

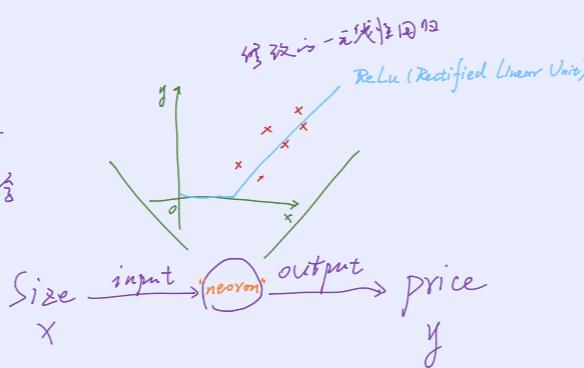
神经网络与深度学习

深度学习概论

什么是神经网络

e.g. 已知房屋大小，预测房价的模型。

很多 neuron 那样所形成的数据模型，称为神经网络模型，称为神经网络。



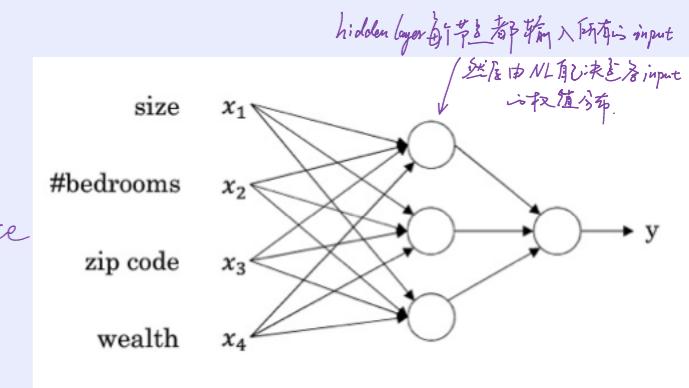
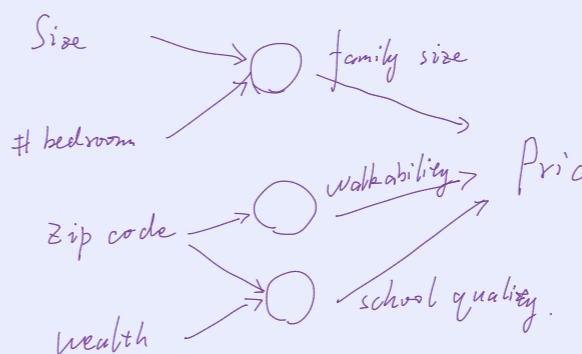
监督学习

已知数据有 output，可以作为训练。

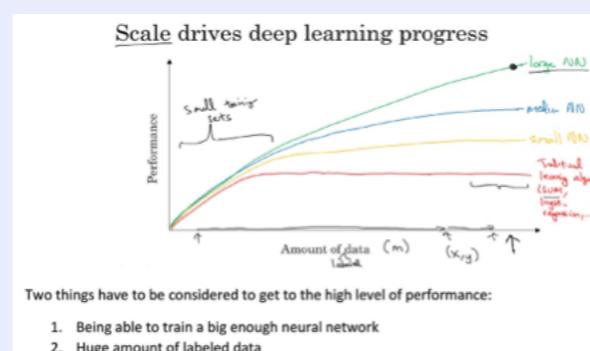
- 结构化数据：以上

- 非结构化数据：音频、图像、文字

e.g.-2. 已知多个因素，预测房价。



Why 深度学习？



神经网络基础

二分类

样本： $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$ 共 m 个样本

其中 $x \in \mathbb{R}^{n_x}$, 即每个 x 有 n_x 维 (n_x 特征) (图片)
 $y \in \{0, 1\}$, 即二分类
 (0 不是猫, 1 是猫)

numpy style: $X. shape = (n_x, m)$

$Y. shape = (1, m)$

$$X = \begin{bmatrix} | & | & \dots & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & \dots & | \\ n_x \times m \end{bmatrix}$$

$$Y = [y_0, y_1, \dots, y_m]$$

Logistic 回归

已知 X (猫图) 欲知 $\hat{y} = P(y=1|x)$ (是猫的概率)

具体模型： $\hat{y} = \delta(w^T x + b)$ 其中 $x, w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$, $\delta(z) = \frac{1}{1+e^{-z}}$
 $\hat{y} \in (0, 1)$

$$\delta'(z) = \delta(z)(1 - \delta(z))$$

当 z 太大时，
 δ 也不会超过 $(0, 1)$ 且单调

Loss function: (单个样本的误差度量)

已知样本集 $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$, 希望 $\hat{y} \approx y$

通常： $L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$, 但 Logistic 因中间选择点非凸，往局部极值，不便梯度下降。

选取： $L(\hat{y}, y) = -[y \log \hat{y} + (1-y) \log(1-\hat{y})]$ 注 $y \in \{0, 1\}$, $\hat{y} \in (0, 1)$ Why? This?

由于 y 表示给定条件 x 的概率，即如果 $y=1$, $P(y|x)=\hat{y}$
 但是为问题相容地，如果 $y=0$, $P(y|x)=1-\hat{y}$ 整合

$L(\hat{y}, y) = -\log(p)$ 由于 $\log(p)$ 不改变单调性 $P(y|x) = \hat{y}^y (1-\hat{y})^{1-y}$

直观理解：若 $y=1$, $L(\hat{y}, y) = -\log \hat{y} \Rightarrow \min_{\hat{y}} L = \max_{\hat{y}} \log \hat{y} = \max_{\hat{y}} \hat{y}$

若 $y=0$, $L(\hat{y}, y) = -\log(1-\hat{y}) \Rightarrow \min_{\hat{y}} L = \max_{\hat{y}} 1 - \hat{y} = \min_{\hat{y}} 1 - \hat{y}$

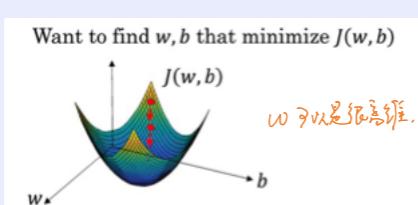
Cost function: (全体样本的误差度量)

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)})$$

$$= -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)})]$$

训练 NL 即调整 w, b , 使得 $J(w, b)$ 最小的过程。learn w, b from dataset according to $J(w, b)$

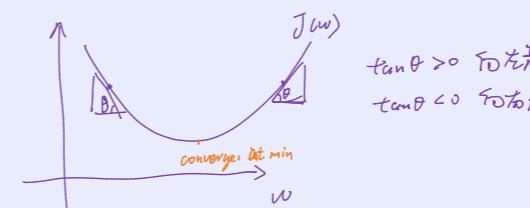
梯度下降 (Back propagation)



Detail:

$$w := w - \alpha \frac{\partial J(w, b)}{\partial w} \quad \text{it's in coding}$$

$$b := b - \alpha \frac{\partial J(w, b)}{\partial b} \quad \text{Learning rate}$$

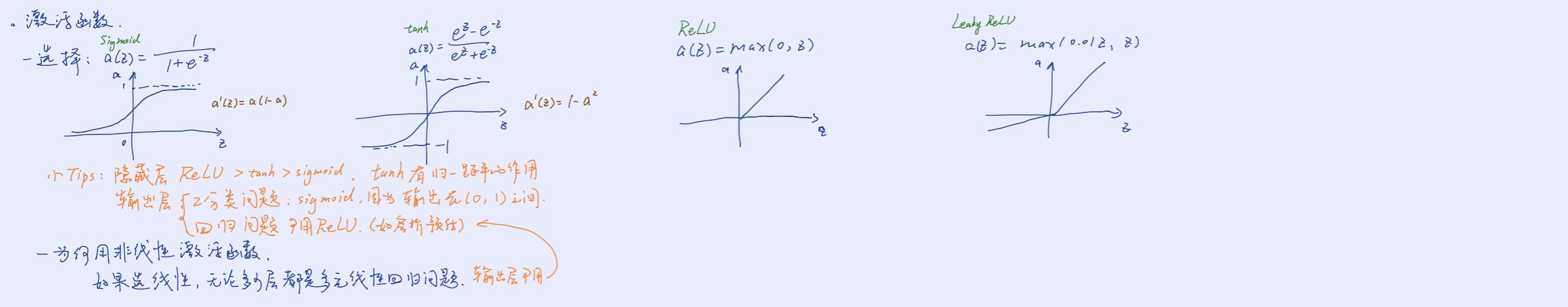


导数与计算图

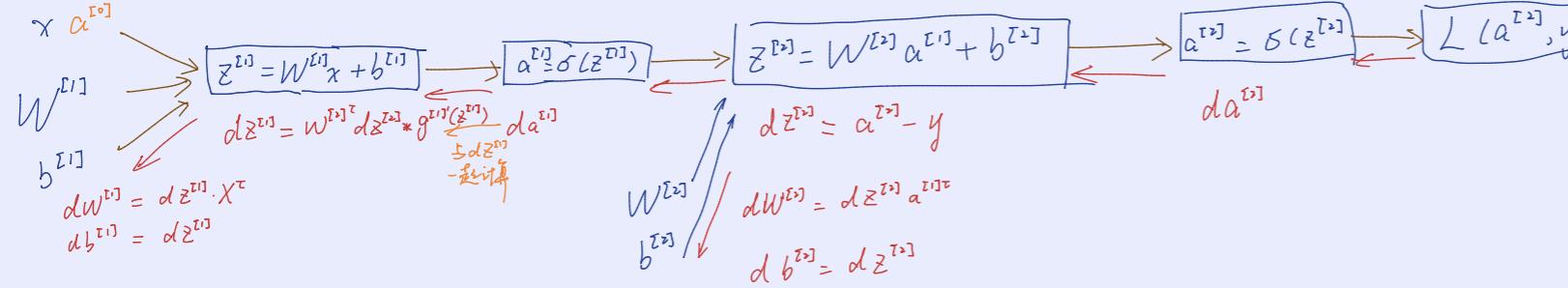
$$\begin{aligned} a &= 5 \\ b &= 3 \\ c &= 2 \end{aligned} \rightarrow \boxed{u = bc} \rightarrow \boxed{v = a+u} \rightarrow \boxed{J = 3v}$$

从左到右很简单，
 从右到左需要导数计算，链式法则

$$\text{直观理解: } a + 0.001, J + ? \quad \frac{?}{0.001} = \frac{\partial J}{\partial a} = \frac{\partial J}{\partial v} \frac{\partial v}{\partial a}$$



。 活跃层 NL 的梯度下降 (计算图表示) (back propagation)



。 小结
Forward Propagation.

$$\begin{aligned} z^{[0]} &= W^{[0]}x + b^{[0]} \\ A^{[0]} &= g^{[0]}(z^{[0]}) \\ z^{[1]} &= W^{[1]}A^{[0]} + b^{[1]} \\ A^{[1]} &= g^{[1]}(z^{[1]}) = \delta(z^{[1]}) \end{aligned}$$

Back propagation.

$$\begin{aligned} dL(a^{[2]}, y) &\rightarrow da^{[2]} \\ da^{[2]} &\rightarrow dz^{[2]} \\ dz^{[2]} &\rightarrow dL(a^{[2]}, y) \\ dL(a^{[2]}, y) &= \frac{1}{m} \sum_{i=1}^m L(a^{[2]}, y_i) \end{aligned}$$

$$\begin{aligned} dL(a^{[2]}, y_i) &\rightarrow dz^{[2]} \\ dz^{[2]} &\rightarrow dW^{[2]} \quad dL(a^{[2]}, y_i) \\ dz^{[2]} &\rightarrow db^{[2]} \end{aligned}$$

$$\begin{aligned} dW^{[2]} &= \frac{1}{m} \sum_{i=1}^m \frac{\partial L(a^{[2]}, y_i)}{\partial z^{[2]}} \cdot dz^{[2]} \\ db^{[2]} &= \frac{1}{m} \sum_{i=1}^m \frac{\partial L(a^{[2]}, y_i)}{\partial b^{[2]}} \cdot dz^{[2]} \end{aligned}$$

$$\begin{aligned} dL(a^{[1]}, y) &\rightarrow da^{[1]} \\ da^{[1]} &\rightarrow dz^{[1]} \\ dz^{[1]} &\rightarrow dL(a^{[1]}, y) \\ dL(a^{[1]}, y) &= \frac{1}{m} \sum_{i=1}^m L(a^{[1]}, y_i) \end{aligned}$$

$$\begin{aligned} dL(a^{[1]}, y_i) &\rightarrow dz^{[1]} \\ dz^{[1]} &\rightarrow dW^{[1]} \quad dL(a^{[1]}, y_i) \\ dz^{[1]} &\rightarrow db^{[1]} \end{aligned}$$

$$\begin{aligned} dW^{[1]} &= \frac{1}{m} \sum_{i=1}^m \frac{\partial L(a^{[1]}, y_i)}{\partial z^{[1]}} \cdot dz^{[1]} \\ db^{[1]} &= \frac{1}{m} \sum_{i=1}^m \frac{\partial L(a^{[1]}, y_i)}{\partial b^{[1]}} \cdot dz^{[1]} \end{aligned}$$

其中: $W^{[l]}$, $b^{[l]}$, $W^{[l]}$, $b^{[l]}$
 $(n^{[0]}, n^{[0]})$, $(n^{[1]}, 1)$, $(n^{[1]}, n^{[1]})$, $(n^{[2]}, 1)$
 $n_x = n^{[0]}, n^{[1]}, n^{[2]} = 1$

$$\text{Cost Function: } J(W^{[0]}, b^{[0]}, W^{[1]}, b^{[1]}, \dots, W^{[L]}, b^{[L]}) = \frac{1}{m} \sum_{i=1}^m L(a^{[L]}, y_i)$$

中间量: $z^{[l]}$, $dL(a^{[l]}, y)$, $dz^{[l]}$, $dL(a^{[l]}, y)$, $dW^{[l]}$, $db^{[l]}$
 $(n^{[l]}, 1)$, $(n^{[l]}, 1)$, $(n^{[l]}, n^{[l]})$, $(n^{[l]}, n^{[l]})$

。 随机初始化.

$$b = np.zeros(n^{[1]}, 1)$$

- $b = 0$ init 该问题

- $W = 0$ init 会导致该层所有节点相同

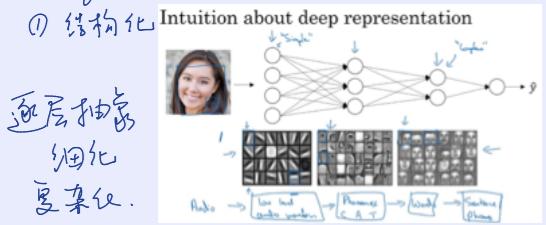
- 用 Sigmoid, tanh 初始化时

$$W = np.random.randn(n^{[0]}, n^{[1]}) * 0.01$$

init值小, 学习快.
收敛快.

深层神经网络

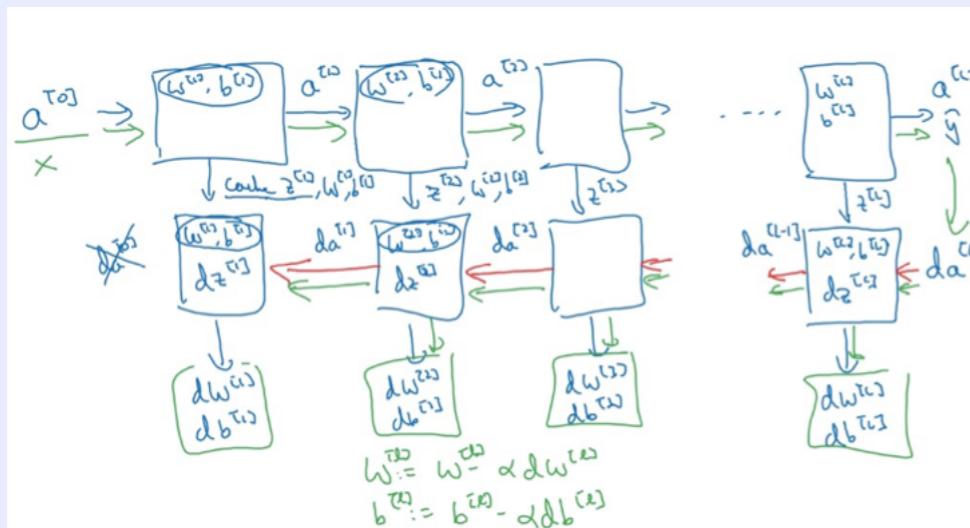
。 Why deep?

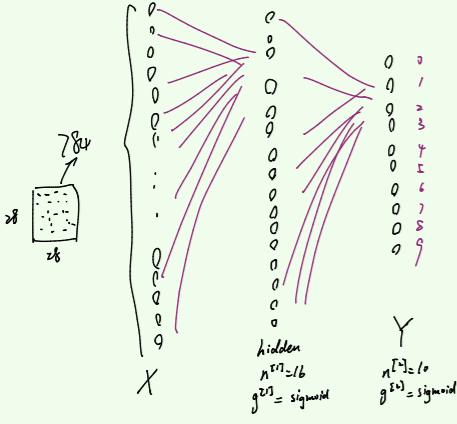


② Circuit theory & DL

导成电路的模型似: 单层: $D(2^n) \xrightarrow{\text{单层}} D$
多层: $D(\log n) \xrightarrow{\text{多层}} D$

。 前向与后向传播





$$X = \begin{bmatrix} | & | & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & | \end{bmatrix}_{784 \times 1}$$

$$Y = \begin{bmatrix} | & | & | \\ y^{(1)} & y^{(2)} & \dots & y^{(m)} \\ | & | & | \end{bmatrix}_{10 \times 1}$$

$$W^{[l]} = \begin{bmatrix} | & | & | \\ w_{11}^{[l]} & w_{12}^{[l]} & \dots & w_{1n}^{[l]} \\ | & | & | \end{bmatrix}_{16 \times 784}$$

$$b^{[l]} = \begin{bmatrix} | & | & | \\ b_{11}^{[l]} & b_{12}^{[l]} & \dots & b_{1n}^{[l]} \\ | & | & | \end{bmatrix}_{16 \times 1}$$

$$Z^{[l]} = \begin{bmatrix} | & | & | \\ w_{11}^{[l]T} & w_{12}^{[l]T} & \dots & w_{1n}^{[l]T} \\ | & | & | \end{bmatrix}_{16 \times 784} \begin{bmatrix} | & | & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & | \end{bmatrix}_{784 \times 1} + \begin{bmatrix} | & | & | \\ b_{11}^{[l]T} & b_{12}^{[l]T} & \dots & b_{1n}^{[l]T} \\ | & | & | \end{bmatrix}_{16 \times m}$$

$$= \begin{bmatrix} | & | & | \\ z^{(1)} & z^{(2)} & \dots & z^{(m)} \\ | & | & | \end{bmatrix}_{16 \times m}$$

$$A^{[l]} = \delta(\begin{bmatrix} | & | & | \\ z^{(1)} & z^{(2)} & \dots & z^{(m)} \\ | & | & | \end{bmatrix}_{16 \times m}) = \begin{bmatrix} | & | & | \\ A^{[l](1)} & A^{[l](2)} & \dots & A^{[l](m)} \\ | & | & | \end{bmatrix}_{16 \times m}$$

$$W^{[l]} = \begin{bmatrix} | & | & | \\ w_{11}^{[l]T} & w_{12}^{[l]T} & \dots & w_{1n}^{[l]T} \\ | & | & | \end{bmatrix}_{16 \times 16}$$

$$b^{[l]} = \begin{bmatrix} | & | & | \\ b_{11}^{[l]T} & b_{12}^{[l]T} & \dots & b_{1n}^{[l]T} \\ | & | & | \end{bmatrix}_{16 \times 1}$$

$$Z^{[l]} = \begin{bmatrix} | & | & | \\ w_{11}^{[l]T} & w_{12}^{[l]T} & \dots & w_{1n}^{[l]T} \\ | & | & | \end{bmatrix}_{16 \times 16} \begin{bmatrix} | & | & | \\ A^{[l](1)} & A^{[l](2)} & \dots & A^{[l](m)} \\ | & | & | \end{bmatrix}_{16 \times m} + \begin{bmatrix} | & | & | \\ b_{11}^{[l]T} & b_{12}^{[l]T} & \dots & b_{1n}^{[l]T} \\ | & | & | \end{bmatrix}_{16 \times m}$$

$$= \begin{bmatrix} | & | & | \\ z^{(1)} & z^{(2)} & \dots & z^{(m)} \\ | & | & | \end{bmatrix}_{16 \times m}$$

$$A^{[l]} = \delta(\begin{bmatrix} | & | & | \\ z^{(1)} & z^{(2)} & \dots & z^{(m)} \\ | & | & | \end{bmatrix}_{16 \times m}) = \begin{bmatrix} | & | & | \\ A^{[l](1)} & A^{[l](2)} & \dots & A^{[l](m)} \\ | & | & | \end{bmatrix}_{16 \times m}$$

$$d\mathcal{E}^{[l]} = \left[\begin{array}{|c|c|c|c|} \hline & A^{[l+1]} & & \\ \hline A^{[l](1)} & A^{[l](2)} & \dots & A^{[l](m)} \\ \hline \end{array} \right]_{16 \times m} - \begin{bmatrix} | & | & | \\ y^{(1)} & y^{(2)} & \dots & y^{(m)} \\ | & | & | \end{bmatrix}_{10 \times m} = \begin{bmatrix} | & | & | \\ d\mathcal{E}^{[l](1)} & d\mathcal{E}^{[l](2)} & \dots & d\mathcal{E}^{[l](m)} \\ | & | & | \end{bmatrix}_{10 \times m}$$

$$dW^{[l]} = \frac{1}{m} \left[\begin{array}{|c|c|c|c|} \hline & d\mathcal{E}^{[l]} & & \\ \hline d\mathcal{E}^{[l](1)} & d\mathcal{E}^{[l](2)} & \dots & d\mathcal{E}^{[l](m)} \\ \hline \end{array} \right]_{10 \times m} \begin{bmatrix} | & | & | \\ A^{[l+1](1)} & A^{[l+1](2)} & \dots & A^{[l+1](m)} \\ \hline \end{bmatrix}_{m \times 16} = \begin{bmatrix} | & | & | \\ dW^{[l](1)} & dW^{[l](2)} & \dots & dW^{[l](m)} \\ | & | & | \end{bmatrix}_{10 \times 16}$$

$$db^{[l]} = \frac{1}{m} \left[\begin{array}{|c|c|c|c|} \hline & d\mathcal{E}^{[l]} & & \\ \hline d\mathcal{E}^{[l](1)} & d\mathcal{E}^{[l](2)} & \dots & d\mathcal{E}^{[l](m)} \\ \hline \end{array} \right]_{10 \times m} \begin{bmatrix} | & | & | \\ 1 & 1 & \dots & 1 \\ | & | & | \end{bmatrix}_{m \times 1} = \begin{bmatrix} | & | & | \\ db^{[l](1)} & db^{[l](2)} & \dots & db^{[l](m)} \\ | & | & | \end{bmatrix}_{10 \times 1}$$

$$d\mathcal{E}^{[l]} = \left[\begin{array}{|c|c|c|c|} \hline & dW^{[l]}_1 & dW^{[l]}_2 & \dots & dW^{[l]}_m \\ \hline dW^{[l]}_1 & dW^{[l]}_2 & \dots & dW^{[l]}_m \\ \hline \end{array} \right]_{10 \times m} \begin{bmatrix} | & | & | \\ d\mathcal{E}^{[l+1]} & & \\ \hline \end{bmatrix}_{16 \times m}$$

elementary wise product

$$\left(\begin{bmatrix} | & | & | \\ A^{[l+1]} & & \\ \hline A^{[l+1](1)} & A^{[l+1](2)} & \dots & A^{[l+1](m)} \\ \hline \end{bmatrix}_{16 \times m} \circ \left(\begin{bmatrix} | & | & | \\ 1 & 1 & \dots & 1 \\ | & | & | \end{bmatrix}_{16 \times m} - \begin{bmatrix} | & | & | \\ A^{[l+1](1)} & A^{[l+1](2)} & \dots & A^{[l+1](m)} \\ \hline \end{bmatrix}_{16 \times m} \right) \right) = \begin{bmatrix} | & | & | \\ & & \\ | & | & | \end{bmatrix}_{16 \times m}$$

$$dW^{[l]} = \frac{1}{m} \left[\begin{array}{|c|c|c|c|} \hline & d\mathcal{E}^{[l]} & & \\ \hline d\mathcal{E}^{[l]} & & & \\ \hline \end{array} \right]_{10 \times m} \begin{bmatrix} | & | & | \\ X^{(1)} & X^{(2)} & \dots & X^{(m)} \\ | & | & | \end{bmatrix}_{m \times 16} = \begin{bmatrix} | & | & | \\ \dots & & \\ | & | & | \end{bmatrix}_{10 \times 16}$$

$$db^{[l]} = \frac{1}{m} \left[\begin{array}{|c|c|c|c|} \hline & d\mathcal{E}^{[l]} & & \\ \hline d\mathcal{E}^{[l]} & & & \\ \hline \end{array} \right]_{10 \times m} \begin{bmatrix} | & | & | \\ 1 & 1 & \dots & 1 \\ | & | & | \end{bmatrix}_{m \times 1} = \begin{bmatrix} | & | & | \\ db^{[l]} & db^{[l]} & \dots & db^{[l]} \\ | & | & | \end{bmatrix}_{10 \times 1}$$

```

init W, b
for i in range(# iterations):
    forward_prop(X, W, b)
    return A2, A, Z
    cost_compute(A2, Y, W, b)
    return cost
    backward_prop(W, b, A, Z, X, Y)
    return dW, db
    update_params(W, b, dW, db)
    return W, b

```

并改善深层神经网络 (超参数调优, 正则化, 优化)

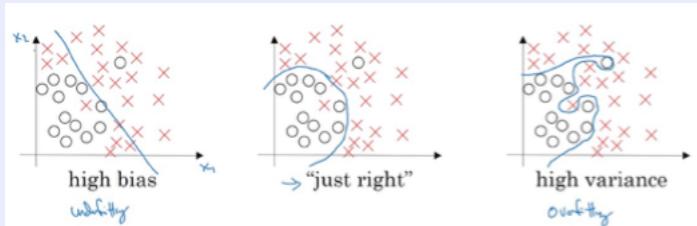
并深入学习的实用层面

训练 / 开发 / 测试集

- 训练集 (training set): 用于训练网络参数.
- 开发集 (dev set): hold-out cross validation set, 用于评估改进模型.
- 测试集 (test set): 网络定型后用于评价该模型性能, 得到无偏估计.

小Tips: training / dev set 要来自于同一分布将更容易训练.

偏差 (Bias) & 方差 (Variance)



Bias过大说明模型的效果不好, 不能达到预期目标.

Variance过大 (Bias不大) 说明存在 overfitting, 即太过度地完美地呈现于 training set, 在样本外不具有 robustness. (不同样本间误差方法)

- 比例 | 小数据: 70/30 (无 dev)
60/20/20 (有 dev)
大数据: 98/1/1
99.5/0.4/0.1

e.g.

case error	①	②	③	④
Training set	1%	15%	15%	0.5%
Dev set	11%	16%	30% (high Bias)	1% (high Variance)

解决 Bias / Variance 过大的方法

- High Bias: ① # layer ↑, # layer nodes (更大之网络)
② # iteration ↑, Better optimization
③ better NN structure, hyper para search

- High Variance: ① Data ↑
② Regularization (L2, dropout, data augmentation)
③ better NN structure, hyperpara search.
④ Cross-Validation done right.

正则化 (Regularization)

$$\|W\|_F^2 = \sum_{j=1}^{n_x} W_j^2 = W^T W$$

- 做法: L2 正则化.

- Logistic Regression: $J(w, b) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \|w\|_2^2$
- Neural Network: $J(W^{[1]}, b^{[1]}, \dots, W^{[L]}, b^{[L]}) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \|W^{[L]}\|_F^2$

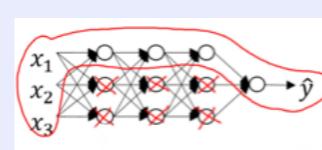
矩阵范数之 Frobinius norm: $\|W\|_F^2 = \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} (W_{ij}^2)$

$$dW^{[l]} = (\text{form_backprop}) + \frac{\lambda}{m} W^{[l]}$$

$$W^{[l]} := W^{[l]} - \alpha dW^{[l]}$$

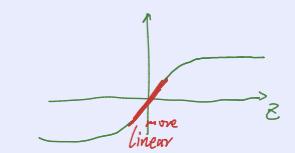
$$= W^{[l]} - \alpha (\text{form_backprop}) - \frac{\alpha \lambda}{m} W^{[l]}$$

= $(1 - \frac{\alpha \lambda}{m}) W^{[l]} - \alpha (\text{form_backprop})$ 其中 $1 - \frac{\alpha \lambda}{m} < 1$, 所以 L2 正则化也称 "Weight Decay"



- 原理: (Why it works)

- 直观理解: 当入 $\rightarrow \infty$, 为使 J 最小化, 梯度下降时 W 会更新得很小, 接近于 0, 这时相当于废弃了一部分节点, 缩小了 NN, 减小了非线性作用.
- 数学解释: 当入 $\rightarrow \infty$, $W^{[l+1]} \downarrow, z^{[l+1]} = W^{[l+1]} a^{[l+1]} + b^{[l+1]} \downarrow, \tanh(z) / \delta(z)$ 在 $0 \in (-\varepsilon, \varepsilon)$ 附近更加线性, 当所有结点都是线性激活的 NN 成了多元线性模型, 不会过拟合.



Dropout 正则化

- 做法: 随机失活, 为每层网络设置一个 keep-prob (保留概率)

$$\text{keep_prob} = 0.8$$

$$\text{d}z = \text{np.random.rand}(a3.shape[0], a3.shape[1]) < \text{keep_prob}$$

$$a3 = \text{np.multiply}(a3, dz)$$

$$a3 /= \text{keep_prob}$$
 (恢复期望值) $z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$ 中 $a^{[l-1]}$ 为期望不改变.

! Tips: 在开发中 (Training, Dev) 可以使用 dropout, 尤其计算机视觉, 但测试中不使用, 否则会使结果随机化.

- 原理:

由于每个 iteration 失活的节点不一样, 这样整个网络要给出准确预测不能太依赖某一输入, 改值可以更好地泛化.

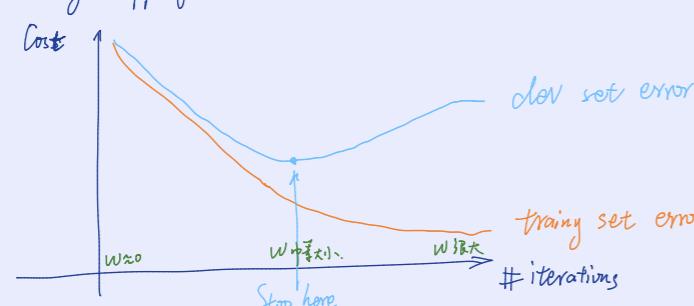
- 不足: Cost function 不能明确规定 (随机消除). 一般除了数据集太小 (计算机视觉方面) 不用 dropout.



数据扩增 (data augmentation)

数据集不足时 通过改变数据 (不失去根本特征的前提下), 提升训练集大小.

Early-stopping.



- 做法:

当 training set error {变小}

且 dev set error {会增加} 的时, 中途停止更新 w, b

- 缺点:

有损 Orthogonalization (改"模块化"), 使问题更加复杂
对梯度分析.

Orthogonalization 实现:

由于 NN 很复杂, 模块化很重要, 整体建分成几步骤:

- ① Optimize cost J
- 工具: 梯度下降, ...

- ② Handling overfitting.
- 工具: 正则化, ...

归一化输入 (Normalizing Input)

- 做法: $X := \frac{X - \mu}{\sigma^2}$

小Tips: 在 test set 中也要使用相同的 μ, σ^2 .

- 原理:

不同量级, Normalize β.

Unnormalized: $\beta = \frac{1}{m} \sum_{i=1}^m \beta_i$

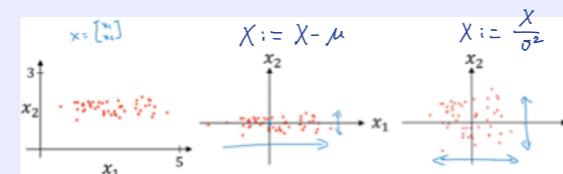
Normalized: $\beta = \frac{1}{m} \sum_{i=1}^m \beta_i - \frac{1}{m} \sum_{i=1}^m \beta_i$

不同量级, 训练更简单有效.

Unnormalized: $w = \frac{1}{m} \sum_{i=1}^m w_i$

Normalized: $w = \frac{1}{m} \sum_{i=1}^m w_i - \frac{1}{m} \sum_{i=1}^m w_i$

不同量级, 是否 Normalization.

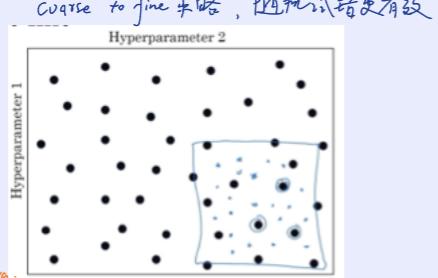


小Tips: Numpy 通常按照行操作, 都做 Normalization.

#超参数与Batch Norm

· 调参流程.

- 先宏观
 - α : learning rate
 - β : Momentum rate
 - # hidden units: 单元数
 - batch size
 - # layers
 - learning rate decay
 - $\beta_1, \beta_2, \epsilon = [0.9, 0.99, 10^{-8}]$ Adam para.



· 调参风格.

Panda: 人工照看, 精细调节.

Cavior: 大量放飞, 草机狂魔.

- 在合适范围内调参

如 α 的测试范围是 $0.0001 \sim 1$, 运行 100 组不同 α 的 ensemble.

Plan A: α Linear 分布在 $(0.0001, 1)$ X 因为 100 组相对于 $0.0001 \sim 1$ 差别太大, Plan A ($0.0001, 1$) 只有 10% ($0, 1, 1$) 在 90% 检索区间效率太低.

Plan B: α log 分布在 $(0.0001, 1)$ ✓ Plan B 更有效地找到全局最优值.

$$y = -4 * np.random.rand()$$

$$\alpha = 10^y$$

· Batch Normalization.

- 动机: 对训练集 X 进行 Norm 可以加速训练, 经过了几层后 $\alpha^{(L)}$ 是不 Normal 的.

那么每层都 Norm 总不美观.

- 一级进: 并不直接操作 $\alpha^{(L)}$, 而是 Normalize $\tilde{z}^{(L)}$

$$\text{等步: } \tilde{z}^{(L)(i)} = \frac{z^{(L)(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

$$\text{第二步: } \hat{z}^{(L)(i)} = \gamma \tilde{z}^{(L)(i)} + \beta$$

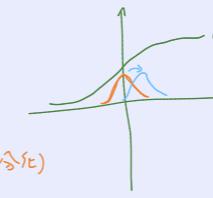
其中 γ 为 rescale 系数 (由于所有 $\tilde{z}^{(L)}$ 都服从正态分布并不一定是我们需要, 通过 rescale 来重新分布)

与 w, b 联动

- 效益 ① 可以使参数搜索更容易.

② NN 对超参数 robust

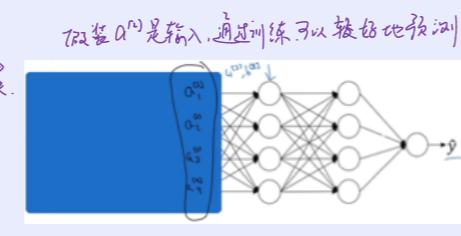
③ 收敛更快(易训练)



- 原理: ① ② Normalizing Input

② 使 NN 更深层, 每层更独立.(见 e.g.)

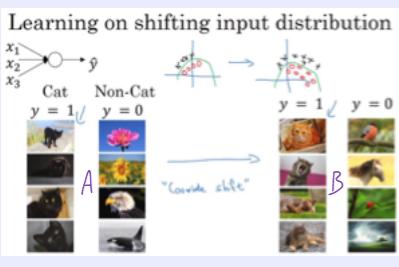
用 A 组训练的 NN 去预测 B 效果不好, 因为 A 中样本都是正态, 与 B 不同分布, 称为 covariate shift



自然常数 $a^{(L)}$ 是否根据不同的输入样本分布而变 (covariate shift)?

那么怎样尽全力保持 $a^{(L)} \rightarrow y$ → 良好预测技巧呢? 答案! 如何稳定? Batch Norm, 首先保证其从 $\alpha^{(L)}$ 变化不大.

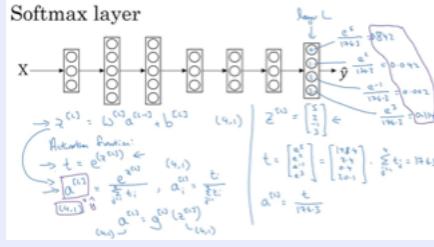
这样, 后面层要到输出较小, 层与层分工更明确, 要独立.



· Tips: 使用 Batch Norm 在 test set 上正确打开方式:
 μ, σ 用各 mini-batch 中 $\mu^{(i)}, \sigma^{(i)}, \dots, \mu^{(n)}, \sigma^{(n)}$ 换取和平均.

· Softmax 分类器.

- 假设: 如果最终不是希望二分类, 而是多分类, 那么最后一层不能只有一个节点.



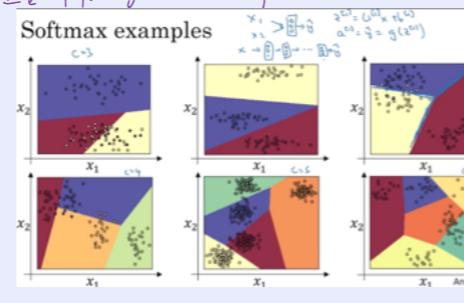
- 计算:

$$L(\hat{y}, y) = -\sum_{j=1}^n y_j \log \hat{y}_j$$

Cross-entropy

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}_i, y_i) \quad \frac{\partial J}{\partial z^{(L)}} = \partial z^{(L)} = \hat{y} - y.$$

正负单节点 Logistic 回归, Softmax 也可以单层使用:



I know soft, what's hard?

$$\begin{aligned} (4,1) & z^{(L)} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \quad t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix} \\ & \text{Soft max} \\ & \alpha^{(L)} = g^{(L)}(z^{(L)}) = \begin{bmatrix} e^5/(e^5 + e^2 + e^{-1} + e^3) \\ e^2/(e^5 + e^2 + e^{-1} + e^3) \\ e^{-1}/(e^5 + e^2 + e^{-1} + e^3) \\ e^3/(e^5 + e^2 + e^{-1} + e^3) \end{bmatrix} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix} \\ & \text{hard max} \end{aligned}$$

· Tensorflow 框架.

e.g. if $f(w) = (w - 5)^2$ → 极小值.

① import numpy as np
import tensorflow as tf

② w = tf.Variable(0, dtype=tf.float32)
cost = w^2 - 10w + 25
train = tf.train.GradientDescentOptimizer(0.01).minimize(cost)

③ init = tf.global_variables_initializer()
init session = tf.Session()

④ session.run(init) # print(session.run(w)) => 0.0
session.run(train) # print(session.run(w)) => 0.1
for i in range(1000):
 session.run(train) # print(session.run(w)) => 6.999.

$$\text{argmin } L(\hat{y}, y) \Rightarrow \text{argmax } (\hat{y}) \Rightarrow \text{argmax } (\hat{y}_i)$$

当 $n^{(L)}=2$, 变化为 Logistic 回归, 且由于 $\alpha^{(L)} + \alpha^{(L)} = 1$, 只需要 $\alpha^{(L)}$ 即可.

e.g. if $f(w) = (w - x)^2$ → 极小值.

① import numpy as np
import tensorflow as tf

coefficients = np.array([2, -1, 1])
x = tf.placeholder(tf.float32, [3, 1])
② w = tf.Variable(0, dtype=tf.float32)
cost = x[0][0] * w + x[1][0] * w + x[2][0]
train = tf.train.GradientDescentOptimizer(0.01).minimize(cost)

③ init = tf.global_variables_initializer()
init session = tf.Session()

④ session.run(init) # print(session.run(w)) => 0.0
session.run(train, feed_dict={x: coefficients})
for i in range(1000):
 session.run(train, feed_dict={x: coefficients})

并行化机器学习项目 并行化学习策略

- 正交化

由于所处理的问题比较复杂，我们通常希望拆分为几个小的问题分步解决。如果几个小问题相互牵连，调整起来十分不方便。

即我们希望各步骤间是正交的。e.g. 当我们开车时，我们希望刹车、油门、档位之间是正交的。

具体而言，四方面应正交：

- Training set 表现良好 or 更好的 NN，优化算法
- Dev set 表现良好 or 正则化，更大的 Training set
- Test set 表现良好 or 更大的 Dev set
- 真实应用中表现良好 or 修改 Dev/Test set，修改 loss func

单一数据的评估指标

明确 Target 才能不跑偏，2 个假阳性作为评判参考：

$$\text{Precision} = \frac{\text{True 1}}{\# \text{predicted 1}} = \frac{\text{True 1}}{\text{True 1} + \text{False 1}}$$

$$\text{Recall} = \frac{\text{True 1}}{\# \text{actual 1}} = \frac{\text{True 1}}{\text{True 1} + \text{False 1}}$$

通常用一个指标来评估模型的好坏会比较方便。如 F1 Score = $\frac{2}{P+R}$ (调和均值)

0	1	0
1	True 1	False 1
0	False 0	True 0

e.g. 1. 使用均值作为评估标准。

Algorithm	US	China	India	Other	Average
A	3%	7%	5%	9%	6%
B	5%	6%	5%	10%	6.5%
C	2%	3%	4%	5%	3.5%
D	5%	8%	7%	2%	5.25%
E	4%	5%	2%	4%	3.75%
F	7%	11%	8%	12%	9.5%

设置 Dev/Test set 评估指标

需求的改变可以通过对指标以及 cost func 的 Tuning 来满足。

e.g. 改变惩罚规则：

由于不希望由分类器向客户推送 porn，所以要在 cost 中对 porn 惩罚。

	error	错误成分
分类器 A	1%	porn
分类器 B	5%	obj

$$\text{原版: } \bar{E}_{\text{ndex}} = \frac{1}{m_{\text{dev}}} \sum_{i=1}^{m_{\text{dev}}} I\{\hat{y}^{(i)} \neq y^{(i)}\}$$

$$\downarrow \text{改进: } \bar{E}_{\text{ndex}} = \frac{1}{\sum w^{(i)}} \sum_{i=1}^{m_{\text{dev}}} w^{(i)} I\{\hat{y}^{(i)} \neq y^{(i)}\} \quad \text{其中 } w^{(i)} = \begin{cases} 1, & x^{(i)} \text{ not porn} \\ 100, & x^{(i)} \text{ is porn.} \end{cases}$$

与人类智能做比较 (Bias/Variance 分析)

因为 MN 由人类创造，以人类水平为参照，可以更容易地以人类已有知识进行改进。

何况在视觉识别等领域，人类水平的误差几乎可以作为 及时止损 (误差上限)

e.g.

Human Err	Avoidable Bias		Variance		Dev Err
	Human Err	Diff	Training Err	Diff	
Case A	1%	7%	8%	2%	10%
Case B	7.5%	0.5%	8%	2%	10%
Case C	9%	-1%	8%	2%	10%

→ 严重减少 Bias

→ 严重减少 Variance.

→ 人类参照失效，难以有效提升。但是至高数据质量是关键的。

误差分析 (用于减小 Avoidable bias)

观察 Dev set 中出错的部分，对出错类型进行分类，占比大的错误更有解决价值。

- 错误 label.

如果不是系统性错误 (所有目标标记为猫)，可以不用管，MN 对随机错误比较 robust。(Cost func = $\frac{1}{m} \sum \dots$ 中 m 很大)

↓ 1. Tips: 善于开项目，先设定目标 → 快速构建模型 → Bias/Variance 分析 → Error 分析

· 不同分布上的训练与测试

只有少量的平凡数据，不足以训练出一个满意的 MN。

网络高清猫有很多，虽然与我们目标不同 (对于机器叫物的识别)

Cat app example ↗ care about this 但可以尽可能 Data from webpages ↗ care about this 但可以尽可能 Data from mobile app ↗ care about this 但可以尽可能

→ 200,000 → 10,000

X Plan A: 200,000 + 10000 = 210,000 \Rightarrow shuffle 后训练 (统一分布)

Dev/Test 评估不完全
且并不符合 All mixed

✓ Plan B: Dev/Test 作用目标与评估标准采用 App 样本，Training 则大量运用 Web 样本作为辅助。
且训练可能成功。

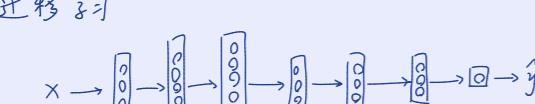
一更复杂的误差分析与数据均衡分析。

Plan C: Training: 200,000 Training-Dev: 3000 Dev-Test: 2500 Total: 200,000 mixed = 200,000 Web + 1000 App

解决方法

- 人工合成 Training set (通过数据 augmentation) (e.g. Use 高清图 blury)
清晰入声 + 噪声 → 平滑声

迁移学习



做法:

由于有些领域缺少样本作为 Training set，使用类似样本进行 pre-training。
之后将最后一层或几层替换，并用真实样本继续 fine-tuning。

原理:

由于同样是视觉识别，端到端训练出前几层，比较基本的底层特征 (固有)，这些线面同样适用于描述 X-ray 图。

小 Tips: · Test/Dev 要朝同一分布，否则意味着机器离谱了。

· Test set 只要足以在 Dev/Testing 系统中给出高方差，不超过 30%~20%。Big data + 1000 太多。

· Dev set 要足以检测不同算法、模型的差异用于评估。Big data + 1% 太少。

e.g. 1

P: 所有被判为“是猫”的结果中，判对了多少

Trade-off

R: 所有的 Test set 中有多少是猫被判断为“是猫”

Classifier	Precision	Recall	F1 Score
A	95%	90%	92.4%
B	98%	85%	91.0%

e.g. 2. 使用均值作为评估标准。

Optimizer	Accuracy	Running time
A	90%	80ms
B	92%	95ms
C	95%	1500ms

设置 Dev/Test set 评估指标

需求的改变可以通过对指标以及 cost func 的 Tuning 来满足。

e.g. 改变惩罚规则：

由于不希望由分类器向客户推送 porn，所以要在 cost 中对 porn 惩罚。

	error	错误成分
分类器 A	1%	porn
分类器 B	5%	obj

$$\text{原版: } \bar{E}_{\text{ndex}} = \frac{1}{m_{\text{dev}}} \sum_{i=1}^{m_{\text{dev}}} I\{\hat{y}^{(i)} \neq y^{(i)}\}$$

$$\downarrow \text{改进: } \bar{E}_{\text{ndex}} = \frac{1}{\sum w^{(i)}} \sum_{i=1}^{m_{\text{dev}}} w^{(i)} I\{\hat{y}^{(i)} \neq y^{(i)}\} \quad \text{其中 } w^{(i)} = \begin{cases} 1, & x^{(i)} \text{ not porn} \\ 100, & x^{(i)} \text{ is porn.} \end{cases}$$

与人类智能做比较 (Bias/Variance 分析)

因为 MN 由人类创造，以人类水平为参照，可以更容易地以人类已有知识进行改进。

何况在视觉识别等领域，人类水平的误差几乎可以作为 及时止损 (误差上限)

e.g.

Human Err	Avoidable Bias		Variance		Dev Err
	Human Err	Diff	Training Err	Diff	
Case A	1%	7%	8%	2%	10%
Case B	7.5%	0.5%	8%	2%	10%
Case C	9%	-1%	8%	2%	10%

→ 严重减少 Bias

→ 严重减少 Variance.

→ 人类参照失效，难以有效提升。但是至高数据质量是关键的。

误差分析 (用于减小 Avoidable bias)

- 错误 label.

如果不是系统性错误 (所有目标标记为猫)，可以不用管，MN 对随机错误比较 robust。(Cost func = $\frac{1}{m} \sum \dots$ 中 m 很大)

↓ 1. Tips: 善于开项目，先设定目标 → 快速构建模型 → Bias/Variance 分析 → Error 分析

· 不同分布上的训练与测试

只有少量的平凡数据，不足以训练出一个满意的 MN。

网络高清猫有很多，虽然与我们目标不同 (对于机器叫物的识别)

Cat app example ↗ care about this 但可以尽可能 Data from webpages ↗ care about this 但可以尽可能 Data from mobile app ↗ care about this 但可以尽可能

→ 200,000 → 10,000

X Plan A: 200,000 + 10000 = 210,000 \Rightarrow shuffle 后训练 (统一分布)

Dev/Test 评估不完全
且并不符合 All mixed

✓ Plan B: Dev/Test 作用目标与评估标准采用 App 样本，Training 则大量运用 Web 样本作为辅助。
且训练可能成功。

卷积神经网络

浅层卷积网络

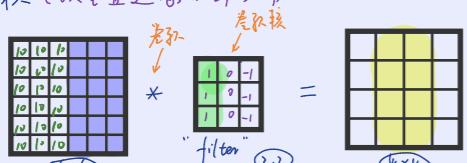
Intro

$$64 \times 64 \times 3 = 128K \quad (12K)$$

$$1000 \times 1000 \times 3 = 3,000,000 \quad (3M) \quad W: 3.84M$$

前面与运算无法接壤的点，更过分地容易发生 padding

卷积 (以垂直边缘识别为例) filter element wise 乘，and sum



filter 不一定局限于垂直、水平边缘检测，
若能与图片自适应(旋转) 岂不美滋滋

- Padding

问题：每次卷积层都缩水带来一些不便 (多次卷积后成为 $1 \times 1 \times 1$)

解决：高维过程中，边缘信息与中心信息权重不一致，有偏差。

"Valid": $n \times n \times f \times f = (n-f+1) \times (n-f+1)$

"Same": $(n+2p-f)^2 \times f^2 = n^2$ 解得 $p = \frac{f-1}{2}$

卷积层

参数：filter size: $f^{[l]}$

padding: $p^{[l]}$

stride: $s^{[l]}$

number of filters: $n_c^{[l]}$

大小: Input: $n_h^{[l-1]} \times n_w^{[l-1]} \times n_c^{[l-1]}$

(activation) output: $n_h^{[l]} \times n_w^{[l]} \times n_c^{[l]}$ 其中

filter: $f^{[l-1]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$

Weight: $f^{[l-1]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$

bias: $1 \times 1 \times 1 \times n_c^{[l]}$

池化层 (pooling layer)

- 方式:

Max pooling \rightarrow
Average pooling \rightarrow
不加 padding



如果有 n_c 个 channel
则会分 n_c 个 pooling

- 大小:

filter size: f
Input: $n_h \times n_w \times n_c$
stride: s
Output: $\lfloor \frac{n_h-f+1}{s} \rfloor \times \lfloor \frac{n_w-f+1}{s} \rfloor \times n_c$

- 含义:

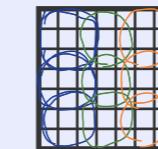
提取特征，缩小范围，减小体积

Mathematical Convolution: filter matrix 先上右翻转再 element-wise 相乘求和。

Machine Learning: 不翻转，实为数字中的 cross-correlation.

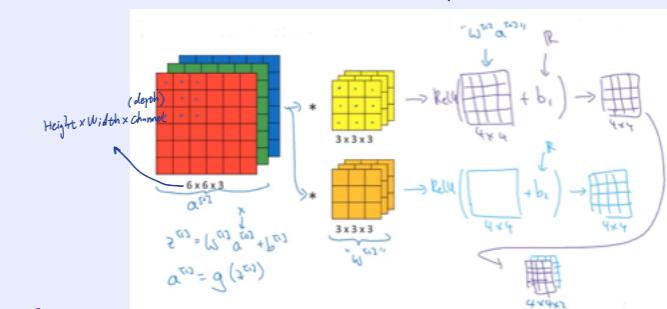
- Stride

$$\text{stride} = 2 \quad f^2 \rightarrow \text{flatt} \left[\frac{n+2p-f}{s} + 1 \right]$$

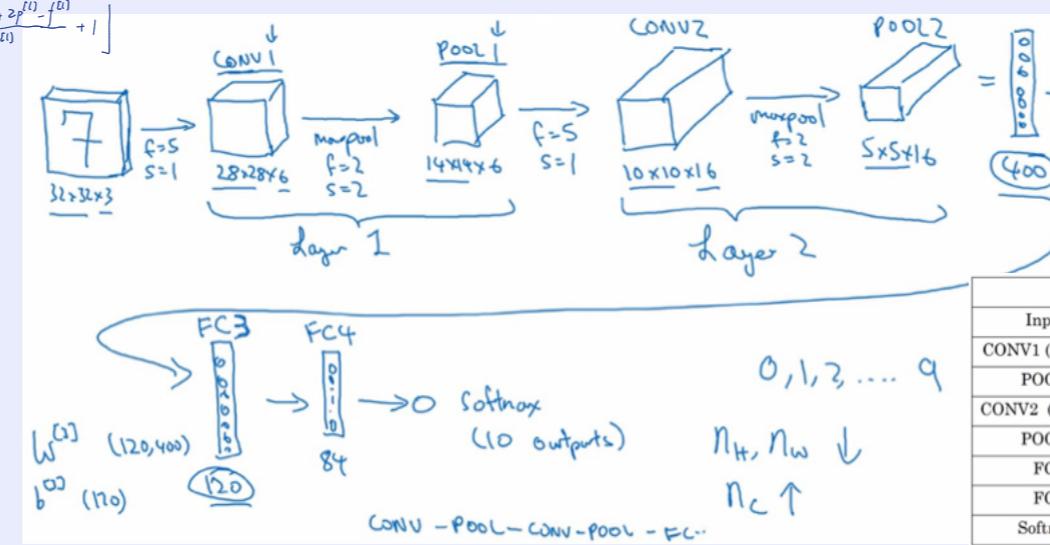


$$* = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

- Multi-Channel & Multi-filter



E.g. RGB 数字图 (32x32 像素), ① Conv NN 识别 0~9.



	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	3,072	0
CONV1 ($f=5, s=1$)	(28,28,8)	6,272	$5 \times 5 \times 8 + 8$
POOL1	(14,14,8)	1,568	0
CONV2 ($f=5, s=1$)	(10,10,16)	1,600	416
POOL2	(5,5,16)	400	0
FC3	(120,1)	120	48,001
FC4	(84,1)	84	10,081
Softmax	(10,1)	10	841

原理：

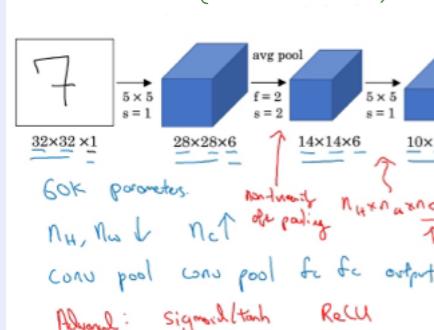
① 参数共享 (parameter sharing): 同一个 filter 可以被用于探测不同位置区域

② 连接稀疏 (sparsity of connection): 每层中，output 只连接 input 中一小部分。

深层卷积网络

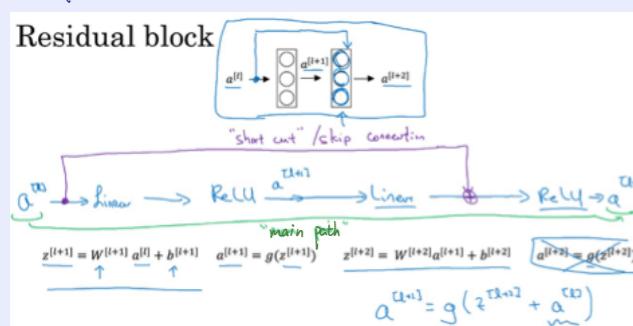
一、经典 CNN

LeNet - 5 (LeCun et.al 1998)



残差网络 (Residual Net) (Kaiming He et.al. 2015)

- 简述



Inception Network.

- 1x1 卷积

单 channel \rightarrow 1x1 卷积相当于 element-wise 相乘

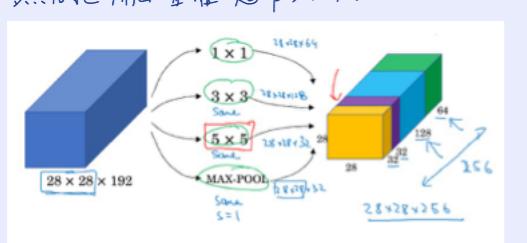
多 channel \rightarrow 1x1 卷积相当于对每个通道进行一个 linear 非线性操作

如果说 pooling 是为了在 $n_h \times n_w$ 维度而递进信息减少，则 1x1 卷积则可以在 n_c 维度进行信息的融合，体积并不线性化。

当然不知道怎样让 filter size 更好，那就一次性的全试了。

- 3x3 卷积

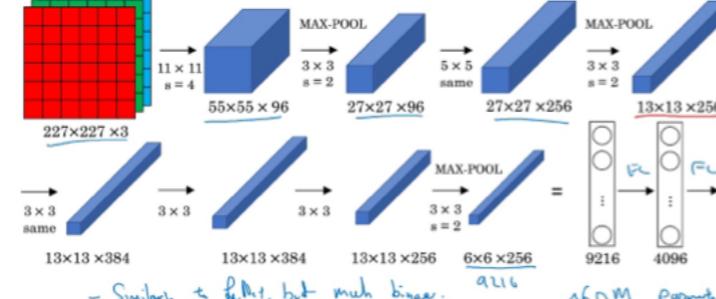
问什么大小的 filter，一律用 same padding 保证 Input/Output 是 $n_h \times n_w \times n_c$ 。然后把输出量在一起作为下一层。



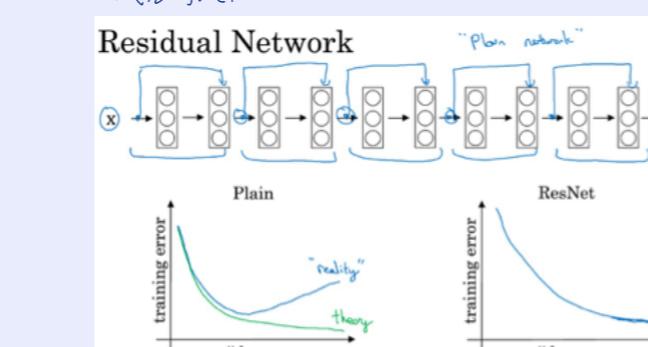
[Szegedy et al. 2014. Going deeper with convolutions]

Inception 名称来源于电影《Inception》: we need to go deeper

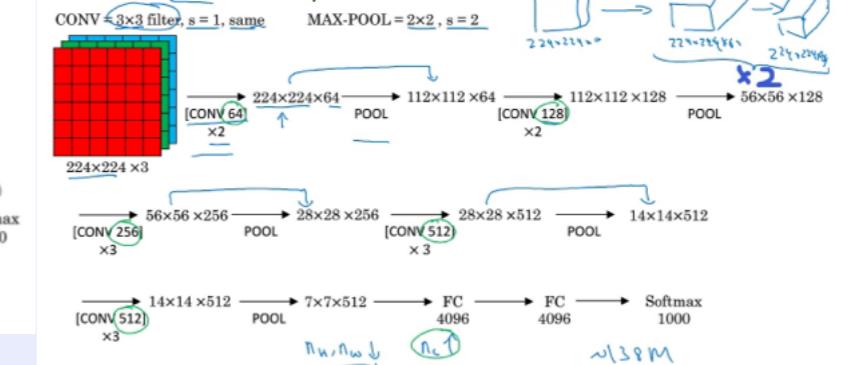
AlexNet (Krizhevsky et.al. 2012)



- 该类同 AlexNet。



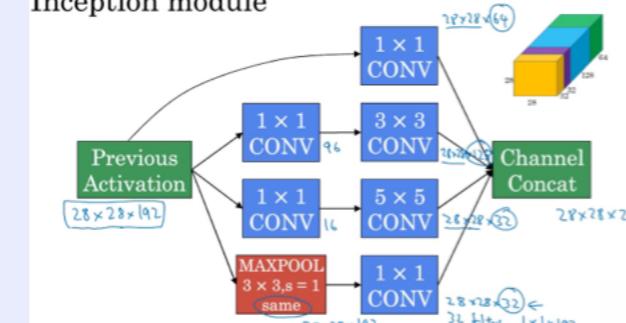
VGG - 16 (Simonyan & Zisserman 2015)



- 该类同 VGG-16。

由于多层以后梯度弥散十分严重，使得网络层数变深反而不利于模型学习。
ResNet 使 a^{l+1} 可以传播到下一层，不仅可以使网络深度归一，并使得网络的深度可以进一步提升。

- Inception Module



- Inception Network

