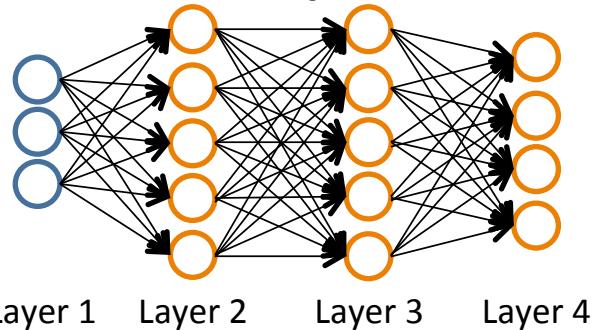


Machine Learning

Neural Networks: Learning

Cost function

Neural Network (Classification)



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

L = total no. of layers in network

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

Binary classification

$y = 0$ 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

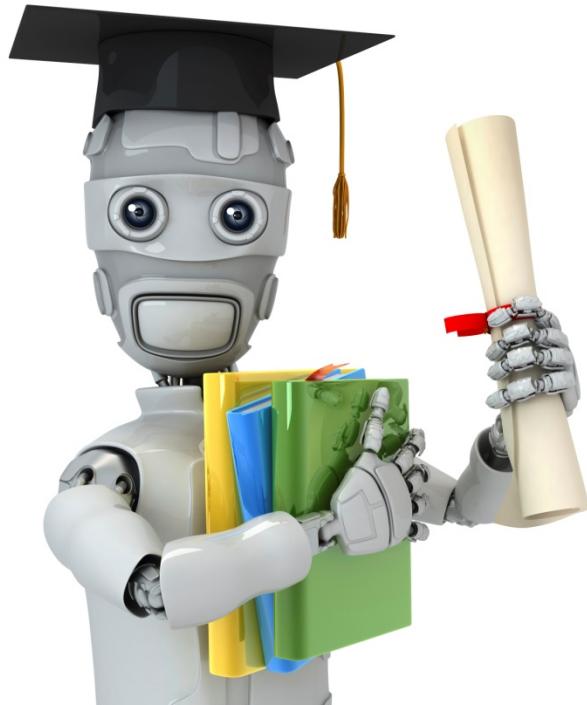
Logistic regression:

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

Neural network:

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

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



Machine Learning

Neural Networks: Learning

Backpropagation algorithm

Gradient computation

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

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

Need code to compute:

$$\rightarrow - \underline{J(\Theta)}$$
$$\rightarrow - \underline{\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)} \quad \leftarrow$$

$$\Theta_{ij}^{(l)} \in \mathbb{R}$$

Gradient computation

Given one training example $(\underline{x}, \underline{y})$:

Forward propagation:

$$\underline{a}^{(1)} = \underline{x}$$

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

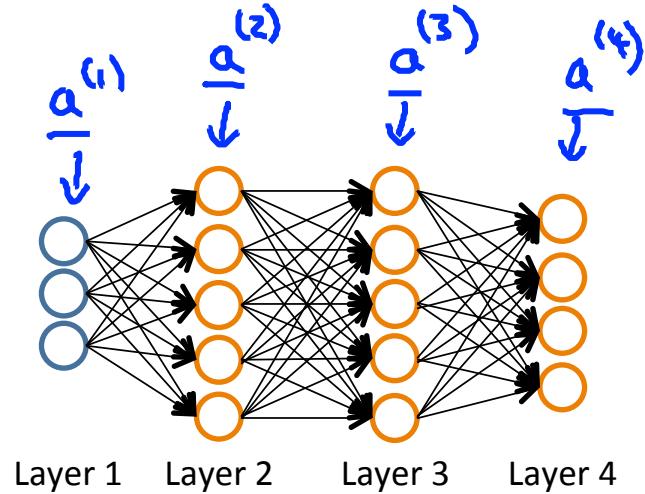
$$\rightarrow \underline{a}^{(2)} = g(\underline{z}^{(2)}) \quad (\text{add } \underline{a}_0^{(2)})$$

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

$$\rightarrow \underline{a}^{(3)} = g(\underline{z}^{(3)}) \quad (\text{add } \underline{a}_0^{(3)})$$

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

$$\rightarrow \underline{a}^{(4)} = \underline{h}_{\Theta}(\underline{x}) = g(\underline{z}^{(4)})$$

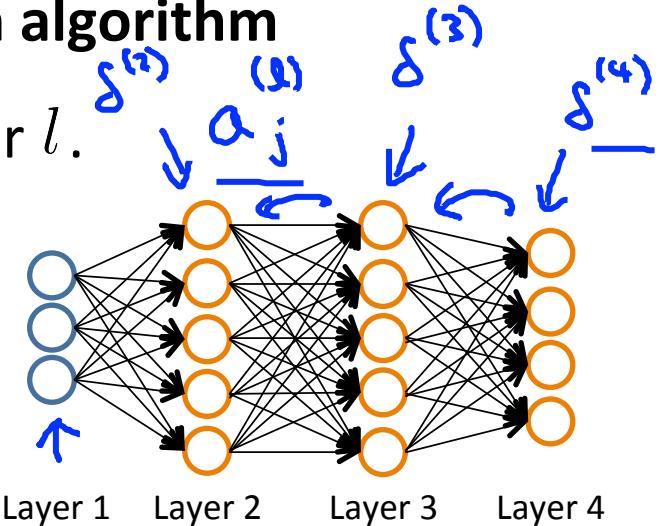


Gradient computation: Backpropagation algorithm

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

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

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



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

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

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

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

$$\frac{a^{(3)}}{a^{(2)}} * \frac{(1-a^{(3)})}{a^{(2)} * (1-a^{(2)})}$$

(ignoring λ ; if
 $\lambda = 0$)

Backpropagation algorithm

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

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

(use to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$)

For $i = 1$ to m ←

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

Set $\underline{a^{(1)}} = \underline{x^{(i)}}$

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

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

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

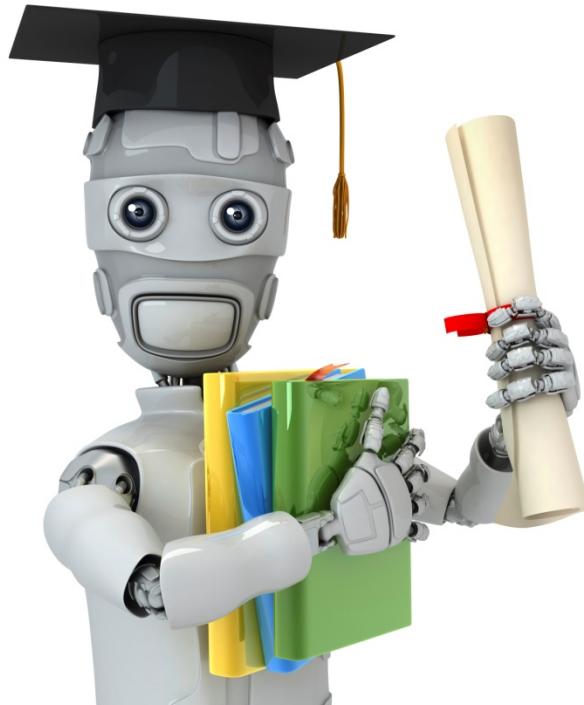
$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + \delta^{(l+1)} (a^{(l)})^T$.

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

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

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

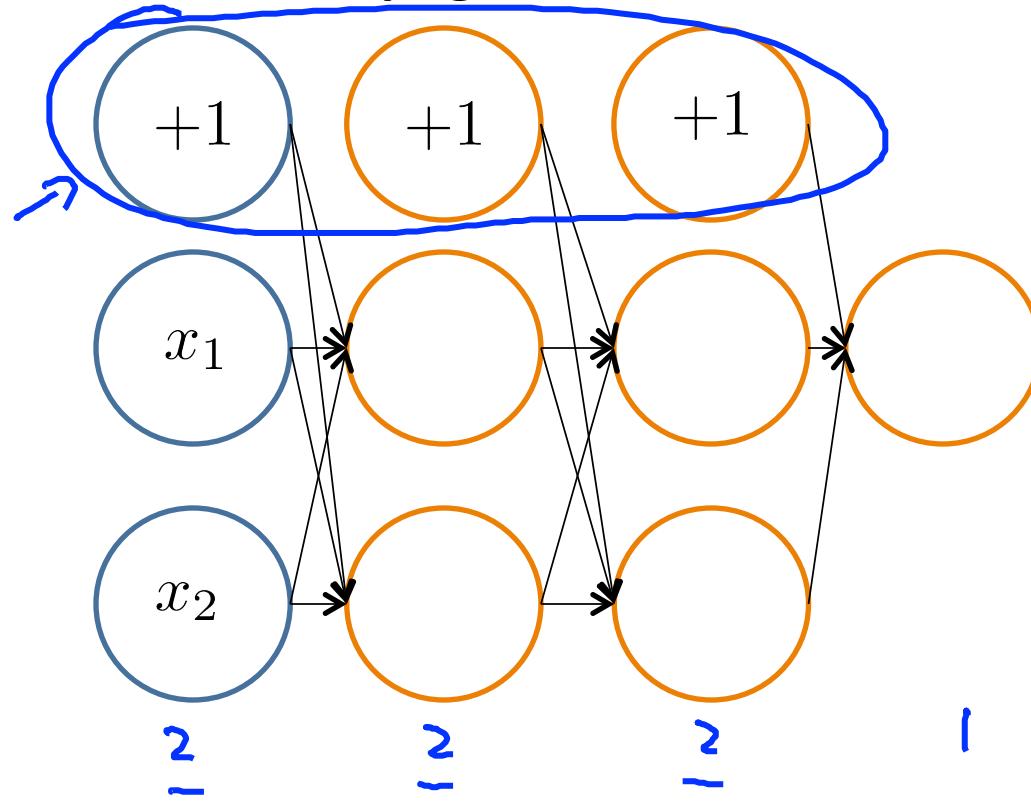


Machine Learning

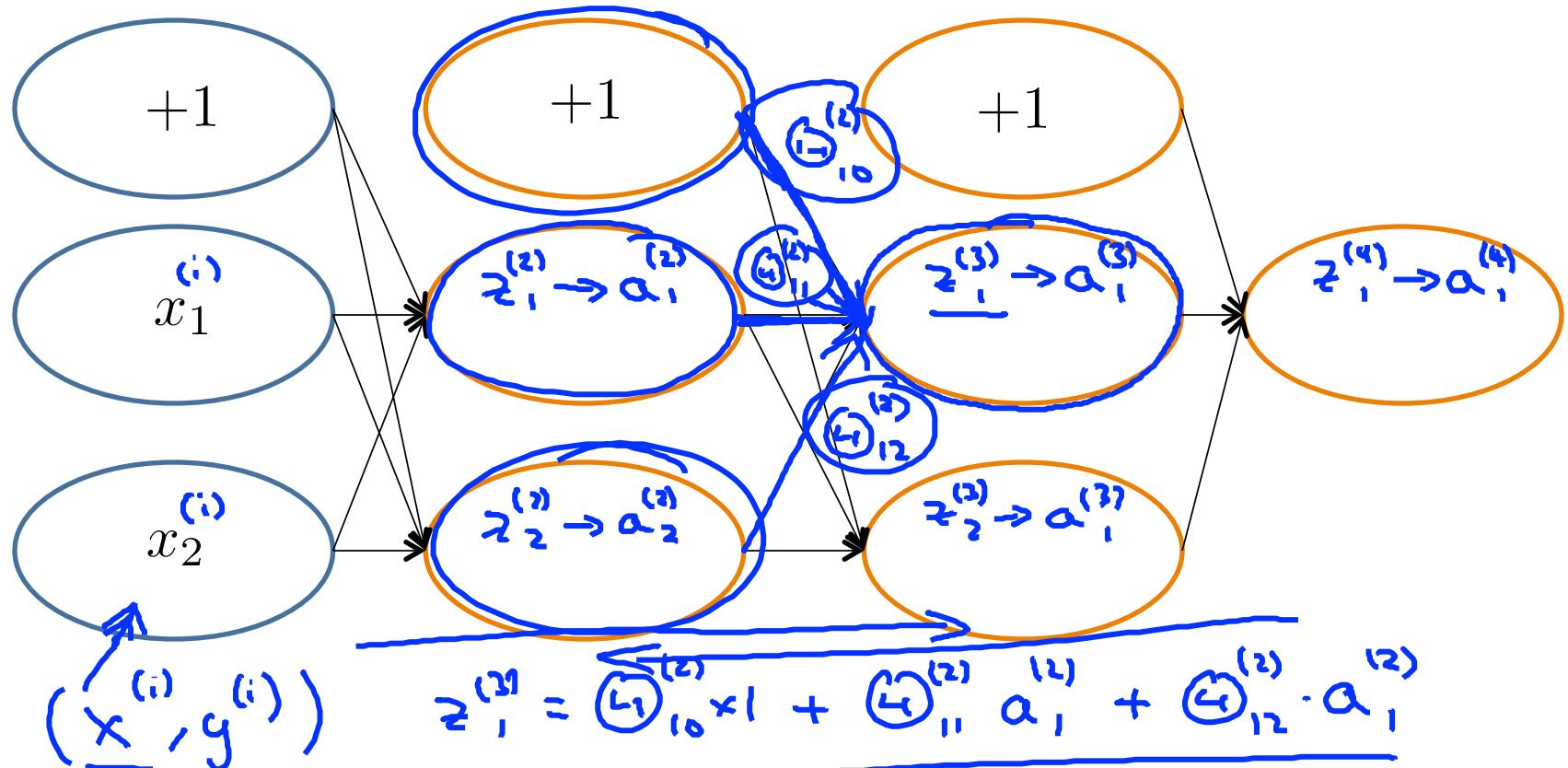
Neural Networks: Learning

Backpropagation intuition

Forward Propagation



Forward Propagation



What is backpropagation doing?

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

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

Focusing on a single example $x^{(i)}, y^{(i)}$, the case of 1 output unit, and ignoring regularization ($\lambda = 0$),

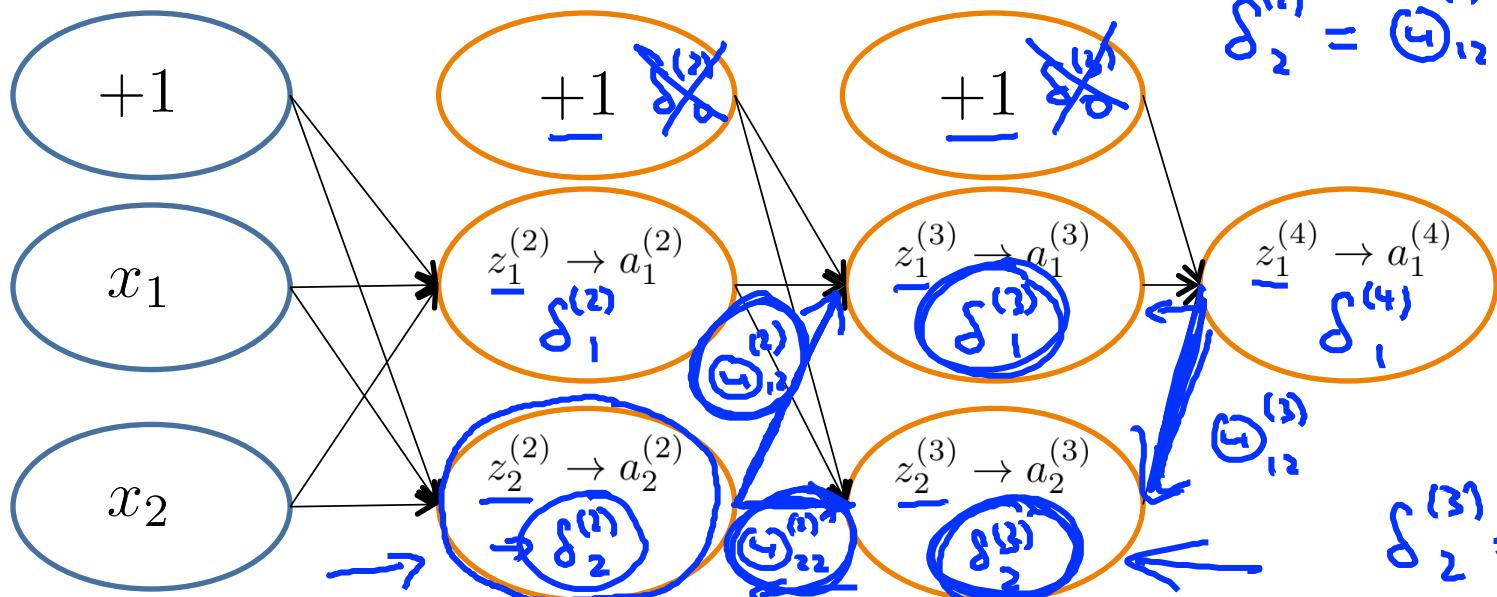
Note: Mistake on lecture, it is supposed to be $1 - h(x)$.

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

(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



$\rightarrow \delta_j^{(l)}$ = “error” of cost for $\underline{a}_j^{(l)}$ (unit j in layer l).

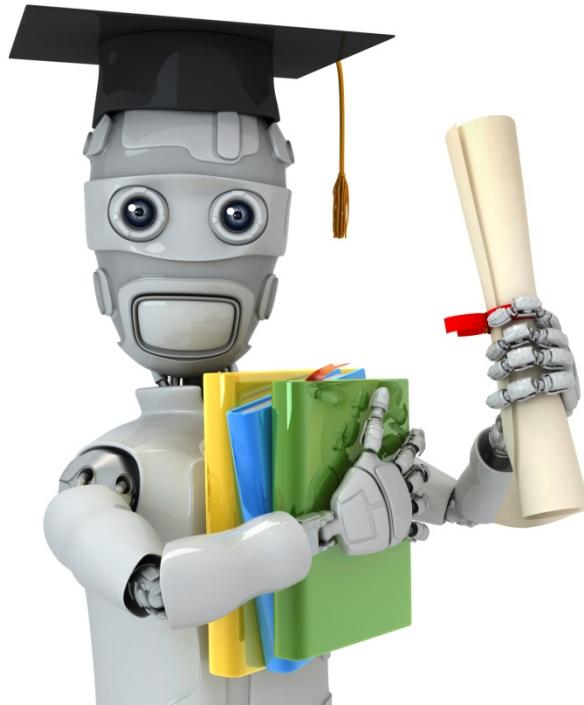
Formally, $\underline{\delta}_j^{(l)} = \frac{\partial \text{cost}(i)}{\partial z_j^{(l)}}$ (for $j \geq 0$), where

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

$$\delta_1^{(4)} = \underline{y}^{(i)} - \underline{a}_1^{(4)}$$

$$\delta_2^{(2)} = \underline{a}_{12}^{(2)} \delta_1^{(3)} + \underline{a}_{22}^{(2)} \delta_2^{(3)}$$

$$\delta_2^{(3)} = \underline{a}_{12}^{(3)} \cdot \delta_1^{(4)}$$



Machine Learning

Neural Networks: Learning

Implementation note: Unrolling parameters

Advanced optimization

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

Diagram annotations:

- An arrow points from the gradient parameter to the text \mathbb{R}^{n+1} .
- An arrow points from the theta parameter to the text \mathbb{R}^{n+1} (vectors).
- An arrow points from the initialTheta parameter to the text "Neural Network ($L=4$):".

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

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

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

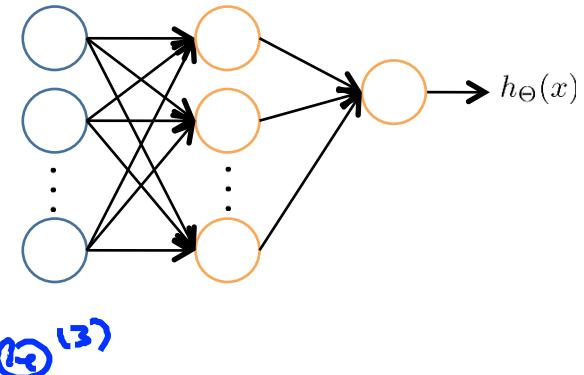
$D^{(1)} \in \mathbb{R}^{10 \times 11}$, $D^{(2)} \in \mathbb{R}^{10 \times 11}$, $D^{(3)} \in \mathbb{R}^{1 \times 11}$

```
→ 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

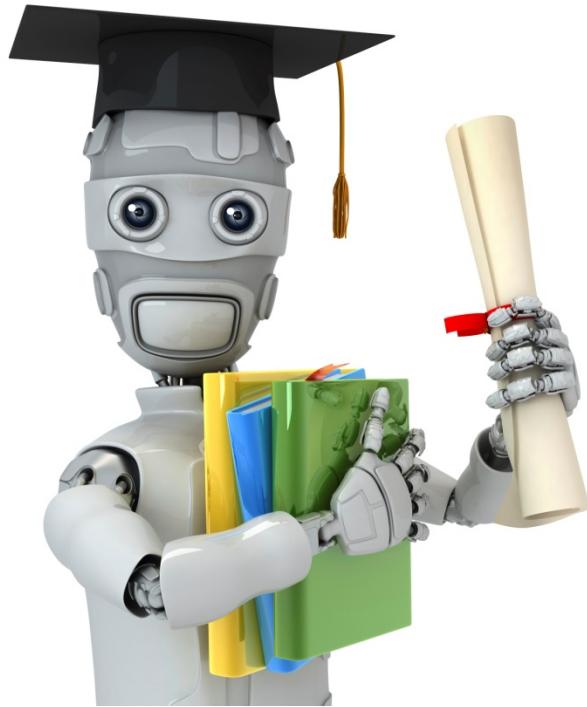
- Have initial parameters $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$.
- Unroll to get `initialTheta` to pass to
- `fminunc(@costFunction, initialTheta, options)`

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

→ From thetaVec, get $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$. reshape

→ Use forward prop/back prop to compute $D^{(1)}, D^{(2)}, D^{(3)}$ $J(\Theta)$
and $D_1^{(1)}, D_2^{(2)}, D_3^{(3)}$

Unroll $D_1^{(1)}, D_2^{(2)}, D_3^{(3)}$ to get gradientVec.

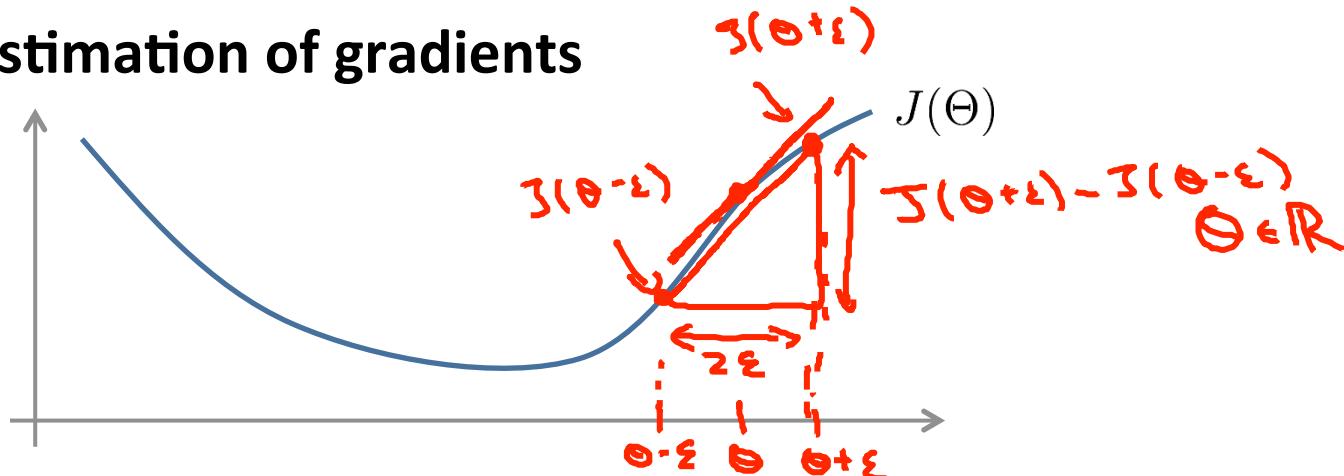


Machine Learning

Neural Networks: Learning

Gradient checking

Numerical estimation of gradients



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

$$\frac{J(\theta + \epsilon) - J(\theta - \epsilon)}{2\epsilon}$$

$$\epsilon = 10^{-4}$$

~~$$\frac{J(\theta + \epsilon) - J(\theta)}{\epsilon}$$~~

Implement: gradApprox = $(J(\text{theta} + \text{EPSILON}) - J(\text{theta} - \text{EPSILON})) / (2 * \text{EPSILON})$

Parameter vector θ

- $\theta \in \mathbb{R}^n$ (E.g. θ is “unrolled” version of $\underline{\Theta^{(1)}}, \underline{\Theta^{(2)}}, \underline{\Theta^{(3)}}$)
- $\theta = [\theta_1, \theta_2, \theta_3, \dots, \theta_n]$
- $\frac{\partial}{\partial \theta_1} J(\theta) \approx \frac{J(\theta_1 + \epsilon, \theta_2, \theta_3, \dots, \theta_n) - J(\theta_1 - \epsilon, \theta_2, \theta_3, \dots, \theta_n)}{2\epsilon}$
- $\frac{\partial}{\partial \theta_2} J(\theta) \approx \frac{J(\theta_1, \theta_2 + \epsilon, \theta_3, \dots, \theta_n) - J(\theta_1, \theta_2 - \epsilon, \theta_3, \dots, \theta_n)}{2\epsilon}$
- ⋮
- $\frac{\partial}{\partial \theta_n} J(\theta) \approx \frac{J(\theta_1, \theta_2, \theta_3, \dots, \theta_n + \epsilon) - J(\theta_1, \theta_2, \theta_3, \dots, \theta_n - \epsilon)}{2\epsilon}$

```

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;

```

$\frac{\partial}{\partial \theta_j} J(\theta)$.

Check that gradApprox \approx DVec ←

From back prop.

$$\begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_i + \epsilon \\ \vdots \\ \theta_n \end{bmatrix} \rightarrow \theta_0 \dots \theta_n$$

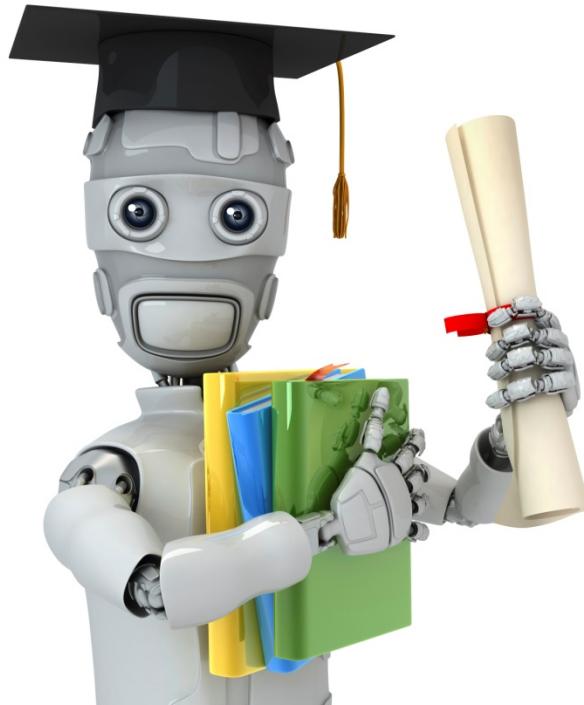
Implementation Note:

- - 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 very slow.

DVec
 $\delta^{(1)}, \delta^{(2)}, \delta^{(3)}$



Machine Learning

Neural Networks: Learning

Random initialization

Initial value of Θ

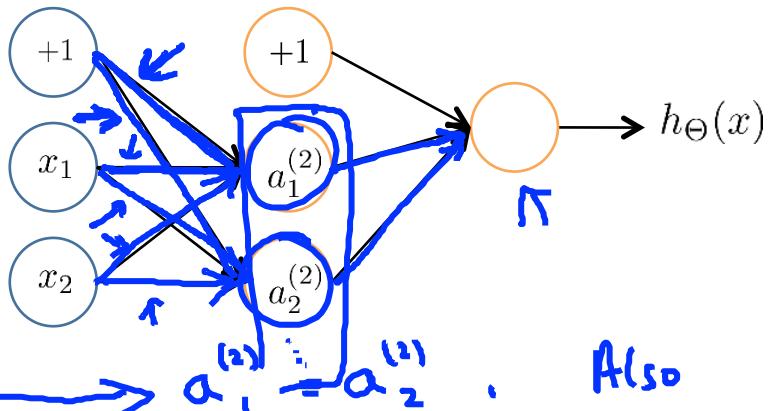
For gradient descent and advanced optimization method, need initial value for Θ .

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

Consider gradient descent

Set initialTheta = zeros(n,1) ?

Zero initialization



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

Also $\delta_i^{(l)} = \delta_j^{(l)}$.

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

$$\underline{\Theta_{01}^{(l)}} = \underline{\Theta_{02}^{(l)}}$$

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

$$\underline{\underline{\Theta_{01}^{(l)}}} = \underline{\underline{\Theta_{02}^{(l)}}}$$

Random initialization: Symmetry breaking

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

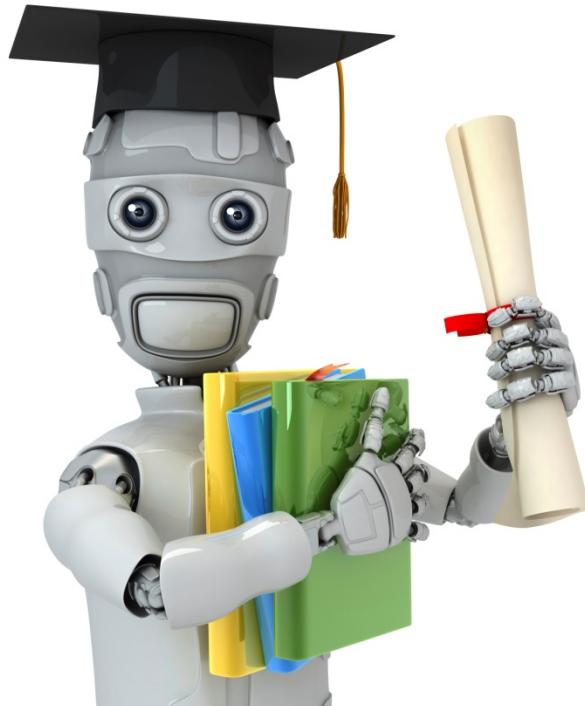
E.g.

Random 10×11 matrix (betw. 0 and 1)

→ Theta1 = $\text{rand}(10, 11) * (2 * \text{INIT_EPSILON})$
- INIT_EPSILON;

$[-\epsilon, \epsilon]$

→ Theta2 = $\text{rand}(1, 11) * (2 * \text{INIT_EPSILON})$
- INIT_EPSILON;



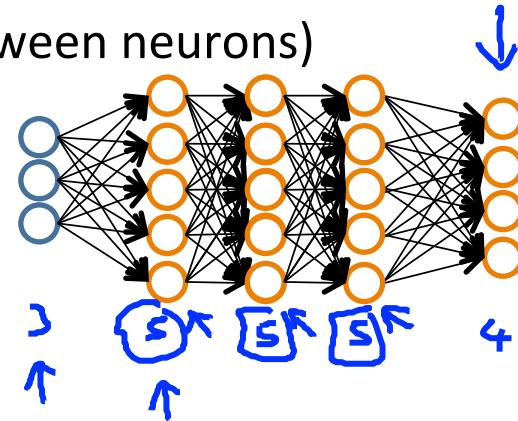
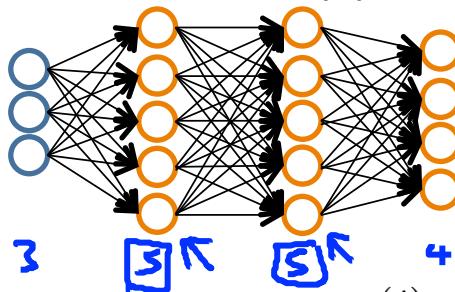
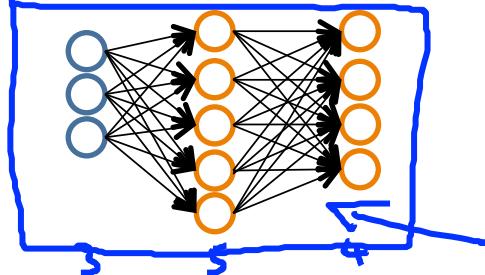
Machine Learning

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)]

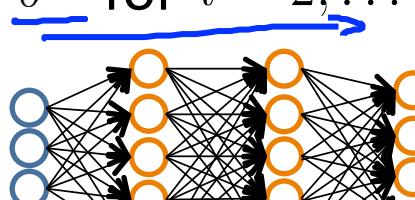
$$y \in \{1, 2, 3, \dots, 10\}$$

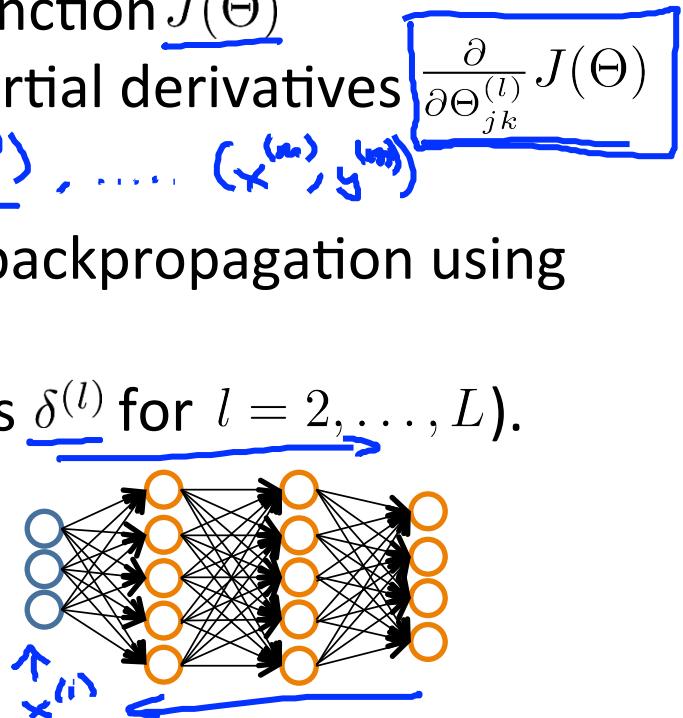
~~$y \in S$~~

$$y = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \text{ or } \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

\leftarrow

Training a neural network

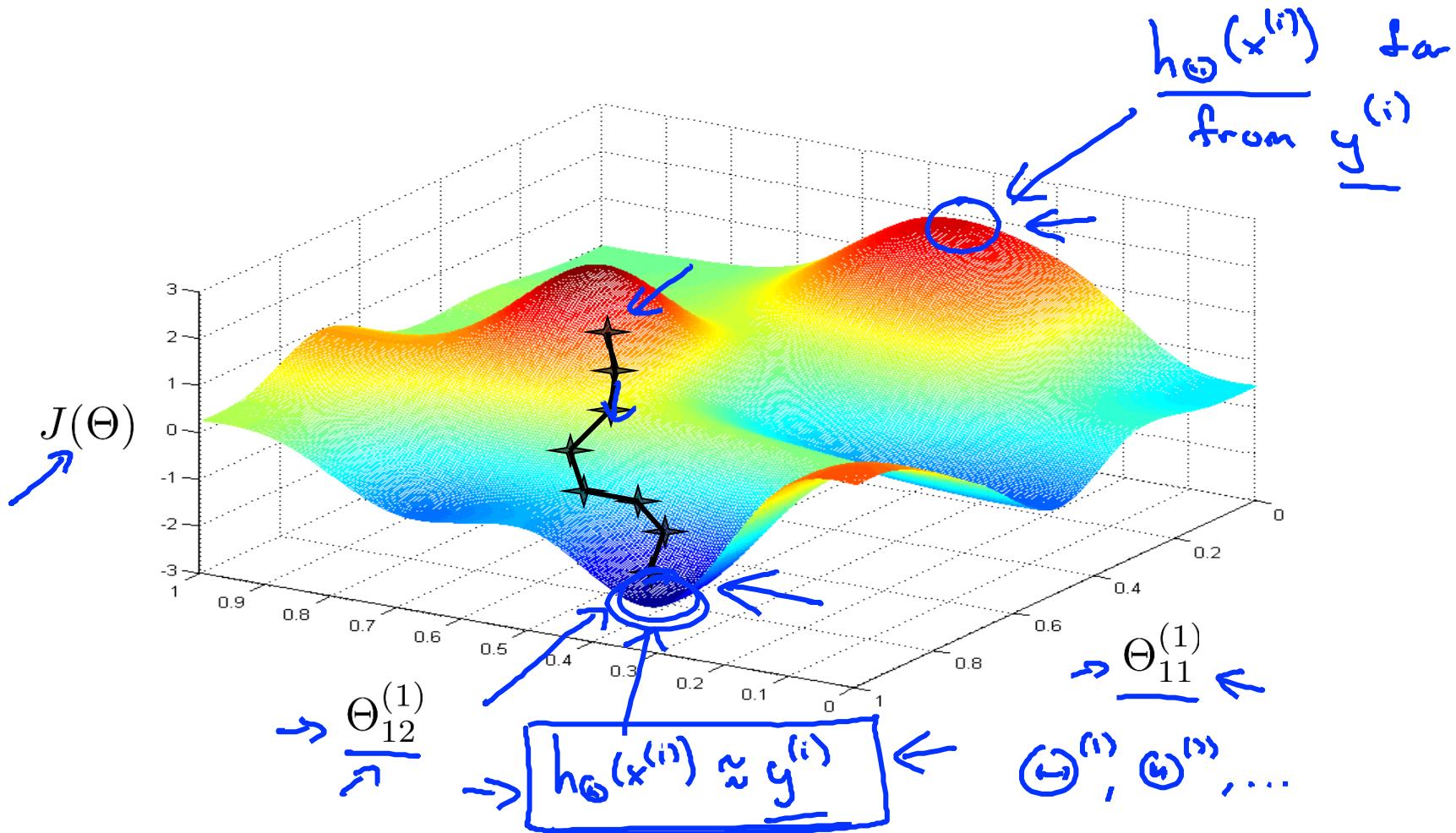
- 1. Randomly initialize weights
 - 2. Implement forward propagation to get $h_{\Theta}(x^{(i)})$ for any $\underline{x}^{(i)}$
 - 3. Implement code to compute cost function $J(\Theta)$
 - 4. Implement backprop to compute partial derivatives $\frac{\partial}{\partial \Theta_j^{(l)}} J(\Theta)$
 - for $i = 1:m$ { $(\underline{x}^{(i)}, y^{(i)})$ $(\underline{x}^{(i)}, y^{(i)})$, ..., $(\underline{x}^{(i)}, y^{(i)})$
 - Perform forward propagation and backpropagation using example $(x^{(i)}, y^{(i)})$
 - Get activations $a^{(l)}$ and delta terms $\delta^{(l)}$ for $l = 2, \dots, L$.
 - $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l)} (a^{(l)})^T$
 - ...
}
 - ...
compute $\frac{\partial}{\partial \Theta_{j,k}^{(l)}} J(\Theta)$.

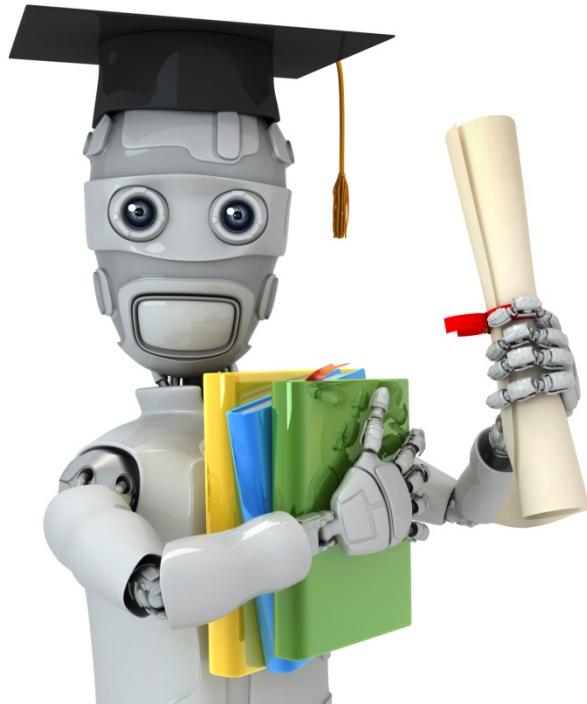


Training a neural network

- 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

$$\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta) \quad \text{non-convex.}$$

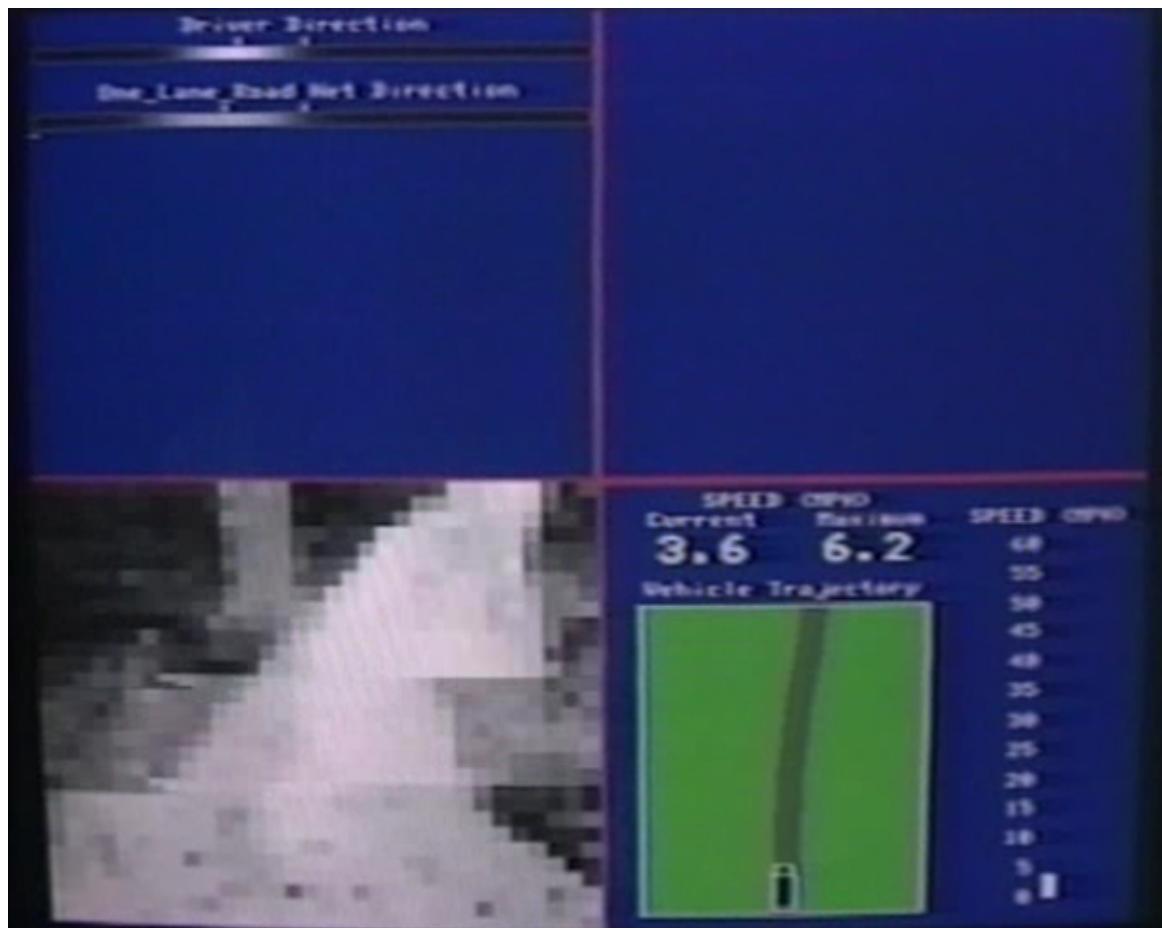




Machine Learning

Neural Networks: Learning

Backpropagation
example: Autonomous
driving (optional)



[Courtesy of Dean Pomerleau]