Simple Neural Networks and Neural Language Models

# Units in Neural Networks

# Neural Network Unit

This is not in your brain

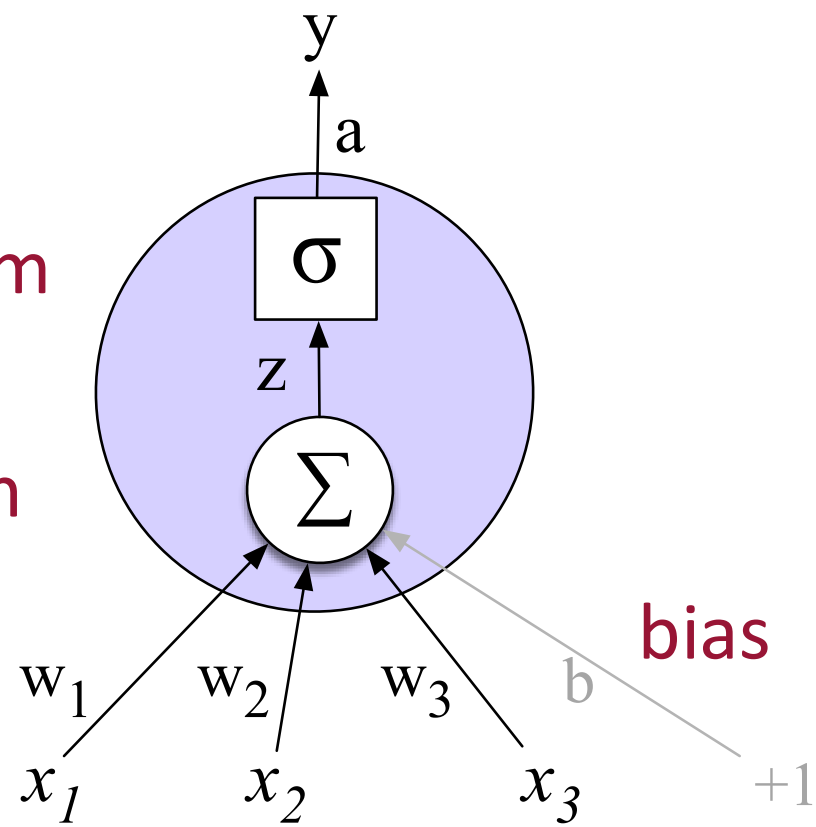
Output value

Non-linear transform

Weighted sum

Weights

Input layer




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:

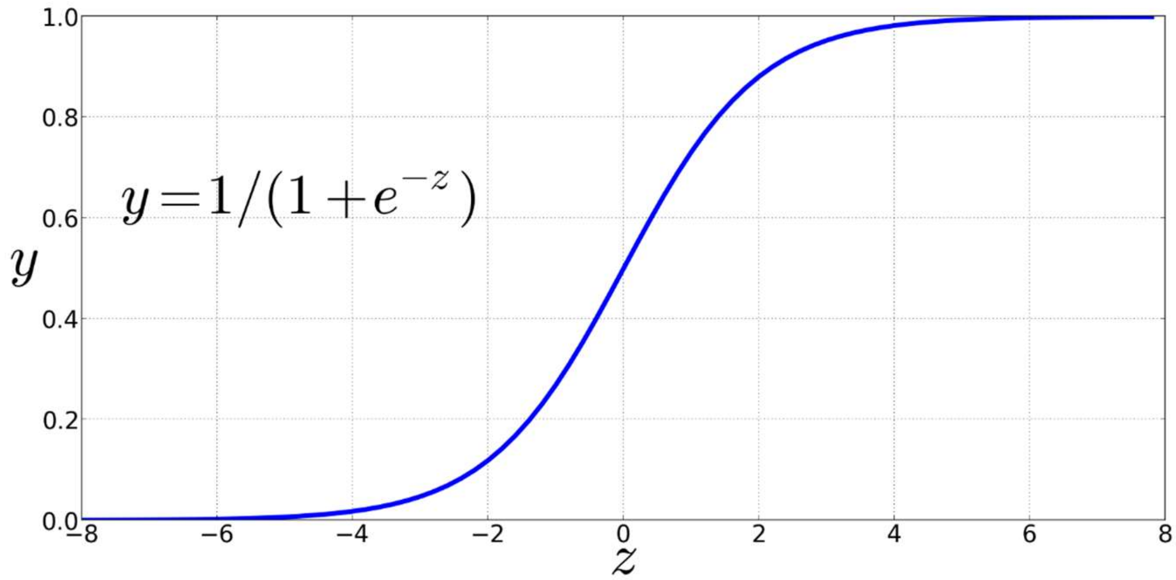
$$y = a = f(z)$$

# Non-Linear Activation Functions

We're already seen the sigmoid for logistic regression:

## Sigmoid

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



# Final function the unit is computing

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

# Final unit again

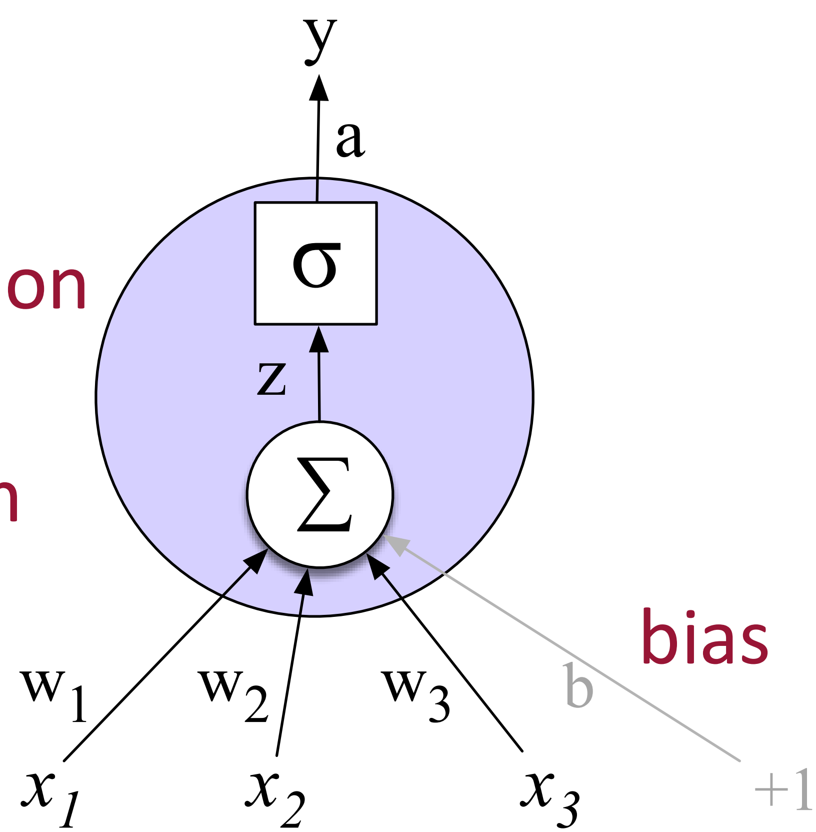
Output value

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]`

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

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

# An example

*Suppose a unit has:*

$w = [0.2, 0.3, 0.9]$

$b = 0.5$

What happens with input x:

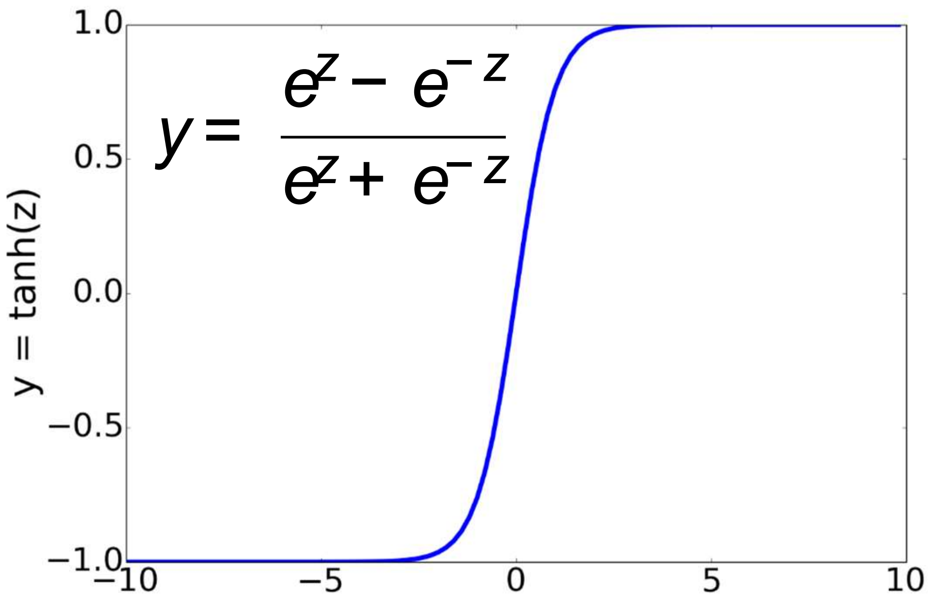
$x = [0.5, 0.6, 0.1]$

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

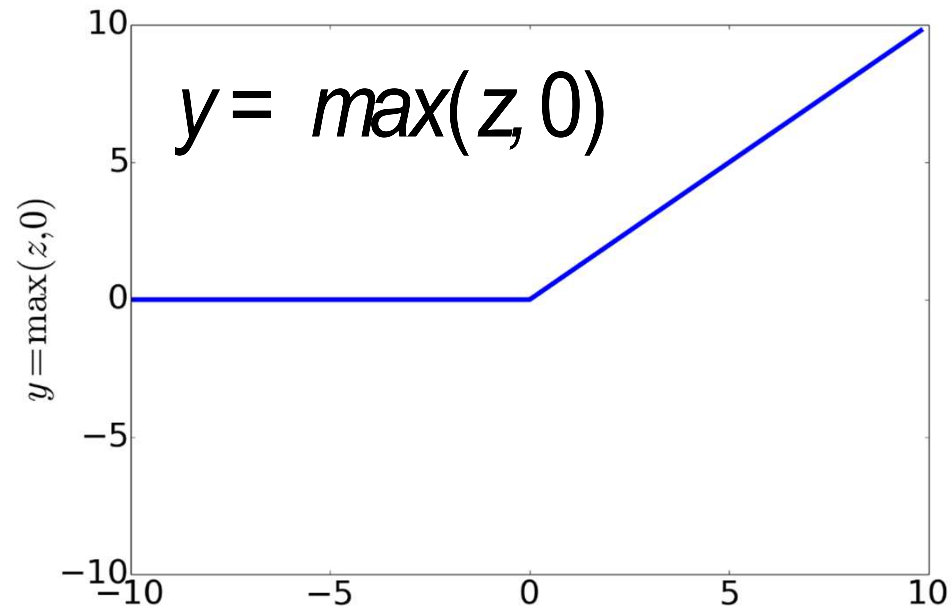
$$\frac{1}{1 + e^{-(.5 \cdot .2 + .6 \cdot .3 + .1 \cdot .9 + .5)}} = \frac{1}{1 + e^{-0.87}} = .70$$

# Non-Linear Activation Functions besides sigmoid

Most Common:




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

| 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

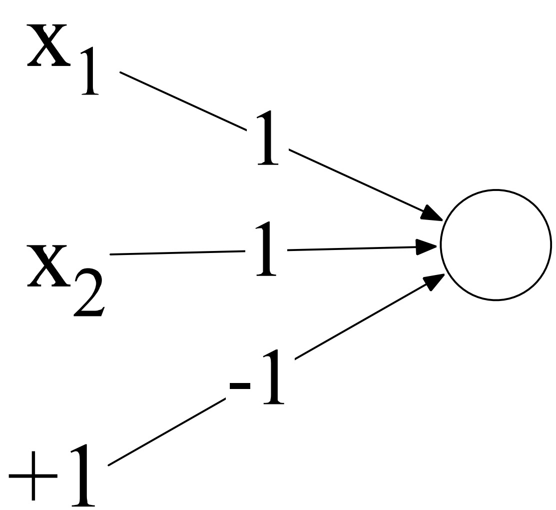
A very simple neural unit

- 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

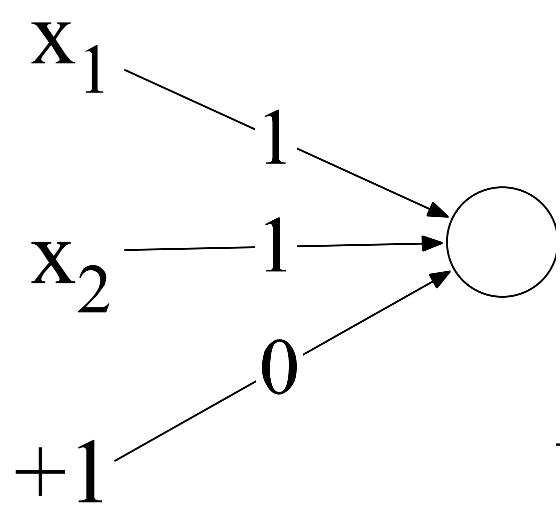
$$y = \begin{cases} 0, & \text{if } w \cdot x + b \le 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases}$$




AND

| x1 | x2 | y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

AND



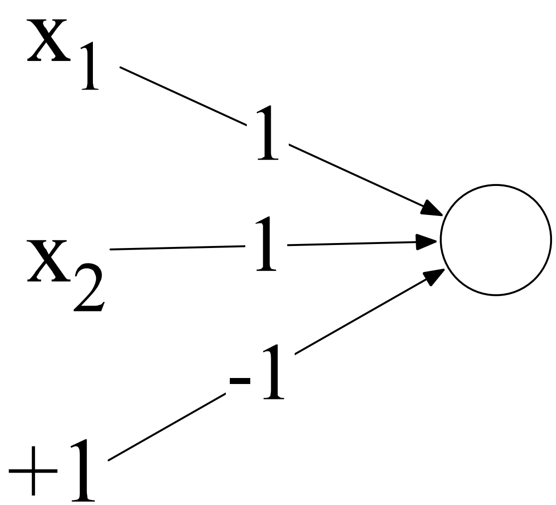

OR

| x1 | x2 | y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

OR

# Easy to build AND or OR with perceptrons

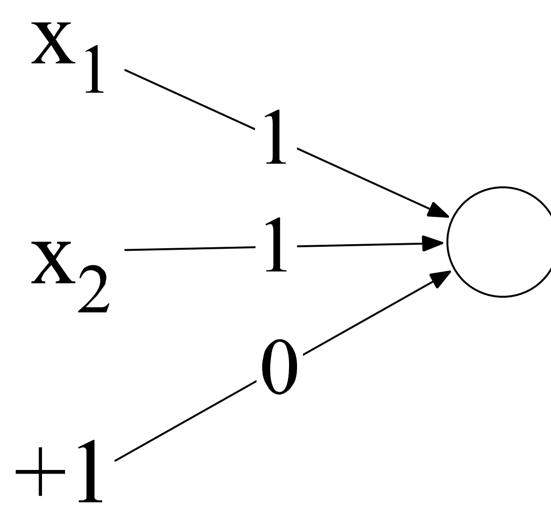
$$y = \begin{cases} 0, & \text{if } w \cdot x + b \leq 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases}$$




AND

| x1 | x2 | y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

AND



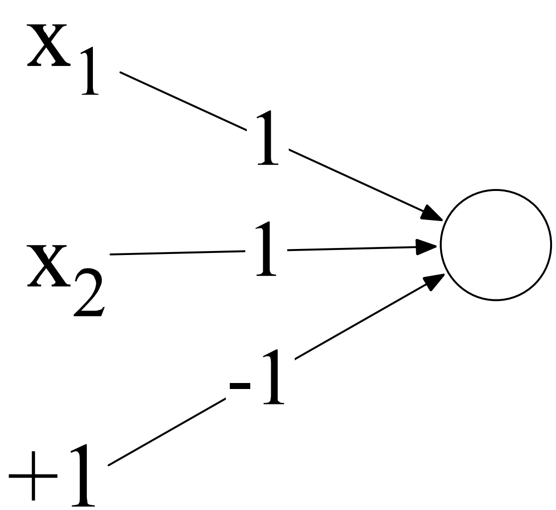

OR

| x1 | x2 | y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

OR

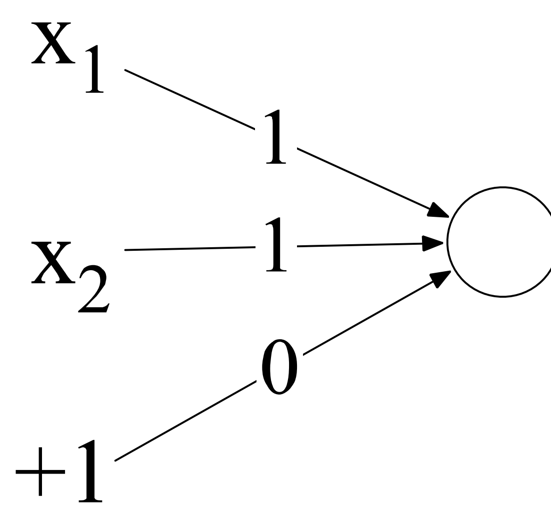
# Easy to build AND or OR with perceptrons

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




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 |

# 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

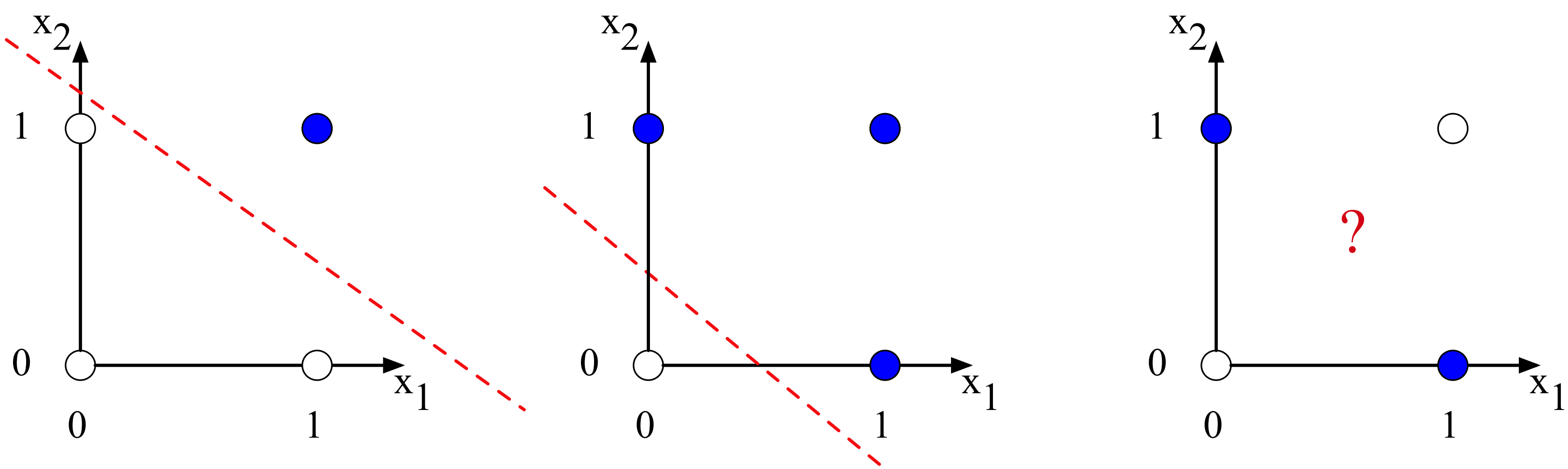
$$w_1x_1 + w_2x_2 + b = 0$$

(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!

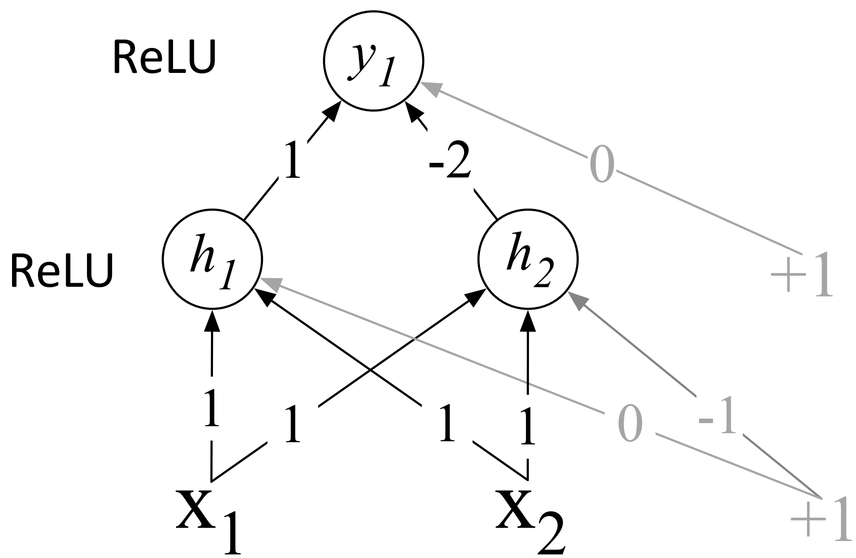
# Solution to the XOR problem

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 |



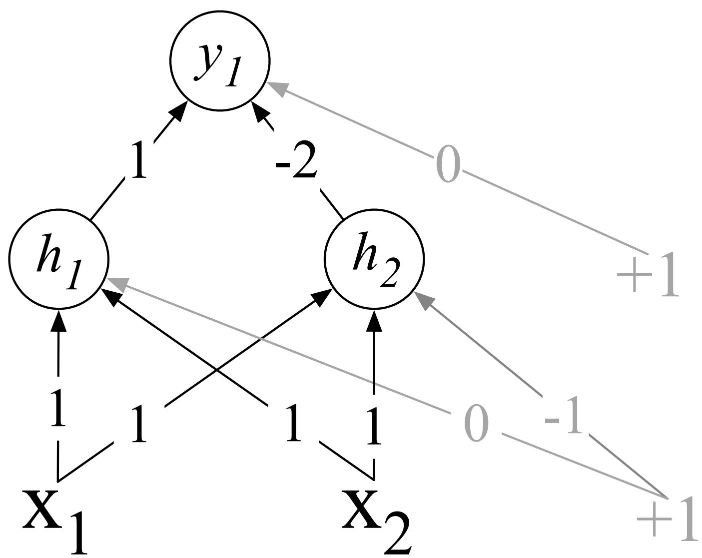
# Solution to the XOR problem

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

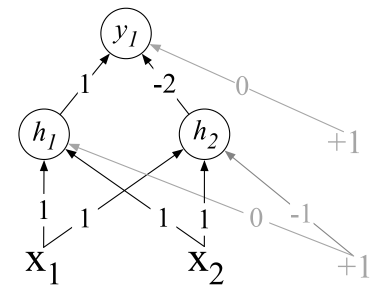
XOR **can** be calculated by a layered network of units.

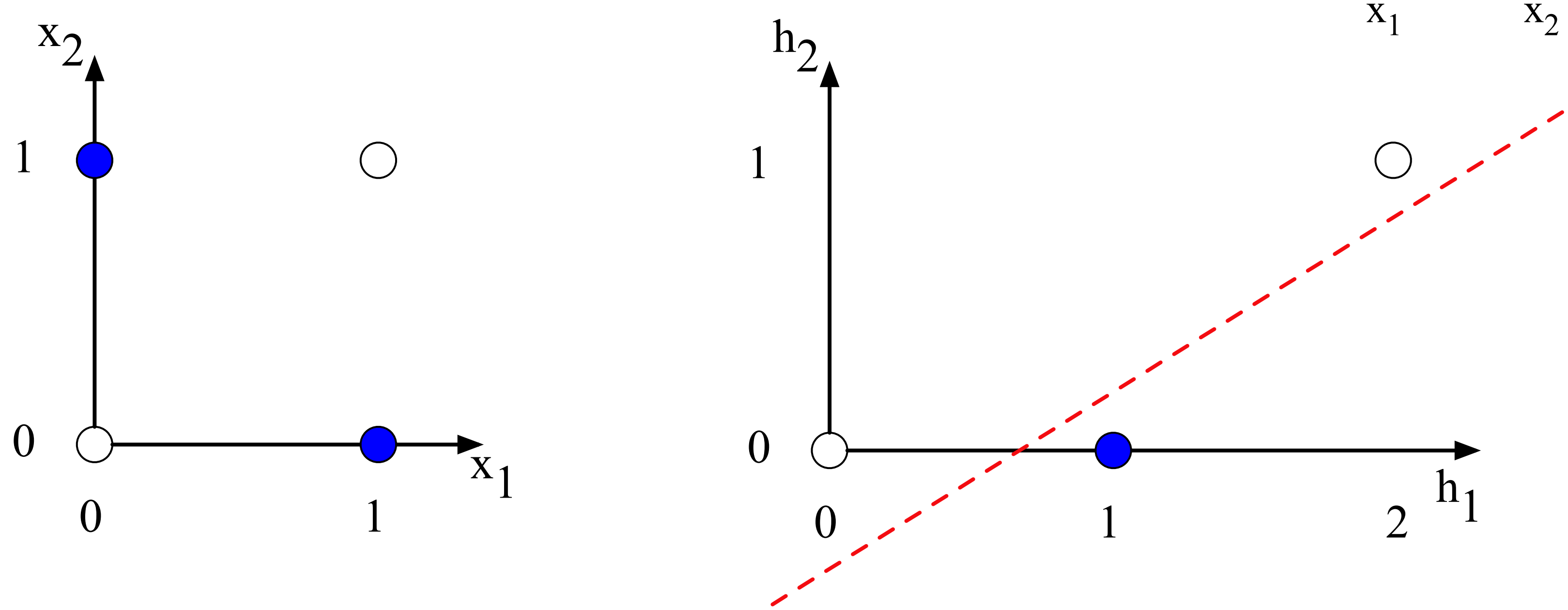
XOR

| 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

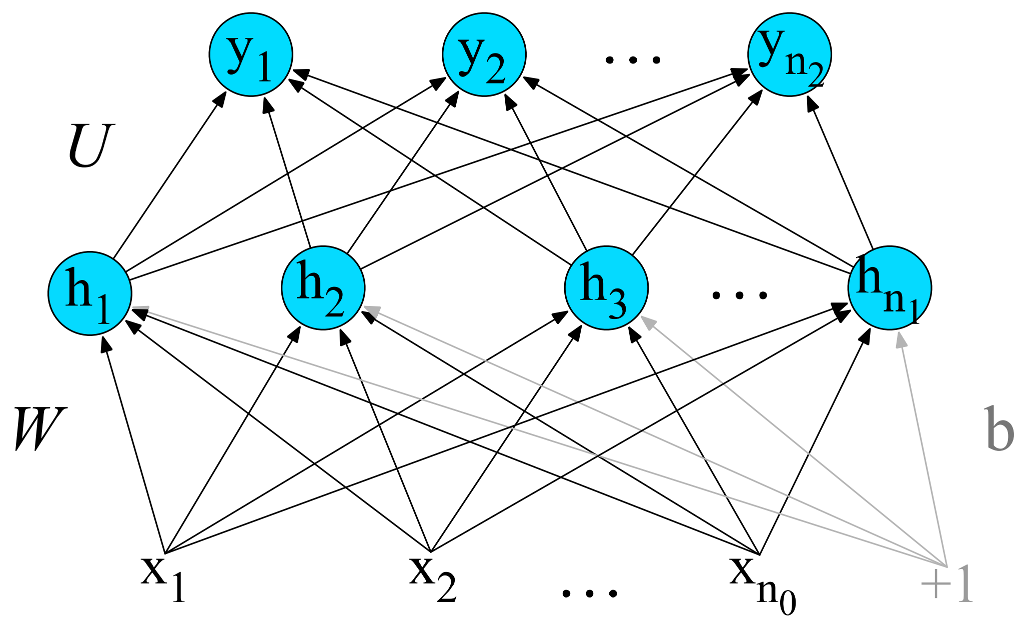
# The XOR problem

Simple Neural Networks and Neural Language Models

# Feedforward Neural Networks

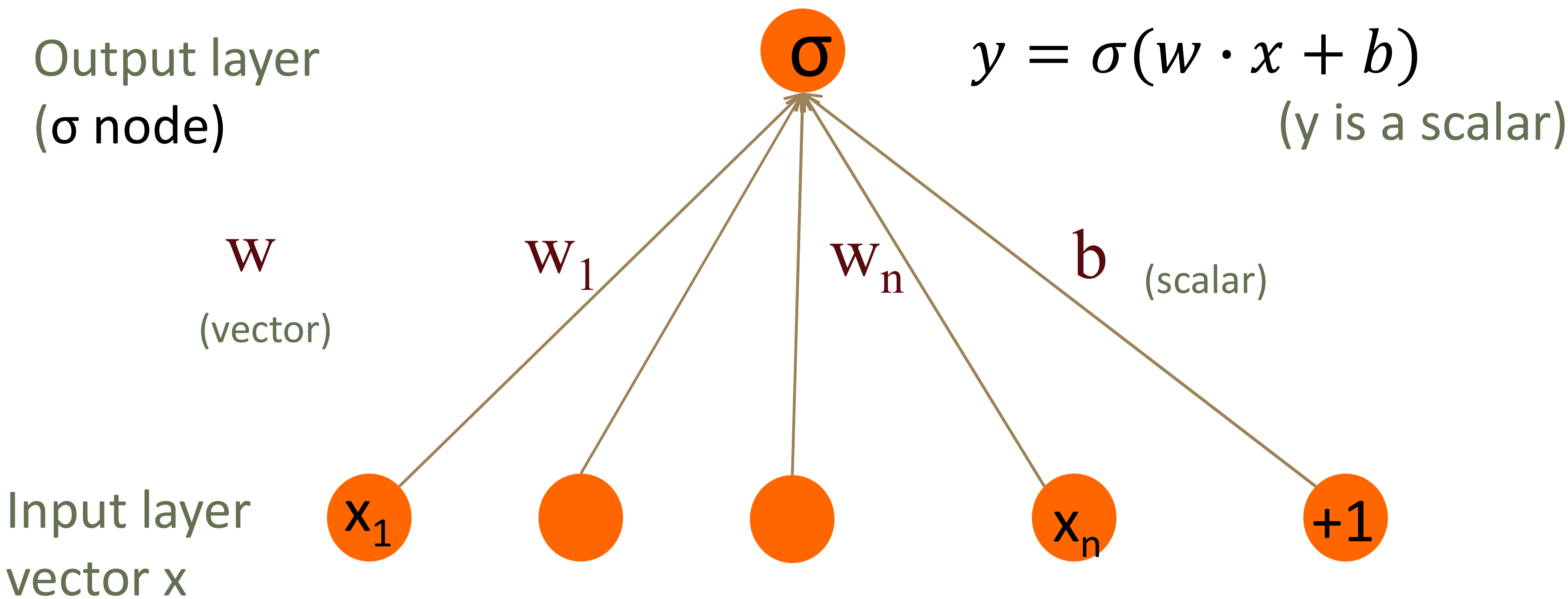
# Feedforward Neural Networks

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



# Binary Logistic Regression as a 1-layer Network

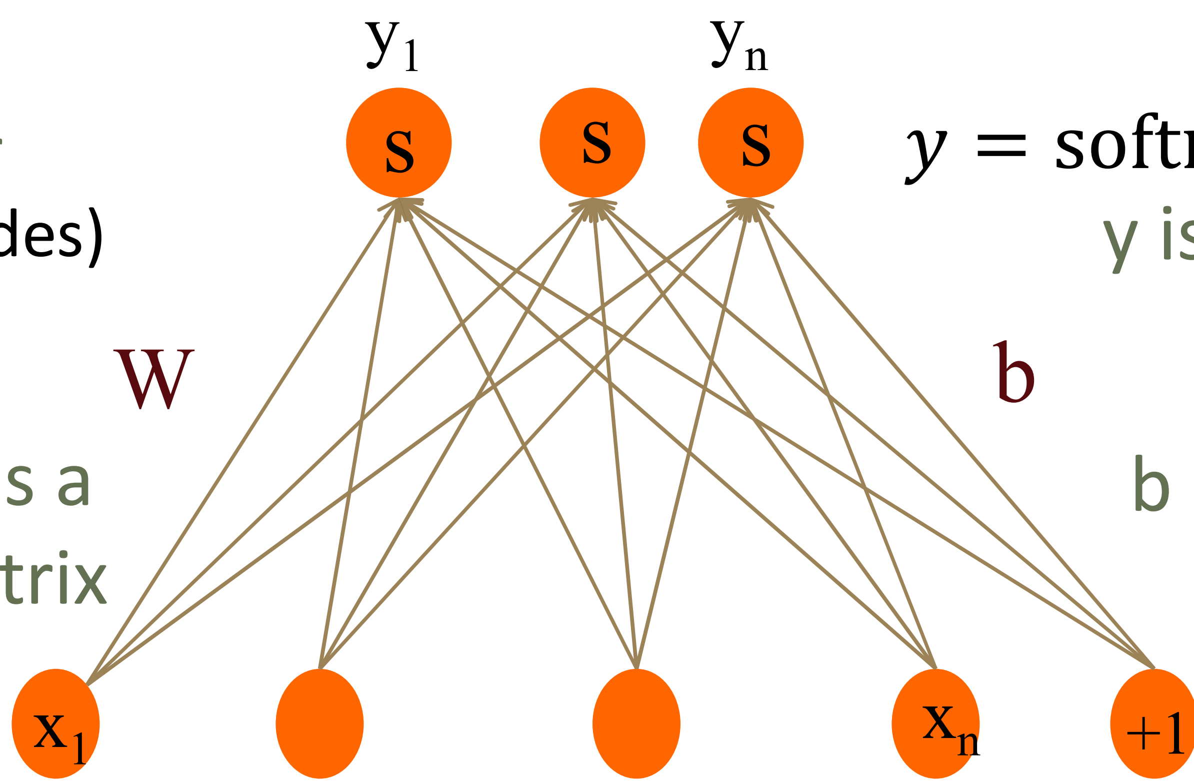
(we don't count the input layer in counting layers!)



# Multinomial Logistic Regression as a 1-layer Network

## Fully connected single layer network

Output layer
(softmax nodes)




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

y is a vector

W is a matrix

b is a vector

Input layer
scalars

# Reminder: softmax: a generalization of sigmoid

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

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

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

Example:

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

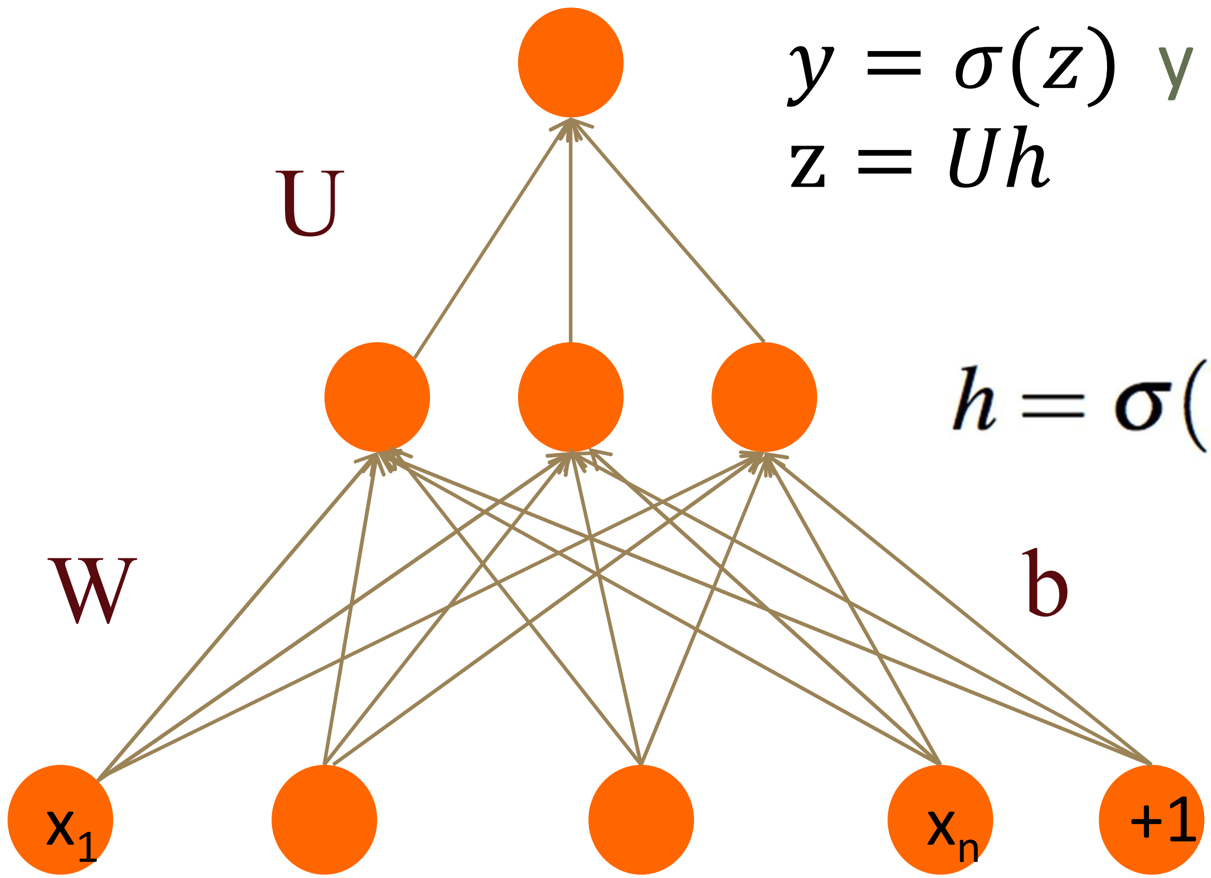
$$\mathrm{softmax}(z) = [0.055, 0.090, 0.006, 0.099, 0.74, 0.010]$$

# Two-Layer Network with scalar output

Output layer
(σ node)

hidden units
(σ node)

Input layer
(vector)



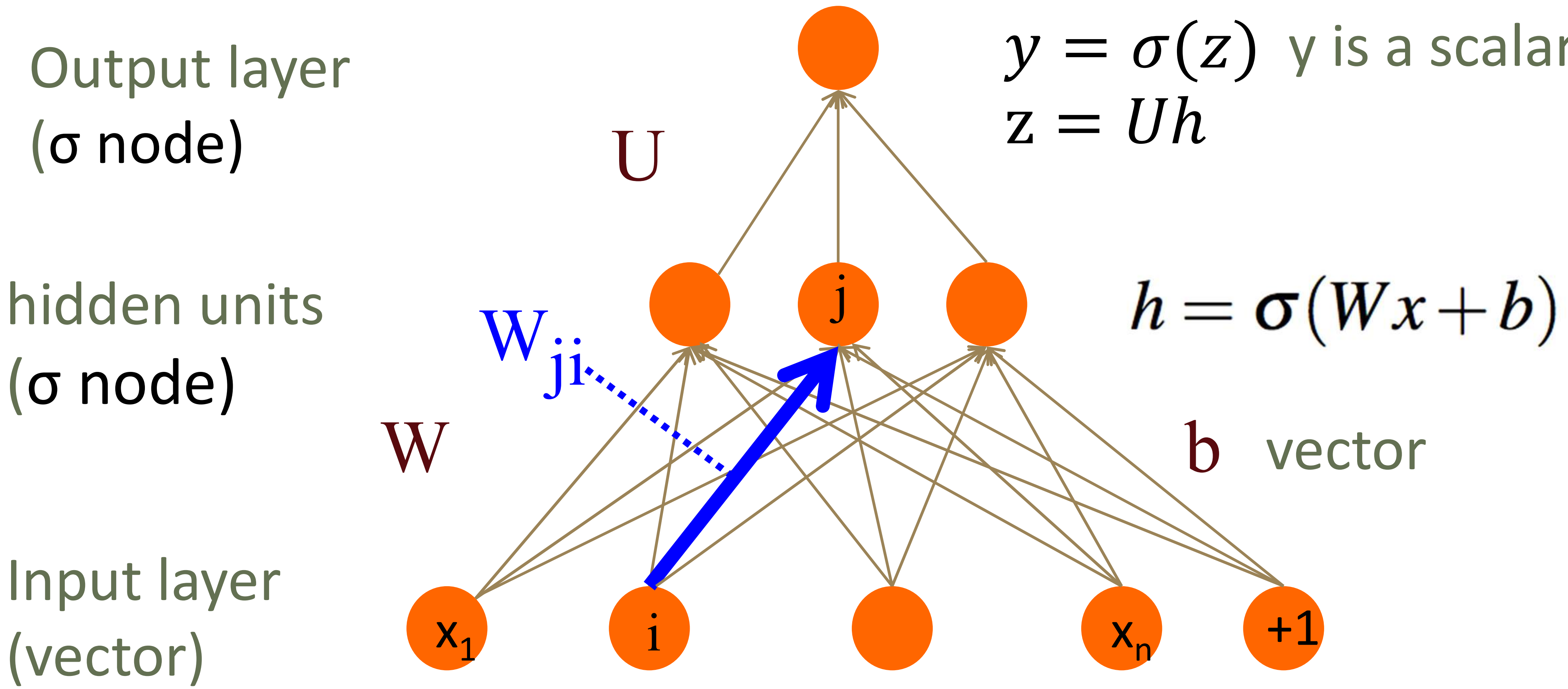

$y = \sigma(z)$ y is a scalar

$z = Uh$

$$h = \boldsymbol{\sigma}(Wx + b)$$

Could be ReLU
Or tanh

# Two-Layer Network with scalar output

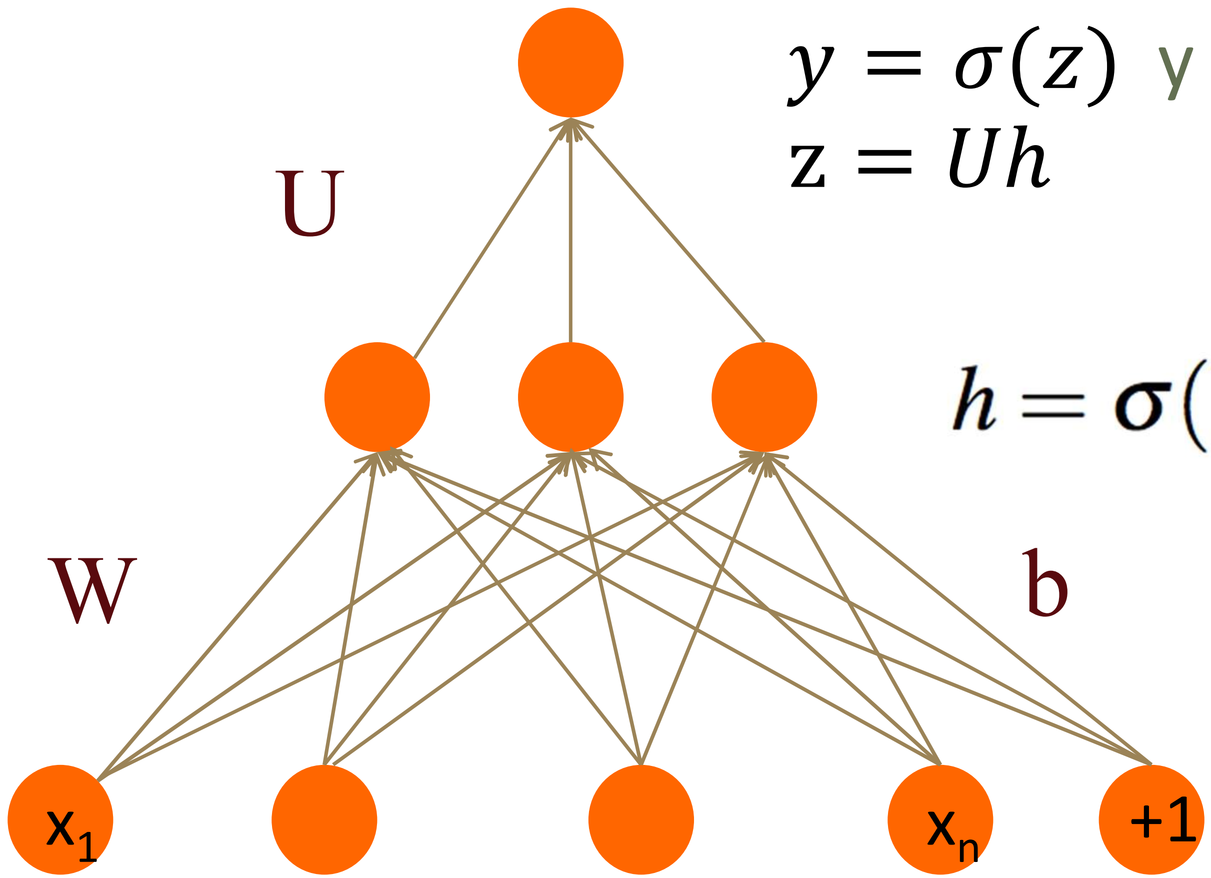


# Two-Layer Network with scalar output

Output layer
(σ node)

hidden units
(σ node)

Input layer
(vector)




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

$$h = \boldsymbol{\sigma}(Wx + b)$$

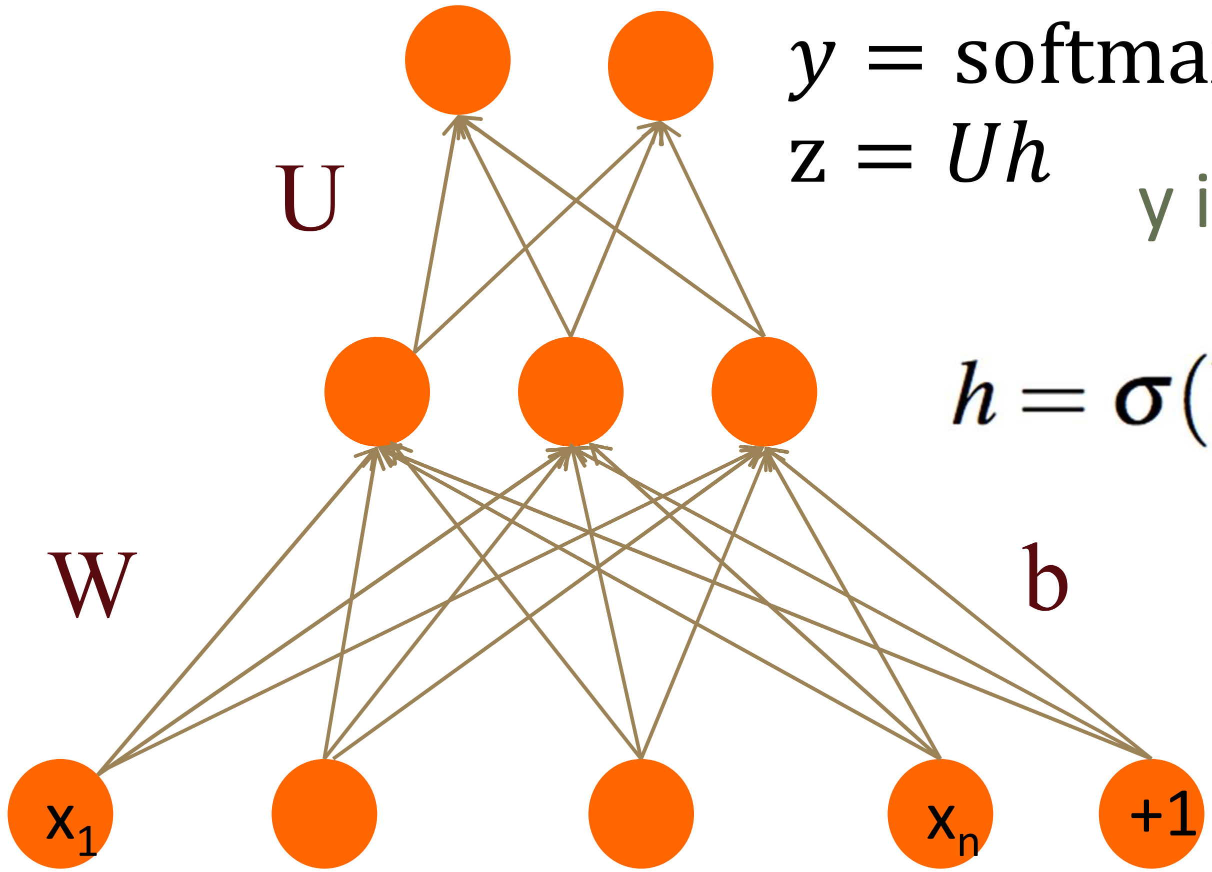
Could be ReLU
Or tanh

# Two-Layer Network with softmax output

Output layer
(σ node)

hidden units
(σ node)

Input layer
(vector)




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

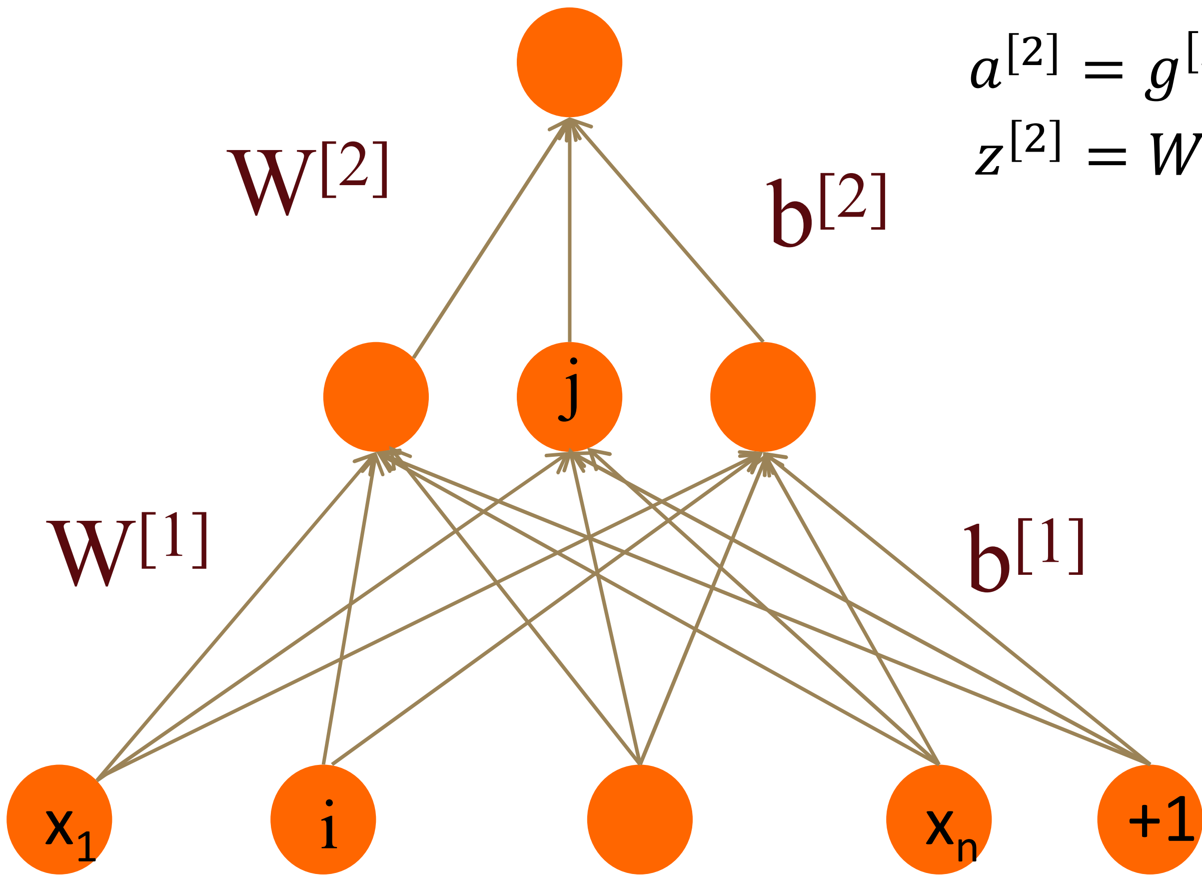
$z = Uh$

y is a vector

$h = \sigma(Wx + b)$

Could be ReLU
Or tanh

# Multi-layer Notation




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

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

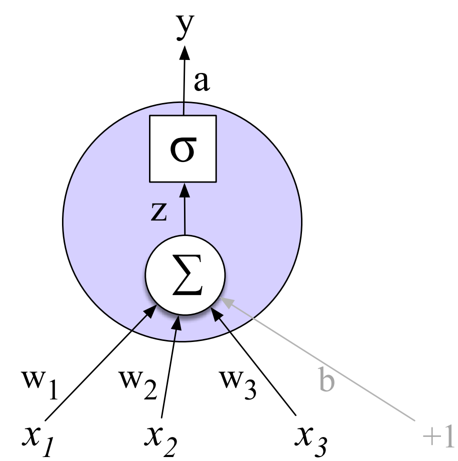
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

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

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

$$a^{[0]}$$

# Multi Layer Notation




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

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

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

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

# 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:

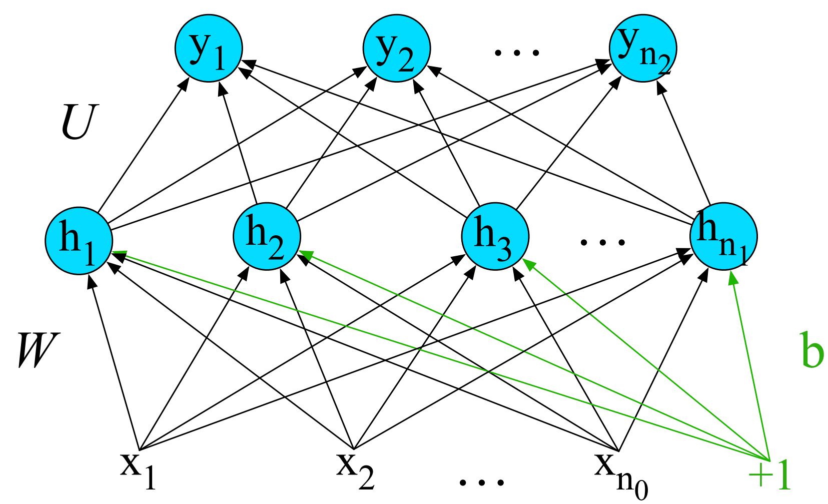
$x = x_0, x_1, x_2, \ldots, x_{n0}$

$$h = \sigma(Wx)$$

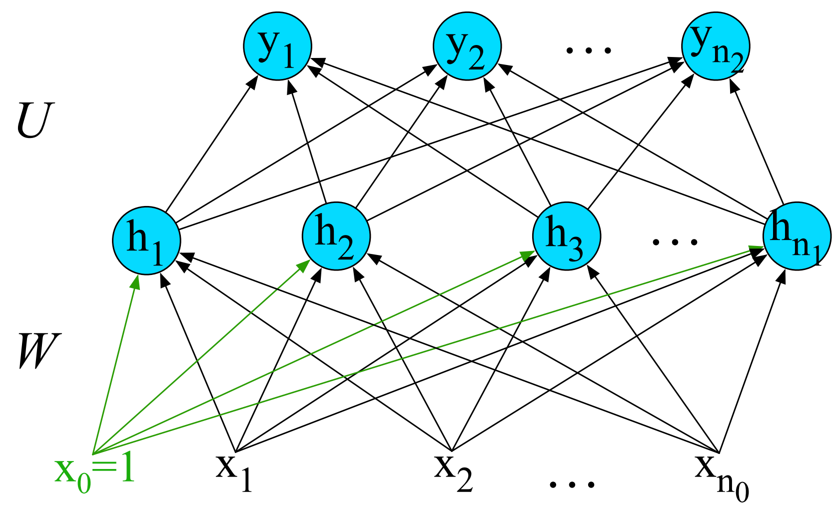
$$\sigma\left(\sum_{i=0}^{n_0} W_{ji} x_i\right)$$

# 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

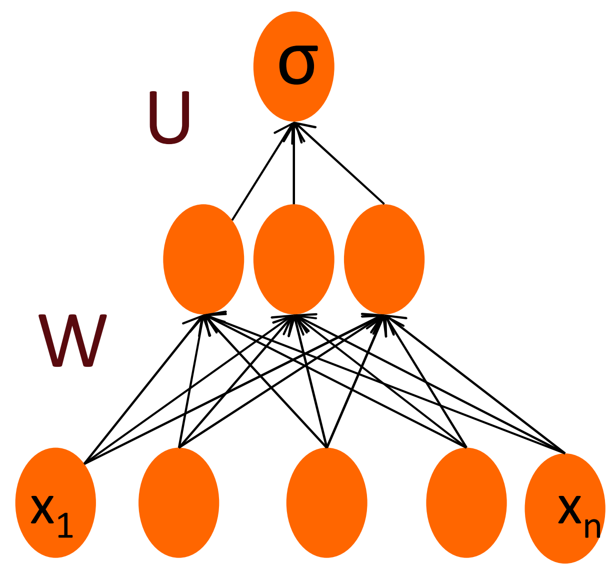
State of the art systems use more powerful neural architectures, but simple models are useful to consider!

# 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

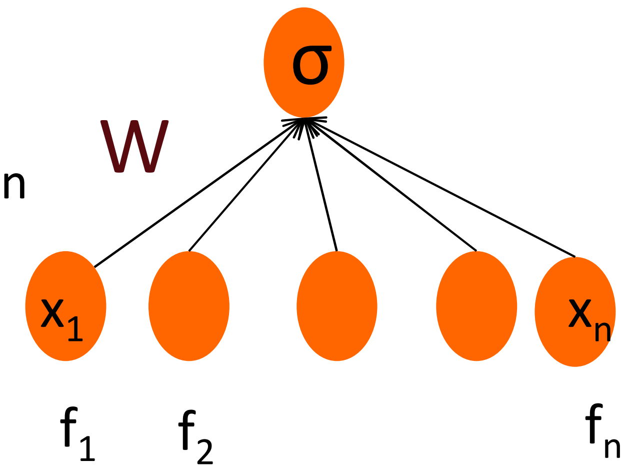


# Sentiment Features

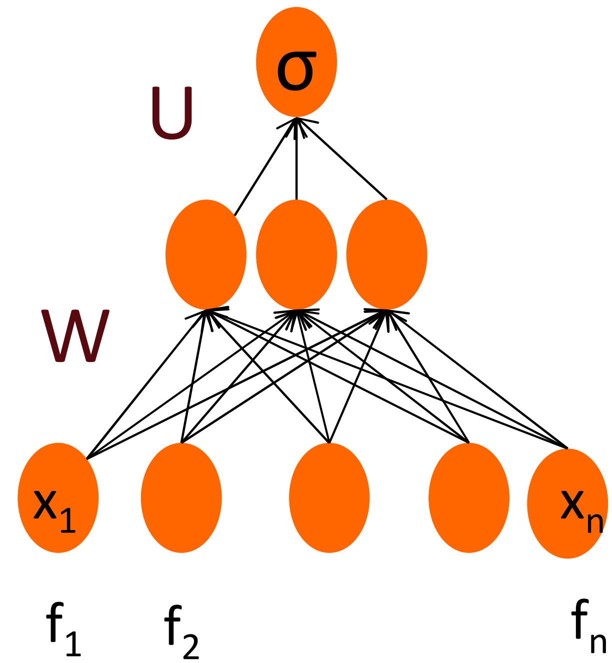
| Var | Definition |
|---|---|
| $x_1$ | $\text{count(positive lexicon)} \in \text{doc)}$ |
| $x_2$ | $\text{count(negative lexicon)} \in \text{doc)}$ |
| $x_3$ | $\begin{cases} 1 & \text{if ``no''} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$ |
| $x_4$ | $\text{count}(\text{1st and 2nd pronouns}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

# Feedforward nets for simple classification

Logistic Regression




2-layer feedforward network




Just adding a hidden layer to logistic regression

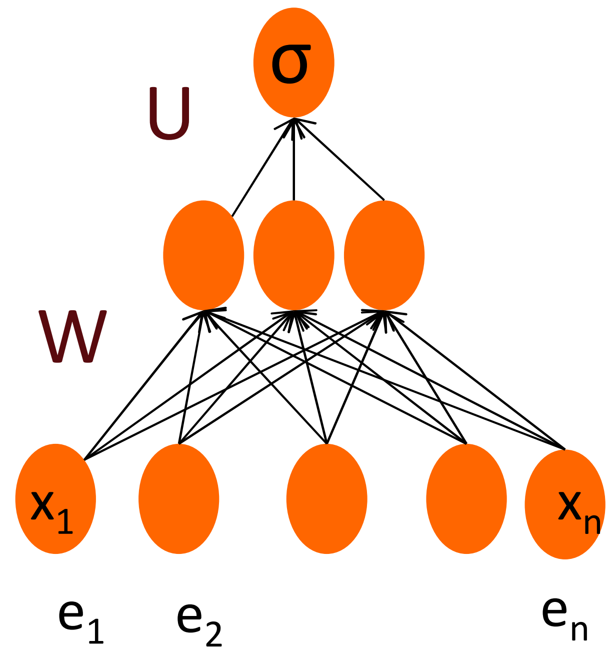
- allows the network to use non-linear interactions between features
- which may (or may not) improve performance.

# Even better: representation learning

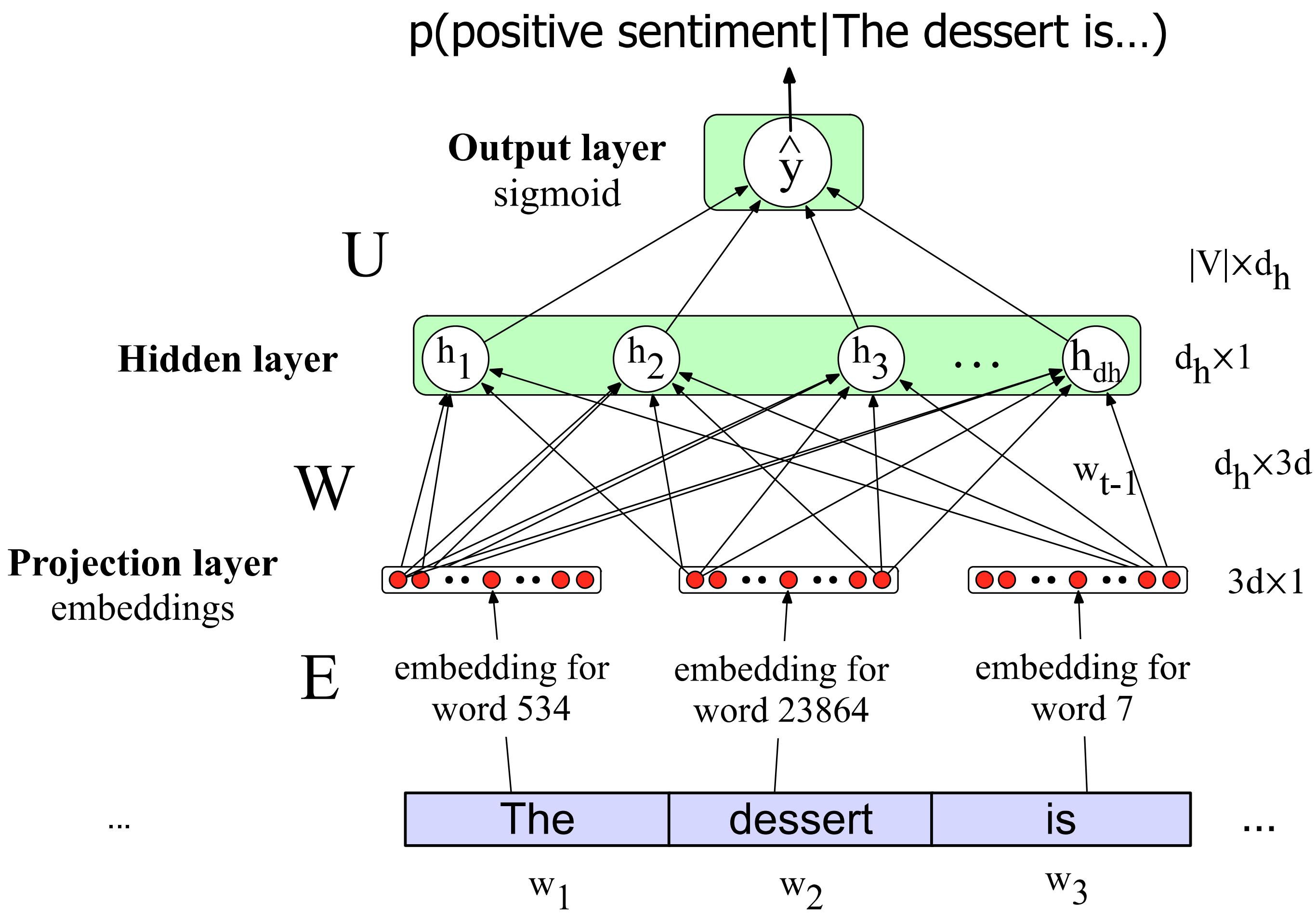
The real power of deep learning comes from the  ability to **learn** features from the data

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

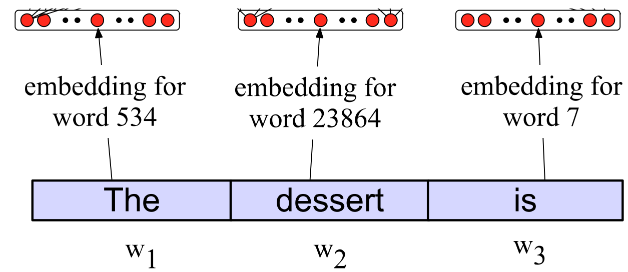
Use learned representations like embeddings!



# 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)

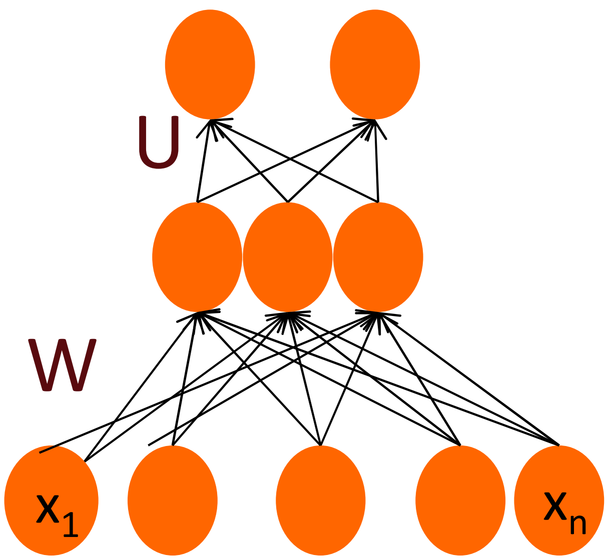
1. Make the input the length of the longest review
   - If shorter then pad with zero embeddings
   - Truncate if you get longer reviews at test time
2. Create a single "sentence embedding" (the same dimensionality as a word) to represent all the words
   - Take the mean of all the word embeddings
 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

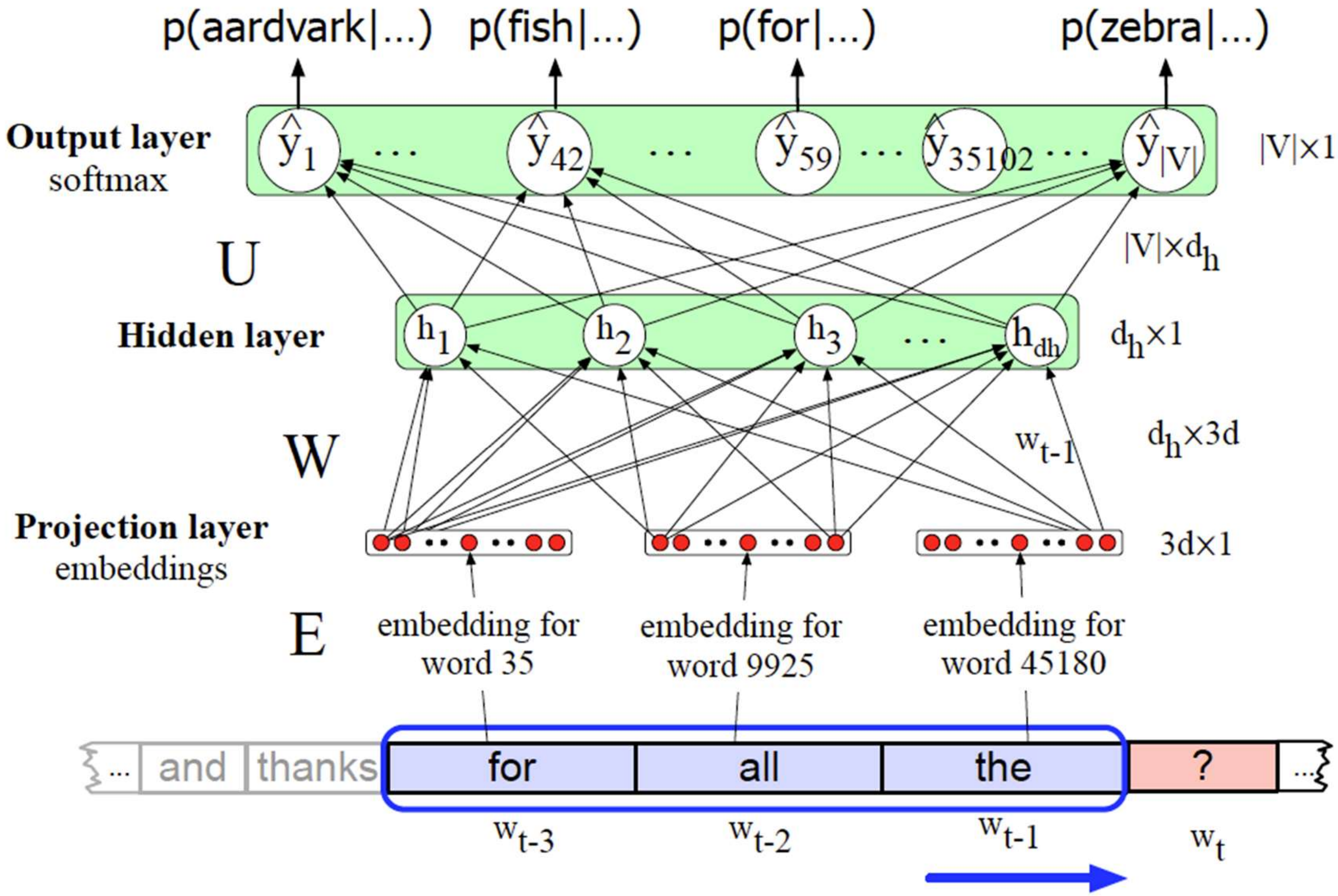
given prior words $w_{t-1}$, $w_{t-2}$, $w_{t-3}$, …

**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

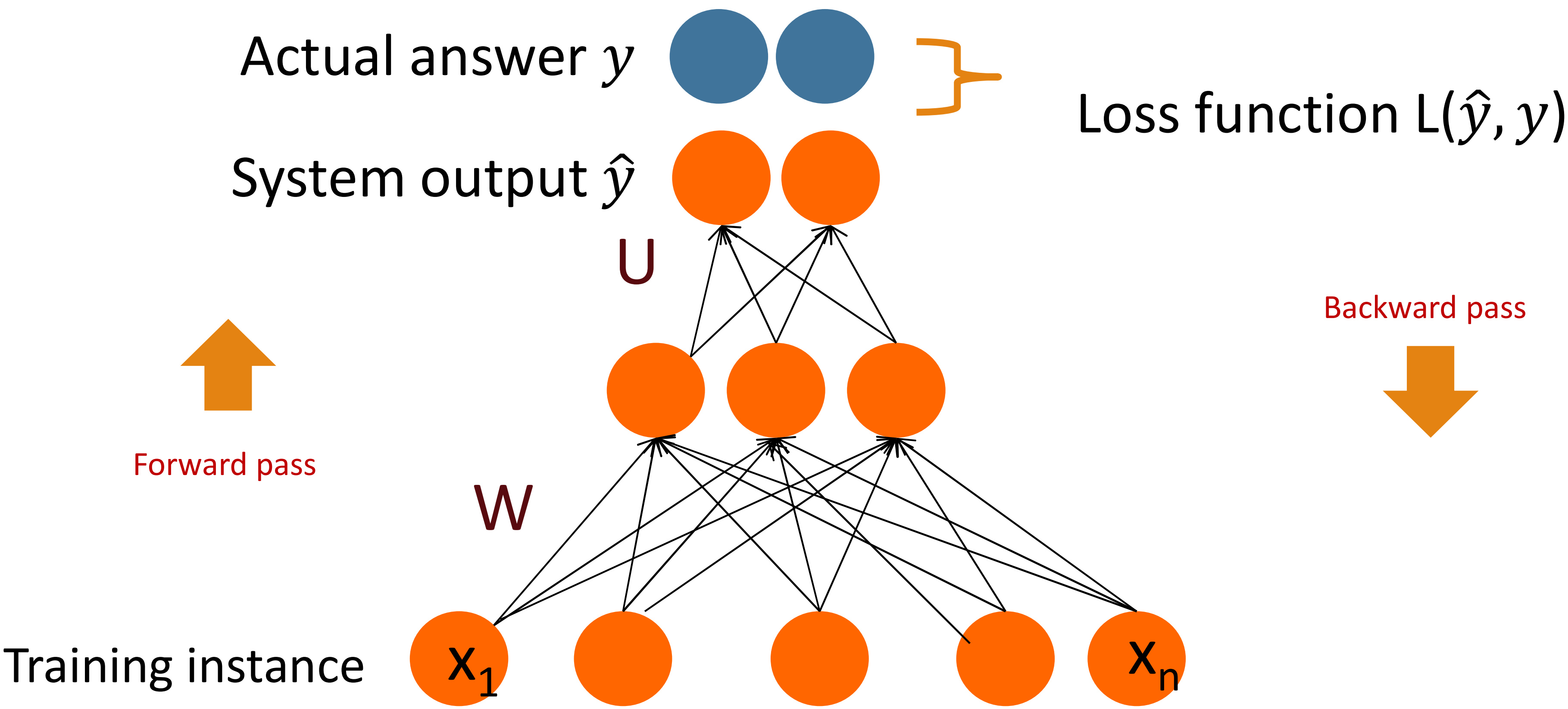
Simple Neural Networks and Neural Language Models

# 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

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

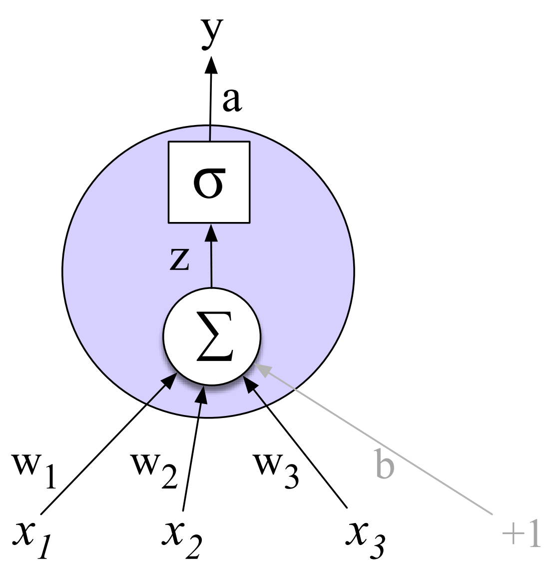
- For logistic regression

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

# Where did that derivative come from?

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

Intuition (see the text for details)




Derivative of the weighted sum

Derivative of the Activation

Derivative of the Loss

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial z} \frac{\partial z}{\partial w_i}$$

# How can I find that gradient for every weight in the network?

These derivatives on the prior slide only give the updates for one weight layer: the last one!

What about deeper networks?

- Lots of layers, different activation functions?

Solution in the next lecture:

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

Simple Neural Networks and Neural Language Models

# Training Neural Nets: Overview

Simple Neural Networks and Neural Language Models

# Computation Graphs and Backward Differentiation

# Why Computation Graphs

For training, we need the derivative of the loss with respect to each weight in every layer of the network

- But the loss is computed only at the very end of the network!

Solution: **error backpropagation** (Rumelhart, Hinton, Williams, 1986)

- **Backprop** is a special case of **backward differentiation**
- Which relies on **computation graphs**.

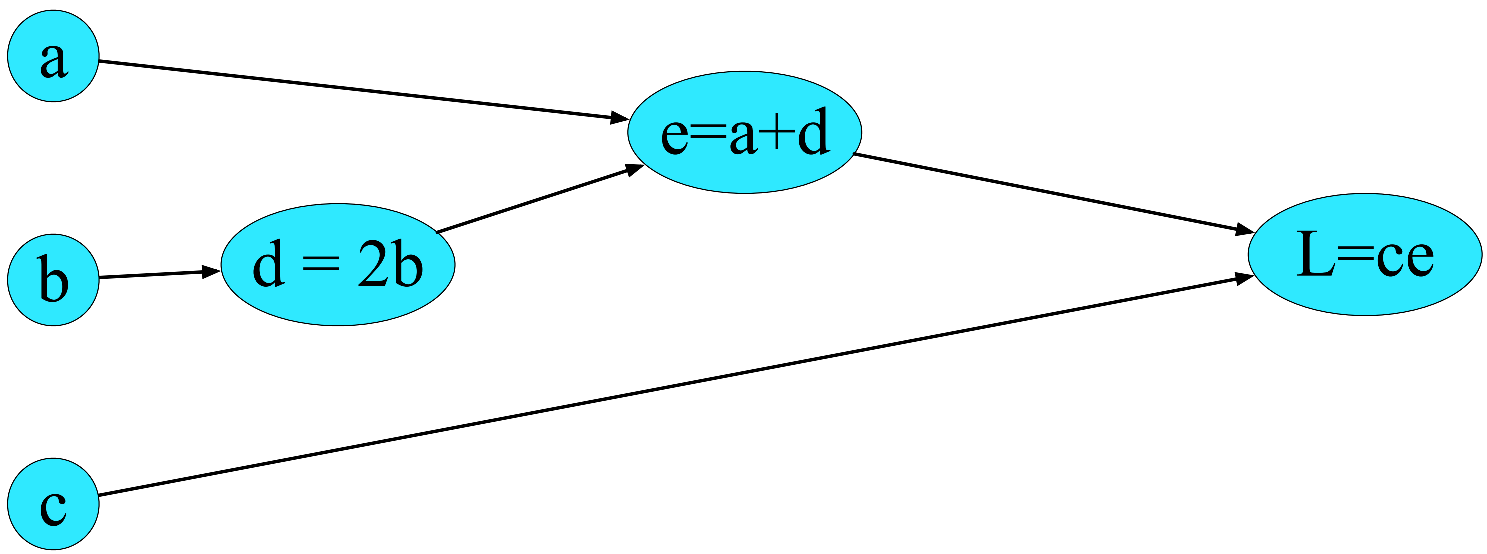
# Computation Graphs

A computation graph represents the process of computing a mathematical expression

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

Computations:

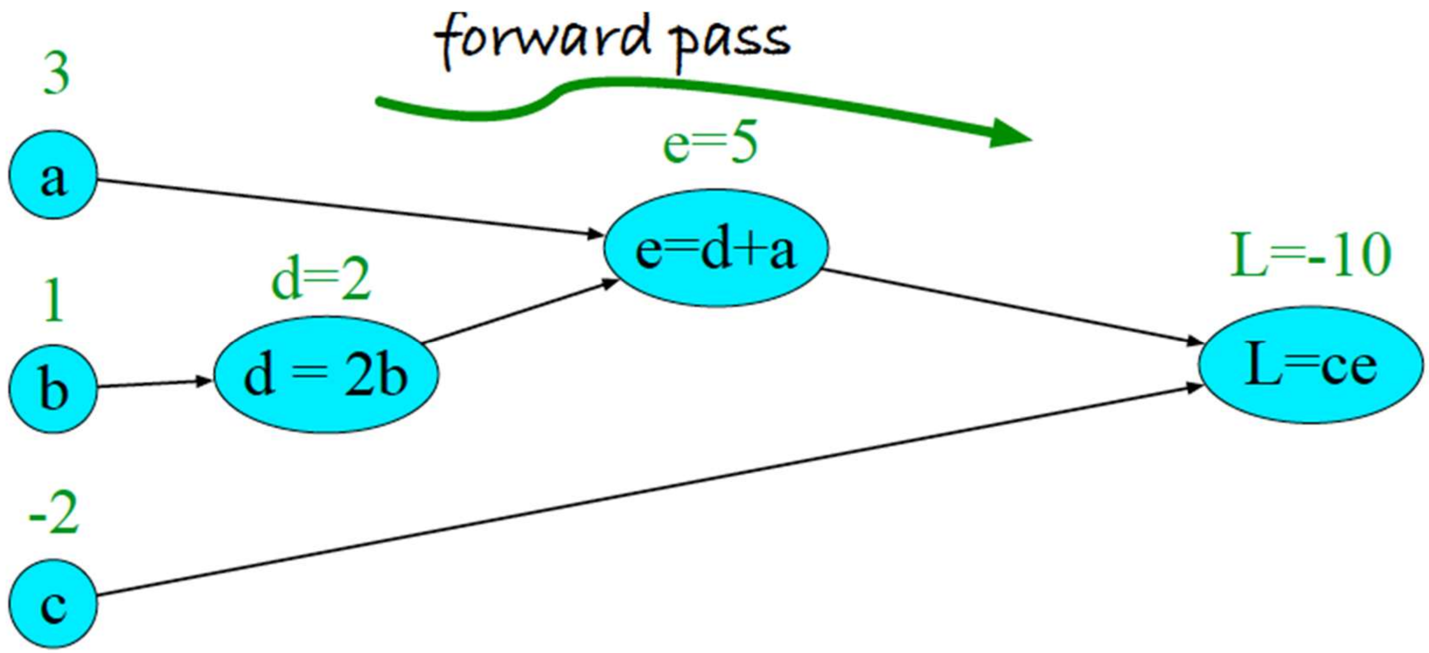
$$\begin{aligned} d &= 2*b \\ e &= a+d \\ L &= c*e \end{aligned}$$



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

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

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

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

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

# Example

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

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

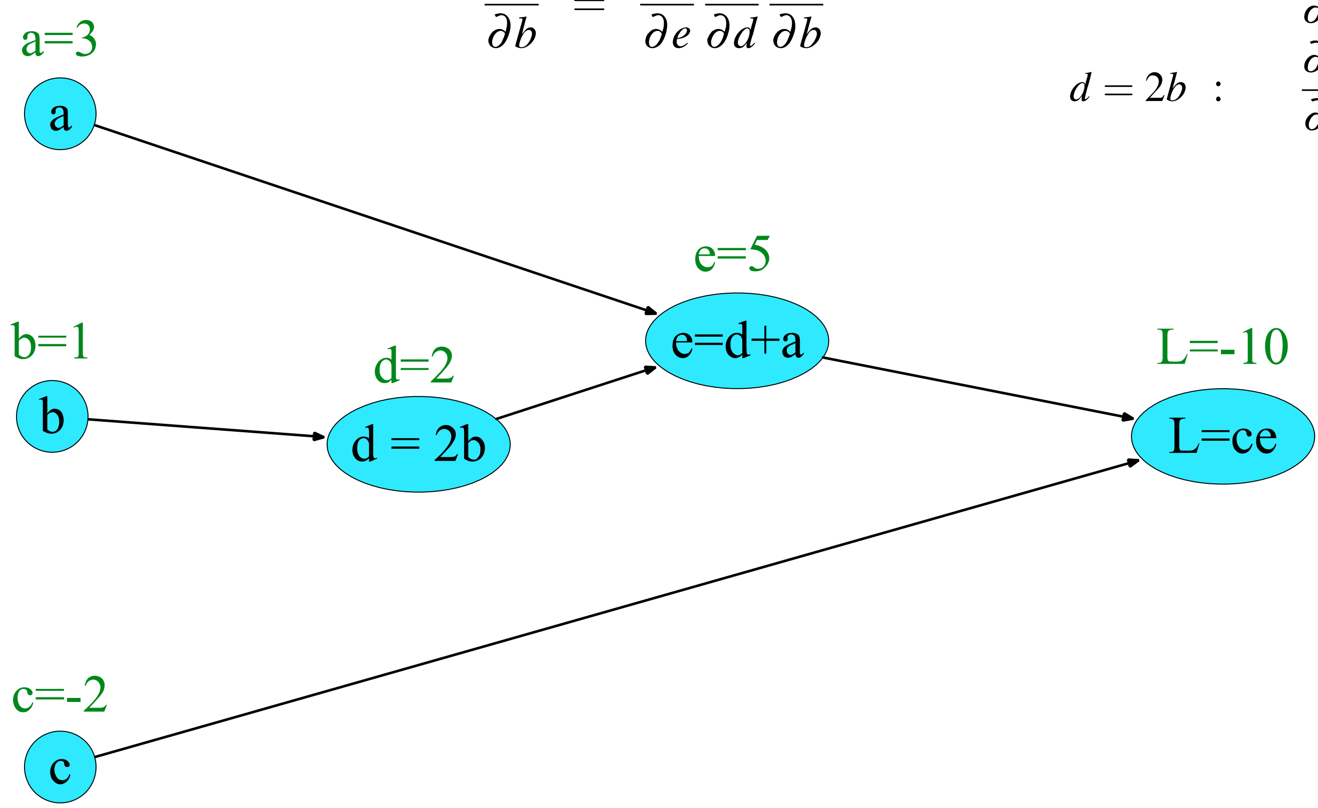
$$\begin{aligned} \frac{\partial L}{\partial a} &= \frac{\partial L}{\partial e}\frac{\partial e}{\partial a} \\ \frac{\partial L}{\partial b} &= \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b} \end{aligned}$$

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

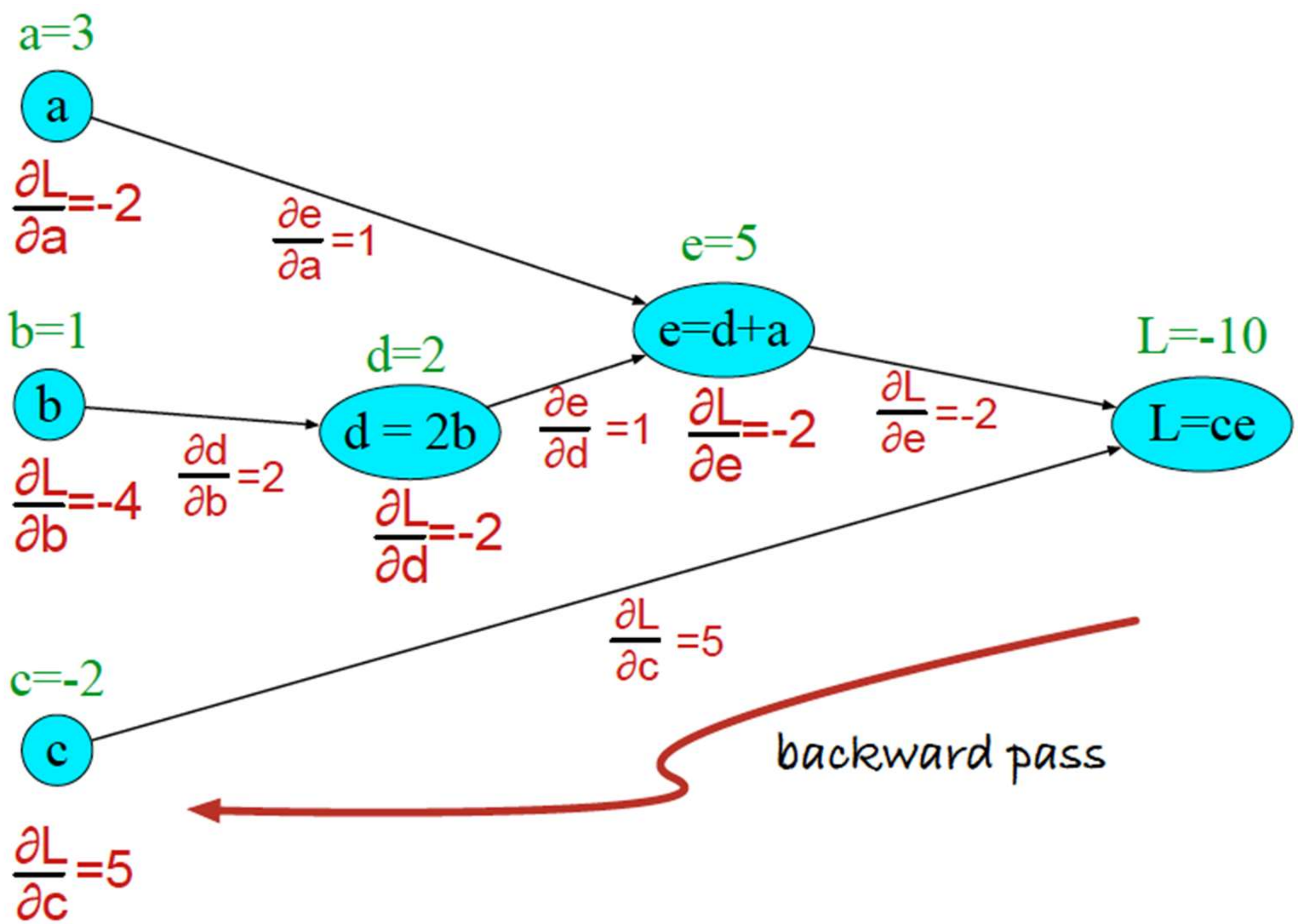
# Example

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

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

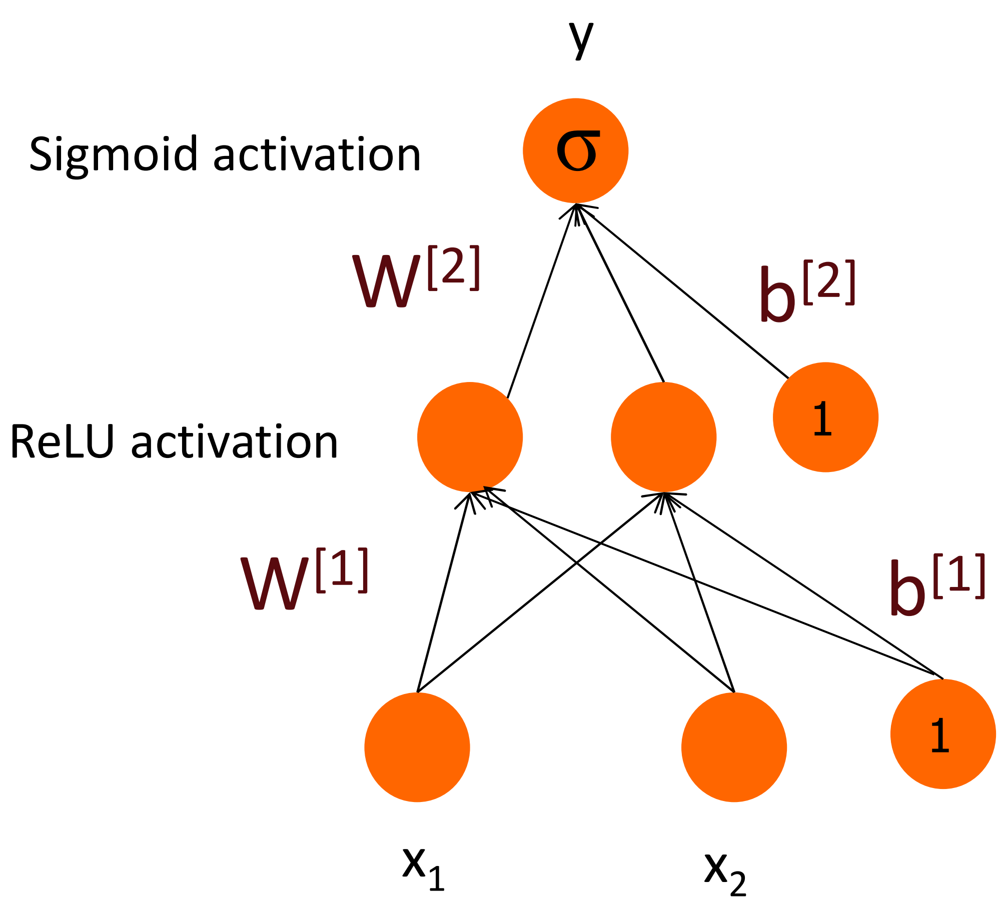


# Example



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




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

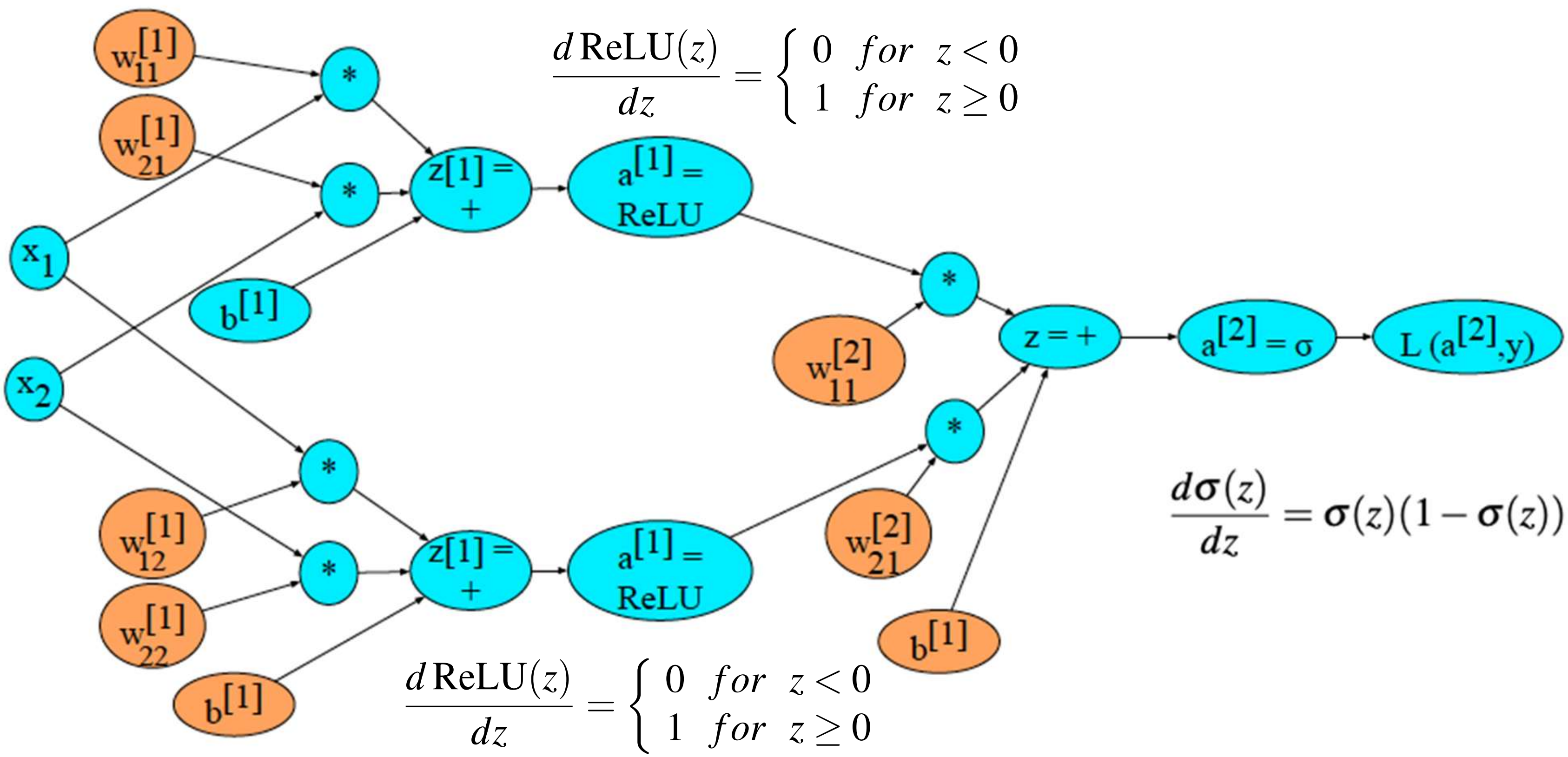
# Backward differentiation on a two layer network

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

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

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

# 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 $z^{[2]}$ )

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

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

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

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

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

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

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

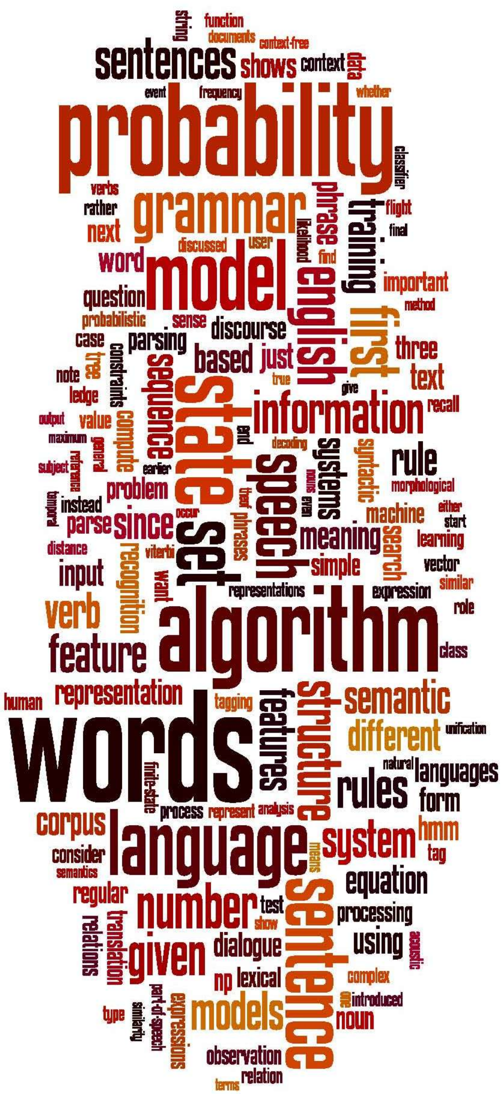
# Summary

For training, we need the derivative of the loss with respect to weights in early layers of the network

- But loss is computed only at the very end of the network!

Solution: **backward differentiation**

Given a computation graph and the derivatives of all the functions in it we can automatically compute the derivative of the loss with respect to these early weights.




# Information Extraction and Named Entity Recognition

Introducing the tasks:

Getting simple structured information out of text

# Information Extraction

Information extraction (IE) systems

- Find and understand limited relevant parts of texts
- Gather information from many pieces of text
- Produce a structured representation of relevant information:
  - *relations* (in the database sense), a.k.a.,
  - a *knowledge base*
- Goals:
  1. Organize information so that it is useful to people
  2. Put information in a semantically precise form that allows further inferences to be made by computer algorithms

# Information Extraction (IE)

IE systems extract clear, factual information

- Roughly: *Who did what to whom when?*

E.g.,

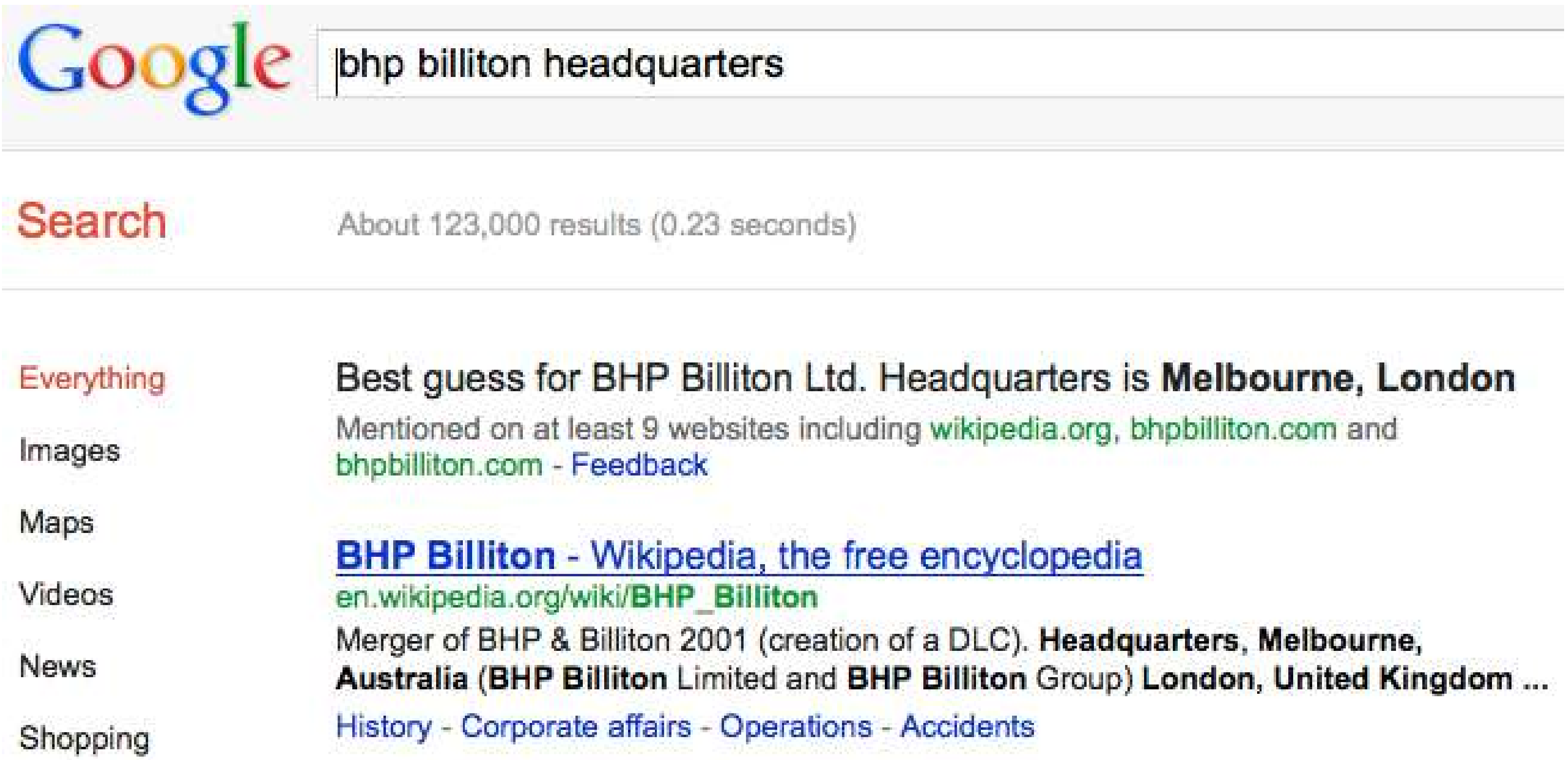
- Gathering earnings, profits, board members, headquarters, etc. from company reports
  - The headquarters of BHP Billiton Limited, and the global headquarters of the combined BHP Billiton Group, are located in Melbourne, Australia.
  - headquarters("BHP Biliton Limited", "Melbourne, Australia")
- Learn drug-gene product interactions from medical research literature

# Low-level information extraction

Is now available – and I think popular – in applications like Apple or Google mail, and web indexing


Often seems to be based on regular expressions and name lists

# Low-level information extraction



# Named Entity Recognition (NER)

A very important sub-task: find and classify names in text, for example:

- The decision by the independent MP Andrew Wilkie to withdraw his support for the minority Labor government sounded dramatic but it should not further threaten its stability. When, after the 2010 election, Wilkie, Rob Oakeshott, Tony Windsor and the Greens agreed to support Labor, they gave just two guarantees: confidence and supply.

# Named Entity Recognition (NER)

A very important sub-task: find and classify names in text, for example:

- The decision by the independent MP Andrew Wilkie to withdraw his support for the minority Labor government sounded dramatic but it should not further threaten its stability. When, after the 2010 election, Wilkie, Rob Oakeshott, Tony Windsor and the Greens agreed to support Labor, they gave just two guarantees: confidence and supply.

# Named Entity Recognition (NER)

A very important sub-task: find and classify names in text, for example:

- The decision by the independent MP Andrew Wilkie to withdraw his support for the minority Labor government sounded dramatic but it should not further threaten its stability. When, after the 2010 election, Wilkie, Rob Oakeshott, Tony Windsor and the Greens agreed to support Labor, they gave just two guarantees: confidence and supply.

Person
Date
Location
Organi-
zation

# Named Entity Recognition (NER)

The uses:

- Named entities can be indexed, linked off, etc.
- Sentiment can be attributed to companies or products
- A lot of IE relations are associations between named entities
- For question answering, answers are often named entities.

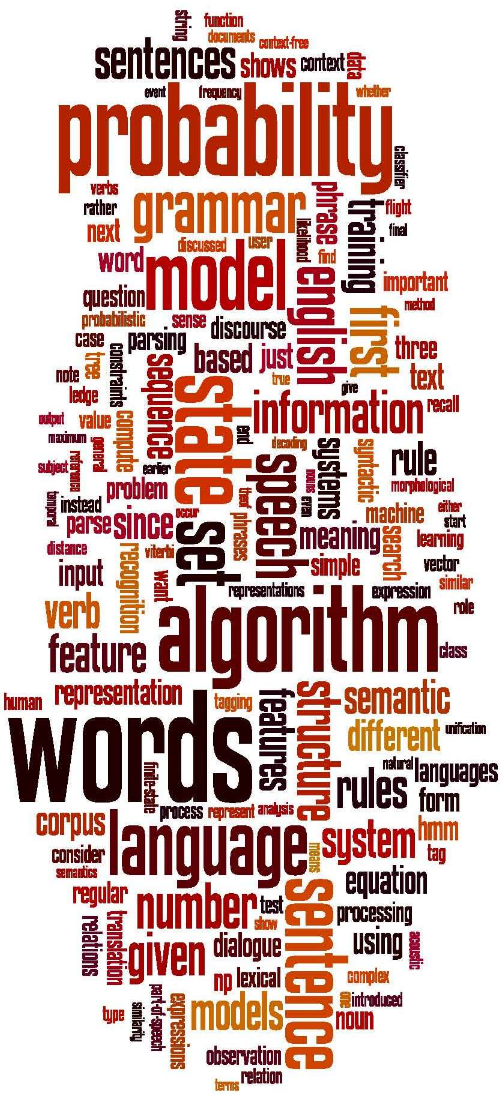
Concretely:

- Many web pages tag various entities, with links to bio or topic pages, etc.
  - Reuters' OpenCalais, Evri, AlchemyAPI, Yahoo's Term Extraction, …
- Apple/Google/Microsoft/… smart recognizers for document content


# Information Extraction and Named Entity Recognition

Introducing the tasks:

Getting simple structured information out of text




# Evaluation of Named Entity Recognition

The extension of Precision, Recall, and the F measure to sequences

# The Named Entity Recognition Task

Task: Predict entities in a text

| | |
|---|---|
| Foreign | ORG |
| Ministry | ORG |
| spokesman | O |
| Shen | PER |
| Guofang | PER |
| told | O |
| Reuters | ORG |
| : | : |

Standard evaluation is per entity, *not* per token

# Precision/Recall/F1 for IE/NER

Recall and precision are straightforward for tasks like IR and text categorization, where there is only one grain size (documents)

The measure behaves a bit funnily for IE/NER when there are *boundary errors* (which are *common*):

- First Bank of Chicago announced earnings …

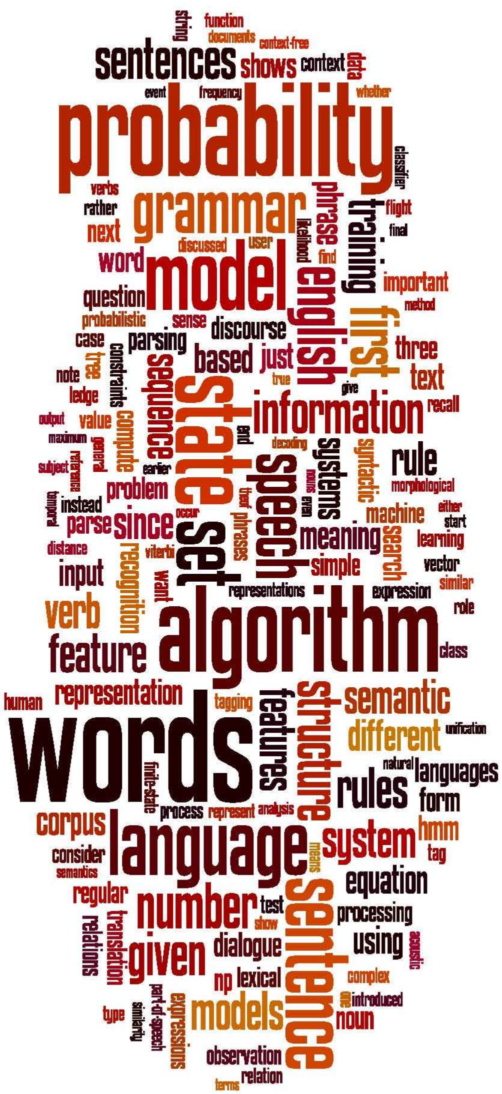
This counts as both a fp and a fn

Selecting *nothing* would have been better

Some other metrics (e.g., MUC scorer) give partial credit (according to complex rules)


# Evaluation of Named Entity Recognition

The extension of Precision, Recall, and the F measure to sequences




# Sequence Models for Named Entity Recognition

# The ML sequence model approach to NER

Training

1. Collect a set of representative training documents
2. Label each token for its entity class or other (O)
3. Design feature extractors appropriate to the text and classes
4. Train a sequence classifier to predict the labels from the data

Testing

1. Receive a set of testing documents
2. Run sequence model inference to label each token
3. Appropriately output the recognized entities

# Encoding classes for sequence labeling

| | IO encoding | IOB encoding |
|---|---|---|
| Fred | PER | B-PER |
| showed | O | O |
| Sue | PER | B-PER |
| Mengqiu | PER | B-PER |
| Huang | PER | I-PER |
| ‘s | O | O |
| new | O | O |
| painting | O | O |

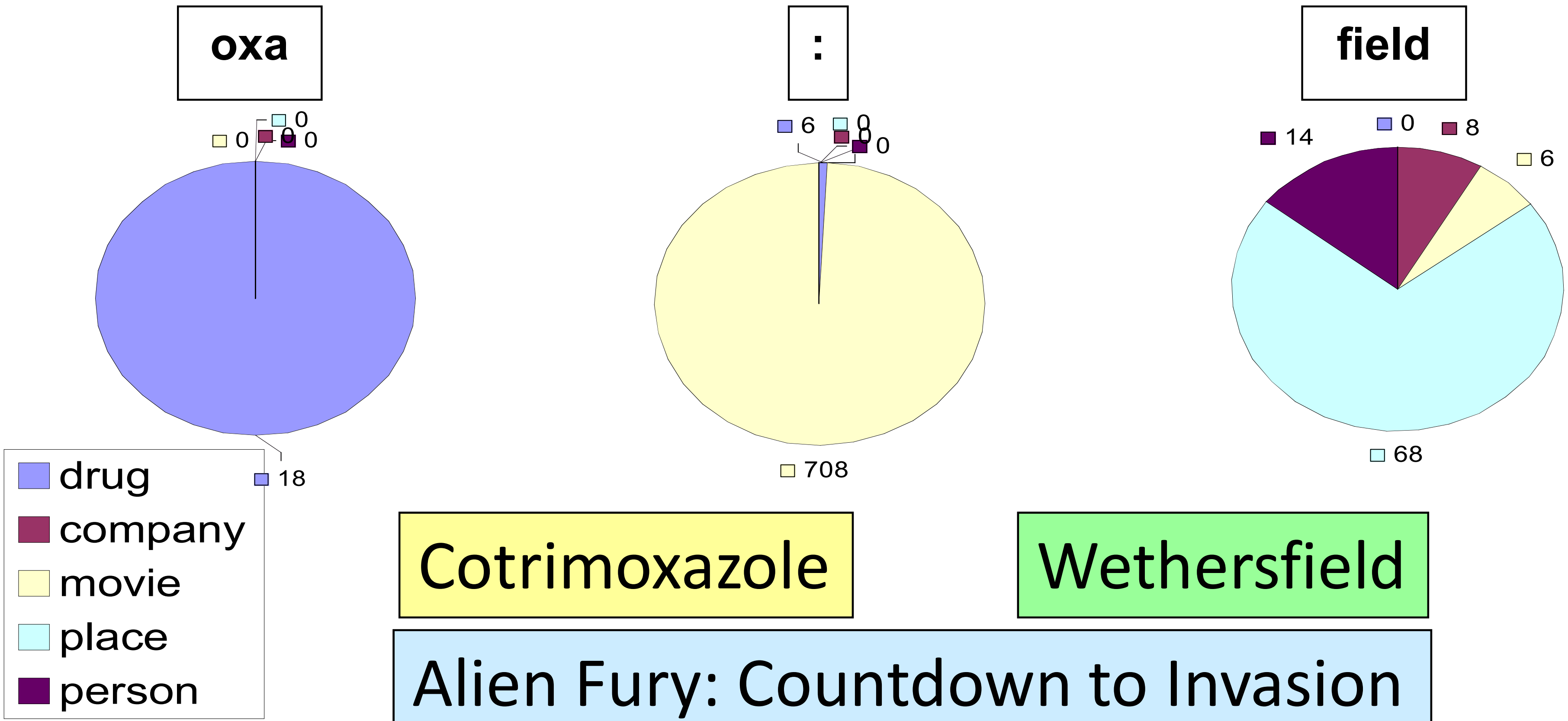
# Features for sequence labeling

Words
- Current word (essentially like a learned dictionary)
- Previous/next word (context)

Other kinds of inferred linguistic classification
- Part-of-speech tags

Label context
- Previous (and perhaps next) label

# Features: Word substrings




Cotrimoxazole

Wethersfield

Alien Fury: Countdown to Invasion

# Features: Word shapes

Word Shapes

- Map words to simplified representation that encodes attributes such as length, capitalization, numerals, Greek letters, internal punctuation, etc.

| | |
|---|---|
| Varicella-zoster | Xx-xxx |
| mRNA | xXXX |
| CPA1 | XXXd |


# Sequence Models for Named Entity Recognition