CS 1671/2071
Human Language Technologies

Session 12: Neural networks, part 1

Michael Miller Yoder

February 19, 2025

University of Pittsburgh | School of Computing and Information

Course logistics

- Homework 2 is due **tomorrow, Thu Feb 20**
- If I emailed your group about choosing different directions or datasets and I haven't heard from you, I'll check in with you this week
- Next project milestone: project proposal due Feb 28
  - I will release instructions for that soon (sorry!)
  - Start thinking about how you would apply approaches we have covered so far (n-gram feature extraction, logistic regression, n-gram language modeling) to your task
  - Feel free to email or book office hours with Michael to discuss

Midterm course evaluation (OMETs)

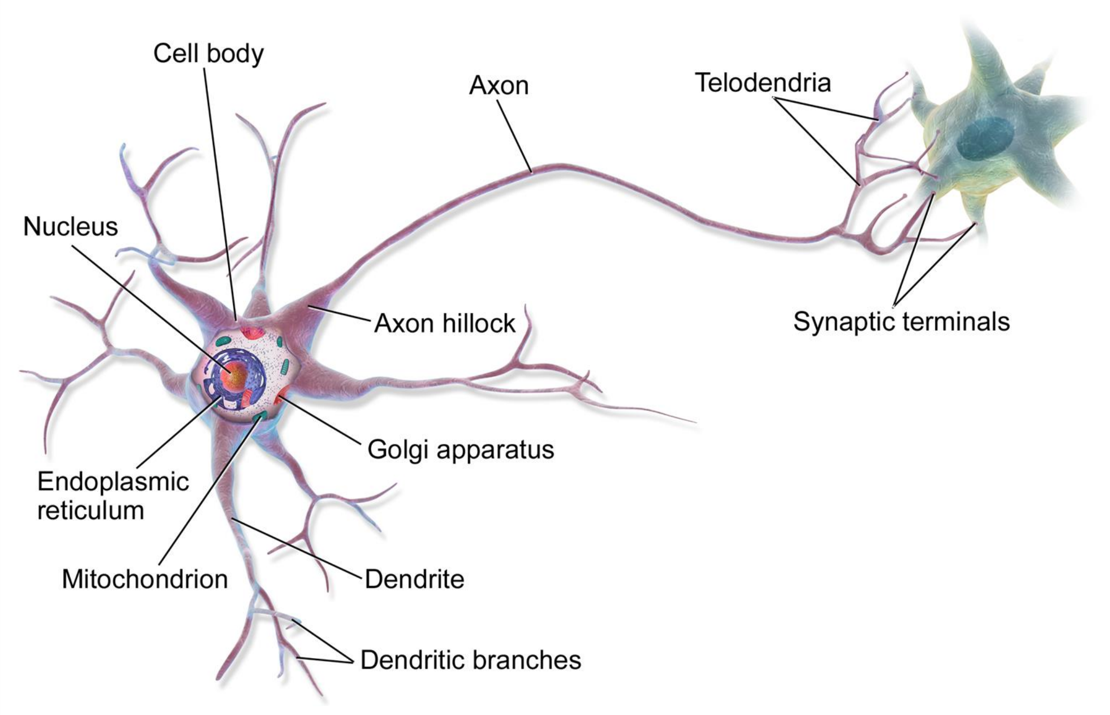
- CS 1671: https://go.blueja.io/BJVNkUaUE0WIdL6VHILkXQ
- CS 2071: https://go.blueja.io/fiEDPP0eM0eQ3kzYBucv6w
- All types of feedback are welcome (critical and positive)
- Completely anonymous, will not affect grades
- Let me know what's working and what to improve on while the course is still running!
- Please be as specific as possible
- Available until **tonight, Wed Feb 19 at 11:59pm**

Lecture overview: neural networks, part 1

- Neural network fundamentals
- Non-linear activation functions
- Feedforward neural networks as classifiers
- Coding activity

Neural network fundamentals

This is in your brain



Neural network unit: This is not in your brain

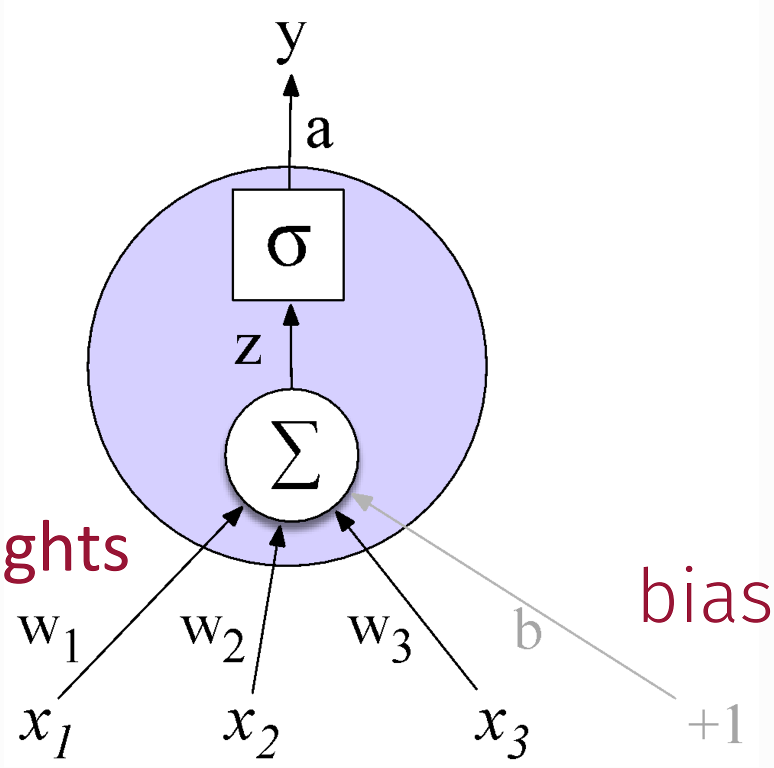
Output value

Non-linear transform

Weighted sum

Weights

Input layer



bias

The Variables in Our Very Important Formula

$\mathbf{x}$ A vector of features of n dimensions (like number of positive sentiment words, length of document, etc.)

$\mathbf{w}$ A vector of weights of n dimensions specifying how discriminative each feature is

b A scalar bias term that shifts z

z The raw score

y A random variable (e.g., $y = 1$ means positive sentiment and $y = 0$ means negative sentiment

Slide adapted from David Mortensen

The Fundamentals

The fundamental equation that describes a unit of a neural network should look very familiar:

$$z = b + \sum_i w_i x_i \tag{1}$$

Which we will represent as

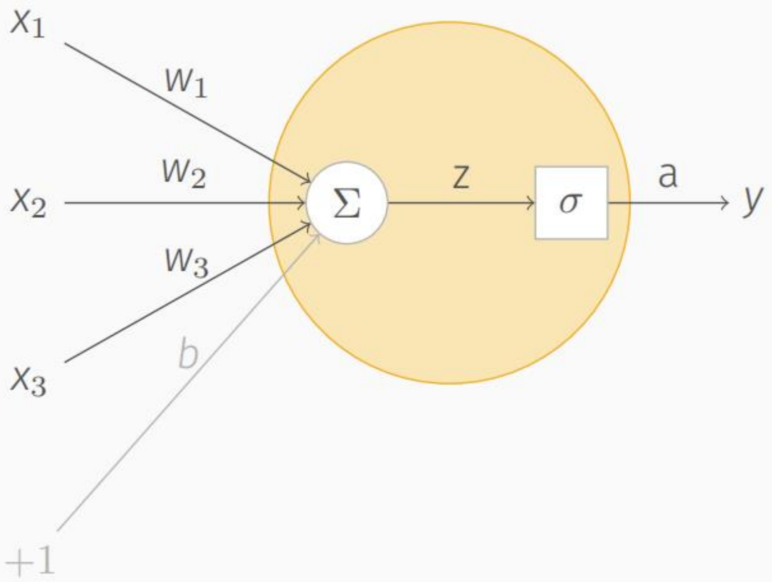
$$z = \mathbf{w} \cdot \mathbf{x} + b \tag{2}$$

But we do not use z directly. Instead, we pass it through a non-linear function, like the sigmoid function:

$$y = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{3}$$

(which has some nice properties even though, in practice, we will prefer other functions like tanh and ReLU).

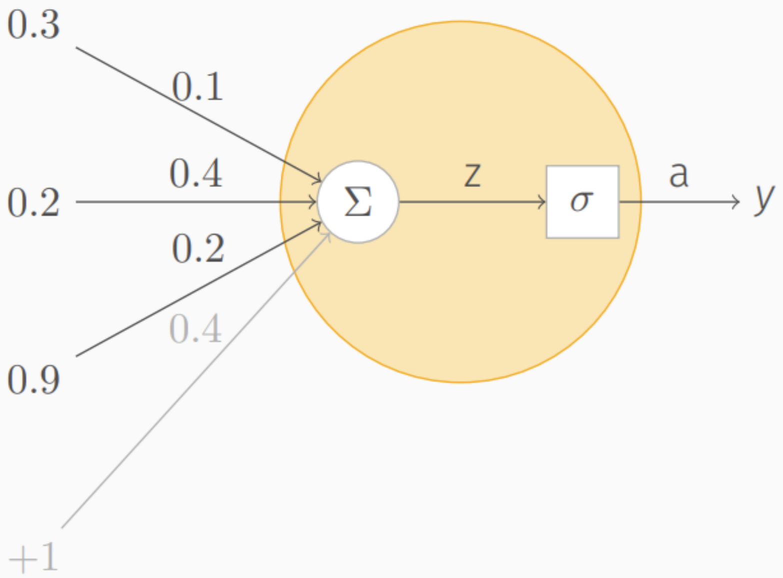
Slide adapted from David Mortensen



Take, for example, a scenario in which our unit has the weights [0.1, 0.4, 0.2] and the bias term 0.4 and the input vector x has the values [0.3, 0.2, 0.9].

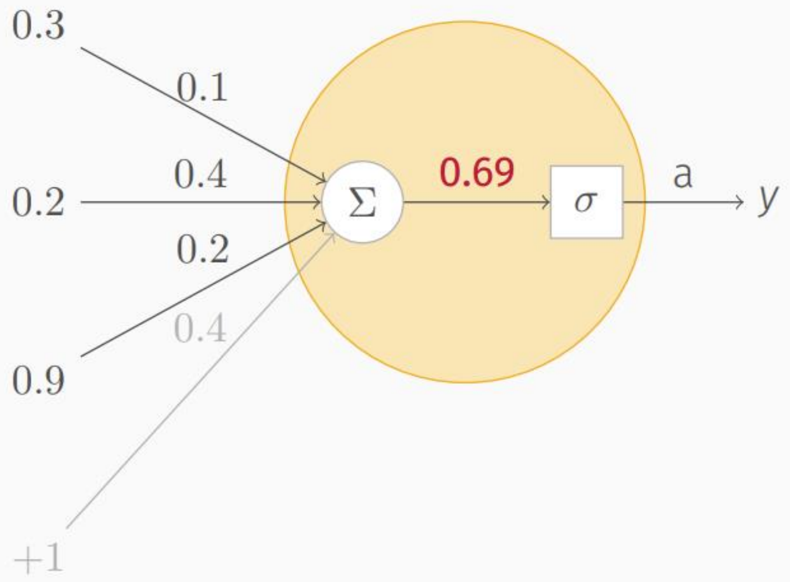
Slide adapted from David Mortensen

Filling in the Input Values and Weights



Slide adapted from David Mortensen

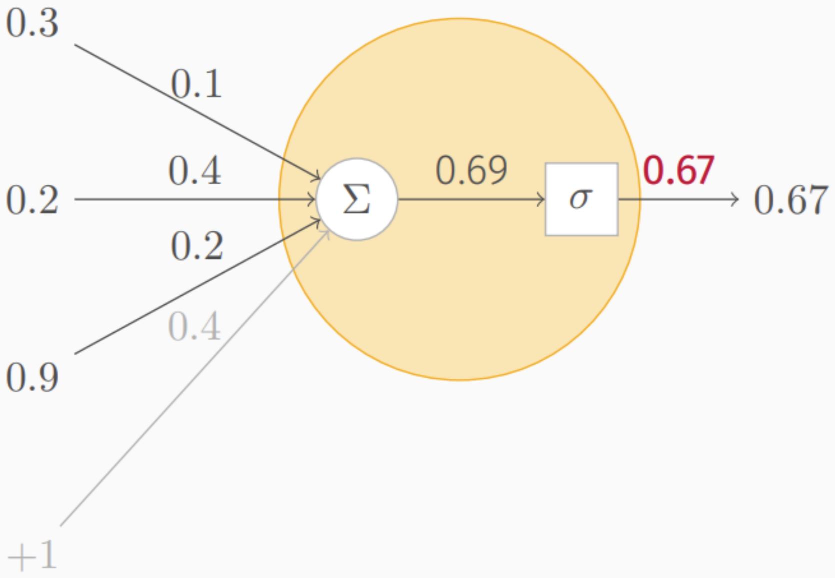
Multiplying the Input Values and Weights and Summing Them (with the Bias Term)



$$z = x_1 w_1 + x_2 w_2 + x_3 w_3 + b = 0.1(0.3) + 0.4(0.2) + 0.2(0.9) + 0.4 = 0.69 \quad (4)$$

Slide adapted from David Mortensen

Applying the Activation Function (Sigmoid)



$$y = \sigma(0.69) = \frac{1}{1 + e^{-0.69}} = 0.67 \tag{5}$$

Slide adapted from David Mortensen

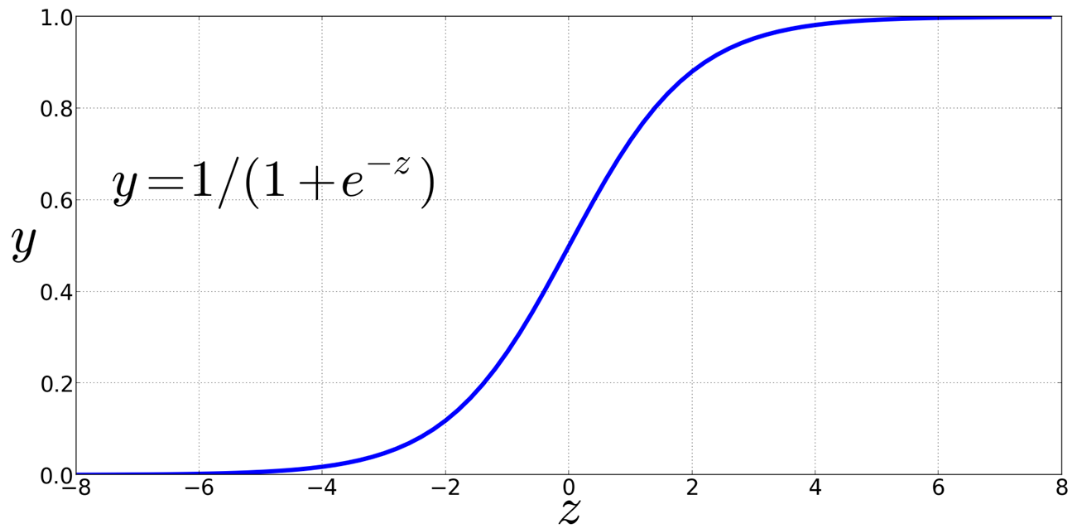
Non-linear activation functions

Non-Linear Activation Functions

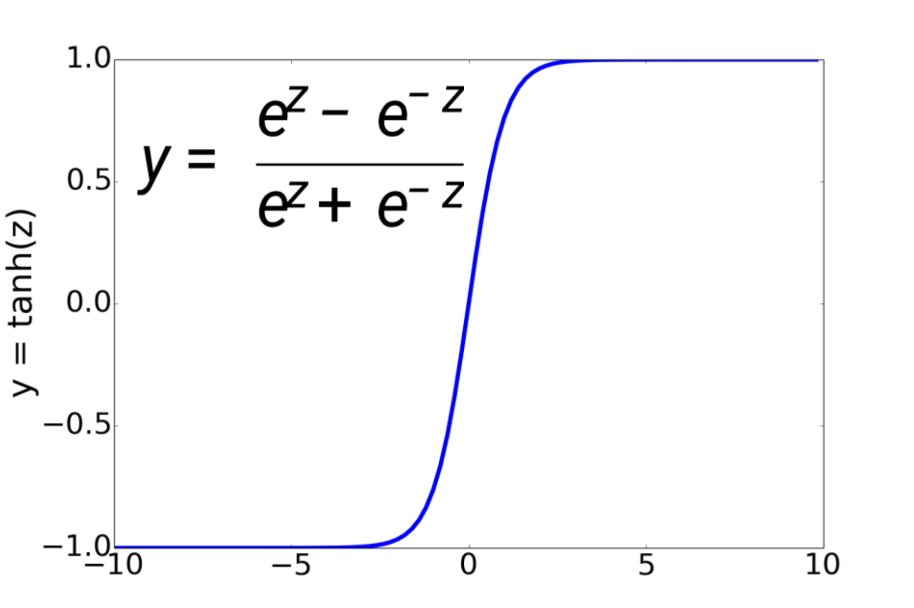
We're already seen the sigmoid for logistic regression:

Sigmoid

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

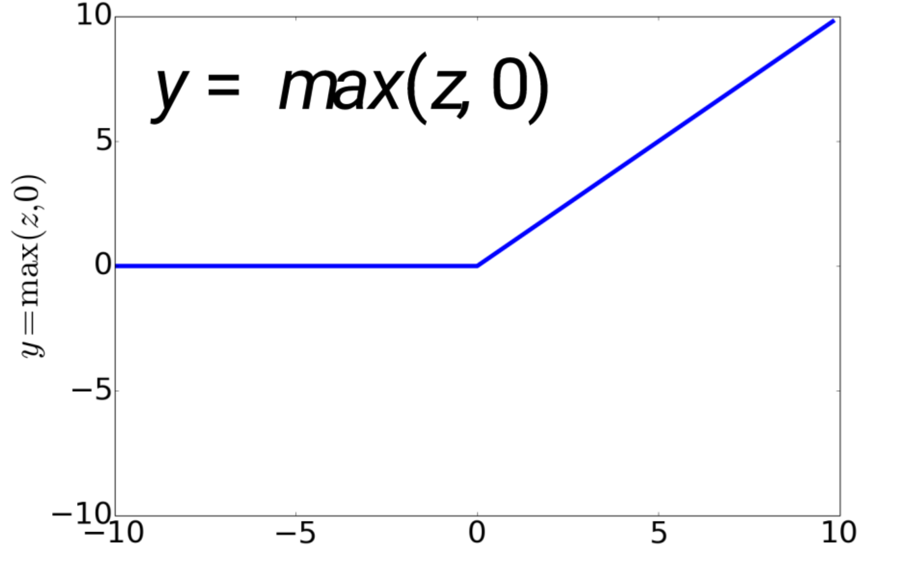


Nonlinear activation functions besides sigmoid



tanh

Most common:



ReLU
Rectified Linear

Slide adapted from Jurafsky & Martin

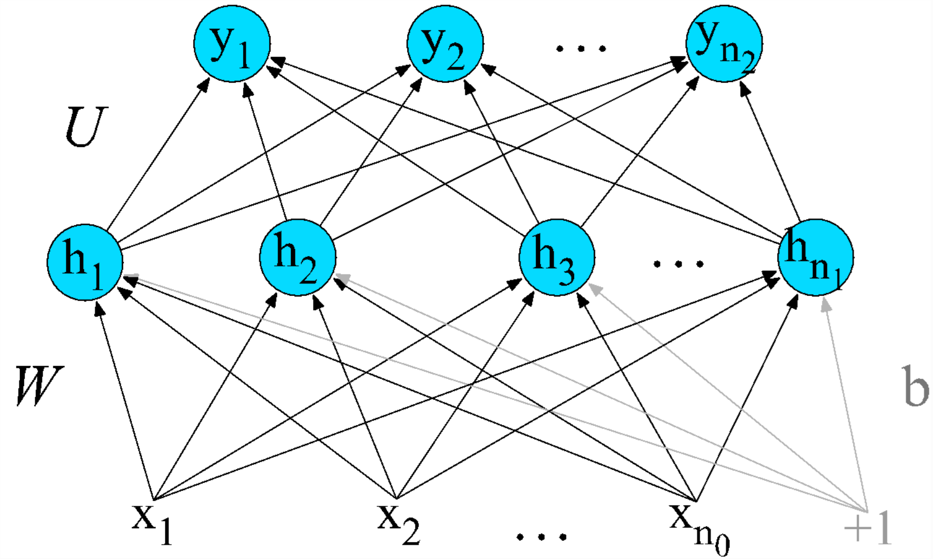
Feedforward neural networks

Adding multiple units to a neural network increases its power to learn patterns in data. **Feedforward Neural Nets (FFNNs or MLPs)**

Slide adapted from David Mortensen

Feedforward Neural Networks

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



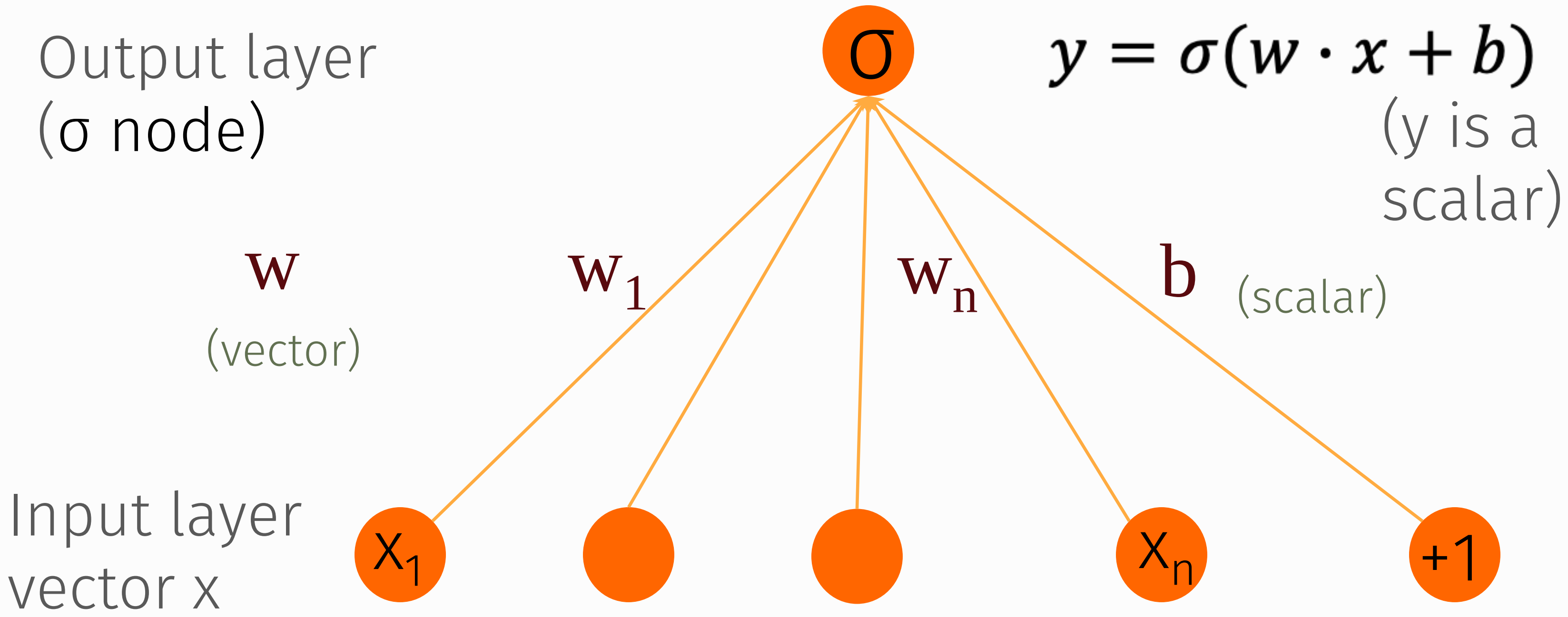
Slide adapted from Jurafsky & Martin

The simplest FFNN is just binary logistic regression
(INPUT LAYER = feature vector)

Slide adapted from David Mortensen

Binary Logistic Regression as a 1-layer Network

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

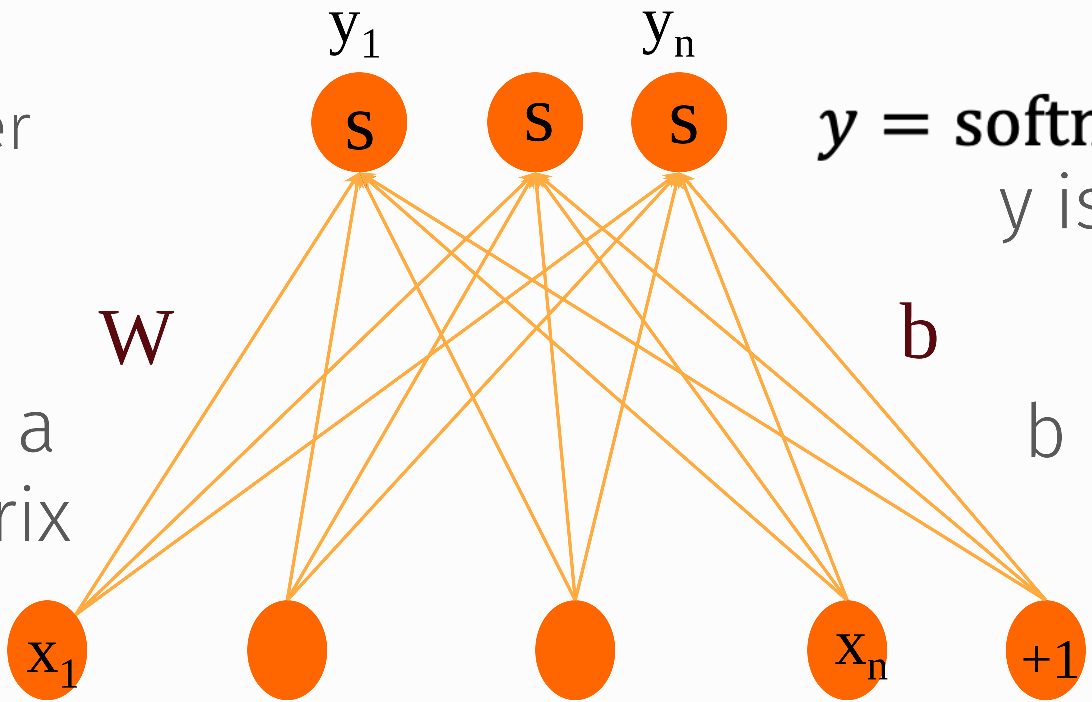


Slide adapted from Jurafsky & Martin

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

Slide adapted from Jurafsky & Martin

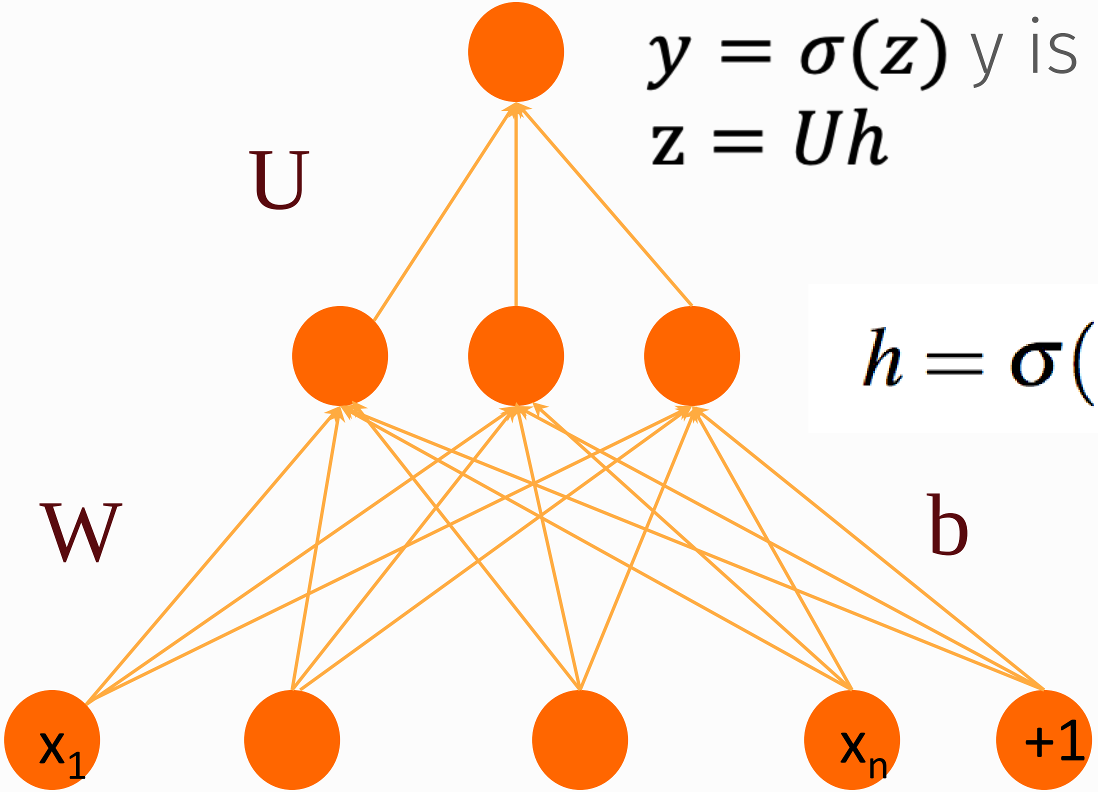
The real power comes when multiple layers are added

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

Slide adapted from Jurafsky & Martin

Two-Layer Network with scalar output

Output layer
(σ node)

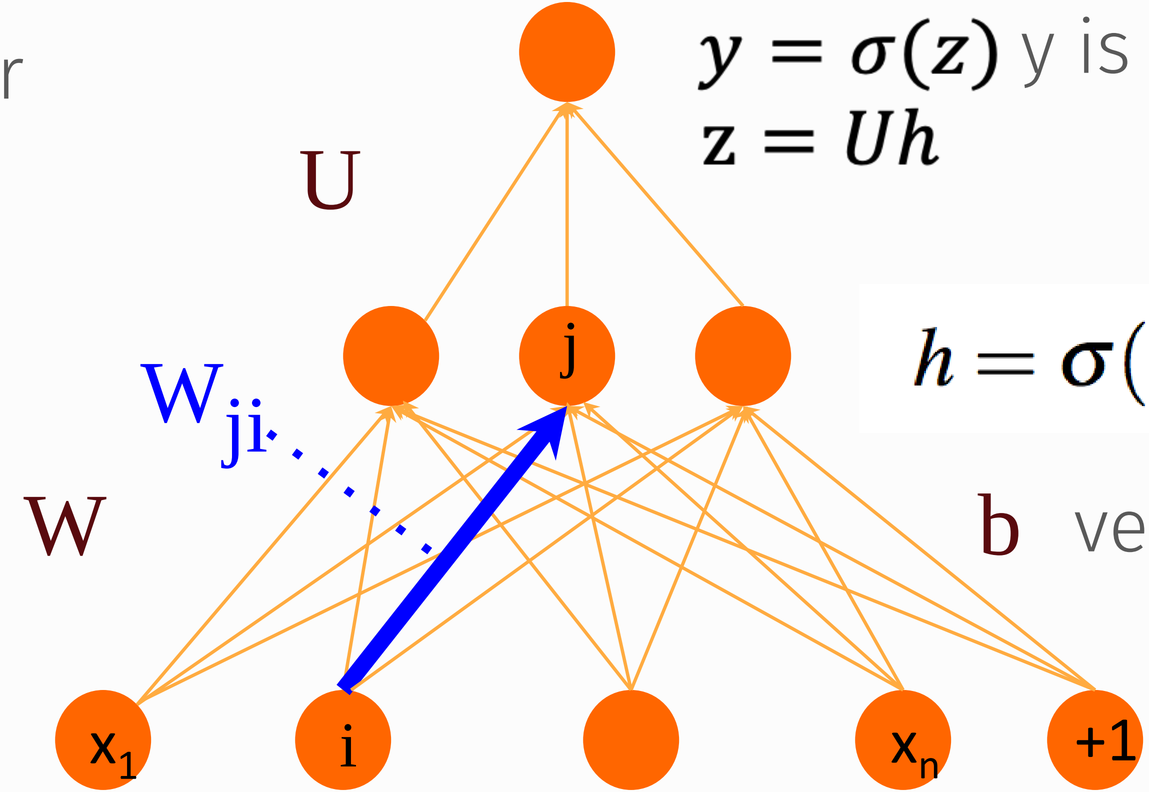
$y = \sigma(z)$ y is a scalar

$z = Uh$

hidden units
(σ node)

$h = \sigma(Wx + b)$

Input layer
(vector)



b vector

Slide adapted from Jurafsky & Martin

Two-Layer Network with scalar output

Output layer
(σ node)

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

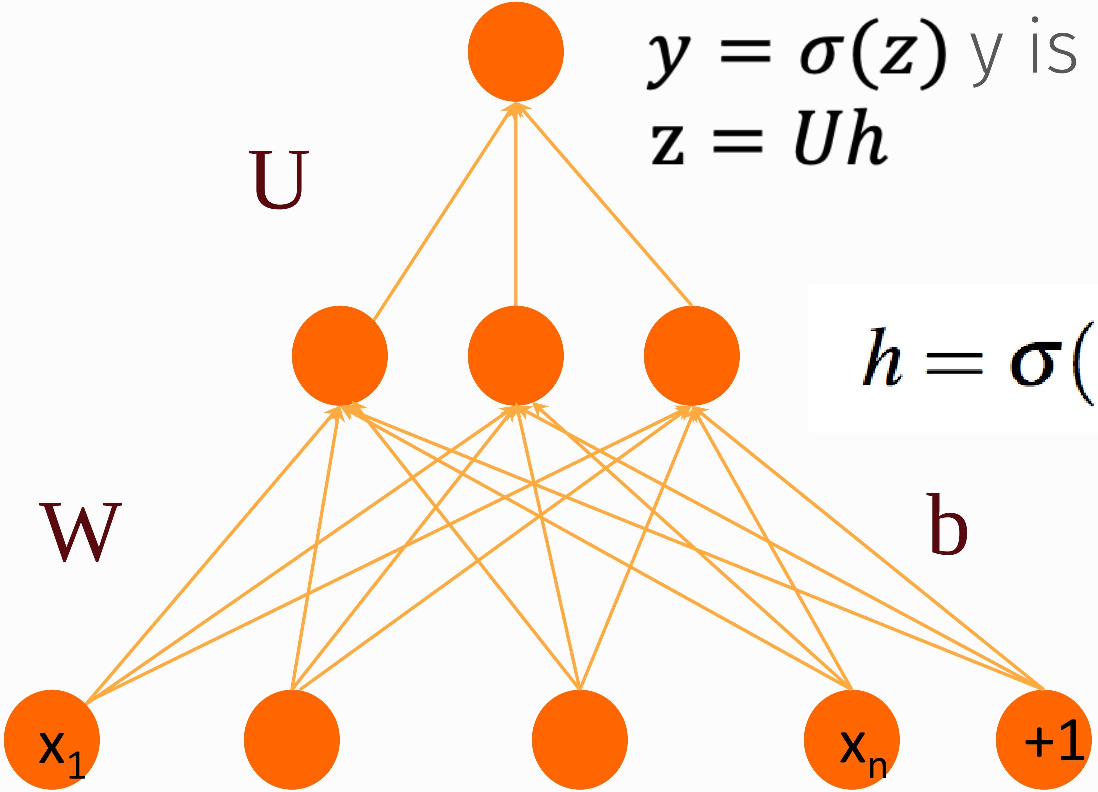
$z = Uh$

hidden units
(σ node)

$h = \sigma(Wx + b)$

Could be
ReLU
Or tanh

Input layer
(vector)



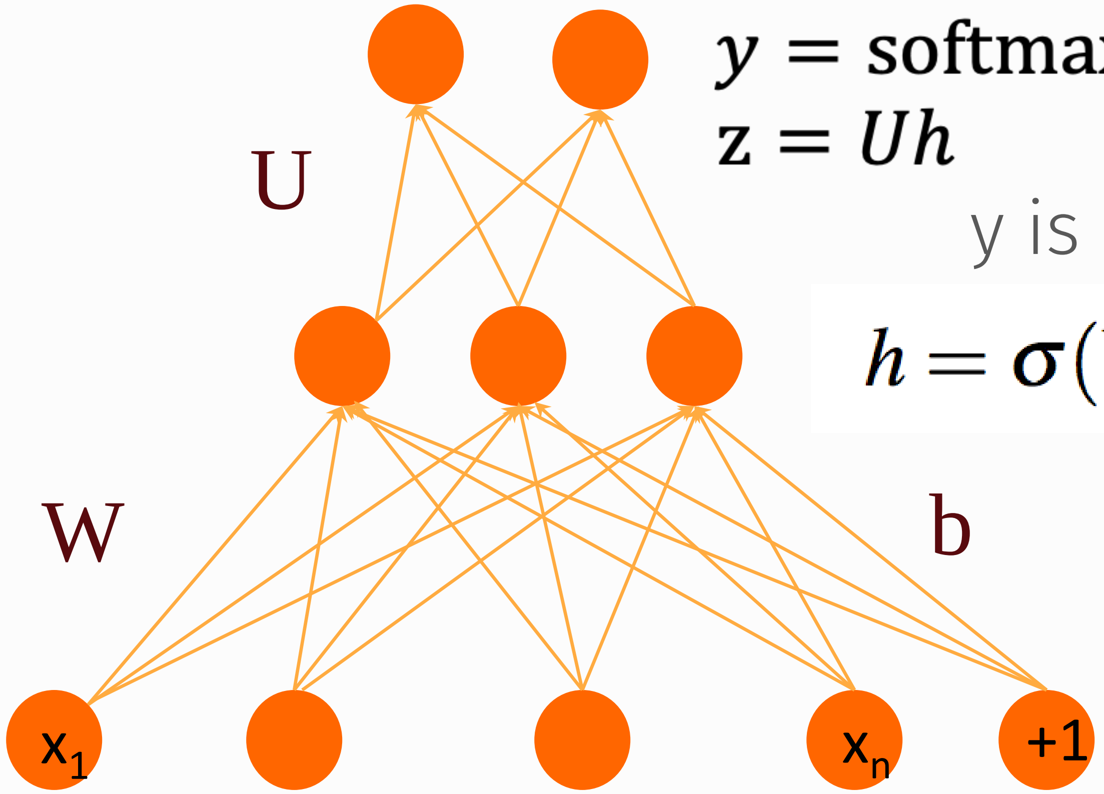
Slide adapted from Jurafsky & Martin

Two-Layer Network with softmax output

Output layer
(σ node)

hidden units
(σ node)

Input layer
(vector)



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

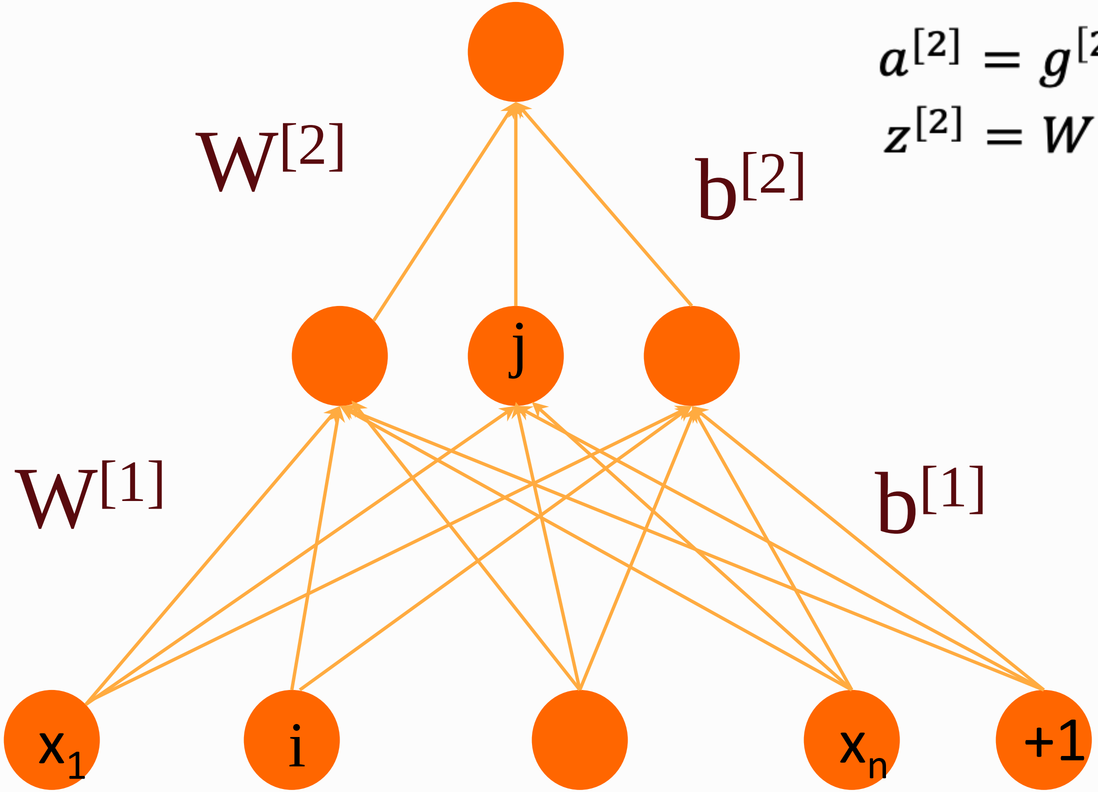
y is a vector

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

Could be
ReLU
Or tanh

Slide adapted from Jurafsky & Martin

Multi-layer Notation



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

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

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

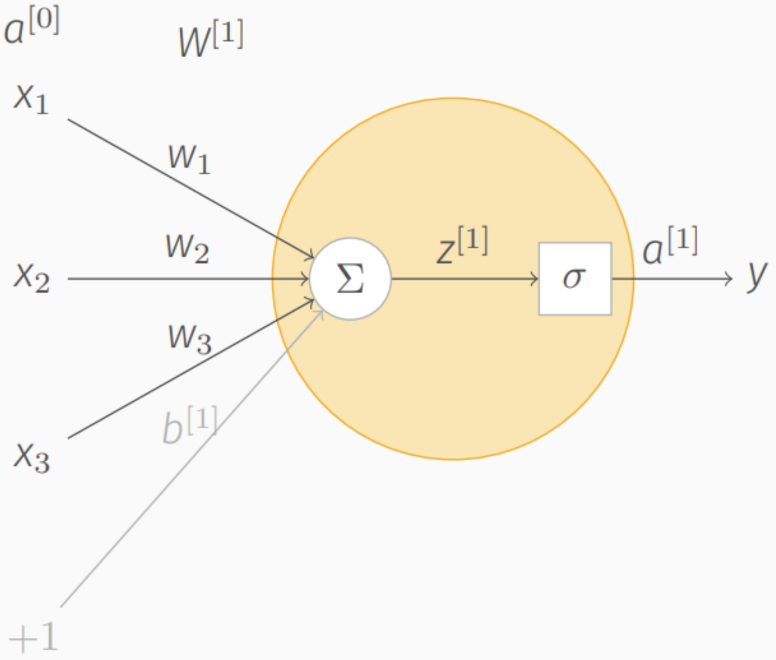
$$a^{[1]} = g^{[1]}(z^{[1]})$$ ReLU

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

$$a^{[0]}$$

Slide adapted from Jurafsky & Martin

A Forward Pass in Terms of Multi-Layer Notation

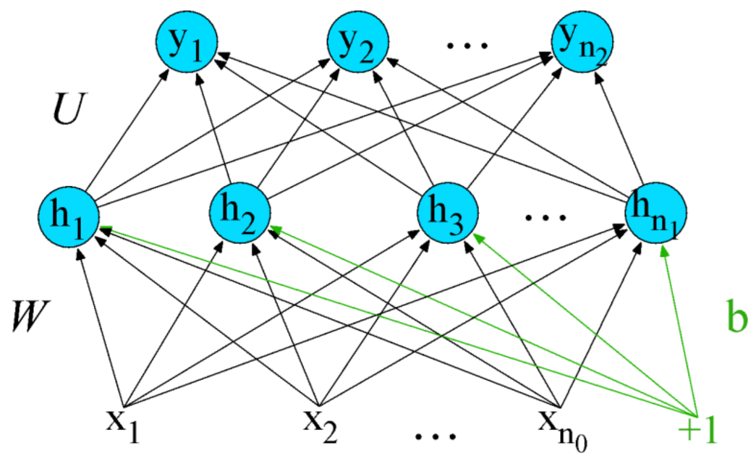


for each $i \in 1..n$ **do**
  $z^{[i]} \leftarrow W^{[i]} a^{[i-1]} + b^{[i]}$
  $a^{[i]} \leftarrow g^{[i]}(z^{[i]})$
end for
$\hat{y} \leftarrow a^{[n]}$

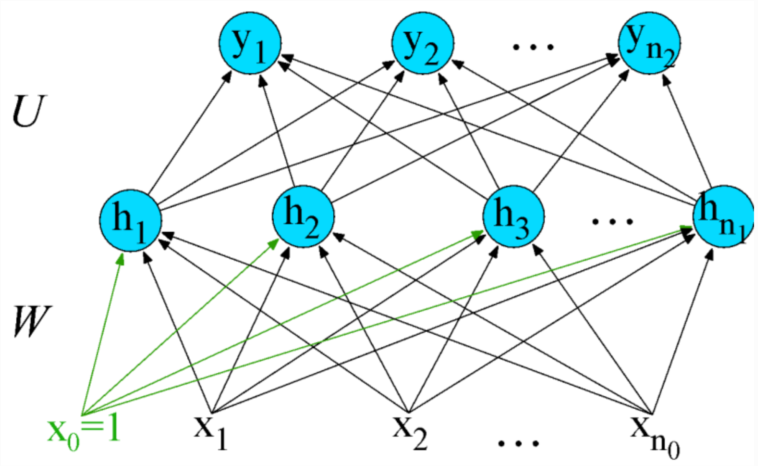
Slide credit: David Mortensen

Replacing the bias unit

Instead of:



We'll do this:



Slide adapted from Jurafsky & Martin

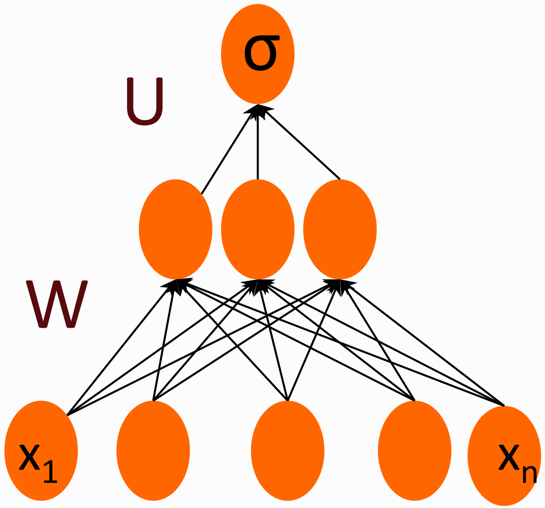
Feedforward neural nets as classifiers

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



Slide adapted from Jurafsky & Martin

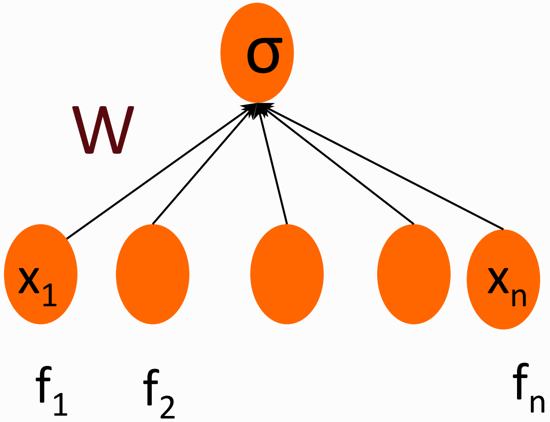
Sentiment Features

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)

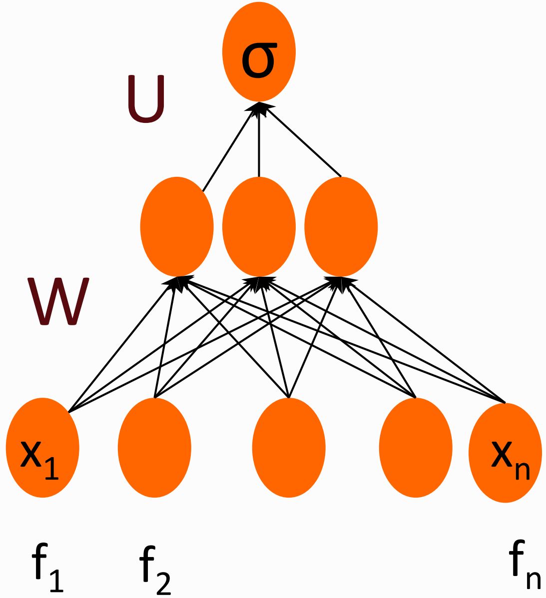
Slide adapted from Jurafsky & Martin

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.

Slide adapted from Jurafsky & Martin

Coding activity

Notebook: feedforward neural network

- Click on this nbgitpuller link
  - Or find the link on the course website
- Open **session12_ffnn.ipynb**

Questions?