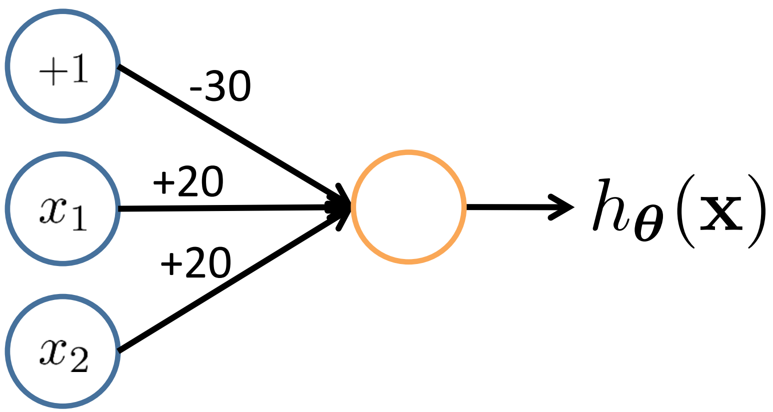
Understanding Representations

Representing Boolean Functions

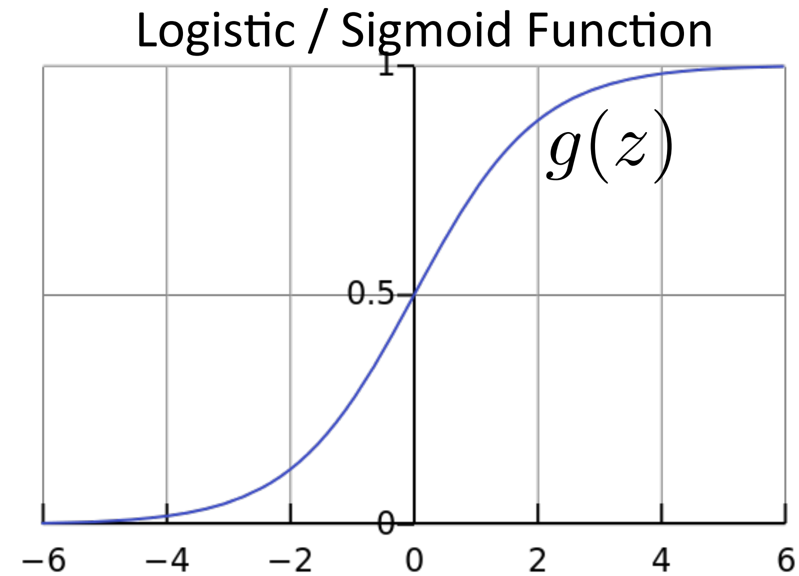
Simple example: AND

$x_1, x_2 \in \{0, 1\}$

$y = x_1 \text{ AND } x_2$



$$h_\Theta(\mathbf{x}) = g(-30 + 20x_1 + 20x_2)$$

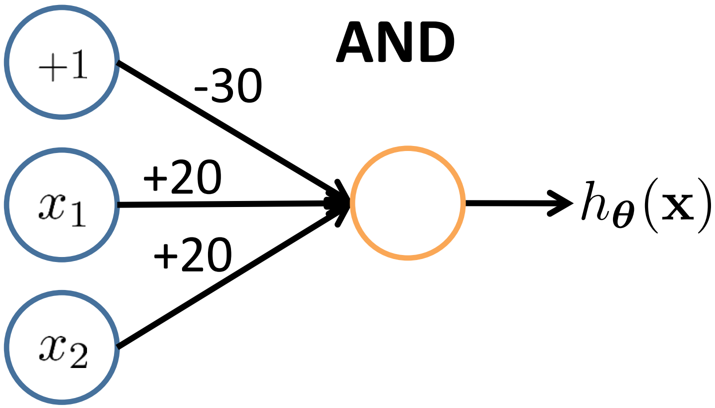


x_1	x_2	$h_\Theta(\mathbf{x})$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

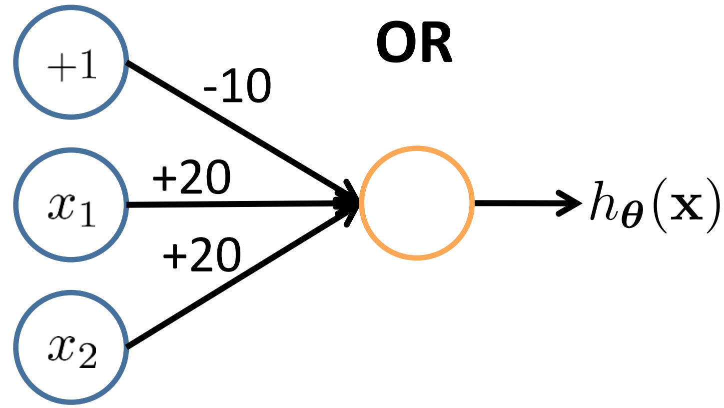
Based on slide and example by Andrew Ng

Representing Boolean Functions

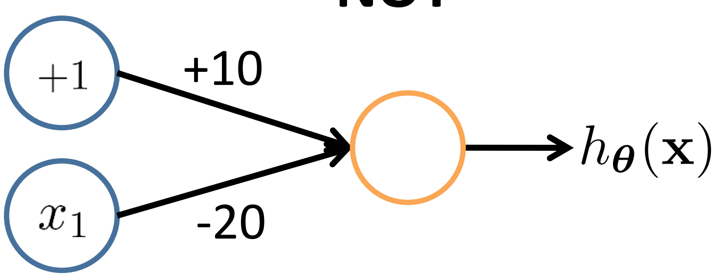
AND



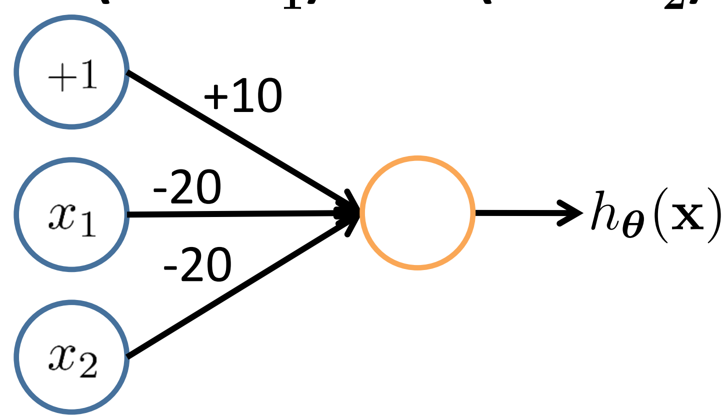
OR



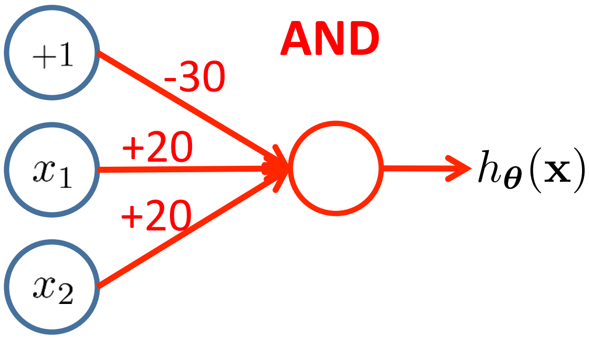
NOT

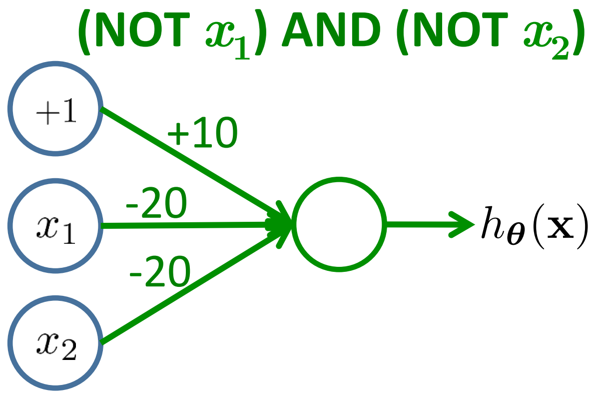


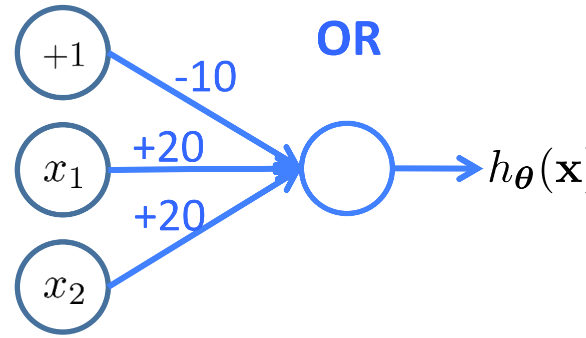
(NOT x_1) AND (NOT x_2)

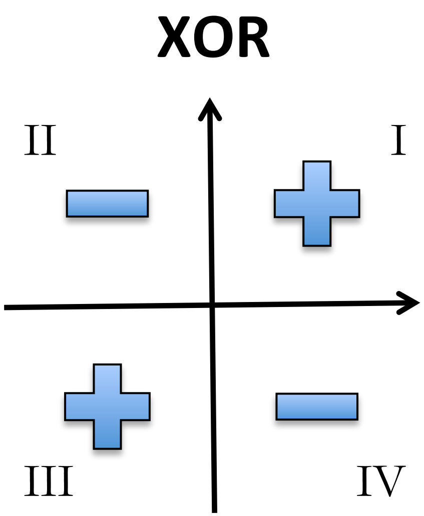


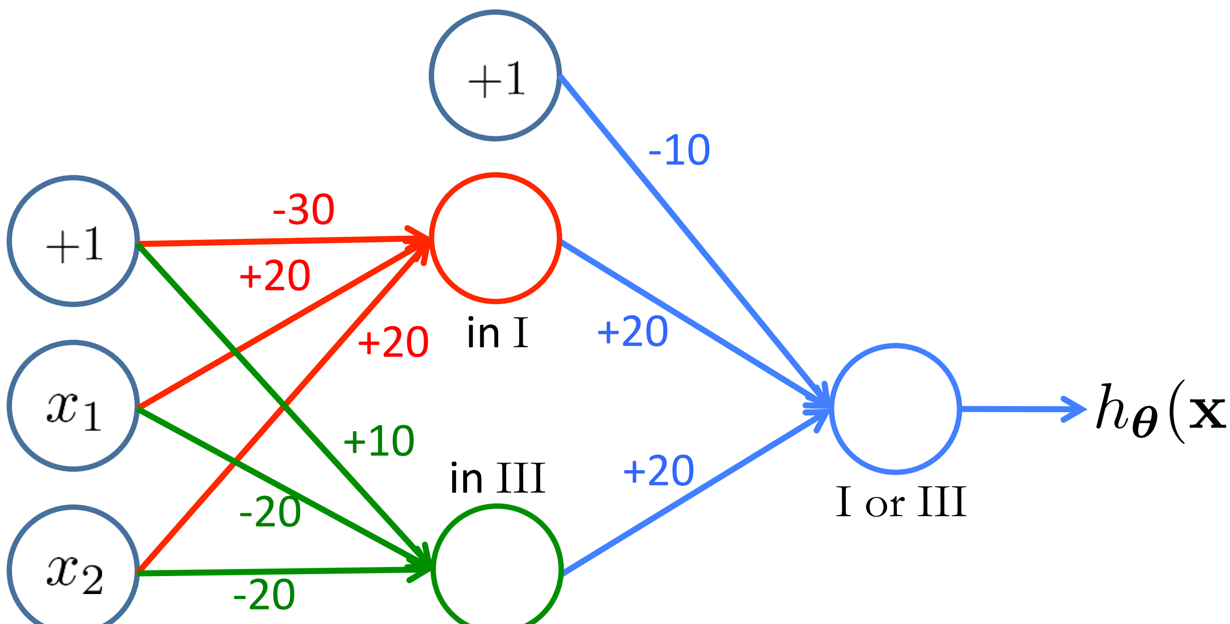
Combining Representations to Create Non-Linear Functions





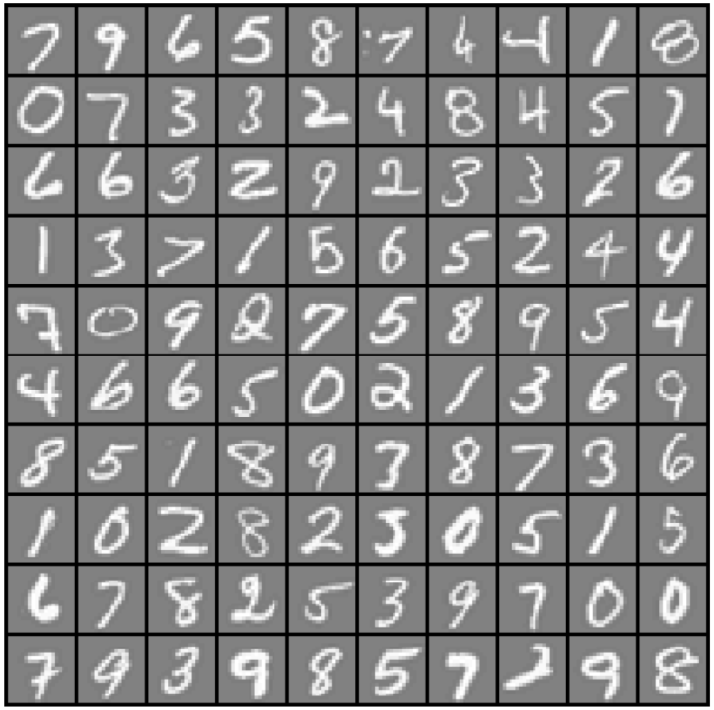




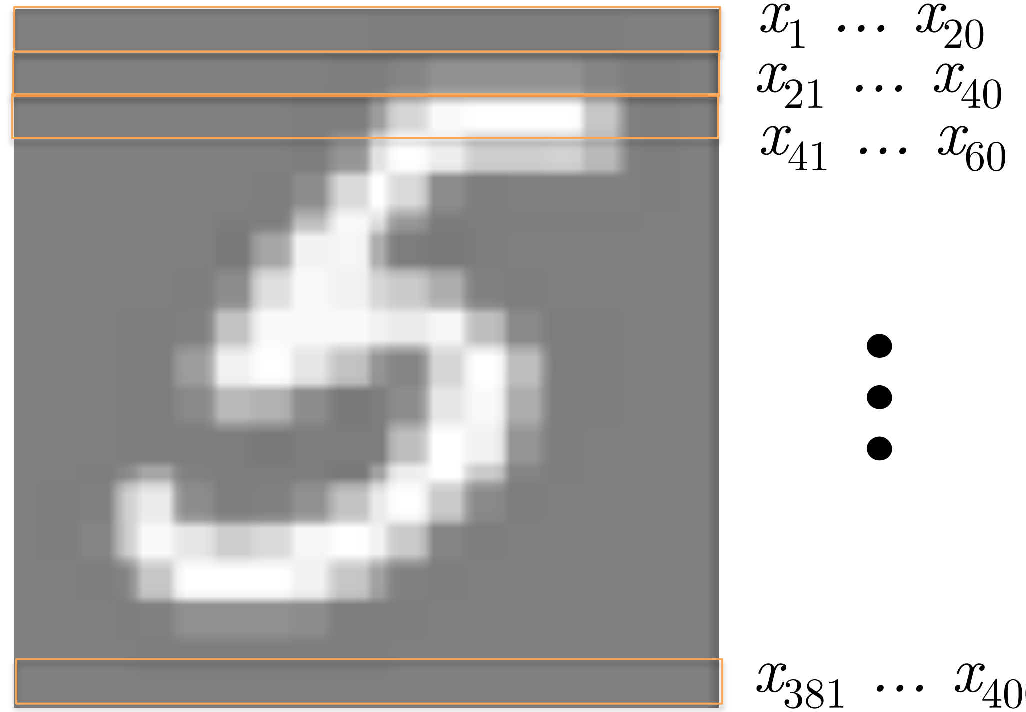


Based on example by Andrew Ng

Layering Representations

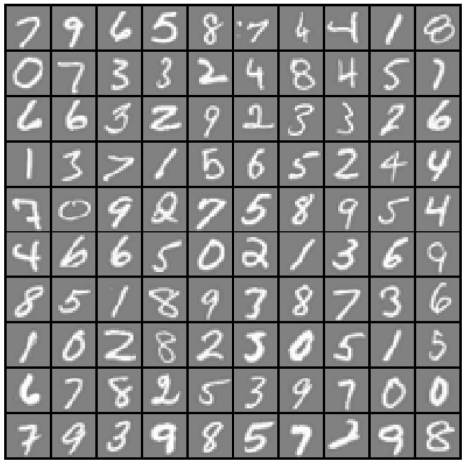


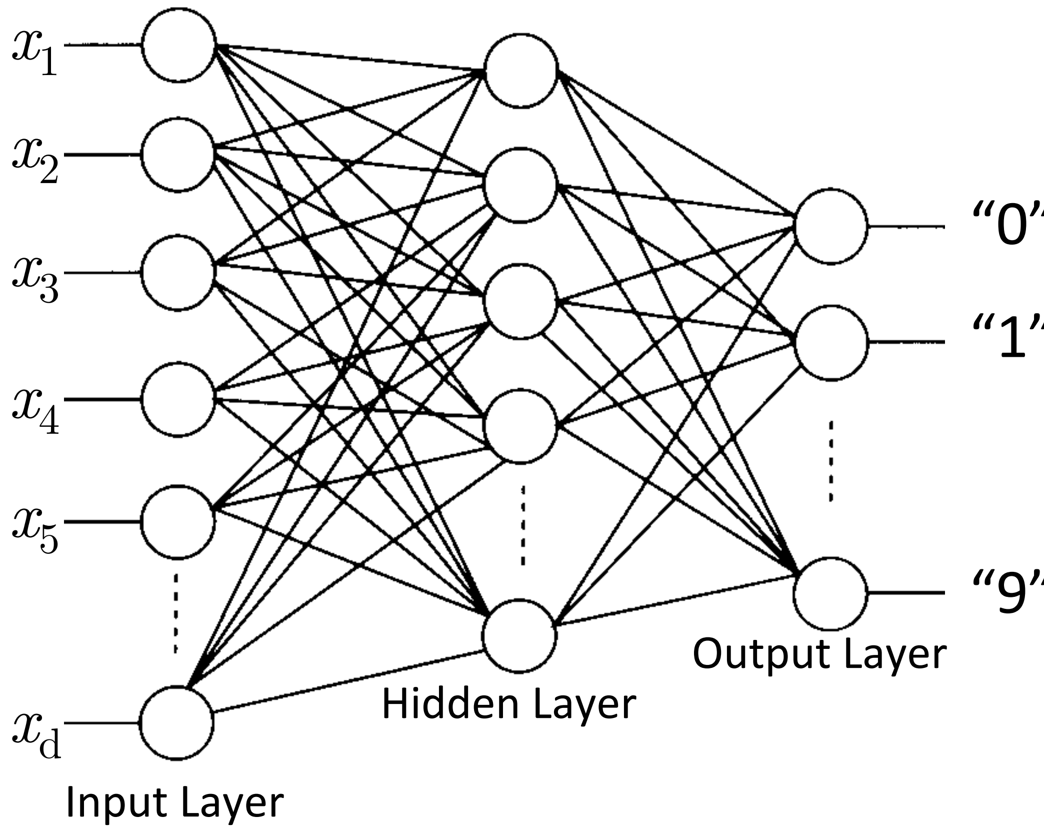
20 × 20 pixel images
d = 400   10 classes

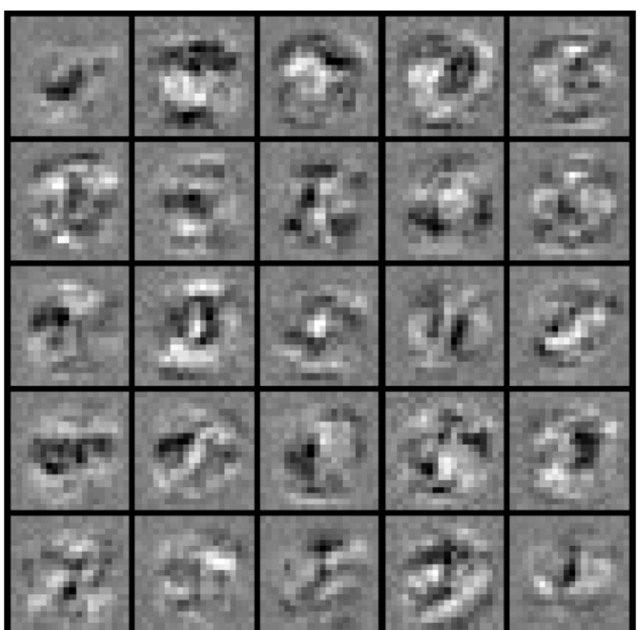


Each image is "unrolled" into a vector $\mathbf{x}$ of pixel intensities

Layering Representations







Visualization of Hidden Layer

Neural Network Learning

Perceptron Learning Rule

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha(y - h(\mathbf{x}))\mathbf{x}$$

Equivalent to the intuitive rules:

- If output is correct, don't change the weights
- If output is low ($h(\mathbf{x})$ = 0, y = 1), increment weights for all the inputs which are 1
- If output is high ($h(\mathbf{x})$ = 1, y = 0), decrement weights for all inputs which are 1

Perceptron Convergence Theorem:

- If there is a set of weights that is consistent with the training data (i.e., the data is linearly separable), the perceptron learning algorithm will converge [Minicksy & Papert, 1969]

Batch Perceptron

Given training data $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$
Let $\boldsymbol{\theta} \leftarrow [0, 0, \ldots, 0]$
Repeat:
  Let $\boldsymbol{\Delta} \leftarrow [0, 0, \ldots, 0]$
  for $i = 1 \ldots n$, do
    if $y^{(i)} \boldsymbol{x}^{(i)} \boldsymbol{\theta} \leq 0$ // prediction for ith instance is incorrect
      $\boldsymbol{\Delta} \leftarrow \boldsymbol{\Delta} + y^{(i)} \boldsymbol{x}^{(i)}$
  $\boldsymbol{\Delta} \leftarrow \boldsymbol{\Delta} / n$ // compute average update
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{\Delta}$
Until $\|\boldsymbol{\Delta}\|_2 < \epsilon$

- Simplest case: $\alpha = 1$ and don't normalize, yields the fixed increment perceptron
- Each increment of outer loop is called an **epoch**

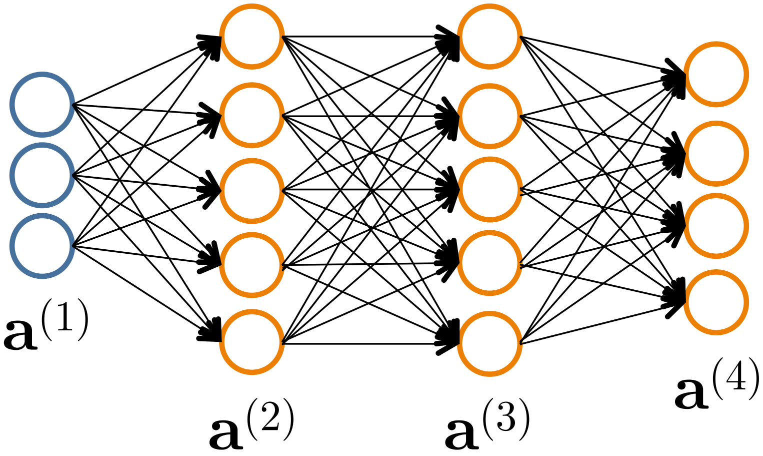
Learning in NN: Backpropagation

- Similar to the perceptron learning algorithm, we cycle through our examples
  - If the output of the network is correct, no changes are made
  - If there is an error, weights are adjusted to reduce the error

- The trick is to assess the blame for the error and divide it among the contributing weights

Forward Propagation

- Given one labeled training instance $(\mathbf{x},\ y)$:



<u>Forward Propagation</u>

- $\mathbf{a}^{(1)} = \mathbf{x}$
- $\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{a}^{(1)}$
- $\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$ [add $a_0^{(2)}$]
- $\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$
- $\mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$ [add $a_0^{(3)}$]
- $\mathbf{z}^{(4)} = \Theta^{(3)}\mathbf{a}^{(3)}$
- $\mathbf{a}^{(4)} = \mathrm{h}_\Theta(\mathbf{x}) = g(\mathbf{z}^{(4)})$

Backpropagation Intuition

- Each hidden node j is "responsible" for some fraction of the error $\delta_j^{(l)}$ in each of the output nodes to which it connects

- $\delta_j^{(l)}$ is divided according to the strength of the connection between hidden node and the output node

- Then, the "blame" is propagated back to provide the error values for the hidden layer

Based on slide by T. Finin, M. desJardins, L Getoor, R. Par