Stanford·CS124 | From Languages to Information (2021)

CS124(2021)· 课程资料包 @ShowMeAI

视频
中英双语字幕

课件
一键打包下载

笔记
官方笔记翻译

代码
作业项目解析

视频 ·B 站 [扫码或点击链接]
https://www.bilibili.com/video/BV1YA411w7ym

课件 & 代码 · 博客 [扫码或点击链接]
http://blog.showmeai.tech/cs124/

文本处理 信息检索 编辑距离 神经嵌入 序列标注 PageRank 推荐系统 社交网络分析 倒排索引 对话系统 协同过滤

Awesome AI Courses Notes Cheatsheets 是 ShowMeAI 资料库的分支系列，覆盖最具知名度的 **TOP20+** 门 AI 课程，旨在为读者和学习者提供一整套高品质中文学习笔记和速查表。

点击课程名称，跳转至课程**资料包**页面，**一键下载**课程全部资料！

机器学习	深度学习	自然语言处理	计算机视觉
Stanford · CS229	Stanford · CS230	Stanford · CS224n	Stanford · CS231n
# Awesome AI Courses Notes Cheatsheets· 持续更新中			
知识图谱	图机器学习	深度强化学习	自动驾驶
Stanford · CS520	Stanford · CS224W	UCBerkeley · CS285	MIT · 6.S094

微信公众号

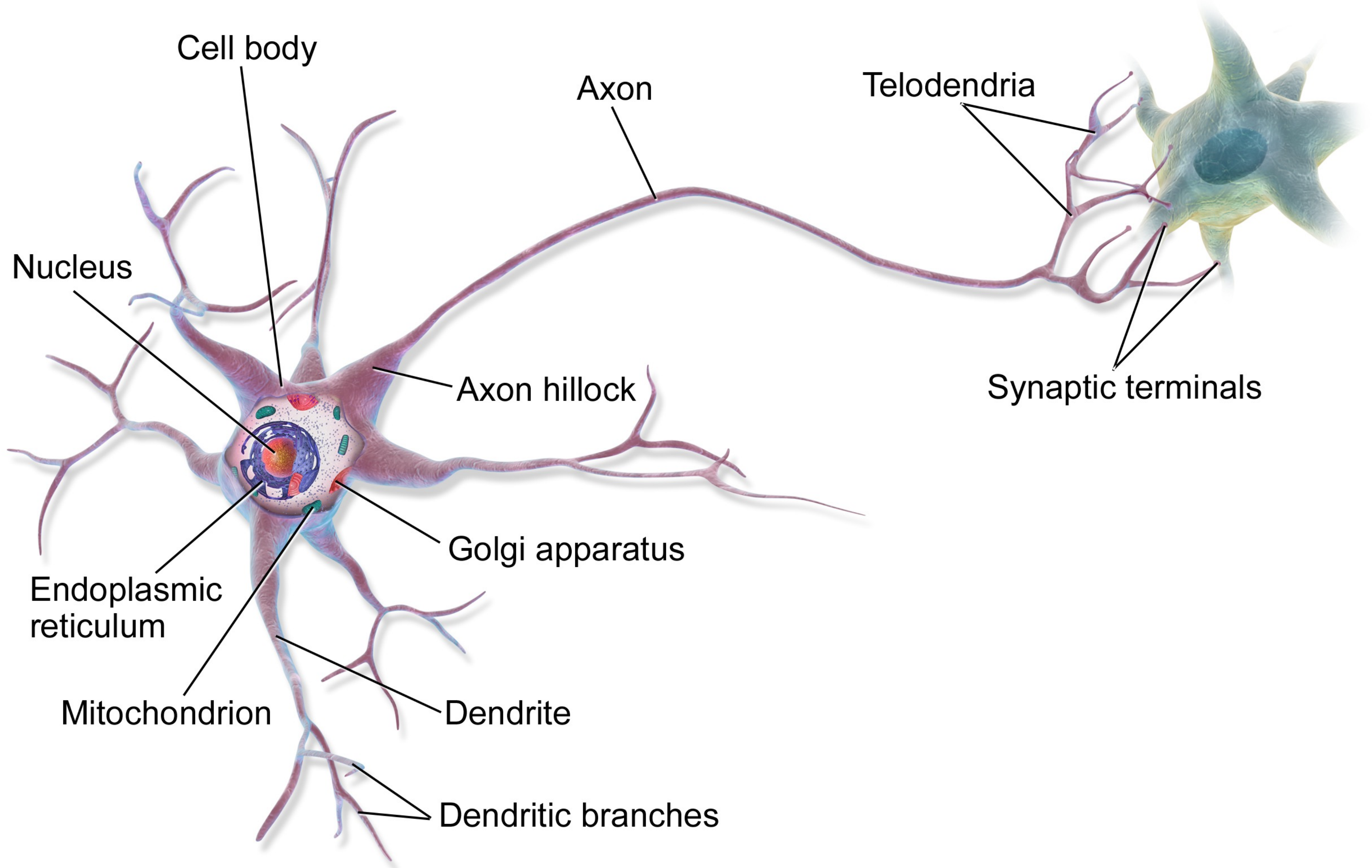
资料下载方式 2：扫码点击底部菜单栏

称为 **AI 内容创作者？** 回复 [添砖加瓦]

Simple Neural Networks and Neural Language Models

Units in Neural Networks

This is in your brain



Neural Network Unit

This is not in your brain

Output value

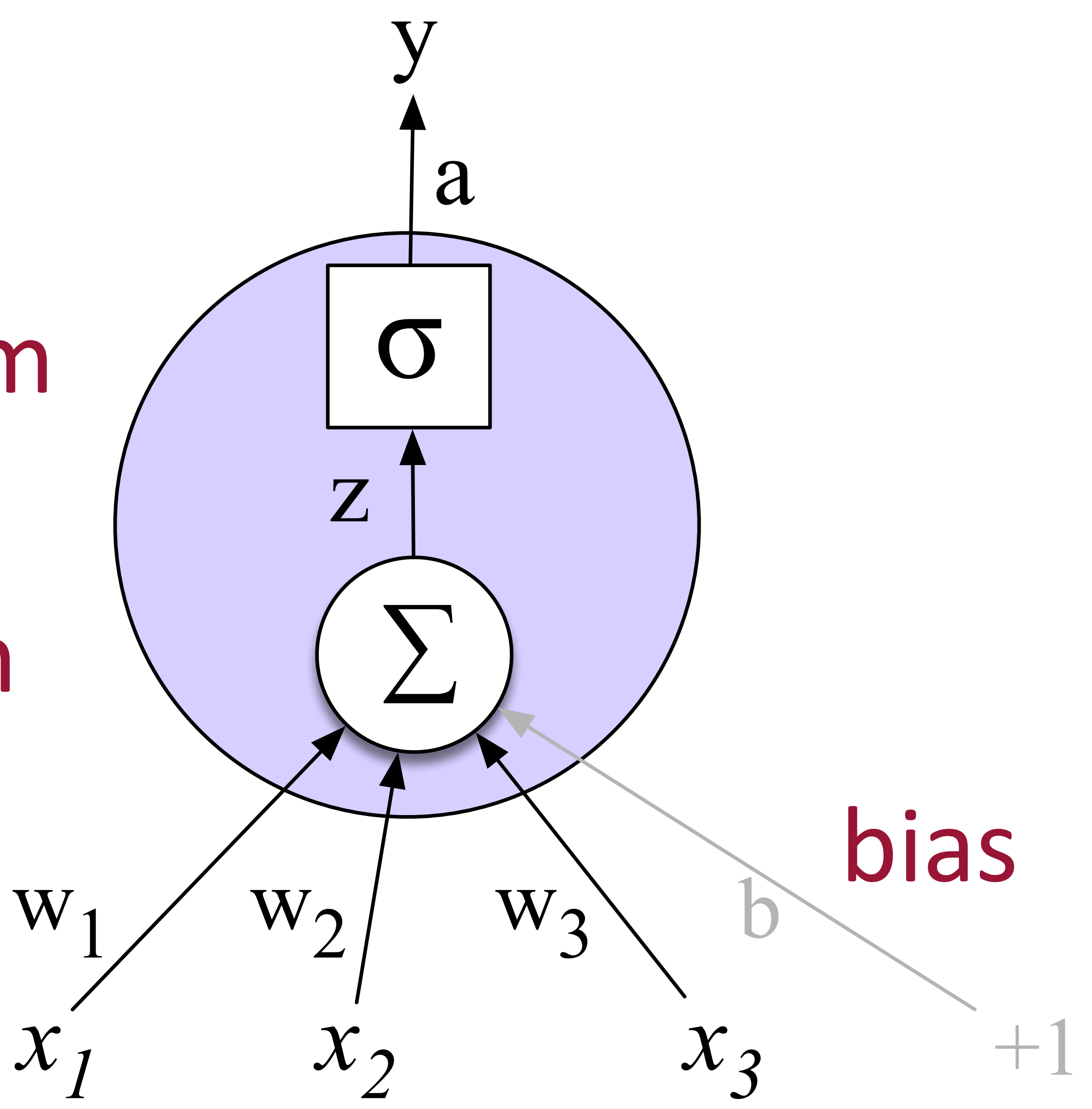
Non-linear transform

Weighted sum

Weights

Input layer

bias



Neural unit

Take weighted sum of inputs, plus a bias

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

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

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

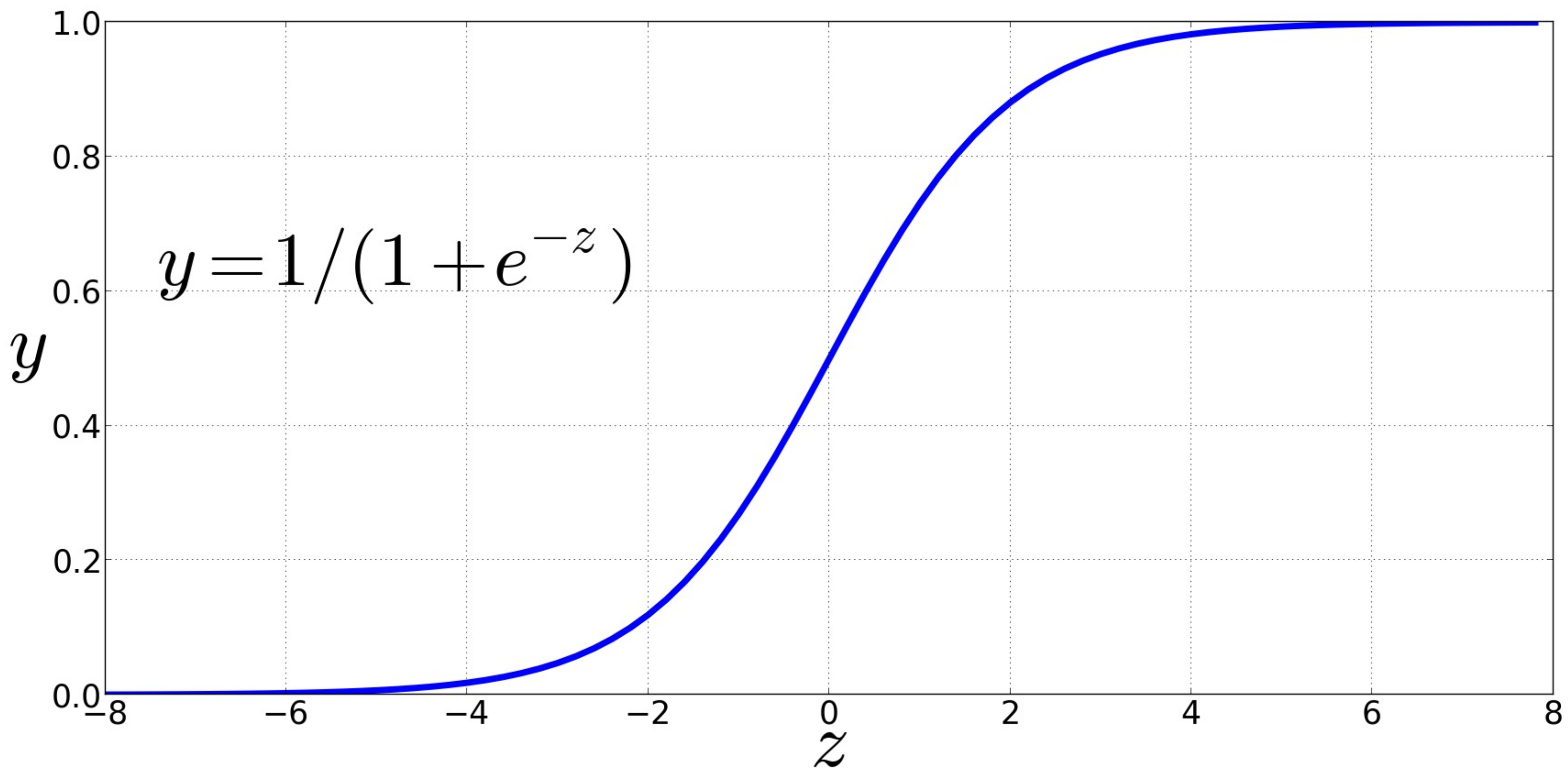
$$y = a = f(z)$$

Non-Linear Activation Functions

We're already seen the sigmoid for logistic regression:

Sigmoid

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



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

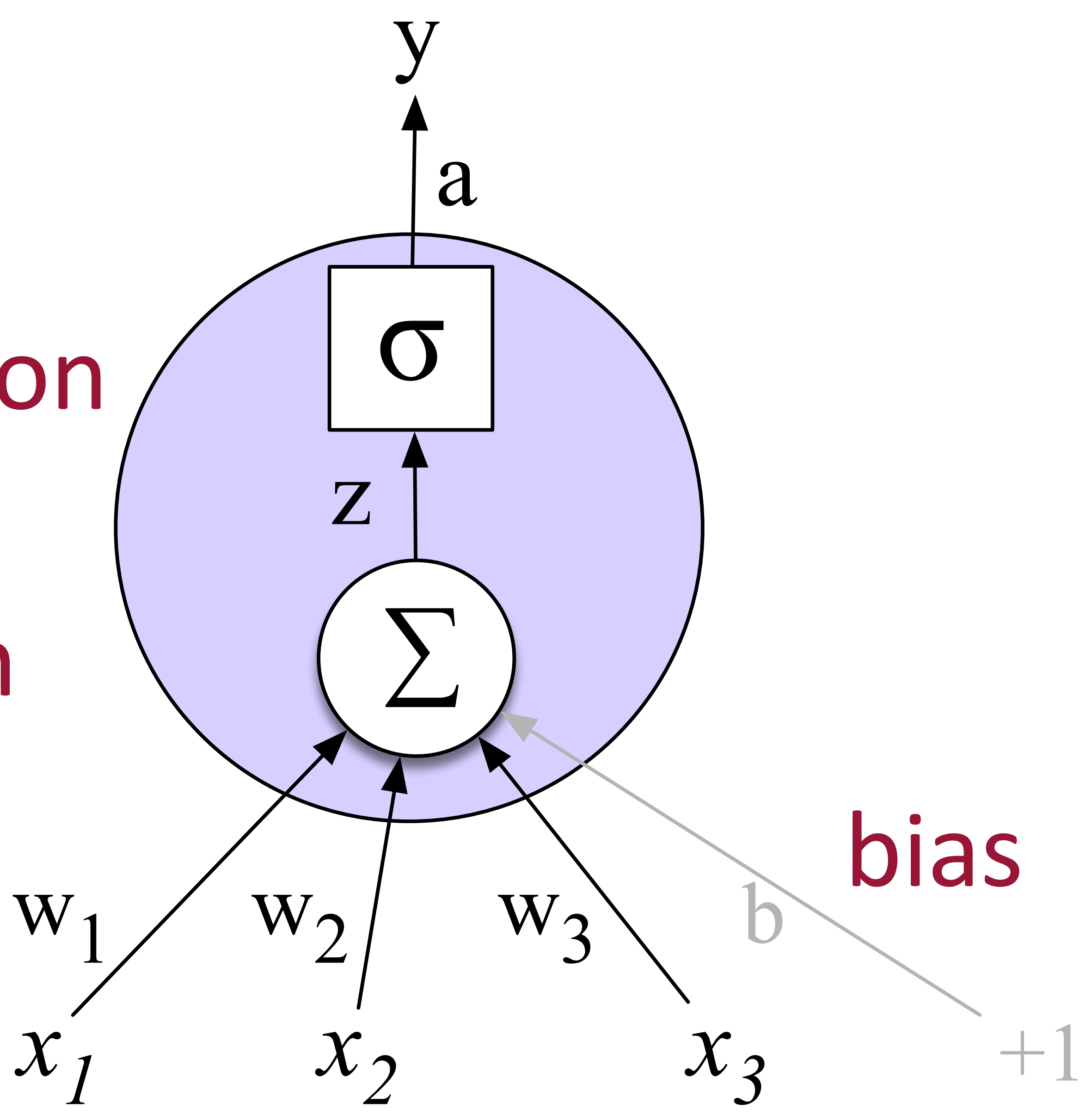
Non-linear activation function

Weighted sum

Weights

Input layer

bias



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]
```

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

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

An example

Suppose a unit has:

$w = [0.2, 0.3, 0.9]$

$b = 0.5$

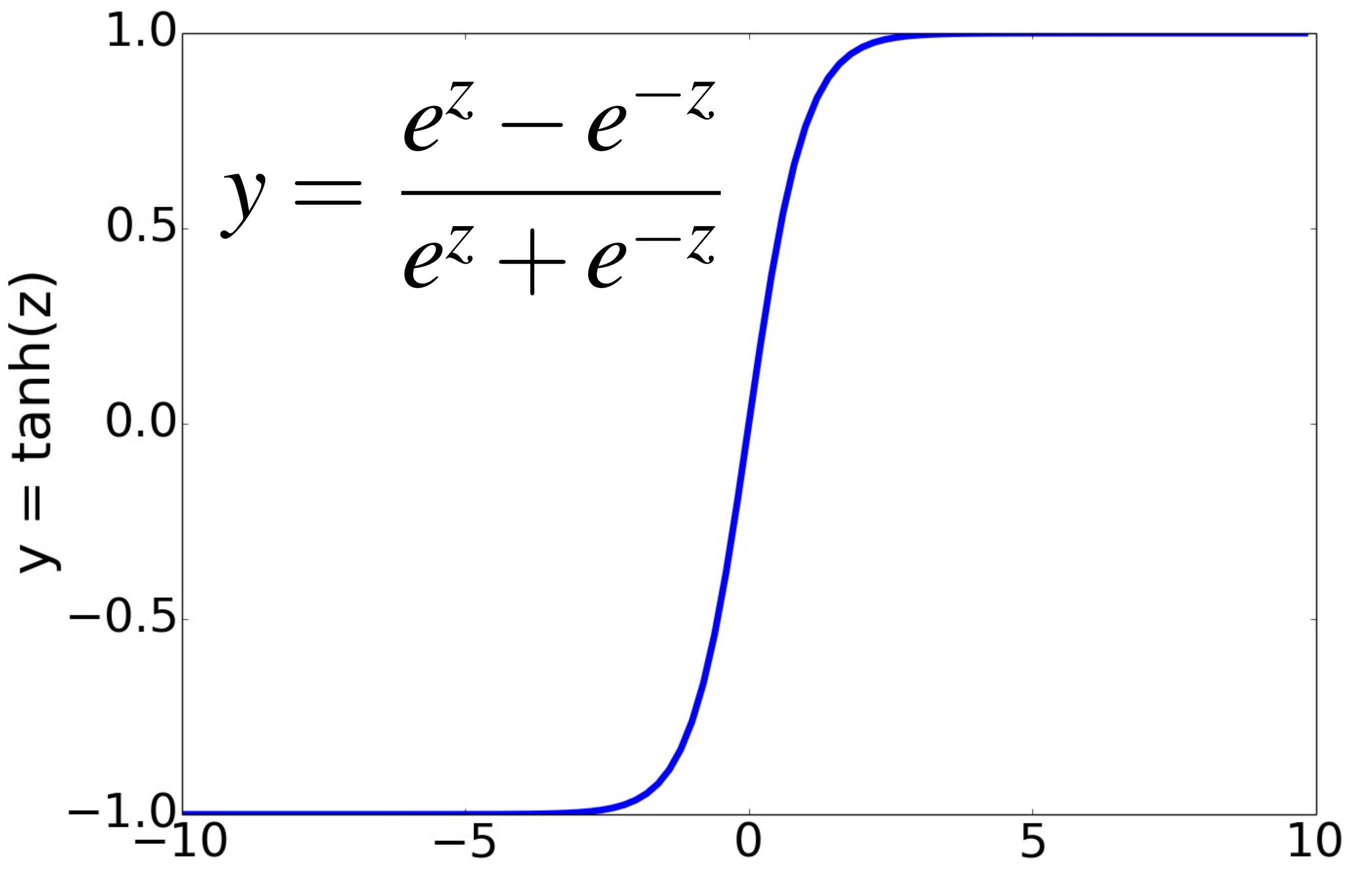
What happens with input x:

$x = [0.5, 0.6, 0.1]$

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

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

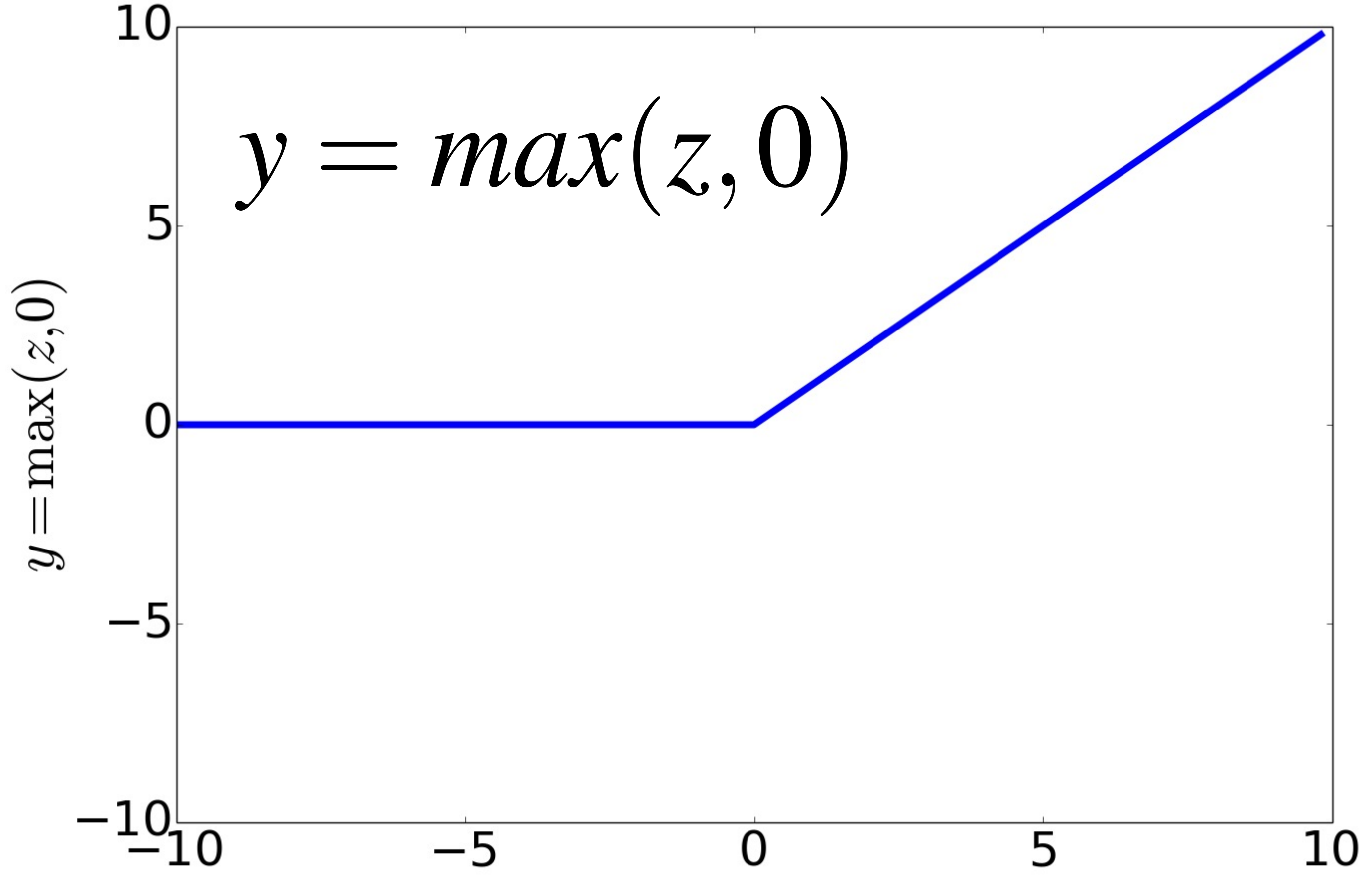
Non-Linear Activation Functions besides sigmoid



$$y = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

tanh

Most Common:



$$y = max(z, 0)$$

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

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

OR

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

XOR

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

Perceptrons

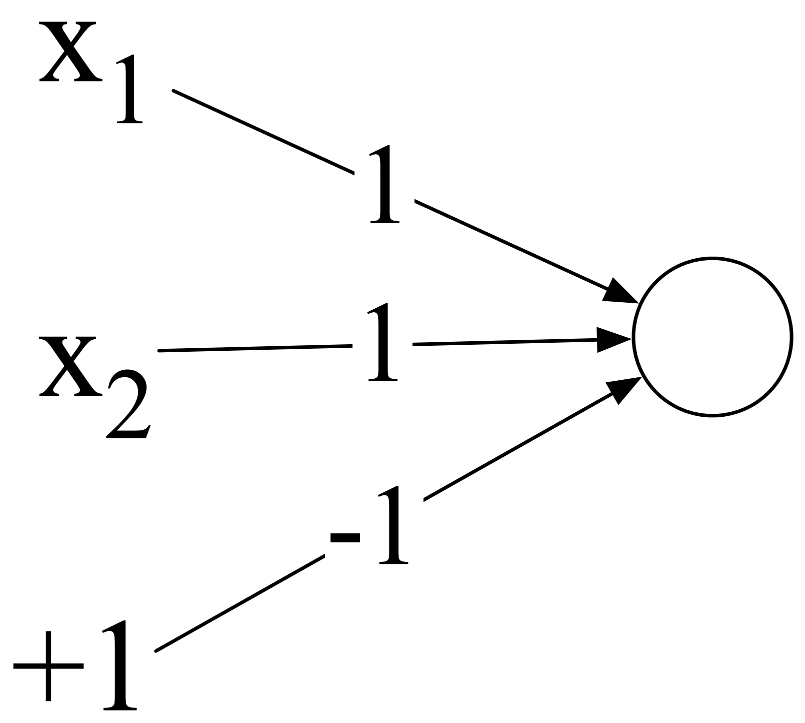
A very simple neural unit

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

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

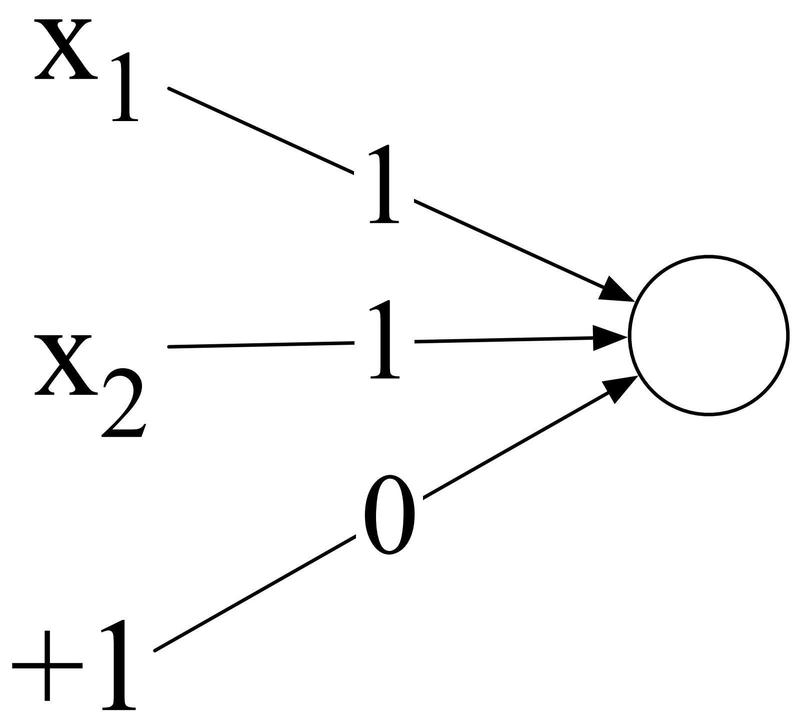
Easy to build AND or OR with perceptrons

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



AND

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

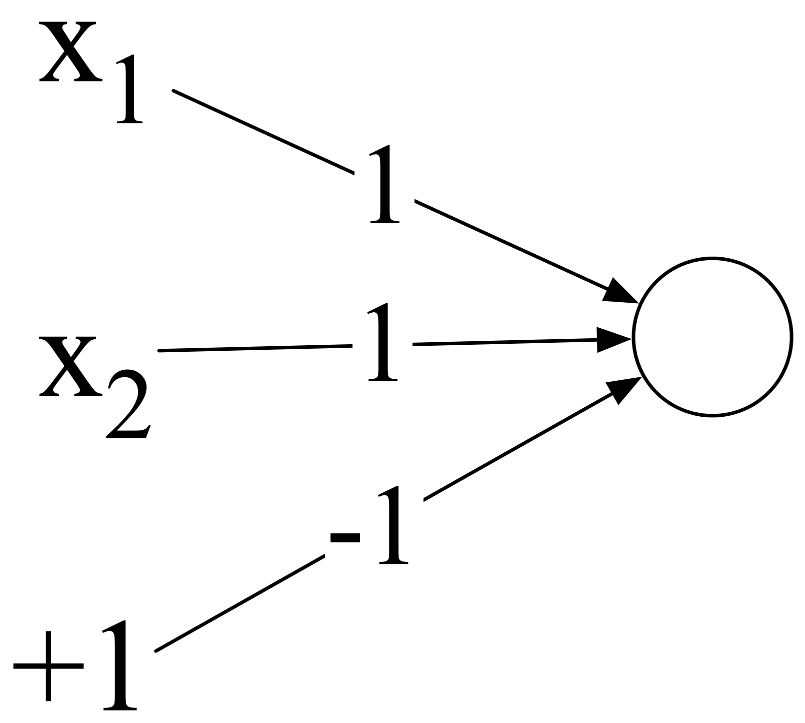


OR

OR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1

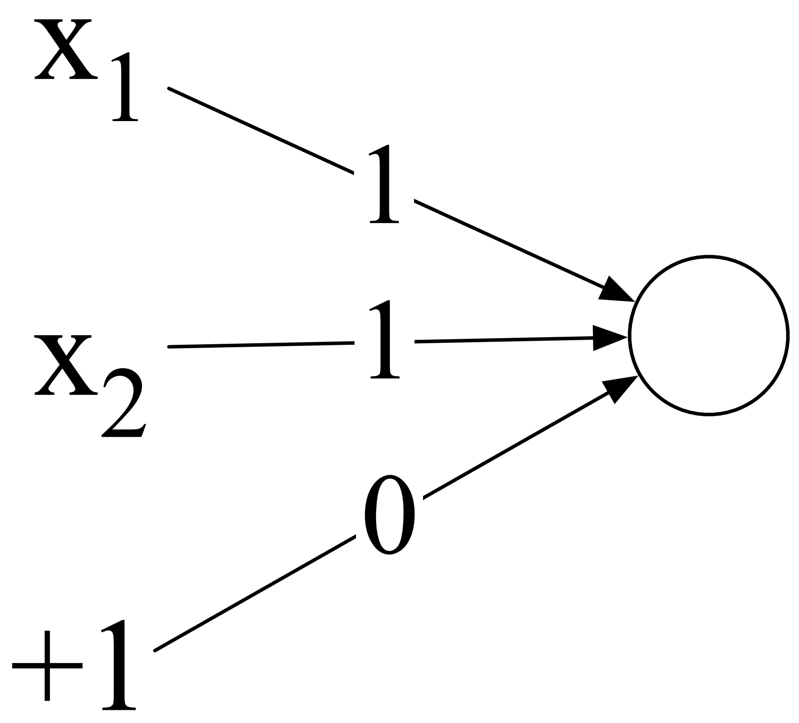
Easy to build AND or OR with perceptrons

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



AND

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

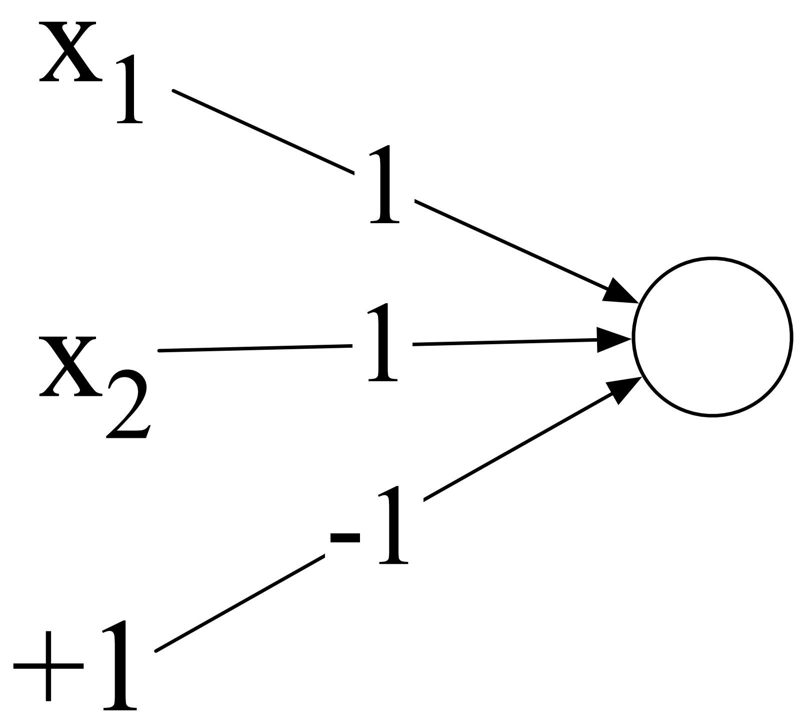


OR

OR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1

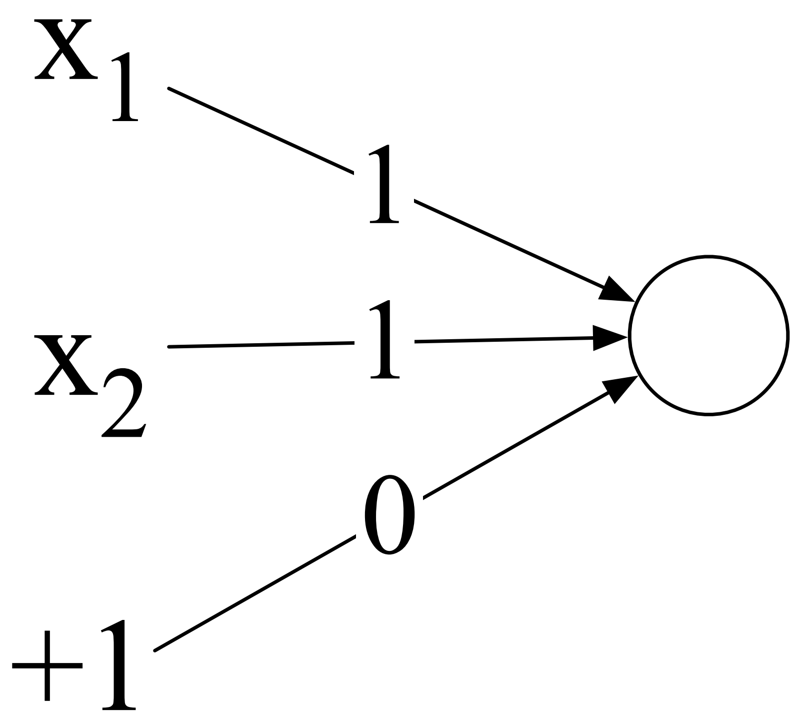
Easy to build AND or OR with perceptrons

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



AND

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



OR

OR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1

Not possible to capture XOR with perceptrons

Pause the lecture and try for yourself!

Why? Perceptrons are linear classifiers

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

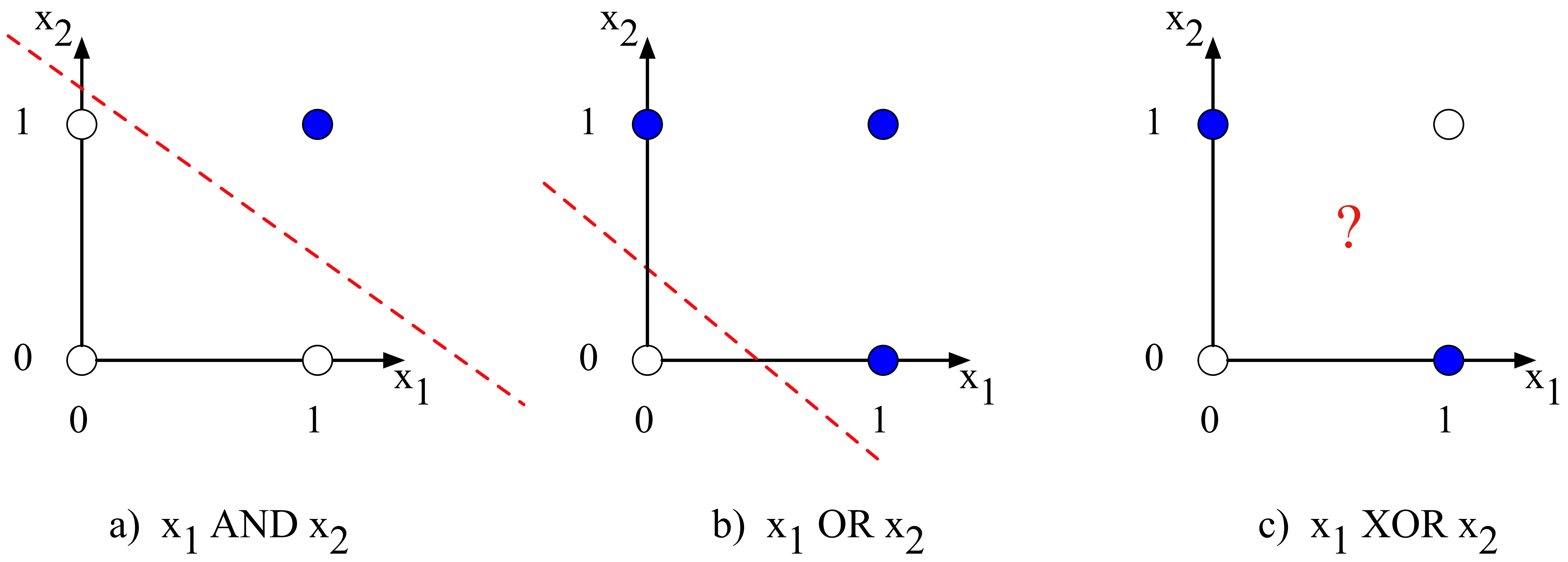
$$w_1x_1 + w_2x_2 + b = 0$$

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

This line acts as a **decision boundary**

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

Decision boundaries



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

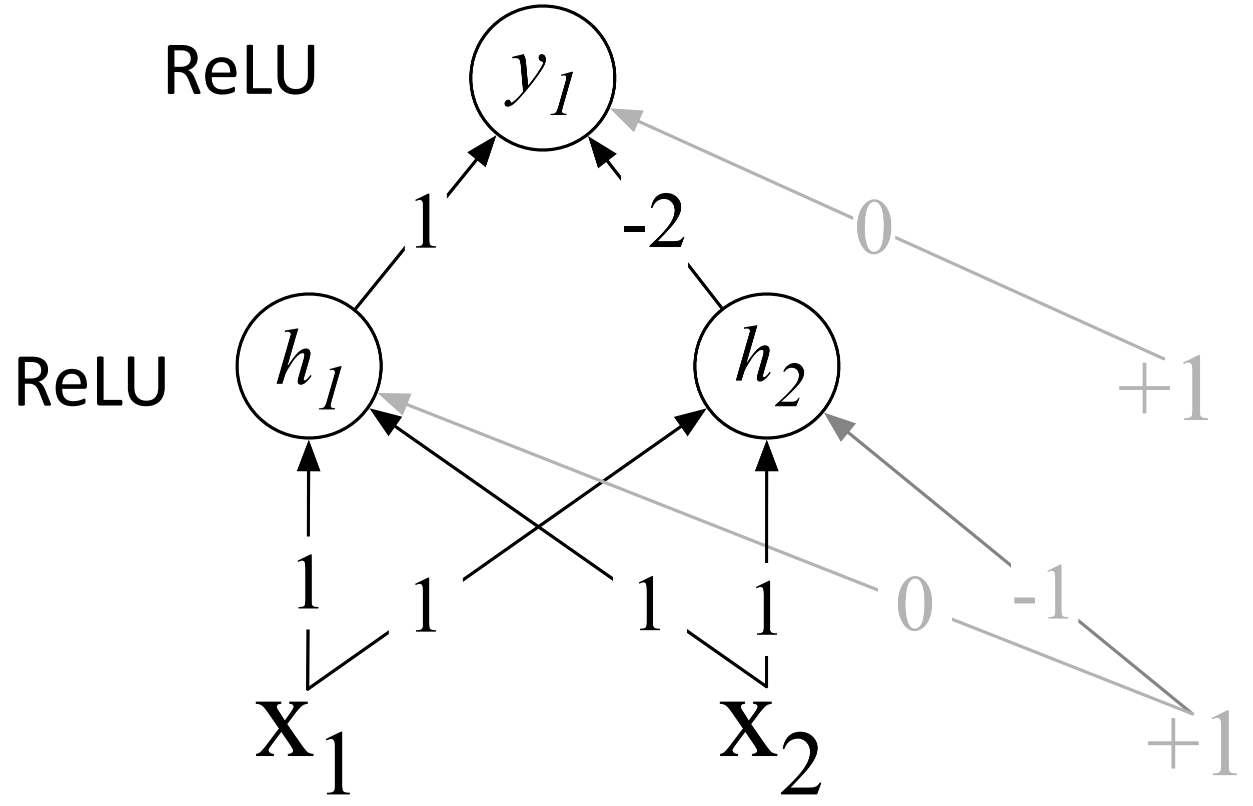
Solution to the XOR problem

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



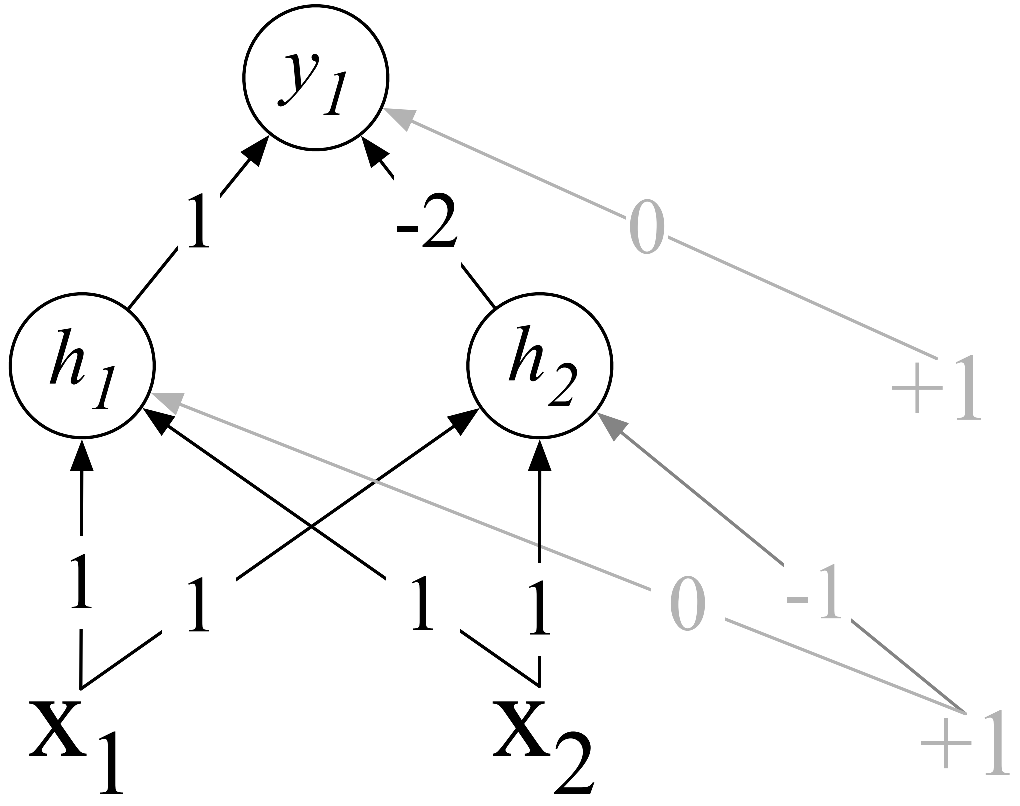
Solution to the XOR problem

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

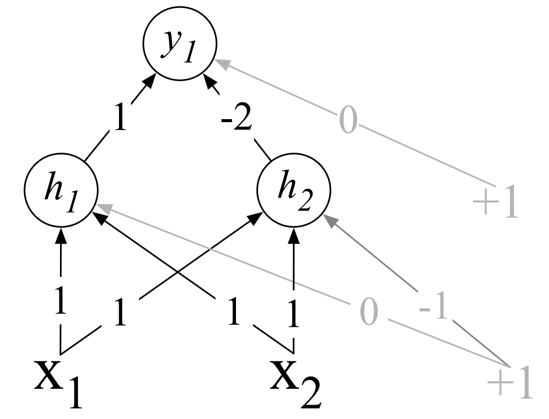
XOR **can** be calculated by a layered network of units.

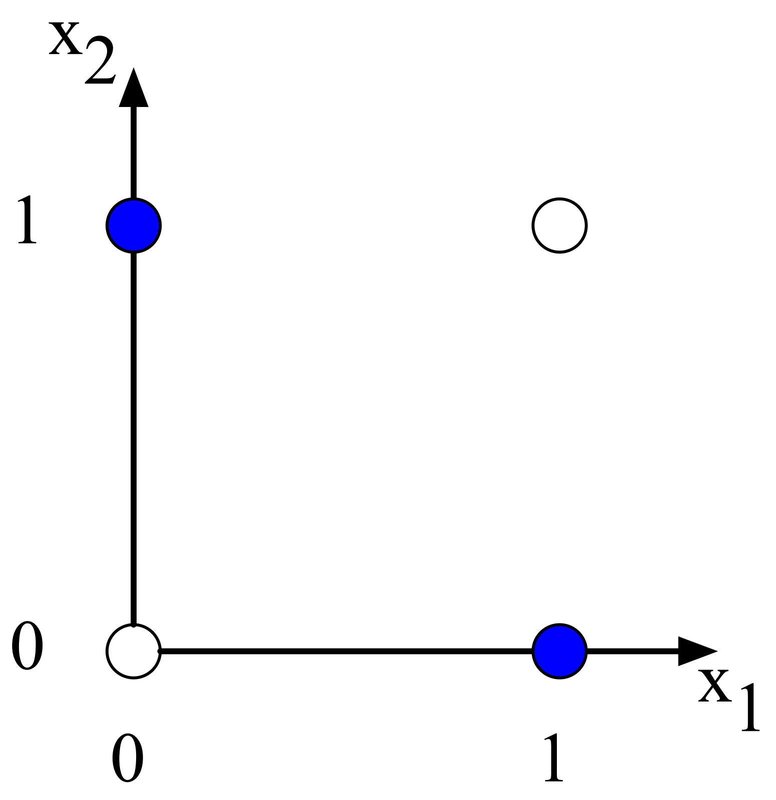
XOR

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

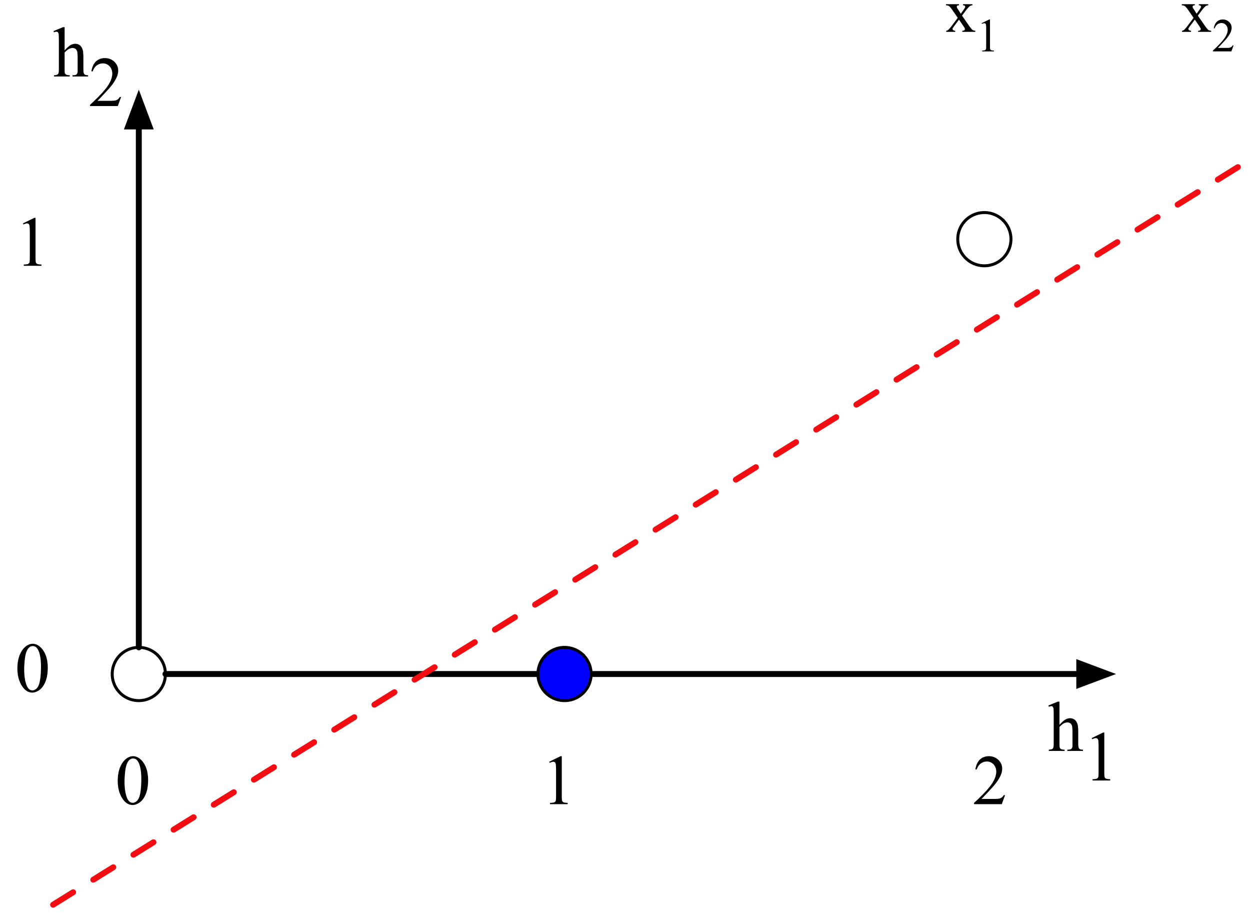


The hidden representation h





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

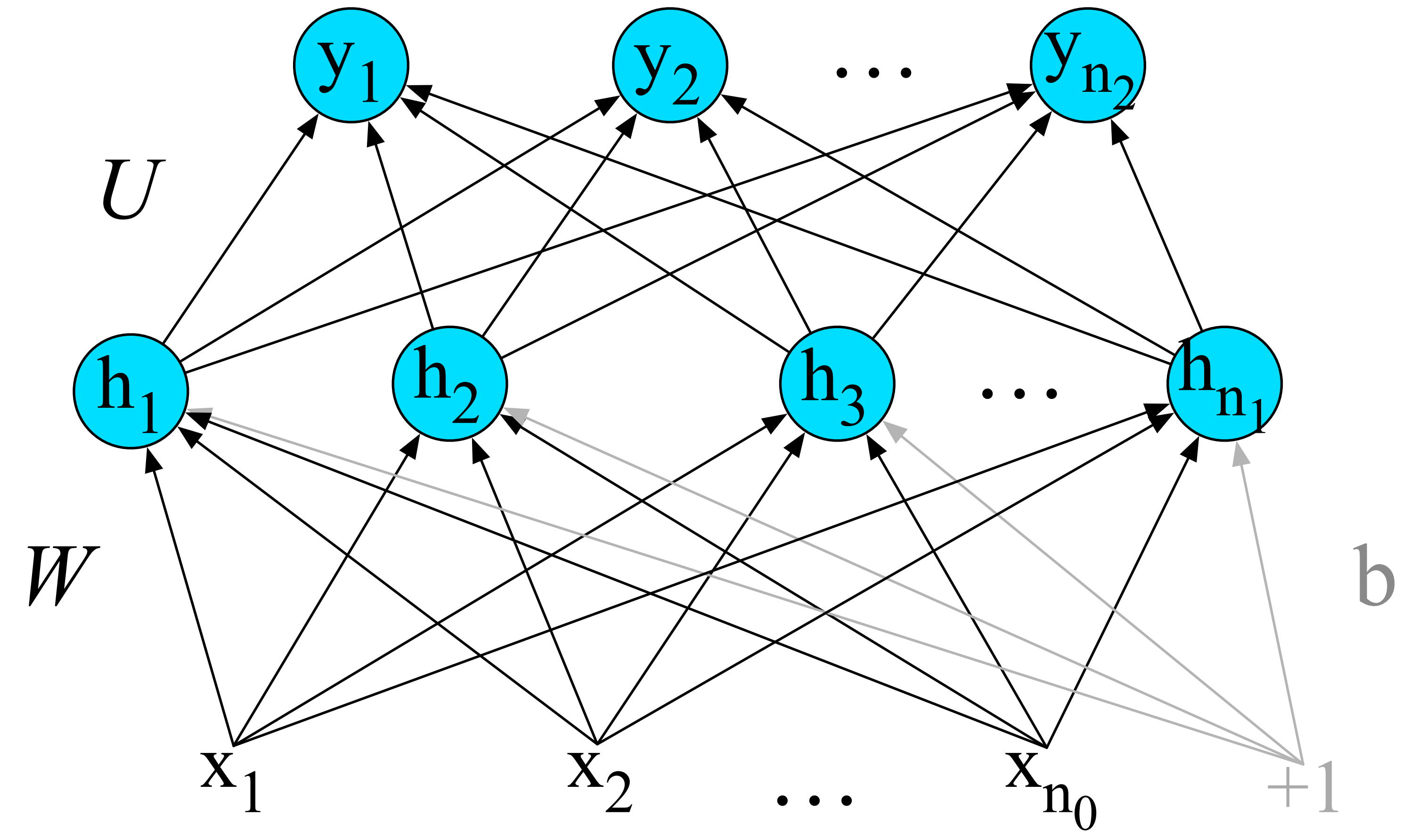
The XOR problem

Simple Neural Networks and Neural Language Models

Feedforward Neural Networks

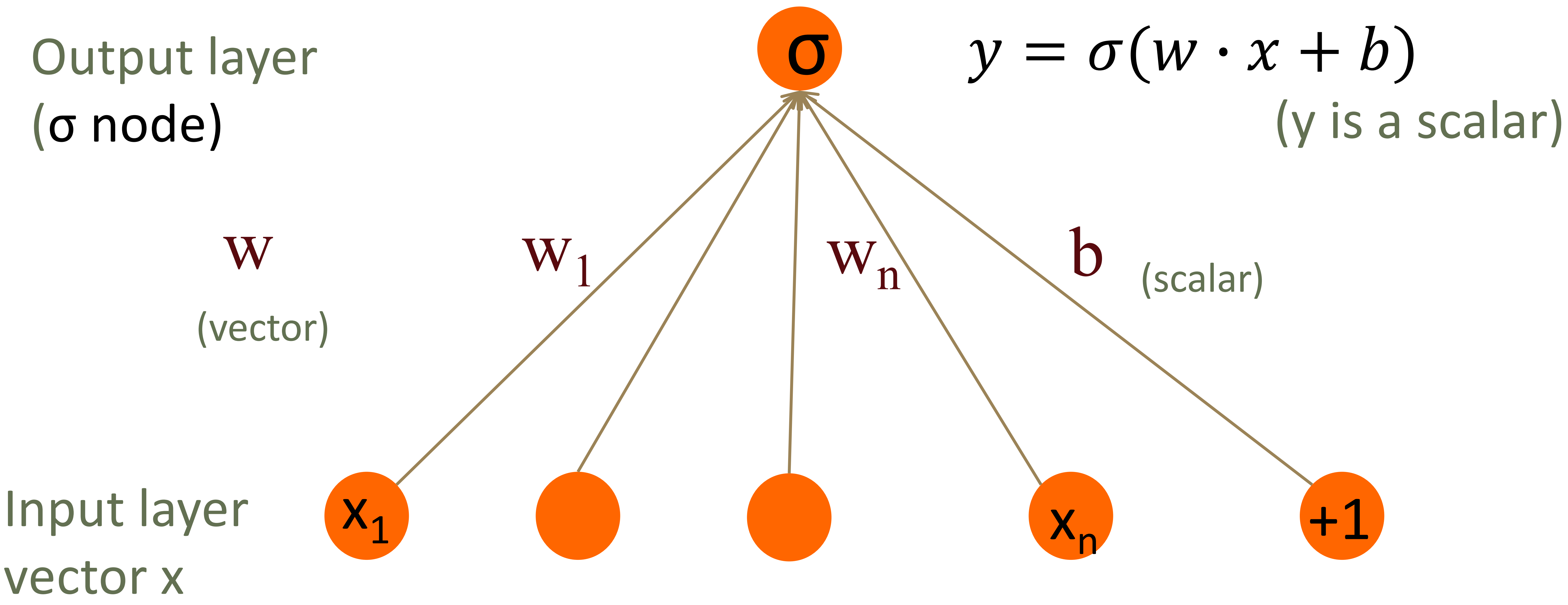
Feedforward Neural Networks

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!)



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

(y is a scalar)

Multinomial Logistic Regression as a 1-layer Network

Fully connected single layer network

Output layer (softmax nodes)

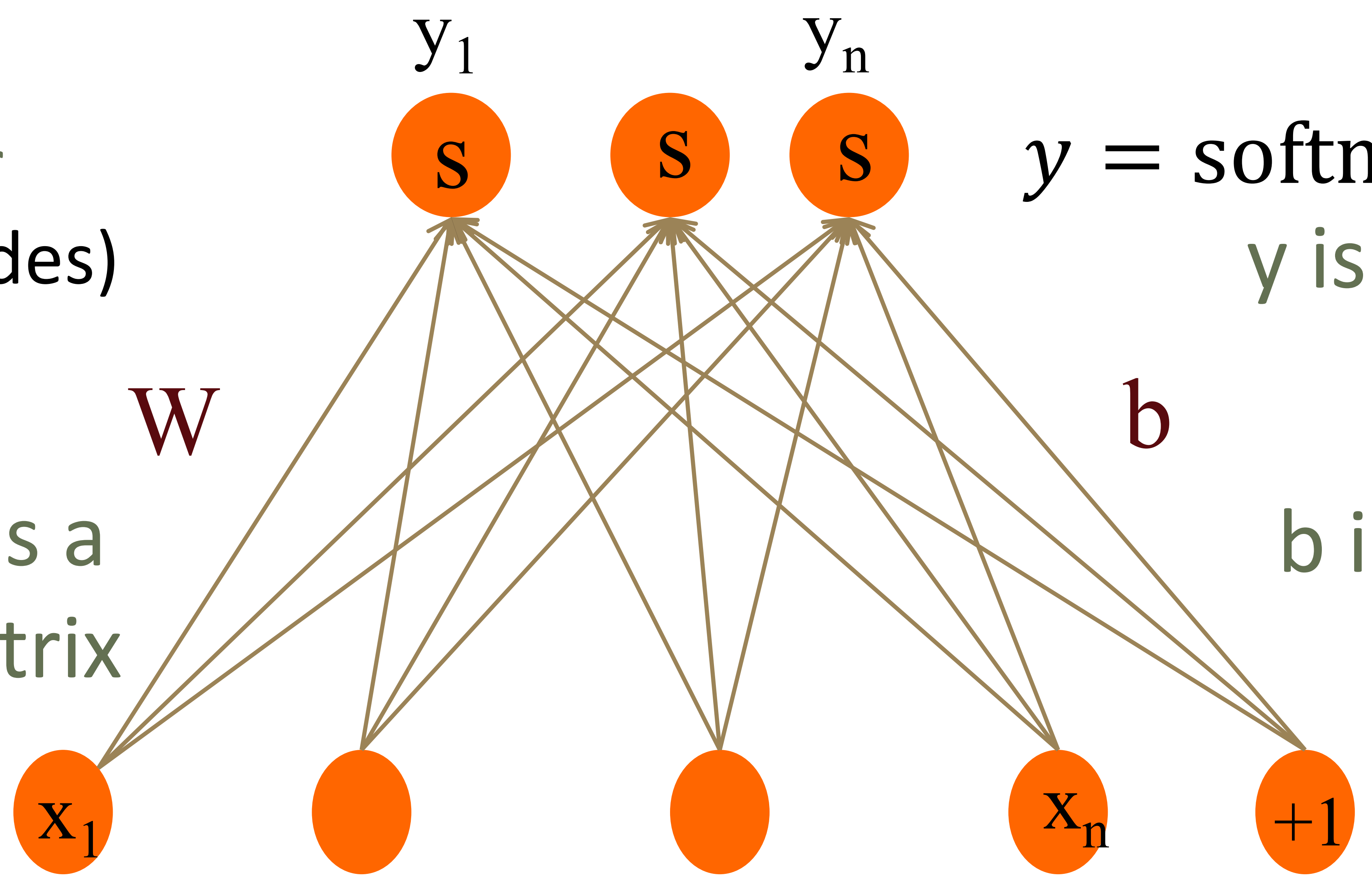
$$y = \mathrm{softmax}(Wx + b)$$

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:

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

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

Example:

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

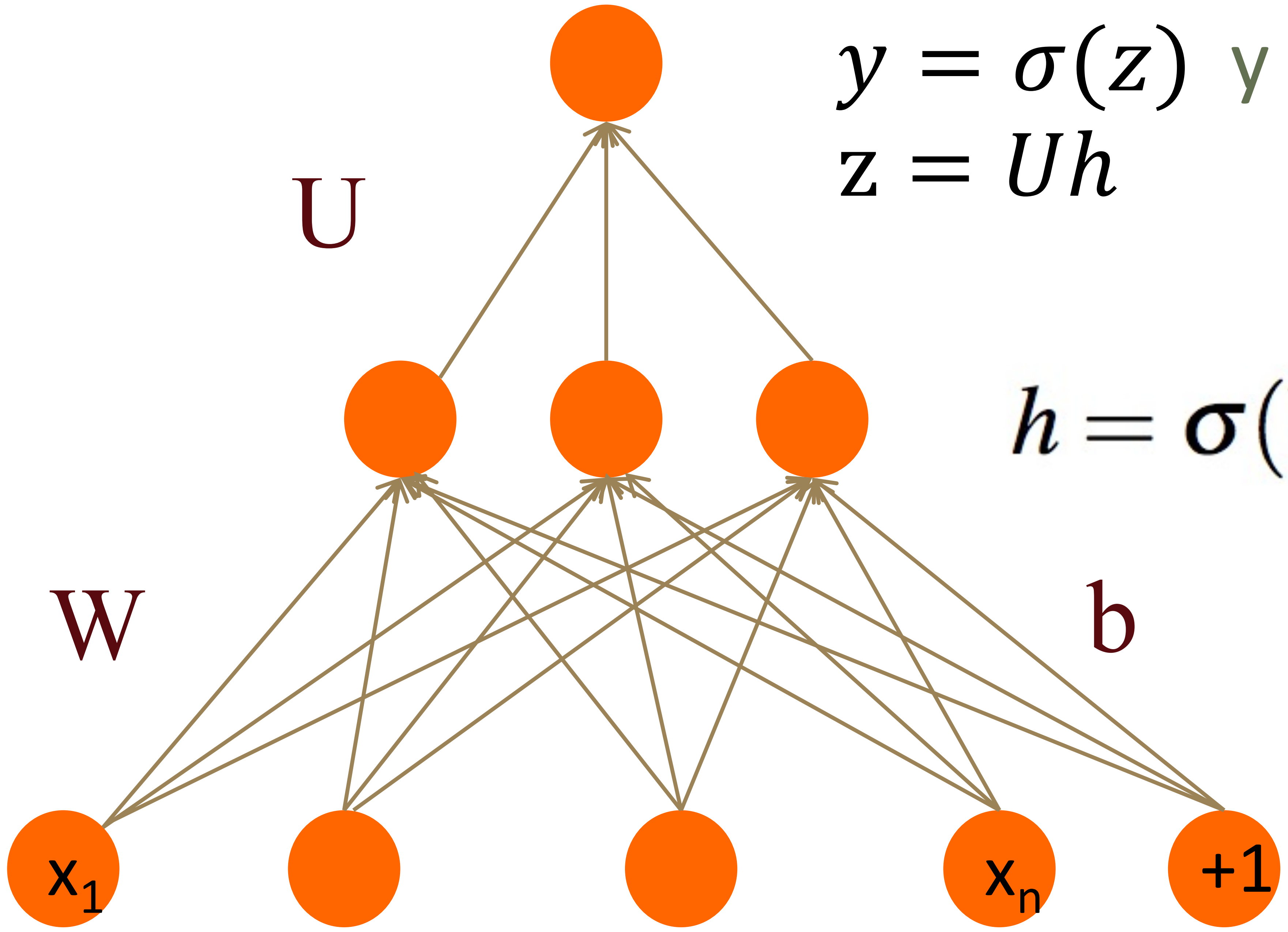
$$\mathrm{softmax}(z) = [0.055, 0.090, 0.006, 0.099, 0.74, 0.010]$$

Two-Layer Network with scalar output

Output layer
(σ node)

hidden units
(σ node)

Input layer
(vector)



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

$z = Uh$

$h = \sigma(Wx + b)$

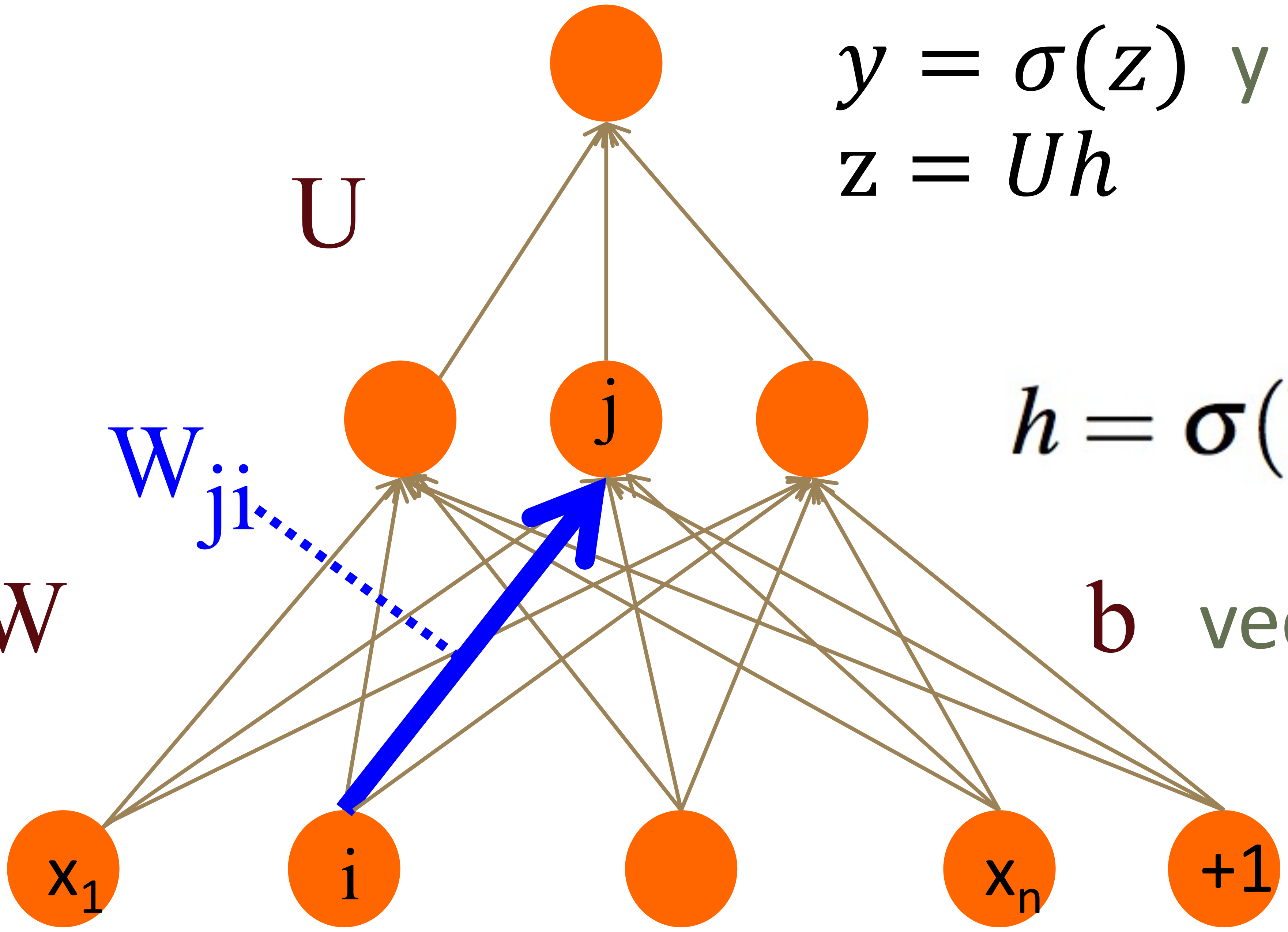
Could be ReLU
Or tanh

Two-Layer Network with scalar output

Output layer
(σ node)

hidden units
(σ node)

Input layer
(vector)



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

$z = Uh$

$h = \sigma(Wx + b)$

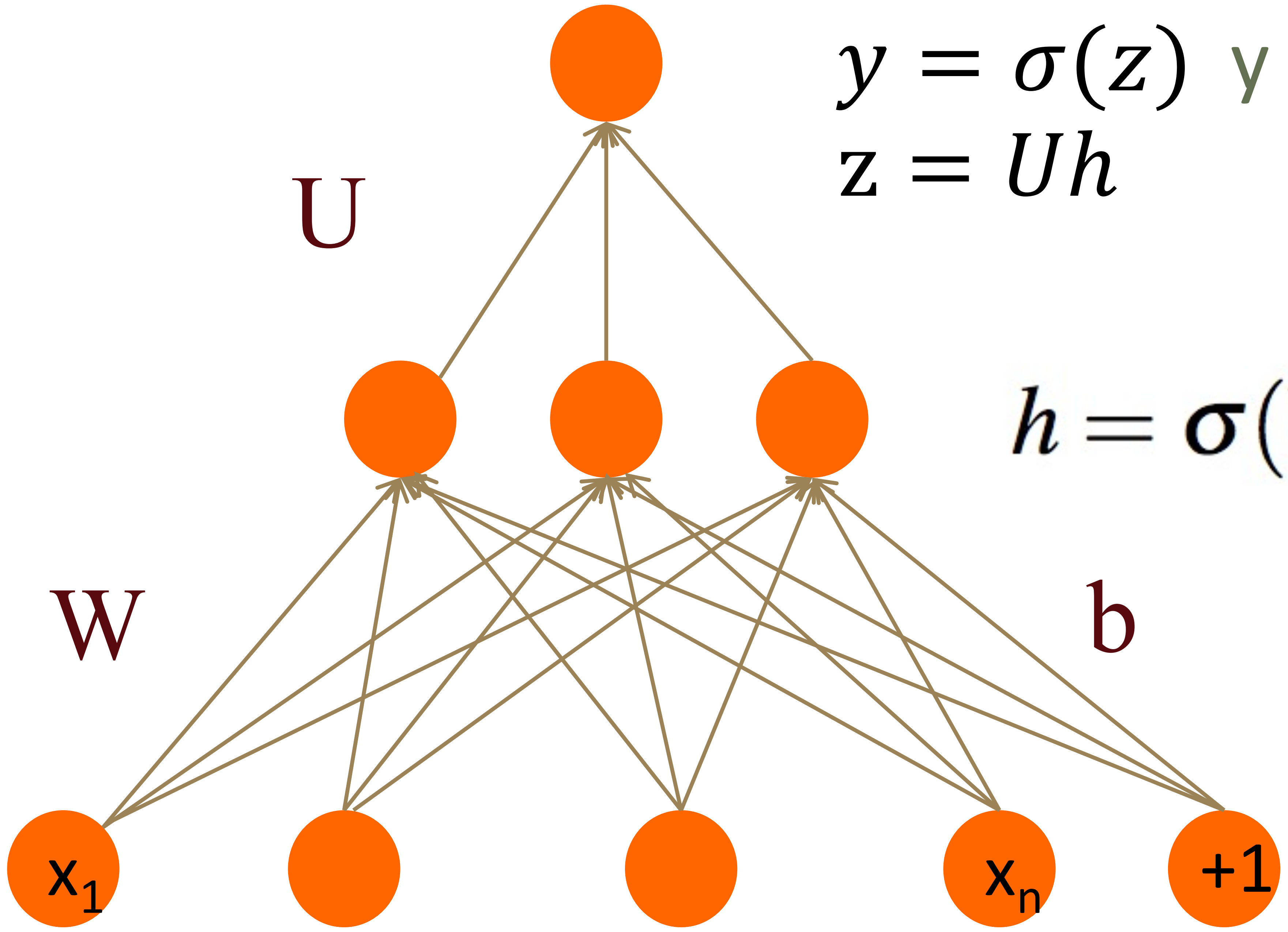
b vector

Two-Layer Network with scalar output

Output layer
(σ node)

hidden units
(σ node)

Input layer
(vector)



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

$z = Uh$

$h = \sigma(Wx + b)$

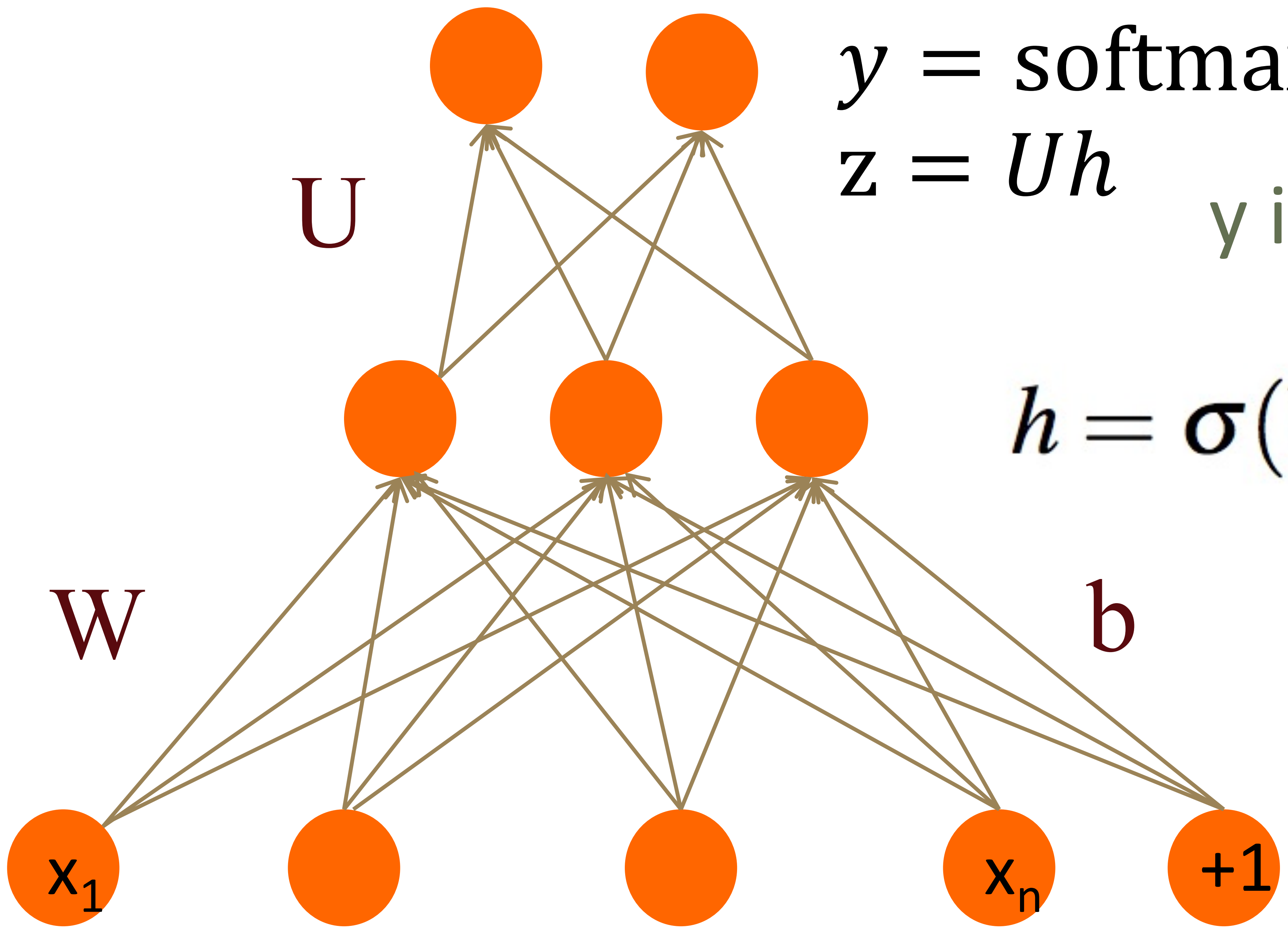
Could be ReLU
Or tanh

Two-Layer Network with softmax output

Output layer
(σ node)

hidden units
(σ node)

Input layer
(vector)



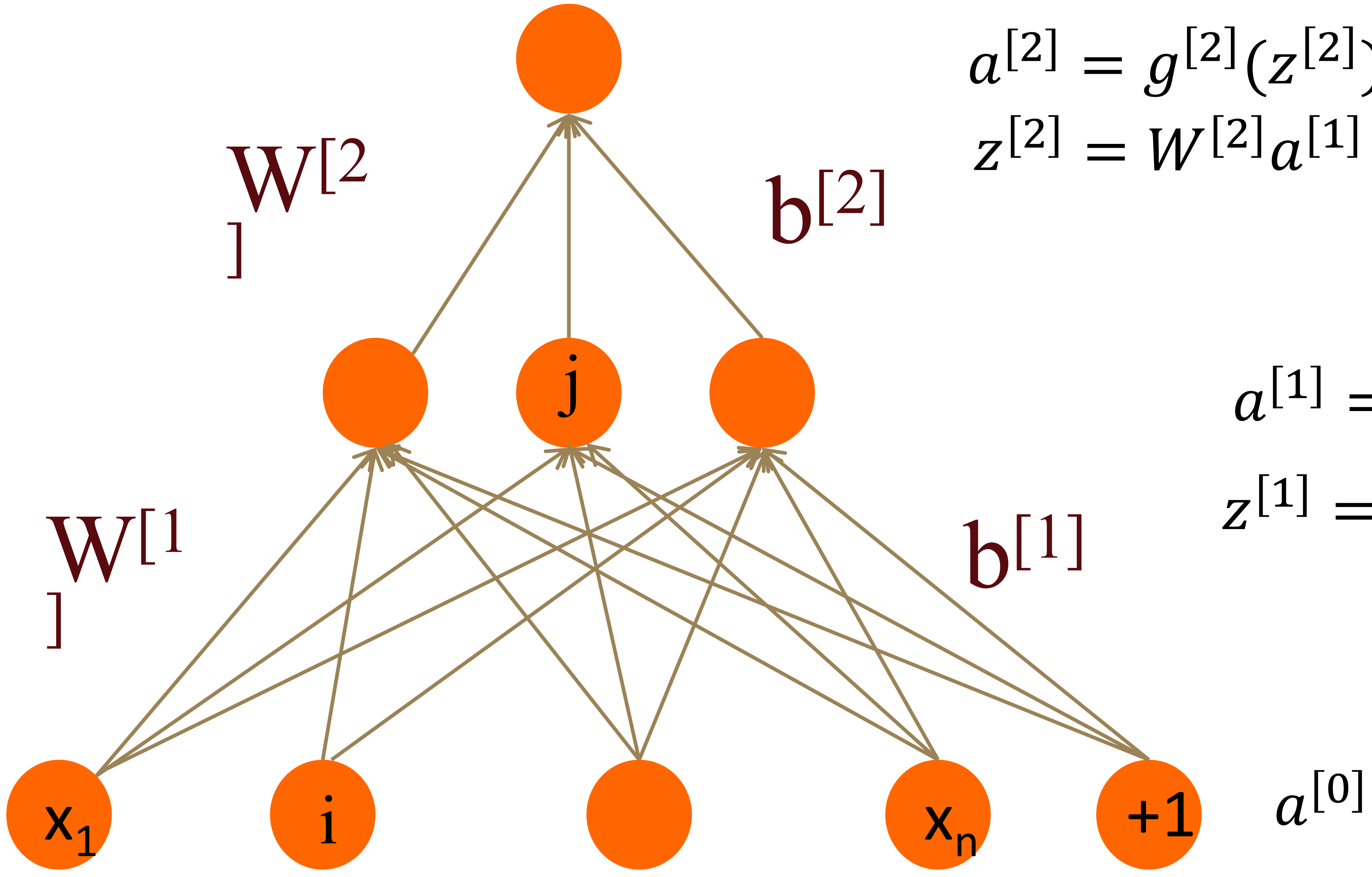
$$y = \text{softmax}(z)$$
$$z = Uh$$

y is a vecto

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

Could be ReLU
Or tanh

Multi-layer Notation



$y = a^{[2]}$

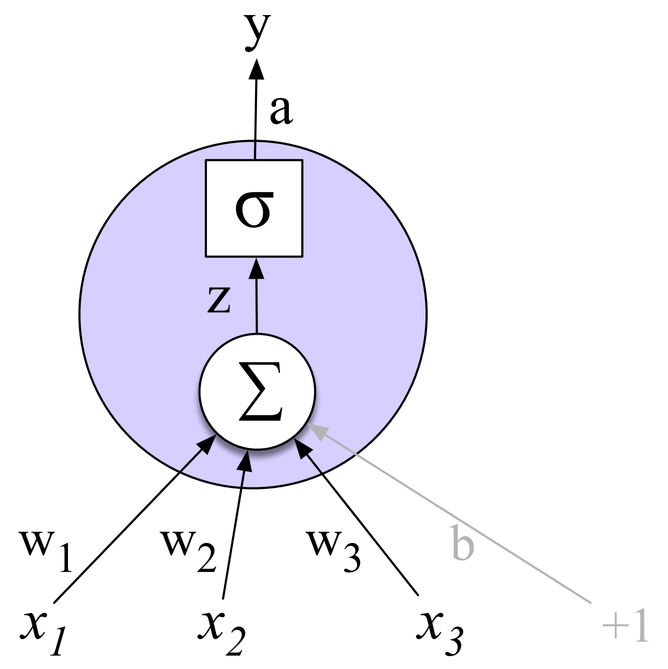
$a^{[2]} = g^{[2]}(z^{[2]})$ sigmoid or softmax

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

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

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

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

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 $a^{[1]}_0=1$, $a^{[2]}_0=1$,…

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:

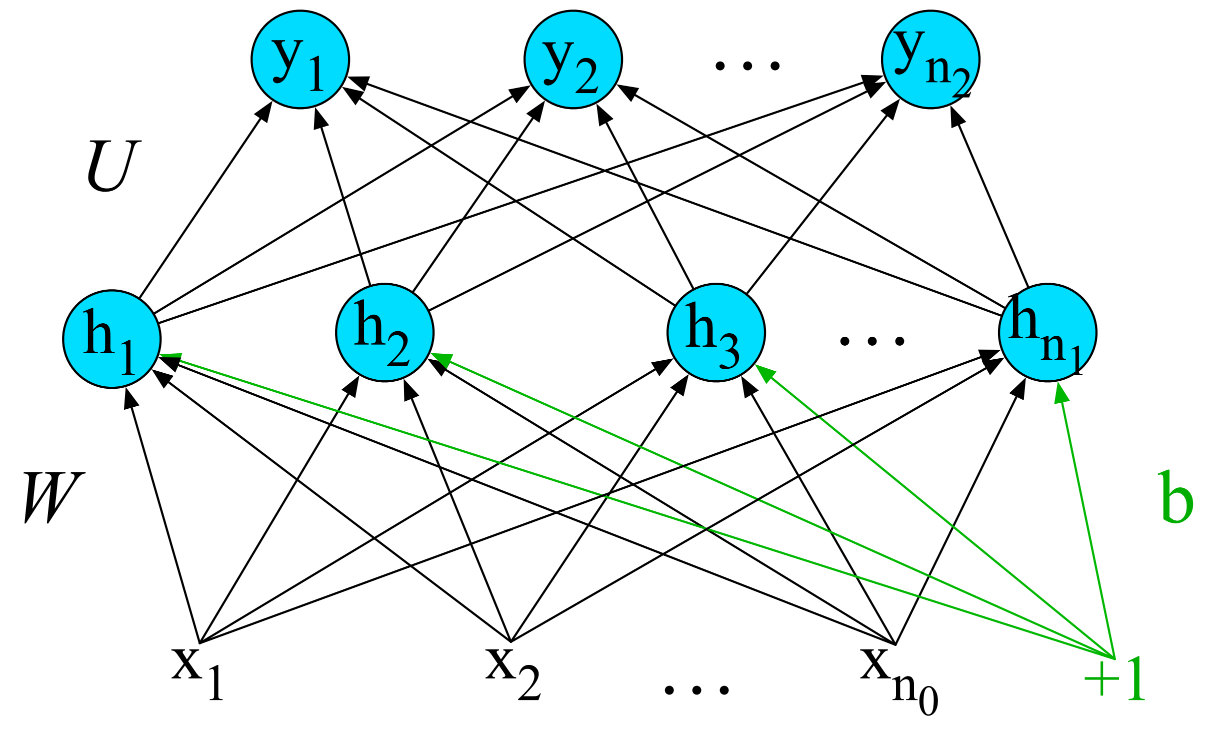
$$x = x_0, x_1, x_2, \ldots, x_{n0}$$

$$h = \sigma(Wx)$$

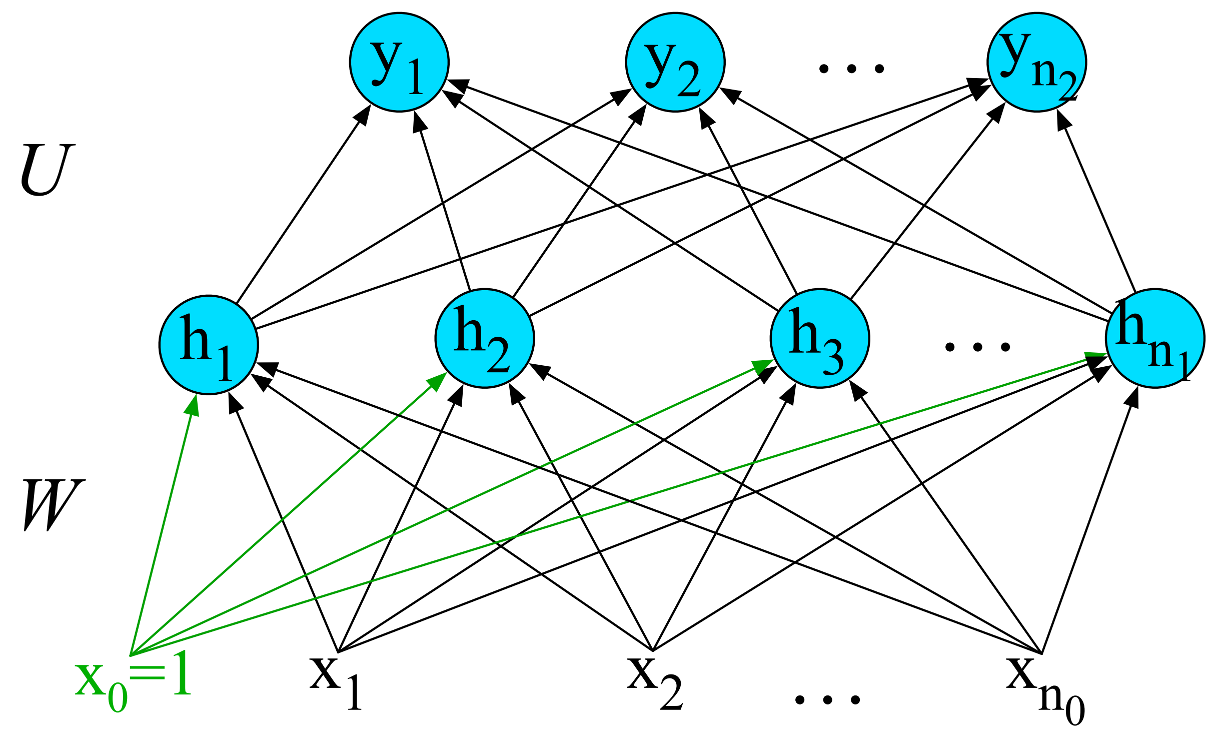
$$\sigma\left(\sum_{i=0}^{n_0} W_{ji}x_i\right)$$

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

Applying feedforward networks to NLP tasks

Use cases for feedforward networks

Let's consider 2 (simplified) sample tasks:

1. Text classification
2. Language modeling

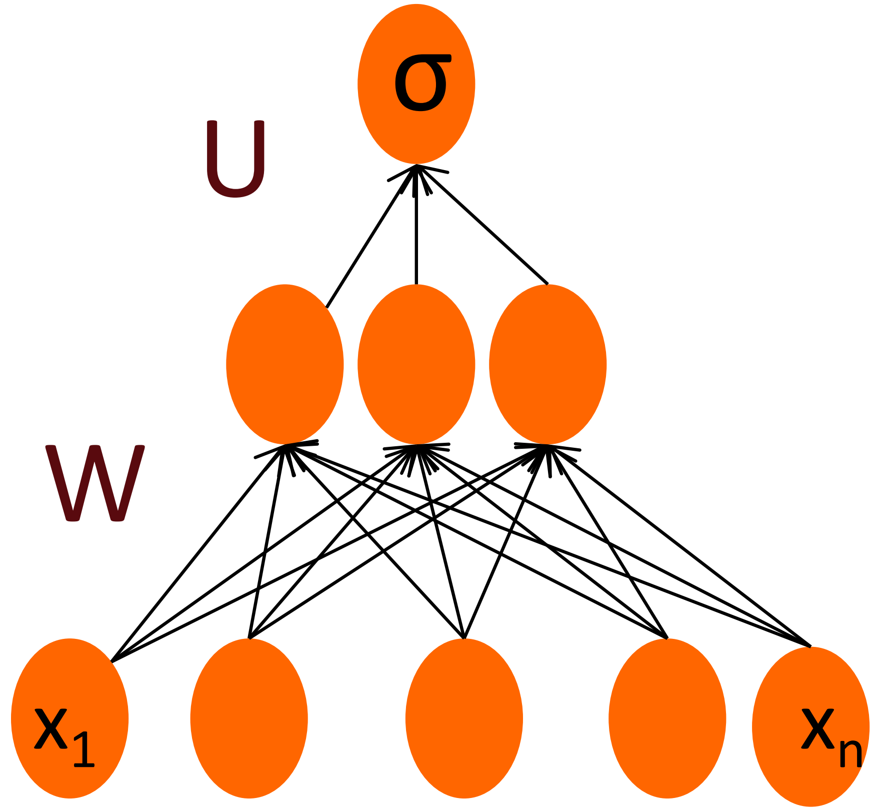
State of the art systems use more powerful neural architectures, but simple models are useful to consider!

Classification: Sentiment Analysis

We could do exactly what we did with logistic regression

Input layer are binary features as before

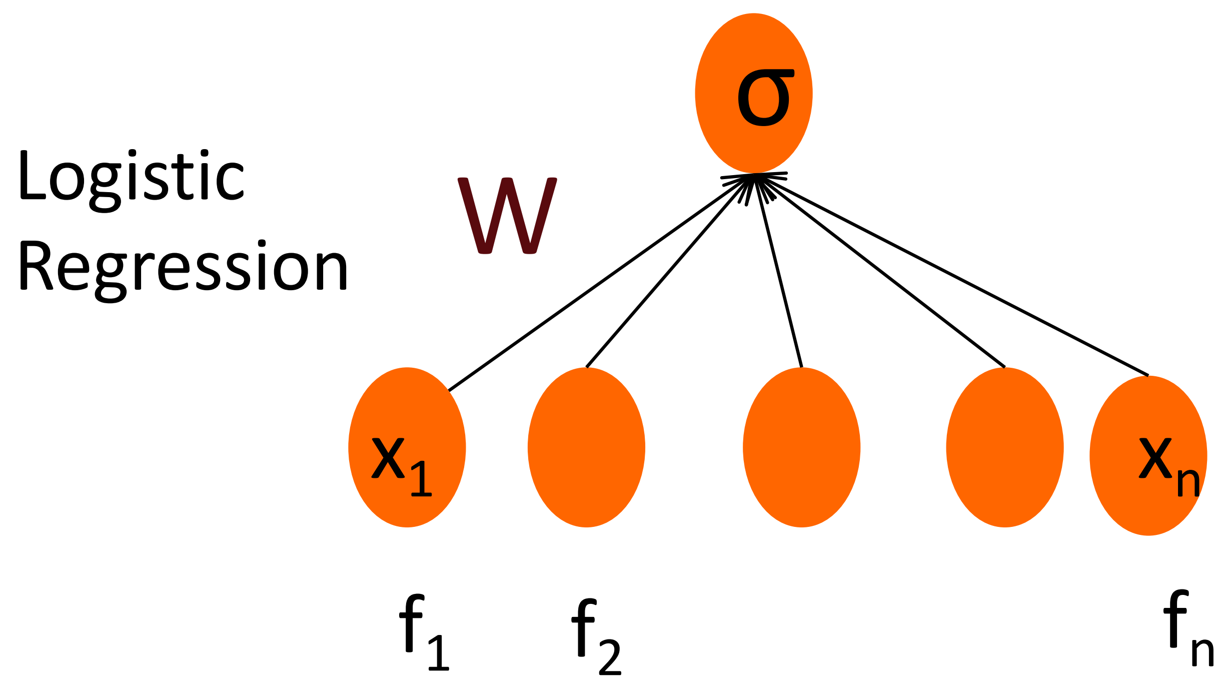
Output layer is 0 or 1

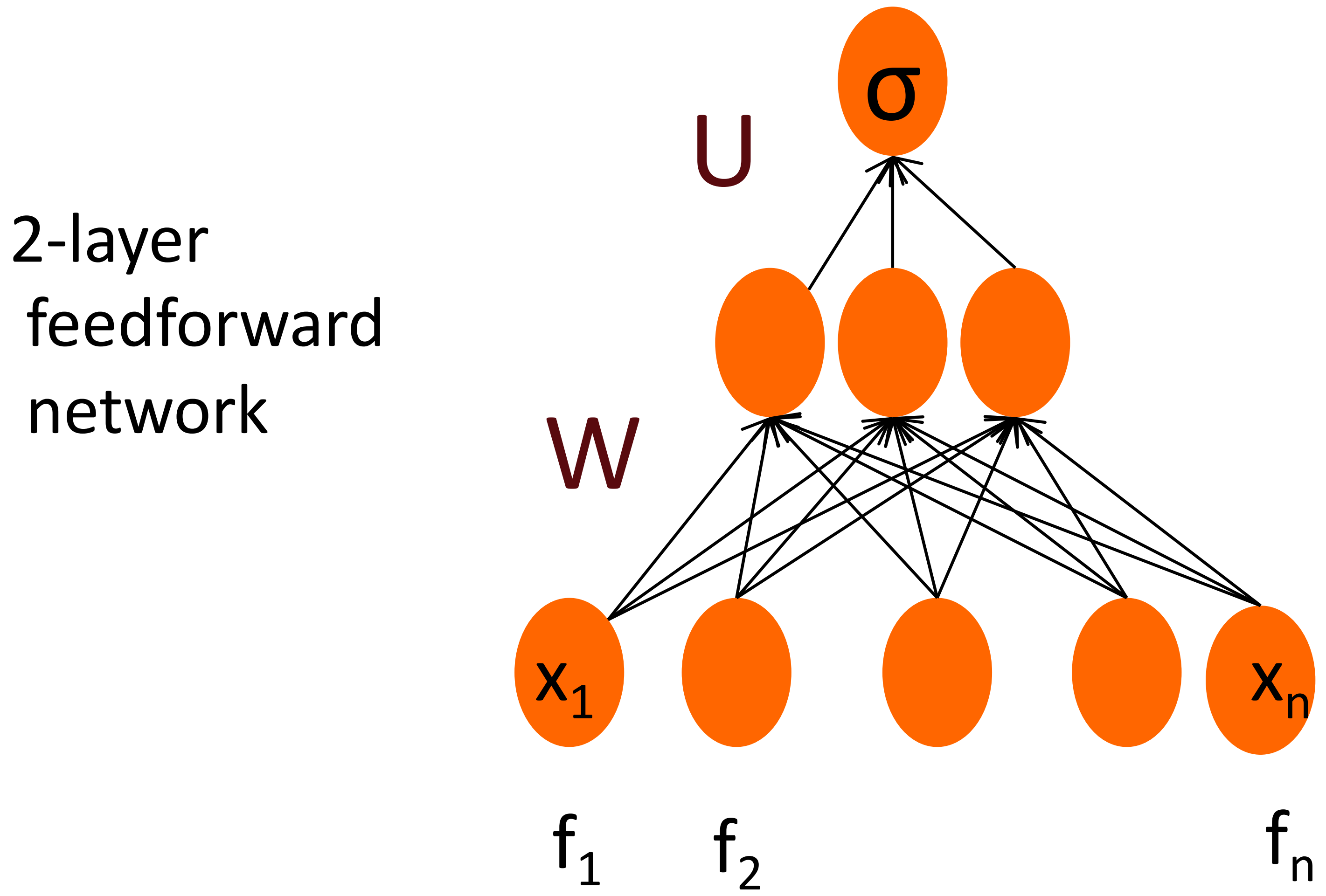


Sentiment Features

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





Just adding a hidden layer to logistic regression

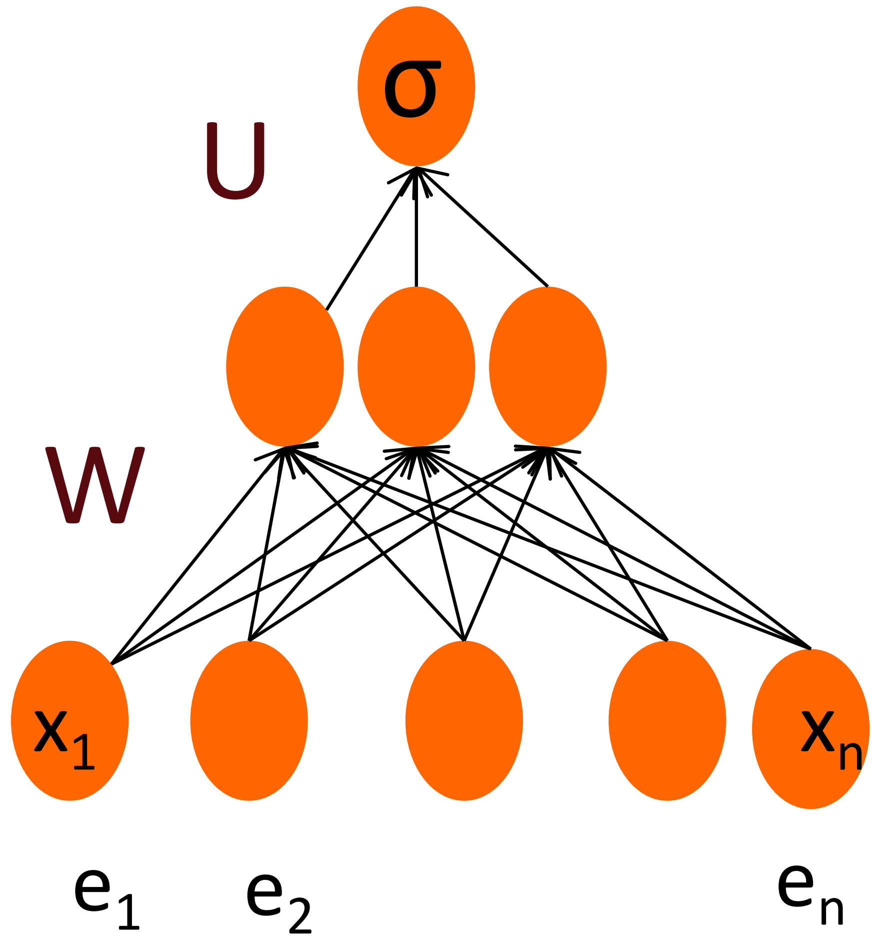
- allows the network to use non-linear interactions between features
- which may (or may not) improve performance.

Even better: representation learning

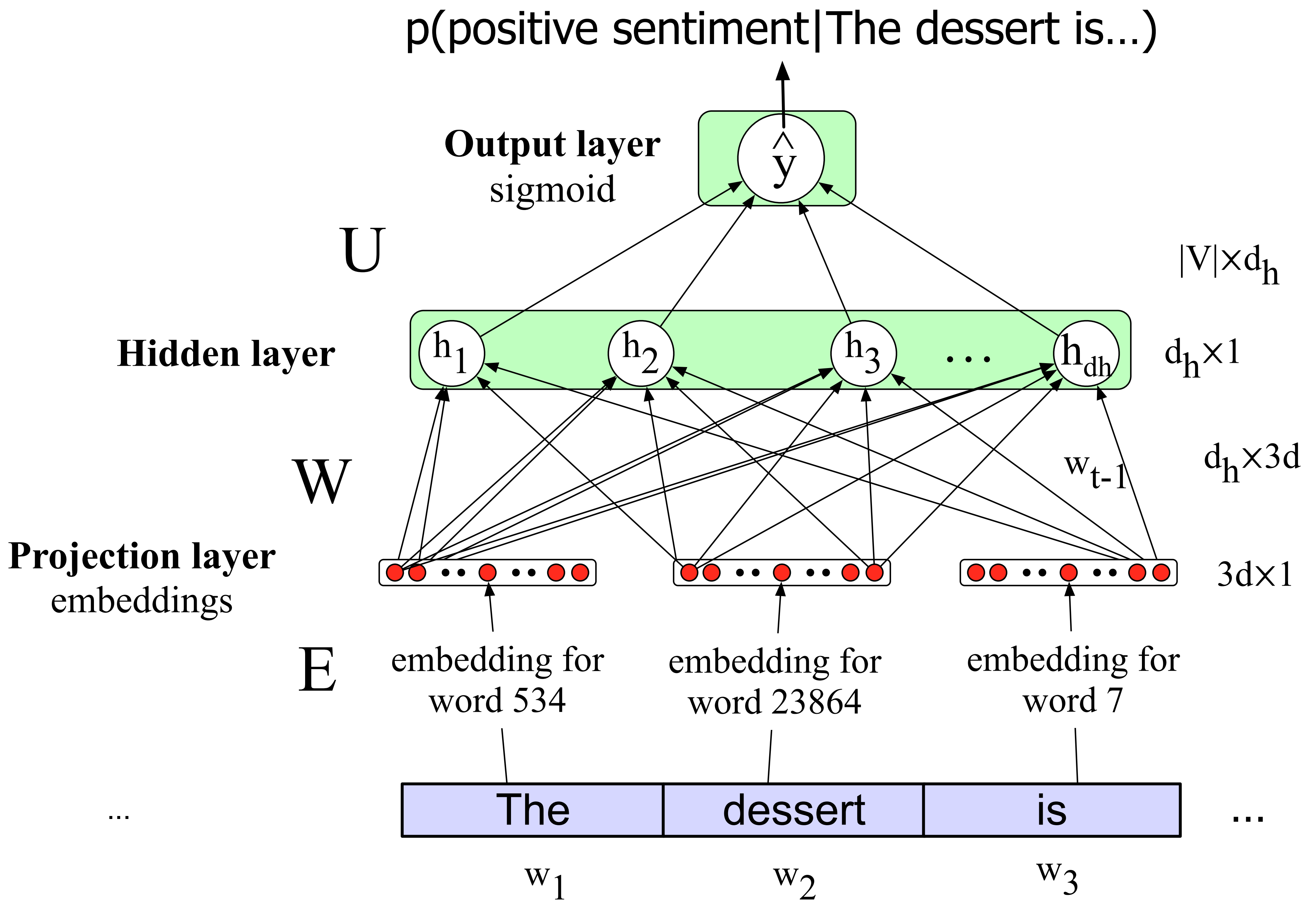
The real power of deep learning comes from the ability to **learn** features from the data

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

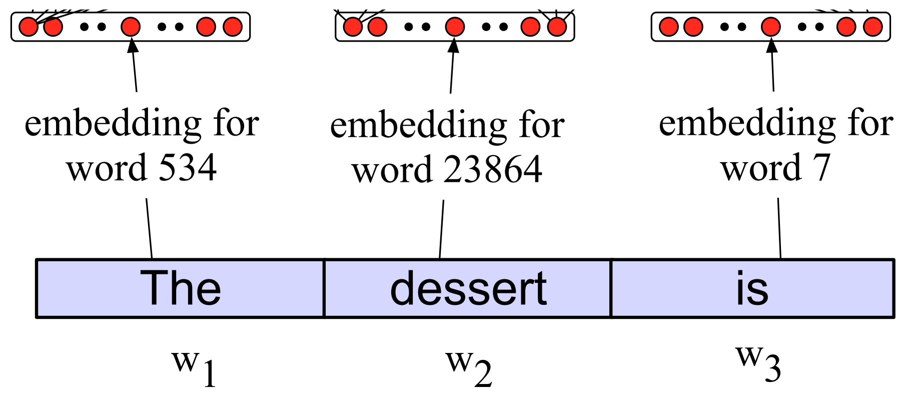
Use learned representations like embeddings!



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)

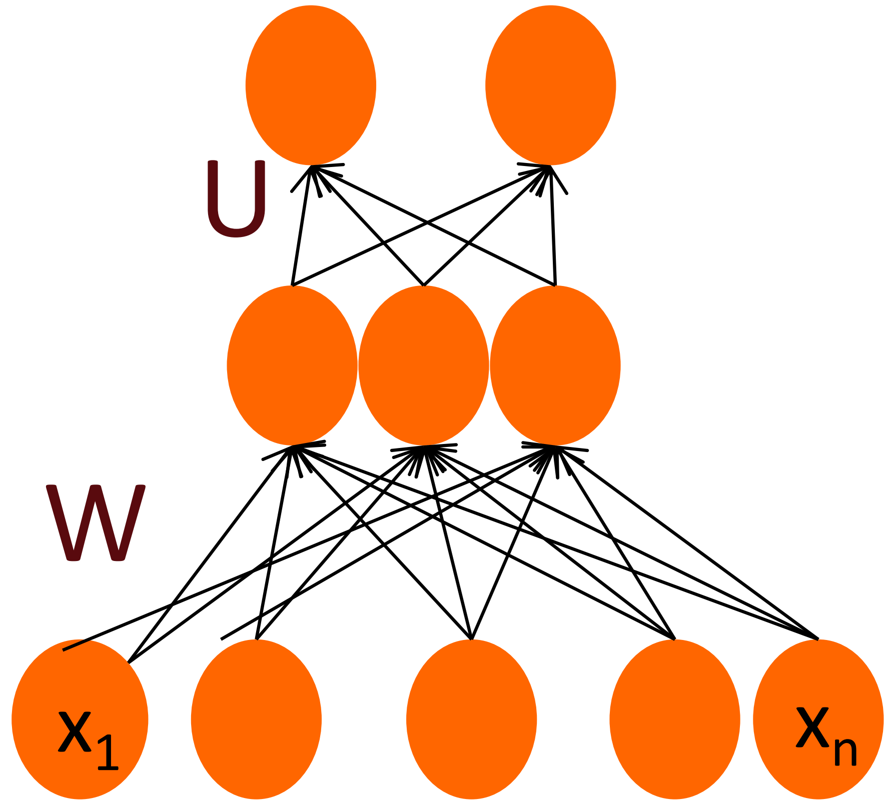
1. Make the input the length of the longest review
   - If shorter then pad with zero embeddings
   - Truncate if you get longer reviews at test time
2. Create a single "sentence embedding" (the same dimensionality as a word) to represent all the words
   - Take the mean of all the word embeddings
 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

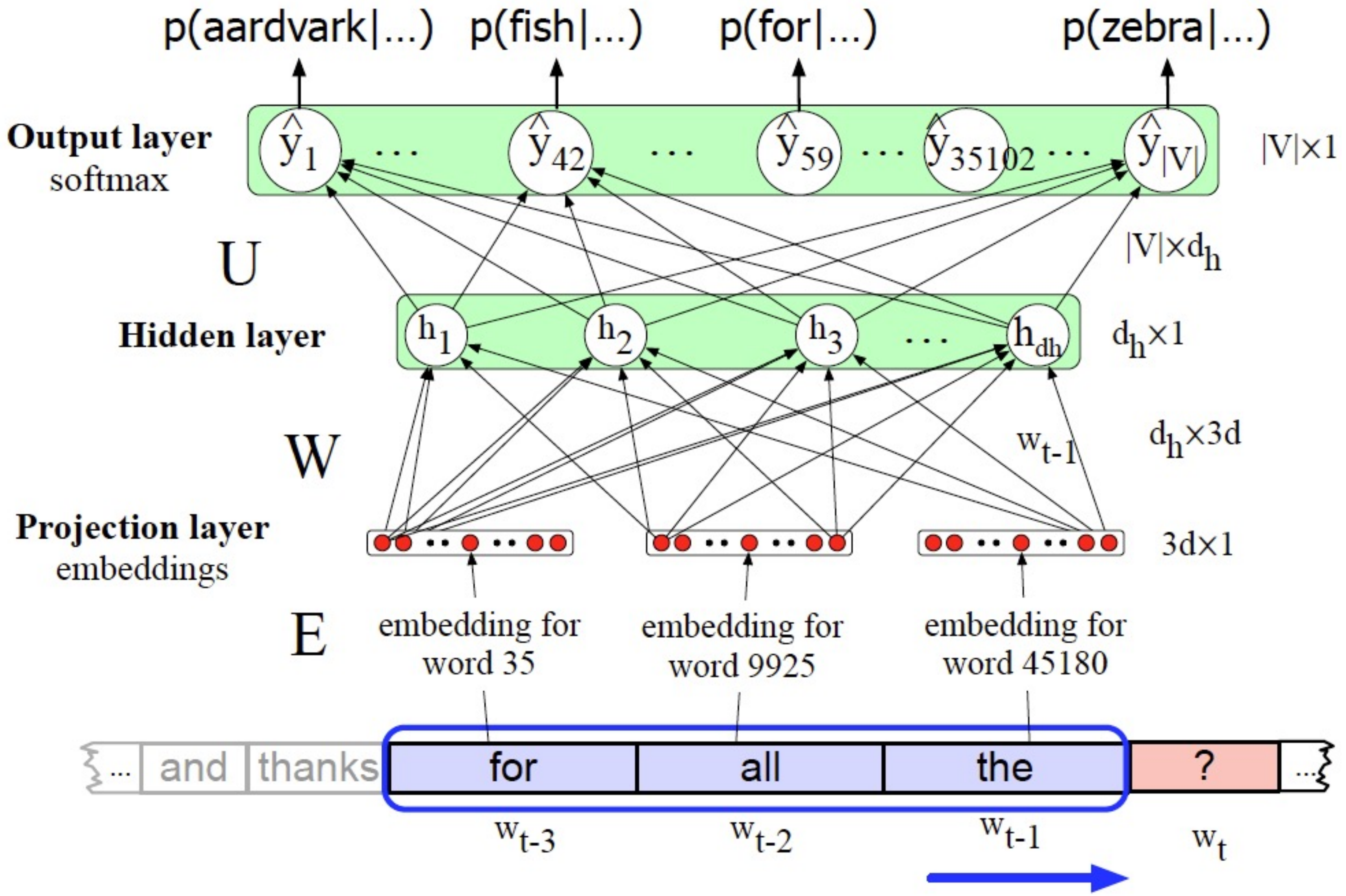
given prior words w_{t-1}, w_{t-2}, w_{t-3}, …

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

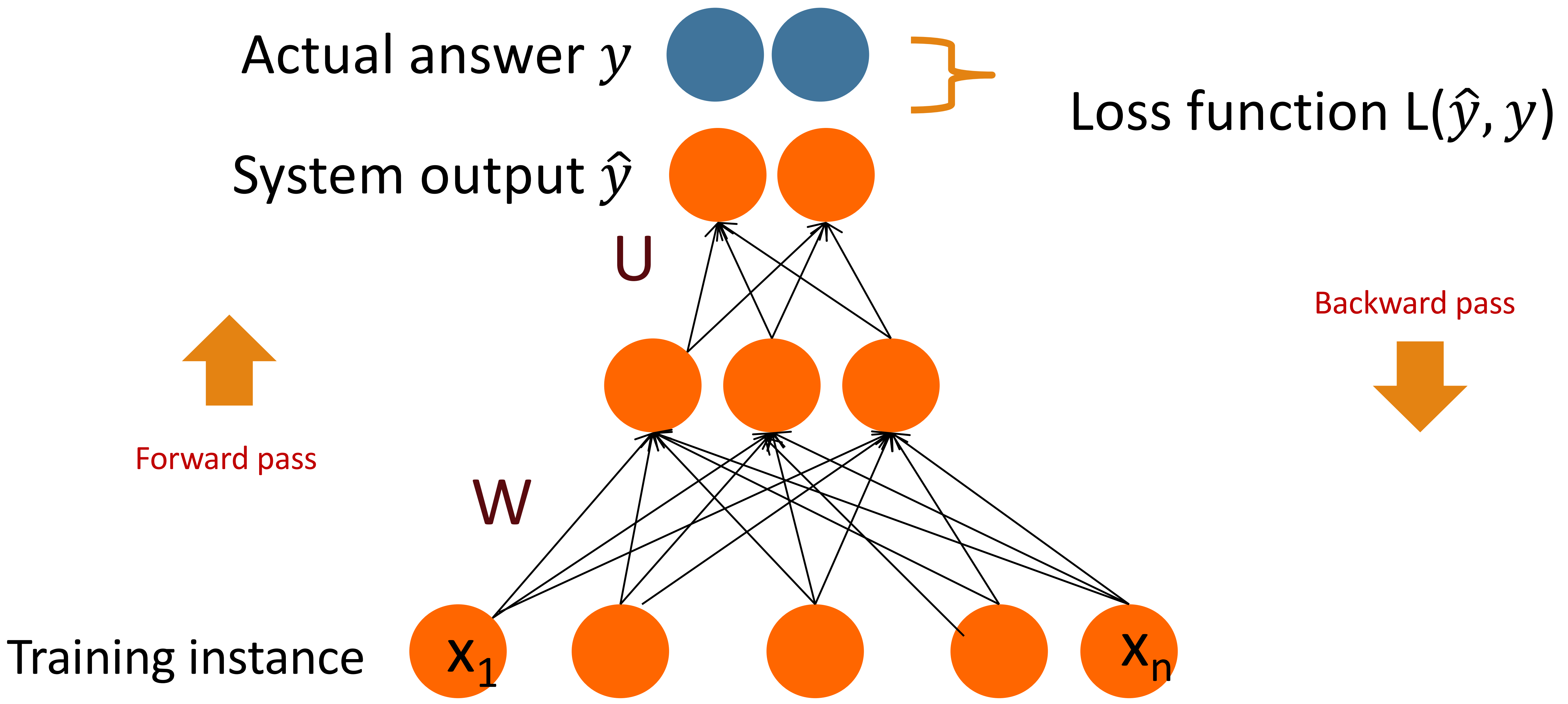
Simple Neural Networks and Neural Language Models

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:

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

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

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

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

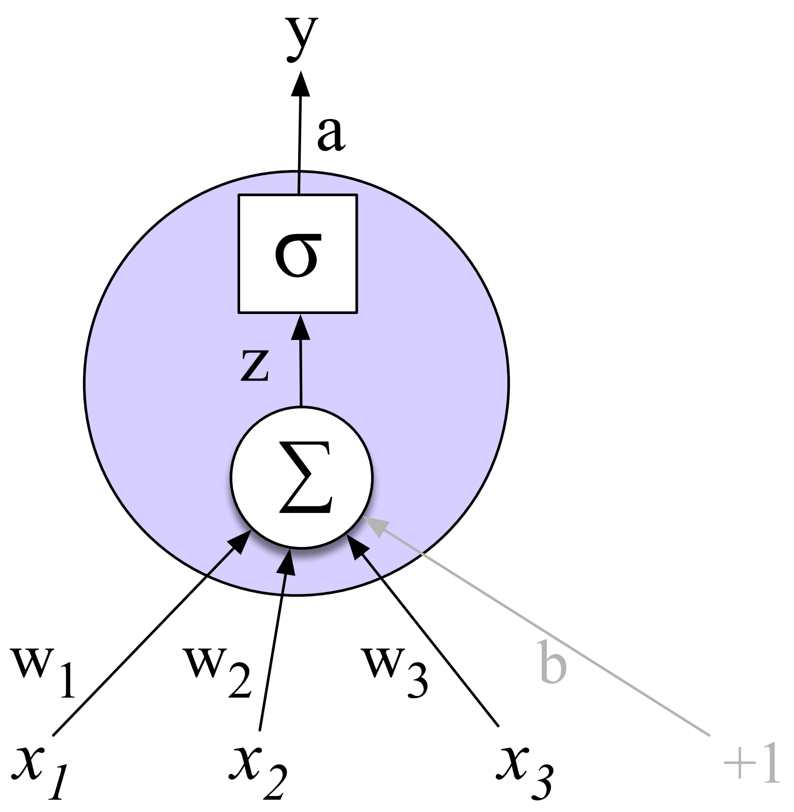
- For logistic regression

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

Where did that derivative come from?

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

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**.

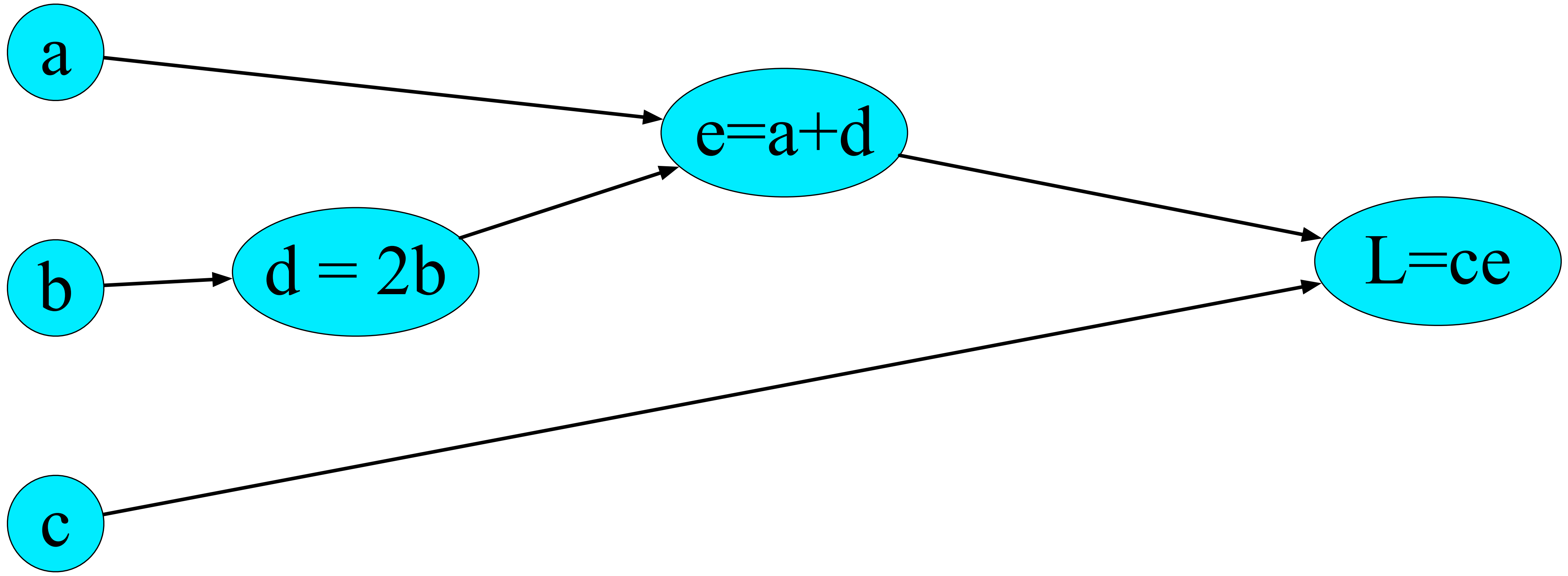
Computation Graphs

A computation graph represents the process of computing a mathematical expression

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

Computations:

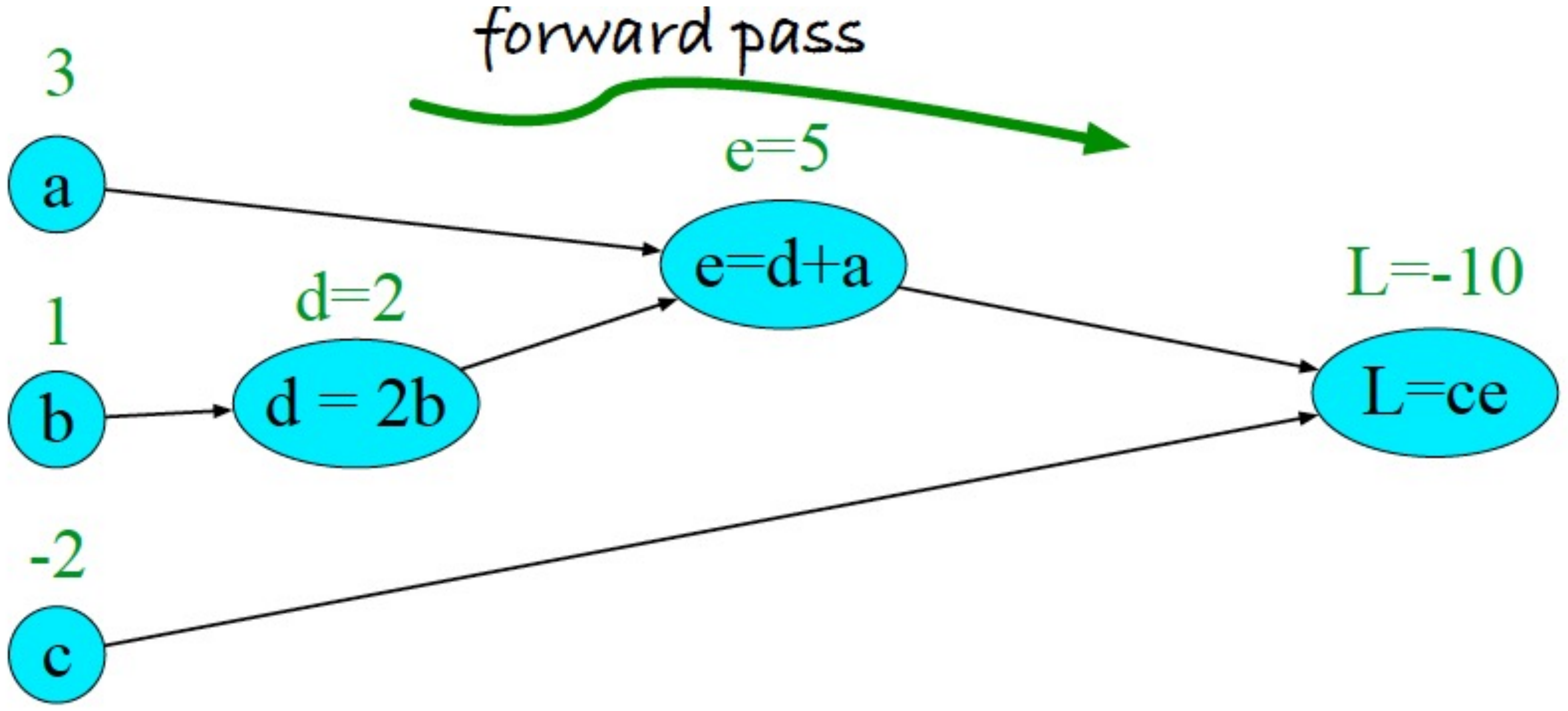
$$\begin{aligned} d &= 2*b \\ e &= a+d \\ L &= c*e \end{aligned}$$



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

Computations:

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



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

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

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

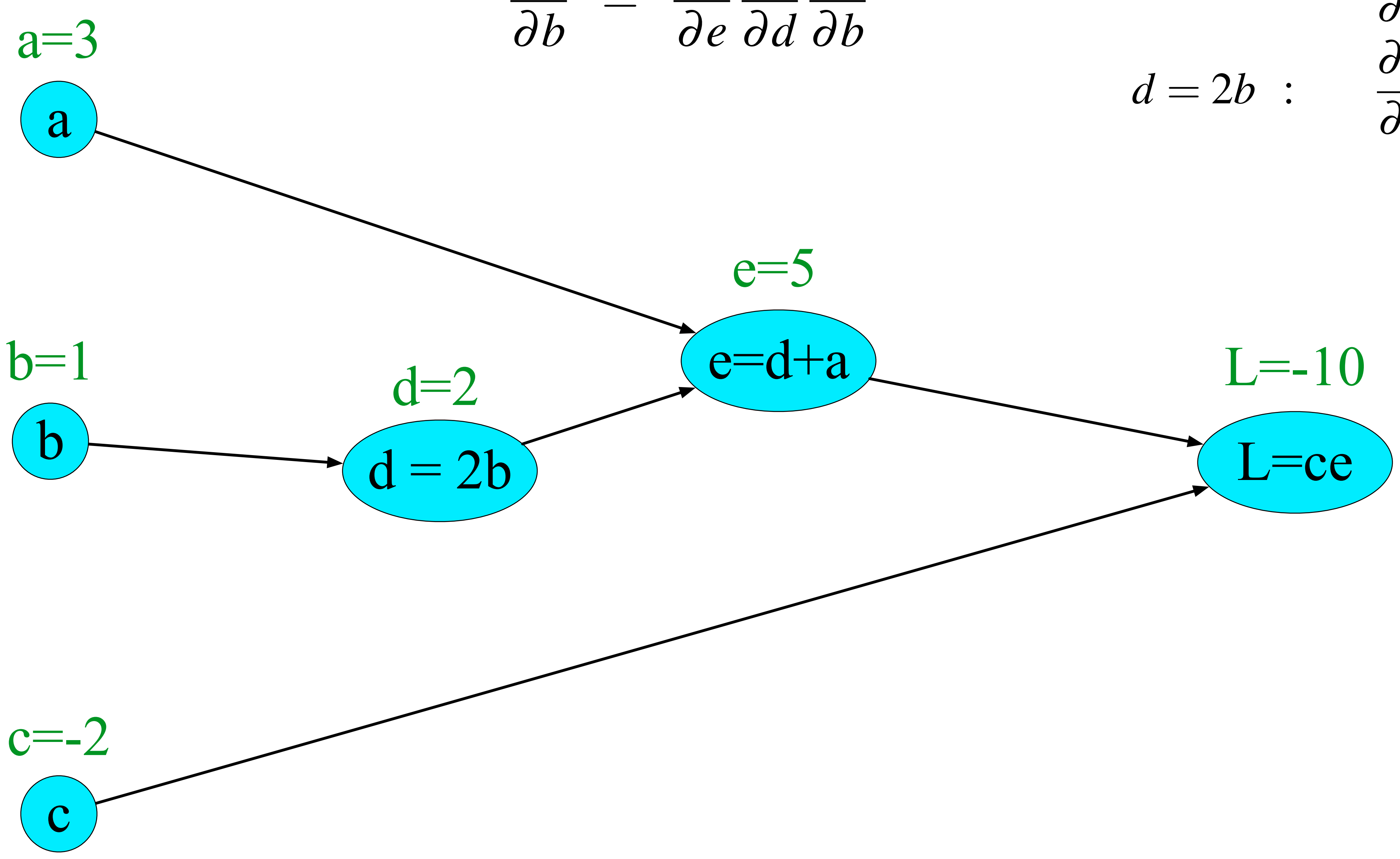
$$\begin{aligned} \frac{\partial L}{\partial a} &= \frac{\partial L}{\partial e}\frac{\partial e}{\partial a} \\ \frac{\partial L}{\partial b} &= \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b} \end{aligned}$$

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

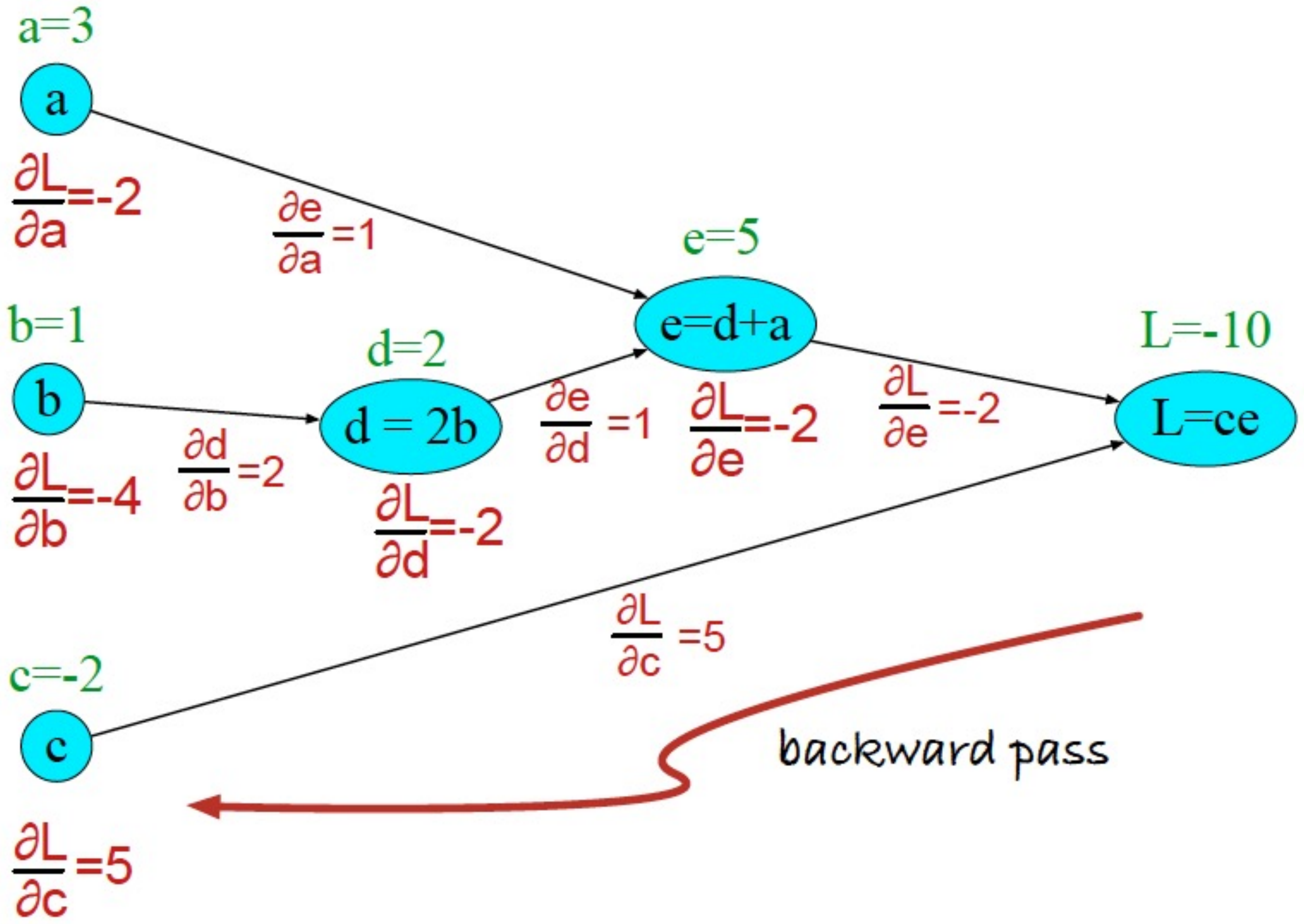
Example

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

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

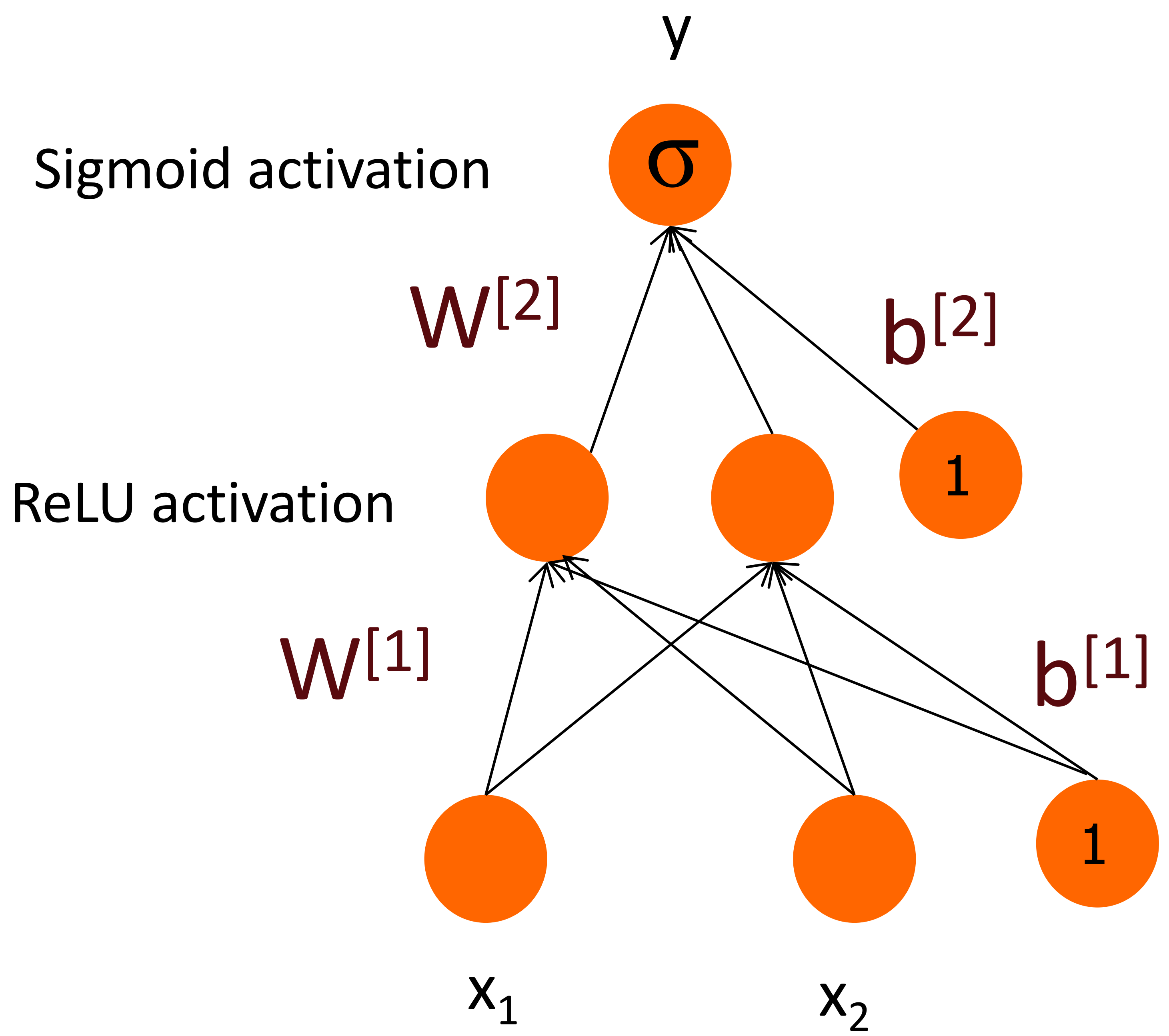


Example



a=3
a
$\frac{\partial L}{\partial a}$=-2
$\frac{\partial e}{\partial a}$=1
e=5
e=d+a
b=1
b
d=2
d = 2b
$\frac{\partial e}{\partial d}$=1
$\frac{\partial L}{\partial e}$=-2
$\frac{\partial L}{\partial e}$=-2
L=-10
L=ce
$\frac{\partial L}{\partial b}$=-4
$\frac{\partial d}{\partial b}$=2
$\frac{\partial L}{\partial d}$=-2
$\frac{\partial L}{\partial c}$=5
c=-2
c
backward pass
$\frac{\partial L}{\partial c}$=5

Backward differentiation on a two layer network



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

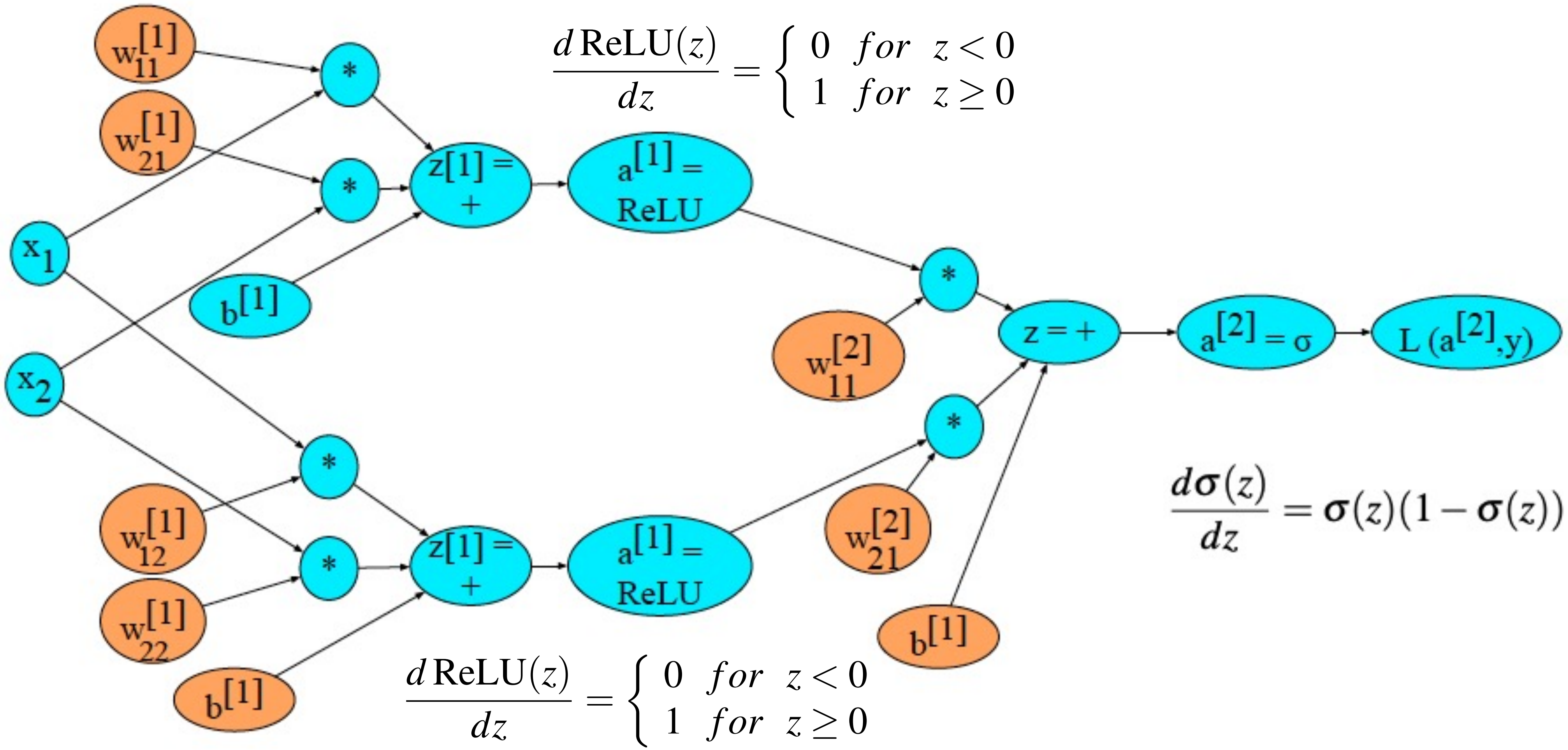
Backward differentiation on a two layer network

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

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

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

Backward differentiation on a 2-layer network



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

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

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

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

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

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

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

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

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

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

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

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

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

Stanford·CS124 | From Languages to Information (2021)

CS124(2021)· 课程资料包 @ShowMeAI

视频	课件	笔记	代码
中英双语字幕	一键打包下载	官方笔记翻译	作业项目解析

视频 ·B 站 [扫码或点击链接]
https://www.bilibili.com/video/BV1YA411w7ym

课件 & 代码 · 博客 [扫码或点击链接]
http://blog.showmeai.tech/cs124/

文本处理 编辑距离 神经嵌入 倒排索引 信息检索 序列标注 PageRank 对话系统 推荐系统 社交网络分析 协同过滤

Awesome AI Courses Notes Cheatsheets 是 ShowMeAI 资料库的分支系列，覆盖最具知名度的 **TOP20+** 门 AI 课程，旨在为读者和学习者提供一整套高品质中文学习笔记和速查表。

点击课程名称，跳转至课程**资料包**页面，**一键下载**课程全部资料！

机器学习	深度学习	自然语言处理	计算机视觉
Stanford · CS229	Stanford · CS230	Stanford · CS224n	Stanford · CS231n
# Awesome AI Courses Notes Cheatsheets· 持续更新中			
知识图谱	**图机器学习**	**深度强化学习**	**自动驾驶**
Stanford · CS520	Stanford · CS224W	UCBerkeley · CS285	MIT · 6.S094

微信公众号

资料下载方式 2：扫码点击底部菜单栏

称为 **AI 内容创作者？** 回复 [添砖加瓦]