Neural Network

cost function

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

L = total no. of layers in network

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

→ l 層上的神經元有幾個。

$J(\theta)$

把輸出層加總在一起

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

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_l+1}\left(\theta_{j,i}^{(l)}\right)^2$$

regularization 時不要加 θ_{0i}，其為 bias 的 weight

(cost function for classification)

Backpropagation Algorithm

$J(\theta)$

把輸出層加總在一起

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

$$+\frac{\lambda}{2m}\sum_{i=1}^{L-1}\sum_{i=1}^{S_i}\sum_{j=1}^{S_i+1}\left(\theta_{j,i}^{(l)}\right)^2$$

Need code to compute:

$\min_\theta J(\theta) \longrightarrow \frac{\partial}{\partial \theta_{i,j}^{(l)}} J(\theta)$

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

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

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

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

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

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

we have training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

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

for $i = 1 : m$

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

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

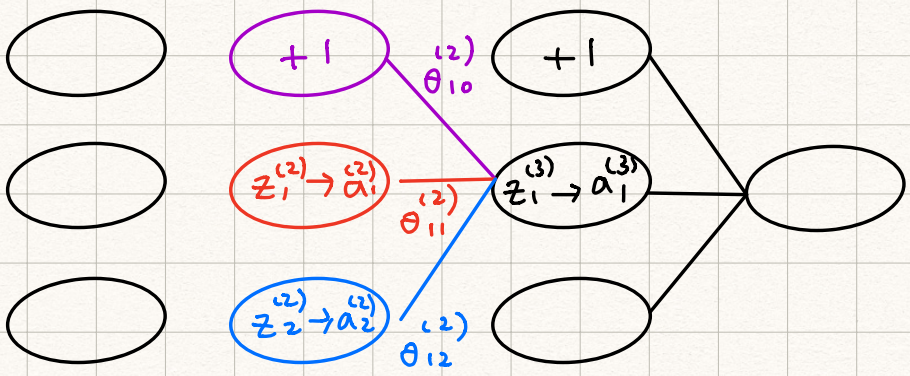
$$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \quad \text{if } j \neq 0$$

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

$$D_{ij}^{(2)} = \frac{\partial}{\partial \Theta_{ij}^{(2)}} J(\Theta) \quad \ast$$

Backpropagation intuition

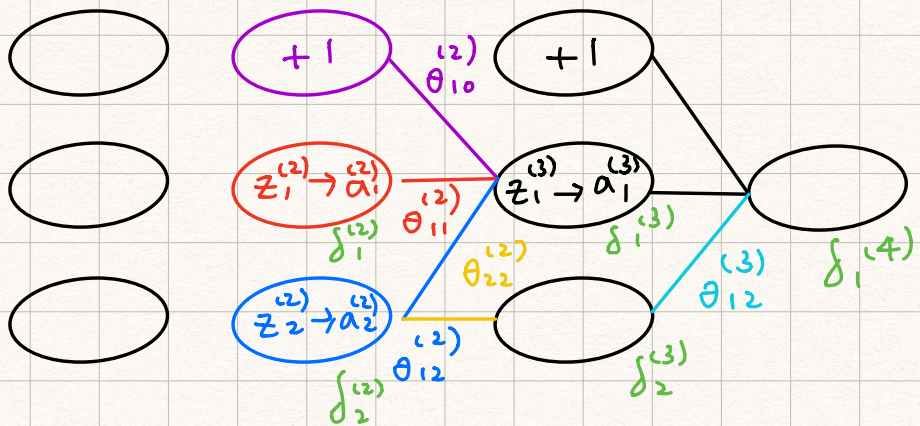
Forward Propagation



$$z_1^{(3)} = \theta_{10}^{(2)} \times 1 + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} \cdot a_1^{(2)}$$

$$z \xrightarrow{\text{sigmoid}} a$$

Backpagation Propagation



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

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

$$\delta_j^{(l)} = \text{"error" of cost for } a_j^{(l)}$$

$$= \frac{\partial}{\partial z_j^{(l)}} cost(i) \quad , \text{ for } j > 0$$

Backprogation in Practice

Advanced optimization

function [jVal, gradient]

代價函數 偏導數值 皆需向量型式

= costFunction (theta)

optTheta = fminunc (@costFunction, initialTheta, options)

matrix size :

$\theta^{(1)} = [\ ]_{10\times 11}$ $\theta^{(2)} = [\ ]_{10\times 11}$

$\theta^{(3)} = [\ ]_{1\times 11}$

- matrix to vectors ?

thetaVec

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

DVec = [D1(:); D2(:); D3(:)];

- vectors to matrix ?

Theta1 = reshape(thetaVec(1:110), 10, 11)

Theta2 = reshape(thetaVec(111:220), 10, 11)

Theta3 = reshape(thetaVec(221:231), 1, 11)

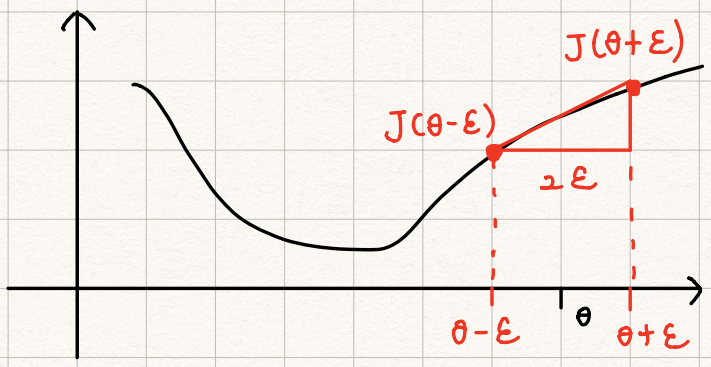
Summary

$\theta^{(1)}$, $\theta^{(2)}$, $\theta^{(3)}$ 合成一向量 initialTheta 後，
丟入 fminunc

從 costFunction(thetaVec) 的 thetaVec 取得
$\theta^{(1)}$, $\theta^{(2)}$, $\theta^{(3)}$ 後，用 fp, bp 計算
$D^{(1)}$, $D^{(2)}$, $D^{(3)}$ and $J(\theta)$

最後，將 $D^{(1)}$, $D^{(2)}$, $D^{(3)}$ 合成一向量 gradientVec

Gradient Checking 梯度檢查



$$\frac{d}{d\theta}J(\theta) = \frac{J(\theta+\varepsilon) - J(\theta-\varepsilon)}{2\varepsilon} \qquad \varepsilon = 10^{-4}$$

Parameter vector θ

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

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

$$\vdots$$

use for loop get gradApprox

```
for i = 1:n
    thetaPlus = theta
    thetaPlus(i) = thetaPlus(i) + ε
    thetaMinus = .....
    thetaMinus(i) = ..... - ε
    gradApprox(i) = (J(thetaPlus) - J(thetaMinus))
end
```

check that gradApprox $\approx$ DVec

↑

From bp

Summary

1. 以 bp 計算 DVec
2. 應用梯度檢驗獲得 gradApprox
3. 確保 gradApprox ≈ DVec
4. 關掉梯度檢驗，以 bp 訓練資料

Random initialization

Initial Theta setting

initialTheta = zeros(n, 1) ?

or initialTheta 裡的值都相等

在 logistic regression 可行，但是

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

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

不論如何計算，所有 θ 得到的更新皆一樣

Symmetry breaking

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

Theta 1
= rand(10, 11) * (2 * INIT_EPSILON)
- INIT_EPSILON

rand(10, 11) 產生 $\{\ \}_{10\times 11}$, 0~1

Put it together

Picking a network architecture

- input: Dimension of feature
- output: Number of classes

- Reasonable default: 1 hidden layyer, or if >1 hidden layer, have same no. of hidden units in every layer
  隱藏層的神經元越多越好

Training a neural network

1. 隨機初始化權重
2. 以 fp 得到 $h_\Theta(x^{(i)})$ for any $x^{(i)}$
3. 計算 cost function $J(\Theta)$
4. 以 bp 計算 $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$
5. 梯度檢驗以確認梯度確實有下降，程式無 bug 後，關掉梯度檢驗
6. 以梯度下降法 or 其他算法來最小化 cost function

   試圖找出某個 Θ 使得 $J(\Theta)$ 為局部最小 or 全域最小

```
for i = 1:m
    Perform fp and bp using example
    (x(i), y(i))
    (Get activation a(l) and delta terms
     d(l) for l = 2 ..... L)
```