Neural Networks: Learning

Cost function

Machine Learning

Neural Network (Classification)

Layer 1   Layer 2   Layer 3   Layer 4

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

$L =$ 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

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

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

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

Need code to compute:

$\rightarrow$ - $J(\Theta)$

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

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

Gradient computation

Given one training example (x, y):

Forward propagation:

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

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

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

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

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

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

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

Layer 1   Layer 2   Layer 3   Layer 4

Gradient computation: Backpropagation algorithm

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

For each output unit (layer L = 4)

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

$(h_\Theta(x))_j$ $\delta^{(4)} = a^{(4)} - y$

Layer 1   Layer 2   Layer 3   Layer 4

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

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

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

$a^{(3)} .* (1 - a^{(3)})$

$a^{(2)} .* (1 - a^{(2)})$

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

(ignore λ; 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).

$\left(\text{used to compute } \dfrac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) \right)$

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

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

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

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

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

    $\boxed{\triangle_{ij}^{(l)} := \triangle_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}}$ $\leftarrow$ $\boxed{i \; j}$

$\triangle^{(l)} := \triangle^{(l)} + \delta^{(l+1)} \left(a^{(l)} \right)^{\mathsf{T}}$.

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

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

Neural Networks: Learning

Backpropagation intuition

Machine Learning

Forward Propagation

Forward Propagation

$+1$

$x_1^{(i)}$

$x_2^{(i)}$

$+1$

$+1$

$z_1^{(2)} \to a_1^{(2)}$

$z_2^{(2)} \to a_2^{(2)}$

$\Theta_{10}^{(2)}$

$\Theta_{11}^{(2)}$

$\Theta_{12}^{(2)}$

$z_1^{(3)} \to a_1^{(3)}$

$z_2^{(3)} \to a_2^{(3)}$

$z_1^{(4)} \to a_1^{(4)}$

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

$$z_1^{(3)} = \Theta_{10}^{(2)} x_1 + \Theta_{11}^{(2)} a_1 + \Theta_{12}^{(2)} \cdot a_1^{(2)}$$

Andrew Ng

What is backpropagation doing?

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

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

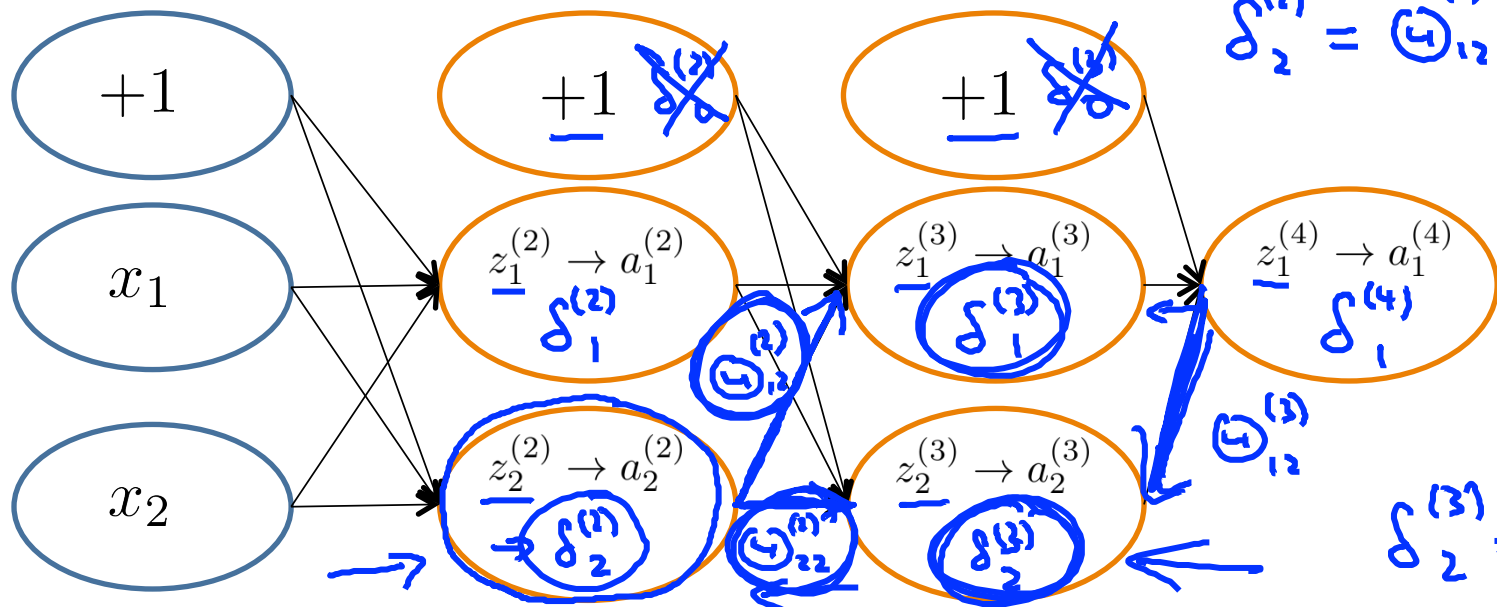Focusing on a single example $x^{(i)}$, $y^{(i)}$, the case of 1 output unit, and ignoring regularization ($\lambda = 0$),

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

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

1-

(Think of $\text{cost}(i) \approx (h_\Theta(x^{(i)}) - y^{(i)})^2$)
I.e. how well is the network doing on example i?

$y^{(i)}$

Forward Propagation

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

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

$+1$ $\delta^{(2)}$

$+1$ $\delta^{(3)}$

$z_1^{(2)} \to a_1^{(2)}$ $\delta_1^{(2)}$

$z_1^{(3)} \to a_1^{(3)}$ $\delta_1^{(3)}$

$z_1^{(4)} \to a_1^{(4)}$ $\delta_1^{(4)}$

$\Theta_{12}^{(2)}$

$z_2^{(2)} \to a_2^{(2)}$ $\delta_2^{(2)}$

$\Theta_{22}^{(2)}$

$z_2^{(3)} \to a_2^{(3)}$ $\delta_2^{(3)}$

$\Theta_{12}^{(3)}$

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

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

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

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

Andrew Ng

Neural Networks: Learning

Implementation note: Unrolling parameters

Machine Learning

Advanced optimization

```
function [jVal, gradient] = costFunction(theta)
    ...
```

$\mathbb{R}^{n+1}$

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

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

Neural Network (L=4):

$\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ - matrices (**Theta1, Theta2, Theta3**)
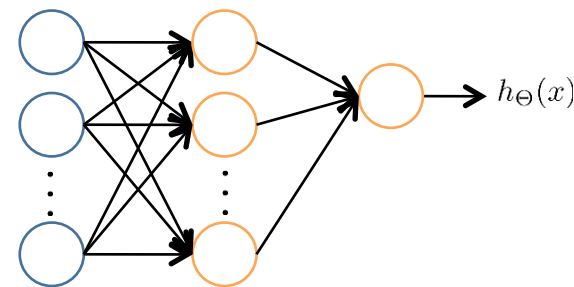
$D^{(1)}, D^{(2)}, D^{(3)}$ - matrices (**D1, D2, D3**)

"Unroll" into vectors

Example

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

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

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

$h_\Theta(x)$

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

Neural Networks: Learning

Gradient checking

Machine Learning

Numerical estimation of gradients

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

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

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

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

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

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

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

$\vdots$

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

```
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;
```

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

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

Check that gradApprox ≈ DVec ←

From backprop.

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:
Learning

Random
initialization

Machine Learning

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

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

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

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

$$\Theta_{01}^{(1)} = \Theta_{02}^{(1)}$$

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

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

Random initialization: Symmetry breaking

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

E.g.

random 10×11 matrix (betw. 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: Learning

Putting it together

Machine Learning

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)

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

$$y$$

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

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

```
for i = 1:m {
```
$(x^{(1)}, y^{(1)}) \quad (x^{(2)}, y^{(2)}), \ldots (x^{(m)}, y^{(m)})$

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

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

$\}$

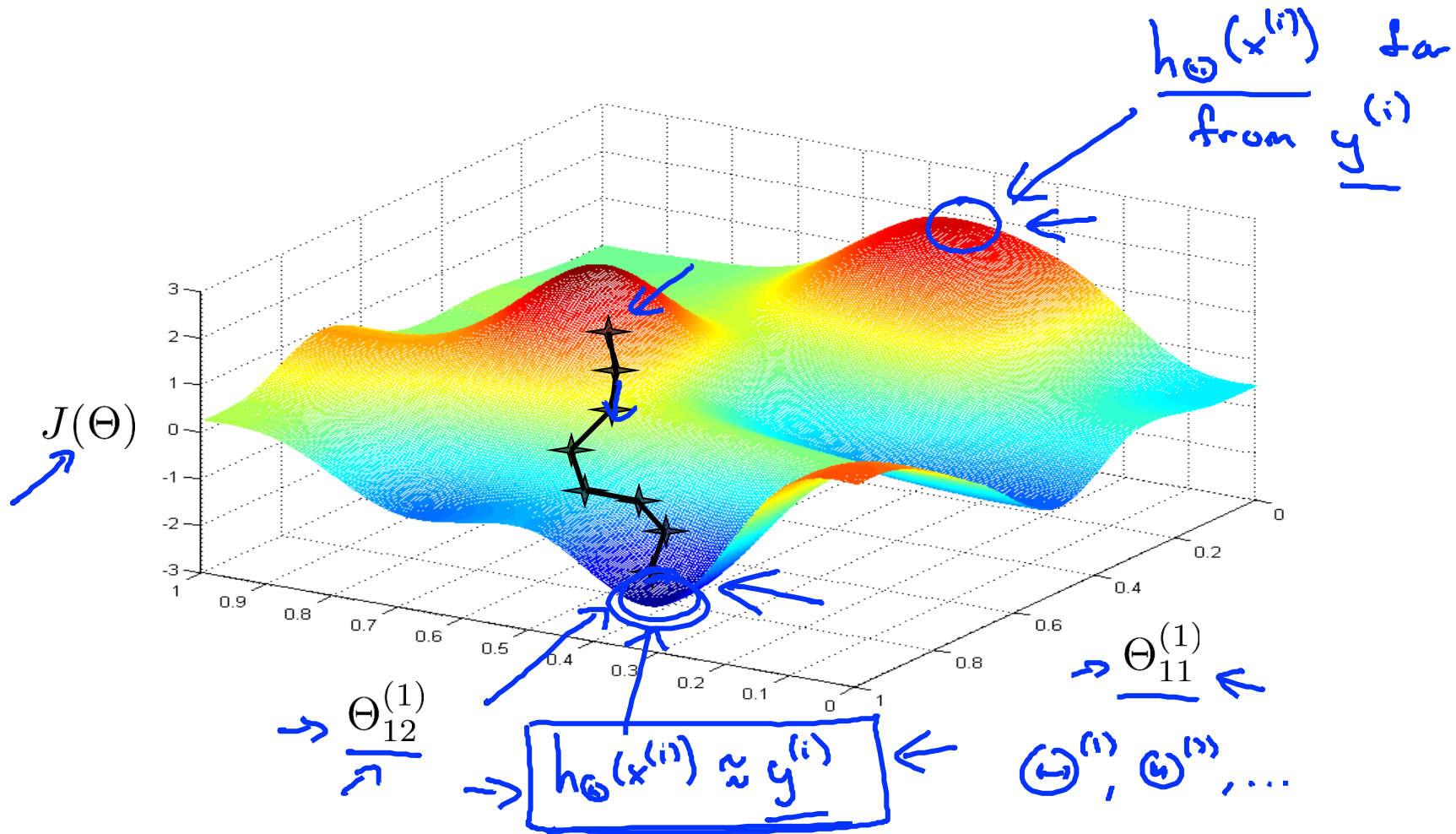compute $\dfrac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$.

$x^{(i)}$

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

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

$$J(\Theta) \quad - \quad \text{non-convex.}$$

$\dfrac{h_\Theta(x^{(i)})}{\text{from}}\ \ \text{far}\ \ y^{(i)}$

$J(\Theta)$

$\dfrac{\Theta^{(1)}_{12}}{}$

$h_\Theta(x^{(i)}) \approx y^{(i)}$

$\Theta^{(1)}_{11}$

$(\cdot)^{(1)}, (\cdot)^{(1)}, \ldots$

Andrew Ng

Machine Learning

Neural Networks: Learning

Backpropagation example: Autonomous driving (optional)

[Courtesy of Dean Pomerleau]