

Natural Language Processing (AI615)

Prepared by Dr. Ashraf Elsayed

Lecture 10





مجموعة جامعة
الجميع الذكية
للتعليم القابضة

MIDOCEAN
UNIVERSITY

Neural Language Models & Part of Speech Tagging

Lecture 10

Prepared by
Dr. Ashraf Elsayed

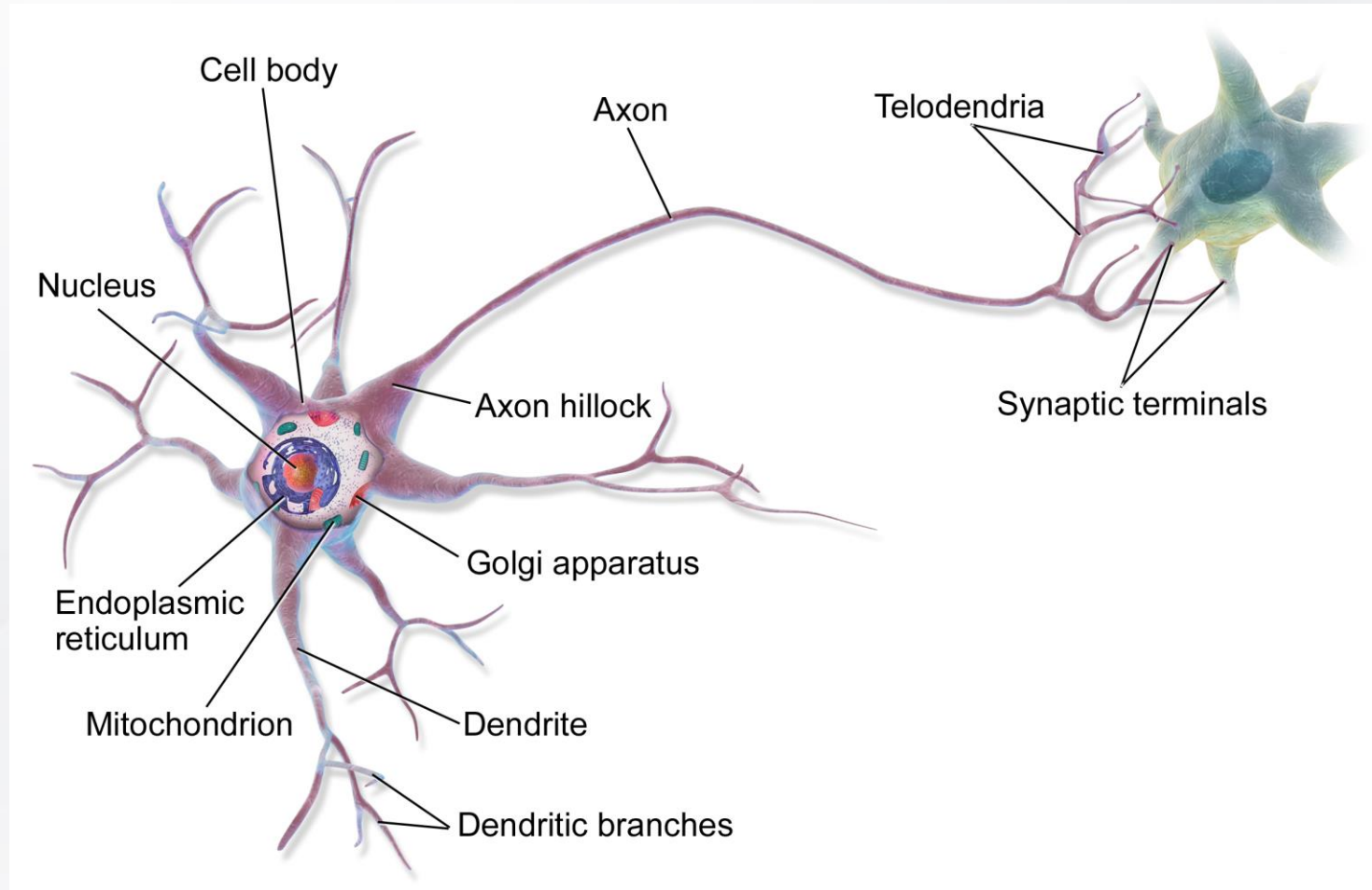
www.midocean.ae

The objectives of this lecture is to:

- Understand Neural Language Models
- Define and understand Feedforward Neural Networks
- Understand Parts of Speech (POS)
- Define and Explain Named Entity Recognition (NER)

Neural Language Models

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

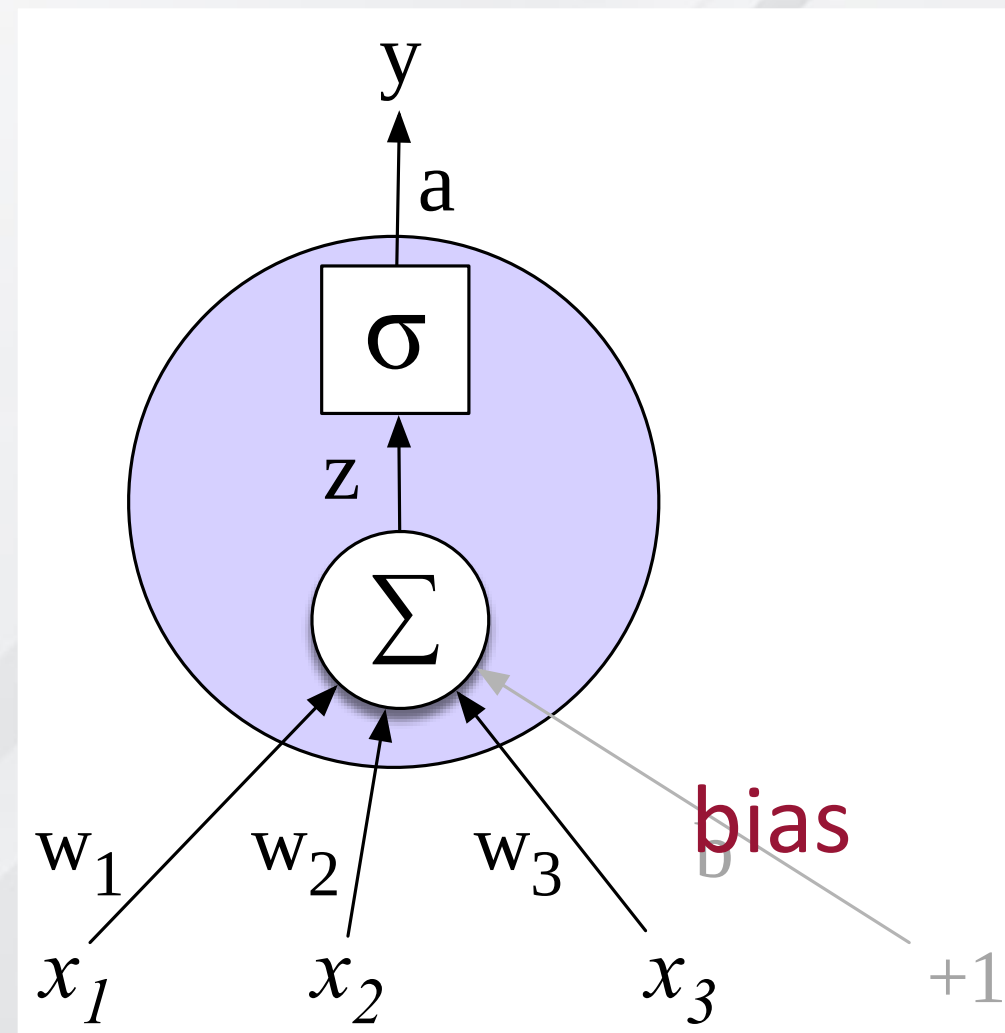
Output value

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 :

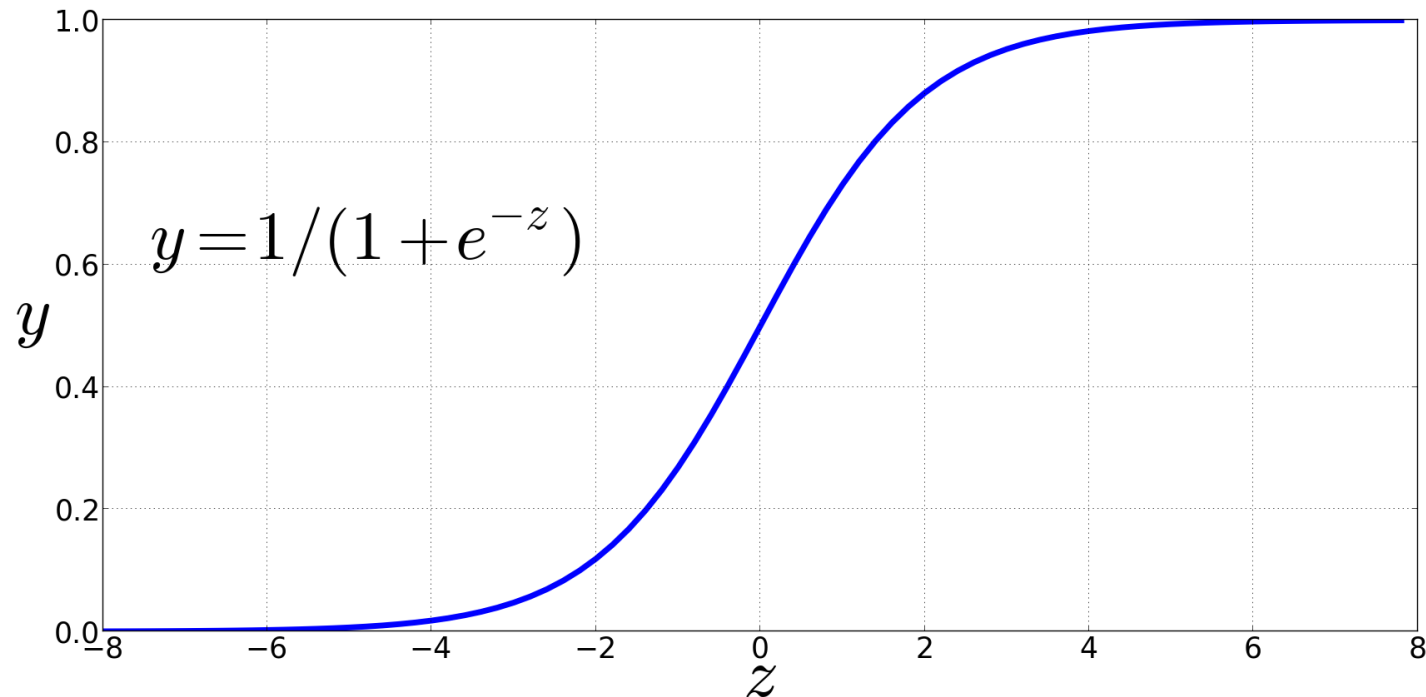
$$y = a = f(z)$$

Non-Linear Activation Functions

We've already seen the sigmoid for logistic regression:

Sigmoid

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



Final function the unit is computing

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

Final unit again

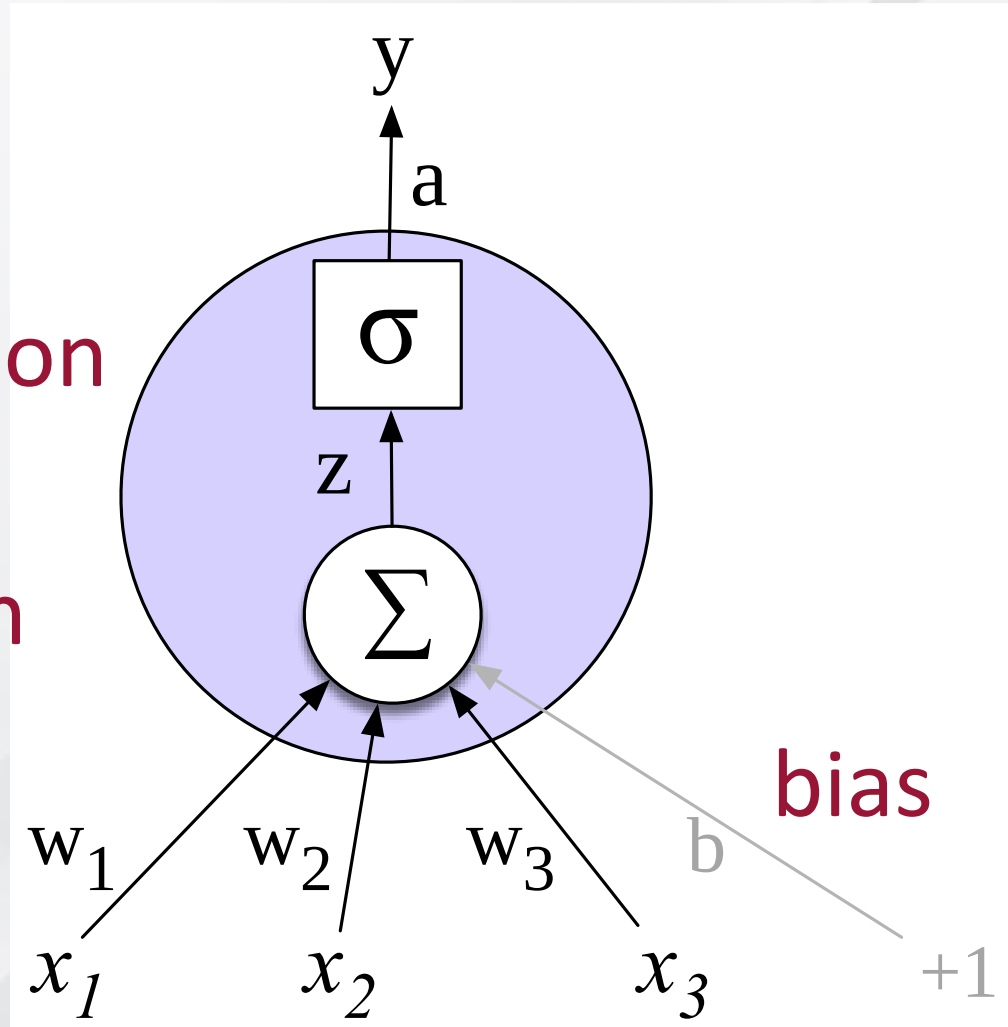
Output value

Non-linear activation function

Weighted sum

Weights

Input layer



An example

- *Suppose a unit has:*
- $w = [0.2, 0.3, 0.9]$
- $b = 0.5$
- What happens with input x :
- $x = [0.5, 0.6, 0.1]$

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

An example

- *Suppose a unit has:*

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

- $b = 0.5$

- What happens with the following input x ?

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

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

An example

- Suppose a unit has:
 - $w = [0.2, 0.3, 0.9]$
 - $b = 0.5$
- What happens with input x :
 - $x = [0.5, 0.6, 0.1]$

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

An example

- Suppose a unit has:
 - $w = [0.2, 0.3, 0.9]$
 - $b = 0.5$
- What happens with input x :
- $x = [0.5, 0.6, 0.1]$

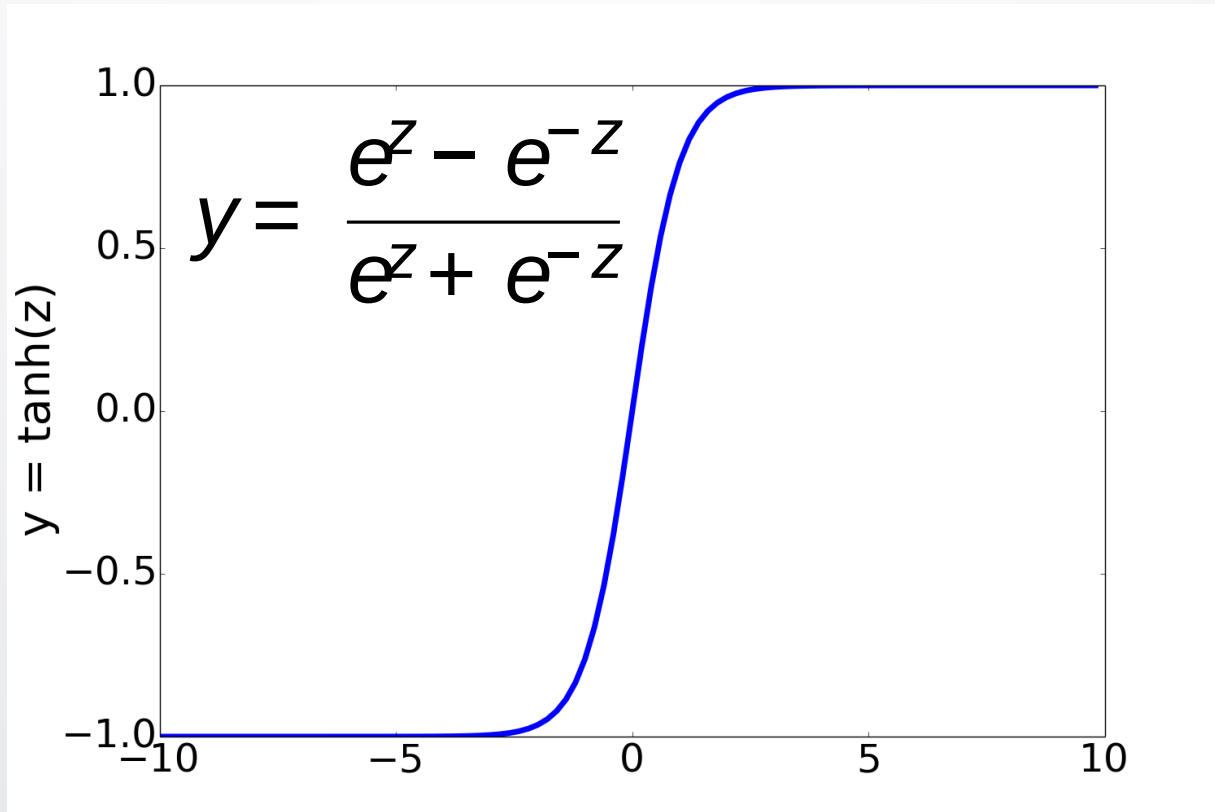
$$y = s(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}} =$$

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

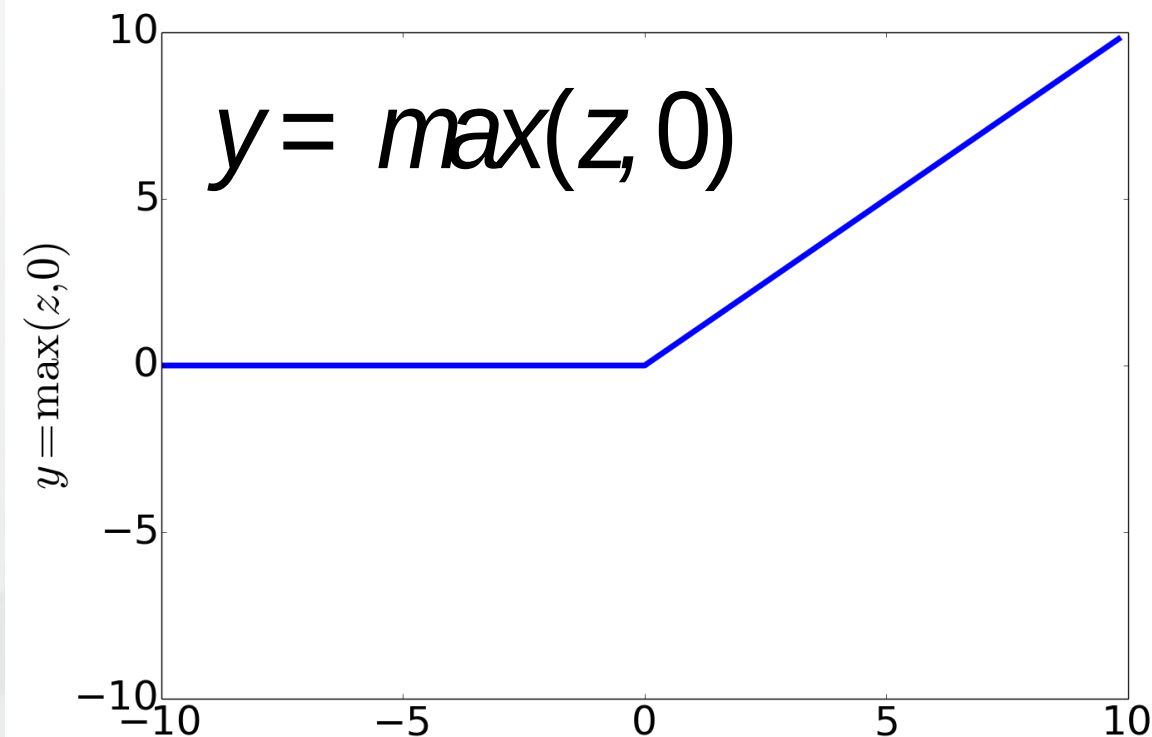
www.midocean.ae

Non-Linear Activation Functions besides sigmoid

Most Common:



tanh

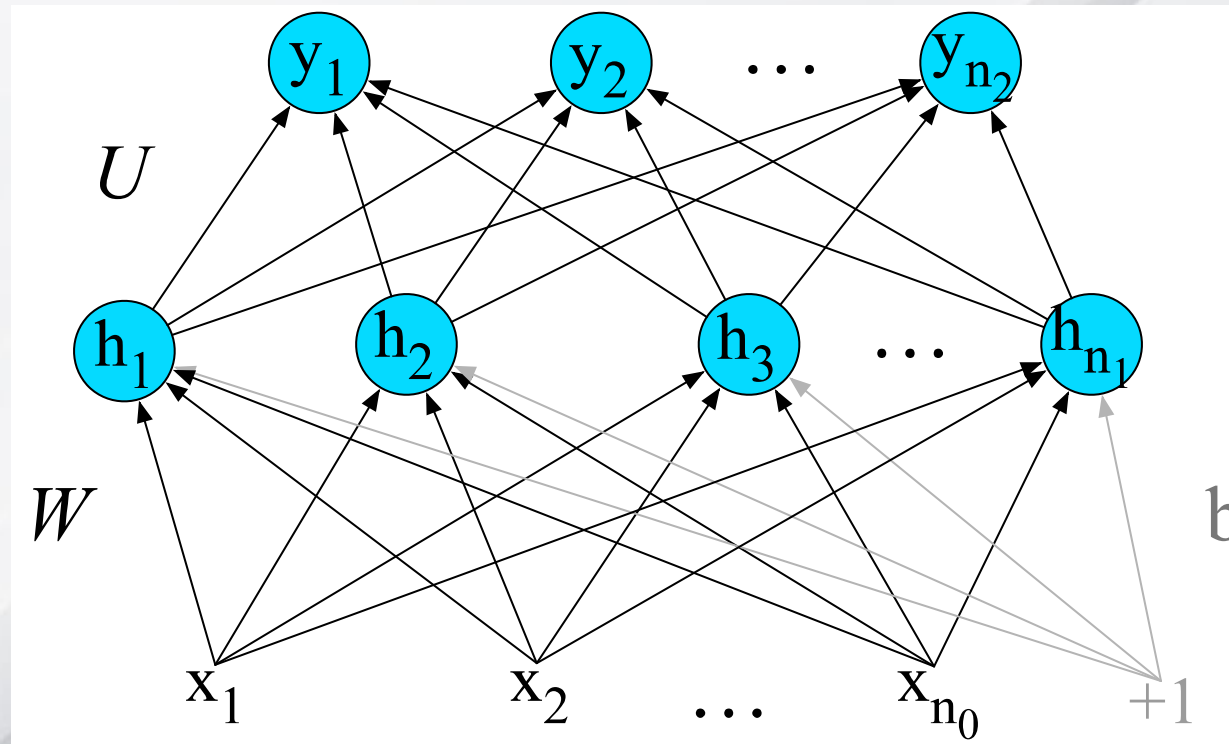


ReLU

Rectified Linear Unit

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)

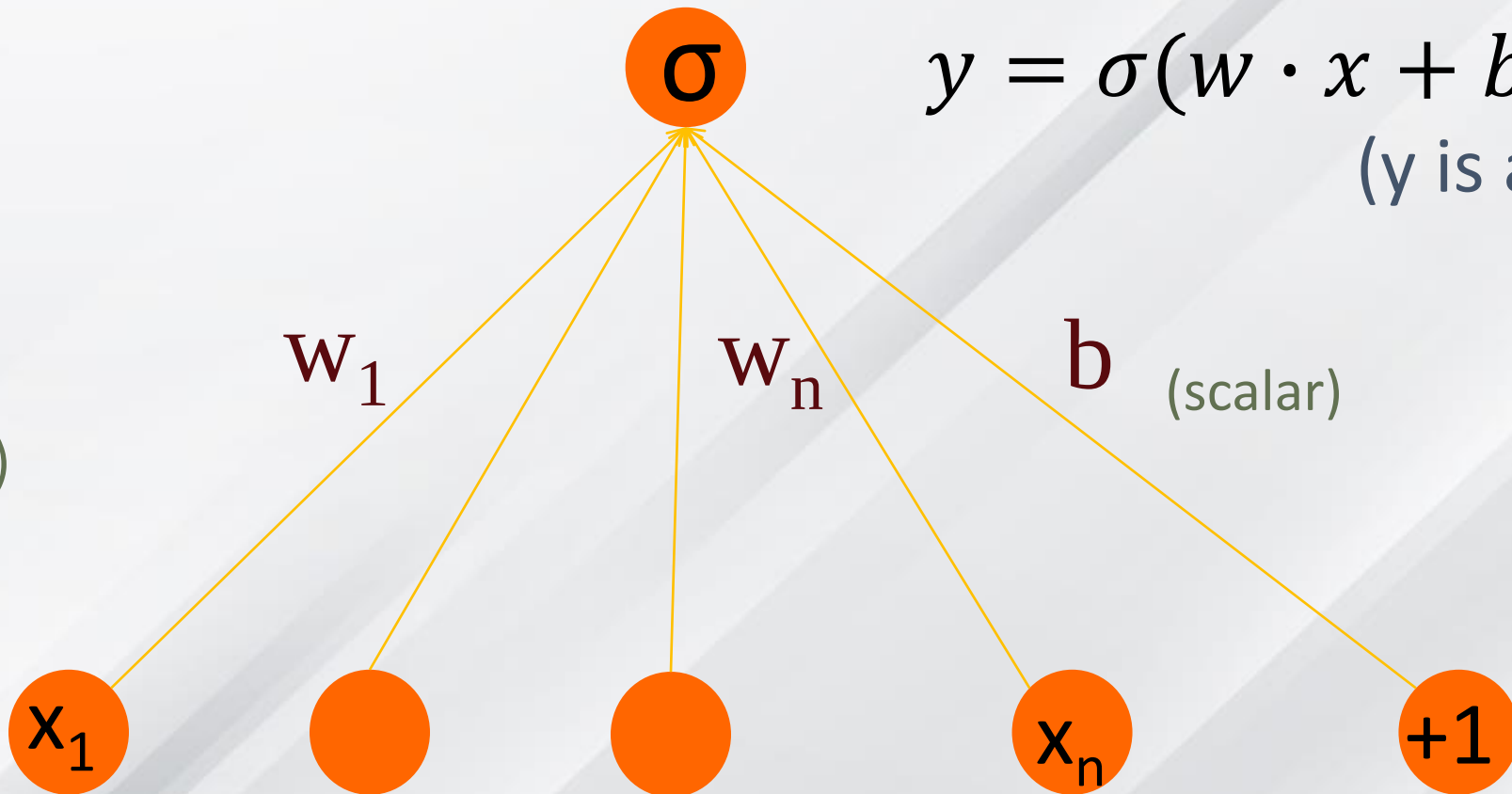
W
(vector)

w_1

w_n

b (scalar)

Input layer
vector x



Multinomial Logistic Regression as a 1-layer Network

Fully connected single layer network

Output layer
(softmax nodes)

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

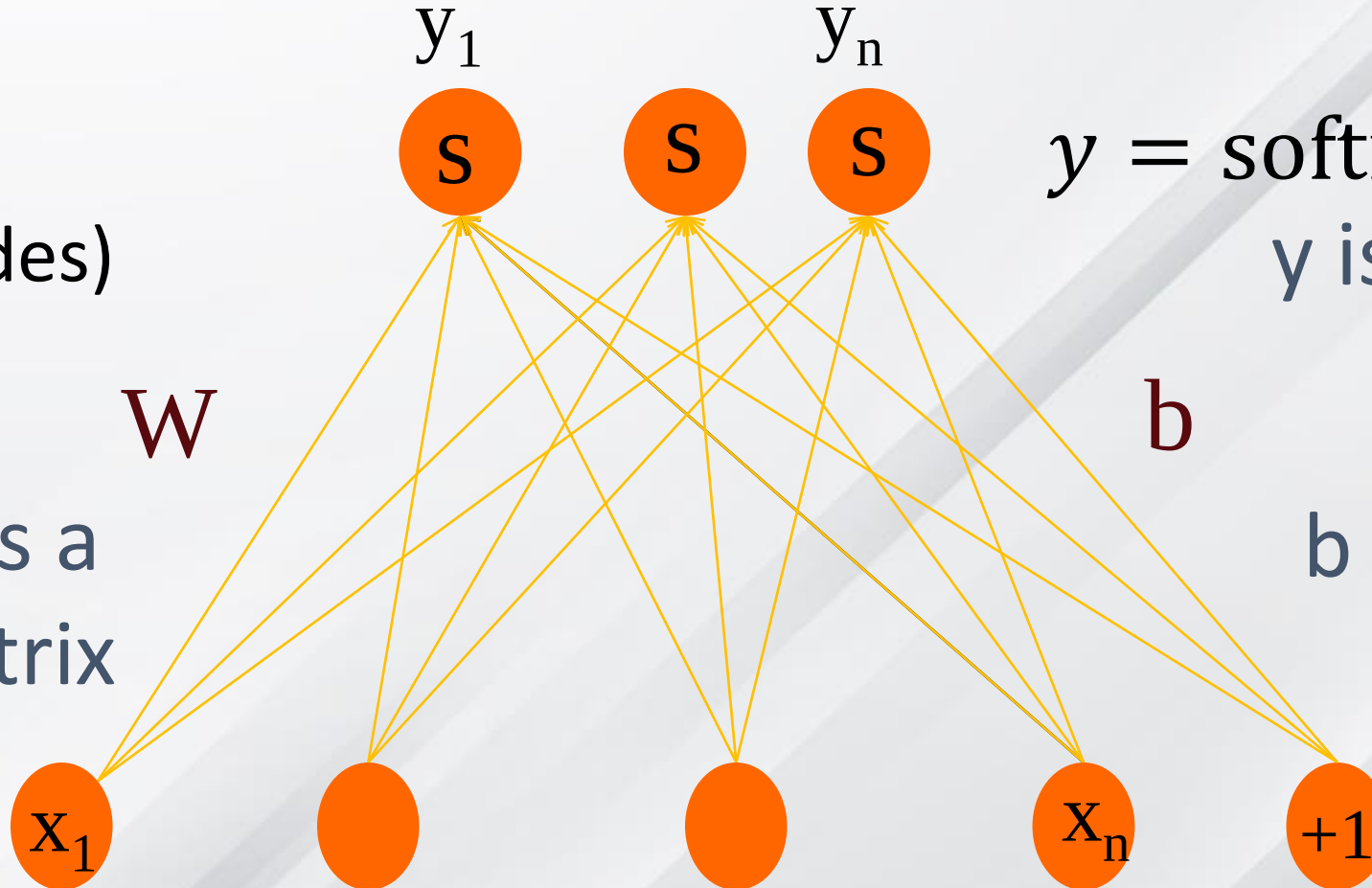
y is a vector

W is a
matrix

b

b is a vector

Input layer
scalars



Reminder: softmax: a generalization of sigmoid

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

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

- Example:

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

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

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

Two-Layer Network with scalar output

Output layer
(σ node)

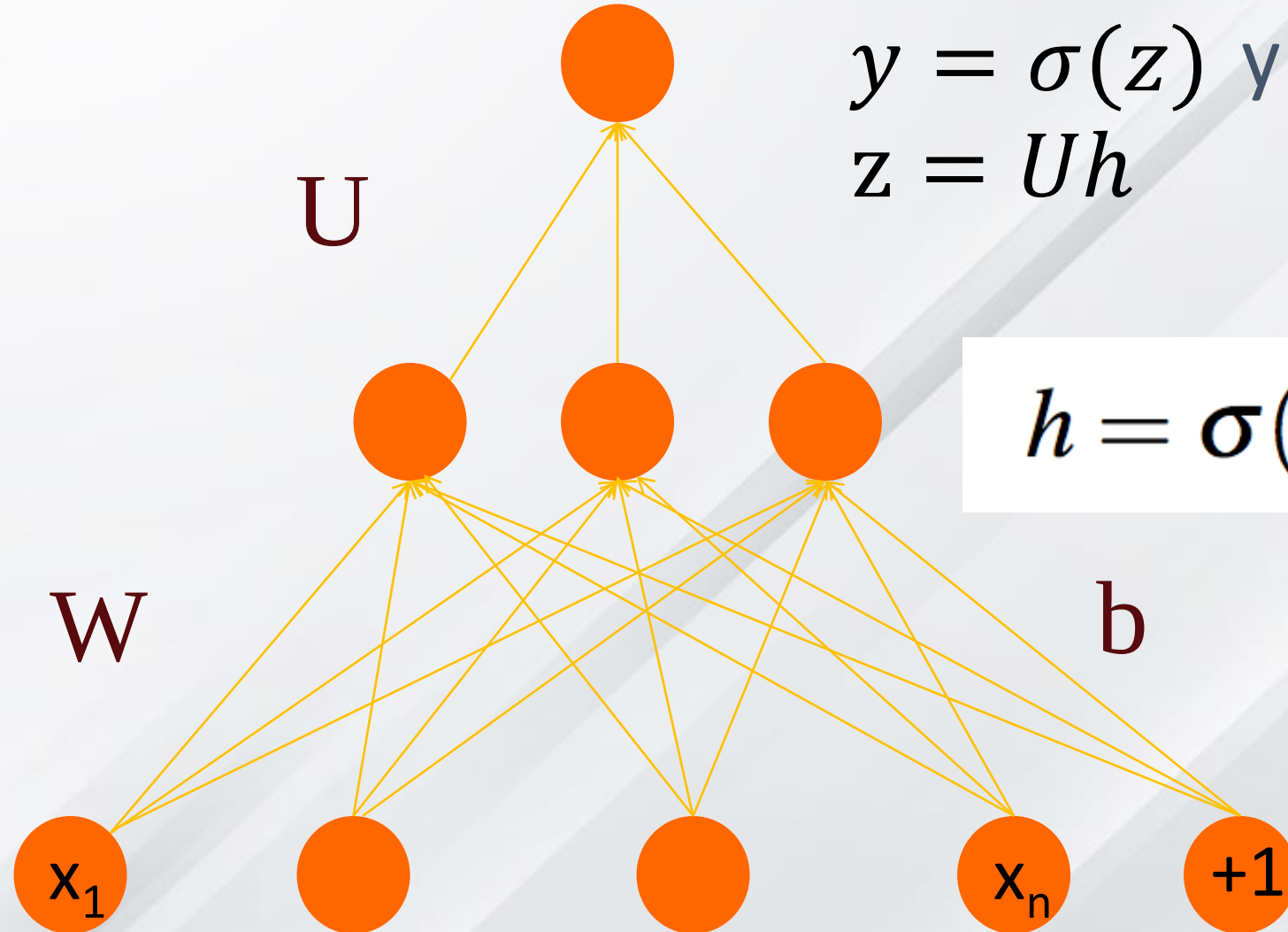
$$y = \sigma(z) \quad y \text{ is a scalar}$$
$$z = Uh$$

hidden units
(σ node)

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

Could be ReLU
Or tanh

Input layer
(vector)



Two-Layer Network with scalar output

Output layer
(σ node)

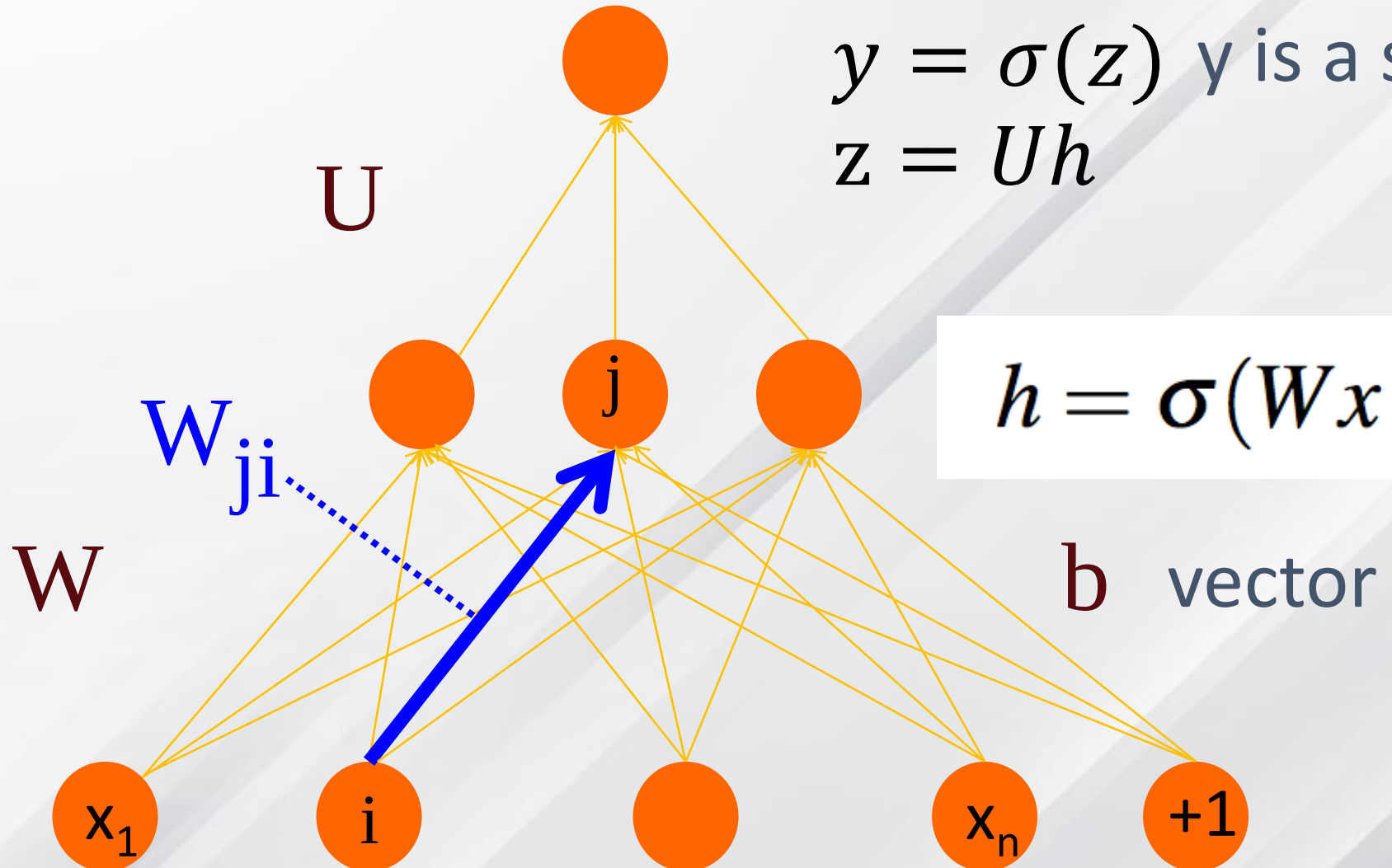
$$y = \sigma(z) \quad y \text{ is a scalar}$$

$$z = Uh$$

hidden units
(σ node)

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

Input layer
(vector)



Two-Layer Network with scalar output

Output layer
(σ node)

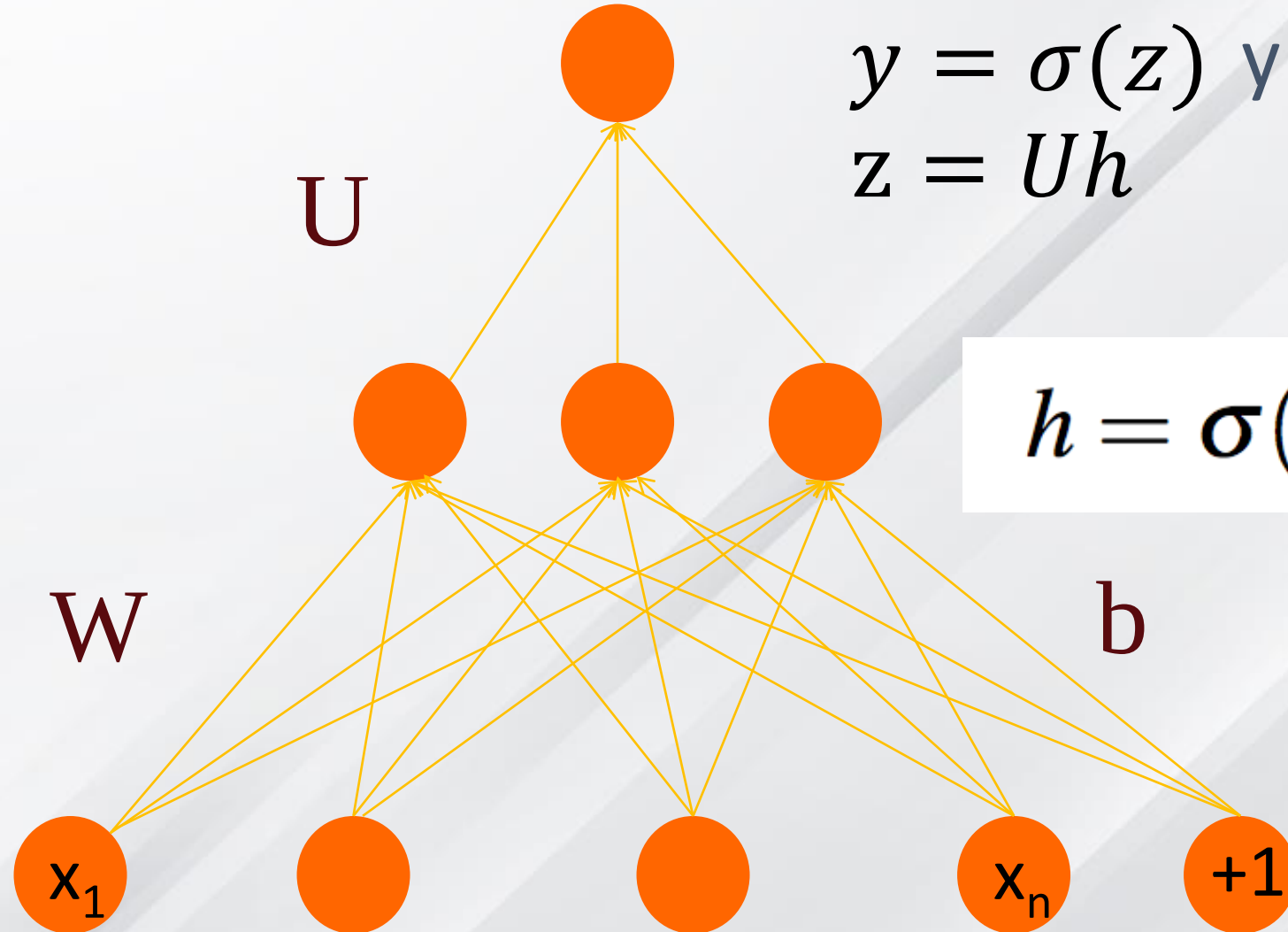
$$y = \sigma(z) \quad y \text{ is a scalar}$$
$$z = Uh$$

hidden units
(σ node)

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

Could be ReLU
Or tanh

Input layer
(vector)

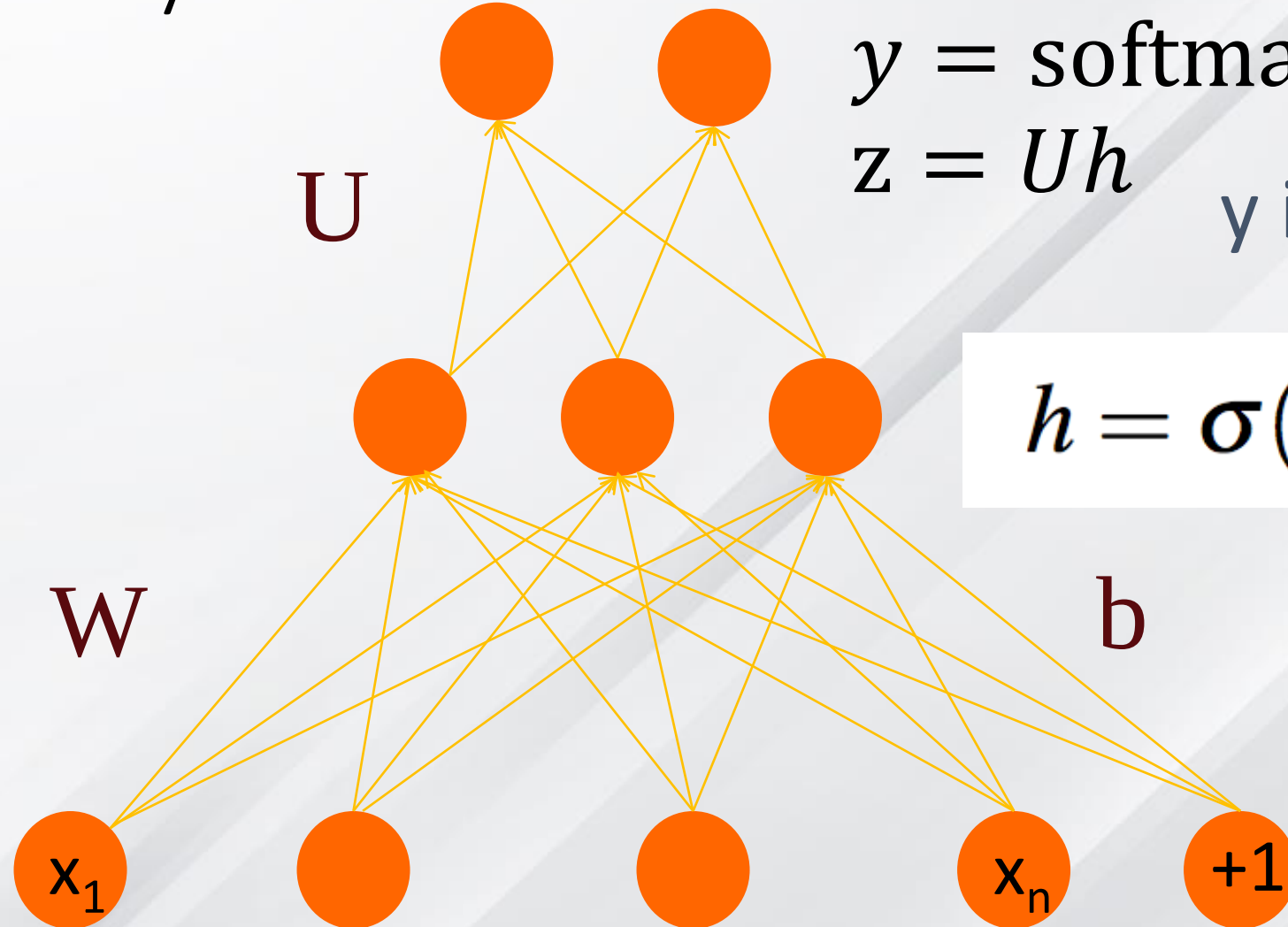


Two-Layer Network with softmax output

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

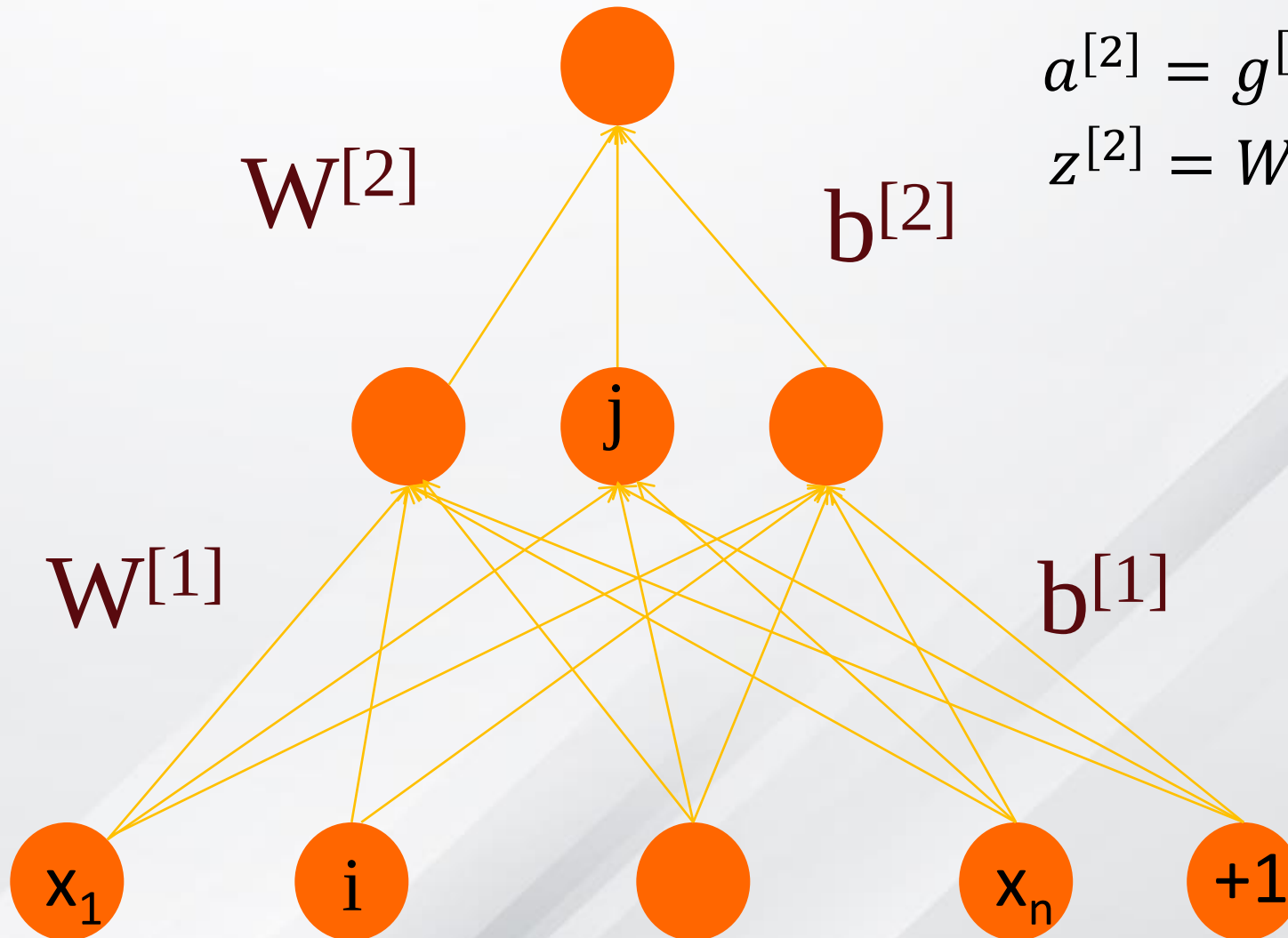
$$y = a^{[2]}$$

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

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

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

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



Multi Layer Notation

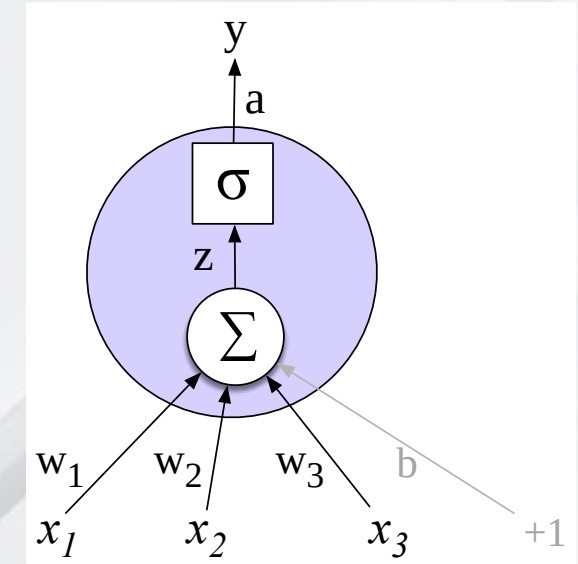
$$z^{[1]} = W^{[1]}a^{[0]} + b^{[1]}$$

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

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

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

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



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:

$$x = x_1, x_2, \dots, x_{n_0}$$

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

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

We'll do this:

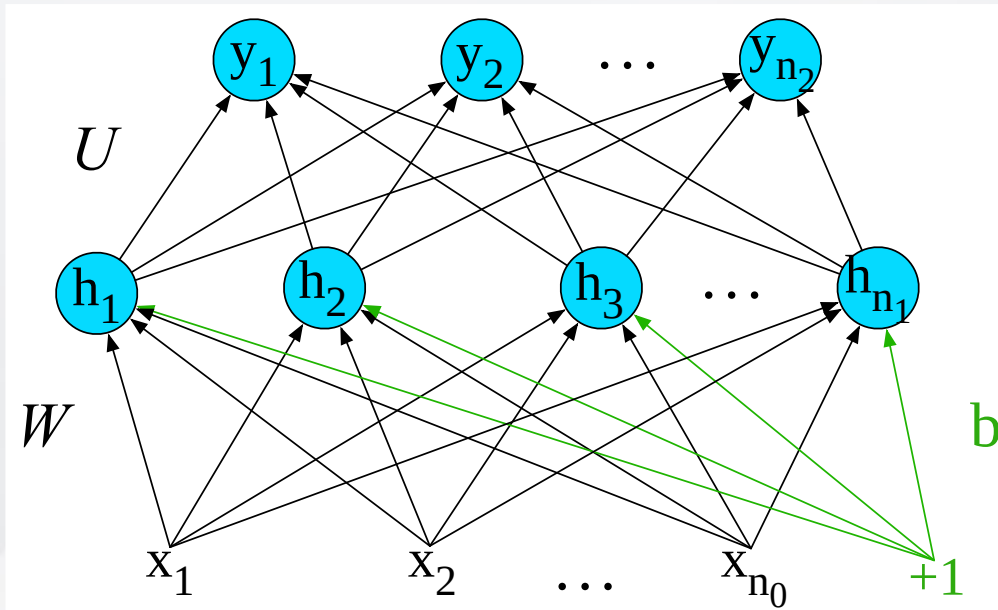
$$x = x_0, x_1, x_2, \dots, x_{n_0}$$

$$h = \sigma(Wx)$$

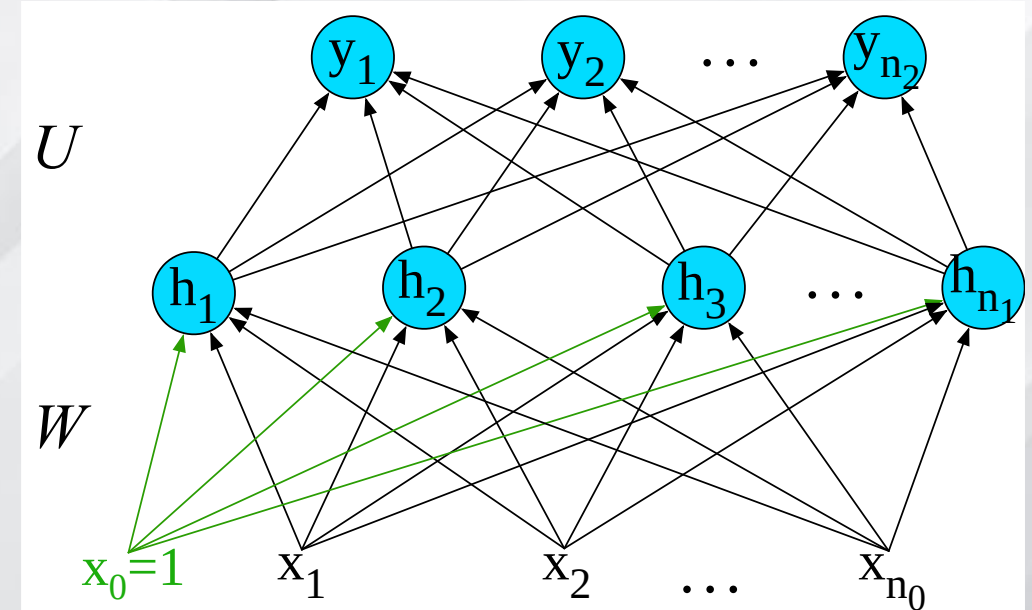
$$\sigma \left(\sum_{i=0}^{n_0} W_{ji} x_i \right)$$

Replacing the bias unit

Instead of:



We'll do this:

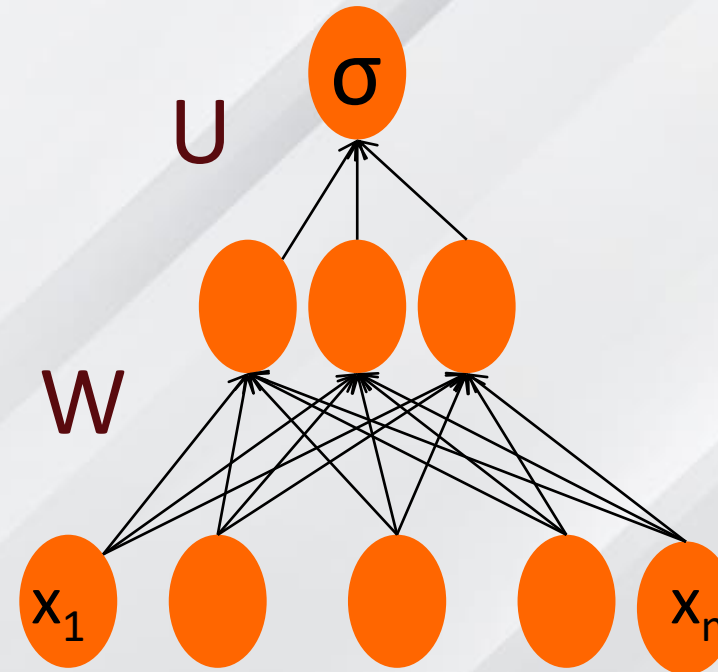


Use cases for feedforward networks

- 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

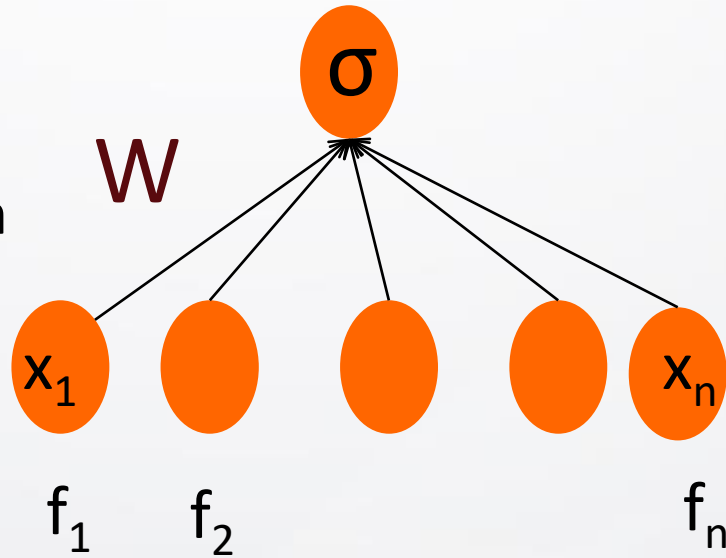


Sentiment Features

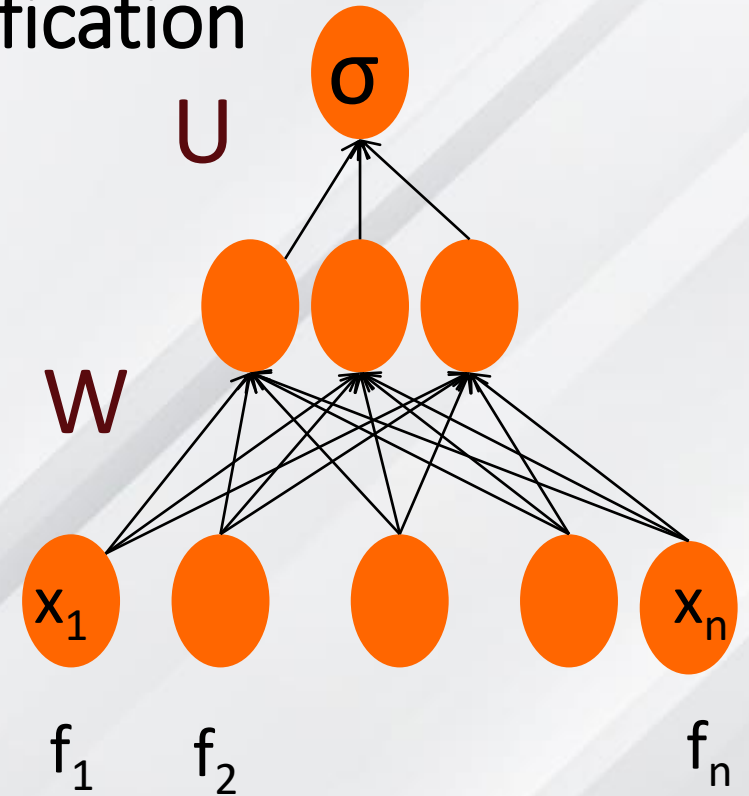
Var	Definition
x_1	$\text{count}(\text{positive lexicon}) \in \text{doc}$
x_2	$\text{count}(\text{negative lexicon}) \in \text{doc}$
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_4	$\text{count}(\text{1st and 2nd pronouns}) \in \text{doc}$
x_5	$\begin{cases}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

Feedforward nets for simple classification

Logistic
Regression



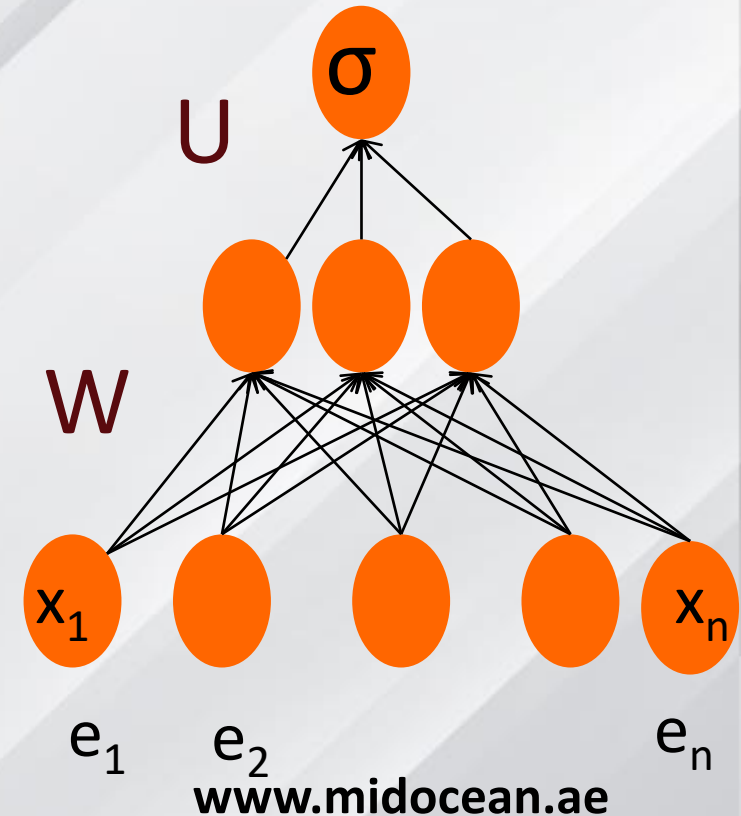
2-layer
feedforward
network



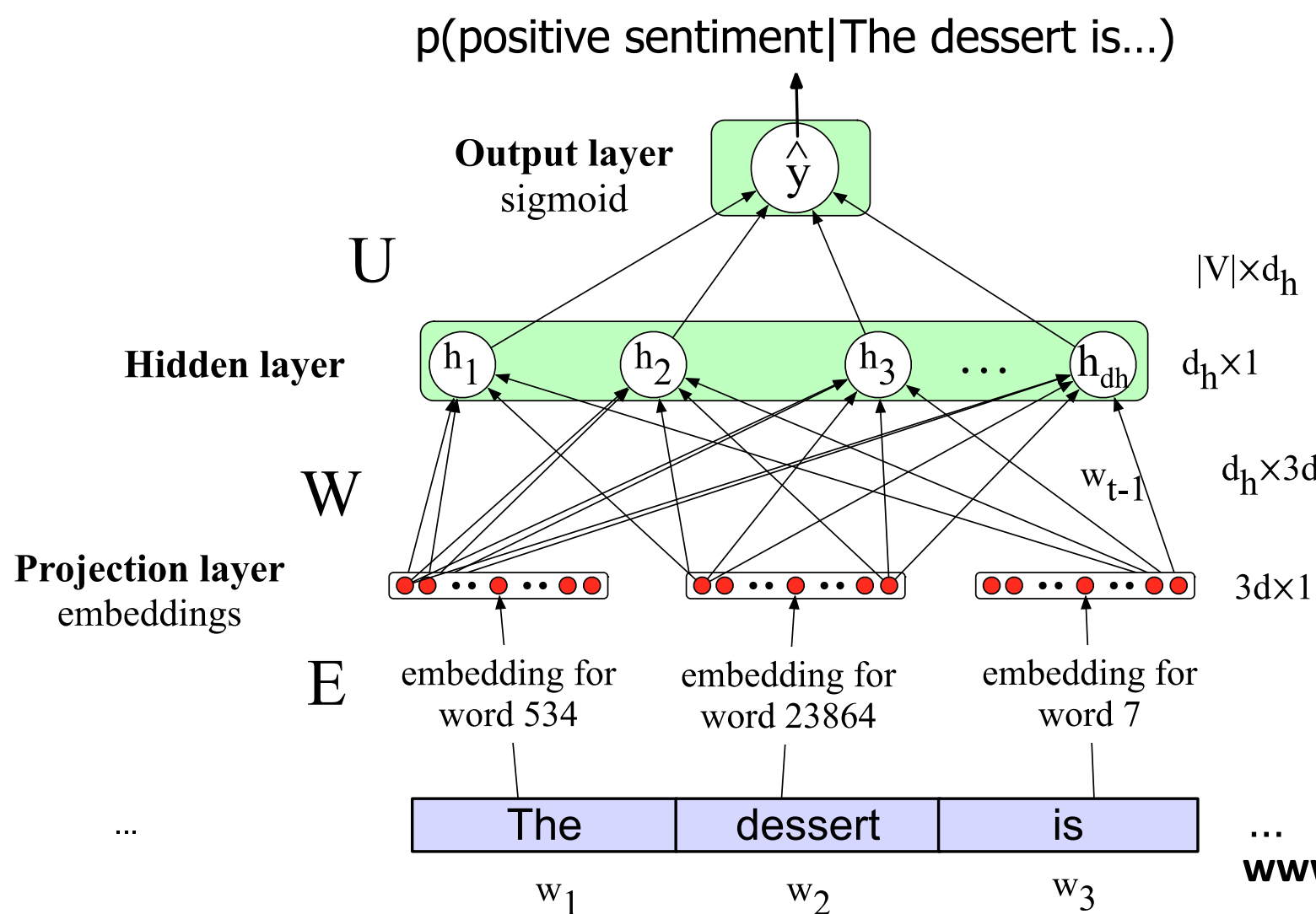
- Just adding a hidden layer to logistic regression
- allows the network to use non-linear interactions between features
- which may (or may not) improve performance.

Even better: representation learning

- The real power of deep learning comes from the ability to **learn** features from the data
- Instead of using hand-built human-engineered features for classification
- Use learned representations like embeddings!

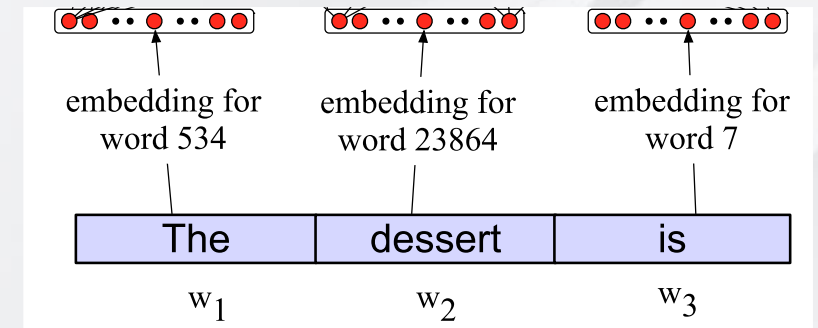


Neural Net Classification with embeddings as input features!



Issue: texts come in different sizes

- This assumes a fixed size length (3)!
- Kind of unrealistic.
- Some simple solutions (more sophisticated solutions later)

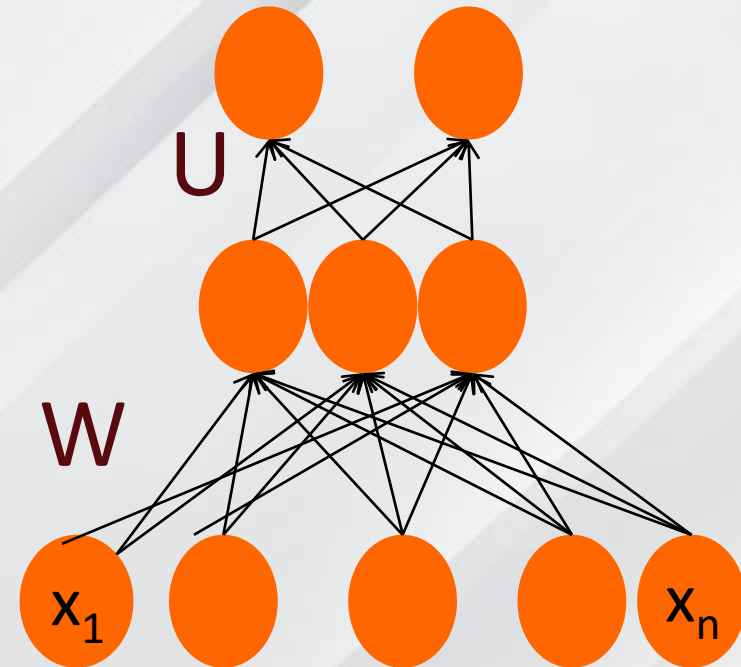


1. Make the input the length of the longest review
 - If shorter then pad with zero embeddings
 - Truncate if you get longer reviews at test time
2. Create a single "sentence embedding" (the same dimensionality as a word) to represent all the words
 - Take the mean of all the word embeddings
 - Take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

Reminder: Multiclass Outputs

- What if you have more than two output classes?
 - Add more output units (one for each class)
 - And use a “softmax layer”

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



Neural Language Models (LMs)

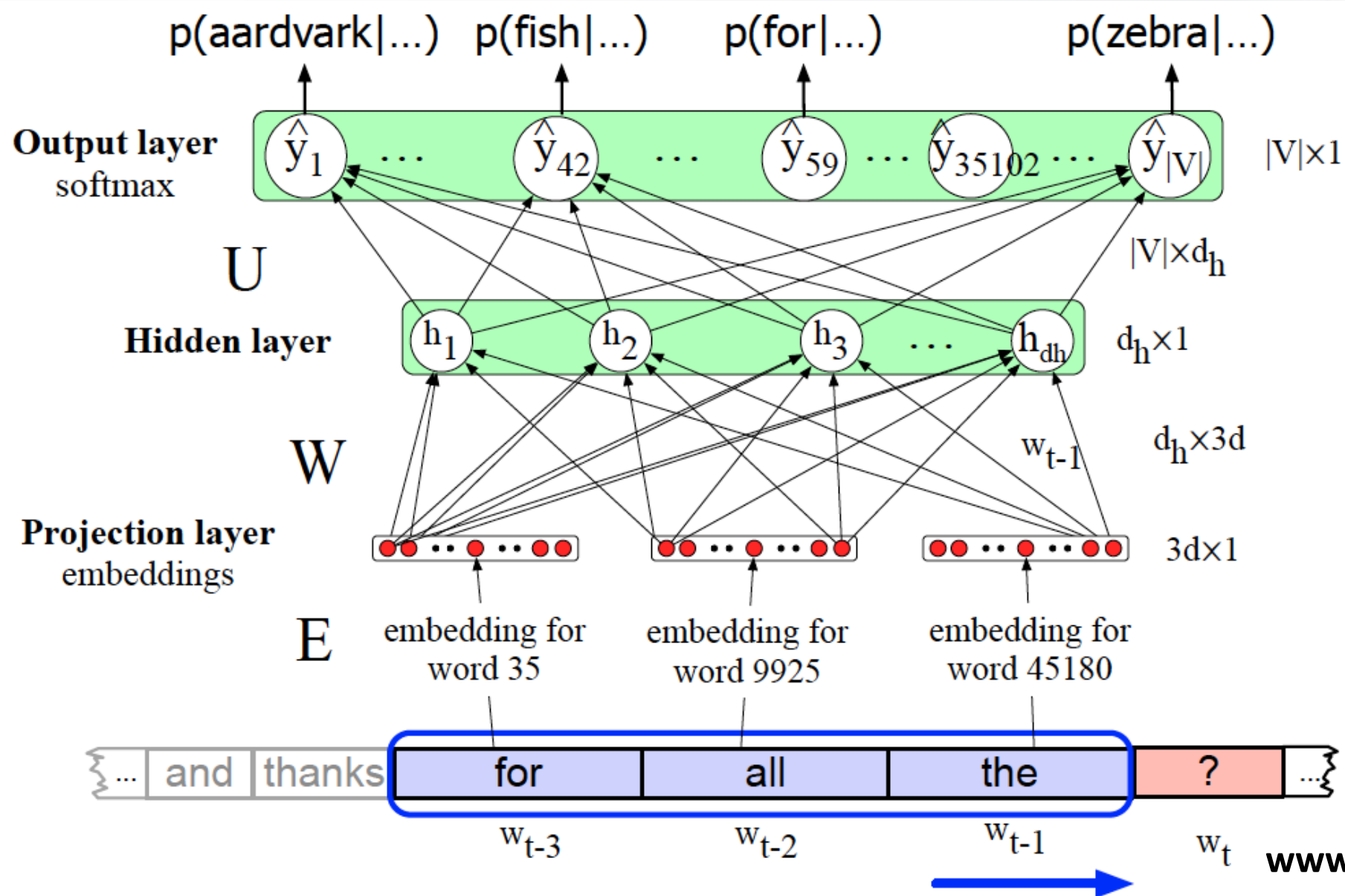
- **Language Modeling:** Calculating the probability of the next word in a sequence given some history.
- We've seen N-gram based LMs
- But neural network LMs far outperform n-gram language models
- State-of-the-art neural LMs are based on more powerful neural network technology like Transformers
- But **simple feedforward LMs** can do almost as well!

Simple feedforward Neural Language Models

- **Task:** predict next word w_t
- given prior words $w_{t-1}, w_{t-2}, w_{t-3}, \dots$
- **Problem:** Now we're dealing with sequences of arbitrary length.
- **Solution:** Sliding windows (of fixed length)

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

Neural Language Model

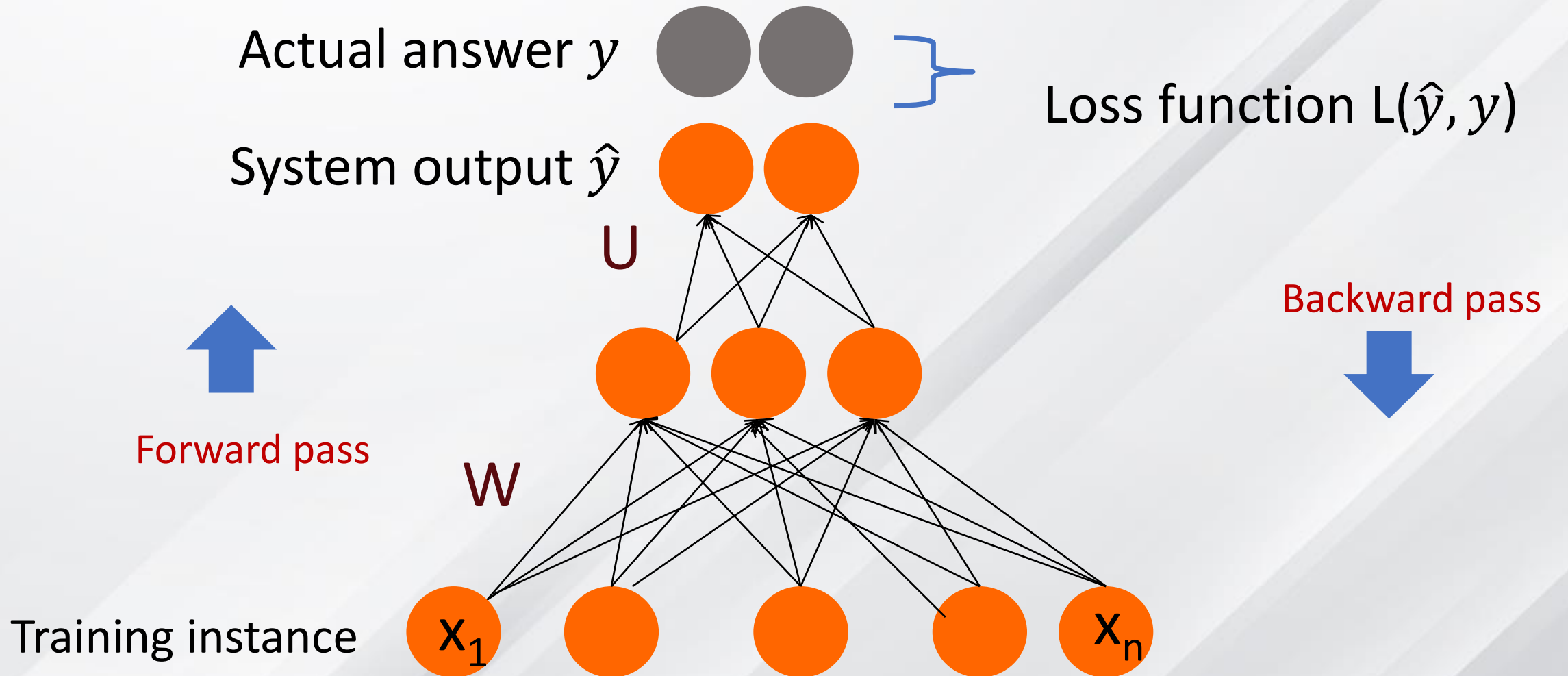


Why Neural LMs work better than N-gram LMs

- **Training data:**
 - We've seen: I have to make sure that the cat gets fed.
 - Never seen: dog gets fed
- **Test data:**
 - I forgot to make sure that the dog gets ____
 - N-gram LM can't predict "fed"!
 - Neural LM can use similarity of "cat" and "dog" embeddings to generalize and predict "fed" after dog

Training Neural Nets: Overview

Intuition: training a 2-layer Network



Intuition: Training a 2-layer network

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

Reminder: Loss Function for binary logistic regression

- A measure for how far off the current answer is to the right answer
- Cross entropy loss for logistic regression:

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

Reminder: gradient descent for weight updates

- Use the derivative of the loss function with respect to weights

$$\frac{d}{dw} L(f(x; w), y)$$

- To tell us how to adjust weights for each training item

- Move them in the opposite direction of the gradient

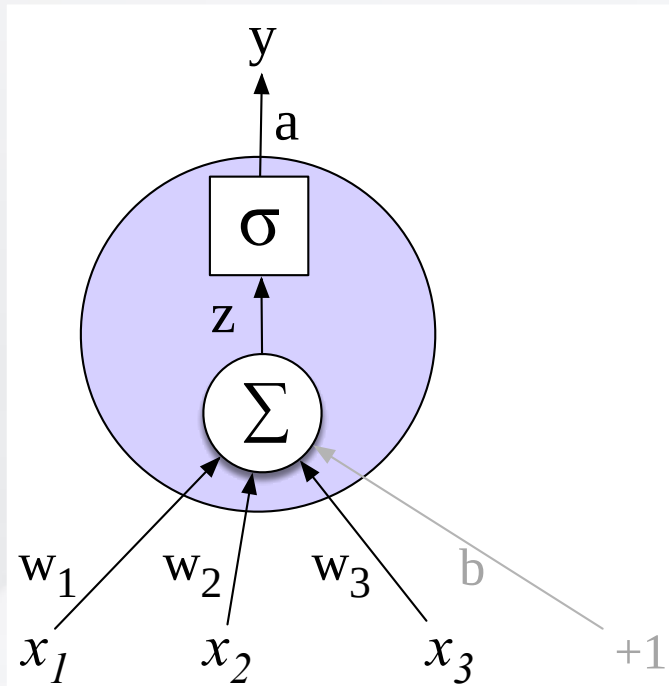
- For logistic regression

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

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

Where did that derivative come from?

- Using the chain rule! $f(x) = u(v(x))$
$$\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx}$$



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

Computation Graphs and Backward Differentiation

Why Computation Graphs

- For training, we need the derivative of the loss with respect to each weight in every layer of the network
- But the loss is computed only at the very end of the network!
- Solution: **error backpropagation** (Rumelhart, Hinton, Williams, 1986)
- **Backprop** is a special case of **backward differentiation**
- Which relies on **computation graphs**.

Computation Graphs

A computation graph represents the process of computing a mathematical expression

Example:

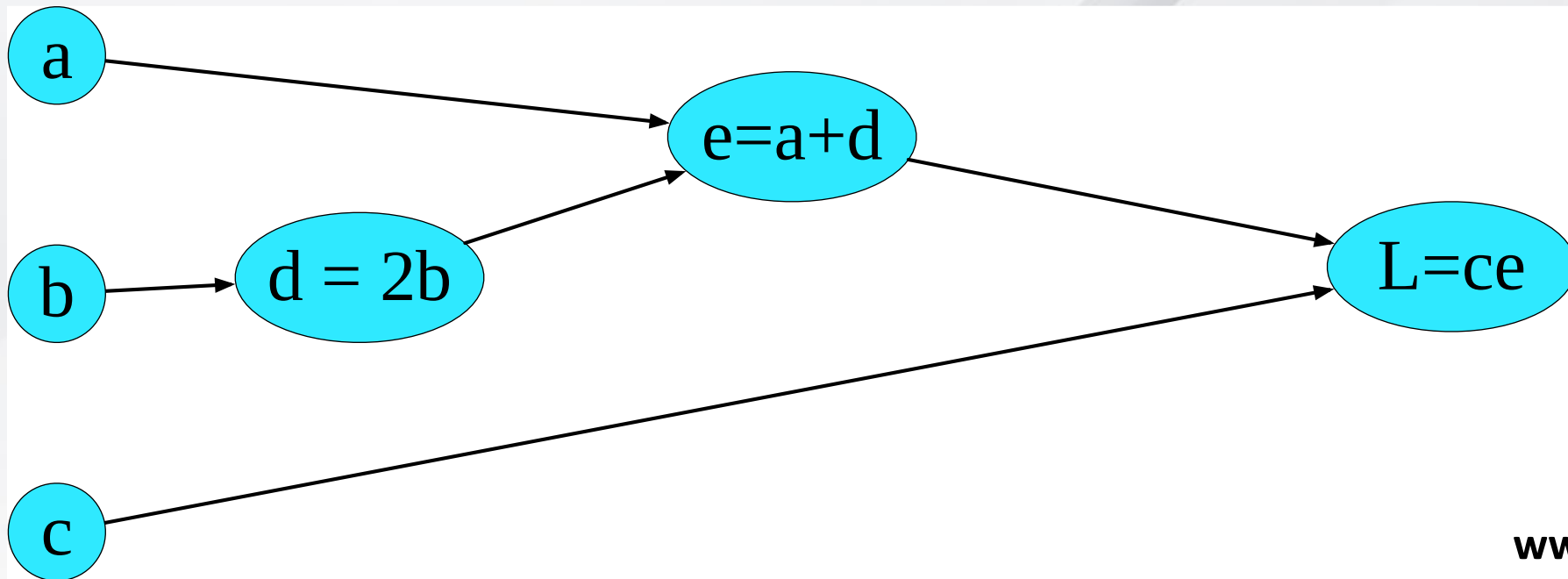
$$L(a, b, c) = c(a + 2b)$$

$$d = 2 * b$$

$$e = a + d$$

$$L = c * e$$

Computations:



Example:

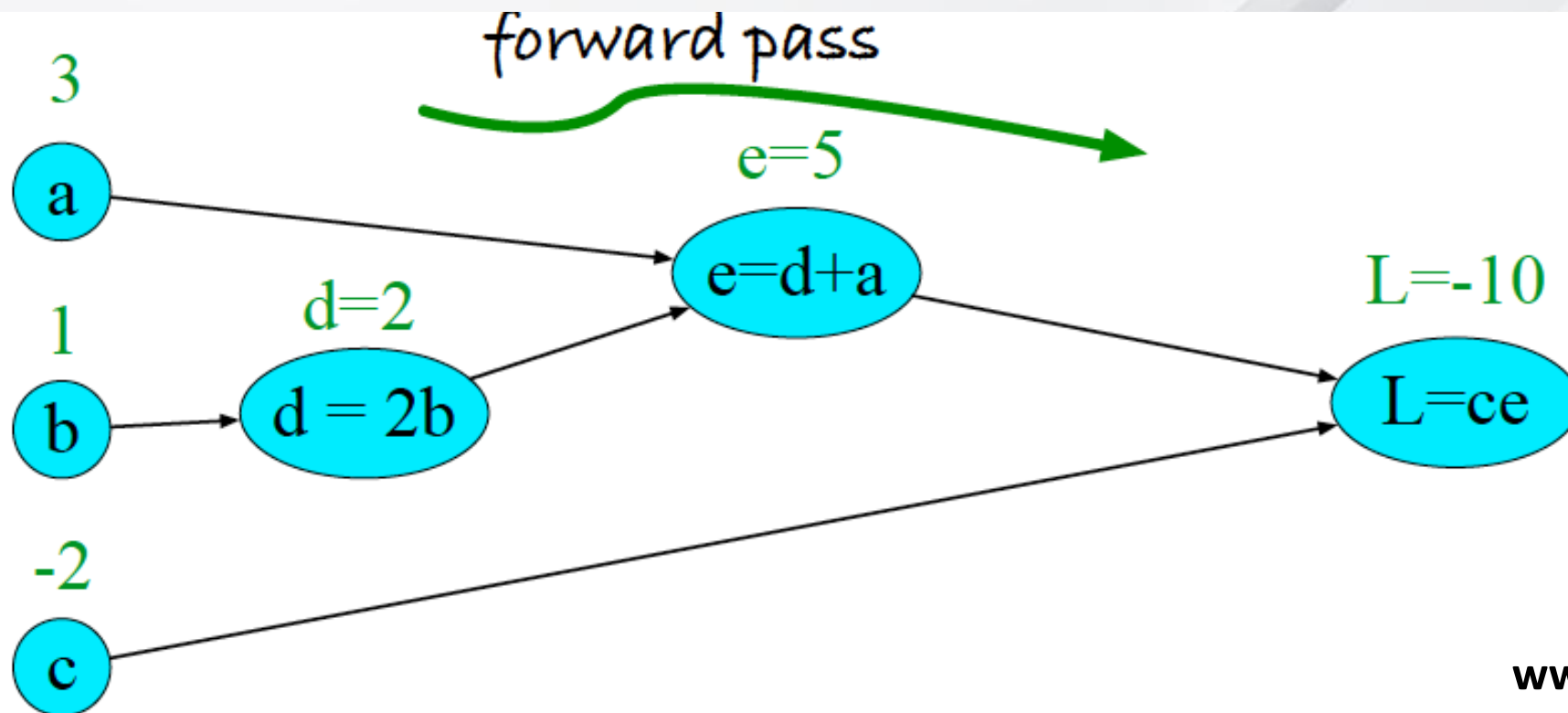
$$L(a, b, c) = c(a + 2b)$$

$$d = 2 * b$$

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

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

$$d = 2 * b$$

$$e = a + d$$

$$L = c * e$$

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

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

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

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

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

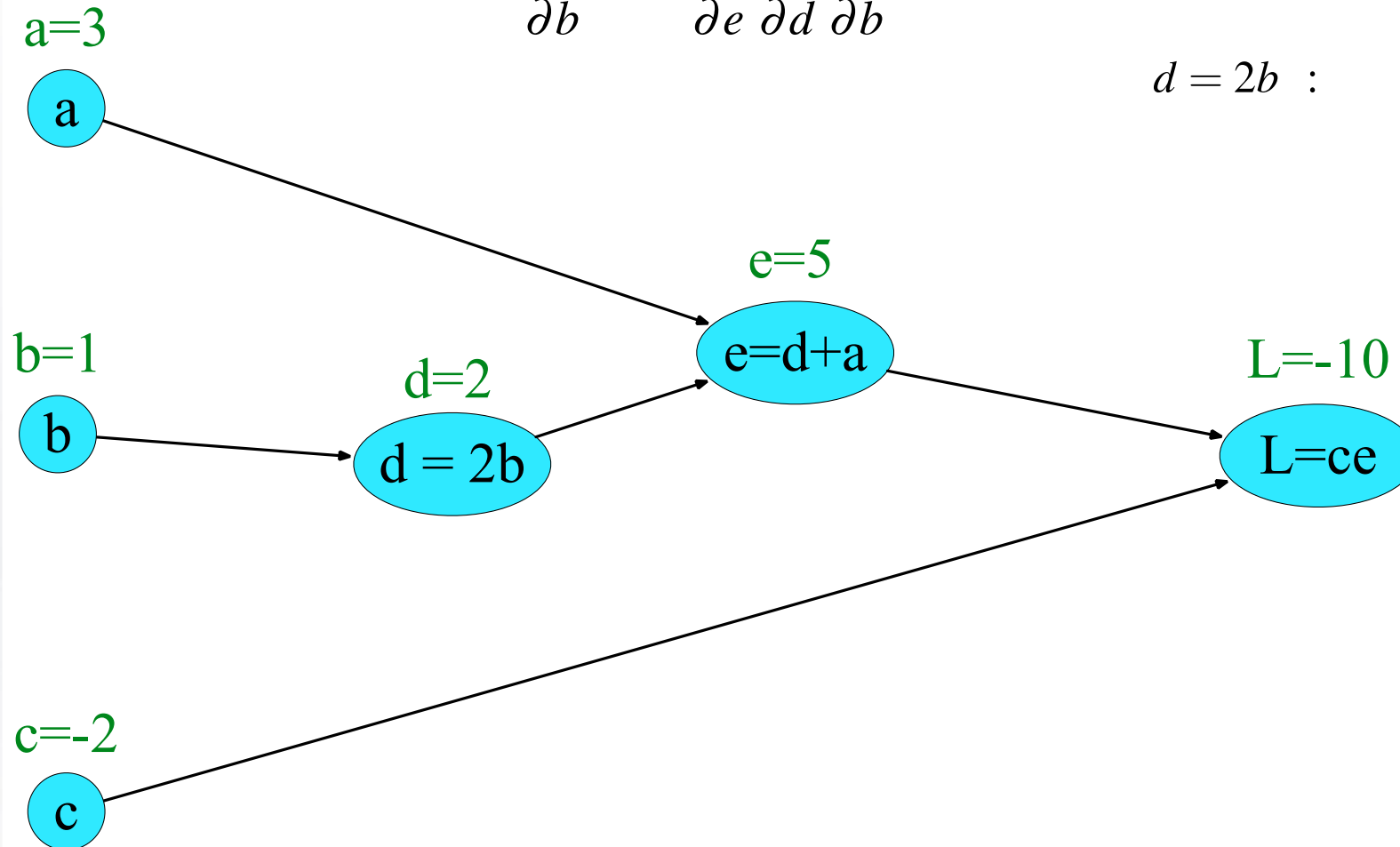
$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e} \frac{\partial e}{\partial a}$$

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

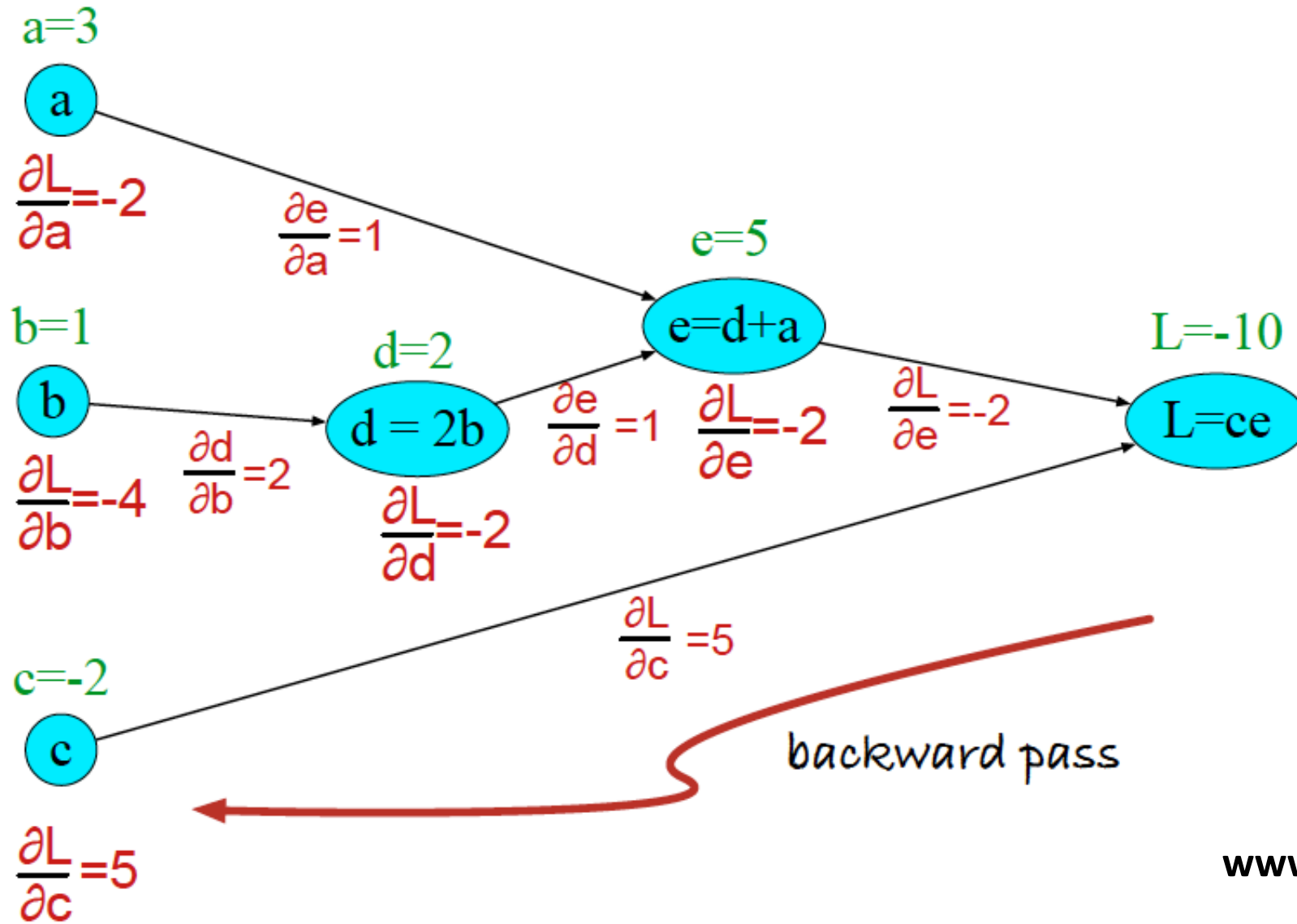
$$L = ce : \quad \frac{\partial L}{\partial e} = c, \quad \frac{\partial L}{\partial c} = e$$

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

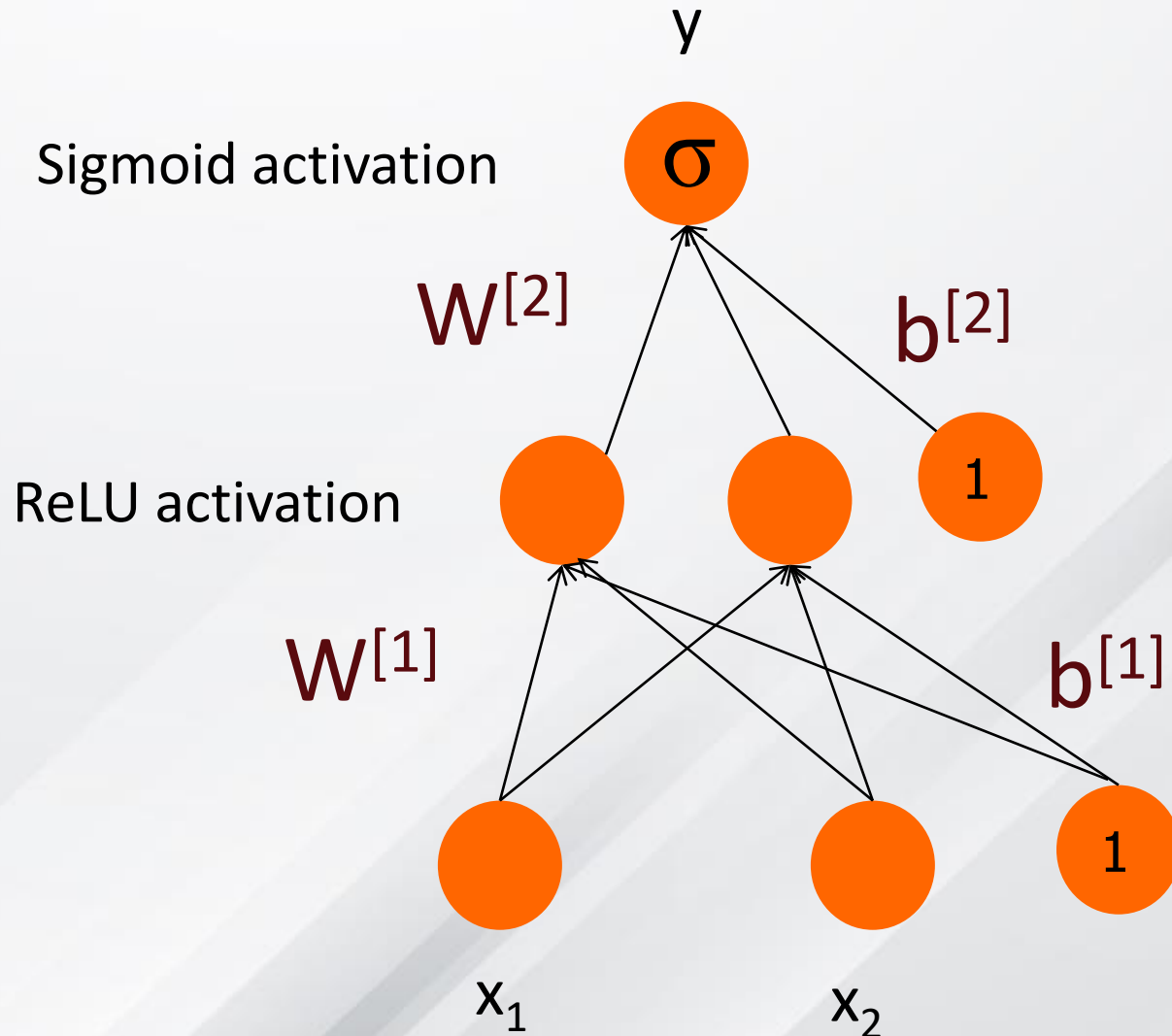
$$d = 2b : \quad \frac{\partial d}{\partial b} = 2$$



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

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

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

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

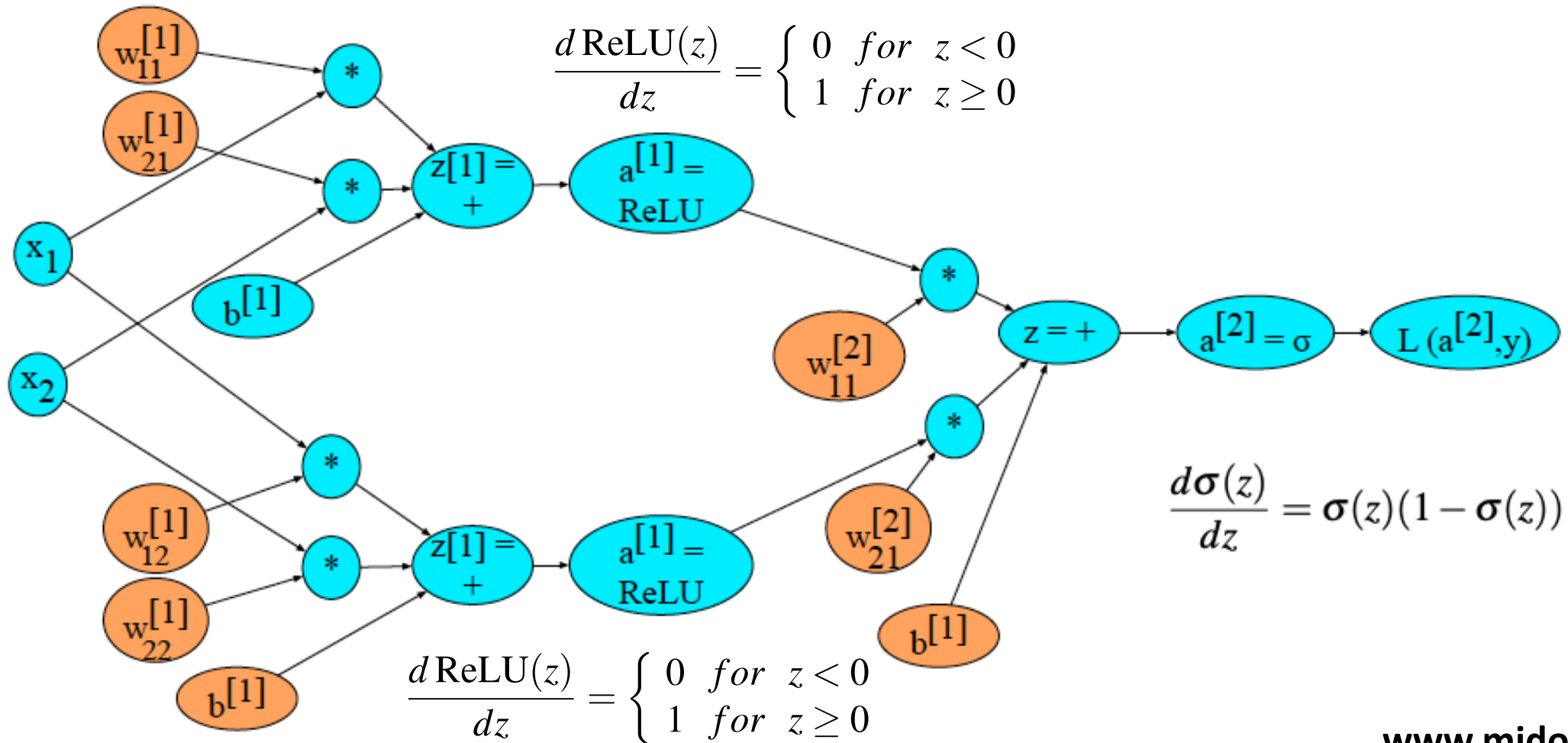
$$a^{[2]} = \sigma(z^{[2]})$$

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

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

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

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 $z^{[2]}$)

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

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

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

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

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

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

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

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

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

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

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

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.

Parts of Speech

- From the earliest linguistic traditions (Yaska and Panini 5th C. BCE, Aristotle 4th C. BCE), the idea that words can be classified into grammatical categories
 - Part of speech, word classes, POS, POS tags
- 8 parts of speech attributed to Dionysius Thrax of Alexandria (c. 1st C. BCE):
 - noun, verb, pronoun, preposition, adverb, conjunction, participle, article
 - These categories are relevant for NLP today.

Two classes of words: Open vs. Closed

- Closed class words
 - Relatively fixed membership
 - Usually **function** words: short, frequent words with grammatical function
 - determiners: *a, an, the*
 - pronouns: *she, he, I*
 - prepositions: *on, under, over, near, by, ...*
- Open class words
 - Usually **content** words: Nouns, Verbs, Adjectives, Adverbs
 - Plus interjections: *oh, ouch, uh-huh, yes, hello*
 - New nouns and verbs like *iPhone* or *to fax*

Open class ("content") words

Nouns

Proper

Janet
Italy

Common

cat, cats
mango

Verbs

Main

eat
went

Adjectives

old green tasty

Adverbs

slowly yesterday

Numbers

122,312
one

Interjections *Ow hello*

... more

Closed class ("function")

Determiners *the some*

Conjunctions *and or*

Pronouns *they its*

Auxiliary

can
had

Prepositions *to with*

Particles *off up*

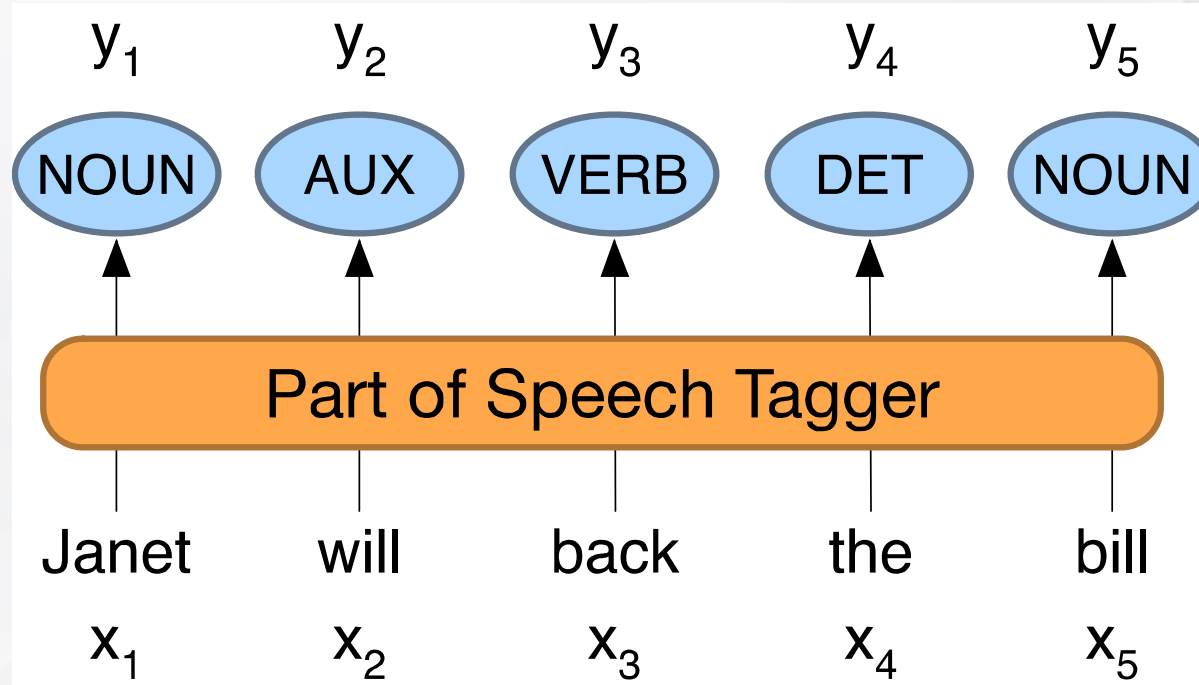
... more

Part-of-Speech Tagging

- Assigning a part-of-speech to each word in a text.
- Words often have more than one POS.
- **book:**
 - VERB: (***Book** that flight*)
 - NOUN: (*Hand me that **book***).

Part-of-Speech Tagging

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



"Universal Dependencies" Tagset

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

Sample "Tagged" English sentences

- There/**PRO** were/**VERB** 70/**NUM** children/**NOUN** there/**ADV** ./**PUNC**
- Preliminary/**ADJ** findings/**NOUN** were/**AUX** reported/**VERB** in/**ADP** today/**NOUN** 's/**PART** New/**PROPN** England/**PROPN** Journal/**PROPN** of/**ADP** Medicine/**PROPN**

Why Part of Speech Tagging?

- Can be useful for other NLP tasks
 - Parsing: POS tagging can improve syntactic parsing
 - MT: reordering of adjectives and nouns (say from Spanish to English)
 - Sentiment or affective tasks: may want to distinguish adjectives or other POS
 - Text-to-speech (how do we pronounce “lead” or “object”?)

How difficult is POS tagging in English?

- Roughly 15% of word types are ambiguous
 - Hence 85% of word types are unambiguous
 - Janet is always PROP, hesitantly is always ADV
- But those 15% tend to be very common.
- So ~60% of word tokens are ambiguous
 - E.g., *back*
 - earnings growth took a *back*/ADJ seat
 - a small building in the *back*/NOUN
 - a clear majority of senators *back*/VERB the bill
 - enable the country to buy *back*/PART debt
 - I was twenty-one *back*/ADV then

POS tagging performance in English

- How many tags are correct? (Tag accuracy)
 - About 97%
 - Hasn't changed in the last 10+ years
 - HMMs, CRFs, BERT perform similarly .
 - Human accuracy about the same
- But baseline is 92%!
 - Baseline is performance of stupidest possible method
 - "Most frequent class baseline" is an important baseline for many tasks
 - Tag every word with its most frequent tag
 - (and tag unknown words as nouns)
 - Partly easy because
 - Many words are unambiguous

Sources of information for POS tagging

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

- Prior probabilities of word/tag
 - "**will**" is usually an AUX
- Identity of neighboring words
 - "**the**" means the next word is probably not a verb
- Morphology and wordshape:
 - Prefixes
 - Suffixes
 - Capitalization

unable: **un-** →

importantly: **-ly** → ADJ

Janet: **CAP** → PROP

Standard algorithms for POS tagging

- Supervised Machine Learning Algorithms:
 - Hidden Markov Models
 - Conditional Random Fields (CRF)/ Maximum Entropy Markov Models (MEMM)
 - Neural sequence models (RNNs or Transformers)
 - Large Language Models (like BERT), finetuned
- All required a hand-labeled training set, all about equal performance (97% on English)
- All make use of information sources we discussed
 - Via human created features: HMMs and CRFs
 - Via representation learning: Neural LMs

Named Entities

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

Named Entity tagging

- The task of named entity recognition (NER):
 - find spans of text that constitute proper names
 - tag the type of the entity.

NER output

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

Why NER?

- Sentiment analysis: consumer's sentiment toward a particular company or person?
- Question Answering: answer questions about an entity?
- Information Extraction: Extracting facts about entities from text.

Why NER is hard

1) Segmentation

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

2) Type ambiguity

[PER Washington] was born into slavery on the farm of James Burroughs.
[ORG Washington] went up 2 games to 1 in the four-game series.
Blair arrived in [LOC Washington] for what may well be his last state visit.
In June, [GPE Washington] passed a primary seatbelt law.

BIO Tagging

- How can we turn this structured problem into a sequence problem like POS tagging, with one label per word?
- [PER Jane Villanueva] of [ORG United] , a unit of [ORG United Airlines Holding] , said the fare applies to the [LOC Chicago] route.

BIO Tagging

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

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

Now we have one tag per token!!!

BIO Tagging

- B: token that *begins* a span
- I: tokens *inside* a span
- O: tokens outside of any span

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

Standard algorithms for NER

- Supervised Machine Learning given a human-labeled training set of text annotated with tags
- Hidden Markov Models
- Conditional Random Fields (CRF)/ Maximum Entropy Markov Models (MEMM)
- Neural sequence models (RNNs or Transformers)
- Large Language Models (like BERT), finetuned

Summary

- Understand Neural Language Models
- Define and understand Feedforward Neural Networks
- Understand Parts of Speech (POS)
- Define and Explain Named Entity Recognition (NER)



مجموعة جامعة
الجميع الذكية
للتعليم القابضة

MIDOCEAN
UNIVERSITY



Thank You

