Machine Learning Lecture Notes

Lecture Notes by Şükrü Ozan, Ph.D.

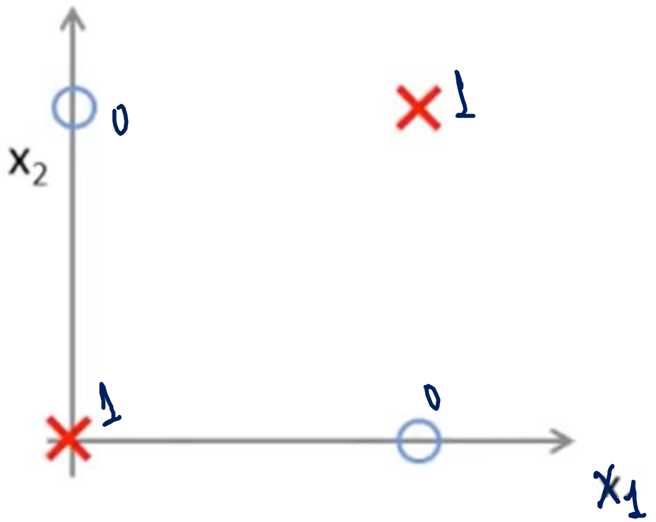
LECTURE 09

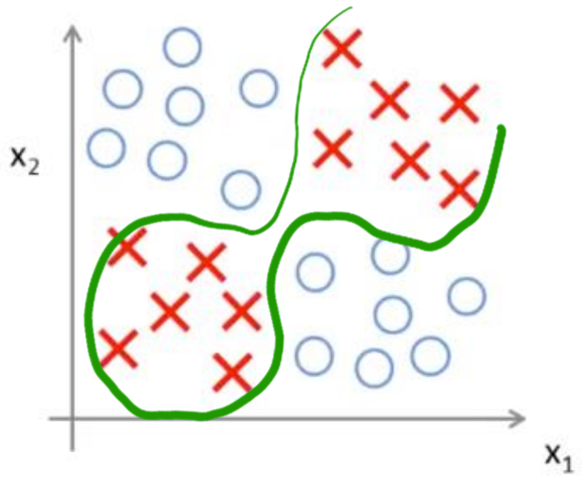
Applications

Examples and Intuitions I

Non-linear classification example: XOR/XNOR

x_1, x_2 are binary (0 or 1).



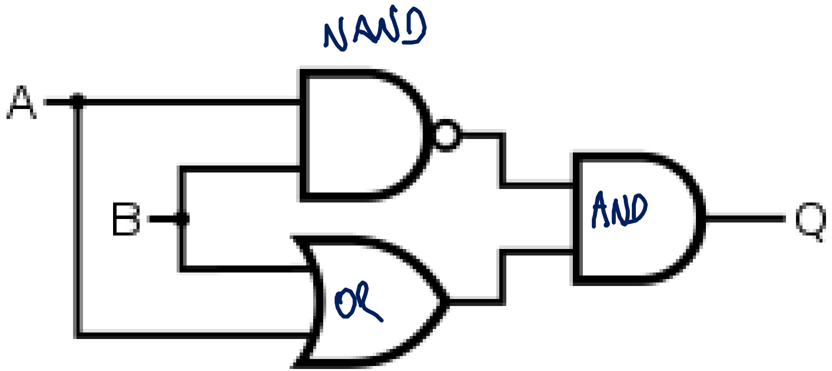


$y = x_1 \text{ XOR } x_2$

$\quad x_1 \text{ XNOR } x_2$

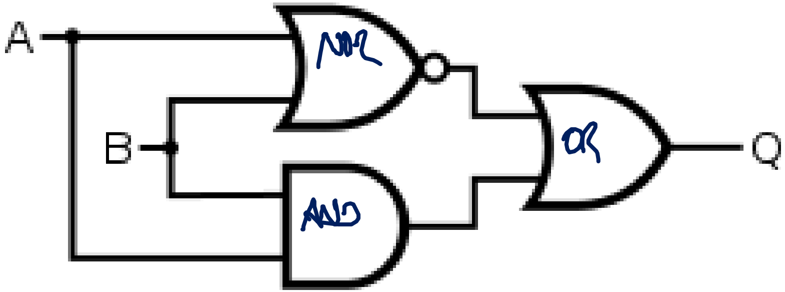
$\quad \text{NOT } (x_1 \text{ XOR } x_2)$

XOR



$$Q=(A+B)\cdot(\bar{A}+\bar{B})$$

XNOR



$$Q=(A\cdot B)+(\bar{A}\cdot\bar{B})$$

$\overline{A \cdot B} = \bar{A} + \bar{B}$

$\overline{A + B} = \bar{A} \cdot \bar{B}$

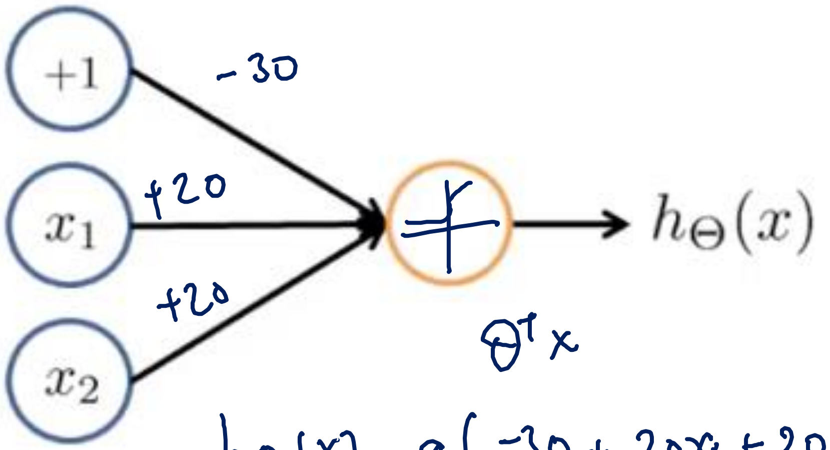
$\cdot$ AND

$+$ OR

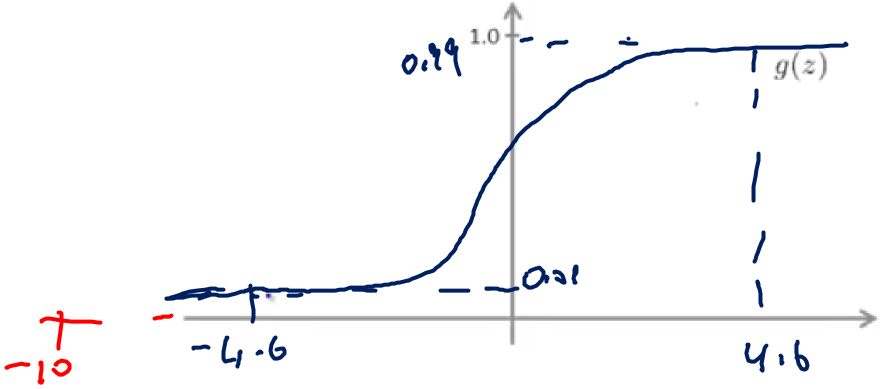
Simple example: AND

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

$y = x_1 \text{ AND } x_2$



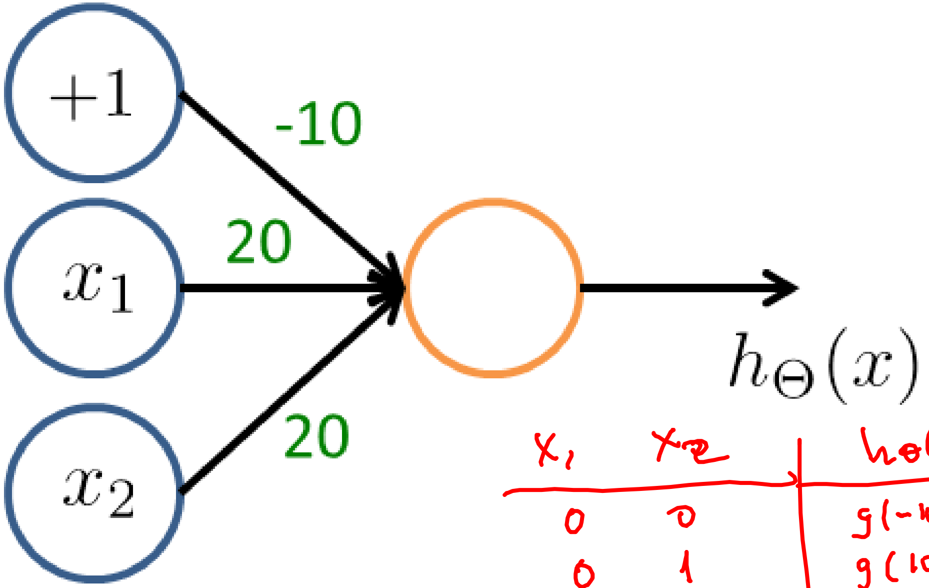
$h_\Theta(x) = g(-30 + 20x_1 + 20x_2)$



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

$h_\Theta(x) \approx x_1 \text{ AND } x_2$

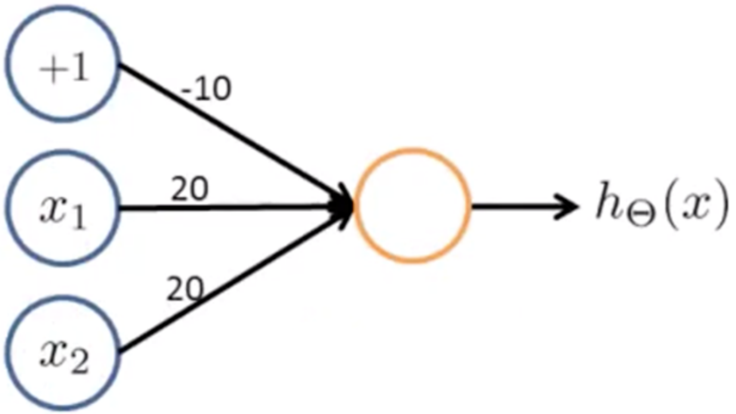
Suppose x_1 and x_2 are binary valued (0 or 1). What boolean function does the network shown below (approximately) compute? (Hint: One possible way to answer this is to draw out a truth table, similar to what we did in the video).



- ○ x_1 AND x_2
- ○ (NOT x_1) OR (NOT x_2)
- ● x_1 OR x_2
- ○ (NOT x_1) AND (NOT x_2)

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

Example: OR function



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

$h_\Theta(x) \approx x_1 \text{ OR } x_2$

+

Examples and Intuitions I

A simple example of applying neural networks is by predicting x_1 AND x_2, which is the logical 'and' operator and is only true if both x_1 and x_2 are 1.

The graph of our functions will look like:

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \rightarrow \begin{bmatrix} g(z^{(2)}) \end{bmatrix} \rightarrow h_\Theta(x)$$

Remember that x_0 is our bias variable and is always 1.

Let's set our first theta matrix as:

$$\Theta^{(1)} = \begin{bmatrix} -30 & 20 & 20 \end{bmatrix}$$

This will cause the output of our hypothesis to only be positive if both x_1 and x_2 are 1. In other words:

$$\begin{aligned} & h_\Theta(x) = g(-30 + 20x_1 + 20x_2) \\ \\ & x_1 = 0 \ \ and \ \ x_2 = 0 \ \ then \ \ g(-30) \approx 0 \\ & x_1 = 0 \ \ and \ \ x_2 = 1 \ \ then \ \ g(-10) \approx 0 \\ & x_1 = 1 \ \ and \ \ x_2 = 0 \ \ then \ \ g(-10) \approx 0 \\ & x_1 = 1 \ \ and \ \ x_2 = 1 \ \ then \ \ g(10) \approx 1 \end{aligned}$$

So we have constructed one of the fundamental operations in computers by using a small neural network rather than using an actual AND gate. Neural networks can also be used to simulate all the other logical gates. The following is an example of the logical operator 'OR', meaning either x_1 is true or x_2 is true, or both:

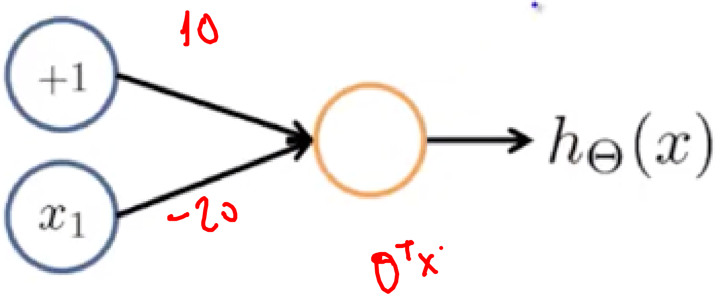
Applications

Examples and Intuitions II

x_1 AND x_2

x_1 OR x_2

Negation:

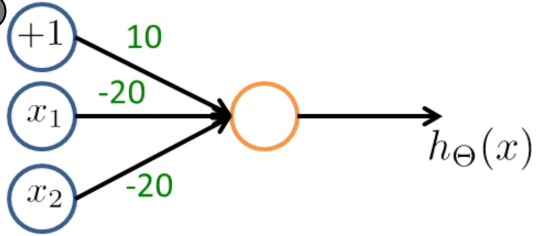


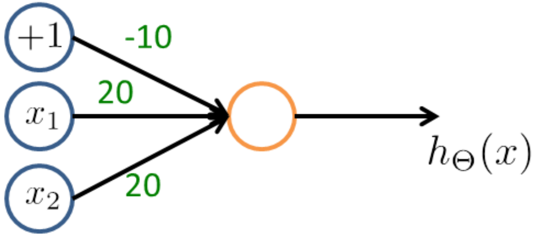
$$h_\theta(x) = g(10 - 20x_1)$$

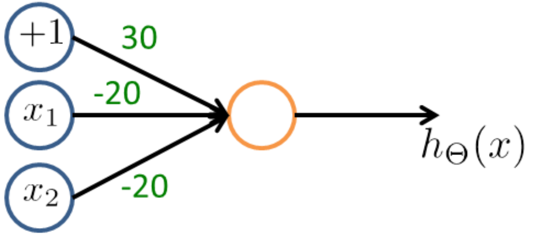
x_1	$h_\Theta(x)$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

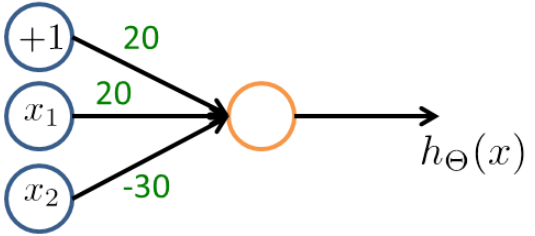
(NOT x_1) AND (NOT x_2)

Suppose that x_1 and x_2 are binary valued (0 or 1). Which of the following networks (approximately) computes the boolean function (NOT x_1) AND (NOT x_2)?





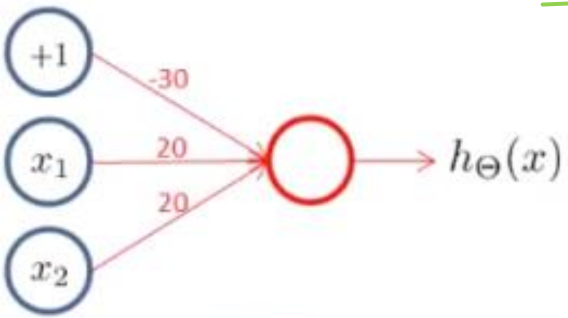




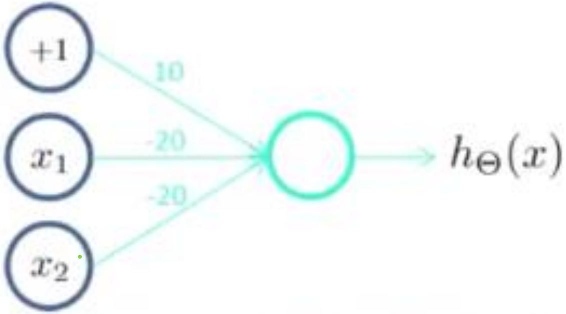
$\bar{X}_1 \cdot \bar{X}_2 = \overline{(X_1+X_2)}$ (NOR)

OR	-10	20	20
NOR	10	-20	-20

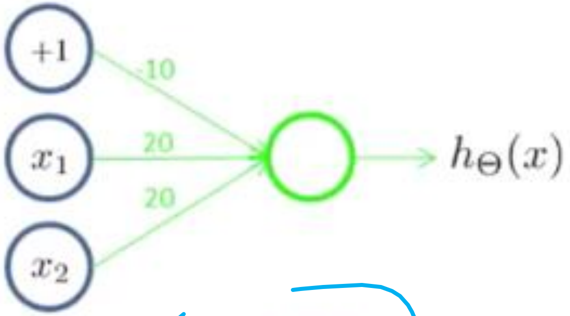
Putting it together: x_1 XNOR x_2



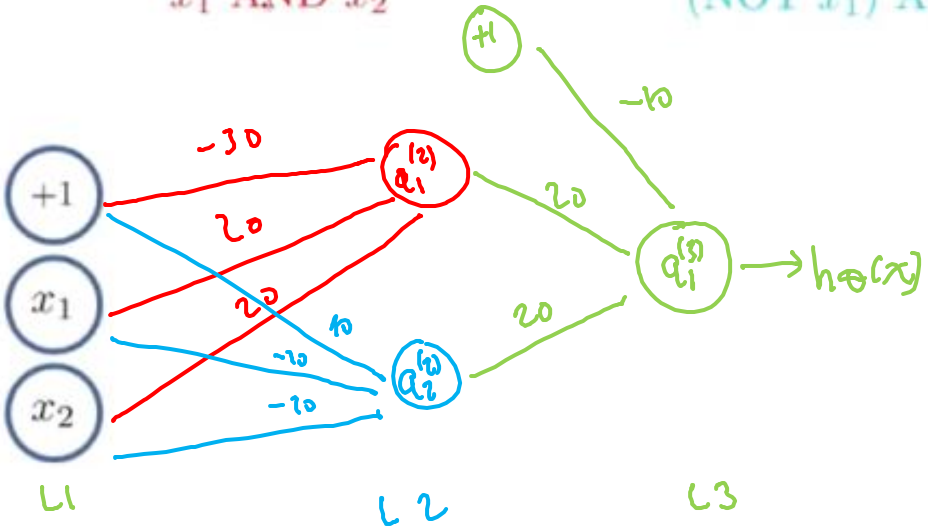
x_1 AND x_2



(NOT x_1) AND (NOT x_2)



x_1 OR x_2



x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

Examples and Intuitions II

The $\Theta^{(1)}$ matrices for AND, NOR, and OR are:

$$\begin{aligned} &AND: \\ &\Theta^{(1)} = \begin{bmatrix} -30 & 20 & 20 \end{bmatrix} \\ &NOR: \\ &\Theta^{(1)} = \begin{bmatrix} 10 & -20 & -20 \end{bmatrix} \\ &OR: \\ &\Theta^{(1)} = \begin{bmatrix} -10 & 20 & 20 \end{bmatrix} \end{aligned}$$

We can combine these to get the XNOR logical operator (which gives 1 if x_1 and x_2 are both 0 or both 1).

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \rightarrow \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \end{bmatrix} \rightarrow \begin{bmatrix} a^{(3)} \end{bmatrix} \rightarrow h_\Theta(x)$$

For the transition between the first and second layer, we'll use a $\Theta^{(1)}$ matrix that combines the values for AND and NOR:

$$\Theta^{(1)} = \begin{bmatrix} -30 & 20 & 20 \\ 10 & -20 & -20 \end{bmatrix}$$

For the transition between the second and third layer, we'll use a $\Theta^{(2)}$ matrix that uses the value for OR:

$$\Theta^{(2)} = \begin{bmatrix} -10 & 20 & 20 \end{bmatrix}$$

Let's write out the values for all our nodes:

$$\begin{aligned} a^{(2)} &= g(\Theta^{(1)} \cdot x) \\ a^{(3)} &= g(\Theta^{(2)} \cdot a^{(2)}) \\ h_\Theta(x) &= a^{(3)} \end{aligned}$$

And there we have the XNOR operator using a hidden layer with two nodes! The following summarizes the above algorithm:

Applications
Multiclass Classification

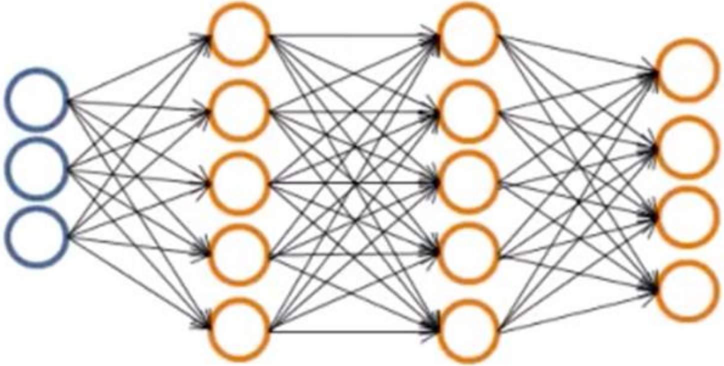
Multiple output units: One-vs-all.

Pedestrian

Car

Motorcycle

Truck

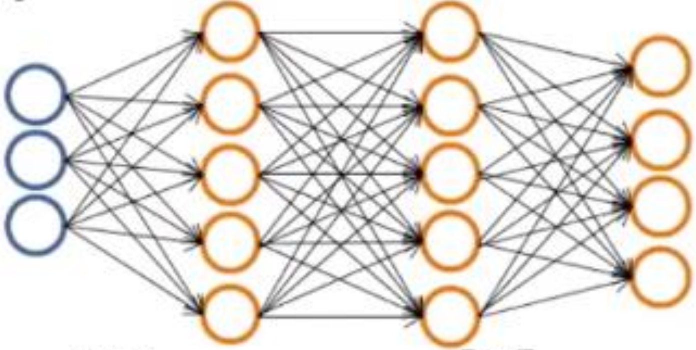


$$h_\Theta(x) \in \mathbb{R}^4$$

Want $h_\Theta(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$, etc.

when pedestrian   when car   when motorcycle

Multiple output units: One-vs-all.



$$h_\Theta(x) \in \mathbb{R}^4$$

Want $h_\Theta(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$, etc.

when pedestrian when car when motorcycle

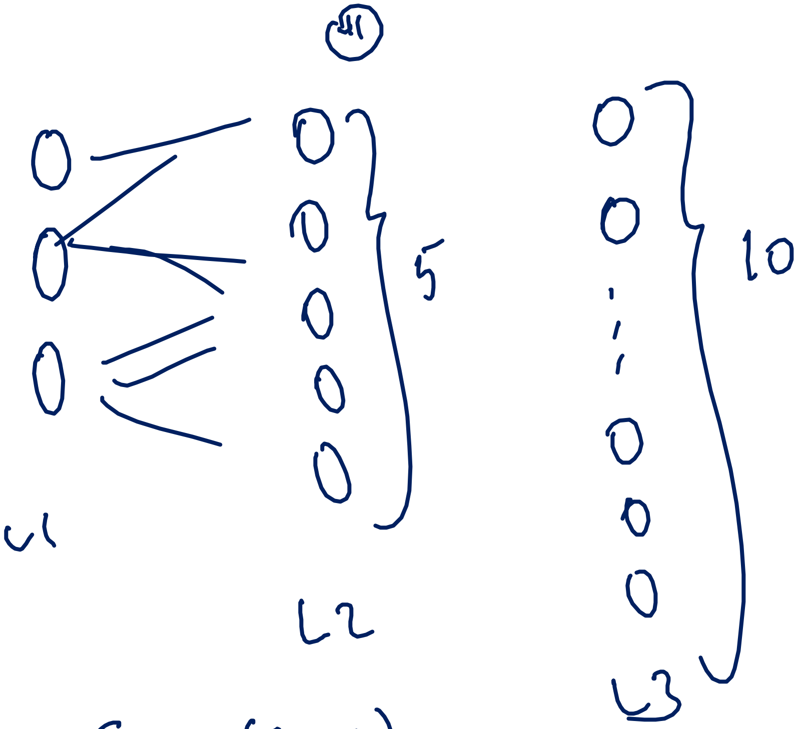
Training set: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$

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

pedestrian car motorcycle truck

Suppose you have a multi-class classification problem with 10 classes. Your neural network has 3 layers, and the hidden layer (layer 2) has 5 units. Using the one-vs-all method described here, how many elements does $\Theta^{(2)}$ have?

- 50
- 55
- 60
- 66



$S_{l+1} \times (s_l + 1)$

10×6

$\Theta^{(2)}$

Each $y^{(i)}$ represents a different image corresponding to either a car, pedestrian, truck, or motorcycle. The inner layers, each provide us with some new information which leads to our final hypothesis function. The setup looks like:

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix} \rightarrow \begin{bmatrix} a_0^{(2)} \\ a_1^{(2)} \\ a_2^{(2)} \\ \cdots \end{bmatrix} \rightarrow \begin{bmatrix} a_0^{(3)} \\ a_1^{(3)} \\ a_2^{(3)} \\ \cdots \end{bmatrix} \rightarrow \cdots \rightarrow \begin{bmatrix} h_\Theta(x)_1 \\ h_\Theta(x)_2 \\ h_\Theta(x)_3 \\ h_\Theta(x)_4 \end{bmatrix}$$

Our resulting hypothesis for one set of inputs may look like:

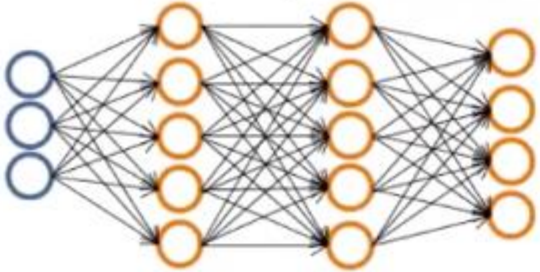
$$h_\Theta(x) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

In which case our resulting class is the third one down, or $h_\Theta(x)_3$, which represents the motorcycle.

Cost Function and Back Propagation

Cost Function

Neural Network (Classification)



$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$

$L =$ total no. of layers in network

$s_l =$ no. of units (not counting bias unit) in layer l

$L=4$

$s_1 = 3 \quad s_2 = 5 \quad s_3 = 5 \quad s_4 = s_L = 4$

Binary classification

$y = 0$ or 1

1 output unit

$h_\Theta \in \mathbb{R} \quad s_L = 1 \quad K = 1$

Multi-class classification (K classes)

$y \in \mathbb{R}^K$ E.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

pedestrian car motorcycle truck

K output units

$h_\Theta(x) \in \mathbb{R}^K \quad s_L = K \quad (K \geq 3)$

Cost function

$\Theta_j \approx 0$

Logistic regression:

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

Neural network:

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

K: # classes

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

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

$\Theta^{(l)}$

L: # layers

$l \neq 0$

$i \neq 0$ (we do not optimize

$j \neq 0$ bias terms)

Suppose we want to try to minimize $J(\Theta)$ as a function of Θ, using one of the advanced optimization methods (fminunc, conjugate gradient, BFGS, L-BFGS, etc.). What do we need to supply code to compute (as a function of Θ)?

- ○ Θ
- ○ $J(\Theta)$
- ○ The (partial) derivative terms $\frac{\partial}{\partial \Theta_{ij}^{(l)}}$ for every i, j, l
- ● $J(\Theta)$ and the (partial) derivative terms $\frac{\partial}{\partial \Theta_{ij}^{(l)}}$ for every i, j, l

Cost Function

Let's first define a few variables that we will need to use:

- L = total number of layers in the network
- s_l = number of units (not counting bias unit) in layer l
- K = number of output units/classes

Recall that in neural networks, we may have many output nodes. We denote $h_\Theta(x)_k$ as being a hypothesis that results in the k^{th} output. Our cost function for neural networks is going to be a generalization of the one we used for logistic regression. Recall that the cost function for regularized logistic regression was:

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

For neural networks, it is going to be slightly more complicated:

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

We have added a few nested summations to account for our multiple output nodes. In the first part of the equation, before the square brackets, we have an additional nested summation that loops through the number of output nodes.

In the regularization part, after the square brackets, we must account for multiple theta matrices. The number of columns in our current theta matrix is equal to the number of nodes in our current layer (including the bias unit). The number of rows in our current theta matrix is equal to the number of nodes in the next layer (excluding the bias unit). As before with logistic regression, we square every term.

Note:

- the double sum simply adds up the logistic regression costs calculated for each cell in the output layer
- the triple sum simply adds up the squares of all the individual Θs in the entire network.
- the i in the triple sum does **not** refer to training example i

Cost Function and Back Propagation

Backpropagation Algorithm

Gradient computation

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

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

Need code to compute:

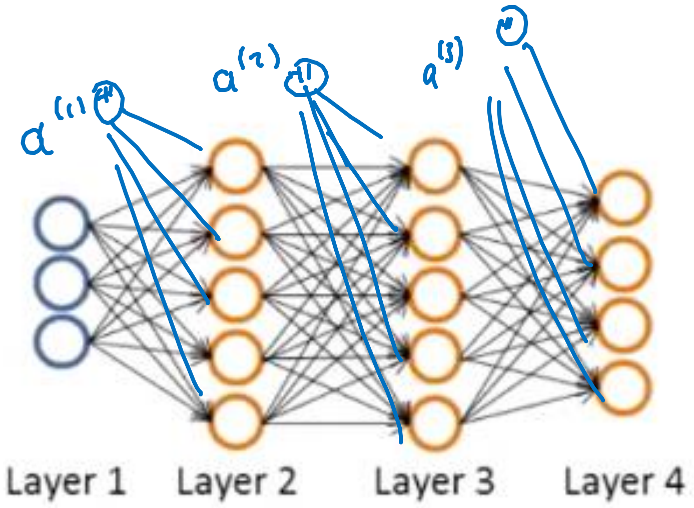
- $J(\Theta)$
- $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

Gradient computation

Given one training example (x, y):

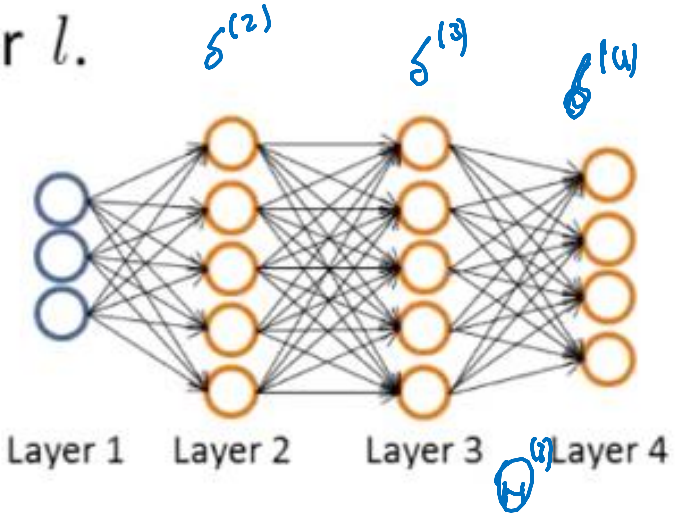
Forward propagation:

$$
\begin{aligned}
a^{(1)} &= x \\
z^{(2)} &= \Theta^{(1)} a^{(1)} \\
a^{(2)} &= g(z^{(2)}) \quad (\text{add } a_0^{(2)}) \\
z^{(3)} &= \Theta^{(2)} a^{(2)} \\
a^{(3)} &= g(z^{(3)}) \quad (\text{add } a_0^{(3)}) \\
z^{(4)} &= \Theta^{(3)} a^{(3)} \\
a^{(4)} &= h_\Theta(x) = g(z^{(4)})
\end{aligned}
$$



Gradient computation: Backpropagation algorithm

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



For each output unit (layer L = 4)

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

$g(z^{(3)}) = a^{(3)}$

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$$

$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$

$\sigma'(x) = \sigma(x)(1-\sigma(x))$

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

$g'(z^{(3)}) = g(z^{(3)})(1 - g(z^{(3)}))$

$a^{(3)}(1 - a^{(3)})$

Backpropagation algorithm

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

Set $\triangle_{ij}^{(l)} = 0$ (for all l, i, j). → This term is used as an accumulator in order to calculate $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

For $i = 1$ to m ← $(x^{(i)}, y^{(i)})$

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

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

This can be implemented as a vectoral operation i.e.:

$\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} a^{(l)T}$

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

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

Suppose you have two training examples $(x^{(1)}, y^{(1)})$ and $(x^{(2)}, y^{(2)})$. Which of the following is a correct sequence of operations for computing the gradient? (Below, FP = forward propagation, BP = back propagation).

- ○ FP using $x^{(1)}$ followed by FP using $x^{(2)}$. Then BP using $y^{(1)}$ followed by BP using $y^{(2)}$.
- ○ FP using $x^{(1)}$ followed by BP using $y^{(2)}$. Then FP using $x^{(2)}$ followed by BP using $y^{(1)}$.
- ○ BP using $y^{(1)}$ followed by FP using $x^{(1)}$. Then BP using $y^{(2)}$ followed by FP using $x^{(2)}$.
- ● FP using $x^{(1)}$ followed by BP using $y^{(1)}$. Then FP using $x^{(2)}$ followed by BP using $y^{(2)}$.

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

Where L is our total number of layers and $a^{(L)}$ is the vector of outputs of the activation units for the last layer. So our "error values" for the last layer are simply the differences of our actual results in the last layer and the correct outputs in y. To get the delta values of the layers before the last layer, we can use an equation that steps us back from right to left:

4. Compute $\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)}$ using $\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)})\ .*\ a^{(l)}\ .*\ (1 - a^{(l)})$

The delta values of layer l are calculated by multiplying the delta values in the next layer with the theta matrix of layer l. We then element-wise multiply that with a function called g', or g-prime, which is the derivative of the activation function g evaluated with the input values given by $z^{(l)}$.

The g-prime derivative terms can also be written out as:

$$g'(z^{(l)}) = a^{(l)}\ .*\ (1 - a^{(l)})$$

5. $\Delta_{i,j}^{(l)} := \Delta_{i,j}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ or with vectorization, $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

Hence we update our new Δ matrix.

- $D_{i,j}^{(l)} := \dfrac{1}{m} \left(\Delta_{i,j}^{(l)} + \lambda \Theta_{i,j}^{(l)} \right)$, if j≠0.
- $D_{i,j}^{(l)} := \dfrac{1}{m} \Delta_{i,j}^{(l)}$ If j=0

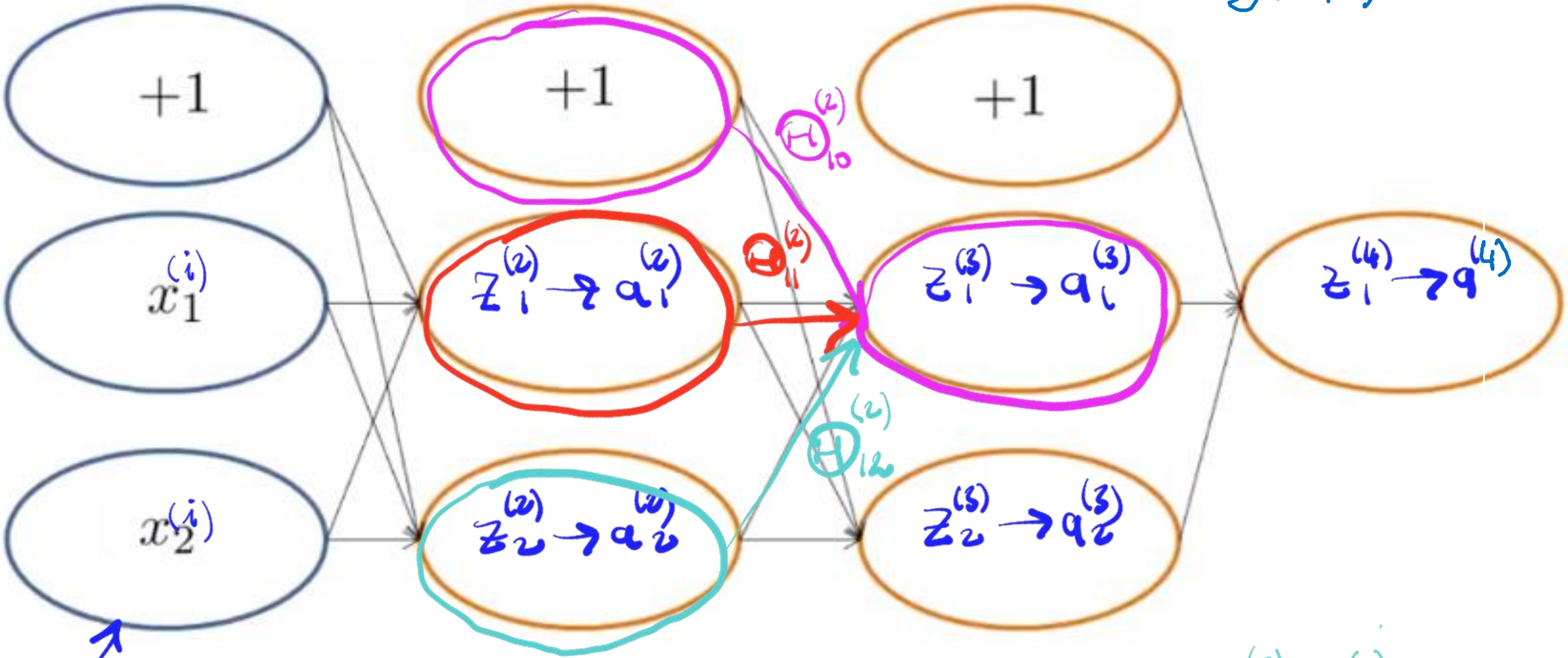
The capital-delta matrix D is used as an "accumulator" to add up our values as we go along and eventually compute our partial derivative. Thus we get $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$= $D_{ij}^{(l)}$

Cost Function and Backpropagation

Backpropagation Intuition

Forward Propagation

$$g(z_1^{(3)}) = a_1^{(3)}$$



$(x^{(i)}, y^{(i)})$

$$z_1^{(3)} = \Theta_{10}^{(2)} \times 1 + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a^{(2)}_{\cancel{1}\,2}$$

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 ($\lambda = 0$),

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

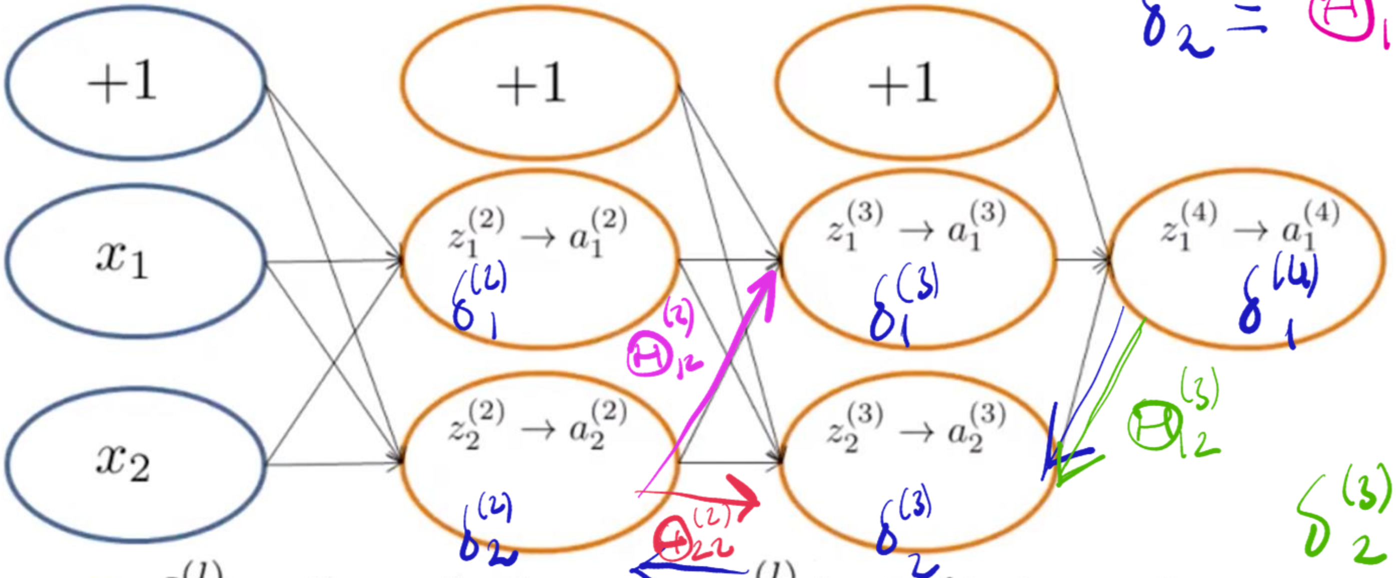
(Think of $\text{cost(i)} \approx (h_\Theta(x^{(i)}) - y^{(i)})^2$)

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

$$\delta_1^{(4)} = a_1^{(4)} - y_i$$

$$\delta_2^{(2)} = \Theta_{12}^{(2)} \delta_1^{(3)} + \Theta_{22}^{(2)} \delta_2^{(3)}$$

Forward Propagation



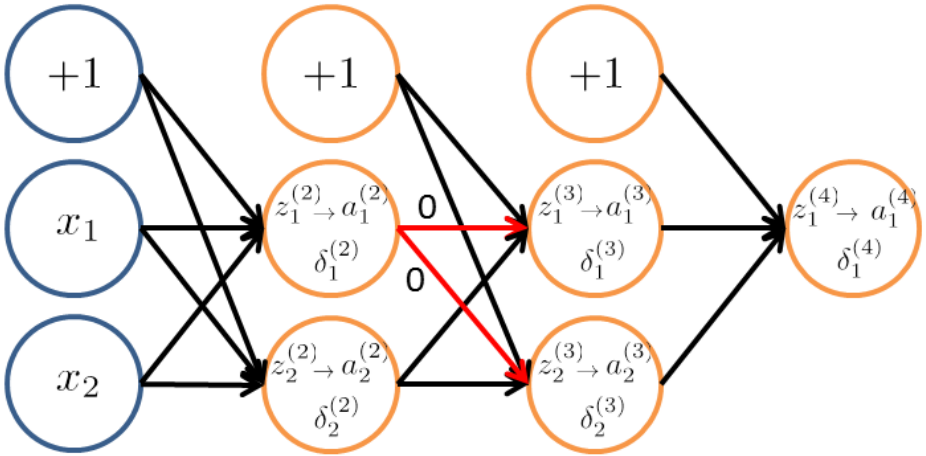
$$\delta_2^{(3)} = \Theta_{12}^{(3)} \delta_1^{(4)}$$

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

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

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

Consider the following neural network:



Suppose both of the weights shown in red ($\Theta_{11}^{(2)}$ and $\Theta_{21}^{(2)}$) are equal to 0. After running backpropagation, what can we say about the value of $\delta_1^{(3)}$?

- ○ $\delta_1^{(3)} > 0$
- ○ $\delta_1^{(3)} = 0$ only if $\delta_1^{(2)} = \delta_2^{(2)} = 0$, but not necessarily otherwise
- ○ $\delta_1^{(3)} \leq 0$ regardless of the values of $\delta_1^{(2)}$ and $\delta_2^{(2)}$
- ● There is insufficient information to tell

Recall that the cost function for a neural network is:

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

If we consider simple non-multiclass classification (k = 1) and disregard regularization, the cost is computed with:

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

Intuitively, $\delta_j^{(l)}$ is the "error" for $a_j^{(l)}$ (unit j in layer l). More formally, the delta values are actually the derivative of the cost function:

$$\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} cost(t)$$

Recall that our derivative is the slope of a line tangent to the cost function, so the steeper the slope the more incorrect we are. Let us consider the following neural network below and see how we could calculate some $\delta_j^{(l)}$: