Neural Networks II

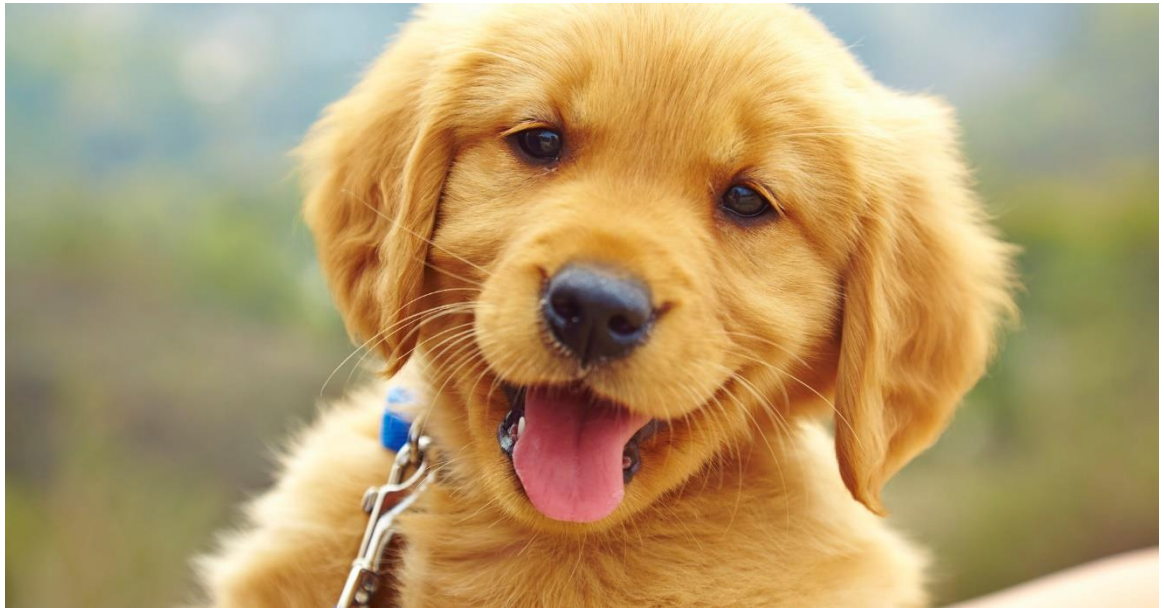
Chen Gao
Virginia Tech

ECE-5424G / CS-5824

Spring 2019

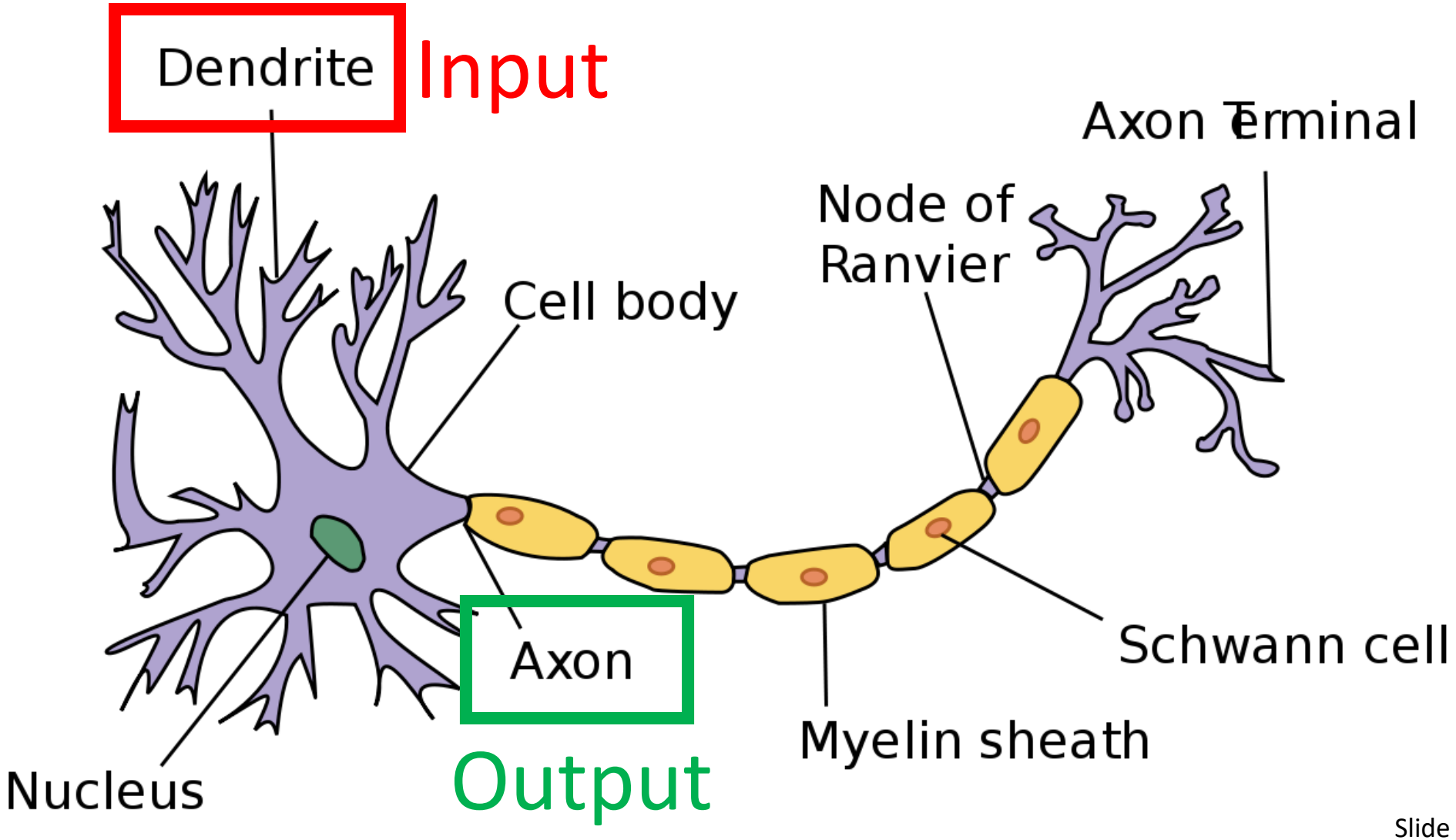
Neural Networks

- Origins: Algorithms that try to mimic the brain.



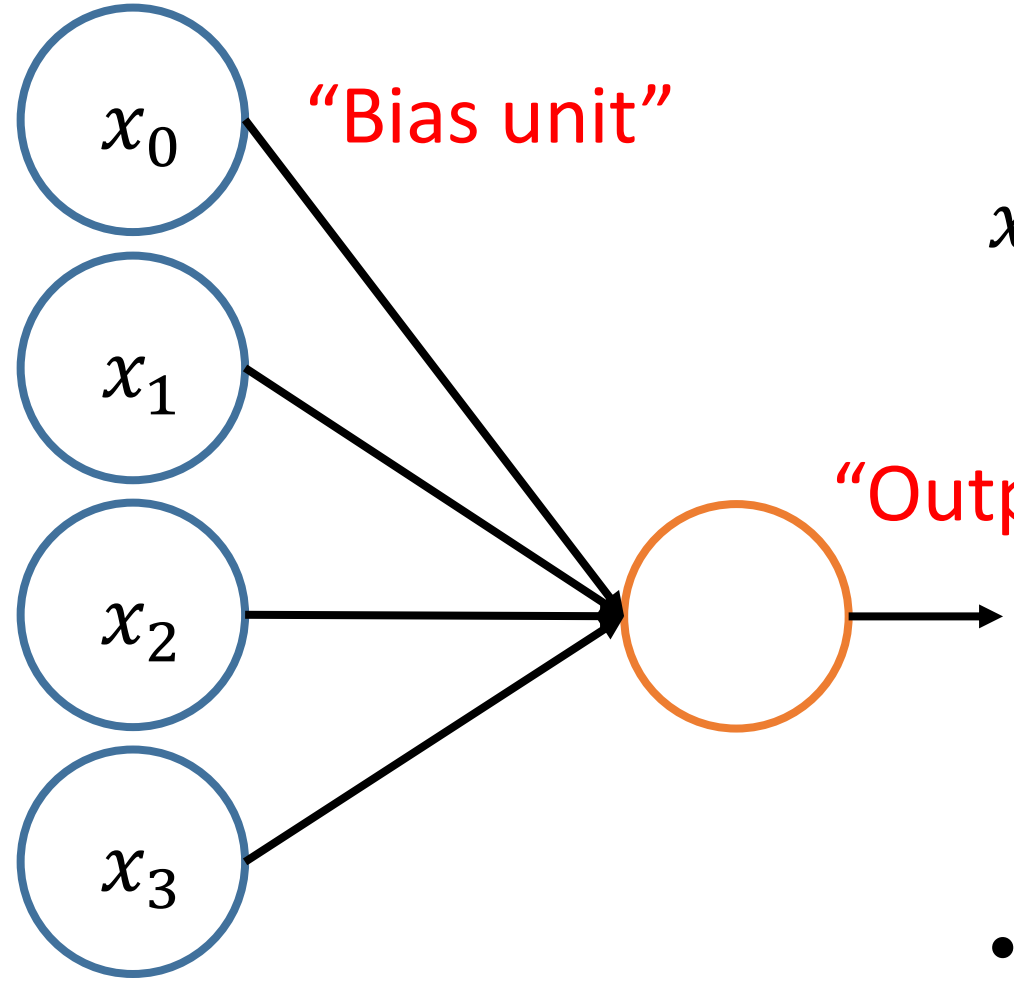
What is this?

A single neuron in the brain



Slide credit: Andrew Ng

An artificial neuron: Logistic unit



$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

"Weights"
"Parameters"

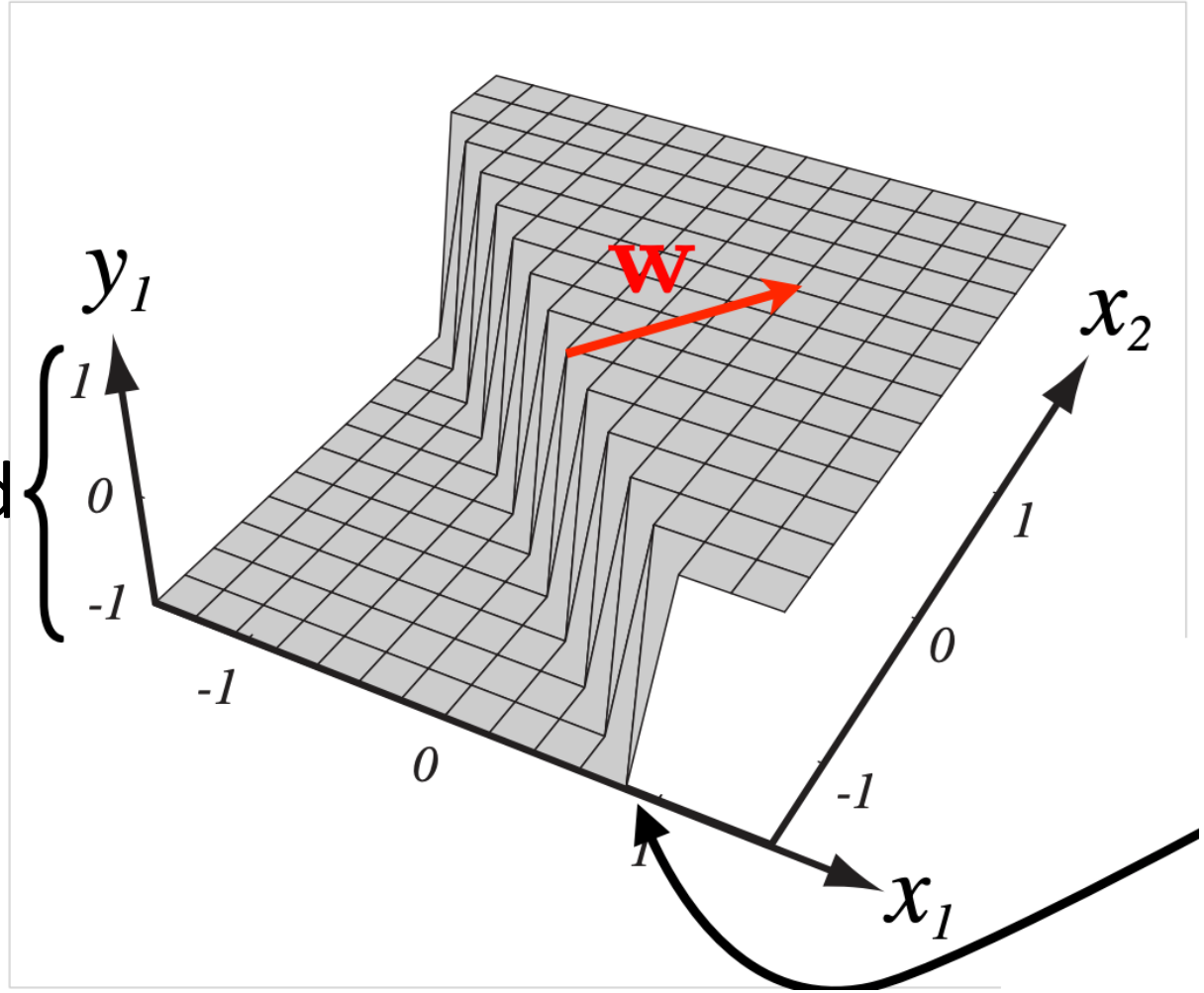
$$h_\theta(x) = \frac{1}{1 + e^{-\theta^\top x}}$$

- Sigmoid (logistic) activation function

Slide credit: Andrew Ng

Visualization of weights, bias, activation function

range determined by g(.)

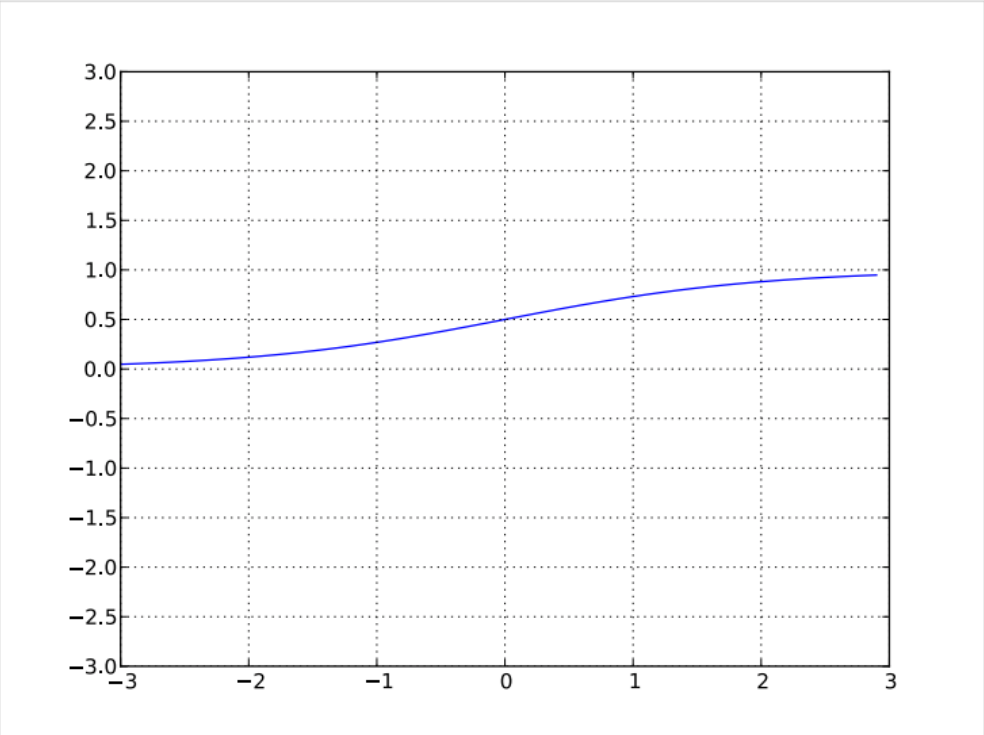


bias b only change the position of the hyperplane

Slide credit: Hugo Larochelle

Activation - sigmoid

- Squashes the neuron's pre-activation between 0 and 1
- Always positive
- Bounded
- Strictly increasing

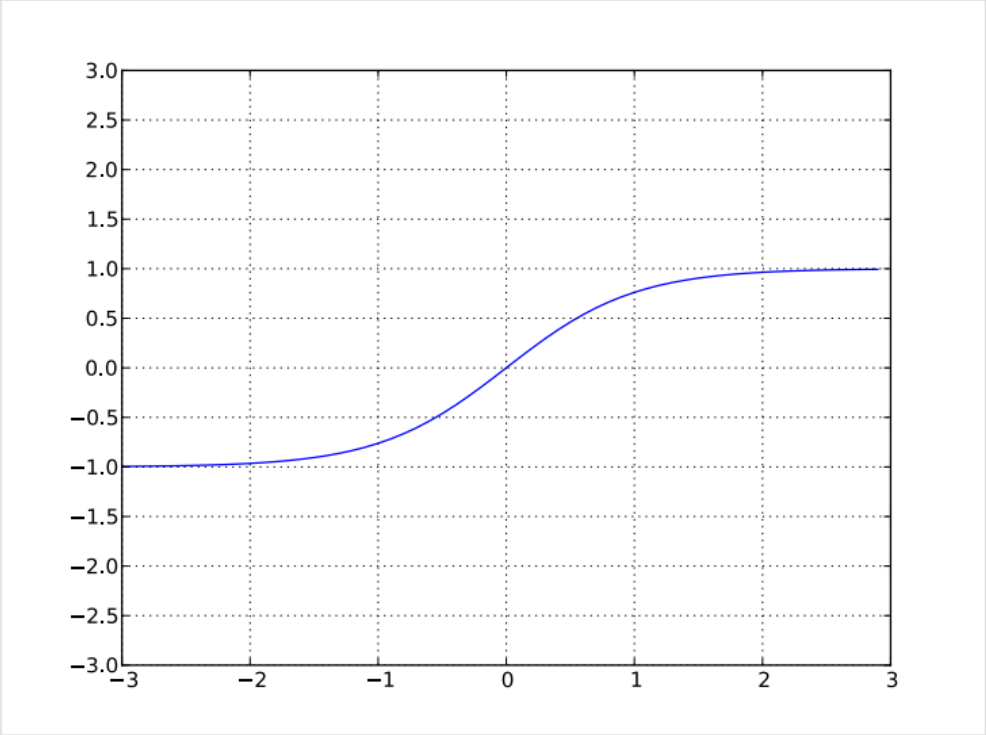


$$g(x) = \frac{1}{1 + e^{-x}}$$

Slide credit: Hugo Larochelle

Activation - hyperbolic tangent (tanh)

- Squashes the neuron's pre-activation between -1 and 1
- Can be positive or negative
- Bounded
- Strictly increasing

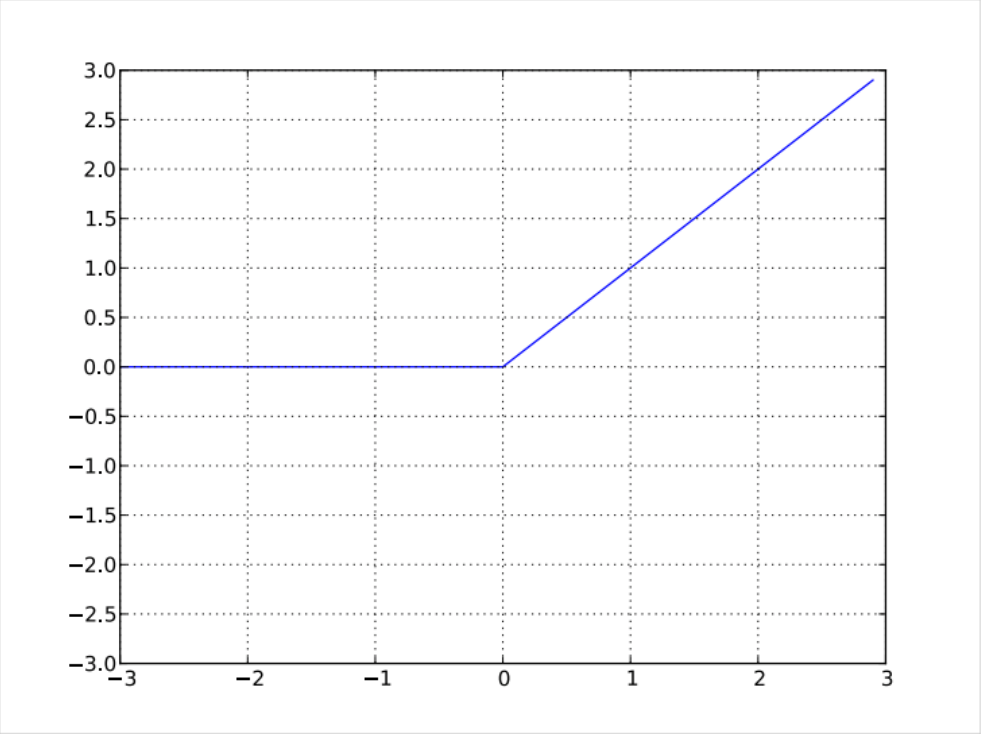


$$g(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Slide credit: Hugo Larochelle

Activation - rectified linear(relu)

- Bounded below by 0
- always non-negative
- Not upper bounded
- Tends to give neurons with sparse activities



$$g(x) = \mathrm{relu}(x) = \max(0, x)$$

Slide credit: Hugo Larochelle

Activation - softmax

- For multi-class classification:
  - we need multiple outputs (1 output per class)
  - we would like to estimate the conditional probability $p(y = c \mid x)$

- We use the softmax activation function at the output:

$$g(x) = \text{softmax}(x) = \left[\frac{e^{x_1}}{\sum_c e^{x_c}} \quad \dots \quad \frac{e^{x_c}}{\sum_c e^{x_c}} \right]$$

Slide credit: Hugo Larochelle

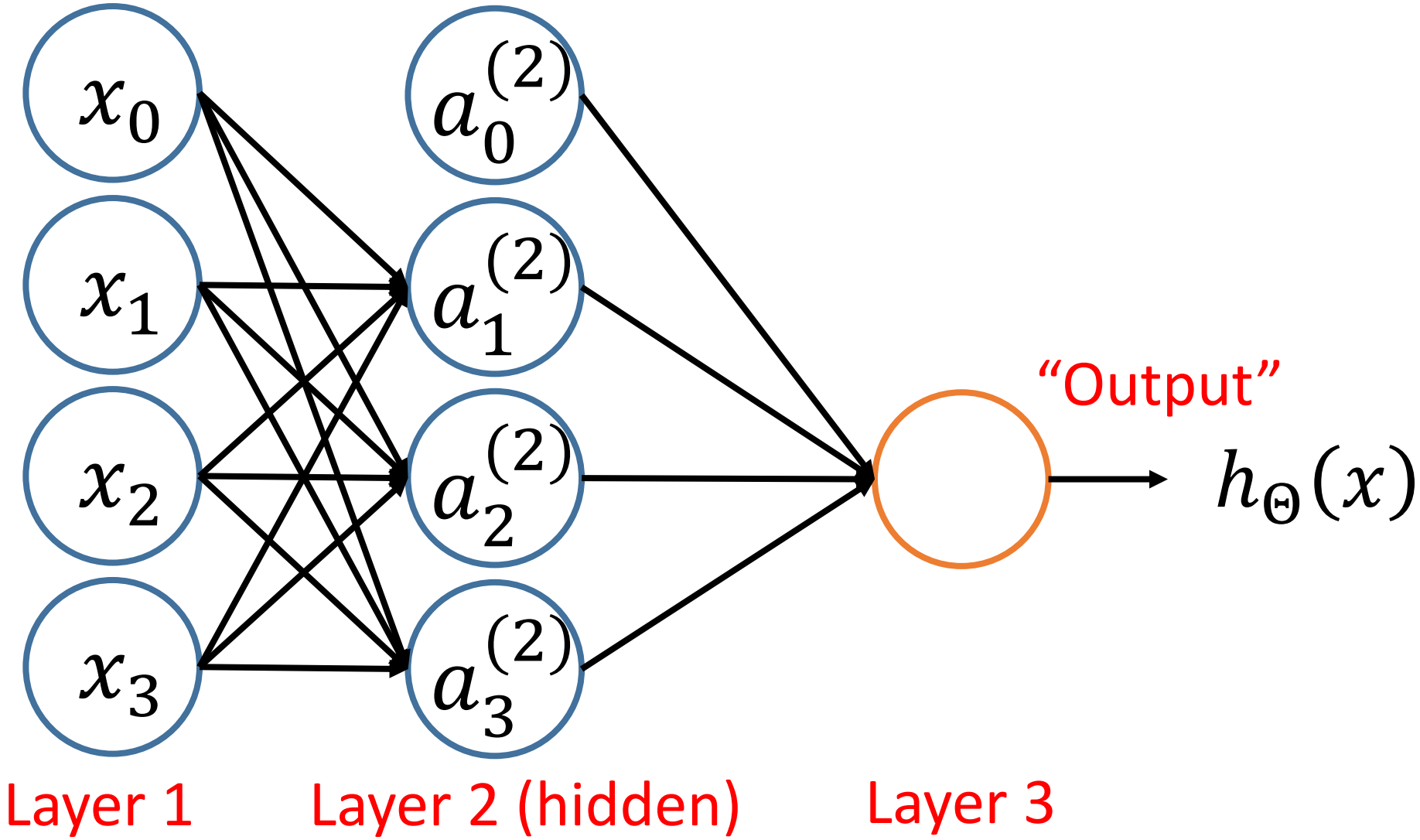
Universal approximation theorem

‘‘a single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units’’

Hornik, 1991

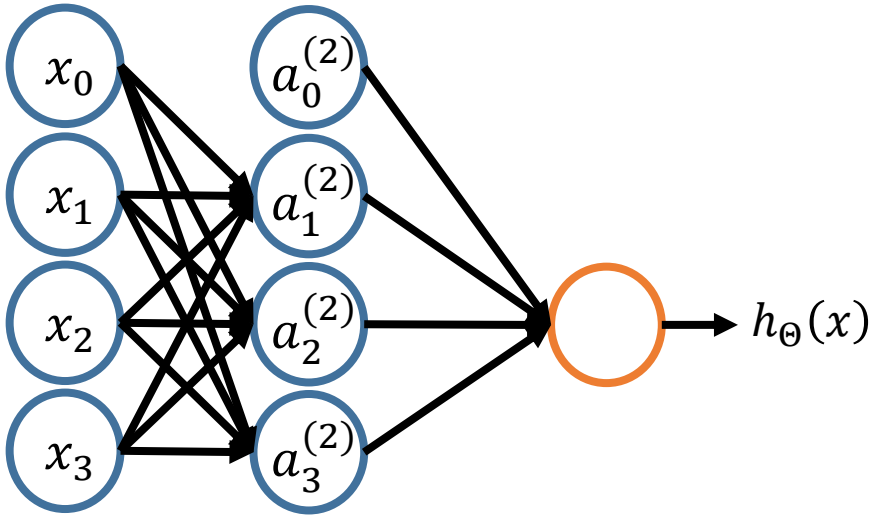
Slide credit: Hugo Larochelle

Neural network – Multilayer



Slide credit: Andrew Ng

Neural network



$a_i^{(j)}$ = "activation" of unit i in layer j

$\Theta^{(j)}$ = matrix of weights controlling function mapping from layer j to layer $j+1$

s_j unit in layer j

s_{j+1} units in layer $j+1$

$$a_1^{(2)} = g\left(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right)$$

$$a_2^{(2)} = g\left(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right)$$

$$a_3^{(2)} = g\left(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3\right)$$

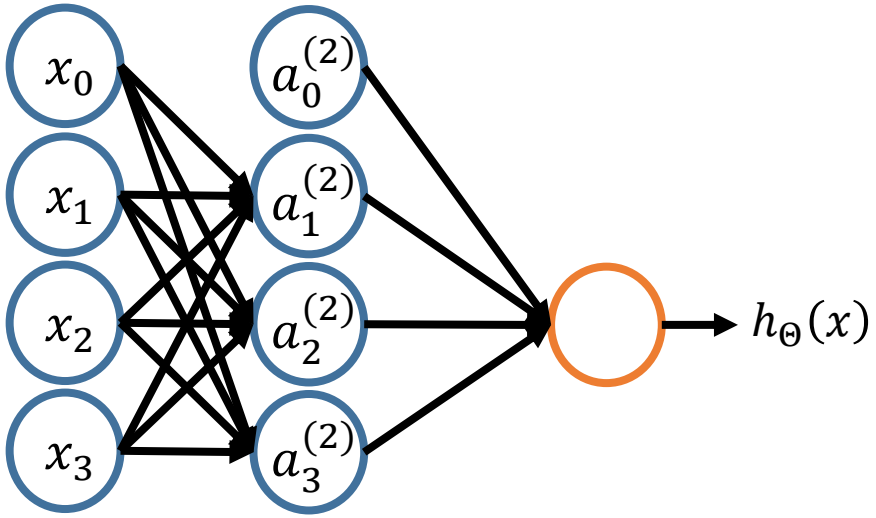
$$h_\Theta(x) = g\left(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(1)}a_1^{(2)} + \Theta_{12}^{(1)}a_2^{(2)} + \Theta_{13}^{(1)}a_3^{(2)}\right)$$

Size of $\Theta^{(j)}$?

$s_{j+1} \times (s_j + 1)$

Slide credit: Andrew Ng

Neural network



$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

"Pre-activation"

$$z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

Why do we need g(.)?

$$a_1^{(2)} = g\left(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3\right) = g(z_1^{(2)})$$

$$a_2^{(2)} = g\left(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3\right) = g(z_2^{(2)})$$

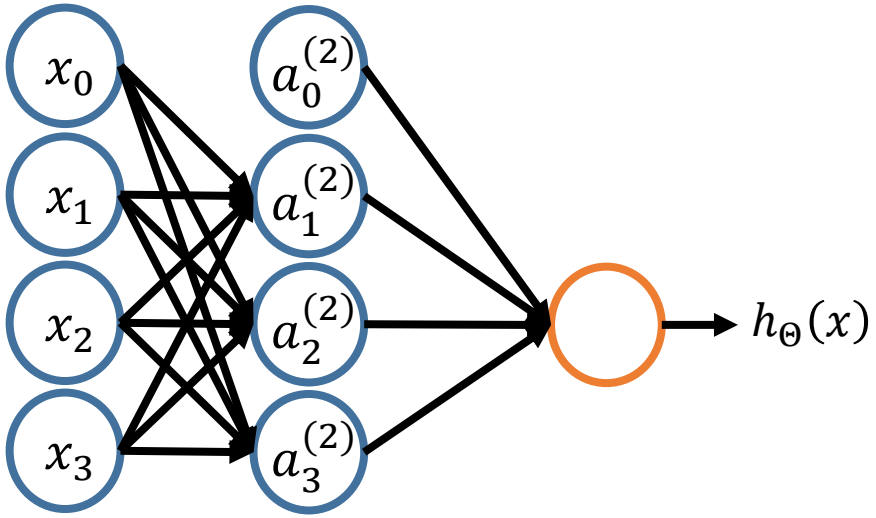
$$a_3^{(2)} = g\left(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3\right) = g(z_3^{(2)})$$

$$h_\Theta(x) = g\left(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(1)} a_1^{(2)} + \Theta_{12}^{(1)} a_2^{(2)} + \Theta_{13}^{(1)} a_3^{(2)}\right) = g(z^{(3)})$$

Slide credit: Andrew Ng

Neural network

"Pre-activation"



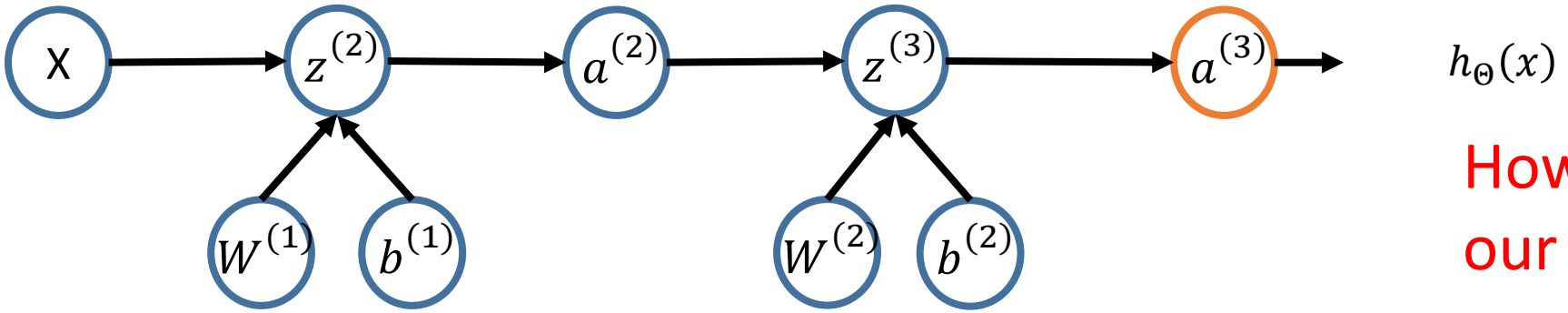
$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

$$a_1^{(2)} = g(z_1^{(2)})$$
$$a_2^{(2)} = g(z_2^{(2)})$$
$$a_3^{(2)} = g(z_3^{(2)})$$
$$h_\Theta(x) = g(z^{(3)})$$

$$z^{(2)} = \Theta^{(1)} x = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g(z^{(2)})$$
$$\text{Add } a_0^{(2)} = 1$$
$$z^{(3)} = \Theta^{(2)} a^{(2)}$$
$$h_\Theta(x) = a^{(3)} = g(z^{(3)})$$

Slide credit: Andrew Ng

Flow graph - Forward propagation



$h_\Theta(x)$

How do we evaluate our prediction?

$$z^{(2)} = \Theta^{(1)}x = \Theta^{(1)}a^{(1)}$$
$$a^{(2)} = g(z^{(2)})$$
$$\text{Add } a_0^{(2)} = 1$$
$$z^{(3)} = \Theta^{(2)}a^{(2)}$$
$$h_\Theta(x) = a^{(3)} = g(z^{(3)})$$

Cost function

Logistic regression:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Neural network:

$$h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th} \text{ output}$$

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)}) \log(1-(h_\Theta(x^{(i)}))_k)\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

Slide credit: Andrew Ng

Gradient computation

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)\right]$$
$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

$$\min_\Theta J(\Theta)$$

Need to compute:

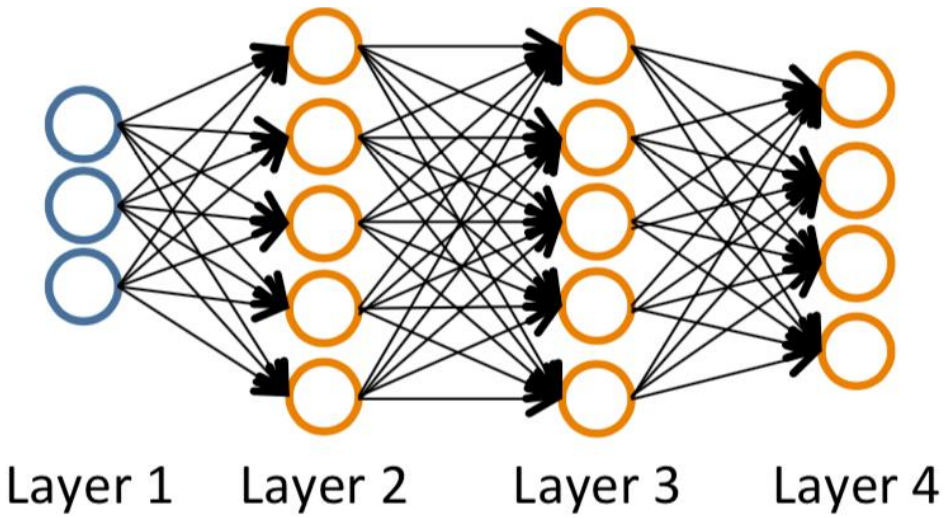
$$\frac{\partial}{\partial\Theta_{ij}^{(l)}} J(\Theta)$$

Slide credit: Andrew Ng

Gradient computation

Given one training example (x, y)

$$a^{(1)} = x$$
$$z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g(z^{(2)}) \text{ (add } a_0^{(2)})$$
$$z^{(3)} = \Theta^{(2)} a^{(2)}$$
$$a^{(3)} = g(z^{(3)}) \text{ (add } a_0^{(3)})$$
$$z^{(4)} = \Theta^{(3)} a^{(3)}$$
$$a^{(4)} = g\left(z^{(4)}\right) = h_\Theta(x)$$

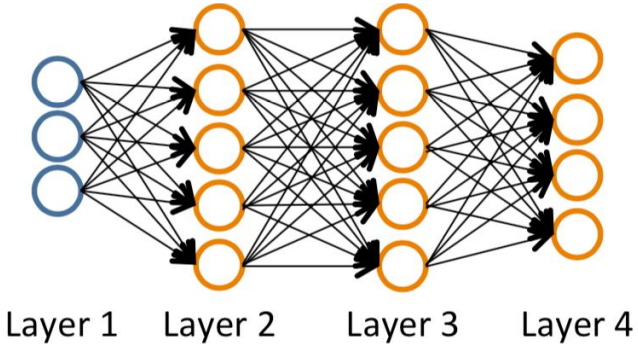


Slide credit: Andrew Ng

Gradient computation: Backpropagation

Intuition: $\delta_j^{(l)}$ = "error" of node j in layer l

For each output unit (layer L = 4)



$$\delta^{(4)} = a^{(4)} - y$$

$$\delta^{(3)} = \delta^{(4)} \frac{\partial \delta^{(4)}}{\partial z^{(3)}} = \delta^{(4)} \frac{\partial \delta^{(4)}}{\partial a^{(4)}} \frac{\partial a^{(4)}}{\partial z^{(4)}} \frac{\partial z^{(4)}}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial z^{(3)}}$$

$$= 1 * \Theta^{(3)^T} \delta^{(4)} .* \; g'\left(z^{(4)}\right) .* \, g'(z^{(3)})$$

$$z^{(3)} = \Theta^{(2)} a^{(2)}$$
$$a^{(3)} = g(z^{(3)})$$
$$z^{(4)} = \Theta^{(3)} a^{(3)}$$
$$a^{(4)} = g\left(z^{(4)}\right)$$

Slide credit: Andrew Ng

Backpropagation algorithm

Training set $\{(x^{(1)}, y^{(1)}) \ \dots (x^{(m)}, y^{(m)})\}$

Set $\Theta^{(1)}$ = 0

For $i = 1$ to m

Set $a^{(1)} = x$

Perform forward propagation to compute $a^{(l)}$ for $l = 2..L$

use $y^{(i)}$ to compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

Compute $\delta^{(L-1)}, \delta^{(L-2)} \dots \delta^{(2)}$

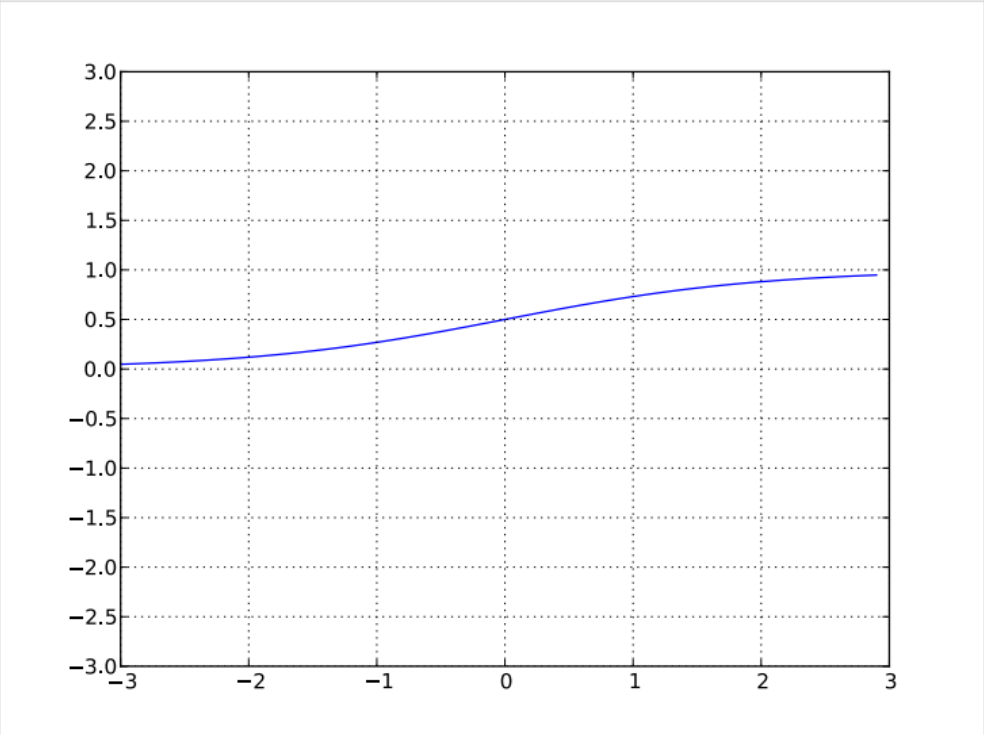
$\Theta^{(l)} = \Theta^{(l)} - \ a^{(l)}\delta^{(l+1)}$

Slide credit: Andrew Ng

Activation - sigmoid

- Partial derivative

$$g'(x) = g(x)\left(1 - g(x)\right)$$



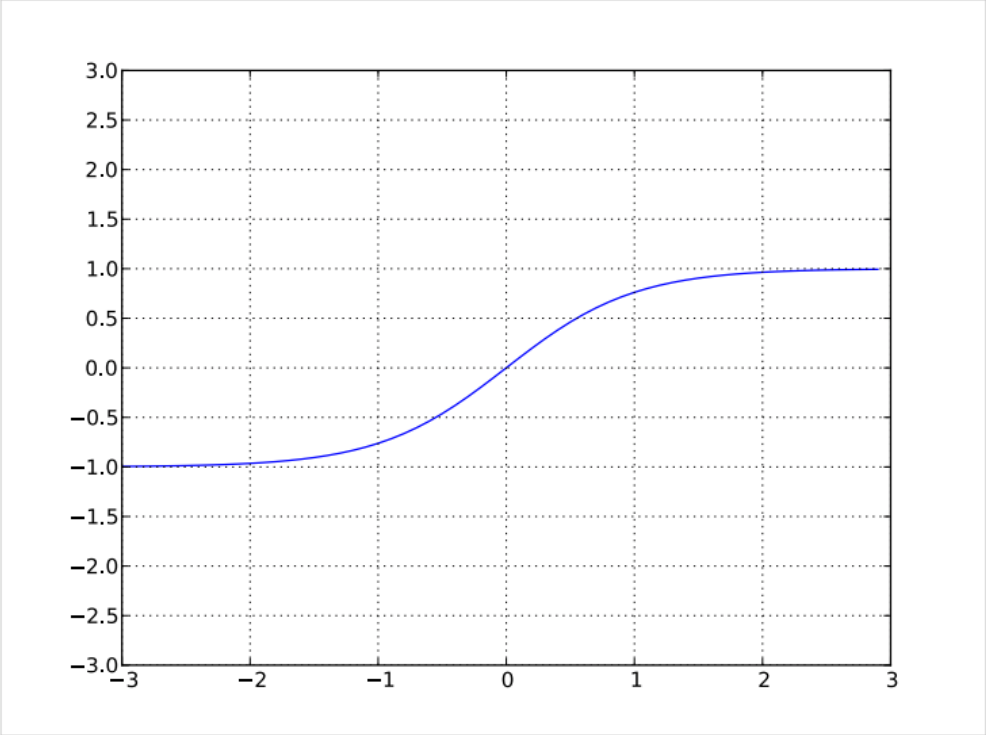
$$g(x) = \frac{1}{1 + e^{-x}}$$

Slide credit: Hugo Larochelle

Activation - hyperbolic tangent (tanh)

- Partial derivative

$$g'(x) = 1 - g(x)^2$$



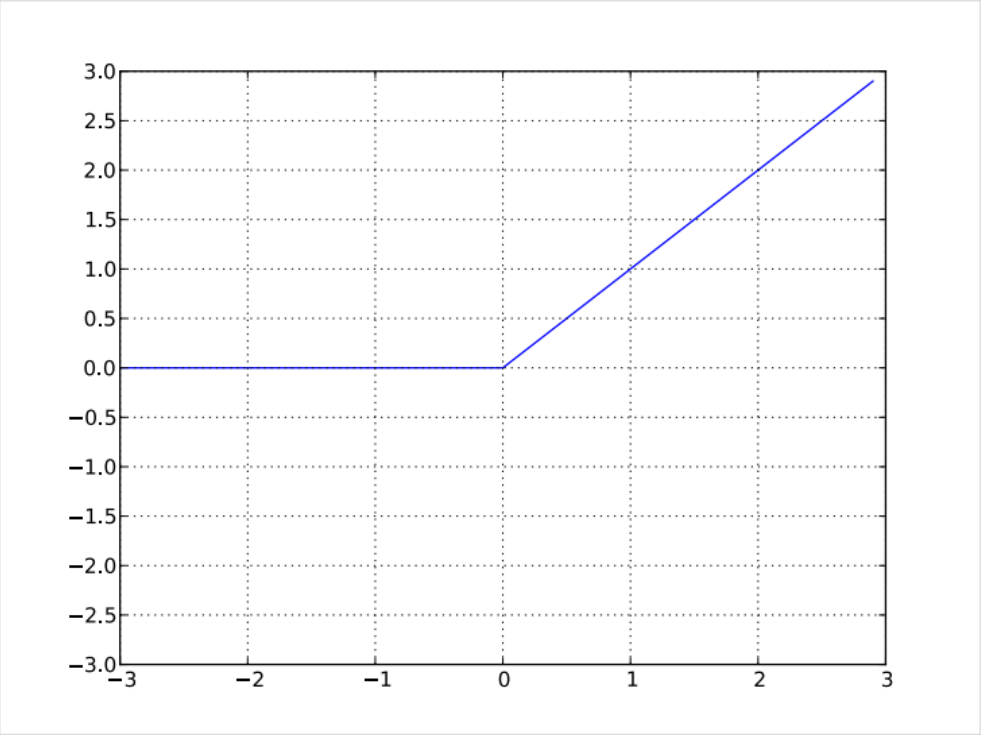
$$g(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Slide credit: Hugo Larochelle

Activation - rectified linear(relu)

- Partial derivative

$$g'(x) = 1_{x>0}$$



$$g(x) = \mathrm{relu}(x) = \max(0, x)$$

Slide credit: Hugo Larochelle

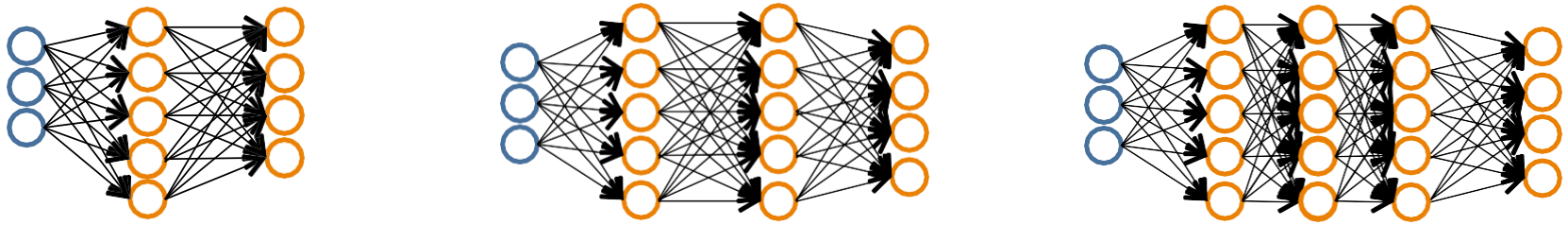
Initialization

- For bias
  - Initialize all to 0
- For weights
  - Can’t initialize all weights to the same value
    - we can show that all hidden units in a layer will always behave the same
    - need to break symmetry
  - Recipe: U[-b, b]
    - the idea is to sample around 0 but break symmetry

Slide credit: Hugo Larochelle

Putting it together

Pick a network architecture



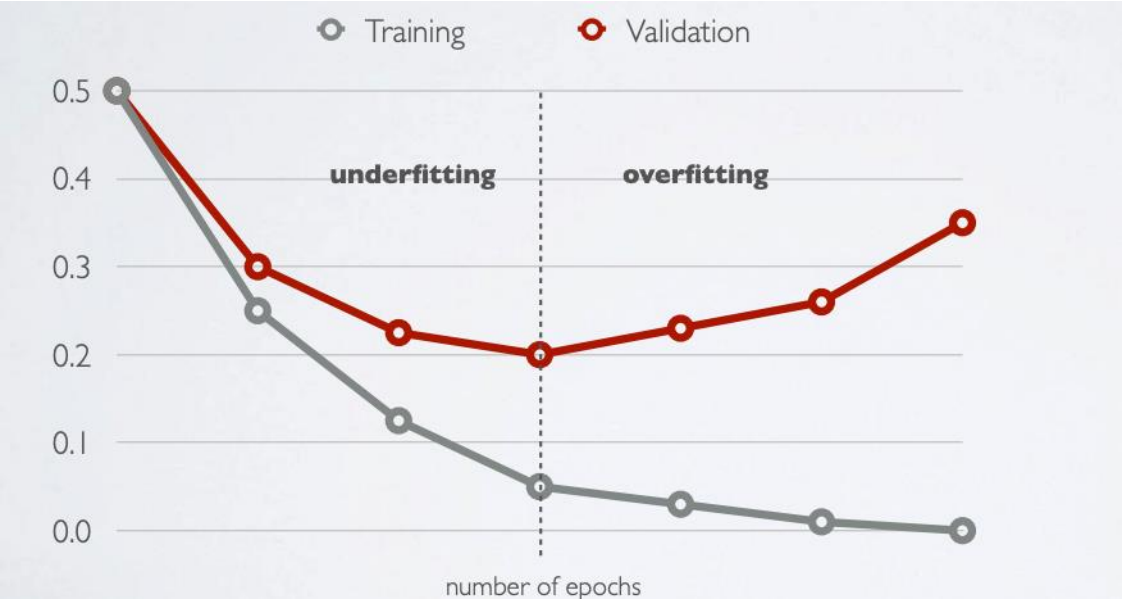
- No. of input units: Dimension of features
- No. output units: Number of classes
- Reasonable default: 1 hidden layer, or if >1 hidden layer, have same no. of hidden units in every layer (usually the more the better)
- Grid search

Slide credit: Hugo Larochelle

Putting it together

Early stopping

- Use a validation set performance to select the best configuration
- To select the number of epochs, stop training when validation set error increases



Slide credit: Hugo Larochelle

Other tricks of the trade

- Normalizing your (real-valued) data
- Decaying the learning rate
  - as we get closer to the optimum, makes sense to take smaller update steps
- mini-batch
  - can give a more accurate estimate of the risk gradient
- Momentum
  - can use an exponential average of previous gradients

Slide credit: Hugo Larochelle

Dropout

- Idea: «cripple» neural network by removing hidden units
  - each hidden unit is set to 0 with probability 0.5
  - hidden units cannot co-adapt to other units
  - hidden units must be more generally useful

Slide credit: Hugo Larochelle