

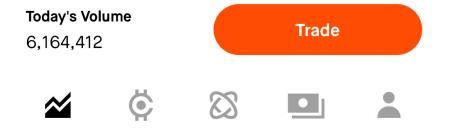
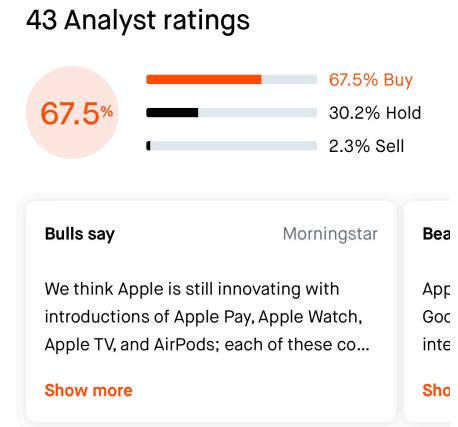
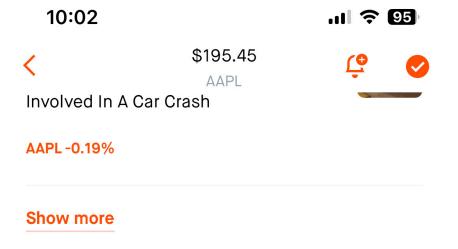


BUS 243

Lecture 7: Introduction to Neural Net

REVIEW HOMEWORK 3

- Part I
- What is an announcement effect?
- Let's consider Robinhood app
- Why not in 10-K?
 - Information need in 10-K?
 - Query?



YELP DATA ANALYSIS

- Original data contains 7M observations
- The given data is 1,000 sample
- Any model should fail
 - Sampling, of course: 0.01 % data
 - Why?
 - Textual data needs large data
 - Why?



- What is the outcome?
 - Integer between 1-5
 - In fact, don't have to convert it into binary
 - NB, logistic regression work well
 - No need other loss function
 - Any concern on ($\text{Stars} \geq 3 \rightarrow 1$) rule?
 - Skewedness?
 - Keep in mind that we are working with SAMPLE
- Ok, then why do need to split the sample into training and test?
 - Very unlikely to have population data in practice
 - Not uncommon to have no test data
 - Assume test data is dataset we've never seen



NORMALIZATION?

- When do we need normalization?
- Consider the workflow when modeling NB
- First, we need to define the vocabulary
 - Tokenize every single sentence and collect the unique token from the **Training** data
 - Why do we need to think about the size of vocabulary?
 - Consider how to treat the tokens not used in test data
- See the most frequent words in the longest sentence: stopping words
 - Understand well without them in the reviews
 - Tense mattered here?
 - Plural terms?
 - Unique words such as the name of POI would be a problem. But not many...



NAÏVE BAYES VS. LOGISTIC REGRESSION

- We learned that Naïve Bayes is the least preferred, but most evaluation metric showed the opposite
- Mathematically, it is quite obvious
 - Too limited observations compared to the number of features
 - Most features are weak
 - Why this is the problem?
- How about hand-written rules? Why does it look the least preferred?
 - Model is wrong (rule is wrong)
 - Even assuming the model is correct, feature has measurement error



- Let's think about two cases

- Complete list of pos and neg words
- Train the model with population data

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) \in doc	3
x_2	count(negative lexicon) \in doc	2
x_3	$\begin{cases} 1 & \text{if "no" } \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns \in doc)	3
x_5	$\begin{cases} 1 & \text{if "!" } \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	$\log(\text{word count of doc})$	$\ln(66) = 4.19$



NEURAL NET FRENZY

- You may hear “AI, Deep learning, Neural net” almost everywhere
- Why?
 - Is it better than any type of traditional approach?
- Neural Net is just a class of models
 - Input, output, and parameters
 - Ok, a lot of parameters...



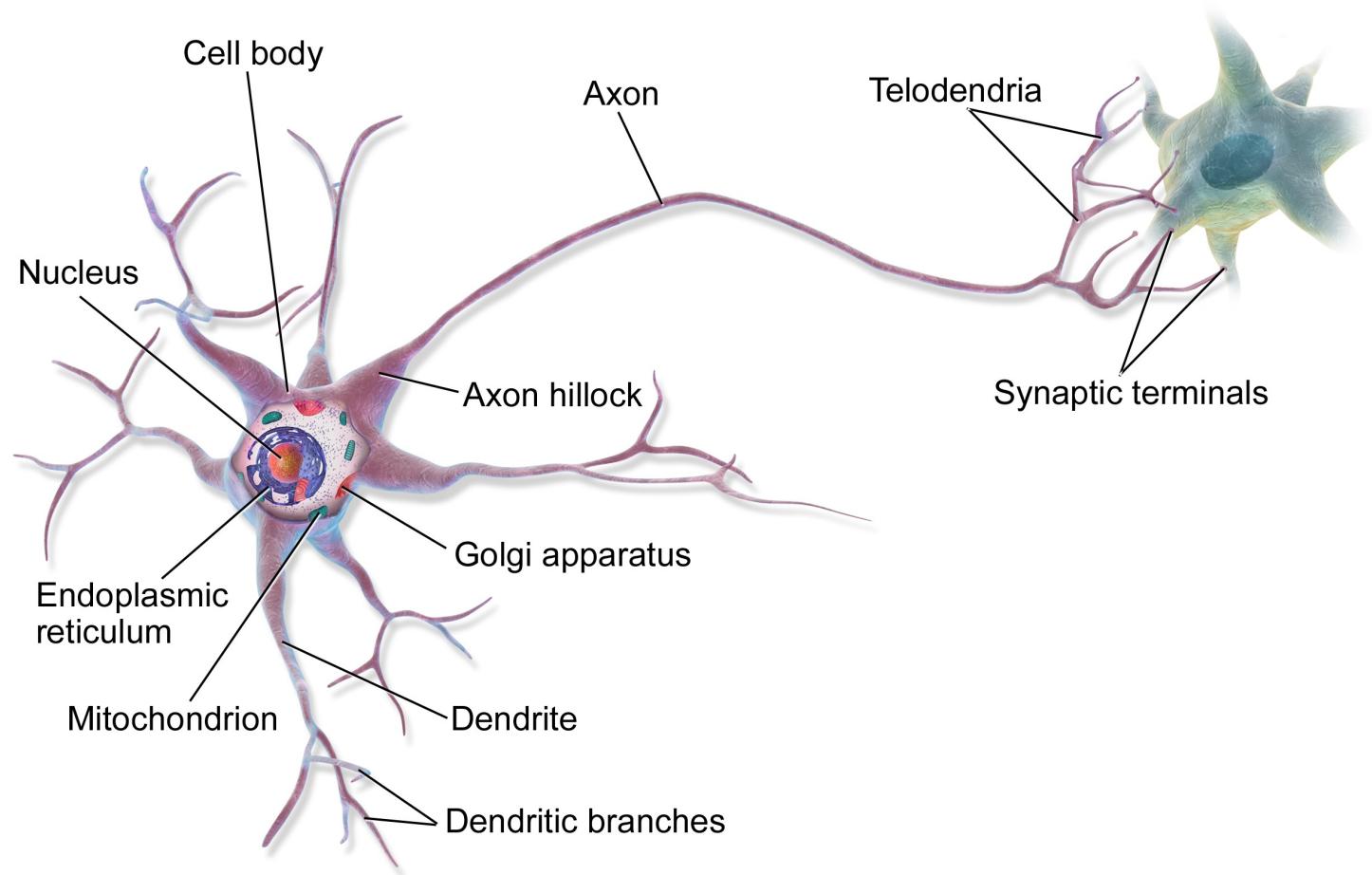
NEURAL NET VS. LOGISTIC REGRESSION

- Experience and domain knowledge are essential for effective feature engineering
 - Labor intensive approach



What is a better model?





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

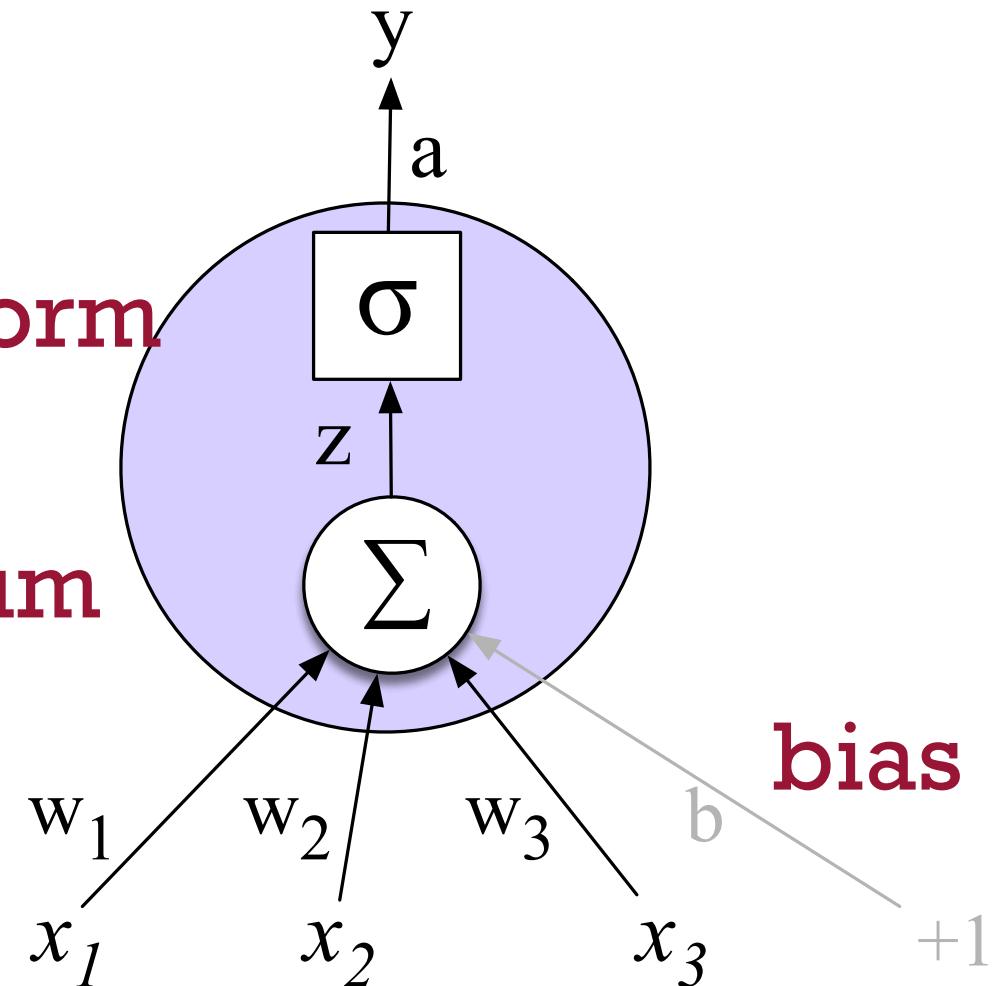
Output value

Non-linear transform

Weighted sum

Weights

Input layer



A NEURAL UNIT

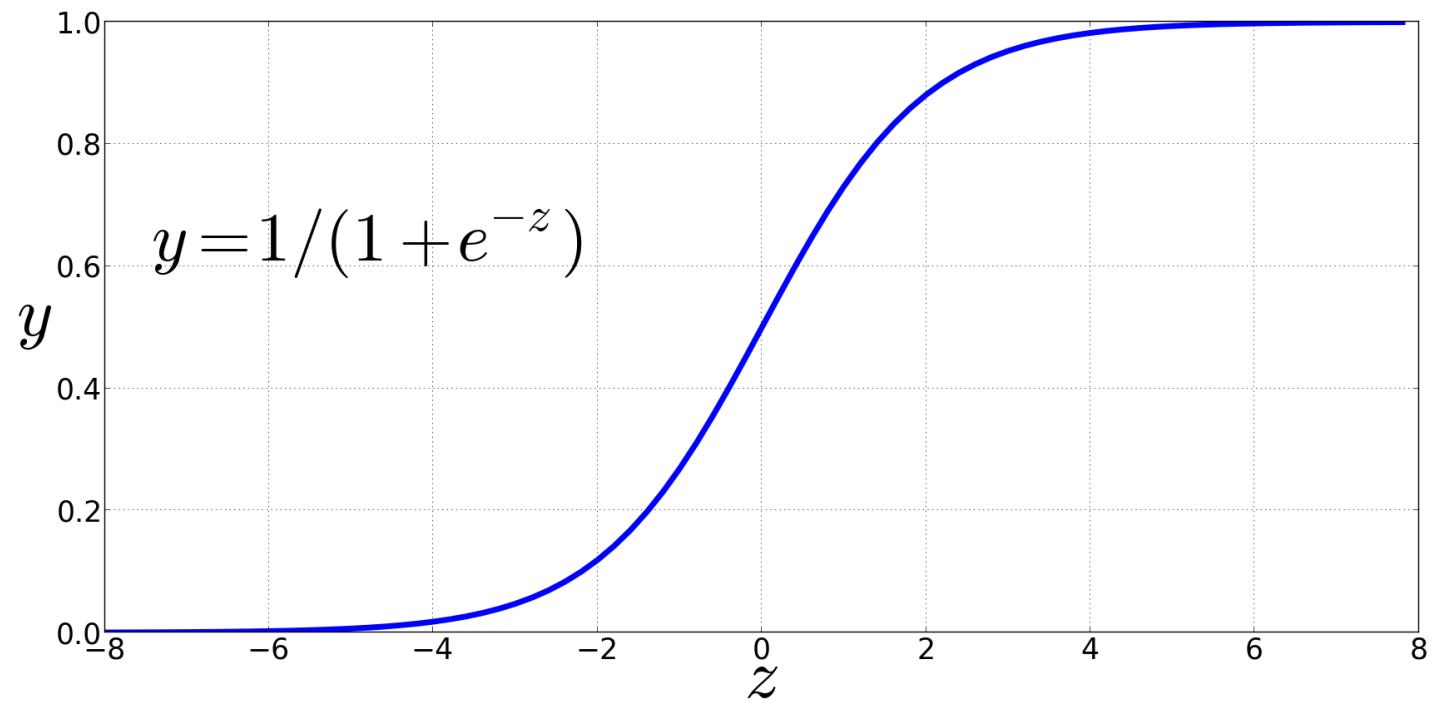
- Take weighted sum of inputs, plus a bias
 - $z = b + \sum_i w_i x_i = b + w \cdot x$
- Then, we apply a nonlinear activation function f :
 - $y = f(z)$
- There are a range of f



NON-LINEAR ACTIVATION FUNCTIONS

Sigmoid

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



FINAL FUNCTION THE UNIT IS COMPUTING

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

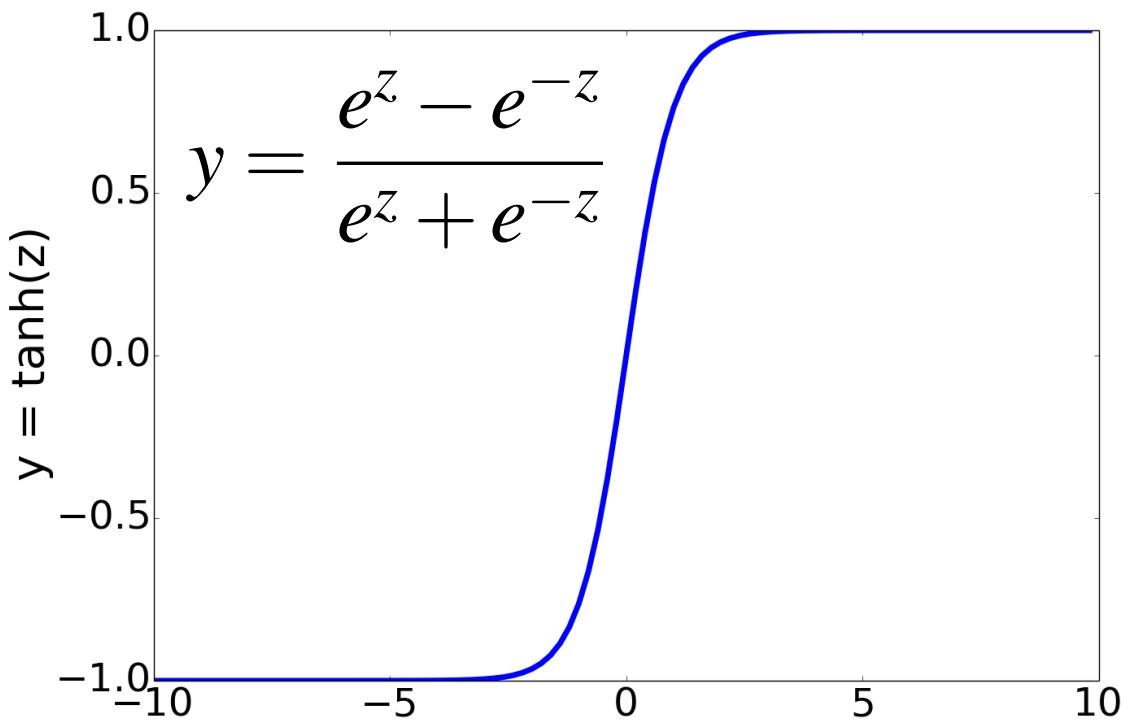


EXAMPLE

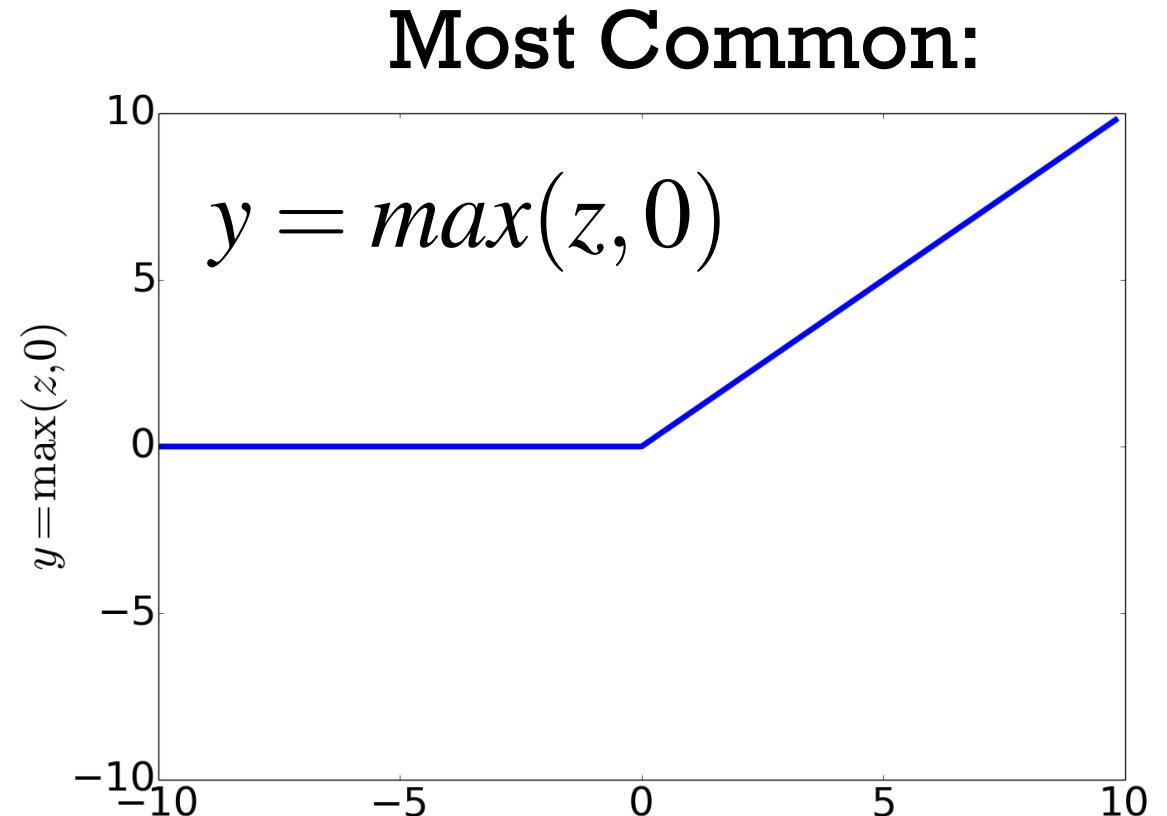
- Suppose a unit has:
 - $w = [0.2, 0.3, 0.9]$
 - $b = 0.5$
- What happens with input x :
 - $x = [0.5, 0.6, 0.1]$
 - $y = \sigma(w \cdot x + b) = \frac{1}{1+e^{-0.87}} = 0.70$



OTHER NON-LINEAR ACTIVATION FUNCTIONS



tanh



ReLU
Rectified Linear Unit

ACTIVATION FUNCTIONS

- Each functions have different properties
 - The tanh() is smoothly differentiable and maps outliers toward the mean
 - Preferred to the sigmoid
 - ReLU produces linear-like results
- The choice of activation function can indeed be influenced by the tasks the neural network is designed to perform
 - But **no** linear activation function!



MULTILAYER NETWORKS

- The strength of neural networks lies in their ability to combine individual units into larger networks
 - Called multi-layer networks (rarely used)
- Linear models including a single neural unit cannot compute XOR problems

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



PERCEPTRON

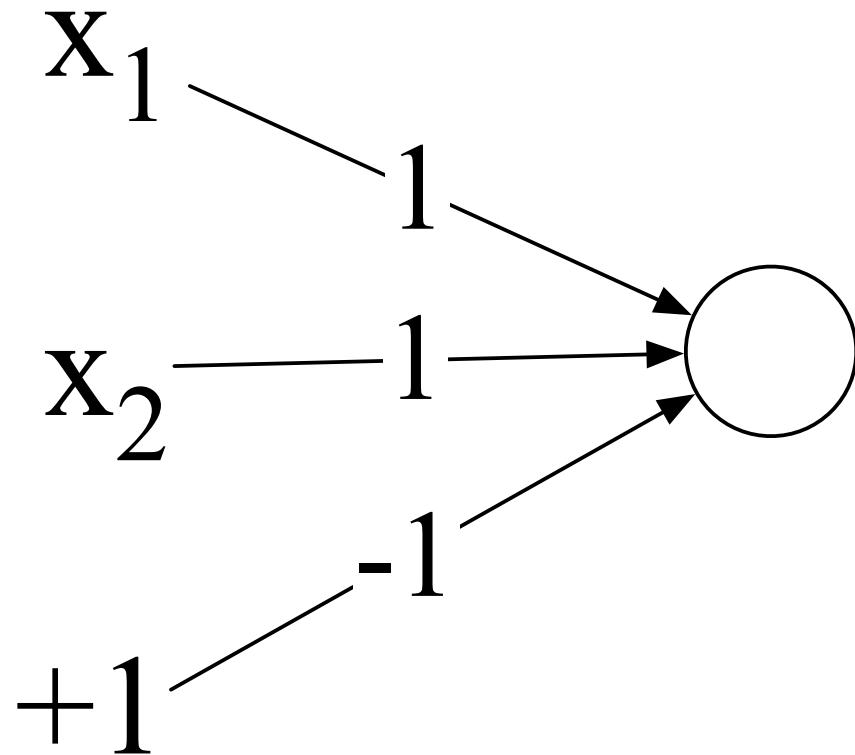
- Consider the perceptron, a very simple neural unit that has a binary output with a step activation function

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

- Easy to build AND or OR with perceptrons



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

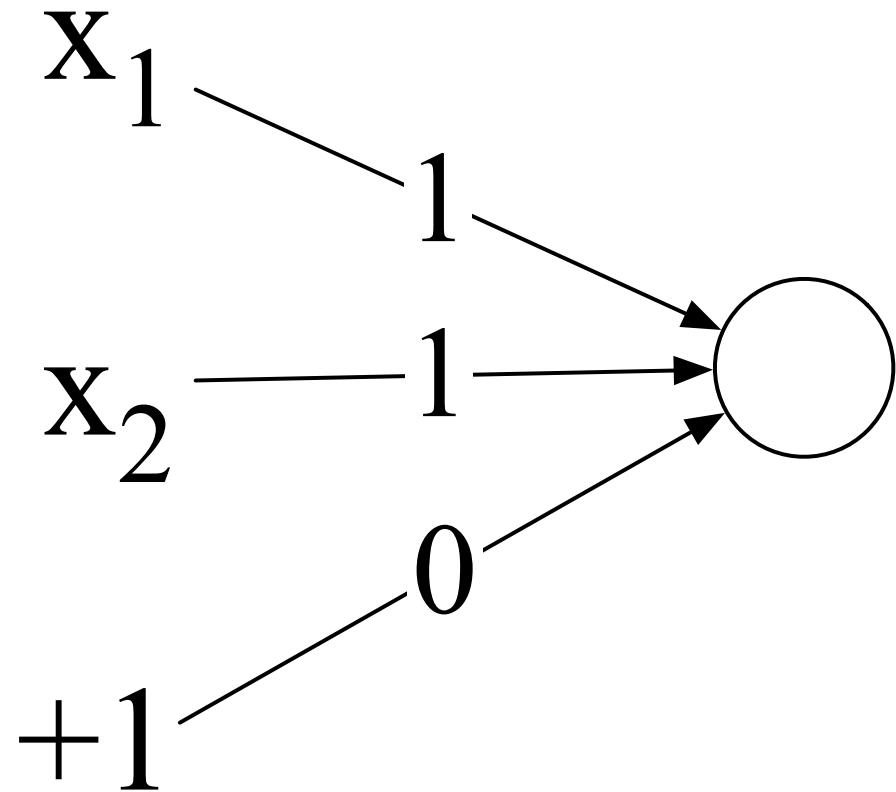


AND

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



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



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



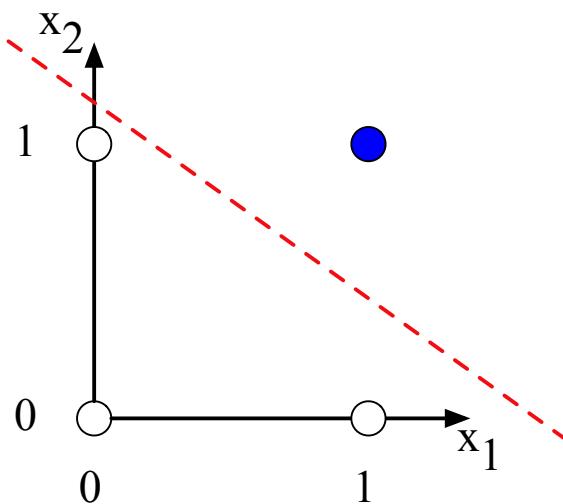
XOR PROBLEM

- Not possible to capture XOR with perceptrons
 - Try it!
- Given x_1, x_2 , perceptron equation is a line
 - $w_1x_1 + w_2x_2 + b = 0$
- This line acts as a decision boundary

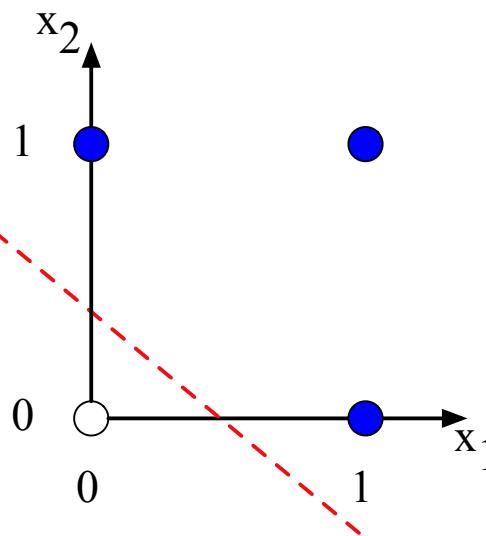


DECISION BOUNDARIES

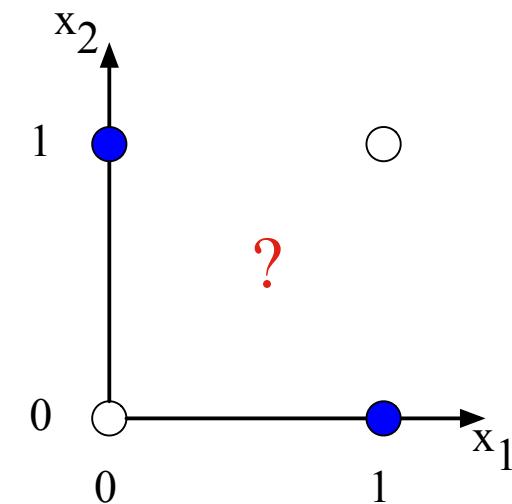
- XOR is not linearly separable function



a) x_1 AND x_2



b) x_1 OR x_2



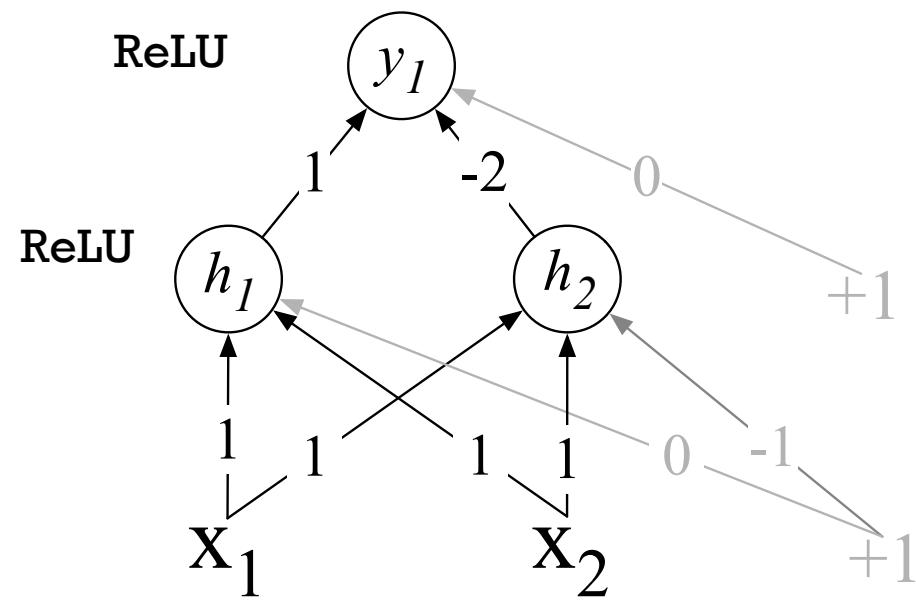
c) x_1 XOR x_2



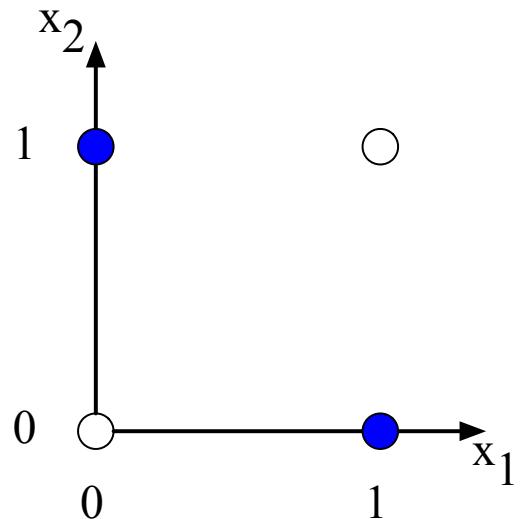
SOLUTION TO THE XOR PROBLEM

- XOR can't be calculated by a single perceptron
- XOR can be calculated by a layered network of units

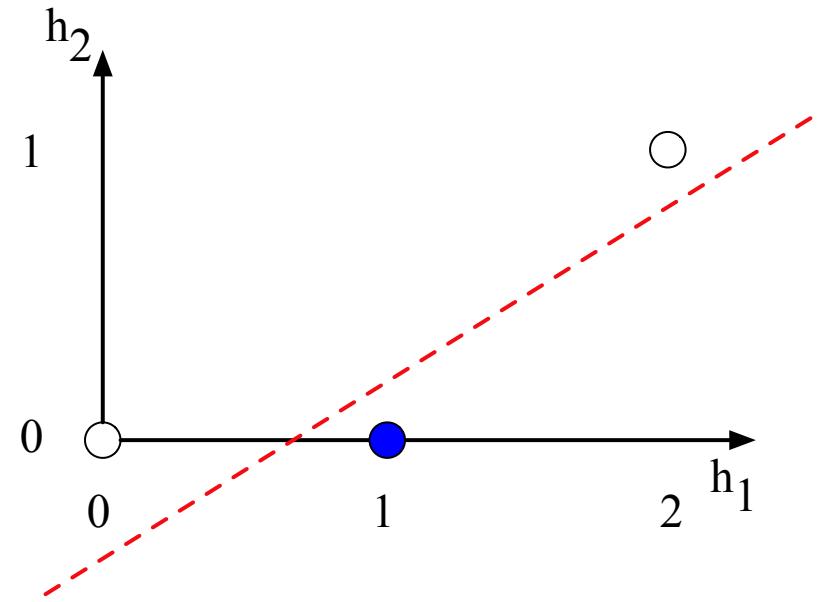
XOR		y
x1	x2	
0	0	0
0	1	1
1	0	1
1	1	0



- We can view the hidden layer of the network as forming a representation of the input



a) The original x space



b) The new (linearly separable) h space

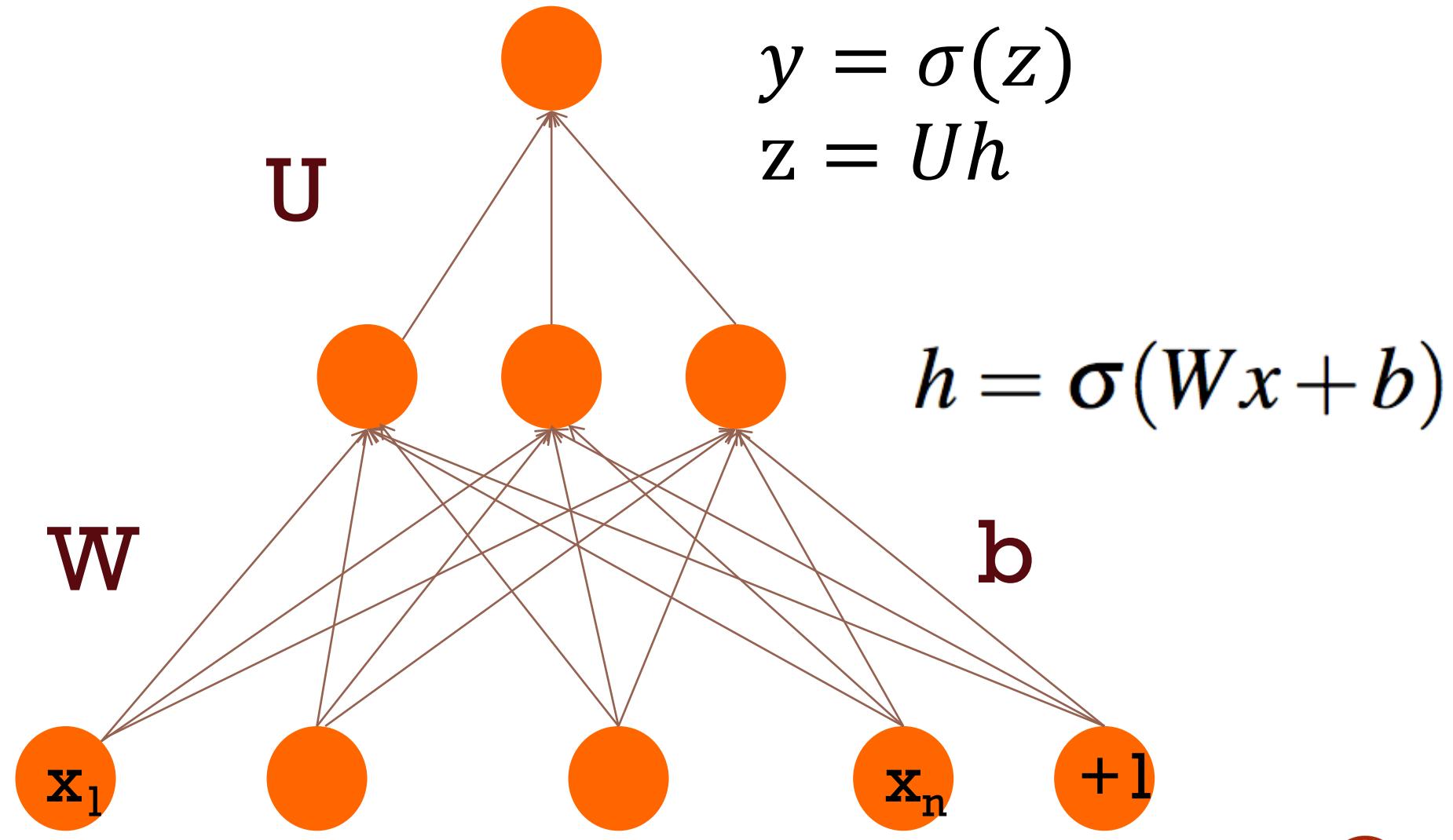
- The hidden layers will learn to form useful representations



Output
layer
(σ node)

hidden units
(σ node)

Input layer
(vector)



FEEDFORWARD NEURAL NETWORKS

- The feedforward network is the simplest NN
 - Fully connected **with no cycles**
- Minimum requirement: input units, hidden units, output units
 - Layers, nodes, units used interchangeably
- The core is the hidden layer



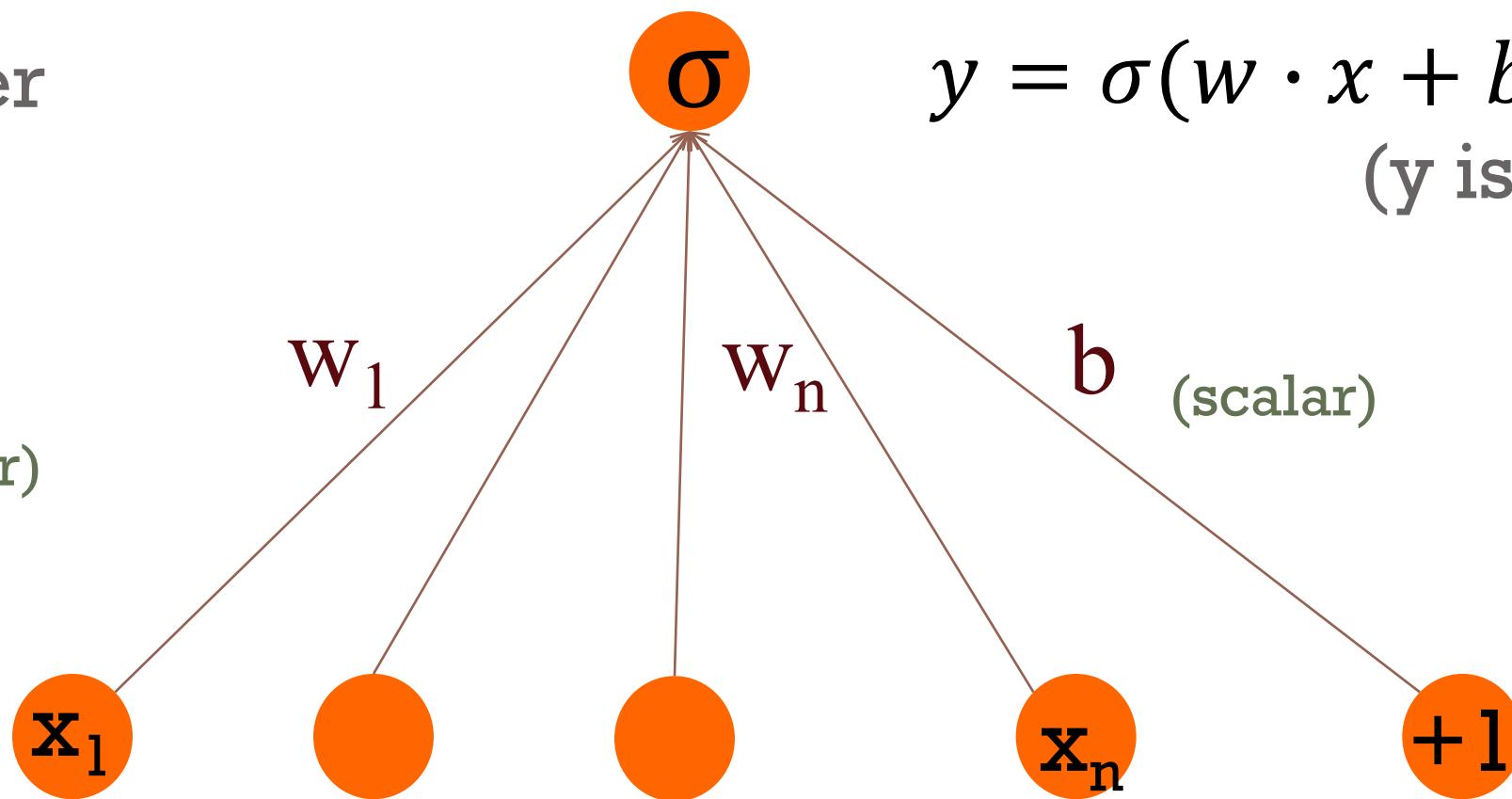
BINARY LOGISTIC REGRESSION AS A 1-LAYER NETWORK

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

Output layer
(σ node)

w
(vector)

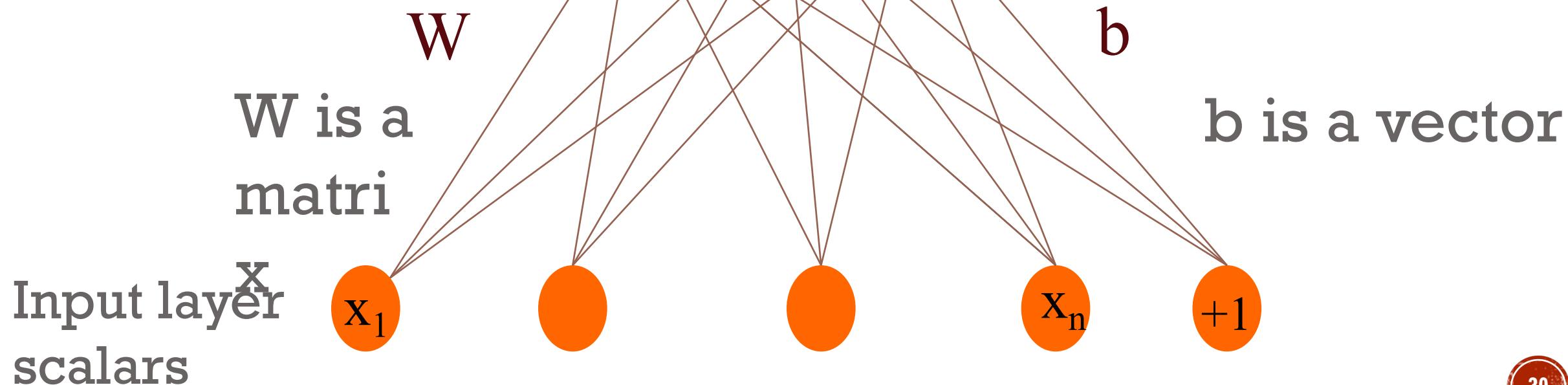
Input layer
vector x



MULTINOMIAL LOGISTIC REGRESSION AS A 1-LAYER NETWORK

Fully connected single layer network

Output layer
(softmax nodes)



Input layer
scalars

SOFTMAX: A GENERALIZATION OF SIGMOID

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

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

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

- Example:

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

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

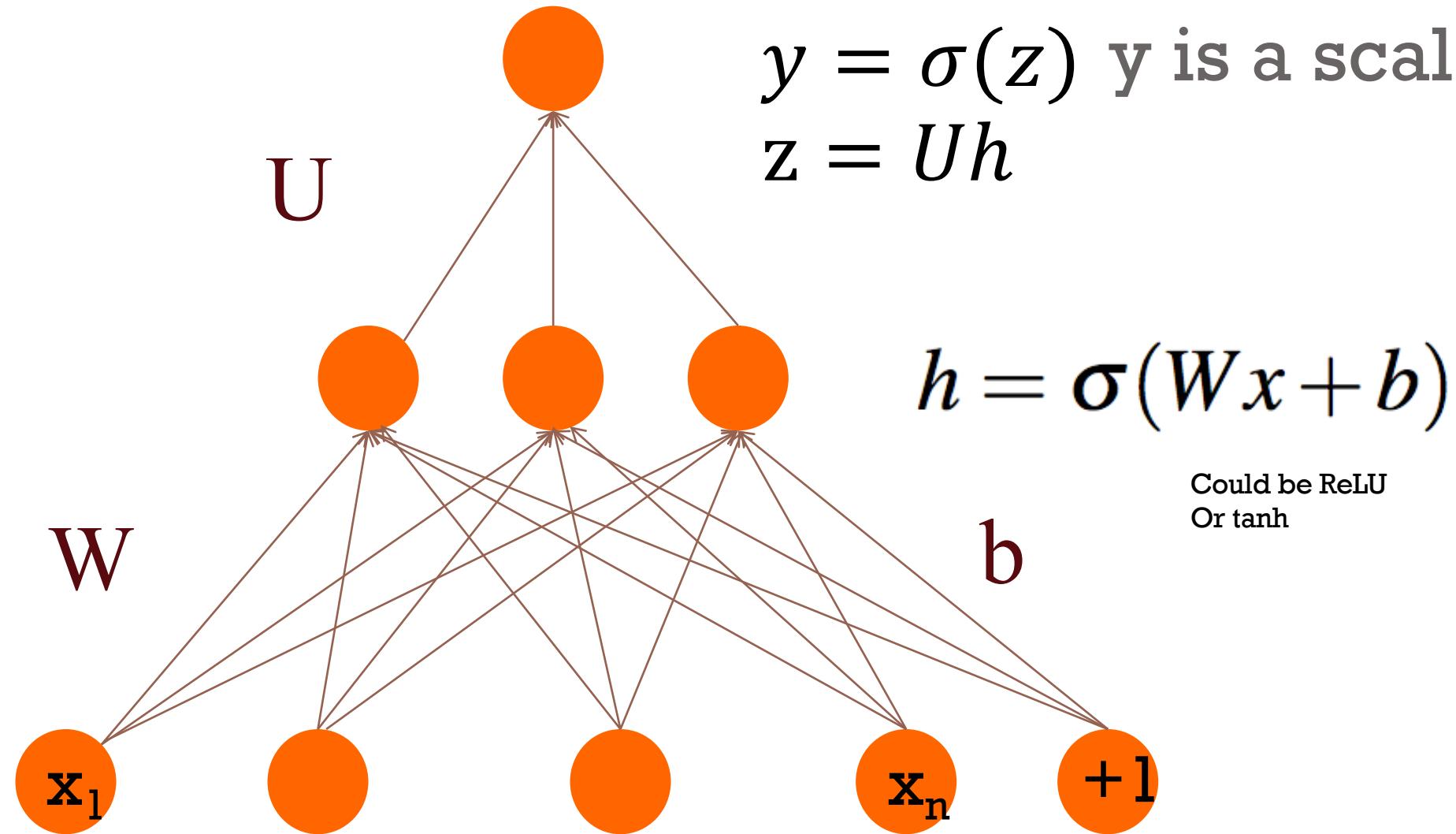


TWO-LAYER NETWORK WITH SCALAR OUTPUT

Output
layer
(σ node)

hidden units
(σ node)

Input layer
(vector)



TWO-LAYER NETWORK WITH SCALAR OUTPUT

Output

layer

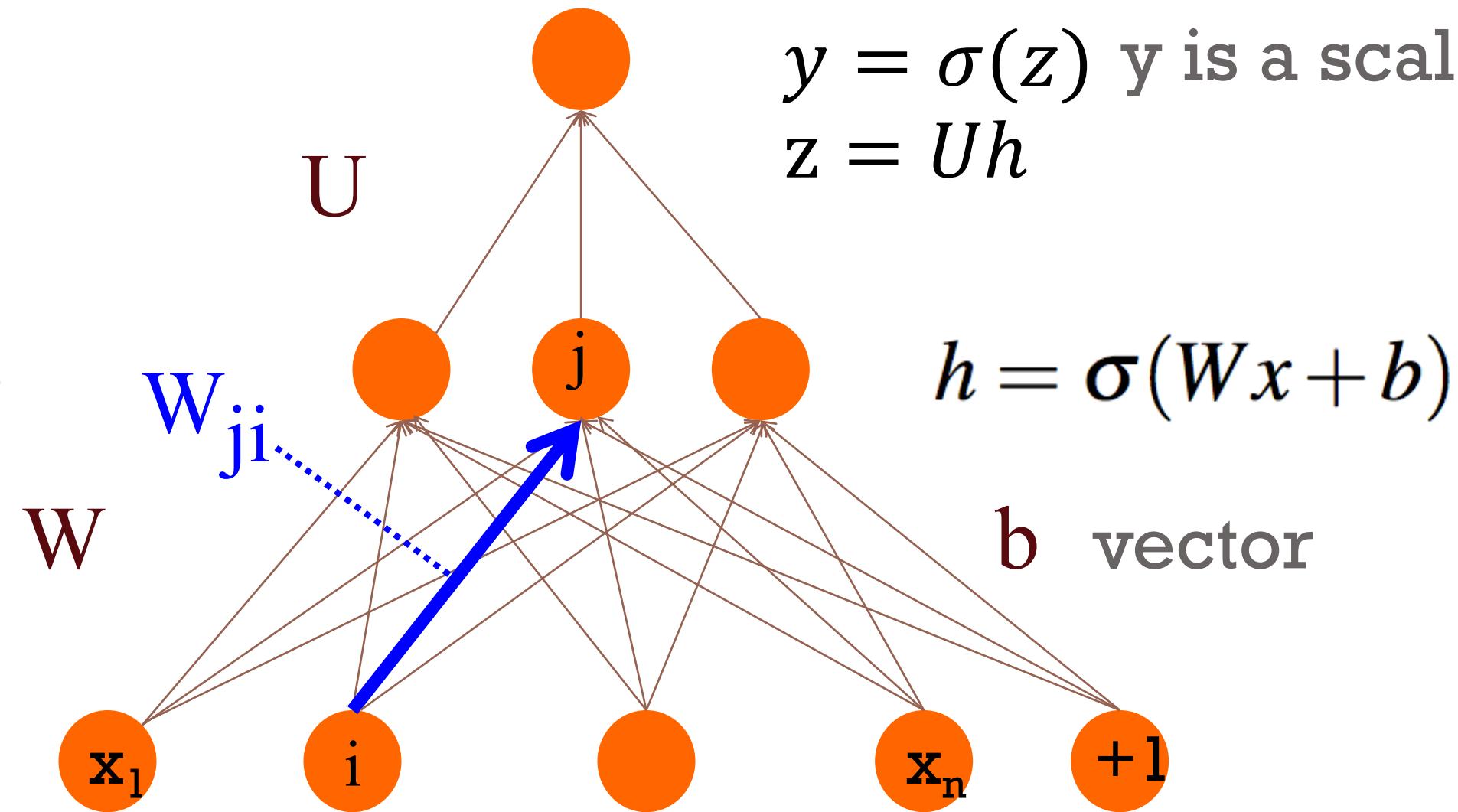
(σ node)

hidden units

(σ node)

Input layer

(vector)



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

$$z = Uh$$

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

b vector

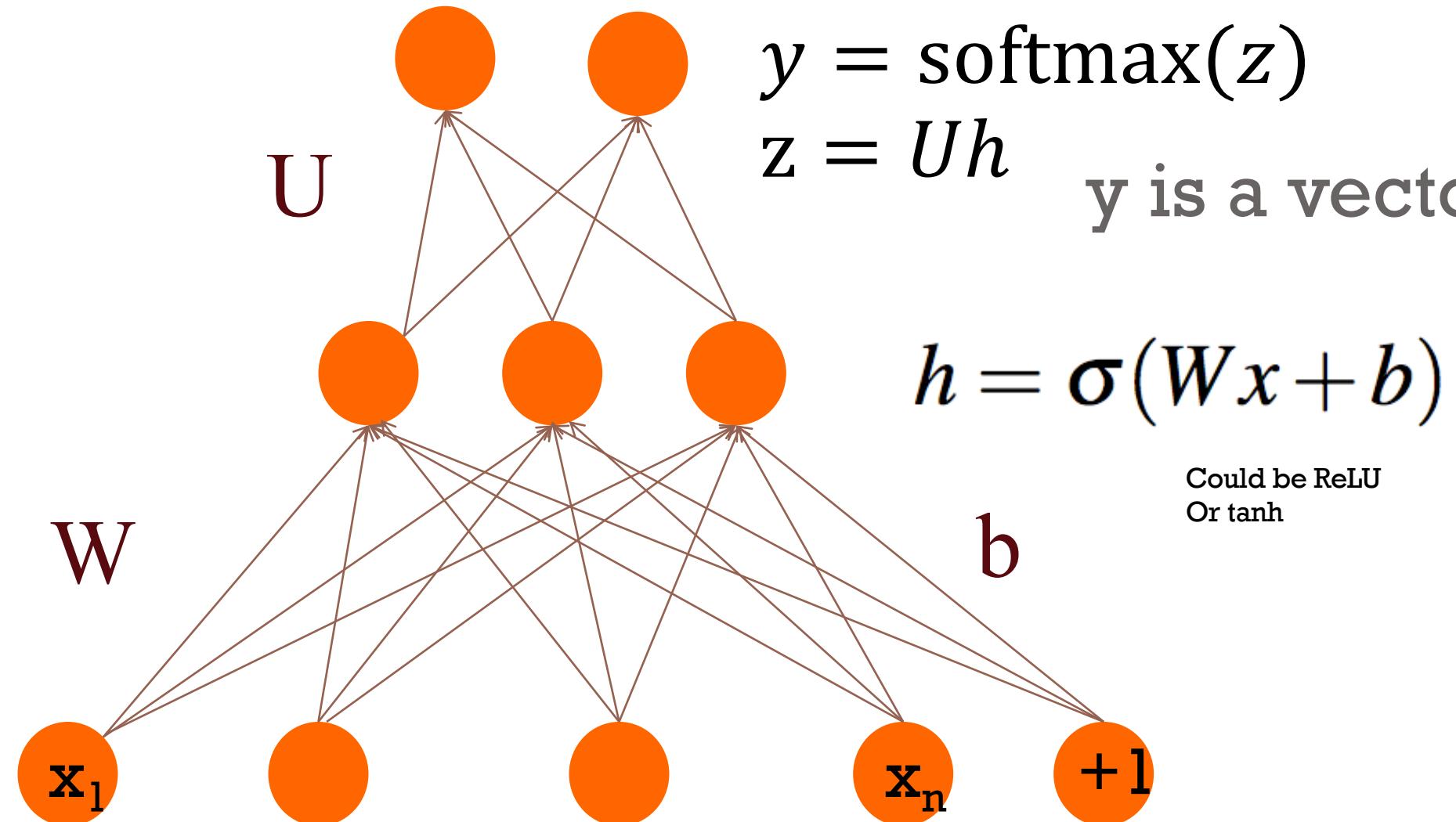


TWO-LAYER NETWORK WITH SOFTMAX OUTPUT

Output
layer
(σ node)

hidden units
(σ node)

Input layer
(vector)



FEEDFORWARD NEURAL NETWORKS

- The resulting value **h** forms a representation of the input
- The role of the output layer is to take this new representation **h** and compute a final output



CLASSIFICATION: SENTIMENT ANALYSIS

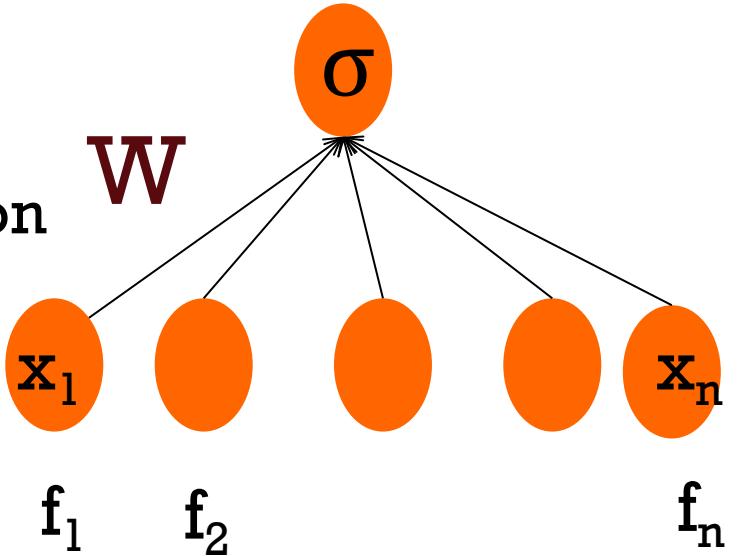
- Consider what we did for logistic regression

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

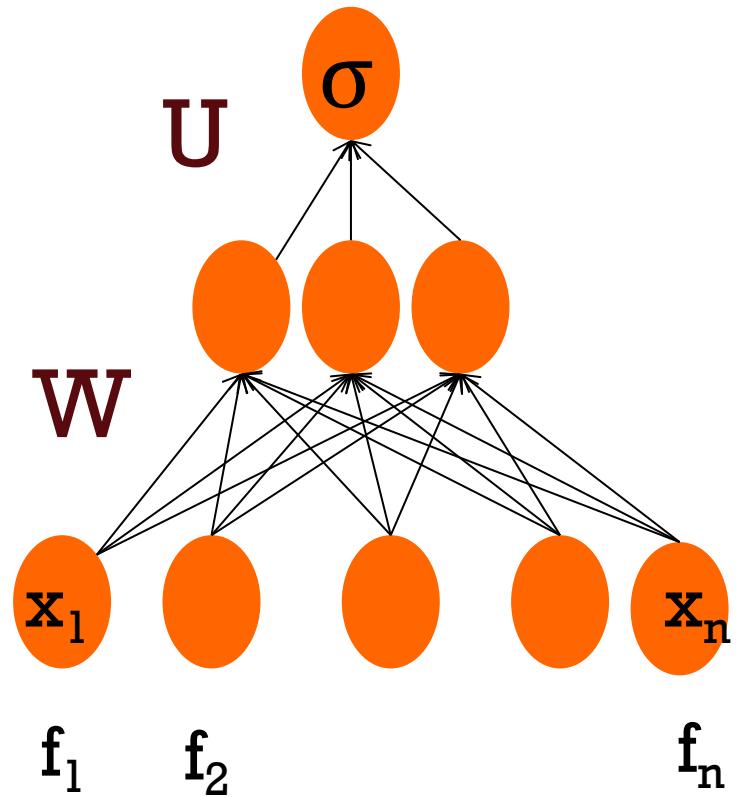


FEEDFORWARD NETS FOR SIMPLE CLASSIFICATION

Logistic
Regression



2-layer
feedforward
network



- Just adding a hidden layer to logistic regression
 - Allow the the network to use non-linear interactions between features
 - Logistic regression assume features are independent
 - May or may not to improve performance

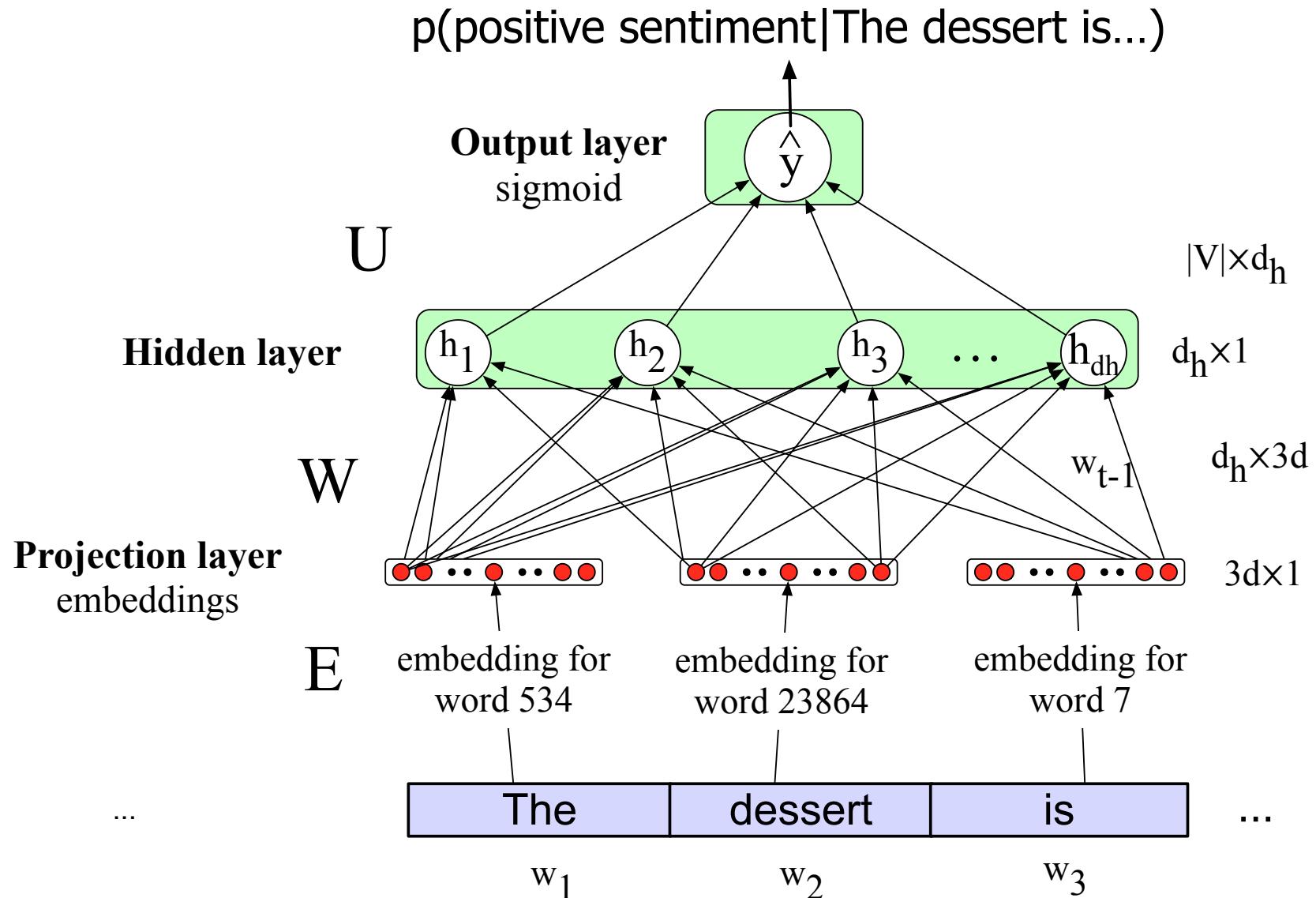


EVEN BETTER: REPRESENTATION LEARNING

- The real power of deep learning comes from the ability to **learn** features from the data
 - Instead of using hand-built features
- Use the word embeddings



NEURAL NET CLASSIFICATION WITH EMBEDDINGS AS INPUT FEATURES!



- This assumes a fixed size length (3), which is unrealistic
- Some solutions
 - Fix the input sentence size
 - If shorter, then pad with zero embeddings
 - Truncate if you get longer reviews at test time
 - Create a single “sentence embedding” to represent all the words in each sentence
 - Take the mean of all the word embedding
 - Take the element-wise max of all the word embeddings

BACKPROPAGATION

- For many years, researchers struggled to find a way to train NN
- Gradient descent was the potential candidate, but it requires computing the gradients of the model's error with regard to the parameters
- Not clear at the time how to do this efficiently with such a complex model containing so many parameters
- In 1970, Linnainmaa proposed the reverse-mode autodiff
 - In just two passes through the network, able to compute the gradients of the NN's error with respect to every single model parameter



- A mini-batch enters the network through the input layer
 - Compute the output of network through every layer: forward pass
 - All intermediate results are preserved
- The algorithm computes the network's output error through a loss
- Then it computes how much each output bias and each connection to the output layer contributed to the error
 - Use chain rule
- Then the algorithm measures how much of these error contributions came from each connection in the layer below, working backward until it reaches the input layer
- Finally, the algorithm performs a gradient descent step to tweak all the connection weights in the network, using the error gradients it just computed.



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

$$d = 2 * b$$

$$e = a + d$$

$$L = c * e$$

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

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

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

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

$$a=3$$

a

$$b=1$$

b

$$c=-2$$

c

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

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

$$e=5$$

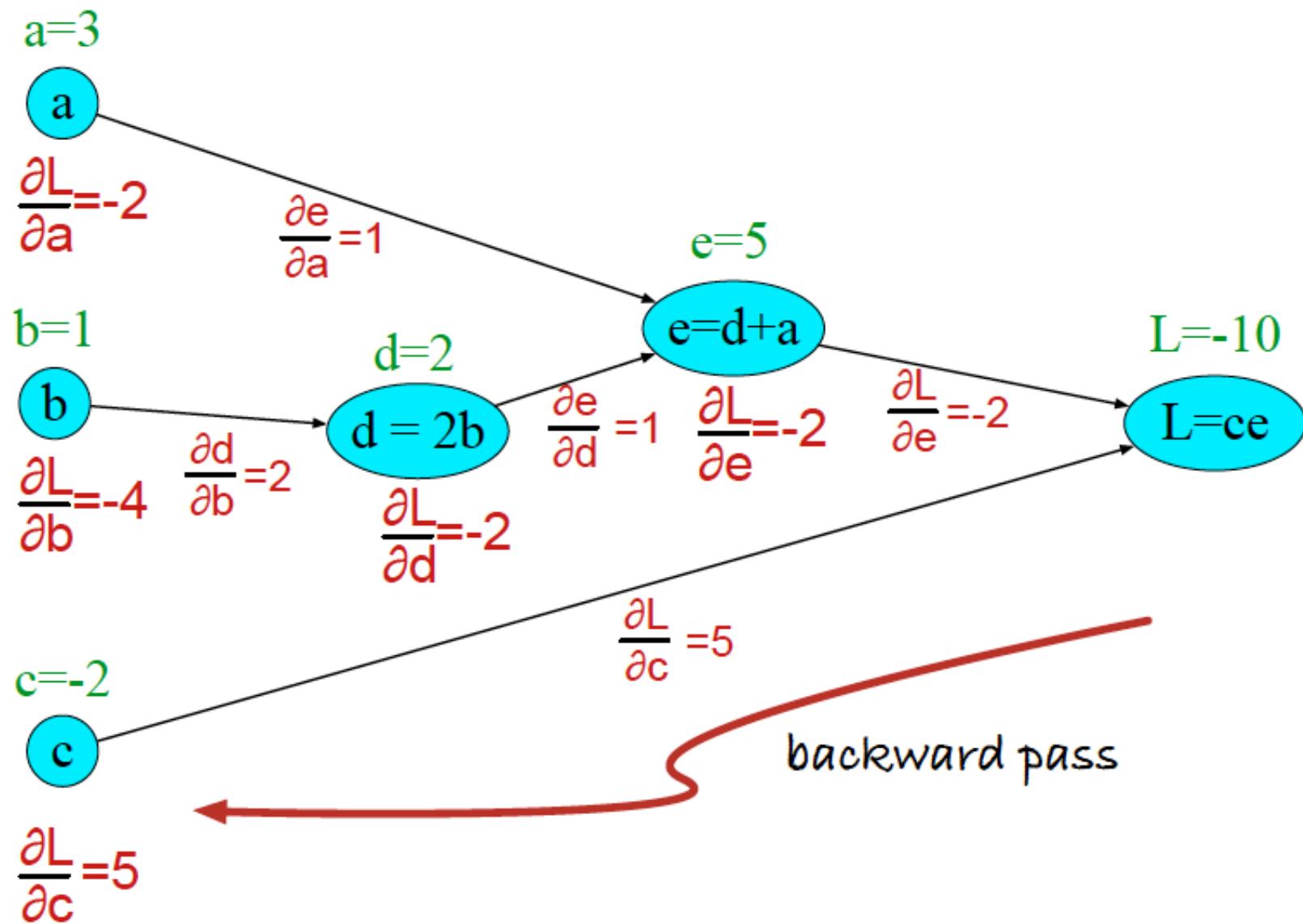
$$d=2$$

$$d = 2b$$

$$e=d+a$$

$$L=-10$$

$$L=ce$$



SUMMARY: PROCESS

- In short, backpropagation makes predictions for a mini-batch (forward pass),
- measures the error, then goes through each layer in reverse to measure the error contribution from each parameter (reverse pass),
- and finally tweaks the connection weights and biases to reduce the error (gradient descent step).

