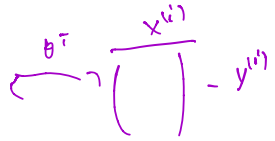
I. Supervised Learning and Optimization

① Linear Regression

a hypothesis class : $h_\theta(x)$ $x^{(i)} \longrightarrow y^{(i)}$



Linear function : $h_\theta(x) = \theta^T x$

cost function: $J(\theta) = \frac{1}{2} \sum_i (h_\theta(x^{(i)}) - y^{(i)})^2$ (also called loss / penalty / objective function)

problem formulation

$= \frac{1}{2} \sum_i (\theta^T x^{(i)} - y^{(i)})^2$ (可用 loop 的方法)

$= \frac{1}{2} (\theta^T \cdot X - Y)^T \cdot (\theta^T X - Y)$ (也可用向量的形式)

Function Minimization:

$\nabla_\theta J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \frac{\partial J(\theta)}{\partial \theta_2} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{bmatrix}$ 其中 $\frac{\partial J(\theta)}{\partial \theta_j} = \sum_i (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$

$g(j) = \underbrace{(\theta^T \cdot X - Y)}_{1 \times 400} \cdot \underbrace{\begin{bmatrix} x_j^{(1)} \\ x_j^{(2)} \\ \vdots \\ x_j^{(400)} \end{bmatrix}}_{400 \times 14}$ $X(j,:)'$ $j = 1, \cdots n$ ($n = 14$)

Exercise 1A: Linear Regression

house_data: $\begin{bmatrix} 1 & 1 & \cdots & 1 & 1 \\ 0.0469 & 6.4441 & \cdots & 1.0025 & 1.1517 \\ \vdots & \vdots & & \vdots & \vdots \\ 22.2000 & 16.1400 & \cdots & 17 & 13.1000 \end{bmatrix}_{15 \times 506}$

train $\begin{cases} \text{train.X}_{14 \times 400} \\ \text{train.y}_{1 \times 400} \end{cases}$ test $\begin{cases} \text{test.X}_{14 \times 106} \\ \text{test.y}_{1 \times 106} \end{cases}$

一列就是一个 example，每一行代表一个 feature

n: 14 { train.X } m: 400

size(data, 1) ⇒ 返回 data 矩阵有的行数

size(data, 2) ⇒ 返回 data 矩阵有的列数

$\theta' \cdot \text{test.X} \Longleftrightarrow \text{test.y}$

$1 \times 14 \quad 14 \times 106 \quad 1 \times 106$

$\theta_{1 \times 14}$ |)

② Logistic Regression

a hypothesis class $h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}} = \sigma(\theta^T x)$ → sigmoid function

$p(y=1|x) = h_\theta(x)$

$p(y=0|x) = 1 - h_\theta(x)$

we label example → 1, if $h_\theta(x) > 0.5$

→ 0, otherwise

loss function:

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

若用梯度下降法(to 减小 $J(\theta)$)

$$\frac{\partial J(\theta)}{\partial \theta_j} = \sum_i (h_\theta(x^{(i)}) - y^{(i)})\, x_j^{(i)} \longrightarrow \text{向量化算法}$$

$$\nabla_\theta J = \sum_i (h_\theta(x^{(i)}) - y^{(i)})\, x^{(i)}$$

5 linear regression 中梯度下降法算法一致，只不过这里的 $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$

$$[y^{(1)}\ y^{(2)}\ \cdots\ y^{(m)}] \cdot \log\begin{bmatrix} h_\theta(x^{(1)}) \\ h_\theta(x^{(2)}) \\ \vdots \\ h_\theta(x^{(m)}) \end{bmatrix}$$

$[\text{theta}]_{n\times 1}$ $[X]_{n\times m}$

for: $y * \log(h(x' \cdot \text{theta})) + (1-y) * \log(1 - h(x' \cdot \text{theta}))$

$$g\begin{bmatrix} g(1) \\ \vdots \\ g(n) \end{bmatrix} = (h(\text{theta}' \cdot X) - y)$$

$1\times m$

temp = theta' * X

用向量化的方法就快很多.

Exercise 1B

Mnist：0、1 数字

M: X = loadMNISTImages('...\train-images-idx3-ubyte') → [ ] 784×60000 (特征标准化 0均值，1方差 值都在 0~1 之间)

y = loadMNISTLabels('...\train-labels-idx1-ubyte') → [...] 1×60,000 0-9 之间的标签

60,000 个中是 0 和 1 的共有 12665 个 (5923 个 0、6742 个 1) / 10000 个中有 0 和 1 共有 2115 个 (980 个 0, 1135 个 1)

train.X 784×12665　　test.X 784×2115　　(Matlab 中也有 ( ).( ) 这种点操作，有点像 C 中的结构体)

train.y 1×12665　　test.y 1×2115

增加一行 1 之后

train.X 785×12665　　test.X 785×2115

n: 785 #features　　m: 12665 #examples

普通的方法梯度：(自己特意写的最慢的方法)

$(785 \times 12665 + 12665 + 785 \times 12665 \times 785 + 785 \times 12665) \times 100$ 7小时→20次 1天多

$= 7.8 \times 10^{11} = 780$亿次 (过于复杂)

③ Debugging: Gradient Checking

$$g_i(\theta) = \frac{\partial}{\partial \theta_i} J(\theta) \quad \text{where} \quad \vec{e}_i = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \leftarrow i\text{-th}$$

$$\theta^{(i+)} = \theta + \varepsilon \times \vec{e}_i$$

$$\theta^{(i-)} = \theta - \varepsilon \times \vec{e}_i$$

$$g_i(\theta) = \frac{J(\theta^{(i+)}) - J(\theta^{(i-)})}{2\varepsilon}$$

④ softmax regression (改代码写的还不是很正确)

是logistic regression的更一般化形式，针对多重分类

$$h_\theta(x) = \begin{bmatrix} p(y=1|x;\theta) \\ p(y=2|x;\theta) \\ \vdots \\ p(y=k|x;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{\theta^{(j)T}x}} \begin{bmatrix} e^{\theta^{(1)T}x} \\ e^{\theta^{(2)T}x} \\ \vdots \\ e^{\theta^{(k)T}x} \end{bmatrix}$$

参数: $\theta = \begin{bmatrix} | & | & | & & | \\ \theta^{(1)} & \theta^{(2)} & \theta^{(3)} & \cdots & \theta^{(k)} \\ | & | & | & & | \end{bmatrix}_{n \times k}$

cost function:

在之前的 Logistic regression 中，我们的损失函数是

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

X_0: 7085×1 785×9

d: 7850×1

X: 785×60000

也可以写作

$$= -\left[\sum_{i=1}^{m}\sum_{k=0}^{1} 1\{y^{(i)}=k\}\log P(y^{(i)}=k \mid x^{(i)};\theta)\right]$$

y: 1×60000

debug时发现的维度匹配

now we have K classes

so

$$J(\theta) = -\left[\sum_{i=1}^{m}\sum_{k=1}^{K} 1\{y^{(i)}=k\}\log P(y^{(i)}=k \mid x^{(i)};\theta)\right]$$

$$= -\left[\sum_{i=1}^{m}\sum_{k=1}^{K} 1\{y^{(i)}=k\}\log \frac{e^{\theta^{(k)T}x^{(i)}}}{\sum_{j=1}^{K} e^{\theta^{(j)T}x^{(i)}}}\right]$$

$$1\{y^{(i)}=1\}\log\frac{e^{\theta^{(1)T}x^{(i)}}}{\sum e^{\theta^{(j)T}x^{(i)}}} + 1\{y^{(2)}=2\}\log\frac{e^{\theta^{(2)T}x^{(i)}}}{\sum e^{\theta^{(j)T}x^{(i)}}}$$

$$\begin{bmatrix} 1\{y^{(1)}=1\} \\ 1\{y^{(1)}=2\} \\ \vdots \\ 1\{y^{(1)}=k\} \end{bmatrix}^T \begin{bmatrix} \log\frac{e^{\theta^{(1)T}x^{(1)}}}{\sum e^{\theta^{(j)T}x^{(1)}}} \\ \vdots \\ \log\frac{e^{\theta^{(k)T}x^{(1)}}}{\sum e^{\theta^{(j)T}x^{(1)}}} \end{bmatrix} + \begin{bmatrix} 1\{y^{(2)}=1\} \\ 1\{y^{(2)}=2\} \\ \vdots \\ 1\{y^{(2)}=k\} \end{bmatrix}^T \begin{bmatrix} \log\frac{e^{\theta^{(1)T}x^{(2)}}}{\sum e^{\theta^{(j)T}x^{(2)}}} \\ \vdots \\ \log\frac{e^{\theta^{(k)T}x^{(2)}}}{\sum e^{\theta^{(j)T}x^{(2)}}} \end{bmatrix}$$

$A = e^{(\theta^T x)}$ [matrix] — sum ↓

B = [0 0 0 0]

bsxfun

y $\quad \theta^T X_{9\times 60000}$

I = sub2ind(size(θ^TX), y, 1: size(θ^TX,2))

- sum(θ^TX(I)) = J

$$\theta = \begin{bmatrix} | & | & & | \\ \theta^{(1)} & \theta^{(2)} & \cdots & \theta^{(k-1)} \\ | & | & & | \end{bmatrix}_{785\times 9}$$

X: $\begin{bmatrix} \end{bmatrix}_{785\times 60000}$

y: $\begin{bmatrix} \end{bmatrix}_{1\times 60000}$

梯度:

$$\nabla_{\theta^{(k)}} J(\theta) = -\sum_{i=1}^{m}\left[x^{(i)}\left(1\{y^{(i)}=k\} - P(y^{(i)}=k \mid x^{(i)};\theta)\right)\right]$$

→ 而这就是一个向量

所以 $\frac{\partial J}{\partial\theta} = \begin{bmatrix} \frac{\partial J}{\partial\theta_0} \\ \frac{\partial J}{\partial\theta_1} \\ \vdots \\ \frac{\partial J}{\partial\theta_n} \end{bmatrix}$ 全部是一个向量

(应该也可以从之前的J(θ)的表达式推过来的)
暂且先不推导了。

向量化写法：

$$g = \begin{bmatrix} | & | & | & & | \\ g^{(1)} & g^{(2)} & g^{(3)} & \cdots & g^{(k)} \\ | & | & | & & | \end{bmatrix}$$

$$g(1) = \nabla_\theta J(\theta) = -\sum_{i=1}^{m} \left[x^{(i)} \left(1\{y^{(i)}=1\} - p(y^{(i)}=1 \mid x^{(i)};\theta) \right) \right]$$

$$= \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} \begin{bmatrix} 1\{y^{(1)}=1\} - p(y^{(1)}=1 \mid x^{(1)};\theta) \\ 1\{y^{(2)}=1\} - p(y^{(2)}=1 \mid x^{(2)};\theta) \\ \vdots \\ 1\{y^{(m)}=1\} - p(y^{(m)}=1 \mid x^{(m)};\theta) \end{bmatrix}$$

$$= \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} \begin{bmatrix} 1\{y^{(1)}=1\} - \frac{e^{\theta^{(1)T}x^{(1)}}}{\sum_{j=1}^{k} e^{\theta^{(j)T}x^{(1)}}} \\ 1\{y^{(2)}=1\} - \frac{e^{\theta^{(1)T}x^{(2)}}}{\sum_{j=1}^{k} e^{\theta^{(j)T}x^{(2)}}} \\ \vdots \end{bmatrix}$$

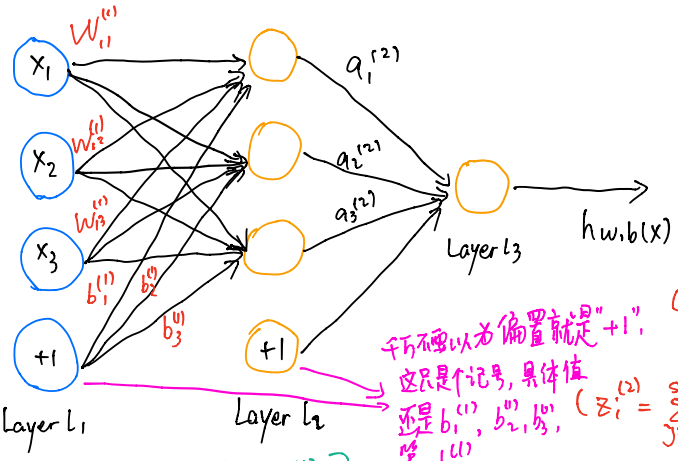
$$g(2) = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} \cdots \begin{bmatrix} 1\{y^{(1)}=2\} - \frac{e^{\theta^{(2)T}x^{(1)}}}{\sum_{j=1}^{k} e^{\theta^{(j)T}x^{(1)}}} \\ 1\{y^{(2)}=2\} - \frac{e^{\theta^{(2)T}x^{(2)}}}{\sum_{j=1}^{k} e^{\theta^{(j)T}x^{(2)}}} \\ \vdots \end{bmatrix}$$

$$\begin{bmatrix} | \\ x_0^{(1)} \\ | \end{bmatrix} k_1 + \begin{bmatrix} | \\ x^{(2)} \\ | \end{bmatrix} k_2 + \begin{bmatrix} | \\ x^{(3)} \\ | \end{bmatrix} k_3 + \cdots + \begin{bmatrix} | \\ x^{(m)} \\ | \end{bmatrix} k_m$$

$$= \begin{bmatrix} | & | & | & & | \\ x^{(1)} & x^{(2)} & x^{(3)} & \cdots & x^{(m)} \\ | & | & | & & | \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \\ k_3 \\ k_4 \end{bmatrix} =$$

I. Supervised Neural Networks

⑤ multi-layer neural networks



千万不要以为偏置就是"+1"，这只是个记号，具体值还是 $b_1^{(1)}, b_2^{(1)}, b_3^{(1)}$ 等 $b_i^{(l)}$

$$W^{(1)} = \begin{bmatrix} W_{11}^{(1)} & W_{12}^{(1)} & W_{13}^{(1)} \\ \cdots & \cdots & \cdots \end{bmatrix}, \quad b^{3\times 3}$$

$$a_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)})$$

记号（矩阵对应图和神经网络）

参数 $(W, b) = (W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)})$

n_l — 层数

s_l — l 层节点数（不包含偏置）

$W_{ij}^{(l)}$ — l 层的 j 元与 $l+1$ 层的 i 元间的权重

$b_i^{(l)}$ — 和 $l+1$ 层的 i 元连接的偏置

$a_i^{(l)}$ — l 层 i 元的激活输出（特别的：$a_i^{(1)} = x_i$）

$z_i^{(l)}$ — l 层 i 元的加权和，包含偏置

$(z_i^{(2)} = \sum_{j=1}^{s_1} W_{ij}^{(1)} x_j + b_i^{(1)}$，∴ 又 $a_i^{(l)} = f(z_i^{(l)}))$

$$\begin{bmatrix} W^{(1)}_{21} & W^{(1)}_{22} & W^{(1)}_{23} \\ W^{(1)}_{31} & W^{(1)}_{32} & W^{(1)}_{33} \end{bmatrix} \in \mathbb{R}^{2\times 3}$$

$$b^{(1)} = \begin{bmatrix} b^{(1)}_1 & b^{(1)}_2 & b^{(1)}_3 \end{bmatrix} \in \mathbb{R}^{1\times 3}$$

$$a^{(2)}_2 = f(W^{(1)}_{21}x_1 + W^{(1)}_{22}x_2 + W^{(1)}_{23}x_3 + b^{(1)}_2)$$

$$a^{(2)}_3 = f(W^{(1)}_{31}x_1 + W^{(1)}_{32}x_2 + W^{(1)}_{33}x_3 + b^{(1)}_3)$$

$$a^{(3)}_1 = h_{W,b}(x) = f(W^{(2)}_{11}a^{(2)}_1 + W^{(2)}_{12}a^{(2)}_2 + W^{(2)}_{13}a^{(2)}_3 + b^{(2)}_1)$$

element-wise：向量之间对应元素的操作

向量化的写法：

$$z^{(2)} = W^{(1)}x + b^{(1)}$$
$$a^{(2)} = f(z^{(2)})$$
$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$
$$h_{W,b}(x) = a^{(3)} = f(z^{(3)})$$

总结来说

$$z^{(l+1)} = W^{(l)}a^{(l)} + b^{(l)}$$
$$a^{(l+1)} = f(z^{(l+1)})$$

→ step forward propagation

feedforward neural network
前馈神经网络
(没有 loop 或 cycle)

→ element-wise product
也叫作 hadamard product

$J(W,b;x,y) = \frac{1}{2}\|h_{W,b}(x) - y\|^2$ 对单个样本 (x,y) 来说，J — cost function

W,b 代表是 W,b 的函数　　x,y 代表给定 (x,y) 样本时

given m examples:

$$J(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)})\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W^{(l)}_{ji}\right)^2$$

参数要随机初始化，不能全设为0

Symmetry breaking.

$$= \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\|h_{W,b}(x^{(i)}) - y^{(i)}\|^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W^{(l)}_{ji}\right)^2$$

pseudo-code：伪代码

average sum-of-squares error

regularization term (weight decay term)　通常不对 $b^{(l)}$ 正则化，相差不大

梯度下降迭代：

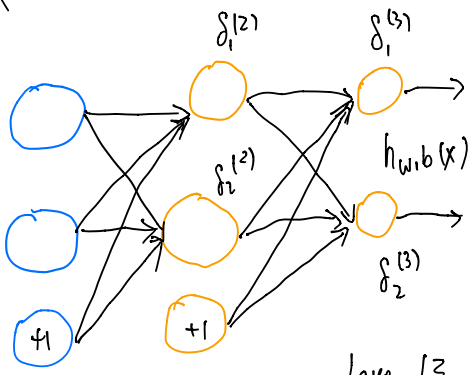
$$W^{(l)}_{ij} = W^{(l)}_{ij} - \alpha\frac{\partial}{\partial W^{(l)}_{ij}}J(W,b)$$

$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W,b)$$

$$z^{(l+1)} = W^{(l)} a^{(l)} + b^{(l)}$$

$$a^{(l+1)} = f(z^{(l+1)})$$

$$\delta_i^{(n_l)}$$



$$a_1^{(2)} = f(W_{11}^{(1)} x_1 + W_{12}^{(1)} x_2 + b_1^{(1)})$$

$$a_2^{(2)} = f(W_{21}^{(1)} x_1 + W_{22}^{(1)} x_2 + b_2^{(1)})$$

$$a_1^{(3)} = h_{W,b}(x) = f(W_{11}^{(2)} a_1^{(2)} + W_{12}^{(2)} a_2^{(2)} + b_1^{(2)})$$

$$a_2^{(3)} = h_{W,b}(x) = f(W_{21}^{(2)} a_1^{(2)} + W_{22}^{(2)} a_2^{(2)} + b_2^{(2)})$$

誤差 $J = \frac{1}{2}\left[\left(y_1 - f(z_1^{(3)})\right)^2 + \left(y_2 - f(z_2^{(3)})\right)^2\right]$

$$z_1^{(3)} = W_{11}^{(2)} a_1^{(2)} + W_{12}^{(2)} a_2^{(2)} + b_1^{(2)}$$
$$= W_{11}^{(2)} f(z_1^{(2)}) + W_{12}^{(2)} f(z_2^{(2)}) + b_1^{(2)}$$

$$z_2^{(2)} = W_{21}^{(2)} a_1^{(2)} + W_{22}^{(2)} a_2^{(2)} + b_2^{(2)}$$
$$= W_{21}^{(2)} f(z_1^{(2)}) + W_{22}^{(2)} f(z_2^{(2)}) + b_2^{(2)}$$

$$\delta_1^{(3)} = \frac{\partial J}{\partial z_1^{(3)}} = \frac{1}{2} \cdot 2(y_1 - f(z_1^{(3)})) \cdot -f'(z_1^{(3)})$$
$$= -(y_1 - a_1^{(3)}) f'(z_1^{(3)})$$

$$\delta_2^{(3)} = -(y_2 - a_2^{(3)}) f'(z_2^{(3)})$$

set: $\longrightarrow$ $\delta_1^{(2)} = \left(W_{11}^{(2)} \delta_1^{(3)} + W_{21}^{(2)} \delta_2^{(3)}\right) f'(z_1^{(2)})$

$$\delta_1^{(2)} = \frac{\partial J}{\partial z_1^{(2)}} = \frac{\partial J}{\partial z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial z_1^{(2)}} = \delta_1^{(3)} \cdot \left(W_{11}^{(2)} f'(z_1^{(2)})\right) = \delta_2^{(3)} \cdot \left(W_{21}^{(2)} f'(z_1^{(2)})\right)$$

$$\frac{\partial J(W,b;x,y)}{\partial W_{11}^{(2)}} = a_1^{(2)} \delta_1^{(3)} = a_1^{(2)} \left(-(y_1 - a_1^{(3)}) f'(z_1^{(3)})\right) \checkmark$$

$$\frac{\partial J(W,b;x,y)}{\partial W_{11}^{(1)}} = a_1^{(1)} \delta_1^{(2)} = a_1^{(1)} \left(W_{11}^{(2)} \delta_1^{(3)} + W_{21}^{(2)} \delta_2^{(3)}\right) f'(z_1^{(2)}) \checkmark$$

$$\frac{\partial J(W,b;x,y)}{\partial b_1^{(2)}} = \delta_1^{(3)} \checkmark$$

$$\frac{\partial J(W,b;x,y)}{\partial b_2^{(2)}} = \delta_2^{(3)} \checkmark$$

公式①：for $l = n_l-1, n_l-2, \cdots 1$ 有 $\delta_i^{(l)} = \left(\sum_{j=1}^{S_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)}\right) f'(z_i^{(l)})$

公式②：$\frac{\partial J(W,b;x,y)}{\partial W_{ij}^{(l)}} = a_j^{(l)} \delta_i^{(l+1)}$

$$\frac{\partial J(W,b;x,y)}{\partial b_i^{(l)}} = \delta_i^{(l+1)}$$

证明公式①：$\delta_i^{(l)} = \frac{\partial J}{\partial z_i^{(l)}} = \sum_{j=1}^{S_{l+1}} \frac{\partial J}{\partial z_j^{(l+1)}} \frac{\partial z_j^{(l+1)}}{\partial z_i^{(l)}}$

$= \sum_{j=1}^{S_{l+1}} \delta_j^{(l+1)} W_{ji}^{(l)} f'(z_i^{(l)})$ 都有的一项，提出来

$= \left(\sum_{j=1}^{S_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)}\right) f'(z_i^{(l)})$

证明公式②：$\frac{\partial J}{\partial W_{ij}^{(l)}} = \frac{\partial J}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial W_{ij}^{(l)}} = \delta_i^{(l+1)} \cdot a_j^{(l)}$

$\frac{\partial J}{\partial b_i^{(l)}} = \frac{\partial J}{\partial z_i^{(l+1)}} \frac{\partial z_i^{(l+1)}}{\partial b_i^{(l)}} = \delta_i^{(l+1)}$ 这一项是1

下面是向量化写法.

向量写法：

$W^{(l)}$: $S_{l+1} \times S_l$

$\delta^{(l+1)}$: $S_{l+1} \times m$

$\delta^{(l)}$: $S_l \times m$

$z^{(l)}$: $S_l \times m$

两个维度相同

相乘时注意维度匹配就行

$\delta^{(n_l)}$ → 本例中：Stack(cell)

$W^{(1)}_{256\times784}$ $W^{(2)}_{10\times256}$

$b^{(1)}_{256\times1}$ $b^{(2)}_{10\times1}$

$[256\times784]\cdot[784\times60000]$

$=[256\times60000]$

$[10\times256]\cdot[256\times60000]$

$=[10\times60000]$

m为example的数量.

Backpropagation Algorithm

非向量化

1. 前向传播，算出 L_2 到 L_{nl} 层的 a_i, z_i
2. 对第 n_l 层（输出层），设 $\delta_i^{(nl)} = -(y_i - a_i^{(nl)}) \cdot f'(z_i^{(nl)})$
3. 对第 $n_l-1, n_l-2, \dots, 2$ 层，设
$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)}\right) f'(z_i^{(l)})$$
4. $$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y) = a_j^{(l)} \delta_i^{(l+1)}$$
$$\frac{\partial}{\partial b_i^{(l)}} J(W,b;x,y) = \delta_i^{(l+1)}$$

} 用 BP 算梯度.

向量化（用 "$\bullet$" 表示 hadamard product）

1. 前向传播，算出 L_2 到 L_{nl} 层的 a_i, z_i
2. 对第 n_l 层（输出层），设 $\delta^{(nl)} = -(y - a^{(nl)}) \bullet f'(z^{(nl)})$
3. 对第 $n_l-1, n_l-2, \dots, 2$ 层，设
$$\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)}\right) \bullet f'(z^{(l)})$$
4. $$\nabla_{W^{(l)}} J(W,b;x,y) = \delta^{(l+1)} (a^{(l)})^T$$
$$\nabla_{b^{(l)}} J(W,b;x,y) = \delta^{(l+1)}$$

Batch Gradient Descent

1. set $\Delta W^{(l)} := 0$, $\Delta b^{(l)} := 0$ for all layer l
2. For $i = 1$ to m {
   1. use backpropagation algorithm to compute $\nabla_{W^{(l)}} J(W,b;x,y)$ and $\nabla_{b^{(l)}} J(W,b;x,y)$
   2. set $\Delta W^{(l)} := \Delta W^{(l)} + \nabla_{W^{(l)}} J(W,b;x,y)$
   $\Delta b^{(l)} := \Delta b^{(l)} + \nabla_{b^{(l)}} J(W,b;x,y)$
3. update the parameters:
$$W^{(l)} = W^{(l)} - \alpha\left[\left(\frac{1}{m}\Delta W^{(l)}\right) + \lambda W^{(l)}\right]$$
$$b^{(l)} = b^{(l)} - \alpha\left[\frac{1}{m}\Delta b^{(l)}\right]$$

Exercise: Supervised Neural Network

ei:
- input_dim = 784
- output_dim = 10
- layer_sizes = [256, 10]
- lambda = 0

这是一个 Struct

而 cell 里面的一个单元也可以为一个 struct，可以用 { } 来索引

Stack(cell)

activation_fun = 'logistic'

$W^{(1)}_{256\times784}$ $W^{(2)}_{10\times256}$ $b^{(1)}_{256\times1}$ $b^{(2)}_{10\times1}$

$$J(\theta) = -\left[\sum_{i=1}^{M}\sum_{k=1}^{K} 1\{y^{(i)}=k\}\log\frac{e^{\theta^{(k)T}h_{w,b}(x^{(i)})}}{\sum_{j=1}^{K} e^{\theta^{(j)T}h_{w,b}(x^{(i)})}}\right]$$

$[256\times784]\cdot[784\times60000]$
$=[256\times60000]$
$[10\times256]\cdot[256\times60000]$
$=[10\times60000]$

$$\delta^{(nl)} = -\sum_{i=1}^{M}\left[\left(1\{y^{(i)}=k\} - P(y^{(i)}=k|x^{(i)};\theta)\right)\right]$$

Ⅲ. Supervised Convolutional Neural Network

⑥ Feature Extraction Using Convolution & pooling

Matlab中二维卷积的定义：Convolve:

$$C(j,k) = \sum_{p}\sum_{q} A(p,q)\,B(j-p+1,\,k-q+1)$$

P和q会遍历所有可能到A(p,q)和B(j-p+1, k-q+1)的合法下标的值

$$\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix} \quad \begin{bmatrix}5 & 6 & 7\\ 8 & 9 & 10\end{bmatrix} \xrightarrow{conv2} \begin{bmatrix}5 & 16 & 19 & 14\\ 23 & 63 & 73 & 48\\ 24 & 59 & 66 & 40\end{bmatrix}$$

A　　B　　A翻转 $\begin{bmatrix}4 & 3\\ 2 & 1\end{bmatrix}$　　C

e.g.

$C(1,1) = \sum_{p}\sum_{q} A(p,q)\,B(2-p,2-q) = A(1,1)B(1,1) + A(1,2)B(1,0) + A(2,1)B(0,1) + A(2,2)B(0,0) = 5$

$C(2,2) = \sum_{p}\sum_{q} A(p,q)\,B(3-p,3-q) = A(1,1)B(2,2) + A(1,2)B(2,1) + A(2,1)B(1,2) + A(2,2)B(1,1) = 9+16+18+20 = 63$

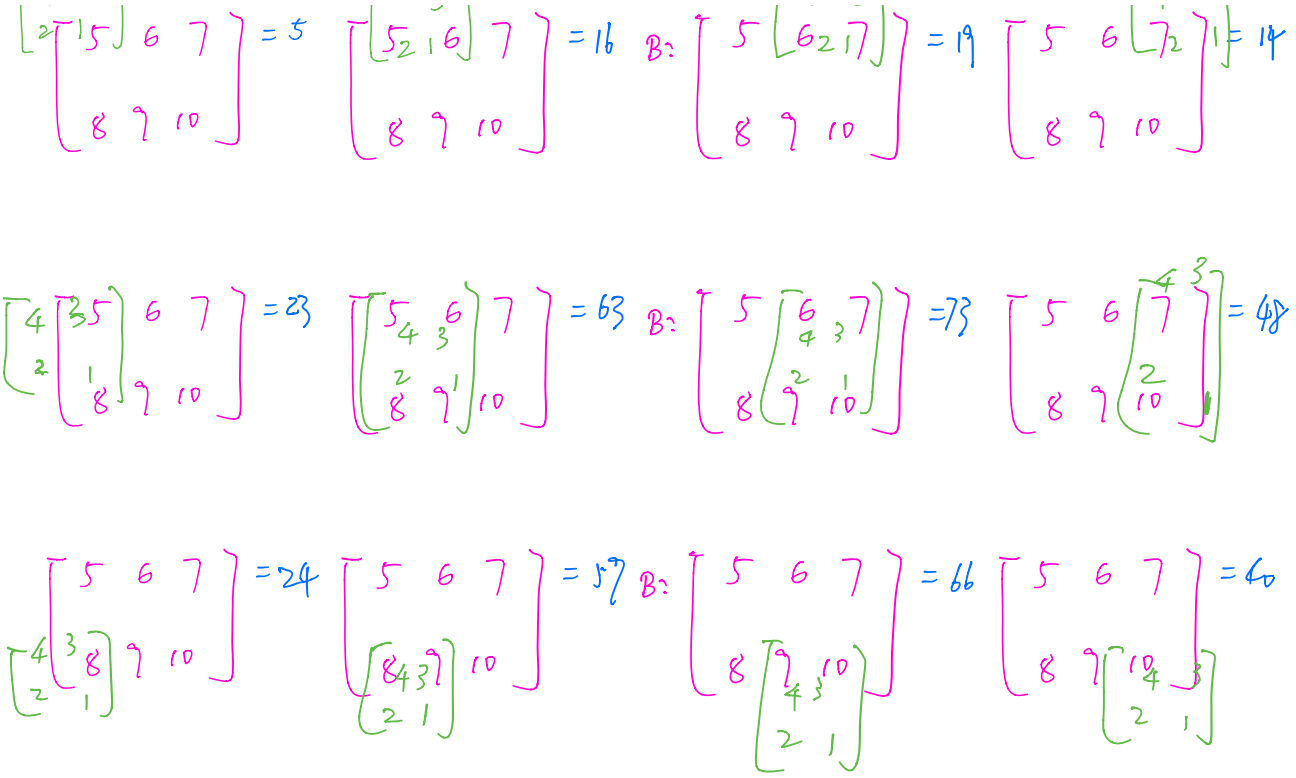
⋮

$A_{3\times3}$, $B_{2\times4}$ → conv2 → $C_{6\times6}$

另一种更直观的解释

↓

$\begin{bmatrix}4 & 3\end{bmatrix}$　$\begin{bmatrix}4 & 2\end{bmatrix}$　$\begin{bmatrix}4 & 3\end{bmatrix}$　$\begin{bmatrix}4 & 3\end{bmatrix}$



(numImages, numFilters, imageRow, imageCol) 需要变换形状

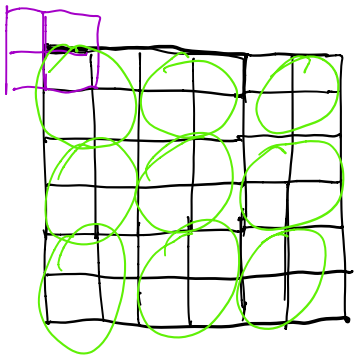
60000 100 28 28

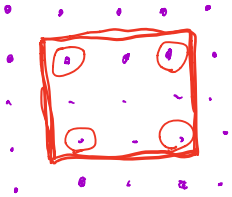
convolvedFeatures 的 shape: (imageRow, imageCol, featureNum, imageNum)

---

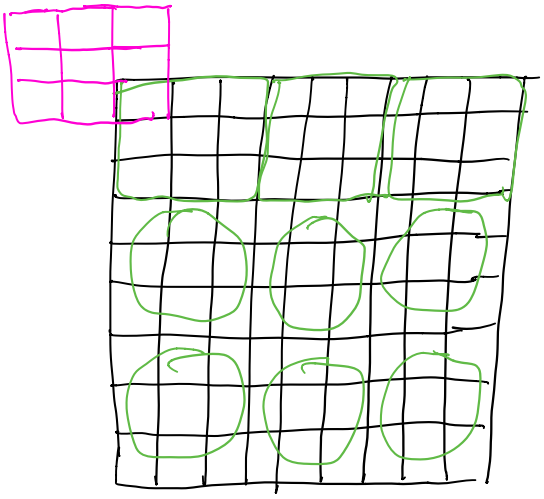
pool:

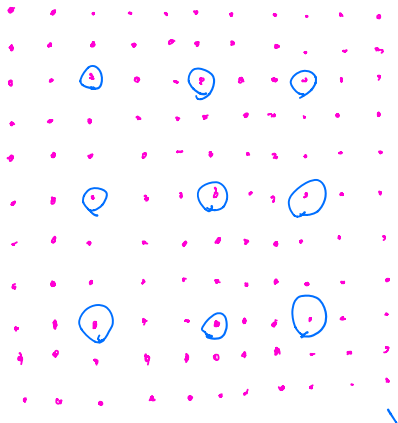
21, 21, 100, 8





用for循环来pool: im((i-1)*poolDim+1 : i*poolDim, (j-1)*poolDim+1 : j*poolDim)





用 conv2 来 pool：(poolDim*(i-1)+1 , poolDim*(j-1)+1)

⑦ Optimization：Stochastic Gradient Descent　　LTribe

good off the shelf 现成的

Batch Optimization Method：能往直收敛至局部最优，实现容易，需要调整的超参数少。

Stochastic Gradient Descent：不一次用整个数据集，只用一个或一些，解决反向传播花销大的问题

通常用的是 mini-batch：256，其中的学习率 α 也要比 BGD 中小

$\theta = \theta - \alpha \nabla_\theta E[J(\theta)]$ ⟶ $\theta = \theta - \alpha \nabla_\theta J(\theta; x^{(i)}, y^{(i)})$

学习率的正确选择是超参数的
- 标准的做法：~~000~~ 一两轮 epoch 过后，减半 α（固定的值）
- 一个更好的：每个 epoch 过后，如果目标差异小于阈值，就减小 α 进行退火。
- 另一个常用的：随着迭代次数 t 退火：$\alpha = \frac{a}{b+t}$

SGD 输入数据最好是打乱顺序的，不是刻意排列的
randomly shuffle

Momentum（动量；势头；冲力）

⇒ 用 SGD 在 local optimum 附近会振荡，做无效的慢慢漫

long shallow ravine　　steep wall

可以用 momentum：

$v = \gamma v + \alpha \nabla_\theta J(\theta; x^{(i)}, y^{(i)})$

$\theta = \theta - v$

以上起起来就是有 θ = θ - 梯度后的速度

$\gamma \in (0,1]$, generally, γ is set to 0.5 until the initial learning stabilizes and then is increased to 0.9 or higher.

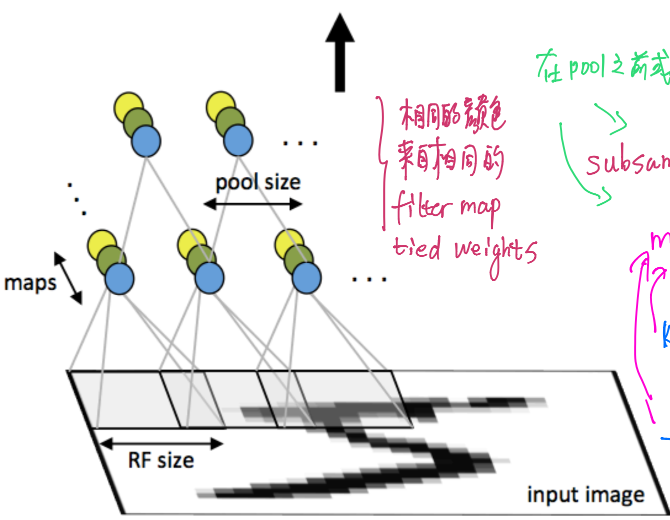
θ, v 维数形状同参数向量, v是当前速度向量, θ是参数向量

⑧ Convolutional Neural Network

architecture
back propagation algorithm

CNN{
- 利用了图像的2维结构
- 在有相同隐神经元时, 参数比全连接网络少

Architecture:



相同的颜色来自相同的 filter map
tied weights

在pool之前或之后会加上偏置和通过激活函数

subsampling: $p \times p$ ($p \in [2,5]$)

map: $m-n+1$ (k个 feature map)

k个 kernel/filter, shape为 $n \times n \times q$ (n 比m小多了, $q \le r$)

输入图片: $m \times m \times r$ (m: height, m: width, r: channels)

Back Propagation

w.r.t. → with respect to 关于;谈及

记 $\delta^{(l+1)}$ 为 (l+1)-st layer 的 error term

如果 l层与 (l+1)层全连接的话, 则 $\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)} \right) \bullet f'(z^{(l)})$

梯度:

$$\nabla_{W^{(l)}} J(W,b;x,y) = \delta^{(l+1)} (a^{(l)})^T$$
$$\nabla_{b^{(l)}} J(W,b;x,y) = \delta^{(l+1)}$$

如果 l层为卷积层与下采样层, 则 $\delta_k^{(l)} = \text{unsample}\left((W_k^{(l)})^T \delta_k^{(l+1)} \right) \bullet f'(z_k^{(l)})$

其中 k 索引着 filter number

梯度：$\nabla_{W_k^{(l)}} J(W,b;x,y) = \sum_{i=1}^{m} (a_i^{(l)}) * \text{rot90}(\delta_k^{(l+1)}, 2)$ → valid convolution

$\nabla_{b_k^{(l)}} J(W,b;x,y) = \sum_{a,b} (\delta_k^{(l+1)})_{a,b}$

$a^{(1)}$ — the input image

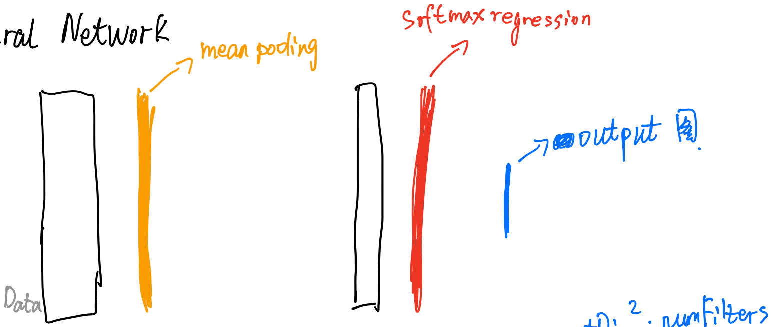
$a^{(l)}$ — the input to the l-th layer ⇒ ??

⑨ Exercise: Convolutional Neural Network

filterDim = 9

numFilters = 20 (网页上说的是10)

poolDim = 2



cnnTrain.m

STEP0: Initialize Parameters and load Data

STEP1: Implement convNet Objective

STEP2: Gradient Check

STEP3: Learn Parameters

STEP4: Test

W_c: (filterDim, filterDim, numFilters) — 9, 9, 20

W_d: (numClasses, hiddenSize) — 10, 2000

hiddenSize → outDim² · numFilters

bc: (numFilters, 1) — 20

bd: (numClasses, 1) — 10

step1a: Forward Propagation

convolution → activation → pooling

→ activationsPooled matrix

(every column → an image)

→ activations (convDim, convDim, numFilters, numImages)

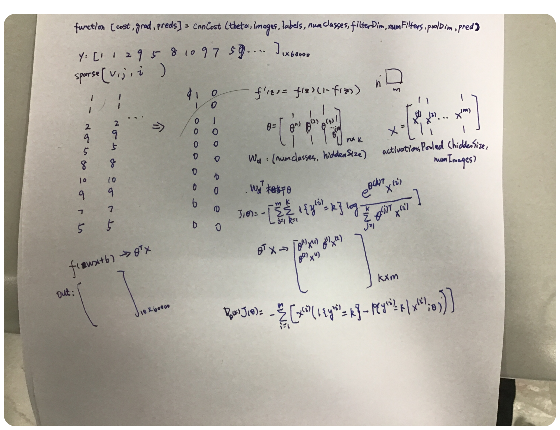
activationsPooled (outputDim, outputDim, numFilters, numImages) — 10, 10, 20

reshape → (hiddenSize, numImages)

W_d * activationsPooled

加上偏置,经过激活函数

→ out (numClasses, numImages)



纸上写的一点笔记

对住躲来了，卷积神经网络下的 back propagation（之前 multi-layer neural networks 的反向传播才懵懵懂懂，现在要着 CNN 的 BP 了！坚持.

跟 forward propagation 正好相反，这时的 error 要从

densely connected layer (softmax) ⟶ subsampling (pooling layer) ⟶ convolutional layer

↑ 像之前那样 ∫(不跨bias)　↑ kron　↑ incoming error * f'(z)

计算梯度的话（Grad_Wc, Grad_bc, Grad_Wd, Grad_bd

densely connected layer ⟶ subsampling ⟶ convolutional layer

↑ 还是和之前一样 W_d, b_d　↑ 因为卷积层有error 所以这里没有到这过　↑ error和原训练图像卷积 W_c, b_c ⇓ 梯度为所有error之和

hadamard积

误差反向传播

如果当前层是全连接层：$\delta^{i,l} = (W^{(l+1)})^T \delta^{i,l+1} \odot \sigma'(z^{i,l})$

如果当前层是卷积层：$\delta^{i,l} = \delta^{i,l+1} * rot180(W^{(l+1)}) \odot \sigma'(z^{i,l})$

如果当前层是池化层：$\delta^{i,l} = upsample(\delta^{i,(l+1)}) \odot \sigma'(z^{i,l})$

} for $l = L-1$ to 2

梯度下降更新权重

如果当前层是全连接层：$W^l = W^l - \alpha \sum_{i=1}^{m} \delta^{i,l} (a^{i,l-1})^T$，$b^l = b^l - \alpha \sum_{i=1}^{m} \delta^{i,l}$

如果当前层是卷积层：对每一个卷积核有：$W^l = W^l - \alpha \sum_{i=1}^{m} \delta^{i,l} * a^{i,l-1}$，$b^l = b^l - \alpha \sum_{i=1}^{m} \sum_{u,v} (\delta^{i,l})_{u,v}$

} for $l = 2$ to L

Step3: learn Parameters:

Batch method: L-BFGS —train→ a convolutional network ⟶ 一次迭代需花费几分钟或者更久

SGD + Momentum

iteration 和 epoch 的区别

velocity vector → minFuncSGD.m

parameter vector

SGD: $\theta = \theta - \alpha \nabla_\theta J(\theta; x^{(i)}, y^{(i)})$

momentum: $v = \gamma v + \alpha \nabla_\theta J(\theta; x^{(i)}, y^{(i)})$

$\theta = \theta - v$

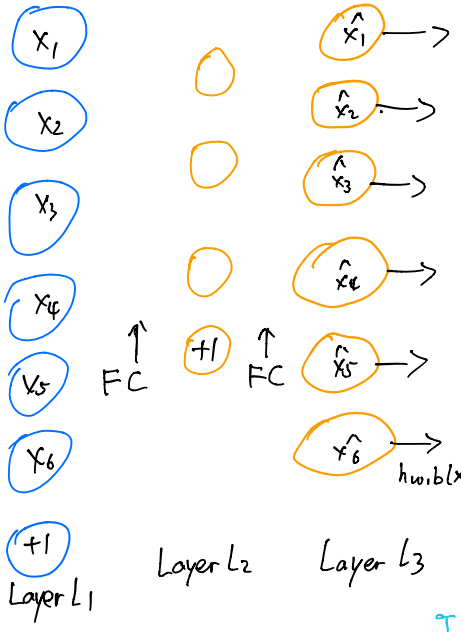
9745

1 257 513 ···· 59649 59905

59745

(10): Autoencoders:

An autoencoder neural network is ① an unsupervised learning algorithm that applies ② backpropagation, setting the ③ target to be equal to the inputs. I.e., it uses $y^{(i)} = x^{(i)}$

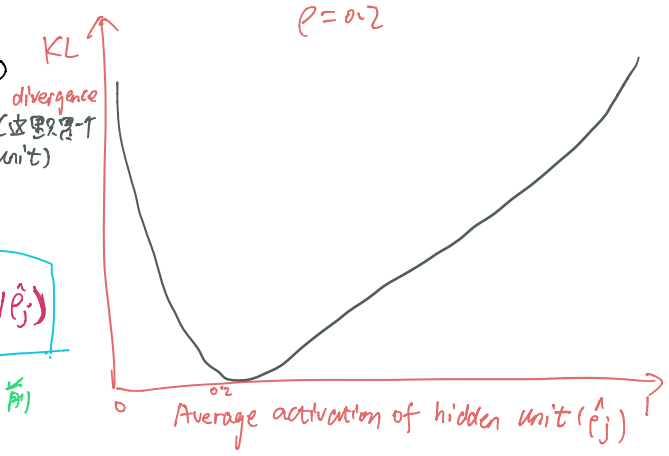


$$\hat{\rho}_j = \frac{1}{m}\sum_{i=1}^{m}\left[a_j^{(2)}(x^{(i)})\right]$$ average activation over the training set

$\hat{\rho}_j = \rho$ (回馈) → 趋近于0，自比较0.05

惩罚项: $\sum_{j=1}^{s_2} \rho \log\frac{\rho}{\hat{\rho}_j} + (1-\rho)\log\frac{1-\rho}{1-\hat{\rho}_j}$ (这里是隐藏层所有作为惩罚)

$= \sum_{j=1}^{s_2} KL(\rho \| \hat{\rho}_j)$ Kullback-Leibler (KL) divergence



overall cost function

$$J_{sparse}(W,b) = J(W,b) + \underbrace{\beta\sum_{j=1}^{s_2} KL(\rho\|\hat{\rho}_j)}_{J'}$$

$$\delta_i^{(2)} = \left(\sum_{j=1}^{s_2} W_{ji}^{(2)}\delta_j^{(3)}\right) f'(z_i^{(2)}) \leftarrow$$ 之前

$$\delta_i^{(2)} = \left(\left(\sum_{j=1}^{s_2} W_{ji}^{(2)}\delta_j^{(3)}\right) + \beta\left(-\frac{\rho}{\hat{\rho}_i} + \frac{1-\rho}{1-\hat{\rho}_i}\right)\right) f'(z_i^{(2)}) \leftarrow$$ 现在

$$\frac{\partial J'}{\partial z_i^{(2)}} = \beta\left(\rho\cdot\frac{\hat{\rho}_i}{\rho}\left(-\frac{\rho}{\hat{\rho}_i^2}\right) + (1-\rho)\cdot\frac{1-\hat{\rho}_i}{1-\rho}\cdot\frac{1-\rho}{(1-\hat{\rho}_i)^2}\right) f'(z_i^{(2)})$$

$$= \beta\left(-\frac{\rho}{\hat{\rho}_i} + \frac{1-\rho}{1-\hat{\rho}_i}\right)$$

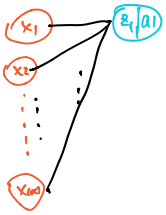
要想让 hidden unit i 最大化地激活，应该使输入 x_j (for $j = 1, \dots, 100$) 为

$$x_j = \frac{W_{ij}^{(1)}}{\sqrt{\sum_{j=1}^{100}\left(W_{ij}^{(1)}\right)^2}}$$ (试着自己证明一下)

visualization: one image per hidden unit. → learned: hidden units learn to detect edges at different positions and orientations

$a_i^{(2)} = f\left(\sum_{j=1}^{100} W_{ij}^{(1)} x_j + b_i^{(1)}\right)$

如果限制 $\|x\|^2 = \sum_{i=1}^{100} x_i^2 \le 1$



不考虑bias的情况下, $a_i^{(2)} = f\left(\sum_{j=1}^{100} W_{ij}^{(1)} x_j\right)$ 要让 $a_i^{(2)}$ 尽可能趋近于1

则 $\sum_{j=1}^{100} W_{ij}^{(1)} x_j$ 要最大 ??

② PCA Whitening

Principal components analysis (PCA) → 降维 → 无监督特征学习

covariance matrix 协方差矩阵

eigenvector / aɪgənvektə(r) / 特征向量, 本征向量

Σ — covariance matrix standard notation

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}(x^{(i)})^T \quad (n \times n)$$

他的特征向量组成的矩阵 U

$$U = \begin{bmatrix} | & | & & | \\ u_1 & u_2 & \cdots & u_n \\ | & | & & | \end{bmatrix}$$ ⇒ 正交矩阵 $U^TU = UU^T = I$

principal

$\lambda_1 \quad \lambda_2 \cdots \lambda_n$

对应最大特征值

$u_1^T x$ — x 在 u_1 方向上的投影

$u_2^T x$ — x 在 u_2 方向上的投影

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}_{n \times m}$$

$$U = \begin{bmatrix} | & | & & | \\ u_1 & u_2 & \cdots & u_n \\ | & | & & | \end{bmatrix}_{n \times n}$$

(在这里 n=2 累了)

$$\begin{bmatrix} u_1^T \\ u_2^T \\ \vdots \\ u_n^T \end{bmatrix} X$$

Rotating

$$X_{rot} = U^T X = \begin{bmatrix} u_1^T x \\ u_2^T x \\ \vdots \end{bmatrix}$$

$$X = U X_{rot}$$

$$U^T \begin{bmatrix} x^{(1)} & x^{(2)} & \cdots & x^{(m)} \end{bmatrix}$$

Reducing

$\lfloor U_k^T X \rfloor_{n \times m}$

$$\hat{x} = \begin{bmatrix} x_{rot,1} \\ \vdots \\ x_{rot,k} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \approx \begin{bmatrix} x_{rot,1} \\ \vdots \\ x_{rot,k} \\ x_{rot,k+1} \\ \vdots \\ x_{rot,n} \end{bmatrix} = x_{rot}$$

$x_{rot,1}^{(i)} = u_1^T x^{(i)}$ 丢掉没用的特征.

也把 $\tilde{x} = \begin{bmatrix} x_{rot,1} \\ \vdots \\ x_{rot,k} \end{bmatrix}_{k \times n} = \begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \\ \vdots \\ \tilde{x}_k \end{bmatrix}$

Recovering

$$\hat{x} = U \begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \\ \vdots \\ \tilde{x}_k \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \sum_{i=1}^{k} u_i \tilde{x}_i$$ (实际上我们是让 $\tilde{x} \in \mathbb{R}^k$ 乘以 U 的前 k 列)

Number k of components to retain

用这个描述 percentage of variance retained : λ_i 为特征值, 且 $\lambda_1 \to \lambda_n$ 降序排列

$$\frac{\sum_{j=1}^{k} \lambda_j}{\sum_{j=1}^{n} \lambda_j}$$

图像有一种性质叫作: stationarity

通常用PCA的时候, 都要让各特征对应的值有 0 均值, 相同的变化范围 (① ②)

对于 natural images, 一般不需要做第②步: variance normalization

只需做第①步:

for all j:

$$\mu^{(i)} := \frac{1}{n} \sum_{j=1}^{n} x_j^{(i)} \qquad x_j^{(i)} := x_j^{(i)} - \mu^{(i)}$$

跟PCA降维联系相对紧密的一个预处理步骤：whitening / sphering

$$X_{PCAwhite,i} = \frac{X_{rot,i}}{\sqrt{\lambda_i}}$$ ，使协方差矩阵为单位矩阵 I；

作用：使输入冗余性降低：
① 令特征之间相关性降低
② 令特征有相同的方差
X_{rot}

如果 R 为正交矩阵（即 $R^T R = RR^T = I$）

则 $RX_{PCAwhite}$ 也有单位协方差矩阵

$X_{ZCAwhite} = UX_{PCAwhite}$ （选 R=U），这样使 $X_{ZCAwhite}$ 最接近 X（原始图像）

ZCA whitening 一般不降维 !!

Regularization

有些 λ_i 接近于零，$\frac{1}{\sqrt{\lambda_i}}$ ⟹ 会很大，所以加个固定值 ε

使：$X_{PCAwhite,i} = \frac{X_{rot,i}}{\sqrt{\lambda_i + \varepsilon}}$

$X \in [-1, 1]$
↓
ε 大约为 10^{-5}（也有平滑，低通滤波的效果）

$X \xrightarrow{\text{ZCA whitening}} X_{ZCAwhite}$

类似人眼，去相邻像素相关化

(12) Sparse coding

↳ 目的：找到一组的基底向量 ϕ_i，使我们可以用其线性组合来表示一个输入向量

$$x = \sum_{i=1}^{k} a_i \phi_i \qquad x \in R^n \quad (k > n)$$

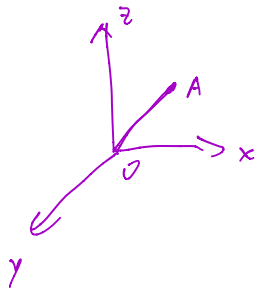
learn an over-complete set of basis vectors to represent input vectors

↓

Sparsity → 要使系数尽可能多的为0

→ 防止对同一个 x 表示时，a_i 的不唯一

就如像在三维空间中 $\vec{OA} = (2,3,4)$，i, j, k 为 x, y, z 轴方向上的基底向量



本来 $\vec{OA} = 2\vec{i} + 3\vec{j} + 4\vec{k} = \sum_{i=1}^{3} a_i \phi_i$

但是现在要用4个或更多向基底来表示 ⟹ over-complete 过完备的

定义 Sparse coding cost function（m 个输入向量）

$$\text{minimize}_{a_i^{(j)}, \phi_i} \sum_{j=1}^{m} \left\| x^{(j)} - \sum_{i=1}^{k} a_i^{(j)} \phi_i \right\|^2 + \lambda \sum_{i=1}^{k} S(a_i^{(j)})$$

这一块体现"无监督"　　$S(\cdot)$ 当 a_i 远离0时就惩罚越厉害

惩罚项的几种选择：

① L_0 norm　$S(a_i) = 1 \ (|a_i| > 0)$　不可导且难优化

② L_1 norm　$S(a_i) = |a_i|$　} 常用

③ log　$S(a_i) = \log(1 + a_i^2)$

一个问题，$\lambda \sum_{i=1}^{k} S(a_i^{(j)})$ 可以很小，当 a_i 减小时，但这会使 $\left\| x^{(j)} - \sum_{i=1}^{k} a_i^{(j)} \phi_i \right\|^2$ 中的 ϕ_i 增大，为了 solve this，加一个限制：$\|\phi\|^2 <$ 常数 C。所以完整的目标函数是：

$$\text{minimize}_{a_i^{(j)}, \phi_i} \sum_{j=1}^{m} \left\| x^{(j)} - \sum_{i=1}^{k} a_i^{(j)} \phi_i \right\|^2 + \lambda \sum_{i=1}^{k} S(a_i^{(j)})$$

$$\text{subject to} \quad \|\phi_i\|^2 \le C, \ \forall i = 1, \dots, k$$

probabilistic interpretation

sparse coding ⟶ a generative model (概率上的可解释性)

L_1 sparsity penalty —$a_i^{(j)}$ (convex)→ 共轭梯度法 (conjugate gradient method)

L_2 norm constraint —ϕ (convex)→ 拉格朗日对偶法 (Lagrange dual)

来一个新数据，得重新优化来得到 "coefficients". —— computationally expensive

ICA (independent component analysis) 独立成分分析

↳ 不是线性无关的基底，而是"正交基底" (orthonormal basis)

↳ basis $(\phi_1, \cdots \phi_n)$ $\begin{cases} \phi_i \cdot \phi_j = 0 & \text{if } i \neq j \\ \phi_i \cdot \phi_j = 1 & \text{if } i = j \end{cases}$

W (列向量之间是一系列基底，正交且稀疏)

sparse coding : mapping features to raw data $x = \sum_{j=1}^{k} a_j \phi_j$

ICA : mapping raw data x to features

orthonormal ICA objective :

minimize $\|Wx\|_1$

s.t. $WW^T = I$

① 学到的基底数 < 输入维度，即 under-complete basis's

② 数据必须是 ZCA whitened with no regularization

优化目标函数，使用梯度下降法：

repeat until done:

1. $W \in W - \alpha \nabla_W \|Wx\|_1$ (必须用 line-search 算法来改变着进行梯度下降)
2. $W \in \text{proj}_U W$ U是满足 $WW^T = I$ 的矩阵空间 ($W \in (WW^T)^{-\frac{1}{2}} W$)

ZCA : U·PCA

U·( )U'·x ⟹ 这里教程中好像还有待补充说明：为什么像 ZCA whitening

RICA (Reconstruction ICA)

用 soft reconstruction penalty 来代替 ICA 的正交约束，为的解决 ICA 的缺陷

$$\min_W \ \lambda\|Wx\|_1 + \frac{1}{2}\|W^TWx - x\|_2^2$$

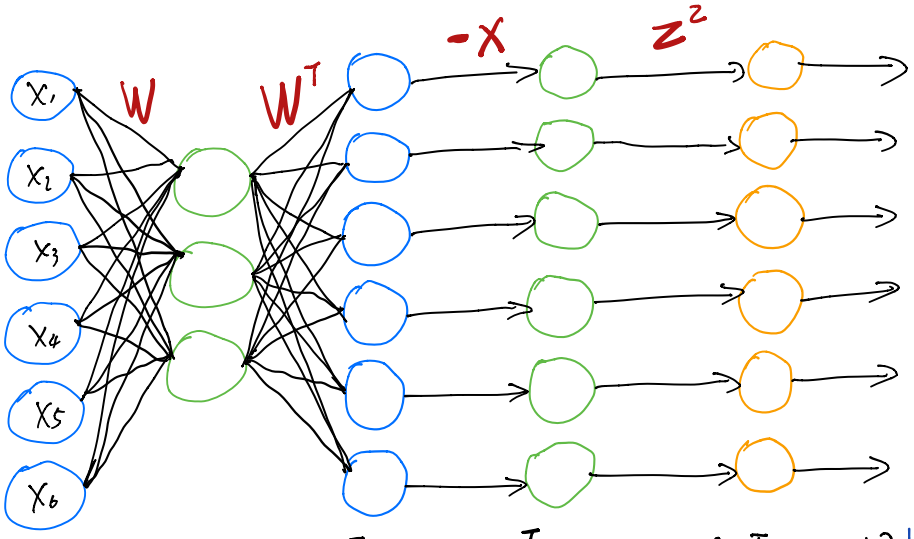
相差个 sigmoid 非线性

sparse autoencoders：

$$\min_W \ \lambda\|\sigma(Wx)\|_1 + \frac{1}{2}\|\sigma(W^T\sigma(Wx)) - x\|_2^2$$

退化控制

RICA 对 over-complete basis 和 non-whitened data 要更鲁棒.



	x	Wx	W^TWx	W^TWx-x	$(W^TWx-x)^2$
层数	1	2	3	4	
权重	W	W^T	I	N/A	
激活函数	$f(z_i)=z_i$	$f(z_i)=z_i$	$f(z_i)=z_i-x_i$	$f(z_i)=z_i^2$	
导数	$f'(z_i)=1$	$f'(z_i)=1$	$f'(z_i)=1$	$f'(z_i)=2z_i$	
Delta	$(W^T\delta^{(2)})\cdot 1$	$((W)^T\delta^{(3)})\cdot 1$	$(I^T\delta^{(4)})\cdot 1$	$f'(z_i)=2z_i$	
Input z to this layer	x	Wx	W^TWx	W^TWx-x	

在神经网络中多次出现 W，对其梯度就是对每个 W 的简单加和.

$\nabla_{W^T} F = \delta^{(3)} (a^{(2)})^T = 2(W^T W x - x) \cdot (Wx)^T$

$\nabla_W F = \delta^{(2)} (a^{(1)})^T = (W)(2(W^T W x - x))(x)^T$

$\nabla_W F = \nabla_W F + (\nabla_{W^T} F)^T$

$= (W)(2(W^T W x - x))(x)^T + 2(Wx)(W^T W x - x)^T$

$\min_W \lambda \|Wx\|_1 + \frac{1}{2} \|W^T W x - x\|_2^2$

梯度维度：50 × 81

$W_{50 \times 81}$

$\times$ 81 × 10000

x	Wx	$(Wx)^2$	$(Wx)^2+\varepsilon$	$\sqrt{(Wx)^2+\varepsilon}$
层数	1	2	3	4
权重	W	I	I	I
激活函数	$f(z_i)=z_i$	$f(z_i)=z_i^2$	$f(z_i)=z_i+\varepsilon_i$	$f(z_i)=\sqrt{z_i}$
导数	$f'(z_i)=1$	$f'(z_i)=2z_i$	$f'(z_i)=1$	$f'(z_i)=\frac{1}{2\sqrt{z_i}}$
Delta	$(W^T\delta^{(2)})\cdot 1$	$(I^T\delta^{(3)})\cdot 2Wx$	$(I^T\delta^{(4)})\cdot 1$	$f'(z_i)=\frac{1}{2\sqrt{z_i}}$
Input z to this layer	x	Wx	$(Wx)^2$	$(Wx)^2+\varepsilon$

$\nabla_W \|Wx\|_1 = (\delta^{(2)})(a^{(1)})^T = \frac{2Wx}{2\sqrt{(Wx)^2+\varepsilon}} x^T \Rightarrow \frac{Wxx^T}{\sqrt{(Wx)^2+\varepsilon}}$

$\delta^{(2)}$ ↑ $(a^{(1)})^T$ ↑

⇒在编程时还要注意一下，看源码实现好了

(13) self-taught learning

Somethings it's not who has the best algorithm that wins; it's who has the most data

很多的无标签数据 —self-taught learning / unsupervised learning→ 特征

+

少量有标签的数据

} 一起 → 喂给 supervised learning

self-taught learning setting：无标签的数据和有标签的数据不一定会有相同的分布

semi-supervised learning：无标签数据必须要和有标签数据同分布

exercise

patches (from unlabeled training set of handwritten digits) (0-9数字)

↓ RICA

filters

labeled dataset (0-4数字)

} convolve → features —softmax regression→ classifier on handwritten digits. (分别0-4数字)

questions: 在阅读CS229 note时产生的问题

① page. 13. σ^2

② page. 15 τ bandwidth parameter

③ page 17 why choose sigmoid function?

④ page 19 linear regression update rule is identical with. logistic regression update rule? why?

$p(x|\eta) = h(x)g(\eta)e^{\eta^T u(x)}$ ⇒指数族一般形式 $p(y;\eta) = b(y)e^{\theta^T T(y) - a(\eta)}$

$p(x|\mu) = \mu^x(1-\mu)^{1-x}$

$e^{\ln \mu^x(1-\mu)^{1-x}}$ $\quad (h_\theta(x))^y(1-h_\theta(x))^{1-y}$

$= e^{\ln \mu^x(1-\mu)^{-x}\cdot(1-\mu)}$

$= (1-\mu)\, e^{x\ln\left(\frac{\mu}{1-\mu}\right)}$

设 $\eta = \ln\frac{\mu}{1-\mu}$, 得 $\mu = \sigma(\eta)$

$p(x|\mu) = [1-\sigma(\eta)]\, e^{\eta\cdot x}$