Vector Semantics & Embeddings

Word Meaning

What do words mean?

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

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

Introductory logic classes:

- The meaning of "dog" is DOG; cat is CAT

$$\forall x \, \mathrm{DOG}(x) \longrightarrow \mathrm{MAMMAL}(x)$$

Old linguistics joke by Barbara Partee in 1967:

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

That seems hardly better!

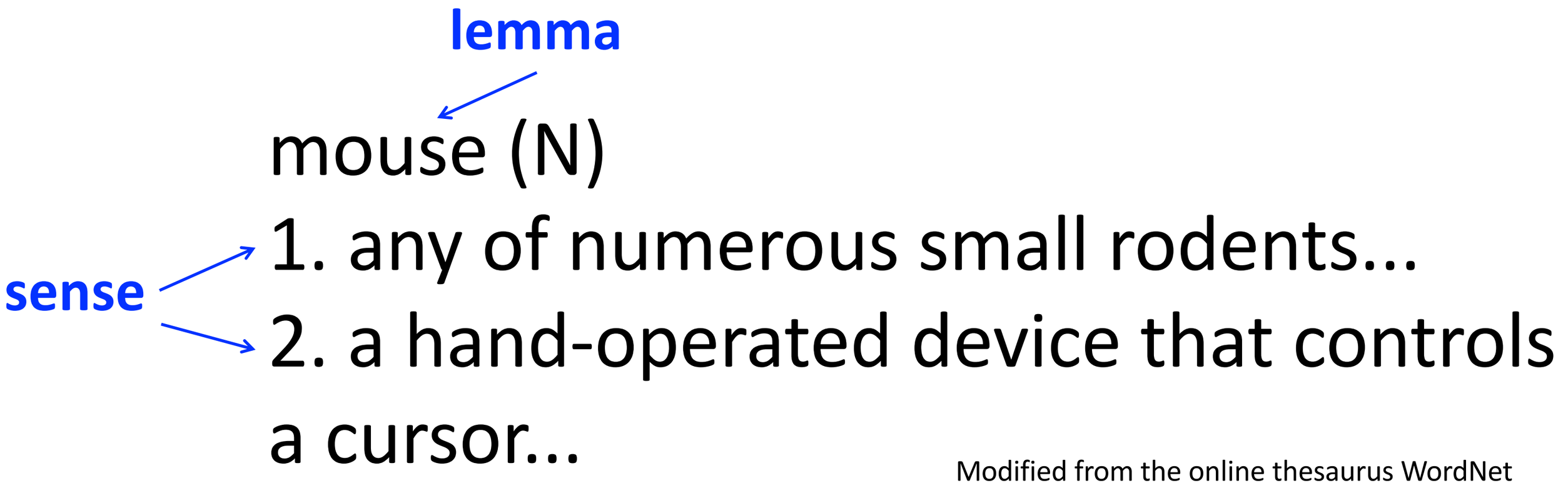
Desiderata

What should a theory of word meaning do for us?

Let's look at some desiderata

From lexical semantics, the linguistic study of word meaning

Lemmas and senses



Modified from the online thesaurus WordNet

A sense or "concept" is the meaning component of a word

Lemmas can be polysemous (have multiple senses)

Relations between senses: Synonymy

Synonyms have the same meaning in some or all contexts.

- filbert / hazelnut
- couch / sofa
- big / large
- automobile / car
- vomit / throw up
- water / H_20

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_20

"H_20" in a surfing guide?

big/large

my big sister != my large sister

The Linguistic Principle of Contrast

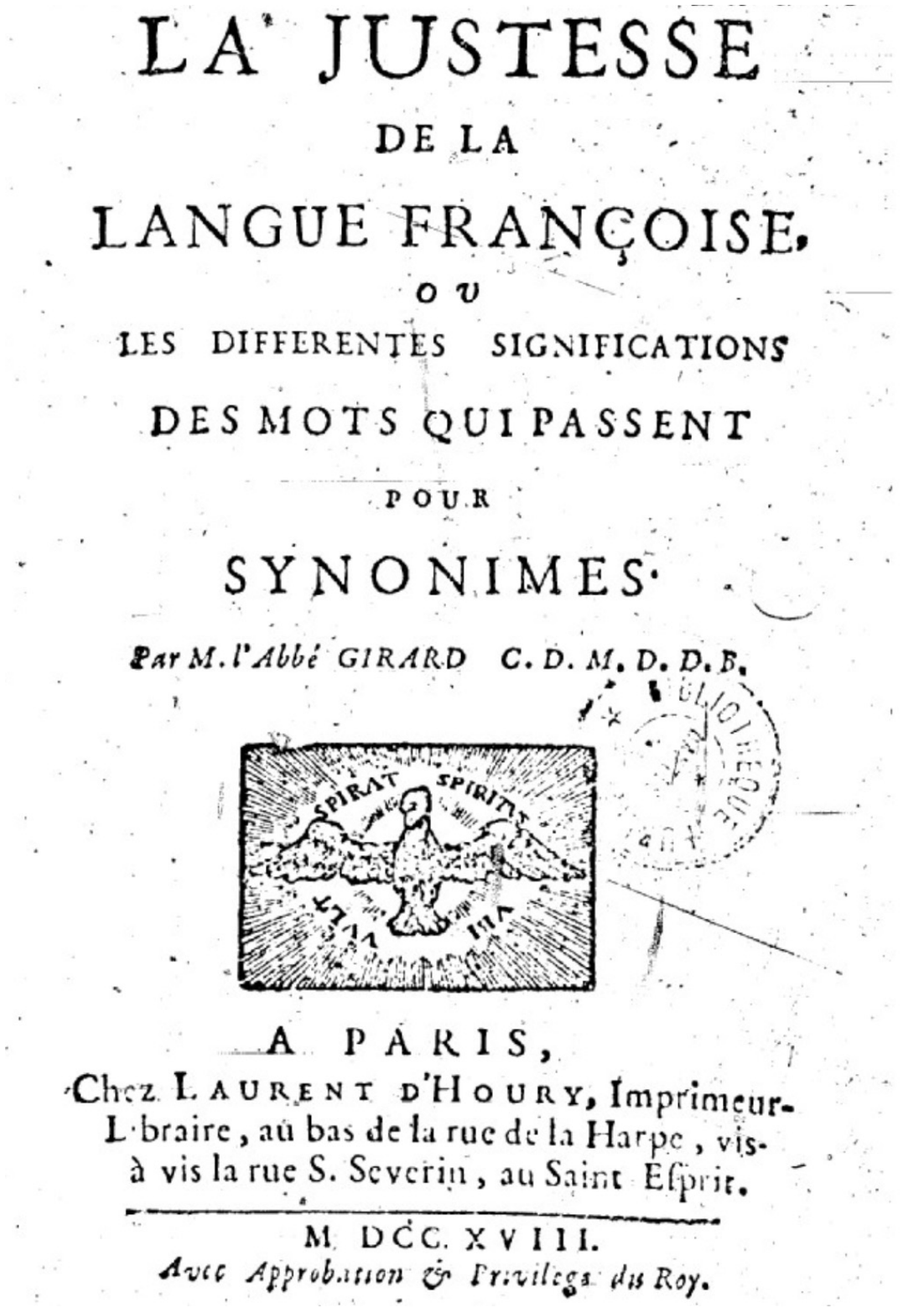
Difference in form → difference in meaning

Abbé Gabriel Girard 1718

Re: "exact" synonyms

"je ne crois pas qu'il y ait de mot ſynonime 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 & Privilege du Roy.

Thanks to Mark Aronoff!

Relation: Similarity

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

```
car, bicycle
cow, horse
```

Ask humans how similar 2 words are

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

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

Relation: Word relatedness

Also called "word association"

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

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

Semantic field

Words that

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

hospitals

surgeon, scalpel, nurse, anaesthetic, hospital

restaurants

waiter, menu, plate, food, menu, chef

houses

door, roof, kitchen, family, bed

Relation: Antonymy

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

Otherwise, they are very similar!

`dark/light` `short/long` `fast/slow` `rise/fall`
`hot/cold` `up/down` `in/out`

More formally: antonyms can

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

Connotation (sentiment)

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

Connotation

Osgood et al. (1957)

Words seem to vary along 3 affective dimensions:

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

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

Values from NRC VAD Lexicon (Mohammad 2018)

So far

Concepts or word senses

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

Have relations with each other

- Synonymy
- Antonymy
- Similarity
- Relatedness
- Connotation

Vector Semantics & Embeddings

Word Meaning

Vector Semantics & Embeddings

Vector Semantics

Computational models of word meaning

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

We'll introduce **vector semantics**

- The standard model in language processing!
- Handles many of our goals!

Ludwig Wittgenstein

PI #43:

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

Let's define words by their usages

One way to define "usage":

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

Zellig Harris (1954):

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

What does recent English borrowing *ongchoi* mean?

Suppose you see these sentences:

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

And you've also seen these:

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

Conclusion:

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

Ongchoi: *Ipomoea aquatica "Water Spinach"*

空心菜

kangkong

rau muống

…



Yamaguchi, Wikimedia Commons, public domain

Idea 1: Defining meaning by linguistic distribution

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

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

3 affective dimensions for a word

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

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

NRC VAD Lexicon
(Mohammad 2018)

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

Idea 1: Defining meaning by linguistic distribution

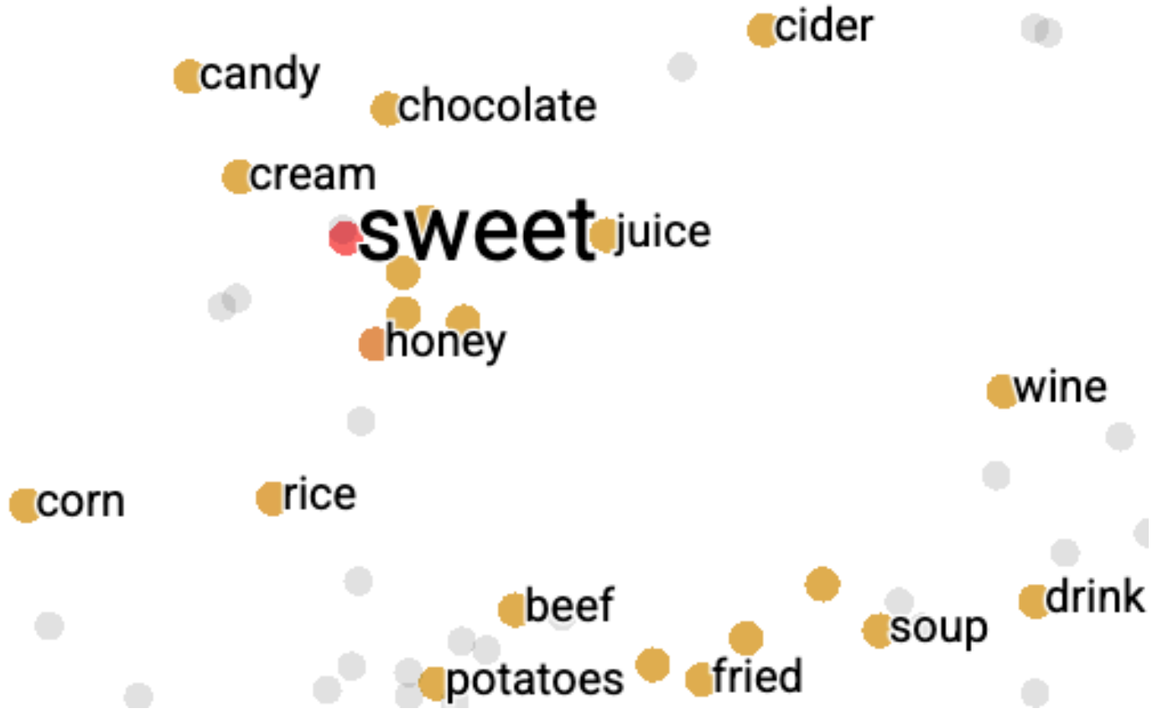
Idea 2: Meaning as a point in multidimensional space

Defining meaning as a point in space based on distribution

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

Similar words are "**nearby in semantic space**"

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



We define meaning of a word as a vector

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

The standard way to represent meaning in NLP

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

Fine-grained model of meaning for similarity

Intuition: why vectors?

Consider sentiment analysis:

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

We'll discuss 2 kinds of embeddings

Simple count embeddings

- **Sparse** vectors
- Words are represented by the **counts** of nearby words
- Later for information retrieval we'll augment this model to create the **tf-idf** model of weighting those counts.

Word2vec

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

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

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

庄子(Zhuangzi), Chapter 26

Vector Semantics & Embeddings

Vector Semantics

Vector Semantics & Embeddings

Count-based embeddings

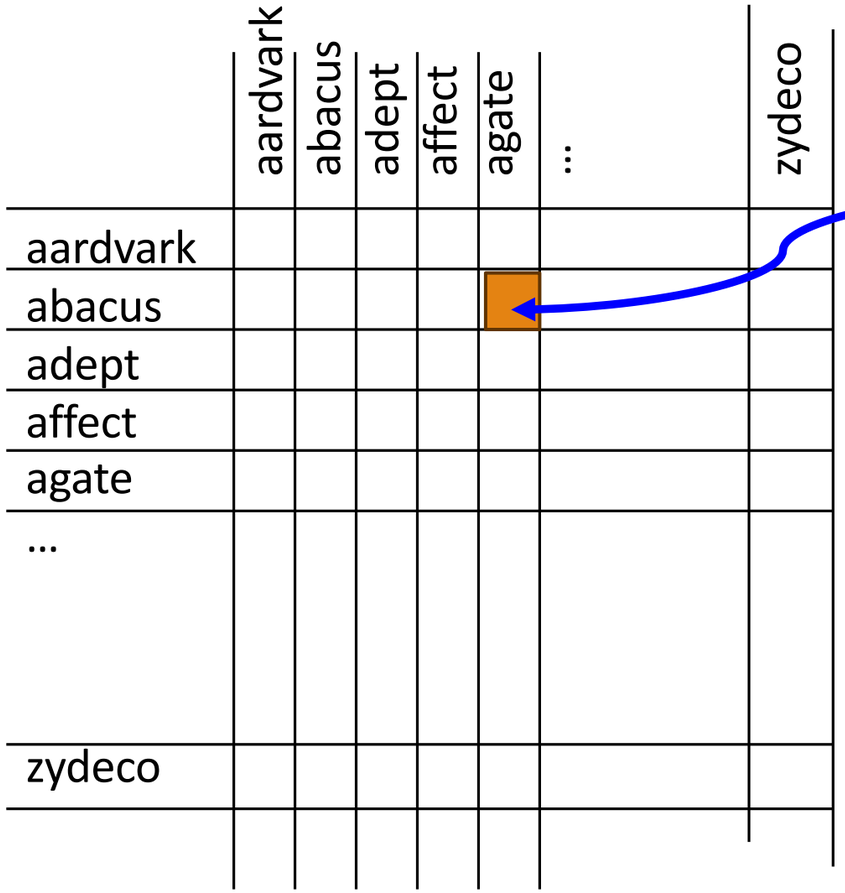
Remember the intuition: words with similar neighborhoods have similar meanings

How to measure a word's neighborhood?

Word-context matrix (a kind of co-occurrence matrix)

- each row represents a word in the vocabulary
- each column represents how often each other word in the vocabulary appears nearby

Word-Context Matrix



How often does **agate** occur near **abacus**?

Word-Context Matrix

What does "nearby" mean?

For right now let's say "within 4 words"

The word-context matrix
One set of 4-word contexts

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

Let's consider a mini-matrix of 3 words. How often do "a", "computer", and "pie occur in the context of "cherry"?

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

The word-context mini-matrix for just 4 words and 3 contexts

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

	a	**computer**	**pie**
cherry	1	0	1
strawberry	0	0	2
digital	0	1	0
information	1	1	0

The word-context mini-matrix for just 4 words and 3 contexts

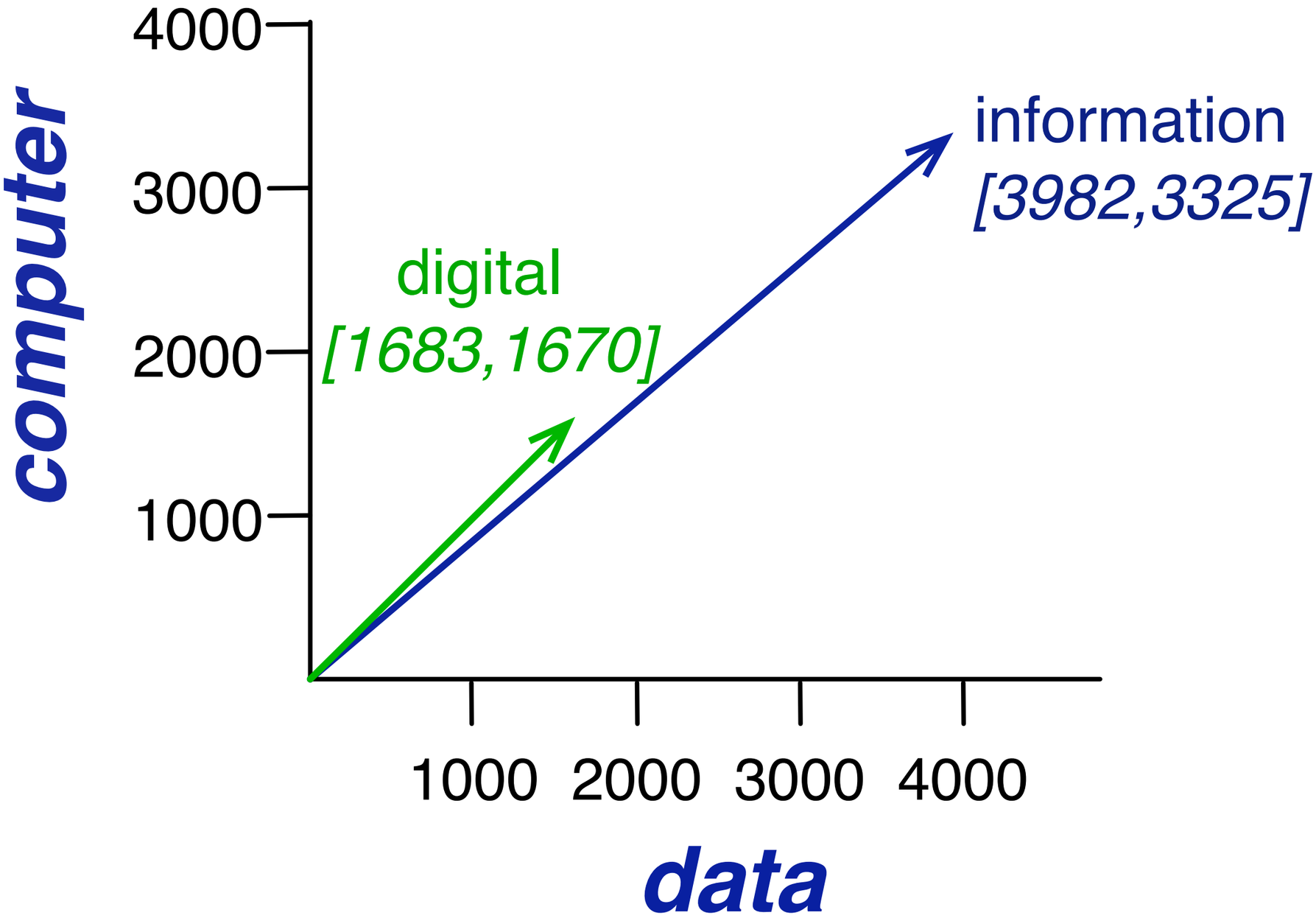
	a	computer	pie
cherry	1	0	1
strawberry	0	0	2
digital	0	1	0
information	1	1	0

- This 4x3 matrix is a subset of full |V| x |V| matrix
- Each word is represented by a row vector with dimensionality [1 x |V|]
- With co-occurrence counts with each other word

A selection from a larger word-context matrix

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	...


4000
3000
2000
1000
computer
information
[3982,3325]
digital
[1683,1670]
1000
2000
3000
4000
data

The word-context matrix

Word context matrix is |V| x |V|

This could be 50,000 x 50,000
Most of these numbers are zero!

So these are sparse vectors

There are efficient algorithms for storing and computing with sparse matrices

Vector Semantics & Embeddings

Count-based 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:

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

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

Dot product can thus be a useful similarity metric between vectors

Problem with raw dot-product

Dot product favors long vectors

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

Vector length:

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

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

So dot product overly favors frequent words

Alternative: cosine for computing word similarity

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

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

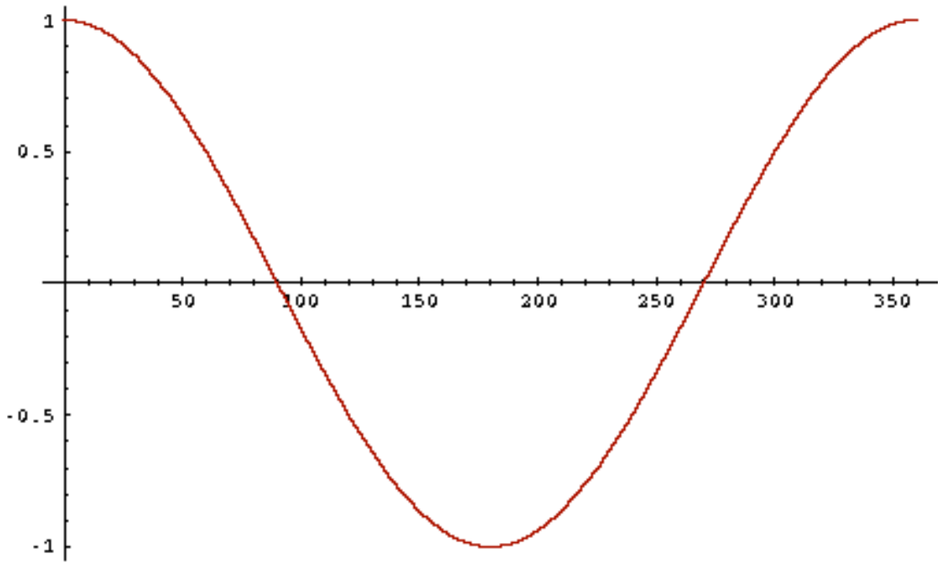
$$\mathbf{a}\cdot\mathbf{b} = |\mathbf{a}||\mathbf{b}|\cos\theta$$
$$\frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{a}||\mathbf{b}|} = \cos\theta$$

Cosine as a similarity metric

-1: vectors point in opposite directions

+1: vectors point in same directions

0: vectors are orthogonal



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

Cosine examples

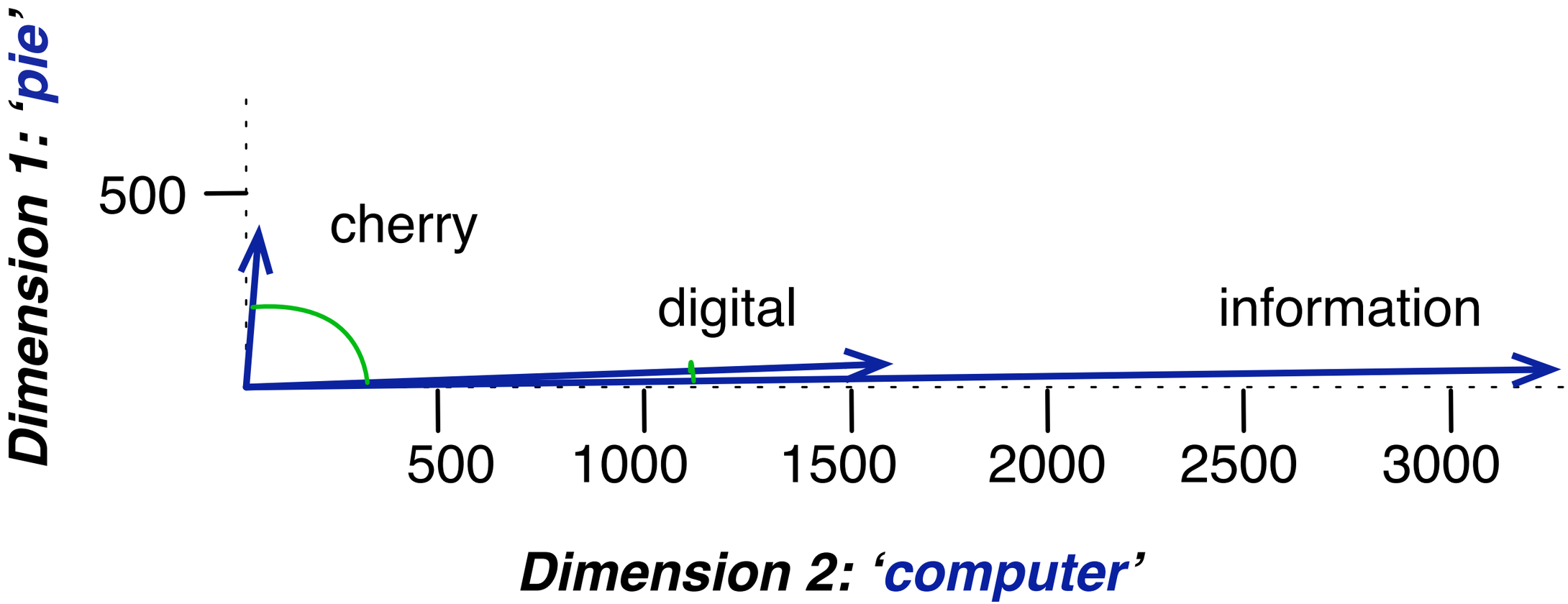
$$\cos(\vec{v},\vec{w}) = \frac{\vec{v}\bullet\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\vec{v}}{|\vec{v}|}\bullet\frac{\vec{w}}{|\vec{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

	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)



Vector Semantics & Embeddings

Cosine for computing word similarity

Vector Semantics & Embeddings

Word2vec

Sparse versus dense vectors

Count vectors (even if weighted by tf-idf)

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

Alternative: learn vectors which are

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

Sparse versus dense vectors

Why dense vectors?

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

Common methods for getting short dense vectors

“Neural Language Model”-inspired models

- Word2vec (skipgram, CBOW), GloVe

Singular Value Decomposition (SVD)

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

Alternative to these "static embeddings":

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

Simple static embeddings you can download!

Word2vec (Mikolov et al)

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

GloVe (Pennington, Socher, Manning)

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

Word2vec

Popular embedding method

- Very fast to train
- Code available on the web

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

Word2vec provides various options. We'll do:

skip-gram with negative sampling (SGNS)

Word2vec

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

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

We don't actually care about this task

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

Big idea: **self-supervision**:

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

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

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

Skip-Gram Training Data

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

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

c1 c2 [target] c3 c4

Skip-Gram Classifier

(assuming a +/- 2 word window)

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

Goal: train a classifier that is given a candidate (**w**ord, **c**ontext) pair
(apricot, jam)
(apricot, aardvark)
…
And assigns each pair a probability:
$P(+|w, c)$
$P(-|w, c) = 1 - P(+|w, c)$

Similarity is computed from dot product

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

- Cosine is just a normalized dot product

So:

- 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

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

To turn this into a probability

We'll use the sigmoid from logistic regression:

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

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

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

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

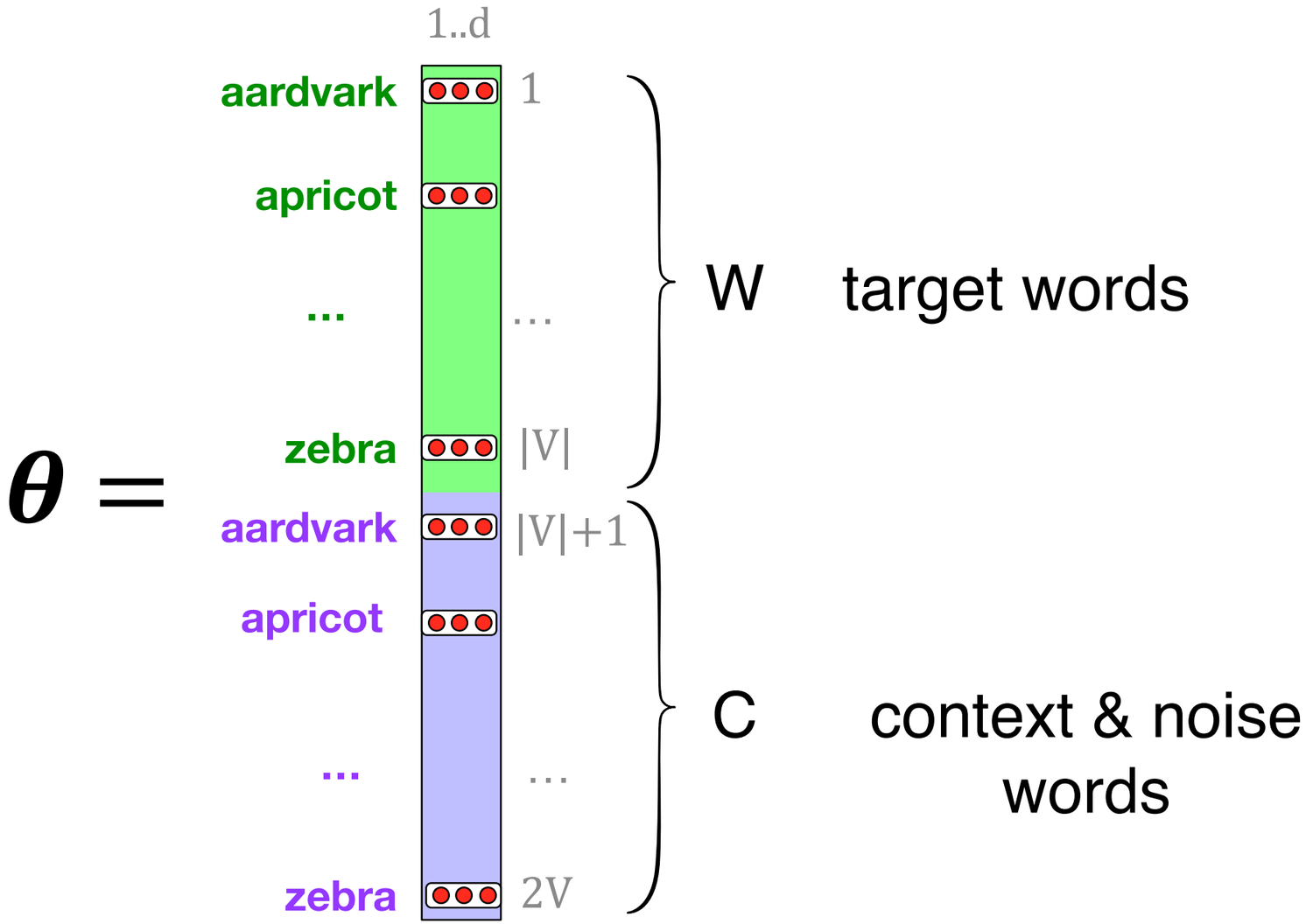
A probabilistic classifier, given

- 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



Vector Semantics & Embeddings

Word2vec

Vector Semantics & Embeddings

Word2vec: Learning the embeddings

Skip-Gram Training data

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

c1 c2 [target] c3 c4



positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

Skip-Gram Training data

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

c1 c2 [target] c3 c4



positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

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

Skip-Gram Training data

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

c1 c2 [target] c3 c4



positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -

t	c	t	c
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

Word2vec: how to learn vectors

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

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

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

Loss function for one *w with* c_{pos} , c_{neg1} …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.

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

Learning the classifier

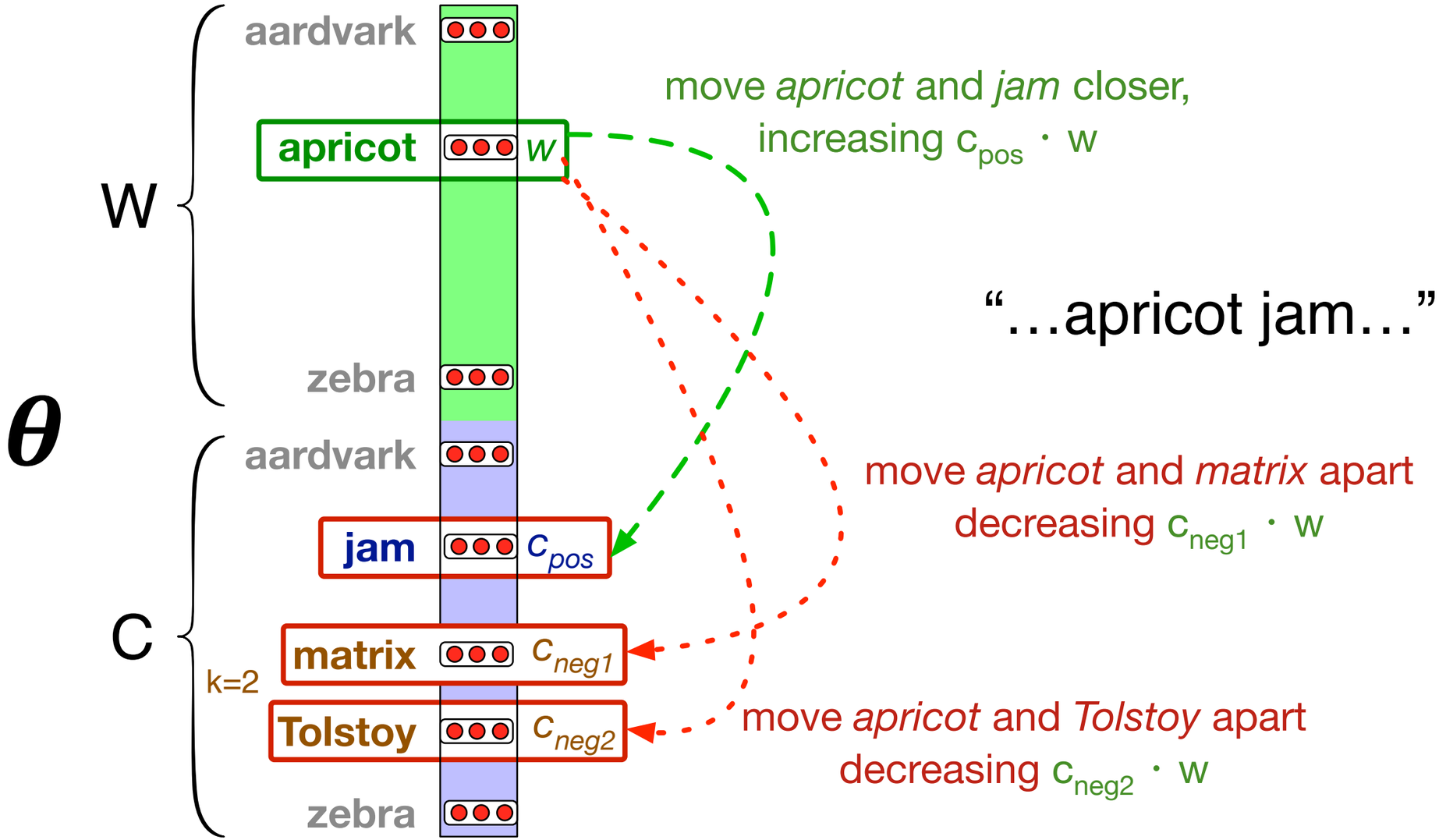
How to learn?

- Stochastic gradient descent!

We'll adjust the word weights to

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

Intuition of one step of gradient descent



Reminder: gradient descent

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

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

The derivatives of the loss function

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

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

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

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

Update equation in SGD

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

$$
\begin{aligned}
c_{pos}^{t+1} &= c_{pos}^{t} - \eta[\sigma(c_{pos}^{t} \cdot w^{t}) - 1]w^{t} \\
c_{neg}^{t+1} &= c_{neg}^{t} - \eta[\sigma(c_{neg}^{t} \cdot w^{t})]w^{t} \\
w^{t+1} &= w^{t} - \eta\left[[\sigma(c_{pos} \cdot w^{t}) - 1]c_{pos} + \sum_{i=1}^{k}[\sigma(c_{\mathrm{neg}_i} \cdot w^{t})]c_{neg_i}\right]
\end{aligned}
$$

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 $\mathbf{w}_i + \mathbf{c}_i$

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

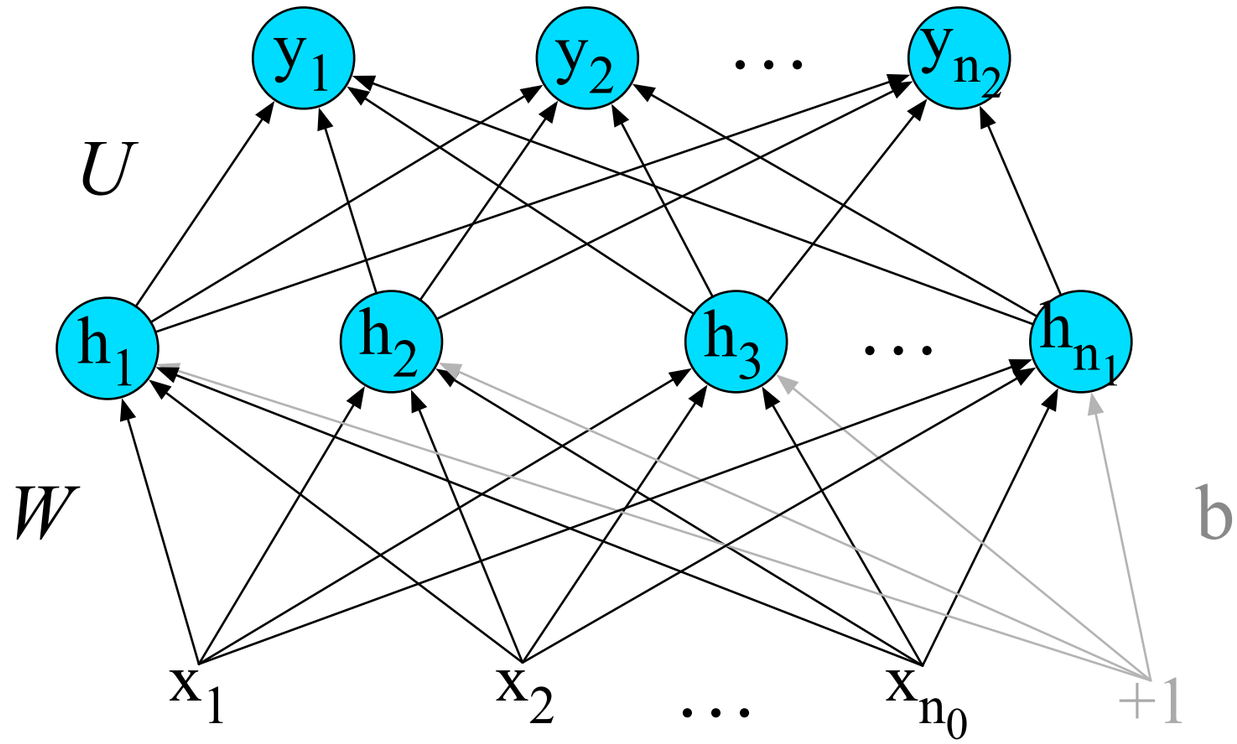
Word2vec: Learning the embeddings

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



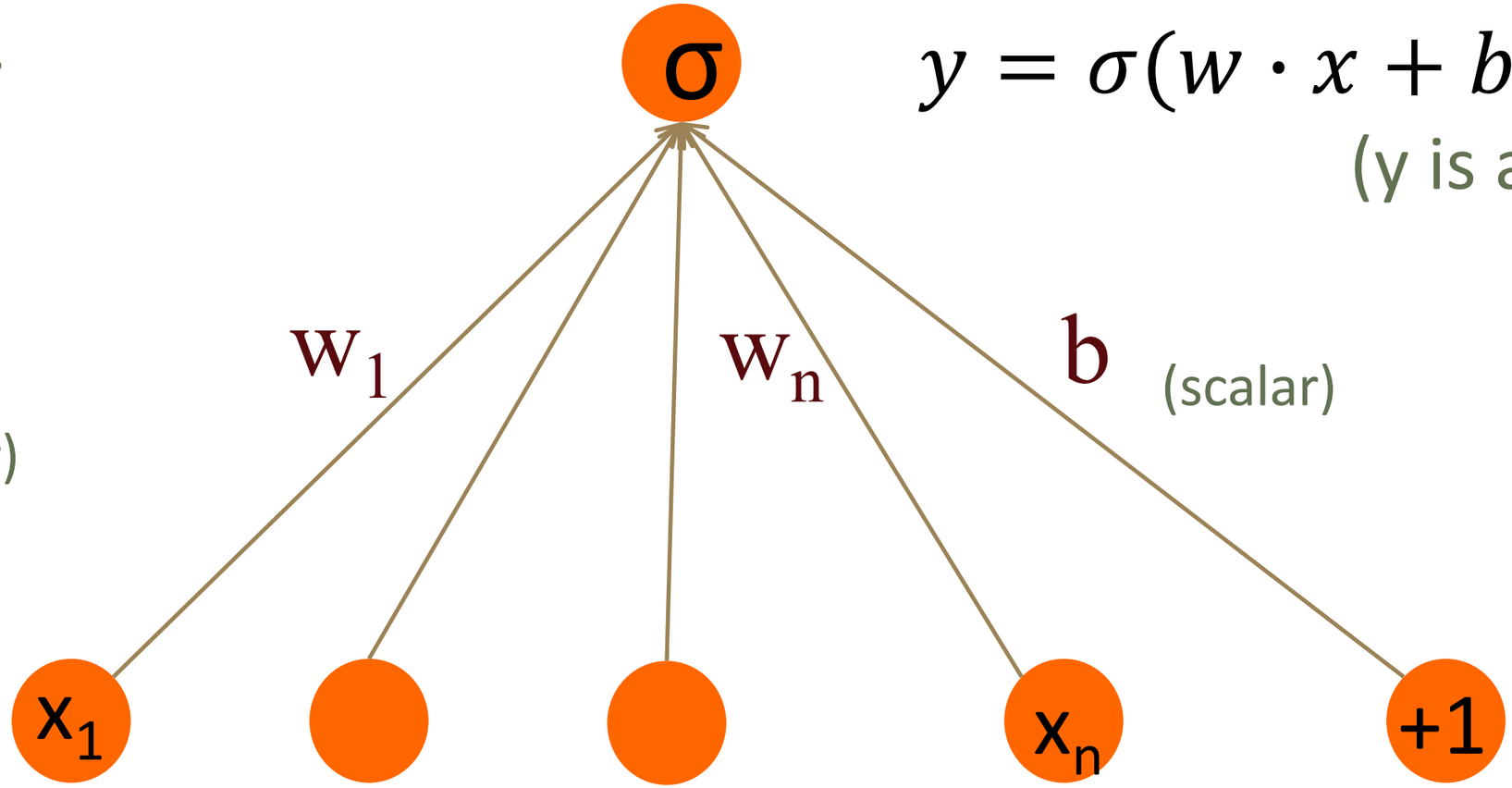
Binary Logistic Regression as a 1-layer Network

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

Output layer
(σ node)

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

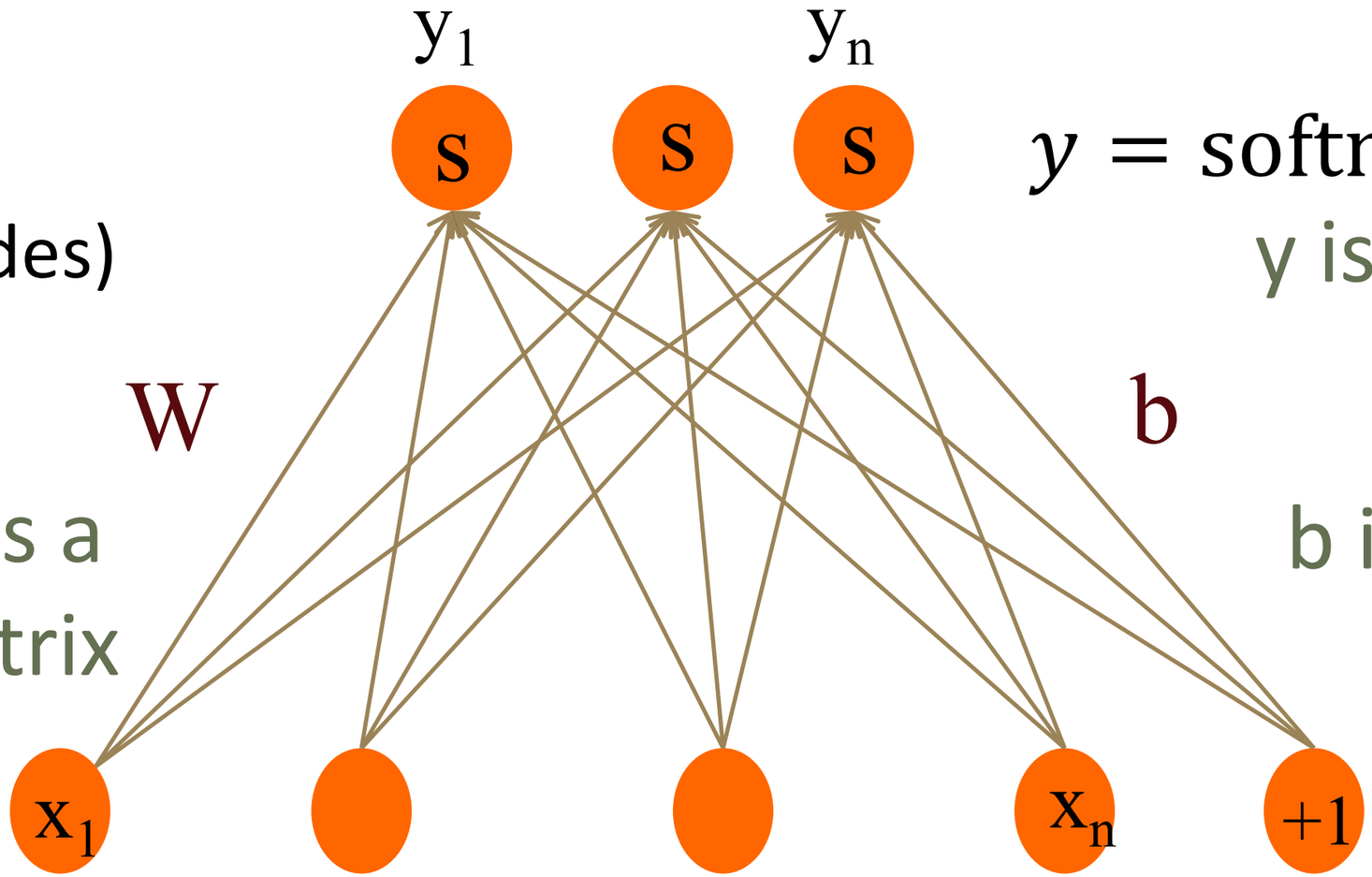
(y is a scalar)



Input layer
vector x

Multinomial Logistic Regression as a 1-layer Network

Fully connected single layer network



Output layer
(softmax nodes)

$$y = \text{softmax}(Wx + b)$$

y is a vector

W is a matrix

b is a vector

Input layer
scalars

Reminder: softmax: a generalization of sigmoid

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

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

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

Example:

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

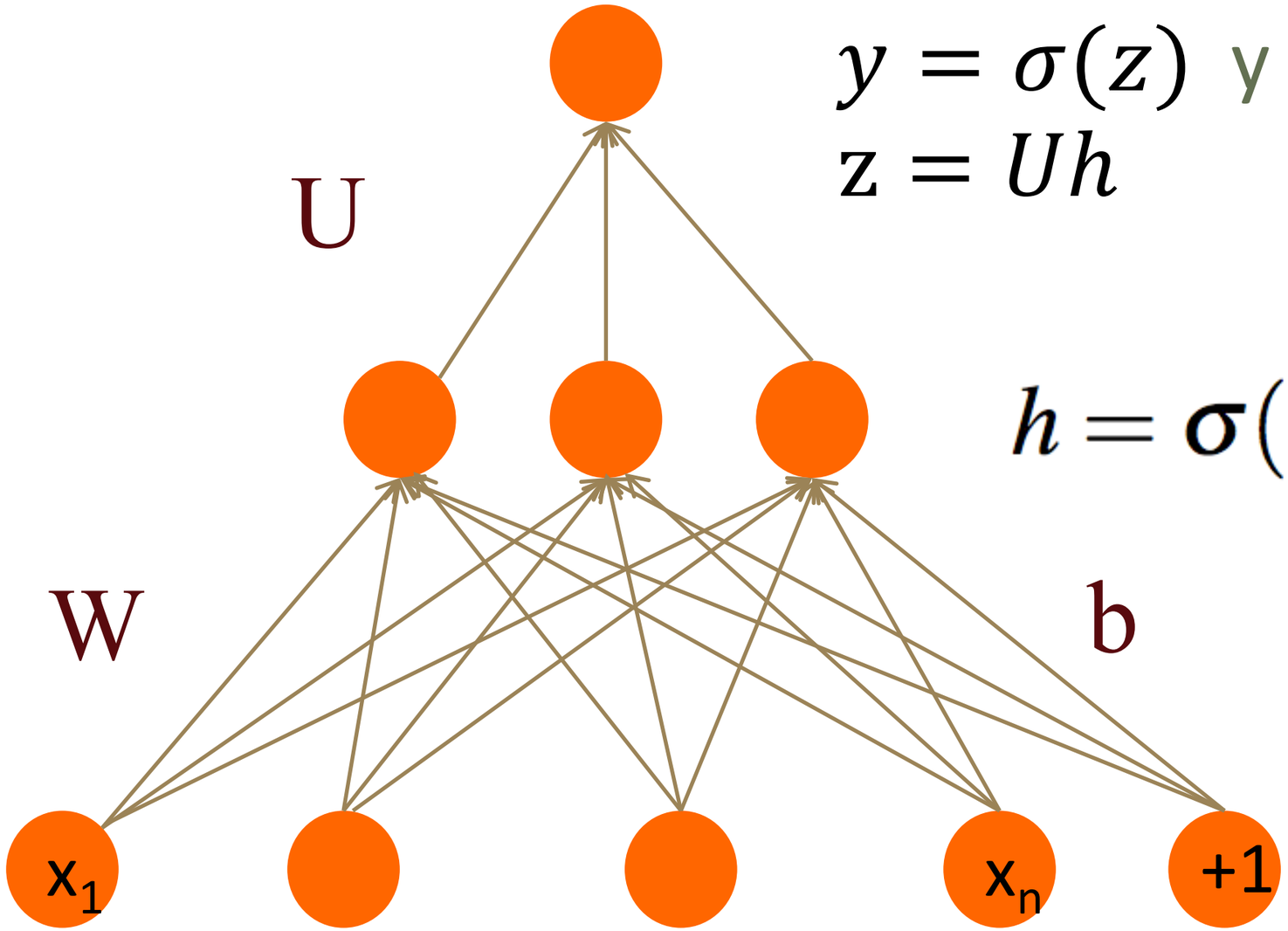
$$\text{softmax}(z) = [0.055, 0.090, 0.006, 0.099, 0.74, 0.010]$$

Two-Layer Network with scalar output

Output layer
(σ node)

hidden units
(σ node)

Input layer
(vector)



$y = \sigma(z)$ y is a scalar

$z = Uh$

$h = \sigma(Wx + b)$

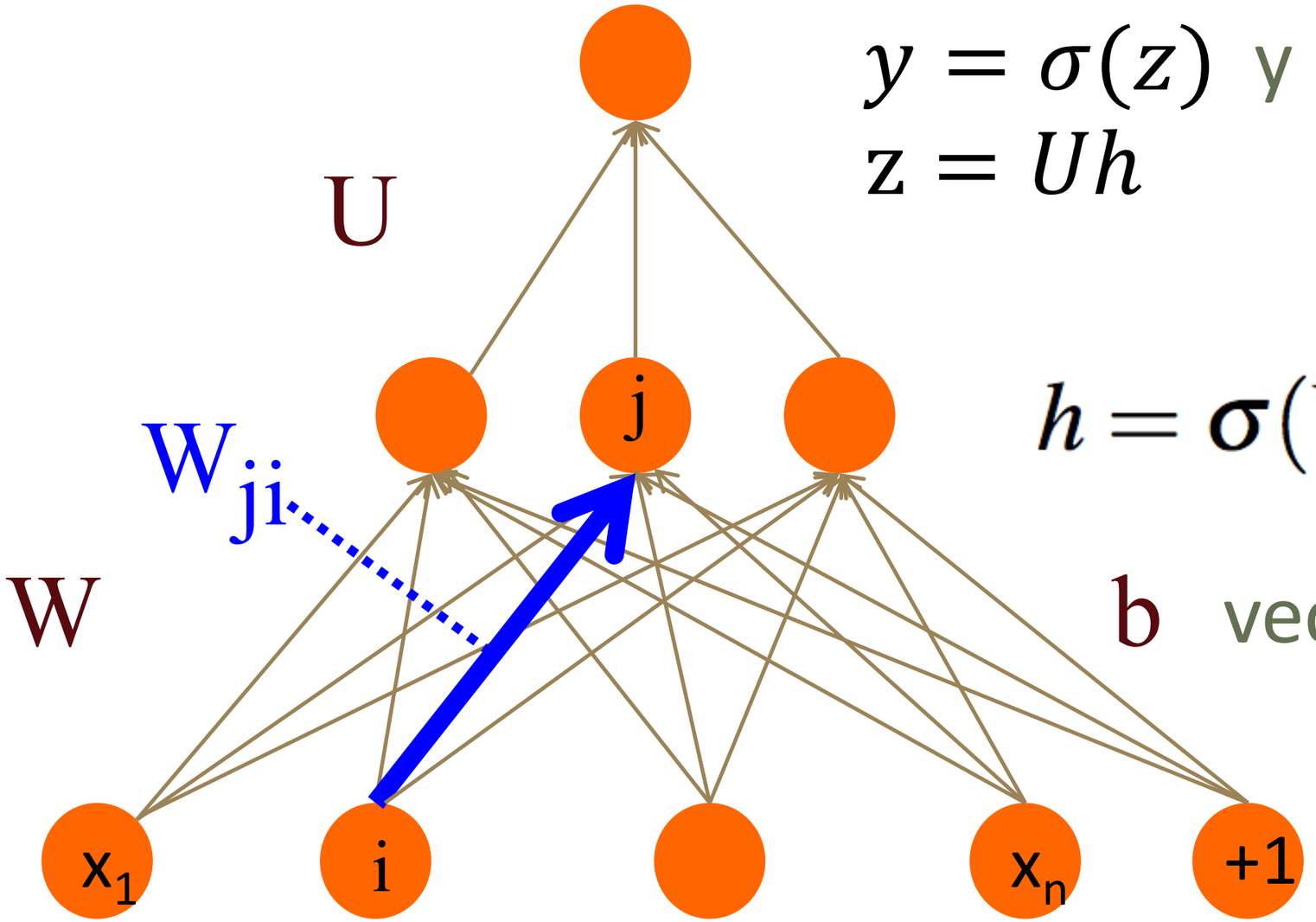
Could be ReLU
Or tanh

Two-Layer Network with scalar output

Output layer
(σ node)

hidden units
(σ node)

Input layer
(vector)



$y = \sigma(z)$ y is a scalar

$z = Uh$

$h = \sigma(Wx + b)$

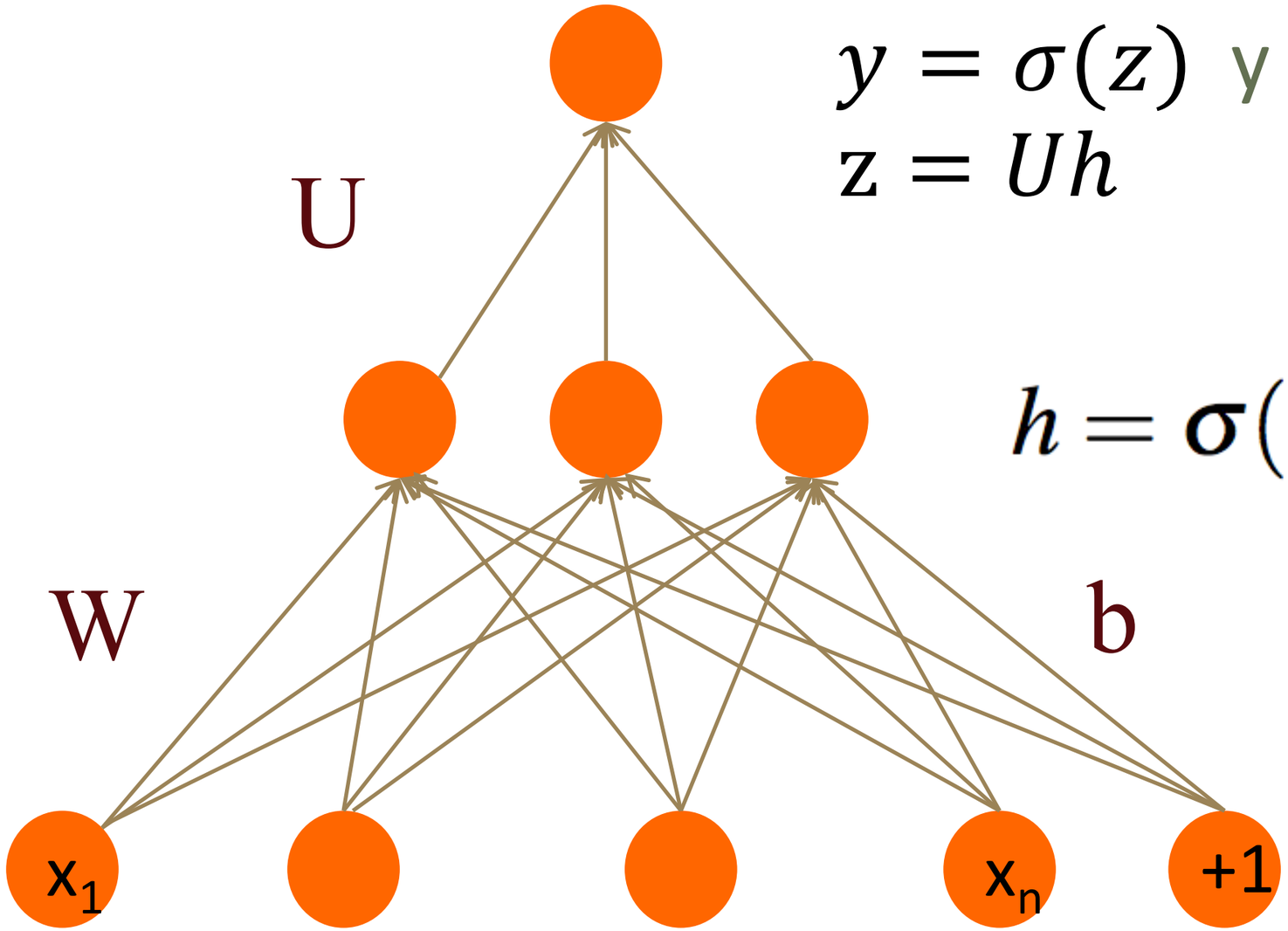
b vector

Two-Layer Network with scalar output

Output layer
(σ node)

hidden units
(σ node)

Input layer
(vector)



$y = \sigma(z)$ y is a scalar

$z = Uh$

$h = \sigma(Wx + b)$

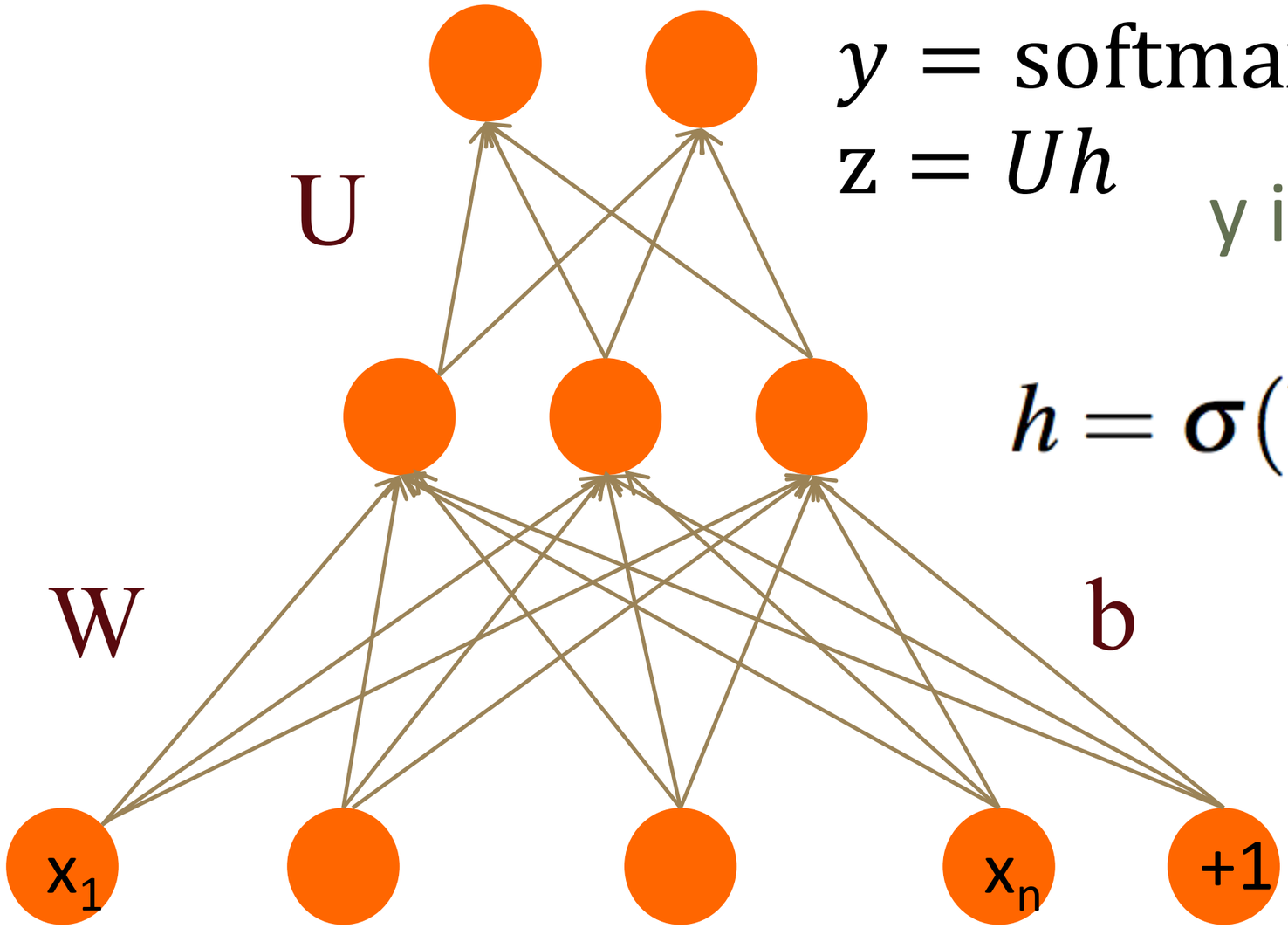
Could be ReLU
Or tanh

Two-Layer Network with softmax output

Output layer
(σ node)

hidden units
(σ node)

Input layer
(vector)



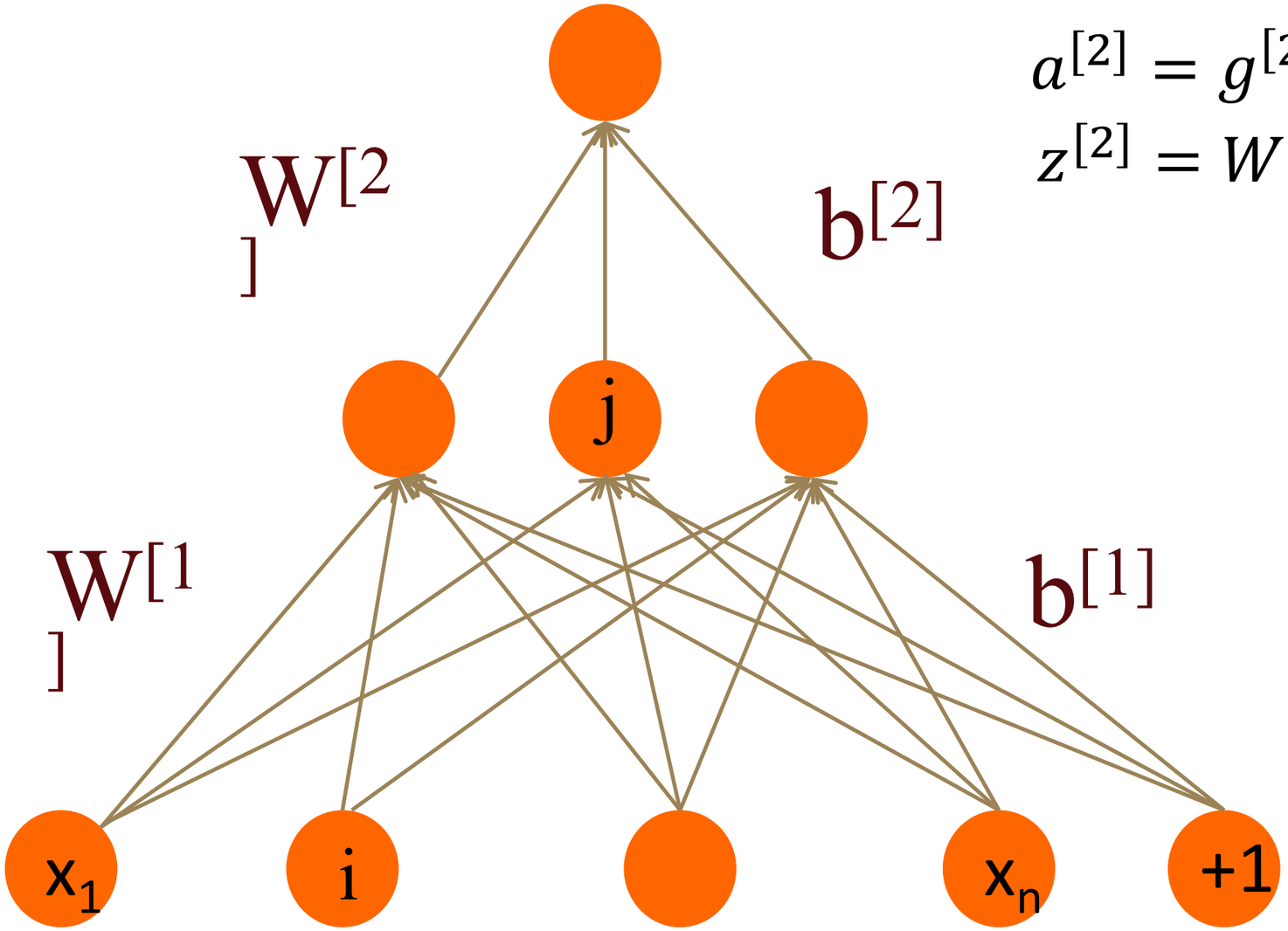
$$y = \text{softmax}(z)$$
$$z = Uh$$

y is a vector

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

Could be ReLU
Or tanh

Multi-layer Notation



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

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

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

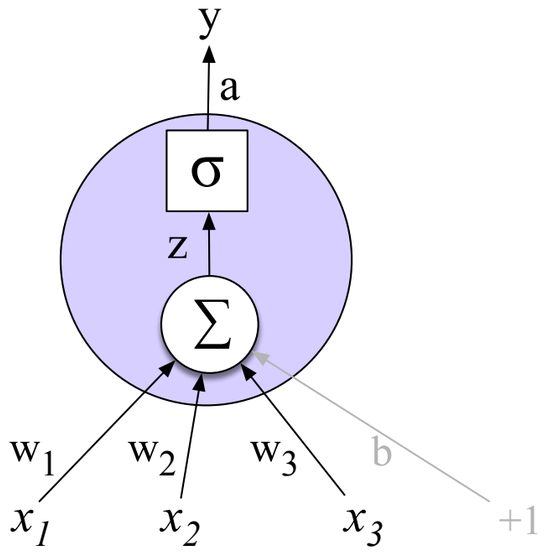
$$a^{[1]} = g^{[1]}(z^{[1]})$$ ReLU

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

$a^{[0]}$

Multi Layer Notation

$$
\begin{aligned}
z^{[1]} &= W^{[1]} a^{[0]} + b^{[1]} \\
a^{[1]} &= g^{[1]}(z^{[1]}) \\
z^{[2]} &= W^{[2]} a^{[1]} + b^{[2]} \\
a^{[2]} &= g^{[2]}(z^{[2]}) \\
\hat{y} &= a^{[2]}
\end{aligned}
$$



for i **in** 1..n

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

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 $a^{[1]}_0=1$, $a^{[2]}_0=1$,…

Replacing the bias unit

Instead of:

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

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

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

We'll do this:

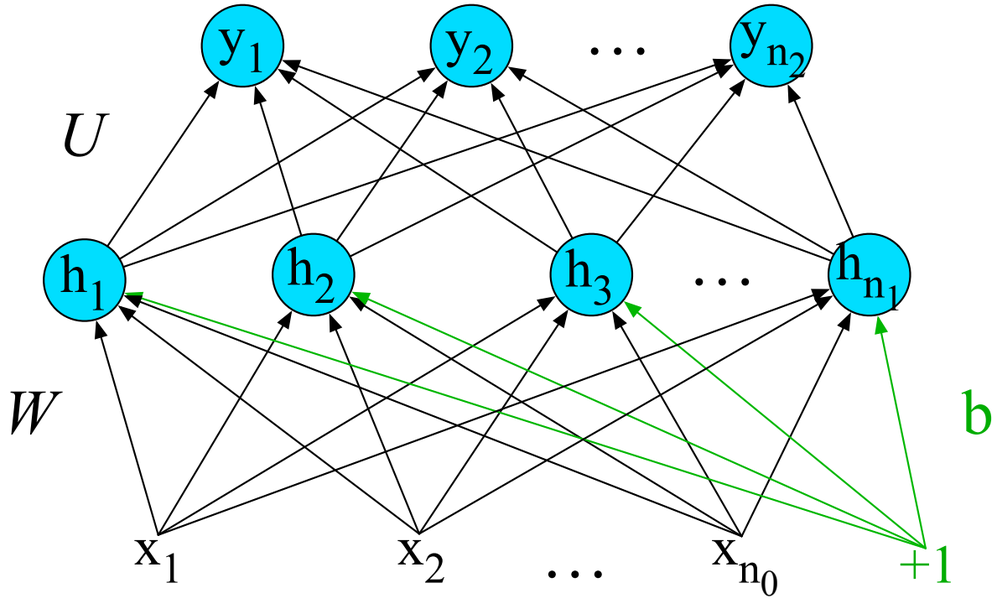
$$x = x_0, x_1, x_2, \ldots, x_{n0}$$

$$h = \sigma(Wx)$$

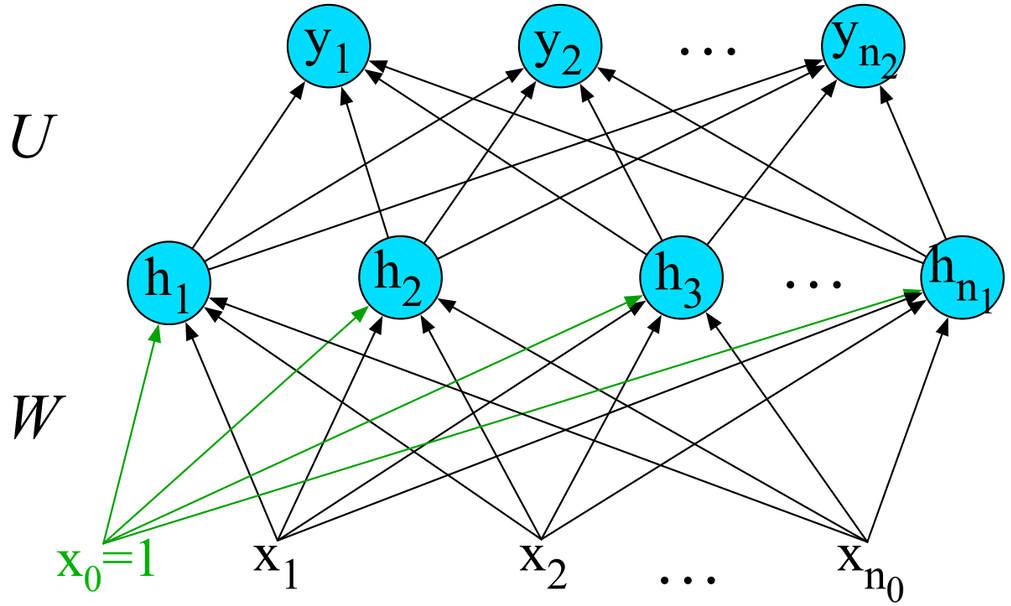
$$\sigma\left(\sum_{i=0}^{n_0} W_{ji}x_i\right)$$

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

Feedforward networks for simple classification

Use cases for feedforward networks

Let's consider 2 (simplified) sample tasks:

1. Text classification
2. Language modeling

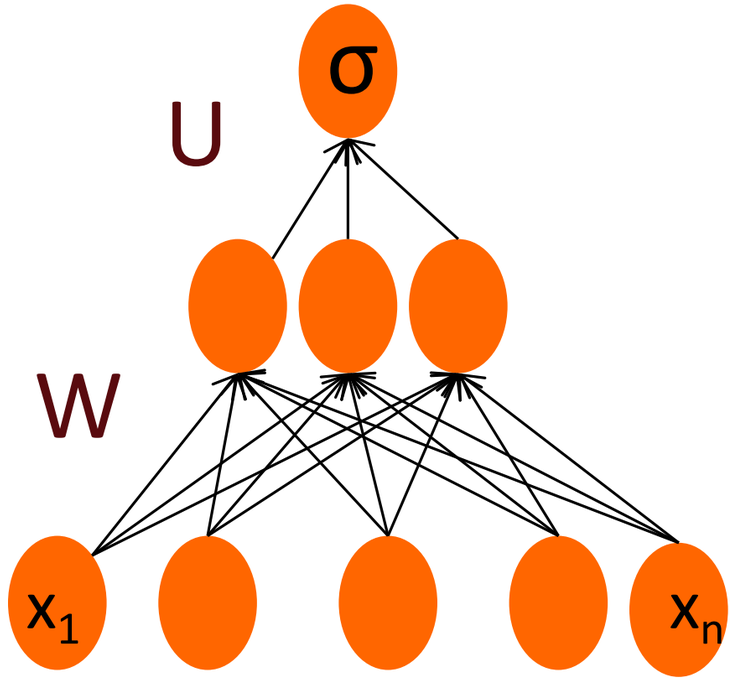
State of the art systems use more powerful neural classifiers like BERT/MLM classifiers, but simple models will introduce some important ideas

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

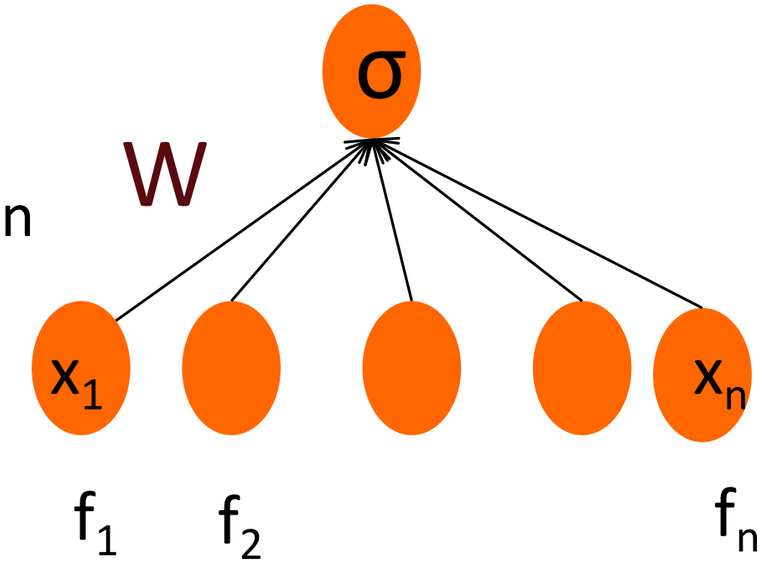


Sentiment Features

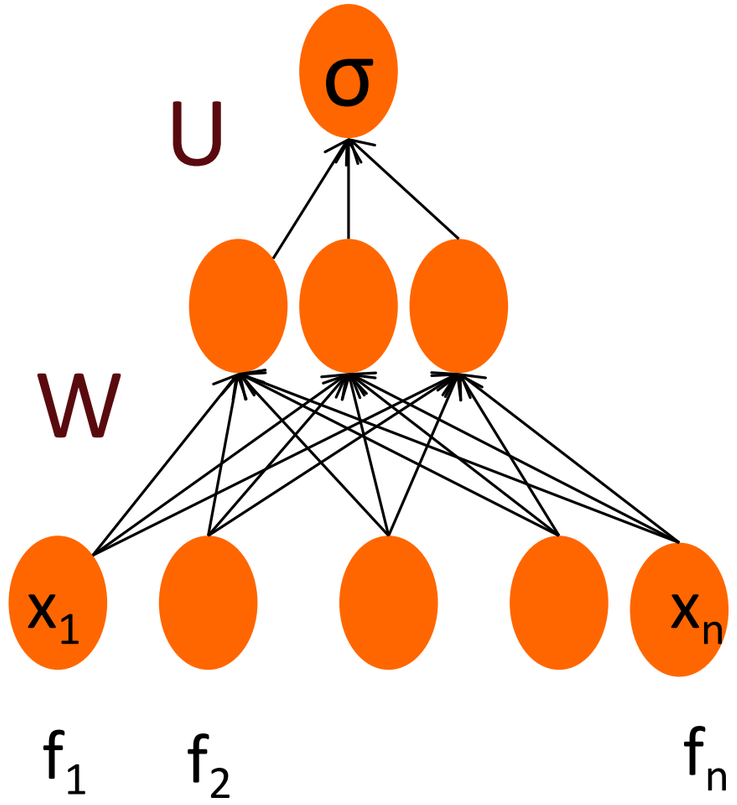
Var	Definition
x_1	count(positive lexicon) $\in$ doc)
x_2	count(negative lexicon) $\in$ doc)
x_3	$\begin{cases} 1 & \text{if } \text{“no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_4	count(1st and 2nd pronouns $\in$ doc)
x_5	$\begin{cases} 1 & \text{if } \text{“!”} \in \text{doc} \\ 0 & \text{otherwise}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

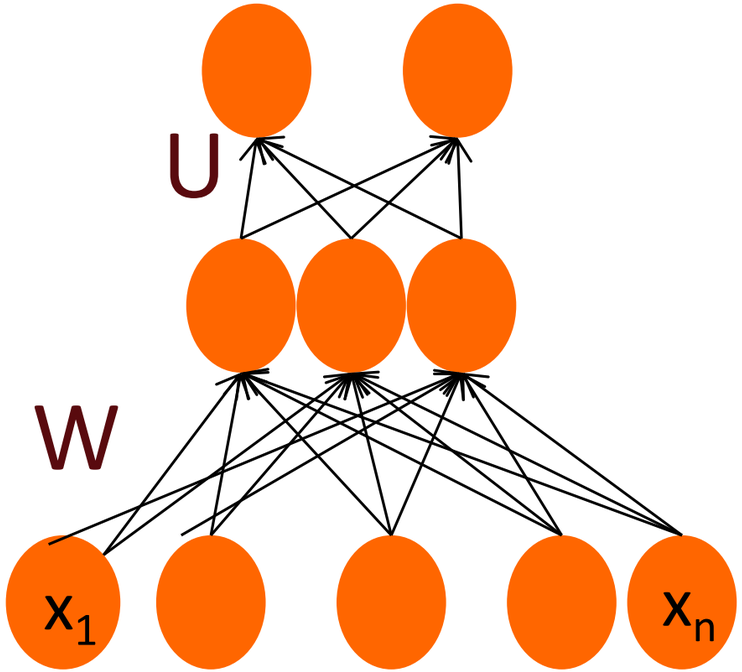
- allows the network to use non-linear interactions between features
- which may (or may not) improve performance.

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”

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



The output layer

$\hat{y}$ could have two nodes (one each for positive and negative), or 3 nodes (positive, negative, neutral).

- $\hat{y}_1$ estimated probability of positive sentiment
- $\hat{y}_2$ probability of negative
- $\hat{y}_3$ probability of neutral

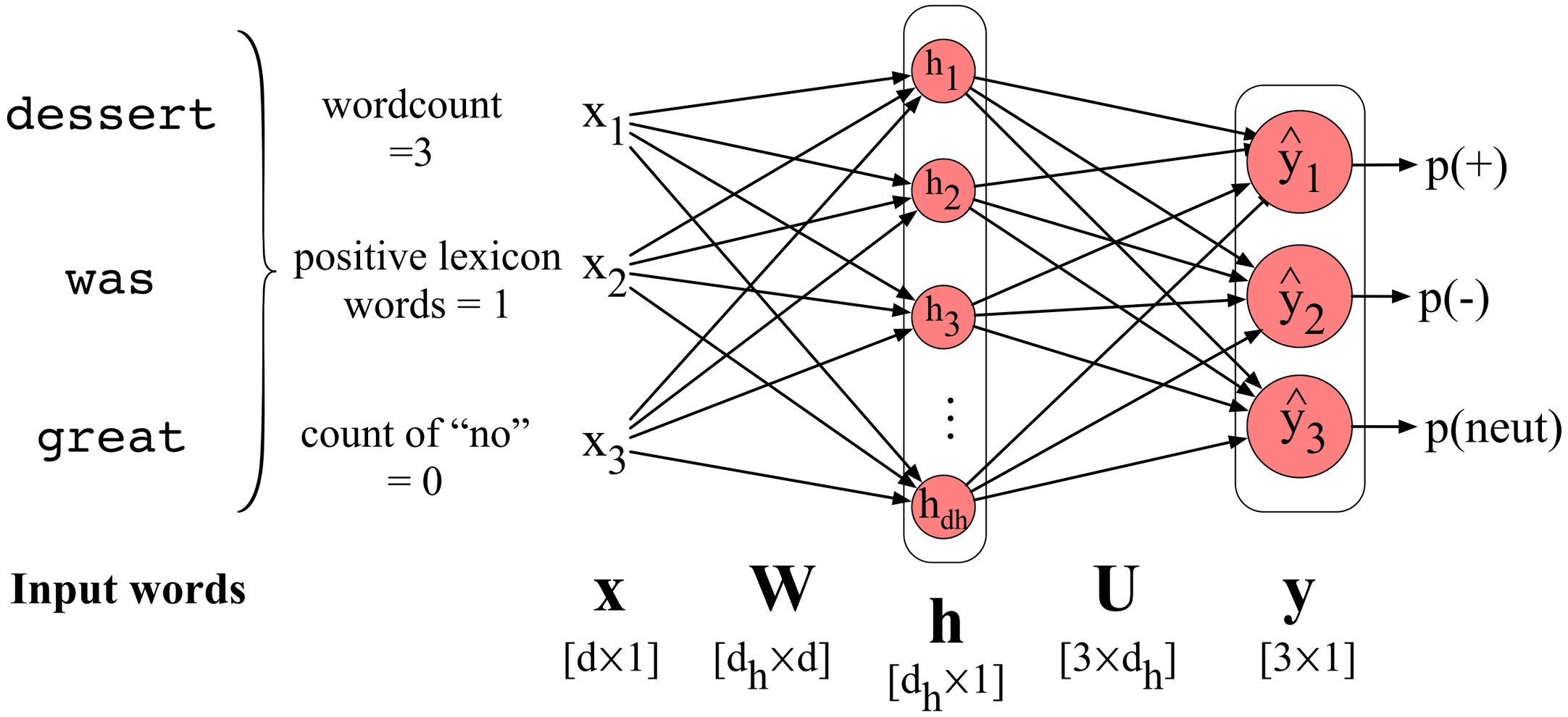
Equations for NN classification with hand features

$$\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_d] \quad \text{(each } \mathbf{x}_i \text{ is a hand-designed feature)}$$

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\mathbf{z} = \mathbf{U}\mathbf{h}$$

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


dessert
was
great
wordcount =3
positive lexicon words = 1
count of "no" = 0
x1
x2
x3
h1
h2
h3
hdh
ŷ1
ŷ2
ŷ3
p(+)
p(-)
p(neut)
Input words
x
[d×1]
W
[dh×d]
h
[dh×1]
U
[3×dh]
y
[3×1]
Input layer
d=3 features
Hidden layer
Output layer
softmax

Vectoring for parallelizing inference

We would like to efficiently classify the whole test set of m observations.

So we vectorize: pack all the input features into **X**

- Each row $\mathbf{x}^{(i)}$ of **X** is a row vector with all the features for example $x^{(i)}$
- Feature dimensionality is *d*, **X** is [*m* x *d*]

Slight changes to equations

Each input is now a row vector

- **X** is of shape [m x d]
- **W** is of shape [d_h x d]
- We'll need to do some reordering and transposing
- Bias vector **b** that used to be [1 x d_h] is now [m x d_h]

$$\mathbf{H} = \sigma(\mathbf{X}\mathbf{W}^\intercal + \mathbf{b})$$
$$\mathbf{Z} = \mathbf{H}\mathbf{U}^\intercal$$
$$\hat{\mathbf{Y}} = \text{softmax}(\mathbf{Z})$$

Simple Neural Networks and Neural Language Models

Feedforward networks for simple classification

Simple Neural Networks and Neural Language Models

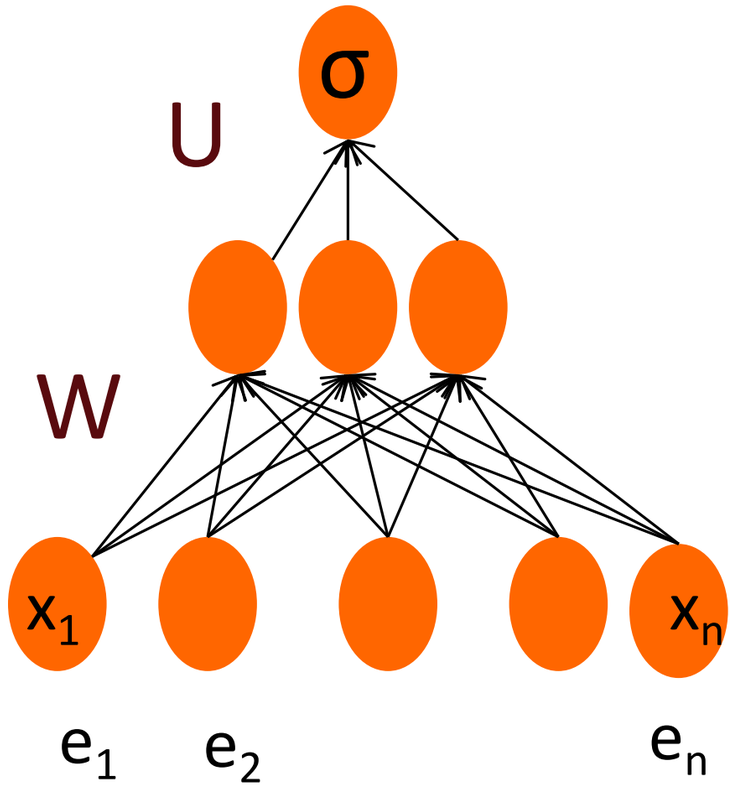
Embeddings as input to feedforward classifiers

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!



Embedding matrix **E**

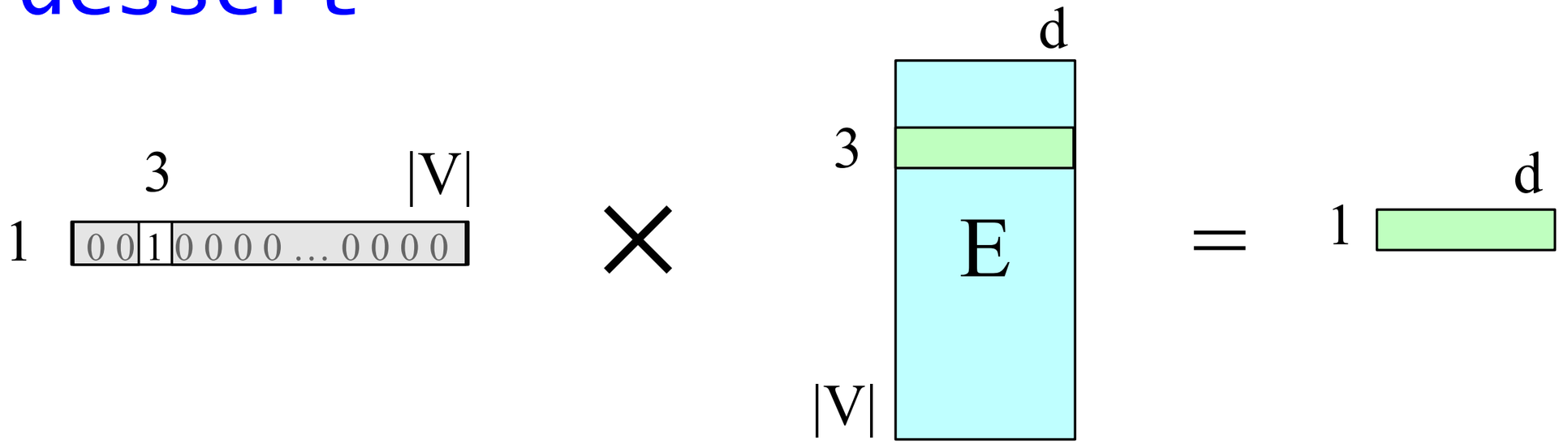
- An embedding is a vector of dimension [1 x *d]* that represents the input token.
- An embedding matrix **E** is a dictionary, one row per token of vocab *V*
- **E** has shape $[|V| \times d]$
- Embedding matrices are central to NLP; they represent input text in LLMs and all NLP tools

Text classification from embeddings

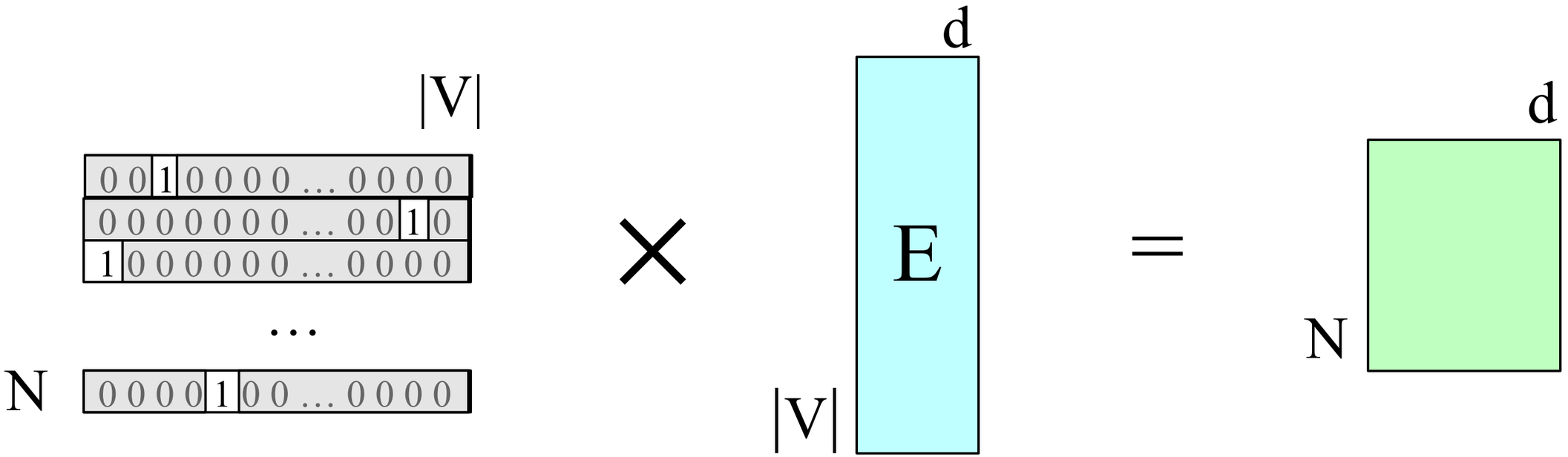
- Given tokenized input: dessert was great
- Select the embedding vectors from **E:**
  1. Convert BPE tokens into vocabulary indices
     - w = [3, 9824, 226]
  2. Use indexing to select the corresponding rows from **E**
     - row 3, row 4000, row 10532

Another way to think of indexing from **E**

- Treat each input word as one-hot vector
  - If `dessert` is index 3: `[0 0 1 0 0 0 0 ... 0 0 0 0]`
    `1 2 3 4 5 6 7 ... ... |V|`
- Multiply it by **E** to select out the embedding for `dessert`



Collecting embeddings from E for *N* words



Given window of N tokens, represented by *N* [1 × *d*] embeddings

Need to return a single class (e.g., pos or neg)

Text comes in different lengths

Given window of N tokens, represented by N [1 × d] embeddings, return a single class (e.g., pos or neg)

1. **Concatenate** all the inputs into one long vector of shape [1×dN], i.e. input is length N of longest review
   - If shorter then pad with zero embeddings
   - Truncate if you get longer reviews at test time
2. **Pool** the inputs into a single short [1 × d] vector. A single "sentence embedding" (the same dimensionality as a word) to represent all the words. Less info, but very efficient and fast.

Pooling

Intuition: exact position not so important for sentiment.

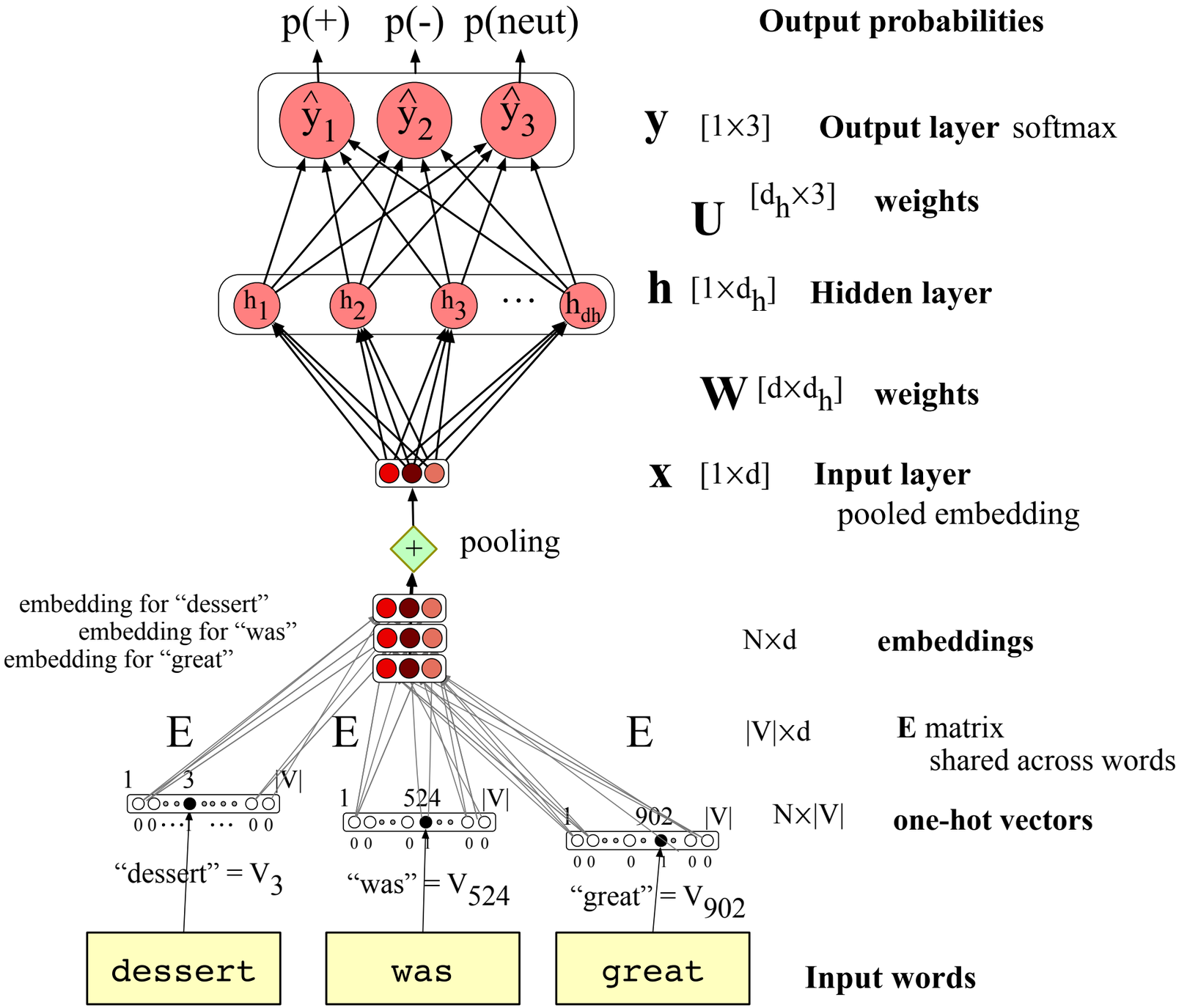
We'll just do some sort of averaging of all the vectors.

Mean pooling:

$$\mathbf{x}_{mean} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{e}(w_i)$$

$$\begin{aligned}
\mathbf{x} &= \text{mean}(\mathbf{e}(w_1), \mathbf{e}(w_2), \ldots, \mathbf{e}(w_n)) \\
\mathbf{h} &= \sigma(\mathbf{x}\mathbf{W} + \mathbf{b}) \\
\mathbf{z} &= \mathbf{h}\mathbf{U} \\
\hat{\mathbf{y}} &= \text{softmax}(\mathbf{z})
\end{aligned}$$

Pooling


p(+) p(-) p(neut)
Output probabilities
$\hat{y}_1$ $\hat{y}_2$ $\hat{y}_3$
y [1×3] Output layer softmax
U [d_h×3] weights
h_1 h_2 h_3 … h_{dh}
h [1×d_h] Hidden layer
W [d×d_h] weights
x [1×d] Input layer
pooled embedding
+
pooling
embedding for "dessert"
embedding for "was"
embedding for "great"
N×d embeddings
E E E
|V|×d E matrix
shared across words
1 3 |V|
1 524 |V|
1 902 |V|
N×|V| one-hot vectors
"dessert" = V_3
"was" = V_{524}
"great" = V_{902}
dessert
was
great
Input words

Simple Neural Networks and Neural Language Models

Embeddings as input to feedforward classifiers

Simple Neural Networks and Neural Language Models

Neural language modeling with feedforward networks

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** introduce many of the important concepts

Simple feedforward Neural Language Models

Task: predict next word w_t

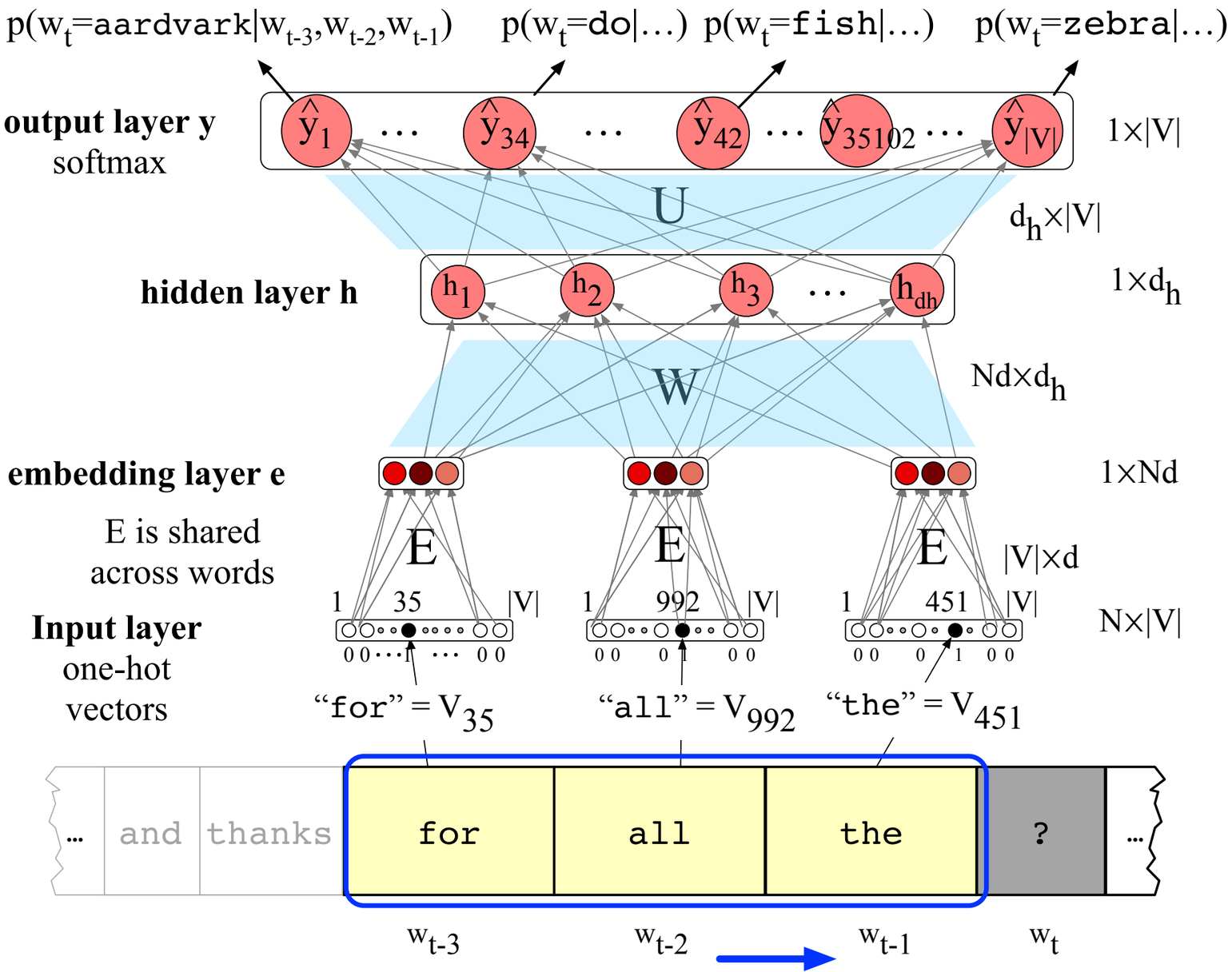
given prior words w_{t-1}, w_{t-2}, w_{t-3}, …

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

Solution: Sliding windows (of fixed length)

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

Inference in a Feedforward Language Model



Feedforward language model equations

$$\mathbf{e} = [\mathbf{E}\mathbf{x}_{t-3}; \mathbf{E}\mathbf{x}_{t-2}; \mathbf{E}\mathbf{x}_{t-1}]$$
$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{e} + \mathbf{b})$$
$$\mathbf{z} = \mathbf{U}\mathbf{h}$$
$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{z})$$

So $\hat{y}_{42}$ is the probability of the next word w_t being V_{42} = `fish`

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

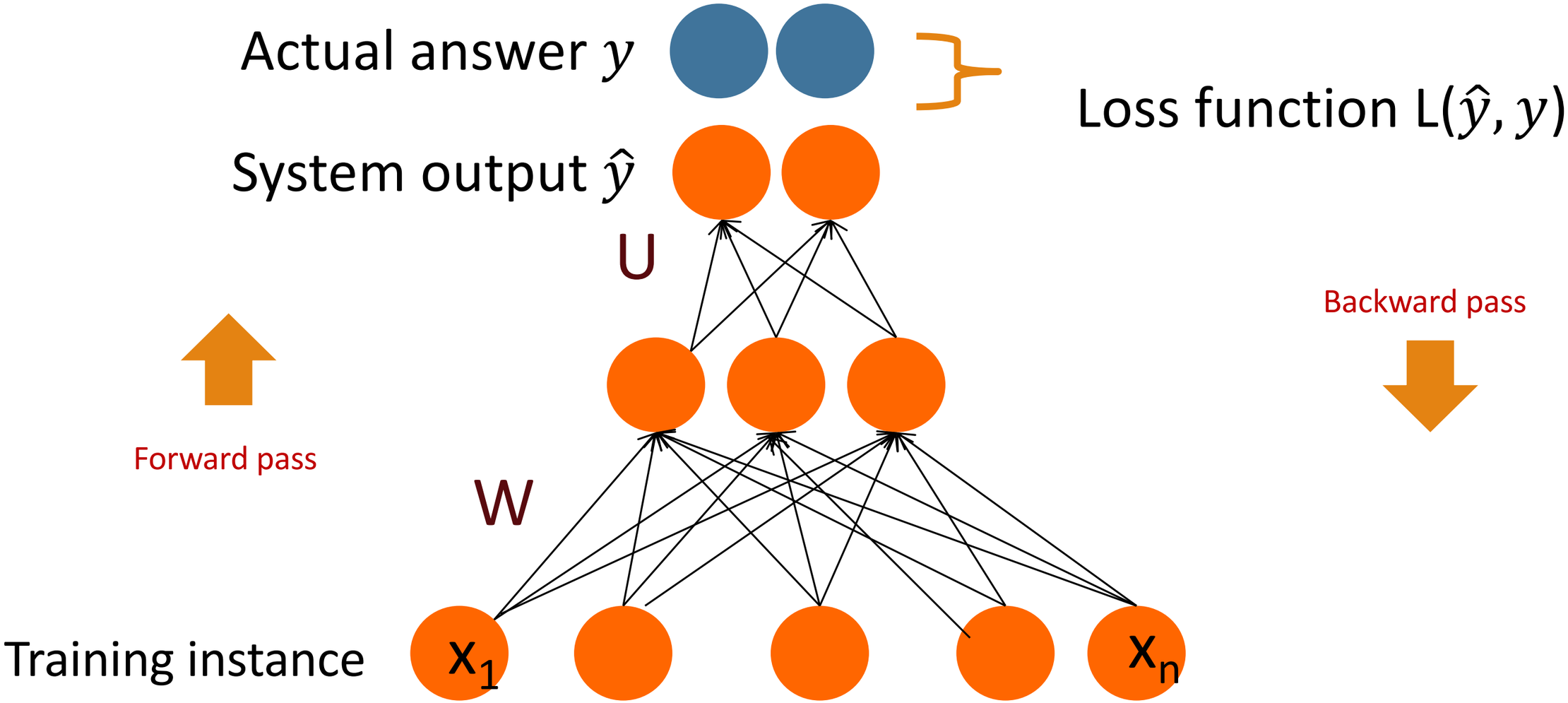
Simple Neural Networks and Neural Language Models

Neural language modeling with feedforward networks

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:

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

$$= -[y \log \sigma(w \cdot x + b) + (1-y)\log(1-\sigma(w \cdot x + b))]$$

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

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

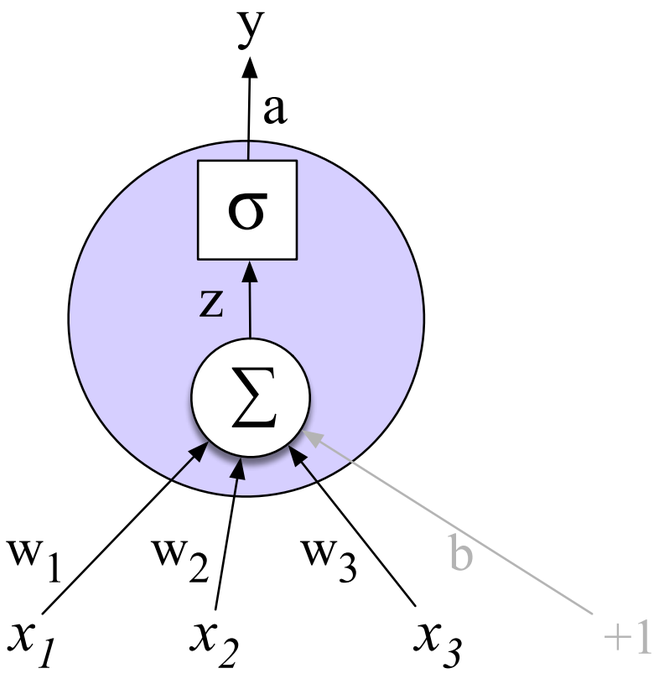
- For logistic regression

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

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**.

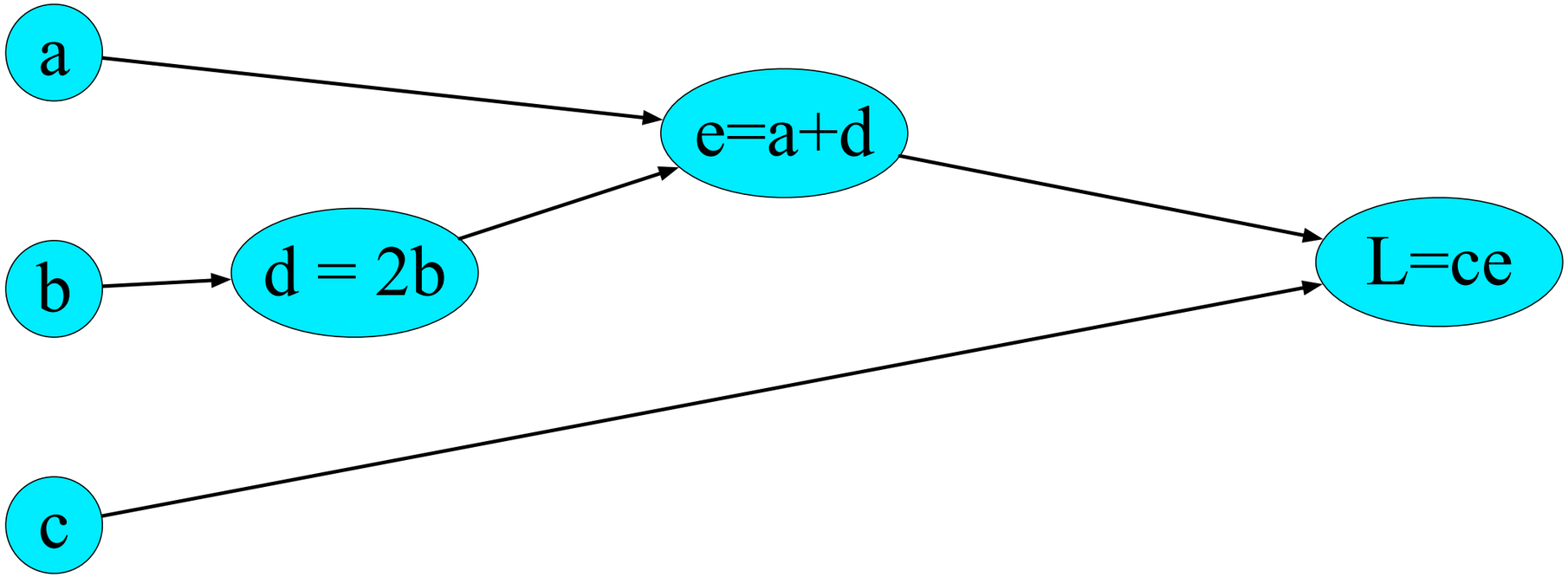
Computation Graphs

A computation graph represents the process of computing a mathematical expression

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

Computations:

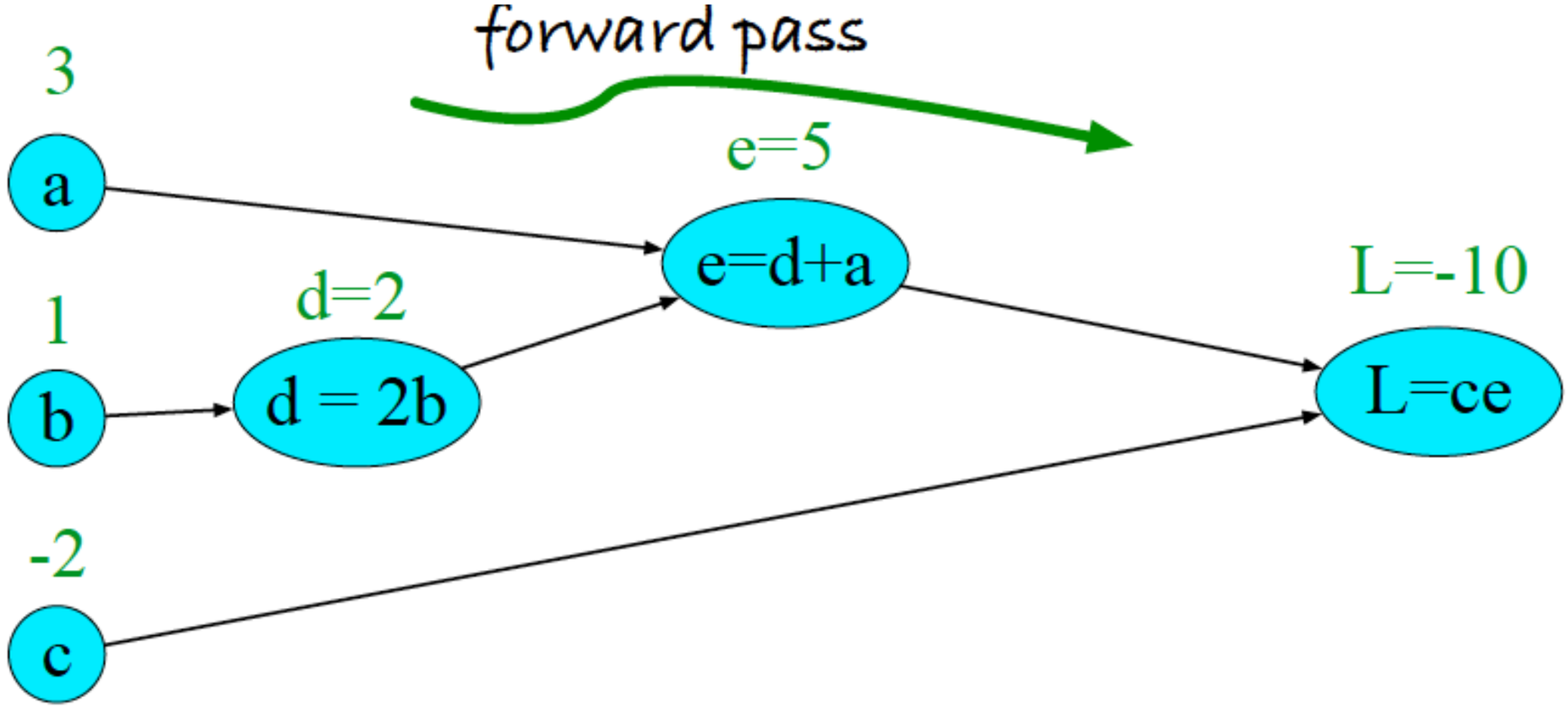
$$d = 2*b$$
$$e = a+d$$
$$L = c*e$$



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

Computations:

$$d = 2*b$$
$$e = a+d$$
$$L = c*e$$



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

$$
\begin{aligned}
d &= 2*b \\
e &= a+d \\
L &= c*e
\end{aligned}
$$

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:

$$f(x) = u(v(x)) \qquad \frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx}$$

$$f(x) = u(v(w(x))) \qquad \frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dw} \cdot \frac{dw}{dx}$$

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

$$
\begin{aligned}
d &= 2*b \\
e &= a+d \\
L &= c*e
\end{aligned}
$$

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

Example

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

$$\begin{aligned} d &= 2*b \\ e &= a+d \\ L &= c*e \end{aligned}$$

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

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

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

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

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

Example

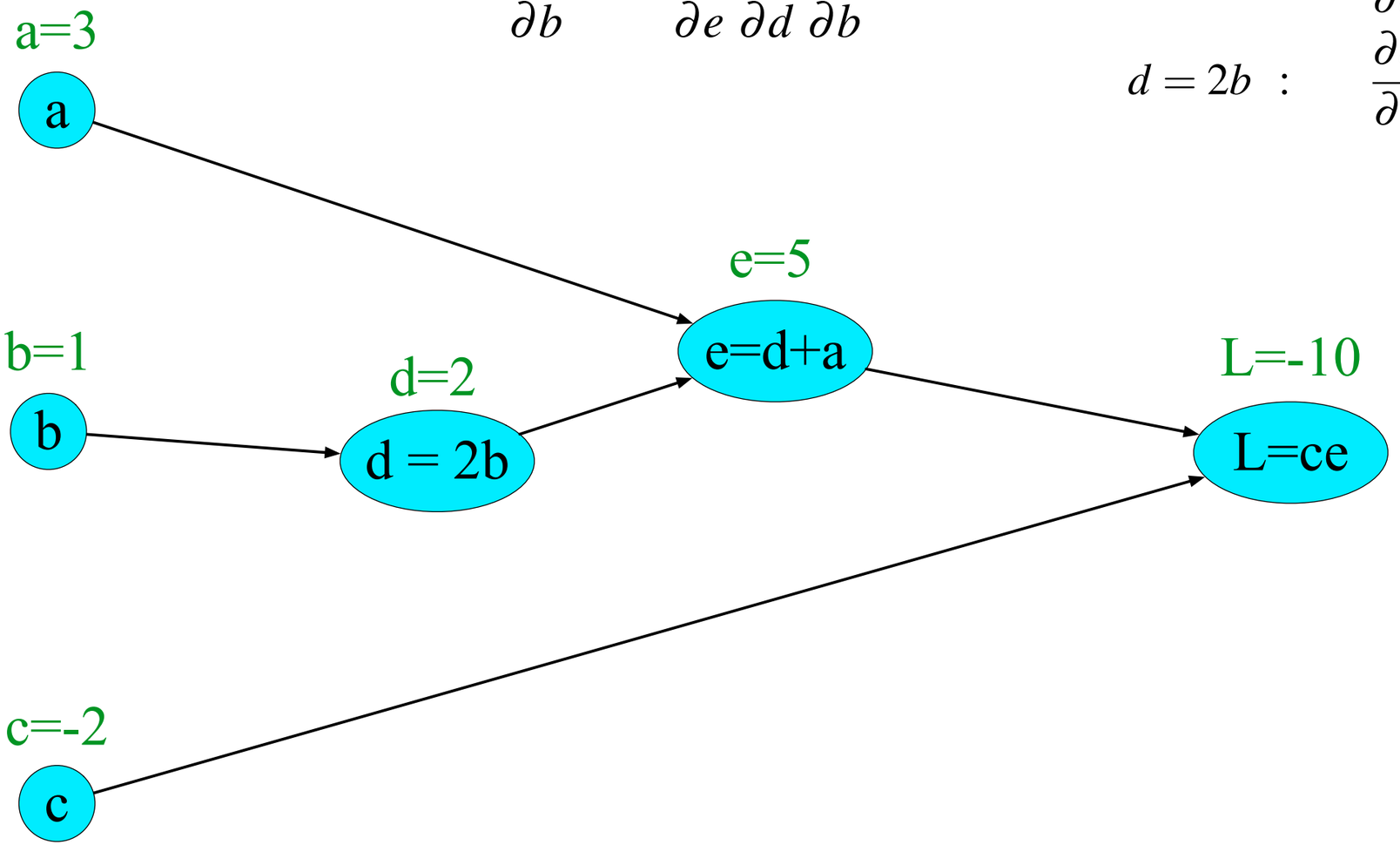
$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial a}$$

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

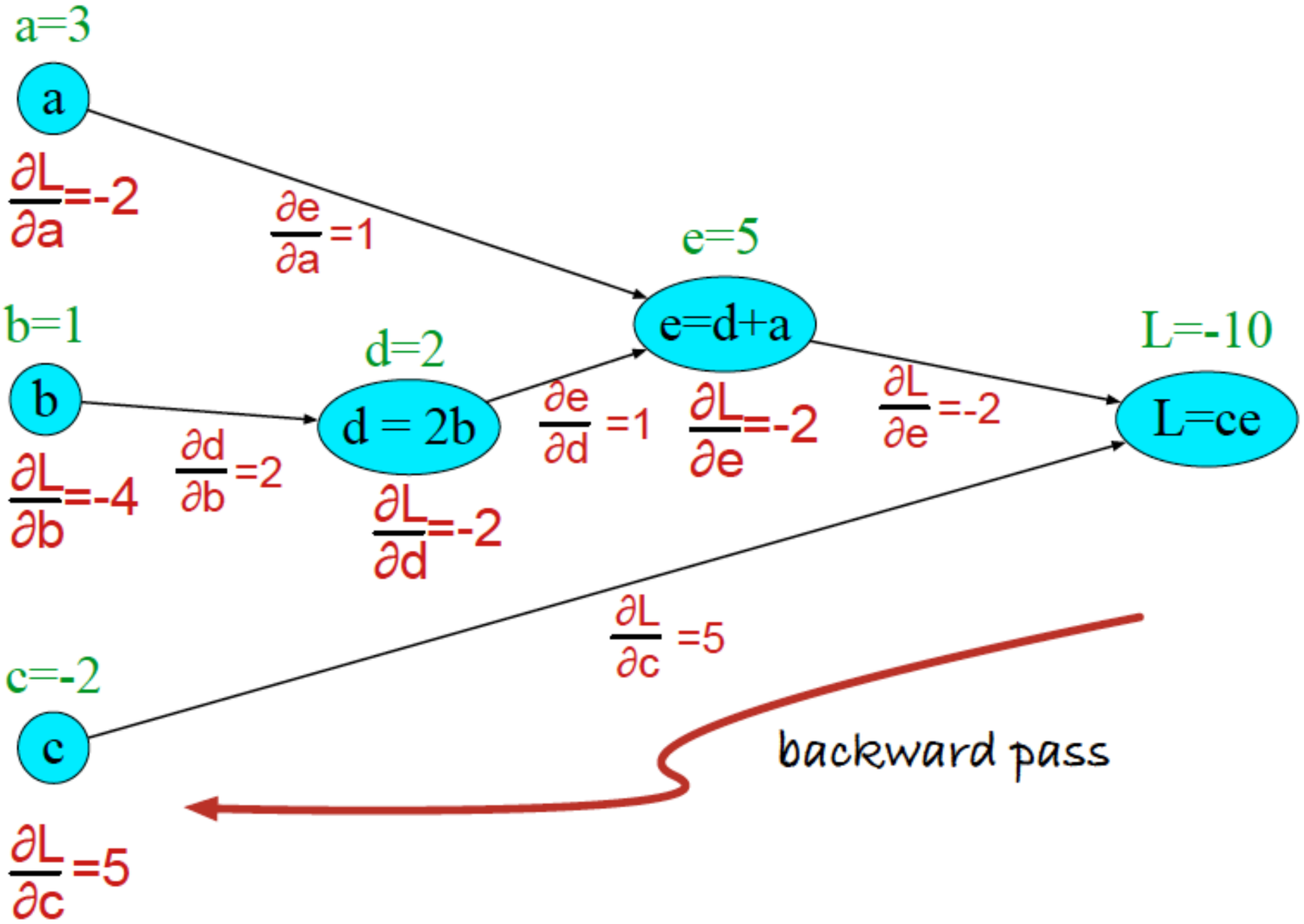
$$L = ce : \quad \frac{\partial L}{\partial e} = c, \frac{\partial L}{\partial c} = e$$

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

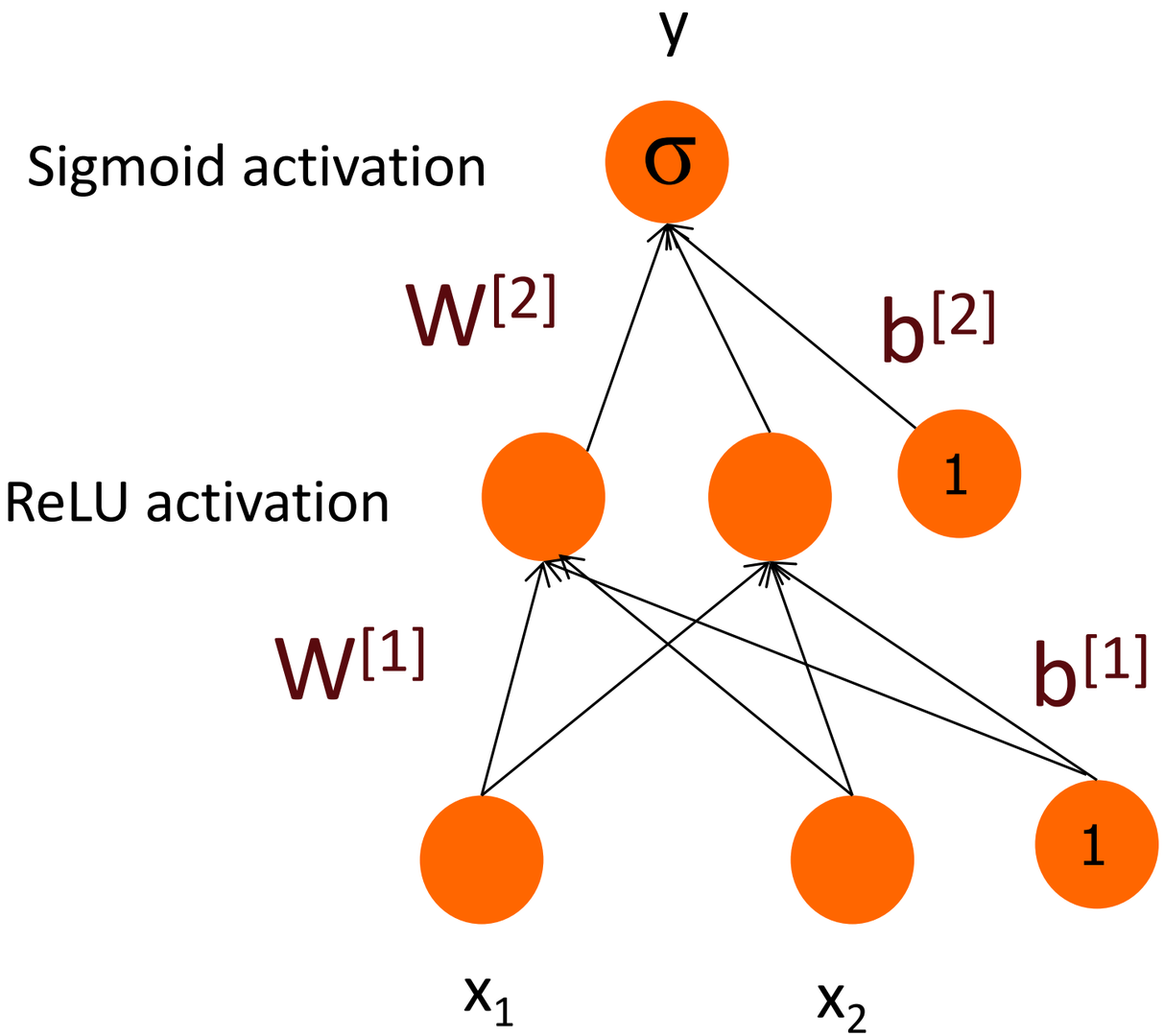
$$d = 2b : \quad \frac{\partial d}{\partial b} = 2$$



Example


a=3
a
∂L/∂a=-2
∂e/∂a=1
e=5
e=d+a
b=1
b
∂L/∂b=-4
∂d/∂b=2
d=2
d = 2b
∂e/∂d=1
∂L/∂d=-2
∂L/∂e=-2
∂L/∂e=-2
L=-10
L=ce
∂L/∂c=5
c=-2
c
backward pass
∂L/∂c=5

Backward differentiation on a two layer network



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

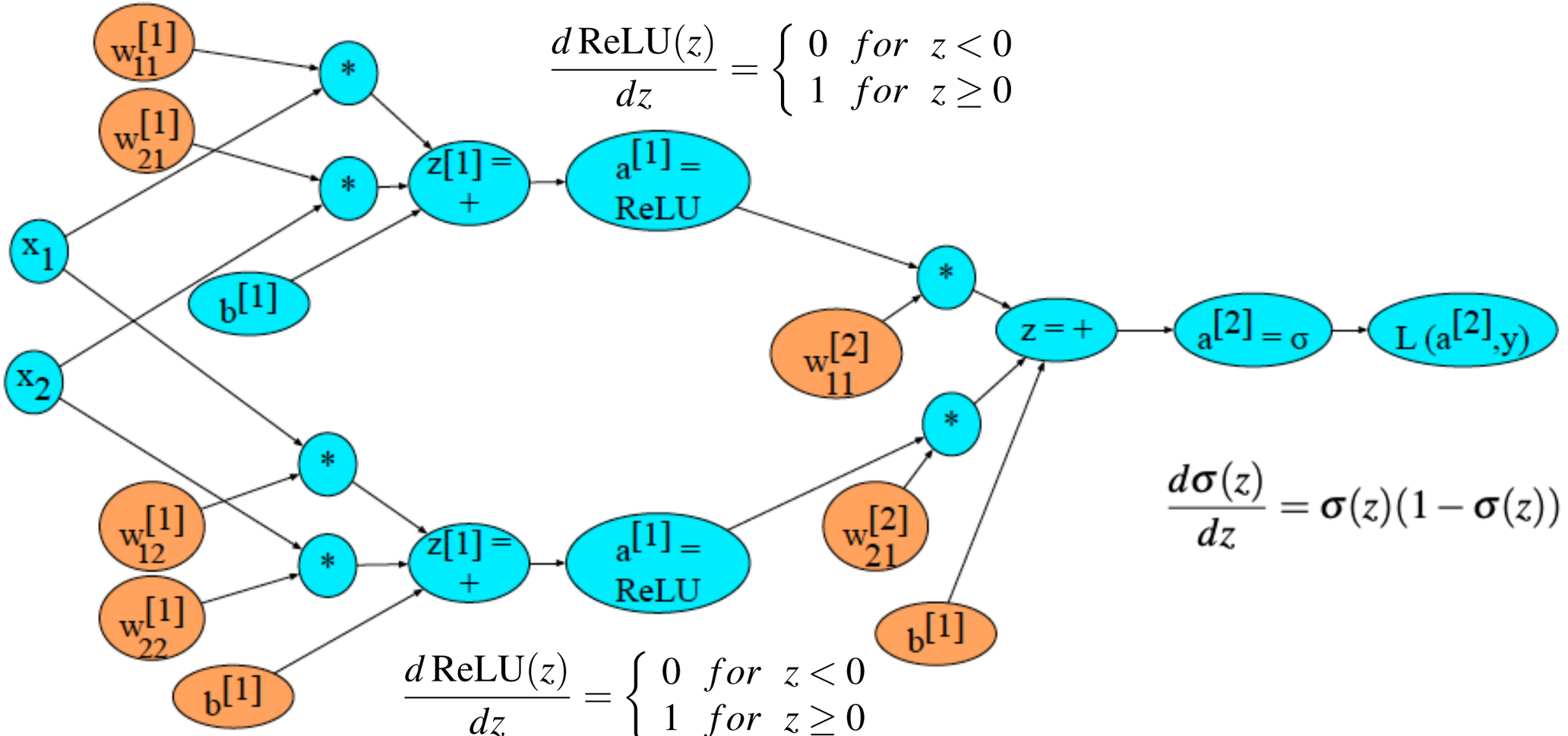
Backward differentiation on a two layer network

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

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

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

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 $z^{[2]}$)

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

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

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

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

$$\frac{\partial L}{\partial a} = -\left(\left(y\frac{\partial \log(a)}{\partial a}\right) + (1-\mathrm{y})\,\frac{\partial \log(1-a)}{\partial a}\right)$$

$$= -\left(\left(y\frac{1}{a}\right) + (1-\mathrm{y})\,\frac{1}{1-a}(-1)\right) = -\left(\frac{y}{a} + \frac{y-1}{1-a}\right)$$

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

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

NATURAL LANGUAGE PROCESSING (23AIPC401)
MODULE 3: EMBEDDINGS & CLASSIFICATION
QUESTION BANK

A. Lexical Semantics

1. Explain the concepts of lemma and word sense with a suitable example. Illustrate how a single lemma can have multiple senses. (06 Marks | L2 | CO3)
2. Differentiate synonymy, similarity and word relatedness with suitable examples. (06 Marks | L2 | CO3)
3. Explain antonymy and its major forms. Illustrate binary, scalar and reversive oppositions with examples. (08 Marks | L2 | CO3)
4. Classify the pairs "car–bicycle" and "coffee–cup" as similar or related, and justify your classification. (06 Marks | L3 | CO3)
5. Explain semantic fields and illustrate how words within a semantic field are structurally related using two domains. (06 Marks | L2 | CO3)
6. Explain connotation and sentiment. Describe valence, arousal and dominance as dimensions of affective meaning. (08 Marks | L2 | CO3)
7. Classify "excellent", "happy", "terrible" and "fake" by likely positive or negative connotation, and explain the distinction between connotation and literal meaning. (06 Marks | L3 | CO3)

B. Vector Semantics

8. Explain vector semantics and how words occurring in similar contexts can be represented by nearby vectors. (06 Marks | L2 | CO3)
9. Illustrate how a word is represented as a point in a multidimensional vector space. Explain the meaning of vector dimensions. (06 Marks | L2 | CO3)
10. Differentiate sparse count-based vectors and dense word embeddings with respect to representation, dimensionality and interpretability. (08 Marks | L2 | CO2)
11. Explain how distributional information is used to represent word meaning in vector semantics, with an example. (06 Marks | L2 | CO3)
12. A word-context matrix has 50,000 rows and 50,000 columns. Calculate its total number of entries. If most entries are zero, explain why the representation is sparse. (06 Marks | L3 | CO2)

C. Simple Count-Based Embeddings

13. Explain how a word-context co-occurrence matrix is constructed from a text corpus. Illustrate the process using a small example. (08 Marks | L2 | CO2)
14. Consider: (1) "cherry pie tastes sweet", (2) "strawberry pie tastes sweet", (3) "digital information is useful". Identify context words and explain how co-occurrence counts form word vectors. (08 Marks | L3 | CO2)
15. Given vectors cherry=[2,1,0], strawberry=[2,1,0] and digital=[0,0,3], calculate pairwise dot products and identify the pair with the strongest dot-product similarity. (06 Marks | L3 | CO2)
16. Given vectors cherry=[4,2,0], strawberry=[3,2,0] and digital=[0,1,5], calculate dot products for (cherry, strawberry) and (cherry, digital), and interpret the results. (08 Marks | L3 | CO2)
17. Explain why a word-context matrix becomes large and sparse as vocabulary size increases. Mention the computational implications. (06 Marks | L2 | CO2)
18. Describe the role of context-window size in constructing count-based word representations. (06 Marks | L2 | CO2)

D. Cosine Similarity

19. Define cosine similarity and explain why it is preferred over raw dot product for comparing word vectors. (06 Marks | L2 | CO2)
20. Given A=[1,2,2] and B=[2,1,2], calculate their dot product, vector magnitudes and cosine similarity. (08 Marks | L3 | CO2)
21. Given A=[2,1,0] and B=[1,2,0], calculate cosine similarity and interpret the result. (06 Marks | L3 | CO2)
22. The vectors for two words are A=[3,4] and B=[6,8]. Calculate cosine similarity and explain what the result indicates about their directions. (06 Marks | L3 | CO2)
23. For A=[1,0], B=[0,1] and C=[1,1], calculate cosine similarities for (A,B), (A,C) and (B,C). Identify the most similar pair. (08 Marks | L3 | CO2)
24. Explain the range of cosine similarity values and interpret values close to 1, 0 and -1. (06 Marks | L2 | CO2)
25. A raw dot-product comparison ranks a frequent word higher because its vector has larger values. Explain how cosine normalization addresses this issue. (06 Marks | L3 | CO2)

E. Word2Vec

26. Explain the basic idea of Word2Vec and how it differs from count-based embedding methods. (06 Marks | L2 | CO3)
27. Describe the Skip-gram model and explain how it uses a target word to predict surrounding context words. (08 Marks | L2 | CO3)
28. Explain the role of positive and negative training examples in Skip-gram with Negative Sampling (SGNS). (06 Marks | L2 | CO3)
29. Given “the student studies machine learning”, use a context-window size of 1 to identify positive target-context pairs for target word “studies”. (06 Marks | L3 | CO3)
30. For “the student studies machine learning”, identify context words for target “machine” using a context window of 2. List the positive training pairs. (06 Marks | L3 | CO3)
31. Describe the steps involved in learning Skip-gram embeddings, from random initialization to retaining the learned vectors. (08 Marks | L2 | CO3)
32. Explain how Word2Vec adjusts embeddings to increase similarity for positive target-context pairs and decrease similarity for negative pairs. (08 Marks | L2 | CO3)
33. A Skip-gram example has target “apricot” and context words “tablespoon”, “of”, “jam” and “a”. Identify positive pairs and explain the use of negative samples. (08 Marks | L3 | CO3)
34. Explain why the classifier used during Word2Vec training can be discarded while the learned embeddings are retained. (06 Marks | L2 | CO3)

G. Feedforward Networks for NLP Classification

35. Explain the architecture of a feedforward neural network, identifying input, hidden and output layers. (06 Marks | L2 | CO3)
36. Illustrate the flow of data through a feedforward network used for text classification, from input features to predicted class. (08 Marks | L2 | CO3)
37. Explain the roles of weights, biases and activation functions in a feedforward neural network. (06 Marks | L2 | CO3)
38. Explain how a feedforward neural network can perform sentiment classification using input features extracted from text. (08 Marks | L2 | CO3)

H. Embeddings as Input to Neural Network Classifiers

39. Explain how pretrained word embeddings can be used as input features to a neural-network classifier. (06 Marks | L2 | CO3)
40. With an example, illustrate how a text document can be converted into a fixed-size vector representation using word embeddings for classification. (08 Marks | L2 | CO3)
41. A document contains “good”, “movie” and “excellent”, with vectors [1,0], [0,1] and [2,2]. Calculate the average document embedding. (06 Marks | L3 | CO3)
42. Given good=[1,2], movie=[2,1] and bad=[-1,-2], calculate the average embedding for “good movie bad”. (06 Marks | L3 | CO3)
43. A vocabulary contains 10,000 words and each embedding has 100 dimensions. Determine the dimensions of the embedding matrix and explain how a document can be represented using it. (06 Marks | L3 | CO3)

I. Training Neural Networks

44. Explain the major steps involved in training a feedforward neural network for NLP classification. (08 Marks | L2 | CO3)
45. Explain the purpose of a loss function in neural-network training and how it indicates prediction error. (06 Marks | L2 | CO3)
46. Explain why backpropagation is required to train a multilayer feedforward neural network. Describe how the error signal is propagated. (08 Marks | L2 | CO3)
47. A neural network predicts [0.7,0.2,0.1] for a sample whose true class is the first class. Identify the predicted class and state whether the prediction is correct. (06 Marks | L3 | CO3)

J. Integrated / Application-Oriented Questions

48. A document-classification task uses pretrained Word2Vec embeddings and a feedforward neural network. Describe the workflow from text preprocessing to class prediction and evaluation. (08 Marks | L3 | CO3)
49. For A=[1,2,2] and B=[2,1,2], calculate cosine similarity and explain how it supports word-similarity analysis. (08 Marks | L3 | CO2)
50. A developer constructs word vectors using a co-occurrence matrix and uses cosine similarity to find related words. Explain the process and identify one limitation of count-based representations. (08 Marks | L3 | CO2)
51. A sentiment system represents documents using word embeddings and classifies them as positive, negative or neutral. Explain the network architecture and role of softmax. (08 Marks | L3 | CO3)
52. Compare count-based embeddings and Word2Vec with respect to how vectors are obtained, representation characteristics and learning approach. (08 Marks | L2 | CO3)
53. An embedding model gives high cosine similarity to “car” and “automobile”, and also to “car” and “road”. Explain how similarity and relatedness differ and why this matters in evaluation. (06 Marks | L3 | CO3)