# Week 4

— Neural Networks: Representation (Non-linear hypotheses)

When we have $n$ features and $n$ is relatively large, the map features we use to add more polynomial features makes so many more features so we can ending overfitting the hypothesis and it requires so much computational effort to minimize the function.

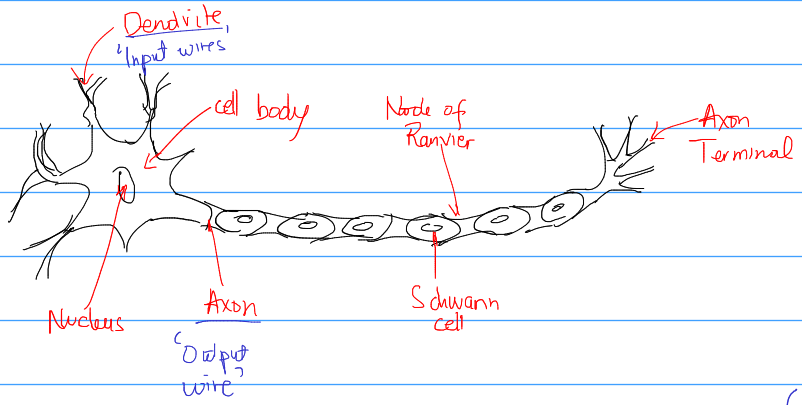
— Neural Networks : Representation (Neurons and the brain)

Origins: Algorithms that try to mimic the brain

Was very widely used in 80s and early 90s; popularity diminished in late 90s

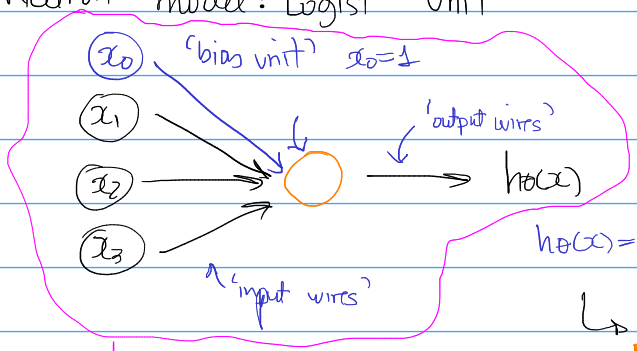
Recent resurgence: State of the art techniques for many applications.

— Neural Networks : Representation (Model Representation I)




Neuron model: Logist Unit

↓ 'weights' = 'parameters'




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

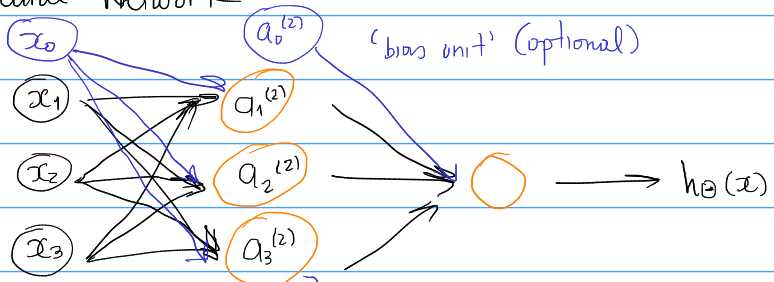
$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

↳ sigmoid (logistic) activation function

[This is like linear regression]

↳ Represent a single neuron.

Neural Network




Layer 1 (Input layer)

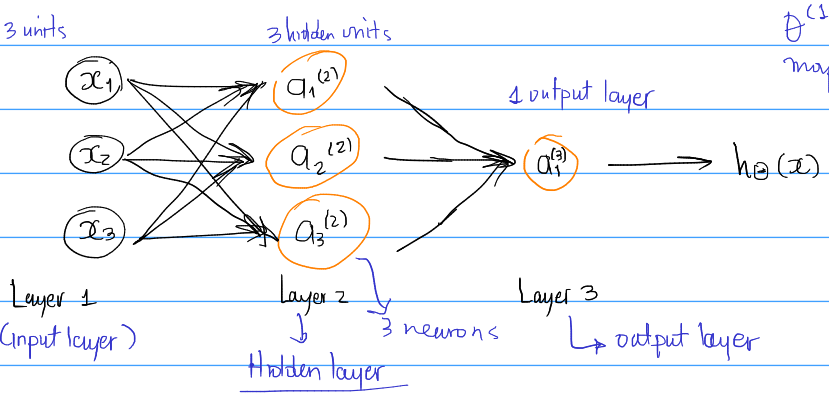
Layer 2 ↳ 3 neurons — Hidden layer

Layer 3 ↳ output layer

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

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

## Neural Network




* Important *

$\Theta^{(1)}$: matrix of weights controlling function mapping from layer $j=1$ to layer $j+1=2$

$\Theta^{(1)} \in \mathbb{R}^{3\times 4}$ (hidden layer (without bias) from $i+1$ layer; input layer (with bias) from $i$ layer)

$$\begin{matrix} & (0) & (1) & (2) & (3) \\ (1) & \Theta_{10}^{(1)} & \Theta_{11}^{(1)} & \Theta_{12}^{(1)} & \Theta_{13}^{(1)} \\ (2) & \Theta_{20}^{(1)} & \Theta_{21}^{(1)} & \Theta_{22}^{(1)} & \Theta_{23}^{(1)} \\ (3) & \Theta_{30}^{(1)} & \Theta_{31}^{(1)} & \Theta_{32}^{(1)} & \Theta_{33}^{(1)} \end{matrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

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

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

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

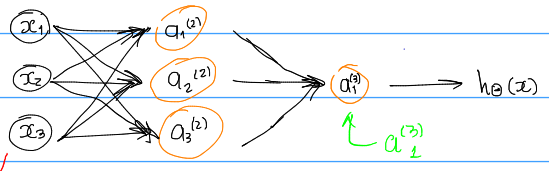
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

$\Theta^{(2)} \in \mathbb{R}^{1\times 4}$

$$\begin{bmatrix} \Theta_{10}^{(2)} & \Theta_{11}^{(2)} & \Theta_{12}^{(2)} & \Theta_{13}^{(2)} \end{bmatrix} \cdot \begin{bmatrix} a_0^{(2)} \\ a_1^{(1)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix}$$

If network has $s_j$ units in layer $j$, $s_{j+1}$ units in layer $j+1$, then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$

## Neural Network: Representation (Model Representation II)




$a^{(1)} = x$

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

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

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

$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}) = g(z_1^{(3)})$$

$a_1^{(3)}$

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

$z^{(2)} = \Theta^{(1)} a^{(1)}$ (add $a_0^{(2)} = 1 \rightarrow a^{(2)} \in \mathbb{R}^4$)

$a^{(2)} = g(z^{(2)})$ ($\mathbb{R}^3$, $\mathbb{R}^3$)

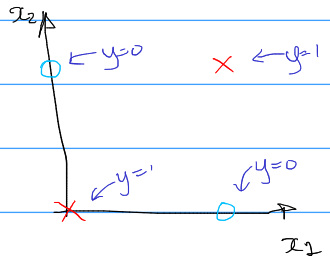
$\Rightarrow z^{(3)} = \Theta^{(2)} a^{(2)}$

$h_\Theta(x) = a^{(3)} = g(z^{(3)})$

**Forward Propagation**

There are several different network architecture

## — Neural Network: representation (Examples and intuitions 1)




$y = x_1$ XOR $x_2$

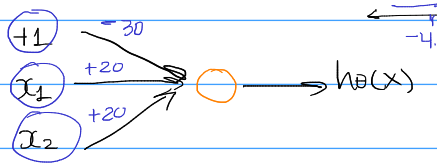
$x_1$ XNOR $x_2$

NOT ($x_1$ XOR $x_2$)

Simple Example: AND

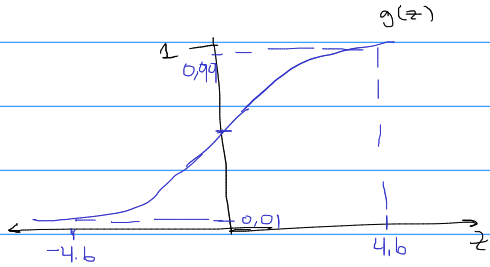
$x_1, x_2 \in \{0,1\}$

$y = x_1$ AND $x_2$




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

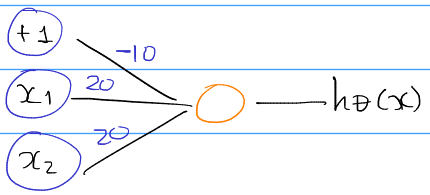
$\theta_{10}^{(1)}$ $\theta_{11}^{(1)}$ $\theta_{12}^{(1)}$




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

$h_\theta(x) \approx x_1$ AND $x_2$

### Example: OR function




$g(-10 + 20x_1 + 20x_2)$

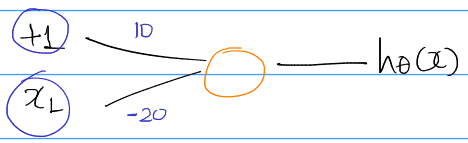
| $x_1$ | $x_2$ | $h_\theta(x)$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

$h_\theta(x) \approx x_1$ OR $x_2$

## — Neural Networks: Representation (Examples and Intuitions II)

### Negation

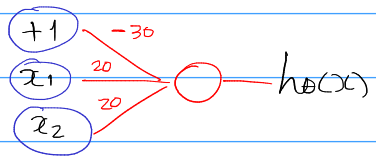
NOT $x_1$



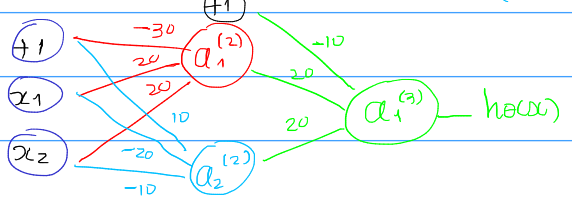

| $x_1$ | $h_\theta(x)$ |
|---|---|
| 0 | $g(10) \approx 1$ |
| 1 | $g(-10) \approx 0$ |

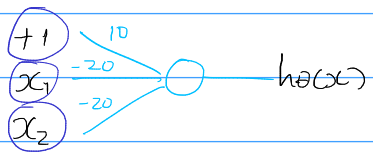
$h_\theta(x) = g(10 - 20x_1)$

## — Putting it together —

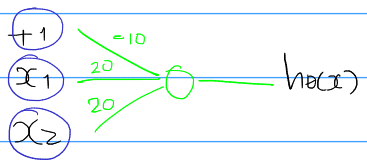



$x_1$ AND $x_2$




(NOT $x_1$) AND (NOT $x_2$)



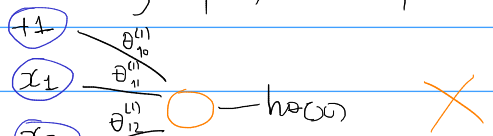

$x_1$ OR $x_2$

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

# Quiz

① – The activation values of the hidden units in a neural network, with the sigmoid activation function applied at every layer, are always in the range $(0,1)$

| $x_1$ | $x_2$ | $h_\theta(x)$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

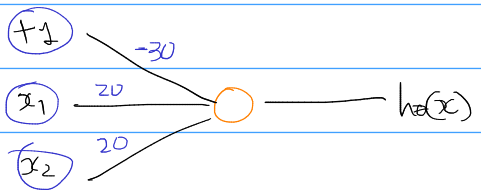



$h_\theta(x) = g(\theta_{10}^{(1)} + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2)$

$\theta_{10}^{(1)}$ must be a positive number

– Any logical function over binary-valued (0 or 1) inputs $x_1$ and $x_2$ can be (approximately) represented using some neural network.

②

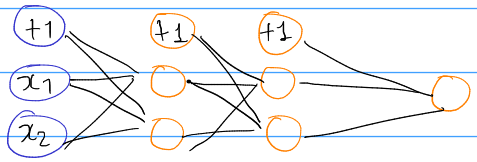



$h_\theta(x) = g(-30 + 20x_1 + 20x_2)$

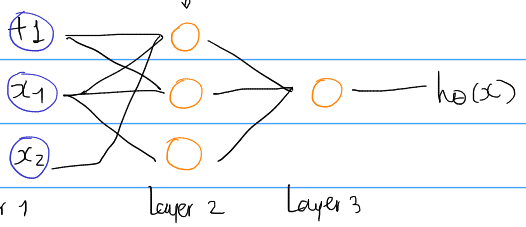
$h_\theta(x) = x_1$ AND $x_2$

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

③ $a_1^{(2)} = g(\theta_{10}^{(2)} a_0^{(2)} + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_2^{(2)})$




④




Compute the activations of hidden layer $a^{(2)} \in \mathbb{R}^3$

$\theta^{(1)} \in \mathbb{R}^{3 \times 3}$ → 3 = lenght($a^{(2)}$); 3 = lenght(input vector)+1

$$\begin{matrix} & (0) & (1) & (2) \\ (1) & \theta_{10}^{(1)} & \theta_{11}^{(1)} & \theta_{12}^{(1)} \\ (2) & \theta_{20}^{(1)} & \theta_{21}^{(1)} & \theta_{22}^{(1)} \\ (3) & \theta_{30}^{(1)} & \theta_{31}^{(1)} & \theta_{32}^{(1)} \end{matrix} \quad \begin{matrix} (1) \\ (2) \\ (3) \end{matrix} \begin{bmatrix} +1 \\ x_1 \\ x_2 \end{bmatrix}$$

$3 \times 3 \quad 3 \times 1 = 3 \times 3$

⑤

$$a^{(2)} = \theta^{(1)} x = \begin{bmatrix} 1 & -1.5 & 3.7 \\ 1 & 5.1 & 2.3 \end{bmatrix} \cdot \begin{bmatrix} +1 \\ x_1 \\ x_2 \end{bmatrix}$$

$(2 \times 3) \quad (3 \times 1)$

$$= \begin{bmatrix} 1 - 1.5x_1 + 3.7x_2 \\ 1 + 5.1x_1 + 2.3x_2 \end{bmatrix} = a^{(2)}$$

vectorized form: $\theta^{(1)} \cdot X = \begin{bmatrix} \theta_{10}^{(1)} + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 \\ \theta_{20}^{(1)} + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 \\ \theta_{30}^{(1)} + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 \end{bmatrix}$

$(3 \times 1)$

$$a^{(3)} = \theta^{(2)} a^{(2)} = \begin{bmatrix} 1 & 0.6 & -0.8 \end{bmatrix} \cdot \begin{bmatrix} +1 \\ 1 - 1.5x_1 + 3.7x_2 \\ 1 + 5.1x_1 + 2.3x_2 \end{bmatrix}$$

$(1 \times 3) \quad (3 \times 1)$

$a^{(3)} = 1 + 0.6(1 - 1.5x_1 + 3.7x_2) - 0.8(1 + 5.1x_1 + 2.3x_2) = h_\theta(x)$ 1

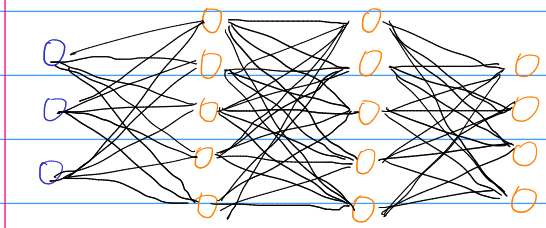
$$a^{(2)} = \theta^{(1)} x = \underset{2\times 3}{\begin{bmatrix} 1 & 5.1 & 2.3 \\ 1 & -1.5 & 3.7 \end{bmatrix}} \underset{3\times 1}{\begin{bmatrix} +1 \\ x_1 \\ x_2 \end{bmatrix}} \rightarrow \begin{bmatrix} 1 + 5.1x_1 + 2.3x_2 \\ 1 - 1.5x_1 + 3.7x_2 \end{bmatrix}$$

$$a^{(3)} = \theta^{(2)} a^{(2)} = \underset{1\times 3}{\begin{bmatrix} 1 & -0.8 & 0.6 \end{bmatrix}} \underset{3\times 1}{\begin{bmatrix} +1 \\ 1 + 5.1x_1 + 2.3x_2 \\ 1 - 1.5x_1 + 3.7x_2 \end{bmatrix}} = 1 - 0.8(1 + 5.1x_1 + 2.3x_2) + 0.6(1 - 1.5x_1 + 3.7x_2) = h_\theta(x)_2$$

$$\Rightarrow h_\theta(x)_1 = h_\theta(x)_2$$

# Week 5

## — Neural Networks: Learning (Cost function)




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

$L$ = Total number of layers in Network

$S_L$ = no. of units (not counting bias unit) in layer $L$.

| Binary Classification | Multi-class classification (K classes) |
|---|---|
| $y=0$ or $1$ | $y\in\mathbb{R}^K$ E.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix},\begin{bmatrix}0\\1\\0\\0\end{bmatrix},\begin{bmatrix}0\\0\\1\\0\end{bmatrix},\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$ |
| 1 output unit | K output units |
| $h_\Theta(x)\in\mathbb{R}$ | |
| $S_L=1$, $K=1$ | $h_\Theta(x)\in\mathbb{R}^K$ $S_L=K$ $(K\geq 3)$ |

Pedestrian Car motorcycle truck

### 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

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

## — Neural Networks: Learning (Backpropagation Algorithm)

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

$\min_\Theta J(\Theta)$ $\quad\Theta_{ij}^{(l)}\in\mathbb{R}$

We need code to compute: $J(\Theta)$, $\frac{\partial}{\partial\Theta_{ij}^{(l)}}J(\Theta)$

### Gradient Computation

Given one training example $(x,y)$:

Forwar Propagation:

$a^{(1)}=x$

$z^{(2)}=\Theta^{(1)}a^{(1)}$

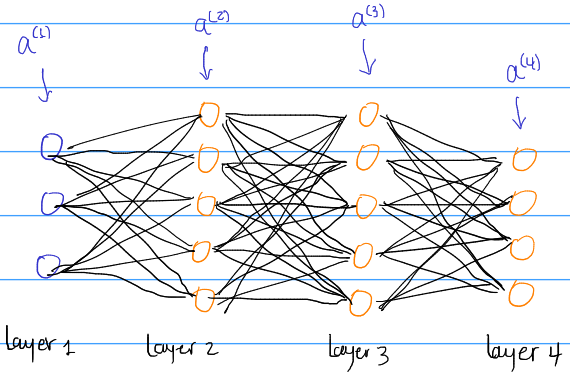
$a^{(2)}=g(z^{(2)})$ (add $a_0^{(2)}$)

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

$a^{(3)}=g(z^{(3)})$ (add $a_0^{(3)}$)

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

$a^{(4)}=h_\Theta(x)=g(z^{(4)})$



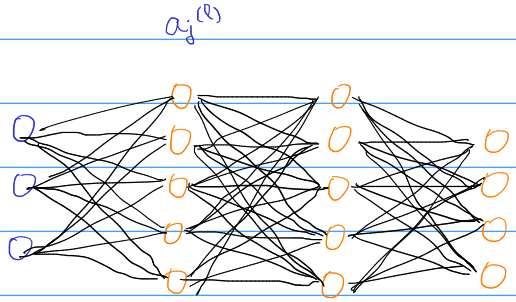
## Gradient Computation: Backpropagation algorithm

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

For each output unit (layer $L=4$)

$\delta_j^{(4)} = a_j^{(4)} - y_j$ $\quad (h_\Theta(x))_j$

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




$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$ $\quad g'(z^{(3)}) = a^{(3)} .* (1 - a^{(3)})$ ?

$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$ $\quad g'(z^{(2)}) = a^{(2)} .* (1 - a^{(2)})$

(No $\delta^{(1)}$)

$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \cdot \delta_i^{(l+1)}$ (ignoring $\lambda$; if $\lambda = 0$)

calculating $g'(z^{(3)})$, let's $z^{(3)} = z$

$g(z^{(3)}) = \frac{1}{1+e^{-z}}$ $\quad g(z^{(3)}) = a^{(3)}$

$g'(z) = -1(1+e^{-z})^{-2} \cdot (-e^{-z})$

$g'(z) = \frac{e^{-z}+1-1}{(1+e^{-z})^2}$ $\quad g(z)$

$g'(z) = \frac{(1+e^{-z})}{(1+e^{-z})^2} - \frac{1}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}} - \frac{1}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right)$

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

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

## Backpropagation algorithm

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

set $\Delta_{ij}^{(l)} = 0$ (for all $l, i, j$) (used to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$)

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

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

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

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

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

$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ $\rightarrow$ (matrix) $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

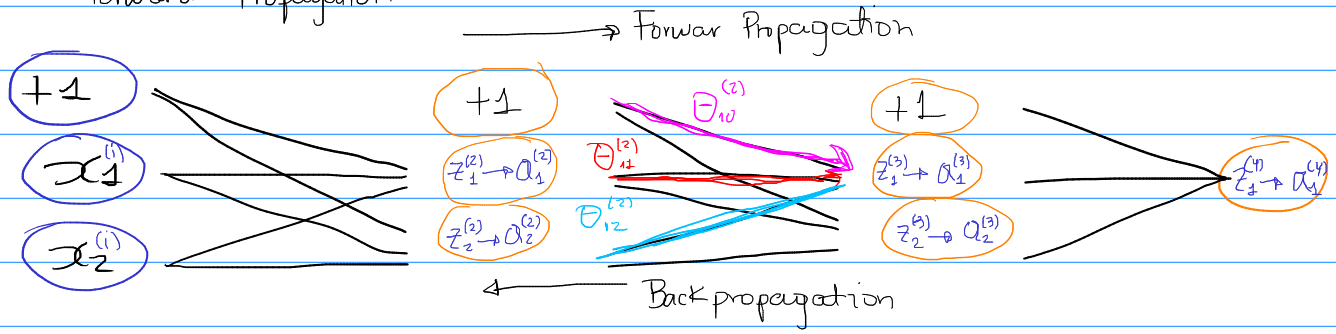
$D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)}$ if $j \neq 0$

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

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

# Neural Networks: Learning (Backpropagation Intuition)

## Forward Propagation




$(x^{(i)}, y^{(i)})$ $\quad z_1^{(3)} = \Theta_{10}^{(2)} \times 1 + \Theta_{11}^{(2)} \times a_1^{(2)} + \Theta_{12}^{(2)} \times a_2^{(2)}$

What is back propagation doing?

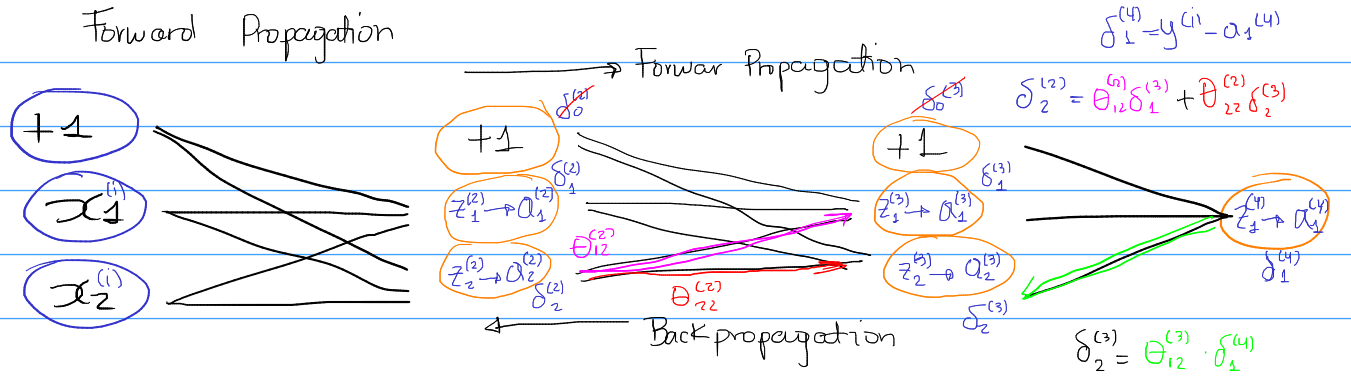
$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log(h_\Theta(x^{(i)})) + (1-y^{(i)}) \log(1-h_\Theta(x^{(i)}))\right] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{S_l} \sum_{j=1}^{S_{l+1}} (\Theta_{ji}^{(l)})^2$$

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?

## Forward Propagation




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

Formally, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(i)$ (for $j \geq 0$), where $\text{cost}(i) = y^{(i)} \log h_\Theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\Theta(x^{(i)}))$

## Neural Networks: Learning (Implementation note: Unrolling parameters)

Example

$s_1 = 10$, $s_2 = 10$, $s_3 = 1$

$\Theta^{(1)} \in \mathbb{R}^{10 \times 11}$, $\Theta^{(2)} \in \mathbb{R}^{10 \times 11}$, $\Theta^{(3)} \in \mathbb{R}^{1 \times 11}$

$D^{(1)} \in \mathbb{R}^{10 \times 11}$, $D^{(2)} \in \mathbb{R}^{10 \times 11}$, $D^{(3)} \in \mathbb{R}^{1 \times 11}$

Code in Octave.

To unroll into vectors.

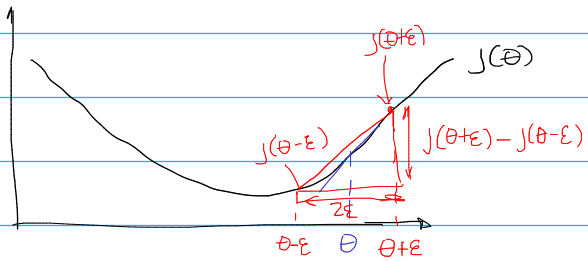
```
thetavec = [Theta1(:); Theta2(:); Theta3(:)]
DVec = [D1(:); D2(:); D3(:)]
```

To get back to matrix representation

```
Theta1 = reshape(thetavec(1:110), 10, 11);
Theta2 = reshape(thetavec(111:220), 10, 11);
Theta3 = reshape(thetavec(221:231), 1, 11);
```

## Neural Networks: Learning (Gradient Checking)

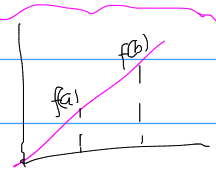
Numerical estimation of gradients




$\theta \in \mathbb{R}$

$$\frac{d}{d\theta} J(\theta) \approx \frac{J(\theta+\varepsilon) - J(\theta-\varepsilon)}{2\varepsilon} \qquad \varepsilon = 10^{-4}$$

$$\frac{J(\theta+\varepsilon) - J(\theta)}{\varepsilon}$$




$y = mx + c$

the slope is $y' = m$

or $m = \frac{f(b) - f(a)}{b - a}$

where $f(b) = mb + c$

$f(a) = ma + c$

$m = \frac{mb + c - ma - c}{b - a}$

$m = \frac{m(b-a)}{b-a} = m$

Implement: gradApprox = (J(theta + EPSILON) − J(Theta − EPSILON))/(2*EPSILON))

Parameter vector $\theta$

~~$\theta \in \mathbb{R}^n$~~ (E.g. $\theta$ is "unrolled" version of $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$)

$\theta = \theta_1, \theta_2, \theta_3, \dots, \theta_n$

$$\frac{\partial}{\partial \theta_1} J(\theta) \approx \frac{J(\theta_1 + \varepsilon, \theta_2, \theta_3, \dots, \theta_n) - J(\theta_1 - \varepsilon, \theta_2, \theta_3, \dots, \theta_n)}{2\varepsilon}$$

$$\frac{\partial}{\partial \theta_2} J(\theta) = \frac{J(\theta_1, \theta_2 + \varepsilon, \theta_3, \dots, \theta_n) - J(\theta_1, \theta_2 - \varepsilon, \theta_3, \dots, \theta_n)}{2\varepsilon}$$

$$\vdots$$

$$\frac{\partial}{\partial \theta_n} J(\theta) = \frac{J(\theta_1, \theta_2, \theta_3, \dots, \theta_n + \varepsilon) - J(\theta_1, \theta_2, \theta_3, \dots, \theta_n - \varepsilon)}{2\varepsilon}$$

Check that gradApprox ≈ DVec. (DVec: From backprop.)

```
for i = 1:n,
    thetaplus = theta;
    thetaplus(i) = thetaplus(i) + Epsilon
    thetaminus = theta;
    thetaminus(i) = thetaminus(i) - Epsilon;
    gradapprox(i) = (J(thetaplus) - J(thetaminus))/(2*Epsilon);
end.
```

→ code

Implementation Note:

- Implement backprop to compute Dvec (unrolled $D^{(1)}, D^{(2)}, D^{(3)}$)
- Implement numerical gradient check to compute gradApprox
- make sure they give similar values
- Turn off gradient checking. Using backpropagation code for learning

Important:

- Be sure to disable your gradient checking code before training your classifier. If you run numerical gradient computation on every iteration of gradient descent (or in the inner loop of costfunction(...)) your code will be very slow.

## Neural Networks: Learning (Random initialization)

### Initial value of $\Theta$

For gradient descent and advanced optimization method, need initial value of $\Theta$

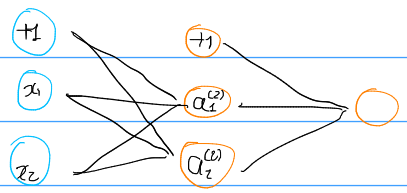
optTheta = fminunc(@costFunction, initialTheta, options)

Consider gradient descent

Set initialTheta = zeros(n,1) ?

### Zero initialization

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




$a_1^{(2)} = a_2^{(2)}$, Also $\delta_1^{(2)} = \delta_2^{(2)}$

$\frac{\partial}{\partial \Theta_{01}^{(1)}} J(\Theta) = \frac{\partial}{\partial \Theta_{02}^{(1)}} J(\Theta)$ $\quad \Theta_{01}^{(1)} = \Theta_{02}^{(1)}$

After each update, parameters corresponding to inputs going into each of two hidden units are identical.

### Random Initialization: Symmetry breaking

Initialize each $\Theta_{ij}^{(l)}$ to a random value in $[-\varepsilon, \varepsilon]$ (i.e. $-\varepsilon \le \Theta_{ij}^{(l)} \le \varepsilon$) → different for the one used in gradient checking

E.g.

Theta1 = rand(10,11) * (2 * INIT_EPSILON) − INIT_EPSILON

Theta2 = rand(1,11) * (2 * INIT_EPSILON) − INIT_EPSILON

x ∈ [0,1]

$x \cdot (2\varepsilon) - \varepsilon = \varepsilon(2x - 1)$

[0,2] → [-1,1] → $[-\varepsilon, \varepsilon]$

## Neural Networks: Learning (Putting it together)

### Training a neural network.

Pick a network architecture (connectivity pattern between neurons)

No. of input units: Dimension of features $x^{(i)}$

No. of 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)

1. Randomly initialize weights
2. Implement forward propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$
3. Implement code to compute cost function $J(\Theta)$
4. Implement backprop to compute partial derivative $\frac{\partial}{\partial \Theta^{(l)}_{jk}} J(\Theta)$

for $i=1:m$ {

Perform forwar propagation and backpropagation using example $(x^{(i)}, y^{(i)})$

(Get activations $a^{(l)}$ and delta terms $\delta^{(l)}$ for $l=2,\dots,L$)

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

}

compute $\frac{\partial}{\partial \Theta^{(l)}_{jk}} J(\Theta)$

5. Use gradient checking to compare $\frac{\partial}{\partial \Theta^{(l)}_{jk}} J(\Theta)$ computed using backpropagation vs. using numerical estimate of gradient of $J(\Theta)$. Then disable gradient checking code.

6. Use gradient descent or advanced optimization method with backpropagation to try to minimize $J(\Theta)$ as a function of parameters $\Theta$. $J(\Theta)$ – non. convex

# Quiz Neural Networks: Learning

(1) $\Delta_{ij}^{(2)} := \Delta_{ij}^{(2)} + \delta_i^{(3)} * (a^{(2)})_j$ $\quad\quad$ $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ → $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

$\Delta^{(2)} = \Delta^{(2)} + \delta^{(3)} * (a^{(2)})^T$

where $l$ = layer
$i$ = training example
$j$ = $j$ unit in layer.

(2) $Theta1 \in \mathbb{R}^{5\times 3}$ (15), $Theta2 \in \mathbb{R}^{4\times 6}$ (24), $15+24=39$.

$thetavec = [thetavec(1:15); thetavec(16:39)]$

⇒ $Theta1 = reshape(thetavec(1:15), 5, 3)$

⇒ $Theta2 = reshape(thetavec(16:39), 4, 6)$

(3) $J(\theta) = 2\theta^4 + 2$ and $\theta = 1$ and $\epsilon = 0{,}01$

$$\frac{\partial}{\partial\theta} J(\theta) \approx \frac{J(\theta+\epsilon) - J(\theta-\epsilon)}{2\epsilon} = \frac{J(1{,}01) - J(0{,}99)}{2 \cdot 0{,}01} = \frac{0.160016}{2 \cdot 0.01} = 8.0008.$$

(4) - Using gradient checking can help verify if one's implementation of backpropagation is bug-free

- For computational efficiency, after we have performed gradient checking to verify that our backpropagation code is correct, we usually disable gradient checking before using backpropagation to train the network.

(5) - Suppose you are training a neural network using gradient descent. Depending on your random initialization, your algorithm may converge to different local optima...

- If we are training a neural network using gradient descent, one reasonable "debugging" step to make sure it is working is to plot $J(\theta)$ as a function of the number of iteration, and make sure it is decreasing ...