

Introduction to Natural Language Processing

By J. H. Wang

Feb. 22, 2023

Outline

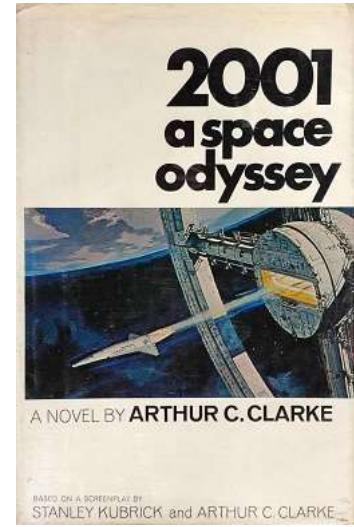
- What is NLP?
- Example Tasks
- The state-of-the-art

What is Natural Language Processing

- (speech and) language processing
- Human language technology
- Computational linguistics
- To give computers ability to process human language
- To enable **human-machine communication**
- E.g. conversational agent

Example: 2001: A Space Odyssey

- A 1968 epic science fiction film produced and directed by Stanley Kubrick
- Based on a novel by Arthur C. Clarke
- HAL 9000 computer
 - Speaking and understanding English
 - Even reading lips
- Conversational agent, dialog system
 - Input: automatic speech recognition, natural language understanding
 - Output: dialogue and response planning, speech synthesis



Modern Examples: Chatbots

- ChatGPT, Bard
- Mobile phone assistants
 - Apple Siri, ...
- Smart speakers
 - Google Home, Amazon Alexa, LINE Clova, ...
- Other applications:
 - Social networking platforms
 - Healthcare
 - Banking
 - ...



Example Language-related Tasks

- Machine Translation
 - Automatically translate a document from one language to another
- Web-based question answering, for example:
 - What year was Abraham Lincoln born?
 - How many states were in the United States that year?
 - How much Chinese silk was exported to England by the end of the 18th century?
 - What do scientists think about the ethics of human cloning?
- Information extraction, word sense disambiguation
- Spelling correction, grammar checking, ...

Question Answering: IBM's Watson

- Won Jeopardy on February 16, 2011!

WILLIAM WILKINSON'S
“AN ACCOUNT OF THE PRINCIPALITIES OF
WALLACHIA AND MOLDOVIA”
INSPIRED THIS AUTHOR’S
MOST FAMOUS NOVEL



Bram Stoker

Information Extraction

Subject: **curriculum meeting**

Date: January 15, 2012

To: Dan Jurafsky

Event: Curriculum mtg
Date: Jan-16-2012
Start: 10:00am
End: 11:30am
Where: Gates 159

Hi Dan, we've now scheduled the curriculum meeting.

It will be in **Gates 159** tomorrow from 10:00-11:30.

-Chris

Create new Calendar entry



Information Extraction & Sentiment Analysis



Attributes:

zoom
affordability
size and weight
flash
ease of use

Size and weight

- ✓ • nice and compact to carry!
- ✓ • since the camera is small and light, I won't have to carry heavy, bulky professional cameras either!
- ✓ • the camera feels flimsy, is plastic and very I very delicate in the handling of this camera

Size and weight	<p>This is an excellent little camera. It's compact when not being used but is very light and easy to hold. The digital image format is great...I love using this camera because it's so easy to move the camera to a PC and download pictures. The only缺点 is it's a bit slow. It takes about 2 seconds to load each picture.</p>
Affordability	<p>I have been looking for a basic camera, something simple which is easy to use and I have found it! It's very good value for money. It does not have any extras like a wide angle lens or anything like this, but this is not what you need to be aiming for though. Never use one without a lens cap.</p>
Size and weight	<p>A really good quality plastic camera which is a delight to look at and good to hold. The build quality is excellent and the camera is very light. It's a great choice for a beginner. This camera is perfect for a walk around with the children - manual functions do not affect the camera's performance. Never use one without a lens cap.</p>
Flash	<p>Overall a lovely camera. This camera looks great, it's very good, looks great and it's very compact and portable. I would say the camera is better than I expected for its price. The design is nice and the camera is well made. However, there are some things I don't like about the camera. Such as the plastic housing.</p>
Ease of use	<p>A nice compact camera, it has a business-like feel to it. It's not too bulky, it's not too heavy and it's not too small. It's a good camera for someone who is new to cameras and doesn't want to spend a lot of money on a camera. The camera is very easy to use and it's very good for a beginner.</p>

Machine Translation

- Fully automatic

Enter Source Text:

這 不 過 是 一 個 時 間 的 問 題 .

Translation from Stanford's *Phrasal*:

This is only a matter of time.

- Helping human translators

Enter Source Text:

تعرض الرئيس اللبناني أميل لحود لـ #حملة عنيفة في مجلس التراب الذي انعقد امس في جلسة تشريعية عادلة تحولت الى "محاكمة" لـ #رئيس الجمهورية على موقف +ه من المحكمة الدولية و "الملاحظات" التي ادلّ بـ #بها حول هذا الموضوع .

Translate Clear

Enter Translation:

lebanese |

president
suffered
exposed
president emile
before
presented
offer

Done!

Language Technology

making good progress

mostly solved

Spam detection

Let's go to Agra!



Buy V1AGRA ...



Part-of-speech (POS) tagging

ADJ ADJ NOUN VERB ADV

Colorless green ideas sleep furiously.

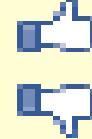
Named entity recognition (NER)

PERSON ORG LOC

Einstein met with UN officials in Princeton

Sentiment analysis

Best roast chicken in San Francisco!



The waiter ignored us for 20 minutes.



Coreference resolution

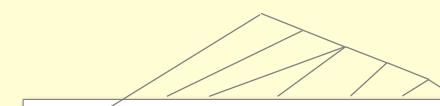
Carter told Mubarak he shouldn't run again.

Word sense disambiguation (WSD)

I need new batteries for my **mouse**.



Parsing



I can see Alcatraz from the window!

Machine translation (MT)

第13届上海国际电影节开幕...



The 13th Shanghai International Film Festival...

Information extraction (IE)

You're invited to our dinner
party, Friday May 27 at 8:30



still really hard

Question answering (QA)

Q. How effective is ibuprofen in reducing fever in patients with acute febrile illness?

Paraphrase

XYZ acquired ABC yesterday

ABC has been taken over by XYZ

Summarization

The Dow Jones is up

The S&P500 jumped

Housing prices rose

Economy is good

Dialog

Where is Citizen Kane playing in SF?



Castro Theatre at 7:30. Do you want a ticket?



Knowledge of Language

- Phonetics and Phonology - knowledge about linguistic sounds
- Morphology - knowledge of the meaningful components of words
- Syntax - knowledge of the structural relationships between words
- Semantics - knowledge of meaning
- Pragmatics - knowledge of the relationship of meaning to the goals and intentions of the speaker
- Discourse - knowledge about linguistic units larger than a single utterance

Ambiguity

- Some input is **ambiguous** if multiple, alternative linguistic structures can be built for it
- For example: *I made her duck.* ?
- What's the meaning?



Possible Answers:

- (1.5) I cooked waterfowl for her.
- (1.6) I cooked waterfowl belonging to her.
- (1.7) I created the (plaster?) duck she owns.
- (1.8) I caused her to quickly lower her head or body.
- (1.9) I waved my magic wand and turned her into undifferentiated waterfowl.

Multiple Meaning Words

duck*

/'dək/ (noun)

a type of swimming bird with webbed feet and a short neck



The **duck** is walking near the lake.

duck*

/'dək/ (verb)

to lower the head quickly in order to avoid being seen or hit



Billy didn't **duck** his head down as he crawled out.

* There are many other definitions of "duck."

Resolving Ambiguities

- Lexical disambiguation
 - Part-of-Speech (POS) Tagging
 - E.g. duck: noun vs. verb
 - Word sense disambiguation
 - E.g. make: create vs. cook
- Syntactic disambiguation
 - Parsing

Why else is natural language understanding difficult?

non-standard English

Great job @justinbieber! Were SOO PROUD of what youve accomplished! U taught us 2 #neversaynever & you yourself should never give up either♥

segmentation issues

the New York-New Haven Railroad
the New York-New Haven Railroad

idioms

dark horse
get cold feet
lose face
throw in the towel

neologisms

unfriend
Retweet
bromance

world knowledge

Mary and Sue are sisters.
Mary and Sue are mothers.

tricky entity names

Where is *A Bug's Life* playing ...
Let It Be was recorded ...
... a mutation on the *for* gene ...

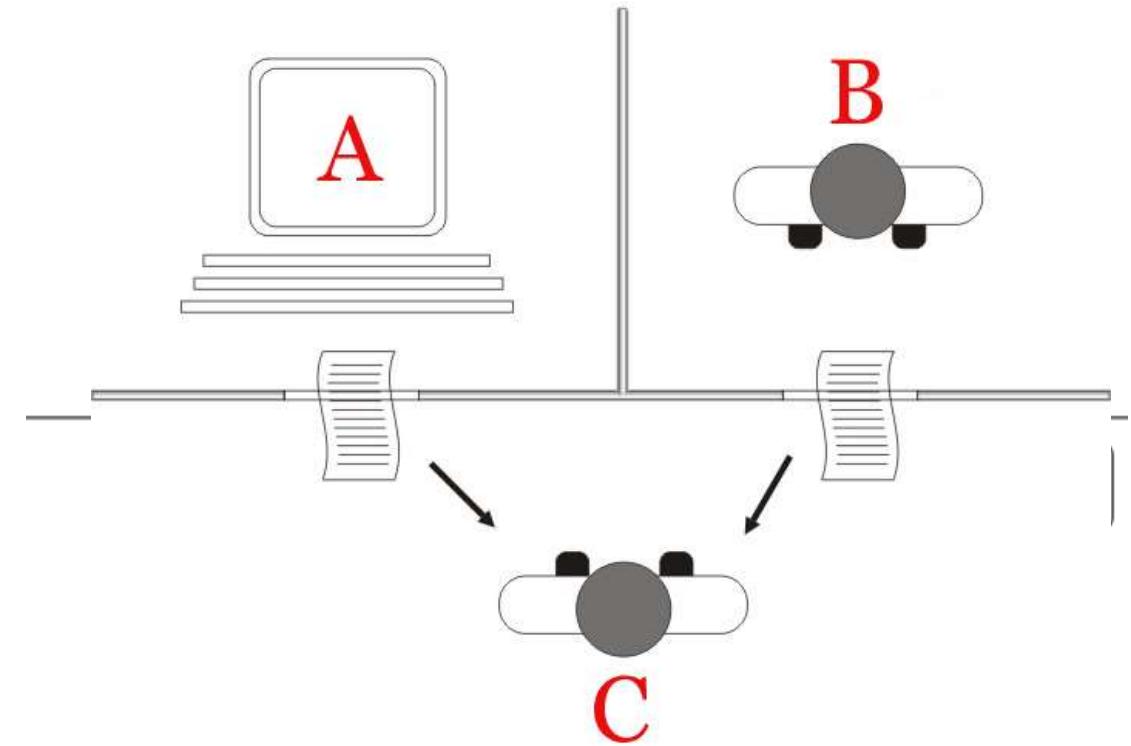
But that's what makes it fun!

Models and Algorithms

- State machines, rule systems, logic, probabilistic models, vector-space models
- State space search algorithm: dynamic programming
- Machine learning algorithm: classification, expectation-maximization (EM), ...
- Deterministic and non-deterministic finite state automata, finite-state transducers
- Regular grammars, context free grammars, ...

Language, Thought, Understanding

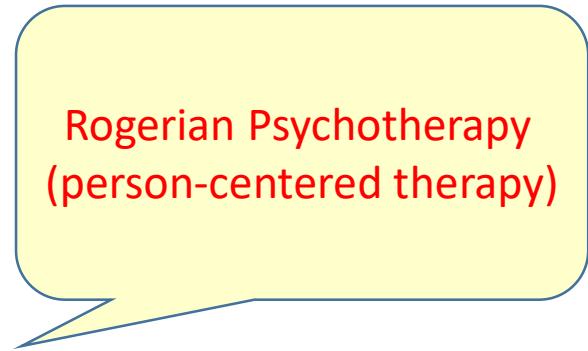
- The ability to process language as skillfully as humans do
 - “Can machines **think**?”
- Turing test: by Alan Turing (1950)
 - “Imitation game”
 - “Can machines **do** what we can do?”
- Conversational agents



- Q: Please write me a sonnet on the topic of the Forth Bridge.
- A: Count me out on this one. I never could write poetry.
- Q: Add 34957 to 70764.
- A: (Pause about 30 seconds and then give answer as) 105621.

Example dialog of ELIZA (1966)

- User1: You are like my father in some ways.
- ELIZA1: WHAT RESEMBLANCE DO YOU SEE
- User2: You are not very aggressive but I think you don't want me to notice that.
- ELIZA2: WHAT MAKES YOU THINK I AM NOT AGGRESSIVE
- User3: You don't argue with me.
- ELIZA3: WHY DO YOU THINK I DON'T ARGUE WITH YOU
- User4: You are afraid of me.
- ELIZA4: DOES IT PLEASE YOU TO BELIEVE I AM AFRAID OF YOU



The State of the Art

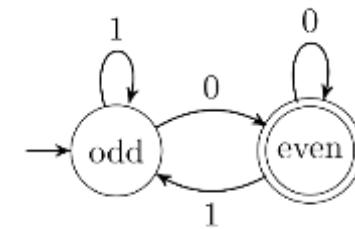
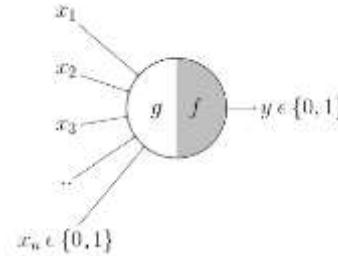
- Travelers calling travel providers interact with conversational agents
- Cars that allow drivers to control by voice
- Searching video on the Web by speech
- Cross-language information retrieval and translation by Google
- Grading and assessing student essays by automated systems
- Interactive virtual agents
- Automated measurement of user opinions, preferences, attitudes in social media
- ...

Brief History of NLP

- Different fields in different departments
 - Computational linguistics: in linguistics
 - Natural language processing: in computer science
 - Speech recognition: in electrical engineering
 - Computational psycholinguistics: in psychology

Foundational Insights: 1940s and 1950s

- Automaton: 1950s
 - Turing (1936)
 - McCulloch-Pitts neuron (1943)
 - Finite automata, regular expressions (1951, 1956)
 - Shannon (1948)
 - Context-free grammar: Chomsky (1956), Backus (1959), Naur (1960)
- Probabilistic or information-theoretic models
 - Shannon: communication, entropy
 - Koenig: sound spectrogram (1946)



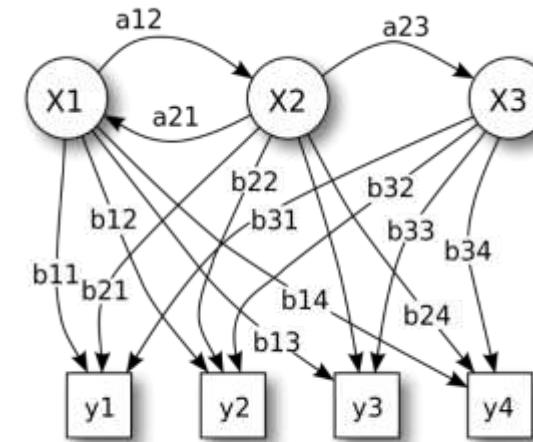
The Two Camps: 1957–1970

- Two paradigms
 - Symbolic
 - Formal language theory, parsing
 - Artificial intelligence: John McCarthy, Marvin Minsky, Claude Shannon, and Nathaniel Rochester (1956)
 - Stochastic
 - Bayesian method



Four Paradigms: 1970–1983

- Stochastic
 - Hidden Markov model
- Logic-based
- Natural language understanding
- Discourse modeling



Empiricism and Finite-State Models Redux: 1983–1993

- Finite-state models
- Probabilistic models

The Field Comes Together: 1994–1999

- Probabilistic and data-driven models had become quite standard throughout natural language processing
- The increases in the speed and memory of computers had allowed commercial exploitation of a number of subareas of speech and language processing, in particular, speech recognition, and spelling and grammar correction
- The rise of the Web emphasized the need for language-based information retrieval and information extraction

The Rise of Machine Learning: 2000–2008

- Large amounts of spoken and written material became widely available through the auspices of the Linguistic Data Consortium (LDC) and other similar organizations
- This increased focus on learning led to a more serious interplay with the statistical machine learning community
 - SVM, maximum entropy, multinomial logistic regression, graphical Bayesian models
- The widespread availability of high-performance computing systems facilitated the training and deployment of systems that could not have been imagined a decade earlier
- Near the end of this period, largely unsupervised statistical approaches began to receive renewed attention
 - Topic modeling, ...

New Potentials for NLP

- Powerful computing resources
- Web as the massive source of information
- Availability of wireless mobile access
- Many new application scenarios

Thanks for Your Attention!

Minimum Edit Distance

J. H. Wang

Mar. 3, 2023

Outline

- Definition of Minimum Edit Distance
- Computing Minimum Edit Distance
- Backtrace for Computing Alignments
- Weighted Minimum Edit Distance
- Minimum Edit Distance in Computational Biology

Minimum Edit Distance

- Definition of Minimum Edit Distance

How similar are two strings?

- Spell correction
 - The user typed “graffe”
Which is closest?
 - graf
 - graft
 - grail
 - giraffe
- Computational Biology
 - Align two sequences of nucleotides

AGGCTATCACCTGACCTCCAGGCCGATGCC
TAGCTATCACGACCGCGGTCGATTGCCCGAC

 - Resulting alignment:

—**AGGCTATCAC**CT**GACCTCCA**GGCCGA--TGCCC---
TAG—**CTATCAC**--**GACC**GC--GGT**CGATT**TGCCCGAC
- Also for Machine Translation, Information Extraction, Speech Recognition

Edit Distance

- The minimum edit distance between two strings
- Is the minimum number of **editing operations**
 - Insertion
 - Deletion
 - Substitution
- Needed to transform one into the other

Minimum Edit Distance

- Two strings and their **alignment**:

I	N	T	E	*	N	T	I	O	N
*	E	X	E	C	U	T	I	O	N

Minimum Edit Distance

I	N	T	E	*	N	T	I	O	N
*	E	X	E	C	U	T	I	O	N
d	s	s		i	s				

- If each operation has cost of 1
 - Distance between these is 5
- If substitutions cost 2 (Levenshtein)
 - Distance between them is 8

Alignment in Computational Biology

- Given a sequence of bases

AGGCTATCACCTGACCTCCAGGCCGATGCC
TAGCTATCACGACC CGCGGT CGATTGCCCGAC

- An alignment:

-**AGGCTATCAC**CT**GACCTCCA**GGCCGA--TGCCC---
TAG-CTATCAC--GACC**GC**--GGT**CGA**TT**TGCCC**GAC

- Given two sequences, align each letter to a letter or gap

Other uses of Edit Distance in NLP

- Evaluating Machine Translation and speech recognition

R Spokesman confirms senior government adviser was appointed
H Spokesman said the senior adviser was appointed

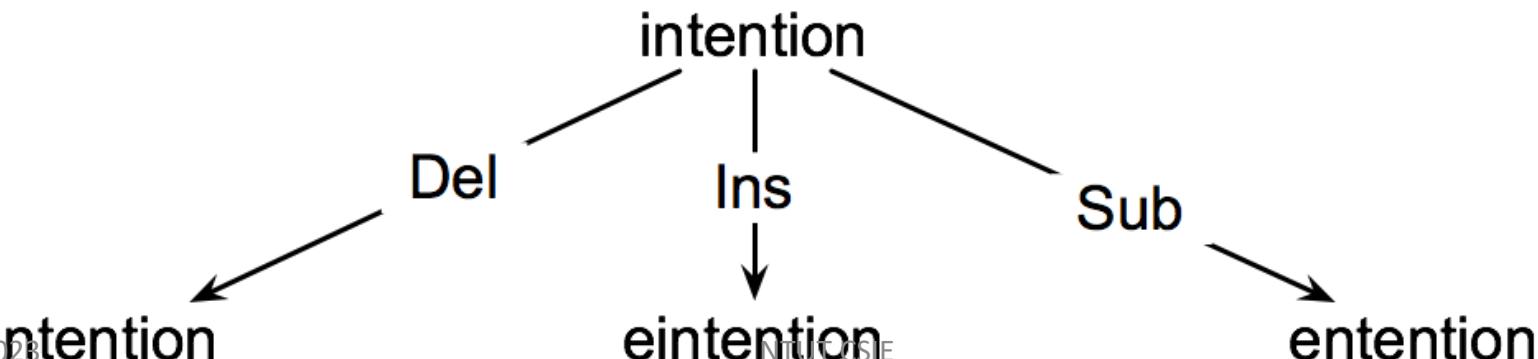
S I D I

- Named Entity Extraction and Entity Coreference

- IBM Inc. announced today
- IBM profits
- Stanford Professor Jennifer Eberhardt announced yesterday
- for Professor Eberhardt...

How to find the Min Edit Distance?

- Searching for a path (sequence of edits) from the start string to the final string:
 - **Initial state:** the word we're transforming
 - **Operators:** insert, delete, substitute
 - **Goal state:** the word we're trying to get to
 - **Path cost:** what we want to minimize: the number of edits



Minimum Edit as Search

- But the space of all edit sequences is huge!
 - We can't afford to navigate naïvely
 - Lots of distinct paths wind up at the same state
 - We don't have to keep track of all of them
 - Just the shortest path to each of those revisited states

Defining Min Edit Distance

- For two strings
 - X of length n
 - Y of length m
- We define $D(i,j)$
 - the edit distance between $X[1..i]$ and $Y[1..j]$
 - i.e., the first i characters of X and the first j characters of Y
 - The edit distance between X and Y is thus $D(n,m)$

Minimum Edit Distance

- Computing Minimum Edit Distance

Dynamic Programming for Minimum Edit Distance

- **Dynamic programming:** A tabular computation of $D(n,m)$
- Solving problems by combining solutions to subproblems.
- Bottom-up
 - We compute $D(i,j)$ for small i,j
 - And compute larger $D(i,j)$ based on previously computed smaller values
 - i.e., compute $D(i,j)$ for all i ($0 < i < n$) and j ($0 < j < m$)

Defining Min Edit Distance (Levenshtein)

- Initialization

$$D(i, 0) = i$$

$$D(0, j) = j$$

- Recurrence Relation:

For each $i = 1 \dots M$

For each $j = 1 \dots N$

$$D(i, j) = \min \left\{ \begin{array}{ll} D(i-1, j) + 1 & \\ D(i, j-1) + 1 & \\ D(i-1, j-1) + 2; & \text{if } X(i) \neq Y(j) \\ 0; & \text{if } X(i) = Y(j) \end{array} \right.$$

- Termination:

$D(N, M)$ is distance

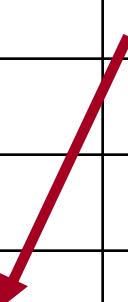
The Edit Distance Table

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

The Edit Distance Table

N	9										
O	8										
I	7										
T	6										
N	5										
E	4										
T	3										
N	2										
I	1										
#	0	1	2	3	4	5	6	7	8	9	
	#	E	X	E	C	U	T	I	O	N	

$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$



Edit Distance

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

N	9										
O	8										
I	7										
T	6										
N	5										
E	4										
T	3										
N	2										
I	1										
#	0	1	2	3	4	5	6	7	8	9	
	#	E	X	E	C	U	T	I	O	N	

The Edit Distance Table

N	9	8	9	10	11	12	11	10	9	8
O	8	7	8	9	10	11	10	9	8	9
I	7	6	7	8	9	10	9	8	9	10
T	6	5	6	7	8	9	8	9	10	11
N	5	4	5	6	7	8	9	10	11	10
E	4	3	4	5	6	7	8	9	10	9
T	3	4	5	6	7	8	7	8	9	8
N	2	3	4	5	6	7	8	7	8	7
I	1	2	3	4	5	6	7	6	7	8
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

Minimum Edit Distance

- Backtrace for Computing Alignments

Computing alignments

- Edit distance isn't sufficient
 - We often need to **align** each character of the two strings to each other
- We do this by keeping a “backtrace”
- Every time we enter a cell, remember where we came from
- When we reach the end,
 - Trace back the path from the upper right corner to read off the alignment

Edit Distance

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

N	9										
O	8										
I	7										
T	6										
N	5										
E	4										
T	3										
N	2										
I	1										
#	0	1	2	3	4	5	6	7	8	9	
	#	E	X	E	C	U	T	I	O	N	

MinEdit with Backtrace

n	9	$\downarrow 8$	$\swarrow \leftarrow \downarrow 9$	$\swarrow \leftarrow \downarrow 10$	$\swarrow \leftarrow \downarrow 11$	$\swarrow \leftarrow \downarrow 12$	$\downarrow 11$	$\downarrow 10$	$\downarrow 9$	$\swarrow 8$	
o	8	$\downarrow 7$	$\swarrow \leftarrow \downarrow 8$	$\swarrow \leftarrow \downarrow 9$	$\swarrow \leftarrow \downarrow 10$	$\swarrow \leftarrow \downarrow 11$	$\downarrow 10$	$\downarrow 9$	$\swarrow 8$	$\leftarrow 9$	
i	7	$\downarrow 6$	$\swarrow \leftarrow \downarrow 7$	$\swarrow \leftarrow \downarrow 8$	$\swarrow \leftarrow \downarrow 9$	$\swarrow \leftarrow \downarrow 10$	$\downarrow 9$	$\swarrow 8$	$\leftarrow 9$	$\leftarrow 10$	
t	6	$\downarrow 5$	$\swarrow \leftarrow \downarrow 6$	$\swarrow \leftarrow \downarrow 7$	$\swarrow \leftarrow \downarrow 8$	$\swarrow \leftarrow \downarrow 9$	$\swarrow 8$	$\leftarrow 9$	$\leftarrow 10$	$\leftarrow \downarrow 11$	
n	5	$\downarrow 4$	$\swarrow \leftarrow \downarrow 5$	$\swarrow \leftarrow \downarrow 6$	$\swarrow \leftarrow \downarrow 7$	$\swarrow \leftarrow \downarrow 8$	$\swarrow \leftarrow \downarrow 9$	$\swarrow \leftarrow \downarrow 10$	$\swarrow \leftarrow \downarrow 11$	$\swarrow \downarrow 10$	
e	4	$\swarrow 3$	$\leftarrow 4$	$\swarrow \leftarrow 5$	$\leftarrow 6$	$\leftarrow 7$	$\leftarrow \downarrow 8$	$\swarrow \leftarrow \downarrow 9$	$\swarrow \leftarrow \downarrow 10$	$\downarrow 9$	
t	3	$\swarrow \leftarrow \downarrow 4$	$\swarrow \leftarrow \downarrow 5$	$\swarrow \leftarrow \downarrow 6$	$\swarrow \leftarrow \downarrow 7$	$\swarrow \leftarrow \downarrow 8$	$\swarrow 7$	$\leftarrow \downarrow 8$	$\swarrow \leftarrow \downarrow 9$	$\downarrow 8$	
n	2	$\swarrow \leftarrow \downarrow 3$	$\swarrow \leftarrow \downarrow 4$	$\swarrow \leftarrow \downarrow 5$	$\swarrow \leftarrow \downarrow 6$	$\swarrow \leftarrow \downarrow 7$	$\swarrow \leftarrow \downarrow 8$	$\downarrow 7$	$\swarrow \leftarrow \downarrow 8$	$\swarrow 7$	
i	1	$\swarrow \leftarrow \downarrow 2$	$\swarrow \leftarrow \downarrow 3$	$\swarrow \leftarrow \downarrow 4$	$\swarrow \leftarrow \downarrow 5$	$\swarrow \leftarrow \downarrow 6$	$\swarrow \leftarrow \downarrow 7$	$\swarrow 6$	$\leftarrow 7$	$\leftarrow 8$	
#	0	1	2	3	4	5	6	7	8	9	
	#	e	x	e	c	u	t	i	o	n	

Adding Backtrace to Minimum Edit Distance

- Base conditions:

$$D(i, 0) = i$$

$$D(0, j) = j$$

Termination:

$D(N, M)$ is distance

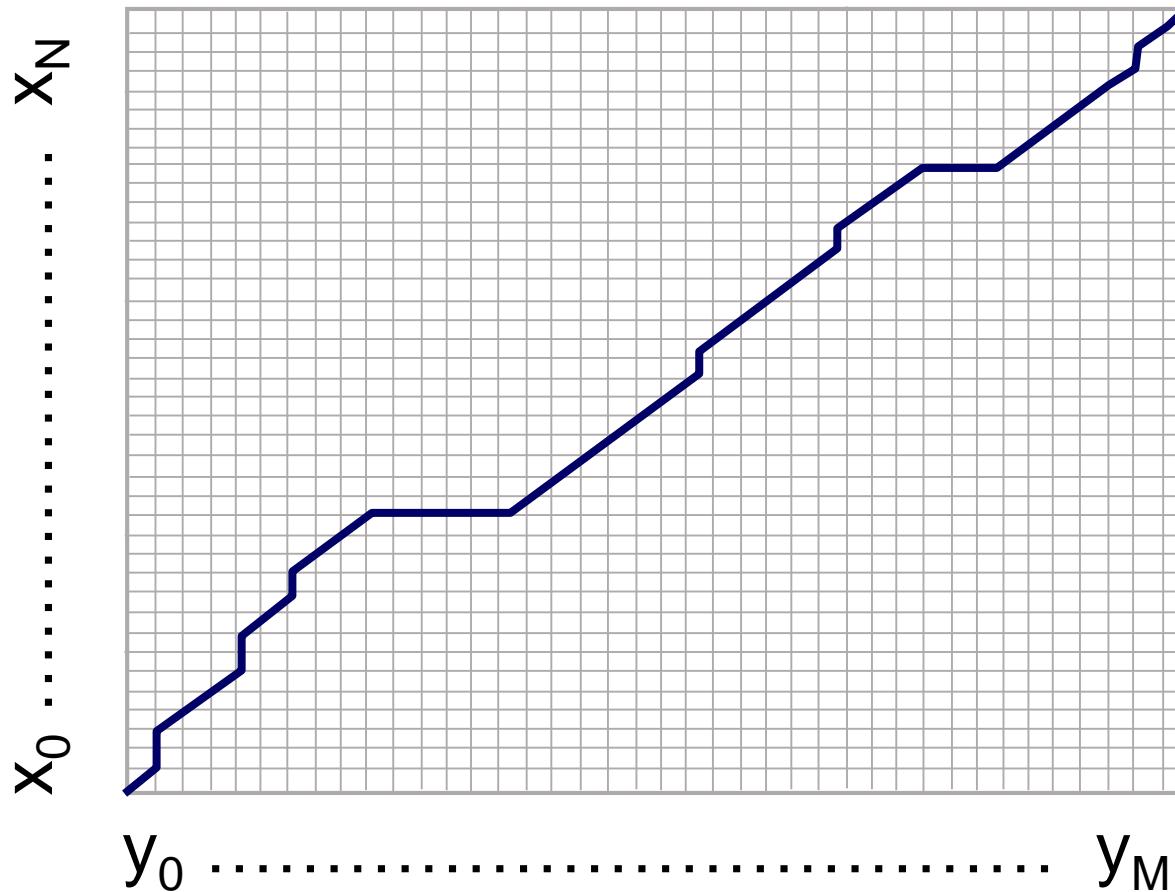
- Recurrence Relation:

For each $i = 1 \dots M$

For each $j = 1 \dots N$

$$D(i, j) = \min \left\{ \begin{array}{l} D(i-1, j) + 1 \text{ deletion} \\ D(i, j-1) + 1 \text{ insertion} \\ D(i-1, j-1) + \begin{cases} 2; & \text{if } X(i) \neq Y(j) \\ 0; & \text{if } X(i) = Y(j) \end{cases} \text{ substitution} \end{array} \right.$$
$$\text{ptr}(i, j) = \left\{ \begin{array}{l} \text{LEFT} \quad \text{insertion} \\ \text{DOWN} \quad \text{deletion} \\ \text{DIAG} \quad \text{substitution} \end{array} \right.$$

The Distance Matrix



Every non-decreasing path
from $(0,0)$ to (M, N)
corresponds to
an alignment
of the two sequences

An optimal alignment is composed
of optimal subalignments

Result of Backtrace

- Two strings and their **alignment**:

I	N	T	E	*	N	T	I	O	N
*	E	X	E	C	U	T	I	O	N

Performance

- Time:
 $O(nm)$
- Space:
 $O(nm)$
- Backtrace
 $O(n+m)$

Minimum Edit Distance

- Weighted Minimum Edit Distance

Weighted Edit Distance

- Why would we add weights to the computation?
 - Spell Correction: some letters are more likely to be mistyped than others
 - Biology: certain kinds of deletions or insertions are more likely than others

Confusion matrix for spelling errors

X	sub[X, Y] = Substitution of X (incorrect) for Y (correct)																										
	Y (correct)																										
a	0	0	7	1	342	0	0	2	118	0	1	0	0	3	76	0	0	1	35	9	9	0	1	0	5	0	
b	0	0	9	9	2	2	3	1	0	0	0	5	11	5	0	10	0	0	2	1	0	0	8	0	0	0	
c	6	5	0	16	0	9	5	0	0	0	1	0	7	9	1	10	2	5	39	40	1	3	7	1	1	0	
d	1	10	13	0	12	0	5	5	0	0	2	3	7	3	0	1	0	43	30	22	0	0	4	0	2	0	
e	388	0	3	11	0	2	2	0	89	0	0	3	0	5	93	0	0	14	12	6	15	0	1	0	18	0	
f	0	15	0	3	1	0	5	2	0	0	0	3	4	1	0	0	0	6	4	12	0	0	2	0	0	0	
g	4	1	11	11	9	2	0	0	0	1	1	3	0	0	2	1	3	5	13	21	0	0	1	0	3	0	
h	1	8	0	3	0	0	0	0	0	0	2	0	12	14	2	3	0	3	1	11	0	0	2	0	0	0	
i	103	0	0	0	146	0	1	0	0	0	0	6	0	0	49	0	0	0	2	1	47	0	2	1	15	0	
j	0	1	1	9	0	0	1	0	0	0	0	2	1	0	0	0	0	0	5	0	0	0	0	0	0	0	
k	1	2	8	4	1	1	2	5	0	0	0	0	5	0	2	0	0	0	0	6	0	0	0	4	0	0	3
l	2	10	1	4	0	4	5	6	13	0	1	0	0	14	2	5	0	11	10	2	0	0	0	0	0	0	
m	1	3	7	8	0	2	0	6	0	0	4	4	0	180	0	6	0	0	9	15	13	3	2	2	3	0	
n	2	7	6	5	3	0	1	19	1	0	4	35	78	0	0	7	0	28	5	7	0	0	1	2	0	2	
o	91	1	1	3	116	0	0	0	25	0	2	0	0	0	14	0	2	4	14	39	0	0	0	0	18	0	
p	0	11	1	2	0	6	5	0	2	9	0	2	7	6	15	0	0	1	3	6	0	4	1	0	0	0	
q	0	0	1	0	0	0	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
r	0	14	0	30	12	2	2	8	2	0	5	8	4	20	1	14	0	0	12	22	4	0	0	1	0	0	
s	11	8	27	33	35	4	0	1	0	1	0	27	0	6	1	7	0	14	0	15	0	0	5	3	20	1	
t	3	4	9	42	7	5	19	5	0	1	0	14	9	5	5	6	0	11	37	0	0	2	19	0	7	6	
u	20	0	0	0	44	0	0	0	64	0	0	0	0	2	43	0	0	4	0	0	0	0	2	0	8	0	
v	0	0	7	0	0	3	0	0	0	0	0	1	0	0	1	0	0	0	8	3	0	0	0	0	0	0	
w	2	2	1	0	1	0	0	2	0	0	1	0	0	0	0	0	7	0	6	3	3	1	0	0	0	0	
x	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0	
y	0	0	2	0	15	0	1	7	15	0	0	0	2	0	6	1	0	7	36	8	5	0	0	1	0	0	
z	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	2	21	3	0	0	0	0	3	0		



Weighted Min Edit Distance

- Initialization:

$$D(0, 0) = 0$$

$$D(i, 0) = D(i-1, 0) + \text{del}[x(i)]; \quad 1 < i \leq N$$

$$D(0, j) = D(0, j-1) + \text{ins}[y(j)]; \quad 1 < j \leq M$$

- Recurrence Relation:

$$D(i, j) = \min \left\{ \begin{array}{ll} D(i-1, j) & + \text{del}[x(i)] \\ D(i, j-1) & + \text{ins}[y(j)] \\ D(i-1, j-1) & + \text{sub}[x(i), y(j)] \end{array} \right.$$

- Termination:

$D(N, M)$ is distance

Where did the name, dynamic programming, come from?

...The 1950s were not good years for mathematical research. [the] Secretary of Defense ...had a pathological fear and hatred of the word, research...

I decided therefore to use the word, “**programming**”.

I wanted to get across the idea that this was dynamic, this was multistage... I thought, let's ... take a word that has an absolutely precise meaning, namely **dynamic**... it's impossible to use the word, **dynamic**, in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It's impossible.

Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to.”

Minimum Edit Distance

- Minimum Edit Distance in Computational Biology

Sequence Alignment

AGGCTATCACCTGACCTCCAGGCCGATGCC
TAGCTATCACGACC CGCGT CGATTGCCCGAC

-AGG**C**TATC**A**C**C**T**G**AC**C**T**C**CA**G**G**C**CG**A**--TG**C**CC---
T**A****G**-CTATC**A**C--GAC**C**GC--GG**T**CGATT**T**G**C**CC**G**AC

Why sequence alignment?

- Comparing genes or regions from different species
 - to find important regions
 - determine function
 - uncover evolutionary forces
- Assembling fragments to sequence DNA
- Compare individuals to looking for mutations

Alignments in two fields

- In Natural Language Processing
 - We generally talk about **distance** (minimized)
 - And **weights**
- In Computational Biology
 - We generally talk about **similarity** (maximized)
 - And **scores**

The Needleman-Wunsch Algorithm

- Initialization:

$$D(i, 0) = -i * d$$

$$D(0, j) = -j * d$$

- Recurrence Relation:

$$D(i, j) = \min \begin{cases} D(i-1, j) - d \\ D(i, j-1) - d \\ D(i-1, j-1) + s[x(i), y(j)] \end{cases}$$

- Termination:

$D(N, M)$ is distance

The Needleman-Wunsch Matrix



(Note that the origin is at the upper left.)

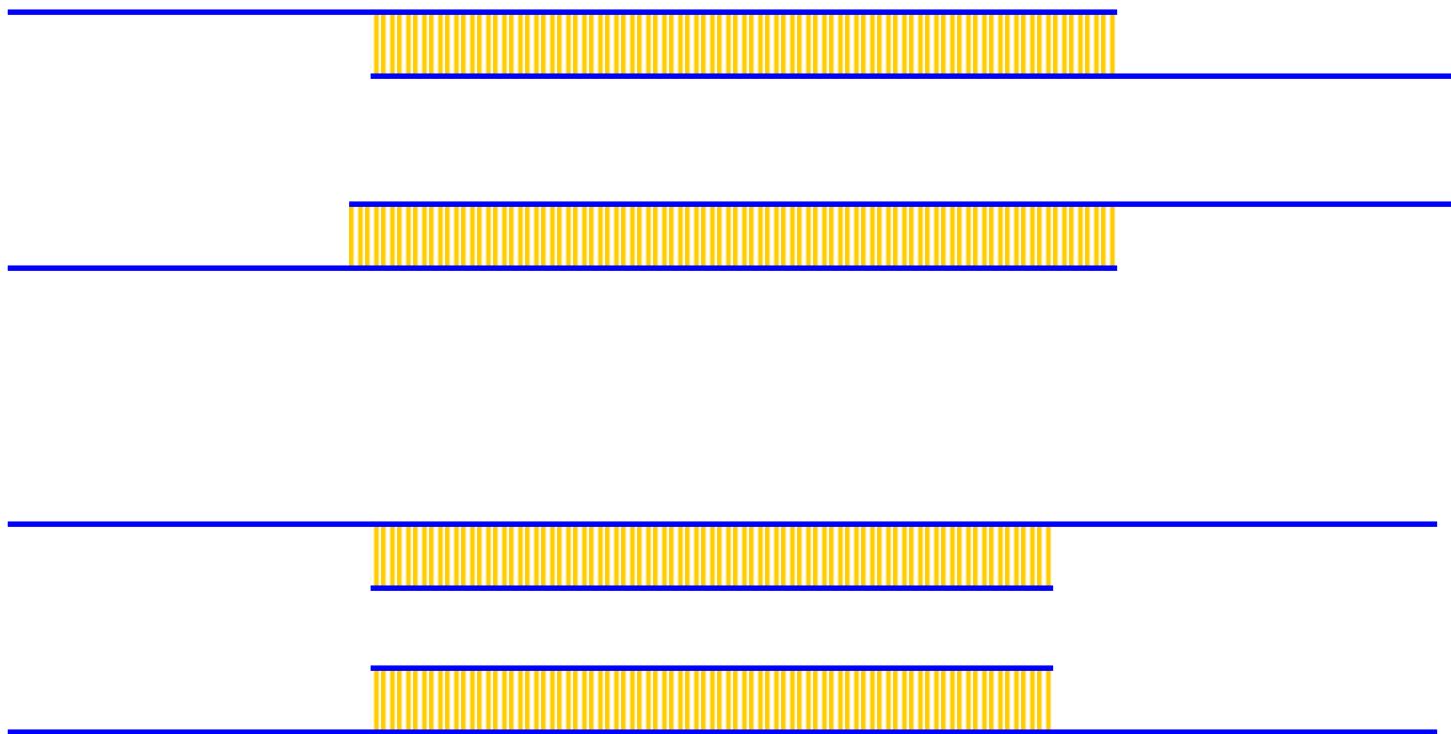
A variant of the basic algorithm:

- Maybe it is OK to have an unlimited # of gaps in the beginning and end:

-----**CTATCAC**CTGACCTCCAGGCCGATGCCCTTCCGGC
GCGAGTTCAT**CTATCAC**--GACCGC--GGTCG-----

- If so, we don't want to penalize gaps at the ends

Different types of overlaps



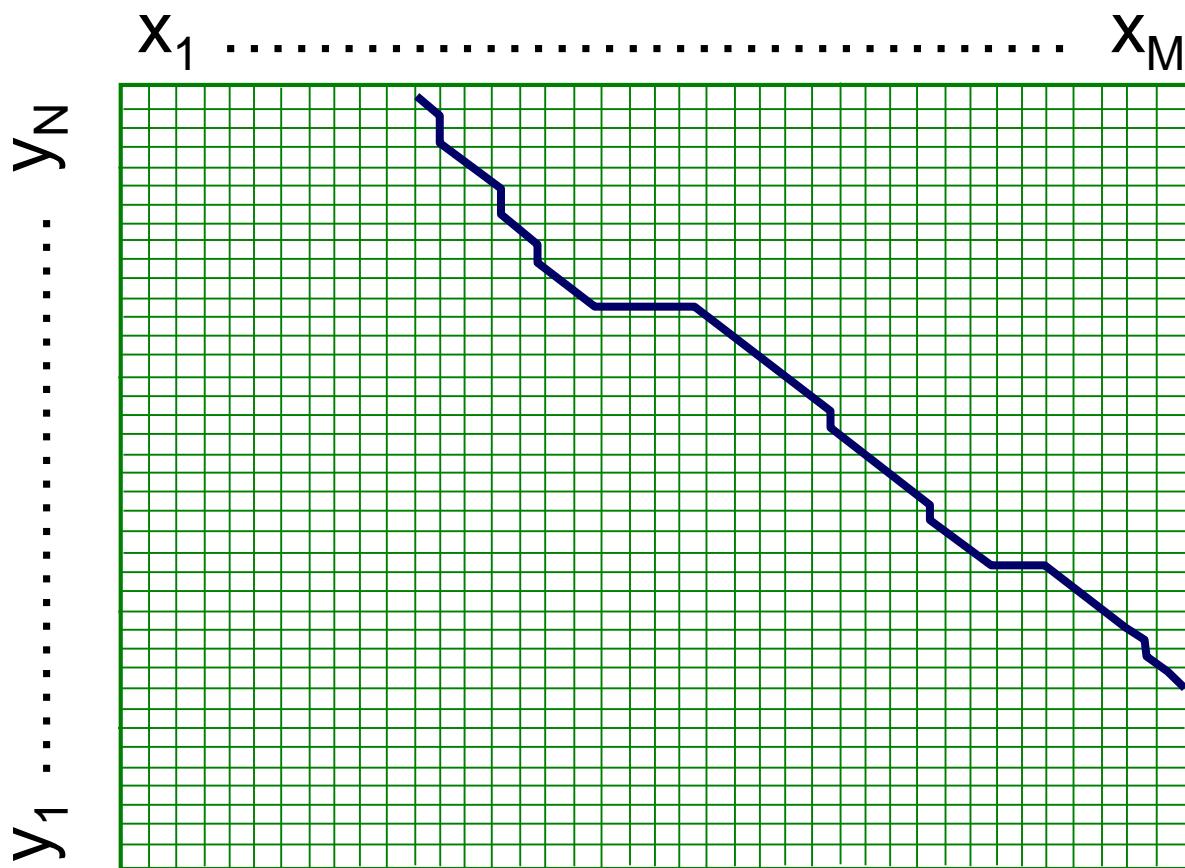
Example:

2 overlapping “reads” from a sequencing project

Example:

Search for a mouse gene within a human chromosome

The Overlap Detection variant



Changes:

1. Initialization

For all $i, j,$

$$F(i, 0) = 0$$

$$F(0, j) = 0$$

2. Termination

$$F_{\text{OPT}} = \max \left\{ \begin{array}{l} \max_i F(i, N) \\ \max_j F(M, j) \end{array} \right\}$$

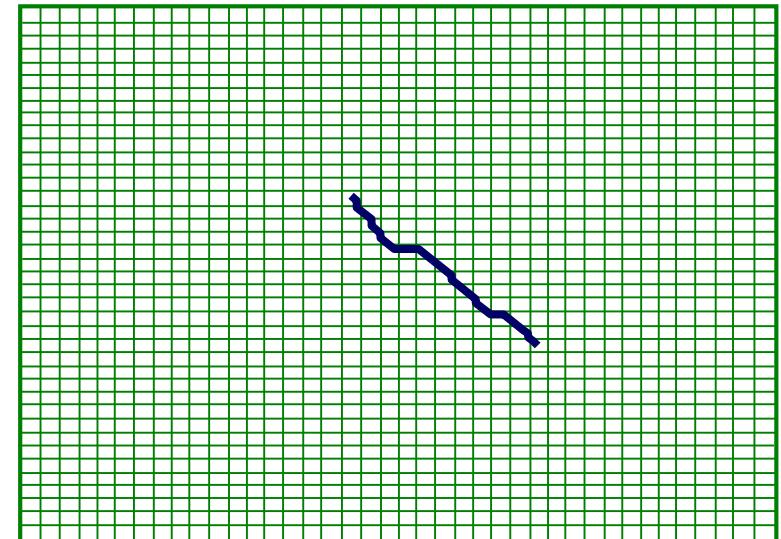
The Local Alignment Problem

Given two strings $x = x_1 \dots x_M,$
 $y = y_1 \dots y_N$

Find substrings x' , y' whose similarity
(optimal global alignment value)
is maximum

$x = \text{aaaaccccccgggggtta}$

$y = \text{ttccccgggaacccaacc}$



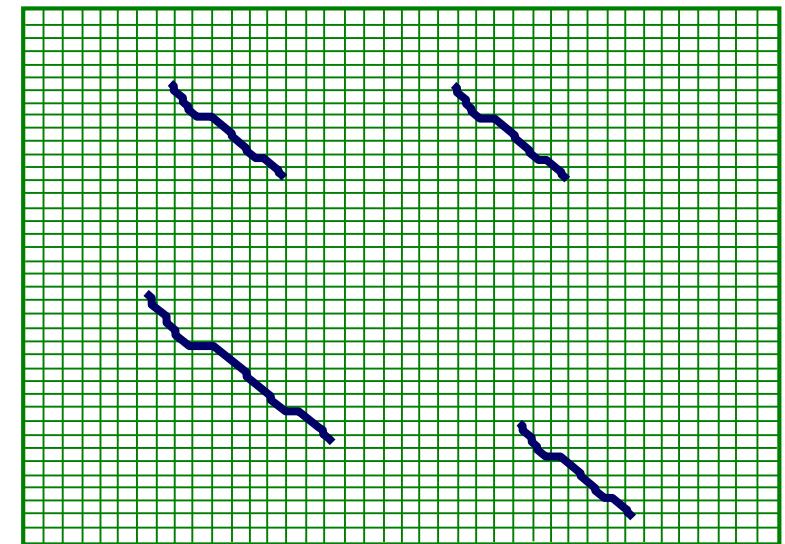
The Smith-Waterman algorithm

Idea: Ignore badly aligning regions

Modifications to Needleman-Wunsch:

Initialization: $F(0, j) = 0$

$$F(i, 0) = 0$$



Iteration:
$$F(i, j) = \max \begin{cases} 0 \\ F(i - 1, j) - d \\ F(i, j - 1) - d \\ F(i - 1, j - 1) + s(x_i, y_j) \end{cases}$$

The Smith-Waterman algorithm

Termination:

1. If we want the **best** local alignment...

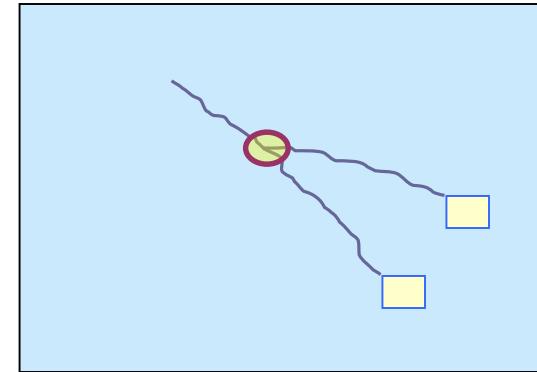
$$F_{OPT} = \max_{i,j} F(i, j)$$

Find F_{OPT} and trace back

2. If we want **all** local alignments **scoring > t**

?? For all i, j find $F(i, j) > t$, and trace back?

Complicated by overlapping local alignments



Local alignment example

X = ATCAT

Y = ATTATC

Let:

$m = 1$ (1 point for match)

$d = 1$ (-1 point for del/ins/sub)

	A	T	T	A	T	C
	0	0	0	0	0	0
A	0					
T	0					
C	0					
A	0					
T	0					

Local alignment example

X = ATCAT

Y = ATTATC

	A	T	T	A	T	C
	0	0	0	0	0	0
A	0	1	0	0	1	0
T	0	0	2	1	0	2
C	0	0	1	1	0	1
A	0	1	0	0	2	1
T	0	0	2	0	1	3

The diagram shows a path of local alignment starting at the top-left cell (0,0) and moving right and down. Arrows point from the top-left cell to the cell (1,1), from (1,1) to (2,2), from (2,2) to (3,3), from (3,3) to (4,4), and finally from (4,4) to the bottom-right cell (5,5). The path highlights the matching segments between the two sequences.

Local alignment example

X = **ATCAT**
Y = **ATTATC**

	A	T	T	A	T	C
	0	0	0	0	0	0
A	0	1	0	0	1	0
T	0	0	2	1	0	2
C	0	0	1	1	0	1
A	0	1	0	0	2	1
T	0	0	2	0	1	3

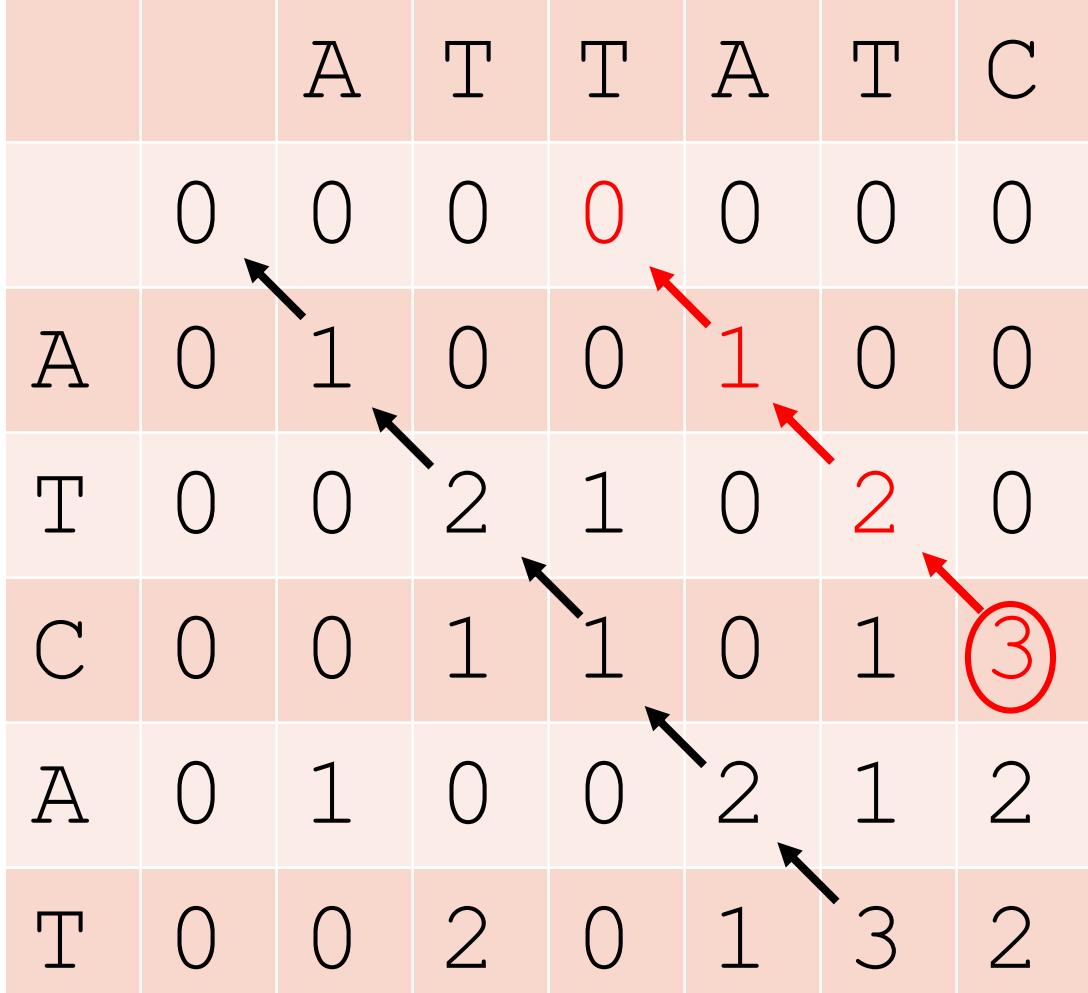
The diagram illustrates a local sequence alignment between two strings, X and Y. The sequences are:
X = ATCAT
Y = ATTATC

The alignment is shown in a grid where rows represent X and columns represent Y. The grid contains numerical scores at each position, with red arrows indicating matches between corresponding symbols in X and Y. Red numbers highlight specific scores: 1 for the first match (A), 2 for the second match (T), and 3 for the third match (A). The final score, 3, is circled in red.

Local alignment example

X = **ATC**AT
Y = ATT**ATC**

	A	T	T	A	T	C
	0 0 0 0 0 0 0					
A	0 1 0 0 1 0 0					
T	0 0 2 1 0 2 0					
C	0 0 1 1 0 1 1					
A	0 1 0 0 2 1 2					
T	0 0 2 0 1 3 2					



The diagram illustrates a local sequence alignment between two strings, X and Y. The sequences are:

X = **ATC**AT

Y = ATT**ATC**

The alignment is shown using a dynamic programming matrix where the columns represent X and the rows represent Y. The matrix cells contain numerical values representing the local alignment score. Arrows point from the sequence Y to the matrix cells corresponding to matches. A red circle highlights the value 3 at position (C,7).

	A	T	T	A	T	C
	0 0 0 0 0 0 0					
A	0 1 0 0 1 0 0					
T	0 0 2 1 0 2 0					
C	0 0 1 1 0 1 1					
A	0 1 0 0 2 1 2					
T	0 0 2 0 1 3 2					

Thanks for Your Attention!

Basic Text Processing

By J. H. Wang

Feb. 23, 2023

Outline

- Words
 - Regular expressions
 - Pre-processing
- Language models
 - N-grams
- Syntax
 - POS tagging
 - Parsing

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'	I have no exquisite reason"
$[^e^]$	Neither e nor ^	Look h <u>e</u> re
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	woodchuck
yours mine	yours mine
a b c	= [abc]
[gG] roundhog [Ww]oodchuck	Woodchuck



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 \wedge $\$$

- \wedge : start of line
- $\$$: end of line

Pattern	Matches
$\wedge [A-Z]$	Palo Alto
$\wedge [^A-Za-z]$	1 <u>"Hello"</u>
$\backslash . \$$	The end <u>.</u>
$. \$$	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 text
- For many hard tasks, we use machine learning classifiers
 - But regular expressions are still used for pre-processing, or as features in the classifiers
 - Can be very useful in capturing generalizations

Basic Text Processing

More Regular Expressions:
Substitutions and ELIZA

Substitutions

- Substitution in Python and UNIX commands:
- `s/regexp1/pattern/`
- e.g.:
- `s/colour/color/`

Capture Groups

- Say we want to put angles around all numbers:
the 35 boxes → *the <35> boxes*
- Use parens () to "capture" a pattern into a numbered register (1, 2, 3...)
- Use \1 to refer to the contents of the register
s / ([0-9] +) /<\1>/

Capture groups: multiple registers

- /the (.*)er they (.*), the \1er we \2/
- Matches
- *the faster they ran, the faster we ran*
- *But not*
- *the faster they ran, the faster we ate*

But suppose we don't want to capture?

Parentheses have a double function: grouping terms, and capturing

Non-capturing groups: add a ?: after paren:

- /(?:some|a few) (people|cats) like some
\1/
- matches
 - some cats like some cats
- **but not**
 - some cats like some some

Lookahead assertions

- `(?= pattern)` is true if pattern matches, but is **zero-width; doesn't advance character pointer**
- `(?! pattern)` true if a pattern does not match
- How to match, at the beginning of a line, any single word that doesn't start with "Volcano":
- `/^ (?!Volcano) [A-Za-z] +/`

Simple Application: ELIZA

- Early NLP system that imitated a Rogerian psychotherapist
 - Joseph Weizenbaum, 1966
- Uses pattern matching to match, e.g.,:
 - “I need X”and translates them into, e.g.
 - “What would it mean to you if you got X?”

Simple Application: ELIZA

Men are all alike.

IN WHAT WAY

They're always bugging us about something or other.

CAN YOU THINK OF A SPECIFIC EXAMPLE

Well, my boyfriend made me come here.

YOUR BOYFRIEND MADE YOU COME HERE

He says I'm depressed much of the time.

I AM SORRY TO HEAR YOU ARE DEPRESSED

How ELIZA works

- s/.* I'M (depressed|sad) .*/I AM SORRY TO HEAR YOU ARE
\1/
- s/.* I AM (depressed|sad) .*/WHY DO YOU THINK YOU ARE
\1/
- s/.* all .*/IN WHAT WAY?/
- s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE?/

Basic Text Processing

Words and Corpora

How many words in a sentence?

- "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 in a sentence?

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 in a corpus?

N = number of tokens

V = vocabulary = set of types, $|V|$ is size of vocabulary

Heaps Law = Herdan's Law = $|V| = kN^\beta$ where often $.67 < \beta < .75$

i.e., vocabulary size grows with $>$ square root of the number of word tokens

	Tokens = N	Types = $ V $
Switchboard phone conversations	2.4 million	20 thousand
Shakespeare	884,000	31 thousand
COCA	440 million	2 million
Google N-grams	1 trillion	13+ million

Corpora

Words don't appear out of nowhere!

A text is produced by

- a specific writer(s),
- at a specific time,
- in a specific variety,
- of a specific language,
- for a specific function.

Corpora vary along dimension like

- **Language:** 7097 languages in the world
- **Variety**, like African American Language varieties
 - AAE Twitter posts might include forms like "*iont*" (*I don't*)
- **Code switching**, e.g., Spanish/English, Hindi/English:
 - S/E: Por primera vez veo a @username actually being hateful! It was beautiful:)
[For the first time I get to see @username actually being hateful! it was beautiful:]
 - H/E: dost tha or ra- hega ... dont wory ... but dherya rakhe
[“he was and will remain a friend ... don’t worry ... but have faith”]
- **Genre:** newswire, fiction, scientific articles, Wikipedia
- **Author Demographics:** writer's age, gender, ethnicity, SES

Corpus datasheets

Gebru et al (2020), Bender and Friedman (2018)

Motivation:

- Why was the corpus collected?
- By whom?
- Who funded it?

Situation: In what situation was the text written?

Collection process: If it is a subsample how was it sampled?
Was there consent? Pre-processing?

- +Annotation process, language variety, demographics, etc.

Basic Text Processing

Word tokenization

Text Normalization

- Every NLP task requires text normalization:
 1. Tokenizing (segmenting) words
 2. Normalizing word formats
 3. Segmenting sentences

Space-based tokenization

- A very simple way to tokenize
 - For languages that use space characters between words
 - Arabic, Cyrillic, Greek, Latin, etc., based writing systems
 - Segment off a token between instances of spaces
- Unix tools for space-based tokenization
 - The "tr" command
 - Inspired by Ken Church's UNIX for Poets
 - Given a text file, output the word tokens and their frequencies

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  
| sort  
| uniq -c
```

Change all non-alpha to newlines

Sort in alphabetical order

Merge and count each type

1945	A
72	AARON
19	ABBESSION
5	ABBOT
...	...

25	Aaron
6	Abate
1	Abates
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

...

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

- Can't just blindly remove punctuation:
 - m.p.h., Ph.D., AT&T, cap'n
 - prices (\$45.55)
 - dates (01/02/06)
 - URLs (<http://www.stanford.edu>)
 - hashtags (#nlproc)
 - email addresses (someone@cs.colorado.edu)
- Clitic: a word that doesn't stand on its own
 - "are" in [we're](#), French "je" in [j'ai](#), "le" in [l'honneur](#)
- When should multiword expressions (MWE) be words?
 - [New York](#), [rock 'n' roll](#)

Tokenization in NLTK

Bird, Loper and Klein (2009), *Natural Language Processing with Python*. O'Reilly

```
>>> text = 'That U.S.A. poster-print costs $12.40...'
>>> pattern = r'''(?x)      # set flag to allow verbose regexps
...     ([A-Z]\. )+        # abbreviations, e.g. U.S.A.
...     | \w+(-\w+)*        # words with optional internal hyphens
...     | \$?\d+(\.\d+)?%?  # currency and percentages, e.g. $12.40, 82%
...     | \.\\.\\.           # ellipsis
...     | [][.,;'"'?():-_`] # these are separate tokens; includes ], [
...     ''
...
>>> nltk.regexp_tokenize(text, pattern)
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']
```

Tokenization in languages without spaces

Many languages (like Chinese, Japanese, Thai) don't use spaces to separate words!

How do we decide where the token boundaries should be?

Word tokenization in Chinese

- Chinese words are composed of characters called "hanzi" (or sometimes just "zi")
- Each one represents a meaning unit called a morpheme
- Each word has on average 2.4 of them
- But deciding what counts as a word is complex and not agreed upon

How to do word tokenization in Chinese?

- 姚明進入總決賽 “Yao Ming reaches the finals”

How to do word tokenization in Chinese?

- 姚明進入總決賽 “Yao Ming reaches the finals”

- 3 words?

- 姚明 進入 總決賽

- YaoMing reaches finals

How to do word tokenization in Chinese?

- 姚明進入總決賽 “Yao Ming reaches the finals”

- 3 words?

- 姚明 進入 總決賽

- YaoMing reaches finals

- 5 words?

- 姚 明 進入 總 決賽

- Yao Ming reaches overall finals

How to do word tokenization in Chinese?

- 姚明進入總決賽 “Yao Ming reaches the finals”

- 3 words?

- 姚明 進入 總決賽

- YaoMing reaches finals

- 5 words?

- 姚 明 進 入 總 決 賽

- Yao Ming reaches overall finals

- 7 characters? (don't use words at all):

- 姚 明 進 入 總 決 賽

- Yao Ming enter enter overall decision game

Word tokenization / segmentation

So in Chinese it's common to just treat each character (*zi*) as a token

- So the **segmentation** step is very simple

In other languages (like Thai and Japanese), more complex word segmentation is required

- The standard algorithms are neural sequence models trained by supervised machine learning

Basic Text Processing

Byte Pair Encoding

Another option for text tokenization

Instead of

- white-space segmentation
- single-character segmentation

Use the data to tell us how to tokenize

Subword tokenization (because tokens can be parts of words as well as whole words)

Subword tokenization

- Three common algorithms:
 - **Byte-Pair Encoding (BPE)** (Sennrich et al., 2016)
 - **Unigram language modeling tokenization** (Kudo, 2018)
 - **WordPiece** (Schuster and Nakajima, 2012)
- All have 2 parts:
 - A token **learner** that takes a raw training corpus and induces a vocabulary (a set of tokens)
 - A token **segmenter** that takes a raw test sentence and tokenizes it according to that vocabulary

Byte Pair Encoding (BPE) token learner

Let vocabulary be the set of all individual characters

$$= \{A, B, C, D, \dots, a, b, c, d, \dots\}$$

- Repeat:
 - Choose the two symbols that are most frequently adjacent in the training corpus (say 'A', 'B')
 - Add a new merged symbol 'AB' to the vocabulary
 - Replace every adjacent 'A' 'B' in the corpus with 'AB'
- Until k merges have been done

BPE token learner algorithm

```
function BYTE-PAIR ENCODING(strings  $C$ , number of merges  $k$ ) returns vocab  $V$ 
     $V \leftarrow$  all unique characters in  $C$            $\#$  initial set of tokens is characters
    for  $i = 1$  to  $k$  do                       $\#$  merge tokens til  $k$  times
         $t_L, t_R \leftarrow$  Most frequent pair of adjacent tokens in  $C$ 
         $t_{NEW} \leftarrow t_L + t_R$                    $\#$  make new token by concatenating
         $V \leftarrow V + t_{NEW}$                        $\#$  update the vocabulary
        Replace each occurrence of  $t_L, t_R$  in  $C$  with  $t_{NEW}$        $\#$  and update the corpus
    return  $V$ 
```

Byte Pair Encoding (BPE) Addendum

Most subword algorithms are run inside space-separated tokens

So we commonly first add a special end-of-word symbol '_' before space in training corpus

Next, separate into letters

BPE token learner

Original (very fascinating 😊) corpus:

low low low low lowest lowest newer newer newer
wider wider new new

Add end-of-word tokens, resulting in this vocabulary:

vocabulary

_, d, e, i, l, n, o, r, s, t, w

BPE token learner

corpus

5 low _
2 lowest _
6 newer _
3 wider _
2 new _

vocabulary

_, d, e, i, l, n, o, r, s, t, w

Merge er to er

corpus

5 low _
2 lowest _
6 newer _
3 wider _
2 new _

vocabulary

_, d, e, i, l, n, o, r, s, t, w, er

BPE

corpus

5 low _
2 lowest _
6 newer _
3 wider _
2 new _

vocabulary

_, d, e, i, l, n, o, r, s, t, w, er

Merge er _ to er_

corpus

5 low _
2 lowest _
6 newer _
3 wider _
2 new _

vocabulary

, d, e, i, l, n, o, r, s, t, w, er, er

BPE

corpus

5 low _
2 low est _
6 new er_
3 wid er_
2 ne w _

vocabulary

, d, e, i, l, n, o, r, s, t, w, er, er

Merge **n e** to **ne**

corpus

5 low _
2 low est _
6 new er_
3 wid er_
2 ne w _

vocabulary

, d, e, i, l, n, o, r, s, t, w, er, er, ne

BPE

The next merges are:

Merge	Current Vocabulary
(ne, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new
(l, o)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo
(lo, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low
(new, er_)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_
(low, _)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_, low_

BPE token segmenter algorithm

On the test data, run each merge learned from the training data:

- Greedily
- In the order we learned them
- (test frequencies don't play a role)

So: merge every **e r** to **er**, then merge **er _** to **er_**, etc.

- Result:
 - Test set "n e w e r _" would be tokenized as a full word
 - Test set "l o w e r _" would be two tokens: "low er_"

Properties of BPE tokens

Usually include frequent words

And frequent subwords

- Which are often morphemes like *-est* or *-er*

A **morpheme** is the smallest meaning-bearing unit of a language

- *unlikeliest* has 3 morphemes *un-*, *likely*, and *-est*

Basic Text Processing

Word Normalization and Other Issues

Word Normalization

- Putting words/tokens in a standard format
 - U.S.A. or USA
 - uhhuh or uh-huh
 - Fed or fed
 - am, is, be, are

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

Represent all words as their lemma, their shared root
= dictionary headword form:

- *am, are, is* → *be*
- *car, cars, car's, cars'* → *car*

- Spanish **quiero** ('I want'), **quieres** ('you want')

→ **querer** 'want'

- *He is reading detective stories*

→ *He be read detective story*

Lemmatization is done by Morphological Parsing

- Morphemes:

- The small meaningful units that make up words
- **Stems**: The core meaning-bearing units
- **Affixes**: Parts that adhere to stems, often with grammatical functions

- Morphological Parsers:

- Parse *cats* into two morphemes *cat* and *s*
- Parse Spanish *amaren* ('if in the future they would love') into morpheme *amar* 'to love', and the morphological features *3PL* and *future subjunctive*

Stemming

- Reduce terms to stems, chopping off affixes crudely

This was not the map we found in Billy Bones's chest, but an accurate copy, complete in all things-names and heights and soundings-with the single exception of the red crosses and the written notes.



Thi wa not the map we found in Billi Bone
s chest but an accur copi complet in all
thing name and height and sound with the
singl except of the red cross and the
written note
.

Porter Stemmer

- Based on a series of rewrite rules run in series
 - A cascade, in which output of each pass fed to next pass
- Some sample rules:

ATIONAL → ATE (e.g., relational → relate)

ING → ϵ if stem contains vowel (e.g., motoring → motor)

SSES → SS (e.g., grasses → grass)

Dealing with complex morphology is necessary for many languages

- e.g., the Turkish word:
- **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'

Sentence Segmentation

!, ? mostly unambiguous but **period** “.” is very ambiguous

- Sentence boundary
- Abbreviations like Inc. or Dr.
- Numbers like .02% or 4.3

Common algorithm: Tokenize first: use rules or ML to classify a period as either (a) part of the word or (b) a sentence-boundary.

- An abbreviation dictionary can help

Sentence segmentation can then often be done by rules based on this tokenization.

Thanks for Your Attention!



Chap. 3: Language Modeling



Outline

- Introduction to n-grams
- Estimating n-gram probabilities
- Evaluation and perplexity
- Generalization and zeros
- Smoothing: Add-one (Laplace) smoothing
- Interpolation, back-off, Web-scale LMs



Language Modeling

Introduction to N-grams

[Modified from Dan Jurafsky's
slides for SLP3]



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
 - 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.!!

Why?



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

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

$P(\text{"its water is so transparent"}) =$

$$\begin{aligned} & P(\text{its}) \times P(\text{water}|\text{its}) \times P(\text{is}|\text{its water}) \\ & \times P(\text{so}|\text{its water is}) \times P(\text{transparent}|\text{its water is so}) \end{aligned}$$



How to estimate these probabilities

- Could we just count and divide?

$$P(\text{the} | \text{its water is so transparent that}) =$$
$$\frac{\text{Count}(\text{its water is so transparent that the})}{\text{Count}(\text{its water is so transparent that})}$$

- No! Too many possible sentences!
- We'll never see enough data for estimating these



Markov Assumption

- Simplifying assumption:



Andrei Markov

$$P(\text{the} \mid \text{its water is so transparent that}) \gg P(\text{the} \mid \text{that})$$

- Or maybe

$$P(\text{the} \mid \text{its water is so transparent that}) \gg P(\text{the} \mid \text{transparent that})$$



Markov Assumption

$$P(w_1 w_2 \dots w_n) \approx \prod_i P(w_i \mid w_{i-k} \dots w_{i-1})$$

- In other words, we approximate each component in the product

$$P(w_i \mid w_1 w_2 \dots w_{i-1}) \approx P(w_i \mid w_{i-k} \dots w_{i-1})$$



Simplest case: Unigram model

$$P(w_1 w_2 \dots w_n) \approx \prod_i P(w_i)$$

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:

$$P(w_i | w_1 w_2 \dots w_{i-1}) \approx P(w_i | w_{i-1})$$

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(s) 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

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 >

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

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

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

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

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

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



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

$P(< s > \text{ I want english food } </ s >) =$

$P(I | < s >)$

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

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

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

$\times P(</ s > | \text{food})$

$= .000031$



What kinds of knowledge?

- $P(\text{english} \mid \text{want}) = .0011$
- $P(\text{chinese} \mid \text{want}) = .0065$
- $P(\text{to} \mid \text{want}) = .66$
- $P(\text{eat} \mid \text{to}) = .28$
- $P(\text{food} \mid \text{to}) = 0$
- $P(\text{want} \mid \text{spend}) = 0$
- $P(\text{i} \mid \langle s \rangle) = .25$



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.googlecode.com/>



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.



Training on the test set

- We can't allow test sentences into the training set
- We will assign it an artificially high probability when we set it in the test set
- “Training on the test set”
- Bad science!
- And violates the honor code



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

{ mushrooms 0.1
pepperoni 0.1
anchovies 0.01
....
fried rice 0.0001
....
and 1e-100



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:

$$\text{Perp} = (\frac{1}{4} * \frac{1}{4} * \frac{1}{4} * \frac{1}{4} * \frac{1}{4} * \dots * \frac{1}{120K} * \frac{1}{120K} * \dots)^{-1/120K}$$

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):

$$\text{Perplexity} = ((\frac{1}{4} * \frac{1}{4} * \frac{1}{4} * \frac{1}{4})^{1/4})^{-1} = 52.6$$



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

Generalization and
zeros



The Shannon Visualization Method

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

$\langle s \rangle$ I
I want
want to
to eat
eat Chinese
Chinese food
food $\langle /s \rangle$

I want to eat Chinese food



Approximating Shakespeare

- | | |
|------------------|---|
| 1
gram | –To him swallowed confess hear both. Which. Of save on trail for are ay device and
rote life have |
| | –Hill he late speaks; or! a more to leg less first you enter |
| 2
gram | –Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live
king. Follow. |
| | –What means, sir. I confess she? then all sorts, he is trim, captain. |
| 3
gram | –Fly, and will rid me these news of price. Therefore the sadness of parting, as they say,
'tis done. |
| | –This shall forbid it should be branded, if renown made it empty. |
| 4
gram | –King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A
great banquet serv'd in; |
| | –It cannot be but so. |



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 author of 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

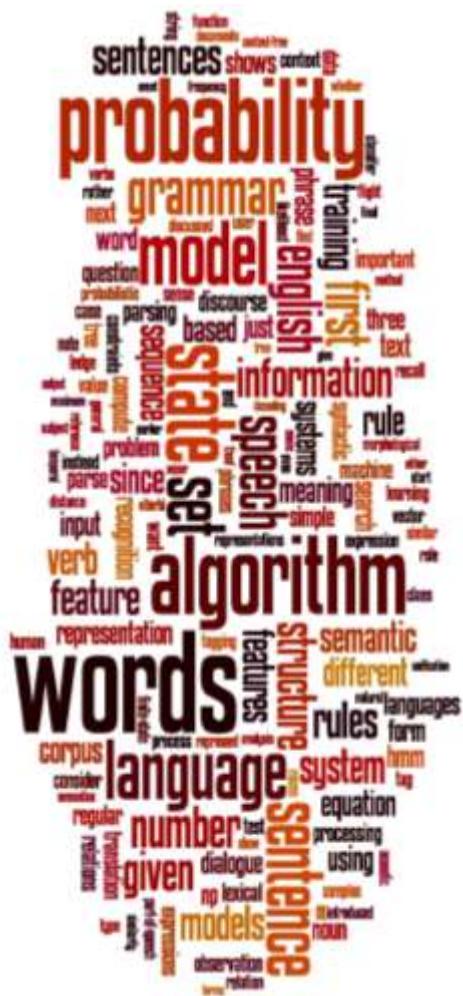
- Training set:
 - ... denied the allegations
 - ... denied the reports
 - ... denied the claims
 - ... denied the request
- Test set
 - ... denied the offer
 - ... denied the loan

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



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

Smoothing: Add-one
(Laplace) smoothing



The intuition of smoothing (from Dan Klein)

- When we have sparse statistics:

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

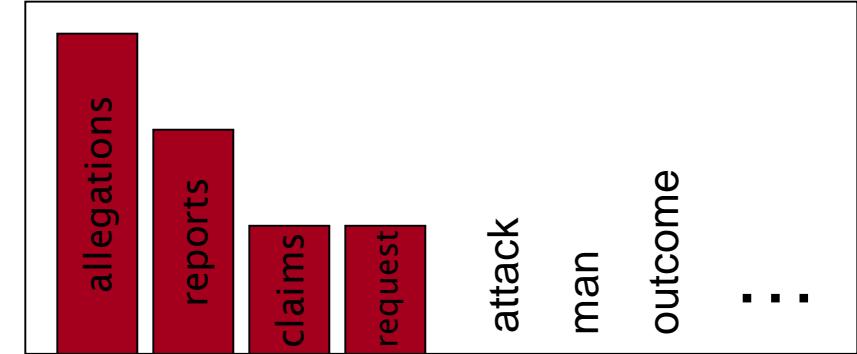
3 allegations

2 reports

1 claims

1 request

7 total



- Steal probability mass to generalize better

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

2.5 allegations

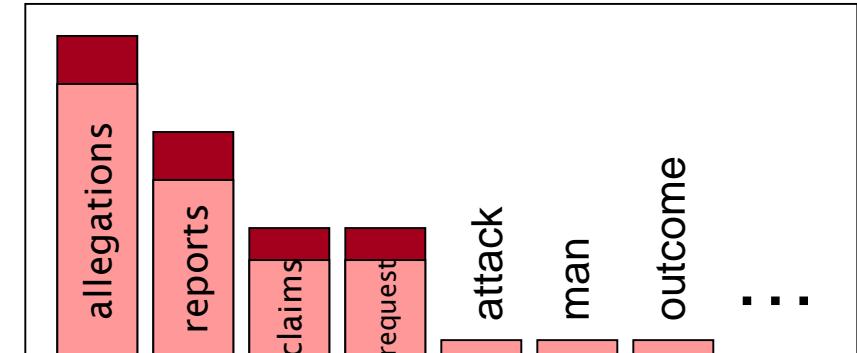
1.5 reports

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!

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

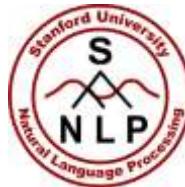
- MLE estimate:
- Add-1 estimate:

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



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

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

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

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



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)!

$$P_{\text{AbsoluteDiscounting}}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + / \overset{\text{discounted bigram}}{(w_{i-1})} \underset{\text{unigram}}{\overset{\swarrow}{P(w)}}$$

- (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 Kongglasses*?
 - “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

$$P_{\text{CONTINUATION}}(w) \propto \frac{1}{|\{w_{i-1} : c(w_{i-1}, w) > 0\}|}$$



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

$$|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}|$$

$$P_{CONTINUATION}(w) = \frac{|\{w_{i-1} : c(w_{i-1}, w) > 0\}|}{|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}|}$$



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:

$$P_{CONTINUATION}(w) = \frac{|\{w_{i-1} : c(w_{i-1}, w) > 0\}|}{\sum |\{w'_{i-1} : c(w'_{i-1}, w') > 0\}|}$$

- 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

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} \left| \{w : c(w_{i-1}, w) > 0\} \right|$$

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

$$P_{KN}(w_i | w_{i-n+1}^{i-1}) = \frac{\max(c_{KN}(w_{i-n+1}^i) - d, 0)}{c_{KN}(w_{i-n+1}^{i-1})} + / (w_{i-n+1}^{i-1}) P_{KN}(w_i | w_{i-n+2}^{i-1})$$

$$c_{KN}(\cdot) = \begin{cases} \uparrow & \text{count}(\cdot) \quad \text{for the highest order} \\ \downarrow & \text{continuationcount}(\cdot) \quad \text{for lower order} \end{cases}$$

Continuation count = Number of unique single word contexts for •
 73



Thanks for Your Attention!

Chap. 4: Naïve Bayes and Sentiment Classification

Outline

- The task of text classification
- The Naïve Bayes Classifier
- Naïve Bayes: Learning
- Sentiment and Binary Naïve Bayes
- More on Sentiment Classification
- Naïve Bayes: Relationship to Language Modeling
- Precision, Recall, and F-Measure
- Evaluation with more than two classes
- Statistical Significance Testing
- The Paired Bootstrap Test
- Avoiding harms in classification

Is this spam?

Subject: Important notice!

From: Stanford University <newsforum@stanford.edu>

Date: October 28, 2011 12:34:16 PM PDT

To: undisclosed-recipients:;

Greats News!

You can now access the latest news by using the link below to login to Stanford University News Forum.

<http://www.123contactform.com/contact-form-StanfordNew1-236335.html>

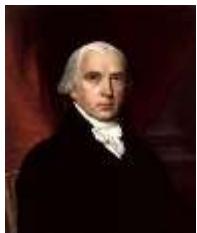
Click on the above link to login for more information about this new exciting forum. You can also copy the above link to your browser bar and login for more information about the new services.

© Stanford University. All Rights Reserved.

Who wrote which Federalist papers?



- 1787-8: anonymous essays try to convince New York to ratify U.S Constitution: Jay, Madison, Hamilton.
- Authorship of 12 of the letters in dispute
- 1963: solved by Mosteller and Wallace using Bayesian methods



James Madison



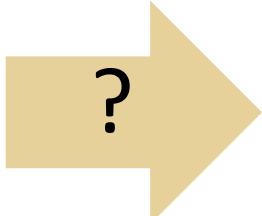
Alexander Hamilton

Male or female author?

1. By 1925 present-day Vietnam was divided into three parts under French colonial rule. The southern region embracing Saigon and the Mekong delta was the colony of Cochinchina; the central area with its imperial capital at Hue was the protectorate of Annam...
2. Clara never failed to be astonished by the extraordinary felicity of her own name. She found it hard to trust herself to the mercy of fate, which had managed over the years to convert her greatest shame into one of her greatest assets...

What is the subject of this article?

MEDLINE Article



MeSH Subject Category Hierarchy

- Antagonists and Inhibitors
- Blood Supply
- Chemistry
- Drug Therapy
- Embryology
- Epidemiology
- ...

Positive or negative movie review?

- + *...zany characters and richly applied satire, and some great plot twists*
- *It was pathetic. The worst part about it was the boxing scenes...*
- + *...awesome caramel sauce and sweet toasty almonds. I love this place!*
- *...awful pizza and ridiculously overpriced...*

Positive or negative movie review?

- + ...zany characters and **richly** applied satire, and some **great** plot twists
- It was **pathetic**. The **worst** part about it was the boxing scenes...
- + ...**awesome** caramel sauce and sweet toasty almonds. I **love** this place!
- ...**awful** pizza and **ridiculously** overpriced...

Why sentiment analysis?

- *Movie*: is this review positive or negative?
- *Products*: what do people think about the new iPhone?
- *Public sentiment*: how is consumer confidence?
- *Politics*: what do people think about this candidate or issue?
- *Prediction*: predict election outcomes or market trends from sentiment

Scherer Typology of Affective States

- **Emotion:** brief organically synchronized ... evaluation of a major event
 - *angry, sad, joyful, fearful, ashamed, proud, elated*
- **Mood:** diffuse non-caused low-intensity long-duration change in subjective feeling
 - *cheerful, gloomy, irritable, listless, depressed, buoyant*
- **Interpersonal stances:** affective stance toward another person in a specific interaction
 - *friendly, flirtatious, distant, cold, warm, supportive, contemptuous*
- **Attitudes:** enduring, affectively colored beliefs, dispositions towards objects or persons
 - *liking, loving, hating, valuing, desiring*
- **Personality traits:** stable personality dispositions and typical behavior tendencies
 - *nervous, anxious, reckless, morose, hostile, jealous*

Scherer Typology of Affective States

- **Emotion:** brief organically synchronized ... evaluation of a major event
 - *angry, sad, joyful, fearful, ashamed, proud, elated*
- **Mood:** diffuse non-caused low-intensity long-duration change in subjective feeling
 - *cheerful, gloomy, irritable, listless, depressed, buoyant*
- **Interpersonal stances:** affective stance toward another person in a specific interaction
 - *friendly, flirtatious, distant, cold, warm, supportive, contemptuous*
- **Attitudes: enduring, affectively colored beliefs, dispositions towards objects or persons**
 - *liking, loving, hating, valuing, desiring*
- **Personality traits:** stable personality dispositions and typical behavior tendencies
 - *nervous, anxious, reckless, morose, hostile, jealous*

Basic Sentiment Classification

- Sentiment analysis is the detection of **attitudes**
- Simple task we focus on in this chapter
 - Is the attitude of this text positive or negative?
- We return to affect classification in later chapters

Summary: Text Classification

- Sentiment analysis
- Spam detection
- Authorship identification
- Language Identification
- Assigning subject categories, topics, or genres
- ...

Text Classification: definition

- *Input:*
 - a document d
 - a fixed set of classes $C = \{c_1, c_2, \dots, c_J\}$
- *Output:* a predicted class $c \in C$

Classification Methods:

Hand-coded rules

- Rules based on combinations of words or other features
 - spam: black-list-address OR (“dollars” AND “you have been selected”)
- Accuracy can be high
 - If rules carefully refined by expert
- But building and maintaining these rules is expensive

Classification Methods: Supervised Machine Learning

- *Input:*
 - a document d
 - a fixed set of classes $C = \{c_1, c_2, \dots, c_J\}$
 - A training set of m hand-labeled documents $(d_1, c_1), \dots, (d_m, c_m)$
- *Output:*
 - a learned classifier $y: d \rightarrow c$

Classification Methods: Supervised Machine Learning

- Any kind of classifier
 - Naïve Bayes
 - Logistic regression
 - Support-vector machines
 - k-Nearest Neighbors
 - ...

Text Classification and Naïve Bayes

Naïve Bayes Classifier

Naïve Bayes Intuition

- Simple (“naïve”) classification method based on Bayes rule
- Relies on very simple representation of document
 - Bag of words

The Bag of Words Representation

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



The bag of words representation

Y(

seen	2
sweet	1
whimsical	1
recommend	1
happy	1
...	...

) = C



Bayes' Rule Applied to Documents and Classes

- For a document d and a class c

$$P(c | d) = \frac{P(d | c)P(c)}{P(d)}$$

Naïve Bayes Classifier (I)

$$c_{MAP} = \operatorname{argmax}_{c \in C} P(c | d)$$

MAP is “maximum a posteriori” = most likely class

$$= \operatorname{argmax}_{c \in C} \frac{P(d | c)P(c)}{P(d)}$$

Bayes Rule

$$= \operatorname{argmax}_{c \in C} P(d | c)P(c)$$

Dropping the denominator

Naïve Bayes Classifier (II)

$$c_{MAP} = \operatorname{argmax}_{c \in C} P(d | c)P(c)$$

$$= \operatorname{argmax}_{c \in C} P(x_1, x_2, \dots, x_n | c)P(c)$$

Document d
represented as
features
x₁...x_n

Naïve Bayes Classifier (IV)

$$c_{MAP} = \operatorname{argmax}_{c \in C} P(x_1, x_2, \dots, x_n | c)P(c)$$

$O(|X|^n \cdot |C|)$ parameters

Could only be estimated if a very, very large number of training examples was available.

How often does this class occur?

We can just count the relative frequencies in a corpus

Multinomial Naïve Bayes Independence Assumptions

$$P(x_1, x_2, \dots, x_n \mid c)$$

- **Bag of Words assumption:** Assume position doesn't matter
- **Conditional Independence:** Assume the feature probabilities $P(x_i \mid c_j)$ are independent given the class c .

$$P(x_1, \dots, x_n \mid c) = P(x_1 \mid c) \bullet P(x_2 \mid c) \bullet P(x_3 \mid c) \bullet \dots \bullet P(x_n \mid c)$$

Multinomial Naïve Bayes Classifier

$$c_{MAP} = \operatorname{argmax}_{c \in C} P(x_1, x_2, \dots, x_n \mid c)P(c)$$

$$c_{NB} = \operatorname{argmax}_{c \in C} P(c_j) \tilde{\bigcap}_{x \in X} P(x \mid c)$$

Applying Multinomial Naive Bayes Classifiers to Text Classification

positions \leftarrow all word positions in test document

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

Problems with multiplying lots of probs

- There's a problem with this:

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

- Multiplying lots of probabilities can result in floating-point underflow!
 $.0006 * .0007 * .0009 * .01 * .5 * .000008....$
- Idea: Use logs, because $\log(ab) = \log(a) + \log(b)$
We'll sum logs of probabilities instead of multiplying probabilities!

We actually do everything in log space

Instead of this: $c_{NB} = \operatorname{argmax}_{c_j \in C} P(c_j) \bigcirc_{i \in positions} P(x_i | c_j)$

This:

$$c_{NB} = \operatorname{argmax}_{c_j \in C} \left[\log P(c_j) + \sum_{i \in positions} \log P(x_i | c_j) \right]$$

Notes:

1) Taking log doesn't change the ranking of classes!

The class with highest probability also has highest log probability!

2) It's a linear model:

Just a max of a sum of weights: a **linear** function of the inputs

So naive bayes is a **linear classifier**

Text Classification and Naïve Bayes

Naïve Bayes: Learning

Learning the Multinomial Naïve Bayes Model

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

$$\hat{P}(c_j) = \frac{\text{doccount}(C = c_j)}{N_{\text{doc}}}$$

$$\hat{P}(w_i | c_j) = \frac{\text{count}(w_i, c_j)}{\sum_{w \in V} \text{count}(w, c_j)}$$

Parameter estimation

$$\hat{P}(w_i | c_j) = \frac{\underset{w \in V}{\text{count}}(w_i, c_j)}{\underset{w \in V}{\text{count}}(w, c_j)}$$

fraction of times word w_i appears
among all words in documents of topic c_j

- Create mega-document for topic j by concatenating all docs in this topic
 - Use frequency of w in mega-document

Problem with Maximum Likelihood

- What if we have seen no training documents with the word ***fantastic*** and classified in the topic **positive (*thumbs-up*)**?

$$\hat{P}(\text{"fantastic"} \mid \text{positive}) = \frac{\underset{w \in V}{\text{count}}(\text{"fantastic"}, \text{positive})}{\sum_{w \in V} \text{count}(w, \text{positive})} = 0$$

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

$$c_{MAP} = \operatorname{argmax}_c \hat{P}(c) \tilde{\bigcap}_i \hat{P}(x_i \mid c)$$

Laplace (add-1) smoothing for Naïve Bayes

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

Multinomial Naïve Bayes: Learning

- From training corpus, extract *Vocabulary*
 - Calculate $P(c_j)$ terms
 - For each c_j in C do
 $docs_j \leftarrow$ all docs with class = c_j
 - Calculate $P(w_k | c_j)$ terms
 - $Text_j \leftarrow$ single doc containing all $docs_j$
 - For each word w_k in *Vocabulary*
 $n_k \leftarrow$ # of occurrences of w_k in $Text_j$
- $$P(c_j) \leftarrow \frac{|docs_j|}{|\text{total \# documents}|}$$
- $$P(w_k | c_j) \leftarrow \frac{n_k + \alpha}{n + \alpha |\text{Vocabulary}|}$$

Unknown words

- What about unknown words
 - that appear in our test data
 - but not in our training data or vocabulary?
- We **ignore** them
 - Remove them from the test document!
 - Pretend they weren't there!
 - Don't include any probability for them at all!
- Why don't we build an unknown word model?
 - It doesn't help: knowing which class has more unknown words is not generally helpful!

Stop words

- Some systems ignore stop words
 - **Stop words:** very frequent words like *the* and *a*.
 - Sort the vocabulary by word frequency in training set
 - Call the top 10 or 50 words the **stopword list**.
 - Remove all stop words from both training and test sets
 - As if they were never there!
- But removing stop words doesn't usually help
 - So in practice most NB algorithms use **all** words and **don't** use stopword lists

Text Classification and Naïve Bayes

**Sentiment and Binary
Naïve Bayes**

Let's do a worked sentiment example!

	Cat	Documents
Training	- just plain boring - entirely predictable and lacks energy - no surprises and very few laughs + very powerful + the most fun film of the summer	
Test	?	predictable with no fun

A worked sentiment example with add-1 smoothing

Cat	Documents
Training	- just plain boring
	- entirely predictable and lacks energy
	- no surprises and very few laughs
	+ very powerful
	+ the most fun film of the summer
Test	? predictable with no fun

3. Likelihoods from training:

$$p(w_i|c) = \frac{\text{count}(w_i, c) + 1}{(\sum_{w \in V} \text{count}(w, c)) + |V|}$$

$$P(\text{"predictable"}|-) = \frac{1+1}{14+20} \quad P(\text{"predictable"}|+) = \frac{0+1}{9+20}$$

$$P(\text{"no"}|-) = \frac{1+1}{14+20}$$

$$P(\text{"no"}|+) = \frac{0+1}{9+20}$$

$$P(\text{"fun"}|-) = \frac{0+1}{14+20}$$

$$P(\text{"fun"}|+) = \frac{1+1}{9+20}$$

1. Prior from training:

$$\hat{P}(c_j) = \frac{N_{c_j}}{N_{\text{total}}} \quad \begin{aligned} P(-) &= 3/5 \\ P(+) &= 2/5 \end{aligned}$$

2. Drop "with"

4. Scoring the test set:

$$P(-)P(S|-) = \frac{3}{5} \times \frac{2 \times 2 \times 1}{34^3} = 6.1 \times 10^{-5}$$

$$P(+)P(S|+) = \frac{2}{5} \times \frac{1 \times 1 \times 2}{29^3} = 3.2 \times 10^{-5}$$

Optimizing for sentiment analysis

For tasks like sentiment, word **occurrence** seems to be more important than word **frequency**.

- The occurrence of the word *fantastic* tells us a lot
- The fact that it occurs 5 times may not tell us much more.

Binary multinomial naive bayes, or binary NB

- Clip our word counts at 1
- Note: this is different than Bernoulli naive bayes; see the textbook at the end of the chapter.

Binary Multinomial Naïve Bayes: Learning

- From training corpus, extract *Vocabulary*
- Calculate $P(c_j)$ terms
 - For each c_j in C do
$$docs_j \leftarrow \text{all docs with class } = c_j$$
$$P(c_j) \leftarrow \frac{|docs_j|}{|\text{total # documents}|}$$
 - Calculate $P(w_k | c_j)$ terms
 - Remove duplicates in $Text_j$ containing all $docs_j$
 - For each word w_k in $Vocabulary$
 - Retain only a single instance of w_k in $Text_j$
$$n_k \leftarrow \# \text{ of occurrences of } w_k \text{ in } Text_j$$

Binary Multinomial Naive Bayes on a test document d

- First remove all duplicate words from d
- Then compute NB using the same equation:

$$c_{NB} = \operatorname{argmax}_{c_j \in C} P(c_j) \tilde{\bigcirc}_{i \in positions} P(w_i | c_j)$$

Binary multinomial naive Baves

Four original documents:

- it was pathetic the worst part was the boxing scenes
 - no plot twists or great scenes
 - + and satire and great plot twists
 - + great scenes great film
-

Binary multinomial naive Bayes

Four original documents:

- it was pathetic the worst part was the boxing scenes
 - no plot twists or great scenes
 - + and satire and great plot twists
 - + great scenes great film
-

	NB Counts	
	+	-
and	2	0
boxing	0	1
film	1	0
great	3	1
it	0	1
no	0	1
or	0	1
part	0	1
pathetic	0	1
plot	1	1
satire	1	0
scenes	1	2
the	0	2
twists	1	1
was	0	2
worst	0	1

Binary multinomial naive Bayes

Four original documents:

- it was pathetic the worst part was the boxing scenes
- no plot twists or great scenes
- + and satire and great plot twists
- + great scenes great film

After per-document binarization:

- it was pathetic the worst part boxing scenes
- no plot twists or great scenes
- + and satire great plot twists
- + great scenes film

	NB Counts	
	+	-
and	2	0
boxing	0	1
film	1	0
great	3	1
it	0	1
no	0	1
or	0	1
part	0	1
pathetic	0	1
plot	1	1
satire	1	0
scenes	1	2
the	0	2
twists	1	1
was	0	2
worst	0	1

Binary multinomial naive Bayes

Four original documents:

- it was pathetic the worst part was the boxing scenes
- no plot twists or great scenes
- + and satire and great plot twists
- + great scenes great film

After per-document binarization:

- it was pathetic the worst part boxing scenes
- no plot twists or great scenes
- + and satire great plot twists
- + great scenes film

	NB Counts		Binary Counts	
	+	-	+	-
and	2	0	1	0
boxing	0	1	0	1
film	1	0	1	0
great	3	1	2	1
it	0	1	0	1
no	0	1	0	1
or	0	1	0	1
part	0	1	0	1
pathetic	0	1	0	1
plot	1	1	1	1
satire	1	0	1	0
scenes	1	2	1	2
the	0	2	0	1
twists	1	1	1	1
was	0	2	0	1
worst	0	1	0	1

Counts can still be 2! Binarization is within-doc!

Text Classification and Naïve Bayes

More on Sentiment Classification

Sentiment Classification: Dealing with Negation

- I really like this movie

I really **don't** like this movie

Negation changes the meaning of "like" to negative.

Negation can also change negative to positive-ish

- **Don't** dismiss this film
- **Doesn't** let us get bored

Sentiment Classification: Dealing with Negation

Das, Sanjiv and Mike Chen. 2001. Yahoo! for Amazon: Extracting market sentiment from stock message boards. In Proceedings of the Asia Pacific Finance Association Annual Conference (APFA).

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up? Sentiment Classification using Machine Learning Techniques. EMNLP-2002, 79—86.

Simple baseline method:

Add NOT_ to every word between negation and following punctuation:

didn't like this movie , but I



didn't NOT_like NOT_this NOT_movie but I

Sentiment Classification: Lexicons

- Sometimes we don't have enough labeled training data
- In that case, we can make use of pre-built word lists
- Called **lexicons**
- There are various publically available lexicons

MPQA Subjectivity Cues Lexicon

Theresa Wilson, Janyce Wiebe, and Paul Hoffmann (2005). Recognizing Contextual Polarity in Phrase-Level Sentiment Analysis. Proc. of HLT-EMNLP-2005.

Riloff and Wiebe (2003). Learning extraction patterns for subjective expressions. EMNLP-2003.

- Home page: https://mpqa.cs.pitt.edu/lexicons/subj_lexicon/
- 6885 words from 8221 lemmas, annotated for intensity (strong/weak)
 - 2718 positive
 - 4912 negative
- + : *admirable, beautiful, confident, dazzling, ecstatic, favor, glee, great*
- - : *awful, bad, bias, catastrophe, cheat, deny, envious, foul, harsh, hate*

The General Inquirer

Philip J. Stone, Dexter C Dunphy, Marshall S. Smith, Daniel M. Ogilvie. 1966. The General Inquirer: A Computer Approach to Content Analysis. MIT Press

- Home page: <http://www.wjh.harvard.edu/~inquirer>
- List of Categories: <http://www.wjh.harvard.edu/~inquirer/homecat.htm>
- Spreadsheet: <http://www.wjh.harvard.edu/~inquirer/inquirerbasic.xls>
- Categories:
 - Positiv (1915 words) and Negativ (2291 words)
 - Strong vs Weak, Active vs Passive, Overstated versus Understated
 - Pleasure, Pain, Virtue, Vice, Motivation, Cognitive Orientation, etc
- Free for Research Use

Using Lexicons in Sentiment Classification

Add a feature that gets a count whenever a word from the lexicon occurs

- E.g., a feature called "**this word occurs in the positive lexicon**" or "**this word occurs in the negative lexicon**"

Now all positive words (*good, great, beautiful, wonderful*) or negative words count for that feature.

Using 1-2 features isn't as good as using all the words.

- But when training data is sparse or not representative of the test set, dense lexicon features can help

Naive Bayes in Other tasks: Spam Filtering

- SpamAssassin Features:
 - Mentions millions of (dollar) ((dollar) NN,NNN,NNN.NN)
 - From: starts with many numbers
 - Subject is all capitals
 - HTML has a low ratio of text to image area
 - "One hundred percent guaranteed"
 - Claims you can be removed from the list

Naive Bayes in Language ID

- Determining what language a piece of text is written in.
Features based on character n-grams do very well
- Important to train on lots of varieties of each language
(e.g., American English varieties like African-American English, or English varieties around the world like Indian English)

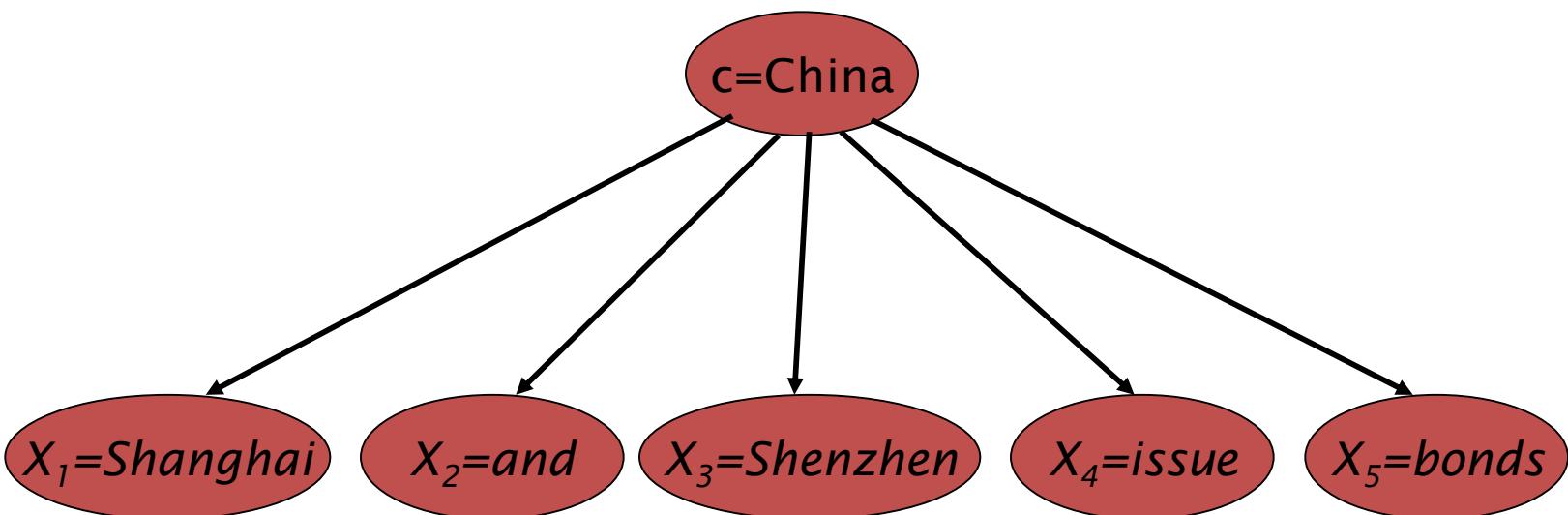
Summary: Naive Bayes is Not So Naive

- Very Fast, low storage requirements
- Work well with very small amounts of training data
- Robust to Irrelevant Features
 - Irrelevant Features cancel each other without affecting results
- Very good in domains with many equally important features
 - Decision Trees suffer from *fragmentation* in such cases – especially if little data
- Optimal if the independence assumptions hold: If assumed independence is correct, then it is the Bayes Optimal Classifier for problem
- A good dependable baseline for text classification
 - **But we will see other classifiers that give better accuracy**

Text Classification and Naïve Bayes

**Naïve Bayes: Relationship to
Language Modeling**

Generative Model for Multinomial Naïve Bayes



Naïve Bayes and Language Modeling

- Naïve bayes classifiers can use any sort of feature
 - URL, email address, dictionaries, network features
- But if, as in the previous slides
 - We use **only** word features
 - we use **all** of the words in the text (not a subset)
- Then
 - Naïve bayes has an important similarity to language modeling.

Each class = a unigram language model

- Assigning each word: $P(\text{word} \mid c)$
- Assigning each sentence: $P(s \mid c) = \angle P(\text{word} \mid c)$

Class *pos*

0.1	I		<u>I</u>	<u>love</u>	<u>this</u>	<u>fun</u>	<u>film</u>
0.1	love		0.1	0.1	.05	0.01	0.1
0.01	this						
0.05	fun						
0.1	film						
							$P(s \mid \text{pos}) = 0.0000005$

Naïve Bayes as a Language Model

- Which class assigns the higher probability to s ?

Model pos		Model neg		I	love	this	fun	film
0.1	I	0.2	I	—	—	—	—	—
0.1	love	0.001	love	0.1	0.1	0.01	0.05	0.1
0.01	this	0.01	this	0.2	0.001	0.01	0.005	0.1
0.05	fun	0.005	fun	—	—	—	—	—
0.1	film	0.1	film	—	—	—	—	—

$P(s|pos) > P(s|neg)$

Text Classification and Naïve Bayes

Precision, Recall, and the F measure

Evaluation

- Let's consider just binary text classification tasks
- Imagine you're the CEO of Delicious Pie Company
- You want to know what people are saying about your pies
- So you build a "Delicious Pie" tweet detector
 - Positive class: tweets about Delicious Pie Co
 - Negative class: all other tweets

The 2-by-2 confusion matrix

gold standard labels

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

Evaluation: Accuracy

- Why don't we use **accuracy** as our metric?
- Imagine we saw 1 million tweets
 - 100 of them talked about Delicious Pie Co.
 - 999,900 talked about something else
- We could build a dumb classifier that just labels every tweet "not about pie"
 - It would get 99.99% accuracy!!! Wow!!!!
 - But useless! Doesn't return the comments we are looking for!
 - That's why we use **precision** and **recall** instead

Evaluation: Precision

- % of items the system detected (i.e., items the system labeled as positive) that are in fact positive (according to the human gold labels)

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

Evaluation: Recall

- % of items actually present in the input that were correctly identified by the system.

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

Why Precision and recall

- Our dumb pie-classifier
 - Just label nothing as "about pie"

Accuracy=99.99%

but

Recall = 0

- (it doesn't get any of the 100 Pie tweets)

Precision and recall, unlike accuracy, emphasize true positives:

- finding the things that we are supposed to be looking for.

A combined measure: F

- F measure: a single number that combines P and R:

$$F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2P + R}$$

- We almost always use balanced F_1 (i.e., $\beta = 1$)

$$F_1 = \frac{2PR}{P + R}$$

Development Test Sets ("Devsets") and Cross-validation

Training set

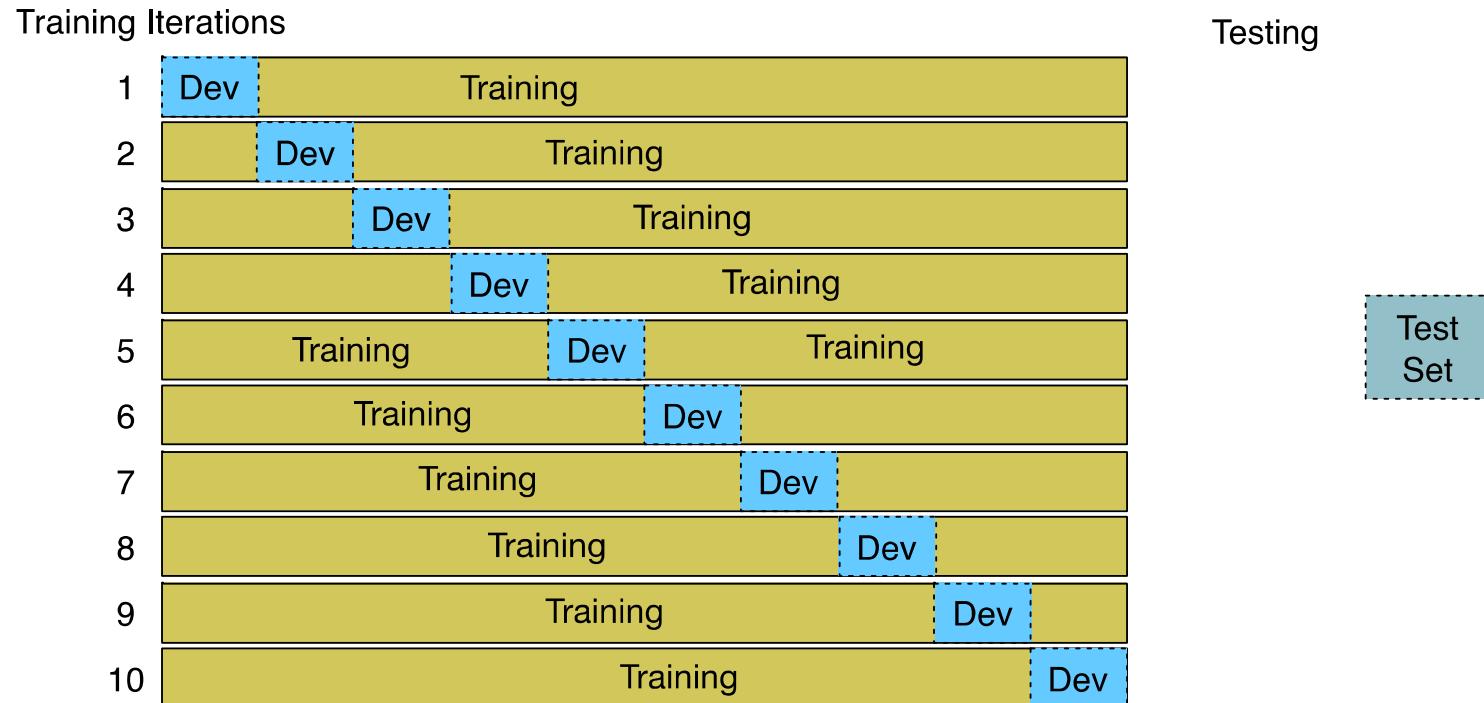
Development Test Set

Test Set

- Train on training set, tune on devset, report on testset
 - This avoids overfitting ('tuning to the test set')
 - More conservative estimate of performance
 - But paradox: want as much data as possible for training, and as much for dev; how to split?

Cross-validation: multiple splits

- Pool results over splits, Compute pooled dev performance



Text Classification and Naïve Bayes

**Text Classification: Evaluation
with more than two classes**

Confusion Matrix for 3-class classification

		<i>gold labels</i>			
		urgent	normal	spam	
<i>system output</i>	urgent	8	10	1	$\text{precision}_u = \frac{8}{8+10+1}$
	normal	5	60	50	$\text{precision}_n = \frac{60}{5+60+50}$
	spam	3	30	200	$\text{precision}_s = \frac{200}{3+30+200}$
		$\text{recall}_u = \frac{8}{8+5+3}$	$\text{recall}_n = \frac{60}{10+60+30}$	$\text{recall}_s = \frac{200}{1+50+200}$	

How to combine P/R from 3 classes to get one metric

- Macroaveraging:
 - compute the performance for each class, and then average over classes
- Microaveraging:
 - collect decisions for all classes into one confusion matrix
 - compute precision and recall from that table.

Macroaveraging and Microaveraging

Class 1: Urgent

true	true	
urgent	not	
system	8	11
urgent	8	340
system	8	340

Class 2: Normal

true	true	
normal	not	
system	60	55
normal	60	55
system	40	212
not	40	212

Class 3: Spam

true	true	
spam	not	
system	200	33
spam	200	33
system	51	83
not	51	83

Pooled

true	true	
yes	no	
system	268	99
yes	268	99
system	99	635
no	99	635

$$\text{precision} = \frac{8}{8+11} = .42$$

$$\text{precision} = \frac{60}{60+55} = .52$$

$$\text{precision} = \frac{200}{200+33} = .86$$

$$\text{microaverage precision} = \frac{268}{268+99} = .73$$

$$\text{macroaverage precision} = \frac{.42+.52+.86}{3} = .60$$

Text Classification and Naive Bayes

Statistical Significance Testing

How do we know if one classifier is better than another?

- Given:
 - Classifier A and B
 - Metric M: $M(A,x)$ is the performance of A on testset x
 - $\delta(x)$: the performance difference between A, B on x:
 - $\delta(x) = M(A,x) - M(B,x)$
- We want to know if $\delta(x)>0$, meaning A is better than B
- $\delta(x)$ is called the **effect size**
- Suppose we look and see that $\delta(x)$ is positive. Are we done?
- No! This might be just an accident of this one test set, or circumstance of the experiment. Instead:

Statistical Hypothesis Testing

- Consider two hypotheses:
 - Null hypothesis: A isn't better than B
 - A is better than B
- We want to rule out H_0
- We create a random variable X ranging over test sets
- And ask, how likely, if H_0 is true, is it that among these test sets we would see the $\delta(x)$ we did see?
- Formalized as the p-value:

$$H_0 : \delta(x) \leq 0$$

$$H_1 : \delta(x) > 0$$

$$P(\delta(X) \geq \delta(x) | H_0 \text{ is true})$$

Statistical Hypothesis Testing

$$P(\delta(X) \geq \delta(x) | H_0 \text{ is true})$$

- In our example, this p-value is the probability that we would see $\delta(x)$ assuming H_0 ($=A$ is not better than B).
 - If H_0 is true but $\delta(x)$ is huge, that is surprising! Very low probability!
- A very small p-value means that the difference we observed is very unlikely under the null hypothesis, and we can reject the null hypothesis
- Very small: .05 or .01
- A result(e.g., “ A is better than B ”) is **statistically significant** if the δ we saw has a probability that is below the threshold and we therefore reject this null hypothesis.

Statistical Hypothesis Testing

- How do we compute this probability?
- In NLP, we don't tend to use parametric tests (like t-tests)
- Instead, we use non-parametric tests based on sampling: artificially creating many versions of the setup.
- For example, suppose we had created zillions of testsets x' .
 - Now we measure the value of $\delta(x')$ on each test set
 - That gives us a distribution
 - Now set a threshold (say .01).
 - So if we see that in 99% of the test sets $\delta(x) > \delta(x')$
 - We conclude that our original test set delta was a real delta and not an artifact.

Statistical Hypothesis Testing

- Two common approaches:
 - approximate randomization
 - bootstrap test
- Paired tests:
 - Comparing two sets of observations in which each observation in one set can be paired with an observation in another.
 - For example, when looking at systems A and B **on the same test set**, we can compare the performance of system A and B on each same observation x_i

Text Classification and Naive Bayes

The Paired Bootstrap Test

Bootstrap test

Efron and Tibshirani, 1993

Can apply to any metric (accuracy, precision, recall, F1, etc).

Bootstrap means to repeatedly draw large numbers of smaller samples with replacement (called **bootstrap samples**) from an original larger sample.

Bootstrap example

Consider a baby text classification example with a test set x of 10 documents, using accuracy as metric.

Suppose these are the results of systems A and B on x , with 4 outcomes (A & B both right, A & B both wrong, A right/B wrong, A wrong/B right):

	either A+B both correct, or										A%	B%	$d()$
	1	2	3	4	5	6	7	8	9	10			
x	AB	AB	AB	AB	AB	AB	AB	AB	AB	AB	.70	.50	.20

Bootstrap example

- Now we create, many, say, $b=10,000$ virtual test sets $x(i)$, each of size $n = 10$.
 - To make each $x(i)$, we randomly select a cell from row x , with replacement, 10 times:

Bootstrap example

- Now we have a distribution! We can check how often A has an **accidental** advantage, to see if the original $\delta(x)$ we saw was very common.
- Now assuming H_0 , that means normally we expect $\delta(x')=0$
- So we just count how many times the $\delta(x')$ we found exceeds the expected 0 value by $\delta(x)$ or more:

$$\text{p-value}(x) = \Pr_{\mathcal{X}^B} \left[\frac{1}{n} \sum_{i=1}^n d(x^{(i)}) - d(x) \geq 0 \right]$$

Bootstrap example

- Alas, it's slightly more complicated.
- We didn't draw these samples from a distribution with 0 mean; we created them from the original test set x , which happens to be biased (by .20) in favor of A .
- So to measure how surprising is our observed $\delta(x)$, we actually compute the p-value by counting how often $\delta(x')$ exceeds the expected value of $\delta(x)$ by $\delta(x)$ or more:

$$\text{p-value}(x) = \mathbb{P}_{\substack{\mathcal{X}^B \\ i=1}} \left[d(x^{(i)}) - d(x) \geq \delta(x) \right]$$

$$= \mathbb{P}_{\substack{\mathcal{X}^B \\ i=1}} \left[d(x^{(i)}) \geq 2d(x) \right]$$

Bootstrap example

- Suppose:
 - We have 10,000 test sets $x(i)$ and a threshold of .01
 - And in only 47 of the test sets do we find that $\delta(x(i)) \geq 2\delta(x)$
 - The resulting p-value is .0047
 - This is smaller than .01, indicating $\delta(x)$ is indeed sufficiently surprising
 - And we reject the null hypothesis and conclude A is better than B .

Paired bootstrap example

After Berg-Tirkpatrick et al (2012)

function BOOTSTRAP(test set x , num of samples b) **returns** $p\text{-value}(x)$

Calculate $\delta(x)$ # how much better does algorithm A do than B on x

$s = 0$

for $i = 1$ **to** b **do**

for $j = 1$ **to** n **do** # Draw a bootstrap sample $x^{(i)}$ of size n

 Select a member of x at random and add it to $x^{(i)}$

 Calculate $\delta(x^{(i)})$ # how much better does algorithm A do than B on $x^{(i)}$

$s \leftarrow s + 1$ **if** $\delta(x^{(i)}) > 2\delta(x)$

$p\text{-value}(x) \approx \frac{s}{b}$ # on what % of the b samples did algorithm A beat expectations?

return $p\text{-value}(x)$ # if very few did, our observed δ is probably not accidental

Text Classification and Naive Bayes

Avoiding Harms in
Classification

Harms in sentiment classifiers

- Kiritchenko and Mohammad (2018) found that most sentiment classifiers assign lower sentiment and more negative emotion to sentences with African American names in them.
- This perpetuates negative stereotypes that associate African Americans with negative emotions

Harms in toxicity classification

- Toxicity detection is the task of detecting hate speech, abuse, harassment, or other kinds of toxic language
- But some toxicity classifiers incorrectly flag as being toxic sentences that are non-toxic but simply mention identities like blind people, women, or gay people.
- This could lead to censorship of discussion about these groups.

What causes these harms?

- Can be caused by:
 - Problems in the training data; machine learning systems are known to amplify the biases in their training data.
 - Problems in the human labels
 - Problems in the resources used (like lexicons)
 - Problems in model architecture (like what the model is trained to optimized)
- Mitigation of these harms is an open research area
- Meanwhile: **model cards**

Model Cards

(Mitchell et al., 2019)

- For each algorithm you release, document:
 - training algorithms and parameters
 - training data sources, motivation, and preprocessing
 - evaluation data sources, motivation, and preprocessing
 - intended use and users
 - model performance across different demographic or other groups and environmental situations

Thanks for Your Attention!

Chap. 5: Logistic Regression

Outline

- Background: generative and discriminative classifiers
- Classification in logistic regression
- Logistic Regression: a text example on sentiment classification
- Learning: Cross-Entropy Loss
- Stochastic Gradient Descent
- Stochastic Gradient Descent: An example and more details
- Regularization
- Multinomial Logistic Regression

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

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

- Logistic Regression

$$\hat{c} = \operatorname{argmax}_{c \in C} \frac{\overbrace{P(c/d)}^{\text{posterior}}}{\overbrace{P(d/c)}^{\text{prior}}}$$

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

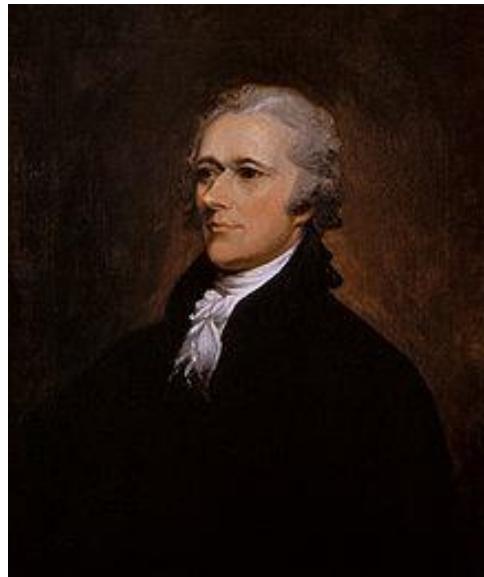
- **Training:** we learn weights w and b using **stochastic gradient descent** and **cross-entropy loss**.
- **Test:** Given a test example x we compute $p(y|x)$ using learned weights w and b , and return whichever label ($y = 1$ or $y = 0$) is higher probability

Logistic Regression

- Classification in 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

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

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

- 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:
 $p(y=1|x; \theta)$
 $p(y=0|x; \theta)$

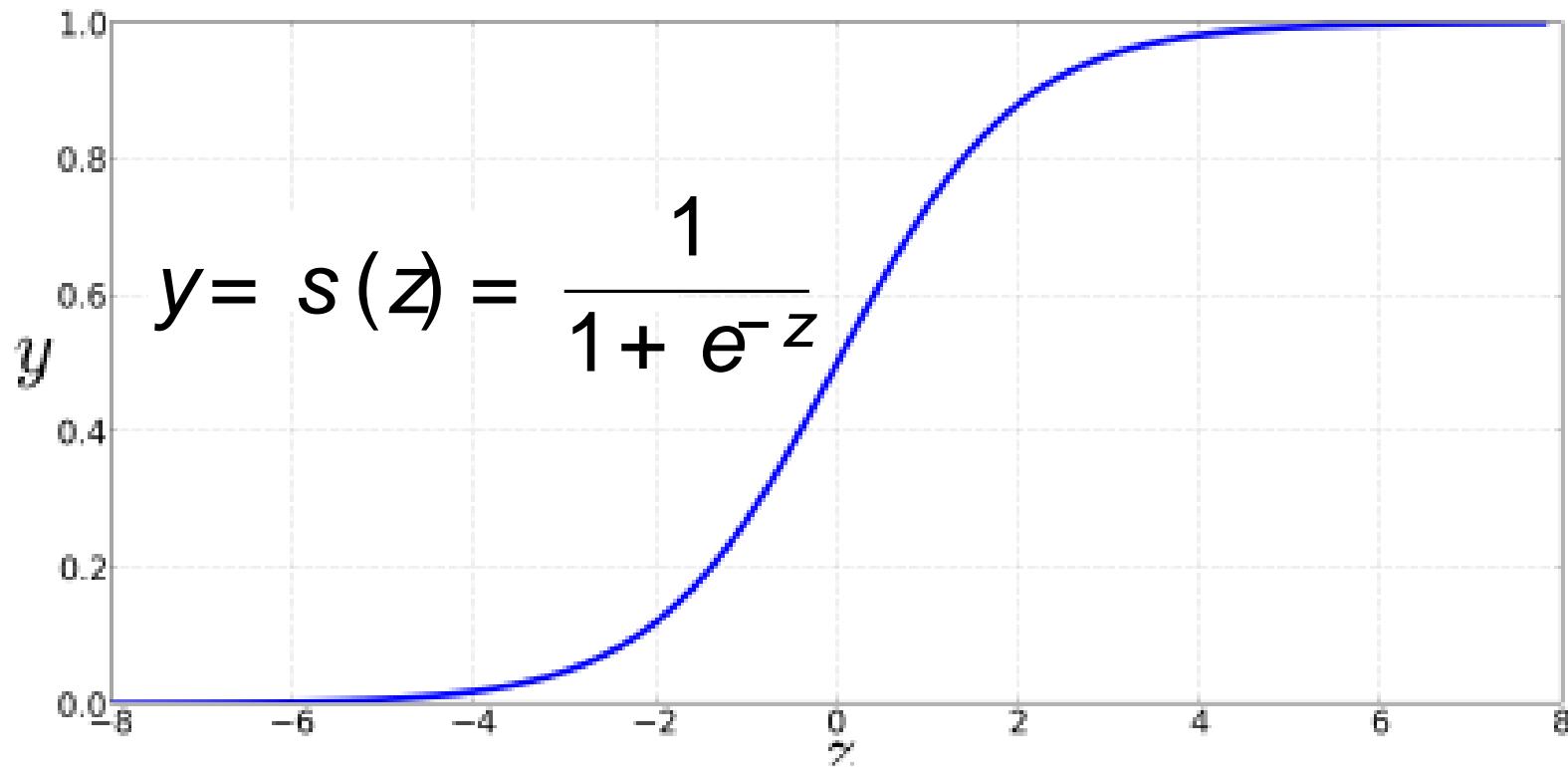
The problem: z isn't a probability, it's just a number!

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

- 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:

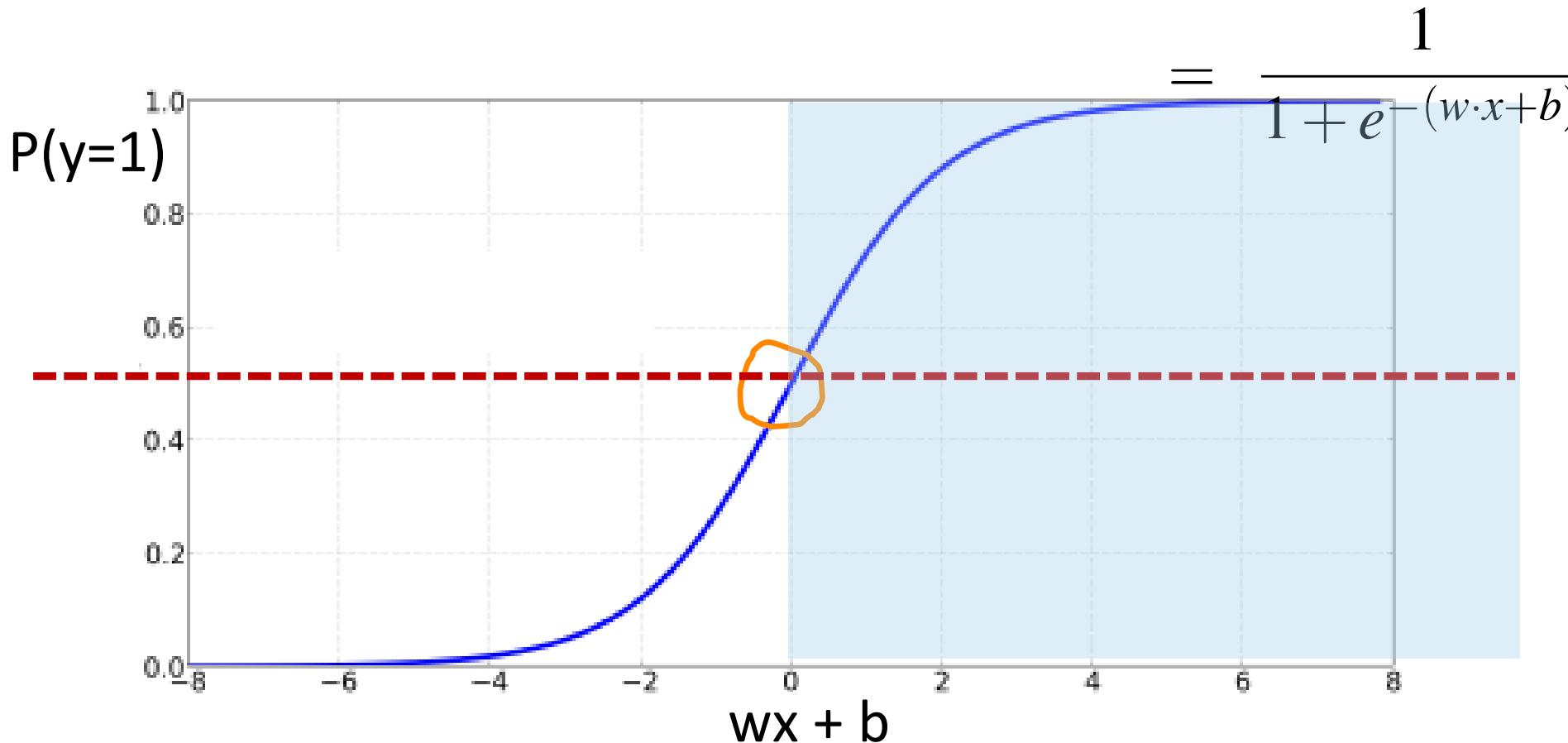
$$\begin{aligned} P(y=0) &= 1 - \sigma(w \cdot x + b) & = \sigma(-(w \cdot x + b)) \\ &= 1 - \frac{1}{1 + \exp(-(w \cdot x + b))} & \text{Because} \\ &= \frac{\exp(-(w \cdot x + b))}{1 + \exp(-(w \cdot x + b))} & 1 - \sigma(x) = \sigma(-x) \end{aligned}$$

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



Turning a probability into a classifier

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

if $w \cdot x + b > 0$
if $w \cdot x + b \leq 0$

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

5.2

Var	Definition	Val
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 $w = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7]$

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

- Learning: Cross-Entropy Loss

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 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 $y=1$)

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

- Is lower than the loss when model was wrong (if true $y=0$):³⁶

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

- Sure enough, loss was bigger when model was wrong!^{1.2}

Logistic Regression

- Stochastic Gradient Descent

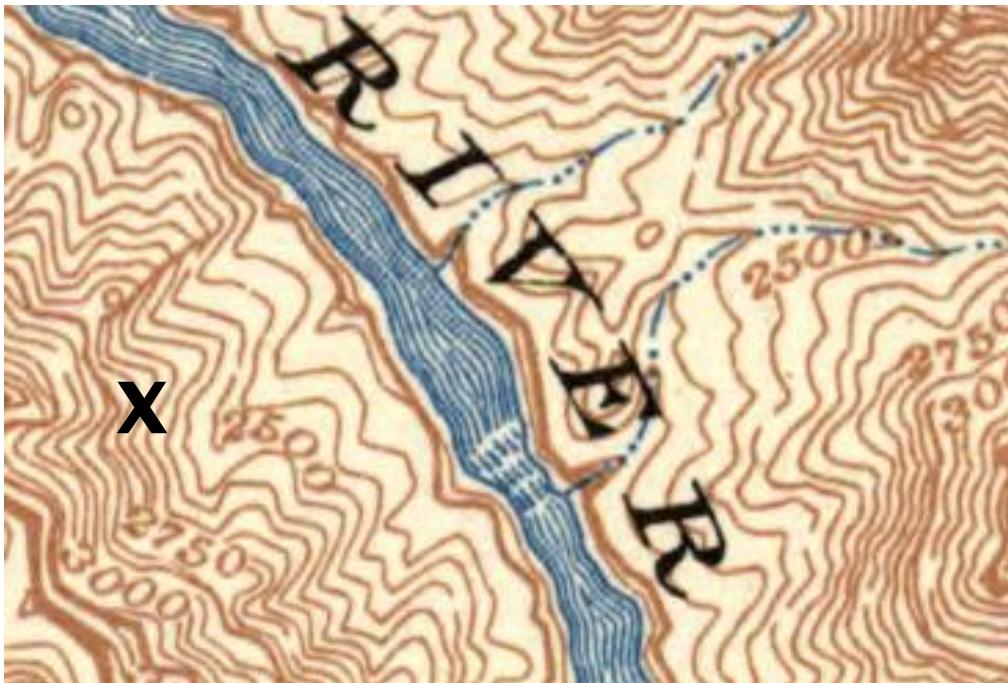
Our goal: minimize the loss

- 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

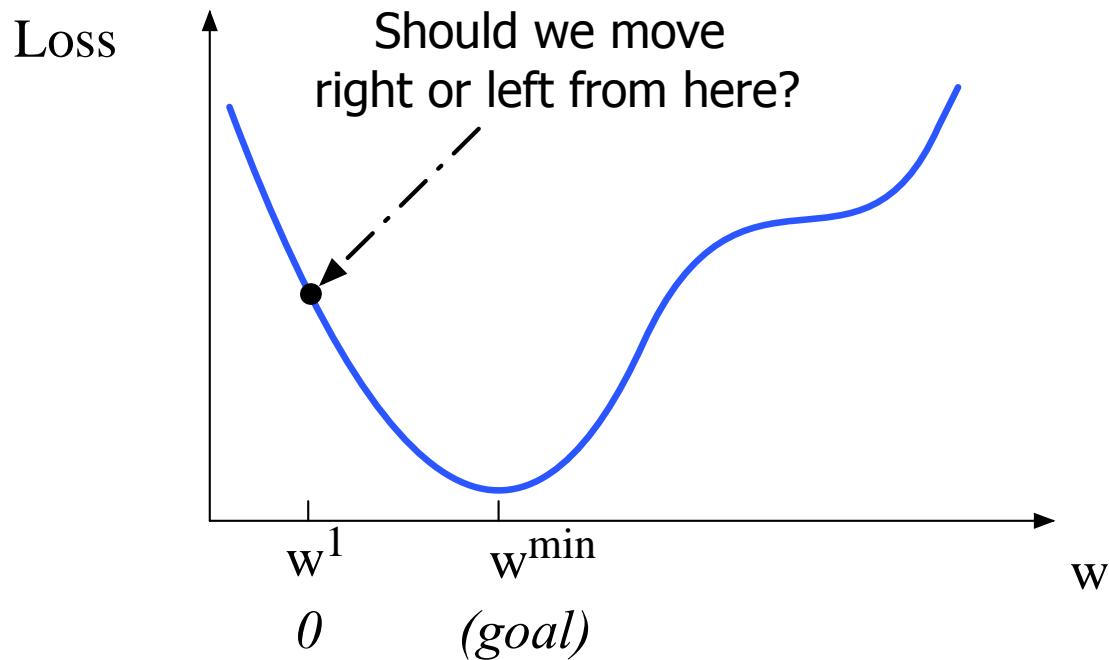
Our goal: minimize the loss

- 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?

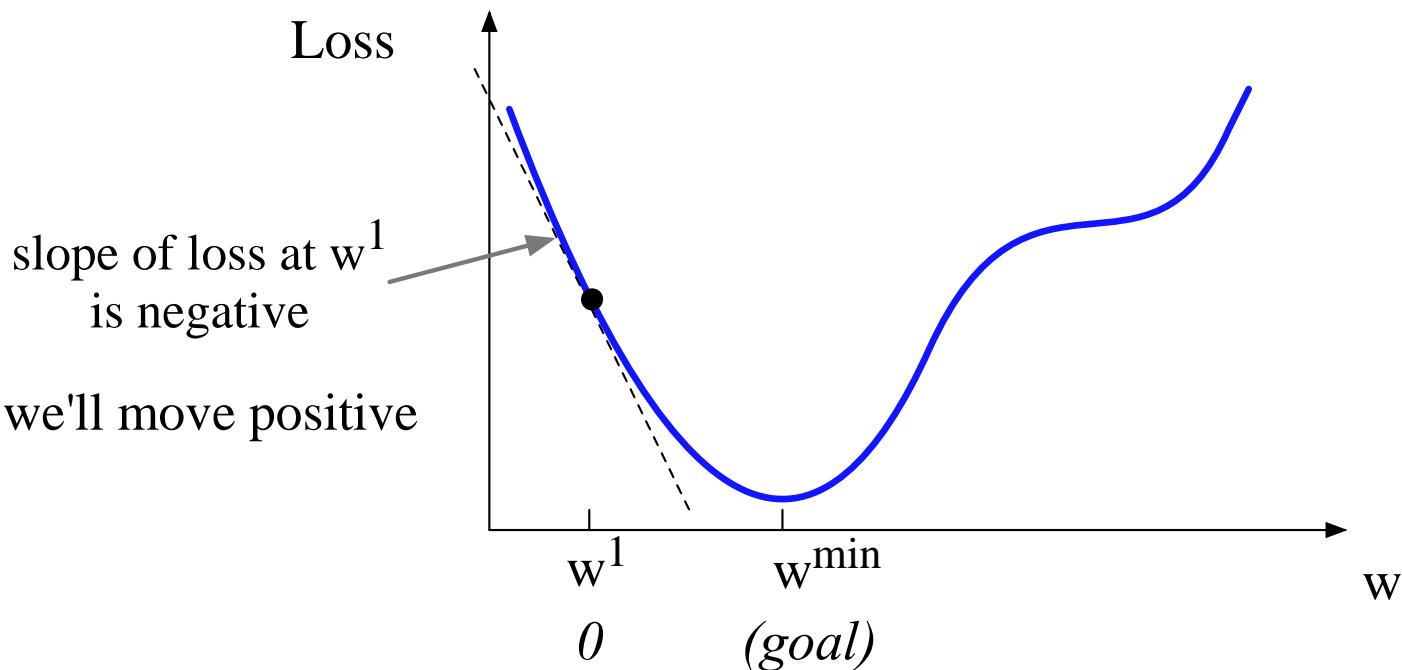
A: Move w in the reverse direction from the slope of the function



Let's first visualize for a single scalar w

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

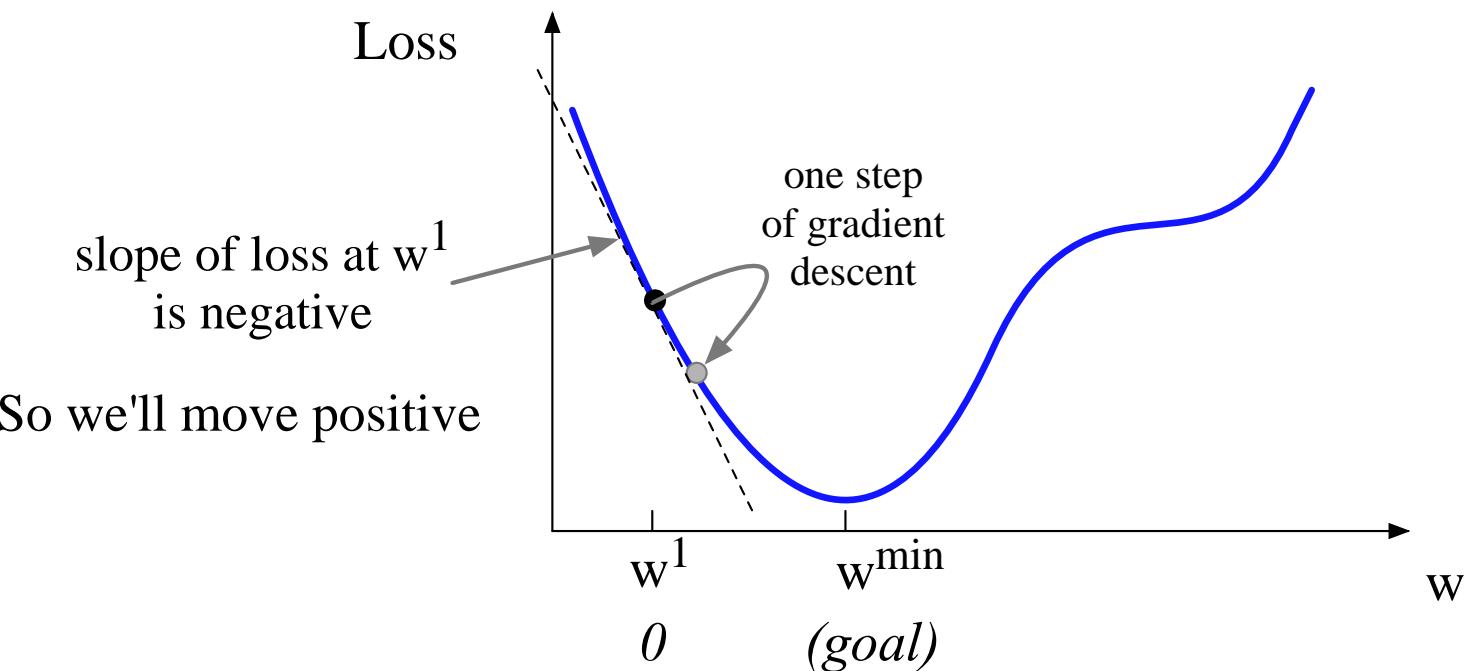
A: Move w in the reverse direction from the slope of the function



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

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

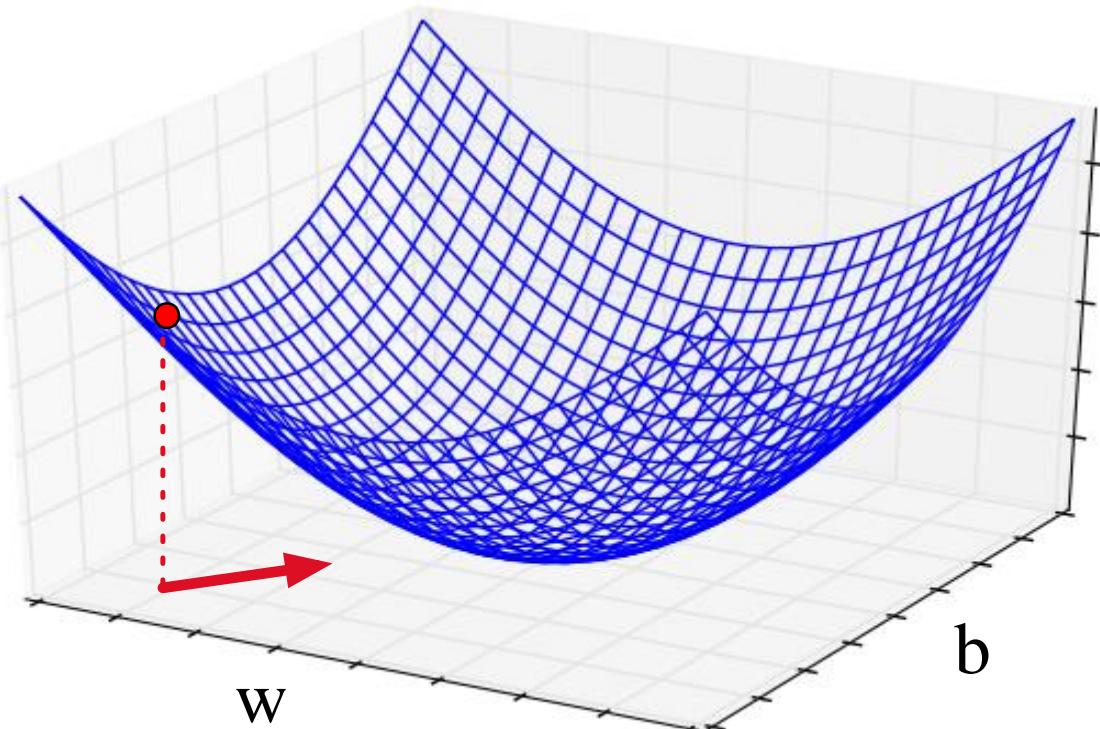
Now let's consider N dimensions

- 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

Cost(w, 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.

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:

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

- **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**
- 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:

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

- **Example of gradient descent**
- Update step for update θ is:

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

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

- 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

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

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:

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

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

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

- Regularization

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

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

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
- 83 • 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 $\rightarrow y=0$

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

- Multinomial: 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

Thanks for Your Attention!

Chap. 6: Vector Semantics and Embeddings

Outline

Word meaning

Vector semantics

Words and vectors

Cosine for computing word similarity

TF-IDF

PPMI

Word2Vec

Word2Vec: learning the embeddings

Properties of 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!

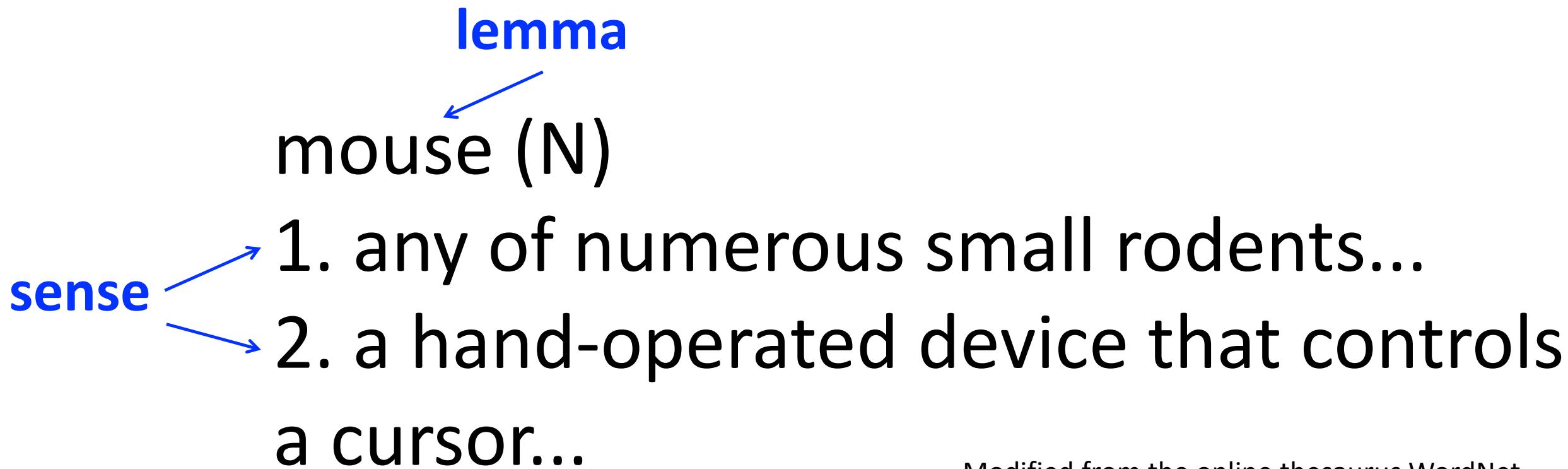
Desiderata

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



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 / H₂O

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

car, bicycle

cow, horse

Ask humans how similar 2 words are

word1	word2	similarity
vanish	disappear	9.8
behave	obey	7.3
belief	impression	5.95
muscle	bone	3.65
modest	flexible	0.98
hole	agreement	0.3

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

Values from NRC VAD Lexicon (Mohammad 2018)

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

Vector Semantics

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 "sauteed"

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

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**

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

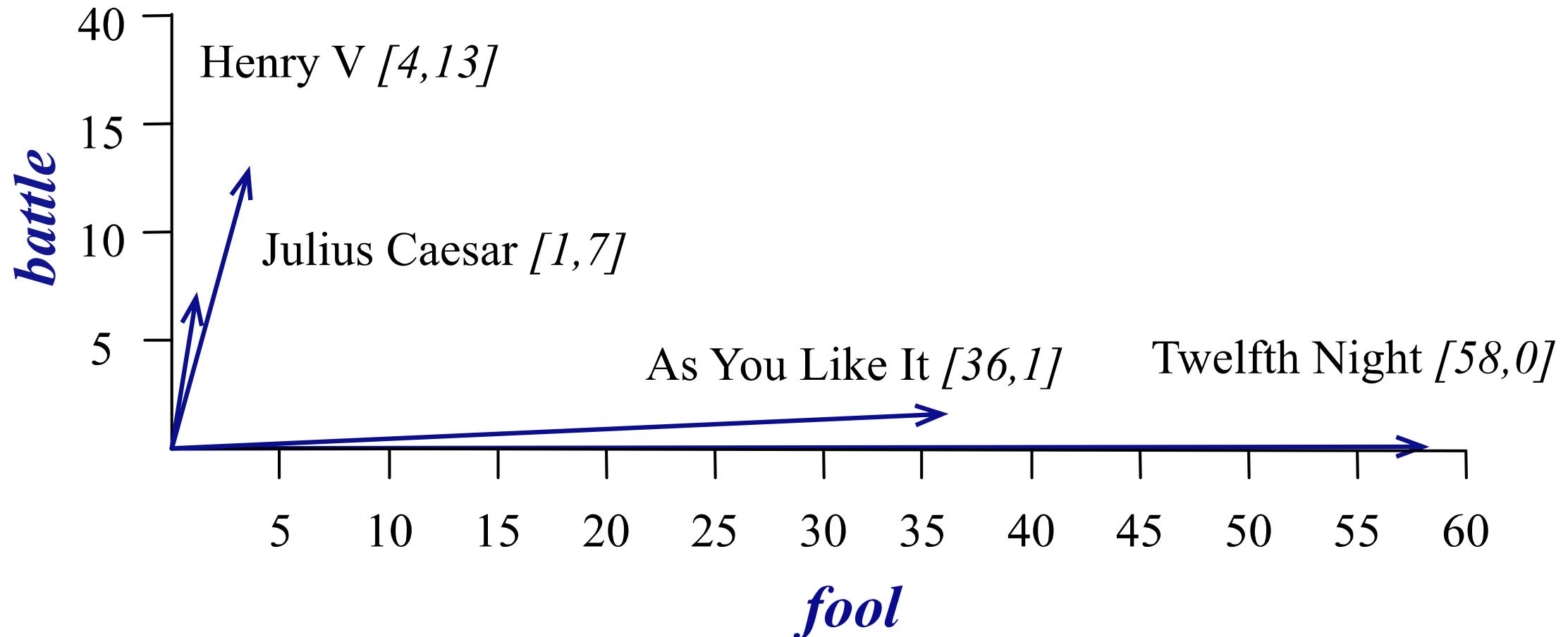
Words and Vectors

Term-document matrix

Each document is represented by a vector of words

	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

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!!!

	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

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"

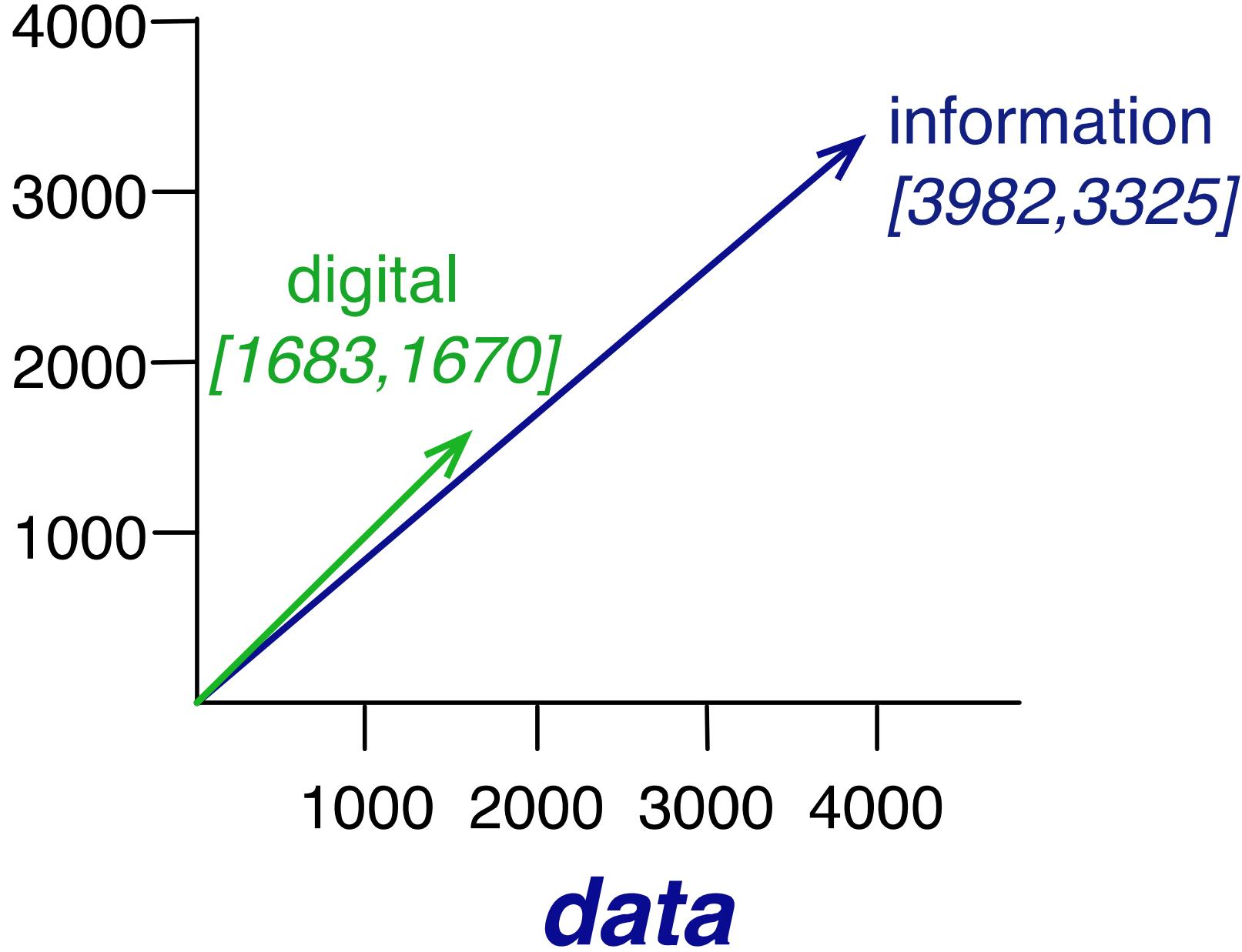
More common: word-word matrix (or "term-context matrix")

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

computer



Cosine for computing word similarity

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

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

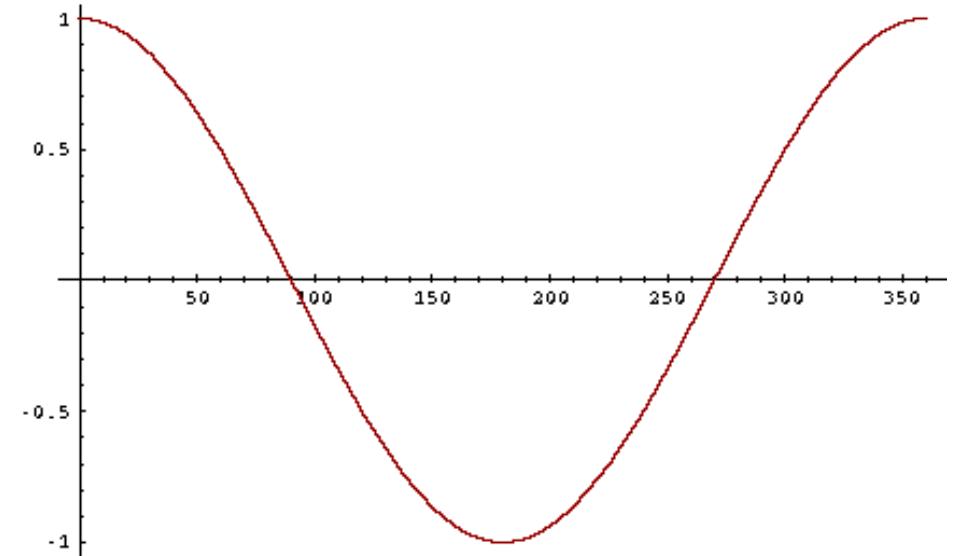
Based on the definition of the dot product between two vectors a and b

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

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

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

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

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

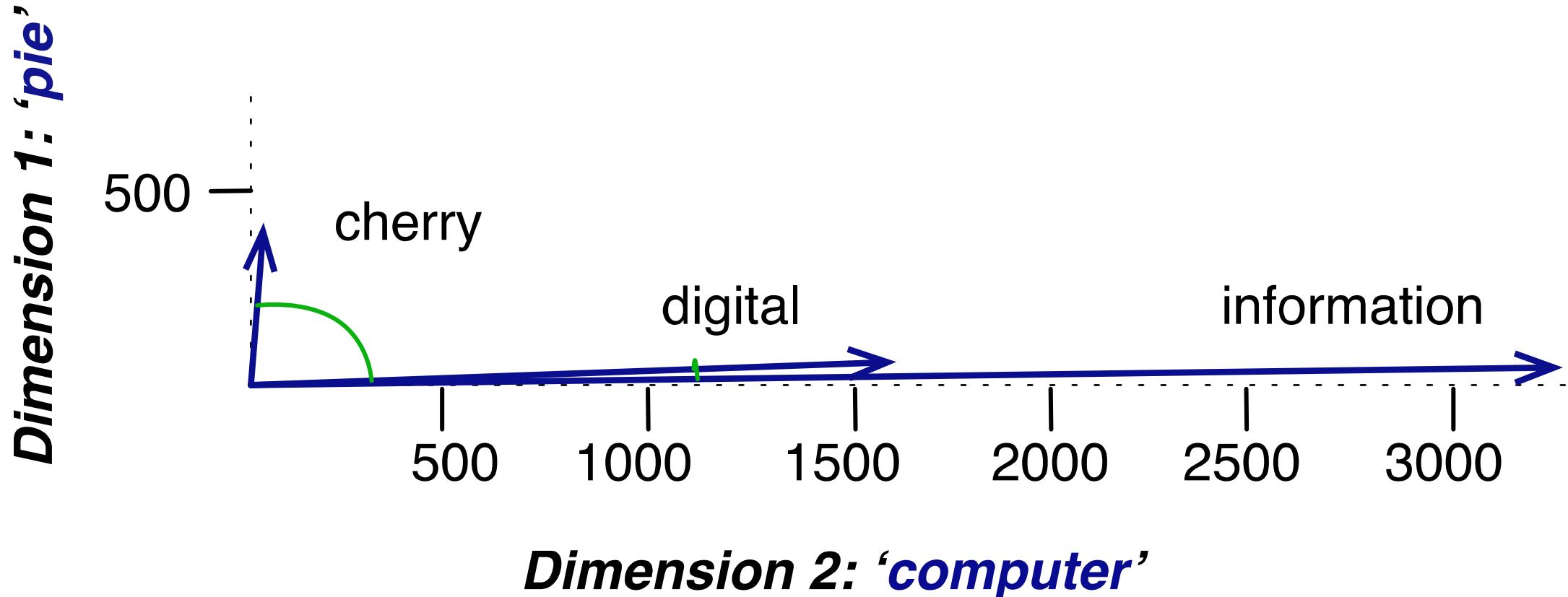
$$\frac{442 * 5 + 8 * 3982 + 2 * 3325}{\sqrt{442^2 + 8^2 + 2^2} \sqrt{5^2 + 3982^2 + 3325^2}} = .017$$

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

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

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

Visualizing cosines (well, angles)



TF-IDF

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

PPMI

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

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

$$p(w=information) = 7703/11716 = .6575$$

$$p(c=data) = 5673/11716 = .4842$$

$$\hat{a}^C f_{ij} \quad \hat{a}^W f_{ij}$$

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

	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

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

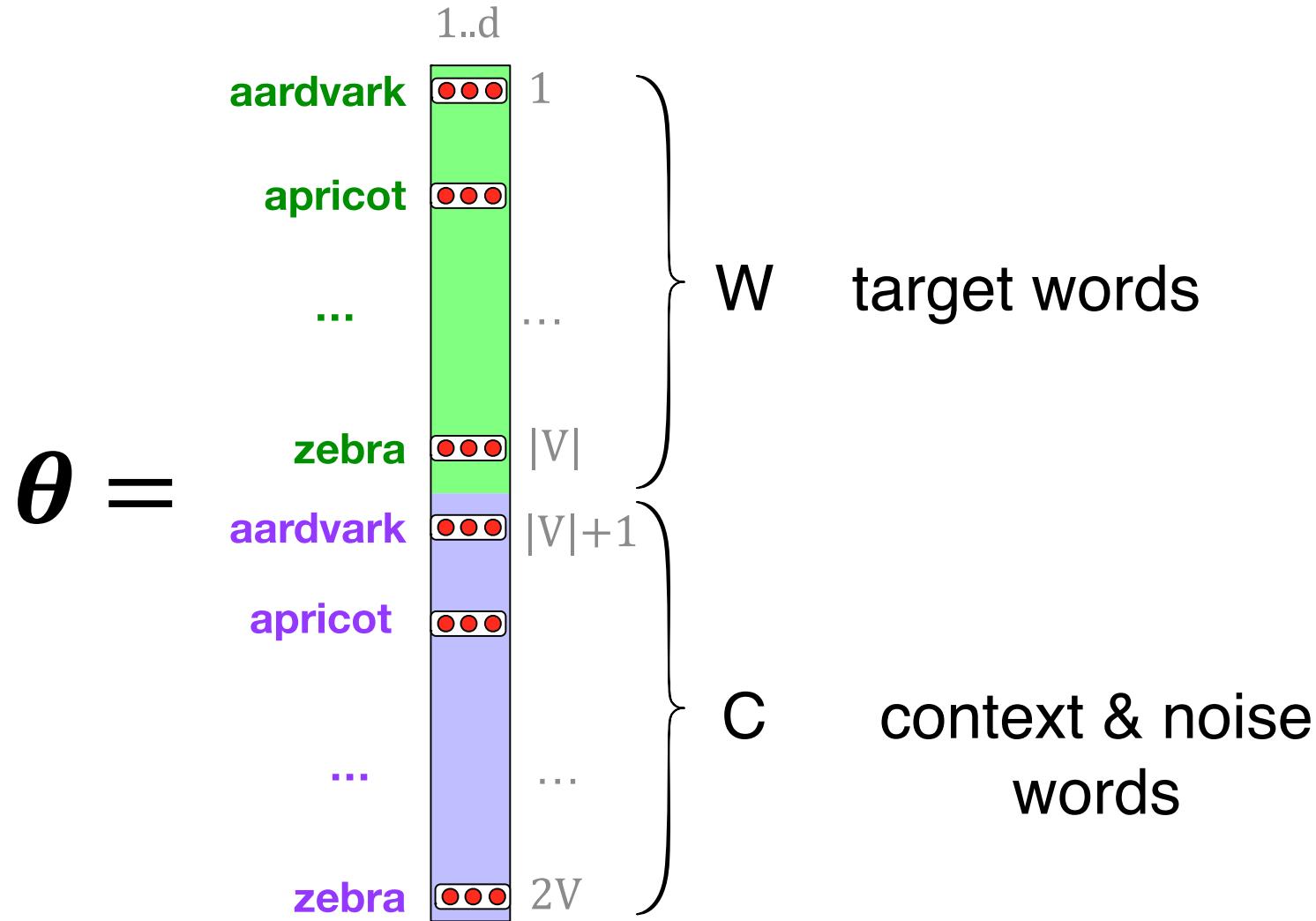
A probabilistic classifier, given

- 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: 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

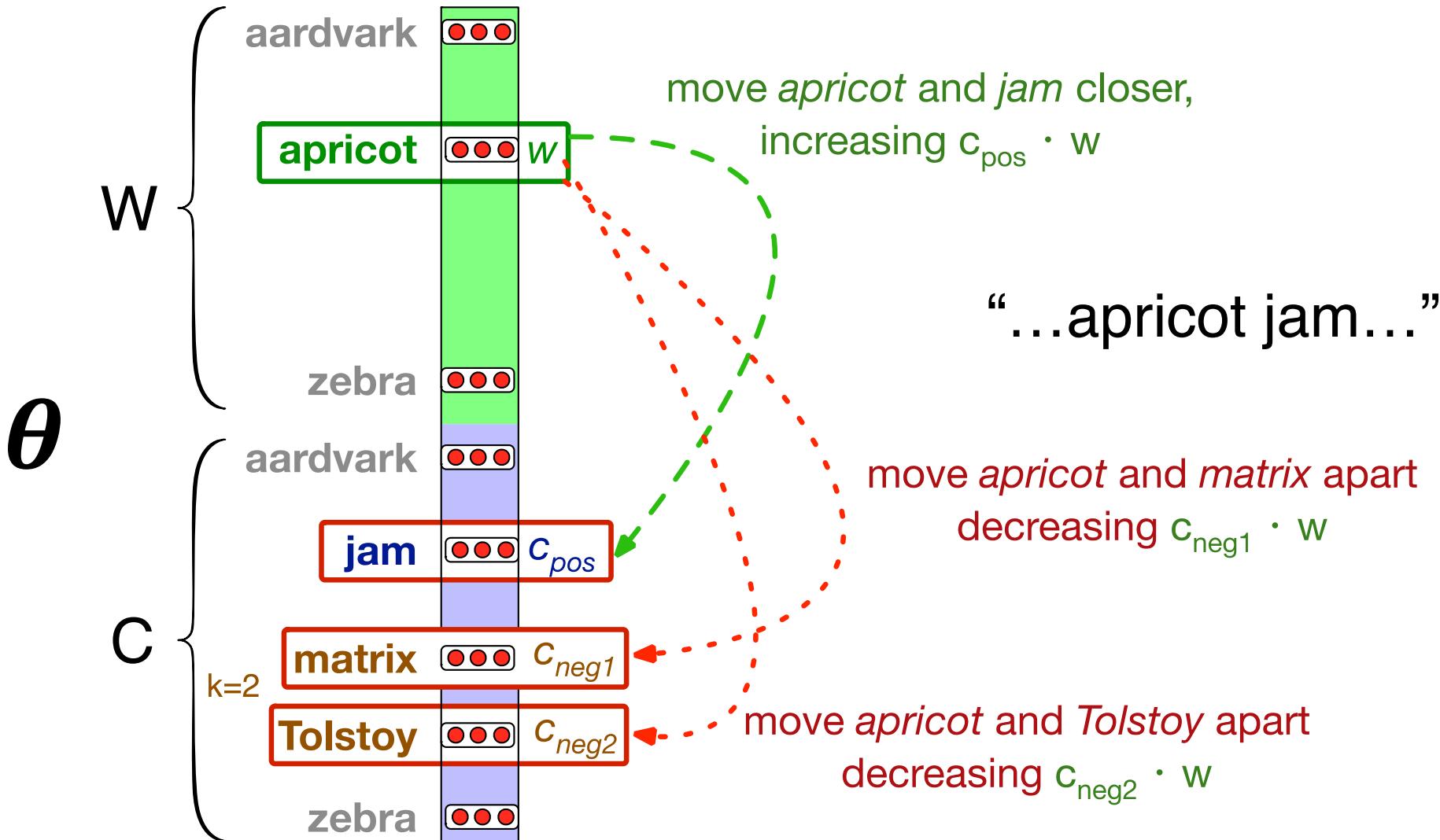
How to learn?

- 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.

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

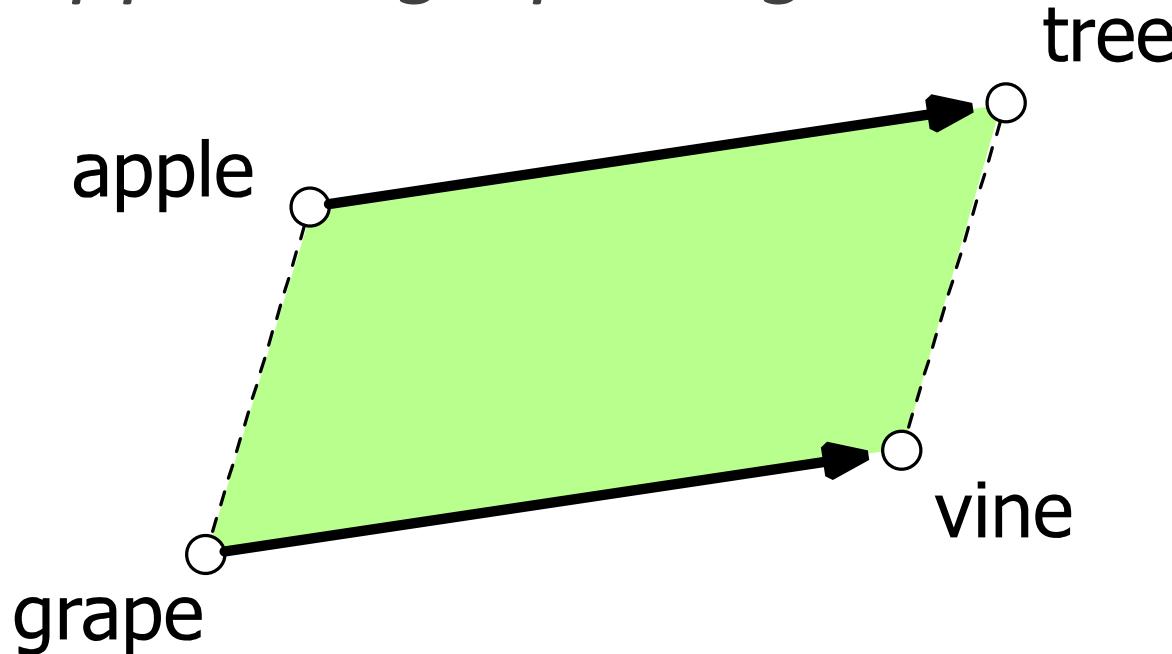
- *Hogwarts* nearest neighbors are Harry Potter world:
 - *Dumbledore, half-blood, Malfoy*

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)

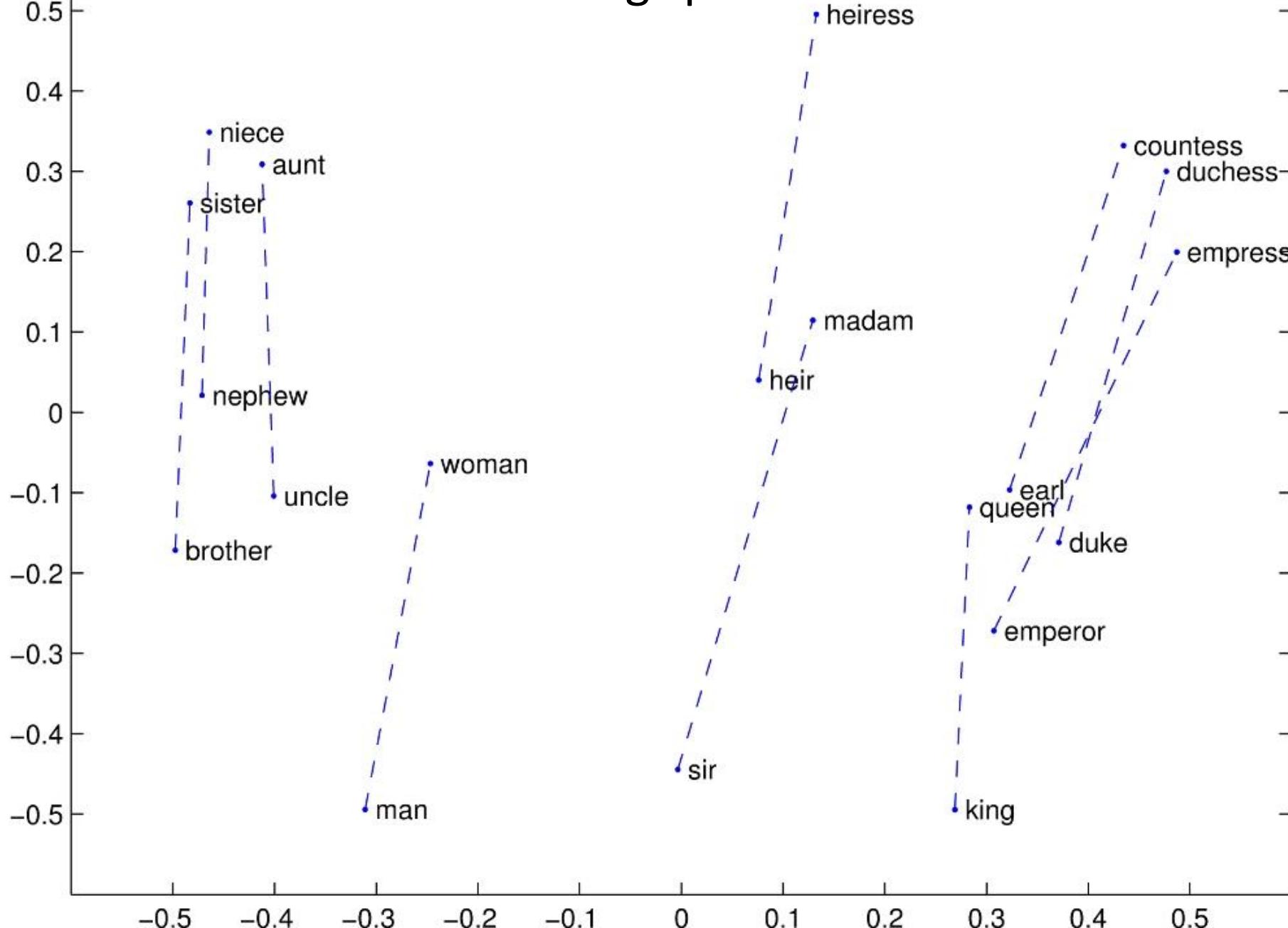
$$\overrightarrow{\text{king}} - \overrightarrow{\text{man}} + \overrightarrow{\text{woman}} \text{ is close to } \overrightarrow{\text{queen}}$$

$$\overrightarrow{\text{Paris}} - \overrightarrow{\text{France}} + \overrightarrow{\text{Italy}} \text{ is close to } \overrightarrow{\text{Rome}}$$

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

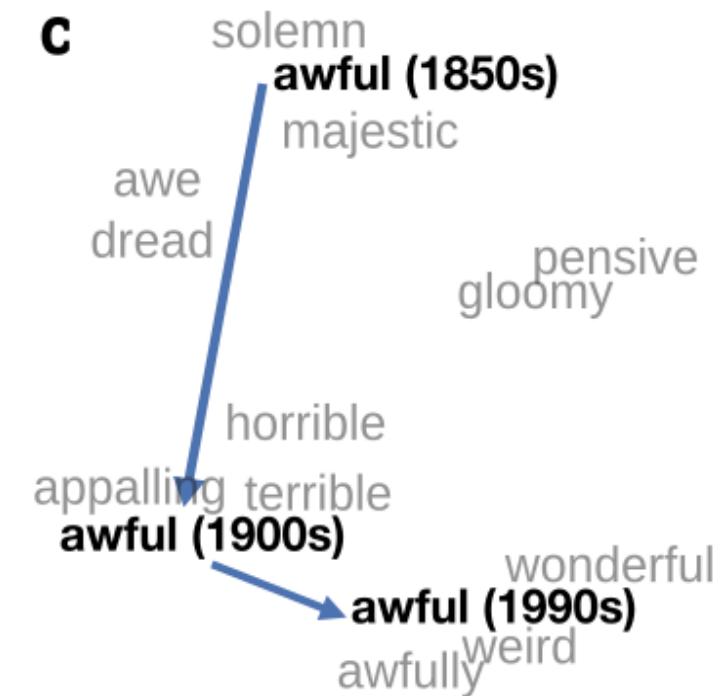
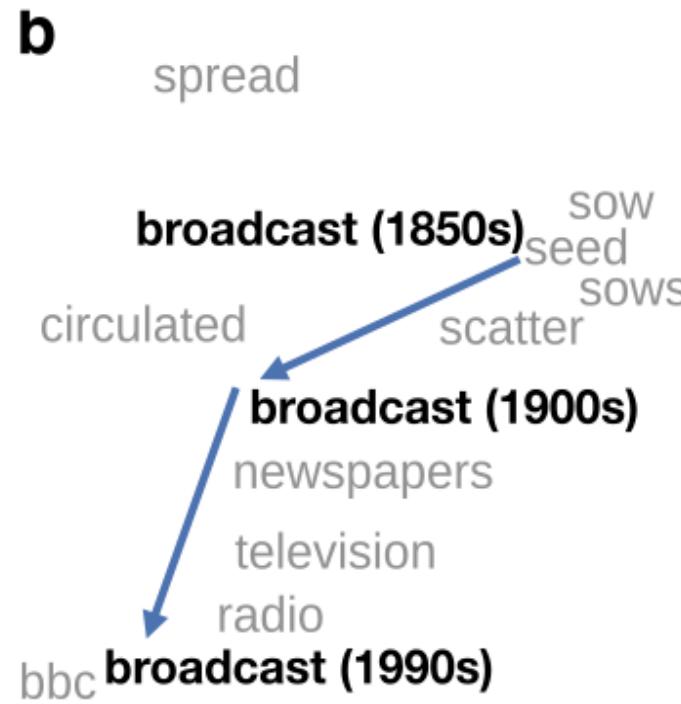
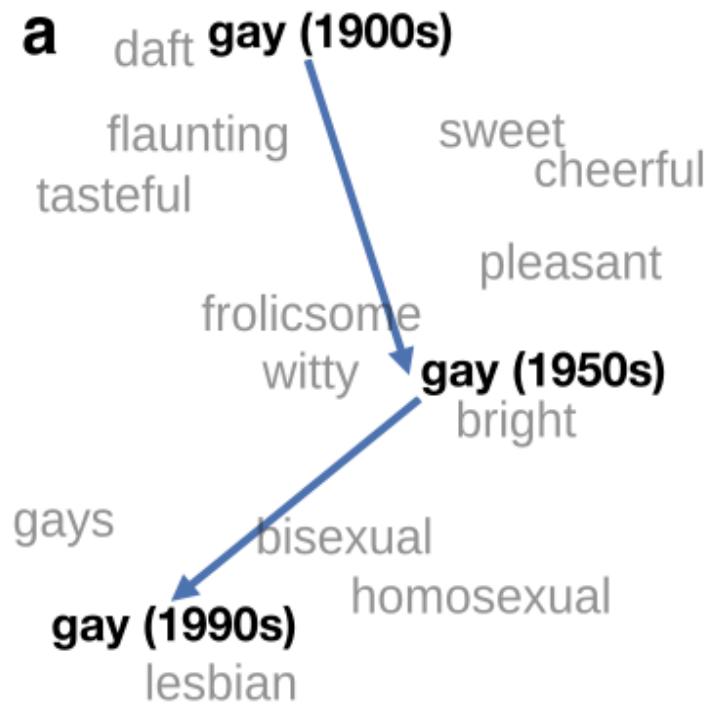
It only seems to work for frequent words, small distances and certain relations (relating countries to capitals, or parts of speech), but not others. (Linzen 2016, Gladkova et al. 2016, Ethayarajh et al. 2019a)

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

Thanks for Your Attention!

Chap. 7: Neural Networks and Neural Language Models

Outline

Units in Neural Networks

The XOR problem

Feedforward neural networks

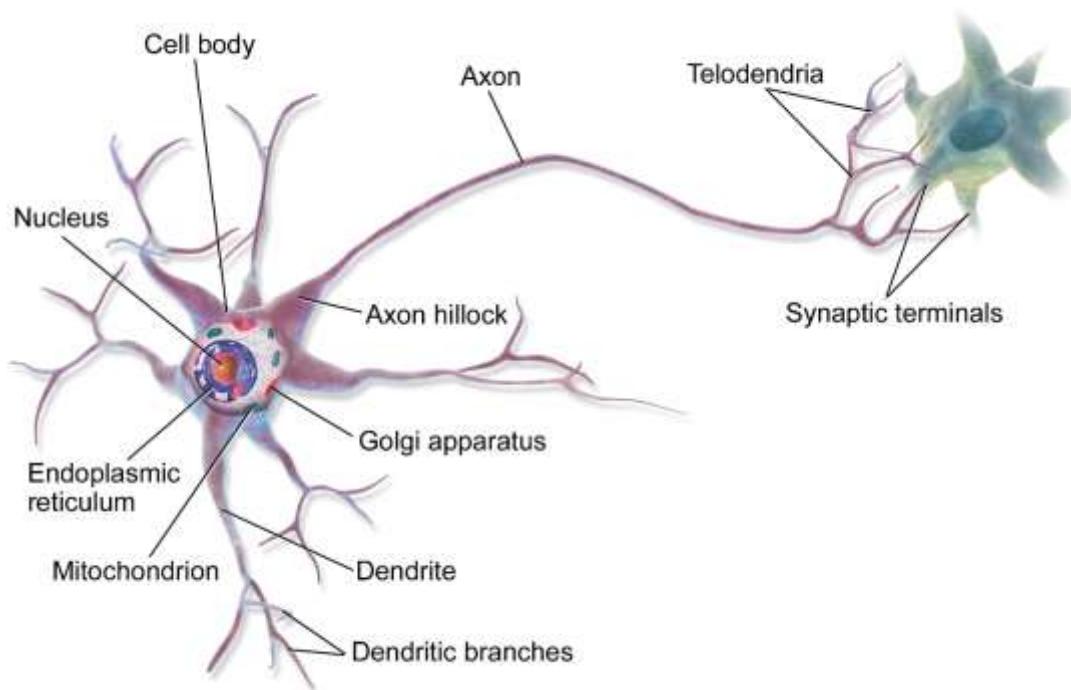
Applying feedforward networks to NLP tasks

Training neural nets: overview

Computation graphs and backward differentiation

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

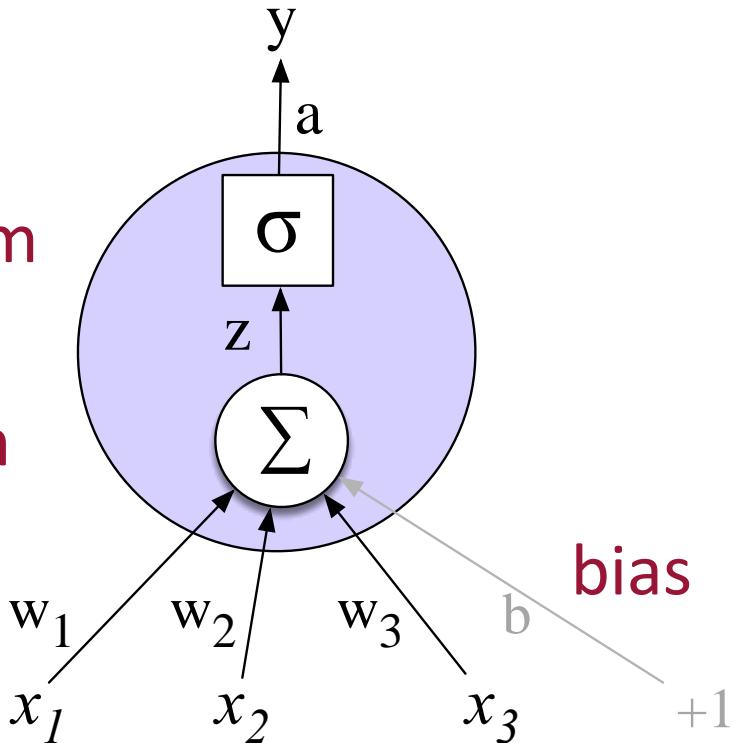
Output value

Non-linear transform

Weighted sum

Weights

Input layer



Neural unit

Take weighted sum of inputs, plus a bias

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

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

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

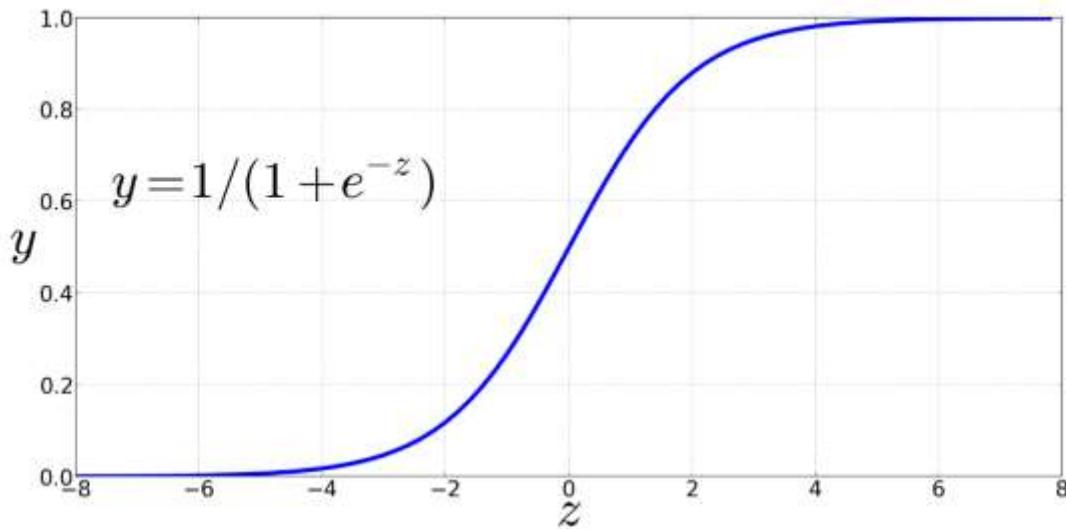
$$y = a = f(z)$$

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

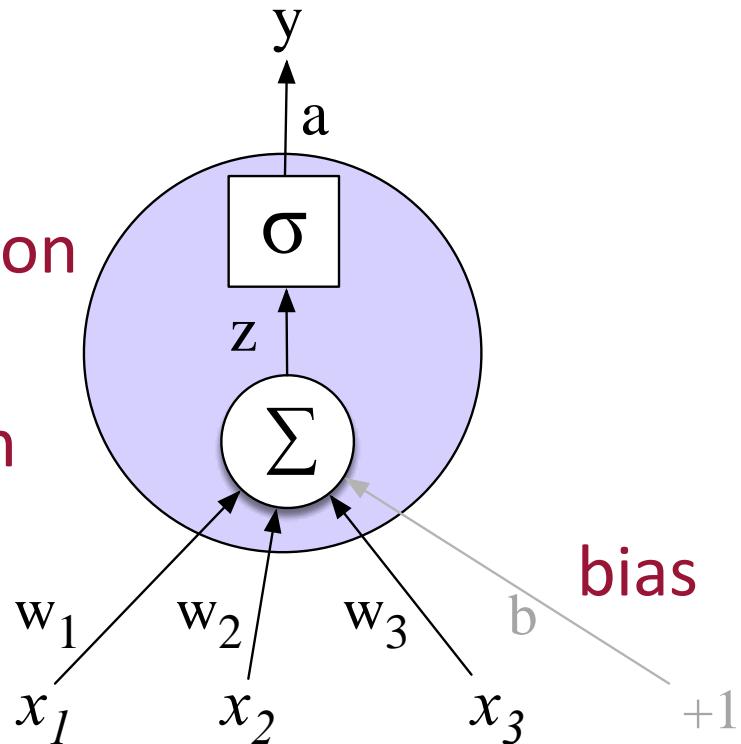
Output value

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 :

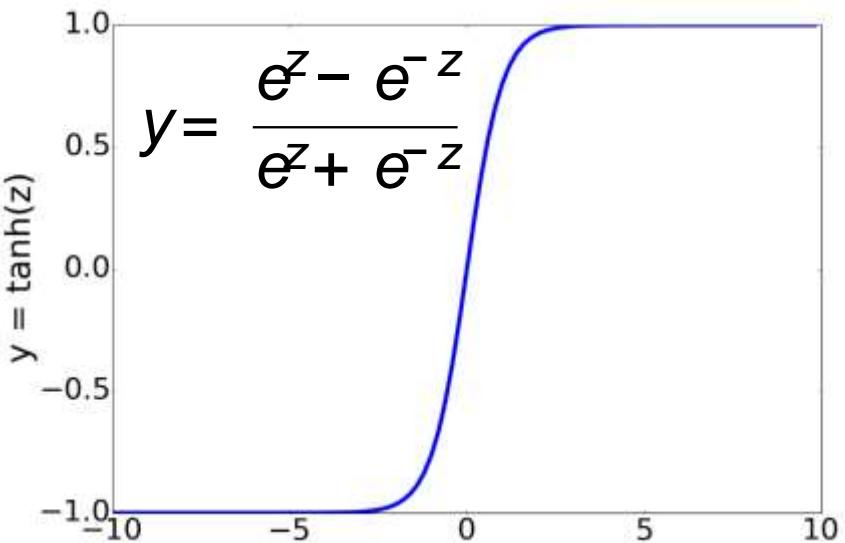
$$x = [0.5, 0.6, 0.1]$$

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

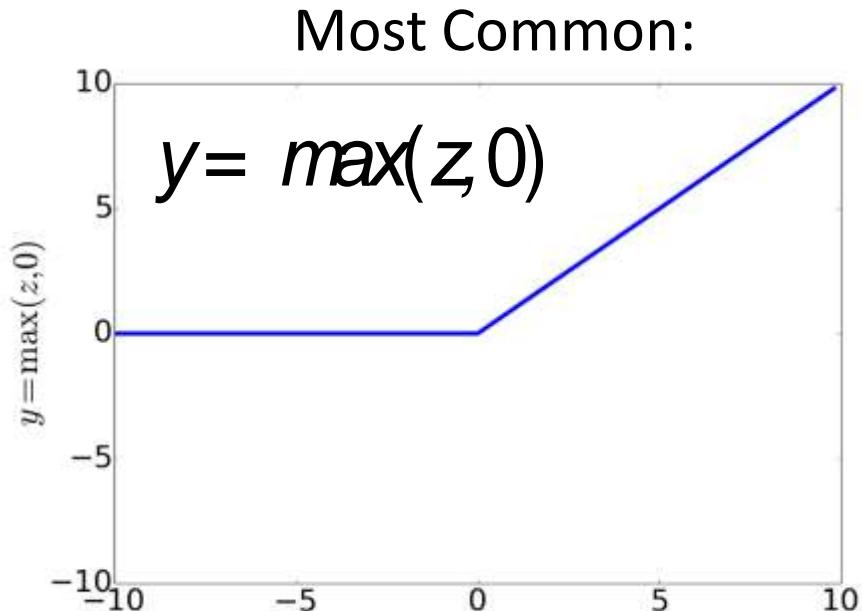
$$y = s(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}} =$$

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

Non-Linear Activation Functions besides sigmoid



tanh



ReLU
Rectified Linear Unit

The XOR problem

The XOR problem

Minsky and Papert (1969)

Can neural units compute simple functions of input?

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

Perceptrons

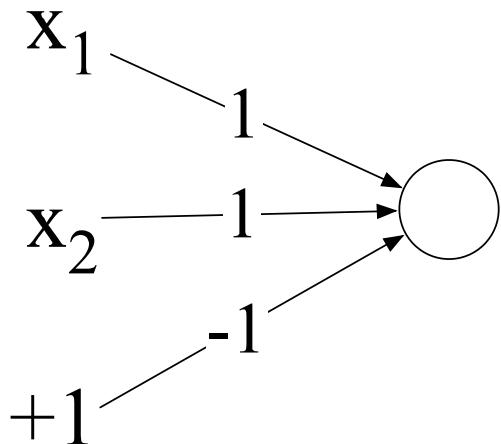
A very simple neural unit

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

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

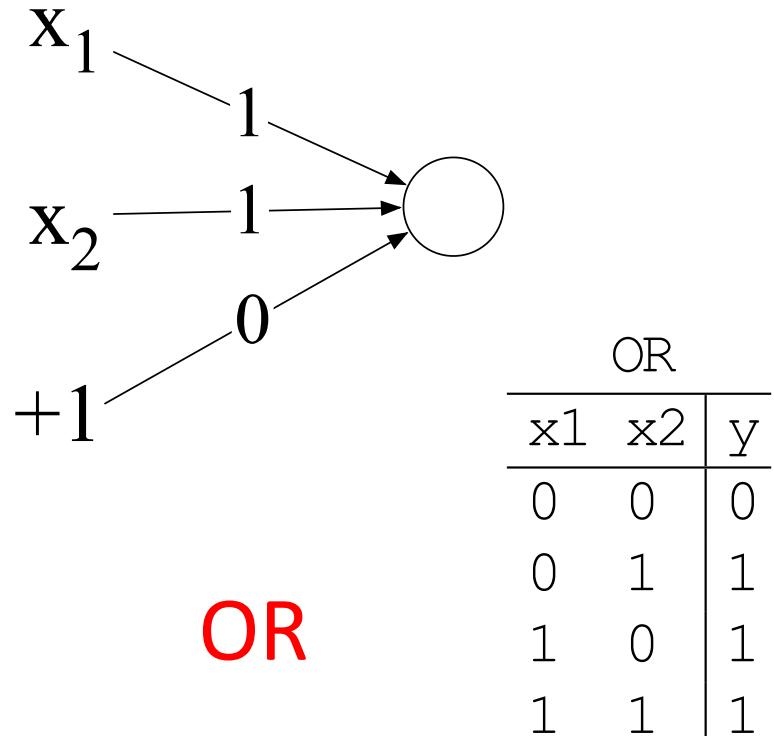
Easy to build AND or OR with perceptrons

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



AND

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

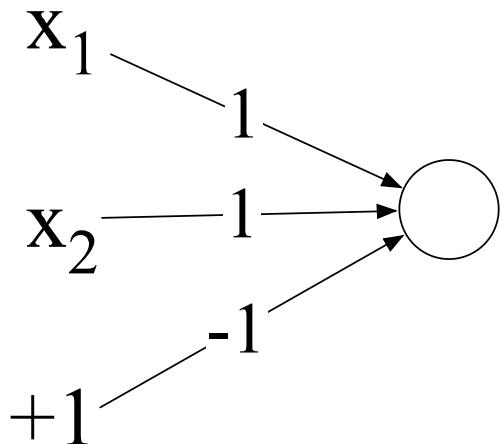


OR

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

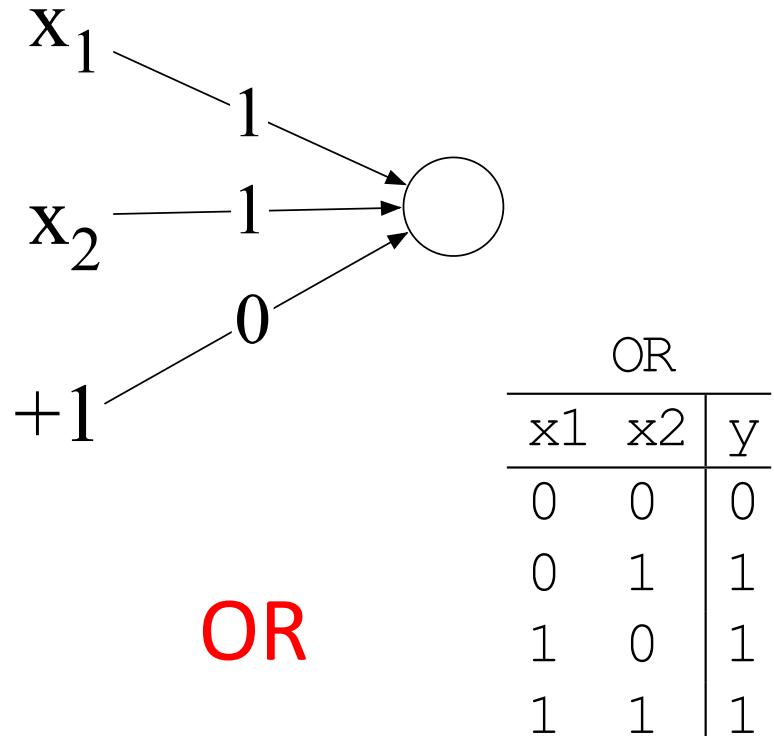
Easy to build AND or OR with perceptrons

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



AND

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

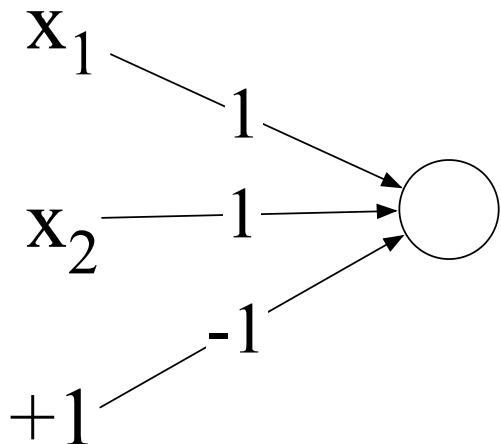


OR

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

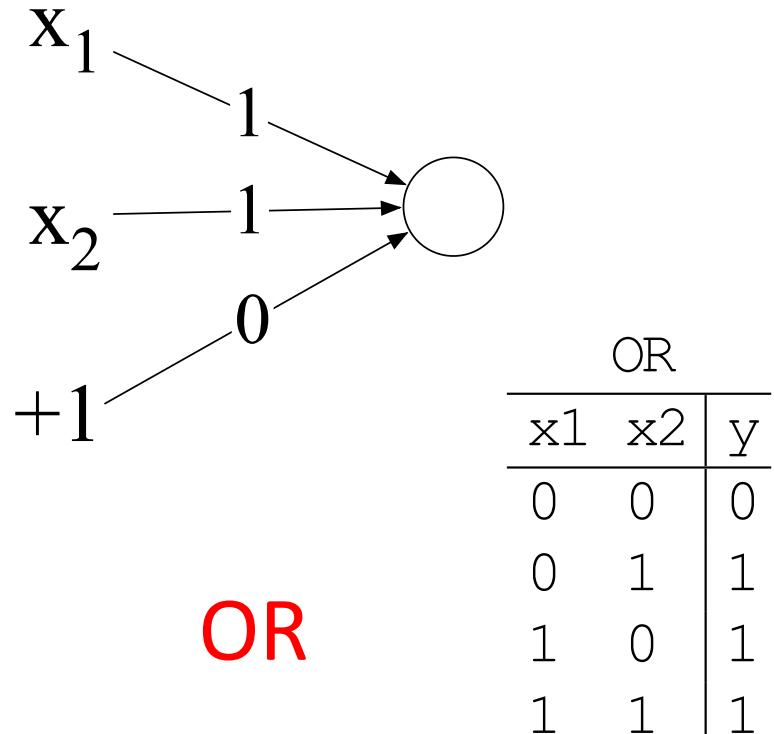
Easy to build AND or OR with perceptrons

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



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

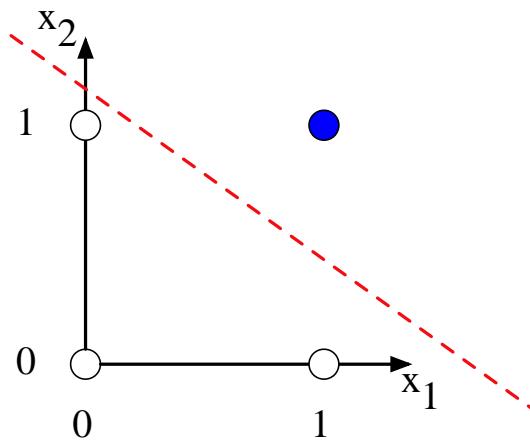
$$w_1x_1 + w_2x_2 + b = 0$$

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

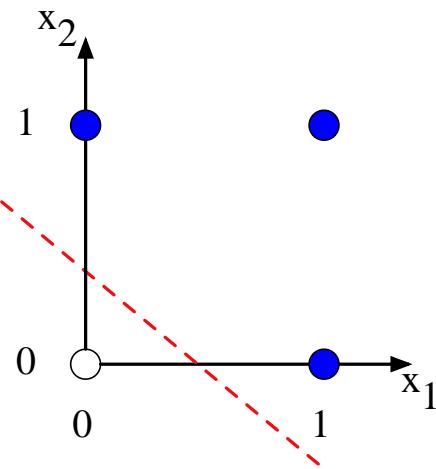
This line acts as a **decision boundary**

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

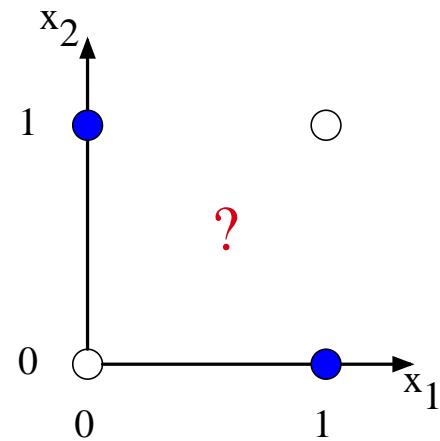
Decision boundaries



a) x_1 AND x_2



b) x_1 OR x_2



c) x_1 XOR x_2

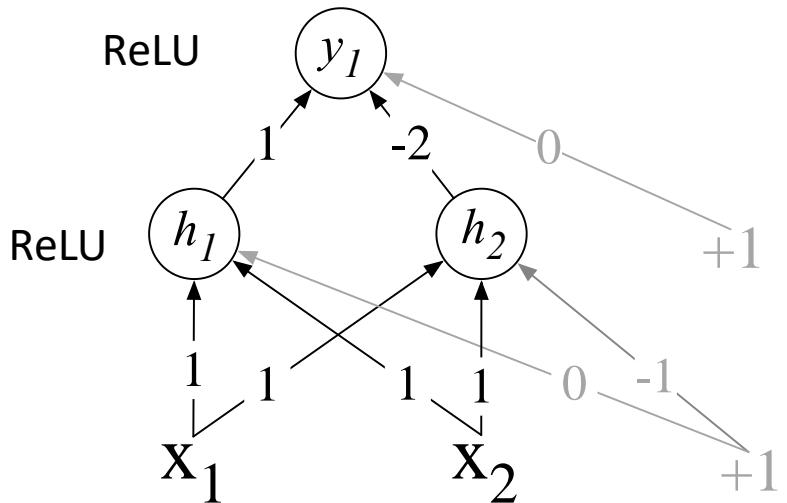
XOR is not a **linearly separable** function!

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

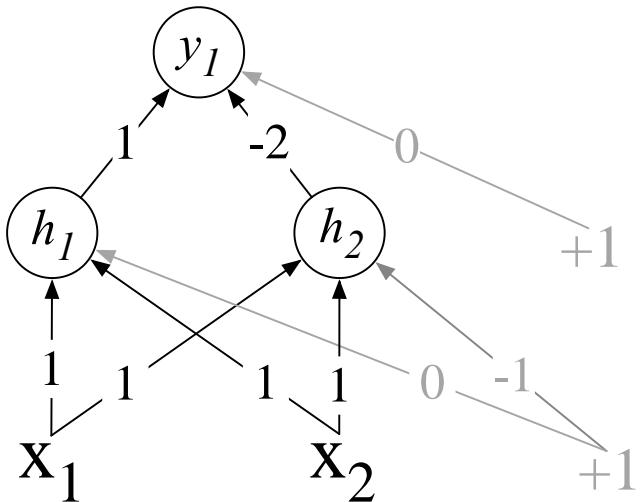


Solution to the XOR problem

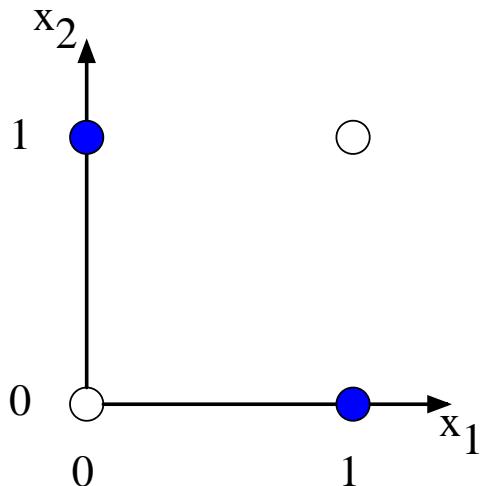
XOR **can't** be calculated by a single perceptron

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

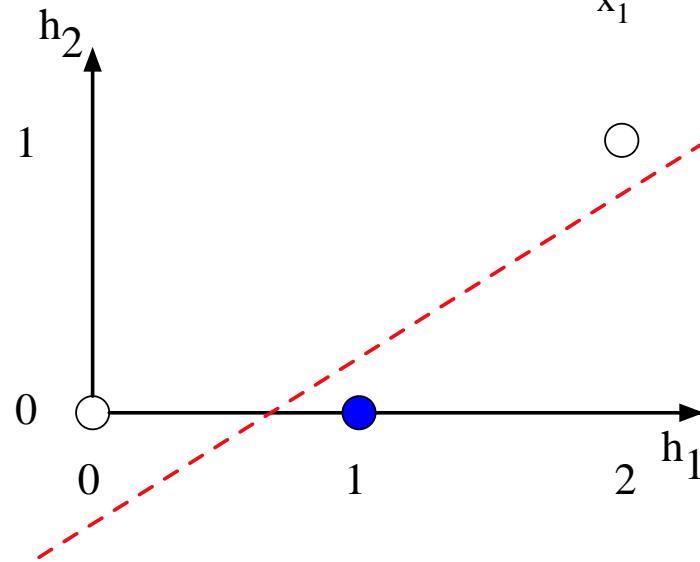
XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0



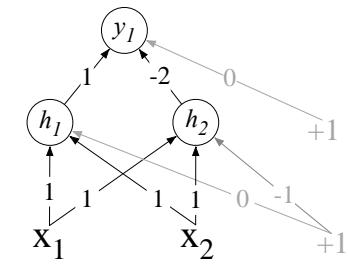
The hidden representation h



a) The original x space



b) The new (linearly separable) h space

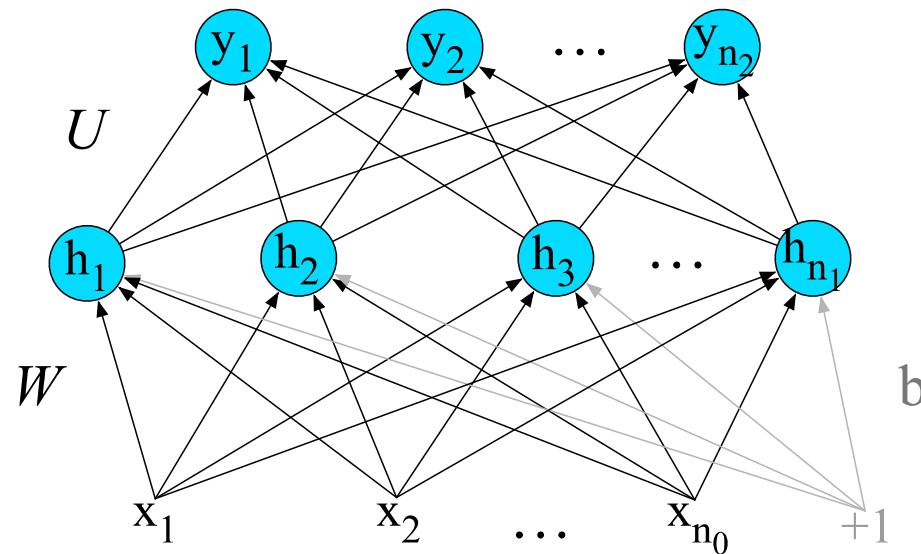


(With learning: hidden layers will learn to form useful representations)

Feedforward Neural Networks

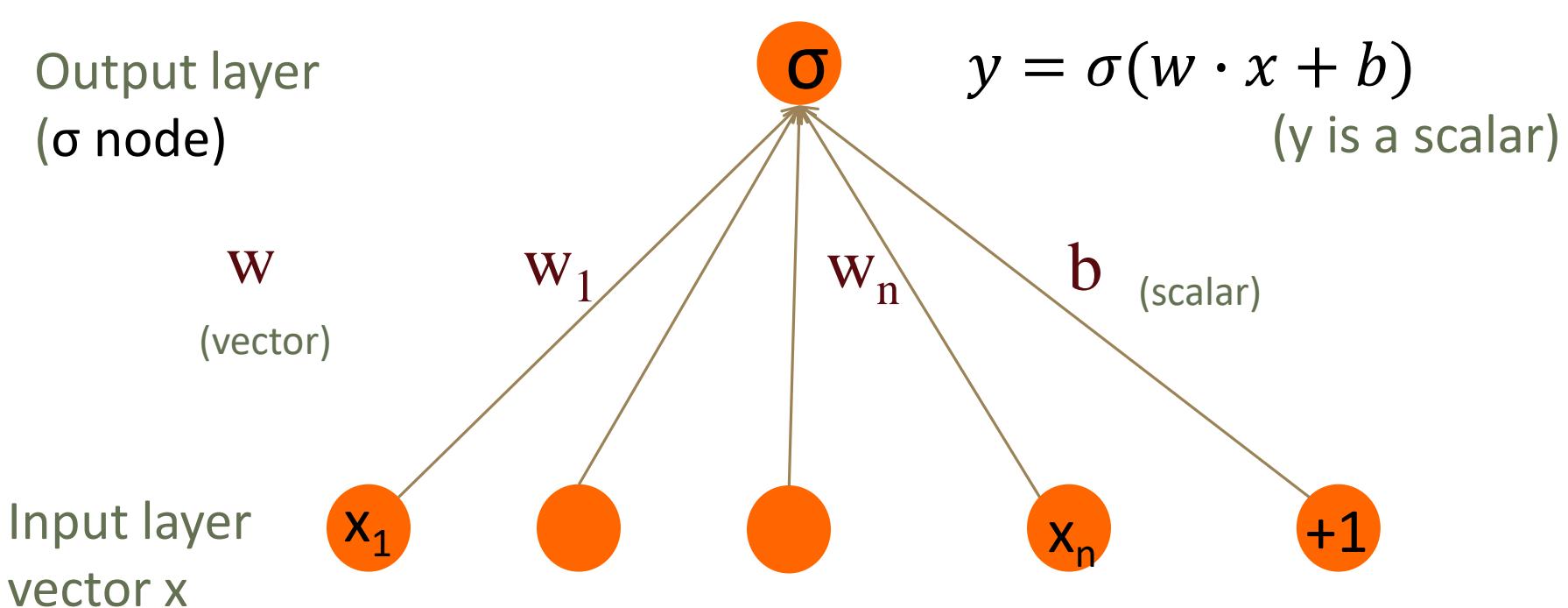
Feedforward Neural Networks

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



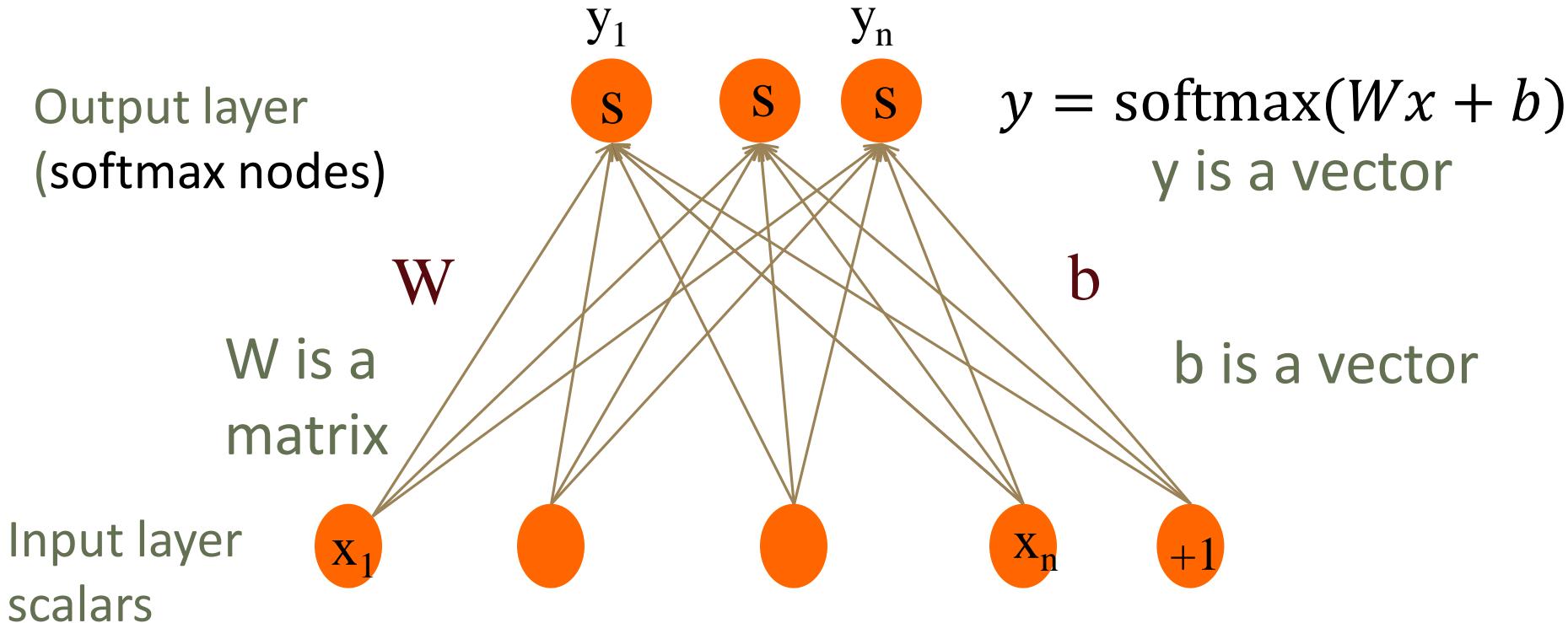
Binary Logistic Regression as a 1-layer Network

(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:

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

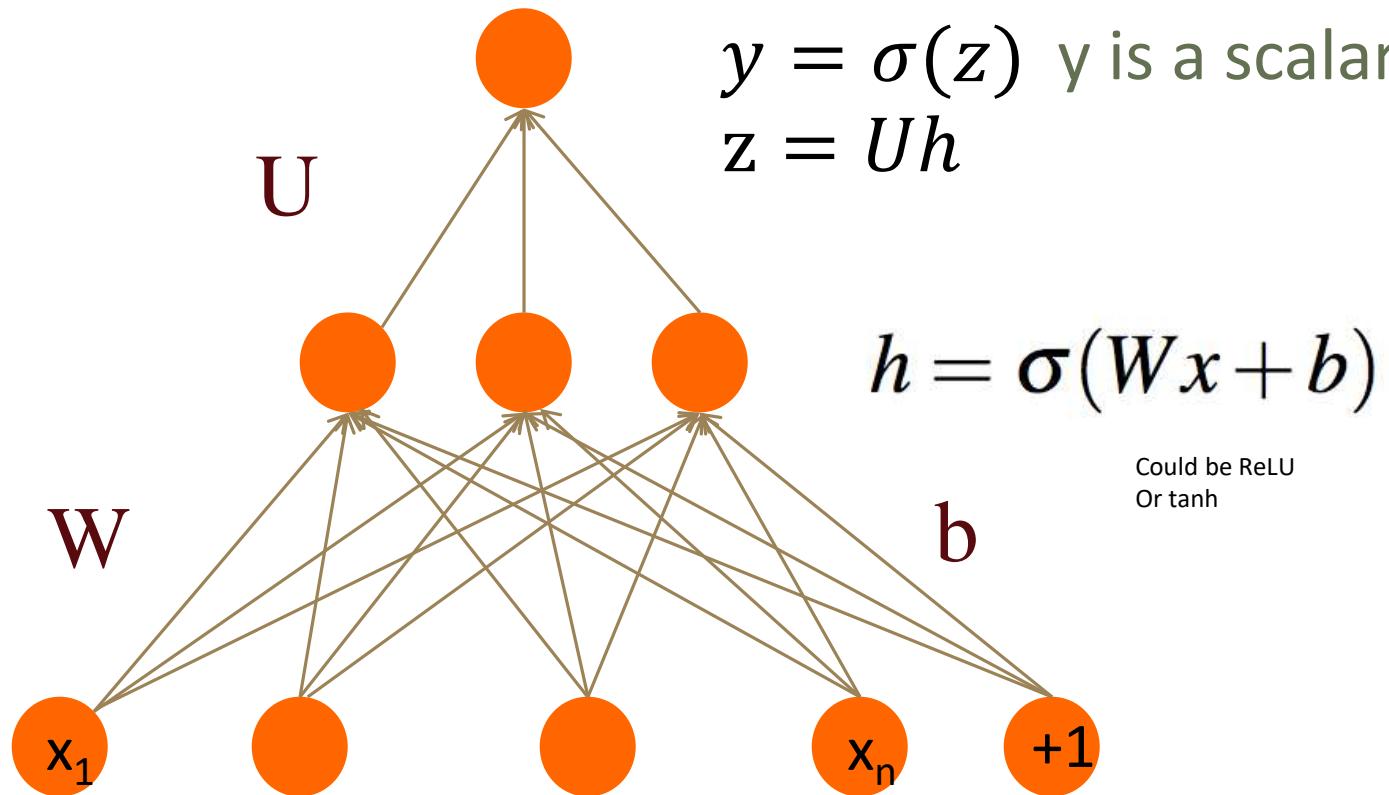
$$\text{softmax}(z) = [0.055, 0.090, 0.006, 0.099, 0.74, 0.010]$$

Two-Layer Network with scalar output

Output layer
(σ node)

hidden units
(σ node)

Input layer
(vector)

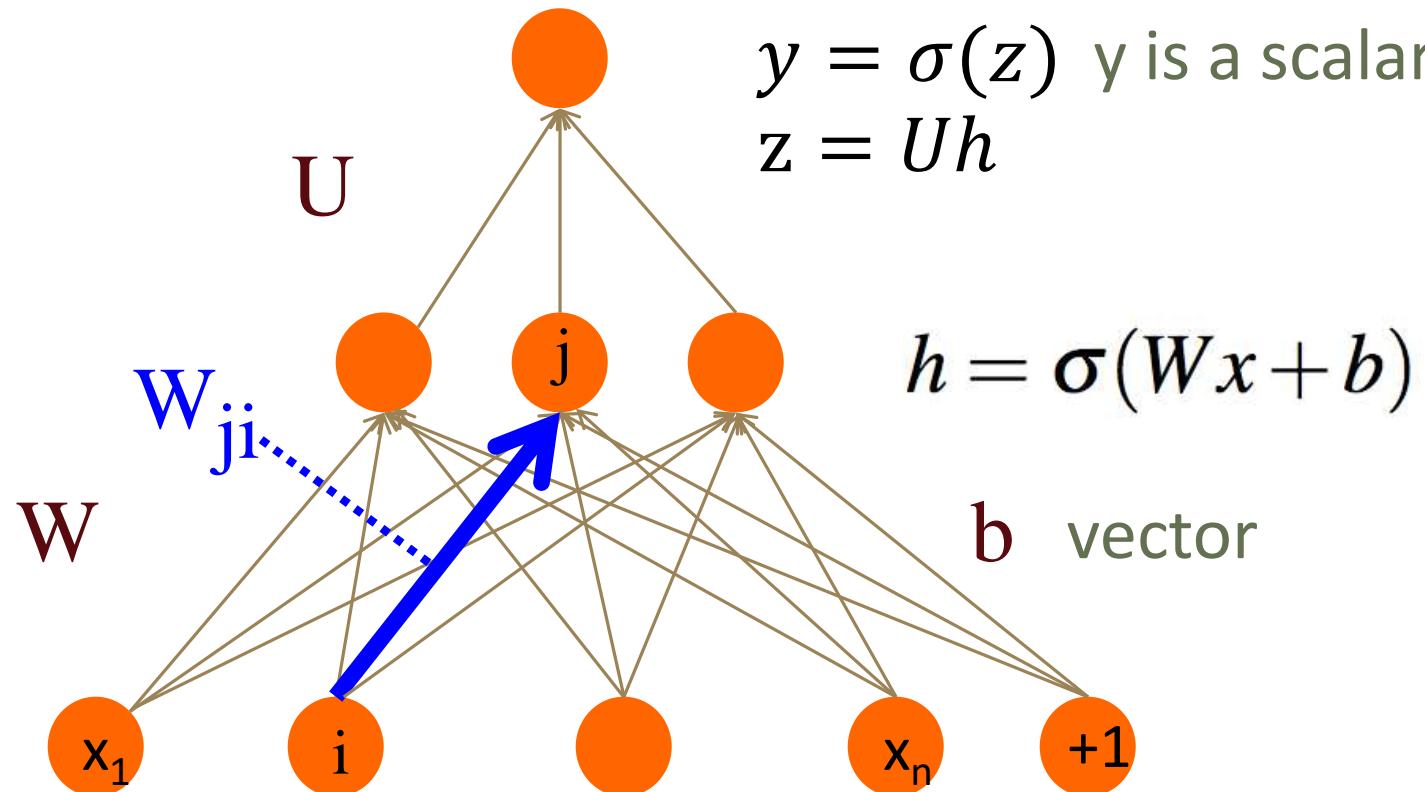


Two-Layer Network with scalar output

Output layer
(σ node)

hidden units
(σ node)

Input layer
(vector)

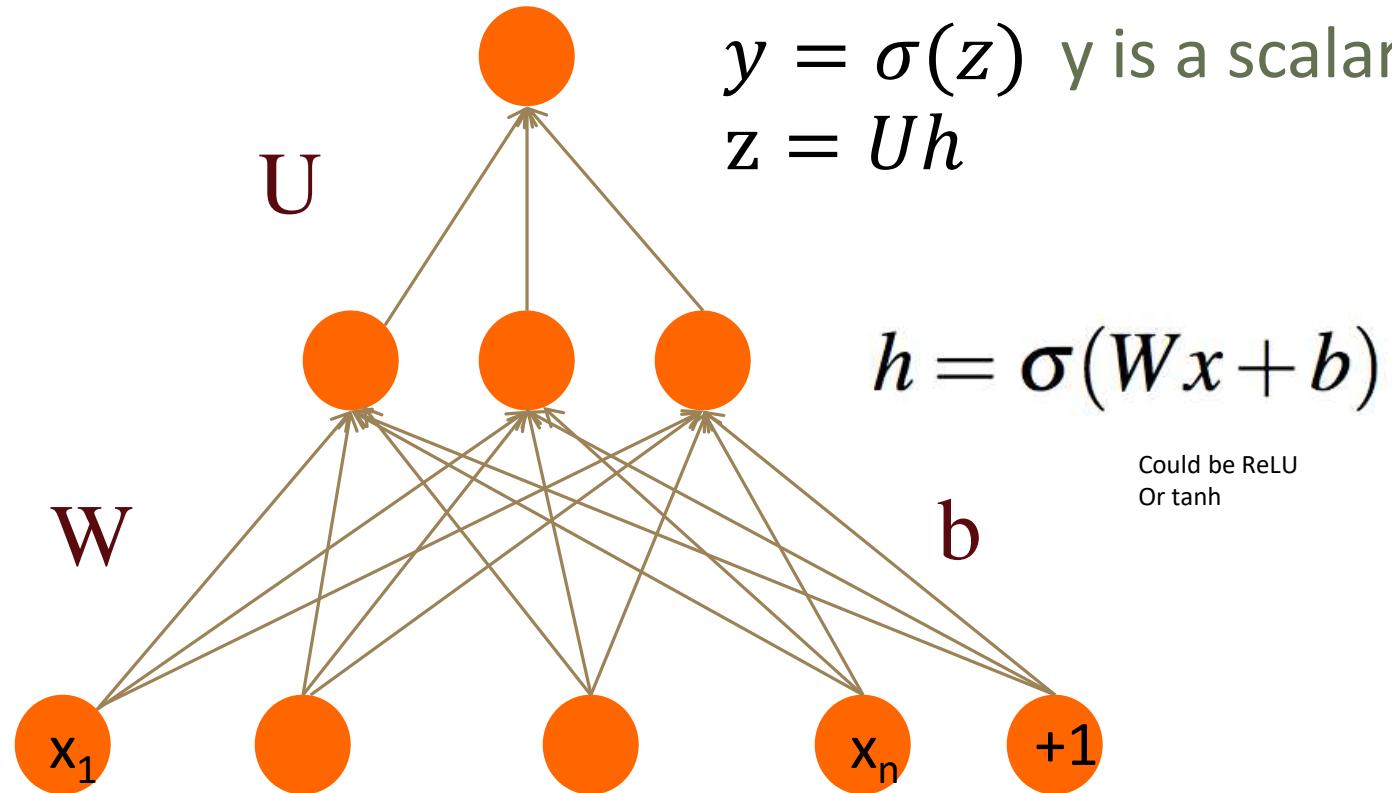


Two-Layer Network with scalar output

Output layer
(σ node)

hidden units
(σ node)

Input layer
(vector)

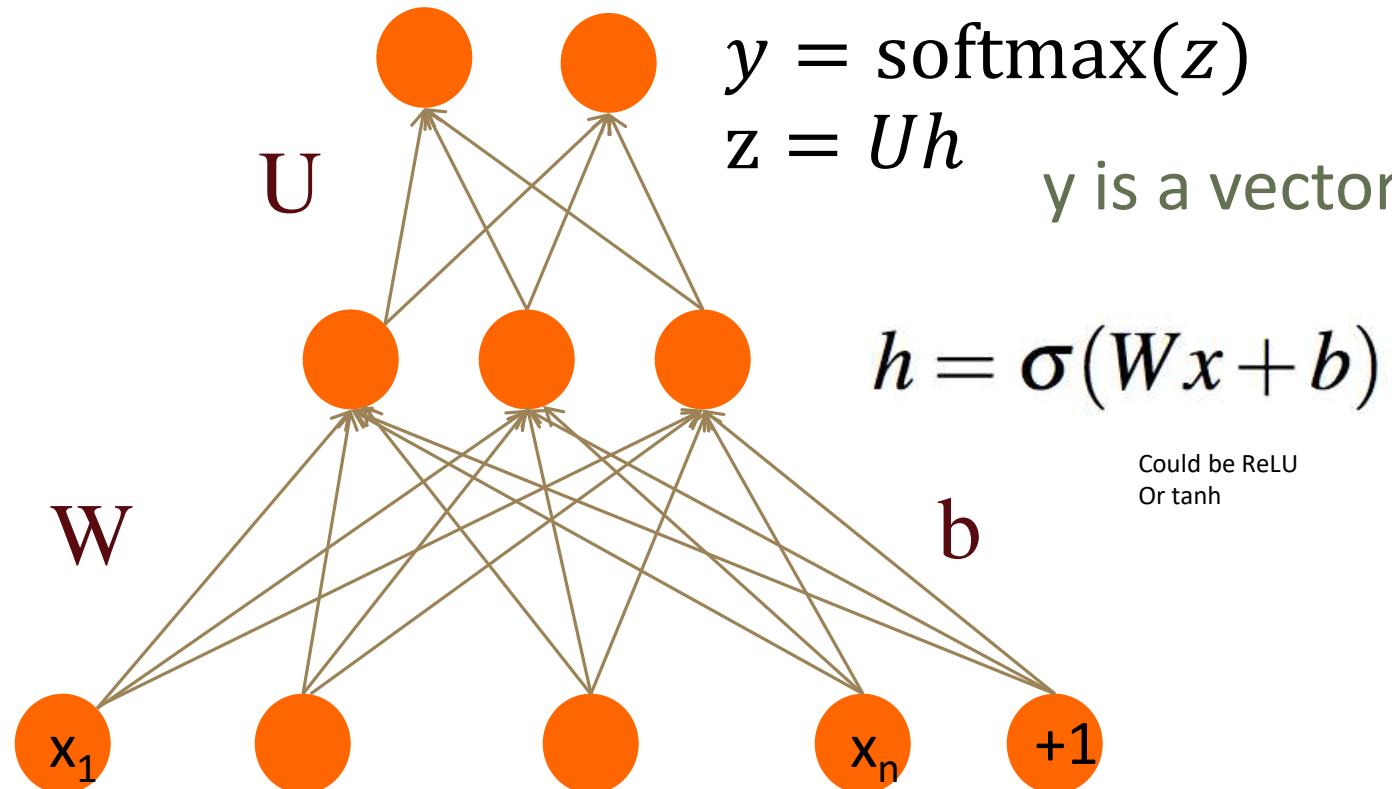


Two-Layer Network with softmax output

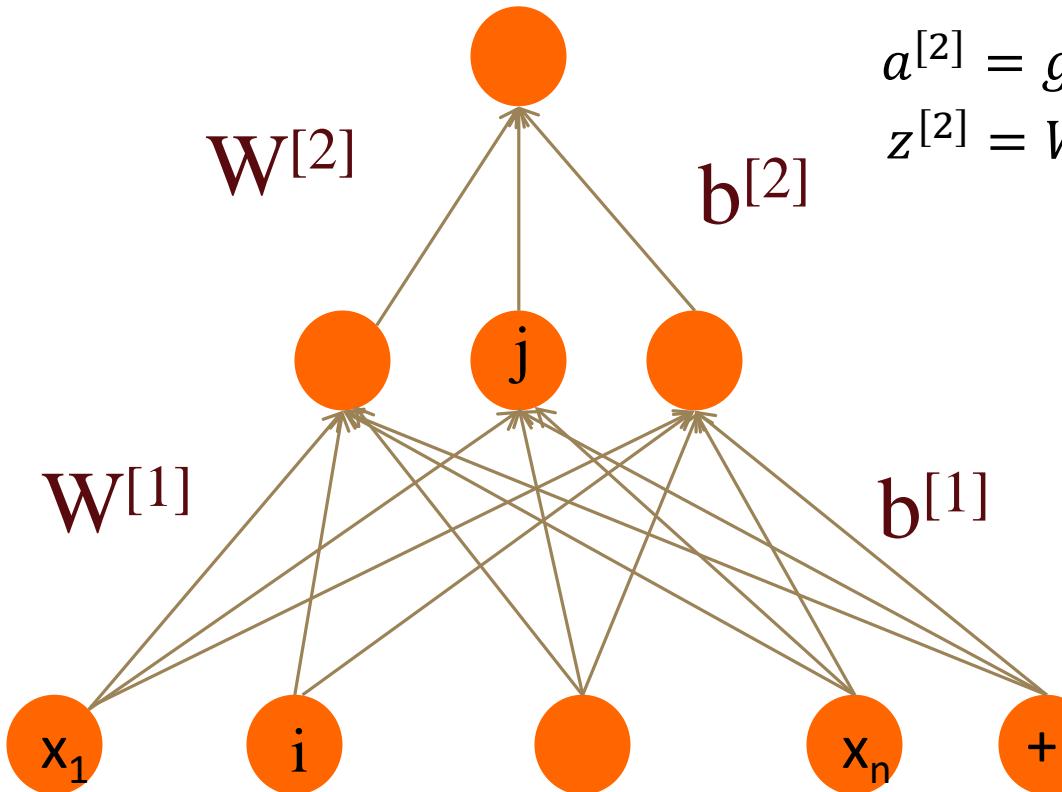
Output layer
(σ node)

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

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

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

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

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

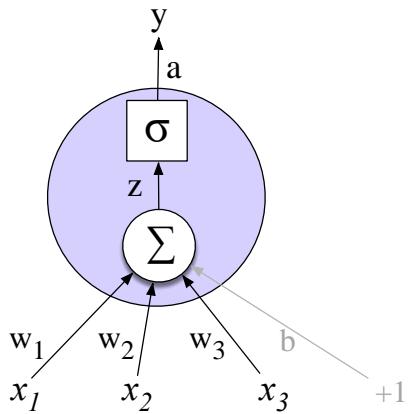
$$\hat{y} = a^{[2]}$$

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:

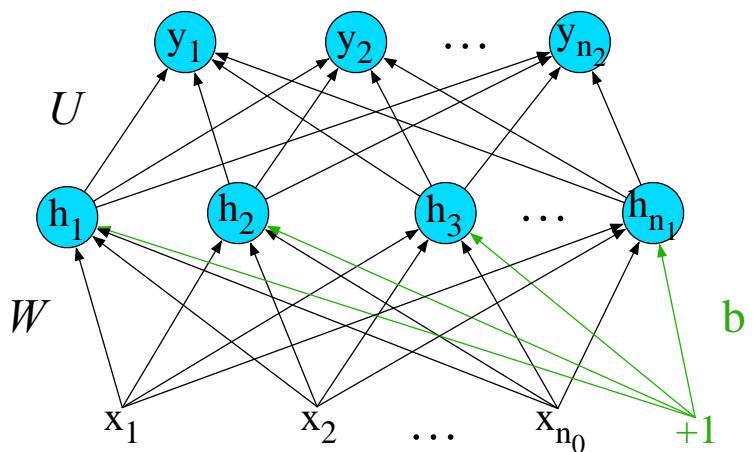
$$x = x_0, x_1, x_2, \dots, x_{n0}$$

$$h = \sigma(Wx)$$

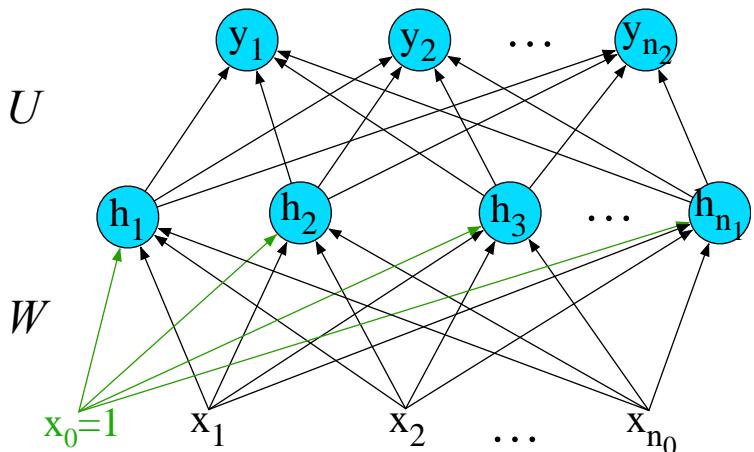
$$\sigma \left(\sum_{i=0}^{n_0} W_{ji} x_i \right)$$

Replacing the bias unit

Instead of:



We'll do this:



Applying feedforward networks to NLP tasks

Use cases for feedforward networks

Let's consider 2 (simplified) sample tasks:

1. Text classification
2. Language modeling

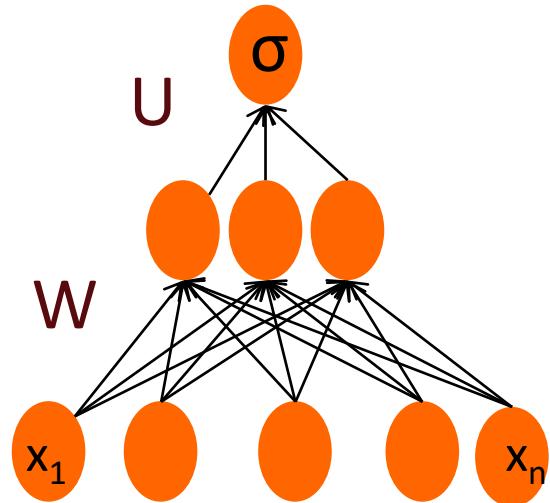
State of the art systems use more powerful neural architectures, but simple models are useful to consider!

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

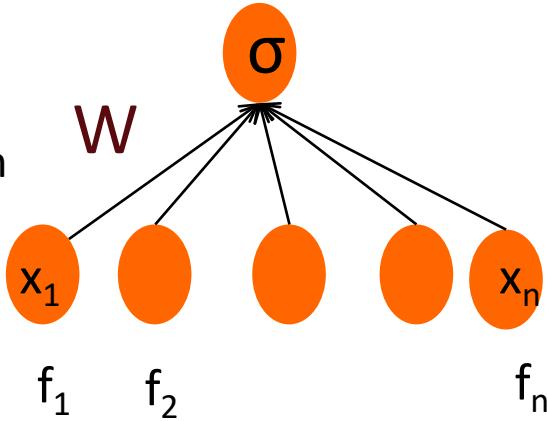


Sentiment Features

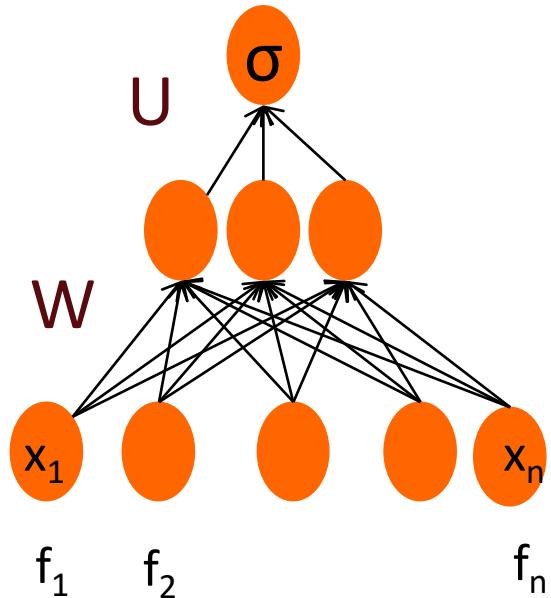
Var	Definition
x_1	count(positive lexicon) \in doc)
x_2	count(negative lexicon) \in doc)
x_3	$\begin{cases} 1 & \text{if “no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_4	count(1st and 2nd pronouns \in doc)
x_5	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$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

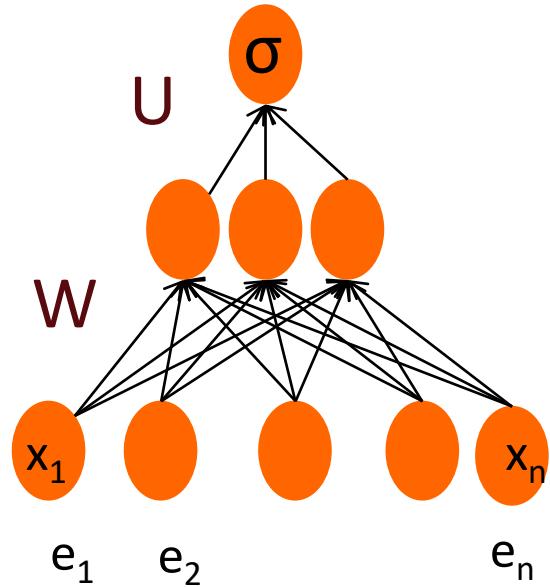
- allows the network to use non-linear interactions between features
- which may (or may not) improve performance

Even better: representation learning

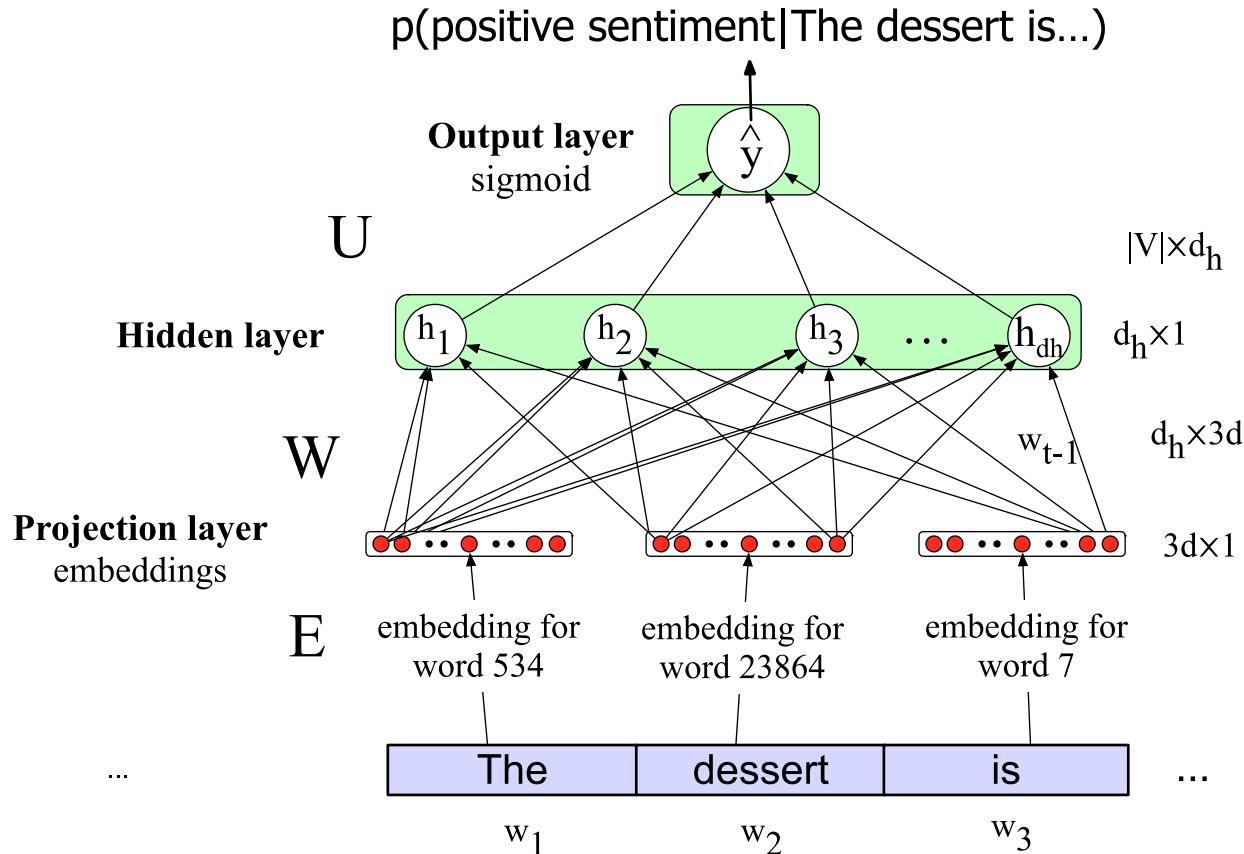
The real power of deep learning comes from the ability to **learn** features from the data

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

Use learned representations like embeddings!



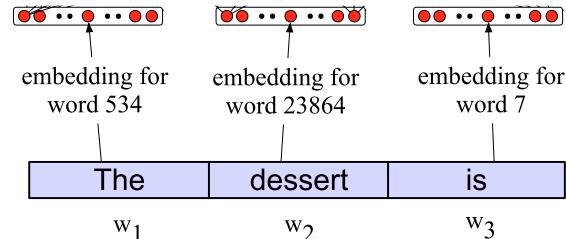
Neural Net Classification with embeddings as input features!



Issue: texts come in different sizes

This assumes a fixed size length (3)!

Kind of unrealistic



Some simple solutions (more sophisticated solutions later)

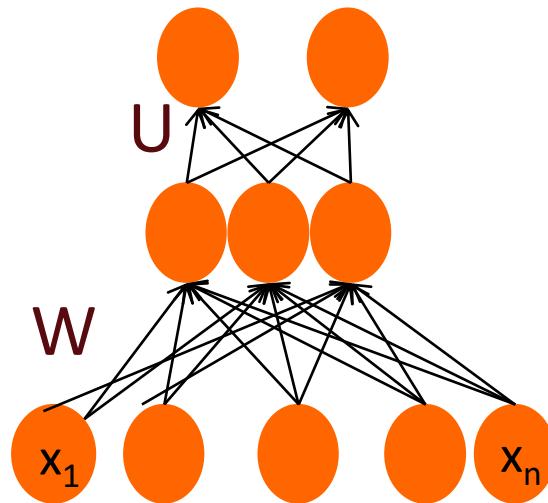
1. Make the input the length of the longest review
 - If shorter then pad with zero embeddings
 - Truncate if you get longer reviews at test time
2. Create a single "sentence embedding" (the same dimensionality as a word) to represent all the words
 - Take the mean of all the word embeddings
 - Take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

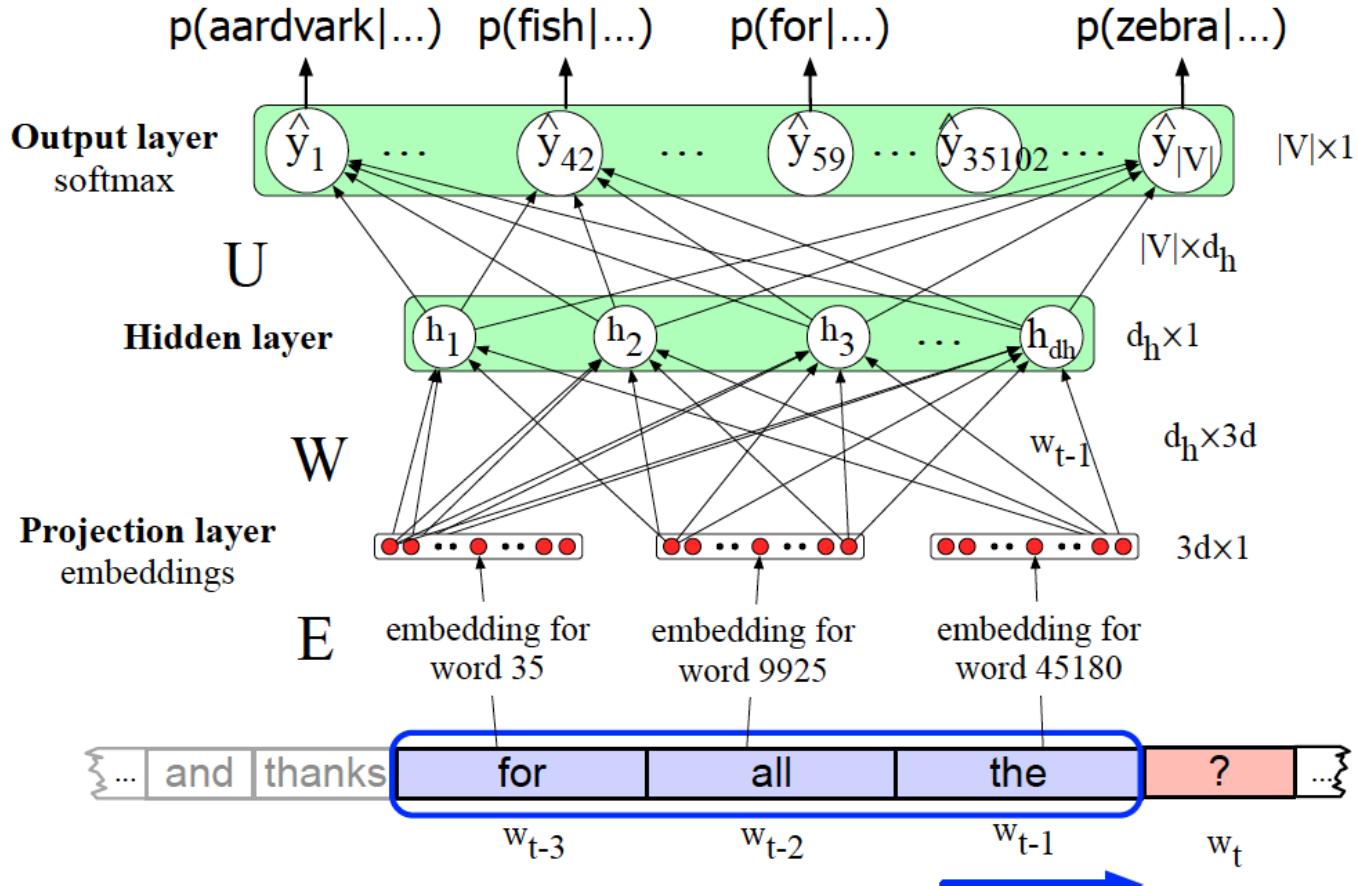
given prior words $w_{t-1}, w_{t-2}, w_{t-3}, \dots$

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:

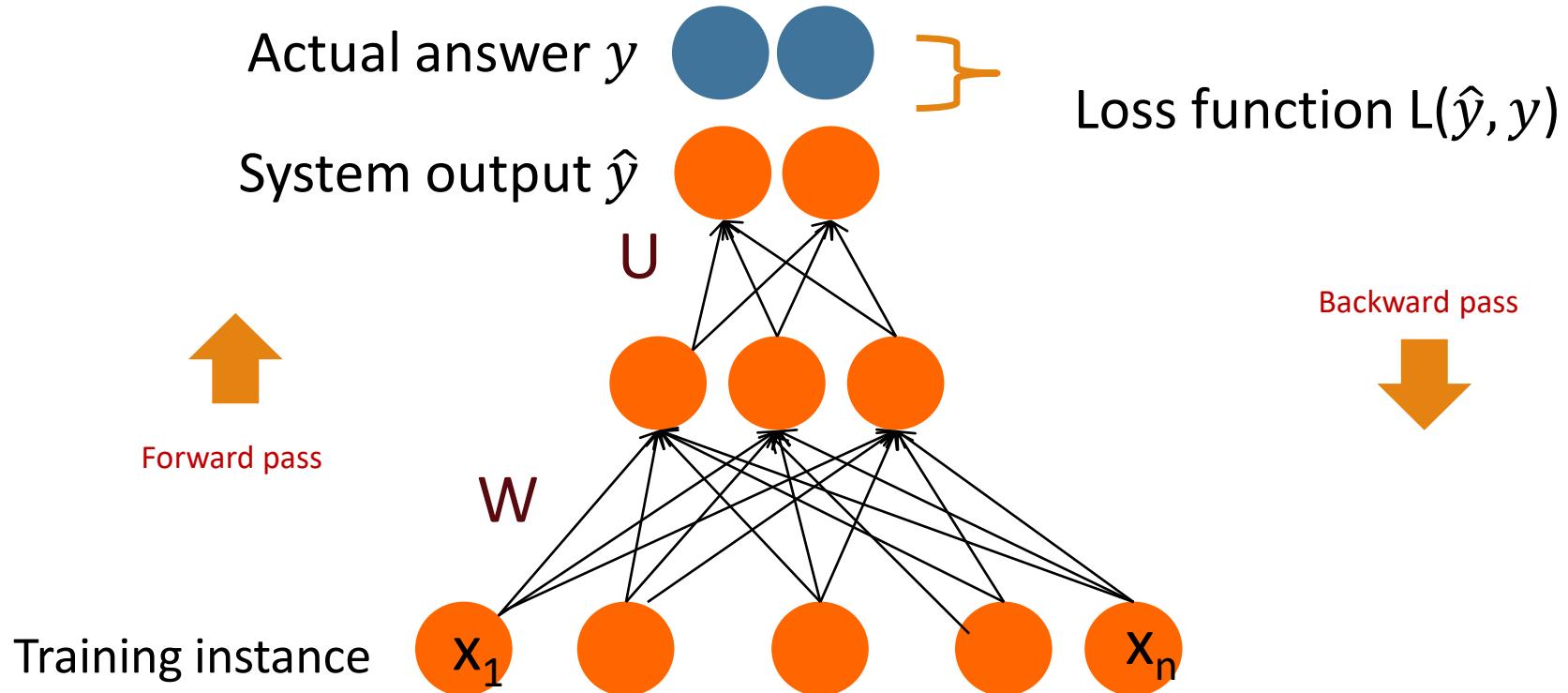
I forgot to make sure that the dog gets __

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

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

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

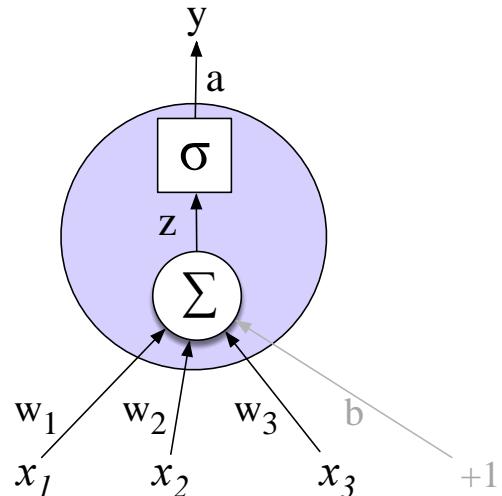
$$w^{t+1} = w^t - h \frac{d}{dw} L(f(x; w), y)$$

- 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!

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

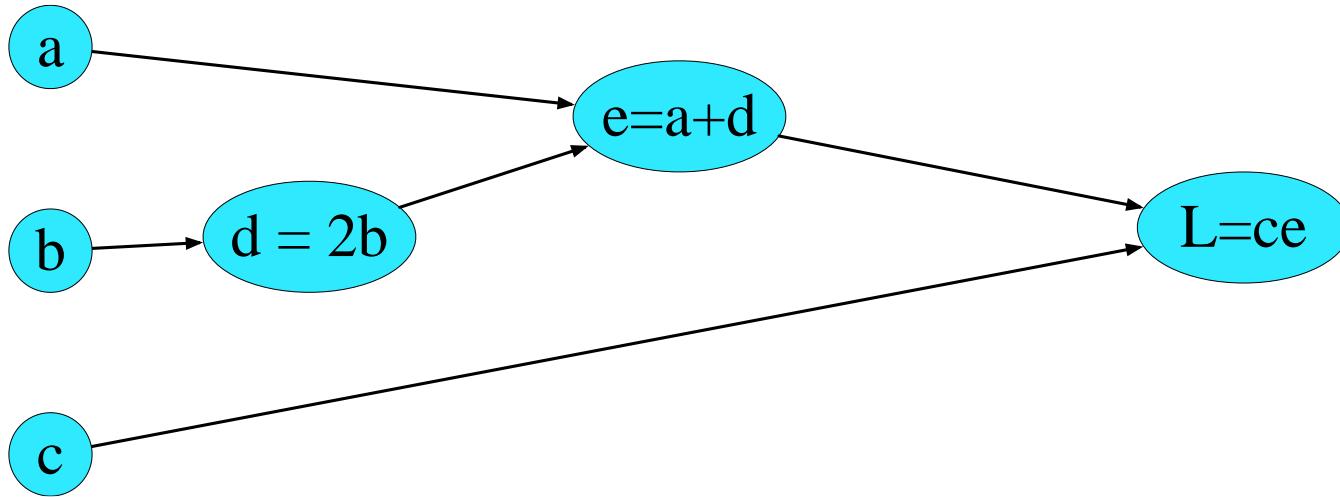
Example: $L(a, b, c) = c(a + 2b)$

$$d = 2 * b$$

Computations:

$$e = a + d$$

$$L = c * e$$



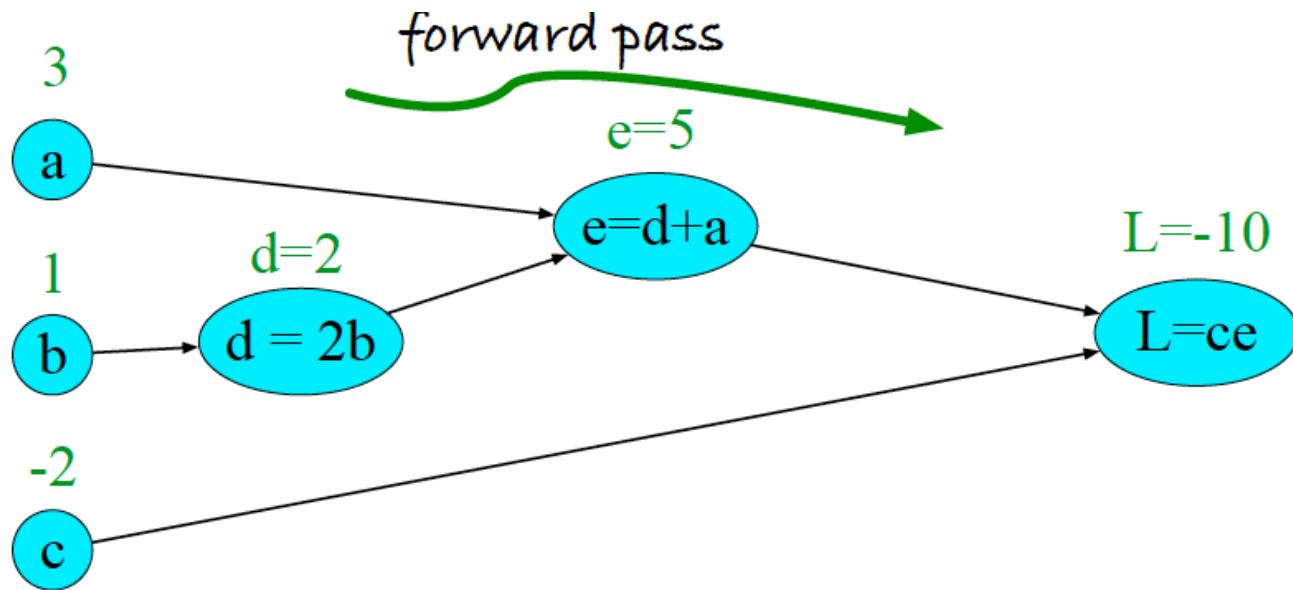
Example: $L(a, b, c) = c(a + 2b)$

$$d = 2 * b$$

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

$$a=3$$



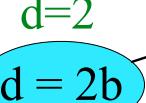
$$b=1$$



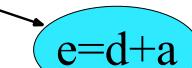
$$c=-2$$



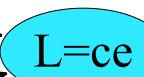
$$d = 2b$$



$$e=5$$



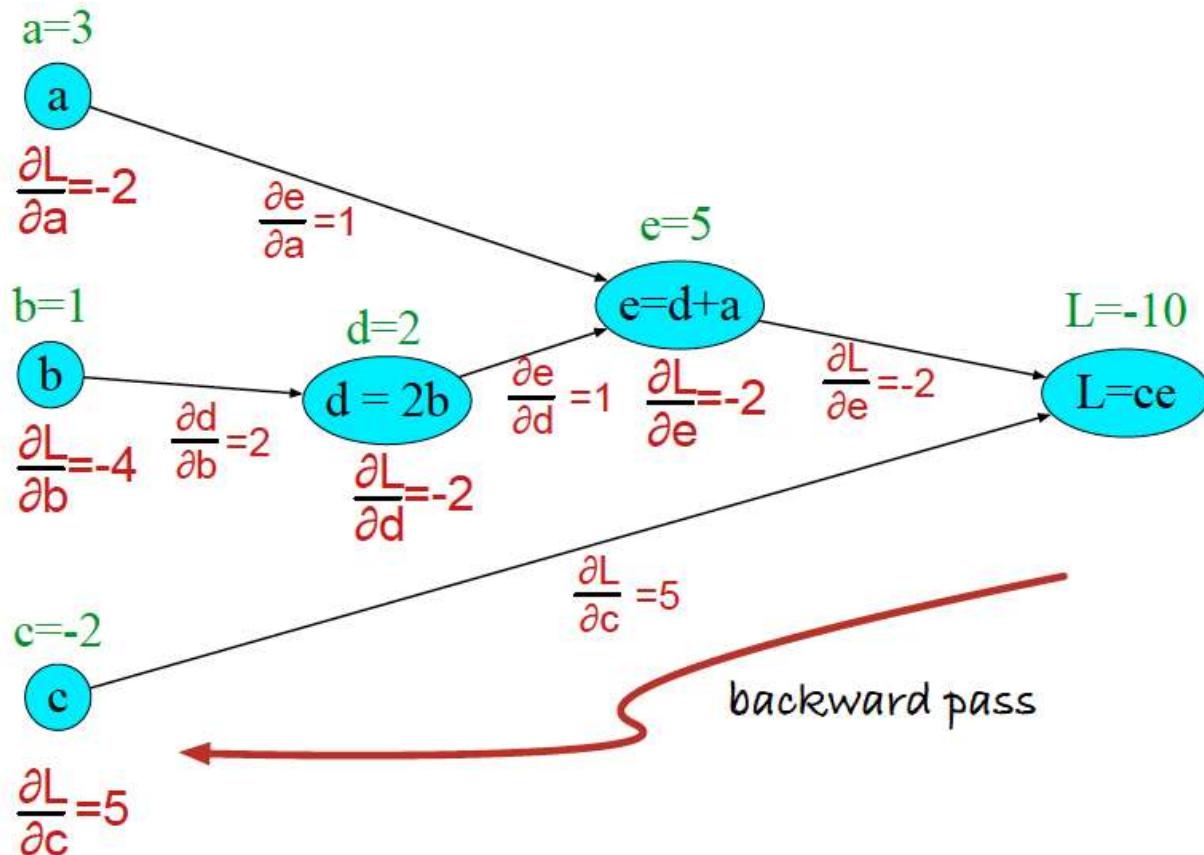
$$L=ce$$



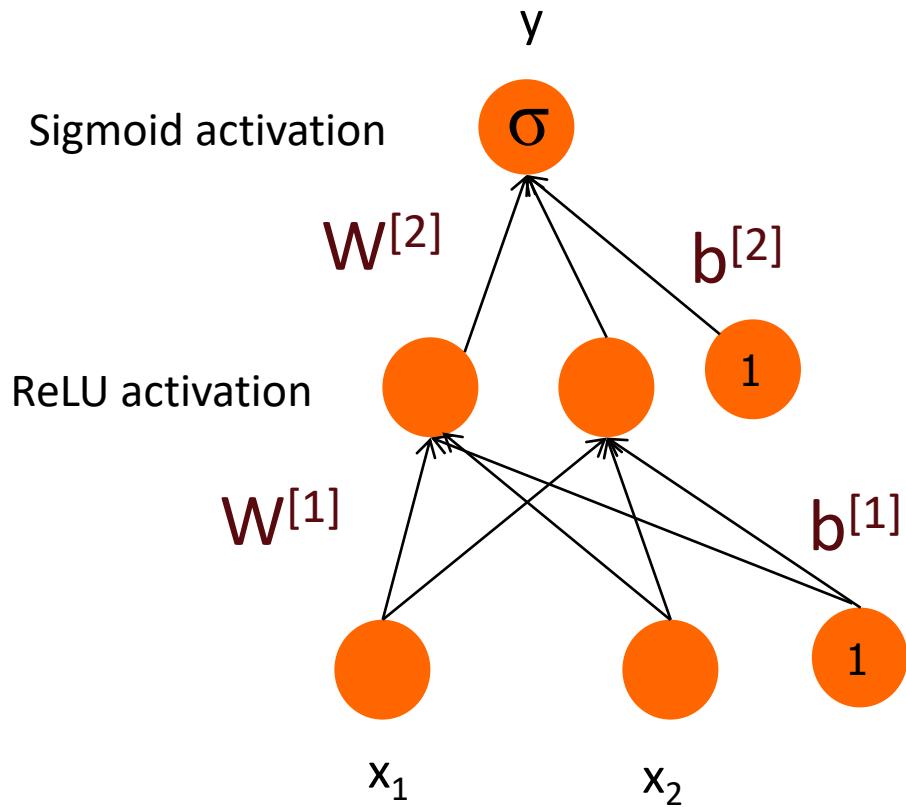
$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e} \frac{\partial e}{\partial a}$$
$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e} \frac{\partial e}{\partial d} \frac{\partial d}{\partial b}$$

$$L = ce : \quad \frac{\partial L}{\partial e} = c, \frac{\partial L}{\partial c} = e$$
$$e = a + d : \quad \frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial d} = 1$$
$$d = 2b : \quad \frac{\partial d}{\partial b} = 2$$

Example



Backward differentiation on a two layer network



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

Backward differentiation on a two layer network

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

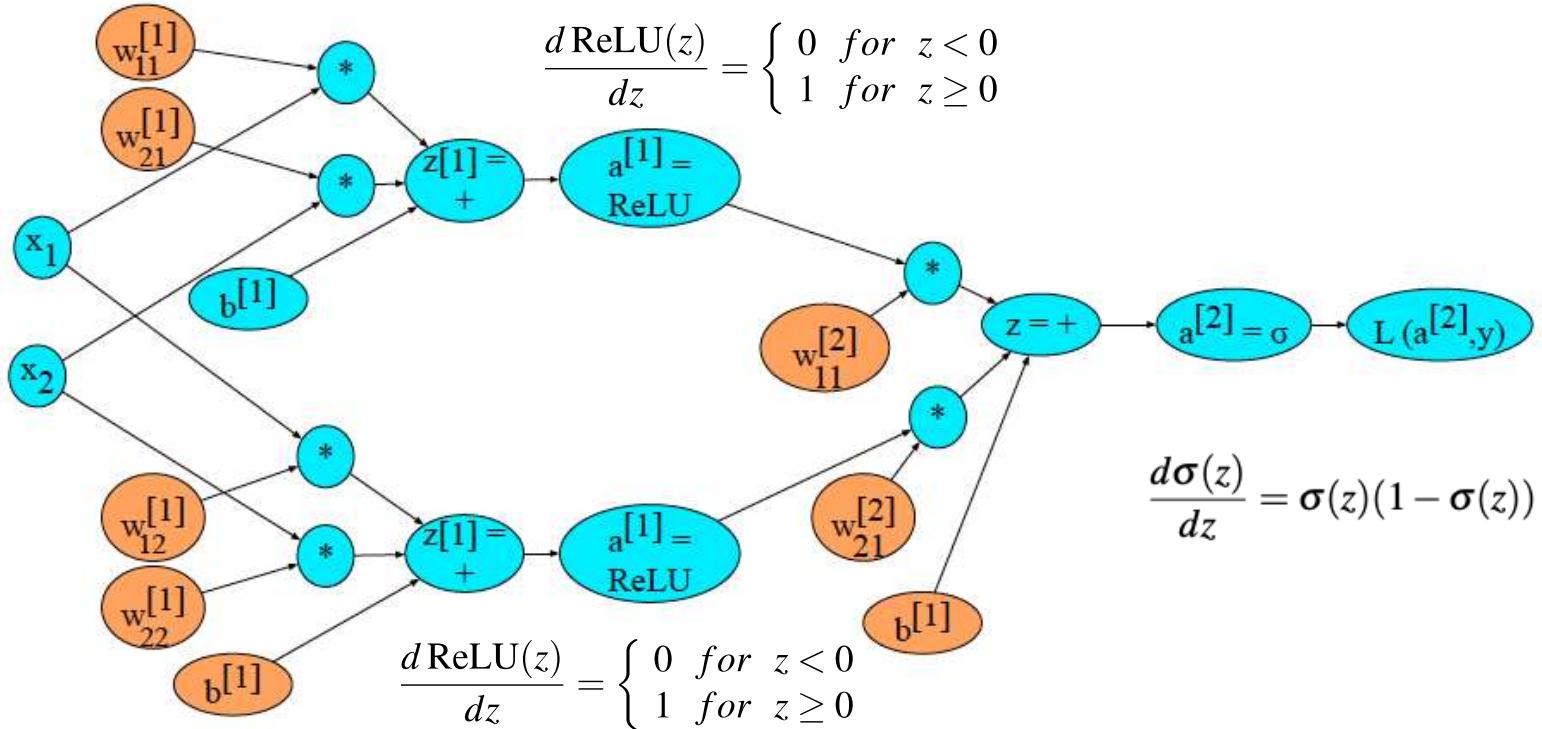
$$a^{[1]} = \text{ReLU}(z^{[1]}) \quad \frac{d \text{ReLU}(z)}{dz} = \begin{cases} 0 & \text{for } z < 0 \\ 1 & \text{for } z \geq 0 \end{cases}$$

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

$$a^{[2]} = \sigma(z^{[2]}) \quad \frac{d\sigma(z)}{dz} = \sigma(z)(1 - \sigma(z))$$

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

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

Thanks for Your Attention!

Chap. 8: Sequence Labeling for Parts of Speech and Named Entities

Outline

Part of Speech Tagging

Named Entity Recognition (NER)

Part of Speech Tagging

Parts of Speech

From the earliest linguistic traditions (Yaska and Panini 5th C. BCE, Aristotle 4th C. BCE), the idea that words can be classified into grammatical categories

- part of speech, word classes, POS, POS tags

8 parts of speech attributed to Dionysius Thrax of Alexandria (c. 1st C. BCE):

- noun, verb, pronoun, preposition, adverb, conjunction, participle, article
- These categories are relevant for NLP today

Two classes of words: Open vs. Closed

Closed class words

- Relatively fixed membership
- Usually **function** words: short, frequent words with grammatical function
 - determiners: *a, an, the*
 - pronouns: *she, he, I*
 - prepositions: *on, under, over, near, by, ...*

Open class words

- Usually **content** words: Nouns, Verbs, Adjectives, Adverbs
 - Plus interjections: *oh, ouch, uh-huh, yes, hello*
 - New nouns and verbs like *iPhone* or *to fax*

Open class ("content") words

Nouns

Proper

Janet

Italy

Common

cat, cats

mango

Verbs

Main

eat

went

Adjectives

old green tasty

Adverbs

slowly yesterday

Numbers

122,312

one

Interjections

Ow hello

... more

Closed class ("function")

Determiners

the some

Conjunctions

and or

Pronouns

they its

Auxiliary

can

had

Prepositions

to with

Particles

off up

... more

Part-of-Speech Tagging

Assigning a part-of-speech to each word in a text

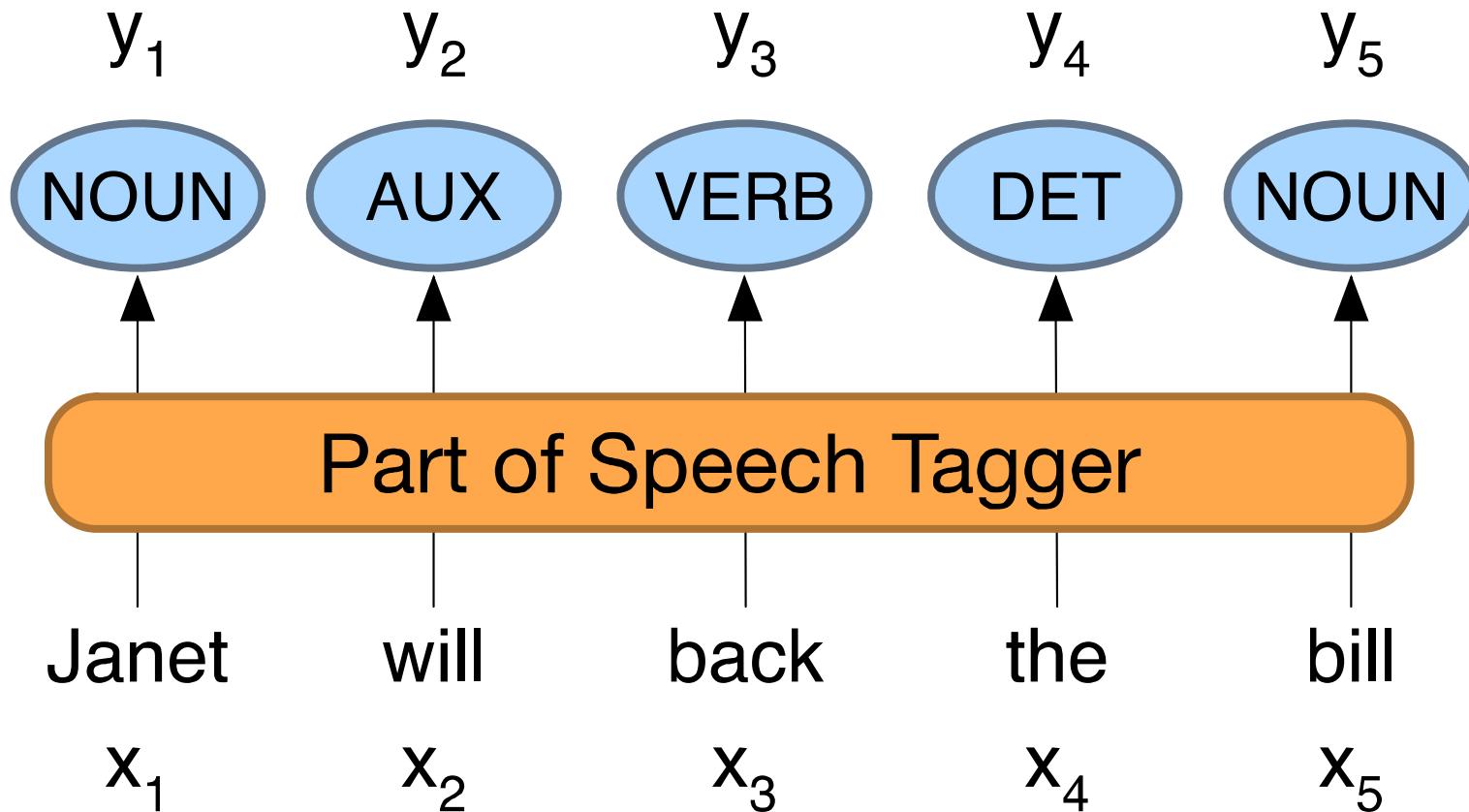
Words often have more than one POS

book:

- VERB: (*Book that flight*)
- NOUN: (*Hand me that book*)

Part-of-Speech Tagging

Map from sequence x_1, \dots, x_n of words to y_1, \dots, y_n of POS tags



"Universal Dependencies" Tagset

Nivre et al. 2016

Tag	Description	Example
Open Class	ADJ Adjective: noun modifiers describing properties	<i>red, young, awesome</i>
	ADV Adverb: verb modifiers of time, place, manner	<i>very, slowly, home, yesterday</i>
	NOUN words for persons, places, things, etc.	<i>algorithm, cat, mango, beauty</i>
	VERB words for actions and processes	<i>draw, provide, go</i>
	PROPN Proper noun: name of a person, organization, place, etc..	<i>Regina, IBM, Colorado</i>
	INTJ Interjection: exclamation, greeting, yes/no response, etc.	<i>oh, um, yes, hello</i>
Closed Class Words	ADP Adposition (Preposition/Postposition): marks a noun's spacial, temporal, or other relation	<i>in, on, by under</i>
	AUX Auxiliary: helping verb marking tense, aspect, mood, etc.,	<i>can, may, should, are</i>
	CCONJ Coordinating Conjunction: joins two phrases/clauses	<i>and, or, but</i>
	DET Determiner: marks noun phrase properties	<i>a, an, the, this</i>
	NUM Numeral	<i>one, two, first, second</i>
	PART Particle: a preposition-like form used together with a verb	<i>up, down, on, off, in, out, at, by</i>
	PRON Pronoun: a shorthand for referring to an entity or event	<i>she, who, I, others</i>
	SCONJ Subordinating Conjunction: joins a main clause with a subordinate clause such as a sentential complement	<i>that, which</i>
Other	PUNCT Punctuation	<i>; , ()</i>
	SYM Symbols like \$ or emoji	<i>\$, %</i>
	X Other	<i>asdf, qwfg</i>

Sample "Tagged" English sentences

There/PRO were/VERB 70/NUM children/NOUN
there/ADV ./PUNC

Preliminary/ADJ findings/NOUN were/AUX
reported/VERB in/ADP today/NOUN 's/PART
New/PROPN England/PROPN Journal/PROPN
of/ADP Medicine/PROPN

Why Part of Speech Tagging?

- Can be useful for other NLP tasks
 - Parsing: POS tagging can improve syntactic parsing
 - MT: reordering of adjectives and nouns (say from Spanish to English)
 - Sentiment or affective tasks: may want to distinguish adjectives or other POS
 - Text-to-speech (how do we pronounce “lead” or "object"?)
- Or linguistic or language-analytic computational tasks
 - Need to control for POS when studying linguistic change like creation of new words, or meaning shift
 - Or control for POS in measuring meaning similarity or difference

How difficult is POS tagging in English?

Roughly 15% of word types are ambiguous

- Hence 85% of word types are unambiguous
- *Janet* is always PROPN, *hesitantly* is always ADV

But those 15% tend to be very common

So ~60% of word tokens are ambiguous

E.g., *back*

earnings growth took a **back**/ADJ seat

a small building in the **back**/NOUN

a clear majority of senators **back**/VERB the bill

enable the country to buy **back**/PART debt

I was twenty-one **back**/ADV then

POS tagging performance in English

How many tags are correct? (Tag accuracy)

- About 97%
 - Hasn't changed in the last 10+ years
 - HMMs, CRFs, BERT perform similarly
 - Human accuracy about the same

But baseline is 92%!

- Baseline is performance of stupidest possible method
 - "Most frequent class baseline" is an important baseline for many tasks
 - Tag every word with its most frequent tag
 - (and tag unknown words as nouns)
- Partly easy because
 - Many words are unambiguous

Sources of information for POS tagging

Janet will back the bill
AUX/NOUN/VERB? NOUN/VERB?

Prior probabilities of word/tag

- "will" is usually an AUX

Identity of neighboring words

- "the" means the next word is probably not a verb

Morphology and wordshape:

- Prefixes unable: un- → ADJ
- Suffixes importantly: -ly → ADJ
- Capitalization Janet: CAP → PROPN

Standard algorithms for POS tagging

Supervised Machine Learning Algorithms:

- Hidden Markov Models
- Conditional Random Fields (CRF)/ Maximum Entropy Markov Models (MEMM)
- Neural sequence models (RNNs or Transformers)
- Large Language Models (like BERT), finetuned

All required a hand-labeled training set, all about equal performance (97% on English)

All make use of information sources we discussed

- Via human created features: HMMs and CRFs
- Via representation learning: Neural LMs

Named Entity Recognition (NER)

Named Entities

- **Named entity**, in its core usage, means anything that can be referred to with a proper name. Most common 4 tags:
 - **PER** (Person): “[Marie Curie](#)”
 - **LOC** (Location): “[New York City](#)”
 - **ORG** (Organization): “[Stanford University](#)”
 - **GPE** (Geo-Political Entity): “[Boulder, Colorado](#)”
- Often multi-word phrases
- But the term is also extended to things that aren't entities:
 - dates, times, prices

Named Entity tagging

The task of named entity recognition (NER):

- find spans of text that constitute proper names
- tag the type of the entity

NER output

Citing high fuel prices, [ORG United Airlines] said [TIME Friday] it has increased fares by [MONEY \$6] per round trip on flights to some cities also served by lower-cost carriers. [ORG American Airlines], a unit of [ORG AMR Corp.], immediately matched the move, spokesman [PER Tim Wagner] said. [ORG United], a unit of [ORG UAL Corp.], said the increase took effect [TIME Thursday] and applies to most routes where it competes against discount carriers, such as [LOC Chicago] to [LOC Dallas] and [LOC Denver] to [LOC San Francisco].

Why NER?

Sentiment analysis: consumer's sentiment toward a particular company or person?

Question Answering: answer questions about an entity?

Information Extraction: Extracting facts about entities from text

Why NER is hard

1) Segmentation

- In POS tagging, no segmentation problem since each word gets one tag
- In NER we have to find and segment the entities!

2) Type ambiguity

[PER Washington] was born into slavery on the farm of James Burroughs.

[ORG Washington] went up 2 games to 1 in the four-game series.

Blair arrived in [LOC Washington] for what may well be his last state visit.

In June, [GPE Washington] passed a primary seatbelt law.

BIO Tagging

How can we turn this structured problem into a sequence problem like POS tagging, with one label per word?

[PER Jane Villanueva] of [ORG United] , a unit of [ORG United Airlines Holding] , said the fare applies to the [LOC Chicago] route.

BIO Tagging

[PER Jane Villanueva] of [ORG United] , a unit of [ORG United Airlines Holding] , said the fare applies to the [LOC Chicago] route.

Words	BIO Label
Jane	B-PER
Villanueva	I-PER
of	O
United	B-ORG
Airlines	I-ORG
Holding	I-ORG
discussed	O
the	O
Chicago	B-LOC
route	O
.	O

Now we have one tag per token!!!

BIO Tagging

B: token that *begins* a span

I: tokens *inside* a span

O: tokens outside of any span

of tags (where n is #entity types):

1 O tag,

n B tags,

n I tags

total of $2n+1$

Words	BIO Label
Jane	B-PER
Villanueva	I-PER
of	O
United	B-ORG
Airlines	I-ORG
Holding	I-ORG
discussed	O
the	O
Chicago	B-LOC
route	O
.	O

BIO Tagging variants: IO and BIOES

[PER Jane Villanueva] of [ORG United] , a unit of [ORG United Airlines Holding] , said the fare applies to the [LOC Chicago] route.

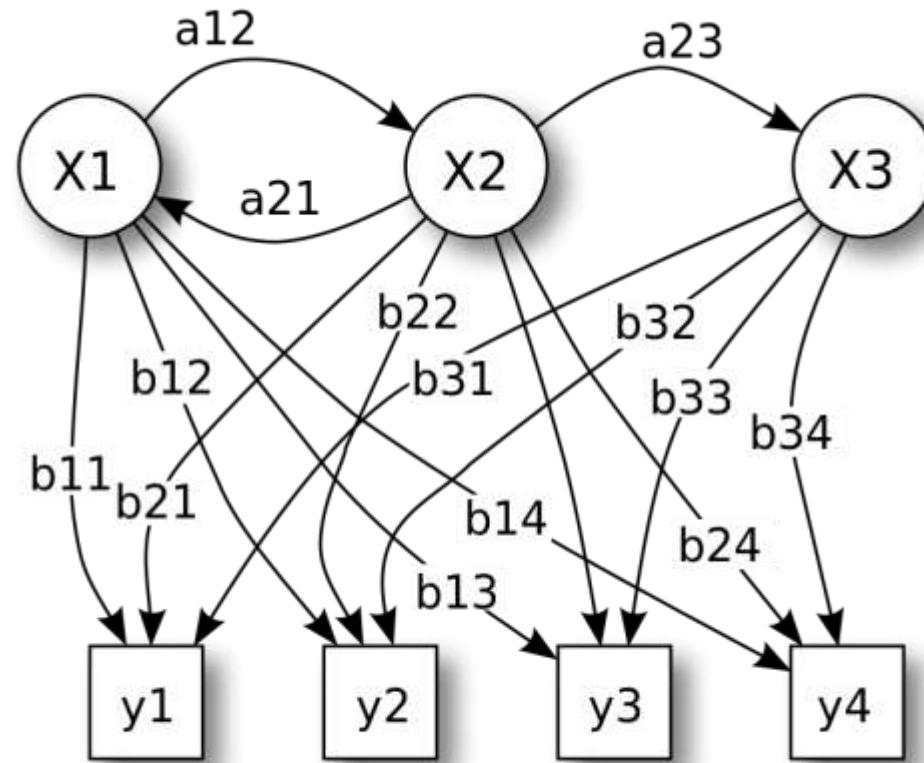
Words	IO Label	BIO Label	BIOES Label
Jane	I-PER	B-PER	B-PER
Villanueva	I-PER	I-PER	E-PER
of	O	O	O
United	I-ORG	B-ORG	B-ORG
Airlines	I-ORG	I-ORG	I-ORG
Holding	I-ORG	I-ORG	E-ORG
discussed	O	O	O
the	O	O	O
Chicago	I-LOC	B-LOC	S-LOC
route	O	O	O
.	O	O	O

Standard algorithms for NER

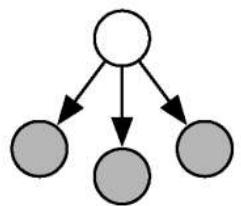
Supervised Machine Learning given a human-labeled training set of text annotated with tags

- Hidden Markov Models
- Conditional Random Fields (CRF)/ Maximum Entropy Markov Models (MEMM)
- Neural sequence models (RNNs or Transformers)
- Large Language Models (like BERT), finetuned

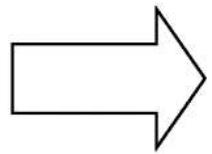
Hidden Markov Model



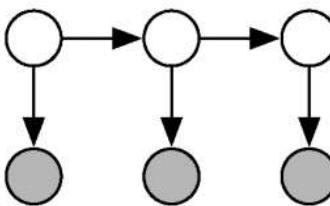
Conditional Random Fields



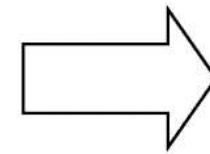
Naive Bayes



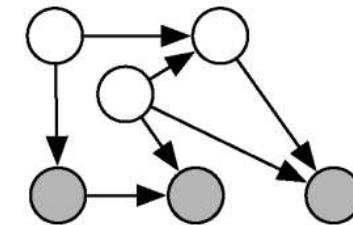
SEQUENCE



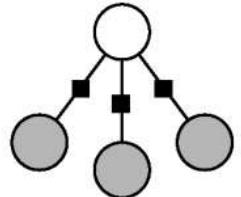
HMMs



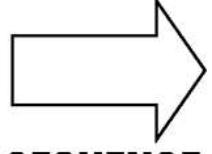
**GENERAL
GRAPHS**



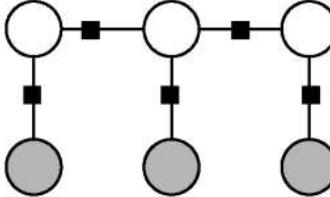
Generative directed models



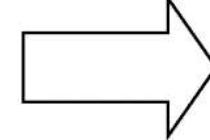
Logistic Regression



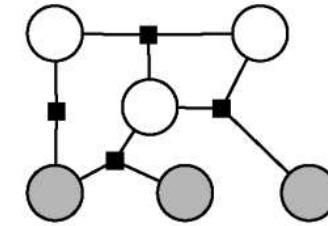
SEQUENCE



Linear-chain CRFs



**GENERAL
GRAPHS**



General CRFs

Thanks for Your Attention!