

# Vector Semantics & Embeddings

Jurafsky and Martin, chapter 6

## Word Meaning

# What do words mean?

In the context of N-gram or text classification methods we've seen so far:

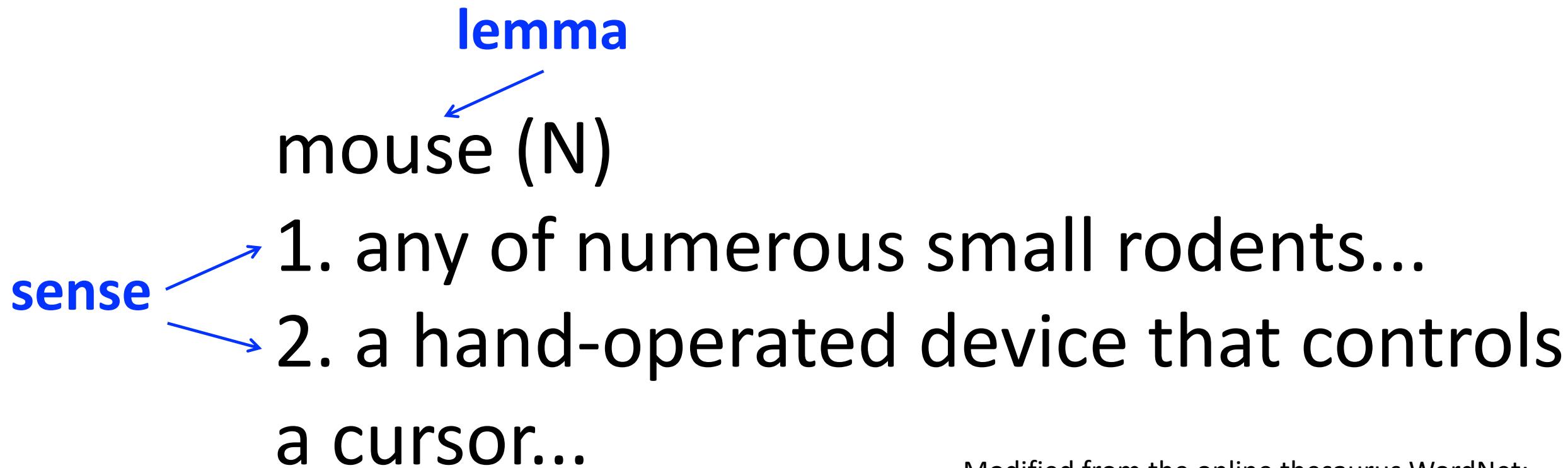
- Words are just strings (or indices  $w_i$  in a vocabulary list)
- Strings can be sorted alphabetically (like lexical entries in a dictionary).
- That's not very satisfactory! Why?

# Homonymy and Polysemy

Word meanings are structured in different ways:

- By different senses:
  - **Homonymy**: words that are identical in form (e.g. *bear*), but have unrelated meanings. Homonyms are listed under different lexical entries in a dictionary and often involve different word classes (e.g. nouns and verbs)
  - **Polysemy**: a word with related senses (e.g. *music*). Polysemous words are listed under a single lexical entry in a dictionary and involve a single word class.

# Lemmas and senses



Modified from the online thesaurus WordNet;  
<https://wordnet.princeton.edu/>

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

# Word similarity and relatedness

- By relatedness or association (e.g. *table* and *chair*; *coffee* and *mug*)
- By semantic fields (e.g. physical objects, events, beliefs or desires)
- By semantic and conceptual relations such as synonymy, hyponymy, antonymy, meronymy

# Relations between senses: Synonymy

Synonyms have the same meaning in some or all contexts.

- filbert / hazelnut
- couch / sofa
- big / large
- automobile / car
- vomit / throw up
- water / H<sub>2</sub>O

# 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/H<sub>2</sub>O

"H<sub>2</sub>O" 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

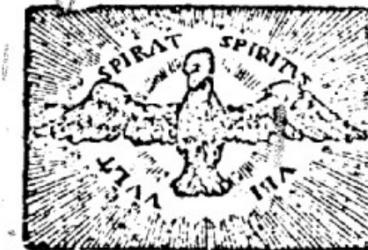
Re: "exact" synonyms

"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 & Privilegi 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
- *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
<b>Valence</b>	love	1.000		toxic	0.008
	happy	1.000		nightmare	0.005
<b>Arousal</b>	elated	0.960		mellow	0.069
	frenzy	0.965		napping	0.046
<b>Dominance</b>	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

# Word Meaning

Vector  
Semantics &  
Embeddings

# Vector Semantics

Vector  
Semantics &  
Embeddings

# Computational models of word meaning

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

We'll introduce **vector semantics**

The standard model in language processing!

Handles many of our goals!

# Ludwig Wittgenstein

PI #43:

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

# Let's define words by their usages

One way to define "usage":

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

Zellig Harris (1954):

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

# The Distributional Approach to Word Meaning

*The distribution of an element will be understood as the sum of all of its environments. An environment of an element A is an existing array of its co-occurrences.* (Harris 1954, p. 146)

*The fact that, for example, not every adjective occurs with every noun can be used as a measure of meaning difference.* (Harris 1954, p. 156)

*You shall know a word by the company it keeps.* (J.R. Firth 1957, p. 2)

# The Distributional Hypothesis (Sahlgren 2008)

➤ *Words which are similar in meaning occur in similar contexts.*

(Rubenstein & Goodenough 1965)

➤ *Words with similar meanings will occur with similar neighbors if enough text material is available.*

(Schütze & Pedersen 1995)

# Sketch-Engine: Collocates of Sweden and Norway

WORD SKETCH DIFFERENCE      English Web 2020 (enTenTen20)      ⓘ

Account expires in April 2022 »  
Get more space +

Sweden 691,676x      Norway 557,184x

SEARCH DOWNLOAD EYE FAVORITES HELP PROFILE

"Sweden/Norway" and/or ...				
Uppsala	3,037	0	...	
Gothenburg	6,277	13	...	
Stockholm	18,481	60	...	
Norway	27,277	372	...	
Finland	15,853	8,526	...	
Switzerland	8,988	5,785	...	
Denmark	18,415	15,332	...	
Netherlands	6,046	7,118	...	
Iceland	1,350	7,960	...	
Sweden	394	27,277	...	
Oslo	64	12,659	...	
Bergen	15	5,046	...	

verbs with "Sweden/Norway" as object				
non-align	21	0	...	
captain	32	0	...	
defeat	590	199	...	
beat	709	326	...	
neighbour	145	110	...	
tour	238	186	...	
neighbor	64	53	...	
rule	151	206	...	
invade	152	830	...	
Christianise	0	9	...	
Christianize	0	21	...	
cede	0	120	...	

verbs with "Sweden/Norway" as subject				
average	128	0	...	
defeat	287	80	...	
invade	101	29	...	
cede	132	42	...	
ban	147	82	...	
rank	188	164	...	
pledge	45	103	...	
rename	24	75	...	
export	46	187	...	
pine	0	40	...	
rat	0	45	...	
spruce	0	136	...	

adjective predicates of "Sweden/Norway"				
Swedish	13	0	...	
U17	10	0	...	
s	18	0	...	
spectacular	84	11	...	
neutral	102	35	...	
international	108	77	...	
socialist	14	13	...	
national	36	52	...	
mountainous	0	17	...	
Norwegian	0	10	...	
+47	0	14	...	
spruce	0	42	...	

# 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
<b>Valence</b>	love	1.000		toxic	0.008
	happy	1.000		nightmare	0.005
<b>Arousal</b>	elated	0.960		mellow	0.069
	frenzy	0.965		napping	0.046
<b>Dominance</b>	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

# Reminder: Vectors in linear/logistic regression

- Encoding data points by a set of features in a high-dimensional vector space
- We will now generalize this type of encoding to the representation of words
- Consult JM, Chapter 5 and the slides on linear/logistic regression and on vectors and vector operators (see readings on moodle for the current chapter)

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

# Vector Semantics

Vector  
Semantics &  
Embeddings

# Words and Vectors

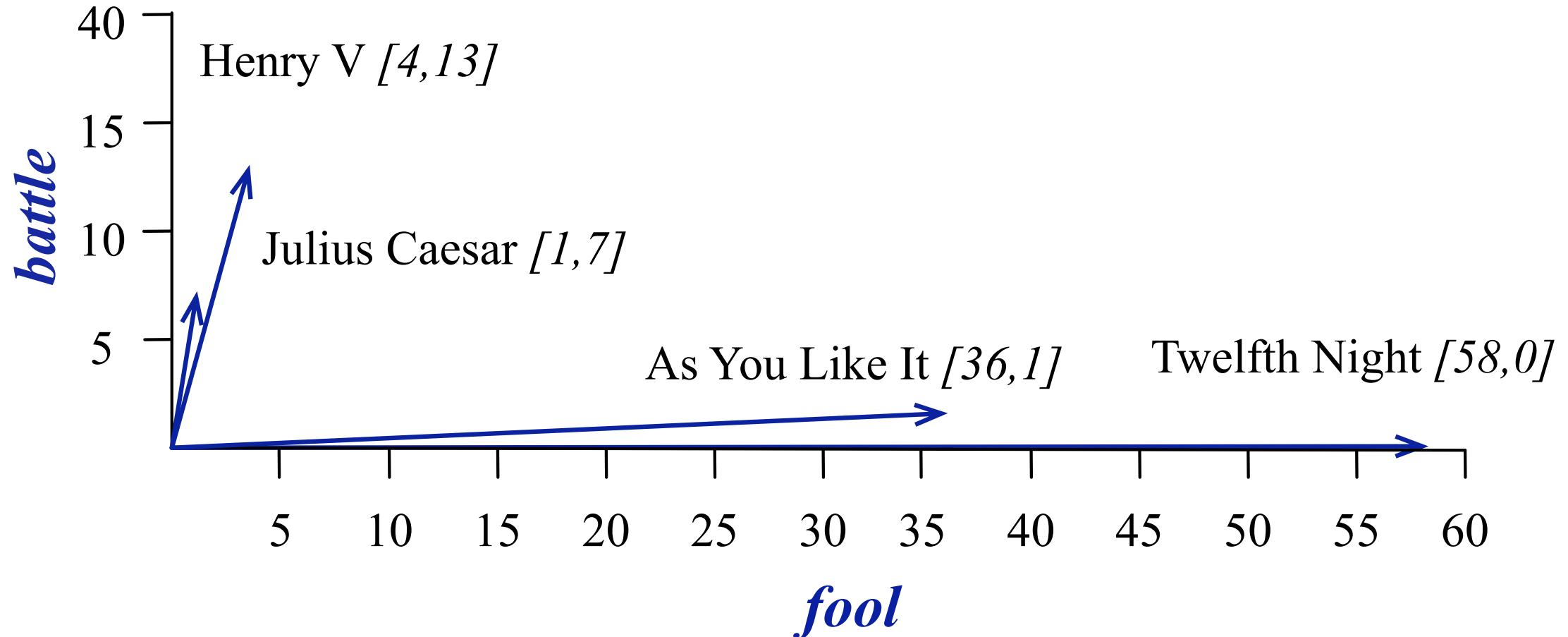
Vector  
Semantics &  
Embeddings

# 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"

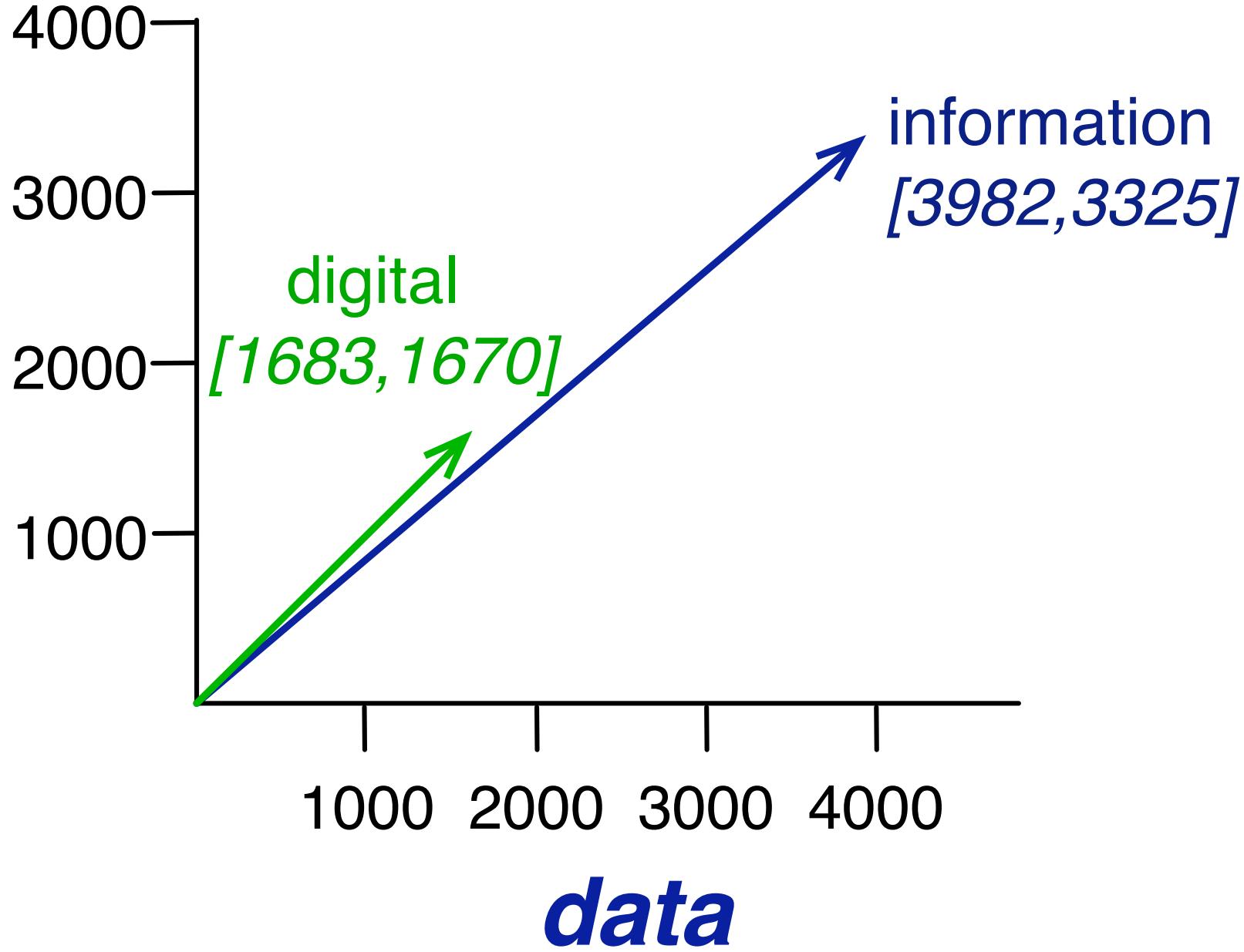
# More common: word-word matrix (or "term-context matrix")

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*



# Words and Vectors

Vector  
Semantics &  
Embeddings

# Vector Semantics & Embeddings

## 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

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

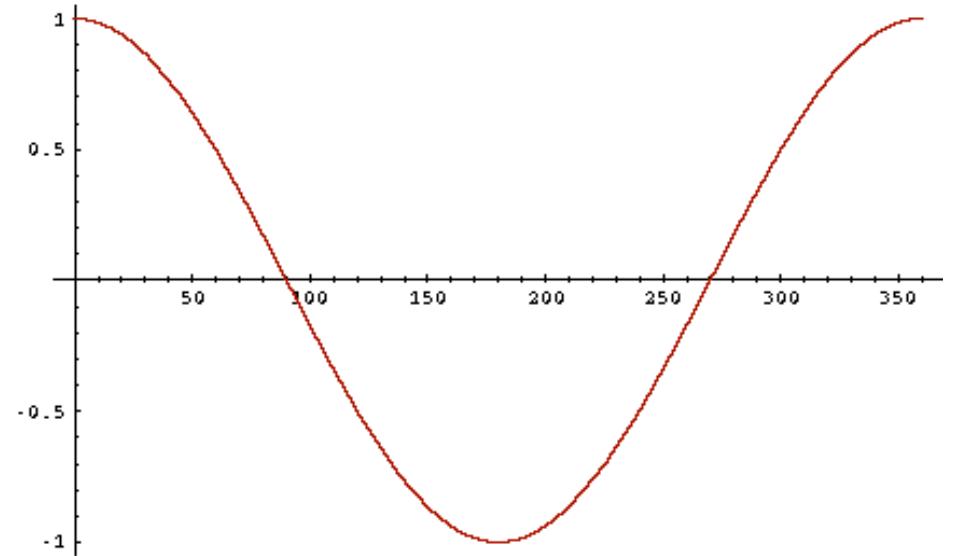
Based on the definition of the dot product between two vectors a and b

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

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

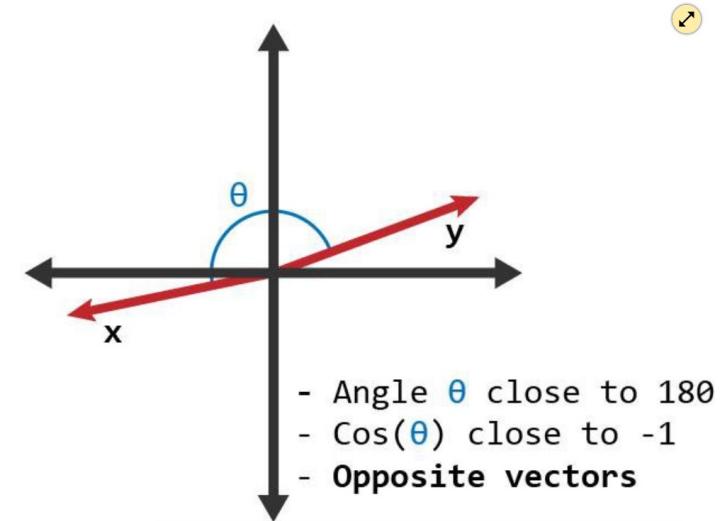
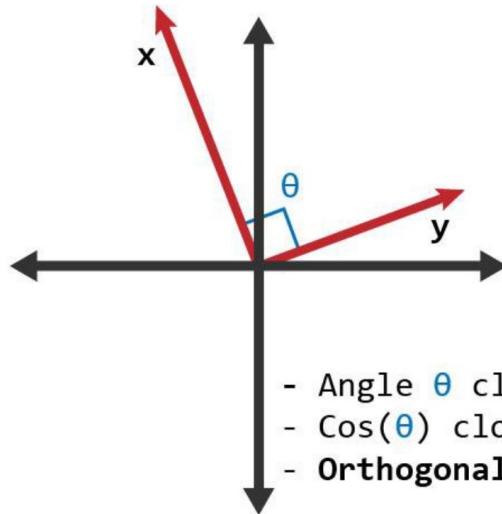
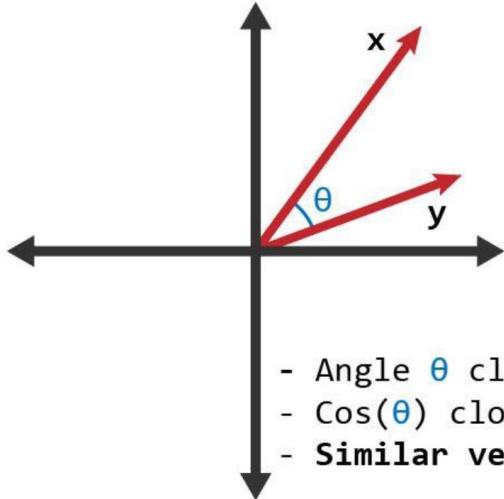
# Cosine as a similarity metric

- 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

# Angles and Cos



Credit to: <https://www.learndatasci.com/glossary/cosine-similarity/>

Recall: the following cosine values:

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

# Cosine examples

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

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

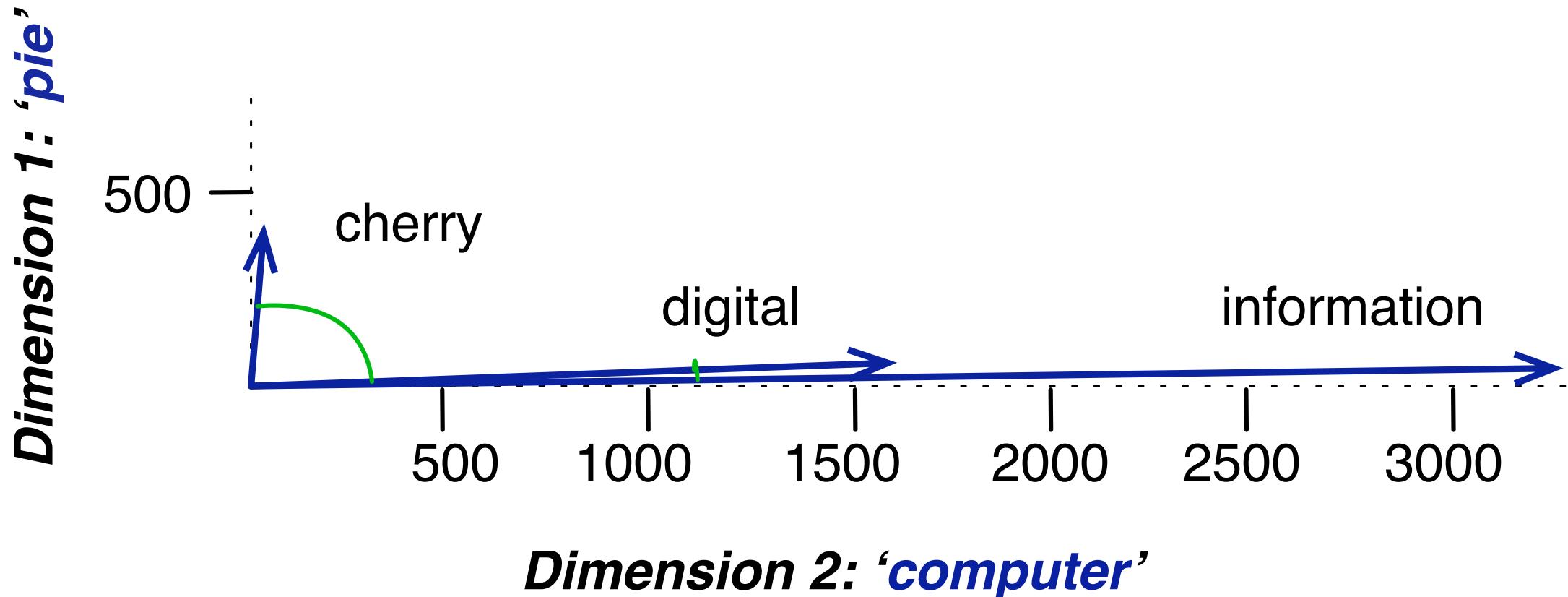
$$\cos(\text{cherry}, \text{information}) =$$

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

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

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

# Visualizing cosines (well, angles)



# 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	...

# Vector Semantics & Embeddings

## Cosine for computing word similarity

# Vector Semantics & Embeddings

## 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:

	<b>Collection Frequency</b>	<b>Document Frequency</b>
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

# Summary

$\text{tf-idf}_{t,d} = \text{tf}_{t,d} \times \text{idf}_f$  assigns to term  $t$  a weight in document  $d$  that is

1. highest when  $t$  occurs many times within a small set of documents (thus lending high discriminating power to those documents)
2. lower when the term occurs fewer times in a document, or occurs in many documents (thus offering a less pronounced relevance signal)
3. Lowest when the term occurs in virtually all documents.

# Summary of definitions

- **collection frequency** ( $cf_t$ ): the total number of occurrences of the term  $t$  in a collection.
- **document frequency** ( $df_t$ ): the number of documents that contain the term  $t$  at least once.
- **term frequency** ( $tf_{t,d}$ ): the number of occurrences of a term  $t$  in a document  $d$ .

# TF-IDF

Vector  
Semantics &  
Embeddings

# Vector Semantics & Embeddings

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  $c_j$

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

	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

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

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

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

	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

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

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

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

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

	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)
<b>cherry</b>	0.0002	0.0007	0.0008	0.0377	0.0021	0.0415
<b>strawberry</b>	0.0000	0.0000	0.0001	0.0051	0.0016	0.0068
<b>digital</b>	0.1425	0.1436	0.0073	0.0004	0.0003	0.2942
<b>information</b>	0.2838	0.3399	0.0323	0.0004	0.0011	0.6575
<b>p(context)</b>	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
<b>cherry</b>	0	0	0	4.38	3.30
<b>strawberry</b>	0	0	0	4.10	5.51
<b>digital</b>	0.18	0.01	0	0	0
<b>information</b>	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  $\alpha = 0.75$ :

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

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

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

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

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

# Word2vec

Vector  
Semantics &  
Embeddings

# Sparse versus dense vectors

tf-idf (or PMI) vectors are

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

Alternative: learn vectors which are

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

# Sparse versus dense vectors

## Why dense vectors?

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

# Common methods for getting short dense vectors

## “Neural Language Model”-inspired models

- Word2vec (skipgram, CBOW), GloVe

## Singular Value Decomposition (SVD)

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

## Alternative to these "static embeddings":

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

# Simple static embeddings you can download!

Word2vec (Mikolov et al)

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

GloVe (Pennington, Socher, Manning)

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

# Word2vec

Popular embedding method

Very fast to train

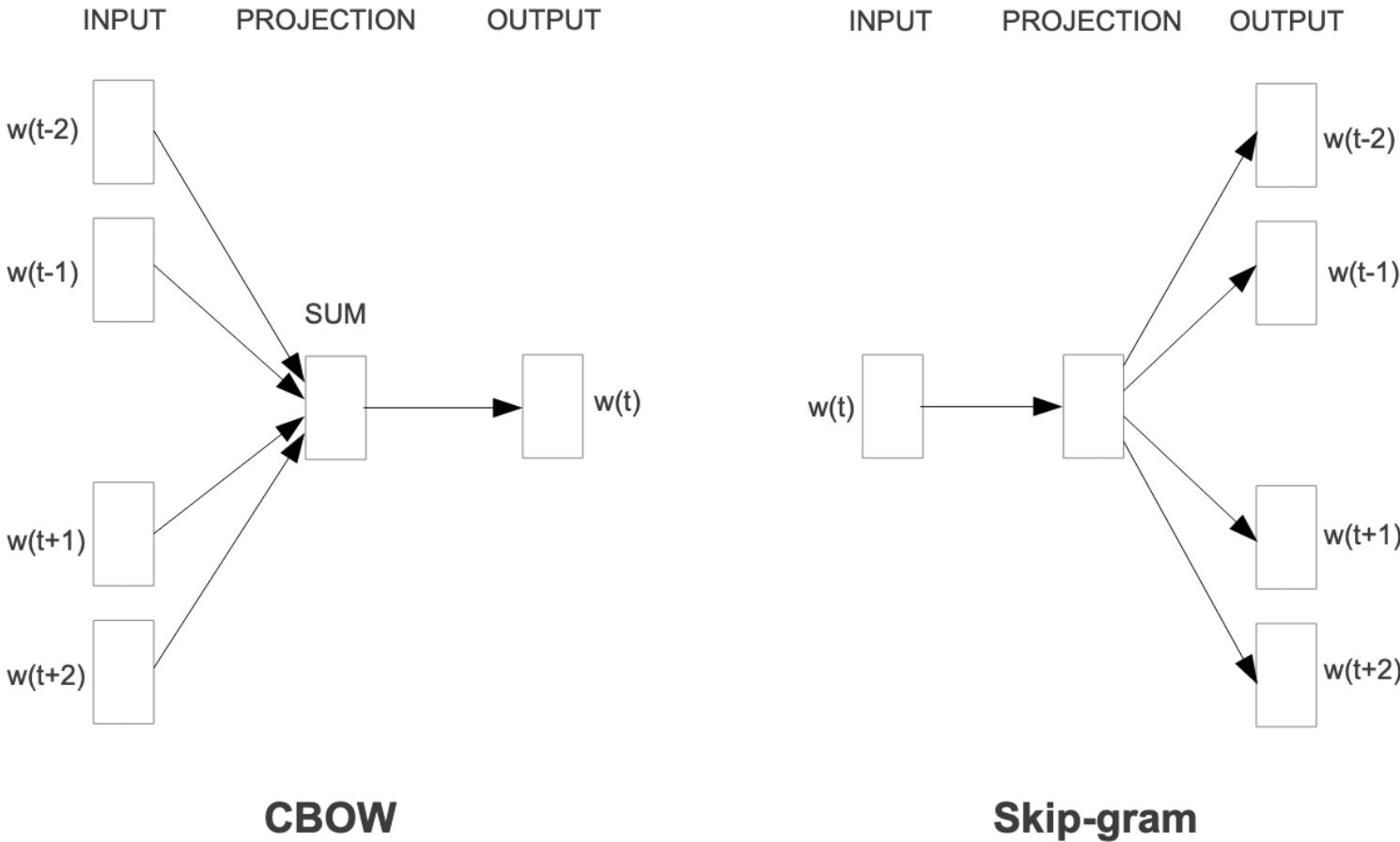
Code available on the web

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

Word2vec provides various options. We'll do:

**skip-gram with negative sampling (SGNS)**

# Word Embeddings created by Self-Supervision from large data sets (Mikolov et al. 2013)



# 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)

# Collecting Training Examples; Slide due to Jay Alammar: <https://jalammar.github.io/illustrated-word2vec/>

Thou shalt not make a machine in the likeness of a human mind

thou	shalt	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

thou	shalt	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

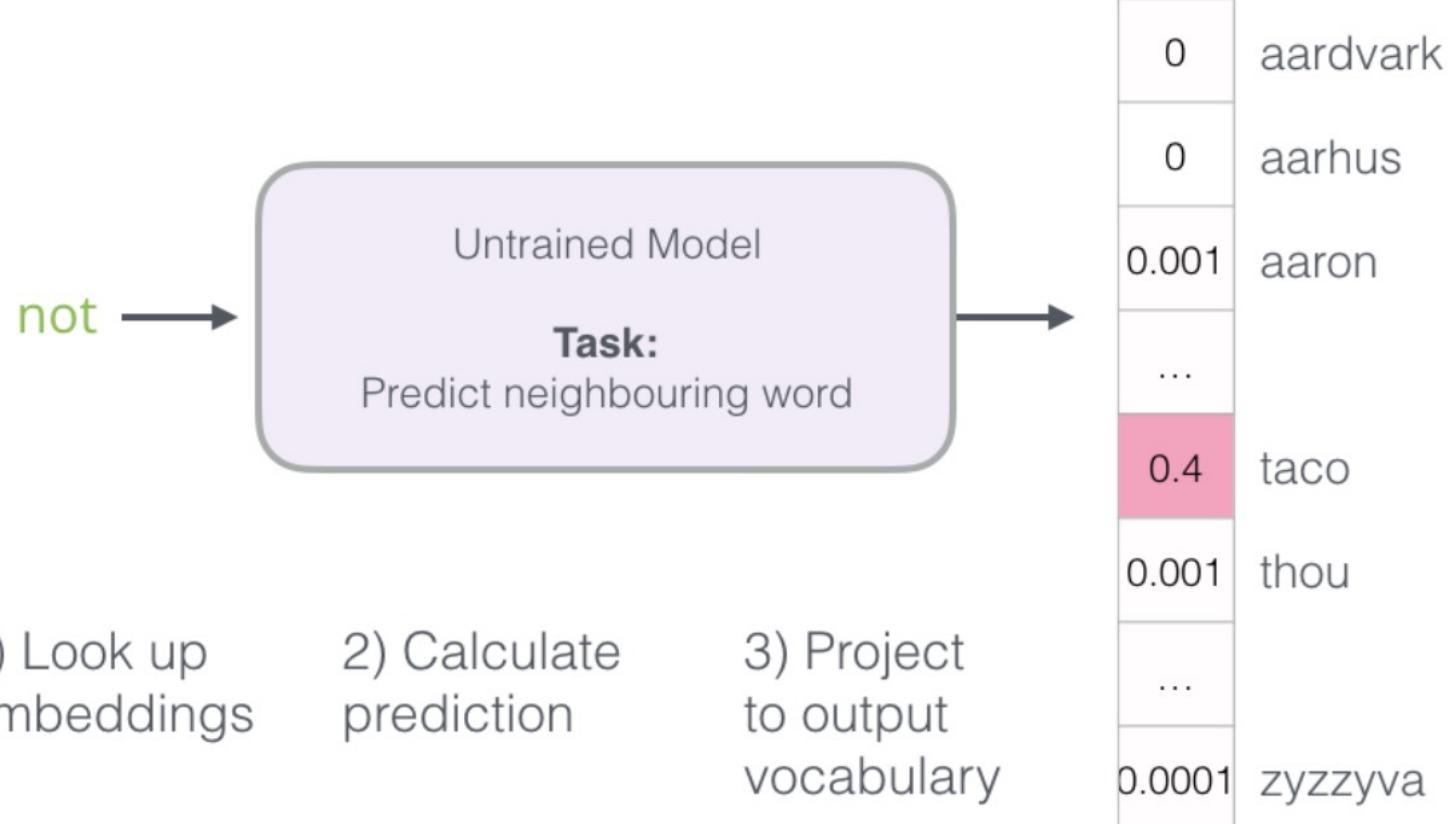
thou	shalt	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

thou	shalt	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

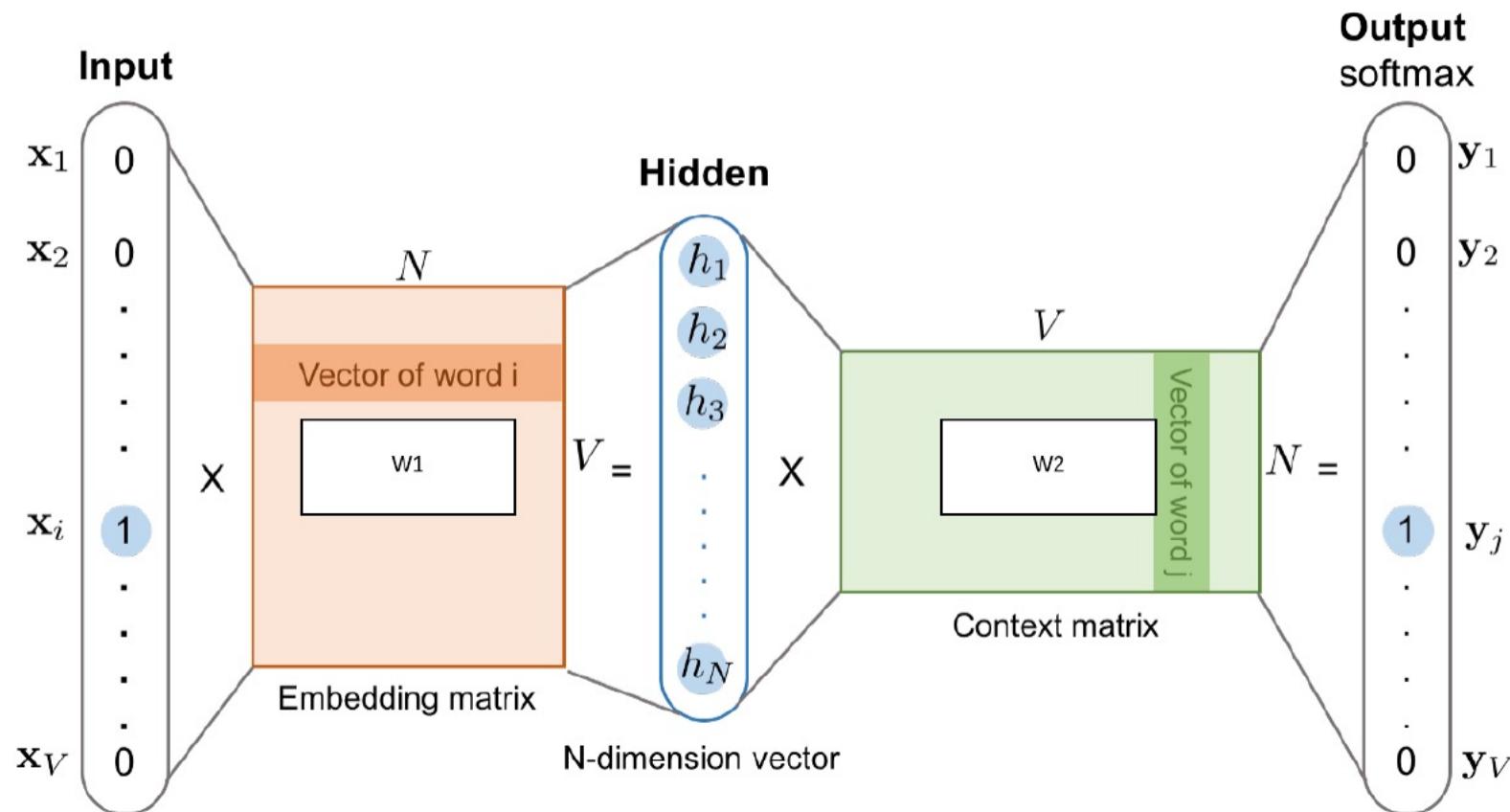
thou	shalt	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine
a	not
a	make
a	machine
a	in
machine	make
machine	a
machine	in
machine	the
in	a
in	machine
in	the
in	likeness

# Training an Embedding Model ; Slide due to Jay Alammar: <https://jalammar.github.io/illustrated-word2vec/>

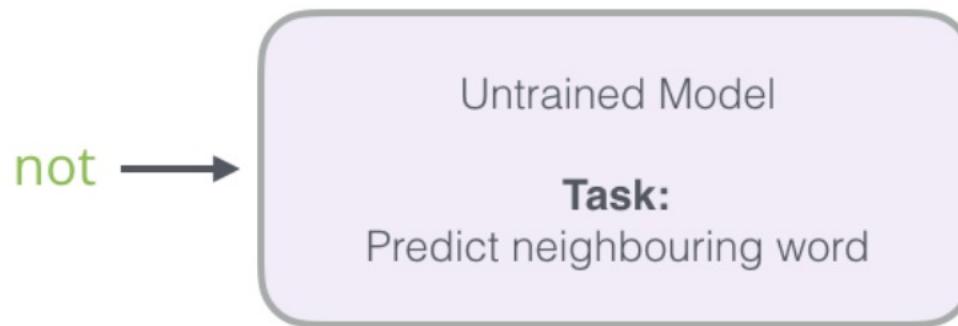


# Word2Vec Architecture with a Hidden Layer



Change task for training; Slide due to Jay Alammar:  
<https://jalammar.github.io/illustrated-word2vec/>

From:



1) Look up  
embeddings

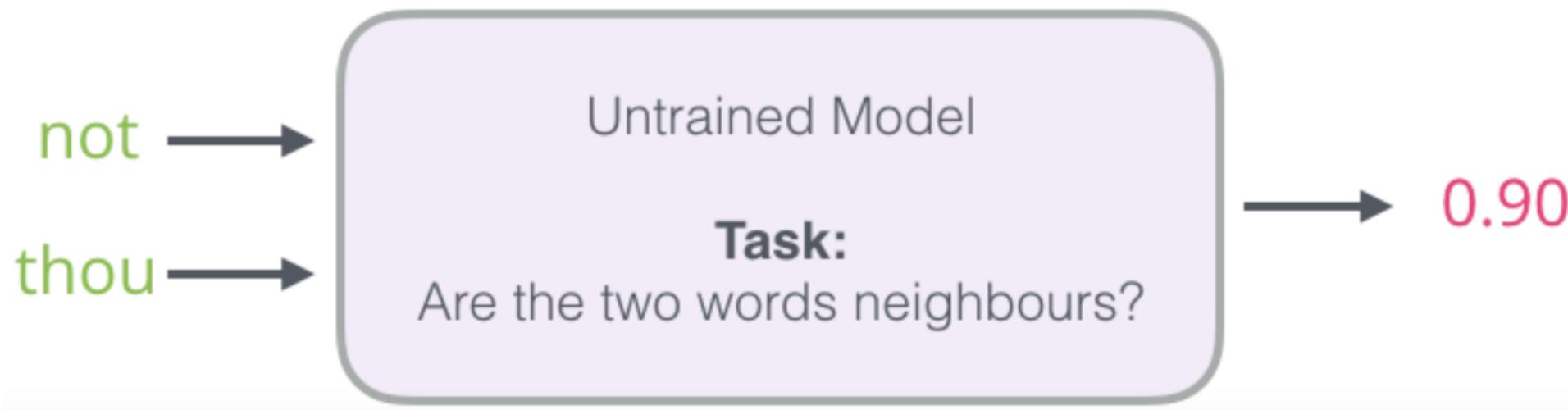
2) Calculate  
prediction

**3) Project  
to output  
vocabulary**

**[Computationally  
Intensive]**

Change task for training; Slide due to Jay Alammar:  
<https://jalammar.github.io/illustrated-word2vec/>

To:



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 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
---------	---------	---------	----

# 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)

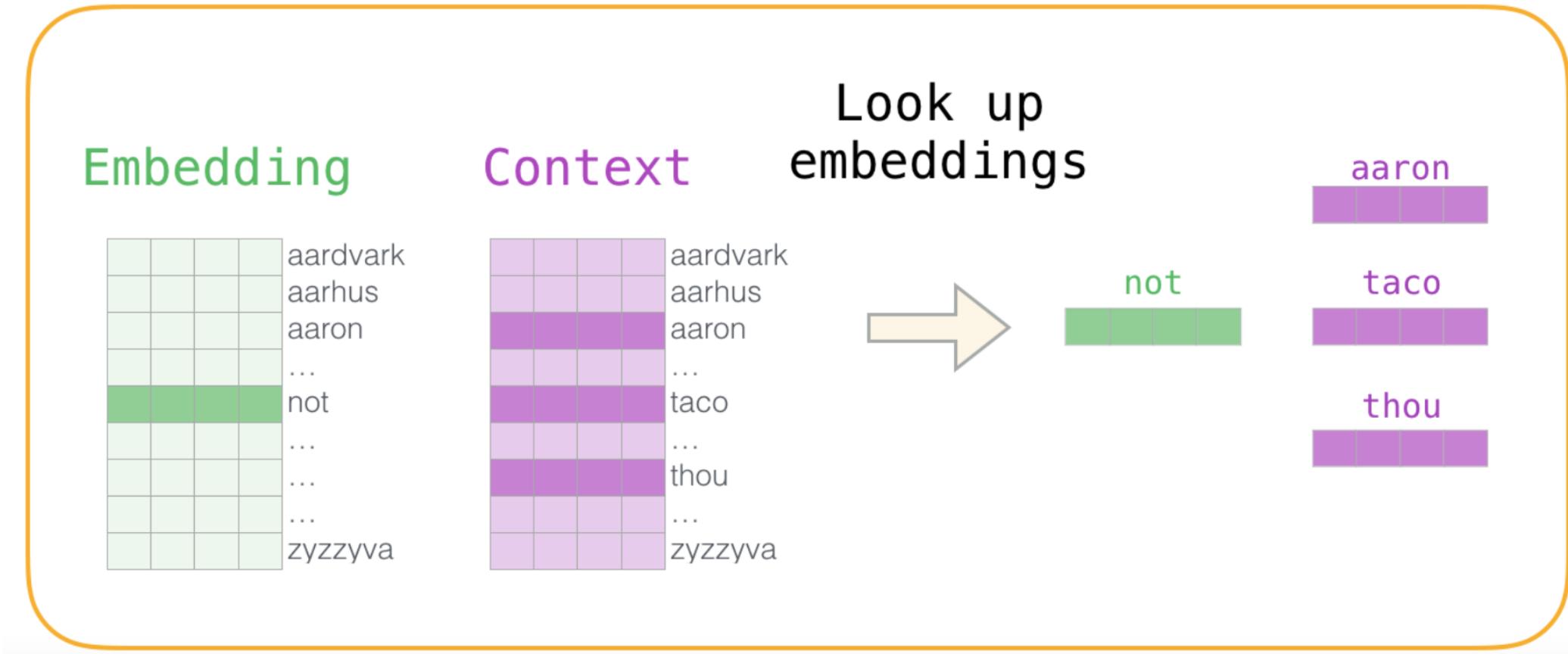
...

And assigns each pair a probability:

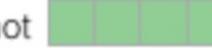
$$P(+ | w, c)$$

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

# Word2Vec Training Process



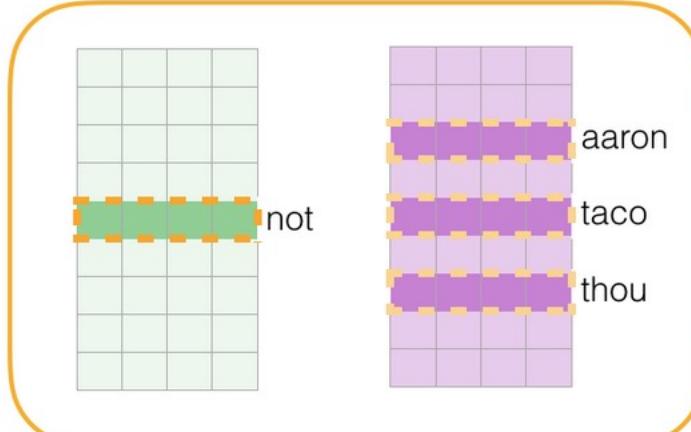
Skigram with Negative Sampling (SGNS); Slide due to Jay Alammar:  
<https://jalammar.github.io/illustrated-word2vec/>

input word	output word	target	input • output	sigmoid()	Error
not 	thou 	1	0.2	0.55	0.45
not 	aaron 	0	-1.11	0.25	-0.25
not 	taco 	0	0.74	0.68	-0.68

$$\text{error} = \text{target} - \text{sigmoid\_scores}$$

# Training

input word	output word	target	input • output	sigmoid()	Error
not	thou	1	0.2	0.55	0.45
not	aaron	0	-1.11	0.25	-0.25
not	taco	0	0.74	0.68	-0.68



**Update Model Parameters**

# Similarity is computed from dot product

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

- Cosine is just a normalized dot product

So:

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

We'll need to normalize to get a probability

- (cosine isn't a probability either)

# Turning dot products into probabilities

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

To turn this into a probability

We'll use the sigmoid from logistic regression:

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

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

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

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

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

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

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

# Skip-gram classifier: summary

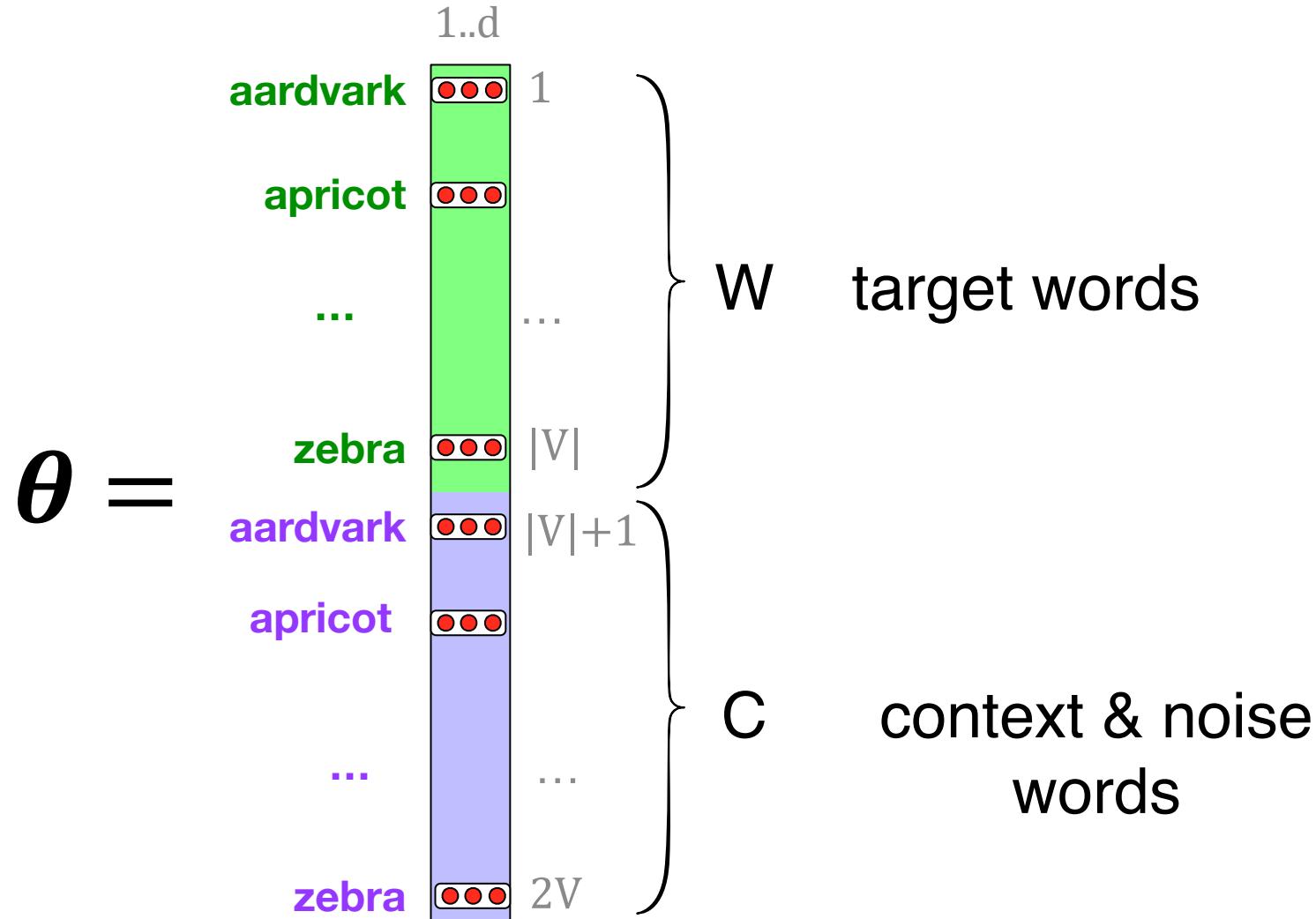
A probabilistic classifier, given

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

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

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

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



# Word2vec

Vector  
Semantics &  
Embeddings

# Vector Semantics & Embeddings

## Word2vec: Learning the embeddings

# Vector Semantics & Embeddings

A quick reminder about binary logistic regression

# Summary of Logistic regression

- Logistic regression is a supervised machine learning classifier that extracts real-valued **features from the input, multiplies each by a weight, sums them, and passes the sum through a sigmoid function** to generate a probability. A threshold is used to make a decision.
- Logistic regression can be used **with two classes** (e.g., positive and negative sentiment) **or with multiple classes** (multinomial logistic regression, for example for n-ary text classification, part-of-speech labeling, etc.).

# Summary of Logistic regression

- Multinomial logistic regression uses the **softmax function** to compute probabilities.
- The weights (vector  $w$  and bias  $b$ ) are learned from a labeled training set via a **loss function**, such as the **cross-entropy loss**, that **must be minimized**.
- Minimizing this loss function is a convex optimization problem, and iterative algorithms like **gradient descent** are used to find the optimal weights.

# 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

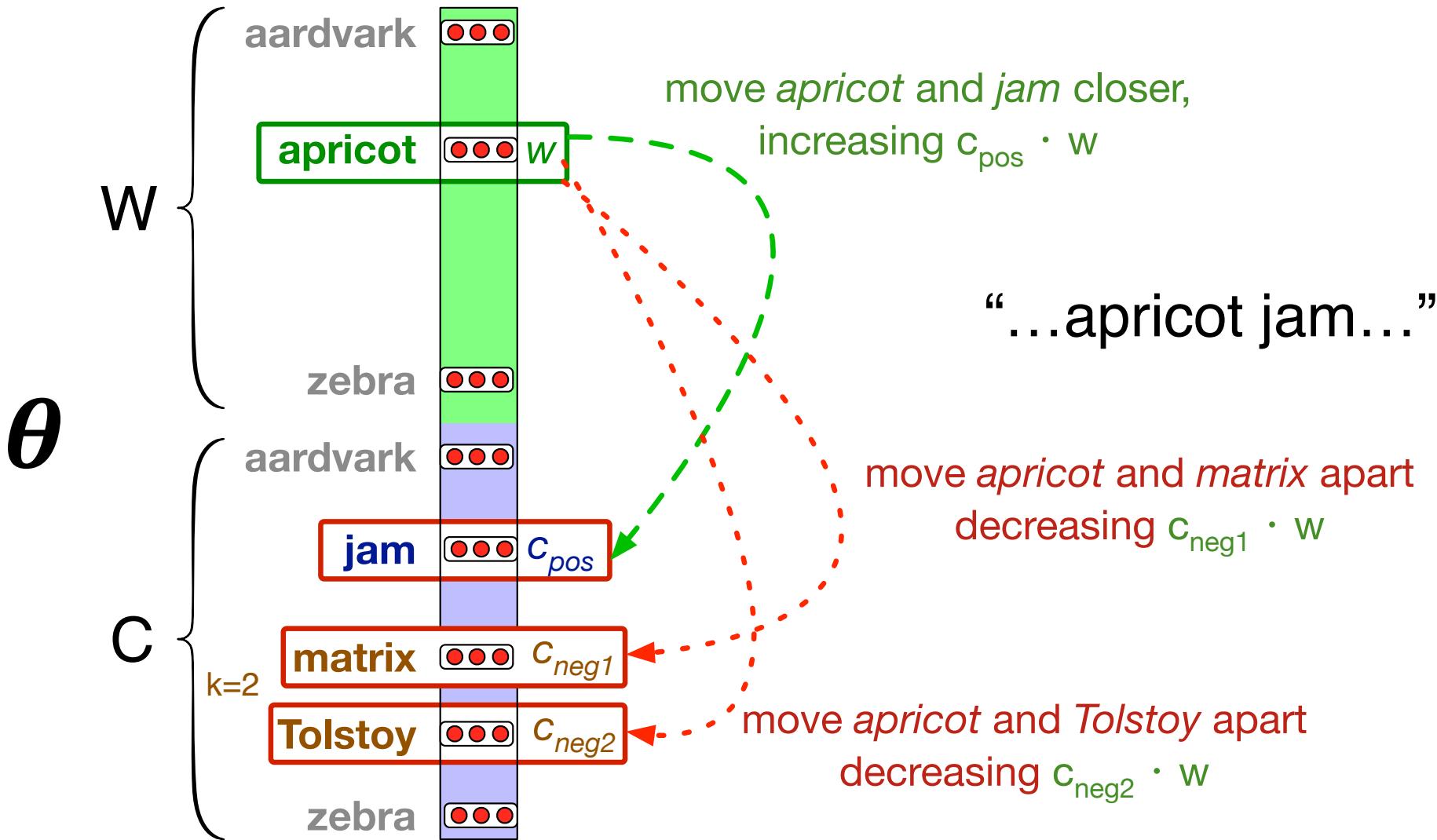
How to learn?

- 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**  $\eta$
  - Higher learning rate means move  $w$  faster

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

# The derivatives of the loss function

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

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

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

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

# Update equation in SGD

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

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

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

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

# Two sets of embeddings

SGNS learns two sets of embeddings

Target embeddings matrix  $W$

Context embedding matrix  $C$

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

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

Start with  $V$  random  $d$ -dimensional vectors as initial embeddings

Train a classifier based on embedding similarity

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

# Vector Semantics & Embeddings

## Word2vec: Learning the embeddings

# Vector Semantics & Embeddings

## Properties of Embeddings

# The kinds of neighbors depend on window size

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

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

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

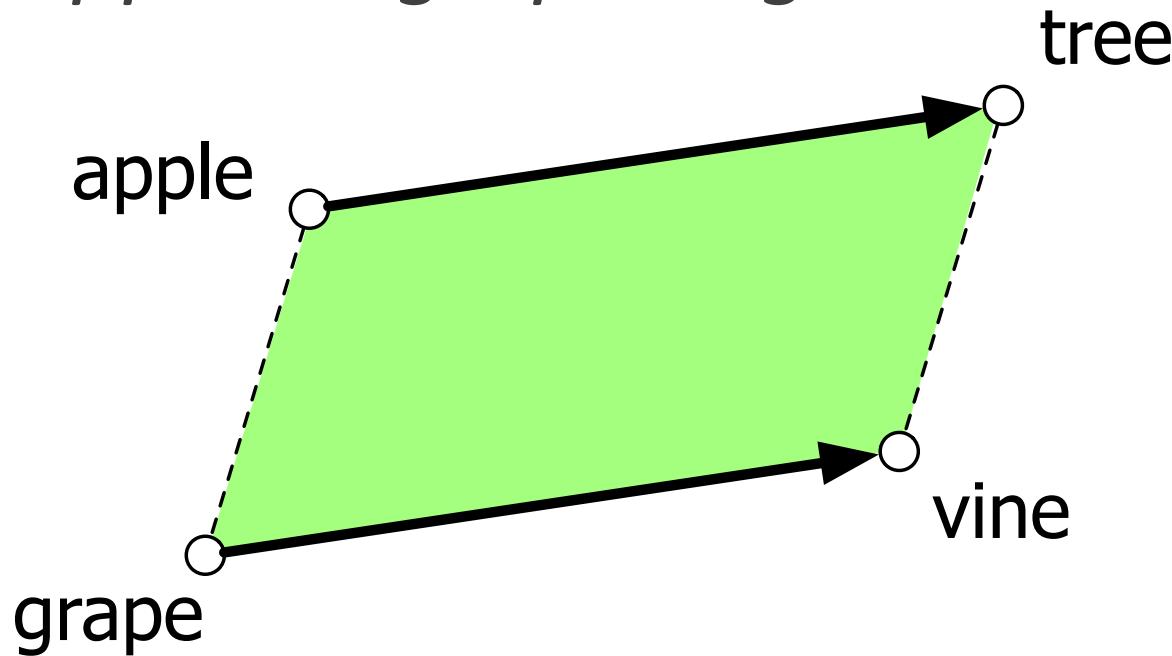
- *Hogwarts* nearest neighbors are Harry Potter world:
  - *Dumbledore, half-blood, Malfoy*

# Analogical relations

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

To solve: "*apple is to tree as grape is to \_\_\_\_\_*"

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



# Analogical relations via parallelogram

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

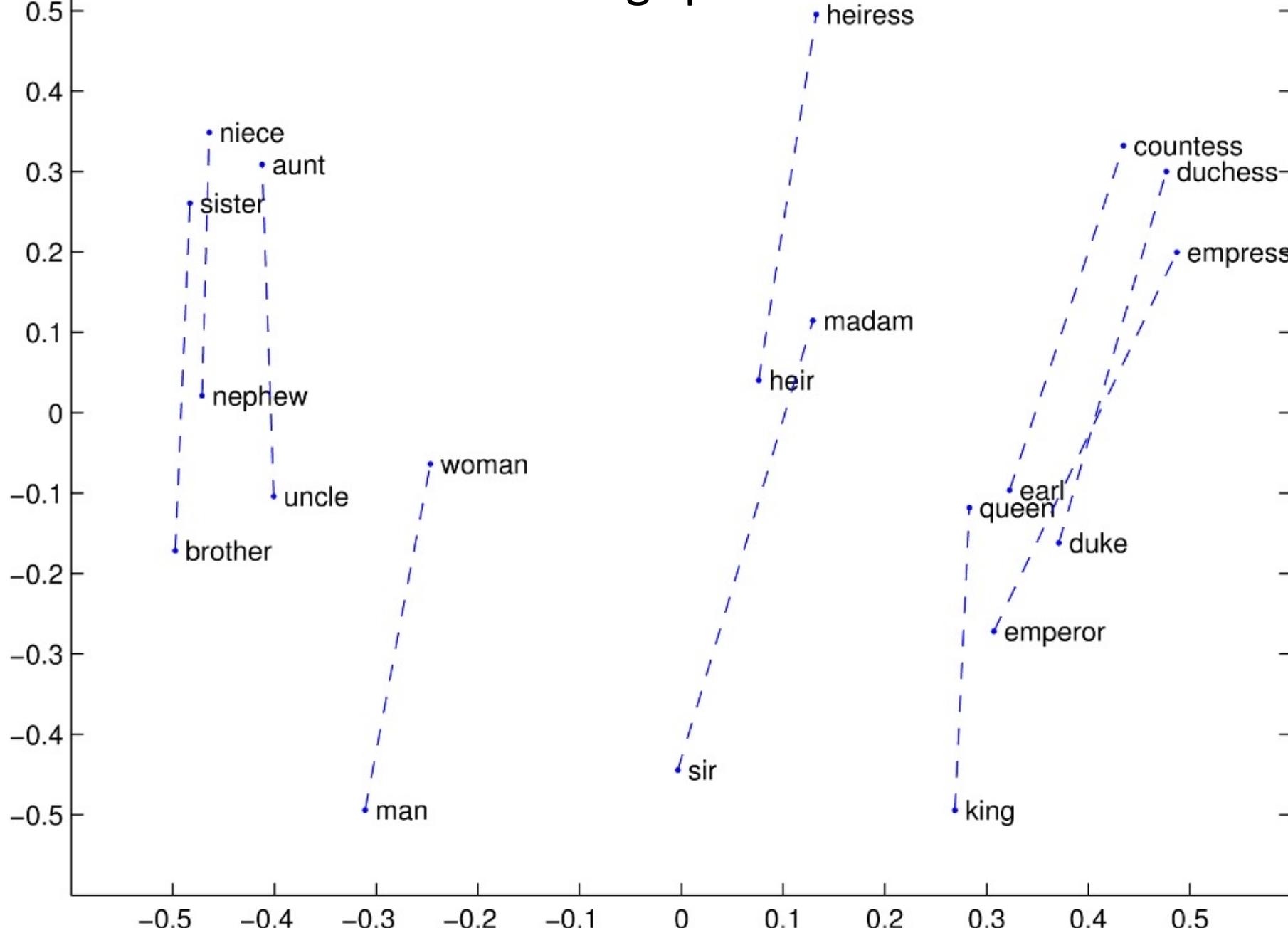
$$\overrightarrow{\text{king}} - \overrightarrow{\text{man}} + \overrightarrow{\text{woman}} \text{ is close to } \overrightarrow{\text{queen}}$$

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

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

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

# Structure in GloVe Embedding space



# Caveats with the parallelogram method

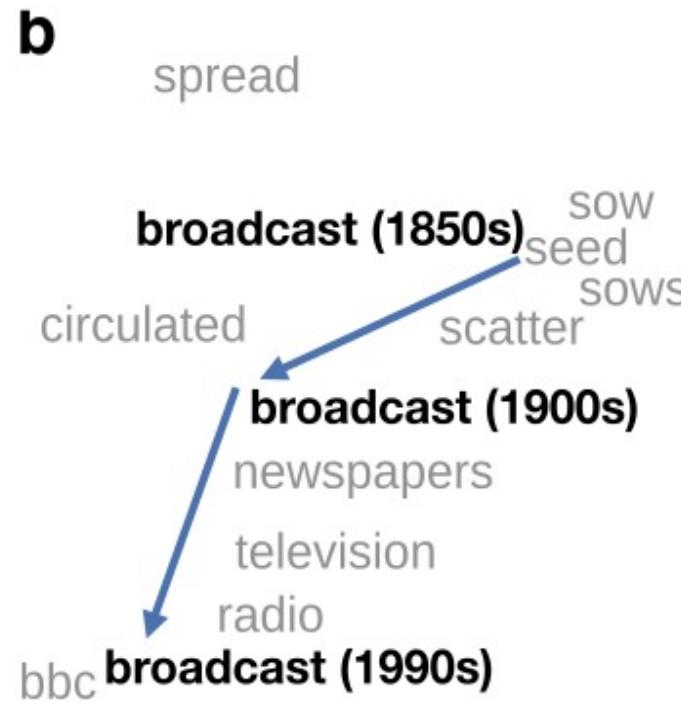
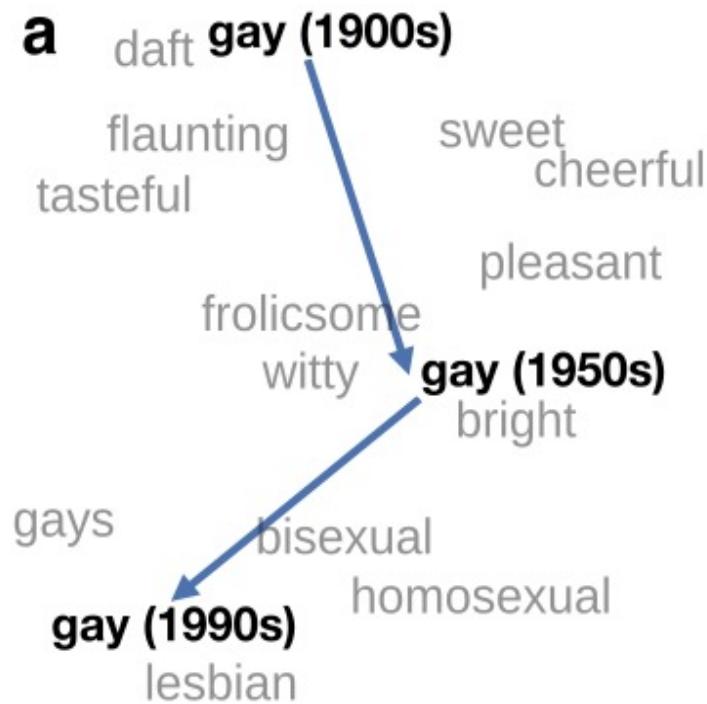
It only seems to work for frequent words, small distances and certain relations (relating countries to capitals, or parts of speech), but not others. (Linzen 2016, Gladkova et al. 2016, Ethayarajh et al. 2019a)

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 20<sup>th</sup> century.
- These match the results of old surveys done in the 1930s

# Vector Semantics & Embeddings

## Properties of Embeddings

# **GloVe: Global Vectors (Pennington, Socher and Manning 2014)**

Erhard Hinrichs

Seminar für Sprachwissenschaft  
Eberhard-Karls Universität Tübingen

# Bibliographical Reference

Jeffrey Pennington, Richard Socher, and Christopher D. Manning.  
2014. GloVe: Global Vectors for Word Representation.  
Proceedings of the 2014 Conference on Empirical Methods in  
Natural Language Processing (EMNLP), 1532–1543.

# The word2vec Algorithm

- ▶ Imposes a sliding window over the whole corpus and goes through whole corpus one focus word at a time.
- ▶ Skipgram with negative sampling predicts surrounding words of each word focus word one at a time.
- ▶ Skipgram with negative sampling is trained by a logistic regression model with cross-entropy loss function.
  - ▶ Captures cooccurrences of words only locally with a given window and does not capture global cooccurrences over the entire corpus.
  - ▶ is inefficient in that repeated cooccurrences are re-computed "from scratch".
- ▶ By contrast: GloVe computes cooccurrences counts over the entire corpus.
- ▶ The GloVe model is trained by a weighted linear regression model with a least squared loss function.

# Previous Approaches for Learning Word Vectors

Global matrix factorization methods such as Latent Semantic Analysis (LSA; Deerwester et al. 1990)

- ▶ widely used in Information Retrieval (IR) and based term-document matrices
- ▶ uses Singular Value Decomposition (SVD) to rerank the dimensions of a matrix from most to least informative
- ▶ LSA, practitioners assume that only the top 300 or so dimensions (out of tens or even hundreds of thousands) are useful for capturing the meaning of texts.
- ▶ downsides of LSA:
  - ▶ not suitable for very large corpora
  - ▶ does not adequately capture the substructure of the vector space and thus does poorly on analogy tasks.

## Previous Approaches for Learning Word Vectors

Local context window methods such as the Skipgram Model

- ▶ uses a sliding window of local contexts over a large corpus.
- ▶ does not directly capture global information of the corpus.

# The GloVe Approach

GloVe is a global log-bilinear regression model. More specifically:

- ▶ a weighted least squares model trained on global word-word co-occurrence counts obtained from a large corpus, rather than on
  - ▶ sparse term-term matrices
  - ▶ a sliding window of local contexts over a large corpus
- ▶ uses a term-term co-occurrence matrix
- ▶ supports fast training
- ▶ good performance even with small corpora and small vectors
- ▶ scalable to huge corpora

## Distinguishing ratios of target words with discriminative and non-discriminative context words

Probability and ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k   ice)$	$1.9 \times 10^{-4}$	$6.6 \times 10^{-5}$	$3.0 \times 10^{-3}$	$1.7 \times 10^{-5}$
$P(k   steam)$	$2.2 \times 10^{-5}$	$7.8 \times 10^{-4}$	$2.2 \times 10^{-3}$	$1.8 \times 10^{-5}$
$P(k   ice) / P(k   steam)$	8.9	$8.5 \times 10^{-2}$	1.36	0.96

with target words: *ice, steam*

with discriminative words: *gas, solid*

with "noise" words: *water, fashion*

## Loss Function for a Weighted Linear (aka: Least Squares) Regression Model

$$J = \sum_{i,j=1}^V f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2, \text{ where} \quad (1)$$

(i)  $f$  is a weighting function:

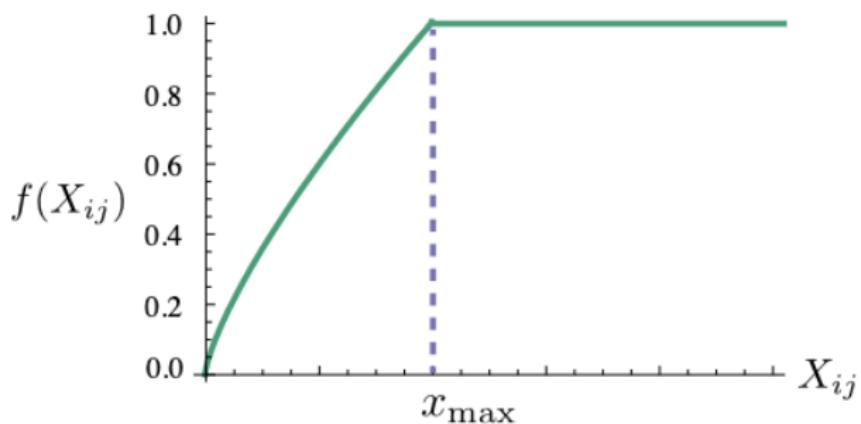
$$f(x) = \begin{cases} (x/x_{max})^\alpha & \text{if } x < x_{max} \\ 1 & \text{otherwise} \end{cases} \quad (2)$$

(ii)  $X_{i,j}$  tabulates the number of times word  $j$  occurs in the context of word  $i$ :  $X_{i,j}/X_i$

# Minimizing the Loss Function $J$ for the GloVe Model by Gradient Descent

- ▶ amounts to updating the word vectors in such a way that the values for  $w_i^T \tilde{w}_j + b_i + \tilde{b}_j$  and for  $\log X_{ij}$  is successively minimized.

**Weighting Function  $f$  with  $\alpha = 3/4$  and  
 $x_{max} = 1$**



## For more detailed discussion

watch the youtube video by **Richard Socher**:

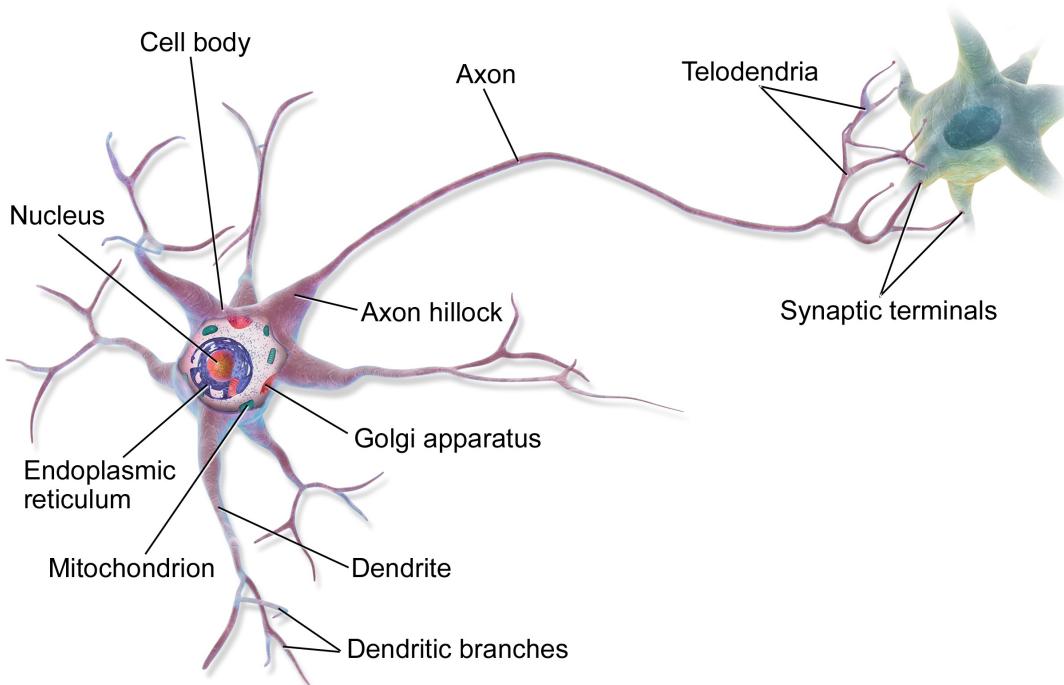
**GloVe: Global Vectors for Word Representation.**

<https://www.youtube.com/watch?v=ASn7ExxLZws&t=2376s>

# Simple Neural Networks and Neural Language Models

## Units in Neural Networks

# This is in your brain



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

# Neural Network Unit

This is not in your brain

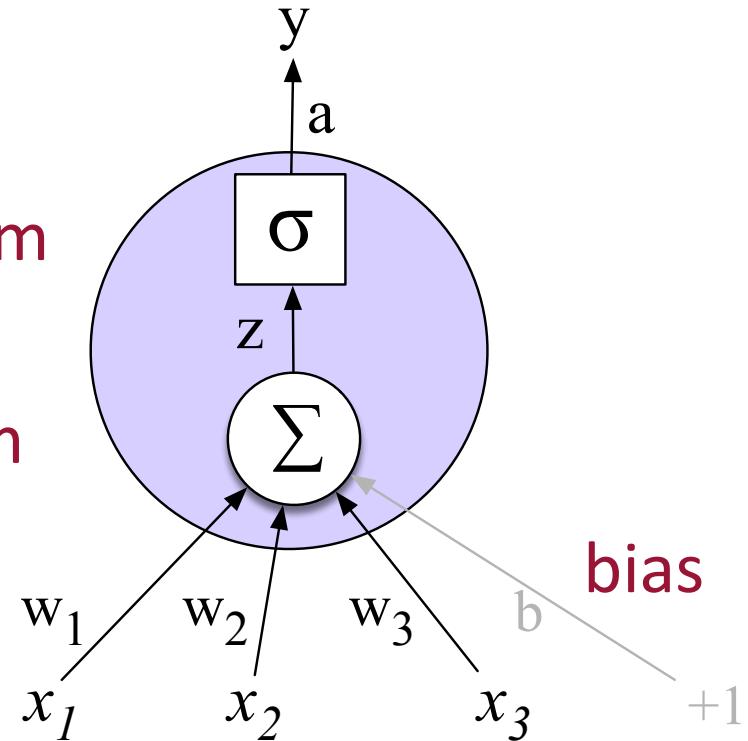
Output value

Non-linear transform

Weighted sum

Weights

Input layer



# Neural unit

Take weighted sum of inputs, plus a bias

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

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

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

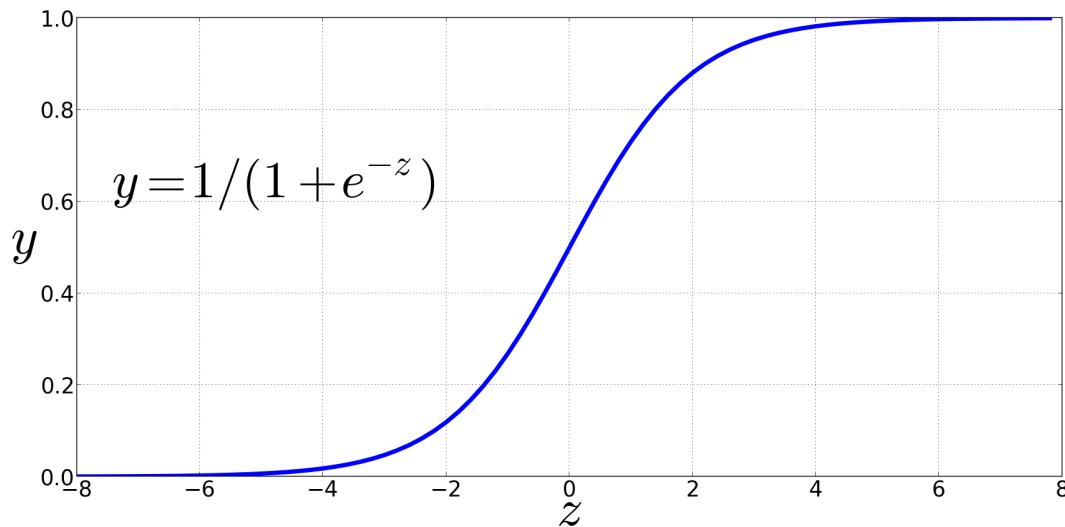
$$y = a = f(z)$$

# Non-Linear Activation Functions

We've already seen the sigmoid for logistic regression:

Sigmoid

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



# Final function the unit is computing

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

# Final unit again

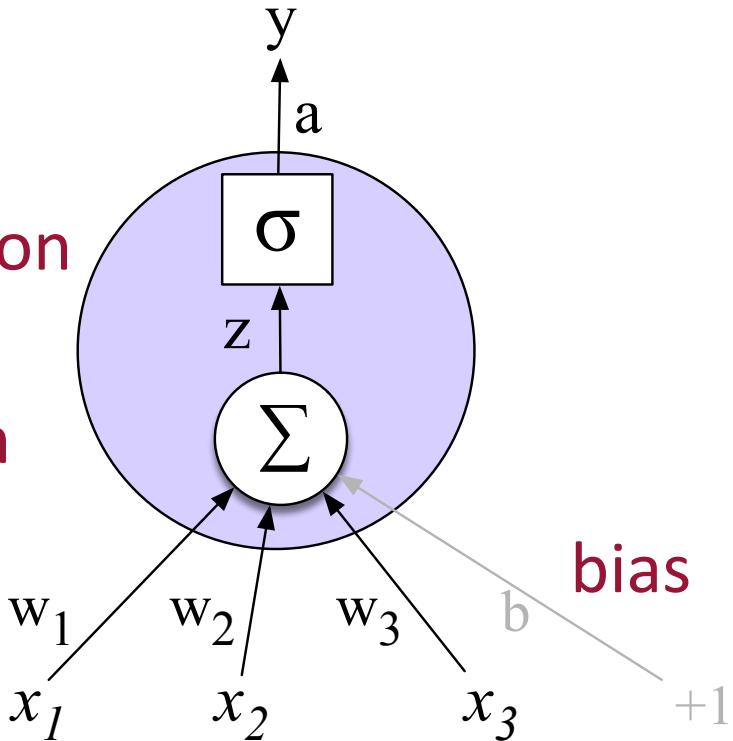
Output value

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  $x$ :

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

How is the output  $y$  computed?

$$y = ?$$

# Neural unit

Take weighted sum of inputs, plus a bias

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

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

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

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

# An example

*Suppose a unit has:*

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

$$b = 0.5$$

What happens with input x:

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

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

# An example

*Suppose a unit has:*

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

$$b = 0.5$$

What happens with the following input  $x$ ?

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

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

# An example

*Suppose a unit has:*

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

$$b = 0.5$$

What happens with input  $x$ :

$$x = [0.5, 0.6, 0.1]$$
$$1$$

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

# An example

*Suppose a unit has:*

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

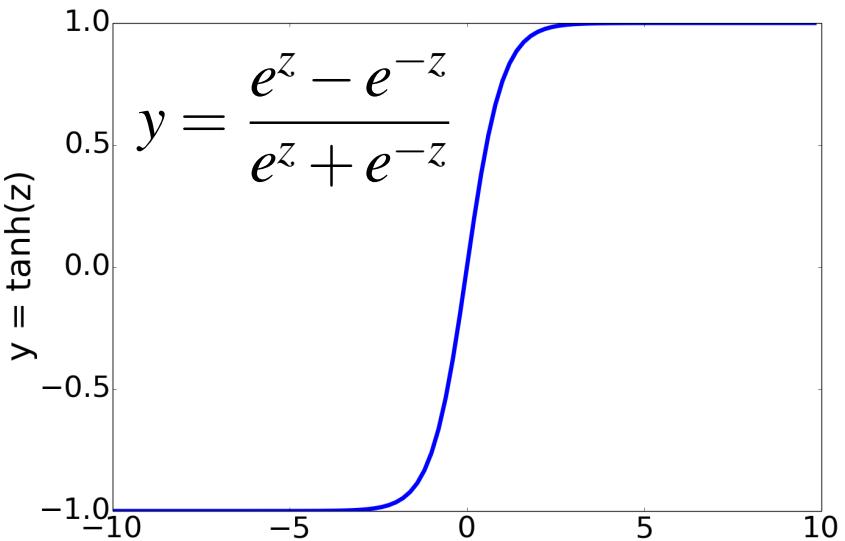
$$b = 0.5$$

What happens with input  $x$ :

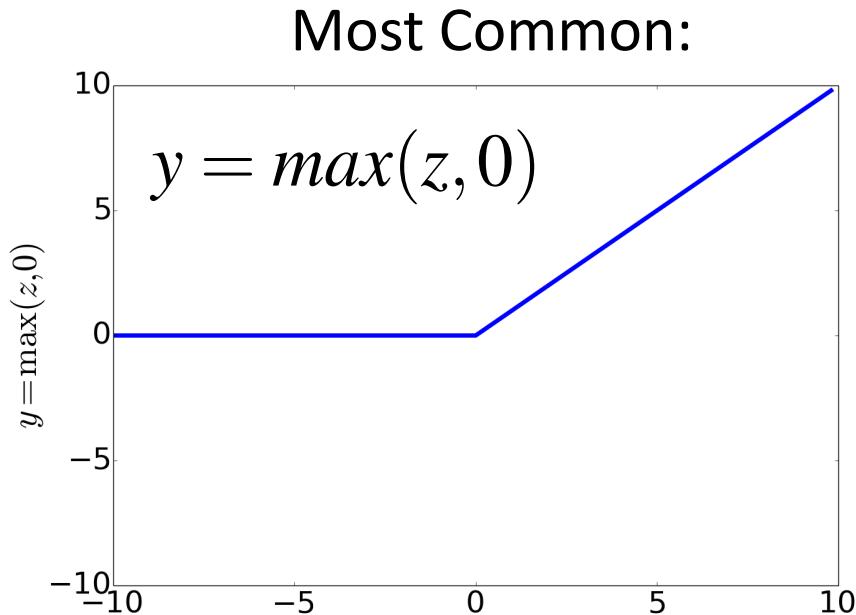
$$x = [0.5, 0.6, 0.1]$$

$$y = \sigma(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}} = \frac{1}{1 + e^{-(.5*.2+.6*.3+.1*.9+.5)}} = \frac{1}{1 + e^{-0.87}} = .70$$

# Non-Linear Activation Functions besides sigmoid



tanh



ReLU  
Rectified Linear Unit

# Simple Neural Networks and Neural Language Models

## Units in Neural Networks

# Simple Neural Networks and Neural Language Models

## The XOR problem

# The XOR problem

Minsky and Papert (1969)

Can neural units compute simple functions of input?

AND		OR		XOR	
x1	x2	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

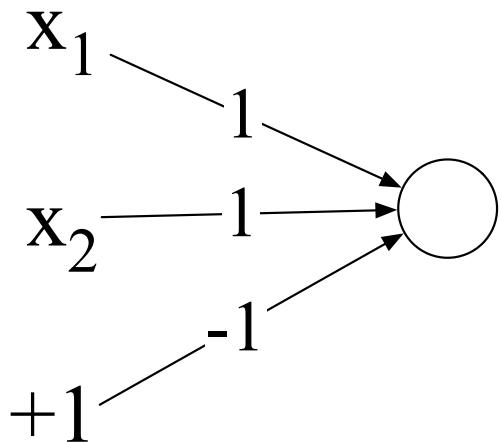
A very simple neural unit

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

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

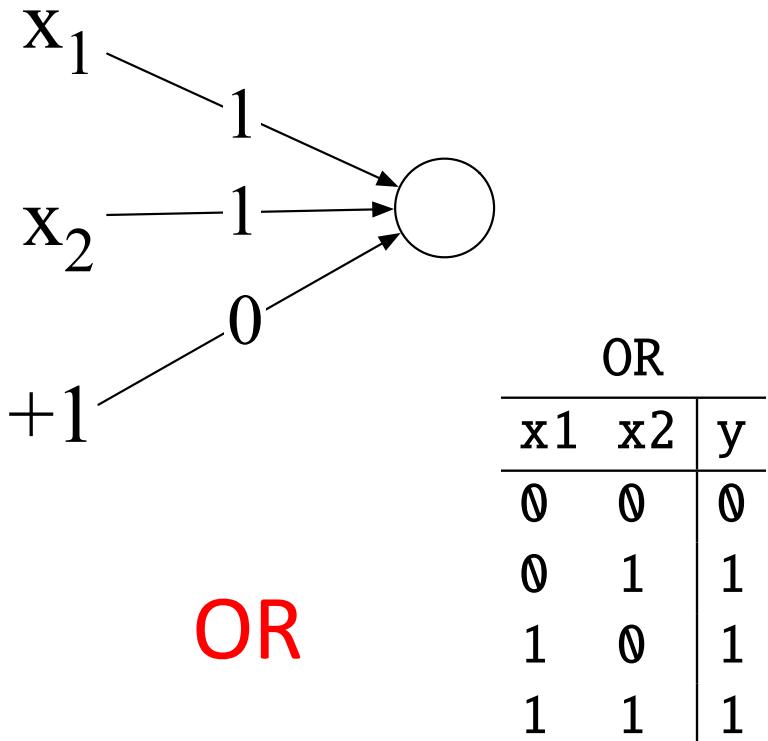
# Easy to build AND or OR with perceptrons

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



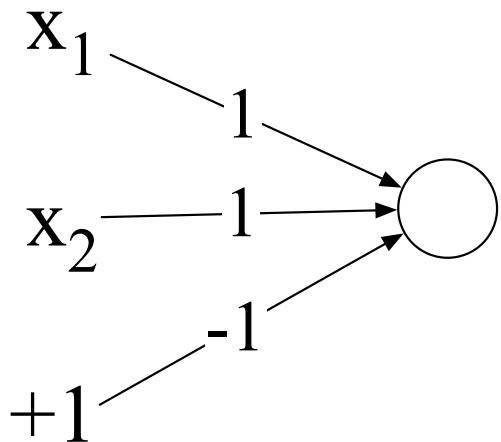
AND

		AND
x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1



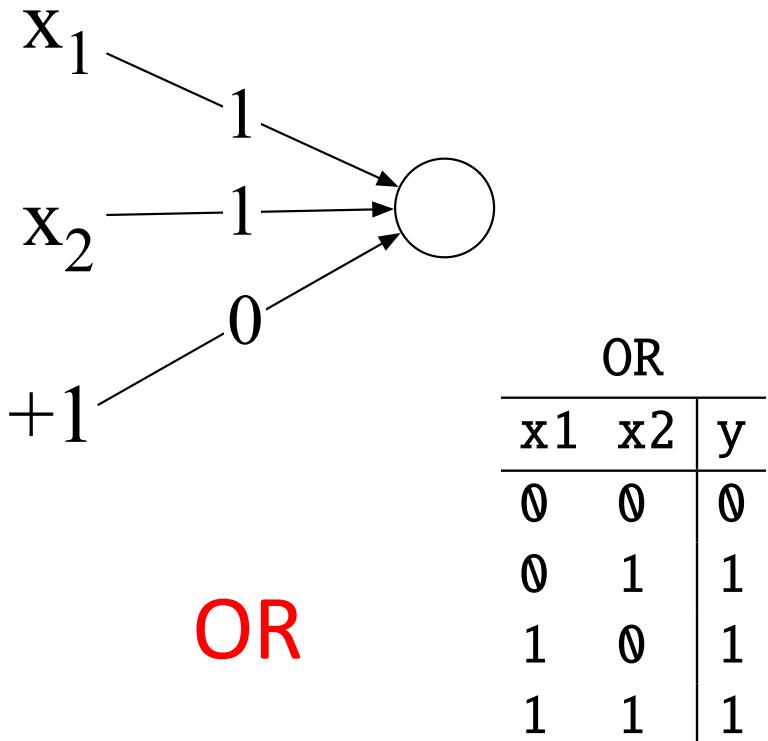
# Easy to build AND or OR with perceptrons

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



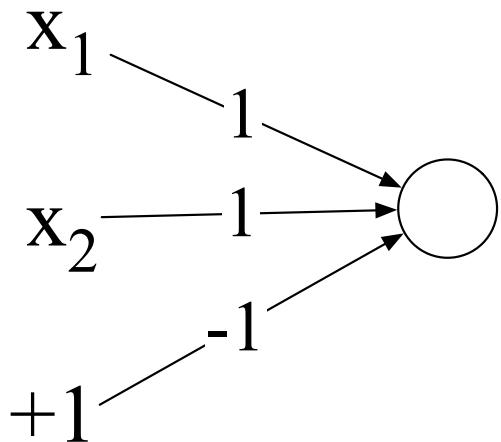
AND

		AND
x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1



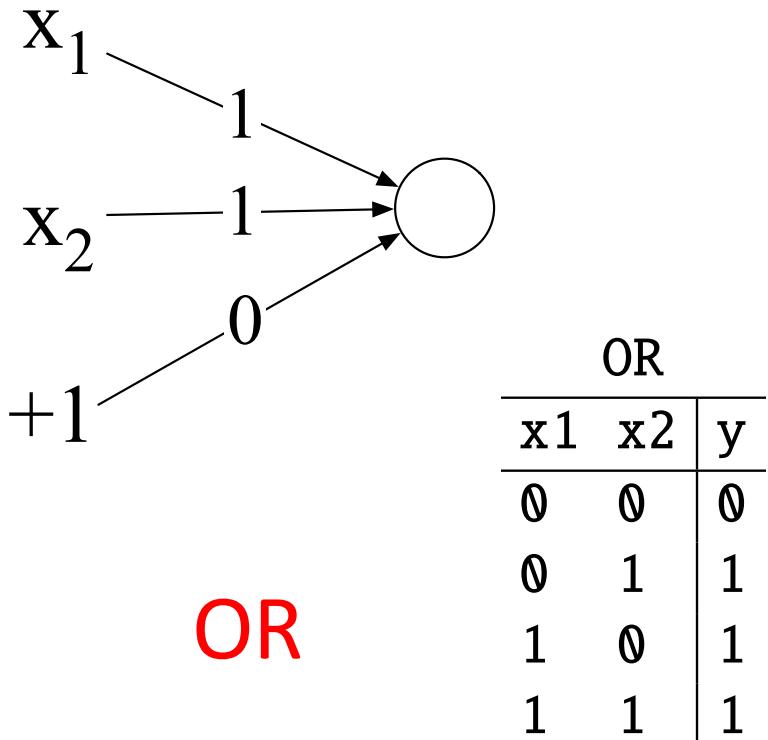
# Easy to build AND or OR with perceptrons

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



AND

		AND
x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1



Not possible to capture XOR with perceptrons

Pause the lecture and try for yourself!

# Why? Perceptrons are linear classifiers

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

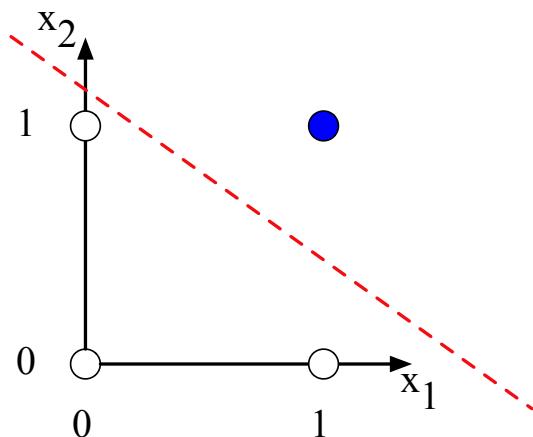
$$w_1x_1 + w_2x_2 + b = 0$$

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

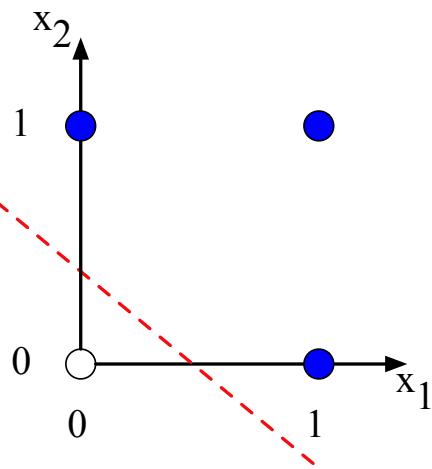
This line acts as a **decision boundary**

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

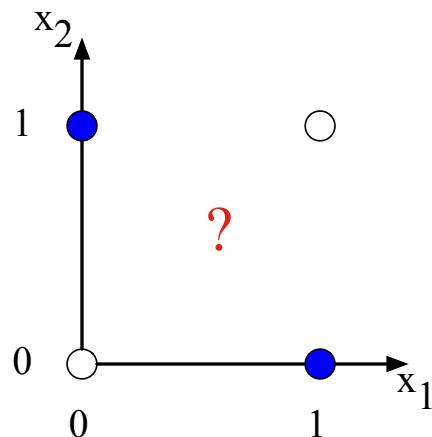
# Decision boundaries



a)  $x_1$  AND  $x_2$



b)  $x_1$  OR  $x_2$



c)  $x_1$  XOR  $x_2$

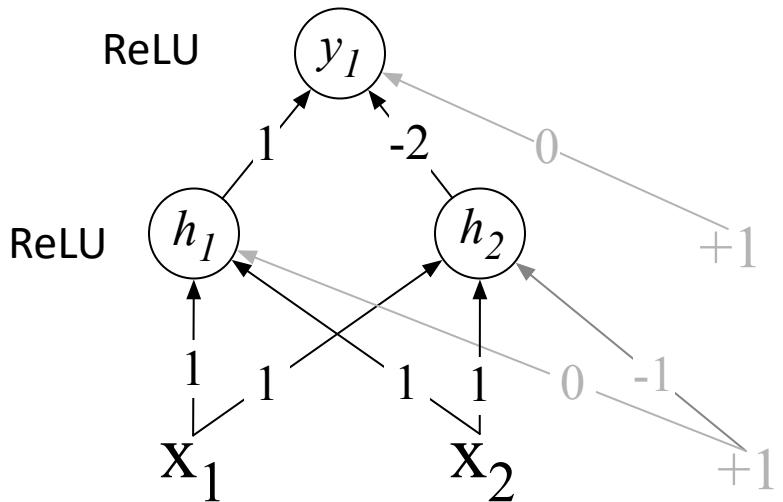
XOR is not a **linearly separable** function!

# Solution to the XOR problem

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

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

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

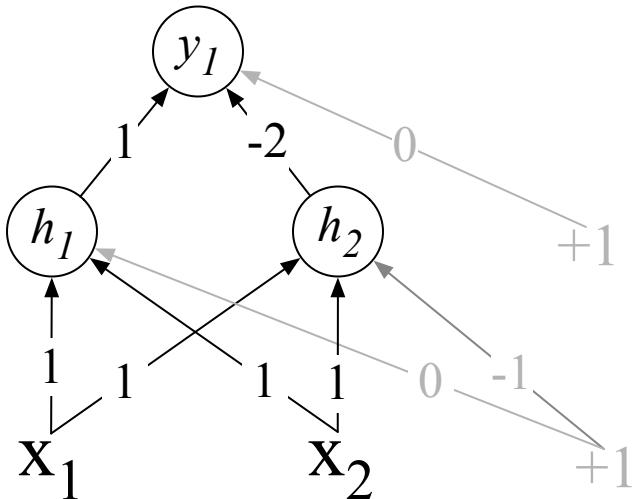


# Solution to the XOR problem

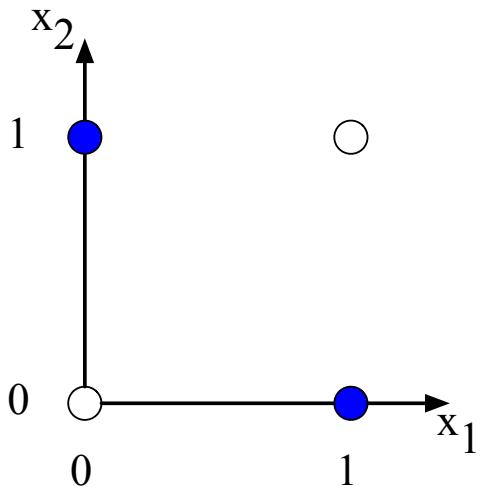
XOR **can't** be calculated by a single perceptron

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

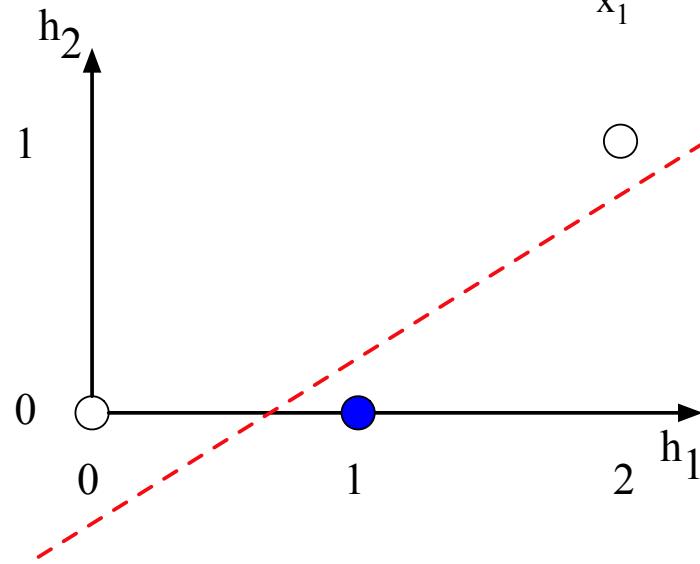
XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0



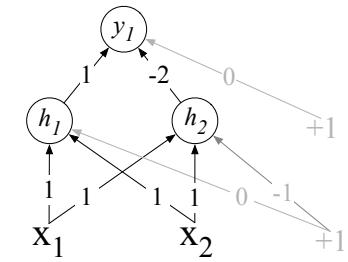
# The hidden representation $h$



a) The original  $x$  space



b) The new (linearly separable)  $h$  space



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

# Simple Neural Networks and Neural Language Models

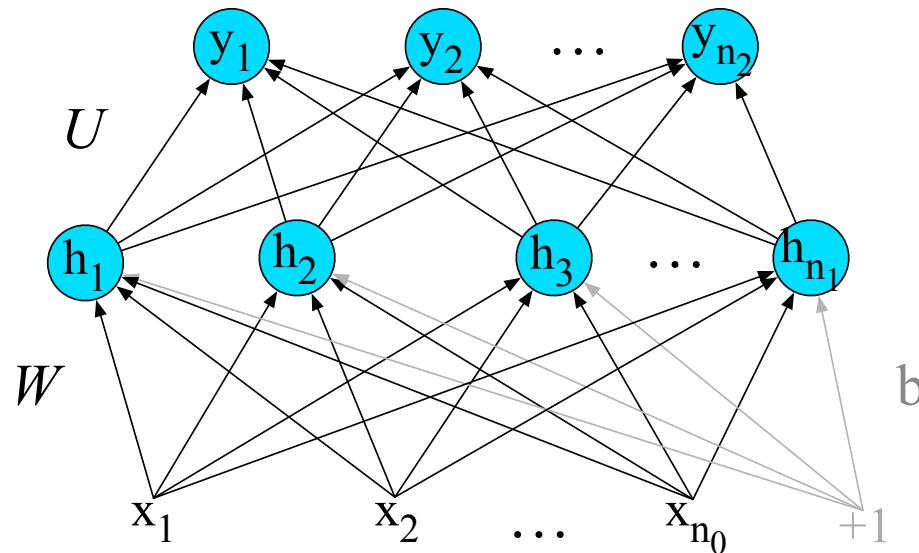
## The XOR problem

# Simple Neural Networks and Neural Language Models

## Feedforward Neural Networks

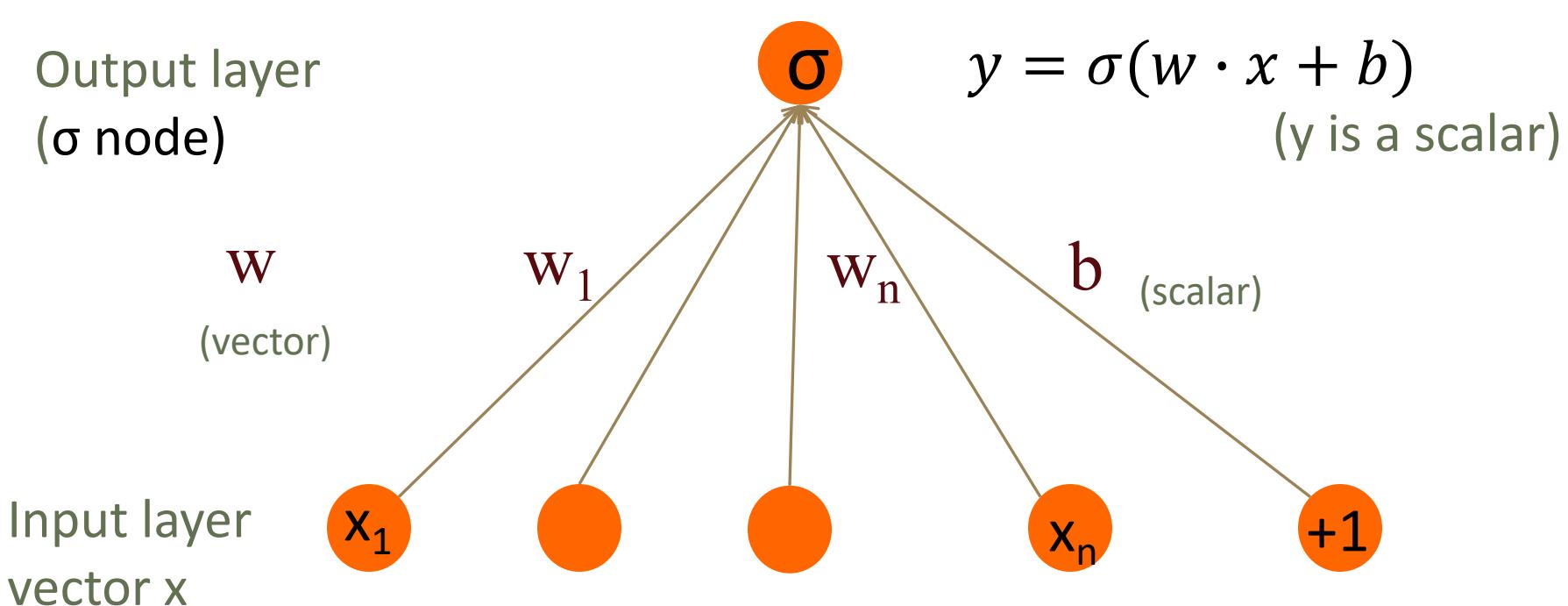
# Feedforward Neural Networks

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



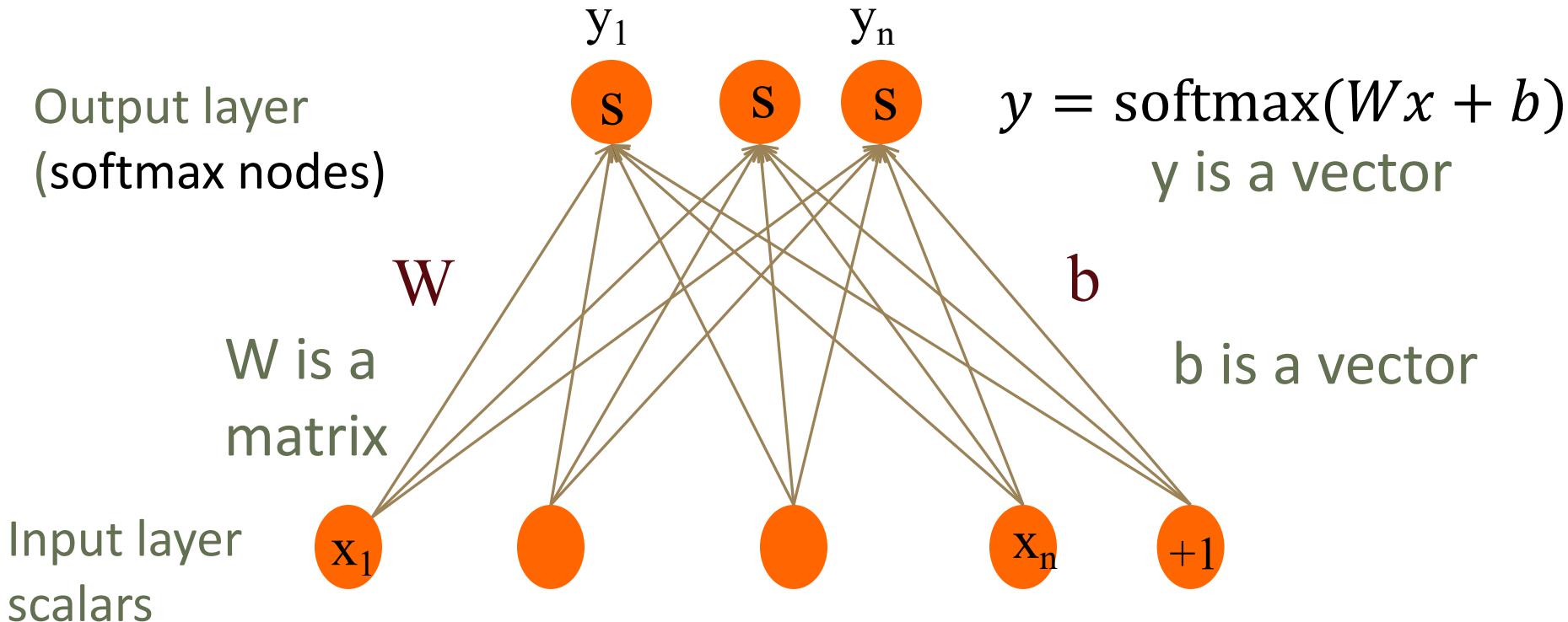
# Binary Logistic Regression as a 1-layer Network

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



# Multinomial Logistic Regression as a 1-layer Network

Fully connected single layer network



Reminder: softmax: a generalization of sigmoid

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

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

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

Example:

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

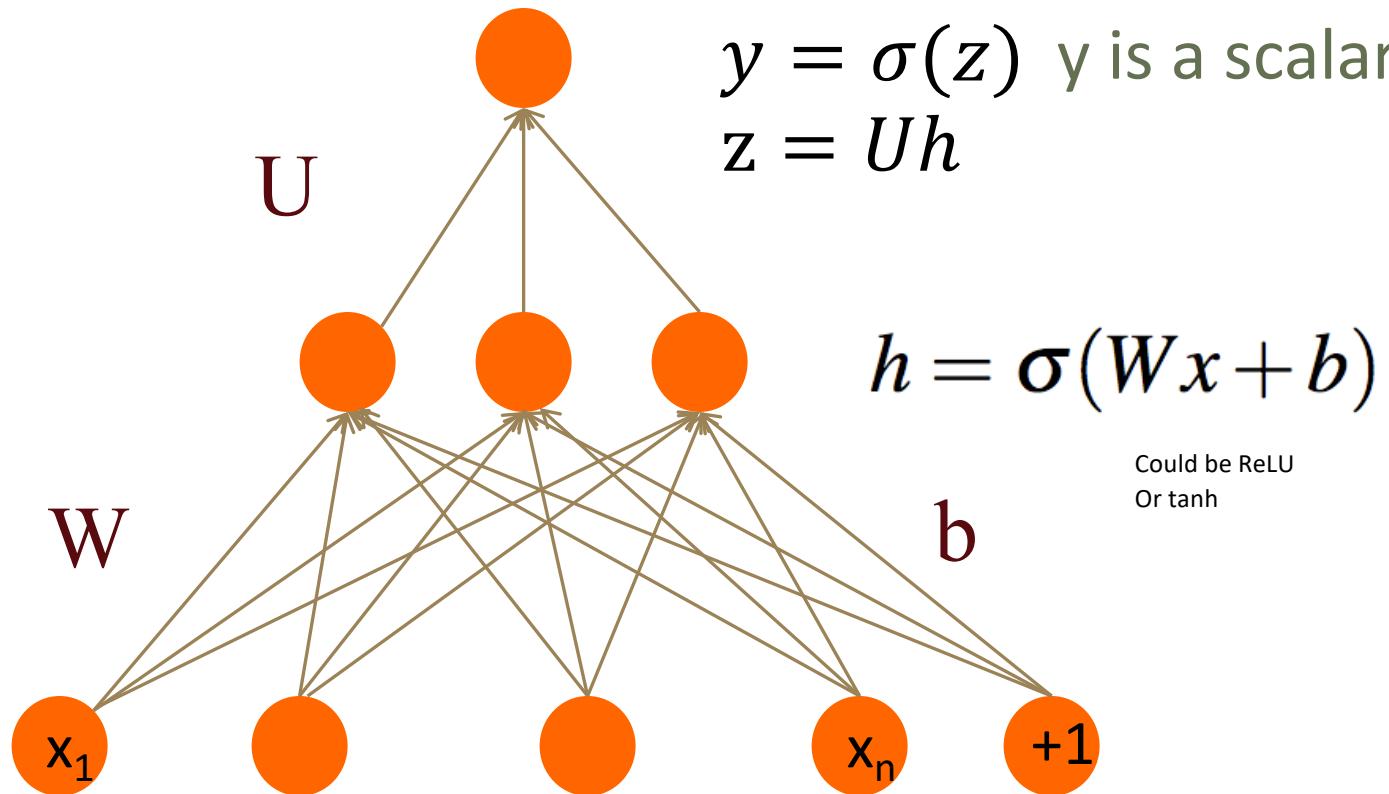
$$\text{softmax}(z) = [0.055, 0.090, 0.006, 0.099, 0.74, 0.010]$$

# Two-Layer Network with scalar output

Output layer  
( $\sigma$  node)

hidden units  
( $\sigma$  node)

Input layer  
(vector)

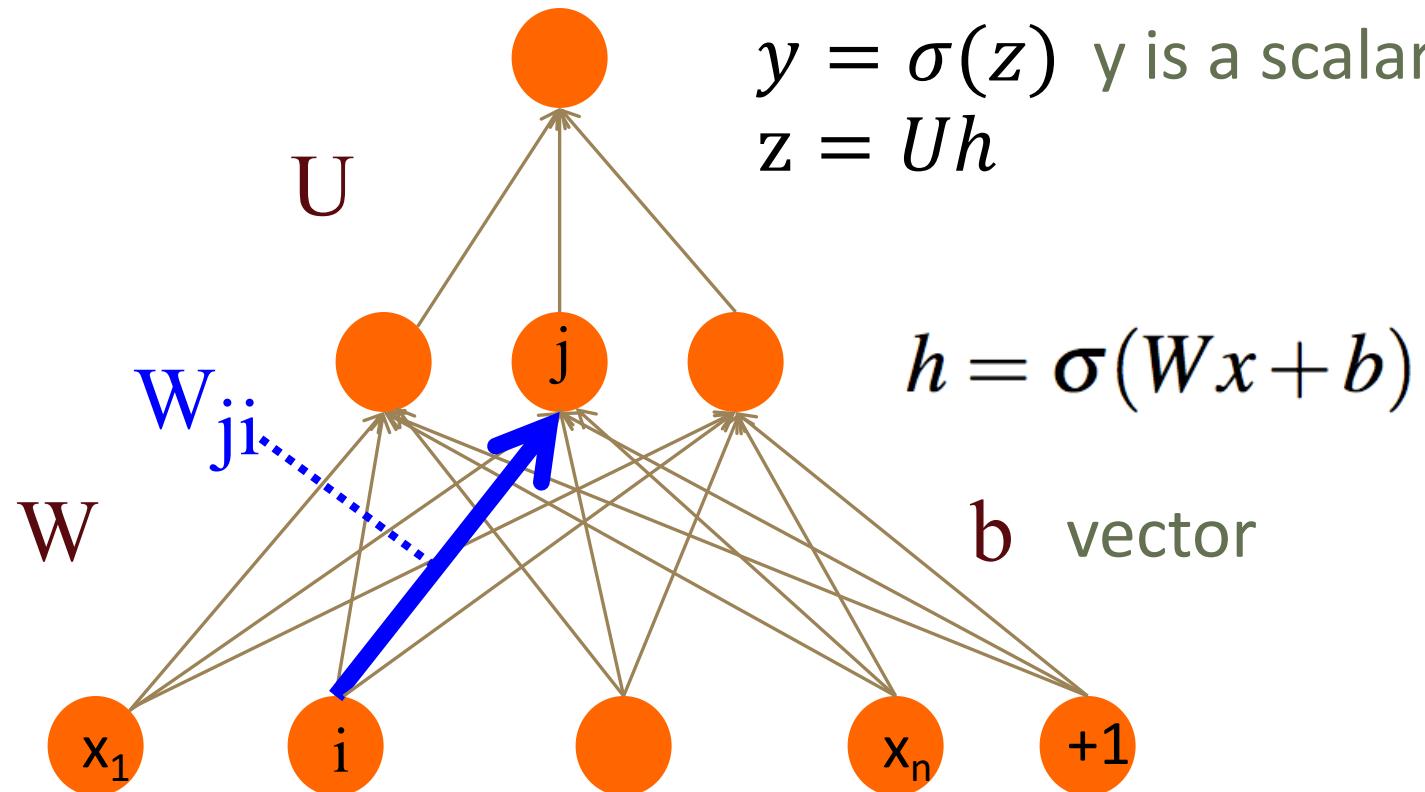


# Two-Layer Network with scalar output

Output layer  
( $\sigma$  node)

hidden units  
( $\sigma$  node)

Input layer  
(vector)

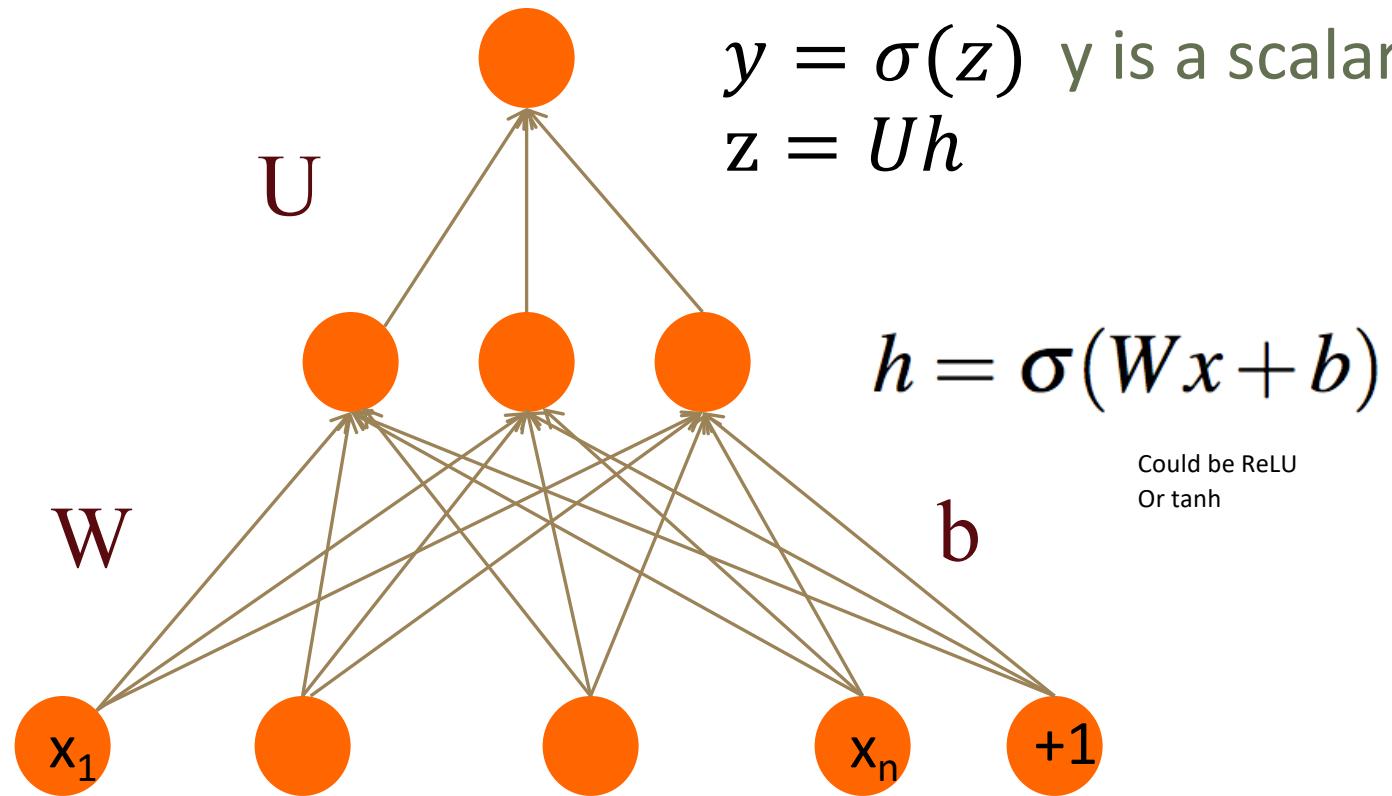


# Two-Layer Network with scalar output

Output layer  
( $\sigma$  node)

hidden units  
( $\sigma$  node)

Input layer  
(vector)

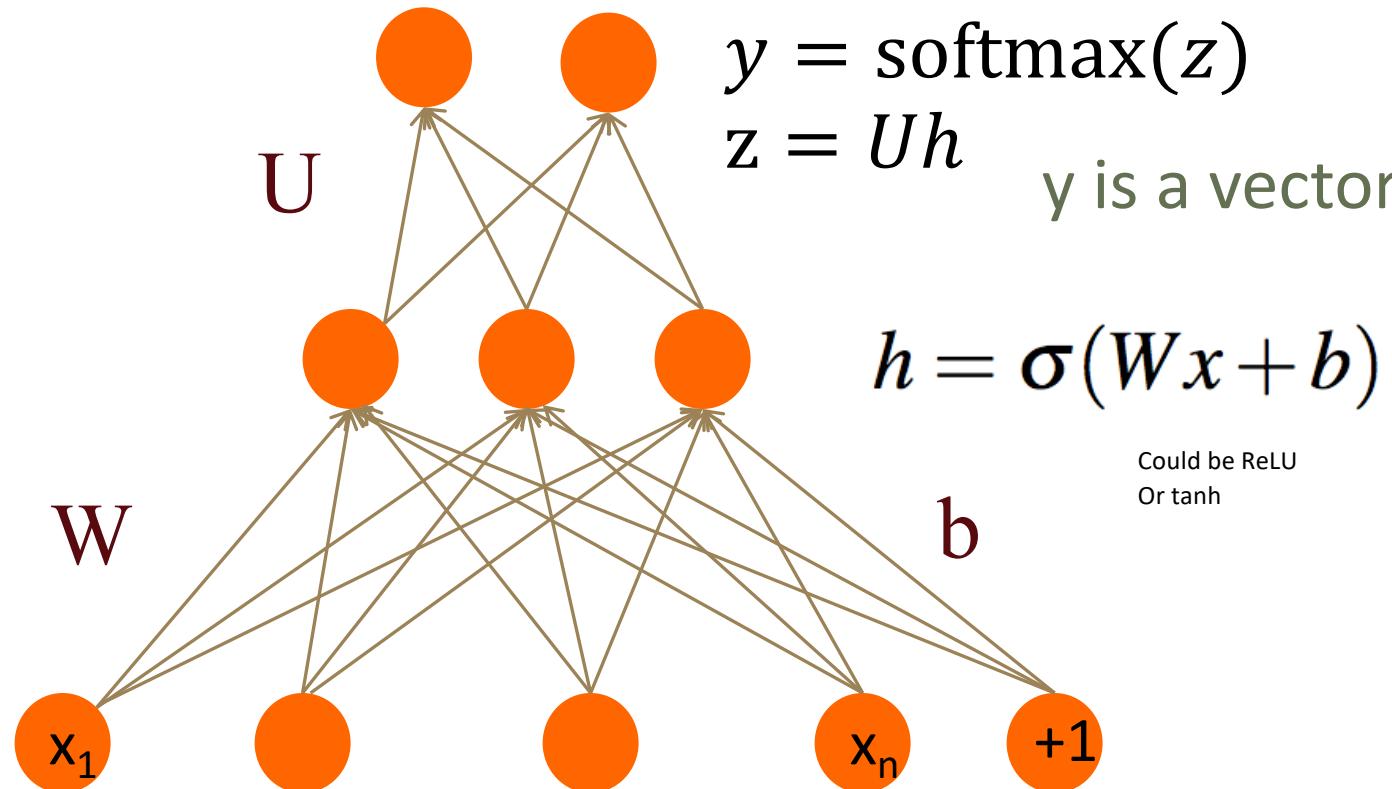


# Two-Layer Network with softmax output

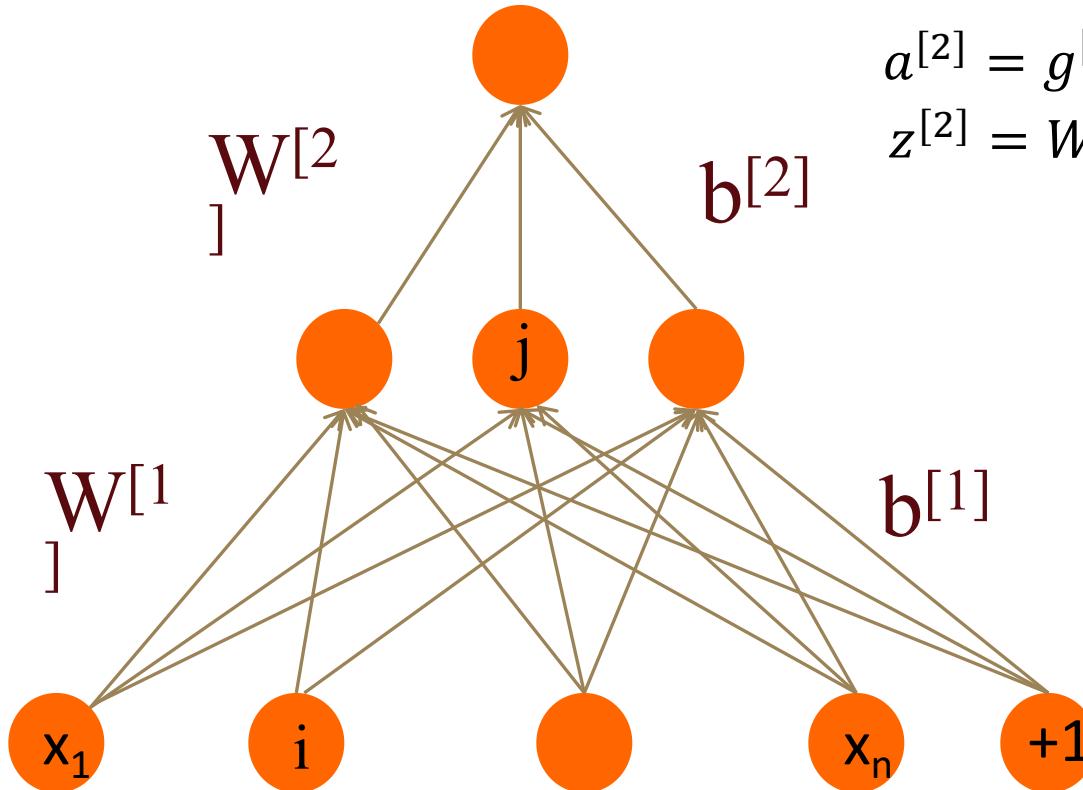
Output layer  
( $\sigma$  node)

hidden units  
( $\sigma$  node)

Input layer  
(vector)



# Multi-layer Notation



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

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

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

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

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

$$a^{[0]}$$

# Multi Layer Notation

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

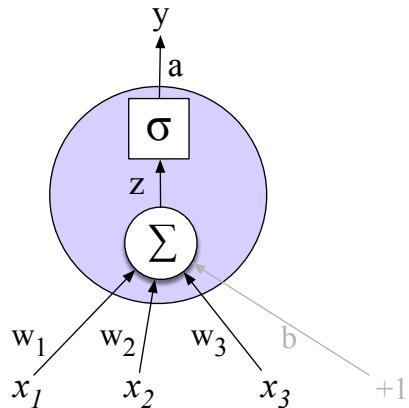
$$a^{[1]} = g^{[1]}(z^{[1]})$$

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

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

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

**for  $i$  in 1..n**

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


# Replacing the bias unit

Let's switch to a notation without the bias unit

Just a notational change

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

# Replacing the bias unit

Instead of:

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

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

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

We'll do this:

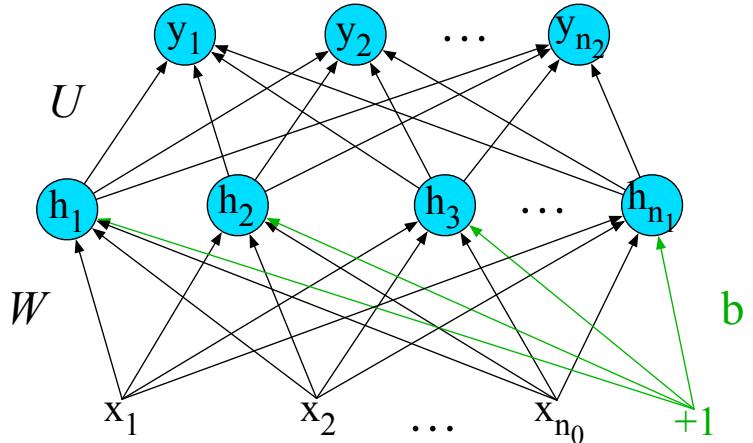
$$x = x_0, x_1, x_2, \dots, x_{n0}$$

$$h = \sigma(Wx)$$

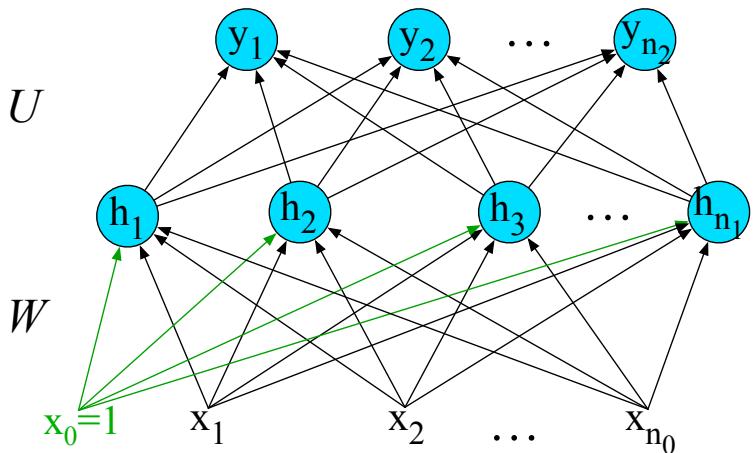
$$\sigma \left( \sum_{i=0}^{n_0} W_{ji} x_i \right)$$

# Replacing the bias unit

Instead of:



We'll do this:



# Simple Neural Networks and Neural Language Models

## Feedforward Neural Networks

# Simple Neural Networks and Neural Language Models

## Applying feedforward networks to NLP tasks

# Use cases for feedforward networks

Let's consider 2 (simplified) sample tasks:

1. Text classification
2. Language modeling

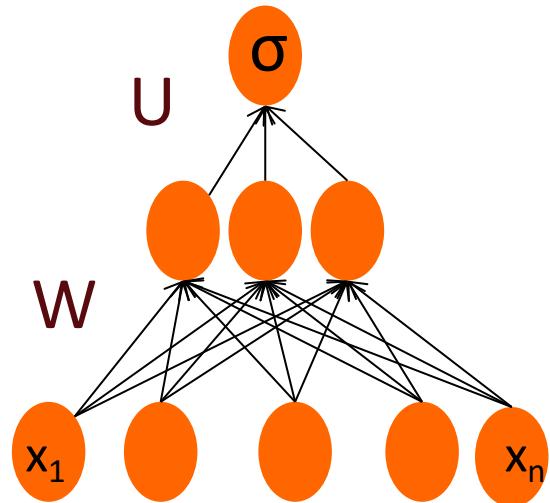
State of the art systems use more powerful neural architectures, but simple models are useful to consider!

# Classification: Sentiment Analysis

We could do exactly what we did with logistic regression

Input layer are binary features as before

Output layer is 0 or 1

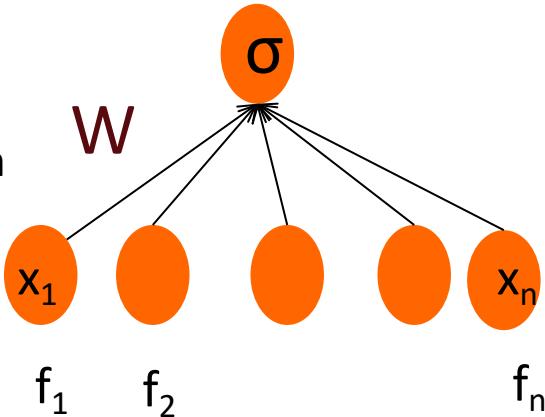


# Sentiment Features

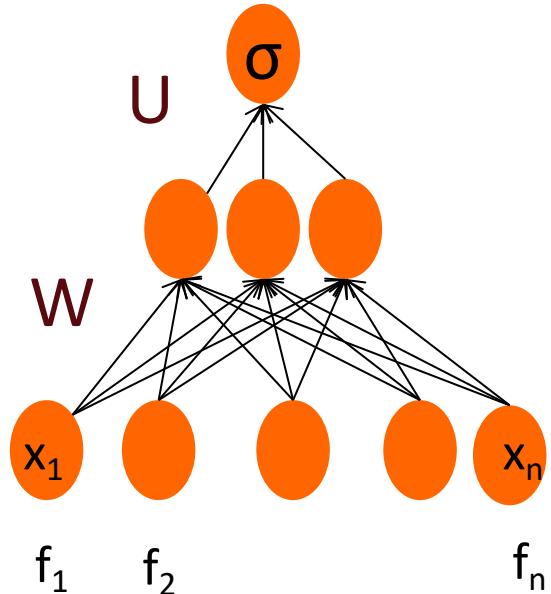
Var	Definition
$x_1$	count(positive lexicon) $\in$ doc)
$x_2$	count(negative lexicon) $\in$ doc)
$x_3$	$\begin{cases} 1 & \text{if “no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
$x_4$	count(1st and 2nd pronouns $\in$ doc)
$x_5$	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise}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

# Feedforward nets for simple classification

Logistic  
Regression



2-layer  
feedforward  
network



Just adding a hidden layer to logistic regression

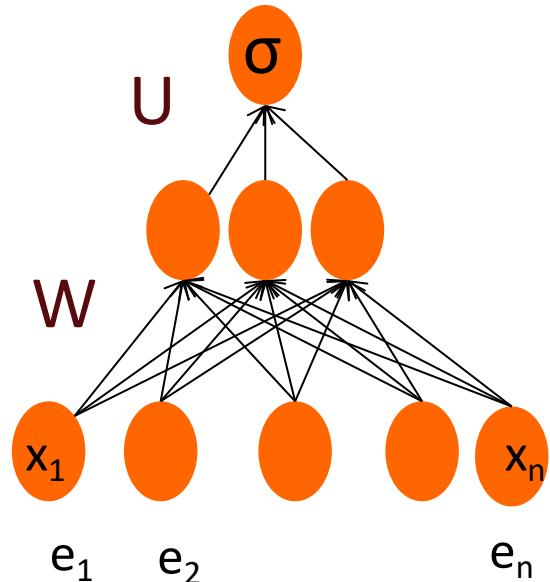
- allows the network to use non-linear interactions between features
- which may (or may not) improve performance.

# Even better: representation learning

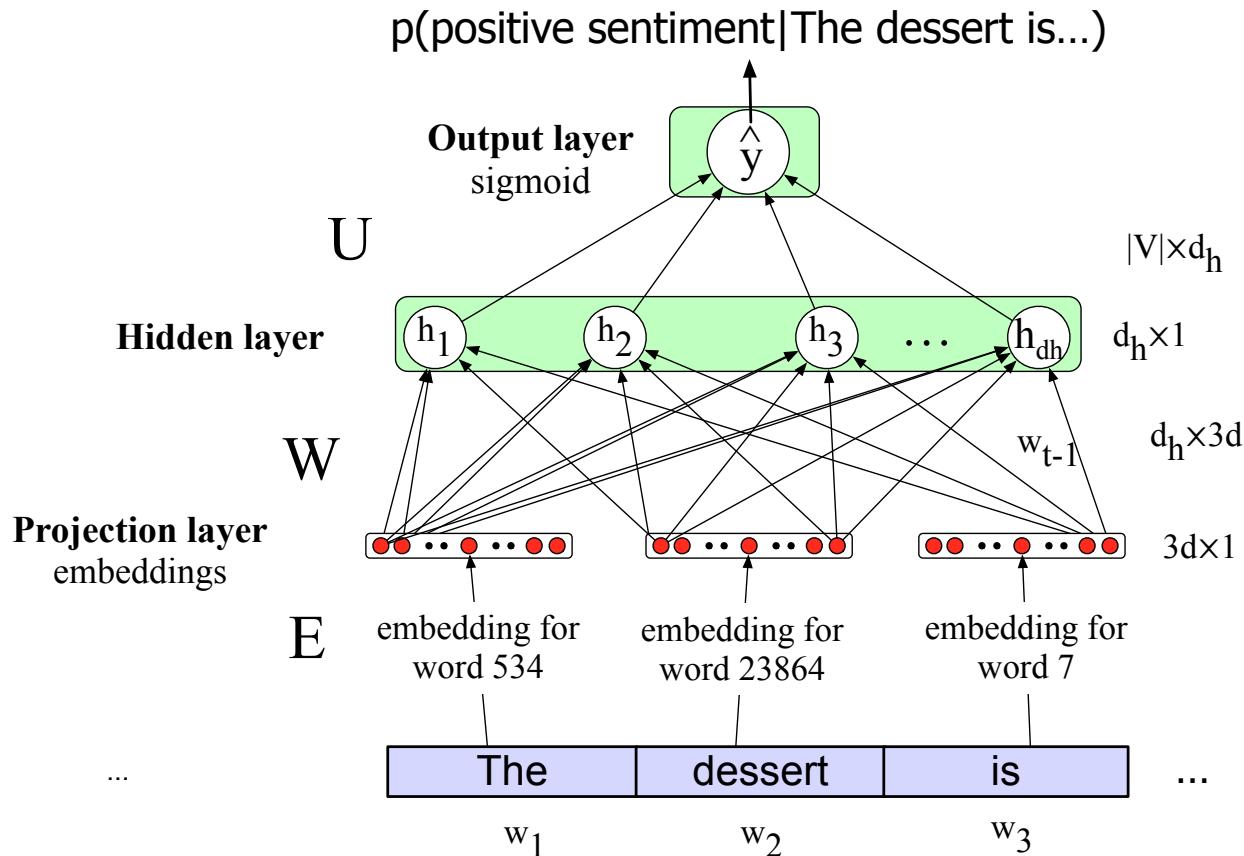
The real power of deep learning comes from the ability to **learn** features from the data

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

Use learned representations like embeddings!



# Neural Net Classification with embeddings as input features!



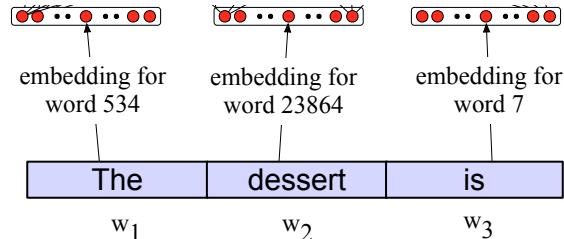
# Issue: texts come in different sizes

This assumes a fixed size length (3)!

Kind of unrealistic.

Some simple solutions (more sophisticated solutions later)

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

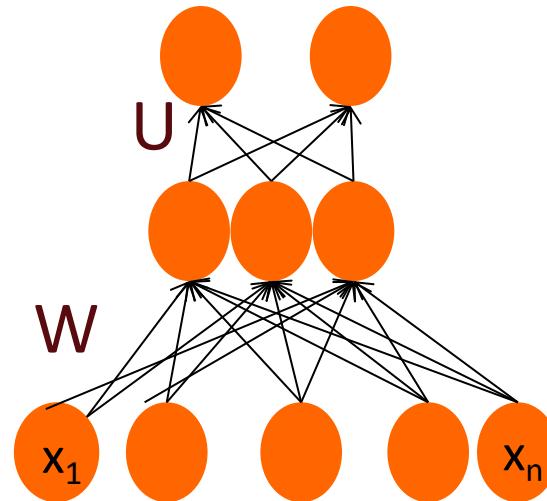


# Reminder: Multiclass Outputs

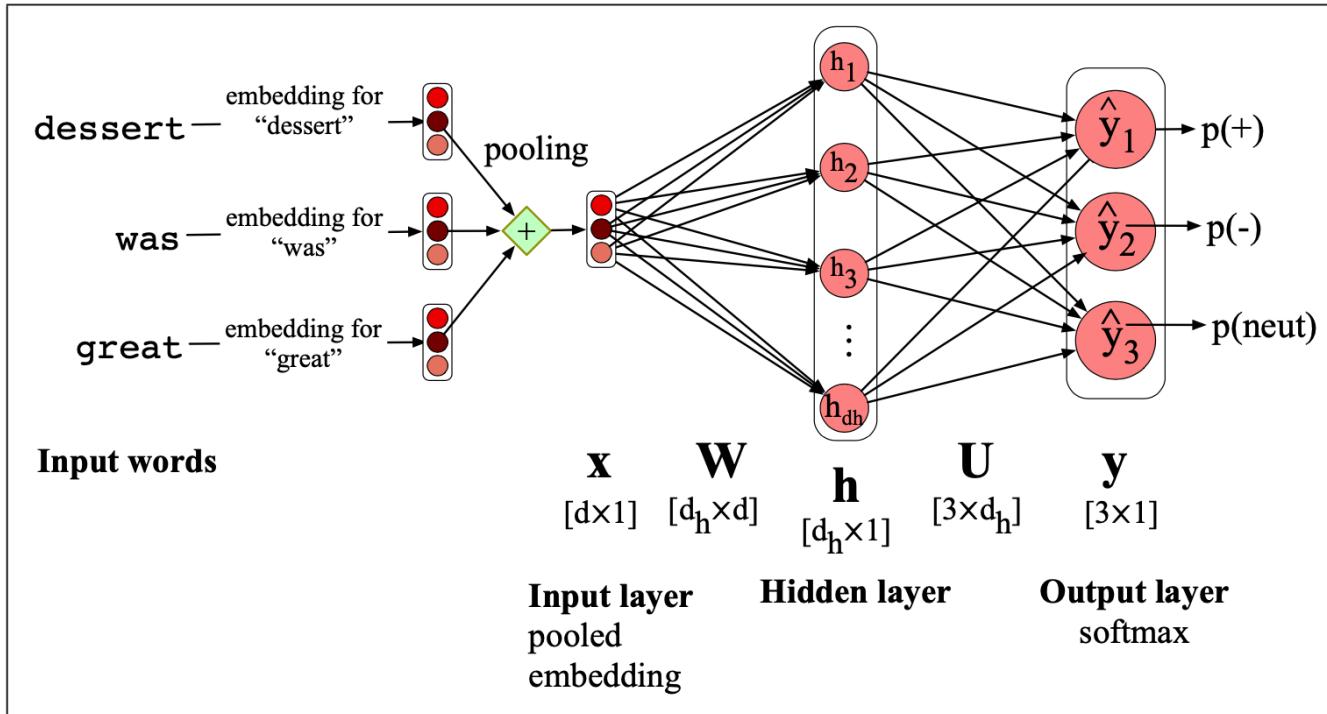
What if you have more than two output classes?

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

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



# Pooling



# Pooling: Vectors and matrices for a single example

$$x = \text{mean}(e(w_1), e(w_2), \dots, e(w_n))$$

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

$$z = Uh$$

$$\hat{y} = \text{softmax}(z)$$

(7.21)

# Generalizing to $m$ test examples

- Pack all input feature vectors for each input  $\mathbf{x}$  into a single matrix  $\mathbf{X}$ , with each row  $i$  a row vector consisting of the pooled embedding for input example  $\mathbf{x}^{(i)}$  (i.e. the vector  $\mathbf{x}^{(i)}$ ).
- If the dimensionality of the pooled input embedding is  $d$ ,  $\mathbf{X}$  will be a matrix of shape  $[m \times d]$ .

$$\mathbf{H} = \sigma(\mathbf{X}\mathbf{W}^T + \mathbf{b})$$

$$\mathbf{Z} = \mathbf{H}\mathbf{U}^T$$

$$\hat{\mathbf{Y}} = \text{softmax}(\mathbf{Z})$$

(7.22)

with the following shapes:

$\mathbf{X} : [m \times d]$ ,  $\mathbf{W} : [d_h \times d]$ ,  $\mathbf{H} : [m \times d_h]$ ,  $\mathbf{U} : 3 \times d_h$ ,  $\mathbf{Z} : [m \times 3]$ ,  $\hat{\mathbf{Y}} : [m \times 3]$

# Neural Language Models (LMs)

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

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

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

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

# Simple feedforward Neural Language Models

**Task:** predict next word  $w_t$

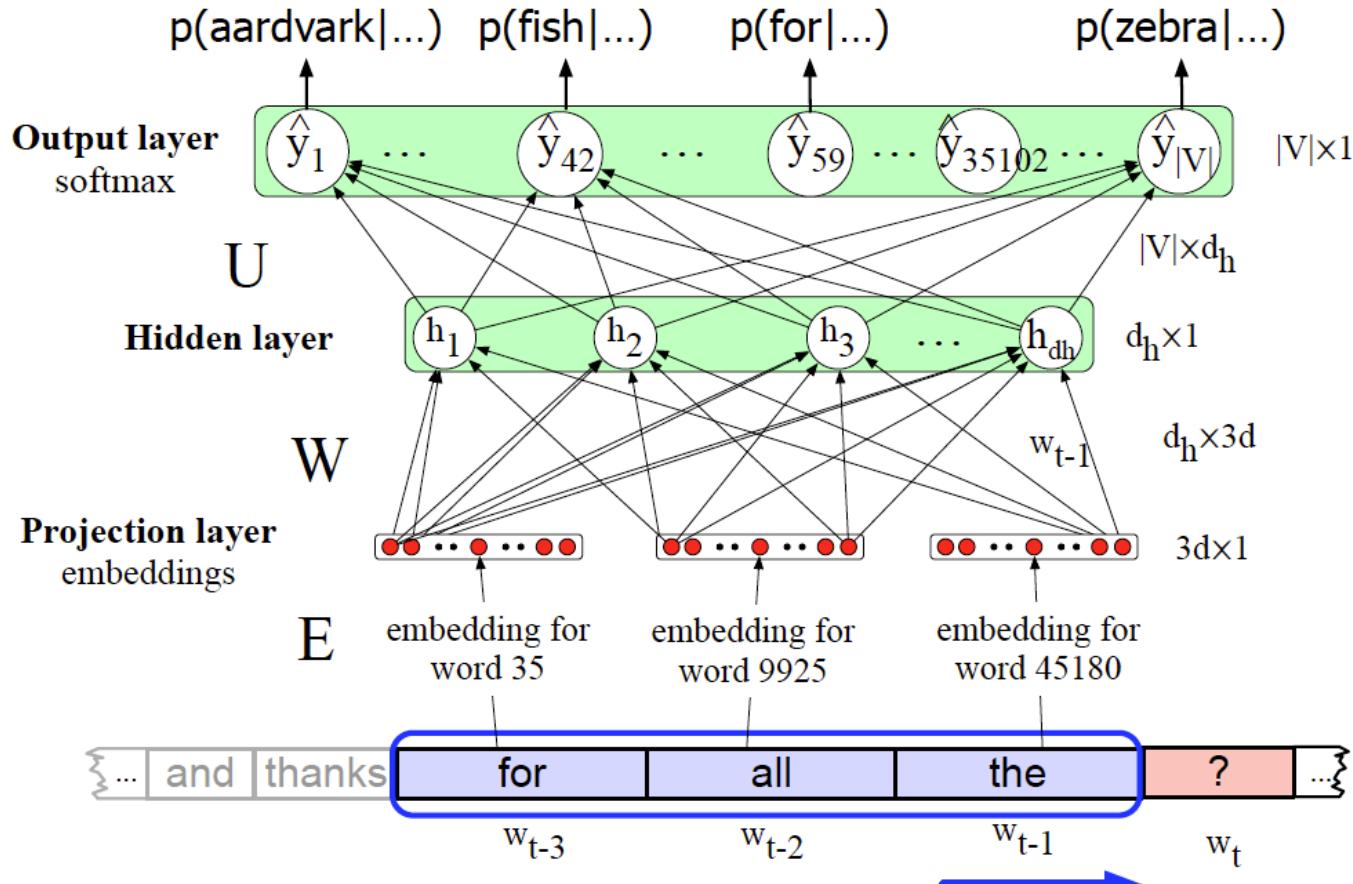
given prior words  $w_{t-1}, w_{t-2}, w_{t-3}, \dots$

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

**Solution:** Sliding windows (of fixed length)

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

# Neural Language Model



# Why Neural LMs work better than N-gram LMs

## Training data:

We've seen: I have to make sure that the cat gets fed.

Never seen: dog gets fed

## Test data:

I forgot to make sure that the dog gets \_\_

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

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

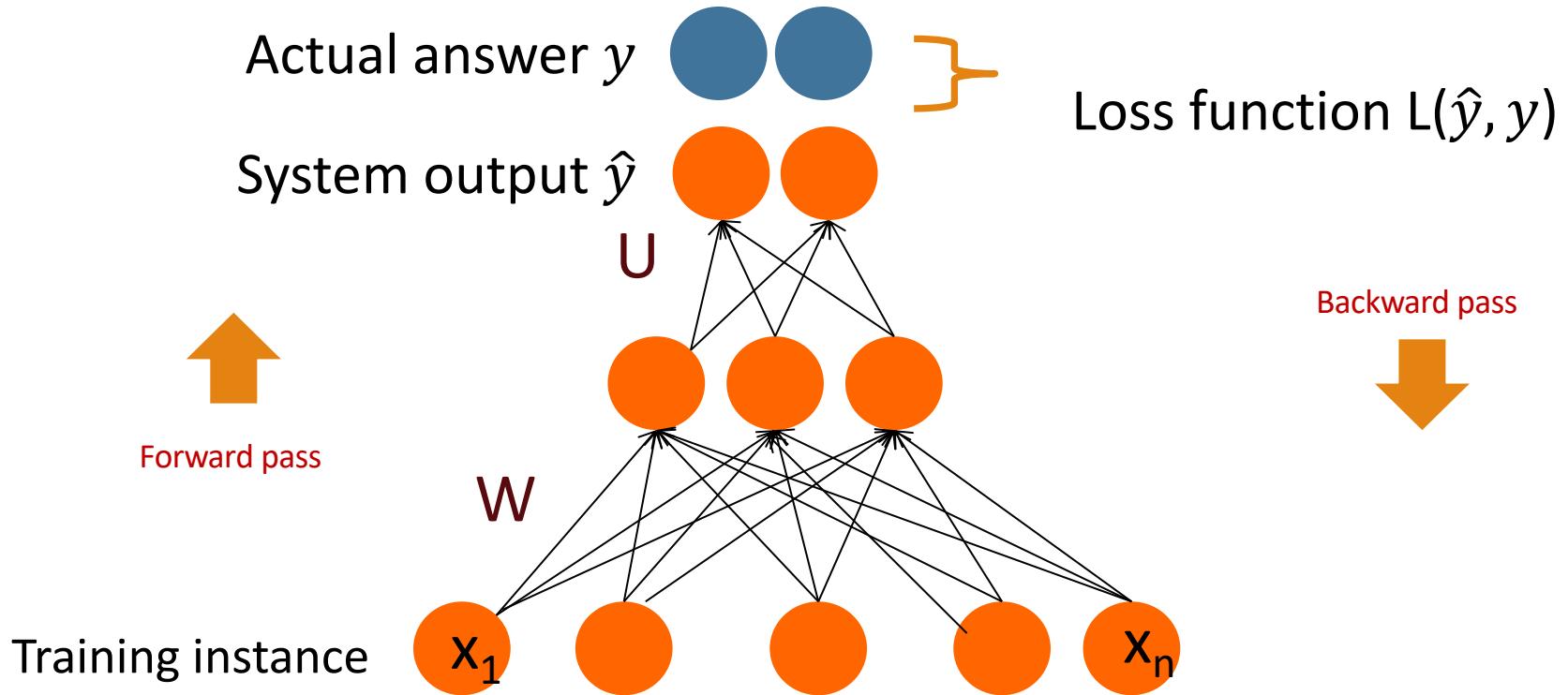
# Simple Neural Networks and Neural Language Models

## Applying feedforward networks to NLP tasks

# Simple Neural Networks and Neural Language Models

## Training Neural Nets: Overview

# Intuition: training a 2-layer Network



# Intuition: Training a 2-layer network

For every training tuple  $(x, y)$

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

# Reminder: Loss Function for binary logistic regression

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

Cross entropy loss for logistic regression:

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

# Reminder: gradient descent for weight updates

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

To tell us how to adjust weights for each training item

- Move them in the opposite direction of the gradient

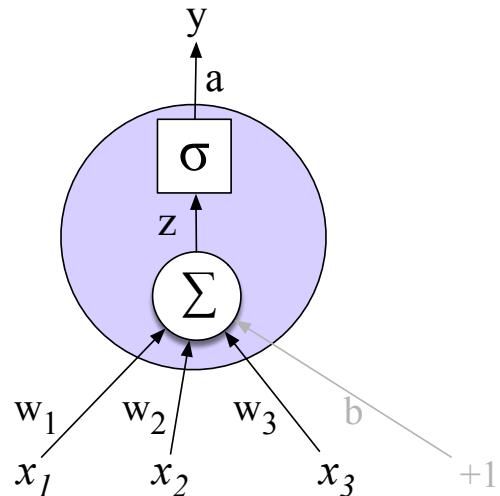
$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x; w), y)$$

- For logistic regression

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

# Where did that derivative come from?

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



Derivative of the weighted sum

Derivative of the Activation

Derivative of the Loss

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

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

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

What about deeper networks?

- Lots of layers, different activation functions?

Solution in the next lecture:

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

# Simple Neural Networks and Neural Language Models

## Training Neural Nets: Overview

Simple Neural  
Networks and  
Neural  
Language  
Models

Computation Graphs and  
Backward Differentiation

# Why Computation Graphs

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

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

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

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

# Computation Graphs

A computation graph represents the process of computing a mathematical expression

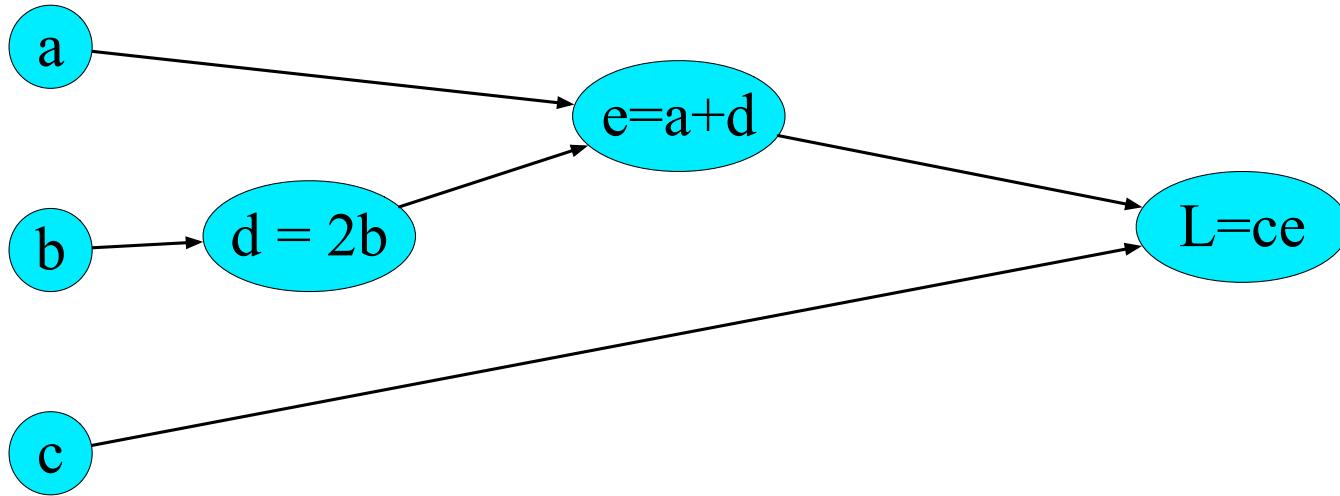
Example:  $L(a,b,c) = c(a + 2b)$

$$d = 2 * b$$

Computations:

$$e = a + d$$

$$L = c * e$$



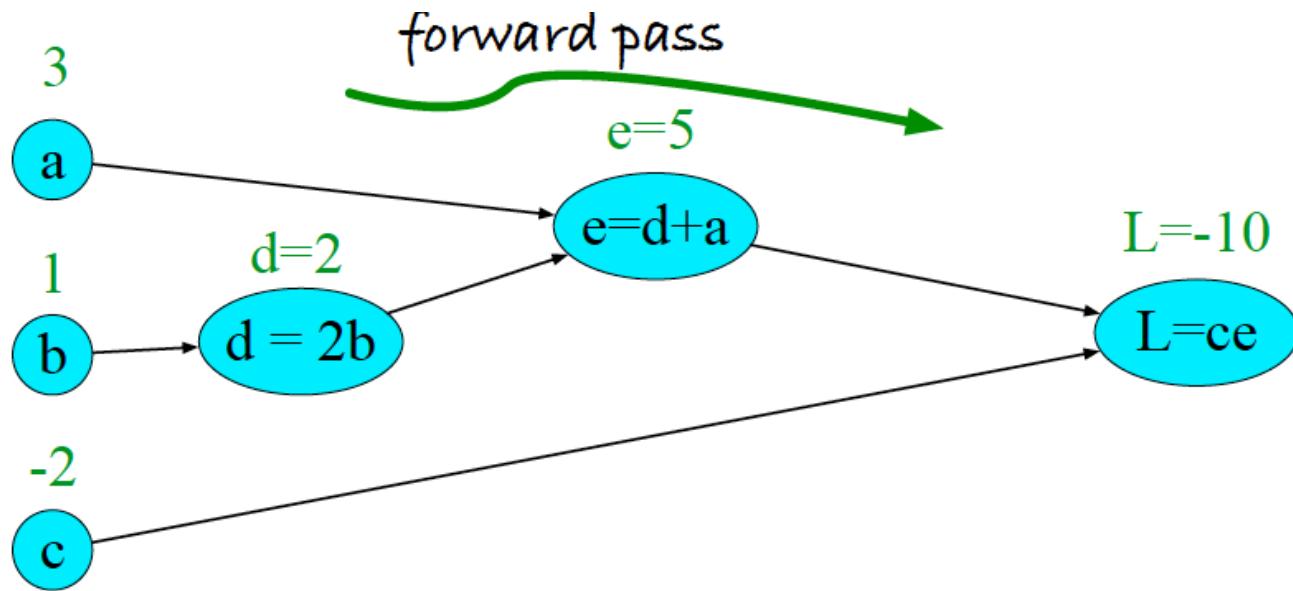
Example:  $L(a, b, c) = c(a + 2b)$

$$d = 2 * b$$

Computations:

$$e = a + d$$

$$L = c * e$$



# Backwards differentiation in computation graphs

The importance of the computation graph comes from the backward pass

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

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

$$d = 2 * b$$

$$e = a + d$$

$$L = c * e$$

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

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

# The chain rule

Computing the derivative of a composite function:

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

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

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

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

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

$$d = 2 * b$$

$$e = a + d$$

$$L = c * e$$

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

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

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

## Example

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

$$d = 2 * b$$

$$e = a + d$$

$$L = c * e$$

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

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

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

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

# Example

$$a=3$$



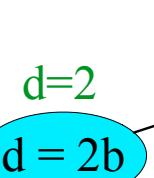
$$b=1$$



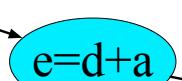
$$c=-2$$



$$d = 2b$$



$$e=5$$



$$L=-10$$



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

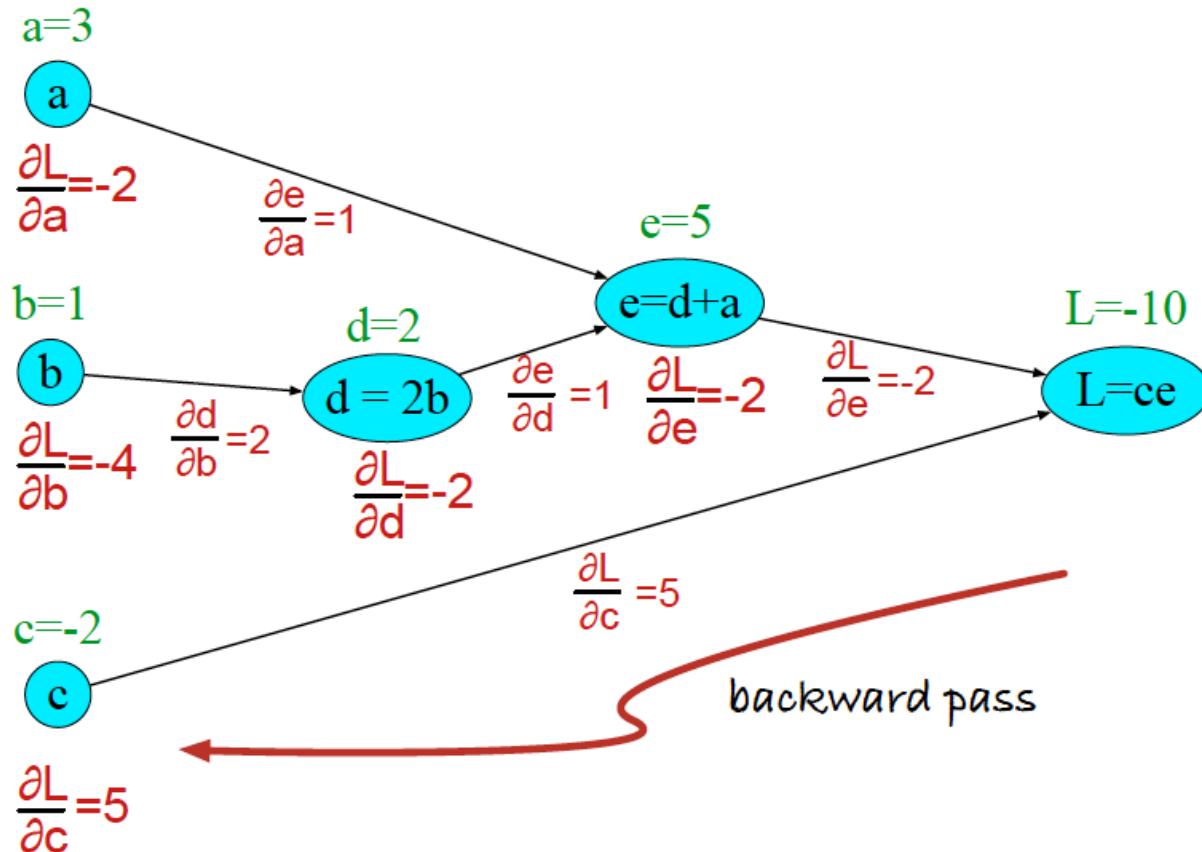
$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e} \frac{\partial e}{\partial d} \frac{\partial d}{\partial b}$$

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

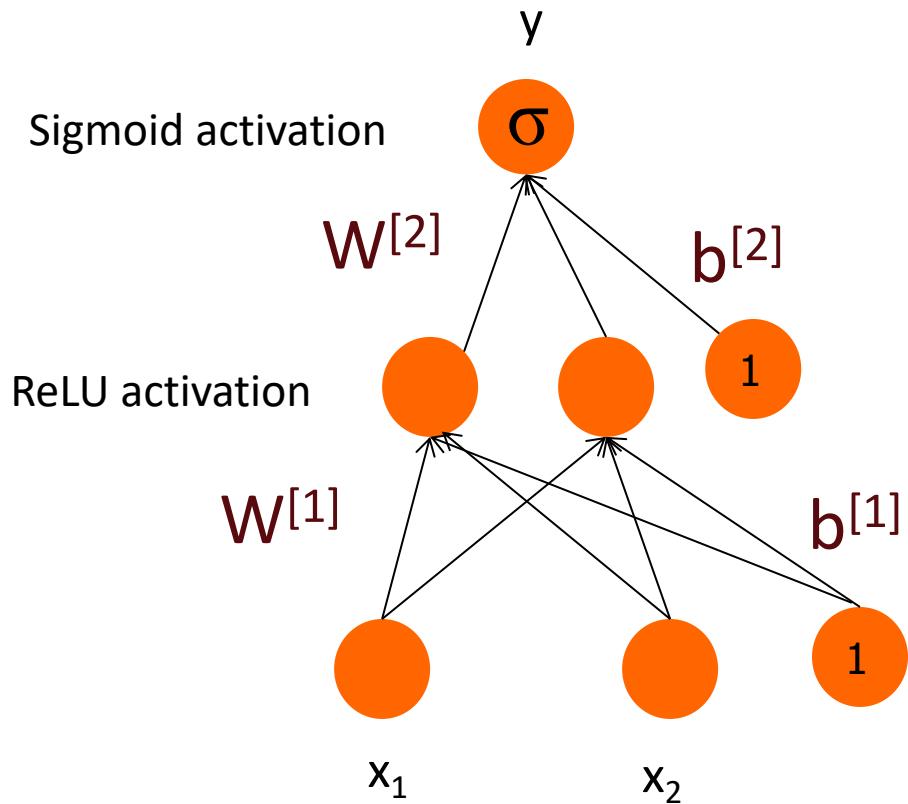
$$e=a+d : \quad \frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial d} = 1$$

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

# Example



# Backward differentiation on a two layer network



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

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

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

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

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

# Backward differentiation on a two layer network

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

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

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

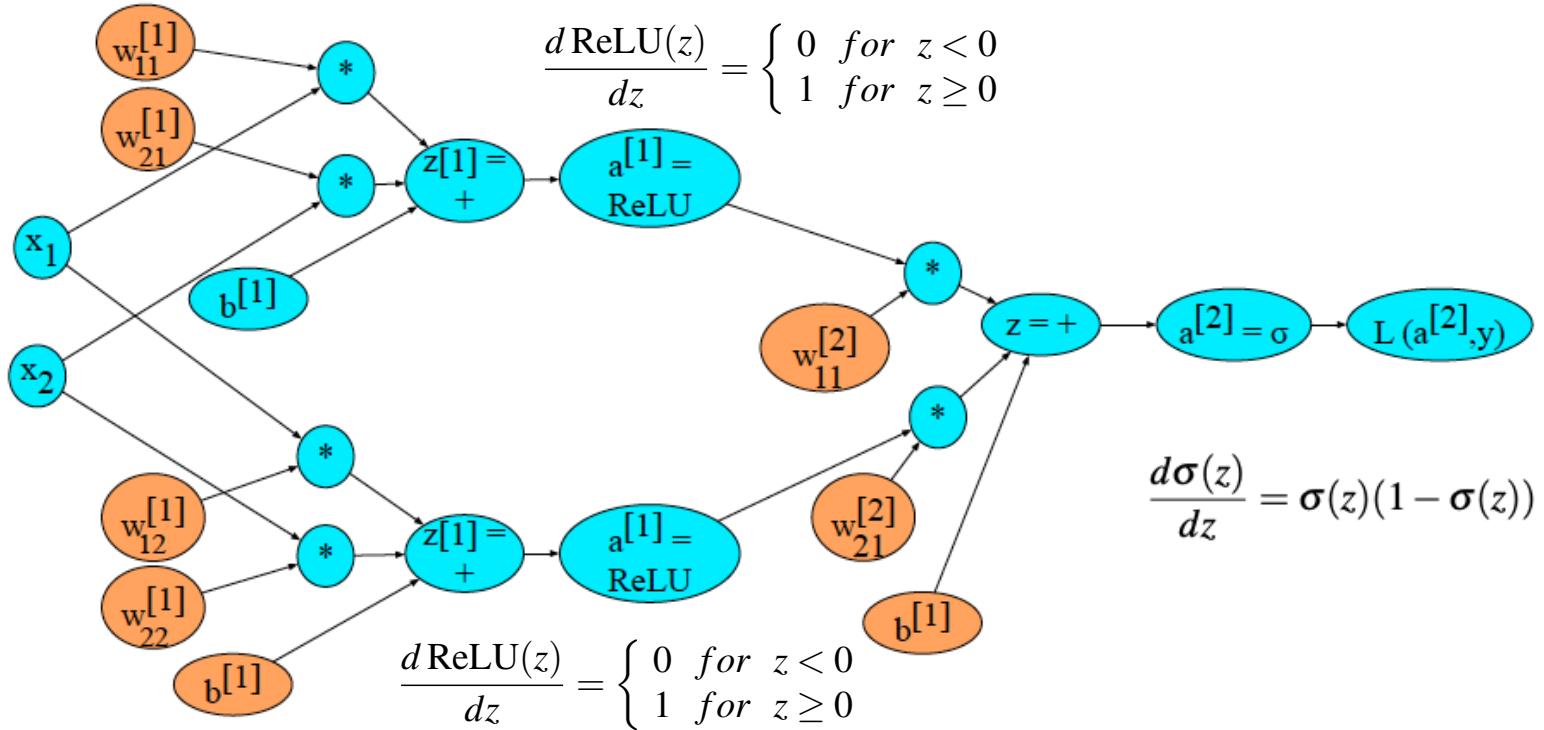
$$a^{[2]} = \sigma(z^{[2]})$$

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

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

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

# Backward differentiation on a 2-layer network



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

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

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

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

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

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

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

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

# Summary

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

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

Solution: **backward differentiation**

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

Simple Neural  
Networks and  
Neural  
Language  
Models

Computation Graphs and  
Backward Differentiation

# **Training Neural Networks**

Erhard Hinrichs

Seminar für Sprachwissenschaft  
Eberhard-Karls Universität Tübingen

# Loss function for a single training example $x$

## 1. Binomial (binary) case

$$L_{CE}(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})] \quad (23)$$

**Remark:** (23) is the cross-entropy loss function  $L_{CE}$  for binary logistic regression; see formula (5.23) in Chapter 5.

$$L_{CE}(\hat{y}, y) = -\log \hat{y}_c \quad (\text{where } c \text{ is the correct class}) \quad (25)$$

**Remark:** (25) is called the *negative log likelihood loss*. (25) is equivalent to (23) for the correct class  $y_c = 1$  since the first summand in (25) equals  $-\log \hat{y}_i$ , and the second summand in (23) equals 0.

# Loss function for a single training example $x$

## 2. Multinomial case with $K$ classes

$$L_{CE}(\hat{\mathbf{y}}, \mathbf{y}) = - \sum_{k=1}^K \mathbf{y}_k \log \hat{\mathbf{y}}_k = - \sum_{k=1}^K \mathbb{1}\{\mathbf{y}_k = 1\} \log \hat{\mathbf{y}}_k \quad (24)$$

$$L_{CE}(\hat{\mathbf{y}}, \mathbf{y}) = - \log \frac{\exp(\mathbf{z}_c)}{\sum_{j=1}^K \exp(\mathbf{z}_j)} \quad (\text{where } c \text{ is the correct class}) \quad (26)$$

### Remarks:

1. This is the cross-entropy loss function  $L_{CE}$  for multinomial logistic regression; see formulae (5.45) to (5.47) in Chapter 5.
2.  $\mathbb{1}\{\cdot\}$  is an indicator function. It is equal to 1 iff  $k$  is the correct class, and is equal to 0 for all other classes. In other words: The indicator function equals 1 only for the correct class, and the CE loss is simply the *negative log likelihood loss* of the output probability corresponding to the correct class.

# Computing the Gradient

- ▶ requires the partial derivative of the loss function with respect to each parameter.
- ▶ for a network with one layer and sigmoid output (that is: a logistic regression model), the formula in (27) can be used. For the derivation of this formula, see Section 5.10.

$$\begin{aligned}\frac{\partial L_{CE}(\hat{y}, y)}{\partial w_j} &= -(\hat{y} - y)\mathbf{x}_j \\ &= \sigma(\mathbf{w} \cdot \mathbf{x} + b) - y\mathbf{x}_j\end{aligned}\tag{27}$$

# Computing the Gradient

- ▶ for a network with one layer and softmax output (that is: a multinomial logistic regression model), the formula in (28) can be used.

$$\begin{aligned}\frac{\partial L_{CE}(\hat{\mathbf{y}}, \mathbf{y})}{\partial \mathbf{w}_{k,i}} &= -(\hat{\mathbf{y}} - \mathbf{y}) \mathbf{x}_j \\ &= -(y_k - p(y_k = 1 | \mathbf{x})) \mathbf{x}_j \\ &= - \left( \mathbf{y}_k - \frac{\exp(\mathbf{w}_k \cdot \mathbf{x} + b_k)}{\sum_{j=1}^K \exp(\mathbf{w}_j \cdot \mathbf{x} + b_j)} \right) \mathbf{x}_i\end{aligned}\tag{28}$$

# Product and Quotient Rule

Product Rule

$$\frac{d}{dx}(f(x) * g(x)) = \frac{d}{dx}(f(x)) * g(x) + f(x) * \frac{d}{dx}(g(x)) \quad (29)$$

Quotient Rule

$$\frac{d}{dx} \frac{f(x)}{g(x)} = \frac{\frac{d}{dx}(f(x)) * g(x) - f(x) * \frac{d}{dx}(g(x))}{[g(x)]^2} \quad (30)$$

# Chain Rule

$$\frac{d}{dx} [f(g(x))] = f' [g(x)] * g'(x) \quad (31)$$

Remarks:

- ▶  $f(u)$  is called *the outside function*
- ▶  $g(x)$  is called *the inside function*
- ▶ The chain rule can be stated in words as follows:  $\frac{df}{dx}$  equals the product of the derivatives of the outside function  $\frac{df}{du}$  and of the inside function  $\frac{du}{dx}$ .
- ▶ The chain rule is often helpful when taking derivatives of functions such as:  $\frac{d}{dx}(5x - 4)^6$ ,  $\frac{d}{dx}\sqrt{x^2 - 1}$ ,  $\frac{d}{dx}\frac{1}{x^2 - 4x + 5}$

# Chain Rule – Example

$$\frac{d}{dx} [f(g(x))] = f' [g(x)] * g'(x) \quad (32)$$

Find the derivative of  $f(x) = (x^2 + 1)^3$

$$\begin{aligned} f'(x) &= 3(x^2 + 1)^{3-1} * 2x^{2-1} \\ &= 3((x^2 + 1)^2(2x)) \\ &= 6x((x^2 + 1)^2) \end{aligned} \quad (33)$$

# Derivative of Sigmoid Function

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \left[ \frac{1}{1 + e^{-x}} \right] \\ &= \frac{(0)(1 + e^{-x}) - (-e^{-x})(1)}{(1 + e^{-x})^2} \\ &= \frac{e^{-x}}{(1 + e^{-x})^2} \\ &= \frac{1}{1 + e^{-x}} \frac{e^{-x}}{1 + e^{-x}}\end{aligned}\tag{34}$$

# Derivative of Sigmoid Function (continued)

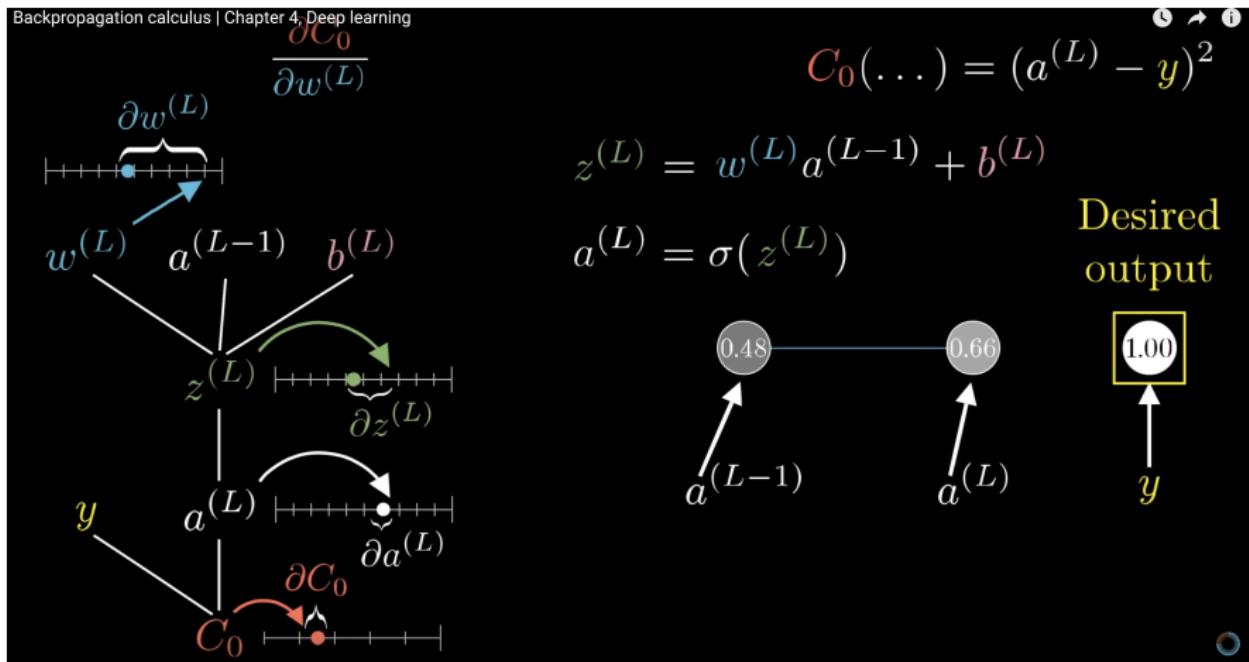
$$\begin{aligned} &= \frac{1}{1 + e^{-x}} \frac{e^{-x} + (1 - 1)}{1 + e^{-x}} \\ &= \frac{1}{1 + e^{-x}} \frac{1 + e^{-x} - 1}{1 + e^{-x}} \\ &= \frac{1}{1 + e^{-x}} \left[ \frac{1 + e^{-x}}{1 + e^{-x}} - \frac{1}{1 + e^{-x}} \right] \quad (35) \\ &= \frac{1}{1 + e^{-x}} \left[ 1 - \frac{1}{1 + e^{-x}} \right] \\ &= \sigma(x)(1 - \sigma(x)) \end{aligned}$$

# Derivatives of ReLU and tanh

$$\frac{dReLU(z)}{dz} = \begin{cases} 0 & \text{for } z < 0 \\ 1 & \text{for } z \geq 0 \end{cases} \quad (36)$$

$$\frac{dtanh(z)}{dz} = 1 - \tanh^2(z) \quad (37)$$

# Backprop in a Simple FFN



# Applying the Chain Rule

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}}$$

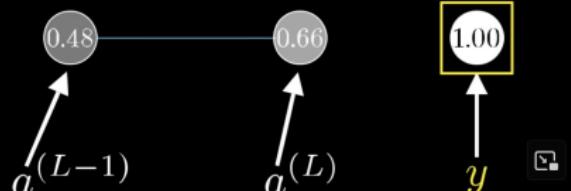
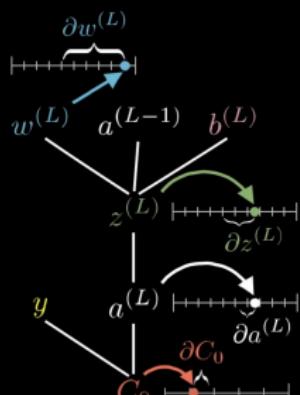
Chain rule

$$C_0(\dots) = (a^{(L)} - y)^2$$

$$z^{(L)} = w^{(L)}a^{(L-1)} + b^{(L)}$$

$$a^{(L)} = \sigma(z^{(L)})$$

Desired  
output



# Computing Relevant Derivatives

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}}$$

$$C_0 = (a^{(L)} - y)^2$$

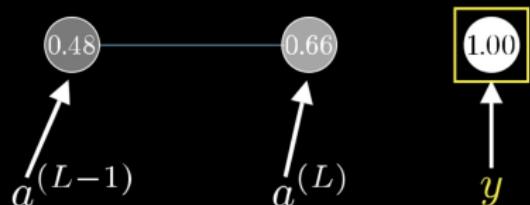
$$\frac{\partial C_0}{\partial a^{(L)}} = 2(a^{(L)} - y)$$

$$a^{(L)} = \sigma(z^{(L)})$$

$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma'(z^{(L)})$$

$$\frac{\partial z^{(L)}}{\partial w^{(L)}} = a^{(L-1)}$$

$$z^{(L)} = w^{(L)}a^{(L-1)} + b^{(L)}$$



# Average of All Training Examples

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}} = a^{(L-1)} \sigma'(z^{(L)}) 2(a^{(L)} - y)$$

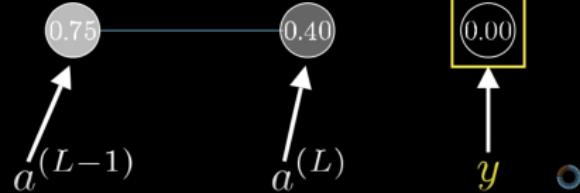
Average of all  
training examples

$$C_0 = (a^{(L)} - y)^2$$

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

$$\underbrace{\frac{\partial C}{\partial w^{(L)}}}_{\text{Derivative of full cost function}} = \overbrace{\frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial w^{(L)}}}^{\sigma(z^{(L)})}$$

Derivative of  
full cost function



# Gradient

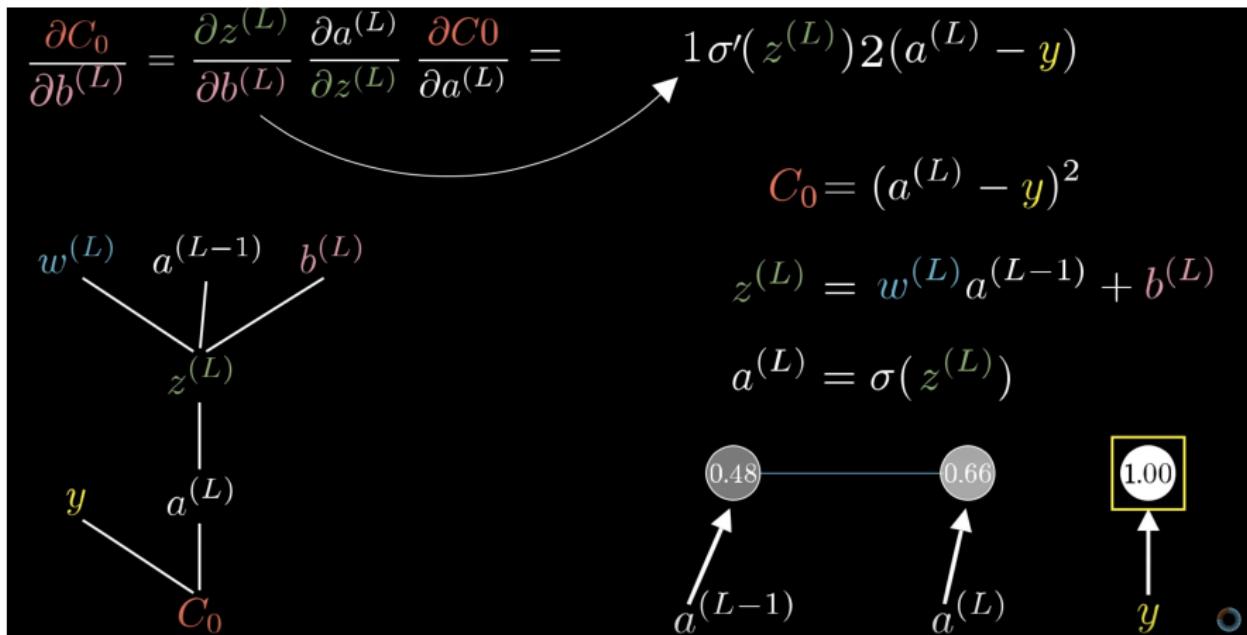
$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}} = a^{(L-1)} \sigma'(z^{(L)}) 2(a^{(L)} - y)$$

$$\nabla C = \begin{bmatrix} \frac{\partial C}{\partial w^{(1)}} \\ \frac{\partial C}{\partial b^{(1)}} \\ \vdots \\ \frac{\partial C}{\partial w^{(L)}} \\ \frac{\partial C}{\partial b^{(L)}} \end{bmatrix}$$

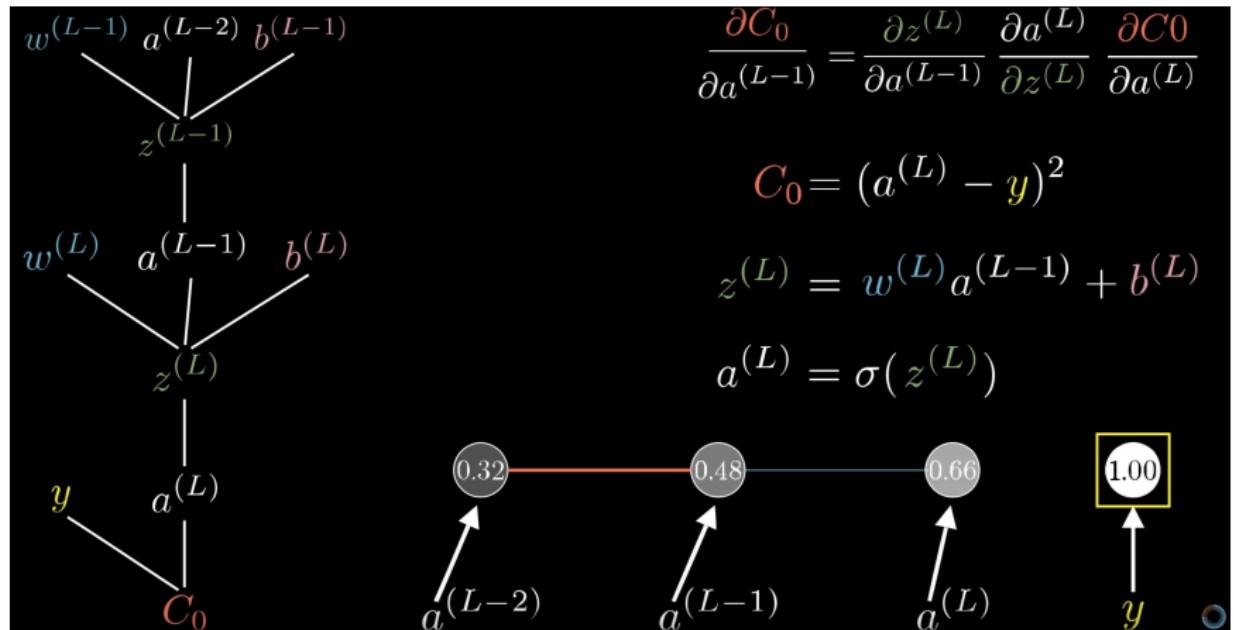
$C_0 = (a^{(L)} - y)^2$   
 $z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$   
 $a^{(L)} = \sigma(z^{(L)})$

The diagram illustrates a single layer of a neural network. It shows two neurons in the layer. The first neuron has an input value of 0.48 and an output value of 0.66. The second neuron has an input value of  $y$  and an output value of 1.00. Arrows point from the input values to the neurons, and arrows point from the neurons to their respective output values. The output value 1.00 is highlighted with a yellow border.

# Derivative of Bias



# Adding Layers

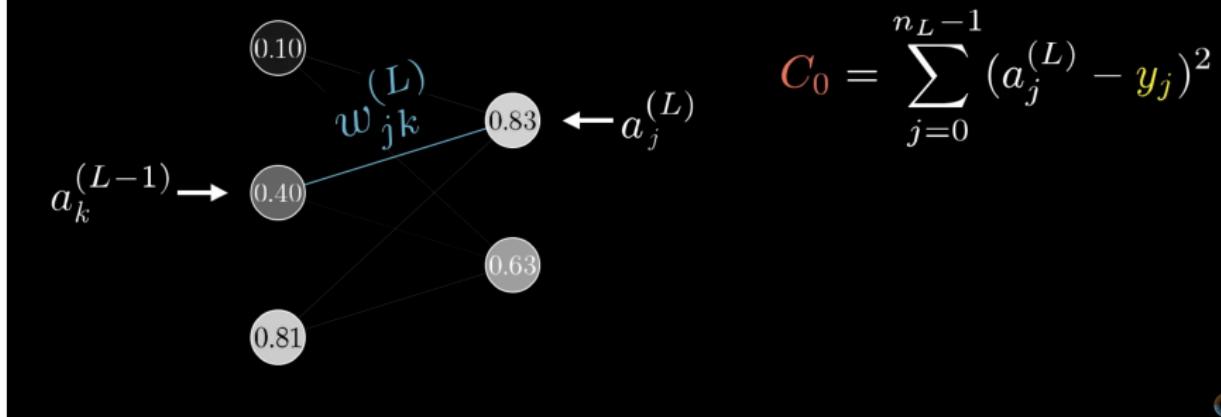


# Adding Nodes Per Layer

$$\frac{\partial C_0}{\partial w_{jk}^{(L)}} = \frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}}$$

$$z_j^{(L)} = \dots + w_{jk}^{(L)} a_k^{(L-1)} + \dots$$

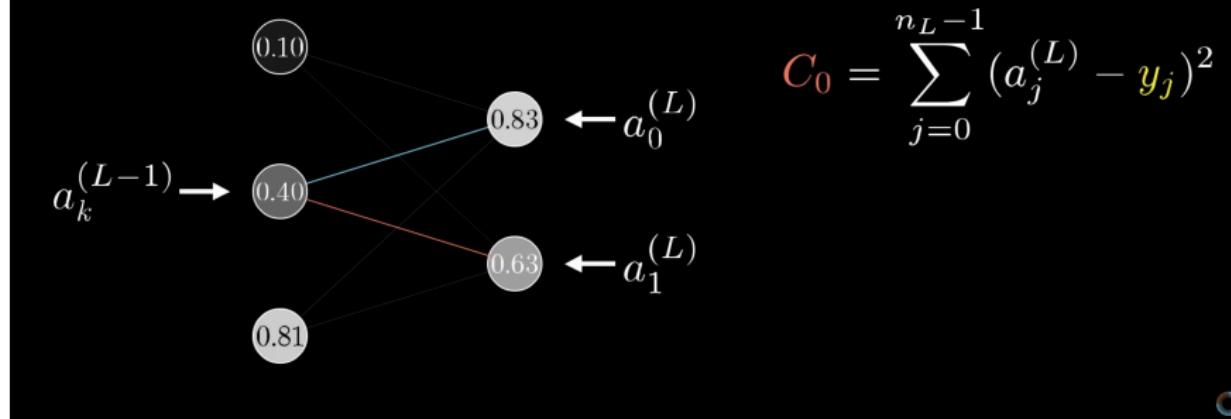
$$a_j^{(L)} = \sigma(z_j^{(L)})$$



# Summing Over Nodes Per Layer

$$\frac{\partial C_0}{\partial a_k^{(L-1)}} = \underbrace{\sum_{j=0}^{n_L-1} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}}}_{\text{Sum over layer L}} \quad z_j^{(L)} = \dots + w_{jk}^{(L)} a_k^{(L-1)} + \dots$$

$$a_j^{(L)} = \sigma(z_j^{(L)})$$

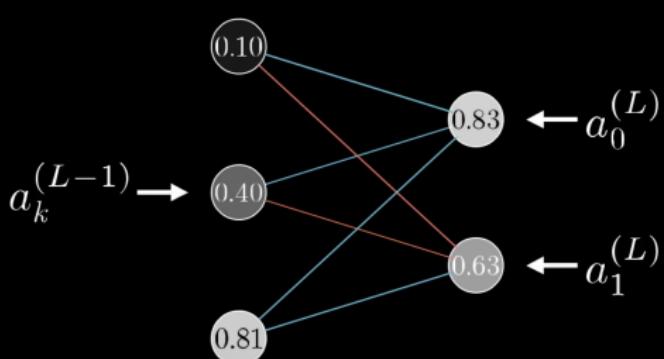


# Summing Over Layers

aylist: Neural networks

$$\frac{\partial a_k^{(L-1)}}{\partial a_k^{(L-1)}} = \underbrace{\sum_{j=0}^{n_L-1} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}}}_{\text{Sum over layer L}} \quad z_j^{(L)} = \cdots + w_{jk}^{(L)} a_k^{(L-1)} + \cdots$$

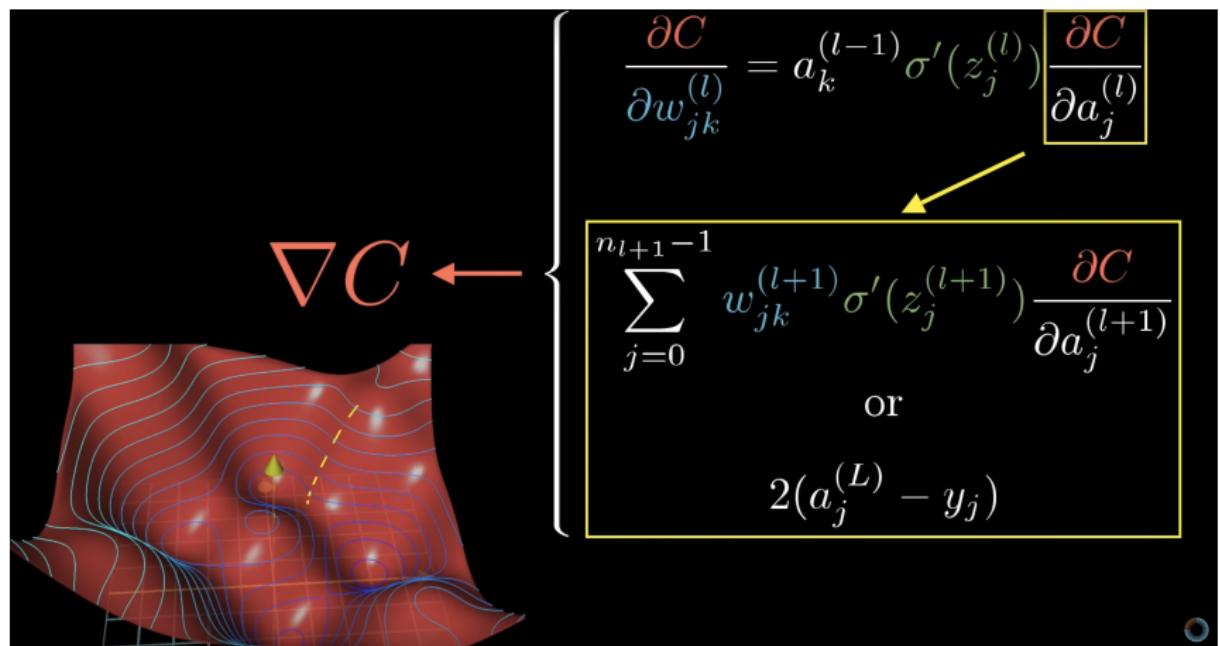
$$a_j^{(L)} = \sigma(z_j^{(L)})$$



$$C_0 = \sum_{j=0}^{n_L-1} (a_j^{(L)} - y_j)^2$$



# Summary



# What is Gradient Descent

- ▶ an algorithm for minimizing the cost function by optimizing the model parameters
- ▶ the algorithm iteratively updates the parameters in the opposite direction of the gradient of the cost function
- ▶ the size of the step at each iteration is determined by the learning rate

# Gradient Descent for Logistic Regression

$$\hat{\theta} = \operatorname{argmin}_{\theta} \frac{1}{m} \sum_{i=1}^m L_{CE}(f(x^{(i)}; \theta), y^{(i)}) \quad (38)$$

$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x; w), y) \quad (39)$$

$$\nabla_{\theta} L(f(x; \theta), y) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x; \theta), y) \\ \frac{\partial}{\partial w_2} L(f(x; \theta), y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x; \theta), y) \end{bmatrix}$$

The final equation for updating  $\theta$  based on the gradient is thus

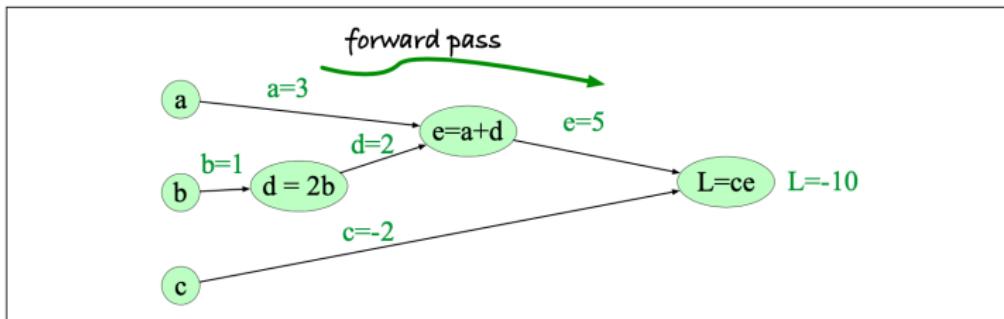
$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x; \theta), y) \quad (40)$$

where  $\eta$  is the learning rate and  $\theta = [\mathbf{w}, b]$  in logistic regression

# Types of Gradient Descent

- ▶ batch gradient descent: at each iteration, the gradient for all training instances is computed, the average of the gradients for all training examples is computed, and the average values are used for updating all parameters.
- ▶ Stochastic gradient descent (SGD): updates the parameters for a single (randomly selected) training example.
- ▶ mini-batch gradient descent: computes the gradient using a small subset, or mini-batch, of training examples

# Computation Graph – A Simple Example



# Backward differentiation on computation graphs

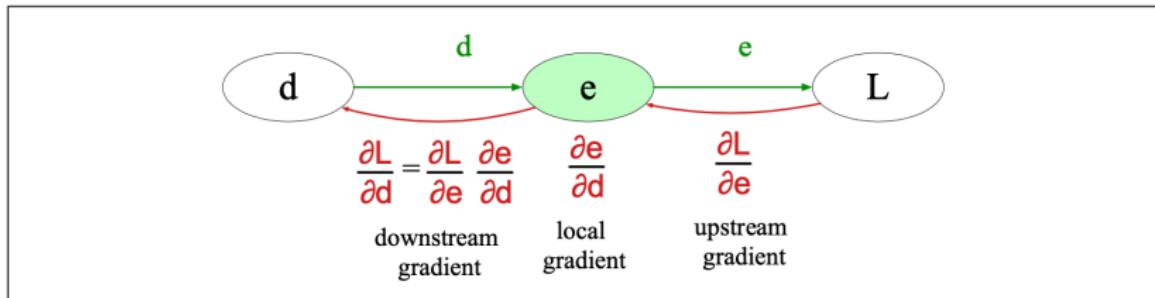
Chain rule:

$$\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx} \quad (41)$$

The chain rule extends to more than two functions. If computing the derivative of a composite function  $f(x) = u(v(w(x)))$ , the derivative of  $f(x)$  is:

$$\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dw} \cdot \frac{dw}{dx} \quad (42)$$

# Backward differentiation on sample computation graph



# Backward differentiation on sample computation graph

$$\frac{\partial L}{\partial c} = e \quad (43)$$

For the other two, we'll need to use the chain rule:

$$\begin{aligned}\frac{\partial L}{\partial a} &= \frac{\partial L}{\partial e} \frac{\partial e}{\partial a} \\ \frac{\partial L}{\partial b} &= \frac{\partial L}{\partial e} \frac{\partial e}{\partial d} \frac{\partial d}{\partial b}\end{aligned} \quad (44)$$

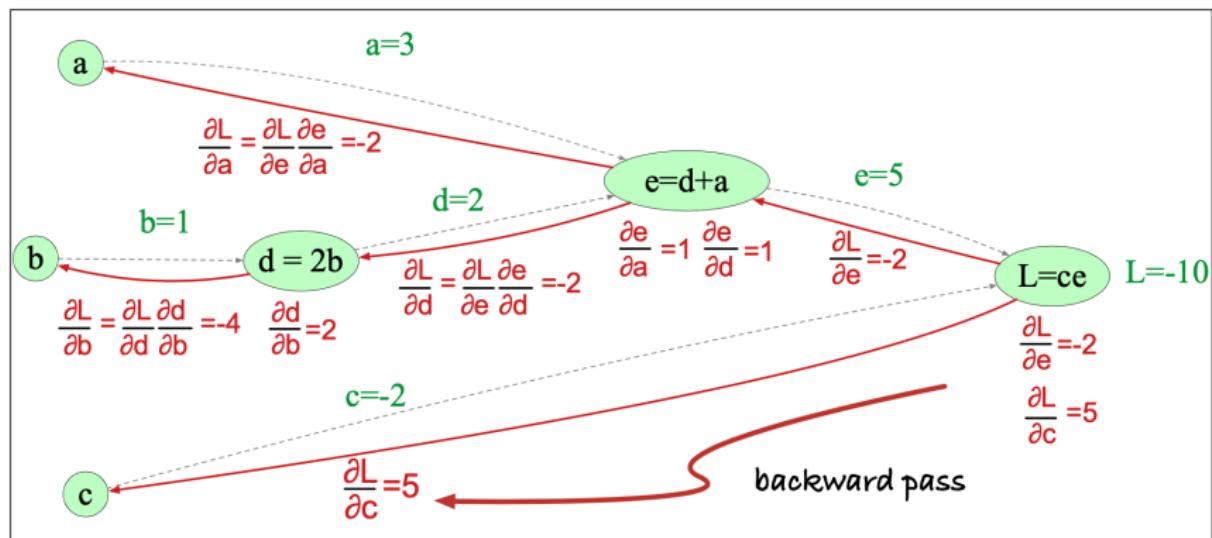
# Backward differentiation on sample computation graph

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

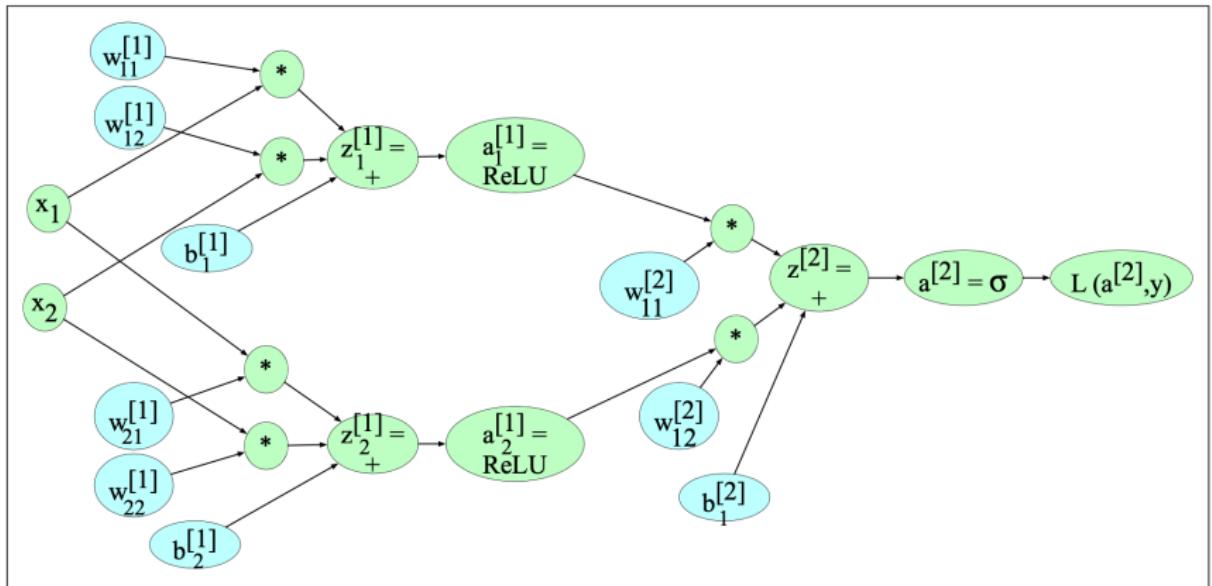
$$e = a + d : \quad \frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial d} = 1$$

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

# Backward differentiation for a neural network



# Backward differentiation for a neural network



# Backward differentiation for a neural network

The function that the computation graph is computing is:

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

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

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

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

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

# Backward differentiation for a neural network

The derivative of the sigmoid  $\sigma$  is:

$$\frac{d\sigma(z)}{dz} = \sigma(z)(1 - \sigma(z)) \quad (45)$$

The derivative of the tanh is:

$$\frac{d \tanh(z)}{dz} = 1 - \tanh^2(z) \quad (46)$$

The derivative of the ReLU is:

$$\frac{d \text{ReLU}(z)}{dz} = \begin{cases} 0 & \text{for } z < 0 \\ 1 & \text{for } z \geq 0 \end{cases} \quad (47)$$

# Derivatives of log functions

Derivative of Common Logarithm

$$\frac{d}{dx} \log_a x = \frac{1}{x * \ln(a)} \quad (48)$$

Derivative of Natural Logarithm

$$\frac{d}{dx} \ln x = \frac{1}{x} \quad (49)$$

# Derivative of the CE Loss Function with Respect to z

$$L_{CE}(a^{[2]}, y) = - \left[ y \log a^{[2]} + (1 - y) \log(1 - a^{[2]}) \right] \quad (50)$$

$$\begin{aligned} \frac{\partial L}{\partial a^{[2]}} &= - \left( \left( y \frac{\partial \log(a^{[2]})}{\partial a^{[2]}} \right) + (1 - y) \frac{\partial \log(1 - a^{[2]})}{\partial a^{[2]}} \right) \\ &= - \left( \left( y \frac{1}{a^{[2]}} \right) + (1 - y) \frac{1}{1 - a^{[2]}} (-1) \right) \\ &= - \left( \frac{y}{a^{[2]}} + \frac{y - 1}{1 - a^{[2]}} \right) \end{aligned} \quad (51)$$

# Derivative of the Sigmoid and the Chain Rule

$$\frac{\partial a^{[2]}}{\partial z} = a^{[2]}(1 - a^{[2]}) \quad (52)$$

$$\begin{aligned}\frac{\partial L}{\partial z} &= \frac{\partial L}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z} \\ &= - \left( \frac{y}{a^{[2]}} + \frac{y-1}{1-a^{[2]}} \right) a^{[2]}(1 - a^{[2]}) \\ &= a^{[2]} - y\end{aligned} \quad (53)$$

# Automatic Differentiation

- ▶ State-of-the-art implementations of backpropagation algorithms obviate the need of having to calculate gradients "by hand".
- ▶ Instead, implementations such as TensorFlow and PyTorch use computation graph formalisms that support automatic, efficient algorithms for calculating gradients on GPU hardware.

# Neural Language Models

$$P(w_t | w_1, \dots, w_{t-1}) \approx P(w_t | w_{t-N+1}, \dots, w_{t-1}) \quad (54)$$

# Embeddings

Projection layer:

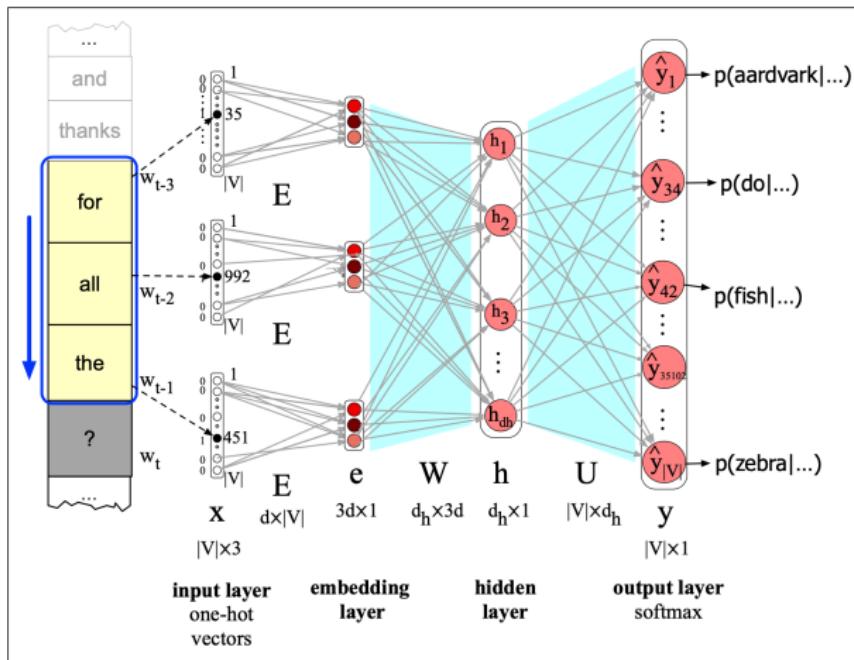
- ▶ **Select three embeddings from  $E$ :** Given the three previous words, we look up their indices, create 3 one-hot vectors, and then multiply each by the embedding matrix  $E$ . Consider  $w_{t-3}$ . The one-hot vector for 'the' (index 35) is multiplied by the embedding matrix  $E$ , to give the first part of the first hidden projection layer layer, called the **projection layer**. Since each row of the input matrix  $E$  is just an embedding for a word, and the input is a one-hot column vector  $x_i$  for word  $V_i$ , the projection layer for input  $w$  will be  $Ex_i = e_i$ , the embedding for word  $i$ . We now concatenate the three embeddings for the context words.

# Embeddings

Projection layer:

- ▶ **Multiply by W:** We now multiply by  $W$  (and add  $b$ ) and pass through the rectified linear (or other) activation function to get the hidden layer  $h$ .
- ▶ **Multiply by U:**  $h$  is now multiplied by  $U$
- ▶ **Apply softmax:** After the softmax, each node  $i$  in the output layer estimates the probability  $P(w_t = i | w_{t-1}, w_{t-2}, w_{t-3})$

# Forward inference in the neural language model



# Embeddings

In summary, if we use  $e$  to represent the projection layer, formed by concatenating the 3 embeddings for the three context vectors, the equations for a neural language model become::

$$e = (Ex_1, Ex_2, \dots, Ex) \quad (55)$$

$$h = \sigma(W_e + b) \quad (56)$$

$$z = Uh \quad (57)$$

$$\hat{y} = \text{softmax}(z) \quad (58)$$

# Training the neural language model

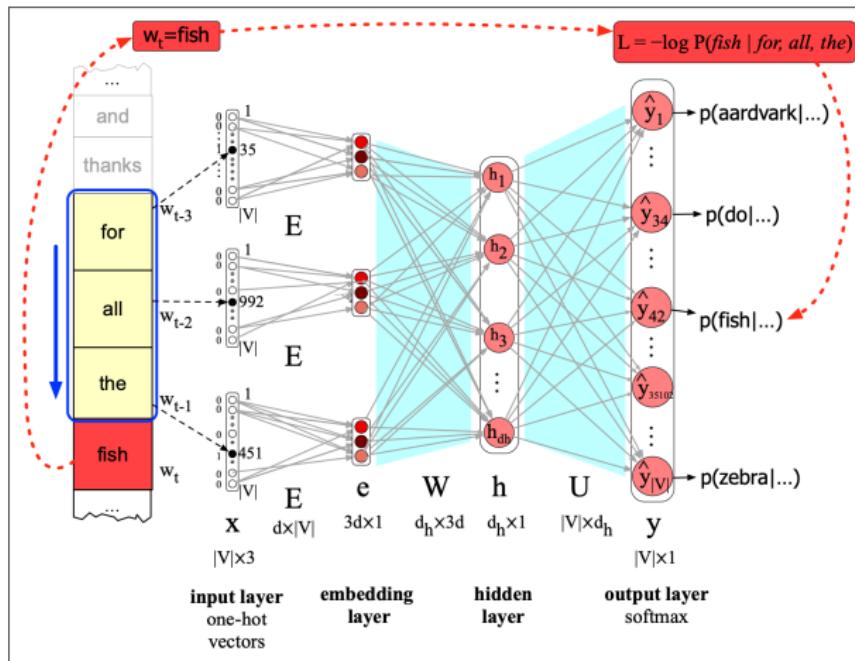
$$L_{CE}(\hat{y}, y) = -\log \hat{y}_i, \quad (\text{where } i \text{ is the correct class}) \quad (59)$$

$$L_{CE} = -\log p(w_t | w_{t-1}, \dots, w_{t-n+1}) \quad (60)$$

The parameter update for stochastic gradient descent for this loss from step  $s$  to  $s + 1$  is then:

$$\theta^{s+1} = \theta^s - \eta \frac{\partial -\log p(w_t | w_{t-1}, \dots, w_{t-n+1})}{\partial \theta} \quad (61)$$

# Backward differentiation for a neural network



# **Measuring Linguistic Regularity**

Erhard Hinrichs

Seminar für Sprachwissenschaft  
Eberhard-Karls Universität Tübingen

# Computing Analogies

**Problem:** analogy question  $a:b :: c:d$  where  $d$  is unknown

**Example:** apple : tree :: grape : ?

**Solution:**

Assume vectors  $x_a, x_b, x_c$  (all normalized to unit norm), and compute:

$$y = x_b - x_a + x_c \quad (1)$$

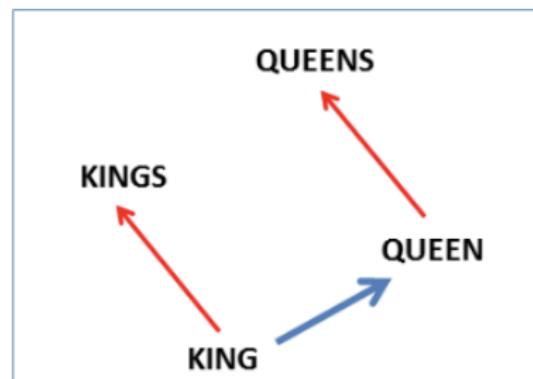
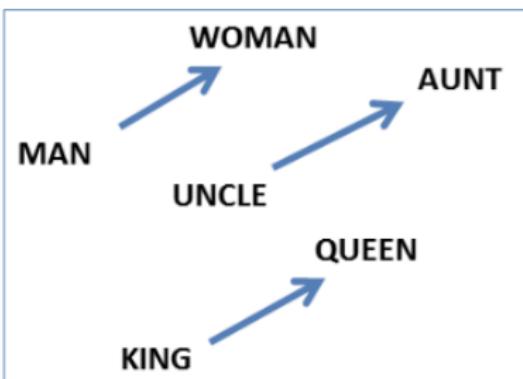
Example:

$$y = \text{tree} - \text{apple} + \text{grape} \quad (2)$$

A bit of terminology: Is this really analogy or rather relational similarity?

# Observation by Mikolov et al. (2013)

We have found that a simple vector offset method based on cosine distance is remarkably effective in solving these questions. In this method, we assume relationships are present as vector offsets, so that in the embedding space, all pairs of words sharing a particular relation are related by the same constant offset.



# Explanation

- ▶  $y$  is the continuous space representation of the word we expect to be the best answer under the parallelogram assumption of analogical learning introduced by Rumelhart and Abrahamson (1973).
- ▶ Of course, no word might exist at that exact position, so we then search for the word whose embedding vector has the greatest cosine similarity to  $y$  and output it.

# Argmin or Argmax - that is the question

Mikolov et al. (2013):

$$w^* = \operatorname{argmax}_w \frac{x_w y}{\|x_w\| \|y\|} \quad (3)$$

Jurafsky and Martin (2024):

$$w^* = \operatorname{argmin}_x \operatorname{distance}(x, b - a + a^*) \quad (4)$$

## Criticism by Joshua Peterson

- ▶ Cosine Similarity obeys the principle of symmetry.
- ▶ Human judgments of similarity do not always obey symmetry.
- ▶ Informants consider North Korea to be more similar to China, than China to North Korea.

## Bibliographical References

- Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. 2013.  
Linguistic Regularities in Continuous Space Word Representations.  
In *Proceedings of NAACL: Human Language Technologies*,  
746–751, Atlanta, Georgia. Association for Computational  
Linguistics.
- Peterson, J. C. , D. Chen, and T. L. Griffiths. 2020.  
Parallelograms revisited: Exploring the limitations of vector space  
models for simple analogies. *Cognition*, 205.
- Rumelhart, D. E. and A. A. Abrahamson. 1973. A model for  
analogical reasoning. *Cognitive Psychology*, 5(1):1–28..
- Tversky, Amos. 1977. Features of similarity. *Psychological Review*,  
84 (4), 327-352

# **Sequence Labeling for Parts of Speech and Named Entities**

Erhard Hinrichs

Seminar für Sprachwissenschaft  
Eberhard-Karls Universität Tübingen

# Three Different Tagsets for American English

- ▶ Universal Dependency (UD) Tagset: 17 distinct tags
- ▶ Penn Treebank Tagset: 45 distinct tags
- ▶ Brown Corpus Tagset: 82 distinct tags

# The Stuttgart-Tübingen Tagset (STTS) for German

- ▶ tagset with 54 distinct tags
- ▶ Tagging guidelines for STTS:  
[https://www.ims.uni-stuttgart.de/documents/  
ressourcen/lexika/tagsets/stts-1999.pdf](https://www.ims.uni-stuttgart.de/documents/ressourcen/lexika/tagsets/stts-1999.pdf)
- ▶ For an overview of the tagset see  
[https://www.ims.uni-stuttgart.de/forschung/  
ressourcen/lexika/germantagsets/#id-cfcbf0a7-0](https://www.ims.uni-stuttgart.de/forschung/ressourcen/lexika/germantagsets/#id-cfcbf0a7-0)

# UD Tagset: English Word Classes (Nivre et al. 2016a)

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, today</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>red, young, awesome</i>
	INTJ Interjection: exclamation, greeting, yes/no response, etc.	<i>oh, um, yes, hello</i>

# UD Tagset: English Word Classes: Continued

Closed Class Words	Tag	Description	Example
	<b>ADP</b>	Adposition (Pre-/Postposition): marks a noun's spacial relation	<i>in, on, by under</i>
	<b>AUX</b>	Auxiliary: helping verb marking time, place, manner	<i>can, may, should, are</i>
	<b>CCONJ</b>	Coordinating Conjunction: joins two phrases/clauses	<i>and, or, but</i>
	<b>DET</b>	Determiner: marks noun phrase properties	<i>a, an, the, this</i>
	<b>NUM</b>	Numeral	<i>one, two first, second</i>

# UD Tagset: English Word Classes: Continued

	<b>Tag</b>	<b>Description</b>	<b>Example</b>
Closed Class Words	<b>PART</b>	Particle: a preposition-like form used together with a verb	<i>up, on, off, in, at, by</i>
	<b>PRON</b>	Pronoun: a shorthand for referring to an entity or event	<i>she, who, I, others</i>
	<b>SCONJ</b>	Subordinating Conjunction: joins a main clause with a subordinate clause such as a sentential complement	<i>that, which</i>
Other	<b>PUNCT</b>	Punctuation	<i>; , ( )</i>
	<b>SYM</b>	Symbols like \$ or emoji	<i>\$, %</i>
	<b>X</b>	Other	<i>asdf, qwfg</i>

# Penn Treebank part-of-speech tags

Tag	Description	Example
CC	coord. conj.	<i>and, but, or</i>
CD	cardinal number	<i>one, two</i>
DT	determiner	<i>a, the</i>
EX	existential 'there'	<i>there</i>
FW	foreign word	<i>mea culpa</i>
IN	preposition/subordin-conj	<i>of, in, by</i>
JJ	adjective	<i>yellow</i>
JJR	comparative adj	<i>bigger</i>
JJS	superlative adj	<i>wildest</i>
LS	list item marker	<i>1, 2, One</i>
MD	modal	<i>can, should</i>
NN	sing or mass noun	<i>llama</i>

# Penn Treebank part-of-speech tags

Tag	Description	Example
NNP	proper noun, sing.	<i>IBM</i>
NNPS	proper noun, plu.	<i>Carolinas</i>
NNS	noun, plural	<i>llamas</i>
PDT	predeterminer	<i>all, both</i>
POS	possessive ending	<i>'s</i>
PRP	personal pronoun	<i>I, you, he</i>
PRP\$	possess. pronoun	<i>your, one's</i>
RB	adverb	<i>quickly</i>
RBR	comparative adv	<i>faster</i>
RBS	superlatv. adv	<i>fastest</i>
RP	particle	<i>up, off</i>
SYM	symbol	<i>+, %, &amp;</i>

# Penn Treebank part-of-speech tags

Tag	Description	Example
TO	"to"	<i>to</i>
UH	interjection	<i>ah, oops</i>
VB	verb base	<i>eat</i>
VBD	verb past tense	<i>ate</i>
VBG	verb gerund	<i>eating</i>
VBN	verb past participle	<i>eaten</i>
VBP	verb non-3sg-pr	<i>eat</i>
VBZ	verb 3sg pres	<i>eats</i>
WDT	wh-determ.	<i>which, that</i>
WP	wh-pronoun	<i>what, who</i>
WP\$	wh-possess.	<i>whose</i>
WRB	wh-adverb	<i>how, where</i>

# Tagging Guidelines for the Penn Treebank

- CC or DT:** Either/**DT** child could sing.  
Either/**CC** a boy could sing or/**CC** a girl could dance.
- CD or JJ:** a 50 3/**JJ** victory (cf. a handy/**JJ** victory)
- IN or RP** the picture we will look at/**IN** next.  
She told off/**RP** her friends.  
She told her friends off/**RP**.  
because/**IN** of/**IN** her late arrival.  
She stepped off/**IN** the train  
\* She stepped the train off/**IN**.
- IN or WDT** the fact that/**IN** you are here.  
a man that/**WDT** I know
- JJ or NP:** English/**JJ** cuisine tends to be uninspired.  
The English/**NNS** tend to be uninspired cooks.  
The West**JJ** German/**JJ** mark  
He is a West/**NP** German/**NP**.

# Tagging Guidelines for the Penn Treebank

**JJ or RB:** rapid/**JJ** growth/**NN**

rapid/**JJ** growing/**VBG** plants

**JJ or VBG:** The conversation became depressing/**JJ**.

an appetizing/**JJ** dish

\*A dish that appetizes

an existing/**VBG** safeguards

safeguards that exist.

**JJ or VBN:** He became interested/**JJ**.

He remains guided/**VBN** by these principles.

They should be kept well-watered/**JJ**.

At the time, I was married/**JJ**.

**NN or RB:** Call me when you get home/**RB**.

Call me when you are at home/**NN**.

Beatrice Santorini (1991). Part-of-Speech Guidelines for the PTB.

[www.cis.upenn.edu/~bies/manuals/tagguide.pdf](http://www.cis.upenn.edu/~bies/manuals/tagguide.pdf)

# Brown part-of-Speech tags

Tag	Description	Examples
.	sentence closer	. ; ? !
(	left paren	
)	right paren	
*	<i>not, n't</i>	
--	dash	
,	comma	
:	colon	
ABL	pre-qualifier	<i>quite, rather</i>
ABN	pre-quantifier	<i>half, all</i>
ABX	pre-quantifier	<i>both</i>
AP	post-determiner	<i>many, several, next</i>
AT	article	<i>a, the, no</i>

# Brown part-of-Speech tags

BE	<i>be</i>	
BED	<i>were</i>	
BEDZ	<i>was</i>	
BEG	<i>being</i>	
BEM	<i>am</i>	
BEN	<i>been</i>	
BER	<i>are, art</i>	
BEZ	<i>is</i>	
CC	coordinating conjunction	<i>and, or</i>
CD	cardinal numeral	<i>one, two, 2</i>
CS	subordinating conjunction	<i>if, although</i>
DO	<i>do</i>	
DOD	<i>did</i>	
DOZ	<i>does</i>	

# Brown part-of-Speech tags

DT	singular determiner	<i>this, that</i>
DTI	singular or plural determiner/quantifier	<i>some, any</i>
DTS	plural determiner	<i>these, those</i>
DTX	determiner/double conjunction	<i>either</i>
EX	existential <i>there</i>	
FW	foreign word (hyphenated before regular tag)	
HL	word occurring in headline (hyphenated after regular tag)	
HV	<i>have</i>	
HVD	<i>had</i> (past tense)	
HVG	<i>having</i>	
HVN	<i>had</i> (past participle)	
HVZ	<i>has</i>	
IN	preposition	
JJ	adjective	
JJR	comparative adjective	
JJS	semantically superlative adjective	<i>chief, top</i>
JJT	morphologically superlative adjective	<i>biggest</i>

# Brown part-of-Speech tags

MD	modal auxiliary	<i>can, should, will</i>
NC	cited word (hyphenated after regular tag)	
NN	singular or mass noun	
NN\$	possessive singular noun	
NNS	plural noun	
NNS\$	possessive plural noun	
NP	proper noun or part of name phrase	
NP\$	possessive proper noun	
NPS	plural proper noun	
NPS\$	possessive plural proper noun	
NR	adverbial noun	<i>home, today, west</i>
NRS	plural adverbial noun	
OD	ordinal numeral	<i>first, 2nd</i>

# Brown part-of-Speech tags

PN	nominal pronoun	<i>everybody, nothing</i>
PN\$	possessive nominal pronoun	
PP\$	possessive personal pronoun	<i>my, our</i>
PP\$\$	second (nominal) possessive pronoun	<i>mine, ours</i>
PPL	singular reflexive/intensive personal pronoun	<i>myself</i>
PPLS	plural reflexive/intensive personal pronoun	<i>ourselves</i>
PPO	objective personal pronoun	<i>me, him, it, them</i>
PPS	3rd. singular nominative pronoun	<i>he, she, it, one</i>
PPSS	other nominative personal pronoun	<i>I, we, they, you</i>

# Brown part-of-Speech tags

QL	qualifier	<i>very, fairly</i>
QLP	post-qualifier	<i>enough, indeed</i>
RB	adverb	
RBR	comparative adverb	
RBT	superlative adverb	
RN	nominal adverb	<i>here, then, indoors</i>
RP	adverb/particle	<i>about, off, up</i>
TL	word occurring in title (hyphenated after regular tag)	
TO	infinitive marker <i>to</i>	
UH	interjection, exclamation	

# Brown part-of-Speech tags

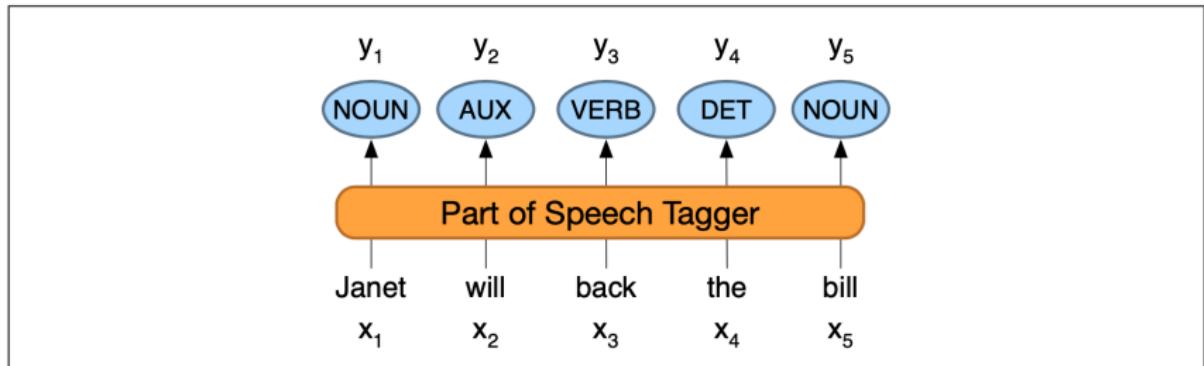
VB	verb, base form	
VBD	verb, past tense	
VBG	verb, present participle/gerund	
VBN	verb, past participle	
VBZ	verb, 3rd. singular present	
WDT	wh- determiner	<i>what, which</i>
WP\$	possessive wh- pronoun	<i>whose</i>
WPO	objective wh- pronoun	<i>whom, which, that</i>
WPS	nominative wh- pronoun	<i>who, which, that</i>
WQL	wh- qualifier	<i>how</i>
WRB	wh- adverb	<i>how, where, when</i>

# Part-of-Speech Tagging

## Example

- ▶ There/PRON/**EX** are/VERB/**VBP** 70/NUM/**CD**  
children/NOUN/**NNS** there/ADV/**RB** ./PUNC/.
  
- ▶ Preliminary/ADJ/**JJ** findings/NOUN/**NNS** were/AUX/**VBD**  
reported/VERB/**VBN** in/ADP/**IN**  
today/NOUN/**NN** 's/PART/**POS** New/PROPN/**NNP**  
England/PROPN/NNP Journal/PROPN/**NNP** of/ADP/**IN**  
Medicine/PROPN/**NNP**

# Part-of-Speech Tagging



# Tag ambiguity in the Brown and WSJ corpora

---

Types:	WSJ	Brown
<b>Unambiguous</b> (1 tag)	44,432 ( <b>86%</b> )	45,799 ( <b>85%</b> )
<b>Ambiguous</b> (2+ tags)	7,025 ( <b>14%</b> )	8,050 ( <b>15%</b> )
Tokens:		
<b>Unambiguous</b> (1 tag)	577,421 ( <b>45%</b> )	384,349 ( <b>33%</b> )
<b>Ambiguous</b> (2+ tags)	711,780 ( <b>55%</b> )	786,646 ( <b>67%</b> )

---

# Tag ambiguity in the Brown and WSJ corpora

## Example

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**].

# A list of generic named entity types

Type	Tag	Sample Categories	Example sentences
People	PER	people, characters	The <b>Turing</b> is a giant of computer science.
Organization	ORG	companies, sports teams	The <b>IPCC</b> warned about the cyclone.
Location	LOC	regions, mountains, seas	<b>Mt. Sanitas</b> is in <b>Sunshine Canyon</b> .
Geo-Political E.	GPE	countries, states	<b>Palo Alto</b> is raising the fees for parking.

# 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 appr. 60% of word tokens are ambiguous.
  - ▶ For example:  
*back* can be an adjective (**JJ**), a noun (**NN**), a finite (**VBP**) or non-finite verb (**VB**), an adverb (**RB**), or a particle (**RP**).

# Tag ambiguity in the Brown and WSJ corpora

## Example

- ▶ earnings growth took a **back/JJ** seat
- ▶ a small building in the **back/NN**
- ▶ a clear majority of senators **back/VBP** the bill
- ▶ Dave began to **back/VB** toward the door
- ▶ enable the country to buy **back/RP** debt
- ▶ I was twenty-one **back/RB** then

## Most Frequent Class Baseline

Always compare a classifier against a baseline at least as good as the most frequent class baseline (assigning each token to the class it occurred in most often in the training set).

# Examples of type ambiguities in the use of the name Washington

[*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.

# Named Entities and Named Entity Tagging

## Example

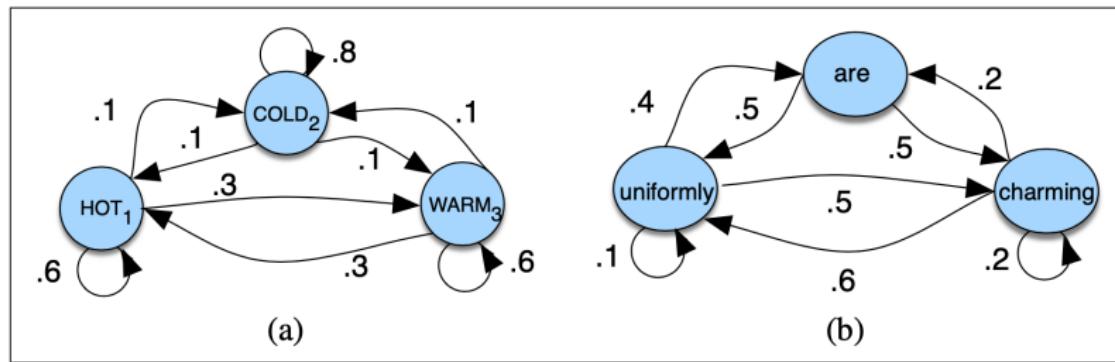
[PER **Jane Villanueva**] of [ORG **United**] , a unit of [ORG **United Airlines Holding**] , said the fare applies to the [LOC **Chicago**] route.

# NER as a sequence model

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

Table: NER as a sequence model, showing IO, BIO, and BIOES taggings

# Markov Chains



# Markov chain

Formally, a Markov chain is specified by the following components:

$$Q = q_1 q_2 \dots q_N$$

a set of  $N$  states

$$A = a_{11} a_{12} \dots a_{N1} \dots a_{NN}$$

a **transition probability matrix**  $A$ , each  $a_{ij}$  representing the probability of moving from state  $i$  to state  $j$ , s.t.

$$\sum_{j=1}^n a_{ij} = 1 \quad \forall i$$

$$\pi = \pi_1, \pi_2, \dots, \pi_N$$

an **initial probability distribution** over states.  $\pi_i$  is the probability that the Markov chain will start in state  $i$ . Some states  $j$  may have  $\pi_j = 0$ , meaning that they cannot be initial states. Also,  $\sum_{i=1}^n \pi_i = 1$

# The Hidden Markov Model

$Q = q_1 q_2 \dots q_N$	a set of $N$ states
$A = a_{11} a_{12} \dots a_{N1} \dots a_{NN}$	a <b>transition probability matrix</b> $A$ , each $a_{ij}$ representing the probability of moving from state $i$ to state $j$ , s.t. $\sum_{j=1}^n a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \dots o_T$	a sequence of $T$ <b>observations</b> , each one drawn from a vocabulary $V = v_1, v_2, \dots, v_V$
$B = b_j(o_t)$	a sequence of <b>observation likelihoods</b> , also called <b>emission probabilities</b> , each expressing the probability of an observation $o_t$ being generated from a state $q_i$
$\pi = \pi_1, \pi_2, \dots, \pi_N$	an <b>initial probability distribution</b> over states. $\pi_i$ is the probability that the Markov chain will start in state $i$ . Some states $j$ may have $\pi_j = 0$ , meaning that they cannot be initial states. Also, $\sum_{i=1}^n \pi_i = 1$

# The Hidden Markov Model

A first-order hidden Markov model instantiates two simplifying assumptions. First, as with a first-order Markov chain, the probability of a particular state depends only on the previous state:

**Markov Assumption:**  $P(q_i|q_1, \dots, q_{i-1}) = P(q_i|q_{i-1}) \quad (1)$

Second, the probability of an output observation  $o_i$  depends only on the state that produced the observation  $q_i$  and not on any other states or any other observations:

**Output Independence:**  $P(o_i|q_1, \dots, q_i, \dots, q_T, o_1, \dots, o_i, \dots, o_T) = P(o_i|q_i) \quad (2)$

# The components of an HMM tagger

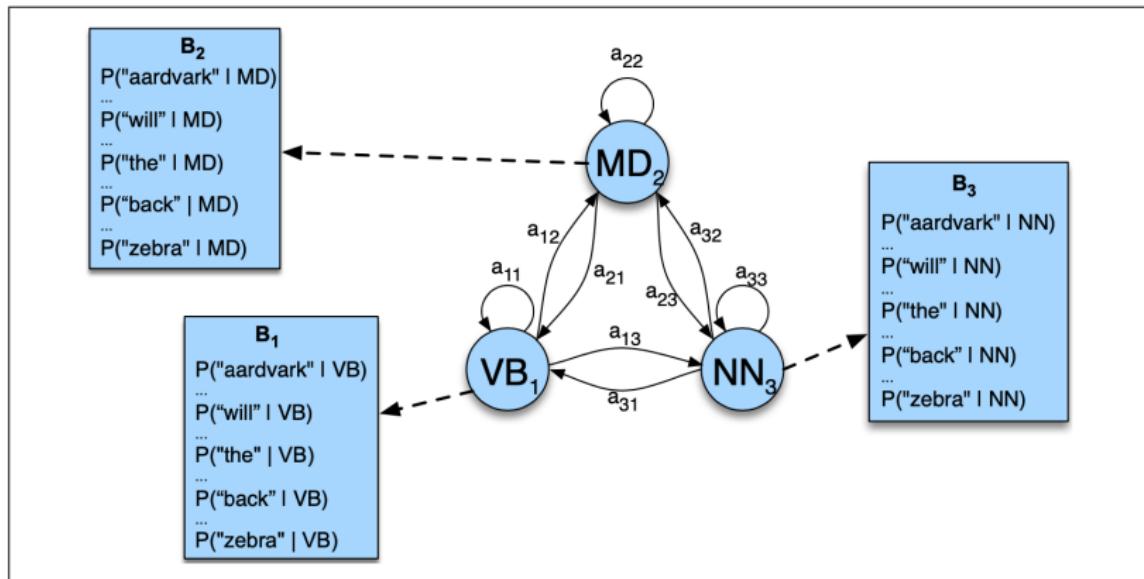
$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})} \quad (3)$$

$$P(VB|MD) = \frac{C(MD, VB)}{C(MD)} = \frac{10471}{13124} = .80 \quad (4)$$

$$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)} \quad (5)$$

$$P(will|MD) = \frac{C(MD, will)}{C(MD)} = \frac{4046}{13124} = .31 \quad (6)$$

# HMM tagging as decoding



# HMM tagging as decoding

$$\hat{t}_{1:n} = \operatorname{argmax}_{t_1 \dots t_n} P(t_1 \dots t_n | w_1 \dots w_n) \quad (7)$$

The way we'll do this in the HMM is to use Bayes' rule to instead compute:

$$\hat{t}_{1:n} = \operatorname{argmax}_{t_1 \dots t_n} \frac{P(w_1 \dots w_n | t_1 \dots t_n) P(t_1 \dots t_n)}{P(w_1 \dots w_n)} \quad (8)$$

$$\hat{t}_{1:n} = \operatorname{argmax}_{t_1 \dots t_n} P(w_1 \dots w_n | t_1 \dots t_n) P(t_1 \dots t_n) \quad (9)$$

$$P(w_1 \dots w_n | t_1 \dots t_n) \approx \prod_{i=1}^n P(w_i | t_i) \quad P(t_1 \dots t_n) \approx \prod_{i=1}^n P(t_i | t_{i-1}) \quad (10)$$

# HMM tagging as decoding

$$\hat{t}_{1:n} = \underset{t_1 \dots t_n}{\operatorname{argmax}} P(t_1 \dots t_n | w_1 \dots w_n) \approx \underset{t_1 \dots t_n}{\operatorname{argmax}} \prod_{i=1}^n \underbrace{P(w_i | t_i)}_{\text{emission}} \underbrace{P(t_i | t_{i-1})}_{\text{transition}}$$

(11)

# The Viterbi Algorithm

```

function VITERBI(observations of len  $T$ ,state-graph of len  $N$ ) returns best-path, path-prob

create a path probability matrix  $viterbi[N,T]$ 
for each state  $s$  from 1 to  $N$  do ; initialization step
     $viterbi[s,1] \leftarrow \pi_s * b_s(o_1)$ 
     $backpointer[s,1] \leftarrow 0$ 
for each time step  $t$  from 2 to  $T$  do ; recursion step
    for each state  $s$  from 1 to  $N$  do
         $viterbi[s,t] \leftarrow \max_{s'=1}^N viterbi[s',t-1] * a_{s',s} * b_s(o_t)$ 
         $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^N viterbi[s',t-1] * a_{s',s} * b_s(o_t)$ 
     $bestpathprob \leftarrow \max_{s=1}^N viterbi[s,T]$  ; termination step
     $bestpathpointer \leftarrow \operatorname{argmax}_{s=1}^N viterbi[s,T]$  ; termination step
     $bestpath \leftarrow$  the path starting at state  $bestpathpointer$ , that follows  $backpointer[]$  to states back in time
return bestpath, bestpathprob

```

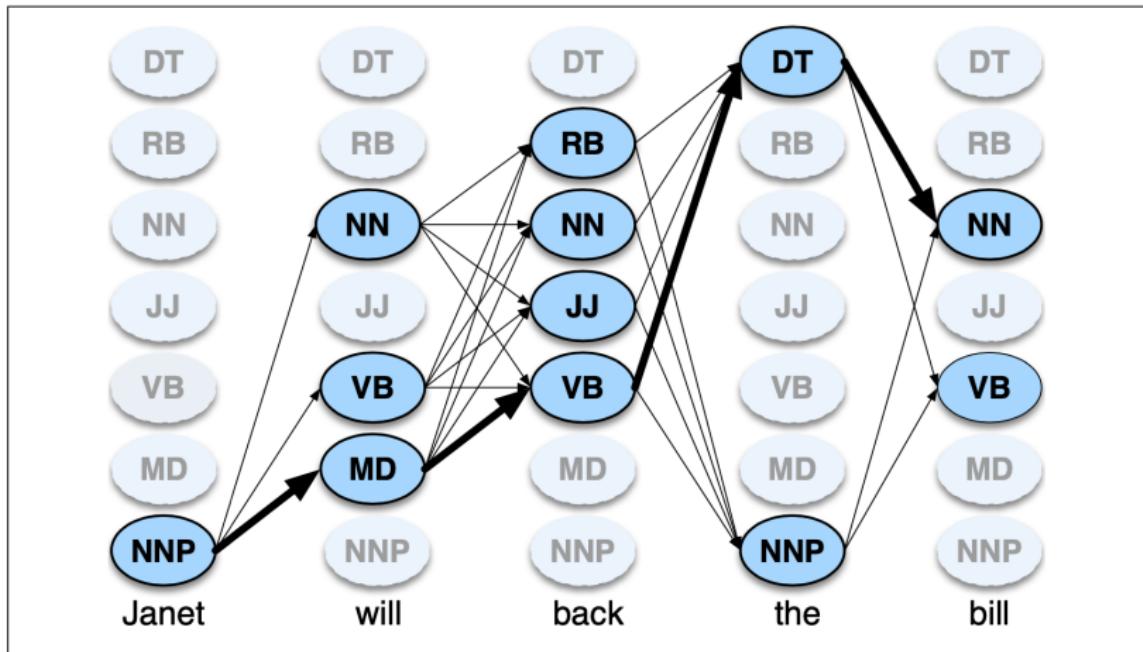
# The Viterbi Algorithm

$$v_t(j) = \max_{q_1, \dots, q_{t-1}} P(q_1 \dots q_{t-1}, o_1, o_2 \dots o_t, q_t = j | \lambda) \quad (12)$$

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t) \quad (13)$$

- $v_{t-1}(i)$  the **previous Viterbi path probability** from the previous time step
- $a_{ij}$  the **transition probability**  $y$  from previous state  $q_i$  to current state  $q_j$
- $b_j(o_t)$  the **state observation likelihood** of the observation symbol  $o_t$  given the current state  $j$

# The Viterbi Algorithm



# Working through an example

	<b>NNP</b>	<b>MD</b>	<b>VB</b>	<b>JJ</b>	<b>NN</b>	<b>RB</b>	<b>DT</b>
<b>&lt;S&gt;</b>	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
<b>NNP</b>	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
<b>MD</b>	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
<b>VB</b>	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
<b>JJ</b>	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
<b>NN</b>	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
<b>RB</b>	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
<b>DT</b>	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

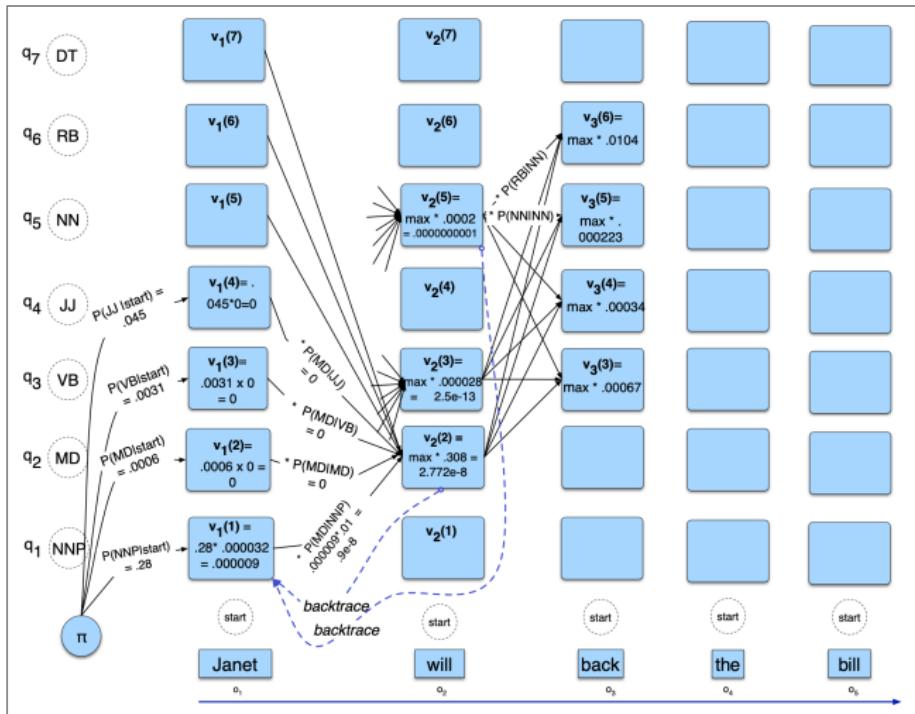
Table: The  $A$  transition probabilities  $P(t_i|t_{i-1})$  computed from the WSJ corpus without smoothing. Rows are labeled with the conditioning event; thus  $P(VB|MD)$  is 0.7968.

## Working through an example

	<b>Janet</b>	<b>will</b>	<b>back</b>	<b>the</b>	<b>bill</b>
<b>NNP</b>	0.000032	0	0	0.000048	0
<b>MD</b>	0	0.308431	0	0	0
<b>VB</b>	0	0.000028	0.000672	0	0.000028
<b>JJ</b>	0	0	0.000340	0	0
<b>NN</b>	0	0.000200	0.000223	0	0.002337
<b>RB</b>	0	0	0.010446	0	0
<b>DT</b>	0	0	0	0.506099	0

Table: Observation likelihoods B computed from the WSJ corpus without smoothing, simplified slightly

# Working through an example



# Conditional Random Fields (CRFs)

$$\begin{aligned}\hat{Y} &= \operatorname{argmax}_Y p(Y|X) \\ &= \operatorname{argmax}_Y p(X|Y)p(Y) \\ &= \operatorname{argmax}_Y \prod_i p(x_i|y_i) \prod_i p(y_i|y_{i-1})\end{aligned}\tag{14}$$

CRF to discriminate among the possible tag sequences:

$$\hat{Y} = \operatorname{argmax}_{Y \in \mathcal{Y}} P(Y|X)\tag{15}$$

# Multinomial logistic regression (Recap from J&M, ch. 5)

- ▶ multi-nominal regression, also called *softmax regression* or *maxent classifier*, is used when more than two labels are needed for classification, that is:
  - ▶ when we want to label the target variable  $\mathbf{y}$  of each data instance  $x^i$  with a unique label  $\mathbf{k}$ , taken from a set of  $K$  classes.
  - ▶ We can represent the correct target value by a **one-hot vector** that assigns the value 1 to the correct class and the value 0 to all other classes.
  - ▶ For each prediction  $\hat{y}$ , we can calculate the probability for each class  $\mathbf{k} \in K$  classes, using the **softmax** function.

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^k \exp(z_j)} \quad 1 \leq i \leq k \quad (16)$$

# Conditional Random Fields (CRFs)

Let's assume we have  $K$  features, with a weight  $w_k$  for each feature  $F_k$ :

$$p(Y|X) = \frac{\exp\left(\sum_{k=1}^K w_k F_k(X, Y)\right)}{\sum_{Y' \in \mathcal{Y}} \exp\left(\sum_{k=1}^K w_k F_k(X, Y')\right)} \quad (17)$$

# Applying Softmax in Logistic Regression

The probability of each output class  $\hat{y}_k$  can be computed as:

$$p(\mathbf{y}_k = 1 | \mathbf{x}) = \frac{\exp(\mathbf{w}_k \cdot \mathbf{x} + \mathbf{b}_k)}{\sum_{j=1}^K \exp(\mathbf{w}_j \cdot \mathbf{x} + b_j)} \quad (18)$$

The vector  $\hat{\mathbf{y}}$  of output probabilities for each of the  $K$  classes can be computed by:

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{X}\mathbf{w} + \mathbf{b}) \quad (19)$$

# Conditional Random Fields (CRFs)

It's common to also describe the same equation by pulling out the denominator into a function  $Z(X)$ :

$$p(Y|X) = \frac{1}{Z(X)} \exp \left( \sum_{k=1}^K w_k F_k(X, Y) \right) \quad (20)$$

$$Z(X) = \sum_{Y' \in \mathcal{Y}} \exp \left( \sum_{k=1}^K w_k F_k(X, Y') \right) \quad (21)$$

$$F_k(X, Y) = \sum_{i=1}^n f_k(y_{i-1}, y_i, X, i) \quad (22)$$

# Features in a CRF POS Tagger

$$\begin{aligned} & \mathbb{1}\{x_i = \text{the}, y_i = \text{DET}\} \\ & \mathbb{1}\{y_i = \text{PROPN}, x_{i+1} = \text{Street}, y_{i-1} = \text{NUM}\} \quad (23) \\ & \mathbb{1}\{y_i = \text{VERB}, y_{i-1} = \text{AUX}\} \end{aligned}$$

$$\begin{aligned} & f_{3743} : y_i = \text{VB} \text{ and } x_i = \text{back} \\ & f_{156} : y_i = \text{VB} \text{ and } y_{i-1} = \text{MD} \quad (24) \\ & f_{99732} : y_i = \text{VB} \text{ and } x_{i-1} = \text{will} \text{ and } x_{i+2} = \text{bill} \end{aligned}$$

# Features in a CRF POS Tagger

- $x_i$  contains a particular prefix (perhaps from all prefixes of length  $\leq 2$ )
- $x_i$  contains a particular suffix (perhaps from all suffixes of length  $\leq 2$ )
- $x_i$ 's word shape
- $x_i$ 's short word shape

For example the word *well-dressed* might generate the following non-zero valued feature values:

$$\text{prefix}(x_i) = w$$

$$\text{prefix}(x_i) = we$$

$$\text{suffix}(x_i) = ed$$

$$\text{suffix}(x_i) = d$$

$$\text{word-shape}(x_i) = xxxx-xxxxxxxx$$

$$\text{short-word-shape}(x_i) = x-x$$

# Features for CRF Named Entity Recognizers

Typical features for a feature-based NER system:

- identity of  $w_i$ , identity of neighboring words
- embeddings for  $w_i$ , embeddings for neighboring words
- part of speech of  $w_i$ , part of speech of neighboring words
- presence of  $w_i$  in a **gazetteer**
- $w_i$  contains a particular prefix (from all prefixes of length  $\leq 4$ )
- $w_i$  contains a particular suffix (from all suffixes of length  $\leq 4$ )
- word shape of  $w_i$ , word shape of neighboring words
- short word shape of  $w_i$ , short word shape of neighboring words
- gazetteer features

# Features in a CRF POS Tagger

$\text{prefix}(x_i) = L$

$\text{prefix}(x_i) = L'$

$\text{prefix}(x_i) = L'O$

$\text{prefix}(x_i) = L'Oc$

$\text{word-shape}(x_i) = X'Xxxxxxxxx$

$\text{suffix}(x_i) = tane$

$\text{suffix}(x_i) = ane$

$\text{suffix}(x_i) = ne$

$\text{suffix}(x_i) = e$

$\text{short-word-shape}(x_i) = X'Xx$

# Features in a CRF POS Tagger

Words	POS	Short shape	Gazetteer	BIO Label
Jane	NNP	Xx	0	B-PER
Villanueva	NNP	Xx	1	I-PER
of	IN	x	0	O
United	NNP	Xx	0	B-ORG
Airlines	NNP	Xx	0	I-ORG
Holding	NNP	Xx	0	I-ORG
discussed	VBD	x	0	O
the	DT	x	0	O
Chicago	NNP	Xx	1	B-LOC
route	NN	x	0	O
.	.	.	0	O

Table: Some NER features for a sample sentence, assuming that Chicago and Villanueva are listed as locations in a gazetteer. We assume features only take on the values 0 or 1, so the first POS feature, for example, would be represented as  $\mathbb{1}\{\text{POS} = \text{NNP}\}$ .

# Inference and Training for CRFs

$$\begin{aligned}\hat{Y} &= \operatorname{argmax}_{Y \in \mathcal{Y}} P(Y|X) \\ &= \operatorname{argmax}_{Y \in \mathcal{Y}} \frac{1}{Z(X)} \exp \left( \sum_{k=1}^K w_k F_k(X, Y) \right) \\ &= \operatorname{argmax}_{Y \in \mathcal{Y}} \exp \left( \sum_{k=1}^K w_k \sum_{i=1}^n f_k(y_{i-1}, y_i, X, i) \right) \\ &= \operatorname{argmax}_{Y \in \mathcal{Y}} \sum_{k=1}^K w_k \sum_{i=1}^n f_k(y_{i-1}, y_i, X, i) \\ &= \operatorname{argmax}_{Y \in \mathcal{Y}} \sum_{i=1}^n \sum_{k=1}^K w_k f_k(y_{i-1}, y_i, X, i)\end{aligned}$$

# Inference and Training for CRFs

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t); \quad 1 \leq j \leq N, 1 < t \leq T \quad (25)$$

which is the HMM implementation of

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) P(s_j | s_i) P(o_t | s_j) \quad 1 \leq j \leq N, 1 < t \leq T \quad (26)$$

The CRF requires only a slight change to this latter formula, replacing the  $a$  and  $b$  prior and likelihood probabilities with the CRF features:

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) \sum_{k=1}^K w_k f_k(y_{i-1}, y_i, X, i) \quad 1 \leq j \leq N, 1 < t \leq T \quad (27)$$

# **Deep Learning Architectures for Sequence Processing**

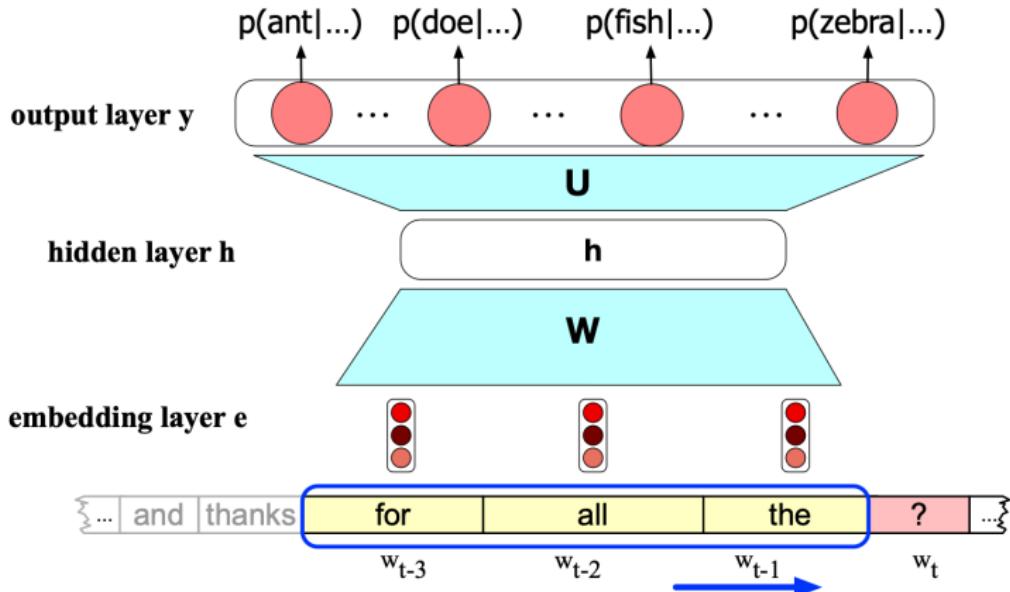
Erhard Hinrichs

Seminar für Sprachwissenschaft  
Eberhard-Karls Universität Tübingen

# Three Alternatives for Dealing with Sequences of Language Data (1)

- ▶ Presentation of an input sequence to a feed-forward network (FFN) as a sliding window of subsequences of fixed length
- ▶ Advancing the sliding window by one token at a time and presenting each subsequence as separate inputs
- ▶ The tokens in each input subsequence are projected to their corresponding embeddings, and the embeddings are concatenated in the projection layer.
- ▶ Disadvantages of this approach
  - ▶ Sliding windows provides only limited context.
  - ▶ No context information is preserved beyond the fixed context window.

# Language Modeling with a FFN



## Language Models Revisited (see J&N, ch. 3)

- ▶  $P(fish|Thanks\ for\ all\ the)$
- ▶ More generally:  $P(w_{1:n}) = \prod_{i=1}^n P(w_i|w_{<i})$  (Chain rule)

# Perplexity

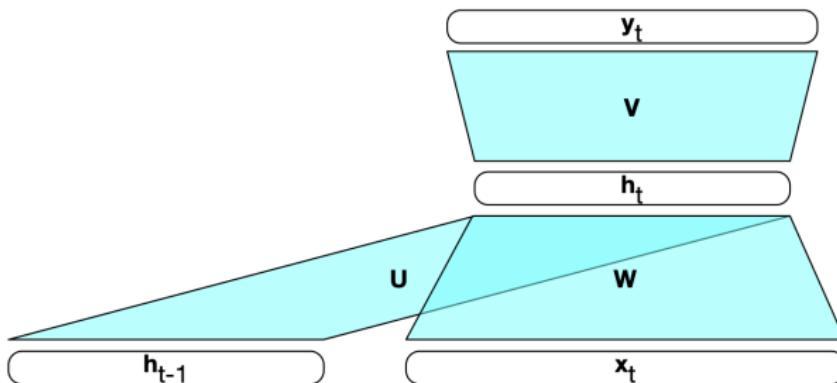
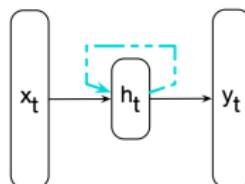
$$\begin{aligned} PP_{\theta}(w_{1:n}) &= P_{\theta}(w_{1:n})^{-\frac{1}{n}} \\ &= \sqrt[n]{\frac{1}{P_{\theta}(w_{1:n})}} \end{aligned} \tag{1}$$

$$PP_{\theta}(w_{1:n}) = \sqrt[n]{\prod_{i=1}^n \frac{1}{P_{\theta}(w_i | w_{1:i-1})}} \tag{2}$$

## Three Alternatives for Dealing with Sequences of Language Data (2)

- ▶ Presenting individual input tokens to a recurrent neural network (RNN)
  - ▶ Enriching the hidden state of each input with output representations of previous tokens in the same sequence.
- ▶ Disadvantages of this approach
  - ▶ Difficulty of modelling long-distance dependencies among tokens
  - ▶ Exploding or vanishing gradients.

# Recurrent Neural Networks



# Inference in RNNs

$$\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t) \quad (3)$$

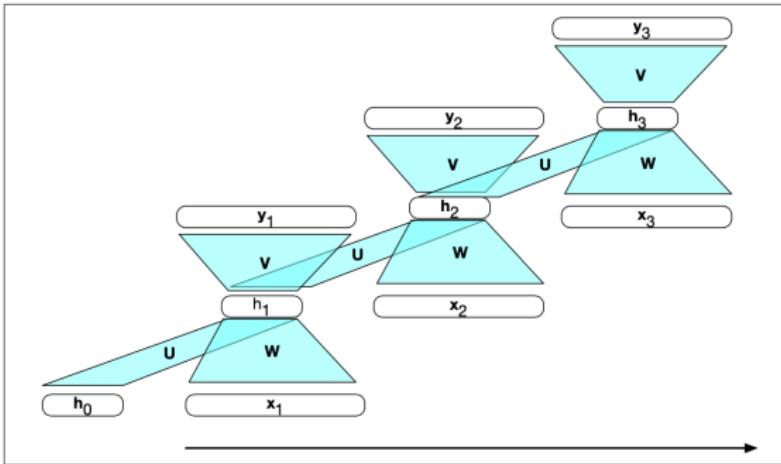
$$\mathbf{y}_t = f(\mathbf{V}\mathbf{h}_t) \quad (4)$$

$$\mathbf{y}_t = \text{softmax}(\mathbf{V}\mathbf{h}_t) \quad (5)$$

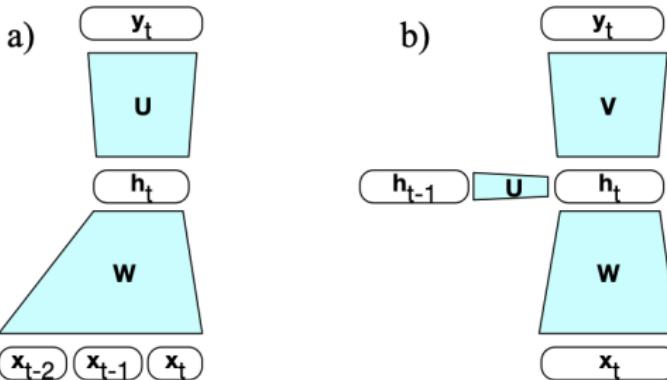
# Forward Inference in RNNs

```
function FORWARDRNN( $x, network$ ) returns output sequence  $y$ 
```

```
     $h^0 \leftarrow 0$ 
    for  $i \leftarrow 1$  to LENGTH( $x$ ) do
         $h_i \leftarrow g(Uh_{i-1} + Wx_i)$ 
         $y_i \leftarrow f(Vh_i)$ 
    return  $y$ 
```



# Comparing FFN and RNN Language Models



**Figure 9.5** Simplified sketch of (a) a feedforward neural language model versus (b) an RNN language model moving through a text.

# RNNs as Language Models

$$\mathbf{e}_t = \mathbf{E}\mathbf{x}_t \quad (6)$$

$$\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{e}_t) \quad (7)$$

$$\mathbf{y}_t = \text{softmax}(\mathbf{V}\mathbf{h}_t) \quad (8)$$

# RNNs as Language Models

$$P(w_{t+1} = i | w_1, \dots, w_t) = \mathbf{y}_t[i] \quad (9)$$

$$P(w_{1:n}) = \prod_{i=1}^n P(w_i | w_{1:i-1}) \quad (10)$$

$$= \prod_{i=1}^n \mathbf{y}_i[w_i] \quad (11)$$

# RNNs as Language Models

$$L_{CE} = - \sum_{w \in V} \mathbf{y}_t[w] \log \hat{\mathbf{y}}_t[w] \quad (12)$$

$$L_{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = -\log \hat{\mathbf{y}}_t[w_{t+1}] \quad (13)$$

## Training a RNN Language Model

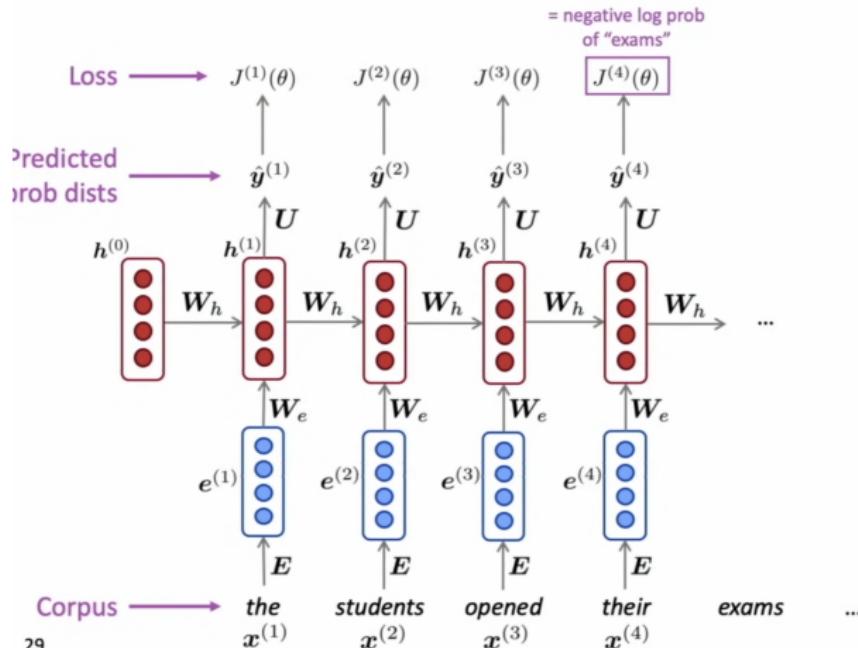
- Get a **big corpus of text** which is a sequence of words  $x^{(1)}, \dots, x^{(T)}$
- Feed into RNN-LM; compute output distribution  $\hat{y}^{(t)}$  for **every step  $t$** .
  - i.e. predict probability dist of **every word**, given words so far
- Loss function on step  $t$  is **cross-entropy** between predicted probability distribution  $\hat{y}^{(t)}$ , and the true next word  $y^{(t)}$  (one-hot for  $x^{(t+1)}$ ):

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

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

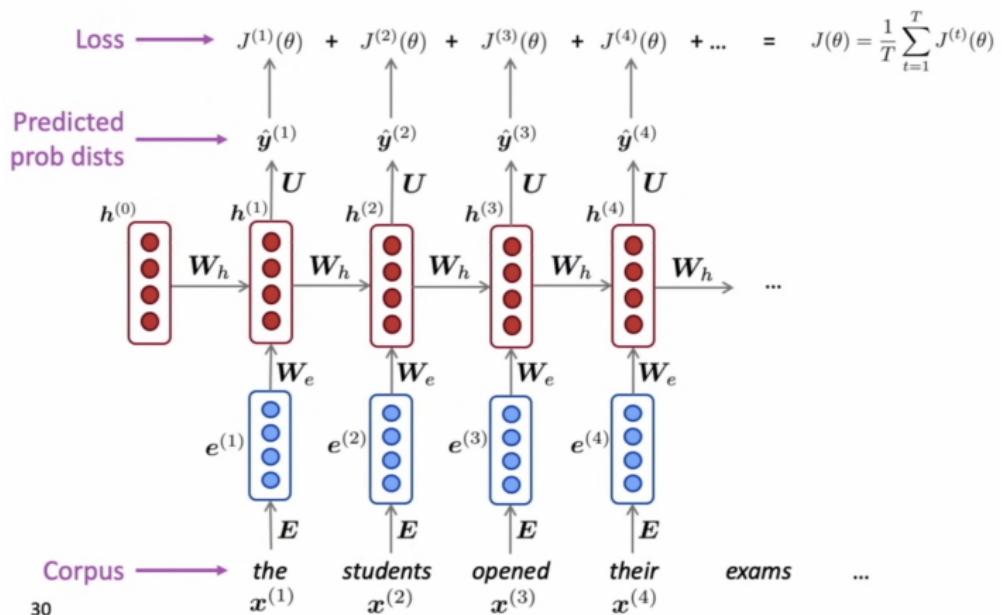
$$J(\theta) = \frac{1}{T} \sum_{t=1}^T J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^T - \log \hat{y}_{x_{t+1}}^{(t)}$$

# Training a RNN Language Model



29

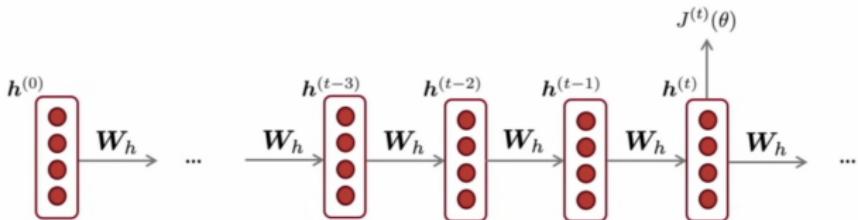
Copyright: Stanford CS224N: NLP with Deep Learning, Lecture 6; Winter 2019;  
<https://www.youtube.com/watch?v=iWea12EAu6U&list=PLoROMvodv4rOhcuXMZkNm7j3fVwBBY42z&index=8>



30

Copyright: Stanford CS224N: NLP with Deep Learning, Lecture 6; Winter 2019;  
<https://www.youtube.com/watch?v=iWeai2EAu6U&list=PLoROMvodv4r0hcuXMZkNm7j3fVwBBY42z&index=8>

## Backpropagation for RNNs



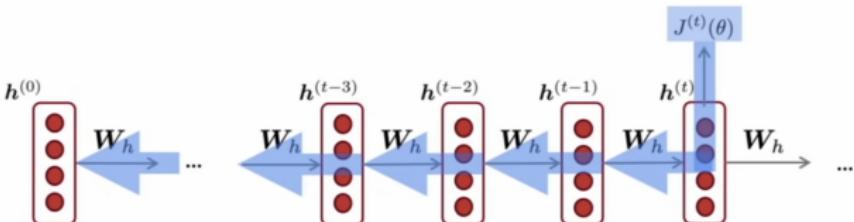
Question: What's the derivative of  $J^{(t)}(\theta)$  w.r.t. the **repeated** weight matrix  $W_h$  ?

Answer: 
$$\frac{\partial J^{(t)}}{\partial \mathbf{W}_h} = \sum_{i=1}^t \frac{\partial J^{(t)}}{\partial \mathbf{W}_h} \Big|_{(i)}$$

“The gradient w.r.t. a repeated weight  
is the sum of the gradient  
w.r.t. each time it appears”

Copyright: Stanford CS224N: NLP with Deep Learning, Lecture 6; Winter 2019;  
<https://www.youtube.com/watch?v=iWeai2EAu6U&list=PLoROMvodv4r0hcuXMZkNm7j3fVwBBY42z&index=8>

## Backpropagation for RNNs



Answer: Backpropagate over timesteps  $i=t, \dots, 0$ , summing gradients as you go.  
This algorithm is called “backpropagation through time”

$$\frac{\partial J^{(t)}}{\partial \mathbf{W}_h} = \sum_{i=1}^t \left. \frac{\partial J^{(t)}}{\partial \mathbf{W}_h} \right|_{(i)}$$

Question: How do we calculate this?

35

# Backpropagation Through Time (BPT)

$$s_t = \tanh(Ux_t + Ws_{t-1}) \quad (14)$$

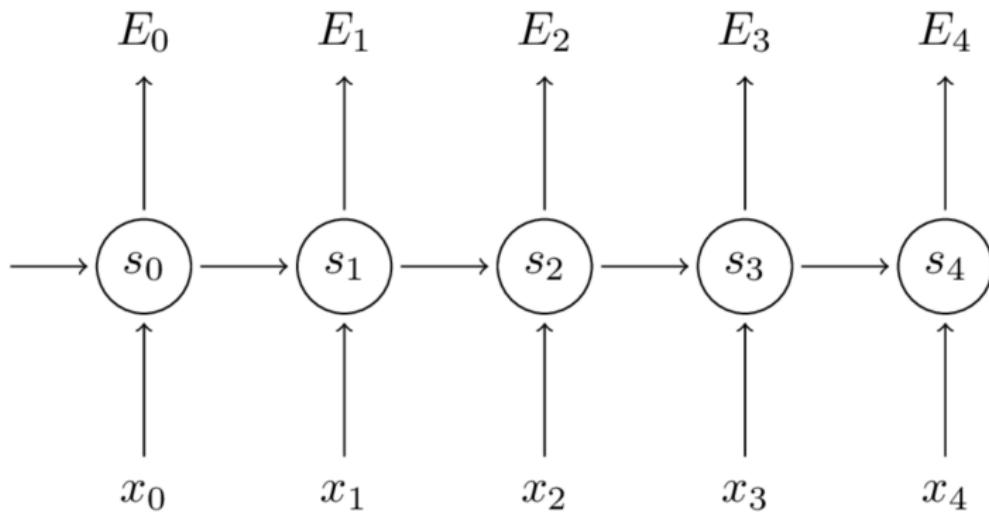
$$\hat{y}_t = \text{softmax}(Vs_t) \quad (15)$$

$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t \quad (16)$$

$$\begin{aligned} E_t(y, \hat{y}) &= \sum_t E_t(y_t, \hat{y}_t) \\ &= - \sum_t y_t \log \hat{y}_t \end{aligned} \quad (17)$$

Copyright: <https://dennybritz.com/posts/wildml/recurrent-neural-networks-tutorial-part-3/>

# Example



Copyright: <https://dennybritz.com/posts/wildml/recurrent-neural-networks-tutorial-part-3/>

## Calculate the gradients of the error with respect to our parameters U,V and W

$$\begin{aligned}\frac{\partial E_3}{\partial V} &= \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial V} \\ &= \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial z_3} \frac{\partial z_3}{\partial V} \\ &= (\hat{y}_3 - y_3) \otimes s_3\end{aligned}\tag{18}$$

Copyright: <https://dennybritz.com/posts/wildml/recurrent-neural-networks-tutorial-part-3/>

## Take note

$\frac{\partial E_3}{\partial V}$  depends only on the values at the current time step:  $\hat{y}_3, y_3, s_3$

But:  $\frac{\partial E_3}{\partial W}$  and  $\frac{\partial E_3}{\partial U}$  depend also on previous time steps:

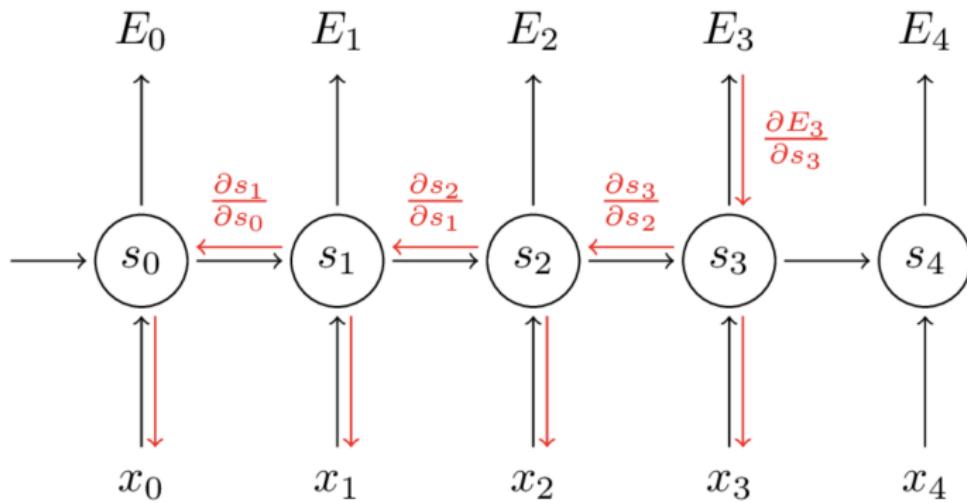
$$s_3 = \tanh(Ux_t + Bs_2) \quad (19)$$

depends on  $s_2$ , which depends on  $W$  and  $s_1$ , and so on.

Therefore:

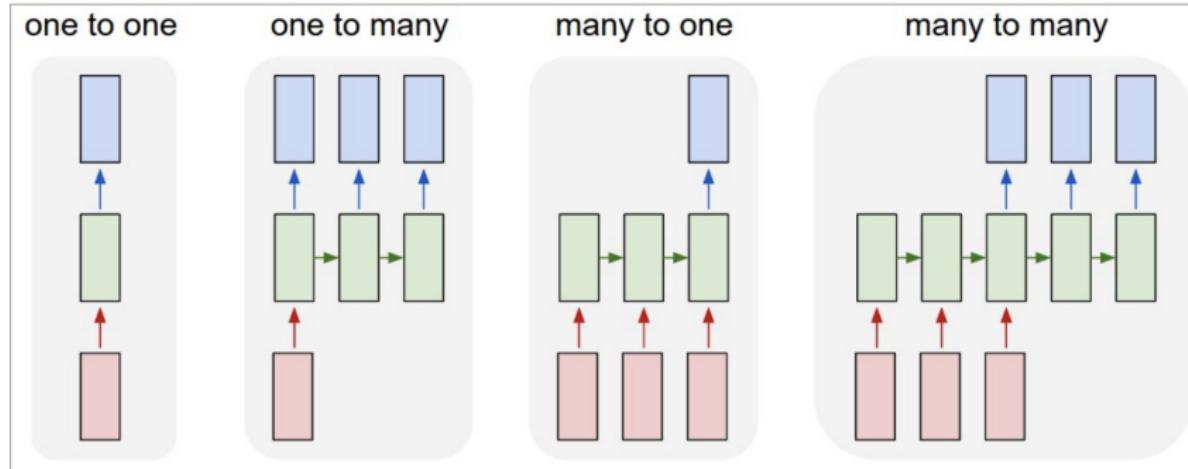
$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^3 \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W} \quad (20)$$

# RNN Unrolled with Gradients



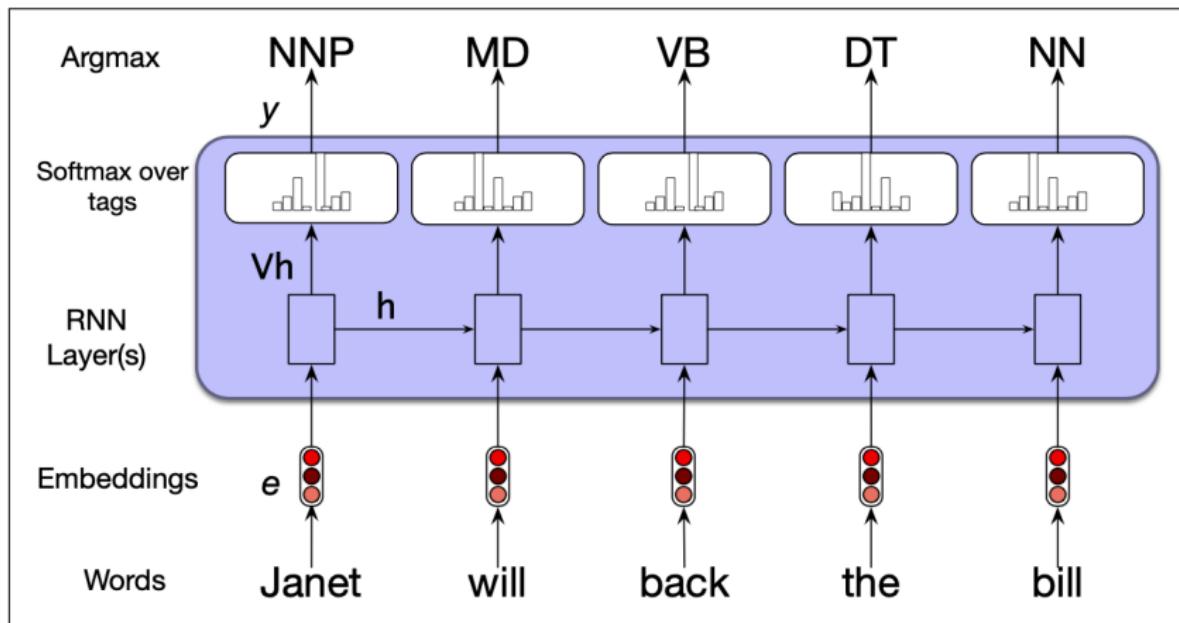
Copyright: <https://dennybritz.com/posts/wildml/recurrent-neural-networks-tutorial-part-3/>

# RNN Architectures Overview

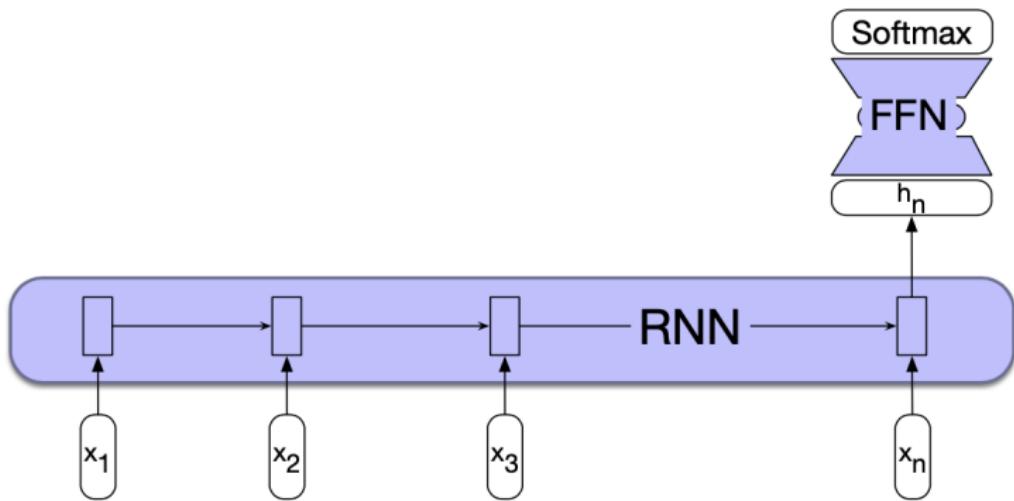


Copyright: Andrej Karpathy blog; <https://karpathy.github.io/2015/05/21/rnn-effectiveness/>

# Sequence Labeling



# RNNs for Sequence Classification



# RNNs for Sequence Classification

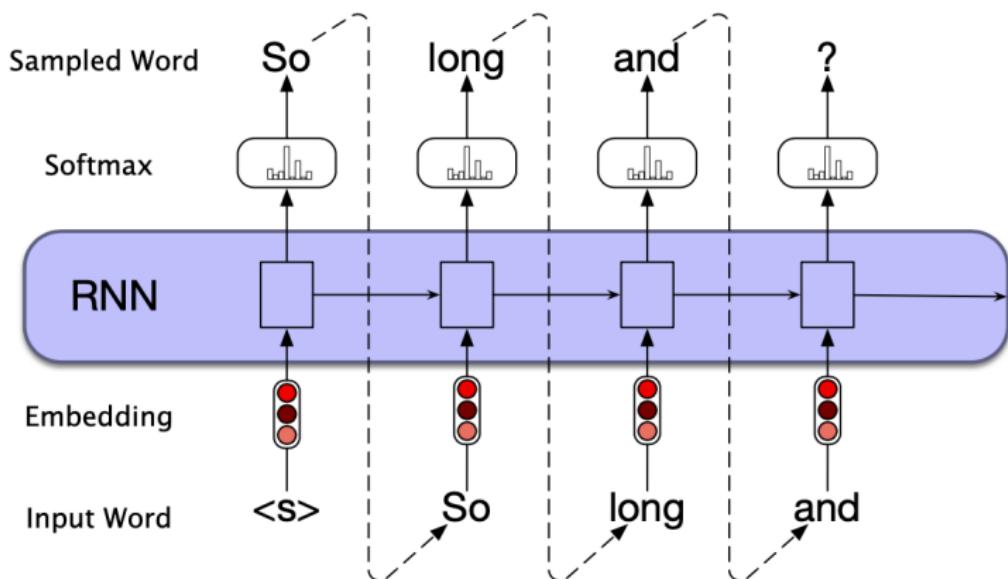
Pooling

$$\mathbf{h}_{mean} = \frac{1}{n} \sum_{i=1}^n \mathbf{h}_i \quad (21)$$

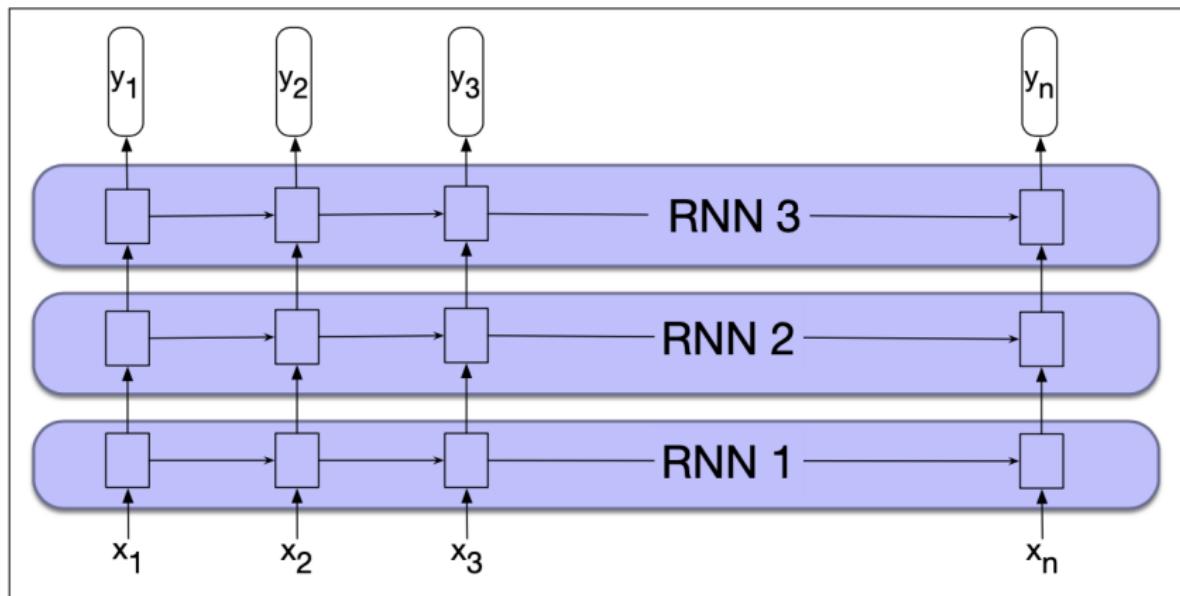
# Autoregressive Generation with RNN-Based Language Models

- ▶ Sample a word in the output from the softmax distribution that results from using the beginning of sentence marker,  $\langle s \rangle$ , as the first input.
- ▶ Use the word embedding for that first word as the input to the network at the next time step, and then sample the next word in the same fashion.
- ▶ Continue generating until the end of sentence marker,  $\langle s \rangle$ , is sampled or a fixed length limit is reached.

# Autoregressive Generation with RNN-Based Language Models



# Stacked RNNs



# Bidirectional RNNs

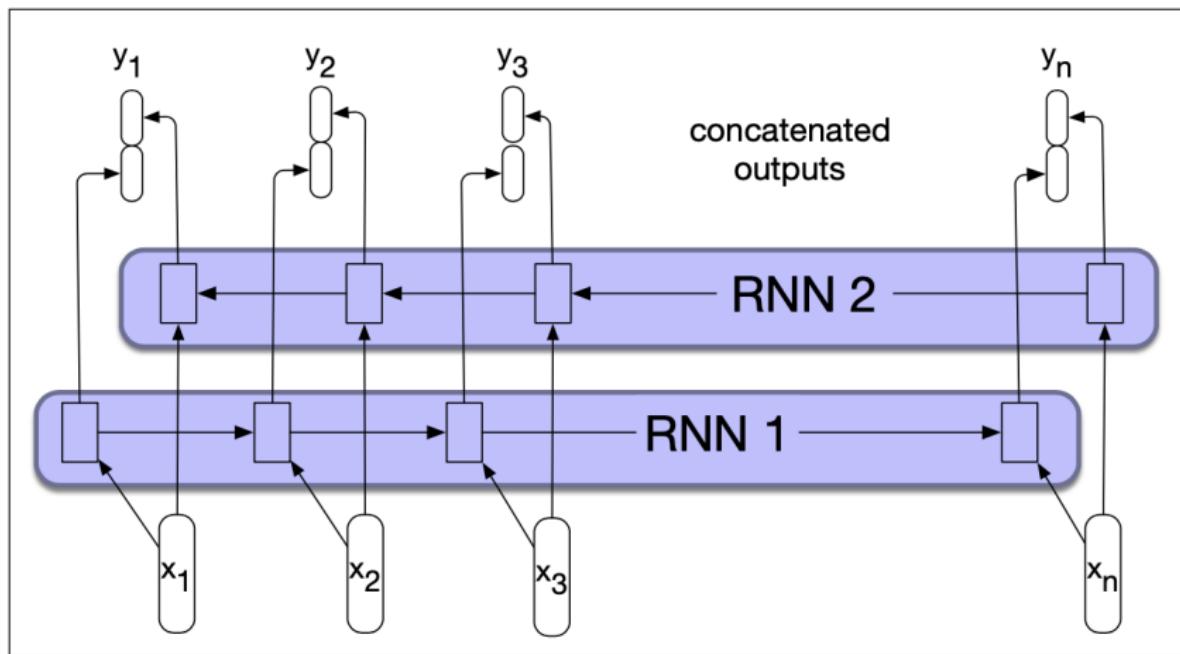
$$\mathbf{h}_t^f = RNN_{forward}(\mathbf{x}_1, \dots, \mathbf{x}_t) \quad (22)$$

$$\mathbf{h}_t^b = RNN_{backward}(\mathbf{x}_t, \dots, \mathbf{x}_n) \quad (23)$$

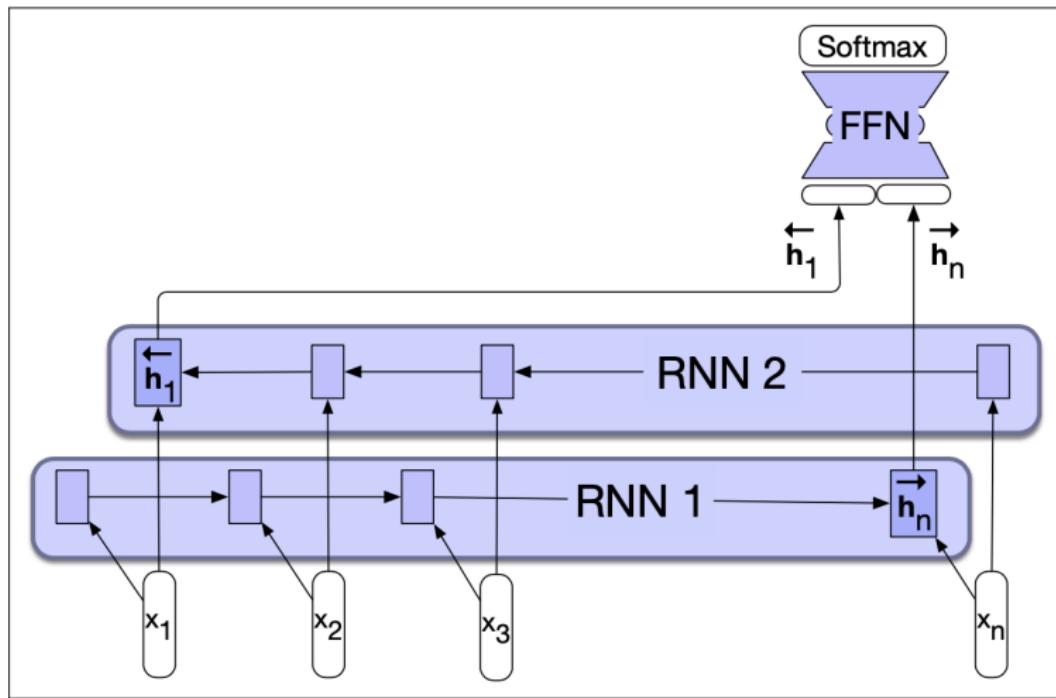
$$\mathbf{h}_t = [\mathbf{h}_t^f; \mathbf{h}_t^b] \quad (24)$$

$$= \mathbf{h}_t^f \oplus \mathbf{h}_t^b \quad (25)$$

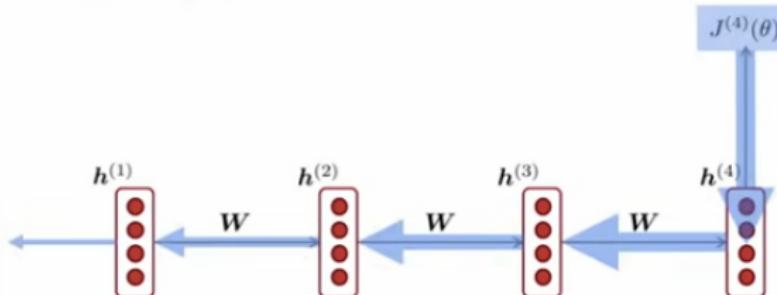
# Bi-directional RNN for Sequence Labelling



# Bi-directional RNN for Sequence Classification



## Vanishing gradient intuition



$$\frac{\partial J^{(4)}}{\partial \mathbf{h}^{(1)}} = \left( \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{h}^{(1)}} \times \frac{\partial \mathbf{h}^{(3)}}{\partial \mathbf{h}^{(2)}} \times \frac{\partial \mathbf{h}^{(4)}}{\partial \mathbf{h}^{(3)}} \right) \times \frac{\partial J^{(4)}}{\partial \mathbf{h}^{(4)}}$$

What happens if these are small?

**Vanishing gradient problem:**  
When these are small, the gradient signal gets smaller and smaller as it backpropagates further

# Long-distance Dependencies

Example

**The flights the airline was cancelling were full.**

**If trains leave the station near my home early, then they tend to be on time.**

**Who does Mike want more than anybody to win the race?**

**Whom does Mike want more than anybody to meet at the party?**

## Three Alternatives for Dealing with Sequences of Language Data (3)

- ▶ Presenting individual input tokens to an LSTM (short for: Long-Short Term Memory; Hochreiter and Schmidhuber 1997) or a GRU (short for: Gated Recurrent Unit; Cho et al. 2014)
- ▶ Basic problem with RNNs: The weights of the hidden layer are responsible for two tasks:
  - ▶ supply information for determining the output at the current time step  $t$
  - ▶ updating and carrying forward information from previous time steps  $t-M$  relevant for future decisions at time steps  $t+N$
- ▶ Basic idea underlying GRUs and LSTMs: separate these two tasks and thereby solving the vanishing/exploding gradient problem.

## **Added Complexity for neural units in GRUs and LSTMs**

- ▶ Introduction of an additional context vector for input and output to neural units (for LSTMs)
- ▶ Encapsulating additional gates within the neural units themselves
- ▶ With different internal structures for GRUs and LSTMs

# Gated Recurrent Unit

*Two most widely used gated recurrent units*

## Gated Recurrent Unit

[Cho et al., EMNLP2014;  
Chung, Gulcehre, Cho, Bengio, DLUFL2014]

$$h_t = u_t \odot \tilde{h}_t + (1 - u_t) \odot h_{t-1}$$

$$\tilde{h} = \tanh(W [x_t] + U(r_t \odot h_{t-1}) + b)$$

$$u_t = \sigma(W_u [x_t] + U_u h_{t-1} + b_u)$$

$$r_t = \sigma(W_r [x_t] + U_r h_{t-1} + b_r)$$

## Long Short-Term Memory

[Hochreiter & Schmidhuber, NC1999;  
Gers, Thesis2001]

$$h_t = o_t \odot \tanh(c_t)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

$$\tilde{c}_t = \tanh(W_c [x_t] + U_c h_{t-1} + b_c)$$

$$o_t = \sigma(W_o [x_t] + U_o h_{t-1} + b_o)$$

$$i_t = \sigma(W_i [x_t] + U_i h_{t-1} + b_i)$$

$$f_t = \sigma(W_f [x_t] + U_f h_{t-1} + b_f)$$

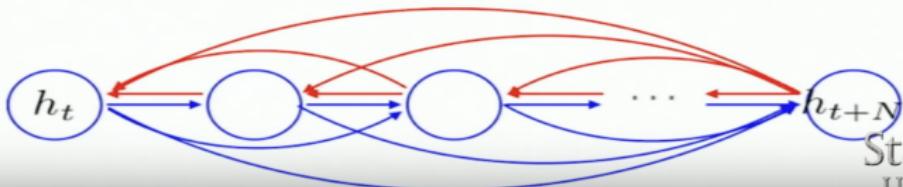
Stanford  
University

## Gated Recurrent Unit

- It implies that the error must backpropagate through all the intermediate nodes:

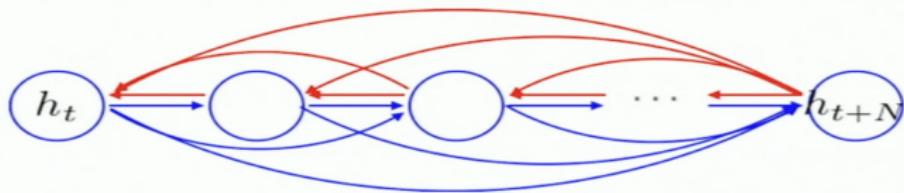


- Perhaps we can create shortcut connections.



## Gated Recurrent Unit

- Perhaps we can create *adaptive* shortcut connections.



$$f(h_{t-1}, x_t) = u_t \odot \tilde{h}_t + (1 - u_t) \odot h_{t-1}$$

- Candidate Update**  $\tilde{h}_t = \tanh(W [x_t] + U h_{t-1} + b)$
- Update gate**  $u_t = \sigma(W_u [x_t] + U_u h_{t-1} + b_u)$

# Hadamard Product

The Hadamard product for a  $3 \times 3$  matrix  $A$  with a  $3 \times 3$  matrix  $B$  is:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \circ \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & a_{13}b_{13} \\ a_{21}b_{21} & a_{22}b_{22} & a_{23}b_{23} \\ a_{31}b_{31} & a_{32}b_{32} & a_{33}b_{33} \end{bmatrix}$$

## Gated Recurrent Unit

- Let the net prune unnecessary connections *adaptively*.

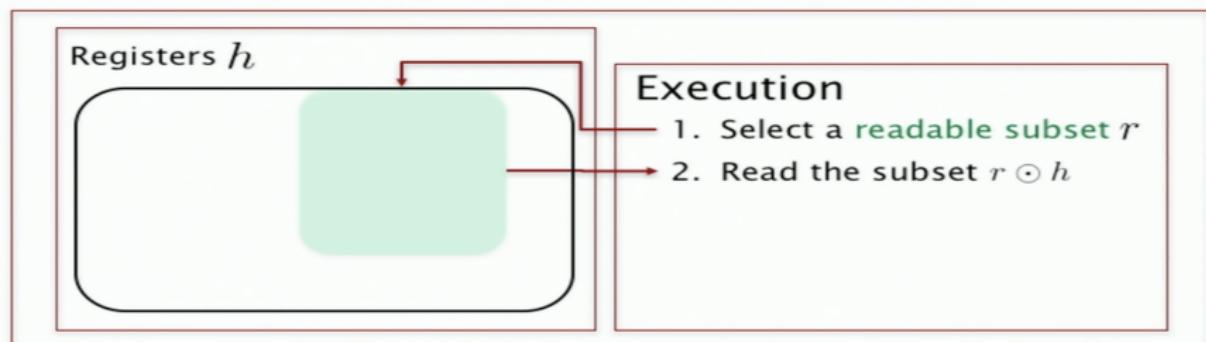


$$f(h_{t-1}, x_t) = u_t \odot \tilde{h}_t + (1 - u_t) \odot h_{t-1}$$

- Candidate Update  $\tilde{h}_t = \tanh(W [x_t] + U(r_t \odot h_{t-1}) + b)$
- Reset gate  $r_t = \sigma(W_r [x_t] + U_r h_{t-1} + b_r)$
- Update gate  $u_t = \sigma(W_u [x_t] + U_u h_{t-1} + b_u)$

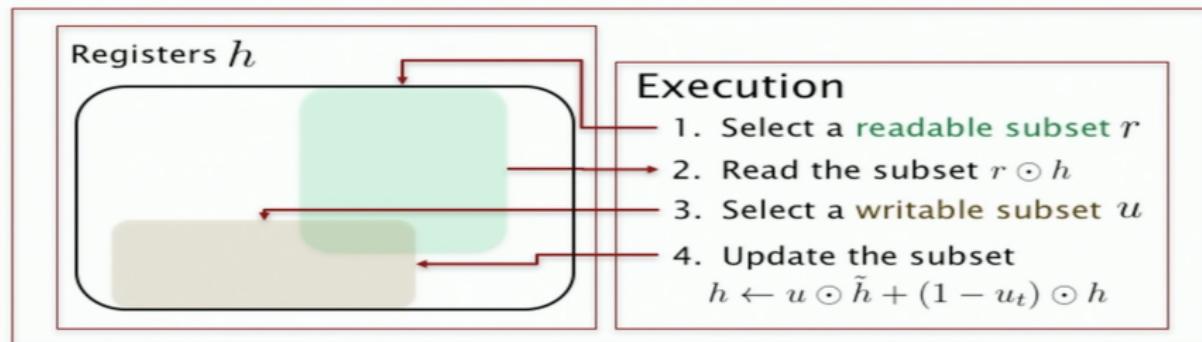
## Gated Recurrent Unit

GRU ...

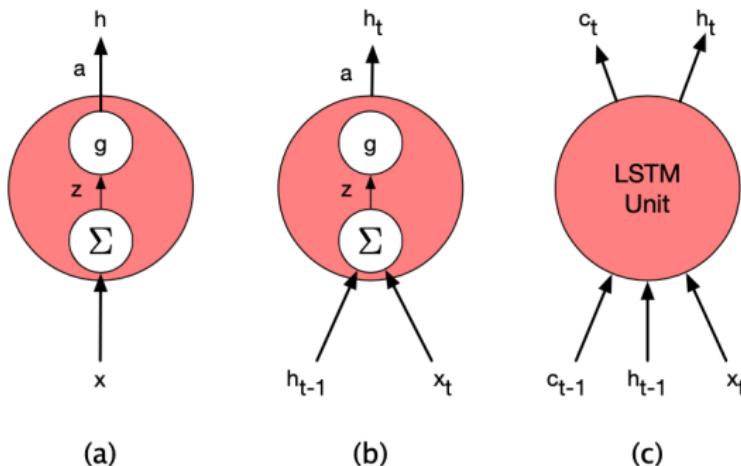


## Gated Recurrent Unit

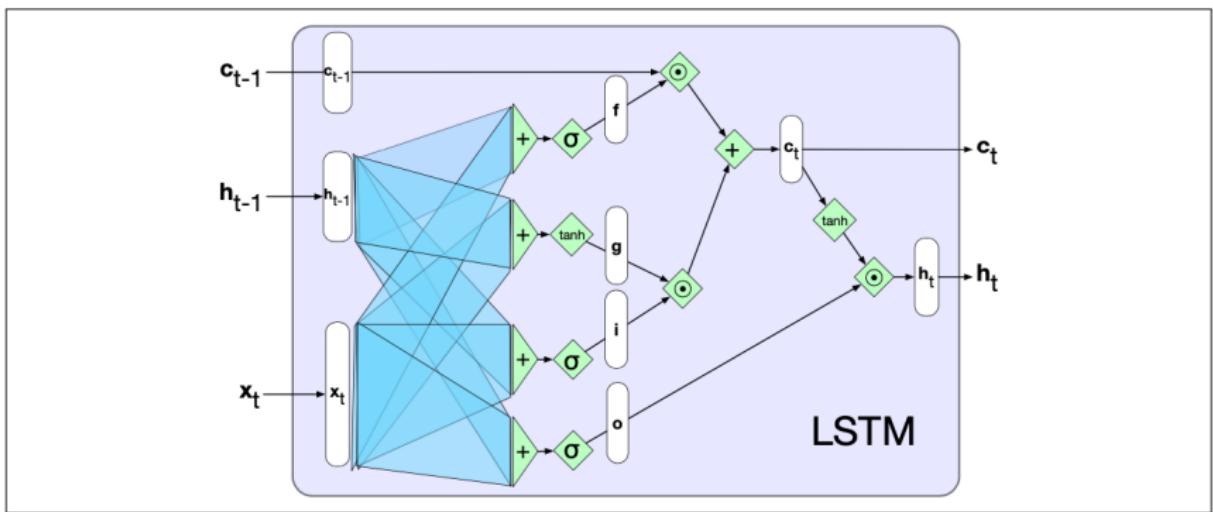
GRU ...



# Gated Units, Layers and Networks



# The LSTM



# The LSTM

$$\mathbf{f}_t = \sigma(\mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{W}_f \mathbf{x}_t) \quad (26)$$

$$\mathbf{k}_t = \mathbf{c}_{t-1} \odot \mathbf{f}_t \quad (27)$$

$$\mathbf{g}_t = \tanh(\mathbf{U}_g \mathbf{h}_{t-1} + \mathbf{W}_g \mathbf{x}_t) \quad (28)$$

# The LSTM

$$\mathbf{i}_t = \sigma(\mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{W}_i \mathbf{x}_t) \quad (29)$$

$$\mathbf{j}_t = \mathbf{g}_t \odot \mathbf{i}_t \quad (30)$$

$$\mathbf{c}_t = \mathbf{j}_t + \mathbf{k}_t \quad (31)$$

$$\mathbf{o}_t = \sigma(\mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{W}_o \mathbf{x}_t) \quad (32)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \quad (33)$$

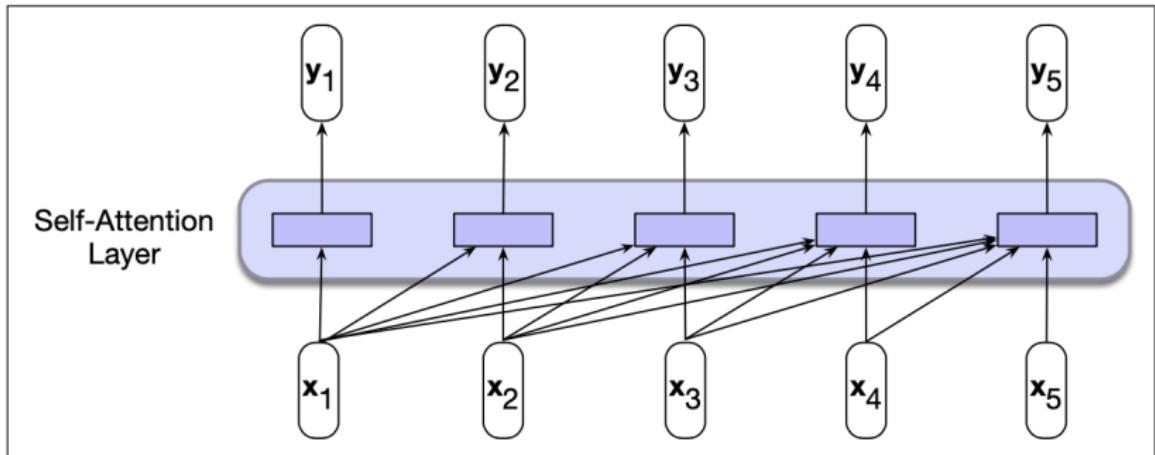
# Transformers

- ▶ Transformers are non-recurrent networks based on (self-)attention.
- ▶ Self-attention allows a network to directly extract and use information from arbitrarily large contexts without the need to pass them through intermediate recurrent connections as in RNNs.
- ▶ A self-attention layer maps input sequences to output sequences of the same length, using attention heads.
- ▶ Attention heads model how the surrounding words are relevant for the processing of the current word.

## Flow of Information in a Self-Attention Layer

- ▶ When processing each item in the input of a self-attention layer, the model has access to all inputs up to and including the one under consideration, but no access to information about inputs beyond the current one.
- ▶ The computation performed for each item is independent of all the other computations. Hence, forward inference and training can proceed in parallel.

# Self-Attention Layer



# Self-Attention Networks: Transformers

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j \quad (34)$$

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \quad \forall j \leq i \quad (35)$$

$$= \frac{\exp(\text{score}(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k=1}^i \exp(\text{score}(\mathbf{x}_i, \mathbf{x}_k))} \quad \forall j \leq i \quad (36)$$

$$\mathbf{y}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j \quad (37)$$

# Self-Attention Networks: Different Roles

An input embedding can play three different roles:

- ▶ **query**: as the *current focus of attention* that is compared to all of the other preceding inputs.
- ▶ **key**: as a *preceding input* that is compared to the current focus of attention.
- ▶ **value** that is used to compute the output for the current focus of attention.

## Self-Attention Networks: Transformers

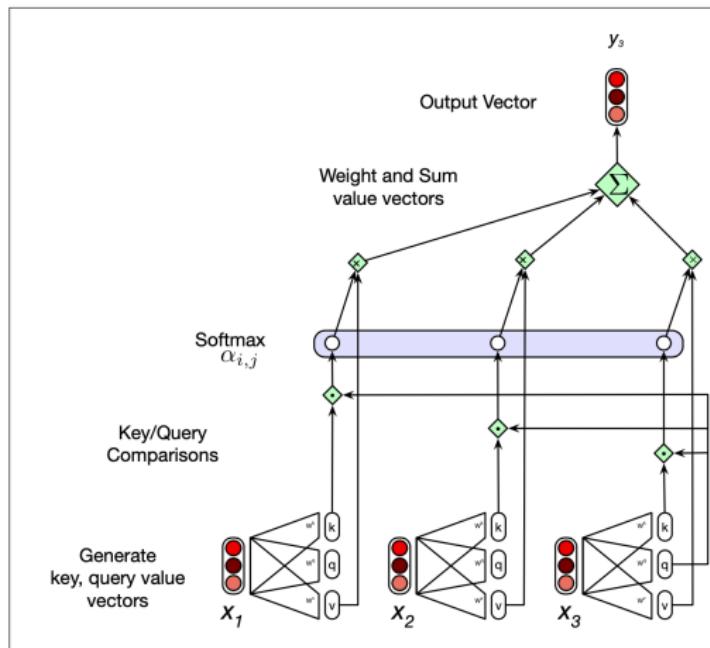
- ▶ The roles of query, key, and value are differentiated by three different weight matrices:  $\mathbf{W}^Q \in \mathbb{R}^{d \times d'}$ ,  $\mathbf{W}^K \in \mathbb{R}^{d \times d'}$ , and  $\mathbf{W}^V \in \mathbb{R}^{d \times d''}$ .
- ▶ The inputs and outputs of transformers, as well as the intermediate vectors after the various layers, all have the same dimensionality  $1 \times d$ .

$$\mathbf{q}_i = \mathbf{W}^Q \mathbf{x}_i; \quad \mathbf{k}_i = \mathbf{W}^K \mathbf{x}_i; \quad \mathbf{v}_i = \mathbf{W}^V \mathbf{x}_i \quad (38)$$

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{q}_i \cdot \mathbf{k}_j \quad (39)$$

$$\mathbf{y}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j \quad (40)$$

# Calculation of an Output Embedding in a Self-Attention Layer



## Self-Attention Networks: Further Adjustments

Scaling the dot product by the square root of the dimensionality

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}} \quad (41)$$

Parallelizing the Computation: packing the input embeddings of the N tokens into a single matrix

$$\mathbf{Q} = \mathbf{XW}^Q; \mathbf{K} = \mathbf{XW}^K; \mathbf{V} = \mathbf{XW}^V; \quad (42)$$

## Final Result: Reducing the Self-Attention Step for an Entire Sequence of N Tokens

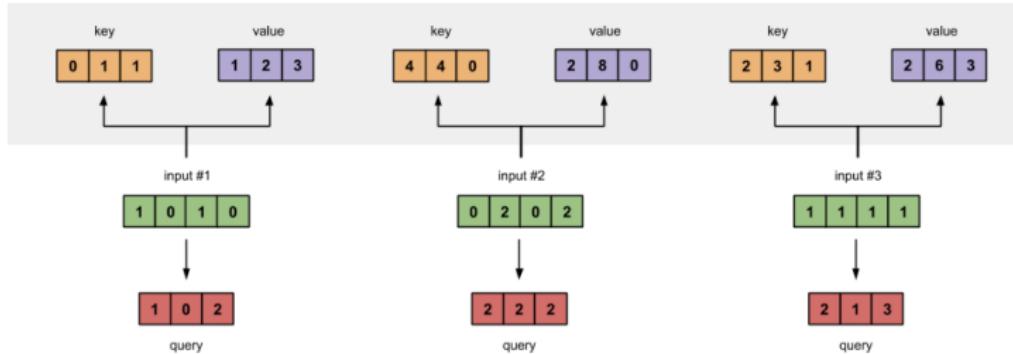
$$\text{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax} \left( \frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} \right) \mathbf{V} \quad (43)$$

# Working through an Example with 3 Input Vectors

1. Prepare inputs
2. Initialise weights
3. Derive key, query and value
4. Calculate attention scores for Input 1
5. Calculate softmax
6. Multiply scores with values
7. Sum weighted values to get Output 1
8. Repeat steps 4–7 for Input 2 and Input 3

This workflow and the illustrations in the next three slides are due to <https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a>

# Step 1-3



# Initialize Weights for Query, Key, and Value

Weights for key:

```
[[0, 0, 1],  
 [1, 1, 0],  
 [0, 1, 0],  
 [1, 1, 0]]
```

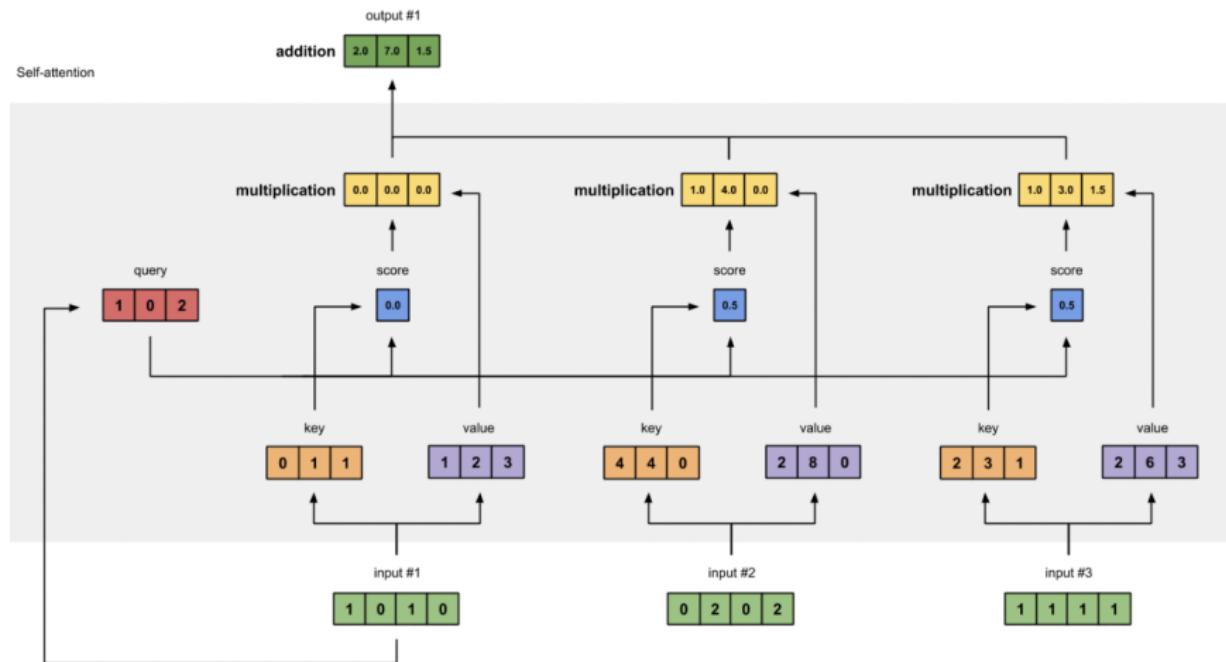
Weights for query:

```
[[1, 0, 1],  
 [1, 0, 0],  
 [0, 0, 1],  
 [0, 1, 1]]
```

Weights for value:

```
[[0, 2, 0],  
 [0, 3, 0],  
 [1, 0, 3],  
 [1, 1, 0]]
```

# Calculate Attention Scores



## Calculate Softmax

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^k \exp(z_j)} \quad 1 \leq i \leq k \quad (44)$$

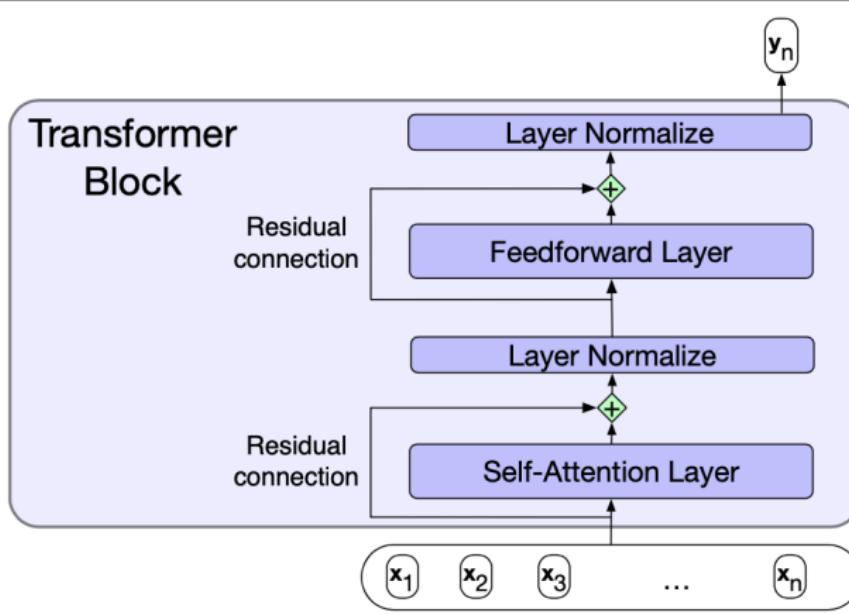
# Zeroing Out the Upper Triangular Portion of a $QT^T$ Matrix

N	q1•k1	-∞	-∞	-∞	-∞
	q2•k1	q2•k2	-∞	-∞	-∞
	q3•k1	q3•k2	q3•k3	-∞	-∞
	q4•k1	q4•k2	q4•k3	q4•k4	-∞
	q5•k1	q5•k2	q5•k3	q5•k4	q5•k5

# Transformer Blocks

- ▶ A transformer block consists of a single attention layer followed by a feed-forward layer with residual connections and layer normalizations following each.
- ▶ Transformer blocks can be stacked to make deeper and more powerful networks.

# Transformer Blocks



# Transformer Blocks

$$\mathbf{z} = \text{LayerNorm}(\mathbf{x} + \text{SelfAttn}(\mathbf{x})) \quad (45)$$

$$\mathbf{y} = \text{LayerNorm}(\mathbf{z} + \text{FFNN}(\mathbf{z})) \quad (46)$$

$$\mu = \frac{1}{d_h} \sum_{i=1}^{d_h} x_i \quad (47)$$

$$\sigma = \sqrt{\frac{1}{d_h} \sum_{i=1}^{d_h} (x_i - \mu)^2} \quad (48)$$

$$\hat{\mathbf{x}} = \frac{(\mathbf{x} - \mu)}{\sigma} \quad (49)$$

$$\text{LayerNorm} = \gamma \hat{\mathbf{x}} + \beta \quad (50)$$

# Multihead Attention Layers

- ▶ are sets of self-attention layers, each with its own set of key, query, and value matrices:
  - ▶  $\mathbf{W}_i^Q \in \mathbb{R}^{d \times d_k}$
  - ▶  $\mathbf{W}_i^K \in \mathbb{R}^{d \times d_k}$
  - ▶  $\mathbf{W}_i^V \in \mathbb{R}^{d \times d_v}$
- ▶ Each member of such a set of self-attention layers is called a **head**
- ▶ Each head gets multiplied by the inputs packed into  $\mathbf{X}$  to produce
  - ▶  $\mathbf{W}_i^Q \in \mathbb{R}^{N \times d_k}$
  - ▶  $\mathbf{W}_i^K \in \mathbb{R}^{N \times d_k}$
  - ▶  $\mathbf{W}_i^V \in \mathbb{R}^{N \times d_v}$

# Multihead Attention Layers

- ▶ The output of a multi-head layer with  $h$  heads consists of  $h$  vectors of shape  $N \times d_v$
- ▶ These outputs are concatenated from each head and then, using a linear projection with weight matrix  $\mathbf{W}_i^O \in \mathbb{R}^{hd_k \times d}$ , reduced to  $N \times d$  output.

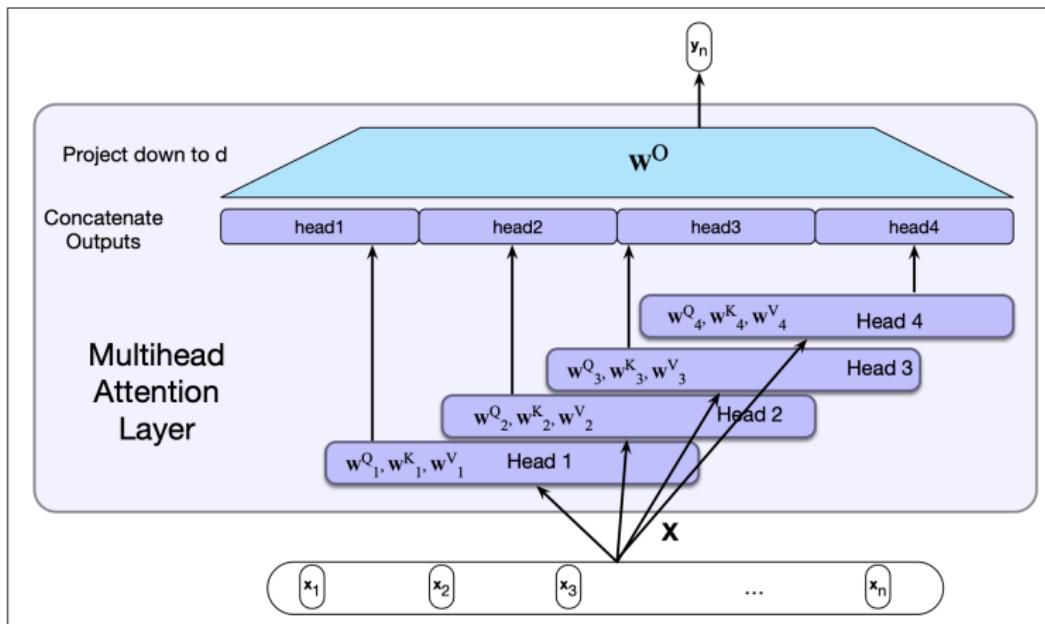
# Multihead Attention

$$\text{MultiHeadAttn}(\mathbf{X}) = (\mathbf{head}_1 \oplus \mathbf{head}_2 \dots \oplus \mathbf{head}_h) \mathbf{W}^O \quad (51)$$

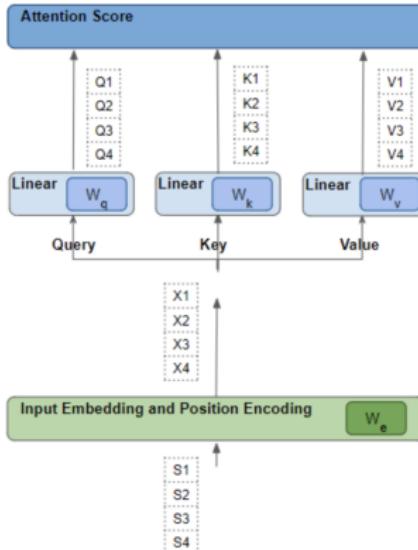
$$\mathbf{Q} = \mathbf{X} \mathbf{W}_i^Q; \quad \mathbf{K} = \mathbf{X} \mathbf{W}_i^K; \quad \mathbf{V} = \mathbf{X} \mathbf{W}_i^V \quad (52)$$

$$\mathbf{head}_i = \text{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \quad (53)$$

# Multihead Attention

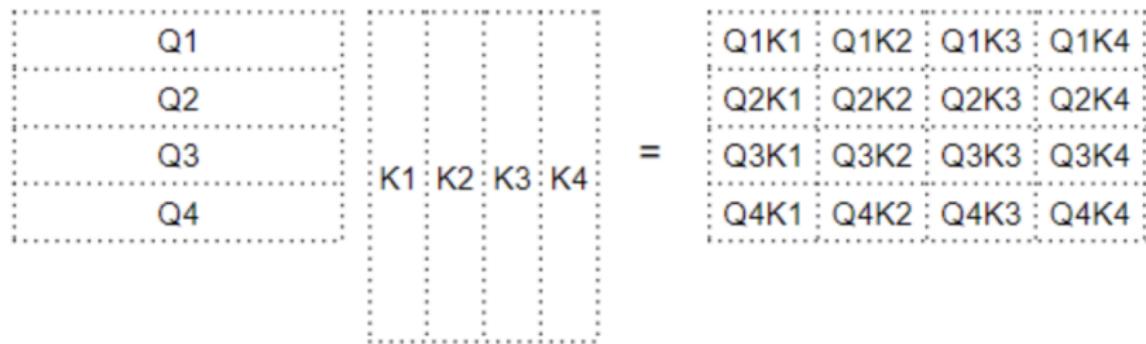


# Multihead Attention: Visual Summary



Slide due to Keitan Doshi, <https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-why-they-work-so-well-d840bd61a9d3>

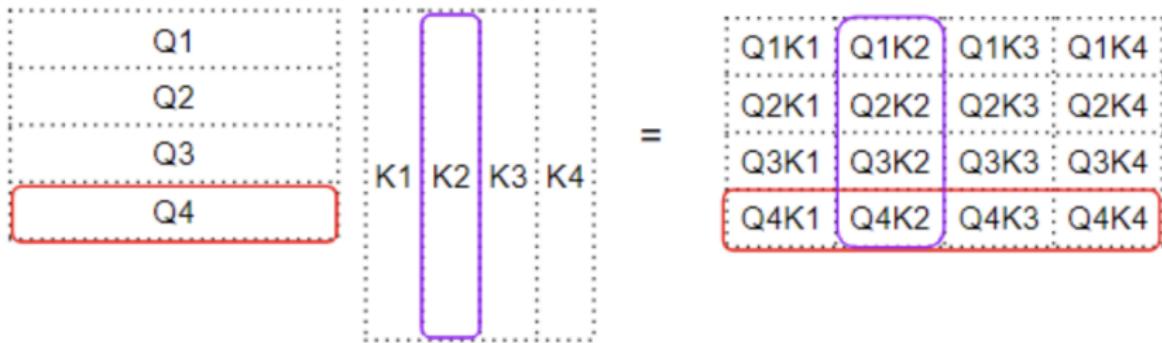
# Multihead Attention: Dot Product between Query and Key matrices



Slide due to Keitan Doshi

<https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-why-they-work-so-well-d840bd61a9d3>

# Multihead Attention: Dot Product between Query and Key matrices



Slide due to Keitan Doshi

<https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-why-they-work-so-well-d840bd61a9d3>

# Dot Product between Query-Key and Value Matrices

$$\begin{array}{|c|c|c|c|} \hline Q1K1 & Q1K2 & Q1K3 & Q1K4 \\ \hline Q2K1 & Q2K2 & Q2K3 & Q2K4 \\ \hline Q3K1 & Q3K2 & Q3K3 & Q3K4 \\ \hline Q4K1 & Q4K2 & Q4K3 & Q4K4 \\ \hline \end{array} \times \begin{array}{|c|} \hline V1 \\ \hline V2 \\ \hline V3 \\ \hline V4 \\ \hline \end{array} = \begin{array}{|c|} \hline Q1K1V1 + Q1K2V2 + Q1K3V3 + Q1K4V4 \\ \hline Q2K1V1 + Q2K2V2 + Q2K3V3 + Q2K4V4 \\ \hline Q3K1V1 + Q3K2V2 + Q3K3V3 + Q3K4V4 \\ \hline Q4K1V1 + Q4K2V2 + Q4K3V3 + Q4K4V4 \\ \hline \end{array}$$
$$= \begin{array}{|c|} \hline Z1 \\ \hline Z2 \\ \hline Z3 \\ \hline Z4 \\ \hline \end{array}$$

Slide due to Keitan Doshi

<https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-why-they-work-so-well-d840bd61a9d3>

# Attention Score for the word blue pays attention to every other word

$$Z_4 = (Q_4 K_1) V_1 + (Q_4 K_2) V_2 + (Q_4 K_3) V_3 + (Q_4 K_4) V_4$$

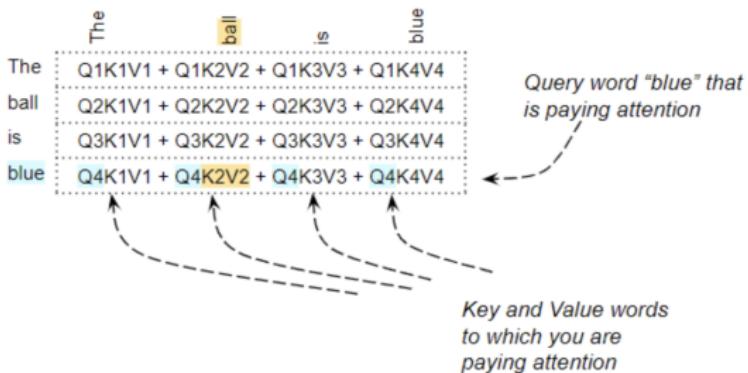
The diagram illustrates the calculation of  $Z_4$  as a weighted sum of vectors  $V_1, V_2, V_3, V_4$ . The equation is  $Z_4 = (Q_4 K_1) V_1 + (Q_4 K_2) V_2 + (Q_4 K_3) V_3 + (Q_4 K_4) V_4$ . Three dashed arrows point from labels above the equation to specific terms in the sum:

- A dashed arrow points down to the term  $(Q_4 K_1) V_1$ , labeled "Fourth word Score".
- A dashed arrow points down to the term  $(Q_4 K_2) V_2$ , labeled "Fourth Query word \* first Key word".
- A dashed arrow points up to the term  $(Q_4 K_3) V_3$ , labeled "Fourth Query word \* second Key word".

Slide due to Keitan Doshi

<https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-why-they-work-so-well-d840bd61a9d3>

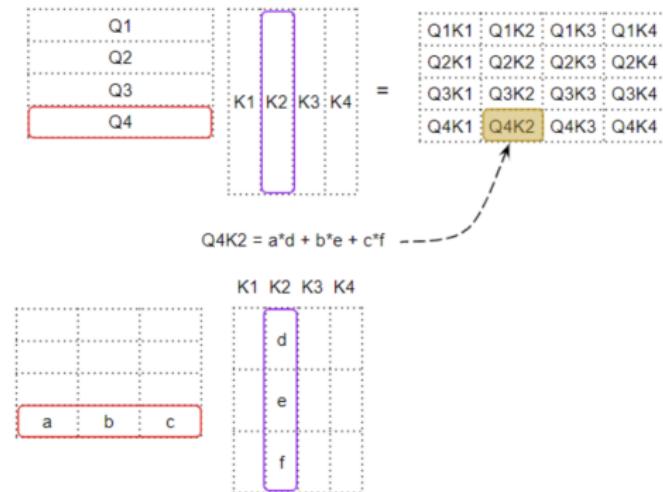
# Dot Product between Query-Key and Value Matrices



Slide due to Keitan Doshi

<https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-why-they-work-so-well-d840bd61a9d3>

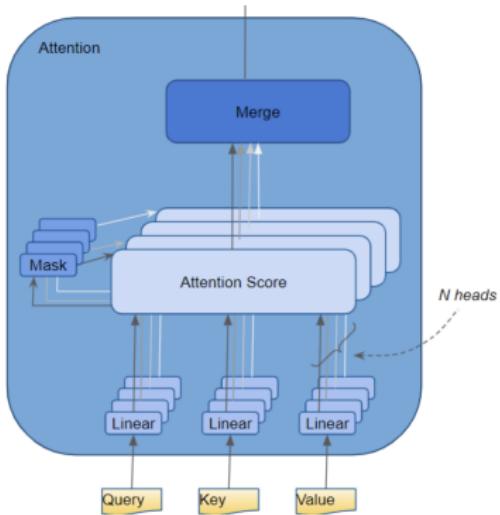
# Each cell is a dot product between two word vectors



Slide due to Keitan Doshi

<https://towardsdatascience.com/transformers-explained-visually-not-just-how-but->

# Multihead Attention: Visual Summary



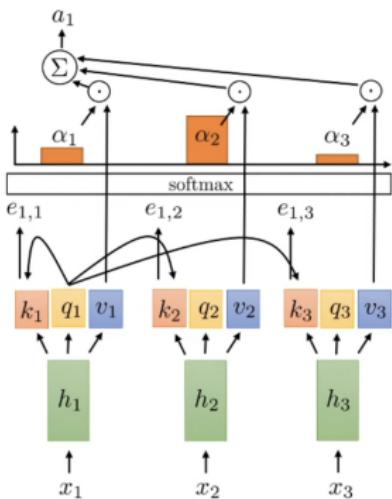
Slide due to Keitan Doshi

<https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-why-they-work-so-well-d840bd61a9d3>

## Modeling word order: positional embedding

- ▶ Transformers models do not come with built-in information about sequence order such as time-step information in an RNN
- ▶ Transformers models do not have any notion of relative or absolute positions of the tokens in a sequence.
- ▶ In order to model word order, positional embeddings can be combined with input embeddings.
- ▶ Positional embeddings that are specific to each position in an input sequence.

# Positional encoding: what is the order?



**what we see:**

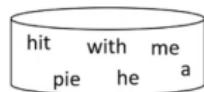
he hit me with a pie

**what naïve self-attention sees:**

a pie hit me with he

a hit with me he pie

he pie me with a hit



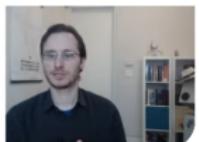
most alternative orderings are nonsense, but some change the meaning

**in general** the position of words in a sentence carries information!

**Idea:** add some information to the representation at the beginning that indicates where it is in the sequence!

$$h_t = f(x_t, t)$$

some function



Slide due to RAIL CS182: Lecture 12:Part 2: Transformers  
[https://www.youtube.com/watch?v=4AzsiCMw\\_-s](https://www.youtube.com/watch?v=4AzsiCMw_-s)

# Positional encoding: sin/cos

Naïve positional encoding: just append  $t$  to the input

$$\bar{x}_t = \begin{bmatrix} x_t \\ t \end{bmatrix}$$

This is not a great idea, because **absolute** position is less important than **relative** position

I walk my dog every day



every single day I walk my dog



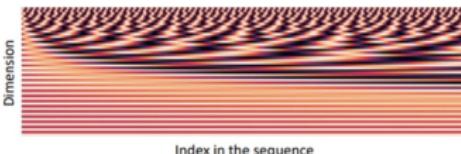
The fact that "my dog" is right after "I walk" is the important part, not its absolute position

we want to represent **position** in a way that tokens with similar **relative** position have similar **positional encoding**

Idea: what if we use **frequency-based** representations?

$$p_t = \begin{bmatrix} \sin(t/10000^{2*1/d}) \\ \cos(t/10000^{2*1/d}) \\ \sin(t/10000^{2*2/d}) \\ \cos(t/10000^{2*2/d}) \\ \dots \\ \sin(t/10000^{2*\frac{d}{2}/d}) \\ \cos(t/10000^{2*\frac{d}{2}/d}) \end{bmatrix}$$

dimensionality of positional encoding

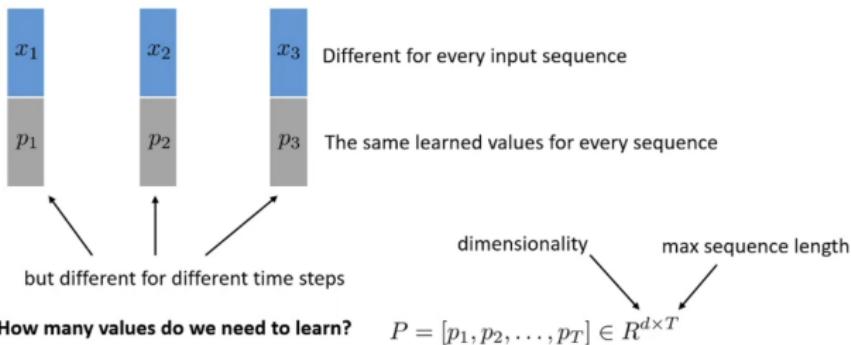


Slide due to RAIL CS182: Lecture 12:Part 2: Transformers

[https://www.youtube.com/watch?v=4AzsiCMw\\_-s](https://www.youtube.com/watch?v=4AzsiCMw_-s)

# Positional encoding: learned

Another idea: just learn a positional encoding



+ more flexible (and perhaps more optimal) than sin/cos encoding

+ a bit more complex, need to pick a max sequence length (and can't generalize beyond it)



Slide due to RAIL CS182: Lecture 12:Part 2: Transformers  
[https://www.youtube.com/watch?v=4AzsiCMw\\_-s](https://www.youtube.com/watch?v=4AzsiCMw_-s)

# How to incorporate positional encoding?

At each step, we have  $x_t$  and  $p_t$

**Simple choice:** just concatenate them

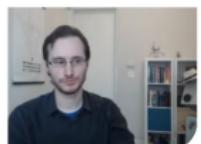
$$\bar{x}_t = \begin{bmatrix} x_t \\ p_t \end{bmatrix}$$

**More often:** just add after **embedding** the input

input to self-attention is  $\text{emb}(x_t) + p_t$

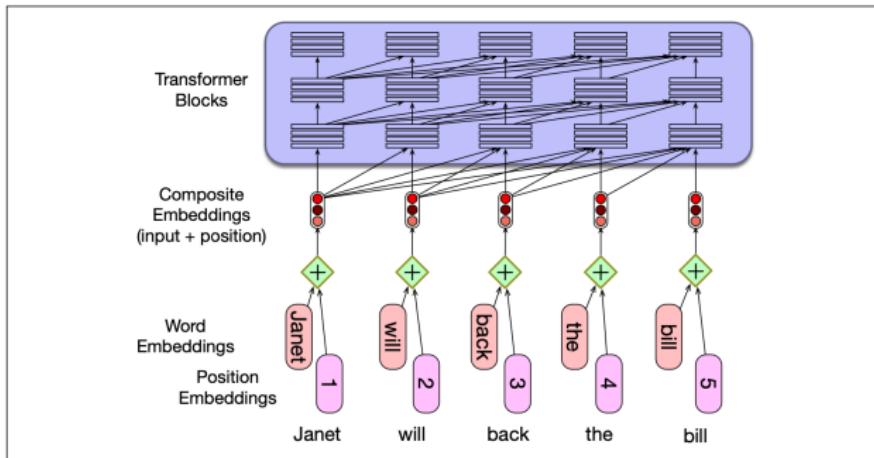


some learned function (e.g., some fully connected layers with linear layers + nonlinearities)

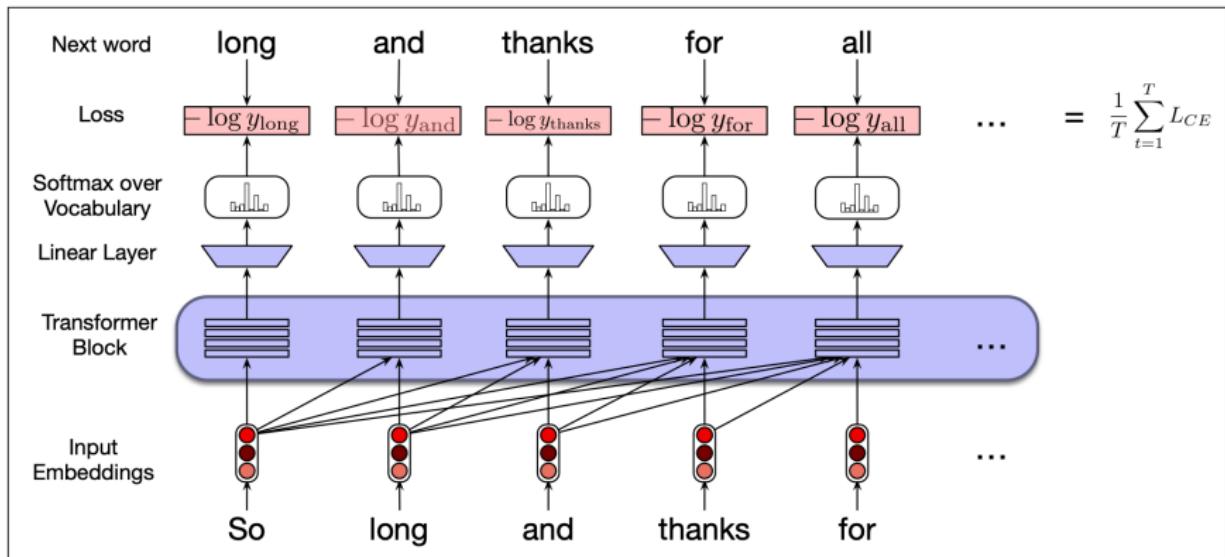


Slide due to RAIL CS182: Lecture 12:Part 2: Transformers  
[https://www.youtube.com/watch?v=4AzsiCMw\\_-s](https://www.youtube.com/watch?v=4AzsiCMw_-s)

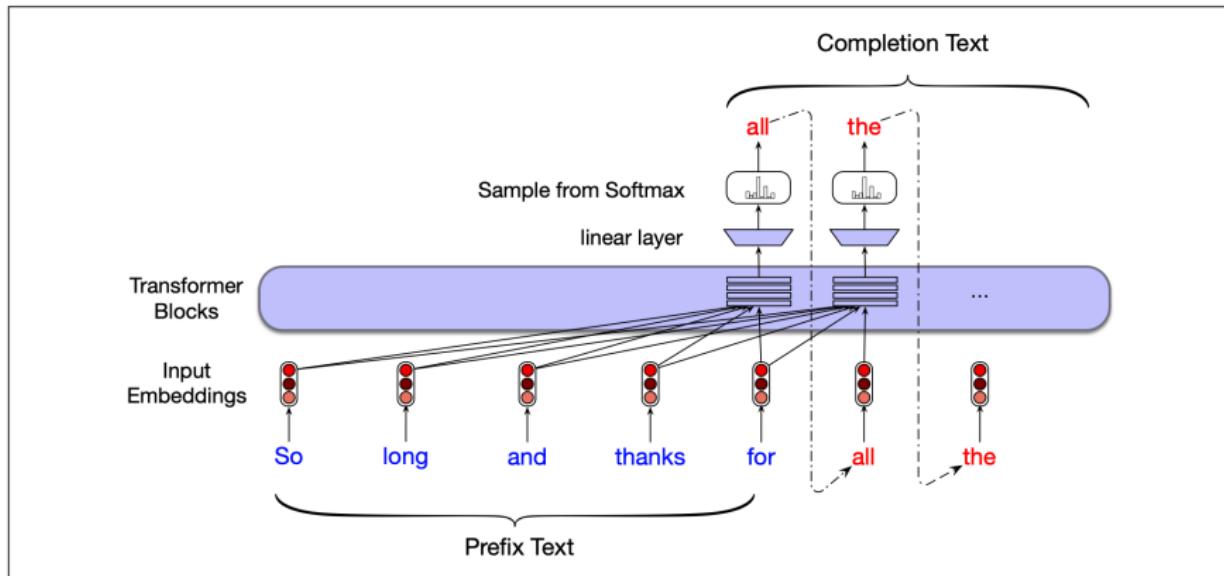
# Multihead Attention



# Transformers as Language Models



# Contextual Generation and Summarization



# Contextual Generation and Summarization

## Original Article

The only thing crazier than a guy in snowbound Massachusetts boxing up the powdery white stuff and offering it for sale online? People are actually buying it. For \$89, self-styled entrepreneur Kyle Waring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box – enough for 10 to 15 snowballs, he says. But not if you live in New England or surrounding states. “We will not ship snow to any states in the northeast!” says Waring’s website, [ShipSnowYo.com](http://ShipSnowYo.com). “We’re in the business of expunging snow!”

## **Contextual Generation and Summarization**

Continued

His website and social media accounts claim to have filled more than 133 orders for snow – more than 30 on Tuesday alone, his busiest day yet. With more than 45 total inches, Boston has set a record this winter for the snowiest month in its history. Most residents see the huge piles of snow choking their yards and sidewalks as a nuisance, but Waring saw an opportunity.

## Contextual Generation and Summarization

Continued

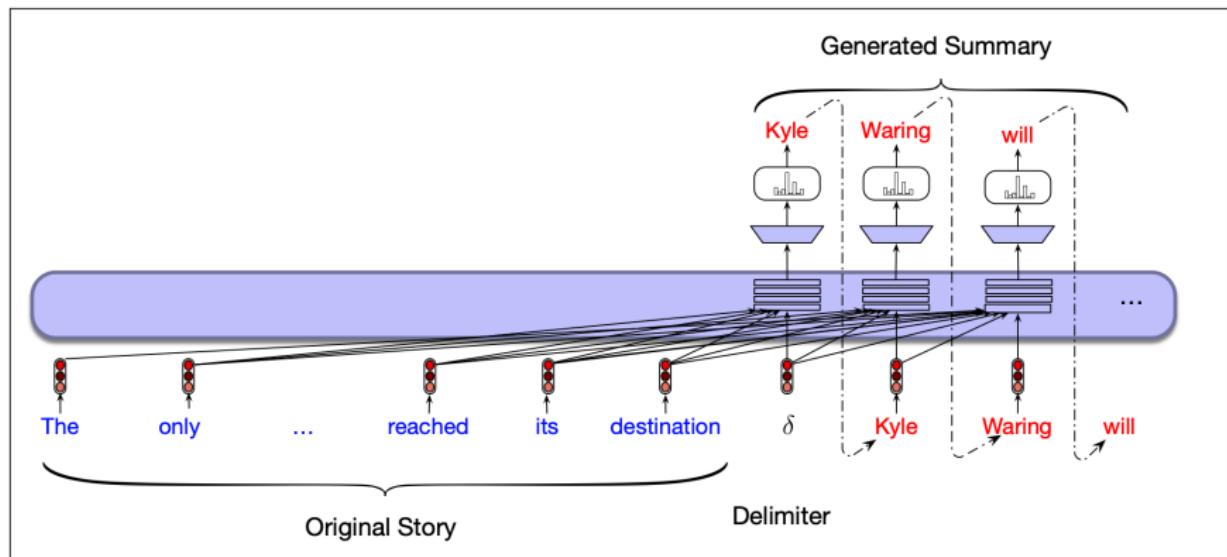
According to Boston.com, it all started a few weeks ago, when Waring and his wife were shoveling deep snow from their yard in Manchester-by-the-Sea, a coastal suburb north of Boston. He joked about shipping the stuff to friends and family in warmer states, and an idea was born. His business slogan: "Our nightmare is your dream!" At first, ShipSnowYo sold snow packed into empty 16.9-ounce water bottles for \$19.99, but the snow usually melted before it reached its destination...

# **Contextual Generation and Summarization**

## Summary

Kyle Waring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box – enough for 10 to 15 snowballs, he says. But not if you live in New England or surrounding states.

# Contextual Generation and Summarization



# **Transformers and Large Language Models**

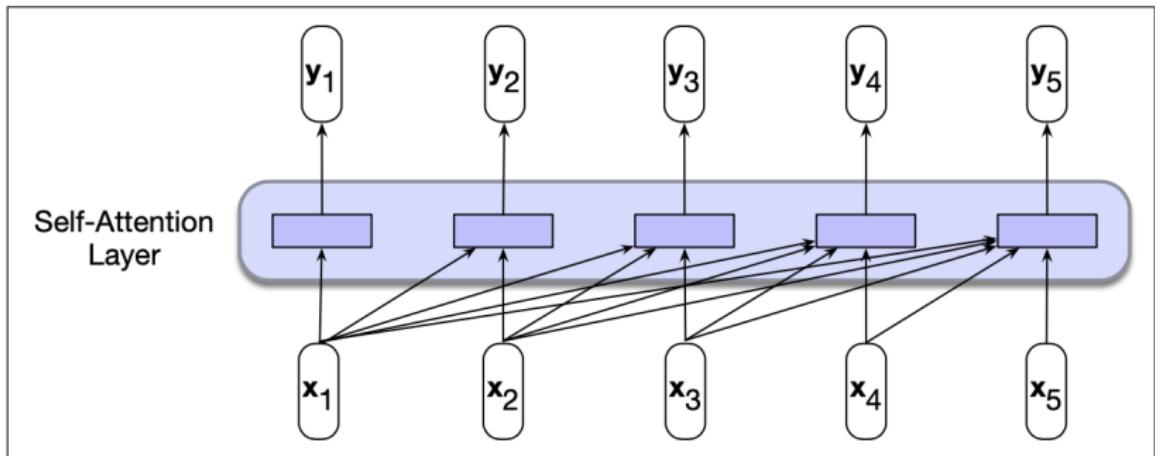
Erhard Hinrichs

Seminar für Sprachwissenschaft  
Eberhard-Karls Universität Tübingen

# Transformers

- ▶ Transformers are non-recurrent networks based on (self-)attention.
- ▶ Self-attention allows a network to directly extract and use information from arbitrarily large contexts without the need to pass them through intermediate recurrent connections as in RNNs.
- ▶ A self-attention layer maps input sequences to output sequences of the same length, using attention heads.
- ▶ Attention heads model how the surrounding words are relevant for the processing of the current word.

# Self-Attention Layer



## Relevant Linguistic Examples Motivating Self-Attention

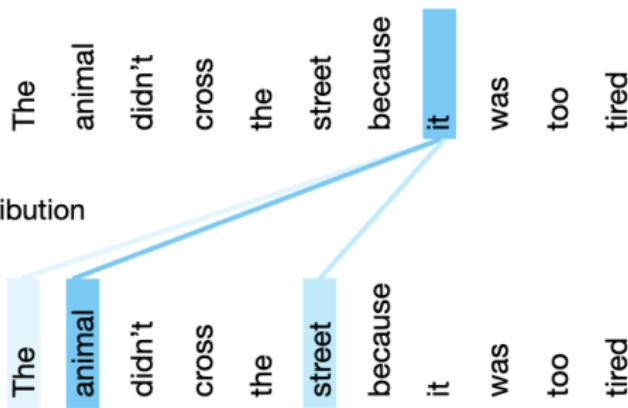
1. The **keys** to the cabinet **are** on the table.
2. The **chicken** crossed the road because **it** wanted to get to the other side.
3. I walked along the **pond**, and noticed that one of the trees long the **bank** had fallen into the **water** after the storm.

# Self-Attention Weight Distribution

Layer 6

self-attention distribution

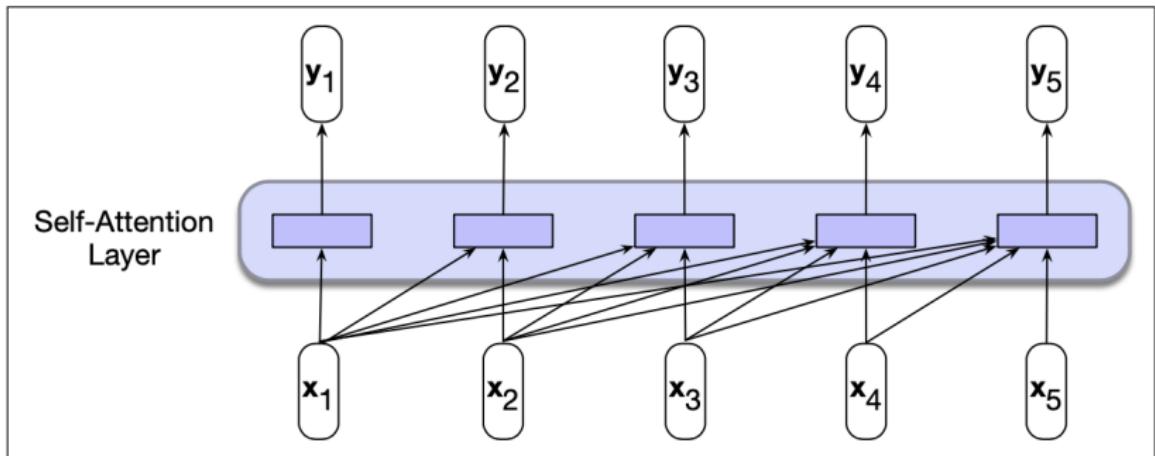
Layer 5



# Flow of Information in a Self-Attention Layer

- ▶ When processing each item in the input of a self-attention layer, the model has access to all inputs up to and including the one under consideration, but no access to information about inputs beyond the current one.
- ▶ The computation performed for each item is independent of all the other computations. Hence, forward inference and training can proceed in parallel.

# Self-Attention Layer



# Self-Attention Networks: Transformers

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j \quad (1)$$

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \quad \forall j \leq i \quad (2)$$

$$= \frac{\exp(\text{score}(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k=1}^i \exp(\text{score}(\mathbf{x}_i, \mathbf{x}_k))} \quad \forall j \leq i \quad (3)$$

$$\mathbf{y}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j \quad (4)$$

# Self-Attention Networks: Different Roles

An input embedding can play three different roles:

- ▶ **query**: as the *current focus of attention* that is compared to all of the other preceding inputs.
- ▶ **key**: as a *preceding input* that is compared to the current focus of attention.
- ▶ **value** that is used to compute the output for the current focus of attention.

## Self-Attention Networks: Transformers

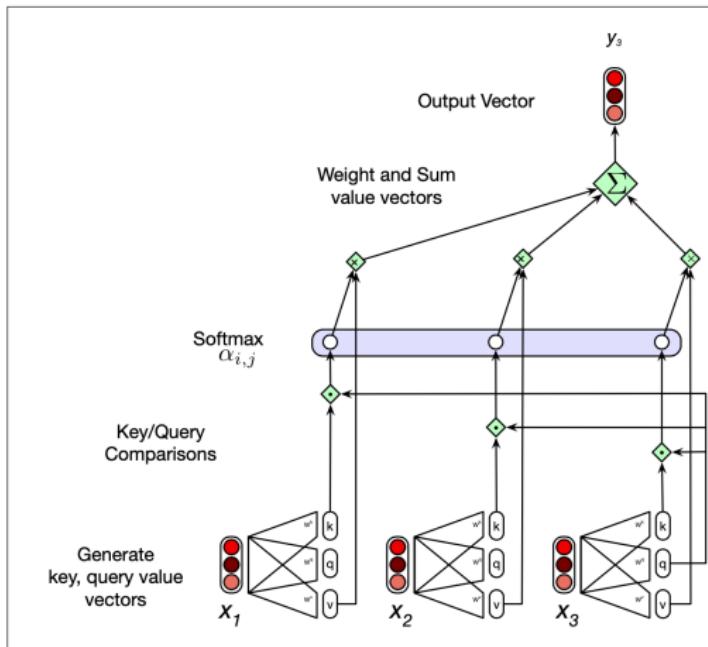
- ▶ The roles of query, key, and value are differentiated by three different weight matrices:  $\mathbf{W}^Q \in \mathbb{R}^{d \times d'}$ ,  $\mathbf{W}^K \in \mathbb{R}^{d \times d'}$ , and  $\mathbf{W}^V \in \mathbb{R}^{d \times d''}$ .
- ▶ The inputs and outputs of transformers, as well as the intermediate vectors after the various layers, all have the same dimensionality  $1 \times d$ .

$$\mathbf{q}_i = \mathbf{W}^Q \mathbf{x}_i; \quad \mathbf{k}_i = \mathbf{W}^K \mathbf{x}_i; \quad \mathbf{v}_i = \mathbf{W}^V \mathbf{x}_i \quad (5)$$

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{q}_i \cdot \mathbf{k}_j \quad (6)$$

$$\mathbf{y}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j \quad (7)$$

# Calculation of an Output Embedding in a Self-Attention Layer



## Self-Attention Networks: Further Adjustments

Scaling the dot product by the square root of the dimensionality

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}} \quad (8)$$

Parallelizing the Computation: packing the input embeddings of the N tokens into a single matrix

$$\mathbf{Q} = \mathbf{XW}^Q; \mathbf{K} = \mathbf{XW}^K; \mathbf{V} = \mathbf{XW}^V; \quad (9)$$

## Final Result: Reducing the Self-Attention Step for an Entire Sequence of N Tokens

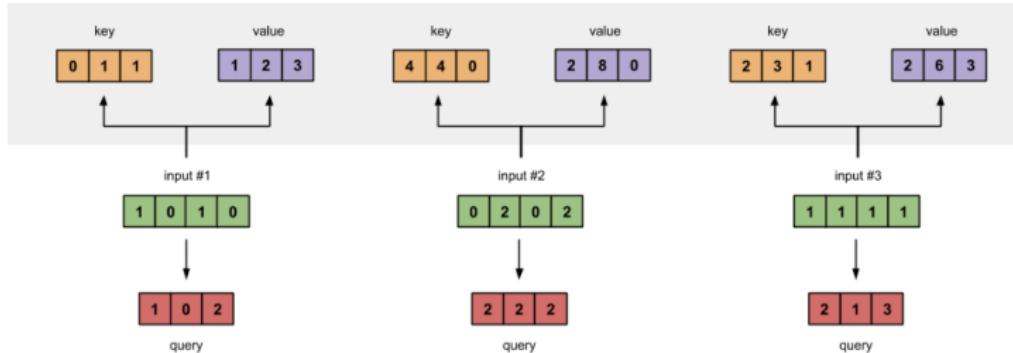
$$\text{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax} \left( \frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} \right) \mathbf{V} \quad (10)$$

# Working through an Example with 3 Input Vectors

1. Prepare inputs
2. Initialise weights
3. Derive key, query and value
4. Calculate attention scores for Input 1
5. Calculate softmax
6. Multiply scores with values
7. Sum weighted values to get Output 1
8. Repeat steps 4–7 for Input 2 and Input 3

This workflow and the illustrations in the next three slides are due to <https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a>

# Step 1-3



# Initialize Weights for Query, Key, and Value

Weights for key:

```
[[0, 0, 1],  
 [1, 1, 0],  
 [0, 1, 0],  
 [1, 1, 0]]
```

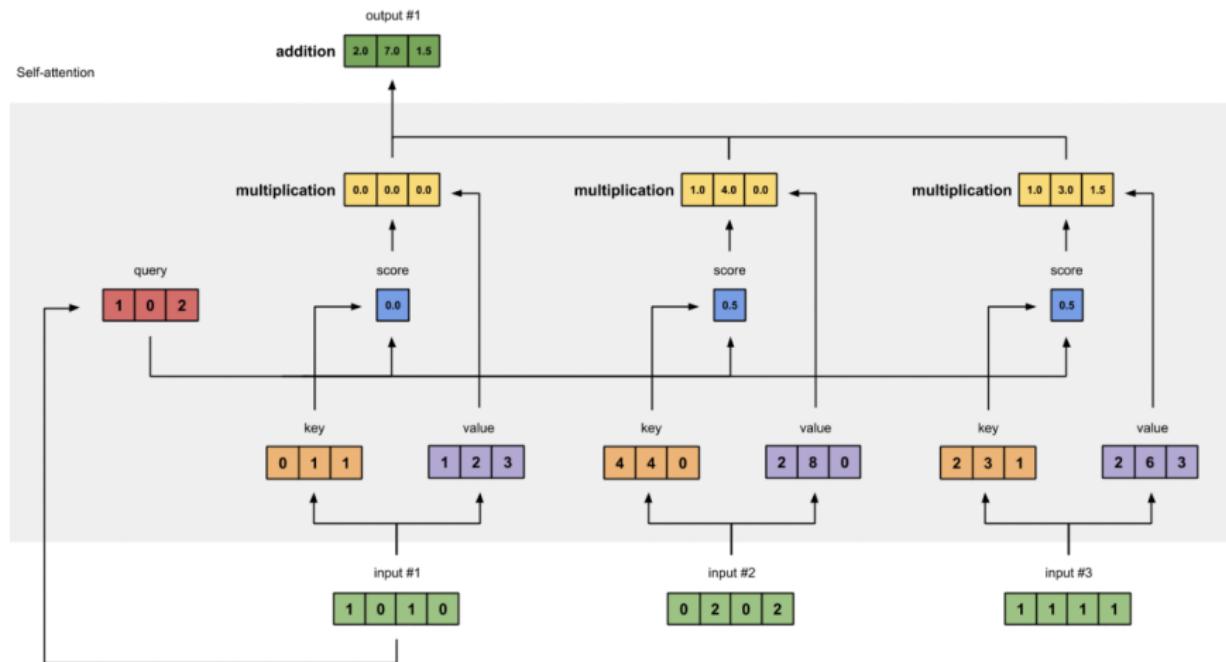
Weights for query:

```
[[1, 0, 1],  
 [1, 0, 0],  
 [0, 0, 1],  
 [0, 1, 1]]
```

Weights for value:

```
[[0, 2, 0],  
 [0, 3, 0],  
 [1, 0, 3],  
 [1, 1, 0]]
```

# Calculate Attention Scores



## Calculate Softmax

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^k \exp(z_j)} \quad 1 \leq i \leq k \quad (11)$$

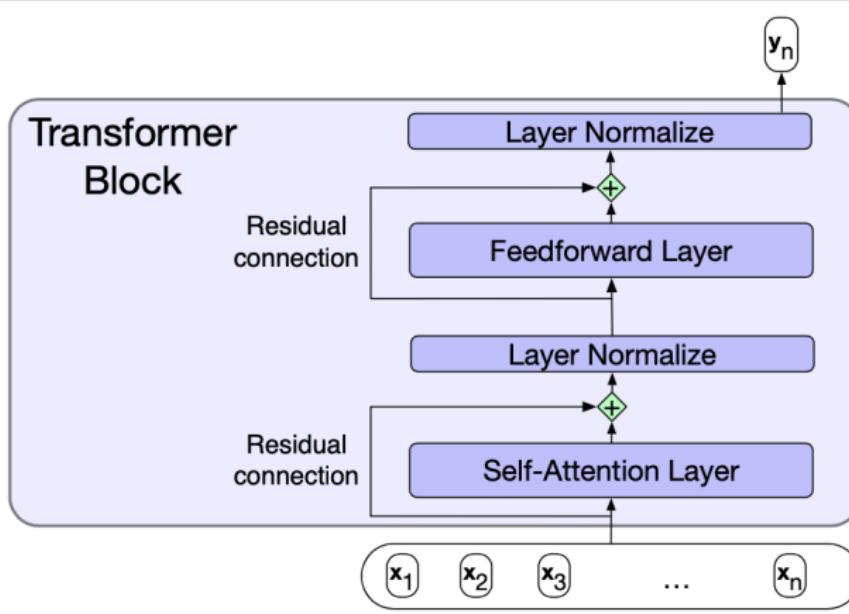
# Zeroing Out the Upper Triangular Portion of a $QT^T$ Matrix

N	q1•k1	-∞	-∞	-∞	-∞
	q2•k1	q2•k2	-∞	-∞	-∞
	q3•k1	q3•k2	q3•k3	-∞	-∞
	q4•k1	q4•k2	q4•k3	q4•k4	-∞
	q5•k1	q5•k2	q5•k3	q5•k4	q5•k5

# Transformer Blocks

- ▶ A transformer block consists of a single attention layer followed by a feed-forward layer with residual connections and layer normalizations following each.
- ▶ Transformer blocks can be stacked to make deeper and more powerful networks.

# Transformer Blocks



# Transformer Blocks

$$\mathbf{z} = \text{LayerNorm}(\mathbf{x} + \text{SelfAttn}(\mathbf{x})) \quad (12)$$

$$\mathbf{y} = \text{LayerNorm}(\mathbf{z} + \text{FFNN}(\mathbf{z})) \quad (13)$$

$$\mu = \frac{1}{d_h} \sum_{i=1}^{d_h} x_i \quad (14)$$

$$\sigma = \sqrt{\frac{1}{d_h} \sum_{i=1}^{d_h} (x_i - \mu)^2} \quad (15)$$

$$\hat{\mathbf{x}} = \frac{(\mathbf{x} - \mu)}{\sigma} \quad (16)$$

$$\text{LayerNorm} = \gamma \hat{\mathbf{x}} + \beta \quad (17)$$

# Multihead Attention Layers

- ▶ are sets of self-attention layers, each with its own set of key, query, and value matrices:
  - ▶  $\mathbf{W}_i^Q \in \mathbb{R}^{d \times d_k}$
  - ▶  $\mathbf{W}_i^K \in \mathbb{R}^{d \times d_k}$
  - ▶  $\mathbf{W}_i^V \in \mathbb{R}^{d \times d_v}$
- ▶ Each member of such a set of self-attention layers is called a **head**
- ▶ Each head gets multiplied by the inputs packed into  $\mathbf{X}$  to produce
  - ▶  $\mathbf{W}_i^Q \in \mathbb{R}^{N \times d_k}$
  - ▶  $\mathbf{W}_i^K \in \mathbb{R}^{N \times d_k}$
  - ▶  $\mathbf{W}_i^V \in \mathbb{R}^{N \times d_v}$

# Multihead Attention Layers

- ▶ The output of a multi-head layer with  $h$  heads consists of  $h$  vectors of shape  $N \times d_v$
- ▶ These outputs are concatenated from each head and then, using a linear projection with weight matrix  $\mathbf{W}_i^O \in \mathbb{R}^{hd_k \times d}$ , reduced to  $N \times d$  output.

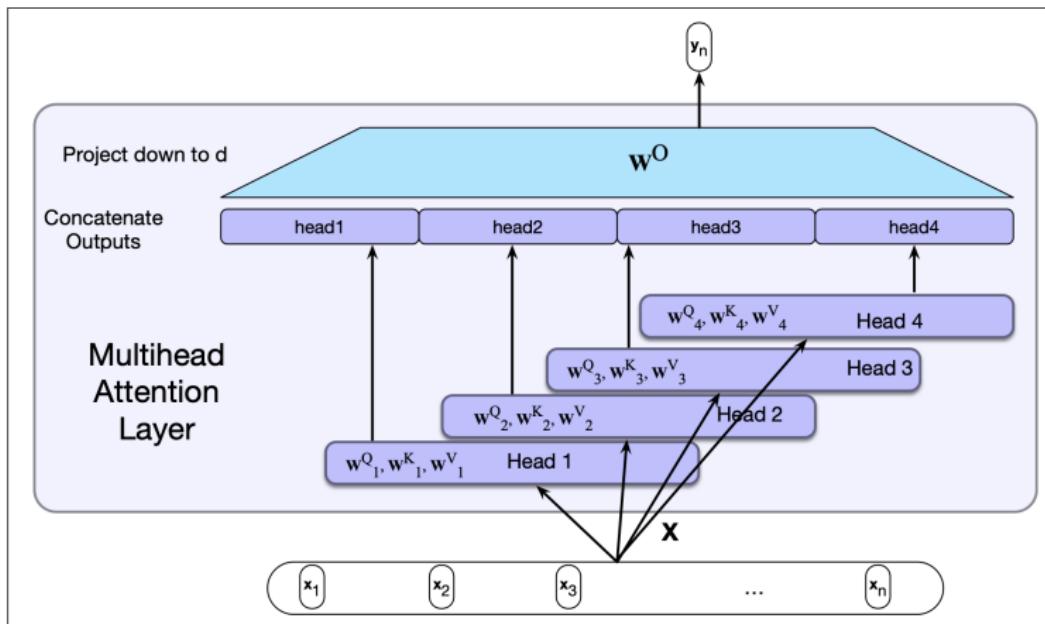
# Multihead Attention

$$\text{MultiHeadAttn}(\mathbf{X}) = (\mathbf{head}_1 \oplus \mathbf{head}_2 \dots \oplus \mathbf{head}_h) \mathbf{W}^O \quad (18)$$

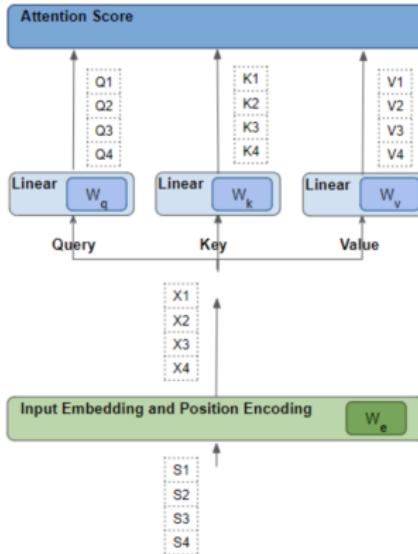
$$\mathbf{Q} = \mathbf{X} \mathbf{W}_i^Q; \quad \mathbf{K} = \mathbf{X} \mathbf{W}_i^K; \quad \mathbf{V} = \mathbf{X} \mathbf{W}_i^V \quad (19)$$

$$\mathbf{head}_i = \text{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \quad (20)$$

# Multihead Attention

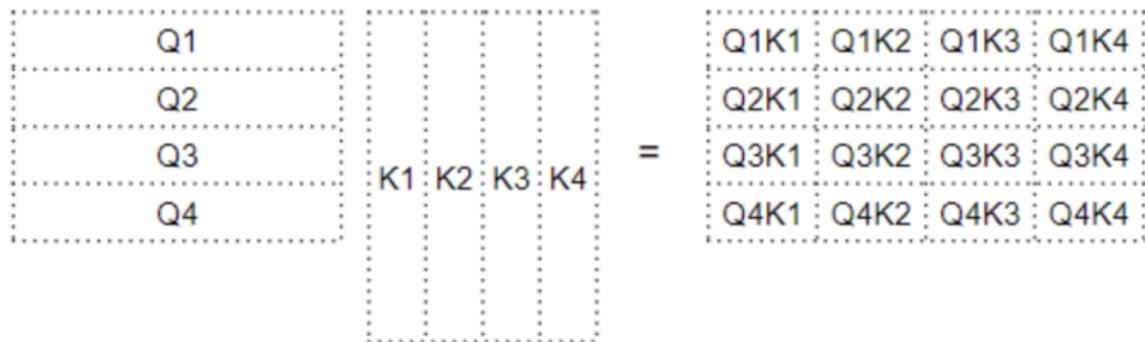


# Multihead Attention: Visual Summary



Slide due to Keitan Doshi, <https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-why-they-work-so-well-d840bd61a9d3>

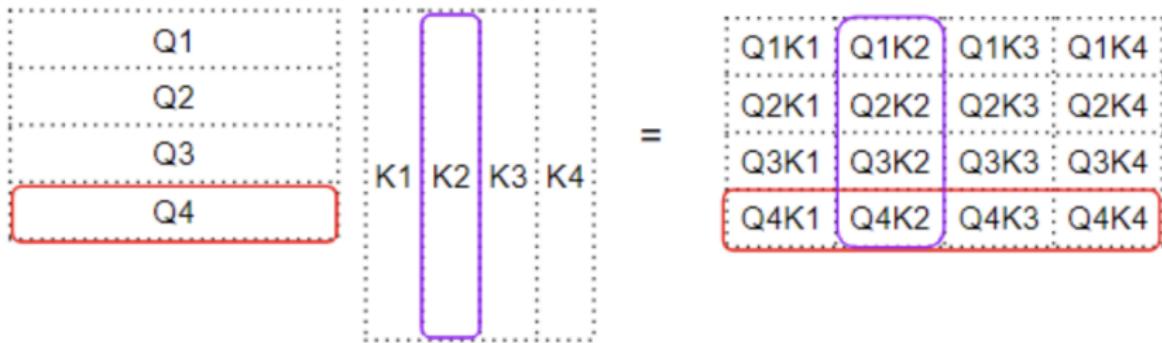
# Multihead Attention: Dot Product between Query and Key matrices



Slide due to Keitan Doshi

<https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-why-they-work-so-well-d840bd61a9d3>

# Multihead Attention: Dot Product between Query and Key matrices



Slide due to Keitan Doshi

<https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-why-they-work-so-well-d840bd61a9d3>

# Dot Product between Query-Key and Value Matrices

$$\begin{array}{|c|c|c|c|} \hline Q1K1 & Q1K2 & Q1K3 & Q1K4 \\ \hline Q2K1 & Q2K2 & Q2K3 & Q2K4 \\ \hline Q3K1 & Q3K2 & Q3K3 & Q3K4 \\ \hline Q4K1 & Q4K2 & Q4K3 & Q4K4 \\ \hline \end{array} \times \begin{array}{|c|} \hline V1 \\ \hline V2 \\ \hline V3 \\ \hline V4 \\ \hline \end{array} = \begin{array}{|c|} \hline Q1K1V1 + Q1K2V2 + Q1K3V3 + Q1K4V4 \\ \hline Q2K1V1 + Q2K2V2 + Q2K3V3 + Q2K4V4 \\ \hline Q3K1V1 + Q3K2V2 + Q3K3V3 + Q3K4V4 \\ \hline Q4K1V1 + Q4K2V2 + Q4K3V3 + Q4K4V4 \\ \hline \end{array}$$
$$= \begin{array}{|c|} \hline Z1 \\ \hline Z2 \\ \hline Z3 \\ \hline Z4 \\ \hline \end{array}$$

Slide due to Keitan Doshi

<https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-why-they-work-so-well-d840bd61a9d3>

# Attention Score for the word blue pays attention to every other word

$$Z_4 = (Q_4 K_1) V_1 + (Q_4 K_2) V_2 + (Q_4 K_3) V_3 + (Q_4 K_4) V_4$$

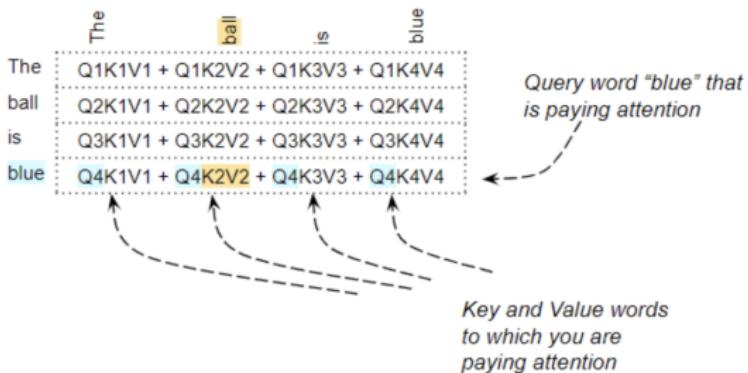
The diagram illustrates the calculation of  $Z_4$  as a weighted sum of vectors  $V_1, V_2, V_3, V_4$ . The equation is  $Z_4 = (Q_4 K_1) V_1 + (Q_4 K_2) V_2 + (Q_4 K_3) V_3 + (Q_4 K_4) V_4$ . Three dashed arrows point from labels above the equation to specific terms in the sum:

- A dashed arrow points down to the term  $(Q_4 K_1) V_1$ , labeled "Fourth word Score".
- A dashed arrow points down to the term  $(Q_4 K_2) V_2$ , labeled "Fourth Query word \* first Key word".
- A dashed arrow points up to the term  $(Q_4 K_3) V_3$ , labeled "Fourth Query word \* second Key word".

Slide due to Keitan Doshi

<https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-why-they-work-so-well-d840bd61a9d3>

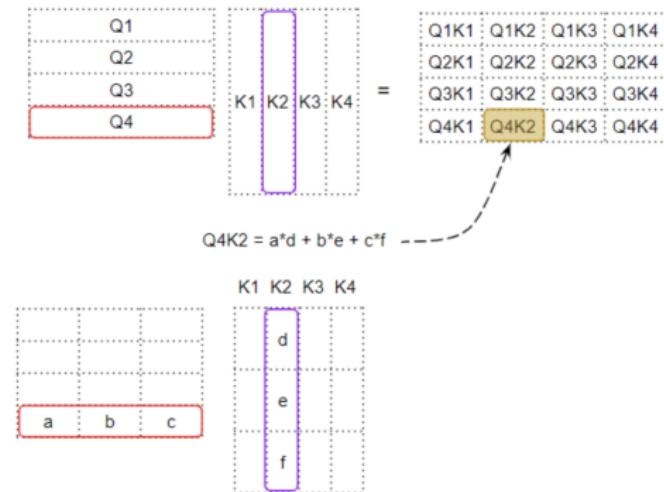
# Dot Product between Query-Key and Value Matrices



Slide due to Keitan Doshi

<https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-why-they-work-so-well-d840bd61a9d3>

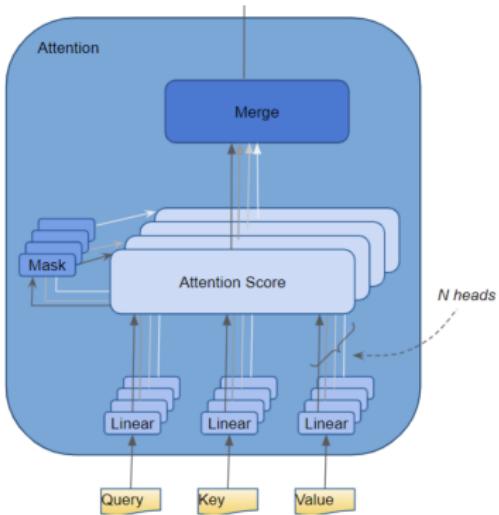
# Each cell is a dot product between two word vectors



Slide due to Keitan Doshi

<https://towardsdatascience.com/transformers-explained-visually-not-just-how-but->

# Multihead Attention: Visual Summary



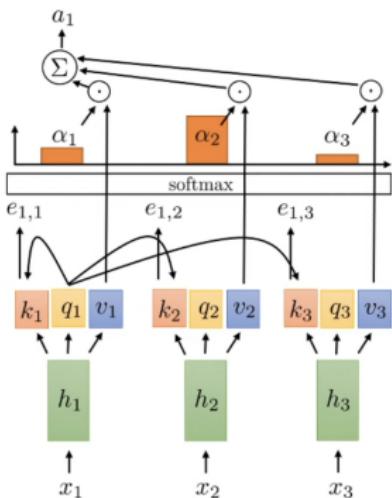
Slide due to Keitan Doshi

<https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-why-they-work-so-well-d840bd61a9d3>

## Modeling word order: positional embedding

- ▶ Transformers models do not come with built-in information about sequence order such as time-step information in an RNN
- ▶ Transformers models do not have any notion of relative or absolute positions of the tokens in a sequence.
- ▶ In order to model word order, positional embeddings can be combined with input embeddings.
- ▶ Positional embeddings that are specific to each position in an input sequence.

# Positional encoding: what is the order?



**what we see:**

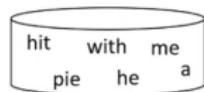
he hit me with a pie

**what naïve self-attention sees:**

a pie hit me with he

a hit with me he pie

he pie me with a hit



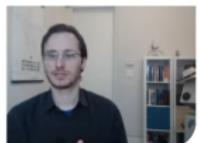
most alternative orderings are nonsense, but some change the meaning

**in general** the position of words in a sentence carries information!

**Idea:** add some information to the representation at the beginning that indicates where it is in the sequence!

$$h_t = f(x_t, t)$$

some function



Slide due to RAIL CS182: Lecture 12:Part 2: Transformers  
[https://www.youtube.com/watch?v=4AzsiCMw\\_-s](https://www.youtube.com/watch?v=4AzsiCMw_-s)

# Positional encoding: sin/cos

Naïve positional encoding: just append  $t$  to the input

$$\bar{x}_t = \begin{bmatrix} x_t \\ t \end{bmatrix}$$

This is not a great idea, because **absolute** position is less important than **relative** position

I walk my dog every day



every single day I walk my dog



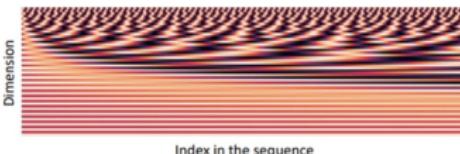
The fact that "my dog" is right after "I walk" is the important part, not its absolute position

we want to represent **position** in a way that tokens with similar **relative** position have similar **positional encoding**

Idea: what if we use **frequency-based** representations?

$$p_t = \begin{bmatrix} \sin(t/10000^{2*1/d}) \\ \cos(t/10000^{2*1/d}) \\ \sin(t/10000^{2*2/d}) \\ \cos(t/10000^{2*2/d}) \\ \dots \\ \sin(t/10000^{2*\frac{d}{2}/d}) \\ \cos(t/10000^{2*\frac{d}{2}/d}) \end{bmatrix}$$

dimensionality of positional encoding

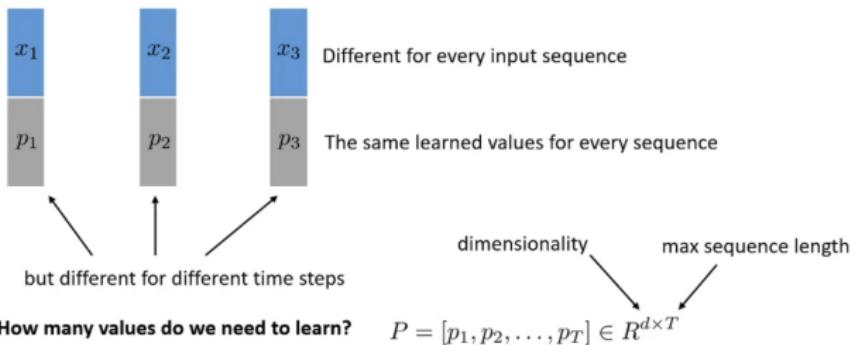


Slide due to RAIL CS182: Lecture 12:Part 2: Transformers

[https://www.youtube.com/watch?v=4AzsiCMw\\_-s](https://www.youtube.com/watch?v=4AzsiCMw_-s)

# Positional encoding: learned

Another idea: just learn a positional encoding



+ more flexible (and perhaps more optimal) than sin/cos encoding

+ a bit more complex, need to pick a max sequence length (and can't generalize beyond it)



Slide due to RAIL CS182: Lecture 12:Part 2: Transformers  
[https://www.youtube.com/watch?v=4AzsiCMw\\_-s](https://www.youtube.com/watch?v=4AzsiCMw_-s)

# How to incorporate positional encoding?

At each step, we have  $x_t$  and  $p_t$

**Simple choice:** just concatenate them

$$\bar{x}_t = \begin{bmatrix} x_t \\ p_t \end{bmatrix}$$

**More often:** just add after **embedding** the input

input to self-attention is  $\text{emb}(x_t) + p_t$

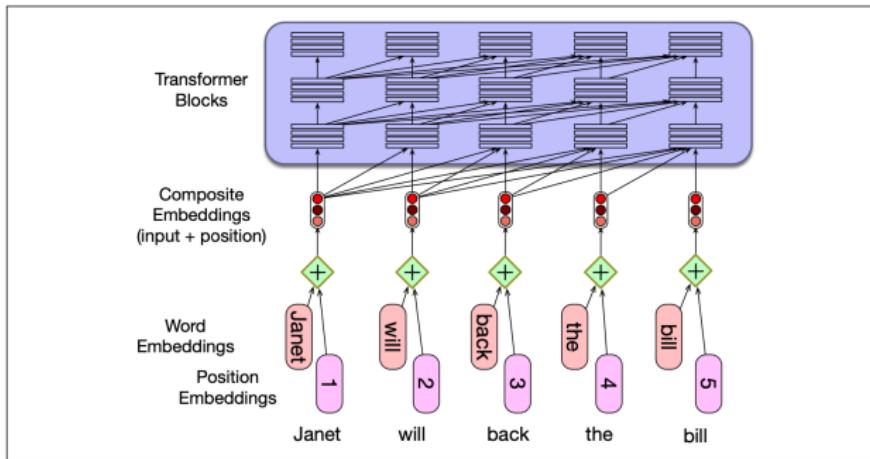


some learned function (e.g., some fully connected layers with linear layers + nonlinearities)

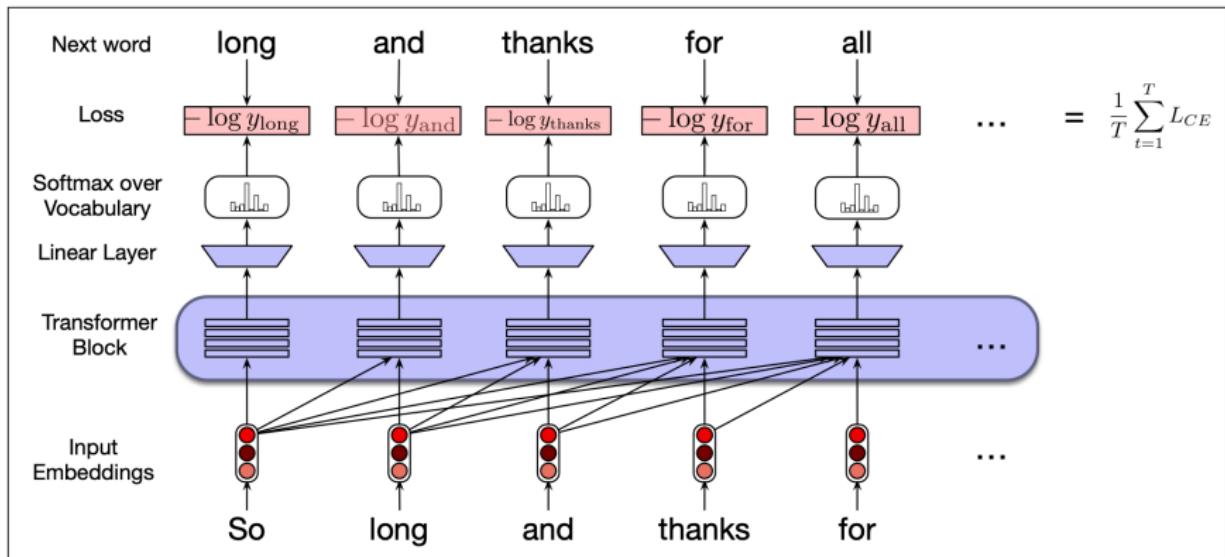


Slide due to RAIL CS182: Lecture 12:Part 2: Transformers  
[https://www.youtube.com/watch?v=4AzsiCMw\\_-s](https://www.youtube.com/watch?v=4AzsiCMw_-s)

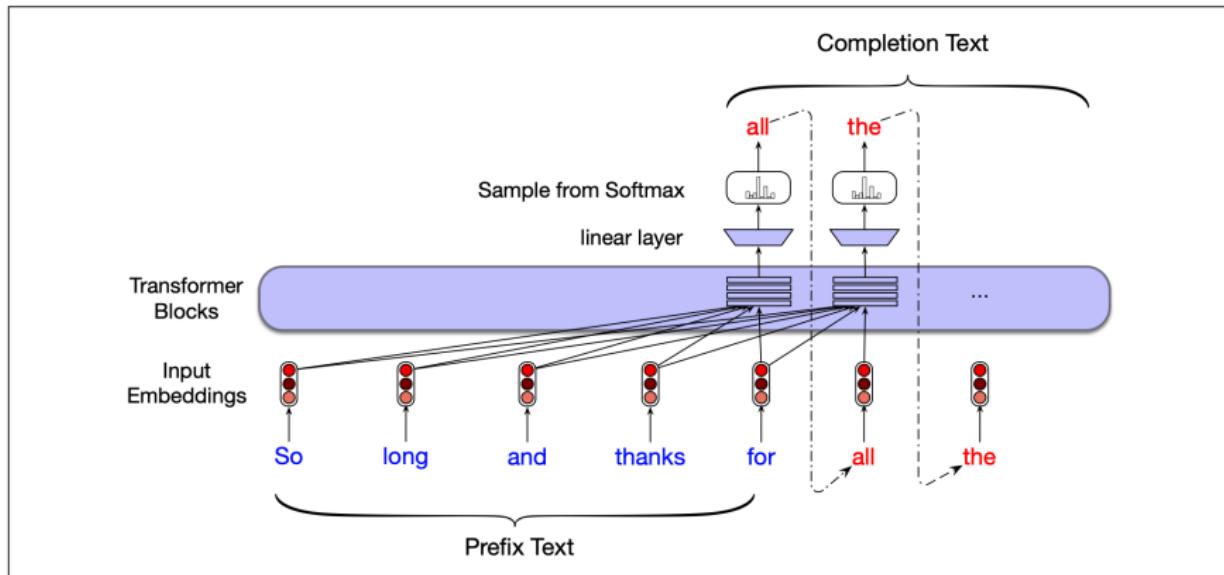
# Multihead Attention



# Transformers as Language Models



# Contextual Generation and Summarization



# Contextual Generation and Summarization

## Original Article

The only thing crazier than a guy in snowbound Massachusetts boxing up the powdery white stuff and offering it for sale online? People are actually buying it. For \$89, self-styled entrepreneur Kyle Waring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box – enough for 10 to 15 snowballs, he says. But not if you live in New England or surrounding states. “We will not ship snow to any states in the northeast!” says Waring’s website, [ShipSnowYo.com](http://ShipSnowYo.com). “We’re in the business of expunging snow!”

## **Contextual Generation and Summarization**

Continued

His website and social media accounts claim to have filled more than 133 orders for snow – more than 30 on Tuesday alone, his busiest day yet. With more than 45 total inches, Boston has set a record this winter for the snowiest month in its history. Most residents see the huge piles of snow choking their yards and sidewalks as a nuisance, but Waring saw an opportunity.

## Contextual Generation and Summarization

Continued

According to Boston.com, it all started a few weeks ago, when Waring and his wife were shoveling deep snow from their yard in Manchester-by-the-Sea, a coastal suburb north of Boston. He joked about shipping the stuff to friends and family in warmer states, and an idea was born. His business slogan: "Our nightmare is your dream!" At first, ShipSnowYo sold snow packed into empty 16.9-ounce water bottles for \$19.99, but the snow usually melted before it reached its destination...

# **Contextual Generation and Summarization**

## Summary

Kyle Waring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box – enough for 10 to 15 snowballs, he says. But not if you live in New England or surrounding states.

# Contextual Generation and Summarization

