

Week : 5

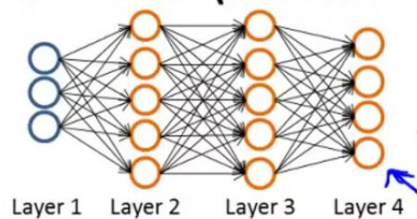
Neural network classification :

There are two types of **neural network**. One is Binary and the other one is Multi-class classification.

Binary Classification : If the number of output classes is less 3 then it is binary classification. In that case the number of output units is just one.

Multi Class Classification : If there number of output classes is more than 2 then it is Multi class classification. In that case the number of output units is equal to output classes. We need to apply one versus all method to get output.

Neural Network (Classification)



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

$$\rightarrow L = \text{total no. of layers in network} \quad \underline{L = 4}$$

$$\rightarrow s_l = \text{no. of units (not counting bias unit) in layer } l \quad s_1 = 3, s_2 = 4, s_4 = s_L = 4$$

Binary classification

$$y = 0 \text{ or } 1 \leftarrow$$

1 output unit \leftarrow

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

$$s_L = 1, \quad \underline{K = 1} \leftarrow$$

$$\rightarrow h_{\Theta}(x)$$

Multi-class classification (K classes)

$$y \in \mathbb{R}^K \quad \text{E.g. } \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \leftarrow$$

pedestrian car motorcycle truck

K output units

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

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

Cost Function :

Cost function needed to be calculated to visualize how much bugg free our algo is.

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:

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

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

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

Diagram illustrating the neural network structure with layers and weights. The diagram shows a sequence of layers with nodes. Weights are represented by circles with subscripts and superscripts, such as $\Theta_{i_0}^{(1)}$, $\Theta_{i_1}^{(2)}$, etc. The output layer is labeled y_k and the input layer is labeled x_0, x_1, \dots . The bias term a_0 is also indicated.

Andrew Ng

In the last part of the cost function I had a confusion why they compute summation of the $\Theta_{11} \Theta_{12} \Theta_{13} \Theta_{14} \dots$

Notice that picture below

3 units

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

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

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

$$\rightarrow h_{\Theta}(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

When we calculate for $a_1^{(2)}$ we need $\Theta_{11} \Theta_{12} \Theta_{13} \Theta_{14}$ all together.

Gradient Computation :

To get the theta which will provide min cost we need to calculate gradient computation.

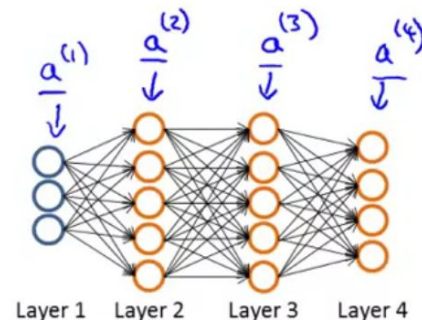
At first we will do forward gradient computation and then backward computation.

Gradient computation

Given one training example (x, y) :

Forward propagation:

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



Gradient computation: Backpropagation algorithm

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

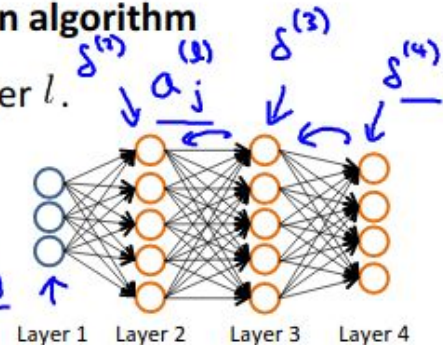
For each output unit (layer $L = 4$)

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

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

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

$$\frac{\partial}{\partial \Theta_{ij}^{(m)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)} \quad (\text{ignoring } \lambda; \text{ if } \lambda = 0) \leftarrow$$



How $g'(z_3) = a_3(1-a_3)$ is explained [here](#).

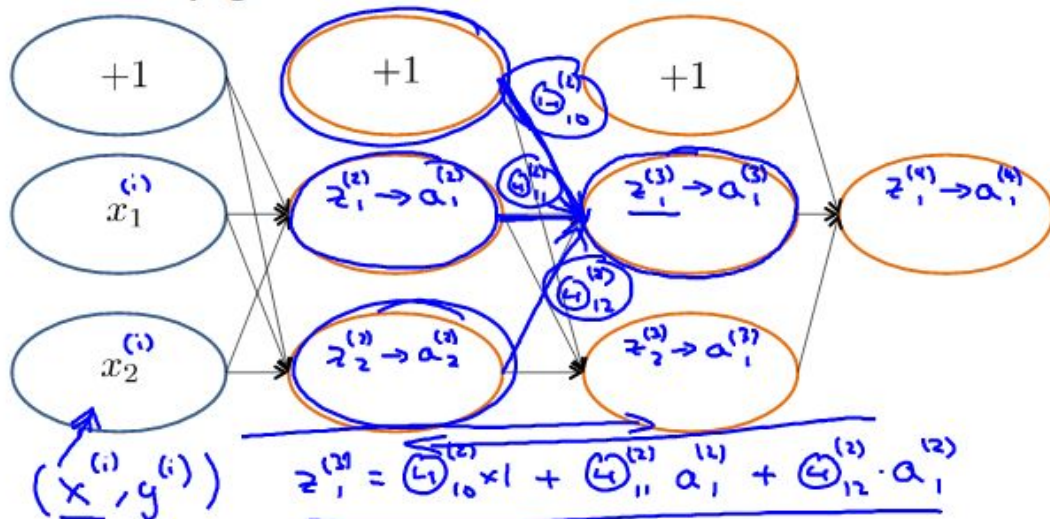
For m number of training sets we can develop a backpropagation algorithm :

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 $m \leftarrow (\underline{x}^{(i)}, \underline{y}^{(i)})$
 - Set $a^{(1)} = \underline{x}^{(i)}$
 - Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \dots, L$
 - Using $y^{(i)}$, compute $\delta^{(L)} = \underline{a}^{(L)} - y^{(i)}$
 - Compute $\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)}$
 - $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)} \leftarrow \delta_i^{(l)}$ $\delta_i^{(l)} := \delta_i^{(l)} + \delta^{(l+1)} (a_j^{(l)})^T$
- $D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)}$ if $j \neq 0$
- $D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)}$ if $j = 0$

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

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 $1 - h(x)$.

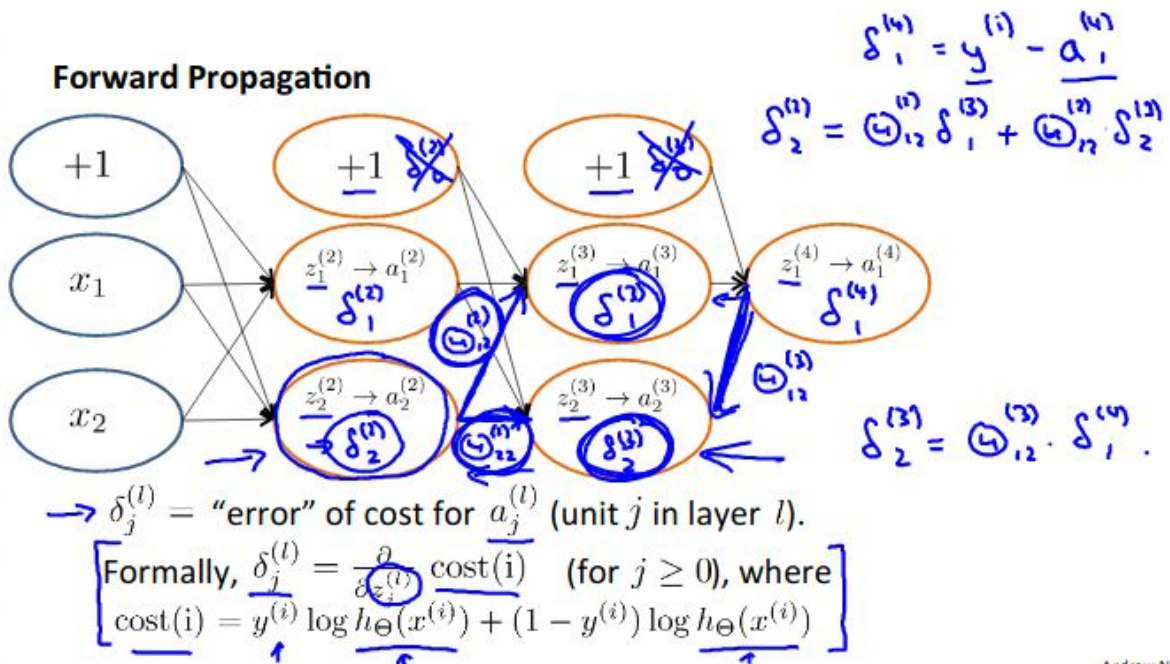
$$\text{cost}(i) = y^{(i)} \log h_{\Theta}(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_{\Theta}(x^{(i)}))$$

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

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

Andrew Ng

If we cut the regularize part of the cost function of the neural network and imagine that we have only one training set, then we can compare it with that simple cost function.



Andrew Ng

Advanced optimization

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

Annotations: $\theta \in \mathbb{R}^{n+1}$, $\text{initialTheta} \in \mathbb{R}^{n+1}$ (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

Andrew Ng

Example

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

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

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

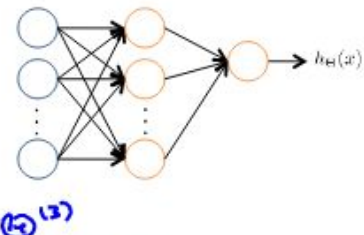
→ thetaVec = [Theta1(:); Theta2(:); Theta3(:)];

→ 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);



Andrew Ng

```
octave:10>
octave:10> theta1 = zeros(10,10);
octave:11> theta2=2*ones(10,20) ;
octave:12> theta3=3*ones(10,1);
octave:13> theta=[theta1(:);theta2(:);theta3(:)];
octave:14> reshape(theta(1:100),10,10);
octave:15> reshape(theta(101:300),10,20);
octave:16> reshape(theta(301:310),10,1);
octave:17>
```

Learning Algorithm

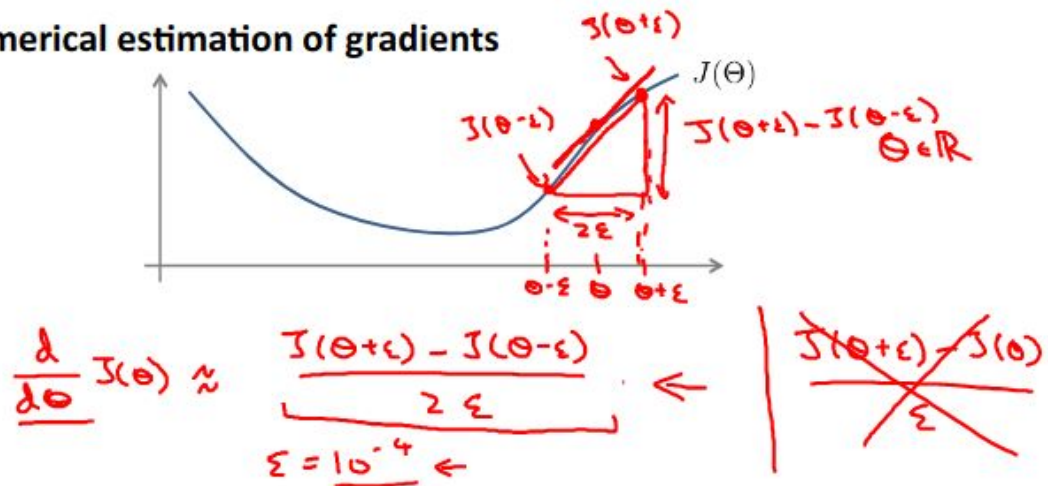
- Have initial parameters $\theta^{(1)}, \theta^{(2)}, \theta^{(3)}$.
- Unroll to get `initialTheta` to pass to
- `fminunc(@costFunction, initialTheta, options)`

`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`.

Andrew Ng

Numerical estimation of gradients



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

Andrew Ng

Parameter vector θ

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

Andrew Ng

```

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;

```

$\left[\begin{matrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_i + \epsilon \\ \vdots \\ \theta_n \end{matrix} \right] \rightarrow \theta_i + \epsilon$
 $\frac{\partial}{\partial \theta_i} J(\theta)$

Check that $\text{gradApprox} \approx \text{DVec} \leftarrow$
 \uparrow
 From backprop.

Andrew Ng

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.

Andrew Ng

Initial value of Θ

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

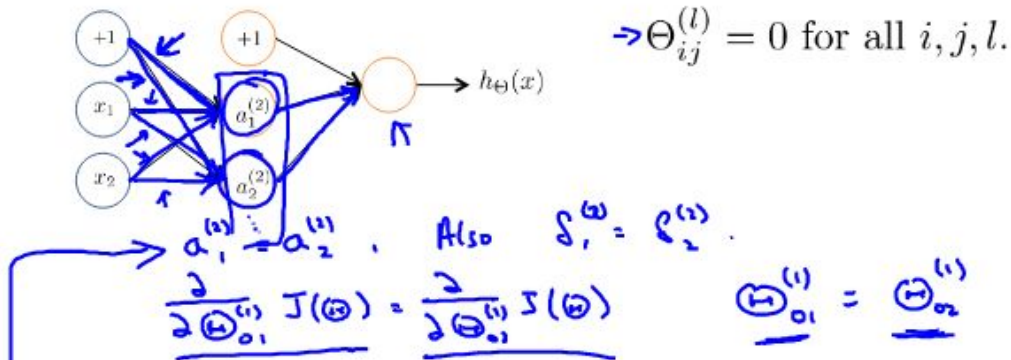
```
optTheta = fminunc(@costFunction,  
    initialTheta, options)
```

Consider gradient descent

Set initialTheta = zeros(n,1) ?

Andrew Ng

Zero initialization



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

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

Andrew Ng

Random initialization: Symmetry breaking

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

E.g.

\rightarrow Theta1 = $\text{rand}(10, 11) * (2 * \text{INIT_EPSILON}) - \text{INIT_EPSILON};$ $[-\epsilon, \epsilon]$

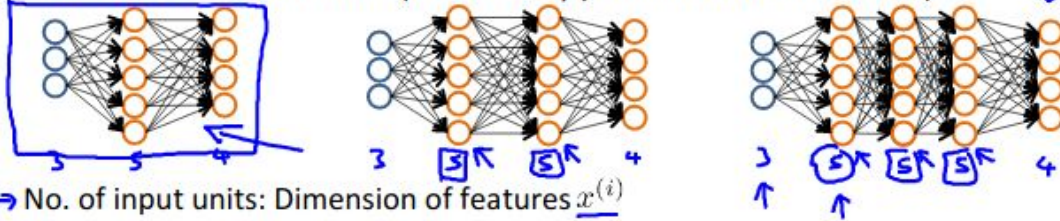
Random 10x11 matrix (betw. 0 and 1)

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

Andrew Ng

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)



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

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

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

Andrew Ng

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 $i = 1:m$ { $(x^{(1)}, y^{(1)})$ $(x^{(2)}, y^{(2)})$, ..., $(x^{(m)}, y^{(m)})$ }

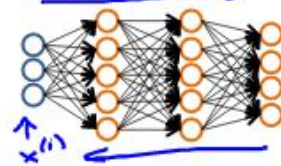
→ Perform forward propagation and backpropagation using example $(x^{(i)}, y^{(i)})$

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

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

...

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



Andrew Ng

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.