Deep Learning (taught by Andrew Ng) – Notes by Xiaofan Wang

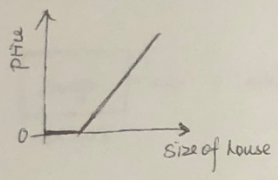
Name: Xiaofan Wang

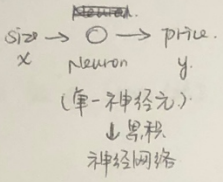
Neural Networks and Deep Learning.

Ch1. Introduction to Deep Learning

一. Neural Network.

1. Housing Price Prediction.





(单一神经元)

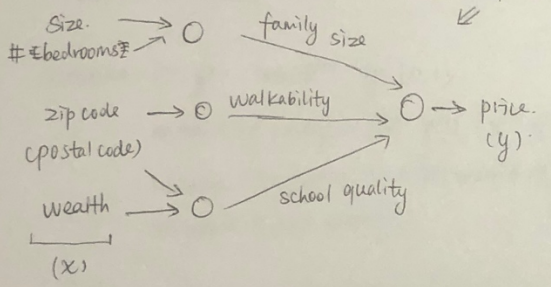
↓累积

神经网络

Relu. (Rectified Linear Unite. 线性整流函数)

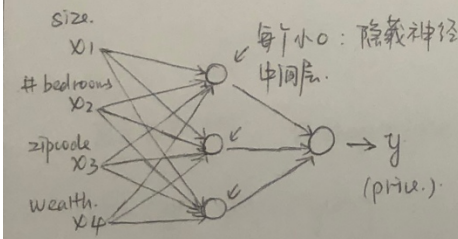
$\max(0, y)$

更大的神经网络



Neural Network: 只需给定训练集中的大量例子的 $x \to y$.

而所有中间部分，NN会自己弄清.（只需给足够多的训练例 $x \to y$，NN就可以很好地拟合一个函数来建立 x, y 之间的映射关系）



输入参数：x_1, x_2, x_3, x_4.

NN的作用是：预测 price "y"

= ~~Supp~~ Supervised Learning with Neural Network.

1.

	Input (x)	Output (y)	Application
①	Home features	price.	Real Estate.
②	Ad, User Info.	Click on Ad? 0/1.	Online Advertising.
③	Image.	Object (1---1000)	Photo tagging.
④	Audio	Text Transcript.	Speech Recognition.
⑤	English.	Chinese.	Machine Translation.
⑥	Image, Radar	position of cars	Autonomous Driving.

在特定问题下，建立 $x \to y$ 映射关系，通过监督学习拟合数据，成为某个复杂系统的一部分.

①②: Standard NN.

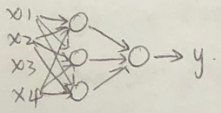
③: CNN

④⑤: RNN.

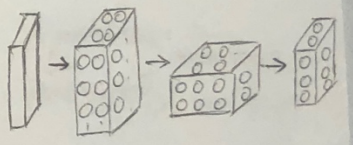
⑥: Customized, hybrid NN architecture.

2. NN Examples:

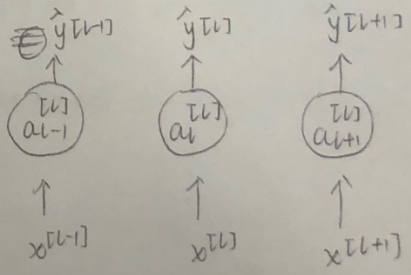
Standard NN.



Convolutional NN.



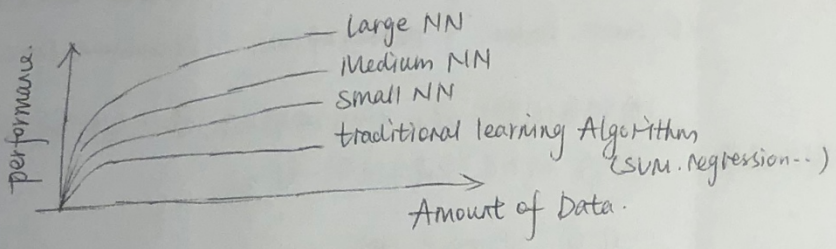
Recurrent NN. (循环神经网络)



3.
- Structured Data：每个特征，都有清晰的含义
- Unstructured Data：
(Image. Text…, Audio)

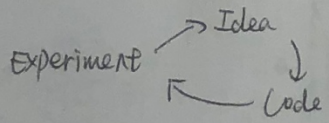
三. Why is Deep Learning taking off?

1. Scale drives deep learning progress



High Performance：足够大的网络，大量数据

- Data
- Computation：Sigmond → ReLu.
(↓) (↓)
- Algorithms



Ch2 Neural Network Basics

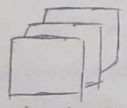
一、Logistic Regression as a Neural Network

1. Binary Classification

Neural Network 不使用 for 循环遍历全部训练样本，使用"前向传播"、"反向传播"

Image → 1 (cat) vs 0 (Non Cat)

64×64



64×64×3 (图像的矩阵包含的元素点数)

$\vec{x} = \begin{bmatrix} \\ \\ \\ \end{bmatrix}$

$n_x = 64 \times 64 \times 3 = 12288$

Notation: (x, y). $x \in \mathbb{R}^{n_x}$, $y \in \{0, 1\}$

m training example: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}) \cdots (x^{(m)}, y^{(m)})\}$

M train: emphasize the total number of sample.

M test = # test example.

$$\vec{X} = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} \quad \begin{matrix} | \\ n_x \\ | \end{matrix}$$

m

$\vec{X} \in \mathbb{R}^{n_x \times m}$ $\vec{X}.\text{shape} = (n_x, m)$

$\vec{Y} = [y^{(1)}, y^{(2)} \cdots y^{(m)}]$

$\vec{Y} \in \mathbb{R}^{1 \times m}$

$\vec{Y}.\text{shape} = (1, m)$

2. Logistic Regression

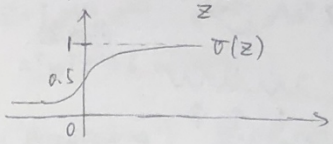
Given x (Image), $\Rightarrow \hat{y} = P(y=1|x)$ $(0,1)$.

输入图片 x，$\hat{y}$ 告诉你这是一张猫图的概率.

$\vec{x} \in \mathbb{R}^{n_x}$

Parameters: $\vec{w} \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$.

Output: $\hat{y} = \underbrace{\vec{w}^T \vec{x} + b}_{z}$



$\sigma(z) = \frac{1}{1+e^{-z}}$

If z large, $\sigma(z) \approx 1$

$x_0 = 1$, $\vec{x} \in \mathbb{R}^{n_x+1}$, $\hat{y} = \sigma(\vec{\theta}^T \cdot \vec{x})$

$$\vec{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_{n_x} \end{bmatrix} \begin{matrix} \} b \\ \} \vec{w} \end{matrix}$$

3. Logistic Regression Cost Function. (Minimize it)

$\hat{y} = \sigma(w^T \cdot \vec{x} + b)$, where $\sigma(z) = \frac{1}{1+e^{-z}}$

Given $\{(x^{(1)}, y^{(1)}), \cdots (x^{(m)}, y^{(m)})\}$, want $\hat{y}^{(i)} \approx y^{(i)}$

$z^{(i)} = w^T \cdot \vec{x}^{(i)} + b$

Loss (error) Function: $\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$

$\mathcal{L}(\hat{y}, y) = -(y \log \hat{y} + (1-y) \log(1-\hat{y}))$

If $y=1 \Rightarrow \mathcal{L}(\hat{y}, y) = -\log \hat{y}$, want $\log \hat{y}$ large, want $\hat{y}$ large.

If $y=0 \Rightarrow \mathcal{L}(\hat{y}, y) = -\log(1-\hat{y})$, want $\log(1-\hat{y})$ large, want $\hat{y}$ small.

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

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

寻找 w b.

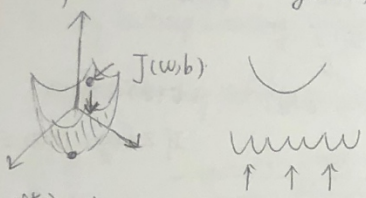
来缩小 J 的整体成本.

4 Gradient Descent. (学习训练 $\Rightarrow w, b$)

Recap: $\hat{y} = \sigma(w^T \cdot x + b)$, $\sigma(z) = \frac{1}{1+e^{-z}}$

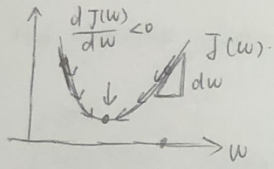
$J(w,b) = \frac{1}{m} \cdot \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) = \frac{-1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \cdot \log(1-\hat{y}^{(i)}) \right]$

want to find w, b that minimize $J(w,b)$.



凸函数.

梯度下降收敛到全局最优值.



Repeat { $w := w - \alpha \cdot \frac{dJ(w)}{dw}$; dw }

↓ Learning Rate.

~~$w = w - \alpha$~~ $w = w - \alpha \, dw$

$J(w,b)$: $\begin{cases} w = w - \alpha \cdot \frac{dJ(w,b)}{dw} \\ b = b - \alpha \cdot \frac{dJ(w,b)}{db} \end{cases}$

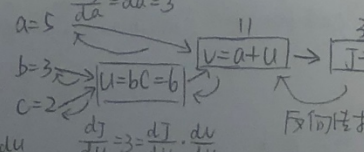
对比 $\begin{cases} \frac{dJ(w,b)}{dw}: \text{当函数只有一个变量} \\ \frac{\partial J(w,b)}{\partial w}: \text{偏导数. 关注其中一个变量在对应点处的斜率.} \end{cases}$

5. Computation Graph.

$\begin{cases} \text{正向传播: 前向计算. NN Output.} \\ \text{反向传播: 计算梯度或微分} \end{cases}$

$J(a,b,c) = 3(a+bc)$

$\begin{cases} u = bc \\ v = a+u \\ J = 3v \end{cases}$



$\frac{dJ}{dc} = \frac{dJ}{du} \frac{du}{dc} = 3 \times 3 = 9$

$\frac{dJ}{db} = \frac{dJ}{du} \frac{du}{db} = 3 \times 2 = 6$.

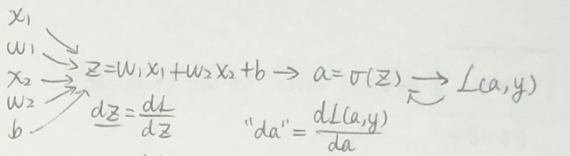
6. Logistic Regression Gradient Descent.

① Logistic Regression Recap (单一样本).

$z = w^T x + b$

$\hat{y} = a = \sigma(z)$

$L(a,y) = -(y \log(a) + (1-y) \log(1-a))$.



$dz = \frac{dL}{dz}$ "da" $= \frac{dL(a,y)}{da}$

$= \frac{dL(a,y)}{dz}$ $= -\frac{y}{a} + \frac{1-y}{1-a}$.

$= a - y$.

$= \frac{dL}{da} \left(\frac{da}{dz} \right) \rightarrow a(1-a)$

$\begin{cases} w_1 = w_1 - \alpha \, dw_1 \\ w_2 = w_2 - \alpha \, dw_2 \\ b = b - \alpha \cdot db \end{cases}$

② On m examples.

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

$a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(w^T x^{(i)} + b)$

$\frac{\partial J(w,b)}{\partial w_1} = \frac{1}{m} \cdot \sum_{i=1}^{m} \frac{\partial L(a^{(i)}, y^{(i)})}{\partial w_1}$

$dw_1^{(i)} = (x^{(i)}, y^{(i)})$

$\begin{cases} (x^{(i)}, y^{(i)}) \\ dw_1^{(i)} \\ dw_2^{(i)} \\ db^{(i)} \end{cases}$

$J = 0. \; dw_1 = 0. \; dw_2 = 0. \; db = 0.$

For i=1 to m.

$z^{(i)} = w^T \cdot x^{(i)} + b.$

$a^{(i)} = \sigma(z^{(i)})$

$J \mathrel{+}= -[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log(1-a^{(i)})]$

(一次下降的迭代) $dz^{(i)} = a^{(i)} - y^{(i)}$

向量化可避免循环 $\begin{cases} dw_1 \mathrel{+}= x_1^{(i)} \cdot dz^{(i)} \\ dw_2 \mathrel{+}= x_2^{(i)} \cdot dz^{(i)} \\ db \mathrel{+}= dz^{(i)} \end{cases}$

$n = 2$ ↕ $dw_3 \; \vdots \; dw_n$

$J /= m$

$dw_1 /= m. \; dw_2 /= m. \; db /= m$

$[dw = np.zeros((n-x, 1))]$

$\begin{cases} dw_1 = \frac{\partial J}{\partial w_1} \\ w_1 = w_1 - \alpha dw_1 \\ w_2 = w_2 - \alpha dw_2 \\ b = b - \alpha db \end{cases}$

二. Vectorization. (使代码达到高效)

1. Vectorization.

$z = w^T \cdot x + b$ $\quad \vec{w} = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix}$ $\quad \vec{x} = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix}$

$\vec{w} \in R^{n_x}$ $\quad \vec{x} \in R^{n_x}$

Non-vectorized:

```
z = 0
for i in range(n_x):
    z += w[i] * x[i]
z += b
```

Vectorized:

```
z = np.dot(w, x) + b
```

($\vec{w}^T \cdot x$)

GPU / CPU >> SIMD. - single instruction multiple data.
(通过内置函数和避免显示for循环, 实现向量化. 提高运行速度)

2. More Examples.

① NN programming Guideline: whenever possible, avoid explicit for-loops.

Non-vectorized:

$u = A \cdot v$

$u_i = \sum_j A_{ij} \cdot v_j$

```
u = np.zeros(n, 1)
for i ---
    for j ---
        u[i] += A[i][j] * v[j]
```

Vectorized:

```
u = np.dot(A, v)
```

(消除两层 for-loops. faster speed for operation)

② Vectors and matrix valued functions.

Need to apply the exponential operation on every element of a matrix/vector.

$\vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$ $\quad \vec{u} = \begin{bmatrix} e^{v_1} \\ e^{v_2} \\ \vdots \\ e^{v_n} \end{bmatrix}$

```
u = np.zeros(n, 1)
for i in range(n):
    u[i] = math.exp(v[i])
```

Numpy 内置函数: import numpy as np

$u = np.exp(v)$.

(np.log(v); np.abs(v); np.maximum(v, 0); v**2; 1/v)

3. Vectorizing Logistic Regression.

$z^{(1)} = w^T x^{(1)} + b$ $\quad z^{(2)} = w^T x^{(2)} + b$ $\quad z^{(3)} = w^T x^{(3)} + b$

$a^{(1)} = \sigma(z^{(1)})$ $\quad a^{(2)} = \sigma(z^{(2)})$ $\quad a^{(3)} = \sigma(z^{(3)})$

$\vec{x} = \begin{bmatrix} \vdots & \vdots & & \vdots \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ \vdots & \vdots & & \vdots \end{bmatrix}$ $\quad (n_x, m)$ $\quad R^{n_x \cdot m}$

$Z = [z^{(1)}, z^{(2)} \cdots z^{(m)}] = w^T \cdot \vec{x} + \underbrace{[b, b \cdots b]}_{1 \times m}$

$= [w^T x^{(1)} + b, w^T x^{(2)} + b \cdots]$

("Broadcasting") $\quad Z = np.dot(w^T, \vec{x}) + \underbrace{b}_{(1,1)}$

$A = [a^{(1)} \; a^{(2)} \cdots a^{(m)}] = \sigma(Z)$

4. Vectorizing LR's Gradient Computation.

$\vec{A} = [a^{(1)} \cdots a^{(m)}]$

$\vec{Y} = [y^{(1)} \cdots y^{(m)}]$

$dz = \vec{A} - \vec{Y} = [a^{(1)} - y^{(1)} \cdots a^{(m)} - y^{(m)}]$

$dw = 0$

$dw \mathrel{+}= x^{(1)} dz^{(1)}$

$\vdots$

$dw \mathrel{/}= m$

$db = 0$

$db \mathrel{+}= dz^{(1)}$

$\vdots$

$db \mathrel{/}= m$

$db = \frac{1}{m} \cdot \sum_{i=1}^{m} dz^{(i)}$

$= \frac{1}{m} \cdot np.sum(dz)$

$dw = \frac{1}{m} \cdot \vec{x} \cdot dz^T$

$= \frac{1}{m} \cdot \begin{bmatrix} x^{(1)} & \cdots & x^{(m)} \\ \vdots & & \vdots \end{bmatrix} \begin{bmatrix} dz^{(1)} \\ \vdots \end{bmatrix}$

$= \frac{1}{m} [x^{(1)} dz^{(1)} \cdots]$

$n \times 1$

LR's Gradient Computation 一次迭代:

$z = w^T \cdot x + b = np.dot(w^T, x) + b$

$\vec{A} = \sigma(z)$

$dz = \vec{A} - \vec{Y}$

$dw = \frac{1}{m} \cdot \vec{x} \cdot dz^T$

$db = \frac{1}{m} \cdot np.sum(dz)$

$w = w - \alpha \cdot dw$

$b = b - \alpha \cdot db$

5. Broad Casting (faster, fewer codes)

Example:

Calories from Carbs, Proteins, Fats in 100g of different foods:

	Apples	Beef	Eggs	Potatoes
Carb	56.0	0.0	4.4	68.0
Protein	1.2	104.0	52.0	8.0
Fat	1.8	135.0	99.0	0.9

$= \vec{A}(3,4)$

Python. Broad Casting:

cal = A.sum(axis=0) · #计算每列和.

percentage = 100 * A / (cal.reshape(1,4)) ·

(3,4)/(1,4) 当不确定矩阵维数. 可调用 reshape.
以保证是正确的行/列向量.
reshape: O(1). 消耗成本低.

Other Example:

$$\begin{bmatrix}1\\2\\3\end{bmatrix} + \begin{bmatrix}100\\100\\100\end{bmatrix} = \begin{bmatrix}101\\102\\103\end{bmatrix}$$

$$\begin{bmatrix}1&2&3\\4&5&6\end{bmatrix} + [100\ 200\ 300] = \begin{bmatrix}101&202&303\\104&205&306\end{bmatrix}$$

General principle: (m,n) +-*/ (1,n) → (m,n)
Matrix (m,1) → (m,n).

(m,1) + R.

[1,2,3] + 100 = [101, 102, 103]

$\begin{bmatrix}1\\2\\3\end{bmatrix} + 100 = \begin{bmatrix}101\\102\\103\end{bmatrix}$

6. Python / Numpy Vectors (eliminate bug)

a = np.random.randn(5). [a.shape = (5,)].

a = np.random.randn(5,1) ⇒ a.shape = (5,1). col vector

a = np.random.randn(1,5) ⇒ a.shape = (1,5) row vector

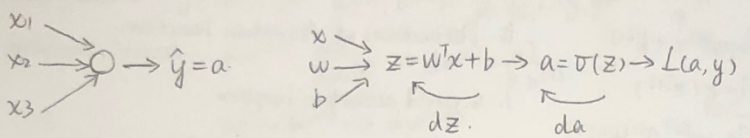
assert (a.shape == (5,1)) ⇒ a = a.reshape((5,1)).

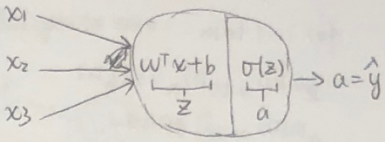
用assert来检查矩阵和数组的维度.

Ch3 Shallow Neural Network

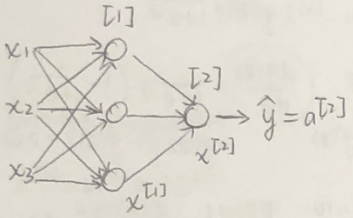
二. Computing NN's output

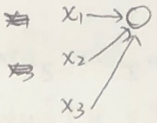
一. Neural Network





$z = w^T x + b$ $a = \sigma(z)$



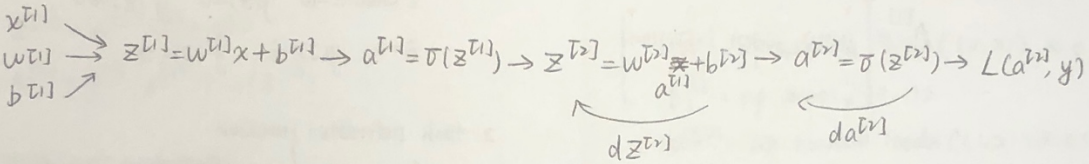


隐藏层的第一个节点：

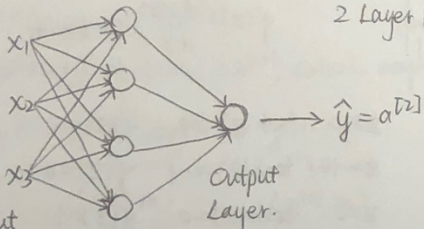
$z^{[1]} = w_1^{[1]T} x + b_1^{[1]}$

$a_1^{[1]} = \sigma(z_1^{[1]})$

$a_j^{[1]}$: node in layer.



二. Neural Network Representation



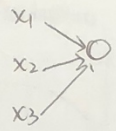
2 Layer NN.

$a^{[2]}$. $\begin{cases} w^T = w^{[2]} \\ b = b^{[2]} \end{cases}$

$a^{[0]} = x$

$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix}$

parameters in Hidden Layer. $\begin{cases} w^{[1]} : (4, 3) \\ b^{[1]} : (4, 1) \end{cases}$



Hidden Layer. 2nd node:

$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}$, $a_2^{[1]} = \sigma(z_2^{[1]})$

3rd node; 4th node:

$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}$, $a_3^{[1]} = \sigma(z_3^{[1]})$

$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}$, $a_4^{[1]} = \sigma(z_4^{[1]})$

$$z^{[1]} = \begin{bmatrix} -- w_1^{[1]T} -- \\ -- w_2^{[1]T} -- \\ -- w_3^{[1]T} -- \\ -- w_4^{[1]T} -- \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix} = \sigma(z^{[1]})$$

III. Vectorizing across multiple examples.

for i = 1 to m:

$z^{[1](i)} = W^{[1]} x^{(i)} + b^{[1]}$

$a^{[1](i)} = \sigma(z^{[1](i)})$

$z^{[2](i)} = W^{[2]} a^{[1](i)} + b^{[2]}$

$a^{[2](i)} = \sigma(z^{[2](i)})$

Given X:

$x = A^{[0]} = a^{[0](i)}$

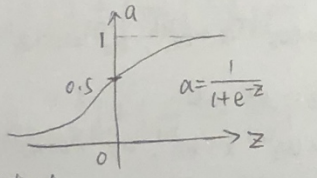
$$\begin{cases} z^{[1]} = W^{[1]} x + b^{[1]} \\ A^{[1]} = \sigma(z^{[1]}) \\ z^{[2]} = W^{[2]} A^{[1]} + b^{[2]} \\ A^{[2]} = \sigma(z^{[2]}) \end{cases}$$

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} \quad (n_x, m)$$

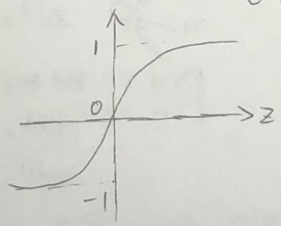
$$Z^{[1]} = \begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ | & | & & | \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} | & | & & | \\ A^{[1](1)} & A^{[1](2)} & \cdots & A^{[1](m)} \\ | & | & & | \end{bmatrix}$$

IV. Activation Functions.



Output ~~Input~~ Layer: sigmoid function

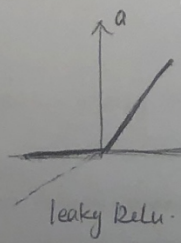


Hidden Layer: tanh function.

(tanh > sigmoid).

不常用.

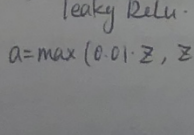
If z is very large/small, k = a ≈ 0. decrease the speed of gradient descent.



(ReLU > tanh > sigmoid).

ReLU: $a = \max(0, z)$

(Rectified Linear Unit 线性修正).



ReLU, leaky ReLU 优点:

对大部分 z 空间，激活函数的导数/斜率 ≠ 0

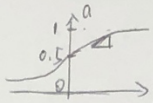
(faster learning rate).

V. Non-linear Activation Functions.

deep NN: many hidden layers

VI. Derivatives of Activation Functions.

1. Sigmoid activation function.



$g(z) = \frac{1}{1+e^{-z}}$

$\frac{dg(z)}{dz} = \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right)$

"g'(z)"

$= g(z) \cdot (1 - g(z)) = a(1-a)$.

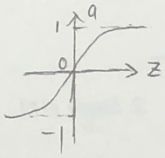
Check:

z large = 10. $g(z) \approx 1$. $g'(z) \approx 0$

z small = -10 $g(z) \approx 0$. $g'(z) \approx 0$.

$z = 0$. $g(z) = \frac{1}{2}$. $g'(z) = \frac{1}{4}$.

2. tanh activation function.



$g(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

$g'(z) = \frac{dg(z)}{dz} = 1 - (\tanh(z))^2 = 1 - a^2$

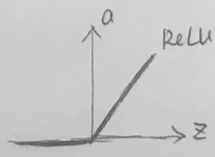
Check:

$z = 10$. $\tanh(z) \approx 1$. $g'(z) \approx 0$

$z = -10$ $\tanh(z) \approx -1$ $g'(z) \approx 0$

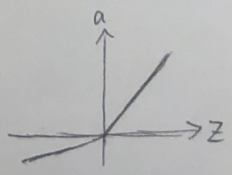
$z = 0$. $\tanh(z) = 0$. $g'(z) = 1$.

3. ReLU & leaky ReLU.



$g(z) = \max(0, z)$

$g'(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$



$g(z) = \max(0.01z, z)$

$g'(z) = \begin{cases} 0.01 & \text{if } z < 0 \\ 1 & \text{if } z > 0. \end{cases}$

11. Gradient Descent for Neural Networks

parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}$

cost Function: $J(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}, y)$

Gradient Descent: $dW^{[1]} = \frac{dJ}{dW^{[1]}}$, $db^{[1]} = \frac{\partial J}{\partial b^{[1]}}$ ...

$W^{[1]} = W^{[1]} - \alpha \cdot dW^{[1]}$

$b^{[1]} = b^{[1]} - \alpha \cdot db^{[1]}$

......

Forward Propagation:

$Z^{[1]} = W^{[1]}x + b^{[1]}$

$A^{[1]} = g^{[1]}(Z^{[1]})$

$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$

$A^{[2]} = g^{[2]}(Z^{[2]}) = \sigma(Z^{[2]})$

Backward propagation:

$dZ^{[2]} = A^{[2]} - Y$. $Y = [y^{(1)}, y^{(2)} ... y^{(m)}]$.

$dW^{[2]} = \frac{1}{m} \cdot dZ^{[2]} A^{[1]T}$ $(n^{[2]}, 1)$

$db^{[2]} = \frac{1}{m} \cdot$ np.sum($dZ^{[2]}$, axis=1, keepdims=True)

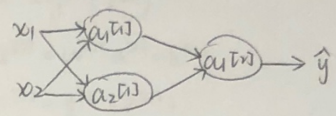
$dZ^{[1]} = \underbrace{W^{[2]T} \cdot dZ^{[2]}}_{(n^{[1]}, m)} * \underbrace{g'^{[1]}(Z^{[1]})}_{(n^{[1]}, m)}$

$dW^{[1]} = \frac{1}{m} \cdot dZ^{[1]} x^T$

$db^{[1]} = \frac{1}{m} \cdot$ np.sum($dZ^{[1]}$, axis=1, keepdims=True)

12. Random Initialization.

If initialize weights to zero:



$W^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$. $b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $W^{[2]} = [0 \ \ 0]$

$a_1^{[1]} = a_2^{[1]}$. $dz_1^{[1]} = dz_2^{[1]}$

$dW = \begin{bmatrix} u & v \\ u & v \end{bmatrix}$ $W^{[1]} = W^{[1]} - \alpha dW$.

$W^{[1]}$ = np.random.randn((2,2)) * 0.01

$b^{[1]}$ = np.zeros((2,1))

$W^{[2]}$ = np.random.randn((1,2)) * 0.01

$b^{[2]} = 0$.

Summary of gradient descent:

$dZ^{[2]} = a^{[2]} - y$

$dW^{[2]} = dZ^{[2]} a^{[1]T}$

$db^{[2]} = dZ^{[2]}$

$dZ^{[1]} = W^{[2]T} \cdot dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$

$dW^{[1]} = dZ^{[1]} \cdot x^T$

$db^{[1]} = dZ^{[1]}$

$dZ^{[2]} = A^{[2]} - Y$

$dW^{[2]} = \frac{1}{m} \cdot dZ^{[2]} A^{[1]T}$

$db^{[2]} = \frac{1}{m} \cdot$ np.sum($dZ^{[2]}$, axis=1, keepdims=True)

$dZ^{[1]} = W^{[2]T} \cdot dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$

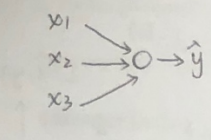
$dW^{[1]} = \frac{1}{m} \cdot dZ^{[1]} x^T$

$db^{[1]} = \frac{1}{m} \cdot$ np.sum($dZ^{[1]}$, axis=1, keepdims=True)

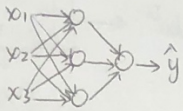
Tips: 如何初始化神经网络的权重. ⇒ 随机初始化参数对训练神经网络很重要.

Ch 4. Deep Neural Network.

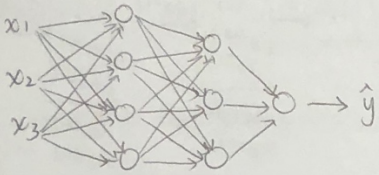
一. Deep L-layer Neural Network.



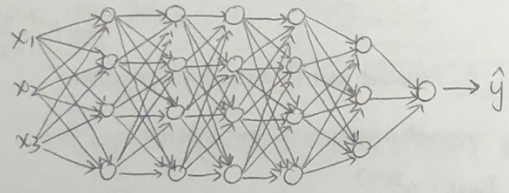
1 layer NN ("Shallow")
Logistic Regression.



1 Hidden Layer.
2 Layer NN.



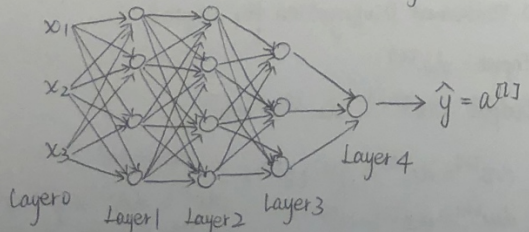
2 Hidden Layer.



5 Hidden Layer ("Deep").

Deep Neural Network Notation:

"4 Layer NN"



Layer0 Layer1 Layer2 Layer3 Layer4

$n^{[l]}$: # units in layer l ($n^{[1]}=4$. $n^{[2]}=4$. $n^{[3]}=3$. $n^{[4]}=1$). $n^{[0]}=n_x=3$

$a^{[l]}$: activation in layer l

二. Forward Propagation in a deep Network.

$$z^{[1]}=w^{[1]}x+b^{[1]} \quad (x=a^{[0]})$$
$$a^{[1]}=g^{[1]}(z^{[1]})$$
$$z^{[2]}=w^{[2]}a^{[1]}+b^{[2]}$$
$$a^{[2]}=g^{[2]}(z^{[2]})$$
$$\vdots$$
$$z^{[4]}=w^{[4]}a^{[3]}+b^{[4]} \Rightarrow \begin{cases} z^{[l]}=w^{[l]}a^{[l-1]}+b^{[l]} \\ a^{[l]}=g^{[l]}(z^{[l]}) \end{cases}$$
$$a^{[4]}=g^{[4]}(z^{[4]})$$

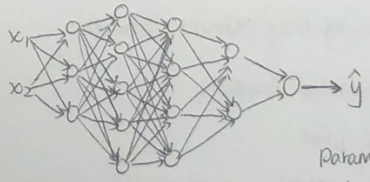
"a". "z". dimensions: $(n^{[l]}, 1)$

Vectorized:

$$Z^{[1]}=w^{[1]}A^{[0]}+b^{[1]} \quad (x=A^{[0]})$$
$$A^{[1]}=g^{[1]}(Z^{[1]})$$
$$Z^{[2]}=w^{[2]}A^{[1]}+b^{[2]}$$
$$A^{[2]}=g^{[2]}(Z^{[2]})$$
$$\hat{Y}=g(Z^{[4]})=A^{[4]}$$

Deep Network ≈ Network only with 1 Hidden Layer.
(只是多重复了n次)

三. Get Matrix Dimensions Right



$$z^{[1]}=w^{[1]}\,x+b^{[1]}$$
$$(3,1) \quad [3,2) \quad (2,1) \quad (3,1)$$
$$(n^{[1]},1) \quad (n^{[1]},n^{[0]}) \quad (n^{[0]},1) \quad (n^{[1]},1)$$

Parameters $w^{[l]}$, $b^{[l]}$

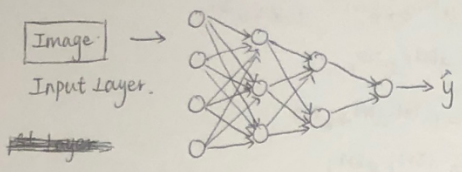
$$\begin{cases} w^{[l]}: (n^{[l]}, n^{[l-1]}) \\ b^{[l]}: (n^{[l]}, 1) \\ dw^{[l]}: (n^{[l]}, n^{[l-1]}) \\ db^{[l]}: (n^{[l]}, 1). \end{cases}$$

Vectorized Implementation:

$$Z^{[1]}=w^{[1]}\cdot\vec{X}+b^{[1]} \to (n^{[1]},1)$$
$$(n^{[1]},m) \quad (n^{[1]},n^{[0]}) \to (n^{[0]},m)$$

$$\begin{cases} z^{[l]}, a^{[l]}: (n^{[l]}, 1) \\ Z^{[l]}, A^{[l]}: (n^{[l]}, m) \\ dz^{[l]}, dA^{[l]}: (n^{[l]}, m) \\ l=0,\ A^{[0]}=\vec{X}: (n^{[0]}, m) \end{cases}$$

四. Deep Representation.



1st Layer: Feature Detector / Edge Detector.

2nd Layer: combine "Edges" together ⇒ form parts of Image

Different Neuron → Different part. (Complex).

3rd Layer: combine "Different parts" into a whole.

⇒ Test Different Types of Faces.

先找寻"简单事物"

然后构建"简单事物". 将其组合在一起以检测"更复杂事物"

⇓

Detect "Simple Objects" —combine→ "More complex Objects"

Circuit theory and deep learning:

There are functions you can compute with a "small" L-layer deep neural network that shallower networks require exponentially more hidden units to compute.

五. Building Blocks of Deep Neural Network.

Forward and Backward Functions:

Layer l: $w^{[l]}$, $b^{[l]}$

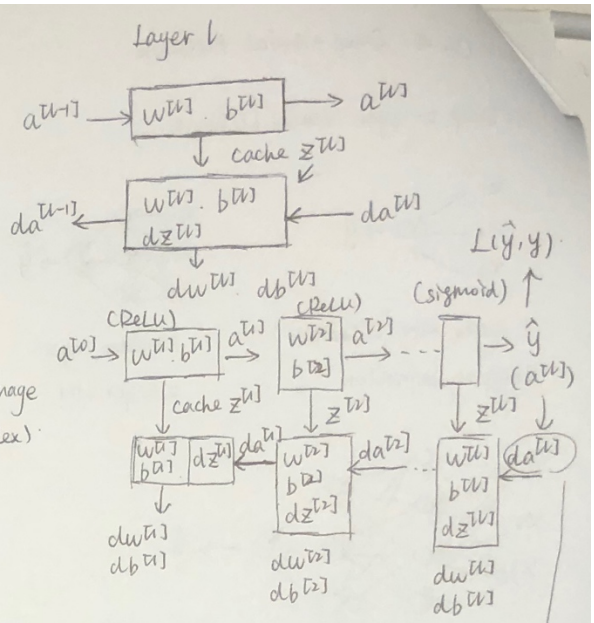
Forward: Input $a^{[l-1]}$. Output $a^{[l]}$, Cache $z^{[l]}$

$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$

$a^{[l]} = g^{[l]}(z^{[l]})$

Backward: Input $da^{[l]}$ Output $da^{[l-1]}$, $dw^{[l]}$, $db^{[l]}$

Cache $z^{[l]}$.



$$\begin{cases} w^{[l]} = w^{[l]} - \alpha \cdot dw^{[l]} \\ b^{[l]} = b^{[l]} - \alpha \cdot db^{[l]} \end{cases}$$

$$da^{[l]} = \frac{-y}{a} + \frac{(1-y)}{(1-a)}$$

$$dA^{[l]} = \frac{-y^{[l]}}{a^{[l]}} + \frac{(1-y^{[l]})}{(1-a^{[l]})}$$

六. Forward and Backward Propagation.

1. Forward propagation for layer l

Input: $a^{[l-1]}$

Output: $a^{[l]}$. Cache($z^{[l]}$) → $w^{[l]}, b^{[l]}$

$z^{[l]} = w^{[l]} \cdot a^{[l-1]} + b^{[l]}$

$a^{[l]} = g^{[l]}(z^{[l]})$

Vectorized:

$Z^{[l]} = w^{[l]} \cdot A^{[l-1]} + b^{[l]}$

$A^{[l]} = g^{[l]}(Z^{[l]})$

2. Backward propagation for layer l

Input: $da^{[l]}$

Output: $da^{[l-1]}$, $dw^{[l]}$, $db^{[l]}$

$dz^{[l]} = da^{[l]} * g^{[l]\prime}(z^{[l]})$

$dw^{[l]} = dz^{[l]} \cdot a^{[l-1]}$

$db^{[l]} = dz^{[l]}$

$da^{[l-1]} = w^{[l]T} \cdot dz^{[l]}$

$dz^{[l]} = w^{[l+1]T} \cdot dz^{[l+1]} * g^{[l]\prime}(z^{[l]})$

Vectorized:

$dZ^{[l]} = dA^{[l]} * g^{[l]\prime}(Z^{[l]})$

$dw^{[l]} = \frac{1}{m} \cdot dz^{[l]} \cdot A^{[l-1]T}$

$db^{[l]} = \frac{1}{m} \cdot np.sum(dz^{[l]}, axis=1, keepdims=True)$

$dA^{[l-1]} = w^{[l]T} \cdot dz^{[l]}$

(算法复杂性可能来源于数据 而非代码)

6. Parameters. VS. Hyperparameters

(使运行高行. 更好地配置超参数)

1. Hyperparameters

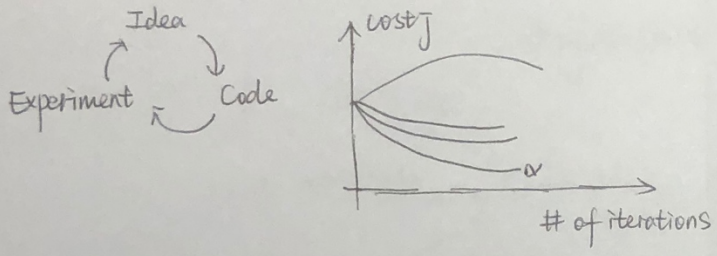
parameters: $W^{[1]}$. $b^{[1]}$. $W^{[2]}$. $b^{[2]}$---

Hyperparameters: learning rate α.

(最终确定了参数 W. b)

- # iterations. (迭代次数)
- # Hidden Layers L.
- # Hidden Units. ($n^{[1]}$. $n^{[2]}$--)
- Choice of Activation Functions.

2. Applied Deep Learning is a very empirical process.



尝试许多不同的配置. 构建模型并运行. 看比较结果.

Name: Xiaofan Wang

Improving Deep Neural Network: Hyperparameter tuning, Regularization and Optimization.

Ch 1. Practical Aspects of Deep Learning

一. Set up Machine Learning Application.

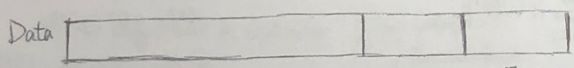
1. Train / Dev / Test Sets.

Applied ML is a highly iterative process:

- # layers
- # hidden units
- learning rates
- activation functions

尝试找到一个越来越好的神经网络而不断迭代.

恰当地将 Dataset 分为 Train / Dev / Test sets 可提高迭代效率
(训练集 / 开发集 / 测试集)



训练算法.

测试许多模型.

评估结果.

比较哪个效果好.

最佳比例:
- 60% / 20% / 20%
- 98% / 1% / 1%.
- 99.5% / 0.4% / 0.1%.

Mismatched Train / Test Distribution:

- Training Set: Cat pictures from webpages (高清)
- Dev / Test Sets: Cat pictures from users using your app (模糊)

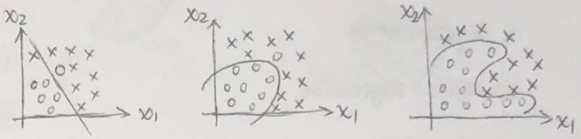
⇒ Make sure Dev / Test come from same distribution.

⇒ Not Having a Test Set might be Okay. (Only Dev Set)

测试集的目的是得到一个无偏估计 来评价最终选取的网络的性能. 当只有开发集时, 用训练集尝试不同的模型结构 并尝试得到一个好模型 (此时无需无偏估计模型)

2. Bias 偏差 (偏离度) & Variance 方差 (集中度).

"偏差-方差困境"



High Bias "underfitting"

"Just Right"

High Variance. "Overfitting"

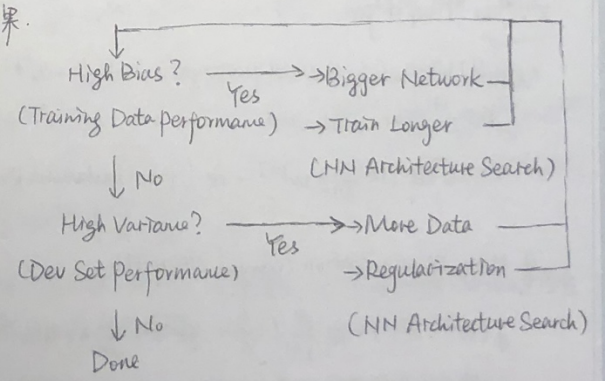
Cat Classification: $\begin{cases} y=1 \\ y=0 \end{cases}$

Training Set Error:	1%	15%	15%	0.5%
Dev Set Error:	11%	16%	30%	1%
	High Var.	High Bias	High Var	low V
			High Bias	low B

此时, 理想误差 (贝叶斯误差) ≈0.

Training / Dev Set from the same Distribution.

3. Basic recipe for Machine Learning.



	Bias	Variance
	↓	↑
(cost: computation time)	↑	↓

Bigger Network: Decrease Bias, No impact on Var

Regularization: Decrease Var, No impact on Bias.

二. Regularizing Neural Network.

(防止高方差. 过拟合)

1. Regularization

① Logistic Regression

$\min_{w,b} J(w,b)$

$w \in R^{n_x}$. $b \in R$

$J(w,b) = \frac{1}{m}\cdot\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\|w\|_2^2 + \underbrace{\frac{\lambda}{2m}\cdot b^2}_{\text{Omit.}}$

L2 Regularization: $\|w\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T\cdot w$

(w will be sparse) $\frac{\lambda}{2m}\cdot\sum_{j=1}^{n_x}|w_j| = \frac{\lambda}{2m}\|w\|_1$

λ = regularization parameter

② Neural Network

$J(w^{[1]}, b^{[1]}, \cdots w^{[l]}, b^{[l]}) = \frac{1}{m}\cdot\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L}\|w^{[l]}\|_F^2$

$\|w^{[l]}\|_F^2 = \sum_{i=1}^{n^{[l-1]}}\sum_{j=1}^{n^{[l]}}(w_{ij}^{[l]})^2$, $w: (n^{[l]}, n^{[l-1]})$

$dw^{[l]} = (\text{from backward propagation}) + \frac{\lambda}{m}\cdot w^{[l]}$ $\left(\frac{\partial J}{\partial w^{[l]}} = dw^{[l]}\right)$

$w^{[l]} = w^{[l]} - \alpha\cdot dw^{[l]}$

$= (1-\alpha\cdot\frac{\lambda}{m})\, w^{[l]} - \alpha\cdot(\text{from backward propagation})$.

2. Why Regularization reduces Overfitting.

$J(w^{[l]}, b^{[l]}) = \frac{1}{m}\cdot\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L}\|w^{[l]}\|_F^2$

$w^{[l]} \approx 0$

隐藏单元的权重接近0. 隐藏单元的影响被消除了.

简化了 Network. 但很可能深度变大. ~~Wait~~ (这个过拟合网络可能是高偏差状态. 但只存在一个中间值使 $w \approx 0$)

⇓

不容易过拟合的小型神经网络.

3. Dropout Regularization.

先遍历 Network 的每层. 每个节点的丢弃概率 0.5.

按随机编码决定每个节点的去留.

Implement Dropout:

Layer = 3. keep.prob = 0.8. (随机设置使 a3 不变)

d3 = np.random.randn(a3.shape[0], a3.shape[1]) < keep.prob

a3 = np.multiply(a3, d3)

~~a3~~ a3 /= keep.prob

50 units → 10 units shut off.

$z^{[4]} = w^{[4]}\cdot a^{[3]} + b^{[4]}$

↑ reduced by 20%.

/= 0.8.

Making Predictions at Test Time:

$a^{[0]} = x$

No DropOut: $z^{[1]} = w^{[1]}a^{[0]} + b^{[1]}$

$a^{[1]} = g^{[1]}(z^{[1]})$

$z^{[2]} = w^{[2]}a^{[1]} + b^{[2]}$

$a^{[2]}$ - -

↓

$\hat{y}$

① 此时. 不用抛硬币来决定哪些隐藏单元要被消除. 不想让输出也是随机的. 会为预测增加噪声.

② 用不同的随机失活的 Network 进行多次预测取平均值. ⇒ 运算效率不高.

会得到几乎相同的预测~

③ 反向随机失活: 保证如果测试过程没有对随机失活算法进行缩放 (scaling), 那么激活函数的期望输出也不会改变. 所以, 不用加入额外的缩放参数.

4. Understand Dropout

Why does drop-out work?

Intuition: Can't rely on any one feature.

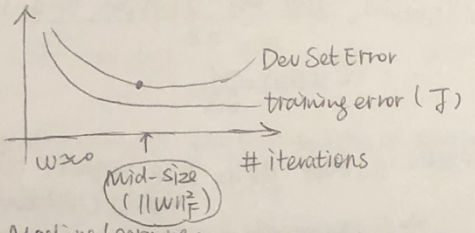
So have to spread out weights. (shrink weights) L_2

不要把太多的权重放在某个上. 对于每个输入都给一个较小权重.

三. Other Regularization Methods.

① Data Augmentation. 数据扩增.

② Early Stopping. 早终止法. (提前终止训练)



Machine Learning:

- Algorithm & Optimize "J" → Gradient / Momentum / RMSProp / Adam
- Not Overfit: Regularization

Orthogonalization 正交化: 同一时间只考虑一个任务.

Early Stopping 优点: 只运行一次梯度下降的过程.

(可用L2代替)

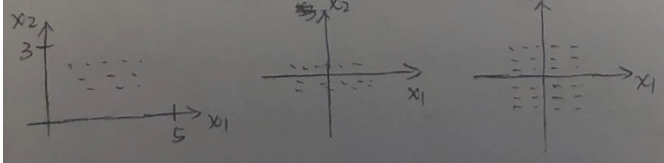
要尝试小w值. 中w值. 大w值.

而不用尝试L2正则化中各种λ.

三. Optimization.

1. Normalize Inputs

Normalize Training Sets

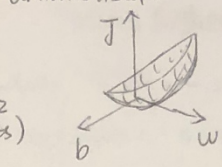


$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$ $\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} x^{(i)**2}$

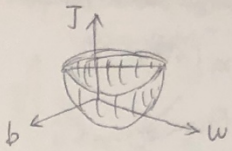
$x = x - \mu$

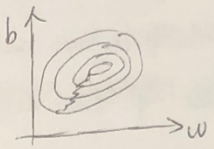
Why normalize Inputs? $J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$

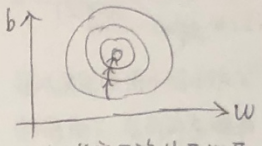
Unnormalized:



Normalized:



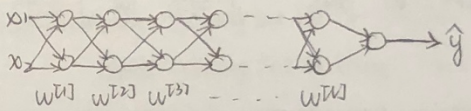




在梯度下降中采用更长的步长.

使特征尺度更近. 优化算法速度.

2. Vanishing / Exploding Gradients



$g(z) = z$. $b^{[l]} = 0$.

$\hat{y} = w^{[l]} \cdot w^{[l-1]} \cdot w^{[l-2]} \cdots w^{[3]} \cdot w^{[2]} \cdot w^{[1]} \cdot x$

$z^{[1]} = w^{[1]} \cdot x$

$a^{[1]} = g(z^{[1]}) = z^{[1]}$

$a^{[2]} = g(z^{[2]}) = g(z^{[2]} \cdot a^{[1]})$

$w^{[l]} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}$

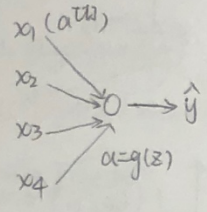
$\hat{y} = w^{[l]} \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^{l-1} \cdot x$

① 对于很深的 Network. L会很大. 以1.5的层数次方的速度变大. y的值就会爆炸

② 如果以0.5代替1.5. 激活函数的值会指数级下降. 此时梯度下降步长很小. 用很长时间才会产生学习.

3. Weight Initialization for Deep Networks.

① Single Neuron Example: (激活函数是 ReLU)



$z = w_1x_1 + w_2x_2 + \cdots + w_nx_n$ (+b)

larger n → smaller w_i

$Var(w_i) = \frac{2}{n}$ (n: 神经元特征数)

$W^{[l]} = np.random.randn(shape) * np.sqrt\left(\frac{2}{n^{[l-1]}}\right)$

ReLU: $g^{[l]}(z) = ReLU(z)$

如果输入经过激活的特征平均值为0. 且标准差为1.

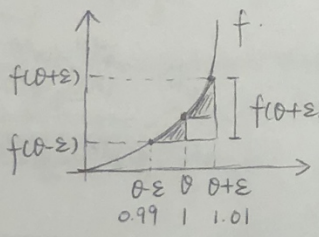
就可使z项也呈现相同的分布性质.

虽然不完全解决问题. 但通过设置权重 W. ~~使w不会比1稍多~~. 使 $W \approx 1$. ($W \neq 1$). 因此梯度不会过快膨胀/消失.

② 激活函数: tanh. $\sqrt{\frac{1}{n^{[l-1]}}}$ 或 $\sqrt{\frac{2}{n^{[l-1]} + n^{[l]}}}$

4. Numerical Approximation of Gradients.

Check Derivative computation:



$f(\theta) = \theta^3$

($\varepsilon = 0.01$)

估计大△ better 小△.

(将两个小△合为一起考虑)

$$g(\theta) \approx \frac{f(\theta+\varepsilon) - f(\theta-\varepsilon)}{2\varepsilon} = \frac{(1.01)^3 - (0.99)^3}{2\times 0.01} = 3.0001$$

$g(\theta) = 3\theta^2 = 3$.

approximate error: 0.0001 (双双侧差值)

单侧差值: 3.0301 − 3 = 0.0301

所以, 双双侧 Better Than 单侧

$$\begin{cases} f'(\theta) = \lim_{\varepsilon\to 0} \frac{f(\theta+\varepsilon) - f(\theta-\varepsilon)}{2\varepsilon} \Rightarrow O(\varepsilon^2) \Rightarrow 0.01^2 \checkmark \\ f'(\theta) = \lim_{\varepsilon\to 0} \frac{f(\theta+\varepsilon) - f(\theta)}{\varepsilon} \Rightarrow O(\varepsilon) \Rightarrow 0.01 \end{cases}$$

5. Gradient Checking.

① Take $W^{[1]}, b^{[1]}, \ldots W^{[L]}, b^{[L]}$ and reshape into a big vector θ.

$\Rightarrow J(W^{[1]}, b^{[1]}, \ldots W^{[L]}, b^{[L]}) = J(\theta)$

② Take $dW^{[1]}, db^{[1]}, \ldots dW^{[L]}, db^{[L]}$ and reshape into a big vector $d\theta$.

$J(\theta) = J(\theta_1, \theta_2, \ldots \theta_L)$.

for each i:

$$d\theta_{approx}[i] = \frac{J(\theta_1, \ldots \theta_i+\varepsilon) - J(\theta_1, \theta_2 \cdots \theta_i - \varepsilon \cdots)}{2\varepsilon} \approx d\theta[i] = \frac{\partial J}{\partial \theta_i}$$

CHeck: $\|d\theta_{app} - d\theta\|_2 \approx 10^{-7}$ → great! (Debug). (10^{-5}) C10^{-3}. worry).

$\Rightarrow \|d\theta_{approx}\|_2 + \|d\theta\|_2$ $\varepsilon = 10^{-7}$

Notes:

- Don't use in training. Only to debug.

  $d\theta_{approx}[i]$: slow. 当用梯度下降时. 用反向传播计算 $d\theta$.

- If algorithm fails grad check, look at components to try to identify bug.

  当 $d(\theta_{approx})$ 与 $d\theta$ 差距很大. ⇒ 某层 dw.

- Remember Regularization.

  $J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L} \|w^{[l]}\|_F^2$

- Doesn't work with Dropout.

- Run at random initialization. perhaps again after some training.

Ch 2 Optimization Algorithms

一. Mini-batch Gradient Descent (Big Data)

Batch vs Mini-batch:

Vectorization allows to efficiently compute on m examples.

$\vec{X} = [\, x^{(1)}\ x^{(2)} \cdots x^{(1000)} \mid x^{(1001)} \cdots x^{(2000)} \mid \cdots \mid \cdots x^{(m)} \,]$

(n_x, m)

$X^{\{1\}}$ $(n_x, 1000)$ $X^{\{2\}}$ $(n_x, 1000)$ $X^{\{5000\}}$ $(n_x, 1000)$

$\vec{Y} = [\, y^{(1)}\ y^{(2)} \cdots y^{(1000)} \mid y^{(1001)} \cdots y^{(2000)} \mid \cdots \mid \cdots y^{(m)} \,]$

$(1, m)$

$y^{\{1\}}$ $(1, 1000)$ $y^{\{2\}}$ $(1, 1000)$ $y^{\{5000\}}$ $(1, 1000)$

If m = 5 million $\Rightarrow$ 5000 mini-batches of 1000 each.

Mini-Batch T: $X^{\{T\}}$, $Y^{\{T\}}$.

Notation: $x^{(i)}$, $z^{[l]}$, $X^{\{T\}}$, $Y^{\{T\}}$.

for $t = 1, \cdots, 5000$

forward prop on $X^{\{t\}}$

$Z^{[1]} = W^{[1]} X^{\{t\}} + b^{[1]}$

$A^{[1]} = g^{[1]}(Z^{[1]})$

$\vdots$

$A^{[L]} = g^{[L]}(Z^{[L]})$

Compute Cost $J^{\{t\}} = \frac{1}{1000} \sum_{i=1}^{l} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2 \cdot 1000} \sum_{l=1}^{L} \| W^{[l]} \|_F^2$

Backward Prop to Compute Gradient ~~Descent~~ Cost $J^{\{t\}}$ $(X^{\{t\}}, Y^{\{t\}})$

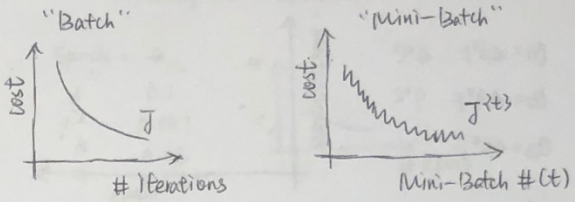
$W^{[l]} = W^{[l]} - \alpha \cdot dW^{[l]}$ $\quad b^{[l]} = b^{[l]} - \alpha \cdot db^{[l]}$

不断训练 Training Set 使它收敛在某个近似收敛值.

中 "Mini-Batch" faster than "Batch"

二. Understand Mini-batch Gradient Descent

1. Training with mini-batch Gradient Descent



每次迭代将遍历整个训练集

每次迭代使用不同小块 $X^{\{t\}}, Y^{\{t\}}$ 趋势向下. 会有许多噪声

2. Choose the Mini-batch Size.

If Mini-batch size = m : Batch Gradient Descent.

$(X^{\{1\}}, Y^{\{1\}}) = (X, Y)$

If Mini-batch size = 1 : Stochastic Gradient Descent.

① Batch: Large Training Set $\Rightarrow$ Too long per iteration.

Small Training Set is Okay. ($m \leq 2000$)

② Stochastic: Use a sample to renew gradient.

Lose speed-up from vectorization

每次只训练一个数据 : Not efficient.

③ In-between: Fastest Learning.

- vectorization.
- 不用等整个训练集都遍历完才运行梯度下降. 每遍历一次训练集. 可执行 5000 次梯度下降算法.

3. Typical Mini-batch Size

64	128	256	512	1024
2^6	2^7	2^8	2^9	2^{10}

Make mini-batch fit in CPU / GPU Memory : $X^{\{t\}}, Y^{\{t\}}$

三. Exponentially Weighted Averages · 指数加权平均

Temperature in London:

$\theta_1 = 40°F$ 4°C
$\theta_2 = 49°F$ 9°C
$\theta_3 = 45°F$
⋮
$\theta_{180} = 60°F$ 15°C
$\theta_{181} = 56°F$
⋮

V_t 近似于 $\frac{1}{1-\beta}$ 天的均温

$\beta = 0.9 \approx 10$ days' temp

$\beta = 0.98 \approx 50$ days' temp

$\beta = 0.5 \approx 2$ days' temp

$V_0 = 0$
$V_1 = 0.9 V_0 + 0.1\theta_1$
$V_2 = 0.9 V_1 + 0.1\theta_2$
$V_3 = 0.9 V_2 + 0.1\theta_3$
⋮
$V_t = 0.9 V_{t-1} + 0.1\theta_t$

$V_t = \beta \cdot V_{t-1} + (1-\beta)\theta_t$

$V_{100} = 0.9 V_{99} + 0.1\theta_{100}$
$V_{99} = 0.9 V_{98} + 0.1\theta_{99}$
$V_{98} = 0.9 V_{97} + 0.1\theta_{98}$

$V_{100} = 0.1\theta_{100} + 0.9(0.1\theta_{99} + 0.9(0.1\theta_{98} + 0.9 V_{97}))$
$= 0.1\theta_{100} + 0.1\times 0.9\,\theta_{99} + 0.1\times(0.9)^2\theta_{98} + \cdots$

Bias correction:

$V_0 = 0$
$V_1 = 0.98 V_0 + 0.02\theta_1$
$V_2 = 0.98 V_1 + 0.02\theta_2$
$= 0.98\times 0.02\theta_1 + 0.02\theta_2$
$= 0.0196\theta_1 + 0.02\theta_2$

$\frac{V_t}{1-\beta^t}$

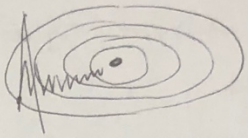
$t=2$: $1-\beta^t = 1-(0.98)^2 = 0.0396$

$\frac{V_2}{0.0396} = \frac{0.0196\theta_1 + 0.02\theta_2}{0.0396}$

当 t is very large, bias correction has not impact on Operation

When t is small, Biascorrection gives a better approximation

四. Gradient Descent with Momentum (动量)



↕ slower learning
⟷ faster learning.

上下振动会减慢梯度下降速度. 无法使用较大的 α.
(如果 α 太大, 可能超调)

On iteration t: Compute dw, db on Mini-Batch.

$V_{dw} = \beta \cdot V_{dw} + (1-\beta)\,dw$
$V_{db} = \beta \cdot V_{db} + (1-\beta)\cdot db$
$w = w - \alpha \cdot V_{dw}$
$b = b - \alpha \cdot V_{db}$

"$V_0 = \beta V_0 + (1-\beta)\theta_t$"

acceleration
将动量项看作球沿碗向下滚的速度.
β: friction.
V: velocity.

Hyperparameters: α. β. ($\beta = 0.9$ 是非常稳健的参数值).

五. RMSprop (均方根传递. "加速")

↕ slow
⟷ fast.

On iteration t: Compute dw. db on Mini-Batch

$S_{dw} = \beta_1 S_{dw} + (1-\beta_1)\,dw^2$ ← small
$S_{db} = \beta_2 S_{db} + (1-\beta_2)\,db^2$ ← large

$w = w - \alpha \cdot \frac{dw}{\sqrt{S_{dw}} + \varepsilon}$ $b = b - \alpha \cdot \frac{db}{\sqrt{S_{db}} + \varepsilon}$

⇓
≈ 0 ($\neq 0$)

六. Adam Optimization Algorithm

(本质：put "Momentum" and "RMSprop" together)

已证明在许多不同种类的神经网络构架中都十分有效.

$V_{dw}=0$. $S_{dw}=0$. $V_{db}=0$. $S_{db}=0$

On iteration t: Compute dw. db on Mini-Batch.

$V_{dw}=\beta_1 \cdot V_{dw}+(1-\beta_1)\,dw$,
$V_{db}=\beta_1 \cdot V_{db}+(1-\beta_1)\,db$ } "Momentum" (β_1)

$S_{dw}=\beta_2 \cdot S_{dw}+(1-\beta_2)\,dw^2$
$S_{db}=\beta_2 \cdot S_{db}+(1-\beta_2)\,db^2$ } "RMSprop" (β_2)

$V_{dw}^{corrected}=V_{dw}/(1-\beta_1^t)$, $V_{db}^{corrected}=V_{db}/(1-\beta_1^t)$

~~Vdw~~ $S_{dw}^{corrected}=S_{dw}/(1-\beta_2^t)$, $S_{db}^{corrected}=S_{dw}/(1-\beta_2^t)$

$$w=w-\alpha\cdot\frac{V_{dw}^{corrected}}{\sqrt{S_{dw}^{corrected}}+\varepsilon},\quad b=b-\alpha\cdot\frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}}+\varepsilon}$$

Hyperparameters:
- α: needs to be tuned. (尝试不同值比较效果)
- $\beta_1=0.9$ (动量算法 dw 的移动平均/加权平均) → 第一阶矩
- $\beta_2=0.999$ (dw^2. db^2 的移动加权平均) → 第二阶矩
- ε: 10^{-8}. (Default) 都可以.

"Adam": Adaptive Moment Estimation (自适应矩估计)

七. Learning Rate Decay

当用适量小样本迭代时，步长会逐渐向最小值靠近. 但不会完全收敛至该点，(由于 α 是个固定值)

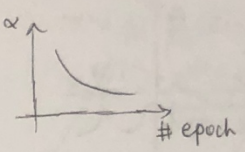
~~Start~~ α Large. → α small

(步长变小) => 最终围绕局极小值点更近的区域摆动

1 epoch = 1 pass through data.

$$\alpha=\frac{1}{1+\text{decay\_rate}*\text{epoch\_num}}\cdot\alpha_0$$

Epoch	α
1	0.1
2	0.067
3	0.05
4	0.04
⋮	⋮

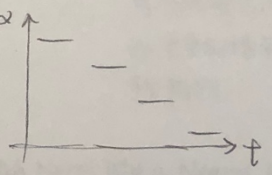


可尝试不同的超参数组合.
寻找一组效果好的数值 (α_0)

Other Learning Rate Decay Methods:

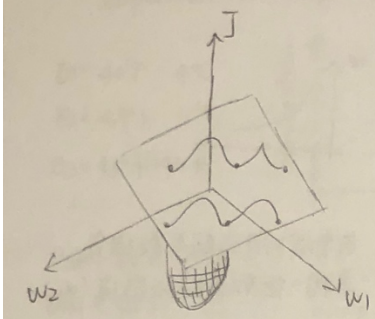
$\alpha=0.95^{epoch\_num}\cdot\alpha_0$ (exponentially decay)

$\alpha=\frac{k}{\sqrt{epoch\_num}}\cdot\alpha_0$ or $\frac{k}{\sqrt{t}}\cdot\alpha_0$

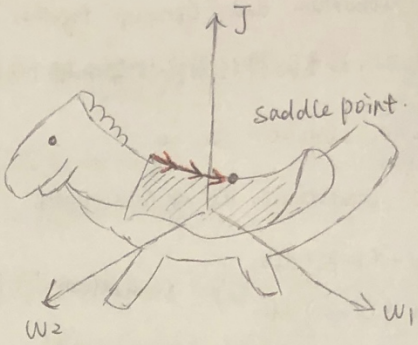


设置一个固定的 α.
还要使它优化的好.
可真正加速算法运行

11. The problem of Local Optima.



容易陷入局部最优
而找不到全局最优.



Problem of Plateaus:
"停滞滞区". 导数长时间接近于0的一段区域.

- Unlikely to get stuck in a bad local optima.
- Plateaus can make learning slow

"Adam Algorithm": 加快停滞区向下移动.

Ch3. Tuning Process

一. Hyperparameter Tuning

1. Tuning Process

对于超参数. Try random values. 而不是在网格中规则抽样.

将更充分地对最重要的超参数尝试多种可能组合.

⇒ 区域限定. 在某区进行更密集的抽样. (区域搜索)

2. Using an appropriate scale to pick hyperparameters.

- 随机抽样
- 在a~b范围内均匀随机取样

3. Hyperparameters tuning in Practice: Pandas vs. Caviar

① Re-test hyperparameters occasionally: 确保这些数值依旧是最优解

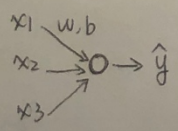
② Babysitting one model

③ Training many models in parallel

二. Batch Normalization

1. Normalize Activations in a Network.

Normalize Inputs to speed up Learning:



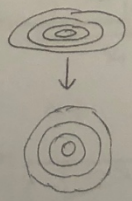
(Logistic Regression.)

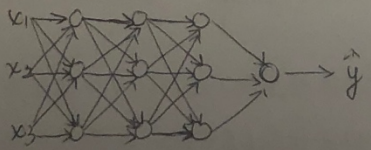
$$\mu = \frac{1}{m}\cdot\sum_{i=1}^{m} x^{(i)}$$

$$x = x - \mu$$

$$\sigma^2 = \frac{1}{m}\cdot\sum_{i=1}^{m} x^{(i)2}$$

$$x = x / \sigma^2$$





Give some intermediate values in NN:

$z^{(1)} \cdots z^{(m)}$ (under) $z^{[l](i)}$

$$\mu = \frac{1}{m}\cdot\sum_{i=1}^{m} z^{(i)}$$

$$\sigma^2 = \frac{1}{m}\cdot\sum_{i=1}^{m} (z_i - \mu)^2$$

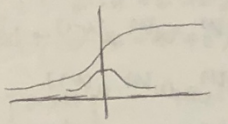
$$z_{norm}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

$$\tilde{z}^{(i)} = \gamma \cdot z_{norm}^{(i)} + \beta \quad (\gamma, \beta: \text{can be learned from model})$$

If $\gamma = \sqrt{\sigma^2 + \varepsilon}$.

$\beta = \mu$.

Then $\tilde{z}^{(i)} \approx z^{(i)}$

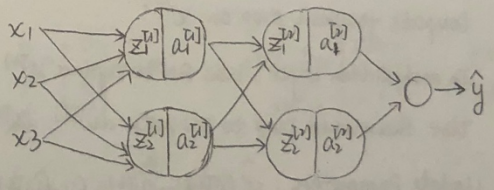


隐藏层归一化后并不一定是均值为0. 方差为1.

如果激活函数是 sigmoid. 可能希望它们有更大方差. 以便更好地利用S函数非线性特性.

2. Fitting Batch Norm into a Neural Network.

① Adding Batch Norm to a Network:



$$X \xrightarrow{W^{[1]}, b^{[1]}} z^{[1]} \xrightarrow[\text{Batch Norm (BN)}]{\beta^{[1]}, \gamma^{[1]}} \tilde{z}^{[1]} \longrightarrow a^{[1]} = g^{[1]}(\tilde{z}^{[1]})$$

$$W^{[1]}, b^{[1]} \rightarrow z^{[1]} \xrightarrow[\text{BN}]{\beta^{[1]}, \gamma^{[1]}} \tilde{z}^{[1]} \rightarrow a^{[1]} \cdots$$

Parameters:

$w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}, \cdots w^{[l]}, b^{[l]}$

$\beta^{[1]}, \gamma^{[1]}, \beta^{[2]}, \gamma^{[2]}, \cdots \beta^{[l]}, \gamma^{[l]}$

$d\beta^{[l]}$

$\beta^{[l]} = \beta^{[l]} - \alpha \cdot d\beta^{[l]}$

tf.nn.batch-normalization.

② Working with Mini-Batches:

$X^{\{1\}} \xrightarrow{W^{[1]}, b^{[1]}} Z^{[1]} \xrightarrow{\beta^{[1]}, \gamma^{[1]}} \tilde{Z}^{[1]} \longrightarrow g^{[1]}(\tilde{Z}^{[1]}) = A^{[1]}$

$\xrightarrow{W^{[2]}, b^{[2]}} Z^{[2]} \rightarrow \cdots$

$X^{\{2\}} \rightarrow Z^{[1]} \xrightarrow[BN]{\beta^{[1]}, \gamma^{[1]}} \tilde{Z}^{[1]} \rightarrow \cdots$

$\cdots$

Parameters: $W^{[1]}$, ~~$b^{[1]}$~~, $\beta^{[1]}$, $\gamma^{[1]}$ (维度: $n^{[1]}, 1$).

$Z^{[1]} = W^{[1]} a^{[1-1]} + b^{[1]}$

$Z^{[1]} = W^{[1]} a^{[1-1]}$

$Z^{[1]}_{Norm}$

$\tilde{Z}^{[1]} = \gamma^{[1]} Z^{[1]}_{Norm} + \boxed{\beta^{[1]}}$

BN算法使层级中各个 $Z^{[1]}$ 的均值为0. 因此无需保留 $b^{[1]}$ 而是用 $\beta^{[1]}$ 来代替. 控制最终偏移量影响的参数.

③ Implement Gradient Descent:

for $t=1 \cdots$ Mini-Batches.

Compute forward prop on $X^{\{t\}}$

In each hidden layer, use BN to replace $Z^{[1]}$ with $\tilde{Z}^{[1]}$

Use Backward prop to compute $dW^{[1]}$, ~~$db^{[1]}$~~, $d\beta^{[1]}$, $d\gamma^{[1]}$

Update parameters: $W^{[1]} = W^{[1]} - \alpha \cdot dW^{[1]}$

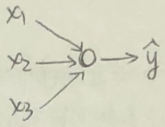
$\beta^{[1]} = \beta^{[1]} - \alpha \cdot d\beta^{[1]}$

$\gamma^{[1]} \cdots$

同样适用于 Momentum · RMSprop · Adam 的梯度下降算法.

3. Why does Batch Norm work?

Learning on shifting Input Distribution:



Cat: $y=1$

Non-Cat: $y=0$.

BN: 减少了隐藏单元值分布的不稳定性.

① 减少输入值变化所产生的问题.

神经网络的后层. 可以有更加稳固的基础)

⇒ 后面层适应前面层变化的力量被减弱.

Batch Norm as Regularization:

- Each mini-batch is scaled by the mean/variance computed on just that mini-batch.
- This adds some noise to the values $Z^{[1]}$ within that mini-batch. So similar to dropout, it adds some noise to each hidden layer's activations.
- This has a slight regularization effect.

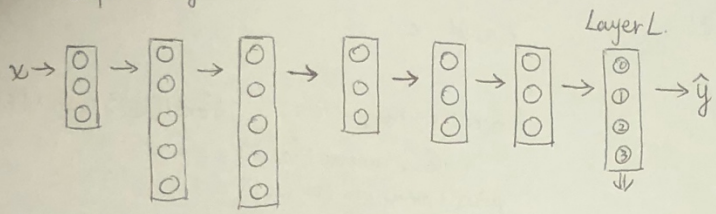
4. Batch Norm at test time.

$\mu = \frac{1}{m} \cdot \sum_i z^{(i)}$. $\sigma^2 = \frac{1}{m} \cdot \sum_i (z^{(i)} - \mu)^2$

$z^{(i)}_{norm} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$, $\tilde{z}^{(i)} = \gamma \cdot z^{(i)}_{norm} + \beta$.

三. Multi-Class Classification

1. Softmax Regression



$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$ (4,1)

Activation Function: $t = e^{(z^{[l]})}$

$$a^{[l]} = \frac{e^{z^{[l]}}}{\sum_{j=1}^{4} t_i}, \quad a_j^{[l]} = \frac{t_i}{\sum_{j=1}^{4} t_i}$$

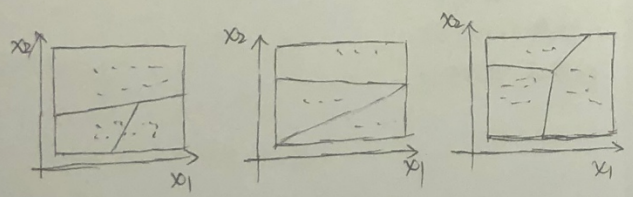
$a^{[l]} = g^{[l]}(z^{[l]})$ (4,1)

$$z^{[l]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \quad t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix} = \begin{bmatrix} 148.4 \\ 7.4 \\ 0.4 \\ 20.1 \end{bmatrix} \cdot \sum_{j=1}^{4} t_j = 176.3$$

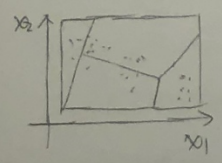
$$a^{[l]} = \frac{t}{176.3}$$

2. Softmax Example

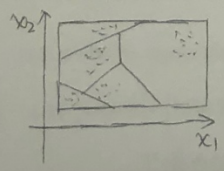
C=3: (决策边界是线性的. 不同的线性函数区分不同类别)



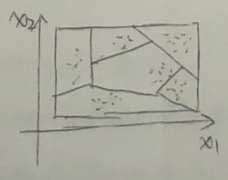
C=4:



C=5:



C=6:



这些图展示了 Softmax 分类器在没有隐藏层时，可以做什么. 有很深的网络，可用更复杂的非线性决策平面来区分不同类别

3. Training a Softmax Classifier

① Understand Softmax:

$$z^{[l]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \quad t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix}$$

$$a^{[l]} = g^{[l]}(z^{[l]}) = \begin{bmatrix} e^5/(e^5+e^2+e^{-1}+e^3) \\ e^2/(e^5+e^2+e^{-1}+e^3) \\ e^{-1}/(e^5+e^2+e^{-1}+e^3) \\ e^3/(e^5+e^2+e^{-1}+e^3) \end{bmatrix} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$$

"Hard Max" $\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

Softmax Regression generalizes logistic regression to C classes.

If C=2, softmax reduces to logistic regression.

$$a^{[l]} = \begin{bmatrix} 0.842 \\ 0.158 \end{bmatrix}$$

② Loss Function

$$\hat{y} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \rightarrow \text{cat.} \qquad a^{[l]} = \hat{y} = \underset{(4,1)}{\begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix}} \qquad C=4 \quad (4,1)$$

$$L(\hat{y}, y) = -\sum_{j=1}^{4} y_j \log \hat{y}_j$$

($L(\hat{y}, y)$ small)

$-y_2 \log \hat{y}_2 = -\log \hat{y}_2$. make $\hat{y}_2$ big.

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

$$Y = [y^{(1)}\ y^{(2)} \dots y^{(m)}] = \begin{bmatrix} 0 & 0 & 0 & \cdots \\ 1 & 0 & 0 & \cdots \\ 0 & 1 & 0 & \cdots \\ 0 & 0 & 1 & \cdots \end{bmatrix} \quad (4,m)$$

$$\hat{Y} = [\hat{y}^{(1)}\ \hat{y}^{(2)} \dots \hat{y}^{(m)}] = \begin{bmatrix} 0.3 & \cdots \\ 0.2 & \cdots \\ 0.1 & \cdots \\ 0.4 & \cdots \end{bmatrix} \quad (4,m)$$

③ Gradient Descent with SoftMax.

Forward prop: $z^{[1]}_{(4,1)} \rightarrow a^{[1]} = \hat{y} \rightarrow L(\hat{y}, y)$

Backward prop: $dz^{[1]}_{(4,1)} = \hat{y} - y$. $\frac{\partial J}{\partial z^{[1]}}$

session.run(train). 在train上运行下一步梯度下降

print(session.run(w))

Output: 0.1

for i in range(1000): 将迭代运行1000次梯度下降

session.run(train)

print(session.run(w))

Output: 4.99999. } Similar.

w(optim)=5.

(7) Programming Frameworks.

1. Deep Learning Frameworks

- Caffe / Caffe 2
- CNTK
- DL4J
- Keras
- Lasagne
- mxnet
- PaddlePaddle
- TensorFlow
- Theano
- Torch

☆ Choose DL Frameworks:

① Ease of programming (development & deployment)

② Running Speed.

③ Truly Open. (Open Source with good governance)

矩阵:

coefficients = np.array([[1.], [-10.], [25.]])

w=tf.Variable(0, ~~tf~~ dtype=tf.float32)

x=tf.placeholder(tf.float32, [3,1])

cost = x[0][0]*w**2 + x[1][0]*w + x[2][0]

⋮

session.run(train, feed_dict={x: coefficients})

Output: Same with ~~1st~~ Before

2. TensorFlow

Motivate Problem:

$J(w) = w^2 - 10w + 25 = (w-5)^2$

(cost function)

$w = 5$.

```
import numpy as np
import tensorflow as tf
w = tf.Variable(0, dtype=~~float32~~ tf.float32)  定义参数. (尝试优化的值)
cost = tf.add(tf.add(w**2), tf.multiply(-10., w)), 25)  定义代价函数J.
train = tf.train.GradientDescentOptimizer(0.01).minimize(cost)  最小化代价函数.
init = tf.global_variables_initializer()
session = tf.Session()  启动了一个TensorFlow的Session.
session.run(init)  初始化全局变量.
sess.run(w)  对变量w求值. (print(session.run(w)))
```

⇐ 如果使用Adam优化器, 只需更改此行.

这三行代码相当于 等同于

```
with tf.Session() as session:
    session.run(init)
    print(session.run(w))
```

=> Output: 0.0

Name: Xiaofan Wang

Structuring Machine Learning Projects

Ch.1

一. Introduction to ML Strategy

1. Why ML Strategy

学习更高效的构建机器学习系统的方法

Example: 开发猫分类模型 ⇒ 系统准确率 90%

Aim: 提升准确率

Ideas:
- collect more data
- Collect more diverse training set
- Train Algorithm longer with gradient descent
- Try Adam instead of gradient descent
- Try bigger or smaller Network
- Try dropout
- Add L2 regularization
- Network Architecture:
  - Activation Functions
  - # Hidden Units
  - ...

☆ 分别写出哪些方法值得尝试，哪些无意义（节省工作时间）

2. Orthogonalization 正交化

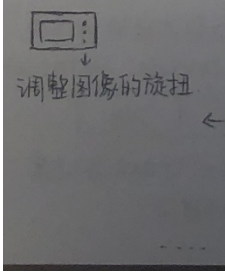
One of the challenges with building ML systems:

So many things you could try / change (有许多可尝试和改变的)

e.g. so many hyperparameters you could tune. 许多可调的超参数.

最高效的机器学习人员：非常清楚需要调整什么来实现一个预期效果. ⇒ Orthogonalization.

① TV tuning example.



(每个旋钮都有一个实际功能)

假设一个旋钮可同时控制

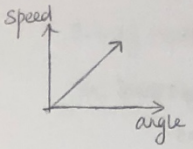
$0.1a + 0.3b - 1.7c - \cdots$

那用户几乎无法调整电视荧幕居中

⇒ Orthogonalization: 确保每个旋钮只可进行一个参数的调整

使画面居中

② car.



$0.3 \times angle - 0.8 \times speed$

$2 \times angle + 0.9 \times speed$

正交控制：可与真正根控制的事物保持一致.

③ Chain of assumptions in ML. (正交化原理)

调整系统旋钮，保证 4 点：

- Fit training set well on cost function (performance)
- Fit Dev Set well on cost function.
- Fit test set well on cost function
- perform well in real world.

二. Set up your goal

1. Single Number Evaluation Metric.

(设置一个单一的量化评估指标)

①

Classifier	Precision	Recall	F1-Score	权衡 PR
A	95%	90%	92.4%	✓
B	98%	85%	91%	

$F1\text{-}Score = \frac{2}{\frac{1}{P}+\frac{1}{R}}$ (调和平均值.)

一个好的验证集和单一量化评估指标可以加速迭代，让 ML-Algorithm 不断改进.

③ Another Example. (error rate)

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

C Better.

选择 C 算法进行迭代

2. Satisficing and Optimizing Metrics (满足指标. 优化指标)

① Another Cat Classification Example

Classifier	Accuracy	Running Time
A	90%	80ms
B	92%	95ms
C	95%	1500ms
	↓	↓
	Optimizing	Satisfying

cost = Accuracy − 0.5 × Running Time.

- Maximize Accuracy.
- satisfy: Running Time ≤ 100ms (满足条件即可)

对准确率和运行时间. 作权衡. or 通盘考虑.

N metrix:
- 1 optimizing.
- N-1 satisfying (达到阈值即可)

② Wake Words / Trigger Words

e.g.
- Amazon Echo: Alexa
- Google: Ok Google.
- Apple: Hey Siri
- 百度: 你好百度.

⇒ 用于告诉某个声控设备你想表达的意思.
以便开口听你讲话的词句.

⇓

- 触发词监测系统的准确度 (即实际唤醒系统的可能性)
- False Positive: 随机唤醒的可能性.

3. Train / Dev / Test Distributions

Cat Classification Dev / Test Sets.

Regions:
- Dev:
  - US
  - UK
  - Other Europe
  - South America
- Test:
  - India
  - China
  - Other Asia
  - Australia

结果不准确. 相当于挨着靶心.
数据来自不同的分布.

⇓

应使8个地区数据随机打乱然后分为 Dev / Test Sets.
(这样 Dev/Test ~~Dev~~ 来自相同分布)

True Story: (different Distribution).

坏 (不准) — Optimizing on dev set on loan approvals for medium income zip codes.
Tested on low income zip codes.

Guideline: 来自相同分布

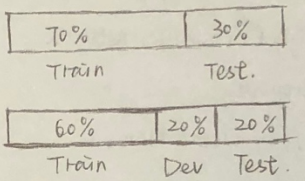
Choose a dev set and test set to reflect data you expect to get in the future and consider important to do well on.

(A or B.)

4. Size of Dev / Test Sets

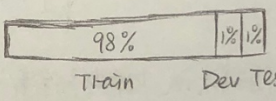
- Dev Set: 评估不同想法
- Test Set: 对最终分类器进行评估

① Old way of splitting Data: (不再适用).



样本量不大时. 很适用
100
1000
10000.

Modern:



(确保 Dev. Test Set 的大小足够达到目的)

数据量很大时. (1000000)
没必要划分 7/3. 6/2/2.
因为深度学习算法极度需要数据, 所以训练集占很大比例.

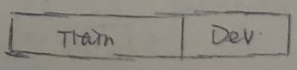
② Size of Test Set

Set test set to be big enough to give high confidence in the overall performance of your system. (足够能保证对系统整体性能评估的高置信度即可)

除非对于"最终系统性能"有非常精确的测量否则测试集不需上百万个样例.

③ *Only Train / Dev Set

在一些应用中. 不需对最终系统的整体性能评估有很高的置信度. ⇒ 只需 Train. Dev Sets. No Test Set.



如果 Dev Set 很大. 过拟合得不厉害. 可取此法. 但不建议.

5. When to change dev/test sets and metrics.

① Cat Dataset ~~Metric~~

Metric: classification error

Algorithm A: 3% error. (pornographic ···)

Algorithm B: 5% error.

⇒ Metric + Dev Set: perfer A.

You / Users: Perfer B.

⇒ 修改评估指标. 修改 Dev/Test Set.

怎是定义评估指标的一种方法

统计分类错误的指标: $\frac{1}{m_{dev}} \cdot \sum_{i=1}^{m_{dev}} L\{y_{pred}^{(i)} \neq y^{(i)}\}$

赋予 pornographic 权重 $w^{(i)}$: $\frac{1}{\sum w^{(i)}} \cdot \sum_{i=1}^{m_{dev}} w^{(i)} L\{y_{pred}^{(i)} \neq y^{(i)}\}$

$w^{(i)} = \begin{cases} 1 & \text{if } x^{(i)} \text{ is non-pornographic} \\ 10 & \text{if } x^{(i)} \text{ is pornographic} \end{cases}$

② Orthogonalization for cat pictures: anti-porn.

- So far we've only discussed how to define a metric (Place Target). to evaluate classifiers. 确定一个指标以衡量分类器在目标上的性能.
- Worry separately about how to do well on this metric.
  单独考虑如何在这个指标上得到很好地性能. (Aim / Shoot at the Target).

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

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

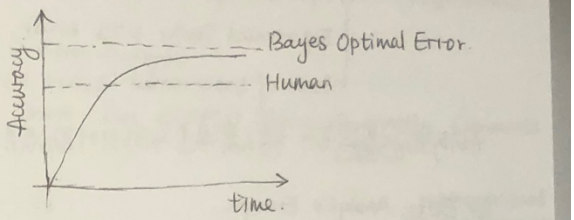
③ If doing well on your metric + dev/test set does not corresponding to doing well on your application, change your metric and dev/test set.

对于团队之作: 进快确定评估指标. Dev/Test Set
可以进行并发迭代.
如果效果不佳. 之后可再做修改.

三. Comparing to Human-Level Performance.

1. Why Human-Level Performance.

ML 取代 Human: ① 算法效果更好
② 解决问题的效率高.



① ML 超越 Human-Level 后, 可得到更好的效果, 但增长速度变慢.

② 可无限逼近最佳理论水平 (Bayes Optimal Error).
最小理论误差值

③ Humans are quite good at a lot of tasks. So long as ML is worse than humans. you can:

- Get labeled data from humans.
- Gain insight from manual error analysis: Why did a person get this right?
- Better analysis of bias/variance

2. Avoidable Bias.

Humans	1%	7% (B)	(B) 0.5%	Avoidable ~~Bayes~~ Bias. 7.5% (图片太模糊)
Training error	8%		8%	2% (V)
Dev error	10%	2% (V)	10%	

Focus on "Bias"
↓
Train bigger Neural Network.
Train longer time
↓
Improve performance of model on Training Set.

Focus on "Variance"
改进模型: 误差小到与 Bayes 误差齐平.
小于 Bayes 误差不可能
除非过拟合.

3. Understand Human-Level Performance.

① Human-Level Error as a proxy for Bayes Error.

Medical Image Classification Example:

Suppose:
- Typical Human 3% error
- Typical Doctor 1% error
- Experienced Doctor 0.7% error.
- Team of Experienced Doctors 0.5% error.

如何定义人类误差：将人类误差当成贝叶斯误差的替代值或估计值.

② Error Analysis Example.

Human (proxy for Bayes Error)	1% 0.7% 0.5%	1% 0.7% 0.5%	(0.7%更难改进) 0.5%
↕ Avoidable Bias	↕ 4%-4.5%	↕ 0-0.5%	↕ 0.2%
Training Error	5%	1%	0.7%
↕ Variance.	↕ 1%.	↕ 4%	↕ 0.1%
Dev Error	6%	5%	0.8%
	⇓	⇓	⇓
	Bias	Variance.	偏差. 方差都可改进

4. Surpass Human-Level performance.

- Online Advertising.
- Product Recommendation.
- Logistics (predicting transit time)
- Loan Approvals.

Structured Data. (Lots of Data): ML > Human-Level (单个监督学习 > 人类水平.)

Not Natural Language processing. (人类更擅长NLP)

NLP一些突破性例外:
- Speech recognition
- Image recognition
- Medical: EGG. Skin cancer.

5. Improve Model Performance.

① Two fundamental assumptions of supervised Learning:
- You can fit the training set pretty well.
- The training set performance generalizes pretty well to the dev/test set.

② Reduce Avoidable Bias and Variance.

Human-Level
↕ Avoidable Bias ⇒
- Train bigger model
- Train longer/better optimization algorithms.
- NN architecture/hyperparameter search. (RNN. CNN)

Training Error.
↕ Variance. ⇘
- More Data.
- Regularization: L2. dropout. Data Augmentation
- NN architecture/hyperparameters search.

Dev Error

Chapter 2.

一. Error Analysis

1. Carrying out error analysis.

① Look at dev examples to evaluate ideas.

Should you try to make your cat classifier do better on dogs?

Error Analysis (手动检测): { Get ~100 mislabeled dev set examples
Count up how many are dogs.

5/100. 5% (error: 10% → 9.5%)

⇓

并不是最好的利用时间的方式.

50/100. 50% (error: 10% → 5%)

⇓

值得付出努力: 专注于减少被错误标识的狗.

② Evaluate Multiple Ideas in Parallel.

Ideas for Cat Detection: (多个改进猫检测器的思路)

- Fix pictures of dogs being recognized as cats.
- Fix great cats (lion, panthers...) being misrecognized.
- Improve performance on blurry images.

Image	Dog.	Great Cats	Blurry	Instagram	Comments
1	✓			✓	Pitbull
2			✓	✓	
3		✓	✓		Rainy Day at zoo.
⋮	⋮	⋮	⋮	⋮	
% of total (分类错误的百分比)	8%	43%	61%	12%	

提高准确率的幅度大.

结果: 不同类别的错误. 有多少处理价值.

总结: 错误分析 → 找到一套在你验证集中错误标识的样本.

→ False positive. False Negative

计算不同类别的误判了数 (判断优先级)

→ 可能有新的错误类别. 新灵感.

2. Cleaning up Incorrectly labeled data (Output y)

① Training Set

- DL Algorithms are quite robust to random errors in the training set, especially in lots of data.
- But Not robust to systematic errors (引起系统性错误)

② Error Analysis.

Image ---	Dog	Great Cat	Blurry	Incorrectly Labeled	Comments
98				✓	labeler missed cat in background.
99		✓			
100				✓	Drawing of a cat. Not a real cat.
% of total	8%	43%	61%	6%	

Overall Dev Set Error: 10%		2%
Errors due to incorrect labels: 0.6% ⓧ		0.6% ✓
Errors due to other causes: 9.4%		1.4%
	(纠正无太大意义)	(纠正有效)

Goal of Dev Set: To help you select between two classifiers A & B.

③ Correcting Incorrect Dev/Test Set Examples.

- Apply same process to your dev and test sets to make sure they continue to come from the same distribution.

(Train and dev/test set may come from slightly different distributions.)

- Consider examining examples your algorithms got right as well as ones it got wrong.
(98%) (2%).

当一些标签错误的样本被纠正时. 很可能预测会变得不准.

总结: 当带领团队工作时. 希望理解这个机器犯的错. ⇒ 手动检查错误.

⇒ 可更好地计划 Next Step.

3. Build your first system quickly, then iterate.

① Speech Recognition Example. 语音识别系统.

- Noisy Background : cafe noise. car noise
- Accented Speech
- Far from microphone.
- Young children's speech
- Stuttering.

可以有很多方面来改善语音识别系统.

②

- Set up dev/test set and metric 建立开发集/测试集和评价指标 (target)
- Build initial system quickly.
  找到训练集，训练它然后看结果.
  开始观察并理解你的系统，对开发/训练集如何价值和度量指标是多少
- Use Bias/Variance Analysis & Error Analysis to prioritize next steps.

二. Mismatched training and dev/test data.

1. Training and testing on different distributions.

① Cat app example.

Data from webpage. 200000

Data from mobile app. 10000 (care about this).

210000 ~~(from different distribution)~~.

× Option 1:

train	dev	test
205000	2500	2500

→ 2381 webpage, 119 app.

优点：来自同一分布. 易于管理.

缺点：Dev Set 中分布比例失衡. 不能重点关注 App.

✓ Option 2: (长期性能更好)

train	dev	test
205000	2500 (app)	2500 (app)

优点：瞄准目标.

缺点：train set 与 dev/test Set 分布不同.

② Speech recognition example.

Training:
- Purchased Data
- Smart speaker control
- Voice keyboard.

500000

Dev/Test:
Speech activated rearview mirror.
20000.

Train	Dev	Test
500K.	10K	10K.

Train	Dev	Test
510K.	5K	5K.

总结：快速建立你的第一个系统，然后迭代. ~~[redacted]~~.

(不适用于你已经驾轻就熟的领域)

(不适用于已有很多可参考的学术论文).

比如：人脸识别

所依托论文，从一开始建立一个复杂系统.

但如果是研究一个新问题：
- 不要考虑太多.
- 不要将系统做的太复杂

快速建立一个系统，然后使用它.

来帮你确定一个优先改进你的系统.

2. Bias and Variance with Mismatched Data Distributions

① Cat classifier example: (Assume humans get ≈0% error).

Training Error: 1%

Dev Error: 10%

⇒ 如果 Train. Dev Set 来自同一分布，说明方差过大.

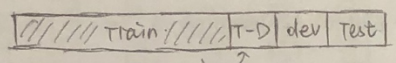
但是 Train Set 训练的很好. 但未能较好泛化到 Dev Set.

⇒ 如果 Train. Dev Set 来自不同分布.

只因高分辨率的图片，造成训练集难度较低.

但 Dev Set 较难. 不是方差问题. 而是 Dev Set 难以精准区分.

② Training-dev Set: Same distribution as training set.
but not used for training.



NN

			(Human) 0%	
Training Error	1%	1%	10%	Avoidable Bias.
Train-Dev Error	9%	1.5%	11%	
Dev Error	10%	10%	12%	Avoidable Bias.
	High Variance.	~~Dismatched~~. Data. (类型问题) Mismatch.		

0%	Avoidable Bias.
10%	
11%	~~High Variance~~.
20%	Data ~~Dismatch~~. Mismatch.

Bias + Data ~~Dismatch~~. Mismatch.

③ Bias / Variance on Mismatched training and Dev/Test Sets.

Human Level	4%	Avoidable Bias
Training Set Error	7%	
Training-Dev Set Error	10%	Variance
Dev Error	12%	Data ~~Dismatch~~. Mismatch.
Test Error	12%	Degree of overfitting to Dev Set.

从上到下应是↑的.

如果 Dev. Test Set 性能差太多，将神经网络调得太偏向 (overtune) 于开发集了.

特别：语音识别.

4%
7%
10%
6%
6%.

④ More general formulation.

	General (语音识别)	rearviewmirror	
Human Level	4%	6%	Bias
Error on Examples trained	Train Set 7%	6%	
Error on not examples trained	Train-Dev Set 10%	Dev/Test 6%	Var
	Data ~~Dismatch~~. Mismatch.		

总结：解决 ~~Data Dismatch~~. Data Mismatch.

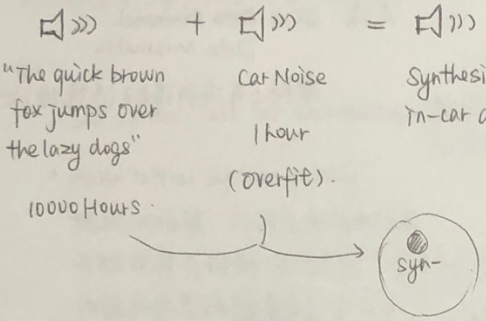
实际上并没有很好 (系统地) 处理方法

3. Address Data Mismatch.

① Carry out manual error analysis to try to understand difference between training and dev/test sets.

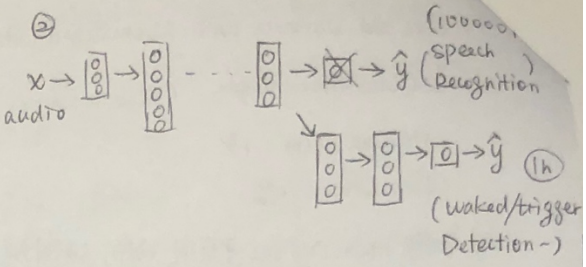
② Make training data more similar.
Or collect more data similar to Dev/Test Sets.

⇒ Artificial Data Synthesis 人工合成（应用：语音识别）.



点估：• 如果遇到 Data Mismatch. ⇒ Error Analysis.
比较 Train Set vs. Dev Set
引申两种数据分布的不同.
可没法获得更多像开发集的训练数据.

• 人工合成，可应用于语音识别.
使用时注意，是否意外地模拟了所有可能样本的子集的数据.



② When transfer Learning makes sense.

⇒ Task A and B have the same input x
(例：都是图片，or 语音).

⇒ You have more data for Task A than Task B.
因为A样本比B样本价值低.
"key"

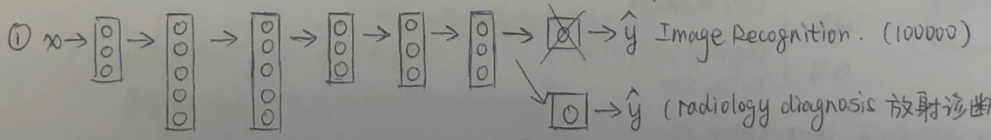
⇒ Low-e Level features from A could be helpful for learning B.

点估：迁移学习的关键是，将B做好才有效.
但B数据少，而A可获得大量数据，并从中学到许多低层次特征.
⇓ Transfer Learning.
将B学好.

(当 A < B 时，对B无增益)
学习.

三. Learning from multiple tasks.

1. Transfer Learning.



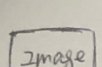
(x, y): 放射影像.
⇒ 然后重新在新的数据集上训练该NN（新放射数据）（数据多）
⇒ 也可只重新训练最后一层权值 $W^{[L]}$
~~(如果数据是很多)~~ (数据少)

然后为NN最后一层创建一个新的随机初始化的权重. ⇒ 使用新建的输出层，来进行放射结果诊断.

当你在做 Image Recognition 时，训练了所有常用的神经网络参数/权值，从而做出图像识别的预测.

3. Multi-task Learning (每个任务将有助帮其它任务).

① Simplified autonomous driving example:



$x^{(i)}$

$y^{(i)}$

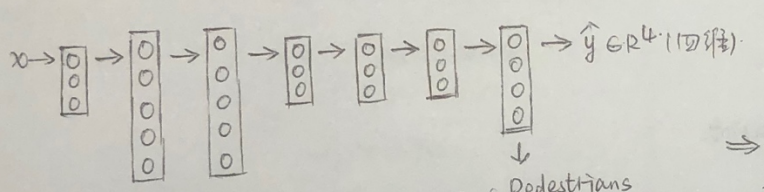
- Pedestrians: 0
- Cars: 1
- Stop Signs: 1
- Traffic Lights: 0

(4, 1).

$$Y = \begin{bmatrix} | & | & & | \\ y^{(1)} & y^{(2)} & \cdots & y^{(m)} \\ | & | & & | \end{bmatrix}$$

(4, m).

② Neural Network Architecture:



为3训练这个神经网络.
需要定义 损失函数:

多标签:
- Pedestrians
- Cars
- Stop Signs
- Traffic Lights

Loss: $\hat{y}^{(i)}$ (4,1) $\rightarrow \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{4} L(\hat{y}_j^{(i)}, y_j^{(i)})$

↓ usual Logistic Loss.

$$y_j^{(i)} \log \hat{y}_j^{(i)} + (1-y_j^{(i)}) \log(1-\hat{y}_j^{(i)})$$

⇒ 与 Soft max 回归不同.
(一个标签).

⇒ 该例是"多标签". 一张图片解决4个问题.
识图告诉你每个图像中的4个对象.
or 训练4个独立的 NN.
但不同的输出之间. 前面的特征可共享.

⇒ $Y = \begin{bmatrix} 1 & 0 & ? & \cdots \\ 0 & 0 & ? & \cdots \\ 1 & 1 & 1 & \cdots \\ 1 & 1 & 0 & \cdots \end{bmatrix}$ 存在许多"?"的 DataSet.
依然可训练你的算法.

③ When Multi-Task Learning Makes Sense.

⇒ Training on a set of tasks that could benefit from having shared lower-level features. (可共享低层次特征).

⇒ Usually: Amount of Data you have for each task is quite similar. (每个单项任务数据量相似).

- Transfer Learning: A (1000 000) → B (1000).
- Multi-Task Learning: 99个任务 { A1 1000, A2 1000, ⋮, A100 1000 }

如果某个任务1000. 其它任务合计有远超1000的数据. 那么, 其它任务可帮你在最后的任务表现的更好.

⇒ Can train a big enough NN to do well on all tasks.
很大的NN使所有任务都有高准确度.
为每个任务单独训练一个NN.
(如果NN不够多大. 会损害准确度.)

总结: 多任务学习应用的比迁移学习少得多.
多任务学习可训练一个NN来完成多任务.
这可使得性能比单独执行任务要好.

- 多任务学习
- 迁移学习: 数据少. 因此应用多.

(例外: 计算机视觉物体检测).
实际上迁移学习应用更多.

四. End-to-End Deep Learning. (端到端深度学习)

1. What is end-to-end deep learning.

一些数据处理系统. 或者是由多个阶段组成的学习系统.

通常可将其替代为单个NN

① Pipeline:

x —MFCC→ Feature —ML→ phonemes → words → y

audio transcript.

② End-to-End:

audio ——————————→ transcript.

End-to-End Deep Learning: 从最新的训练数据中, 直接学习 $x \to y$ 的映射.

(需要大量数据. 才有好结果)

当数据少时, 传统的 pipeline 同样有效.

③ audio - - - → phonemes → - - - transcript.

不彻底的端到端的学习.

④ Face Recognition.

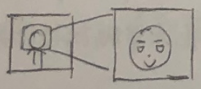
Image (x)

→ Identity (y)

函数映射. (一个模型同时学习所有步骤).

但当人位置改变时. 不再适用.

↓ 只可获取少许 (x, y).



→ Identity

将图片放入NN.

放大该人的脸部.

进行识别.

可得到许多 (x, y) 标签.

⑤ More examples.

- Machine Translation: English → Text Analysis → ··· → French.

  (x, y) 可得大量数据. English → - - - - → French.

  English French

- Estimating Child's Age: Image → Bones → Age

  Image → - - - → Age. (不可行)

2. Whether to use end-to-end learning.

① Pros and Cons of end-to-end learning.

Pros:
- Let the data speak.
- less Hand-designing of components needed

Cons:
- may need large amount of data.
- Exclude potentially useful hand-designed components.

② Applying End-to-End Deep Learning.

Key Question: Do you have sufficient data to learn a function of the complexity needed to map x to y.

CNN Name: Xiaofan Wang

Ch 1 Foundations of Convolutional Neural Network

一. Computer Vision

1. Image Classification: Cat (0/1)
   - Object Detection
   - Neural Style Transfer: Figure mixed with picture.

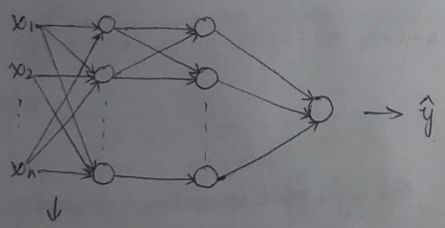
2. One Challenge in CV: inputs can get really big.

eg. picture: 64 × 64 × 3 (3 color channels) RGB

the input features has dimension 12288

eg. Larger Image: 1000 pixel × 1000 pixel × 3

3 million input features → 3 million dimensional



1st hidden layer: 1000 hidden units.

Total weights: Matrix $\vec{W_1}$ (1000, 3 million)

If use standard/fully connected Network

~~3 billion~~ 3 billion parameters: very large

Problems:
- difficult to prevent overfitting
- computation. Memory: infeasible

Solution: better implement convolution operation.

更好地实现卷积运算

(CNN的基本构建块)

二. Edge Detection
- Vertical Edge
- Horizontal Edge

1. Vertical Edge Detector

$$\begin{bmatrix} 3 & 0 & 1^{-1} & \cdots \\ 1 & 5 & 8^{-1} & \cdots \\ 2 & 7 & 2^{-1} & \cdots \\ 0 & 1 & 3 & \cdots \\ 4 & 2 & 1 & \cdots \\ 2 & 4 & 5 & \cdots \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = 4 \times 4 \text{ Matrix}$$

(Edge Detector)

卷积运算符

filter (kernel)

6 × 6 (Image)

~~Another~~ Image. Another

Function: conv-forward
- conv2d
- tf.nn.conv2d

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ \cdots & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{bmatrix}$$

6 × 6

(找到亮暗过渡边缘)

2. Vertical and Horizontal Edge Detection

1	0	-1
1	0	-1
1	0	-1

Vertical

rotate 90°

1	1	1
0	0	0
-1	-1	-1

Horizontal

3. different Filter to detect edges

1	0	-1
2	0	-2
1	0	-1

Sobel Filter

增重中间行

(more Stable)

3	0	-3
10	0	-10
3	0	-3

Scharr Filter

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

0°
90°
45°
70°
Any Edge

Backforward 9 Parameters

6×6 * 3×3 → 4×4

Image (conv) Filter (Good) 边缘检测器

三. Padding (对基本卷积操作的改进)

6×6 * 3×3 → 4×4

$n\times n$ $f\times f$ $(n-f+1)\times(n-f+1)$

Problems: 图像变小.

图像边缘丢失大量信息.

填充像素:

$\to p=2$ * 3×3 → 6×6.

$f\times f$

$6\times6 \to 8\times8$

$n\times n$ $(n+2p-f+1)\times(n+2p-f+1)$

填充多少像素? Valid and Same Convolutions

- Valid: No padding ($p=0$) $n\times n$ * $f\times f$ → $(n-f+1)\times(n-f+1)$; 6×6 * 3×3 → 4×4
- Same: Output Size = Input Size $(n+2p-f+1)\times(n+2p-f+1)$

$p=\frac{f-1}{2}$ $(n+2p-f+1=n)$

四. Strided Convolutions

$n\times n$ Image * $f\times f$ filter → $\left\lfloor \frac{n+2p-f}{s}+1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s}+1 \right\rfloor$

padding p Stride s

$\lfloor z \rfloor$ = floor(z)

分子分母不可整除时, 向下取整.

五. Cross-correlation 交叉相关 (卷积操作)

Filter:

3	4	5
1	0	2
-1	9	7

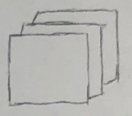
7	2	5
9	0	4
-1	1	3

$(A*B)*C = A*(B*C)$

In Math: Associativity.

六. Convolutions Over Volume

1. Convolutions on RGB Images



$6\times6\times3$ * $3\times3\times3$ → 4×4

Height × width × channel (RGB 3个通道分别有 Filter)

27 parameters

2. Multiple Filters

$6\times6\times3$ * (Vertical Edge) $3\times3\times3$ $=4\times4$

* (Horizontal Edge) $3\times3\times3$ $=4\times4$

$4\times4\times2$

$n\times n\times n_c$ * $f\times f\times n_c$ → $(n-f+1)\times(n-f+1)\times n_c'$

n_c' = # Filters

七. One layer of a convolutional Network

$6\times6\times3$ $a^{[0]}$ * $3\times3\times3$ → $4\times4+b_1$ → 4×4

* $3\times3\times3$ → $4\times4+b_2$ → 4×4

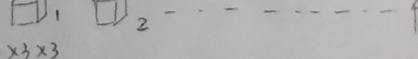
$w^{[1]}$ (ReLU)

$w^{[1]}\cdot a^{[0]} = z^{[1]}$

$$\begin{cases} z^{[1]} = w^{[1]}\cdot a^{[0]} + b^{[1]} \\ a^{[1]} = g(z^{[1]}) \end{cases}$$

Number of parameters in one layer

If you have 10 filters that are $3\times3\times3$ in one layer of a neural network, how many parameters does the layer have?



$3\times3\times3$ 1, 2, ..., 10 (不易过拟合)

27 parameters + bias = 28 parameters

280 parameters

(无论输入图像多大, 参数数量固定是280, 可用这10个Filter来检测各种特征. 非常大的图像只是很少的参数).

If layer l is a convolution layer :

$f^{[l]}$: filter size.

$p^{[l]}$: padding

$s^{[l]}$: stride.

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

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

Output : $n_H^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

$$n_w^{[l]} = \left[\frac{n_w^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right]$$

Each Filter is : $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$

Activations : $a^{[l]} \rightarrow n_H^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

$A^{[l]} \rightarrow m \times n_H^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

$n_H \times n_w \times n_c$

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

Bias : $n_c^{[l]} - (1, 1, 1, n_c^{[l]})$

Why Convolutions ?

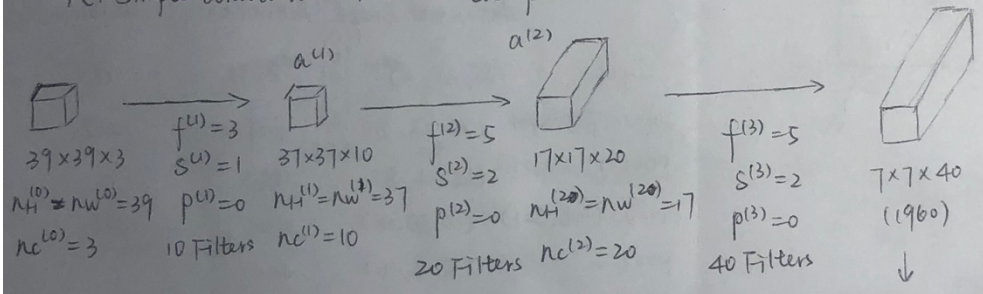
1. Parameter Sharing.

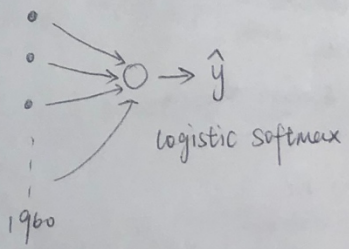
A feature detector (e.g. vertical edge detector) that's useful in one part of the image is probably useful in another part of the image.

2. Sparity of Connections

In each layer, each output value depends only on a small number of inputs.

N. Simple Convolutional Network Example.





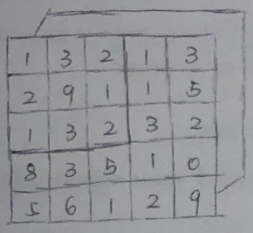
Types of layer in a convolutional Network :

- Convolution (Conv)
- pooling (Pool)
- Fully-Connected (FC).

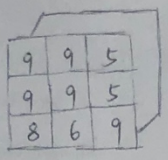
(使用反向传播来训练这些网络).

10. Pooling Layers

1. Max Pooling



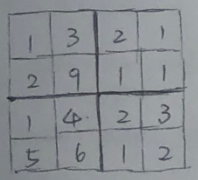
5×5×2.

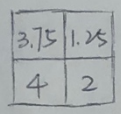


3×3×2.

$\frac{f=3}{s=1} = 3 \cdot \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$

2. Average Pooling





3. Summary of Pooling

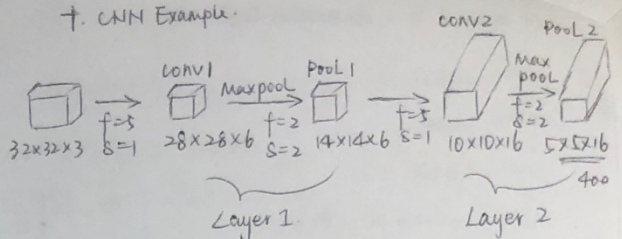
Hyperparameters:

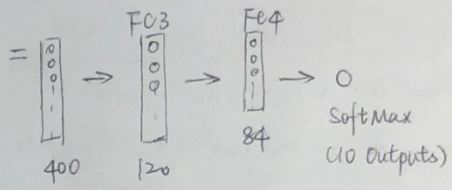
- f: Filter Size. { f=2, s=2 ; f=3, s=2 }
- s: Stride.
- Max or Average Pooling

$n_H \times n_w \times n_c$

$\left\lfloor \frac{n_H - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n_w - f}{s} + 1 \right\rfloor \times n_c$

十. CNN Example.





$W^{[1]} = (120, 400)$

$b^{[1]} = 120$

深入下去，n_H、n_w会变小，n_c会变大。

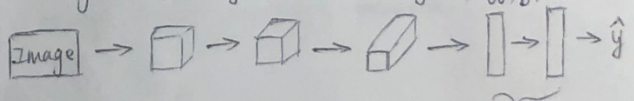
	Activation Shape	Activation Size	# Parameters
Input:	(32, 32, 3)	3072	0
CONV1 (f=5, S=1)	(28, 28, 8)	6272	208
POOL 1	(14, 14, 8)	1568	0
CONV2 (f=5, S=1)	(10, 10, 16)	1600	416
POOL 2	(5, 5, 16)	400	0
FC3	(120, 1)	120	48001
FC4	(84, 1)	84	10081
SoftMax	(10, 1)	10	841

多看些实例：如何用构件来创造高效的神经网络

如何组合。

Putting it Together:

Training Set $(x^{(1)}, y^{(1)}) \cdots (x^{(m)}, y^{(m)})$



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

Use gradient descent to optimize parameters to reduce J

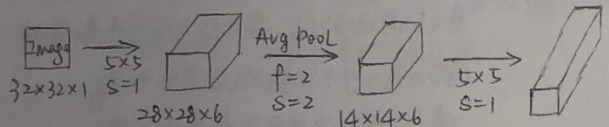
Ch 2 Deep Convolutional Models (Case studies)

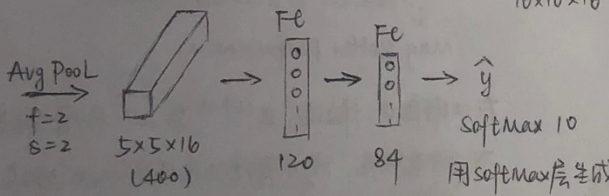
Outline:

- Classic Networks: LeNet-5, AlexNet, VGG
- ResNet
- Inception

一. Classic Networks

1. LeNet-5. (小型神经网络)





用softMax层生成10个分类结果.

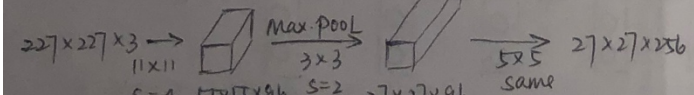
$n_H, n_W \downarrow$. $n_C \uparrow$

CONV pool. conv pool. fc fc output.

为了节省计算量和参数数量: 不同的 Filter 处理不同 Channel.

这种方法很复杂, 现在不再使用.

2. AlexNet (有点像 LeNet-5. 但更大~)



Max Pool

→ 13×13×256 → 13×13×384 → 13×13×384

3×3 S=2 | 3×3 same | 3×3

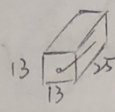
→ 13×13×256 → Max Pool → 6×6×256 = (9216) → FC 9216 → FC 4096 → FC 4096 → SoftMax 1000

3×3 | 3×3 S=2

① AlexNet: Similar to LeNet-5 but much Larger.
有更多隐藏神经元. 在更多数据上训练.
⇒ better performance.

② ReLU 激活函数

不太用
③ Multiple GPU.
④ Local Response Normalization (LRN)
局部响应归一化: 一个位置的高和宽. 然后看跨越通道的这个位置. 然后归一化.
(针对处理每个位置.)



3. VGG-16. (非常大的神经网络. 很简洁)

CONV = 3×3 Filter, S=1, same.

224×224×3 → CONV 64 ×2 → 224×224×64 → POOL → 112×112×64

→ CONV 128 ×2 → 112×112×128 → POOL → 56×56×128.

→ CONV 256 ×3 → 56×56×256 → POOL → 28×28×256.

→ CONV 512 ×3 → 28×28×512 → POOL → 14×14×512

→ CONV 512 ×3 → 14×14×512 → POOL → 7×7×512

→ FC 4096 → FC 4096 → SoftMax 1000.

$n_H, n_W \downarrow$. $n_C \uparrow$. 这种pattern减少或增加很有条理.

二. Residual Networks (ResNets)

太深的NN训练起来很难，因为"梯度消失""爆炸"...

↓

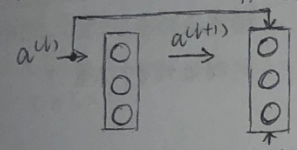
Skip Connection：从一层中得到激活 → 突然迅速给下一层 → deeper layer (并且是同时进行)

↓

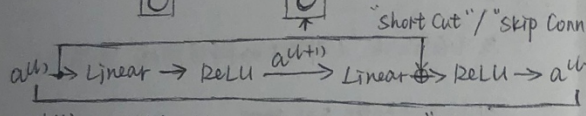
训练很深很深的残差网络 (ResNets)

1. Residual Block

快捷路径是加在ReLU非线性之前的。



(残差块)



$z^{(l+1)} = w^{(l+1)} \cdot a^{(l)} + b^{(l+1)}$

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

$z^{(l+2)} = w^{(l+2)} \cdot a^{(l+1)} + b^{(l+2)}$

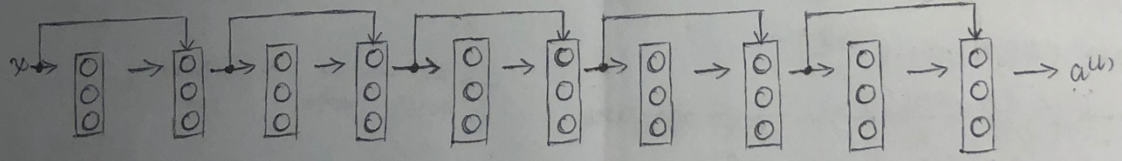
$a^{(l+2)} = g(z^{(l+2)})$

Skip connection：从一层把两层后层传递到更深的神经网络中去

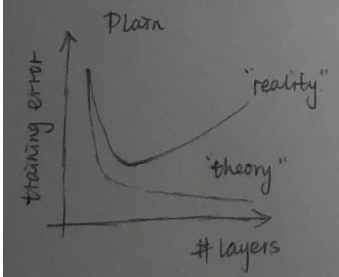
使用残差块可以训练更深层的神经网络

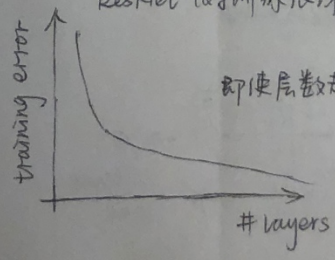
建立一个ResNet的方法：通过大量的残差块，将它们堆叠起来，形成一个深层网络 (plain Network)

2. Residual Network



5个残差块堆积在一起，⇒ 形成一个残差网络

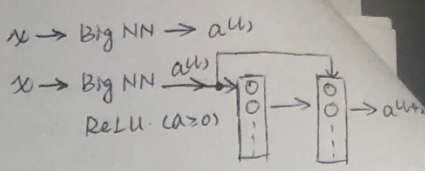




即使层数越来越深，仍可让 training error ↓

(不会有性能下降退的现象，即使算点会达到平原阶段，这时再加层也不会有太大帮助)

三. Why ResNets Work



$a^{(l+2)} = g(z^{(l+2)} + a^{(l)})$

$= g(w^{(l+2)} \cdot a^{(l+1)} + b^{(l+2)} + a^{(l)})$

($w^{(l+2)} \cdot a^{(l+1)} \approx 0$，$b^{(l+2)} \approx 0$)

$= g(a^{(l)}) = a^{(l)}$

中间额外添加两层，所有项都学到有意义的东西的话，要比学习恒等函数要好。

(额外层几乎不会影响总体的表现)

梯度下降从这里可以靠最终结果)

↓

May Better performance.

在此维度中，假设 $z^{(l+2)}$ 与 $a^{(l)}$ 在同一维度，有一堆卷积层，偶尔有池化层或类似池化的层，可用 Ws 调整维度。一个全连接的层来最终用 softmax 做预测

$z^{(l+2)} + a^{(l)}$ 3×3 same pool Ws

↓

FC. Softmax 1000.

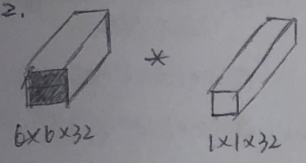
四. Network In Network and 1×1 convolutions.

1. Simple:

$6\times6\times1 \ * \ \boxed{2} \ = \ 6\times6\times1$

(just multiply by 2)

2.



$= 6\times6\times$ #Filters.

$n_c^{[l+1]}$

将 Left 32 × Filter 32 (1内核).

再将一个非线性映射 ReLU作用于它.

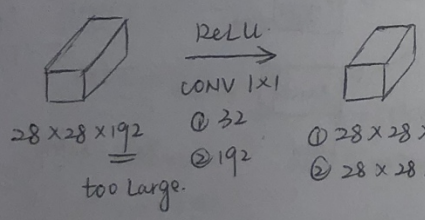
类似于：已有一个神经元，接收一个32个数的输入向量.

Output：过滤器数个输出值.

⇓

1×1 conv (Network In Network)

3. n_c↓.



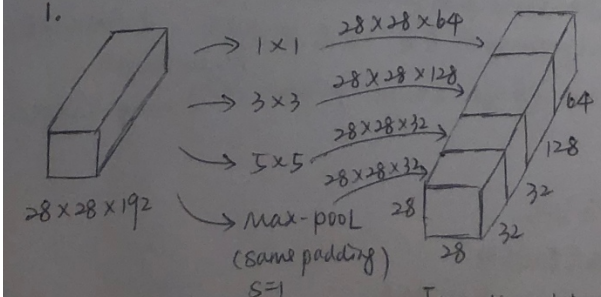
① CONV 1×1：decrease ~~the~~ n_c ⇒ 减小计算量.

pool Layer：decrease n_H. n_W.

② CONV 1×1：增加非线性

五. Inception Network Motivation.

1.

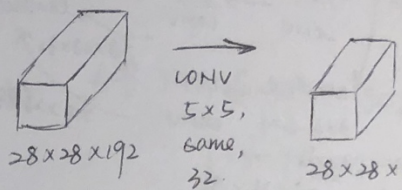


[Inception Network的核心]

Inception Network基础特点：①不用为选一个卷积核的大小或 pooling.

②所有可能都做. 将所有输出结果都连起来.

2. Problem of Computational Cost.



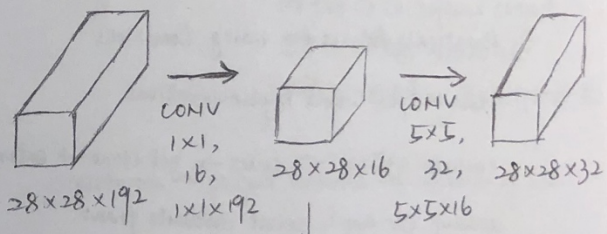
$28\times28\times32 \ \times \ 5\times5\times192 = 120$ Million

(1.2 Billion).

Too Large Comp~Cost.

⇓

1×1 CONV：decrease the Computational Cost $\frac{1}{10}$~



$28\times28\times192\times16 = 2.4$M

瓶颈层.

$28\times28\times32\times5\times5\times16$

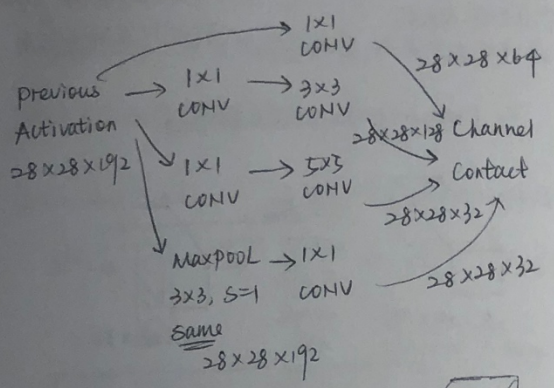
= 1000万

1.2 Billion VS. 1240万次乘法.

(合理实现瓶颈层. 又不影响整体性能).

六. Inception Network. (重复多次使用 Inception Module).

Inception Module:

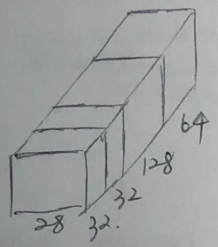


(28×28×256)

旁枝: FC. 将隐藏层作为输入来预测 (Normalization)

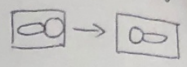
↓

prevent Overfitting.



3. Data Augmentation. (Adjust hyperpara

(1) Common Augmentation Method: ① Mirroring (垂直镜)

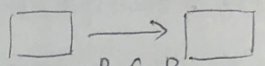


② random Cropping. (随机剪裁)

③ rotation Shearing (旋转)

④ Local warping (局部弯曲)

(2) Color Shifting.



R. G. B.

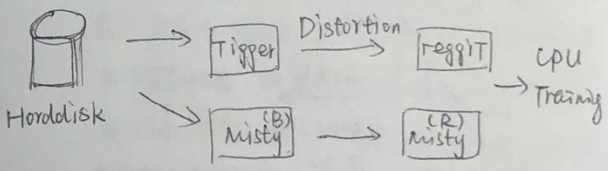
+20. -20. +20. (不同的扰动)

-20. +20. +20

+5. 0. +50

(3) Implement Distortions during Training.

(CPU Thread.)



t. Practical Advice for using ConvNets.

1. Using Open-Source implementations.

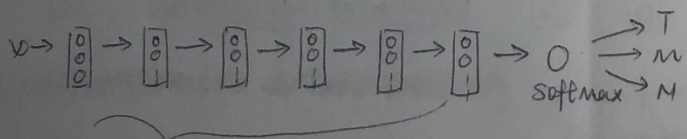
resnets github. → codes → git clone. + enter.

develop CV App: select favourite frame

look for open-source with codes

(Traing too long time. & many GPU. Dataset. → Transfer Learning Directly).

2. Transfer Learning. (Small Dataset. → useful -)



freeze. freeze=1

training parameter=0.

Using Dataset. = Initialize Network. (Weights)

↓ Gradient.

renew weights.

& Network Layers.

freeze. train. { Gradient. Delete.

数据集越多

冻结层数越少.

自己training Layer 越多

Ch 3. Detection Algorithms.

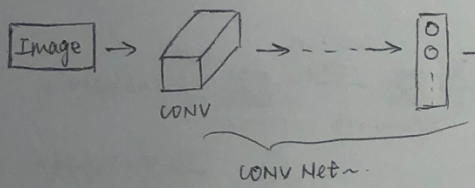
一、Object Localization.

1. Localization and Detection.

- Image Classification
- Classification with Localization } [commonly] 1 Object. (识别正中央的一个明显对象)
- Detection ⇒ Multiple Objects (识别来自不同类别的多个对象)

2. Classification with Localization

在多层卷积神经网络中输入一张图片，这会导致我们的卷积网络中有一个特征向量. 将特征向量传给一个 softMax. 输出预测结果.



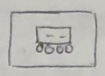
例: ① pedestrian
② car
③ motorcycle
④ background.
(分类类别)
⇒ 一个有4个结果的 SoftMax.

Need to output b_x, b_y, b_h, b_w
(class label 1-4)

Defining the Target Label y:

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \quad \begin{matrix} \\ \\ \\ \\ \\ \} \\ \\ \\ \end{matrix} \begin{matrix} \\ \\ \\ \\ \\ 0 \\ 1 \\ 0 \end{matrix}$$

↑ 损失值是元素平方差之和

$x =$ 

$$\mathcal{L}(\hat{y}, y) = \begin{cases} (\hat{y}_1 - y_1)^2 + \cdots (\hat{y}_8 - y_8)^2 & (y_1 = 1) \\ (\hat{y}_1 - y_1)^2 & (y_1 = 0) \end{cases}$$

↓ 其它元素都不重要. 只关注 Network.

$y_1 = 1$ 时, pay attention to Network ⇒ 可用均方差 penalize 预测平方偏差. 针对8个组件的数据输出

$y_1 = 0$ 时, don't care other elements.

& 也可用 Log Likelihood 特征损失

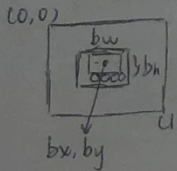
对于 c_1, c_2, c_3. Softmax output, 其中之一的元素, 可用均方差

如果 p_c. 可用类似逻辑回归损失.

Network → Output numbers → Location of Sth

powerful Idea

例:



若定位图中汽车, 可以改变 Network. 通过得到更多的输出单位, 使这个神经网络可以输出一个边框 ⇒ b_x, b_y, b_h, b_w ⇒ boundary box

(~~obj~~ extra 4 parameters)

用 supervised Learning 让算法输出的不仅仅是类标签, 还有4个变量 ⇒ boundary box.

(~~Assi~~ Estimation: $b_x = 0.5$. $b_y = 0.7$. $b_h = 0.3$. $b_w = 0.4$)

二. Landmark Detection.

有标签的训练集：

x, y { l_{1x}, l_{1y} ; l_{2x}, l_{2y} ; l_{3x}, l_{3y} ; l_{4x}, l_{4y} ; ⋮ ; l_{64x}, l_{64y} }

想要识别的特征的(x,y)标签必须在不同图片中保持一致性.

三. Object Detection. (ConvNet. Sliding-Wins)

1. Car detection example.

Training Set:

x	y
car	1
car	1
car	1
Tree	0
Mountain	0

car → ConvNet → y

2. Sliding Windows detection.

滑动小窗口，遍历Image. => ConvNet.

(Stride)

↓

将小窗口Area个. 重复遍历Image.

Algorithm: Input: Small Frame. → ConvNet.

Output: y=1 → Detect a ~~car~~ car

Disadvantage: Large Computation Cost.

每个图像都单独通过ConvNet运算

① 如果 Stride↑不密集. → Number of Frame ↓

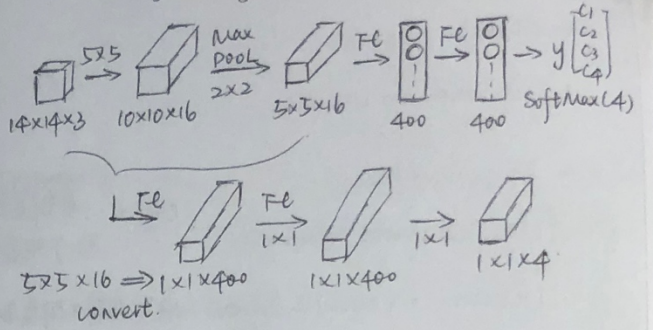
→ 粗糙的颗粒度会影响算法表现.

② 如果 Small Stride. → Number of Frame ↑

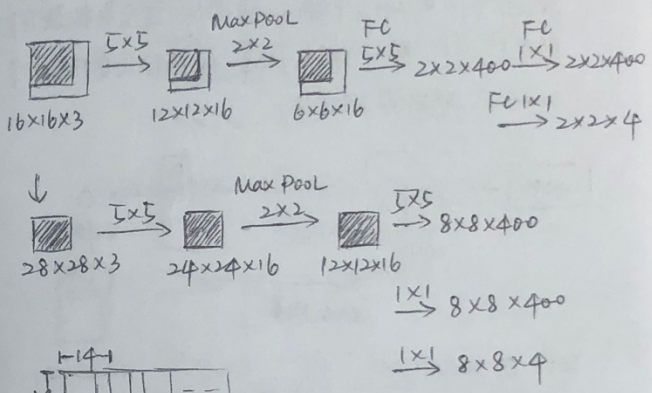
→ 精细的颗粒度 → 小区域密集 → Large Cost

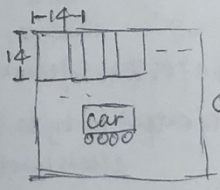
四. Convolutional Implementation of Sliding Windows

1. Turning FC Layer into Convolutional Layers.



2. Convolution Implementation of Sliding Windows.





① Slide Until Indentifying a Car.

(Output/Slide by order)

✓ ② ConvNet → Forward propagation.

(Better) → y predict.

高效许多. (但边框定位每不精确)

Problem.

五. Bounding Box Predictions.

(Solution for Sliding-Wins Algorithms.

More accurate to predict Boundary Frame)

car 不一定是"方框".

"方框"不一定总是准确框住车"

1. YOLO Algorithm (论文较难懂)

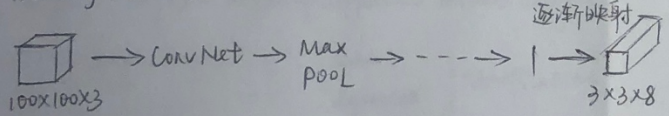
Labels for Training:

For each grid cell:

$$y: \begin{bmatrix} Pc \\ bx \\ by \\ bw \\ bh \\ C_1 \\ C_2 \\ C_3 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \quad \begin{bmatrix} 1 \\ bx \\ by \\ bw \\ bh \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

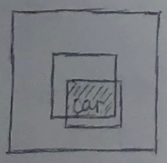
9: 3×3 (8 Dimension 个) (3×3×8)

Training Network:



六. Intersection Over Union.

1. Evaluating Object Localization.



The result is good or bad?

交并比或 IoU 函数: 两边界框交集 / 两边界框并集.

= size of ▨ / size of □

"correct" if IoU ≥ 0.5

More Generally, IoU is a measure of the overlap between two bounding boxes. IoU是两个边界框重叠程度的一个度量值.

七. Non-Max Suppression 非最大值抑制.

Algorithm may ~~has~~ has many testings for the same target.

Non-Max Suppression makes sure the algorithm has only one testing for each ~~car~~ target (Object).

运行算法时, 对每个目标, 也许会得到多个检测结果

↓

清理检测, 每辆车只会得到一个检测结果.

↓

The probability of each testing result ($P_c \cdot C_1$)

~~box~~ Select the largest

& Suppress other results

1. Non-Max: 输出有着最大可能性的分类判断

抑制那些 Non-Max 的邻近方框.

2. Non-Max Suppression Algorithm

① Each output prediction is: $\begin{bmatrix} P_c \\ bx \\ by \\ bh \\ bw \end{bmatrix}$ No $\begin{bmatrix} C_1 \\ C_2 \\ C_3 \end{bmatrix}$

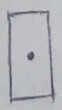
② Discard all boxes with ($P_c \leq 0.6$)

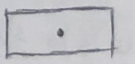
③ Remaining Boxes:

- Pick the box with the largest P_c Output that as a prediction.
- Discard any remaining box with IoU ≥ 0.5 with the box output in the previous step.

八. Anchor Box.

1. Over-lapping Objects

Anchor Box 1: 

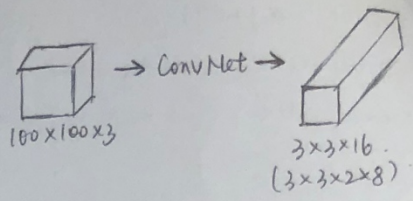
Anchor Box 2: 

$$y = \begin{bmatrix} P_c \\ bx \\ by \\ bh \\ bw \\ C_1 \\ C_2 \\ C_3 \\ P_c \\ C_3 \end{bmatrix} \begin{matrix} \} \text{ Anchor Box 1 (8 parameters)} \\ \\ \\ \\ \\ \\ \\ \\ \} \text{ Anchor Box 2. (8 parameters)} \\ \\ \end{matrix}$$

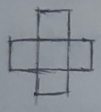
2. Anchor Box Algorithm (处理两个对象出现在同一格子的情况)

Previously: Each object in training image is assigned to grid cell that contains that object's midpoint.

将训练图像上的对象指定给对应目标中心点的网格单元. (Output y: 3×3×8)



With Two Anchor Boxes: Each object in training image is assigned to grid cell that contains object's midpoint and anchor box for the grid cell with highest IoU.



Coutput y: 3×3×16)

$$y=\begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \quad \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 1 \\ 0 \\ 0 \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Anchor Box 1

Anchor Box 2.

Outputting the non-max suppressed outputs:

① For each grid cell, get 2 predicted bounding boxes

② Get rid of low probability predictions.

③ For each class (pedestrian. car. motorcycle) use non-max suppression to generate final predictions.

3. Putting it together: YOLO Algorithm

{ pedestrian
car
motorcycle.

$$y=\begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \quad \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

① y is $3\times3\times2\times8$ = (维度.) $5+3C$.

↓

Anchor Box.

$3\times3\times16$.

② $19\times19\times16$

$19\times19\times5\times\#$ → $19\times19\times40$

↓

5 Anchor Boxes

Ch4 Face Recognition.

一. Face verification VS Face Recognition.

1. Verification
   - Input image, name/ID
   - Output whether the input image is that of the claimed person.
2. Recognition
   - Has a database of K persons
   - Get an input image
   - Output ID if the image is any of the K persons (for "not recognised")

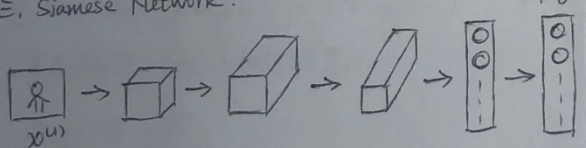
二. One-Shot Learning. 单样本学习. (输入2张图, 看"相似"or"不同")

1. Learning from one example to recognize the person again
2. Learning a "similarity" Function.

d(img1, img2) = degree of difference between images.

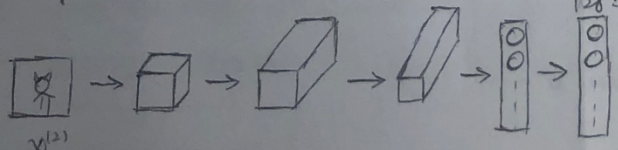
If $d(img1, img2) \leq \tau$ "same"
$> \tau$ "different" } Verification

三. Siamese Network.



专注同样本身. "128". 由 Network 深处的某个全连接层计算而来.

128: f(x1) = encoding of $x^{(1)}$



使用相同的参数, 得到一个不同的. 128个数字组成的向量.

$d(x^{(1)}, x^{(2)}) = \|f(x^{(1)}) - f(x^{(2)})\|_2^2$.

Parameters of NN define an encoding $f(x^{(i)})$

Learn parameters so that:

If $x^{(i)}, x^{(j)}$ are the same person, $\|f(x^{(i)}) - f(x^{(j)})\|^2$ is small

If $x^{(i)}, x^{(j)}$ are different, $\|f(x^{(i)}) - f(x^{(j)})\|^2$ is large.

当改变所有层的神经网络的参数. 可以使用反向传播来更改参数以满足条件

四. Triplet Loss Function (梯度下降的三元组损失函数)

1. Aim: 为了学习神经网络的参数. 并以此获得一个优良的人脸图片的编码.

2. Learning Objective:

Anchor positive — Similar. Anchor Negative

Want: $\frac{\|f(A) - f(P)\|^2}{d(A,P)} \leq \frac{\|f(A) - f(N)\|^2}{d(A,N)}$

$\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 \leq 0$

-0 -0 $+\alpha$ (margin para-)

Given 3 images A, P, N:

$\mathcal{L}(A,P,N) = \max(\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha, 0)$

>0

$J = \sum_{i=1}^{m} \mathcal{L}(A^{(i)}, P^{(i)}, N^{(i)})$ 需要一些成对的 A, P. 同一人的. (许多对)

Training set: 10k pics of 1K persons

如果每个人只有一张 pic. 无法训练

2. Face Verification Supervised Learning.

x	y	
	1	"same"
	0	"Different"
⋮	0	
Image.	1	

3. Choosing the triplets A, P, N.

During training if A, P, N are chosen randomly,

~~d~~ $d(A,P)+\alpha \le d(A,N)$ is easily satisfied.

$\|f(A)-f(P)\|^2+\alpha \le \|f(A)-f(N)\|^2$

Choose triplets that're "hard" to train on:

$d(A,P)+\alpha \le d(A,N)$

$d(A,P) \approx d(A,N)$

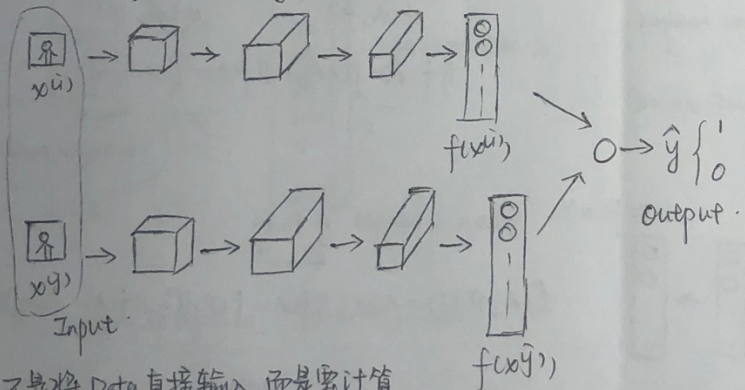
选择三元组，增强学习算法的计算效率.

4. Training Set Using Triplet Loss:

$J_{min.}\ d(x^{(i)}, x^{(j)})$

五. Face Verification and Binary Classification.

1. Learning the similarity Function



不是将 Data 直接输入，而是计算编码之间的不同.

$$\hat{y} = \sigma\left(\sum_{k=1}^{128} \| f(x^{(i)})_k - f(x^{(j)})_k \| * w_k + b\right)$$

$$\approx \frac{(f(x^{(i)})_k - f(x^{(j)})_k)^2}{f(x^{(i)})_k + f(x^{(j)})_k}$$ Chi-Square. χ^2

precompute: 无需每次都为每个数据库中的员工计算编码.

Ch 5. Neural Style Transfer.

一. Neural Transfer.

Content. Style.

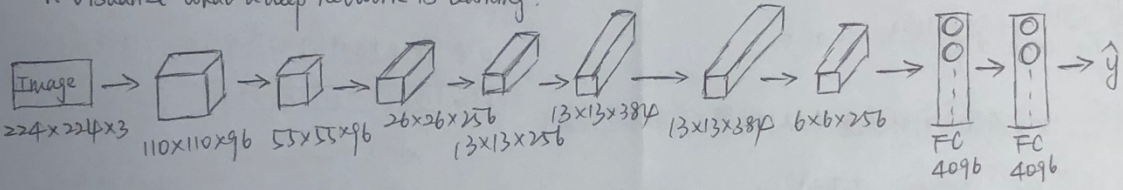
↓

Generated Image.

In order to achieve neural transfer, look at features extracted by ConvNet at various layers. the shallow and the deeper layers of a ConvNet.

二. Deep ConvNets Learning.

1. Visualize what a deep network is learning.



⇒ Pick a unit in layer 1. Find the nine image patches that maximize the unit's activation.

⇒ Repeat for other units.

2. Visualize deep layers: Layer 2. 3. ---I---

层数越深，模式成特征越复杂

Layer 1：边.

Layer 2：纹理.

Deeper Layer：very complex objects

三. Neural Style Transfer Cost Function.

Content (C) Style (S)

↓

Generated Image (G).

$$J(G) = \alpha \cdot J_{content}(C, G) + \beta \cdot J_{style}(S, G)$$

Find the generated image G:

① Initiate G randomly: G 100×100×3. RGB.

② Use gradient descent to minimize J(G).

$$G = G - \frac{\partial J(G)}{\partial G}$$ (相当于在更新图像G的像素值).

四. Content Cost Function.

$$J(G) = \alpha \cdot J_{content}(C, G) + \beta \cdot J_{style}(S, G)$$

① Use hidden layer l to compute $J_{content}$.

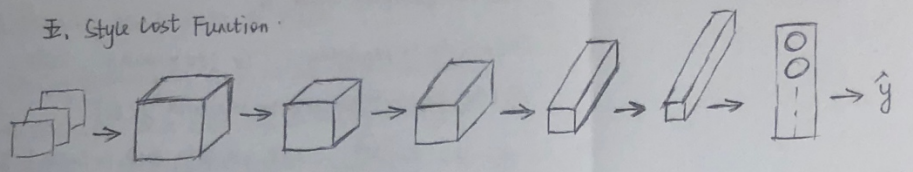
② Use pre-trained ConvNet (Eg. VGG network ···)

③ Let $a^{[l](C)}$ and $a^{[l](G)}$ be the activation of layer l on the images.

④ If $a^{[l](C)}$ and $a^{[l](G)}$ are similar, both images have similar content.

$$J_{content}(C, G) = \frac{1}{2} \| a^{[l](C)} - a^{[l](G)} \|^2$$

五. Style Cost Function.



Use layer l's activation to measure "style".

Define style as correlation between activations across channels.

Style Matrix.

$a^{[l]}_{i,j,k}$ = activation at (i, j, k). (i: H, j: W, k: C)

$G^{[l]}$ is $n_c^{[l]} \times n_c^{[l]}$ ↓ 记录每对通道的相关性.

$$\Rightarrow G^{[l](S)}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a^{[l](S)}_{ijk} \cdot a^{[l](S)}_{ijk'}$$

$k = 1, \cdots, n_c^{[l]}$, k 遍历所有的通道. (从1到第 l 层的总通道数)

$$\Rightarrow G^{[l](G)}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a^{[l](G)}_{ijk} \cdot a^{[l](G)}_{ijk'}$$ "Gram matrix"

$$\beta \cdot J^{[l]}_{style}(S, G) = \frac{1}{(\cdots)} \left\| G^{[l](S)} - G^{[l](G)} \right\|_F^2$$

$$= \frac{1}{(2 n_W^{[l]} n_H^{[l]} n_c^{[l]})^2} \cdot \sum_k \sum_{k'} \left(G^{[l](S)}_{kk'} - G^{[l](G)}_{kk'} \right)^2$$

$$J_{style}(S, G) = \sum_l \lambda^{[l]} J^{[l]}_{style}(S, G)$$

$$\Downarrow$$

$$J(G) = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$$

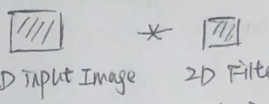
总代价函数 = 内容代价函数 + 风格代价函数,

然后使用梯度下降法. 或采用更复杂的优化算法

来试图寻找 Image G $\Rightarrow$ $J(G)$ min. $\Rightarrow$ 神经网络艺术作品.

六. 1D. 3D generalizations of models.

一. Convolutions in 2D and 1D. Matrix. List. (2D → 1D).

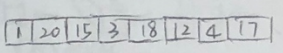


2D Input Image 14×14 * 2D Filter 5×5

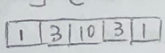
$14 \times 14 \times 3$ * $5 \times 5 \times 3$ ↓ $10 \times 10 \times 16$ * $5 \times 5 \times 16$ ↓ $6 \times 6 \times 32$.

例:

1D 电图. (按时间顺序量测每一以电压序列)



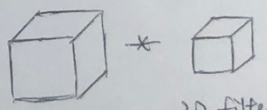
*



14×1 * 5×1 ⇓ 10×16 * 5×16 ⇓ 6×32.

用同样的一组特征去检测在这些时间序列中不同位置的心跳. ⇒ ConvNet 可用于 1D Data. (递归神经网络).

二. 3D convolution.



$14 \times 14 \times 14 \times 1$ * $5 \times 5 \times 5 \times 1$

$\Rightarrow 10 \times 10 \times 10 \times 16$ * $5 \times 5 \times 5 \times 16$ (16 filters).

$\Rightarrow 6 \times 6 \times 6 \times 32$. (32 filters)

Name: Xiaofan Wang

Sequence Models.

Chapter 1. Recurrent Neural Network.

一. Sequence Models.

1. Example of Sequence Data.

- Speech Recognition.
- Music Generation
- Sentiment Classification.
- DNA Sequence Analysis
- Machine Translation
- Video Activity Recognition.
- Name Entity Recognition.

2. Notation.

x: Harry Potter and Hermione Granger invented a new spell.

$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ ... $x^{<t>}$ ... $x^{<9>}$ (让标代表单词位置)

⇒ y: 1 1 0 1 1 0 0 0 0

$y^{<1>}$ $y^{<2>}$ ... $y^{<t>}$ ... $y^{<9>}$

$\begin{cases} T_x = 9 \text{ (序列长度)} \\ T_y = 9 \end{cases}$

$x^{(i)<t>}$: i 的序列中的第 t 个元素.

$y^{(i)<t>}$

$\begin{cases} T_x^{(i)} = 9 \text{ (序列长度)} \\ T_y^{(i)} = 9 \end{cases}$

3. Representing Words

x: Harry Potter and Hermione Granger invented a new spell.

$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ .... $x^{<t>}$ ... $x^{<9>}$

Vocabulary. 将每个词作一个 "one-hot" 的表示

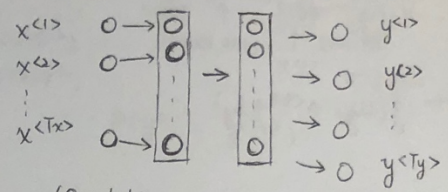
a	1
Aaron	2
⋮	⋮
And	367
⋮	⋮
Harry	4075
⋮	⋮
Potter	6830
⋮	⋮
zulu	10000 (将1万词作为词汇构建字典)

$x^{<1>}$: 4075 位为 1

$x^{<2>}$: 6830 位为 1

二. Recurrent Neural Network Model.

1. Why not a standard network.



(One-hot)

将 9 个 Input Words 放入标准的神经网络中.

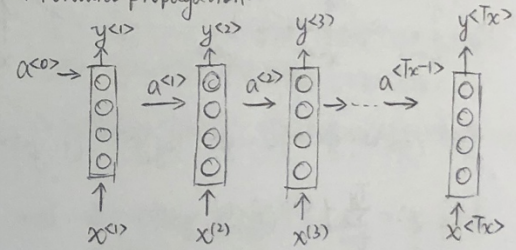
然后输出 9 个 Output Words. { 0 / 1 ⇒ 每个单词是否是人名的一部分.

Problems:

① Inputs. Outputs can be different lengths in different examples.

② Don't share features learned across different positions of text. 像这种朴素的 NN, 并不会共享那些从不同文本位置学到的特征.

2. Forward Propagation.



$a^{<0>} = \vec{0}$

$a^{<1>} = g_1(W_{aa} a^{<0>} + W_{ax} \cdot x^{<1>} + b_a)$ ← tanh/ReLu

$\hat{y}^{<1>} = g_2(W_{ya} a^{<1>} + b_y)$ ← sigmoid.

$a^{<t>} = g(W_{aa} \cdot a^{<t-1>} + W_{ax} \cdot x^{<t>} + b_a)$

$\hat{y}^{<t>} = g(W_{ya} a^{<t>} + b_y)$

3. Simplified RNN Notation

$$a^{<t>} = g(W_{aa} \cdot a^{<t-1>} + W_{ax} \cdot x^{<t>} + b_a)$$

(W_{aa}: (100, 100), $a^{<t-1>}$: 100, W_{ax}: (100, 10000), $x^{<t>}$: 10000)

$$\hat{y}^{<t>} = g(W_{ya} \cdot a^{<t>} + b_y)$$

$$\Downarrow$$

$$\hat{y}^{<t>} = g(W_y \cdot a^{<t>} + b_y)$$

这里的 W_a, b_a 用于计算的参数如，激活输出的值

$$\Rightarrow a^{<t>} = g(W_a [a^{<t-1>}, x^{<t>}] + b_a)$$

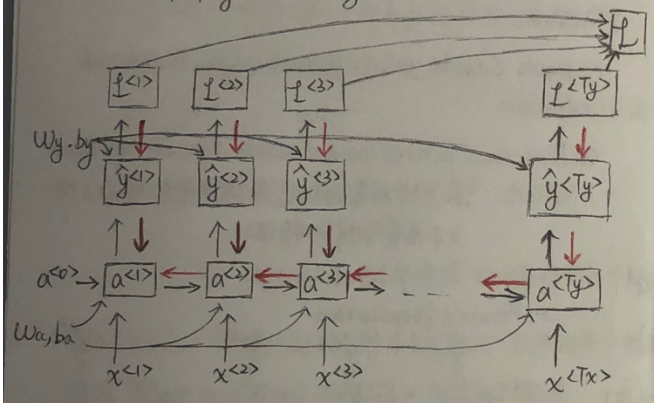
$$W_a = [W_{aa} | W_{ax}] \updownarrow 100$$

(100, 10000) $\leftrightarrow$ 100 $\leftrightarrow$ 10000

$$[a^{<t-1>}, x^{<t>}] = \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} \begin{matrix} \updownarrow 100 \\ \updownarrow 10000 \end{matrix} \quad \updownarrow 10100$$

$$[W_{aa} | W_{ax}] \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} = W_{aa} \cdot a^{<t-1>} + W_{ax} \cdot x^{<t>}$$

4. Backpropagation through time.



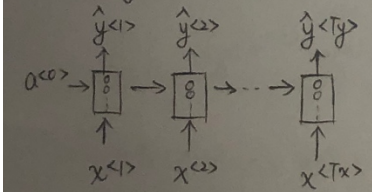
$$\mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>}) = -y^{<t>} \log \hat{y}^{<t>} - (1 - y^{<t>}) \log (1 - \hat{y}^{<t>})$$

$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{Ty} \mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>})$$

5 Different Types of RNN.

Examples of RNN Architectures:

① $Tx = Ty$.

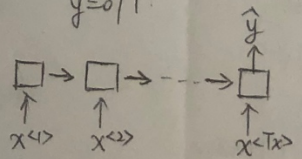


Many-to-Many

② Sentiment Classification

x = test

y = 0/1

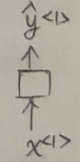


Many-To-One.

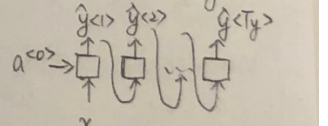
(输入：许多单词

输出：一个数字)

③ One to One.

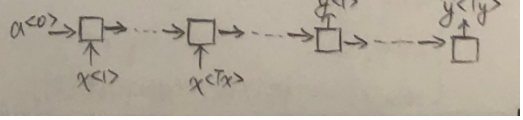


④ One To Many.



(Music Generation)

⑤ Many TO Many. ($Tx \neq Ty$. Machine Translation)



三. Language Model and Sequence Generation.

1. Language Modeling.

Speech Recognition: { The apple and pair salad. ($P_1 = 3.2 \times 10^{-13}$)
The apple and pear salad. ✓ ($P_2 = 5.7 \times 10^{-10}$)

语音识别系统选出第二个句子.
由于语言模型 (Language Model).
给出的两个句子的概率.

2. Language Modelling with an RNN

Training Set: Large corpus of English text. 单词大文集.

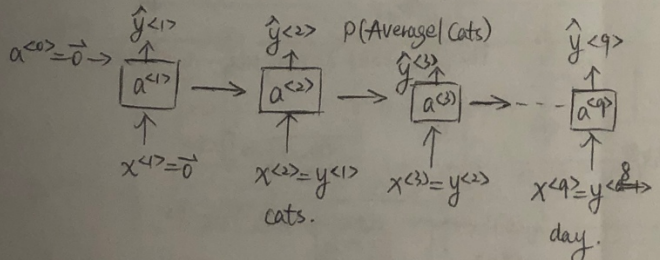
例: Cats average 15 hours of sleep a day. <EOS>.
$y^{<1>}\ y^{<2>}\ ---\ y^{<9>}$
句子结束标记.

The Egyptian Mau is a bread of cat. <EOS>
<UNK> 未知单词
对 UNK 的概率建模, 而不是对特定未知词建模.

完成标记化: 将输入句子映射成各个标记, 或单词表中单词集合.
然后, 构建 RNN 来为这些不同序列的概率建模.

3. RNN Model (从左至右每次预测一个单词的概率).

P(a). P(Aaron) --- P(Zulu) P(<UNK>) P(<EOS>).



$\mathcal{L}(\hat{y}^{<t>}, y^{<t>}) = -\sum_i y_i^{<t>} \log \hat{y}_i^{<t>}$

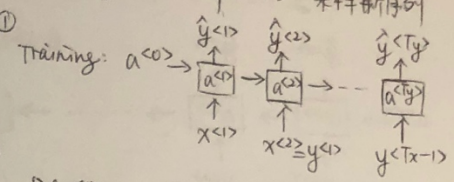
为了训练这个神经网络, 要定义一个代价函数.

$\mathcal{L} = \sum_t \mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>})$

整个句子的概率分布: $P(y^{<1>}, y^{<2>}, y^{<3>}) = P(y^{<1>}) P(y^{<2>} | y^{<1>}) \cdot P(y^{<3>} | y^{<1>}, y^{<2>})$

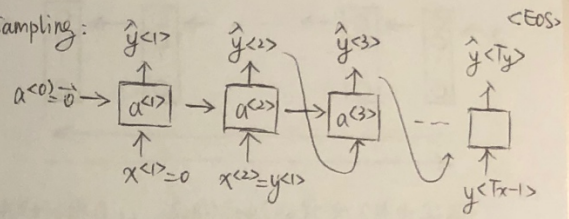
4. Sampling Novel Sequences 采样新序列

①



$P(y^{<1>}, --y^{<Tx>})$

Sampling:



以上, 构建了一个字级 RNN (用的方法).

<UNK>

② Character-Level Language Model 字符 RNN

Vocabulary = [a, aaron, ..., zulu, <UNK>]

Vocabulary = [a, b, c---, z, ⊔, ., ,, ;, 0---9---A---Z]

优点: 某 "Mau" 未知单词, 可用单个字符表示概率.
缺点: 序列更长. 计算更复杂 (不如字级 RNN)

⇒ 字符 RNN 用的较少. (特殊情况下使用: 许多未知词 ---)

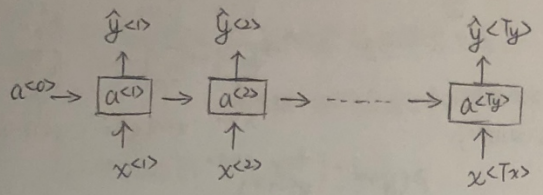
小结: 建立一个 RNN 来观察英文文本.
构建单词级别和字符级别的模型.
在你培训的语言模型中采样.

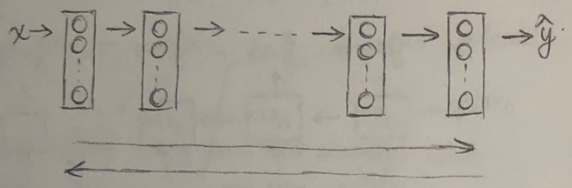
语言模型中的文本示例: News | Shakespeare.
(有文化级别的语言模型)

⇒ 如果该模型是在新闻文章中受训的, 它会生成左边文本.

⇒ 如果该模型在 Shakespeare 上训练的, 生成右 ~

5. Vanishing Gradients with RNNs.





假设这是个很深的神经网络，从 $\hat{y}$ 得到梯度很难反向传播去影响前期层的权重.

RNN：也存在类似问题.（很难由前面单词 ⇒ 判断：单复数）

- The cat which ---, was full.
- The cats which ---, were full.

RNN 模型：有附近效应.（output $\hat{y}$ 主要受接近 $\hat{y}$ 值的影响）

这里的 Output $\hat{y}$ 很难被 Input x 影响到. 因为 ~~Both~~

Input x 是非常早期的序列. 所以无论 Output $\hat{y}$ 是多.

都不可通过反向传播到序列的开始. 从而去修改 NN 的前期序列.

RNN：不擅长捕获远程依赖关系.

NaN：梯度爆炸.

Solution：Gradient Clipping

梯度向量如果大于某个临界值，要重新缩放某些向量. 使其不那么大.

Γ_u 门控很容易设为 0. 只要 "$w_u[\ ]+b_u$" 是很大的负值.

在这附近的值 更新门控值基本上为 0.（非常接近 0）

$\Gamma_u = 0.000001$ 不受梯度弥散. 因为 $\Gamma_u \approx 0$ 时，$c^{<t>} = c^{<t-1>}$

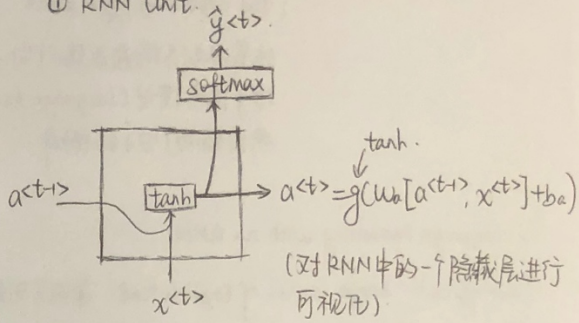
~~时~~. c 可保持很久不变. 这可显著帮助梯度弥散 ~ 允许 NN 有很长范围的依赖性.

$\tilde{c}^{<t>}$ 100 维 ⇒ Γ_u 100 维. 近似取 0、1

6. Gated Recurrent Unit.（GRU）门控循环单元.

GRU 修改了 RNN 的隐藏层. 从而更好地捕捉长距离关系，同时有助于减少梯度消失.

① RNN Unit



（对 RNN 中的一个隐藏层进行可视化）

② GRU

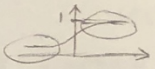
c = memory cell

$c^{<t>} = a^{<t>}$

$\Rightarrow \tilde{c}^{<t>} = \tanh(W_c[c^{<t-1>}, x^{<t>}] + b_c)$

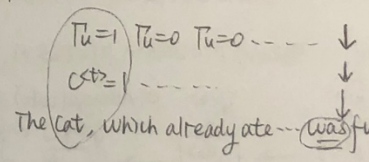
$\Rightarrow \Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$

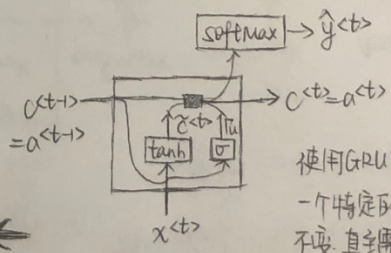
（Gamma Update）



$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1-\Gamma_u) * c^{<t-1>}$

element-wise（元素乘法）





使用 GRU 的好处：何时更新一个特定的内存单元，然后保持不变. 直到需要时，在早些被设置的值也可从该记忆单元取出.

③ Full GRU.

$\tilde{c}^{\langle t\rangle} = \tanh(W_c[\Gamma_r * c^{\langle t-1\rangle}, x^{\langle t\rangle}] + b_c)$. 计算记忆单元的候选新值.

Γ_r 相关性. 如何通过 c 来计算下一个 c.

$\Gamma_u = \sigma(W_u[c^{\langle t-1\rangle}, x^{\langle t\rangle}] + b_u)$.

$\Gamma_r = \sigma(W_r[c^{\langle t-1\rangle}, x^{\langle t\rangle}] + b_r)$.

$c^{\langle t\rangle} = \Gamma_u * \tilde{c}^{\langle t\rangle} + (1-\Gamma_u) * c^{\langle t-1\rangle}$

7. LSTM (long Short Term Memory) Unit.

① GRU and LSTM

GRU (更简单, 可扩展性有利于构建较大的模型)

$\tilde{c}^{\langle t\rangle} = \tanh(W_c[\Gamma_r * c^{\langle t-1\rangle}, x^{\langle t\rangle}] + b_c)$.

$\Gamma_u = \sigma(W_u[c^{\langle t-1\rangle}, x^{\langle t\rangle}] + b_u)$

$\Gamma_r = \sigma(W_r[c^{\langle t-1\rangle}, x^{\langle t\rangle}] + b_r)$

$c^{\langle t\rangle} = \Gamma_u * \tilde{c}^{\langle t\rangle} + (1-\Gamma_u) * c^{\langle t-1\rangle}$

$a^{\langle t\rangle} = c^{\langle t\rangle}$.

Γ_f

LSTM (更强大和灵活, 有三个门控)

$\tilde{c}^{\langle t\rangle} = \tanh(W_c[a^{\langle t-1\rangle}, x^{\langle t\rangle}] + b_c)$

(update) $\Gamma_u = \sigma(W_u[a^{\langle t-1\rangle}, x^{\langle t\rangle}] + b_u)$.

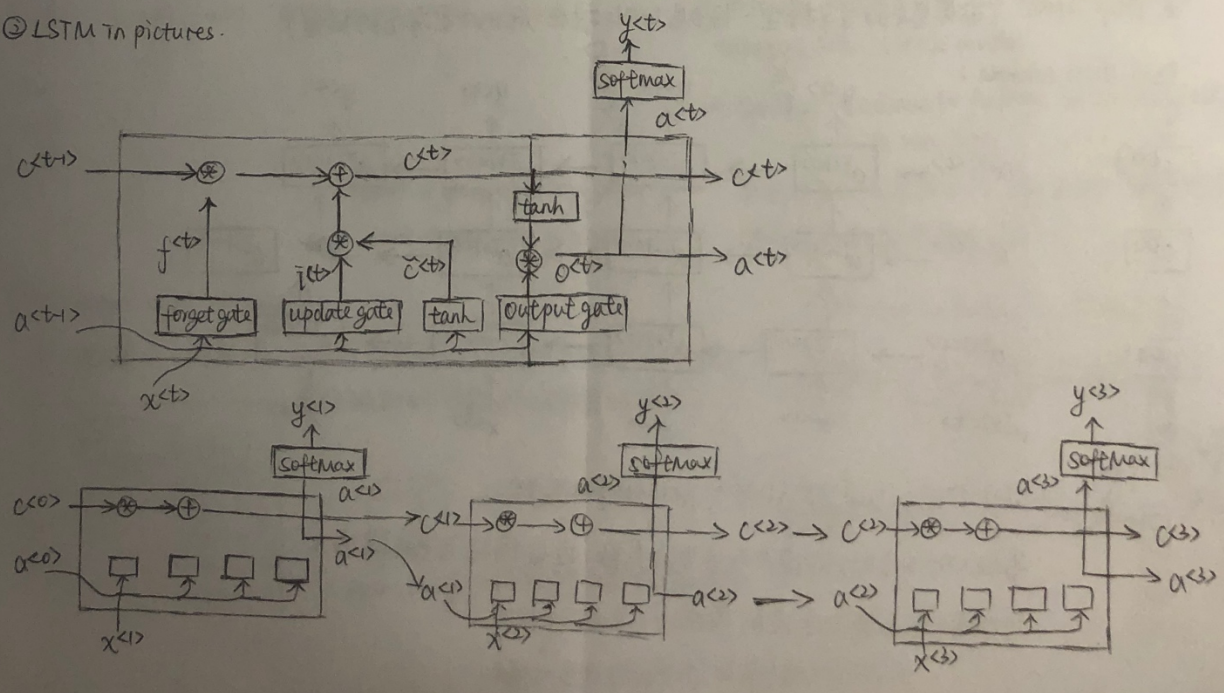
(forget) $\Gamma_f = \sigma(W_f[a^{\langle t-1\rangle}, x^{\langle t\rangle}] + b_f)$.

(output) $\Gamma_o = \sigma(W_o[a^{\langle t-1\rangle}, x^{\langle t\rangle}] + b_o)$.

$c^{\langle t\rangle} = \Gamma_u * \tilde{c}^{\langle t\rangle} + \Gamma_f * c^{\langle t-1\rangle}$

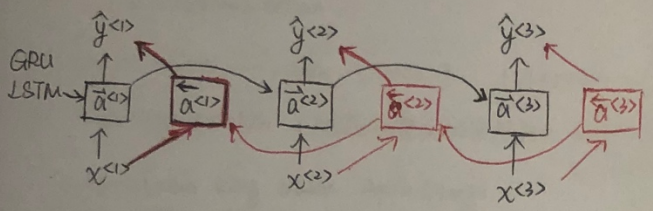
$a^{\langle t\rangle} = \Gamma_o * \tanh(c^{\langle t\rangle})$.

② LSTM in pictures.



(四) Bidirectional RNN: 双向循环神经网络 (BRNN)

在一个时间点获得序列中前部分的和后部分的信息

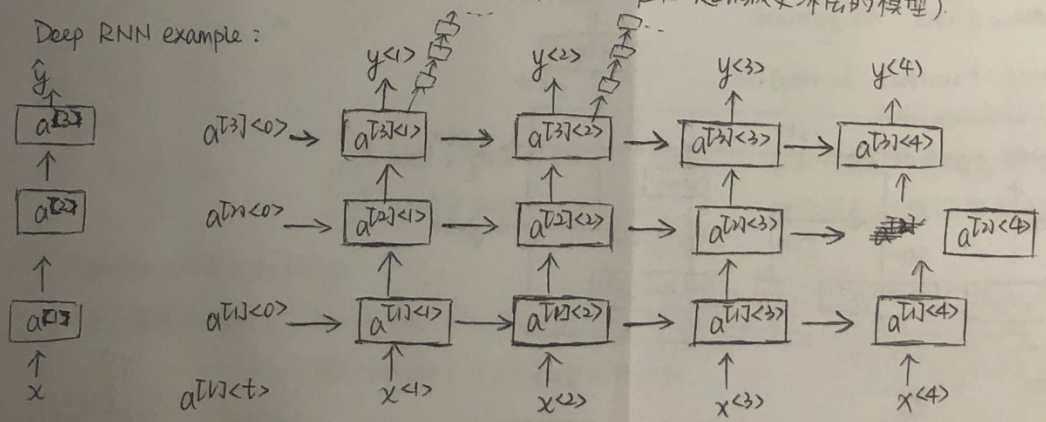


- 对于处理NLP问题的大量文本，带有LSTM的双向RNN是非常常用的。(可向前，可向后)
- 对基本的RNN结构，或GRU或LSTM进行的修改，可用来预测任何地方，在序列的中间可利用来自整个序列的信息。
- BRNN缺点：需要整个序列，才可在任何地方预测。

例：建立语音识别系统，BRNN会让你考虑整个演讲内容，但如果是前向实现，需等人停止说话，得到整个话语才可以实际处理它，并进行语音识别预测。

五. Deep RNNs. (非常复杂的学习模型：将多层RNNs堆在一起形成更深层的模型)

Deep RNN example:



$a^{[2]<t>} = g(W_a^{[2]}[a^{[2]<t-1>}, a^{[1]<t>}] + b_a^{[2]})$

当加入时间七时，即使只有很少的层数，这些网络也已变得很大。

可以有一些无横向连接的模块。

Chapter 2 Word Representation

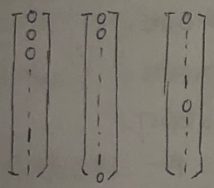
一、Introduction to Word Embeddings.

1. Word Representation.

V=[a, aaron, ..., zulu, <UNK>] |V|=10000.

① 1-hot representation.（独热向量间的乘积为0. 无法度量任意两个向量之间的距离）

Man (5391) Woman (9853) King (4914) Queen (7157) Apple (456) Orange (6257).



⇓

无法知晓 Apple 与 Orange 之间的相似度是否高于 King. Queen. ～

② Featurized-representation = word embedding.

（特征化表示：文字嵌入）

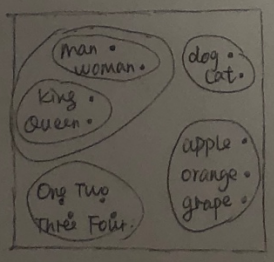
	Man	Woman	King	Queen	Apple	Orange
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.09	0.01	0.02	0.01	0.95	0.97
...						
Size						
Cost						
⋮						

300 维

I want a glass of orange juice.
I want a glass of apple juice.

③ Visualize word embeddings.

t-SNE 算法（词嵌入）：可以学习相似的特征对于相关联的概念，最终映射到一个更相似的特征向量.



300维特征空间

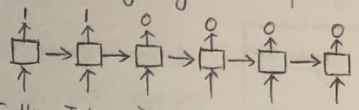
Apple Orange（将它们嵌入在300维特征空间不同的点上）

↓

映射到更低维的空间，使其可画在二维空间.

2. Using Word Embeddings.

① Named entity recognition example.



Sally Johnson is an orange farmer.
Robert Lin is an apple farmer.

学习单词嵌入的算法，可检查大量文本～（未标记）

⇓

采取该词嵌入，将其应用到命名中（识别任务中训练集会变小）

例：Orange. Apple. Grape ⇒ Fruit.

可作迁移学习：将知识迁移到任务上.

如：命名实体识别.

对于这个任务，也许有较小的已标记的训练集.（双向RNN）.

② Transfer Learning and Word Embeddings.

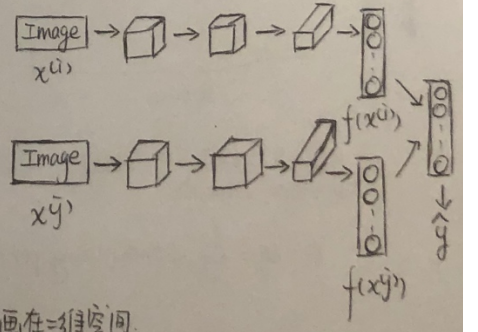
⇒ Learn word embeddings from large text corpus. (Or download pre-trained embedding online.)

⇒ Transfer embedding to new task with smaller training set. (100k words).

⇒ Optional: Continue to finetune the word embeddings with new data.

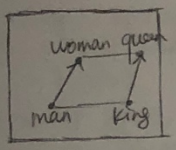
在新任务上训练模型时，可以选择性微调参数.继续用新数据调整单词嵌入.

③ Relation to face coding (embedding)



3. Properties of Word Embeddings.

① Analogies using word vectors:



$e_{man} - e_{woman} \approx e_{king} - e_{w}$.

$\arg\max_{w} \; sim(e_w, e_{king} - e_{man} + e_{woman})$

Find word w.

测量嵌入词 w 和等式右边量有多相似.

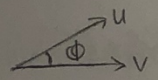
然后找到能最大化其相似度的词.

② Cosine Similarity

$sim(e_w, e_{king} - e_{man} + e_{woman})$

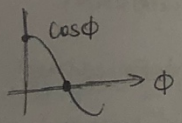
$$sim(u, v) = \frac{u^T v}{\|u\|_2 \|v\|_2}$$ ($u \cdot v$ 内在乘积). (u, v 高度相似, 内积乘积大)

(欧几里德)



$\|u - v\|^2$. 欧几里得距离.

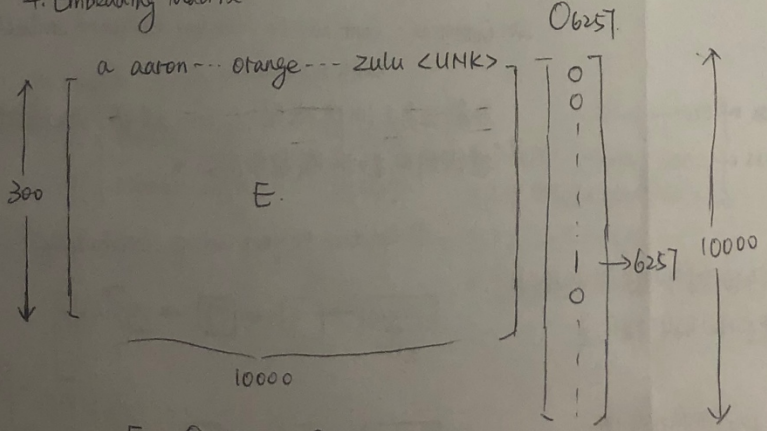
是对非相似度的测量.



嵌入词类比相似根绘: Man: Woman as Boy: Girl.

(类比推理系统)

4. Embedding Matrix.



$E \cdot O_{6257} = e_{6257}$

(300, 10K) (10K, 1)

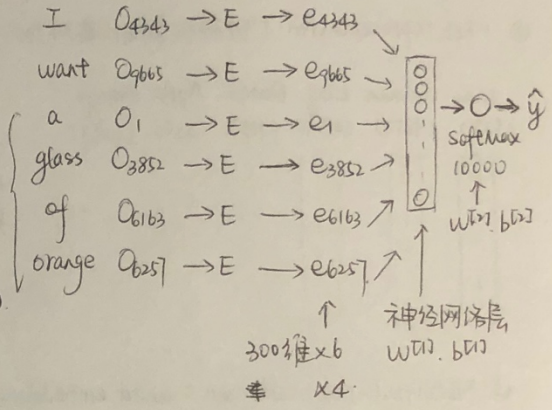
$E \cdot O_j = e_j$ (embedding for word j)

二. Learning word embeddings.

1. Neural Language Model

I want a glass of orange juice.

4343 9665 1 3852 6163 6257



2. Other Context/ Target Pairs.

I want a glass of orange juice to go along with my cereal.

Target.

Context: Last 4 words.

构建语言模型: 用目标词前的几个词作上下文.

学习语言模型本身: 4 words on left & right.

(学习词嵌入) Last 1 word.

Nearby 1 word.

3. Word2Vec

① Skip-grams

I want a glass of orange juice to go along with my cereal.

Context Target

建立监督式学习.（输入一单词，会试图跳过一些单词去预测另一些单词）

② Model：

Vocabulary Size = 10000k. $x \longrightarrow y$

Context. c("orange") $\longrightarrow$ Target t("juice")
6257 4834

$O_c \rightarrow E \longrightarrow e_c \longrightarrow$ SoftMax $\rightarrow \hat{y}$
$(e_c = Eo_c)$

SoftMax: $p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10000} e^{\theta_j^T e_c}}$ ⇒ 计算很慢.

（解决方法：分层 SoftMax.）⇒ 树上的每个内部节点，都可用一个二元分类器表示.

$\mathcal{L}(\hat{y}, y) = -\sum_{i=1}^{10000} y_i \log \hat{y}_i$，$\theta_t$ = parameter associate with output.

不完美平衡树.
- 对于常见词：总需几次遍历
- 对于陌生词：路径深（偏僻）.

$y = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$ ← 4834.

优化损失函数的话，可得到一系列相当好的嵌入矢量.

③ How to sample the context c.?

- the. of. and. ···· 出现多
- orange. apple. ··· 出现少

⇒ 并不是全从训练文本集中随机均匀采样的.

⇒ 有不同的启发式可用来平衡常见词汇、罕见词汇.