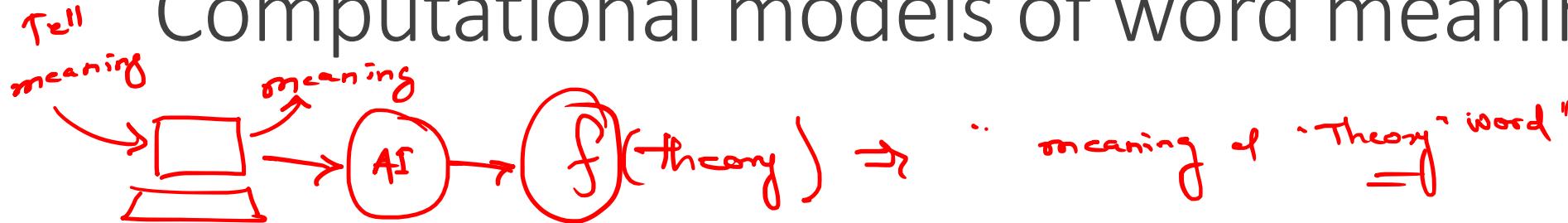
Word Meaning

Vector
Semantics &
Embeddings

Vector Semantics

Vector
Semantics &
Embeddings

Computational models of word meaning



Can we build a theory of how to represent word meaning, that accounts for at least some of the desiderata?

We'll introduce vector semantics

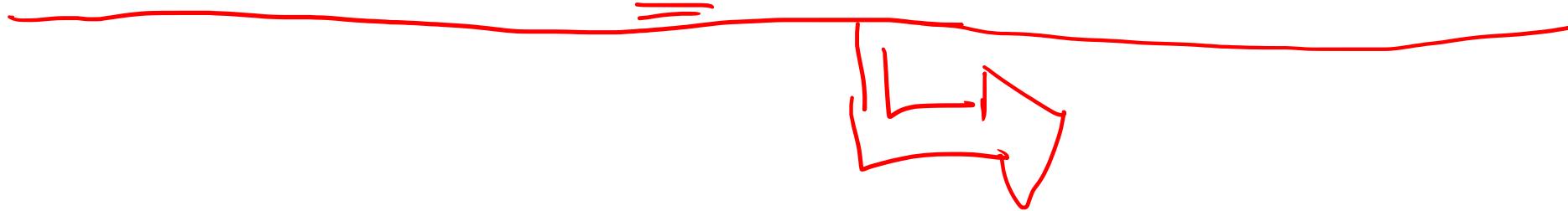
The standard model in language processing!

Handles many of our goals!

Ludwig Wittgenstein

PI #43:

"The meaning of a word is its use in the language"

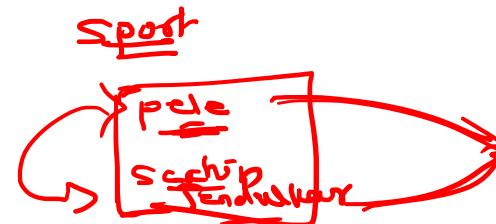


Let's define words by their usages

One way to define "usage":

words are defined by their environments (the words around them)

Zellig Harris (1954):



If A and B have almost identical environments we say that they are synonyms.

What does recent English borrowing *ongchoi* mean?

Suppose you see these sentences:

- Ong choi is delicious sautéed with garlic.
- Ong choi is superb over rice
- Ong choi leaves with salty sauces

And you've also seen these:

- ...spinach **sautéed with garlic over rice**
- Chard stems and leaves are delicious
- Collard greens and other salty leafy greens

Conclusion:

- Ongchoi is a leafy green like spinach, chard, or collard greens
- We could conclude this based on words like "leaves" and "delicious" and "sautéed"

Ongchoi: *Ipomoea aquatica* "Water Spinach"

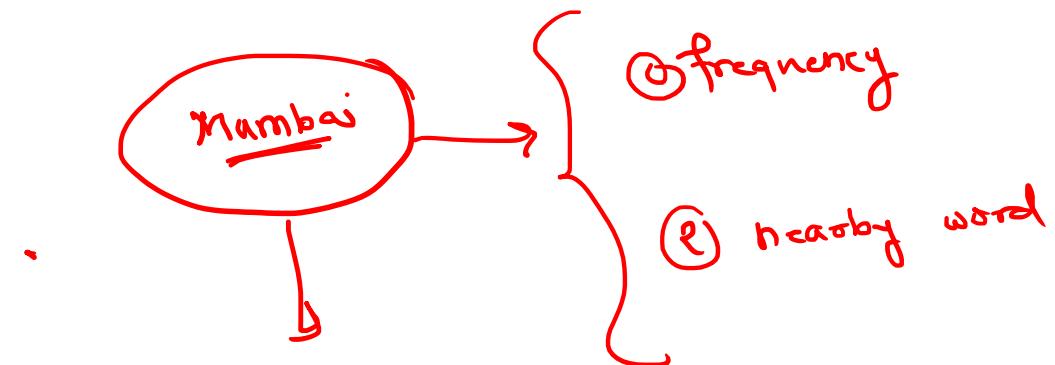
空心菜
kangkong
rau muống
...



Yamaguchi, Wikimedia Commons, public domain

Idea 1: Defining meaning by linguistic distribution

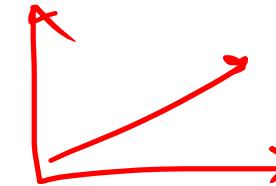
Let's define the meaning of a word by its distribution in language use, meaning its neighboring words or grammatical environments.



Idea 2: Meaning as a point in space (Osgood et al. 1957)

3 affective dimensions for a word

- valence: pleasantness
- arousal: intensity of emotion
- dominance: the degree of control exerted



	Word	Score		Word	Score
Valence ~~~~~	love	1.000		toxic	0.008
	happy	1.000		nightmare	0.005
Arousal ✓	elated	0.960		mellow	0.069
	frenzy	0.965		napping	0.046
Dominance ✓	powerful	0.991		weak	0.045
	leadership	0.983		empty	0.081

NRC VAD Lexicon
(Mohammad 2018)

Hence the connotation of a word is a vector in 3-space

Idea 1: Defining meaning by linguistic distribution

Idea 2: Meaning as a point in multidimensional space

Defining meaning as a point in space based on distribution

Each word = a vector (not just "good" or " w_{45} ")

Similar words are "nearby in semantic space"

We build this space automatically by seeing which words are **nearby in text**



We define meaning of a word as a vector

Called an "embedding" because it's embedded into a space (see textbook)

The standard way to represent meaning in NLP

Every modern NLP algorithm uses embeddings as the representation of word meaning

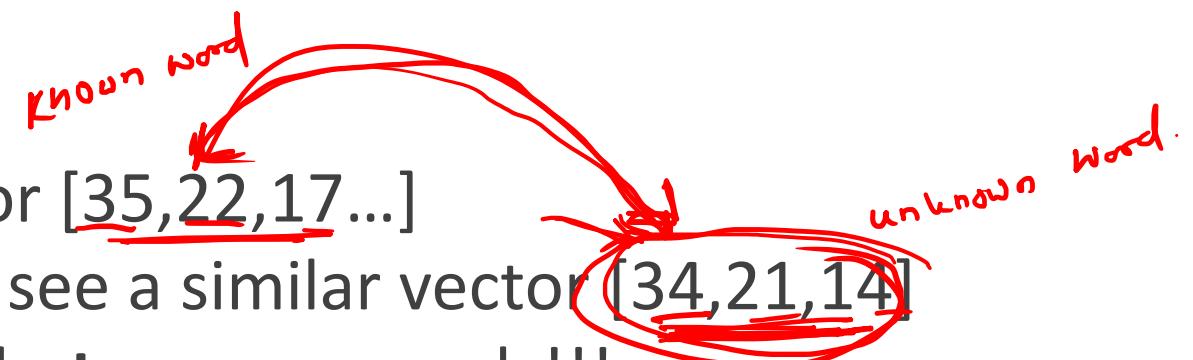
Fine-grained model of meaning for similarity



Intuition: why vectors?

Consider sentiment analysis:

- With words, a feature is a word identity
 - Feature 5: 'The previous word was "terrible"
 - requires **exact same word** to be in training and test
- With **embeddings**:
 - Feature is a word vector
 - 'The previous word was vector [35,22,17...]
 - Now in the test set we might see a similar vector [34,21,14]
 - We can generalize to **similar but unseen words!!!**



We'll discuss 2 kinds of embeddings

~~tf-idf~~

- Information Retrieval workhorse!
- A common baseline model
- Sparse vectors
- Words are represented by (a simple function of) the **counts** of nearby words

} frequency distribution approach

~~Word2vec~~

- Dense vectors
- Representation is created by training a classifier to **predict** whether a word is likely to appear nearby
- Later we'll discuss extensions called **contextual embeddings**

} NLP approach.

From now on:
Computing with meaning representations
instead of string representations

荃者所以在鱼，得鱼而忘荃 Nets are for fish;
Once you get the fish, you can forget the net.
言者所以在意，得意而忘言 Words are for meaning;
Once you get the meaning, you can forget the words
庄子(Zhuangzi), Chapter 26

Vector Semantics

Vector
Semantics &
Embeddings

Words and Vectors

Vector
Semantics &
Embeddings

Term-document matrix

Each document is represented by a vector of words

frequency

time given term appears into provided document

	Document 1 <i>As You Like It</i>	Document 2 <i>Twelfth Night</i>	D3 <i>Julius Caesar</i>	D4 <i>Henry V</i>
T ₁	1	0	7	13
T ₂	14	80	62	89
T ₃	36	58	1	4
T ₄	20	15	2	3

Term *Document 1* *Document 2* *D3* *D4*

T₁ battle 1 0 7 13

T₂ good 14 80 62 89

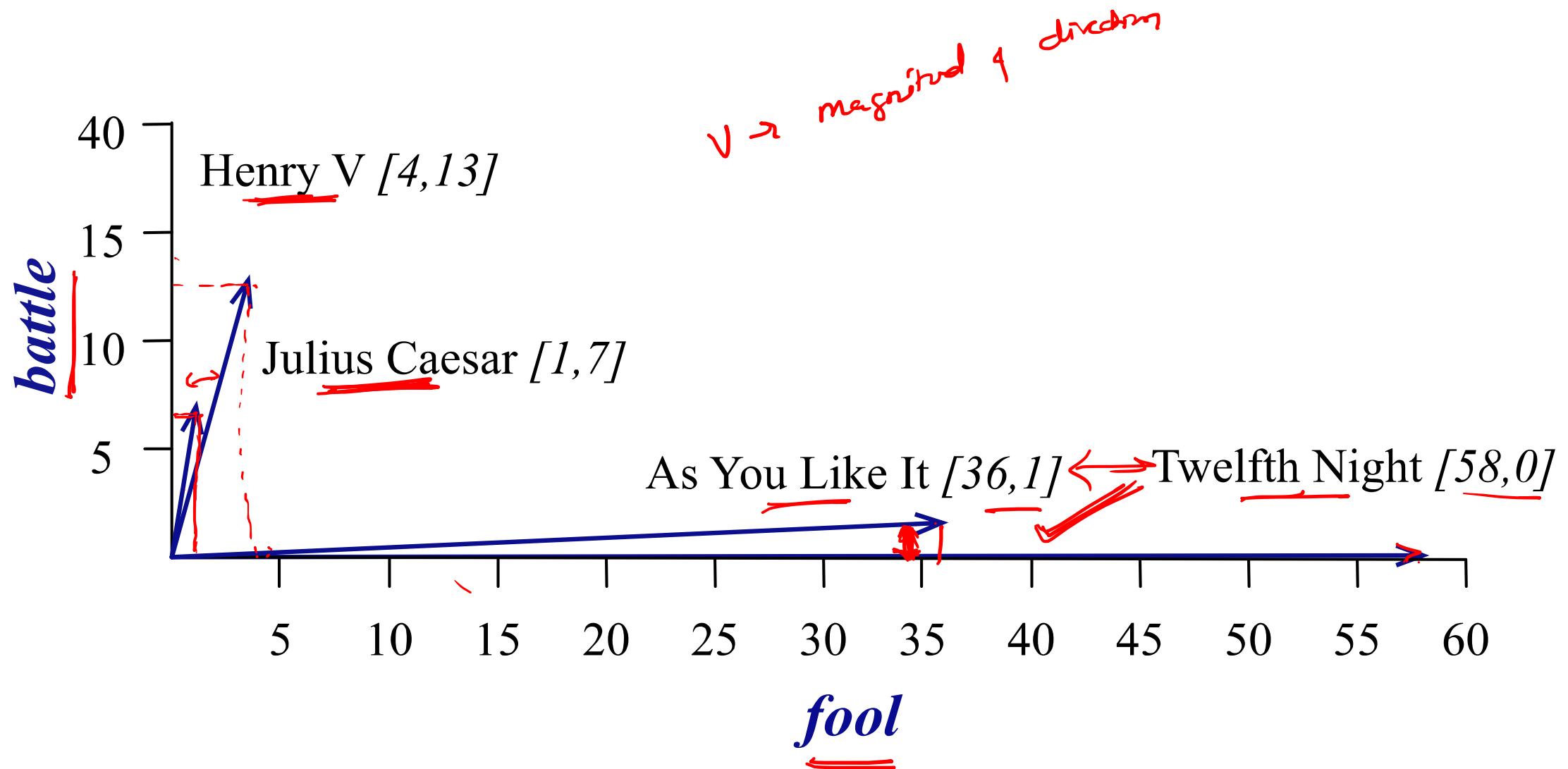
T₃ fool 36 58 1 4

T₄ wit 20 15 2 3

frequency

time given term appears into provided document

Visualizing document vectors



Vectors are the basis of information retrieval

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Vectors are similar for the two comedies

But comedies are different than the other two

Comedies have more *fools* and *wit* and fewer *battles*.

Idea for word meaning: Words can be vectors too!!!

	<u>As You Like It</u>	<u>Twelfth Night</u>	<u>Julius Caesar</u>	<u>Henry V</u>
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

battle is "the kind of word that occurs in Julius Caesar and Henry V"

fool is "the kind of word that occurs in comedies, especially Twelfth Night"

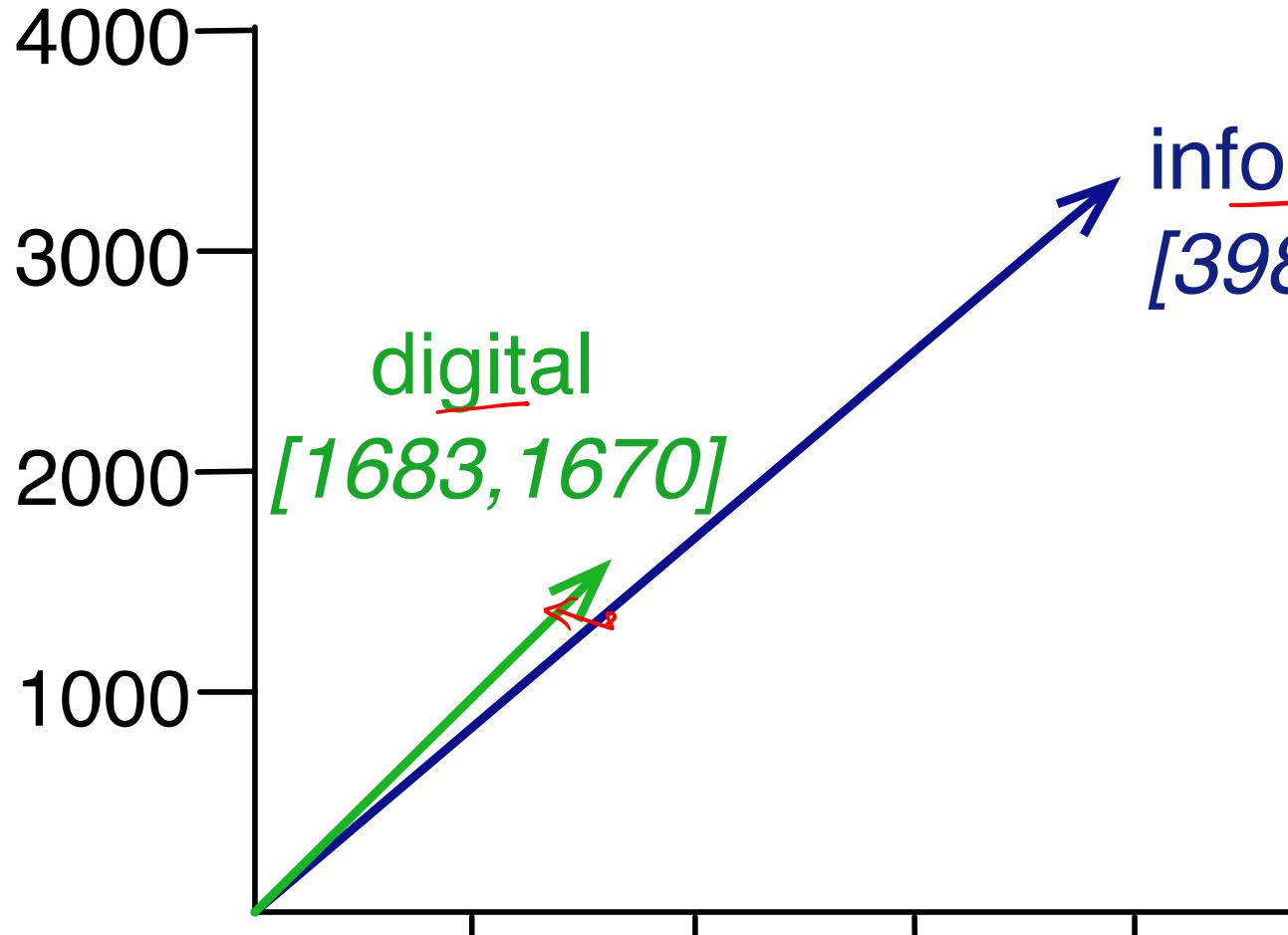
More common: word-word matrix (or "term-context matrix")

Two **words** are similar in meaning if their context vectors are similar

is traditionally followed by **cherry** pie, a traditional dessert
often mixed, such as **strawberry** rhubarb pie. Apple pie
computer peripherals and personal **digital** assistants. These devices usually
a computer. This includes **information** available on the internet

	aardvark	...	<u>computer</u>	<u>data</u>	<u>result</u>	<u>pie</u>	<u>sugar</u>	...
<u>cherry</u>	0	...	2	8	9	442	25	...
<u>strawberry</u>	0	...	0	0	1	60	19	...
<u>digital</u>	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

computer



data

Words and Vectors

Vector
Semantics &
Embeddings

Vector Semantics & Embeddings

Cosine for computing word similarity

- ① Linguistics
 - ② word meaning.
 - ③ limitations
 - ④ Vectors
 - How to build the vector representation for word
 - compute the similarity?
 - ⑤ How to
-

Computing word similarity: Dot product and cosine

The dot product between two vectors is a scalar:

$$\text{dot product}(\mathbf{v}, \mathbf{w}) = \mathbf{v} \cdot \mathbf{w} = \sum_{i=1}^N v_i w_i = v_1 w_1 + v_2 w_2 + \dots + v_N w_N$$

The dot product tends to be high when the two vectors have large values in the same dimensions

Dot product can thus be a useful similarity metric between vectors

Problem with raw dot-product

Dot product favors long vectors

Dot product is higher if a vector is longer (has higher values in many dimension)

Vector length:

$$|\mathbf{v}| = \sqrt{\sum_{i=1}^N v_i^2}$$

Frequent words (of, the, you) have long vectors (since they occur many times with other words).

So dot product overly favors frequent words

Alternative: cosine for computing word similarity

$$\text{cosine}(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}| |\vec{w}|} = \frac{\sum_{i=1}^N v_i w_i}{\sqrt{\sum_{i=1}^N v_i^2} \sqrt{\sum_{i=1}^N w_i^2}}$$

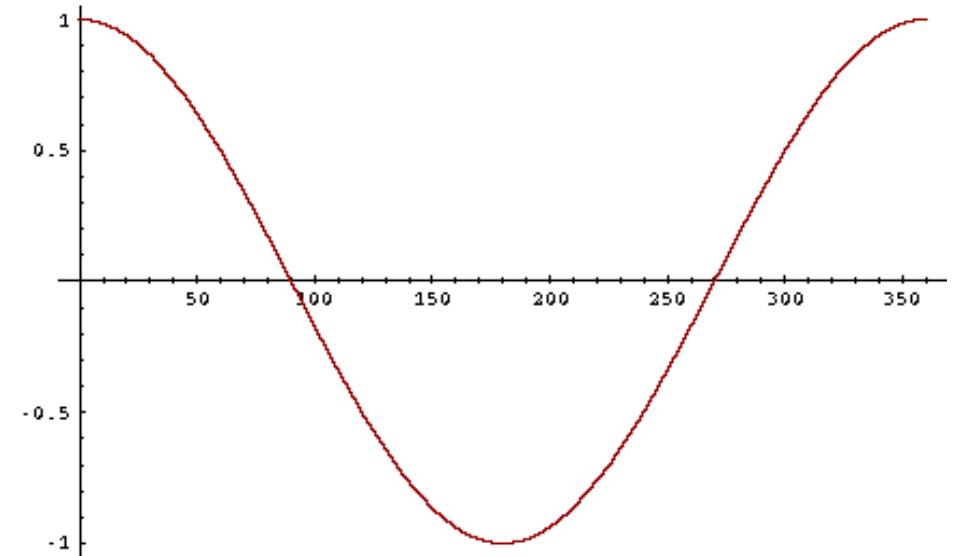
Based on the definition of the dot product between two vectors a and b

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| |\mathbf{b}| \cos \theta$$

$$\frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}| |\mathbf{b}|} = \cos \theta$$

Cosine as a similarity metric

- 1: vectors point in opposite directions
- +1: vectors point in same directions
- 0: vectors are orthogonal



But since raw frequency values are non-negative, the cosine for term-term matrix vectors ranges from 0–1

Cosine examples

$$\cos(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{\|\vec{v}\| \|\vec{w}\|} = \frac{\vec{v}}{\|\vec{v}\|} \cdot \frac{\vec{w}}{\|\vec{w}\|} = \frac{\sum_{i=1}^N v_i w_i}{\sqrt{\sum_{i=1}^N v_i^2} \sqrt{\sum_{i=1}^N w_i^2}}$$

$$\cos(\text{cherry}, \text{information}) =$$

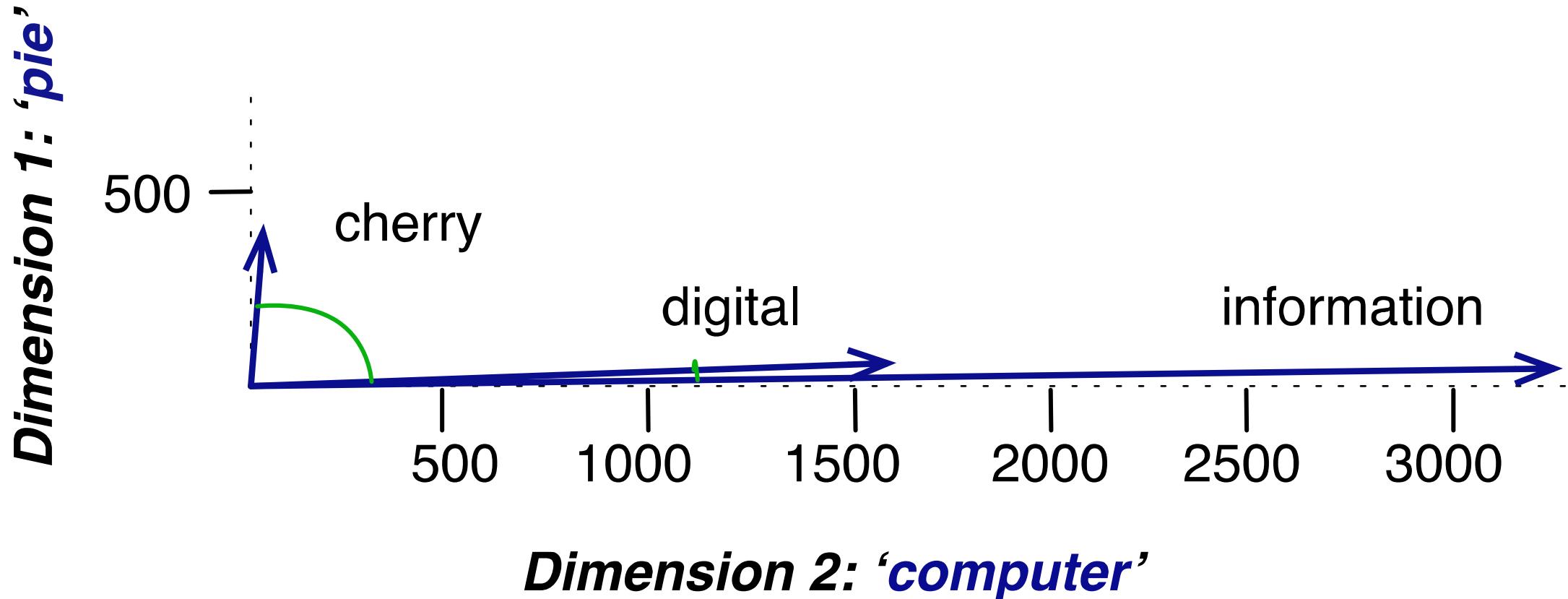
$$\frac{442 * 5 + 8 * 3982 + 2 * 3325}{\sqrt{442^2 + 8^2 + 2^2} \sqrt{5^2 + 3982^2 + 3325^2}} = .017$$

$$\cos(\text{digital}, \text{information}) =$$

$$\frac{5 * 5 + 1683 * 3982 + 1670 * 3325}{\sqrt{5^2 + 1683^2 + 1670^2} \sqrt{5^2 + 3982^2 + 3325^2}} = .996$$

	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325

Visualizing cosines (well, angles)



Vector Semantics & Embeddings

Cosine for computing word similarity

TF-IDF

Vector
Semantics &
Embeddings

But raw frequency is a bad representation

- The co-occurrence matrices we have seen represent each cell by word frequencies.
- Frequency is clearly useful; if *sugar* appears a lot near *apricot*, that's useful information.
- But overly frequent words like *the*, *it*, or *they* are not very informative about the context
- It's a paradox! How can we balance these two conflicting constraints?

Two common solutions for word weighting

tf-idf: tf-idf value for word t in document d:

$$w_{t,d} = \text{tf}_{t,d} \times \text{idf}_t$$

Words like "the" or "it" have very low idf

PMI: (Pointwise mutual information)

- $\text{PMI}(w_1, w_2) = \log \frac{p(w_1, w_2)}{p(w_1)p(w_2)}$

See if words like "good" appear more often with "great" than we would expect by chance

Term frequency (tf)

$$\text{tf}_{t,d} = \text{count}(t,d)$$

Instead of using raw count, we squash a bit:

$$\text{tf}_{t,d} = \log_{10}(\text{count}(t,d)+1)$$

Document frequency (df)

df_t is the number of documents t occurs in.

(note this is not collection frequency: total count across all documents)

"Romeo" is very distinctive for one Shakespeare play:

	Collection Frequency	Document Frequency
Romeo	113	1
action	113	31

Inverse document frequency (idf)

$$\text{idf}_t = \log_{10} \left(\frac{N}{\text{df}_t} \right)$$

N is the total number of documents
in the collection

Word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.246
wit	34	0.037
fool	36	0.012
good	37	0
sweet	37	0

What is a document?

Could be a play or a Wikipedia article

But for the purposes of tf-idf, documents can be
anything; we often call each paragraph a document!

Final tf-idf weighted value for a word

$$w_{t,d} = \text{tf}_{t,d} \times \text{idf}_t$$

Raw counts:

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

tf-idf:

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

TF-IDF

Vector
Semantics &
Embeddings

PPMI

Vector
Semantics &
Embeddings

Pointwise Mutual Information

Pointwise mutual information:

Do events x and y co-occur more than if they were independent?

$$\text{PMI}(X, Y) = \log_2 \frac{P(x, y)}{P(x)P(y)}$$

PMI between two words: (Church & Hanks 1989)

Do words x and y co-occur more than if they were independent?

$$\text{PMI}(\textit{word}_1, \textit{word}_2) = \log_2 \frac{P(\textit{word}_1, \textit{word}_2)}{P(\textit{word}_1)P(\textit{word}_2)}$$

Positive Pointwise Mutual Information

- PMI ranges from $-\infty$ to $+\infty$
- But the negative values are problematic
 - Things are co-occurring **less than** we expect by chance
 - Unreliable without enormous corpora
 - Imagine w_1 and w_2 whose probability is each 10^{-6}
 - Hard to be sure $p(w_1, w_2)$ is significantly different than 10^{-12}
 - Plus it's not clear people are good at "unrelatedness"
 - So we just replace negative PMI values by 0
 - Positive PMI (**PPMI**) between word1 and word2:

$$\text{PPMI}(word_1, word_2) = \max\left(\log_2 \frac{P(word_1, word_2)}{P(word_1)P(word_2)}, 0\right)$$

Computing PPMI on a term-context matrix

Matrix F with W rows (words) and C columns (contexts)

f_{ij} is # of times w_i occurs in context c_j

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^W \sum_{j=1}^C f_{ij}}$$
$$p_{i^*} = \frac{\sum_{j=1}^C f_{ij}}{\sum_{i=1}^W \sum_{j=1}^C f_{ij}}$$
$$p_{*j} = \frac{\sum_{i=1}^W f_{ij}}{\sum_{i=1}^W \sum_{j=1}^C f_{ij}}$$

	computer	data	result	pie	sugar	count(w)
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
count(context)	4997	5673	473	512	61	11716

$$pmi_{ij} = \log_2 \frac{p_{ij}}{p_{i^*} p_{*j}}$$

$$ppmi_{ij} = \begin{cases} pmi_{ij} & \text{if } pmi_{ij} > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$p_{ij} = \frac{\sum_{i=1}^W \sum_{j=1}^C f_{ij}}{\sum_{i=1}^W \sum_{j=1}^C f_{ij}}$$

	computer	data	result	pie	sugar	count(w)
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
count(context)	4997	5673	473	512	61	11716

$$p(w=information, c=data) = 3982/111716 = .3399$$

$$p(w=information) = 7703/111716 = .6575$$

$$p(c=data) = 5673/111716 = .4842$$

$$p(w_i) = \frac{\sum_{j=1}^C f_{ij}}{N} \quad p(c_j) = \frac{\sum_{i=1}^W f_{ij}}{N}$$

	p(w,context)					p(w)
	computer	data	result	pie	sugar	p(w)
cherry	0.0002	0.0007	0.0008	0.0377	0.0021	0.0415
strawberry	0.0000	0.0000	0.0001	0.0051	0.0016	0.0068
digital	0.1425	0.1436	0.0073	0.0004	0.0003	0.2942
information	0.2838	0.3399	0.0323	0.0004	0.0011	0.6575
p(context)	0.4265	0.4842	0.0404	0.0437	0.0052	

$$pmi_{ij} = \log_2 \frac{p_{ij}}{p_i^* p_{*j}}$$

	p(w,context)					p(w)
	computer	data	result	pie	sugar	p(w)
cherry	0.0002	0.0007	0.0008	0.0377	0.0021	0.0415
strawberry	0.0000	0.0000	0.0001	0.0051	0.0016	0.0068
digital	0.1425	0.1436	0.0073	0.0004	0.0003	0.2942
information	0.2838	0.3399	0.0323	0.0004	0.0011	0.6575
p(context)	0.4265	0.4842	0.0404	0.0437	0.0052	

$$pmi(\text{information}, \text{data}) = \log_2 (.3399 / (.6575 * .4842)) = .0944$$

Resulting PPMI matrix (negatives replaced by 0)

	computer	data	result	pie	sugar
cherry	0	0	0	4.38	3.30
strawberry	0	0	0	4.10	5.51
digital	0.18	0.01	0	0	0
information	0.02	0.09	0.28	0	0

Weighting PMI

PMI is biased toward infrequent events

- Very rare words have very high PMI values

Two solutions:

- Give rare words slightly higher probabilities
- Use add-one smoothing (which has a similar effect)

Weighting PMI: Giving rare context words slightly higher probability

Raise the context probabilities to $\alpha = 0.75$:

$$\text{PPMI}_\alpha(w, c) = \max(\log_2 \frac{P(w, c)}{P(w)P_\alpha(c)}, 0)$$

$$P_\alpha(c) = \frac{\text{count}(c)^\alpha}{\sum_c \text{count}(c)^\alpha}$$

This helps because $P_\alpha(c) > P(c)$ for rare c

Consider two events, $P(a) = .99$ and $P(b) = .01$

$$P_\alpha(a) = \frac{.99^{.75}}{.99^{.75} + .01^{.75}} = .97 \quad P_\alpha(b) = \frac{.01^{.75}}{.01^{.75} + .01^{.75}} = .03$$

Word2vec

Vector
Semantics &
Embeddings

Sparse versus dense vectors

tf-idf (or PMI) vectors are

- **long** (length $|V| = 20,000$ to $50,000$)
- **sparse** (most elements are zero)

Alternative: learn vectors which are

- **short** (length $50-1000$)
- **dense** (most elements are non-zero)

Sparse versus dense vectors

Why dense vectors?

- Short vectors may be easier to use as **features** in machine learning (fewer weights to tune)
- Dense vectors may **generalize** better than explicit counts
- Dense vectors may do better at capturing synonymy:
 - *car* and *automobile* are synonyms; but are distinct dimensions
 - a word with *car* as a neighbor and a word with *automobile* as a neighbor should be similar, but aren't
- **In practice, they work better**

Common methods for getting short dense vectors

“Neural Language Model”-inspired models

- Word2vec (skipgram, CBOW), GloVe

Singular Value Decomposition (SVD)

- A special case of this is called LSA – Latent Semantic Analysis

Alternative to these "static embeddings":

- Contextual Embeddings (ELMo, BERT)
- Compute distinct embeddings for a word in its context
- Separate embeddings for each token of a word

Simple static embeddings you can download!

Word2vec (Mikolov et al)

<https://code.google.com/archive/p/word2vec/>

GloVe (Pennington, Socher, Manning)

<http://nlp.stanford.edu/projects/glove/>

Word2vec

Popular embedding method

Very fast to train

Code available on the web

Idea: **predict** rather than **count**

Word2vec provides various options. We'll do:

skip-gram with negative sampling (SGNS)

Word2vec

Instead of **counting** how often each word w occurs near "*apricot*"

- Train a classifier on a binary **prediction** task:
 - Is w likely to show up near "*apricot*"?

We don't actually care about this task

- But we'll take the learned classifier weights as the word embeddings

Big idea: **self-supervision**:

- A word c that occurs near *apricot* in the corpus cats as the gold "correct answer" for supervised learning
- No need for human labels
- Bengio et al. (2003); Collobert et al. (2011)

Approach: predict if candidate word c is a "neighbor"

1. Treat the target word t and a neighboring context word c as **positive examples**.
2. Randomly sample other words in the lexicon to get negative examples
3. Use logistic regression to train a classifier to distinguish those two cases
4. Use the learned weights as the embeddings

Skip-Gram Training Data

Assume a +/- 2 word window, given training sentence:

...lemon, a [tablespoon of apricot jam, a] pinch...

c1 c2 [target] c3 c4

Skip-Gram Classifier

(assuming a +/- 2 word window)

...lemon, a [tablespoon of apricot jam, a] pinch...
c1 c2 [target] c3 c4

Goal: train a classifier that is given a candidate (word, context) pair
(apricot, jam)
(apricot, aardvark)

...

And assigns each pair a probability:

$$P(+ | w, c)$$

$$P(- | w, c) = 1 - P(+ | w, c)$$

Similarity is computed from dot product

Remember: two vectors are similar if they have a high dot product

- Cosine is just a normalized dot product

So:

- $\text{Similarity}(w, c) \propto w \cdot c$

We'll need to normalize to get a probability

- (cosine isn't a probability either)

Turning dot products into probabilities

$$\text{Sim}(w, c) \approx w \cdot c$$

To turn this into a probability

We'll use the sigmoid from logistic regression:

$$P(+|w, c) = \sigma(c \cdot w) = \frac{1}{1 + \exp(-c \cdot w)}$$

$$\begin{aligned} P(-|w, c) &= 1 - P(+|w, c) \\ &= \sigma(-c \cdot w) = \frac{1}{1 + \exp(c \cdot w)} \end{aligned}$$

How Skip-Gram Classifier computes $P(+|w, c)$

$$P(+|w, c) = \sigma(c \cdot w) = \frac{1}{1 + \exp(-c \cdot w)}$$

This is for one context word, but we have lots of context words.
We'll assume independence and just multiply them:

$$P(+|w, c_{1:L}) = \prod_{i=1}^L \sigma(c_i \cdot w)$$

$$\log P(+|w, c_{1:L}) = \sum_{i=1}^L \log \sigma(c_i \cdot w)$$

Skip-gram classifier: summary

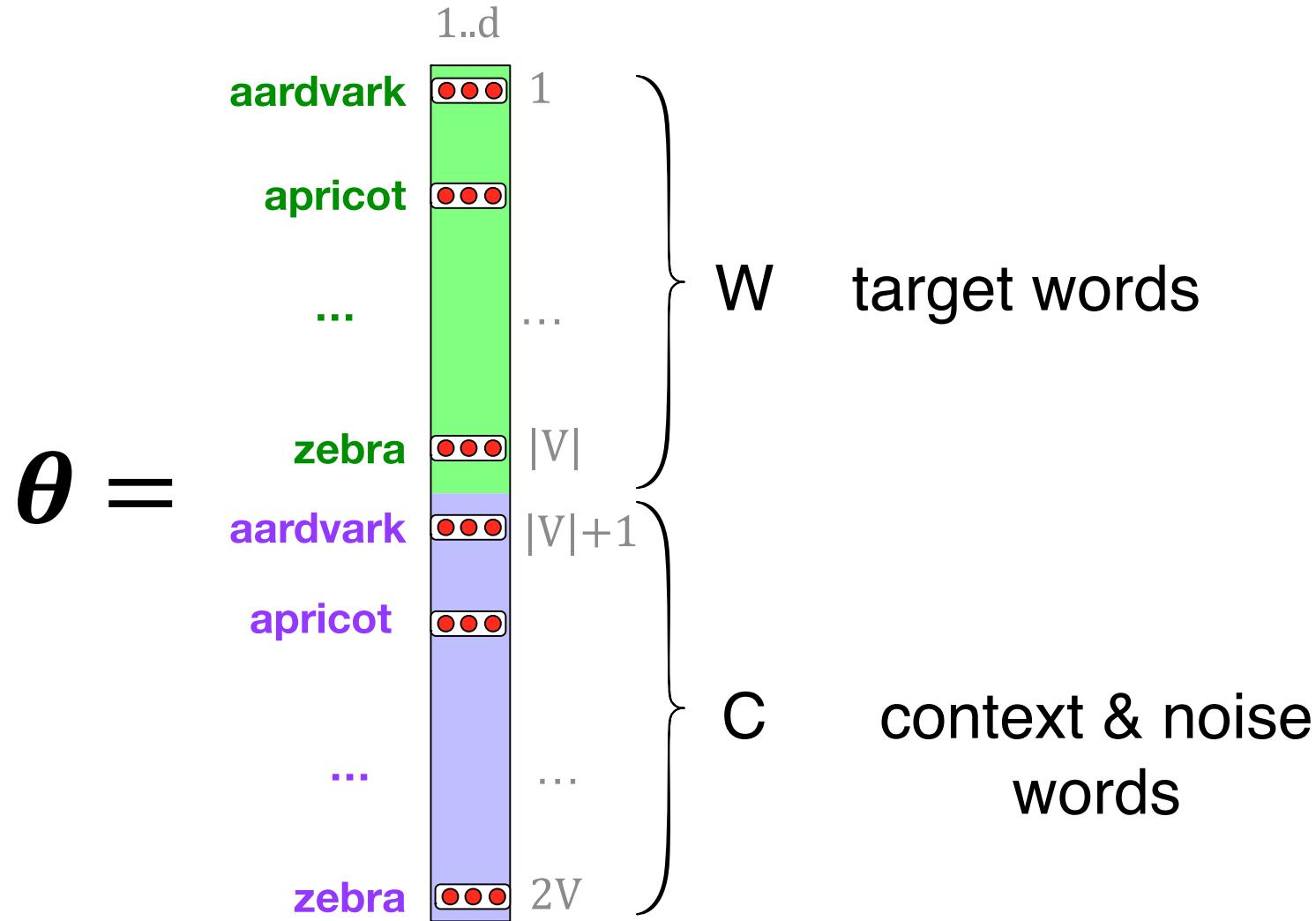
A probabilistic classifier, given

- a test target word w
- its context window of L words $c_{1:L}$

Estimates probability that w occurs in this window based on similarity of w (embeddings) to $c_{1:L}$ (embeddings).

To compute this, we just need embeddings for all the words.

These embeddings we'll need: a set for w, a set for c



Word2vec

Vector
Semantics &
Embeddings

Vector Semantics & Embeddings

Word2vec: Learning the embeddings

Skip-Gram Training data

...lemon, a [tablespoon of apricot jam, a] pinch...

c1

c2 [target]

c3

c4



positive examples +

t

c

apricot tablespoon

apricot of

apricot jam

apricot a

Skip-Gram Training data

...lemon, a [tablespoon of apricot jam, a] pinch...

c1 c2 [target] c3 c4



positive examples +

t c

apricot tablespoon

apricot of

apricot jam

apricot a

For each positive example we'll grab k negative examples, sampling by frequency

Skip-Gram Training data

...lemon, a [tablespoon of apricot jam, a] pinch...

c1 c2 [target] c3 c4



positive examples +

t	c
---	---

apricot	tablespoon
---------	------------

apricot	of
---------	----

apricot	jam
---------	-----

apricot	a
---------	---

negative examples -

t	c	t	c
---	---	---	---

apricot	aardvark	apricot	seven
---------	----------	---------	-------

apricot	my	apricot	forever
---------	----	---------	---------

apricot	where	apricot	dear
---------	-------	---------	------

apricot	coaxial	apricot	if
---------	---------	---------	----

Word2vec: how to learn vectors

Given the set of positive and negative training instances, and an initial set of embedding vectors

The goal of learning is to adjust those word vectors such that we:

- **Maximize** the similarity of the **target word, context word** pairs (w, c_{pos}) drawn from the positive data
- **Minimize** the similarity of the (w, c_{neg}) pairs drawn from the negative data.

Loss function for one w with c_{pos} , $c_{neg1} \dots c_{negk}$

Maximize the similarity of the target with the actual context words, and minimize the similarity of the target with the k negative sampled non-neighbor words.

$$\begin{aligned} L_{CE} &= -\log \left[P(+|w, c_{pos}) \prod_{i=1}^k P(-|w, c_{neg_i}) \right] \\ &= - \left[\log P(+|w, c_{pos}) + \sum_{i=1}^k \log P(-|w, c_{neg_i}) \right] \\ &= - \left[\log P(+|w, c_{pos}) + \sum_{i=1}^k \log (1 - P(+|w, c_{neg_i})) \right] \\ &= - \left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^k \log \sigma(-c_{neg_i} \cdot w) \right] \end{aligned}$$

Learning the classifier

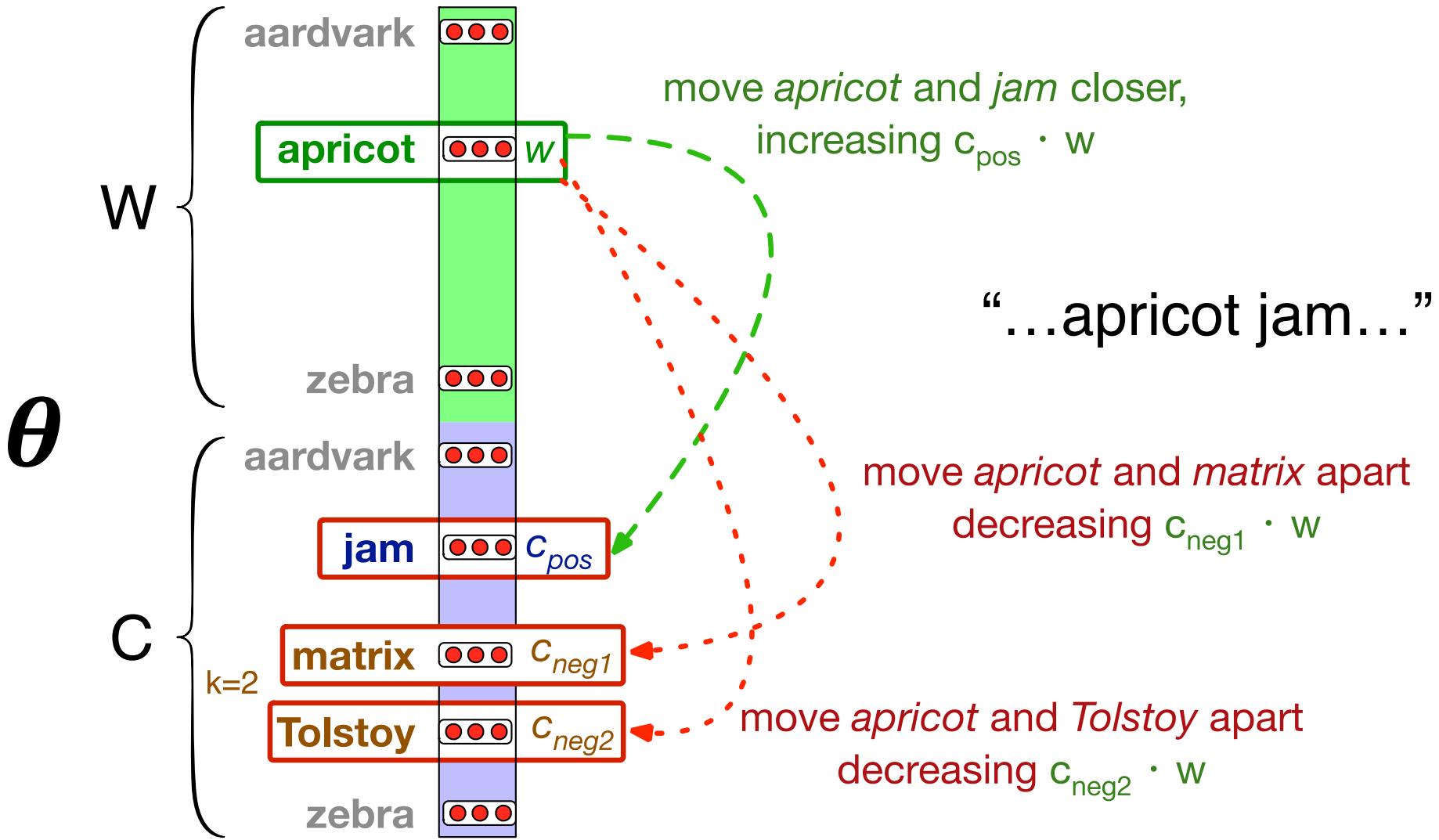
How to learn?

- Stochastic gradient descent!

We'll adjust the word weights to

- make the positive pairs more likely
- and the negative pairs less likely,
- over the entire training set.

Intuition of one step of gradient descent



Reminder: gradient descent

- At each step
 - Direction: We move in the reverse direction from the gradient of the loss function
 - Magnitude: we move the value of this gradient $\frac{d}{dw} L(f(x; w), y)$ weighted by a **learning rate** η
 - Higher learning rate means move w faster

$$w^{t+1} = w^t - h \frac{d}{dw} L(f(x; w), y)$$

The derivatives of the loss function

$$L_{CE} = - \left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^k \log \sigma(-c_{neg_i} \cdot w) \right]$$

$$\frac{\partial L_{CE}}{\partial c_{pos}} = [\sigma(c_{pos} \cdot w) - 1]w$$

$$\frac{\partial L_{CE}}{\partial c_{neg}} = [\sigma(c_{neg} \cdot w)]w$$

$$\frac{\partial L_{CE}}{\partial w} = [\sigma(c_{pos} \cdot w) - 1]c_{pos} + \sum_{i=1}^k [\sigma(c_{neg_i} \cdot w)]c_{neg_i}$$

Update equation in SGD

Start with randomly initialized C and W matrices, then incrementally do updates

$$c_{pos}^{t+1} = c_{pos}^t - \eta [\sigma(c_{pos}^t \cdot w^t) - 1] w^t$$

$$c_{neg}^{t+1} = c_{neg}^t - \eta [\sigma(c_{neg}^t \cdot w^t)] w^t$$

$$w^{t+1} = w^t - \eta \left[[\sigma(c_{pos} \cdot w^t) - 1] c_{pos} + \sum_{i=1}^k [\sigma(c_{neg_i} \cdot w^t)] c_{neg_i} \right]$$

Two sets of embeddings

SGNS learns two sets of embeddings

Target embeddings matrix W

Context embedding matrix C

It's common to just add them together,
representing word i as the vector $w_i + c_i$

Summary: How to learn word2vec (skip-gram) embeddings

Start with V random d -dimensional vectors as initial embeddings

Train a classifier based on embedding similarity

- Take a corpus and take pairs of words that co-occur as positive examples
- Take pairs of words that don't co-occur as negative examples
- Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance
- Throw away the classifier code and keep the embeddings.

Vector Semantics & Embeddings

Word2vec: Learning the embeddings

Vector Semantics & Embeddings

Properties of Embeddings

The kinds of neighbors depend on window size

Small windows (C= +/- 2) : nearest words are syntactically similar words in same taxonomy

- *Hogwarts* nearest neighbors are other fictional schools
 - *Sunnydale, Evernight, Blandings*

Large windows (C= +/- 5) : nearest words are related words in same semantic field

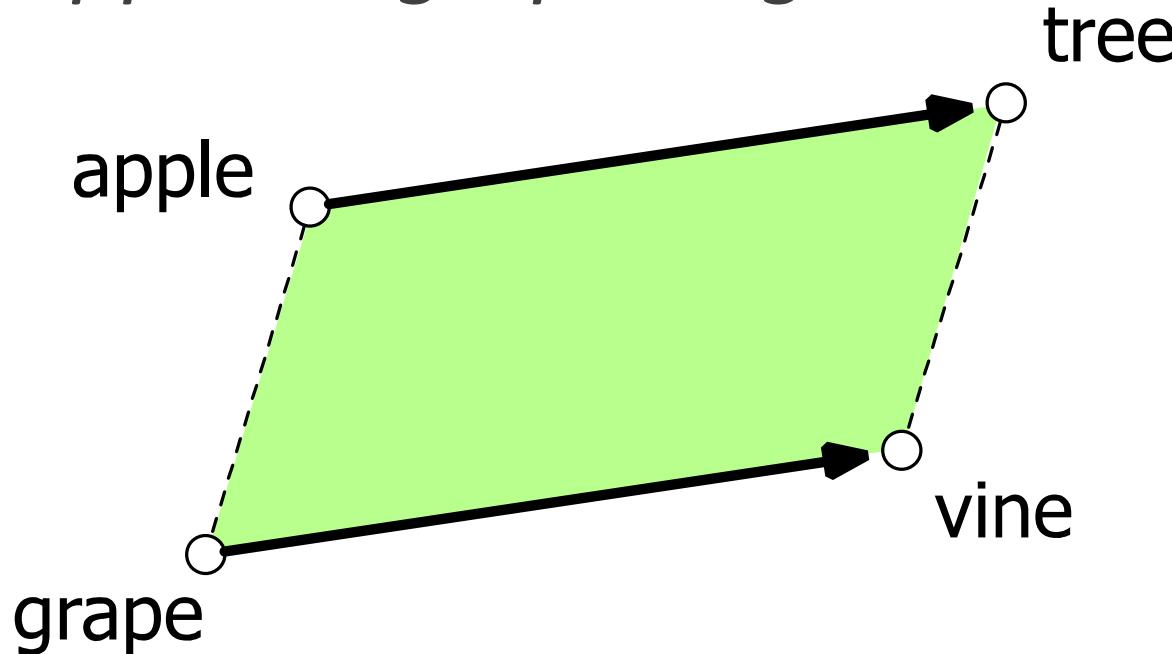
- *Hogwarts* nearest neighbors are Harry Potter world:
 - *Dumbledore, half-blood, Malfoy*

Analogical relations

The classic parallelogram model of analogical reasoning
(Rumelhart and Abrahamson 1973)

To solve: "*apple* is to *tree* as *grape* is to _____"

Add $\overrightarrow{\text{tree}} - \overrightarrow{\text{apple}}$ to $\overrightarrow{\text{grape}}$ to get $\overrightarrow{\text{vine}}$



Analogical relations via parallelogram

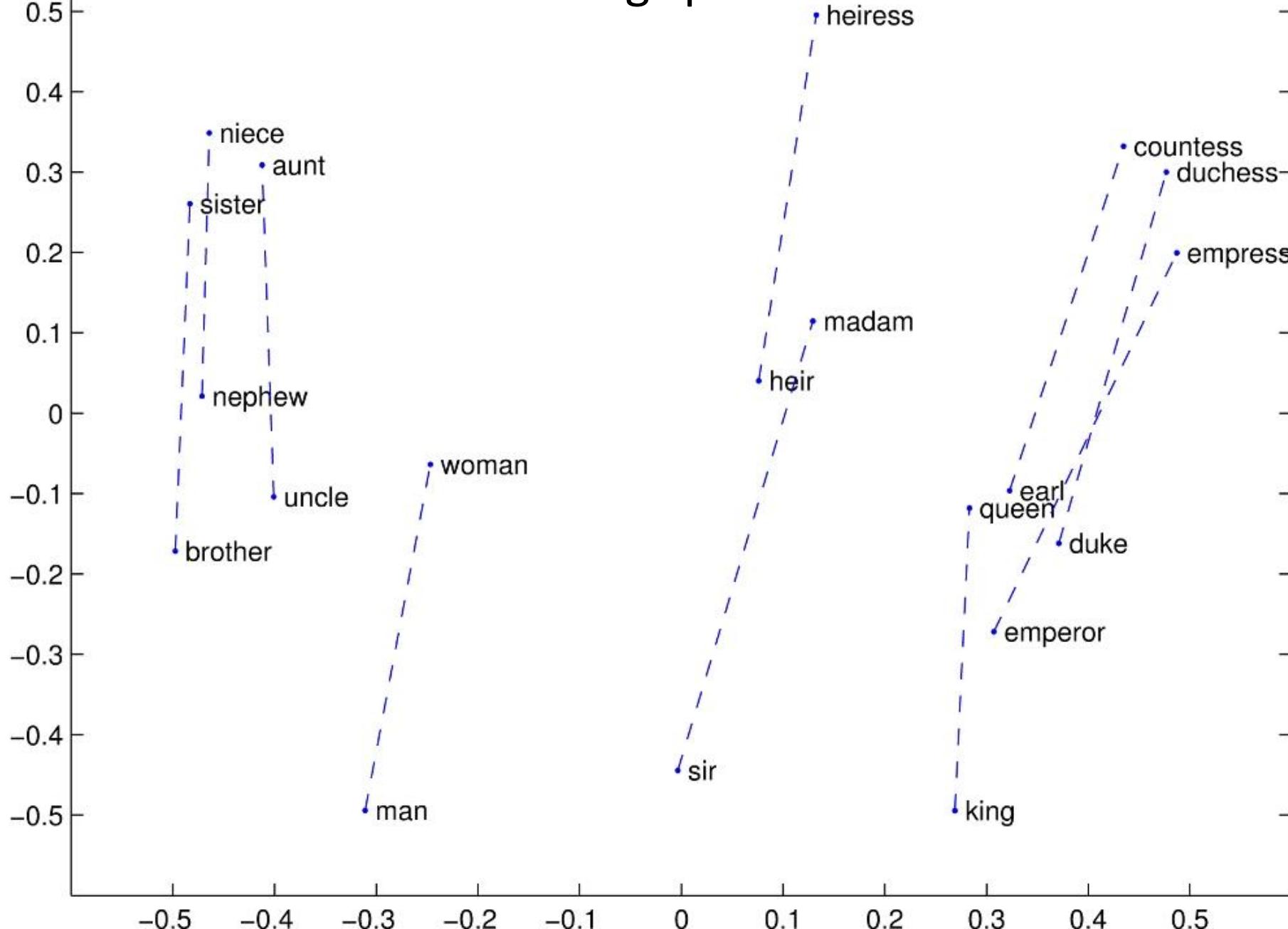
The parallelogram method can solve analogies with both sparse and dense embeddings (Turney and Littman 2005, Mikolov et al. 2013b)

$$\begin{array}{c} \xrightarrow{\hspace{1cm}} \xrightarrow{\hspace{1cm}} \xrightarrow{\hspace{1cm}} \\ \text{king} - \text{man} + \text{woman} \text{ is close to } \text{queen} \end{array}$$
$$\begin{array}{c} \xrightarrow{\hspace{1cm}} \xrightarrow{\hspace{1cm}} \xrightarrow{\hspace{1cm}} \\ \text{Paris} - \text{France} + \text{Italy} \text{ is close to } \text{Rome} \end{array}$$

For a problem $a:a^*::b:b^*$, the parallelogram method is:

$$\hat{b}^* = \operatorname{argmax}_x \operatorname{distance}(x, a^* - a + b)$$

Structure in GloVe Embedding space



Caveats with the parallelogram method

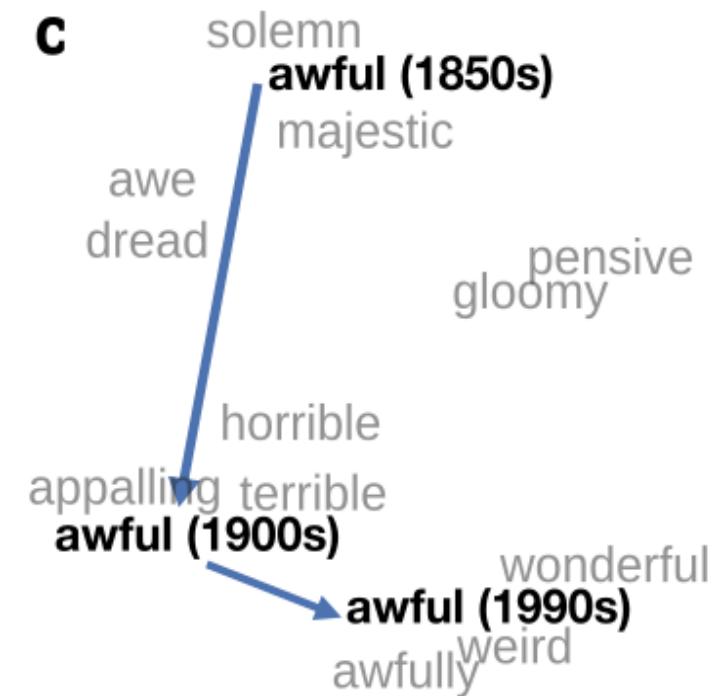
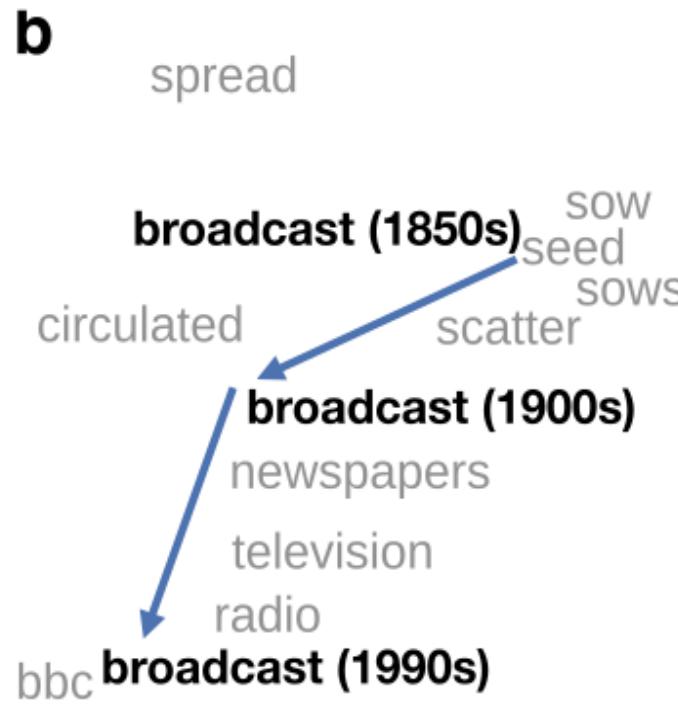
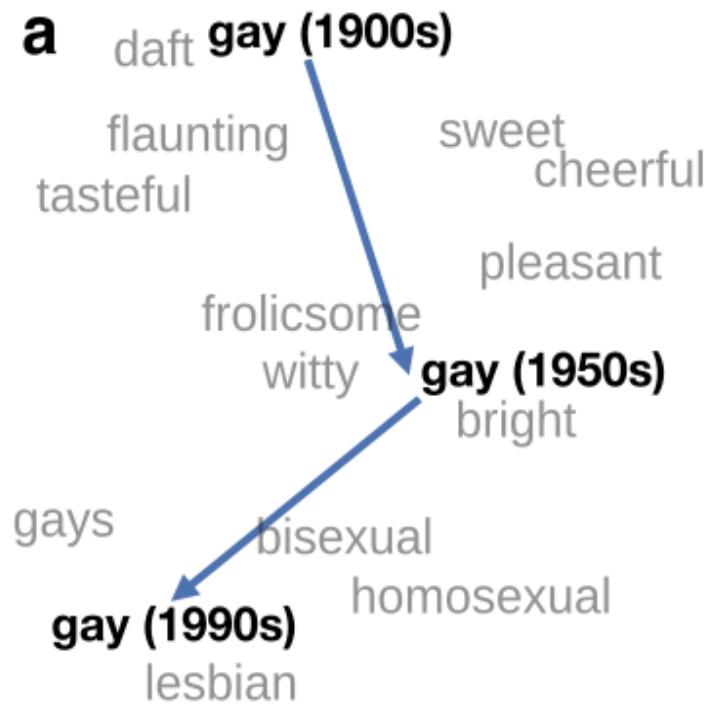
It only seems to work for frequent words, small distances and certain relations (relating countries to capitals, or parts of speech), but not others. (Linzen 2016, Gladkova et al. 2016, Ethayarajh et al. 2019a)

Understanding analogy is an open area of research
(Peterson et al. 2020)

Embeddings as a window onto historical semantics

Train embeddings on different decades of historical text to see meanings shift

~30 million books, 1850-1990, Google Books data



Embeddings reflect cultural bias!

Bolukbasi, Tolga, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." In *NeurIPS*, pp. 4349-4357. 2016.

Ask “Paris : France :: Tokyo : x”

- x = Japan

Ask “father : doctor :: mother : x”

- x = nurse

Ask “man : computer programmer :: woman : x”

- x = homemaker

Algorithms that use embeddings as part of e.g., hiring searches for programmers, might lead to bias in hiring

Historical embedding as a tool to study cultural biases

Garg, N., Schiebinger, L., Jurafsky, D., and Zou, J. (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. Proceedings of the National Academy of Sciences 115(16), E3635–E3644.

- Compute a **gender or ethnic bias** for each adjective: e.g., how much closer the adjective is to "woman" synonyms than "man" synonyms, or names of particular ethnicities
 - Embeddings for **competence** adjective (*smart, wise, brilliant, resourceful, thoughtful, logical*) are biased toward men, a bias slowly decreasing 1960-1990
 - Embeddings for **dehumanizing** adjectives (barbaric, monstrous, bizarre) were biased toward Asians in the 1930s, bias decreasing over the 20th century.
- These match the results of old surveys done in the 1930s

Vector Semantics & Embeddings

Properties of Embeddings

Word Sense Disambiguation

Word Sense Disambiguation (WSD)

- Given
 - A word in context
 - A fixed inventory of potential word senses
 - Decide which sense of the word this is
- Why? Machine translation, QA, speech synthesis
- What set of senses?
 - English--to--Spanish MT: set of Spanish translations
 - Speech Synthesis: homographs like *bass* and *bow*
 - In general: the senses in a thesaurus like WordNet

Two variants of WSD task

- Lexical Sample task
 - Small pre-selected set of target words (*line, plant*)
 - And inventory of senses for each word
 - **Supervised machine learning: train a classifier for each word**
- All-words task
 - Every word in an entire text
 - A lexicon with senses for each word
 - Data sparseness: can't train word--specific classifiers

WSD Methods

- Supervised Machine Learning
- Thesaurus/Dictionary Methods
- Semi--Supervised Learning

Word Sense Disambiguation

Supervised
Machine Learning

Supervised Machine Learning Approaches

- Supervised machine learning approach:
 - a **training corpus** of words tagged in context with their sense
 - used to train a classifier that can tag words in new text
- Summary of what we need:
 - the **tag set** (“sense inventory”)
 - the **training corpus**
 - A set of **features** extracted from the training corpus
 - A **classifier**

Supervised WSD 1: WSD Tags

- What's a tag?
A dictionary sense?
- For example, for WordNet an instance of “bass” in a text has 8 possible tags or labels (bass1 through bass8).

8 senses of “bass” in WordNet

1. bass -(the lowest part of the musical range)
2. bass, bass part -(the lowest part in polyphonic music)
3. bass, basso -(an adult male singer with the lowest voice)
4. sea bass, bass -(flesh of lean-fleshed saltwater fish of the family Serranidae)
5. freshwater bass, bass -(any of various North American lean-fleshed freshwater fishes especially of the genus Micropterus)
6. bass, bass voice, basso -(the lowest adult male singing voice)
7. bass -(the member with the lowest range of a family of musical instruments)
8. bass -(nontechnical name for any of numerous edible marine and freshwater spiny-finned fishes)

Inventory of sense tags for *bass*

WordNet Sense	Spanish Translation	Roget Category	Target Word in Context
bass ⁴	lubina	FISH/INSECT	... fish as Pacific salmon and striped bass and...
bass ⁴	lubina	FISH/INSECT	... produce filets of smoked bass or sturgeon...
bass ⁷	bajo	MUSIC	... exciting jazz bass player since Ray Brown...
bass ⁷	bajo	MUSIC	... play bass because he doesn't have to solo...

Supervised WSD 2: Get a corpus

- Lexical sample task:
 - *Line-hard-serve* corpus -4000 examples of each
 - *Interest* corpus -2369 sense-tagged examples
- All words:
 - **Semantic concordance:** a corpus in which each open-class word is labeled with a sense from a specific dictionary/thesaurus.
 - SemCor: 234,000 words from Brown Corpus, manually tagged with WordNet senses
 - SENSEVAL-3 competition corpora -2081 tagged word tokens

SemCor

<wf pos=PRP>He</wf>
<wf pos=VB lemma=recognize wnsn=4 lexsn=2:31:00::>**recognized**</wf>
<wf pos=DT>the</wf>
<wf pos>NN lemma=gesture wnsn=1 lexsn=1:04:00::>**gesture**</wf>
<punc>.</punc>

Supervised WSD 3: Extract feature vectors

Intuition from Warren Weaver (1955):

“If one examines the words in a book, one at a time as through an opaque mask with a hole in it one word wide, then it is obviously impossible to determine, one at a time, the meaning of the words...

But if one lengthens the slit in the opaque mask, until one can see not only the central word in question but also say N words on either side, then if N is large enough one can unambiguously decide the meaning of the central word...

The practical question is : “What minimum value of N will, at least in a tolerable fraction of cases, lead to the correct choice of meaning for the central word?”

Feature vectors

- A simple representation for each observation
(each instance of a target word)
 - **Vectors** of sets of feature/value pairs
 - Represented as a ordered list of values
 - These vectors represent, e.g., the window of words around the target

Two kinds of features in the vectors

- Collocational features and **bag-of-words** features
 - **Collocational**
 - Features about words at **specific** positions near target word
 - Often limited to just word identity and POS
 - **Bag-of-words**
 - Features about words that occur anywhere in the window (regardless of position)
 - **Typically limited to frequency counts**

Examples

- Example text (WSJ):
An electric guitar and **bass** player stand off to one side not really part of the scene
- Assume a window of +/-2 from the target

Examples

- Example text (WSJ)

An electric **guitar** **and** **bass** **player** **stand** off to one side not really part of the scene,

- Assume a window of +/-2 from the target

Collocational features

- Position-specific information about the words and collocations in window

-  guitar and bass player stand

$[w_{i-2}, \text{POS}_{i-2}, w_{i-1}, \text{POS}_{i-1}, w_{i+1}, \text{POS}_{i+1}, w_{i+2}, \text{POS}_{i+2}, w_{i-2}^{i-1}, w_i^{i+1}]$

[guitar, NN, and, CC, player, NN, stand, VB, and guitar, player stand]

- word 1,2,3 grams in window of ± 3 is common

Bag-of-words features

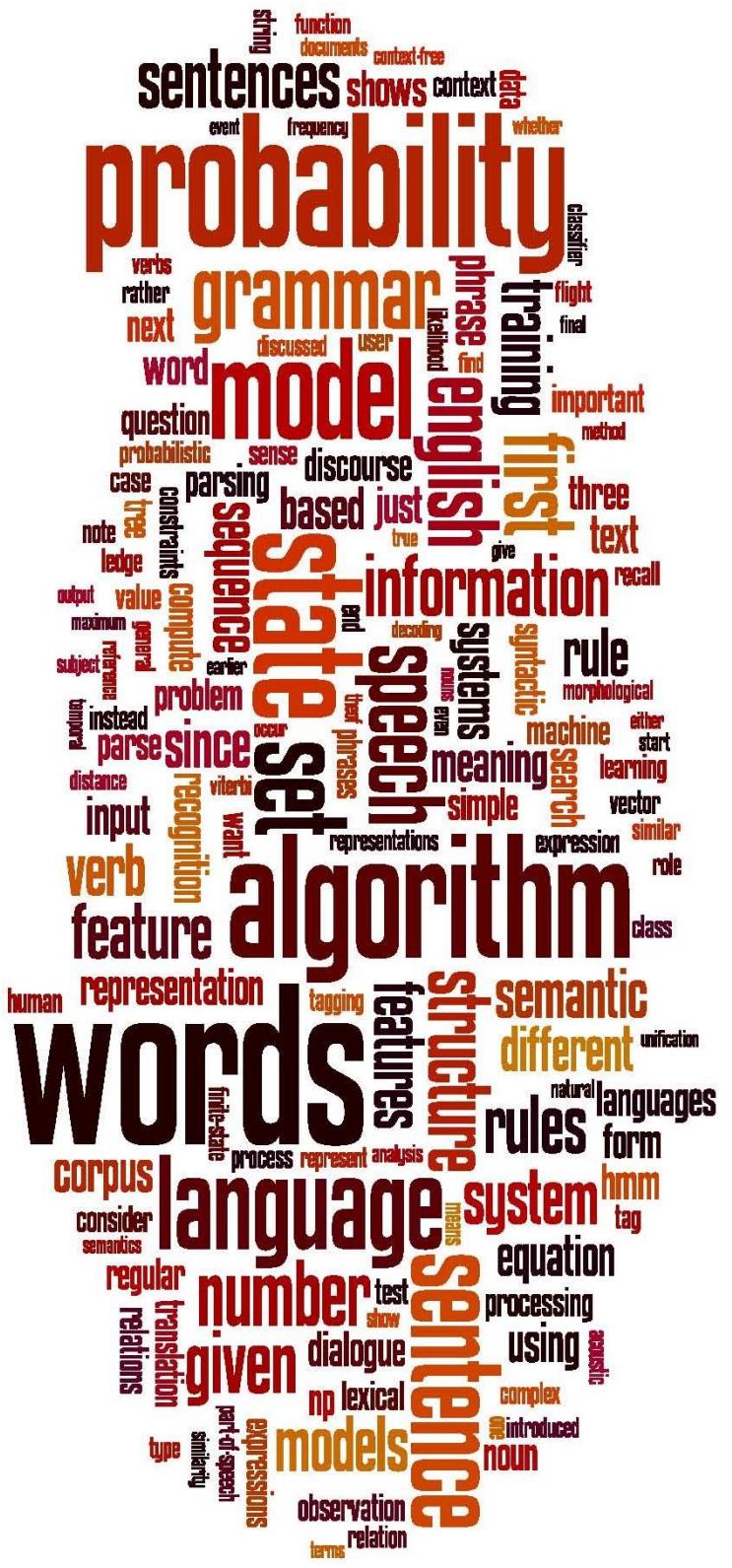
- “an unordered set of words” – position ignored
- Counts of words occur within the window.
- First choose a vocabulary
- Then count how often each of those terms occurs in a given window
 - sometimes just a binary “indicator” 1 or 0

Co--Occurrence Example

- Assume we've settled on a possible vocabulary of 12 words in “bass” sentences:

[fishing, big, sound, player, fly, rod, pound, double, runs, playing, guitar, band]

- The vector for:
guitar and bass player stand
[0,0,0,1,0,0,0,0,0,1,0]



Word Sense Disambiguation

Classification

Classification: definition

- *Input:*
 - a word w and some features f
 - a fixed set of classes $C = \{c_1, c_2, \dots, c_J\}$
- *Output:* a predicted class $c \in C$

Classification Methods: Supervised Machine Learning

- *Input:*
 - a word w in a text window d (which we'll call a "document")
 - a fixed set of classes $C = \{c_1, c_2, \dots, c_J\}$
 - A training set of m hand-labeled text windows again called "documents" $(d_1, c_1), \dots, (d_m, c_m)$
- *Output:*
 - a learned classifier $\gamma: d \rightarrow c$

Classification Methods: Supervised Machine Learning

- Any kind of classifier
 - Naive Bayes
 - Logistic regression
 - Neural Networks
 - Support--vector machines
 - k--Nearest Neighbors
 - ...

Applying Naive Bayes to WSD

- $P(c)$ is the prior probability of that sense
 - Counting in a labeled training set.
- $P(w|c)$ conditional probability of a word given a particular sense
 - $P(w|c) = \text{count}(w,c)/\text{count}(c)$
- We get both of these from a tagged corpus like SemCor
- Can also generalize to look at other features besides words.
 - Then it would be $P(f|c)$
 - Conditional probability of a feature given a sense

$$\hat{P}(c) = \frac{N_c}{N}$$

$$\hat{P}(w | c) = \frac{\text{count}(w, c) + 1}{\text{count}(c) + |V|}$$

Priors:

$$P(f) = \frac{3}{4}$$

$$P(g) = \frac{1}{4}$$

Conditional Probabilities:

$$P(\text{line} | f) = (1+1) / (8+6) = 2/14$$

$$P(\text{guitar} | f) = (0+1) / (8+6) = 1/14$$

$$P(\text{jazz} | f) = (0+1) / (8+6) = 1/14$$

$$P(\text{line} | g) = (1+1) / (3+6) = 2/9$$

$$P(\text{guitar} | g) = (1+1) / (3+6) = 2/9$$

	Doc	Words	Class
Training	1	fish smoked fish	f
	2	fish line	f
	3	fish haul smoked	f
	4	guitar jazz line	g
Test	5	line guitar jazz jazz	?

$$V = \{\text{fish, smoked, line, haul, guitar, jazz}\}$$

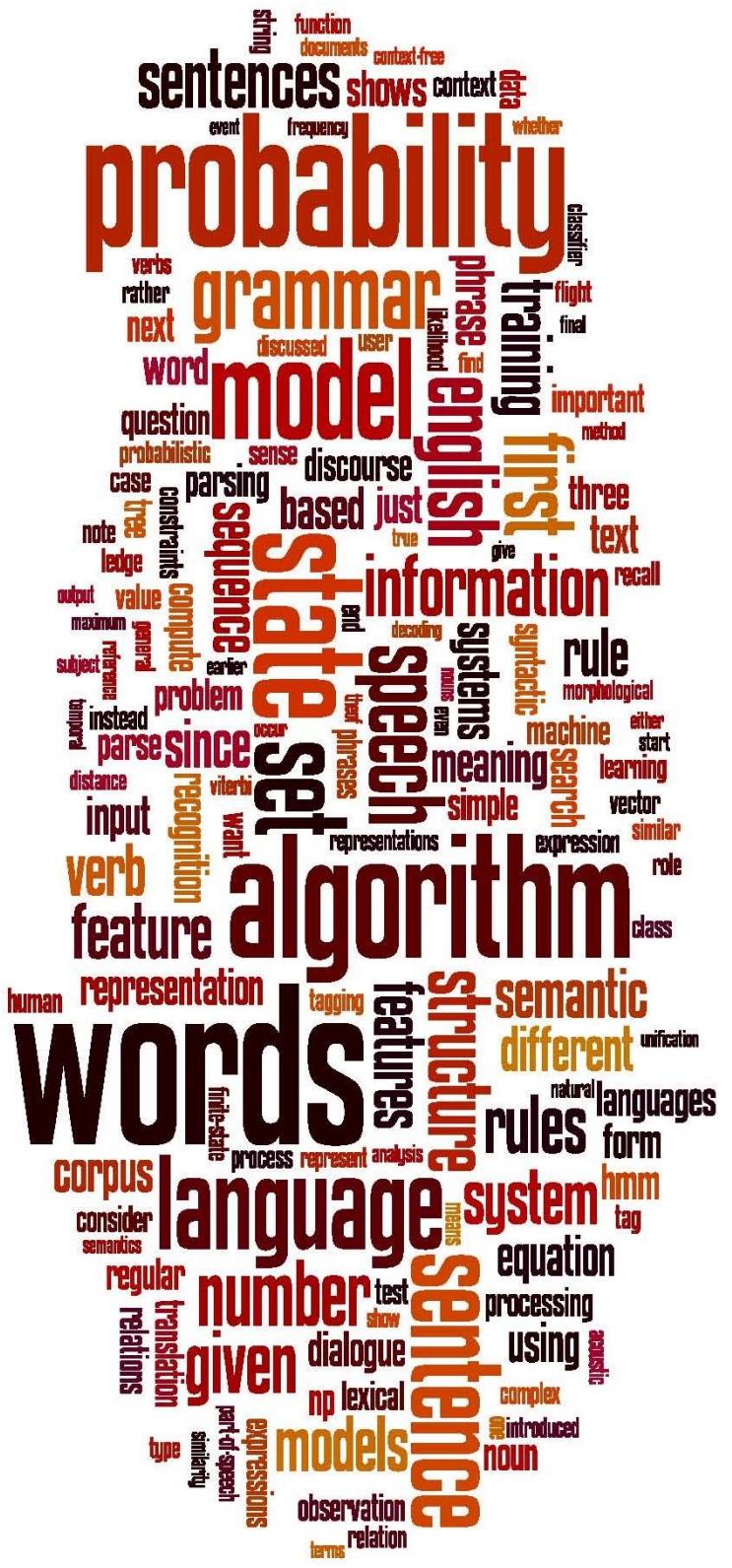
Choosing a class:

$$P(f | d5) \propto 3/4 * 2/14 * (1/14)^2 * 1/14$$

$$\approx 0.00003$$

$$P(g | d5) \propto 1/4 * 2/9 * (2/9)^2 * 2/9$$

$$\approx 0.0006$$



Word Sense Disambiguation

Evaluations and Baselines

WSD Evaluations and baselines

- Best evaluation: **extrinsic ('end-to-end', 'task-based') evaluation**
 - Embed WSD algorithm in a task and see if you can do the task better!
- What we often do for convenience: **intrinsic evaluation**
 - Exact match **sense accuracy**
 - % of words tagged identically with the human-manual sense tags
 - Usually evaluate using **held-out data** from same labeled corpus
- Baselines
 - Most frequent sense
 - The Lesk algorithm

Most Frequent Sense

- WordNet senses are ordered in frequency order
- So “most frequent sense” in WordNet = “take the first sense”
- Sense frequencies come from the *SemCor* corpus

Freq	Synset	Gloss
338	plant ¹ , works, industrial plant	buildings for carrying on industrial labor
207	plant ² , flora, plant life	a living organism lacking the power of locomotion
2	plant ³	something planted secretly for discovery by another
0	plant ⁴	an actor situated in the audience whose acting is rehearsed but seems spontaneous to the audience

Ceiling

- Human inter-annotator agreement
 - Compare annotations of two humans
 - On same data
 - Given same tagging guidelines
- Human agreements on all-words corpora with WordNet style senses
 - 75%–80%

Word Sense Disambiguation

**Dictionary and
Thesaurus Methods**

The Simplified Lesk algorithm

- Let's disambiguate “**bank**” in this sentence:

The **bank** can guarantee deposits will eventually cover future tuition costs because it invests in adjustable--rate mortgage securities.

- given the following two WordNet senses:

bank ¹	Gloss:	a financial institution that accepts deposits and channels the money into lending activities
	Examples:	“he cashed a check at the bank”, “that bank holds the mortgage on my home”
bank ²	Gloss:	sloping land (especially the slope beside a body of water)
	Examples:	“they pulled the canoe up on the bank”, “he sat on the bank of the river and watched the currents”

The Simplified Lesk algorithm

Choose sense with most word overlap between gloss and context
(not counting function words)

The **bank** can guarantee **deposits** will eventually cover future tuition costs because it invests in adjustable--rate **mortgage** securities.

bank ¹	Gloss:	a financial institution that accepts deposits and channels the money into lending activities
	Examples:	“he cashed a check at the bank”, “that bank holds the mortgage on my home”
bank ²	Gloss:	sloping land (especially the slope beside a body of water)
	Examples:	“they pulled the canoe up on the bank”, “he sat on the bank of the river and watched the currents”

The Corpus Lesk algorithm

- Assumes we have some sense--labeled data (like SemCor)
- Take all the sentences with the relevant word sense:

*These short, "streamlined" meetings usually are sponsored by local **banks**¹, Chambers of Commerce, trade associations, or other civic organizations.*
- Now add these to the gloss + examples for each sense, call it the “signature” of a sense.
- Choose sense with most word overlap between context and signature.

Corpus Lesk: IDF weighting

- Instead of just removing function words
 - Weigh each word by its ‘promiscuity’ across documents
 - Down-weights words that occur in every ‘document’ (gloss, example, etc)
 - These are generally function words, but is a more fine-grained measure
- Weigh each overlapping word by **inverse document frequency**

Corpus Lesk: IDF weighting

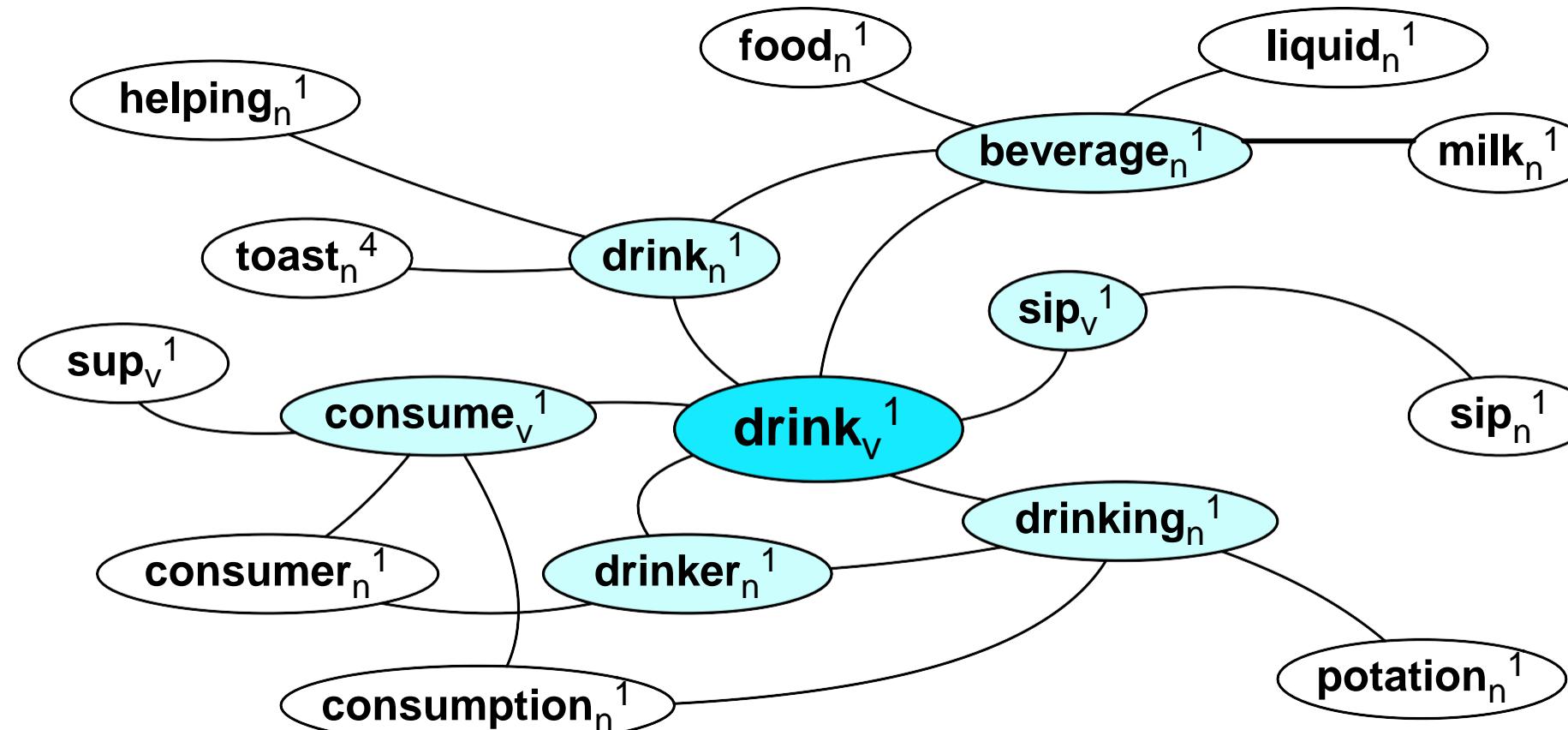
- Weigh each overlapping word by **inverse document frequency**
 - N is the total number of documents
 - df_i = “document frequency of word i ”
 - $= \#$ of documents with word i

$$idf_i = \log \left(\frac{N}{df_i} \right)$$

$$score(sense_i, context_j) = \sum_{w \in overlap(signature, context)} idf_w$$

Graph-based methods

- First, WordNet can be viewed as a graph
 - senses are nodes
 - relations (hypernymy, meronymy) are edges
 - Also add edge between word and unambiguous gloss words



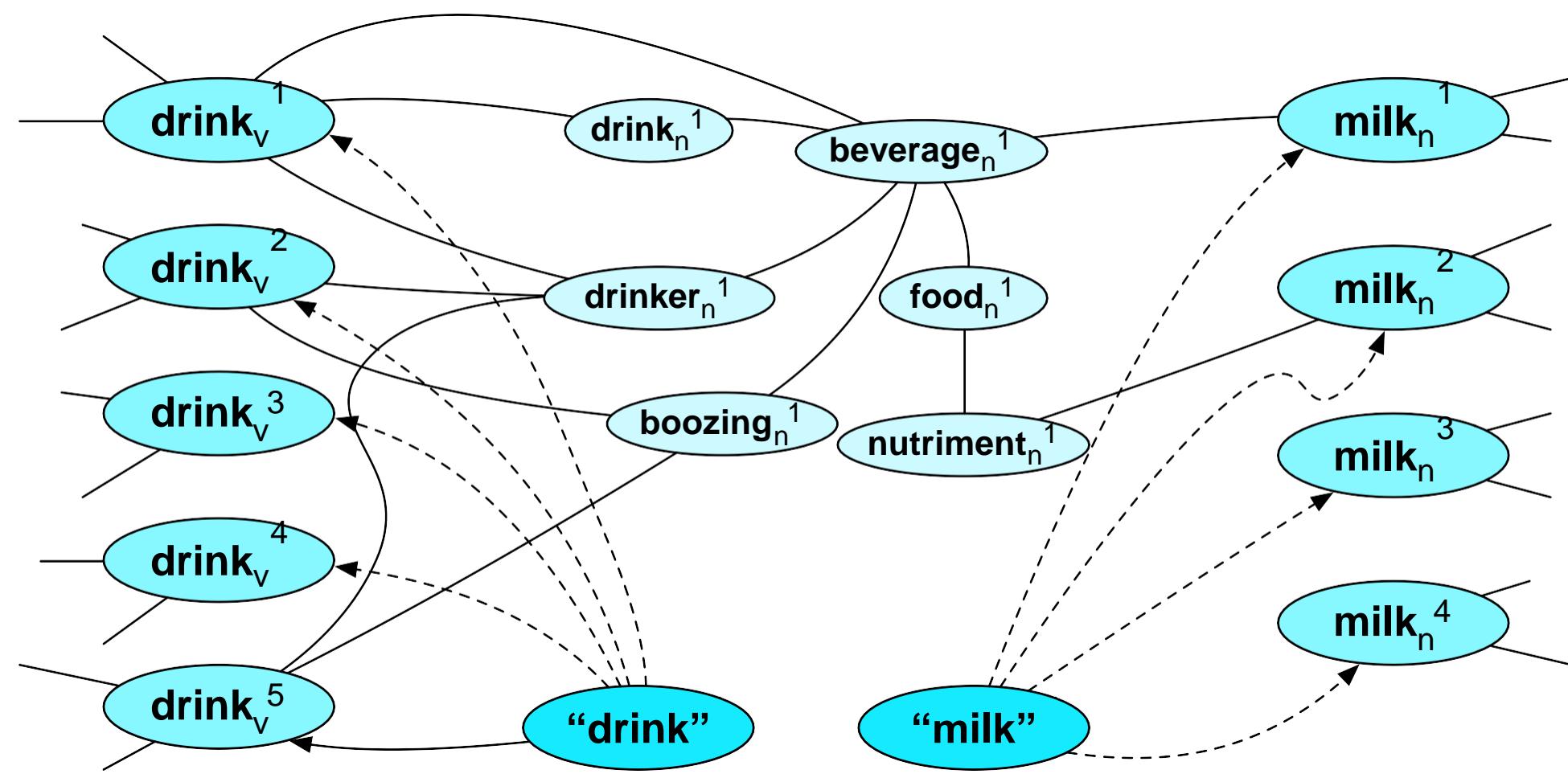
How to use the graph for WSD

- Insert target word and words in its sentential context into the graph, with directed edges to their senses

“She drank some milk”

- Now choose the *most central* sense

Add some probability to “drink” and “milk” and compute node with highest “pagerank”



Word Sense Disambiguation

Semi-Supervised
Learning

Semi--Supervised Learning

Problem: supervised and dictionary--based approaches require large hand-built resources

What if you don't have so much training data?

Solution: Bootstrapping

Generalize from a very small hand--labeled seed--set.

Bootstrapping

- For bass
 - Rely on “One sense per collocation” rule
 - A word reoccurring in collocation with the same word will almost surely have the same sense.
 - the word play occurs with the music sense of bass
 - the word fish occurs with the fish sense of bass

Sentences extracting using “fish” and “play”

We need more good teachers – right now, there are only a half a dozen who can **play** the free **bass** with ease.

An electric guitar and **bass player** stand off to one side, not really part of the scene, just as a sort of nod to gringo expectations perhaps.

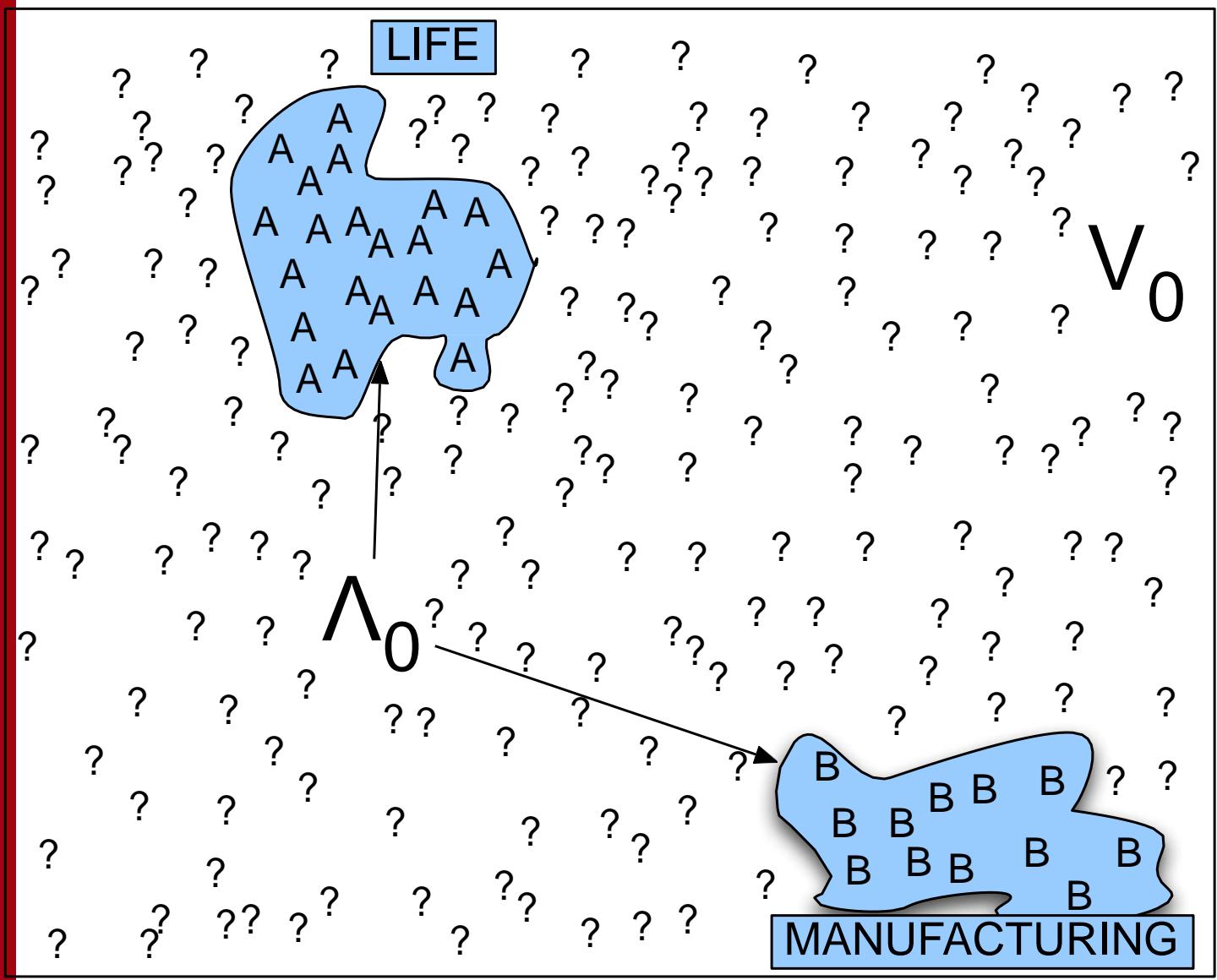
The researchers said the worms spend part of their life cycle in such **fish** as Pacific salmon and striped **bass** and Pacific rockfish or snapper.

And it all started when **fishermen** decided the striped **bass** in Lake Mead were too skinny.

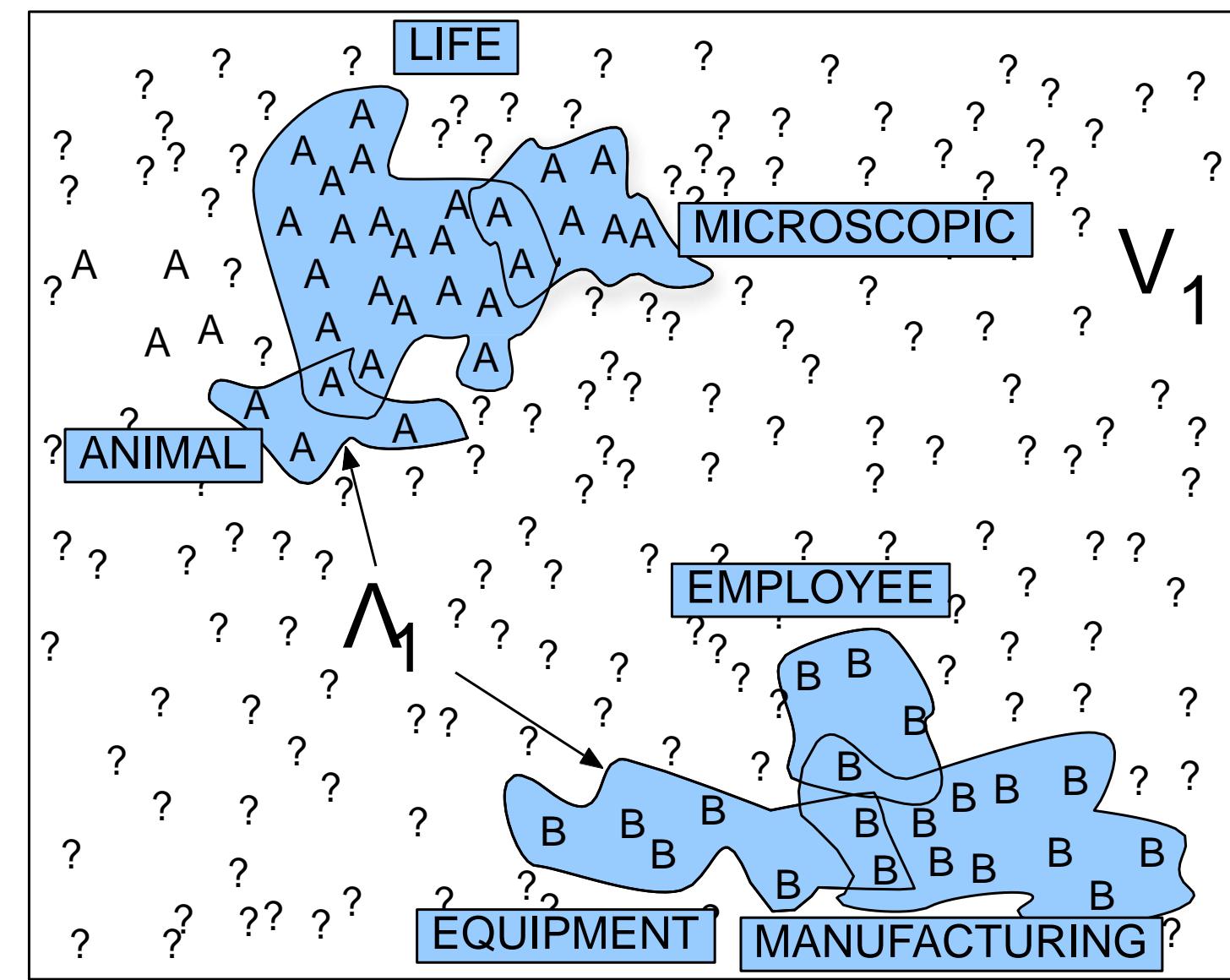
Summary: generating seeds

- 1) Hand labeling
- 2) “One sense per collocation”:
 - A word reoccurring in collocation with the same word will almost surely have the same sense.
- 3) “One sense per discourse”:
 - The sense of a word is highly consistent within a document -Yarowsky (1995)
 - (At least for non-function words, and especially topic-specific words)

Stages in the Yarowsky bootstrapping algorithm for the word “plant”



(a)



(b)

Summary

- Word Sense Disambiguation: choosing correct sense in context
- Applications: MT, QA, etc.
- Three classes of Methods
 - Supervised Machine Learning: Naive Bayes classifier
 - Thesaurus/Dictionary Methods
 - Semi-Supervised Learning
- Main intuition
 - There is lots of information in a word's context
 - Simple algorithms based just on word counts can be surprisingly good

Language Modeling

Introduction to N-grams

Probabilistic Language Models

Today's goal: assign a probability to a sentence

- Machine Translation:
 - $P(\text{high winds tonite}) > P(\text{large winds tonite})$
- Spell Correction

Why?

- The office is about fifteen minuets from my house
 - $P(\text{about fifteen minutes from}) > P(\text{about fifteen minuets from})$
- Speech Recognition
 - $P(\text{I saw a van}) \gg P(\text{eyes awe of an})$
- + Summarization, question-answering, etc., etc.!!

Probabilistic Language Modeling

Goal: compute the probability of a sentence or sequence of words:

$$P(W) = P(w_1, w_2, w_3, w_4, w_5 \dots w_n)$$

Related task: probability of an upcoming word:

$$P(w_5 | w_1, w_2, w_3, w_4)$$

A model that computes either of these:

$P(W)$ or $P(w_n | w_1, w_2 \dots w_{n-1})$ is called a **language model**.

Better: **the grammar** But **language model** or **LM** is standard

How to compute $P(W)$

How to compute this joint probability:

- $P(\text{its, water, is, so, transparent, that})$

Intuition: let's rely on the Chain Rule of Probability

Reminder: The Chain Rule

Recall the definition of conditional probabilities

$$p(B|A) = P(A,B)/P(A) \quad \text{Rewriting: } P(A,B) = P(A)P(B|A)$$

More variables:

$$P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)$$

The Chain Rule in General

$$P(x_1, x_2, x_3, \dots, x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)\dots P(x_n|x_1, \dots, x_{n-1})$$

The Chain Rule applied to compute joint probability of words in sentence

$$P(w_1 w_2 \square \dots w_n) = \tilde{\bigcirc}_i P(w_i | w_1 w_2 \square \dots w_{i-1})$$

$P(\text{"its water is so transparent"}) =$

$P(\text{its}) \times P(\text{water}|\text{its}) \times P(\text{is}|\text{its water})$

$\times P(\text{so}|\text{its water is}) \times P(\text{transparent}|\text{its water is so})$

How to estimate these probabilities

Could we just count and divide?

$P(\text{the} \mid \text{its water is so transparent that}) =$

$\frac{\text{Count}(\text{its water is so transparent that the})}{\text{Count}(\text{its water is so transparent that})}$

No! Too many possible sentences!

We'll never see enough data for estimating these

Markov Assumption



Andrei Markov

Simplifying assumption:

$$P(\text{the} \mid \text{its water is so transparent that}) \gg P(\text{the} \mid \text{that})$$

Or maybe

$$P(\text{the} \mid \text{its water is so transparent that}) \gg P(\text{the} \mid \text{transparent that})$$

Markov Assumption

$$P(w_1 w_2 \square \dots w_n) \gg \tilde{\bigcirc} P(w_i \mid w_{i-k} \square \dots w_{i-1})$$

In other words, we approximate each component in the product

$$P(w_i \mid w_1 w_2 \square \dots w_{i-1}) \gg P(w_i \mid w_{i-k} \square \dots w_{i-1})$$

Simplest case: Unigram model

$$P(w_1 w_2 \square \dots w_n) \gg \tilde{\bigcirc}_i P(w_i)$$

Some 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:

$$P(w_i | w_1 w_2 \square w_{i-1}) \gg P(w_i | w_{i-1})$$

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

N-gram models

We can extend to trigrams, 4-grams, 5-grams

In general this is an insufficient model of language

- because language has **long-distance dependencies**:

“The computer which I had just put into the machine room
on the fifth floor crashed.”

But we can often get away with N-gram models

Language Modeling

Introduction to N-grams

Language Modeling

Estimating N-gram Probabilities

Estimating bigram probabilities

The Maximum Likelihood Estimate

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

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

An example

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

<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

Out of 9222 sentences

	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 unigrams:

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

Result:

	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

$$P(< s > \mid \text{I want english food } < /s >) =$$

$$P(\text{I} \mid < s >)$$

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

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

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

$$\times P(< /s > \mid \text{food})$$

$$= .000031$$

What kinds of knowledge?

$P(\text{english} | \text{want}) = .0011$

$P(\text{chinese} | \text{want}) = .0065$

$P(\text{to} | \text{want}) = .66$

$P(\text{eat} | \text{to}) = .28$

$P(\text{food} | \text{to}) = 0$

$P(\text{want} | \text{spend}) = 0$

$P(\text{i} | <\text{s}>) = .25$

Practical Issues

We do everything in log space

- Avoid underflow
- (also adding is faster than multiplying)

$$\log(p_1 \cdot p_2 \cdot p_3 \cdot p_4) = \log p_1 + \log p_2 + \log p_3 + \log p_4$$

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 Book N-grams

<http://ngrams.googlelabs.com/>

Language Modeling

Estimating N-gram Probabilities

Language Modeling

Evaluation and Perplexity

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?

We train parameters of our model on a **training set**.

We test the model’s performance on data we haven’t seen.

- A **test set** is an unseen dataset that is different from our training set, totally unused.
- An **evaluation metric** tells us how well our model does on the test set.

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 (in-vivo) evaluation of N-gram models

Extrinsic evaluation

- Time-consuming; can take days or weeks

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 _____

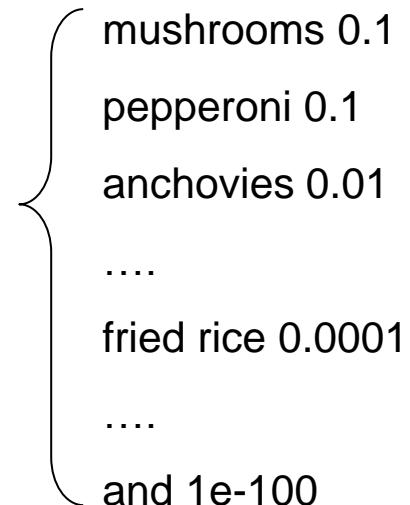
The 33rd President of the US was _____

I saw a _____

- 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



Perplexity

The best language model is one that best predicts an unseen test set

- Gives the highest $P(\text{sentence})$

Perplexity is the inverse probability of the test set, normalized by the number of words:

Chain rule:

For bigrams:

$$PP(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}}$$

$$PP(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_1 \dots w_{i-1})}}$$

$$PP(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_{i-1})}}$$

Minimizing perplexity is the same as maximizing probability

The Shannon Game intuition for perplexity

From Josh Goodman

Perplexity is weighted equivalent branching factor

How hard is the task of recognizing digits '0,1,2,3,4,5,6,7,8,9'

- Perplexity 10

How hard is recognizing (30,000) names at Microsoft.

- Perplexity = 30,000

Let's imagine a call-routing phone system gets 120K calls and has to recognize

- "Operator" (let's say this occurs 1 in 4 calls)
- "Sales" (1 in 4)
- "Technical Support" (1 in 4)
- 30,000 different names (each name occurring 1 time in the 120K calls)
- What is the perplexity? Next slide

The Shannon Game intuition for perplexity

Josh Goodman: imagine a call-routing phone system gets 120K calls and has to recognize

- "Operator" (let's say this occurs 1 in 4 calls)
- "Sales" (1 in 4)
- "Technical Support" (1 in 4)
- 30,000 different names (each name occurring 1 time in the 120K calls)

We get the perplexity of this sequence of length 120K by first multiplying 120K probabilities (90K of which are 1/4 and 30K of which are 1/120K), and then taking the inverse 120,000th root:

$$\text{Perp} = (\frac{1}{4} * \frac{1}{4} * \frac{1}{4} * \frac{1}{4} * \dots * \frac{1}{120K} * \frac{1}{120K} * \dots)^{-1/120K}$$

But this can be arithmetically simplified to just $N = 4$: the operator (1/4), the sales (1/4), the tech support (1/4), and the 30,000 names (1/120,000):

$$\text{Perplexity} = ((\frac{1}{4} * \frac{1}{4} * \frac{1}{4} * \frac{1}{4})^{1/4})^{-1/4} = 52.6$$

Perplexity as branching factor

Let's suppose a sentence consisting of random digits

What is the perplexity of this sentence according to a model that assign $P=1/10$ to each digit?

$$\begin{aligned} \text{PP}(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \left(\frac{1}{10}\right)^{-\frac{1}{N}} \\ &= \frac{1}{10}^{-1} \\ &= 10 \end{aligned}$$

Lower perplexity = better model

Training 38 million words, test 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Language Modeling

Evaluation and Perplexity

Language Modeling

Generalization and zeros

The Shannon Visualization Method

Choose a random bigram

<s> I

(<s>, w) according to its probability

I want

Now choose a random bigram
(w, x) according to its probability

want to

And so on until we choose </s>

to eat

Then string the words together

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

2
gram

–Hill he late speaks; or! a more to leg less first you enter
–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live
king. Follow.

3
gram

–What means, sir. I confess she? then all sorts, he is trim, captain.
–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say,
'tis done.

4
gram

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

N=884,647 tokens, V=29,066

Shakespeare produced 300,000 bigram types out of $V^2 = 844$ million possible bigrams.

- So 99.96% of the possible bigrams were never seen (have zero entries in the table)

Quadrigrams worse: What's coming out looks like Shakespeare because it *is* Shakespeare

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

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.

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

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

Zero probability bigrams

Bigrams with zero probability

- mean that we will assign 0 probability to the test set!

And hence we cannot compute perplexity (can't divide by 0)!

Language Modeling

Generalization and zeros

Language Modeling

Smoothing: Add-one
(Laplace) smoothing

The intuition of smoothing (from Dan Klein)

When we have sparse statistics:

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

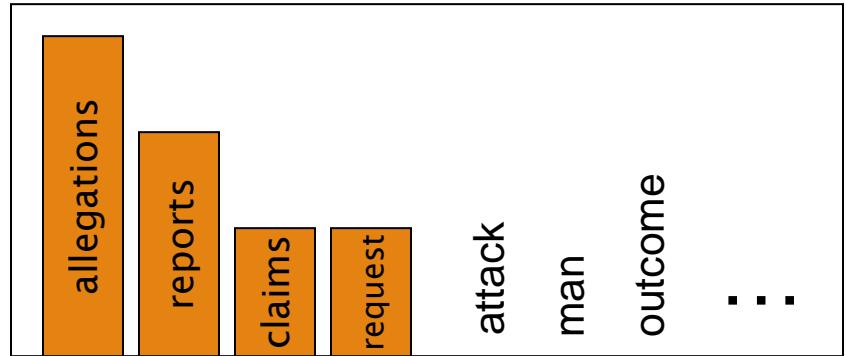
3 allegations

2 reports

1 claims

1 request

7 total



Steal probability mass to generalize better

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

2.5 allegations

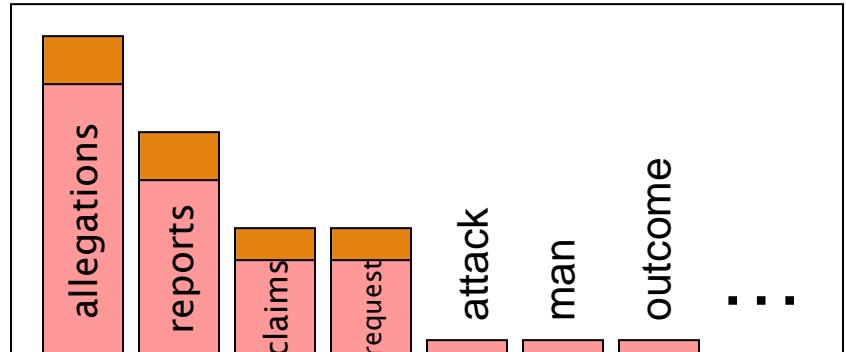
1.5 reports

0.5 claims

0.5 request

2 other

7 total



Add-one estimation

Also called Laplace smoothing

Pretend we saw each word one more time than we did
Just add one to all the counts!

MLE estimate:

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

Add-1 estimate:

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

Maximum Likelihood Estimates

The maximum likelihood estimate

- of some parameter of a model M from a training set T
- maximizes the likelihood of the training set T given the model M

Suppose the word “bagel” occurs 400 times in a corpus of a million words

What is the probability that a random word from some other text will be “bagel”?

MLE estimate is $400/1,000,000 = .0004$

This may be a bad estimate for some other corpus

- But it is the **estimate** that makes it **most likely** that “bagel” will occur 400 times in a million word corpus.

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

$$P^*(w_n | w_{n-1}) = \frac{C(w_{n-1} w_n) + 1}{C(w_{n-1}) + V}$$

	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

$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n) + 1] \times C(w_{n-1})}{C(w_{n-1}) + V}$$

	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 estimation 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

- For text classification
- In domains where the number of zeros isn't so huge.

Language Modeling

Smoothing: Add-one
(Laplace) smoothing

Language Modeling

Interpolation, Backoff, and Web-Scale LMs

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:

- mix unigram, bigram, trigram

Interpolation works better

Linear Interpolation

Simple interpolation

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

$$\sum_i \lambda_i = 1$$

Lambdas conditional on context:

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

How to set the lambdas?

Use a **held-out** corpus

Training Data

Held-Out
Data

Test
Data

Choose λ s to maximize the probability of held-out data:

- Fix the N-gram probabilities (on the training data)
- Then search for λ s that give largest probability to held-out set:

$$\log P(w_1 \dots w_n \mid M(\text{/}_1 \dots \text{/}_k)) = \sum_i \log P_{M(\text{/}_1 \dots \text{/}_k)}(w_i \mid w_{i-1})$$

Unknown words: Open versus 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

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

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
- 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)

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 smoothing:

- OK for text categorization, not for language modeling

The most commonly used method:

- Extended Interpolated Kneser-Ney

For very large N-grams like the Web:

- Stupid backoff

Advanced Language Modeling

Discriminative models:

- choose n-gram weights to improve a task, not to fit the training set

Parsing-based models

Caching Models

- Recently used words are more likely to appear

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

- These turned out to perform very poorly for speech recognition (why?)

Language Modeling

Interpolation, Backoff, and Web-Scale LMs

Language Modeling

Advanced:
Kneser-Ney Smoothing

Absolute discounting: just subtract a little from each count

Suppose we wanted to subtract a little from a count of 4 to save probability mass for the zeros

How much to subtract ?

Church and Gale (1991)'s clever idea

Divide up 22 million words of AP Newswire

- Training and held-out set
- for each bigram in the training set
- see the actual count in the held-out set!

It sure looks like $c^* = (c - .75)$

Bigram count in training	Bigram count in heldout set
0	.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

Absolute Discounting Interpolation

Save ourselves some time and just subtract 0.75 (or some d)!

$$P_{\text{AbsoluteDiscounting}}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + / (\overset{\swarrow}{w_{i-1}}) P(w)$$

discounted bigram Interpolation weight
 ↑
 unigram

- (Maybe keeping a couple extra values of d for counts 1 and 2)

But should we really just use the regular unigram $P(w)$?

Kneser-Ney Smoothing I

Better estimate for probabilities of lower-order unigrams!

- Shannon game: *I can't see without my reading _____?* *K^{glasses}*
- “Kong” turns out to be more common than “glasses”
- ... but “Kong” always follows “Hong”

The unigram is useful exactly when we haven’t seen this bigram!

Instead of $P(w)$: “How likely is w ”

$P_{\text{continuation}}(w)$: “How likely is w to appear as a novel continuation?”

- For each word, count the number of bigram types it completes
- Every bigram type was a novel continuation the first time it was seen

$$P_{\text{CONTINUATION}}(w) \propto |\{w_{i-1} : c(w_{i-1}, w) > 0\}|$$

Kneser-Ney Smoothing II

How many times does w appear as a novel continuation:

$$P_{CONTINUATION}(w) \propto |\{w_{i-1} : c(w_{i-1}, w) > 0\}|$$

Normalized by the total number of word bigram types

$$\left| \{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\} \right|$$

$$P_{CONTINUATION}(w) = \frac{\left| \{w_{i-1} : c(w_{i-1}, w) > 0\} \right|}{\left| \{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\} \right|}$$

Kneser-Ney Smoothing III

Alternative metaphor: The number of # of word types seen to precede w

$$|\{w_{i-1} : c(w_{i-1}, w) > 0\}|$$

normalized by the # of words preceding all words:

$$P_{CONTINUATION}(w) = \frac{|\{w_{i-1} : c(w_{i-1}, w) > 0\}|}{\sum_{w'} |\{w'_{i-1} : c(w'_{i-1}, w') > 0\}|}$$

A frequent word (Kong) occurring in only one context (Hong) will have a low continuation probability

Kneser-Ney Smoothing IV

$$P_{KN}(w_i \mid w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + / (w_{i-1}) P_{CONTINUATION}(w_i)$$

λ is a normalizing constant; the probability mass we've discounted

$$/ (w_{i-1}) = \frac{d}{c(w_{i-1})} \left| \{w : c(w_{i-1}, w) > 0\} \right|$$

the normalized discount

The number of word types that can follow w_{i-1}
= # of word types we discounted
= # of times we applied normalized discount

Kneser-Ney Smoothing: Recursive formulation

$$P_{KN}(w_i | w_{i-n+1}^{i-1}) = \frac{\max(c_{KN}(w_{i-n+1}^i) - d, 0)}{c_{KN}(w_{i-n+1}^{i-1})} + / (w_{i-n+1}^{i-1}) P_{KN}(w_i | w_{i-n+2}^{i-1})$$

$$c_{KN}(\cdot) = \begin{cases} \hat{\cdot} & \text{count}(\cdot) \text{ for the highest order} \\ \dagger & \text{continuationcount}(\cdot) \text{ for lower order} \end{cases}$$

Continuation count = Number of unique single word contexts for •

Language Modeling

Advanced:
Kneser-Ney Smoothing