DS4023 Machine Learning

Lecture 4:
Neural Networks

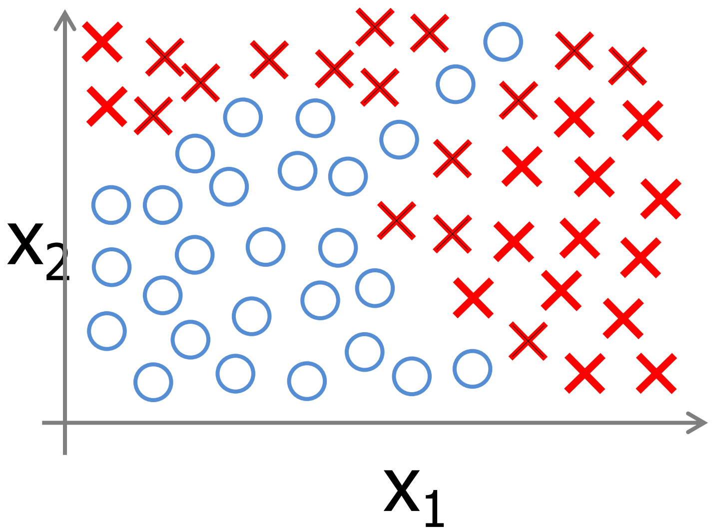
Mathematical Sciences
United International College

Reference: Andrew Ng's Machine learning course

Outline

- Non-linear hypothesis
- Model representation
- Multiclass classification
- Cost function
- Error Backpropagation algorithm

Non-linear hypothesis



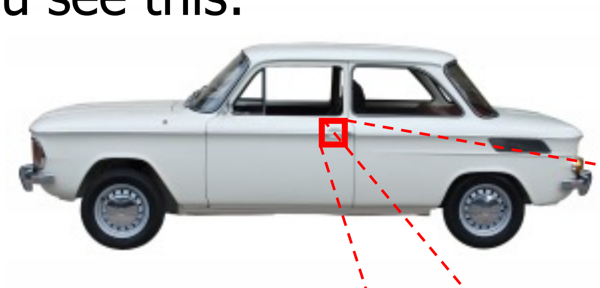
$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \theta_6 x_1 x_2^2 + \ldots)$$

$x_1 =$ size
$x_2 =$ # bedrooms
$x_3 =$ # floors
$x_4 =$ age
$\ldots$
x_{100}

Non-linear hypothesis

What is this?

You see this:

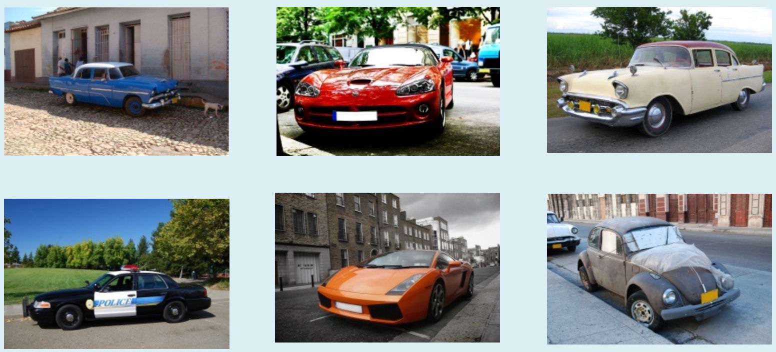


But the camera sees this:

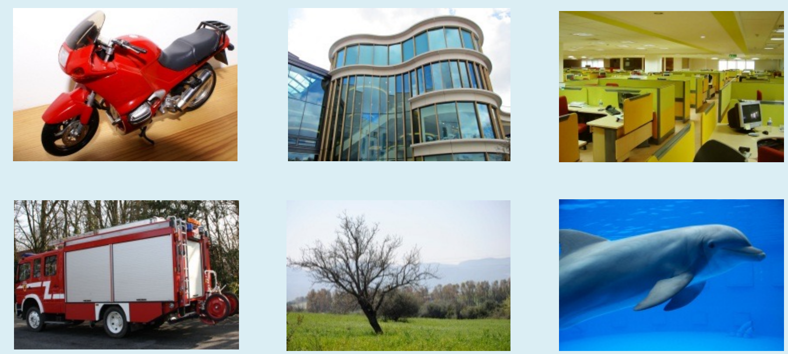
194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50

Non-linear hypothesis

Computer Vision: Car detection



Cars

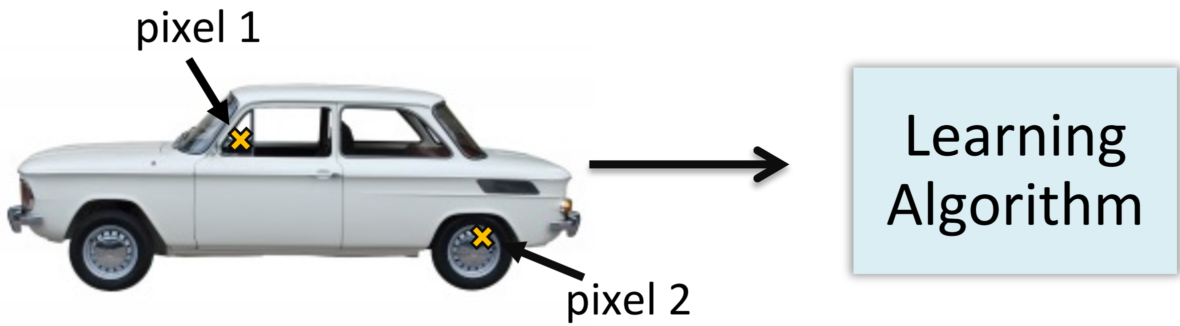


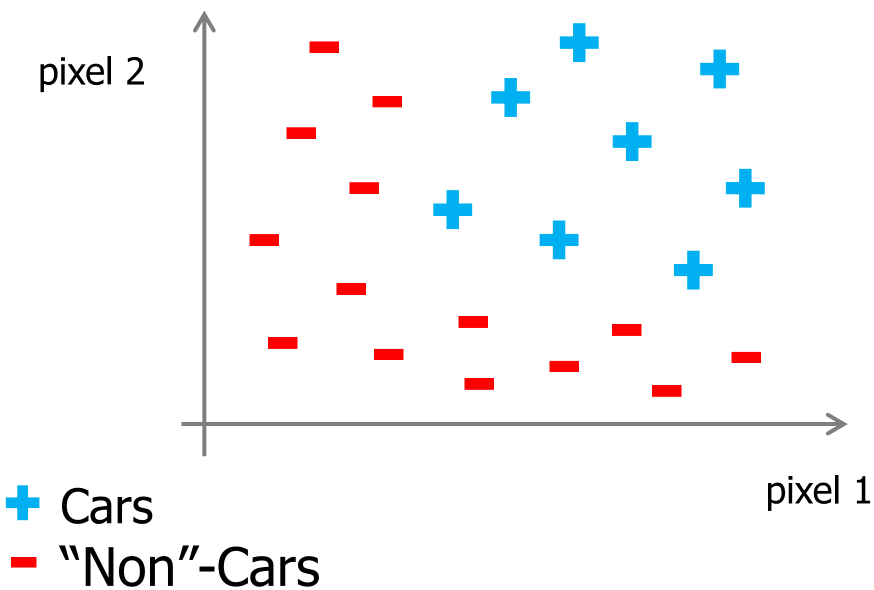
Not a car

Testing:

What is this?

Non-linear hypothesis





50 x 50 pixel images→ 2500 pixels

$n = 2500$ (7500 if RGB)

$$x = \begin{bmatrix} \text{pixel 1 intensity} \\ \text{pixel 2 intensity} \\ \vdots \\ \text{pixel 2500 intensity} \end{bmatrix}$$

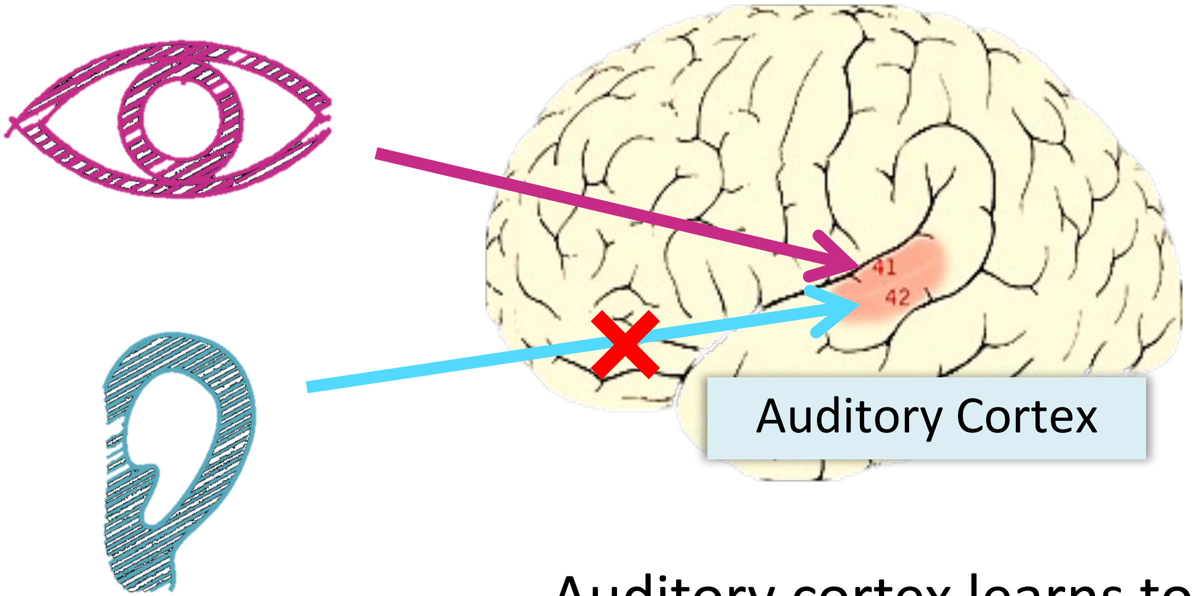
Quadratic features ($x_i \times x_j$): ≈3 million features

Outline

- Non-linear hypothesis
- Model representation
- Multiclass classification
- Cost function
- Error Backpropagation algorithm

Neurons and the brain

The "one learning algorithm" hypothesis

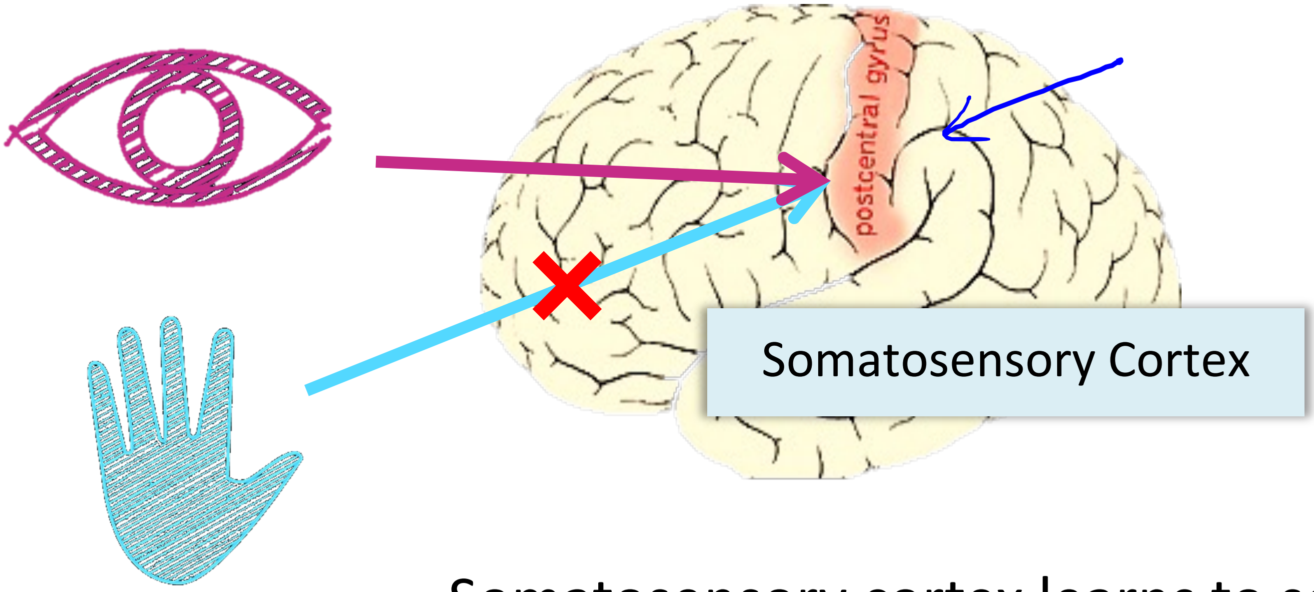


Auditory cortex learns to see

[Roe et al., 1992]

Neurons and the brain

The "one learning algorithm" hypothesis



Somatosensory cortex learns to see

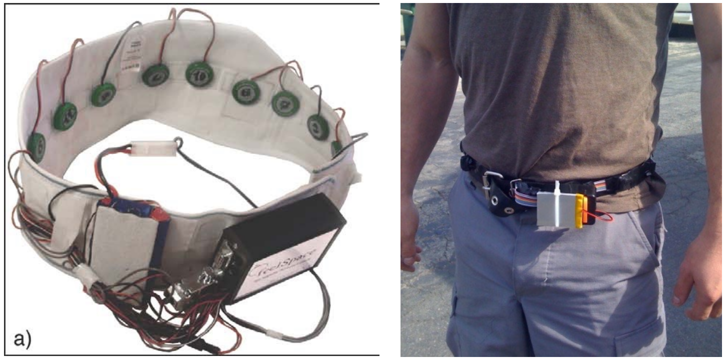
[Metin & Frost, 1989]

Neurons and the brain

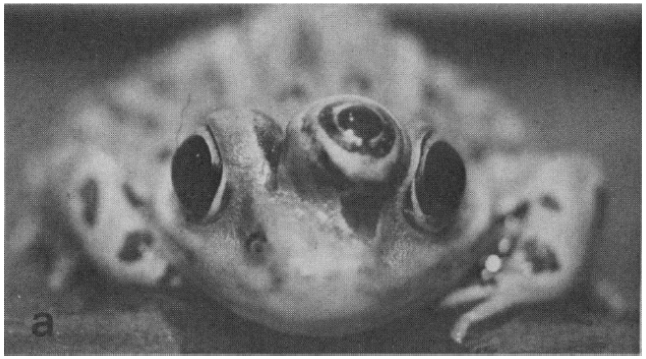
Sensor representations in the brain

Seeing with your tongue

Human echolocation (sonar)



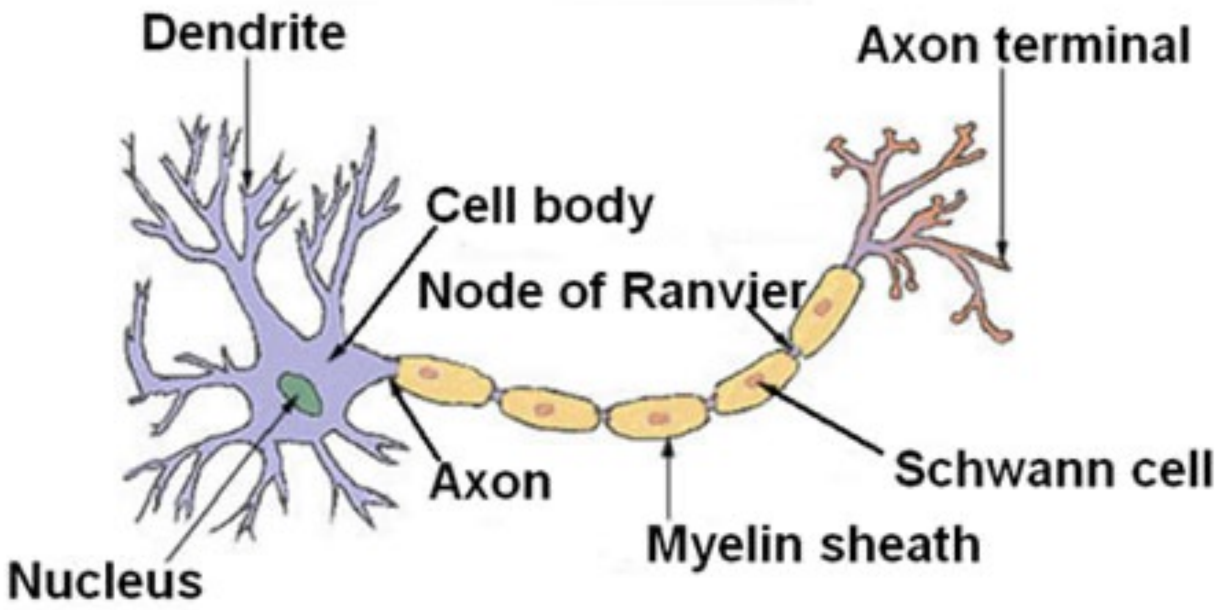
Haptic belt: Direction sense



Implanting a 3rd eye

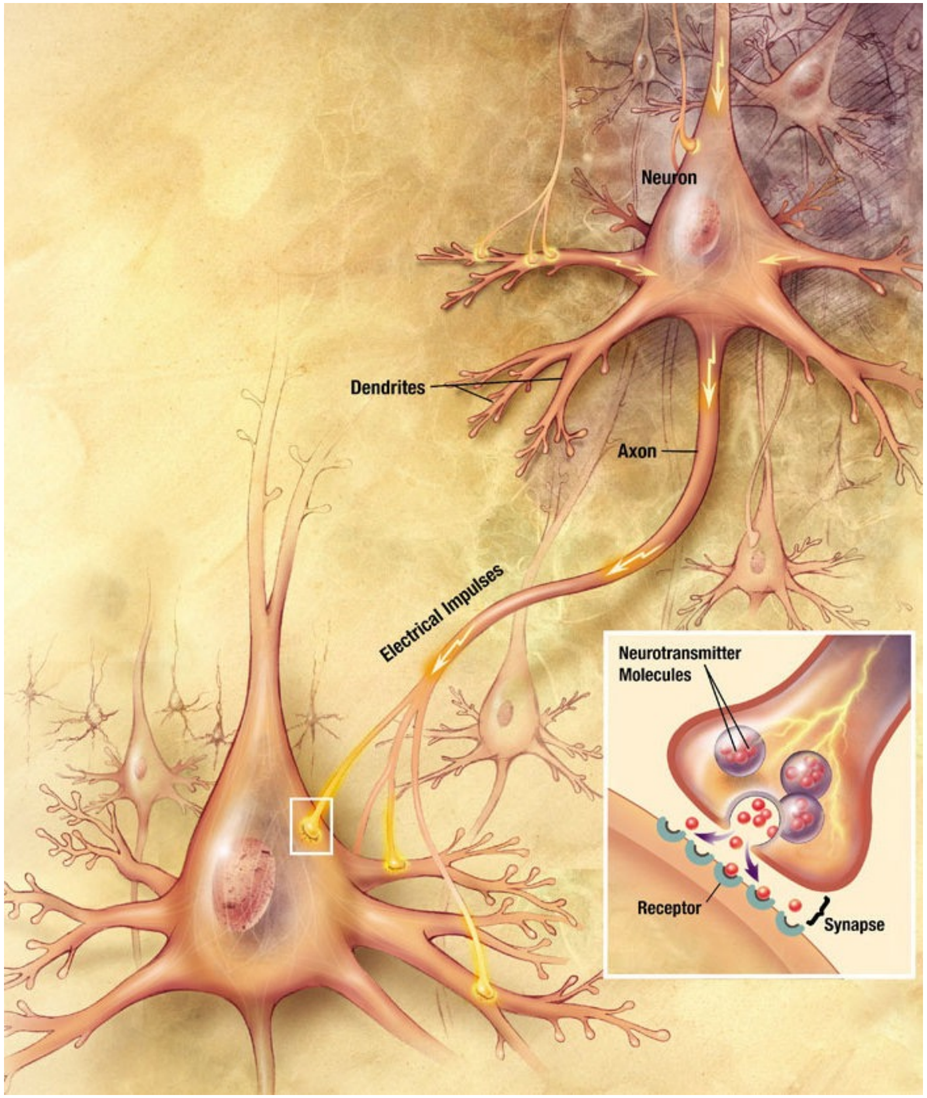
Neurons and the brain

Neuron in the brain



Neurons and the brain

Neurons in the brain



[Credit: US National Institutes of Health, National Institute on Aging]

Neurons and the brain

Neural Networks

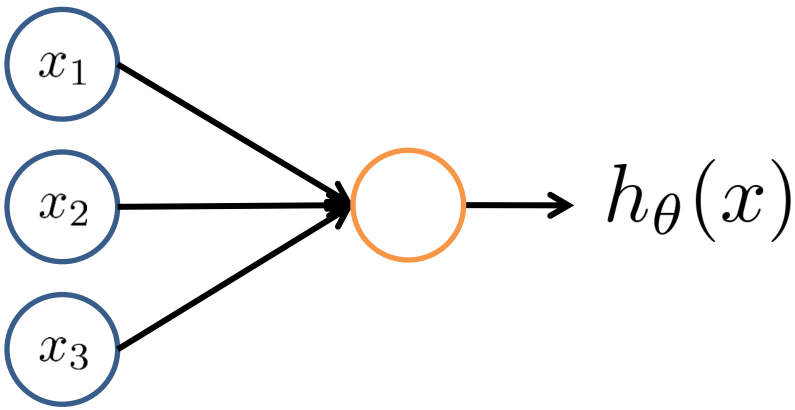
Origins: Algorithms that try to mimic the brain.
Was very widely used in 80s and early 90s; popularity diminished in late 90s.

Recent resurgence: State-of-the-art technique for many applications

Neurons and the brain

Neuron model: Logistic unit

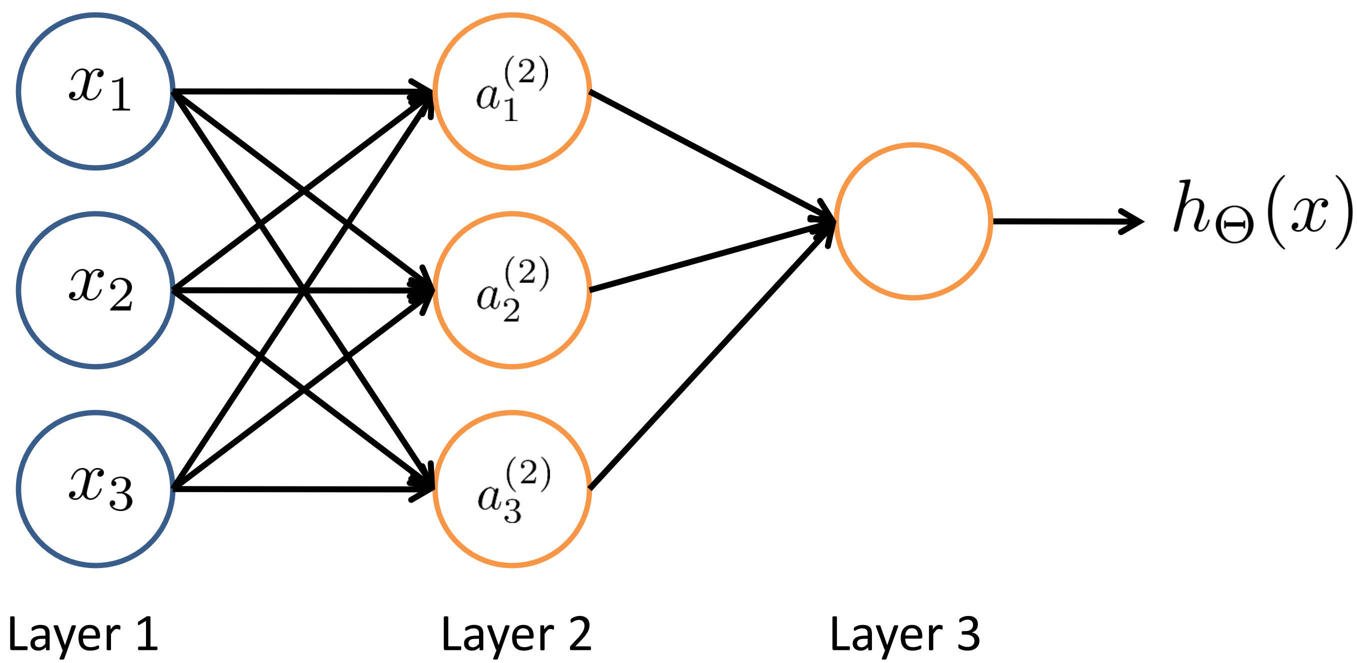
$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$



Sigmoid (logistic) activation function.

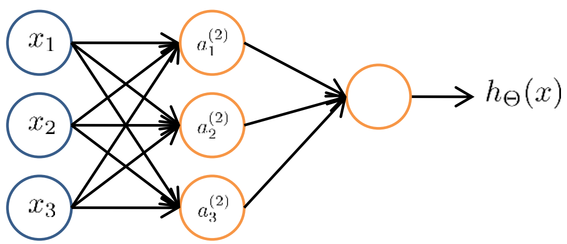
Neurons and the brain

Neural Network



Neurons and the brain

Neural Network



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

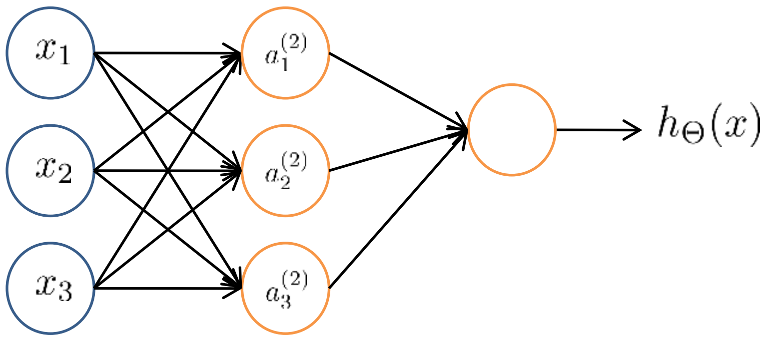
$\Theta^{(j)} =$ matrix of weights controlling function mapping from layer j to layer $j+1$

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

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

Forward propagation: Vectorized implementation



$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$

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

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

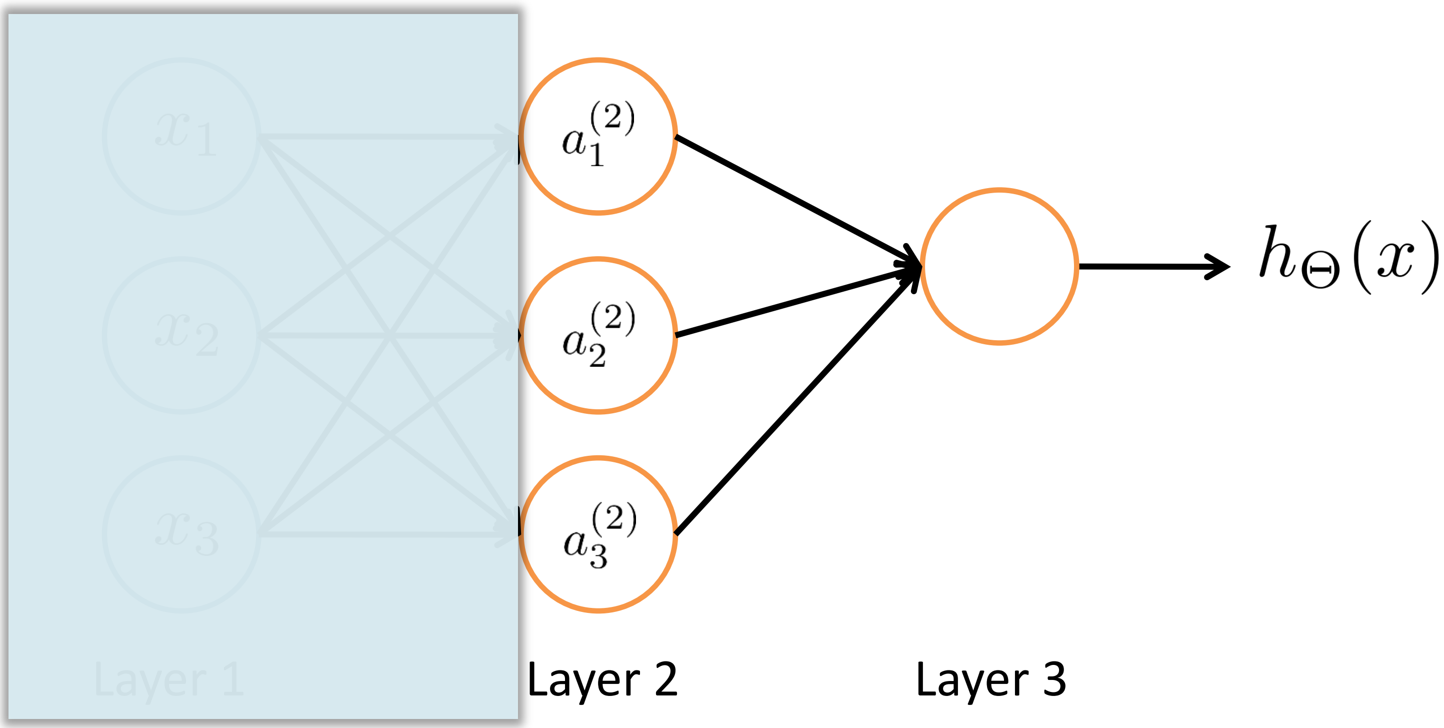
$$z^{(2)} = \Theta^{(1)}x$$
$$a^{(2)} = g(z^{(2)})$$

Add $a_0^{(2)} = 1$.

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

Neural Network

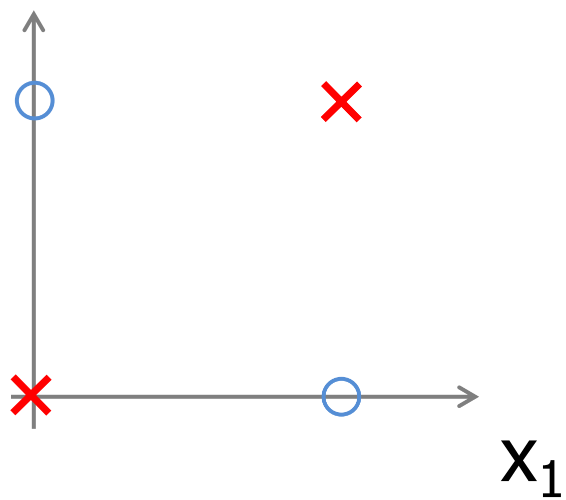
Neural Network learning its own features

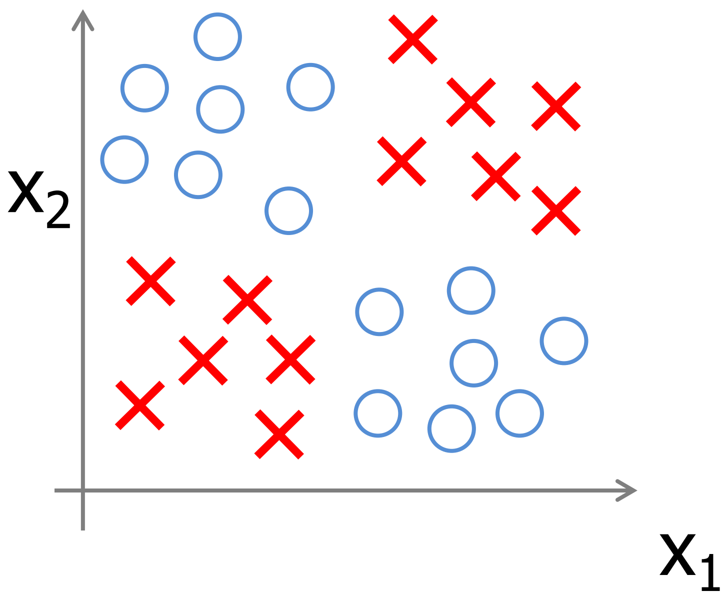


Neural Network

Non-linear classification example: XOR/XNOR

x_1, x_2 are binary (0 or 1).





$$y = x_1 \text{ XOR } x_2$$
$$x_1 \text{ XNOR } x_2$$
$$\text{NOT } (x_1 \text{ XOR } x_2)$$

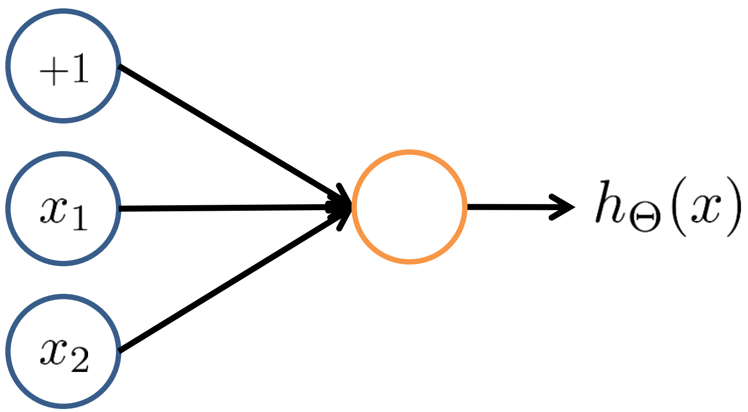
$$x_1 \text{ XNOR } x_2$$
$$= (x_1 \text{ AND } x_2) \text{ OR } ((\text{NOT } x_1) \text{ AND } (\text{NOT } x_2))$$

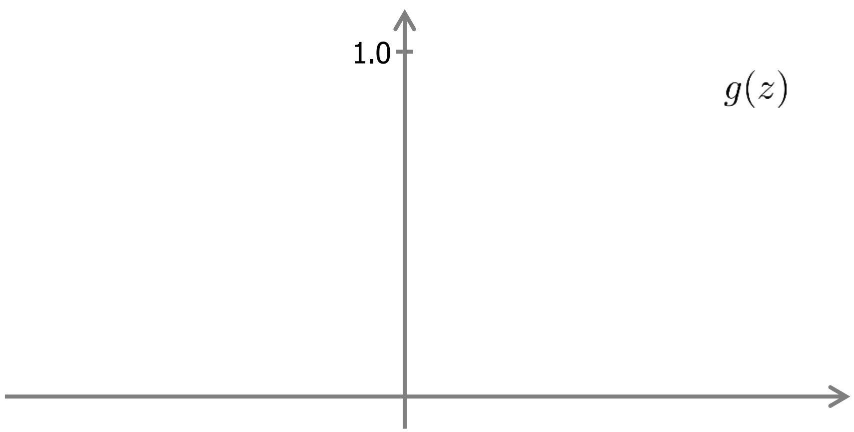
Neural Network

Simple example: AND

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

$y = x_1 \text{ AND } x_2$





The problem becomes the following: can you find a proper $h_\Theta(x)$, so that the function meets the following table:

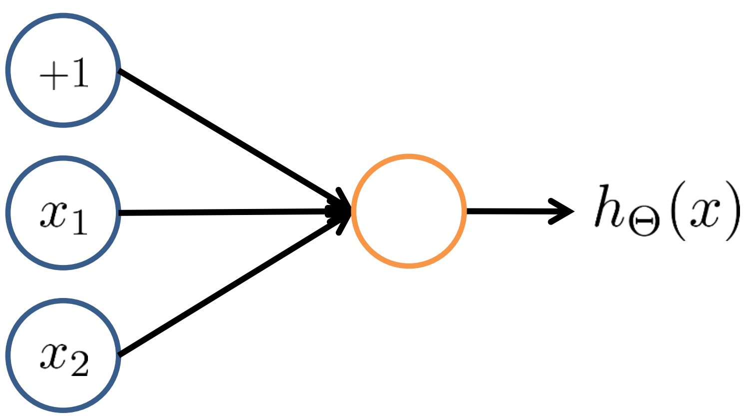
x_1	x_2	$h_\Theta(x)$
0	0	0
0	1	0
1	0	0
1	1	1

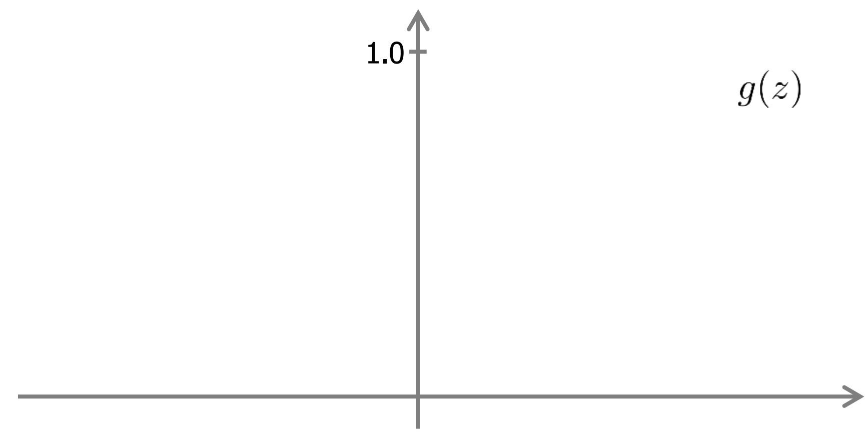
Neural Network

Simple example: OR

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

$x_1 \text{ OR } x_2$





The problem becomes the following: can you find a proper $h_\Theta(x)$, so that the function meets the following table:

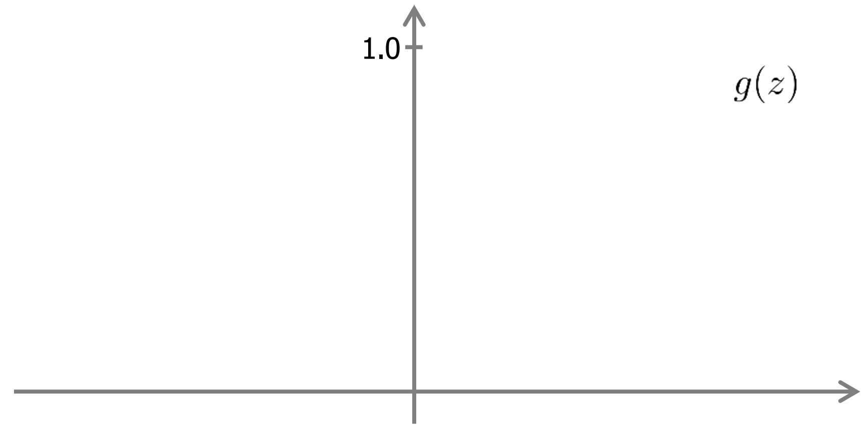
x_1	x_2	$h_\Theta(x)$
0	0	0
0	1	1
1	0	1
1	1	1

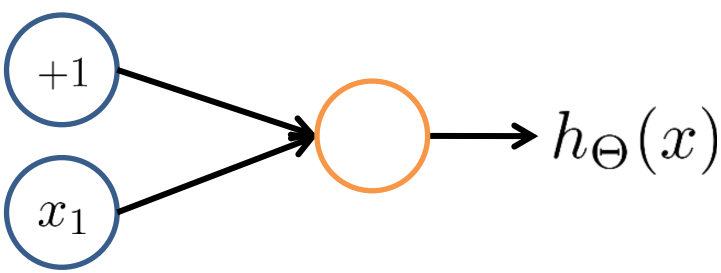
Neural Network

Simple example: NOT

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

$\text{NOT } x_1$





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

The problem becomes the following: can you find a proper $h_\Theta(x)$, so that the function meets the following table:

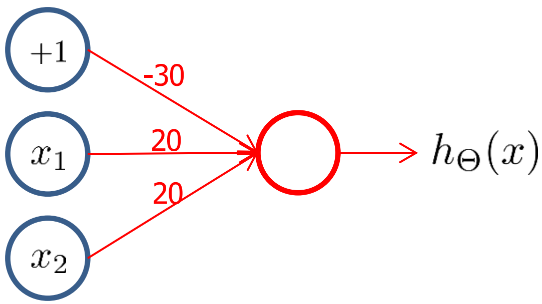
x_1	$h_\Theta(x)$
0	1
1	0

$(\text{NOT } x_1) \text{ AND } (\text{NOT } x_2)$?

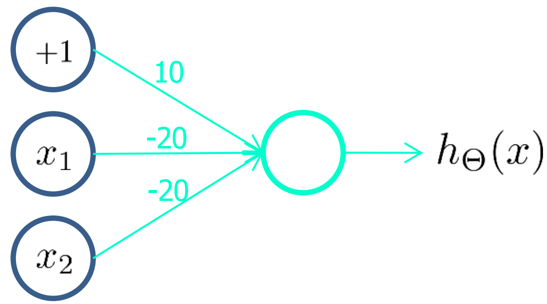
Neural Network

Putting it together: $x_1 \text{ XNOR } x_2$

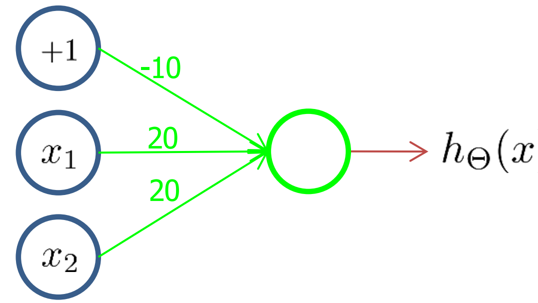
$= (x_1 \text{ AND } x_2)$ OR $((\text{NOT } x_1) \text{ AND } (\text{NOT } x_2))$



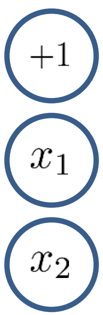
$x_1 \text{ AND } x_2$



$(\text{NOT } x_1) \text{ AND } (\text{NOT } x_2)$



$x_1 \text{ OR } x_2$



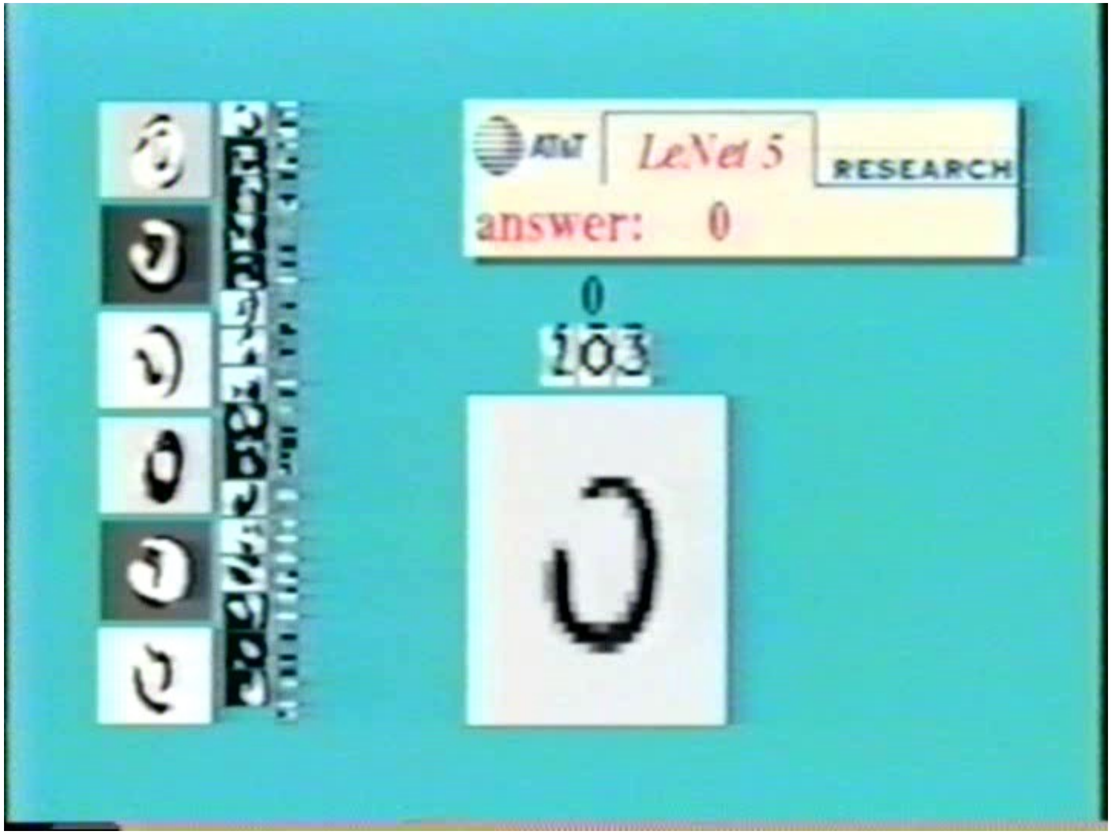
x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

Outline

- Non-linear hypothesis
- Model representation
- Multiclass classification
- Cost function
- Error Backpropagation algorithm

Multi-class classification

Handwritten digit classification



[Courtesy of Yann LeCun]

Multi-class classification

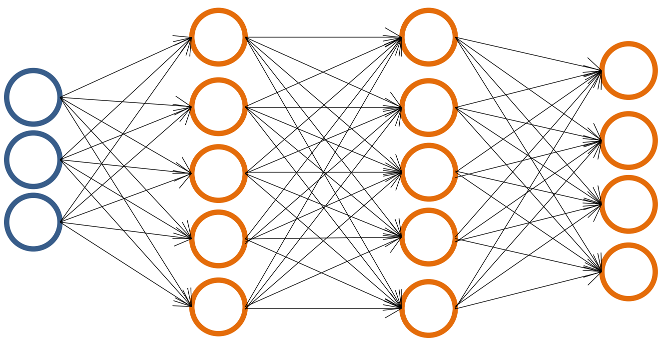
Multiple output units: One-vs-all.

Pedestrian

Car

Motorcycle

Truck



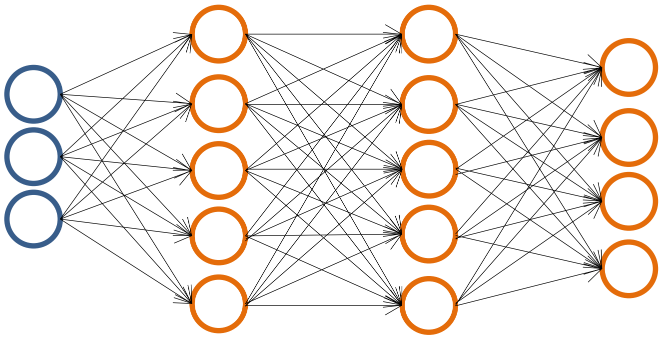
$$h_\Theta(x) \in \mathbb{R}^4$$

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

Multi-class classification

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

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

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

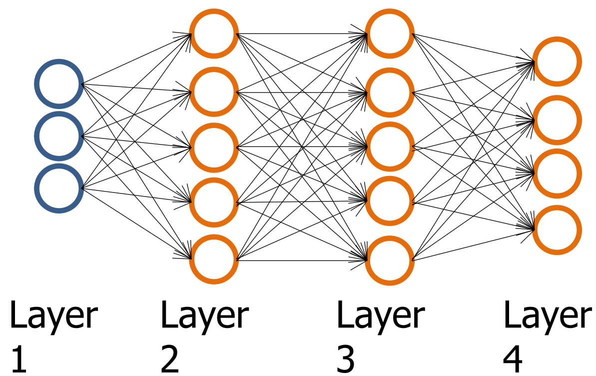
pedestrian car motorcycle truck

Outline

- Non-linear hypothesis
- Model representation
- Multiclass classification
- Cost function
- Error Backpropagation algorithm

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

Binary classification

$y = 0 \text{ or } 1$

1 output unit

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

Cost function

Logistic regression:

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

Neural network:

$h_\Theta(x) \in \mathbb{R}^K$ $\quad (h_\Theta(x))_k = k^{th}\ output\ in\ vector\ R^K$

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

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

Cost function

Gradient computation

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

$$\min_{\Theta} J(\Theta)$$

Need code to compute:

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

Lab Exercise 6

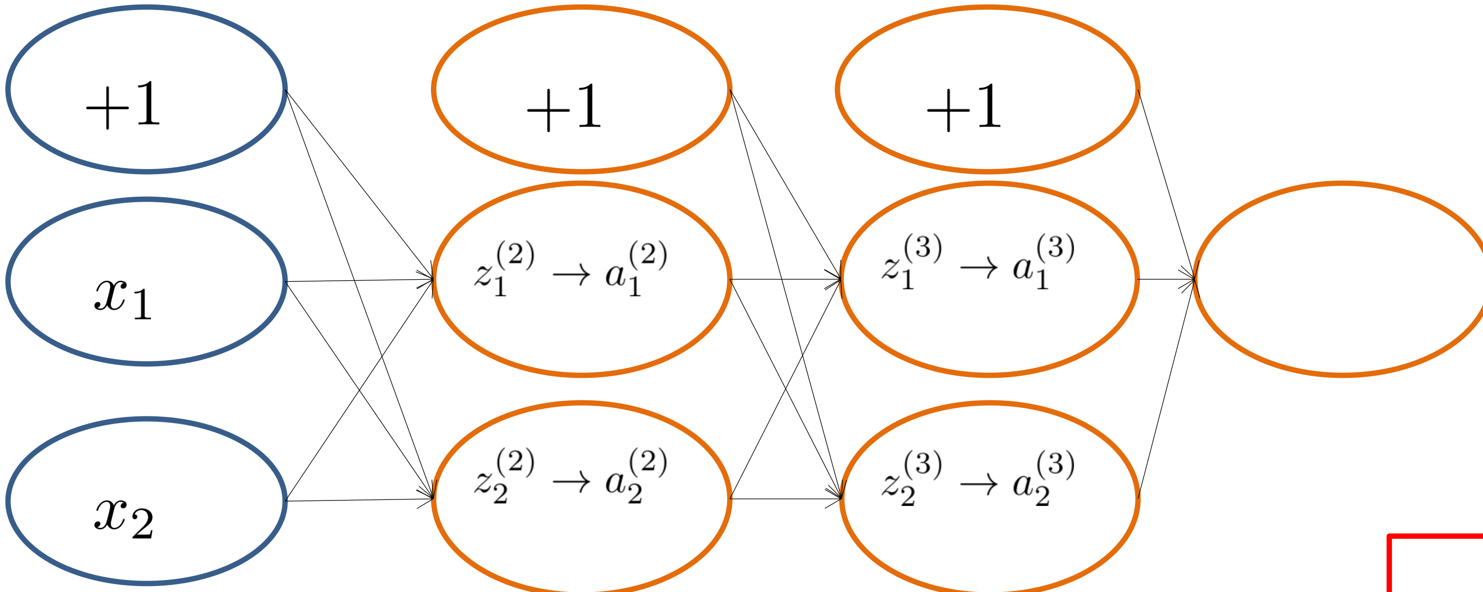
- In this exercise, you will implement a neural network and get to see it work on data.
- Download *NeuralNetworks.ipynb* and ex4D*ata1.mat* and ex4weights.*mat* from iSpace, and finish the code implementation in section 1&2 (section 3&4is for next lab)
- Submit the completed notebook on iSpace.

Outline

- Non-linear hypothesis
- Model representation
- Multiclass classification
- Cost function
- Error Backpropagation algorithm

Error Backpropagation algorithm

Let's review Forward Propagation



δ_j^l ="error" of node j in layer l.

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

?

$$\delta_j^l = \frac{\partial}{\partial z_j^{(l)}} J(\Theta) \qquad j \geq 0$$

Error Backpropagation algorithm

Important formulas for calculating the gradients

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

$$\delta^{(L)} = \frac{\partial}{\partial z^{(L)}} J(\Theta) = \frac{\partial J(\Theta)}{\partial g} \frac{\partial g}{\partial z^{(L)}} = \frac{\partial J(\Theta)}{\partial g} .* \, g\,'(z^L)$$

$$\delta^{(l)} = (\Theta^{(l)})^T \delta^{(l+1)} .* \, g\,'(z^l)$$

Error Backpropagation algorithm

Gradient computation: Backpropagation algorithm

Intuition: δ_j^l ="error" of node j in layer l.

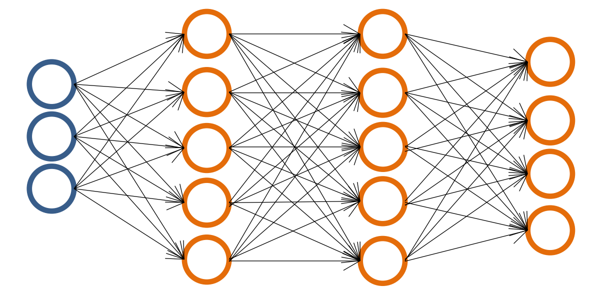
For each output unit (layer L = 4)

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

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

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

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



Error Backpropagation algorithm

Gradient computation: Backpropagation algorithm

For one data sample input x

1. Input x
2. Feedforward: for each layer $l = 2,3,\ldots,L$ compute

   $z^l = \Theta^{l-1}a^{l-1} + \Theta_0^{l-1}$, $a^l = g(z^l)$ where Θ_0^l is the bias in layer $l-1$

3. Error in output layer: $\delta^{(L)} = \frac{\partial J(\Theta)}{\partial g} .* g'(z^L)$
4. Backpropagate the error: $l = L-1, L-2, \ldots, 2$, compute:

   $\delta^{(l)} = (\Theta^{(l)})^T \delta^{(l+1)} .* g'(z^l)$

5. Calculate: the gradient of the cost function

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

Error Backpropagation algorithm

Gradient computation: Backpropagation algorithm

For a whole dataset :$\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

1. Input $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$
2. For each training sample x, perform the following:
    1. Feedforward: for each layer $l = 2,3, \ldots, L$ compute
    $z^{x,l} = \Theta^{l-1} a^{x,l-1} + \Theta_0^{l-1}$, $a^{x,l} = g(z^{x,l})$ where Θ_0^{l-1} is the bias in layer $l-1$
    2. Error in output layer: $\delta^{(x,L)} = \frac{\partial J^x(\Theta)}{\partial g} .* g'(z^{x,L})$
    3. Backpropagate the error: $l = L-1, L-2, \ldots, 2$, compute:$\delta^{(x,l)} = (\Theta^{(l)})^T \delta^{(x,l+1)} .* g'(z^{x,l})$
3. Calculate: the gradient of the cost function

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

4. Use gradient descent to update parameters:

$$\boxed{\Theta_{ij}^{(l)} \rightarrow \Theta_{ij}^{(l)} - \frac{\alpha}{m} \sum_x a_j^{(x,l)} \delta_i^{(x,l+1)},}$$ where $a_j^{(x,l)} = 1, \quad \text{when } j = 0$

Error Backpropagation algorithm

Gradient computation: with regularization terms

For a whole dataset : $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

1. Repeat steps 1-3 in the previous slides to calculate the gradients without regularization terms
2. Add the regularization term $\frac{\lambda}{m}\Theta_{ij}^{(l)}$ to the gradient
3. Use gradient descent to update parameters:

$\boxed{\Theta_{ij}^{(l)} \rightarrow \Theta_{ij}^{(l)} - \frac{\alpha}{m}\sum_x a_j^{(x,l)}\delta_i^{(x,l+1)} - \frac{\alpha\lambda}{m}\Theta_{ij}^{(l)}}$, where $a_j^{(x,l)} = 1$, when $j = 0$

Error Backpropagation algorithm

Proof of the equations:

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

Proof :

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

by the chain rule in calculus for multivariate functions

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

$$\frac{\partial z_k^{(l+1)}}{\partial \Theta_{ij}^{(l)}} = 0 \text{ if } \mathrm{k} \neq i$$

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

$$z_i^{(l+1)} = \sum_j \Theta_{ij}^{(l)} a_j^{(l)}$$

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

(by definition of $\delta_i^{(l+1)}$)

Error Backpropagation algorithm

Proof of the equations:

$$\delta^{(l)} = (\Theta^{(l)})^T \delta^{(l+1)} .* \, g\,'(z^l)$$

Proof :

$$\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} J(\Theta)$$

$$= \sum_k \frac{\partial J(\Theta)}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}}$$

by the chain rule in calculus
for multivariate functions

$$= \sum_k \delta_k^{(l+1)} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}}$$

(by definition of $\delta_k^{(l+1)}$)

$$z_k^{(l+1)} = \sum_j \Theta_{kj}^{(l)} a_j^{(l)} = \sum_j \Theta_{kj}^{(l)} g(z_j^{(l)})$$

$$= \sum_k \delta_k^{(l+1)} \Theta_{kj}^{(l)} g'(z_j^{(l)})$$

$$= g'(z_j^{(l)}) \sum_k \delta_k^{(l+1)} \Theta_{kj}^{(l)}$$

By vector-ize the above formula $\delta^{(l)} = (\Theta^{(l)})^T \delta^{(l+1)} .* \, g\,'(z^l)$

Error Backpropagation algorithm

Two details about implementing backpropagation

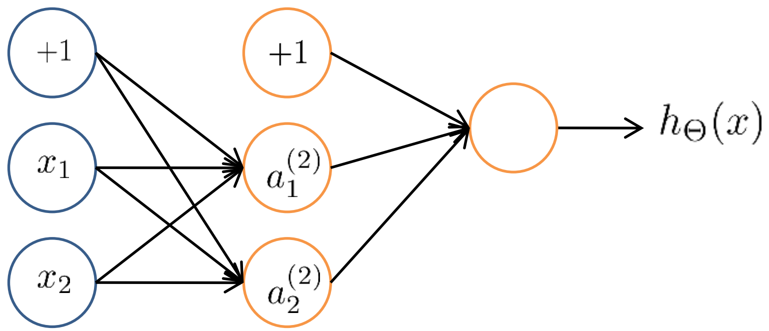
1. Random initialization

2. Gradient checking

Error Backpropagation algorithm

Random initialization

Zero initialization



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

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

$$a_1^{(2)} = a_2^{(2)}$$

Error Backpropagation algorithm

Random initialization: Symmetry breaking

Initialize each $\Theta_{ij}^{(l)}$ to a random value in $[-\epsilon, \epsilon]$
(i.e. $-\epsilon \leq \Theta_{ij}^{(l)} \leq \epsilon$)

E.g.

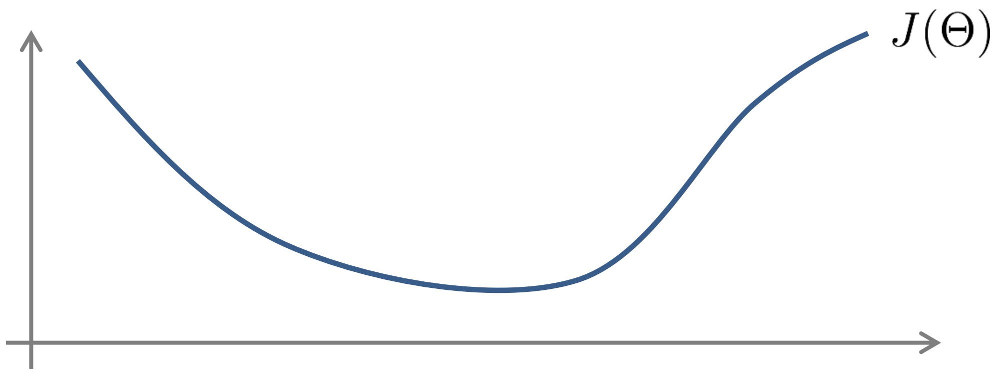
```
Theta1 =  rand(10,11)*(2*INIT_EPSILON)
          - INIT_EPSILON;

Theta2 =  rand(1,11)*(2*INIT_EPSILON)
          - INIT_EPSILON;
```

Error Backpropagation algorithm

Gradient checking

Numerical estimation of gradients



Implement: `gradApprox = (J(theta + EPSILON) – J(theta – EPSILON)) /(2*EPSILON)`

Error Backpropagation algorithm

Gradient checking

Parameter vector θ

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

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

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

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

$$\vdots$$

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

Error Backpropagation algorithm

Gradient checking

```
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;
```

Check that

`gradApprox` ≈ `Gradient that you compute with back prop`

Error Backpropagation algorithm

Gradient checking

Implementation Note:

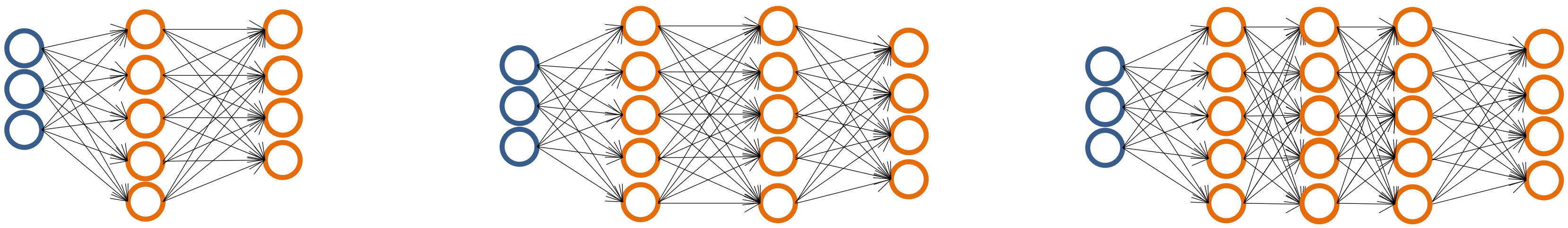
- Implement backprop to compute **gradients**
- Implement numerical gradient check to compute **gradApprox**.
- Make sure they give similar values.
- Turn off gradient checking. Using backprop code for learning.

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 <u>very</u> slow.

Summary

Training a neural network

Pick a network architecture (connectivity pattern between neurons)



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

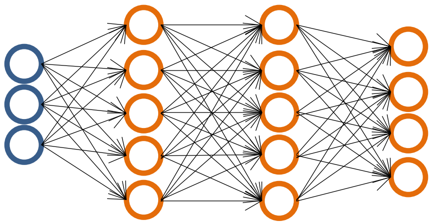
Summary

Training a neural network

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

```
for i = 1:m
```

Perform forward propagation and backpropagation using example $(x^{(i)}, y^{(i)})$
(Get activations $a^{(l)}$ and delta terms $\delta^{(l)}$ for $l = 2, \ldots, L$).



Lab Exercise 7

- In this exercise, you will implement a neural network and get to see it work on data.
- Download *NeuralNetworks.ipynb* and ex4D*ata1.mat* and ex4weights.*mat* from iSpace, and finish the code implementation in section 3&4
- Submit the completed notebook on iSpace.