

## 目录 - 所有分区页面

- 神经网络基础

- (1) Logistic回归
- (2) logistic回归损失函数
- (3) 梯度下降
- (4) 计算图
- (5) 使用计算图求导
- (6) logistic回归中的梯度下降算法
- (7) 向量化logistic回归的梯度输出
- (8) logistic回归损失函数的解释

- 神经网络

- (1) 神经网络表示
- (2) 计算神经网络的输出
- (3) 激活函数
- (4) 深层网络中的向前传播
- (5) 神经网络的梯度下降
- (6) 搭建深层神经网络块
- (7) 超参数
- (8) 机器学习基础
- (9) 正则化
- (10) 为什么正则化可以减少过拟合
- (11) Dropout正则化
- (12) 理解Dropout
- (13) 归一化输入
- (14) 梯度爆炸与梯度消失

- (15) 神经网络的权重初始化
- (15) 梯度数值逼近
- (16) 梯度检测注意事项
- 优化算法
  - (1) 指数加权平均
  - (2) 理解指数加权平均
  - (3) 指数加权平均的偏差修正
  - (4) 动量梯度下降法
  - (5) RMSprop 算法
  - (6) Adam 优化算法
  - (7) 学习率衰退
  - (8) 局部最优问题
  - (9) 调试处理
  - (10) 为超参数选择合适的范围
  - (11) 单模型与平行多模型
- Batch归一化
  - (1) 网络中的正规化激活函数
  - (2) 将BatchNorm拟合进神经网络
  - (3) BatchNorm为什么奏效
  - (4) BatchNorm时的测试
- SoftMax回归
  - (1) Softmax回归
  - (2) 训练一个Softmax分类器
- 多任务与端到端学习
  - (1) 迁移学习
- 卷积神经网络
  - (1) 边缘检测
  - (2) 更多边缘检测
  - (3) Padding

- [\(4\) 卷积步长](#)
- [\(5\) 三维卷积](#)
- [\(6\) 单层卷积网络](#)
- [\(7\) 简单卷积网络示例](#)
- [\(8\) 池化层](#)
- [\(9\) 卷积神经网络示例](#)
- [\(10\) 为什么使用卷积](#)
- [\(11\) 经典模型](#)
- [\(12\) 残差网络](#)
- [\(13\) 残差网络有什么用](#)
- [\(14\) 网络中的网络以及1\\*1卷积](#)
- [\(15\) 谷歌Inception网络简介](#)
- [\(16\) Inception网络](#)
- [\(17\) 迁移学习](#)

- 目标探测

- [\(1\) 目标定位](#)
- [\(2\) 特征点检测](#)
- [\(3\) 目标检测](#)
- [\(4\) 卷积的滑动窗口实现](#)
- [\(5\) Bounding Box预测](#)
- [\(6\) 交并比](#)
- [\(7\) 非极大值抑制](#)

- 循环神经网络

- [\(1\) 数学符号](#)
- [\(2\) 循环神经网络](#)
- [\(3\) 通过时间的反向传播](#)
- [\(4\) 不同类型的循环神经网络](#)
- [\(5\) 语言模型与序列生成](#)
- [\(6\) 新序列采样](#)

- [\(7\) 带有神经网络的梯度消失](#)
- [\(8\)GRU单元\(门控循环单元\)](#)
- [\(9\) 长短期记忆单元\(LSTM\)](#)
- [\(10\) 双向神经网络](#)
- [\(11\) 深层循环神经网络](#)
- [\(12\) 词汇表征](#)
- [\(13\) 使用词嵌入](#)
- [\(14\) 词嵌入的特性](#)
- [\(15\) 嵌入矩阵](#)
- [\(16\) 学习词嵌入](#)
- [\(17\) Word2Vec](#)
- [\(18\) 负样本](#)
- [\(19\) Glove词向量](#)
- [\(20\) 情绪分类](#)
- [从序列到序列模型](#)
  - [\(1\) 基础模型](#)
  - [\(2\) 选择最有可能的句子](#)
  - [\(3\) 定向搜索](#)
  - [\(4\) 改进定向搜索](#)
  - [\(5\) 注意力模型直观理解](#)
  - [\(6\) 注意力模型](#)

# (1) Logistic 回归

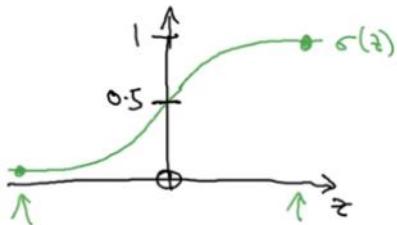
2022年2月15日 20:08

## Logistic Regression

Given  $x$ , want  $\hat{y} = P(y=1|x)$   
 $x \in \mathbb{R}^{n_x}$

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

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



$$x_0 = 1, \quad x \in \mathbb{R}^{n_x+1}$$

$$\hat{y} = \sigma(w^T x)$$

$$w = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{n_x} \end{bmatrix} \quad \left. \begin{array}{l} \theta_0 \} b \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{n_x} \end{array} \right\} w$$

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

If  $z$  large  $\sigma(z) \approx \frac{1}{1+e^{-z}}$

If  $z$  large negative number

$$\sigma(z) = \frac{1}{1+e^{-z}} \approx \frac{1}{1+e^{Bignum}} \approx 0$$

Andrew

这部分与机器学习里的一样，我们令输出  $y = \sigma(z) = \sigma(w^T x + b)$ ，其中  $\sigma(z) = \frac{1}{1+e^z}$  这样设置可以令函数在 0-1 之间

注:  $\theta_0 = b$ ,  $\begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} = w$

## (2) logistic 回归 损失函数

2022年2月15日 20:23

### Logistic Regression cost function

$$\rightarrow \hat{y}^{(i)} = \sigma(w^T \underline{x}^{(i)} + b), \text{ where } \sigma(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}} \quad z^{(i)} = w^T \underline{x}^{(i)} + b$$

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

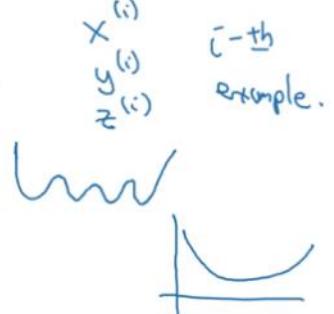
**Loss** (error) function:  $L(\hat{y}, y) = \frac{1}{2} (\hat{y} - y)^2$

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

If  $y=1$ :  $L(\hat{y}, y) = - \log \hat{y} \leftarrow$  Want  $\log \hat{y}$  large, want  $\hat{y}$  large.

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

**Cost** function:  $J(w, b) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log (1-\hat{y}^{(i)})]$



### (3) 梯度下降

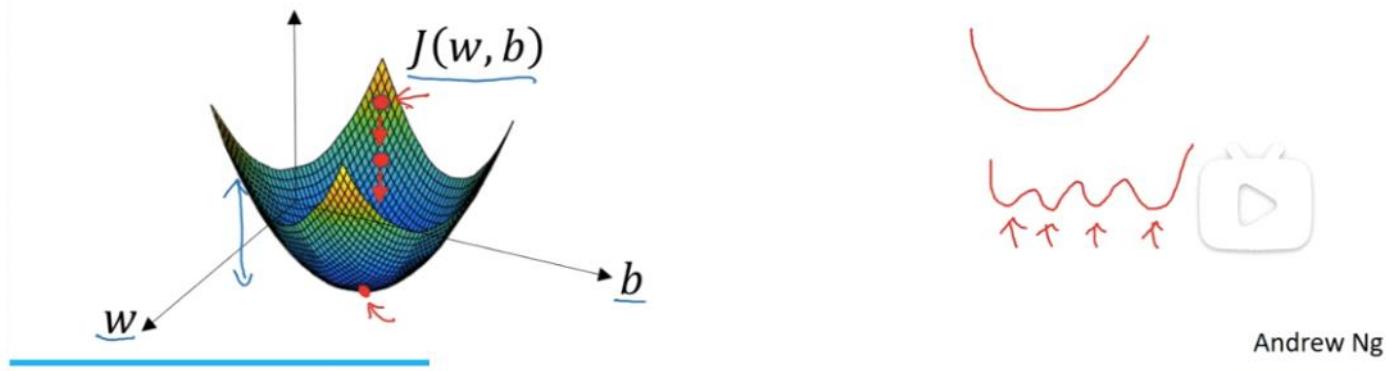
2022年2月15日 21:21

## Gradient Descent

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

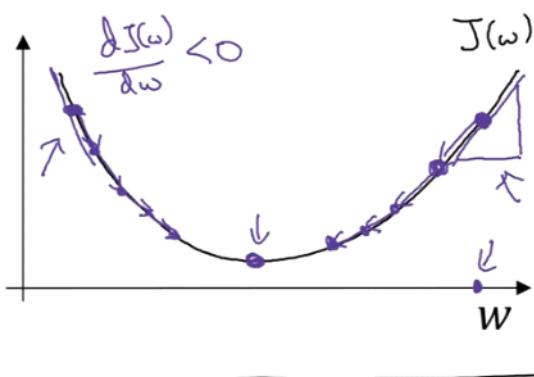
$$J(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^m y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

Want to find  $w, b$  that minimize  $J(w, b)$



这里的和机器学习里面一致，只是需要注意的是参数 $\theta$ 被拆分成了 $w$ 和 $b$ 。

## Gradient Descent



Repeat {

$$\left. \begin{array}{l} w := w - \alpha \frac{\partial J(w)}{\partial w} \\ w := w - \alpha dw \end{array} \right\}$$

learning rate

$$\frac{\partial J(w)}{\partial w} = ?$$

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

① "partial derivative"

这里的更新公式变为

$$w = w - \alpha \frac{dJ(w, b)}{dw}$$

$$b = b - \alpha \frac{dJ(w, b)}{db}$$

## (4) 计算图

2022年2月15日 21:30

神经网络的计算是按照正向或反向传播过程来实现的，首先计算出神经网络的输出，紧接着进行一个反向传输，后者我们用来计算出对应的梯度或导数。

### Computation Graph

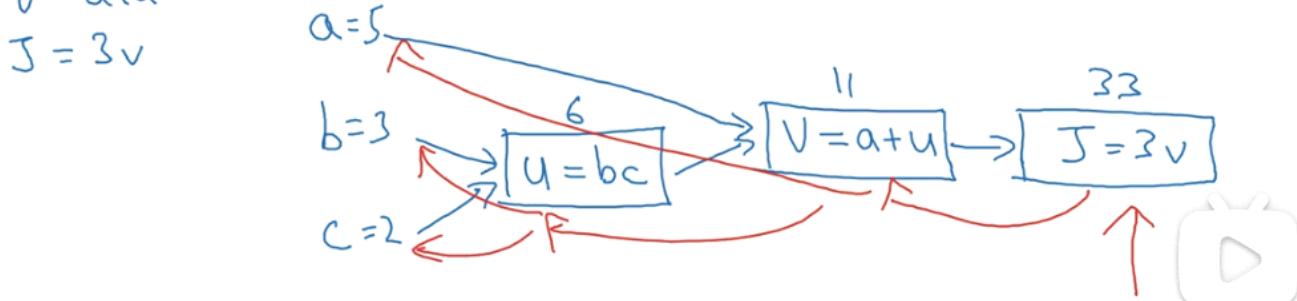
$$J(a, b, c) = 3(a + bc) = 3(5 + 3 \cdot 2) = 33$$

$\underbrace{a}_{u}$   
 $\underbrace{bc}_{v}$   
 $\underbrace{3}_{J}$

$$u = bc$$

$$v = a + u$$

$$J = 3v$$



### 计算图：

如图我们想要计算函数  $J(a, b, c)$  的值其中

$$J(a, b, c) = 3(a + bc) = 3(5 + 3 * 2) = 33$$

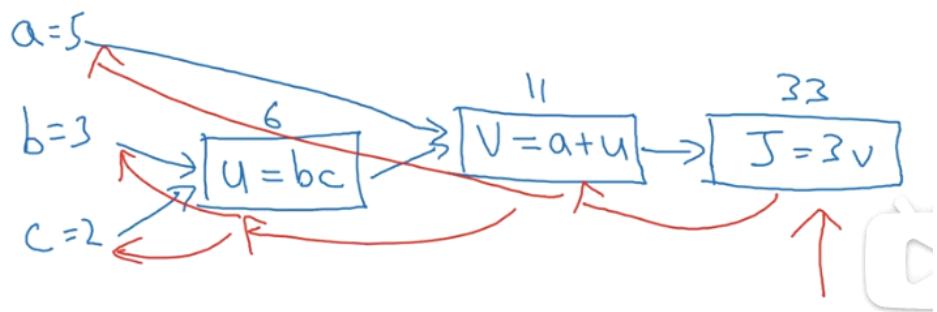
我们计算函数  $J$ ，可以进行如下步骤的拆分

$$U = bc$$

$$V = a + u$$

$$J = 3v$$

我们可以得到如下的计算图



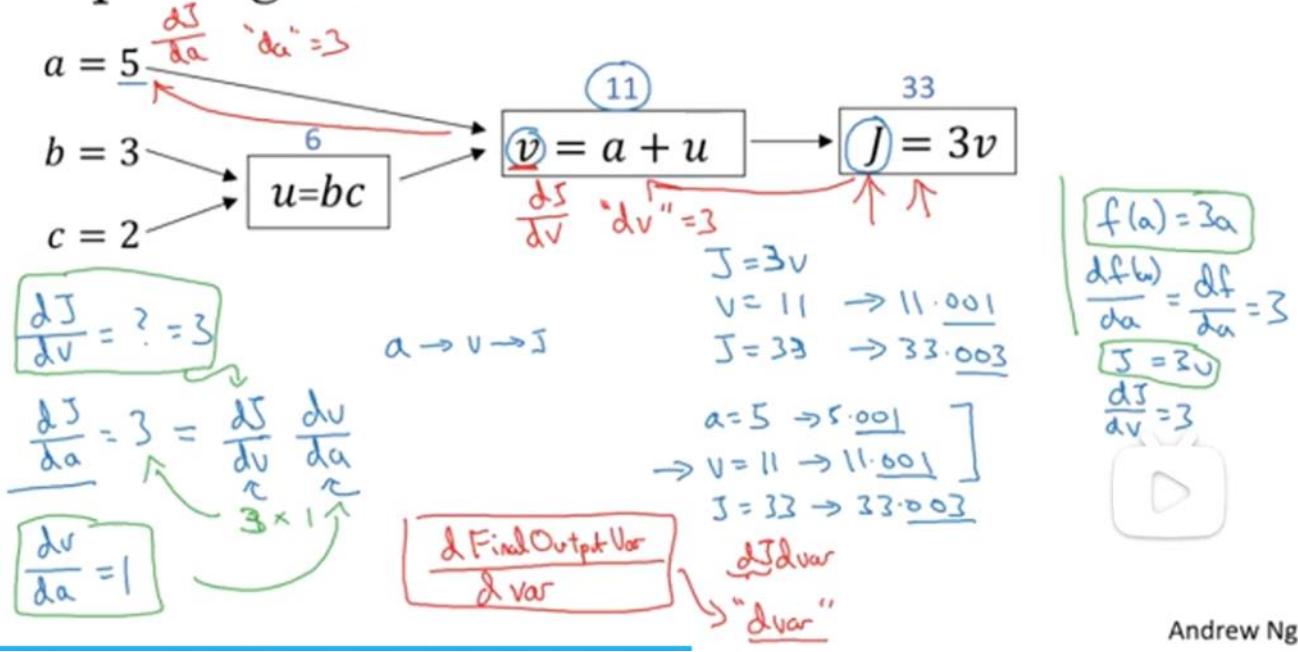
当结果有不同或者一些特殊的输出变量时，比如 $J$ 也是我们想要优化，在logistic回归中， $J$ 也是想要最小化的成本函数，可以看出，一个从左到右的过程，可以计算出函数 $J$ 的值，在接下来的章节中我们将会看到为了计算导数，从右到左的过程。

## (5) 使用计算图求导

2022年2月15日 21:54

网易云课堂

### Computing derivatives



Andrew Ng

假设我们要计算  $\frac{dJ}{dv}$ , 我们应该怎么计算呢?

我们若改变v的值, 那么J将如何改变呢?

我们有  $J = 3v$ , 我们将  $v = 11$ , 变化为  $v = 11.001$ , 而  $J = 33$  变化为  $33.003$ , 即我们可以得到我们在x轴变动1, 而  $\frac{dJ}{dv} = 3$ , 我们y轴将变化的动静为x轴的3倍, 又当我们令  $a = 5$  变成  $a = 5.001$ ,  $v = 11.001$ ,  $J = 33.003$ , 即  $a$  的波动影响  $v$ , 继而影响  $J$ , 又我们有链式法则  $\frac{dJ}{da} = \frac{dJ}{dv} \frac{dv}{da} = 3$ , 即若我们要计算  $\frac{dJ}{da}$  需要先计算出  $\frac{dJ}{dv}$  和  $\frac{dv}{da}$

## Computing derivatives

$$\begin{array}{l}
 \begin{array}{l}
 \frac{\partial J}{\partial a} = 5 \quad \text{---} \\
 \underline{\frac{\partial a}{\partial a}} = 3 \quad \text{---} \\
 b = 3 \quad \text{---} \\
 \underline{\frac{\partial J}{\partial b}} = \underline{\frac{\partial b}{\partial b}} = 6 \quad \text{---} \\
 c = 2 \quad \text{---} \\
 \underline{\frac{\partial J}{\partial c}} = \underline{\frac{\partial c}{\partial c}} = 9 \quad \text{---}
 \end{array}
 \end{array}$$

我们通过上面的方法可以依次计算出各导数。

## (6) logistic回归中的梯度下降算法

2022年2月16日 11:04

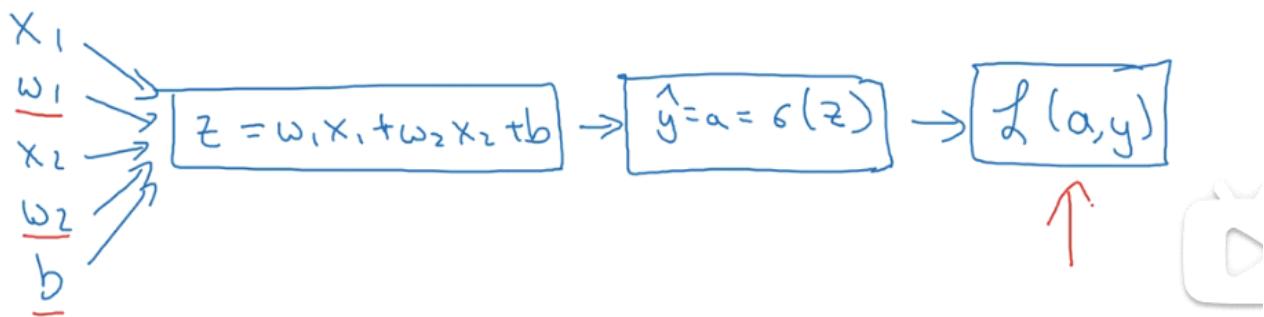
本节我们将讨论怎样去计算偏导数，来实现logistic回归的梯度下降法，本节中我们将用计算图的方式来计算梯度。

## Logistic regression recap

$$\rightarrow z = w^T x + b$$

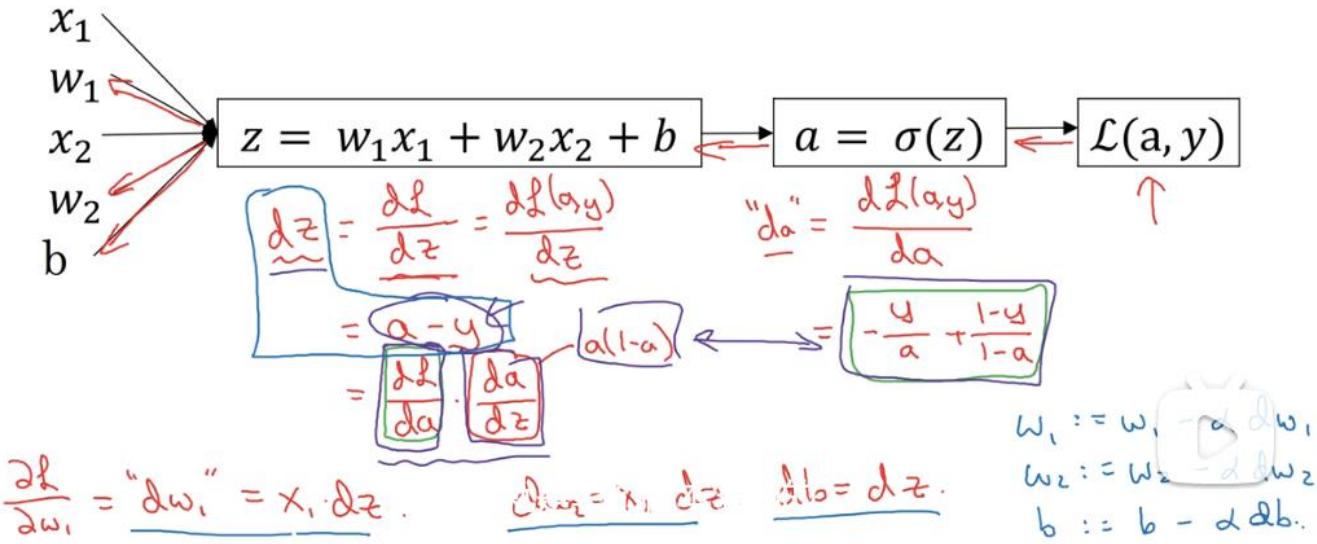
$$\rightarrow \hat{y} = a = \sigma(z)$$

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



我们有逻辑回归定义如上，我们有两个特征 $x_1, x_2$ ，我们有输入 $x_1, x_2, w_1, w_2, b$ 。

# Logistic regression derivatives



我们有函数定义

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

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

我们有

$$da = \frac{d\mathcal{L}(a,y)}{da} = -\frac{y}{a} + \frac{1-y}{1-a}$$

$$dz = \frac{d\mathcal{L}}{da} \frac{da}{dz} = a - y$$

$$\frac{da}{dz} = \frac{da}{dz} = -\left[\frac{-e^z}{e^z + 1}\right] = \frac{e^z}{(e^z + 1)^2} = a\left(\frac{1}{e^z + 1}\right) = a(1 - a)$$

我们通过这种方法求出各参数的偏导，然后通过偏导进行梯度更新，我们有公式

$$w_1 = w_1 - a(dw_1)$$

$$w_2 = w_2 - a(dw_2)$$

$$b = b - a(db)$$

## (7) 向量化 logistic 回归的梯度输出

2022年2月16日 12:05

# Implementing Logistic Regression

```
J = 0, dw1 = 0, dw2 = 0, db = 0
for i = 1 to m:
    z(i) = wTx(i) + b
    a(i) = σ(z(i))
    J += -[y(i) log a(i) + (1 - y(i)) log(1 - a(i))]
    dz(i) = a(i) - y(i)
    dw1 += x1(i)dz(i)
    dw2 += x2(i)dz(i)
    db += dz(i)
J = J/m, dw1 = dw1/m, dw2 = dw2/m
db = db/m
```

$$\begin{aligned}z &= w^T X + b \\&= n p \cdot \text{dot}(w, X) + b \\A &= \sigma(z) \\d\bar{z} &= A - Y \\dw &= \frac{1}{m} X d\bar{z}^T \\db &= \frac{1}{m} np \cdot \text{sum}(d\bar{z}) \\w &:= w - \alpha dw \\b &:= b - \alpha db\end{aligned}$$



左边是普通实现，右边是向量化的实现。

## (8) logistic 回归损失函数的解释

2022年2月16日 12:53

### Logistic regression cost function

$$\begin{aligned} &\rightarrow \left\{ \begin{array}{ll} \text{If } y = 1: & p(y|x) = \hat{y} \\ \text{If } y = 0: & p(y|x) = 1 - \hat{y} \end{array} \right\} p(y|x) \\ &\rightarrow p(y|x) = \hat{y}^y (1-\hat{y})^{(1-y)} \\ &\text{If } y=1: p(y|x) = \hat{y} \stackrel{\circ}{=} 1 \\ &\text{If } y=0: p(y|x) = \hat{y}^0 (1-\hat{y})^{(1-y)} = 1 * (1-\hat{y}) = 1-\hat{y} \\ &\uparrow \log p(y|x) = \log \hat{y}^y (1-\hat{y})^{(1-y)} = y \log \hat{y} + (1-y) \log(1-\hat{y}) \end{aligned}$$

Andrew Ng

我们有

$$p(y|x) \begin{cases} \text{if } y = 1: p(y|x) = \hat{y} \\ \text{if } y = 0: p(y|x) = 1 - \hat{y} \end{cases}$$

我们可以令

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

我们当

$$\text{if } y=1 \quad p(y|x) = \hat{y}(1 - \hat{y})^{(0)} = \hat{y}$$

$$\text{if } y=0 \quad p(y|x) = 1 * (1 - \hat{y})^{(1)} = 1 - \hat{y}$$

以上是 $p(y|x)$ 的定义，由于 $\log$ 函数是严格单调递增函数，则最大化的 $\log p(y|x)$

$$\log p(y|x) = \log \hat{y}^y (1 - \hat{y})^{(1-y)}$$

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

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

这就是我们前面提到的成本函数的负值，前面有一个负号的原因

是，当训练学习算法时，希望算法输出的概率值是最大的，然而在logistic回归中，我们需要最小化损失函数，因此最小化损失函数就是最大化  $\log p(y|x)$

## Cost on $m$ examples

$$\begin{aligned} \log p(\text{labels in training set}) &= \log \prod_{i=1}^m p(y^{(i)}|x^{(i)}) \leftarrow \\ \log p(\dots) &= \sum_{i=1}^m \underbrace{\log p(y^{(i)}|x^{(i)})}_{-\mathcal{L}(\hat{y}^{(i)}, y^{(i)})} \\ &= -\sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) \\ (\text{Cost: minimize}) \quad J(w, b) &= \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) \end{aligned}$$

Maximum likelihood estimator ↗



$m$ 个训练样本的总体成本函数如何表示？

假设所有训练样本服从同一分布且相互独立，也即独立同分布的所有这些样本的联合概率，就是每个样本概率的乘积，我们有公式  
 $P(\text{labels in training set}) = \prod_{i=1}^m p(y^{(i)}, x^{(i)})$

如果你想要做最大似然估计，需要寻找一组参数使得给定样本的观测概率最大，但令这个概率最大化，等价于令其对数最大化在等式两边取对数，训练集的标签出现的概率的对数为

$$\log p(\dots) = \sum_{i=1}^m \log(p(y^{(i)}, x^{(i)}))$$

在统计学中的最大似然估计，需要求出一组参数，使这个式子取最大值

$$\sum_{i=1}^m \log(p(y^{(i)}, x^{(i)})) = \sum_{i=1}^m -\mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

由于训练模型时，目标让成本函数最小化，所以我们不是直接用最

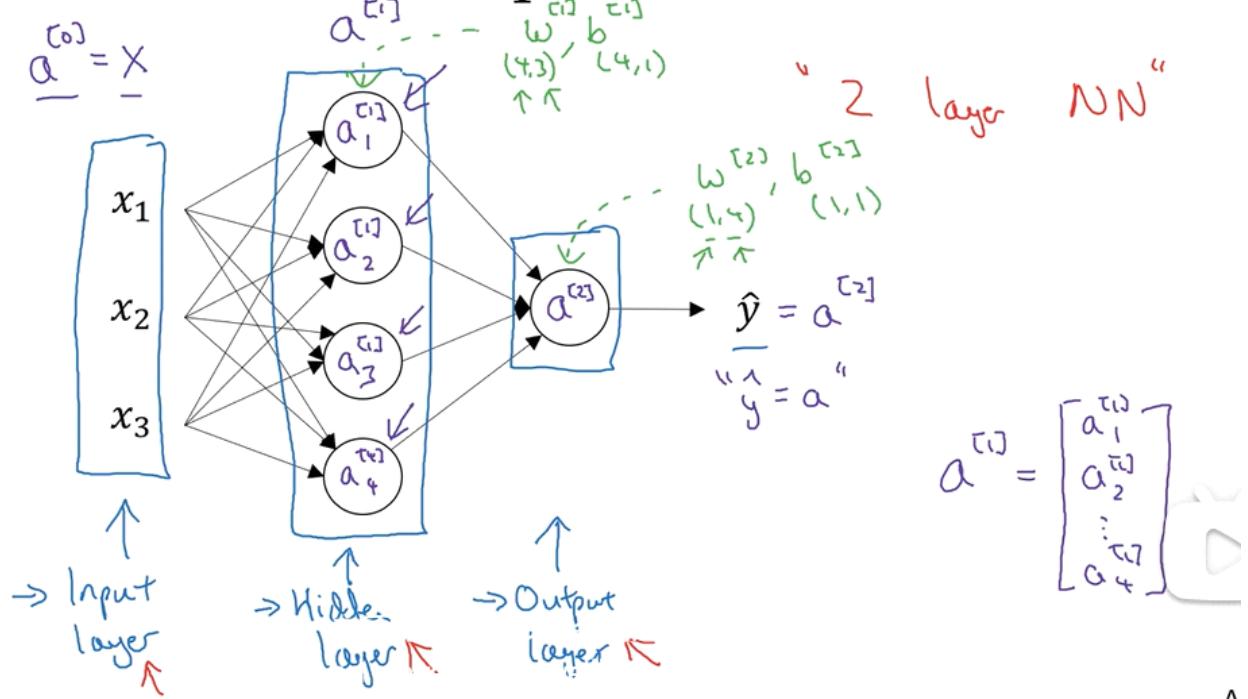
大似然概率，我们需要去掉负号，最后可以对成本函数进行缩放，我们有前面加一个额外常数因子 $\frac{1}{m}$ ，所以我们有公式

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

# (1) 神经网络表示

2022年2月16日 19:24

## Neural Network Representation

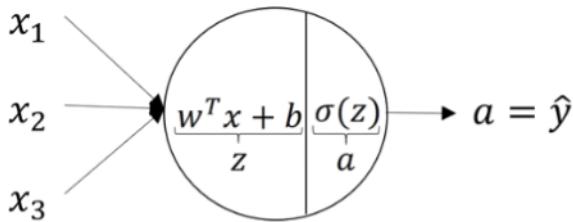


我们有神经网络参数  $w^{(1)}, b^{(1)}$ ，其中  $w^{(1)}$  是一个  $4*3$  的矩阵，其中 4 表示有四个单元，3 表示有三个输入， $b^{(1)}$  是一个  $4*1$  的矩阵，其中 4 为四个单元，1 应该是一个偏置输入(单元)。

## (2) 计算神经网络的输出

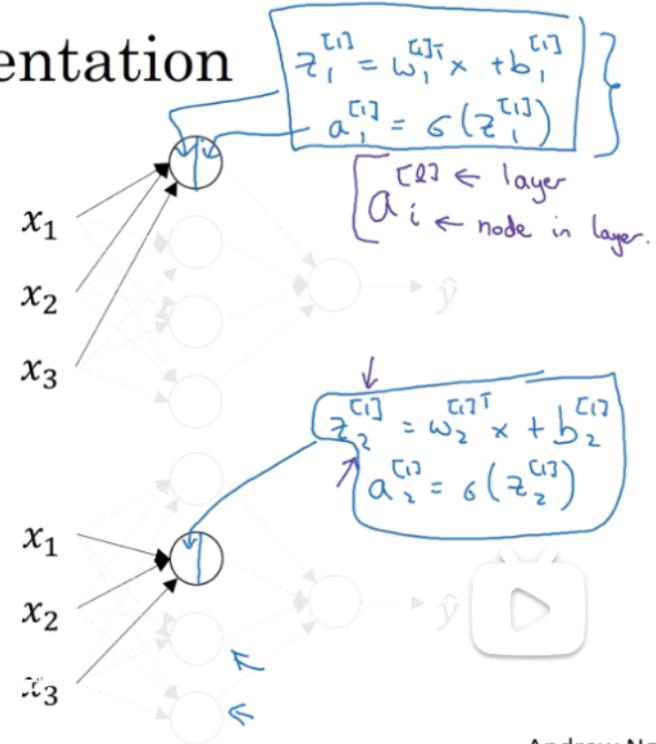
2022年2月17日 15:01

### Neural Network Representation



$$z = w^T x + b$$

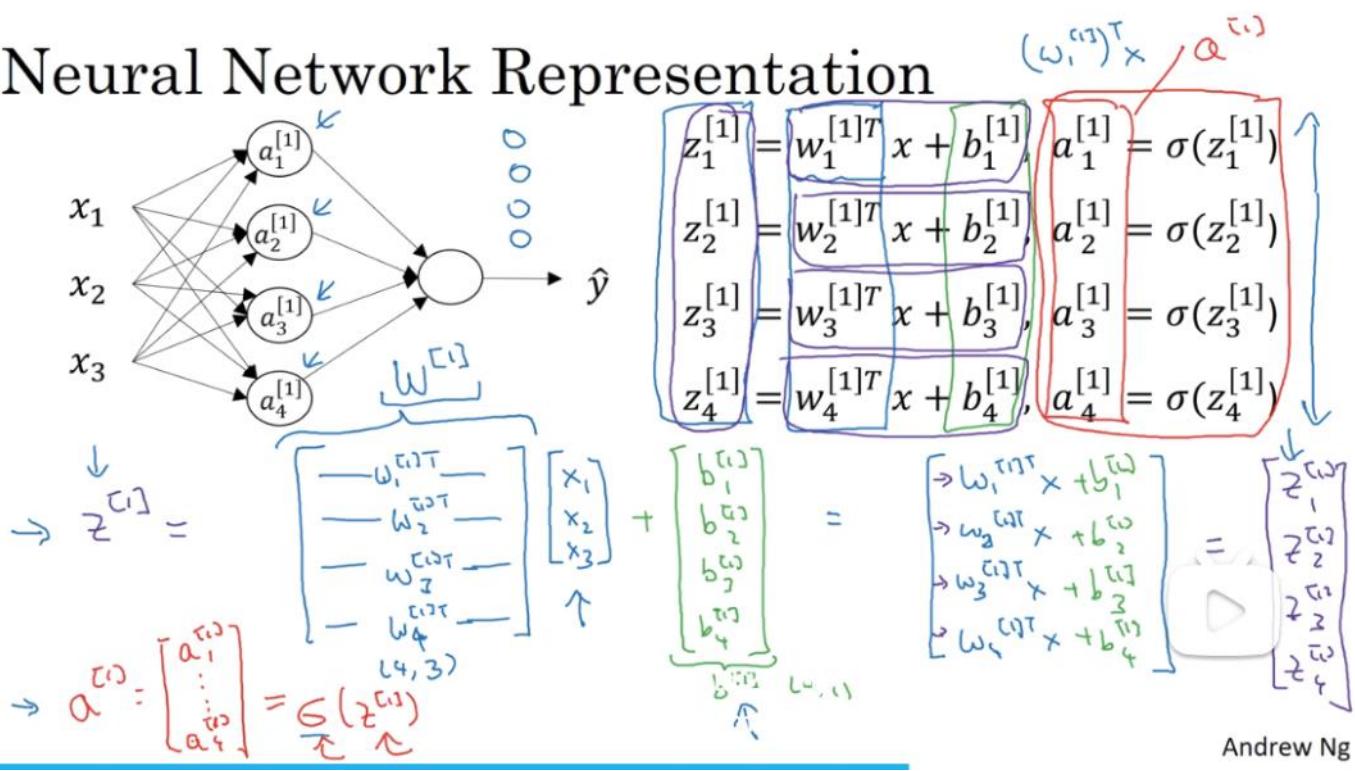
$$a = \sigma(z)$$



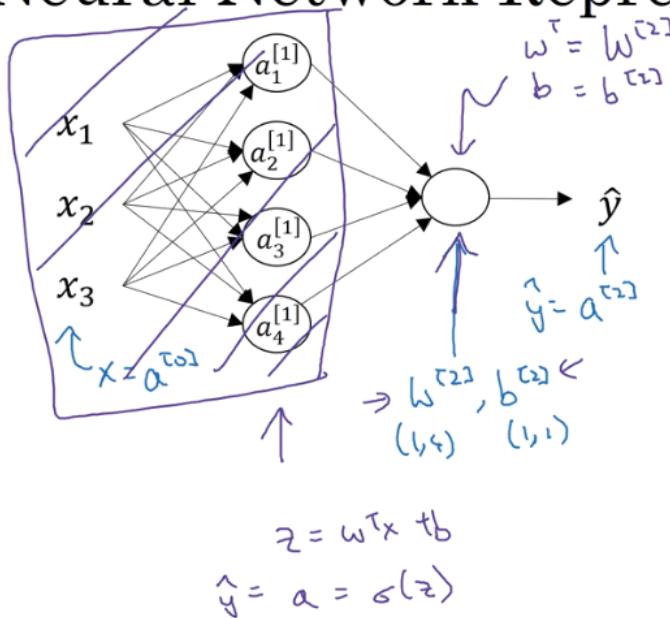
Andrew Ng

我们暂且把神经网络中的一个隐藏单元与logitst回归对比，可以看到的是logistic回归与神经网络中的计算输出相似。

# Neural Network Representation



# Neural Network Representation learning



$$\rightarrow z^{[1]} = W^{[1]} x + b^{[1]}$$

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

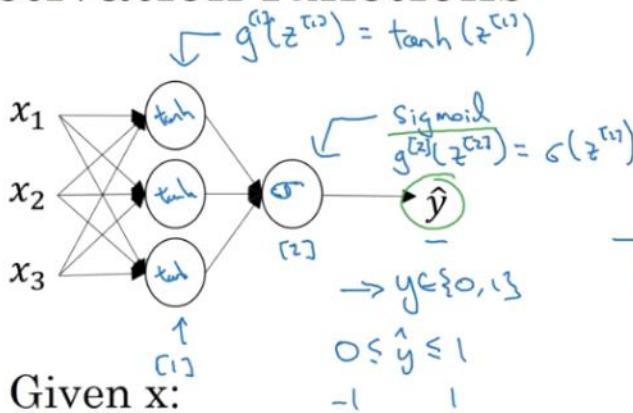
$$\rightarrow z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$

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

### (3) 激活函数

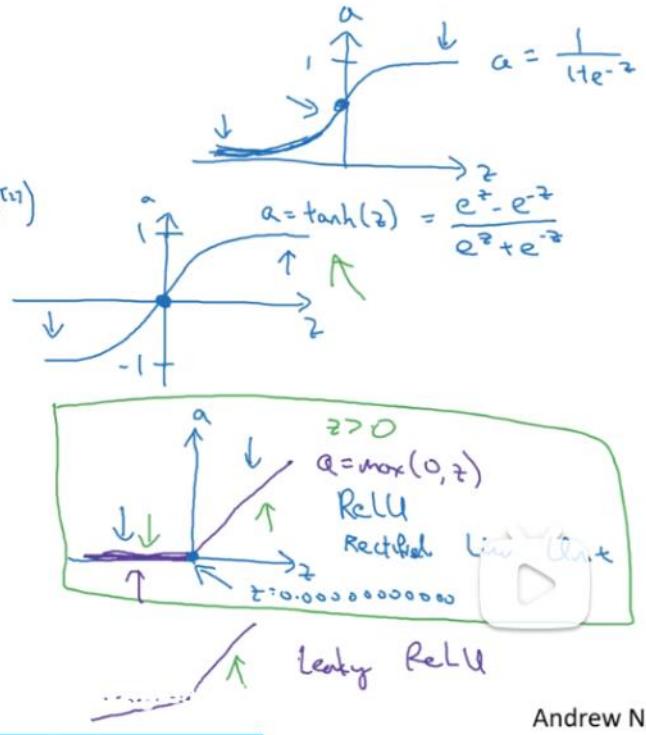
2022年2月17日 17:11

## Activation functions



Given  $x$ :

$$\begin{aligned} z^{[1]} &= W^{[1]}x + b^{[1]} \\ \rightarrow a^{[1]} &= \sigma(z^{[1]}) \quad g(z^{[1]}) \\ z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\ \rightarrow a^{[2]} &= \sigma(z^{[2]}) \quad g(z^{[2]}) \end{aligned}$$



Andrew Ng

## 激活函数

在神经网络的正向传播中，我们使用的函数 $\sigma$ ，这个 $\sigma$ 函数就是所谓的激活函数，在一般的情况下，我们可以使用不同的函数 $g(z)$ ，其中 $g$ 可以是非线性函数，不一定是 $\sigma$ 函数，比如说 $\sigma$ 函数介于0-1之间，但有一个函数总比 $\sigma$ 函数要表现的好，即

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

函数介于-1到1之间，在数字上是 $\sigma$ 函数平移后的版本，事实证明 $g$ 函数总比 $\sigma$ 函数表现要好，因为现在函数介于-1到1之间，激活函数的平均值更加接近于0，就像你在训练算法时，你可能需要平移所有数据，让数据的平均值为0。使用 $\tanh$ 可有类似于数据中心化的效果。使得数据的平均值更加接近于0，这让下层的学习更加方便。 $\tanh$ 函数几乎在任何场合都比 $\sigma$ 函数要更加优越，一个例外是输出层，因为如果 $y$ 是0或是1，那么希望 $y$ 介于0-1之间要更加合理，在这种情况下使用 $\sigma$ 函数作为输出层更加合理。

而 $\tanh$ 函数和 $\sigma$ 函数都有一个缺点是如果 $z$ 非常大或非常小，那么导数

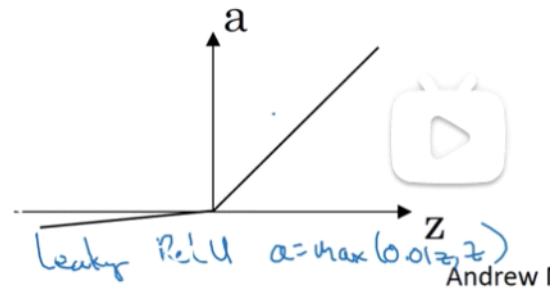
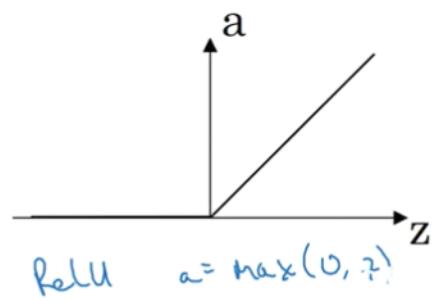
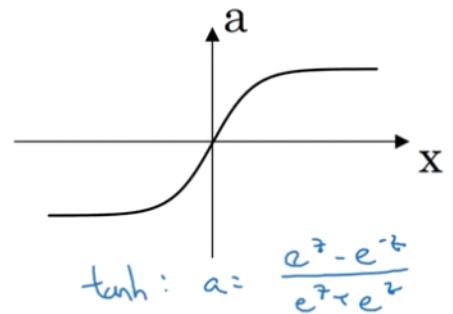
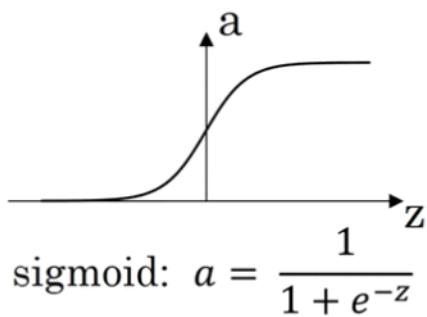
的梯度或者说这个函数的斜率可能更小，所以当z非常大或非常小时，函数的斜率更接近于0。

在机器学习中，加一个有效的工具是修正线性单元，公式为  
 $a = \max(0, z)$

只要z为1，那么导数就为1，如果为负就为0，如果你实际使用这个函数时，z刚好为0，导数是没有定义的，如果编程实现，那么z刚好等于0的概率是非常低的，可以有z=0时，给导数赋值，可以赋值为0或1。

现在修正线性单元作为激活函数已经作为默认选择了，而修正线性单元也有一个缺点，就是当z为负时，导数为0，但修正线性单元还有另一个版本叫做带漏洞的修正线性单元。当z为负时，函数不再为0，它有一个很平缓的斜率

## Pros and cons of activation functions

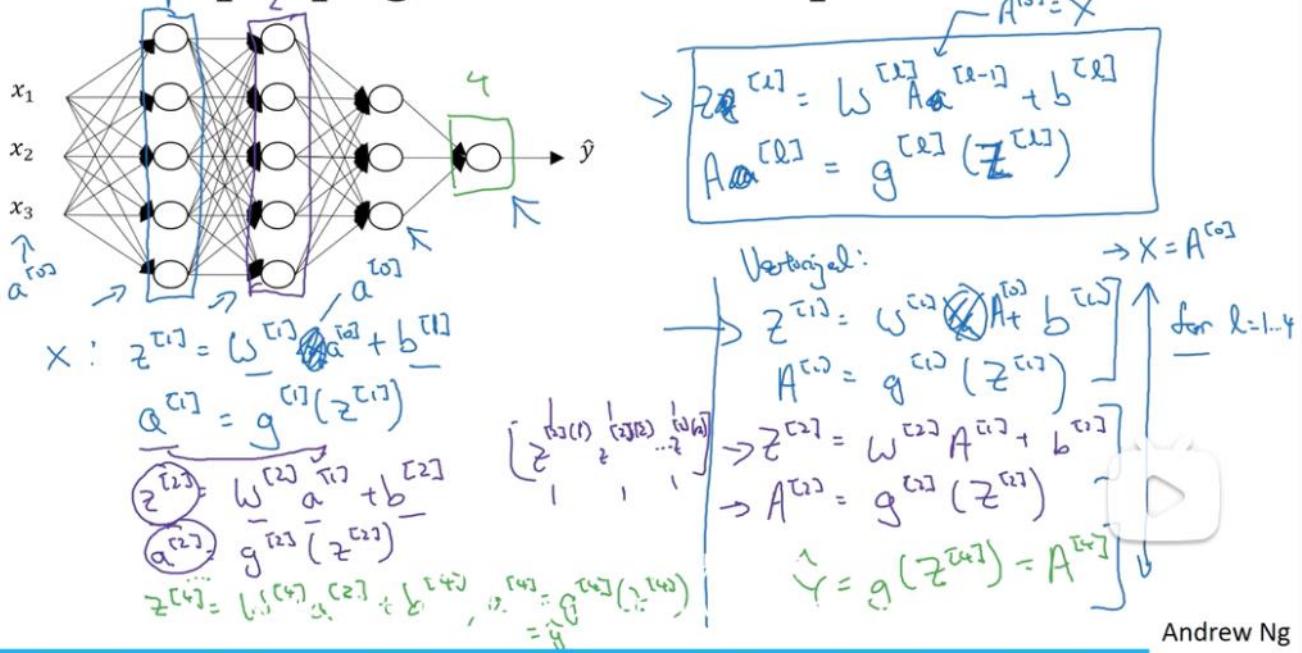


sigmoid函数除非在二元分类中做输出层，否则不用，tanh函数在任何场合下都比较优越，修正线性单元作为默认。

## (4) 深层网络中的向前传播

2022年2月17日 20:30

### Forward propagation in a deep network



## (5) 神经网络的梯度下降

2022年2月17日 21:33

### Gradient descent for neural networks

Parameters:  $w^{(l)}, b^{(l)}$ ,  $w^{(l+1)}, b^{(l+1)}$   $n_x = n^{(0)}, n^{(1)}, \underline{n^{(2)} = 1}$

Cost function:  $J(w^{(0)}, b^{(0)}, w^{(1)}, b^{(1)}) = \frac{1}{m} \sum_{i=1}^m l(\hat{y}_i, y_i)$

Gradient descent:

→ Repeat {

→ Compute predict  $(\hat{y}^{(i)}, i=1 \dots m)$   
 $\frac{\partial J}{\partial w^{(l)}} = \frac{\partial J}{\partial w^{(l)}}, \frac{\partial J}{\partial b^{(l)}} = \frac{\partial J}{\partial b^{(l)}}, \dots$

$$w^{(l+1)} := w^{(l)} - \alpha \frac{\partial J}{\partial w^{(l)}}$$

$$b^{(l+1)} := b^{(l)} - \alpha \frac{\partial J}{\partial b^{(l)}}$$



我们有参数和代价函数如上图所示，我们需要通过求  $dw^{(l)}$  和  $db^{(l)}$  来更新梯度下降，使得参数  $w^{(l)}$  和  $b^{(l)}$  来更新。

注：可以结果机器学习的神经网络

### Formulas for computing derivatives

Forward propagation:

$$z^{(1)} = w^{(1)}X + b^{(1)}$$

$$A^{(1)} = g^{(1)}(z^{(1)}) \leftarrow$$

$$z^{(2)} = w^{(2)}A^{(1)} + b^{(2)}$$

$$A^{(2)} = g^{(2)}(z^{(2)}) = \underline{g}(z^{(2)})$$

Back propagation:

$$\frac{\partial z^{(2)}}{\partial A^{(2)}} = A^{(2)\top} \leftarrow$$

$$\frac{\partial w^{(2)}}{\partial A^{(1)}} = \frac{1}{m} \frac{\partial z^{(2)}}{\partial A^{(1)}} A^{(1)\top}$$

$$\frac{\partial b^{(2)}}{\partial A^{(1)}} = \frac{1}{m} \text{np.sum}(\frac{\partial z^{(2)}}{\partial A^{(1)}}), \text{axis}=1, \text{keepdims=True}$$

$$\frac{\partial z^{(1)}}{\partial X} = \underbrace{w^{(1)\top}}_{(n^{(0)}, m)} \underbrace{\frac{\partial z^{(2)}}{\partial A^{(1)}}}_{\text{element-wise product}} \times \underbrace{g^{(1)\top}(z^{(1)})}_{(n^{(1)}, m)}$$

$$\frac{\partial w^{(1)}}{\partial X} = \frac{1}{m} \frac{\partial z^{(1)}}{\partial X} X^\top$$

$$\frac{\partial b^{(1)}}{\partial X} = \frac{1}{m} \text{np.sum}(\frac{\partial z^{(1)}}{\partial A^{(1)}}, \text{axis}=1, \text{keepdims=True})$$



左为正向传播的计算公式，右边反向传播计算的公式。

## (6) 搭建深层神经网络块

2022年2月17日 20:46

### Forward and backward functions

The diagram illustrates the forward and backward passes through a neural network layer. On the left, a network structure shows inputs \$x\_1, x\_2, x\_3, x\_4\$ entering layer \$l\$, which contains four neurons. The output of layer \$l\$ is \$\hat{y}\$. Below this, handwritten notes define layer \$l\$ as \$(W^{(l)}, b^{(l)})\$. The forward pass is described as inputting \$a^{(l-1)}\$ and outputting \$a^{(l)}\$. The activation function is \$g^{(l)}(z^{(l)})\$ and the pre-activation is \$z^{(l)} = W^{(l)}a^{(l-1)} + b^{(l)}\$. The backward pass is shown on the right, where the input is \$da^{(l)}\$ and the output is \$(da^{(l-1)}, dw^{(l)}, db^{(l)})\$. It uses a cache (\$\text{cache}(z^{(l)})\$) to store intermediate values.

layer  $l$ :  $W^{(l)}, b^{(l)}$

→ Forward: Input  $a^{(l-1)}$ , output  $a^{(l)}$

$z^{(l)} = W^{(l)}a^{(l-1)} + b^{(l)}$  cache  $\text{cache}(z^{(l)})$

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

→ Backward: Input  $da^{(l)}$  cache( $z^{(l)}$ ) output  $da^{(l-1)}, \frac{da}{dw^{(l)}}, \frac{da}{db^{(l)}}$

layer  $l$

$a^{(l-1)}$  →  $a^{(l)}$

$da^{(l)}$  ←  $da^{(l-1)}$

$dw^{(l)}$

$db^{(l)}$

Andrew Ng

在第 $l$ 层中我们有参数  $w^{(l)}, b^{(l)}$

我们有正向传播的输入和输出，输入  $a^{(l-1)}$ ，输出  $a^{(l)}$

我们有

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

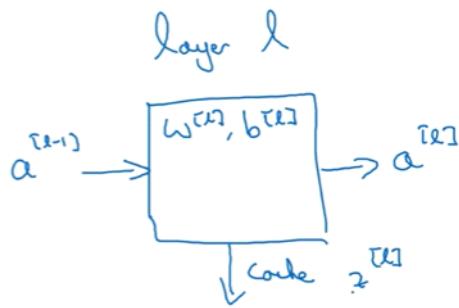
$$a^{(l)} = g^{(l)}(z^{(l)})$$

以上就是我们从输入走向输出的公式，之后我们就可以把  $z^{(l)}$  的值缓存起来

我们有反向传播

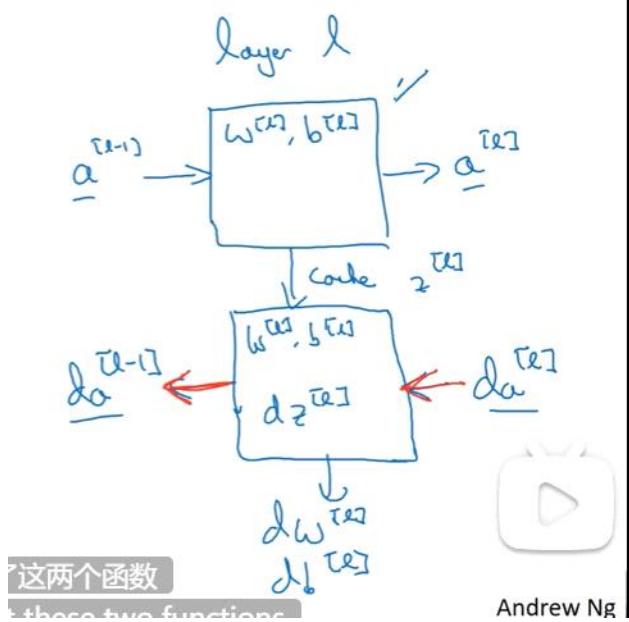
输入:  $da^{(l)}, z^{(l)}$

输出:  $da^{(l-1)}, dw^{(l)}, db^{(l)}$



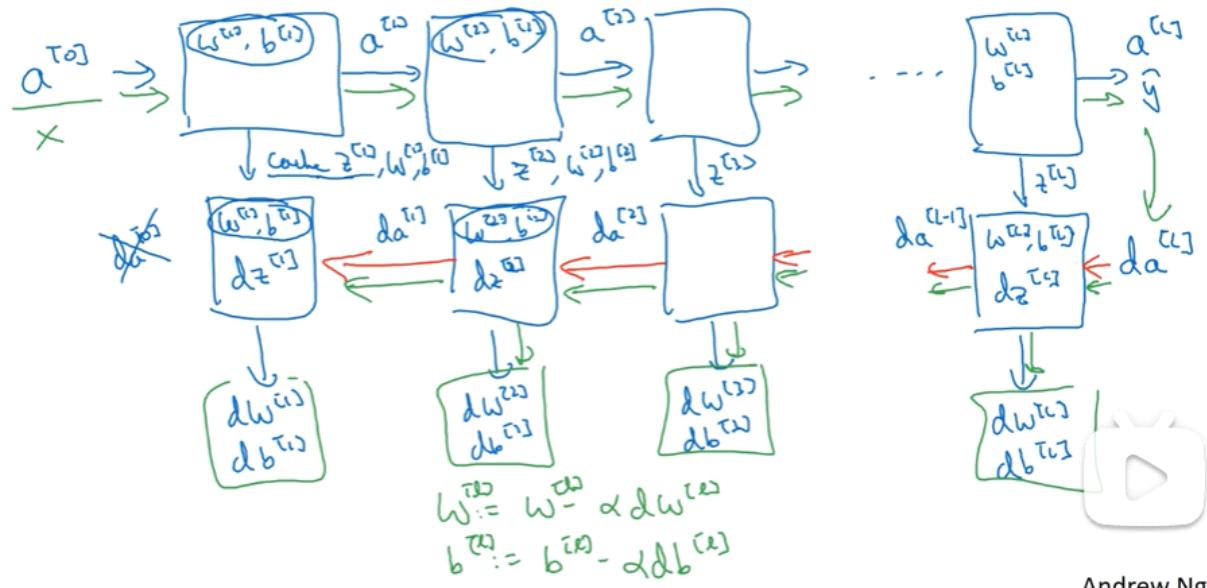
总结起来就是，第l的正向传播会通过输入 $a^{(l-1)}$ ，由 $w^{(l)}$ ,  $b^{(l)}$ 作为参数，输出 $a^{(l)}$ ，以及缓存 $z^{(l)}$

## FUNCTIONS



反向传播的第l层，把 $da^{(l)}$ 和 $z^{(l)}$ 作为输入，再经过参数 $w^{(l)}, b^{(l)}$ ，最后输出 $da^{(l-1)}$ ，在此期间可以计算 $dz^{(l)}, dw^{(l)}, db^{(l)}$

# Forward and backward functions



Andrew Ng

## 正向和反向函数

我们把输入特征  $a^{(0)}$  放入第一层并计算第一层的激活函数，如上图所示，经过一系列的正向和反向传播计算，最后我们计算出了各层参数的导数，并把这些导数运用于梯度计算中。

## (7) 超参数

2022年2月17日 21:16

### What are hyperparameters?

Parameters:  $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]} \dots$

Hyperparameters:

- learning rate  $\alpha$
- #iterations
- #hidden layers  $L$
- #hidden units  $n^{[1]}, n^{[2]}, \dots$
- choice of activation function

Others: Momentum, mini-batch size, regularizations, ...

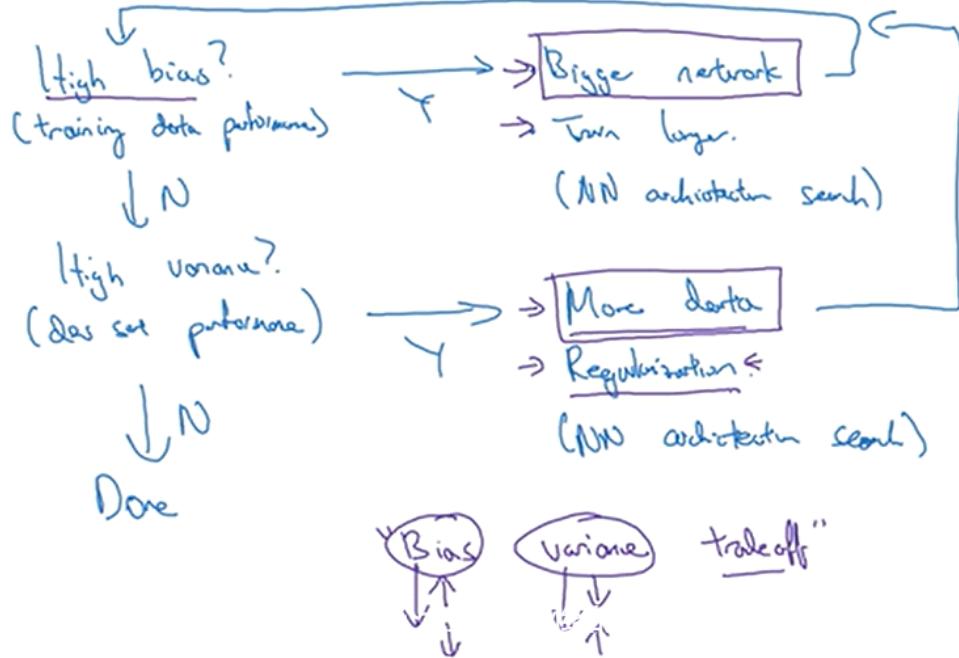
超参数是指那样可以控制参数的变量

如超参数学习率 $a$ ,迭代次数以及隐藏层等都可以决定参数  
 $w$ 和 $b$ 。

## (8) 机器学习基础

2022年2月18日 11:42

### Basic recipe for machine learning



Andrew

当我们出现高偏差时，可以考虑构成更大的神经网络或者花费更多的时间，当我们出现高方差时，可以考虑使用更多的数据以及正则化。

## (9) 正则化

2022年2月18日 12:21

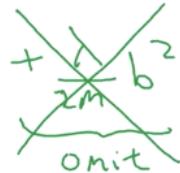
### Logistic regression

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

$$w \in \mathbb{R}^{n_x}, b \in \mathbb{R}$$

$\lambda$  = regularization parameter  
lambd

$$J(w,b) = \underbrace{\frac{1}{m} \sum_{i=1}^m \mathcal{L}(y^{(i)}, \hat{y}^{(i)})}_{\text{L2 regularization}} + \frac{\lambda}{2m} \|w\|_2^2$$



$$\|w\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w \leftarrow$$

$$L_1 \text{ regularization } \frac{\lambda}{2m} \sum_{j=1}^{n_x} |w_j| = \frac{\lambda}{2m} \|w\|_1$$

w will be sparse



这就是在逻辑回归函数中实现L2正则化的过程

我们有函数  $J(w,b)$  为

$$J(w,b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2} \|w\|_2^2$$

其中  $\|w\|_2^2$  为

$$\|w\|_2^2 = \sum_{j=1}^n w_j^2 = w^T w$$

上面式子为 L2 正则项，我们通常选择这个，我们通常通过验证集和交叉集来选定参数  $\lambda$

# Neural network

$$J(\omega^{[0]}, b^{[0]}, \dots, \omega^{[L]}, b^{[L]}) = \underbrace{\frac{1}{m} \sum_{i=1}^m \ell(y^{(i)}, \hat{y}^{(i)})}_{\text{"Frobenius norm"}} + \underbrace{\frac{\lambda}{2m} \sum_{l=1}^L \|\omega^{[l]}\|_F^2}_{\text{正则项}}$$

$$\|\omega^{[l]}\|_F^2 = \sum_{i=1}^{n^{[l-1]}} \sum_{j=1}^{n^{[l]}} (\omega_{ij}^{[l]})^2$$

$\omega: (n^{[L-1]}, n^{[L]}).$

"Frobenius norm"

$$\|\cdot\|_2^2$$

$$\|\cdot\|_F^2$$

$$\delta \omega^{[l]} = (\text{from backprop}) + \frac{\lambda}{m} \omega^{[l]}$$

$$\rightarrow \underline{\omega^{[l]}} := \omega^{[l]} - \alpha \delta \omega^{[l]}$$

"Weight decay"

$$\underline{\omega^{[l]}} := \omega^{[l]} - \alpha \left[ (\text{from backprop}) + \frac{\lambda}{m} \omega^{[l]} \right]$$

$$\frac{\partial J}{\partial \omega^{[l]}} = \delta \omega^{[l]}$$

Andrew

6 人正在看，已装填 101 条弹幕



发个弹幕见证当下

弹幕礼仪 >

发送

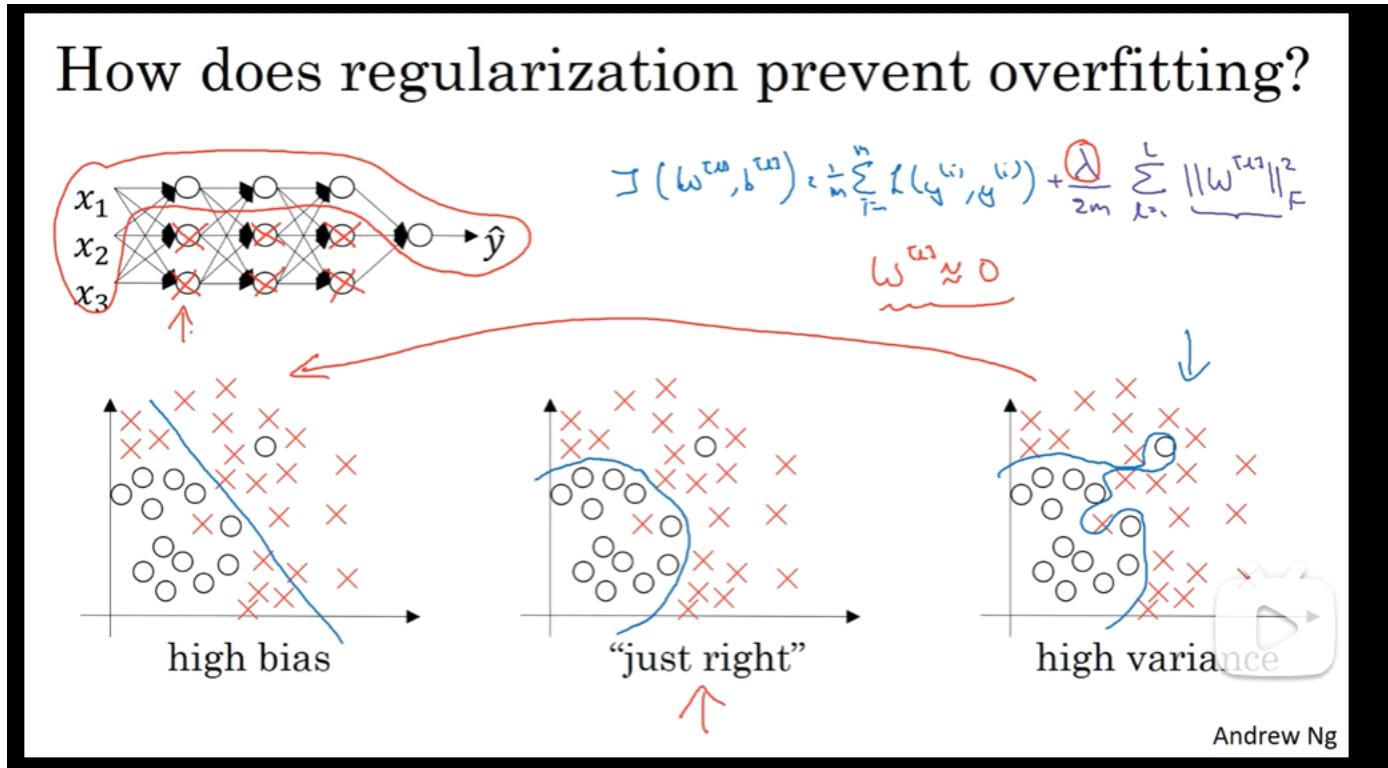
$\|\omega\|_F^2$  通常是一个第  $n-1$  到  $n$  映射的矩阵，我们在进行导数计算时，也需要把正则项计算进行，有

$$\underline{\omega^{(l)}} = \omega^{(l)} - a \left[ (\text{from backprop}) + \frac{\lambda}{m} \omega^{(l)} \right]$$

$$\underline{\omega^{(l)}} = \omega^{(l)} - \frac{a\lambda}{m} \omega^{(l)} - a(\text{from backprop})$$

## (10) 为什么正则化可以减少过拟合

2022年2月18日 15:15



我们有代价函数

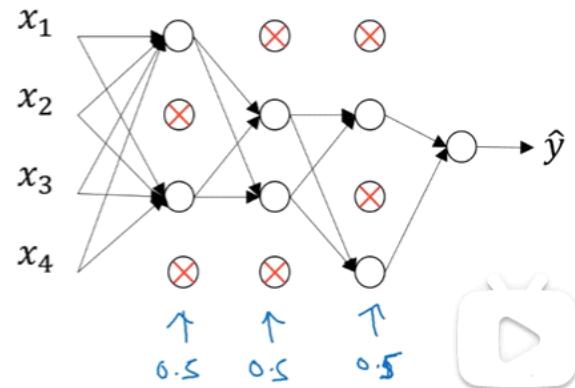
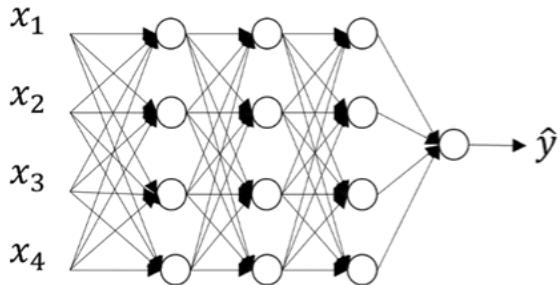
$$J(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_{l=1}^L \|w^{(l)}\|_F^2$$

我们添加正则项，它可以避免数据权值矩阵过大，为什么参数可以减少拟合呢，直观上理解就是如果正则化 $\lambda$ 设置得足够大，权重矩阵 $w$ 被设置为接近于0值。因为当参数 $\lambda$ 足够大时，会使整个公式的大小对于参数 $\lambda$ 的影响接近于0。直观上理解就是把多隐藏单元的权重设为0，于是基本上消除了这些隐藏单元的许多影响，这样的情况下，这个神经网络被大大简化了的神经网络会变成一个很小的网络，小到如同一个逻辑回归单元，可深度却很大，它会使这个网络从过拟合的状态，更接近于左图的高偏差状态，但是 $\lambda$ 会存在一个中间值，于是会有一个接近"ust right"的中间状态，直观理解就是 $\lambda$ 增加到足够大, $w$ 会接近于0，直观上我们认为大量的隐藏单元被完全消除了，其实不然，实际上是该神经网络的所有隐藏单元依然存在，但它们的影响变得更小了

## (11) Dropout 正则化

2022年2月18日 16:15

### Dropout regularization



Andrew Ng

我们有dropout会遍历网络的每一层，并设置消除神经网络节点的概率，假设网络中的每一层，每个节点都以抛硬币的方式设置概率，每个节点得以保留和消除的概率都是0.5，设置完节点概率，我们会消除一些节点，然后删掉从该节点进出的连线，最后得到一个节点更少，规模更小的网络，然后用backprop方式进行训练，右图是一个网络节点精简后的一个样本，对于其它样本，我们也保留一类节点集合，删除其它类型的节点集合，对于每一个训练样本，我们都将采用一个精简后的神经网络来训练它。

# Implementing dropout (“Inverted dropout”)

Illustrate with layer  $l=3$ .  $\text{keep-prob} = \frac{0.8}{x}$   $0.2$

$$\rightarrow d^3 = \underbrace{\text{np.random.rand}(a^3.shape[0], a^3.shape[1]) < \text{keep-prob}}$$

$$a^3 = \underbrace{\text{np.multiply}(a^3, d^3)}_{\uparrow \uparrow} \quad \# a^3 *= d^3.$$

$$\rightarrow a^3 /= \cancel{\text{keep-prob}} \leftarrow$$

50 units.  $\rightsquigarrow$  10 units shut off

$$z^{(4)} = w^{(4)} \cdot \underbrace{\frac{a^{(3)}}{x}}_{\text{reduced by } 20\%} + b^{(4)}$$

$$1 = 0.8$$

Test



Andrew Ng

## 实现Dropout(反向随机失活)

我们考虑一个三层神经网络，首先我们定义向量 $d$ ，我们有 $d^3$ 是一个三层的Dropout向量，我们重构矩阵 $d^3$ 使得小于keep-prob，keep-prob代表它表示保留某个隐藏单元的概率，如keep-prob=0.8，它意味着消除任意一个隐藏单元的概率是0.2， $d^3$ 的这个式子的作用就是生成随机矩阵， $d^3$ 是一个矩阵每个样本和每个隐藏单元，其在 $d^3$ 中的对应值为1的概率是0.8，其对应值为0的概率为0.2，我们有公式

$$a^3 = \text{np.multiply}(a^3, d^3)$$

这条公式等价于

$$a^3 *= d^3$$

它的作用就是过滤 $d^3$ 中所有等于0的元素，而各个元素等于0的概率只有0.2，乘法运算最终把 $d^3$ 中相应元素归零，我们又有公式

$$a^3 /= \text{keep-prob}$$

这个公式是因为，从50个矩阵单元到10个矩阵单元，因为我们有公式

$$z^{(4)} = w^{(4)} a^{(3)} + b^{(4)}$$

因为 $a^3$ 减少了百分之二十的矩阵，所以我们需要把 $a^3$ 除以0.8，它将会修正或弥补我们所需的那0.2。 $a^3$ 的期望值不变。

# Making predictions at test time

$$a^{(0)} = X$$

No drop out.

$$z^{(1)} = w^{(1)} a^{(0)} + b^{(1)}$$

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

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

$$a^{(2)} = \dots$$

$$\downarrow$$
  
 $\hat{y}$



Andrew Ng

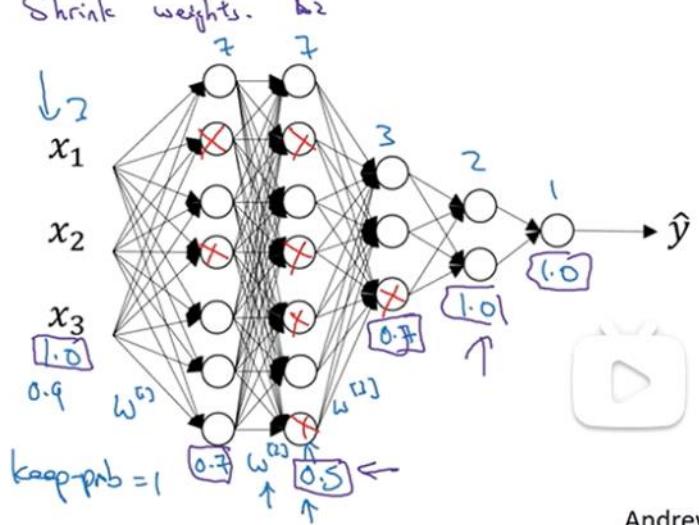
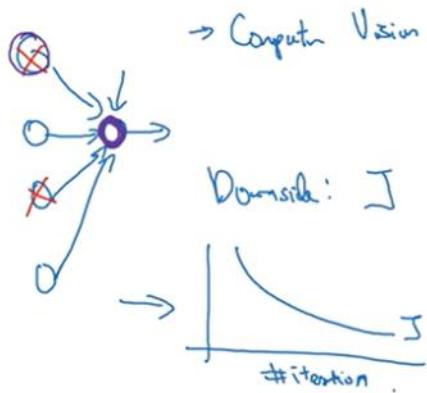
我们在测试集不使用Dropout正则化。

## (12) 理解Dropout

2022年2月18日 17:22

### Why does drop-out work?

Intuition: Can't rely on any one feature, so have to spread out weights.  $\rightarrow$  Shrink weights.  $b_2$



我们已经对dropout随机删除网络中的神经单元有了一个直观的理解，好像每次迭代之后，神经网络都会变得比以前更小，因此采用一个较小的神经网络好像和使用正则化的效果是一样的，因为我们无法去信赖于任何特征，所以我们去加大权重，dropout将产生收缩权重的平方范数的效果。和之前的L2正则化相似，实施Dropout的结果是它会压缩权重并完成一些预防过拟合的外层正则化，事实证明Dropout被正式作为一种正则化的替代形式。

总的来说，Dropout与L2的正则化是类似的，与L2正则化不同的是被应用的方式不同，Dropout也会有所不同，甚至更适用于不同的输入范围。实施Dropout的另一个细节是，其中一个要选择的参数是keep-prob，它代表每一层上保留单元的概率，所以不同层的keep-prob也可以变化。

注:Dropout=1代表保留所有的单元，其中我们在输入层和输出层一般不使用Dropout。

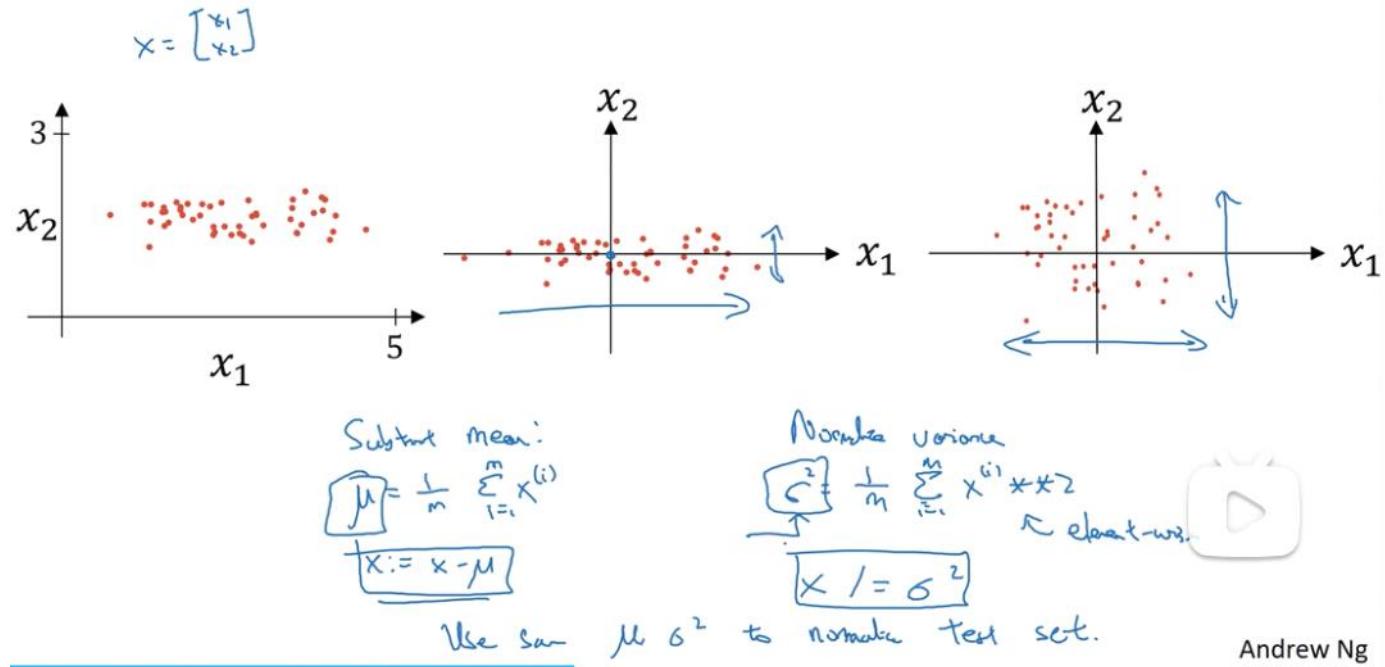
我们通常先使用梯度下降算法，并通过可视化显示保证代价函数有

在下降，然后再使用Dropout。

## (13) 归一化输入

2022年2月18日 20:01

### Normalizing training sets



### 标准化训练集

归一化输入需要两步

第一步是需要均值化(图1变图2), 我们使用公式

$$\bar{U} = \frac{1}{m} \sum_{i=1}^m x^{(i)}$$

$$x := x - \bar{U}$$

第二步是需要归一化方差(图2 变图3), 因为特征 $x_1$ 的方差要比 $x_2$ 的方差要大得多, 所以需要归一化方差

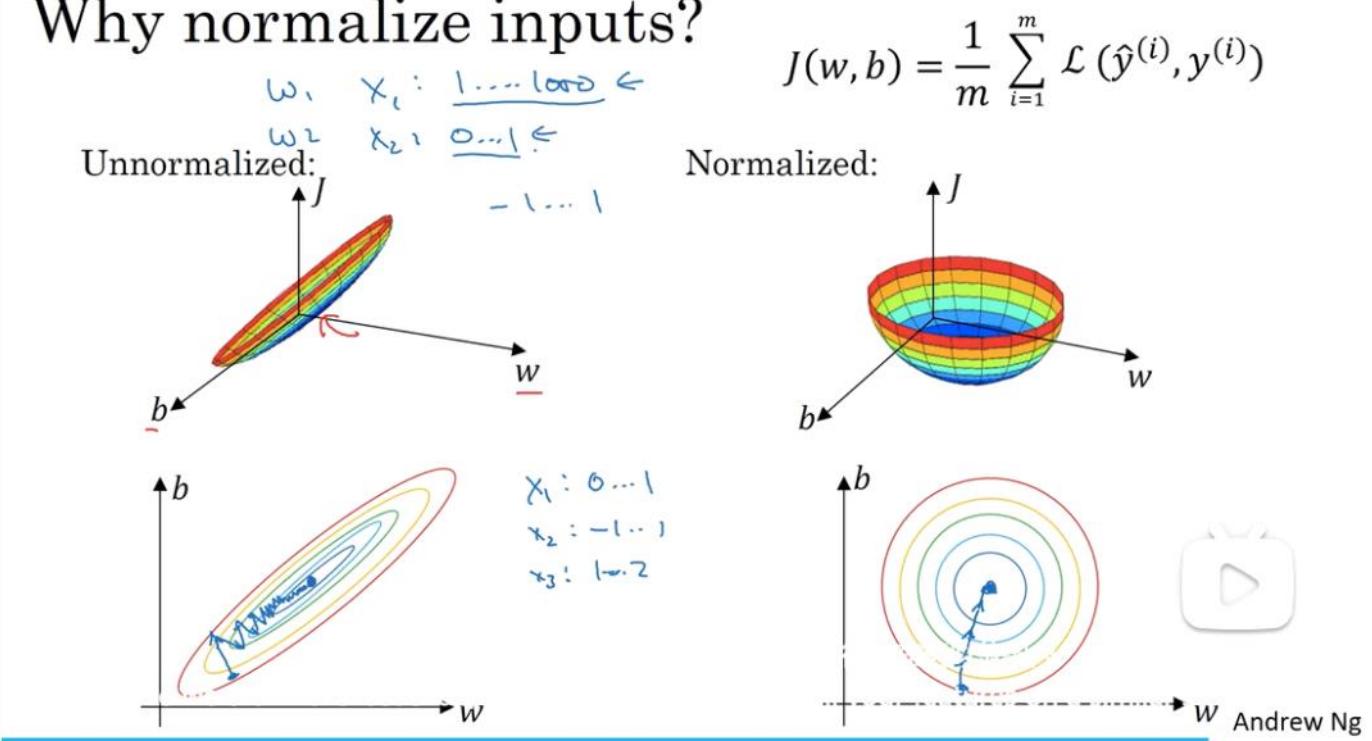
$$\sigma^2 = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \bar{U})^2$$

$$x^{(i)} = \frac{x^{(i)} - \bar{U}}{\sigma}$$

最后 $x_1$ 和 $x_2$ 的方差都为1。

注:需要在同样的集合中进行上述公式的操作。

# Why normalize inputs?



为什么需要归一化输入?

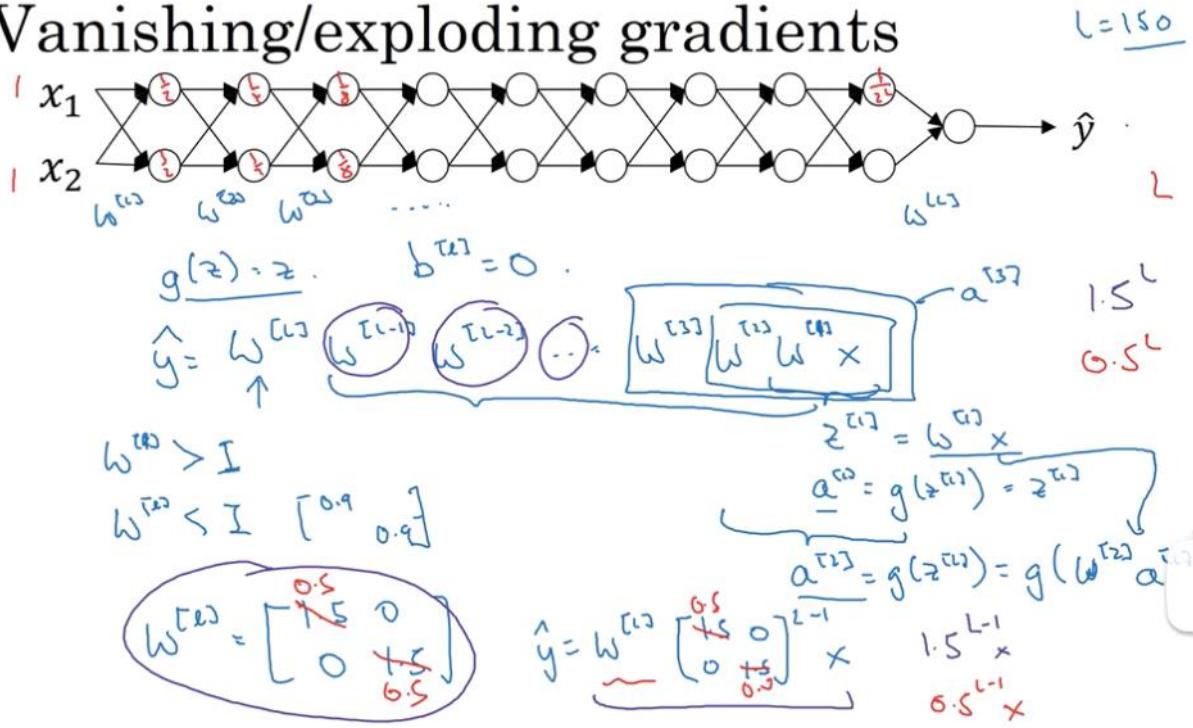
因为在非归一化输入时,如左图,它在使用梯度下降时,会比归一化输入的梯度下降所用的时间要慢。

## (14) 梯度爆炸与梯度消失

2022年2月18日 20:20

在训练神经网络尤其是深度神经网络时，所面临的一个问题是梯度消失或梯度爆炸，即当你训练网络时导数或坡度变得非常大或者非常小，甚至以指数方式变小，这样加大了训练难度，这节将使你更加明智的选择随机初始化权重，从而避免这个问题。

### Vanishing/exploding gradients



假设在训练一个极深的神经网络，假设我们有参数 $w^{(1)}, \dots, w^{(l)}$ ，且有公式和参数

$$g(z) = z$$

$$b^{(l)} = 0$$

且有公式

$$\hat{y} = w^{(l)}w^{(l-1)}w^{(l-2)} \dots w^{(1)}x$$

$$z^{(1)} = w^{(1)}x$$

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

$$a^{(2)} = g(z^{(2)}) = g(w^{(2)}a^{(1)})$$

又我们有参数矩阵

$$w^{(l)} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}$$

故有公式

$$\hat{y} = w^{(l)} \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^{l-1} x$$

这样会遭成梯度爆炸，对于一个神经网络而言在如此矩阵中，若 $L$ 较大，那么 $\hat{y}$ 的值也会非常大，它将呈指数级增长，它增长率为 $1.5^l$   
 $\hat{y}$ 的值将爆炸式的增长，相反，如果它的权重为0.5即

$$\hat{y} = w^{(l)} \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}^{l-1} x$$

效果见图中的神经网络，在这个神经网络中，激活函数以指数级递减。

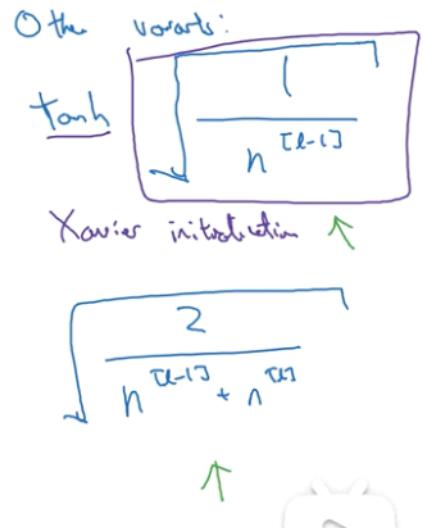
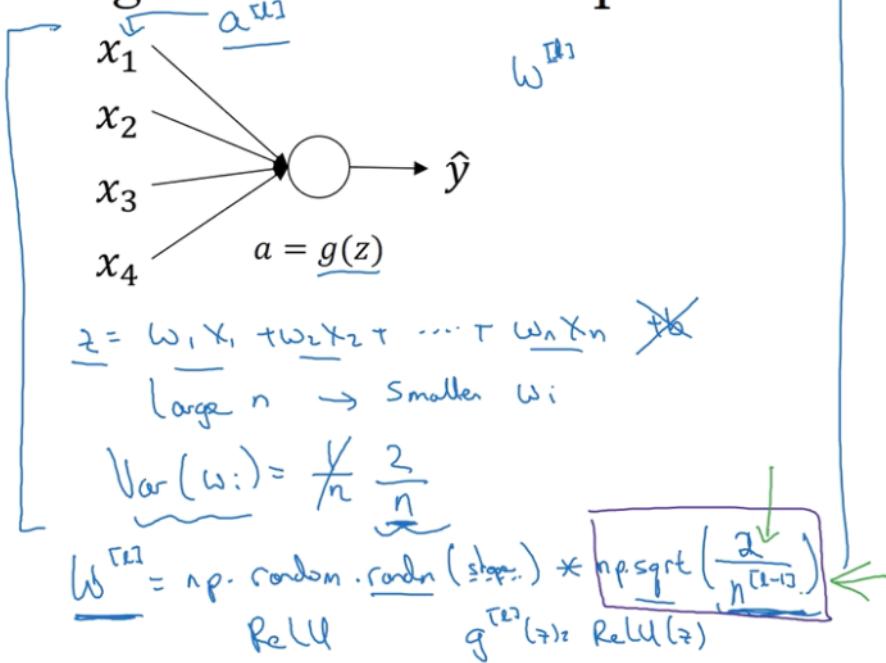
总的来说，当参数 $w^{(l)} > 1$ 时，可能会出现梯度爆炸，当参数 $w^{(l)} < 1$ 时，可能会出现梯度消失。

## (15) 神经网络的权重初始化

2022年2月19日 20:09

我们对于梯度爆炸与梯度消失，想出了一个不完整的解决方案，虽然不能彻底解决问题，但是它却有用，有助于我们为神经网络更谨慎地选择随机初始化参数。

### Single neuron example



Andrew Ng

我们先来看一个神经单元权重初始化的例子，然后再演变到整个深度网络，我们的神经单元有四个输入特征 $x_1, \dots, x_4$ ，经过 $a = g(z)$ 处理，最终得到

$$z = w_1 x_1 + \dots + w_n x_n$$

我们暂时令 $b=0$ ，暂时忽略 $b$ ，为了预防 $z$ 值过大或过小，我们可以看到 $n$ 越大，我们希望 $w_1$ 越小，因为 $z$ 是由 $x$ 和 $w$ 组合而成，所以我们希望式子中的每项值更小，最合理的方法就是设置

$$\text{var}(w_i) = \frac{1}{n}$$

其中 $n$ 是神经元的输入特征数量，即

$$w^{(l)} = w^{(l)} * \left(\frac{1}{n^{(l-1)}}\right)^2$$

若我们使用的是ReLU激活函数，我们使用

$$\text{var}(w_i) = \frac{2}{n}$$

在本例中，逻辑回归的特征是不变的，但一般情况下，L层上的每个神经元都有 $n(L-1)$ 个输入，如果激活函数的输入特征被零均值，标准方差，方差是1， $z$ 也会被调整到相似的范围，这虽然没有彻底解决梯度爆炸与梯度消失，但它确实降低了坡度消失和爆炸问题，因为它给权重矩阵W设置了合理值，因为它不能比1大很多，也不能比1小很多，所以梯度没有爆炸或消失过快，若你要使用tanh函数，则要使用

$$\sqrt{\frac{1}{n^{(l-1)}}}$$

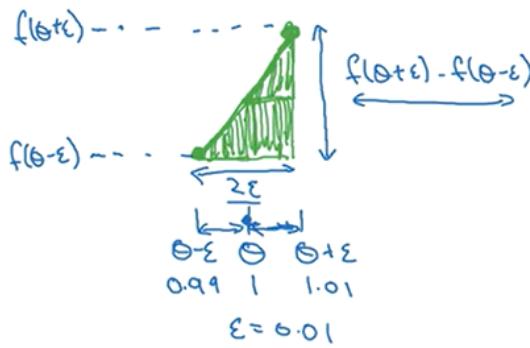
## (15) 梯度数值逼近

2022年2月19日 20:52

在实施backprop时，有一个测试叫作梯度检验，它的作用是确保backprop的正确实施，为了渐近实施梯度逼近，我们先来谈谈如何对计算梯度做数值逼近。

### Checking your derivative computation

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



$$\frac{f(\theta+\epsilon) - f(\theta-\epsilon)}{2\epsilon} \approx g(\theta)$$

$$\frac{(1.01)^3 - (0.99)^3}{2(0.01)} = 3.0001 \approx 3$$

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

approx error: 0.0001

(prev slide: 3.0301. error: 0.03)

$$\left\{ f'(\theta) = \lim_{\epsilon \rightarrow 0} \frac{f(\theta+\epsilon) - f(\theta-\epsilon)}{2\epsilon} \right. \quad \begin{matrix} O(\epsilon^2) \\ 0.01 \\ 0.0001 \end{matrix} \quad \left| \quad \frac{f(\theta+\epsilon) - f(\theta)}{\epsilon} \right. \quad \begin{matrix} \text{error: } O(\epsilon) \\ 0.01 \end{matrix}$$

Andrew

我们一般使用双边误差，而不使用单边误差，因为双边误差要比单边误差更精确。

注：这部分感觉有错误，暂时不清楚。

## (16) 梯度检测注意事项

2022年2月19日 21:22

# Gradient checking implementation notes

- Don't use in training – only to debug  
 $\frac{\partial \text{loss}[\cdot]}{\partial \theta_{\text{app}}[\cdot]} \longleftrightarrow \frac{\partial \text{loss}[\cdot]}{\partial \theta[\cdot]}$
- If algorithm fails grad check, look at components to try to identify  
 $\frac{\partial b}{\partial \theta}$      $\frac{\partial w}{\partial \theta}$
- Remember regularization.  
 $J(\theta) = \frac{1}{m} \sum_i \ell(y^{(i)}, \hat{y}^{(i)}) + \frac{\lambda}{2m} \sum_j \|w^{(j)}\|_F^2$   
 $\delta\theta = \text{gradient of } J \text{ wrt. } \theta$
- Doesn't work with dropout.     $J$     keep\_prob = 1.0
- Run at random initialization; perhaps again after some training.

第一、我们不使用在训练集中，我们只用这个去调试看反向传播是否正确。

第二、如果算法校验失败，查看部件并尝试去识别。

第三、如果使用正则化，请注意正则项

第四、不要使用dropout正则化

第五、

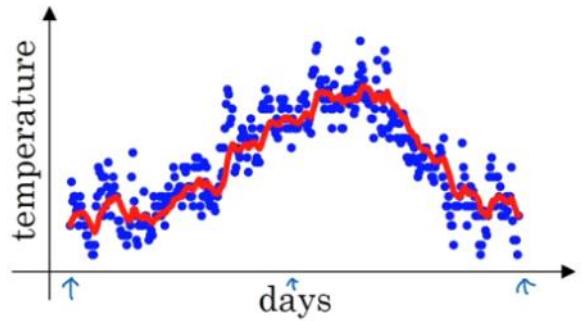
## (1) 指数加权平均

2022年2月20日 11:51

我们将展示几个优化算法，它们比梯度下降算法要快，要理解这些算法，我们会用到时指数加权平均。

### Temperature in London

$$\begin{aligned}\theta_1 &= 40^{\circ}\text{F} \quad 4^{\circ}\text{C} \\ \theta_2 &= 49^{\circ}\text{F} \quad 9^{\circ}\text{C} \\ \theta_3 &= 45^{\circ}\text{F} \quad : \\ &\vdots \\ \theta_{180} &= 60^{\circ}\text{F} \quad 15^{\circ}\text{C} \\ \theta_{181} &= 56^{\circ}\text{F} \quad : \\ &\vdots\end{aligned}$$



$$\begin{aligned}v_0 &= 0 \\ v_1 &= 0.9 v_0 + 0.1 \theta_1 \\ v_2 &= 0.9 v_1 + 0.1 \theta_2 \\ v_3 &= 0.9 v_2 + 0.1 \theta_3 \\ &\vdots \\ v_t &= 0.9 v_{t-1} + 0.1 \theta_t\end{aligned}$$



我们有天气数据如上图所示，看起来有些杂乱，如果要计算趋势的话，也就是温度的局部平均值或者说移动平均值。我们要做的是首先使，每天需要使用0.9的加权数之前的数值，加上当日温度的0.1

$$v_0 = 0$$

$$v_1 = 0.9 v_0 + 0.1 \theta_1$$

$$v_2 = 0.9 v_1 + 0.1 \theta_2$$

$\vdots$

$$v_t = 0.9 v_{t-1} + 0.1 \theta_t$$

这样计算，我们用红线作图，便得到了移动平均值，每日温度的指数加权平均值。

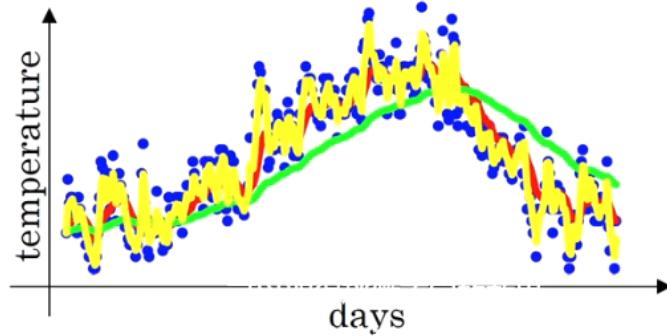
# Exponentially weighted averages

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

$\beta = 0.9$  :  $\approx 10$  days' tapering.  
 $\beta = 0.98$  :  $\approx 50$  days  
 $\beta = 0.5$  :  $\approx 2$  days

$V_t$  is approximately  
 Average over  
 $\rightarrow \approx \frac{1}{1-\beta}$  days'  
 temperature.

$$\frac{1}{1-0.98} = 50$$



对于  $V_t$  我们有公式

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

我们有公式

$$\frac{1}{1 - \beta}$$

代表  $\frac{1}{1-\beta}$  天的平均温度。

如若  $\beta = 0.9$ , 则有

$$\frac{1}{1 - 0.9} = 10$$

那么  $V_t$  公式代表 10 天的平均温度。

## (2) 理解指数加权平均

2022年2月20日 14:50

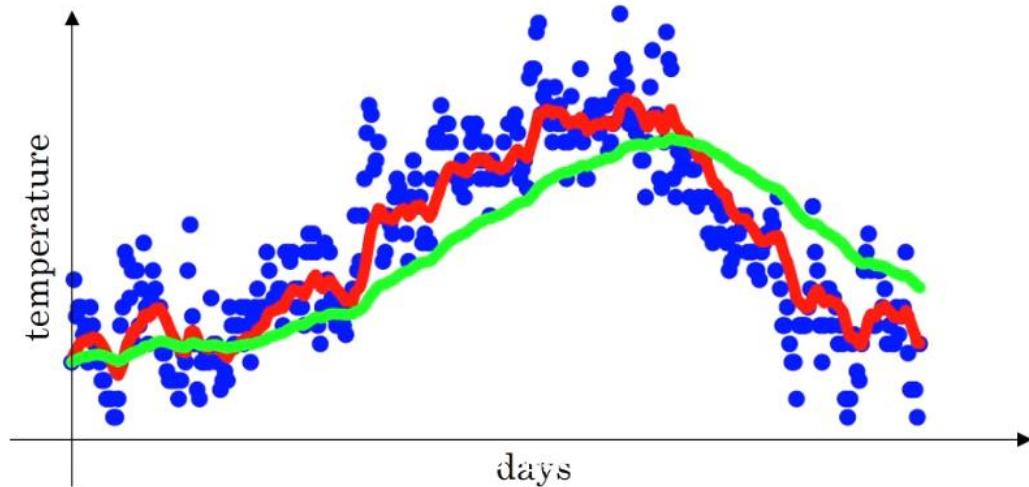
### Exponentially weighted averages

$$v_t = \beta v_{t-1} + (1 - \beta) \theta_t$$

$\beta = 0.9$

$0.98$

$0.5$



Andrew I

红色条线是当参数为 $\beta = 0.9$ 的结果，绿色条件是当参数为 $\beta = 0.98$ 的结果。

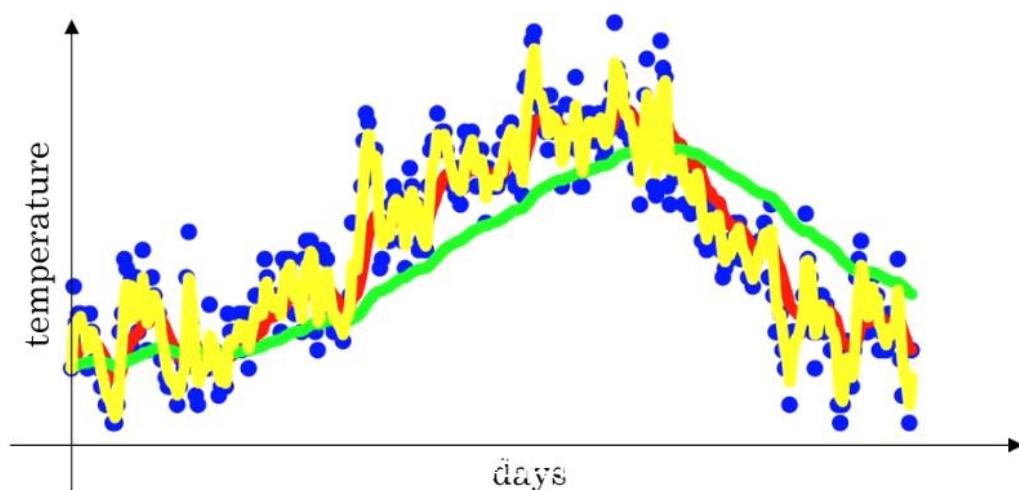
### Exponentially weighted averages

$$v_t = \beta v_{t-1} + (1 - \beta) \theta_t$$

$\beta = 0.9$

$0.98$

$0.5$



黄色线条是当参数 $\beta = 0.5$ 的结果。

## Exponentially weighted averages



$$v_t = \beta v_{t-1} + (1 - \beta) \theta_t$$

$$v_{100} = 0.9 v_{99} + 0.1 \theta_{100}$$

$$v_{99} = 0.9 v_{98} + 0.1 \theta_{99}$$

$$v_{98} = 0.9 v_{97} + 0.1 \theta_{98}$$

...

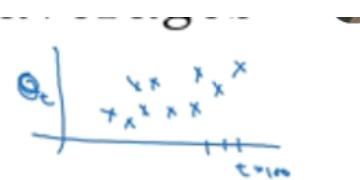
$$\begin{aligned} v_{100} &= 0.1 \theta_{100} + 0.9 (0.1 \theta_{99} + 0.9 (0.1 \theta_{98} + \dots)) \\ &= 0.1 \theta_{100} + 0.1 \times 0.9 \cdot \theta_{99} + 0.1 (0.9)^2 \theta_{98} + 0.1 (0.9)^3 \theta_{97} + \dots \end{aligned}$$

99号 $\theta$  98号 $\theta$  97号 $\theta$  等等

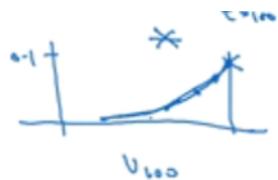
θ 99, θ 98, θ 97, θ 96, and so on.

Andrew Ng

我们有公式拆解如上。



假设我们有一些日期的温度，横轴是 $t$ ，纵轴是 $\theta$ ，所以100号 $\theta$ 有个数值，99号 $\theta$ 有个数值...，这些是数日的温度数值。



然后我们构建一个指数衰减函数，从0.1开始到0.9乘以0.1，到0.9的平方乘以0.1，以此类推，所以就有了这个指数衰减函数。

最后你可能想知道的是到底需要平均多少天的温度，实际上

$$0.9^{10} \approx \frac{1}{e}$$

这表明十天之后，下降三分之一，相当于峰值的 $1/e$ ，又因当 $\beta = 0.9$ 的时候，我们说仿佛在计算一个指数加权平均数，只关注了过去10天的温度，因为十天后权重下降到不到当日权重的三分之一。我们又有

$$0.98^{50} \approx \frac{1}{e}$$

即 $0.98$ 的50次方大约等于 $1/e$ ，所以前50天这个数值比 $1/e$ 大，数值会快速衰减，所以本质上这是一个下降幅度很大的函数。可以看作平均了50天的温度。

## Implementing exponentially weighted averages

云课堂

$$v_0 = 0$$

$$v_1 = \beta v_0 + (1 - \beta) \theta_1$$

$$v_2 = \beta v_1 + (1 - \beta) \theta_2$$

$$v_3 = \beta v_2 + (1 - \beta) \theta_3$$

...

$$V_\theta := 0$$

$$V_\theta := \beta V + (1-\beta) \theta,$$

$$V_\theta := \beta V + (1-\beta) \theta_2$$

⋮

$$\rightarrow V_\theta = 0$$

Repeat

Get next  $\theta_t$

$$V_\theta := \beta V_\theta + (1-\beta) \theta_t$$

所以在接下来的视频中

So for things, we'll see some examples on the next few videos,

Andrew

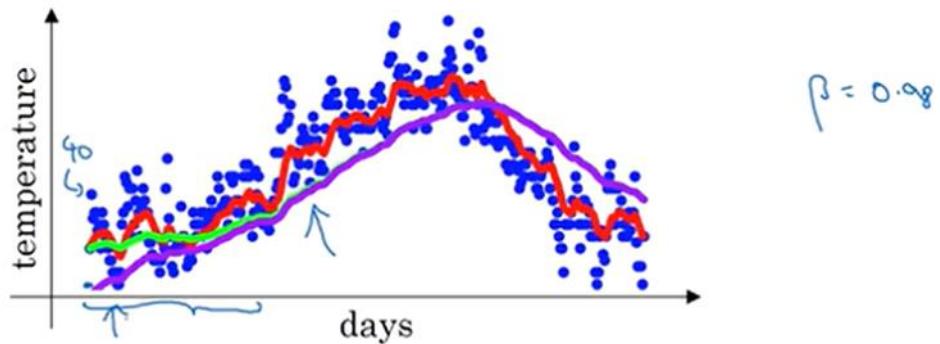


指数加权平均的优势在于内存占用较小，不用保存n个天气的数值以及求和。

### (3) 指数加权平均的偏差修正

2022年2月20日 15:31

## Bias correction



$$\rightarrow v_t = \beta v_{t-1} + (1 - \beta) \theta_t$$

$$v_0 = 0$$

$$v_1 = 0.98 v_0 + 0.02 \theta_1$$

$$\begin{aligned} v_2 &= 0.98 v_1 + 0.02 \theta_2 \\ &= 0.98 \times 0.02 \times \theta_1 + 0.02 \theta_2 \\ &= 0.0196 \theta_1 + 0.02 \theta_2 \end{aligned}$$

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

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

$$\frac{v_2}{0.0396} = \frac{0.0196 \theta_1 + 0.02 \theta_2}{0.0396}$$

Andrew Ng

红色的线条对应着  $\beta = 0.9$ , 绿色的线条对应着  $\beta = 0.98$ , 而我们执行公式

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

我们得到的是紫色的线条, 我们可以知道  $v_2$  不能很好的估算这一年的前两天的温度, 所以我们使用公式

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

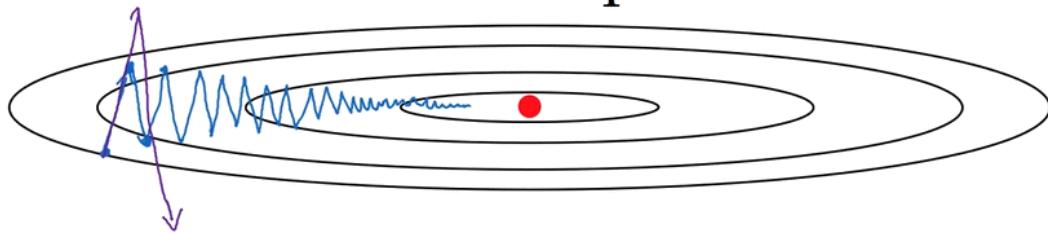
来进行修正, 使得从紫色的线条变成绿色的线条, 执行指数加权平均可能会导致起始阶段有一定的误差, 所以使用偏差修正, 但是当  $t$  持续增大时, 偏差修正几乎没有任何作用了, 若关注于初始时的偏差可以使用, 若不关注可不使用。

## (4) 动量梯度下降法

2022年2月20日 16:15

动量梯度下降法运行速度几乎总是快于标准的梯度下降法，基本原理是计算梯度的指数加权平均数并利用该梯度更新权重。

### Gradient descent example



在纵轴上 你希望学习慢一点

on the vertical axis you want your learning to be a bit slower,



当我们运行梯度下降法时，如上图所示，假设我们开始运行梯度下降法，如果进行梯度下降法的一次迭代，无论是batch或mini-batch下降法，在进行一次次的迭代中慢慢摆进最小值，这种上下波动减缓了梯度下降法的速度，我们就无法使用更大的学习率，如果要用更大的学习率，结果可能偏差函数的范围，以了避免摆动过大，我要需要用一个较小的学习率，从另一个角度看，在纵轴上我们希望学习得慢一些，因为不想要这些摆动，但是在横轴上，我们希望加快学习，我们希望快速从左向右移动，移向最小值，所以运用Momentum梯度下降法，动量梯度下降法我们给这个碗状的目标函数一个加速度。

# Implementation details

$$V_{dw} = 0, V_{db} = 0$$

On iteration  $t$ :

Compute  $dW, db$  on the current mini-batch

$$v_{dw} = \beta v_{dw} + (1 - \beta) \underline{dW}$$

$$v_{db} = \beta v_{db} + (1 - \beta) \underline{db}$$

$$W = W - \alpha v_{dw}, b = \underline{b} - \alpha v_{db}$$

~~$V_{dw}$~~   
 ~~$V_{db}$~~



Hyperparameters:  $\alpha, \beta$

↑↑

$$\beta = 0.9$$

~~average over last  $\approx 10$  gradients~~

Andrew

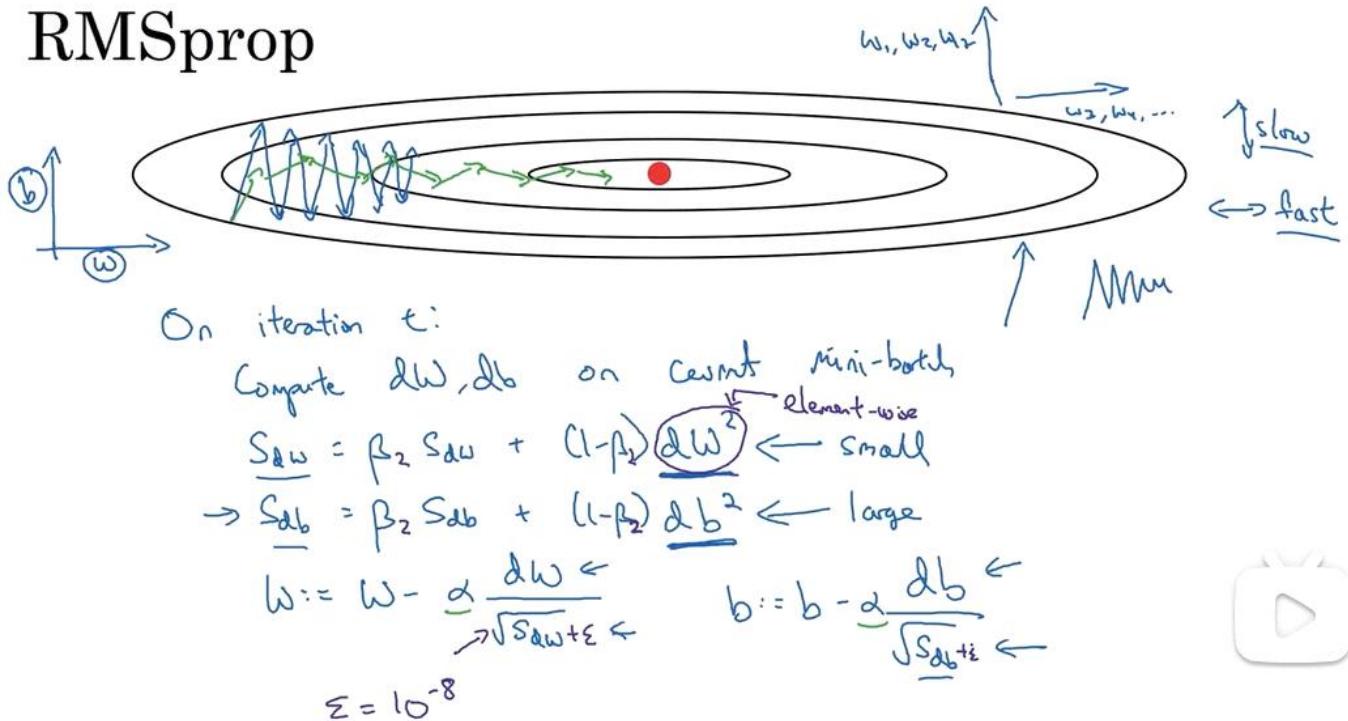
实现算法如上图所示，其中  $V_{dw} = 0, V_{db} = 0$ 。我们比较常令  $\beta = 0.9$ ，其中参数  $a$  表示学习率

## (5) RMSprop 算法

2022年2月20日 19:13

RMSprop 算法也可以加快梯度下降。

### RMSprop



Andrew

我们假设  $w$  是横轴，而  $b$  是纵轴，我们有公式

On iteration  $t$ :

Compute  $dW, db$  on mini-batch

$$S_{dw} = \beta_2 S_{dw} + (1 - \beta_2) (dW)^2$$

$$S_{db} = \beta_2 S_{db} + (1 - \beta_2) (db)^2$$

$$w := w - \alpha \frac{dW}{\sqrt{S_{dw} + \varepsilon}}$$

$$b := b - \alpha \frac{db}{\sqrt{S_{db} + \varepsilon}}$$

我们需要把  $db$  设置成一个较大的数，把  $dW$  设置成一个较小的数，因为从函数倾斜角度上来看，在纵轴上也就是  $b$  方向上要大于横轴上，也就是  $w$  方向上， $(db)^2$  较大，所以  $S_{db}$  也会比较大，相比之下  $dW$  比较小，也就是  $S_{dw}$  比较小，结果就是纵轴上的更新要被除以一个较大的数，这样就能消除摆动，而水平方向的更新被除以一个较小的数，其中

参数 $\epsilon = 10^{-8}$ 比较合理。

## (6) Adam优化算法

2022年2月20日 19:36

Adam算法是将Momentum算法与RMSprop算法进行结合。

### Adam optimization algorithm

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

On iterator t:

Compute  $\delta w, \delta b$  using current mini-batch

$$V_{dw} = \beta_1 V_{dw} + (1 - \beta_1) \delta w, \quad V_{db} = \beta_1 V_{db} + (1 - \beta_1) \delta b \quad \leftarrow \text{"moment"} \beta_1$$

$$S_{dw} = \beta_2 S_{dw} + (1 - \beta_2) \delta w^2, \quad S_{db} = \beta_2 S_{db} + (1 - \beta_2) \delta b^2 \quad \leftarrow \text{"RMSprop"} \beta_2$$

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

$$S_{dw}^{correct} = S_{dw} / (1 - \beta_2^t), \quad S_{db}^{correct} = S_{db} / (1 - \beta_2^t)$$

$$w := w - \alpha \frac{V_{dw}^{correct}}{\sqrt{S_{dw}^{correct} + \epsilon}}$$

$$b := b - \alpha \frac{V_{db}^{correct}}{\sqrt{S_{db}^{correct} + \epsilon}}$$



我们有Adam算法

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

On iterator t:

使用mini-batch 计算  $dw, db$

$$V_{dw} = \beta_1 V_{dw} + (1 - \beta_1) dw$$

$$V_{db} = \beta_1 V_{db} + (1 - \beta_1) db$$

$$S_{dw} = \beta_2 S_{dw} + (1 - \beta_2) (dw)^2$$

$$S_{db} = \beta_2 S_{db} + (1 - \beta_2) (db)^2$$

然后进行偏差修正

$$V_{dw}^{corrord} = \frac{V_{dw}}{(1 - \beta_1^t)}$$

$$V_{db}^{corrord} = \frac{V_{db}}{(1 - \beta_1^t)}$$

$$S_{dw}^{corrord} = \frac{S_{dw}}{(1 - \beta_2^t)}$$

$$V_{db}^{corrord} = \frac{S_{db}}{(1 - \beta_2^t)}$$

$$W := W - a \frac{V_{dw}^{coorord}}{\sqrt{S_{dw}^{coorord} + \epsilon}}$$

$$b := b - a \frac{V_{db}^{coorord}}{\sqrt{S_{db}^{coorord} + \epsilon}}$$

这里的参数 $\beta_1$ 为momentum梯度下降的参数，这里的参数 $\beta_2$ 为RMSprop梯度下降的参数。

## Hyperparameters choice:

- $\rightarrow \alpha$ : needs to be tune
- $\rightarrow \beta_1$ : 0.9  $\rightarrow (\underline{du})$
- $\rightarrow \beta_2$ : 0.999  $\rightarrow (\underline{dw^2})$
- $\rightarrow \epsilon$ :  $10^{-8}$

Adam: Adaptive moment estimation



我们对超参数的选择有

参数 $a$ 需要经过一系列的调试寻找合适的值

参数 $\beta_1$ 推荐使用0.9

参数 $\beta_2$ 推荐使用0.999

参数 $\epsilon$ 推荐使用 $10^{-8}$

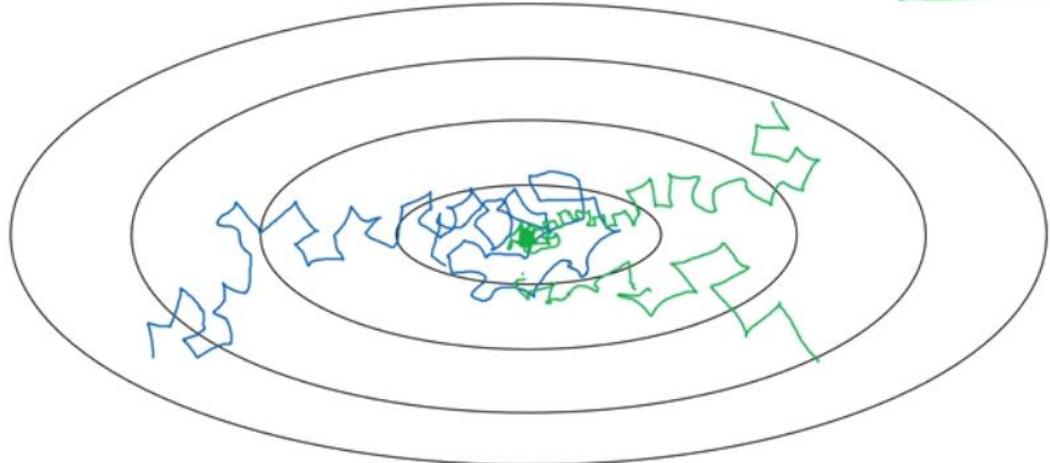
## (7) 学习率衰退

2022年2月21日 12:26

加快学习速率的另一个方法是随着时间慢慢减少学习率。

### Learning rate decay

Slowly reduce  $\alpha$



小一些的学习率能让你步伐小一些

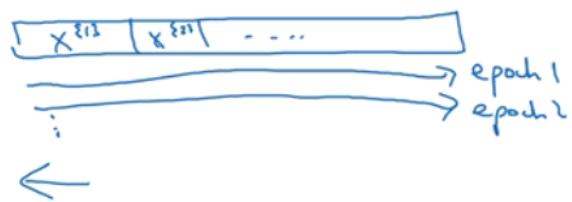
我们有在梯度下降的过程中可能会产生噪音的情况，然后因为学习率比较大大会在最小值的地方进行徘徊，而若我们在接近最小值的地方，减小学习率 $\alpha$ 就会更加精确最小值。

# Learning rate decay

1 epoch = 1 pass through data.

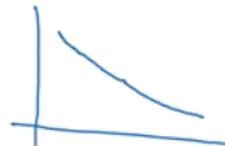
$$\alpha = \frac{1}{1 + \text{decay-rate} * \text{epoch-num}} \alpha_0$$

Epoch	$\alpha$
1	0.1
2	0.67
3	0.5
4	0.4
:	:



$$\alpha_0 = 0.2$$

$$\text{decay-rate} = 1$$



人们还会用其它的公式

there are a few other ways that people use.

我们有学习率公式

$$\alpha = \frac{1}{1 + \text{decag-rate} * \text{epoch-num}} \alpha_0$$

我们需要手动调整 $\text{decag-rate}$ 和 $\text{epoch-num}$ 的值，图中有一个例子。

## Other learning rate decay methods

forms

$$\left\{ \begin{array}{l} \alpha = 0.95^{\text{epoch-num}} \cdot \alpha_0 \quad - \text{exponentially decay.} \\ \alpha = \frac{k}{\sqrt{\text{epoch-num}}} \cdot \alpha_0 \quad \text{or} \quad \frac{k}{\sqrt{t}} \cdot \alpha_0 \end{array} \right.$$

decent staircase

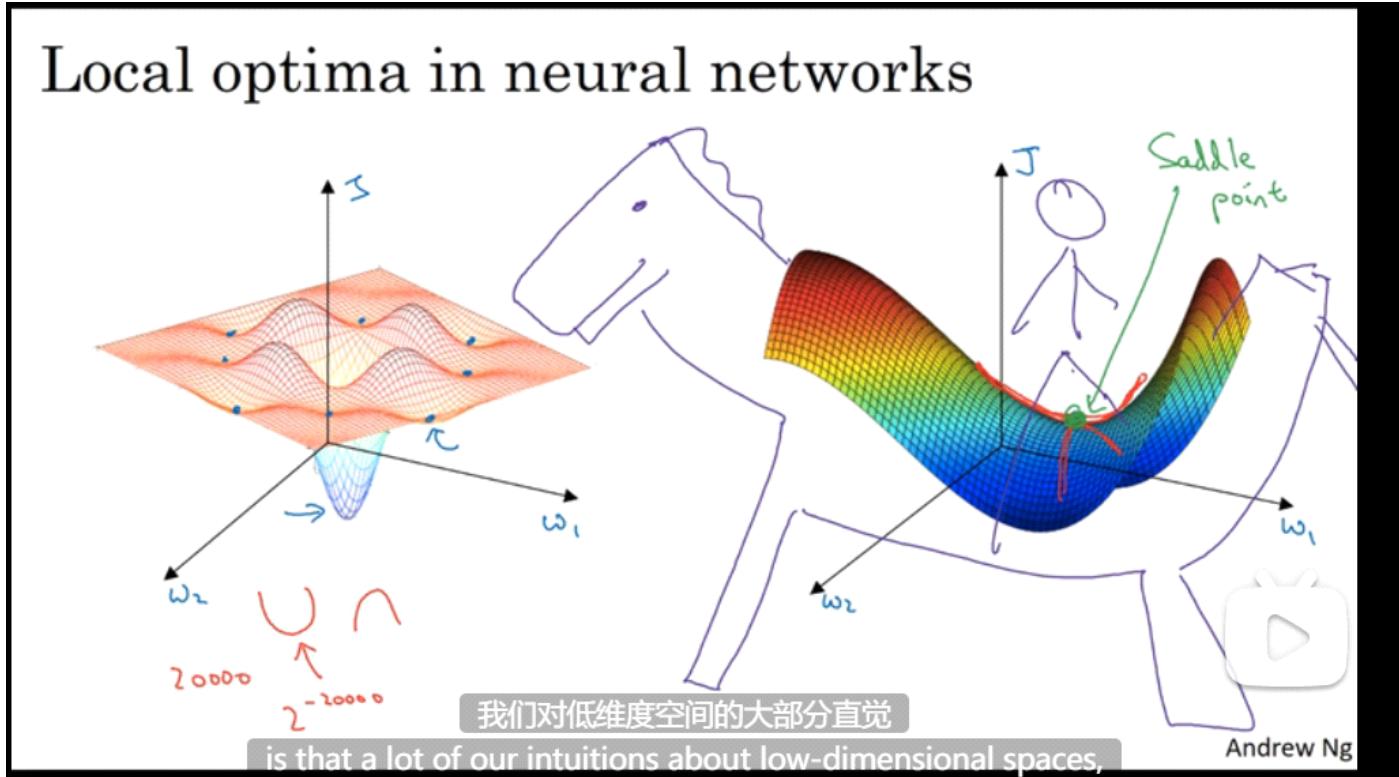
Manual decay. 但有的时候人们也会这么做



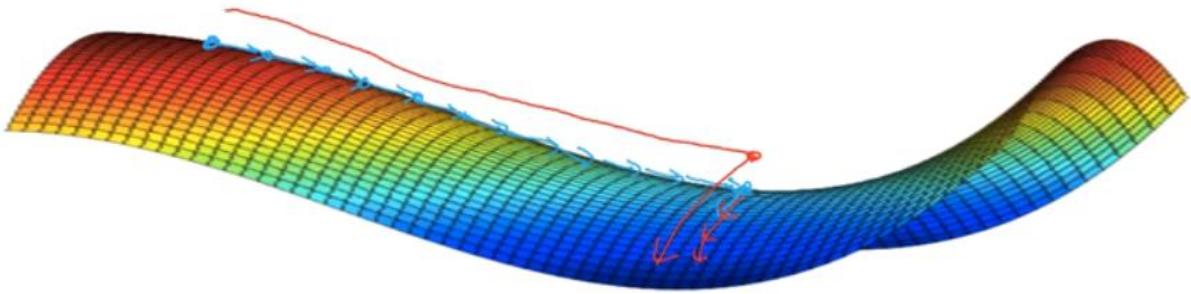
其它调整学习速率的公式如上图所示。

## (8) 局部最优问题

2022年2月21日 16:06



## Problem of plateaus



- Unlikely to get stuck in a bad local optima

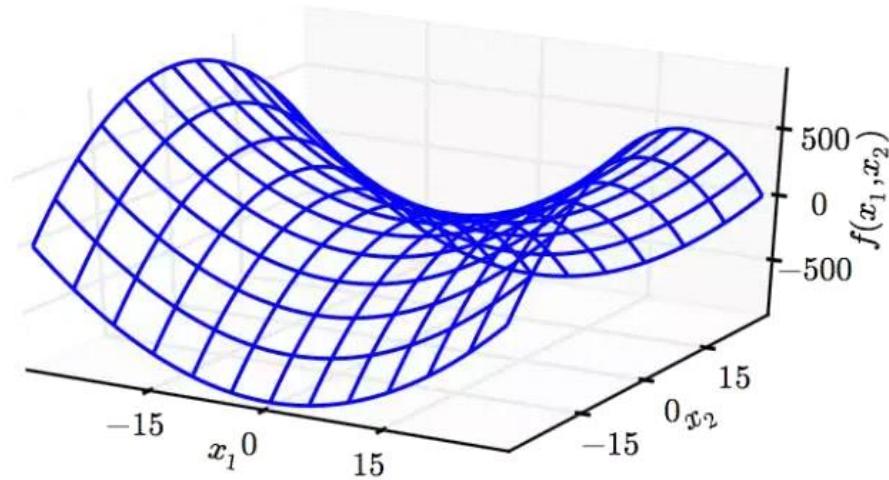


存在大量参数

在深度学习早期，人们常担忧是否会在梯度下降中会陷入局部最优，这是一种对于直观的错误，对于存在大量特征的神经网络中，会存在大量的鞍点。梯度下降会陷入鞍点

中，然后再跨过鞍点。

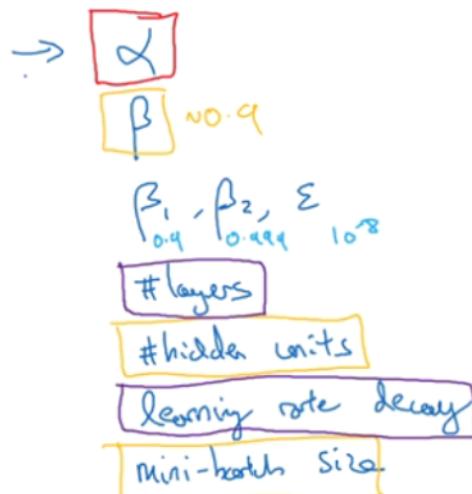
注：鞍点在二阶导的各个方面和维度不都是最小值，而极小值是。



## (9) 调试处理

2022年2月21日 16:46

# Hyperparameters



现在 如果你尝试调整一些超参数

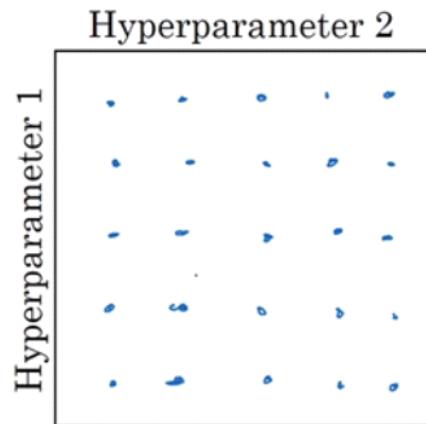
Now, if you're trying to tune some set of hyperparameters,

Andrew Ng



我们对于超参数的重要程度，分别为(由高到低)红、黄、蓝、紫。

Try random values: Don't use a grid



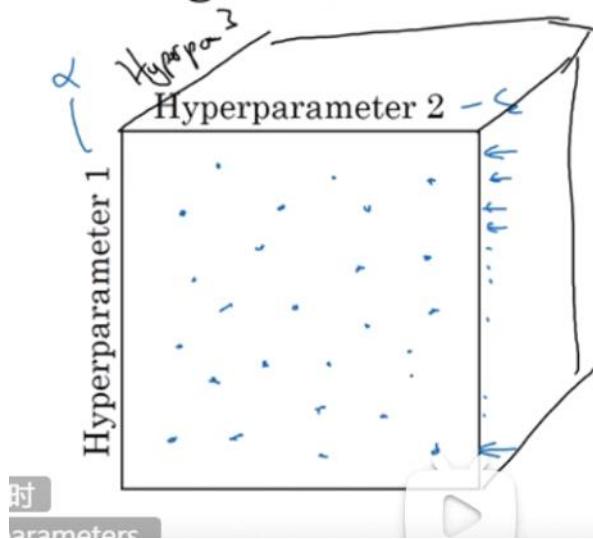
但对于这个例子 你可以尝试这所有的25个点 然后选择哪个参数效果最好

but you try out in this example all 25 points and then pick whichever hyperparameter works best.



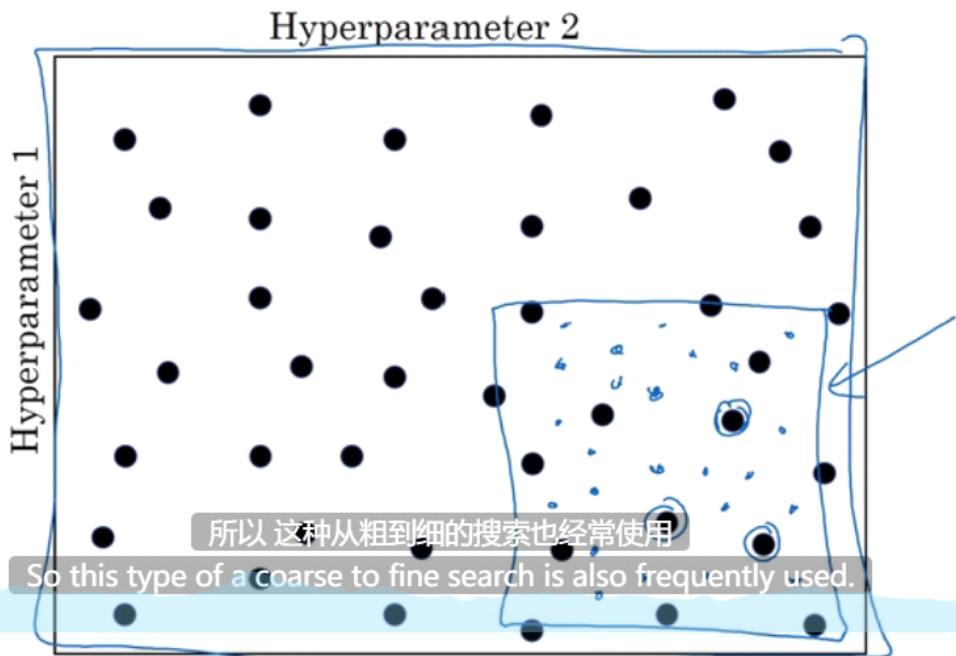
当参数较少时，我们可以尝试所有的点，然后选择哪个参数效果更好。

use a grid



另一种策略是，进行随机的选择超参数，这是因为我们无法确定哪一个参数更加重要，所以进行随机的选择，假设我们应用Adama优化算法，横轴为 $a$ ，纵轴为 $\varepsilon$ ，我们知道的是在Adama优化算法在极小范围内的取值作用不大，所以我们取横排的五个相同的 $a$ ，但 $\varepsilon$ 都不同的值，其结果却相差不大，若我们采用随机取值，对 $a$ 和 $\varepsilon$ 随机，这样的取值会使得结果大不相同，这样可以快速进行筛选。

## Coarse to fine



还有一种策略是从粗到细的策略，我们可以通过先找到一个效果还算不错的点，然后搜索其周围的点，来寻找更好的参数。

## (10) 为超参数选择合适的范围

2022年2月21日 17:44

在上一节的我们看到超参数范围内，随机聚类可以加速检索，但随机取值并不是在有效值范围内的随机均匀取值，而选择合适的步进值，用于探究这些超参数。

### Picking hyperparameters at random

$$\rightarrow n^{(l)} = 50, \dots, 100$$



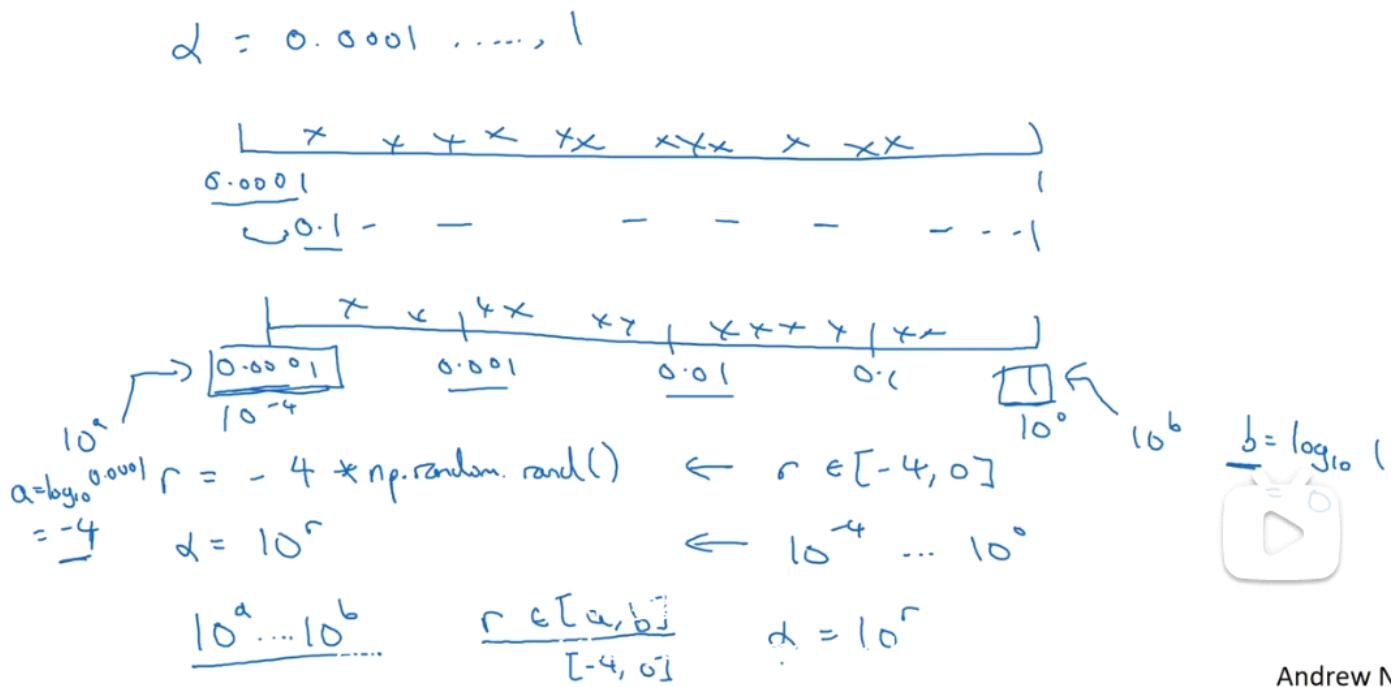
$$\rightarrow \# \text{layers } L : 2 - 4$$

2, 3, 4



假设我们要选取隐藏单元的数量  $n^{(l)}$ ，假设选择的范围是 50-100 中的某个值，我们可以随机的在这里面取值，或者我们要选择神经网络的层数，如在 2-4 里面都是比较合理的，不过这对于有些超参数而言就不太合适。

# Appropriate scale for hyperparameters



如上图所示，我们有一个超参数 $\alpha$ ，其取值在 $0.0001-1$ 之间，当我们对其进行搜索时，会得到会在 $0.1-1$ 之间使用百分之九十的资源，而在 $0.0001-0.1$ 之间只使用百分之十的资源去寻找合适的值，这显示是不合理的，因为可能合适的值就在于 $0.0001-0.1$ 之间，所以我们使用对数搜索法，这样就使得在 $0.0001-0.1$ 之间，有更多的资源去搜索，故我们有公式

$$a = 10^r$$

$$r \in [-4, 0]$$

$$r \in [a, b]$$

我们需要做的是在 $[a,b]$ 区间随机均匀的给 $r$ 取值。

# Hyperparameters for exponentially weighted averages

$$\beta = 0.9 \dots 0.999$$

$\downarrow$

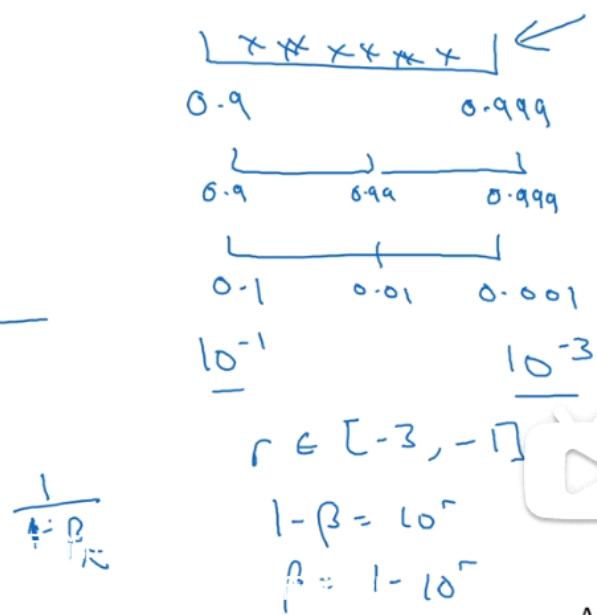
10                    1000

$$1 - \beta = 0.1 \dots 0.001$$

$$\beta: 0.900 \rightarrow 0.9005 \quad \} \sim 10$$

$$\beta: 0.999 \rightarrow 0.9995$$

$\sim 1000 \quad \sim 2000$



Andrew

另一个棘手的例子是给 $\beta$ 取值，假设我们认为 $\beta$ 是0.9-0.999之间的某个值，当计算指数的加权平均值时，取0.9就像在10个值中计算平均值，取0.999就像是在1000个值中取平均，如果想要0.9-0.999之间搜索，就不能使用线性轴取值，不要随机均匀的在此区间内取值，所以考虑这个问题的最好办法就是，探索公式

$$1 - \beta$$

这个取值范围在0.1-0.001之间，我们再次应用对数搜索法，就有

$$r \in [-3, -1]$$

$$1 - \beta = 10^r$$

$$\beta = 1 - 10^r$$

这样就变成了 $\beta$ 随机取值，那么为什么我们要这么做，为什么不用线性探索法呢？

这是因为当 $\beta$ 接近1时，所得的结果的灵敏度会变化，即使 $\beta$ 有微小的变化，如果 $\beta$ 在0.9-0.9005之间取值，将会无关紧要，所得的结果几乎不会变化，但若 $\beta$ 在0.999-0.9995之间取值，这将会对算法产生巨大的变化，因为 $\beta$ 在0.9-0.9005之间取值，是根据10个值取平均，而因为 $\beta$ 在0.999-0.9995之间取值，是从1000个值到2000个值取平均的值，因为公式

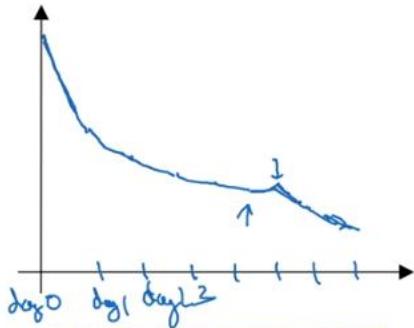
$$\frac{1}{1 - \beta}$$

当 $\beta$ 接近1时， $\beta$ 会对细微的变化变得很敏感，所以在整个取值过程中，需要更加密集地取值，当 $\beta$ 接近1的区间内，或者说当 $1 - \beta$ 接近于0时，这样就可以更加有效的分布取样点。

## (11) 单模型与平行多模型

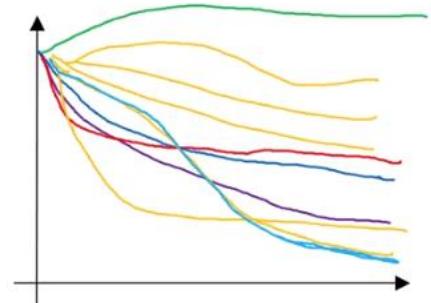
2022年2月22日 11:20

### Babysitting one model



Panda ↪

### Training many models in parallel



Caviar ↪

所以同时试验大量的模型是很困难的  
that it's difficult to train  
a lot of models at the same time.

Andrew N

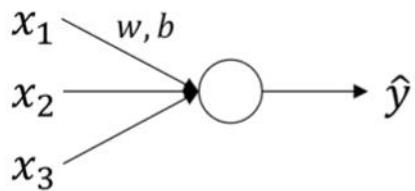
当我们计算机资源较少时，可以运行单个模型来不断调整超参数，而当计算机资源较多时，可以考虑运行平行的多个模型，来调整超参数。

# (1) 网络中的正规化激活函数

2022年2月22日 11:53

Batch归一化会使的参数搜索问题变得更加容易，使神经网络对超参数的选择更加稳定，超参数的范围会更庞大，工作效果也会更好，也会使我们很容易的训练甚至是深层网络。

## Normalizing inputs to speed up learning



$$\begin{aligned}\mu &= \frac{1}{m} \sum_i x^{(i)} \\ X &= X - \mu \\ \sigma^2 &= \frac{1}{m} \sum_i (x^{(i)})^2 \quad \text{← element-wise} \\ X &= X / \sigma^2\end{aligned}$$

A diagram showing two sets of data points. The first set is elongated and elliptical. The second set is more compact and circular. An arrow points from the first set to the second, illustrating the effect of normalization.



变成更圆的东西 更易于算法优化

to something that is more round and easier for an algorithm like grading to send to  
optimize.

在logistic回归中，我们常使用归一化

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

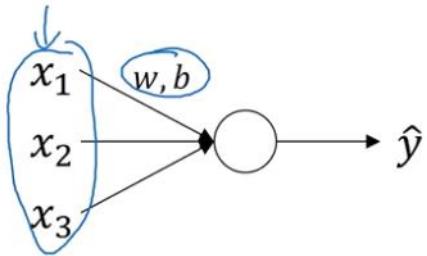
$$x = x - u$$

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

$$x = \frac{x}{\sigma^2}$$

通过归一化，我们可以使得狭长的东西变成更加圆的东西，更易于算法的优化。

# Normalizing inputs to speed up learning



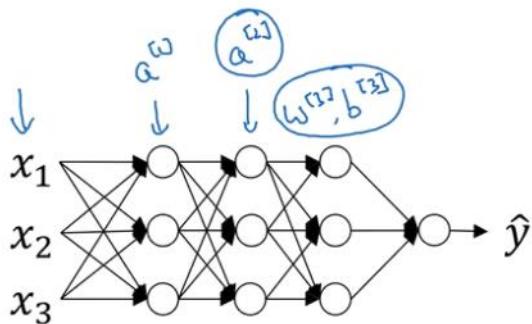
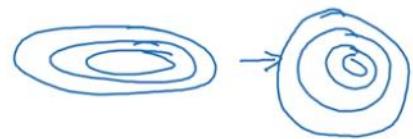
$$\mu = \frac{1}{m} \sum_i x^{(i)}$$

$$X = X - \mu$$

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

$$X = X / \sigma^2$$

← element-wise



Can we normalize  $\frac{\alpha^{[2]}}{w^{[2]}, b^{[2]}}$  so  
as to train  $w^{[2]}, b^{[2]}$  faster

Normalize  $\frac{z^{[2]}}{\uparrow}$



Andrew Ng

而我们在深层网络中，会把第二层的结果变成第三层输入，那么若我们想训练这些参数，如 $w^3, b^3$ ，那么我们对参数 $a^{(2)}$ 进行归一化会对参数 $w^3, b^3$ 起到更好的训练效果，如我们在logistic回归中，通过训练参数 $x$ 对参数 $w, b$ 都起到了更好的训练效果。

所以我们能否通过归一化参数 $a^{(2)}$ ，能更加快速的训练参数 $w^{(3)}, b^{(3)}$ ，因为 $a^{(2)}$ 是下一层的输入，所以就会影响 $w^{(3)}, b^{(3)}$ 的训练，这就是Batch归一化的作用。

我们严格来说真正归一化的不是 $a^{(2)}$ ，而是 $z^{(2)}$ ，在深度学习的文献中关于在激活函数之前是否就将值 $z^{(2)}$ 归一化，或者是否应该在应用失活函数 $a^{(2)}$ 后再规范值，在实践中，我们经常做的是归一 $z^{(2)}$ 。

# Implementing Batch Norm

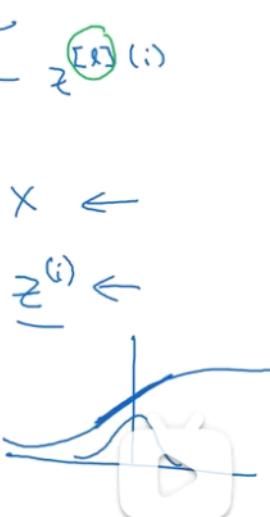
Given some intermediate values in NN  $z^{(1)}, \dots, z^{(m)}$

$$\begin{cases} \mu = \frac{1}{m} \sum z^{(i)} \\ \sigma^2 = \frac{1}{m} \sum (z^{(i)} - \mu)^2 \\ z_{\text{norm}}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}} \\ \hat{z}^{(i)} = \vartheta z_{\text{norm}}^{(i)} + \beta \end{cases}$$

If  $\vartheta = \sqrt{\sigma^2 + \epsilon}$  ←  
 $\beta = \mu$  ←  
 then  $\hat{z}^{(i)} = z^{(i)}$

$x \leftarrow$   
 $\hat{z}^{(i)} \leftarrow$

learnable parameters of model.



Use  $\hat{z}^{(i)}$  instead of  $\frac{u}{\sigma}$ .

我们有一些在神经网络中的隐藏值有  $z^{(1)}, \dots, z^{(m)}$ , 其中  $z^{(1)}, \dots, z^{(m)}$  为  $z^{(l)(i)}$ , 我们有公式

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

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^m (z^{(i)} - u)^2$$

$$z_{\text{norm}}^{(i)} = \frac{z^{(i)} - u}{\sqrt{\sigma^2 + \epsilon}}$$

$$\hat{z}^{(i)} = \vartheta z_{\text{norm}}^{(i)} + \beta$$

其中参数  $\vartheta$  和  $\beta$  为学习模型参数, 他们的作用是可以随意设置  $\hat{z}^{(i)}$  的平均值。

事实上, 若  $\vartheta = \sqrt{\sigma^2 + \epsilon}, \beta = u$ ,

那么

$$\hat{z}^{(i)} = z^{(i)}$$

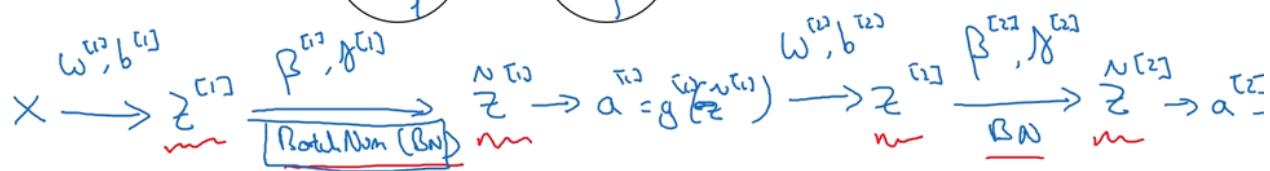
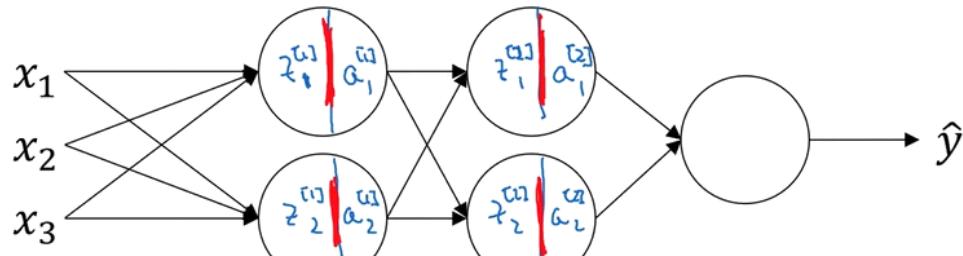
通过赋予  $\vartheta$  和  $\beta$  其它值, 可以使我们构造含其他平台和方差的隐藏单元值, 所以在网络匹配这个单元的方式, 通过计算之后, 通过公式  $\hat{z}^{(l)(i)}$  代替  $z^{(l)(i)}$  方便神经网络的计算。

Batch 归一化的作用是它适用的归一化过程不只是输入层, 同样适用于神经网络中的深度隐藏层

## (2) 将BatchNorm拟合进神经网络

2022年2月22日 15:02

# Adding Batch Norm to a network



Parameters:  $\{w^{(1)}, b^{(1)}, w^{(2)}, b^{(2)}, \dots, w^{(l)}, b^{(l)}, \beta^{(1)}, \gamma^{(1)}, \dots, \beta^{(l)}, \gamma^{(l)}\}$

$$\frac{\partial \beta^{(l)}}{\partial \beta} = \frac{\partial \beta^{(l)}}{\partial \beta^{(l)}} = 1$$

$\text{tf.nn.batch_normalization}$

Andrew Ng

我们有神经网络如上图所示，我们有工作流程如下

$$x \xrightarrow{w^{(1)}, b^{(1)}} z^{(1)} \xrightarrow{\beta^{(1)}, \vartheta^{(1)} (\text{batch norm})} \hat{z}^{(1)} \rightarrow a^{(1)}$$

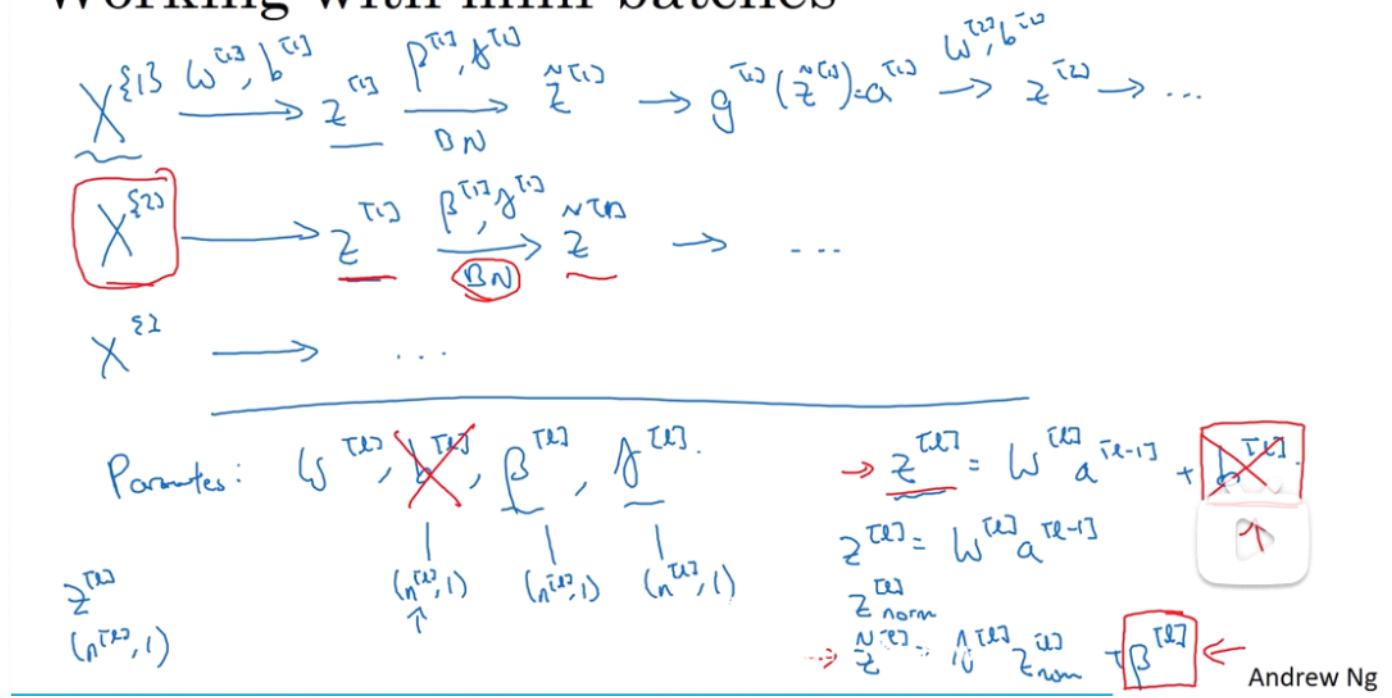
$$= g(\hat{z}^{(1)}) \xrightarrow{w^{(2)}, b^{(2)}} z^{(2)} \xrightarrow{\beta^{(2)}, \vartheta^{(2)} (\text{batch norm})} \hat{z}^{(2)} \rightarrow a^{(1)}$$

所有我们有参数

$$w^{(1)}, b^{(1)}, \dots, w^{(l)}, b^{(l)}$$

$$\beta^{(1)}, \vartheta^{(1)}, \dots, \beta^{(l)}, \vartheta^{(l)}$$

# Working with mini-batches



我们会将 $x^{(1)}$ 使用mini-batch梯度下降得到 $z^{(1)}$ , 然后通过 $z^{(1)}$ 正规化成 $\hat{z}^{(1)}$ , 然后通过激活函数 $g$ 变成 $a^{(1)}$ 。

我们有参数 $w^{(l)}, b^{(l)}, \beta^{(l)}, \vartheta^{(l)}$ , 我们有公式

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

其中参数 $b^{(l)}$ 可以暂定为0。

所以我们有公式

$$z^{(l)} = w^{(l)}a^{(l-1)}$$

Batch归一化做的是, 先将 $z^{(l)}$ 归一化, 结果为均值0和标准方差, 再由 $\vartheta$ 和 $\beta$ 重新缩放, 但这意味着 $b^{(l)}$ 的值是多少, 都是要被减去的, 因为在Batch归一化的过程中, 要计算 $z^{(l)}$ 的均值, 再减去平均值。在此例的mini-batch中增加任何常数, 数值都不会改变, 因为加上的任何常数都将会被均值减法所抵消。所以, 如果在使用Batch归一化, 其实可以消除参数 $b^{(l)}$ 。

$$\hat{z}^{(l)} = \vartheta^{(l)}z_{norm}^{(l)} + \beta^{(l)}$$

# Implementing gradient descent

for  $t = 1 \dots \text{num MiniBatches}$   
Compute forward prop on  $X^{[t]}$ .

In each hidden layer, use BN to map  $\underline{z}^{[t]}$  with  $\underline{\hat{z}}^{[t]}$ .

Use backprop to compute  $\underline{dw}^{[t]}, \underline{db}^{[t]}, \underline{d\beta}^{[t]}, \underline{dy}^{[t]}$

Update parameters  $\left. \begin{array}{l} w^{[t]} := w^{[t]} - \alpha dw^{[t]} \\ \beta^{[t]} := \beta^{[t]} - \alpha d\beta^{[t]} \\ y^{[t]} := \dots \end{array} \right\} \leftarrow$

Works w/ momentum, RMSprop, Adam.



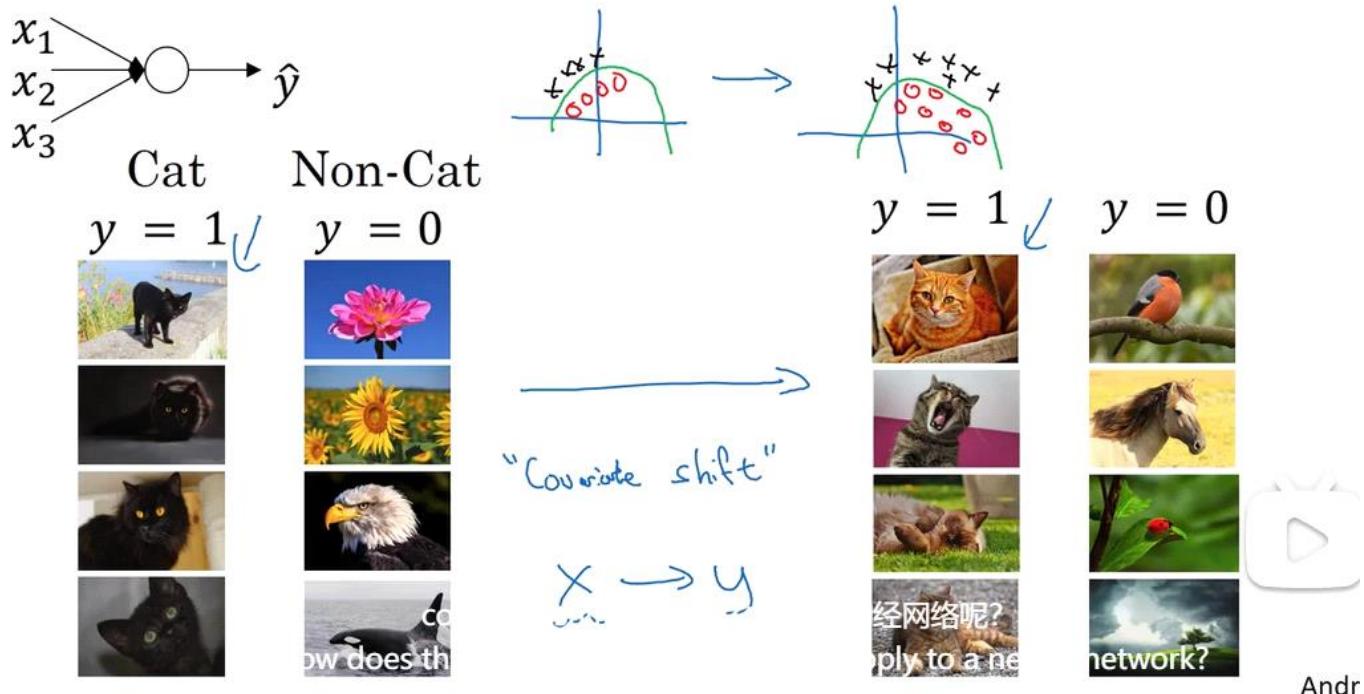
来更新由Batch归一化添加到算法中的 $\beta$ 和 $y$ 参数

### (3) BatchNorm为什么奏效

2022年2月22日 16:26

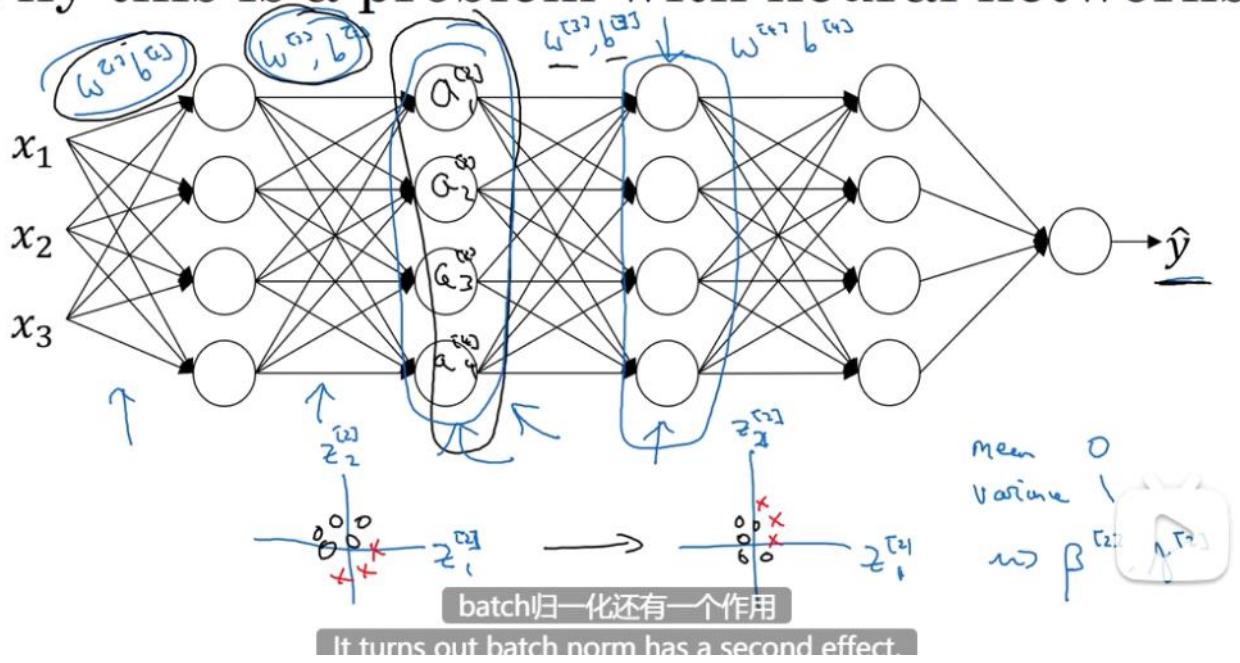
我们已经看到如何归一化输入特征值 $x$ , 使其均值为0, 方差为1。那它又是怎样加速学习的, 与其有一些从0到1, 从1到1000的特征值, 通过归一化所有输入特征值 $x$ , 以获得类似范围的值, 可加速学习BatchNorm起作用的原理与其类似, 但它不仅仅对这里的输入值, 还有隐藏单元的值, 这只是Batch归一化作用的冰山一角。

## Learning on shifting input distribution



我们在左边有黑猫的照片, 我们可以通过神经网络识别出黑猫的照片, 而右边是带颜色的猫, 那么我们如果通过能识别出黑猫的基础上进行一系列的操作使得它能识别出有色彩的猫, 这可能会涉及数据分布问题, 即左边的数据分布范围可能与右边的数据分布范围不一样, 这可能会导致学习算法要重新学习。

# Why this is a problem with neural networks?



我们有对于第二层而言，第一层是输入层，而对于第三层来说，第二层是输入层，我们需要通过把这些数据(输入层，和要应用层的前几个隐藏层)都要进行归一化，使得它们的数据范围一致，这样会使得它们的数据范围变得稳定，至少他们的均值与方差也会是均值0，方差1。或者不一定必须是均值0，方差1，而由 $\beta$ 和 $\theta$ 决定的值。

## Batch Norm as regularization

- Each mini-batch is scaled by the mean/variance computed on just that mini-batch.  $\hat{z}^{[l]}$   $\mu, \sigma^2$
- This adds some noise to the values  $z^{[l]}$  within that minibatch. So similar to dropout, it adds some noise to each hidden layer's activations.  $\mu, \sigma^2$
- This has a slight regularization effect.

$$\text{mini-batch : } 64 \longrightarrow 512$$

我认为正规化几乎是一个意想不到的副作用

BatchNorm有轻微的正则化效果

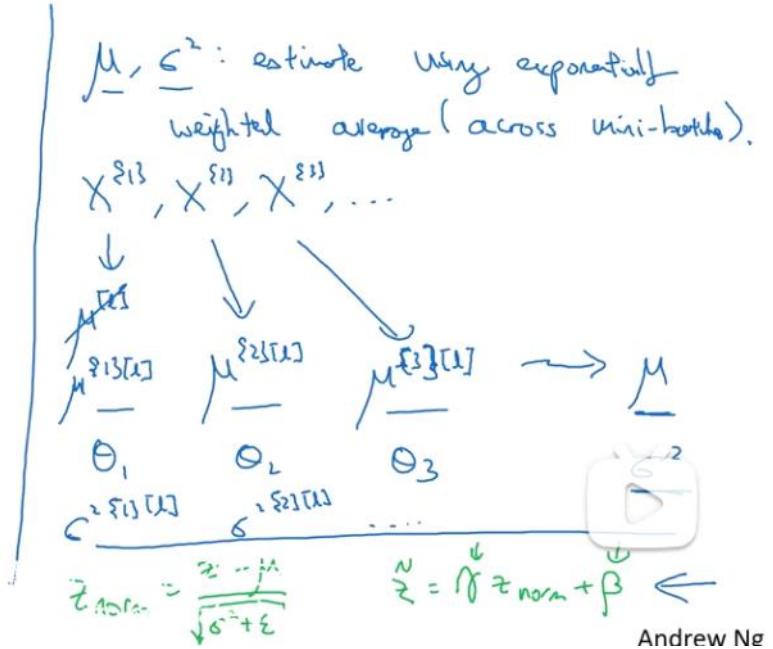
第一、batch归一化中非直观的一件事是每个mini-batch计算的由均值和方差缩放的，因为在mini-batch上计算的均值和方差，而不是在整个数据集上，均值和方差有一些小噪音，因为它只在你的mini-batch上计算，如64或128或256，或者更大的训练集，因为均值和方差有一点小噪音。

## (4) BatchNorm 时的测试

2022年2月22日 17:12

### Batch Norm at test time

$$\begin{aligned}\rightarrow \mu &= \frac{1}{m} \sum_i z^{(i)} \\ \rightarrow \sigma^2 &= \frac{1}{m} \sum_i (z^{(i)} - \mu)^2 \\ \rightarrow z_{\text{norm}}^{(i)} &= \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}} \\ \rightarrow \tilde{z}^{(i)} &= \gamma z_{\text{norm}}^{(i)} + \beta\end{aligned}$$



# (1) Softmax回归

2022年2月22日 20:24

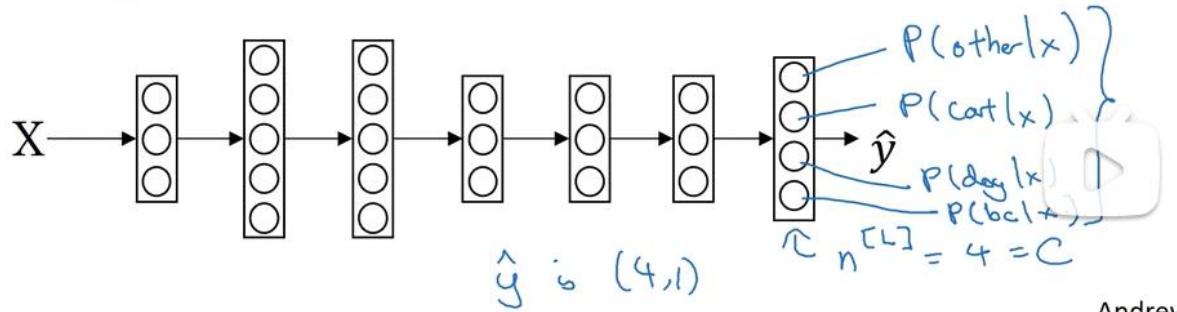
有一种logistic回归的一般形式叫做Softmax回归，能让我们在试图识别某一分类时做出预测，或者说是一种多种分类中的一个，不只是识别两个分类。

Recognizing cats, dogs, and baby chicks, other



3      1      2      0      3      2      0      1

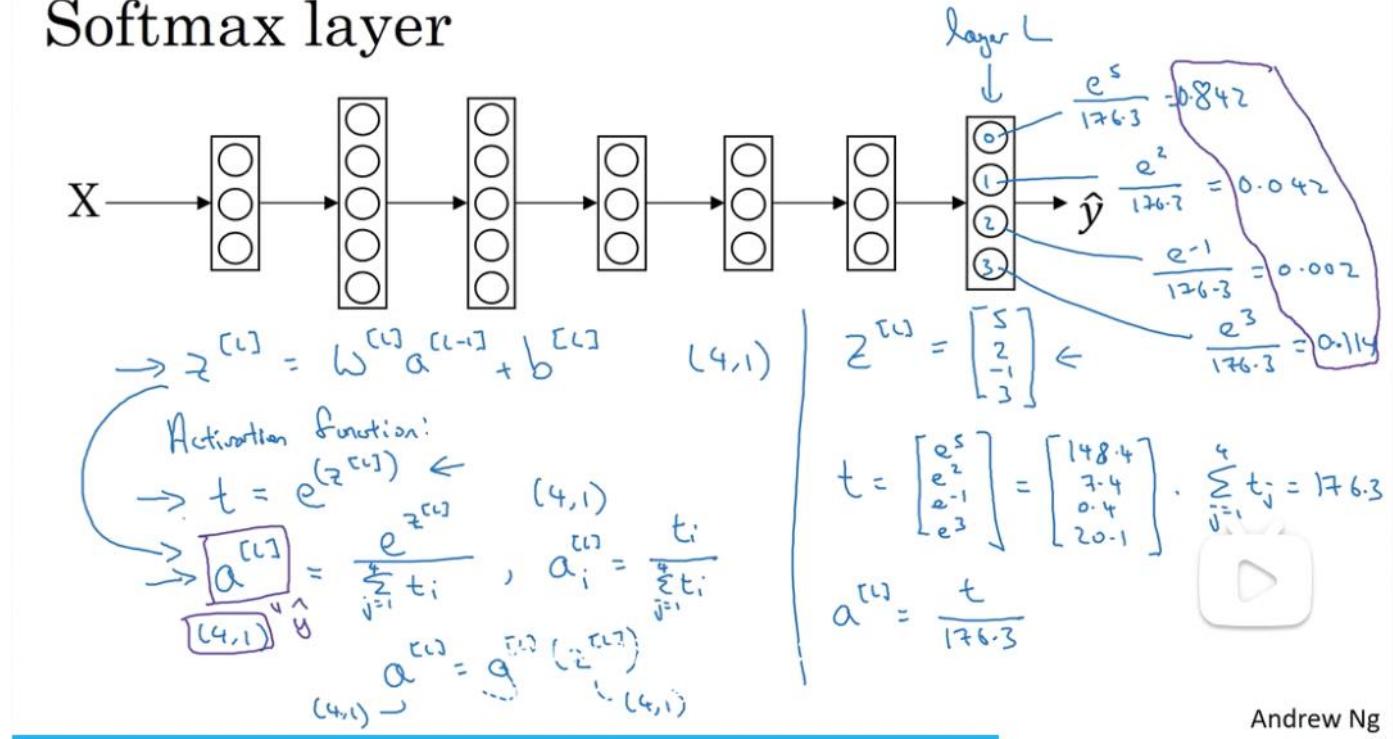
$$C = \# \text{classes} = 4 \quad (0, \dots, 3)$$



Andrew I

我们可以通过图中，可以得到我们需要识别出四种类别，我们需要在最后的输出层，输出各个节点(类型)的概率，这些节点的和加起来为1， $\hat{y}$ 将是一个4\*1维的矩阵。

# Softmax layer



我们需在最后一层实行softmax层，我们有如下公式

$$z^{(l)} = w^{(l)} a^{(l-1)} + b^{(l)} \quad z^{(l)} \in 4 * 1$$

激活函数

$$t = e^{(z^{(l)})}$$

$$a^{(l)} = \frac{e^{(z^{(l)})}}{\sum_{j=1}^4 t_j}$$

$$a_i^{(l)} = \frac{t_i}{\sum_{j=1}^4 t_j}$$

其中  $a^{(l)}$  是一个  $4 * 1$  维的向量

我们有

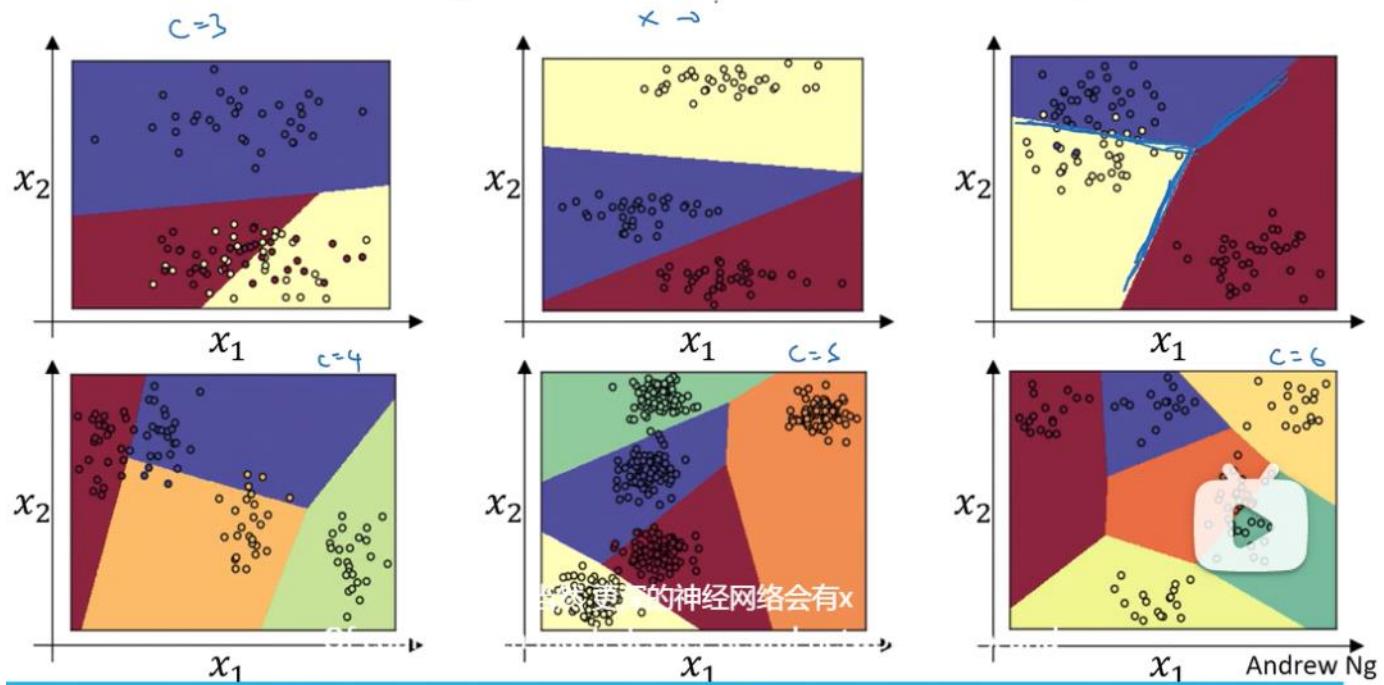
$$a^{(l)} = g^{(l)}(z^{(l)})$$

$g^{(l)}$  为上面函数的包装，这几个节点的概率之和为 1。

# Softmax examples

$$x_1 \quad x_2 \quad \rightarrow \boxed{\hat{y}}$$

$$\begin{aligned} z^{(i)} &= \omega^{(i)} x + b^{(i)} \\ a^{(i)} &= \hat{y} = g(z^{(i)}) \end{aligned}$$



## (2) 训练一个Softmax分类器

2022年2月23日 11:13

# Understanding softmax

$$(4,1)$$

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

"soft max"

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

"hard max"

Softmax regression generalizes logistic regression to  $C$  classes.

If  $C=2$ , softmax reduces to logistic regression.  $\alpha^{[L]} = \begin{bmatrix} 0.842 \\ 0.158 \end{bmatrix}$

## Loss function

$$(4,1)$$

$$y = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \text{- cat } y_2=1$$

$$y_1 = y_3 = y_4 = 0$$

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

small

$$\alpha^{[L]} \approx \hat{y} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix} \leq$$

$$\mathcal{J}(w^{(l)}, b^{(l)}, \dots) = \frac{1}{m} \sum_{i=1}^m \ell(\hat{y}^{(i)}, y^{(i)})$$

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

$$Y = [y^{(1)} \ y^{(2)} \ \dots \ y^{(m)}]$$

$$= \begin{bmatrix} 0 & 0 & 1 & \dots \\ 1 & 0 & 0 & \dots \\ 0 & 1 & 0 & \dots \\ 0 & 0 & 0 & \dots \end{bmatrix}$$

$$(4, m)$$

$$\hat{Y} = [\hat{y}^{(1)} \ \dots \ \hat{y}^{(m)}]$$

$$= \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix} \ \dots$$



Andrew Ng

损失函数

我们假设向量

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

表示一张猫的图片，它表示类1，现在我们假设你的神经网络输出是一个总和为1的概率的向量。

$$a^{(l)} = \hat{y} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix}$$

其中0.2表示一只猫，但却分配到20%是猫的概率，所以在本例中表现不佳，那么什么样的损失函数来训练这个神经网络，

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

通过上述的 $y$ 和 $\hat{y}$ ，我们就有公式

$$\mathcal{L}(\hat{y}, y) = - \sum_{j=1}^4 y_j \log \hat{y}_j = -\log \hat{y}_2$$

这就意味着如果学习算法试图将它变小，因为梯度下降是要用来减少训练集损失的，要使它变小的唯一方式就是要使 $\mathcal{L}(\hat{y}, y)$ 变小，想要做到这一点就是要 $\hat{y}_2$ 尽可能的大，但是因为 $\hat{y}_2$ 是概率所以它不会超过1，这样可以帮助我们找到训练集中的真实类别，然后试图使该类别相应的概率尽可能地高。

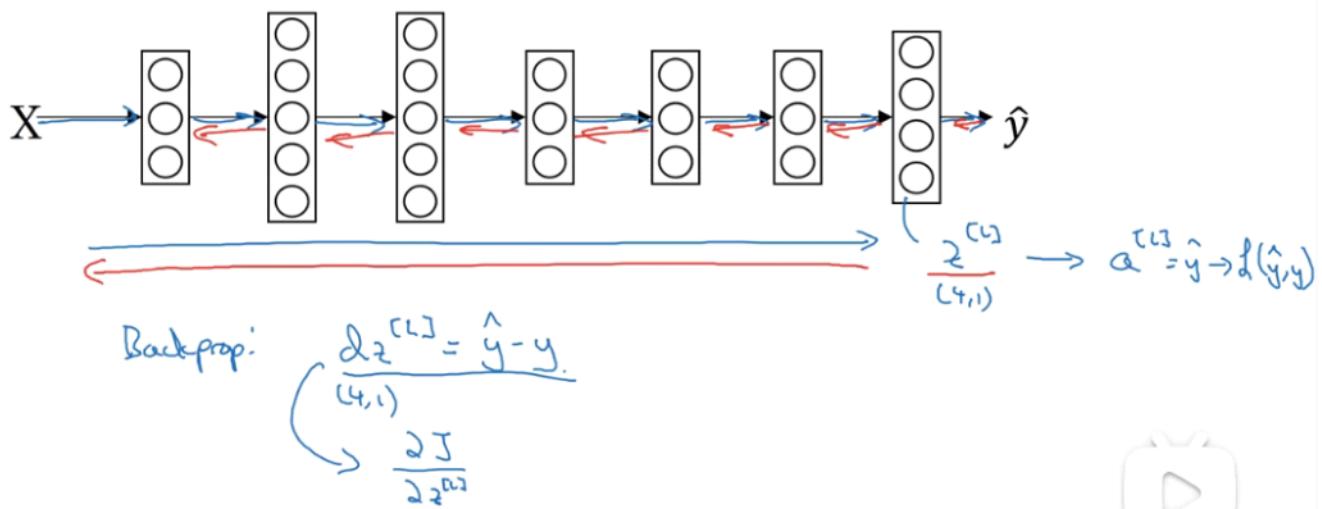
整体整训集的损失有公式

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

我们有真实矩阵，输出矩阵

$$\begin{aligned} Y &= [y^{(1)} \ y^{(2)} \ \dots \ y^{(m)}] \\ &= \begin{bmatrix} 0 & 0 & 1 & \dots \\ 1 & 0 & 0 & \dots \\ 0 & 1 & 0 & \dots \\ 0 & 0 & 0 & \dots \end{bmatrix}_{(4, m)} \end{aligned} \quad \begin{aligned} \hat{Y} &= [\hat{y}^{(1)} \ \dots \ \hat{y}^{(m)}] \\ &= \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix} \dots \end{aligned}$$

# Gradient descent with softmax



但其实在这周的初级练习中你不会用到它

Although you won't actually need this in this week's primary exercise

Andrew Ng



我们有在反向传播的矩阵中有

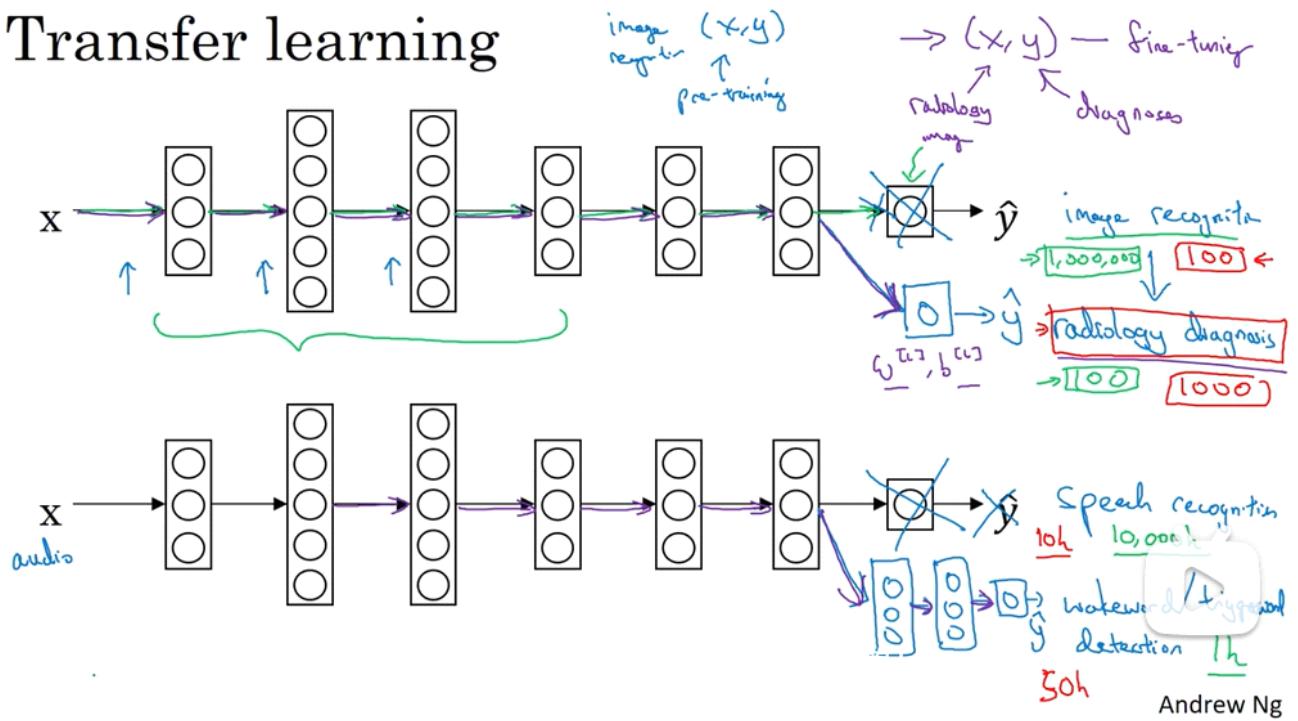
$$\frac{dJ}{dz^{(l)}} = \hat{y} - y$$

# (1) 迁移学习

2022年2月26日 12:47

迁移学习可以将神经网络中一个任务中习得的知识，并将这些知识应用到另一个独立的任务中，例如我们可以将一个已对训练的神经网络能够识别像猫这样的对象，然后使用那些知识或者部分习得的知识去帮助我们更好的去阅读X射线的扫描图，这就是所谓的迁移学习。

## Transfer learning



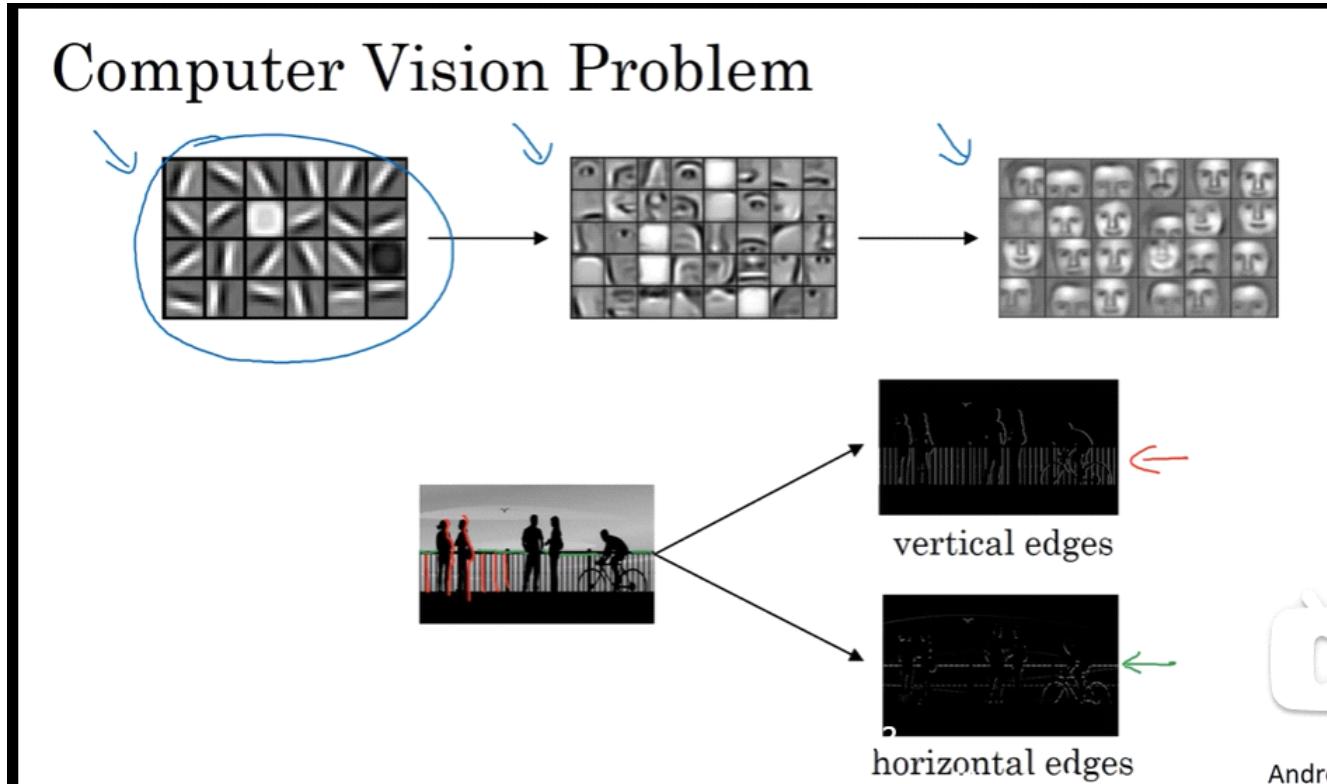
首先我们用一个神经网络，并在  $x, y$  对上训练，其中  $x$  是图像， $y$  是某些对象，如果把这个神经网络用来适应或者迁移，在不同任务中学到的知识，如阅读X射线的扫描图，我们可以做的是，把神经网络最后的输出层拿走，然后为最后一层重新赋予随机权重，然后让它在放射诊断数据上训练，具体来说，在第一阶段训练过程中，当我们在进行图像识别任务训练时，我们可以训练神经网络的所有常用参数，所有权重，所有的层，然后我们就可以得到一个能够做图像识别预测的网络，在训练这个神经网络后，实现迁移学习，我们现在要做的是，把数据集换成新的  $x, y$  对，现在变成放射科图像。, 而  $y$  是我们想要预测的诊断，我们要做的是初始化最后一层权重，我们称之为  $w^{(l)}, b^{(l)}$  随机初始化，现在我们在这个新数据集上重新训练网络，要用放射科数据集从新训练神经网

络有几种做法。若我们的训练集数据很小，我们可能只需要重新训练最后一层的权重，就是 $w^{(l)}, b^{(l)}$ 并保持其它参数不变。若我们有足够的数据，我们就可以重新训练神经网络中剩下的所有层，经验规则是，如果有一个数据集较小的数据集，就只训练输出层前的最后一层，或者也许是最后一两层，但是如果我们有很多数据，那么我们可以重新训练网络中的所有参数，如果我们重新训练神经网络中的所有参数，那么这个图像识别数据的初期训练阶段，有时称为预训练。因为我们再用图像识别数据，去预先初始化，或者预训练神经网络的权重，然后如果以后更新所有权重，然后在放射科数据上训练，有时这个过程叫做微调，在这个例子中，我们把图像中学到的知识应用或迁移到放射诊断上来，

## (1) 边缘检测

2022年2月26日 13:13

卷积运算是卷积神经网络最基本的组成部分。



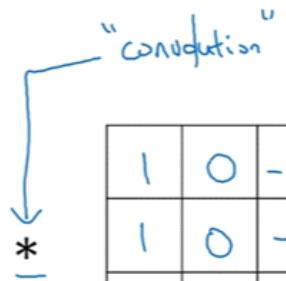
给定这样的一张照片，我们想让电脑搞清楚这张照片究竟有什么物体，我们想做的第一件事可能就是去检测图像中的垂直边缘，比如这张图片里的栏杆，对应垂直线，我们可能也想要去检测水平边缘，比如说一些很明显的水平线，那么如何在图像中检测这些边缘呢？

# Vertical edge detection

$$3 \times 1 + 1 \times 1 + 2 \times 1 + 0 \times 0 + 5 \times 0 + 7 \times 0 + 1 \times -1 + 8 \times -1 + 2 \times -1 = -5$$

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

$6 \times 6$



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

$3 \times 3$   
filter

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

$4 \times 4$



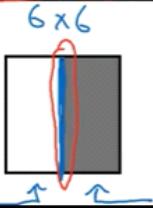
python: conv-forward  
tensorflow: tf.nn.conv2d  
keras: Conv2D

Andrew Ng

如图，我们正在进行垂直边缘检测，把图像的矩阵与过滤器进行卷积操作进行垂直边缘检测，为什么我们可以这样进行垂直边缘检测呢？

# Vertical edge detection

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0



$6 \times 6$

\*

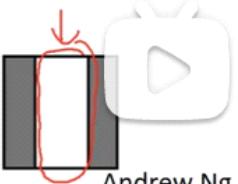
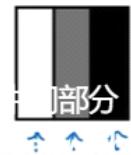
1	0	-1
1	0	-1
1	0	-1

$3 \times 3$

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

$4 \times 4$



Andrew Ng

我们可以看到最左边的矩阵一半表示白色，一半表示黑色，而我们的过滤器也是由1, 0, -1组成，这左边的图像矩阵相对应即左为白，右

为黑，中间不需要考虑，那么最后得到的结果是左右都为0，中间为正数，与之对应的图像是左右都为黑，中间为白，这样我可就把最左边图像的垂直边缘探测出来了。

## (2) 更多边缘检测

2022年2月26日 17:04

在这节中，我们将会学习到如何区分正边和负边，这实际上就是由亮到暗与由暗到亮的区别，也就是边缘的过渡。

### Vertical edge detection examples

$$\begin{array}{|c|c|c|c|c|c|c|} \hline 10 & 10 & 10 & 0 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 & 0 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline \end{array} = \boxed{\begin{array}{|c|c|c|c|c|c|c|} \hline 0 & 30 & 30 & 0 & & & \\ \hline 0 & 30 & 30 & 0 & & & \\ \hline 0 & 30 & 30 & 0 & & & \\ \hline 0 & 30 & 30 & 0 & & & \\ \hline \end{array}}$$
  
$$\begin{array}{|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 10 & 10 & 10 & 10 \\ \hline 0 & 0 & 0 & 10 & 10 & 10 & 10 \\ \hline 0 & 0 & 0 & 10 & 10 & 10 & 10 \\ \hline 0 & 0 & 0 & 10 & 10 & 10 & 10 \\ \hline 0 & 0 & 0 & 10 & 10 & 10 & 10 \\ \hline 0 & 0 & 0 & 10 & 10 & 10 & 10 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline \end{array} = \boxed{\begin{array}{|c|c|c|c|c|c|c|} \hline 0 & -30 & -30 & 0 & & & \\ \hline 0 & -30 & -30 & 0 & & & \\ \hline 0 & -30 & -30 & 0 & & & \\ \hline 0 & -30 & -30 & 0 & & & \\ \hline \end{array}}$$

这两种明暗变化的区别  
versus the dark to light edges

Andrew Ng

第一个卷积运算是从亮到暗的过程，第二个卷积运算是由黑到暗的过程。

# Vertical and Horizontal Edge Detection

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

Vertical

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

Horizontal

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

$6 \times 6$



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

0	0	0	0
30	10	-10	-30
30	10	-10	-30
0	0	0	0



Andrew Ng

垂直与水平边缘检测，我们如上图所示，分别是垂直边缘检测过滤器与水平边缘检测的过滤器。

## Learning to detect edges

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



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

Sobel filter

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

Scharr filter

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

convolution

$w_1$	$w_2$	$w_3$
$w_4$	$w_5$	$w_6$
$w_7$	$w_8$	$w_9$

$3 \times 3$

=

$45^\circ$

$70^\circ$

$73^\circ$

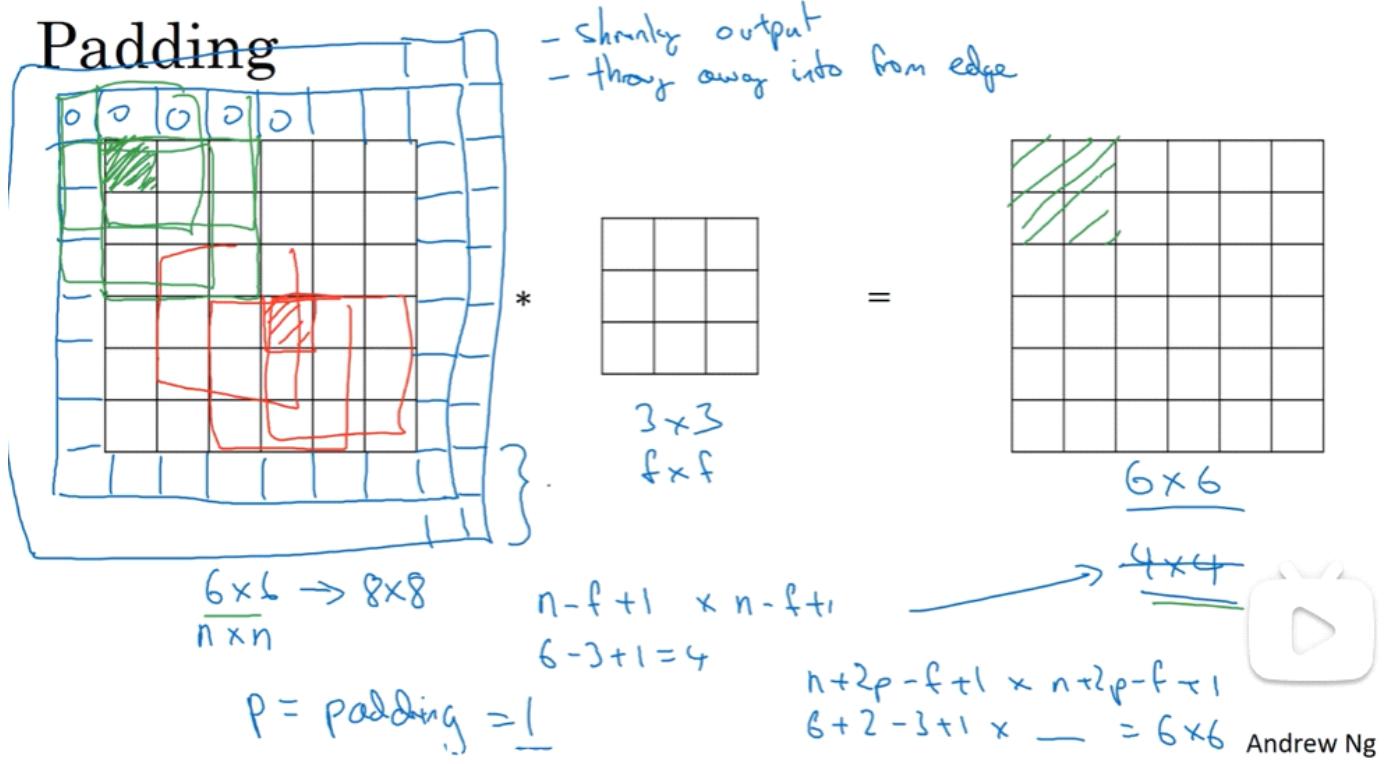



Andrew Ng

我们可以把过滤器的矩阵当成参数进行训练。

### (3) Padding

2022年2月26日 17:55



我们左边有一个 $6 \times 6$ 的矩阵，过滤器是一个 $3 \times 3$ 的矩阵，我们经过映射后得到了一个 $4 \times 4$ 的矩阵。左边矩阵( $n \times n$ )相乘与过滤器( $f \times f$ )相乘会得到一个输出矩阵 $(n-f+1)(n-f+1)$ 。这样的操作有两个缺点，第一个缺点为我们的输出矩阵会缩小，经过几次这样操作后，可能会得到一个 $1 \times 1$ 的矩阵，另一个缺点是它的图像的边缘信息都丢失了，为了解决这些问题在卷积操作之前，填充这幅图像，我们可以在最左的那个矩阵进行一些填充，最后得到的输出矩阵是一个较大的矩阵。

## Valid and Same convolutions

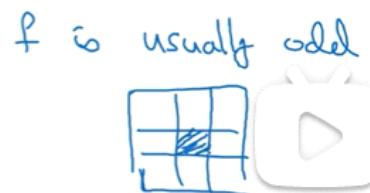
$\nearrow n \text{ padding}$

“Valid”:  $n \times n \quad * \quad f \times f \quad \rightarrow \frac{n-f+1}{f} \times \frac{n-f+1}{f}$

$6 \times 6 \quad * \quad 3 \times 3 \quad \rightarrow \quad 4 \times 4$

“Same”: Pad so that output size is the same as the input size.

$$\begin{aligned} & n + 2p - f + 1 = n \quad \Rightarrow \quad p = \frac{f-1}{2} \\ & 3 \times 3 \quad f = \frac{3-1}{2} = 1 \quad | \quad \begin{matrix} 5 \times 5 \\ f=5 \end{matrix} \quad p=2 \end{aligned}$$



Andrew Ng

## (4) 卷积步长

2022年2月26日 19:06

卷积的步幅是另一个构建卷积网络的基本操作。

### Strided convolution

Diagram illustrating strided convolution. A 7x7 input matrix is multiplied by a 3x3 filter matrix. The input has padding p=1 and stride s=2. The output is a 3x3 matrix.

Input (7x7):

2	3	7	4	6	2	9
6	6	9	8	7	4	3
3	4	8	3	8	9	7
7	8	3	6	6	3	4
4	2	1	8	3 <sup>3</sup>	4 <sup>4</sup>	6 <sup>4</sup>
3	2	4	1	9 <sup>1</sup>	8 <sup>0</sup>	3 <sup>2</sup>
0	1	3	9	2 <sup>-1</sup>	1 <sup>0</sup>	4 <sup>3</sup>

Filter (3x3):

3	4	4
1	0	2
-1	0	3

Stride = 2

Output (3x3):

91	100	83
69	91	127
44	72	74

$$\begin{matrix} n \times n \\ \text{padding } p \\ \text{filter } f \times f \end{matrix} * \begin{matrix} f \times f \\ \text{strides } s \\ s=2 \end{matrix}$$

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$
$$\frac{\frac{7+0-3}{2}+1}{2} = \frac{4}{2} + 1 = 3$$



Andrew Ng

我们可以自己设置步长，我们有左边矩阵  $n \times n$ ，过滤器为  $f \times f$ ，我们有 padding 设置为  $p$ ，然后步长为  $s$ ，则有最后输出矩阵有公式。

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

# Summary of convolutions

$n \times n$  image       $f \times f$  filter

padding  $p$       stride  $s$

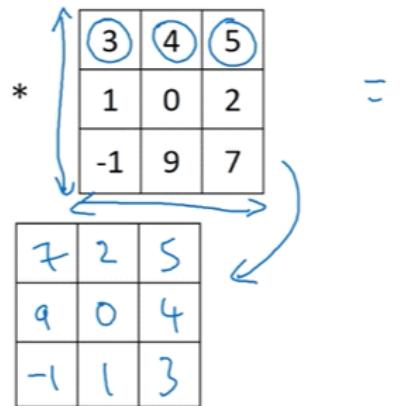
$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \quad \times \quad \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$



Technical note on cross-correlation vs. convolution

Convolution in math textbook:

2	3	2	7	5	4	6	2
6	6	6	9	4	8	7	4
3	4	1	8	3	3	8	9
7	8	3	6	6	6	3	3
4	2	1	8	3	3	4	4
3	2	4	1	9	8		



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

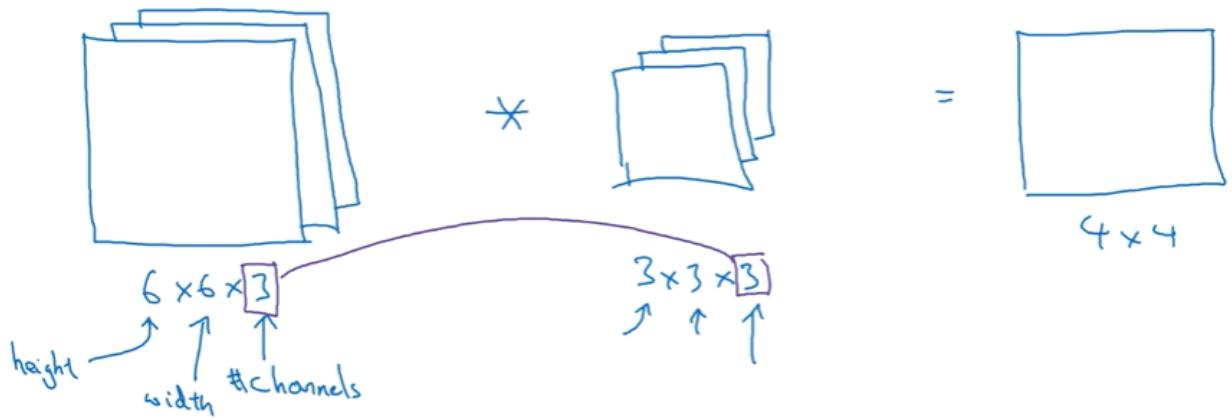
Andrew Ng

## (5) 三维卷积

2022年2月26日 21:01

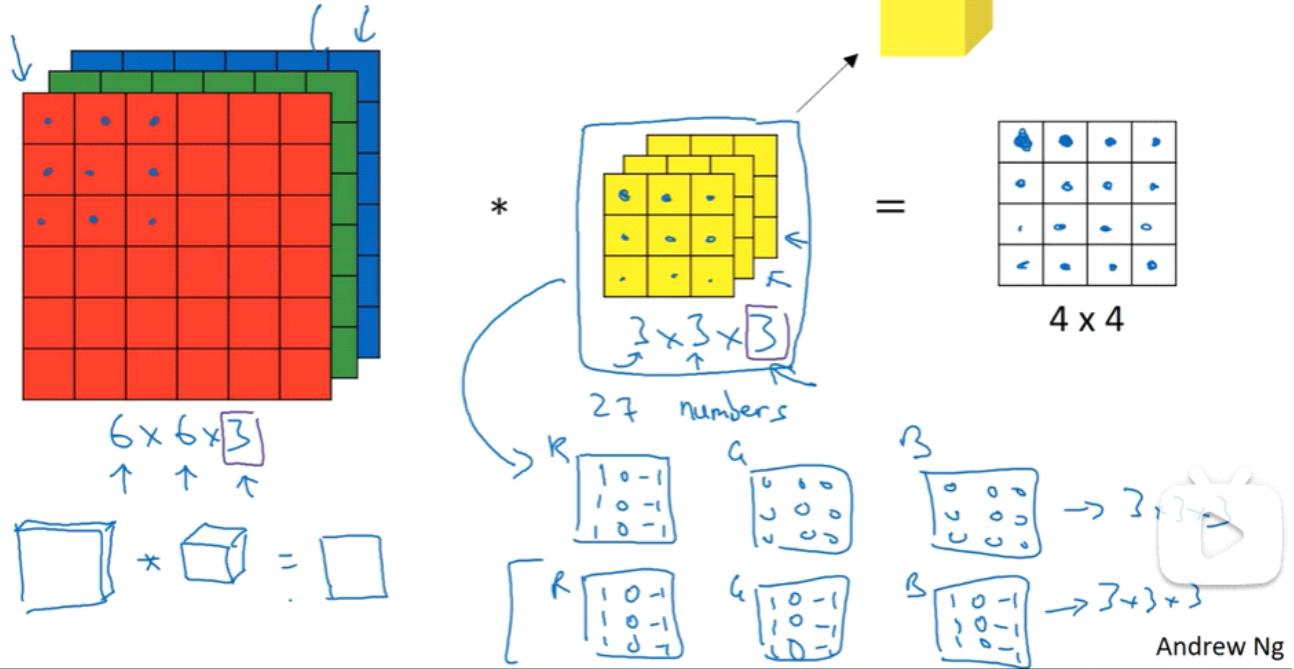
我们已经知道了如何在二维图像上建立卷积网络的，那么在三维立体上如何建立的呢？

### Convolutions on RGB images



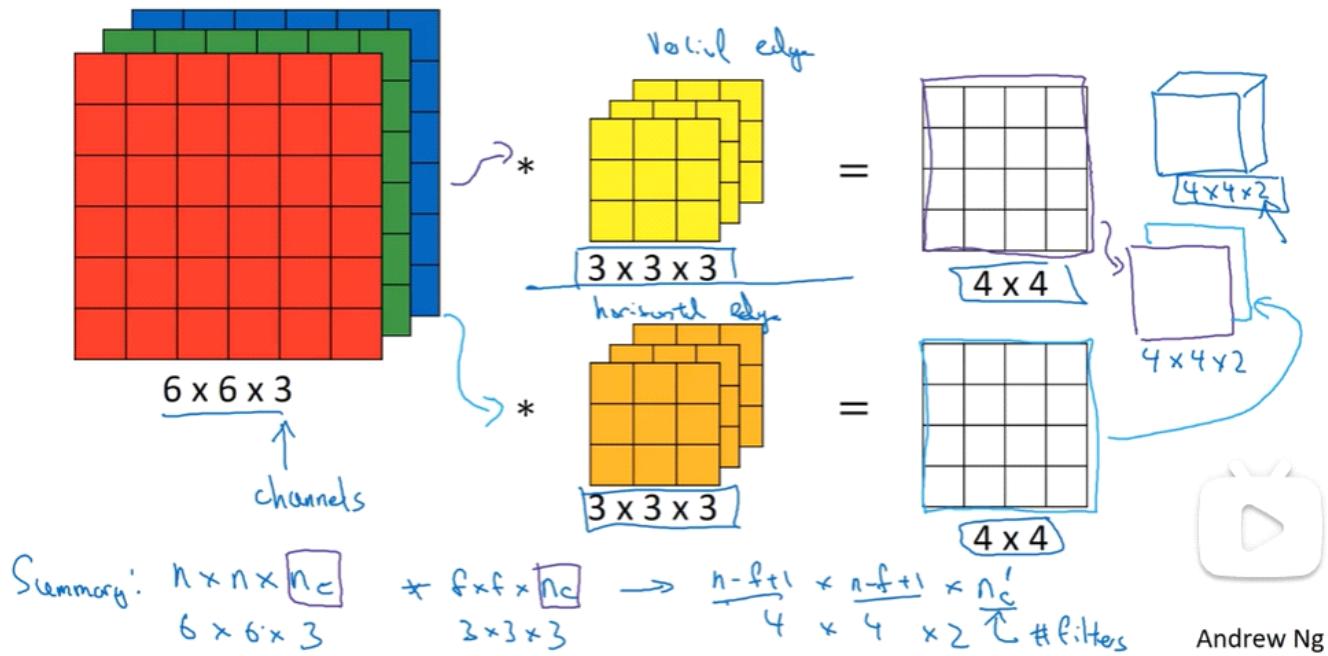
我们在三维立体上由长、宽以及通道组成，过滤器也类似。

# Convolutions on RGB image



我们有左边有 $6 \times 6 \times 3$ 的立体矩阵，过滤器有 $3 \times 3 \times 3$ 的立体矩阵，我们把他们进行一次27个数的卷积，最后就得到一个 $4 \times 4$ 的矩阵。

## Multiple filters



另外还有一个概念，若我们不仅仅想要检测垂直边缘和水平边缘，同时还想要检测45度角的边缘，如图，我们第一个过滤器是检测

垂直边缘，第二个过滤器是检测水平边缘，这两个输出叠加在一起会组成一个 $4*4*2$ 的矩阵，总结一下，我们有公式

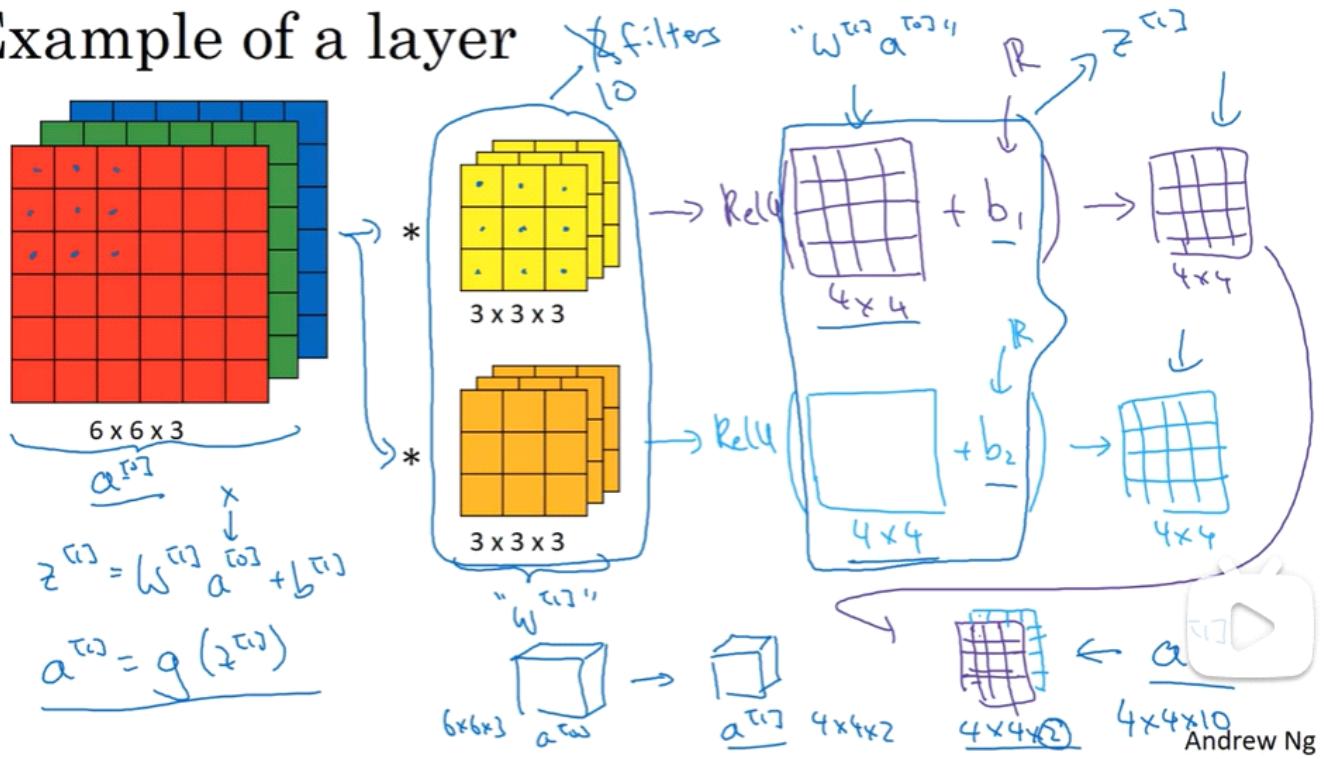
$$n \times n \times n_c * f \times f \times n_c = (n - f + 1) \times (n - f + 1) \times n_c$$

其中 $n$ 为最左边矩阵的维度， $f$ 为过滤器的维度， $n_c$ 为通道数量。

## (6) 单层卷积网络

2022年2月27日 11:35

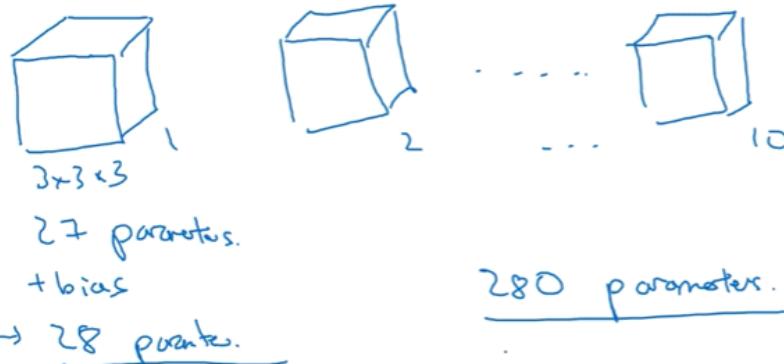
### Example of a layer



上图，为矩阵进行卷积得到的输出矩阵。

### Number of parameters in one layer

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



Andrew Ng

假设我们有10个过滤器，神经网络的一层是 $3*3*3$ ，那么这一层有多

少个参数呢？

因为我们是 $3 \times 3 \times 3$ ，所以我们有27个参数，再加上偏差 $b$ ，所以一共有28个参数，再加上我们有10个过滤器，所以我们一共有280个参数，用这10个过滤器来提取特征，如垂直边缘、水平边缘和其它特征，即使这些图片很大，参数却很少，这就是卷积神经网络中的一个特征，叫做避免过拟合。

## Summary of notation

If layer  $l$  is a convolution layer:

$f^{[l]}$  = filter size

$p^{[l]}$  = padding

$s^{[l]}$  = stride

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

→ Each filter is:  $f^{[l]} \times f^{[l]} \times n_c^{[l]}$

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

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

bias:  $n_c^{[l]} - (1, 1, 1, n_c^{[l]})$  #f: filters in layer l.

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

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

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

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



如果l是卷积层，那么有

$f^{[l]}$  为卷积层

$p^{[l]}$  为 padding

$s^{[l]}$  为步幅

输入为:  $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$

输出为:  $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

$n_c^{(l)}$  为过滤器的数量

任何一个过滤器的数量是:  $f^{(l)} \times f^{(l)} \times n_c^{(l-1)}$

激活函数有:  $a^{(l)} \rightarrow n_H^{(l)} \times n_W^{(l)} \times n_c^{(l)}$

权重: 过滤器的集合 \* 过滤器的数量，故有公式

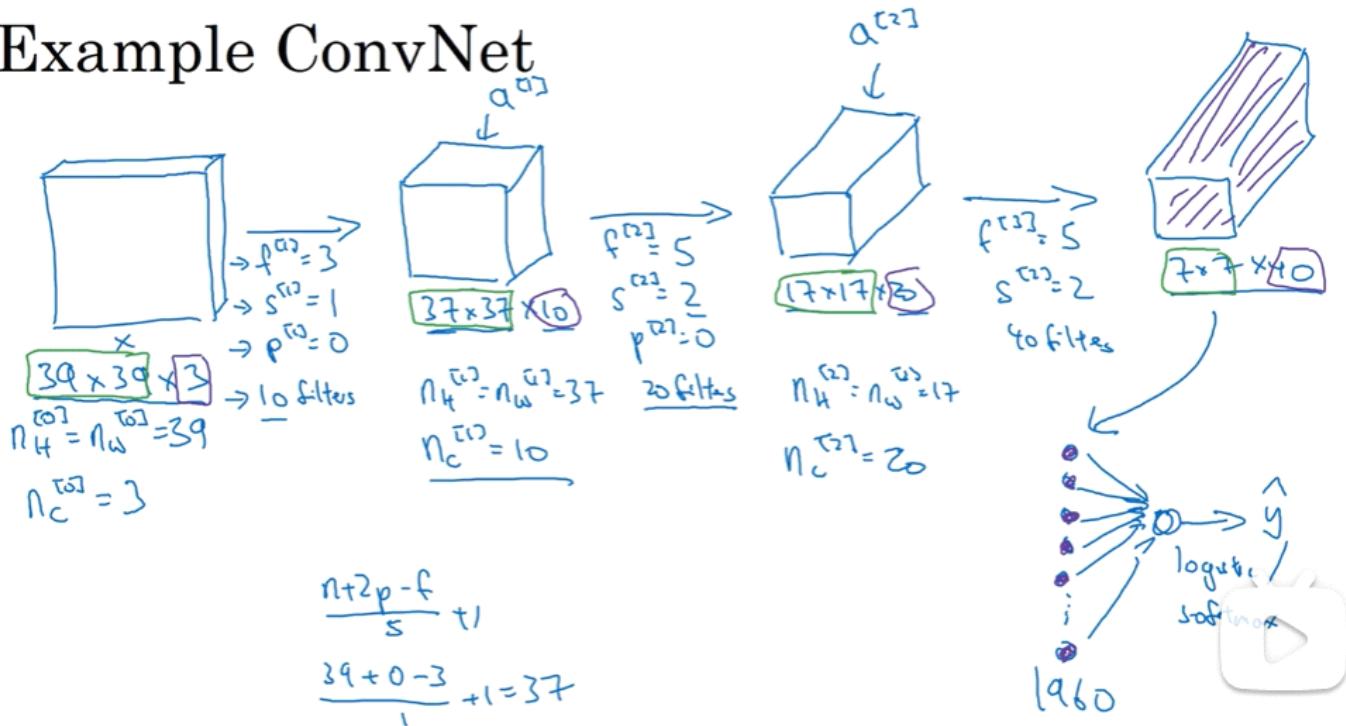
$f^{(l)} \times f^{(l)} \times n_c^{(l-1)} \times n_c^{(l)}$

偏差: $n_C^{(l)} \rightarrow (1,1,1, n_C^{(l)})$

## (7) 简单卷积网络示例

2022年2月27日 20:21

### Example ConvNet



Andrew Ng

假设我们有一张图片，我们想要做图片分类或者图片识别，把这张图片输入定义为 $x$ ，然后辨别图片中有没有猫，用0或1表示，这是一个分类问题，我们来构建适用于这项任务的卷积网络，我们从这张图片中提取一些信息，我们使用卷积网络进行提取，我们设 $f^{(1)} = 3, s^{(1)} = 1, p^{(1)} = 0, filters = 10$ ，即长宽为3，步幅为1，过滤器为10，又我们有公式

$$\frac{n + 2p + f}{s} + 1 = \frac{39 + 0 - 3}{1} + 1 = 37$$

所以这个就是卷积的结果，之后继续卷积，然后得到一个 $7 \times 7 \times 40$ 的向量，为了预测结果，我们把这个长向量填充到softMax回归函数中。

# Types of layer in a convolutional network:

- Convolution (CONV) ← }
- Pooling (POOL) ← }
- Fully connected (FC) ← }



你就可以利用它们构建更强大的网络了

在典型层中的卷积神经网络有

卷积层(CONN)

池化层(POOL)

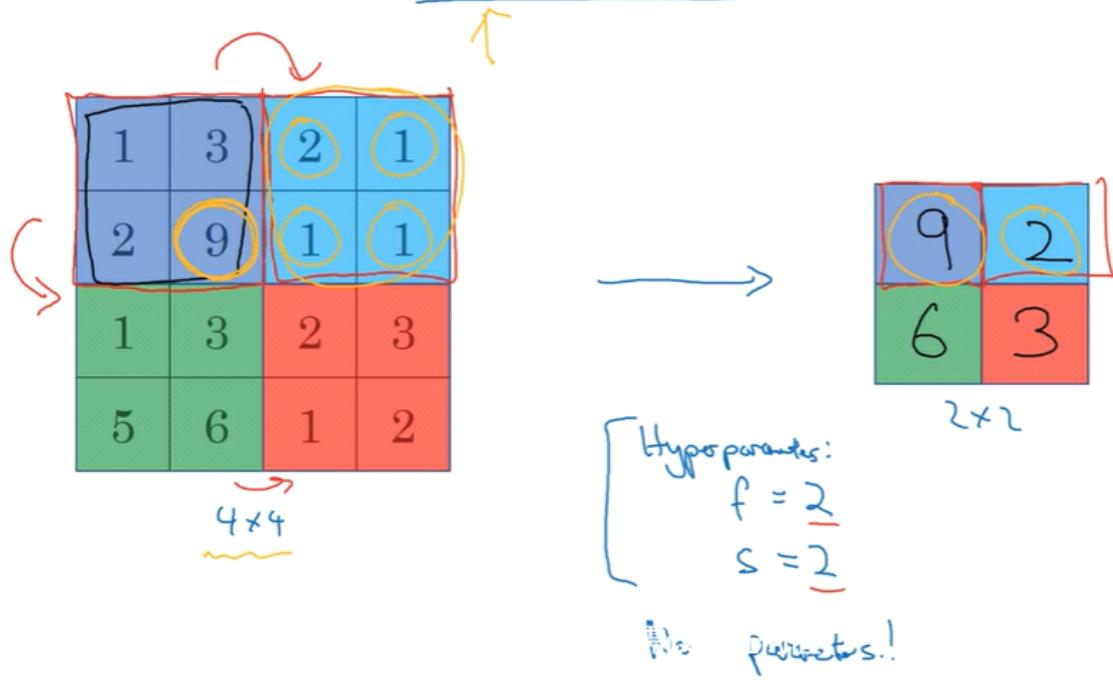
全连接(FC)

## (8) 池化层

2022年2月27日 20:37

除了卷积层，在卷积网络中也经常使用池化层，来缩减模型大小，加快速度，同时提高所提取的鲁棒性。

### Pooling layer: Max pooling

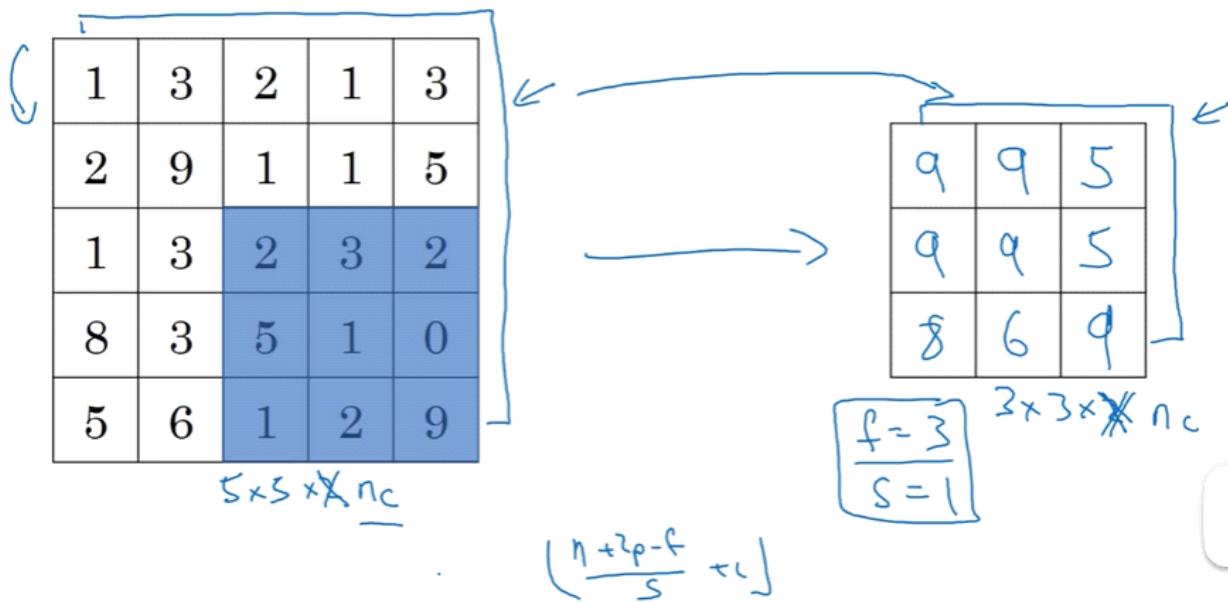


池化层中的最大池化，我们有一个为 $4 \times 4$ 的矩阵，我们令参数 $f=2, s=2$ 进行最大池化，即取 $4 \times 4$ 的矩阵中的最大数，并且一次偏移两块，最后得到一个 $2 \times 2$ 的矩阵。



Andrew Ng

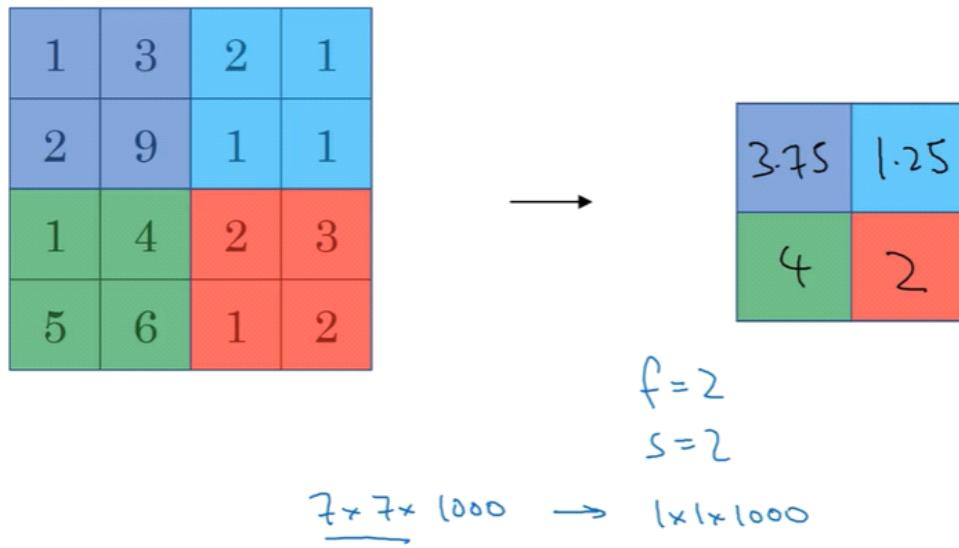
# Pooling layer: Max pooling



Andrew Ng



# Pooling layer: Average pooling



Andrew Ng

平均池化，求一个 $2 \times 2$ 的矩阵进行相加，求平均值，最后得到一个 $2 \times 2$ 的平均池化的矩阵。

# Summary of pooling

Hyperparameters:

- f : filter size       $f=2, s=2$   
 $f=3, s=2$
- s : stride
- Max or average pooling
- $\Rightarrow p: \cancel{\text{padding}}$

No parameters to learn!

$$n_H \times n_w \times n_c$$

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

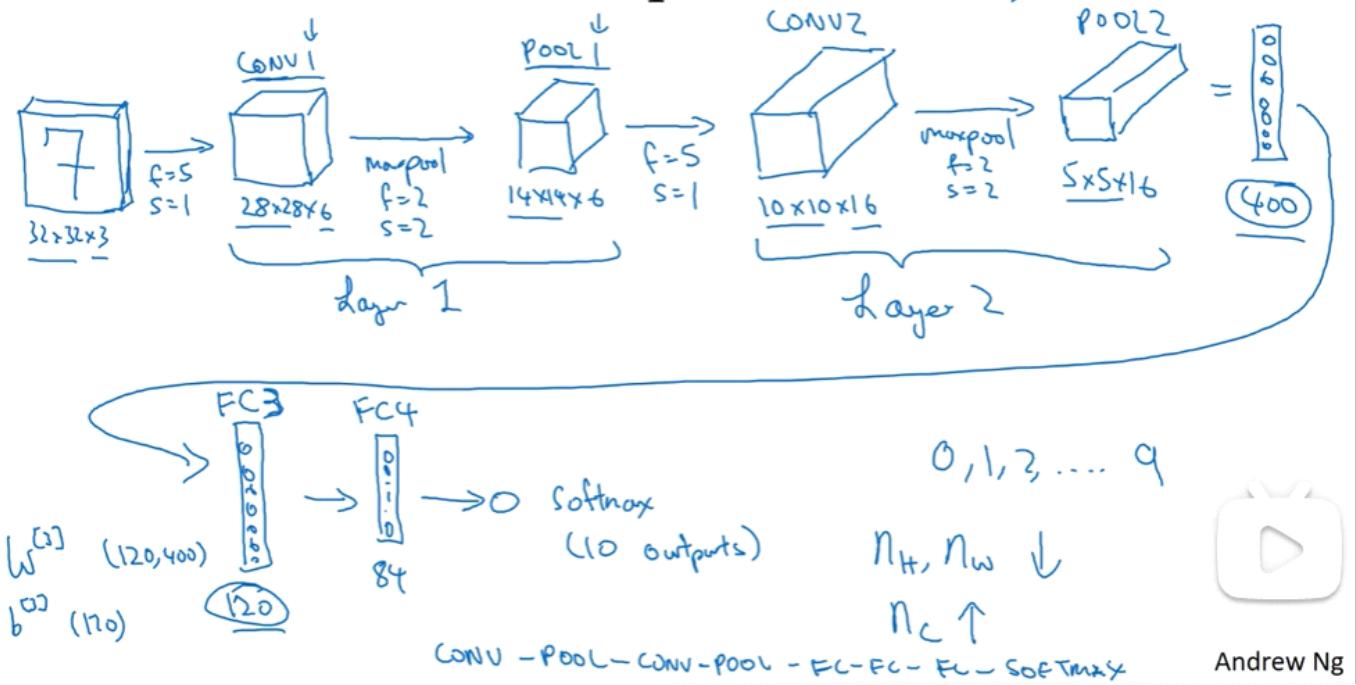


最大池化只是计算神经网络某一层的静态属性

## (9) 卷积神经网络示例

2022年2月27日 20:53

### Neural network example (LeNet-5)



神经网络例子，这个例子为LeNet-5模型，我们把第一个卷积层和第一个池化层看作第一层，把第二个卷积层和第二个池化层看作第二层，然后输出一个具有400个参数的向量，然后连接一个全连接FC3，这个全连接具有 $120 \times 400$ 的参数，然后有120个偏差，再连接一个全连接层FC4，FC4具有一个84个参数的向量，最后输出一个SoftMax。

# Neural network example

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	3,072 $a^{(0)}$	0
CONV1 (f=5, s=1)	(28,28,8)	6,272	208 ←
POOL1	(14,14,8)	1,568	0 ←
CONV2 (f=5, s=1)	(10,10,16)	1,600	416 ←
POOL2	(5,5,16)	400	0 ←
FC3	(120,1)	120	48,001 ↗
FC4	(84,1)	84	10,081 ↗
Softmax	(10,1)	10	841 ↗

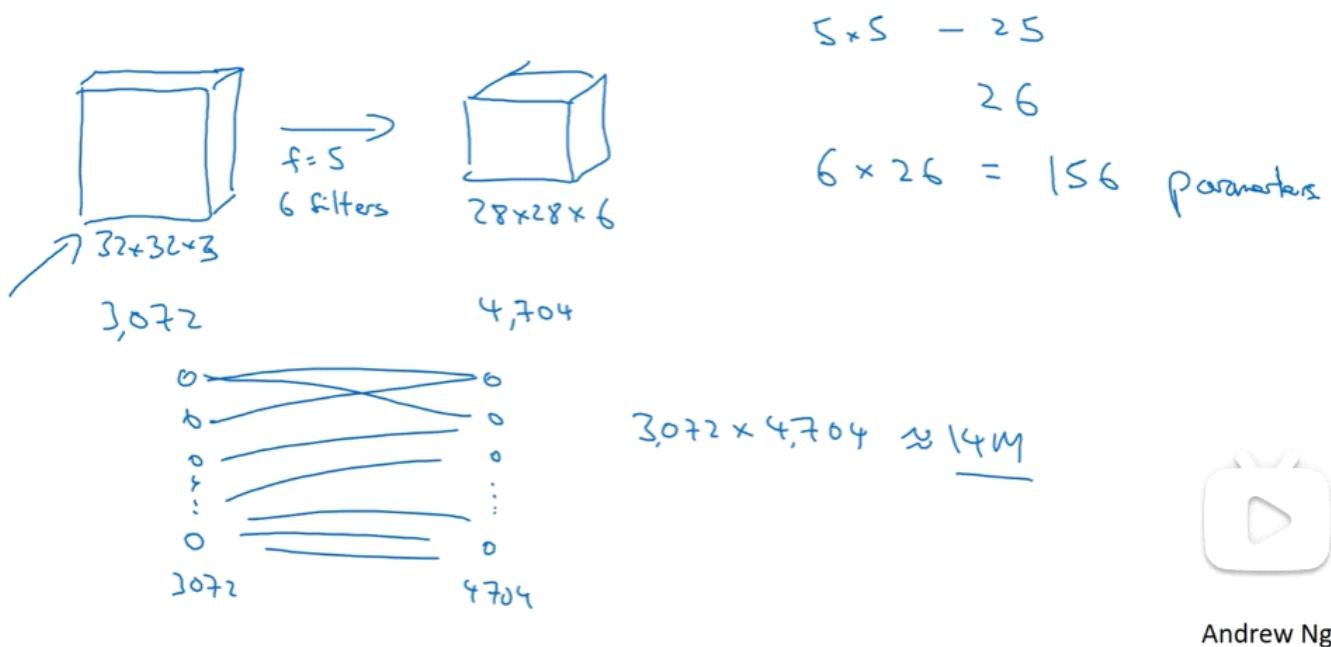
Andrew Ng

## (10) 为什么使用卷积

2022年2月27日 21:38

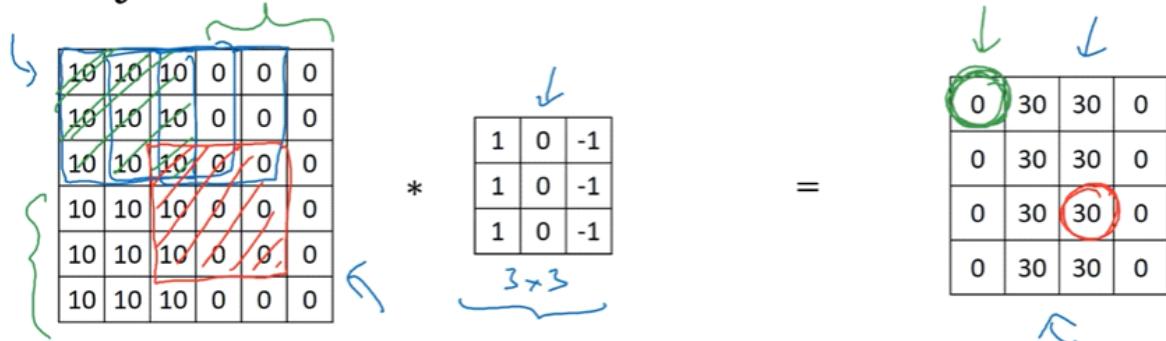
我们来分析一下卷积在神经网络中如此受用的原因，然后对如何整合这些卷积，如何通过一个标注过的训练集训练卷积神经网络做个简单概括。

### Why convolutions



和只用全连层相比，卷积层的两个特征的主要优势为参数共享和稀释连接，若我们所左边的立方体与右边的立方体相连接，则有 $3072 \times 4704 = 14M$ ，我们大概需要有14M的参数，而若我们运用 $5 \times 5$ 的卷积层，则我们只需要 $25+1$ 个偏差层，而有6个过滤层，则有156个参数。

# Why convolutions



**Parameter sharing:** A feature detector (such as a vertical edge detector) that's useful in one part of the image is probably useful in another part of the image.

→ **Sparsity of connections:** In each layer, each output value depends only on a small number of inputs.

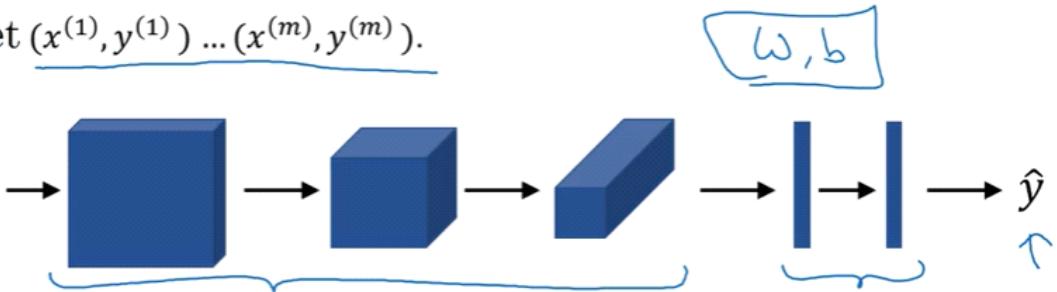


Andrew Ng



## Putting it together

Training set  $(x^{(1)}, y^{(1)}) \dots (x^{(m)}, y^{(m)})$ .



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

Use gradient descent to optimize parameters to reduce  $J$



Andrew Ng

在卷积层和池化层都有自己的参数W和B，有代价函数Cost  $J = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$ ，并且使梯度下降去最小化函数J。

## (11) 经典模型

2022年2月27日 22:01

# Outline

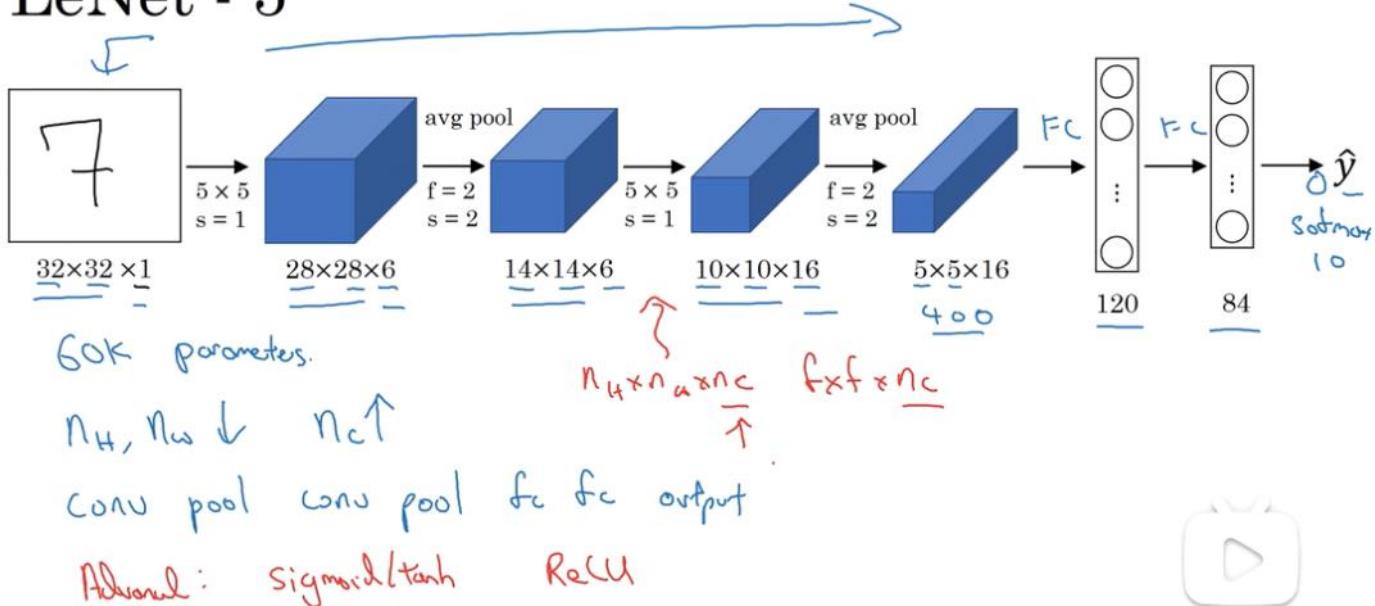
Classic networks:

- LeNet-5 ←
- AlexNet ←
- VGG ←

ResNet (152)



## LeNet - 5

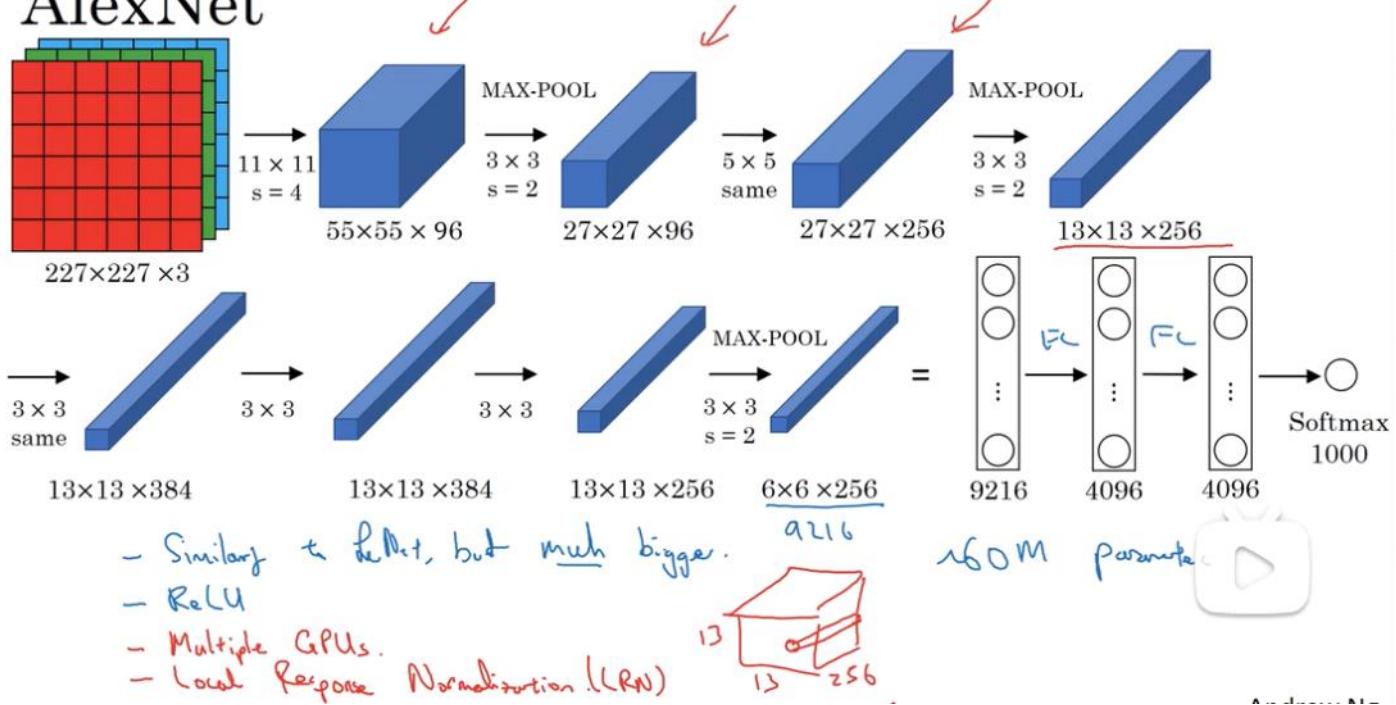


论文中提到的这些复杂细节

And so, the paper talks about those details,

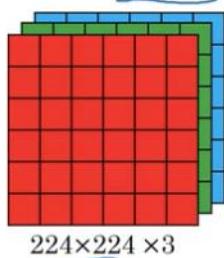


# AlexNet



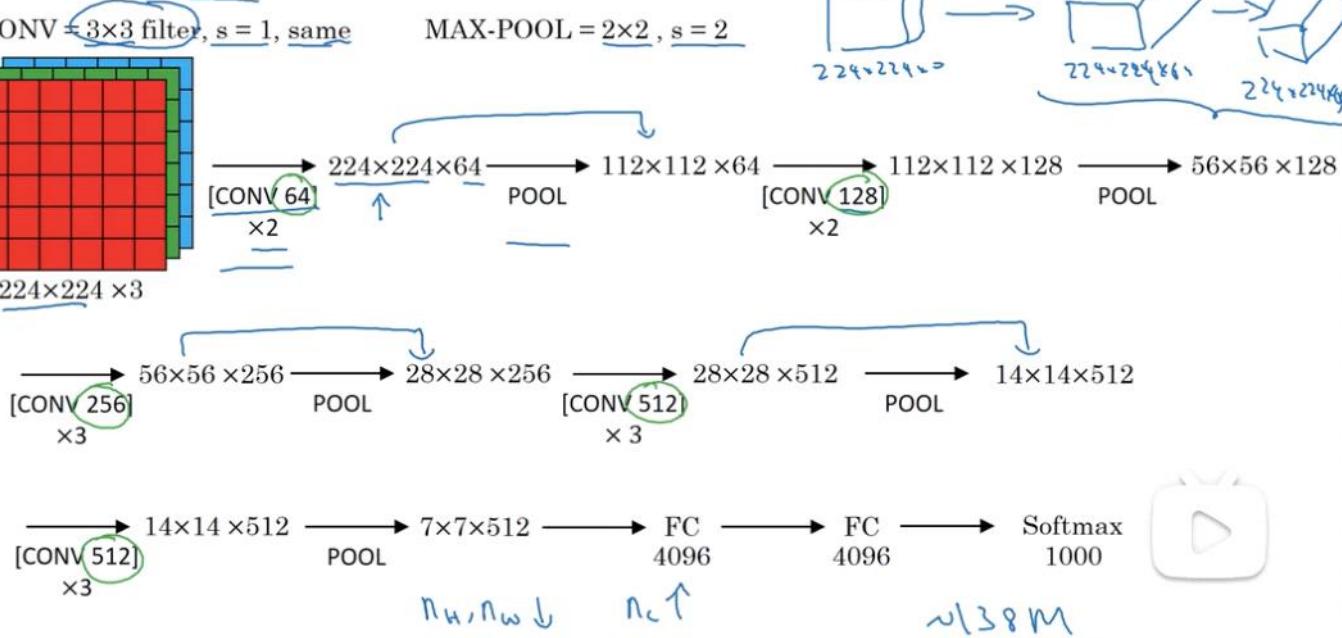
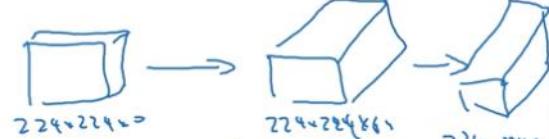
# VGG - 16

CONV =  $3 \times 3$  filter,  $s=1$ , same



# VGG-19

MAX-POOL =  $2 \times 2$ ,  $s=2$



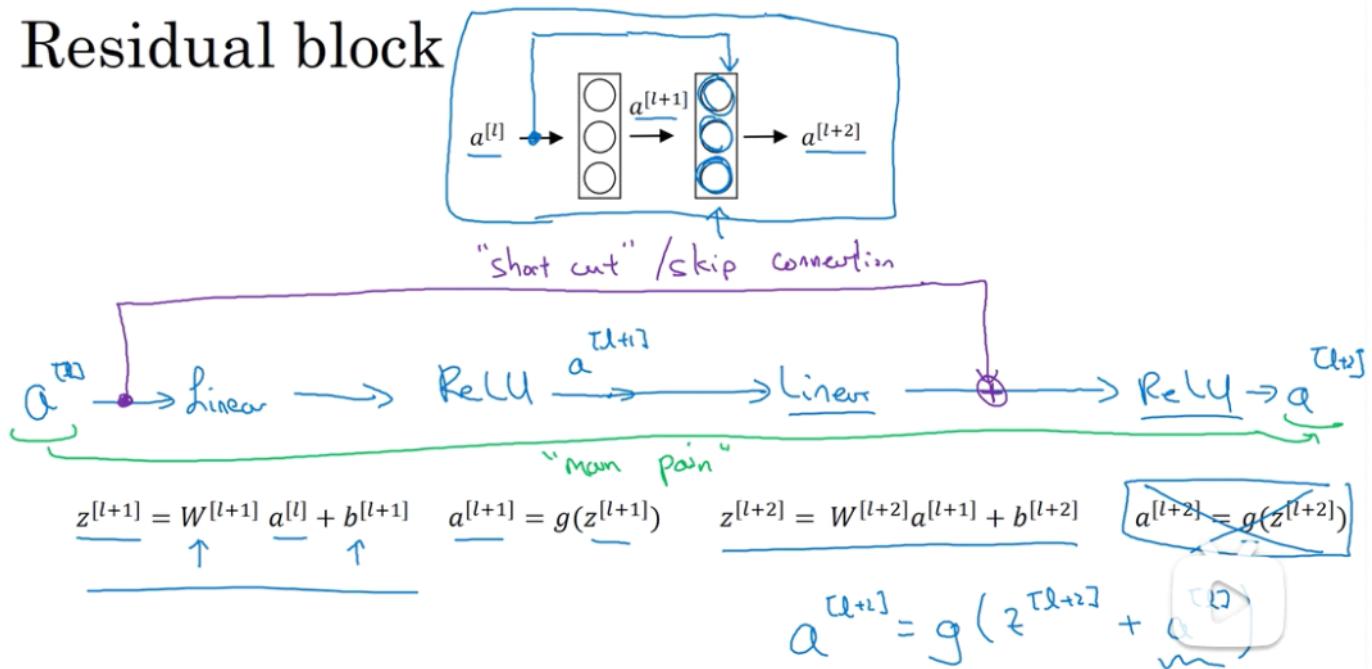
Andrew Ng

## (12) 残差网络

2022年2月27日 22:04

非常非常深的网络是很难训练的，因为存在梯度消失和梯度爆炸的问题，这节我们选择远跳连接，它可从某一网络层获取激活，然后迅速反馈给另一层，甚至是神经网络的更深层，我们可以利用跳远连接构建能够训练深度网络的ResNets，有时深度能到超过100层。

### Residual block



Andrew Ng

我们有从  $a^{(l)}$  到  $a^{(l+2)}$  进行映射，我们有过程

$$a^{(l)} \rightarrow \text{linear} \rightarrow \text{Relu} \rightarrow a^{(l+1)} \rightarrow \text{linear} \rightarrow \text{Relu} \rightarrow a^{(l+2)}$$

我们有

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

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

$$z^{(l+2)} = W^{(l+2)} a^{(l+1)} + b^{(l+2)}$$

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

当我们用  $a^{(l)}$  到  $a^{(l+2)}$  的映射，我们有公式

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

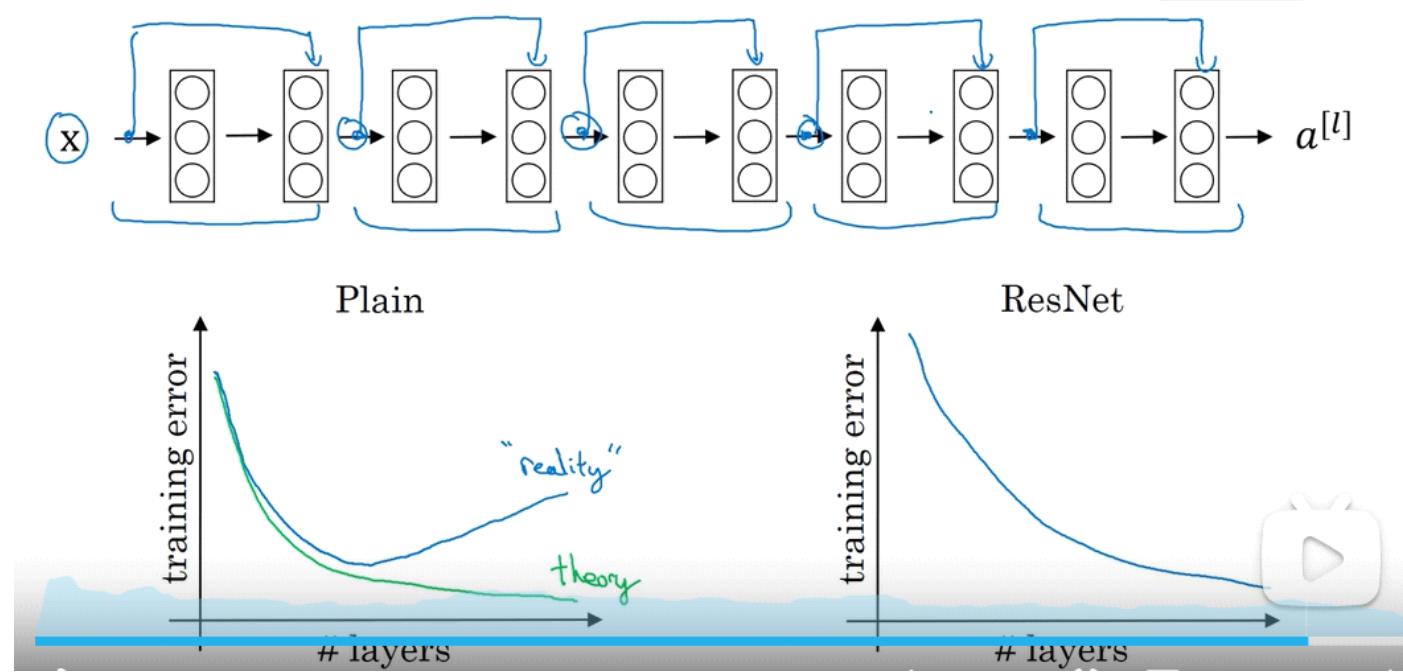
我们这次需要加上  $a^{(l)}$ ，即我们需要加上这个  $a^{(l)}$  产生的一个残差块。

# Residual Network

"Plain network"



+ 关注

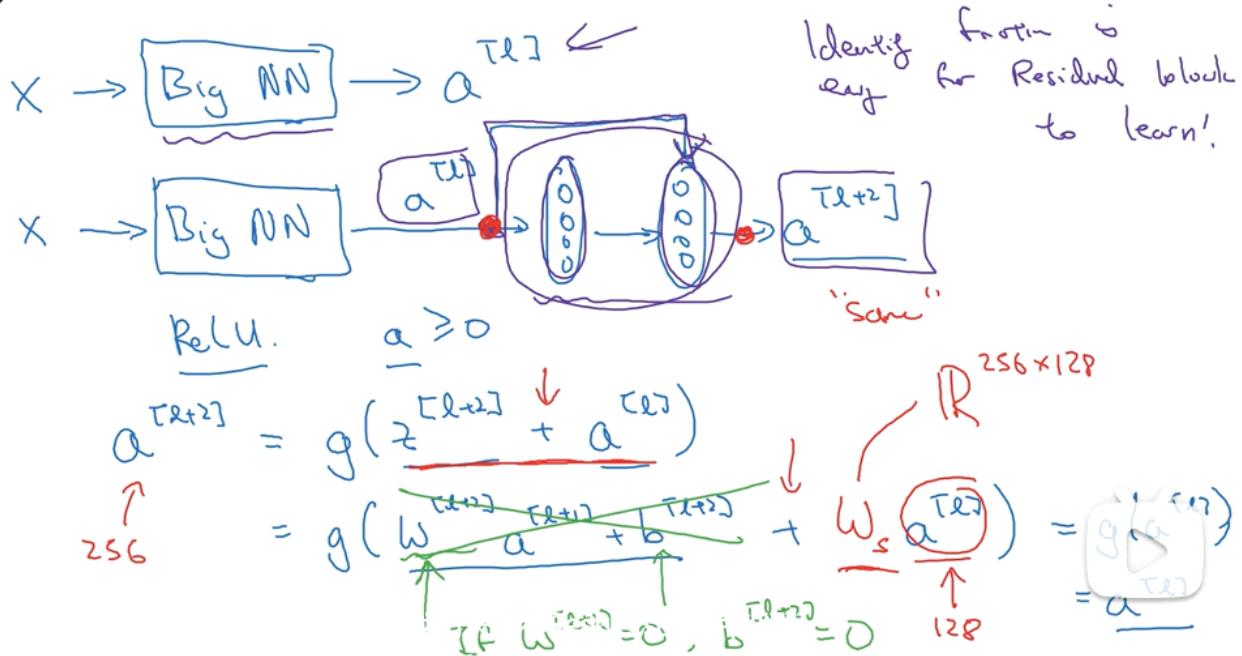


如图这是一个普通网络，我们通过跳连接构成一个残差网络，如果我们使用标准优化算法训练一个普通网络，比如说是梯度下降或者其它热门的其它算法，如果没有多余的残差，没有这些捷径，或者远跳连接，凭经验，我们会发现随着网络深度的加深，训练错误会减少，然后增多，而理论上，随着网络深度的加深，应该训练得越来越好才对，也就是说，理论上网络深度越深越好，但实际上，如果没有残差网络，对于一个普通网络来说，深度越深意味着用优化算法越难训练，实际上，随着网络深度的加深，训练错误会越来越多，但是有了ResNet就不一样了，即使网络加深，训练的表现却不错，比如说出错误会减少，就算是训练深达100层的网络也不例外。

(13) 残差网络有什么用

2022年2月28日 12:30

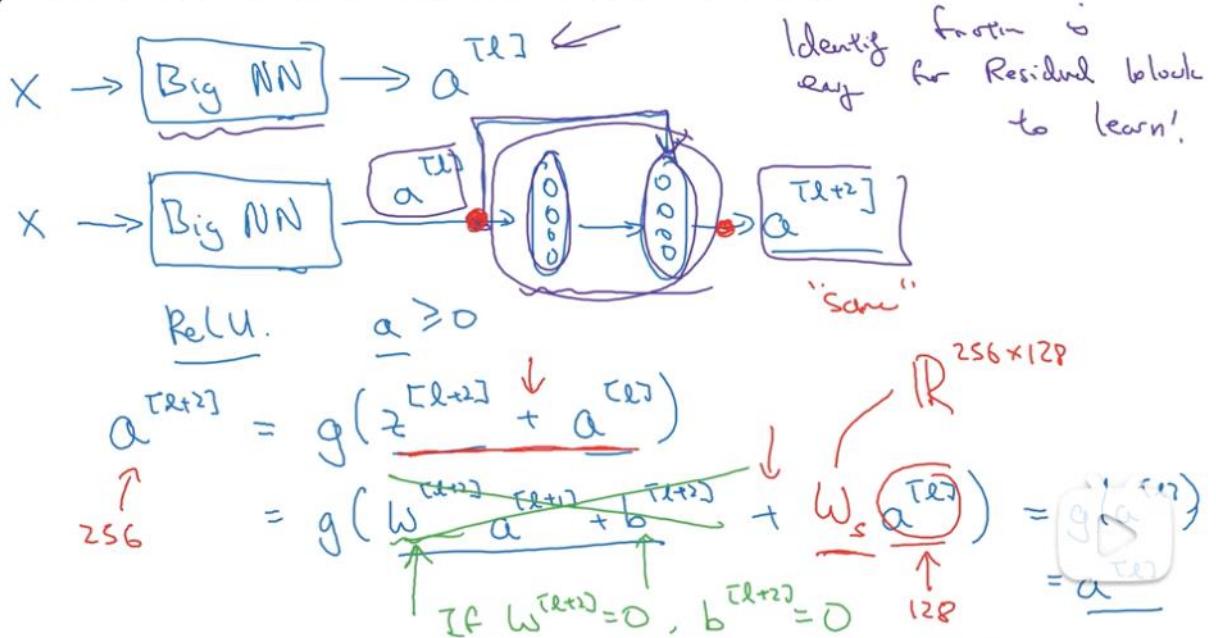
# Why do residual networks work?



这个例子解释了如何在构建更深层次的ResNet网络的同时，还不降低它们在训练集的效率，通常来讲，网络在训练集上表现好，才能在Hold-Out交叉验证集或Dev测试集上有好的表现，至少在训练好ResNet是第一步。

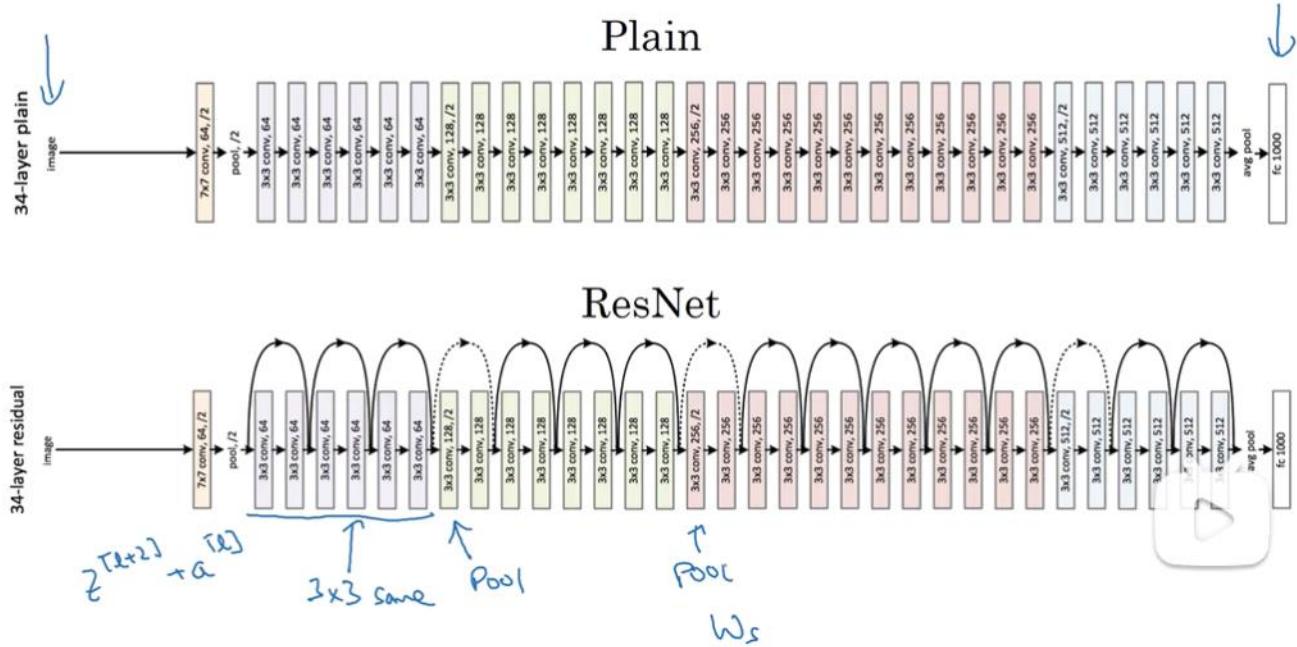
上节我们了解到一个网络深度越深，它在训练集上训练网络的效率有所减弱，这也是有时候我们不希望加深网络的原因，而事实并非如此，至少在训练ResNet网络时，并不完全如此。

# Why do residual networks work?



假设有一个大型神经网络，其输入X，输出激活值 $a^{(l)}$ ，若我们想要增加这个神经网络的深度，那么用BigNN表示，输出为 $a^{(l)}$ ，再给这个网络添加两层，最后输出为 $a^{(l+2)}$ ，即具有近路连接的残差块。

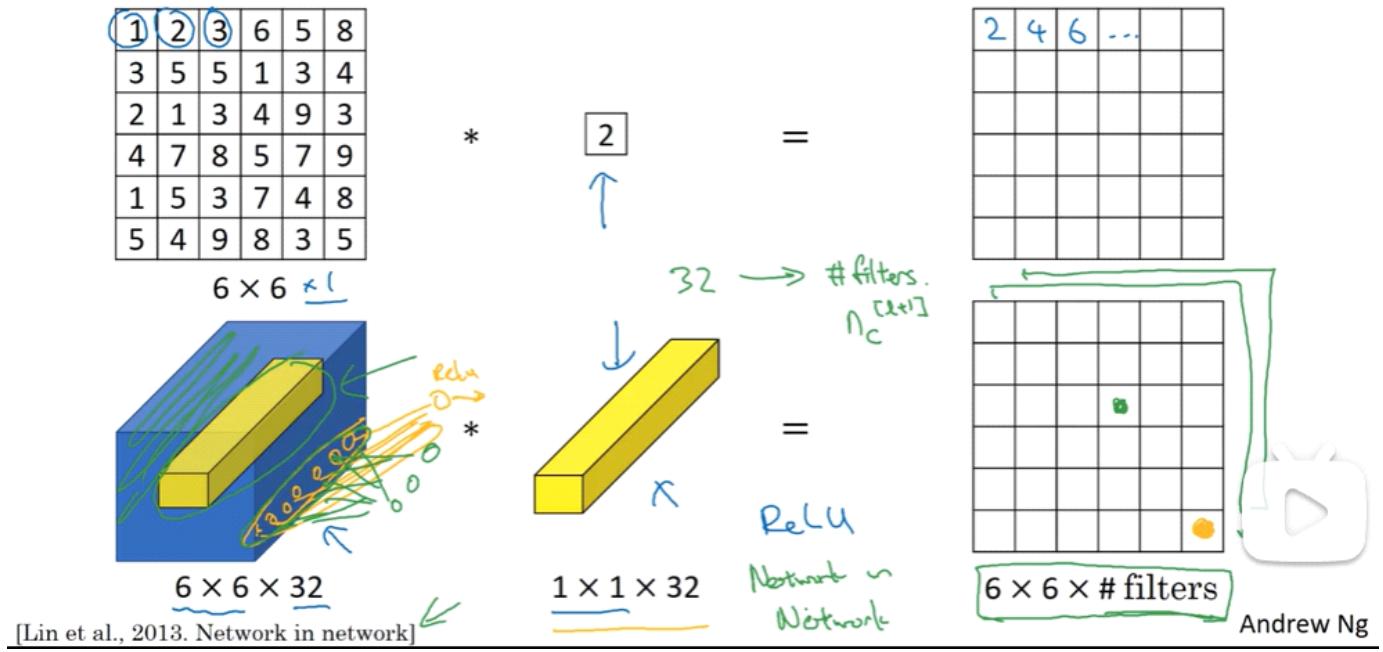
## ResNet



## (14) 网络中的网络以及 $1 \times 1$ 卷积

2022年2月28日 15:30

### Why does a $1 \times 1$ convolution do?



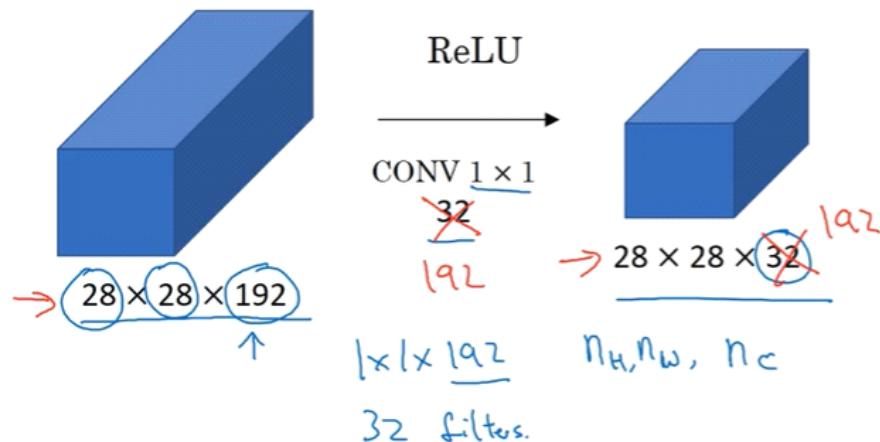
我们有输入一张 $6 \times 6 \times 1$ 的图片，然后对它做卷积，其过滤器大小为 $1 \times 1 \times 1$ ，结果相当于把这个图片乘以数字2，所以前三个单元格分别是2 4 6，用 $1 \times 1$ 的过滤器进行卷积，使用用处不大，只是对输入矩阵乘以某个数字，但这仅仅是对于 $6 \times 6 \times 1$ 的信道图片来说， $1 \times 1$ 卷积效果不佳。

如果是一张 $6 \times 6 \times 32$ 的图片，那么使用 $1 \times 1$ 过滤器进行卷积效果更好，具体来说 $1 \times 1$ 卷积所实现的功能是遍历这36单元格，计算左图中32个数字和过滤器中，32个数字的元素乘积，然后应用ReLU非线性函数，我们以其中一个单元格为例，他是这个输入层上的某个切片，用这36个数字乘以这个输入层上的 $1 \times 1$ 的切片得到一个实数，像这样把它画在输出中。

这个 $1 \times 1 \times 32$ 过滤器中的32个数字可以这样理解，一个神经元是输入是32个数字，乘以相同高度和宽度上某个切片的32个数字，这32个数字具有不同信道，然后32个权重，然后应用ReLU非线性函数，一般来说，如果过滤器不止一个，而是多个，就好像有多个输入单元，其输入内容为一个切片上所有数字，输出结果是 $6 \times 6 \times$ 过滤器数量，所以 $1 \times 1$ 卷积可以从根本上理解为这32单元都应用了一个全连接网络，全连接层的作用

是输入32个数字，标记为 $n_c^{(l+1)}$ ，在36个单元上重复过程，输出的是 $6 \times 6$ 过滤器的数量。

## Using $1 \times 1$ convolutions



我们放在下节课将

Tin et al. 2012 Network in network! Let's go on to that in the next video.



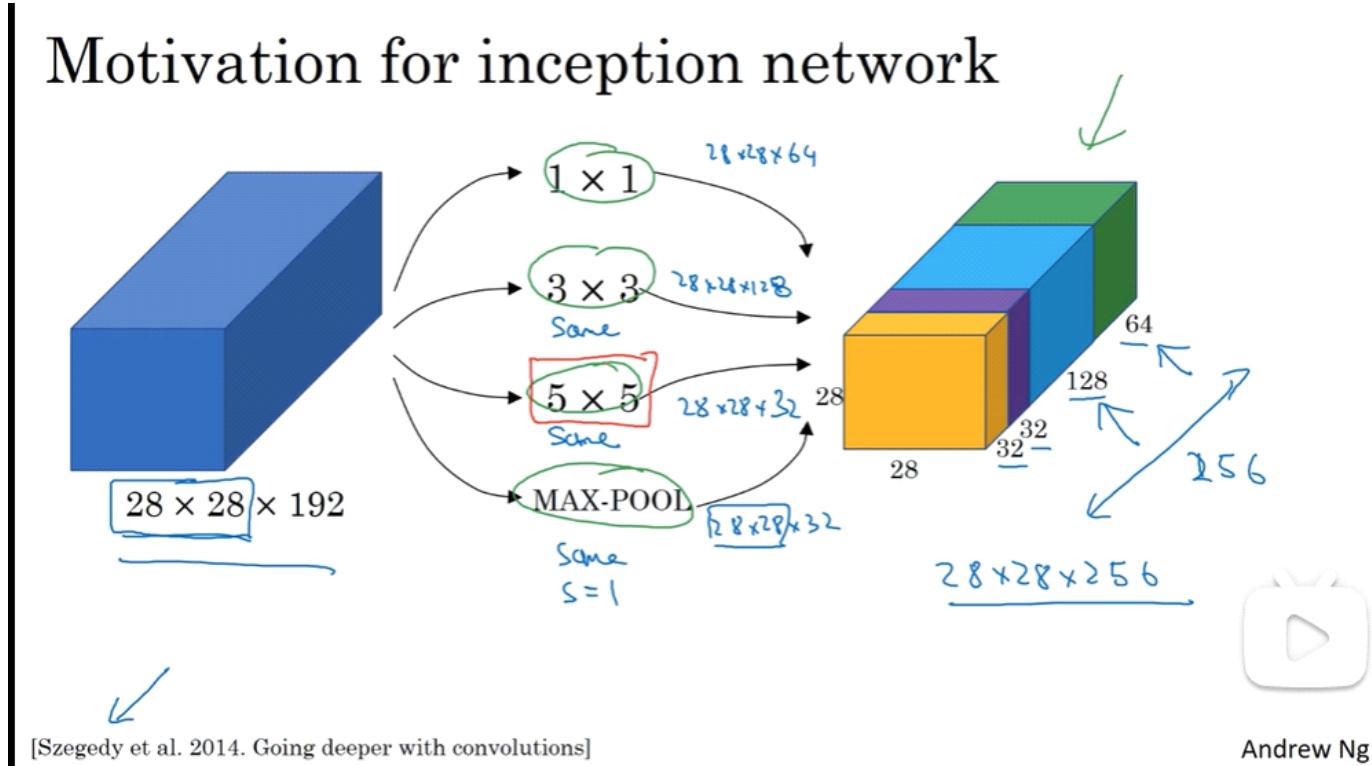
Andrew Ng

假设这是一个 $28 \times 28 \times 192$ 的输入层，我们使用池化层压缩它的高度和宽度，但是如果信道数量很大，该如何把它压缩为 $28 \times 28 \times 32$ 维度的层呢，我们可以用32个大小为 $1 \times 1$ 的过滤器，严格来讲，每个过滤器的大小都是 $1 \times 1 \times 192$ 维，因为过滤器中信道的数量必须保持一致，因此使用32个过滤器，输出层为 $28 \times 28 \times 32$ ，这是压缩 $n_C$ 的方法，然而对于池化层，我们只是压缩了这些层的高度和宽度，接下来我们看看在某些网络中 $1 \times 1$ 卷积是如何压缩信道数量并减少计算的，当然如果我们想要保持数量192不变，这也是可行的。 $1 \times 1$ 卷积只是添加了非线性函数，当然我们也可以让网络学习更复杂的函数，当然也可以让网络学习更复杂的函数。

## (15) 谷歌Inception网络简介

2022年2月28日 17:09

构建卷积层时，我们要决定过滤器的大小究竟是 $1 \times 3$ ,  $3 \times 3$ ,  $5 \times 5$ ，或者要不要添加池化层，而Inception网络的作用就是代替我们来决定，虽然网络架构因此变得更加复杂，但是网络表现却非常好，我们来了解一下其中的原理。



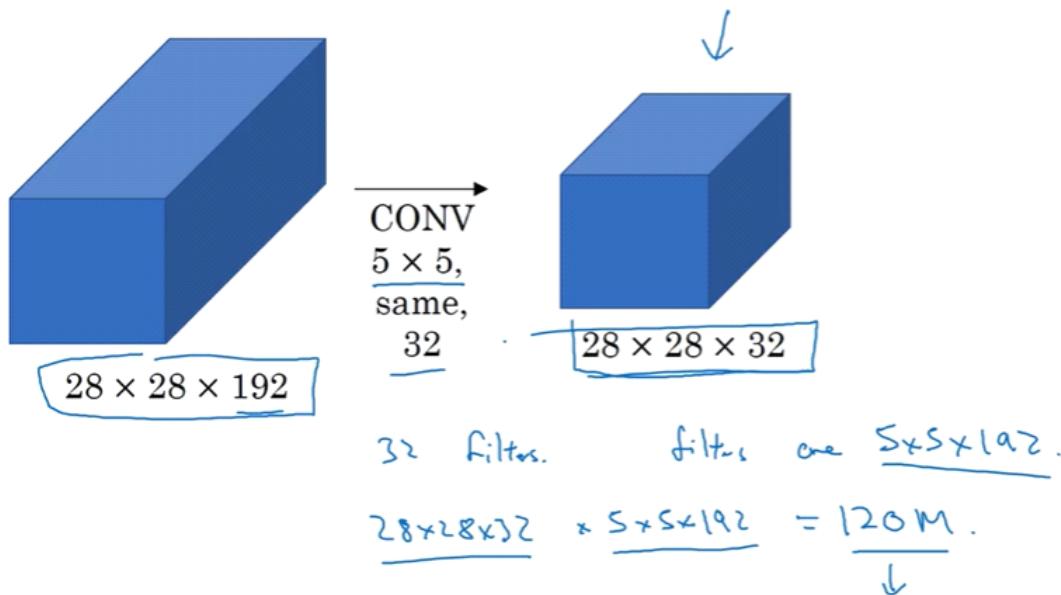
[Szegedy et al. 2014. Going deeper with convolutions]

Andrew Ng

例如，这是 $28 \times 28 \times 192$ 维度的输入层，Inception网络或Inception网络层的作用是代替人工确定卷积层中的过滤类型，或者确定是否需要创建卷积层或池化层，如果使用 $1 \times 1$ 卷积，输出结果会是 $28 \times 28 \times$ 某个值，假设输出为 $28 \times 28 \times 64$ ，并且这里只有一个层，如果用 $3 \times 3$ 的过滤器，那么输出是 $28 \times 28 \times 128$ ，然后我们把第二个值堆积到第一个值上，为了匹配维度，我们应用相同卷积，输出维度依然 $28 \times 28$ 和输入维度相同，即高度与宽度相同，这里是 $28 \times 28 \times 128$ ，或者你会说，我们想提升网络表现，用 $5 \times 5$ 过滤器或许会更好，则它的输出为 $28 \times 28 \times 32$ ，我们再次使用相同卷积，保持维度不变，或许我们不想要卷积层，那就使用池化操作，得到一些不同的输出结果，我们把它也堆积起来，这里池

化输出是 $28*28*32$ ，为了匹配所有维度，我们需要对最大池化使用padding，这是一种特殊的池化形式，因为如果输入的宽度和高度为 $28*28$ ，则输出的相应维度也是 $28*28$ ，然后再进行池化，padding不变，步幅为1，有了这样的inception模块，我们就可以输入某个量，因为它累加了所有数字，这里输出为 $32+32+128+34=256$ ，Inception模块的输入为 $28*28*129$ ，输出为 $28*28*256$ ，这就Inception网络的核心内容，基本思想是Inception网络不需要认为决定使用哪个过滤器或者需要池化，而是由网络自行确定这些参数，你可以给网络添加这些参数的所有可能的值。然后把这些输出连接起来，让网络自己学习它需要什么样的参数，采用哪些过滤器组合，不难发现，所描述的Inception层有一个问题，就是计算代成本。

## The problem of computational cost



Andrew Ng

这是一个 $28*28*192$ 的输入块，执行一个 $5*5$ 的卷积，它有32个过滤器，输出为 $28*28*32$ ，我们来计算这个 $28*28*32$ 输出的计算成本。它有32个过滤器，因为输出有32个信道，每个过滤器大小为 $5*5*192$ ，输出大小为 $20*20*32$ ，所以要计算 $28*28*32$ 个数字，对于输出的每个数字，我们都要执行 $5*5*192$ 次乘法运算，所以乘法运算的总次数为每个输出值所需的乘法，运算次数乘以输出值个数，把这些数相乘。结果

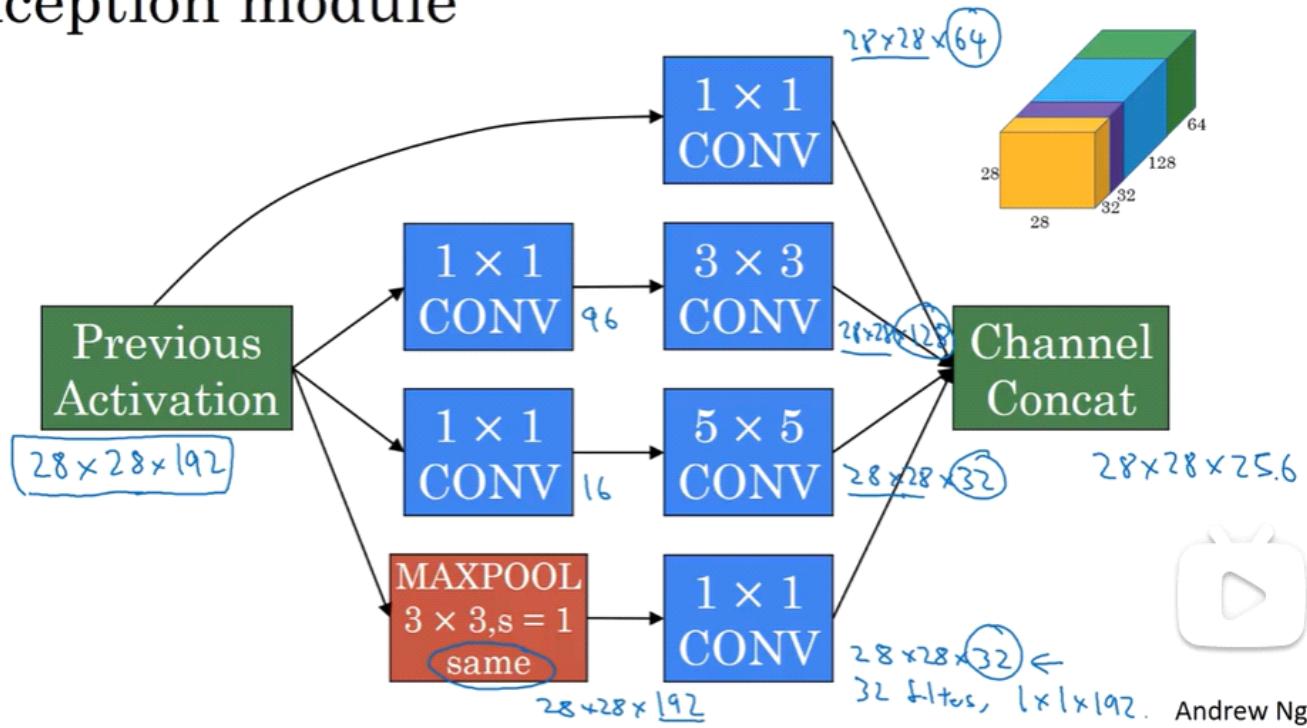
为1.2亿。

## (16) Inception 网络

2022年2月28日 19:57

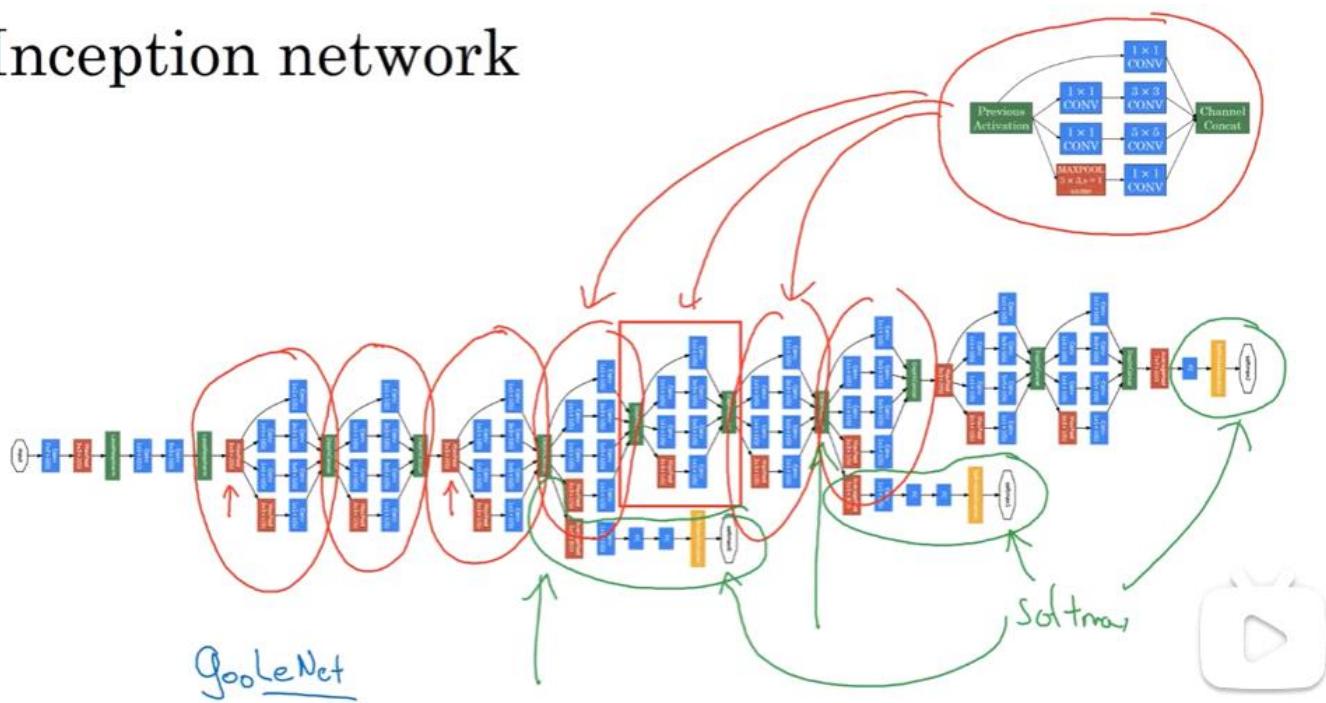
在这节中，我们将学到如何将这些模块组合起来，构筑我们自己的Inception网络。

### Inception module



我们有Inception模型。

# Inception network



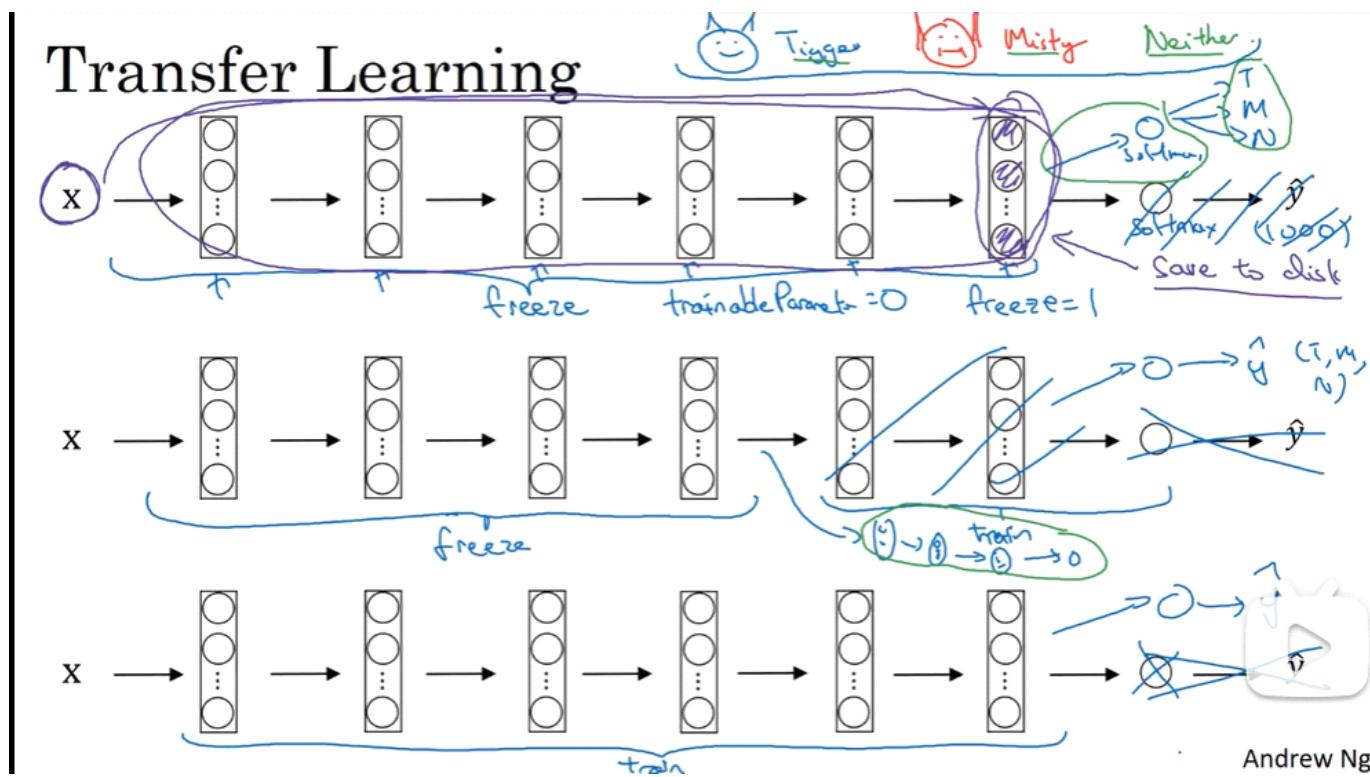
[Szegedy et al., 2014, Going Deeper with Convolutions]

Andrew Ng

## (17) 迁移学习

2022年2月28日 20:09

如果我们要做一个计算机视觉的应用，相比于从头训练权重，或者说从随机初始化权重开始，如果我们下载别人已经训练好网络结构的权重，我们通常能够进展的相当快，用这个作为预训练，然后转换到我们感兴趣的任務上。



我们通过别从已经训练好的参数，然后我们再一些层的进行训练，最后通过Softmaxf进行输出。

## (1) 目标定位

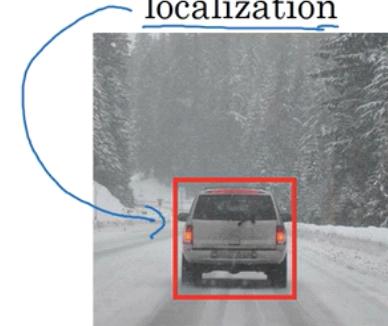
2022年2月28日 20:17

# What are localization and detection?

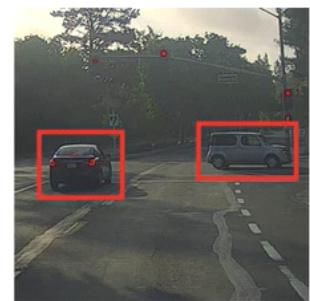
Image classification



Classification with  
localization



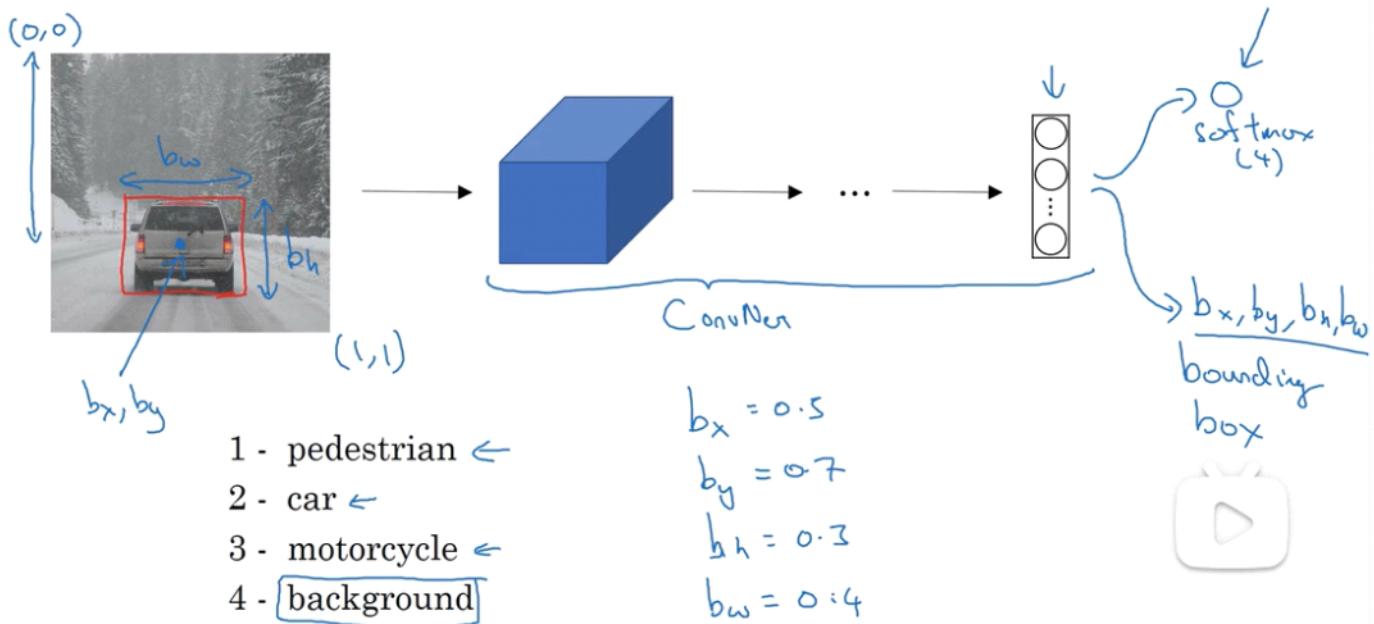
Detection



这节课我们要学习定位分类问题，这意味着，我们不仅要用算法判断图片中是不是一辆汽车，还要在图片中标记出它的位置，用边框或红色方框把汽车圈起来，这就是定位分类问题。

而在对象检测问题中，图片可以含有多个对象，甚至单张图片中会有多个不同分类的对象，因此，图片分类的思路可以帮助学习分类定位，而对象定位的思路又有助于学习对象检测。

# Classification with localization



假设，输入一张图片到多层卷积神经网络，它会输出一个特征向量，并反馈给SoftMax单元来预测图片类型，那么对象可能包括以下几类：

- 1、行人
- 2、车辆
- 3、摩托车
- 4、背景，即图片中不含有前三种对象。

这四个分类就是SoftMax函数可能输出的结果，这就是标准的分类 Pipeline，如果还想定位图片中汽车的位置，那么该怎么做呢？

我们可以让神经网络多输出几个单元，输出一个边界框，具体说让神经网络再多输出四个数字 $b_x, b_y, b_h, b_w$ ，这四个数字是被检测对象边界框的参数化表示。

我们约定符号表示，图片左上角的坐标为 $(0, 0)$ ，右下角的坐标为 $(1, 1)$ ，要想确定边界框的具体位置，需要指定红色方框的中心点，这个点表示 $(b_x, b_y)$ ，边界框的高度为 $b_h$ ，宽度为 $b_w$ 。

所以训练集不仅包括神经网络要预测的对象分类标签，还包含表示边界框的这四个数字，接着采用监督学习，输出一个分类标签，还有这四个参数的值，从而给出被检测对象的边界框位置，此例中 $b_x$ 的理解值是0.5， $b_y$ 大约是0.7，表示汽车位于距离图片底部3/10的位

置， $b_h$ 约为0.3，因为红色方框的高度是图片高度的0.3倍， $b_w$ 约为0.4，红色方框的宽度为图片宽度的0.4。

## Defining the target label $y$

- 1 - pedestrian
- 2 - car ↗
- 3 - motorcycle
- 4 - background ↗

$$l(\hat{y}, y) = \begin{cases} (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 \\ + \dots + (\hat{y}_8 - y_8)^2 & \text{if } y_1 = 1 \\ (\hat{y}_1 - y_1)^2 & \text{if } y_1 = 0 \end{cases}$$

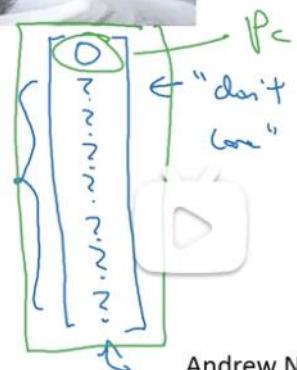
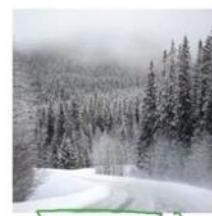
Need to output  $b_x, b_y, b_h, b_w$ , class label (1-4)



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

is there an object?

$(x, y)$



Andrew Ng

如何为监督学习任务定义目标标签 $y$ ，有四个分类，神经网络输出的是这四个数字和一个分类标签或者分类标签出现的概率。

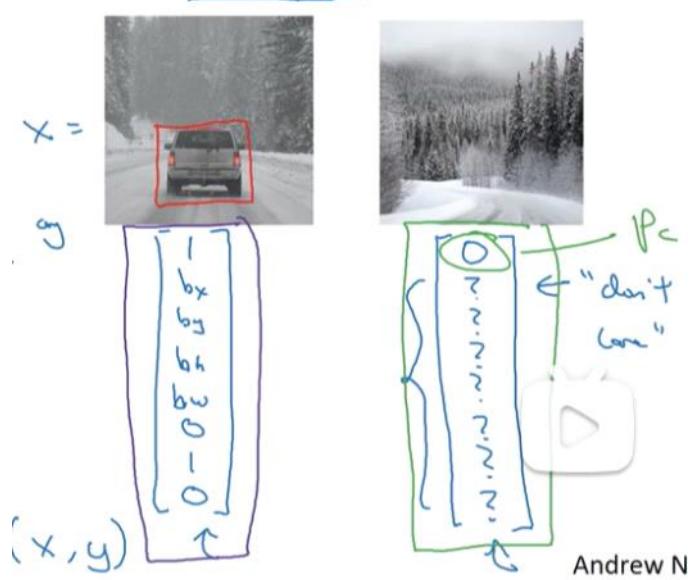
目标标签Y定义如下：

$$y = \begin{pmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}$$

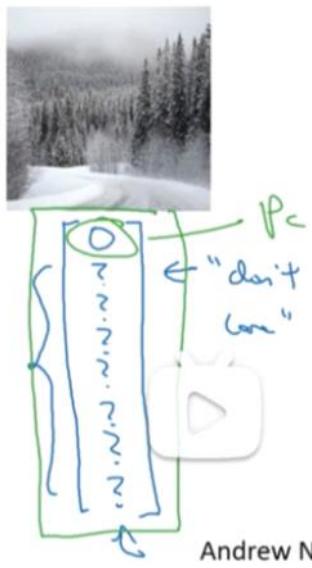
$p_c$ 表示是否含有对象，如果对象属于前三类，则 $p_c = 1$ ，如果是背景，则图片中没有要检测的对象， $p_c = 0$ 。

如果检测到对象，就输出被检测对象的边界框参数 $b_x, b_y, b_h, b_w$ 。

如果存在某个对象， $p_c = 1$ ，同时输出 $c_1, c_2, c_3$ ，表示对象属于1-3的哪一类，是行人、汽车还是摩托车。我们假设图种只含有一个对象，所以针对这个分类定位问题，图片最多只会出现一个对象。



假如这是一张训练集的图片，标记为X, Y当中，第一个元素 $p_c = 1$ ，因为图中有一辆车， $b_x, b_y, b_h, b_w$ 指明边界框的位置，所以标签训练集需要标签的边界框，图片中是一辆车，所以结果属于分类2，因为定位目标不是行人或摩托车，而是汽车，所以 $c_1 = 0, c_2 = 1, c_3 = 0$ ， $c_1, c_2, c_3$ 最多有一个等于0，这是图片中只有一个检测对象的情况，如果图片中没有检测对象呢？



假设训练的是一张这样的图片，在此处键入公式。这种情况下  $p_c = 0$ ，Y 的其它参数将没有意义，因为图片中不存在检测对象，所以不用考虑网络输出中边界框的大小，也不用考虑图片中的对象是属于

$c_1, c_2, c_3$  的哪一类。

针对给定的被标记的训练样本，不论图片中是否含有定位对象，构建输入图上 $X$ 和分类标签 $Y$ 的具体过程都是如此。最后介绍一下训练神经网络的损失函数。

$$\mathcal{L}(\hat{y}, y) = (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 + (\hat{y}_8 - y_8)^2 \quad y = 1$$

其参数为类别 $y$ 和网络输出 $\hat{y}$ ，因为当 $p_c = 1$ 时，有八个参数

$$\mathcal{L}(\hat{y}, y) = (\hat{y}_1 - y_1)^2 \quad y = 0$$

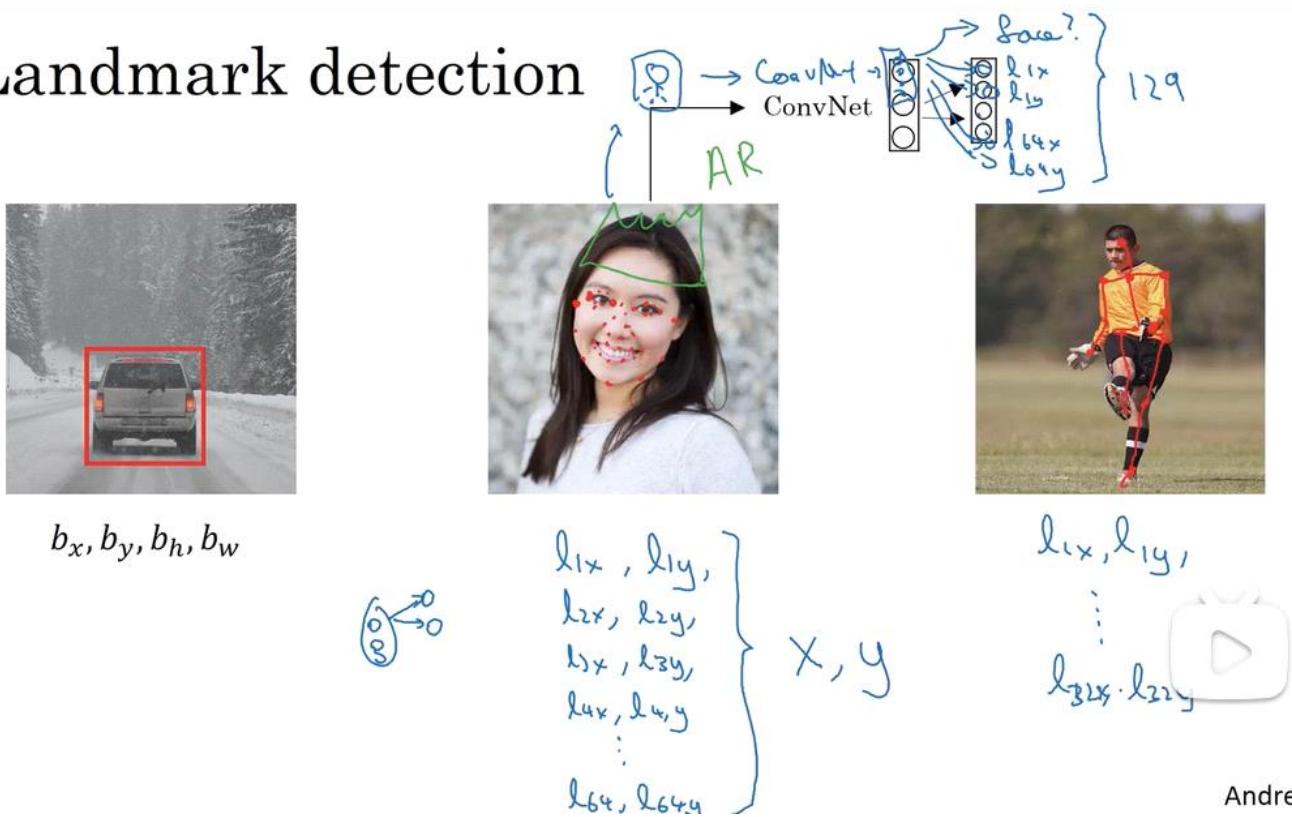
即 $p_c = 0$ ，我们就不用考虑后面的情况了。

## (2) 特征点检测

2022年2月28日 21:41

神经网络可以通过输出图片上特征点的(x,y)的坐标来实现对目标特征的识别。

### Landmark detection



假设我们在进行人脸识别程序，我们希望算法可以给出眼角的具体位置，眼角坐标为 $x, y$ ，我们可以让神经网络的最后一层，多输出  $l_x, l_y$ ，作为眼角的坐标值。

如果想知道两只眼睛的四个眼角的具体位置，那么，从左到右依次用四个特征点来表示这四个眼角，对神经网络稍微做些修改，输出第一个特征点( $l_{1x}, l_{1y}$ )，第二个特征点( $l_{2x}, l_{2y}$ )，依次类推，这四个脸部特征点的位置就可以通过神经网络输出了。

若我们还想要得到更多特征点输出值，一些眼睛的特征点，还可以根据嘴部的关键点输出值，来确定嘴的形状，从而判断人物是在微笑还是在皱眉，也可以提取鼻子周围的关键特征点，为了方便，我们假设有脸部64个特征点，我们生成包含这些特征点的标签训练集。然后利用神经网络输出脸部关键特征点的位置。

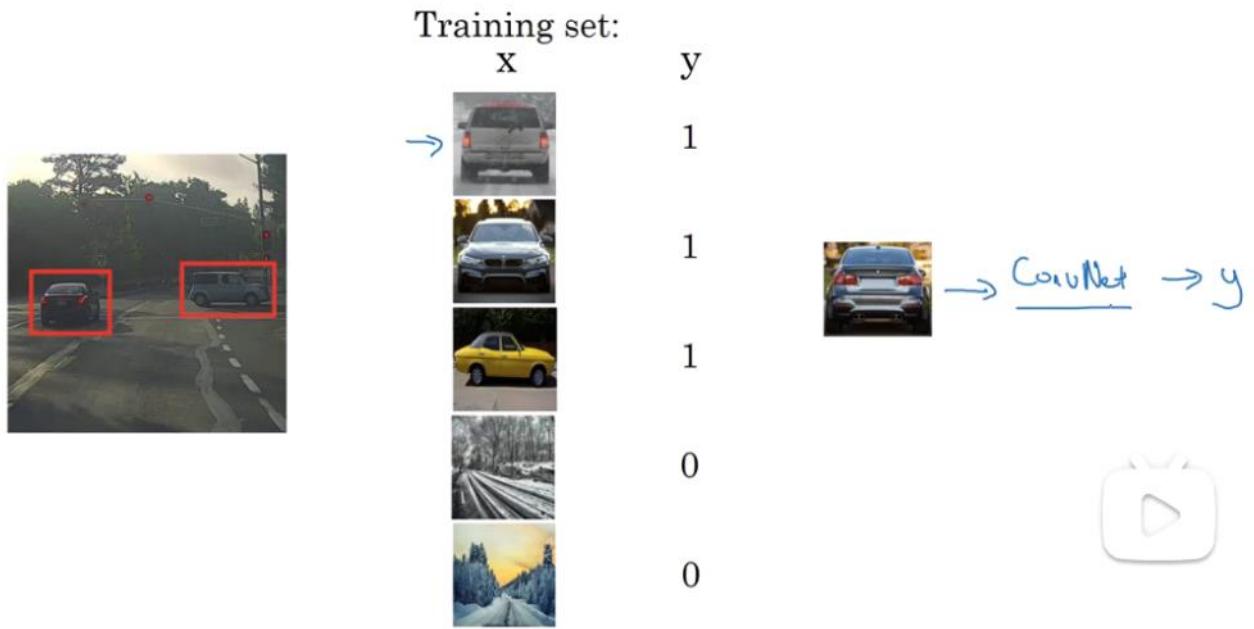
具体做法是，准备一个卷积网络和一些特征集，将人脸图片输入卷积网络，输出0或1，1表示有人脸，0表示没有，然后输出 $(l_{1x}, l_{1y}) \dots (l_{64x}, l_{64y})$ 。

### (3) 目标检测

2022年3月1日 10:35

我们将学习如何通过卷积网络进行对象检测，采用是基于滑动窗口的目标检测算法。

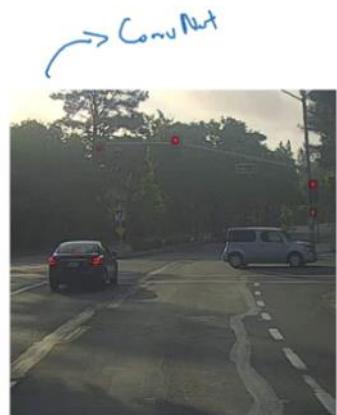
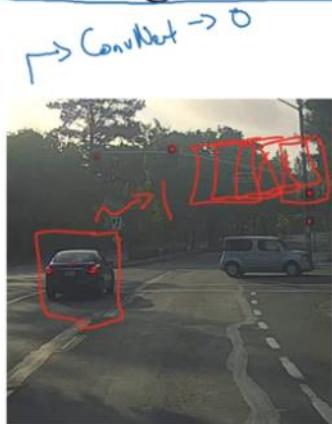
## Car detection example



假如我们想要构建一个汽车检测算法，，步骤是

首先创建一个标签训练集，也就是X和Y表示适当剪切的汽车图片样本，这张图片X是一个正样本，因为它需要一辆汽车图片，出于我们对于这个训练集的期望，我们可以一开始可以使用适当剪切的图片，就是整张图片X几乎都被汽车占据。我们可以照张照片，然后进行剪切。剪掉汽车以外的部分。使汽车居于中心位置，并基本占据整张图片，有了这个训练集，我们就可以开始训练卷积网络了，输入这些适当剪切过的图像，卷积网络输出 Y，0或1表示图片中有没有汽车。训练完这个网络，就可以用它来实现滑动窗口目标检测。

## Sliding windows detection



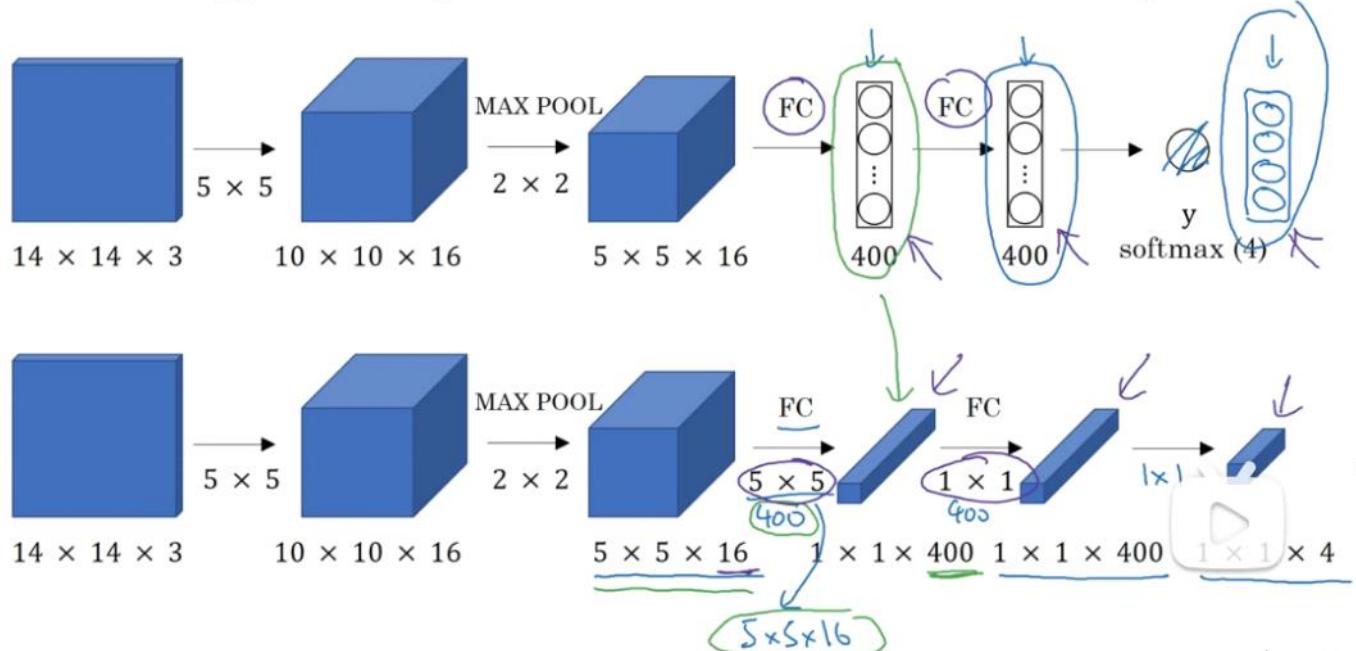
Computation cost

## (4) 卷积的滑动窗口实现

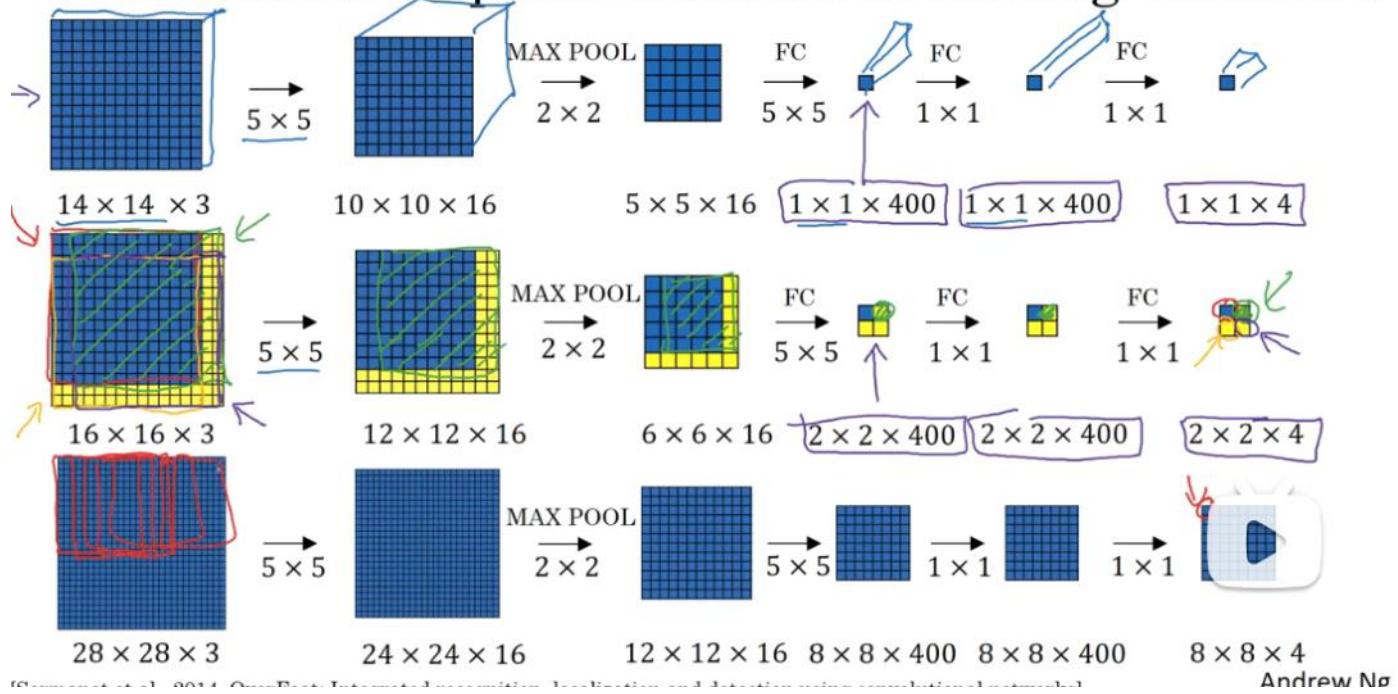
2022年3月1日 11:04

今天我们将讲如何在卷积层上应用滑动窗口算法。

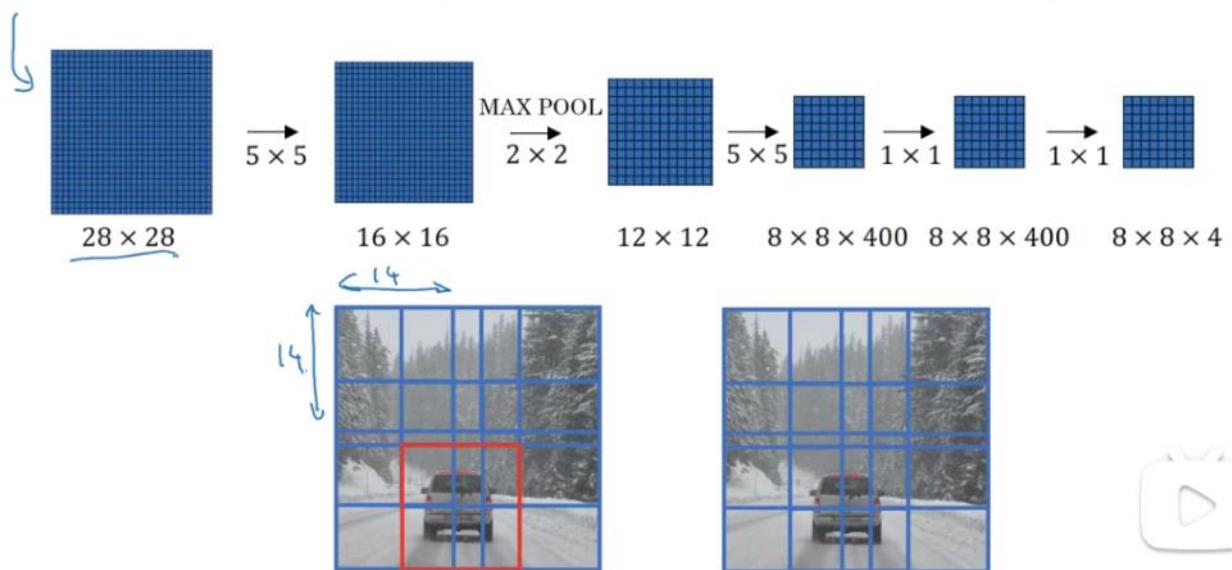
### Turning FC layer into convolutional layers



### Convolution implementation of sliding windows



# Convolution implementation of sliding windows



总结滑动窗口的实现：

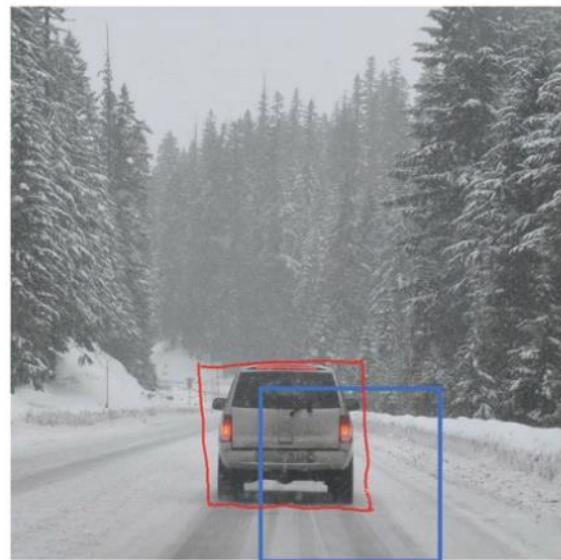
在图片上剪切一块区域，假设它的大小是 $14 \times 14$ ，把它输入到卷积网络中，继续输入下一块区域，大小同样是 $14 \times 14$ ，重复操作，直到某个区域识别到汽车，我们不用依靠连接的卷积来识别图片中的汽车，比如，我们可以对大小为 $28 \times 28$ 的整张图片进行卷积操作，一次得到所有预测的值。

## (5) Bounding Box预测

2022年3月1日 15:24

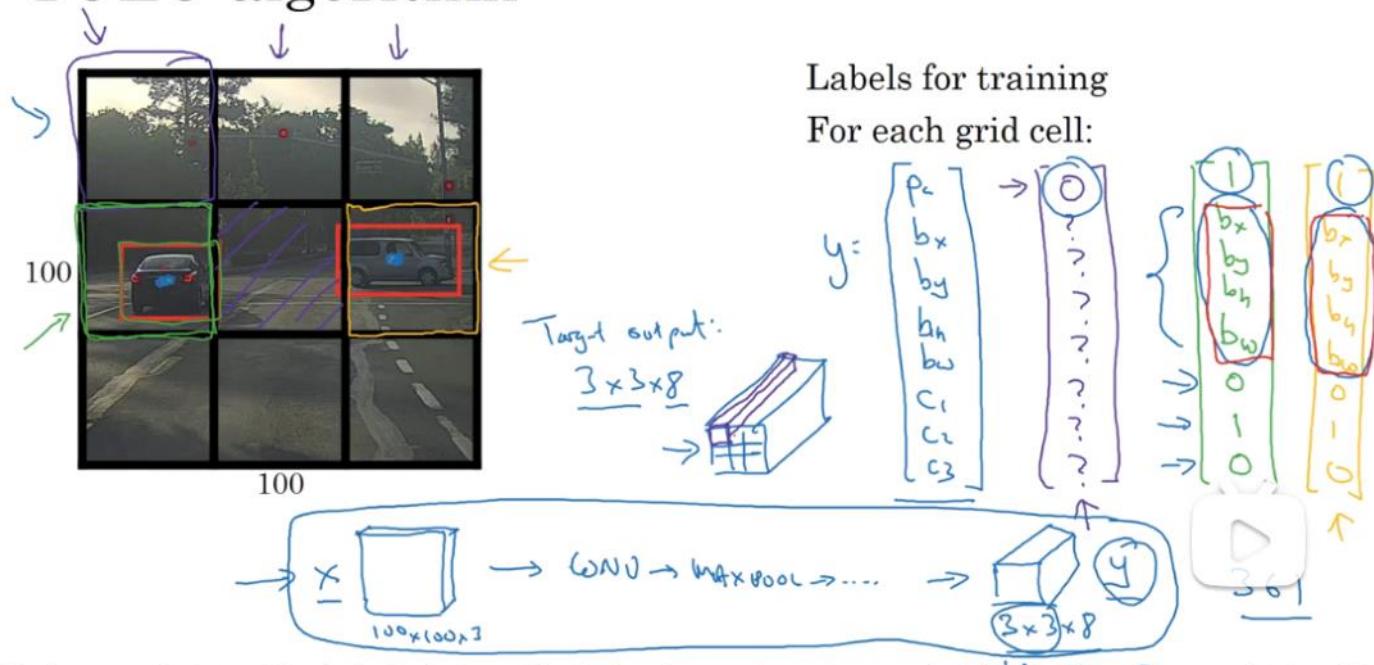
卷积的滑动窗口实现虽然效率较高，但不能输出最精准的边界框。

## Output accurate bounding boxes



在这张图中，我们实现卷积的滑动窗口，但这并不能很好的去识别汽车的边界框。

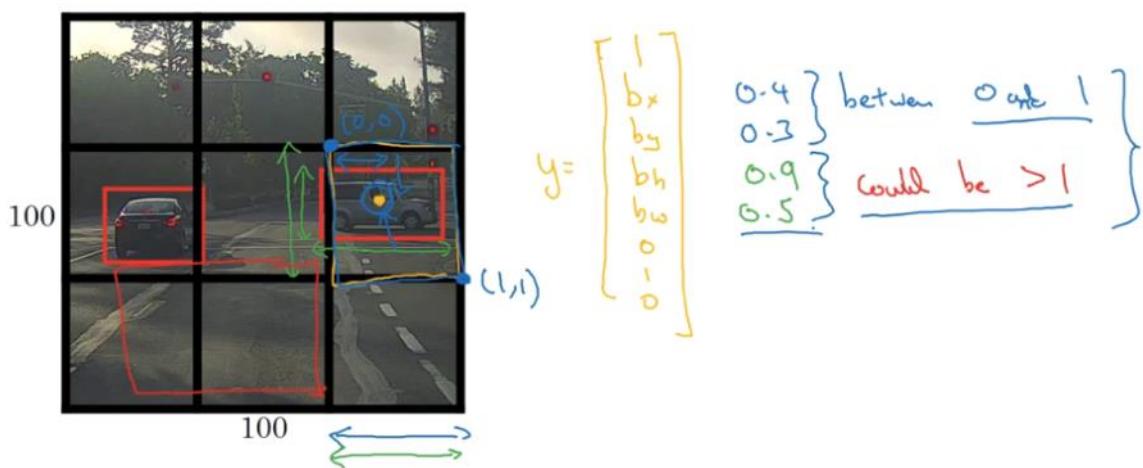
## YOLO algorithm



YOLO算法的基本思路是使用图像分类和定位算法，然后将算法应用到9个格子上，更为具体的我们对于9个格子中的每一个都指定一个标签 $y$ ，YOLO算法做的是取两个对象的中点，然后将这个对象分配给包含对象中点的格子上。

因为这里有 $3 \times 3$ 的格子，然后对于每个格子，我们都还有一个8维向量 $Y$ ，所以目标输出尺寸是 $3 \times 3 \times 8$ ，这个算法的优点在于神经网络可以输出精确的边界框。重申一下，我们把对象分配到一个格子的过程是，我们观察对象的中点，然后将这个对象分配到其中点所在的格子，所以即使对象可以横跨多个格子，也只会被分配到9个格子其中之一。

## Specify the bounding boxes



[Redmon et al. 2015. You Only Look Once: Unified real-time object detection]

Andrew Ng

我们如何指定边框？

在YOLO算法中，对于这个方框，我们约定左上这个点是 $(0,0)$ ，然后右下这个点是 $(1, 1)$ ，要指定橙色中点的位置，从图中看 $b_x$ 大于是0.4， $b_y$ 大约是0.3， $b_h$ 大约是0.9， $b_w$ 大约是0.5。

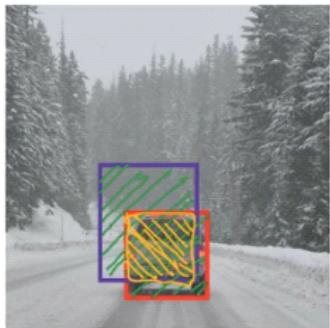
即 $b_x$ 和 $b_y$ 大约在0到1之间， $b_w$ 和 $b_h$ 在1之外是比较合理的。

## (6) 交并比

2022年3月1日 16:05

如果判断对象检测算法动作良好呢？本节将介绍交并比函数，这个用来评价对象检测算法。

### Evaluating object localization



Intersection over Union (IoU)

$$= \frac{\text{Size of } \begin{array}{c} \text{yellow} \\ \cap \end{array}}{\text{Size of } \begin{array}{c} \text{yellow} \\ \cup \text{green} \end{array}}$$

“Correct” if  $\text{IoU} \geq 0.5$  ←

0.6 ←



More generally, IoU is a measure of the overlap between two bounding boxes.

Andrew Ng

在对象检测任务中，我们希望能够同时定位对象，实际边界为红色条线，算法给出的边际为紫色线条，那么这个结果是好还是坏？

所以交并比函数做的是计算两个边界框交集和并集之比，两个边界框的并集是绿色和黄色重合区域，那么交并比就是计算交集的大小，即黄色区域的面积。

即有公式

$$IoU = \frac{\text{黄色区域}}{\text{绿色区域}}$$

一般约定在计算机中若IoU大于0.5，那么结果是可以接受的。

## (7) 非极大值抑制

2022年3月1日 16:12

到目前为止，我们的算法是对同一对象做出多次检测，所以算法不是对某个对象检测出一次，而是检测出多次。非最大值抑制可以确保我们的算法对每个对象只检测一次，

### Non-max suppression example



Andrew Ng

假设我们需要在这张图中检测行人和汽车，我们可能在这张图上放 $19 \times 19$ 个格子，理论上这辆车只有一个中点，所以它应该分配到一个格子中，实践中跑对象分类和定位 算法时，对于每个格子都跑一次，所以这个格子可能会想这辆车中点应该在格子内部，这几个格子可能也会这样想

# (1) 数学符号

2022年4月12日 20:36

## Motivating example

NLP

x: Harry Potter and Hermione Granger invented a new spell.  
 $\rightarrow x^{(1)} \quad x^{(2)} \quad x^{(3)} \quad \dots \quad x^{(t)} \quad \dots \quad x^{(9)}$   
 $T_x = 9$

$\rightarrow y: \quad | \quad | \quad 0 \quad | \quad | \quad 0 \quad 0 \quad 0 \quad 0$   
 $y^{(1)} \quad y^{(2)} \quad y^{(3)} \quad \dots \quad y^{(t)} \quad \dots \quad y^{(9)}$   
 $T_y = 9$

$x^{(i)<t>} \quad T_x^{(i)} = 9 \quad 15$   
 $y^{(i)<t>} \quad T_y^{(i)}$

Andrew Ng

我们有句子X: Harry Potter and Hermione Granger invented a new spell.

我们有Y: 1 1 0 1 1 0 0 0

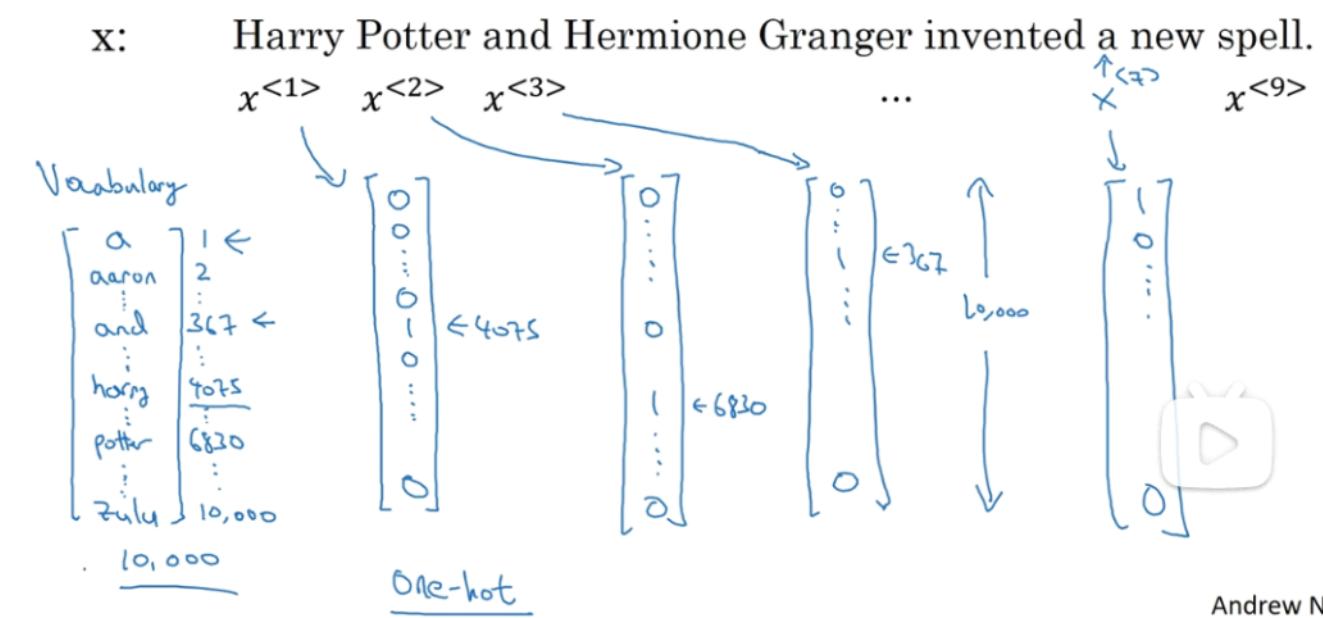
$$y^{(1)}, y^{(2)}, y^{(3)}, \dots, y^{(9)}$$

其中 $x^{(i)<t>}$ 表示第*i*个样本中第*t*个序列，我们有 $T_x$ 表示输入样本的序列长度，而符号 $T_x^{(i)}$ 表示第*i*个样本的输入序列长度。

符号 $y^{(i)<t>}$ 表示第*i*个样本中训练样本中第*t*个序列，符号 $T_y^{(i)}$ 表第*i*个训练样本的输出长度。

# Representing words

$$x^{(t)} \rightarrow y^{(t)}$$



Andrew Ng

我们有句子X:Harry Potter and Hermione Granger invented a new spell.

我们可以通过词典来进行映射

Vocabulary

我们有词汇

A,aaron,...,and,...harvy,potter,...,zulu

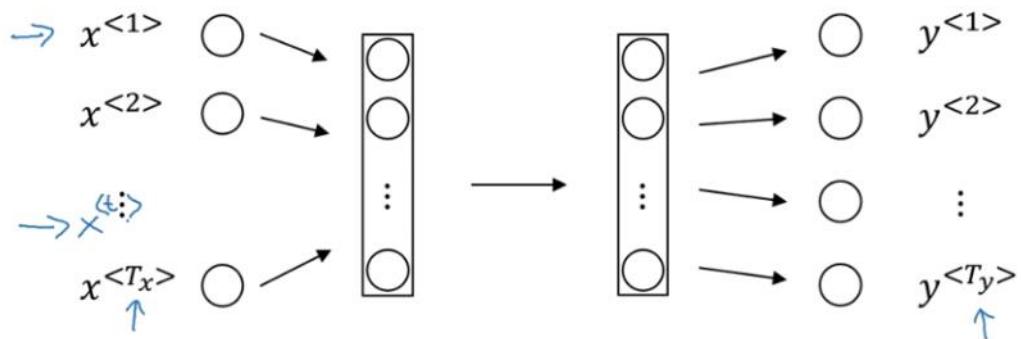
1,2,...,367,...,4075,...,6830,...,10,000

通过词典的映射把单词映射成为一个个数字,继而我们可以one-hot的方式来对单词进行数字化表示。

## (2) 循环神经网络

2022年4月12日 21:51

### Why not a standard network?



#### Problems:

- Inputs, outputs can be different lengths in different examples.
- Doesn't share features learned across different positions of text.

Andrew Ng

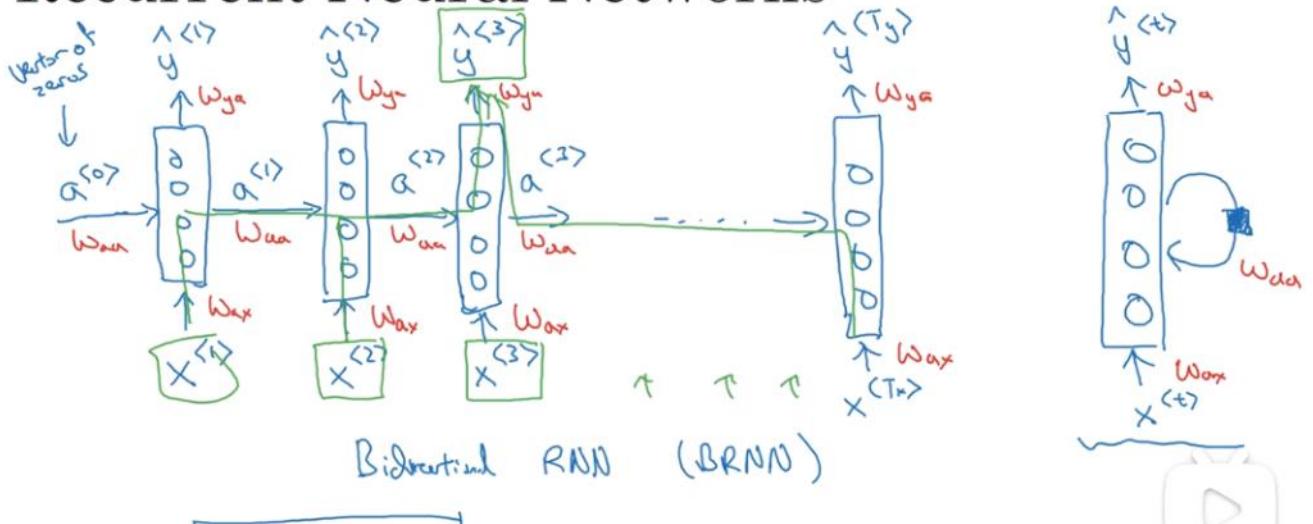
#### 为什么不使用标准的神经网络？

一、因为在不同的例子中，输入与输出有着不同的长度。

二、单纯的神经网络不共享从文本的不同位置上学到的特征。

具体来说，如果神经网络已经学习到了在位置 $x^{(1)}$ 出现的Harray可能是人名的一部分，那么如果Harray出现在比如 $x^{(5)}$ 的位置上，我们也希望能够自动识别其为人名的一部分。——这是标准神经网络无法做到的。

# Recurrent Neural Networks



He said, "Teddy Roosevelt was a great President."

He said, "Teddy bears are on sale!"

Andrew I

上图为循环神经网络，在上图的循环神经网络中，在预测 $y^{(3)}$ 时，不仅要使用 $x^{(3)}$ 的信息，还需要使用 $x^{(1)}, x^{(2)}$ 的信息来帮助预测 $y^{(3)}$ 。

如，我们有句子

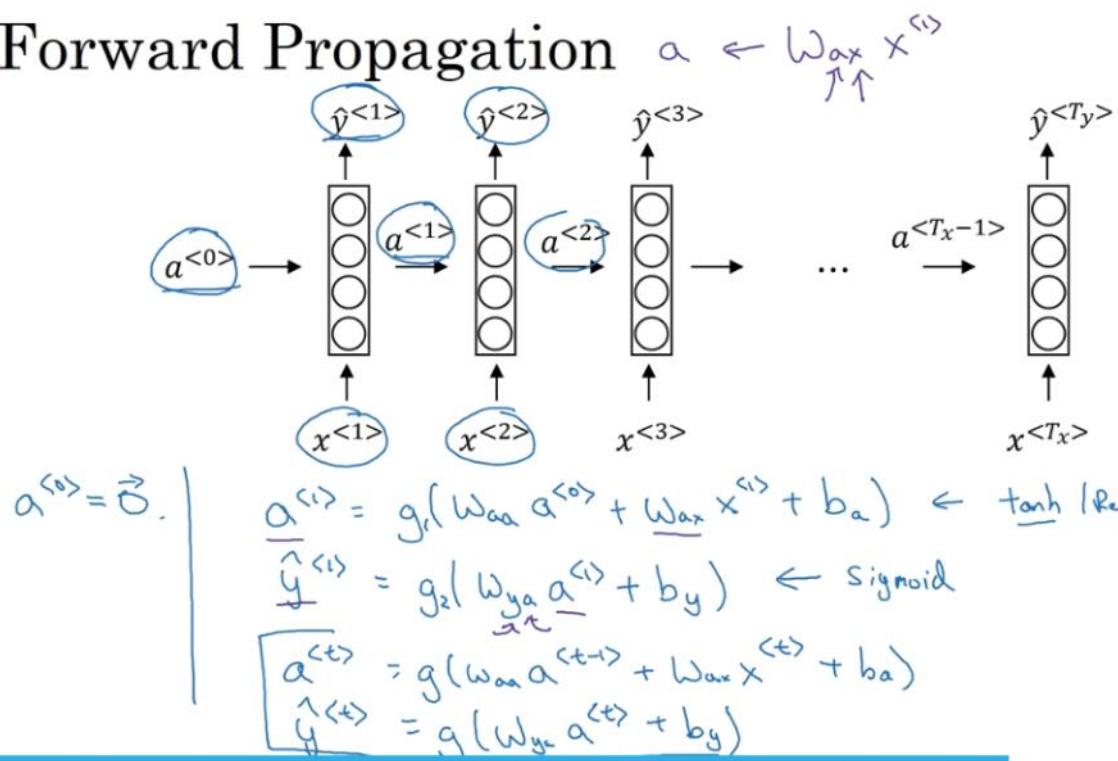
He said, "Teddy Roosevelt was a great President."

为了判断Teddy是否是人名，我们仅通过he said来判断是远远不够的，因为也可能是这样的句子。

He said, "Teddy bears are on sale!"

所以仅通过前面的信息来预测是不够，我们还需要借助后面的信息。

# Forward Propagation



Andrew Ng

我们有向量  $a^{<0>} = \vec{0}$ ,

$$a^{<1>} = g_1(w_{aa} a^{<0>} + w_{ax} x^{<1>} + b_a) \quad \text{激活函数为 tanh/ReLU}$$

$$\hat{y}^{<1>} = g_2(w_{ya} a^{<1>} + b_y) \quad \text{激活函数 sigmoid}$$

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

$$\hat{y}^{<1>} = g(w_{ya} a^{<t>} + b_y)$$

## Simplified RNN notation

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

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

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

$[W_{aa}; W_{ax}] = \begin{bmatrix} 100 & \\ & 1000 \end{bmatrix} = W_a$

$[a^{<t-1>}, x^{<t>}] = \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix}$

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

Andrew Ng

RNN简写的注意事项：

$$a^{} = g(W_{aa}a^{} + W_{ax}x^{} + b_a)$$

我们可以简写成

$$a^{} = g(W_a[a^{}, a^{}] + b_a)$$

$$[W_{aa}|W_{ax}] = W_a$$

$$[W_{aa}|W_{ax}] \begin{bmatrix} a^{} \\ a^{} \end{bmatrix} = W_{aa}a^{} + W_{ax}x^{}$$

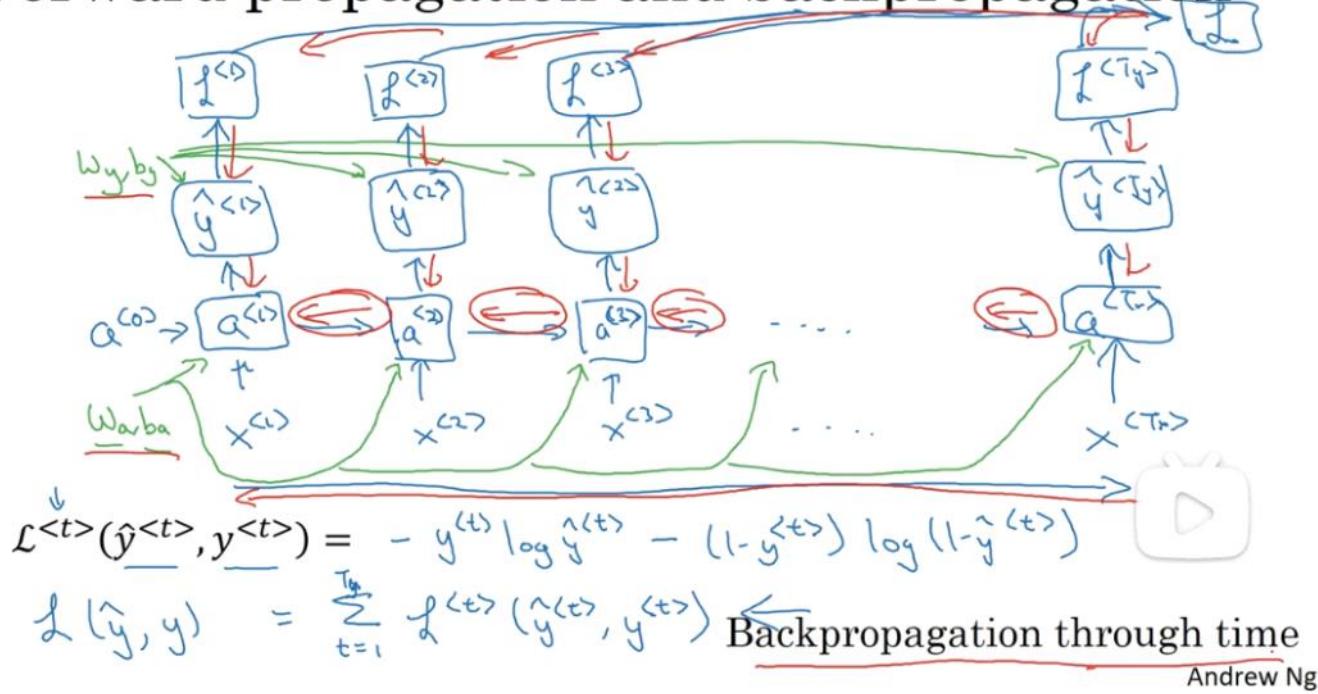
$\hat{y}^{} = g(W_{ya}a^{} + b_y)$  可以简写成

$$\hat{y}^{} = g(W_ya^{} + b_y)$$

### (3) 通过时间的反向传播

2022年4月12日 23:13

## Forward propagation and backpropagation



我们由  $x^{<1>} \rightarrow a^{<1>} \rightarrow \hat{y}^{<1>} \rightarrow \tau^{<1>}$ , 即我们有  
 $x^{<t>} \rightarrow a^{<t>} \rightarrow \hat{y}^{<t>} \rightarrow \tau^{<t>}$

其中

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

我们需要把所有序列的损失函数相加有

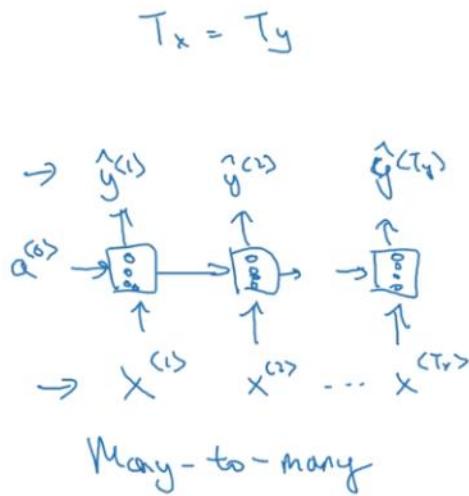
$$\mathcal{L}(\hat{y}, y) = \sum_{t=1} \tau^{<t>}(\hat{y}^{<t>}, y^{<t>})$$

以来优化这个目标函数  $\mathcal{L}(\hat{y}, y)$ , 然后我们需要通过反向传播来求导数来进行梯度下降所需要数据。

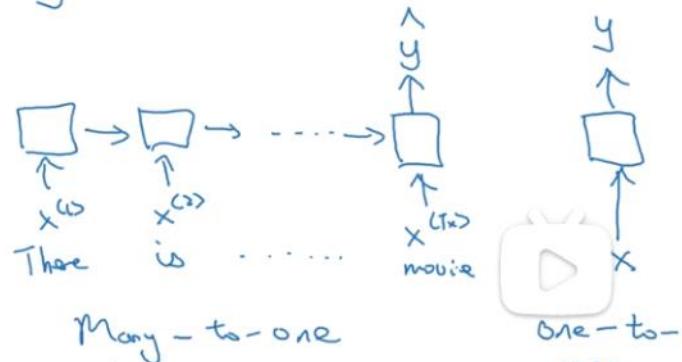
## (4) 不同类型的循环神经网络

2022年4月13日 10:40

### Examples of RNN architectures

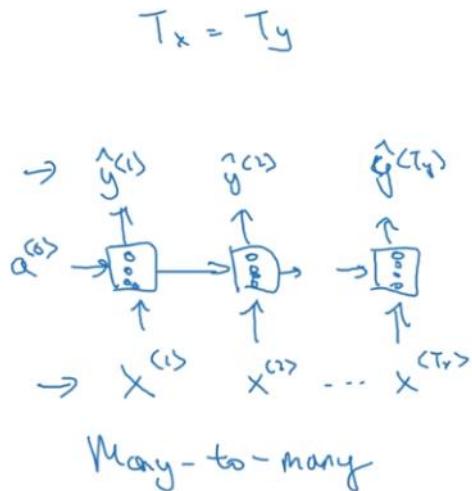


Sentiment classification  
 $x = \text{text}$   
 $y = 0/1 \quad 1\cdots 5$



Andrew Ng

当输入长度与输出长度相同，我们有多对多结构。

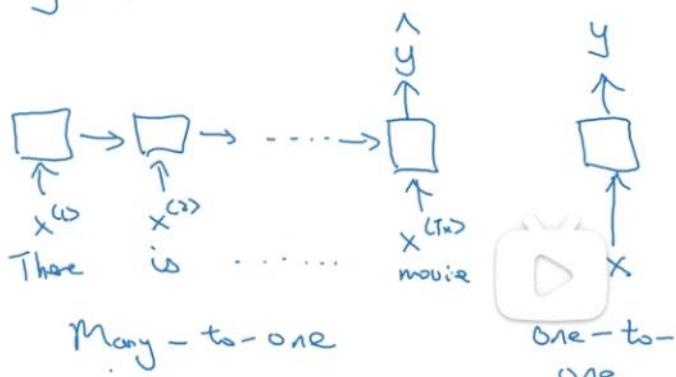


我们在进行情感文本分类中，如输入X为一段文本，输出Y可能为从0到5的数字，通常采用多对一结构。

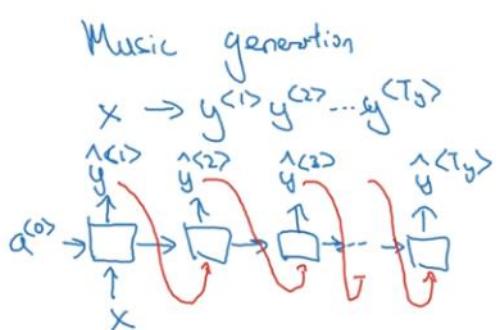
Sentiment classification

$x = \text{text}$

$y = 0/1 \quad 1 \dots 5$

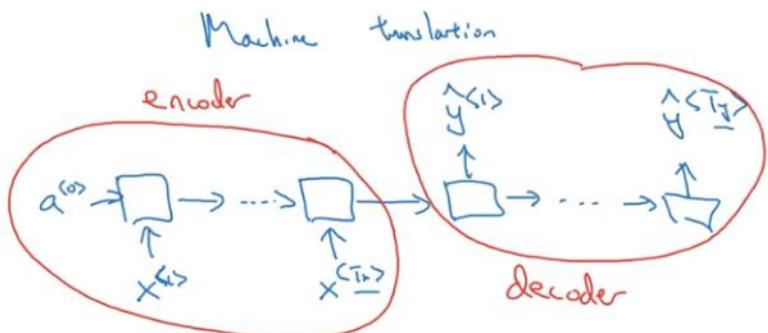


## Examples of RNN architectures



One-to-many

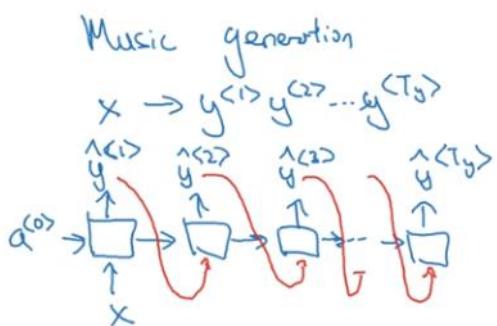
$x = \phi$



Many-to-many



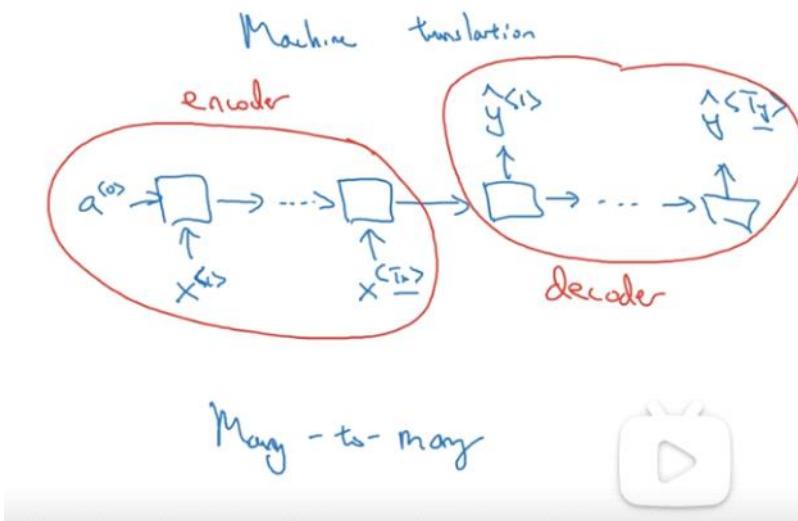
我们在进行音乐生成时，通常采用一多对的结构



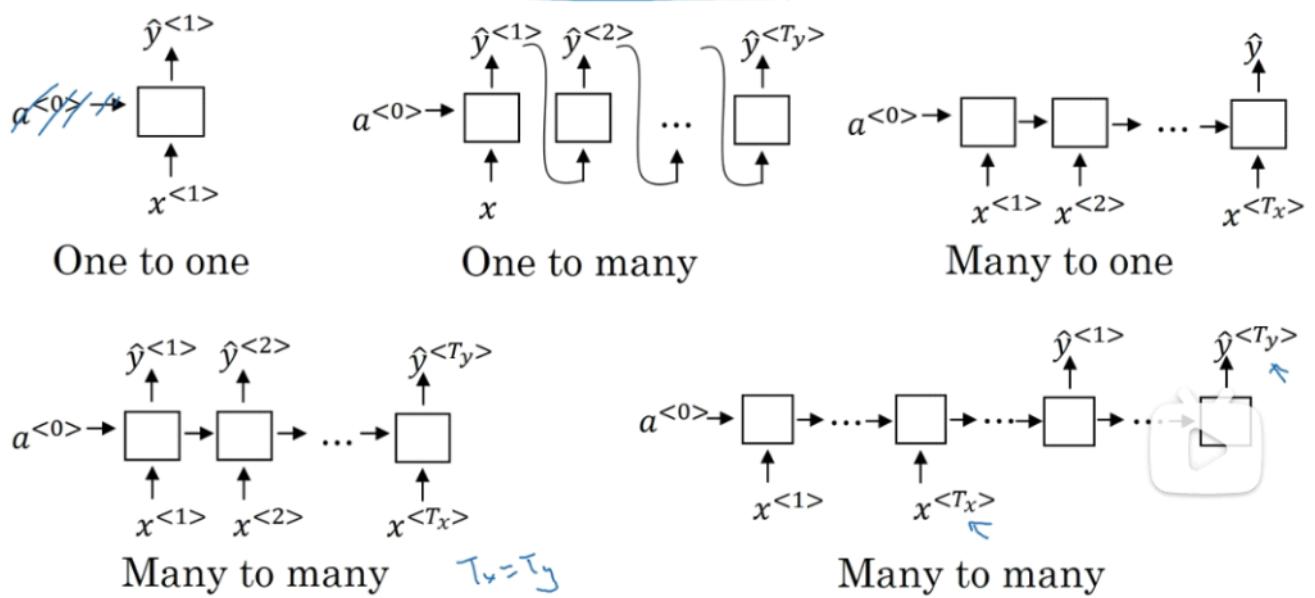
One-to-many

$x = \phi$

我们在进行机器翻译时，通常采用多对多结构



## Summary of RNN types



## (5) 语言模型与序列生成

2022年4月13日 11:03

# What is language modelling?

Speech recognition

The apple and pair salad.

→ The apple and pear salad.

$$P(\text{The apple and pair salad}) = 3.2 \times 10^{-13}$$

$$P(\text{The apple and pear salad}) = 5.7 \times 10^{-10}$$

$$P(\text{sentence}) = ?$$

$$P(y^{<1>} , y^{<2>} , \dots , y^{<T_y>})$$



我们有句子文本

文本1: The apple and pair salad.

文本2: The apple and pear salad.

我们通过语言模型得到的文本1的概率为

$$P(\text{The apple and pair salad}) = 3.2 \times 10^{-13}$$

$$P(\text{The apple and pear salad}) = 5.7 \times 10^{-10}$$

那么我们有语言模型

$$P(y^{<1>} , y^{<2>} , \dots , y^{<T_y>})$$

# Language modelling with an RNN

Training set: large corpus of english text.

Tokenize

Cats average 15 hours of sleep a day.  $\downarrow <\text{EOS}>$

$y^{(1)} \quad y^{(2)} \quad y^{(3)} \quad \dots \quad y^{(8)} \quad y^{(9)}$   
 $x^{(1)} = y^{(t-1)}$

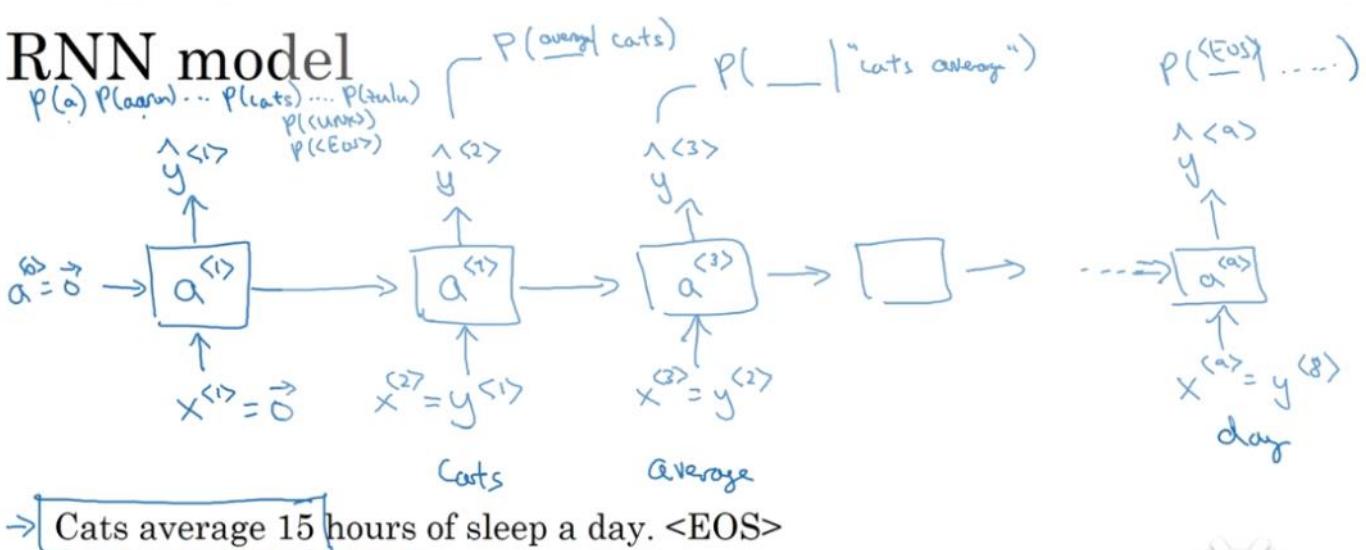
The Egyptian Mau is a bread of cat.  $<\text{EOS}>$

10,000

$<\text{UNK}>$

Andrew Ng

## RNN model



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

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

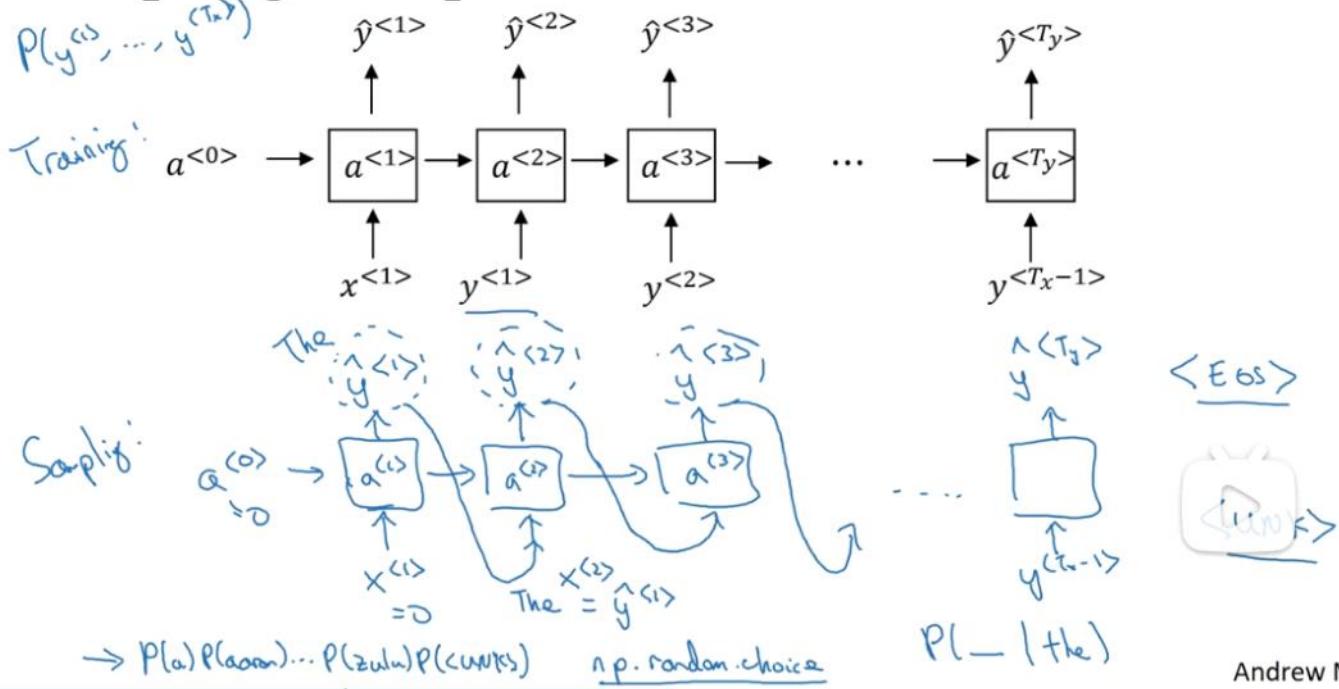


Andrew Ng

### (6) 新序列采样

2022年4月13日 12:34

## Sampling a sequence from a trained RNN

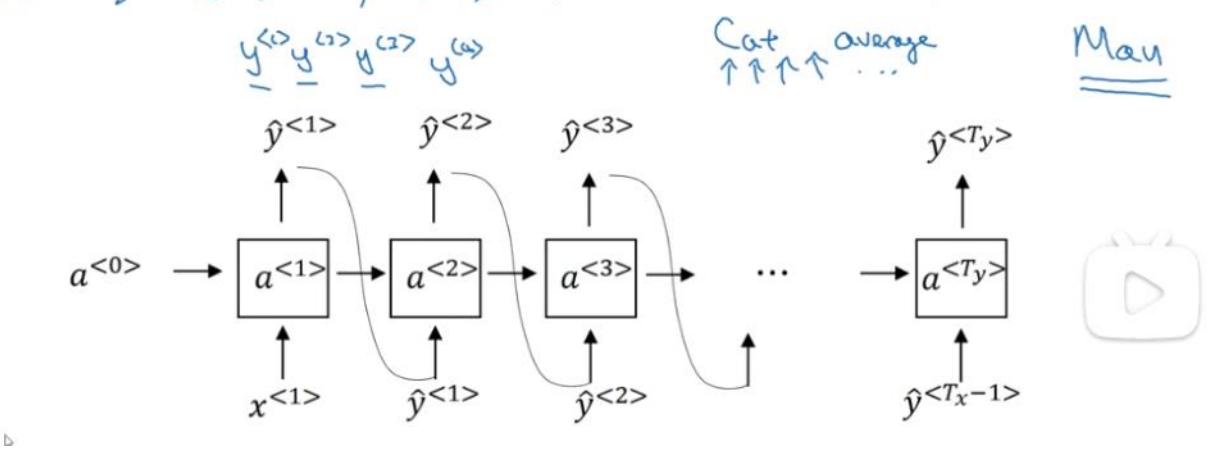


Andrew Ng

## Character-level language model

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

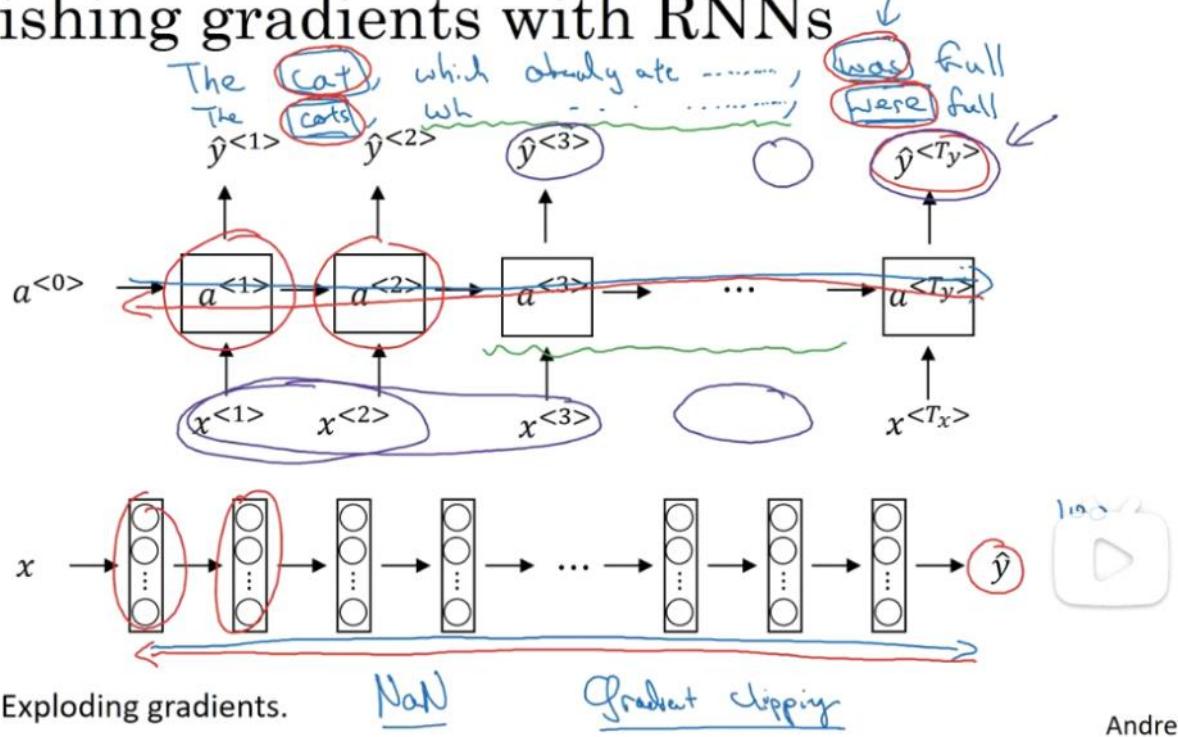
$\rightarrow \text{Vocabulary} = [a, b, c, \dots, z, \cup, \circ, \rightarrow, ;, 0, \dots, 9, A, \dots, Z]$



## (7) 带有神经网络的梯度消失

2022年4月13日 12:42

### Vanishing gradients with RNNs



我们在进行RNN传播时，是先从左向右进行传播，再从右向左进行传播，但是当层数高达一定数量时会存在从右向左进行传播，但所传递出来的值很难影响到最前面的值。

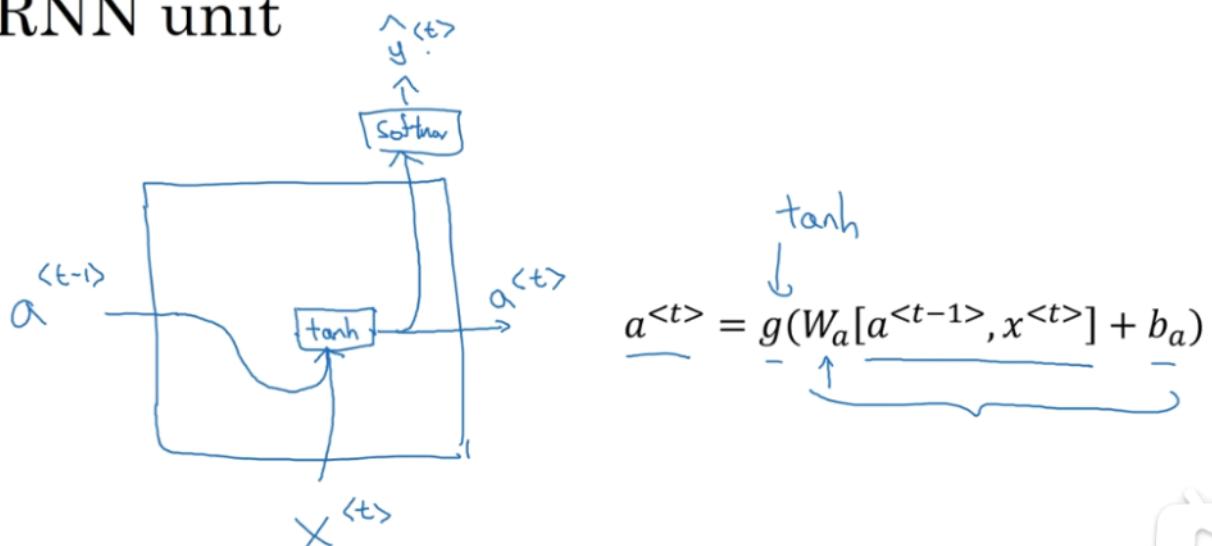
一个有效减少梯度爆炸的是进行梯度修剪，即观察梯度向量，如果它大于某个阈值，缩放梯度向量，保证它不会太大，可以通过一些最大值来进行修剪的方法

## (8)GRU单元(门控循环单元)

2022年4月13日 13:31

GRU单元改变了RNN的隐藏层，使其能够更好的捕捉深度连接。并改善了梯度消失问题。

### RNN unit



这就是RNN隐藏层的单元

the RNN unit of the hidden layer

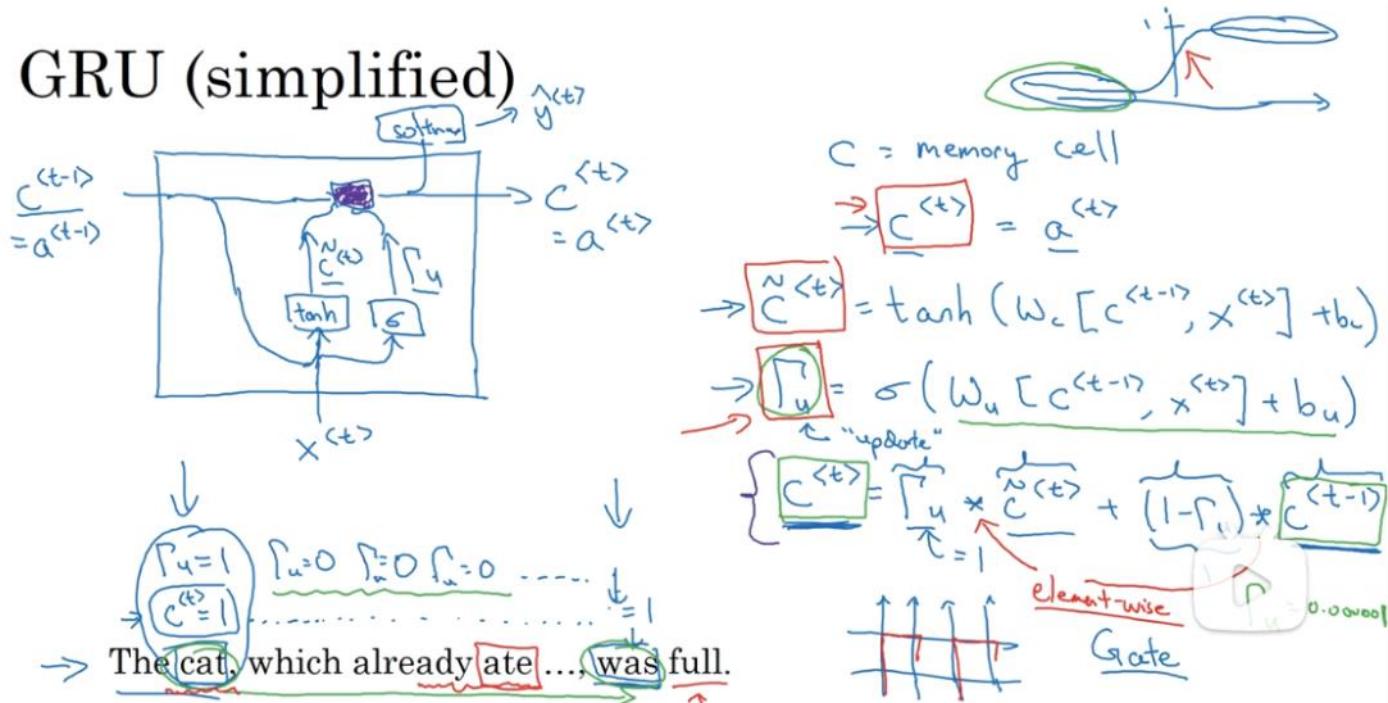
Andrew Ng

RNN隐藏层单元，我们有RNN中T处的激活单元

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

其中函数g为tanh函数。

## GRU (simplified)



[Cho et al., 2014. On the properties of neural machine translation: Encoder-decoder approaches] ↗  
[Chung et al., 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling]

Andrew Ng

## 简化后的GRU

我们需要通过主语the cat,来推断出这单词was表示是单数。

我们有公式

C=memory cell

$$c^{} = a^{}$$

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

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

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

# Full GRU

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

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

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

LSTM

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



The cat, which ate already, was full.

如果你看学术文章的话

If you look at the academic literature,

## 完整的GRU

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

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

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

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

## (9) 长短期记忆单元(LSTM)

2022年4月13日 14:20

# GRU and LSTM

## GRU

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

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

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

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

## LSTM

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

$$\underline{\Gamma_u} = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u) \quad \text{(update)}$$

$$\underline{\Gamma_f} = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f) \quad \text{(forget)}$$

$$\underline{\Gamma_o} = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o) \quad \text{(output)}$$

$$c^{<t>} = \underline{\Gamma_u} * \underline{\tilde{c}^{<t>}} + \underline{\Gamma_f} * c^{<t-1>} \\ a^{<t>} = \underline{\Gamma_o} * c^{<t>} \quad \boxed{\text{Output}}$$

[Hochreiter & Schmidhuber 1997. Long short-term memory] ↙

Andrew Ng

## RGU

$\Gamma_u$  为更新门

$\Gamma_r$  为遗弃门

## LSTM

$\Gamma_u$  为更新门

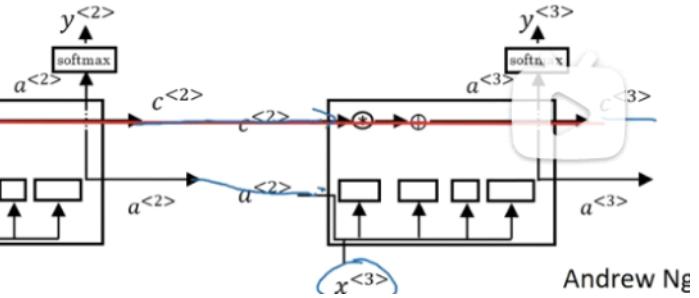
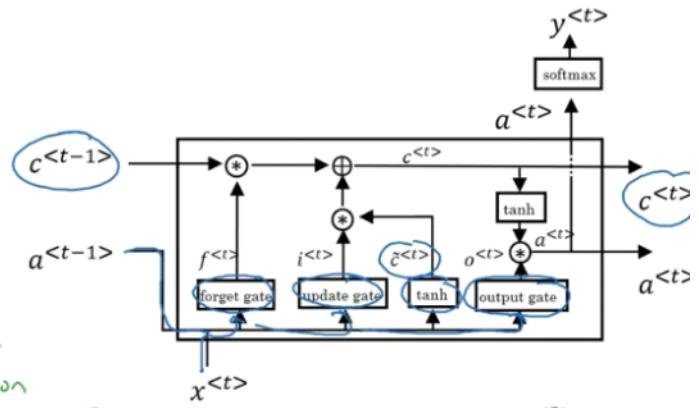
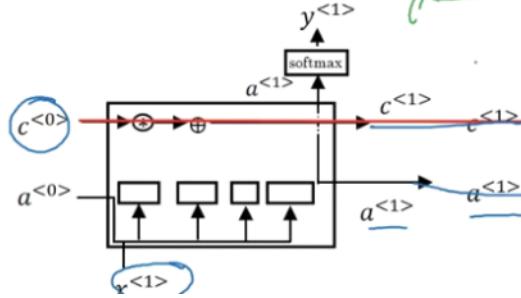
$\Gamma_r$  为遗弃门

$\Gamma_o$  为输出门

# LSTM in pictures

$$\begin{aligned}\tilde{c}^{<t>} &= \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c) \\ \Gamma_u &= \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u) \\ \Gamma_f &= \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f) \\ \Gamma_o &= \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o) \\ c^{<t>} &= \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>} \\ a^{<t>} &= \Gamma_o * \tanh c^{<t>}\end{aligned}$$

peephole connection

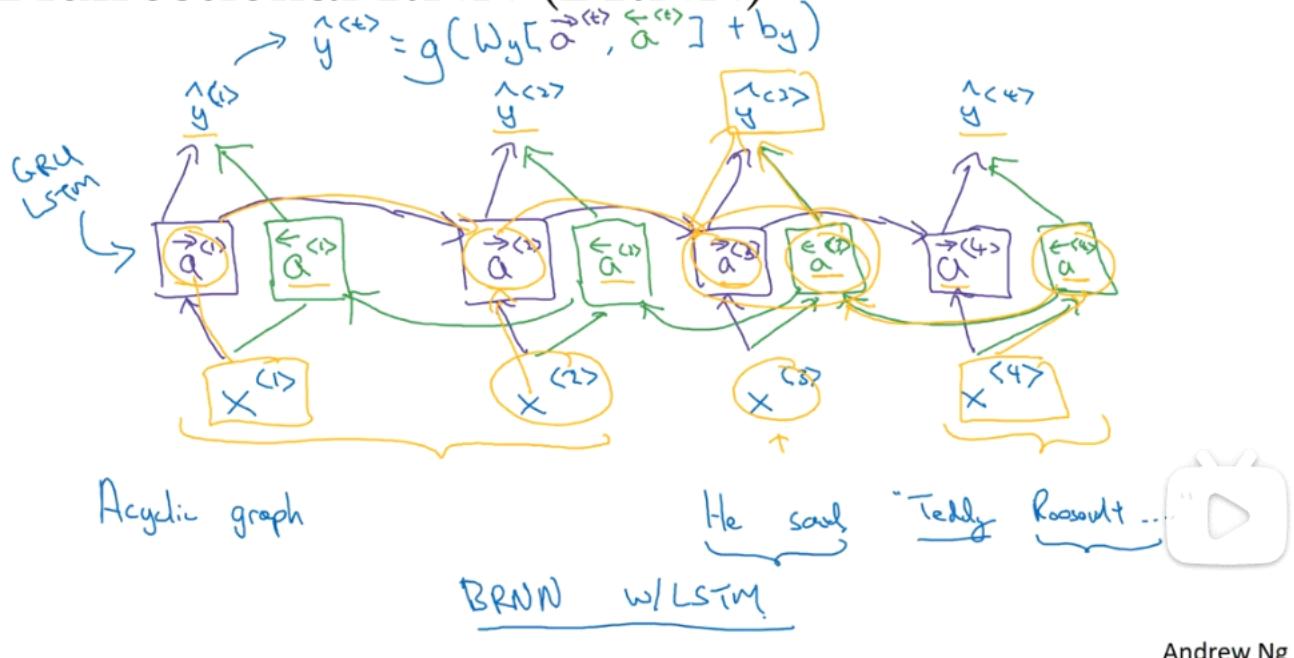


Andrew Ng

## (10) 双向神经网络

2022年4月13日 14:59

## Bidirectional RNN (BRNN)



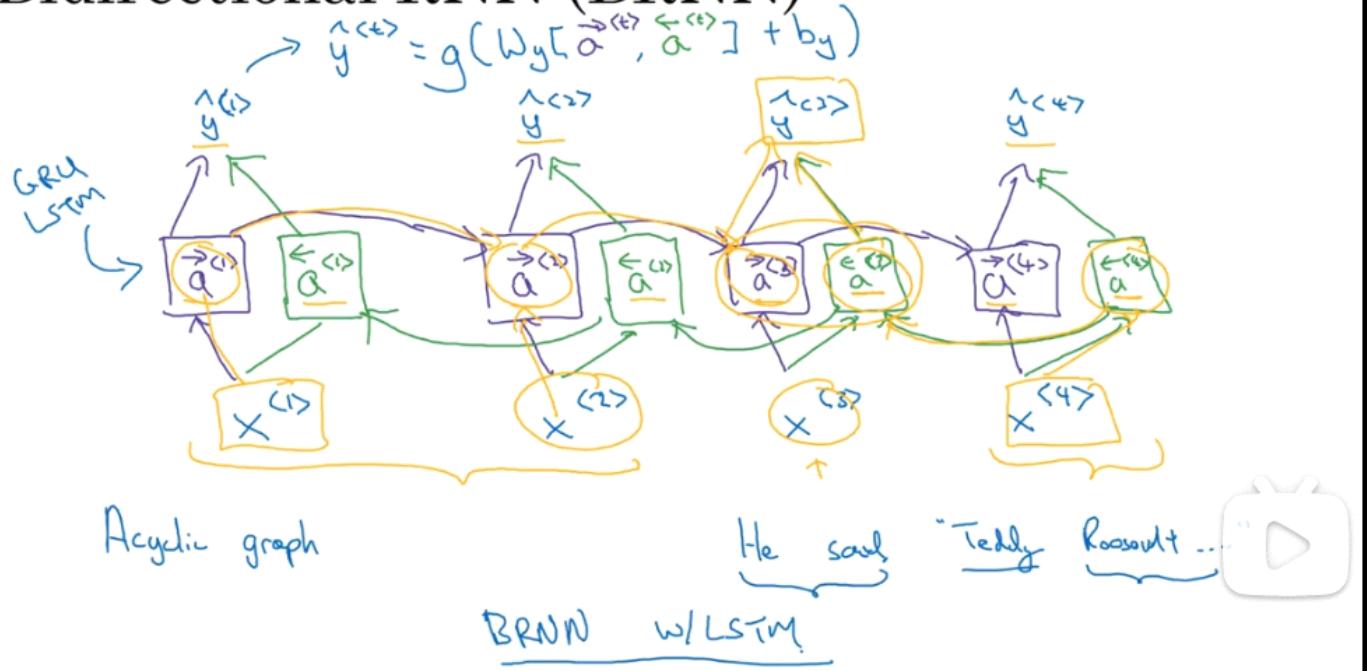
通过从正向进行传播信息再返回进行传递信息，使得对于前面的单词能够知道后面的单词，最终输出结果。

## (11) 深层循环神经网络

2022年4月13日 17:41

当要学习非常复杂的函数时，通常我们会把RNN的多个层堆叠在一起，构建更深的模型。

### Bidirectional RNN (BRNN)



Andrew Ng

## (12) 词汇表征

2022年4月13日 17:52

词嵌入，这是语言表示的一种方式，可以让算法自动的理解一些类似的词，比如男人对女人，国王与王后。

通过词嵌入的概念，我们可以构建NLP应用，即使模型标记的训练集相对较小。

## Word representation

$$V = [a, aaron, \dots, zulu, \text{UNK}]$$

$$|V| = 10,000$$

1-hot representation

Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ \rightarrow \\ 1 \\ \vdots \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$
$O_{5391}$	$O_{9853}$				

比橙子与其他词的关系更近

orange is not any closer as the relationship between

I want a glass of orange juice.

I want a glass of apple\_\_\_\_\_.



Andrew Ng

采用one-hot方法，无法类比橙子与其他词之间哪个关系更近。

即使用one-hot向量，取国王与王后的内积发现为0，无法知道两者之间的关系。

# Featurized representation: word embedding

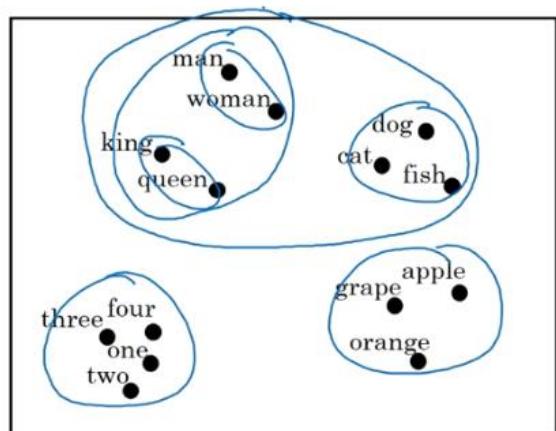
	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1		-0.95	0.97	0.00
300 Royal	0.01	0.62	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97
size						
cost						
altit.						
verb						

Andrew Ng

I want a glass of orange juice.  
I want a glass of apple juice.

通过如此进行向量之间的类比。

## Visualizing word embeddings



300 D  
↓  
2D



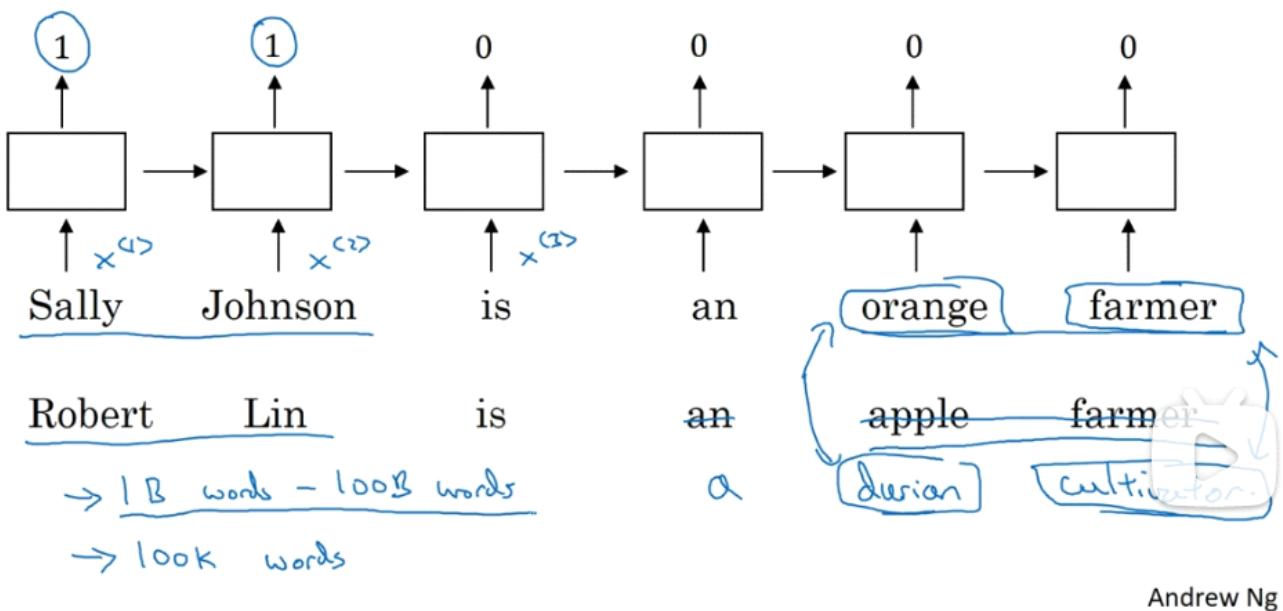
[van der Maaten and Hinton., 2008. Visualizing data using t-SNE]

Andrew Ng

## (13) 使用词嵌入

2022年4月13日 18:17

### Named entity recognition example



通过使用词嵌入使得orange farmer 与 durian cultizer 进行类比相似。

### Transfer learning and word embeddings

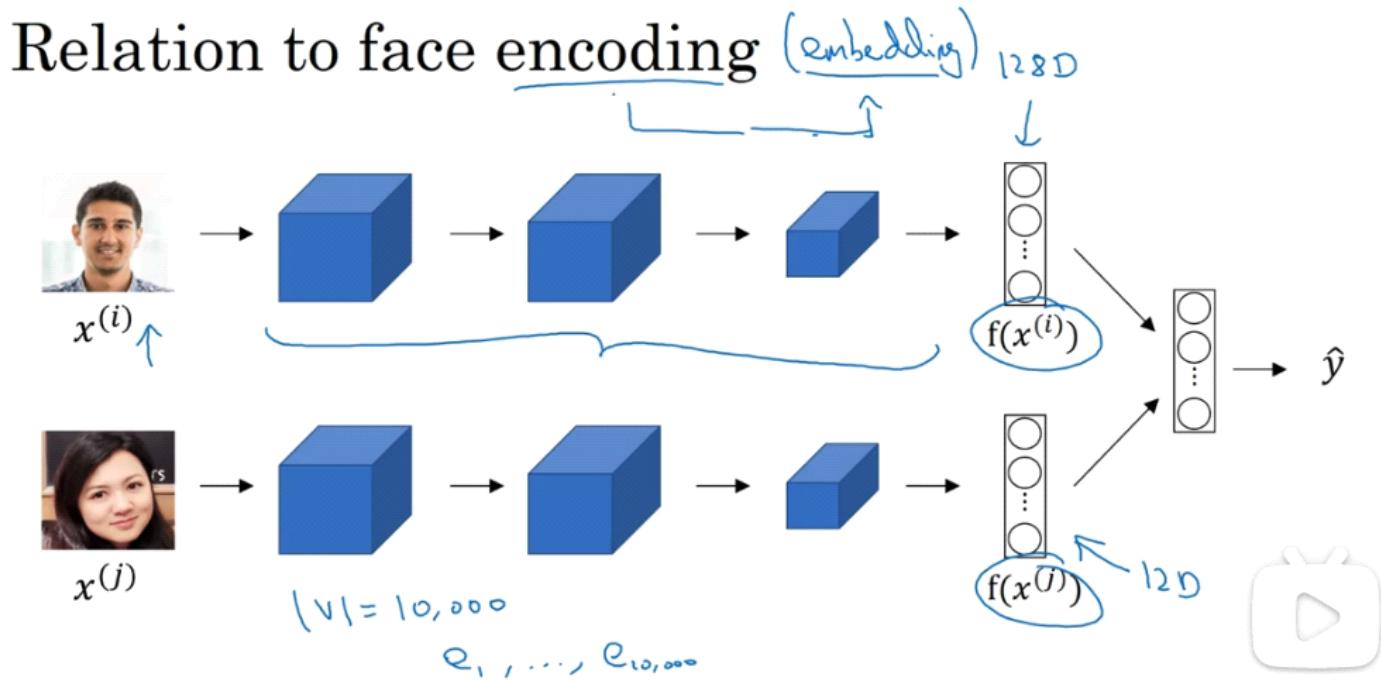
- A [ 1. Learn word embeddings from large text corpus. (1-100B words)  
(Or download pre-trained embedding online.)
- B [ 2. Transfer embedding to new task with smaller training set.  
(say, 100k words) → 10,000 → 300
3. Optional: Continue to finetune the word embeddings with new data.



Andrew Ng

## 迁移学习与词嵌入

- 一、从大量文本中学习词嵌入(或下载已经训练好的词嵌入)
- 二、迁移嵌入与一些小训练集合到新任务上。
- 三、进行微调



[Taigman et. al., 2014. DeepFace: Closing the gap to human level performance]

Andrew Ng

## (14) 词嵌入的特性

2022年4月13日 20:59

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
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	0.70	0.69	0.03	-0.02
Food	0.09	0.01	0.02	0.01	0.95	0.97

$e_{\text{man}} - e_{\text{woman}} \approx e_{\text{king}} - e_{\text{queen}}$

$e_{\text{man}} - e_{\text{woman}} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

$e_{\text{king}} - e_{\text{queen}} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

Andrew Ng

我们通过词嵌入得到的向量，我们有

$$e_{\text{man}} - e_{\text{woman}} = \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

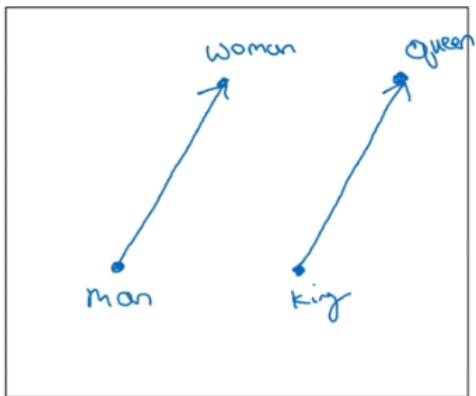
$$e_{\text{king}} - e_{\text{queen}} = \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

我们可以得到

$e_{\text{man}}$  与  $e_{\text{woman}}$  类比于  $e_{\text{king}}$  和  $e_{\text{queen}}$ ，即

$$e_{\text{man}} - e_{\text{woman}} \approx e_{\text{king}} - e_{\text{queen}}$$

# Analogies using word vectors



300 D

Find word w:  $\arg \max_w$

$$e_{man} - e_{woman} \approx e_{king} - e_w$$

$$\text{sim}(e_w, e_{king} - e_{man} + e_{woman})$$

30 - 75%.

Andrew Ng



通过上面我们可以知道，man与woman之间的距离，约等于king与queen之间的距离，所以我们有公式

$$e_{man} - e_{woman} \approx e_{king} - e_w$$

我们想要寻找w,即我们想要找到单词queen,则有

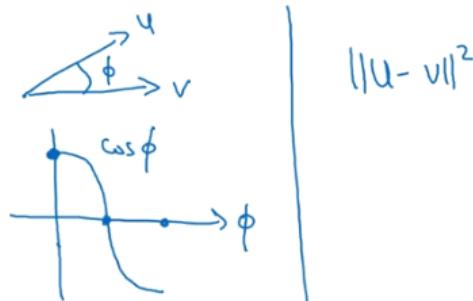
$$\arg \max_w \text{sim}(e_w, e_{king} - e_{man} + e_{woman})$$

则公式只有30-75的概率。

# Cosine similarity

$$\rightarrow \boxed{\text{sim}(e_w, e_{king} - e_{man} + e_{woman})}$$

$$\text{sim}(u, v) = \frac{u^T v}{\|u\|_2 \|v\|_2}$$



Man:Woman as Boy:Girl  
Ottawa:Canada as Nairobi:Kenya  
Big:Bigger as Tall:Taller  
Yen:Japan as Ruble:Russia



只要你再大型的文本语料库上

just by running

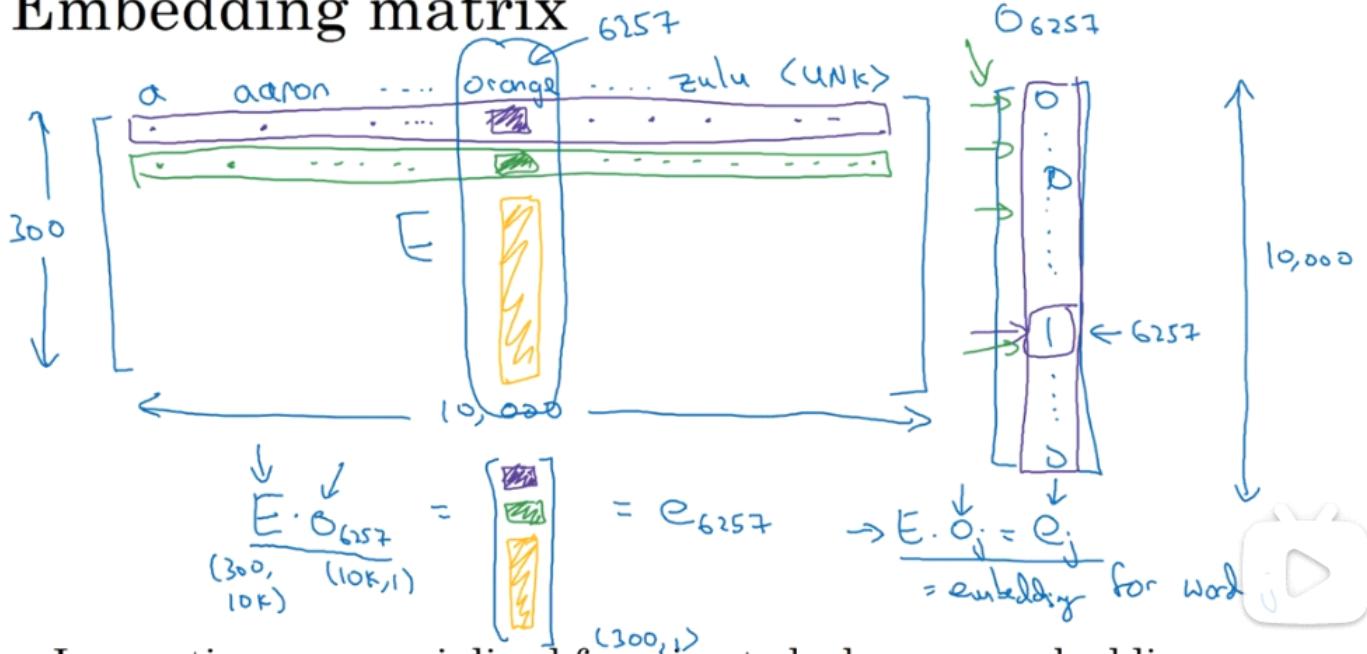
Andrew Ng

$$\text{sim}(u, v) = \frac{u^T v}{\|u\| \|v\|}$$

## (15) 嵌入矩阵

2022年4月13日 21:19

### Embedding matrix



In practice, use specialized function to look up an embedding.

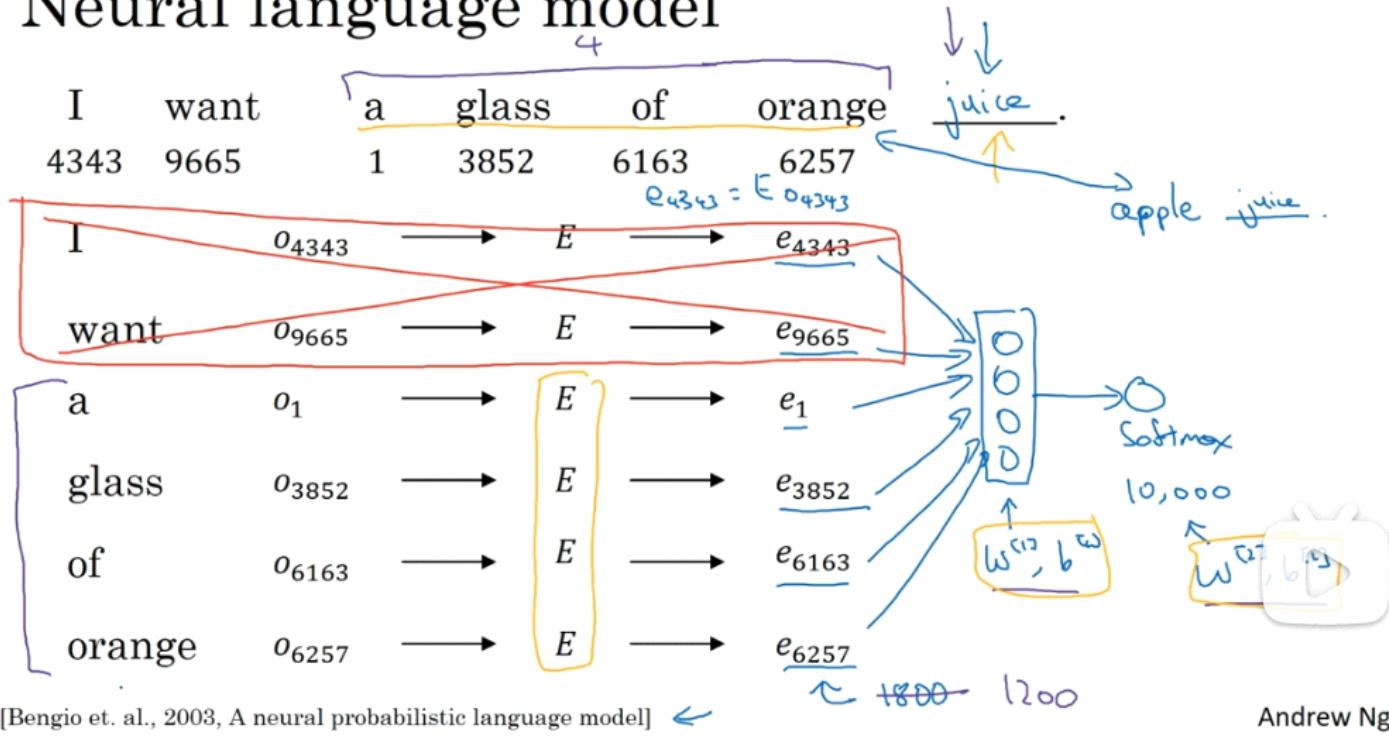
$\rightarrow$  Embedding

Andrew Ng

## (16) 学习词嵌入

2022年4月13日 22:05

### Neural language model



我们把one-hot进行矩阵E的映射成为向量e

通过一块文本的映射再通过softmax进行输出得到最终得到的预测词juice。

我们有参数矩阵E与 $w^{(1)}, b^{(1)}, w^{(2)}, b^{(2)}$

## Other context/target pairs

I want a glass of orange juice to go along with my cereal.

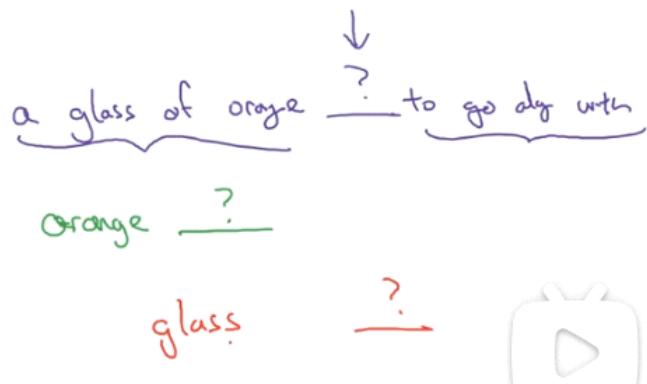
Context                      Target

Context: Last 4 words.

4 words on left & right

Last 1 word

Nearby 1 word



我们将在下节视频中把它公式化  
And we'll formalize this in the next video.

Andrew Ng



通过左右四个单词去预测中间的单词

预测后一个单词

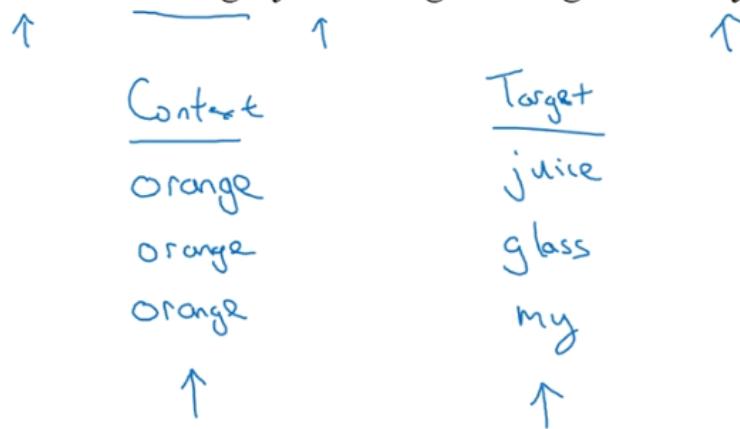
预测附近的单词

## (17) Word2Vec

2022年4月13日 22:13

### Skip-grams

I want a glass of orange juice to go along with my cereal.



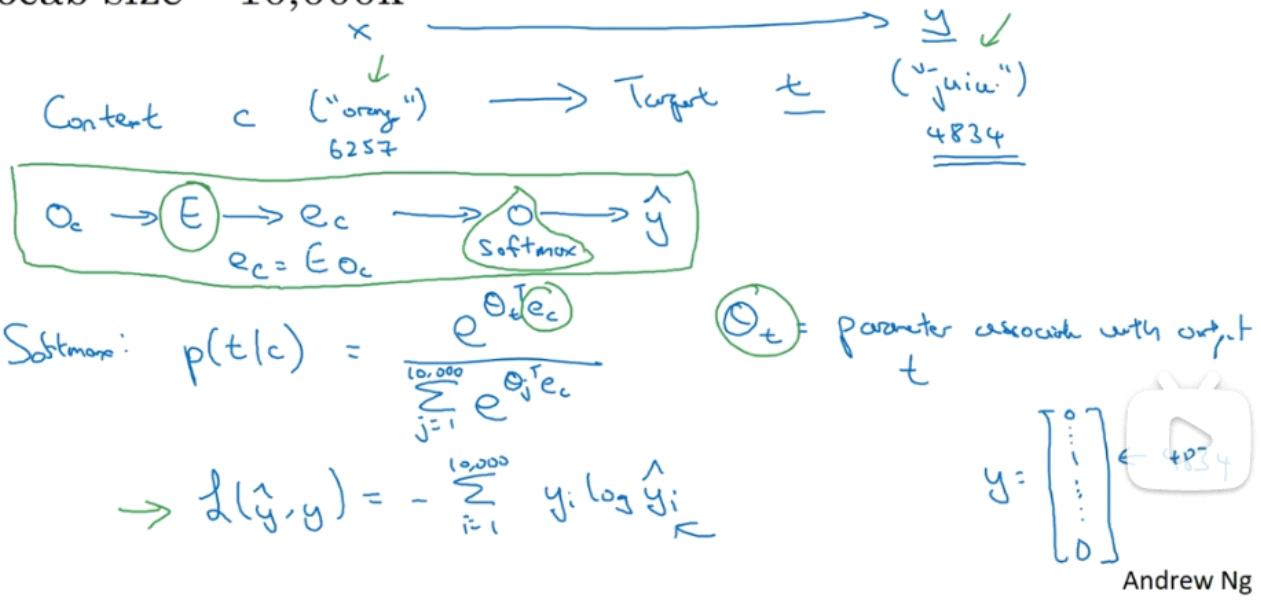
[Mikolov et. al., 2013. Efficient estimation of word representations in vector space.] ↩

Andrew Ng

预测单词间距	目标	文本
一个单词间距	orange	Juice
两个单词间距	orange	Glass
十个单词间距	Orange	My

# Model

Vocab size = 10,000k



Andrew Ng

我们通过  $C(\text{orange})$  预测目标  $T(\text{juice})$ , 通过

$$o_c \rightarrow E \rightarrow e_c \rightarrow \text{softmax} \rightarrow \hat{y}$$

Softmax:

$$p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$$

$\theta_t$  是与输出  $t$  有关的参数, 即某个词  $t$  和标签相符的概率是多少

$$\mathcal{L}(\hat{y}, y) = \sum_{j=1}^{10,000} y_j \log \hat{y}_j$$

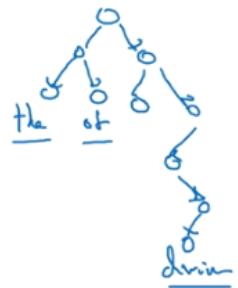
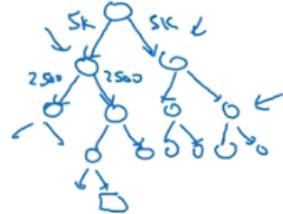
$$y = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix} = 4834$$

# Problems with softmax classification

$$p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$$

Hierarchical softmax.

$\log |V|$



How to sample the context  $c$ ?

→ the, of, a, and, to, ...

→ orange, apple, durian

$Q_{\text{durian}}$

$t$   
 $c \rightarrow t$

$P(c)$



Andrew Ng

怎样去采样文本c

如我们通过文本C去得到预测T

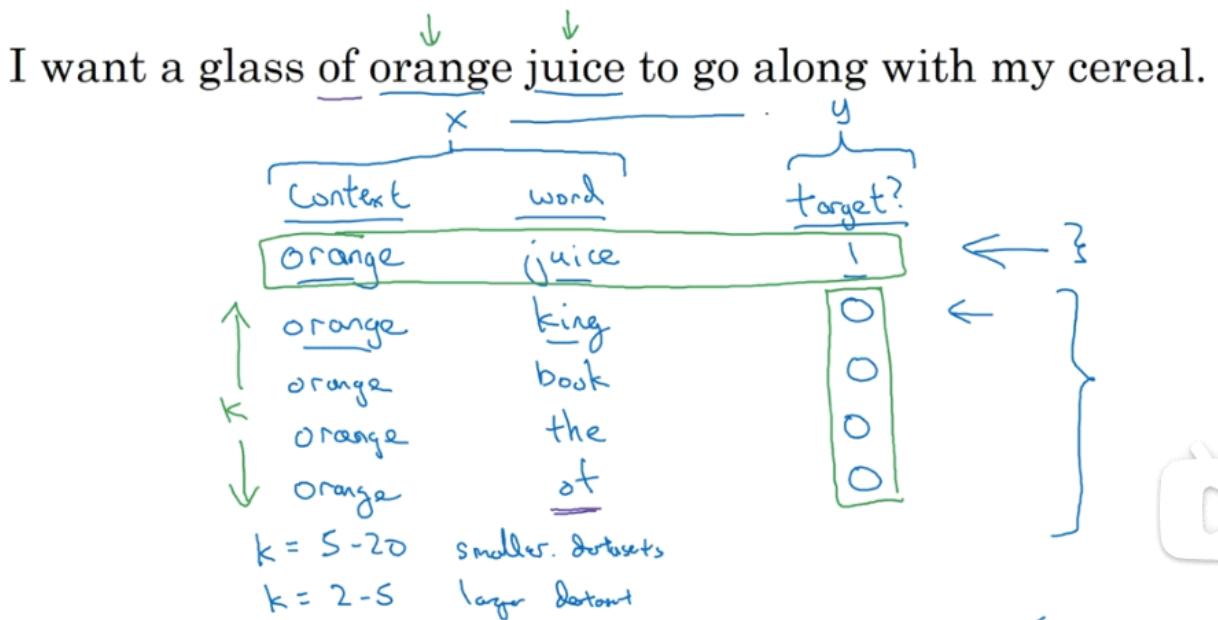
一种选择是可以对语料库均匀且随机地进行采样，会有一些词比较常见如the,of,a诸如此类的表现的比较频繁。这么做的话会发现，从上下文到目标词的映射会较为频繁地得到这些种类的词，但是其它词像orange apple或durian就不会那么频繁地出现，我们不会想要我们的训练集都是这些出现的比较频繁的词，因为这会导致我们花大部分的力气来更新这些频繁出现的单词e\_c，但是我们其实是想要大部分更新durian这些更少出现的词嵌入。

## (18) 负样本

2022年4月13日 22:43

上文为skip-gram模型如何帮助我们构造一个监督学习任务把上下文映射到了目标词上，如何让它学习到一个实用的词嵌入，但是它缺点在于softmax计算起来得很慢。在本文中，我们会看到一个改善过的学习问题叫做负采样，负采样与skip-gram模型相似的事情，但是用了一个更加有效的学习算法。

## Defining a new learning problem



[Mikolov et. al., 2013. Distributed representation of words and phrases and their compositionality]

Andrew Ng

我们有文本

I want a glass of orange juice to go along with my cereal.

我们有上下文-目标词

Context	Word	Target
Orange	Juice	1

这为正样本，我们有

Orange	King	0
--------	------	---

这是负样本，负样本的数量用K表示

通常在小数据集中K=5-20

## 在大数据集中 $K=2-5$

Model

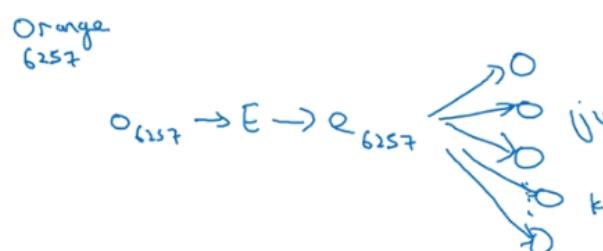
The diagram shows the softmax function being applied to a context vector  $e_c$  to produce word probabilities. The softmax output is then used as input to a logistic regression model to predict a target word  $y$ .

**Softmax:**  $p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$

**Predicting  $y=1$ :**  $P(y=1 | c, t) = \sigma(\theta_t^T e_c)$

**Table of Data:**

context	word	target?
orange	juice	1
orange	king	0
orange	book	0
orange	the	0
orange	of	0



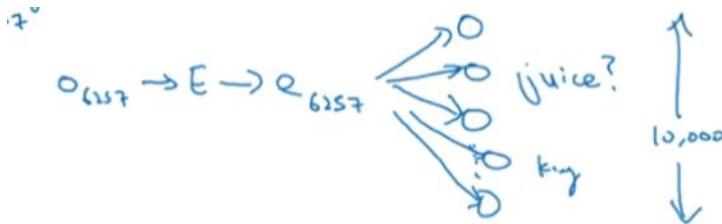
10,000 binary classification problem

## 我们有公式

$$p(y = 1 | c, t) = \sigma(\theta_t^T e_c)$$

其中 $\sigma$ 为logit回归

我们有流程



最后有10000个二分类回归，因为如此训练太耗时了，所以我选取K个来进行训练，而不是全部训练。

# Selecting negative examples

context	word	target?
orange	juice	1
orange	king	0
orange	book	0
orange	the	0
orange	of	0

$$P(\omega_i) = \frac{f(\omega_i)^{3/4}}{\sum_{j=1}^{10,000} f(\omega_j)^{3/4}}$$
$$\frac{1}{|V|}$$



有理论证明的

very theoretically justified,

Andrew Ng

## (19) Glove词向量

2022年4月14日 10:52

# GloVe (global vectors for word representation)

I want a glass of orange juice to go along with my cereal.

c, t

$$x_{ij} = \# \text{times } i \text{ appears in context of } j.$$

$\begin{matrix} \uparrow & \uparrow \\ c & t \end{matrix}$        $\begin{matrix} \downarrow & \downarrow \\ i & t \end{matrix}$        $\begin{matrix} \uparrow & \uparrow \\ j & c \end{matrix}$

$$x_{ij} = x_{ji} \leftarrow$$



[Pennington et. al., 2014. GloVe: Global vectors for word representation]

Andrew Ng

GLove的做法是使关系开始明确化，假定 $x_{ij}$ 是单词*i*在单词*j*上下文出现的次数，这里的*ij*和 $tc$ 功能一样。

## Model

$$\text{minimize} \sum_{i=1}^{10,000} \sum_{j=1}^{10,000} f(x_{ij}) (\Theta_i^T e_j + b_i + b_j^T - \log \frac{x_{ij}}{\Theta_i^T e_c})^2$$

Annotations:

- $f(x_{ij}) = 0$  or  $x_{ij} = 0$ . "0 log 0" = 0
- weight term:  $\Theta_i^T e_c$
- this, is, at, a, ... duration
- $\Theta_i, e_i$  are symmetric
- $e_w^{(\text{final})} = \frac{e_w + \Theta_w}{2}$



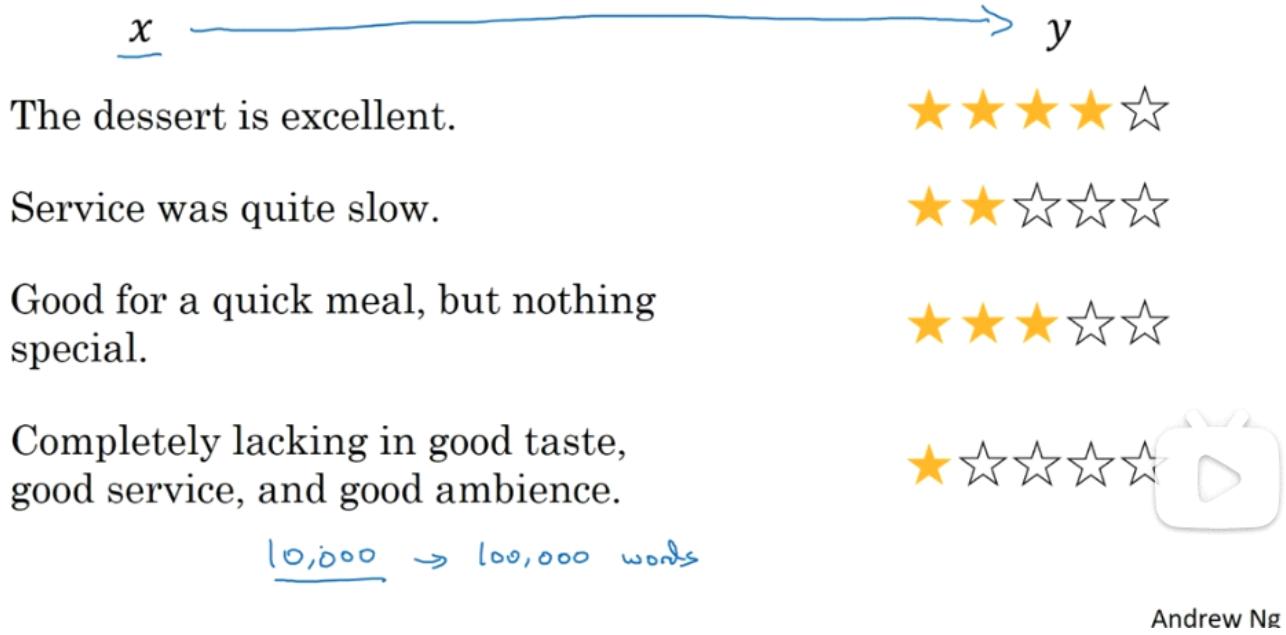
当 $x_{ij} = 0$ ,  $f(x_{ij}) = 0$

Andrew Ng

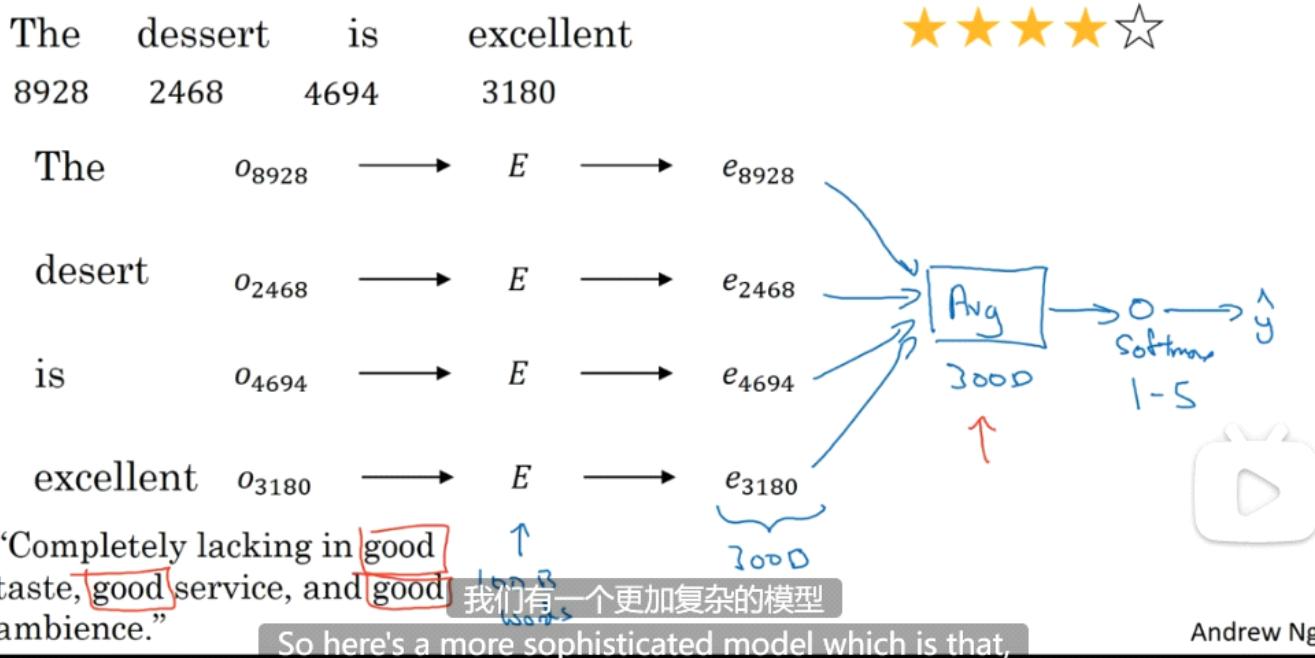
## (20) 情绪分类

2022年4月14日 11:13

# Sentiment classification problem



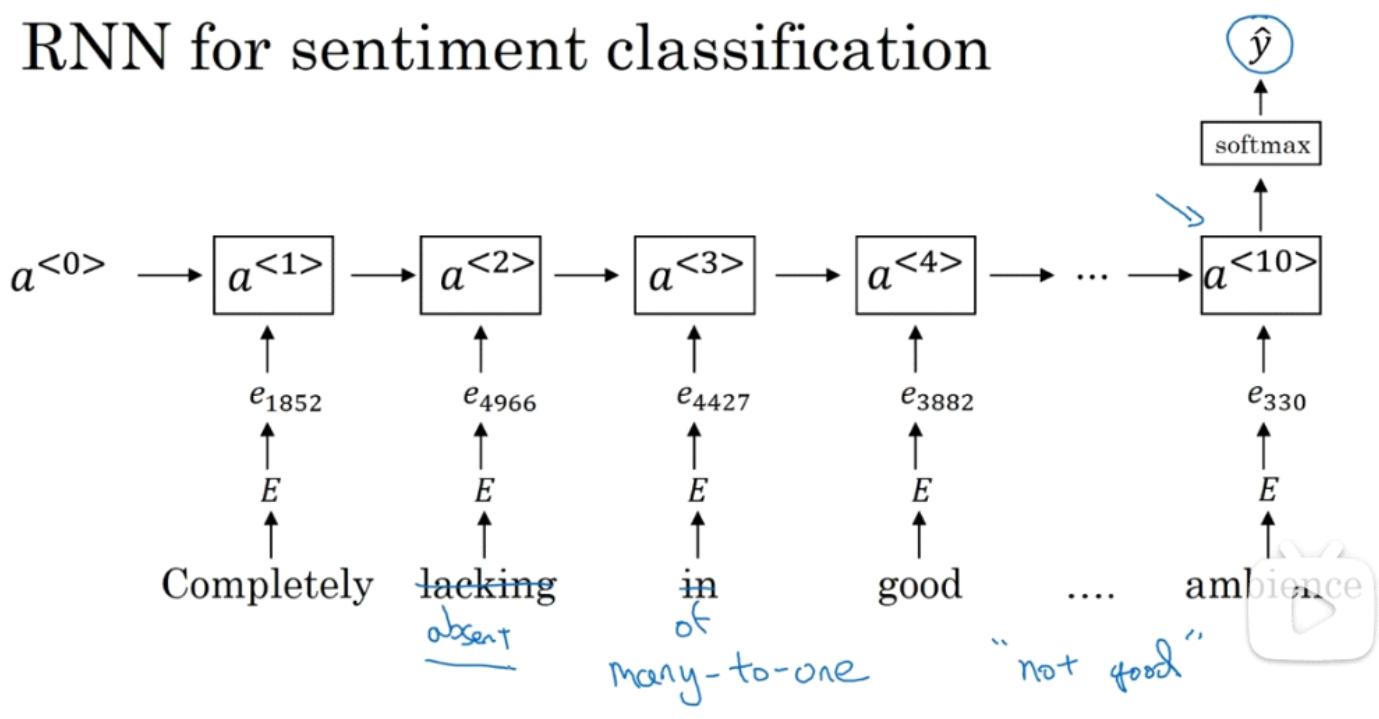
# Simple sentiment classification model



我们有一个简单的情感文本分类模型，有文本  
Completely lacking in good taste, good service, and good ambience, 这个文  
本在模型中不会统计语序，出现了三次的good, 模型可能会以为这一个

好的评价，其实这是一个坏的评价。

## RNN for sentiment classification



Andrew Ng

采用此模型会有一个较好的文本情感文本，不会忽略语序。

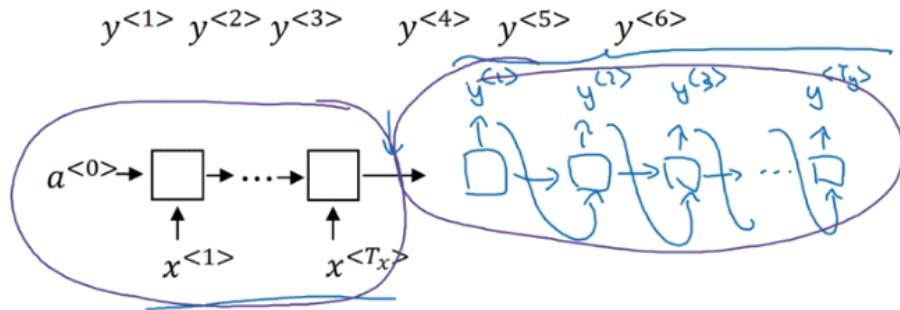
# (1) 基础模型

2022年4月14日 12:15

## Sequence to sequence model

$x^{<1>} \quad x^{<2>} \quad x^{<3>} \quad x^{<4>} \quad x^{<5>}$   
Jane visite l'Afrique en septembre

→ Jane is visiting Africa in September.



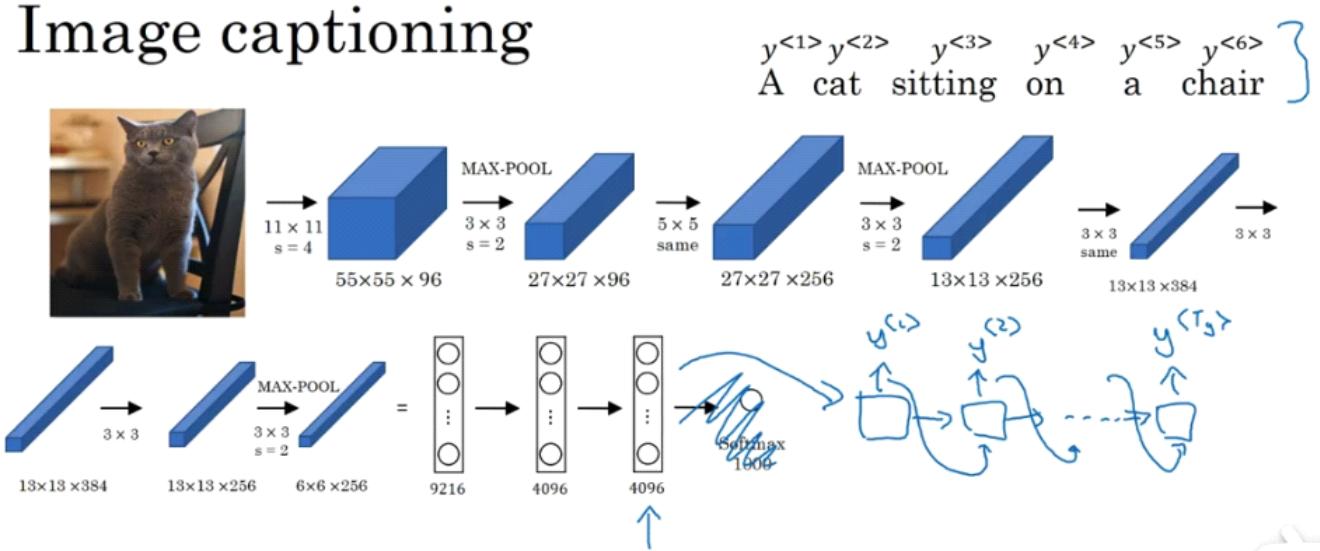
[Sutskever et al., 2014. Sequence to sequence learning with neural networks] [英语翻译](#)

[Cho et al., 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation] [English translation](#) ↩

Andrew Ng

从序列到序列模型，图中采用编码器与解码器对机器进行翻译。

## Image captioning



[Mao et. al., 2014. Deep captioning with multimodal recurrent neural networks] ↩

[Vinyals et. al., 2014. Show and tell: Neural image caption generator] ↩

[Karpathy and Li, 2015. Deep visual-semantic alignments for generating image descriptions] ↩

Andrew Ng

把图片通过卷积神经网络进行训练再把他输入进RNN网络中得到一个对

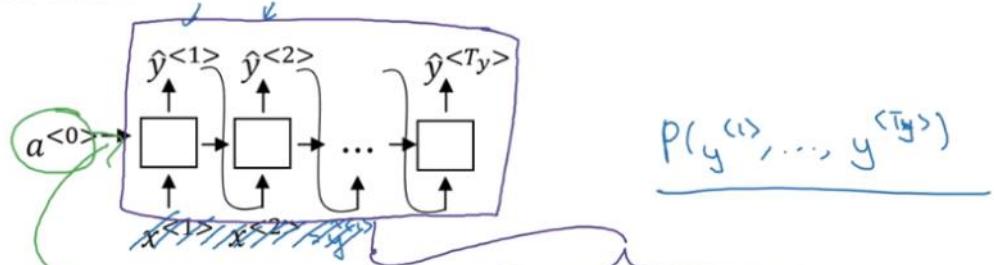
于图片文字的描述。

## (2) 选择最有可能的句子

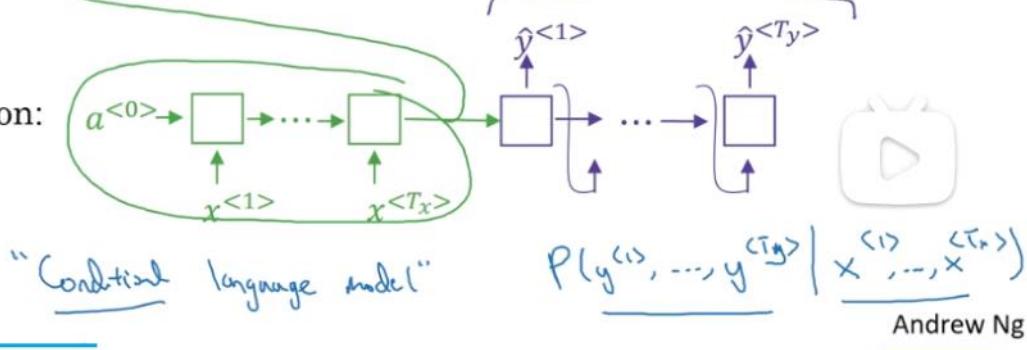
2022年4月14日 12:25

# Machine translation as building a conditional language model

Language model:



Machine translation:



通过语言模型得到句子，再通过机器翻译模型翻译出句子。

## Finding the most likely translation

Jane visite l'Afrique en septembre.

$$P(y^{<1>} \dots, y^{<T_y>} | x)$$

- Jane is visiting Africa in September.
- Jane is going to be visiting Africa in September.
- In September, Jane will visit Africa.
- Her African friend welcomed Jane in September.

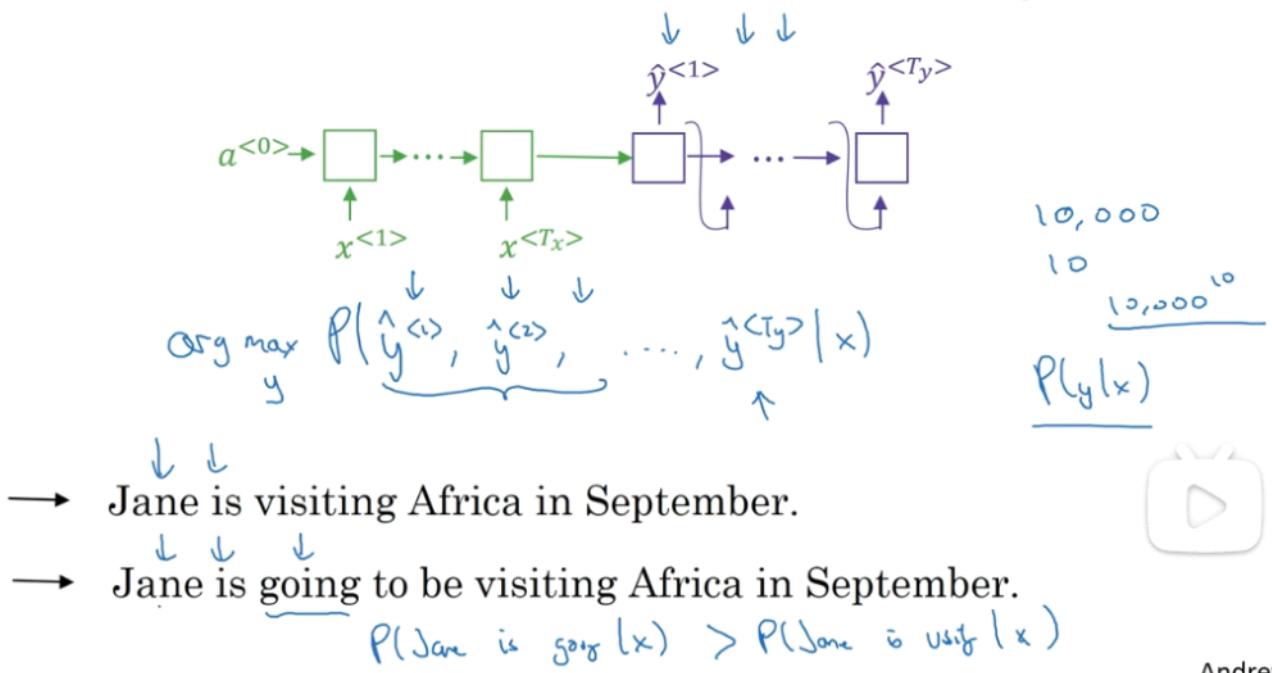


$$\arg \max_{y^{<1>} \dots, y^{<T_y>}} P(y^{<1>} \dots, y^{<T_y>} | x)$$

Andrew Ng

通过给入francX输出最大可能的英语句子。

## Why not a greedy search?



Andrew Ng



为什么不使用贪心算法？

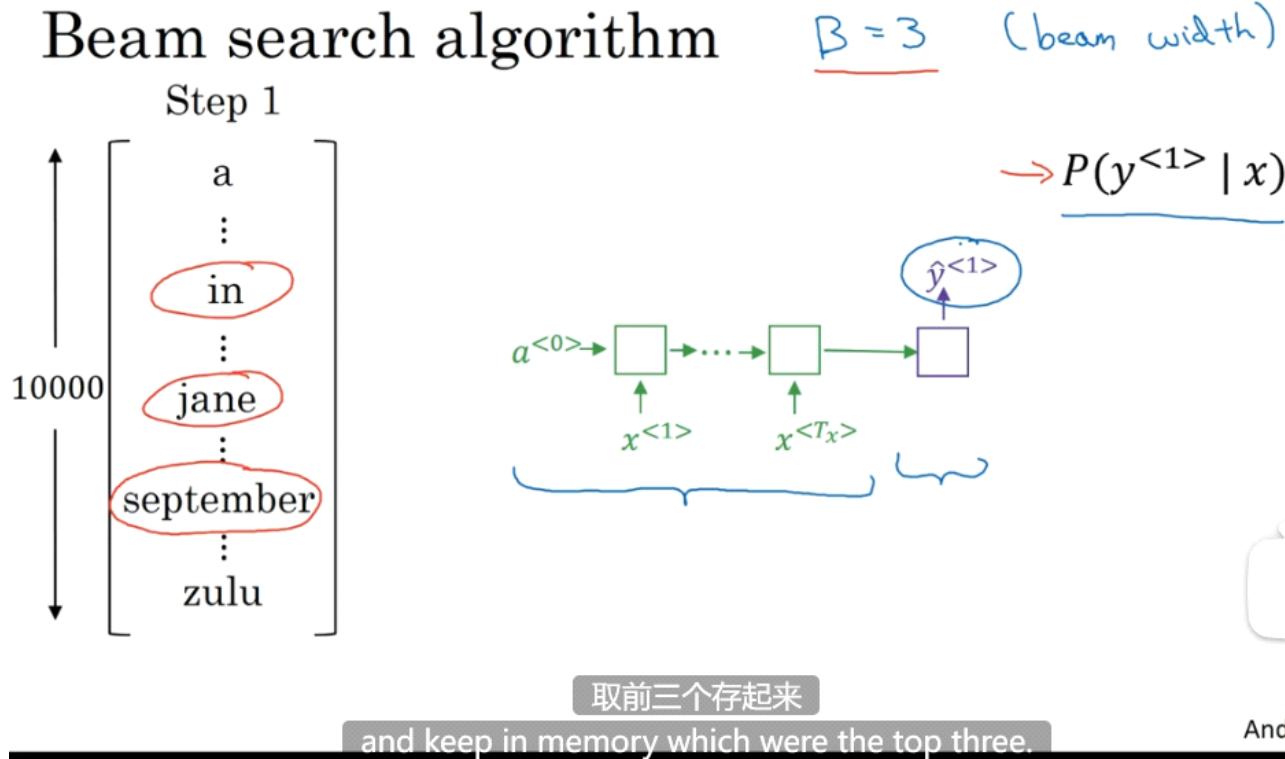
这里的贪心算法与计算机科学中的贪心算法一样，他为先中第一个最可能的词，接着选中第二个最可能词，一直往下。

- Jane is visiting Africa in September.  
→ Jane is going to be visiting Africa in September.
- $P(\text{Jane is going } | x) > P(\text{Jane is visit } | x)$

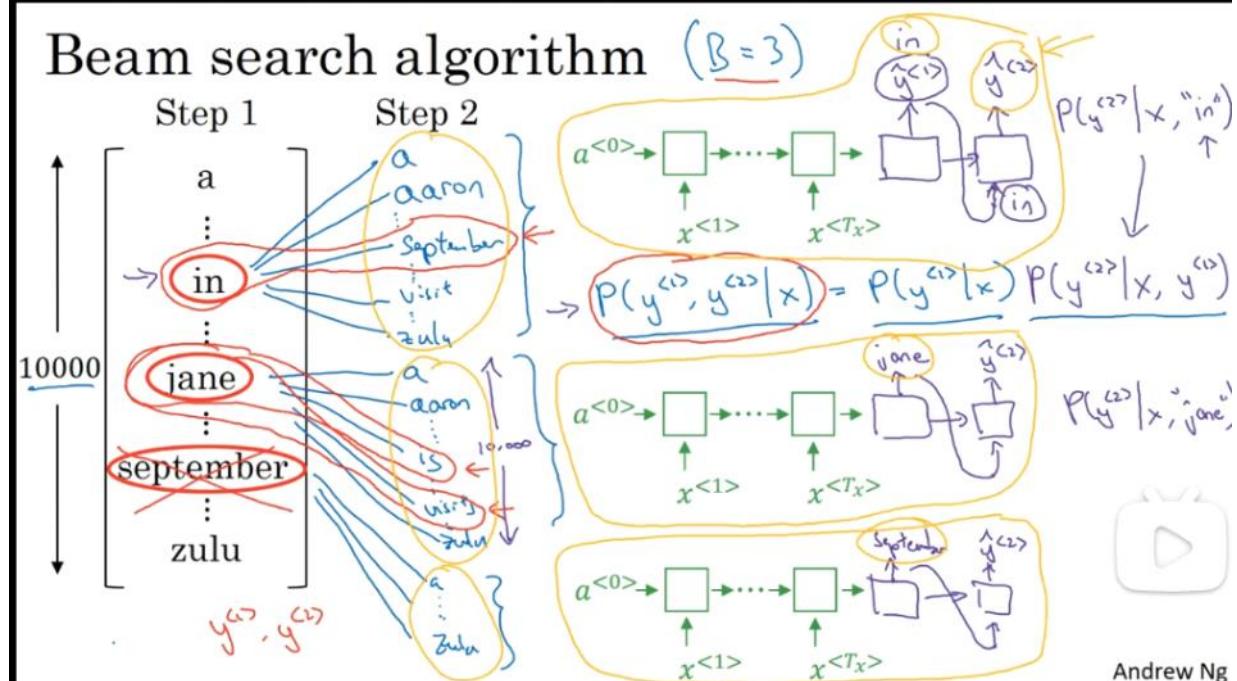
这是因为我们有上面两个翻译，第一个为选一整个句子最大概率的翻译，第二个为使用贪心算法翻译出来的句子，第一个翻译得更加准确。

### (3) 定向搜索

2022年4月14日 12:50



第一步为当B=3里，我们取前三个最有可能的单词，并把他们进行存放。



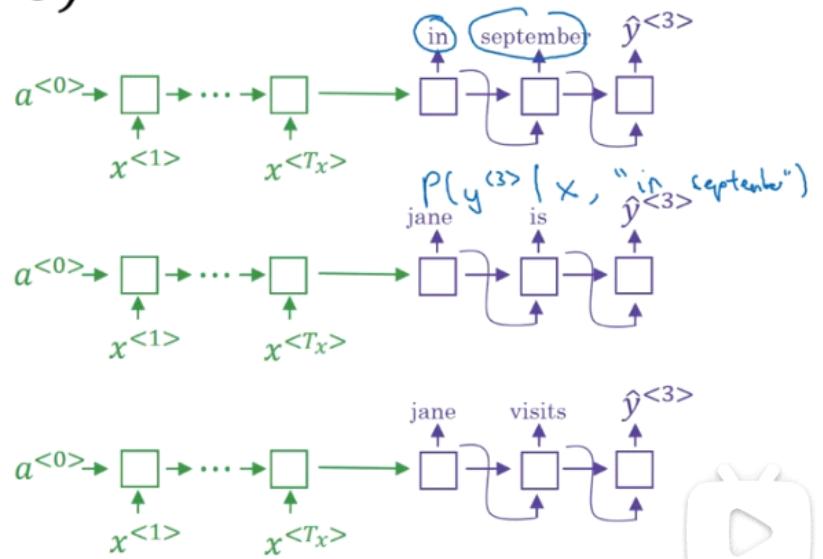
第二步，从In的后面可以跟的单词的映射表中选取三个最有可能的单词，并进行存放。

# Beam search ( $B = 3$ )

in september  
 in      aaron  
 in      jane  
 in      zulu

jane is  
 jane      visits  
 jane      zulu

jane visits  
 jane      africa  
 jane      zulu



$$P(y^{<1>} , y^{<2>} | x)$$

jane visits africa in september. <EOS>

Andrew Ng

如此往复运行定向搜索。

## (4) 改进定向搜索

2022年4月14日 15:45

### Length normalization

$$\arg \max_y \prod_{t=1}^{T_y} P(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>})$$

$P(y^{<1>} \dots y^{<T_y>} | x) = \frac{P(y^{<1>} | x)}{P(y^{<2>} | x, y^{<1>}) \dots P(y^{<T_y>} | x, y^{<1>}, \dots, y^{<T_y-1>})}$

$\log P(y|x) \leftarrow$   
 $P(y|x) \leftarrow$

$$\arg \max_y \sum_{t=1}^{T_y} \log P(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>}) \leftarrow$$

$T_y = 1, 2, 3, \dots, 30.$

$\alpha = 0.7$   
 $\alpha = 1$   
 $\alpha = 0$

$\rightarrow \frac{1}{T_y^\alpha} \sum_{t=1}^{T_y} \log P(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>})$

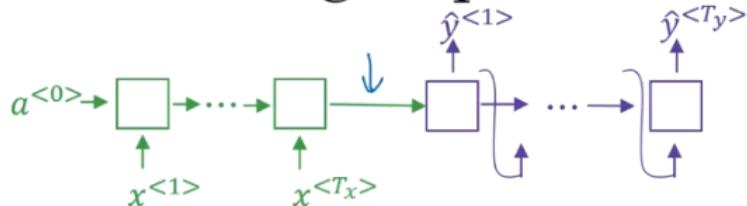
Andrew Ng

使用 $\log$ 函数使得式子更加稳定

## (5) 注意力模型直观理解

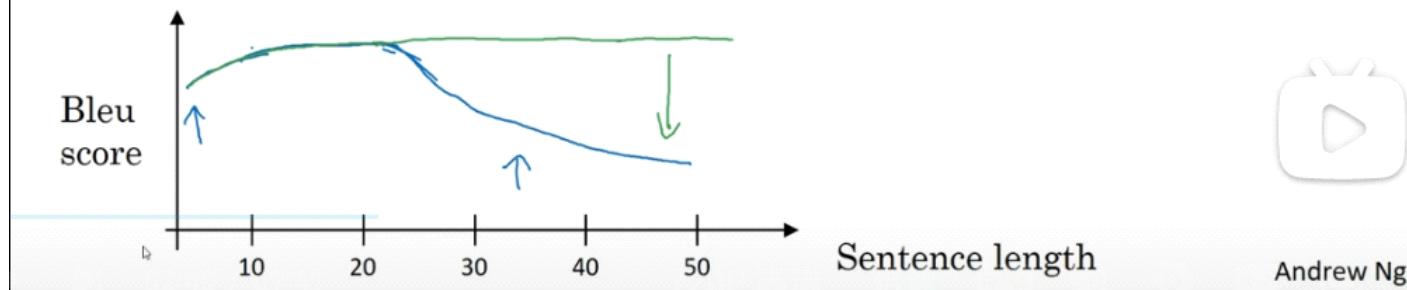
2022年4月14日 15:51

### The problem of long sequences



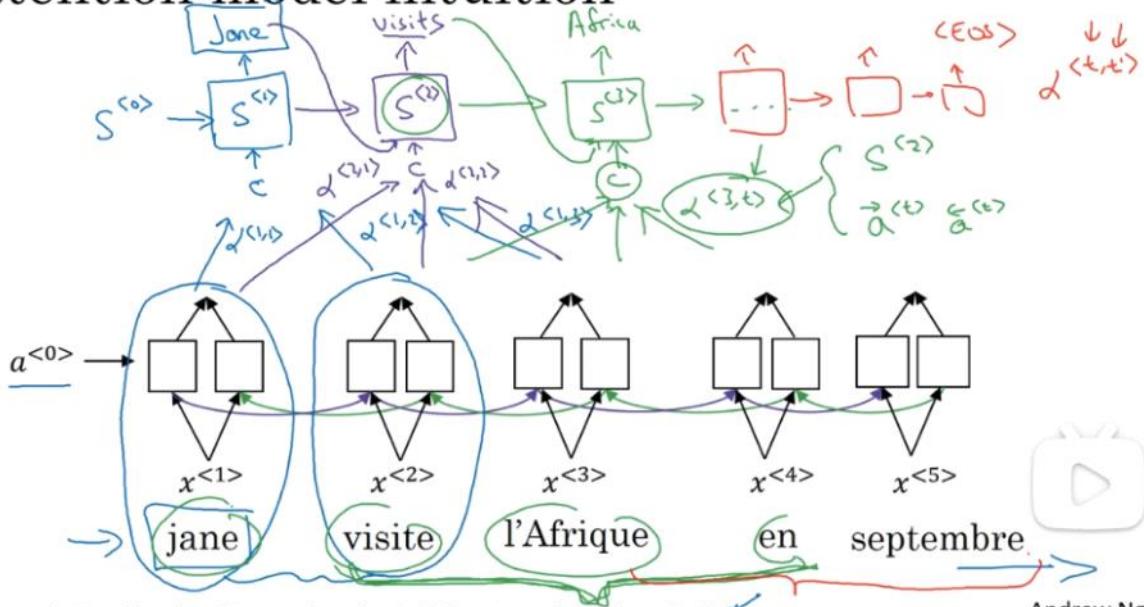
Jane s'est rendue en Afrique en septembre dernier, a apprécié la culture et a rencontré beaucoup de gens merveilleux; elle est revenue en parlant comment son voyage était merveilleux, et elle me tente d'y aller aussi.

Jane went to Africa last September, and enjoyed the culture and met many wonderful people; she came back raving about how wonderful her trip was, and is tempting me to go too.



机器翻译会先阅读各个句子，再逐一进行翻译，而人工翻译是一部分一部分的进行翻译，使用编码解码器模型在10-20词之内的句子会表现良好，若超过这个句子则有所下降。

### Attention model intuition



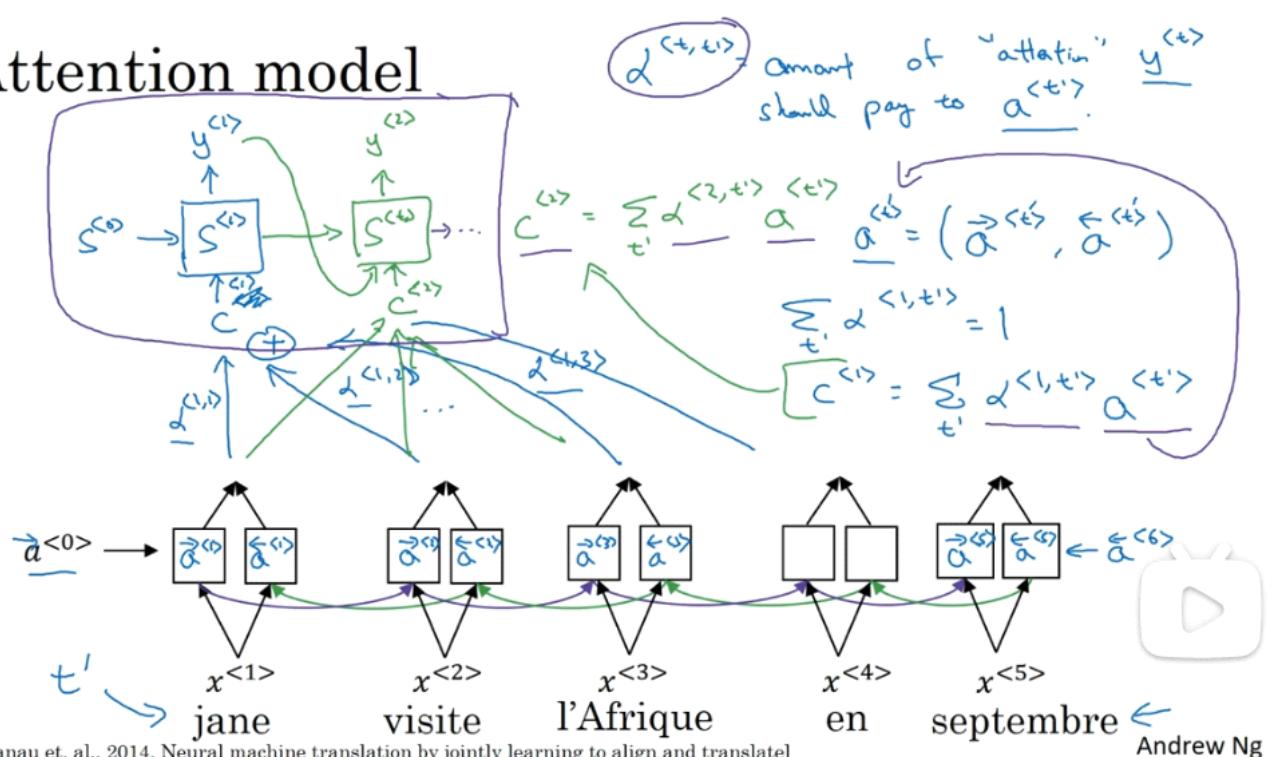
该模型为每一个词的生成都有一个注意力权重，表明每一步的应该放多

和注意力。

## (6) 注意力模型

2022年4月14日 16:03

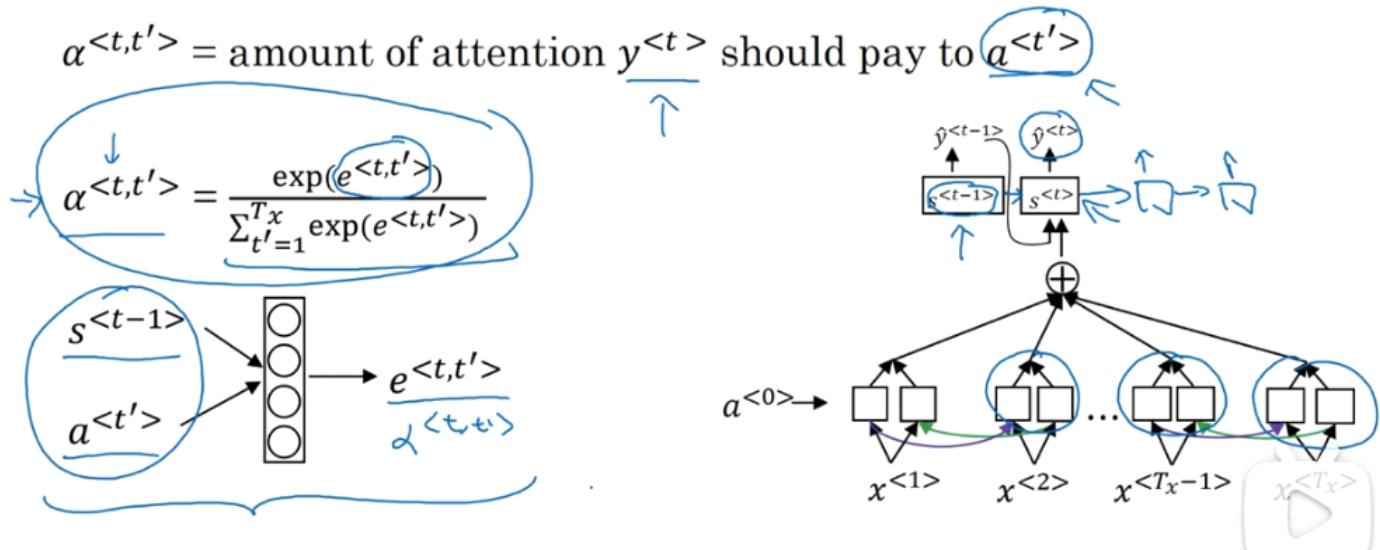
### Attention model



[Bahdanau et. al., 2014. Neural machine translation by jointly learning to align and translate]

Andrew Ng

### Computing attention $\alpha^{<t,t'>}$



[Bahdanau et. al., 2014. Neural machine translation by jointly learning to align and translate]  
[Xu et. al., 2015. Show, attend and tell: Neural image caption generation with visual attention]

Andrew Ng