Neural Networks

Win 2019-20

Neural Networks

- Why neural networks?
- Model representation
- Examples and intuitions
- **Multi-class classification**

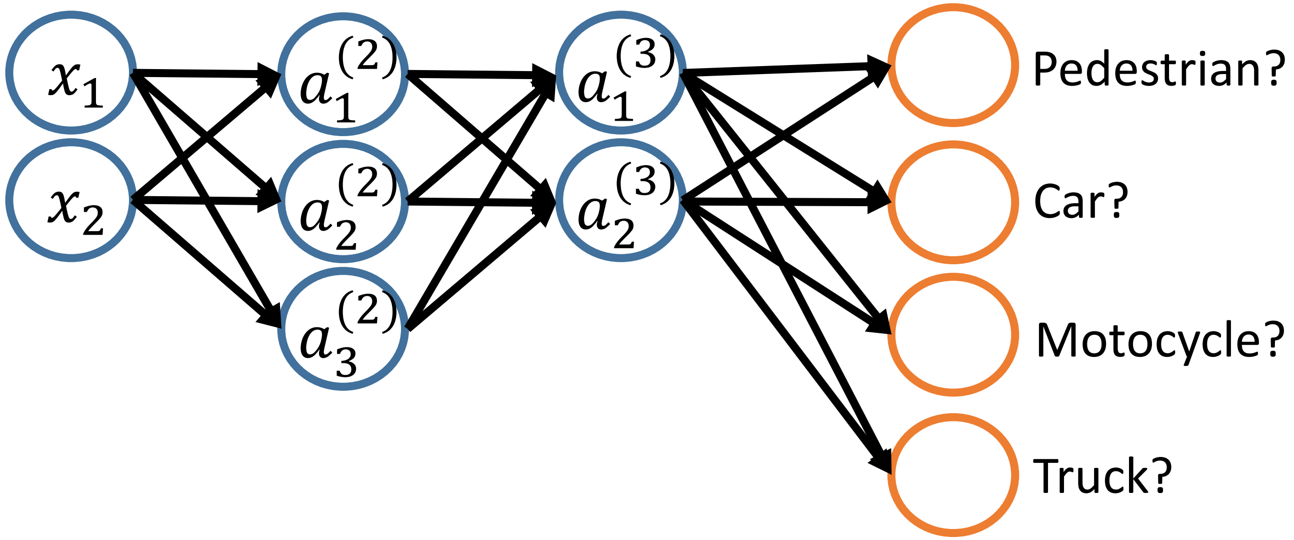
Multiple output units: One-vs-all

Pedestrian

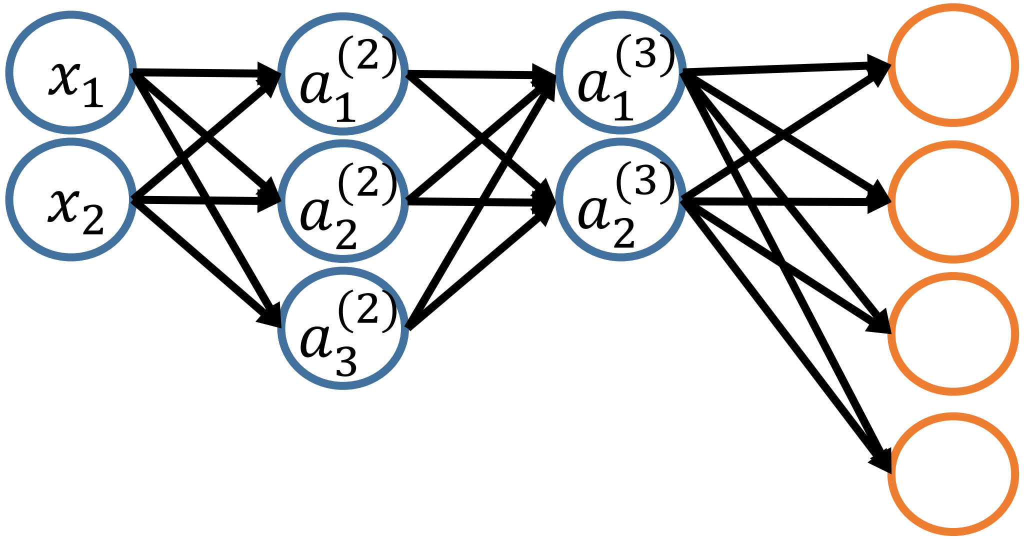
Car

Motorcycle

Truck



Multiple output units: One-vs-all

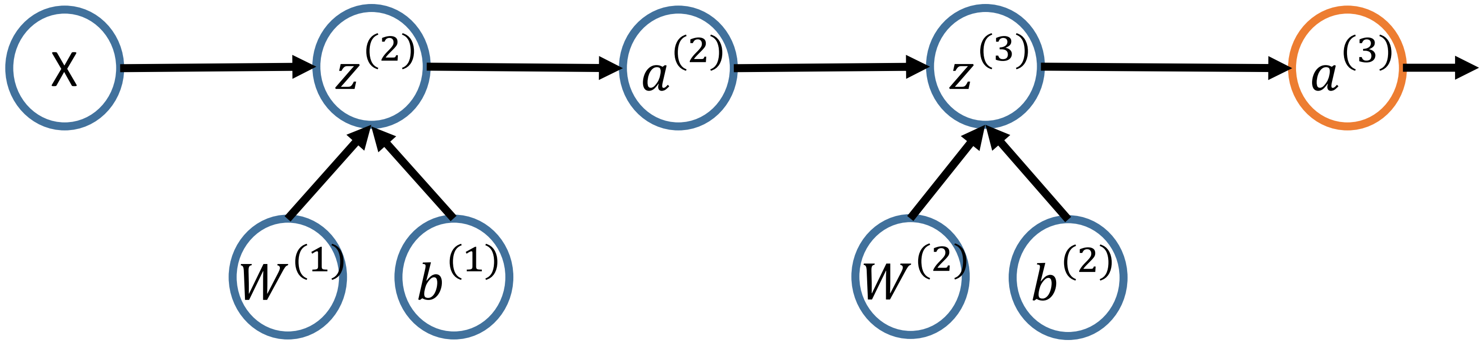


$h_\Theta(x) \in R^4$

Training set : $\left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \cdots \left(x^{(m)}, y^{(m)}\right),$

$y^{(i)}$ one of $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

Flow graph - Forward propagation

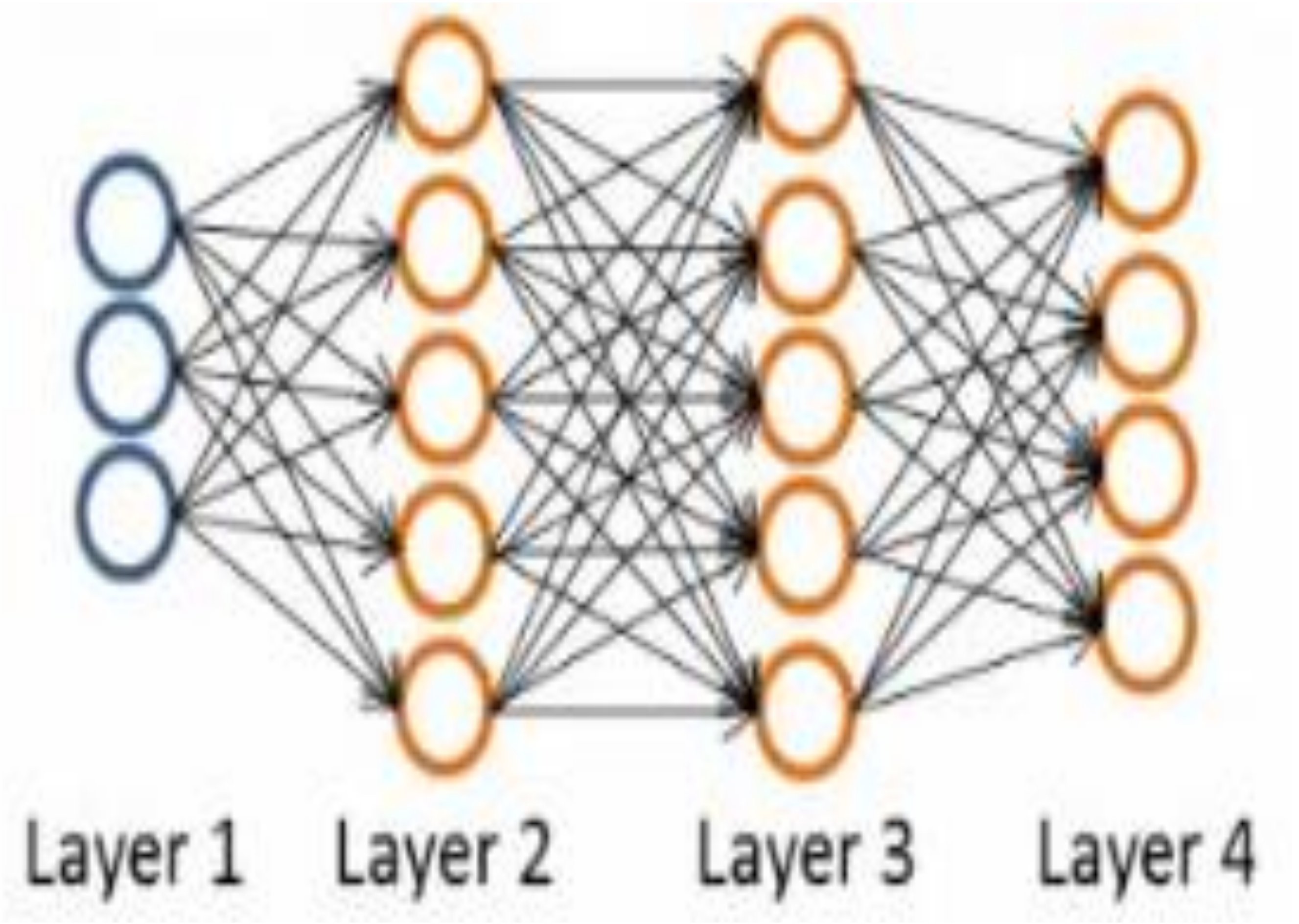


$h_\Theta(x)$

How do we evaluate our prediction?

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

Neural Network cost function



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

Cost function

- Our cost function now outputs a *k* dimensional vector
  - $h_\Theta(x)$ is a k dimensional vector, so $h_\Theta(x)_i$ refers to the ith value in that vector
- Cost function J(Ө) is
  - [-1/m] times a sum of a similar term to which we had for logic regression
  - But now this is also a sum from k = 1 through to K (K is number of output nodes)
    - Summation is a sum over the k output units - i.e. for each of the possible classes
    - So if we had 4 output units then the sum is k = 1 to 4 of the logistic regression over each of the four output units in turn

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:

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

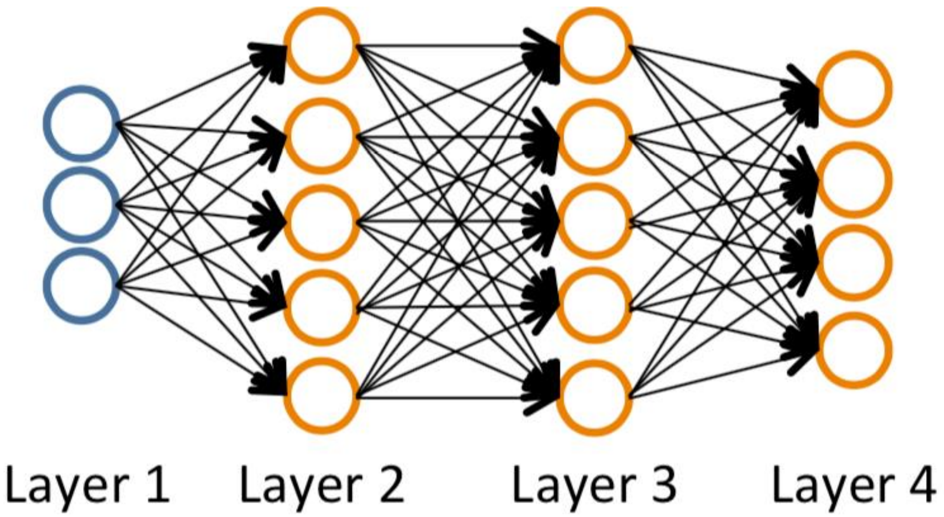
- This is a massive regularization summation term, it's a fairly straightforward triple nested summation.
- This is also called a weight decay term ,as before, the lambda value determines the important of the two halves
- The regularization term is similar to that in logistic regression. So, we have a cost function, but how do we minimize this

Slide credit: Andrew Ng

Gradient computation

Given one training example (x, y)

$$a^{(1)} = x$$
$$z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g(z^{(2)}) (\text{add } \mathrm{a}_0^{(2)})$$
$$z^{(3)} = \Theta^{(2)} a^{(2)}$$
$$a^{(3)} = g(z^{(3)}) (\text{add } \mathrm{a}_0^{(3)})$$
$$z^{(4)} = \Theta^{(3)} a^{(3)}$$
$$a^{(4)} = g\left(z^{(4)}\right) = h_\Theta(x)$$

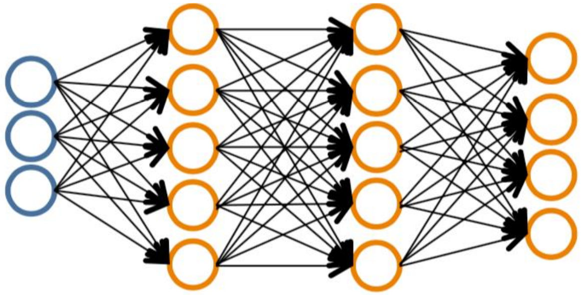


Slide credit: Andrew Ng

Gradient computation: Backpropagation

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

For each output unit (layer L = 4)



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

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

$$\delta^{(3)} = (\Theta^2)^T \delta^3 .*(a^{(3)} .* (1 - a^{(3)})$$

$$\delta^{(2)} = (\Theta^2)^T \delta^2 .*(a^{(2)} .* (1 - a^{(2)})$$

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

Slide credit: Andrew Ng

Why do we do this?

- We do all this to get all the δ terms, and we want the δ terms because through a very complicated derivation you can use δ to get the partial derivative of ϴ with respect to individual parameters (if you ignore regularization, or regularization is 0)

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^l \delta_i^{(l+1)}$$

- By doing back propagation and computing the delta terms you can then compute the **partial derivative terms**
- We need the partial derivatives to minimize the cost function!

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

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

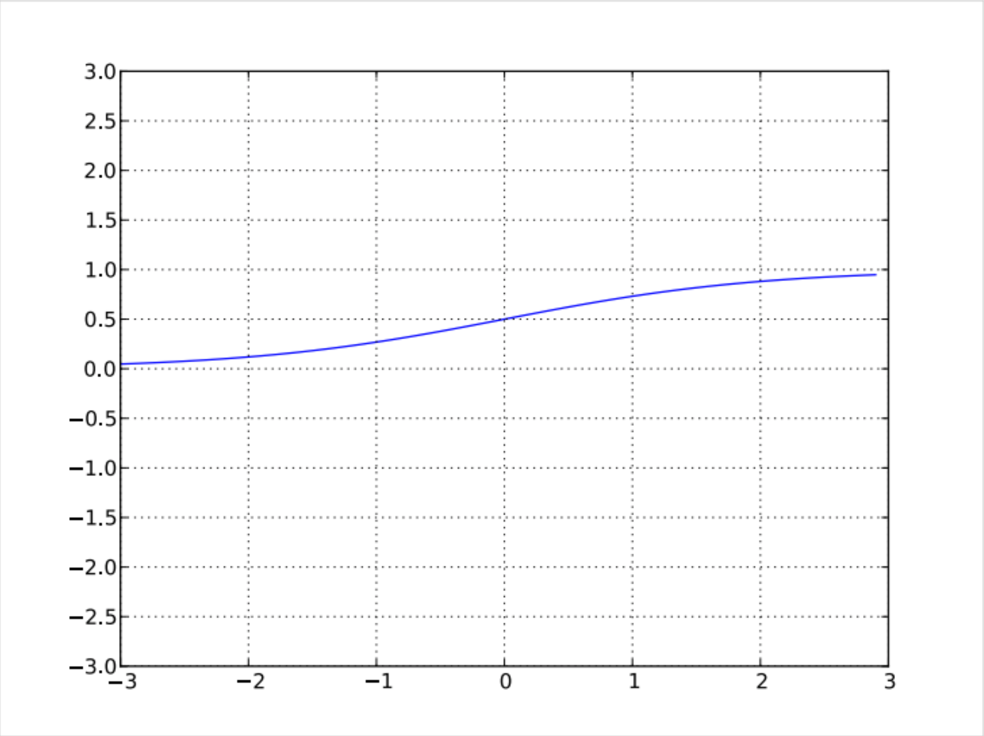
Slide credit: Andrew Ng

Activation - sigmoid

- Partial derivative

$$g'(x) = g(x)\big(1 - g(x)\big)$$

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



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

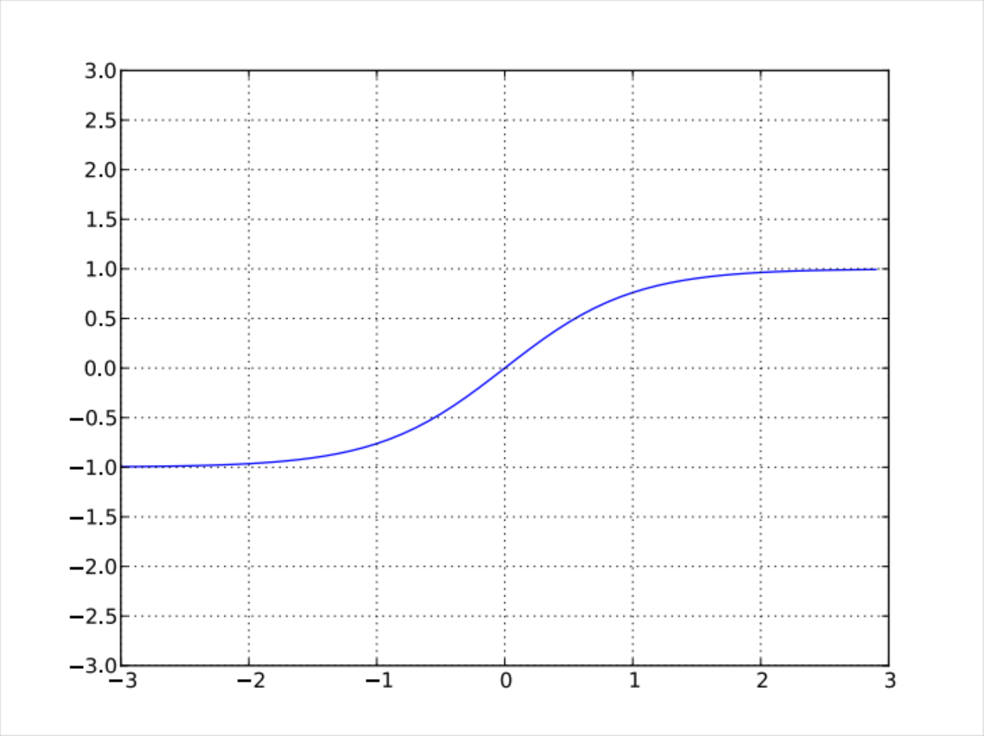
Slide credit: Hugo Larochelle

Activation - hyperbolic tangent (tanh)

- Partial derivative

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

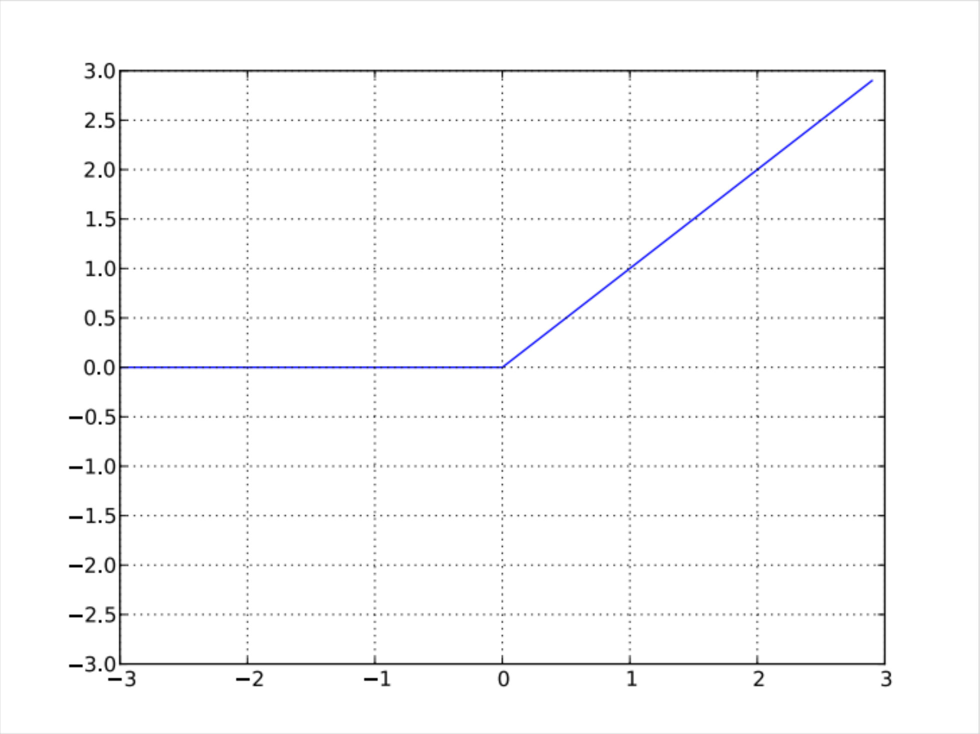
- Squashes the neuron's pre-activation between -1 and 1
- Can be positive or negative
- Bounded
- Strictly increasing



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

Slide credit: Hugo Larochelle

Activation - rectified linear(relu)



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

Slide credit: Hugo Larochelle

Backpropagation algorithm

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

Set $\triangle_{ij}^{(l)} = 0$ (for all l, i, j).

For $i = 1 \text{ to } m$

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

Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$

Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

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

$\triangle_{ij}^{(l)} := \triangle_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

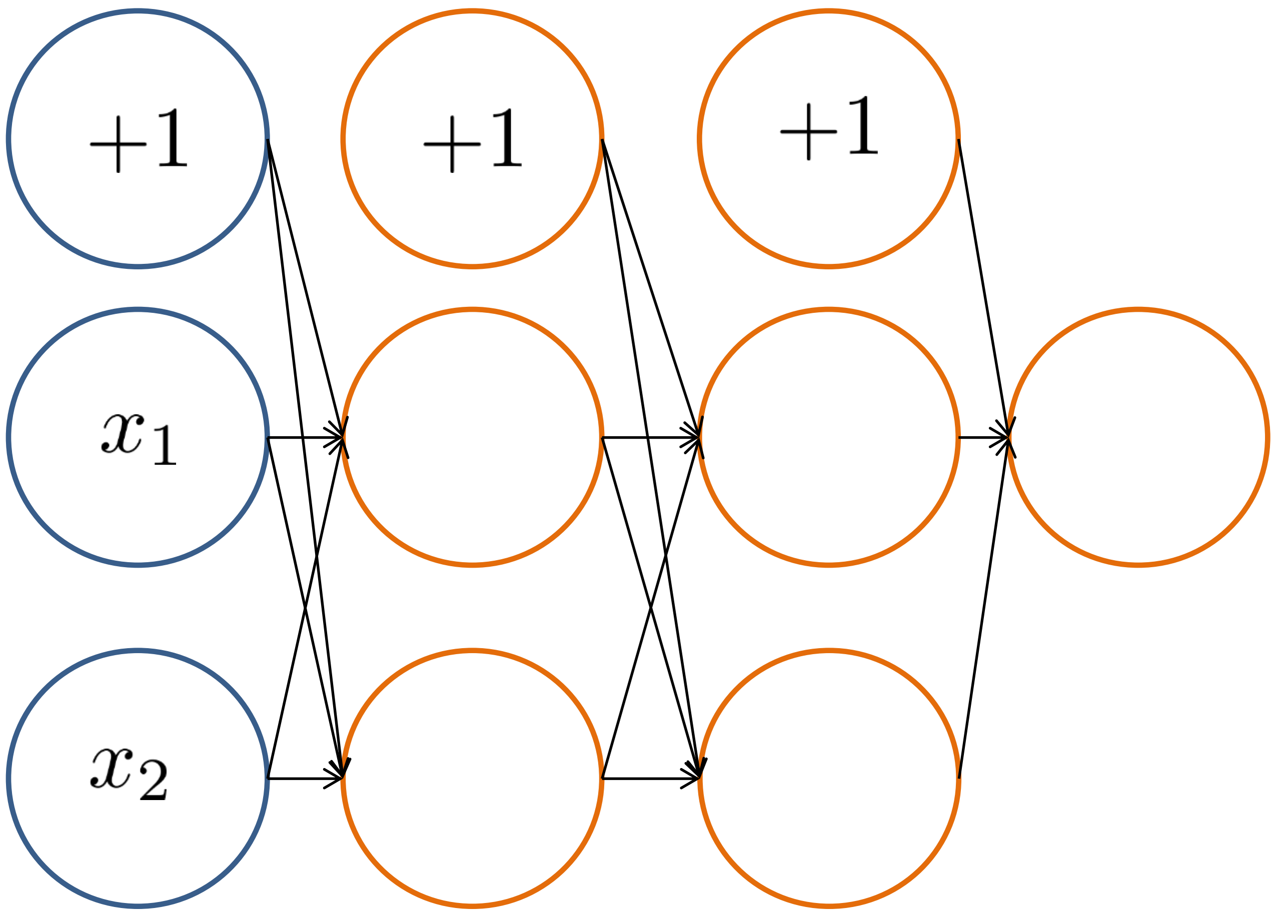
$$D_{ij}^{(l)} := \frac{1}{m} \triangle_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \text{ if } j \neq 0$$

$$D_{ij}^{(l)} := \frac{1}{m} \triangle_{ij}^{(l)} \qquad \text{if } j = 0$$

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

Derivative

Forward Propagation



What is backpropagation doing?

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

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

Focusing on a single example $x^{(i)}$, $y^{(i)}$, the case of 1 output unit, and ignoring regularization ($\lambda = 0$),

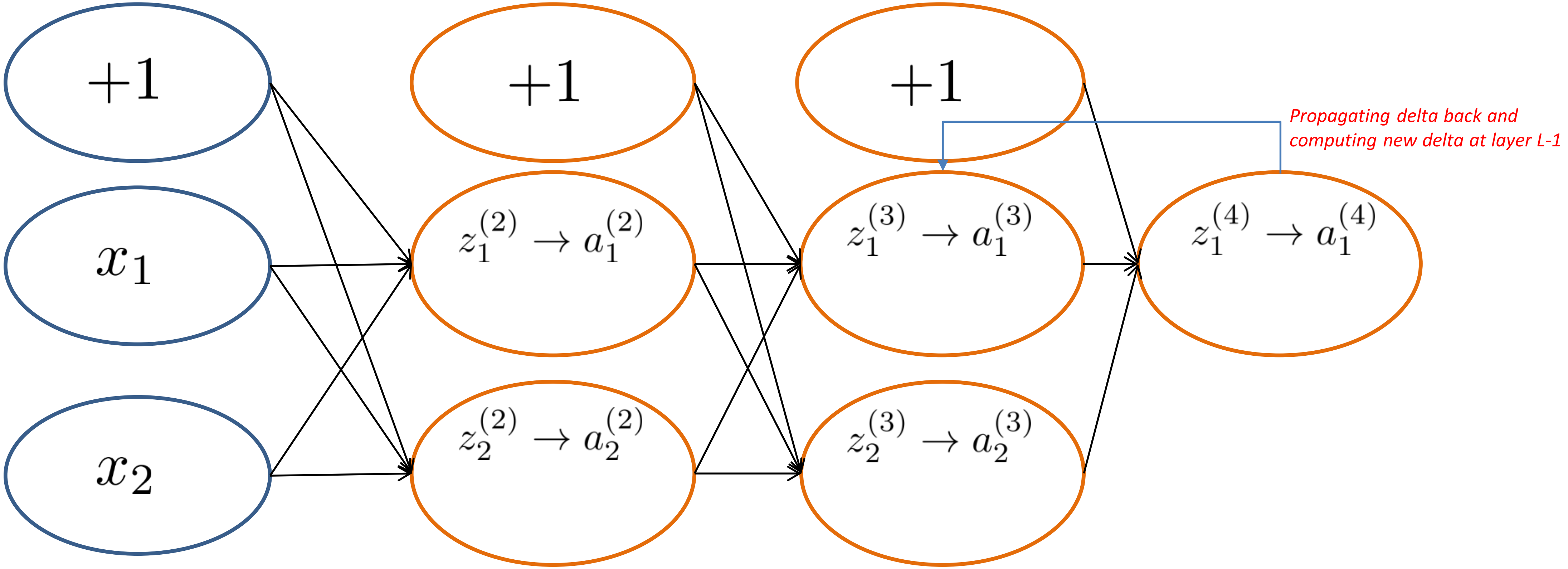
You can think of cost function as a mean square error function to get a better intuition of back propogation algorithm

$$\text{cost(i)} = y^{(i)}\log h_\Theta(x^{(i)}) + (1-y^{(i)})\log h_\Theta(x^{(i)})$$

(Think of $\text{cost(i)} \approx (h_\Theta(x^{(i)}) - y^{(i)})^2$)

I.e. how well is the network doing on example i?

Forward Propagation



$\delta_j^{(l)}$ = "error" of cost for $a_j^{(l)}$ (unit j in layer l).

Formally, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(\text{i})$ (for $j \geq 0$), where

$$\text{cost}(\text{i}) = y^{(i)} \log h_\Theta(x^{(i)}) + (1 - y^{(i)}) \log h_\Theta(x^{(i)})$$

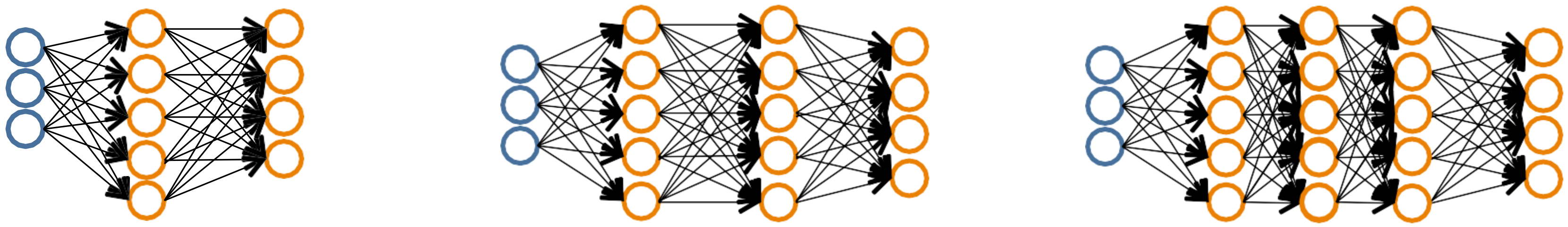
Initialization

- 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

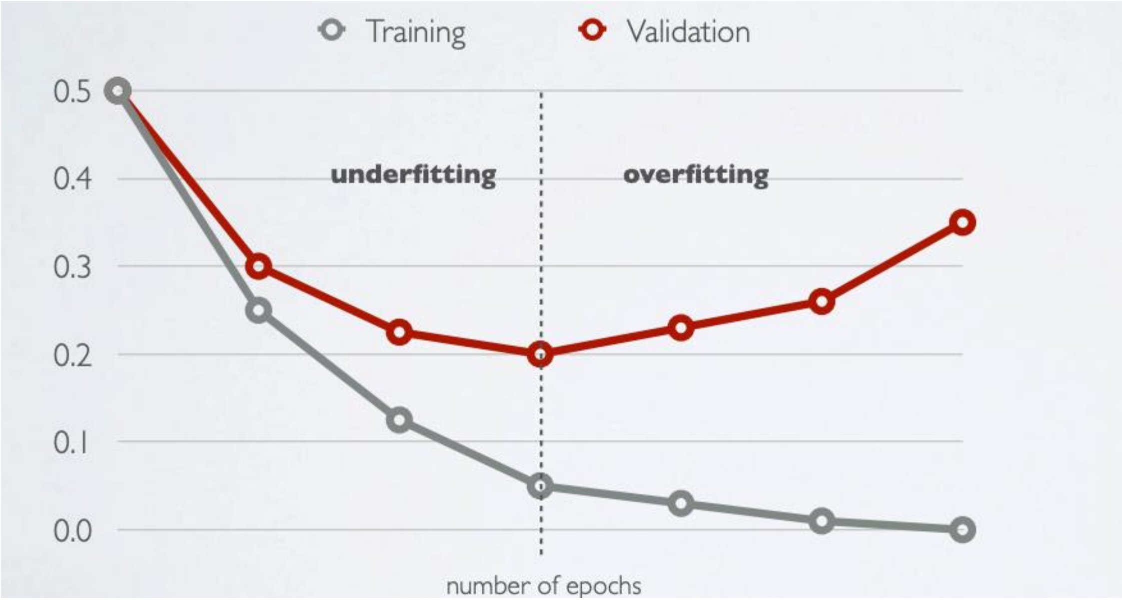


- No. of input units: Dimension of features
- No. output units: Number of classes
- Reasonable default: 1 hidden layer, or if >1 hidden layer, have same no. of hidden units in every layer (usually the more the better)
- Grid search

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