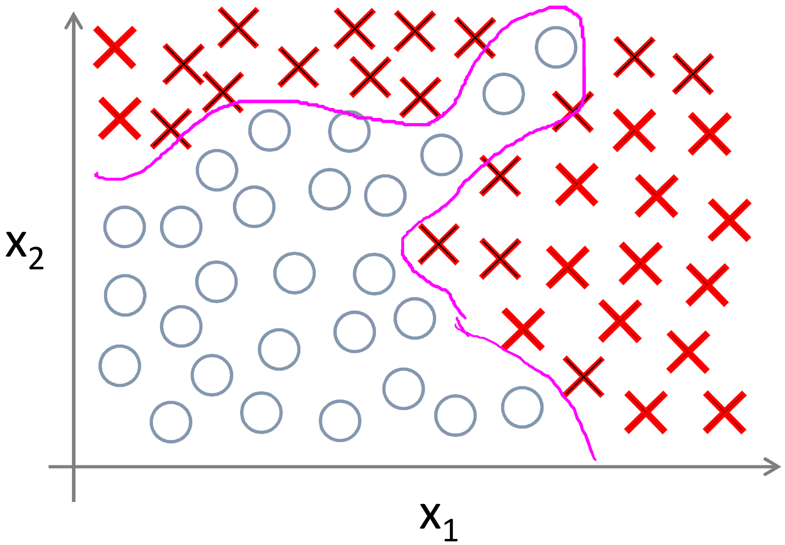
Neural networks

- **Duration**: 4 hrs
- Neural networks: representation
  - Non-linear hypotheses
  - Model representation
  - Multi-class classification
- Neural networks: learning
  - Cost function
  - Backpropagation algorithm
  - Implementation notes: unrolling parameters
  - Implementation notes: gradient checking
  - Random initialization
  - Putting it together

Source: Machine Learning, Andrew Ng, coursera.org

Non-linear Classification



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

$x_1^2, x_1x_2, x_1x_3, x_1x_4, \dots, x_1x_{100}$

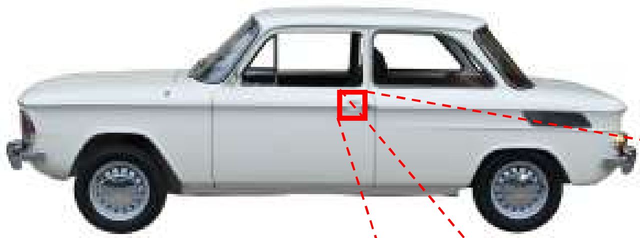
$x_2^2, x_2x_3, x_2x_4, \dots, x_2x_{100}$

$\frac{(100+1).100}{2} \approx$ 5000 features

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

What is this?

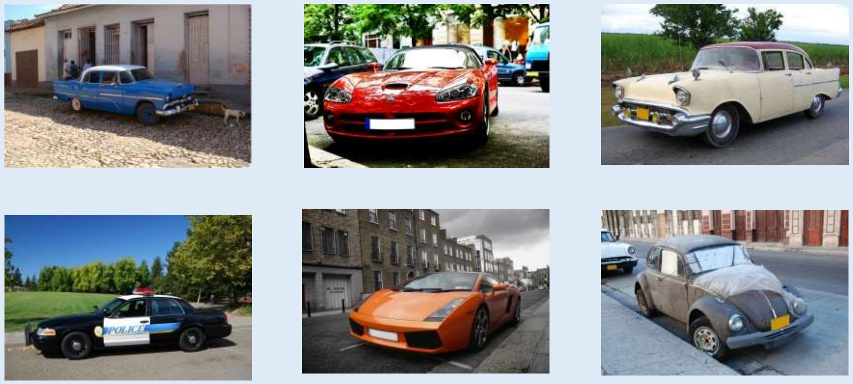
You see this:



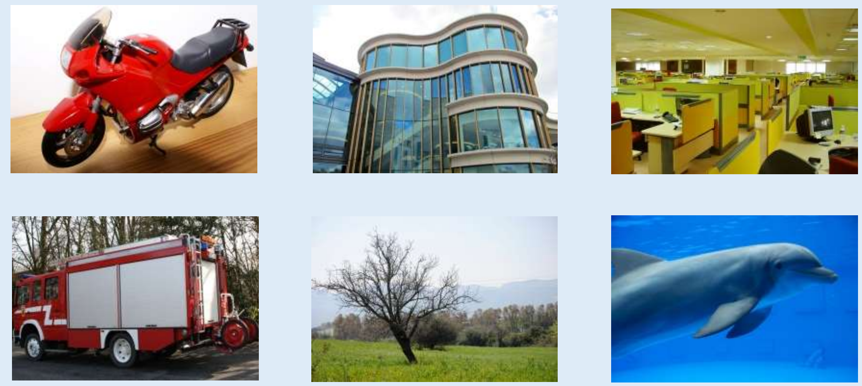
But the camera sees this:

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



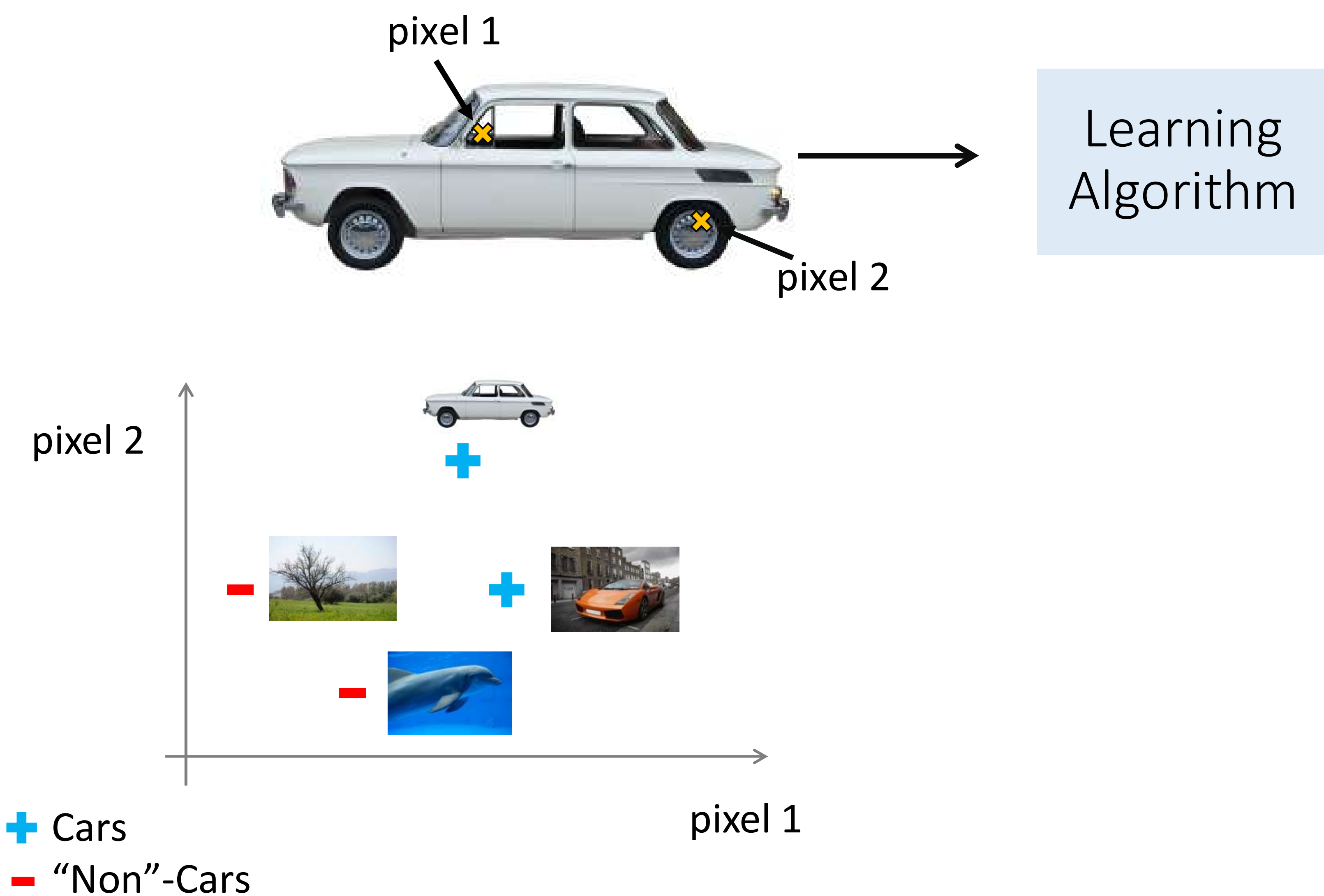
Cars

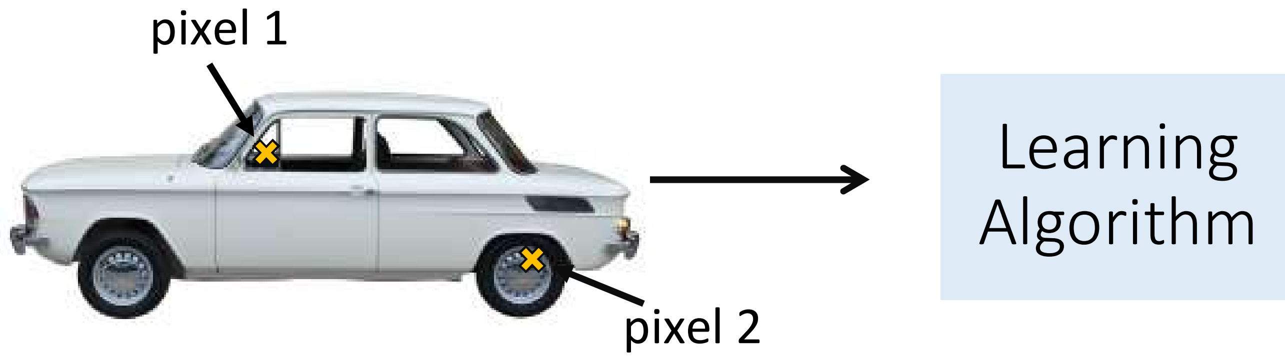


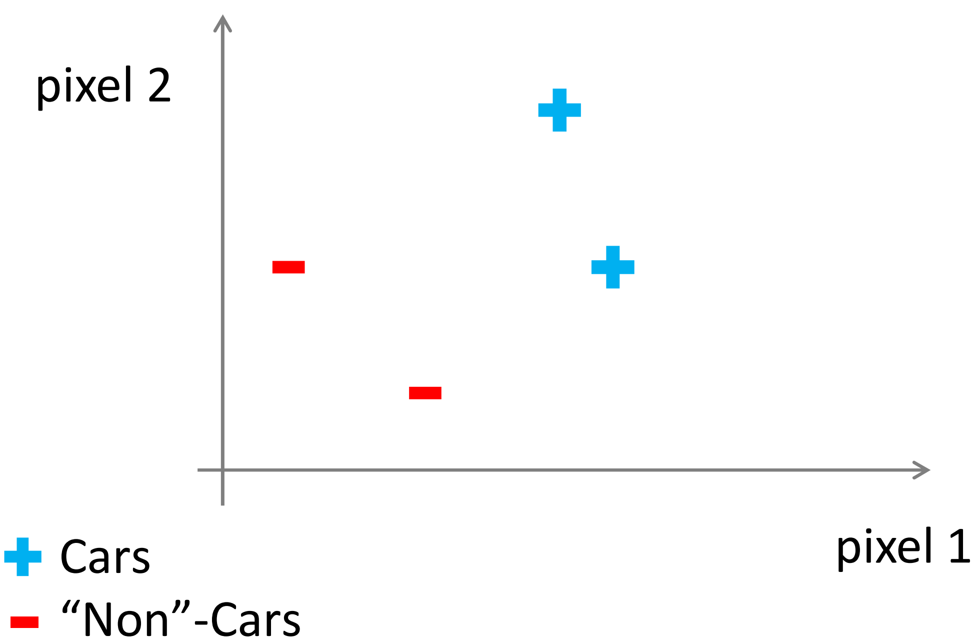
Not a car

Testing:

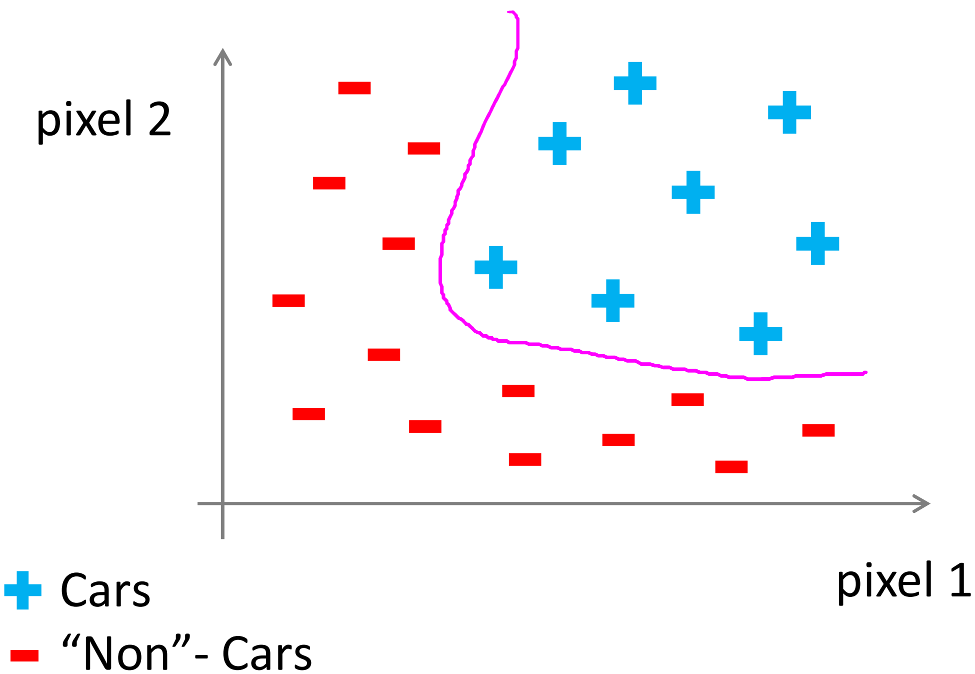
What is this?


pixel 1
pixel 2
Learning Algorithm
pixel 2
pixel 1
Cars
“Non”-Cars


pixel 1
pixel 2
Learning
Algorithm


pixel 2
pixel 1
Cars
"Non"-Cars

Learning Algorithm



50 x 50 pixel images→ 2500 pixels

$n = 2500$ (7500 if RGB)

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

0-255

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

Neural Networks

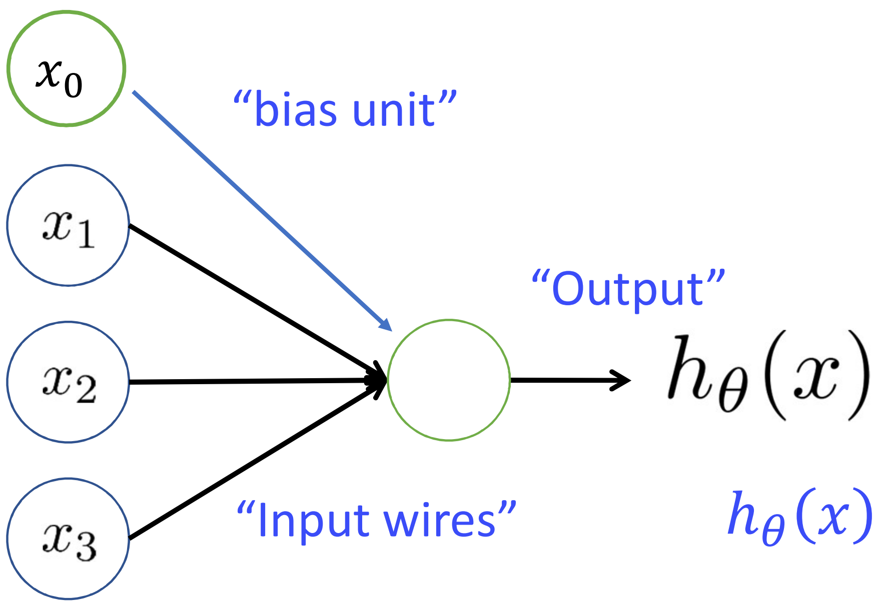
Origins: Algorithms that try to mimic the brain.
Was very widely used in 80s and early 90s; popularity diminished in late 90s.
Recent resurgence: State-of-the-art technique for many applications

Neural networks

- Neural networks: representation
  - Non-linear hypotheses
  - Model representation
  - Multi-class classification
- Neural networks: learning
  - Cost function
  - Backpropagation algorithm
  - Implementation notes: unrolling parameters
  - Implementation notes: gradient checking
  - Random initialization
  - Putting it together

Source: Machine Learning, Andrew Ng, coursera.org

Neuron model: Logistic unit



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

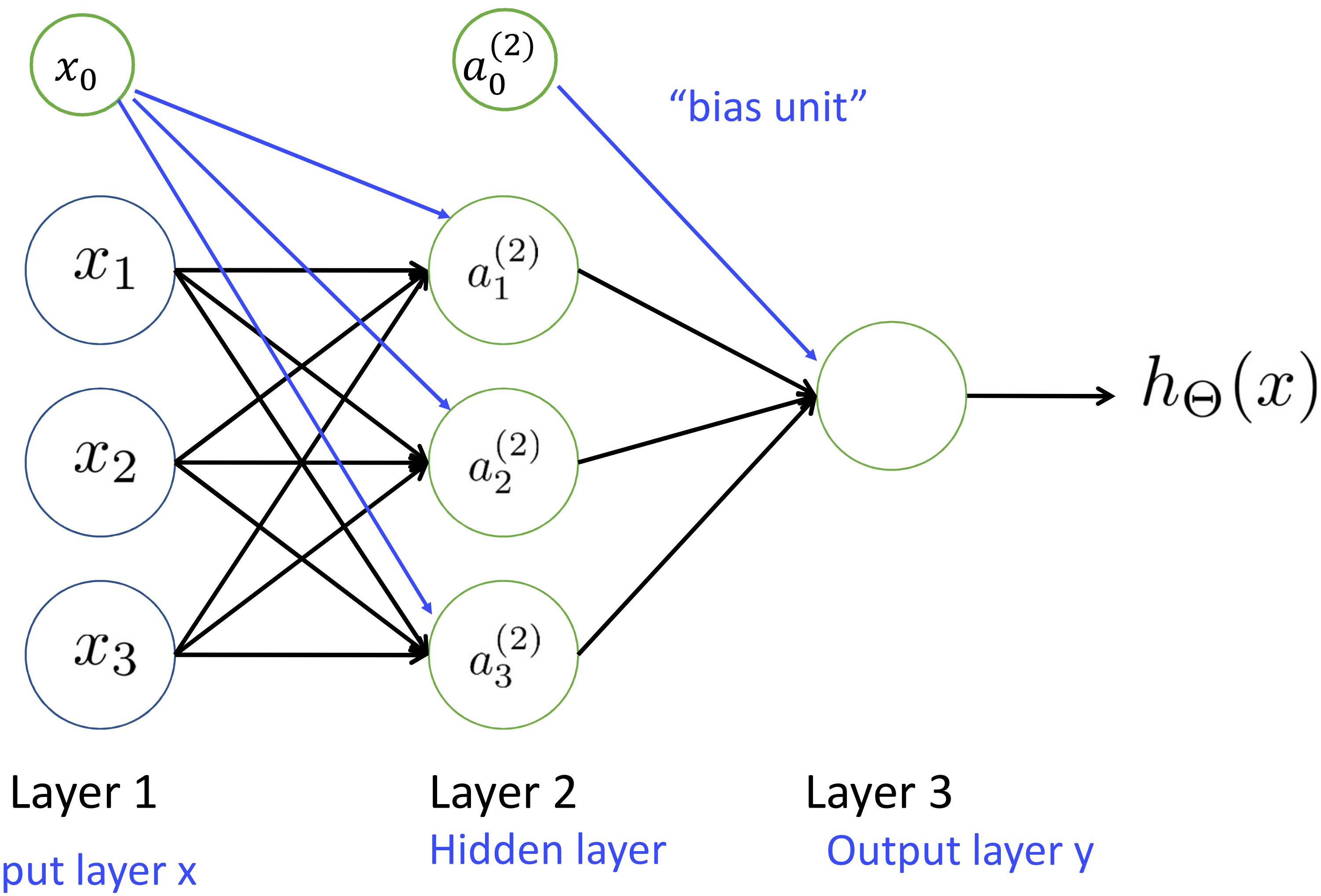
"Weights" parameters

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

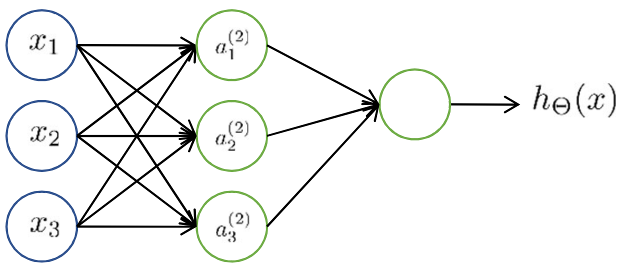
Sigmoid (logistic) activation function.

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

Neural Network



Neural Network



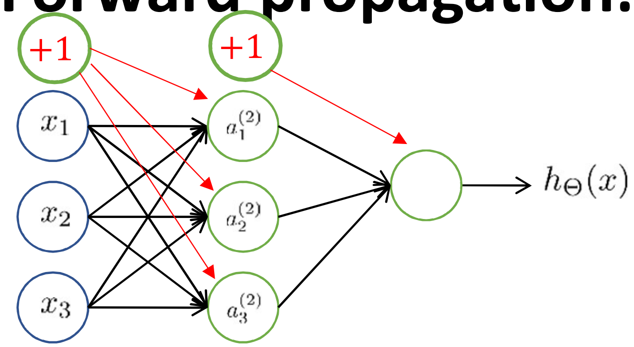
$a_i^{(j)} =$ "activation" of unit i in layer j

$\Theta^{(j)} =$ 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



$a^{(1)} = x$

$a_1^{(2)} = g(z_1^{(2)})$

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

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

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

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

$a_3^{(2)} = g(z_3^{(2)})$

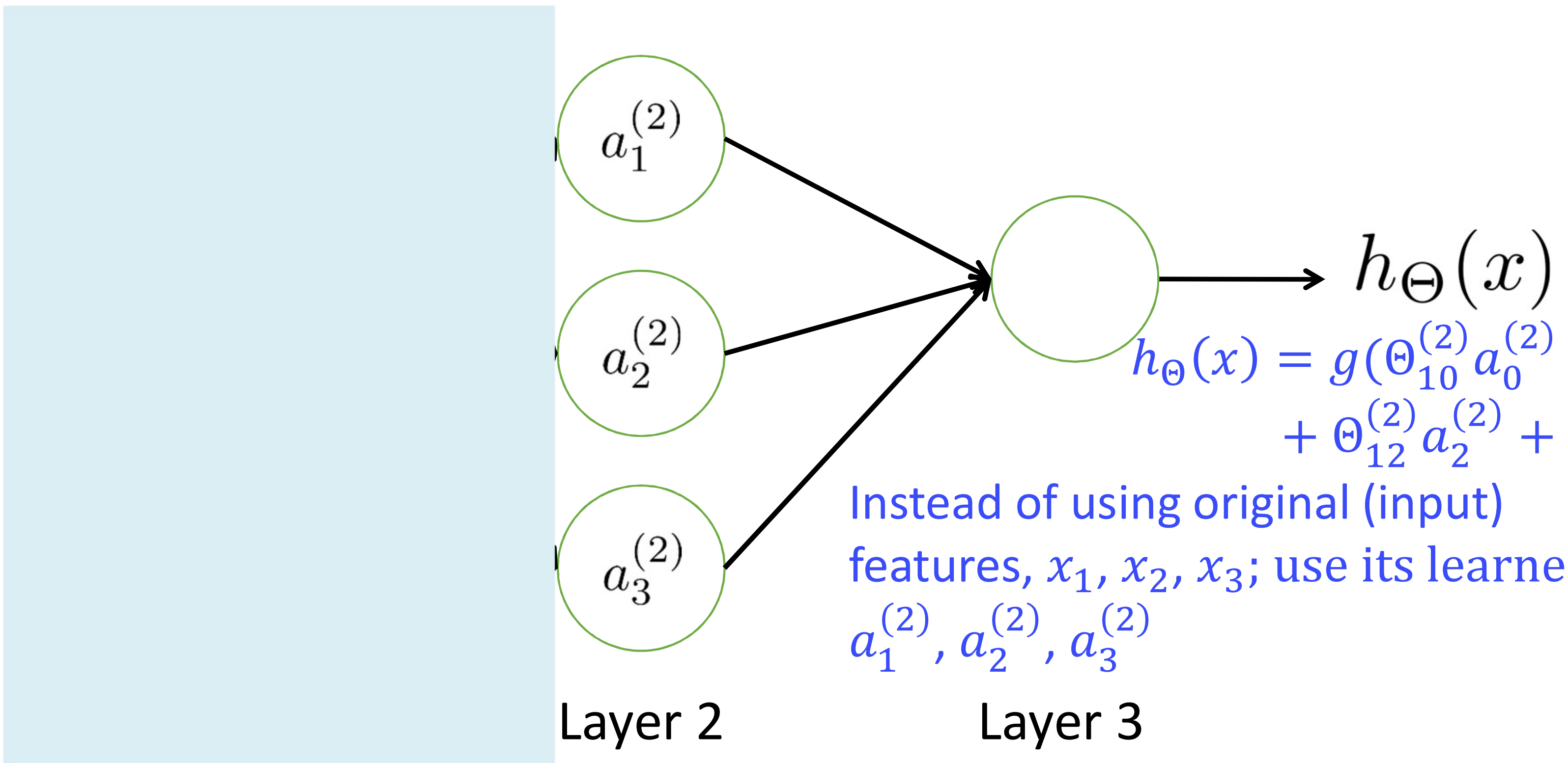
$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

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

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

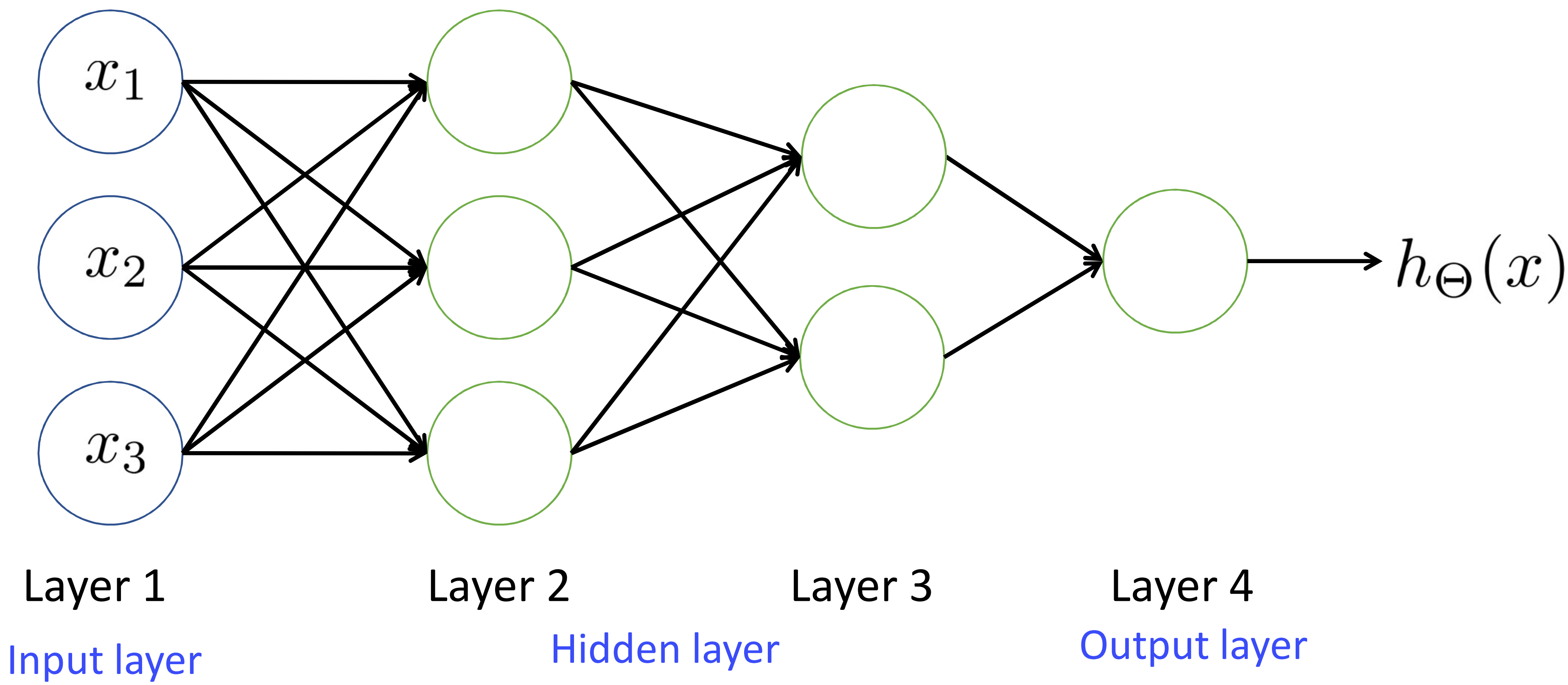


$$h_\Theta(x) = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}$$

Instead of using original (input) features, x_1, x_2, x_3; use its learned features $a_1^{(2)}$, $a_2^{(2)}$, $a_3^{(2)}$

$\Theta^{(1)}$

Other network architectures



Neural networks

- Neural networks: representation
  - Non-linear hypotheses
  - Model representation
  - Multi-class classification
- Neural networks: learning
  - Cost function
  - Backpropagation algorithm
  - Implementation notes: unrolling parameters
  - Implementation notes: gradient checking
  - Random initialization
  - Putting it together

Source: Machine Learning, Andrew Ng, coursera.org

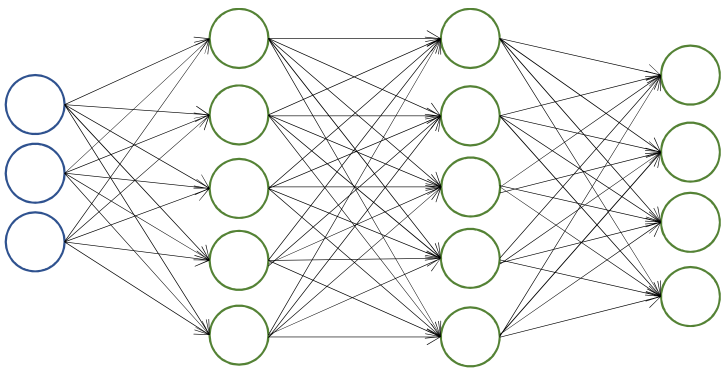
Multiple output units: One-vs-all.


Pedestrian


Car


Motorcycle


Truck



Pedestrian?

Car?

Motorcycle ?

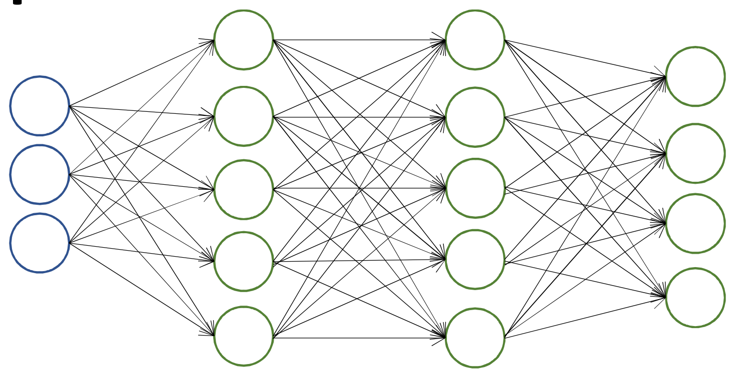
Truck?

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

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

pedestrian car motorcycle truck

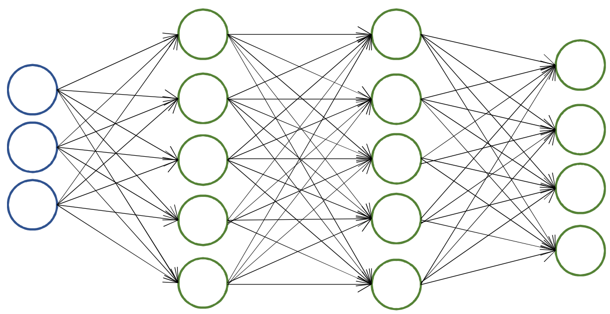
NOT use $y \epsilon \{1,2,3,4,5, \ldots\}$ as used previously

Neural networks

- Neural networks: representation
  - Non-linear hypotheses
  - Model representation
  - Multi-class classification
- Neural networks: learning
  - Cost function
  - Backpropagation algorithm
  - Implementation notes: unrolling parameters
  - Implementation notes: gradient checking
  - Random initialization
  - Putting it together

Source: Machine Learning, Andrew Ng, coursera.org

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:

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

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

Neural networks

- Neural networks: representation
  - Non-linear hypotheses
  - Model representation
  - Multi-class classification
- Neural networks: learning
  - Cost function
  - Backpropagation algorithm
  - Implementation notes: unrolling parameters
  - Implementation notes: gradient checking
  - Random initialization
  - Putting it together

Source: Machine Learning, Andrew Ng, coursera.org

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:

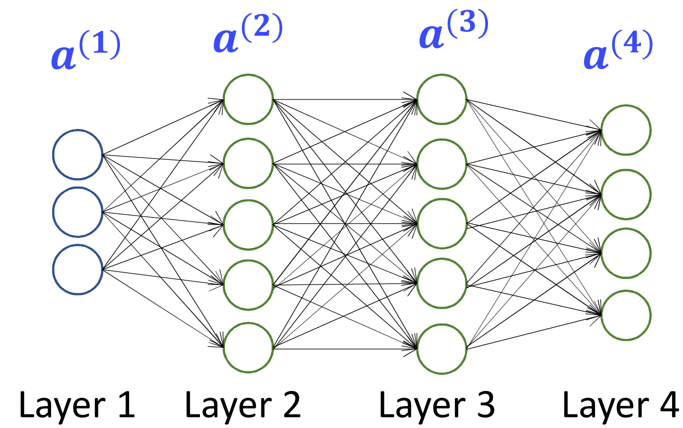
- $J(\Theta)$
- $\frac{\partial}{\partial\Theta_{ij}^{(l)}} J(\Theta)$

Gradient computation

Given one training example (x, y):

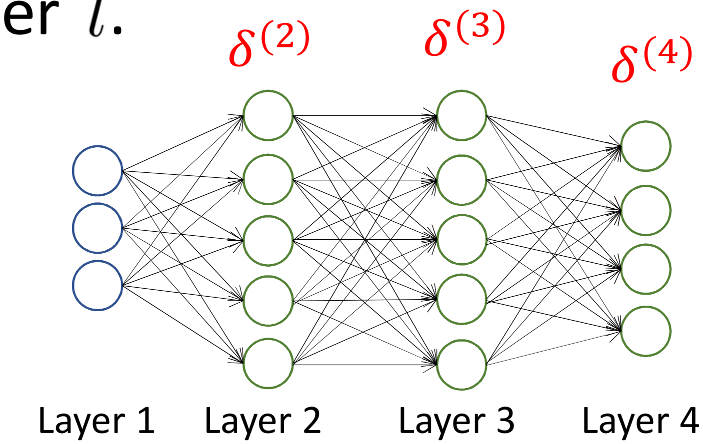
Forward propagation:

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



Gradient computation: Backpropagation algorithm

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



For each output unit (layer L = 4)

$\delta_j^{(4)} = a_j^{(4)} - y_j$ $\quad \delta^{(4)} = a^{(4)} - y$

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

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

No $\delta^{(1)}$

$$\frac{\partial}{\partial \Theta_i^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)} \ ignoring\ \lambda, if \lambda = 0$$

Backpropagation algorithm

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

Set $\triangle_{ij}^{(l)} = 0$ (for all l, i, j). Used to compute $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$

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

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

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

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

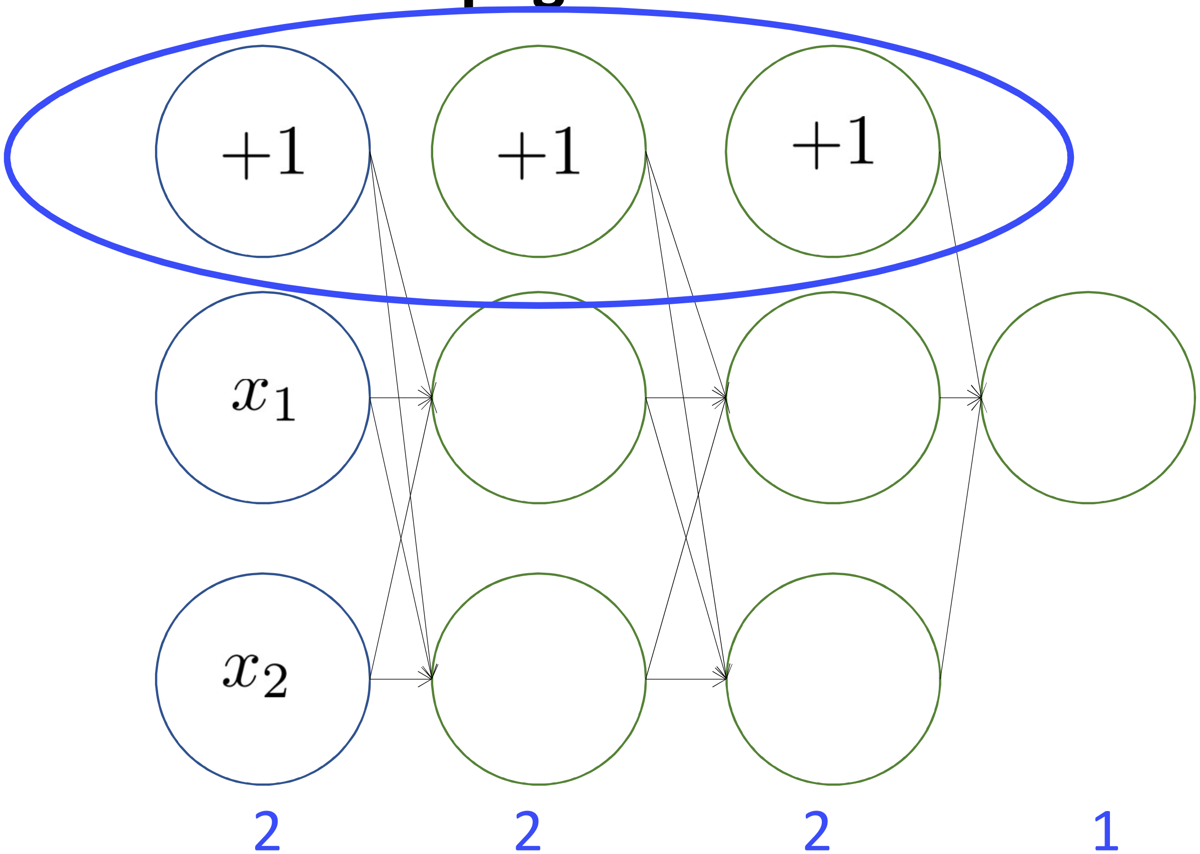
$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

Neural networks

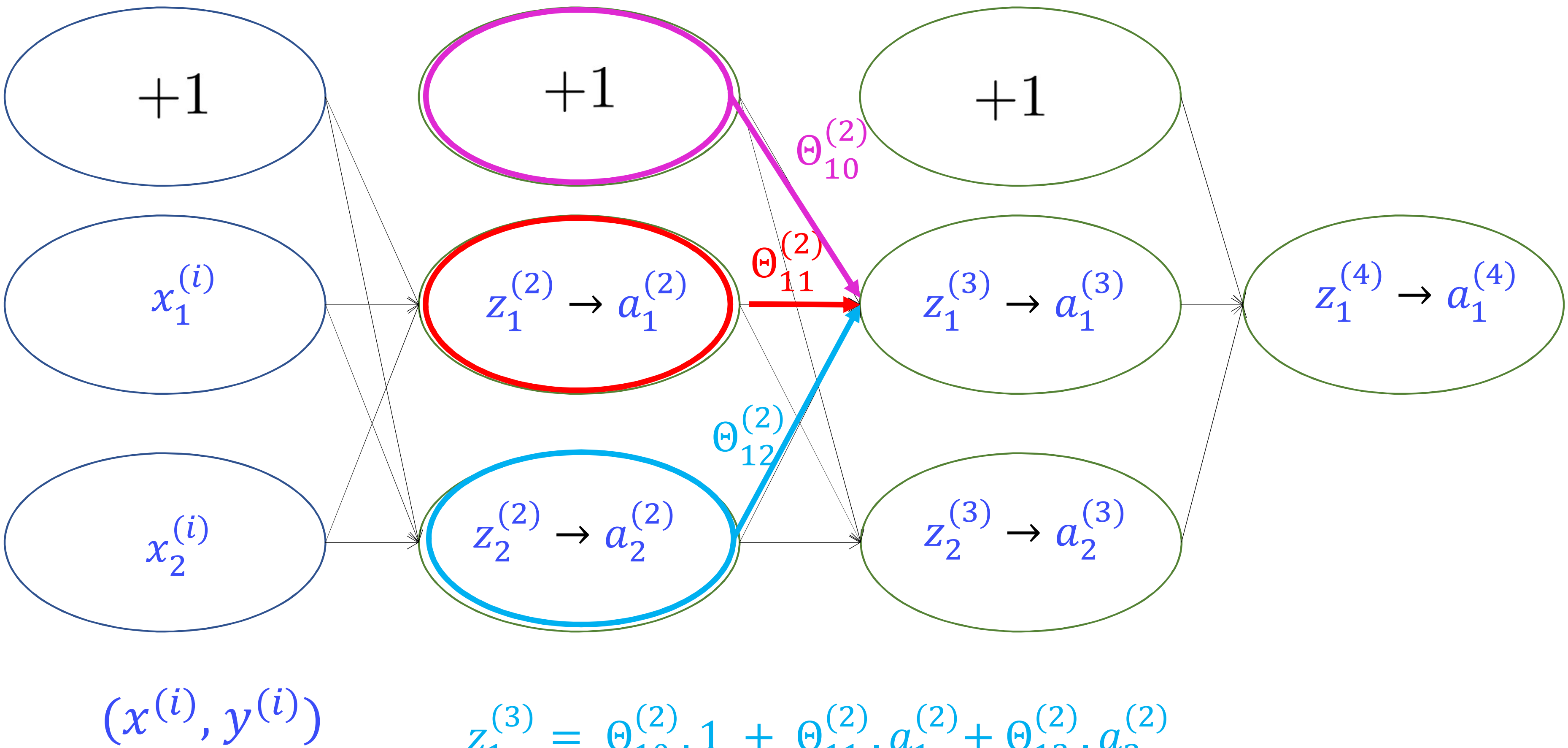
- Neural networks: representation
  - Non-linear hypotheses
  - Model representation
  - Multi-class classification
- Neural networks: learning
  - Cost function
  - Backpropagation algorithm
  - Implementation notes: unrolling parameters
  - Implementation notes: gradient checking
  - Random initialization
  - Putting it together

Source: Machine Learning, Andrew Ng, coursera.org

Forward Propagation



Forward Propagation



$$z_1^{(3)} = \Theta_{10}^{(2)}.1 + \Theta_{11}^{(2)}.a_1^{(2)} + \Theta_{12}^{(2)}.a_2^{(2)}$$

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 ($\lambda = 0$),

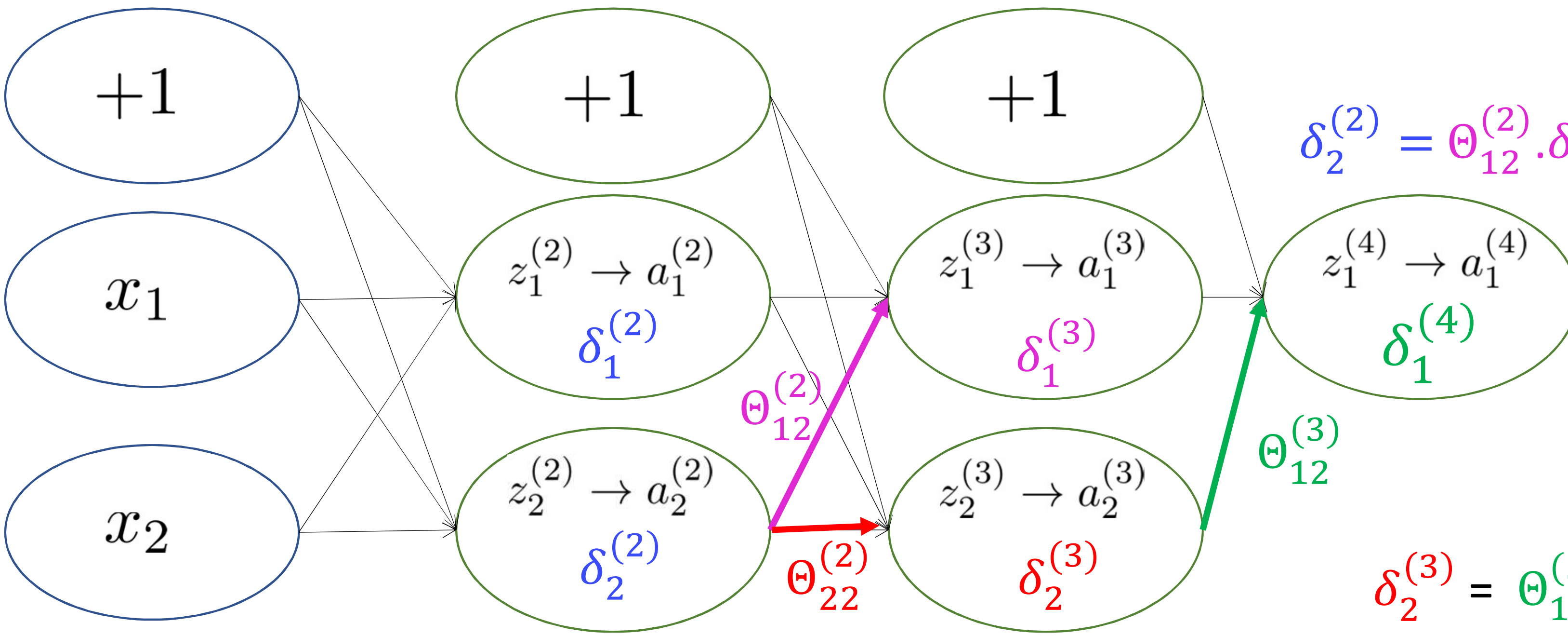
$$\text{cost(i)} = y^{(i)} \log h_\Theta(x^{(i)}) + (1-y^{(i)}) \log h_\Theta(x^{(i)})$$

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

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

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

Forward Propagation



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

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

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

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

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

Neural networks

- Neural networks: representation
  - Non-linear hypotheses
  - Model representation
  - Multi-class classification
- Neural networks: learning
  - Cost function
  - Backpropagation algorithm
  - Implementation notes: unrolling parameters
  - Implementation notes: gradient checking
  - Random initialization
  - Putting it together

Source: Machine Learning, Andrew Ng, coursera.org

Advanced optimization

```
function [jVal, gradient] = costFunction(theta)
    ...
optTheta = fminunc(@costFunction, initialTheta, options)
```

$\mathbb{R}^{n+1}$ $\mathbb{R}^{n+1}$ (vectors)

Neural Network (L=4):

$\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ matrices (`Theta1, Theta2, Theta3`)

$D^{(1)}, D^{(2)}, D^{(3)}$ matrices (`D1, D2, D3`)

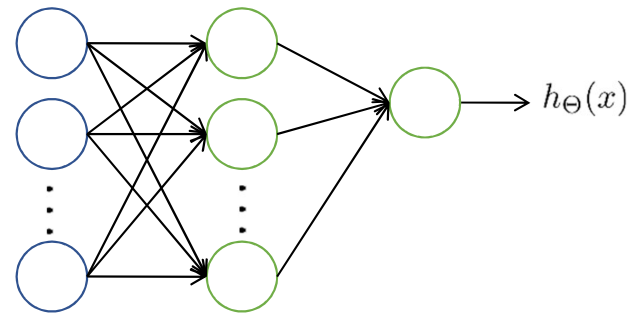
"Unroll" into vectors

Example

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

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

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



$\Theta^{(1)}$ $\Theta^{(2)}$ $\Theta^{(3)}$

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

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

Learning Algorithm

Have initial parameters $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$.

Unroll to get `initialTheta` to pass to

```
fminunc(@costFunction, initialTheta, options)
```

```
function [jval, gradientVec] = costFunction(thetaVec)
```

From `thetaVec`, get $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$. *Reshape*

Use forward prop/back prop to compute $D^{(1)}, D^{(2)}, D^{(3)}$ and $J(\Theta)$.

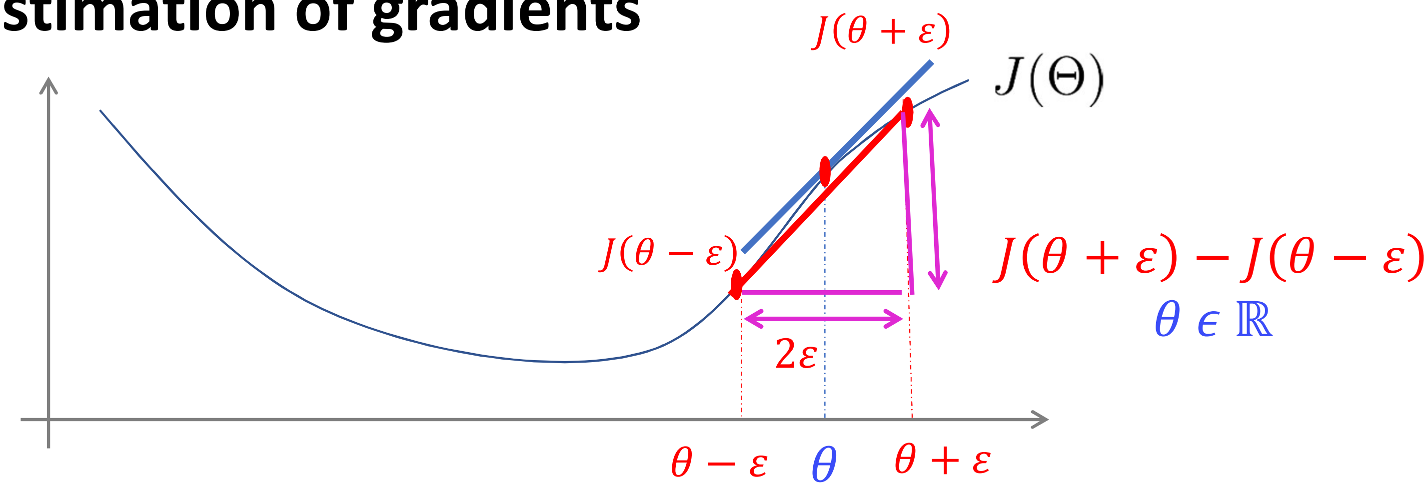
Unroll $D^{(1)}, D^{(2)}, D^{(3)}$ to get `gradientVec`.

Neural networks

- Neural networks: representation
  - Non-linear hypotheses
  - Model representation
  - Multi-class classification
- Neural networks: learning
  - Cost function
  - Backpropagation algorithm
  - Implementation notes: unrolling parameters
  - Implementation notes: gradient checking
  - Random initialization
  - Putting it together

Source: Machine Learning, Andrew Ng, coursera.org

Numerical estimation of gradients



$$\frac{d}{d\theta} J(\theta) \approx \frac{J(\theta + \varepsilon) - J(\theta - \varepsilon)}{2\varepsilon}$$

$\varepsilon = 10^{-4}$

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

Parameter vector θ

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

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

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

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

$$\vdots$$

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

$$\begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_1 + \varepsilon \\ \vdots \\ \theta_n \end{bmatrix}$$

```
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;
```

$\frac{\partial}{\partial \theta_i} J(\theta)$

Check that `gradApprox` ≈ $\boxed{\texttt{DVec}}$

From backpropagation

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 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.

Neural networks

- Neural networks: representation
  - Non-linear hypotheses
  - Model representation
  - Multi-class classification
- Neural networks: learning
  - Cost function
  - Backpropagation algorithm
  - Implementation notes: unrolling parameters
  - Implementation notes: gradient checking
  - Random initialization
  - Putting it together

Source: Machine Learning, Andrew Ng, coursera.org

Initial value of Θ

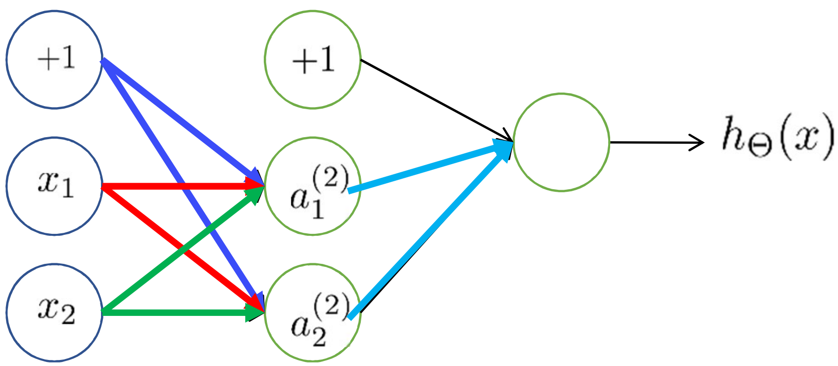
For gradient descent and advanced optimization method, need initial value for Θ.

```
optTheta = fminunc(@costFunction,
    initialTheta, options)
```

Consider gradient descent

Set `initialTheta = zeros(n,1)`?

Zero initialization



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

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

$$\frac{\partial}{\partial \Theta_{10}^{(1)}} J(\Theta) = \frac{\partial}{\partial \Theta_{20}^{(1)}} J(\Theta) \qquad \Theta_{10}^{(1)} = \Theta_{20}^{(1)}$$

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 $[-\epsilon, \epsilon]$
(i.e. $-\epsilon \leq \Theta_{ij}^{(l)} \leq \epsilon$)

E.g.

random 10×11 matrix (between 0 and 1)

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

$[-\epsilon, \epsilon]$

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

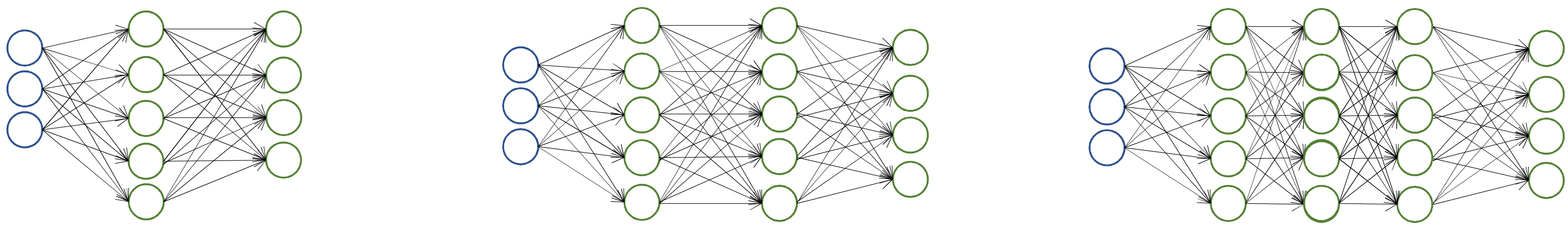
Neural networks

- Neural networks: representation
  - Non-linear hypotheses
  - Model representation
  - Multi-class classification
- Neural networks: learning
  - Cost function
  - Backpropagation algorithm
  - Implementation notes: unrolling parameters
  - Implementation notes: gradient checking
  - Random initialization
  - Putting it together

Source: Machine Learning, Andrew Ng, coursera.org

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

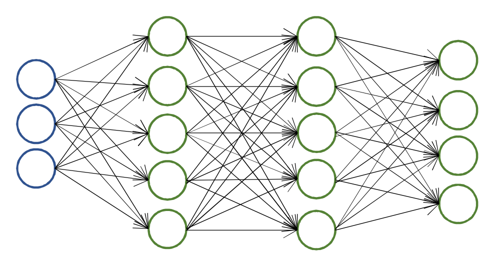
1. Randomly initialize weights
2. Implement forward propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$
3. Implement code to compute cost function $J(\Theta)$
4. Implement backprop to compute partial derivatives $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$

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

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

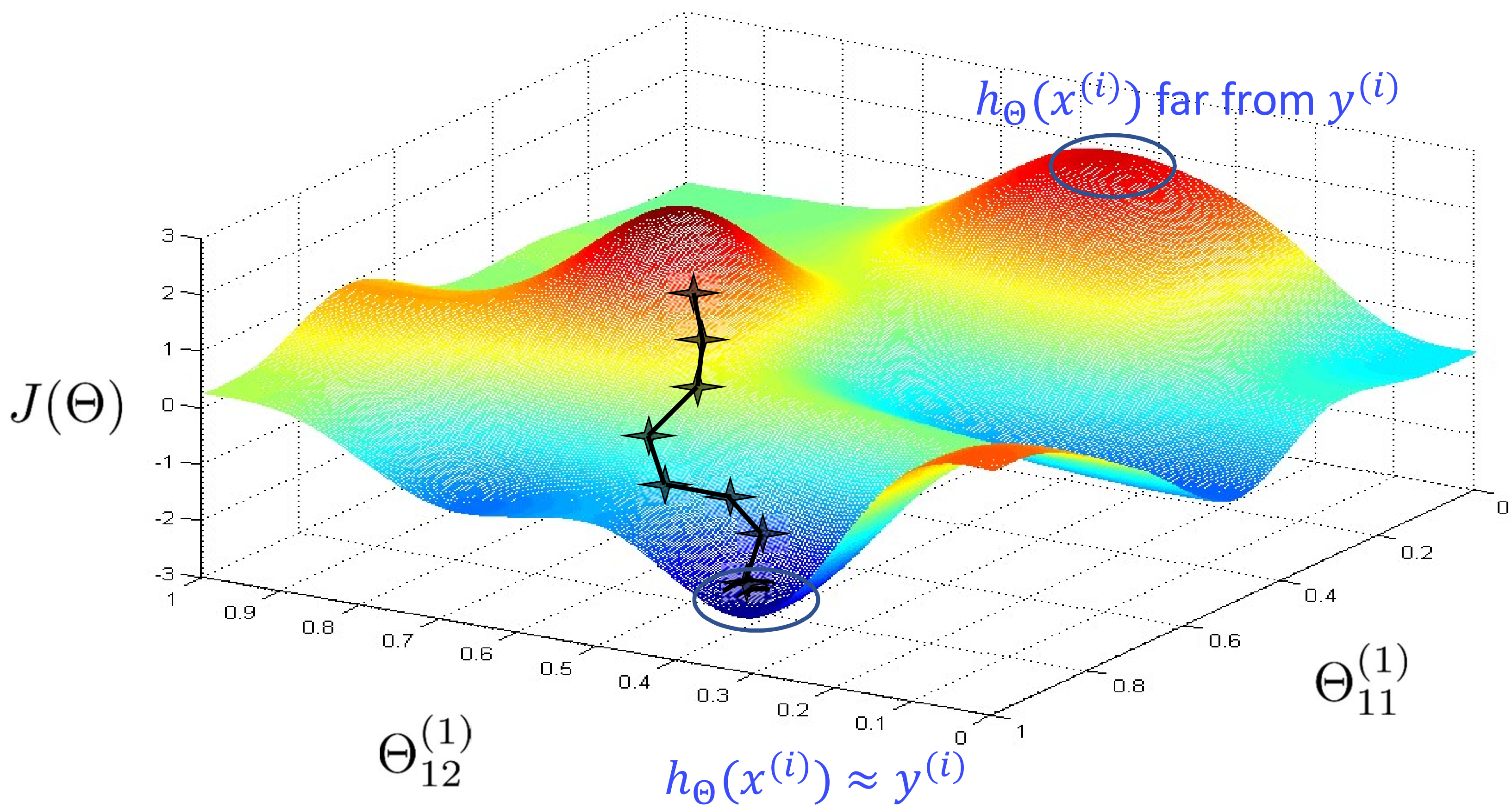


compute $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$

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

$J(\Theta)$ – non-convex


$h_\Theta(x^{(i)})$ far from $y^{(i)}$
$J(\Theta)$
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1
$\Theta_{12}^{(1)}$
$\Theta_{11}^{(1)}$
$h_\Theta(x^{(i)}) \approx y^{(i)}$