

# Simple Neural Networks and Neural Language Models

## Units in Neural Networks

# Neural Network Unit

This is not in your brain

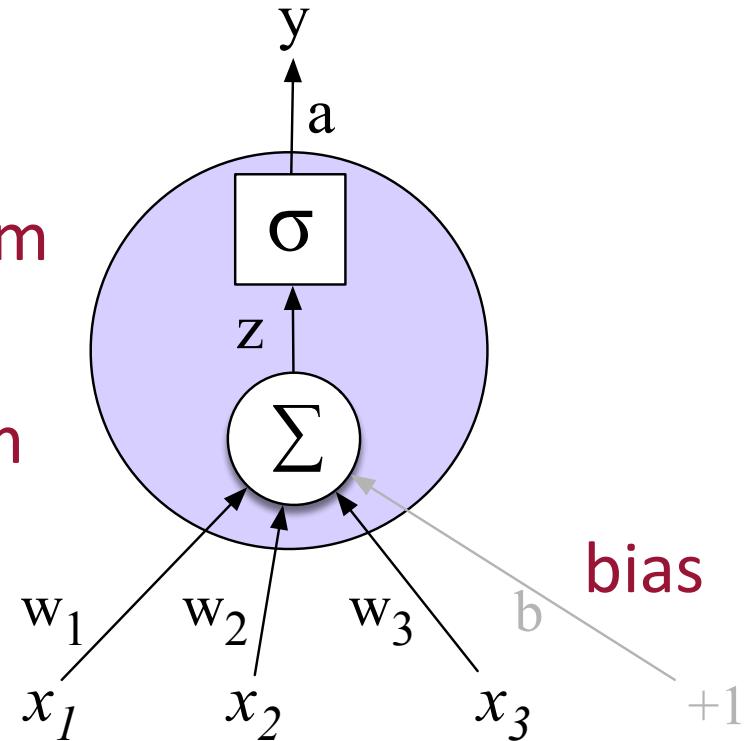
Output value

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  $f$ :

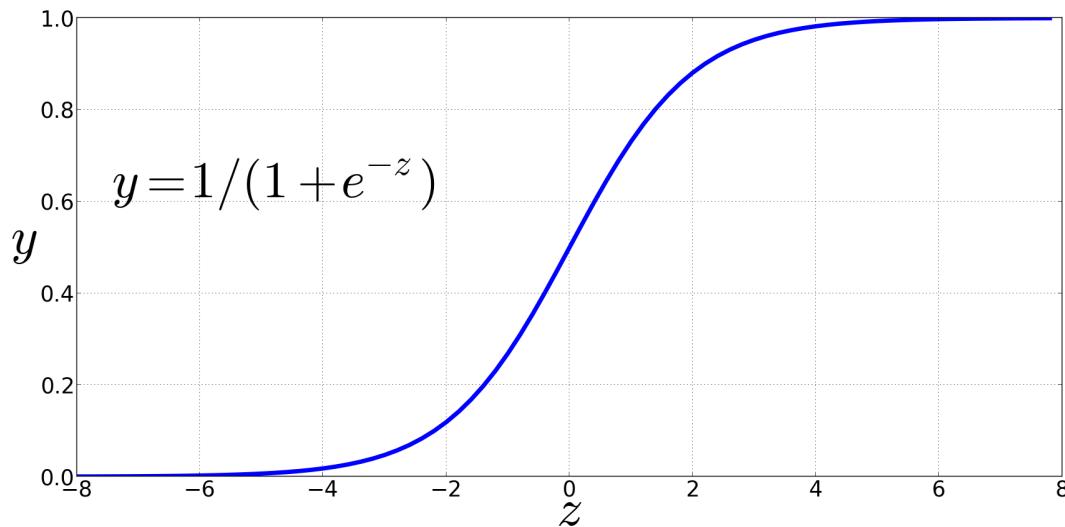
$$y = a = f(z)$$

# Non-Linear Activation Functions

We'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

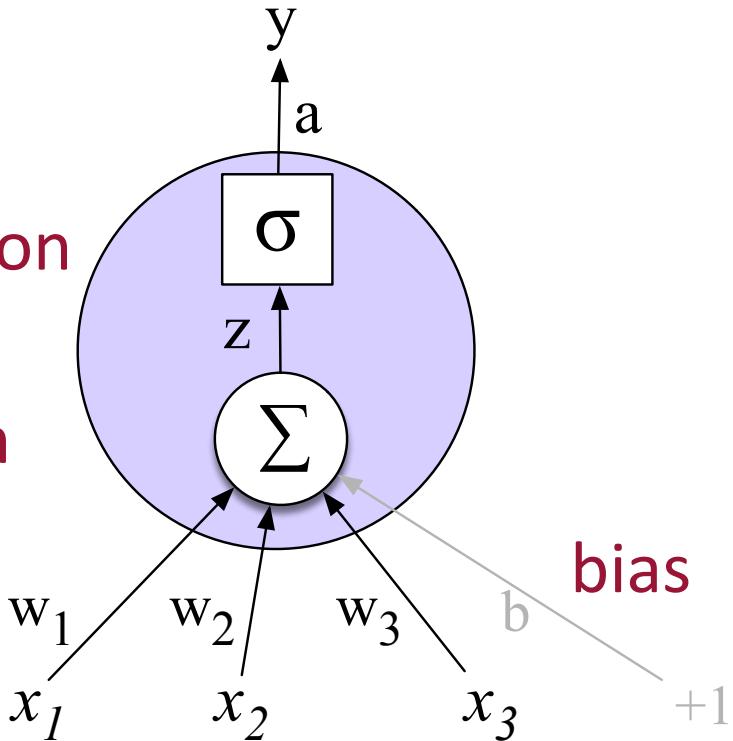
Output value

Non-linear activation function

Weighted sum

Weights

Input layer



# 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  $x$ ?

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

$$1$$

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

# An example

*Suppose a unit has:*

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

$$b = 0.5$$

What happens with input  $x$ :

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

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

# An example

*Suppose a unit has:*

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

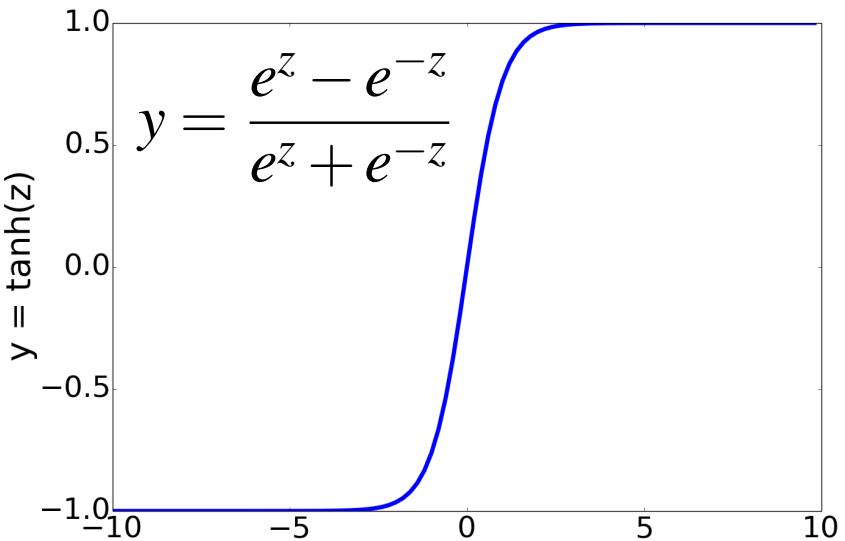
$$b = 0.5$$

What happens with input  $x$ :

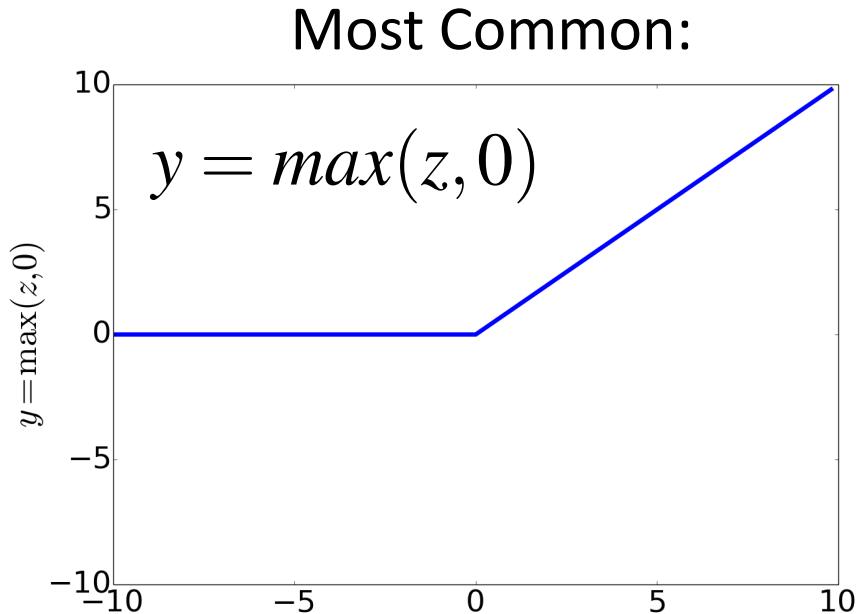
$$x = [0.5, 0.6, 0.1]$$

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

# Non-Linear Activation Functions besides sigmoid



tanh



ReLU  
Rectified Linear Unit

# Simple Neural Networks and Neural Language Models

## Units in Neural Networks

# Simple Neural Networks and Neural Language Models

## The XOR problem

# The XOR problem

Minsky and Papert (1969)

Can neural units compute simple functions of input?

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

# Perceptrons

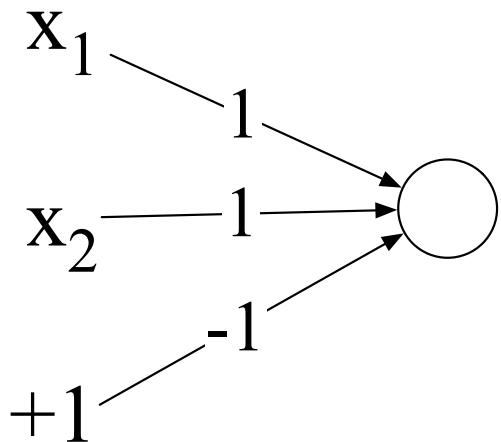
A very simple neural unit

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

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

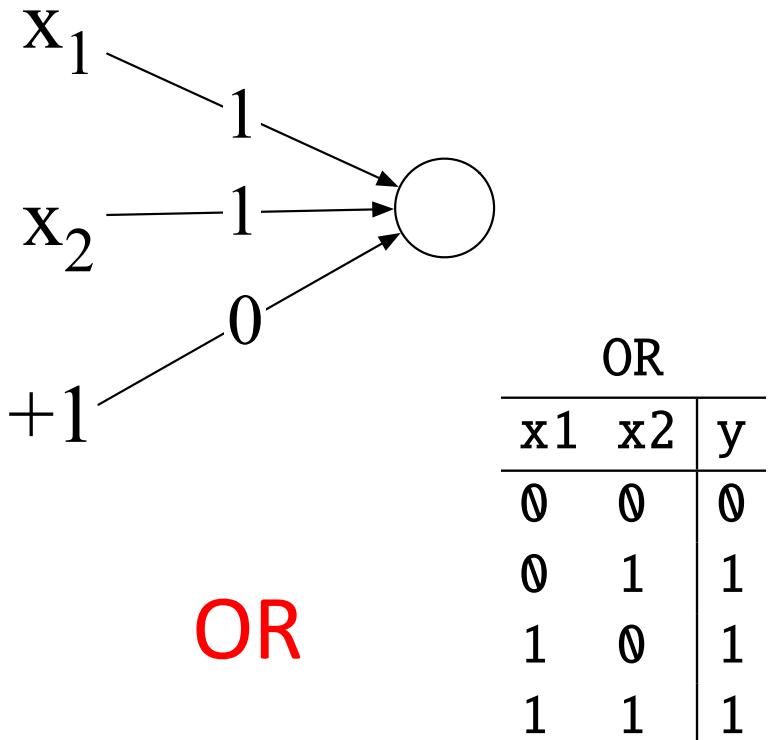
# Easy to build AND or OR with perceptrons

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



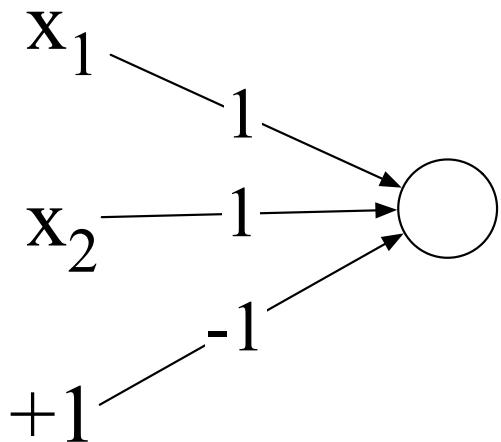
AND

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



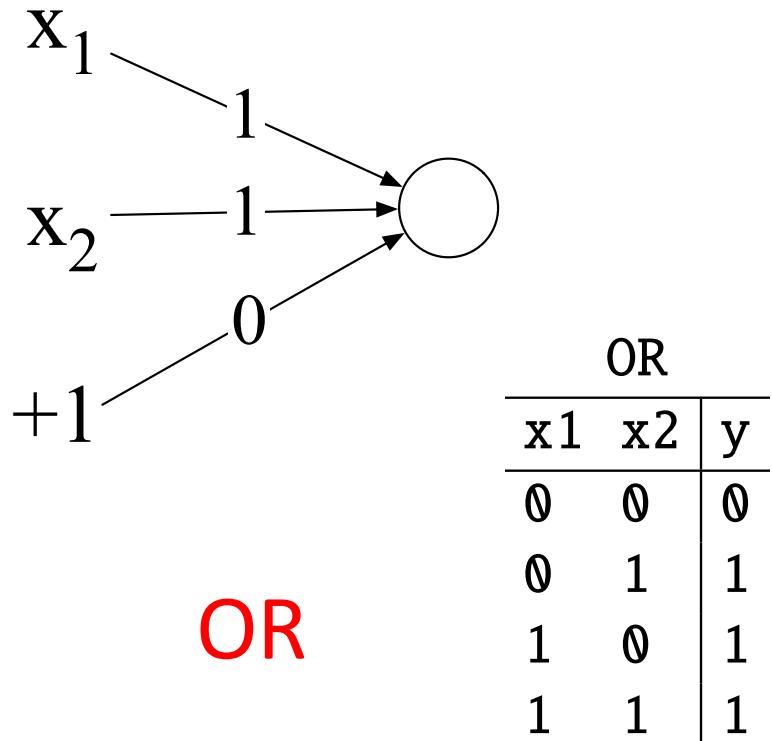
# Easy to build AND or OR with perceptrons

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



AND

		AND
$x_1$	$x_2$	$y$
0	0	0
0	1	0
1	0	0
1	1	1

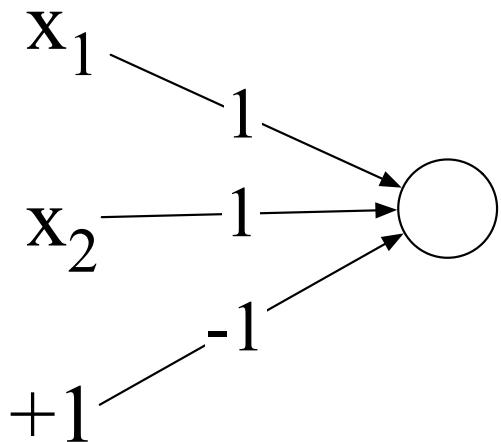


OR

		OR
$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	1

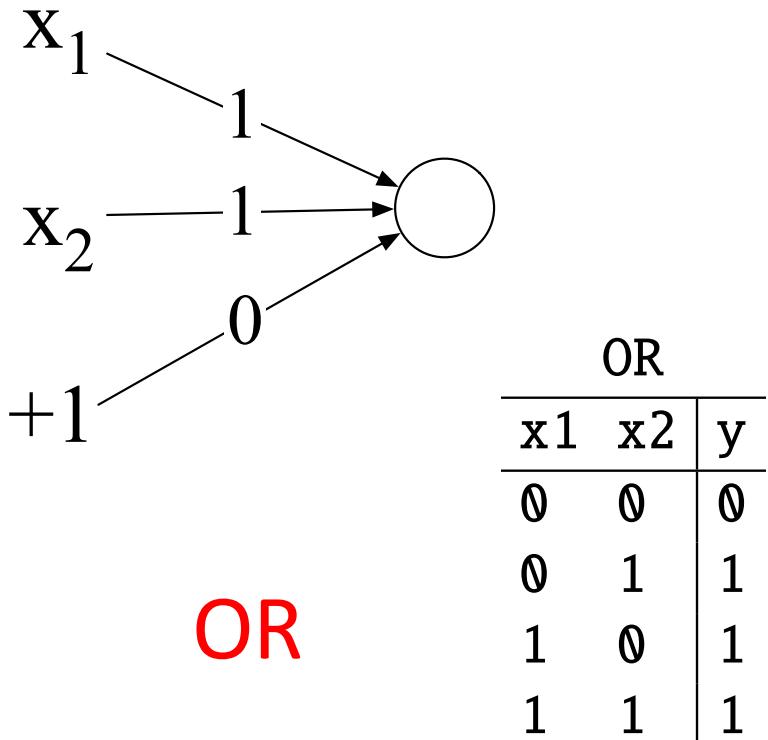
# Easy to build AND or OR with perceptrons

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



AND

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



Not possible to capture XOR with perceptrons

Pause the lecture and try for yourself!

# Why? Perceptrons are linear classifiers

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

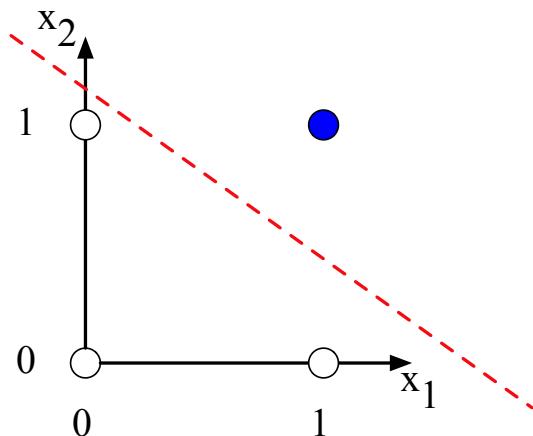
$$w_1x_1 + w_2x_2 + b = 0$$

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

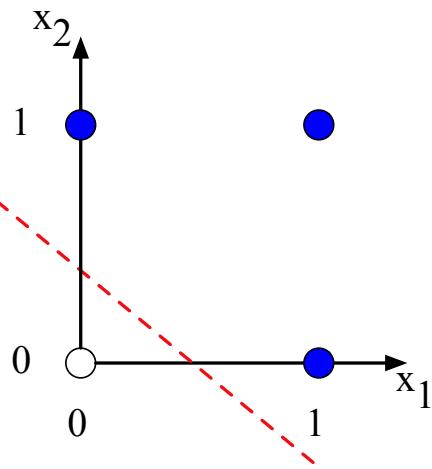
This line acts as a **decision boundary**

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

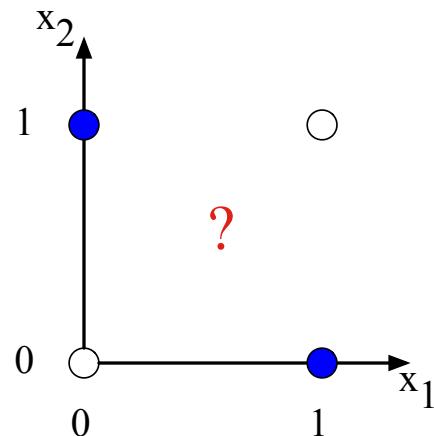
# Decision boundaries



a)  $x_1$  AND  $x_2$



b)  $x_1$  OR  $x_2$



c)  $x_1$  XOR  $x_2$

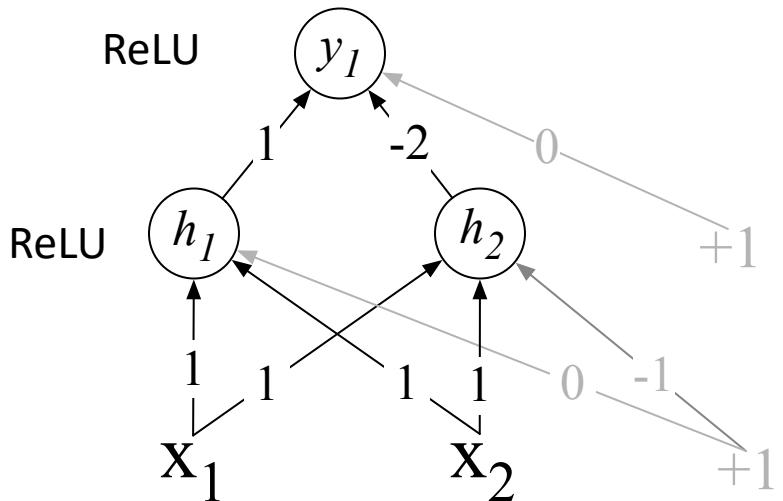
XOR is not a **linearly separable** function!

# Solution to the XOR problem

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

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

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

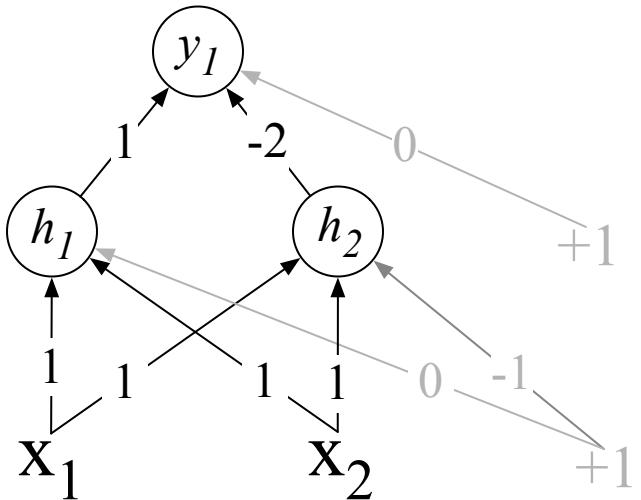


# Solution to the XOR problem

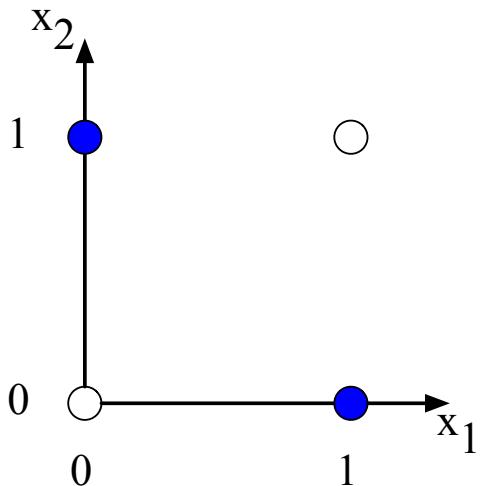
XOR **can't** be calculated by a single perceptron

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

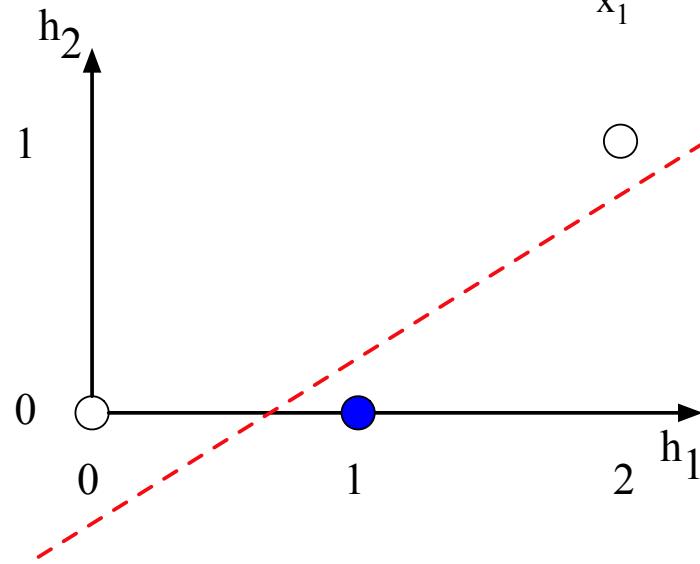
XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0



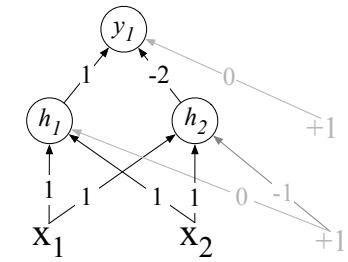
# The hidden representation $h$



a) The original  $x$  space



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



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

# Simple Neural Networks and Neural Language Models

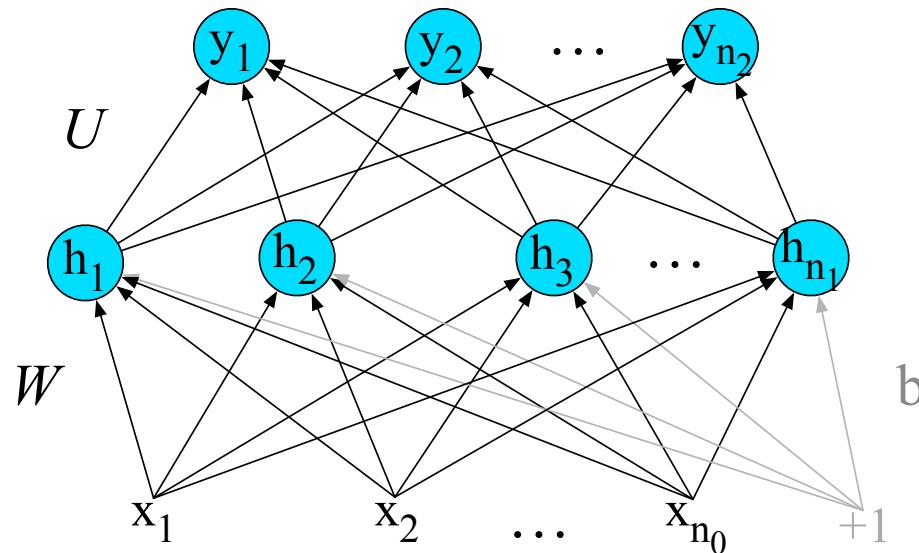
## The XOR problem

# Simple Neural Networks and Neural Language Models

## Feedforward Neural Networks

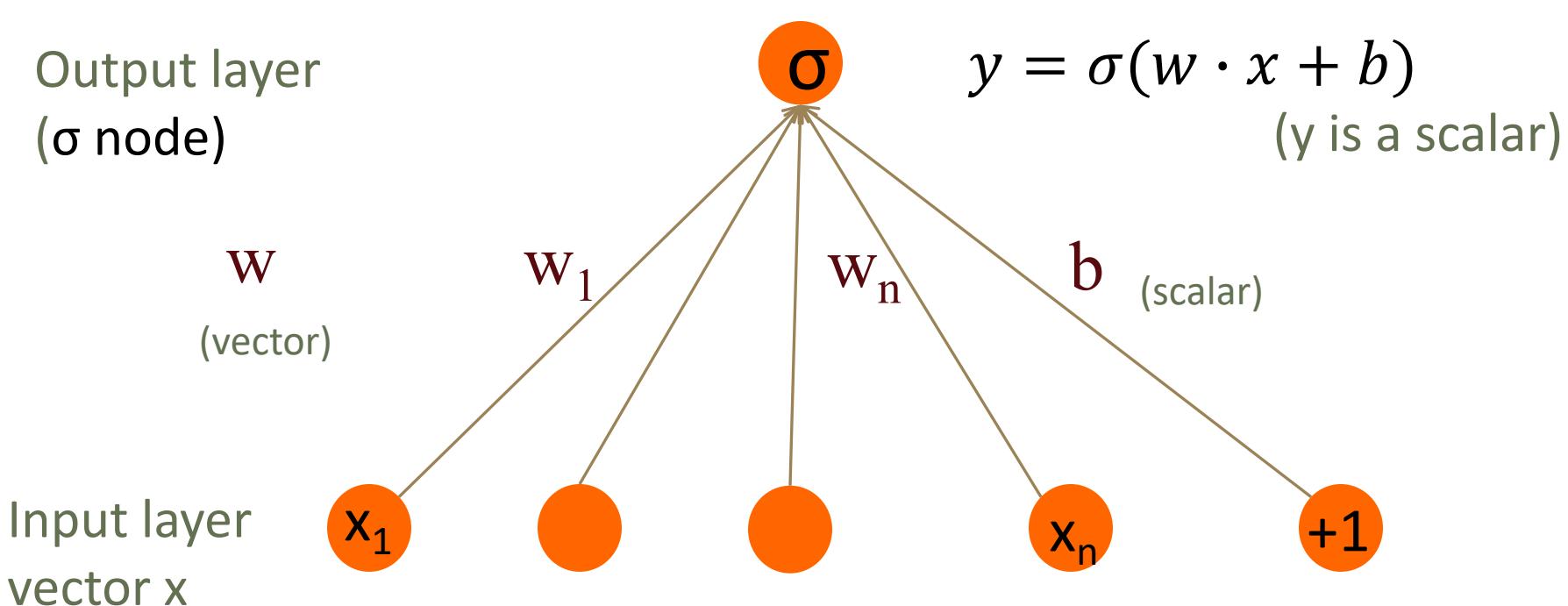
# Feedforward Neural Networks

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



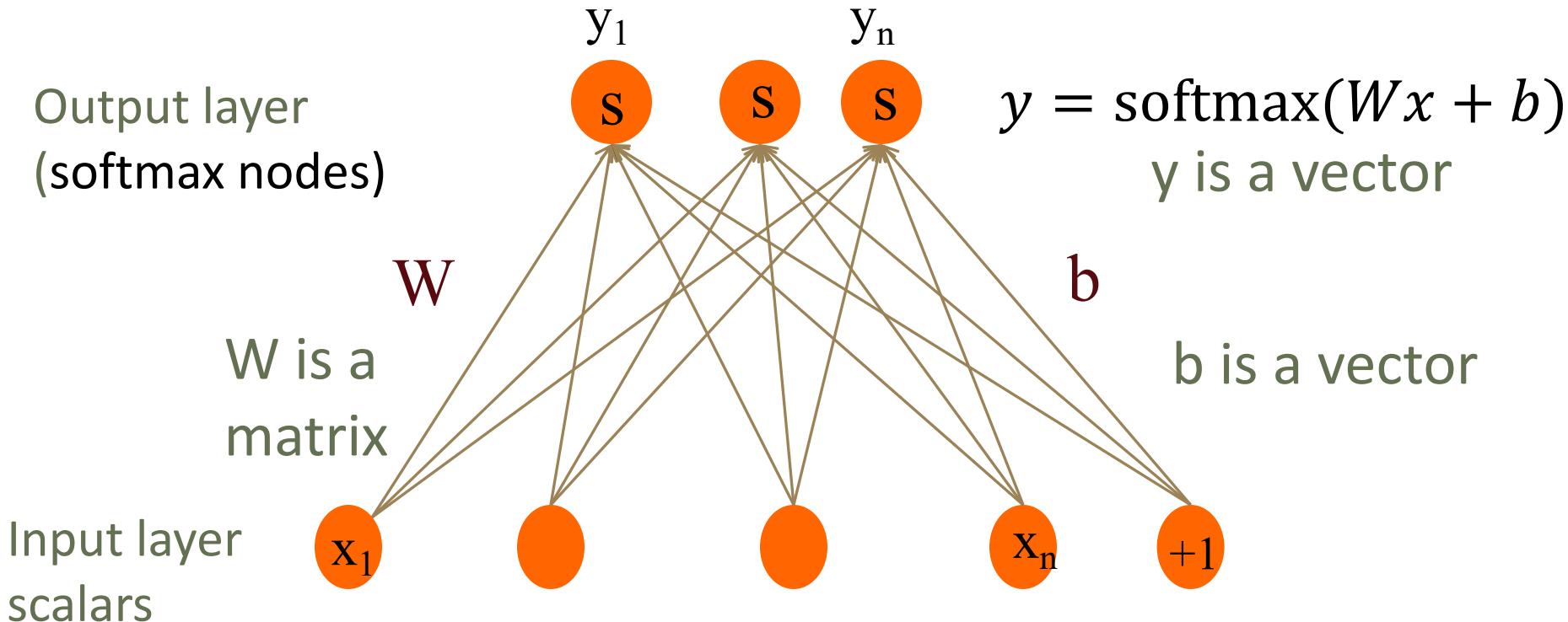
# Binary Logistic Regression as a 1-layer Network

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



# Multinomial Logistic Regression as a 1-layer Network

Fully connected single layer network



Reminder: softmax: a generalization of sigmoid

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

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

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

Example:

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

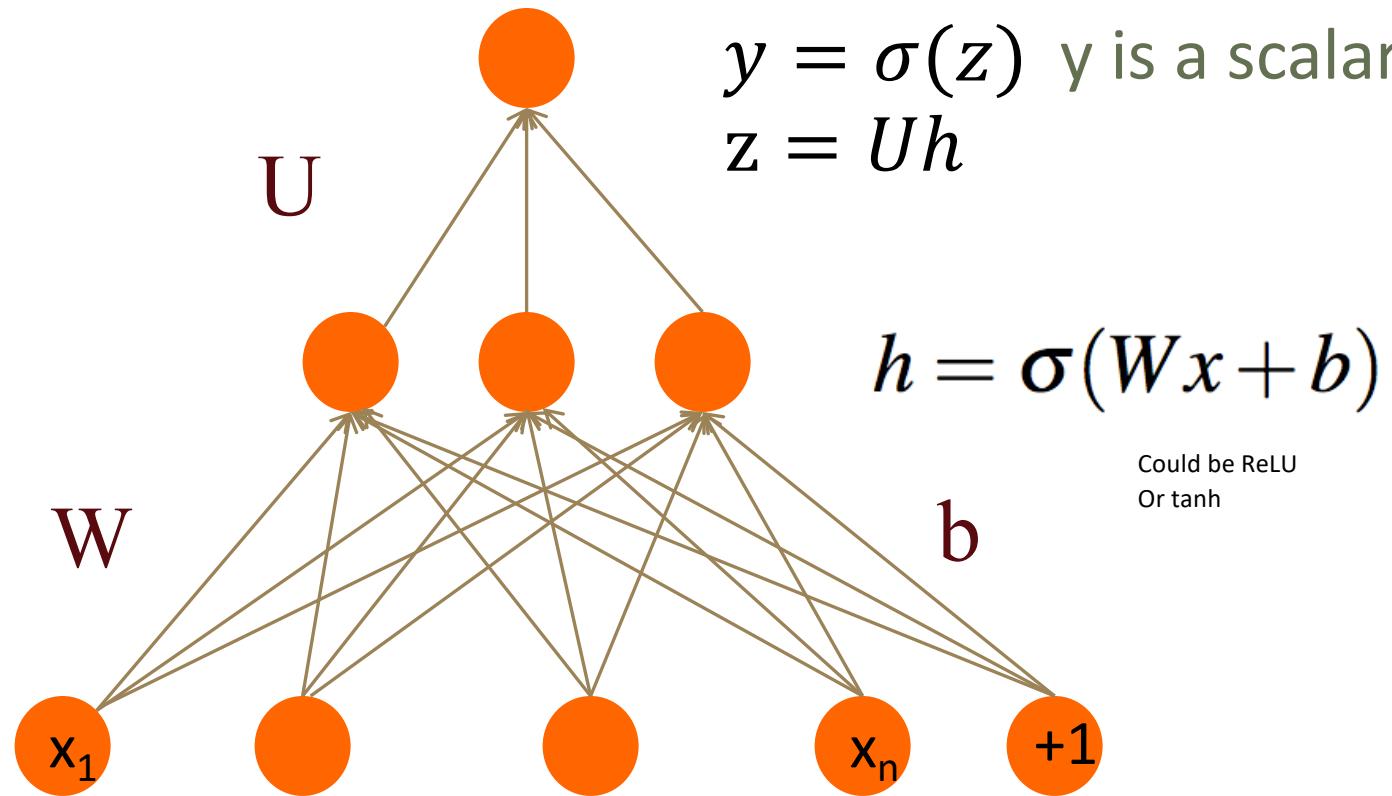
$$\text{softmax}(z) = [0.055, 0.090, 0.006, 0.099, 0.74, 0.010]$$

# Two-Layer Network with scalar output

Output layer  
( $\sigma$  node)

hidden units  
( $\sigma$  node)

Input layer  
(vector)

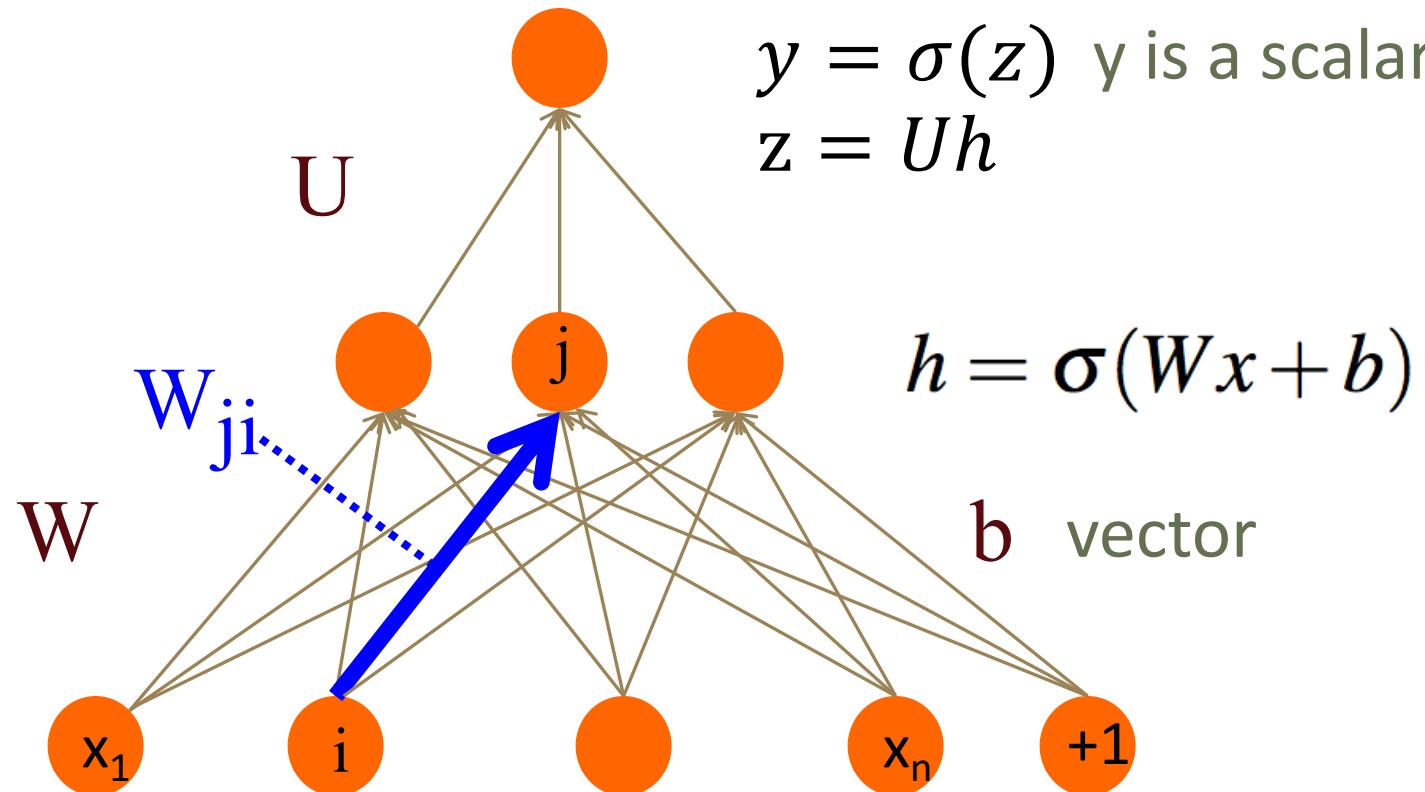


# Two-Layer Network with scalar output

Output layer  
( $\sigma$  node)

hidden units  
( $\sigma$  node)

Input layer  
(vector)

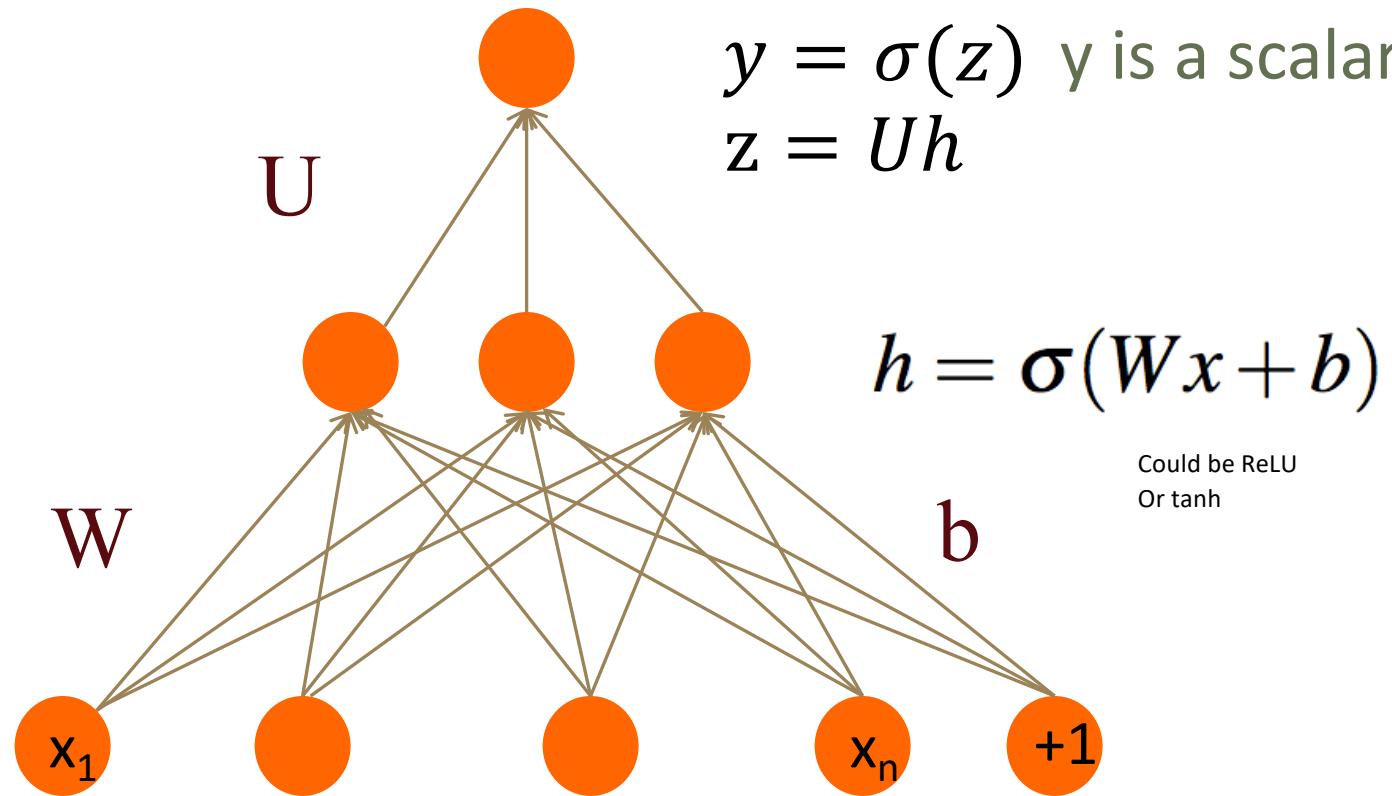


# Two-Layer Network with scalar output

Output layer  
( $\sigma$  node)

hidden units  
( $\sigma$  node)

Input layer  
(vector)

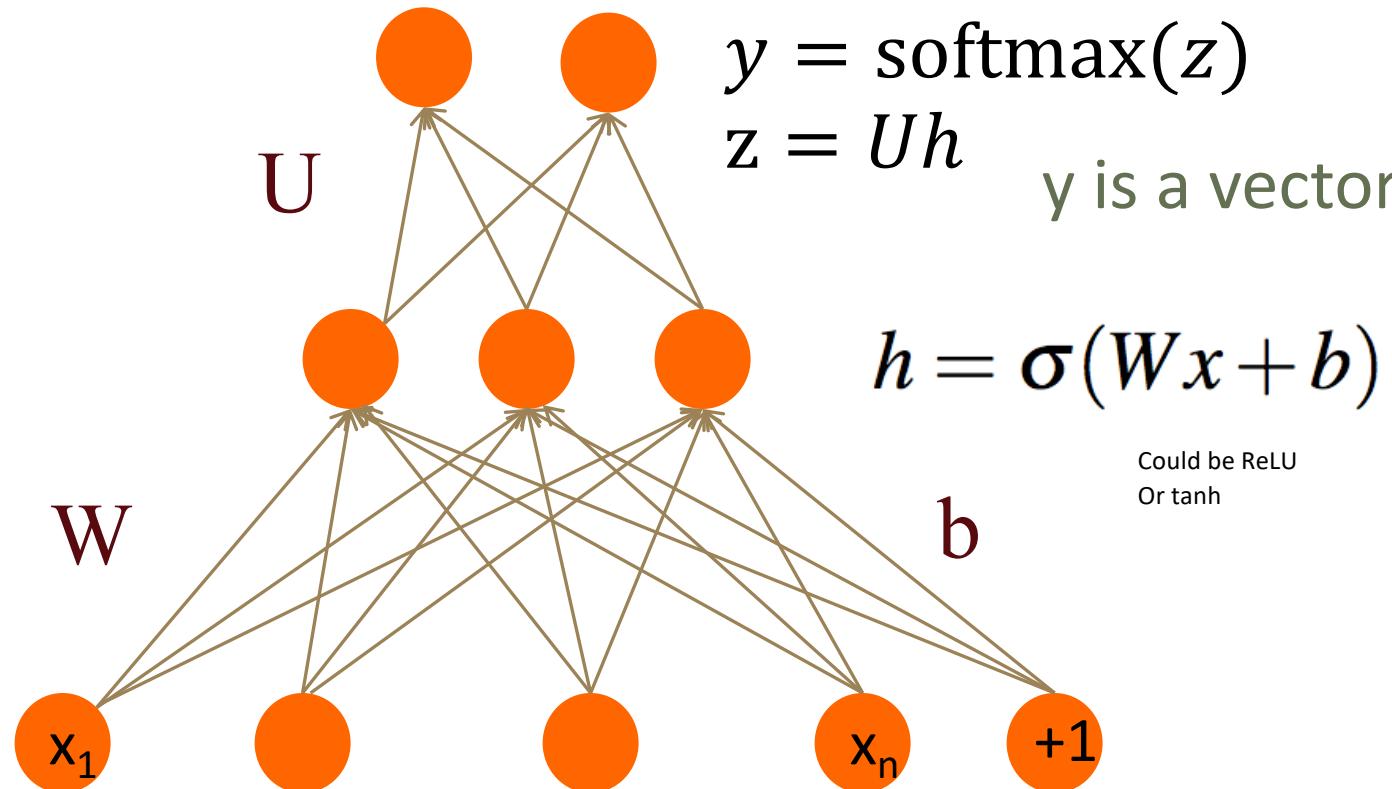


# Two-Layer Network with softmax output

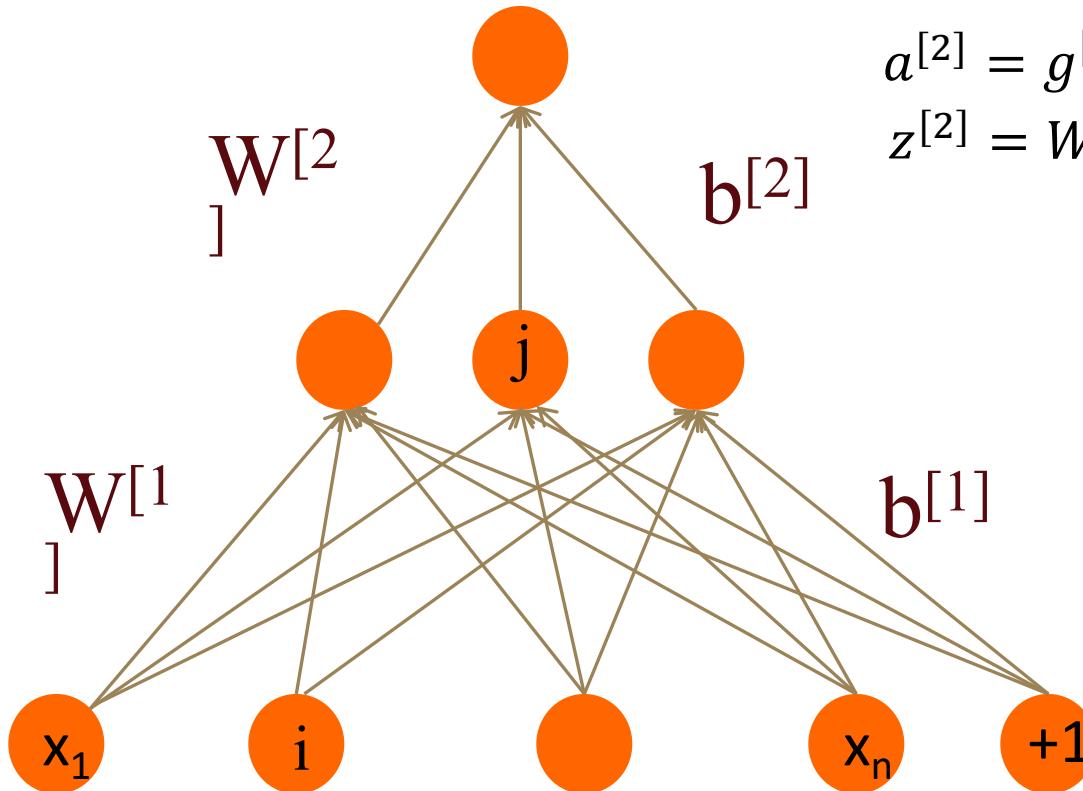
Output layer  
( $\sigma$  node)

hidden units  
( $\sigma$  node)

Input layer  
(vector)



# Multi-layer Notation



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

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

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

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

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

$$a^{[0]}$$

# Multi Layer Notation

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

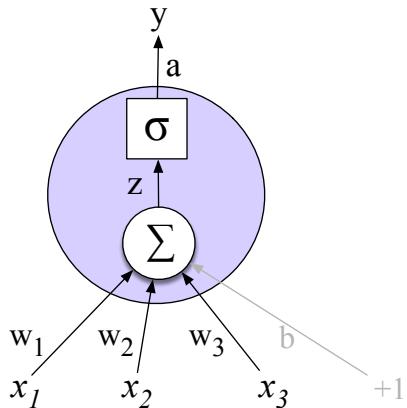
$$a^{[1]} = g^{[1]}(z^{[1]})$$

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

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

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

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

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


# Replacing the bias unit

Let's switch to a notation without the bias unit

Just a notational change

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

# Replacing the bias unit

Instead of:

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

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

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

We'll do this:

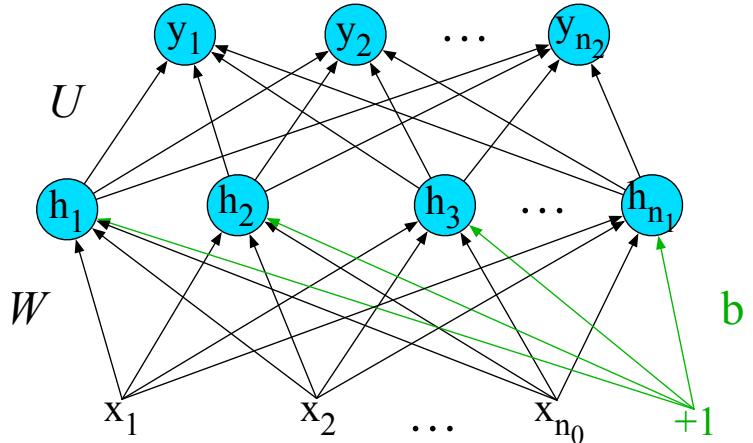
$$x = x_0, x_1, x_2, \dots, x_{n0}$$

$$h = \sigma(Wx)$$

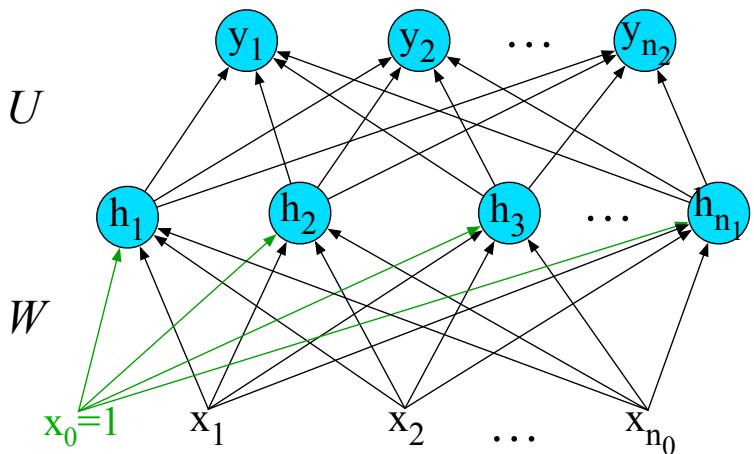
$$\sigma \left( \sum_{i=0}^{n_0} W_{ji} x_i \right)$$

# Replacing the bias unit

Instead of:



We'll do this:



# Simple Neural Networks and Neural Language Models

## Feedforward Neural Networks

# Simple Neural Networks and Neural Language Models

## Applying feedforward networks to NLP tasks

# Use cases for feedforward networks

Let's consider 2 (simplified) sample tasks:

1. Text classification
2. Language modeling

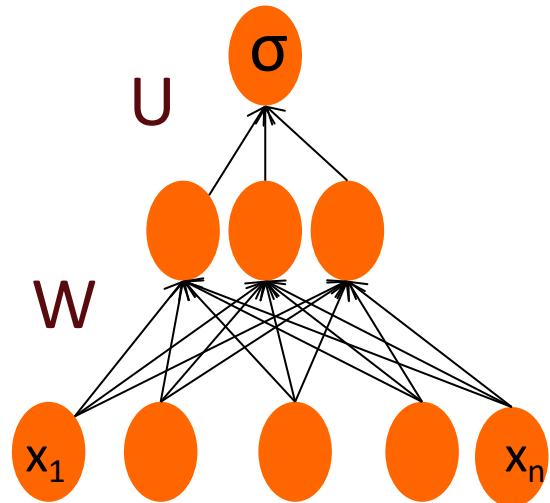
State of the art systems use more powerful neural architectures, but simple models are useful to consider!

# Classification: Sentiment Analysis

We could do exactly what we did with logistic regression

Input layer are binary features as before

Output layer is 0 or 1

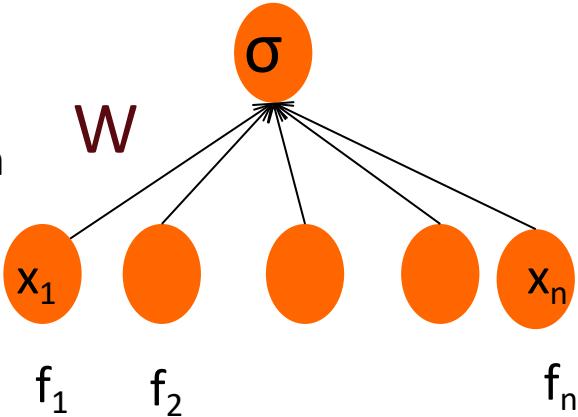


# Sentiment Features

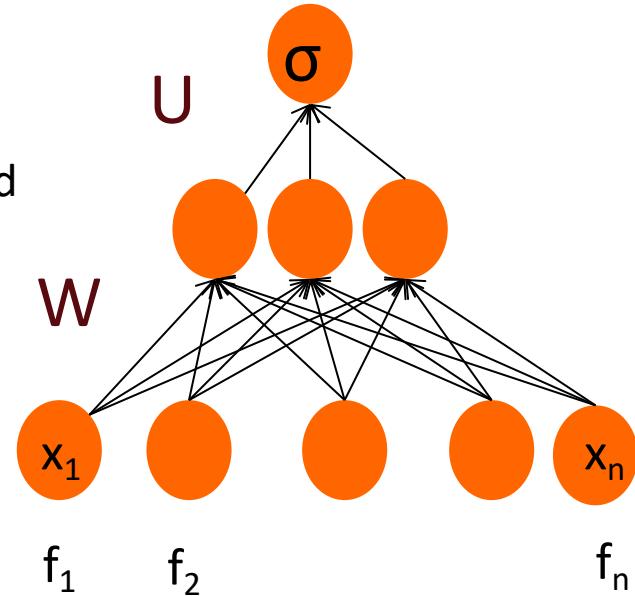
Var	Definition
$x_1$	count(positive lexicon) $\in$ doc)
$x_2$	count(negative lexicon) $\in$ doc)
$x_3$	$\begin{cases} 1 & \text{if “no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
$x_4$	count(1st and 2nd pronouns $\in$ doc)
$x_5$	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise}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

# Feedforward nets for simple classification

Logistic  
Regression



2-layer  
feedforward  
network



Just adding a hidden layer to logistic regression

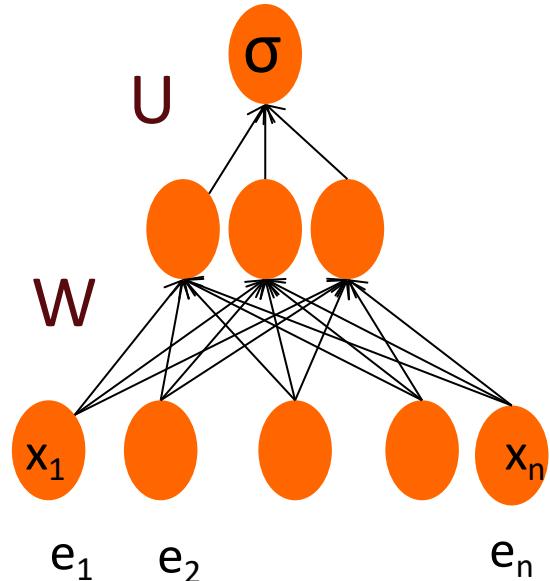
- allows the network to use non-linear interactions between features
- which may (or may not) improve performance.

# Even better: representation learning

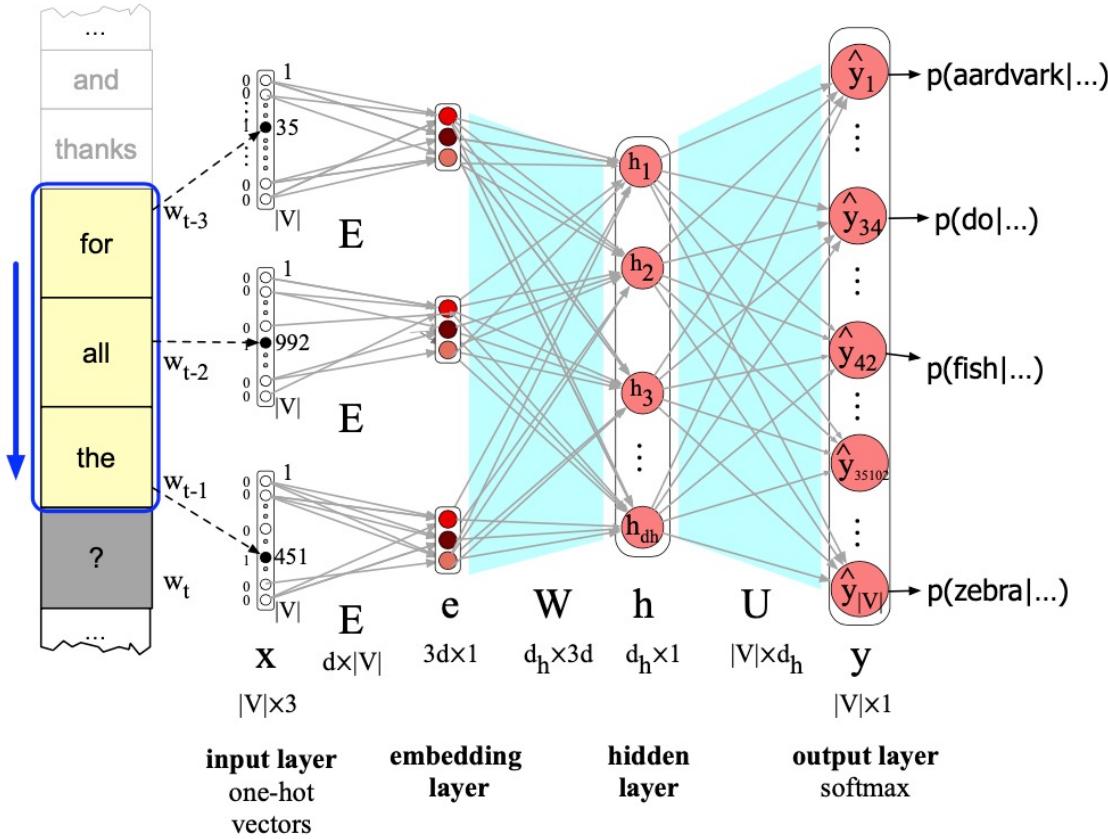
The real power of deep learning comes from the ability to **learn** features from the data

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

Use learned representations like embeddings!



# Neural Net Classification with embeddings as input features!



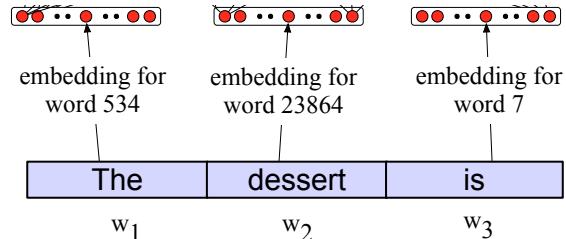
# Issue: texts come in different sizes

This assumes a fixed size length (3)!

Kind of unrealistic.

Some simple solutions (more sophisticated solutions later)

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

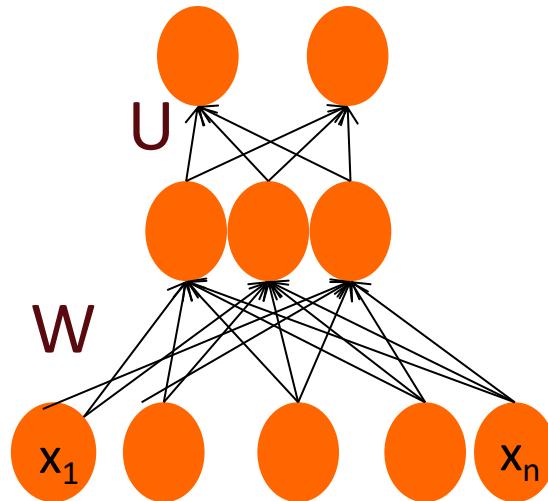


# Reminder: Multiclass Outputs

What if you have more than two output classes?

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

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



# Neural Language Models (LMs)

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

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

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

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

# Simple feedforward Neural Language Models

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

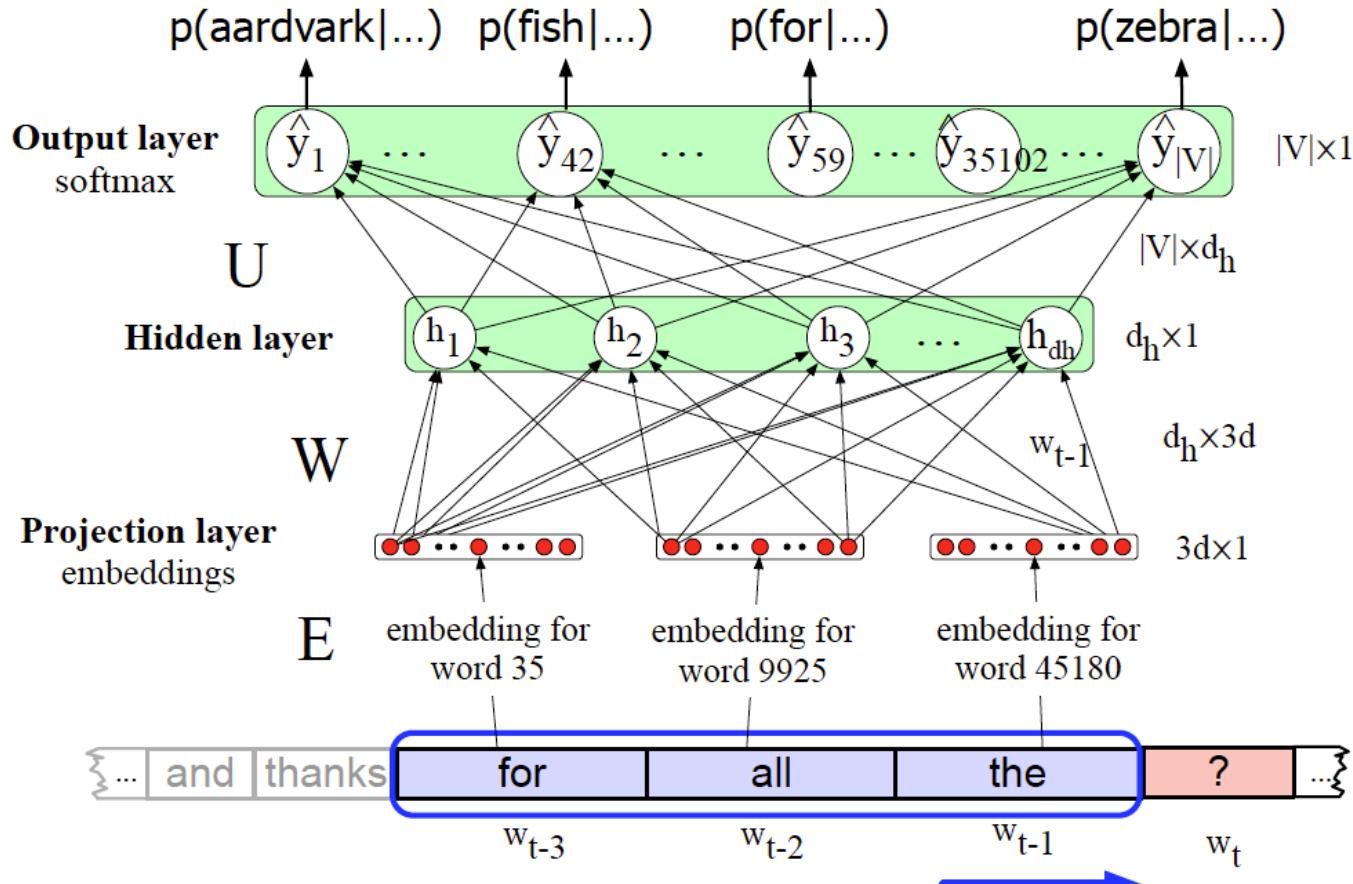
given prior words  $w_{t-1}, w_{t-2}, w_{t-3}, \dots$

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

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

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

# Neural Language Model



# Why Neural LMs work better than N-gram LMs

## Training data:

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

Never seen: dog gets fed

## Test data:

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

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

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

# Simple Neural Networks and Neural Language Models

## Applying feedforward networks to NLP tasks