



清华大学  
Tsinghua University

数据科学研究院  
Institute for Data Science

# 深度神经网络

青年AI自强计划计算机视觉课

# 0: 引言——一拳超人&深度学习

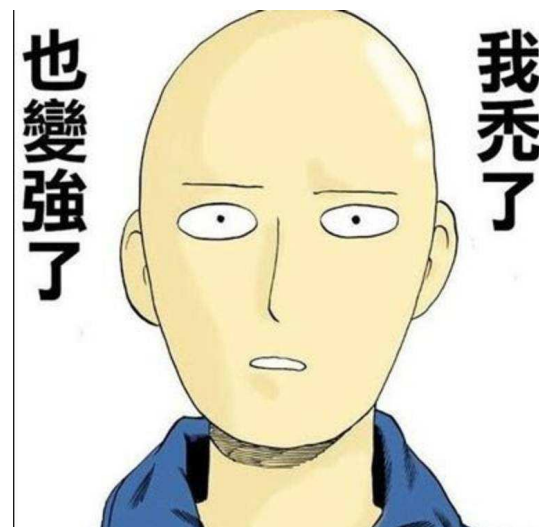


清华大学  
Tsinghua University

数据科学研究院  
Institute for Data Science



深度学习：  
我“深”了，也变强了



# 0: 引言



清华大学  
Tsinghua University

数据科学研究院  
Institute for Data Science

虽然被证明了有  
Universality性质



浅层神经网络  
仍不敌传统机  
器视觉算法



忽然他变“深”了



真的假的？

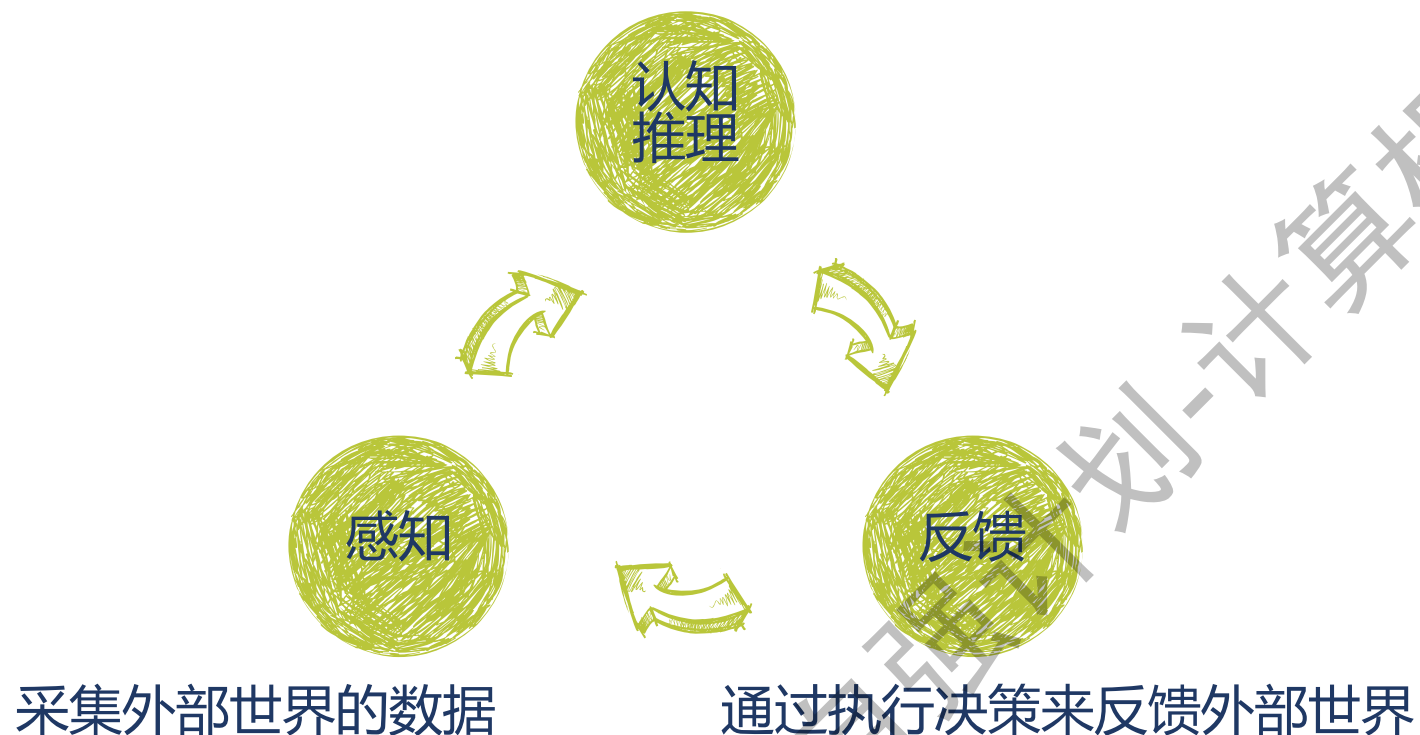


横扫天下，一招搞定

## 1.1: 广泛涉猎-CV



深度处理数据并形成决策



智能产生的过程

### CV领域

传统“模式识别”算法，目前仅在结构化场景中还保有一些为数不多的阵地，并且这些阵地也在逐步被深度学习蚕食。

对于其他所有通用场景，深度学习已经一统天下了。



## 1.2: NLP



清华大学  
Tsinghua University

数据科学研究院  
Institute for Data Science

### NLP的 派别之争

上世纪70年代



关于如何分词  
产生了两个派别

想通过语法规则  
来进行分词的“**规则派**”

想通过对大量真实数据学习  
来进行分词的“**经验派**”

同一细分学科的不同派别  
**竟然不一起开会?**



因为别人根本听不懂  
你们在说什么

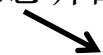


你说的



hi, 小姐姐, 最近hinton在一篇关于CapsuleNet的论文中提出了一种“动态路由算法”来替代反向传播更新参数, 我觉得很受启发, 你读了么, 觉得怎么样呀?

她听的



Hi, 小姐姐, 最近%¥#\*\*&%¥#  
我觉得很受启发, 你读了么, 觉得怎么样呀?

相同专业背景的人  
容易自来熟

例如你想跟一个不懂  
深度学习的小姐姐搭讪

## 1.2: NLP



清华大学  
Tsinghua University

数据科学研究院  
Institute for Data Science



原本应该亲近的人  
竟然如此水火不容

最终的结果

依靠“语法规则”的分词  
准确率在达到某一个瓶颈  
后就再难提升

依靠对大量数据学习来分词的，  
随着累计数据的不断增加，  
准确率在缓慢的提升

最终具备“大数据”思维的  
“经验派”取得了胜利  
**NLP的主流转向ML**

SQuAD1.1 Leaderboard

Rank	Model	EM	F1
	Human Performance Stanford University (Rajpurkar et al. '16)	82.304	91.221
1 Oct 05, 2018	BERT (ensemble) Google AI Language <a href="https://arxiv.org/abs/1810.04805">https://arxiv.org/abs/1810.04805</a>	87.433	93.160
2 Sep 09, 2018	nlnet (ensemble) Microsoft Research Asia	85.356	91.202
3 Jul 11, 2018	QANet (ensemble) Google Brain & CMU	84.454	90.490

不久前谷歌发布了BERT网络，在著名的NLP数据集上横扫了整个榜单。号称已经在准确率上全面超越了人类。  
当然，BERT模型也属于深度学习算法的范畴内，真相如大家所见，深度学习也已经有在NLP领域“一统江湖”的趋势了。

# 1.3: GAN



非结构化数据感知  
已是DNN天下



输出与反馈呢



输出图像和语言来改变其他人思想的  
如电影、文字出版物等等



输出策略来改变世界  
如口渴了会驱动肢体寻取水源

计算机如何输出  
图像和语言

GAN  
生成对抗网络



名字很直接



Goodfellow

好 小伙

GAN是他



喝醉后发明的

听说喝酒也能助力AI创新?

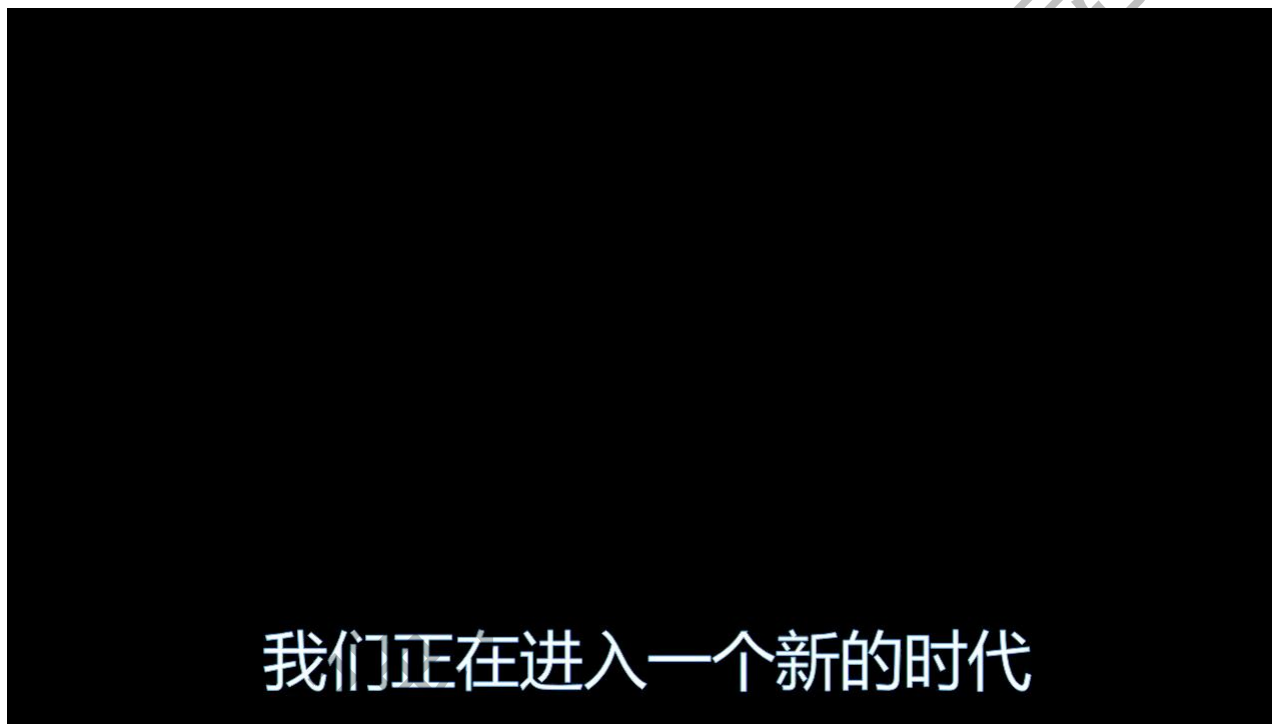


GAN由这个慈眉善目  
又有点害羞的好小伙发明

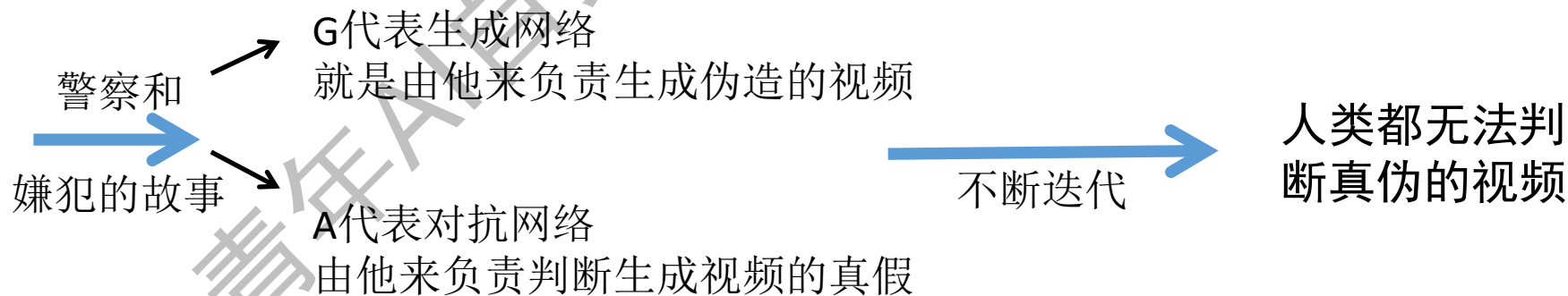
# 1.3: GAN



一个直观的视频，认识GAN



GAN原理



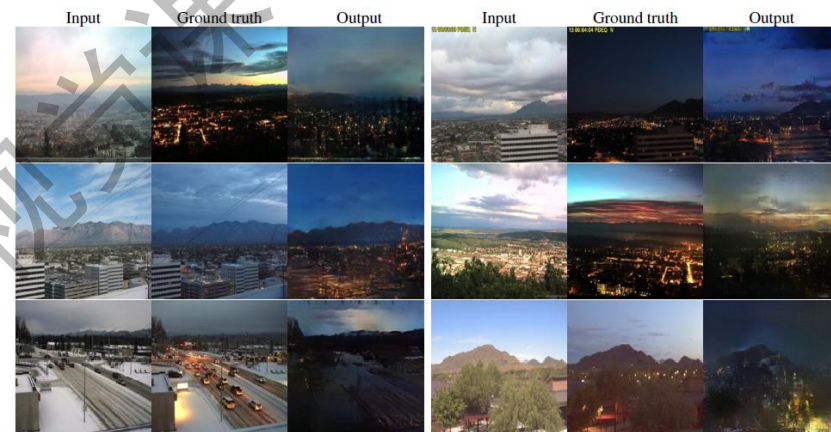
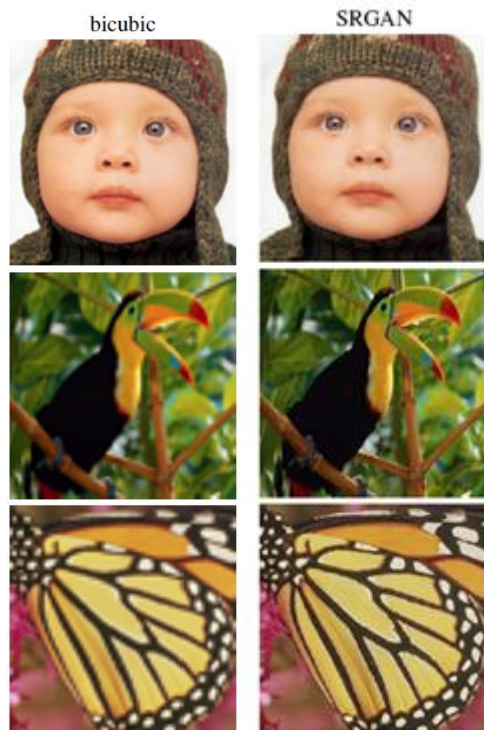


# 1.3: GAN



清华大学  
Tsinghua University

数据科学研究院  
Institute for Data Science



This small blue bird has a short pointy beak and brown on its wings

This bird is completely red with black wings and pointy beak

A small sized bird that has a cream belly and a short pointed bill

A small bird with a black head and wings and features grey wings



GAN被誉为  
“20年来最酷的深度学习思想”

又是DNN?



How are you  
怎么 是 你

# 1.4: reinforcement learning-强化学习



清华大学  
Tsinghua University

数据科学研究院  
Institute for Data Science



老狗三宝：

CNN、MCTS、RL

新狗两宝：

MCTS、RL

RL  
黑盒解释



输入规则  
(什么情况下会受到奖励)



输出策略  
(最短时间内获取最大奖励)

RL的早已产生，知道遇见DNN才发生质变

How old are you  
怎么 老 是 你



# 1.4: RL



## 广义的“策略” - 棍找孔

这么容易  
不算策略？

对于计算机

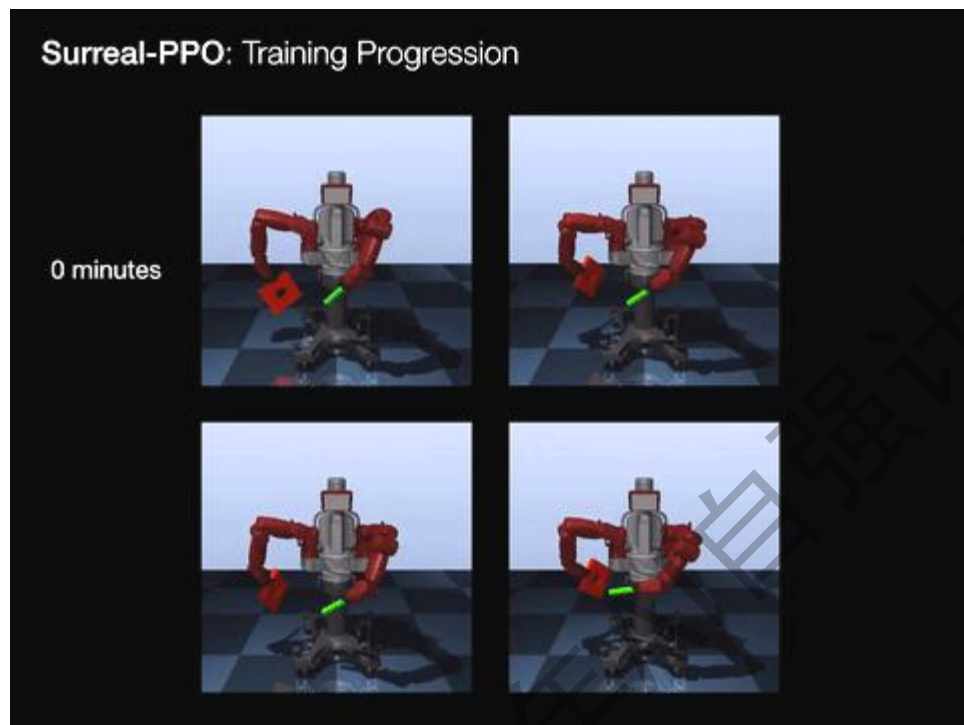


尝试用数学语言建模  
并用编程思想描述

十分复杂



什么是棍子，什么是孔  
相对位置，每次移动的最小单位  
每次移动的方向等々等々



李飞飞团队强化学习案例



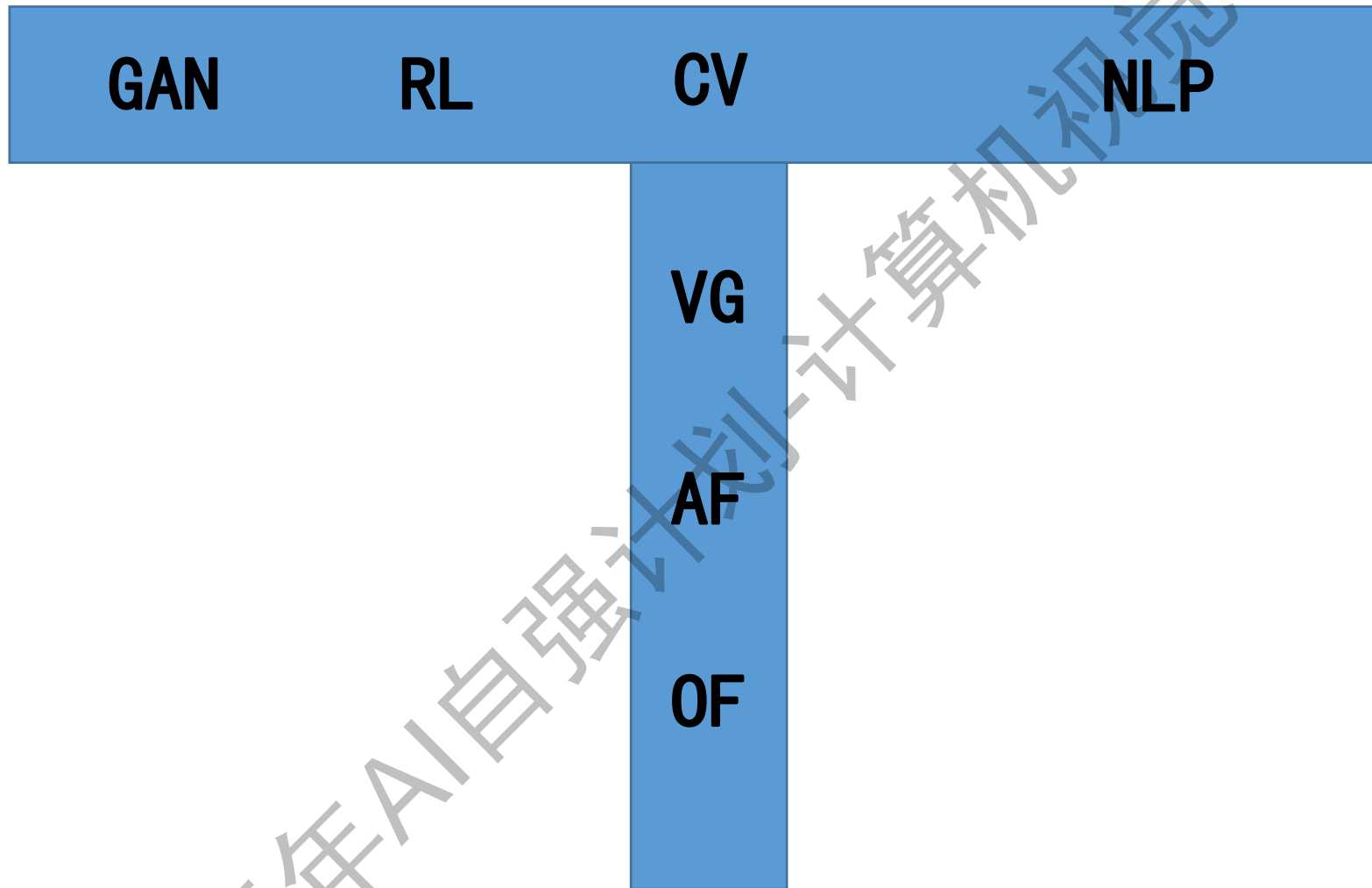
自动驾驶/机器人属综合问题  
但也多多少少用到了DNN知识

## 1.5: 小结



### 本次课程的结构

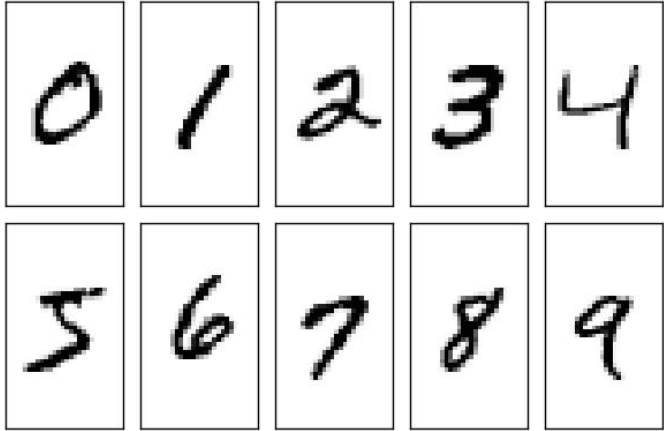
先看热闹



再看门道



## 本次作业数据集介绍

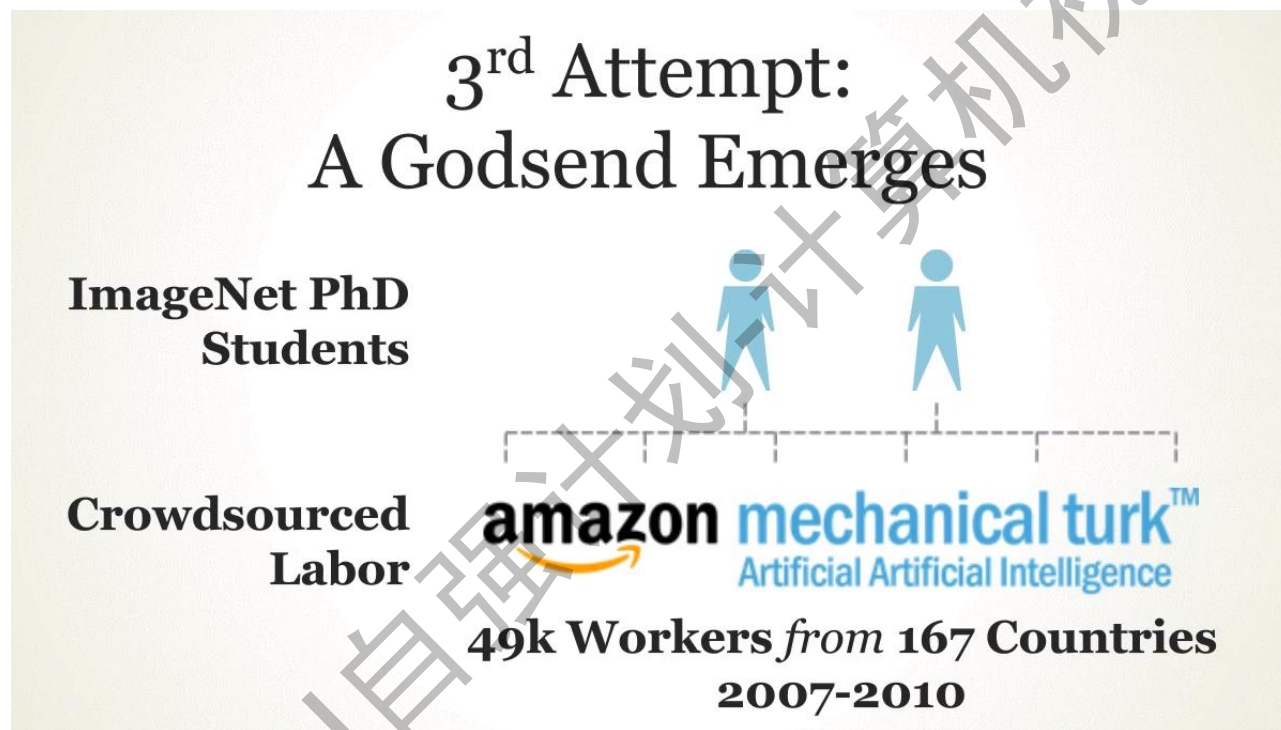


	MNIST	ImageNet
样本数量	7万	1420万
类别	10	预期10万，目前2.2万
图像尺寸	28*28	至少为227*227
格式	灰度	灰度（不敢用RGB）

## 2.1: ImageNet



这种数量级的数据集是如何标注的？

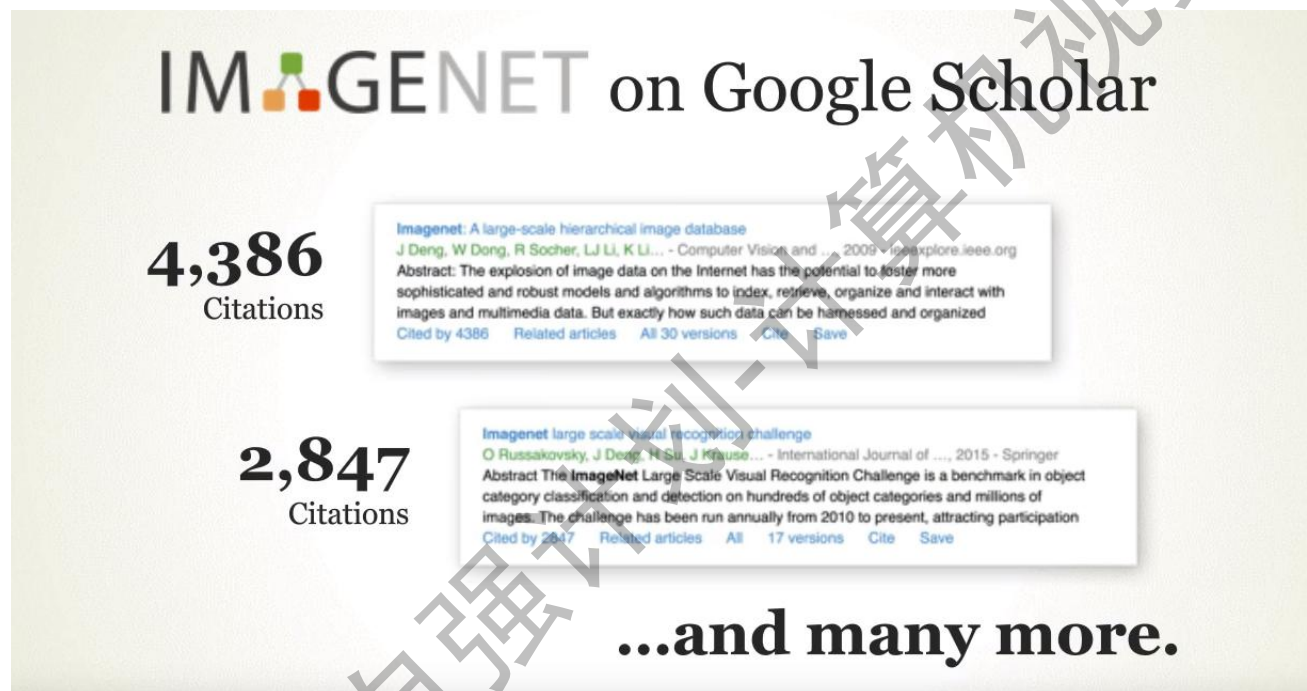


来自167个国家的5万人花了3年的时间，只为了大家能顺利完成这次作业  
是不是想想还有点小激动呢？

## 2.1: ImageNet



如此巨制的影响力自然巨大



**ImageNet: A Large-Scale Hierarchical Image Database.**  
**J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei,**  
**IEEE Computer Vision and Pattern Recognition (CVPR), 2009**

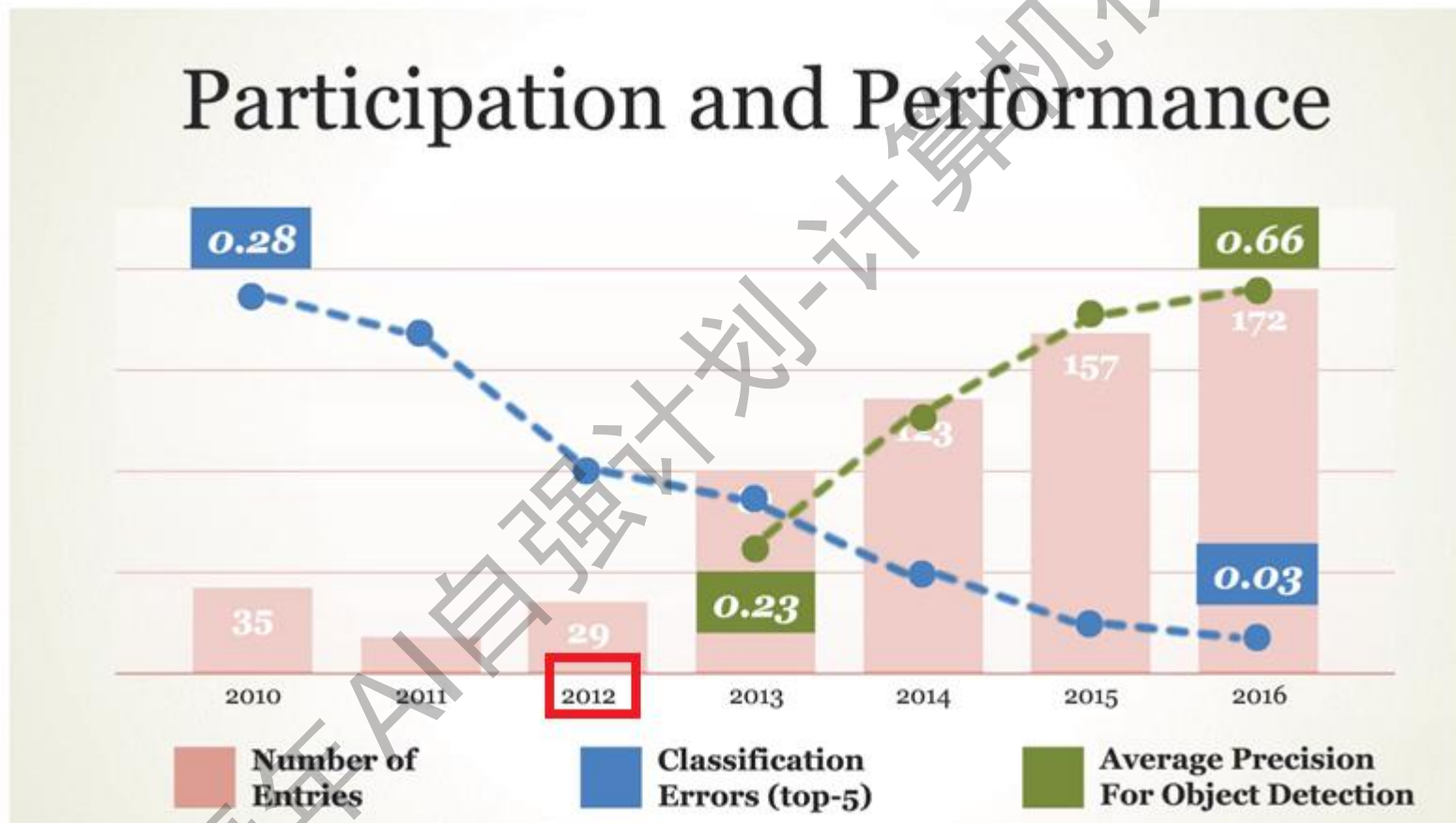
## 2.1: ImageNet



### 相关比赛

#### ImageNet Large Scale Visual Recognition Challenge 2010 (ILSVRC2010)

Held as a "taster competition" in conjunction with PASCAL Visual Object Classes Challenge 2010 (VOC2010).



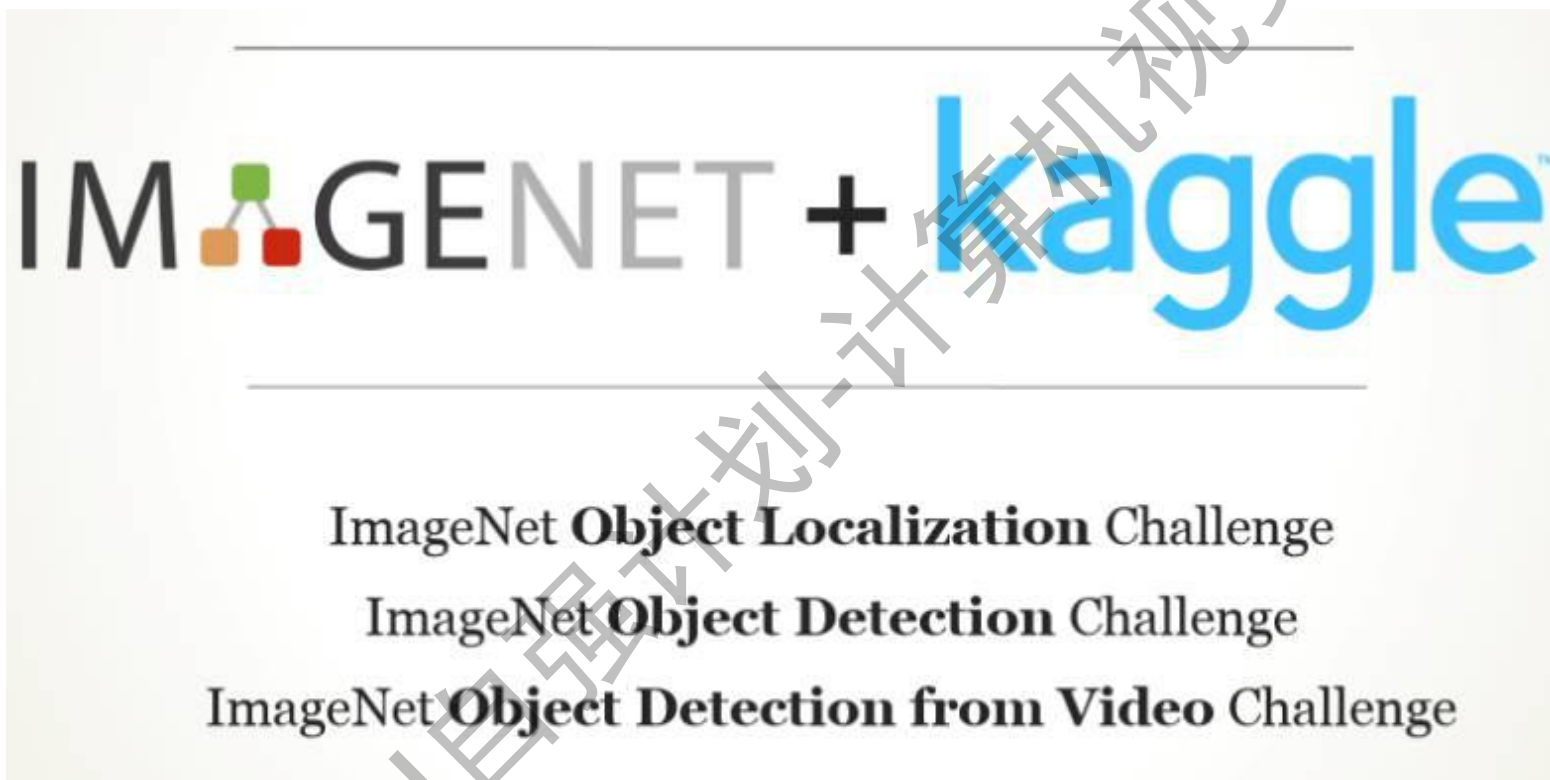
2012年的AlexNet开始，进入深度学习时代



## 2.1: ImageNet



比赛停办



2017年最后一届ImageNet挑战赛上，算法在分类任务上全面超过人类  
随后ImageNet由Kaggle托管，我们在转化任务中也会与Kaggle亲密接触

## 2.2: 本次作业简化后的数据



### 本次作业数据集-可视化



二分类：狗或其他

共有近5万张图片，狗狗2万多张

每张图片分辨率为227\*227

随机挑出5000张作为测试集，剩余训练集

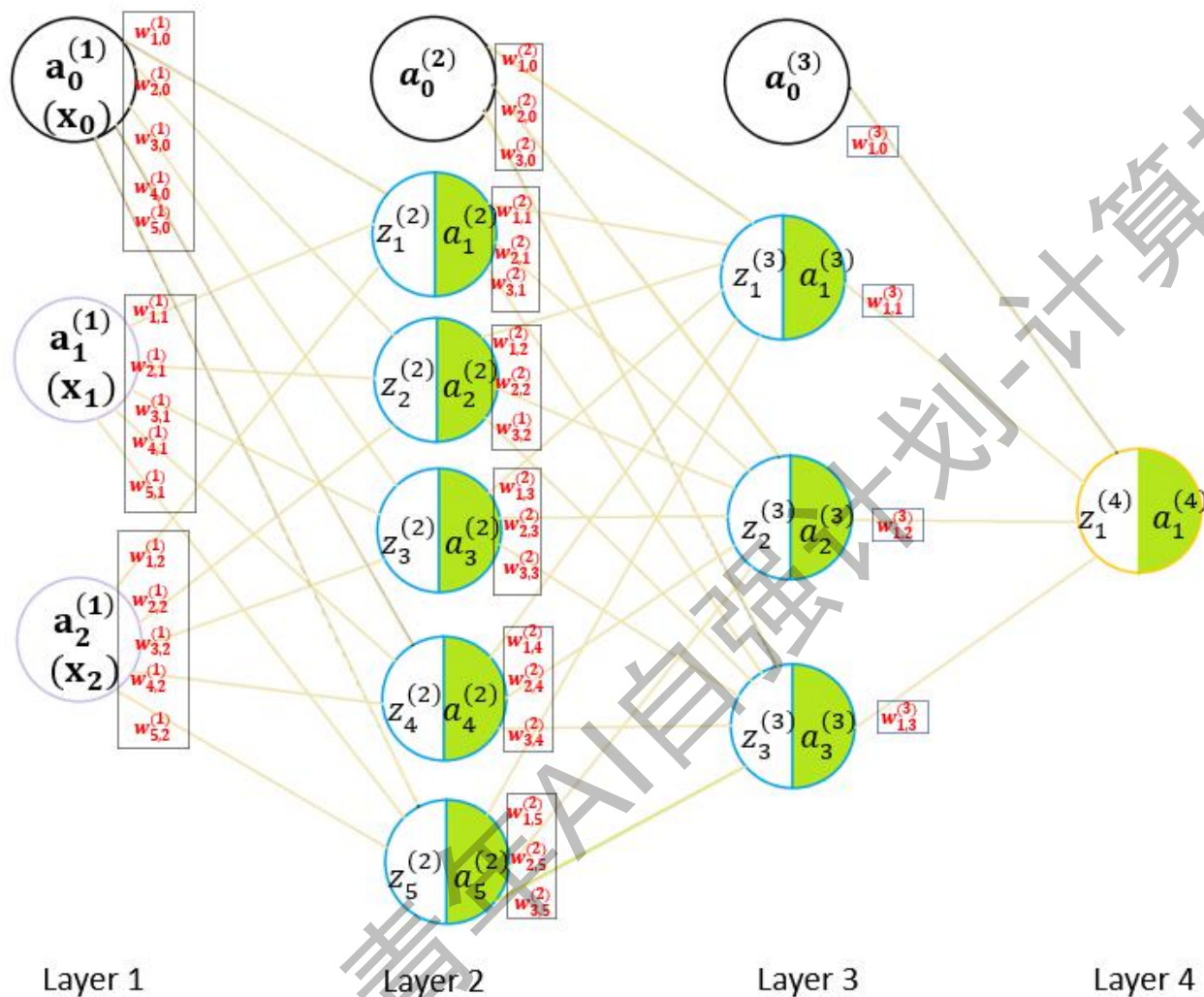
$$X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_{51,529}^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_{51,529}^{(2)} \\ \dots & \dots & \dots & \dots \\ x_1^{(43,555)} & x_2^{(43,555)} & \dots & x_{51,529}^{(43,555)} \end{bmatrix}$$

本次训练集样本-已有质变

### 3: 上一讲知识回顾



一个恰当的错误示范：浅层BP网络拟合本次数据集



前馈传播

$$a^{(1)} = x$$

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

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

$$z^{(3)} = w^{(2)} a^{(2)}$$

$$a^{(3)} = g(z^{(3)})$$

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

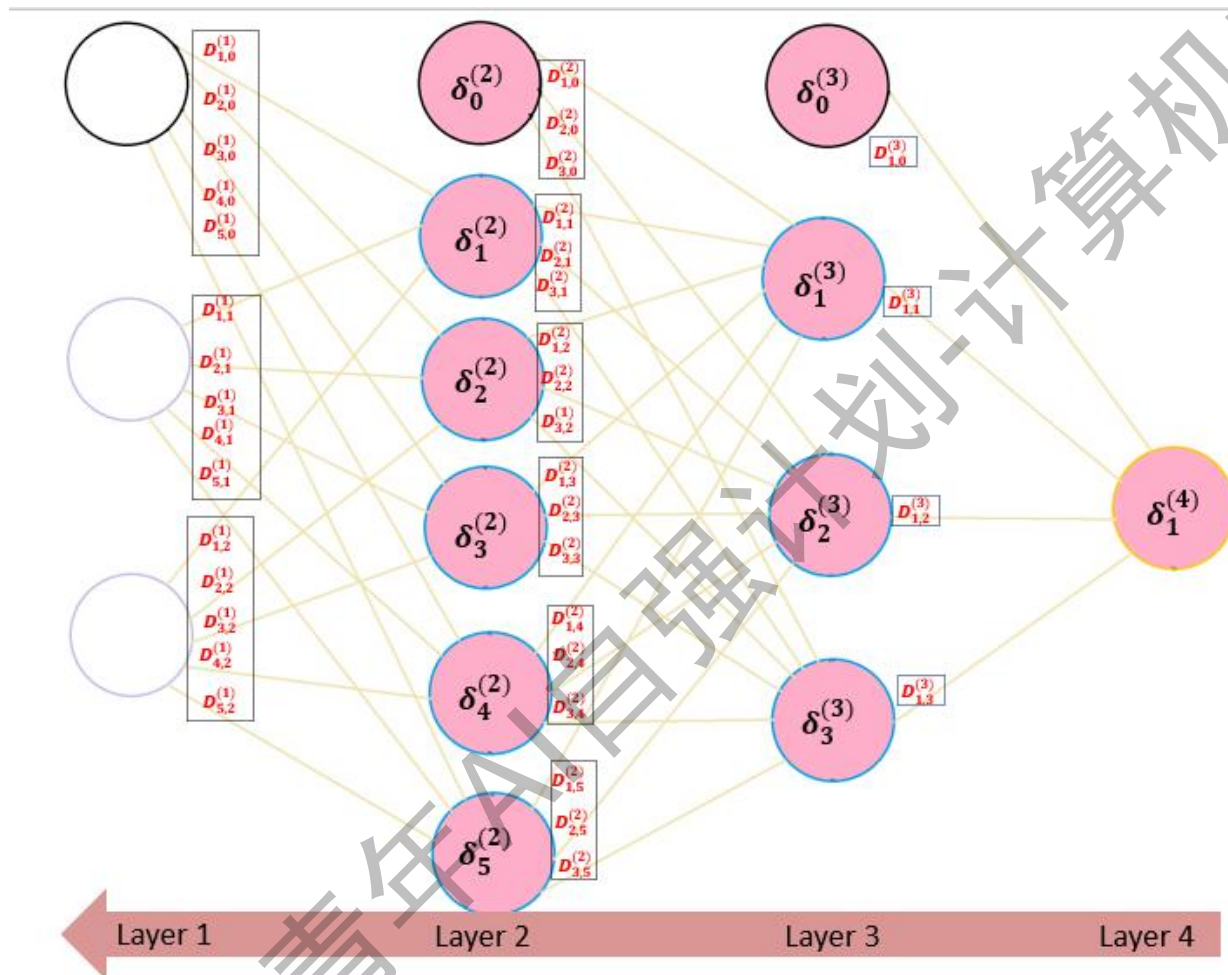
$$a^{(4)} = g(z^{(4)})$$



### 3: 上一讲知识回顾



一个恰当的错误示范：浅层BP网络拟合本次数据集



反向传播

$$\delta^{(4)} = a^{(4)} - y_{\text{target}}$$

$$\delta^{(3)} = (w^{(3)})^T \delta^{(4)} .* g'(z^{(3)})_{\text{elementwise}}$$

$$\text{其中: } g'(z^{(3)}) = a^{(3)} .* (1 - a^{(3)})_{\text{elementwise}}$$

$$\delta^{(2)} = (w^{(2)})^T \delta^{(3)} .* g'(z^{(2)})_{\text{elementwise}}$$

$$\delta^{(l)} = (w^{(l)})^T \delta^{(l+1)} .* g'(z^{(l)})_{\text{elementwise}}$$

$$g'(z^{(l)}) = a^{(l)} .* (1 - a^{(l)})_{\text{elementwise}}$$



### 3: 上一讲知识回顾



一个恰当的错误示范：浅层BP网络拟合本次数据集

梯度递减-更新参数

最终结果

二分类50%准确率重现江湖

求每个样本（共  $m$  个）上的导数并求和。

1、第1个样本上计算：

$$\Delta_{ij}^{(l)} = a_j^{(l)} \delta_i^{(l+1)}$$

2、计算剩余  $m-1$  个样本，并累加

$$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$$

3、累加和过除以  $m$

$$D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)}$$

4、更新参数

$$w_{ij}^{(l)} = w_{ij}^{(l)} - \alpha D_{ij}^{(l)}$$

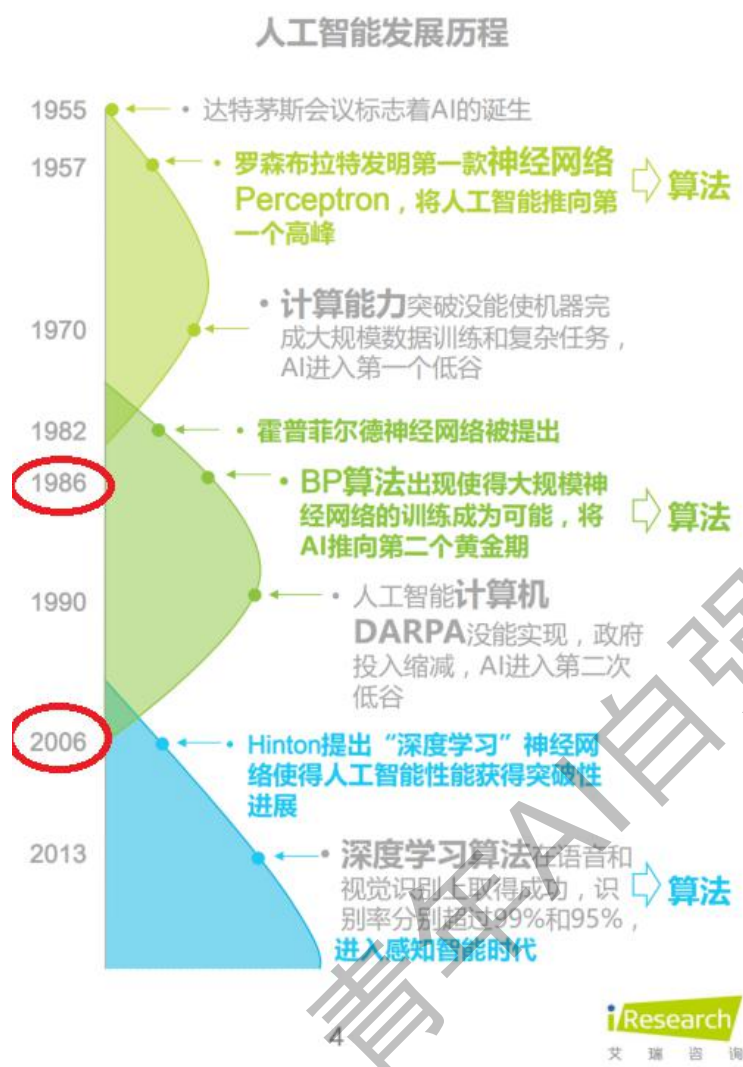
条件设置	模型精度	
	训练集 / %	测试集 / %
输入图像：灰度图 训练数据：37214 测试数据：5000 模型结构：4 层... 227*227, 5, 3, 1 Loss-function: 交叉熵 优化方法: GradientDescentOptimizer Batch Size: 64 Epochs: 10 Lr: 0.001 & learning-rate-decay 初始化: xavier_initializer 激活函数: relu(隐层) & sigmoid(输出) 正则: L2	48.58	50.00

拟合能力不足需要继续加深，可以直接加深吗？

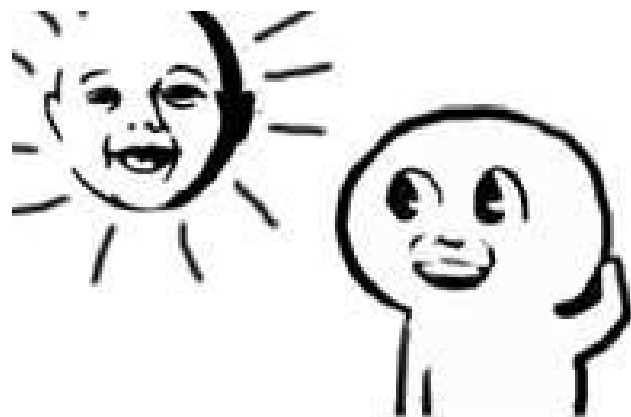
# 4.1、梯度弥散/消失-小实验



一个问题限制了神经网络20年无法变深



1986-2006, 确实是20年  
小提示:  
这个问题会导致权值无法更新



阳光下的懵逼

Mengbi Under the Wuke

Warning: 即将进入本次课程最重要环节

本次课程确实是“0门槛”，但有一个隐藏条件：

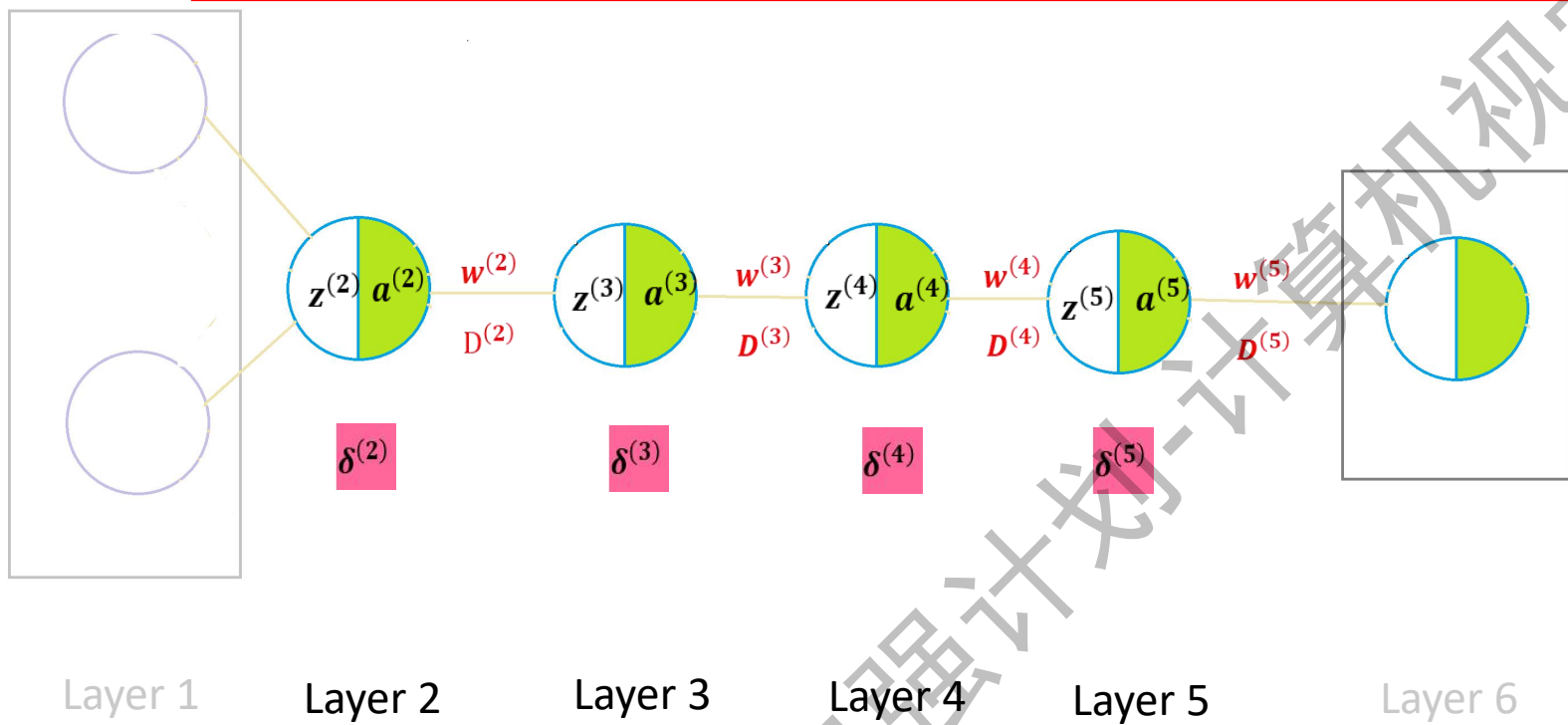
从第一讲开始看

如果后续内容令你懵逼，不要害怕

请在学堂在线搜索“青年AI”快速回顾

# 4. 1、梯度弥散/消失-小实验

## 简化问题-网络结构简化



- 隐藏层多了1倍
  - 不需要矩阵化表示
  - 不需要  $\Delta_{ij}^{(l)}$
- 没有下标
  - \*符号没有了
  - alpha=1
- 非关键项没画出
  - 不用转置了

前馈传播

$$a^{(1)} = x$$

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

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

反向传播

$$\delta^{(6)} = a^{(6)} - y$$

$$\delta^{(l)} = w^{(l)} \delta^{(l+1)} g'(z^{(l)})$$

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

梯度递减

$$D^{(l)} = a^{(l)} \delta^{(l+1)}$$

$$w^{(l)} = w^{(l)} - D^{(l)}$$



# 4. 1、梯度弥散/消失-小实验



## 输入数据简化 & 模拟规则

输入数据

	身高 (m) X1		月薪 (元) X2		标注
	原始值	归一化后	原始值	归一化后	
1	1.95252	0.427262	8962.728	0.264914	1
2	1.753169	0.022738	6875.233	0.073273	1
3	1.86274	0.24508	11199.48	0.470257	1
4	1.752128	0.020626	9702.441	0.332822	1
5	1.712846	-0.05909	9525.321	0.316562	1

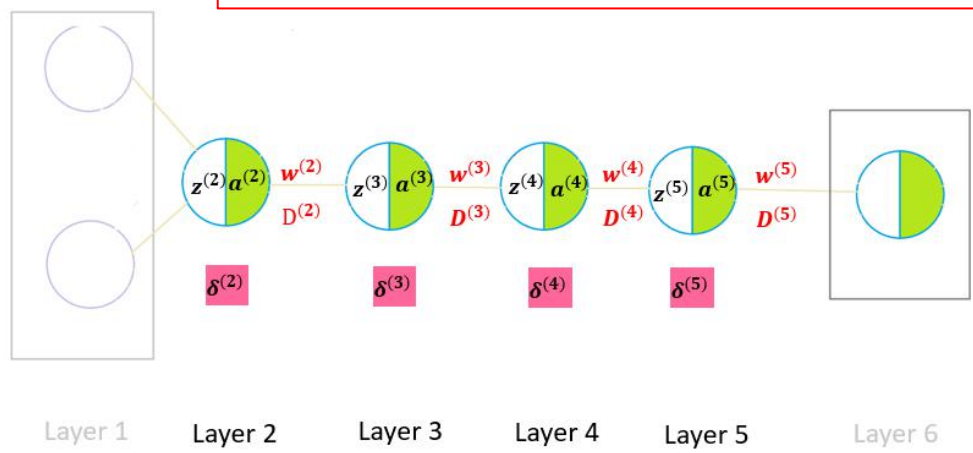
- 采用第二讲数据
- 每次迭代输入1个样本
- 一共只迭代5次

模拟过程

- 1、初始化权值；
- 2、输入样本并计算前馈传播；
- 3、通过前馈传播计算所有误差；
- 4、通过误差计算每一层权值的导数；
- 5、模拟权值更新。

# 4. 1、梯度弥散/消失-小实验

开始模拟



## 1、权值初始化

- 邀请4位同学上台代表4个权值
- 明确初始点与终点
- 按照计算结果向终点更新

## 2、前馈传播计算

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

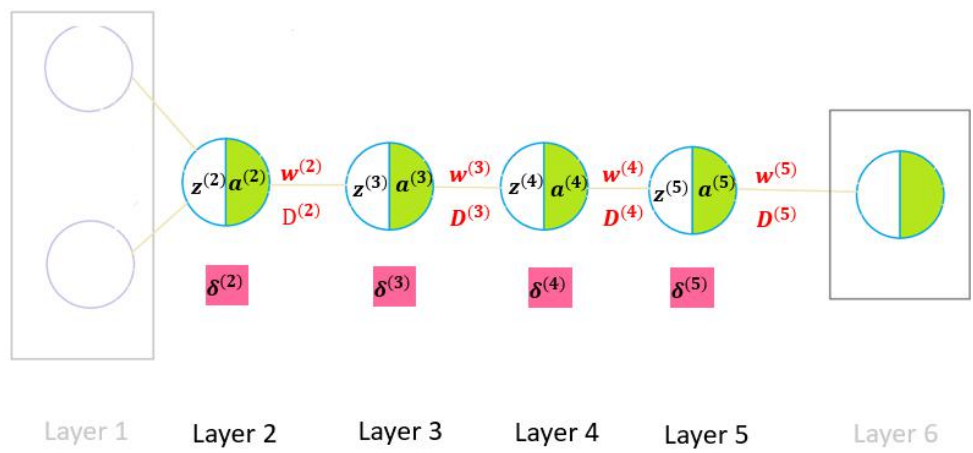
a1	0.42726183643262816 0.26491365101710074
a2	0.844510043
a3	0.86348123
a4	0.865702198
a5	0.865960202
a6	0.865990147

## 3、误差计算

$$\delta^{(6)} = a^{(6)} - y$$
$$\delta^{(l)} = w^{(l)} \delta^{(l+1)} g'(z^{(l)})$$
$$g'(z^{(l)}) = a^{(l)}(1 - a^{(l)})$$

delta6	-0.134009853
delta5	-0.015554943
delta4	-0.001808447
delta3	-2.13E-04
delta2	-2.80E-05

# 4.1、梯度弥散/消失-小实验



## 4、梯度计算

$$D^{(5)} = a^{(5)} \delta^{(6)}$$

$$D^{(5)} = a^{(5)} (a^{(6)} - y)$$

$$D^{(4)} = a^{(4)} w^{(5)} g'(z^{(5)}) \delta^{(6)}$$

$$D^{(4)} = a^{(4)} w^{(5)} a^{(5)} (1 - a^{(5)}) (a^{(6)} - y)$$

$$D^{(3)} = a^{(3)} w^{(4)} g'(z^{(4)}) w^{(5)} g'(z^{(5)}) \delta^{(6)}$$

$$D^{(3)} = a^{(3)} w^{(4)} w^{(5)} a^{(4)} (1 - a^{(4)}) a^{(5)} (1 - a^{(5)}) (a^{(6)} - y)$$

$$D^{(2)} = a^{(2)} w^{(3)} g'(z^{(3)}) w^{(4)} g'(z^{(4)}) w^{(5)} g'(z^{(5)}) \delta^{(6)}$$

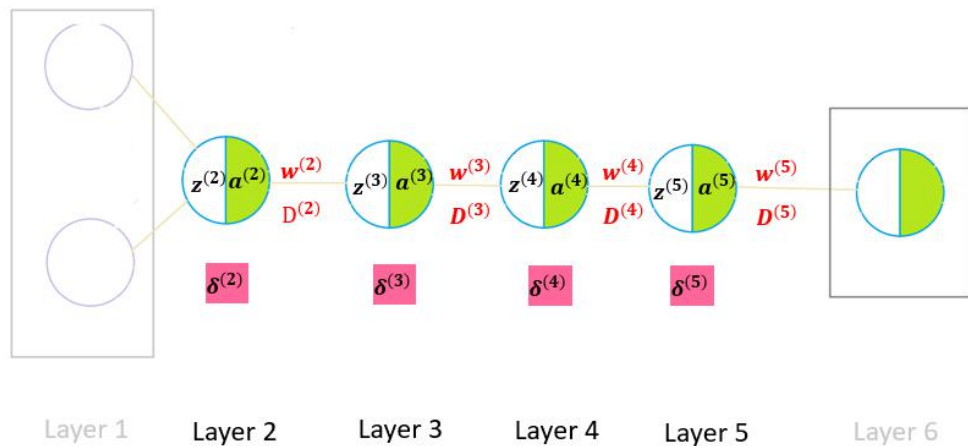
$$D^{(2)} = a^{(2)} w^{(3)} w^{(4)} w^{(5)} a^{(3)} (1 - a^{(3)}) a^{(4)} (1 - a^{(4)}) a^{(5)} (1 - a^{(5)}) (a^{(6)} - y)$$

$$D^{(l)} = a^{(l)} \delta^{(l+1)}$$

$$\delta^{(6)} = a^{(6)} - y$$
$$\delta^{(l)} = w^{(l)} \delta^{(l+1)} g'(z^{(l)})$$
$$g'(z^{(l)}) = a^{(l)} (1 - a^{(l)})$$

# 4.1、梯度弥散/消失-小实验

## 持续迭代模拟



## 5、更新规则

$$w^{(l)} = w^{(l)} - D^{(l)}$$

- 要直观的反映到现实世界
- 将计算值放大60倍为步数
- 每一步的大小为1块地砖
- 按照计算结果向终点更新

	1-计算值	1-实际步数	2-计算值	2-实际步数	3-计算值	3-实际步数	4-计算值	4-实际步数	5-计算值	5-实际步数
D5	-0.12	-7.2	-0.094	-5.64	-0.081	-4.86	-0.067	-4.02	-0.068	-4.08
D4	-0.013	-0.78	-0.0119	-0.714	-0.01	-0.6	-0.0097	-0.582	-0.0088	-0.528
D3	-0.0016	-0.096	-0.0014	-0.084	-0.0013	-0.078	-0.0011	-0.066	-0.001	-0.063
D2	-0.00018	-0.0108	-0.00016	-0.0096	-0.00015	-0.009	-0.00013	-0.0078	-0.00012	-0.0072

采访一下各位权值的感受

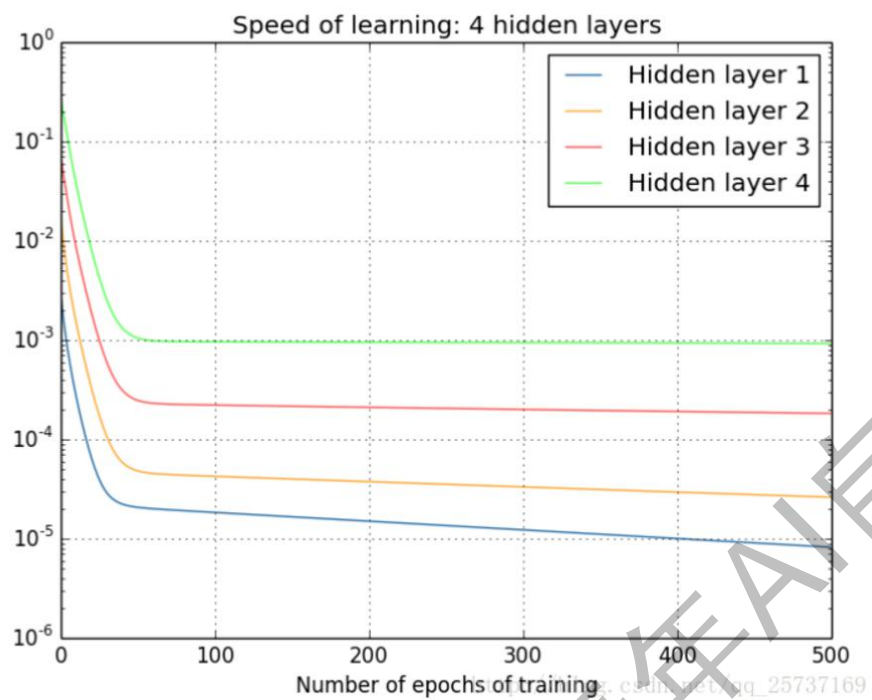
距离输出层越远，梯度越小，权值更新的越慢



# 4. 1、梯度弥散/消失-小实验

## 小结

	1-计算值	1-实际步数	2-计算值	2-实际步数	3-计算值	3-实际步数	4-计算值	4-实际步数	5-计算值	5-实际步数
D5	-0.12	-7.2	-0.094	-5.64	-0.081	-4.86	-0.067	-4.02	-0.068	-4.08
D4	-0.013	-0.78	-0.0119	-0.714	-0.01	-0.6	-0.0097	-0.582	-0.0088	-0.528
D3	-0.0016	-0.096	-0.0014	-0.084	-0.0013	-0.078	-0.0011	-0.066	-0.001	-0.063
D2	-0.00018	-0.0108	-0.00016	-0.0096	-0.00015	-0.009	-0.00013	-0.0078	-0.00012	-0.0072



- 横轴是迭代次数，越往右侧越大
- 纵轴是每一次迭代更新量D的取值大小，越往下越小，取值越小代表更新越慢
- 4条不同颜色的曲线，代表了4个不同隐藏层
- 随着迭代次数的增加，每一层的更新速度都在减小
- 每层之间更新速度相差1个数量级，距离输出层越远更新越慢

如果隐藏层继续增加会怎样？

# 4.1、梯度弥散/消失-小实验

## 详细过程数据

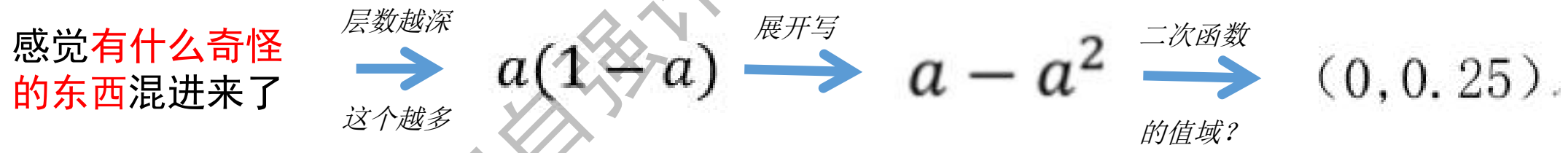
lr=1	参数	样本1	样本2	样本3	样本4	样本5
前向	a1	0.42726183643262816 0.26491365101710074	0.022738348122339595 0.07327263834999487	0.24508024911229243 0.47025690528852293	0.020625885863829768 0.3328224388374417	-0.0590857100936783 0.3165621031353404
	a2	0.844510043	0.749517347	0.847536926	0.794708552	0.778612798
	a3	0.86348123	0.851935833	0.863920453	0.857622732	0.855677899
	a4	0.865702198	0.864721822	0.866449181	0.866012666	0.866055607
	a5	0.865960202	0.868974414	0.871897244	0.87424135	0.876374878
	a6	0.865990147	0.89127452	0.908727055	0.921425419	0.93108018
	w1	1 1	1.000011960584024 1.0000074158787702	1.00001284411393 1.0000102629882888	1.0000183137030152 1.0000207579671578	1.0000188668444303 1.0000296835413627
	w2	1	1.000180035	1.000335133	1.000481464	1.000612034
	w3	1	1.00156156	1.002956955	1.00422199	1.00537111
	w4	1	1.013465948	1.025412799	1.0361054	1.045757047
	w5	1	1.1160472	1.210526859	1.290107489	1.358800636
后向	delta6	-0.134009853	-0.10872548	-0.091272945	-0.078574581	-0.06891982
	delta5	-0.015554943	-0.01381583	-0.012340714	-0.011144926	-0.010146039
	delta4	-0.001808447	-0.001637911	-0.001464295	-0.00133989	-0.001230829
	delta3	-2.13E-04	-0.000206931	-0.000172654	-1.64E-04	-0.000152815
	delta2	-2.80E-05	-3.89E-05	-2.23E-05	-2.68E-05	-2.64E-05
	△5	-0.1160472	-0.09447966	-0.079580629	-0.068693148	-0.060399599
	△4	-0.013465948	-0.01194685	-0.010692601	-0.009651647	-0.008787034
	△3	-0.00156156	-0.001395395	-0.001265035	-0.00114912	-0.001053193
	△2	-1.80E-04	-0.000155098	-0.000146331	-1.31E-04	-0.000118984
	△1	-1.1960584024080716e-05 -7.41587877019681e-06	-8.835299073628376e-07 -2.8471095184789783e-06	-5.469589083618191e-06 -1.0494978869079214e-05	-5.53141415172832e-07 -8.925574204919652e-06	1.5573614911281898e-06 -8.343838606524198e-06

# 4. 2、梯度弥散/消失 – 揭示原因

回顾梯度计算过程

$$D^{(2)} = a^{(2)} w^{(3)} g'(z^{(3)}) w^{(4)} g'(z^{(4)}) w^{(5)} g'(z^{(5)}) \delta^{(6)}$$
$$D^{(2)} = a^{(2)} w^{(3)} w^{(4)} w^{(5)} a^{(3)}(1 - a^{(3)}) a^{(4)}(1 - a^{(4)}) a^{(5)}(1 - a^{(5)}) (a^{(6)} - y)$$

梯度消失的秘密

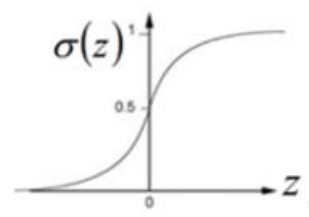
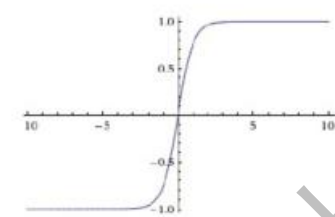
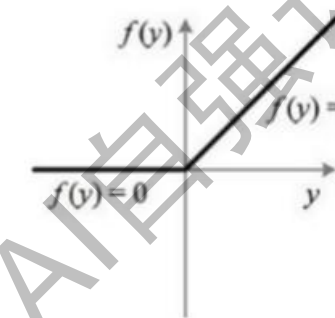


由于激活函数sigmoid天然的缺陷  
导致网络每加深一层，梯度会缩小至少4倍（一般为10倍）



# 5.1、梯度消失/爆炸解决方法

激活函数有问题怎么破？

编号	激活函数名称	函数公式	函数图形	函数求导公式	备注
1	Sigmoid	公式： $\sigma(z) = \frac{1}{1 + e^{-z}}$ 值域：(0, 1)		公式： $\sigma'(z) = \sigma(z) * (1 - \sigma(z))$ 值域：(0, 0.25)	优点：可以将函数值压缩到(0,1)之间，提高模型的稳定性。 不足：函数导数最大值为 0.25，使得反向传播时，梯度传播逐层递减，浅层参数几乎更新不到，即产生梯度消失现象；且它是非 0 均值分布。
2	Tanh	公式： $\tanh(x) = 2\sigma(2x) - 1$ 值域：(-1, 1)		公式： $\tanh'(x) = 1 - \tanh^2(x)$ 值域：(0, 1)	优点：与 sigmoid 相比，是 tanh 是 0 均值分布，导数值域更大，缓解了反向传播时梯度消失的问题。 不足：由于反向传播时，梯度仍然是有损传播，因此，模型层数较多时，仍然存在梯度消失问题。
3	Relu	公式： $\text{relu}(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$ 值域：[0, +∞)		公式： $\text{relu}'(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$ 值域：{0, 1}	优点：与 sigmoid\tanh 相比，relu 作为激活函数的模型，反向传播时梯度可以无损失传播，解决了梯度消失问题。 不足：由于 relu 值是非负的，当较大梯度流过神经元时，会导致神经元 “dead”，即不会再被激活。

关注激活函数导数的值域

目前一般采用Relu作为激活函数



# 5.2、梯度爆炸

ReLU也有问题：梯度爆炸

再次回顾梯度计算过程

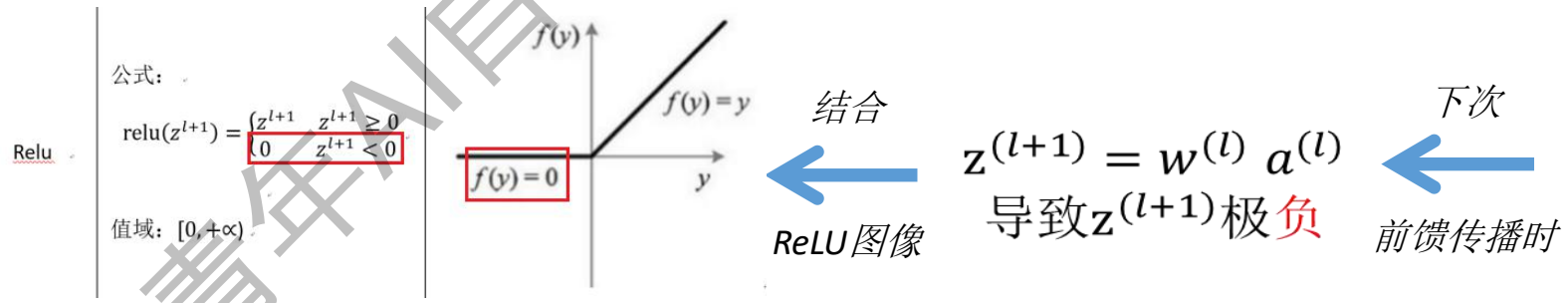
$$D^{(2)} = a^{(2)} \boxed{w^{(3)} w^{(4)} w^{(5)}} a^{(3)} (1 - a^{(3)}) a^{(4)} (1 - a^{(4)}) a^{(5)} (1 - a^{(5)}) (a^{(6)} - y)$$

梯度爆炸的秘密

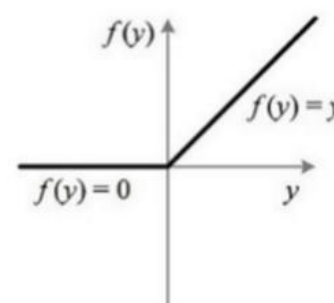
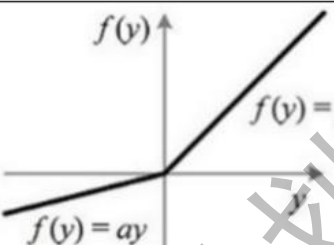
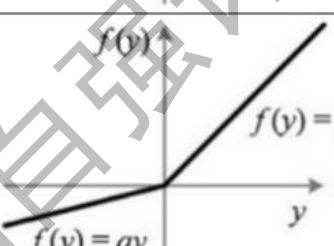
感觉又有什么奇怪的东西混进来了



$z^{(l+1)}$ 神经元  
被“炸死了”  
( $a^{(l+1)}$ 恒为0)

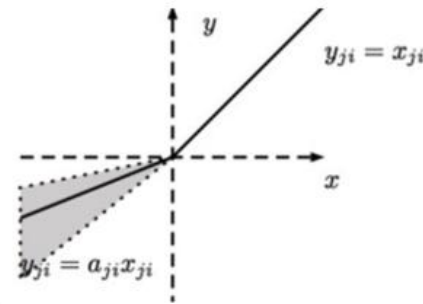
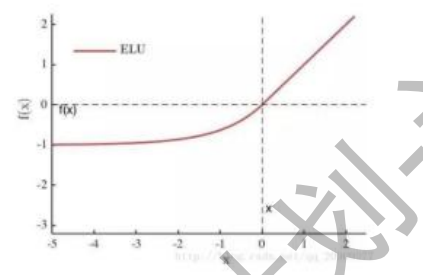
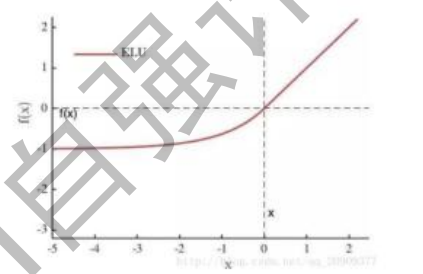


# 5. 3、ReLU家族简介

编号	激活函数名称	函数公式	函数图形	函数求导公式	备注
1	Relu	公式： $\text{relu}(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$ 值域： $[0, +\infty)$		公式： $\text{relu}'(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$ 值域：1	优点：与 sigmoid/tanh 相比，relu 作为激活函数的模型，反向传播时梯度可以无损失传播，解决了梯度消失问题。 不足：由于 relu 值是非负的，当较大梯度流过神经元时，会导致神经元“dead”，即不会再被激活。
2	Leaky ReLU “L-Relu”	公式： $\text{L\_Relu}(x) = \begin{cases} x & x \geq 0 \\ \alpha x & x < 0 \end{cases}$ 值域： $[-\infty, +\infty)$		公式： $\text{L\_Relu}'(x) = \begin{cases} 1 & x \geq 0 \\ \alpha & x < 0 \end{cases}$ 值域： $\alpha$ or 1	优点：与 relu 相比，L-Relu 保留了负半轴的值，缓解了 relu 中神经元“dead”的问题。 不足：L-Relu 负半轴的保留参数 $\alpha$ 是经验值设定的；另外， $\alpha$ 通常是一个很小的值，如 0.001，负半轴信息保留的不多。
3	Parametric ReLU “P-ReLU”	公式： $\text{P\_Relu}(x) = \begin{cases} x & x \geq 0 \\ \alpha x & x < 0 \end{cases}$ 值域： $[-\infty, +\infty)$		公式： $\text{P\_Relu}'(x) = \begin{cases} 1 & x \geq 0 \\ \alpha & x < 0 \end{cases}$ 值域： $\alpha$ or 1	优点：与 L-Relu 相比，P-Relu 负半轴的保留参数 $\alpha$ 是训练中不断更新的，而非经验值设定。 不足：负半轴值域是 $[-\infty, 0)$ ，不能保证是一个噪声稳定的去激活状态。

激活函数看两点：原公式和求导公式

# 5. 3、ReLU家族简介

4	Randomized ReLU “R-ReLU”	<p>公式:</p> $R\_Relu(x) = \begin{cases} x & x \geq 0 \\ \alpha_{j,i}x & x < 0 \end{cases}$ <p>值域: <math>[-\alpha, +\alpha)</math></p>	 <p>公式:</p> $R\_Relu'(x) = \begin{cases} 1 & x \geq 0 \\ \alpha_{j,i} & x < 0 \end{cases}$ <p>值域: <math>\alpha_{j,i}</math> or 1</p>	<p>优点: 与 L-Relu 相比, R-Relu 负半轴的保留参数 <math>\alpha</math> 是在随机分布中选取的, 而非经验值设定。</p> <p>不足: 负半轴值域是 <math>[-\alpha, 0)</math>, 不能保证是一个噪声稳定的去激活状态。</p>
5	ELU	<p>公式:</p> $ELU(x) = \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$ <p>值域: <math>(-\alpha, +\alpha)</math></p>	 <p>公式:</p> $ELU'(x) = \begin{cases} 1 & x \geq 0 \\ ELU(x) + \alpha & x < 0 \end{cases}$ <p>值域: <math>(0, \alpha)</math>, and 1</p>	<p>优点: 与 L-Relu/ P-Relu/R-Relu 相比, ELU 负半轴的软饱和, 使得对输入变化和噪声更鲁棒; 另外, 输出均值接近于 0, 且利用了负半轴信息, 收敛速度更快。</p> <p>不足: 存在指数运算。</p>
6	SELU	<p>公式:</p> $SELU(x) = \lambda \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$ <p>值域: <math>(-\lambda\alpha, +\alpha)</math></p>	 <p>公式:</p> $SELU'(x) = \lambda \begin{cases} 1 & x \geq 0 \\ SELU(x) + \alpha & x < 0 \end{cases}$ <p>值域: <math>(0, \lambda\alpha)</math>, and <math>\lambda</math></p>	<p>优点: 与 ELU 相比, SELU 承接了 ELU 的所有优势, 公式上只是多乘以了 <math>\lambda(\lambda &gt; 1)</math>, 它的输出可以自动地收敛到零均值和单位方差, 达到 <u>batchnorm</u> 的效果。</p> <p>不足: 存在指数运算。</p>

实际情况是TF仍然默认采用原版ReLU

# 5.4、通用解决方法 - 初始化

我是来“自杀”凑数的  
请忽略我

七分天注定  
三分靠打拼

我们  
都是正规军  
我的命运我  
做主

编号	初始化方法	权重计算	适用的激活函数	备注
1	Zero Initialization	$W_{i,j} = 0$	Sigmoid	0 值初始化，在包含隐层的深度学习中会导致训练失败，不推荐
2	Random Initialization	$W_{i,j} = randn()$ 其中， $randn()$ 为随机函数	任何激活函数	若选用的随机分布方法（正态分布、均匀分布等）不合适，会导致优化陷入困境。
3	Xavier Initialization	$W_{i,j} \sim N\left(0, \sqrt{\frac{2}{n_{in} + n_{out}}}\right)$ $W_{i,j} \sim U\left(-\sqrt{\frac{6}{n_{in} + n_{out}}}, \sqrt{\frac{6}{n_{in} + n_{out}}}\right)$ 其中， $N()$ 表示正态分布； $U()$ 表示均匀分布； $n_{in}$ 表示前一层神经元的个数； $n_{out}$ 表示当前层神经元个数	Tanh (强烈建议)	2010 年，Xavier Glorot, Yoshua Bengio 提出的。基于 tanh 有严格的理论推导。 论文：Understanding the difficulty of training deep feedforward neural network 论文链接： <a href="http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf">http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf</a>
4	MSRA Initialization	$W_{i,j} \sim N\left(0, \sqrt{\frac{2}{n_{in}}}\right)$ $W_{i,j} \sim U\left(-\sqrt{\frac{6}{n_{in}}}, \sqrt{\frac{6}{n_{in}}}\right)$ 其中， $N()$ 表示正态分布； $U()$ 表示均匀分布； $n_{in}$ 表示前一层神经元的个数； $n_{out}$ 表示当前层神经元个数	Relu 家族 (强烈建议)	2015 年，何凯明提出的。基于 relu 有严格的理论推导。 论文：Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification 论文链接： <a href="https://arxiv.org/pdf/1502.01852.pdf">https://arxiv.org/pdf/1502.01852.pdf</a>
5	Pre-train Initialization	$W_{i,j} = model_{other}(W_{i,j})$ 其中， $model_{other}(W_{i,j})$ 表示其他已有模型对应的 $W_{i,j}$	任何激活函数	很好的训练策略，推荐

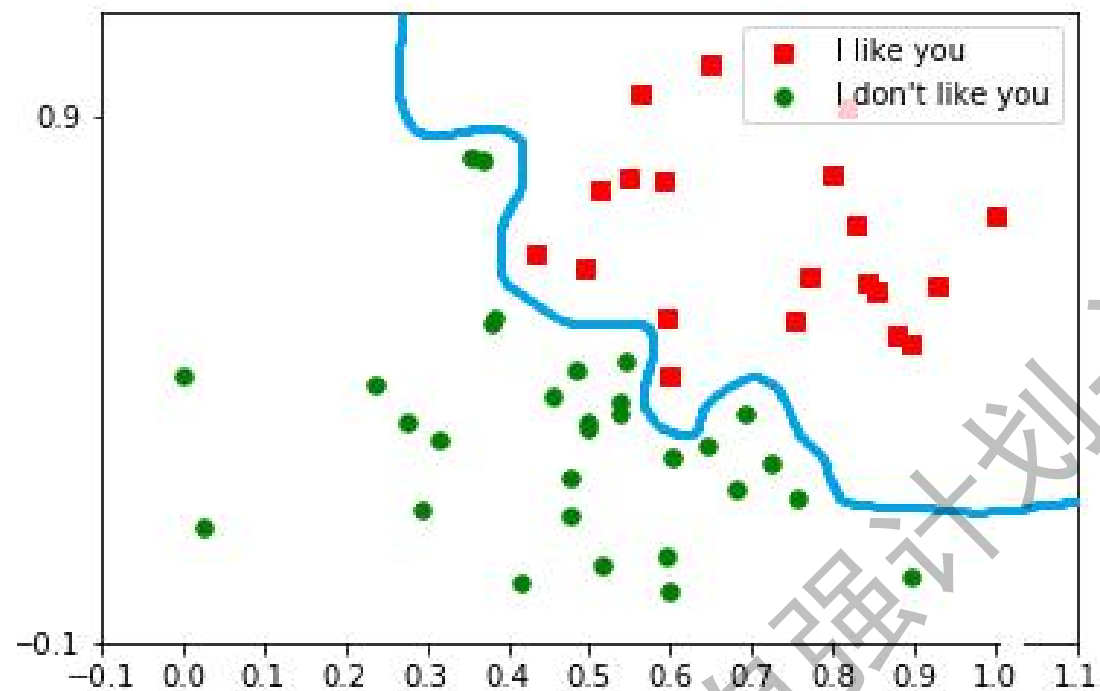
我“开挂”会随便告诉别人吗？



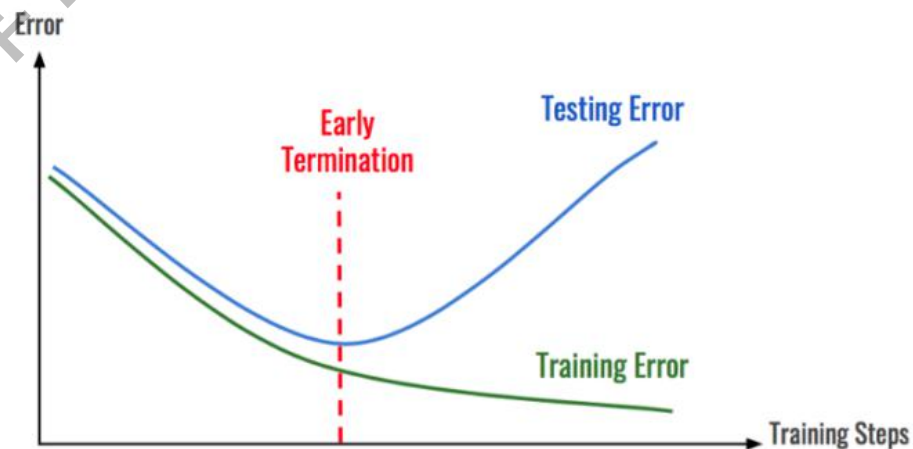
## 6.1、过拟合问题 early-stopping



啥是过拟合？



过拟合的直观体现



过拟合会引起的loss变化  
&解决办法

Early-stopping  $\approx$  找准时机“踩刹车”

## 6.2、lr-decay



车跑的太快了（ $\alpha$ 太大了），怎么办？

公式： $\alpha$  随着迭代次数增加而逐步降低

$$\alpha = \frac{1}{1 + \text{decay.rate} * \text{iteration}} \alpha^0$$

$$\alpha = \text{decay.rate}^{\text{iteration}} \alpha^0$$

TF自动实现

```
learning_rate = tf.train.exponential_decay(lr, global_step, decay_steps=sample_size/batch_size, decay_rate=0.98)
```

直观认识

前期大步跑，后期小步走

## 6.3、数据增强



为什么我们需要尽可能多的数据？--DNN+“全集”



Color Jittering  
亮度、饱和度、对比度等



Random Scale



Random Crop



## 6.3、数据增强



Horizontal/Vertical Flip



Rotation/Reflection



Noise

其实最有效的还是多找“真实数据”



## 6. 4、其他方法



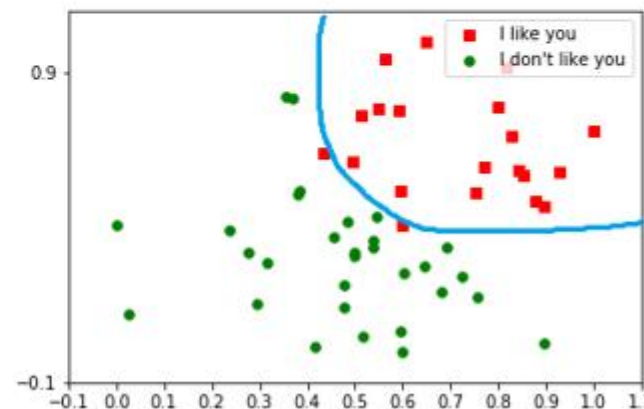
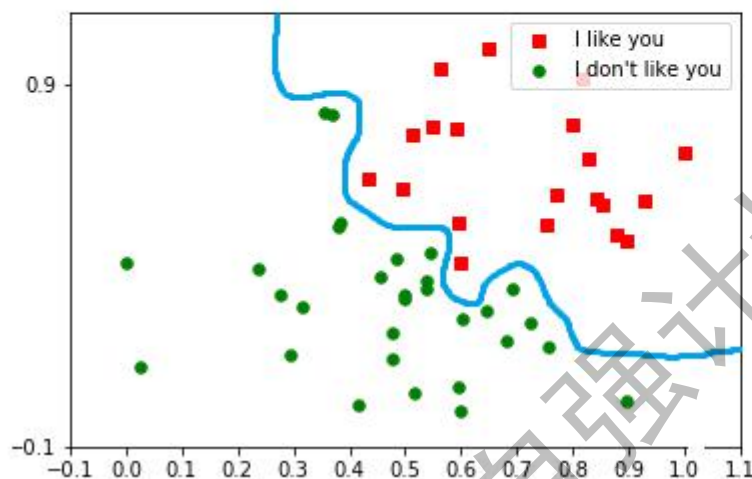
L2正则，使用较多

$$J(w) = J_0(w) + \frac{1}{2m} \lambda \sum_{j=1}^n w_j^2$$

$\lambda$ 的本质是惩罚 $w$   
即 $\lambda$ 的取值越大， $w$ 的取值越小

$$J(w) = J_0(w) + \frac{1}{2m} \lambda \sum_{j=1}^n |w_j|$$

L1正则



另有随AlexNet一起提出的drop-out，将在下次课随AlexNet一起介绍

## 7.1、处理大数据的小技巧 mini-batch



$$X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_{51,529}^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_{51,529}^{(2)} \\ \dots & \dots & \dots & \dots \\ x_1^{(43,555)} & x_2^{(43,555)} & \dots & x_{51,529}^{(43,555)} \end{bmatrix}$$

假设有 $m$ 个样本  
(此处 $m=43,555$ )

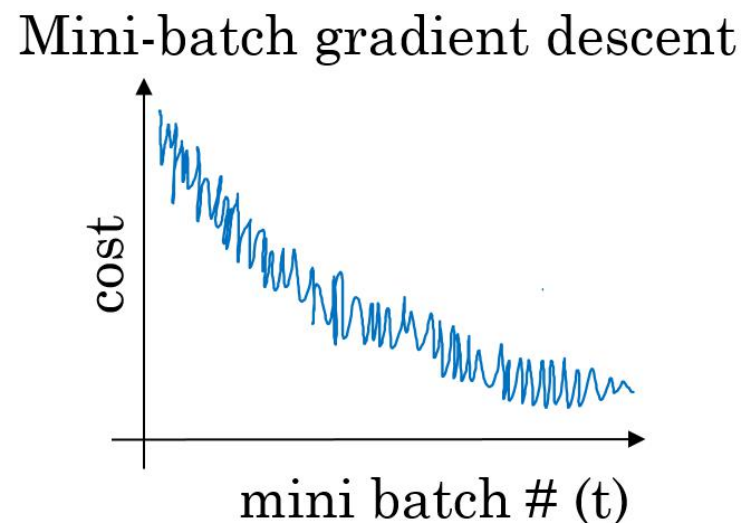
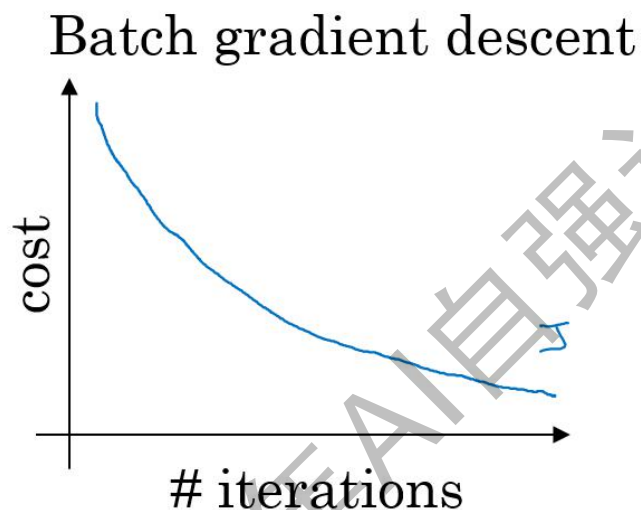
样本太多了  
一次装不进内存

$$X = \begin{bmatrix} x^{\{1\}} \\ x^{\{2\}} \\ \dots \\ x^{\{\frac{n}{m}\}} \end{bmatrix}$$

切分成好多小包  
每个小包 $n$ 个样本

一次丢一个  
mini-batch给模型迭代

$\frac{n}{m}$ 个样本迭代完一轮  
是一个epoch



Mini-batch是否有效

## 7.1、处理大数据的小技巧 mini-batch



清华大学  
Tsinghua University

数据科学研究院  
Institute for Data Science

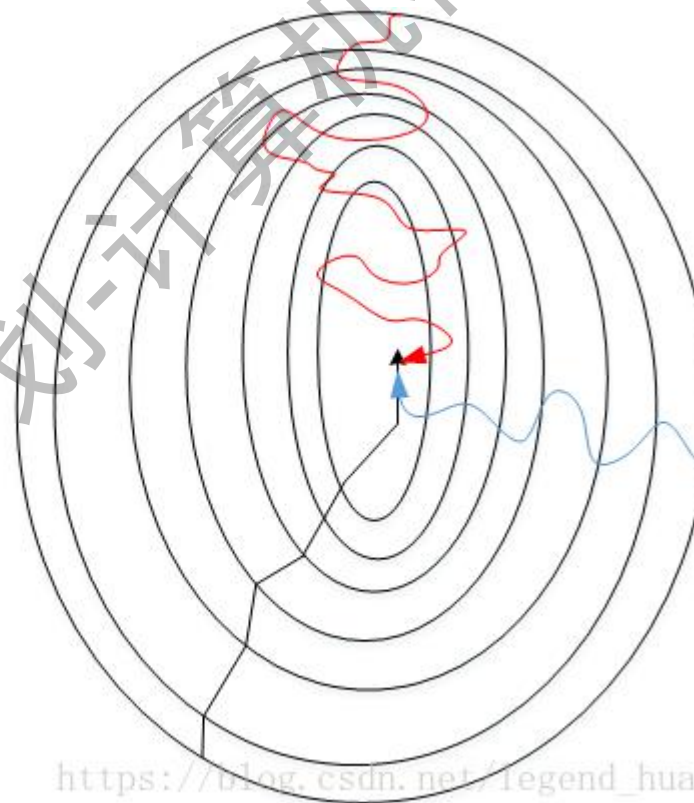
Min-batch的特例

随机梯度递减SGD  
(Stochastic Gradient Decent)

Mini-batch gradient descent

BGD  
(Batch Gradient Descent)

batch\_size: 1  
batch\_size: mini\_batch  
batch\_size: full\_batch



## 7.2、batch-norm



Mini-batch导致训练震荡，是否有解决办法？

$$D^{(2)} = a^{(2)} w^{(3)} g'(\boxed{z^{(3)}}) \quad w^{(4)} g'(\boxed{z^{(4)}}) \quad w^{(5)} g'(\boxed{z^{(5)}}) \quad \delta^{(6)}$$

Loss震荡意味着  
权值更新较大

回顾  
梯度公式

$z^{(l)}$ 代表了输入信号  
对权值更新影响很大

如果采用  
mini-batch

每一次输入的  
样本都不相同

$z^{(l)}$ 的取值变化很大

权值更新剧烈

解决 办法

对每一层的 $z^{(l)}$ 进行归一化



## 7.2、batch-norm



### Batch-norm 说明

假设第 $l$ 层共有 $m$ 个神经元  
每个神经元的 $z^{(i)}$  ( $i=1,2,\dots,m$ ) 经过batch-norm处理后的结果 $\tilde{z}^{(i)}$ , 则:

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

- $\gamma$ 和 $\beta$ 为新的超参, 其本质是改变 $\tilde{z}^{(i)}$ 的分布
- $\gamma$ 和 $\beta$ 也仿照权值, 由梯度更新
- 引入batch-norm后,  $\beta$ 可以替代原有的偏置 $b$
- 在进行测试时,  $\mu$ 以及 $\sigma$ 有评估方法, 一般由TF自行评估 (full-batch或者最近的若干个mini-batch)

注意采用mini-batch时最好配套batch-norm

# 8.1：尾声-训练结果



条件设置	模型精度	
	训练集· /%	测试集· /%
输入图像：灰度图 训练数据：37214 测试数据：5000 模型层数：4 Loss-function：交叉熵 Batch Size：128 Epoches：10 Lr：0.1·&&·learning·rate·decay· 初始化：xavier_initializer 激活函数：relu(隐层)·&&·softmax(输出)	92.43	93.94

加深  
模型层数

正确的  
激活函数

良好的  
初始化

足够的  
数据

Mini-batch  
&Batch-  
norm

避免  
过拟合

## 8.2：尾声-作业说明



作业1：浅层神经网络拟合 ImageNet；

知识回顾

作业2：“梯度消失”小实验

初步了解调参辅助工具

作业3：DNN拟合 ImageNet

感受DNN的力量



- 1、姚超被老板捉走开组会了
- 2、本次答疑时间为5min



扫码加好友进群



关注直播间公告