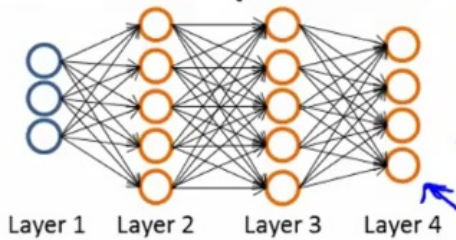


1. 神经网络符号约定：L=网络总层数；sl=网络中单元数。

Neural Network (Classification)



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

→ $L =$ total no. of layers in network $L = 4$

→ $s_l =$ no. of units (not counting bias unit) in layer l $s_1 = 3, s_2 = 5, s_4 = s_L = 4$

Binary classification

$y = 0$ or 1 ←

1 output unit ←

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

$$s_L = 1, K = 1 \leftarrow$$



Multi-class classification (K classes)

$y \in \mathbb{R}^K$ E.g. $\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ ←
pedestrian car motorcycle truck

K output units

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

$$s_L = K \quad (K \geq 3)$$

K输出单元代价函数：

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:

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

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

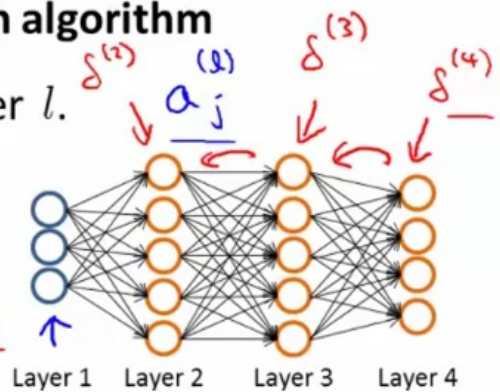
$$+ \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2$$



2. 反向传播(Backpropagation)算法

Gradient computation: Backpropagation algorithm

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



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

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

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

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

$$\frac{\partial J(\Theta)}{\partial \Theta_{ij}^{(2)}} = a_j^{(2)} \delta_i^{(3)}$$

$$\frac{\partial J(\Theta)}{\partial \Theta_{ij}^{(2)}} = a_j^{(2)} \delta_i^{(3)} \quad (\text{ignoring } \lambda; \text{ if } \lambda = 0) \leftarrow$$

Backpropagation algorithm

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

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

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

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

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

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

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

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

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

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

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

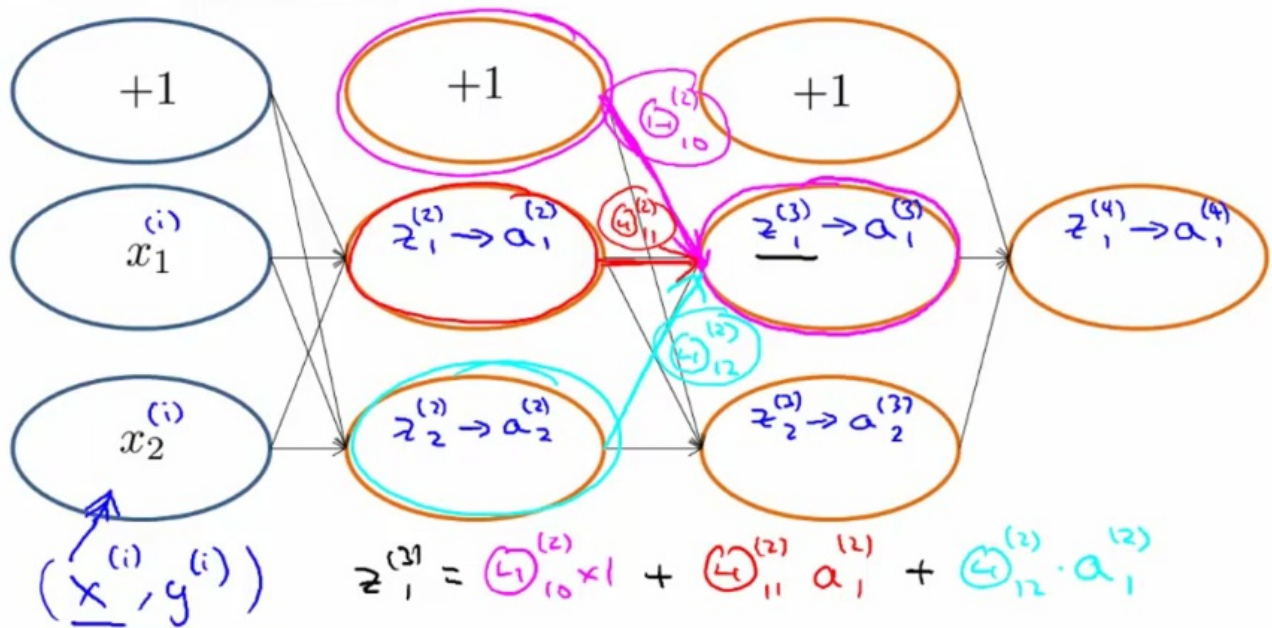
代价函数对

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

可以看出来，it is really 复杂.

直观理解：前向传播：

Forward Propagation



What is backpropagation doing?

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 ($\lambda = 0$),

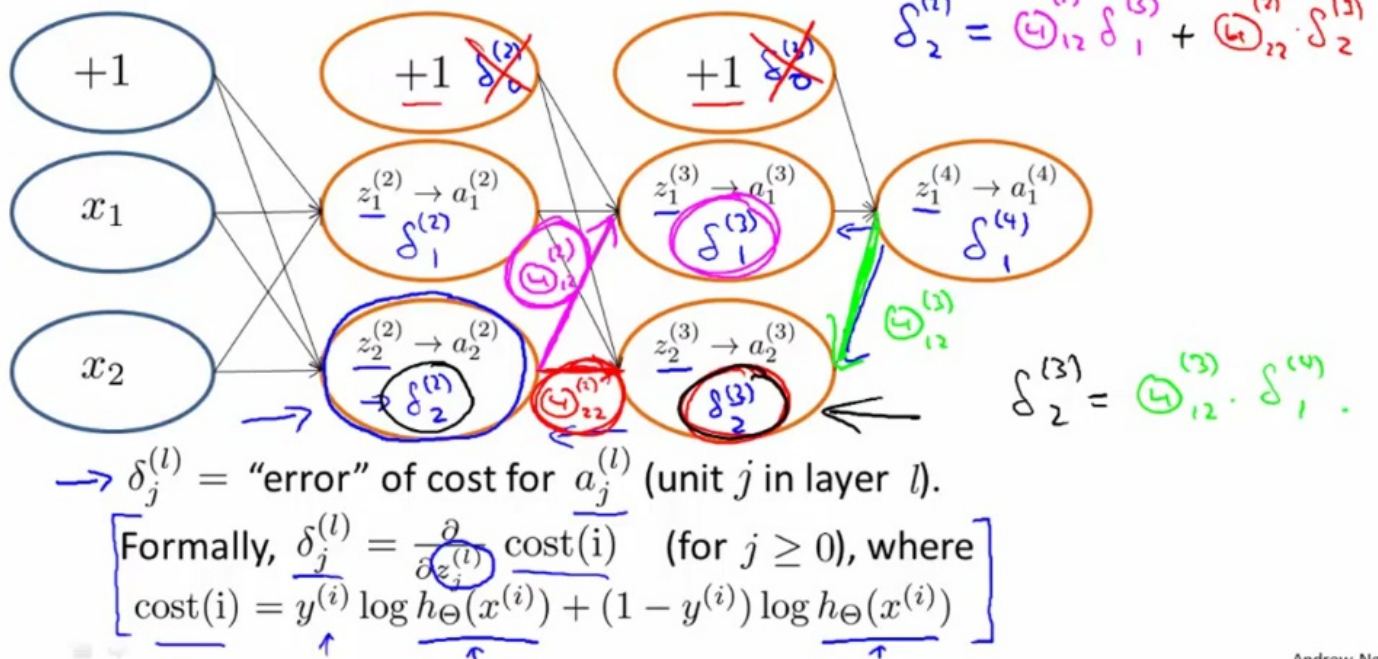
$$\text{cost}(i) = y^{(i)} \log h_{\Theta}(x^{(i)}) + (1 - y^{(i)}) \log h_{\Theta}(x^{(i)})$$

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

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

差的平方

Forward Propagation



Andrew Ng

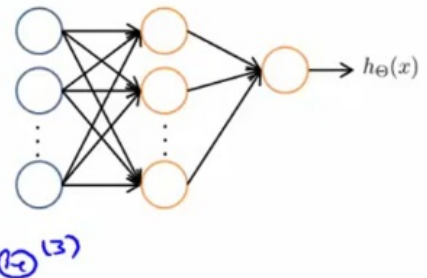
高级优化：把矩阵展开成向量：展开成列向量，还原时候使用reshape还原。

Example

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

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

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



$$\rightarrow \text{thetaVec} = [\text{Theta1}(:); \text{Theta2}(:); \text{Theta3}(:)];$$

$$\rightarrow \text{DVec} = [D1(:); D2(:); D3(:)];$$

$$\text{Theta1} = \text{reshape}(\text{thetaVec}(1:110), 10, 11);$$

$$\rightarrow \text{Theta2} = \text{reshape}(\text{thetaVec}(111:220), 10, 11);$$

$$\rightarrow \text{Theta3} = \text{reshape}(\text{thetaVec}(221:231), 1, 11);$$

学习参数的过程：

Learning Algorithm

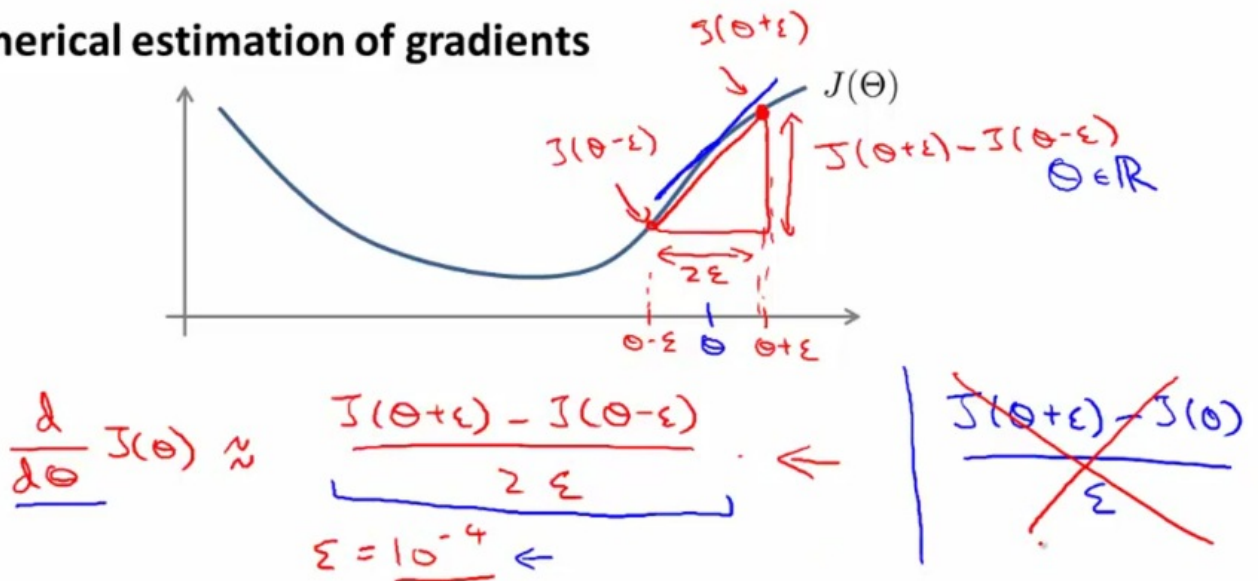
- Have initial parameters $\theta^{(1)}, \theta^{(2)}, \theta^{(3)}$.
- Unroll to get `initialTheta` to pass to
- `fminunc(@costFunction, initialTheta, options)`

```
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`.

梯度检验 (Gradient Checking) : 估计导数 :

Numerical estimation of gradients



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

Andrew Ng

向量形式 :

Parameter vector θ

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

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

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

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

⋮

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

然后除以2

代码：（比较反向传播方法得到的梯度和估计的梯度）

```
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;
```

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

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

Check that $\text{gradApprox} \approx \text{DVec}$

↑
From backprop.

使用反向传播的方法计算导数要比估计导数的方法快的多，所以在检验梯度计算的是正确之后，在实现中要关闭梯度估计，否则代码会运行的慢的多。

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:

$\begin{matrix} \downarrow & \downarrow & \downarrow \\ f^{(4)} & f^{(3)} & f^{(2)} \end{matrix}$ \rightarrow DVec

- - 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.

初始化参数的选择：随机初始化

Initial value of Θ

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

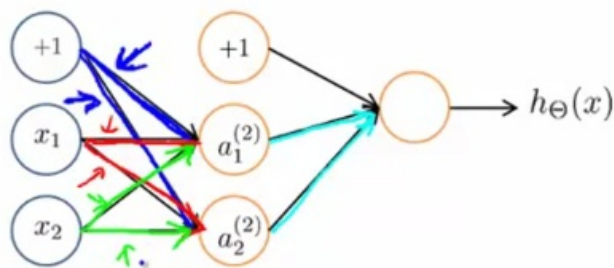
```
optTheta = fminunc(@costFunction,  
    initialTheta, options)
```

Consider gradient descent

Set initialTheta = zeros(n,1) ?

虽然在逻辑回归的方法中可以初始化theta为全0，但是训练神经网络的时候，不能初始化权重为0.

Zero initialization



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

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

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

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

也是同样的情况

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

使用随机初始化的方式：----打破权重对称性（相同权重的情况）

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;` [-ε, ε]

Random 10x11 matrix (betw. 0 and 1)

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

注意这里的epsilon和梯度检验中的epsilon没有关系。

Consider this procedure for initializing the parameters of a neural network:

1. Pick a random number $r = \text{rand}(1,1) * (2 * \text{INIT_EPSILON}) - \text{INIT_EPSILON}$;
2. Set $\Theta_{ij}^{(l)} = r$ for all i, j, l .

Does this work?

- ☐ Yes, because the parameters are chosen randomly.
- ☐ Yes, unless we are unlucky and get $r=0$ (up to numerical precision).
- ☐ Maybe, depending on the training set inputs $x(i)$.
- ☒ No, because this fails to break symmetry.

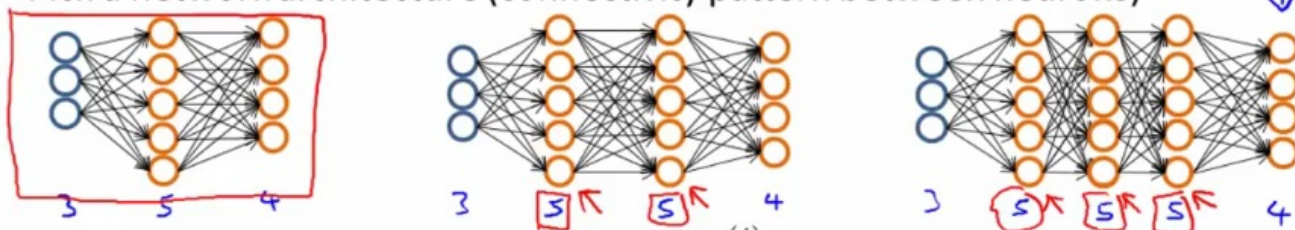
Correct Response

训练神经网络过程综述：

(1) 选择网络架构：

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)

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

~~$y=5$~~

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

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

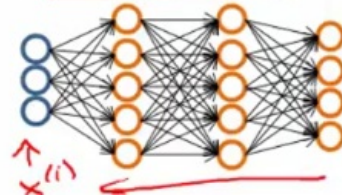
→ for $i = 1:m$ { $(x^{(1)}, y^{(1)})$ $(x^{(2)}, y^{(2)})$, ..., $(x^{(m)}, y^{(m)})$ }
 → Perform forward propagation and backpropagation using example $(x^{(i)}, y^{(i)})$

(Get activations $a^{(l)}$ and delta terms $\delta^{(l)}$ for $l = 2, \dots, L$).

$$\Delta^{(2)} := \Delta^{(2)} + \delta^{(2)} (a^{(1)})^T$$

...

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

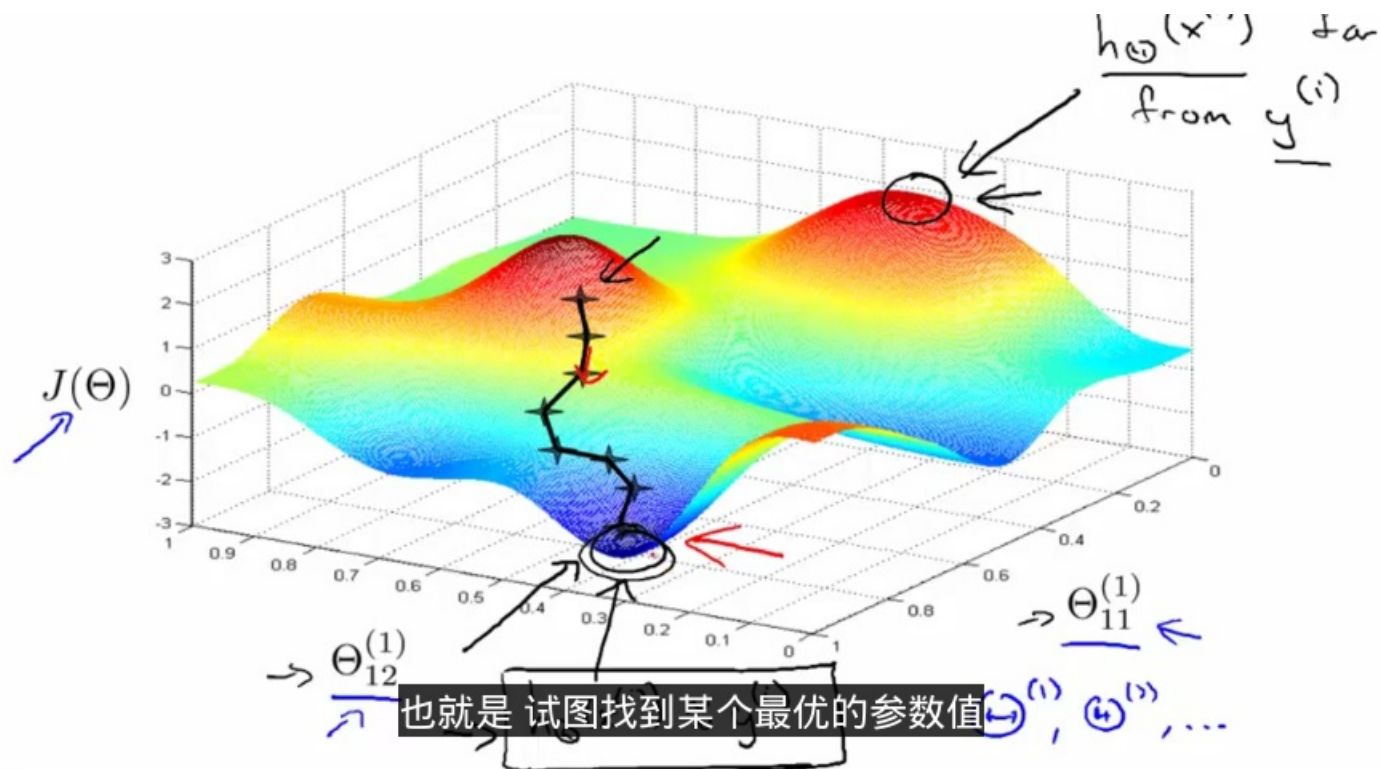


Training a neural network

- 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.



反向传播的实现：

```

1. for i = 1:m
2.     a1 = [1; X(i,:)'];
3.     z2 = Theta1 * a1;
4.     a2 = [1; sigmoid(z2)];
5.     z3 = Theta2 * a2;
6.     a3 = sigmoid(z3);
7.     yy = zeros(num_labels, 1);
8.     yy(y(i)) = 1;
9.     delta3 = a3 - yy;
10.    delta2 = Theta2(:,2:end)' * delta3 .* sigmoidGradient(z2);
11.    Theta2_grad = Theta2_grad + delta3 * (a2');
12.    Theta1_grad = Theta1_grad + delta2 * (a1');
13. end
14. Theta1_grad = Theta1_grad ./ m;
15. Theta2_grad = Theta2_grad ./ m;

```

完整代价函数代码：

```

1. function [J grad] = nnCostFunction(nn_params, ...
2.                                     input_layer_size, ...
3.                                     hidden_layer_size, ...
4.                                     num_labels, ...
5.                                     X, y, lambda)
6. %NNCOSTFUNCTION Implements the neural network cost function for a two layer
7. %neural network which performs classification
8. % [J grad] = NNCOSTFUNCTION(nn_params, hidden_layer_size, num_labels, ...
9. % X, y, lambda) computes the cost and gradient of the neural network. The
10. % parameters for the neural network are "unrolled" into the vector
11. % nn_params and need to be converted back into the weight matrices.
12. %
13. % The returned parameter grad should be a "unrolled" vector of the
14. % partial derivatives of the neural network.
15. %
16. %
17. % Reshape nn_params back into the parameters Theta1 and Theta2, the weight matrices
18. % for our 2 layer neural network
19. Theta1 = reshape(nn_params(1:hidden_layer_size * (input_layer_size + 1)), ...
20.                  hidden_layer_size, (input_layer_size + 1));
21.
22. Theta2 = reshape(nn_params((1 + (hidden_layer_size * (input_layer_size + 1))):end), ...
23.                  num_labels, (hidden_layer_size + 1));
24.
25. % Setup some useful variables
26. m = size(X, 1);

```



```

27.
28. % You need to return the following variables correctly
29. J = 0;
30. Theta1_grad = zeros(size(Theta1));
31. Theta2_grad = zeros(size(Theta2));
32.
33. % ===== YOUR CODE HERE =====
34. % Instructions: You should complete the code by working through the
35. % following parts.
36. %
37. % Part 1: Feedforward the neural network and return the cost in the
38. % variable J. After implementing Part 1, you can verify that your
39. % cost function computation is correct by verifying the cost
40. % computed in ex4.m
41. %
42. % Part 2: Implement the backpropagation algorithm to compute the gradients
43. % Theta1_grad and Theta2_grad. You should return the partial derivatives of
44. % the cost function with respect to Theta1 and Theta2 in Theta1_grad and
45. % Theta2_grad, respectively. After implementing Part 2, you can check
46. % that your implementation is correct by running checkNNGradients
47. %
48. % Note: The vector y passed into the function is a vector of labels
49. % containing values from 1..K. You need to map this vector into a
50. % binary vector of 1's and 0's to be used with the neural network
51. % cost function.
52. %
53. % Hint: We recommend implementing backpropagation using a for-loop
54. % over the training examples if you are implementing it for the
55. % first time.
56. %
57. % Part 3: Implement regularization with the cost function and gradients.
58. %
59. % Hint: You can implement this around the code for
60. % backpropagation. That is, you can compute the gradients for
61. % the regularization separately and then add them to Theta1_grad
62. % and Theta2_grad from Part 2.
63. %
64.
65. a2 = sigmoid([ones(m, 1), X]*Theta1');
66. h = sigmoid([ones(m, 1), a2]*Theta2');
67. for k = 1:num_labels
68.     J = J - [(y==k)', ((1-(y==k)))']*[log(h(:,k));log(1-h(:,k))]./m;
69. end
70.
71. t1 = ones(hidden_layer_size, input_layer_size);
72. t2 = ones(num_labels, hidden_layer_size);
73. t1 = Theta1(:,2:end).^2;
74. t2 = Theta2(:,2:end).^2;
75. J = J + (sum(t1(:))+sum(t2(:)))*lambda/(2*m);
76.
77. a1 = zeros(input_layer_size+1, 1);
78. z2 = zeros(hidden_layer_size, 1);
79. a2 = zeros(hidden_layer_size+1, 1);
80. z3 = zeros(num_labels, 1);
81. a3 = zeros(num_labels, 1);
82. yy = zeros(num_labels, 1);
83. delta3 = zeros(num_labels, 1);
84. delta2 = zeros(hidden_layer_size, 1);
85.
86. for i = 1:m
87.     a1 = [1; X(i,:)'];
88.     z2 = Theta1 * a1;
89.     a2 = [1; sigmoid(z2)];
90.     z3 = Theta2 * a2;
91.     a3 = sigmoid(z3);
92.     yy(y(i)) = 1;
93.     delta3 = a3 - yy;
94.     delta2 = Theta2(:,2:end)'*delta3.*sigmoidGradient(z2);
95.     Theta2_grad = Theta2_grad + delta3*(a2');
96.     Theta1_grad = Theta1_grad + delta2*(a1');
97. end
98.
99. Theta1_grad = Theta1_grad ./ m;
100. Theta2_grad = Theta2_grad ./ m;
101.

```

```
102. Theta1_grad(:,2:end) = Theta1_grad(:,2:end) + Theta1(:,2:end).*lambda./m;  
103. Theta2_grad(:,2:end) = Theta2_grad(:,2:end) + Theta2(:,2:end).*lambda./m;  
104. % -----  
105.  
106. % =====  
107.  
108. % Unroll gradients  
109. grad = [Theta1_grad(:) ; Theta2_grad(:)];  
110.  
111.  
112. end
```