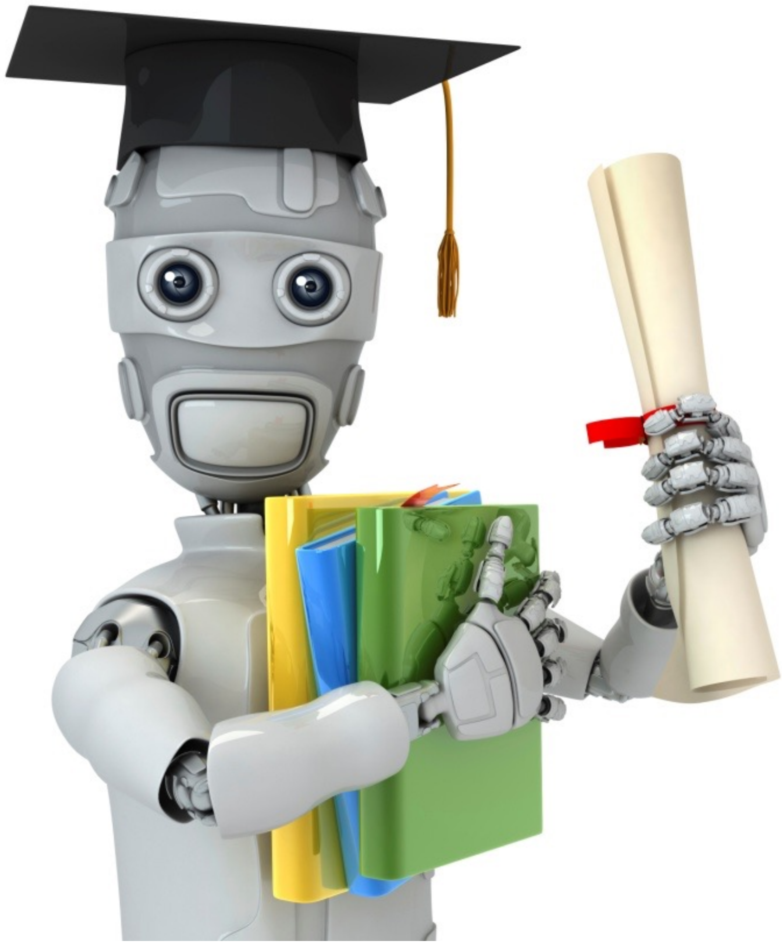
Machine Learning

Lecture 4: Neural Networks[1]

Mohamad GHASSANY

EFREI Paris

1: Slides of this lecture are from Andrew Ng's machine learning course

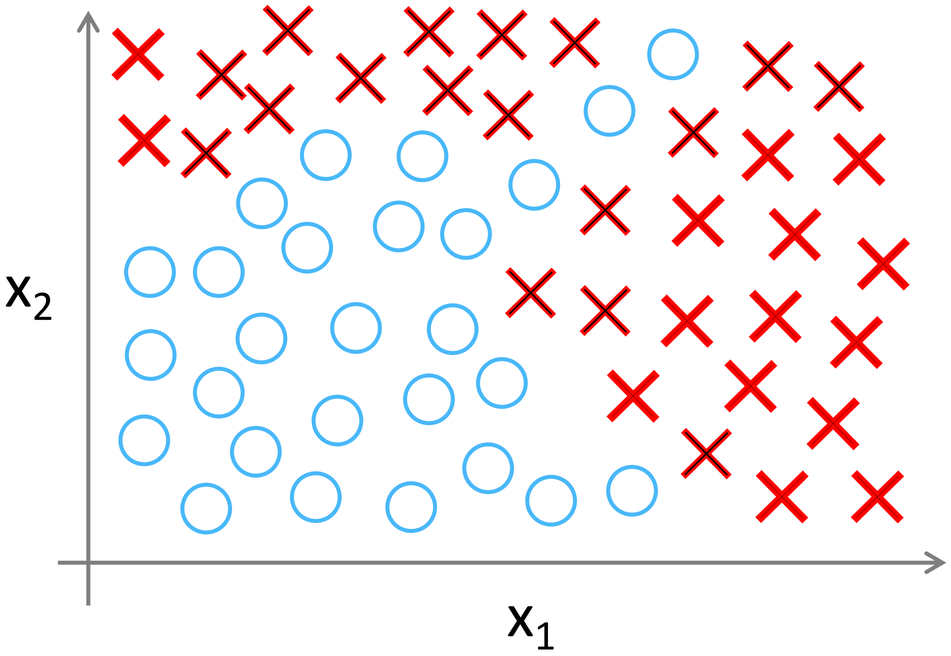


Machine Learning

Neural Networks: Representation

Non-linear hypotheses

Non-linear Classification



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

x_1 = size

x_2 = # bedrooms

x_3 = # floors

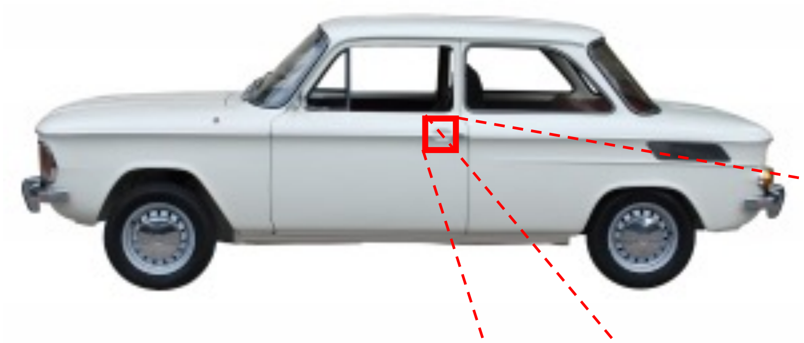
x_4 = age

$\ldots$

x_{100}

What is this?

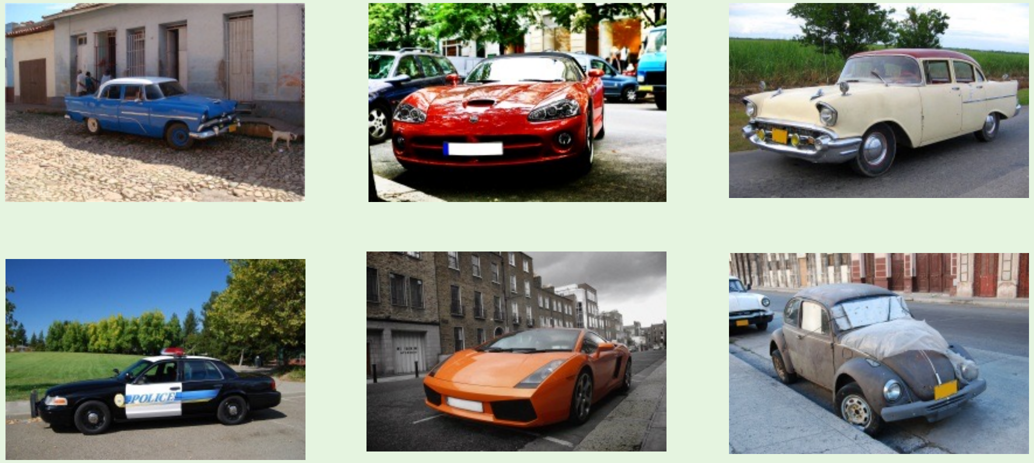
You see this:



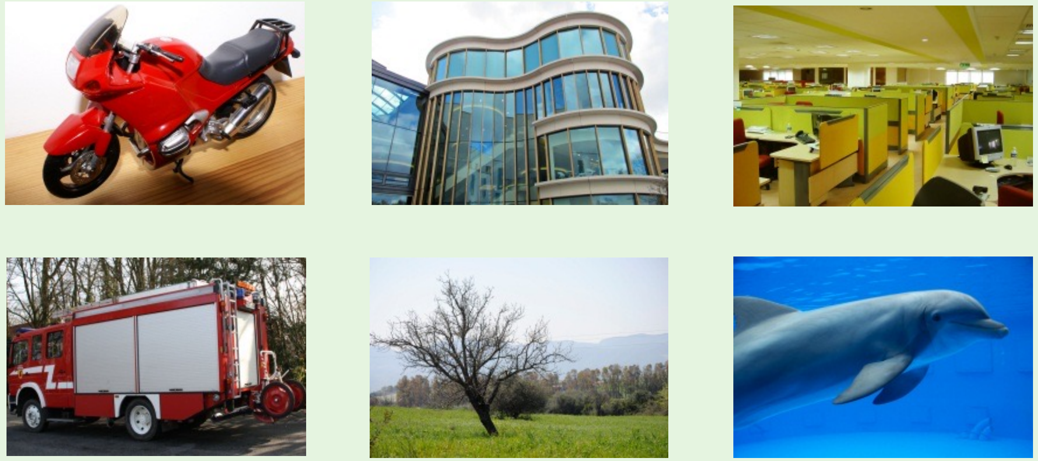
But the camera sees this:

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

Computer Vision: Car detection



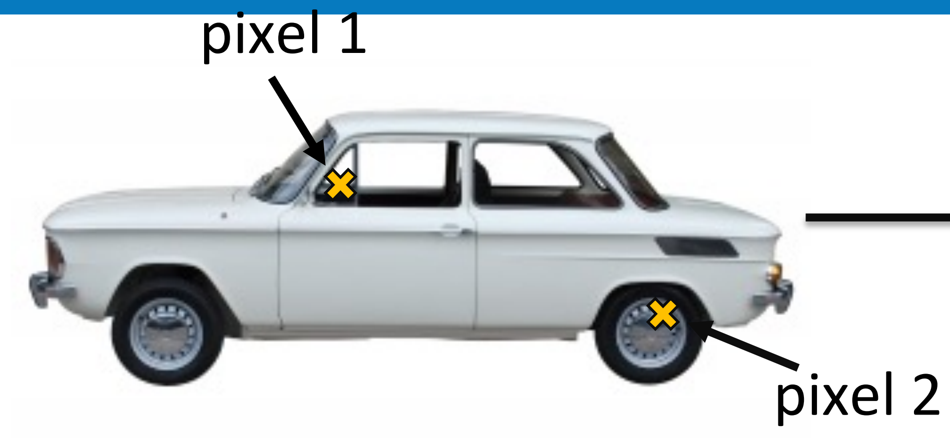
Cars



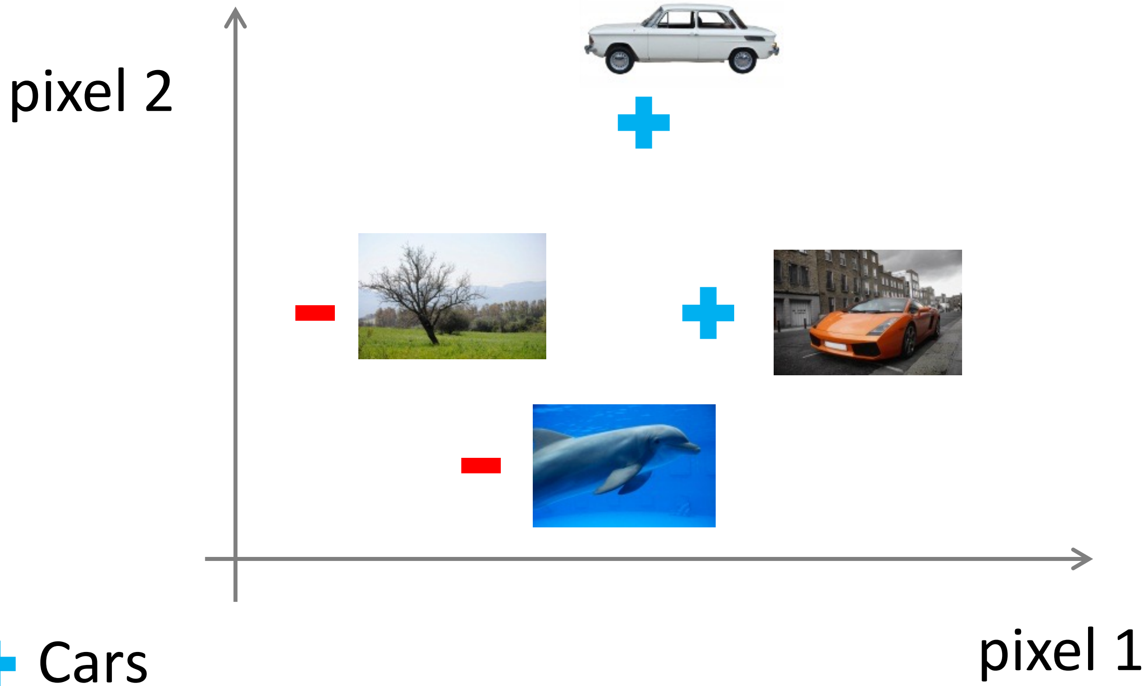
Not a car

Testing:

What is this?

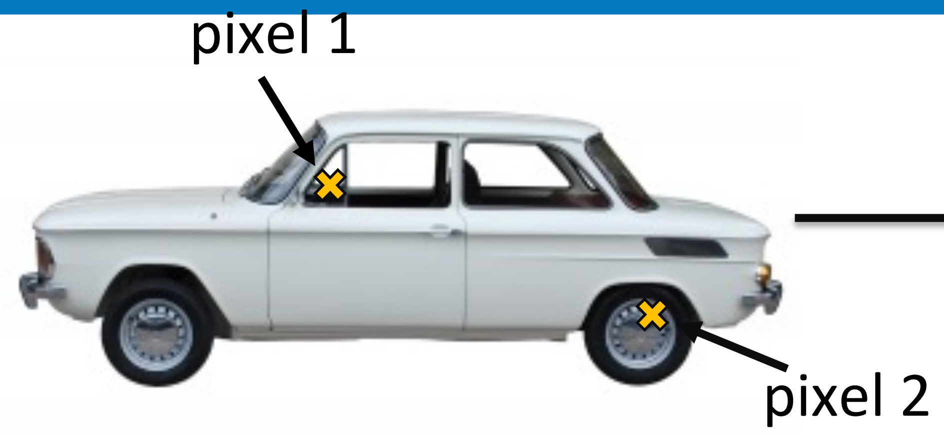


Learning Algorithm

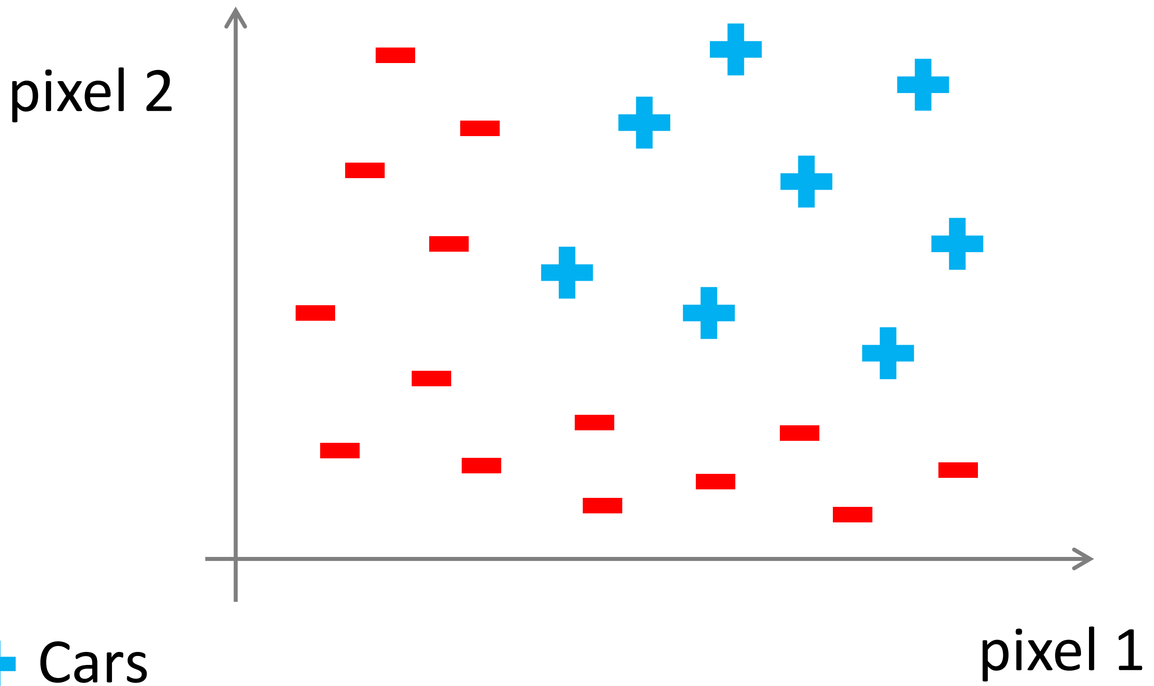


+ Cars

− “Non”-Cars



Learning Algorithm



+ Cars

– “Non”-Cars

50 x 50 pixel images→ 2500 pixels

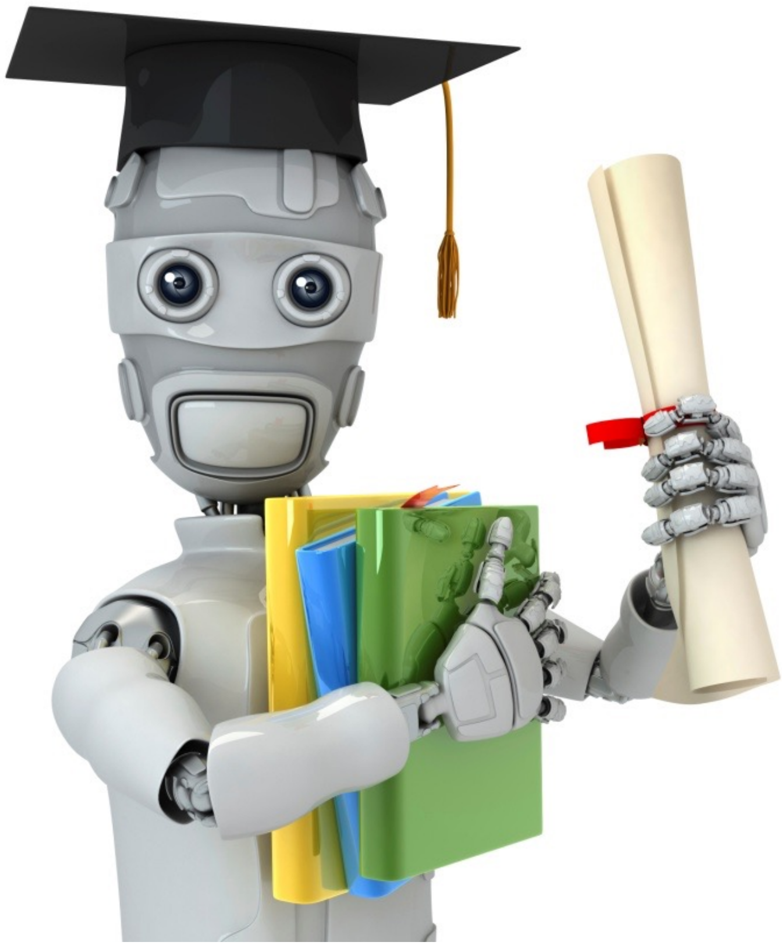
$n = 2500$ (7500 if RGB)

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

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

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

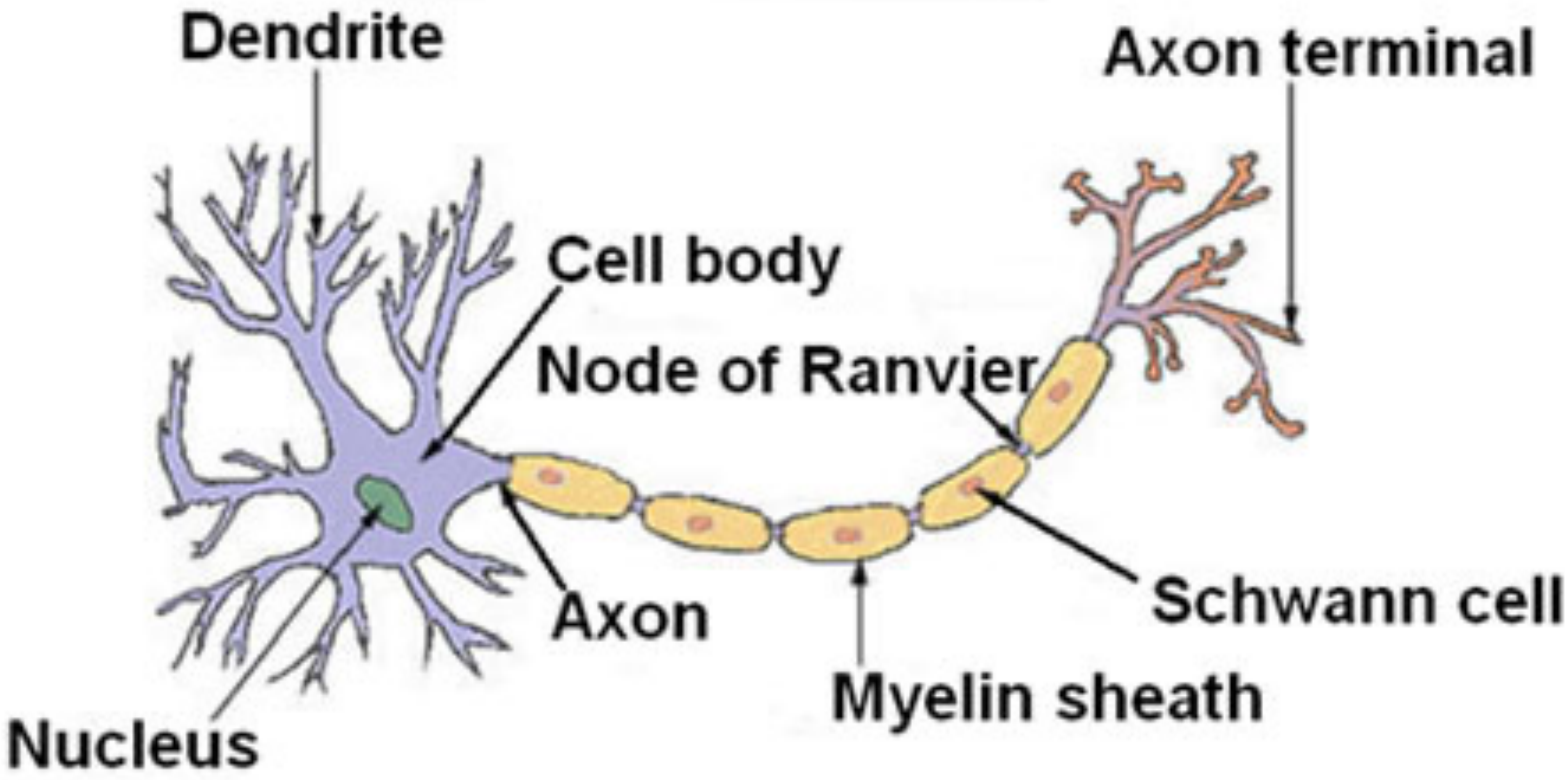


Machine Learning

Neural Networks: Representation

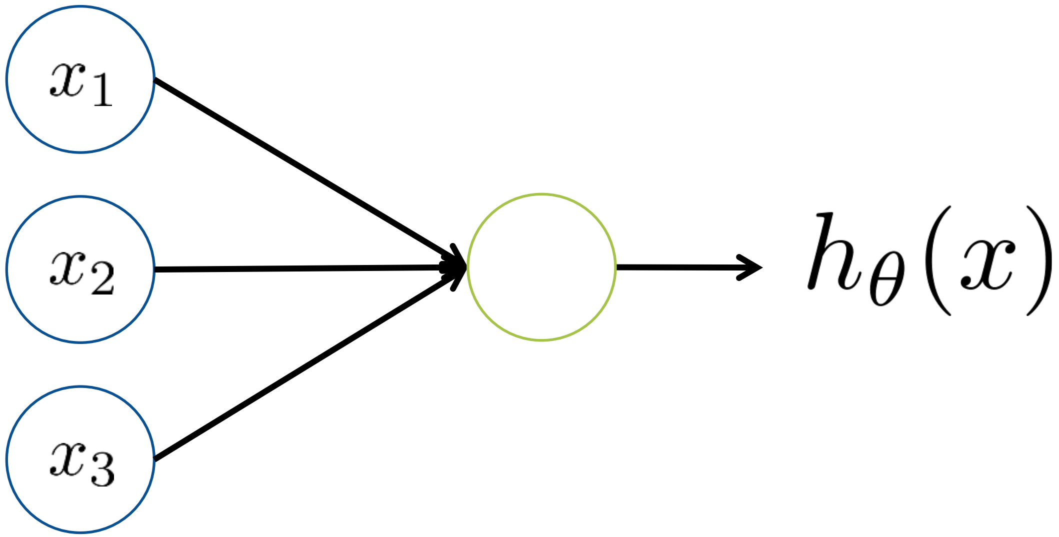
Model representation I

Neuron in the brain



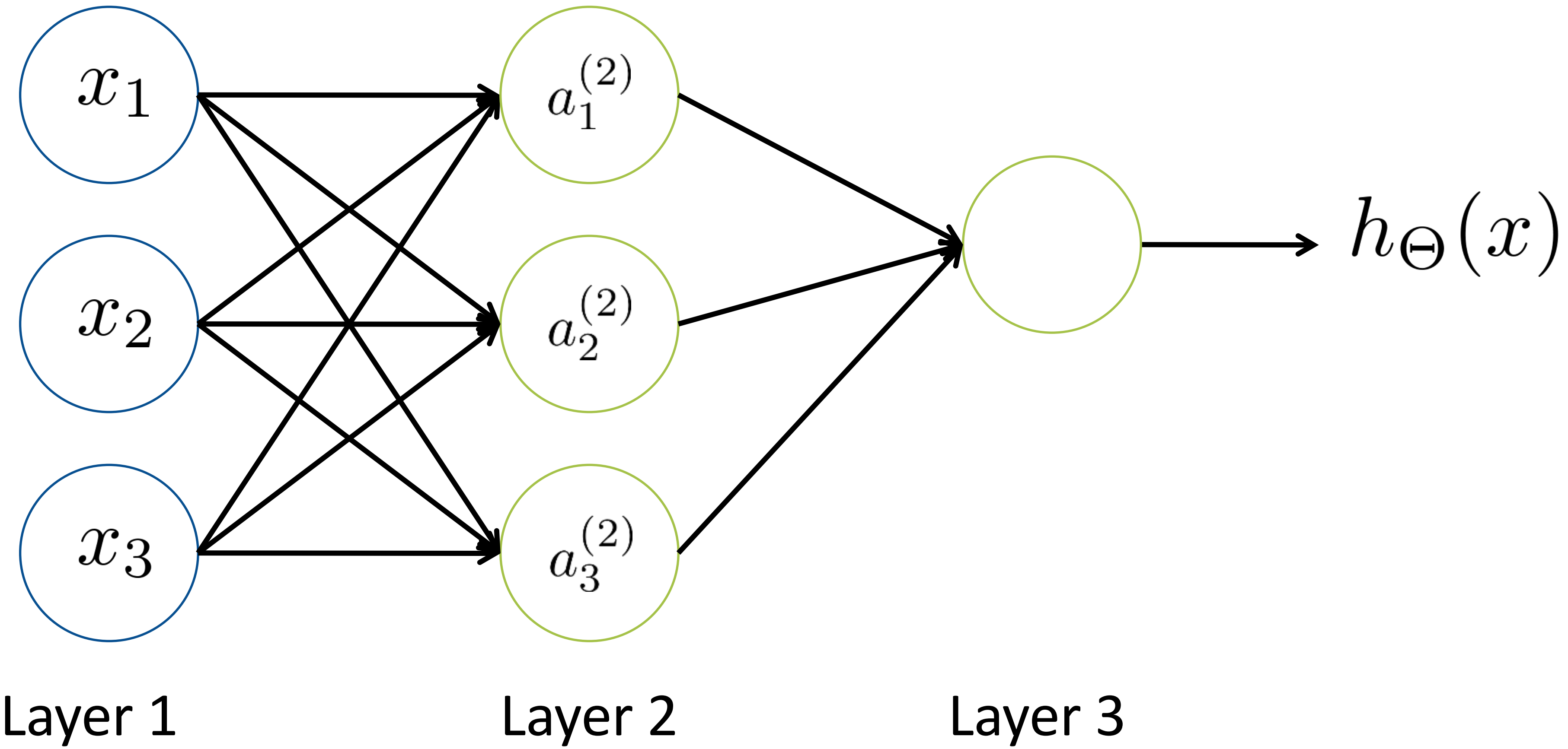
Neuron model: Logistic unit

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

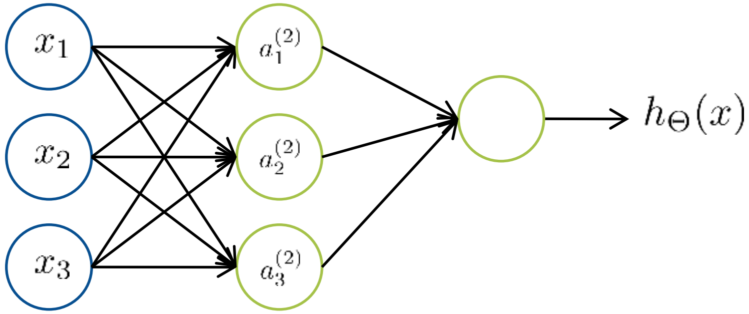


Sigmoid (logistic) activation function.

Neural Network



Neural Network



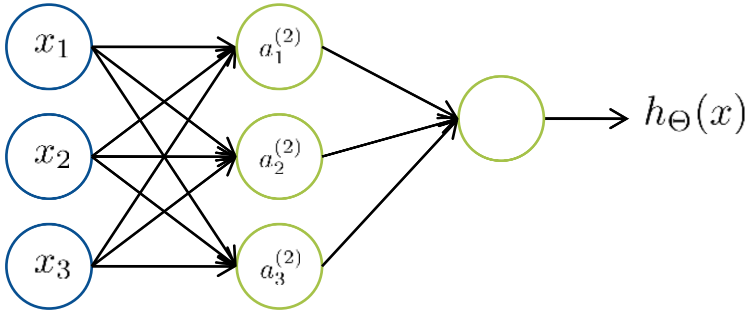
$a_i^{(j)} =$ "activation" of unit i in layer j

$\Theta^{(j)} =$ 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 $s_{j+1} \times (s_j + 1)$.

Forward propagation: Vectorized implementation



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

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

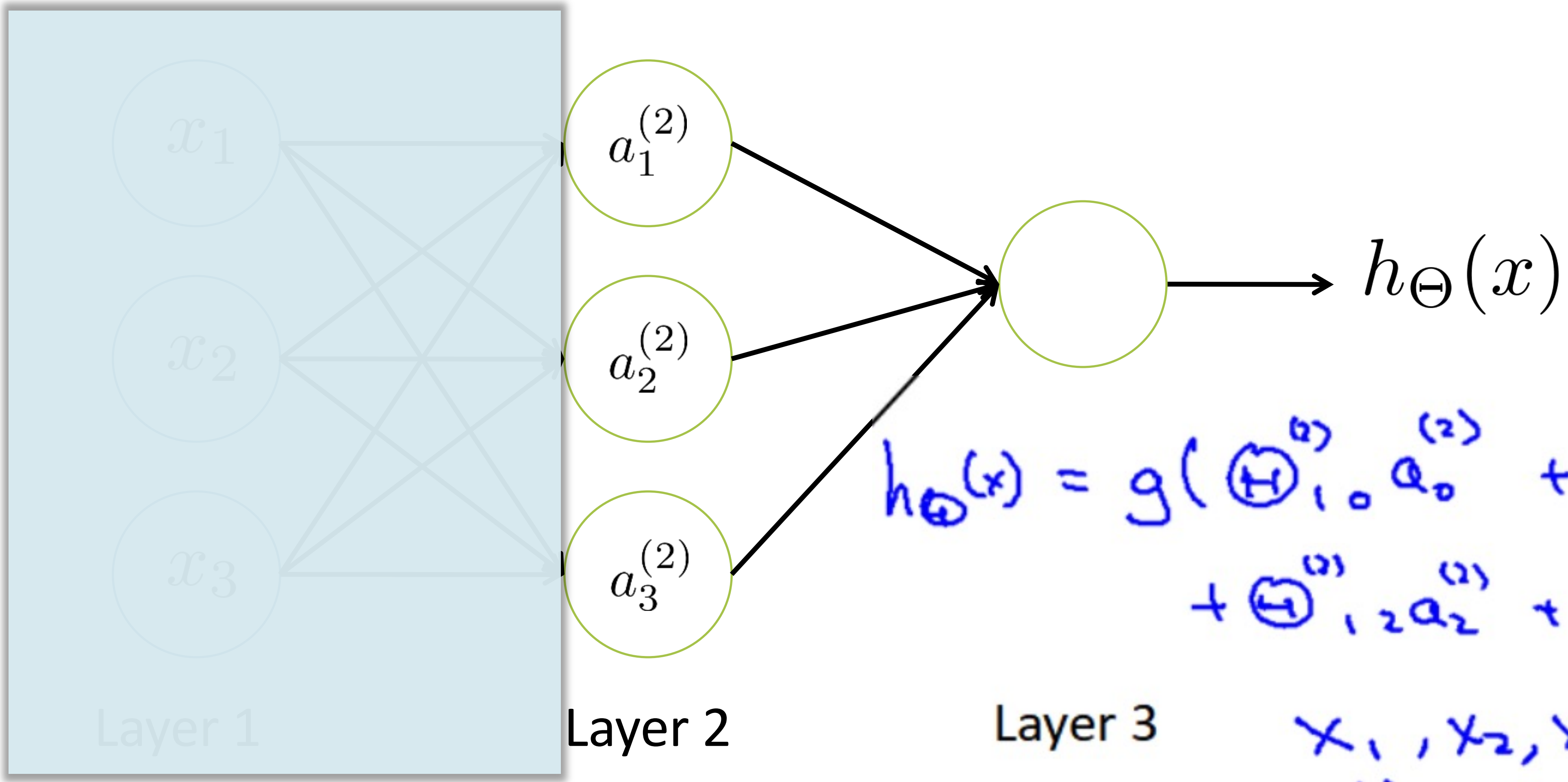
$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

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

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

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

Neural Network learning its own features

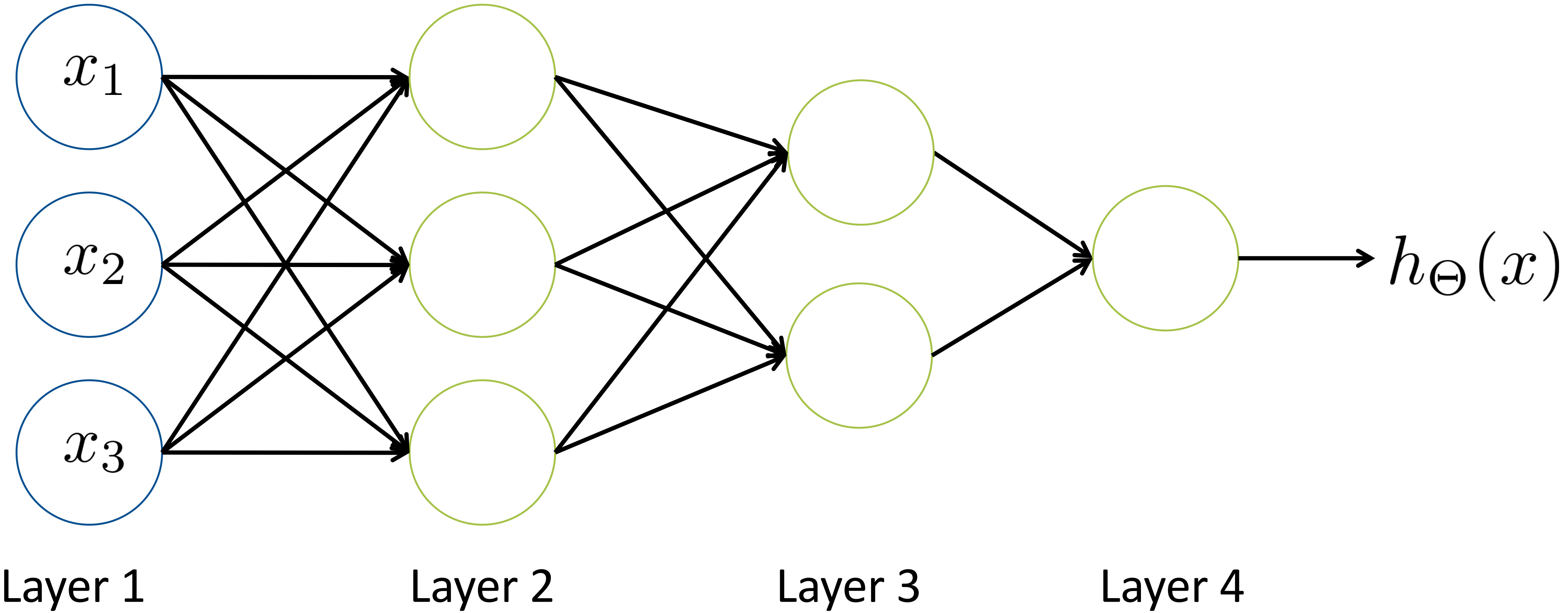


$$h_\Theta(x) = g\left(\Theta^{(2)}_{10} a^{(2)}_0 + \Theta^{(1)}_{11} a^{(2)}_1 + \Theta^{(2)}_{12} a^{(2)}_2 + \Theta^{(2)}_{13} a^{(2)}_3\right)$$

x_1, x_2, x_3

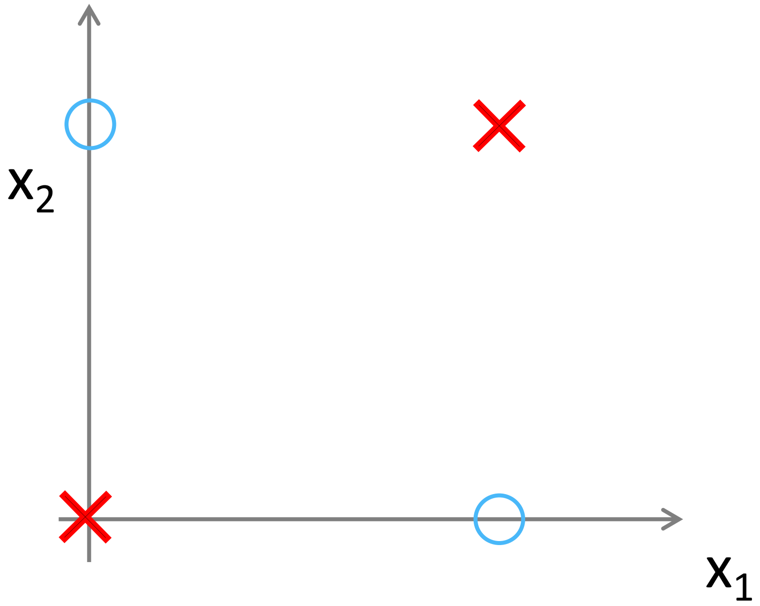
$\rightarrow a^{(2)}_1, a^{(2)}_2, a^{(2)}_3$

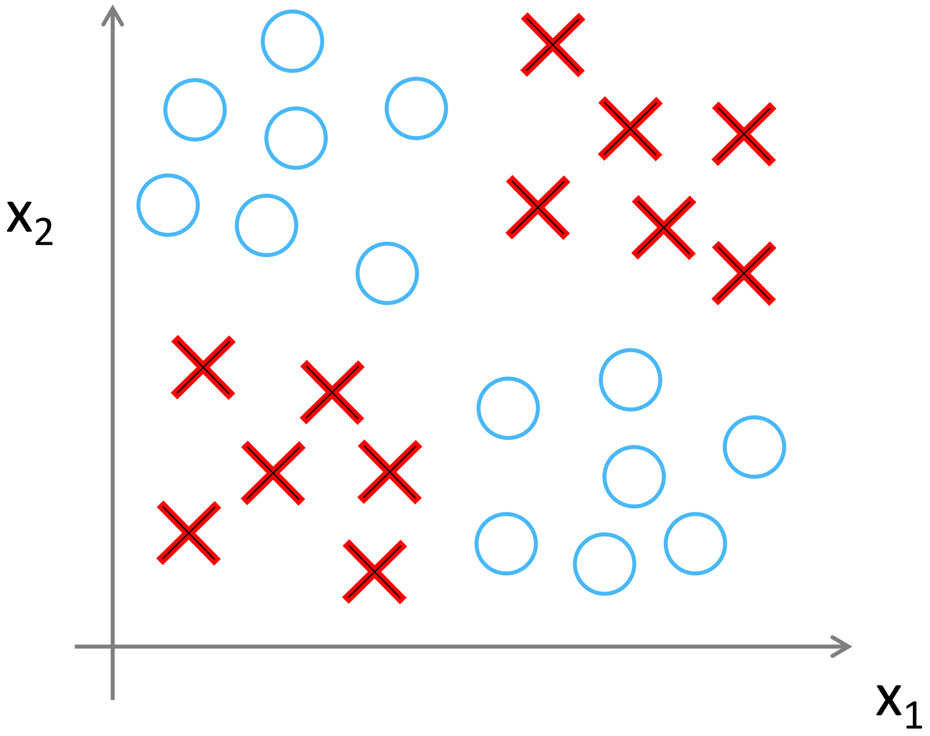
Other network architectures



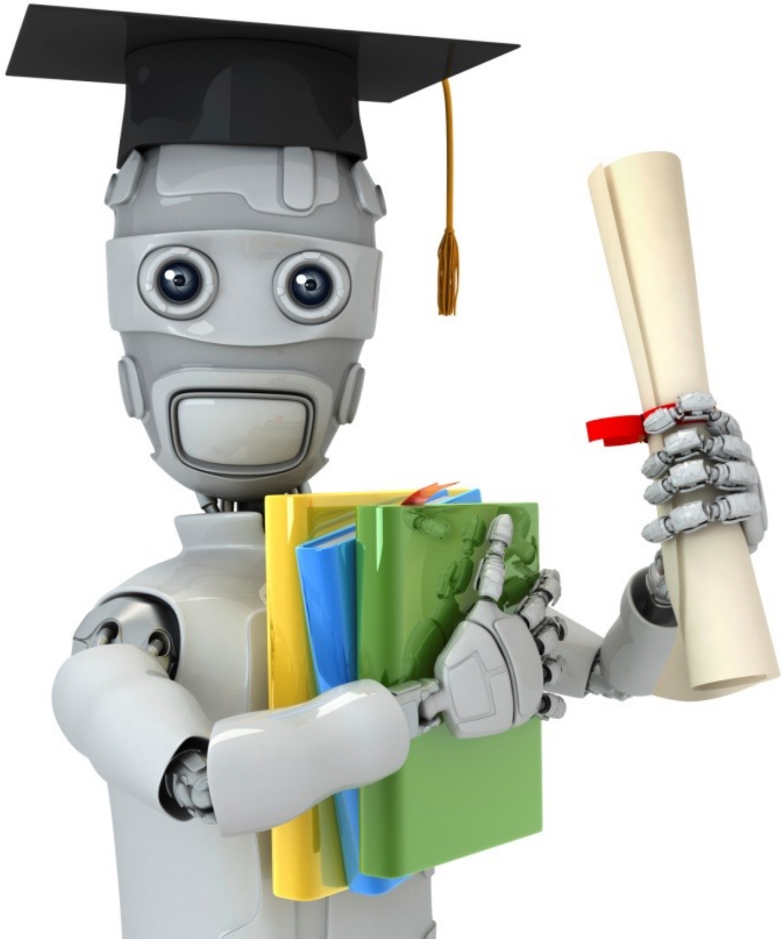
Non-linear classification example: XOR/XNOR

x_1, x_2 are binary (0 or 1).





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



Machine Learning

Neural Networks: Representation

Multi-class classification

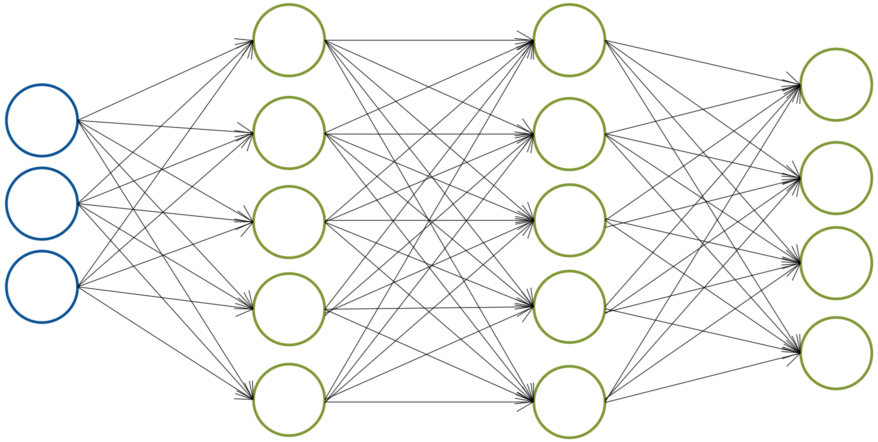
Multiple output units: One-vs-all.

Pedestrian

Car

Motorcycle

Truck

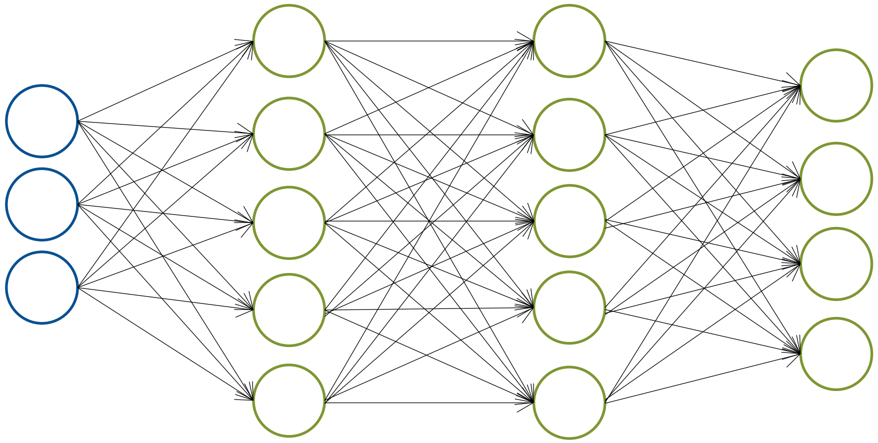


$$h_\Theta(x) \in \mathbb{R}^4$$

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

Multiple output units: One-vs-all.



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

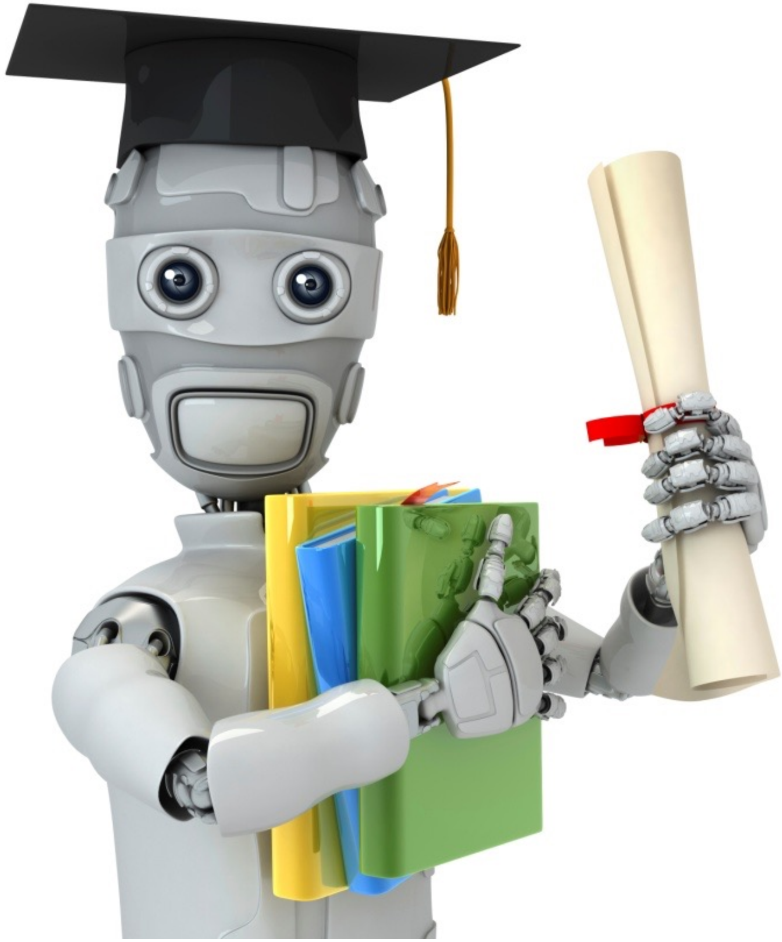
Want $h_\Theta(x) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, etc.

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

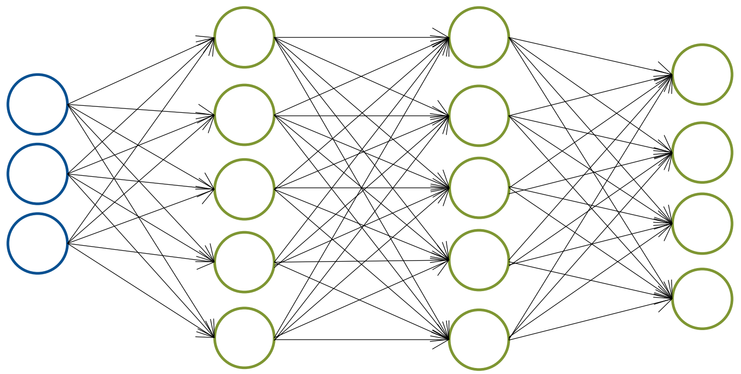


Neural Networks: Learning

Cost function

Machine Learning

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:

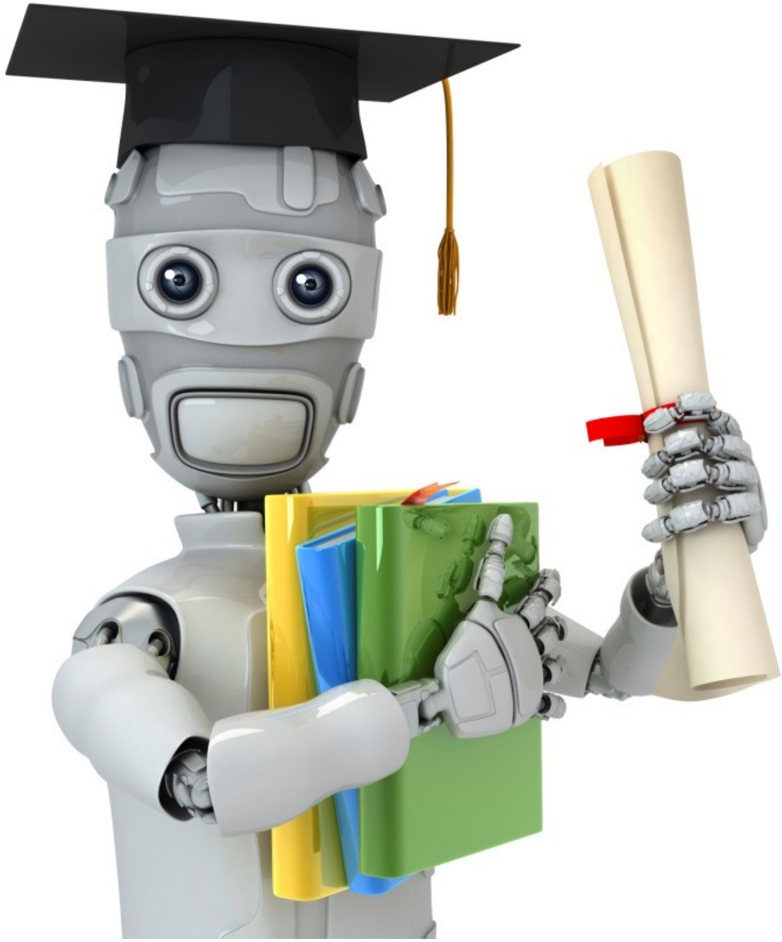
$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Neural network:

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

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

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



Machine Learning

Neural Networks: Learning

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:

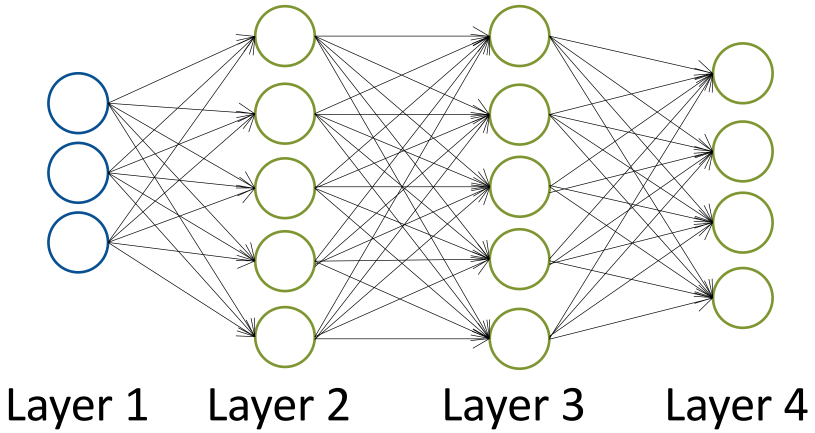
- $J(\Theta)$
- $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

Gradient computation

Given one training example (x, y):

Forward propagation:

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

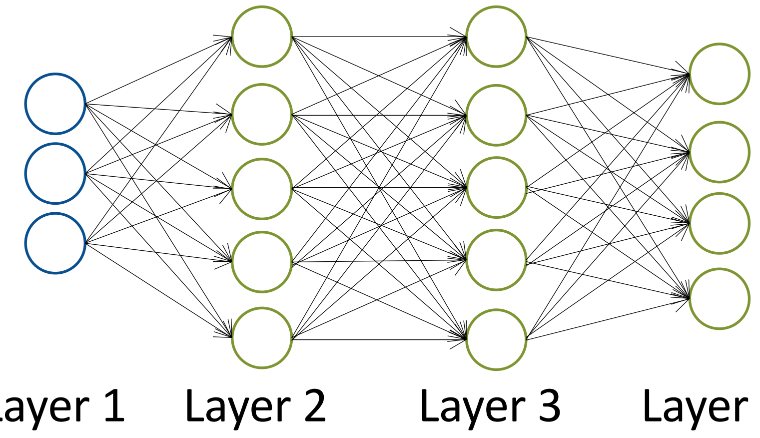


Gradient computation: Backpropagation algorithm

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

For each output unit (layer L = 4)

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



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

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

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

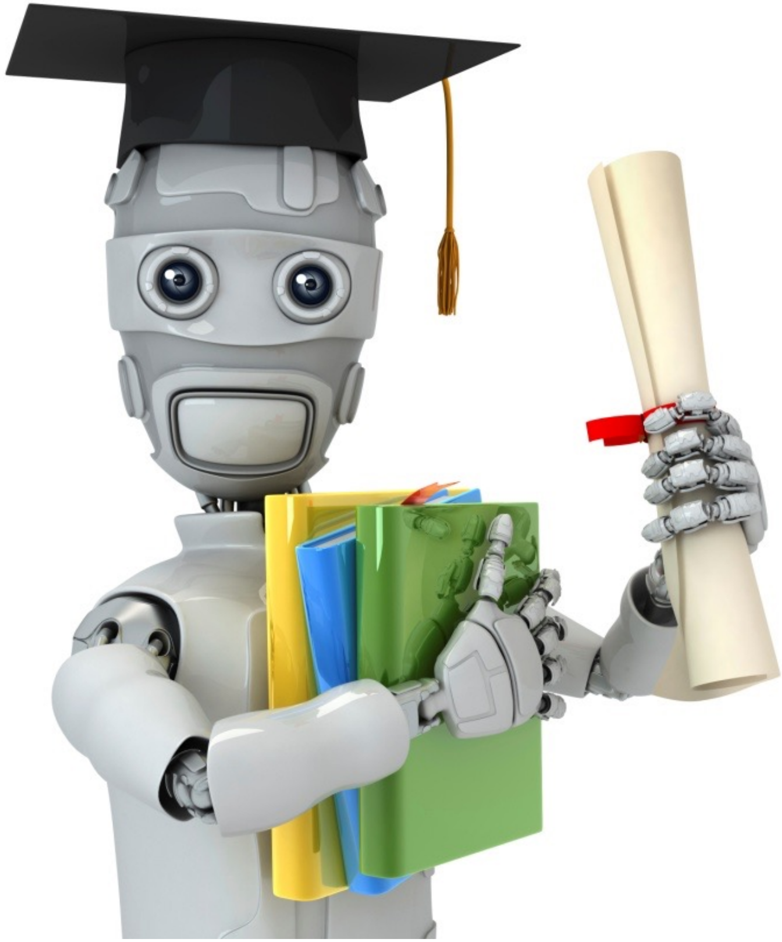
Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

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

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

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

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



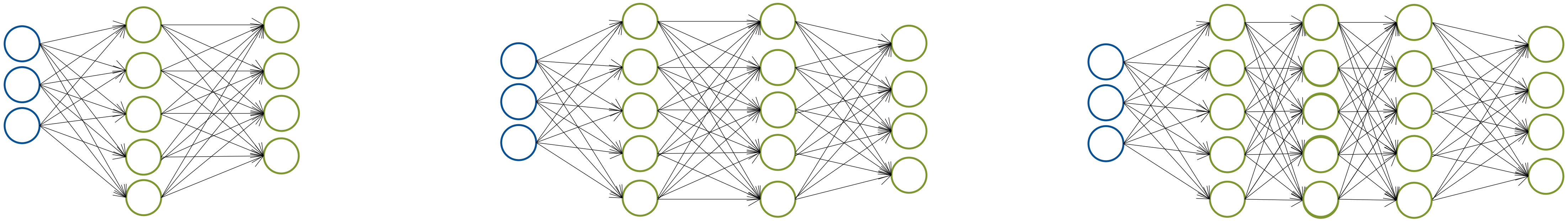
Machine Learning

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. 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)

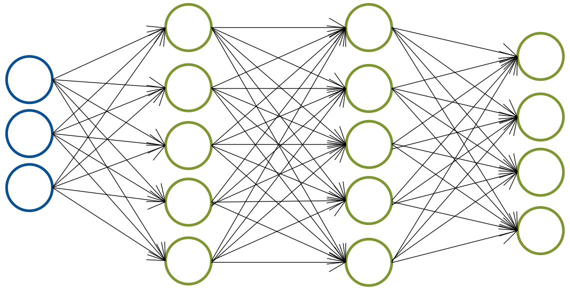
Training a neural network

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 $l = 2, \dots, L$).



Training a neural network

5. Use gradient checking to compare $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ 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 Θ