前言
Introduction

光环国际

北京光环致成国际管理咨询股份有限公司（简称光环国际，股票代码：838504）成立于 2001 年，从项目管理起步，现已成为中国专业的 IT 管理培训机构，是专注于培养 IT 经理人的行业深耕者，是新一代 IT 技术的前沿传播者，是 IT 在线教育的成功实践者。

随着 IT 技术革命的不断深化，国内外众多行业或将面临革命性变局，IT 管理者、IT 高端技术人才是这场革命的核心角色。光环国际聚焦在拥有巨大发展前景的 IT 高端人才培训领域，以“成为 IT 经理人的终身学习伙伴，打造 IT 技术管理人才的贴身智库”为目标，助力企业顺利实现战略目标，帮助优秀的企业和个人更成功。

十七年来，不忘初心，光环国际坚持与时俱进的创新思维，知行合一的匠人精神，朴实地追求教育真谛，打造前沿的 IT 教育产品，成为深耕 IT 领域综合教育解决方案的提供商。

光环国际总部位于北京，在上海、广州、深圳、天津、武汉、西安等地设有分公司。

光环国际 AI 大数据学友会

光环学友会成立于 2004 年，是光环国际为老学友搭建的学习提升、实战交流、拓展人脉的知识服务平台。光环学友会一直秉承“分享智慧，交友共进，做前沿技术和前沿智慧同路人”的理念，目前光环国际的学友会遍布一线城市和全国：北京，上海，广州，深圳，天津和远程都有我们的学友会。

在 2017 年光环国际单独创立了光环 AI 大数据学友会，为了更为专项的为人工智能工程师直通车班和大数据开发实战转型班的学员提供专业的服务。

声明

目录
Contents

一、HR 面试题

1) 请你自我介绍一下

一般人回答这个问题过于平常，只会说姓名，年龄，爱好工作经验，这些在简历上部有。其实，企业最希望知道的是求职能否胜任工作，包括：最强的技能、最深入研究的知识领域、个性中最积极的部分，做过最成功的事，主要的成就等，这些都可以和学习无关，也可以和学习有关，但要突出积极的个性和做事的能力，说得合情合理企业才会相信。企业很重视一个人的礼貌，求职者要尊重考官，在回答每个问题之后都说一句 “谢谢”，企业喜欢有礼貌的求职者。

2) 为什么要离职？

① 回答这个问题时一定要小心，就算在前一个工作受到再大的委屈，对公司有多少的怨言，都千万不要表现来，尤其要避免对公司本身主管的批评，避免面试官的负面情绪及印象。建议此时最好的回答是将问题归咎在自己身上，例如觉得工作没有学习发展的空间，自己想在面试工作的相关产业中多加学习，或是前一份工作与自己的生涯规划不合等等，回答的答案最好是积极正面的；

② 我希望能获得一份更好的工作，如果机会来临，我会抓住。我觉得目前的工作，已达到顶峰，没有升迁机会。

3) 你觉得你个性上最大的优点是什么？

沉着冷静、条理清楚、立场坚定、顽强向上、乐于助人和关心他人、适应能力和幽默感很强、乐观和友爱。我在某公司经过一到两年的培训及项目实战，加上实习工作，使我适合这份工作。

4) 说说你最大的缺点

这个问题企业问的概率很大，通常不希望听到直接问答的缺点是什么，如果求职者说自己小心眼、爱忌妒人、非常懒、脾气大、工作效率低，企业肯定不会录用你。绝对不要自作聪明地回答 “我最大的缺点是过于追求完美”，有的人以为这样回答会显得自己比较出色，但事实上，他已经岌岌可危了。企业喜欢求职者从自己

的优点说起，中间加一些小缺点，最后再把问题转回到优点上，突出优点的部分，企业喜欢聪明的求职者。

5) 你对加班的看法

实际上公司问这个问题，并不证明一定要加班，只是想测试你是否愿意为公司奉献。

参考：如果是工作需要我义不容辞加班，我现在单身没有家庭负担，可以全身的投入到工作中，但同时我也会提高工作效率，减少不必要的加班。

6) 你的职业规划是什么？

这是每一个应聘者都不希望被问到的问题，但是几乎每个人都会问到，比较多的答案是“管理者”，但是近几年来，许多公司都以建立专业的技术途径。这些工作地位往往被称作“顾问”、“参议技师．或“高级软件工程师”等等。当然，说其他一些你感兴趣的职位也是可以的，比如产品销售部经理，生产部经理等一些与你专业背景相关的工作。要知道，考官总是喜欢有进取心的求职者，此时如果说“不知道”或许就会使你丧失一个好机会。最普通的回答应该是“我准备在技术领域有所作为”或“我希望能够按照公司管理路线发展。

7) 你对薪资的要求

如果你对薪酬的要求太低，那显然贬低自己的能力；如果你对薪酬的要求太高，那又会显得你分量过重，公司受用不起，一些雇主通常都事先对求职岗位定下开支预算，因而他们第一次提出的价钱往往是他们所能给予的最高价钱，他们问你只不过想证实一下这笔钱是否足以提起你的兴趣；

参考①：我对工资没有硬性要求，我相信贵公司在处理我的问题上会友善合理，我注重的是找对工作机会，所以只要条件公平我则不会计较太多；

参考②：我受过系统的软件编程的训练，不需要进行大量的培训，而且我本人也对编程特别感兴趣。因此我希望公司能够根据我的情况和市场标准的水平，给我合理的薪水；

参考③：如果你必须说出具体数目，请不要说出一个宽泛的范围，那样你只能

得到最低限度的数字，最好给出一个具体的数字，这样表明你已经对当下做了调查，知道像自己这样水平的雇员有什么样的价值。

8) 你最擅长的技术方向是什么？

说和你要应聘的职位相关技术，表现一下自己的热诚没什么坏处。

9) 谈谈你对跳槽的看法？

① 正常的“跳槽”能够促进人才流动的应该支持；

② 频繁的跳槽对公司和个人双方都不利的，应该反对。

10) 你朋友对你的评价？

想从侧面了解一下你的性格及与人相处的问题。

参考①：我的朋友说我是一个值得信赖的人。因为，我一旦答应别人的事情，我就一定会做到。如果我做不到，我就不会轻易许诺。

参考②：我觉的我是一个比较随和的人，与不同的人都能够友好的相处，在我与人相处时，我总能站在别人的角度考虑问题。

11) 你还有什么想要问的吗？

这个问题看上去可有可无，其实很关键，企业不喜欢“没问题”的人，因为其很注重员工的个性和创新能力。企业不喜欢求职者问福利之类的问题，如果有人这样问：贵公司对新入公司的员工有没有什么培训项目，我可以参加吗？或者说贵公司的晋升机制是什么样的？企业将很欢迎，因为体现出你对学习的热情和对公司的忠诚度以及你的上进心。

12) 最能概括你自己的三个词是什么？

我经常用的三个词是，适应能力强，有责任心和做事有始有终，结合具体例子向主考官解释。

13) **请说出你选择这份工作的动机**

这是想知道面试者对这份工作的热忱及理解度，并筛选因一时兴起而来应试的

人，如果是无经验者，可以强调“就算职种不同也希望有机会发挥之前的经验”。

14) 你做过的哪件事最令自己感到骄傲？

这是考官给你的一次机会，让你展现自己把握命运的能力，这会体现你潜在的领导能力以及你被提升的可能性。记住：你的前途取决于你的知识、社交能力和综合表现。

15) 如果你的工作出现失误，给本公司造成经济损失，你认为该怎么办？

① 我本意是为公司努力工作，如果造成经济损失，我认为首要的问题是想方法去弥补或挽回经济损失。如果我无能力负责，希望单位帮助解决；

② 分清责任，各负其责，如果是我的责任，我甘愿受罚：如果是一个我负责的团队中别人的失误，也不能幸灾乐祸，作为一个团队，需要相互提携共同完成工作，安慰同事并且帮助同事查找原因总结经验；

③ 总结经验教训，一个人的一生不可能不犯错，重要的是能从自己的或者是别人的错错误中吸取经验教训，并在今后的工作中避免发生同类的错误。检讨自己的工作方法，分析问题的深度和力度是否不够，以至出现了本可以避免的错误。

16) 工作中你难以和同事、上司相处，你该怎么办？

① 我会服从领导的指挥，配合同事的工作。

② 我会从自身找原因，仔细分析是不是自己工作做得不好让领导不满意，同事看不惯，还要看看是不是我为人处事方面做得不够好，如果是这样的话，我会努力改正。

③ 如果我找不到原因，我会找机会跟他们沟通，请他们指出我的不足，有问题就及时改正。

④ 作为优秀的员工，应该时刻以大局为重，即使在一段时间内，领导和同事对我不理解，我也会做好本职工作，虚心向他们学习，我相信，他们会看见我在努力，总有一天会对我微笑的。

17) 你能为我们公司带来什么呢？

① 假如你可以的话，试着告诉他们你可以减低他们的费用——“我已经接受过某公司多年经验，立刻就可以上岗工作”。

② 企业很想知道员工未来能为企业做什么，求职者应该再次重复自己的优势，然后说“就我的能力，我可以做一个优秀的员工在组织中发挥能力，给组织带来高效率和更多的收益”，企业喜欢求职者就申请的职位表白自己的能力。

18) 说说你对行业、技术发展趋势的看法

企业对这个问题很感兴趣，只有有备而来的求职者能够过关，求职者可以直接在网上查找对你所申请的行业部门信息，只有深入了解才能产生独特的见解。企业认为最聪明的求职者是对所面试的公司预先了解很多，包括公司各个部门发展情况，在面试回答问题的时候可以提到所了解的情况，企业欢迎进入企业的人是“知己”，而不是“盲人”。

19) 对工作的期望目标何在？

这是面试者用来评断求职者是否对他自己有一定程度的期望，对这份工作是否了解的问题。对于工作有确实目标的人，通常学习较快，自然对于新工作比较容易进入状况。在这里建议你最好针对工作的性质找出一个确实的答案，比如面试开发岗位，一定要说“自己会在短期之内做出什么样的成果或层级（目标要切合实际），为达到这个目标，我一定会努力学习，而我相信以我认真负责的态度，定可以达到这个目标”。其他岗位也可以比照这个方式回答，只要把目标方向稍作修改即可。

20) 何时可以到岗？

大多数企业会关心就职时间，最好是回答“如果被录用的话”到岗可按公司规定上班，但如果还未辞职，上班时间又太近，似乎有些强人所难，因为交接至少要一个月的时间，应进一步说明原因，录取公司应该会通融的。

二、 技术笔试题库

◆ 人工智能篇

1) 熵、联合熵、条件熵、相对熵、互信息的定义

熵

熵是用来衡量一个系统混论程度的物理量，代表一个系统中蕴含多少信息量，信息量越大表明一个系统不确定性就越大，就存在越多的可能性。熵（entropy）就用来衡量整个系统的总体信息量，其计算公式如下

$$H_s = \sum_{i=1}^{n} p_i I_e = -\sum_{i=1}^{n} p_i \log_2 p_i$$

至于这个公式怎么导出的，这里可以直观的理解一下。

熵是平均信息量，也可以理解为不确定性。例如进行决赛的巴西和南非，假设根据经验判断，巴西夺冠的几率是 80%，南非夺冠的几率是 20%，则谁能获得冠军的信息量就变为 -0.8 * log2 0.8-0.2 * log2 0.2 = 0.257 + 0.464 = 0.721，小于 1 bit 了。经验减少了判断所需的信息量，消除了不确定性。

而且通过计算可以发现，巴西夺冠的几率越高，计算出的熵就越小，即越是确定的情况，不确定性越小，信息量越少。如果巴西 100%夺冠，那么熵是 0，相当于没有任何信息。当两队几率都是 50%最难判断，所熵达到最大值 1。其实之前的 -log2 1/2= 1 bit 是简化了的计算过程，其结果也是通过熵的公式来计算的 -0.5 * log2 0.5 - 0.5* log2 0.5 = 1 bit，计算信息量要综合考虑每种结果的可能性。

另一个会迷惑的问题是熵会大于 1 吗？答案当然是肯定的，刚刚计算的最大值为 1bit，是因为最终的结果只有两种情况。在有四支球队的时候，其最大值就是 -0.25 * log20.25 - 0.25 * log2 0.25 - 0.25 * log2 0.25 - 0.25 * log2 0.25 =2 bit，当四支球队夺冠概率不等的时候，熵会小于 2 bit。

联合熵

联合熵（Joint Entropy）： 对于服从联合分布为$p(x,y)$的一对离散型随机变量(X,Y)，其联合熵$H(X,Y)$定义为：

$$H(X,Y)=-\sum_{x\in X}\sum_{y\in Y}p(x,y)\log p(x,y)=-E\log p(X,Y)。$$

联合熵是为了导出条件熵和互信息的一个定义，性质：大于每个独立的熵，2 个变量的联合熵大于或等于这 2 个变量中任一个的独立熵。

$$H(X,Y)\geq max[H(X),H(Y)]$$

$$H(X_1,...,X_n)\geq max[H(X_1),...,H(X_n)]$$

少于独立熵的，和 2 个变量的联合熵少于或等于 2 个变量的独立熵之和。这是次可加性的一个例子。该不等式有且只有在和均为统计独立的时候相等。

$$H(X,Y)\leq H(X)+H(Y)$$

$$H(X_1,...,X_n)\leq H(X_1)+...+H(X_n)$$

这表明，两个变量关联之后不确定性会增大，但是又由于相互有制约关系，不确定小于单独两个变量的不确定度之和。

条件熵

$$H(X|Y)=-\sum_{x\in X,y\in Y}p(x,y)\log P(x|y)$$

性质：就是在事件 X 的前提下，事件 Y 的熵。

链式法则：$H(X,Y)=H(X)+H(Y|X)$

用处：决策树的特征选择，实际上使用的信息增益，就是用 G(D,A)=H(Y)-H(Y|X)。可以看出在 X 的条件下，Y 的不确定度下降了多少。

相对熵

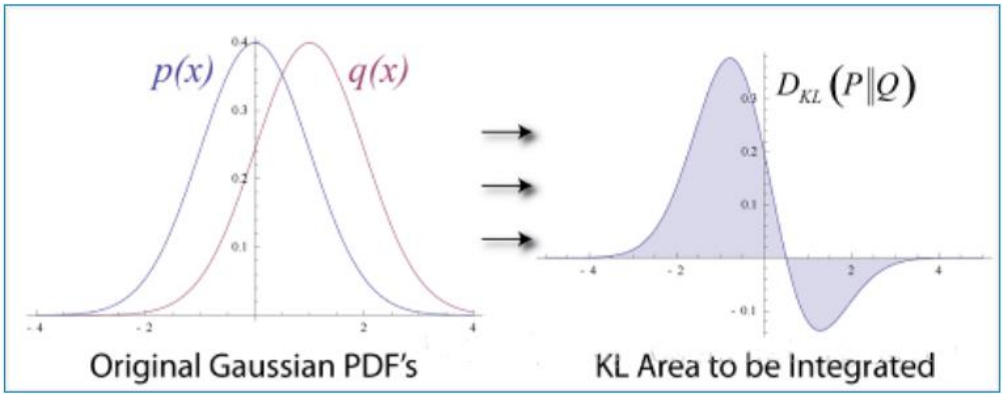
相对熵（Kullback Leibler 距离）： 两个概率密度函数 $p(x)$ 和 $q(x)$ 之间的相对熵定义为：

$$D(p\|q)=\sum_{e} p(x)\log\frac{p(x)}{q(x)}=E\ \log\frac{p(X)}{q(X)}$$

相对熵描述的是两个随机分布之间距离的度量，即当真实分布为 p 而假定分布为 q 时的无效性。

相对熵也叫交叉熵。相对熵越大，两个函数差异越大；反之，相对熵越小，两个函数差异越小。

用处：在聚类算法中，使用相对熵代替欧几里得距离，计算连个节点的相关度，据说效果不错。度量两个随机变量的差异性。



互信息

互信息： 给定两个随机变量 X 和 Y，它们的联合概率密度函数为 $p(x,y)$，其边际（边缘）概率密度函数分别是 $p(x)$ 和 $q(x)$，则互信息 $I(X;Y)$ 定义为联合分布 $p(x,y)$ 和乘积分布 $p(x)\,q(x)$ 之间的相对熵，即

$$I(X;Y)=\sum_{e}\sum_{e} p(x,y)\log\frac{p(x,y)}{p(x)p(y)}=E_{(\cdot,\cdot)}\log\frac{P(x,y)}{P(x)p(y)}=H(X)-H(X|Y).$$

了解 Y 的前提下，消除 X 的不确定度。（注意和链式法则不一样）

2) 机器学习中特征的理解

- 特征选择：原有特征选择出子集，不改变原来的特征空间。
- 特征选择的方法：

  ① Filter 方法

Chi-squared test(卡方检验)

information gain(信息增益)，详细可见“简单易学的机器学习算法——决策树之 ID3 算法”

correlation coefficient scores(相关系数)

② Wrapper 方法

将子集的选择看作是一个搜索寻优问题，生成不同的组合，对组合进行评价，再与其他的组合进行比较。这样就将子集的选择看作是一个是一个优化问题，这里有很多的优化算法可以解决，尤其是一些启发式的优化算法，如 GA，PSO，DE，ABC 等，详见“优化算法——人工蜂群算法(ABC)”，“优化算法——粒子群算法(PSO)”。

③ Embedded 方法

在模型既定的情况下学习出对提高模型准确性最好的属性。这句话并不是很好理解，其实是讲在确定模型的过程中，挑选出那些对模型的训练有重要意义的属性。主要方法正则化可参考“简单易学的机器学习算法--岭回归(RidgeRegression)”，岭回归就是在基本线性回归的过程中加入了正则项。

- 降维：将原有的特征重组成为包含信息更多的特征，改变了原有的特征空间
- 降维的主要方法：
  ① Principal Component Analysis(主成分分析)
  ② Singular Value Decomposition(奇异值分解)
  ③ Sammon’s Mapping(Sammon 映射)

注：数据和特征决定了机器学习的上限，而模型和算法只是逼近这个上限而已

3) L1 和 L2 正则的区别，如何选择 L1 和 L2 正则？

L1Norm 和 L2 Norm 的区别（核心：L2 对大数，对 outlier 更敏感!）：

L1 优点是能够获得 sparse 模型，对于 large-scale 的问题来说这一点很重要，因为可以减少存储空间。缺点是加入 L1 后目标函数在原点不可导，需要做特殊处理。

L2 优点是实现简单，能够起到正则化的作用。缺点就是 L1 的优点，无法获得 sparse 模型。

实际上 L1 也是一种妥协的做法，要获得真正 sparse 的模型，要用 L0 正则化。参考：机器学习中的范数规则化之（一）L0、L1 与 L2 范数

4) 线性分类器与非线性分类器的区别及优劣；

区别：

所谓线性分类器即用一个超平面将正负样本分离开，表达式为 y=wx 。这里是强调的是平面。而非线性的分类界面没有这个限制，可以是曲面，多个超平面的组合等。

典型的线性分类器有感知机，LDA，逻辑斯特回归，SVM（线性核）；

典型的非线性分类器有朴素贝叶斯，kNN，决策树，SVM（非线性核）

优缺点：

- 线性分类器判别简单、易实现、且需要的计算量和存储量小。为解决比较复杂的线性不可分样本分类问题，提出非线性判别函数。：超曲面，非线性判别函数计算复杂，实际应用上受到较大的限制。在线性分类器的基础上，用分段线性分类器可以实现复杂的分类面。解决问题比较简便的方法是采用多个线性分界面将它们分段连接，用分段线性判别划分去逼近分界的超曲面。
- 2.如果一个问题是非线性问题并且它的类边界不能够用线性超平面估计得很好，那么非线性分类器通常会比线性分类器表现得更精准。如果一个问题是线性的，那么最好使用简单的线性分类器来处理。

5) SVM 的损失函数推导

SVM 的损失函数定义如下：

$$L_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + \Delta)$$

例：假设有 3 个分类，并且得到了分值 s=[13,-7,11]。其中第一个类别是正确类别，即 $y_i = 0$。同时假设 Δ 是 10（后面会详细介绍该超参数）。上面的公式是将所有不正确分类（$j \neq y_i$）加起来，所以我们得到两个部分：

$$Li = max(0, -7 - 13 + 10) + max(0, 11 - 13 + 10)$$

第一个部分结果是 0，这是因为[-7-13+10]得到的是负数，经过 max(0,-)函数处理后得到 0。这一对类别分数和标签的损失值是 0，这是因为正确分类的得分 13 与错误分类的得分-7 的差为 20，高于边界值 10。而 SVM 只关心差距至少要大于 10，更大的差值还是算作损失值为 0。

第二个部分计算[11-13+10]得到 8。虽然正确分类的得分比不正确分类的得分要高（13>11），但是比 10 的边界值还是小了，分差只有 2，这就是为什么损失值等于 8。简而言之，SVM 的损失函数想要正确分类类别 y_i 的分数比不正确类别分数高，而且至少要高 Δ。如果不满足这点，就开始计算损失值。

那么在这次的模型中，我们面对的是线性评分函数（$f(x_i, W) = Wx_i$），所以我们可以将损失函数的公式稍微改写一下：

$$L_i = \sum_{j \neq y_i} max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)$$

其中 w_j 是权重 W 的第 j 行，被变形为列向量。然而，一旦开始考虑更复杂的评分函数 f 公式，这样做就不是必须的了。

关于 0 的阀值：max(0,-)函数，它常被称为折叶损失（hinge loss）。有时候会听到人们使用平方折叶损失 SVM（即 L2-SVM），它使用的是 $max(0, -)^2$，将更强烈（平方的而不是线性的）地惩罚过界的边界值。不使用平方是更标准的版本，但是在某些数据集中，平方折叶损失会工作得更好。可以通过交叉验证来决定到底使用哪个。

【注释】第 4 题参考答案出处--知乎-杜客

6) SVM 和神经网络的优缺点

- **SVM 的优点**

非线性映射是 SVM 方法的理论基础,SVM 利用内积核函数代替向高维空间的非线性映射；

对特征空间划分的最优超平面是 SVM 的目标,最大化分类边际的思想是 SVM 方法的核心；

支持向量是 SVM 的训练结果,在 SVM 分类决策中起决定作用的是支持向量.

SVM 是一种有坚实理论基础的新颖的小样本学习方法.它基本上不涉及概率测度及大数定律等,因此不同于现有的统计方法.从本质上看,它避开了从归纳到演绎的传统过程,实现了高效的从训练样本到预报样本的“转导推理”,大大简化了通常的分类和回归等问题.

SVM 的最终决策函数只由少数的支持向量所确定,计算的复杂性取决于支持向量的数目,而不是样本空间的维数,这在某种意义上避免了“维数灾难”.

少数支持向量决定了最终结果,这不但可以帮助我们抓住关键样本、“剔除”大量冗余样本,而且注定了该方法不但算法简单,而且具有较好的“鲁棒”性.这种“鲁棒”性主要体现在:

① 增、删非支持向量样本对模型没有影响；

② 支持向量样本集具有一定的鲁棒性；

③ 有些成功的应用中,SVM 方法对核的选取不敏感；

- **SVM 的缺点**

① SVM 算法对大规模训练样本难以实施；

由于 SVM 是借助二次规划来求解支持向量,而求解二次规划将涉及 m 阶矩阵的计算（m 为样本的个数）,当 m 数目很大时该矩阵的存储和计算将耗费大量的机器内存和运算时间.针对以上问题的主要改进有有 J.Platt 的 SMO 算法、T.Joachims 的 SVM、C.J.C.Burges 等的 PCGC、张学工的 CSVM 以及 O.L.Mangasarian 等的 SOR 算法

② 用 SVM 解决多分类问题存在困难

经典的支持向量机算法只给出了二类分类的算法,而在数据挖掘的实际应

用中,一般要解决多类的分类问题.可以通过多个二类支持向量机的组合来解决.主要有一对多组合模式、一对一组合模式和 SVM 决策树；再就是通过构造多个分类器的组合来解决.主要原理是克服 SVM 固有的缺点,结合其他算法的优势,解决多类问题的分类精度.如：与粗集理论结合,形成一种优势互补的多类问题的组合分类器。

- **神经网络优点：**

  有很强的非线性拟合能力，可映射任意复杂的非线性关系，而且学习规则简单，便于计算机实现。具有很强的鲁棒性、记忆能力、非线性映射能力以及强大的自学习能力，因此有很大的应用市场。

- **神经网络缺点：**

  ① 最严重的问题是没能力来解释自己的推理过程和推理依据。

  ② 不能向用户提出必要的询问，而且当数据不充分的时候，就无法进行工作。

  ③ 把一切问题的特征都变为数字，把一切推理都变为，其结果势必是丢失信息。

  ④ 理论和学习算法还有待于进一步完善和提高。

7) 监督学习、无监督学习与强化学习

- **Supervised Learning -- 监督学习**

  监督学习，就是人们常说的分类，通过已有的训练样本（即已知数据以及其对应的输出）去训练得到一个最优模型（这个模型属于某个函数的集合，最优则表示在某个评价准则下是最佳的），再利用这个模型将所有的输入映射为相应的输出，对输出进行简单的判断从而实现分类的目的，也就具有了对未知数据进行分类的能力。

8) word2vec 的简介、训练方法及过程概述

- **word2vec 的简介**

  word2vec 最初是 Tomas Mikolov 发表的一篇文章[1]，同时开源了相应的代码，作用是将所有词语投影到 KK 维的向量空间，每个词语都可以用一个 KK 维向量表示。

为什么要将词用向量来表示呢？这样可以给词语一个数学上的表示，使之可以适用于某些算法或数学模型。

- **word2vec 训练方法**

通常将词语表示成向量有如下两种方法：

① one-hot 表示法

假如语料库里一共有 N 个词，one-hot 表示即是为每个词分配一个唯一的索引，并且将每个词表示为 N 维的向量，在该词索引对应的维度值为 1，其余维度均为 0。如一共有三个词：今天、天气、真好，那么三个词的词向量分别可以是[1,0,0],[0,1,0],[0,0,1][1,0,0],[0,1,0],[0,0,1]。这种简单的表示方法已经可以解决相当一部分 NLP 的问题，不过仍然存在不足，即词向量与词向量之间都是相互独立的，我们无法通过这种词向量得知两个词在语义上是否相似，并且如果 N 非常大，这种高维稀疏的表示也有可能引发维度灾难。为了解决上述问题，就有了词向量的第二种表示方法。

② Distributed 表示法

word2vec 就是通过这种方法将词表示为向量，即通过训练将词表示为限定维度 K 的实数向量，这种非稀疏表示的向量很容易求它们之间的距离(欧式、余弦等)，从而判断词与词语义上的相似性。如 K=3 时，我们得到的实数向量可以是[0.5,0.22,0.7][0.5,0.22,0.7]这样。不过 Distributed 表示法并不是 word2vec 诞生才有的，这种方法早在 1986 年 Hinton 就提出了[2]。word2vec 之所以会产生这么大的影响，是因为它采用了简化的模型，使得训练速度大为提升，让 word embedding 这项技术(也就是词的 distributed 表示)变得较为实用。

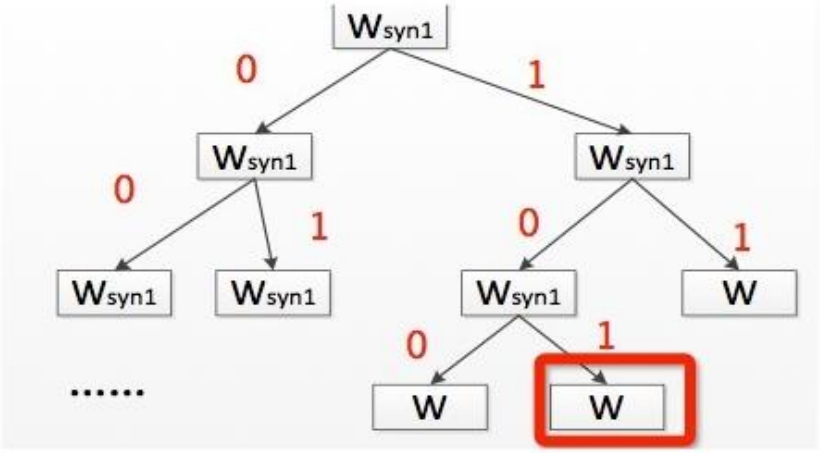
- **word2vec 训练过程**

① 为了训练出词向量，要先准备语料。先将中文语料分好词，分词的方法有很多，这里不做详细说明。再去除一些无意义的词，比如纯数字

1523523523，乱码 fasdfkalsjfwek 等，这样得到的结果看起来更干净。如果是用 google 开源的 word2vec 实现，那就将所有语料保存在一个文本里，可以所有词用一行表示，也可以有换行符 \n ，google 的代码里自动将换行符替换为 <\s> ，所以在结果词向量里会看到这么一个词，不要觉得奇怪;

② 扫描语料库，统计每个词出现的次数，保存在一个 hash 表里;

③ 根据各词的词频建立哈夫曼树。哈夫曼树是一棵最优二叉树，哈夫曼树中的每个叶子节点都有一个权值，并且所有叶子节点的权值乘上其到根节点路径的长度的累加和最小。因此权值较大的叶子节点往往比较靠近根节点。每个词汇最终都是哈夫曼树的叶子节点，词频就是相应的权值，所有的非叶子节点代表了某一类的词，如下图中的 W_{syn1} ，每一个 W_{syn1} 也是与叶子节点的词一样，为 K 维的向量。哈夫曼树建立好以后，每个词都会有一个二进制的哈夫曼编码，用于表示从根节点到该词汇的路径。比如我们假设往左子节点走编码为 0，往右为 1，下图中方框内的词的哈夫曼编码就是 101;



④ 初始化词向量与哈夫曼树非叶子节点的向量。向量的维度是我们给定的参数 K，google 的代码里将词向量的每个维度都随机初始化为 $\pm 0.00X$ ，将非叶子节点每个维度初始化为 0。word2vec 最终得到的是词向量，但并不是执行一系列复杂的运算，在最终算出一堆的词向量，词向量在一开始就已经存在了，整个运算过程只是不断的迭代优

化，让被随机初始化的词向量在词向量空间里逐渐挪到属于它自己的位置

⑤ 训练，也就是迭代最优化。再回到语料库，逐句的读取一系列的词，然后用 CNN 算出梯度，再更新词向量的值、非叶子节点处向量的词。通常神经网络训练过程，会指定 max epoch，与 early stop 的 patience，并且独立保存一个验证集，用于每一个 epoch 后用验证集算一下 error，然后用 early stop 策略来控制是否停止训练。但是 word2vec 的实现更简单粗暴，如果语料文档读完了，就终止，或者是每个线程遍历过的词数超过了一个值。

```
if (feof(fi)) break;
if (word_count > train_words / num_threads) break;
```

word2vec 用的是神经网络模型，分为两种，cbow 与 skip-gram，每个模型的训练方法又分别有两种，hierarchical sofmax 与 negative sampling。

【注释】第 8 题参考答案出处--thriving_fcl 的博客

9) K-means 算法和高斯混合模型的异同

- **Kmeans 聚类**

聚类算法除了 Kmeans，还有其它的聚类方式，比如层次聚类、基于模糊等价关系的模糊聚类。Kmeans 只不过是这些聚类算法中最为常见的一中，也是使用得最多的一种。因为 Kmeans 本身的复杂度高（O(N^3)），所以 Kmeans 有很多其它的变种：

① 针对浮点型数据的运算，为提高运算效率，将浮点型 Round 成整形（我操作的话一般先乘一个系数，以使数据分布在整个整数空间，降低 Round 的数据损失）

② 层次化的 Kmeans 聚类(HIKM)

先给出一般 Kmeans 聚类问题的描述：

① 给定数据集： $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$

② 将数据 group 成 K 个 clusters；

注：相对于监督学习算法，Kmeans 虽然没有 label 信息，但聚类数 K 却是已知的，也就是说使用 Kmeans 得提前计划聚成多少个 cluster。

- **k-means 算法的具体步骤**

① 选定 K 个中心 μ_k 的初值。这个过程通常是针对具体的问题有一些启发式的选取方法，或者大多数情况下采用随机选取的办法。因为前面说过 k-means 并不能保证全局最优，而是否能收敛到全局最优解其实和初值的选取有很大的关系，所以有时候我们会多次选取初值跑 k-means ，并取其中最好的一次结果；

② 将每个数据点归类到离它最近的那个中心点所代表的 cluster 中；

③ 用公式 $\mu_k = \frac{1}{N_k}\sum_{j\in \text{cluster}_k} x_j$ 计算出每个 cluster 的新的中心点；

④ 重复第二步，一直到迭代了最大的步数或者前后的 J 的值相差小于一个阈值为止；

参考《机器学习》周志华书中 k-means 的定义，要求的目标函数就是 E 函数。

9.4.1 k 均值算法

给定样本集 $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$，“$k$ 均值”(k-means)算法针对聚类所得簇划分 $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ 最小化平方误差

$$E = \sum_{i=1}^{k} \sum_{\boldsymbol{x}\in C_i} ||\boldsymbol{x} - \boldsymbol{\mu}_i||_2^2 , \tag{9.24}$$

其中 $\boldsymbol{\mu}_i = \frac{1}{|C_i|}\sum_{\boldsymbol{x}\in C_i} \boldsymbol{x}$ 是簇 C_i 的均值向量. 直观来看, 式(9.24) 在一定程度上刻画了簇内样本围绕簇均值向量的紧密程度, E 值越小则簇内样本相似度越高.

- **高斯混合模型（GMM）**

GMM 和 k-means 很像，不过 GMM 是学习概率密度函数（GMM 除了用在 clustering 上之外，还被用于 density estimation ），k-means 的结果是每个数据点被分配到其中某一个 cluster ，而 GMM 则给出这些数据点被分配到每个 cluster 的概率，又称作 soft assignment，也称作软聚类 。

得出一个概率有很多好处，因为它的信息量比简单的一个结果要多，比如，我可以把这个概率转换为一个 score ，表示算法对自己得出的这个结果的把

握。也许可以对同一个任务，用多个方法得到结果，最后选取“把握”最大的那个结果；另一个很常见的方法是在诸如疾病诊断之类的场所，机器对于那些很容易分辨的情况（患病或者不患病的概率很高）可以自动区分，而对于那种很难分辨的情况，比如，49% 的概率患病，51% 的概率正常，如果仅仅简单地使用 50% 的阈值将患者诊断为“正常”的话，风险非常大，因此，在机器对自己的结果把握很小的情况下，会“拒绝发表评论”，而把这个任务留给有经验的医生去解决。

高斯判别分析法（GDA），是通过计算样本的后验概率来进行判别，而后验概率是通过假设多元高斯模型来计算得来的。高斯模型的参数：均值、协方差，是由已标定（分类）的样本得来，所以可以看做是一种监督学习方法。

在 GMM 模型（属于无监督学习），给定未分类的 m 个样本（n 维特征），假设可分为 k 个类，要求用 GMM 算法对其进行分类。如果知道每个类的高斯参数，则可以向 GDA 算法那样计算出后验概率进行判别。但遗憾的是，输入的样本未被标定，也就是说得不到高斯参数：均值、协方差。这就引出 EM（Expectation Maximization Algorithm：期望最大化）算法。

- **高斯混合模型聚类算法 EM 步骤**

① 猜测有几个类别，既有几个高斯分布

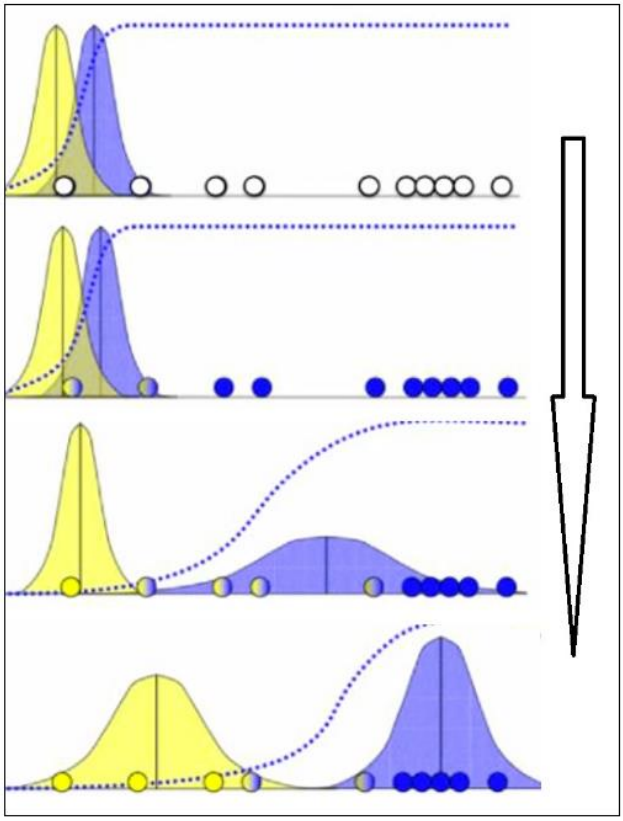
② 针对每一个高斯分布，随机给其均值和方差进行赋值

③ 针对每一个样本，计算其在各个高斯分布下的概率

$$f(x)=\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

④ 针对每一个高斯分布，每一个样本对该高斯分布的贡献可以由其下的概率表示，如概率大则表示贡献大，反之亦然。这样把样本对该高斯分布的贡献作为权重来计算加权的均值和方差。之后替代其原本的均值和方差。

⑤ 重复 3~4 直到每一个高斯分布的均值和方差收敛。



注：当高斯混合模型的特征值维数大于一维时，在计算加权的时候还要计算协方差，即要考虑不同维度之间的相互关联。

- **异同处**

① GMM 和 K-means 很相似，相似点在于两者的分类受初始值影响；两者可能限于局部最优解；两者类别的个数都要靠猜测；

② K-means 属于硬聚类，而 GMM 属于混合式软聚类，一个样本 70%属于 A，30%属于 B；同时多维的 GMM 在计算均值和方差时使用了协方差，应用了不同维度之间的相互约束关系；

③ 从欧式距离和马氏距离来说， K-means 是欧式距离，而 GMM 是马氏距离，当协方差非对角线元素为 0 时，马氏距离也就变成了欧式距离。

【注释】第 9 题参考答案出处--一梦南柯博客

10) 如何解决过拟合问题

模型在训练集表现好，在真实数据表现不好，即模型的泛化能力不够。从另外一个方面来讲，

模型在达到经验损失最小的时候，模型复杂度较高，结构风险没有达到最优。

学习方法上：限制机器的学习，使机器学习特征时学得不那么彻底，因此这样就可以降低机器学到局部特征和错误特征的几率，使得识别正确率得到优化.数据上：要防止过拟合，做好特征的选取。训练数据的选取也是很关键的，良好的训练数据本身的局部特征应尽可能少，噪声也尽可能小.个 ID 位置，修改为一样即可解决。

11) 为什么不用逻辑回归，而要用 GBM？

GB 是 Gradient Boosting。引用知乎答主 Frankenstein 的话，从决策边界上看，线性回归的决策边界是一条直线，逻辑回归的决策边界是一条曲线，GBM 的决策边界可能是很多条线。逻辑回归只能处理回归问题，而 GBM 还可以用于解决分类或排序问题。

12) 为什么很多做人脸的 Paper 会最后加入一个 Local Connected Conv？

a. 人脸检测，使用 6 个基点

b. 二维剪切，将人脸部分裁剪出来

c. 67 个基点，然后 Delaunay 三角化，在轮廓处添加三角形来避免不连续

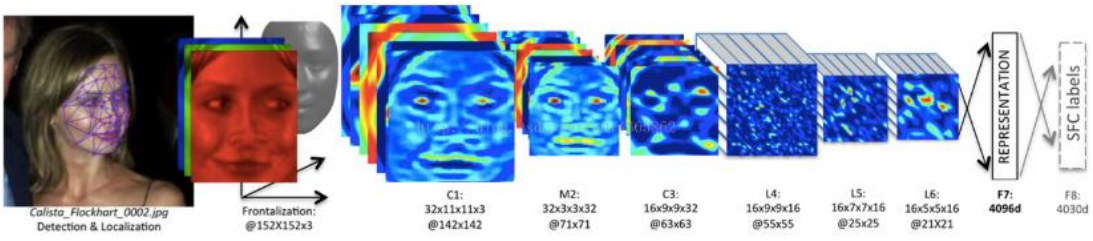
d. 将三角化后的人脸转换成 3D 形状

e. 三角化后的人脸变为有深度的 3D 三角网

f. 将三角网做偏转，使人脸的正面朝前

g. 最后放正的人脸

h. 一个新角度的人脸（在论文中没有用到）



这一步的作用就是使用 3D 模型来将人脸对齐，从而使 CNN 发挥最大的效果，经过 3D 对齐以后，形成的图像都是 152×152 的图像，输入到上述网络结构中，该结构的参数如下：

Conv：32 个 11×11×3 的卷积核

max-pooling: 3×3， stride=2

Conv: 16 个 9×9 的卷积核

Local-Conv: 16 个 9×9 的卷积核，Local 的意思是卷积核的参数不共享

Local-Conv: 16 个 7×7 的卷积核，参数不共享

Local-Conv: 16 个 5×5 的卷积核，参数不共享

Fully-connected: 4096 维

Softmax: 4030 维

前三层的目的在于提取低层次的特征，比如简单的边和纹理。其中 Max-pooling 层使得卷积的输出对微小的偏移情况更加鲁棒。但没有用太多的 Max-pooling 层，因为太多的 Max-pooling 层会使得网络损失图像信息。

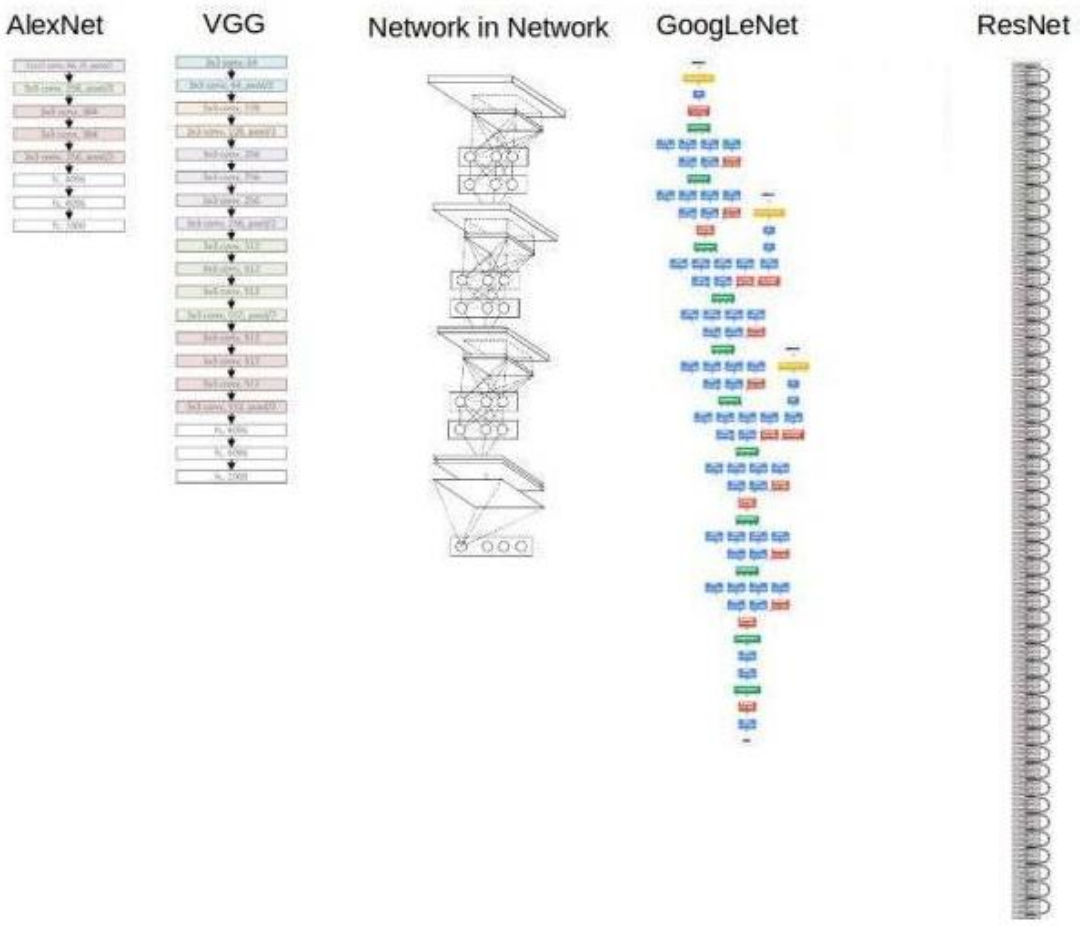
后面三层都是使用参数不共享的卷积核，之所以使用参数不共享，有如下原因：对齐的人脸图片中，不同的区域会有不同的统计特征，卷积的局部稳定性假设并不存在，所以使用相同的卷积核会导致信息的丢失不共享的卷积核并不增加抽取特征时的计算量，而会增加训练时的计算量使用不共享的卷积核，需要训练的参数量大大增加，因而需要很大的数据量，然而这个条件本文刚好满足。

全连接层将上一层的每个单元和本层的所有单元相连，用来捕捉人脸图像不同位置的特征之间的相关性。其中，第 7 层（4096-d）被用来表示人脸。全连接层的输出可以用于 Softmax 的输入，Softmax 层用于分类。

【注释】第 12 题参考答案出处--小鹏的专栏的博客

13) CNN 常用的几个模型

基于 imagenet 上 1.2 million 数据训练出来的经典模型



- **AlexNet**

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

我们训练了一个大型的深度卷积神经网络，来将在 ImageNet LSVRC-2010 大赛中的 120 万张高清图像分为 1000 个不同的类别。对测试 数据，我们得到了 top-1 误差率 37.5%，以及 top-5 误差率 17.0%，这个效果比之前最顶尖的都要好得多。该神经网络有 6000 万个参数和 650,000 个神经元，由五个卷积层，以及某些卷积层后跟着的 max-pooling 层，和三个全连接层，还有排在最后的

1000-way 的 softmax 层组成。为了使训练速度更快，我们使用了非饱和的神经元和一个非常高效的 GPU 关于卷积运算的工具。为了减少全连接层的过拟合，我们采用了最新开发的正则化方法，称为“dropout”，它已被证明是非常有效的。在 ILSVRC-2012 大赛中，我们又输入了该模型的一个变体，并依靠 top-5 测试误差率 15.3%取得了胜利，相比较下，次优项的错误率是 26.2%。

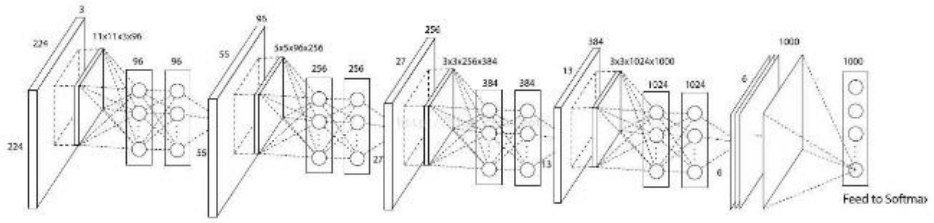
【注释】© 1987 – 2018 Neural Information Processing Systems Foundation, Inc.

- **Network in Network**

Network In Network

Min Lin[1,2], Qiang Chen[2], Shuicheng Yan[2]
[1]Graduate School for Integrative Sciences and Engineering
[2]Department of Electronic & Computer Engineering
National University of Singapore, Singapore
{linmin,chenqiang,eleyans}@nus.edu.sg

作者对传统 CNN 的两个改进，利用其进行 1000 物体分类问题，最后设计了一个：4 层的 NIN+全局均值池化，网络如下：



这篇文献很有价值，实现方式也很简单，一开始我还以为需要 caffe 的 c++源码来实现 NIN 网络，结果发现实现 NIN 的源码实现方式其实就是一个 1*1 的卷积核，实现卷积运算，所以实现起来相当容易，不需要自己写源码，只需要简简单单的把卷积核的大小变一下，然后最后一层的全连接层直接用 avg pooling 替换一下就 ok 了。网络浅显易懂，简单实现，却可以改进原来的网络，提高精度，减小模型大小。

【注释】论文地址：https://arxiv.org/pdf/1312.4400v3.pdf

- **VGG**

Published as a conference paper at ICLR 2015

VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

Karen Simonyan* **& Andrew Zisserman+**
Visual Geometry Group, Department of Engineering Science, University of Oxford
{karen,az}@robots.ox.ac.uk

vgg 继承了 lenet 以及 alexnet 的一些框架，尤其是跟 alexnet 框架非常像，vgg 也是 5 个 group 的卷积、2 层 fc 图像特征、一层 fc 分类特征，可以看做和 alexnet 一样总共 8 个 part。根据前 5 个卷积 group，每个 group 中的不同配置，vgg 论文中给出了 A~E 这五种配置，卷积层数从 8 到 16 递增。从论文中可以看到从 8 到 16 随着卷积层的一步步加深，貌似通过加深卷积层数也已经到达准确率提升的瓶颈了。

【注释】论文地址：https://arxiv.org/pdf/1409.1556.pdf

- **GoogLeNet**

Going Deeper with Convolutions

Christian Szegedy[1], Wei Liu[2], Yangqing Jia[1], Pierre Sermanet[1], Scott Reed[3],
Dragomir Anguelov[1], Dumitru Erhan[1], Vincent Vanhoucke[1], Andrew Rabinovich[4]
[1]Google Inc. [2]University of North Carolina, Chapel Hill
[3]University of Michigan, Ann Arbor [4]Magic Leap Inc.
[1]{szegedy,jiayq,sermanet,dragomir,dumitru,vanhoucke}@google.com
[2]wliu@cs.unc.edu, [3]reedscott@umich.edu, [4]arabinovich@magicleap.com

GoogLeNet 的计算效率明显高于 VGGNet，大约只有 500 万参数，只相当于 Alexnet 的 1/12(GoogLeNet 的 caffemodel 大约 50M，VGGNet 的 caffemodel 则要超过 600M)。GoogLeNet 的表现很好，但是，如果想要通过简单地放大 Inception 结构来构建更大的网络，则会立即提高计算消耗。

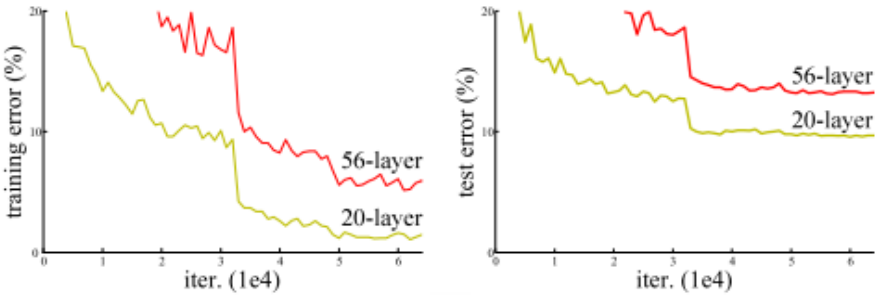
【注释】论文地址：http://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Szegedy_Going_Deeper_With_2015_CVPR_paper.pdf

- **Resnet**

Deep Residual Learning for Image Recognition

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

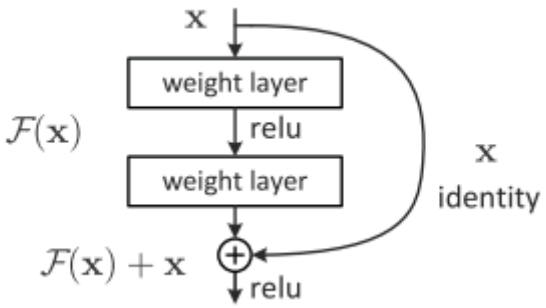
深度学习网络的深度对最后的分类和识别的效果有着很大的影响，所以正常想法就是能把网络设计的越深越好，但是事实上却不是这样，常规的网络的堆叠（plain network）在网络很深的时候，效果却越来越差了。



这里其中的原因之一即是网络越深，梯度消失的现象就越来越明显，网络的训练效果也不会很好。 但是现在浅层的网络（shallower network）又无法明显提升网络的识别效果了，所以现在要解决的问题就是怎样在加深网络的情况下又解决梯度消失的问题。

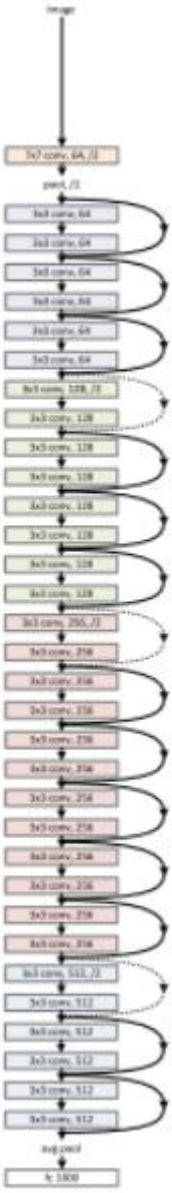
- **ResNet 的解决方案**

ResNet 引入了残差网络结构（residual network），通过残差网络，可以把网络层弄的很深，据说现在达到了 1000 多层，最终的网络分类的效果也是非常好，残差网络的基本结构如下图所示



通过在输出个输入之间引入一个 shortcut connection,而不是简单的堆叠

网络，这样可以解决网络由于很深出现梯度消失的问题，从而可可以把网络做的很深，ResNet 其中一个网络结构如下图所示



之前一直在探究残差网络提出的由来，作者是基于先前的什么知识才提出残差网络的，咋一看感觉残差网络提出的很精巧，其实就是很精巧，但是现在感觉非要从残差的角度进行解读感觉不太好理解，真正起作用的应该就是 shortcut 连接了，这才是网络的关键之处。

【注释】论文地址：https://arxiv.org/pdf/1512.03385v1.pdf

14) 梯度爆炸

梯度爆炸就是由于初始化权值过大，前面层会比后面层变化的更快，就会导致权值越来越大，梯度爆炸的现象就发生了。

- **如何确定是否出现梯度爆炸**

训练过程中出现梯度爆炸会伴随一些细微的信号，如：

① 模型无法从训练数据中获得更新（如低损失）；

② 模型不稳定，导致更新过程中的损失出现显著变化；

③ 训练过程中，模型损失变成 NaN。

- **如何修复梯度爆炸问题**

① 重新设计网络模型

在深度神经网络中，梯度爆炸可以通过重新设计层数更少的网络来解决。使用更小的批尺寸对网络训练也有好处。另外也许是学习率的原因，学习率过大导致的问题，减小学习率。在循环神经网络中，训练过程中在更少的先前时间步上进行更新（沿时间的截断反向传播，truncated Backpropagation through time）可以缓解梯度爆炸问题。

② 使用 ReLU 激活函数

在深度多层感知机神经网络中，梯度爆炸的发生可能是因为激活函数，如之前很流行的 Sigmoid 和 Tanh 函数。使用 ReLU 激活函数可以减少梯度爆炸。采用 ReLU 激活函数是最适合隐藏层的新实践。

③ 使用长短期记忆网络

在循环神经网络中，梯度爆炸的发生可能是因为某种网络的训练本身就存在不稳定性，如随时间的反向传播本质上将循环网络转换成深度多层感知机神经网络。使用长短期记忆（LSTM）单元和相关的门类型神经元结构可以减少梯度爆炸问题。采用 LSTM 单元是适合循环神经网络的序列预测的最新最好实践。

④ 使用梯度截断（Gradient Clipping）

在非常深且批尺寸较大的多层感知机网络和输入序列较长的 LSTM 中，仍然有可能出现梯度爆炸。如果梯度爆炸仍然出现，你可以在训练过程中检查和限制梯度的大小。这就是梯度截断。

⑤ 使用权重正则化（Weight Regularization）

如果梯度爆炸仍然存在，可以尝试另一种方法，即检查网络权重的大小，并惩罚产生较大权重值的损失函数。该过程被称为权重正则化，通常使用的是 L1 惩罚项（权重绝对值）或 L2 惩罚项（权重平方）。

【注释】参考答案出处--hank 的 DL 之路

15) 【MachineLearning】多分类与多标签算法的定义与区别

- **Multiclass classification** means a classification task with more than two classes; e.g., classify a set of images of fruits which may be oranges, apples, or pears. Multiclass classification makes the assumption that each sample is assigned to one and only one label: a fruit can be either an apple or a pear but not both at the same time.
- **Multilabel classification** assigns to each sample a set of target labels. This can be thought as predicting properties of a data-point that are not mutually exclusive, such as topics that are relevant for a document. A text might be about any of religion, politics, finance or education at the same time or none of these.
- **Multioutput-multiclass** classification and multi-task classification means that a single estimator has to handle several joint classification tasks. This is a generalization of the multi-label classification task, where the set of classification problem is restricted to binary classification, and of the multi-class classification task. The output format is a 2d numpy array or sparse matrix.

  The set of labels can be different for each output variable. For instance a sample could be assigned “pear” for an output variable that takes possible values in a finite set of species such as “pear”, “apple”, “orange” and “green” for a second output variable that takes possible values in a finite set of colors such as “green”, “red”, “orange”, “yellow”…

  This means that any classifiers handling multi-output multiclass or multi-task classification task supports the multi-label classification task as a special case. Multi-task classification is similar to the multi-output classification task with different model formulations. For more information, see the relevant estimator

documentation.

Multiclass classification 就是多分类问题，比如年龄预测中把人分为小孩，年轻人，青年人和老年人这四个类别。Multiclass classification 与 binary classification 相对应，性别预测只有男、女两个值，就属于后者。

Multilabel classification 是多标签分类，比如一个新闻稿 A 可以与{政治，体育，自然}有关，就可以打上这三个标签。而新闻稿 B 可能只与其中的{体育，自然}相关，就只能打上这两个标签。

Multioutput-multiclass classification 和 multi-task classification 指的是同一个东西。仍然举前边的新闻稿的例子，定义一个三个元素的向量，该向量第 1、2 和 3 个元素分别对应是否（分别取值 1 或 0）与政治、体育和自然相关。那么新闻稿 A 可以表示为[1,1,1]，而新闻稿 B 可以表示为[0,1,1]，这就可以看成是 multi-task classification 问题了。 从这个例子也可以看出，Multilabel classification 是一种特殊的 multi-task classification 问题。之所以说它特殊，是因为一般情况下，向量的元素可能会取多于两个值，比如同时要求预测年龄和性别，其中年龄有四个取值，而性别有两个取值。

	$K = 2$	$K > 2$
$L = 1$	**binary**	**multi-class**
$L > 1$	**multi-label**	**multi-output**†

† also known as multi-target, multi-dimensional.

Figure : For L target variables (labels), each of K values.

multi-output can be cast to multi-label, just as multi-class can be cast to binary.

【注释】参考答案出处--窃・格瓦拉

◆ 大数据篇

1) fsimage 和 editlog 的区别

HDFS 中的 NameNode 被称为元数据节点 DataNode 称为数据节点。NameNode 维护了文件与数据块的映射表以及数据块与数据节点的映射表，而真正的数据是存储在 DataNode 上。

对于 NameNode 如何存储这些信息，它维护两个文件，一个是 fsimage，一个是 editlog。

- **fsimage 与 editlog 作用**

① fsimage 保存了最新的元数据检查点，在 HDFS 启动时加载 fsimage 的信息，包含了整个 HDFS 文件系统的所有目录和文件的信息。对于文件来说包括了数据块描述信息、修改时间、访问时间等；对于目录来说包括修改时间、访问权限控制信息（目录所属用户，所在组）等；

② editlog 主要是在 NameNode 已经启动情况下对 HDFS 进行的各种更新操作进行记录，HDFS 客户端执行所有的写操作都会被记录到 editlog 中。

- **工作原理**

从最新检查点后，hadoop 将对每个文件的操作都保存在 edits 中，为避免 edits 不断增大，secondary namenode 就会周期性合并 fsimage 和 edits 成新的 fsimage，edits 再记录新的变化。

这种机制有个问题：因 edits 存放在 Namenode 中，当 Namenode 挂掉，edits 也会丢失，导致利用 secondary namenode 恢复 Namenode 时，会有部分数据丢失。

2) hadoop 系统的构成

① Hadoop 是一个能够对大量数据进行分布式处理的软件框架,以一种可靠、高效、可伸缩的方式进行数据处理，其有许多元素构成，以下是其组成元素：①Hadoop Common :Hadoop 体系最底层的一个模块,为 Hadoop 各子项目提供各种工具，如：配置文件和日志操作等。

② HDFS:分布式文件系统,提供高吞吐量的应用程序数据访问,对外部客

户机而言,HDFS 就像一个传统的分级文件系统。可以创建、删除、移动或重命名文件等等。但是 HDFS 的架构是基于一组特定的节点构建的，这是由它自身的特点决定的。这些节点包括 NameNode(仅一个), 它在 HDFS 内部提供元数据服务；DataNode，它为 HDFS 提供存储块。由于仅存在一个 NameNode，因此这是 HDFS 的一个缺点（单点失败）。存储在 HDFS 中的文件被分成块，然后将这些块复制到多个计算机中（DataNode）。这与传统的 RAID 架构大不相同。块的大小(通常为 64MB)和复制的块数量在创建文件时由客户机决定。NameNode 可以控制所有文件操作。HDFS 内部的所有通信都基于标准的 TCP/IP 协议；

③ MapReduce ：一个分布式海量数据处理的软件框架集计算集群；

④ Avro ：doug cutting 主持的 RPC 项目，主要负责数据的序列化。有点类似 Google 的 protobuf 和 Facebook 的 thrift。avro 用来做以后 hadoop 的 RPC,使 hadoop 的 RPC 模块通信速度更快、数据结构更紧凑；

⑤ Hive ：类似 CloudBase,也是基于 hadoop 分布式计算平台上的提供 data warehouse 的 sql 功能的一套软件。使得存储在 hadoop 里面的海量数据的汇总，即席查询简单化。hive 提供了一套 QL 的查询语言，以 sql 为基础，使用起来很方便；

⑥ HBase ：基于 Hadoop Distributed File System，是一个开源的，基于列存储模型的可扩展的分布式数据库，支持大型表的存储结构化数据。

⑦ Pig ：是一个并行计算的高级的数据流语言和执行框架 ，SQL-like 语言，是在 MapReduce 上构建的一种高级查询语言，把一些运算编译进 MapReduce 模型的 Map 和 Reduce 中，并且用户可以定义自己的功能；

⑧ ZooKeeper ：Google 的 Chubby 一个开源的实现。Google 的 Chubby 一个开源的实现。它是一个针对大型分布式系统的可靠协调系统,，提供的功能包括：配置维护、名字服务、分布式同步、组服务等；ZooKeeper 的目标就是封装好复杂易出错的关键服务，将简单易用的接口和性能高效、功能稳定的系统提供给用户；

⑨ Chukwa ：一个管理大型分布式系统的数据采集系统 由 yahoo 贡献；

⑩ Cassandra ： 无单点故障的可扩展的多主数据库；

⑪ Mahout ： 一个可扩展的机器学习和数据挖掘库。

3) hadoop 的优化

① 优化的思路可以从配置文件和系统以及代码的设计思路来优化。

② 配置文件的优化：调节适当的参数，在调参数时要进行测试。

③ 代码的优化：combiner 的个数尽量与 reduce 的个数相同，数据的类型保持一致，可以减少拆包与封包的进度。

④ 系统的优化：可以设置 linux 系统打开最大的文件数预计网络的带宽 MTU 的配置。

⑤ 为 job 添加一个 Combiner，可以大大的减少 shuffer 阶段的 maoTask 拷贝过来给远程的 reduce task 的数据量，一般而言 combiner 与 reduce 相同。

⑥ 在开发中尽量使用 stringBuffer 而不是 string，string 的模式是 read-only 的，如果对它进行修改，会产生临时的对象，二 stringBuffer 是可修改的，不会产生临时对象。

⑦ 修改一下配置：以下是修改 mapred-site.xml 文件。

a) 修改最大槽位数：槽位数是在各个 tasktracker 上的 mapred-site.xml 上设置的，默认都是 2。

```
<property>
<name>mapred.tasktracker.map.tasks.maximum</name>
<value>2</value>
</property>
<property>
<name>mapred.tasktracker.reduce.tasks.maximum</name>
<value>2</value>
</property>
```

b) 调整心跳间隔：集群规模小于 300 时，心跳间隔为 300 毫秒

mapreduce.jobtracker.heartbeat.interval.min 心跳时间

mapred.heartbeats.in.second 集群每增加多少节点，时间增加下面的值

mapreduce.jobtracker.heartbeat.scaling.factor 集群每增加上面的个数，心跳增多少

c) 启动带外心跳

mapreduce.tasktracker.outofband.heartbeat 默认是 false

d) 配置多块磁盘

mapreduce.local.dir

e) 配置 RPC hander 数目

mapred.job.tracker.handler.count 默认是 10，可以改成 50，根据机器的能力

f) 配置 HTTP 线程数目

tasktracker.http.threads 默认是 40，可以改成 100 根据机器的能力

g) 选择合适的压缩方式，以 snappy 为例：

```
<property>
<name>mapred.compress.map.output</name>
<value>true</value>
</property>
<property>
<name>mapred.map.output.compression.codec</name>
<value>org.apache.hadoop.io.compress.SnappyCodec</value>
</property>
```

4) 有 10 个文件，每个文件 1G，每个文件的每一行存放的都是用户的 query，每个文件的 query 都可能重复。要求你按照 query 的频度排序，还是典型的 TOP K 算法？

- **方案一：**

  顺序读取 10 个文件，按照 hash(query)%10 的结果将 query 写入到另外 10 个文件（记为）中。这样新生成的文件每个的大小大约也 1G（假设 hash 函数是随机的）。 找一台内存在 2G 左右的机器，依次对用 hash_map(query,

query_count)来统计每个 query 出现的次数。利用快速/堆/归并排序按照出现次数进行排序。将排序好的 query 和对应的 query_cout 输出到文件中。这样得到了 10 个排好序的文件（记为）。 对这 10 个文件进行归并排序（内排序与外排序相结合）。

- **方案二：**

  一般 query 的总量是有限的，只是重复的次数比较多而已，可能对于所有的 query，一次性就可以加入到内存了。这样，我们就可以采用 trie 树/hash_map 等直接来统计每个 query 出现的次数，然后按出现次数做快速/堆/归并排序就可以了。

- **方案三：**

  与方案 1 类似，但在做完 hash，分成多个文件后，可以交给多个文件来处理，采用分布式的架构来处理（比如 MapReduce），最后再进行合并。

5) 在 2.5 亿个整数中找出不重复的整数，注，内存不足以容纳这 2.5 亿个整数。

- **方案一：**

  采用 2-Bitmap（每个数分配 2bit，00 表示不存在，01 表示出现一次，10 表示多次，11 无意义）进行，共需内存 2^32 * 2 bit=1 GB 内存，还可以接受。然后扫描这 2.5 亿个整数，查看 Bitmap 中相对应位，如果是 00 变 01，01 变 10，10 保持不变。所描完事后，查看 bitmap，把对应位是 01 的整数输出即可。

- **方案二：**

  也可采用与第 1 题类似的方法，进行划分小文件的方法。然后在小文件中找出不重复的整数，并排序。然后再进行归并，注意去除重复的元素。

6) 给 40 亿个不重复的 unsigned int 的整数，没排过序的，然后再给一个数，如何快速判断这个数是否在那 40 亿个数当中？

- **方案一：**

oo，申请 512M 的内存，一个 bit 位代表一个 unsigned int 值。读入 40 亿个数，设置相应的 bit 位，读入要查询的数，查看相应 bit 位是否为 1，为 1 表

示存在，为 0 表示不存在。

- **方案二：**

也可采用与第 1 题类似的方法，进行划分小文件的方法。然后在小文件中找出不重复的整数，并排序。然后再进行归并，注意去除重复的元素。这个问题在《编程珠玑》里有很好的描述，大家可以参考下面的思路，探讨一下： 又因为 2^32 为 40 亿多，所以给定一个数可能在，也可能不在其中； 这里我们把 40 亿个数中的每一个用 32 位的二进制来表示 ，假设这 40 亿个数开始放在一个文件中。 然后将这 40 亿个数分成两类:

① 最高位为 0

② 最高位为 1

并将这两类分别写入到两个文件中，其中一个文件中数的个数<=20 亿，而另一个>=20 亿（这相当于折半了）； 与要查找的数的最高位比较并接着进入相应的文件再查找 再然后把这个文件为又分成两类:

① 次最高位为 0

② 次最高位为 1

并将这两类分别写入到两个文件中，其中一个文件中数的个数<=10 亿，而另一个>=10 亿（这相当于折半了）； 与要查找的数的次最高位比较并接着进入相应的文件再查找。

.....

以此类推，就可以找到了,而且时间复杂度为 O(logn)

- **附：**

这里，再简单介绍下，位图方法： 使用位图法判断整形数组是否存在重复 ,判断集合中存在重复是常见编程任务之一，当集合中数据量比较大时我们通常希望少进行几次扫描，这时双重循环法就不可取了。

位图法比较适合于这种情况，它的做法是按照集合中最大元素 max 创建一个长度为 max+1 的新数组，然后再次扫描原数组，遇到几就给新数组的第几位置上 1，如遇到 5 就给新数组的第六个元素置 1，这样下次再遇到 5 想置位时发现新数组的第六个元素已经是 1 了，这说明这次的数据肯定和以前的数据存在着重复。这 种给新数组初始化时置零其后置一的做法类似于位图的处理方法故称位图法。

它的运算次数最坏的情况为 2N。如果已知数组的最大值即能事先给新数组定长的话效 率还能提高一倍。

7) 介绍哈希函数

又称为散列函数、散列算法、杂凑函数等 是一种单向密码体制：从明文到密文的不可逆映射，可将任意长度的输入变换为固定长度的输出，生成消息的“数据指纹”（也称消息摘要或散列值）， 在数据完整性认证和数字签名等领域有广泛的应用。

- **分类**

改动检测码 MDC(Manipulation Detection Code)，不带密钥哈希函数，检测消息有无篡改，消息认证码 MAC(Message Authentication Code)，带密钥哈希函数，认证消息源真实性与消息完整性。

对任意长度的输入消息，产生固定长度的输出，公式表示：

h=H(M)

M：任意长度的消息

H：哈希函数

h：固定长度的哈希值

- **性质**

输入：任意有限长度消息；输出：固定长度哈希值。

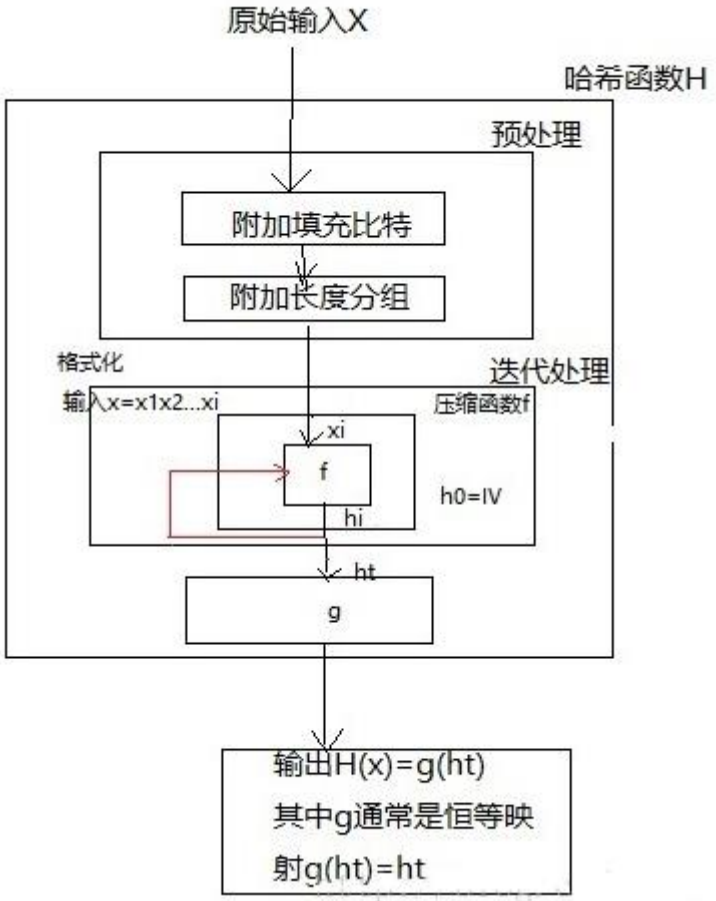
可计算性：对于任意给定消息，计算哈希值容易。

单向性：对给定哈希值 h，要找到 M 使 H(M)＝h 在计算上不可行。

抗弱碰撞性：对于给定消息 M1，找到 M2 ≠M1 ，且满足 H(M1)＝H(M2)在计算上是不可行的。

抗强碰撞性：到任何满足 H(M1)＝H(M2)的消息对(M1, M2)，在计算上是不可行的。

- **一般模型**



核心：

设计无碰撞的压缩函数 f；

敌手对算法的攻击重点是 f 的内部结构；

f 和分组密码一样是由多轮处理组成；

攻击者对各轮的位模式进行分析，找出 f 的碰撞；

f 是压缩函数，其碰撞不可避免；

设计 f 时，应保证找出碰撞在计算上是不可行的；

两个重要的迭代型哈希函数：

Hash 算法众多，MD5(128 位)和 SHA(160 位)最著名；

MD5、SHA-1。

8) 什么是 RDD 宽依赖和窄依赖

RDD 和它依赖的 parent RDD(s)的关系有两种不同的类型，即窄依赖（narrow dependency）和宽依赖（wide dependency）。

① 窄依赖指的是每一个 parent RDD 的 Partition 最多被子 RDD 的一个 Partition 使用；

② 宽依赖指的是多个子 RDD 的 Partition 会依赖同一个 parent RDD 的 Partition。

9) spark 数据处理中，什么是数据倾斜，一般有哪些方法来处理数据倾斜？

- **数据倾斜（data skew）**

是由于外部输入数据或者 spark 处理中 repartition key 本身数据分布不均导致 RDD 中各个 partition 数据分布不均匀，而在接下来 stage 的处理过程中，导致每个 partition 的处理时间也不一样，极端情况热点数据 partition 的处理时间是其他时间的几十、上百倍，从而导致无法真正的并行化进而 spark 程序处理性能骤降。

- **几类解决方案**

① 增大并行度：spark.default.parallelism， 适用大量不同的 Key 被分配到了相同的 Task 造成该 Task 数据量过大。该方法通过增加并行度将热点 parition 进一步打散，一定程度上能缓解数据倾斜，但不能根本上消除数据倾斜；

② 自定义 HashPartitioner：将原本被分配到同一个 Task 的不同 Key 分配到不同 Task。同样适用大量不同的 Key 被分配到了相同的 Task 造成该 Task 数据量过大，使用自定义的 Partitioner 实现类代替默认的 HashPartitioner，尽量将所有不同的 Key 均匀分配到不同的 Task 中，可以根本上解决上述原因导致的数据倾斜问题，但对于同一个 key 对应数据集非常大的场景，还是无法解决；

③ 将 Reduce side Join 转变为 Map side Join：该方法主要针对在需要 join 的场景，在采用 reduce side join 的时候，通常需要对两个表都按 key 进行 shuffle，并且如果大表中的 key 分布不均匀的情况下，从而导致数据倾斜。如果需要 join 的另一张表是小表（也就是足够小到可 broadcast，不会导致 executor 内存 OOM），可以采用 Map side join，这种场景下所有 join 都在 map 阶段就完成，从而避免进行按 key shuffle；

④ 将 skew 的 key 单独进行处理，并为 skew 的 key 增加随机前缀或者后缀：这种方法本质是一种两部法，把原先的 RDD 拆为 UnSkewRDD 和 SkewRDD，对于 UnSkewRDD 仍旧正常处理，对于 SkewRDD 的 key 进行一定的处理（比

如随机前缀或者后缀），然后进一步 repartition，以增加处理并发度，最后将两个 UnSkewResultRDD 和 SkewResultRDD 进行 Union，从而根本上消除数据倾斜的问题。这种方法相对前面的方法相对更复杂，一个是需要对数据进行细致的分析，发现 skew key 的特点并过滤出来，另一个在处理流程上需要分别处理，增加了代码的开发复杂度，但在效果是确实是根本上消除了数据倾斜，比较常用在两个大表之间的 join 上。

10) 说明下 spark streaming 中 updateStateByKey 和 mapWithState 两个操作，以及两个操作在实现上有什么不同？

对于 updateStateByKey： 统计全局的 key 的状态，但是就算没有数据输入，他也会在每一个批次的时候返回之前的 key 的状态。

这样的缺点是如果数据量太大的话，我们需要 checkpoint 数据会占用较大的存储。而且效率也不高。

对于 mapWithState：也是用于全局统计 key 的状态，但是它如果没有数据输入，便不会返回之前的 key 的状态，有一点增量的感觉。这样做的好处是，我们可以只是关心那些已经发生的变化的 key，对于没有数据输入，则不会返回那些没有变化的 key 的数据。这样的话，即使数据量很大，checkpoint 也不会像 updateStateByKey 那样，占用太多的存储。

11) 例举出几种 Storm 中的 stream grouping 的类型

- **Shuffle Grouping**

随机分组(随机派发 stream 里面的 tuple，保证每个 bolt 接收到的 tuple 数目大致相同)。

- **Fields Grouping**

字段分组(指定某个字段作为分组条件，那么字段值相同的 tuple 会被分到同一个 Blot 的 task 进行处理)。

- **All Grouping**

全部分组(就是对于每一个 tuple，所有的 bolt 都会收到)。

- **None Grouping**

无分组(这种分组方法和第一种 shuffle Grouping 效果是一样的)。

- **Global Grouping**

全局分组(spout/bolt 发射的所有 tuple 全部进入下游 bolt 的同一个 task，通常选择下游 bolt 中 id 最小的 task)。

- **Direct Grouping**

直接分组(允许 spout/bolt 决定其发射出的任一 tuple 由下游 bolt 的哪个 task 接收并处理)。

12) Yarn 调优

- **CPU 配置**

yarn.app.mapreduce.am.resource.cpu-vcores // ApplicationMaster 虚拟 CPU 内核(建议设置成 1)

yarn.nodemanager.resource.cpu-vcores // 容器虚拟 CPU 内核(建议 cpu 全部内核数都给他-默认值 8)

- **内存配置**

yarn.nodemanager.resource.memory-mb // 容器内存(24G)

mapreduce.map.memory.mb // Map 任务内存(1G)

mapreduce.reduce.memory.mb // Reduce 任务内存(1G)

yarn.scheduler.minimum-allocation-mb // 最小容器内存(1G)

yarn.scheduler.maximum-allocation-mb // 最大容器内存(24G)

yarn.scheduler.increment-allocation-mb // 容器内存增量(512M)

- **同一个 Map 或者 Reduce 并行执行(如果有较慢的任务，会在别的机器上启动一个相同的 map 或 reduce 重复执行，谁成功了就把落后的 kill 掉)**

mapreduce.map.speculative // Map 任务推理执行(默认是不勾选的，建议勾选)；

mapreduce.reduce.speculative // Reduce 任务推理执行(默认是不勾选的，建议勾选)。

- **JVM 重用**

mapreduce.job.ubertask.enable // 启用 Ubertask 优化(默认是不勾选的，建议勾选)；

mapreduce.job.ubertask.maxmaps // Ubertask 最大 Map -> 超过多少个 map 启用 jvm 重用；

mapreduce.job.ubertask.maxreduces // Ubertask 最大 Reduce -> 超过多少 Reduce 启用 jvm 重用；

mapreduce.job.ubertask.maxbytes // Ubertask 最大作业大小 -> application 的输入大小的阀值，默认为 block 大小。

13) HDFS 工作原理

- **写入数据**

① 客户端通过调用 DistributedFileSystem 的 create 方法创建新文件；

② DistributedFileSystem 通过 RPC 调用 namenode 去创建一个新文件，创建前，namenode 会做各种校验，比如文件是否存在，客户端有无权限等；

③ 前两步结束后，会返回 FSDataOutputStream 的对象，客户端开始写数据到 DFSOutputStream，把数据切成一个个小的 packet，然后排成队列 data quene；

④ DataStreamer 会去处理接受 data quene，如果副本数是 3，那么就找到 3 个 datanode，把他们排成一个 pipeline。DataStreamer 把 packet 按队列输出到管道的第一个 datanode 中，第一个 datanode 又把 packet 输出到第二个 datanode 中，以此类推；

⑤ DFSOutputStream 还有一个对列叫 ack quene，也是由 packet 组成，等待 datanode 的收到响应，当 pipeline 中的所有 datanode 都表示已经收到的时候，这时 akc quene 才会把对应的 packet 包移除掉；

⑥ 客户端完成写数据后调用 close 方法关闭写入流；

⑦ 收到最后一个 ack 后，通知 datanode 把文件标视为已完成。

- **源码分析**

客户端需要获得 FSDataOutputStream 流,需要调用 FileSystem.create 方法,这里面又调用了 DistributedFileSystem.create 方法,接着又调用了 DFSClient.create 的方法,DFSClient 可以连接到 Hadoop 的文件系统，它里面有一个字段是 ClientProtocol(和 NameNode 进程沟通和连接,读/写数据块),而 ClientProtocol 是通过一个代理类 ProxyAndInfo 创建的,

创建的时候使用了 java.net.InetSocketAddress(如果是主机名+端口号将尝试解析主机名)它继承了 SocketAddress。

14) Kafka 简介及各个组件介绍

- **介绍**

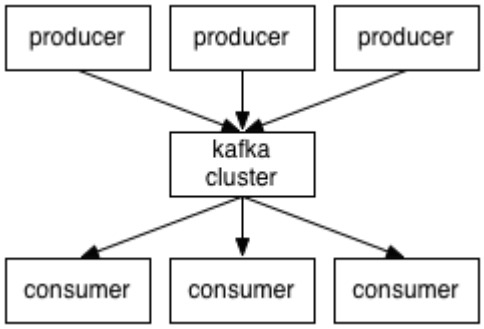
Kafka 是一种分布式发布-订阅消息系统，它提供了一种独特的消息系统功能。

① Kafka 维护的消息流称为 topic；

② 发布消息者称为 producer；

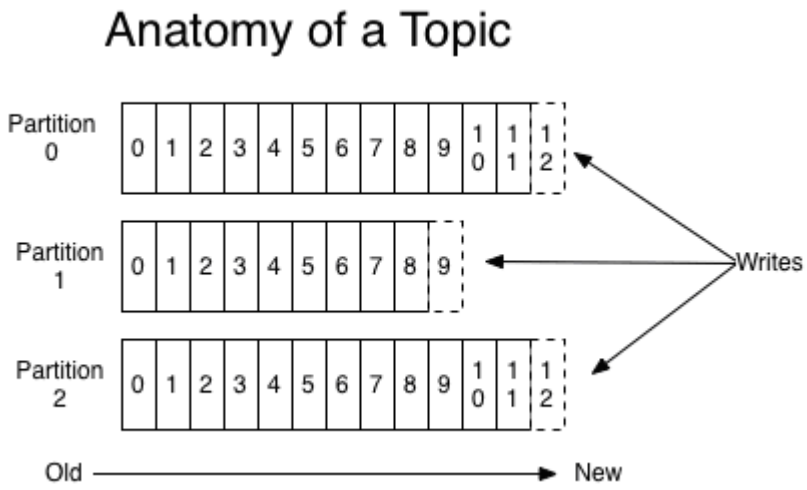
③ 订阅并消费消息的称为 consumers；

④ Kafka 运行在多 server 的集群之上，每个 server 称为 broker。



- **组件**

① Topics and Logs



一个 Topic 可以认为是一类消息，Kafka 集群将每个 topic 将被分成多个 partition(区)，逻辑上如上图所示。

每一个 partition 都是一个有序的、不可变的消息序列，它在存储层面是以 append log 文件形式存在的。任何发布到此 partition 的消息都会被直接追加到 log 文件的尾部。每条消息在文件中的位置称为 offset(偏移量)，offset 为一个 long 型数字，它是唯一标记一条消息。

Kafka 集群保留了所有以发布消息，即使消息被消费，消息仍然会被保留一段时间。例如，如果 log 被设置为保留两天，那么在一条消息被消费之后的两天内仍然有效，之后它将会被丢弃以释放磁盘空间。Kafuka 的性能相对于数据量来说是恒定的，所以保留大量的数据并不是问题。

每个 consumer(消费者)的基础元数据只有一个，那就是 offset，它表示消息在 log 文件中的位置，它由 consumer 所控制，通常情况下，offset 将会”线性”的向前驱动，也就是说消息将依次顺序被消费。而事实上，consumer 可以通过设置 offset 来消费任意位置的消息。例如，consumer 可以重置 offset 来从新处理消息。这些特性意味着 Kafkaconsumer 非常轻量级，它可以随意切入和离开，而不会对集群里其他的 consumer 造成太大的影响。比如，你可以使用 tail 命令工具来查看任意 topic 的内容，而不会影响消息是否被其他 consumer 所消费。在消息系统中采用 Partitions 设计方式的目的有多个。首先，允许更大的数据容量，每个 topic 可以拥有多个 partitions，每个独立的 patition 运行于 servers 之上，因此，topic 几乎能够容纳任意大小的数据量。第二点，partitions 都是并行单位。

② Partition

Kafka 集群中，一个 Topic 的多个 partitions 被分布在多个 server 上。每个 server 负责 partitions 中消息的读写操作。每个 partition 可以被备份到多台 server 上，以提高可靠性。

每一个 patition 中有一个 leader 和若干个 follower。leader 处理 patition 内所有的读写请求，而 follower 是 leader 的候补。如果 leader 挂了，其中一个 follower 会自动成为新的 leader。每一台 server 作为担任一些 partition 的 leader，同时也担任其他 patition 的 follower，以此达到集群内的负载均衡。

Producer 将消息发送的指定 topic 中，producer 决定将消息发送到哪个 partition 中。比如基于”round-robin”方式实现简单的负载均衡或者通过其他

的一些算法等。

③ Consumers

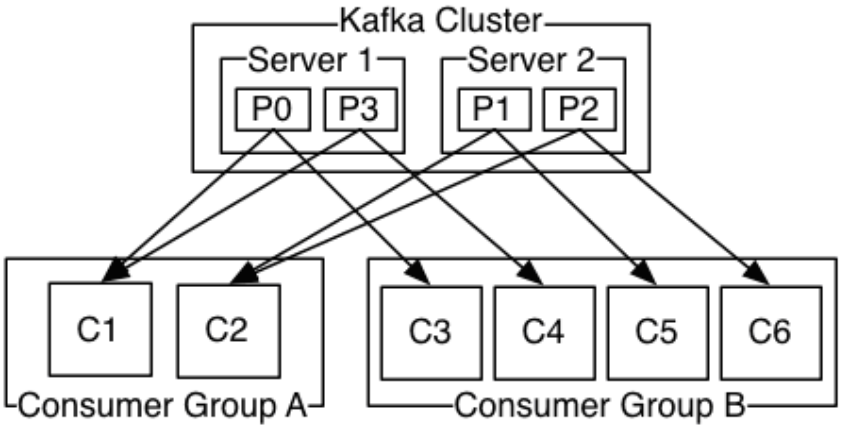
消息基本上有两种模式：queuing(队列模式) 和 publish-subscribe(发布-订阅模式) ， 在队列模式中，consumer 池从 server 中读取消息，每个消息都会到达一个 consumer。在发布-订阅模式中，消息被广播到所有的 consumer。Kafka 提供了 consumer group 这个抽象概念来概括这两种模式。

每个 consumer 属于一个 consumer group, 如果 consumer group 订阅了 topic，那么它会接收到该 topic 发布的每条消息，该消息只会被分配到一个 consumer 上。consumer 实例可以部署在不同的进程或机器上。

如果所有的 consumer 都具有相同的 group，这种情况和 queue 模式很像，消息将会在 consumers 之间负载均衡。

如果所有的 consumer 都具有不同的 group，那这就是”发布-订阅”，消息将会广播给所有的消费者。然而，我们发现大多数情况下 topic 只有少量的逻辑上的订阅者 consumer group，每个 group 由许多的 consumer 实例组成，以提高扩展性和容错性。这就是发布-订阅模式，订阅者是 consumer 集群而非单个进程。相比于传统的消息系统，Kafka 具有更强的序列保证。

传统的队列在 server 上保持有序，如果多个 consumer 从队列中消费，队列会按序弹出，然后消息被异步分配到 consumer 上，因此，消息到达 consumer 时可能会破坏顺序。这意味着在并行处理过程中，消息处理是无序的。为了解决这个问题，消息系统的 exclusive consumer 机制只允许单进程从队列中消费消息，当然，这就是说，没有了并行处理能力。



Kafka 具有更好的解决方案。通过 parallelism—the partition—within the topics 机制，Kafka 能够提提供有序保证，使 consumer 池能够负载均衡。这是通过把 topic 中的 partition 分派给 consumer group 中的 consumer 来实现的，因此，每个 partition 由 group 中一个确定的 consumer 来消费。通过这种方式我们保证了 consumer 是指定 partition 的唯一 reader，并且按顺序消费数据。由于有很多 partition，这种方式使得 consumer 实例可以负载均衡。

kafka 只能保证一个 partition 中的消息被某个 consumer 消费时，消息是顺序的。事实上，从 Topic 角度来说，消息仍不是有序的。如果你需要 topic 范围内的有序，那么你可以只使用一个 partition，这也就是说，group 中也只有一个 consumer。

④ Guarantees

在更高的层面，Kafka 给出以下保证：

1) 发送到 partitions 中的消息将会按照它接收的顺序追加到日志中；

2) 对于消费者而言,它们消费消息的顺序和 log 中消息顺序一致；

3) 如果 Topic 的”replication factor“为 N，那么允许 N-1 个 kafka 实例失效。

15) spark stage 如何划分

① DAGScheduler 中的 handleJobSubmitted 方法根据最后一个 Rdd（所谓最后一个 Rdd 指的是 DAG 图中触发 action 操作的 Rdd）生成 finalStage；

② 生成 finalStage 的过程中调用了 getParentStagesAndId 方法，通过该方法，从最后一个 Rdd 开始向上遍历 Rdd 的依赖（可以理解为其父 Rdd），如果遇到其依赖为 shuffle 过程，则调用 getShuffleMapStage 方法生成该 shuffle 过程所在的 stage。完成 Rdd 遍历后，所有的 stage 划分完成；

③ getShuffleMapStage 方法从传入的 Rdd 开始遍历，直到遍历到 Rdd 的依赖为 shuffle 为止，生成一个 stage；

④ stage 的划分就此结束。