Deeplearning.ai 学习笔记

陆恒 luhengok@163.com

2018 年 4 月 16 日

1 神经网络参数调试

1.1 参数初始化

介绍几种参数初始化的方法，来应对梯度爆炸或消失的问题
Xavier initialization 泽维尔 Xavier Glorot ,Yoshua Bengio.2010
Relu, Gaussion distribution: $\sigma = \sqrt{2}\sqrt{\frac{2}{n_{inputs}+n_{outputs}}}$

He initialization Kaiming He et al.2015 何恺明
Relu, Gaussion distribution: $\sigma = \sqrt{2}\sqrt{\frac{1}{n_{inputs}}}$

1.2 最优化算法

以下给出了几个最优化算法来应对传统的批梯度下降的缺点
一，指数加权平均
$v_t = \beta v_{t-1} + (1-\beta)\theta_t$，其中 $v_{t-1} \approx \frac{1}{1-\beta}$ 个之前的数据的指数平均
指数加权平均为什么叫” 指数”

$$v_{100} = 0.9v_{99} + 0.1\theta_{100}$$
$$v_{99} = 0.9v_{98} + 0.1\theta_{99}$$
$$v_{98} = 0.9v_{97} + 0.1\theta_{98}$$
$$v_{97} = 0.9v_{96} + 0.1\theta_{97}$$
.....
$$v_1 = 0.9v_0 + 0.1\theta_0$$
$$v_0 = 0$$
$$v_{100} = 0.9(0.9v_{98} + 0.1\theta_{99}) + 0.1\theta_{100}$$
.....
$$\beta^{\frac{1}{1-\beta}} \approx \frac{1}{e}$$
$$\lim_{\epsilon \to 0} 1 - \epsilon^{\frac{1}{\epsilon}} = \frac{1}{e}$$

二，偏差修正
为什么我们需要偏差修正?
$v_1 = 0.9v_0 + 0.1\theta_0$
$v_0 = 0$ 那么最开始的指数加权平均会有偏差
修正前：$v_t = \beta v_{t-1} + (1-\beta)\theta_t$
修正后：$v_t = \beta v_{t-1} + (1-\beta)\theta_t$，$v_t := \frac{v_t}{1-\beta^t}$
一般来讲，这个修正不会用，原因是在现实情况下，不会关心最开始的几个数据

三，momentun

$$
\begin{aligned}
v_{dw} &= \beta v_{dw} + (1-\beta)dw \\
v_{db} &= \beta v_{db} + (1-\beta)db \\
w &:= w - \alpha v_{dw} \\
b &:= b - \alpha v_{db}
\end{aligned}
$$

做个假设，假设 w 是最优化方向的梯度群，b 是非最优化方向的梯度群 (震荡)，那么指数加权平均可以通过平均来抵消震荡，并在最优化方向加速。$\beta = 0.9$

四，RMSprop
Root mean squared

$$
\begin{aligned}
s_{dw} &= \beta s_{dw} + (1-\beta)dw^2 \\
s_{db} &= \beta s_{db} + (1-\beta)db^2 \\
w &:= w - \alpha \frac{dw}{\sqrt{s_{dw}}} \\
b &:= b - \alpha \frac{db}{\sqrt{s_{db}}}
\end{aligned}
$$

假设 w 是最优化方向的梯度群，b 是非最优化方向的梯度群 (震荡), 那么我们想要 w 更新的更快，那就要 s_{dw} 小，dw 小，在实际操作中，防止 s_{dw} 为 0，修改为:

$$
\begin{aligned}
w &:= w - \alpha \frac{dw}{\sqrt{s_{dw} + \epsilon}} \\
\beta &= 0.999, \epsilon = 10^{-8}
\end{aligned}
$$

五，Adam
Adaptive moment estimation

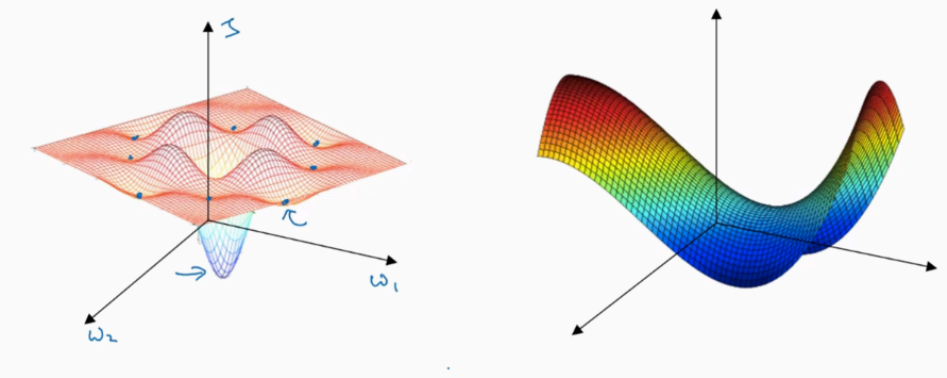
$$v_{dw} = \beta_1 v_{dw} + (1-\beta_1)dw$$
$$v_{db} = \beta_1 v_{db} + (1-\beta_1)db$$
$$s_{dw} = \beta_2 s_{dw} + (1-\beta_2)dw^2$$
$$s_{db} = \beta_2 s_{db} + (1-\beta_2)db^2$$
$$v_{dw}{}^{correct} = \frac{v_{dw}}{1-\beta_1^t}$$
$$v_{db}{}^{correct} = \frac{v_{db}}{1-\beta_1^t}$$
$$s_{dw}{}^{correct} = \frac{s_{dw}}{1-\beta_2^t}$$
$$s_{db}{}^{correct} = \frac{s_{dw}}{1-\beta_2^t}$$
$$w := w - \alpha \frac{v_{dw}^{correct}}{\sqrt{s_{dw}^{correct}+\epsilon}}$$
$$b := b - \alpha \frac{v_{db}^{correct}}{\sqrt{s_{db}^{correct}+\epsilon}}$$
$$\beta_1 = 0.9$$
$$\beta_2 = 0.999$$
$$\epsilon = 10^{-8}$$

六，学习率衰减

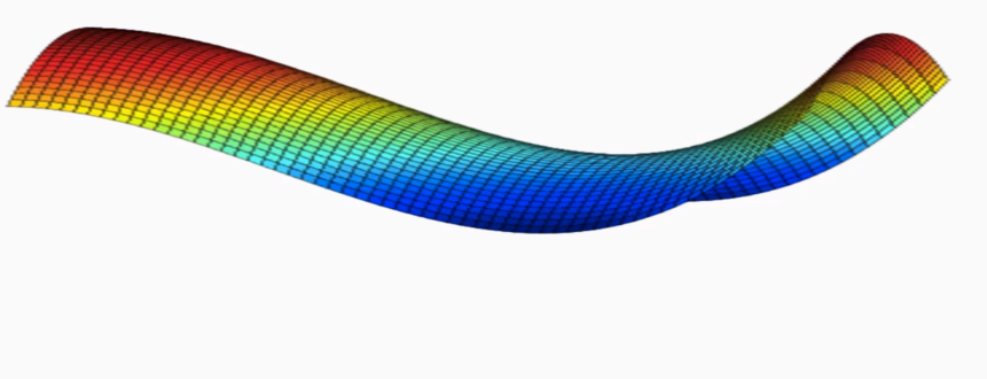
$\alpha = \frac{1}{1+rate_{decay}*epoch}\alpha_0$

七，局部最优的问题

按照 Andrew 的观点，随着人们对最优化算法的不断了解，大部分梯度为零的点，并不是局部最优点，而是鞍点 (saddle point)



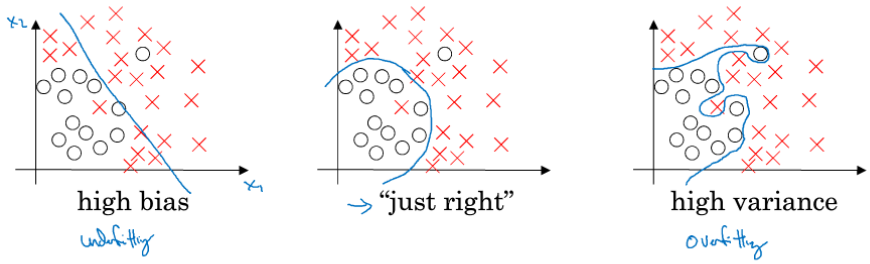
优化算法的问题不再是陷入局部最优，而是如何更快的越过平稳期 (plateaus)



1.3 偏置与方差

偏置 bias 欠拟合 underfitting

方差 variance 过拟合 overfitting



1.4 正则化

这边给出了几个正则化的方法

一，L1,L2 正则

这个是 L2 正则

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

二，Dropout

三，Early stopping

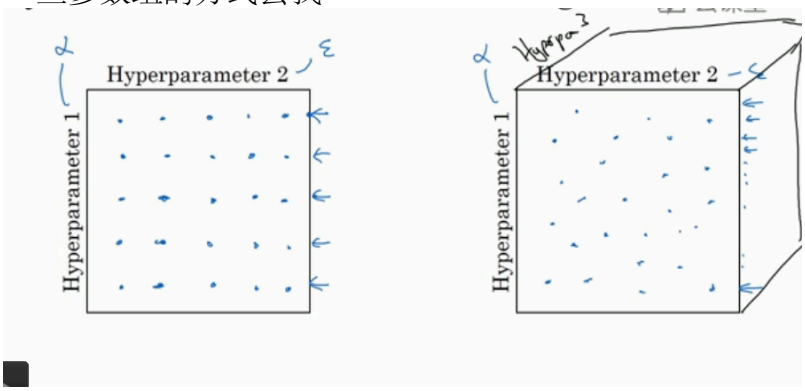
1.5 超参数的调试

调试超参数

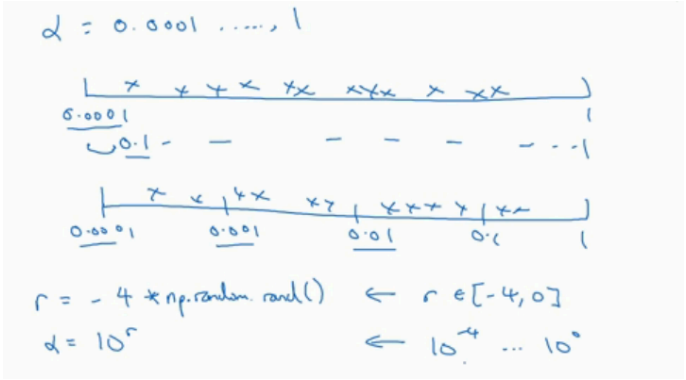
一，调试超参数

区别于传统的机器学习，深度学习的参数选择，不用 grid search, 而是用随机选取

一些参数组的方式去找



二，用合适的尺度去寻找超参数



三，超参数的训练实践

Panda 要么只训练一个模型，然后小心照看

Caviar 要么训练多个模型，取最好的一个

四，Batch Norm

normalize inputs, 对输入值进行正则化，自然语言处理和图像一般用不到，但是其他的情况可以试试

那么在神经网络的第二层开始，可以把第二层输出（也就是第三层的输入）的值进行 normalize.

$$\mu = \frac{1}{m}\Sigma_i z^i$$
$$\sigma^2 = \frac{1}{m}\Sigma_i (z_i - \mu)^2$$
$$z^{(i)}_{norm} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$
$$\tilde{z}^{(i)} = \gamma z^{(i)}_{norm} + \beta$$

Batch norm 的反向传播
在每一个隐藏层，除了参数 w 之外，反向传播还要更新的有每一层上的 γ,β
测试时候的 Batch norm
在训练的时候，在每一个 batch 上，对 l 层的 $z^{(i)(l)}_{norm}$,$\tilde{z}^{(i)(l)}$ 做指数加权平均来一直保持追踪，最后在测试的时候，用指数加权平均后的 $z^{(i)(l)}_{norm}$,$\tilde{z}^{(i)(l)}$，还有当前已经训练好的 w,γ,β 来预测.

2 机器学习的策略

如何判断用以下什么办法改进机器学习的性能:
1，收集更多数据（数据更多，比如，收集 20000 个人脸）
2，收集更多样的训练数据（单种数据的类型更多，比如，1 个人再收集 10 个脸）
3，在梯度下降的时候，训练时间更长
4，使用 Adam，替换梯度下降
5，尝试更小（大）的网络
6，尝试 dropout
7，加上 $L2$ 正则
8，改变网络结构（比如：激活函数，隐藏层）

2.1 正交化

1，确保在训练集上 ok，达到人类水平：
更大的网络，更优的优化算法,.....
2，dev set
正则化，更大的训练集
3，test set
更大的 dev set(在 dev set 上过拟合了)
4，真实世界
改变 dev set, 或者改正 cost function(开发集分布设置不正确，或者 cost function 测量的不对)
注意：early stop 同时调整了 train set 和 dev set, 不是很正交化

2.2 单一数字评估标准

这边的意思是在选择哪个算法好的时候，如果有多个评判标准，不好选，所以我们将这多个评判标准组合成一个
$F1 = \frac{2}{\frac{1}{P}+\frac{1}{R}}$ 为 P 查准率，R 查全率的调和平均数 (harmonic mean)

2.3 满足和优化指标

satisficing and optimizing metrics

Classifier	Accuracy	Running time
A	90%	80 ms
B	92%	95 ms
C	95%	1 500 ms

$$maximize\ accuracy$$
$$subject\ to\ running\ time \leqslant 100ms$$

2.4 training set, dev set, test set

如何设置训练集，开发集，测试集

1，数据的分布：

核心思想是各个集的分布是相同的 (尤其是 dev test)

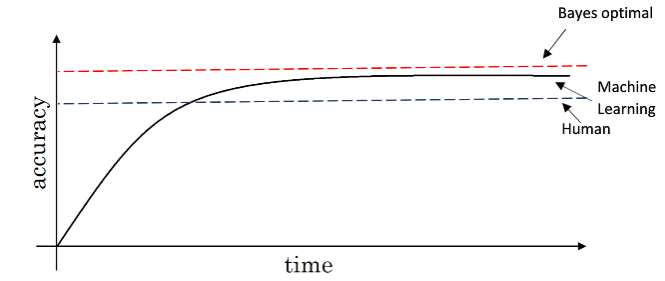
2，数据集大小的划分：

区别于之前的机器学习 7/2/1/，现在用 98/1/1/

3，改变 dev,test 数据集

2.5 人的表现

只要我们的算法比人类水平低，我们可以从以下几个方面看：



1，获取更多人类标注数据

2，从人类角度思考问题，为什么人类可以判断正确，机器判断错误

3，进一步分析偏置/方差

2.6 可以避免的偏置

相对于人类水平，机器可以达到的最高水平 (可以避免的偏置 bias)

	Classification error (%)	
	Scenario A	Scenario B
Humans	1	7.5
Training error	8	8
Development error	10	10

A
训练误差和人类水平之间差的很大，是高偏置问题 (bias)，欠拟合，解决思路就是降低偏置：更大的网络，训练更长的时间
B
训练误差和人类水平之间的差值不大，但是和开发集的误差比较大，要解决过拟合问题，思路就是降低方差：使用正则化，更多的训练数据

2.7 理解和超越人的表现

理解人类的表现:
人类的水平可以用来大约估计贝叶斯水平

	Classification error (%)
Typical human	3.0
Typical doctor	1.0
Experienced doctor	0.7
Team of experienced doctors	0.5

这边 bayes error 小于等于 0.5%

A
人类水平的选择无关紧要，可避免的 bias 在 4% 到 4.5% 之间，方差为 1%，关注

	Classification error (%)		
	Scenario A	Scenario B	Scenario C
Human (proxy for Bayes error)	1	1	0.5
	0.7	0.7	
	0.5	0.5	
Training error	5	1	0.7
Development error	6	5	0.8

降低 bias

B

人类水平的选择无关紧要，可避免的 bias 在 0% 到 0.5% 之间，方差为 4%，关注降低 variance

C

Bayes error 选择为 0.5%，应为我们不能比人类水平更好，不然训练集过拟合了，可避免的 bias0.2%，variance 0.1%，关注降低 bias

总结

1，人类错误水平可以估计贝叶斯错误

2，如果人类错误和训练错误的差值比训练错误与开发集错误的差值大，则关注降低偏置

3，如果人类错误和训练错误的差值比训练错误与开发集错误的差值小，则关注降低方差

超过人类的表现:

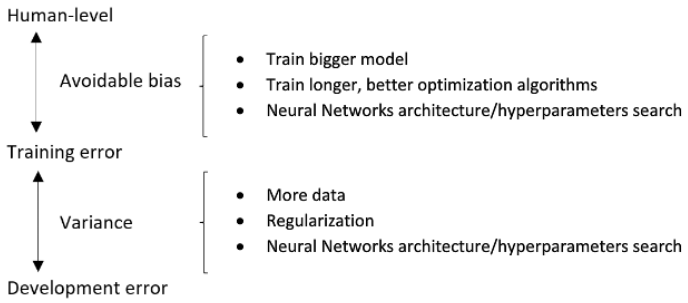
	Classification error (%)	
	Scenario A	Scenario B
Team of humans	0.5	0.5
One human	1.0	1
Training error	0.6	0.3
Development error	0.8	0.4

A

Bayes error 为 0.5%，可避免的偏置为 0.1%，方差为 0.2%
B
在这种情况下，我们暂时没有足够信息说是过拟合还是超越了人类表现
structured data(结构化数据)，容易超越人类:
1，在线广告
2，推荐系统
3，预测物流时间
4，贷款批准
5.，.....
natural perception task(自然感知任务)，人类往往变现的更好：1，语音识别
2，自然语言处理
3，图像识别

2.8 总结

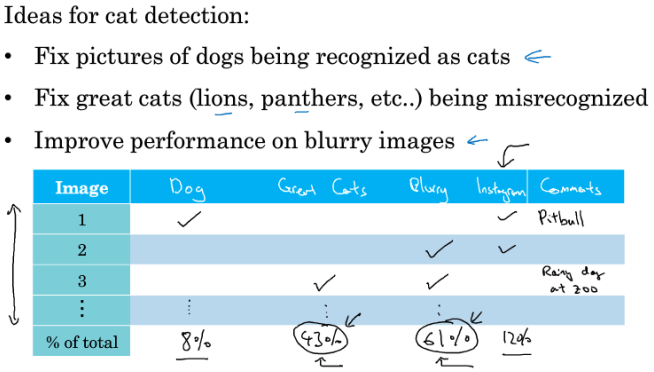
核心是找到现在的问题是解决可避免的偏置问题还是解决方差



2.9 误差分析

手工误差分析

人工找出比如说 100 个错误的 dev set 数据，人工分析错误的原因，记录如下的一张表，从改进后最可能减少误差的地方开始入手



2.10 标注数据有误

当标注数据有错误的时候，是否值得花时间去修正?

training set:

1，随机 (random) 错误一般不需要修正，原因是深度学习的鲁棒性

2，系统化的错误 (systemertic) 需要修正，比如说一个标注人员持续的将白色的狗标注成猫

dev set:

首先我们用手动误差分析的方法，人工分析 100 个错误数据，在这 100 个错误数据中，找出标注错误的数据，然后计算如下指标:

1，整体的开发集错误

2，标注错误导致的错误

3，其它情况导致的错误

通过简单的计算各种错误在总误差上的占比，可以清楚的知道要不要修正标注错误

如何修正错误的 dev/test set:

1，保证在 dev set 和 test set 上的分布要是同一种分布
2，考虑算法预测' 错' 的，真实的是标错的，但是预测对了，并且考虑算法预测' 对' 的，真实的是标错的，但是算法也预测错了. 这个情况不建议在实际中用的原因是，算法往往 2% 是错的，98% 是对的，那么考虑对的成本比考虑错的成本大好多
3，train 和 dev/test 的分布略有不同

2.11 快速搭建系统

快速搭建第一个系统
1，建立 train/dev/test set，建立误差度量
2，快速搭建初始系统
3，偏置/方差分析
4，误差分析
5，迭代.....

2.12 在不同分布上进行 training 和 testing

如何处理训练数据和开发数据来自于不同分布的问题?
假设要做一个猫的检测，我们从网上爬下了 200K 个清晰的图片，但是我们拿到了的开发集 (这个开发集和用户最终要检测的数据是同分布的) 是 10K 个模糊的猫图片
1，将所有的图片混在一起，205k 的 training set，2.5k 的 dev set，2.5k 的 test set
2，先在 10k 里面取出 2.5k 的 dev set，2.5k 的 test set，然后剩下的 205K 做 training set
指导思想是让 dev/test set 里的数据分布和真实情况一致，并将一部分这些数据放入训练数据中去

2.13 在不匹配数据划分上的偏置和方差

data mismatch proplem 数据不匹配问题

当 training set 和 dev set 的分布一样时，我们很容易通过加入人类水平，来看问题是偏置问题还是方差问题，但是当我的 training set 和 dev set 的分布不一样时我无法知道这个差值是因为 dev 数据在 training set 里面没有，还是因为分布不一样. 为了解决这个问题，在分布不一致的时候，我们需要引入 training-dev set，这个 set 来源于 training set，但是我们不用来做训练.

human level,train,train-dev,dev,test

human level

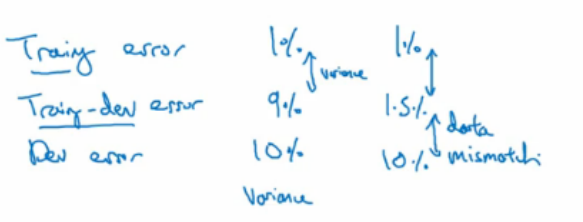
train error

train-dev error

dev error

test error

如何处理数据不匹配问题?



并没有系统的解决方案，但是有如下的一些尝试:

1，使用手动误差分析，看，了解训练集和开发测试集之间的具体差异

2，将训练数据变得和 dev/test 数据更相似，或者收集更多和 dev/test 数据相似的数据

2.14 迁移学习

一些特点

1，任务 A 和 B 有相同的输入 X

2，A 的数据比 B 多

3，A 网络的低层结构对 B 有用

2.15 多任务学习

一些特点

1，需要这个大任务在训练的过程中有可以共享的低层网络结构

2，需要每个小任务的数据个数差不多

3，网络结构足够大的情况下，多任务一般比训练多个单任务效果好

4，迁移学习比多任务学习更常见，多任务学习多用于视觉上的目标检测

2.16 端到端学习

end-to-end learning 的一些特点

1，这个 end2end，颠覆了之前的 pipeline

2，要求大量数据

3，有些任务还是拆分成一步一步的效果更好

优点:

1，让数据说话

2，更少的需要手动设计一些主件

缺点:

1，需要大量数据

2，会失去潜在的有用的手工设计的主件

要不要用 end-to-end learning 的关键在于：是否能够拿到大量充足的数据

3 卷积神经网络

3.1 边缘检测

这边要明白卷积核的概念

3.2 padding，步长，卷积，池化

一些小但是重要的概念

padding

stride

convolutions

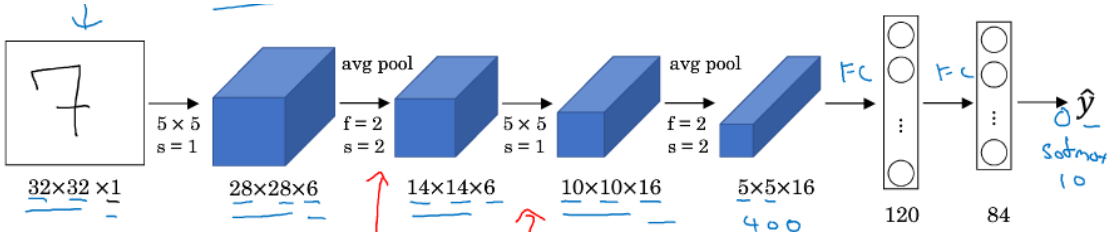
pooling

3.3 经典网络

介绍几个经典网络结构

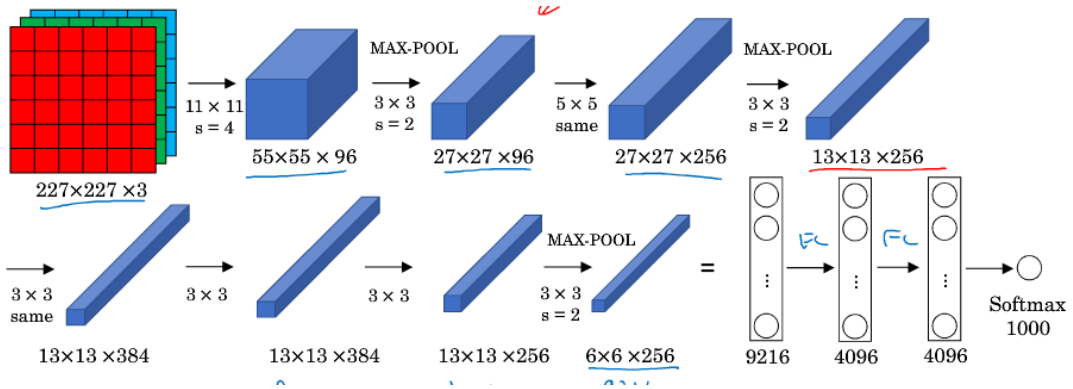
LeNet-5

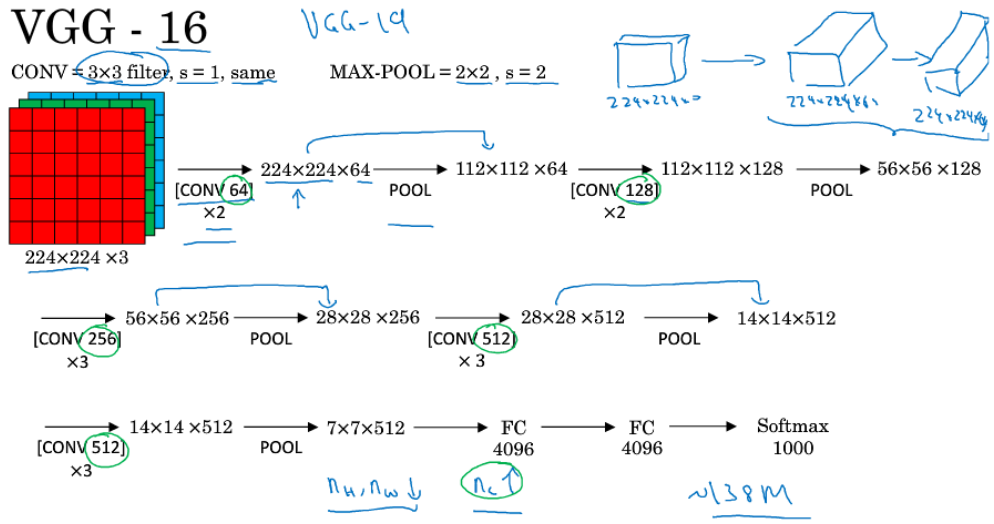
input-conv-pool-conv-pool-fc-fc-output



input-conv-pool-conv-pool-conv-conv-conv-pool-fc-fc-softmax

VGG-16


227×227 ×3
11 × 11
s = 4
55×55 × 96
MAX-POOL
3 × 3
s = 2
27×27 ×96
5 × 5
same
27×27 ×256
MAX-POOL
3 × 3
s = 2
13×13 ×256
3 × 3
same
13×13 ×384
3 × 3
13×13 ×384
3 × 3
13×13 ×256
MAX-POOL
3 × 3
s = 2
6×6 ×256
=
9216
FC
4096
FC
4096
Softmax
1000


VGG - 16
CONV = 3×3 filter, s = 1, same
MAX-POOL = 2×2 , s = 2
224×224 ×3
[CONV 64]
×2
224×224×64
POOL
112×112 ×64
[CONV 128]
×2
112×112 ×128
POOL
56×56 ×128
[CONV 256]
×3
56×56 ×256
POOL
28×28 ×256
[CONV 512]
× 3
28×28 ×512
POOL
14×14×512
[CONV 512]
×3
14×14 ×512
POOL
7×7×512
FC
4096
FC
4096
Softmax
1000

3.4 残差网络

Residual Networks(ResNets)

$$z^{[l+1]} = w^{[l+1]}a^{[l]} + b^{[l+1]} \quad a^{[l+1]} = g(z^{[l+1]})$$
$$z^{[l+2]} = w^{[l+2]}a^{[l+1]} + b^{[l+2]} \quad a^{[l+2]} = g(z^{[l+2]})$$
$$change\ to\ a^{[l+2]} = g(z^{[l+2]+a^{[l]}})$$

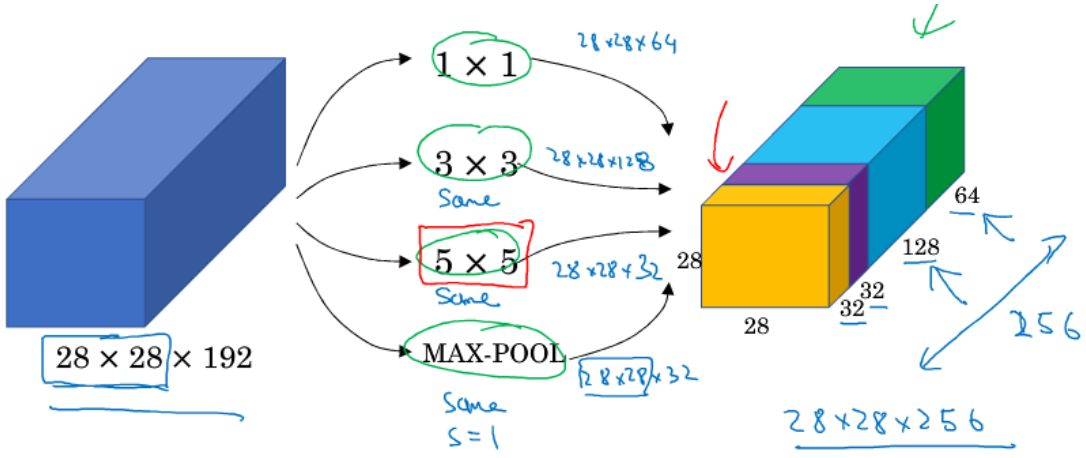
resnet 有助于生成更深的网络

3.5 1X1 卷积

1X1 卷积可以用来改变隐层的 channel 深度

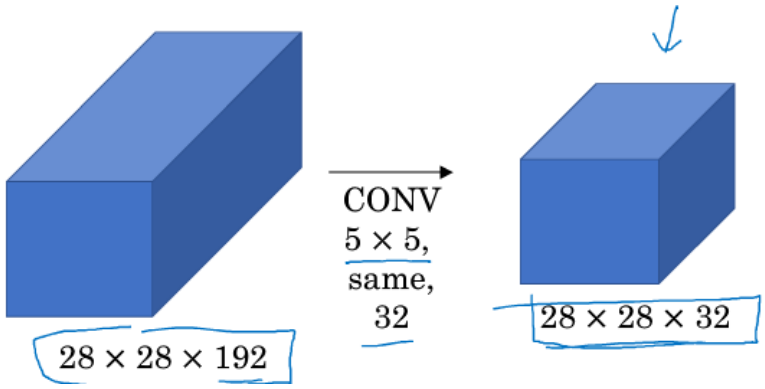
3.6 inception

inception network 的思想是将 1X1,3X3,5X5,MAX-POOL 都放在一层修改网络结构来降低复杂度



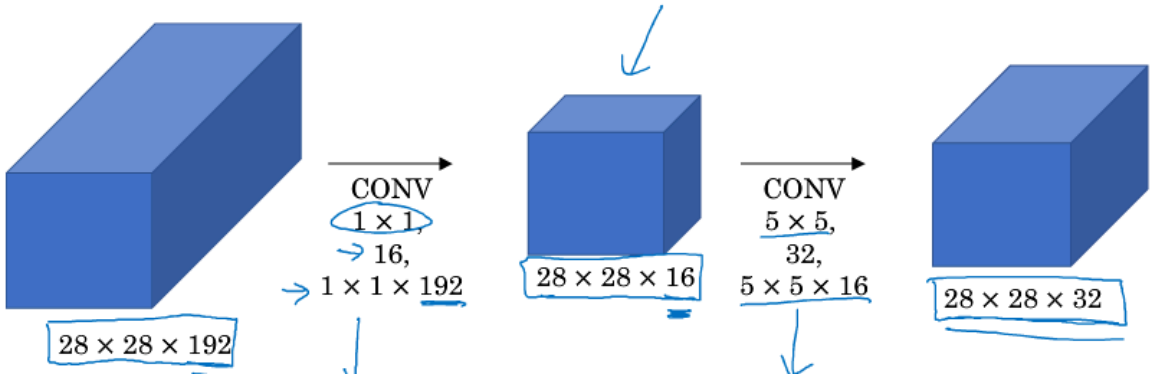
input:28x28x192 conv:5x5 same 32 output:28x28x32

计算复杂度:5x5x192x28x28x32=120M



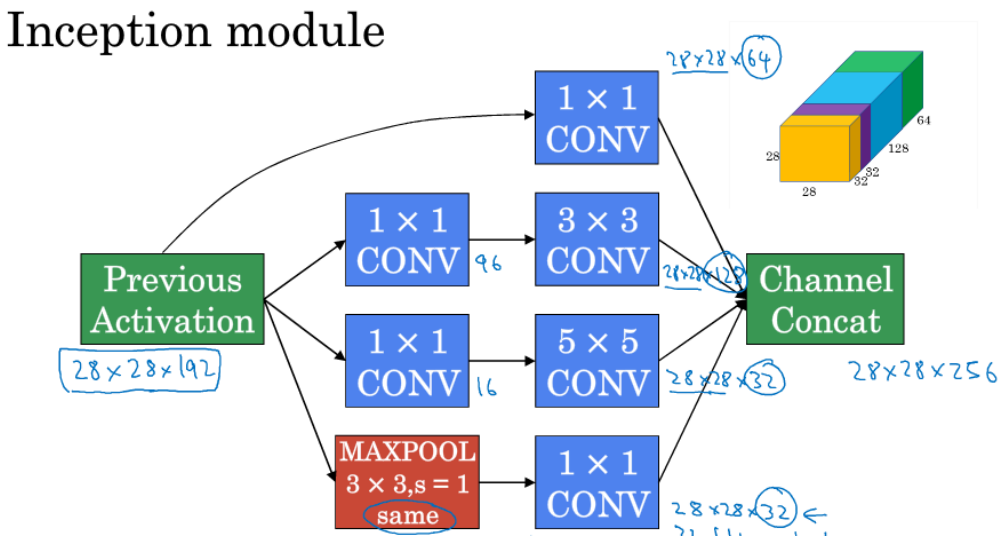
修改后:input:28x28x192 conv:1x1 16 bottleneck layer: 28x28x16 conv: 5x5 32 output: 28x28x32

计算复杂度:5x5x16x28x28x32+1x1x192x28x28x16=12.4M



综合上述的 1：将所有的卷积和池化叠加在一起，2：每一个卷积拆成 1x1conv 和 5x5conv(中间有一个 bottleneck layer) 3：max-pool 用 same padding 来使得维度一致，三点结合，得到 inception 块

注意到 googlenet 里面会在某些隐层后面直接加 softmax 来提前预测



3.7 迁移学习

视觉领域的迁移学习

1，freeze softmax 之前的所有隐层，鉴于这种情况，可以直接利用之前的得到的权重，代入输入，算出 softmax 之前的输出，存入硬盘，相当于只训练 softmax

2，只 freeze 前面部分的隐层，要么利用之前的网络结构重新修改参数，要么直接改变网络结构重新训练

3，将之前得到的权重当做是初始化

3.8 数据扩充

以下是扩充数据的一些思路：

1，镜像对称

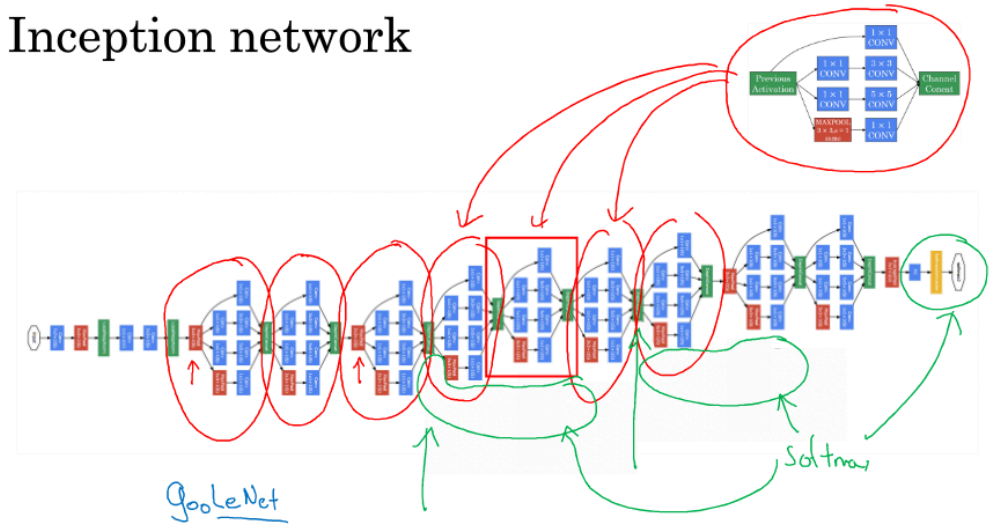
2，随机剪裁，有可能裁的不好

3，旋转

4，剪切图像

5，局部弯曲

...



1，色彩转换 (color shifting), 给 RGB 三个 channel 加上不同的失真值
PCA color augmentation 在 AlexNet paper 里面讲了，主要作用是在做颜色增强的时候，使得颜色多的变得多，颜色少的变得少

3.9 目标定位

b_x, b_y, b_h, b_w
目标定位需要输出 bounding box 的坐标和大小，还有类标

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

3.10 特征点检测

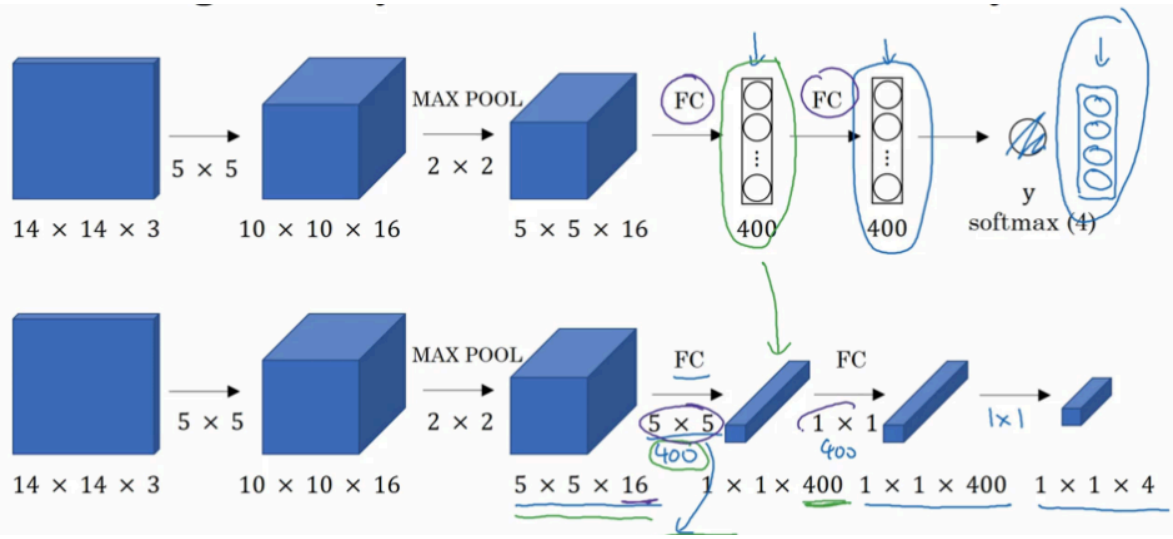
landmark detection
人为标注特征点，输入神经网络

3.11 目标检测

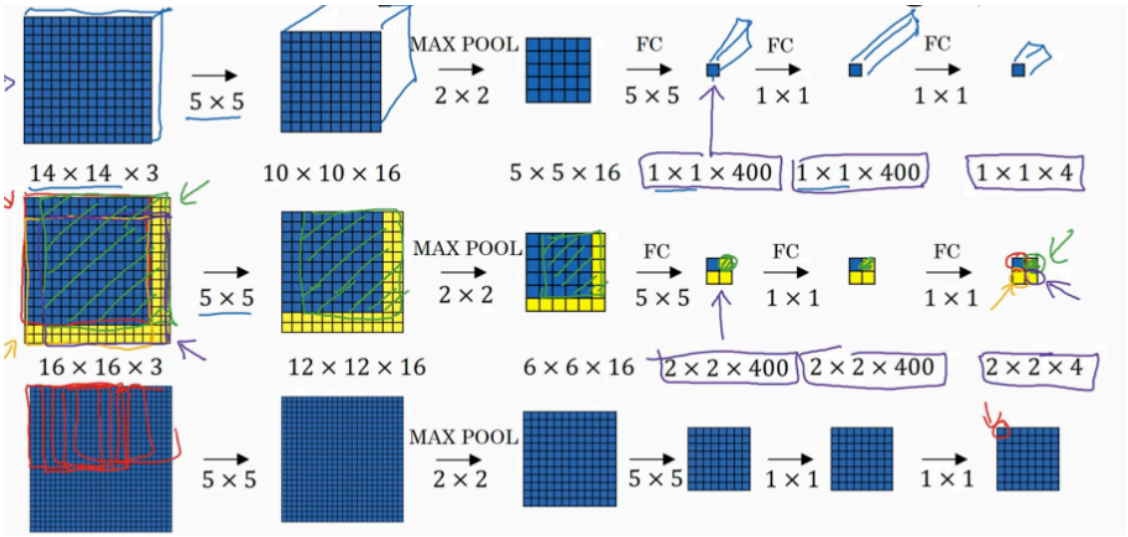
用滑动窗口的方式构建目标检测系统
Training set :x y，剪裁合理的目标图片包含正负样例
选择合适的窗口大小，在检测目标上滑动
改变窗口的大小，再滑动一遍
问题是计算成本太高

3.12 滑动窗口

如何高效的实现滑动窗口的目标检测
1，把 FC 层转化为卷积层，相当于 1x1xchannel



2，事实上，在计算的时候，不需要真正的滑动，这样，大量重复的地方可以计算一次



3.13 bounding box

如何获得更精准的边界框

YOLO you only look once

对要进行目标检测的图片进行格子划分，然后，对每一个单元格:

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

3.14 交并比

如何评价目标检测算法？intersection over union
$iou = \frac{i}{u}$

3.15 非极大值抑制

Non-max suppression
算法可能对同一个物体进行了多次检测，使用非极大值抑制可以保证算法只对物体检测一次
p_c 是个概率值，首先找到最高的 p_c 然后看能不能抑制和它交并比比较高的，然后进行下一个
具体步骤：
假设每一个格子的输出为：这边假设只预测一个物体 $\begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \end{bmatrix}$
去除 p_c 低于阈值的
while 还有剩余的 box:
选择 p_c 最高的 box
去除与上一步的 box IOU 值大于 0.5 的
如果是多个目标检测，对每个目标都独立的运作 non-max suppression

3.16 Anchor boxes

如何让算法对一个格子预测多个目标？使用 Anchor boxes(锚)
假设在一个图片中，两个物体有交叉，怎么办
预先定义两个或者多个 anchor box 形状的 box，这边假设是两个，那么，现在的输出变为

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

当不使用 anchor box 时，算法为：训练图像里的每一个目标物体都被指派到包含该物体中心点的格子单元

使用 anchor box 算法：训练图像里的每一个目标物体都被指派到包含该物体中心点的格子单元，不仅如此，还在这个格子单元里面给这个目标物体分了一个最高 IOU 的 anchor box

事实上，两个物体有 overlap 的概率很小，anchor box 的作用更多的是能够更有争对性的处理，比如说区别高高瘦瘦的人和矮矮长长的汽车

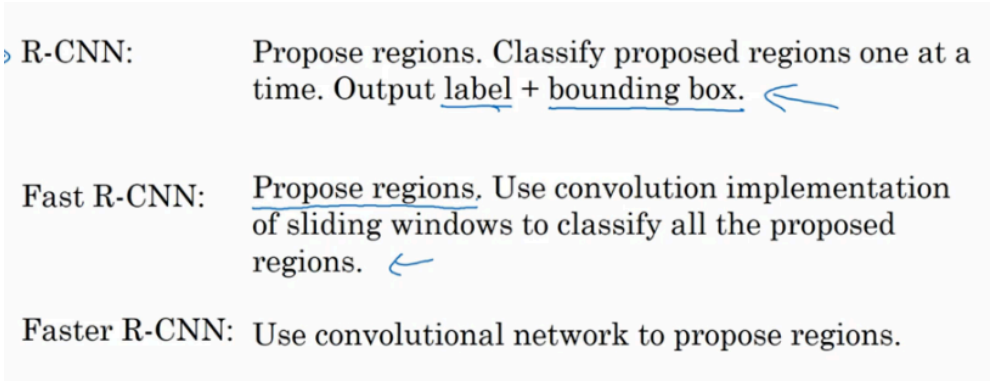
手工挑选几个 anchor box 的形状，或者，使用 k-means 来选取具有代表性的形状

3.17 yolo

综合起所有的技术
构造训练数据的 label: 3x3x2x8 19x19x2x8 19x19x5x8
预测：3x3x2x8 19x19x2x8 19x19x5x8
应用 non-max suppressed:
假设使用 2 个 anchor box
对每一个格子单元，得到两个预测的边界框
去除低概率的 bounding box
对每一种类别 (label) 单独使用非极大值抑制来生成最终的预测

3.18 RPN 网络

region proposals：R-CNN 候选区域
图片上的某些区域是明显没有目标的，所以可以不去卷积
跑一个图像分割算法，找出可能存在对象的区域
改进
Andrew Ng 更倾向于 YOLO



3.19 人脸识别

分两种:

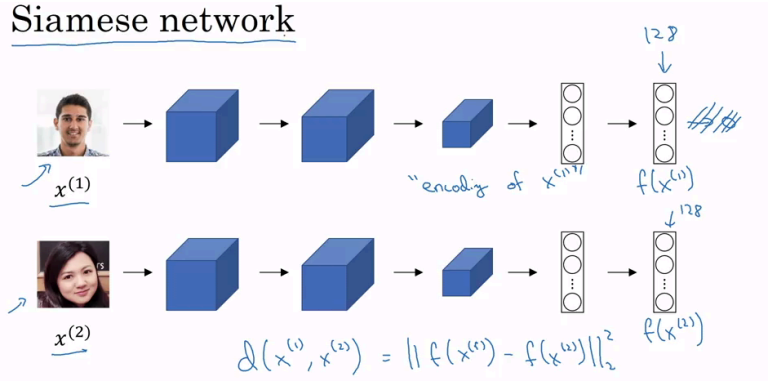
验证

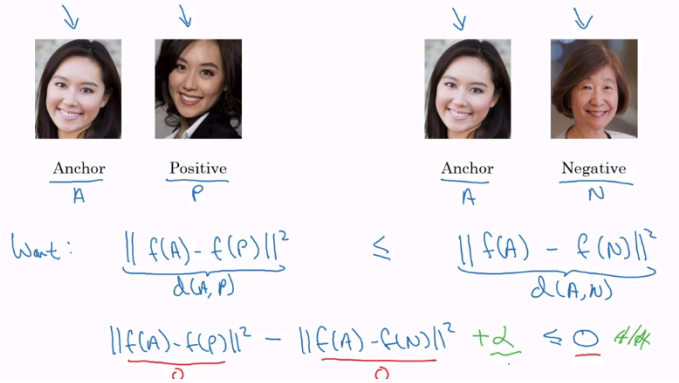
识别

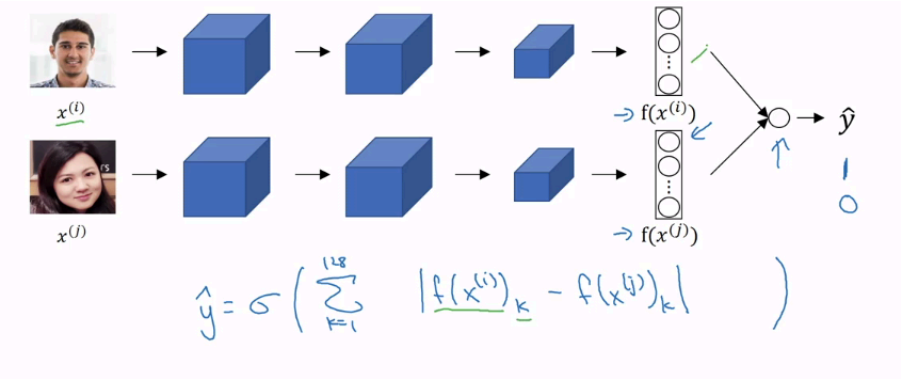
3.20 one-shot

需要通过单单一张图片来识别

3.21 siamess







3.22 triplet loss

3.23 面部验证与二分类

3.24 神经风格转移

3.25 代价函数

3.26 风格损失函数

3.27 一维到三维的推广

3.28 python 代码记忆

np.exp() v = v.reshape((v.shape[0]*v.shape[1],v.shape[2])) image = np.array([])

$\|x\| = np.linalg.norm(x, axis = 1, keepdims = True)$

np.dot(x1,x2) np.outer(x1,x2) np.multiply(x1,x2) np.sum() np.abs() ‘np.dot(x,x)‘ $= \sum_{j=0}^{n} x_j^2$

import matplotlib.pyplot as plt a = np.random.randn(28,28) plt.imshow(a)

np.zeros((w.shape[0],w.shape[1]))

np.squeeze() 去除维度中 dim=1 的维度，降低维度