

# 深度学习 500 问



# 目录

Chapter 1. 数学基础	9
1. 向量和矩阵	9
2. 导数和偏导数	13
3. 特征值和特征向量	15
4. 概率分布与随机变量	16
5. 常见概率分布	20
6. 期望、方差、协方差、相关系数	22
Chapter 2. 机器学习基础	25
1. 基本概念	25
2. 机器学习学习方式	27
3. 分类算法	29
4. 2.9 逻辑回归	40
5. 2.10 代价函数	42
6. 2.11 损失函数	48
7. 梯度下降	52
8. 2.14 线性判别分析 (LDA)	56
9. 主成分分析 (PCA)	62
10. 模型评估	67
11. 决策树	77
12. 支持向量机	79
13. 贝叶斯分类器	85
14. EM 算法	89
15. 降维和聚类	91
Chapter 3. 深度学习基础	99
1. 基本概念	99
2. 网络操作与计算	101

3. 超参数	103
4. 激活函数	104
5. 3.5 Batch_Size	110
6. 3.6 归一化	111
7. 3.7 预训练与微调 (fine tuning)	116
8. 3.8 权重偏差初始化	117
9. 3.9 学习率	119
10. Dropout 系列问题	121
11. 深度学习中常用的数据增强方法?	123
12. 如何理解 Internal CovariateShift?	123
3. 4.3 ZFNet	128
4. 4.4 Network in Network	129
6. 4.6 GoogLeNet	133
7. Restnet	136
8. Densenet	136
9. 4.7 为什么现在的 CNN 模型都是在 GoogleNet、VGGNet 或者 AlexNet 上调整的?	136
Chapter 5. 循环神经网络 (RNN)	141
Chapter 6. 循环神经网络 (RNN)	155
1. 6.1 为什么需要 RNN?	155
2. 6.2 图解 RNN 基本结构	155
3. 6.3 RNNs 典型特点?	157
4. 6.4 CNN 和 RNN 的区别?	158
5. 6.5 RNNs 和 FNNs 有什么区别?	158
6. 6.6 RNNs 训练和传统 ANN 训练异同点?	158
7. 6.7 为什么 RNN 训练的时候 Loss 波动很大	159
8. 6.8 标准 RNN 前向输出流程	159
9. 6.9 BPTT 算法推导	159
10. 6.9 RNN 中为什么会出现梯度消失?	160
11. 6.10 如何解决 RNN 中的梯度消失问题?	161
12. 6.11 LSTM	161
13. 6.12 LSTMs 与 GRUs 的区别	163
14. 6.13 RNNs 在 NLP 中典型应用?	163
15. 6.13 常见的 RNNs 扩展和改进模型	163

Chapter 7. 生成对抗网络	169
1. 7.1 GAN 基本概念	169
2. 7.2 GAN 的生成能力评价	172
3. 7.3 其他常见的生成式模型有哪些?	174
4. 7.4 GAN 的改进与优化	175
5. 7.3 GAN 的应用 (图像翻译)	178
6. 7.4 GAN 的应用 (文本生成)	181
7. 7.5 GAN 在其他领域的应用	181
6. 8.6 目标检测的常用数据集	194
7. 8.7 目标检测常用标注工具	195
8. 8.8 目标检测工具和框架 (贡献者: 北京理工大学-明奇)	197
9. TODO	198
8. 9.7 PSPNet	204
9. 9.8 DeepLab 系列	205
14. 9.13 全景分割 (贡献者: 北京理工大学-明奇)	209
Chapter 11. 迁移学习	223
1. 迁移学习基础知识	223
Chapter 12. 网络搭建及训练	231
1. TensorFlow	231
2. TensorFlow 是什么?	231
3. TensorFlow 的设计理念是什么?	231
4. TensorFlow 特点有哪些?	232
6. TensorFlow 编程模型是怎样的?	233
7. 如何基于 tensorflow 搭建 VGG16	237
8. Pytorch	238
9. Pytorch 是什么?	238
10. 为什么选择 Pytorch?	238
11. PyTorch 的架构是怎样的?	239
12. Pytorch 与 tensorflow 之间的差异在哪里?	239
13. Pytorch 有哪些常用工具包?	240
14. Caffe	240
15. 什么是 Caffe?	240
16. Caffe 的特点是什么?	240

17. Caffe 的设计思想是怎样的?	240
18. Caffe 架构是怎样的?	241
19. Caffe 的有哪些接口?	243
20. 有哪些经典的网络模型值得我们去学习的?	244
21. 10.6 网络训练有哪些技巧吗?	248
Chapter 13. 优化算法	253
1. 如何解决训练样本少的问题	253
2. 深度学习是否能胜任所有数据集?	253
3. 有没有可能找到比已知算法更好的算法?	253
4. 什么是共线性, 如何判断和解决共线性问题?	254
5. 权值初始化方法有哪些?	254
6. 如何防止梯度下降陷入局部最优解?	255
7. 为什么需要激活函数?	256
8. 常见的损失函数有哪些?	257
9. 如何进行特征选择 (feature selection)?	258
10. 梯度消失/梯度爆炸原因, 以及解决方法	259
11. 深度学习为什么不用二阶优化?	262
12. 为什么要设置单一数字评估指标, 设置指标的意义?	262
13. 训练/验证/测试集的定义及划分	262
14. 什么是 TOP5 错误率?	263
15. 什么是泛化误差, 如何理解方差和偏差?	263
16. 如何提升模型的稳定性?	264
17. 有哪些改善模型的思路	264
18. 如何快速构建有效初始模型?	265
19. 如何通过模型重新观察数据?	266
20. 如何解决数据不匹配问题?	266
1. 超参数概念	271
2. 14.3 网络训练中的超参调整策略	274
3. 14.4 合理使用预训练网络	277
4. 14.5 如何改善 GAN 的性能	279
Chapter 15. 第十五章异构计算, GPU 和框架选型指南	285
Chapter 16. 第十六章 NLP	297

1. 16.0 NLP 发展史简述	297
2. 16.1 如何理解序列到序列模型?	302
3. 16.2 序列到序列模型有什么限制吗?	302
4. 16.3 如果不采用序列到序列模型, 可以考虑用其它模型方法吗?	302
5. 16.4 如何理解词向量?	302
6. 16.5 词向量哪家好?	302
7. 16.6 解释一下注意力机制的原理?	302
8. 16.7 注意力机制是不是适用于所有场景呢? 它的鲁棒性如何?	302
9. 16.8 怎么将原有的模型加上注意力机制呢?	302
10. 16.9 通俗地解释一下词法分析是什么? 有什么应用场景?	302
11. 16.10 深度学习中的词法分析有哪些常见模型呢?	302
12. 16.11 通俗地解释一下知识图谱是什么? 有什么应用场景?	302
13. 16.12 深度学习中的知识图谱有哪些常见模型呢?	302
14. 16.13 深度学习中的机器翻译有哪些常见模型呢?	302
15. 16.14 机器翻译的通俗实现以及部署过程是怎样的呢?	302
16. 16.15 通俗地解释一下文本情感分析是什么? 常见的应用场景是?	302
17. 16.16 最常用的情感分析模型是什么呢? 如何快速部署呢?	302
18. 16.17 通俗地解释一下问答系统? 它涵盖哪些领域? 常见的应用场景是?	302
19. 16.18 常见的问答系统模型是什么? 如何快速部署呢?	302
20. 16.19 图像文字生成是什么? 它的技术原理是什么?	302
21. 16.20 常见的图像文字生成模型是什么?	302
22. 16.21 NLP 的无监督学习发展动态是怎样的? 有哪些领域在尝试无监督学习?	302
23. 16.22 NLP 和强化学习的结合方式是怎样的? 有哪些方向在尝试强化学习?	302
24. 16.23 NLP 和元学习? 元学习如何能够和 NLP 结合起来?	302
25. 16.24 能说一下各自领域最常用且常见的基准模型有哪些吗?	302
 Chapter 17. 模型压缩及移动端部署	305
 Chapter 18. 第十八章 __ 后端架构选型、离线及实时计算	353
1. 18.1 为什么需要分布式计算?	353
5. 18.5 如何进行离线计算?	364
6. 18.6 如何使用分布式框架提高模型训练速度?	364
7. 18.7 深度学习分布式计算框架如何在移动互联网中应用?	364
8. 18.8 如何在个性化推荐中应用深度学习分布式框架?	364
9. 18.9 如何评价个性化推荐系统的效果?	364

Chapter 19. 第十八章后端架构选型及应用场景	369
1. 18.1 为什么需要分布式计算?	369
4. 18.4 如何进行离线计算?	378
5. 18.5 如何设计一个人机交互系统?	390
6. 18.6 如何设计个性化推荐系统?	394

## CHAPTER 1

# 数学基础

深度学习通常需要哪些数学基础?<sup>1</sup> 深度学习里的数学到底难在哪里? 通常初学者都会有这些问题, 在网络推荐及书本推荐里, 经常看到会列出一系列数学科目, 比如微积分、线性代数、概率论、复变函数、数值计算、最优化理论、信息论等等.<sup>2 3</sup> 这些数学知识有相关性, 但实际上按照这样的知识范围来学习, 学习成本会很久, 而且会很枯燥, 本章我们通过选举一些数学基础里容易混淆的一些概念做以介绍, 帮助大家更好的理清这些易混淆概念之间的关系.

## 1. 向量和矩阵

### 1.1. 标量、向量、矩阵、张量之间的联系.

#### 1.1.1. 标量.<sup>4</sup>

定义 1.1. 一个标量表示一个单独的数, 它不同于线性代数中研究的其他大部分对象(通常是多个数的数组). 我们用斜体表示标量. 标量通常被赋予小写的变量名称.

#### 1.1.2. 向量.

定义 1.2. 一个向量表示一组有序排列的数.<sup>5</sup> 通过次序中的索引, 我们可以确定每个单独的数. 通常我们赋予向量粗体的小写变量名称, 比如  $\mathbf{x}$ . 向量中的元素可以通过带脚标的斜体表示. 向量  $\mathbf{X}$  的第一个元素是  $\mathbf{X}_1$ , 第二个元素是  $\mathbf{X}_2$ , 以此类推. 我们也会注明存储在向量中的元素的类型(实数、虚数等).<sup>6</sup>

#### 1.1.3. 矩阵.

定义 1.3. 矩阵是具有相同特征和纬度的对象的集合, 表现为一张二维数据表. 其意义是一个对象表示为矩阵中的一行, 一个特征表示为矩阵中的一列, 每个特征都有数值型的取值. 通常会赋予矩阵粗体的大写变量名称, 比如  $\mathbf{A}$ .

<sup>1</sup>删掉了又

<sup>2</sup>数学, 计算机文献中, 所有的句号都改成了点.

<sup>3</sup>优化理论改为了最优化理论

<sup>4</sup>英语是在第一次出现概念的时候出现, 所以这儿都删去了.

<sup>5</sup>在数学中, 向量一般指线性空间中方的元素.

<sup>6</sup>存储

### 1.1.4. 张量.

定义 1.4. 在某些情况下，我们会讨论坐标超过两维的数组。一般地，一个数组中的元素分布在若干维坐标的规则网格中，我们将其称之为张量。使用  $A$  来表示张量。<sup>7</sup> 张量  $A$  中坐标为  $(i, j, k)$  的元素记作  $A_{(i,j,k)}$ .

### 1.1.5. 四者之间关系.

标量是 0 阶张量，向量是一阶张量。举例：

- 标量就是知道棍子的长度，但是你不会知道棍子指向哪儿。
- 向量就是不但知道棍子的长度，还知道棍子指向前面还是后面。
- 张量就是不但知道棍子的长度，也知道棍子指向前面还是后面，还能知道这棍子又向上/下和左/右偏转了多少。

## 1.2. 张量与矩阵的区别.

- 从代数角度讲，矩阵它是向量的推广。向量可以看成一维的“表格”（即分量按照顺序排成一排），矩阵是二维的“表格”（分量按照纵横位置排列），那么  $n$  阶张量就是所谓的  $n$  维的“表格”。张量的严格定义是利用线性映射来描述。
- 从几何角度讲，矩阵是一个真正的几何量，也就是说，它是一个不随参照系的坐标变换而变化的东西。向量也具有这种特性。
- 张量可以用  $3 \times 3$  矩阵形式来表达。<sup>8</sup>
- 表示标量的数和表示向量的三维数组也可分别看作  $1 \times 1$ ,  $1 \times 3$  的矩阵。<sup>9</sup>

**1.3. 矩阵和向量相乘结果.** 若使用爱因斯坦求和约定 (Einstein summation convention)，矩阵  $A, B$  相乘得到矩阵  $C$  可以用下式表示：

$$(1) \quad a_{ik} * b_{kj} = c_{ij}, \text{<sup>10</sup>}$$

其中， $a_{ik}, b_{kj}, c_{ij}$  分别表示矩阵  $A, B, C$  的元素， $k$  出现两次，是一个哑变量 (Dummy Variables) 表示对该参数进行遍历求和。而矩阵和向量相乘可以看成是矩阵相乘的一个特殊情况，例如：矩阵  $B$  是一个  $n \times 1$  矩阵。<sup>11</sup>

## 1.4. 向量和矩阵的范数归纳.

<sup>7</sup>删掉了“A”

<sup>8</sup>这儿有错误，矩阵怎么表示张量。

<sup>9</sup>这儿也有错误。

<sup>10</sup>去掉了 tag，公式后面也要加逗号。

<sup>11</sup> $n \times 1$  的矩阵改成了  $n \times 1$  矩阵，这个更符合大家的习惯。

1.4.1. 向量的范数. 定义一个向量为:  $\vec{a} = [-5, 6, 8, -10]$ . 任意一组向量设为  $\vec{x} = (x_1, x_2, \dots, x_N)$ . 其不同范数求解如下:

定义 1.5. 向量的 1 范数 向量的各个元素的绝对值之和, 既

$$(2) \quad \|\vec{x}\|_1 = \sum_{i=1}^N |x_i|, \text{<sup>12</sup>}$$

上述向量  $\vec{a}$  的 1 范数结果就是: 29.

定义 1.6. 向量的 2 范数 向量的每个元素的平方和再开平方根, 既

$$(3) \quad \|\vec{x}\|_2 = \sqrt{\sum_{i=1}^N |x_i|^2}, \text{<sup>13</sup>}$$

上述  $\vec{a}$  的 2 范数结果就是: 15.

定义 1.7. 向量的负无穷范数 向量的所有元素的绝对值中最小的, 既

$$(4) \quad \|\vec{x}\|_{-\infty} = \min |x_i|, \text{<sup>14</sup>}$$

上述向量  $\vec{a}$  的负无穷范数结果就是: 5.

定义 1.8. 向量的正无穷范数 向量的所有元素的绝对值中最大的

$$(5) \quad \|\vec{x}\|_{+\infty} = \max |x_i|,$$

上述向量  $\vec{a}$  的正无穷范数结果就是: 10.

定义 1.9. 向量的 p 范数 向量的 p 范数定义为: <sup>15</sup>

$$(6) \quad L_p = \|\vec{x}\|_p = \sqrt[p]{\sum_{i=1}^N |x_i|^p},$$

<sup>12</sup>加上了逗号.

<sup>13</sup>加上了逗号.

<sup>14</sup>加上了逗号

<sup>15</sup>加了一句向量的 p 范数定义为.

### 1.4.2. 矩阵的范数. 定义一个矩阵

$$A = \begin{pmatrix} -1 & 2 & -3 \\ 4 & -6 & 6 \end{pmatrix},$$

任意矩阵定义为:  $A_{m \times n}$ , 其元素为  $a_{ij}$ . 矩阵的范数定义为

$$(7) \quad \|A\|_p := \sup_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p},$$

当向量取不同范数时, 相应得到了不同的矩阵范数.

定义 1.10. 矩阵的 1 范数 (列范数) 矩阵的每一列上的元素绝对值先求和, 再从中取个最大的, (列和最大),

$$(8) \quad \|A\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^m |a_{ij}|,$$

上述矩阵  $A$  的 1 范数先得到  $[5, 8, 9]$ , 再取最大的最终结果就是: 9.

定义 1.11. 矩阵的 2 范数 矩阵  $A^T A$  的最大特征值开平方根,

$$(9) \quad \|A\|_2 = \sqrt{\lambda_{\max}(A^T A)},$$

其中,  $\lambda_{\max}(A^T A)$  为  $A^T A$  的特征值绝对值的最大值. -

上述矩阵  $A$  的 2 范数得到的最终结果是: 10.0623.

定义 1.12. 矩阵的无穷范数 (行范数) 矩阵的每一行上的元素绝对值先求和, 再从中取个最大的, (行和最大),

$$(10) \quad \|A\|_\infty = \max_{1 \leq i \leq m} \sum_{j=1}^n |a_{ij}|,$$

上述矩阵  $A$  的行范数先得到 [616], 再取最大的最终结果就是: 16.

定义 1.13. 矩阵的核范数 矩阵的奇异值 (将矩阵 svd 分解) 之和, 这个范数可以用来低秩表示 (因为最小化核范数, 相当于最小化矩阵的秩——低秩), 上述矩阵  $A$  最终结果就是: 10.9287.

定义 1.14. 矩阵的  $L_0$  范数<sup>16</sup> 矩阵的非 0 元素的个数, 通常用它来表示稀疏,  $L_0$  范数越小 0 元素越多, 也就越稀疏, 上述矩阵  $A$  最终结果就是: 6.

---

<sup>16</sup> $L_0$  应该是下标

定义 1.15. 矩阵的  $L_1$  范数<sup>17</sup> 矩阵中的每个元素绝对值之和，它是  $L_0$  范数的最优凸近似，因此它也可以表示稀疏，上述矩阵  $A$  最终结果就是：22.

定义 1.16. 矩阵的 F 范数 矩阵的各个元素平方之和再开平方根，

$$(11) \quad \|A\|_F = \sqrt{\left(\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2\right)},$$

它通常也叫做矩阵的  $L_2$  范数，它的优点在于它是一个凸函数，可以求导求解，易于计算，上述矩阵  $A$  最终结果就是：10.0995.

定义 1.17. 矩阵的  $L_{21}$  范数 矩阵先以每一列为单位，求每一列的 F 范数（也可认为是向量的 2 范数），然后再将得到的结果求  $L_1$  范数（也可认为是向量的 1 范数），很容易看出它是介于  $L_1$  和  $L_2$  之间的一种范数，上述矩阵  $A$  最终结果就是：17.1559.

定义 1.18. 矩阵的  $p$  范数 矩阵的  $p$  范数定义为

$$(12) \quad \|A\|_p = \sqrt[p]{\left(\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^p\right)},$$

**1.5. 如何判断一个矩阵为正定.** 判定一个矩阵是否为正定，通常有以下几个方面：

- 顺序主子式全大于 0；
- 存在可逆矩阵  $C$  使  $C^T C$  等于该矩阵；
- 正惯性指数等于  $n$ ；
- 合同于单位矩阵  $E$ （即：规范形为  $E$ ）
- 标准形中主对角元素全为正；
- 特征值全为正；
- 是某基的度量矩阵.

## 2. 导数和偏导数

### 2.1. 导数偏导计算.

---

<sup>17</sup>这儿也应该是下标

2.1.1. 导数定义. 导数 (derivative) 代表了在自变量变化趋于无穷小的时候, 函数值的变化与自变量的变化的比值. 几何意义是这个点的切线. 物理意义是该时刻的 (瞬时) 变化率.

注意: 在一元函数中, 只有一个自变量变动, 也就是说只存在一个方向的变化率, 这也就是为什么一元函数没有偏导数的原因. 在物理学中有平均速度和瞬时速度之说. 平均速度有

$$v = \frac{s}{t}$$

其中  $v$  表示平均速度,  $s$  表示路程,  $t$  表示时间. 这个公式可以改写为

$$\bar{v} = \frac{\Delta s}{\Delta t} = \frac{s(t_0 + \Delta t) - s(t_0)}{\Delta t}$$

其中  $\Delta s$  表示两点之间的距离, 而  $\Delta t$  表示走过这段距离需要花费的时间. 当  $\Delta t$  趋向于 0 ( $\Delta t \rightarrow 0$ ) 时, 也就是时间变得很短时, 平均速度也就变成了在  $t_0$  时刻的瞬时速度, 表示成如下形式:

$$v(t_0) = \lim_{\Delta t \rightarrow 0} \bar{v} = \lim_{\Delta t \rightarrow 0} \frac{\Delta s}{\Delta t} = \lim_{\Delta t \rightarrow 0} \frac{s(t_0 + \Delta t) - s(t_0)}{\Delta t}$$

实际上, 上式表示的是路程  $s$  关于时间  $t$  的函数在  $t = t_0$  处的导数. 一般的, 这样定义导数: 如果平均变化率的极限存在, 即有

$$\lim_{\Delta x \rightarrow 0} \frac{\Delta y}{\Delta x} = \lim_{\Delta x \rightarrow 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}$$

则称此极限为函数  $y = f(x)$  在点  $x_0$  处的导数. 记作  $f'(x_0)$  或  $y'|_{x=x_0}$  或  $\frac{dy}{dx}|_{x=x_0}$  或  $\frac{df(x)}{dx}|_{x=x_0}$ .

通俗地说, 导数就是曲线在某一点切线的斜率.

### 偏导数:

既然谈到偏导数 (partial derivative), 那就至少涉及到两个自变量. 以两个自变量为例,  $z = f(x, y)$ , 从导数到偏导数, 也就是从曲线来到了曲面. 曲线上的一点, 其切线只有一条. 但是曲面上的一点, 切线有无数条. 而偏导数就是指多元函数沿着坐标轴的变化率.

注意: 直观地说, 偏导数也就是函数在某一点上沿坐标轴正方向的的变化率.

设函数  $z = f(x, y)$  在点  $(x_0, y_0)$  的领域内有定义, 当  $y = y_0$  时,  $z$  可以看作关于  $x$  的一元函数  $f(x, y_0)$ , 若该一元函数在  $x = x_0$  处可导, 即有

$$\lim_{\Delta x \rightarrow 0} \frac{f(x_0 + \Delta x, y_0) - f(x_0, y_0)}{\Delta x} = A$$

函数的极限  $A$  存在. 那么称  $A$  为函数  $z = f(x, y)$  在点  $(x_0, y_0)$  处关于自变量  $x$  的偏导数, 记作  $f_x(x_0, y_0)$  或  $\frac{\partial z}{\partial x}|_{y=y_0}$  或  $\frac{\partial f}{\partial x}|_{y=y_0}$  或  $z_x|_{y=y_0}^{x=x_0}$ .

偏导数在求解时可以将另外一个变量看做常数，利用普通的求导方式求解，比如  $z = 3x^2 + xy$  关于  $x$  的偏导数就为  $z_x = 6x + y$ ，这个时候  $y$  相当于  $x$  的系数。

某点  $(x_0, y_0)$  处的偏导数的几何意义为曲面  $z = f(x, y)$  与面  $x = x_0$  或面  $y = y_0$  交线在  $y = y_0$  或  $x = x_0$  处切线的斜率。

**2.2. 导数和偏导数有什么区别？** 导数和偏导没有本质区别，如果极限存在，都是当自变量的变化量趋于 0 时，函数值的变化量与自变量变化量比值的极限。

- 一元函数，一个  $y$  对应一个  $x$ ，导数只有一个。
- 二元函数，一个  $z$  对应一个  $x$  和一个  $y$ ，有两个导数：一个是  $z$  对  $x$  的导数，一个是  $z$  对  $y$  的导数，称之为偏导。
- 求偏导时要注意，对一个变量求导，则视另一个变量为常数，只对改变量求导，从而将偏导的求解转化成了一元函数的求导。

### 3. 特征值和特征向量

#### 3.1. 特征值分解与特征向量.

- 特征值分解可以得到特征值 (eigenvalues) 与特征向量 (eigenvectors)；
- 特征值表示的是这个特征到底有多重要，而特征向量表示这个特征是什么。

如果说一个向量  $\vec{v}$  是方阵  $A$  的特征向量，将一定可以表示成下面的形式：

$$A\nu = \lambda\nu$$

$\lambda$  为特征向量  $\vec{v}$  对应的特征值。特征值分解是将一个矩阵分解为如下形式：

$$A = Q \sum Q^{-1}$$

其中， $Q$  是这个矩阵  $A$  的特征向量组成的矩阵， $\sum$  是一个对角矩阵，每一个对角线元素就是一个特征值，里面的特征值是由大到小排列的，这些特征值所对应的特征向量就是描述这个矩阵变化方向（从主要的变化到次要的变化排列）。也就是说矩阵  $A$  的信息可以由其特征值和特征向量表示。

**3.2. 奇异值与特征值有什么关系。** 那么奇异值和特征值是怎么对应起来的呢？我们将一个矩阵  $A$  的转置乘以  $A$ ，并对  $A^T A$  求特征值，则有下面的形式：

$$(A^T A)V = \lambda V$$

这里  $V$  就是上面的右奇异向量，另外还有：

$$\sigma_i = \sqrt{\lambda_i}, u_i = \frac{1}{\sigma_i} AV$$

这里的  $\sigma$  就是奇异值， $u$  就是上面说的左奇异向量。【证明那个哥们也没给】 奇异值  $\sigma$  跟特征值类似，在矩阵  $\Sigma$  中也是从大到小排列，而且  $\sigma$  的减少特别的快，在很多情况下，前 10% 甚至 1% 的奇异值的和就占了全部的奇异值之和的 99% 以上了。也就是说，我们也可以用前  $r$  ( $r$  远小于  $mn$ ) 个的奇异值来近似描述矩阵，即部分奇异值分解：

$$A_{m \times n} \approx U_{m \times r} \sum_{r \times r} V_{r \times n}^T$$

右边的三个矩阵相乘的结果将会是一个接近于  $A$  的矩阵，在这儿， $r$  越接近于  $n$ ，则相乘的结果越接近于  $A$ 。

#### 4. 概率分布与随机变量

**4.1. 机器学习为什么要使用概率。** 事件的概率是衡量该事件发生的可能性的量度。虽然在一次随机试验中某个事件的发生是带有偶然性的，但那些可在相同条件下大量重复的随机试验却往往呈现出明显的数量规律。

机器学习除了处理不确定量，也需处理随机量。不确定性和随机性可能来自多个方面，使用概率论来量化不确定性。

概率论在机器学习中扮演着一个核心角色，因为机器学习算法的设计通常依赖于对数据的概率假设。

例如在机器学习 (Andrew Ng) 的课中，会有一个朴素贝叶斯假设就是条件独立的一个例子。该学习算法对内容做出假设，用来分辨电子邮件是否为垃圾邮件。假设无论邮件是否为垃圾邮件，单词  $x$  出现在邮件中的概率条件独立于单词  $y$ 。很明显这个假设不是不失一般性的，因为某些单词几乎总是同时出现。然而，最终结果是，这个简单的假设对结果的影响并不大，且无论如何都可以让我们快速判别垃圾邮件。

#### 4.2. 变量与随机变量有什么区别。随机变量 (random variable)

表示随机现象（在一定条件下，并不总是出现相同结果的现象称为随机现象）中各种结果的实值函数（一切可能的样本点）。例如某一时间内公共汽车站等车乘客人数，电话交换台在一定时间内收到的呼叫次数等，都是随机变量的实例。

随机变量与模糊变量的不确定性的本质差别在于，后者的测定结果仍具有不确定性，即模糊性。

### 变量与随机变量的区别:

当变量的取值的概率不是 1 时, 变量就变成了随机变量; 当随机变量取值的概率为 1 时, 随机变量就变成了变量.

比如:

当变量  $x$  值为 100 的概率为 1 的话, 那么  $x = 100$  就是确定了的, 不会再有变化, 除非有进一步运算. 当变量  $x$  的值为 100 的概率不为 1, 比如为 50 的概率是 0.5, 为 100 的概率是 0.5, 那么这个变量就是会随不同条件而变化的, 是随机变量, 取到 50 或者 100 的概率都是 0.5, 即 50%.

**4.3. 随机变量与概率分布的联系.** 一个随机变量仅仅表示一个可能取得的状态, 还必须给定与之相伴的概率分布来制定每个状态的可能性. 用来描述随机变量或一族随机变量的每一个可能的状态的可能性大小的方法, 就是 **概率分布 (probability distribution)**.

随机变量可以分为离散型随机变量和连续型随机变量.

相应的描述其概率分布的函数是

概率质量函数 (Probability Mass Function, PMF): 描述离散型随机变量的概率分布, 通常用大写字母  $P$  表示.

概率密度函数 (Probability Density Function, PDF): 描述连续型随机变量的概率分布, 通常用小写字母  $p$  表示.

**4.4. 离散型随机变量和概率质量函数.** PMF 将随机变量能够取得的每个状态映射到随机变量取得该状态的概率.

- 一般而言,  $P(x)$  表示时  $X = x$  的概率.
- 有时候为了防止混淆, 要明确写出随机变量的名称  $P(x=x)$
- 有时候需要先定义一个随机变量, 然后制定它遵循的概率分布  $x$  服从  $P(x)$

PMF 可以同时作用于多个随机变量, 即联合概率分布 (joint probability distribution)  $P(X = x, Y = y)^*$  表示  $X = x$  和  $Y = y$  同时发生的概率, 也可以简写成  $P(x, y)$ .

如果一个函数  $P$  是随机变量  $X$  的 PMF, 那么它必须满足如下三个条件

- $P$  的定义域必须是的所有可能状态的集合
- $\sum_{x \in X} P(x) \leq 1$ .
- $\sum_{x \in X} P(x) = 1$ . 我们把这一条性质称之为归一化的 (normalized)

**4.5. 连续型随机变量和概率密度函数.** 如果一个函数  $p$  是  $x$  的 PDF, 那么它必须满足如下几个条件

- $p$  的定义域必须是  $x$  的所有可能状态的集合.

- $x X, p(x) 0$ . 注意, 我们并不要求  $p(x) 1$ , 因为此处  $p(x)$  不是表示的对应此状态具体的概率, 而是概率的一个相对大小 (密度). 具体的概率, 需要积分去求.
- $p(x)dx = 1$ , 积分下来, 总和还是 1, 概率之和还是 1.

注: PDF  $p(x)$  并没有直接对特定的状态给出概率, 给出的是密度, 相对的, 它给出了落在面积为  $x$  的无限小的区域内的概率为  $p(x)x$ . 由此, 我们无法求得具体某个状态的概率, 我们可以求得的是某个状态  $x$  落在某个区间  $[a, b]$  内的概率为  $\int_a^b p(x)dx$ .

#### 4.6. 举例理解条件概率. 条件概率公式如下:

$$P(A|B) = P(A \cap B)/P(B)$$

说明: 在同一个样本空间  $\Omega$  中的事件或者子集  $A$  与  $B$ , 如果随机从  $\Omega$  中选出的一个元素属于  $B$ , 那么下一个随机选择的元素属于  $A$  的概率就定义为在  $B$  的前提下  $A$  的条件概率. 条件概率文氏图示意如图 1.1 所示.

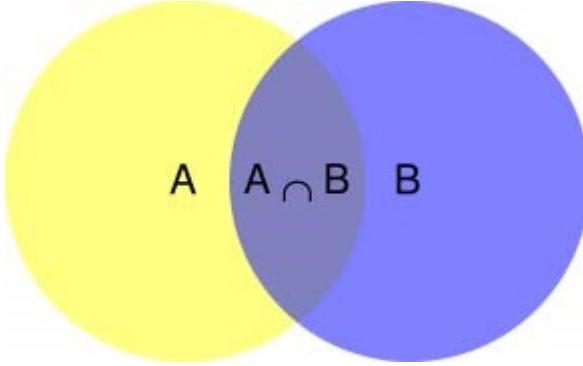


图 1.1 条件概率文氏图示意

根据文氏图, 可以很清楚地看到在事件  $B$  发生的情况下, 事件  $A$  发生的概率就是  $P(A \cap B)$  除以  $P(B)$ .

举例: 一对夫妻有两个小孩, 已知其中一个是女孩, 则另一个是女孩子的概率是多少? (面试、笔试都碰到过)

**穷举法:** 已知其中一个是女孩, 那么样本空间为男女, 女女, 女男, 则另外一个仍然是女生的概率就是  $1/3$ .

**条件概率法:**  $P(|) = P() / P()$ , 夫妻有两个小孩, 那么它的样本空间为女女, 男女, 女男, 男男, 则  $P()$  为  $1/4$ ,  $P = 1 - P() = 3/4$ , 所以最后  $1/3$ .

这里大家可能会误解, 男女和女男是同一种情况, 但实际上类似姐弟和兄妹是不同情况.

#### 4.7. 联合概率与边缘概率联系区别. 区别:

联合概率: 联合概率指类似于  $P(X = a, Y = b)$  这样, 包含多个条件, 且所有条件同时成立的概率. 联合概率是指在多元的概率分布中多个随机变量分别满足各自条件的概率.

边缘概率: 边缘概率是某个事件发生的概率, 而与其它事件无关. 边缘概率指类似于  $P(X = a)$ ,  $P(Y = b)$  这样, 仅与单个随机变量有关的概率.

### 联系:

联合分布可求边缘分布, 但若只知道边缘分布, 无法求得联合分布.

**4.8. 条件概率的链式法则.** 由条件概率的定义, 可直接得出下面的乘法公式:

乘法公式设  $A, B$  是两个事件, 并且  $P(A) > 0$ , 则有

$$P(AB) = P(B|A)P(A)$$

推广

$$P(ABC) = P(C|AB)P(B|A)P(A)$$

一般地, 用归纳法可证: 若  $P(A_1 A_2 \dots A_n) > 0$ , 则有

$$P(A_1 A_2 \dots A_n) = P(A_n | A_1 A_2 \dots A_{n-1})P(A_{n-1} | A_1 A_2 \dots A_{n-2}) \dots P(A_2 | A_1)P(A_1) = P(A_1) \prod_{i=2}^n P(A_i | A_1 A_2 \dots A_{i-1})$$

任何多维随机变量联合概率分布, 都可以分解成只有一个变量的条件概率相乘形式.

**4.9. 独立性和条件独立性. 独立性** 两个随机变量  $x$  和  $y$ , 概率分布表示成两个因子乘积形式, 一个因子只包含  $x$ , 另一个因子只包含  $y$ , 两个随机变量相互独立 (independent).

条件有时为不独立的事件之间带来独立, 有时也会把本来独立的事件, 因为此条件的存在, 而失去独立性.

举例:  $P(XY) = P(X)P(Y)$ , 事件  $X$  和事件  $Y$  独立. 此时给定  $Z$ ,

$$P(X, Y|Z) \neq P(X|Z)P(Y|Z)$$

事件独立时, 联合概率等于概率的乘积. 这是一个非常好的数学性质, 然而不幸的是, 无条件的独立是十分稀少的, 因为大部分情况下, 事件之间都是互相影响的.

### 条件独立性

给定  $Z$  的情况下,  $X$  和  $Y$  条件独立, 当且仅当

$$X \perp Y | Z \iff P(X, Y|Z) = P(X|Z)P(Y|Z)$$

$X$  和  $Y$  的关系依赖于  $Z$ , 而不是直接产生.

举例定义如下事件:

$X$ : 明天下雨;

$Y$ : 今天的地面是湿的;

$Z$ : 今天是否下雨;

$Z$  事件的成立, 对  $X$  和  $Y$  均有影响, 然而, 在  $Z$  事件成立的前提下, 今天的地面情况对明天是否下雨没有影响.

## 5. 常见概率分布

**5.1. Bernoulli 分布.** Bernoulli 分布 (伯努利分布, 0-1 分布) 是单个二值随机变量分布, 单参数  $\phi [0,1]$  控制,  $\phi$  给出随机变量等于 1 的概率. 主要性质有:

$$\begin{aligned} P(x=1) &= \phi \\ P(x=0) &= 1 - \phi \\ P(x=x) &= \phi^x(1-\phi)^{1-x} \end{aligned}$$

其期望和方差为:

$$\begin{aligned} E_x[x] &= \phi \\ Var_x(x) &= \phi(1-\phi) \end{aligned}$$

**适用范围:** 伯努利分布适合对离散型随机变量建模.

Multinoulli 分布也叫范畴分布, 是单个  $k$  值随机分布, 经常用来表示对象分类的分布. 其中  $k$  是有限值. Multinoulli 分布由向量  $\vec{p} \in [0, 1]^{k-1}$  参数化, 每个分量  $p_i$  表示第  $i$  个状态的概率, 且  $p_k = 1 - 1^T p$ . 这里  $1^T$  表示元素全为 1 的列向量的转置, 其实就是对于向量  $p$  中除了  $k$  的概率之和. 可以重写为  $p_k = 1 - \sum_0^{k-1} p_i$ .

补充二项分布、多项分布:

二项分布, 通俗点硬币抛多次. 二项分布 (Binomial distribution) 是  $n$  重伯努利试验成功次数的离散概率分布.

多项式分布 (Multinomial Distribution) 是二项式分布的推广. 二项式做  $n$  次伯努利实验, 规定了每次试验的结果只有两个, 如果现在还是做  $n$  次试验, 只不过每次试验的结果可以有多  $m$  个, 且  $m$  个结果发生的概率互斥且和为 1, 则发生其中一个结果  $X$  次的概率就是多项式分布.

**5.2. 高斯分布.** 高斯也叫正态分布 (Normal Distribution), 概率度函数如下:

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

其中,  $\mu$  和  $\sigma$  分别是均值和方差, 中心峰值  $x$  坐标由  $\mu$  给出, 峰的宽度受  $\sigma$  控制, 最大点在  $x = \mu$  处取得, 拐点为  $x = \mu \pm \sigma$

正态分布中,  $\pm 1\sigma$ 、 $\pm 2\sigma$ 、 $\pm 3\sigma$  下的概率分别是 68.3%、95.5%、99.73%, 这 3 个数最好记住.

此外, 令  $\mu = 0, \sigma = 1$  高斯分布即简化为标准正态分布:

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

对概率密度函数高效求值:

$$N(x; \mu, \beta^{-1}) = \sqrt{\frac{\beta}{2\pi}} \exp\left(-\frac{1}{2}\beta(x - \mu)^2\right)$$

其中,  $\beta = \frac{1}{\sigma^2}$  通过参数  $\beta$  来控制分布精度.

**5.3. 何时采用正态分布.** 问: 何时采用正态分布? 答: 缺乏实数上分布的先验知识, 不知选择何种形式时, 默认选择正态分布总是不会错的, 理由如下:

1. 中心极限定理告诉我们, 很多独立随机变量均近似服从正态分布, 现实中很多复杂系统都可以被建模成正态分布的噪声, 即使该系统可以被结构化分解.
2. 正态分布是具有相同方差的所有概率分布中, 不确定性最大的分布, 换句话说, 正态分布是对模型加入先验知识最少的分布.

正态分布的推广: 正态分布可以推广到  $R^n$  空间, 此时称为**多位正态分布**, 其参数是一个正定对称矩阵  $\Sigma$ :

$$N(\vec{x}; \vec{\mu}, \Sigma) = \sqrt{\frac{1}{(2\pi)^n \det(\Sigma)}} \exp\left(-\frac{1}{2}(\vec{x} - \vec{\mu})^T \Sigma^{-1} (\vec{x} - \vec{\mu})\right)$$

对多位正态分布概率密度高效求值:

$$N(\vec{x}; \vec{\mu}, \vec{\beta}^{-1}) = \sqrt{\det(\vec{\beta})} (2\pi)^n \exp\left(-\frac{1}{2}(\vec{x} - \vec{\mu})^T \vec{\beta} (\vec{x} - \vec{\mu})\right)$$

此处,  $\vec{\beta}$  是一个精度矩阵.

**5.4. 指数分布.** 深度学习中, 指数分布用来描述在  $x = 0$  点处取得边界点的分布, 指数分布定义如下:

$$p(x; \lambda) = \lambda I_{x \geq 0} \exp(-\lambda x)$$

指数分布用指示函数  $I_{x \geq 0}$  来使  $x$  取负值时的概率为零.

**5.5. Laplace 分布 (拉普拉斯分布).** 一个联系紧密的概率分布是 Laplace 分布 (Laplace distribution), 它允许我们在任意一点  $\mu$  处设置概率质量的峰值

$$\text{Laplace}(x; \mu; \gamma) = \frac{1}{2\gamma} \exp\left(-\frac{|x - \mu|}{\gamma}\right)$$

**5.6. Dirac 分布和经验分布.** Dirac 分布可保证概率分布中所有质量都集中在一个点上. Diract 分布的狄拉克  $\delta$  函数 (也称为单位脉冲函数) 定义如下:

$$p(x) = \delta(x - \mu), x \neq \mu$$

$$\int_a^b \delta(x - \mu) dx = 1, a < \mu < b$$

Dirac 分布经常作为经验分布 (empirical distribution) 的一个组成部分出现

$$\hat{p}(\vec{x}) = \frac{1}{m} \sum_{i=1}^m \delta(\vec{x} - \vec{x}^{(i)})$$

, 其中,  $m$  个点  $x^1, \dots, x^m$  是给定的数据集, 经验分布将概率密度  $\frac{1}{m}$  赋给了这些点.

当我们在训练集上训练模型时, 可以认为从这个训练集上得到的经验分布指明了采样来源.

**适用范围:** 狄拉克 函数适合对连续型随机变量的经验分布.

## 6. 期望、方差、协方差、相关系数

**6.1. 期望.** 在概率论和统计学中, 数学期望 (或均值, 亦简称期望) 是试验中每次可能结果的概率乘以其结果的总和. 它反映随机变量平均取值的大小.

- 线性运算:  $E(ax + by + c) = aE(x) + bE(y) + c$
- 推广形式:  $E(\sum_{k=1}^n a_i x_i + c) = \sum_{k=1}^n a_i E(x_i) + c$
- 函数期望: 设  $f(x)$  为  $x$  的函数, 则  $f(x)$  的期望为
  - 离散函数:  $E(f(x)) = \sum_{k=1}^n f(x_k)P(x_k)$
  - 连续函数:  $E(f(x)) = \int_{-\infty}^{+\infty} f(x)p(x)dx$

注意:

- 函数的期望大于等于期望的函数(Jensen(詹森)不等式, 即  $E(f(x)) \geq f(E(x))$ )
- 一般情况下, 乘积的期望不等于期望的乘积.
- 如果  $X$  和  $Y$  相互独立, 则  $E(xy) = E(x)E(y)$ .

**6.2. 方差.** 概率论中方差用来度量随机变量和其数学期望（即均值）之间的偏离程度. 方差是一种特殊的期望. 定义为:

$$Var(x) = E((x - E(x))^2)$$

方差性质:

- 1)  $Var(x) = E(x^2) - E(x)^2$
- 2) 常数的方差为 0;
- 3) 方差不满足线性性质;
- 4) 如果  $X$  和  $Y$  相互独立,  $Var(ax + by) = a^2Var(x) + b^2Var(y)$

**6.3. 协方差.** 协方差是衡量两个变量线性相关性强度及变量尺度. 两个随机变量的协方差定义为:

$$Cov(x, y) = E((x - E(x))(y - E(y)))$$

方差是一种特殊的协方差. 当  $X = Y$  时,  $Cov(x, y) = Var(x) = Var(y)$ .

协方差性质:

- 1) 独立变量的协方差为 0.
- 2) 协方差计算公式:

$$Cov\left(\sum_{i=1}^m a_i x_i, \sum_{j=1}^n b_j y_j\right) = \sum_{i=1}^m \sum_{j=1}^n a_i b_j Cov(x_i y_j)$$

3) 特殊情况:

$$Cov(a + bx, c + dy) = bdCov(x, y)$$

**6.4. 相关系数.** 相关系数是研究变量之间线性相关程度的量. 两个随机变量的相关系数定义为:

$$Corr(x, y) = \frac{Cov(x, y)}{\sqrt{Var(x)Var(y)}}$$

相关系数的性质:

- 1) 有界性. 相关系数的取值范围是  $[-1, 1]$ , 可以看成无量纲的协方差.
- 2) 值越接近 1, 说明两个变量正相关性 (线性) 越强. 越接近-1, 说明负相关性越强, 当为 0 时, 表示两个变量没有相关性.



## CHAPTER 2

# 机器学习基础

机器学习起源于上世纪 50 年代，1959 年在 IBM 工作的 Arthur Samuel 设计了一个下棋程序，这个程序具有学习的能力，它可以在不断的对弈中提高自己。由此提出了“机器学习”这个概念，它是一个结合了多个学科如概率论，优化理论，统计等，最终在计算机上实现自我获取新知识，学习改善自己的这样一个研究领域。机器学习是人工智能的一个子集，目前已经发展出许多有用的方法，比如支持向量机，回归，决策树，随机森林，强化方法，集成学习，深度学习等等，一定程度上可以帮助人们完成一些数据预测，自动化，自动决策，最优化等初步替代脑力的任务。本章我们主要介绍下机器学习的基本概念、监督学习、分类算法、逻辑回归、代价函数、损失函数、LDA、PCA、决策树、支持向量机、EM 算法、聚类和降维以及模型评估有哪些方法、指标等等。

## 1. 基本概念

**1.1. 大话理解机器学习本质.** 机器学习 (Machine Learning, ML)，顾名思义，让机器去学习。这里，机器指的是计算机，是算法运行的物理载体，你也可以把各种算法本身当做一个有输入和输出的机器。那么到底让计算机去学习什么呢？对于一个任务及其表现的度量方法，设计一种算法，让算法能够提取中数据所蕴含的规律，这就叫机器学习。如果输入机器的数据是带有标签的，就称作有监督学习。如果数据是无标签的，就是无监督学习。

**1.2. 什么是神经网络.** 神经网络就是按照一定规则将多个神经元连接起来的网络。不同的神经网络，具有不同的连接规则。例如全连接 (Full Connected, FC) 神经网络，它的规则包括：

- (1) 有三种层：输入层，输出层，隐藏层。
- (2) 同一层的神经元之间没有连接。
- (3) fully connected 的含义：第 N 层的每个神经元和第 N-1 层的所有神经元相连，第 N-1 层神经元的输出就是第 N 层神经元的输入。
- (4) 每个连接都有一个权值。

**神经网络架构** 图 2-1 就是一个神经网络系统，它由很多层组成。输入层负责接收信息，比如一只猫的图片。输出层是计算机对这个输入信息的判断结果，它是不是猫。隐藏层就是对输入信息的传递和加工处理。

图 2-1 神经网络系统

**1.3. 各种常见算法图示.** 日常使用机器学习的任务中，我们经常会遇见各种算法，图 2-2 是各种常见算法的图示。

图 2-2 各种常见算法图示

**1.4. 计算图的导数计算.** 计算图导数计算是反向传播，利用链式法则和隐式函数求导。

假设  $z = f(u, v)$  在点  $(u, v)$  处偏导连续， $(u, v)$  是关于  $t$  的函数，在  $t$  点可导，求  $z$  在  $t$  点的导数。

根据链式法则有

$$\frac{dz}{dt} = \frac{\partial z}{\partial u} \cdot \frac{du}{dt} + \frac{\partial z}{\partial v} \cdot \frac{dv}{dt}$$

链式法则用文字描述：“由两个函数凑起来的复合函数，其导数等于是里边函数代入外边函数的值之导数，乘以里边函数的导数。”

为了便于理解，下面举例说明：

$$f(x) = x^2, g(x) = 2x + 1$$

则：

$$f[g(x)]' = 2[g(x)] \times g'(x) = 2[2x + 1] \times 2 = 8x + 4$$

**1.5. 理解局部最优与全局最优.** 笑谈局部最优和全局最优

柏拉图有一天问老师苏格拉底什么是爱情？苏格拉底叫他到麦田走一次，摘一颗最大的麦穗回来，不许回头，只可摘一次。柏拉图空着手出来了，他的理由是，看见不错的，却不知道是不是最好的，一次次侥幸，走到尽头时，才发现还不如前面的，于是放弃。苏格拉底告诉他：“这就是爱情。”这故事让我们明白了一个道理，因为生命的一些不确定性，所以全局最优解是很难找到的，或者说根本就不存在，我们应该设置一些限定条件，然后在这个范围内寻找最优解，也就是局部最优解——有所斩获总比空手而归强，哪怕这种斩获只是一次有趣的经历。柏拉图有一天又问什么是婚姻？苏格拉底叫他到树林走一次，选一棵最好的树做圣诞树，也是不许回头，只许选一次。这次他一身疲惫地拖了一棵看起来直挺、翠绿，却有点稀疏的杉树回来，他的理由是，有了上回的教训，好不容易看见一棵看似不错的，又发现时间、体力已经快不够用了，也不管是不是最好的，就拿回来了。苏格拉底告诉他：“这就是婚姻。”

优化问题一般分为局部最优和全局最优。其中，

- (1) 局部最优，就是在函数值空间的一个有限区域内寻找最小值；而全局最优，是在函数值空间整个区域寻找最小值问题。
- (2) 函数局部最小点是它的函数值小于或等于附近点的点，但是有可能大于较远距离的点。
- (3) 全局最小点是那种它的函数值小于或等于所有的可行点。

**1.6. 大数据与深度学习之间的关系.** 首先来看大数据、机器学习及数据挖掘三者简单的定义：

**大数据**通常被定义为“超出常用软件工具捕获，管理和处理能力”的数据集。**机器学习**关心的问题是如何构建计算机程序使用经验自动改进。**数据挖掘**是从数据中提取模式的特定算法的应用，在数据挖掘中，重点在于算法的应用，而不是算法本身。

**机器学习和数据挖掘**之间的关系如下：数据挖掘是一个过程，在此过程中机器学习算法被用作提取数据集中的潜在有价值模式的工具。大数据与深度学习关系总结如下：

- (1) 深度学习是一种模拟大脑的行为。可以从所学习对象的机制以及行为等等很多相关联的方面进行学习，模仿类型行为以及思维。
- (2) 深度学习对于大数据的发展有帮助。深度学习对于大数据技术开发的每一个阶段均有帮助，不管是数据的分析还是挖掘还是建模，只有深度学习，这些工作才会有希望一一得到实现。
- (3) 深度学习转变了解决问题的思维。很多时候发现问题到解决问题，走一步看一步不是一个主要的解决问题的方式了，在深度学习的基础上，要求我们从开始到最后都要基于一个目标，为了需要优化的那个最终目标去进行处理数据以及将数据放入到数据应用平台上去，这就是端到端 (End to End)。
- (4) 大数据的深度学习需要一个框架。在大数据方面的深度学习都是从基础的角度出发的，深度学习需要一个框架或者一个系统。总而言之，将你的大数据通过深度分析变为现实，这就是深度学习和大数据的最直接关系。

## 2. 机器学习学习方式

根据数据类型的不同，对一个问题的建模有不同的方式。依据不同的学习方式和输入数据，机器学习主要分为以下四种学习方式。

**2.1. 监督学习.** 特点：监督学习是使用已知正确答案的示例来训练网络。已知数据和其一一对应的标签，训练一个预测模型，将输入数据映射到标签的过程。

常见应用场景：监督式学习的常见应用场景如分类问题和回归问题。

算法举例：常见的有监督机器学习算法包括支持向量机 (Support Vector Machine, SVM)，朴素贝叶斯 (Naive Bayes)，逻辑回归 (Logistic Regression)，K 近邻 (K-Nearest Neighborhood, KNN)，决策树 (Decision Tree)，随机森林 (Random Forest)，AdaBoost 以及线性判别分析

(Linear Discriminant Analysis, LDA) 等。深度学习 (Deep Learning) 也是大多数以监督学习的方式呈现。

**2.2. 非监督式学习.** 定义：在非监督式学习中，数据并不被特别标识，适用于你具有数据集但无标签的情况。学习模型是为了推断出数据的一些内在结构。

常见应用场景：常见的应用场景包括关联规则的学习以及聚类等。

算法举例：常见算法包括 Apriori 算法以及 k-Means 算法。

**2.3. 半监督式学习.** 特点：在此学习方式下，输入数据部分被标记，部分没有被标记，这种学习模型可以用来进行预测。

常见应用场景：应用场景包括分类和回归，算法包括一些对常用监督式学习算法的延伸，通过对已标记数据建模，在此基础上，对未标记数据进行预测。

算法举例：常见算法如图论推理算法 (Graph Inference) 或者拉普拉斯支持向量机 (Laplacian SVM) 等。

**2.4. 弱监督学习.** 特点：弱监督学习可以看做是有多个标记的数据集合，次集合可以是空集，单个元素，或包含多种情况（没有标记，有一个标记，和有多个标记）的多个元素。数据集的标签是不可靠的，这里的不可靠可以是标记不正确，多种标记，标记不充分，局部标记等。已知数据和其一一对应的弱标签，训练一个智能算法，将输入数据映射到一组更强的标签的过程。标签的强弱指的是标签蕴含的信息量的多少，比如相对于分割的标签来说，分类的标签就是弱标签。

算法举例：举例，给出一张包含气球的图片，需要得出气球在图片中的位置及气球和背景的分割线，这就是已知弱标签学习强标签的问题。

在企业数据应用的场景下，人们最常用的可能就是监督式学习和非监督式学习的模型。在图像识别等领域，由于存在大量的非标识的数据和少量的可标识数据，目前半监督式学习是一个很热的话题。

**2.5. 监督学习有哪些步骤.** 监督学习是使用已知正确答案的示例来训练网络，每组训练数据有一个明确的标识或结果。想象一下，我们可以训练一个网络，让其从照片库中（其中包含气球的照片）识别出气球的照片。以下就是我们在这个假设场景中所要采取的步骤。

**步骤 1：数据集的创建和分类** 首先，浏览你的照片（数据集），确定所有包含气球的照片，并对其进行标注。然后，将所有照片分为训练集和验证集。目标就是在深度网络中找一函数，这个函数输入是任意一张照片，当照片中包含气球时，输出 1，否则输出 0。

**步骤 2：数据增强 (Data Augmentation)** 当原始数据搜集和标注完毕，一般搜集的数据并不一定包含目标在各种扰动下的信息。数据的好坏对于机器学习模型的预测能力至关重要

要，因此一般会进行数据增强。对于图像数据来说，数据增强一般包括，图像旋转，平移，颜色变换，裁剪，仿射变换等。

**步骤 3：特征工程（Feature Engineering）** 一般来讲，特征工程包含特征提取和特征选择。常见的手工特征（Hand-Crafted Feature）有尺度不变特征变换（Scale-Invariant Feature Transform, SIFT），方向梯度直方图（Histogram of Oriented Gradient, HOG）等。由于手工特征是启发式的，其算法设计背后的出发点不同，将这些特征组合在一起的时候有可能会产生冲突，如何将组合特征的效能发挥出来，使原始数据在特征空间中的判别性最大化，就需要用到特征选择的方法。在深度学习方法大获成功之后，人们很大一部分不再关注特征工程本身。因为，最常用到的卷积神经网络（Convolutional Neural Networks, CNNs）本身就是一种特征提取和选择的引擎。研究者提出的不同的网络结构、正则化、归一化方法实际上就是深度学习背景下的特征工程。

**步骤 4：构建预测模型和损失** 将原始数据映射到特征空间之后，也就意味着我们得到了比较合理的输入。下一步就是构建合适的预测模型得到对应输入的输出。而如何保证模型的输出和输入标签的一致性，就需要构建模型预测和标签之间的损失函数，常见的损失函数（Loss Function）有交叉熵、均方差等。通过优化方法不断迭代，使模型从最初的初始化状态一步步变化为有预测能力的模型的过程，实际上就是学习的过程。

**步骤 5：训练** 选择合适的模型和超参数进行初始化，其中超参数比如支持向量机中核函数、误差项惩罚权重等。当模型初始化参数设定好后，将制作好的特征数据输入到模型，通过合适的优化方法不断缩小输出与标签之间的差距，当迭代过程到了截止条件，就可以得到训练好的模型。优化方法最常见的就是梯度下降法及其变种，使用梯度下降法的前提是优化目标函数对于模型是可导的。

**步骤 6：验证和模型选择** 训练完训练集图片后，需要进行模型测试。利用验证集来验证模型是否可以准确地挑选出含有气球在内的照片。在此过程中，通常会通过调整和模型相关的各种事物（超参数）来重复步骤 2 和 3，诸如里面有多少个节点，有多少层，使用怎样的激活函数和损失函数，如何在反向传播阶段积极有效地训练权值等等。

**步骤 7：测试及应用** 当有了一个准确的模型，就可以将该模型部署到你的应用程序中。你可以将预测功能发布为 API（Application Programming Interface, 应用程序编程接口）调用，并且你可以从软件中调用该 API，从而进行推理并给出相应的结果。

### 3. 分类算法

分类算法和回归算法是对真实世界不同建模的方法。分类模型是认为模型的输出是离散的，例如大自然的生物被划分为不同的种类，是离散的。回归模型的输出是连续的，例如人的身高变化过程是一个连续过程，而不是离散的。

因此，在实际建模过程时，采用分类模型还是回归模型，取决于你对任务（真实世界）的分析和理解。

**3.1. 常用分类算法的优缺点？** 接下来我们介绍常用分类算法的优缺点，如表 2-1 所示。

表 2-1 常用分类算法的优缺点

算 法	优 点	缺 点
--------	--------	--------

Bayes1) 1)  
贝 所 需 要 假 设 属性 之 间 相 互 独 立,  
叶 需 估 计 的 参 数 少, 对 于 缺 失 数据 不 敏 感。  
斯 分 类 法

2) 有 着 坚 实 的 数 学 基 础, 以 及 稳 定 的 分 类 效 率。

欢 吃 番 茄、 鸡 蛋, 却 不 喜 欢 吃 番 茄 炒 蛋)。

2) 需 要 知 道 先 验 概

Decision Tree 1)

Tree 不 对 于 各 类 别 样 本 数 量 不 一 致 数据, 信息 增 益 偏 向 于 那 些 具 有 更 多 数 值 的 特 征。

2) 适 合 高 维 数 据。

3) 简 单 易 于 理 解。

4) 短 时 间 内 处 理 大 量 数 据, 2) 易 于 过 拟 合。

3) 忽 略 属 性, 得 到 可 能 的 结 果。

SVM 1) 1)

支 可 对  
持 以 缺  
向 解 失  
量 决 数  
机 小 据  
样 本 敏  
本 感。

2)

机 内  
器 存  
学 消  
习 耗  
的 大,  
问 难  
题。 以

2) 解

提 释。  
高 3)  
泛 运  
化 行 和  
性 调  
能。 参

3) 略  
可 烦  
以 人。

解决  
高维、  
非线性  
问题。

超  
高  
维  
文  
本  
分  
类  
仍

KNN 1) 思想简单, 理论成熟, 既可以用来做分类也可以用来做回归; 2) 可用于非线性分类; 3) 训练时间复杂度为 $O(n^2)$ ; 4) 对于样本分类不均衡的问题, 会产生误判。K 近邻 1) 计算量太大。2) 对于样本分类不均衡的问题, 会产生误判。3) 需要大量的内存。4) 输出的可解释性不强。

Logistlc 特  
Re- 速 征  
gres- 度 处  
sion 快。 理  
逻 2) 复 杂。  
辑 简 需 要  
回 单 易 归 于  
归 一 化 和 较  
于 理 多 的 特  
理, 直 接 看 到  
接 看 到 各 个 特  
解, 程, 征 工  
直 接 看 到 各 个 特 征 程。  
接 看 到 各 个 特 征 的 权 重。

3)

能 容 易 地 更 新 模 型 吸 收 新 的 数 据。

4)

如 果 想 要

Neural) 1)  
Net- 分 需  
work 类 要  
神 准 大  
经 确 量  
网 率 参  
络 高。 数  
2) (网  
并 络  
行 拓  
处 扑、  
理 阀  
能 值、  
力 阈  
强。 值)。  
3) 2)  
分 结  
布 果  
式 难  
存 以  
储 解  
和 释。  
学 3)  
习 训  
能 练  
力 时  
强。 间  
4) 过  
鲁 长。  
棒 性  
性 较  
强， 不  
易 受  
噪 声  
声 影  
响。

Adaboosting对

ad- out-  
aboostier

是 比  
一 较  
种 敏  
有 感

很 高 精 度 的 分 类 器。

2)

可 以 使 用 各 种 方 法 构 建 子 分 类 器,

Ad-  
aboost

算 法 提 供 的 是 框 架。

3)

当 使

		预测类别			
实际类别	是	Yes	No	总计	
	Yes	TP	FN	P (实际为 Yes)	
	No	FP	TN	N (实际为 No)	
	总计	P' (被分为 Yes)	N' (被分为 No)	P+N	

图 1. 图 2-3 术语的混淆矩阵

**3.2. 2.8.2 分类算法的评估方法.** 分类评估方法主要功能是用来评估分类算法的好坏，而评估一个分类器算法的好坏又包括许多项指标。了解各种评估方法，在实际应用中选择正确的评估方法是十分重要的。- 几个常用术语 这里首先介绍几个常见的模型评价术语，现在假设我们的分类目标只有两类，计为正例（positive）和负例（negative）分别是：1) True positives(TP): 被正确地划分为正例的个数，即实际为正例且被分类器划分为正例的实例数；2) False positives(FP): 被错误地划分为正例的个数，即实际为负例但被分类器划分为正例的实例数；3) False negatives(FN): 被错误地划分为负例的个数，即实际为正例但被分类器划分为负例的实例数；4) True negatives(TN): 被正确地划分为负例的个数，即实际为负例且被分类器划分为负例的实例数。

表 2-2 四个术语的混淆矩阵

表 2-2 是这四个术语的混淆矩阵，做以下说明：1)  $P=TP+FN$  表示实际为正例的样本个数。2) True、False 描述的是分类器是否判断正确。3) Positive、Negative 是分类器的分类结果，如果正例计为 1、负例计为 -1，即  $positive=1$ 、 $negative=-1$ 。用 1 表示 True，-1 表示 False，那么实际的类标 =  $TF*PN$ ， $TF$  为 true 或 false， $PN$  为 positive 或 negative。4) 例如 True positives(TP) 的实际类标 =  $1*1=1$  为正例，False positives(FP) 的实际类标 =  $(-1)*1=-1$  为负例，False negatives(FN) 的实际类标 =  $(-1)*(-1)=1$  为正例，True negatives(TN) 的实际类标 =  $1*(-1)=-1$  为负例。

### • 评价指标

- 1) 正确率 (accuracy) 正确率是我们最常见的评价指标， $accuracy = (TP+TN)/(P+N)$ ，正确率是被分对的样本数在所有样本数中的占比，通常来说，正确率越高，分类器越好。

图 2.

- 2) 错误率 (error rate) 错误率则与正确率相反, 描述被分类器错分的比例,  $\text{error rate} = (\text{FP}+\text{FN})/(\text{P}+\text{N})$ , 对某一个实例来说, 分对与分错是互斥事件, 所以  $\text{accuracy} = 1 - \text{error rate}$ .
- 3) 灵敏度 (sensitivity)  $\text{sensitivity} = \text{TP}/\text{P}$ , 表示的是所有正例中被分对的比例, 衡量了分类器对正例的识别能力。
- 4) 特异性 (specificity)  $\text{specificity} = \text{TN}/\text{N}$ , 表示的是所有负例中被分对的比例, 衡量了分类器对负例的识别能力。
- 5) 精度 (precision)  $\text{precision} = \text{TP}/(\text{TP}+\text{FP})$ , 精度是精确性的度量, 表示被分为正例的示例中实际为正例的比例。
- 6) 召回率 (recall) 召回率是覆盖面的度量, 度量有多个正例被分为正例,  $\text{recall} = \text{TP}/(\text{TP}+\text{FN}) = \text{TP}/\text{P} = \text{sensitivity}$ , 可以看到召回率与灵敏度是一样的。
- 7) 其他评价指标计算速度: 分类器训练和预测需要的时间; 鲁棒性: 处理缺失值和异常值的能力; 可扩展性: 处理大数据集的能力; 可解释性: 分类器的预测标准的可理解性, 像决策树产生的规则就是很容易理解的, 而神经网络的一堆参数就不好理解, 我们只好把它看成一个黑盒子。
- 8) 精度和召回率反映了分类器分类性能的两个方面。如果综合考虑查准率与查全率, 可以得到新的评价指标 F1-score, 也称为综合分类率:  $F1 = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$ 。  
为了综合多个类别的分类情况, 评测系统整体性能, 经常采用的还有微平均F1 (micro-averaging) 和宏平均

(1) 宏平均F1与微平均F1是以两种不同的平均方式求的全局F1指标。

(2) 宏平均F1的计算方法先对每个类别单独计算F1值, 再取这些F1值的算术平均值作为全局指标。

(3) 微平均F1的计算方法是先累加计算各个类别的a、b、c、d的值, 再由这些值求出F1值。

(4) 由两种平均F1的计算方式不难看出, 宏平均F1平等对待每一个类别, 所以它的值主要受到稀有类别的影响。

#### • ROC 曲线和 PR 曲线

如图2-3, ROC曲线是 (Receiver Operating Characteristic Curve, 受试者工作特征曲线) 的简称, 是图 2-3 ROC 曲线

PR 曲线是 Precision Recall Curve 的简称, 描述的是 precision 和 recall 之间的关系, 以 recall 为横坐标, precision 为纵坐标绘制的曲线。该曲线的所对应的面积 AUC 实际上是目标检测中常用的评价指标平均精度 (Average Precision, AP)。AP 越高, 说明模型性能越好。

**3.3. 2.8.3 正确率能很好的评估分类算法吗.** 不同算法有不同特点，在不同数据集上有不同的表现效果，根据特定的任务选择不同的算法。如何评价分类算法的好坏，要做具体任务具体分析。对于决策树，主要用正确率去评估，但是其他算法，只用正确率能很好的评估吗？答案是否定的。正确率确实是一个很直观很好的评价指标，但是有时候正确率高并不能完全代表一个算法就好。比如对某个地区进行地震预测，地震分类属性分为 0：不发生地震、1 发生地震。我们都知道，不发生的概率是极大的，对于分类器而言，如果分类器不加思考，对每一个测试样例的类别都划分为 0，达到 99% 的正确率，但是，问题来了，如果真的发生地震时，这个分类器毫无察觉，那带来的后果将是巨大的。很显然，99% 正确率的分类器并不是我们想要的。出现这种现象的原因主要是数据分布不均衡，类别为 1 的数据太少，错分了类别 1 但达到了很高的正确率却忽视了研究者本身最为关注的情况。

**3.4. 什么样的分类器是最好的.** 对某一个任务，某个具体的分类器不可能同时满足或提高所有上面介绍的指标。如果一个分类器能正确分对所有的实例，那么各项指标都已经达到最优，但这样的分类器往往不存在。比如之前说的地震预测，既然不能百分百预测地震的发生，但实际情况中能容忍一定程度的误报。假设在 1000 次预测中，共有 5 次预测发生了地震，真实情况中有一次发生了地震，其他 4 次则为误报。正确率由原来的  $999/1000=99.9$  下降为  $996/1000=99.6$ 。召回率由  $0/1=0\%$  上升为  $1/1=100\%$ 。对此解释为，虽然预测失误了 4 次，但真的地震发生前，分类器能预测对，没有错过，这样的分类器实际意义更为重大，正是我们想要的。在这种情况下，在一定正确率前提下，要求分类器的召回率尽量高。

## 4. 2.9 逻辑回归

**4.1. 2.9.1 回归划分.** 广义线性模型家族里，依据因变量不同，可以有如下划分：

- (1) 如果是连续的，就是多重线性回归。
- (2) 如果是二项分布，就是逻辑回归。
- (3) 如果是泊松（Poisson）分布，就是泊松回归。
- (4) 如果是负二项分布，就是负二项回归。
- (5) 逻辑回归的因变量可以是二分类的，也可以是多分类的，但是二分类的更为常用，也更加容易解释。所以实际中最常用的就是二分类的逻辑回归。

**4.2. 逻辑回归适用性.** 逻辑回归可用于以下几个方面：

- (1) 用于概率预测。用于可能性预测时，得到的结果有可比性。比如根据模型进而预测在不同的自变量情况下，发生某病或某种情况的概率有多大。
- (2) 用于分类。实际上跟预测有些类似，也是根据模型，判断某人属于某病或属于某种情况的概率有多大，也就是看一下这个人有多大的可能性是属于某病。进行分类时，仅需要设定一个阈值即可，可能性高于阈值是一类，低于阈值是另一类。

(3) 寻找危险因素。寻找某一疾病的危险因素等。

(4) 仅能用于线性问题。只有当目标和特征是线性关系时，才能用逻辑回归。在应用逻辑回归时注意两点：一是当知道模型是非线性时，不适用逻辑回归；二是当使用逻辑回归时，应注意选择和目标为线性关系的特征。

(5) 各特征之间不需要满足条件独立假设，但各个特征的贡献独立计算。

**4.3. 生成模型和判别模型的区别.** 生成模型：由数据学习联合概率密度分布  $P(X, Y)$ ，然后求出条件概率分布  $P(Y|X)$  作为预测的模型，即生成模型： $P(Y|X) = P(X, Y) / P(X)$  (贝叶斯概率)。基本思想是首先建立样本的联合概率密度模型  $P(X, Y)$ ，然后再得到后验概率  $P(Y|X)$ ，再利用它进行分类。典型的生成模型有朴素贝叶斯，隐马尔科夫模型等

判别模型：由数据直接学习决策函数  $Y=f(X)$  或者条件概率分布  $P(Y|X)$  作为预测的模型，即判别模型。基本思想是有限样本条件下建立判别函数，不考虑样本的产生模型，直接研究预测模型。典型的判别模型包括 k 近邻，感知机，决策树，支持向量机等。这些模型的特点都是输入属性  $X$  可以直接得到后验概率  $P(Y|X)$ ，输出条件概率最大的作为最终的类别（对于二分类任务来说，实际得到一个 score，当 score 大于 threshold 时则为正类，否则为负类）。

举例：

判别式模型举例：要确定一个羊是山羊还是绵羊，用判别模型的方法是从历史数据中学习到模型，然后通过提取这只羊的特征来预测出这只羊是山羊的概率，是绵羊的概率。

生成式模型举例：利用生成模型是根据山羊的特征首先学习出一个山羊的模型，然后根据绵羊的特征学习出一个绵羊的模型，然后从这只羊中提取特征，放到山羊模型中看概率是多少，在放到绵羊模型中看概率是多少，哪个大就是哪个。

联系和区别：

生成方法的特点：上面说到，生成方法学习联合概率密度分布  $P(X, Y)$ ，所以就可以从统计的角度表示数据的分布

判别方法的特点：判别方法直接学习的是决策函数  $Y=f(X)$  或者条件概率分布  $P(Y|X)$ 。不能反映训练数据本身的特点

最后，由生成模型可以得到判别模型，但由判别模型得不到生成模型。

**4.4. 逻辑回归与朴素贝叶斯有什么区别.** 逻辑回归与朴素贝叶斯区别有以下几个方面：

- (1) 逻辑回归是判别模型，朴素贝叶斯是生成模型，所以生成和判别的所有区别它们都有。
- (2) 朴素贝叶斯属于贝叶斯，逻辑回归是最大似然，两种概率哲学间的区别。
- (3) 朴素贝叶斯需要条件独立假设。
- (4) 逻辑回归需要求特征参数间是线性的。

**4.5. 线性回归与逻辑回归的区别.** 线性回归与逻辑回归的区别如下描述：

(1) 线性回归的样本的输出，都是连续值， $y \in (-\infty, +\infty)$ ，而逻辑回归中  $y \in (0, 1)$ ，只能取 0 和 1。

(2) 对于拟合函数也有本质上的差别：

线性回归： $f(x) = \theta^T x = \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$

逻辑回归： $f(x) = P(y=1|x; \theta) = g(\theta^T x)$ ，其中， $g(z) = \frac{1}{1+e^{-z}}$

可以看出，线性回归的拟合函数，是对  $f(x)$  的输出变量  $y$  的拟合，而逻辑回归的拟合函数是对为 1 类样本的概率的拟合。

那么，为什么要以 1 类样本的概率进行拟合呢，为什么可以这样拟合呢？

$\theta^T x = 0$  就相当于是 1 类和 0 类的决策边界：

当  $\theta^T x > 0$ ，则  $y > 0.5$ ；若  $\theta^T x \rightarrow +\infty$ ，则  $y \rightarrow 1$ ，即  $y$  为 1 类；

当  $\theta^T x < 0$ ，则  $y < 0.5$ ；若  $\theta^T x \rightarrow -\infty$ ，则  $y \rightarrow 0$ ，即  $y$  为 0 类；

这个时候就能看出区别，在线性回归中  $\theta^T x$  为预测值的拟合函数；而在逻辑回归中  $\theta^T x$  为决策边界。下表 2-3 为线性回归和逻辑回归的区别。

表 2-3 线性回归和逻辑回归的区别

	线性回归	逻辑回归
目的	预测	分类
$y^{(i)}$	未知	$(0, 1)$
函数	拟合函数	预测函数
参数计算方式	最小二乘法	极大似然估计

下面具体解释一下：

- 拟合函数和预测函数什么关系呢？简单来说就是将拟合函数做了一个逻辑函数的转换，转换后使得  $y^{(i)} \in (0, 1)$ ；
- 最小二乘和最大似然估计可以相互替代吗？回答当然是不行了。我们来看看两者依仗的原理：最大似然估计是计算使得数据出现的可能性最大的参数，依仗的自然是 Probability。而最小二乘是计算误差损失。

## 5. 2.10 代价函数

### 5.1. 2.10.1 为什么需要代价函数.

- 为了得到训练逻辑回归模型的参数，需要一个代价函数，通过训练代价函数来得到参数。
- 用于找到最优解的目的函数。

**5.2. 代价函数作用原理.** 在回归问题中，通过代价函数来求解最优解，常用的是平方误差代价函数。假设函数图像如图 2-4 所示，当参数发生变化时，假设函数状态也会随着变化。

图 2-4  $h(x) = A + Bx$  函数示意图

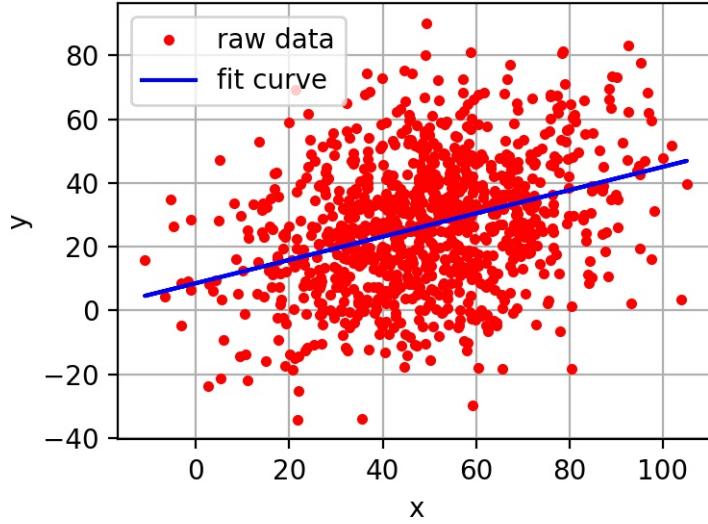


图 3.

想要拟合图中的离散点，我们需要尽可能找到最优的  $A$  和  $B$  来使这条直线更能代表所有数据。如何找到最优解呢，这就需要使用代价函数来求解，以平方误差代价函数为例，假设函数为  $h(x) = \theta_0 x$ 。**平方误差代价函数的主要思想**就是将实际数据给出的值与拟合出的线的对应值做差，求出拟合出的直线与实际的差距。在实际应用中，为了避免因个别极端数据产生的影响，采用类似方差再取二分之一的方式来减小个别数据的影响。因此，引出代价函数：

$$J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)})^2$$

**最优解即为代价函数的最小值**  $\min J(\theta_0, \theta_1)$ 。如果是 1 个参数，代价函数一般通过二维曲线便可直观看出。如果是 2 个参数，代价函数通过三维图像可看出效果，参数越多，越复杂。当参数为 2 个时，代价函数是三维图像，如下图 2-5 所示。

图 2-5 代价函数三维图像

**5.3. 为什么代价函数要非负。** 目标函数存在一个下界，在优化过程当中，如果优化算法能够使目标函数不断减小，根据单调有界准则，这个优化算法就能证明是收敛有效的。只要设计的目标函数有下界，基本上都可以，代价函数非负更为方便。

#### 5.4. 常见代价函数. (1) 二次代价函数 (quadratic cost):

$$J = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2$$

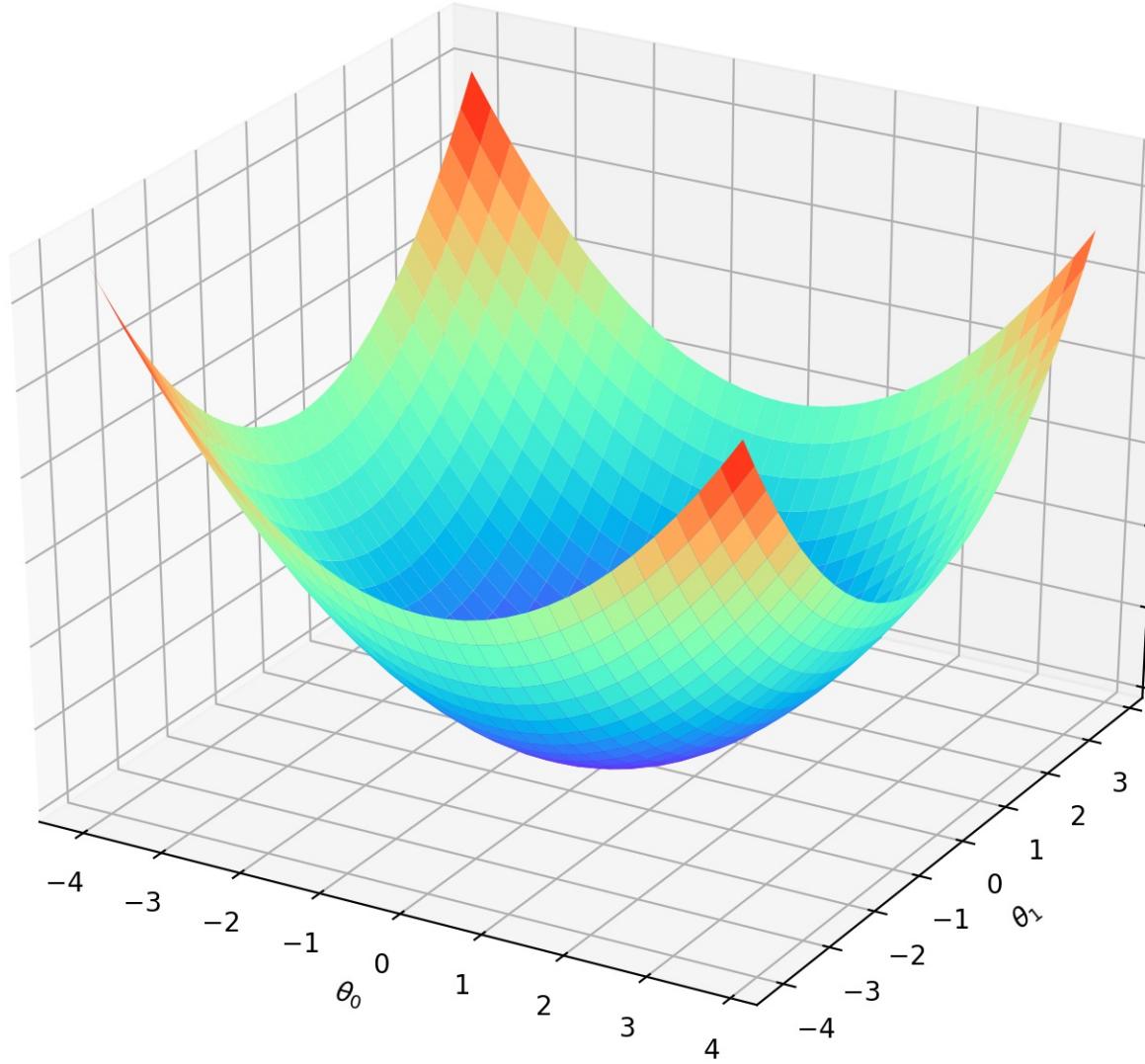


图 4.

其中,  $J$  表示代价函数,  $x$  表示样本,  $y$  表示实际值,  $a$  表示输出值,  $n$  表示样本的总数。使用一个样本为例简单说明, 此时二次代价函数为:

$$J = \frac{(y - a)^2}{2}$$

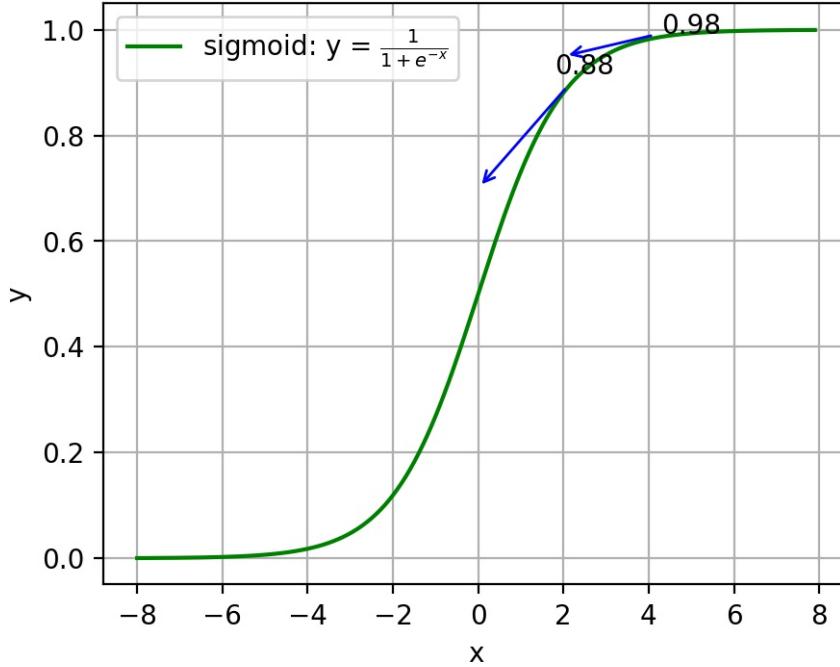


图 5.

假如使用梯度下降法 (Gradient descent) 来调整权值参数的大小，权值  $w$  和偏置  $b$  的梯度推导如下：

$$\frac{\partial J}{\partial w} = (y - a)\sigma'(z)x, \quad \frac{\partial J}{\partial b} = (y - a)\sigma'(z)$$

其中， $z$  表示神经元的输入， $\sigma$  表示激活函数。权值  $w$  和偏置  $b$  的梯度跟激活函数的梯度成正比，激活函数的梯度越大，权值  $w$  和偏置  $b$  的大小调整得越快，训练收敛得就越快。

注：神经网络常用的激活函数为 sigmoid 函数，该函数的曲线如下图 2-6 所示：

图 2-6 sigmoid 函数曲线

如上图所示，对 0.88 和 0.98 两个点进行比较：假设目标是收敛到 1.0。0.88 离目标 1.0 比较远，梯度比较大，权值调整比较大。0.98 离目标 1.0 比较近，梯度比较小，权值调整比较小。调整方案合理。假如目标是收敛到 0。0.88 离目标 0 比较近，梯度比较大，权值调整比较大。0.98 离目标 0 比较远，梯度比较小，权值调整比较小。调整方案不合理。原因：在使用 sigmoid 函数的情况下，初始的代价（误差）越大，导致训练越慢。

## (2) 交叉熵代价函数 (cross-entropy):

$$J = -\frac{1}{n} \sum_x [y \ln a + (1 - y) \ln (1 - a)]$$

其中,  $J$  表示代价函数,  $x$  表示样本,  $y$  表示实际值,  $a$  表示输出值,  $n$  表示样本的总数。权值  $w$  和偏置  $b$  的梯度推导如下:

$$\frac{\partial J}{\partial w_j} = \frac{1}{n} \sum_x x_j (\sigma(z) - y) \quad \frac{\partial J}{\partial b} = \frac{1}{n} \sum_x (\sigma(z) - y)$$

当误差越大时, 梯度就越大, 权值  $w$  和偏置  $b$  调整就越快, 训练的速度也就越快。**二次代价函数适合输出神经元是线性的情况, 交叉熵代价函数适合输出神经元是 S 型函数的情况。**

(3) 对数似然代价函数 (log-likelihood cost): 对数似然函数常用来作为 softmax 回归的代价函数。深度学习中普遍的做法是将 softmax 作为最后一层, 此时常用的代价函数是对数似然代价函数。对数似然代价函数与 softmax 的组合和交叉熵与 sigmoid 函数的组合非常相似。对数似然代价函数在二分类时可以化简为交叉熵代价函数的形式。在 tensorflow 中: 与 sigmoid 搭配使用的交叉熵函数: `tf.nn.sigmoid_cross_entropy_with_logits()`。与 softmax 搭配使用的交叉熵函数: `tf.nn.softmax_cross_entropy_with_logits()`。在 pytorch 中: 与 sigmoid 搭配使用的交叉熵函数: `torch.nn.BCEWithLogitsLoss()`。与 softmax 搭配使用的交叉熵函数: `torch.nn.CrossEntropyLoss()`。

对数似然函数:

我们将似然函数作为机器学习模型的损失函数, 并且用在分类问题中。这时似然函数是直接作用于模型的输出的 (损失函数就是为了衡量当前参数下 model 的预测值 predict 距离真实值 label 的大小, 所以似然函数用作损失函数时当然也是为了完成该任务), 所以对于似然函数来说, 这里的样本集就成了 label 集 (而不是机器学习意义上的样本集 X 了), 这里的参数也不是机器学习 model 的参数, 而是 predict 值。

其实作为损失函数的似然函数并不关心你当前的机器学习 model 的参数是怎样的, 毕竟它此时所接收的输入只有两部分: 1、predict。2、label。3、分布模型 (predict 服从的分布)。

显然这里的 label 就是似然函数的观测值, 即样本集。而它眼里的模型, 当然就是 predict 这个随机变量所服从的概率分布模型。它的目的, 就是衡量 predict 背后的模型对于当前观测值的解释程度。而每个样本的 predict 值, 恰恰就是它所服从的分布模型的参数。

比如此时我们的机器学习任务是一个 4 个类别的分类任务, 机器学习 model 的输出就是当前样本 X 下的每个类别的概率, 如  $\text{predict}=[0.1, 0.1, 0.7, 0.1]$ , 而该样本的标签是类别 3, 表示成向量就是  $\text{label}=[0, 0, 1, 0]$ 。那么  $\text{label}=[0, 0, 1, 0]$  就是似然函数眼里的样本, 然后我们可以假设 predict 这个随机变量背后的模型是单次观测下的多项式分布, (因为 softmax 本身是基于多项式分布的)。

回顾:

伯努利分布, 也叫做 (0, 1) 分布, 贝努利分布可以看成是将一枚硬币 (只有正反两个面, 代表两个类别) 向上扔出, 出现某个面 (类别) 的概率情况, 因此其概率密度函数为:

$$f(x) = p^x(1-p)^{1-x} = \begin{cases} p, & x = 1 \\ q, & x = 0 \end{cases}$$

这是理解似然函数做损失函数的关键！另外，贝努利分布的模型参数就是其中一个类别的发生概率。

而二项分布呢，就是将贝努利实验重复 n 次（各次实验之间是相互独立的）。

而多项式分布呢，就是将二项分布推广到多个面（类别）。

所以，单次观测下的多项式分布就是贝努利分布的多类推广！即：

$$f_{multi}(x; p) = \prod_{i=1}^C p_i^{x_i}$$

其中，C 代表类别数。p 代表向量形式的模型参数，即各个类别的发生概率，如  $p=[0.1, 0.1, 0.7, 0.1]$ ，则  $p_1=0.1, p_3=0.7$  等。即，多项式分布的模型参数就是各个类别的发生概率！x 代表 one-hot 形式的观测值，如 x= 类别 3，则  $x=[0, 0, 1, 0]$ 。xi 代表 x 的第 i 个元素，比如 x= 类别 3 时， $x_1=0, x_2=0, x_3=1, x_4=0$ 。

想一下，机器学习 model 对某个样本的输出，就代表各个类别发生的概率。但是，对于当前这一个样本而言，它肯定只能有一个类别，所以这一个样本就可以看成是一次实验（观察），而这次实验（观察）的结果要服从上述各个类别发生的概率，那不就是服从多项式分布嘛！而且是单次观察！各个类别发生的概率 predict 当然就是这个多项式分布的参数。

总结一下，对于多类分类问题，似然函数就是衡量当前这个以 predict 为参数的单次观测下的多项式分布模型与样本值 label 之间的似然度。

所以，根据似然函数的定义，单个样本的似然函数即：

$$L = f_{multi}(label; predict)$$

所以，整个样本集（或者一个 batch）的似然函数即：

$$L = \prod_X f_{multi}(label; predict) = \prod_X \prod_{i=1}^C predict(i)^{label(i)}$$

所以在累乘号前面加上 log 函数后，就成了所谓的对数似然函数：

$$L = \sum_X \sum_{i=1}^C label(i) \log(predict(i))$$

而最大化对数似然函数就等效于最小化负对数似然函数，所以前面加个负号就和交叉熵的形式相同的了。

交叉熵定义：对于某种分布的随机变量  $X \sim p(x)$ , 有一个模型  $q(x)$  用于近似  $p(x)$  的概率分布，则分布  $X$  与模型  $q$  之间的交叉熵即：

$$H(X, q) = - \sum_x p(x) \log q(x)$$

这里  $X$  的分布模型即样本集 label 的真实分布模型，这里模型  $q(x)$  即想要模拟真实分布模型的机器学习模型。可以说交叉熵是直接衡量两个分布，或者说两个 model 之间的差异。而似然函数则是解释以 model 的输出为参数的某分布模型对样本集的解释程度。因此，可以说这两者是“同貌不同源”，但是“殊途同归”啦。

tips:

最大似然估计：

给定一堆数据，假如我们知道它是从某一种分布中随机取出来的，可是我们并不知道这个分布具体的参，即“模型已定，参数未知”。例如，我们知道这个分布是正态分布，但是不知道均值和方差；或者是二项分布，但是不知道均值。最大似然估计 (MLE, Maximum Likelihood Estimation) 就可以用来估计模型的参数。**MLE 的目标是找出一组参数，使得模型产生出观测数据的概率最大。**

**5.5. 2.10.5 为什么用交叉熵代替二次代价函数.** (1) **为什么不用二次方代价函数**由上一节可知，权值  $w$  和偏置  $b$  的偏导数为  $\frac{\partial J}{\partial w} = (a - y)\sigma'(z)x$ ,  $\frac{\partial J}{\partial b} = (a - y)\sigma'(z)$ , 偏导数受激活函数的导数影响，sigmoid 函数导数在输出接近 0 和 1 时非常小，会导致一些实例在刚开始训练时学习得非常慢。

(2) **为什么要用交叉熵**交叉熵函数权值  $w$  和偏置  $b$  的梯度推导为：

$$\frac{\partial J}{\partial w_j} = \frac{1}{n} \sum_x x_j (\sigma(z) - y) \quad \frac{\partial J}{\partial b} = \frac{1}{n} \sum_x (\sigma(z) - y)$$

由以上公式可知，权重学习的速度受到  $\sigma(z) - y$  影响，更大的误差，就有更快的学习速度，避免了二次代价函数方程中因  $\sigma'(z)$  导致的学习缓慢的情况。

## 6. 2.11 损失函数

**6.1. 2.11.1 什么是损失函数.** 损失函数 (Loss Function) 又叫做误差函数，用来衡量算法的运行情况，估量模型的预测值与真实值的不一致程度，是一个非负实值函数，通常使用  $L(Y, f(x))$  来表示。损失函数越小，模型的鲁棒性就越好。损失函数是经验风险函数的核心部分，也是结构风险函数重要组成部分。

**6.2. 2.11.2 常见的损失函数.** 机器学习通过对算法中的目标函数进行不断求解优化，得到最终想要的结果。分类和回归问题中，通常使用损失函数或代价函数作为目标函数。损失函数用来评价预测值和真实值不一样的程度。通常损失函数越好，模型的性能也越好。损失函数可分为经验风险损失函数和结构风险损失函数。经验风险损失函数指预测结果和实际结果的差别，结构风险损失函数是在经验风险损失函数上加上正则项。下面介绍常用的损失函数：

(1) **0-1 损失函数**如果预测值和目标值相等，值为 0，如果不相等，值为 1。

$$L(Y, f(x)) = \begin{cases} 1, & Y \neq f(x) \\ 0, & Y = f(x) \end{cases}$$

一般的在实际使用中，相等的条件过于严格，可适当放宽条件：

$$L(Y, f(x)) = \begin{cases} 1, & |Y - f(x)| \geq T \\ 0, & |Y - f(x)| < T \end{cases}$$

(2) **绝对值损失函数**和 0-1 损失函数相似，绝对值损失函数表示为：

$$L(Y, f(x)) = |Y - f(x)|$$

(3) **平方损失函数**

$$L(Y, f(x)) = \sum_N (Y - f(x))^2$$

这点可从最小二乘法和欧几里得距离角度理解。最小二乘法的原理是，最优拟合曲线应该使所有点到回归直线的距离和最小。

(4) **对数损失函数**

$$L(Y, P(Y|X)) = -\log P(Y|X) = -\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^M y_{ij} \log(p_{ij})$$

其中，Y 为输出变量，X 为输入变量，L 为损失函数。N 为输入样本量，M 为可能的类别数， $y_{ij}$  是一个二值指标，表示类别 j 是否是输入实例  $x_i$  的真实类别。 $p_{ij}$  为模型或分类器预测输入实例  $x_i$  属于类别 j 的概率。

常见的逻辑回归使用的就是对数损失函数，有很多人认为逻辑回归的损失函数是平方损失，其实不然。逻辑回归它假设样本服从伯努利分布（0-1 分布），进而求得满足该分布的似然函数，接着取对数求极值等。逻辑回归推导出的经验风险函数是最小化负的似然函数，从损失函数的角度看，就是对数损失函数。形式上等价于二分类的交叉熵损失函数。

(6) **指数损失函数**指数损失函数的标准形式为：

$$L(Y, f(x)) = \exp(-Y f(x))$$

例如 AdaBoost 就是以指数损失函数为损失函数。

(7) **Hinge 损失函数** Hinge 损失函数的标准形式如下：

$$L(y) = \max(0, 1 - ty)$$

统一的形式：

$$L(Y, f(x)) = \max(0, Yf(x))$$

其中  $y$  是预测值，范围为  $(-1,1)$ ， $t$  为目标值，其为-1 或 1。

在线性支持向量机中，最优化问题可等价于

$$\min_{w,b} \sum_{i=1}^N (1 - y_i(wx_i + b)) + \lambda \|w\|^2$$

上式相似于下式

$$\frac{1}{m} \sum_{i=1}^N l(wx_i + by_i) + \|w\|^2$$

其中  $l(wx_i + by_i)$  是 Hinge 损失函数， $\|w\|^2$  可看做为正则化项。

### 6.3. 2.11.3 逻辑回归为什么使用对数损失函数.

假设逻辑回归模型

$$P(y = 1|x; \theta) = \frac{1}{1 + e^{-\theta^T x}}$$

假设逻辑回归模型的概率分布是伯努利分布，其概率质量函数为：

$$P(X = n) = \begin{cases} 1 - p, & n = 0 \\ p, & n = 1 \end{cases}$$

其似然函数为：

$$L(\theta) = \prod_{i=1}^m P(y = 1|x_i)^{y_i} P(y = 0|x_i)^{1-y_i}$$

对数似然函数为：

$$\ln L(\theta) = \sum_{i=1}^m [y_i \ln P(y = 1|x_i) + (1 - y_i) \ln P(y = 0|x_i)] = \sum_{i=1}^m [y_i \ln P(y = 1|x_i) + (1 - y_i) \ln(1 - P(y = 1|x_i))]$$

对数函数在单个数据点上的定义为：

$$cost(y, p(y|x)) = -y \ln p(y|x) - (1 - y) \ln(1 - p(y|x))$$

则全局样本损失函数为：

$$cost(y, p(y|x)) = - \sum_{i=1}^m [y_i \ln p(y_i|x_i) + (1 - y_i) \ln(1 - p(y_i|x_i))]$$

由此可看出，对数损失函数与极大似然估计的对数似然函数本质上是相同的。所以逻辑回归直接采用对数损失函数。

**6.4. 对数损失函数是如何度量损失的.** 例如，在高斯分布中，我们需要确定均值和标准差。如何确定这两个参数？最大似然估计是比较常用的方法。最大似然的目标是找到一些参数值，这些参数值对应的分布可以最大化观测到数据的概率。因为需要计算观测到所有数据的全概率，即所有观测到的数据点的联合概率。现考虑如下简化情况：

- (1) 假设观测到每个数据点的概率和其他数据点的概率是独立的。
- (2) 取自然对数。假设观测到单个数据点  $x_i (i = 1, 2, \dots, n)$  的概率为：

$$P(x_i; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x_i - \mu)^2}{2\sigma^2}\right)$$

- (3) 其联合概率为：

$$P(x_1, x_2, \dots, x_n; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x_1 - \mu)^2}{2\sigma^2}\right) \times \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x_2 - \mu)^2}{2\sigma^2}\right) \times \dots \times \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x_n - \mu)^2}{2\sigma^2}\right)$$

对上式取自然对数，可得：

$$\ln(P(x_1, x_2, \dots, x_n; \mu, \sigma)) = \ln\left(\frac{1}{\sigma\sqrt{2\pi}}\right) - \frac{(x_1 - \mu)^2}{2\sigma^2} + \ln\left(\frac{1}{\sigma\sqrt{2\pi}}\right) - \frac{(x_2 - \mu)^2}{2\sigma^2} + \dots + \ln\left(\frac{1}{\sigma\sqrt{2\pi}}\right) - \frac{(x_n - \mu)^2}{2\sigma^2}$$

根据对数定律，上式可以化简为：

$$\ln(P(x_1, x_2, \dots, x_n; \mu, \sigma)) = -n \ln(\sigma) - \frac{n}{2} \ln(2\pi) - \frac{1}{2\sigma^2} [(x_1 - \mu)^2 + (x_2 - \mu)^2 + \dots + (x_n - \mu)^2]$$

然后求导为：

$$\frac{\partial \ln(P(x_1, x_2, \dots, x_n; \mu, \sigma))}{\partial \mu} = \frac{n}{\sigma^2} [\mu - (x_1 + x_2 + \dots + x_n)]$$

上式左半部分为对数损失函数。损失函数越小越好，因此我们令等式左半的对数损失函数为0，可得：

$$\mu = \frac{x_1 + x_2 + \dots + x_n}{n}$$

同理，可计算  $\sigma$ 。

## 7. 梯度下降

**7.1. 机器学习中为什么需要梯度下降.** 梯度下降是机器学习中常见优化算法之一，梯度下降法有以下几个作用：

- (1) 梯度下降是迭代法的一种，可以用于求解最小二乘问题。
- (2) 在求解机器学习算法的模型参数，即无约束优化问题时，主要有梯度下降法（Gradient Descent）和最小二乘法。
- (3) 在求解损失函数的最小值时，可以通过梯度下降法来一步步的迭代求解，得到最小化的损失函数和模型参数值。
- (4) 如果我们需要求解损失函数的最大值，可通过梯度上升法来迭代。梯度下降法和梯度上升法可相互转换。
- (5) 在机器学习中，梯度下降法主要有随机梯度下降法和批量梯度下降法。

**7.2. 梯度下降法缺点.** 梯度下降法缺点有以下几点：

- (1) 靠近极小值时收敛速度减慢。
- (2) 直线搜索时可能会产生一些问题。
- (3) 可能会“之字形”地下降。

梯度概念也有需注意的地方：

- (1) 梯度是一个向量，即有方向有大小。
- (2) 梯度的方向是最大方向导数的方向。
- (3) 梯度的值是最大方向导数的值。

**7.3. 梯度下降法直观理解.** 梯度下降法经典图示如下图 2.7 所示：

图 2.7 梯度下降法经典图示

形象化举例，由上图 2.7 所示，假如最开始，我们在一座大山上的某处位置，因为到处都是陌生的，不知道下山的路，所以只能摸索着根据直觉，走一步算一步，在此过程中，每走到一个位置的时候，都会求解当前位置的梯度，沿着梯度的负方向，也就是当前最陡峭的位置向下走一步，然后继续求解当前位置梯度，向这一步所在位置沿着最陡峭最易下山的位置走一步。不断循环求梯度，就这样一步步地走下去，一直走到我们认为已经到了山脚。当然这样走下去，有可能我们不能走到山脚，而是到了某一个局部的山势低处。由此，从上面的解释可以看出，梯度下降不一定能够找到全局的最优解，有可能是一个局部的最优解。当然，如果损失函数是凸函数，梯度下降法得到的解就一定是全局最优解。

**核心思想归纳:**

- (1) 初始化参数，随机选取取值范围内的任意数；
- (2) 迭代操作：a) 计算当前梯度；b) 修改新的变量；c) 计算朝最陡的下坡方向走一步；d) 判断是否需要终止，如否，返回 a)；

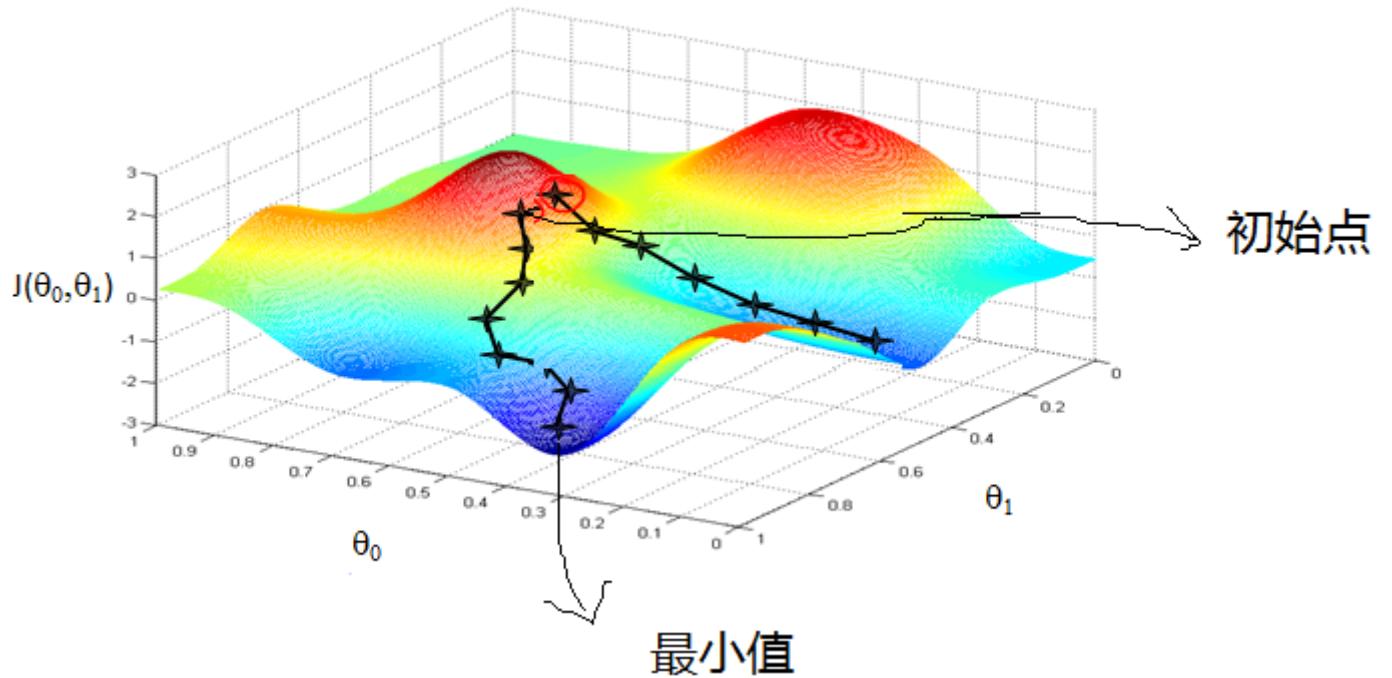


图 6.

(3) 得到全局最优解或者接近全局最优解。

#### 7.4. 梯度下降法算法描述. 梯度下降法算法步骤如下:

(1) 确定优化模型的假设函数及损失函数。举例，对于线性回归，假设函数为：

$$h_{\theta}(x_1, x_2, \dots, x_n) = \theta_0 + \theta_1 x_1 + \dots + \theta_n x_n$$

其中， $\theta_i, x_i (i = 0, 1, 2, \dots, n)$  分别为模型参数、每个样本的特征值。对于假设函数，损失函数为：

$$J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{2m} \sum_{j=0}^m (h_{\theta}(x_0^{(j)}, x_1^{(j)}, \dots, x_n^{(j)}) - y_j)^2$$

(2) 相关参数初始化。主要初始化  $\theta_i$ 、算法迭代步长  $\alpha$ 、终止距离  $\zeta$ 。初始化时可以根据经验初始化，即  $\theta$  初始化为 0，步长  $\alpha$  初始化为 1。当前步长记为  $\varphi_i$ 。当然，也可随机初始化。

(3) 迭代计算。

1) 计算当前位置时损失函数的梯度, 对  $\theta_i$ , 其梯度表示为:

$$\frac{\partial}{\partial \theta_i} J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{2m} \sum_{j=0}^m (h_\theta(x_0^{(j)}, x_1^{(j)}, \dots, x_n^{(j)}) - y_j)^2$$

2) 计算当前位置下降的距离。

$$\varphi_i = \alpha \frac{\partial}{\partial \theta_i} J(\theta_0, \theta_1, \dots, \theta_n)$$

3) 判断是否终止。确定是否所有  $\theta_i$  梯度下降的距离  $\varphi_i$  都小于终止距离  $\zeta$ , 如果都小于  $\zeta$ , 则算法终止, 当然的值即为最终结果, 否则进入下一步。4) 更新所有的  $\theta_i$ , 更新后的表达式为:

$$\theta_i = \theta_i - \alpha \frac{\partial}{\partial \theta_i} J(\theta_0, \theta_1, \dots, \theta_n)$$

$$\theta_i = \theta_i - \alpha \frac{1}{m} \sum_{j=0}^m (h_\theta(x_0^{(j)}, x_1^{(j)}, \dots, x_n^{(j)}) - y_j) x_i^{(j)}$$

5) 令上式  $x_0^{(j)} = 1$ , 更新完毕后转入 1)。由此, 可看出, 当前位置的梯度方向由所有样本决定, 上式中  $\frac{1}{m}$ 、 $\alpha \frac{1}{m}$  的目的是为了便于理解。

**7.5. 如何对梯度下降法进行调优.** 实际使用梯度下降法时, 各项参数指标不能一步就达到理想状态, 对梯度下降法调优主要体现在以下几个方面:

(1) **算法迭代步长  $\alpha$  选择.** 在算法参数初始化时, 有时根据经验将步长初始化为 1。实际取值取决于数据样本。可以从大到小, 多取一些值, 分别运行算法看迭代效果, 如果损失函数在变小, 则取值有效。如果取值无效, 说明要增大步长。但步长太大, 有时会导致迭代速度过快, 错过最优解。步长太小, 迭代速度慢, 算法运行时间长。

(2) **参数的初始值选择.** 初始值不同, 获得的最小值也有可能不同, 梯度下降有可能得到的是局部最小值。如果损失函数是凸函数, 则一定是最优解。由于有局部最优解的风险, 需要多次用不同初始值运行算法, 关键损失函数的最小值, 选择损失函数最小化的初值。

(3) **标准化处理.** 由于样本不同, 特征取值范围也不同, 导致迭代速度慢。为了减少特征取值的影响, 可对特征数据标准化, 使新期望为 0, 新方差为 1, 可节省算法运行时间。

**7.6. 随机梯度和批量梯度区别.** 随机梯度下降 (SGD) 和批量梯度下降 (BGD) 是两种主要梯度下降法, 其目的是增加某些限制来加速运算求解。下面通过介绍两种梯度下降法的求解思路, 对其进行比较。假设函数为:

$$h_\theta(x_0, x_1, \dots, x_n) = \theta_0 x_0 + \theta_1 x_1 + \dots + \theta_n x_n$$

损失函数为：

$$J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{2m} \sum_{j=0}^m (h_\theta(x_0^j, x_1^j, \dots, x_n^j) - y^j)^2$$

其中， $m$  为样本个数， $j$  为参数个数。

1、批量梯度下降的求解思路如下： a) 得到每个  $\theta$  对应的梯度：

$$\frac{\partial}{\partial \theta_i} J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{m} \sum_{j=0}^m (h_\theta(x_0^j, x_1^j, \dots, x_n^j) - y^j) x_i^j$$

b) 由于是求最小化风险函数，所以按每个参数  $\theta$  的梯度负方向更新  $\theta_{-i}$ ：

$$\theta_i = \theta_i - \frac{1}{m} \sum_{j=0}^m (h_\theta(x_0^j, x_1^j, \dots, x_n^j) - y^j) x_i^j$$

c) 从上式可以注意到，它得到的虽然只是一个全局最优解，但每迭代一步，都要用到训练集所有的数据，如果样本数据很大，这种方法迭代速度就很慢。相比而言，随机梯度下降可避免这种问题。

2、随机梯度下降的求解思路如下： a) 相比批量梯度下降对应所有的训练样本，随机梯度下降法中损失函数对应的是训练集中每个样本的粒度。损失函数可以写成如下这种形式，

$$J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{m} \sum_{j=0}^m (y^j - h_\theta(x_0^j, x_1^j, \dots, x_n^j))^2 = \frac{1}{m} \sum_{j=0}^m cost(\theta, (x^j, y^j))$$

b) 对每个参数  $\theta$  按梯度方向更新  $\theta$ ：

$$\theta_i = \theta_i + (y^j - h_\theta(x_0^j, x_1^j, \dots, x_n^j))$$

c) 随机梯度下降是通过每个样本来迭代更新一次。随机梯度下降伴随的一个问题是噪音较批量梯度下降要多，使得随机梯度下降并不是每次迭代都向着整体最优化方向。

**小结：**随机梯度下降法、批量梯度下降法相对来说都比较极端，简单对比如下：

方法	特点
批量梯度下降	a) 采用所有数据来梯度下降。b) 批量梯度下降法在样本量很大的时候，训练速度慢。
随机梯度下降	a) 随机梯度下降用一个样本来梯度下降。b) 训练速度很快。c) 随机梯度下降法仅仅用一个样本决定梯度方向，导致解有可能不是全局最优。d) 收敛速度来说，随机梯度下降法一次迭代一个样本，导致迭代方向变化很大，不能很快的收敛到局部最优解。

下面介绍能结合两种方法优点的小批量梯度下降法。

**3、小批量（Mini-Batch）梯度下降的求解思路**如下对于总数为  $m$  个样本的数据，根据样本的数据，选取其中的  $n(1 < n < m)$  个子样本来迭代。其参数  $\theta$  按梯度方向更新  $\theta_i$  公式如下：

$$\theta_i = \theta_i - \alpha \sum_{j=t}^{t+n-1} (h_\theta(x_0^j, x_1^j, \dots, x_n^j) - y^j) x_i^j$$

**7.7. 2.12.7 各种梯度下降法性能比较.** 下表简单对比随机梯度下降 (SGD)、批量梯度下降 (BGD)、小批量梯度下降 (Mini-batch GD)、和 Online GD 的区别：

	BGD	SGD	Mini-batch GD	Online GD
训练集	固定	固定	固定	实时更新
单次迭代样本数	整个训练集	单个样本	训练集的子集	根据具体算法定
算法复杂度	高	低	一般	低
时效性	低	一般	一般	高
收敛性	稳定	不稳定	较稳定	不稳定

BGD、SGD、Mini-batch GD，前面均已讨论过，这里介绍一下 Online GD。

Online GD 于 Mini-batch GD/SGD 的区别在于，所有训练数据只用一次，然后丢弃。这样做的优点在于可预测最终模型的变化趋势。

Online GD 在互联网领域用的较多，比如搜索广告的点击率 (CTR) 预估模型，网民的点击行为会随着时间改变。用普通的 BGD 算法（每天更新一次）一方面耗时较长（需要对所有历史数据重新训练）；另一方面，无法及时反馈用户的点击行为迁移。而 Online GD 算法可以实时的依据网民的点击行为进行迁移。

## 8. 2.14 线性判别分析 (LDA)

**8.1. 2.14.1 LDA 思想总结.** 线性判别分析 (Linear Discriminant Analysis, LDA) 是一种经典的降维方法。和主成分分析 PCA 不考虑样本类别输出的无监督降维技术不同，LDA 是一种监督学习的降维技术，数据集的每个样本有类别输出。

LDA 分类思想简单总结如下：

1. 多维空间中，数据处理分类问题较为复杂，LDA 算法将多维空间中的数据投影到一条直线上，将  $d$  维数据转化成 1 维数据进行处理。
2. 对于训练数据，设法将多维数据投影到一条直线上，同类数据的投影点尽可能接近，异类数据点尽可能远离。
3. 对数据进行分类时，将其投影到同样的这条直线上，再根据投影点的位置来确定样本的类别。

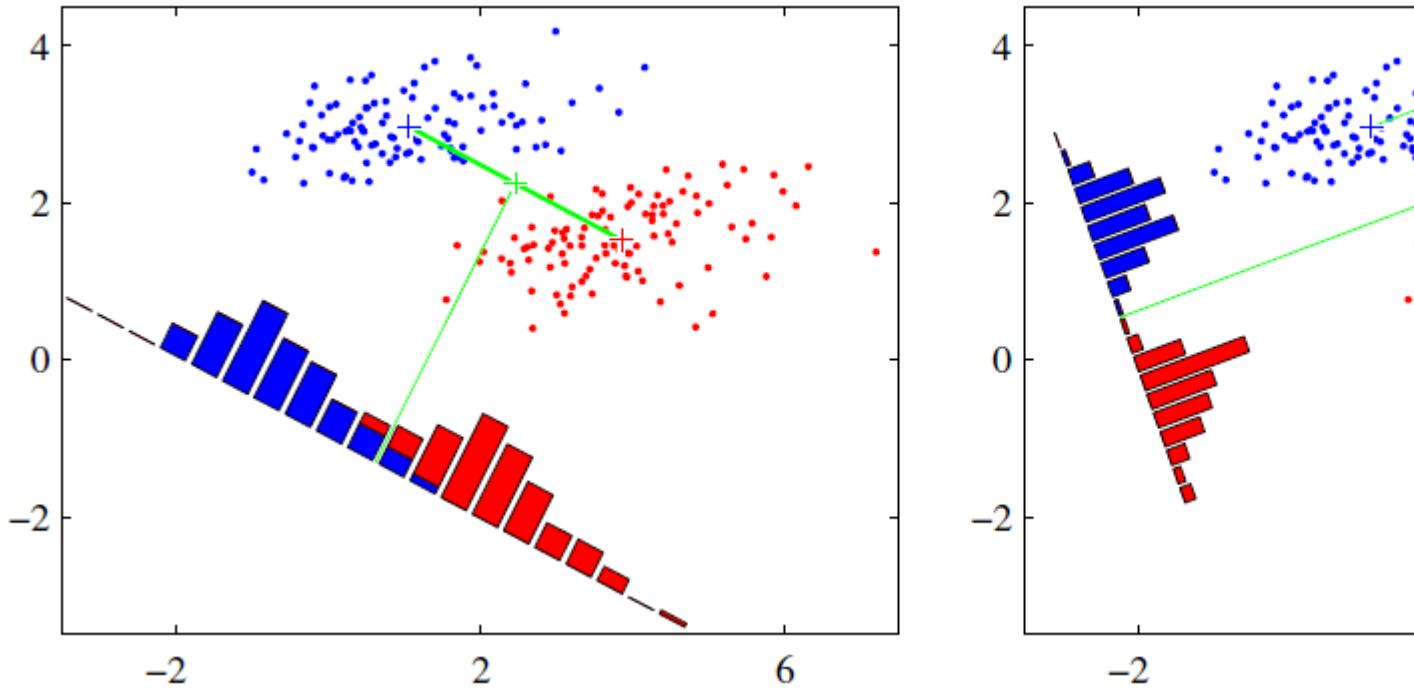


图 7.

如果用一句话概括 LDA 思想，即“投影后类内方差最小，类间方差最大”。

**8.2. 2.14.2 图解 LDA 核心思想.** 假设有红、蓝两类数据，这些数据特征均为二维，如下图所示。我们的目标是将这些数据投影到一维，让每一类相近的数据的投影点尽可能接近，不同类别数据尽可能远，即图中红色和蓝色数据中心之间的距离尽可能大。

左图和右图是两种不同的投影方式。

左图思路：让不同类别的平均点距离最远的投影方式。

右图思路：让同类别的数据挨得最近的投影方式。

从上图直观看出，右图红色数据和蓝色数据在各自的区域来说相对集中，根据数据分布直方图也可看出，所以右图的投影效果好于左图，左图中间直方图部分有明显交集。

以上例子是基于数据是二维的，分类后的投影是一条直线。如果原始数据是多维的，则投影后的分类面是一低维的超平面。

**8.3. 2.14.3 二类 LDA 算法原理.** 输入：数据集  $D = (\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \dots, (\mathbf{x}_m, \mathbf{y}_m)$ ，其中样本  $\mathbf{x}_i$  是  $n$  维向量， $\mathbf{y}_i \in \{0, 1\}$ ，降维后的目标维度  $d$ 。定义

$N_j(j=0,1)$  为第  $j$  类样本个数;  
 $X_j(j=0,1)$  为第  $j$  类样本的集合;  
 $u_j(j=0,1)$  为第  $j$  类样本的均值向量;  
 $\Sigma_j(j=0,1)$  为第  $j$  类样本的协方差矩阵。  
其中

$$u_j = \frac{1}{N_j} \sum_{\mathbf{x} \in X_j} \mathbf{x} (j=0,1) \sum_j = \sum_{\mathbf{x} \in X_j} (\mathbf{x} - u_j)(\mathbf{x} - u_j)^T (j=0,1)$$

假设投影直线是向量  $\mathbf{w}$ , 对任意样本  $\mathbf{x}_i$ , 它在直线  $w$  上的投影为  $\mathbf{w}^T \mathbf{x}_i$ , 两个类别的中心点  $u_0, u_1$  在直线  $w$  的投影分别为  $\mathbf{w}^T u_0, \mathbf{w}^T u_1$ 。

LDA 的目标是让两类别的数据中心间的距离  $\|\mathbf{w}^T u_0 - \mathbf{w}^T u_1\|_2^2$  尽量大, 与此同时, 希望同类样本投影点的协方差  $\mathbf{w}^T \Sigma_0 \mathbf{w}, \mathbf{w}^T \Sigma_1 \mathbf{w}$  尽量小, 最小化  $\mathbf{w}^T \Sigma_0 \mathbf{w} + \mathbf{w}^T \Sigma_1 \mathbf{w}$ 。定义类内散度矩阵

$$S_w = \sum_0 + \sum_1 = \sum_{\mathbf{x} \in X_0} (\mathbf{x} - u_0)(\mathbf{x} - u_0)^T + \sum_{\mathbf{x} \in X_1} (\mathbf{x} - u_1)(\mathbf{x} - u_1)^T$$

类间散度矩阵  $S_b = (u_0 - u_1)(u_0 - u_1)^T$

据上分析, 优化目标为

$$\arg \max$$

$\mathbf{w}^T \mathbf{J}(\mathbf{w}) = \|\mathbf{w}^T u_0 - \mathbf{w}^T u_1\|_2^2 \frac{\mathbf{w}^T \Sigma_0 \mathbf{w} + \mathbf{w}^T \Sigma_1 \mathbf{w}}{\mathbf{w}^T (\Sigma_0 + \Sigma_1) \mathbf{w}} = \frac{\mathbf{w}^T (u_0 - u_1)(u_0 - u_1)^T \mathbf{w}}{\mathbf{w}^T (\Sigma_0 + \Sigma_1) \mathbf{w}} = \frac{\mathbf{w}^T S_b \mathbf{w}}{\mathbf{w}^T S_w \mathbf{w}}$  根据广义瑞利商的性质, 矩阵  $S_w^{-1} S_b$  的最大特征值为  $J(\mathbf{w})$  的最大值, 矩阵  $S_w^{-1} S_b$  的最大特征值对应的特征向量即为  $\mathbf{w}$ 。

#### 8.4. LDA 算法流程总结.

LDA 算法降维流程如下:

输入: 数据集  $D = (x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)$ , 其中样本  $x_i$  是  $n$  维向量,  $y_i \in C_1, C_2, \dots, C_k$ , 降维后的目标维度  $d$ 。

输出: 降维后的数据集  $\bar{D}$ 。

步骤: 1. 计算类内散度矩阵  $S_w$ 。2. 计算类间散度矩阵  $S_b$ 。3. 计算矩阵  $S_w^{-1} S_b$ 。4. 计算矩阵  $S_w^{-1} S_b$  的最大的  $d$  个特征值。5. 计算  $d$  个特征值对应的  $d$  个特征向量, 记投影矩阵为  $W$ 。6. 转化样本集的每个样本, 得到新样本  $P_i = W^T x_i$ 。7. 输出新样本集  $\bar{D} = (p_1, y_1), (p_2, y_2), \dots, (p_m, y_m)$

异同点 LDA

相同点 1. 两者均可以对数据进行降维; 2. 两者在降维时均使用了矩阵特征分解的思想; 3. 两者都假设数据

不同点 有监督的降维方法;

降维最多降到  $k-1$  维;  
可以用于降维，还可以用于分类;  
选择分类性能最好的投影方向;  
更明确，更能反映样本间差异;

### 8.5. LDA 和 PCA 区别.

优缺点	简要说明
-----	------

优点 1. 可以使用类别先验知识；  
2. 以标签、类别衡量差异性的有监督降维方式，相对于PCA的模糊性，其目的更

缺点 1.

LDA

不适合对非高斯分布样本进行降维；

2.

LDA

降维最多降到分类数  $k-1$  维；

3.

LDA

在样本分类信息依赖方差

## 8.6. LDA 优缺点.

# 9. 主成分分析 (PCA)

### 9.1. 主成分分析 (PCA) 思想总结.

1. PCA 就是将高维的数据通过线性变换投影到低维空间上去。
2. 投影思想：找出最能够代表原始数据的投影方法。被 PCA 降掉的那些维度只能是那些噪声或是冗余的数据。
3. 去冗余：去除可以被其他向量代表的线性相关向量，这部分信息量是多余的。
4. 去噪声，去除较小特征值对应的特征向量，特征值的大小反映了变换后在特征向量方向上变换的幅度，幅度越大，说明这个方向上的元素差异也越大，要保留。
5. 对角化矩阵，寻找极大线性无关组，保留较大的特征值，去除较小特征值，组成一个投影矩阵，对原始样本矩阵进行投影，得到降维后的新样本矩阵。
6. 完成 PCA 的关键是——协方差矩阵。协方差矩阵，能同时表现不同维度间的相关性以及各个维度上的方差。协方差矩阵度量的是维度与维度之间的关系，而非样本与样本之间。
7. 之所以对角化，因为对角化之后非对角上的元素都是 0，达到去噪声的目的。对角化后的协方差矩阵，对角线上较小的新方差对应的就是那些该去掉的维度。所以我们只取那些含有较大能量 (特征值) 的维度，其余的就舍掉，即去冗余。

**9.2. 图解 PCA 核心思想.** PCA 可解决训练数据中存在数据特征过多或特征累赘的问题。核心思想是将  $m$  维特征映射到  $n$  维 ( $n < m$ )，这  $n$  维形成主元，是重构出来最能代表原始数据的正交特征。

假设数据集是  $m$  个  $n$  维， $(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(m)})$ 。如果  $n = 2$ ，需要降维到  $n' = 1$ ，现在想找到某一维度方向代表这两个维度的数据。下图有  $u_1, u_2$  两个向量方向，但是哪个向量才是我们所想要的，可以更好代表原始数据集的呢？

从图可看出， $u_1$  比  $u_2$  好，为什么呢？有以下两个主要评价指标：1. 样本点到这个直线的距离足够近。2. 样本点在这个直线上的投影能尽可能的分开。

如果我们需要降维的目标维数是其他任意维，则：1. 样本点到这个超平面的距离足够近。2. 样本点在这个超平面上的投影能尽可能的分开。

### 9.3. PCA 算法推理. 下面以基于最小投影距离为评价指标推理：

假设数据集是  $m$  个  $n$  维， $(x^{(1)}, x^{(2)}, \dots, x^{(m)})$ ，且数据进行了中心化。经过投影变换得到新坐标为  $w_1, w_2, \dots, w_n$ ，其中  $w$  是标准正交基，即  $\|w\|_2 = 1$ ,  $w_i^T w_j = 0$ 。

经过降维后，新坐标为  $w_1, w_2, \dots, w_n$ ，其中  $n'$  是降维后的目标维数。样本点  $x^{(i)}$  在新坐标系下的投影为  $z^{(i)} = (z_1^{(i)}, z_2^{(i)}, \dots, z_{n'}^{(i)})$ ，其中  $z_j^{(i)} = w_j^T x^{(i)}$  是  $x^{(i)}$  在低维坐标系里第  $j$  维的坐标。

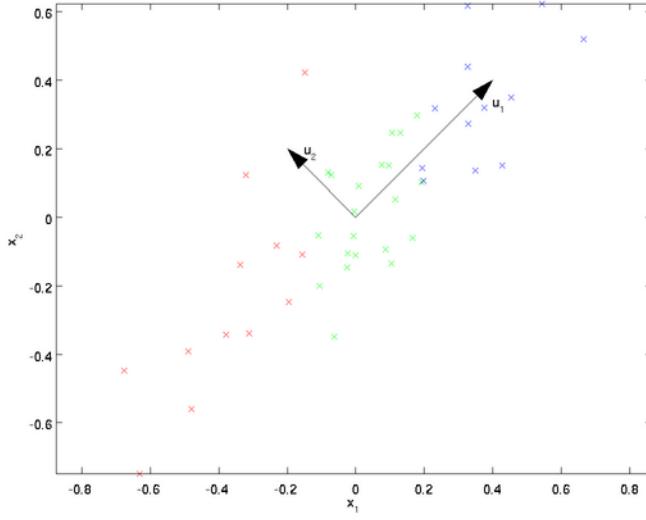


图 8.

如果用  $z^{(i)}$  去恢复  $x^{(i)}$ ，则得到的恢复数据为  $\hat{x}^{(i)} = \sum_{j=1}^{n'} x_j^{(i)} w_j = W z^{(i)}$ ，其中  $W$  为标准正交基组成的矩阵。

考虑到整个样本集，样本点到这个超平面的距离足够近，目标变为最小化  $\sum_{i=1}^m \|\hat{x}^{(i)} - x^{(i)}\|_2^2$ 。对此式进行推理，可得：

$$\sum_{i=1}^m \|\hat{x}^{(i)} - x^{(i)}\|_2^2 = \sum_{i=1}^m \|W z^{(i)} - x^{(i)}\|_2^2 = \sum_{i=1}^m (W z^{(i)})^T (W z^{(i)}) - 2 \sum_{i=1}^m (W z^{(i)})^T x^{(i)} + \sum_{i=1}^m (x^{(i)})^T x^{(i)} = \sum_{i=1}^m (z^{(i)})^T (z^{(i)}) - 2$$

在推导过程中，分别用到了  $\bar{x}^{(i)} = W z^{(i)}$ ，矩阵转置公式  $(AB)^T = B^T A^T$ ， $W^T W = I$ ， $z^{(i)} = W^T x^{(i)}$  以及矩阵的迹，最后两步是将代数和转为矩阵形式。由于  $W$  的每一个向量  $w_j$  是标准正交基， $\sum_{i=1}^m x^{(i)} (x^{(i)})^T$  是数据集的协方差矩阵， $\sum_{i=1}^m x^{(i)} (x^{(i)})^T$  是一个常量。最小化  $\sum_{i=1}^m \|\hat{x}^{(i)} - x^{(i)}\|_2^2$  又可等价于

$$\underbrace{\arg \min_W}_{W} -\text{tr}(W^T X X^T W) \text{ s.t. } W^T W = I$$

利用拉格朗日函数可得到

$$J(W) = -\text{tr}(W^T X X^T W) + \lambda(W^T W - I)$$

对  $W$  求导，可得  $-X X^T W + \lambda W = 0$ ，也即  $X X^T W = \lambda W$ 。 $X X^T$  是  $n'$  个特征向量组成的矩阵， $\lambda$  为  $X X^T$  的特征值。 $W$  即为我们想要的矩阵。对于原始数据，只需要  $z^{(i)} = W^T X^{(i)}$ ，就可把原始数据集降维到最小投影距离的  $n'$  维数据集。

基于最大投影方差的推导，这里就不再赘述，有兴趣的同仁可自行查阅资料。

**9.4. PCA 算法流程总结.** 输入： $n$  维样本集  $D = (x^{(1)}, x^{(2)}, \dots, x^{(m)})$ ，目标降维的维数  $n'$ 。

输出：降维后的新样本集  $D' = (z^{(1)}, z^{(2)}, \dots, z^{(m)})$ 。

主要步骤如下：1. 对所有的样本进行中心化， $x^{(i)} = x^{(i)} - \frac{1}{m} \sum_{j=1}^m j = 1 x^{(j)}$ 。2. 计算样本的协方差矩阵  $XX^T$ 。3. 对协方差矩阵  $XX^T$  进行特征值分解。4. 取出最大的  $n'$  个特征值对应的特征向量  $w_1, w_2, \dots, w_{n'}$ 。5. 标准化特征向量，得到特征向量矩阵  $W$ 。6. 转化样本集中的每个样本  $z^{(i)} = W^T x^{(i)}$ 。7. 得到输出矩阵  $D' = (z^{(1)}, z^{(2)}, \dots, z^{(n)})$ 。注：在降维时，有时不明确目标维数，而是指定降维到的主成分比重阈值  $k(k \in (0, 1])$ 。假设  $n$  个特征值为  $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$ ，则  $n'$  可从  $\sum_{i=1}^{n'} \lambda_i \geq k \times \sum_{i=1}^n \lambda_i$  得到。

简  
要  
优缺 说  
点 明

优点 1. 仅需要以方差衡量信息量, 不受数据集以外的因素影响。

2. 各主成分之间正交, 可消除原始数据成

缺点 1.

主成分各个特征维度的含义具有一定模糊性，不如原始样本特征的解释性强。

2.

方差小的非主成分

### 9.5. PCA 算法主要优缺点.

**9.6. 降维的必要性及目的. 降维的必要性:** 1. 多重共线性和预测变量之间相互关联。多重共线性会导致解空间的不稳定，从而可能导致结果的不连贯。2. 高维空间本身具有稀疏性。一维正态分布有 68% 的值落于正负标准差之间，而在十维空间上只有 2%。3. 过多的变量，对查找规律造成冗余麻烦。4. 仅在变量层面上分析可能会忽略变量之间的潜在联系。例如几个预测变量可能落入仅反映数据某一方面特征的一个组内。

**降维的目的:** 1. 减少预测变量的个数。2. 确保这些变量是相互独立的。3. 提供一个框架来解释结果。相关特征，特别是重要特征更能从数据中明确地显示出来；如果只有二维或者三维的话，更便于可视化展示。4. 数据在低维下更容易处理、更容易使用。5. 去除数据噪声。6. 降低算法运算开销。

**9.7. KPCA 与 PCA 的区别.** 应用 PCA 算法前提是假设存在一个线性超平面，进而投影。那如果数据不是线性的呢？该怎么办？这时候就需要 KPCA，数据集从  $n$  维映射到线性可分的高维  $N > n$ ，然后再从  $N$  维降维到一个低维度  $n' (n' < n < N)$ 。

KPCA 用到了核函数思想，使用了核函数的主成分分析一般称为核主成分分析 (Kernelized PCA，简称 KPCA)。

假设高维空间数据由  $n$  维空间的数据通过映射  $\phi$  产生。

$n$  维空间的特征分解为：

$$\sum_{i=1}^m x^{(i)} (x^{(i)})^T W = \lambda W$$

其映射为

$$\sum_{i=1}^m \phi(x^{(i)}) \phi(x^{(i)})^T W = \lambda W$$

通过在高维空间进行协方差矩阵的特征值分解，然后用和 PCA 一样的方法进行降维。由于 KPCA 需要核函数的运算，因此它的计算量要比 PCA 大很多。

## 10. 模型评估

**10.1. 模型评估常用方法？.** 一般情况下来说，单一评分标准无法完全评估一个机器学习模型。只用 good 和 bad 偏离真实场景去评估某个模型，都是一种欠妥的评估方式。下面介绍常用的分类模型和回归模型评估方法。

**分类模型常用评估方法：**

指标	描述
Accuracy	准确率
Precision	精准度/查准率

Recall	召回率/查全率
P-R 曲线	查准率为纵轴, 查全率为横轴, 作图
F1	F1 值
Confusion Matrix	混淆矩阵
ROC	ROC 曲线
AUC	ROC 曲线下的面积

### 回归模型常用评估方法:

指标	描述
Mean Square Error (MSE, RMSE)	平均方差
Absolute Error (MAE, RAE)	绝对误差
R-Squared	R 平方值

**10.2. 2.16.2 误差、偏差和方差有什么区别和联系.** 在机器学习中, Bias(偏差), Error(误差), 和 Variance(方差) 存在以下区别和联系:

#### 对于 Error :

- 误差 (error): 一般地, 我们把学习器的实际预测输出与样本的真是输出之间的差异称为“误差”。
- $\text{Error} = \text{Bias} + \text{Variance} + \text{Noise}$ , Error 反映的是整个模型的准确度。

#### 对于 Noise:

噪声: 描述了在当前任务上任何学习算法所能达到的期望泛化误差的下界, 即刻画了学习问题本身的难度。

#### 对于 Bias:

- Bias 衡量模型拟合训练数据的能力 (训练数据不一定是整个 training dataset, 而是只用于训练它的那一部分数据, 例如: mini-batch), Bias 反映的是模型在样本上的输出与真实值之间的误差, 即模型本身的精准度。
- Bias 越小, 拟合能力越高 (可能产生 overfitting); 反之, 拟合能力越低 (可能产生 underfitting)。
- 偏差越大, 越偏离真实数据, 如下图第二行所示。

#### 对于 Variance:

- 方差公式:  $S_N^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2$
- Variance 描述的是预测值的变化范围, 离散程度, 也就是离其期望值的距离。方差越大, 数据的分布越分散, 模型的稳定程度越差。

- Variance 反映的是模型每一次输出结果与模型输出期望之间的误差，即模型的稳定性。
- Variance 越小，模型的泛化的能力越高；反之，模型的泛化的能力越低。
- 如果模型在训练集上拟合效果比较优秀，但是在测试集上拟合效果比较差劣，则方差较大，说明模型的稳定程度较差，出现这种现象可能是由于模型对训练集过拟合造成的。如下图右列所示。

**10.3. 经验误差与泛化误差.** 经验误差 (empirical error): 也叫训练误差 (training error)，模型在训练集上的误差。

泛化误差 (generalization error): 模型在新样本集 (测试集) 上的误差称为“泛化误差”。

**10.4. 图解欠拟合、过拟合.** 根据不同的坐标方式，欠拟合与过拟合图解不同。  
**1. 横轴为训练样本数量，纵轴为误差**

如上图所示，我们可以直观看出欠拟合和过拟合的区别：

模型欠拟合：在训练集以及测试集上同时具有较高的误差，此时模型的偏差较大；

模型过拟合：在训练集上具有较低的误差，在测试集上具有较高的误差，此时模型的方差较大。

模型正常：在训练集以及测试集上，同时具有相对较低的偏差以及方差。

### 2. 横轴为模型复杂程度，纵轴为误差

红线为测试集上的 Error, 蓝线为训练集上的 Error

模型欠拟合：模型在点 A 处，在训练集以及测试集上同时具有较高的误差，此时模型的偏差较大。

模型过拟合：模型在点 C 处，在训练集上具有较低的误差，在测试集上具有较高的误差，此时模型的方差较大。

模型正常：模型复杂程度控制在点 B 处为最优。

### 3. 横轴为正则项系数，纵轴为误差

红线为测试集上的 Error, 蓝线为训练集上的 Error

模型欠拟合：模型在点 C 处，在训练集以及测试集上同时具有较高的误差，此时模型的偏差较大。

模型过拟合：模型在点 A 处，在训练集上具有较低的误差，在测试集上具有较高的误差，此时模型的方差较大。它通常发生在模型过于复杂的情况下，如参数过多等，会使得模型的预测性能变弱，并且增加数据的波动性。虽然模型在训练时的效果可以表现的很完美，基本上记住了数据的全部特点，但这种模型在未知数据的表现能力会大减折扣，因为简单的模型泛化能力通常都是很弱的。

模型正常：模型复杂程度控制在点 B 处为最优。

**10.5. 如何解决过拟合与欠拟合. 如何解决欠拟合:** 1. 添加其他特征项。组合、泛化、相关性、上下文特征、平台特征等特征是特征添加的重要手段，有时候特征项不够会导致模型欠拟合。2. 添加多项式特征。例如将线性模型添加二次项或三次项使模型泛化能力更强。例如，FM (Factorization Machine) 模型、FFM (Field-aware Factorization Machine) 模型，其实都是线性模型，增加了二阶多项式，保证了模型一定的拟合程度。3. 可以增加模型的复杂程度。4. 减小正则化系数。正则化的目的是用来防止过拟合的，但是现在模型出现了欠拟合，则需要减少正则化参数。

**如何解决过拟合:** 1. 重新清洗数据，数据不纯会导致过拟合，此类情况需要重新清洗数据。2. 增加训练样本数量。3. 降低模型复杂程度。4. 增大正则项系数。5. 采用 dropout 方法，dropout 方法，通俗的讲就是在训练的时候让神经元以一定的概率不工作。6. early stopping。7. 减少迭代次数。8. 增大学习率。9. 添加噪声数据。10. 树结构中，可以对树进行剪枝。11. 减少特征项。

欠拟合和过拟合这些方法，需要根据实际问题，实际模型，进行选择。

**10.6. 交叉验证的主要作用.** 为了得到更为稳健可靠的模型，对模型的泛化误差进行评估，得到模型泛化误差的近似值。当有多个模型可以选择时，我们通常选择“泛化误差”最小的模型。

交叉验证的方法有许多种，但是最常用的是：留一交叉验证、 $k$  折交叉验证。

### 10.7. 理解 $k$ 折交叉验证.

1. 将含有  $N$  个样本的数据集，分成  $K$  份，每份含有  $N/K$  个样本。选择其中 1 份作为测试集，另外  $K-1$  份作为训练集，测试集就有  $K$  种情况。
2. 在每种情况中，用训练集训练模型，用测试集测试模型，计算模型的泛化误差。
3. 交叉验证重复  $K$  次，每份验证一次，平均  $K$  次的结果或者使用其它结合方式，最终得到一个单一估测，得到模型最终的泛化误差。
4. 将  $K$  种情况下，模型的泛化误差取均值，得到模型最终的泛化误差。
  
5. 一般  $2 \leq K \leq 10$ 。 $k$  折交叉验证的优势在于，同时重复运用随机产生的子样本进行训练和验证，每次的结果验证一次，10 折交叉验证是最常用的。
6. 训练集中样本数量要足够多，一般至少大于总样本数的 50%。
7. 训练集和测试集必须从完整的数据集中均匀取样。均匀取样的目的是希望减少训练集、测试集与原数据集之间的偏差。当样本数量足够多时，通过随机取样，便可以实现均匀取样的效果。

### 10.8. 混淆矩阵. 第一种混淆矩阵:

真实情况 T or F	预测为正例 1, P	预测为负例 0, N
-------------	------------	------------

本来 label 标记为 1, 预测结果真为 T、假为 F    TP(预测为 1, 实际为 1)    FN(预测为 0, 实际为 1)  
 本来 label 标记为 0, 预测结果真为 T、假为 F    FP(预测为 1, 实际为 0)    TN(预测为 0, 实际也为 0)

第二种混淆矩阵:

预测情况 P or N	实际 label 为 1, 预测对了为 T	实际 label 为 0, 预测对了为 T
预测为正例 1, P	TP(预测为 1, 实际为 1)	FP(预测为 1, 实际为 0)
预测为负例 0, N	FN(预测为 0, 实际为 1)	TN(预测为 0, 实际也为 0)

### 10.9. 错误率及精度.

1. 错误率 (Error Rate): 分类错误的样本数占样本总数的比例。
2. 精度 (accuracy): 分类正确的样本数占样本总数的比例。

**10.10. 查准率与查全率.** 将算法预测的结果分成四种情况: 1. 正确肯定 (True Positive,TP) : 预测为真, 实际为真 2. 正确否定 (True Negative,TN) : 预测为假, 实际为假 3. 错误肯定 (False Positive,FP): 预测为真, 实际为假 4. 错误否定 (False Negative,FN): 预测为假, 实际为真

则:

$$\text{查准率 (Precision)} = \text{TP} / (\text{TP} + \text{FP})$$

**理解:** 预测出为阳性的样本中, 正确的有多少。区别准确率 (正确预测出的样本, 包括正确预测为阳性、阴性, 占总样本比例)。例, 在所有我们预测有恶性肿瘤的病人中, 实际上有恶性肿瘤的病人的百分比, 越高越好。

$$\text{查全率 (Recall)} = \text{TP} / (\text{TP} + \text{FN})$$

**理解:** 正确预测为阳性的数量占总样本中阳性数量的比例。例, 在所有实际上有恶性肿瘤的病人中, 成功预测有恶性肿瘤的病人的百分比, 越高越好。

**10.11. ROC 与 AUC.** ROC 全称是“受试者工作特征” (Receiver Operating Characteristic)。

ROC 曲线的面积就是 AUC (Area Under Curve)。

AUC 用于衡量“二分类问题”机器学习算法性能 (泛化能力)。

ROC 曲线, 通过将连续变量设定出多个不同的临界值, 从而计算出一系列真正率和假正率, 再以假正率为横坐标、真正率为纵坐标绘制成曲线, 曲线下面积越大, 推断准确性越高。在 ROC 曲线上, 最靠近坐标图左上方的点为假正率和真正率均较高的临界值。

对于分类器, 或者说分类算法, 评价指标主要有 Precision, Recall, F-score。下图是一个 ROC 曲线的示例。

ROC 曲线的横坐标为 False Positive Rate (FPR)，纵坐标为 True Positive Rate (TPR)。其中

$$TPR = \frac{TP}{TP + FN}, FPR = \frac{FP}{FP + TN}$$

下面着重介绍 ROC 曲线图中的四个点和一条线。第一个点 (0,1)，即  $FPR=0, TPR=1$ ，这意味着  $FN$  (False Negative) =0，并且  $FP$  (False Positive) =0。意味着这是一个完美的分类器，它将所有的样本都正确分类。第二个点 (1,0)，即  $FPR=1, TPR=0$ ，意味着这是一个最糟糕的分类器，因为它成功避开了所有的正确答案。第三个点 (0,0)，即  $FPR=TPR=0$ ，即  $FP$  (False Positive) = $TP$  (True Positive) =0，可以发现该分类器预测所有的样本都为负样本 (Negative)。第四个点 (1,1)，即  $FPR=TPR=1$ ，分类器实际上预测所有的样本都为正样本。经过以上分析，ROC 曲线越接近左上角，该分类器的性能越好。

ROC 曲线所覆盖的面积称为 AUC (Area Under Curve)，可以更直观的判断学习器的性能，AUC 越大则性能越好。

**10.12. 如何画 ROC 曲线.** 下图是一个示例，图中共有 20 个测试样本，“Class”一栏表示每个测试样本真正的标签 (p 表示正样本，n 表示负样本)，“Score”表示每个测试样本属于正样本的概率。

步骤：1、假设已经得出一系列样本被划分为正类的概率，按照大小排序。2、从高到低，依次将“Score”值作为阈值 threshold，当测试样本属于正样本的概率大于或等于这个 threshold 时，我们认为它为正样本，否则为负样本。举例来说，对于图中的第 4 个样本，其“Score”值为 0.6，那么样本 1, 2, 3, 4 都被认为是正样本，因为它们的“Score”值都大于等于 0.6，而其他样本则都认为是负样本。3、每次选取一个不同的 threshold，得到一组 FPR 和 TPR，即 ROC 曲线上的一点。以此共得到 20 组 FPR 和 TPR 的值。4、根据 3、中的每个坐标点，画图。

**10.13. 如何计算 TPR, FPR.** 1、分析数据  $y\_true = [0, 0, 1, 1]$ ;  $scores = [0.1, 0.4, 0.35, 0.8]$ ; 2、列表

样本	预测属于 P 的概率 (score)	真实类别
$y[0]$	0.1	N
$y[1]$	0.4	N
$y[2]$	0.35	P
$y[3]$	0.8	P

3、将截断点依次取为 score 值，计算 TPR 和 FPR。当截断点为 0.1 时：说明只要  $score >= 0.1$ ，它的预测类别就是正例。因为 4 个样本的 score 都大于等于 0.1，所以，所有

样本的预测类别都为 P。 scores = [0.1, 0.4, 0.35, 0.8]; y\_true = [0, 0, 1, 1]; y\_pred = [1, 1, 1, 1]; 正例与反例信息如下：

	正例	反例
正例	TP=2	FN=0
反例	FP=2	TN=0

由此可得：  $TPR = TP/(TP+FN) = 1$ ;  $FPR = FP/(TN+FP) = 1$ ;

当截断点为 0.35 时： scores = [0.1, 0.4, 0.35, 0.8]; y\_true = [0, 0, 1, 1]; y\_pred = [0, 1, 1, 1]; 正例与反例信息如下：

	正例	反例
正例	TP=2	FN=0
反例	FP=1	TN=1

由此可得：  $TPR = TP/(TP+FN) = 1$ ;  $FPR = FP/(TN+FP) = 0.5$ ;

当截断点为 0.4 时： scores = [0.1, 0.4, 0.35, 0.8]; y\_true = [0, 0, 1, 1]; y\_pred = [0, 1, 0, 1]; 正例与反例信息如下：

	正例	反例
正例	TP=1	FN=1
反例	FP=1	TN=1

由此可得：  $TPR = TP/(TP+FN) = 0.5$ ;  $FPR = FP/(TN+FP) = 0.5$ ;

当截断点为 0.8 时： scores = [0.1, 0.4, 0.35, 0.8]; y\_true = [0, 0, 1, 1]; y\_pred = [0, 0, 0, 1]; 正例与反例信息如下：

	正例	反例
正例	TP=1	FN=1
反例	FP=0	TN=2

由此可得：  $TPR = TP/(TP+FN) = 0.5$ ;  $FPR = FP/(TN+FP) = 0$ ;

4、根据 TPR、FPR 值，以 FPR 为横轴，TPR 为纵轴画图。

#### 10.14. 如何计算 AUC.

- 将坐标点按照横坐标 FPR 排序。
- 计算第  $i$  个坐标点和第  $i + 1$  个坐标点的间距  $dx$  。

- 获取第  $i$  或者  $i + 1$  个坐标点的纵坐标  $y$ 。
- 计算面积微元  $ds = ydx$ 。
- 对面积微元进行累加，得到 AUC。

**10.15. 为什么使用 Roc 和 Auc 评价分类器.** 模型有很多评估方法，为什么还要使用 ROC 和 AUC 呢？因为 ROC 曲线有个很好的特性：当测试集中的正负样本的分布变换的时候，ROC 曲线能够保持不变。在实际的数据集中经常会出现样本类不平衡，即正负样本比例差距较大，而且测试数据中的正负样本也可能随着时间变化。

**10.16. 直观理解 AUC.** 下图展现了三种 AUC 的值：

AUC 是衡量二分类模型优劣的一种评价指标，表示正例排在负例前面的概率。其他评价指标有精确度、准确率、召回率，而 AUC 比这三者更为常用。一般在分类模型中，预测结果都是以概率的形式表现，如果要计算准确率，通常都会手动设置一个阈值来将对应的概率转化成类别，这个阈值也就很大程度上影响了模型准确率的计算。举例：现在假设有一个训练好的二分类器对 10 个正负样本（正例 5 个，负例 5 个）预测，得分按高到低排序得到的最好预测结果为  $[1, 1, 1, 1, 1, 0, 0, 0, 0, 0]$ ，即 5 个正例均排在 5 个负例前面，正例排在负例前面的概率为 100%。然后绘制其 ROC 曲线，由于是 10 个样本，除去原点我们需要描 10 个点，如下：

描点方式按照样本预测结果的得分高低从左至右开始遍历。从原点开始，每遇到 1 便向  $y$  轴正方向移动  $y$  轴最小步长 1 个单位，这里是  $1/5=0.2$ ；每遇到 0 则向  $x$  轴正方向移动  $x$  轴最小步长 1 个单位，这里也是 0.2。不难看出，上图的 AUC 等于 1，印证了正例排在负例前面的概率的确为 100%。

假设预测结果序列为  $[1, 1, 1, 1, 0, 1, 0, 0, 0, 0]$ 。

计算上图的 AUC 为 0.96 与计算正例与排在负例前面的概率  $0.8 \times 1 + 0.2 \times 0.8 = 0.96$  相等，而左上角阴影部分的面积则是负例排在正例前面的概率  $0.2 \times 0.2 = 0.04$ 。

假设预测结果序列为  $[1, 1, 1, 0, 1, 0, 1, 0, 0, 0]$ 。

计算上图的 AUC 为 0.88 与计算正例与排在负例前面的概率  $0.6 \times 1 + 0.2 \times 0.8 + 0.2 \times 0.6 = 0.88$  相等，左上角阴影部分的面积是负例排在正例前面的概率  $0.2 \times 0.2 \times 3 = 0.12$ 。

**10.17. 代价敏感错误率与代价曲线.** 不同的错误会产生不同代价。以二分法为例，设置代价矩阵如下：

当判断正确的时候，值为 0，不正确的時候，分别为  $Cost_{01}$  和  $Cost_{10}$ 。

$Cost_{10}$ : 表示实际为反例但预测成正例的代价。

$Cost_{01}$ : 表示实际为正例但是预测为反例的代价。

**代价敏感错误率** = 样本中由模型得到的错误值与代价乘积之和 / 总样本。其数学表达式为：

$$E(f; D; cost) = \frac{1}{m} \left( \sum_{x_i \in D^+} (f(x_i) \neq y_i) \times Cost_{01} + \sum_{x_i \in D^-} (f(x_i) \neq y_i) \times Cost_{10} \right)$$

$D^+$   $D^-$  分别代表样例集的正例子集和反例子集， $x$  是预测值， $y$  是真实值。

**代价曲线**：在均等代价时，ROC 曲线不能直接反应出模型的期望总体代价，而代价曲线可以。代价曲线横轴为 [0,1] 的正例函数代价：

$$P(+)Cost = \frac{p * Cost_{01}}{p * Cost_{01} + (1 - p) * Cost_{10}}$$

其中  $p$  是样本为正例的概率。

代价曲线纵轴维 [0,1] 的归一化代价：

$$Cost_{norm} = \frac{FNR * p * Cost_{01} + FNR * (1 - p) * Cost_{10}}{p * Cost_{01} + (1 - p) * Cost_{10}}$$

其中 FPR 为假阳率， $FNR=1-TPR$  为假阴率。

注：ROC 每个点，对应代价平面上一条线。

例如，ROC 上 (TPR,FPR)，计算出  $FNR=1-TPR$ ，在代价平面上绘制一条从 (0,FPR) 到 (1,FNR) 的线段，面积则为该条件下期望的总体代价。所有线段下界面积，所有条件下学习器的期望总体代价。

**10.18. 模型有哪些比较检验方法.** 正确性分析：模型稳定性分析，稳健性分析，收敛性分析，变化趋势分析，极值分析等。有效性分析：误差分析，参数敏感性分析，模型对比检验等。有用性分析：关键数据求解，极值点，拐点，变化趋势分析，用数据验证动态模拟等。高效性分析：时空复杂度分析与现有进行比较等。

**10.19. 为什么使用标准差.** 方差公式为： $S_N^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2$

标准差公式为： $S_N = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2}$

样本标准差公式为： $S_N = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2}$

与方差相比，使用标准差来表示数据点的离散程度有 3 个好处：1、表示离散程度的数字与样本数据点的数量级一致，更适合对数据样本形成感性认知。

2、表示离散程度的数字单位与样本数据的单位一致，更方便做后续的分析运算。

3、在样本数据大致符合正态分布的情况下，标准差具有方便估算的特性：68% 的数据点落在平均值前后 1 个标准差的范围内、95% 的数据点落在平均值前后 2 个标准差的范围内，而 99% 的数据点将会落在平均值前后 3 个标准差的范围内。

**10.20. 类别不平衡产生原因.** 类别不平衡 (class-imbalance) 是指分类任务中不同类别的训练样例数目差别很大的情况。

产生原因：

分类学习算法通常都会假设不同类别的训练样例数目基本相同。如果不同类别的训练样例数目差别很大，则会影响学习结果，测试结果变差。例如二分类问题中有 998 个反例，正例有 2 个，那学习方法只需返回一个永远将新样本预测为反例的分类器，就能达到 99.8% 的精度；然而这样的分类器没有价值。### 2.16.21 常见的类别不平衡问题解决方法 防止类别不平衡对学习造成的影响，在构建分类模型之前，需要对分类不平衡性问题进行处理。主要解决方法有：

#### 1、扩大数据集

增加包含小类样本数据的数据，更多的数据能得到更多的分布信息。

#### 2、对大类数据欠采样

减少大类数据样本个数，使与小样本个数接近。缺点：欠采样操作时若随机丢弃大类样本，可能会丢失重要信息。代表算法：EasyEnsemble。其思想是利用集成学习机制，将大类划分为若干个集合供不同的学习器使用。相当于对每个学习器都进行欠采样，但对于全局则不会丢失重要信息。

#### 3、对小类数据过采样

过采样：对小类的数据样本进行采样来增加小类的数据样本个数。

代表算法：SMOTE 和 ADASYN。

SMOTE：通过对训练集中的小类数据进行插值来产生额外的小类样本数据。

新的少数类样本产生的策略：对每个少数类样本  $a$ ，在  $a$  的最近邻中随机选一个样本  $b$ ，然后在  $a$ 、 $b$  之间的连线上随机选一点作为新合成的少数类样本。

ADASYN：根据学习难度的不同，对不同的少数类别的样本使用加权分布，对于难以学习的少数类的样本，产生更多的综合数据。通过减少类不平衡引入的偏差和将分类决策边界自适应地转移到困难的样本两种手段，改善了数据分布。

#### 4、使用新评价指标

如果当前评价指标不适用，则应寻找其他具有说服力的评价指标。比如准确度这个评价指标在类别不均衡的分类任务中并不适用，甚至进行误导。因此在类别不均衡分类任务中，需要使用更有说服力的评价指标来对分类器进行评价。

#### 5、选择新算法

不同的算法适用于不同的任务与数据，应该使用不同的算法进行比较。

#### 6、数据代价加权

例如当分类任务是识别小类，那么可以对分类器的小类样本数据增加权值，降低大类样本的权值，从而使得分类器将重点集中在小类样本身上。

## 7、转化问题思考角度

例如在分类问题时，把小类的样本作为异常点，将问题转化为异常点检测或变化趋势检测问题。异常点检测即是对那些罕见事件进行识别。变化趋势检测区别于异常点检测在于其通过检测不寻常的变化趋势来识别。

## 8、将问题细化分析

对问题进行分析与挖掘，将问题划分成多个更小的问题，看这些小问题是否更容易解决。

# 11. 决策树

**11.1. 决策树的基本原理.** 决策树 (Decision Tree) 是一种分而治之的决策过程。一个困难的预测问题，通过树的分支节点，被划分成两个或多个较为简单的子集，从结构上划分为不同的子问题。将依规则分割数据集的过程不断递归下去 (Recursive Partitioning)。随着树的深度不断增加，分支节点的子集越来越小，所需要提的问题数也逐渐简化。当分支节点的深度或者问题的简单程度满足一定的停止规则 (Stopping Rule) 时，该分支节点会停止分裂，此为自上而下的停止阈值 (Cutoff Threshold) 法；有些决策树也使用自下而上的剪枝 (Pruning) 法。

**11.2. 决策树的三要素？** 一棵决策树的生成过程主要分为下 3 个部分：

1、特征选择：从训练数据中众多的特征中选择一个特征作为当前节点的分裂标准，如何选择特征有着很多不同量化评估标准，从而衍生出不同的决策树算法。

2、决策树生成：根据选择的特征评估标准，从上至下递归地生成子节点，直到数据集不可分则决策树停止生长。树结构来说，递归结构是最容易理解的方式。

3、剪枝：决策树容易过拟合，一般来需要剪枝，缩小树结构规模、缓解过拟合。剪枝技术有预剪枝和后剪枝两种。

## 11.3. 决策树学习基本算法.

**11.4. 决策树算法优缺点. 决策树算法的优点：**

- 1、决策树算法易理解，机理解释起来简单。
- 2、决策树算法可以用于小数据集。
- 3、决策树算法的时间复杂度较小，为用于训练决策树的数据点的对数。
- 4、相比于其他算法智能分析一种类型变量，决策树算法可处理数字和数据的类别。
- 5、能够处理多输出的问题。
- 6、对缺失值不敏感。
- 7、可以处理不相关特征数据。
- 8、效率高，决策树只需要一次构建，反复使用，每一次预测的最大计算次数不超过决策树的深度。

**决策树算法的缺点：**

- 1、对连续性的字段比较难预测。
- 2、容易出现过拟合。
- 3、当类别太多时，错误可能就会增加的比较快。
- 4、在处理特征关联性比较强的数据时表现得不太好。
- 5、对于各类别样本数量不一致的数据，在决策树当中，信息增益的结果偏向于那些具有更多数值的特征。

#### 11.5. 熵的概念以及理解.

熵：度量随机变量的不确定性。  
定义：假设随机变量  $X$  的可能取值有  $x_1, x_2, \dots, x_n$ ，对于每一个可能的取值  $x_i$ ，其概率为  $P(X = x_i) = p_i, i = 1, 2, \dots, n$ 。随机变量的熵为：

$$H(X) = - \sum_{i=1}^n p_i \log_2 p_i$$

对于样本集合，假设样本有  $k$  个类别，每个类别的概率为  $\frac{|C_k|}{|D|}$ ，其中  $|C_k||D|$  为类别为  $k$  的样本个数， $|D|$  为样本总数。样本集合  $D$  的熵为：

$$H(D) = - \sum_{k=1}^k \frac{|C_k|}{|D|} \log_2 \frac{|C_k|}{|D|}$$

**11.6. 信息增益的理解.** 定义：以某特征划分数据集前后的熵的差值。熵可以表示样本集合的不确定性，熵越大，样本的不确定性就越大。因此可以使用划分前后集合熵的差值来衡量使用当前特征对于样本集合  $D$  划分效果的好坏。假设划分前样本集合  $D$  的熵为  $H(D)$ 。使用某个特征  $A$  划分数据集  $D$ ，计算划分后的数据子集的熵为  $H(D|A)$ 。

则信息增益为：

$$g(D, A) = H(D) - H(D|A)$$

注：在决策树构建的过程中我们总是希望集合往最快到达纯度更高的子集合方向发展，因此我们总是选择使得信息增益最大的特征来划分当前数据集  $D$ 。

思想：计算所有特征划分数据集  $D$ ，得到多个特征划分数据集  $D$  的信息增益，从这些信息增益中选择最大的，因而当前结点的划分特征便是使信息增益最大的划分所使用的特征。

另外这里提一下信息增益比相关知识： =

信息增益比本质：在信息增益的基础上乘上一个惩罚参数。特征个数较多时，惩罚参数较小；特征个数较少时，惩罚参数较大。

惩罚参数：数据集  $D$  以特征  $A$  作为随机变量的熵的倒数。

**11.7. 剪枝处理的作用及策略.** 剪枝处理是决策树学习算法用来解决过拟合问题的一种办法。

在决策树算法中，为了尽可能正确分类训练样本，节点划分过程不断重复，有时候会造成决策树分支过多，以至于将训练样本集自身特点当作泛化特点，而导致过拟合。因此可以采用剪枝处理来去掉一些分支来降低过拟合的风险。

剪枝的基本策略有预剪枝（pre-pruning）和后剪枝（post-pruning）。

预剪枝：在决策树生成过程中，在每个节点划分前先估计其划分后的泛化性能，如果不能提升，则停止划分，将当前节点标记为叶结点。

后剪枝：生成决策树以后，再自下而上对非叶结点进行考察，若将此节点标记为叶结点可以带来泛化性能提升，则修改之。

## 12. 支持向量机

**12.1. 什么是支持向量机.** 支持向量：在求解的过程中，会发现只根据部分数据就可以确定分类器，这些数据称为支持向量。

支持向量机（Support Vector Machine, SVM）：其含义是通过支持向量运算的分类器。

在一个二维环境中，其中点 R, S, G 点和其它靠近中间黑线的点可以看作为支持向量，它们可以决定分类器，即黑线的具体参数。

支持向量机是一种二分类模型，它的目的是寻找一个超平面来对样本进行分割，分割的原则是边界最大化，最终转化为一个凸二次规划问题来求解。由简至繁的模型包括：

当训练样本线性可分时，通过硬边界（hard margin）最大化，学习一个线性可分支持向量机；

当训练样本近似线性可分时，通过软边界（soft margin）最大化，学习一个线性支持向量机；

当训练样本线性不可分时，通过核技巧和软边界最大化，学习一个非线性支持向量机；

### 12.2. 支持向量机能解决哪些问题. 线性分类

在训练数据中，每个数据都有  $n$  个的属性和一个二分类类别标志，我们可以认为这些数据在一个  $n$  维空间里。我们的目标是找到一个  $n-1$  维的超平面，这个超平面可以将数据分成两部分，每部分数据都属于同一个类别。

这样的超平面有很多，假如我们要找到一个最佳的超平面。此时，增加一个约束条件：要求这个超平面到每边最近数据点的距离是最大的，成为最大边距超平面。这个分类器即为最大边距分类器。

### 非线性分类

SVM 的一个优势是支持非线性分类。它结合使用拉格朗日乘子法（Lagrange Multiplier）和 KKT（Karush Kuhn Tucker）条件，以及核函数可以生成非线性分类器。

**12.3. 核函数特点及其作用.** 引入核函数目的：把原坐标系里线性不可分的数据用核函数 Kernel 投影到另一个空间，尽量使得数据在新的空间里线性可分。

核函数方法的广泛应用，与其特点是分不开的：

- 1) 核函数的引入避免了“维数灾难”，大大减小了计算量。而输入空间的维数 n 对核函数矩阵无影响。因此，核函数方法可以有效处理高维输入。
- 2) 无需知道非线性变换函数  $\Phi$  的形式和参数。
- 3) 核函数的形式和参数的变化会隐式地改变从输入空间到特征空间的映射，进而对特征空间的性质产生影响，最终改变各种核函数方法的性能。
- 4) 核函数方法可以和不同的算法相结合，形成多种不同的基于核函数技术的方法，且这两部分的设计可以单独进行，并可以为不同的应用选择不同的核函数和算法。

**12.4. SVM 为什么引入对偶问题.** 1，对偶问题将原始问题中的约束转为了对偶问题中的等式约束，对偶问题往往更加容易求解。

2，可以很自然的引用核函数（拉格朗日表达式里面有内积，而核函数也是通过内积进行映射的）。

3，在优化理论中，目标函数  $f(x)$  会有多种形式：如果目标函数和约束条件都为变量  $x$  的线性函数，称该问题为线性规划；如果目标函数为二次函数，约束条件为线性函数，称该最优化问题为二次规划；如果目标函数或者约束条件均为非线性函数，称该最优化问题为非线性规划。每个线性规划问题都有一个与之对应的对偶问题，对偶问题有非常良好的性质，以下列举几个：

- a, 对偶问题的对偶是原问题；
- b, 无论原始问题是否是凸的，对偶问题都是凸优化问题；
- c, 对偶问题可以给出原始问题一个下界；
- d, 当满足一定条件时，原始问题与对偶问题的解是完全等价的。

**12.5. 如何理解 SVM 中的对偶问题.** 在硬边界支持向量机中，问题的求解可以转化为凸二次规划问题。

假设优化目标为

$$(1) \quad \begin{aligned} & \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 \\ & s.t. y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1, i = 1, 2, \dots, m. \end{aligned}$$

**step 1.** 转化问题：

$$(2) \quad \min_{\mathbf{w}, b} \max_{\alpha_i \geq 0} \frac{1}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^m \alpha_i (1 - y_i(\mathbf{w}^T \mathbf{x}_i + b))$$

上式等价于原问题，因为若满足 (1) 中不等式约束，则 (2) 式求 max 时,  $\alpha_i(1 - y_i(\mathbf{w}^T \mathbf{x}_i + b))$  必须取 0，与 (1) 等价；若不满足 (1) 中不等式约束，(2) 中求 max 会得到无穷大。交换 min 和 max 获得其对偶问题：

$$\max_{\alpha_i \geq 0} \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^m \alpha_i (1 - y_i (\mathbf{w}^T \mathbf{x}_i + b))$$

交换之后的对偶问题和原问题并不相等，上式的解小于等于原问题的解。

**step 2.** 现在的问题是如何找到问题 (1) 的最优值的一个最好的下界？

$$(3) \quad \frac{1}{2} \|\mathbf{w}\|^2 < v - y_i (\mathbf{w}^T \mathbf{x}_i + b) \leq 0$$

若方程组 (3) 无解，则  $v$  是问题 (1) 的一个下界。若 (3) 有解，则

$$\forall \alpha > 0, \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^m \alpha_i (1 - y_i (\mathbf{w}^T \mathbf{x}_i + b)) < v$$

由逆否命题得：若

$$\exists \alpha > 0, \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^m \alpha_i (1 - y_i (\mathbf{w}^T \mathbf{x}_i + b)) \geq v$$

则 (3) 无解。

那么  $v$  是问题

(1) 的一个下界。要求得一个好的下界，取最大值即可

$$\max_{\alpha_i \geq 0} \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^m \alpha_i (1 - y_i (\mathbf{w}^T \mathbf{x}_i + b))$$

**step 3.** 令

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^m \alpha_i (1 - y_i (\mathbf{w}^T \mathbf{x}_i + b))$$

$p^*$  为原问题的最小值，对应的  $w, b$  分别为  $w^*, b^*$ ，则对于任意的  $a > 0$ ：

$$p^* = \frac{1}{2} \|\mathbf{w}^*\|^2 \geq L(\mathbf{w}^*, b, \alpha) \geq \min_{\mathbf{w}, b} L(\mathbf{w}, b, \alpha)$$

则  $\min_{\mathbf{w}, b} L(\mathbf{w}, b, \alpha)$  是问题 (1) 的一个下界。

此时，取最大值即可求得好的下界，即

$$\max_{\alpha_i \geq 0} \min_{\mathbf{w}, b} L(\mathbf{w}, b, \alpha)$$

核函数	表达式	备注
Linear Kernel 线性核	$k(x, y) = x^t y + c$	
Polynomial Kernel 多项式核	$k(x, y) = (ax^t y + c)^d$	$d \geq 1$ 为多项式的次数
Exponential Kernel 指数核	$k(x, y) = \exp(-\frac{\ x-y\ }{2\sigma^2})$	$\sigma > 0$
Gaussian Kernel 高斯核	$k(x, y) = \exp(-\frac{\ x-y\ ^2}{2\sigma^2})$	$\sigma$ 为高斯核的带宽, $\sigma > 0$ ,
Laplacian Kernel 拉普拉斯核	$k(x, y) = \exp(-\frac{\ x-y\ }{\sigma})$	$\sigma > 0$
ANOVA Kernel	$k(x, y) = \exp(-\sigma(x^k - y^k)^2)^d$	
Sigmoid Kernel	$k(x, y) = \tanh(ax^t y + c)$	$\tanh$ 为双曲正切函数, $a > 0, c < 0$

## 12.6. 常见的核函数有哪些.

### 12.7. SVM 主要特点. 特点:

- (1) SVM 方法的理论基础是非线性映射, SVM 利用内积核函数代替向高维空间的非线性映射。
- (2) SVM 的目标是对特征空间划分得到最优超平面, SVM 方法核心是最大化分类边界。
- (3) 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。
- (4) SVM 是一种有坚实理论基础的新颖的适用小样本学习方法。它基本上不涉及概率测度及大数定律等, 也简化了通常的分类和回归等问题。
- (5) 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”。
- (6) 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:  
增、删非支持向量样本对模型没有影响;  
支持向量样本集具有一定的鲁棒性;  
有些成功的应用中, SVM 方法对核的选取不敏感

- (7) SVM 学习问题可以表示为凸优化问题，因此可以利用已知的有效算法发现目标函数的全局最小值。而其他分类方法（如基于规则的分类器和人工神经网络）都采用一种基于贪心学习的策略来搜索假设空间，这种方法一般只能获得局部最优解。
- (8) SVM 通过最大化决策边界的边缘来控制模型的能力。尽管如此，用户必须提供其他参数，如使用核函数类型和引入松弛变量等。
- (9) SVM 在小样本训练集上能够得到比其它算法好很多的结果。SVM 优化目标是结构化风险最小，而不是经验风险最小，避免了过拟合问题，通过 margin 的概念，得到对数据分布的结构化描述，减低了对数据规模和数据分布的要求，有优秀的泛化能力。
- (10) 它是一个凸优化问题，因此局部最优解一定是全局最优解的优点。

### 12.8. SVM 主要缺点.

- (1) SVM 算法对大规模训练样本难以实施

SVM 的空间消耗主要是存储训练样本和核矩阵，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。

如果数据量很大，SVM 的训练时间就会比较长，如垃圾邮件的分类检测，没有使用 SVM 分类器，而是使用简单的朴素贝叶斯分类器，或者是使用逻辑回归模型分类。

- (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：与粗糙集理论结合，形成一种优势互补的多类问题的组合分类器。

- (3) 对缺失数据敏感，对参数和核函数的选择敏感

支持向量机性能的优劣主要取决于核函数的选取，所以对于一个实际问题而言，如何根据实际的数据模型选择合适的核函数从而构造 SVM 算法。目前比较成熟的核函数及其参数的选择都是人为的，根据经验来选取的，带有一定的随意性。在不同的问题领域，核函数应当具有不同的形式和参数，所以在选取时候应该将领域知识引入进来，但是目前还没有好的方法来解决核函数的选取问题。

### 12.9. 逻辑回归与 SVM 的异同. 相同点：

- LR 和 SVM 都是分类算法。
- LR 和 SVM 都是监督学习算法。

- LR 和 SVM 都是判别模型。
- 如果不考虑核函数，LR 和 SVM 都是线性分类算法，也就是说他们的分类决策面都是线性的。说明：LR 也是可以用核函数的。但 LR 通常不采用核函数的方法。（计算量太大）

不同点：

1、LR 采用 log 损失，SVM 采用合页 (hinge) 损失。逻辑回归的损失函数：

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m [y^i \log h_\theta(x^i) + (1 - y^i) \log(1 - h_\theta(x^i))]$$

支持向量机的目标函数：

$$L(w, n, a) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^n \alpha_i (y_i(w^T x_i + b) - 1)$$

逻辑回归方法基于概率理论，假设样本为 1 的概率可以用 sigmoid 函数来表示，然后通过极大似然估计的方法估计出参数的值。

支持向量机基于几何边界最大化原理，认为存在最大几何边界的分类面为最优分类面。

2、LR 对异常值敏感，SVM 对异常值不敏感。

支持向量机只考虑局部的边界线附近的点，而逻辑回归考虑全局。LR 模型找到的那个超平面，是尽量让所有点都远离他，而 SVM 寻找的那个超平面，是只让最靠近中间分割线的那些点尽量远离，即只用到那些支持向量的样本。

支持向量机改变非支持向量样本并不会引起决策面的变化。

逻辑回归中改变任何样本都会引起决策面的变化。

3、计算复杂度不同。对于海量数据，SVM 的效率较低，LR 效率比较高

当样本较少，特征维数较低时，SVM 和 LR 的运行时间均比较短，SVM 较短一些。准确率的话，LR 明显比 SVM 要高。当样本稍微增加些时，SVM 运行时间开始增长，但是准确率赶超了 LR。SVM 时间虽长，但在可接受范围内。当数据量增长到 20000 时，特征维数增长到 200 时，SVM 的运行时间剧烈增加，远远超过了 LR 的运行时间。但是准确率却和 LR 相差无几。(这其中主要原因是大量非支持向量参与计算，造成 SVM 的二次规划问题)

4、对非线性问题的处理方式不同

LR 主要靠特征构造，必须组合交叉特征，特征离散化。SVM 也可以这样，还可以通过核函数 kernel (因为只有支持向量参与核计算，计算复杂度不高)。由于可以利用核函数，SVM 则可以通过对偶求解高效处理。LR 则在特征空间维度很高时，表现较差。

5、SVM 的损失函数就自带正则。

损失函数中的  $1/2\|w\|^2$  项，这就是为什么 SVM 是结构风险最小化算法的原因!!! 而 LR 必须另外在损失函数上添加正则项!!! \*\*

6、SVM 自带结构风险最小化，LR 则是经验风险最小化。

7、SVM 会用核函数而 LR 一般不用核函数。

### 13. 贝叶斯分类器

#### 13.1. 图解极大似然估计.

极大似然估计的原理，用一张图片来说明，如下图所示：  
例：有两个外形完全相同的箱子，1号箱有99只白球，1只黑球；2号箱有1只白球，99只黑球。在一次实验中，取出的是黑球，请问是从哪个箱子中取出的？

一般的根据经验想法，会猜测这只黑球最像是从2号箱取出，此时描述的“最像”就有“最大似然”的意思，这种想法常称为“最大似然原理”。

**13.2. 极大似然估计原理.** 总结起来，最大似然估计的目的就是：利用已知的样本结果，反推最有可能（最大概率）导致这样结果的参数值。

极大似然估计是建立在极大似然原理的基础上的一个统计方法。极大似然估计提供了一种给定观察数据来评估模型参数的方法，即：“模型已定，参数未知”。通过若干次试验，观察其结果，利用试验结果得到某个参数值能够使样本出现的概率为最大，则称为极大似然估计。

由于样本集中的样本都是独立同分布，可以只考虑一类样本集  $D$ ，来估计参数向量  $\vec{\theta}$ 。记已知的样本集为：

$$D = \vec{x}_1, \vec{x}_2, \dots, \vec{x}_n$$

似然函数 (likelihood function)：联合概率密度函数  $p(D|\vec{\theta})$  称为相对于  $\vec{x}_1, \vec{x}_2, \dots, \vec{x}_n$  的  $\vec{\theta}$  的似然函数。

$$l(\vec{\theta}) = p(D|\vec{\theta}) = p(\vec{x}_1, \vec{x}_2, \dots, \vec{x}_n | \vec{\theta}) = \prod_{i=1}^n p(\vec{x}_i | \vec{\theta})$$

如果  $\hat{\vec{\theta}}$  是参数空间中能使似然函数  $l(\vec{\theta})$  最大的  $\vec{\theta}$  值，则  $\hat{\vec{\theta}}$  应该是“最可能”的参数值，那么  $\hat{\vec{\theta}}$  就是  $\vec{\theta}$  的极大似然估计量。它是样本集的函数，记作：

$$\hat{\vec{\theta}} = d(D) = \arg \max_{\vec{\theta}} l(\vec{\theta})$$

$\hat{\vec{\theta}}(\vec{x}_1, \vec{x}_2, \dots, \vec{x}_n)$  称为极大似然函数估计值。

**13.3. 贝叶斯分类器基本原理.** 贝叶斯决策论通过相关概率已知的情况下利用误判损失来选择最优的类别分类。

假设有  $N$  种可能的分类标记，记为  $Y = c_1, c_2, \dots, c_N$ ，那对于样本  $x$ ，它属于哪一类呢？

计算步骤如下：

step 1. 算出样本  $x$  属于第  $i$  个类的概率，即  $P(c_i|x)$ ；

step 2. 通过比较所有的  $P(c_i|x)$ ，得到样本  $x$  所属的最佳类别。

step 3. 将类别  $c_i$  和样本  $\mathbf{x}$  代入到贝叶斯公式中，得到：

$$P(c_i|\mathbf{x}) = \frac{P(\mathbf{x}|c_i)P(c_i)}{P(\mathbf{x})}.$$

一般来说， $P(c_i)$  为先验概率， $P(\mathbf{x}|c_i)$  为条件概率， $P(\mathbf{x})$  是用于归一化的证据因子。对于  $P(c_i)$  可以通过训练样本中类别为  $c_i$  的样本所占的比例进行估计；此外，由于只需要找出最大的  $P(\mathbf{x}|c_i)$ ，因此我们并不需要计算  $P(\mathbf{x})$ 。

为了求解条件概率，基于不同假设提出了不同的方法，以下将介绍朴素贝叶斯分类器和半朴素贝叶斯分类器。

**13.4. 朴素贝叶斯分类器.** 假设样本  $\mathbf{x}$  包含  $d$  个属性，即  $\mathbf{x} = x_1, x_2, \dots, x_d$ 。于是有：

$$P(\mathbf{x}|c_i) = P(x_1, x_2, \dots, x_d|c_i)$$

这个联合概率难以从有限的训练样本中直接估计得到。于是，朴素贝叶斯（Naive Bayesian，简称 NB）采用了“属性条件独立性假设”：对已知类别，假设所有属性相互独立。于是有：

$$P(x_1, x_2, \dots, x_d|c_i) = \prod_{j=1}^d P(x_j|c_i)$$

这样的话，我们就可以很容易地推出相应的判定准则了：

$$h_{nb}(\mathbf{x}) = \arg \max_{c_i \in Y} P(c_i) \prod_{j=1}^d P(x_j|c_i)$$

#### 条件概率 $P(x_j|c_i)$ 的求解

如果  $x_j$  是标签属性，那么我们可以通过计数的方法估计  $P(x_j|c_i)$

$$P(x_j|c_i) = \frac{P(x_j, c_i)}{P(c_i)} \approx \frac{\#(x_j, c_i)}{\#(c_i)}$$

其中， $\#(x_j, c_i)$  表示在训练样本中  $x_j$  与  $c_i$  共同出现的次数。

如果  $x_j$  是数值属性，通常我们假设类别中  $c_i$  的所有样本第  $j$  个属性的值服从正态分布。我们首先估计这个分布的均值 和方差，然后计算  $x_j$  在这个分布中的概率密度  $P(x_j|c_i)$ 。

**13.5. 举例理解朴素贝叶斯分类器.** 使用经典的西瓜训练集如下：

编号	色泽	根蒂	敲声	纹理	脐部	触感	密度	含糖率	好瓜
1	青绿	蜷缩	浊响	清晰	凹陷	硬滑	0.697	0.460	是
2	乌黑	蜷缩	沉闷	清晰	凹陷	硬滑	0.774	0.376	是
3	乌黑	蜷缩	浊响	清晰	凹陷	硬滑	0.634	0.264	是

4	青绿	蜷缩	沉闷	清晰	凹陷	硬滑	0.608	0.318	是
5	浅白	蜷缩	浊响	清晰	凹陷	硬滑	0.556	0.215	是
6	青绿	稍蜷	浊响	清晰	稍凹	软粘	0.403	0.237	是
7	乌黑	稍蜷	浊响	稍糊	稍凹	软粘	0.481	0.149	是
8	乌黑	稍蜷	浊响	清晰	稍凹	硬滑	0.437	0.211	是
9	乌黑	稍蜷	沉闷	稍糊	稍凹	硬滑	0.666	0.091	否
10	青绿	硬挺	清脆	清晰	平坦	软粘	0.243	0.267	否
11	浅白	硬挺	清脆	模糊	平坦	硬滑	0.245	0.057	否
12	浅白	蜷缩	浊响	模糊	平坦	软粘	0.343	0.099	否
13	青绿	稍蜷	浊响	稍糊	凹陷	硬滑	0.639	0.161	否
14	浅白	稍蜷	沉闷	稍糊	凹陷	硬滑	0.657	0.198	否
15	乌黑	稍蜷	浊响	清晰	稍凹	软粘	0.360	0.370	否
16	浅白	蜷缩	浊响	模糊	平坦	硬滑	0.593	0.042	否
17	青绿	蜷缩	沉闷	稍糊	稍凹	硬滑	0.719	0.103	否

对下面的测试例“测 1”进行分类：

编号	色泽	根蒂	敲声	纹理	脐部	触感	密度	含糖率	好瓜
测 1	青绿	蜷缩	浊响	清晰	凹陷	硬滑	0.697	0.460	?

首先，估计类先验概率  $P(c_j)$ ，有

$$P(=) = \frac{8}{17} = 0.471 \quad P(=) = \frac{9}{17} = 0.529$$

然后，为每个属性估计条件概率（这里，对于连续属性，假定它们服从正态分布）

$$P_1 = P = | = = \frac{3}{8} = 0.375$$

$$P_2 = P = | = = \frac{3}{9} \approx 0.333$$

$$P_3 = P = | = = \frac{5}{8} = 0.625$$

$$P_4 = P = | = = \frac{3}{9} = 0.333$$

$$P_5 = P = | = = \frac{6}{8} = 0.750$$

$$P_{|} = P = | = = \frac{4}{9} \approx 0.444$$

$$P_{|} = P = | = = \frac{7}{8} = 0.875$$

$$P_{|} = P = | = = \frac{2}{9} \approx 0.222$$

$$P_{|} = P = | = = \frac{6}{8} = 0.750$$

$$P_{|} = P = | = = \frac{2}{9} \approx 0.222$$

$$P_{|} = P = | = = \frac{6}{8} = 0.750$$

$$P_{|} = P = | = = \frac{6}{9} \approx 0.667$$

$$\begin{aligned} \rho_{0.697|} &= \rho = 0.697| = \\ &= \frac{1}{\sqrt{2\pi} \times 0.129} \exp\left(-\frac{(0.697 - 0.574)^2}{2 \times 0.129^2}\right) \approx 1.959 \end{aligned}$$

$$\begin{aligned} \rho_{0.697|} &= \rho = 0.697| = \\ &= \frac{1}{\sqrt{2\pi} \times 0.195} \exp\left(-\frac{(0.697 - 0.496)^2}{2 \times 0.195^2}\right) \approx 1.203 \end{aligned}$$

$$\begin{aligned} \rho_{0.460|} &= \rho = 0.460| = \\ &= \frac{1}{\sqrt{2\pi} \times 0.101} \exp\left(-\frac{(0.460 - 0.279)^2}{2 \times 0.101^2}\right) \approx 0.788 \end{aligned}$$

$$\begin{aligned} \rho_{0.460|} &= \rho = 0.460| = \\ &= \frac{1}{\sqrt{2\pi} \times 0.108} \exp\left(-\frac{(0.460 - 0.154)^2}{2 \times 0.108^2}\right) \approx 0.066 \end{aligned}$$

于是有

$$P(=) \times P_{|} \times P_{|} \times P_{|} \times P_{|} \times P_{|} \times P_{|} \times p_{0.697|} \times p_{0.460|} \approx 0.063 P(=) \times P_{|} \times P_{|} \times P_{|} \times P_{|} \times P_{|} \times P_{|} \times p_{0.697|} \times p_{0.460|} \approx 0.063 \times 0.066 \approx 0.0041$$

由于  $0.063 > 6.80 \times 10^{-5}$ , 因此, 朴素贝叶斯分类器将测试样本“测 1”判别为“好瓜”。

**13.6. 半朴素贝叶斯分类器.** 朴素贝叶斯采用了“属性条件独立性假设”，半朴素贝叶斯分类器的基本想法是适当考虑一部分属性间的相互依赖信息。**独依赖估计** (One-Dependence Estimator, 简称 ODE) 是半朴素贝叶斯分类器最常用的一种策略。顾名思义，独依赖是假设每个属性在类别之外最多依赖一个其他属性，即：

$$P(\mathbf{x}|c_i) = \prod_{j=1}^d P(x_j|c_i, \text{pa}_j)$$

其中  $\text{pa}_j$  为属性  $x_i$  所依赖的属性，成为  $x_i$  的父属性。假设父属性  $\text{pa}_j$  已知，那么可以使用下面的公式估计  $P(x_j|c_i, \text{pa}_j)$

$$P(x_j|c_i, \text{pa}_j) = \frac{P(x_j, c_i, \text{pa}_j)}{P(c_i, \text{pa}_j)}$$

## 14. EM 算法

**14.1. EM 算法基本思想.** 最大期望算法 (Expectation-Maximization algorithm, EM)，是一类通过迭代进行极大似然估计的优化算法，通常作为牛顿迭代法的替代，用于对包含隐变量或缺失数据的概率模型进行参数估计。

最大期望算法基本思想是经过两个步骤交替进行计算：

第一步是计算期望 (E)，利用对隐藏变量的现有估计值，计算其最大似然估计值；

第二步是最大化 (M)，最大化在 E 步上求得的最大似然值来计算参数的值。

M 步上找到的参数估计值被用于下一个 E 步计算中，这个过程不断交替进行。

**14.2. EM 算法推导.** 对于  $m$  个样本观察数据  $\mathbf{x} = (x^1, x^2, \dots, x^m)$ ，现在想找出样本的模型参数  $\theta$ ，其极大化模型分布的对数似然函数为：

$$\theta = \arg \max_{\theta} \sum_{i=1}^m \log P(x^{(i)}; \theta)$$

如果得到的观察数据有未观察到的隐含数据  $\mathbf{z} = (z^{(1)}, z^{(2)}, \dots, z^{(m)})$ ，极大化模型分布的对数似然函数则为：

$$(a) \quad \theta = \arg \max_{\theta} \sum_{i=1}^m \log P(x^{(i)}; \theta) = \arg \max_{\theta} \sum_{i=1}^m \log \sum_{z^{(i)}} P(x^{(i)}, z^{(i)}; \theta)$$

由于上式不能直接求出  $\theta$ ，采用缩放技巧：

$$(1) \quad \begin{aligned} \sum_{i=1}^m \log \sum_{z^{(i)}} P(x^{(i)}, z^{(i)}; \theta) &= \sum_{i=1}^m \log \sum_{z^{(i)}} Q_i(z^{(i)}) \frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \\ &\geq \sum_{i=1}^m \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \end{aligned}$$

上式用到了 Jensen 不等式：

$$\log \sum_j \lambda_j y_j \geq \sum_j \lambda_j \log y_j, \quad \lambda_j \geq 0, \sum_j \lambda_j = 1$$

并且引入了一个未知的新分布  $Q_i(z^{(i)})$ 。

此时，如果需要满足 Jensen 不等式中的等号，所以有：

$$\frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} = c, c$$

由于  $Q_i(z^{(i)})$  是一个分布，所以满足

$$\sum_z Q_i(z^{(i)}) = 1$$

综上，可得：

$$Q_i(z^{(i)}) = \frac{P(x^{(i)}, z^{(i)}; \theta)}{\sum_z P(x^{(i)}, z^{(i)}; \theta)} = \frac{P(x^{(i)}, z^{(i)}; \theta)}{P(x^{(i)}; \theta)} = P(z^{(i)} | x^{(i)}; \theta)$$

如果  $Q_i(z^{(i)}) = P(z^{(i)} | x^{(i)}; \theta)$ ，则第 (1) 式是我们的包含隐藏数据的对数似然的一个下界。如果我们能极大化这个下界，则也在尝试极大化我们的对数似然。即我们需要最大化下式：

$$\arg \max_{\theta} \sum_{i=1}^m \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$$

简化得：

$$\arg \max_{\theta} \sum_{i=1}^m \sum_{z^{(i)}} Q_i(z^{(i)}) \log P(x^{(i)}, z^{(i)}; \theta)$$

以上即为 EM 算法的 M 步， $\sum_{z^{(i)}} Q_i(z^{(i)}) \log P(x^{(i)}, z^{(i)}; \theta)$  可理解为  $\log P(x^{(i)}, z^{(i)}; \theta)$  基于条件概率分布  $Q_i(z^{(i)})$  的期望。以上即为 EM 算法中 E 步和 M 步的具体数学含义。

**14.3. 图解 EM 算法.** 考虑上一节中的 (a) 式，表达式中存在隐变量，直接找到参数估计比较困难，通过 EM 算法迭代求解下界的最大值到收敛为止。

图片中的紫色部分是我们的目标模型  $p(x|\theta)$ ，该模型复杂，难以求解析解，为了消除隐变量  $z^{(i)}$  的影响，我们可以选择一个不包含  $z^{(i)}$  的模型  $r(x|\theta)$ ，使其满足条件  $r(x|\theta) \leq p(x|\theta)$ 。

求解步骤如下：

- (1) 选取  $\theta_1$ ，使得  $r(x|\theta_1) = p(x|\theta_1)$ ，然后对此时的  $r$  求取最大值，得到极值点  $\theta_2$ ，实现参数的更新。
- (2) 重复以上过程到收敛为止，在更新过程中始终满足  $r \leq p$ .

**14.4. EM 算法流程.** 输入: 观察数据  $x = (x^{(1)}, x^{(2)}, \dots, x^{(m)})$ , 联合分布  $p(x, z; \theta)$ , 条件分布  $p(z|x; \theta)$ , 最大迭代次数  $J$

1) 随机初始化模型参数  $\theta$  的初值  $\theta^0$ 。

2) *for j from 1 to j:*

a) E 步。计算联合分布的条件概率期望:

$$Q_i(z^{(i)}) = P(z^{(i)}|x^{(i)}, \theta^j)$$

$$L(\theta, \theta^j) = \sum_{i=1}^m \sum_{z^{(i)}} P(z^{(i)}|x^{(i)}, \theta^j) \log P(x^{(i)}, z^{(i)}; \theta)$$

b) M 步。极大化  $L(\theta, \theta^j)$ , 得到  $\theta^{j+1}$ :

$$\theta^{j+1} = \arg \max_{\theta} L(\theta, \theta^j)$$

c) 如果  $\theta^{j+1}$  收敛, 则算法结束。否则继续回到步骤 a) 进行 E 步迭代。

输出: 模型参数  $\theta$ 。

## 15. 降维和聚类

**15.1. 图解为什么会产生维数灾难.** 假如数据集包含 10 张照片, 照片中包含三角形和圆两种形状。现在来设计一个分类器进行训练, 让这个分类器对其他的照片进行正确分类 (假设三角形和圆的总数是无限大), 简单的, 我们用一个特征进行分类:

图 2.21.1.a

从上图可看到, 如果仅仅只有一个特征进行分类, 三角形和圆几乎是均匀分布在这条线段上, 很难将 10 张照片线性分类。那么, 增加一个特征后的情况会怎么样:

图 2.21.1.b

增加一个特征后, 我们发现仍然无法找到一条直线将猫和狗分开。所以, 考虑需要再增加一个特征:

图 2.21.1.c

图 2.21.1.d

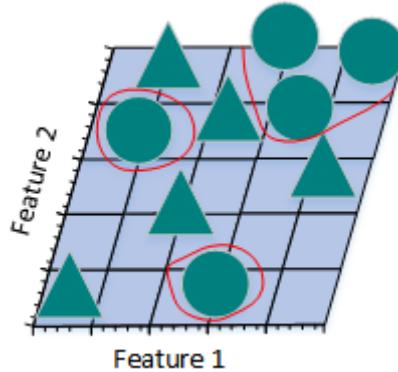
此时, 可以找到一个平面将三角形和圆分开。

现在计算一下不同特征数是样本的密度:

(1) 一个特征时, 假设特征空间时长度为 5 的线段, 则样本密度为  $10 \div 5 = 2$ 。

(2) 两个特征时, 特征空间大小为  $5 \times 5 = 25$ , 样本密度为  $10 \div 25 = 0.4$ 。

(3) 三个特征时, 特征空间大小是  $5 \times 5 \times 5 = 125$ , 样本密度为  $10 \div 125 = 0.08$ 。



以此类推，如果继续增加特征数量，样本密度会越来越稀疏，此时，更容易找到一个超平面将训练样本分开。当特征数量增长至无限大时，样本密度就变得非常稀疏。

下面看一下将高维空间的分类结果映射到低维空间时，会出现什么情况？

图 2.21.1.e

上图是将三维特征空间映射到二维特征空间后的结果。尽管在高维特征空间时训练样本线性可分，但是映射到低维空间后，结果正好相反。事实上，增加特征数量使得高维空间线性可分，相当于在低维空间内训练一个复杂的非线性分类器。不过，这个非线性分类器太过“聪明”，仅仅学到了一些特例。如果将其用来辨别那些未曾出现在训练样本中的测试样本时，通常结果不太理想，会造成过拟合问题。

图 2.21.1.f

上图所示的只采用 2 个特征的线性分类器分错了一些训练样本，准确率似乎没有图 2.21.1.e 的高，但是，采用 2 个特征的线性分类器的泛化能力比采用 3 个特征的线性分类器要强。因为，采用 2 个特征的线性分类器学到的不只是特例，而是一个整体趋势，对于那些未曾出现过的样本也可以比较好地辨别开来。换句话说，通过减少特征数量，可以避免出现过拟合问题，从而避免“维数灾难”。

上图从另一个角度诠释了“维数灾难”。假设只有一个特征时，特征的值域是 0 到 1，每一个三角形和圆的特征值都是唯一的。如果我们希望训练样本覆盖特征值值域的 20%，那么就需要三角形和圆总数的 20%。我们增加一个特征后，为了继续覆盖特征值值域的 20% 就需要三角形和圆总数的  $45\%(0.45^2 \approx 0.2)$ 。继续增加一个特征后，需要三角形和圆总数的  $58\%(0.58^3 \approx 0.2)$ 。随着特征数量的增加，为了覆盖特征值值域的 20%，就需要更多的训练样本。如果没有足够的训练样本，就可能会出现过拟合问题。

通过上述例子，我们可以看到特征数量越多，训练样本就会越稀疏，分类器的参数估计就会越不准确，更加容易出现过拟合问题。“维数灾难”的另一个影响是训练样本的稀疏性并不是均匀分布的。处于中心位置的训练样本比四周的训练样本更加稀疏。

假设有一个二维特征空间，如上图所示的矩形，在矩形内部有一个内切的圆形。由于越接近圆心的样本越稀疏，因此，相比于圆形内的样本，那些位于矩形四角的样本更加难以分类。当维数变大时，特征超空间的容量不变，但单位圆的容量会趋于 0，在高维空间中，大多数训练数据驻留在特征超空间的角落。散落在角落的数据要比处于中心的数据难于分类。

### 15.2. 怎样避免维数灾难. 有待完善!!!

解决维度灾难问题:

主成分分析法 PCA, 线性判别法 LDA

奇异值分解简化数据、拉普拉斯特征映射

Lasso 缩减系数法、小波分析法、

**15.3. 聚类和降维有什么区别与联系.** 聚类用于找寻数据内在的分布结构, 既可以作为一个单独的过程, 比如异常检测等等。也可作为分类等其他学习任务的前驱过程。聚类是标准的无监督学习。

1) 在一些推荐系统中需确定新用户的类型, 但定义“用户类型”却可能不太容易, 此时往往可先对原有的用户数据进行聚类, 根据聚类结果将每个簇定义为一个类, 然后再基于这些类训练分类模型, 用于判别新用户的类型。

2) 而降维则是为了缓解维数灾难的一个重要方法, 就是通过某种数学变换将原始高维属性空间转变为一个低维“子空间”。其基于的假设就是, 虽然人们平时观测到的数据样本虽然是高维的, 但是实际上真正与学习任务相关的是个低维度的分布。从而通过最主要的几个特征维度就可以实现对数据的描述, 对于后续的分类很有帮助。比如对于 Kaggle (数据分析竞赛平台之一) 上的泰坦尼克号生还问题。通过给定一个乘客的许多特征如年龄、姓名、性别、票价等, 来判断其是否能在海难中生还。这就需要首先进行特征筛选, 从而能够找出主要的特征, 让学习到的模型有更好的泛化性。

聚类和降维都可以作为分类等问题的预处理步骤。

但是他们虽然都能实现对数据的约减。但是二者适用的对象不同, 聚类针对的是数据点, 而降维则是对于数据的特征。另外它们有着很多种实现方法。聚类中常用的有 K-means、层次聚类、基于密度的聚类等; 降维中常用的则 PCA、Isomap、LLE 等。

**15.4. 有哪些聚类算法优劣衡量标准.** 不同聚类算法有不同的优劣和不同的适用条件。可以从以下方面进行衡量判断: 1、算法的处理能力: 处理大的数据集的能力, 即算法复杂度; 处理数据噪声的能力; 处理任意形状, 包括有间隙的嵌套的数据的能力; 2、算法是否需要预设条件: 是否需要预先知道聚类个数, 是否需要用户给出领域知识;

3、算法的数据输入属性: 算法处理的结果与数据输入的顺序是否相关, 也就是说算法是否独立于数据输入顺序; 算法处理有很多属性数据的能力, 也就是对数据维数是否敏感, 对数据的类型有无要求。

**15.5. 聚类和分类有什么区别.** 聚类 (Clustering) 聚类, 简单地说就是把相似的东西分到一组, 聚类的时候, 我们并不关心某一类是什么, 我们需要实现的目标只是把相似的东西聚到一起。一个聚类算法通常只需要知道如何计算相似度就可以开始工作了, 因此聚类通常并不需要使用训练数据进行学习, 在机器学习中属于无监督学习。

## 分类 (Classification)

分类，对于一个分类器，通常需要你告诉它“这个东西被分为某某类”。一般情况下，一个分类器会从它得到的训练集中进行学习，从而具备对未知数据进行分类的能力，在机器学习中属于监督学习。

算法名称	可伸缩性	适合的数据类型	高维性	异常数据抗干扰性	聚类形状	算法效率
WAVECLUSTER	很高	数值型	很高	较高	任意形状	很高
ROCK	很高	混合型	很高	很高	任意形状	一般
BIRCH	较高	数值型	较低	较低	球形	很高
CURE	较高	数值型	一般	很高	任意形状	较高
K-PROTOTYPES	一般	混合型	较低	较低	任意形状	一般
DENCLUE	较低	数值型	较高	一般	任意形状	较高
OPTIGRID	一般	数值型	较高	一般	任意形状	一般
CLIQUE	较高	数值型	较高	较高	任意形状	较低
DBSCAN	一般	数值型	较低	较高	任意形状	一般
CLARANS	较低	数值型	较低	较高	球形	较低

### 15.6. 不同聚类算法特点性能比较.

**15.7. 四种常用聚类方法之比较.** 聚类就是按照某个特定标准把一个数据集分割成不同的类或簇，使得同一个簇内的数据对象的相似性尽可能大，同时不在同一个簇中的数据对象的差异性也尽可能地大。即聚类后同一类的数据尽可能聚集到一起，不同类数据尽量分离。主要的聚类算法可以划分为如下几类：划分方法、层次方法、基于密度的方法、基于网格的方法以及基于模型的方法。下面主要对 k-means 聚类算法、凝聚型层次聚类算法、神经网络聚类算法之 SOM，以及模糊聚类的 FCM 算法通过通用测试数据集进行聚类效果的比较和分析。

**15.8. k-means 聚类算法.** k-means 是划分方法中较经典的聚类算法之一。由于该算法的效率高，所以在对大规模数据进行聚类时被广泛应用。目前，许多算法均围绕着该算法进行扩展和改进。k-means 算法以  $k$  为参数，把  $n$  个对象分成  $k$  个簇，使簇内具有较高的相似度，而簇间的相似度较低。k-means 算法的处理过程如下：首先，随机地选择  $k$  个对象，每个对象初始地代表了一个簇的平均值或中心；对剩余的每个对象，根据其与各簇中心的距离，将它赋给最近的簇；然后重新计算每个簇的平均值。这个过程不断重复，直到准则函数收敛。通常，采用平方误差准则，其定义如下：

$$E = \sum_{i=1}^k \sum_{p \in C_i} \|p - m_i\|^2$$

这里  $E$  是数据中所有对象的平方误差的总和,  $p$  是空间中的点,  $m_i$  是簇  $C_i$  的平均值 [9]。该目标函数使生成的簇尽可能紧凑独立, 使用的距离度量是欧几里得距离, 当然也可以用其他距离度量。

**算法流程:** 输入: 包含  $n$  个对象的数据和簇的数目  $k$ ; 输出:  $n$  个对象到  $k$  个簇, 使平方误差准则最小。步骤: (1) 任意选择  $k$  个对象作为初始的簇中心; (2) 根据簇中对象的平均值, 将每个对象(重新)赋予最类似的簇; (3) 更新簇的平均值, 即计算每个簇中对象的平均值; (4) 重复步骤 (2)、(3) 直到簇中心不再变化;

**15.9. 层次聚类算法.** 根据层次分解的顺序是自底向上的还是自上向下的, 层次聚类算法分为凝聚的层次聚类算法和分裂的层次聚类算法。凝聚型层次聚类的策略是先将每个对象作为一个簇, 然后合并这些原子簇为越来越大的簇, 直到所有对象都在一个簇中, 或者某个终结条件被满足。绝大多数层次聚类属于凝聚型层次聚类, 它们只是在簇间相似度的定义上有所不同。

#### 算法流程:

注: 以采用最小距离的凝聚层次聚类算法为例:

(1) 将每个对象看作一类, 计算两两之间的最小距离; (2) 将距离最小的两个类合并成一个新类; (3) 重新计算新类与所有类之间的距离; (4) 重复 (2)、(3), 直到所有类最后合并成一类。

**15.10. SOM 聚类算法.** SOM 神经网络 [11] 是由芬兰神经网络专家 Kohonen 教授提出的, 该算法假设在输入对象中存在一些拓扑结构或顺序, 可以实现从输入空间( $n$  维)到输出平面(2 维)的降维映射, 其映射具有拓扑特征保持性质, 与实际的大脑处理有很强的理论联系。

SOM 网络包含输入层和输出层。输入层对应一个高维的输入向量, 输出层由一系列组织在 2 维网格上的有序节点构成, 输入节点与输出节点通过权重向量连接。学习过程中, 找到与之距离最短的输出层单元, 即获胜单元, 对其更新。同时, 将邻近区域的权值更新, 使输出节点保持输入向量的拓扑特征。

#### 算法流程:

(1) 网络初始化, 对输出层每个节点权重赋初值; (2) 从输入样本中随机选取输入向量并且归一化, 找到与输入向量距离最小的权重向量; (3) 定义获胜单元, 在获胜单元的邻近区域调整权重使其向输入向量靠拢; (4) 提供新样本、进行训练; (5) 收缩邻域半径、减小学习率、重复, 直到小于允许值, 输出聚类结果。

**15.11. FCM 聚类算法.** 1965 年美国加州大学柏克莱分校的扎德教授第一次提出了‘集合’的概念。经过十多年的发展, 模糊集合理论渐渐被应用到各个实际应用方面。为克服非此即彼的分类缺点, 出现了以模糊集合论为数学基础的聚类分析。用模糊数学的方法进行聚类分析, 就是模糊聚类分析 [12]。

FCM 算法是一种以隶属度来确定每个数据点属于某个聚类程度的算法。该聚类算法是传统硬聚类算法的一种改进。

设数据集  $X = x_1, x_2, \dots, x_n$ , 它的模糊  $c$  划分可用模糊矩阵  $U = [u_{ij}]$  表示, 矩阵  $U$  的元素  $u_{ij}$  表示第  $j(j = 1, 2, \dots, n)$  个数据点属于第  $i(i = 1, 2, \dots, c)$  类的隶属度,  $u_{ij}$  满足如下条件:

$$(13) \quad \begin{aligned} \sum_{i=1}^c u_{ij} &= 1 \quad \forall j \\ u_{ij} &\in [0, 1] \quad \forall i, j \\ \sum_{j=1}^c u_{ij} &> 0 \quad \forall i \end{aligned}$$

目前被广泛使用的聚类准则是取类内加权误差平方和的极小值。即:

$$(min) J_m(U, V) = \sum_{j=1}^n \sum_{i=1}^c u_{ij}^m d_{ij}^2(x_j, v_i)$$

其中  $V$  为聚类中心,  $m$  为加权指数,  $d_{ij}(x_j, v_i) = \|v_i - x_j\|$ 。

**算法流程:**

- (1) 标准化数据矩阵;
- (2) 建立模糊相似矩阵, 初始化隶属矩阵;
- (3) 算法开始迭代, 直到目标函数收敛到极小值;
- (4) 根据迭代结果, 由最后的隶属矩阵确定数据所属的类, 显示最后的聚类结果。

**15.12. 四种聚类算法试验.** 选取专门用于测试分类、聚类算法的国际通用的 UCI 数据库中的 IRIS 数据集, IRIS 数据集包含 150 个样本数据, 分别取自三种不同的莺尾属植物 setosa、versicolor 和 virginica 的花朵样本, 每个数据含有 4 个属性, 即萼片长度、萼片宽度、花瓣长度、花瓣宽度, 单位为 cm。在数据集上执行不同的聚类算法, 可以得到不同精度的聚类结果。基于前面描述的各算法原理及流程, 可初步得如下聚类结果。

聚类方法	聚错样本数	运行时间/s	平均准确率/ (%)
K-means	17	0.146001	89
层次聚类	51	0.128744	66
SOM	22	5.267283	86
FCM	12	0.470417	92

**注:**

- (1) 聚错样本数: 总的聚错的样本数, 即各类中聚错的样本数的和;

- (2) 运行时间：即聚类整个过程所耗费的时间，单位为 s；
- (3) 平均准确度：设原数据集有 k 个类，用  $c_i$  表示第 i 类， $n_i$  为  $c_i$  中样本的个数， $m_i$  为聚类正确的个数，则  $m_i/n_i$  为第 i 类中的精度，则平均精度为： $avg = \frac{1}{k} \sum_{i=1}^k \frac{m_i}{n_i}$ 。



## CHAPTER 3

# 深度学习基础

## 1. 基本概念

**1.1. 神经网络组成？** 神经网络类型众多，其中最为重要的是多层感知机。为了详细地描述神经网络，我们先从最简单的神经网络说起。

### 感知机

多层感知机中的特征神经元模型称为感知机，由 *Frank Rosenblatt* 于 1957 年发明。

简单的感知机如下图所示：

其中  $x_1, x_2, x_3$  为感知机的输入，其输出为：

$$output = \begin{cases} 0, & \text{if } \sum_i w_i x_i \leq threshold \\ 1, & \text{if } \sum_i w_i x_i > threshold \end{cases}$$

假如把感知机想象成一个加权投票机制，比如 3 位评委给一个歌手打分，打分分别为 4 分、1 分、-3 分，这 3 位评分的权重分别是 132，则该歌手最终得分为  $4 \times 1 + 1 \times 3 + (-3) \times 2 = 1$ 。按照比赛规则，选取的  $threshold$  为 3，说明只有歌手的综合评分大于 3 时，才可顺利晋级。对照感知机，该选手被淘汰，因为：

$$\sum_i w_i x_i < threshold = 3, output = 0$$

用  $-b$  代替  $threshold$ ，输出变为：

$$output = \begin{cases} 0, & \text{if } \mathbf{w} \cdot \mathbf{x} + b \leq 0 \\ 1, & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \end{cases}$$

设置合适的  $\mathbf{x}$  和  $b$ ，一个简单的感知机单元的与非门表示如下：

当输入为 0, 1 时，感知机输出为  $0 \times (-2) + 1 \times (-2) + 3 = 1$ 。

复杂一些的感知机由简单的感知机单元组合而成：

### 多层感知机

多层感知机由感知机推广而来，最主要的特点是有多个神经元层，因此也叫深度神经网络。相比于单独的感知机，多层感知机的第  $i$  层的每个神经元和第  $i - 1$  层的每个神经元都有连接。

输出层可以不止有 1 个神经元。隐藏层可以只有 1 层，也可以有多层。输出层为多个神经元的神经网络例如下图所示：

### 1.2. 神经网络有哪些常用模型结构？

下图包含了大部分常用的模型：

**1.3. 如何选择深度学习开发平台？** 现有的深度学习开源平台主要有 Caffe, PyTorch, MXNet, CNTK, Theano, TensorFlow, Keras, fastai 等。那如何选择一个适合自己的平台呢，下面列出一些衡量做参考。

#### 参考 1：与现有编程平台、技能整合的难易程度

主要是前期积累的开发经验和资源，比如编程语言，前期数据集存储格式等。

#### 参考 2：与相关机器学习、数据处理生态整合的紧密程度

深度学习研究离不开各种数据处理、可视化、统计推断等软件包。考虑建模之前，是否具有方便的数据预处理工具？建模之后，是否具有方便的工具进行可视化、统计推断、数据分析。

#### 参考 3：对数据量及硬件的要求和支持

深度学习在不同应用场景的数据量是不一样的，这也就导致我们可能需要考虑分布式计算、多 GPU 计算的问题。例如，对计算机图像处理研究的人员往往需要将图像文件和计算任务分部到多台计算机节点上进行执行。当下每个深度学习平台都在快速发展，每个平台对分布式计算等场景的支持也在不断演进。

#### 参考 4：深度学习平台的成熟程度

成熟程度的考量是一个比较主观的考量因素，这些因素可包括：社区的活跃程度；是否容易和开发人员进行交流；当前应用的势头。

#### 参考 5：平台利用是否多样性？

有些平台是专门为深度学习研究和应用进行开发的，有些平台对分布式计算、GPU 等构架都有强大的优化，能否用这些平台/软件做其他事情？比如有些深度学习软件是可以用来求解二次型优化；有些深度学习平台很容易被扩展，被运用在强化学习的应用中。

### 1.4. 为什么使用深层表示？

1. 深度神经网络是一种特征递进式的学习算法，浅层的神经元直接从输入数据中学习一些低层次的简单特征，例如边缘、纹理等。而深层的特征则基于已学习到的浅层特征继续学习更高级的特征，从计算机的角度学习深层的语义信息。
2. 深层的网络隐藏单元数量相对较少，隐藏层数目较多，如果浅层的网络想要达到同样的计算结果则需要指数级增长的单元数量才能达到。

### 1.5. 为什么深层神经网络难以训练？

1. 梯度消失梯度消失是指通过隐藏层从后向前看，梯度会变的越来越小，说明前面层的学习会显著慢于后面层的学习，所以学习会卡住，除非梯度变大。  
梯度消失的原因受到多种因素影响，例如学习率的大小，网络参数的初始化，激活函数的边缘效应等。在深层神经网络中，每一个神经元计算得到的梯度都会传递给前一层，较浅层的神经元接收到的梯度受到之前所有层梯度的影响。如果计算得到的梯度值非常小，随着层数增多，求出的梯度更新信息将会以指数形式衰减，就会发生梯度消失。下图是不同隐含层的学习速率：
2. 梯度爆炸在深度网络或循环神经网络（Recurrent Neural Network, RNN）等网络结构中，梯度可在网络更新的过程中不断累积，变成非常大的梯度，导致网络权重值的大幅更新，使得网络不稳定；在极端情况下，权重值甚至会溢出，变为  $Nan$  值，再也无法更新。
3. 权重矩阵的退化导致模型的有效自由度减少。

参数空间中学习的退化速度减慢，导致减少了模型的有效维数，网络的可用自由度对学习中梯度范数的贡献不均衡，随着相乘矩阵的数量（即网络深度）的增加，矩阵的乘积变得越来越退化。在有硬饱和边界的非线性网络中（例如 ReLU 网络），随着深度增加，退化过程会变得越来越快。Duvenaud 等人 2014 年的论文里展示了关于该退化过程的可视化：

随着深度的增加，输入空间（左上角所示）会在输入空间中的每个点处被扭曲成越来越细的单丝，只有一个与细丝正交的方向影响网络的响应。沿着这个方向，网络实际上对变化变得非常敏感。

**1.6. 深度学习和机器学习有什么不同？** 机器学习：利用计算机、概率论、统计学等知识，输入数据，让计算机学会新知识。机器学习的过程，就是训练数据去优化目标函数。

**深度学习：**是一种特殊的机器学习，具有强大的能力和灵活性。它通过学习将世界表示为嵌套的层次结构，每个表示都与更简单的特征相关，而抽象的表示则用于计算更抽象的表示。

传统的机器学习需要定义一些手工特征，从而有目的的去提取目标信息，非常依赖任务的特异性以及设计特征的专家经验。而深度学习可以从大数据中先学习简单的特征，并从其逐渐学习到更为复杂抽象的深层特征，不依赖人工的特征工程，这也是深度学习在大数据时代受欢迎的一大原因。

## 2. 网络操作与计算

**2.1. 前向传播与反向传播？** 神经网络的计算主要有两种：前向传播（forward propagation, FP）作用于每一层的输入，通过逐层计算得到输出结果；反向传播（backward propagation, BP）作用于网络的输出，通过计算梯度由深到浅更新网络参数。

**前向传播**

假设上一层结点  $i, j, k, \dots$  等一些结点与本层的结点  $w$  有连接，那么结点  $w$  的值怎么算呢？就是通过上一层的  $i, j, k, \dots$  等结点以及对应的连接权值进行加权和运算，最终结果再加上一个偏置项（图中为了简单省略了），最后在通过一个非线性函数（即激活函数），如  $ReLU$ ,  $sigmoid$  等函数，最后得到的结果就是本层结点  $w$  的输出。

最终不断的通过这种方法一层层的运算，得到输出层结果。

### 反向传播

由于我们前向传播最终得到的结果，以分类为例，最终总是有误差的，那么怎么减少误差呢，当前应用广泛的一个算法就是梯度下降算法，但是求梯度就要求偏导数，下面以图中字母为例讲解一下：

设最终误差为  $E$  且输出层的激活函数为线性激活函数，对于输出那么  $E$  对于输出节点  $y_l$  的偏导数是  $y_l - t_l$ ，其中  $t_l$  是真实值， $\frac{\partial y_l}{\partial z_l}$  是指上面提到的激活函数， $z_l$  是上面提到的加权和，那么这一层的  $E$  对于  $z_l$  的偏导数为  $\frac{\partial E}{\partial z_l} = \frac{\partial E}{\partial y_l} \frac{\partial y_l}{\partial z_l}$ 。同理，下一层也是这么计算，只不过  $\frac{\partial E}{\partial y_k}$  计算方法变了，一直反向传播到输入层，最后有  $\frac{\partial E}{\partial x_i} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial z_j}$ ，且  $\frac{\partial z_j}{\partial x_i} = w_{ij}$ 。然后调整这些过程中的权值，再不断进行前向传播和反向传播的过程，最终得到一个比较好的结果。

**2.2. 如何计算神经网络的输出？** 如上图，输入层有三个节点，我们将其依次编号为 1、2、3；隐藏层的 4 个节点，编号依次为 4、5、6、7；最后输出层的两个节点编号为 8、9。比如，隐藏层的节点 4，它和输入层的三个节点 1、2、3 之间都有连接，其连接上的权重分别为是  $w_{\{41\}}, w_{\{42\}}, w_{\{43\}}$ 。

为了计算节点 4 的输出值，我们必须先得到其所有上游节点（也就是节点 1、2、3）的输出值。节点 1、2、3 是输入层的节点，所以，他们的输出值就是输入向量本身。按照上图画出的对应关系，可以看到节点 1、2、3 的输出值分别是  $x_1, x_2, x_3$ 。

$$a_4 = \sigma(w^T \cdot a) = \sigma(w_{41}x_4 + w_{42}x_2 + w_{43}x_3 + w_{4b})$$

其中  $w_{\{4b\}}$  是节点 4 的偏置项。

同样，我们可以继续计算出节点 5、6、7 的输出值  $a_5, a_6, a_7$ 。

计算输出层的节点 8 的输出值  $y_1$ :

$$y_1 = \sigma(w^T \cdot a) = \sigma(w_{84}a_4 + w_{85}a_5 + w_{86}a_6 + w_{87}a_7 + w_{8b})$$

其中  $w_{\{8b\}}$  是节点 8 的偏置项。

同理，我们还可以计算出  $y_2$ 。这样输出层所有节点的输出值计算完毕，我们就得到了在输入向量  $x_1, x_2, x_3, x_4$  时，神经网络的输出向量  $y_1, y_2$ 。这里我们也看到，输出向量的维度和输出层神经元个数相同。

**2.3. 如何计算卷积神经网络输出值？** 假设有一个  $5 \times 5$  的图像，使用一个  $3 \times 3$  的 filter 进行卷积，想得到一个  $3 \times 3$  的 Feature Map，如下所示：

$x_{\{i,j\}}$  表示图像第  $i$  行第  $j$  列元素。 $w_{\{m,n\}}$  表示 filter 第  $m$  行第  $n$  列权重。 $w_b$  表示 filter 的偏置项。表  $a_{i,j}$  示 feature map 第  $i$  行第  $j$  列元素。 $f$  表示激活函数，这里以  $ReLU$  函数为例。

卷积计算公式如下：

$$a_{i,j} = f\left(\sum_{m=0}^2 \sum_{n=0}^2 w_{m,n} x_{i+m, j+n} + w_b\right)$$

当步长为 1 时，计算 feature map 元素  $a_{\{0,0\}}$  如下：

$$a_{\{0,0\}} = f(\sum_{\{m=0\}}^2 \sum_{\{n=0\}}^2 2w_{\{m,n\}} x_{\{0+m, 0+n\}} + w_b) = \text{relu}(w_{\{0,0\}} x_{\{0,0\}} + w_{\{0,1\}} x_{\{0,1\}} + \dots)$$

**3.1. 什么是超参数？** 超参数：在机器学习的上下文中，超参数是在开始学习过程之前设置值的参数，而不是通过训练得到的参数数据。通常情况下，需要对超参数进行优化，给学习机选择一组最优超参数，以提高学习的性能和效果。

超参数通常存在于：

- 1. 定义关于模型的更高层次的概念，如复杂性或学习能力。
- 2. 不能直接从标准模型培训过程中的数据中学习，需要预先定义。
- 3. 可以通过设置不同的值，训练不同的模型和选择更好的测试值来决定

超参数具体来讲比如算法中的学习率 (learning rate)、梯度下降法迭代的数量 (iterations)、隐藏层数目 (hidden layers)、隐藏层单元数目、激活函数 (activation function) 都需要根据实际情况来设置，这些数字实际上控制了最后的参数和的值，所以它们被称作超参数。

**3.2. 如何寻找超参数的最优值？** 在使用机器学习算法时，总有一些难调的超参数。例如权重衰减大小，高斯核宽度等等。这些参数需要人为设置，设置的值对结果产生较大影响。常见设置超参数的方法有：

1. 猜测和检查：根据经验或直觉，选择参数，一直迭代。
2. 网格搜索：让计算机尝试在一定范围内均匀分布的一组值。
3. 随机搜索：让计算机随机挑选一组值。
4. 贝叶斯优化：使用贝叶斯优化超参数，会遇到贝叶斯优化算法本身就需要很多的参数的困难。
5. MITIE 方法，好初始猜测的前提下进行局部优化。它使用 BOBYQA 算法，并有一个精心选择的起始点。由于 BOBYQA 只寻找最近的局部最优解，所以这个方法是否成功很大

程度上取决于是否有一个好的起点。在 MITIE 的情况下，我们知道一个好的起点，但这不是一个普遍的解决方案，因为通常你不会知道好的起点在哪里。从好的方面来说，这种方法非常适合寻找局部最优解。稍后我会再讨论这一点。

6. 最新提出的 LIPO 的全局优化方法。这个方法没有参数，而且经验证比随机搜索方法好。

**3.3. 超参数搜索一般过程？** 超参数搜索一般过程：1. 将数据集划分成训练集、验证集及测试集。2. 在训练集上根据模型的性能指标对模型参数进行优化。3. 在验证集上根据模型的性能指标对模型的超参数进行搜索。4. 步骤 2 和步骤 3 交替迭代，最终确定模型的参数和超参数，在测试集中验证评价模型的优劣。

其中，搜索过程需要搜索算法，一般有：网格搜索、随机搜过、启发式智能搜索、贝叶斯搜索。

## 4. 激活函数

### 4.1. 为什么需要非线性激活函数？为什么需要激活函数？

1. 激活函数对模型学习、理解非常复杂和非线性的函数具有重要作用。
2. 激活函数可以引入非线性因素。如果不使用激活函数，则输出信号仅是一个简单的线性函数。线性函数一个一级多项式，线性方程的复杂度有限，从数据中学习复杂函数映射的能力很小。没有激活函数，神经网络将无法学习和模拟其他复杂类型的数据，例如图像、视频、音频、语音等。
3. 激活函数可以把当前特征空间通过一定的线性映射转换到另一个空间，让数据能够更好的被分类。

### 为什么激活函数需要非线性函数？

1. 假若网络中全部是线性部件，那么线性的组合还是线性，与单独一个线性分类器无异。这样就做不到用非线性来逼近任意函数。
2. 使用非线性激活函数，以便使网络更加强大，增加它的能力，使它可以学习复杂的事物，复杂的表单数据，以及表示输入输出之间非线性的复杂的任意函数映射。使用非线性激活函数，能够从输入输出之间生成非线性映射。

### 4.2. 常见的激活函数及图像.

#### 1. sigmoid 激活函数

函数的定义为： $f(x) = \frac{1}{1+e^{-x}}$ ，其值域为 (0, 1)。

函数图像如下：

#### 2. tanh 激活函数

函数的定义为： $f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ ，值域为 (-1, 1)。

函数图像如下：

#### 3. Relu 激活函数

函数的定义为:  $f(x) = \max(0, x)$ , 值域为  $[0, +)$ ;

函数图像如下:

#### 4. Leak Relu 激活函数

函数定义为:  $f(x) = \begin{cases} ax, & x < 0 \\ x, & x \geq 0 \end{cases}$

$$ax, \quad x < 0$$

$$x, \quad x \geq 0$$

, 值域为  $(-, +)$ 。

图像如下 ( $a = 0.5$ ):

#### 5. SoftPlus 激活函数

函数的定义为:  $f(x) = \ln(1 + ex)$ , 值域为  $(0, +)$ 。

函数图像如下:

#### 6. softmax 函数

函数定义为:  $\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}$ 。

Softmax 多用于多分类神经网络输出。

### 4.3. 3.4.3 常见激活函数的导数计算 ?.

对常见激活函数, 导数计算如下:

原函数	函数表达式	导数	备注
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = \frac{1}{1+e^{-x}} \left(1 - \frac{1}{1+e^{-x}}\right) = f(x)(1-f(x))$	当 $x = 10$ , 或 $x = -10$ , $f'(x) \approx 0$ , 当 $x = 0 f'(x) = 0.25$
Tanh	$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$f'(x) = -(tanh(x))^2$	当 $x = 10$ , 或 $x = -10$ , $f'(x) \approx 0$ , 当 $x = 0 f'(x) = 1$
Relu	$f(x) = \max(0, x)$	$c(u) = \begin{cases} 0, & x < 0 \\ 1, & x > 0 \\ undefined, & x = 0 \end{cases}$	通常 $x = 0$ 时, 给定其导数 为 1 和 0

### 4.4. 激活函数有哪些性质 ?.

1. 非线性: 当激活函数是非线性的, 一个两层的神经网络就可以基本上逼近所有的函数。但如果激活函数是恒等激活函数的时候, 即  $f(x) = x$ , 就不满足这个性质, 而且如果 MLP 使用的是恒等激活函数, 那么其实整个网络跟单层神经网络是等价的;
2. 可微性: 当优化方法是基于梯度的时候, 就体现了该性质;

3. 单调性：当激活函数是单调的时候，单层网络能够保证是凸函数；
4.  $f(x)x$ ：当激活函数满足这个性质的时候，如果参数的初始化是随机的较小值，那么神经网络的训练将会很高效；如果不满足这个性质，那么就需要详细地去设置初始值；
5. 输出值的范围：当激活函数输出值是有限的时候，基于梯度的优化方法会更加稳定，因为特征的表示受有限权值的影响更显著；当激活函数的输出是无限的时候，模型的训练会更加高效，不过在这种情况下，一般需要更小的 Learning Rate。

**4.5. 如何选择激活函数？** 选择一个适合的激活函数不容易，需要考虑很多因素，通常的做法是，如果不确定哪一个激活函数效果更好，可以把它们都试试，然后在验证集或者测试集上进行评价。然后看哪一种表现的更好，就去使用它。

以下是常见的选择情况：

1. 如果输出是 0、1 值（二分类问题），则输出层选择 sigmoid 函数，然后其它的所有单元都选择 Relu 函数。
2. 如果在隐藏层上不确定使用哪个激活函数，那么通常会使用 Relu 激活函数。有时，也会使用 tanh 激活函数，但 Relu 的一个优点是：当是负值的时候，导数等于 0。
3. sigmoid 激活函数：除了输出层是一个二分类问题基本不会用它。
4. tanh 激活函数：tanh 是非常优秀的，几乎适合所有场合。
5. ReLu 激活函数：最常用的默认函数，如果不确定用哪个激活函数，就使用 ReLu 或者 Leaky ReLU，再去尝试其他的激活函数。
6. 如果遇到了一些死的神经元，我们可以使用 Leaky ReLU 函数。

#### 4.6. 使用 ReLu 激活函数的优点？

1. 在区间变动很大的情况下，ReLu 激活函数的导数或者激活函数的斜率都会远大于 0，在程序实现就是一个 if-else 语句，而 sigmoid 函数需要进行浮点四则运算，在实践中，使用 ReLu 激活函数神经网络通常会比使用 sigmoid 或者 tanh 激活函数学习的更快。
2. sigmoid 和 tanh 函数的导数在正负饱和区的梯度都会接近于 0，这会造成梯度弥散，而 Relu 和 Leaky ReLU 函数大于 0 部分都为常数，不会产生梯度弥散现象。
3. 需注意，Relu 进入负半区的时候，梯度为 0，神经元此时不会训练，产生所谓的稀疏性，而 Leaky ReLU 不会产生这个问题。

#### 4.7. 3.4.7 什么时候可以用线性激活函数？

1. 输出层，大多使用线性激活函数。
2. 在隐含层可能会使用一些线性激活函数。
3. 一般用到的线性激活函数很少。

#### 4.8. 3.4.8 怎样理解 Relu (< 0 时) 是非线性激活函数？

根据图像可看出具有如下特点：

1. 单侧抑制；

2. 相对宽阔的兴奋边界;
3. 稀疏激活性;

ReLU 函数从图像上看，是一个分段线性函数，把所有的负值都变为 0，而正值不变，这样就成为单侧抑制。

因为有了这单侧抑制，才使得神经网络中的神经元也具有了稀疏激活性。

**稀疏激活性：**从信号方面来看，即神经元同时只对输入信号的少部分选择性响应，大量信号被刻意的屏蔽了，这样可以提高学习的精度，更好更快地提取稀疏特征。当  $x < 0$  时，ReLU 硬饱和，而当  $x > 0$  时，则不存在饱和问题。ReLU 能够在  $x > 0$  时保持梯度不衰减，从而缓解梯度消失问题。

**4.9. 3.4.9 Softmax 定义及作用.** Softmax 是一种形如下式的函数：

$$P(i) = \frac{\exp(\theta_i^T x)}{\sum_{k=1}^K \exp(\theta_k^T x)}$$

其中， $\theta_i$  和  $x$  是列向量， $\theta_i^T x$  可能被换成函数关于  $x$  的函数  $f_i(x)$

通过 softmax 函数，可以使得  $P(i)$  的范围在  $[0, 1]$  之间。在回归和分类问题中，通常  $\theta$  是待求参数，通过寻找使得  $P(i)$  最大的  $\theta_i$  作为最佳参数。

但是，使得范围在  $[0, 1]$  之间的方法有很多，为啥要在前面加上以  $e$  的幂函数的形式呢？参考 logistic 函数：

$$P(i) = \frac{1}{1 + \exp(-\theta_i^T x)}$$

这个函数的作用就是使得  $P(i)$  在负无穷到 0 的区间趋向于 0，在 0 到正无穷的区间趋向 1.。同样 softmax 函数加入了  $e$  的幂函数正是为了两极化：正样本的结果将趋近于 1，而负样本的结果趋近于 0。这样为多类别提供了方便（可以把  $P(i)$  看做是样本属于类别的概率）。可以说，Softmax 函数是 logistic 函数的一种泛化。

softmax 函数可以把它的输入，通常被称为 logits 或者 logit scores，处理成 0 到 1 之间，并且能够把输出归一化到和为 1。这意味着 softmax 函数与分类的概率分布等价。它是一个网络预测多酚类问题的最佳输出激活函数。

**4.10. 3.4.10 Softmax 函数如何应用于多分类？** softmax 用于多分类过程中，它将多个神经元的输出，映射到  $(0, 1)$  区间内，可以看成概率来理解，从而来进行多分类！

假设我们有一个数组， $V_i$  表示  $V$  中的第  $i$  个元素，那么这个元素的 softmax 值就是

$$S_i = \frac{e^{V_i}}{\sum_j e^{V_j}}$$

从下图看，神经网络中包含了输入层，然后通过两个特征层处理，最后通过 softmax 分析器就能得到不同条件下的概率，这里需要分成三个类别，最终会得到  $y = 0, y = 1, y = 2$  的概率值。

继续看下面的图，三个输入通过 softmax 后得到一个数组  $[0.05, 0.10, 0.85]$ ，这就是 soft 的功能。

更形象的映射过程如下图所示：

\*\*\*\*

softmax 直白来说就是将原来输出是  $3, 1, -3$  通过 softmax 函数一作用，就映射成为  $(0, 1)$  的值，而这些值的累和为 1（满足概率的性质），那么我们就可以将它理解成概率，在最后选取输出结点的时候，我们就可以选取概率最大（也就是值对应最大的）结点，作为我们的预测目标！

#### 4.11. 3.4.11 交叉熵代价函数定义及其求导推导. (贡献者：黄钦建 - 华南理工大学)

神经元的输出就是  $a = \sigma(z)$ ，其中  $z = \sum w_j i_j + b$  是输入的带权和。

$$C = -\frac{1}{n} \sum [y \ln a + (1 - y) \ln(1 - a)]$$

其中 n 是训练数据的总数，求和是在所有的训练输入 x 上进行的，y 是对应的目标输出。

表达式是否解决学习缓慢的问题并不明显。实际上，甚至将这个定义看做是代价函数也不是显而易见的！在解决学习缓慢前，我们来看看交叉熵为何能够解释成一个代价函数。

将交叉熵看做是代价函数有两点原因。

第一，它是非负的， $C > 0$ 。可以看出：式子中的求和中的所有独立的项都是负数的，因为对数函数的定义域是  $(0, 1)$ ，并且求和前面有一个负号，所以结果是非负。

第二，如果对于所有的训练输入 x，神经元实际的输出接近目标值，那么交叉熵将接近 0。

假设在这个例子中， $y = 0$  而  $a = 0$ 。这是我们想到得到的结果。我们看到公式中第一个项就消去了，因为  $y = 0$ ，而第二项实际上就是  $-\ln(1 - a) = 0$ 。反之， $y = 1$  而  $a = 1$ 。所以在实际输出和目标输出之间的差距越小，最终的交叉熵的值就越低了。（这里假设输出结果不是 0，就是 1，实际分类也是这样的）

综上所述，交叉熵是非负的，在神经元达到很好的正确率的时候会接近 0。这些其实就是我们想要的代价函数的特性。其实这些特性也是二次代价函数具备的。所以，交叉熵就是很好的选择了。但是交叉熵代价函数有一个比二次代价函数更好的特性就是它避免了学习速度下降的问题。为了弄清楚这个情况，我们来算算交叉熵函数关于权重的偏导数。我们将  $a = \sigma(z)$  代入到公式中应用两次链式法则，得到：

$$\begin{aligned} & \text{to} & \frac{\partial C}{\partial w_j} &= -\frac{1}{n} \sum \frac{\partial}{\partial w_j} [y \ln a + (1 - y) \ln(1 - a)] \\ & = -\frac{1}{n} \sum \frac{\partial}{\partial a} [y \ln a + (1 - y) \ln(1 - a)] * \frac{\partial a}{\partial w_j} \\ & = -\frac{1}{n} \sum \left( \frac{y}{a} - \frac{1-y}{1-a} \right) * \frac{\partial a}{\partial w_j} \end{aligned}$$

$$= -\frac{1}{n} \sum \left( \frac{y}{\varsigma(z)} - \frac{1-y}{1-\varsigma(z)} \right) \frac{\partial \varsigma(z)}{\partial w_j}$$

$$= -\frac{1}{n} \sum \left( \frac{y}{\varsigma(z)} - \frac{1-y}{1-\varsigma(z)} \right) \varsigma'(z) x_j \quad (14)$$

根据  $\varsigma(z) = \frac{1}{1+e^{-z}}$  的定义, 和一些运算, 我们可以得到  $\varsigma'(z) = \varsigma(z)(1-\varsigma(z))$ 。化简后可得:

$$\frac{\partial C}{\partial w_j} = \frac{1}{n} \sum x_j (\varsigma(z) - y)$$

这是一个优美的公式。它告诉我们权重学习的速度受到  $\varsigma(z) - y$ , 也就是输出中的误差的控制。更大的误差, 更快的学习速度。这是我们直觉上期待的结果。特别地, 这个代价函数还避免了像在二次代价函数中类似方程中  $\varsigma'(z)$  导致的学习缓慢。当我们使用交叉熵的时候,  $\varsigma'(z)$  被约掉了, 所以我们不再需要关心它是不是变得很小。这种约除就是交叉熵带来的特效。实际上, 这也并不是非常奇迹的事情。我们在后面可以看到, 交叉熵其实只是满足这种特性的一种选择罢了。

根据类似的方法, 我们可以计算出关于偏置的偏导数。我这里不再给出详细的过程, 你可以轻易验证得到:

$$\frac{\partial C}{\partial b} = \frac{1}{n} \sum (\varsigma(z) - y)$$

再一次, 这避免了二次代价函数中类似  $\varsigma'(z)$  项导致的学习缓慢。

#### 4.12. 3.4.12 为什么 Tanh 收敛速度比 Sigmoid 快? (贡献者: 黄钦建 - 华南理工大学)

首先看如下两个函数的求导:

$$\tanh'(x) = 1 - \tanh(x)^2 \in (0, 1)$$

$$s'(x) = s(x) * (1 - s(x)) \in (0, \frac{1}{4}]$$

由上面两个公式可知  $\tanh(x)$  梯度消失的问题比 sigmoid 轻, 所以 Tanh 收敛速度比 Sigmoid 快。

#### 3.4.13

#### 4.13. 3.4.12 内聚外斥 - Center Loss. (贡献者: 李世轩 - 加州大学伯克利分校)

在计算机视觉任务中, 由于其简易性, 良好的表现, 与对分类任务的概率性理解, Cross Entropy Loss (交叉熵代价) + Softmax 组合被广泛应用于以分类任务为代表的任务中。在此应用下, 我们可将其学习过程进一步理解为: 更相似 (同类/同物体) 的图像在特征域中拥有“更近的距离”, 相反则“距离更远”。换而言之, 我们可以进一步理解为其学习了一种低类内距离 (Intra-class Distance) 与高类间距离 (Inter-class Distance) 的特征判别模型。在此 Center Loss 则可以高效的计算出这种具判别性的特征。不同于传统的 Softmax Loss, Center Loss 通过学习“特征中心”从而最小化其类内距离。其表达形式如下:

$$L_C = \frac{1}{2} \sum_{i=1}^m \|x_i - c_{y_i}\|_2^2$$

其中  $x_i$  表示 FCN(全连接层) 之前的特征,  $c_{y_i}$  表示第  $y_i$  个类别的特征中心,  $m$  表示 mini-batch 的大小. 我们很清楚的看到  $L_C$  的终极目标为最小化每个特征与其特征中心的方差, 即最小化类内距离. 其迭代公式为:

$$\frac{\partial L_C}{\partial x_i} = x_i - c_{y_i}$$

$$\Delta c_j = \frac{\sum_{i=1}^m \delta(y_i=j) \cdot (c_j - x_i)}{1 + \sum_{i=1}^m \delta(y_i=j)}$$

其中  $\delta(condition) = \{$

1      condition is True

0      otherwise

结合 Softmax, 我们可以搭配二者使用, 适当平衡这两种监督信号. 在 Softmax 拉开类间距离的同时, 利用 Center Loss 最小化类内距离. 例如:

$$\begin{aligned} & \text{to} & L = L_S + \lambda L_C \\ & = - \sum_{i=1}^m \log \frac{e^{W_y^T x_i + b_{y_i}}}{\sum_{j=1}^m e^{W_j^T x_i + b_j}} + \frac{\lambda}{2} \sum_{i=1}^m \|x_i - c_{y_i}\|_2^2 \\ (15) \end{aligned}$$

即便如此, Center Loss 仍有它的不足之处: 其特征中心为存储在网络模型之外的额外参数, 不能与模型参数一同优化. 这些额外参数将与记录每一步特征变化的自动回归均值估计 (autoregressive mean estimator) 进行更迭. 当需要学习的类别数量较大时, mini-batch 可能无力提供足够的样本进行均值估计. 若此 Center Loss 将需要平衡两种监督损失来以确定更迭, 其过程需要一个对平衡超参数的搜索过程, 使得其择值消耗昂贵.

### 5. 3.5 Batch\_Size

#### 5.1. 3.5.1 为什么需要 Batch\_Size ?

Batch 的选择, 首先决定的是下降的方向.

如果数据集比较小, 可采用全数据集的形式, 好处是:

1. 由全数据集确定的方向能够更好地代表样本总体, 从而更准确地朝向极值所在的方向。
2. 由于不同权重的梯度值差别巨大, 因此选取一个全局的学习率很困难。Full Batch Learning 可以使用 Rprop 只基于梯度符号并且针对性单独更新各权值。

对于更大的数据集, 假如采用全数据集的形式, 坏处是: 1. 随着数据集的海量增长和内存限制, 一次性载入所有的数据进来变得越来越不可行。2. 以 Rprop 的方式迭代, 会由于各个 Batch 之间的采样差异性, 各次梯度修正值相互抵消, 无法修正。这才有了后来 RMSProp 的妥协方案。

**5.2. 3.5.2 Batch\_Size 值的选择.** 假如每次只训练一个样本, 即  $\text{Batch\_Size} = 1$ 。线性神经元在均方误差代价函数的错误面是一个抛物面, 横截面是椭圆。对于多层神经元、非线性网络, 在局部依然近似是抛物面。此时, 每次修正方向以各自样本的梯度方向修正, 横冲直撞各自为政, 难以达到收敛。

既然 Batch\_Size 为全数据集或者 Batch\_Size = 1 都有各自缺点，可不可以选择一个适中的 Batch\_Size 值呢？

此时，可采用批梯度下降法（Mini-batches Learning）。因为如果数据集足够充分，那么用一半（甚至少得多）的数据训练算出来的梯度与用全部数据训练出来的梯度是几乎一样的。

### 5.3. 3.5.3 在合理范围内，增大 Batch\_Size 有何好处？

1. 内存利用率提高了，大矩阵乘法的并行化效率提高。
2. 跑完一次 epoch（全数据集）所需的迭代次数减少，对于相同数据量的处理速度进一步加快。
3. 在一定范围内，一般来说 Batch\_Size 越大，其确定的下降方向越准，引起训练震荡越小。

### 5.4. 3.5.4 盲目增大 Batch\_Size 有何坏处？

1. 内存利用率提高了，但是内存容量可能撑不住了。
2. 跑完一次 epoch（全数据集）所需的迭代次数减少，要想达到相同的精度，其所花费的时间大大增加了，从而对参数的修正也就显得更加缓慢。
3. Batch\_Size 增大到一定程度，其确定的下降方向已经基本不再变化。

### 5.5. 3.5.5 调节 Batch\_Size 对训练效果影响到底如何？

1. Batch\_Size 太小，模型表现效果极其糟糕（error 飙升）。
2. 随着 Batch\_Size 增大，处理相同数据量的速度越快。
3. 随着 Batch\_Size 增大，达到相同精度所需要的 epoch 数量越来越多。
4. 由于上述两种因素的矛盾，Batch\_Size 增大到某个时候，达到时间上的最优。
5. 由于最终收敛精度会陷入不同的局部极值，因此 Batch\_Size 增大到某些时候，达到最终收敛精度上的最优。

## 6. 3.6 归一化

### 6.1. 3.6.1 归一化含义？

1. 归纳统一样本的统计分布性。归一化在 0 – 1 之间是统计的概率分布，归一化在 -1 -- +1 之间是统计的坐标分布。
2. 无论是为了建模还是为了计算，首先基本度量单位要同一，神经网络是以样本在事件中的统计分别几率来进行训练（概率计算）和预测，且 sigmoid 函数的取值是 0 到 1 之间的，网络最后一个节点的输出也是如此，所以经常要对样本的输出归一化处理。
3. 归一化是统一在 0 – 1 之间的统计概率分布，当所有样本的输入信号都为正值时，与第一隐含层神经元相连的权值只能同时增加或减小，从而导致学习速度很慢。
4. 另外在数据中常存在奇异样本数据，奇异样本数据存在所引起的网络训练时间增加，并可能引起网络无法收敛。为了避免出现这种情况及后面数据处理的方便，加快网络学习速度，

可以对输入信号进行归一化，使得所有样本的输入信号其均值接近于 0 或与其均方差相比很小。

### 6.2. 3.6.2 为什么要归一化？

1. 为了后面数据处理的方便，归一化的确可以避免一些不必要的数值问题。
2. 为了程序运行时收敛加快。
3. 同一量纲。样本数据的评价标准不一样，需要对其量纲化，统一评价标准。这算是应用层面的需求。
4. 避免神经元饱和。啥意思？就是当神经元的激活在接近 0 或者 1 时会饱和，在这些区域，梯度几乎为 0，这样，在反向传播过程中，局部梯度就会接近 0，这会有效地“杀死”梯度。
5. 保证输出数据中数值小的不被吞食。

**6.3. 3.6.3 为什么归一化能提高求解最优解速度？** 上图是代表数据是否均一化的最优解寻解过程（圆圈可以理解为等高线）。左图表示未经归一化操作的寻解过程，右图表示经过归一化后的寻解过程。

当使用梯度下降法寻求最优解时，很有可能走“之字型”路线（垂直等高线走），从而导致需要迭代很多次才能收敛；而右图对两个原始特征进行了归一化，其对应的等高线显得很圆，在梯度下降进行求解时能较快的收敛。

因此如果机器学习模型使用梯度下降法求最优解时，归一化往往非常有必要，否则很难收敛甚至不能收敛。

### 6.4. 3.6.4 3D 图解未归一化. 例子：

假设  $w_1$  的范围在  $[-10, 10]$ ，而  $w_2$  的范围在  $[-100, 100]$ ，梯度每次都前进 1 单位，那么在  $w_1$  方向上每次相当于前进了  $1/20$ ，而在  $w_2$  上只相当于  $1/200$ ！某种意义上来说，在  $w_2$  上前进的步长更小一些，而  $w_1$  在搜索过程中会比  $w_2$  “走”得更快。

这样会导致，在搜索过程中更偏向于  $w_1$  的方向。走出了“L”形状，或者成为“之”字形。

### 6.5. 3.6.5 归一化有哪些类型？

#### 1. 线性归一化

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

适用范围：比较适用在数值比较集中的情况。

缺点：如果  $\max$  和  $\min$  不稳定，很容易使得归一化结果不稳定，使得后续使用效果也不稳定。

#### 2. 标准差标准化

$$x' = \frac{x - \mu}{\sigma}$$

含义：经过处理的数据符合标准正态分布，即均值为 0，标准差为 1 其中  $\mu$  为所有样本数据的均值， $\sigma$  为所有样本数据的标准差。

### 3. 非线性归一化

适用范围：经常用在数据分化比较大的场景，有些数值很大，有些很小。通过一些数学函数，将原始值进行映射。该方法包括  $\log$ 、指数，正切等。

**6.6. 3.6.6 局部响应归一化作用.** LRN 是一种提高深度学习准确度的技术方法。LRN 一般是在激活、池化函数后的一种方法。

在 AlexNet 中，提出了 LRN 层，对局部神经元的活动创建竞争机制，使其中响应比较大对值变得相对更大，并抑制其他反馈较小的神经元，增强了模型的泛化能力。

**6.7. 3.6.7 理解局部响应归一化.** 局部响应归一化原理是仿造生物学上活跃的神经元对相邻神经元的抑制现象（侧抑制），其公式如下：

$$b_{x,y}^i = a_{x,y}^i / (k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a_{x,y}^j)^2)^\beta$$

其中，1)  $a$ : 表示卷积层（包括卷积操作和池化操作）后的输出结果，是一个四维数组 [batch,height,width,channel]。

- batch: 批次数（每一批为一张图片）。
  - height: 图片高度。
  - width: 图片宽度。
  - channel: 通道数。可以理解成一批图片中的某一个图片经过卷积操作后输出的神经元个数，或理解为处理后的图片深度。
- 2)  $a_{\{x,y\}i}$  表示在这个输出结构中的一个位置  $[a, b, c, d]$ ，可以理解成在某一张图中的某一个通道下的某个高度和某个宽度位置的点，即第  $a$  张图的第  $d$  个通道下的高度为  $b$  宽度为  $c$  的点。
- 3)  $N$ : 论文公式中的  $N$  表示通道数 (channel)。
- 4)  $a$  n/2 k input, depth\_radius, bias k, n,  $\alpha$ ,  $\beta$  都是超参数，一般设置  $k = 2, n = 5, \alpha = 1 * e - 4, \beta = 0.75$
- 5)  $\sum \sum$  叠加的方向是沿着通道方向的，即每个点值的平方和是沿着  $a$  中的第 3 维 channel 方向的，也就是一个点同方向的前面  $n/2$  个通道（最小为第 0 个通道）和后  $n/2$  个通道（最大为第  $d - 1$  个通道）的点的平方和（共  $n + 1$  个点）。而函数的英文注解中也说明了把

input 当成是  $d$  个 3 维的矩阵，说白了就是把 input 的通道数当作 3 维矩阵的个数，叠加的方向也是在通道方向。

简单的示意图如下：

**6.8. 3.6.8 什么是批归一化 (Batch Normalization).** 以前在神经网络训练中，只是对输入层数据进行归一化处理，却没有在中间层进行归一化处理。要知道，虽然我们对输入数据进行了归一化处理，但是输入数据经过  $\sigma(WX + b)$  这样的矩阵乘法以及非线性运算之后，其数据分布很可能被改变，而随着深度网络的多层运算之后，数据分布的变化将越来越大。如果我们能在网络的中间也进行归一化处理，是否对网络的训练起到改进作用呢？答案是肯定的。

这种在神经网络中间层也进行归一化处理，使训练效果更好的方法，就是批归一化 Batch Normalization (BN)。

**6.9. 3.6.9 批归一化 (BN) 算法的优点.** 下面我们来说一下 BN 算法的优点：1. 减少了人为选择参数。在某些情况下可以取消 dropout 和 L2 正则项参数，或者采取更小的 L2 正则项约束参数；2. 减少了对学习率的要求。现在我们可以使用初始很大的学习率或者选择了较小的学习率，算法也能够快速训练收敛；3. 可以不再使用局部响应归一化。BN 本身就是归一化网络（局部响应归一化在 AlexNet 网络中存在）4. 破坏原来的数据分布，一定程度上缓解过拟合（防止每批训练中某一个样本经常被挑选到，文献说这个可以提高 1% 的精度）。5. 减少梯度消失，加快收敛速度，提高训练精度。

**6.10. 3.6.10 批归一化 (BN) 算法流程.** 下面给出 BN 算法在训练时的过程

输入：上一层输出结果  $X = \{x_1, x_2, \dots, x_m\}$ ，学习参数  $\gamma, \beta$

算法流程：

1. 计算上一层输出数据的均值

$$\mu_\beta = \frac{1}{m} \sum_{i=1}^m (x_i)$$

其中， $m$  是此次训练样本 batch 的大小。

2. 计算上一层输出数据的标准差

$$\sigma_\beta^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_\beta)^2$$

3. 归一化处理，得到

$$\hat{x}_i = \frac{x_i + \mu_\beta}{\sqrt{\sigma_\beta^2 + \epsilon}}$$

其中  $\epsilon$  是为了避免分母为 0 而加进去的接近于 0 的很小值

4. 重构，对经过上面对归一化处理得到的数据进行重构，得到

$$y_i = \gamma \hat{x}_i + \beta$$

其中， $\gamma, \beta$  为可学习参数。

注：上述是 BN 训练时的过程，但是当在投入使用时，往往只是输入一个样本，没有所谓的均值  $\mu\{\beta\}$  和标准差  $\sigma\{\beta\}^2$ 。此时，均值  $\mu\{\beta\}$  是计算所有 batch  $\mu\{\beta\}$  值的平均值得到，标准差  $\sigma\{\beta\}^2$  采用每个 batch

**6.12. 3.6.12 Weight Normalization 和 Batch Normalization 比较.** Weight Normalization 和 Batch Normalization 都属于参数重写（Reparameterization）的方法，只是采用的方式不同。

Weight Normalization 是对网络权值  $W$  进行 normalization，因此也称为 Weight Normalization；

Batch Normalization 是对网络某一层输入数据进行 normalization。

Weight Normalization 相比 Batch Normalization 有以下三点优势：

1. Weight Normalization 通过重写深度学习网络的权重  $W$  的方式来加速深度学习网络参数收敛，没有引入 minibatch 的依赖，适用于 RNN (LSTM) 网络 (Batch Normalization 不能直接用于 RNN，进行 normalization 操作，原因在于：1) RNN 处理的 Sequence 是变长的；2) RNN 是基于 time step 计算，如果直接使用 Batch Normalization 处理，需要保存每个 time step 下，mini batch 的均值和方差，效率低且占内存)。
2. Batch Normalization 基于一个 mini batch 的数据计算均值和方差，而不是基于整个 Training set 来做，相当于进行梯度计算式引入噪声。因此，Batch Normalization 不适用于对噪声敏感的强化学习、生成模型 (Generative model: GAN, VAE) 使用。相反，Weight Normalization 对通过标量  $g$  和向量  $v$  对权重  $W$  进行重写，重写向量  $v$  是固定的，因此，基于 Weight Normalization 的 Normalization 可以看做比 Batch Normalization 引入更少的噪声。
3. 不需要额外的存储空间来保存 mini batch 的均值和方差，同时实现 Weight Normalization 时，对深度学习网络进行正向信号传播和反向梯度计算带来的额外计算开销也很小。因此，要比采用 Batch Normalization 进行 normalization 操作时，速度快。但是 Weight Normalization 不具备 Batch Normalization 把网络每一层的输出  $Y$  固定在一个变化范围的作用。因此，采用 Weight Normalization 进行 Normalization 时需要特别注意参数初始值的选择。

### 6.13. 3.6.13 Batch Normalization 在什么时候用比较合适？（贡献者：黄钦建 - 华南理工大学）

在 CNN 中，BN 应用在非线性映射前。在神经网络训练时遇到收敛速度很慢，或梯度爆炸等无法训练的状况时可以尝试 BN 来解决。另外，在一般使用情况下也可以加入 BN 来加快训练速度，提高模型精度。

BN 比较适用的场景是：每个 mini-batch 比较大，数据分布比较接近。在进行训练之前，要做好充分的 shuffle，否则效果会差很多。另外，由于 BN 需要在运行过程中统计每个 mini-batch 的一阶统计量和二阶统计量，因此不适用于动态的网络结构和 RNN 网络。

## 7. 3.7 预训练与微调 (fine tuning)

### 7.1. 3.7.1 为什么无监督预训练可以帮助深度学习？深度网络存在问题：

1. 网络越深，需要的训练样本数越多。若用监督则需大量标注样本，不然小规模样本容易造成过拟合。深层网络特征比较多，会出现的多特征问题主要有多样本问题、规则化问题、特征选择问题。
2. 多层神经网络参数优化是个高阶非凸优化问题，经常得到收敛较差的局部解；
3. 梯度扩散问题，BP 算法计算出的梯度随着深度向前而显著下降，导致前面网络参数贡献很小，更新速度慢。

#### 解决方法：

逐层贪婪训练，无监督预训练 (unsupervised pre-training) 即训练网络的第一个隐藏层，再训练第二个...最后用这些训练好的网络参数值作为整体网络参数的初始值。

经过预训练最终能得到比较好的局部最优解。

### 7.2. 3.7.2 什么是模型微调 fine tuning. 用别人的参数、修改后的网络和自己的数据进行训练，使得参数适应自己的数据，这样一个过程，通常称之为微调 (fine tuning).

#### 模型的微调举例说明：

我们知道，CNN 在图像识别这一领域取得了巨大的进步。如果想将 CNN 应用到我们自己的数据集上，这时通常就会面临一个问题：通常我们的 dataset 都不会特别大，一般不会超过 1 万张，甚至更少，每一类图片只有几十或者十几张。这时候，直接应用这些数据训练一个网络的想法就不可行了，因为深度学习成功的一个关键性因素就是大量带标签数据组成的训练集。如果只利用手头上这点数据，即使我们利用非常好的网络结构，也达不到很高的 performance。这时候，fine-tuning 的思想就可以很好解决我们的问题：我们通过对 ImageNet 上训练出来的模型（如 CaffeNet, VGGNet, ResNet）进行微调，然后应用到我们自己的数据集上。

### 7.3. 3.7.3 微调时候网络参数是否更新？答案：会更新。

1. finetune 的过程相当于继续训练，跟直接训练的区别是初始化的时候。
2. 直接训练是按照网络定义指定的方式初始化。
3. finetune 是用你已经有的参数文件来初始化。

#### 7.4. 3.7.4 fine-tuning 模型的三种状态.

1. 状态一：只预测，不训练。特点：相对快、简单，针对那些已经训练好，现在要实际对未知数据进行标注的项目，非常高效；
2. 状态二：训练，但只训练最后分类层。特点：fine-tuning 的模型最终的分类以及符合要求，现在只是在他们的基础上进行类别降维。
3. 状态三：完全训练，分类层 + 之前卷积层都训练特点：跟状态二的差异很小，当然状态三比较耗时和需要训练 GPU 资源，不过非常适合 fine-tuning 到自己想要的模型里面，预测精度相比状态二也提高不少。

### 8. 3.8 权重偏差初始化

#### 8.1. 3.8.1 全部初始化为 0. 偏差初始化陷阱：都初始化为 0。

**产生陷阱原因：**因为并不知道在训练神经网络中每一个权重最后的值，但是如果进行了恰当的数据归一化后，我们可以有理由认为有一半的权重是正的，另一半是负的。令所有权重都初始化为 0，如果神经网络计算出来的输出值是一样的，神经网络在进行反向传播算法计算出来的梯度值也一样，并且参数更新值也一样。更一般地说，如果权重初始化为同一个值，网络就是对称的。

**形象化理解：**在神经网络中考虑梯度下降的时候，设想你在爬山，但身处直线形的山谷中，两边是对称的山峰。由于对称性，你所在之处的梯度只能沿着山谷的方向，不会指向山峰；你走了一步之后，情况依然不变。结果就是你只能收敛到山谷中的一个极大值，而走不到山峰上去。

**8.2. 3.8.2 全部初始化为同样的值. 偏差初始化陷阱：都初始化为一样的值。**以一个三层网络为例：首先看下结构

它的表达式为：

$$a_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)})$$

$$a_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)})$$

$$a_3^{(2)} = f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)})$$

$$h_{W,b}(x) = a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})$$

$$xa_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)})a_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 +$$

如果每个权重都一样，那么在多层网络中，从第二层开始，每一层的输入值都是相同的了也就是  $a1 = a2 = a3 = \dots$ ，既然都一样，就相当于一个输入了，为啥呢??

如果是反向传递算法（如果这里不明白请看上面的连接），其中的偏置项和权重项的迭代的偏导数计算公式如下

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) = a_{-j}\{(l)\} \delta_{-i}\{(l+1)\}$$

$$\begin{aligned} &\frac{\partial}{\partial b_i^{(l)} J(W, b; x, y) = \delta_{-i}\{(l+1)\}} \\ &\text{的计算公式} \end{aligned}$$

$$\delta_i^{(l)} = (\sum_{j=1}^{s_{t+1}} W_{ji}^{(l)} \delta_j^{(l+1)}) f'(z_i^{(l)})$$

如果用的是 sigmoid 函数

$$f'(z_i^{(l)}) = a_i^{(l)}(1 - a_i^{(l)})$$

把后两个公式代入，可以看出所得到的梯度下降法的偏导相同，不停的迭代，不停的相同，不停的迭代，不停的相同……，最后就得到了相同的值（权重和截距）。

**8.3. 3.8.3 初始化为小的随机数.** 将权重初始化为很小的数字是一个普遍的打破网络对称性的解决办法。这个想法是，神经元一开始都是随机的、独一无二的，所以它们会计算出不同的更新，并将自己整合到整个网络的各个部分。一个权重矩阵的实现可能看起来像  $W = 0.01np.random.randn(D, H)$ ，其中 `randn` 是从均值为 0 的单位标准高斯分布进行取样。通过这个公式（函数），每个神经元的权重向量初始化为一个从多维高斯分布取样的随机向量，所以神经元在输入空间中指向随机的方向（so the neurons point in random direction in the input space）。应该是指输入空间对于随机方向有影响）。其实也可以从均匀分布中来随机选取小数，但是在实际操作中看起来似乎对最后的表现并没有太大的影响。

备注：并不是数字越小就会表现的越好。比如，如果一个神经网络层的权重非常小，那么在反向传播算法就会计算出很小的梯度（因为梯度 gradient 是与权重成正比的）。在网络不断的反向传播过程中将极大地减少“梯度信号”，并可能成为深层网络的一个需要注意的问题。

**8.4. 3.8.4 用  $1/\sqrt{n}$  校准方差.** 上述建议的一个问题是，随机初始化神经元的输出的分布有一个随输入量增加而变化的方差。结果证明，我们可以通过将其权重向量按其输入的平方根（即输入的数量）进行缩放，从而将每个神经元的输出的方差标准化到 1。也就是说推荐的启发式方法（heuristic）是将每个神经元的权重向量按下面的方法进行初始化： $w = np.random.randn(n)/\sqrt{n}$ ，其中  $n$  表示输入的数量。这保证了网络中所有的神经元最初的输出分布大致相同，并在经验上提高了收敛速度。

**8.5. 3.8.5 稀疏初始化 (Sparse Initialization).** 另一种解决未校准方差问题的方法是把所有的权重矩阵都设为零，但是为了打破对称性，每个神经元都是随机连接地（从如上面所介绍的一个小的高斯分布中抽取权重）到它下面的一个固定数量的神经元。一个典型的神经元连接的数目可能是小到 10 个。

**8.6. 3.8.6 初始偏差.** 将偏差初始化为零是可能的，也是很常见的，因为非对称性破坏是由权重的小随机数导致的。因为 ReLU 具有非线性特点，所以有些人喜欢使用将所有的偏差设定为小的常数值如 0.01，因为这样可以确保所有的 ReLU 单元在最开始就激活触发（fire）并因此能够获得和传播一些梯度值。然而，这是否能够提供持续的改善还不太清楚（实际上一些结果表明这样做反而使得性能更加糟糕），所以更通常的做法是简单地将偏差初始化为 0.

## 9. 3.9 学习率

**9.1. 3.9.1 学习率的作用.** 在机器学习中，监督式学习通过定义一个模型，并根据训练集上的数据估计最优参数。梯度下降法是一个广泛被用来最小化模型误差的参数优化算法。梯度下降法通过多次迭代，并在每一步中最小化成本函数（cost 来估计模型的参数。学习率（learning rate），在迭代过程中会控制模型的学习进度。

在梯度下降法中，都是给定的统一的学习率，整个优化过程中都以确定的步长进行更新，在迭代优化的前期中，学习率较大，则前进的步长就会较长，这时便能以较快的速度进行梯度下降，而在迭代优化的后期，逐步减小学习率的值，减小步长，这样将有助于算法的收敛，更容易接近最优解。故而如何对学习率的更新成为了研究者的关注点。在模型优化中，常用到的几种学习率衰减方法有：分段常数衰减、多项式衰减、指数衰减、自然指数衰减、余弦衰减、线性余弦衰减、噪声线性余弦衰减

参数名称	参数说明
learning_rate	初始学习率
global_step	用于衰减计算的全局步数，非负，用于逐步计算衰减指数
decay_steps	衰减步数，必须是正值，决定衰减周期
decay_rate	衰减率

参数名称	参数说明
end_learning_rate	最低的最终学习率
cycle	学习率下降后是否重新上升
alpha	最小学习率
num_periods	衰减余弦部分的周期数
initial_variance	噪声的初始方差
variance_decay	衰减噪声的方差

## 9.2. 学习率衰减常用参数有哪些。

**9.3. 分段常数衰减.** 分段常数衰减需要事先定义好的训练次数区间，在对应区间置不同的学习率的常数值，一般情况刚开始的学习率要大一些，之后要越来越小，要根据样本量的大小设置区间的间隔大小，样本量越大，区间间隔要小一点。下图即为分段常数衰减的学习率变化图，横坐标代表训练次数，纵坐标代表学习率。

**9.4. 指数衰减.** 以指数衰减方式进行学习率的更新，学习率的大小和训练次数指数相关，其更新规则为：

$$\text{decayed\_learning\_rate} = \text{learning\_rate} * \text{decay\_rate}^{\frac{\text{global\_step}}{\text{decay\_steps}}}$$

这种衰减方式简单直接，收敛速度快，是最常用的学习率衰减方式，如下图所示，绿色的为学习率随训练次数的指数衰减方式，红色的即为分段常数衰减，它在一定的训练区间内保持学习率不变。

**9.5. 自然指数衰减.** 它与指数衰减方式相似，不同的在于它的衰减底数是  $e$ ，故而其收敛的速度更快，一般用于相对比较容易训练的网络，便于较快的收敛，其更新规则如下

$$\text{decayed\_learning\_rate} = \text{learning\_rate} * e^{\frac{-\text{decay\_rate}}{\text{global\_step}}}$$

下图为分段常数衰减、指数衰减、自然指数衰减三种方式的对比图，红色的即为分段常数衰减图，阶梯型曲线。蓝色线为指数衰减图，绿色即为自然指数衰减图，很明可以看到自然指数衰减方式下的学习率衰减程度要大于一般指数衰减方式，有助于更快的收敛。

**9.6. 多项式衰减.** 应用多项式衰减的方式进行更新学习率，这里会给出初始学习率和最低学习率取值，然后将会按照给定的衰减方式将学习率从初始值衰减到最低值，其更新规则如下式所示。

$$\text{global\_step} = \min(\text{global\_step}, \text{decay\_steps})$$

$$\text{decayed\_learning\_rate} = (\text{learning\_rate} - \text{end\_learning\_rate}) * \left(1 - \frac{\text{global\_step}}{\text{decay\_steps}}\right)^{\text{power}} + \text{end\_learning\_rate}$$

需要注意的是，有两个机制，降到最低学习率后，到训练结束可以一直使用最低学习率进行更新，另一个是再次将学习率调高，使用  $\text{decay\_steps}$  的倍数，取第一个大于  $\text{global\_steps}$  的结果，如下式所示。它是用来防止神经网络在训练的后期由于学习率过小而导致的网络一直在某个局部最小值附近震荡，这样可以通过在后期增大学习率跳出局部极小值。

$$\text{decay\_steps} = \text{decay\_steps} * \text{ceil}\left(\frac{\text{global\_step}}{\text{decay\_steps}}\right)$$

如下图所示，红色线代表学习率降低至最低后，一直保持学习率不变进行更新，绿色线代表学习率衰减到最低后，又会再次循环往复的升高降低。

**9.7. 余弦衰减.** 余弦衰减就是采用余弦的相关方式进行学习率的衰减，衰减图和余弦函数相似。其更新机制如下式所示：

$$\text{global\_step} = \min(\text{global\_step}, \text{decay\_steps})$$

$$\text{cosine\_decay} = 0.5 * \left(1 + \cos\left(\pi * \frac{\text{global\_step}}{\text{decay\_steps}}\right)\right)$$

$$\text{decayed} = (1 - \alpha) * \text{cosine\_decay} + \alpha$$

$$\text{decayed\_learning\_rate} = \text{learning\_rate} * \text{decayed}$$

如下图所示，红色即为标准的余弦衰减曲线，学习率从初始值下降到最低学习率后保持不变。蓝色的线是线性余弦衰减方式曲线，它是学习率从初始学习率以线性的方式下降到最低学习率值。绿色噪声线性余弦衰减方式。

## 10. Dropout 系列问题

### 10.1. 为什么要正则化？

- 深度学习可能存在过拟合问题——高方差，有两个解决方法，一个是正则化，另一个是准备更多的数据，这是非常可靠的方法，但你可能无法时时刻刻准备足够多的训练数据或者获取更多数据的成本很高，但正则化通常有助于避免过拟合或减少你的网络误差。

2. 如果你怀疑神经网络过度拟合了数据，即存在高方差问题，那么最先想到的方法可能是正则化，另一个解决高方差的方法就是准备更多数据，这也是非常可靠的办法，但你可能无法时时准备足够多的训练数据，或者，获取更多数据的成本很高，但正则化有助于避免过度拟合，或者减少网络误差。

**10.2. 为什么正则化有利于预防过拟合？** 左图是高偏差，右图是高方差，中间是 Just Right，这几张图我们在前面课程中看到过。

**10.3. 理解 dropout 正则化.** Dropout 可以随机删除网络中的神经单元，它为什么可以通过正则化发挥如此大的作用呢？

直观上理解：不要依赖于任何一个特征，因为该单元的输入可能随时被清除，因此该单元通过这种方式传播下去，并为单元的四个输入增加一点权重，通过传播所有权重，dropout 将产生收缩权重的平方范数的效果，和之前讲的 L2 正则化类似；实施 dropout 的结果它会压缩权重，并完成一些预防过拟合的外层正则化；L2 对不同权重的衰减是不同的，它取决于激活函数倍增的大小。

#### 10.4. dropout 率的选择.

1. 经过交叉验证，隐含节点 dropout 率等于 0.5 的时候效果最好，原因是 0.5 的时候 dropout 随机生成的网络结构最多。
2. dropout 也可以被用作一种添加噪声的方法，直接对 input 进行操作。输入层设为更接近 1 的数。使得输入变化不会太大 (0.8)
3. 对参数  $w$  的训练进行球形限制 (max-normalization)，对 dropout 的训练非常有用。
4. 球形半径  $c$  是一个需要调整的参数，可以使用验证集进行参数调优。
5. dropout 自己虽然也很牛，但是 dropout、max-normalization、large decaying learning rates and high momentum 组合起来效果更好，比如 max-norm regularization 就可以防止大的 learning rate 导致的参数 blow up。
6. 使用 pretraining 方法也可以帮助 dropout 训练参数，在使用 dropout 时，要将所有参数都乘以  $1/p$ 。

**10.5. dropout 有什么缺点？** dropout 一大缺点就是代价函数  $J$  不再被明确定义，每次迭代，都会随机移除一些节点，如果再三检查梯度下降的性能，实际上是很难进行复查的。定义明确的代价函数  $J$  每次迭代后都会下降，因为我们所优化的代价函数  $J$  实际上并没有明确定义，或者说在某种程度上很难计算，所以我们失去了调试工具来绘制这样的图片。我通常会关闭 dropout 函数，将 keep-prob 的值设为 1，运行代码，确保  $J$  函数单调递减。然后打开 dropout 函数，希望在 dropout 过程中，代码并未引入 bug。我觉得你也可以尝试其它方法，虽然我们并没有关于这些方法性能的数据统计，但你可以把它们与 dropout 方法一起使用。

## 11. 深度学习中常用的数据增强方法 ?

(贡献者: 黄钦建 - 华南理工大学)

- Color Jittering: 对颜色的数据增强: 图像亮度、饱和度、对比度变化 (此处对色彩抖动的理解不知是否得当);
- PCA Jittering: 首先按照 RGB 三个颜色通道计算均值和标准差, 再在整个训练集上计算协方差矩阵, 进行特征分解, 得到特征向量和特征值, 用来做 PCA Jittering;
- Random Scale: 尺度变换;
- Random Crop: 采用随机图像差值方式, 对图像进行裁剪、缩放; 包括 Scale Jittering 方法 (VGG 及 ResNet 模型使用) 或者尺度和长宽比增强变换;
- Horizontal/Vertical Flip: 水平/垂直翻转;
- Shift: 平移变换;
- Rotation/Reflection: 旋转/仿射变换;
- Noise: 高斯噪声、模糊处理;
- Label Shuffle: 类别不平衡数据的增广;

## 12. 如何理解 Internal CovariateShift ?

(贡献者: 黄钦建 - 华南理工大学)

深度神经网络模型的训练为什么会很困难? 其中一个重要的原因是, 深度神经网络涉及到很多层的叠加, 而每一层的参数更新会导致上层的输入数据分布发生变化, 通过层层叠加, 高层的输入分布变化会非常剧烈, 这就使得高层需要不断去重新适应底层的参数更新。为了训好模型, 我们需要非常谨慎地去设定学习率、初始化权重、以及尽可能细致的参数更新策略。

Google 将这一现象总结为 Internal Covariate Shift, 简称 ICS。什么是 ICS 呢?

大家都知道在统计机器学习中的一个经典假设是“源空间 (source domain) 和目标空间 (target domain) 的数据分布 (distribution) 是一致的”。如果不一致, 那么就出现了新的机器学习问题, 如 transfer learning / domain adaptation 等。而 covariate shift 就是分布不一致假设之下的一一个分支问题, 它是指源空间和目标空间的条件概率是一致的, 但是其边缘概率不同。

大家细想便会发现, 的确, 对于神经网络的各层输出, 由于它们经过了层内操作作用, 其分布显然与各层对应的输入信号分布不同, 而且差异会随着网络深度增大而增大, 可是它们所能“指示”的样本标记 (label) 仍然是不变的, 这便符合了 covariate shift 的定义。由于是对层间信号的分析, 也即是“internal”的来由。

**那么 ICS 会导致什么问题?**

简而言之, 每个神经元的输入数据不再是“独立同分布”。

其一，上层参数需要不断适应新的输入数据分布，降低学习速度。

其二，下层输入的变化可能趋向于变大或者变小，导致上层落入饱和区，使得学习过早停止。

其三，每层的更新都会影响到其它层，因此每层的参数更新策略需要尽可能的谨慎。





### 2.2. 4.2.2 模型结构. 图 4.3 AlexNet 网络结构图

如图 4.3 所示，除去下采样（池化层）和局部响应规范化操作（Local Responsible Normalization, LRN），AlexNet 一共包含 8 层，前 5 层由卷积层组成，而剩下的 3 层为全连接层。网络结构分为上下两层，分别对应两个 GPU 的操作过程，除了中间某些层 ( $C_3$  卷积层和  $F_{6-8}$  全连接层会有 GPU 间的交互)，其他层两个 GPU 分别计算结果。最后一层全连接层的输出作为  $softmax$  的输入，得到 1000 个图像分类标签对应的概率值。除去 GPU 并行结构的设计，AlexNet 网络结构与 LeNet 十分相似，其网络的参数配置如表 4.2 所示。

表 4.2 AlexNet 网络参数配置

网络层	输入尺寸	核尺寸	输出尺寸	可训练参数量
卷积层 $C_1$	$224 \times 224 \times 3$	$11 \times 11 \times$ $3/4, 48(\times 2_{GPU})$	$55 \times 55 \times$ $48(\times 2_{GPU})$	$(11 \times 11 \times 3 + 1) \times 48 \times 2$
下采样层	$55 \times 55 \times$ $S_{max}^*$	$3 \times 3/2(\times 2_{GPU})$	$27 \times 27 \times$ $48(\times 2_{GPU})$	0
卷积层 $C_2$	$27 \times 27 \times$ $48(\times 2_{GPU})$	$5 \times 5 \times$ $48/1, 128(\times 2_{GPU})$	$27 \times 27 \times$ $128(\times 2_{GPU})$	$(5 \times 5 \times 48 + 1) \times 128 \times 2$
下采样层	$27 \times 27 \times$ $S_{max}$	$3 \times 3/2(\times 2_{GPU})$ $128(\times 2_{GPU})$	$13 \times 13 \times$ $128(\times 2_{GPU})$	0
卷积层 $C_3$	$13 \times 13 \times 128 \times$ $2_{GPU}$	$3 \times 3 \times$ $256/1, 192(\times 2_{GPU})$	$13 \times 13 \times$ $192(\times 2_{GPU})$	$(3 \times 3 \times 256 + 1) \times 192 \times 2$
卷积层 $C_4$	$13 \times 13 \times$ $192(\times 2_{GPU})$	$3 \times 3 \times$ $192/1, 192(\times 2_{GPU})$	$13 \times 13 \times$ $192(\times 2_{GPU})$	$(3 \times 3 \times 192 + 1) \times 192 \times 2$
卷积层 $C_5$	$13 \times 13 \times$ $192(\times 2_{GPU})$	$3 \times 3 \times$ $192/1, 128(\times 2_{GPU})$	$13 \times 13 \times$ $128(\times 2_{GPU})$	$(3 \times 3 \times 192 + 1) \times 128 \times 2$
下采样层	$13 \times 13 \times$ $S_{max}$	$3 \times 3/2(\times 2_{GPU})$ $128(\times 2_{GPU})$	$6 \times 6 \times$ $128(\times 2_{GPU})$	0
全连接层	$6 \times 6 \times 128 \times 2_{GPU}$ $F_6^*$	$9216 \times$ $2048(\times 2_{GPU})$	$1 \times 1 \times$ $2048(\times 2_{GPU})$	$(9216 + 1) \times 2048 \times 2$

网络层	输入尺寸	核尺寸	输出尺寸	可训练参数量
全连接层	$1 \times 1 \times 2048 \times 2_{GPU}$	$4096 \times 2_{GPU}$	$1 \times 1 \times 2048(\times 2_{GPU})$	$(4096 + 1) \times 2_{GPU}$
$F_7$				
全连接层	$1 \times 1 \times 2048 \times 2_{GPU}$	$4096 \times 1000$	$1 \times 1 \times 1000$	$(4096 + 1) \times 1000$
$F_8$				

卷积层  $C_1$  输入为  $224 \times 224 \times 3$  的图片数据，分别在两个 GPU 中经过核为  $11 \times 11 \times 3$ 、步长 (stride) 为 4 的卷积卷积后，分别得到两条独立的  $55 \times 55 \times 48$  的输出数据。

下采样层  $S_{max}$  实际上是嵌套在卷积中的最大池化操作，但是为了区分没有采用最大池化的卷积层单独列出来。在  $C_{1-2}$  卷积层中的池化操作之后 (ReLU 激活操作之前)，还有一个 LRN 操作，用作对相邻特征点的归一化处理。

卷积层  $C_3$  的输入与其他卷积层不同， $13 \times 13 \times 192 \times 2_{GPU}$  表示汇聚了上一层网络在两个 GPU 上的输出结果作为输入，所以在进行卷积操作时通道上的卷积核维度为 384。

全连接层  $F_{6-8}$  中输入数据尺寸也和  $C_3$  类似，都是融合了两个 GPU 流向的输出结果作为输入。

### 2.3. 4.2.3 模型特性.

- 所有卷积层都使用 ReLU 作为非线性映射函数，使模型收敛速度更快
- 在多个 GPU 上进行模型的训练，不但可以提高模型的训练速度，还能提升数据的使用规模
- 使用 LRN 对局部的特征进行归一化，结果作为 ReLU 激活函数的输入能有效降低错误率
- 重叠最大池化 (overlapping max pooling)，即池化范围  $z$  与步长  $s$  存在关系  $z > s$  (如  $S_{max}$  中核尺度为  $3 \times 3/2$ )，避免平均池化 (average pooling) 的平均效应
- 使用随机丢弃技术 (dropout) 选择性地忽略训练中的单个神经元，避免模型的过拟合

## 3. 4.3 ZFNet

**3.1. 4.3.1 模型介绍.** ZFNet 是由 Matthew D.Zeiler 和 Rob Fergus 在 AlexNet 基础上提出的大型卷积网络，在 2013 年 ILSVRC 图像分类竞赛中以 11.19% 的错误率获得冠军 (实际上原 ZFNet 所在的队伍并不是真正的冠军，原 ZFNet 以 13.51% 错误率排在第 8，真正的冠军是 Clarifai 这个队伍，而 Clarifai 这个队伍所对应的一家初创公司的 CEO 又是

图 4.5 (a) ZFNet 第一层输出的特征图 (b) AlexNet 第一层输出的特征图 (c) AlexNet 第二层输出的特征图 (d) ZFNet 第二层输出的特征图

表 4.3 ZFNet 网络参数配置 | 网络层 | 输入尺寸 | 核尺寸 | 输出尺寸 | 可训练参数量 |

:-----  :-----  :-----  :-----
卷积层 $C_1$ *   $224 \times 224 \times 3$   $7 \times 7 \times 3/2, 96$
$110 \times 110 \times 96$   $(7 \times 7 \times 3 + 1) \times 96$     下采样层 $S_{max}$   $110 \times 110 \times 96$   $3 \times 3/2$   $55 \times 55 \times 96$
0     卷积层 $C_2$ *   $55 \times 55 \times 96$   $5 \times 5 \times 96/2, 256$   $26 \times 26 \times 256$   $(5 \times 5 \times 96 + 1) \times 256$
下采样层 $S_{max}$   $26 \times 26 \times 256$   $3 \times 3/2$   $13 \times 13 \times 256$   0     卷积层 $C_3$   $13 \times 13 \times 256$
$3 \times 3 \times 256/1, 384$   $13 \times 13 \times 384$   $(3 \times 3 \times 256 + 1) \times 384$     卷积层 $C_4$   $13 \times 13 \times 384$
$3 \times 3 \times 384/1, 384$   $13 \times 13 \times 384$   $(3 \times 3 \times 384 + 1) \times 384$     卷积层 $C_5$   $13 \times 13 \times 384$
$3 \times 3 \times 384/1, 256$   $13 \times 13 \times 256$   $(3 \times 3 \times 384 + 1) \times 256$     下采样层 $S_{max}$   $13 \times 13 \times 256$
$3 \times 3/2, 6 \times 6 \times 256$   0     全连接层 $F_6$   $6 \times 6 \times 256$   $9216 \times 4096$   $1 \times 1 \times 4096$   $(9216 + 1) \times 4096$
全连接层 $F_7$   $1 \times 1 \times 4096$   $4096 \times 4096$   $1 \times 1 \times 4096$   $(4096 + 1) \times 4096$     全连接层 $F_8$
$1 \times 1 \times 4096$   $4096 \times 1000$   $1 \times 1 \times 1000$   $(4096 + 1) \times 1000$   > 卷积层 $C_1$ 与 AlexNet 中的 $C_1$ 有所不同，采用 $7 \times 7 \times 3/2$ 的卷积核代替 $11 \times 11 \times 3/4$ ，使第一层卷积输出的结果可以包含更多的中频率特征，对后续网络层中多样化的特征组合提供更多选择，有利于捕捉更细致的特征。>> 卷积层 $C_2$ 采用了步长 2 的卷积核，区别于 AlexNet 中 $C_2$ 的卷积核步长，所以输出的维度有所差异。

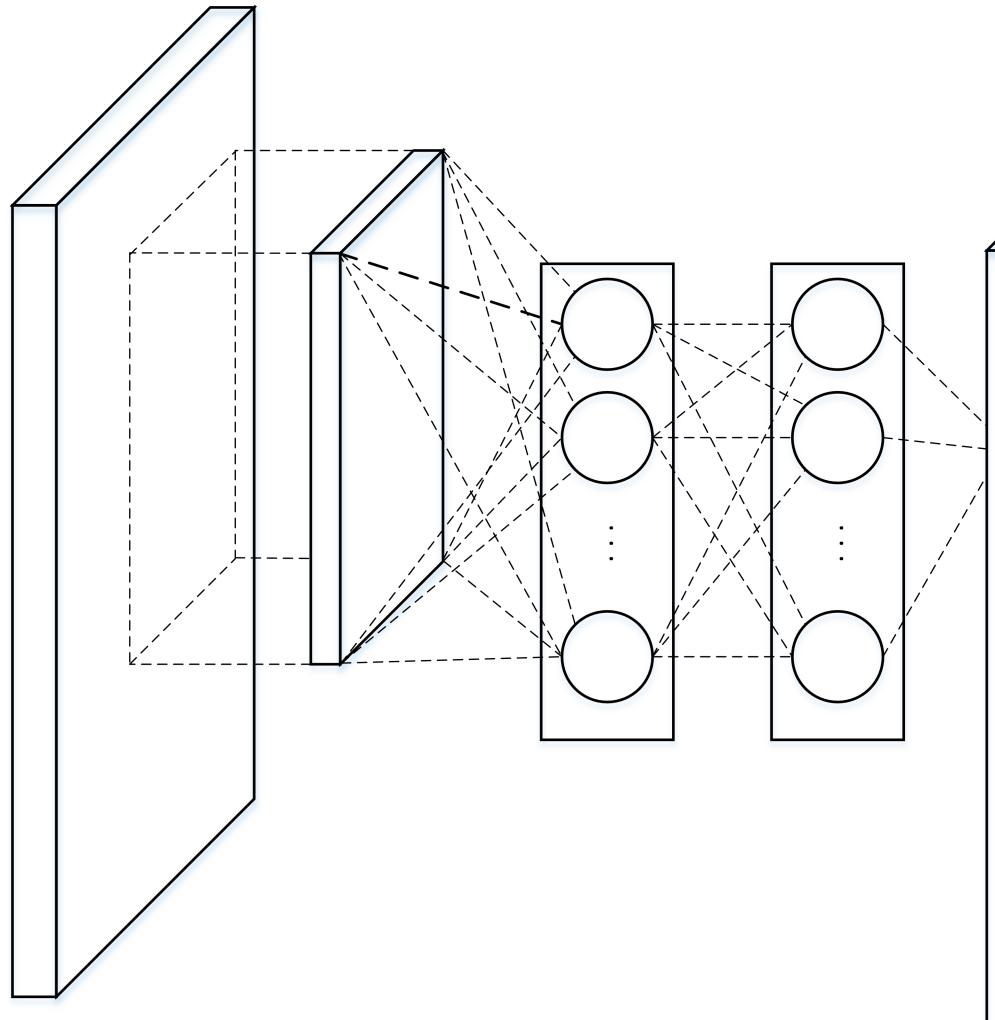
**3.3. 4.3.3 模型特性.** ZFNet 与 AlexNet 在结构上几乎相同，此部分虽属于模型特性，但准确地说应该是 ZFNet 原论文中可视化技术的贡献。

- 可视化技术揭露了激发模型中每层单独的特征图。
- 可视化技术允许观察在训练阶段特征的演变过程且诊断出模型的潜在问题。
- 可视化技术用到了多层解卷积网络，即由特征激活返回到输入像素空间。
- 可视化技术进行了分类器输出的敏感性分析，即通过阻止部分输入图像来揭示那部分对于分类是重要的。
- 可视化技术提供了一个非参数的不变性来展示来自训练集的哪一块激活哪个特征图，不仅需要裁剪输入图片，而且自上而下的投影来揭露来自每块的结构激活一个特征图。
- 可视化技术依赖于解卷积操作，即卷积操作的逆过程，将特征映射到像素上。

#### 4. 4.4 Network in Network

**4.1. 4.4.1 模型介绍.** Network In Network (NIN) 是由 *MinLin* 等人提出，在 CIFAR-10 和 CIFAR-100 分类任务中达到当时的最好水平，因其网络结构是由三个多层感知机堆叠而被

成为 NIN<sup>[5]</sup>。NIN 以一种全新的角度审视了卷积神经网络中的卷积核设计，通过引入子网络结构代替纯卷积中的线性映射部分，这种形式的网络结构激发了更复杂的卷积神经网络的结构设计，其中下一节中介绍的 GoogLeNet 的 Inception 结构就是来源于这个思想。



#### 4.2. 4.4.2 模型结构.

图 4.6 NIN 网络结构图

NIN 由三层的多层感知卷积层（MLPConv Layer）构成，每一层多层感知卷积层内部由若干层的局部全连接层和非线性激活函数组成，代替了传统卷积层中采用的线性卷积核。在网络推理（inference）时，这个多层感知器会对输入特征图的局部特征进行划窗计算，并且每个划窗的局部特征图对应的乘积的权重是共享的，这两点是和传统卷积操作完全一致的，最大的不同在于多层感知器对局部特征进行了非线性的映射，而传统卷积的方式是线性的。NIN

的网络参数配置表 4.4 所示（原论文并未给出网络参数，表中参数为编者结合网络结构图和 CIFAR-100 数据集以  $3 \times 3$  卷积为例给出）。

表 4.4 NIN 网络参数配置（结合原论文 NIN 结构和 CIFAR-100 数据给出）

网络层	输入尺寸	核尺寸	输出尺寸	参数个数
局部全连	$32 \times 32 \times 3$	$(3 \times 3) \times$	$30 \times 30 \times 16$	$(3 \times 3 \times$
接层 $L_{11}$		$16/1$		$3 + 1) \times 16$
*				
全连接层	$30 \times 30 \times 16$	$16 \times 16$	$30 \times 30 \times 16$	$((16 + 1) \times$
$L_{12}$ *				$16)$
局部全连	$30 \times 30 \times 16$	$(3 \times 3) \times$	$28 \times 28 \times 64$	$(3 \times 3 \times$
接层 $L_{21}$		$64/1$		$16 + 1) \times 64$
全连接层	$28 \times 28 \times 64$	$64 \times 64$	$28 \times 28 \times 64$	$((64 + 1) \times$
$L_{22}$				$64)$
局部全连	$28 \times 28 \times 64$	$(3 \times 3) \times$	$26 \times 26 \times 100$	$(3 \times 3 \times 64 +$
接层 $L_{31}$		$100/1$		$1) \times 100$
全连接层	$26 \times 26 \times$	$100 \times 100$	$26 \times 26 \times 100$	$((100 +$
$L_{32}$	$100$			$1) \times 100)$
全局平均	$26 \times 26 \times$	$26 \times 26 \times$	$1 \times 1 \times 100$	$0$
采样	$100$	$100/1$		
$GAP$ *				

局部全连接层  $L_{11}$  实际上是对原始输入图像进行划窗式的全连接操作，因此划窗得到的输出特征尺寸为  $30 \times 30$  ( $\frac{32-3k+1}{1_{\text{stride}}} = 30$ ) 全连接层  $L_{12}$  是紧跟  $L_{11}$  后的全连接操作，输入的特征是划窗后经过激活的局部响应特征，因此仅需连接  $L_{11}$  和  $L_{12}$  的节点即可，而每个局部全连接层和紧接的全连接层构成代替卷积操作的多层感知卷积层（MLPConv）。全局平均采样层或全局平均池化层  $GAP$  (Global Average Pooling) 将  $L_{32}$  输出的每一个特征图进行全局的平均池化操作，直接得到最后的类别数，可以有效地减少参数量。

#### 4.3. 4.4.3 模型特点.

- 使用多层感知机结构来代替卷积的滤波操作，不但有效减少卷积核数过多而导致的参数数量暴涨问题，还能通过引入非线性的映射来提高模型对特征的抽象能力。

### 5.2. 4.5.2 模型结构. 图 4.7 VGG16 网络结构图

在原论文中的 VGGNet 包含了 6 个版本的演进，分别对应 VGG11、VGG11-LRN、VGG13、VGG16-1、VGG16-3 和 VGG19，不同的后缀数值表示不同的网络层数（VGG11-LRN 表示在第一层中采用了 LRN 的 VGG11，VGG16-1 表示后三组卷积块中最后一层卷积采用卷积核尺寸为  $1 \times 1$ ，相应的 VGG16-3 表示卷积核尺寸为  $3 \times 3$ ），本节介绍的 VGG16 为 VGG16-3。图 4.7 中的 VGG16 体现了 VGGNet 的核心思路，使用  $3 \times 3$  的卷积组合代替大尺寸的卷积（2 个  $3 \times 35 \times 5$  卷积拥有相同的感受视野），网络参数设置如表 4.5 所示。

表 4.5 VGG16 网络参数配置

网络层	输入尺寸	核尺寸	输出尺寸	参数个数
卷积层 $C_{11}$	$224 \times 224 \times 3$	$3 \times 3 \times 64/1$	$224 \times 224 \times 64$	$(3 \times 3 \times 3 + 1) \times 64$
卷积层 $C_{12}$	$224 \times 224 \times 64$	$3 \times 3 \times 64/1$	$224 \times 224 \times 64$	$(3 \times 3 \times 64 + 1) \times 64$
下采样层	$224 \times 224 \times 64$	$2 \times 2/2$	$112 \times 112 \times 64$	0
$S_{max1}$				
卷积层 $C_{21}$	$112 \times 112 \times 64$	$3 \times 3 \times 128/1$	$112 \times 112 \times 128$	$(3 \times 3 \times 64 + 1) \times 128$
卷积层 $C_{22}$	$112 \times 112 \times 128$	$3 \times 3 \times 128/1$	$112 \times 112 \times 128$	$(3 \times 3 \times 128 + 1) \times 128$
下采样层	$112 \times 112 \times 128$	$2 \times 2/2$	$56 \times 56 \times 128$	0
$S_{max2}$				
卷积层 $C_{31}$	$56 \times 56 \times 128$	$3 \times 3 \times 256/1$	$56 \times 56 \times 256$	$(3 \times 3 \times 128 + 1) \times 256$
卷积层 $C_{32}$	$56 \times 56 \times 256$	$3 \times 3 \times 256/1$	$56 \times 56 \times 256$	$(3 \times 3 \times 256 + 1) \times 256$
卷积层 $C_{33}$	$56 \times 56 \times 256$	$3 \times 3 \times 256/1$	$56 \times 56 \times 256$	$(3 \times 3 \times 256 + 1) \times 256$
下采样层	$56 \times 56 \times 256$	$2 \times 2/2$	$28 \times 28 \times 256$	0
$S_{max3}$				
卷积层 $C_{41}$	$28 \times 28 \times 256$	$3 \times 3 \times 512/1$	$28 \times 28 \times 512$	$(3 \times 3 \times 256 + 1) \times 512$
卷积层 $C_{42}$	$28 \times 28 \times 512$	$3 \times 3 \times 512/1$	$28 \times 28 \times 512$	$(3 \times 3 \times 512 + 1) \times 512$
卷积层 $C_{43}$	$28 \times 28 \times 512$	$3 \times 3 \times 512/1$	$28 \times 28 \times 512$	$(3 \times 3 \times 512 + 1) \times 512$
下采样层	$28 \times 28 \times 512$	$2 \times 2/2$	$14 \times 14 \times 512$	0
$S_{max4}$				
卷积层 $C_{51}$	$14 \times 14 \times 512$	$3 \times 3 \times 512/1$	$14 \times 14 \times 512$	$(3 \times 3 \times 512 + 1) \times 512$
卷积层 $C_{52}$	$14 \times 14 \times 512$	$3 \times 3 \times 512/1$	$14 \times 14 \times 512$	$(3 \times 3 \times 512 + 1) \times 512$
卷积层 $C_{53}$	$14 \times 14 \times 512$	$3 \times 3 \times 512/1$	$14 \times 14 \times 512$	$(3 \times 3 \times 512 + 1) \times 512$
下采样层	$14 \times 14 \times 512$	$2 \times 2/2$	$7 \times 7 \times 512$	0
$S_{max5}$				

网络层	输入尺寸	核尺寸	输出尺寸	参数个数
全连接层 $FC_1$	$7 \times 7 \times 512$	$(7 \times 7 \times 512) \times 4096$	$1 \times 4096$	$(7 \times 7 \times 512 + 1) \times 4096$
全连接层 $FC_2$	$1 \times 4096$	$4096 \times 4096$	$1 \times 4096$	$(4096 + 1) \times 4096$
全连接层 $FC_3$	$1 \times 4096$	$4096 \times 1000$	$1 \times 1000$	$(4096 + 1) \times 1000$

### 5.3. 4.5.3 模型特性.

- 整个网络都使用了同样大小的卷积核尺寸  $3 \times 3$  和最大池化尺寸  $2 \times 2$ 。
- $1 \times 1$  卷积的意义主要在于线性变换，而输入通道数和输出通道数不变，没有发生降维。
- 两个  $3 \times 3$  的卷积层串联相当于 1 个  $5 \times 5$  的卷积层，感受野大小为  $5 \times 5$ 。同样地，3 个  $3 \times 3$  的卷积层串联的效果则相当于 1 个  $7 \times 7$  的卷积层。这样的连接方式使得网络参数量更小，而且多层的激活函数令网络对特征的学习能力更强。
- VGGNet 在训练时有一个小技巧，先训练浅层的简单网络 VGG11，再复用 VGG11 的权重来初始化 VGG13，如此反复训练并初始化 VGG19，能够使训练时收敛的速度更快。
- 在训练过程中使用多尺度的变换对原始数据做数据增强，使得模型不易过拟合。

## 6. 4.6 GoogLeNet

**6.1. 4.6.1 模型介绍.** GoogLeNet 作为 2014 年 ILSVRC 在分类任务上的冠军，以 6.65% 的错误率力压 VGGNet 等模型，在分类的准确率上面相比过去两届冠军 ZFNet 和 AlexNet 都有很大的提升。从名字 **GoogLeNet** 可以知道这是来自谷歌工程师所设计的网络结构，而名字中 **GoogLeNet** 更是致敬了 LeNet<sup>[0]</sup>。GoogLeNet 中最核心的部分是其内部子网络结构 Inception，该结构灵感来源于 NIN，至今已经经历了四次版本迭代 ( $Inception_{v1-v4}$ )。

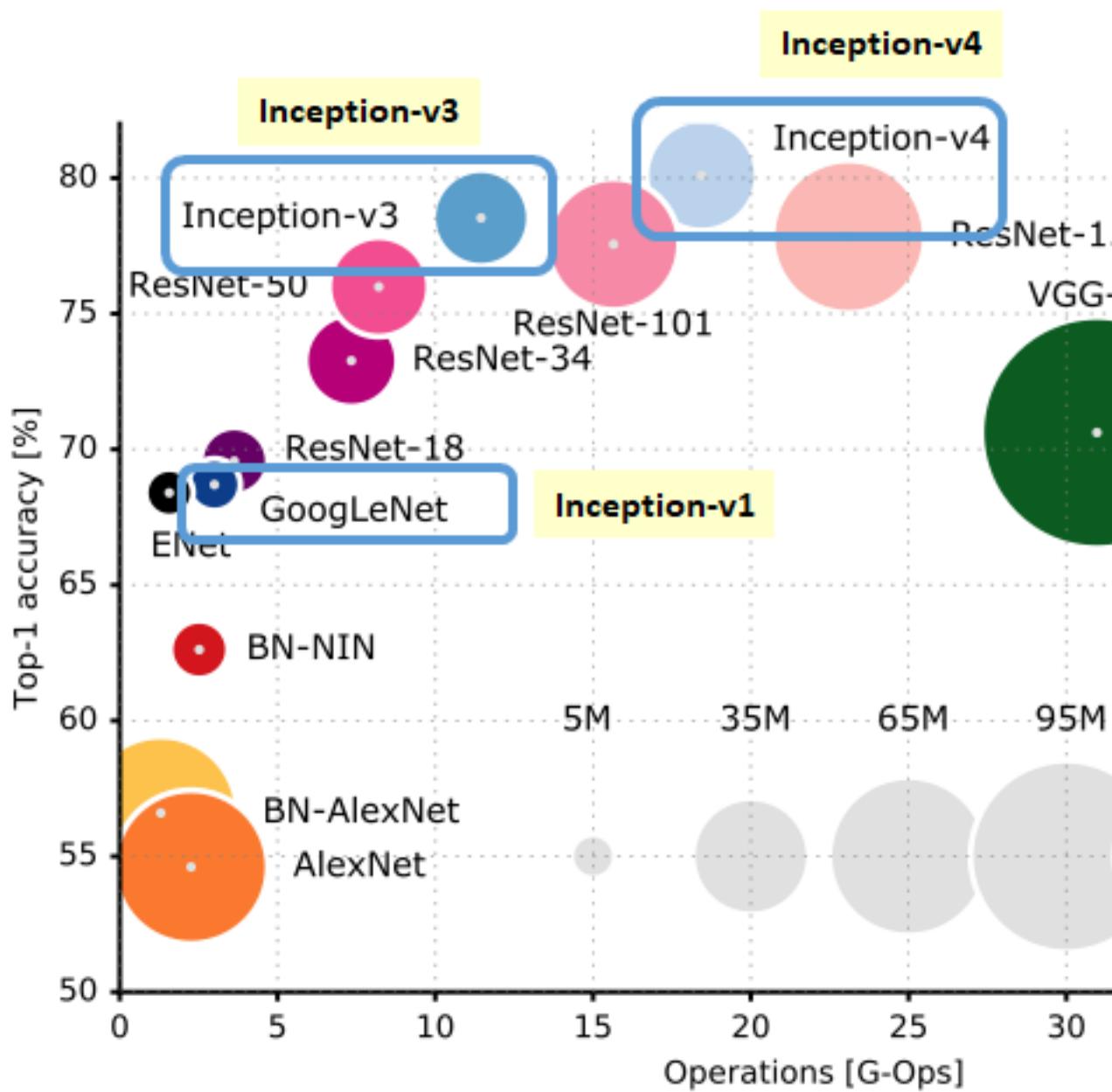
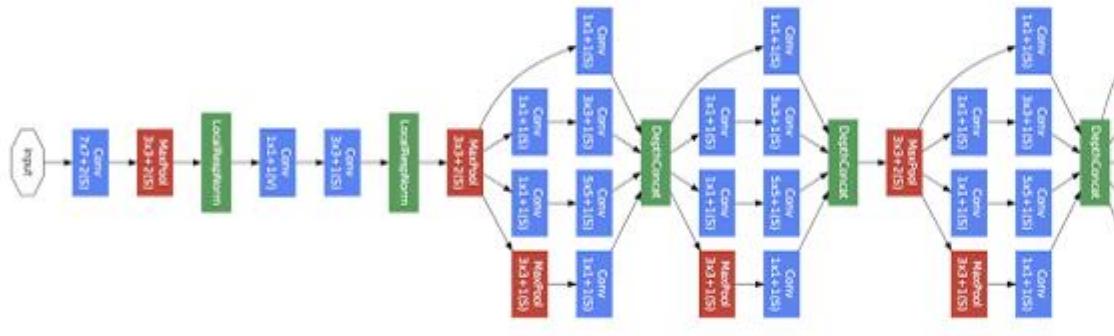


图 4.8 Inception 性能比较图

# GoogLeNet



## 6.2. 4.6.2 模型结构.

图 4.9 GoogLeNet 网络结构图 如图 4.9 中所示，GoogLeNet 相比于以前的卷积神经网络结构，除了在深度上进行了延伸，还对网络的宽度进行了扩展，整个网络由许多块状子网络的堆叠而成，这个子网络构成了 Inception 结构。图 4.9 为 Inception 的四个版本： $Inception_{v1}$  在同一层中采用不同的卷积核，并对卷积结果进行合并; $Inception_{v2}$  组合不同卷积核的堆叠形

图 4.10 Inception<sub>v1-4</sub> 结构图表 4.6 GoogLeNet 中 Inception<sub>v1</sub> 网络参数配置

网络层	输入尺寸	核尺寸	输出尺寸	参数个数
卷积层 $C_{11}$	$H \times W \times C_1$	$1 \times 1 \times C_2/2$	$\frac{H}{2} \times \frac{W}{2} \times C_2$	$(1 \times 1 \times C_1 + 1) \times C_2$
卷积层 $C_{21}$	$H \times W \times C_2$	$1 \times 1 \times C_2/2$	$\frac{H}{2} \times \frac{W}{2} \times C_2$	$(1 \times 1 \times C_2 + 1) \times C_2$
卷积层 $C_{22}$	$H \times W \times C_2$	$3 \times 3 \times C_2/1$	$H \times W \times C_2/1$	$(3 \times 3 \times C_2 + 1) \times C_2$
卷积层 $C_{31}$	$H \times W \times C_1$	$1 \times 1 \times C_2/2$	$\frac{H}{2} \times \frac{W}{2} \times C_2$	$(1 \times 1 \times C_1 + 1) \times C_2$
卷积层 $C_{32}$	$H \times W \times C_2$	$5 \times 5 \times C_2/1$	$H \times W \times C_2/1$	$(5 \times 5 \times C_2 + 1) \times C_2$
下采样层 $S_{41}$	$H \times W \times C_1$	$3 \times 3/2$	$\frac{H}{2} \times \frac{W}{2} \times C_2$	0
卷积层 $C_{42}$	$\frac{H}{2} \times \frac{W}{2} \times C_2$	$1 \times 1 \times C_2/1$	$\frac{H}{2} \times \frac{W}{2} \times C_2$	$(3 \times 3 \times C_2 + 1) \times C_2$
合并层 $M$	$\frac{H}{2} \times \frac{W}{2} \times$ $C_2(\times 4)$	拼接	$\frac{H}{2} \times \frac{W}{2} \times$ $(C_2 \times 4)$	0

### 6.3. 4.6.3 模型特性.

- 采用不同大小的卷积核意味着不同大小的感受野，最后拼接意味着不同尺度特征的融合；
- 之所以卷积核大小采用 1、3 和 5，主要是为了方便对齐。设定卷积步长 stride=1 之后，只要分别设定 pad=0、1、2，那么卷积之后便可以得到相同维度的特征，然后这些特征就可以直接拼接在一起了；
- 网络越到后面，特征越抽象，而且每个特征所涉及的感受野也更大了，因此随着层数的增加，3x3 和 5x5 卷积的比例也要增加。但是，使用 5x5 的卷积核仍然会带来巨大的计算量。为此，文章借鉴 NIN2，采用 1x1 卷积核来进行降维。

#

## 7. Restnet

## 8. Densenet

### 9. 4.7 为什么现在的 CNN 模型都是在 GoogleNet、VGGNet 或者 AlexNet 上调整的？

- 评测对比：为了让自己的结果更有说服力，在发表自己成果的时候会同一个标准的 baseline 及在 baseline 上改进而进行比较，常见的比如各种检测分割的问题都会基于 VGG 或者 Resnet101 这样的基础网络。
- 时间和精力有限：在科研压力和工作压力中，时间和精力只允许大家在有限的范围探索。

- 模型创新难度大：进行基本模型的改进需要大量的实验和尝试，并且需要大量的实验积累和强大灵感，很有可能投入产出比较小。
- 资源限制：创造一个新的模型需要大量的时间和计算资源，往往在学校和小型商业团队不可行。
- 在实际的应用场景中，其实是有大量的非标准模型的配置。







## CHAPTER 5

# 循环神经网络 (RNN)

**0.1. 为什么需要 RNN ?.** 时间序列数据是指在不同时间点上收集到的数据，这类数据反映了某一事物、现象等随时间的变化状态或程度。一般的神经网络，在训练数据足够、算法模型优越的情况下，给定特定的  $x$ ，就能得到期望  $y$ 。其一般处理单个的输入，前一个输入和后一个输入完全无关，但实际应用中，某些任务需要能够更好的处理序列的信息，即前面的输入和后面的输入是有关系的。比如：当我们在理解一句话意思时，孤立的理解这句话的每个词不足以理解整体意思，我们通常需要处理这些词连接起来的整个序列；当我们处理视频的时候，我们也不能只单独的去分析每一帧，而要分析这些帧连接起来的整个序列。为了解决一些这样类似的问题，能够更好的处理序列的信息，RNN 就由此诞生了。

## 0.2. 图解 RNN 基本结构.

0.2.1. 基本的单层网络结构. 在进一步了解 RNN 之前，先给出最基本的单层网络结构，输入是  $x$ ，经过变换  $Wx+b$  和激活函数  $f$  得到输出  $y$ :

0.2.2. 图解经典 RNN 结构. 在实际应用中，我们还会遇到很多序列形的数据，如：

- 自然语言处理问题。 $x_1$  可以看做是第一个单词， $x_2$  可以看做是第二个单词，依次类推。
- 语音处理。此时， $x_1, x_2, x_3, \dots$  是每帧的声音信号。
- 时间序列问题。例如每天的股票价格等等。

其单个序列如下图所示：

前面介绍了诸如此类的序列数据用原始的神经网络难以建模，基于此，RNN 引入了隐状态  $h$  (hidden state)， $h$  可对序列数据提取特征，接着再转换为输出。

为了便于理解，先计算  $h_1$ :

注：图中的圆圈表示向量，箭头表示对向量做变换。

RNN 中，每个步骤使用的参数  $U, W, b$  相同， $h_2$  的计算方式和  $h_1$  类似，其计算结果如下：

计算  $h_3, h_4$  也相似，可得：

接下来，计算 RNN 的输出  $y_1$ ，采用 *Softmax* 作为激活函数，根据  $y_n = f(Wx + b)$ ，得  $y_1$ :

使用和  $y_1$  相同的参数  $V, c$ , 得到  $y_1, y_2, y_3, y_4$  的输出结构:

以上即为最经典的 RNN 结构, 其输入为  $x_1, x_2, x_3, x_4$ , 输出为  $y_1, y_2, y_3, y_4$ , 当然实际中最大值为  $y_n$ , 这里为了便于理解和展示, 只计算 4 个输入和输出。从以上结构可看出, RNN 结构的输入和输出等长。

0.2.3. *vector-to-sequence* 结构. 有时我们要处理的问题输入是一个单独的值, 输出是一个序列。此时, 有两种主要建模方式:

方式一: 可只在其中的某一个序列进行计算, 比如序列第一个进行输入计算, 其建模方式如下:

方式二: 把输入信息 X 作为每个阶段的输入, 其建模方式如下:

0.2.4. *sequence-to-vector* 结构. 有时我们要处理的问题输入是一个序列, 输出是一个单独的值, 此时通常在最后的一个序列上进行输出变换, 其建模如下所示:

0.2.5. *Encoder-Decoder* 结构. 原始的 *sequence-to-sequence* 结构的 RNN 要求序列等长, 然而我们遇到的大部分问题序列都是不等长的, 如机器翻译中, 源语言和目标语言的句子往往并没有相同的长度。

其建模步骤如下:

**步骤一:** 将输入数据编码成一个上下文向量  $c$ , 这部分称为 Encoder, 得到  $c$  有多种方式, 最简单的方法就是把 Encoder 的最后一个隐状态赋值给  $c$ , 还可以对最后的隐状态做一个变换得到  $c$ , 也可以对所有的隐状态做变换。其示意如下所示:

**步骤二:** 用另一个 RNN 网络 (我们将其称为 Decoder) 对其进行编码, 方法一是将步骤一中的  $c$  作为初始状态输入到 Decoder, 示意图如下所示:

方法二是将  $c$  作为 Decoder 的每一步输入, 示意图如下所示:

网络结

构      结构图示  
1 vs N

应用场景举例

1、从图像生成文字, 输入为图像的特征, 输出为一段  
句子 2、根据图像生成语音或音乐, 输入为图像特征,  
输出为一段语音或音乐

N vs 1

1、输出一段文字, 判断其所属类别 2、输入一个句子,  
判断其情感倾向 3、输入一段视频, 判断其所属类别

N vs M

1、机器翻译, 输入一种语言文本序列, 输出另外一种  
语言的文本序列 2、文本摘要, 输入文本序列, 输出这  
段文本序列摘要 3、阅读理解, 输入文章, 输出问题答  
案 4、语音识别, 输入语音序列信息, 输出文字序列

0.2.6. 以上三种结构各有怎样的应用场景.

0.2.7. 6.2.7 图解 RNN 中的 *Attention* 机制. 在上述通用的 Encoder-Decoder 结构中, Encoder 把所有的输入序列都编码成一个统一的语义特征  $c$  再解码, 因此,  $c$  中必须包含原始序列中的所有信息, 它的长度就成了限制模型性能的瓶颈。如机器翻译问题, 当要翻译的句子较长时, 一个  $c$  可能存不下那么多信息, 就会造成翻译精度的下降。Attention 机制通过在每个时间输入不同的  $c$  来解决此问题。

引入了 Attention 机制的 Decoder 中, 有不同的  $c$ , 每个  $c$  会自动选择与当前输出最匹配的上下文信息, 其示意图如下所示:

**举例**, 比如输入序列是“我爱中国”, 要将此输入翻译成英文:

假如用  $a_{ij}$  衡量 Encoder 中第  $j$  阶段的  $h_j$  和解码时第  $i$  阶段的相关性,  $a_{ij}$  从模型中学得, 和 Decoder 的第  $i - 1$  阶段的隐状态、Encoder 第  $j$  个阶段的隐状态有关, 比如  $a_{3j}$  的计算示意如下所示:

最终 Decoder 中第  $i$  阶段的输入的上下文信息  $c_i$  来自于所有  $h_j$  对  $a_{ij}$  的加权和。

其示意图如下图所示:

在 Encoder 中,  $h_1, h_2, h_3, h_4$  分别代表“我”, “爱”, “中”, “国”所代表信息。翻译的过程中,  $c_1$  会选择和“我”最相关的上下午信息, 如上图所示, 会优先选择  $a_{11}$ , 以此类推,  $c_2$  会优先选择相关性较大的  $a_{22}$ ,  $c_3$  会优先选择相关性较大的  $a_{33}a_{34}$ , 这就是 attention 机制。

### 0.3. 6.3 RNNs 典型特点 ?

1. RNNs 主要用于处理序列数据。对于传统神经网络模型, 从输入层到隐含层再到输出层, 层与层之间一般为全连接, 每层之间神经元是无连接的。但是传统神经网络无法处理数据间的前后关联问题。例如, 为了预测句子的下一个单词, 一般需要该词之前的语义信息。这是因为一个句子中前后单词是存在语义联系的。
2. RNNs 中当前单元的输出与之前步骤输出也有关, 因此称之为循环神经网络。具体的表现形式为当前单元会对之前步骤信息进行储存并应用于当前输出的计算中。隐藏层之间的节点连接起来, 隐藏层当前输出由当前时刻输入向量和之前时刻隐藏层状态共同决定。
3. 标准的 RNNs 结构图, 图中每个箭头代表做一次变换, 也就是说箭头连接带有权值。
4. 在标准的 RNN 结构中, 隐层的神经元之间也是带有权值的, 且权值共享。
5. 理论上, RNNs 能够对任何长度序列数据进行处理。但是在实践中, 为了降低复杂度往往假设当前的状态只与之前某几个时刻状态相关, 下图便是一个典型的 RNNs:

输入单元 (Input units): 输入集  $\{x_0, x_1, \dots, x_t, x_{t+1}, \dots\}$ ,

输出单元 (Output units): 输出集  $\{y_0, y_1, \dots, y_t, y_{t+1}, \dots\}$ ,

隐藏单元 (Hidden units): 输出集  $\{s_0, s_1, \dots, s_t, s_{t+1}, \dots\}$ 。

**图中信息传递特点:**

1. 一条单向流动的信息流是从输入单元到隐藏单元。
2. 一条单向流动的信息流从隐藏单元到输出单元。

3. 在某些情况下，RNNs 会打破后者的限制，引导信息从输出单元返回隐藏单元，这些被称为“Back Projections”。
4. 在某些情况下，隐藏层的输入还包括上一时刻隐藏层的状态，即隐藏层内的节点可以自连也可以互连。
5. 当前单元 (cell) 输出是由当前时刻输入和上一时刻隐藏层状态共同决定。

类别	特点描述
相同点	1、传统神经网络的扩展。2、前向计算产生结果，反向计算模型更新。3、每层神经网络横向可以多个神经元共存，纵向可以有多层神经网络连接。
不同点	1、CNN 空间扩展，神经元与特征卷积；RNN 时间扩展，神经元与多个时间输出计算 2、RNN 可以用于描述时间上连续状态的输出，有记忆功能，CNN 用于静态输出

#### 0.4. 6.4 CNN 和 RNN 的区别？

#### 0.5. 6.5 RNNs 和 FNNs 有什么区别？

1. 不同于传统的前馈神经网络 (FNNs)，RNNs 引入了定向循环，能够处理输入之间前后关联问题。
2. RNNs 可以记忆之前步骤的训练信息。**定向循环结构如下图所示：**

#### 0.6. RNNs 训练和传统 ANN 训练异同点？相同点：

1. RNNs 与传统 ANN 都使用 BP (Back Propagation) 误差反向传播算法。

#### 不同点：

1. RNNs 网络参数  $W, U, V$  是共享的 (具体在本章 6.2 节中已介绍)，而传统神经网络各层参数间没有直接联系。
2. 对于 RNNs，在使用梯度下降算法中，每一步的输出不仅依赖当前步的网络，还依赖于之前若干步的网络状态。

**0.7. 为什么 RNN 训练的时候 Loss 波动很大.** 由于 RNN 特有的 memory 会影响后期其他的 RNN 的特点，梯度时大时小，learning rate 没法个性化的调整，导致 RNN 在 train 的过程中，Loss 会震荡起伏，为了解决 RNN 的这个问题，在训练的时候，可以设置临界值，当梯度大于某个临界值，直接截断，用这个临界值作为梯度的大小，防止大幅震荡。

**0.8. 标准 RNN 前向输出流程.** 以  $x$  表示输入， $h$  是隐层单元， $o$  是输出， $L$  为损失函数， $y$  为训练集标签。 $t$  表示  $t$  时刻的状态， $V, U, W$  是权值，同一类型的连接权值相同。以下图为例进行说明标准 RNN 的前向传播算法：

对于  $t$  时刻:

$$h^{(t)} = \phi(Ux^{(t)} + Wh^{(t-1)} + b)$$

其中  $\phi()$  为激活函数, 一般会选择  $\tanh$  函数,  $b$  为偏置。

$t$  时刻的输出为:

$$o^{(t)} = Vh^{(t)} + c$$

模型的预测输出为:

$$\hat{y}^{(t)} = \sigma(o^{(t)})$$

其中  $\sigma$  为激活函数, 通常 RNN 用于分类, 故这里一般用 softmax 函数。

**0.9. BPTT 算法推导.** BPTT (back-propagation through time) 算法是常用的训练 RNN 的方法, 其本质还是 BP 算法, 只不过 RNN 处理时间序列数据, 所以要基于时间反向传播, 故叫随时间反向传播。BPTT 的中心思想和 BP 算法相同, 沿着需要优化的参数的负梯度方向不断寻找更优的点直至收敛。需要寻优的参数有三个, 分别是  $U$ 、 $V$ 、 $W$ 。与 BP 算法不同的是, 其中  $W$  和  $U$  两个参数的寻优过程需要追溯之前的历史数据, 参数  $V$  相对简单只需关注目前, 那么我们就来先求解参数  $V$  的偏导数。

$$\frac{\partial L^{(t)}}{\partial V} = \frac{\partial L^{(t)}}{\partial o^{(t)}} \cdot \frac{\partial o^{(t)}}{\partial V}$$

RNN 的损失也是会随着时间累加的, 所以不能只求  $t$  时刻的偏导。

$$L = \sum_{t=1}^n L^{(t)}$$

$$\frac{\partial L}{\partial V} = \sum_{t=1}^n \frac{\partial L^{(t)}}{\partial o^{(t)}} \cdot \frac{\partial o^{(t)}}{\partial V}$$

$W$  和  $U$  的偏导的求解由于需要涉及到历史数据, 其偏导求起来相对复杂。为了简化推导过程, 我们假设只有三个时刻, 那么在第三个时刻  $L$  对  $W$ ,  $L$  对  $U$  的偏导数分别为:

$$\frac{\partial L^{(3)}}{\partial W} = \frac{\partial L^{(3)}}{\partial o^{(3)}} \frac{\partial o^{(3)}}{\partial h^{(3)}} \frac{\partial h^{(3)}}{\partial W} + \frac{\partial L^{(3)}}{\partial o^{(3)}} \frac{\partial o^{(3)}}{\partial h^{(3)}} \frac{\partial h^{(3)}}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial W} + \frac{\partial L^{(3)}}{\partial o^{(3)}} \frac{\partial o^{(3)}}{\partial h^{(3)}} \frac{\partial h^{(3)}}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial h^{(1)}} \frac{\partial h^{(1)}}{\partial W}$$

$$\frac{\partial L^{(3)}}{\partial U} = \frac{\partial L^{(3)}}{\partial o^{(3)}} \frac{\partial o^{(3)}}{\partial h^{(3)}} \frac{\partial h^{(3)}}{\partial U} + \frac{\partial L^{(3)}}{\partial o^{(3)}} \frac{\partial o^{(3)}}{\partial h^{(3)}} \frac{\partial h^{(3)}}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial U} + \frac{\partial L^{(3)}}{\partial o^{(3)}} \frac{\partial o^{(3)}}{\partial h^{(3)}} \frac{\partial h^{(3)}}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial h^{(1)}} \frac{\partial h^{(1)}}{\partial U}$$

可以观察到, 在某个时刻的对  $W$  或是  $U$  的偏导数, 需要追溯这个时刻之前所有时刻的信息。根据上面两个式子得出  $L$  在  $t$  时刻对  $W$  和  $U$  偏导数的通式:

$$\frac{\partial L^{(t)}}{\partial W} = \sum_{k=0}^t \frac{\partial L^{(t)}}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial h^{(t)}} \left( \prod_{j=k+1}^t \frac{\partial h^{(j)}}{\partial h^{(j-1)}} \right) \frac{\partial h^{(k)}}{\partial W}$$

$$\frac{\partial L^{(t)}}{\partial U} = \sum_{k=0}^t \frac{\partial L^{(t)}}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial h^{(t)}} \left( \prod_{j=k+1}^t \frac{\partial h^{(j)}}{\partial h^{(j-1)}} \right) \frac{\partial h^{(k)}}{\partial U}$$

整体的偏导公式就是将其按时刻再一一加起来。

**0.10. RNN 中为什么会出现梯度消失？** 首先来看  $\tanh$  函数的函数及导数图如下所示：  
 $\text{sigmoid}$  函数的函数及导数图如下所示：

从上图观察可知， $\text{sigmoid}$  函数的导数范围是  $(0,0.25]$ ， $\tanh$  函数的导数范围是  $(0,1]$ ，他们的导数最大都不大于 1。

基于 6.8 中式 (9-10) 中的推导，RNN 的激活函数是嵌套在里面的，如果选择激活函数为  $\tanh$  或  $\text{sigmoid}$ ，把激活函数放进去，拿出中间累乘的那部分可得：

$$\prod_{j=k+1}^t \frac{\partial h^j}{\partial h^{j-1}} = \prod_{j=k+1}^t \tanh' \cdot W_s$$

$$\prod_{j=k+1}^t \frac{\partial h^j}{\partial h^{j-1}} = \prod_{j=k+1}^t \text{sigmoid}' \cdot W_s$$

**梯度消失现象：**基于上式，会发现累乘会导致激活函数导数的累乘，如果取  $\tanh$  或  $\text{sigmoid}$  函数作为激活函数的话，那么必然是一堆小数在做乘法，结果就是越乘越小。随着时间序列的不断深入，小数的累乘就会导致梯度越来越小直到接近于 0，这就是“梯度消失”现象。

实际使用中，会优先选择  $\tanh$  函数，原因是  $\tanh$  函数相对于  $\text{sigmoid}$  函数来说梯度较大，收敛速度更快且引起梯度消失更慢。

**0.11. 如何解决 RNN 中的梯度消失问题？** 上节描述的梯度消失是在无限的利用历史数据而造成，但是 RNN 的特点本来就是能利用历史数据获取更多的可利用信息，解决 RNN 中的梯度消失方法主要有：

1、选取更好的激活函数，如 Relu 激活函数。ReLU 函数的左侧导数为 0，右侧导数恒为 1，这就避免了“梯度消失”的发生。但恒为 1 的导数容易导致“梯度爆炸”，但设定合适的阈值可以解决这个问题。

2、加入 BN 层，其优点包括可加速收敛、控制过拟合，可以少用或不用 Dropout 和正则、降低网络对初始化权重不敏感，且能允许使用较大的学习率等。

2、改变传播结构，LSTM 结构可以有效解决这个问题。下面将介绍 LSTM 相关内容。

## 0.12. LSTM.

0.12.1. *LSTM* 的产生原因. RNN 在处理长期依赖 (时间序列上距离较远的节点) 时会遇到巨大的困难, 因为计算距离较远的节点之间的联系时会涉及雅可比矩阵的多次相乘, 会造成梯度消失或者梯度膨胀的现象。为了解决该问题, 研究人员提出了许多解决办法, 例如 ESN (Echo State Network), 增加有漏单元 (Leaky Units) 等等。其中最成功应用最广泛的就是门限 RNN (Gated RNN), 而 LSTM 就是门限 RNN 中最著名的一种。有漏单元通过设计连接间的权重系数, 从而允许 RNN 累积距离较远节点间的长期联系; 而门限 RNN 则泛化了这样的思想, 允许在不同时刻改变该系数, 且允许网络忘记当前已经累积的信息。

0.12.2. 图解标准 *RNN* 和 *LSTM* 的区别. 所有 RNN 都具有一种重复神经网络模块的链式的形式。在标准的 RNN 中, 这个重复的模块只有一个非常简单的结构, 例如一个 tanh 层, 如下图所示:

LSTM 同样是这样的结构, 但是重复的模块拥有一个不同的结构。不同于单一神经网络层, 这里是有四个, 以一种非常特殊的方式进行交互。

注: 上图图标具体含义如下所示:

上图中, 每一条黑线传输着一整个向量, 从一个节点的输出到其他节点的输入。粉色的圈代表 pointwise 的操作, 诸如向量的和, 而黄色的矩阵就是学习到的神经网络层。合在一起的线表示向量的连接, 分开的线表示内容被复制, 然后分发到不同的位置。

0.12.3. 6.11.3 *LSTM* 核心思想图解. LSTM 的关键就是细胞状态, 水平线在图上方贯穿运行。细胞状态类似于传送带。直接在整个链上运行, 只有一些少量的线性交互。信息在上面流传保持不变会很容易。示意图如下所示:

LSTM 有通过精心设计的称作为“门”的结构来去除或者增加信息到细胞状态的能力。门是一种让信息选择式通过的方法。他们包含一个 sigmoid 神经网络层和一个 pointwise 乘法操作。示意图如下:

LSTM 拥有三个门, 分别是忘记层门, 输入层门和输出层门, 来保护和控制细胞状态。

### 忘记层门

作用对象: 细胞状态。

作用: 将细胞状态中的信息选择性的遗忘。

操作步骤: 该门会读取  $h_{t-1}$  和  $x_t$ , 输出一个在 0 到 1 之间的数值给每个在细胞状态  $C_{t-1}$  中的数字。1 表示“完全保留”, 0 表示“完全舍弃”。示意图如下:

### 输入层门

作用对象: 细胞状态

作用: 将新的信息选择性的记录到细胞状态中。

操作步骤:

步骤一, sigmoid 层称“输入层门”决定什么值我们将要更新。

步骤二, tanh 层创建一个新的候选值向量  $\tilde{C}_t$  加入到状态中。其示意图如下:

步骤三：将  $c_{t-1}$  更新为  $c_t$ 。将旧状态与  $f_t$  相乘，丢弃掉我们确定需要丢弃的信息。接着加上  $i_t * \tilde{C}_t$  得到新的候选值，根据我们决定更新每个状态的程度进行变化。其示意图如下：

**输出层门作用对象：隐层  $h_t$**

作用：确定输出什么值。

操作步骤：

步骤一：通过 sigmoid 层来确定细胞状态的哪个部分将输出。

步骤二：把细胞状态通过 tanh 进行处理，并将它和 sigmoid 门的输出相乘，最终我们仅会输出我们确定输出的那部分。

其示意图如下所示：

#### 0.12.4. LSTM 流行的变体. 增加 peephole 连接

在正常的 LSTM 结构中，Gers F A 等人提出增加 peephole 连接，可以门层接受细胞状态的输入。示意图如下所示：

**对忘记门和输入门进行同时确定**

不同于之前是分开确定什么忘记和需要添加什么新的信息，这里是一同做出决定。示意图如下所示：

#### Gated Recurrent Unit

由 Kyunghyun Cho 等人提出的 Gated Recurrent Unit (GRU)，其将忘记门和输入门合成了一个单一的更新门，同样还混合了细胞状态和隐藏状态，和其他一些改动。其示意图如下：最终的模型比标准的 LSTM 模型要简单，也是非常流行的变体。

#### 0.13. LSTMs 与 GRUs 的区别. LSTMs 与 GRUs 的区别如图所示：

从上图可以看出，二者结构十分相似，**不同在于**：

1. new memory 都是根据之前 state 及 input 进行计算，但是 GRUs 中有一个 reset gate 控制之前 state 的进入量，而在 LSTMs 里没有类似 gate；
2. 产生新的 state 的方式不同，LSTMs 有两个不同的 gate，分别是 forget gate (f gate) 和 input gate(i gate)，而 GRUs 只有一种 update gate(z gate)；
3. LSTMs 对新产生的 state 可以通过 output gate(o gate) 进行调节，而 GRUs 对输出无任何调节。

#### 0.14. RNNs 在 NLP 中典型应用？ (1) 语言模型与文本生成 (Language Modeling and Generating Text)

给定一组单词序列，需要根据前面单词预测每个单词出现的可能性。语言模型能够评估某个语句正确的可能性，可能性越大，语句越正确。另一种应用便是使用生成模型预测下一个单词的出现概率，从而利用输出概率的采样生成新的文本。

#### (2) 机器翻译 (Machine Translation)

机器翻译是将一种源语言语句变成意思相同的另一种源语言语句，如将英语语句变成同样意思的中文语句。与语言模型关键的区别在于，需要将源语言语句序列输入后，才进行输出，即输出第一个单词时，便需要从完整的输入序列中进行获取。

### (3) 语音识别 (Speech Recognition)

语音识别是指给定一段声波的声音信号，预测该声波对应的某种指定源语言语句以及计算该语句的概率值。

### (4) 图像描述生成 (Generating Image Descriptions)

同卷积神经网络一样，RNNs 已经在对无标图像描述自动生成中得到应用。CNNs 与 RNNs 结合也被应用于图像描述自动生成。

## 0.15. 常见的 RNNs 扩展和改进模型.

### 0.15.1. Simple RNNs(SRNs).

1. SRNs 是一个三层网络，其在隐藏层增加了上下文单元。下图中的  $y$  是隐藏层， $u$  是上下文单元。上下文单元节点与隐藏层中节点的连接是固定的，并且权值也是固定的。上下文节点与隐藏层节点一一对应，并且值是确定的。
2. 在每一步中，使用标准的前向反馈进行传播，然后使用学习算法进行学习。上下文每一个节点保存其连接隐藏层节点上一步输出，即保存上文，并作用于当前步对应的隐藏层节点状态，即隐藏层的输入由输入层的输出与上一步的自身状态所决定。因此 SRNs 能够解决标准多层感知机 (MLP) 无法解决的对序列数据进行预测的问题。SRNs 网络结构如下图所示：

0.15.2. Bidirectional RNNs. Bidirectional RNNs(双向网络) 将两层 RNNs 叠加在一起，当前时刻输出 (第  $t$  步的输出) 不仅仅与之前序列有关，还与之后序列有关。例如：为了预测一个语句中的缺失词语，就需要该词汇的上下文信息。Bidirectional RNNs 是一个相对较简单的 RNNs，是由两个 RNNs 上下叠加在一起组成的。输出由前向 RNNs 和后向 RNNs 共同决定。如下图所示：

0.15.3. Deep RNNs. Deep RNNs 与 Bidirectional RNNs 相似，其也是多层 RNNs 叠加，因此每一步的输入有了多层网络。该网络具有更强大的表达与学习能力，但是复杂性也随之提高，同时需要更多的训练数据。Deep RNNs 的结构如下图所示：

### 0.15.4. Echo State Networks (ESNs). ESNs 特点：

1. 它的核心结构为一个随机生成、且保持不变的储备池 (Reservoir)。储备池是大规模随机生成稀疏连接 (SD 通常保持  $1\% \sim 5\%$ ，SD 表示储备池中互相连接的神经元占总神经元个数  $N$  的比例) 的循环结构；
2. 从储备池到输出层的权值矩阵是唯一需要调整的部分；
3. 简单的线性回归便能够完成网络训练；

**ESNs 基本思想：**

使用大规模随机连接的循环网络取代经典神经网络中的中间层，从而简化网络的训练过程。网络中的参数包括：(1)  $W$  - 储备池中节点间连接权值矩阵；(2)  $W_{in}$  - 输入层到储备池之间连接权值矩阵，表明储备池中的神经元之间是相互连接；(3)  $W_{back}$  - 输出层到储备池之间的反馈连接权值矩阵，表明储备池会有输出层来的反馈；(4)  $W_{out}$  - 输入层、储备池、输出层到输出层的连接权值矩阵，表明输出层不仅与储备池连接，还与输入层和自己连接。(5)  $W_{outbias}$  - 输出层的偏置项。

ESNs 的结构如下图所示：

0.15.5. *Gated Recurrent Unit Recurrent Neural Networks.* GRUs 是一般的 RNNs 的变型版本，其主要是从以下两个方面进行改进。

1. 以语句为例，序列中不同单词处的数据对当前隐藏层状态的影响不同，越前面的影响越小，即每个之前状态对当前的影响进行了距离加权，距离越远，权值越小。
2. 在产生误差 error 时，其可能是由之前某一个或者几个单词共同造成，所以应当对对应的单词 weight 进行更新。GRUs 的结构如下图所示。GRUs 首先根据当前输入单词向量 word vector 以及前一个隐藏层状态 hidden state 计算出 update gate 和 reset gate。再根据 reset gate、当前 word vector 以及前一个 hidden state 计算新的记忆单元内容 (new memory content)。当 reset gate 为 1 的时候，new memory content 忽略之前所有 memory content，最终的 memory 是由之前的 hidden state 与 new memory content 一起决定。

#### 0.15.6. *Bidirectional LSTMs.*

1. 与 bidirectional RNNs 类似，bidirectional LSTMs 有两层 LSTMs。一层处理过去的训练信息，另一层处理将来的训练信息。
2. 在 bidirectional LSTMs 中，通过前向 LSTMs 获得前向隐藏状态，后向 LSTMs 获得后向隐藏状态，当前隐藏状态是前向隐藏状态与后向隐藏状态的组合。

#### 0.15.7. *Stacked LSTMs.*

1. 与 deep rnns 类似，stacked LSTMs 通过将多层 LSTMs 叠加起来得到一个更加复杂的模型。
2. 不同于 bidirectional LSTMs，stacked LSTMs 只利用之前步骤的训练信息。

0.15.8. *ClockworkRNNs(CW-RNNs).* CW-RNNs 是 RNNs 的改良版本，其使用时钟频率来驱动。它将隐藏层分为几个块 (组，Group/Module)，每一组按照自己规定的时钟频率对输入进行处理。为了降低 RNNs 的复杂度，CW-RNNs 减少了参数数量，并且提高了网络性能，加速网络训练。CW-RNNs 通过不同隐藏层模块在不同时钟频率下工作来解决长时依赖问题。将时钟时间进行离散化，不同的隐藏层组将在不同时刻进行工作。因此，所有的隐藏层组在每一步不会全部同时工作，这样便会加快网络的训练。并且，时钟周期小组的神经元不会连接到时钟周期大组的神经元，只允许周期大的神经元连接到周期小的 (组与组之间的连接以及信息

传递是有向的)。周期大的速度慢, 周期小的速度快, 因此是速度慢的神经元连速度快的神经元, 反之则不成立。

CW-RNNs 与 SRNs 网络结构类似, 也包括输入层 (Input)、隐藏层 (Hidden)、输出层 (Output), 它们之间存在前向连接, 输入层到隐藏层连接, 隐藏层到输出层连接。但是与 SRN 不同的是, 隐藏层中的神经元会被划分为若干个组, 设为  $g$ , 每一组中的神经元个数相同, 设为  $k$ , 并为每一个组分配一个时钟周期  $T_i \in \{T_1, T_2, \dots, T_g\}$ , 每一组中的所有神经元都是全连接, 但是组  $j$  到组  $i$  的循环连接则需要满足  $T_j > T_i$ 。如下图所示, 将这些组按照时钟周期递增从左到右进行排序, 即  $T_1 < T_2 < \dots < T_g$ , 那么连接便是从右到左。例如: 隐藏层共有 256 个节点, 分为四组, 周期分别是  $[1, 2, 4, 8]$ , 那么每个隐藏层组  $256/4=64$  个节点, 第一组隐藏层与隐藏层的连接矩阵为  $64 \times 6464 \times 12864 \times (3 \times 64) = 64 \times 19264 \times (4 \times 64) = 64 \times 256$

**CW-RNNs 的网络结构如下图所示:**

#### 0.15.9. CNN-LSTMs.

1. 为了同时利用 CNN 以及 LSTMs 的优点, CNN-LSTMs 被提出。在该模型中, CNN 用于提取对象特征, LSTMs 用于预测。CNN 由于卷积特性, 其能够快速而且准确地捕捉对象特征。LSTMs 的优点在于能够捕捉数据间的长时依赖性。







## CHAPTER 6

# 循环神经网络 (RNN)

## 1. 6.1 为什么需要 RNN ?

时间序列数据是指在不同时间点上收集到的数据，这类数据反映了某一事物、现象等随时间的变化状态或程度。一般的神经网络，在训练数据足够、算法模型优越的情况下，给定特定的  $x$ ，就能得到期望  $y$ 。其一般处理单个的输入，前一个输入和后一个输入完全无关，但实际应用中，某些任务需要能够更好的处理序列的信息，即前面的输入和后面的输入是有关系的。比如：

当我们在理解一句话意思时，孤立的理解这句话的每个词不足以理解整体意思，我们通常需要处理这些词连接起来的整个序列；当我们处理视频的时候，我们也不能只单独的去分析每一帧，而要分析这些帧连接起来的整个序列。为了解决一些这样类似的问题，能够更好的处理序列的信息，RNN 就由此诞生了。

## 2. 6.2 图解 RNN 基本结构

**2.1. 6.2.1 基本的单层网络结构.** 在进一步了解 RNN 之前，先给出最基本的单层网络结构，输入是  $x$ ，经过变换  $Wx+b$  和激活函数  $f$  得到输出  $y$ ：

**2.2. 6.2.2 图解经典 RNN 结构.** 在实际应用中，我们还会遇到很多序列形的数据，如：

- 自然语言处理问题。 $x_1$  可以看做是第一个单词， $x_2$  可以看做是第二个单词，依次类推。
- 语音处理。此时， $x_1, x_2, x_3, \dots$  是每帧的声音信号。
- 时间序列问题。例如每天的股票价格等等。

其单个序列如下图所示：

前面介绍了诸如此类的序列数据用原始的神经网络难以建模，基于此，RNN 引入了隐状态  $h$  (hidden state)， $h$  可对序列数据提取特征，接着再转换为输出。

为了便于理解，先计算  $h_1$ ：

注：图中的圆圈表示向量，箭头表示对向量做变换。

RNN 中，每个步骤使用的参数  $U, W, b$  相同， $h_2$  的计算方式和  $h_1$  类似，其计算结果如下：

计算  $h_3, h_4$  也相似，可得：

接下来，计算 RNN 的输出  $y_1$ ，采用 *Softmax* 作为激活函数，根据  $y_n = f(Wx + b)$ ，得  $y_1$ ：

使用和  $y_1$  相同的参数  $V, c$ ，得到  $y_1, y_2, y_3, y_4$  的输出结构：

以上即为最经典的 RNN 结构，其输入为  $x_1, x_2, x_3, x_4$ ，输出为  $y_1, y_2, y_3, y_4$ ，当然实际中最大值为  $y_n$ ，这里为了便于理解和展示，只计算 4 个输入和输出。从以上结构可看出，RNN 结构的输入和输出等长。

**2.3. 6.2.3 vector-to-sequence 结构.** 有时我们要处理的问题输入是一个单独的值，输出是一个序列。此时，有两种主要建模方式：

方式一：可只在其中的某一个序列进行计算，比如序列第一个进行输入计算，其建模方式如下：

方式二：把输入信息 X 作为每个阶段的输入，其建模方式如下：

**2.4. 6.2.4 sequence-to-vector 结构.** 有时我们要处理的问题输入是一个序列，输出是一个单独的值，此时通常在最后的一个序列上进行输出变换，其建模如下所示：

**2.5. 6.2.5 Encoder-Decoder 结构.** 原始的 sequence-to-sequence 结构的 RNN 要求序列等长，然而我们遇到的大部分问题序列都是不等长的，如机器翻译中，源语言和目标语言的句子往往并没有相同的长度。

其建模步骤如下：

**步骤一：**将输入数据编码成一个上下文向量  $c$ ，这部分称为 Encoder，得到  $c$  有多种方式，最简单的方法就是把 Encoder 的最后一个隐状态赋值给  $c$ ，还可以对最后的隐状态做一个变换得到  $c$ ，也可以对所有的隐状态做变换。其示意如下所示：

**步骤二：**用另一个 RNN 网络（我们将其称为 Decoder）对其进行编码，方法一是将步骤一中的  $c$  作为初始状态输入到 Decoder，示意图如下所示：

方法二是将  $c$  作为 Decoder 的每一步输入，示意图如下所示：

### 网络结

构      结构图示  
1 vs N

N vs 1

### 应用场景举例

1、从图像生成文字，输入为图像的特征，输出为一段  
句子 2、根据图像生成语音或音乐，输入为图像特征，  
输出为一段语音或音乐

1、输出一段文字，判断其所属类别 2、输入一个句子，  
判断其情感倾向 3、输入一段视频，判断其所属类别

N vs M

- 1、机器翻译，输入一种语言文本序列，输出另外一种语言的文本序列
- 2、文本摘要，输入文本序列，输出这段文本序列摘要
- 3、阅读理解，输入文章，输出问题答案
- 4、语音识别，输入语音序列信息，输出文字序列

## 2.6. 以上三种结构各有怎样的应用场景。

**2.7. 6.2.7 图解 RNN 中的 Attention 机制.** 在上述通用的 Encoder-Decoder 结构中，Encoder 把所有的输入序列都编码成一个统一的语义特征  $c$  再解码，因此， $c$  中必须包含原始序列中的所有信息，它的长度就成了限制模型性能的瓶颈。如机器翻译问题，当要翻译的句子较长时，一个  $c$  可能存不下那么多信息，就会造成翻译精度的下降。Attention 机制通过在每个时间输入不同的  $c$  来解决此问题。

引入了 Attention 机制的 Decoder 中，有不同的  $c$ ，每个  $c$  会自动选择与当前输出最匹配的上下文信息，其示意图如下所示：

**举例，** 比如输入序列是“我爱中国”，要将此输入翻译成英文：

假如用  $a_{ij}$  衡量 Encoder 中第  $j$  阶段的  $h_j$  和解码时第  $i$  阶段的相关性， $a_{ij}$  从模型中学得，和 Decoder 的第  $i - 1$  阶段的隐状态、Encoder 第  $j$  个阶段的隐状态有关，比如  $a_{3j}$  的计算示意如下所示：

最终 Decoder 中第  $i$  阶段的输入的上下文信息  $c_i$  来自于所有  $h_j$  对  $a_{ij}$  的加权和。

其示意图如下图所示：

在 Encoder 中， $h_1, h_2, h_3, h_4$  分别代表“我”，“爱”，“中”，“国”所代表信息。翻译的过程中， $c_1$  会选择和“我”最相关的上下午信息，如上图所示，会优先选择  $a_{11}$ ，以此类推， $c_2$  会优先选择相关性较大的  $a_{22}$ ， $c_3$  会优先选择相关性较大的  $a_{33}a_{34}$ ，这就是 attention 机制。

## 3. 6.3 RNNs 典型特点？

1. RNNs 主要用于处理序列数据。对于传统神经网络模型，从输入层到隐含层再到输出层，层与层之间一般为全连接，每层之间神经元是无连接的。但是传统神经网络无法处理数据间的前后关联问题。例如，为了预测句子的下一个单词，一般需要该词之前的语义信息。这是因为一个句子中前后单词是存在语义联系的。
2. RNNs 中当前单元的输出与之前步骤输出也有关，因此称之为循环神经网络。具体的表现形式为当前单元会对之前步骤信息进行储存并应用于当前输出的计算中。隐藏层之间的节点连接起来，隐藏层当前输出由当前时刻输入向量和之前时刻隐藏层状态共同决定。
3. 标准的 RNNs 结构图，图中每个箭头代表做一次变换，也就是说箭头连接带有权值。
4. 在标准的 RNN 结构中，隐层的神经元之间也是带有权值的，且权值共享。

5. 理论上，RNNs 能够对任何长度序列数据进行处理。但是在实践中，为了降低复杂度往往假设当前的状态只与之前某几个时刻状态相关，下图便是一个典型的 RNNs：

输入单元 (Input units): 输入集  $\{x_0, x_1, \dots, x_t, x_{t+1}, \dots\}$ ，  
 输出单元 (Output units): 输出集  $\{y_0, y_1, \dots, y_t, y_{t+1}, \dots\}$ ，  
 隐藏单元 (Hidden units): 输出集  $\{s_0, s_1, \dots, s_t, s_{t+1}, \dots\}$ 。

**图中信息传递特点：**

1. 一条单向流动的信息流是从输入单元到隐藏单元。
2. 一条单向流动的信息流从隐藏单元到输出单元。
3. 在某些情况下，RNNs 会打破后者的限制，引导信息从输出单元返回隐藏单元，这些被称为“Back Projections”。
4. 在某些情况下，隐藏层的输入还包括上一时刻隐藏层的状态，即隐藏层内的节点可以自连也可以互连。
5. 当前单元 (cell) 输出是由当前时刻输入和上一时刻隐藏层状态共同决定。

#### 4. 6.4 CNN 和 RNN 的区别？

类别	特点描述
相同点	1、传统神经网络的扩展。2、前向计算产生结果，反向计算模型更新。3、每层神经网络横向可以多个神经元共存，纵向可以有多层神经网络连接。
不同点	1、CNN 空间扩展，神经元与特征卷积；RNN 时间扩展，神经元与多个时间输出计算 2、RNN 可以用于描述时间上连续状态的输出，有记忆功能，CNN 用于静态输出

#### 5. 6.5 RNNs 和 FNNs 有什么区别？

1. 不同于传统的前馈神经网络 (FNNs)，RNNs 引入了定向循环，能够处理输入之间前后关联问题。
2. RNNs 可以记忆之前步骤的训练信息。**定向循环结构如下图所示：**

#### 6. 6.6 RNNs 训练和传统 ANN 训练异同点？

**相同点：**

1. RNNs 与传统 ANN 都使用 BP (Back Propagation) 误差反向传播算法。

**不同点：**

1. RNNs 网络参数 W,U,V 是共享的 (具体在本章 6.2 节中已介绍)，而传统神经网络各层参数间没有直接联系。

2. 对于 RNNs，在使用梯度下降算法中，每一步的输出不仅依赖当前步的网络，还依赖于之前若干步的网络状态。

### 7. 6.7 为什么 RNN 训练的时候 Loss 波动很大

由于 RNN 特有的 memory 会影响后期其他的 RNN 的特点，梯度时大时小，learning rate 没法个性化的调整，导致 RNN 在 train 的过程中，Loss 会震荡起伏，为了解决 RNN 的这个问题，在训练的时候，可以设置临界值，当梯度大于某个临界值，直接截断，用这个临界值作为梯度的大小，防止大幅震荡。

### 8. 6.8 标准 RNN 前向输出流程

以  $x$  表示输入， $h$  是隐层单元， $o$  是输出， $L$  为损失函数， $y$  为训练集标签。 $t$  表示  $t$  时刻的状态， $V, U, W$  是权值，同一类型的连接权值相同。以下图为例进行说明标准 RNN 的前向传播算法：

对于  $t$  时刻：

$$h^{(t)} = \phi(Ux^{(t)} + Wh^{(t-1)} + b)$$

其中  $\phi()$  为激活函数，一般会选择 tanh 函数， $b$  为偏置。

$t$  时刻的输出为：

$$o^{(t)} = Vh^{(t)} + c$$

模型的预测输出为：

$$\hat{y}^{(t)} = \sigma(o^{(t)})$$

其中  $\sigma$  为激活函数，通常 RNN 用于分类，故这里一般用 softmax 函数。

### 9. 6.9 BPTT 算法推导

BPTT (back-propagation through time) 算法是常用的训练 RNN 的方法，其本质还是 BP 算法，只不过 RNN 处理时间序列数据，所以要基于时间反向传播，故叫随时间反向传播。BPTT 的中心思想和 BP 算法相同，沿着需要优化的参数的负梯度方向不断寻找更优的点直至收敛。需要寻优的参数有三个，分别是  $U$ 、 $V$ 、 $W$ 。与 BP 算法不同的是，其中  $W$  和  $U$  两个参数的寻优过程需要追溯之前的历史数据，参数  $V$  相对简单只需关注目前，那么我们就来先求解参数  $V$  的偏导数。

$$\frac{\partial L^{(t)}}{\partial V} = \frac{\partial L^{(t)}}{\partial o^{(t)}} \cdot \frac{\partial o^{(t)}}{\partial V}$$

RNN 的损失也是会随着时间累加的，所以不能只求 t 时刻的偏导。

$$L = \sum_{t=1}^n L^{(t)}$$

$$\frac{\partial L}{\partial V} = \sum_{t=1}^n \frac{\partial L^{(t)}}{\partial o^{(t)}} \cdot \frac{\partial o^{(t)}}{\partial V}$$

W 和 U 的偏导的求解由于需要涉及到历史数据，其偏导求起来相对复杂。为了简化推导过程，我们假设只有三个时刻，那么在第三个时刻 L 对 W, L 对 U 的偏导数分别为：

$$\frac{\partial L^{(3)}}{\partial W} = \frac{\partial L^{(3)}}{\partial o^{(3)}} \frac{\partial o^{(3)}}{\partial h^{(3)}} \frac{\partial h^{(3)}}{\partial W} + \frac{\partial L^{(3)}}{\partial o^{(3)}} \frac{\partial o^{(3)}}{\partial h^{(3)}} \frac{\partial h^{(3)}}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial W} + \frac{\partial L^{(3)}}{\partial o^{(3)}} \frac{\partial o^{(3)}}{\partial h^{(3)}} \frac{\partial h^{(3)}}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial h^{(1)}} \frac{\partial h^{(1)}}{\partial W}$$

$$\frac{\partial L^{(3)}}{\partial U} = \frac{\partial L^{(3)}}{\partial o^{(3)}} \frac{\partial o^{(3)}}{\partial h^{(3)}} \frac{\partial h^{(3)}}{\partial U} + \frac{\partial L^{(3)}}{\partial o^{(3)}} \frac{\partial o^{(3)}}{\partial h^{(3)}} \frac{\partial h^{(3)}}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial U} + \frac{\partial L^{(3)}}{\partial o^{(3)}} \frac{\partial o^{(3)}}{\partial h^{(3)}} \frac{\partial h^{(3)}}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial h^{(1)}} \frac{\partial h^{(1)}}{\partial U}$$

可以观察到，在某个时刻的对 W 或是 U 的偏导数，需要追溯这个时刻之前所有时刻的信息。根据上面两个式子得出 L 在 t 时刻对 W 和 U 偏导数的通式：

$$\frac{\partial L^{(t)}}{\partial W} = \sum_{k=0}^t \frac{\partial L^{(t)}}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial h^{(t)}} \left( \prod_{j=k+1}^t \frac{\partial h^{(j)}}{\partial h^{(j-1)}} \right) \frac{\partial h^{(k)}}{\partial W}$$

$$\frac{\partial L^{(t)}}{\partial U} = \sum_{k=0}^t \frac{\partial L^{(t)}}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial h^{(t)}} \left( \prod_{j=k+1}^t \frac{\partial h^{(j)}}{\partial h^{(j-1)}} \right) \frac{\partial h^{(k)}}{\partial U}$$

整体的偏导公式就是将其按时刻再一一加起来。

### 10. 6.9 RNN 中为什么会出现梯度消失？

首先来看 tanh 函数的函数及导数图如下所示：

sigmoid 函数的函数及导数图如下所示：

从上图观察可知，sigmoid 函数的导数范围是 (0,0.25]，tanh 函数的导数范围是 (0,1]，他们的导数最大都不大于 1。

基于 6.8 中式 (9-10) 中的推导，RNN 的激活函数是嵌套在里面的，如果选择激活函数为 tanh 或 sigmoid，把激活函数放进去，拿出中间累乘的那部分可得：

$$\prod_{j=k+1}^t \frac{\partial h^j}{\partial h^{j-1}} = \prod_{j=k+1}^t \tanh' \cdot W_s$$

$$\prod_{j=k+1}^t \frac{\partial h^j}{\partial h^{j-1}} = \prod_{j=k+1}^t \text{sigmoid}' \cdot W_s$$

**梯度消失现象：**基于上式，会发现累乘会导致激活函数导数的累乘，如果取 tanh 或 sigmoid 函数作为激活函数的话，那么必然是一堆小数在做乘法，结果就是越乘越小。随着时间序列的不断深入，小数的累乘就会导致梯度越来越小直到接近于 0，这就是“梯度消失”现象。

实际使用中，会优先选择 tanh 函数，原因是 tanh 函数相对于 sigmoid 函数来说梯度较大，收敛速度更快且引起梯度消失更慢。

### 11. 6.10 如何解决 RNN 中的梯度消失问题？

上节描述的梯度消失是在无限的利用历史数据而造成，但是 RNN 的特点本来就是能利用历史数据获取更多的可利用信息，解决 RNN 中的梯度消失方法主要有：

1、选取更好的激活函数，如 Relu 激活函数。ReLU 函数的左侧导数为 0，右侧导数恒为 1，这就避免了“梯度消失”的发生。但恒为 1 的导数容易导致“梯度爆炸”，但设定合适的阈值可以解决这个问题。

2、加入 BN 层，其优点包括可加速收敛、控制过拟合，可以少用或不用 Dropout 和正则、降低网络对初始化权重不敏感，且能允许使用较大的学习率等。

2、改变传播结构，LSTM 结构可以有效解决这个问题。下面将介绍 LSTM 相关内容。

## 12. 6.11 LSTM

**12.1. 6.11.1 LSTM 的产生原因.** RNN 在处理长期依赖（时间序列上距离较远的节点）时会遇到巨大的困难，因为计算距离较远的节点之间的联系时会涉及雅可比矩阵的多次相乘，会造成梯度消失或者梯度膨胀的现象。为了解决该问题，研究人员提出了许多解决办法，例如 ESN (Echo State Network)，增加有漏单元 (Leaky Units) 等等。其中最成功应用最广泛的就是门限 RNN (Gated RNN)，而 LSTM 就是门限 RNN 中最著名的一种。有漏单元通过设计连接间的权重系数，从而允许 RNN 累积距离较远节点间的长期联系；而门限 RNN 则泛化了这样的思想，允许在不同时刻改变该系数，且允许网络忘记当前已经累积的信息。

**12.2. 6.11.2 图解标准 RNN 和 LSTM 的区别.** 所有 RNN 都具有一种重复神经网络模块的链式的形式。在标准的 RNN 中，这个重复的模块只有一个非常简单的结构，例如一个 tanh 层，如下图所示：

LSTM 同样是这样的结构，但是重复的模块拥有一个不同的结构。不同于单一神经网络层，这里有四个，以一种非常特殊的方式进行交互。

注：上图图标具体含义如下所示：

上图中，每一条黑线传输着一个向量，从一个节点的输出到其他节点的输入。粉色的圈代表 pointwise 的操作，诸如向量的和，而黄色的矩阵就是学到的神经网络层。合在一起的线表示向量的连接，分开的线表示内容被复制，然后分发到不同的位置。

**12.3. 6.11.3 LSTM 核心思想图解.** LSTM 的关键就是细胞状态，水平线在图上方贯穿运行。细胞状态类似于传送带。直接在整个链上运行，只有一些少量的线性交互。信息在上面流传保持不变会很容易。示意图如下所示：

LSTM 有通过精心设计的称作为“门”的结构来去除或者增加信息到细胞状态的能力。门是一种让信息选择式通过的方法。他们包含一个 sigmoid 神经网络层和一个 pointwise 乘法操作。示意图如下：

LSTM 拥有三个门，分别是忘记层门，输入层门和输出层门，来保护和控制细胞状态。

### 忘记层门

作用对象：细胞状态。

作用：将细胞状态中的信息选择性的遗忘。

操作步骤：该门会读取  $h_{t-1}$  和  $x_t$ ，输出一个在 0 到 1 之间的数值给每个在细胞状态  $C_{t-1}$  中的数字。1 表示“完全保留”，0 表示“完全舍弃”。示意图如下：

### 输入层门

作用对象：细胞状态

作用：将新的信息选择性的记录到细胞状态中。

操作步骤：

步骤一，sigmoid 层称“输入门层”决定什么值我们将要更新。

步骤二，tanh 层创建一个新的候选值向量  $\tilde{C}_t$  加入到状态中。其示意图如下：

步骤三：将  $c_{t-1}$  更新为  $c_t$ 。将旧状态与  $f_t$  相乘，丢弃掉我们确定需要丢弃的信息。接着加上  $i_t * \tilde{C}_t$  得到新的候选值，根据我们决定更新每个状态的程度进行变化。其示意图如下：

### 输出层门作用对象：隐层 $h_t$

作用：确定输出什么值。

操作步骤：

步骤一：通过 sigmoid 层来确定细胞状态的哪个部分将输出。

步骤二：把细胞状态通过 tanh 进行处理，并将它和 sigmoid 门的输出相乘，最终我们仅会输出我们确定输出的那部分。

其示意图如下所示：

### 12.4. 6.11.4 LSTM 流行的变体. 增加 peephole 连接

在正常的 LSTM 结构中，Gers F A 等人提出增加 peephole 连接，可以门层接受细胞状态的输入。示意图如下所示：

#### 对忘记门和输入门进行同时确定

不同于之前是分开确定什么忘记和需要添加什么新的信息，这里是一同做出决定。示意图如下所示：

### Gated Recurrent Unit

由 Kyunghyun Cho 等人提出的 Gated Recurrent Unit (GRU)，其将忘记门和输入门合成了一个单一的更新门，同样还混合了细胞状态和隐藏状态，和其他一些改动。其示意图如下：最终的模型比标准的 LSTM 模型要简单，也是非常流行的变体。

### 13. 6.12 LSTMs 与 GRUs 的区别

LSTMs 与 GRUs 的区别如图所示：

从上图可以看出，二者结构十分相似，**不同在于**：

1. new memory 都是根据之前 state 及 input 进行计算，但是 GRUs 中有一个 reset gate 控制之前 state 的进入量，而在 LSTMs 里没有类似 gate；
2. 产生新的 state 的方式不同，LSTMs 有两个不同的 gate，分别是 forget gate (f gate) 和 input gate(i gate)，而 GRUs 只有一种 update gate(z gate)；
3. LSTMs 对新产生的 state 可以通过 output gate(o gate) 进行调节，而 GRUs 对输出无任何调节。

### 14. 6.13 RNNs 在 NLP 中典型应用？

#### (1) 语言模型与文本生成 (Language Modeling and Generating Text)

给定一组单词序列，需要根据前面前单词预测每个单词出现的可能性。语言模型能够评估某个语句正确的可能性，可能性越大，语句越正确。另一种应用便是使用生成模型预测下一个单词的出现概率，从而利用输出概率的采样生成新的文本。

#### (2) 机器翻译 (Machine Translation)

机器翻译是将一种源语言语句变成意思相同的另一种源语言语句，如将英语语句变成同样意思的中文语句。与语言模型关键的区别在于，需要将源语言语句序列输入后，才进行输出，即输出第一个单词时，便需要从完整的输入序列中进行获取。

#### (3) 语音识别 (Speech Recognition)

语音识别是指给定一段声波的声音信号，预测该声波对应的某种指定源语言语句以及计算该语句的概率值。

#### (4) 图像描述生成 (Generating Image Descriptions)

同卷积神经网络一样，RNNs 已经在对无标图像描述自动生成中得到应用。CNNs 与 RNNs 结合也被应用于图像描述自动生成。

### 15. 6.13 常见的 RNNs 扩展和改进模型

#### 15.1. 6.13.1 Simple RNNs(SRNs).

1. SRNs 是一个三层网络，其在隐藏层增加了上下文单元。下图中的  $y$  是隐藏层， $u$  是上下文单元。上下文单元节点与隐藏层中节点的连接是固定的，并且权值也是固定的。上下文节点与隐藏层节点一一对应，并且值是确定的。
2. 在每一步中，使用标准的前向反馈进行传播，然后使用学习算法进行学习。上下文每一个节点保存其连接隐藏层节点上一步输出，即保存上文，并作用于当前步对应的隐藏层节点状态，即隐藏层的输入由输入层的输出与上一步的自身状态所决定。因此 SRNs 能够解决标准多层感知机 (MLP) 无法解决的对序列数据进行预测的问题。SRNs 网络结构如下图所示：

**15.2. 6.13.2 Bidirectional RNNs.** Bidirectional RNNs(双向网络) 将两层 RNNs 叠加在一起，当前时刻输出 (第  $t$  步的输出) 不仅仅与之前序列有关，还与之后序列有关。例如：为了预测一个语句中的缺失词语，就需要该词汇的上下文信息。Bidirectional RNNs 是一个相对较简单的 RNNs，是由两个 RNNs 上下叠加在一起组成的。输出由前向 RNNs 和后向 RNNs 共同决定。如下图所示：

**15.3. 6.13.3 Deep RNNs.** Deep RNNs 与 Bidirectional RNNs 相似，其也是又多层 RNNs 叠加，因此每一步的输入有了多层网络。该网络具有更强大的表达与学习能力，但是复杂性也随之提高，同时需要更多的训练数据。Deep RNNs 的结构如下图所示：

#### 15.4. 6.13.4 Echo State Networks (ESNs). ESNs 特点：

1. 它的核心结构为一个随机生成、且保持不变的储备池 (Reservoir)。储备池是大规模随机生成稀疏连接 (SD 通常保持  $1\% \sim 5\%$ ，SD 表示储备池中互相连接的神经元占总神经元个数  $N$  的比例) 的循环结构；
2. 从储备池到输出层的权值矩阵是唯一需要调整的部分；
3. 简单的线性回归便能够完成网络训练；

#### ESNs 基本思想：

使用大规模随机连接的循环网络取代经典神经网络中的中间层，从而简化网络的训练过程。网络中的参数包括：(1)  $W$  - 储备池中节点间连接权值矩阵；(2)  $W_{in}$  - 输入层到储备池之间连接权值矩阵，表明储备池中的神经元之间是相互连接；(3)  $W_{back}$  - 输出层到储备池之间的反馈连接权值矩阵，表明储备池会有输出层来的反馈；(4)  $W_{out}$  - 输入层、储备池、输出层到输出层的连接权值矩阵，表明输出层不仅与储备池连接，还与输入层和自己连接。(5)  $W_{outbias}$  - 输出层的偏置项。

ESNs 的结构如下图所示：

**15.5. 6.13.4 Gated Recurrent Unit Recurrent Neural Networks.** GRUs 是一般的 RNNs 的变型版本，其主要是从以下两个方面进行改进。

1. 以语句为例，序列中不同单词处的数据对当前隐藏层状态的影响不同，越前面的影响越小，即每个之前状态对当前的影响进行了距离加权，距离越远，权值越小。
2. 在产生误差 error 时，其可能是由之前某一个或者几个单词共同造成，所以应当对对应的单词 weight 进行更新。GRUs 的结构如下图所示。GRUs 首先根据当前输入单词向量 word vector 以及前一个隐藏层状态 hidden state 计算出 update gate 和 reset gate。再根据 reset gate、当前 word vector 以及前一个 hidden state 计算新的记忆单元内容 (new memory content)。当 reset gate 为 1 的时候，new memory content 忽略之前所有 memory content，最终的 memory 是由之前的 hidden state 与 new memory content 一起决定。

#### 15.6. 6.13.5 Bidirectional LSTMs.

1. 与 bidirectional RNNs 类似，bidirectional LSTMs 有两层 LSTMs。一层处理过去的训练信息，另一层处理将来的训练信息。
2. 在 bidirectional LSTMs 中，通过前向 LSTMs 获得前向隐藏状态，后向 LSTMs 获得后向隐藏状态，当前隐藏状态是前向隐藏状态与后向隐藏状态的组合。

#### 15.7. 6.13.6 Stacked LSTMs.

1. 与 deep rnns 类似，stacked LSTMs 通过将多层 LSTMs 叠加起来得到一个更加复杂的模型。
2. 不同于 bidirectional LSTMs，stacked LSTMs 只利用之前步骤的训练信息。

**15.8. 6.13.7 Clockwork RNNs(CW-RNNs).** CW-RNNs 是 RNNs 的改良版本，其使用时钟频率来驱动。它将隐藏层分为几个块 (组，Group/Module)，每一组按照自己规定的时钟频率对输入进行处理。为了降低 RNNs 的复杂度，CW-RNNs 减少了参数数量，并且提高了网络性能，加速网络训练。CW-RNNs 通过不同隐藏层模块在不同时钟频率下工作来解决长时依赖问题。将时钟时间进行离散化，不同的隐藏层组将在不同时刻进行工作。因此，所有的隐藏层组在每一步不会全部同时工作，这样便会加快网络的训练。并且，时钟周期小组的神经元不会连接到时钟周期大组的神经元，只允许周期大的神经元连接到周期小的 (组与组之间的连接以及信息传递是有向的)。周期大的速度慢，周期小的速度快，因此是速度慢的神经元连速度快的神经元，反之则不成立。

CW-RNNs 与 SRNs 网络结构类似，也包括输入层 (Input)、隐藏层 (Hidden)、输出层 (Output)，它们之间存在前向连接，输入层到隐藏层连接，隐藏层到输出层连接。但是与 SRN 不同的是，隐藏层中的神经元会被划分为若干个组，设为  $g$ ，每一组中的神经元个数相同，设为  $k$ ，并为每一个组分配一个时钟周期  $T_i \in \{T_1, T_2, \dots, T_g\}$ ，每一组中的所有神经元都是全连接，但是组  $j$  到组  $i$  的循环连接则需要满足  $T_j > T_i$ 。如下图所示，将这些组按照时钟周期递增从左到右进行排序，即  $T_1 < T_2 < \dots < T_g$ ，那么连接便是从右到左。例如：隐藏层共有 256 个节点，分为四组，周期分别是 [1,2,4,8]，那么每个隐藏层组  $256/4=64$  个节点，第一组隐藏层

与隐藏层的连接矩阵为  $64 \times 6464 \times 12864 \times (3 \times 64) = 64 \times 19264 \times (4 \times 64) = 64 \times 256$  矩阵。这就解释了上一段中速度慢的组连接到速度快的组，反之则不成立。

CW-RNNs 的网络结构如下图所示：

#### 15.9. 6.13.8 CNN-LSTMs.

1. 为了同时利用 CNN 以及 LSTMs 的优点，CNN-LSTMs 被提出。在该模型中，CNN 用于提取对象特征，LSTMs 用于预测。CNN 由于卷积特性，其能够快速而且准确地捕捉对象特征。LSTMs 的优点在于能够捕捉数据间的长时依赖性。





## CHAPTER 7

# 生成对抗网络

## 1. 7.1 GAN 基本概念

**1.1. 如何通俗理解 GAN ?.** 生成对抗网络 (GAN, Generative adversarial network) 自从 2014 年被 Ian Goodfellow 提出以来，掀起了一股研究热潮。GAN 由生成器和判别器组成，生成器负责生成样本，判别器负责判断生成器生成的样本是否为真。生成器要尽可能迷惑判别器，而判别器要尽可能区分生成器生成的样本和真实样本。

在 GAN 的原作 [1] 中，作者将生成器比喻为印假钞票的犯罪分子，判别器则类比为警察。犯罪分子努力让钞票看起来逼真，警察则不断提升对于假钞的辨识能力。二者互相博弈，随着时间的进行，都会越来越强。那么类比于图像生成任务，生成器不断生成尽可能逼真的假图像。判别器则判断图像是否是真实的图像，还是生成的图像，二者不断博弈优化。最终生成器生成的图像使得判别器完全无法判别真假。

**1.2. 7.1.2 GAN 的形式化表达.** 上述例子只是简要介绍了一下 GAN 的思想，下面对于 GAN 做一个形式化的，更加具体的定义。通常情况下，无论是生成器还是判别器，我们都可以用神经网络来实现。那么，我们可以把通俗化的定义用下面这个模型来表示：

上述模型左边是生成器  $G$ ，其输入是  $z$ ，对于原始的 GAN， $z$  是由高斯分布随机采样得到的噪声。噪声  $z$  通过生成器得到了生成的假样本。

生成的假样本与真实样本放到一起，被随机抽取送入到判别器  $D$ ，由判别器去区分输入的样本是生成的假样本还是真实的样本。整个过程简单明了，生成对抗网络中的“生成对抗”主要体现在生成器和判别器之间的对抗。

**1.3. 7.1.3 GAN 的目标函数是什么 ?.** 对于上述神经网络模型，如果想要学习其参数，首先需要一个目标函数。GAN 的目标函数定义如下：

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

这个目标函数可以分为两个部分来理解：

第一部分：判别器的优化通过  $\max_D V(D, G)$  实现， $V(D, G)$  为判别器的目标函数，其第一项  $\mathbb{E}_{x \sim p_{data}(x)}[\log D(x)]$  表示对于从真实数据分布中采用的样本，其被判别器判定为真实样

本概率的数学期望。对于真实数据分布中采样的样本，其预测为正样本的概率当然是越接近 1 越好。因此希望最大化这一项。第二项  $E_{z \sim p_z(z)}[\log(1 - D(G(z)))]$  表示：对于从噪声  $P_z(z)$  分布当中采样得到的样本，经过生成器生成之后得到的生成图片，然后送入判别器，其预测概率的负对数的期望，这个值自然是越大越好，这个值越大，越接近 0，也就代表判别器越好。

第二部分：生成器的优化通过  $\min_G \max_D V(D, G)$  来实现。注意，生成器的目标不是  $\min_G V(D, G)$ ，即生成器不是最小化判别器的目标函数，二是最小化判别器目标函数的最大值，判别器目标函数的最大值代表的是真实数据分布与生成数据分布的 JS 散度（详情可以参阅附录的推导），JS 散度可以度量分布的相似性，两个分布越接近，JS 散度越小。

**1.4. 7.1.4 GAN 的目标函数和交叉熵有什么区别？** 判别器目标函数写成离散形式即为：

$$V(D, G) = -\frac{1}{m} \sum_{i=1}^{i=m} \log D(x^i) - \frac{1}{m} \sum_{i=1}^{i=m} \log(1 - D(\tilde{x}^i))$$

可以看出，这个目标函数和交叉熵是一致的，即判别器的目标是 **最小化交叉熵损失**，生成器的目标是 **最小化生成数据分布和真实数据分布的 JS 散度**。

[1]: Goodfellow, Ian, et al. “Generative adversarial nets.” Advances in neural information processing systems. 2014.

**1.5. 7.1.5 GAN 的 Loss 为什么降不下去？** 对于很多 GAN 的初学者在实践过程中可能会纳闷，为什么 GAN 的 Loss 一直降不下去。GAN 到底什么时候才算收敛？其实，作为一个训练良好的 GAN，其 Loss 就是降不下去的。衡量 GAN 是否训练好了，只能由人肉眼去看生成的图片质量是否好。不过，对于没有一个很好的评价是否收敛指标的问题，也有许多学者做了一些研究，后文提及的 WGAN 就提出了一种新的 Loss 设计方式，较好的解决了难以判断收敛性的问题。下面我们分析一下 GAN 的 Loss 为什么降不下去？对于判别器而言，GAN 的 Loss 如下：

$$\min_G \max_D V(D, G) = E_{x \sim p_{data}(x)}[\log D(x)] + E_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

从  $\min_G \max_D V(D, G)$  可以看出，生成器和判别器的目的相反，也就是说两个生成器网络和判别器网络互为对抗，此消彼长。不可能 Loss 一直降到一个收敛的状态。

- 对于生成器，其 Loss 下降快，很有可能是判别器太弱，导致生成器很轻易的就“愚弄”了判别器。
- 对于判别器，其 Loss 下降快，意味着判别器很强，判别器很强则说明生成器生成的图像不够逼真，才使得判别器轻易判别，导致 Loss 下降很快。

也就是说，无论是判别器，还是生成器。loss 的高低不能代表生成器的好坏。一个好的 GAN 网络，其 GAN Loss 往往是不断波动的。

看到这里可能有点让人绝望，似乎判断模型是否收敛就只能看生成的图像质量了。实际上，后文探讨的 WGAN，提出了一种新的 loss 度量方式，让我们可以通过一定的手段来判断模型是否收敛。

**1.6. 7.1.6 生成式模型、判别式模型的区别？** 对于机器学习模型，我们可以根据模型对数据的建模方式将模型分为两大类，生成式模型和判别式模型。如果我们要训练一个关于猫狗分类的模型，对于判别式模型，只需要学习二者差异即可。比如说猫的体型会比狗小一点。而生成式模型则不一样，需要学习猫张什么样，狗张什么样。有了二者的长相以后，再根据长相去区分。具体而言：

- 生成式模型：由数据学习联合概率分布  $P(X, Y)$ ，然后由  $P(Y|X) = P(X, Y)/P(X)$  求出概率分布  $P(Y|X)$  作为预测的模型。该方法表示了给定输入  $X$  与产生输出  $Y$  的生成关系
- 判别式模型：由数据直接学习决策函数  $Y=f(X)$  或条件概率分布  $P(Y|X)$  作为预测模型，即判别模型。判别方法关心的是对于给定的输入  $X$ ，应该预测什么样的输出  $Y$ 。

对于上述两种模型，从文字上理解起来似乎不太直观。我们举个例子来阐述一下，对于性别分类问题，分别用不同的模型来做：

1) 如果用生成式模型：可以训练一个模型，学习输入人的特征  $X$  和性别  $Y$  的关系。比如现在有下面一批数据：

Y (性别)	0	1
X (特征)	0	1/4 3/4
	1	3/4 1/4

这个数据可以统计得到，即统计人的特征  $X=0,1,\dots$  的时候，其类别为  $Y=0,1$  的概率。统计得到上述联合概率分布  $P(X, Y)$  后，可以学习一个模型，比如让二维高斯分布去拟合上述数据，这样就学习到了  $X, Y$  的联合分布。在预测时，如果我们希望给一个输入特征  $X$ ，预测其类别，则需要通过贝叶斯公式得到条件概率分布才能进行推断：

$$P(Y|X) = \frac{P(X, Y)}{P(X)} = \frac{P(X, Y)}{P(X|Y)P(Y)}$$

2) 如果用判别式模型：可以训练一个模型，输入人的特征  $X$ ，这些特征包括人的五官，穿衣风格，发型等。输出则是对于性别的判断概率，这个概率服从一个分布，分布的取值只有两个，要么男，要么女，记这个分布为  $Y$ 。这个过程学习了一个条件概率分布  $P(Y|X)$ ，即输入特征  $X$  的分布已知条件下， $Y$  的概率分布。

显然，从上面的分析可以看出。判别式模型似乎要方便很多，因为生成式模型要学习一个  $X, Y$  的联合分布往往需要很多数据，而判别式模型需要的数据则相对少，因为判别式模型更关注输入特征的差异性。不过生成式既然使用了更多数据来生成联合分布，自然也能够提供更多的信息，现在有一个样本  $(X, Y)$ ，其联合概率  $P(X, Y)$  经过计算特别小，那么可以认为这个样本是异常样本。这种模型可以用来做 outlier detection。

**1.7. 7.1.7 什么是 mode collapsing?** 某个模式 (mode) 出现大量重复样本，例如：上图左侧的蓝色五角星表示真实样本空间，黄色的是生成的。生成样本缺乏多样性，存在大量重复。比如上图右侧中，红框里面人物反复出现。

### 1.8. 7.1.8 如何解决 mode collapsing ?.

#### 方法一：针对目标函数的改进方法

为了避免前面提到的由于优化 maxmin 导致 mode 跳来跳去的问题，UnrolledGAN 采用修改生成器 loss 来解决。具体而言，UnrolledGAN 在更新生成器时更新  $k$  次生成器，参考的 Loss 不是某一次的 loss，是判别器后面  $k$  次迭代的 loss。注意，判别器后面  $k$  次迭代不更新自己的参数，只计算 loss 用于更新生成器。这种方式使得生成器考虑到了后面  $k$  次判别器的变化情况，避免在不同 mode 之间切换导致的模式崩溃问题。此处务必和迭代  $k$  次生成器，然后迭代 1 次判别器区分开 [8]。DRAGAN 则引入博奔论中的无后悔算法，改造其 loss 以解决 mode collapse 问题 [9]。前文所述的 EBGAN 则是加入 VAE 的重构误差以解决 mode collapse。

#### 方法二：针对网络结构的改进方法

Multi agent diverse GAN(MAD-GAN) 采用多个生成器，一个判别器以保障样本生成的多样性。具体结构如下：

相比于普通 GAN，多了几个生成器，且在 loss 设计的时候，加入一个正则项。正则项使用余弦距离惩罚三个生成器生成样本的一致性。

MRGAN 则添加了一个判别器来惩罚生成样本的 mode collapse 问题。具体结构如下：

输入样本  $x$  通过一个 Encoder 编码为隐变量  $E(x)$ ，然后隐变量被 Generator 重构，训练时，Loss 有三个。 $D_M$  和  $R$  (重构误差) 用于指导生成 real-like 的样本。而  $D_D$  则对  $E(x)$  和  $z$  生成的样本进行判别，显然二者生成样本都是 fake samples，所以这个判别器主要用于判断生成的样本是否具有多样性，即是否出现 mode collapse。

#### 方法三：Mini-batch Discrimination

Mini-batch discrimination 在判别器的中间层建立一个 mini-batch layer 用于计算基于 L1 距离的样本统计量，通过建立该统计量，实现了一个 batch 内某个样本与其他样本有多接近。这个信息可以被判别器利用到，从而甄别出哪些缺乏多样性的样本。对生成器而言，则要试图生成具有多样性的样本。

## 2. 7.2 GAN 的生成能力评价

**2.1. 7.2.1 如何客观评价 GAN 的生成能力？** 最常见评价 GAN 的方法就是主观评价。主观评价需要花费大量人力物力，且存在以下问题：

- 评价带有主管色彩，有些 bad case 没看到很容易造成误判
- 如果一个 GAN 过拟合了，那么生成的样本会非常真实，人类主观评价得分会非常高，可是这并不是一个好的 GAN。

因此，就有许多学者提出了 GAN 的客观评价方法。

**2.2. 7.2.2 Inception Score.** 对于一个在 ImageNet 训练良好的 GAN，其生成的样本丢给 Inception 网络进行测试的时候，得到的判别概率应该具有如下特性： - 对于同一个类别的图片，其输出的概率分布应该趋向于一个脉冲分布。可以保证生成样本的准确性。 - 对于所有类别，其输出的概率分布应该趋向于一个均匀分布，这样才不会出现 mode dropping 等，可以保证生成样本的多样性。

因此，可以设计如下指标：

$$IS(P_g) = e^{E_{x \sim P_g}[KL(p_M(y|x) \| p_M(y))]} GAN$$

$p_M(y|x)p_M(y)KL$  Inception Score Inception Score ISM 根据前面分析，如果是一个训练良好的 GAN， $p_M(y|x)$  趋近于脉冲分布， $p_M(y)$  趋近于均匀分布。二者 KL 散度会很大。Inception Score 自然就高。实际实验表明，Inception Score 和人的主观判别趋向一致。IS 的计算没有用到真实数据，具体值取决于模型 M 的选择。

**特点：**可以一定程度上衡量生成样本的多样性和准确性，但是无法检测过拟合。Mode Score 也是如此。不推荐在和 ImageNet 数据集差别比较大的数据上使用。

**2.3. 7.2.3 Mode Score.** Mode Score 作为 Inception Score 的改进版本，添加了关于生成样本和真实样本预测的概率分布相似性度量一项。具体公式如下：

$$MS(P_g) = e^{E_{x \sim P_g}[KL(p_M(y|x) \| p_M(y)) - KL(p_M(y) \| p_M(y^*))]}$$

**2.4. 7.2.4 Kernel MMD (Maximum Mean Discrepancy).** 计算公式如下：

$$MMD^2(P_r, P_g) = E_{x_r \sim P_r, x_g \sim P_g} [\|\Sigma_{i=1}^{n1} k(x_r) - \Sigma_{i=1}^{n2} k(x_g)\|]$$

对于 Kernel MMD 值的计算，首先需要选择一个核函数  $k$ ，这个核函数把样本映射到再生希尔伯特空间 (Reproducing Kernel Hilbert Space, RKHS)，RKHS 相比于欧几里得空间有许多优点，对于函数内积的计算是完备的。将上述公式展开即可得到下面的计算公式：

$$MMD^2(P_r, P_g) = E_{x_r, x_r' \sim P_r, x_g, x_g' \sim P_g} [k(x_r, x_r') - 2k(x_r, x_g) + k(x_g, x_g')]$$

MMD 值越小，两个分布越接近。

**特点：**可以一定程度上衡量模型生成图像的优劣性，计算代价小。推荐使用。

**2.5. 7.2.5 Wasserstein distance.** Wasserstein distance 在最优传输问题中通常也叫做推土机距离。这个距离的介绍在 WGAN 中有详细讨论。公式如下：

$$WD(P_r, P_g) = \min_{\omega \in \mathbb{R}^{m \times n}} \sum_{i=1}^n \sum_{j=1}^m \omega_{ij} d(x_i^r, x_j^g)$$

$$s.t. \sum_{i=1}^n w_{i,j} = p_r(x_i^r), \forall i; \sum_{j=1}^m w_{i,j} = p_g(x_j^g), \forall j$$

Wasserstein distance 可以衡量两个分布之间的相似性。距离越小，分布越相似。

**特点：**如果特征空间选择合适，会有一定的效果。但是计算复杂度为  $O(n^3)$  太高

**2.6. 7.2.6 Fréchet Inception Distance (FID).** FID 距离计算真实样本，生成样本在特征空间之间的距离。首先利用 Inception 网络来提取特征，然后使用高斯模型对特征空间进行建模。根据高斯模型的均值和协方差来进行距离计算。具体公式如下：

$$FID(\mathbb{P}_r, \mathbb{P}_g) = \|\mu_r - \mu_g\| + Tr(C_r + C_g - 2(C_r C_g)^{1/2})$$

$\mu, C$  分别代表协方差和均值。

$\mu, C$  分别代表协方差和均值。

**特点：**尽管只计算了特征空间的前两阶矩，但是鲁棒，且计算高效。

**2.7. 7.2.7 1-Nearest Neighbor classifier.** 使用留一法，结合 1-NN 分类器（别的也行）计算真实图片，生成图像的精度。如果二者接近，则精度接近 50%，否则接近 0%。对于 GAN 的评价问题，作者分别用正样本的分类精度，生成样本的分类精度去衡量生成样本的真实性，多样性。- 对于真实样本  $x_r$ ，进行 1-NN 分类的时候，如果生成的样本越真实。则真实样本空间  $\mathbb{R}$  将被生成的样本  $x_g$  包围。那么  $x_r$  的精度会很低。- 对于生成的样本  $x_g$ ，进行 1-NN 分类的时候，如果生成的样本多样性不足。由于生成的样本聚在几个 mode，则  $x_g$  很容易就和  $x_r$  区分，导致精度会很高。

**特点：**理想的度量指标，且可以检测过拟合。

**2.8. 7.2.8 其他评价方法.** AIS, KDE 方法也可以用于评价 GAN，但这些方法不是 model agnostic metrics。也就是说，这些评价指标的计算无法只利用：生成的样本，真实样本来计算。

### 3. 7.3 其他常见的生成式模型有哪些？

**3.1. 7.3.1 什么是自回归模型：pixelRNN 与 pixelCNN ?.** 自回归模型通过对图像数据的概率分布  $p_{data}(x)$  进行显式建模，并利用极大似然估计优化模型。具体如下：

$$p_{data}(x) = \prod_{i=1}^n p(x_i | x_1, x_2, \dots, x_{i-1})$$

上述公式很好理解，给定  $x_1, x_2, \dots, x_{i-1}$  条件下，所有  $p(x_i)$  的概率乘起来就是图像数据的分布。如果使用 RNN 对上述依然关系建模，就是 pixelRNN。如果使用 CNN，则是 pixelCNN。具体如下 [5]：

显然，不论是对于 pixelCNN 还是 pixelRNN，由于其像素值是一个个生成的，速度会很慢。语音领域大火的 WaveNet 就是一个典型的自回归模型。

**3.2. 7.3.2 什么是 VAE ?.** PixelCNN/RNN 定义了一个易于处理的密度函数，我们可以直接优化训练数据的似然；对于变分自编码器我们将定义一个不易处理的密度函数，通过附加的隐变量  $z$  对密度函数进行建模。VAE 原理图如下 [6]：

在 VAE 中，真实样本  $X$  通过神经网络计算出均值方差（假设隐变量服从正太分布），然后通过采样得到采样变量  $Z$  并进行重构。VAE 和 GAN 均是学习了隐变量  $z$  到真实数据分布的映射。但是和 GAN 不同的是：

- GAN 的思路比较粗暴，使用一个判别器去度量分布转换模块（即生成器）生成分布与真实数据分布的距离。
- VAE 则没有那么直观，VAE 通过约束隐变量  $z$  服从标准正太分布以及重构数据实现了分布转换映射  $X = G(z)$

### 生成式模型对比

- 自回归模型通过对概率分布显式建模来生成数据
- VAE 和 GAN 均是：假设隐变量  $z$  服从某种分布，并学习一个映射  $X = G(z)$ ，实现了隐变量分布  $z$  与真实数据分布  $p_{data}(x)$  的转换。
- GAN 使用判别器去度量映射  $X = G(z)$  的优劣，而 VAE 通过隐变量  $z$  与标准正太分布的 KL 散度和重构误差去度量。

## 4. 7.4 GAN 的改进与优化

**4.1. 7.4.1 如何生成指定类型的图像——条件 GAN.** 条件生成对抗网络 (CGAN, Conditional Generative Adversarial Networks) 作为一个 GAN 的改进，其一定程度上解决了 GAN 生成结果的不确定性。如果在 Mnist 数据集上训练原始 GAN，GAN 生成的图像是完全不确定的，具体生成的是数字 1，还是 2，还是几，根本不可控。为了让生成的数字可控，我们可以把数据集做一个切分，把数字 0~9 的数据集分别拆分开训练 9 个模型，不过这样太麻烦了，也不现实。因为数据集拆分不仅仅是分类麻烦，更主要在于，每一个类别的样本少，拿去训练 GAN 很有可能导致欠拟合。因此，CGAN 就应运而生了。我们先看一下 CGAN 的网络结构：从网络结构图可以看到，对于生成器 Generator，其输入不仅仅是随机噪声的采样  $z$ ，还有欲生成图像的标签信息。比如对于 mnist 数据生成，就是一个 one-hot 向量，某一维度为 1 则表示生成某个数字的图片。同样地，判别器的输入也包括样本的标签。这样就使得判别器和生成

器可以学习到样本和标签之间的联系。Loss 如下：

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x|y)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z|y)))]$$

Loss 设计和原始 GAN 基本一致，只不过生成器，判别器的输入数据是一个条件分布。在具体编程实现时只需要对随机噪声采样 z 和输入条件 y 做一个级联即可。

**4.2. 7.4.2 CNN 与 GAN——DCGAN.** 前面我们聊的 GAN 都是基于简单的神经网络构建的。可是对于视觉问题，如果使用原始的基于 DNN 的 GAN，则会出现许多问题。如果输入 GAN 的随机噪声为 100 维的随机噪声，输出图像为 256x256 大小。也就是说，要将 100 维的信息映射为 65536 维。如果单纯用 DNN 来实现，那么整个模型参数会非常巨大，而且学习难度很大（低维度映射到高维度需要添加许多信息）。因此，DCGAN 就出现了。具体而言，DCGAN 将传统 GAN 的生成器，判别器均采用 GAN 实现，且使用了一下 tricks：

- 将 pooling 层 convolutions 替代，其中，在 discriminator 上用 strided convolutions 替代，在 generator 上用 fractional-strided convolutions 替代。
- 在 generator 和 discriminator 上都使用 batchnorm。
- 移除全连接层，global pooling 增加了模型的稳定性，但伤害了收敛速度。
- 在 generator 的除了输出层外的所有层使用 ReLU，输出层采用 tanh。
- 在 discriminator 的所有层上使用 LeakyReLU。

网络结构图如下：

**4.3. 7.4.3 如何理解 GAN 中的输入随机噪声？** 为了了解输入随机噪声每一个维度代表的含义，作者做了一个非常有趣的工作。即在隐空间上，假设知道哪几个变量控制着某个物体，那么僵这几个变量挡住是不是就可以将生成图片中的某个物体消失？论文中的实验是这样的：首先，生成 150 张图片，包括有窗户的和没有窗户的，然后使用一个逻辑斯底回归函数来进行分类，对于权重不为 0 的特征，认为它和窗户有关。将其挡住，得到新的生成图片，结果如下：此外，将几个输入噪声进行算数运算，可以得到语义上进行算数运算的非常有趣的結果。类似于 word2vec。

**4.4. 7.4.4 GAN 为什么容易训练崩溃？** 所谓 GAN 的训练崩溃，指的是训练过程中，生成器和判别器存在一方压倒另一方的情况。GAN 原始判别器的 Loss 在判别器达到最优的时候，等价于最小化生成分布与真实分布之间的 JS 散度，由于随机生成分布很难与真实分布有不可忽略的重叠以及 JS 散度的突变特性，使得生成器面临梯度消失的问题；可是如果不把判别器训练到最优，那么生成器优化的目标就失去了意义。因此需要我们小心的平衡二者，要把判别器训练的不好也不坏才行。否则就会出现训练崩溃，得不到想要的结果。

**4.5. 7.4.5 WGAN 如何解决训练崩溃问题？** WGAN 作者提出了使用 Wasserstein 距离，以解决 GAN 网络训练过程难以判断收敛性的问题。Wasserstein 距离定义如下：

$$L = \mathbb{E}_{x \sim p_{data}(x)}[f_w(x)] - \mathbb{E}_{x \sim p_g(x)}[f_w(x)]$$

通过最小化 Wasserstein 距离，得到了 WGAN 的 Loss：

- WGAN 生成器 Loss:  $-\mathbb{E}_{x \sim p_g(x)}[f_w(x)]$
- WGAN 判别器 Loss:  $L = -\mathbb{E}_{x \sim p_{data}(x)}[f_w(x)] + \mathbb{E}_{x \sim p_g(x)}[f_w(x)]$

从公式上 GAN 似乎总是让人摸不着头脑，在代码实现上来说，其实就以下几点：

- 判别器最后一层去掉 sigmoid
- 生成器和判别器的 loss 不取 log
- 每次更新判别器的参数之后把它们的绝对值截断到不超过一个固定常数 c

**4.6. 7.4.6 WGAN-GP: 带有梯度正则的 WGAN.** 实际实验过程发现，WGAN 没有那么好用，主要原因在于 WAGN 进行梯度截断。梯度截断将导致判别网络趋向于一个二值网络，造成模型容量的下降。于是作者提出使用梯度惩罚来替代梯度裁剪。公式如下：

$$L = -\mathbb{E}_{x \sim p_{data}(x)}[f_w(x)] + \mathbb{E}_{x \sim p_g(x)}[f_w(x)] + \lambda \mathbb{E}_{x \sim p_x(x)}[\|\nabla_x(D(x))\|_p - 1]^2 BNbatchLayerNormalizationWasserstein! [W]$$

由于上式是对每一个梯度进行惩罚，所以不适合使用 BN，因为它会引入同个 batch 中不同样本的相互依赖关系。如果需要的话，可以选择 Layer Normalization。实际训练过程中，就可以通过 Wasserstein 距离来度量模型收敛程度了：上图纵坐标是 Wasserstein 距离，横坐标是迭代次数。可以看出，随着迭代的进行，Wasserstein 距离趋于收敛，生成图像也趋于稳定。

**4.7. 7.4.7 LSGAN.** LSGAN (Least Squares GAN) 这篇文章主要针对标准 GAN 的稳定性和图片生成质量不高做了一个改进。作者将原始 GAN 的交叉熵损失采用最小二乘损失替代。LSGAN 的 Loss：

$$\min_D J(D) = \min_D [\frac{1}{2} \mathbb{E}_{x \sim p_{data}(x)}[D(x) - a]^2 + \frac{1}{2} \mathbb{E}_{z \sim p_z(z)}[D(G(z)) - b]^2]$$

$$\min_G J(G) = \min_G \frac{1}{2} \mathbb{E}_{z \sim p_z(z)}[D(G(z)) - c]^2$$

实际实现的时候非常简单，最后一层去掉 sigmoid，并且计算 Loss 的时候用平方误差即可。之所以这么做，作者在原文给出了一张图：上面是作者给出的基于交叉熵损失以及最小二乘损失的 Loss 函数。横坐标代表 Loss 函数的输入，纵坐标代表输出的 Loss 值。可以看出，随着输入的增大，sigmoid 交叉熵损失很快趋于 0，容易导致梯度饱和问题。如果使用右边的 Loss 设计，则只在  $x=0$  点处饱和。因此使用 LSGAN 可以很好的解决交叉熵损失的问题。

#### 4.8. 7.4.8 如何尽量避免 GAN 的训练崩溃问题？

- 归一化图像输入到 (-1, 1) 之间；Generator 最后一层使用 tanh 激活函数
- 生成器的 Loss 采用： $\min(\log 1-D)$ 。因为原始的生成器 Loss 存在梯度消失问题；训练生成器的时候，考虑反转标签，real=fake, fake=real
- 不要在均匀分布上采样，应该在高斯分布上采样
- 一个 Mini-batch 里面必须只有正样本，或者负样本。不要混在一起；如果用不了 Batch Norm，可以用 Instance Norm
- 避免稀疏梯度，即少用 ReLU, MaxPool。可以用 LeakyReLU 替代 ReLU，下采样可以用 Average Pooling 或者 Convolution + stride 替代。上采样可以用 PixelShuffle, ConvTranspose2d + stride
- 平滑标签或者给标签加噪声；平滑标签，即对于正样本，可以使用 0.7-1.2 的随机数替代；对于负样本，可以使用 0-0.3 的随机数替代。给标签加噪声：即训练判别器的时候，随机翻转部分样本的标签。
- 如果可以，请用 DCGAN 或者混合模型：KL+GAN, VAE+GAN。
- 使用 LSGAN, WGAN-GP
- Generator 使用 Adam, Discriminator 使用 SGD
- 尽快发现错误；比如：判别器 Loss 为 0，说明训练失败了；如果生成器 Loss 稳步下降，说明判别器没发挥作用
- 不要试着通过比较生成器，判别器 Loss 的大小来解决训练过程中的模型坍塌问题。  
比如：While Loss D > Loss A: Train D While Loss A > Loss D: Train A
- 如果有标签，请尽量利用标签信息来训练
- 给判别器的输入加一些噪声，给 G 的每一层加一些人工噪声。
- 多训练判别器，尤其是加了噪声的时候
- 对于生成器，在训练，测试的时候使用 Dropout

### 5. 7.3 GAN 的应用（图像翻译）

**5.1. 7.3.1 什么是图像翻译？** GAN 作为一种强有力的生成模型，其应用十分广泛。最为常见的应用就是图像翻译。所谓图像翻译，指从一副图像到另一副图像的转换。可以类比机器翻译，一种语言转换为另一种语言。常见的图像翻译任务有：- 图像去噪 - 图像超分辨 - 图像补全 - 风格迁移 - ...

本节将介绍一个经典的图像翻译网络及其改进。图像翻译可以分为有监督图像翻译和无监督图像翻译：

- 有监督图像翻译：原始域与目标域存在一一对应数据
- 无监督图像翻译：原始域与目标域不存在一一对应数据

**5.2. 7.3.2 有监督图像翻译: pix2pix.** 在这篇 paper 里面, 作者提出的框架十分简洁优雅(好用的算法总是简洁优雅的)。相比以往算法的大量专家知识, 手工复杂的 loss。这篇 paper 非常粗暴, 使用 CGAN 处理了一系列的转换问题。下面是一些转换示例:

上面展示了许多有趣的结果, 比如分割图 → 街景图, 边缘图 → 真实图。对于第一次看到的时候还是很惊艳的, 那么这个是怎么做到的呢? 我们可以设想一下, 如果是我们, 我们自己会如何设计这个网络?

### 直观的想法?

最直接的想法就是, 设计一个 CNN 网络, 直接建立输入-输出的映射, 就像图像去噪问题一样。可是对于上面的问题, 这样做会带来一个问题。**生成图像质量不清晰。**

拿左上角的分割图 → 街景图为例, 语义分割图的每个标签比如“汽车”可能对应不同样式, 颜色的汽车。那么模型学习到的会是所有不同汽车的平均, 这样会造成模糊。

### 如何解决生成图像的模糊问题?

这里作者想了一个办法, 即加入 GAN 的 Loss 去惩罚模型。GAN 相比于传统生成式模型可以较好的生成高分辨率图片。思路也很简单, 在上述直观想法的基础上加入一个判别器, 判断输入图片是否是真实样本。模型示意图如下:

上图模型和 CGAN 有所不同, 但它是一个 CGAN, 只不过输入只有一个, 这个输入就是条件信息。原始的 CGAN 需要输入随机噪声, 以及条件。这里之所以没有输入噪声信息, 是因为在实际实验中, 如果输入噪声和条件, 噪声往往被淹没在条件 C 当中, 所以这里直接省去了。

**5.3. 7.3.3 其他图像翻译的 tricks.** 从上面两点可以得到最终的 Loss 由两部分构成: - 输出和标签信息的 L1 Loss。

- GAN Loss
- 测试也使用 Dropout, 以使输出多样化

$$G^* = \arg \min_G \max_D \Gamma_{cGAN}(G, D) + \lambda \Gamma_{L1}(G)$$

采用 L1 Loss 而不是 L2 Loss 的理由很简单, L1 Loss 相比于 L2 Loss 保边缘 (L2 Loss 基于高斯先验, L1 Loss 基于拉普拉斯先验)。

GAN Loss 为 LSGAN 的最小二乘 Loss, 并使用 PatchGAN(进一步保证生成图像的清晰度)。PatchGAN 将图像拆分成很多个 Patch, 并对每一个 Patch 使用判别器进行判别(实际代码实现有更巧妙的办法), 将所有 Patch 的 Loss 求平均作为最终的 Loss。

**5.4. 7.3.4 如何生成高分辨率图像和高分辨率视频?.** pix2pix 提出了一个通用的图像翻译框架。对于高分辨率的图像生成以及高分辨率的视频生成, 则需要利用更好的网络结构以及

更多的先验只是。pix2pixHD 提出了一种多尺度的生成器以及判别器等方式从而生成高分辨率图像。Vid2Vid 则在 pix2pixHD 的基础上利用光流，时序约束生成了高分辨率视频。

**5.5. 7.3.5 有监督的图像翻译的缺点？** 许多图像翻译算法如前面提及的 pix2pix 系列，需要一一对应的图像。可是在许多应用场景下，往往没有这种一一对应的强监督信息。比如说以下一些应用场景：以第一排第一幅图为例，要找到这种一一配对的数据是不现实的。因此，无监督图像翻译算法就被引入了。

#### 5.6. 7.3.6 无监督图像翻译：CycleGAN. 模型结构

总体思路如下，假设有两个域的数据，记为 A, B。对于上图第一排第一幅图 A 域就是普通的马，B 域就是斑马。由于 A->B 的转换缺乏监督信息，于是，作者提出采用如下方法进行转换：  
 >a. A->fake\_B->rec\_A  
 b. B->fake\_A->rec\_B

对于 A 域的所有图像，学习一个网络 G\_B，该网络可以生成 B。对于 B 域的所有图像，也学习一个网络 G\_A，该网络可以生成 G\_B。

训练过程分成两步，首先对于 A 域的某张图像，送入 G\_B 生成 fake\_B，然后对 fake\_B 送入 G\_A，得到重构后的 A 图像 rec\_A。对于 B 域的某一张图像也是类似。重构后的图像 rec\_A/rec\_B 可以和原图 A/B 做均方误差，实现了有监督的训练。此处值得注意的是 A->fake\_B(B->fake\_A) 和 fake\_A->rec\_B(fake\_B->rec\_A) 的网络是一模一样的。下图是形象化的网络结构图：cycleGAN 的生成器采用 U-Net，判别器采用 LS-GAN。

#### Loss 设计

总的 Loss 就是 X 域和 Y 域的 GAN Loss，以及 Cycle consistency loss：

$$L(G, F, D_X, D_Y) = L_{GAN}(G, D_Y, X, Y) + L_{GAN}(F, D_X, Y, X) + \lambda L_{cycle}(G, F)$$

整个过程 End to end 训练，效果非常惊艳，利用这一框架可以完成非常多有趣的任务

**5.7. 7.3.7 多领域的无监督图像翻译：StarGAN.** cycleGAN 模型较好的解决了无监督图像转换问题，可是这种单一域的图像转换还存在一些问题：

- 要针对每一个域训练一个模型，效率太低。举例来说，我希望可以将橘子转换为红苹果和青苹果。对于 cycleGAN 而言，需要针对红苹果，青苹果分别训练一个模型。
- 对于每一个域都需要搜集大量数据，太麻烦。还是以橘子转换为红苹果和青苹果为例。不管是红苹果还是青苹果，都是苹果，只是颜色不一样而已。这两个任务信息是可以共享的，没必要分别训练两个模型。而且针对红苹果，青苹果分别取搜集大量数据太费事。

starGAN 则提出了一个多领域的无监督图像翻译框架，实现了多个领域的图像转换，且对于不同领域的数据可以混合在一起训练，提高了数据利用率

## 6. 7.4 GAN 的应用（文本生成）

**6.1. 7.4.1 GAN 为什么不适合文本任务？** GAN 在 2014 年被提出之后，在图像生成领域取得了广泛的研究应用。然后在文本领域却一直没有很惊艳的效果。主要在于文本数据是离散数据，而 GAN 在应用于离散数据时存在以下几个问题：

- GAN 的生成器梯度来源于判别器对于正负样本的判别。然而，对于文本生成问题，RNN 输出的是一个概率序列，然后取 argmax。这会导致生成器 Loss 不可导。还可以站在另一个角度理解，由于是 argmax，所以参数更新一点点并不会改变 argmax 的结果，这也使得 GAN 不适合离散数据。
- GAN 只能评估整个序列的 loss，但是无法评估半句话，或者是当前生成单词对后续结果好坏的影响。
- 如果不加 argmax，那么由于生成器生成的都是浮点数值，而 ground truth 都是一 hot encoding，那么判别器只要判别生成的结果是不是 0/1 序列组成的就可以了。这容易导致训练崩溃。

**6.2. 7.4.2 seqGAN 用于文本生成.** seqGAN 在 GAN 的框架下，结合强化学习来做文本生成。模型示意图如下：

在文本生成任务，seqGAN 相比较于普通 GAN 区别在以下几点：

- 生成器不取 argmax。
- 每生成一个单词，则根据当前的词语序列进行蒙特卡洛采样生成完成的句子。然后将句子送入判别器计算 reward。
- 根据得到的 reward 进行策略梯度下降优化模型。

## 7. 7.5 GAN 在其他领域的应用

**7.1. 7.5.1 数据增广.** GAN 的良好生成特性近年来也开始被用于数据增广。以行人重识别为例，有许多 GAN 用于数据增广的工作 [1-4]。行人重识别问题一个难点在于不同摄像头下拍摄的人物环境，角度差别非常大，导致存在较大的 Domain gap。因此，可以考虑使用 GAN 来产生不同摄像头下的数据进行数据增广。以论文 [1] 为例，本篇 paper 提出了一个 cycleGAN 用于数据增广的方法。具体模型结构如下：

cycleGAN 数据增广

对于每一对摄像头都训练一个 cycleGAN，这样就可以实现将一个摄像头下的数据转换成另一个摄像头下的数据，但是内容（人物）保持不变。在 CVPR19 中，[9] 进一步提升了图像的生成质量，进行了“淘宝换衣”式的高质量图像生成（如下图），提供了更高质量的行人训练数据。

**7.2. 7.5.2 图像超分辨与图像补全.** 图像超分辨与补全均可以作为图像翻译问题，该类问题的处理办法也大都是训练一个端到端的网络，输入是原始图片，输出是超分辨率后的图片，或者是补全后的图片。文献 [5] 利用 GAN 作为判别器，使得超分辨率模型输出的图片更加清晰，更符合人眼主管感受。日本早稻田大学研究人员 [6] 提出一种全局 + 局部一致性的 GAN 实现图像补全，使得修复后的图像不仅细节清晰，且具有整体一致性。

**7.3. 7.5.3 语音领域.** 相比于图像领域遍地开花，GAN 在语音领域则应用相对少了很多。这里零碎的找一些 GAN 在语音领域进行应用的例子作为介绍。文献 [7] 提出了一种音频去噪的 SEGAN，缓解了传统方法支持噪声种类稀少，泛化能力不强的问题。Donahue 利用 GAN 进行语音增强，提升了 ASR 系统的识别率。





往往经过 RPN 后输出的不止一个矩形框，所以这里我们是对多个 RoI 进行 Pooling。

### **RoI Pooling 的输入**

输入有两部分组成：

1. 特征图 (feature map): 指的是上面所示的特征图，在 Fast RCNN 中，它位于 RoI Pooling 之前，在 Faster RCNN 中，它是与 RPN 共享那个特征图，通常我们常常称之为“share\_conv”；
2. RoIs，其表示所有 RoI 的  $N \times 5$  的矩阵。其中  $N$  表示 RoI 的数量，第一列表示图像 index，其余四列表示其余的左上角和右下角坐标。

在 Fast RCNN 中，指的是 Selective Search 的输出；在 Faster RCNN 中指的是 RPN 的输出，一堆矩形候选框，形状为  $1 \times 5 \times 1 \times 1$  (4 个坐标 + 索引 index)，其中值得注意的是：坐标的参考系不是针对 feature map 这张图的，而是针对原图的（神经网络最开始的输入）。其实关于 ROI 的坐标理解一直很混乱，到底是根据谁的坐标来。其实很好理解，我们已知原图的大小和由 Selective Search 算法提取的候选框坐标，那么根据“映射关系”可以得出特征图 (featurwe map) 的大小和候选框在 feature map 上的映射坐标。至于如何计算，其实就是比值问题，下面会介绍。所以这里把 ROI 理解为原图上各个候选框 (region proposals)，也是可以的。

注：说句题外话，由 Selective Search 算法提取的一系列可能含有 object 的 bounding box，这些通常称为 region proposals 或者 region of interest (ROI)。

### **RoI 的具体操作**

1. 根据输入 image，将 ROI 映射到 feature map 对应位置

注：映射规则比较简单，就是把各个坐标除以“输入图片与 feature map 的大小的比值”，得到了 feature map 上的 box 坐标

2. 将映射后的区域划分为相同大小的 sections (sections 数量与输出的维度相同)
3. 对每个 sections 进行 max pooling 操作

这样我们就可以从不同大小的方框得到固定大小的相应的 feature maps。值得一提的是，输出的 feature maps 的大小不取决于 ROI 和卷积 feature maps 大小。RoI Pooling 最大的好处就在于极大地提高了处理速度。

### **RoI Pooling 的输出**

### 2.3. 8.2.3 Faster R-CNN. Faster R-CNN 有哪些创新点？

Fast R-CNN 依赖于外部候选区域方法，如选择性搜索。但这些算法在 CPU 上运行且速度很慢。在测试中，Fast R-CNN 需要 2.3 秒来进行预测，其中 2 秒用于生成 2000 个 ROI。Faster R-CNN 采用与 Fast R-CNN 相同的设计，只是它用内部深层网络代替了候选区域方法。新的候选区域网络（RPN）在生成 ROI 时效率更高，并且以每幅图像 10 毫秒的速度运行。

图 8.1.13 Faster R-CNN 的流程图 Faster R-CNN 的流程图与 Fast R-CNN 相同，采用外部候选区域方法代替了内部深层网络。

#### 图 8.1.14 候选区域网络

候选区域网络（RPN）将第一个卷积网络的输出特征图作为输入。它在特征图上滑动一个  $3 \times 3$  的卷积核，以使用卷积网络（如下所示的 ZF 网络）构建与类别无关的候选区域。其他深度网络（如 VGG 或 ResNet）可用于更全面的特征提取，但这需要以速度为代价。ZF 网络最后会输出 256 个值，它们将馈送到两个独立的全连接层，以预测边界框和两个 objectness 分数，这两个 objectness 分数度量了边界框是否包含目标。我们其实可以使用回归器计算单个 objectness 分数，但为简洁起见，Faster R-CNN 使用只有两个类别的分类器：即带有目标的类别和不带有目标的类别。

图 8.1.15 对于特征图中的每一个位置，RPN 会做  $k$  次预测。因此，RPN 将输出  $4 \times k$  个坐标和每个位置上  $2 \times k$  个得分。下图展示了  $8 \times 8$  的特征图，且有一个  $3 \times 3$  的卷积核执行运算，它最后输出  $8 \times 8 \times 3$  个 ROI（其中  $k=3$ ）。下图（右）展示了单个位置的 3 个候选区域。

图 8.1.16 假设最好涵盖不同的形状和大小。因此，Faster R-CNN 不会创建随机边界框。相反，它会预测一些与左上角名为锚点的参考框相关的偏移量（如  $x, y$ ）。我们限制这些偏移量的值，因此我们的猜想仍然类似于锚点。

图 8.1.17 要对每个位置进行  $k$  个预测，我们需要以每个位置为中心的  $k$  个锚点。每个预测与特定锚点相关联，但不同位置共享相同形状的锚点。

图 8.1.18 这些锚点是精心挑选的，因此它们是多样的，且覆盖具有不同比例和宽高比的现实目标。这使得我们可以用更好的猜想来指导初始训练，并允许每个预测专门用于特定的形状。该策略使早期训练更加稳定和简便。

图 8.1.19 Faster R-CNN 使用更多的锚点。它部署 9 个锚点框：3 个不同宽高比的 3 个不同大小的锚点（Anchor）框。每一个位置使用 9 个锚点，每个位置会生成  $2 \times 9$  个 objectness 分数和  $4 \times 9$  个坐标。

### 2.4. 8.2.4 R-FCN. R-FCN 有哪些创新点？

R-FCN 仍属于 two-stage 目标检测算法：RPN+R-FCN

1. Fully convolutional
2. 位置敏感得分图（position-sensitive score maps）

- 在 FPN 基础上明显有大物体涨点，同时由于高分辨率，小物体也有不错的提升。
- 膨胀卷积提供的大感受野使得分类也不逊色

### 3.2 Results analysis

- 从 AP50 看出，高好 1.7；从 AP80 看出，高了 3.7。由此可以看出确实提高了检测性能。（
- 从定位性能来看，大物体的提升比小物体更多。作者认为是高分辨率解决了大物体边界模糊的问题。其实有一种解释：小目标没有大目标明显，因为膨胀卷积核降采样都会丢失小目标，只是膨胀卷积可能离散采样不至于像降采样直接给到后面没了，但是没有根本性的解决，所以小目标不大。
- AR 指标也有类似结论
- AR50 体现了小目标的查全率更好，这也印证上面分析的：相对降采样，膨胀卷积丢失会好点。此下大目标效果虽然提升不大但是也很高了，作者表示 DetNet 擅长找到更精确的定位目标，在 AR85 的高指标就能看出。
- AR85 看大目标丢失少，说明能够像 VGG 一样对大目标效果优良。关于小目标的效果平平，作者认为没有必要太高，因为 FPN 结构对小目标已经利用地很充分了，这里即使不高也没事。

### 3.3 Discussion \* 关于 stage

为了研究 backbone 对检测的影响，首先研究 stage 的作用。前 4 个还好说，和 ResNet 一样，但是 P5 P6 就不同，没有尺度的变化，和传统意义的 stage 不一样了，需要重新定义。这里 DetNet 也是类似 ResNet 的方法，虽然没有尺度变化，但是 AB 模块的位置还是保持了，B 开启一个 stage（听上去有点牵强）。如下图，认为新加的仍属于 P5。

验证方法是做了实验，将 P6 开始的 block 换成上图所示的 A 模块对比效果如下图。发现还是加了 B 效果更好。（但是这个 stage 和传统意义很不一样，所以很多性质不能相提并论，只是 B 模块的改变也不好判定什么）

**2.8. 8.2.8 CBNet.** 本部分介绍一篇在 COCO 数据集达到最高单模型性能——mAP 53.3 的网络，论文于 2019.9.3 发布在 ArXiv，全名是 *CBNet: A Novel Composite Backbone Network Architecture for Object Detection*

#### 1. Introduction

名义上是单模型，实际是多模型的特征融合，只是和真正的多模型策略略有不同。作者的起点是，设计新的模型往往需要在 ImageNet 上进行预训练，比较麻烦。因而提出的 Composite Backbone Network (CBNet)，采用经典网络的多重组合的方式构建网络，一方面可以提取到更有效的特征，另一方面也能够直接用现成的预训练参数（如 ResNet，ResNeXt 等）比较简单高效。

## 2. Proposed method Architecture of CBNet

如上图，模型中采用  $K$  个 ( $K > 1$ ) 相同的结构进行紧密联结。其中两个相同 backbone 的叫 Dual-Backbone (DB)，三个叫 Triple- Backbone (TB)； $L$  代表 backbone 的 stage 数目，这里统一设置为  $L=5$ 。其中，和前任工作不同的地方在于，这里将不同的 stage 信息进行复用回传，以便获取更好的特征（为什么 work 不好说）。

### 2.2 Other possible composite styles

相关工作的其他类似结构，大同小异。要么是前面 backbone 的 stage 往后传播，要么是往前一个传播，每个都有一篇论文，应该都会给出不同的解释；第四个结构不太一样，是类似 densnet 的结构，但是密集连接 + 多 backbone assemble 的内存消耗不出意外会非常大。但是脱离这些体系来看，多 backbone 的结构类似多模型的 assemble，和单模型有点不公平。

## 3. Experiment

- result

COCO 数据集上的结果。看来提升还是有的。但是也能看出，大趋势上，三阶级联效果不如两阶的提升大，也是这部分的特征提升空间有限的缘故，到底哪部分在 work 不好说。下图的研究就更说明这一点了，斜率逐渐减小。

- Comparisons of different composite styles

他的级联网络相比，作者的阐述点只落脚于特征的利用情况，但是这个东西本身就很玄乎，不好说到底怎么算利用得好。硬要说这种做法的解释性，大概就是将 backbone 方向的后面高级语义特征传播回前面进行加强，相当于横向的 FPN 传播。

- Number of backbones in CBNet

速度慢是必然的，FPN+ResNeXt 为 8fps，加上两个 backbone 后为 5.5FPS；如果减去 backbone 的前两个 stage，可以节省部分参数达到 6.9FPS，而精度下降不大（整体速度太低，这个实验意义不大）

- Sharing weights for CBNet

- 从中可以看出其实权重是否 share 区别不大，不到一个点的降幅，参数量减少。
- Effectiveness of basic feature enhancement by CBNet

**2. 怎样对先验框进行匹配？** SSD 在训练的时候只需要输入图像和图像中每个目标对应的 ground truth. 先验框与 ground truth 的匹配遵循两个原则：

(1) 对图片中的每个 ground truth, 在先验框中找到与其 IOU 最大的先验框，则该先验框对应的预测边界框与 ground truth 匹配。

(2) 对于 (1) 中每个剩下的没有与任何 ground truth 匹配到的先验框，找到与其 IOU 最大的 ground truth, 若其与该 ground truth 的 IOU 值大于某个阈值（一般设为 0.5），则该先验框对应的预测边界框与该 ground truth 匹配。

按照这两个原则进行匹配，匹配到 ground truth 的先验框对应的预测边界框作为正样本，没有匹配到 ground truth 的先验框对应的预测边界框作为负样本。尽管一个 ground truth 可以与多个先验框匹配，但是 ground truth 的数量相对先验框还是很少，按照上面的原则进行匹配还是会造成员本远多于正样本的情况。为了使正负样本尽量均衡（一般保证正负样本比例约为 1: 3），SSD 采用 hard negative mining, 即对负样本按照其预测背景类的置信度进行降序排列，选取置信度较小的 top-k 作为训练的负样本。

### 3. 怎样得到预测的检测结果？

最后分别在所选的特征层上使用  $3 \times 3$  卷积核预测不同 default boxes 所属的类别分数及其预测的边界框 location。由于对于每个 box 需要预测该 box 属于每个类别的置信度（假设有 c 类，包括背景，例如 20 class 的数据集合， $c=21$ ）和该 box 对应的预测边界框的 location(包含 4 个值，即该 box 的中心坐标和宽高)，则每个 box 需要预测  $c+4$  个值。所以对于某个所选的特征层，该层的卷积核个数为  $(c+4) \times$  该层的 default box 个数。最后将每个层得到的卷积结果进行拼接。对于得到的每个预测框，取其类别置信度的最大值，若该最大值大于置信度阈值，则最大值所对应的类别即为该预测框的类别，否则过滤掉此框。对于保留的预测框根据它对应的先验框进行解码得到其真实的位置参数（这里还需注意要防止预测框位置超出图片），然后根据所属类别置信度进行降序排列，取 top-k 个预测框，最后进行 NMS，过滤掉重叠度较大的预测框，最后得到检测结果。

SSD 优势是速度比较快，整个过程只需要一步，首先在图片不同位置按照不同尺度和宽高比进行密集抽样，然后利用 CNN 提取特征后直接进行分类与回归，所以速度比较快，但均匀密集采样会造成正负样本不均衡的情况使得训练比较困难，导致模型准确度有所降低。另外，SSD 对小目标的检测没有大目标好，因为随着网络的加深，在高层特征图中小目标的信息丢失掉了，适当增大输入图片的尺寸可以提升小目标的检测效果。

#### 3.2. 8.3.2 DSSD. DSSD 有哪些创新点？

1. Backbone：将 ResNet 替换 SSD 中的 VGG 网络，增强了特征提取能力
2. 添加了 Deconvolution 层，增加了大量上下文信息

为了解决 SSD 算法检测小目标困难的问题，DSSD 算法将 SSD 算法基础网络从 VGG-16 更改为 ResNet-101，增强网络特征提取能力，其次参考 FPN 算法思路利用去 Deconvolution

YOLOv2 结合 Dimension Clusters, 通过对边界框的位置预测进行约束, 使模型更容易稳定训练, 这种方式使得模型的 mAP 值提升了约 5%。

### (7) Fine-Grained Features

YOLOv2 借鉴 SSD 使用多尺度的特征图做检测, 提出 pass through 层将高分辨率的特征图与低分辨率的特征图联系在一起, 从而实现多尺度检测。YOLOv2 提取 Darknet-19 最后一个 max pool 层的输入, 得到  $26 \times 26 \times 512$  的特征图。经过  $1 \times 1 \times 64$  的卷积以降低特征图的维度, 得到  $26 \times 26 \times 64$  的特征图, 然后经过 pass through 层的处理变成  $13 \times 13 \times 256$  的特征图 (抽取原特征图每个  $2 \times 2$  的局部区域组成新的 channel, 即原特征图大小降低 4 倍, channel 增加 4 倍), 再与  $13 \times 13 \times 1024$  大小的特征图连接, 变成  $13 \times 13 \times 1280$  的特征图, 最后在这些特征图上做预测。使用 Fine-Grained Features, YOLOv2 的性能提升了 1%.

### (8) Multi-Scale Training

YOLOv2 中使用的 Darknet-19 网络结构中只有卷积层和池化层, 所以其对输入图片的大小没有限制。YOLOv2 采用多尺度输入的方式训练, 在训练过程中每隔 10 个 batches, 重新随机选择输入图片的尺寸, 由于 Darknet-19 下采样总步长为 32, 输入图片的尺寸一般选择 32 的倍数 {320,352,...,608}。采用 Multi-Scale Training, 可以适应不同大小的图片输入, 当采用低分辨率的图片输入时, mAP 值略有下降, 但速度更快, 当采用高分辨率的图片输入时, 能得到较高 mAP 值, 但速度有所下降。

YOLOv2 借鉴了很多其它目标检测方法的一些技巧, 如 Faster R-CNN 的 anchor boxes, SSD 中的多尺度检测。除此之外, YOLOv2 在网络设计上做了很多 tricks, 使它能在保证速度的同时提高检测准确率, Multi-Scale Training 更使得同一个模型适应不同大小的输入, 从而可以在速度和精度上进行自由权衡。

## YOLOv2 的训练

YOLOv2 的训练主要包括三个阶段。第一阶段: 先在 ImageNet 分类数据集上预训练 Darknet-19, 此时模型输入为  $224 \times 224$ , 共训练 160 个 epochs。第二阶段: 将网络的输入调整为  $448 \times 448$ , 继续在 ImageNet 数据集上 finetune 分类模型, 训练 10 个 epochs, 此时分类模型的 top-1 准确度为 76.5%, 而 top-5 准确度为 93.3%。第三个阶段: 修改 Darknet-19 分类模型为检测模型, 并在检测数据集上继续 finetune 网络。网络修改包括 (网路结构可视化): 移除最后一个卷积层、global avgpooling 层以及 softmax 层, 并且新增了三个  $3 \times 3 \times 2048$  卷积层, 同时增加了一个 passthrough 层, 最后使用  $1 \times 1$  卷积层输出预测结果。

### 3.5. 8.3.5 YOLO9000. [github: http://pjreddie.com/yolo9000/](http://pjreddie.com/yolo9000/)

YOLO9000 是在 YOLOv2 的基础上提出的一种联合训练方法, 可以检测超过 9000 个类别的模型。YOLOv2 混合目标检测数据集和分类数据集, 用目标检测数据集及其类别标记信

作者对 one-stage 检测器准确率不高的问题进行探究，发现主要问题在于正负类别不均衡（简单-难分类别不均衡）。

We discover that the extreme foreground-background class imbalance encountered during training of dense detectors is the central cause.

作者建议通过重新设计标准的交叉熵损失（cross entropy loss）来解决这种类别不平衡（class imbalance）问题，即提出 Focal Loss。

We propose to address this class imbalance by reshaping the standard cross entropy loss such that it down-weights the loss assigned to well-classified examples. Our novel Focal Loss focuses training on a sparse set of hard examples and prevents the vast number of easy negatives from overwhelming the detector during training.

结合 Focal Loss 的 one-stage 检测器称为 RetinaNet，该检测器在 COCO 上 mAP 可以和特征金字塔网络（feature pyramid network, FPN）或者 Mask R-CNN 接近，

**问：什么是类别不均衡（class imbalance）？**

答：负样本的数量极大于正样本的数量，比如包含物体的区域（正样本）很少，而不包含物体的区域（负样本）很多。比如检测算法在早期会生成一大波的 bbox。而一幅常规的图片中，顶多就那么几个 object。这意味着，绝大多数的 bbox 属于 background。

**问：样本的类别不均衡会带来什么问题？**

答：由于大多数都是简单易分的负样本（属于背景的样本），使得训练过程不能充分学习到属于那些有类别样本的信息；其次简单易分的负样本太多，可能掩盖了其他有类别样本的作用（这些简单易分的负样本仍产生一定幅度的 loss，见下图蓝色曲线，数量多会对 loss 起主要贡献作用，因此就主导了梯度的更新方向，掩盖了重要的信息）

This imbalance causes two problems: (1) training is inefficient as most locations are easy negatives that contribute no useful learning signal; (2) en masse, the easy negatives can overwhelm training and lead to degenerate models.

简单来说，因为 bbox 数量爆炸。正是因为 bbox 中属于 background 的 bbox 太多了，所以如果分类器无脑地把所有 bbox 统一归类为 background，accuracy 也可以刷得很高。于是乎，分类器的训练就失败了。分类器训练失败，检测精度自然就低了。

**问：为什么在 two-stage 检测器中，没有出现类别不均衡（class imbalance）问题呢？**

答：因为通过 RPN 阶段可以减少候选目标区域，而在分类阶段，可以固定前景与背景比值（foreground-to-background ratio）为 1:3，或者使用 OHEM（online hard example mining）使得前景和背景的数量达到均衡。

## 2. Image Data Augmentation techniques

### 2.1 Data Augmentations based on basic image manipulations

- Geometric transformations

如果数据集潜在的表征能够被观察和分离，那么简单的几何变换就能取得很好的效果。对于复杂的数据集如医学影像，数据小而且训练集和测试集的偏差大，几何变换等增强的合理运用就很关键。

- Flipping

作者提到了要衡量普遍性的观点。但是这种变换对于数字数据集不具有安全性。

- Color space

主要提及的识别 RGB 通道上的变换，将三通道图进行分离，以及直方图变换增强等。（颜色空间更多增强方式可以参考 A Preliminary Study on Data Augmentation of Deep Learning for Image Classification）

- Cropping

通常在输入图片的尺寸不一时会进行按中心的裁剪操作。裁剪某种程度上和平移操作有相似性。根据裁剪幅度变化，该操作具有一定的不安全性。

- Rotation

大幅度的旋转对数字集会有不安全性的考虑。

- Translation

平移也需要合理设计。如车站人脸检测，只需要中心检测时，就可以加合适的平移增强。平移后空出部分填 0 或者 255，或用高斯分布噪声。

- Noise injection

在像素上叠加高斯分布的随机噪声。

- Color space transformations

由于实际图像中一定存在光线偏差，所以光线的增强十分有必要（但是 IJCV 的光流文章指出，3D 建模的灯光增强实在是很难学习到，所以对于光线增强的效果不如几何也可能因为光线的复杂度更高，数据样本远远不够）。色彩变换十分多样，如像素限制、像素矩阵变换、像素值颠倒等；灰度图和彩图相比，计算时间成本大大较少，但是据实验效果会下降一些，很明显因为特征的维度被降维了；还有尝试将 RGB 映射到其他的色彩空间进行学习，YUV,CMY,HSV 等。

除了计算大内存消耗和时间长等缺点，色彩变换也面临不安全性，比如识别人脸的关键信息是黄白黑，但是大量增强出红绿蓝，会丢信息。颜色变换的增强方法是从色彩空间角度拟合偏置，效果有限的可能性是多样的：1. 真实几何多样性比颜色更简单 2. 色彩的变化多样性更多，导致增强不够反而学不好，颜色空间的欠拟合 3.

**变换不安全**

- AutoAugment

谷歌最早做的自学习增强方法，走的 NAS 的思路 RL+RNN 搜索增强空间，还有后来最近发的检测增强也是大同小异，基本就是换汤不换药，问题在于搜索空间太大，复现搜索过于依赖硬件条件（普通实验室玩不起）

### 3. Design considerations for image Data Augmentation

#### 3.1 Test-time augmentation

许多论文指出在检测阶段进行同等的数据增强能够获得较好的效果。归结可以认为是训练检测阶段的一致性。当然，这种手段时间成本太高，只在如医学影像等追求精度的关键领域可以使用。

#### 3.2 Curriculum learning

Bengio 团队早年在 ICML 提出的观点，确实合理，一开始就进行大量的增强容易导致网络不收敛。从一个数据集学习到的数据增强也可以迁移到其他数据集。

3.3 Resolution impact 高清 ( $1920 \times 1080 \times 3$ ) 或 4K ( $3840 \times 2160 \times 3$ ) 等高分辨率图像需要更多的处理和内存来训练深度 CNN。然而下一代模型更倾向于使用这样更高分辨率的图像。因为模型中常用的下采样会造成图像中信息的丢失，使图像识别更困难。研究人员发现，高分辨率图像和低分辨率图像一起训练的模型集合，比单独的任何一个模型都要好。某个实验（这里就不注明引用了）在  $256 \times 256$  图像和  $512 \times 512$  图像上训练的模型分别获得 7.96% 和 7.42% 的 top-5 error。汇总后，他们的 top-5 error 变低，为 6.97%。随着超分辨率网络的发展，将图像放大到更高的分辨率后训练模型，能够得到更好更健壮的图像分类器。

#### 3.4 Final dataset size

数据增强的形式可以分为在线和离线增强。前者是在加载数据时增强，可能造成额外的内存消耗（现在都是数据容量不变的随机增强）。

此外作者提到了一个比较有意思的观点：当前数据集尤其是进行增广后是十分庞大的，明显能够在一定程度上缩小数据集但是保持性能下降不多的子集效率会高得多。

#### 3.5 Alleviating class imbalance with Data Augmentation

这也是值得借鉴的一点。通过增强在一定程度上解决类别不平衡问题。但增强需要仔细设计，否则会面对已经学习较好的类别或者场景造成过拟合等问题。

##### 5.2. 8.5.2 OHEM.

5.3. 8.5.3 NMS: Soft NMS/ Polygon NMS/ Inclined NMS/ ConvNMS/ Yes-Net NMS/ Softer NMS.

##### 5.4. 8.5.4 Multi Scale Training/Testing.

##### 5.5. 8.5.5 建立小物体与 context 的关系.

这里进一步改进了一下 label smooth 的公式而已，在原来基础上除了个类别数。

**2.3 Data Preprocessing** 就是数据增强，没什么其他的。至于分类也是几何变换和色彩变换。这么分区别其实是否变换 label。但是将真实世界就这么简单地分解过于粗糙了。好不容易谷歌的增强考虑到了如何学习一下检测任务的增强，但是也只是加了 bbox\_only 的增强，就效果而言，一般；而且就实际来说，合理性和有效性有待商榷。

作者认为，两阶段网络的 RPN 生成就是对输入的任意裁剪，所以这个增强就够了；这老哥膨胀了，two-stage 就不用裁剪的增强，虽然两阶段能提供一些不变性，但是用了一般来说都是更好的。

**2.4 Training Schedule Revamping** 训练策略上：余弦学习率调整 +warmup

**2.5 Synchronized Batch Normalization** 跨多卡同步正则化，土豪专区，穷人退避

**2.6 Random shapes training for single-stage object detection networks**

多尺度训练，每经过一定的 iteration 更换一种尺度。举例是 yolov3 的尺度范围。

## 6. 8.6 目标检测的常用数据集

**6.1. 8.6.1 PASCAL VOC.** VOC 数据集是目标检测经常用的一个数据集，自 2005 年起每年举办一次比赛，最开始只有 4 类，到 2007 年扩充为 20 个类，共有两个常用的版本：2007 和 2012。学术界常用 5k 的 train/val 2007 和 16k 的 train/val 2012 作为训练集，test 2007 作为测试集，用 10k 的 train/val 2007+test 2007 和 16k 的 train/val 2012 作为训练集，test2012 作为测试集，分别汇报结果。

**6.2. 8.6.2 MS COCO.** COCO 数据集是微软团队发布的一个可以用来图像 recognition+segmentation+captioning 数据集，该数据集收集了大量包含常见物体的日常场景图片，并提供像素级的实例标注以更精确地评估检测和分割算法的效果，致力于推动场景理解的研究进展。依托这一数据集，每年举办一次比赛，现已涵盖检测、分割、关键点识别、注释等机器视觉的中心任务，是继 ImageNet Chanllenge 以来最有影响力的学术竞赛之一。

相比 ImageNet，COCO 更加偏好目标与其场景共同出现的图片，即 non-iconic images。这样的图片能够反映视觉上的语义，更符合图像理解的任务要求。而相对的 iconic images 则更适合浅语义的图像分类等任务。

COCO 的检测任务共含有 80 个类，在 2014 年发布的数据规模分 train/val/test 分别为 80k/40k/40k，学术界较为通用的划分是使用 train 和 35k 的 val 子集作为训练集 (trainval35k)，使用剩余的 val 作为测试集 (minival)，同时向官方的 evaluation server 提交结果 (test-dev)。除此之外，COCO 官方也保留一部分 test 数据作为比赛的评测集。

**6.3. 8.6.3 Google Open Image.** Open Image 是谷歌团队发布的数据集。最新发布的 Open Images V4 包含 190 万图像、600 个种类，1540 万个 bounding-box 标注，是当前最大

的带物体位置标注信息的数据集。这些边界框大部分都是由专业注释人员手动绘制的，确保了它们的准确性和一致性。另外，这些图像是非常多样化的，并且通常包含有多个对象的复杂场景（平均每个图像 8 个）。

**6.4. 8.6.4 ImageNet.** ImageNet 是一个计算机视觉系统识别项目，是目前世界上图像识别最大的数据库。ImageNet 是美国斯坦福的计算机科学家，模拟人类的识别系统建立的。能够从图片识别物体。Imagenet 数据集文档详细，有专门的团队维护，使用非常方便，在计算机视觉领域研究论文中应用非常广，几乎成为了目前深度学习图像领域算法性能检验的“标准”数据集。Imagenet 数据集有 1400 多万幅图片，涵盖 2 万多个类别；其中有超过百万的图片有明确的类别标注和图像中物体位置的标注。

**6.5. 8.6.5 DOTA.** DOTA 是遥感航空图像检测的常用数据集，包含 2806 张航空图像，尺寸大约为 4kx4k，包含 15 个类别共计 188282 个实例，其中 14 个主类，small vehicle 和 large vehicle 都是 vehicle 的子类。其标注方式为四点确定的任意形状和方向的四边形。航空图像区别于传统数据集，有其自己的特点，如：尺度变化性更大；密集的小物体检测；检测目标的不确定性。数据划分为 1/6 验证集，1/3 测试集，1/2 训练集。目前发布了训练集和验证集，图像尺寸从 800x800 到 4000x4000 不等。

## 7. 8.7 目标检测常用标注工具

**7.1. 8.7.1 LabelImg.** LabelImg 是一款开源的图像标注工具，标签可用于分类和目标检测，它是用 Python 编写的，并使用 Qt 作为其图形界面，简单好用。注释以 PASCAL VOC 格式保存为 XML 文件，这是 ImageNet 使用的格式。此外，它还支持 COCO 数据集格式。

**7.2. 8.7.2 labelme.** labelme 是一款开源的图像/视频标注工具，标签可用于目标检测、分割和分类。灵感是来自于 MIT 开源的一款标注工具 LabelMe。labelme 具有的特点是：

- 支持图像的标注的组件有：矩形框，多边形，圆，线，点（rectangle, polygons, circle, lines, points）
- 支持视频标注
- GUI 自定义
- 支持导出 VOC 格式用于 semantic/instance segmentation
- 支持导出 COCO 格式用于 instance segmentation

**7.3. 8.7.3 Labelbox.** Labelbox 是一家为机器学习应用程序创建、管理和维护数据集的服务提供商，其中包含一款部分免费的数据标签工具，包含图像分类和分割，文本，音频和视频注释的接口，其中图像视频标注具有的功能如下：

- 可用于标注的组件有：矩形框，多边形，线，点，画笔，超像素等（bounding box, polygons, lines, points, brush, subpixels）
- 标签可用于分类，分割，目标检测等
- 以 JSON / CSV / WKT / COCO / Pascal VOC 等格式导出数据
- 支持 Tiled Imagery (Maps)
- 支持视频标注（快要更新）

**7.4. 8.7.4 RectLabel.** RectLabel 是一款在线免费图像标注工具，标签可用于目标检测、分割和分类。具有的功能或特点：

- 可用的组件：矩形框，多边形，三次贝塞尔曲线，直线和点，画笔，超像素
- 可只标记整张图像而不绘制
- 可使用画笔和超像素
- 导出为 YOLO, KITTI, COCO JSON 和 CSV 格式
- 以 PASCAL VOC XML 格式读写
- 使用 Core ML 模型自动标记图像
- 将视频转换为图像帧

**7.5. 8.7.5 CVAT.** CVAT 是一款开源的基于网络的交互式视频/图像标注工具，是对加州视频标注工具（Video Annotation Tool）项目的重新设计和实现。OpenCV 团队正在使用该工具来标注不同属性的数百万个对象，许多 UI 和 UX 的决策都基于专业数据标注团队的反馈。具有的功能

- 关键帧之间的边界框插值
- 自动标注（使用 TensorFlow OD API 和 Intel OpenVINO IR 格式的深度学习模型）

**7.6. 8.7.6 VIA.** VGG Image Annotator (VIA) 是一款简单独立的手动注释软件，适用于图像，音频和视频。VIA 在 Web 浏览器中运行，不需要任何安装或设置。页面可在大多数现代 Web 浏览器中作为离线应用程序运行。

- 支持标注的区域组件有：矩形，圆形，椭圆形，多边形，点和折线

**7.7. 8.7.6 其他标注工具.** liblabel，一个用 MATLAB 写的轻量级语义/示例 (semantic/instance) 标注工具。ImageTagger：一个开源的图像标注平台。Anno-Mage：一个利用深度学习模型半自动图像标注工具，预训练模型是基于 MS COCO 数据集，用 RetinaNet 训练的。

当然还有一些数据标注公司，可能包含更多标注功能，例如对三维目标检测的标注（3D Bounding box Labelling），激光雷达点云的标注（LIDAR 3D Point Cloud Labeling）等。

## 8. 8.8 目标检测工具和框架（贡献者：北京理工大学-明奇）

各种不同的算法虽然部分官方会有公布代码，或者 github 上有人复现，但是囿于安装环境不一，实现的框架（pytorch、C++、Caffe、tensorflow、MXNet 等）不同，每次更换算法都需要重新安装环境，并且代码之间的迁移性差，十分不方便。所以为了方便将不同的算法统一在一个代码库中，不同的大厂都提出了自己的解决方案。如 facebook 的 Detectron、商汤科技的 mmdetection、SimpleDet 等。其中 Detectron 最早，所以用户量最大，其次是国内近段时间崛起的 mmdetection，下面介绍该目标检测工具箱。

### 1. Introduction

MMDetection 的特点：

- 模块化设计：将不同网络的部分进行切割，模块之间具有很高的复用性和独立性（十分便利，可以任意组合）
- 高效的内存使用
- SOTA

### 2. Support Frameworks

- 单阶段检测器  
SSD、RetinaNet、FCOS、FSAF
- 两阶段检测器  
Faster R-CNN、R-FCN、Mask R-CNN、Mask Scoring R-CNN、Grid R-CNN
- 多阶段检测器  
Cascade R-CNN、Hybrid Task Cascade
- 通用模块和方法  
soft-NMS、DCN、OHEM、Train from Scratch、M2Det、GN、HRNet、Libra R-CNN

### 3. Architecture

模型表征：划分为以下几个模块：

Backbone (ResNet 等)、Neck (FPN)、DenseHead (AnchorHead)、RoIExtractor、RoIHead (BBoxHead/MaskHead)

结构图如下：

### 4. Notice

- 1x 代表 12epoch 的 COCO 训练，2x 类似推导
- 由于 batch-size 一般比较小 (1/2 这样的量级)，所以大多数地方默认冻结 BN 层。可以使用 GN 代替。

5. **参考链接** mmdetection 代码高度模块化，十分好用和便利，更详细的文档直接参见官方文档：<https://github.com/open-mmlab/mmdetection>  
注释版的 mmdetection 代码（更新至 v1.0.0）：<https://github.com/ming71/mmdetection-annotated>

使用方法简介：安装记录（可能过时，以官方文档为准）：<https://ming71.github.io/mmdetection-memo.html> 使用方法（截止更新日期，如果过时以官方为准）：<https://ming71.github.io/mmdetection-instruction.html>

## 9. TODO

- [ ] 目标检测基础知识：mAP、IoU 和 NMS 等
- [ ] 目标检测评测指标
- [ ] 目标检测常见标注工具
- [ ] 完善目标检测的技巧汇总
- [ ] 目标检测的现在难点和未来发展





如上图所示：

(1) 在 CNN 中，猫的图片输入到 AlexNet，得到一个长为 1000 的输出向量，表示输入图像属于每一类的概率，其中在“tabby cat”这一类统计概率最高，用来做分类任务。

(2) FCN 与 CNN 的区别在于把 CNN 最后的全连接层转换成卷积层，输出的是一张已经带有标签的图片，而这个图片就可以做语义分割。

(3) CNN 的强大之处在于它的多层结构能自动学习特征，并且可以学到多个层次的特征：较浅的卷积层感知域较小，学到一些局部区域的特征；较深的卷积层具有较大的感知域，能够学到更加抽象一些的特征。高层的抽象特征对物体的大小、位置和方向等敏感性更低，从而有助于识别性能的提高，所以我们常常可以将卷积层看作是特征提取器。

#### 3.4. 9.2.4 全连接层和卷积层如何相互转化？ 两者相互转换的可能性：

全连接层和卷积层之间唯一的不同就是卷积层中的神经元只与输入数据中的一个局部区域连接，并且在卷积列中的神经元共享参数。然而在两类层中，神经元都是计算点积，所以它们的函数形式是一样的。因此，将此两者相互转化是可能的：

(1) 对于任一个卷积层，都存在一个能实现和它一样的前向传播函数的全连接层。权重矩阵是一个巨大的矩阵，除了某些特定块，其余部分都是零。而在其中大部分块中，元素都是相等的。

(2) 任何全连接层都可以被转化为卷积层。比如 VGG16 中第一个全连接层是  $25088 * 4096$  的数据尺寸，将它转化为  $512 * 7 * 7 * 4096$  的数据尺寸，即一个  $K=4096$  的全连接层，输入数据体的尺寸是  $7 * 7 * 512$ ，这个全连接层可以被等效地看做一个  $F=7, P=0, S=1, K=4096$  的卷积层。换句话说，就是将滤波器的尺寸设置为和输入数据体的尺寸一致  $7 * 7$ ，这样输出就变为  $1 * 1 * 4096$ ，本质上和全连接层的输出是一样的。

#### 输出激活数据体深度是由卷积核的数目决定的 ( $K=4096$ )。

在两种变换中，将全连接层转化为卷积层在实际运用中更加有用。假设一个卷积神经网络的输入是  $227 \times 227 \times 3$  的图像，一系列的卷积层和下采样层将图像数据变为尺寸为  $7 \times 7 \times 512$  的激活数据体，AlexNet 的处理方式为使用了两个尺寸为 4096 的全连接层，最后一个有 1000 个神经元的全连接层用于计算分类评分。我们可以将这 3 个全连接层中的任意一个转化为卷积层：

(1) 第一个连接区域是  $[7 \times 7 \times 512]$  的全连接层，令其滤波器尺寸为  $F=7, K=4096$ ，这样输出数据体就为  $[1 \times 1 \times 4096]$ 。

(1) 使用全卷积神经网络。(全卷积神经网络就是卷积取代了全连接层，全连接层必须固定图像大小而卷积不用，所以这个策略使得，你可以输入任意尺寸的图片，而且输出也是图片，所以这是一个端到端的网络。)

(2) 左边的网络是收缩路径：使用卷积和 maxpooling。

(3) 右边的网络是扩张路径：使用上采样产生的特征图与左侧收缩路径对应层产生的特征图进行 concatenate 操作。(pooling 层会丢失图像信息和降低图像分辨率且是不可逆的操作，对图像分割任务有一些影响，对图像分类任务的影响不大，为什么要做上采样？因为上采样可以补足一些图片的信息，但是信息补充的肯定不完全，所以还需要与左边的分辨率比较高的图片相连接起来（直接复制过来再裁剪到与上采样图片一样大小），这就相当于在高分辨率和更抽象特征当中做一个折衷，因为随着卷积次数增多，提取的特征也更加有效，更加抽象，上采样的图片是经历多次卷积后的图片，肯定是比较高效和抽象的图片，然后把它与左边不怎么抽象但更高分辨率的特征图片进行连接）。

(4) 最后再经过两次反卷积操作，生成特征图，再用两个 1X1 的卷积做分类得到最后的两张 heatmap，例如第一张表示的是第一类的得分，第二张表示第二类的得分 heatmap，然后作为 softmax 函数的输入，算出概率比较大的 softmax 类，选择它作为输入给交叉熵进行反向传播训练。

下面是 U-Net 模型的代码实现：(贡献者：黄钦建 - 华南理工大学)

```
def get_unet():
    inputs = Input((img_rows, img_cols, 1))
    conv1 = Conv2D(32, (3, 3), activation='relu', padding='same')(inputs)
    conv1 = Conv2D(32, (3, 3), activation='relu', padding='same')(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
    # pool1 = Dropout(0.25)(pool1)
    # pool1 = BatchNormalization()(pool1)

    conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')(pool1)
    conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
    # pool2 = Dropout(0.5)(pool2)
    # pool2 = BatchNormalization()(pool2)

    conv3 = Conv2D(128, (3, 3), activation='relu', padding='same')(pool2)
    conv3 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv3)
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)
```

```
# pool3 = Dropout(0.5)(pool3)
# pool3 = BatchNormalization()(pool3)

conv4 = Conv2D(256, (3, 3), activation='relu', padding='same')(pool3)
conv4 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv4)
pool4 = MaxPooling2D(pool_size=(2, 2))(conv4)
# pool4 = Dropout(0.5)(pool4)
# pool4 = BatchNormalization()(pool4)

conv5 = Conv2D(512, (3, 3), activation='relu', padding='same')(pool4)
conv5 = Conv2D(512, (3, 3), activation='relu', padding='same')(conv5)

up6 = concatenate([Conv2DTranspose(256, (2, 2), strides=(
    2, 2), padding='same')(conv5), conv4], axis=3)
# up6 = Dropout(0.5)(up6)
# up6 = BatchNormalization()(up6)
conv6 = Conv2D(256, (3, 3), activation='relu', padding='same')(up6)
conv6 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv6)

up7 = concatenate([Conv2DTranspose(128, (2, 2), strides=(
    2, 2), padding='same')(conv6), conv3], axis=3)
# up7 = Dropout(0.5)(up7)
# up7 = BatchNormalization()(up7)
conv7 = Conv2D(128, (3, 3), activation='relu', padding='same')(up7)
conv7 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv7)

up8 = concatenate([Conv2DTranspose(64, (2, 2), strides=(
    2, 2), padding='same')(conv7), conv2], axis=3)
# up8 = Dropout(0.5)(up8)
# up8 = BatchNormalization()(up8)
conv8 = Conv2D(64, (3, 3), activation='relu', padding='same')(up8)
conv8 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv8)

up9 = concatenate([Conv2DTranspose(32, (2, 2), strides=(
```

Residual convolution unit 就是普通的去除了 BN 的 residual unit；

Multi-resolution fusion 是先对多输入的 feature map 都用一个卷积层进行 adaptation(都化到最小的 feature map 的 shape)，再上采样再做 element-wise 的相加。注意如果是像 RefineNet-4 那样的单输入 block 这一部分就直接 pass 了；

Chained residual pooling 中的 ReLU 对接下来池化的有效性很重要，还可以使模型对学习率的变化没这么敏感。这个链式结构能从很大范围区域上获取背景 context。另外，这个结构中大量使用了 identity mapping 这样的连接，无论长距离或者短距离的，这样的结构允许梯度从一个 block 直接向其他任一 block 传播。

Output convolutions 就是输出前再加一个 RCU。

### 8. 9.7 PSPNet

场景解析对于无限制的开放词汇和不同场景来说是具有挑战性的。本文使用文中的 pyramid pooling module 实现基于不同区域的上下文集成，提出了 PSPNet，实现利用上下文信息的能力进行场景解析。

作者认为，FCN 存在的主要问题是没有采取合适的策略来用全局的信息，本文的做法就是借鉴 SPPNet 来设计了 PSPNet 解决这个问题。

很多 State-of-the-art 的场景解析框架都是基于 FCN 的。基于 CNN 的方法能够增强动态物体的理解，但是在无限制词汇和不同场景中仍然面临挑战。举个例子，如下图。

FCN 认为右侧框中是汽车，但是实际上是船，如果参考上下文的先验知识，就会发现左边是一个船屋，进而推断是框中是船。FCN 存在的主要问题就是不能利用好全局的场景线索。

对于尤其复杂的场景理解，之前都是采用空间金字塔池化来做的，和之前方法不同（为什么不同，需要参考一下经典的金字塔算法），本文提出了 pyramid scene parsing network(PSPNet)。

本文的主要贡献如下：

- (1) 提出了 PSPNet 在基于 FCN 的框架中集成困难的上下文特征
- (2) 通过基于深度监督误差开发了针对 ResNet 的高效优化策略
- (3) 构建了一个用于 state-of-the-art 的场景解析和语义分割的实践系统（具体是什么？）

通过观察 FCN 的结果，发现了如下问题：

- (1) 关系不匹配 (Mismatched Relationship)
- (2) 易混淆的类别 (Confusion Categories)
- (3) 不显眼的类别 (Inconspicuous Classes)

总结以上结果发现，以上问题部分或者全部与上下文关系和全局信息有关系，因此本文提出了 PSPNet。框架如下：

## 9. 9.8 DeepLab 系列

**9.1. 9.8.1 DeepLabv1.** DeepLab 是结合了深度卷积神经网络 (DCNNs) 和概率图模型 (DenseCRFs) 的方法。

在实验中发现 DCNNs 做语义分割时精准度不够的问题，根本原因是 DCNNs 的高级特征的平移不变性，即高层次特征映射，根源在于重复的池化和下采样。

针对信号下采样或池化降低分辨率，DeepLab 是采用的 atrous (带孔) 算法扩展感受野，获取更多的上下文信息。

分类器获取以对象中心的决策是需要空间变换的不变性，这天然地限制了 DCNN 的定位精度，DeepLab 采用完全连接的条件随机场 (CRF) 提高模型捕获细节的能力。

除空洞卷积和 CRFs 之外，论文使用的 tricks 还有 Multi-Scale features。其实就是 U-Net 和 FPN 的思想，在输入图像和前四个最大池化层的输出上附加了两层的 MLP，第一层是 128 个  $3 \times 3$  卷积，第二层是 128 个  $1 \times 1$  卷积。最终输出的特征与主干网的最后一层特征图融合，特征图增加  $5 \times 128 = 640$  个通道。

实验表示多尺度有助于提升预测结果，但是效果不如 CRF 明显。

论文模型基于 VGG16，在 Titan GPU 上运行速度达到了 8FPS，全连接 CRF 平均推断需要 0.5s，在 PASCAL VOC-2012 达到 71.6% IOU accuracy。

**9.2. 9.8.2 DeepLabv2.** DeepLabv2 是相对于 DeepLabv1 基础上的优化。DeepLabv1 在三个方向努力解决，但是问题依然存在：特征分辨率的降低、物体存在多尺度，DCNN 的平移不变性。

因 DCNN 连续池化和下采样造成分辨率降低，DeepLabv2 在最后几个最大池化层中去除下采样，取而代之的是使用空洞卷积，以更高的采样密度计算特征映射。

物体存在多尺度的问题，DeepLabv1 中是用多个 MLP 结合多尺度特征解决，虽然可以提供系统的性能，但是增加特征计算量和存储空间。

论文受到 Spatial Pyramid Pooling (SPP) 的启发，提出了一个类似的结构，在给定的输入上以不同采样率的空洞卷积并行采样，相当于以多个比例捕捉图像的上下文，称为 ASPP (atrous spatial pyramid pooling) 模块。

DCNN 的分类不变形影响空间精度。DeepLabv2 是采样全连接的 CRF 在增强模型捕捉细节的能力。

论文模型基于 ResNet，在 NVidia Titan X GPU 上运行速度达到了 8FPS，全连接 CRF 平均推断需要 0.5s，在耗时方面和 DeepLabv1 无差异，但在 PASCAL VOC-2012 达到 79.7 mIOU。

**10.1. 9.9.1 Mask-RCNN 的网络结构示意图.** 其中黑色部分为原来的 Faster-RCNN, 红色部分为在 Faster 网络上的修改:

- 1) 将 ROI Pooling 层替换成了 ROIAlign;
- 2) 添加并列的 FCN 层 (Mask 层);

先来概述一下 Mask-RCNN 的几个特点 (来自于 Paper [Mask R-CNN](#) 的 Abstract):

- 1) 在边框识别的基础上添加分支网络, 用于语义 Mask 识别;
- 2) 训练简单, 相对于 Faster 仅增加一个小小的 Overhead, 可以跑到 5FPS;
- 3) 可以方便的扩展到其他任务, 比如人的姿态估计等;
- 4) 不借助 Trick, 在每个任务上, 效果优于目前所有的 single-model entries; 包括 COCO 2016 的 Winners。

**10.2. 9.9.2 RCNN 行人检测框架.** 来看下后面两种 RCNN 方法与 Mask 结合的示意图:

图中灰色部分是原来的 RCNN 结合 ResNet or FPN 的网络, 下面黑色部分为新添加的并联 Mask 层, 这个图本身与上面的图也没有什么区别, 旨在说明作者所提出的 Mask RCNN 方法的泛化适应能力: 可以和多种 RCNN 框架结合, 表现都不错。

### 10.3. 9.9.3 Mask-RCNN 技术要点.

#### 1. 技术要点 1 - 强化的基础网络

通过 ResNeXt-101+FPN 用作特征提取网络, 达到 state-of-the-art 的效果。

#### 2. 技术要点 2 - ROIAlign

采用 ROIAlign 替代 RoiPooling (改进池化操作)。引入了一个插值过程, 先通过双线性插值到  $14 \times 14$ , 再 pooling 到  $7 \times 7$ , 很大程度上解决了仅通过 Pooling 直接采样带来的 Misalignment 对齐问题。

PS: 虽然 Misalignment 在分类问题上影响并不大, 但在 Pixel 级别的 Mask 上会存在较大误差。

后面我们把结果对比贴出来 (Table2 c & d), 能够看到 ROIAlign 带来较大的改进, 可以看到, Stride 越大改进越明显。

#### 3. 技术要点 3 - Loss Function

每个 ROIAlign 对应  $K \times m^2$  维度的输出。K 对应类别个数, 即输出 K 个 mask, m 对应池化分辨率 ( $7 \times 7$ )。Loss 函数定义:

$$L_{mask}(Cl_{s_k}) = Sigmoid(Cl_{s_k})$$

$L_{mask}(Cl_{s_k}) = Sigmoid(Cl_{s_k})$ , 平均二值交叉熵 (average binary cross-entropy) Loss, 通过逐像素的 Sigmoid 计算得到。

### 11.2. 9.10.2 图像级别标记. 论文地址: Constrained Convolutional Neural Networks for Weakly Supervised Segmentation (ICCV 2015)

UC Berkeley 的 Deepak Pathak 使用了一个具有图像级别标记的训练数据来做弱监督学习。训练数据中只给出图像中包含某种物体，但是没有其位置信息和所包含的像素信息。该文章的方法将 image tags 转化为对 CNN 输出的 label 分布的限制条件，因此称为 Constrained convolutional neural network (CCNN)。

该方法把训练过程看作是有线性限制条件的最优化过程：

$$\underset{\theta, P}{\text{minimize}} \quad D(P(X) || Q(X|\theta)) \text{ subject to} \quad A \vec{P} \geq \vec{b}, \sum_X P(X) = 1$$

其中的线性限制条件来自于训练数据上的标记，例如一幅图像中前景类别像素个数期望值的上界或者下界（物体大小）、某个类别的像素个数在某图像中为 0，或者至少为 1 等。该目标函数可以转化为一个 loss function，然后通过 SGD 进行训练。

实验中发现单纯使用 Image tags 作为限制条件得到的分割结果还比较差，在 PASCAL VOC 2012 test 数据集上得到的 mIoU 为 35.6%，加上物体大小的限制条件后能达到 45.1%，如果再使用 bounding box 做限制，可以达到 54%。FCN-8s 可以达到 62.2%，可见弱监督学习要取得好的结果还是比较难。

### 11.3. 9.10.3 DeepLab+bounding box+image-level labels\*\*. 论文地址: Weakly-and Semi-Supervised Learning of a DCNN for Semantic Image Segmentation

Google 的 George Papandreou 和 UCLA 的 Liang-Chieh Chen 等在 DeepLab 的基础上进一步研究了使用 bounding box 和 image-level labels 作为标记的训练数据。使用了期望值最大化算法 (EM) 来估计未标记的像素的类别和 CNN 的参数。

对于 image-level 标记的数据，我们可以观测到图像的像素值和图像级别的标记，但是不知道每个像素的标号，因此把  $y$  当做隐变量。使用如下的概率图模式：

$$P(x, y, z; \theta) = P(x) \left( \prod_{m=1}^M P(y_m|x; \theta) \right) P(z|y)$$

这篇论文是通过 EM 算法来学习模型的参数  $\theta$ ，具体推导过程可参考原论文。

对于给出 bounding box 标记的训练图像，该方法先使用 CRF 对该训练图像做自动分割，然后在分割的基础上做全监督学习。通过实验发现，单纯使用图像级别的标记得到的分割效果较差，但是使用 bounding box 的训练数据可以得到较好的结果，在 VOC2012 test 数据集上得到 mIoU 62.2%。另外如果使用少量的全标记图像和大量的弱标记图像进行结合，可以得到与全监督学习 (70.3%) 接近的分割结果 (69.0%)。

该方法在 Siftflow 数据集上得到了比较好的结果，比 state-of-the-art 的结果提高了 10% 以上。

小结：在弱标记的数据集上训练图像分割算法可以减少对大量全标记数据的依赖，在大多数应用中会更加贴合实际情况。弱标记可以是图像级别的标记、边框和部分像素的标记等。训练的方法一般看做是限制条件下的最优化方法。另外 EM 算法可以用于 CNN 参数和像素类别的联合求优。

### 11.5. 9.10.5 弱监督分割最新进展（贡献者：明奇-北京理工大学）。

- **bbox 监督**

1. Learning to Segment via Cut-and-Paste (ECCV 2018)

利用 GAN 对抗学习的思想，在 cut-paste 思想指导下利用 bbox 弱监督进行实例分割。采用对抗学习的思想，网络主体分为两大部分：mask 生成器和合成图像判别器。具体过程为：(1) 在图像上截取 gt，经过特征提取后预测一个 bbox 内 gt 的 mask；(2) 在原图上随机 cut 一个背景图像，将 bbox 内按照生成的 mask 提取出物体分割结果，然后 paste 到原图裁剪的背景上去；(3) 合成的图像经过判别器进行真假判断。通过生成器生成更好 mask 来使得判别器更难判别，在对抗学习中提升两者的性能，逐渐获得更好的结果。

2. Simple Does It: Weakly Supervised Instance and Semantic Segmentation (CVPR2017) 本文做的是 bbox 弱监督语义/实例分割任务，能达到全监督分割效果 (DeepLabv1) 的 95%。主要工作为：讨论了使用弱监督语义标签进行迭代训练的方法，以及其限制和不足之处；证明了通过类似 GrabCut 的算法能通过 bbox 生成分割训练标签方法的可行性，可以避免像上面的迭代方法重新调整网络训练策略；在 VOC 数据集上逼近监督学习的分割任务效果。作者的启发是：将 bbox level 的 mask 送入网络训练后得到分割 mask 的比输入的 bbox mask 要好（这是很好的 insight）。因此启发的操作是：将 bbox level 标注作为初始 mask 输入优化，每次得到的标注作为 gt 进行下一轮的迭代，从而不断获得更好的效果。效果图如下：在此基础上，再加上优化的 GrabCut+ 算法，以及部分区域的筛选，以及 BSDS500 的边界预测信息整合到一起，能够达到很好的弱监督迭代分割效果。

- **分类监督**

1. Weakly Supervised Learning of Instance Segmentation with Inter-pixel Relations(CVPR2019)

使用分类标注作为弱监督信息，在 CAM 提取到特征的基础上，进一步设计 IRNet 学习额外的特征约束，从而到达更好的弱监督实例分割效果。为了解决 CAM 应用到实例分割的上述局限，设计 IRNet。其组成为两部分：(1) 不分类别的实例响应图 (2) pairwise semantic affinitie。其中通过不分类别的实例响应图和 CAM 结合，约束后得到 instance-wise CAMS；另一个分支预先预测物体的边界然后得到 pairwise semantic affinitie（关于

## 14. 9.13 全景分割（贡献者：北京理工大学-明奇）

全景分割的开山之作：何恺明的 *Panoptic Segmentation*

### 1. Introduction

语义分割通过带孔全卷积网络，根据不同的 stuff 进行划分；实例分割则是在目标检测的基础上基于检测框进行物体的分割。缺少一种框架可以将两者进行融合实现既能分割背景又能分割实例，而这在自动驾驶和 AR 技术中大有作为。由此提出的全景分割任务能将两者进行结合。

全景分割的思路很直观：为图像的每个像素分配语义 label 和类内实例 id，前者用于区分语义信息，后者用于分割实例（因此 stuff 不具有实例 id）。提出全景分割时，只是启发式地将语意分割和实例分割两种任务的输出进行后处理的融合（如 NMS），并以此建立 PS 任务的 baseline。为了评价全景分割的质量，提出 panoptic quality (PQ) 标准，将背景和物体的评价纳入一个完整的框架下。示意图如下：

### 2. Panoptic Segmentation

#### • Task format

全景分割的标注方法：

像素级的标注，标出类别 label 和类内实例 id。如果某像素的这两个信息都能匹配，则可以将该像素匹配到某个类别和实例中去；类外的像素可以分配空标签，即并不是所有的像素都要有语义类别。

#### • Stuff and thing labels

对于 stuff 和 thing（背景填充和物体实例）的标签，交集是空集，并集是所有可能的 label 空间。这两者是互相独立不相关的（很好理解，像素属于那个类和它属于哪个物体不具有相关性）。

#### • Relationship

都是像素级的 label，需要为每个像素分配对应的标签。但是实例分割基于 region 的，允许重叠的 segmentation，而全景分割和语义分割一样是像素级的 label，不允许重叠标签的出现。

#### • Confidence scores

这一点上更像语义分割而不是实例分割，对于 PS 不需要置信分数评价分割质量。提到这个，作者认为语义分割和全景分割可以直接利用人工标注的 label 进行对比从而评价当前 mask 的质量；而实例分割在选择 mask 时评价的是分类置信度，这个并没有人工标注进行参考，因此难以把握。

### 3. Panoptic Segmentation Metric

用于衡量全景分割效果的指标应具有：完备性；可解释性；简洁性。于是提出了 PQ 指标，可分为两步：分割匹配、在匹配上进行计算 PQ。

#### 3.1 Segment Matching

定义 match：预测的 segmentation 和 gt 的 iou 大于 0.5，说明两者 can match。再结合全景分割的不可重叠性，不难得到：最多只有一个预测的 segmentation 可以 match gt。

#### 3.2 PQ Computation

PQ 的计算类似 mAP，也是类内求取，然后求类间的平均值，以便不敏感类别不平衡。对于每一类，可以根据 gt 与预测的 segmentation 分为三类（下图描述）：

TP: 预测为正，实际为正，描述 match 较好的

FP: 预测为正，实际为负，描述 match 错的

FN: 预测为负，实际为正，描述没 match 出来的 gt

通过上述三类可以计算得到 PQ 值公式：

式中出去 FP 与 FN 后，剩下的式子描述的是 match 的 segmentation 的平均 IoU，加上 FP 与 FN 是为了惩罚 match 失败的分割实例。

有意思的是，对上述式子进行简单的恒等变化：

第一项评价的是 match 分割的质量，第二项类似于 F1 得分。因此可以 PQ 分解为：

$$PQ = SQ * RQ$$

##### • Void labels

gt 中可能出现两种像素标注为空的情况：超出类别的像素和模糊不清的像素（难以分类）。在评估结果时，这些空的标签不予以评估。具体而言：

- (1) 在 matching 部分，预测出为 void 的像素会被移出 prediction 并不参与 IoU 计算；
- (2) matching 后，unmatched prediction 按照一般情况会计算 FP FN，但是对于空标签情况，如果该 prediction 含有的 void 像素块超过一定匹配阈值就会被移除，并不算作 FP 计算得分。

##### • Group labels

有时区分相同语义类别的实例个体标注比较困难，因此有提出组标签的标注方法。但对于 PQ 计算而言：

- (1) matching 部分不使用组标签，而是严格区分实例
- (2) matching 后，对于包含一部分相同类别像素点的 unmatched predicted segments，这一部分将被去除并不视作 false positives

### 3.3 Comparison to Existing Metrics \* Semantic segmentation metrics

衡量语义分割的标准有像素级精度，平均精度，IoU。但是其只专注于像素级的划分，不能反映物体实例级别的分割性能。

- **Instance segmentation metrics**

度量为 AP，主要是引入了置信度分数 confidence score 对检测目标进行打分。（两者不是完全的隔绝，实例分割也有用 IoU 监督的，而 confidence score 是否能够反映 mask 的真实质量也有存疑过，这个标准也不是固定的）

- **Panoptic quality**

PQ 的度量可以分解成 SQ 和 RQ，SQ 反映了语义分割的像素级 IoU 性能，RQ 专注于检测识别的效果，因此将两者统一到一个框架下。

分割效果：

TODO

- 图像分割数据集标注工具
- 图像分割评价标准
- 全景分割
- UNet++







**3.1. 强化学习和监督式学习的区别:** 监督式学习就好比你在学习的时候，有一个导师在旁边指点，他知道怎么是对的怎么是错的，但在很多实际问题中，例如 chess, go，这种有成千上万种组合方式的情况，不可能有一个导师知道所有可能的结果。

而这时，强化学习会在没有任何标签的情况下，通过先尝试做出一些行为得到一个结果，通过这个结果是对还是错的反馈，调整之前的行为，就这样不断的调整，算法能够学习到在什么样的情况下选择什么样的行为可以得到最好的结果。

就好比你有一只还没有训练好的小狗，每当它把屋子弄乱后，就减少美味食物的数量（惩罚），每次表现不错时，就加倍美味食物的数量（奖励），那么小狗最终会学到一个知识，就是把客厅弄乱是不好的行为。

两种学习方式都会学习出输入到输出的一个映射，监督式学习出的是之间的关系，可以告诉算法什么样的输入对应着什么样的输出，强化学习出的是给机器的反馈 reward function，即用来判断这个行为是好是坏。另外强化学习的结果反馈有延时，有时候可能需要走了很多步以后才知道以前的某一步的选择是好还是坏，而监督学习做了比较坏的选择会立刻反馈给算法。

而且强化学习面对的输入总是在变化，每当算法做出一个行为，它影响下一次决策的输入，而监督学习的输入是独立同分布的。

通过强化学习，一个 agent 可以在探索和开发 (exploration and exploitation) 之间做权衡，并且选择一个最大的回报。

exploration 会尝试很多不同的事情，看它们是否比以前尝试过的更好。

exploitation 会尝试过去经验中最有效的行为。

一般的监督学习算法不考虑这种平衡，就只是是 exploitative。

**3.2. 强化学习和非监督式学习的区别:** 非监督式不是学习输入到输出的映射，而是模式。例如在向用户推荐新闻文章的任务中，非监督式会找到用户先前已经阅读过类似的文章并向他们推荐其一，而强化学习将通过向用户先推荐少量的新闻，并不断获得来自用户的反馈，最后构建用户可能会喜欢的文章的“知识图”。

对非监督学习来说，它通过对没有概念标记的训练例进行学习，以发现训练例中隐藏的结构性知识。这里的训练例的概念标记是不知道的，因此训练样本的歧义性最高。对强化学习来说，它通过对没有概念标记、但与一个延迟奖赏或效用（可视为延迟的概念标记）相关联的训练例进行学习，以获得某种从状态到行动的映射。这里本来没有概念标记的概念，但延迟奖赏可被视为一种延迟概念标记，因此其训练样本的歧义性介于监督学习和非监督学习之间。

除了上述深度强化学习算法，还有深度迁移强化学习、分层深度强化学习、深度记忆强化学习以及多智能体强化学习等算法。## 10.5 深度迁移强化学习算法传统深度强化学习算法每次只能解决一种游戏任务，无法在一次训练中完成多种任务。迁移学习和强化学习的结合也是深度强化学的一种主要思路。

Parisotto 等提出了一种基于行为模拟的深度迁移强化学习算法。该算法通过监督信号的指导，使得单一的策略网络学习各自的策略，并将知识迁移到新任务中。Rusa 等提出策略蒸馏(policy distillation)深度迁移强化学习算法。策略蒸馏算法中分为学习网络和指导网络，通过这两个网络 Q 值的偏差来确定目标函数，引导学习网络逼近指导网络的值函数空间。此后，Rusa 等又提出了一种基于渐进神经网络 (progressive neural networks, PNN) 的深度迁移强化学习算法。PNN 是一种把神经网络和神经网络连起来的算法。它在一系列序列任务中，通过渐进的方式来存储知识和提取特征，完成了对知识的迁移。PNN 最终实现多个独立任务的训练，通过迁移加速学习过程，避免灾难性遗忘。Fernando 等提出了路径网络 (PathNet)[45]。PathNet 可以说是 PNN 的进阶版。PathNet 把网络中每一层都看作一个模块，把构建一个网络看成搭积木，也就是复用积木。它跟 PNN 非常类似，只是这里不再有列，而是不同的路径。PathNet 将智能体嵌入到神经网络中，其中智能体的任务是为新任务发现网络中可以复用的部分。智能体是网络之中的路径，其决定了反向传播过程中被使用和更新的参数范围。在一系列的 Atari 强化学习任务上，PathNet 都实现了正迁移，这表明 PathNet 在训练神经网络上具有通用性应用能力。PathNet 也可以显著提高 A3C 算法超参数选择的鲁棒性。Schaul 等提出了一种通用值函数逼近器 (universal value function approximators, UVFAs) 来泛化状态和目标空间。UVFAs 可以将学习到的知识迁移到环境动态特性相同但目标不同的新任务中。## 10.6 分层深度强化学习算法分层强化学习可以将最终目标分解为多个子任务来学习层次化的策略，并通过组合多个子任务的策略形成有效的全局策略。Kulkarni 等提出了分层 DQN(hierarchical deep Q-network, h-DQN) 算法。h-DQN 基于时空抽象和内在激励分层，通过在不同的时空尺度上设置子目标对值函数进行层次化处理。顶层的值函数用于确定宏观决策，底层的值函数用于确定具体行动。Krishnamurthy 等在 h-DQN 的基础上提出了基于内部选择的分层深度强化学习算法。该模型结合时空抽象和深度神经网络，自动地完成子目标的学习，避免了特定的内在激励和人工设定中间目标，加速了智能体的学习进程，同时也增强了模型的泛化能力。Kulkarni 等基于后续状态表示法提出了深度后续强化学习 (deep successor reinforcement learning, DSRL)。DSRL 通过阶段性地分解子目标和学习子目标策略，增强了对未知状态空间的探索，使得智能体更加适应那些存在延迟反馈的任务。Vezhnevets 等受封建 (feudal) 强化学习算法的启发，提出一种分层深度强化学习的架构 FeUDal 网络 (FuNs)[49]。FuNs 框架使用一个管理员模块和一个工人模块。管理员模块在较低的时间分辨率下工作，设置抽象目标并传递给工人模块去执行。FuNs 框架创造了一个稳定的自然层次结构，并且允许两个模块以互补的方式学习。实验证明，FuNs 有助于处理长期信用分配和记忆任务，在 Atari 视频游戏和迷宫游戏中都取得了不错的效果。

## 10.7 深度记忆强化学习算法传统的深度强化学习模型不具备记忆、认知、推理等高层次的能力, 尤其是在面对状态部分可观察和延迟奖赏的情形时. Junhyuk 等通过在传统的深度强化学习模型中加入外部的记忆网络部件和反馈控制机制, 提出反馈递归记忆 Q 网络 (feedback recurrent memory Q-network, FRMQN). FRMQN 模型具备了一定的记忆与推理功能, 通过反馈控制机制, FRMQN 整合过去存储的有价值的记忆和当前时刻的上下文状态, 评估动作值函数并做出决策. FRMQN 初步模拟了人类的主动认知与推理能力, 并完成了一些高层次的认知任务. 在一些未经过训练的任务中, FRMQN 模型表现出了很强的泛化能力. Blundell 等设计出一种模型无关的情节控制算法 (model-free episode control, MFEC). MFEC 可以快速存储和回放状态转移序列, 并将回放的序列整合到结构化知识系统中, 使得智能体在面对一些复杂的决策任务时, 能快速达到人类玩家的水平. MFEC 通过反向经验回放, 使智能体拥有初步的情节记忆. 实验表明, 基于 MFEC 算法的深度强化学习不仅可以在 Atari 游戏中学习除有效策略, 还可以处理一些三维场景的复杂任务. Pritzel 等在 MFEC 的基础上进一步提出了神经情节控制 (neural episodic control, NEC), 有效提高了深度强化学习智能体的记忆能力和学习效率 [53]. NEC 能快速吸收新经验并依据新经验来采取行动. 价值函数包括价值函数渐变状态表示和价值函数快速更新估计两部分. 大量场景下的研究表明, NEC 的学习速度明显快于目前最先进的通用深度强化学习智能体. ## 10.8 多智能体深度强化学习算法在一些复杂场景中, 涉及到多智能体的感知决策问题, 这时需要将单一模型扩展为多个智能体之间相互合作、通信及竞争的多智能体深度强化学习系统. Foerster 等提出了一种称为分布式深度递归 Q 网络 (deep distributed recurrent Q-networks, DDRQN) 的模型, 解决了状态部分可观测状态下的多智能体通信与合作的挑战性难题 [54]. 实验表明, 经过训练的 DDRQN 模型最终在多智能体之间达成了一致的通信协议 1536 控制理论与应用第 34 卷议, 成功解决了经典的红蓝帽子问题. 让智能体学会合作与竞争一直以来都是人工智能领域内的一项重要研究课题, 也是实现通用人工智能的必要条件. Lowe 等提出了一种用于合作-竞争混合环境的多智能体 actor-critic 算法 (multi-agent deepdeterministic policy gradient, MADDPG) [55]. MADDPG 对 DDPG 强化学习算法进行了延伸, 可实现多智能体的集中式学习和分布式执行, 让智能体学习彼此合作和竞争. 在多项测试任务中, MADDPG 的表现都优于 DDPG. ## 10.9 强化学习开源框架谷歌 TensorFlow Agents —TensorFlow 的加强版, 它提供许多工具, 通过强化学习可以实现各类智能应用程序的构建与训练. 这个框架能够将 OpenAI Gym 接口扩展至多个并行环境, 并允许各代理立足 TensorFlow 之内实现以执行批量计算. 其面向 OpenAI Gym 环境的批量化接口可与 TensorFlow 实现全面集成, 从而高效执行各类算法. 该框架还结合有 BatchPPO, 一套经过优化的近端策略优化算法实现方案. 其核心组件包括一个环境打包器, 用于在外部过程中构建 OpenAI Gym 环境; 一套批量集成, 用于实现 TensorFlow 图步并以强化学习运算的方式重置函数; 外加用于将 TensorFlow 图形批处理流程与强化学习算法纳入训练特内单一却步的组件.

Roboschool: Roboschool 提供开源软件以通过强化学习构建并训练机器人模拟。其有助于在同一环境当中对多个代理进行强化学习训练。通过多方训练机制，您可以训练同一代理分别作为两方玩家（因此能够自我对抗）、使用相同算法训练两套代理，或者设置两种算法进行彼此对抗。Roboschool 由 OpenAI 开发完成，这一非营利性组织的背后赞助者包括 Elon Musk、Sam Altman、Reid Hoffman 以及 Peter Thiel。其与 OpenAI Gym 相集成，后者是一套用于开发及评估强化学习算法的开源工具集。OpenAI Gym 与 TensorFlow、Theano 以及其它多种深度学习库相兼容。OpenAI Gym 当中包含用于数值计算、游戏以及物理引擎的相关代码。Roboschool 基于 Bullet 物理引擎，这是一套开源许可物理库，并被其它多种仿真软件——例如 Gazebo 与 Virtual Robot Experimentation Platform（简称 V-REP）所广泛使用。其中包含多种强化学习算法，具体以怨报德异步深度强化学习方法、Actor-Critic with Experience Replay、Actor- Critic using Kronecker-Factored Trust Region、深度确定性策略梯度、近端策略优化以及信任域策略优化等等。

Coach：英特尔公司的开源强化学习框架，可以对游戏、机器人以及其它基于代理的智能应用进行智能代理的建模、训练与评估。Coach 提供一套模块化沙箱、可复用组件以及用于组合新强化学习算法并在多种应用领域内训练新智能应用的 Python API。该框架利用 OpenAI Gym 作为主工具，负责与不同强化学习环境进行交换。其还支持其它外部扩展，其中包括 Roboschool、gym-extensions、PyBullet 以及 ViZDoom。Coach 的环境打包器允许用户向其中添加自定义强化学习环境，从而解决其它学习问题。该框架能够在桌面计算机上高效训练强化学习代理，并利用多核 CPU 处理相关任务。其能够为一部分强化学习算法提供单线程与多线程实现能力，包括异步优势 Actor-Critic、深度确定性策略梯度、近端策略优化、直接未来预测以及规范化优势函数。所有算法皆利用面向英特尔系统作出优化的 TensorFlow 完成，其中部分算法亦适用于英特尔的 Neon 深度学习框架。Coach 当中包含多种强化学习代理实现方案，其中包括从单线程实现到多线程实现的转换。其能够开发出支持单与多工作程序（同步或异步）强化学习实现方法的新代理。此外，其还支持连续与离散操作空间，以及视觉观察空间或仅包含原始测量指标的观察空间。## 10.10 深度强化学习算法小结基于值函数概念的 DQN 及其相应的扩展算法在离散状态、离散动作的控制任务中已经表现了卓越的性能，但是受限于值函数离散型输出的影响，在连续型控制任务上显得捉襟见肘。基于策略梯度概念的，以 DDPG, TRPO 等为代表的策略型深度强化学习算法则更适用于处理基于连续状态空间的连续动作的控制输出任务，并且算法在稳定性和可靠性上具有一定的理论保证，理论完备性较强。采用 actor-critic 架构的 A3C 算法及其扩展算法，相比于传统 DQN 算法，这类算法的数据利用效率更高，学习速率更快，通用性、可扩展应用性更强，达到的表现性能更优，但算法的稳定性无法得到保证。而其他的如深度迁移强化学习、分层深度强化学习、深度记忆强化学习和多智能体深度强化学习等算法都是现在的研究热点，通过这些算法能应对更为复杂的场景问题、系统环境及控制任务，是目前深度强化学习算法研究的前沿领域。

展望未来，人工智能开发者们需要尽可能掌握上述框架以及其中所使用的各类强化学习算法。此外，还需要强化自身对于多代理强化学习架构的理解，因为其中多种框架都大量利用前沿博弈论研究成果。最后，还需要熟悉深度强化学习知识。







## 迁移学习

本章主要简明地介绍了迁移学习的基本概念、迁移学习的必要性、研究领域和基本方法。重点介绍了几大类常用的迁移学习方法：数据分布自适应方法、特征选择方法、子空间学习方法、以及目前最热门的深度迁移学习方法。除此之外，我们也结合最近的一些研究成果对未来迁移学习进行了一些展望。并提供了一些迁移学习领域的常用学习资源，以方便感兴趣的读者快速开始学习。

### 1. 迁移学习基础知识

**1.1. 什么是迁移学习？** 找到目标问题的相似性，迁移学习任务就是从相似性出发，将旧领域 (domain) 学习过的模型应用在新领域上。

#### 1.2. 为什么需要迁移学习？

1. **大数据与少标注的矛盾：** 虽然有大量的数据，但往往都是没有标注的，无法训练机器学习模型。人工进行数据标注太耗时。
2. **大数据与弱计算的矛盾：** 普通人无法拥有庞大的数据量与计算资源。因此需要借助于模型的迁移。
3. **普适化模型与个性化需求的矛盾：** 即使是在同一个任务上，一个模型也往往难以满足每个人的个性化需求，比如特定的隐私设置。这就需要在不同人之间做模型的适配。
4. **特定应用（如冷启动）的需求。**

**1.3. 迁移学习的基本问题有哪些？** 基本问题主要有 3 个：

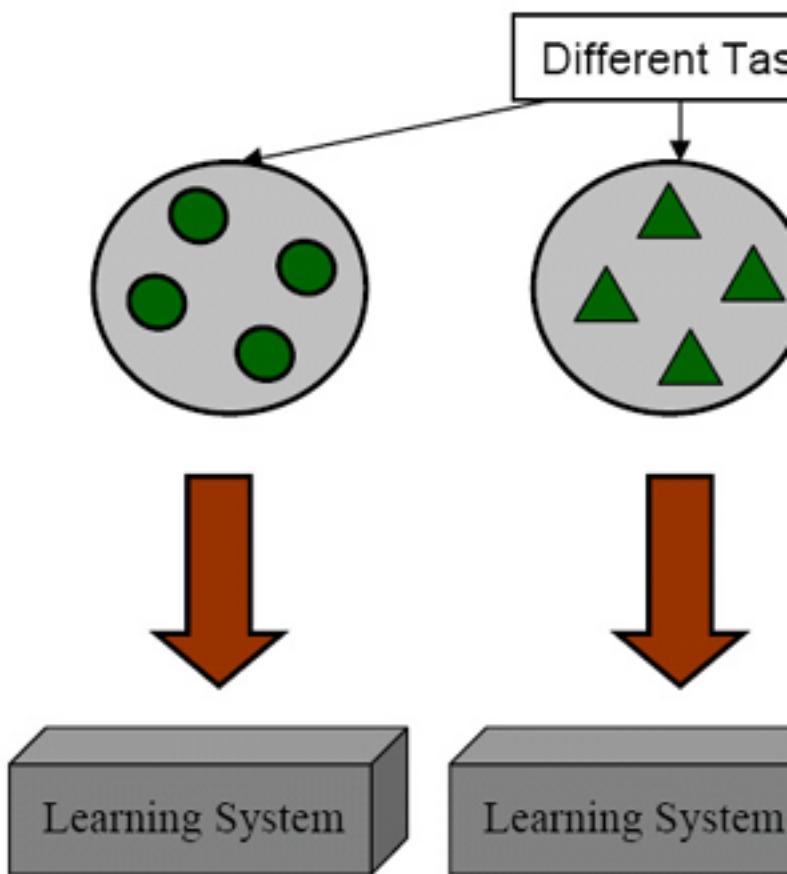
- **How to transfer:** 如何进行迁移学习？(设计迁移方法)
- **What to transfer:** 给定一个目标领域，如何找到相对应的源领域，然后进行迁移？(源领域选择)
- **When to transfer:** 什么时候可以进行迁移，什么时候不可以？(避免负迁移)

#### 1.4. 迁移学习有哪些常用概念？

- 基本定义
- **域 (Domain):** 数据特征和特征分布组成，是学习的主体
  - **源域 (Source domain):** 已有知识的域

	迁移学习	传统机器学习
数据分布	训练和测试数据不需要同分布	训练和测试数据同分布
数据标签	不需要足够的数据标注	足够的数据标注
建模	可以重用之前的模型	每个任务分别建模

## Learning Process of Traditional Machine Learning



## (a) Traditional Machine Learning

### 1.5. 迁移学习与传统机器学习有什么区别？

#### # ## 11.1.6 迁移学习的核心及度量准则？

迁移学习的总体思路可以概括为：开发算法来最大限度地利用有标注的领域的知识，来辅助目标领域的知识获取和学习。

迁移学习的核心是：找到源领域和目标领域之间的相似性，并加以合理利用。这种相似性非常普遍。比如，不同人的身体构造是相似的；自行车和摩托车的骑行方式是相似的；国际象

棋和中国象棋是相似的；羽毛球和网球的打球方式是相似的。这种相似性也可以理解为不变量。以不变应万变，才能立于不败之地。

**有了这种相似性后，下一步工作就是，如何度量和利用这种相似性。**度量工作的目标有两点：一是很好地度量两个领域的相似性，不仅定性地告诉我们它们是否相似，更定量地给出相似程度。二是以度量为准则，通过我们所要采用的学习手段，增大两个领域之间的相似性，从而完成迁移学习。

**一句话总结：相似性是核心，度量准则是重要手段。**

### 1.6. 迁移学习与其他概念的区别？

1. 迁移学习与多任务学习关系：

- **多任务学习**：多个相关任务一起协同学习；
- **迁移学习**：强调信息复用，从一个领域 (domain) 迁移到另一个领域。

2. 迁移学习与领域自适应：**领域自适应**：使两个特征分布不一致的 domain 一致。

3. 迁移学习与协方差漂移：**协方差漂移**：数据的条件概率分布发生变化。

Reference:

1. 王晋东, [迁移学习简明手册](#)

2. Ben-David, S., Blitzer, J., Crammer, K., Kulesza, A., Pereira, F., & Vaughan, J. W. (2010). A theory of learning from different domains. *Machine learning*, 79(1-2), 151-175.

3. Tan, B., Song, Y., Zhong, E. and Yang, Q., 2015, August. Transitive transfer learning. In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 1155-1164). ACM.

**1.7. 什么是负迁移？产生负迁移的原因有哪些？**负迁移 (Negative Transfer) 指的是，在源域上学习到的知识，对于目标域上的学习产生负面作用。

产生负迁移的原因主要有：- 数据问题：源域和目标域压根不相似，谈何迁移？- 方法问题：源域和目标域是相似的，但是，迁移学习方法不够好，没找到可迁移的成分。

负迁移给迁移学习的研究和应用带来了负面影响。在实际应用中，找到合理的相似性，并且选择或开发合理的迁移学习方法，能够避免负迁移现象。

**1.8. 迁移学习的基本思路？**迁移学习的总体思路可以概括为：开发算法来最大限度地利用有标注的领域的知识，来辅助目标领域的知识获取和学习。

1. 找到目标问题的相似性，迁移学习任务就是从相似性出发，将旧领域 (domain) 学习过的模型应用在新领域上。
2. 迁移学习，是指利用数据、任务、或模型之间的相似性，将在旧领域学习过的模型，应用于新领域的一种学习过程。

图 33: 深度网络迁移实验结果 1

实是到了这一步, feature 变得越来越 specific, 所以下降了。那对于第 6 第 7 层为什么精度又不变了? 那是因为, 整个网络就 8 层, 我们固定了第 6 第 7 层, 这个网络还能学什么呢? 所以很自然地, 精度和原来的 B 网络几乎一致!

对 BnB+ 来说, 结果基本上都保持不变。说明 finetune 对模型结果有着很好的促进作用!

我们重点关注 AnB 和 AnB+。对 AnB 来说, 直接将 A 网络的前 3 层迁移到 B, 貌似不会有影响, 再一次说明, 网络的前 3 层学到的几乎都是 general feature! 往后, 到了第 4 第 5 层的时候, 精度开始下降, 我们直接说: 一定是 feature 不 general 了! 然而, 到了第 6 第 7 层, 精度出现了小小的提升后又下降, 这又是为什么? 作者在这里提出两点 co-adaptation 和 feature representation。就是说, 第 4 第 5 层精度下降的时候, 主要是由于 A 和 B 两个数据集的差异比较大, 所以会下降; 至于第 6 第 7 层, 由于网络几乎不迭代了, 学习能力太差, 此时 feature 学不到, 所以精度下降得更厉害。

再看 AnB+。加入了 finetune 以后, AnB+ 的表现对于所有的 n 几乎都非常好, 甚至比 baseB (最初的 B) 还要好一些! 这说明: finetune 对于深度迁移有着非常好的促进作用!

把上面的结果合并就得到了下面一张图 (图34):

至此, AnB 和 BnB 基本完成。作者又想, 是不是我分 A 和 B 数据的时候, 里面存在一些比较相似的类使结果好了? 比如说 A 里有猫, B 里有狮子, 所以结果会好? 为了排除这些影响, 作者又分了一下数据集, 这次使得 A 和 B 里几乎没有相似的类别。在这个条件下再做 AnB, 与原来精度比较 (0% 为基准) 得到了下图 (图35) :

这个图说明了什么呢? 简单: 随着可迁移层数的增加, 模型性能下降。但是, 前 3 层仍然还是可以迁移的! 同时, 与随机初始化所有权重比较, 迁移学习的精度是很高的! 总之:

- 深度迁移网络要比随机初始化权重效果好;
- 网络层数的迁移可以加速网络的学习和优化。

### 3.9. 什么是深度网络自适应?. 基本思路

深度网络的 finetune 可以帮助我们节省训练时间, 提高学习精度。但是 finetune 有它的先天不足: 它无法处理训练数据和测试数据分布不同的情况。而这一现象在实际应用中比比皆是。因为 finetune 的基本假设也是训练数据和测试数据服从相同的数据分布。这在迁移学习中也是不成立的。因此, 我们需要更进一步, 针对深度网络开发出更好的方法使之更好地完成迁移学习任务。

以我们之前介绍过的数据分布自适应方法为参考, 许多深度学习方法 [Tzeng et al.,2014, Long et al.,2015a] 都开发出了自适应层 (AdaptationLayer) 来完成源域和目标域数据的自适应。自适应能够使得源域和目标域的数据分布更加接近, 从而使得网络的效果更好。

从上述的分析我们可以得出, 深度网络的自适应主要完成两部分的工作:







## 网络搭建及训练

### 1. TensorFlow

#### 2. TensorFlow 是什么？

TensorFlow 支持各种异构平台，支持多 CPU/GPU、服务器、移动设备，具有良好的跨平台的特性；TensorFlow 架构灵活，能够支持各种网络模型，具有良好的通用性；此外，TensorFlow 架构具有良好的可扩展性，对 OP 的扩展支持，Kernel 特化方面表现出众。

TensorFlow 最初由 Google 大脑的研究员和工程师开发出来，用于机器学习和神经网络方面的研究，于 2015.10 宣布开源，在众多深度学习框架中脱颖而出，在 Github 上获得了最多的 Star 量。

#### 3. TensorFlow 的设计理念是什么？

TensorFlow 的设计理念主要体现在两个方面：

(1) 将图定义和图运算完全分开。TensorFlow 被认为是一个“符号主义”的库。我们知道，编程模式通常分为命令式编程 (imperative style programming) 和符号式编程 (symbolic style programming)。命令式编程就是编写我们理解的通常意义上的程序，很容易理解和调试，按照原有逻辑执行。符号式编程涉及很多的嵌入和优化，不容易理解和调试，但运行速度相对有所提升。现有的深度学习框架中，Torch 是典型的命令式的，Caffe、MXNet 采用了两种编程模式混合的方法，而 TensorFlow 完全采用符号式编程。

符号式计算一般是先定义各种变量，然后建立一个数据流图，在数据流图中规定各个变量间的计算关系，最后需要对据流图进行编译，但此时的数据流图还是一个空壳儿，里面没有任何实际数据，只有把需要运算的输入放进去后，才能在整个模型中形成数据流，从而形成输出值。

例如：

```
t = 8 + 9
print(t)
```

在传统的程序操作中，定义了 t 的运算，在运行时就执行了，并输出 17。而在 TensorFlow 中，数据流图中的节点，实际上对应的是 TensorFlow API 中的一个操作，并没有真正去运行：

```
import tensorflow as tf
t = tf.add(8,9)
print(t)

#输出 Tensor{"Add_1:0",shape={},dtype=int32}
```

(2) TensorFlow 中涉及的运算都要放在图中，而图的运行只发生在会话（session）中。开启会话后，就可以用数据去填充节点，进行运算；关闭会话后，就不能进行计算了。因此，会话提供了操作运行和 Tensor 求值的环境。

例如：

```
import tensorflow as tf
#创建图
a = tf.constant([4.0,5.0])
b = tf.constant([6.0,7.0])
c = a * b
#创建会话
sess = tf.Session()
#计算c
print(sess.run(c)) #进行矩阵乘法，输出[24.,35.]
sess.close()
```

#### 4. TensorFlow 特点有哪些？

**4.1. 高度的灵活性。** TensorFlow 并不仅仅是一个深度学习库，只要可以把你计算过程表示成一个数据流图的过程，我们就可以使用 TensorFlow 来进行计算。TensorFlow 允许我们用计算图的方式建立计算网络，同时又可以很方便的对网络进行操作。用户可以基于 TensorFlow 的基础上用 python 编写自己的上层结构和库，如果 TensorFlow 没有提供我们需要的 API 的，我们也可以自己编写底层的 C++ 代码，通过自定义操作将新编写的功能添加到 TensorFlow 中。

**4.2. 真正的可移植性。** TensorFlow 可以在 CPU 和 GPU 上运行，可以在台式机、服务器、移动设备上运行。你想在你的笔记本上跑一下深度学习的训练，或者又不想修改代

**5.1. 整个系统从底层到上层可分为七层:** 设备层: 硬件计算资源, 支持 CPU、GPU

网络层: 支持两种通信协议

数值计算层: 提供最基础的计算, 有线性计算、卷积计算

高维计算层: 数据的计算都是以数组的形式参与计算

计算图层: 用来设计神经网络的结构

工作流层: 提供轻量级的框架调用

构造层: 最后构造的深度学习网络可以通过 TensorBoard 服务端可视化

## 6. TensorFlow 编程模型是怎样的?

TensorFlow 的编程模型: 让向量数据在计算图里流动。那么在编程时至少有这几个过程:

1. 构建图, 2. 启动图, 3. 给图输入数据并获取结果。

**6.1. 构建图.** TensorFlow 的图的类型是 `tf.Graph`, 它包含着计算节点和 `tensor` 的集合。

这里引用了两个新概念: `tensor` 和计算节点。我们先介绍 `tensor`, 一开始我们就介绍了, 我们需要把数据输入给启动的图才能获取计算结果。那么问题来了, 在构建图时用什么表示中间计算结果? 这个时候 `tensor` 的概念就需要引入了。类型是 `tf.Tensor`, 代表某个计算节点的输出, 一定要看清楚是“代表”。它主要有两个作用:

1. 构建不同计算节点之间的数据流
2. 在启动图时, 可以设置某些 `tensor` 的值, 然后获取指定 `tensor` 的值。这样就完成了计算的输入输出功能。

如下代码所示:

```
inImage = tf.placeholder(tf.float32,[32,32,3],"inputImage")
processedImage = tf.image.per_image_standardization(inImage,"processedImage")
```

这里 `inImage` 和 `processedImage` 都是 `tensor` 类型。它们代表着计算节点输出的数据, 数据的值具体是多少在启动图的时候才知道。上面两个方法调用都传递了一个字符串, 它是计算节点的名字, 最好给节点命名, 这样我们可以在图上调用 `get_tensor_by_name(name)` 获取对应的 `tensor` 对象, 十分方便。`(tensor 名字为 ":" )`

创建 `tensor` 时, 需要指定类型和 `shape`。对不同 `tensor` 进行计算时要求类型相同, 可以使用 `tf.cast` 进行类型转换。同时也要求 `shape` (向量维度) 满足运算的条件, 我们可以使用 `tf.reshape` 改变 `shape`。

现在了解计算节点的概念, 其功能是对 `tensor` 进行计算、创建 `tensor` 或进行其他操作, 类型是 `tf.Operation`。获取节点对象的方法为 `get_operation_by_name(name)`。

构建图, 如下代码:

```

g=tf.Graph()

with g.as_default():
    input_data=tf.placeholder(tf.float32,[None,2],"input_data")
    input_label=tf.placeholder(tf.float32,[None,2],"input_label")

    W1=tf.Variable(tf.truncated_normal([2,2]),name="W1")
    B1=tf.Variable(tf.zeros([2]),name="B1")

    output=tf.add(tf.matmul(input_data,W1),B1,name="output")
    cross_entropy=tf.nn.softmax_cross_entropy_with_logits(logits=output,labels=input_label)

    train_step=tf.train.AdamOptimizer().minimize(cross_entropy,name="train_step")

    initer=tf.global_variables_initializer()

```

上面的代码中我们创建了一个图，并在上面添加了很多节点。我们可以通过调用 `get_default_graph()` 获取默认的图。

`Input_data`,`input_label`,`W1`,`B1`,`output`,`cross_entropy` 都是 `tensor` 类型, `train_step`, `initer`, 是节点类型。

有几类 `tensor` 或节点比较重要，下面介绍一下：

`placeholder`. Tensorflow, 顾名思义, `tensor` 代表张量数据, `flow` 代表流, 其最初的设计理念就是构建一张静态的数据流图。图是有各个计算节点连接而成, 计算节点之间流动的便是中间的张量数据。要想让张量数据在我们构建的静态计算图中流动起来, 就必须有最初的输入数据流。而 `placeholder`, 翻译过来叫做占位符, 顾名思义, 是给我们的输入数据提供一个接口, 也就是说我们的一切输入数据, 例如训练样本数据, 超参数数据等都可以通过占位符接口输送到数据流图之中。使用实例如下代码:

```

import tensorflow as tf
x = tf.placeholder(dtype=tf.float32,shape=[],name='x')
y = tf.placeholder(dtype=tf.float32,shape=[],name='y')
z = x*y
with tf.Session() as sess:
    prod = sess.run(z,feed_dict={x:1.,y:5.2})
    print(prod)

```

[out]:5.2

2. variable. 无论是传统的机器学习算法，例如线性支持向量机（Support Vector Machine, SVM），其数学模型为  $y = w \cdot x + b$ ，还是更先进的深度学习算法，例如卷积神经网络（Convolutional Neural Network, CNN）单个神经元输出的模型  $y = w^T x + b$ 。可以看到， $w$  和  $b$  就是我们要求的模型，模型的求解是通过优化算法（对于 SVM，使用 SMO[1] 算法，对于 CNN，一般基于梯度下降法）来一步一步更新  $w$  和  $b$  的值直到满足停止条件。因此，大多数机器学习的模型中的  $w$  和  $b$  实际上是以变量的形式出现在代码中的，这就要求我们在代码中定义模型变量。

```
import tensorflow as tf
a = tf.Variable(2.)
b = tf.Variable(3.)
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer()) #变量初始化
    print(sess.run(a*b))
[out]:6.
```

[1] Platt, John. “Sequential minimal optimization: A fast algorithm for training support vector machines.” (1998).

initializer. 由于 tensorflow 构建的是静态的计算流图，在开启会话之前，所有的操作都不会被执行。因此为了执行在计算图中所构建的赋值初始化计算节点，需要在开启会话之后，在会话环境下运行初始化。如果计算图中定义了变量，而会话环境下未执行初始化命令，则程序报错，代码如下：

```
import tensorflow as tf
a = tf.Variable(2.)
b = tf.Variable(3.)
with tf.Session() as sess:
    #sess.run(tf.global_variables_initializer()) #注释掉初始化命令
    print(sess.run(a*b))
[Error]: Attempting to use uninitialized value Variable
```

**6.2. 启动图。** 先了解 session 的概念，然后才能更好的理解图的启动。 图的每个运行实例都必须在一个 session 里，session 为图的运行提供环境。Session 的类型是 `tf.Session`，在实例化 session 对象时我们需要给它传递一个图对象，如果不显示给出将使用默认的图。Session 有一个 `graph` 属性，我们可以通过它获取 session 对应的图。

代码如下：

```

numOfBatch=5
datas=np.zeros([numOfBatch,2],np.float32)
labels=np.zeros([numOfBatch,2],np.float32)

sess=tf.Session(graph=g)
graph=sess.graph
sess.run([graph.get_operation_by_name("initer")])

dataHolder=graph.get_tensor_by_name("input_data:0")
labelHolder=graph.get_tensor_by_name("input_label:0")
train=graph.get_operation_by_name("train_step")
out=graph.get_tensor_by_name("output:0")

for i in range(200):
    result=sess.run([out,train],feed_dict={dataHolder:datas,labelHolder:labels})
    if i%100==0:
        saver.save(sess,"./moules")

sess.close()

```

代码都比较简单，就不介绍了。不过要注意 2 点：1. 别忘记运行初始化节点，2. 别忘记 close 掉 session 对象以释放资源。

给图输入数据并获取结果. 代码:

```

for i in range(200):
    result=sess.run([out,train],feed_dict={dataHolder:datas,labelHolder:labels})

```

这里主要用到了 session 对象的 run 方法，它用来运行某个节点或 tensor 并获取对应的值。我们一般会一次传递一小部分数据进行 mini-batch 梯度下降来优化模型。

我们需要把我们需要运行的节点或 tensor 放入一个列表，然后作为第一个参数（不考虑 self）传递给 run 方法，run 方法会返回一个计算结果的列表，与我们传递的参数一一对应。

如果我们运行的节点依赖某个 placeholder，那我们必须给这个 placeholder 指定值，怎么指定代码里面很清楚，给关键字参数 feed\_dict 传递一个字典即可，字典里的元素的 key 是 placeholder 对象，value 是我们指定的值。值的数据的类型必须和 placeholder 一致，包括 shape。值本身的类型是 numpy 数组。

这里再解释一个细节，在定义 placeholder 时代码如下：

```
input_data=tf.placeholder(tf.float32,[None,2],"input_data")
input_label=tf.placeholder(tf.float32,[None,2],"input_label")
```

shape 为 [None,2]，说明数据第一个维度是不确定的，然后 TensorFlow 会根据我们传递的数据动态推断第一个维度，这样我们就可以在运行时改变 batch 的大小。比如一个数据是 2 维，一次传递 10 个数据对应的 tensor 的 shape 就是 [10,2]。可不可以把多个维度指定为 None？理论上不可以！

## 7. 如何基于 tensorflow 搭建 VGG16

介绍完关于 tensorflow 的基础知识，是时候来一波网络搭建实战了。虽然网上有很多相关教程，但我想从最标准的 tensorflow 代码和语法出发（而不是调用更高级的 API，失去了原来的味道），向大家展示如何搭建其标准的 VGG16 网络架构。话不多说，上代码：

```
[] import numpy as np import tensorflow as tf
def get_weight_variable(shape): return tf.get_variable('weight', shape=shape, initializer=tf.truncated_normal_initializer(stddev=0.1))
def get_bias_variable(shape): return tf.get_variable('bias', shape=shape, initializer=tf.constant_initializer(0))
def conv2d(x, w, padding = 'SAME', s=1): x = tf.nn.conv2d(x, w, strides=[1, s, s, 1], padding = padding) return x
def maxPoolLayer(x): return tf.nn.max_pool(x, ksize = [1, 2, 2, 1], strides = [1, 2, 2, 1], padding = 'SAME')
def conv2d_layer(x,in_chs, out_chs, ksize, layer_name): with tf.variable_scope(layer_name):
w = get_weight_variable([ksize, ksize, in_chs, out_chs]) b = get_bias_variable([out_chs])
y = tf.nn.relu(tf.bias_add(conv2d(x,w,padding = 'SAME', s=1), b)) return y
def fc_layer(x,in_kernels, out_kernels, layer_name): with tf.variable_scope(layer_name):
w = get_weight_variable([in_kernels,out_kernels]) b = get_bias_variable([out_kernels]) y = tf.nn.relu(tf.bias_add(tf.matmul(x,w),b)) return y
def VGG16(x): conv1_1 = conv2d_layer(x,tf.get_shape(x).as_list()[-1], 64, 3, 'conv1_1')
conv1_2 = conv2d_layer(conv1_1,64, 64, 3, 'conv1_2') pool1 = maxPoolLayer(conv1_2)
conv2_1 = conv2d_layer(pool1,64, 128, 3, 'conv2_1') conv2_2 = conv2d_layer(conv2_1,128, 128, 3, 'conv2_2') pool2 = maxPoolLayer(conv2_2)
conv3_1 = conv2d_layer(pool2,128, 256, 3, 'conv3_1') conv3_2 = conv2d_layer(conv3_1,256, 256, 3, 'conv3_2') conv3_3 = conv2d_layer(conv3_2,256, 256, 3, 'conv3_3') pool3 = maxPoolLayer(conv3_3)
```

```

conv4_1 = conv2d_layer(pool3,256, 512, 3, 'conv4_1') conv4_2 = conv2d_layer(conv4_1,512,
512, 3, 'conv4_2') conv4_3 = conv2d_layer(conv4_2,512, 512, 3, 'conv4_3') pool4 = max-
PoolLayer(conv4_3)

conv5_1 = conv2d_layer(pool4,512, 512, 3, 'conv5_1') conv5_2 = conv2d_layer(conv5_1,512,
512, 3, 'conv5_2') conv5_3 = conv2d_layer(conv5_1,512, 512, 3, 'conv5_3') pool5 = max-
PoolLayer(conv5_3)

pool5_flatten_dims = int(np.prod(pool5.get_shape().as_list()[1:])) pool5_flatten =
tf.reshape(pool5,[-1,pool5_flatten_dims])

fc_6 = fc_layer(pool5_flatten, pool5_flatten_dims, 4096, 'fc6') fc_7 = fc_layer(fc_6,
4096, 4096, 'fc7') fc_8 = fc_layer(fc_7, 4096, 10, 'fc8')

return fc_8

```

## 8. Pytorch

### 9. Pytorch 是什么？

Pytorch 是 torch 的 python 版本，是由 Facebook 开源的神经网络框架，专门针对 GPU 加速的深度神经网络（DNN）编程。Torch 是一个经典的对多维矩阵数据进行操作的张量（tensor）库，在机器学习和其他数学密集型应用有广泛应用。与 Tensorflow 的静态计算图不同，pytorch 的计算图是动态的，可以根据计算需要实时改变计算图。但由于 Torch 语言采用 Lua，导致在国内一直很小众，并逐渐被支持 Python 的 Tensorflow 抢走用户。作为经典机器学习库 Torch 的端口，PyTorch 为 Python 语言使用者提供了舒适的写代码选择。

### 10. 为什么选择 Pytorch ?

**10.1. 简洁：** PyTorch 的设计追求最少的封装，尽量避免重复造轮子。不像 TensorFlow 中充斥着 session、graph、operation、name\_scope、variable、tensor、layer 等全新的概念，PyTorch 的设计遵循 tensor→variable(autograd)→nn.Module 三个由低到高的抽象层次，分别代表高维数组（张量）、自动求导（变量）和神经网络（层/模块），而且这三个抽象之间联系紧密，可以同时进行修改和操作。简洁的设计带来的另外一个好处就是代码易于理解。PyTorch 的源码只有 TensorFlow 的十分之一左右，更少的抽象、更直观的设计使得 PyTorch 的源码十分易于阅读。

**10.2. 速度：** PyTorch 的灵活性不以速度为代价，在许多评测中，PyTorch 的速度表现胜过 TensorFlow 和 Keras 等框架。框架的运行速度和程序员的编码水平有极大关系，但同样的算法，使用 PyTorch 实现的那个更有可能快过用其他框架实现的。

**10.3. 易用:** PyTorch 是所有的框架中面向对象设计的最优雅的一个。PyTorch 的面向对象的接口设计来源于 Torch，而 Torch 的接口设计以灵活易用而著称，Keras 作者最初就是受 Torch 的启发才开发了 Keras。PyTorch 继承了 Torch 的衣钵，尤其是 API 的设计和模块的接口都与 Torch 高度一致。PyTorch 的设计最符合人们的思维，它让用户尽可能地专注于实现自己的想法，即所思即所得，不需要考虑太多关于框架本身的束缚。

**10.4. 活跃的社区:** PyTorch 提供了完整的文档，循序渐进的指南，作者亲自维护的论坛供用户交流和求教问题。Facebook 人工智能研究院对 PyTorch 提供了强力支持，作为当今排名前三的深度学习研究机构，FAIR 的支持足以确保 PyTorch 获得持续的开发更新，不至于像许多由个人开发的框架那样昙花一现。

## 11. PyTorch 的架构是怎样的？

PyTorch(Caffe2) 通过混合前端，分布式训练以及工具和库生态系统实现快速，灵活的实验和高效生产。PyTorch 和 TensorFlow 具有不同计算图实现形式，TensorFlow 采用静态图机制(预定义后再使用)，PyTorch 采用动态图机制(运行时动态定义)。PyTorch 具有以下高级特征：

混合前端：新的混合前端在急切模式下提供易用性和灵活性，同时无缝转换到图形模式，以便在 C++ 运行时环境中实现速度，优化和功能。  
分布式训练：通过利用本地支持集合操作的异步执行和可从 Python 和 C++ 访问的对等通信，优化了性能。  
Python 优先：PyTorch 为了深入集成到 Python 中而构建的，因此它可以与流行的库和 Cython 和 Numba 等软件包一起使用。  
丰富的工具和库：活跃的研究人员和开发人员社区建立了丰富的工具和库生态系统，用于扩展 PyTorch 并支持从计算机视觉到强化学习等领域的开发。

本机 ONNX 支持：以标准 ONNX（开放式神经网络交换）格式导出模型，以便直接访问与 ONNX 兼容的平台，运行时，可视化工具等。  
C++ 前端：C++ 前端是 PyTorch 的纯 C++ 接口，它遵循已建立的 Python 前端的设计和体系结构。它旨在实现高性能，低延迟和裸机 C++ 应用程序的研究。使用 GPU 和 CPU 优化的深度学习张量库。

## 12. Pytorch 与 tensorflow 之间的差异在哪里？

上面也将了 PyTorch 最大优势是建立的神经网络是动态的，对比静态的 Tensorflow，它能更有效地处理一些问题，比如说 RNN 变化时间长度的输出。各有各的优势和劣势。两者都是大公司发布的，Tensorflow (Google) 宣称在分布式训练上下了很大的功夫，那就默认 Tensorflow 在分布式训练上要超出 Pytorch (Facebook)，还有 tensorboard 可视化工具，但是 Tensorflow 的静态计算图使得在 RNN 上有一点点被动(虽然它用其他途径解决了)，不过用 PyTorch 的时候，会对这种动态的 RNN 有更好的理解。而且 Tensorflow 的高度工业化，它的

底层代码很难看懂，Pytorch 好那么一点点，如果深入 PytorchAPI，至少能比看 Tensorflow 多看懂一点点 Pytorch 的底层在干啥。

### 13. Pytorch 有哪些常用工具包？

torch : 类似 NumPy 的张量库，强 GPU 支持； torch.autograd : 基于 tape 的自动区别库，支持 torch 之中的所有可区分张量运行； torch.nn : 为最大化灵活性未涉及、与 autograd 深度整合的神经网络库； torch.optim: 与 torch.nn 一起使用的优化包，包含 SGD、RMSProp、LBFGS、Adam 等标准优化方式； torch.multiprocessing: python 多进程并发，进程之间 torch Tensors 的内存共享； torch.utils: 数据载入器。具有训练器和其他便利功能； torch.legacy(.nn/.optim) : 处于向后兼容性考虑，从 Torch 移植来的 legacy 代码；

### 14. Caffe

#### 15. 什么是 Caffe ?

Caffe 的全称应该是 Convolutional Architecture for Fast Feature Embedding，它是一个清晰、高效的深度学习框架，它是开源的，核心语言是 C++，它支持命令行、Python 和 Matlab 接口，它既可以在 CPU 上运行也可以在 GPU 上运行。它的 license 是 BSD 2-Clause。

#### 16. Caffe 的特点是什么？

(1)、模块化：Caffe 从一开始就设计得尽可能模块化，允许对新数据格式、网络层和损失函数进行扩展。

(2)、表示和实现分离：Caffe 的模型 (model) 定义是用 Protocol Buffer 语言写进配置文件的。以任意有向无环图的形式，Caffe 支持网络架构。Caffe 会根据网络的需要来正确占用内存。通过一个函数调用，实现 CPU 和 GPU 之间的切换。

(3)、测试覆盖：在 Caffe 中，每一个单一的模块都对应一个测试。

(4)、python 和 Matlab 接口：同时提供 Python 和 Matlab 接口。

(5)、预训练参考模型：针对视觉项目，Caffe 提供了一些参考模型，这些模型仅应用在学术和非商业领域，它们的 license 不是 BSD。

#### 17. Caffe 的设计思想是怎样的？

基本上，Caffe 沿用了神经网络的一个简单假设—所有的计算都是以 layer 的形式表示的，layer 做的事情就是 take 一些数据，然后输出一些计算以后的结果，比如说卷积，就是输入一个图像，然后和这一层的参数 (filter) 做卷积，然后输出卷积的结果。每一个 layer 需要做两个计算：forward 是从输入计算输出，然后 backward 是从上面给的 gradient 来计算相

对于输入的 gradient，只要这两个函数实现了以后，我们就可以把很多层连接成一个网络，这个网络做的事情就是输入我们的数据（图像或者语音或者 whatever），然后来计算我们需要的输出（比如说识别的 label），在 training 的时候，我们可以根据已有的 label 来计算 loss 和 gradient，然后用 gradient 来 update 网络的参数，这个就是 Caffe 的一个基本流程。

基本上，最简单地用 Caffe 上手的方法就是先把数据写成 Caffe 的格式，然后设计一个网络，然后用 Caffe 提供的 solver 来做优化看效果如何，如果你的数据是图像的话，可以从现有的网络，比如说 alexnet 或者 googlenet 开始，然后做 fine tuning，如果你的数据稍有不同，比如说是直接的 float vector，你可能需要做一些 custom 的 configuration，Caffe 的 logistic regression example 兴许会很有帮助。

Fine tune 方法：fine tuning 的想法就是说，在 imagenet 那么大的数据集上 train 好一个很牛的网络了，那别的 task 上肯定也不错，所以我们可以把 pretrain 的网络拿过来，然后只重新 train 最后几层，重新 train 的意思是说，比如我以前需要 classify imagenet 的一千类，现在我只想识别是狗还是猫，或者是不是车牌，于是我就可以把最后一层 softmax 从一个 40961000 的分类器变成一个 40962 的分类器，这个 strategy 在应用中非常好使，所以我们经常会先在 imagenet 上 pretrain 一个网络，因为我们知道 imagenet 上 training 的大概过程会怎么样。

## 18. Caffe 架构是怎样的？

Caffe 的架构与其它的深度学习框架稍微不同，它没有根据算法实现过程的方式来进行编码，而是以系统级的抽象作为整体架构，逐层的封装实现细节，使得上层的架构变得很清晰。Caffe 的整体架构如下：

**18.1. SyncedMem.** 这个类的主要功能是封装 CPU 和 GPU 的数据交互操作。一般来说，数据的流动形式都是：硬盘->CPU 内存->GPU 内存->CPU 内存->（硬盘），所以在写代码的过程中经常会写 CPU/GPU 之间数据传输的代码，同时还要维护 CPU 和 GPU 两个处理端的内存指针。这些事情处理起来不会很难，但是会很繁琐。因此 SyncedMem 的出现就是把 CPU/GPU 的数据传输操作封装起来，只需要调用简单的接口就可以获得两个处理端同步后的数据。

**18.2. Blob.** Blob 是用于存储数据的对象，在 Caffe 中各种数据（图像输入、模型参数）都是以 Blob 的形式在网络中传输的，Blob 提供统一的存储操作接口，可用来保存训练数据、模型参数等，同时 Blob 还能在 CPU 和 GPU 之间进行同步以支持 CPU/GPU 的混合运算。这个类做了两个封装：一个是操作数据的封装，使用 Blob 可以操纵高维的数据，快速访问其中的数据，变换数据的维度等；另一个是对原始数据和更新量的封装，每一个 Blob 中都有 data 和 diff 两个数据指针，data 用于存储原始数据，diff 用于存储反向传播

(Backpropagation) 的梯度更新值。Blob 使用了 SyncedMem，这样便于访问不同的处理端。Blob 基本实现了整个 Caffe 数据结构部分的封装，在 Net 类中可以看到所有的前后向数据和参数都用 Blob 来表示就足够了。数据的抽象到这个就可以了，接下来作层级的抽象。神经网络的前后向计算可以做到层与层之间完全独立，只要每个层按照一定的接口规则实现，就可以确保整个网络的正确性。

**18.3. Layer.** Layer 是网络 Net 的基本单元，也是 Caffe 中能在外部进行调整的最小网络结构单元，每个 Layer 都有输入 Blob 和输出 Blob。Layer (层) 是 Caffe 中最庞大最繁杂的模块，它是神经网络的基本计算单元。由于 Caffe 强调模块化设计，因此只允许每个 layer 完成一类特定的计算，例如 convolution 操作、pooling、非线性变换、内积运算，以及数据加载、归一化和损失计算等。Caffe 中 layer 的种类有很多，具体的种类及功能请看官方文档。在创建一个 Caffe 模型的时候，也是以 Layer 为基础进行的。Layer 是一个父类，它的下面还有各种实现特定功能的子类，例如 data\_layer，conv\_layer，loss\_layer 等。Layer 是通过 LayFactory 来创建的。

**18.4. Net.** Net 是一个完整的深度网络，包含输入层、隐藏层、输出层，在 Caffe 中一般是一个卷积神经网络 (Convolution Neural Network, CNN)。通过定义不同类型的 Layer，并用 Blob 将不同的 Layer 连接起来，就能产生一个 Net。Net 将数据 Blob 和层 Layer 组合起来做进一步的封装，对外提供了初始化和前后传播的接口，使得整体看上去和一个层的功能类似，但内部的组合可以是多种多样的。值得一提的是，每一层的输入输出数据统一保存在 Net 中，同时每个层内的参数指针也保存在 Net 中，不同的层可以通过 WeightShare 共享相同的参数，因此可以通过配置来实现多个神经网络层之间共享参数的功能。一个 Net 由多个 Layer 组成。一个典型的网络从 data layer (从磁盘中载入数据) 出发到 loss layer 结束。

**18.5. Solver.** 有了 Net 就可以进行神经网络的前后向传播计算了，但是还缺少神经网络的训练和预测功能，Solver 类进一步封装了训练和预测相关的一些功能。它还提供了两个接口：一个是更新参数的接口，继承 Solver 可以实现不同的参数更新方法，如 Momentum, Nesterov, Adagrad 等，因此可以使用不同的优化算法。另一个接口是训练过程中每一轮特定状态下的可注入的一些回调函数，在代码中这个回调点的直接使用者就是多 GPU 训练算法。Solver 定义了针对 Net 网络模型的求解方法，记录网络的训练过程，保存网络模型参数，中断并恢复网络的训练过程。自定义 Solver 能够实现不同的神经网络求解方式。阅读 Solver 的代码可以了解网络的求解优化过程。Solver 是一个父类，它下面还有实现不同优化方法的子类，例如 sgd\_solver, adagrad\_sovler 等，Solver 是通过 SolverFactory 来创建的。

**18.6. Proto.** caffe.proto 位于 .../src/caffe/proto 目录下，在这个文件夹下还有一个.pb.cc 和一个.pb.h 文件，这两个文件都是由 caffe.proto 编译而来的。在 caffe.proto 中定义了很多结构

化数据，包括：BlobProto、Datum、FillerParameter、NetParameter、SolverParameter、SolverState、LayerParameter、ConcatParameter、ConvolutionParameter、DataParameter、DropoutParameter、HDF5DataParameter、HDF5OutputParameter、ImageDataParameter、InfogainLossParameter、InnerProductParameter、LRNParameter、MemoryDataParameter、PoolingParameter、PowerParameter、WindowDataParameter、V0LayerParameter。

**18.7. IO.** 除了上面的东西之外，还需要输入数据和参数。DataReader 和 DataTransformer 帮助准备输入数据，Filler 对参数进行初始化，一些 Snapshot 方法可以对模型进行持久化。

## 19. Caffe 的有哪些接口？

Caffe 深度学习框架支持多种编程接口，包括命令行、Python 和 Matlab，下面将介绍如何使用这些接口。

**19.1. Caffe Python 接口.** Caffe 提供 Python 接口，即 Pycaffe，具体实现在 caffe、python 文件夹内。在 Python 代码中 import caffe，可以 load models (导入模型)、forward and backward (前向、反向迭代)、handle IO (数据输入输出)、visualize networks (绘制 net) 和 instrument model solving (自定义优化方法)。所有的模型数据、计算参数都是暴露在外、可供读写的。  
(1)caffe.Net 是主要接口，负责导入数据、校验数据、计算模型。  
(2)caffe.Classssifier 用于图像分类。  
(3)caffe.Detector 用于图像检测。  
(4)caffe.SGDSolver 是露在外的 solver 的接口。  
(5)caffe.io 处理输入输出，数据预处理。  
(6)caffe.draw 可视化 net 的结构。  
(7)caffe blobs 以 numpy ndarrays 的形式表示，方便而且高效。

**19.2. Caffe MATLAB 接口.** MATLAB 接口 (Matcaffe) 在 caffe/matlab 目录的 caffe 软件包。在 matcaffe 的基础上，可将 Caffe 整合到 MATLAB 代码中。MATLAB 接口包括：  
(1)MATLAB 中创建多个网络结构。  
(2) 网络的前向传播 (Forward) 与反向传播 (Backward) 计算。  
(3) 网络中的任意一层以及参数的存取。  
(4) 网络参数保存至文件或从文件夹加载。  
(5)blob 和 network 形状调整。  
(6) 网络参数编辑和调整。  
(7) 创建多个 solvers 进行训练。  
(8) 从 solver 快照 (Snapshots) 恢复并继续训练。  
(9) 访问训练网络 (Train nets) 和测试网络 (Test nets)。  
(10) 迭代后网络交由 MATLAB 控制。  
(11)MATLAB 代码融合梯度算法。

**19.3. Caffe 命令行接口.** 命令行接口 Cmdcaffe 是 Caffe 中用来训练模型、计算得分以及方法判断的工具。Cmdcaffe 存放在 caffe/build/tools 目录下。

**caffe train.**      caffe train 命令用于模型学习, 具体包括: (1)caffe train 带 solver.prototxt 参数完成配置。 (2)caffe train 带 snapshot mode\_iter\_1000.solverstate 参数加载 solver snapshot。 (3)caffe train 带 weights 参数 model.caffemodel 完成 Fine-tuning 模型初始化。

**caffe test.**      caffe test 命令用于测试运行模型的得分, 并且用百分比表示网络输出的最终结果, 比如 accuracy/loss 作为其结果。测试过程中, 显示每个 batch 的得分, 最后输出全部 batch 的平均得分值。

**caffe time.**      caffe time 命令用来检测系统性能和测量模型相对执行时间, 此命令通过逐层计时与同步, 执行模型检测。

参考文献: 1. 深度学习: Caffe 之经典模型讲解与实战 / 乐毅, 王斌

#### 19.4. 网络搭建有什么原则?

**19.5. 新手原则.** 刚入门的新手不建议直接上来就开始搭建网络模型。比较建议的学习顺序如下: - 1. 了解神经网络工作原理, 熟悉基本概念及术语。- 2. 阅读经典网络模型论文 + 实现源码 (深度学习框架视自己情况而定)。- 3. 找数据集动手跑一个网络, 可以尝试更改已有的网络模型结构。- 4. 根据自己的项目需要设计网络。

**19.6. 深度优先原则。** 通常增加网络深度可以提高准确率, 但同时会牺牲一些速度和内存。但深度不是盲目堆起来的, 一定要在浅层网络有一定效果的基础上, 增加深度。深度增加是为了增加模型的准确率, 如果浅层都学不到东西, 深了也没效果。

**19.7. 卷积核 size 一般为奇数。** 卷积核为奇数有以下好处: - 1 保证锚点刚好在中间, 方便以 central pixel 为标准进行滑动卷积, 避免了位置信息发生偏移。- 2 保证在填充 (Padding) 时, 在图像之间添加额外的零层, 图像的两边仍然对称。

**19.8. 卷积核不是越大越好。** AlexNet 中用到了一些非常大的卷积核, 比如  $11 \times 11$ 、 $5 \times 5$  卷积核, 之前人们的观念是, 卷积核越大, 感受野越大, 看到的图片信息越多, 因此获得的特征越好。但是大的卷积核会导致计算量的暴增, 不利于模型深度的增加, 计算性能也会降低。于是在 VGG、Inception 网络中, 利用 2 个  $3 \times 3$  卷积核的组合比 1 个  $5 \times 5$  卷积核的效果更佳, 同时参数量 ( $3 \times 3 \times 2 + 1 = 19 < 26 = 5 \times 5 \times 1 + 1$ ) 被降低, 因此后来  $3 \times 3$  卷积核被广泛应用在各种模型中。

### 20. 有哪些经典的网络模型值得我们去学习的?

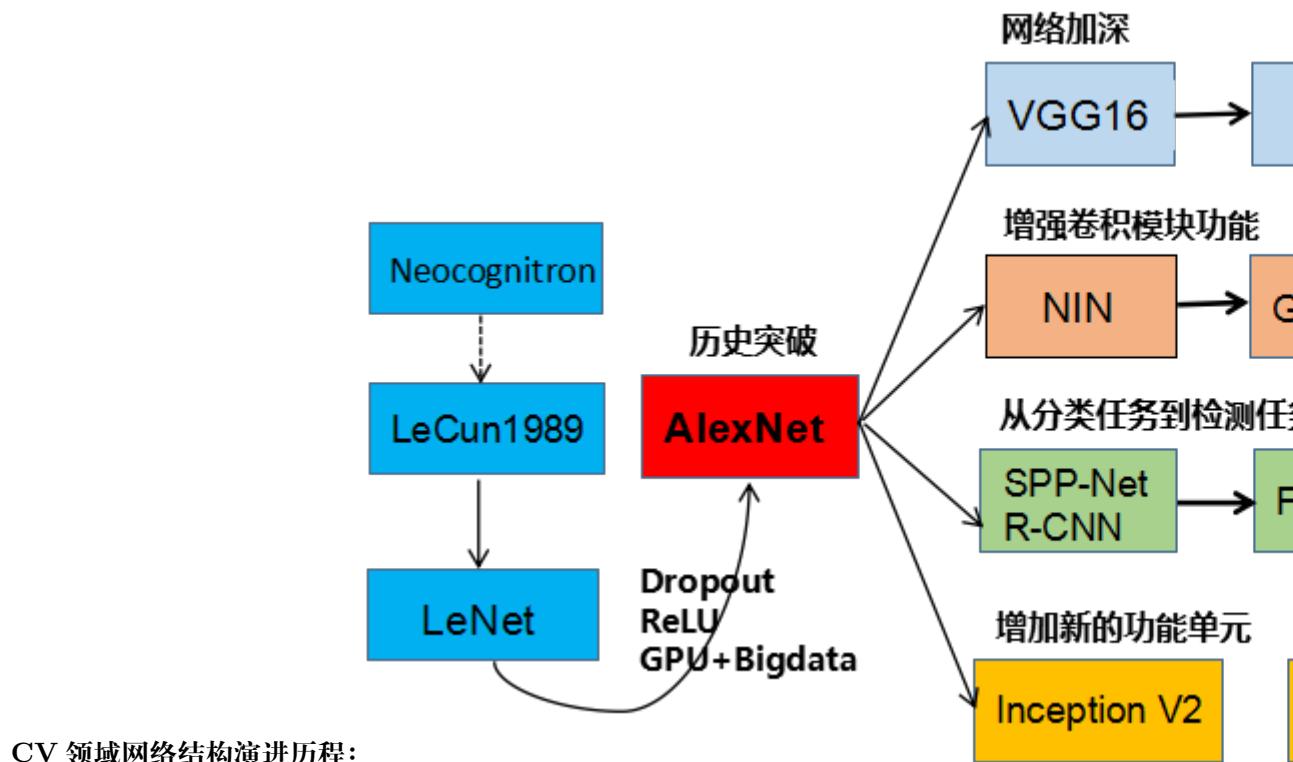
提起经典的网络模型就不得不提起计算机视觉领域的经典比赛: ILSVRC . 其全称是 ImageNet Large Scale Visual Recognition Challenge. 正是因为 ILSVRC 2012 挑战赛上的 AlexNet 横空出世, 使得全球范围内掀起了一波深度学习热潮。这一年也被称作“深度学习元年”。而在

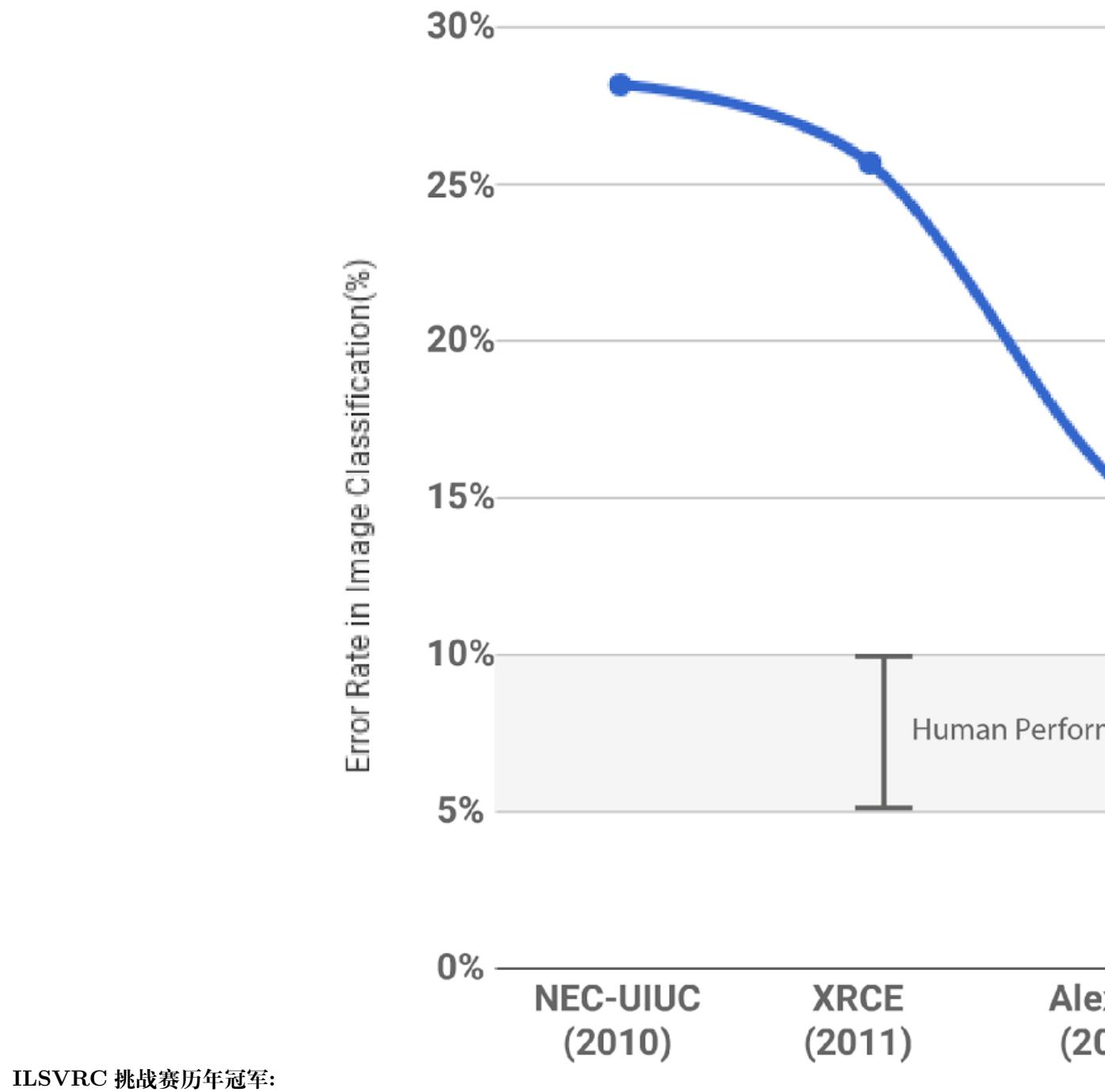
序号	年份	网络名称	获得荣誉
----	----	------	------

历年 ILSVRC 比赛中每次刷新比赛记录的那些神经网络也成为了人们心中的经典，成为学术界与工业届竞相学习与复现的对象，并在此基础上展开新的研究。

序号	年份	网络名称	获得荣誉
1	2012	AlexNet	ILSVRC 图像分类冠军
2	2014	VGGNet	ILSVRC 图像分类亚军
3	2014	GoogLeNet	ILSVRC 图像分类冠军
4	2015	ResNet	ILSVRC 图像分类冠军
5	2017	SeNet	ILSVRC 图像分类冠军

- 1 AlexNet 论文:[ImageNet Classification with Deep Convolutional Neural Networks](#) 代码实现:[tensorflow](#) 主要特点：> - 1. 第一次使用非线性激活函数 ReLU。> - 2. 增加防加过拟合方法：Dropout 层，提升了模型鲁棒性。> - 3. 首次使用数据增强。  
> - 4. 首次使用 GPU 加速运算。
- 2 VGGNet 论文:[Very Deep Convolutional Networks for Large-Scale Image Recognition](#) 代码实现:[tensorflow](#) 主要特点：> - 1. 网络结构更深。> - 2. 普遍使用小卷积核。
- 3 GoogLeNet 论文:[Going Deeper with Convolutions](#) 代码实现:[tensorflow](#) 主要特点：> - 1. 增强卷积模块功能。> 主要的创新在于他的 Inception，这是一种网中网（Network In Network）的结构，即原来的结点也是一个网络。Inception 一直在不断发展，目前已经 V2、V3、V4。其中 1\*1 卷积主要用来降维，用了 Inception 之后整个网络结构的宽度和深度都可扩大，能够带来 2-3 倍的性能提升。> - 2. 连续小卷积代替大卷积，保证感受野不变的同时，减少了参数数目。
- 4 ResNet 论文:[Deep Residual Learning for Image Recognition](#) 代码实现:[tensorflow](#) 主要特点：> 解决了“退化”问题，即当模型的层次加深时，错误率却提高了。
- 5 SeNet 论文:[Squeeze-and-Excitation Networks](#) 代码实现:[tensorflow](#) 主要特点：> 提出了 feature recalibration，通过引入 attention 重新加权，可以得到抑制无效特征，提升有效特征的权重，并很容易地和现有网络结合，提升现有网络性能，而计算量不会增加太多。





此后,ILSVRC 挑战赛的名次一直是衡量一个研究机构或企业技术水平的重要标尺。ILSVRC 2017 已是最后一届举办.2018 年起, 将由 WebVision 竞赛 (Challenge on Visual Understanding by Learning from Web Data) 来接棒。因此, 即使 ILSVRC 挑战赛停办了, 但其对深度学习的深远影响和巨大贡献, 将永载史册。

## 21. 10.6 网络训练有哪些技巧吗 ?

### 21.1. 10.6.1. 合适的数据集。

- 1 没有明显脏数据 (可以极大避免 Loss 输出为 NaN)。
- 2 样本数据分布均匀。

**21.2. 合适的预处理方法.** 关于数据预处理, 在 Batch Normalization 未出现之前预处理的主要做法是减去均值, 然后除去方差。在 Batch Normalization 出现之后, 减均值除方差的做法已经没有必要了。对应的预处理方法主要是数据筛查、数据增强等。

**21.3. 网络的初始化.** 网络初始化最粗暴的做法是参数赋值为全 0, 这是绝对不可取的。因为如果所有的参数都是 0, 那么所有神经元的输出都将相同, 那在 back propagation 的时候同一层内所有神经元的行为也是相同的, 这可能会直接导致模型失效, 无法收敛。吴恩达视频中介绍的方法是将网络权重初始化均值为 0、方差为 1 符合的正态分布的随机数据。

**21.4. 小规模数据试练.** 在正式开始训练之前, 可以先用小规模数据进行试练。原因如下:  
- 1 可以验证自己的训练流程对否。- 2 可以观察收敛速度, 帮助调整学习速率。- 3 查看 GPU 显存占用情况, 最大化 batch\_size(前提是进行了 batch normalization, 只要显卡不爆, 尽量挑大的)。

### 21.5. 设置合理 LearningRate.

- 1 太大。Loss 爆炸、输出 NaN 等。
- 2 太小。收敛速度过慢, 训练时长大大大延长。
- 3 可变的学习速率。比如当输出准确率达到某个阈值后, 可以让 Learning Rate 减半继续训练。

**21.6. 损失函数.** 损失函数主要分为两大类: 分类损失和回归损失  
>1. 回归损失: >>>  
- 1 均方误差 (MSE 二次损失 L2 损失) > 它是我们的目标变量与预测值变量差值平方。  
>> - 2 平均绝对误差 (MAE L1 损失) > 它是我们的目标变量与预测值变量差值绝对值。  
> 关于 MSE 与 MAE 的比较。MSE 更容易解决问题, 但是 MAE 对于异常值更加鲁棒。更多关于 MAE 和 MSE 的性能, 可以参考[L1vs.L2 Loss Function](#)

2. 分类损失: > - 1 交叉熵损失函数。是目前神经网络中最常用的分类目标损失函数。> - 2 合页损失函数 > 合页损失函数广泛在支持向量机中使用, 有时也会在损失函数中使用。缺点: 合页损失函数是对错误越大的样本施以更严重的惩罚, 但是这样会导致损失函数对噪声敏感。







## 优化算法

### 1. 如何解决训练样本少的问题

目前大部分的深度学习模型仍然需要海量的数据支持。例如 ImageNet 数据就拥有 1400 多万的图片。而现实生产环境中，数据集通常较小，只有几万甚至几百个样本。这时候，如何在这种情况下应用深度学习呢？

(1) 利用预训练模型进行迁移微调（fine-tuning），预训练模型通常在特征上拥有很好的语义表达。此时，只需将模型在小数据集上进行微调就能取得不错的效果。这也是目前大部分小数据集常用的训练方式。视觉领域内，通常会 ImageNet 上训练完成的模型。自然语言处理领域，也有 BERT 模型等预训练模型可以使用。 (2) 单样本或者少样本学习（one-shot, few-shot learning），这种方式适用于样本类别远大于样本数量的情况等极端数据集。例如有 1000 个类别，每个类别只提供 1-5 个样本。少样本学习同样也需要借助预训练模型，但有别于微调的在于，微调通常仍然在学习不同类别的语义，而少样本学习通常需要学习样本之间的距离度量。例如孪生网络（Siamese Neural Networks）就是通过训练两个同种结构的网络来判别输入的两张图片是否属于同一类。上述两种是常用训练小样本数据集的方式。此外，也有些常用的手段，例如数据集增强、正则或者半监督学习等方式来解决小样本数据集的训练问题。

### 2. 深度学习是否能胜任所有数据集？

深度学习并不能胜任目前所有的数据环境，以下列举两种情况：

(1) 深度学习能取得目前的成果，很大一部分原因依赖于海量的数据集以及高性能密集计算硬件。因此，当数据集过小时，需要考虑与传统机器学习相比，是否在性能和硬件资源效率更具有优势。(2) 深度学习目前在视觉，自然语言处理等领域都有取得不错的成果。这些领域最大的特点就是具有局部相关性。例如图像中，人的耳朵位于两侧，鼻子位于两眼之间，文本中单词组成句子。这些都是具有局部相关性的，一旦被打乱则会破坏语义或者有不同的语义。所以当数据不具备这种相关性的时候，深度学习就很难取得效果。

### 3. 有没有可能找到比已知算法更好的算法？

在最优化理论发展中，有个没有免费午餐的定律，其主要含义在于，在不考虑具体背景和细节的情况下，任何算法和随机猜的效果期望是一样的。即，没有任何一种算法能优于其他一

切算法，甚至不比随机猜好。深度学习作为机器学习领域的一个分支同样符合这个定律。所以，虽然目前深度学习取得了非常不错的成果，但是我们同样不能盲目崇拜。

优化算法本质上是在寻找和探索更符合数据集和问题的算法，这里数据集是算法的驱动力，而需要通过数据集解决的问题就是算法的核心，任何算法脱离了数据都会没有实际价值，任何算法的假设都不能脱离实际问题。因此，实际应用中，面对不同的场景和不同的问题，可以从多个角度针对问题进行分析，寻找更优的算法。

#### 4. 什么是共线性，如何判断和解决共线性问题？

对于回归算法，无论是一般回归还是逻辑回归，在使用多个变量进行预测分析时，都可能存在多变量相关的情况，这就是多重共线性。共线性的存在，使得特征之间存在冗余，导致过拟合。

常用判断是否存在共线性的方法有：

- (1) 相关性分析。当相关性系数高于 0.8，表明存在多重共线性；但相关系数低，并不能表示不存在多重共线性；
  - (2) 方差膨胀因子 VIF。当 VIF 大于 5 或 10 时，代表模型存在严重的共线性问题；
  - (3) 条件系数检验。当条件数大于 100、1000 时，代表模型存在严重的共线性问题。
- 通常可通过 PCA 降维、逐步回归法和 LASSO 回归等方法消除共线性。

#### 5. 权值初始化方法有哪些？

在深度学习的模型中，从零开始训练时，权重的初始化有时候会对模型训练产生较大的影响。良好的初始化能让模型快速、有效的收敛，而糟糕的初始化会使得模型无法训练。

目前，大部分深度学习框架都提供了各类初始化方式，其中一般常用的会有如下几种：**1. 常数初始化 (constant)**

把权值或者偏置初始化为一个常数。例如设置为 0，偏置初始化为 0 较为常见，权重很少会初始化为 0。TensorFlow 中也有 zeros\_initializer、ones\_initializer 等特殊常数初始化函数。

#### 2. 高斯初始化 (gaussian)

给定一组均值和标准差，随机初始化的参数会满足给定均值和标准差的高斯分布。高斯初始化是很常用的初始化方式。特殊地，在 TensorFlow 中还有一种截断高斯分布初始化 (truncated\_normal\_initializer)，其主要为了将超过两个标准差的随机数重新随机，使得随机数更稳定。

#### 3. 均匀分布初始化 (uniform)

给定最大最小的上下限，参数会在该范围内以均匀分布方式进行初始化，常用上下限为 (0, 1)。

#### 4. xavier 初始化 (uniform)

在 batchnorm 还未出现之前，要训练较深的网络，防止梯度弥散，需要依赖非常好的初始化方式。xavier 就是一种比较优秀的初始化方式，也是目前最常用的初始化方式之一。其目的是为了使得模型各层的激活值和梯度在传播过程中的方差保持一致。本质上 xavier 还是属于均匀分布初始化，但与上述的均匀分布初始化有所不同，xavier 的上下限将在如下范围内进行均匀分布采样：

$$[-\sqrt{\frac{6}{n+m}}, \sqrt{\frac{6}{n+m}}]$$

其中，n 为所在层的输入维度，m 为所在层的输出维度。

### 6. kaiming 初始化 (msra 初始化)

kaiming 初始化，在 caffe 中也叫 msra 初始化。kaiming 初始化和 xavier 一样都是为了防止梯度弥散而使用的初始化方式。kaiming 初始化的出现是因为 xavier 存在一个不成立的假设。xavier 在推导中假设激活函数都是线性的，而在深度学习中常用的 ReLu 等都是非线性的激活函数。而 kaiming 初始化本质上是高斯分布初始化，与上述高斯分布初始化有所不同，其是个满足均值为 0，方差为  $2/n$  的高斯分布：

$$[0, \sqrt{\frac{2}{n}}]$$

其中，n 为所在层的输入维度。

除上述常见的初始化方式以外，不同深度学习框架下也会有不同的初始化方式，读者可自行查阅官方文档。

## 6. 如何防止梯度下降陷入局部最优解？

梯度下降法 (GD) 及其一些变种算法是目前深度学习里最常用于求解凸优化问题的优化算法。神经网络很可能存在很多局部最优解，而非全局最优解。为了防止陷入局部最优，通常会采用如下一些方法，当然，这并不能保证一定能找到全局最优解，或许能得到一个比目前更优的局部最优解也是不错的：

### (1) stochastic GD /Mini-Batch GD

在 GD 算法中，每次的梯度都是从所有样本中累计获取的，这种情况最容易导致梯度方向过于稳定一致，且更新次数过少，容易陷入局部最优。而 stochastic GD 是 GD 的另一种极端更新方式，其每次都只使用一个样本进行参数更新，这样更新次数大大增加也就不容易陷入局部最优。但引出的一个问题的在于其更新方向过多，导致不易于进一步优化。Mini-Batch GD 便是两种极端的折中，即每次更新使用一小批样本进行参数更新。Mini-Batch GD 是目前最常用的优化算法，严格意义上 Mini-Batch GD 也叫做 stochastic GD，所以很多深度学习框架上都叫做 SGD。(2) 动量 动量也是 GD 中常用的方式之一，SGD 的更新方式虽然有效，但每次只依赖于当前批样本的梯度方向，这样的梯度方向依然很可能很随机。动量就是用来减

少随机，增加稳定性。其思想是模仿物理学的动量方式，每次更新前加入部分上一次的梯度量，这样整个梯度方向就不容易过于随机。一些常见情况时，如上次梯度过大，导致进入局部最小点时，下一次更新能很容易借助上次的大梯度跳出局部最小点。

### (3) 自适应学习率

无论是 GD 还是动量重点优化角度是梯度方向。而学习率则是用来直接控制梯度更新幅度的超参数。自适应学习率的优化方法有很多，例如 Adagrad 和 RMSprop。两种自适应学习率的方式稍有差异，但主要思想都是基于历史的累计梯度去计算一个当前较优的学习率。

## 7. 为什么需要激活函数？

(1) 非线性：即导数不是常数。这个条件是多层神经网络的基础，保证多层网络不退化成单层线性网络。这也是激活函数的意义所在。

(2) 几乎处处可微：可微性保证了在优化中梯度的可计算性。传统的激活函数如 sigmoid 等满足处处可微。对于分段线性函数比如 ReLU，只满足几乎处处可微（即仅在有限个点处不可微）。对于 SGD 算法来说，由于几乎不可能收敛到梯度接近零的位置，有限的不可微点对于优化结果不会有很大影响 [1]。

(3) 计算简单：非线性函数有很多。极端的说，一个多层神经网络也可以作为一个非线性函数，类似于 Network In Network[2] 中把它当做卷积操作的做法。但激活函数在神经网络前向的计算次数与神经元的个数成正比，因此简单的非线性函数自然更适合用作激活函数。这也是 ReLU 之流比其它使用 Exp 等操作的激活函数更受欢迎的其中一个原因。

(4) 非饱和性 (saturation)：饱和指的是在某些区间梯度接近于零（即梯度消失），使得参数无法继续更新的问题。最经典的例子是 Sigmoid，它的导数在  $x$  为比较大的正值和比较小的负值时都会接近于 0。更极端的例子是阶跃函数，由于它在几乎所有位置的梯度都为 0，因此处处饱和，无法作为激活函数。ReLU 在  $x > 0$  时导数恒为 1，因此对于再大的正值也不会饱和。但同时对于  $x < 0$ ，其梯度恒为 0，这时候它也会出现饱和的现象（在这种情况下通常称为 dying ReLU）。Leaky ReLU[3] 和 PReLU[4] 的提出正是为了解决这一问题。

(5) 单调性 (monotonic)：即导数符号不变。这个性质大部分激活函数都有，除了诸如  $\sin$ 、 $\cos$  等。个人理解，单调性使得在激活函数处的梯度方向不会经常改变，从而让训练更容易收敛。

(6) 输出范围有限：有限的输出范围使得网络对于一些比较大的输入也会比较稳定，这也是为什么早期的激活函数都以此类函数为主，如 Sigmoid、Tanh。但这导致了前面提到的梯度消失问题，而且强行让每一层的输出限制到固定范围会限制其表达能力。因此现在这类函数仅用于某些需要特定输出范围的场合，比如概率输出（此时 loss 函数中的 log 操作能够抵消其梯度消失的影响 [1]）、LSTM 里的 gate 函数。

(7) 接近恒等变换 (identity): 即约等于  $x$ 。这样的好处是使得输出的幅值不会随着深度的增加而发生显著的增加，从而使网络更为稳定，同时梯度也能够更容易地回传。这个与非线性是有点矛盾的，因此激活函数基本只是部分满足这个条件，比如 TanH 只在原点附近有线性区（在原点为 0 且在原点的导数为 1），而 ReLU 只在  $x>0$  时为线性。这个性质也让初始化参数范围的推导更为简单 [5][4]。额外提一句，这种恒等变换的性质也被其他一些网络结构设计所借鉴，比如 CNN 中的 ResNet[6] 和 RNN 中的 LSTM。

(8) 参数少: 大部分激活函数都是没有参数的。像 PReLU 带单个参数会略微增加网络的大小。还有一个例外是 Maxout[7]，尽管本身没有参数，但在同样输出通道数下  $k$  路 Maxout 需要的输入通道数是其它函数的  $k$  倍，这意味着神经元数目也需要变为  $k$  倍；但如果不算维持输出通道数的情况下，该激活函数又能将参数个数减少为原来的  $k$  倍。

(9) 归一化 (normalization): 这个是最近才出来的概念，对应的激活函数是 SELU[8]，主要思想是使样本分布自动归一化到零均值、单位方差的分布，从而稳定训练。在这之前，这种归一化的思想也被用于网络结构的设计，比如 Batch Normalization[9]。

## 8. 常见的损失函数有哪些？

机器学习通过对算法中的目标函数进行不断求解优化，得到最终想要的结果。分类和回归问题中，通常使用损失函数或代价函数作为目标函数。

损失函数用来评价预测值和真实值不一样的程度。通常损失函数越好，模型的性能也越好。

损失函数可分为经验风险损失和结构风险损失。经验风险损失是根据已知数据得到的损失。结构风险损失是为了防止模型被过度拟合已知数据而加入的惩罚项。

下面介绍常用的损失函数: (1) 0-1 损失函数

如果预测值和目标值相等，值为 0，如果不相等，值为 1:

$$L(Y, f(x)) = \begin{cases} 1 & , Y \neq f(x), \\ 0 & , Y = f(x). \end{cases}$$

一般的在实际使用中，相等的条件过于严格，可适当放宽条件：

$$L(Y, f(x)) = \begin{cases} 1 & , |Y - f(x)| \geq T, \\ 0 & , |Y - f(x)| < T. \end{cases}$$

(2) 绝对值损失函数

和 0-1 损失函数相似，绝对值损失函数表示为：

$$L(Y, f(x)) = |Y - f(x)|.$$

### (3) 平方损失函数

$$L(Y|f(x)) = \sum_N (Y - f(x))^2.$$

这点可从最小二乘法和欧几里得距离角度理解。最小二乘法的原理是，最优拟合曲线应该使所有点到回归直线的距离和最小。

### (4) log 对数损失函数

$$L(Y, P(Y|X)) = -\log P(Y|X).$$

常见的逻辑回归使用的就是对数损失函数，有很多人认为逻辑回归的损失函数式平方损失，其实不然。逻辑回归它假设样本服从伯努利分布，进而求得满足该分布的似然函数，接着取对数求极值等。逻辑回归推导出的经验风险函数是最小化负的似然函数，从损失函数的角度看，就是 log 损失函数。

### (5) 指数损失函数

指数损失函数的标准形式为：

$$L(Y|f(x)) = \exp[-yf(x)].$$

例如 AdaBoost 就是以指数损失函数为损失函数。

### (6) Hinge 损失函数

Hinge 损失函数的标准形式如下：

$$L(y) = \max(0, 1 - ty).$$

其中 y 是预测值，范围为 (-1,1), t 为目标值，其为-1 或 1。

在线性支持向量机中，最优化问题可等价于：

$$\min_{w,b} \sum_{i=1}^N (1 - y_i(wx_i + b)) + \lambda \|w^2\|$$

$$\frac{1}{m} \sum_{i=1}^N l(wx_i + by_i)) + \|w^2\|$$

其中  $l(wx_i + by_i))$  是 Hinge 损失函数， $\|w^2\|$  可看做为正则化项。

## 9. 如何进行特征选择 (feature selection)?

**9.1. 特征类型有哪些？** 对象本身会有许多属性。所谓特征，即能在某方面最能表征对象的一个或者一组属性。一般地，我们可以把特征分为如下三个类型：

- (1) 相关特征: 对于特定的任务和场景具有一定帮助的属性, 这些属性通常能有效提升算法性能;
- (2) 无关特征: 在特定的任务和场景下完全无用的属性, 这些属性对对象在本目标环境下完全无用;
- (3) 冗余特征: 同样是在特定的任务和场景下具有一定帮助的属性, 但这类属性已过多的存在, 不具有产生任何新的信息的能力。

**9.2. 如何考虑特征选择.** 当完成数据预处理之后, 对特定的场景和目标而言很多维度上的特征都是不具有任何判别或者表征能力的, 所以需要对数据在维度上进行筛选。一般地, 可以从以下两个方面考虑来选择特征:

- (1) 特征是否具有发散性: 某个特征若在所有样本上的都是一样的或者接近一致, 即方差非常小。也就是说所有样本的都具有一致的表现, 那这些就不具有任何信息。
- (2) 特征与目标的相关性: 与目标相关性高的特征, 应当优先选择。

**9.3. 特征选择方法分类.** 根据特征选择的形式又可以将特征选择方法分为 3 种: (1) 过滤法: 按照发散性或者相关性对各个特征进行评分, 设定阈值或者待选择阈值的个数, 选择特征。

(2) 包装法: 根据目标函数(通常是预测效果评分), 每次选择若干特征, 或者排除若干特征。

(3) 嵌入法: 先使用某些机器学习的算法和模型进行训练, 得到各个特征的权值系数, 根据系数从大到小选择特征。

**9.4. 特征选择目的.** (1) 减少特征维度, 使模型泛化能力更强, 减少过拟合;

- (2) 降低任务目标的学习难度;
- (3) 一组优秀的特征通常能有效的降低模型复杂度, 提升模型效率

## 10. 梯度消失/梯度爆炸原因, 以及解决方法

**10.1. 为什么要使用梯度更新规则?** 目前深度学习的火热, 其最大的功臣之一就是反向传播。反向传播, 即根据损失评价函数计算的误差, 计算得到梯度, 通过梯度反向传播的方式, 指导深度网络权值的更新优化。这样做的原因在于, 深层网络由许多非线性层堆叠而来, 每一层非线性层都可以视为是一个非线性函数, 因此整个深度网络可以视为是一个复合的非线性多元函数:

$$F(x) = f_n(\cdots f_3(f_2(f_1(x) * \theta_1 + b) * \theta_2 + b) \cdots)$$

我们最终的目的是希望这个多元函数可以很好的完成输入到输出之间的映射, 假设不同的输入, 输出的最优解是  $g(x)$ , 那么, 优化深度网络就是为了寻找到合适的权值, 满足  $\text{Loss} = L(g(x), F(x))$

图 13.8.1

**10.2. 梯度消失/爆炸产生的原因?** 本质上，梯度消失和爆炸是一种情况。在深层网络中，由于网络过深，如果初始得到的梯度过小，或者传播途中在某一层上过小，则在之后的层上得到的梯度会越来越小，即产生了梯度消失。梯度爆炸也是同样的。一般地，不合理的初始化以及激活函数，如 sigmoid 等，都会导致梯度过大或者过小，从而引起消失/爆炸。

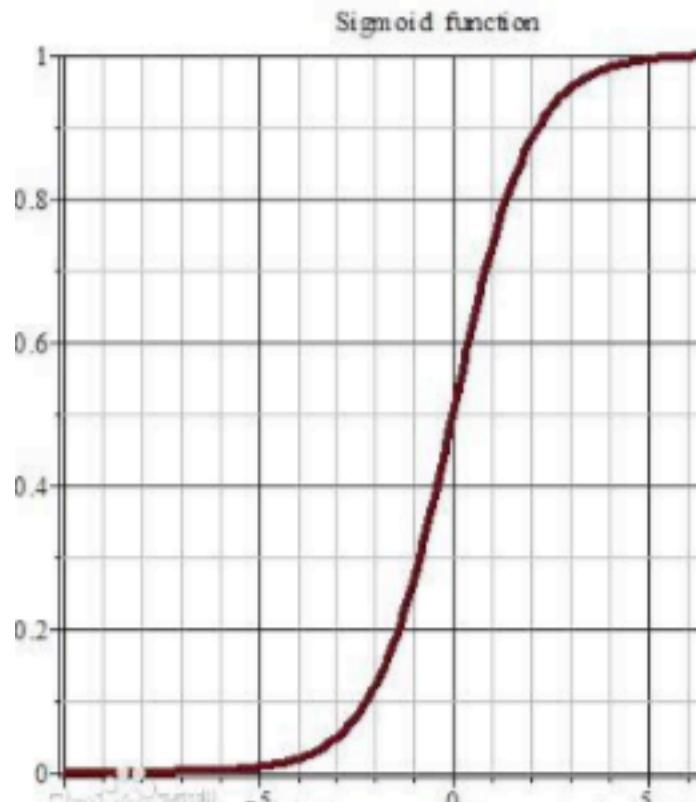
下面分别从网络深度角度以及激活函数角度进行解释：

### (1) 网络深度

若在网络很深时，若权重初始化较小，各层上的相乘得到的数值都会 0-1 之间的小数，而激活函数梯度也是 0-1 之间的数。那么连乘后，结果数值就会变得非常小，导致**梯度消失**。若权重初始化较大，大到乘以激活函数的导数都大于 1，那么连乘后，可能会导致求导的结果很大，形成**梯度爆炸**。

### (2) 激活函数

如果激活函数选择不合适，比如使用 sigmoid，梯度消失就会很明显了，原因看下图，左图是 sigmoid 的函数图，右边是其导数的图像，如果使用 sigmoid 作为损失函数，其梯度是不可能超过



0.25 的，这样经过链式求导之后，很容易发生梯度消失。

图 13.8.2 sigmoid 函数与其导数

### 10.3. 梯度消失、爆炸的解决方案. 1、预训练加微调

此方法来自 Hinton 在 2006 年发表的一篇论文, Hinton 为了解决梯度的问题, 提出采取无监督逐层训练方法, 其基本思想是每次训练一层隐节点, 训练时将上一层隐节点的输出作为输入, 而本层隐节点的输出作为下一层隐节点的输入, 此过程就是逐层“预训练”(pre-training); 在预训练完成后, 再对整个网络进行“微调”(fine-tunning)。Hinton 在训练深度信念网络 (Deep Belief Networks 中, 使用了这个方法, 在各层预训练完成后, 再利用 BP 算法对整个网络进行训练。此思想相当于是先寻找局部最优, 然后整合起来寻找全局最优, 此方法有一定的好处, 但是目前应用的不是很多了。

### 2、梯度剪切、正则

梯度剪切这个方案主要是针对梯度爆炸提出的, 其思想是设置一个梯度剪切阈值, 然后更新梯度的时候, 如果梯度超过这个阈值, 那么就将其强制限制在这个范围之内。这可以防止梯度爆炸。

另外一种解决梯度爆炸的手段是采用权重正则化 (weights regularization) 比较常见的是 L1 和 L2 正则。

### 3、ReLU、leakReLU 等激活函数

(1) ReLU: 其函数的导数在正数部分是恒等于 1, 这样在深层网络中, 在激活函数部分就不存在导致梯度过大或者过小的问题, 缓解了梯度消失或者爆炸。同时也方便计算。当然, 其也存在一些缺点, 例如过滤到了负数部分, 导致部分信息的丢失, 输出的数据分布不在以 0 为中心, 改变了数据分布。

(2) leakrelu: 就是为了解决 relu 的 0 区间带来的影响, 其数学表达为:  $\text{leakrelu} = \max(k*x, 0)$  其中 k 是 leak 系数, 一般选择 0.01 或者 0.02, 或者通过学习而来。

### 4、batchnorm

Batchnorm 是深度学习发展以来提出的最重要的成果之一了, 目前已经被广泛的应用到了各大网络中, 具有加速网络收敛速度, 提升训练稳定性效果, Batchnorm 本质上是解决反向传播过程中的梯度问题。Batchnorm 全名是 Batch Normalization, 简称 BN, 即批规范化, 通过规范化操作将输出信号 x 规范化到均值为 0, 方差为 1 保证网络的稳定性。

### 5、残差结构

残差的方式, 能使得深层的网络梯度通过跳级连接路径直接返回到浅层部分, 使得网络无论多深都能将梯度进行有效的回传。

### 6、LSTM

LSTM 全称是长短期记忆网络 (long-short term memory networks), 是不容易发生梯度

消失的，主要原因在于 LSTM 内部复杂的“门”(gates)。在计算时，将过程中的梯度进行了抵消。

## 11. 深度学习为什么不用二阶优化？

目前深度学习中，反向传播主要是依靠一阶梯度。二阶梯度在理论和实际上都是可以应用都网络中的，但相比于一阶梯度，二阶优化会存在以下一些主要问题：

- (1) 计算量大，训练非常慢。
- (2) 二阶方法能够更快地求得更高精度的解，这在浅层模型是有益的。而在神经网络这类深层模型中对参数的精度要求不高，甚至不高的精度对模型还有益处，能够提高模型的泛化能力。
- (3) 稳定性。二阶方法能更快求高精度的解，同样对数据本身要的精度也会相应的变高，这就会导致稳定性上的问题。

## 12. 为什么要设置单一数字评估指标，设置指标的意义？

在训练模型时，无论是调整超参数，还是调整不同的模型算法，我们都需要一个有效的评价指标，这个评价标准能帮助我们快速了解新的尝试后模型的性能是否更优。例如在分类时，我们通常会选择选择准确率，当样本不平衡时，查准率和查全率又会是更好的评价指标。所以在训练模型时，如果设置了单一数字的评估指标通常能很快的反应出我们模型的改进是否直接产生了收益，从而加速我们的算法改进过程。若在训练过程中，发现优化目标进一步深入，现有指标无法完全反应进一步的目标时，就需要重新选择评估指标了。

## 13. 训练/验证/测试集的定义及划分

训练、验证、测试集在机器学习领域是非常重要的三个内容。三者共同组成了整个项目的性能的上限和走向。

训练集：用于模型训练的样本集合，样本占用量是最大的；

验证集：用于训练过程中的模型性能评价，跟着性能评价才能更好的调参；

测试集：用于最终模型的一次最终评价，直接反应了模型的性能。

在划分上，可以分两种情况：

1、在样本量有限的情况下，有时候会把验证集和测试集合并。实际中，若划分为三类，那么训练集：验证集：测试集 =6:2:2；若是两类，则训练集：验证集 =7:3。这里需要主要在数据量不够多的情况下，验证集和测试集需要占的数据比例比较多，以充分了解模型的泛化性。

2、在海量样本的情况下，这种情况在目前深度学习中会比较常见。此时由于数据量巨大，我们不需要将过多的数据用于验证和测试集。例如拥有 1 百万样本时，我们按训练集：验证集：测试集 =98:1:1 的比例划分，1% 的验证和 1% 的测试集都已经拥有了 1 万个样本。这已足够验证模型性能了。

此外，三个数据集的划分不是一次就可以的，若调试过程中发现，三者得到的性能评价差异很大时，可以重新划分以确定是数据集划分的问题导致还是由模型本身导致的。其次，若评价指标发生变化，而导致模型性能差异在三者上很大时，同样可重新划分确认排除数据问题，以方便进一步的优化。

#### 14. 什么是 TOP5 错误率？

通常对于分类系统而言，系统会对某个未知样本进行所有已知样本的匹配，并给出该未知样本在每个已知类别上的概率。其中最大的概率就是系统判定最可能的一个类别。TOP5 则就是在前五个最大概率的类别。TOP5 错误率，即预测最可能的五类都不是该样本类别的错误率。

TOP5 错误率通常会用于在类别数量很多或者细粒度类别的模型系统。典型地，例如著名的 ImageNet，其包含了 1000 个类别。通常就会采用 TOP5 错误率。

#### 15. 什么是泛化误差，如何理解方差和偏差？

一般情况下，我们评价模型性能时都会使用泛化误差。泛化误差越低，模型性能越好。泛化误差可分解为方差、偏差和噪声三部分。这三部分中，噪声是个不可控因素，它的存在是算法一直无法解决的问题，很难约减，所以我们更多考虑的是方差和偏差。

方差和偏差在泛化误差上可做如下分解，假设我们的预测值为  $g(x)$ ，真实值为  $f(x)$ ，则均方误差为

$$E((g(x)f(x))^2)$$

这里假设不考虑噪声， $g$  来代表预测值， $f$  代表真实值， $g^- = E(g)$  代表算法的期望预测，则有如下表达：

$$\begin{aligned} E(g - f)^2 &= E(g^2 - 2gf + f^2) \\ &= E(g^2) - \bar{g}^2 + (\bar{g} - f)^2 \\ &= E(g^2) - 2\bar{g}^2 + \bar{g}^2 + (\bar{g} - f)^2 \\ &= E(g^2) - 2g\bar{g}^2 + \bar{g}^2 + (\bar{g} - f)^2 \\ &= \underbrace{E(g - \bar{g})^2}_{var(x)} + \underbrace{(\bar{g} - f)^2}_{bias^2(x)} \end{aligned}$$

有上述公式可知，方差描述是理论期望和预测值之间的关系，这里的理论期望通常是指所有适用于模型的各种不同分布类型的数据集；偏差描述为真实值和预测值之间的关系，这里的真实值通常指某一个特定分布的数据集合。

所以综上方差表现为模型在各类分布数据的适应能力，方差越大，说明数据分布越分散，而偏差则表现为在特定分布上的适应能力，偏差越大越偏离真实值。

## 16. 如何提升模型的稳定性？

评价模型不仅要从模型的主要指标上的性能，也要注重模型的稳定性。模型的稳定性体现在对不同样本之间的体现的差异。如模型的方差很大，那可以从如下几个方面进行考虑：

(1) 正则化 (L2, L1, dropout): 模型方差大，很可能来自于过拟合。正则化能有效的降低模型的复杂度，增加对更多分布的适应性。

(2) 提前停止训练：提前停止是指模型在验证集上取得不错的性能时停止训练。这种方式本质和正则化是一个道理，能减少方差的同时增加的偏差。目的为了平衡训练集和未知数据之间在模型的表现差异。

(3) 扩充训练集：正则化通过控制模型复杂度，来增加更多样本的适应性。那增加训练集让模型适应不同类型的数据本身就是一种最简单直接的方式提升模型稳定的方法，也是最可靠的一种方式。与正则有所不同的是，扩充数据集既可以减小偏差又能减小方差。

(4) 特征选择：过高的特征维度会使模型过拟合，减少特征维度和正则一样可能会处理好方差问题，但是同时会增大偏差。但需要注意的是若过度删减特征，很可能会删除很多有用的特征，降低模型的性能。所以需要多注意删减的特征对模型的性能的影响。

## 17. 有哪些改善模型的思路

改善模型本质是如何优化模型，这本身是个很宽泛的问题。也是目前学界一直探索的目的，而从目前常规的手段上来说，一般可取如下几点。

**17.1. 数据角度.** 增强数据集。无论是有监督还是无监督学习，数据永远是最重要的驱动力。更多的类型数据对良好的模型能带来更好的稳定性和对未知数据的可预见性。对模型来说，“看到过的总比没看到的更具有判断的信心”。但增大数据并不是盲目的，模型容限能力不高的情况下即使增大数据也对模型毫无意义。而从数据获取的成本角度，对现有数据进行有效的扩充也是个非常有效且实际的方式。良好的数据处理，常见的处理方式如数据缩放、归一化和标准化等。

**17.2. 模型角度.** 模型的容限能力决定着模型可优化的空间。在数据量充足的前提下，对同类型的模型，增大模型规模来提升容限无疑是最直接和有效的手段。但越大的参数模型优化也会越难，所以需要在合理的范围内对模型进行参数规模的修改。而不同类型的模型，在不同数据上的优化成本都可能不一样，所以在探索模型时需要尽可能挑选优化简单，训练效率更高的模型进行训练。

**17.3. 调参优化角度.** 如果你知道模型的性能为什么不再提高了，那已经向提升性能跨出了一大步。超参数调整本身是一个比较大的问题。一般可以包含模型初始化的配置，优化算法的选取、学习率的策略以及如何配置正则和损失函数等等。这里需要提出的是对于同一优化算

法，相近参数规模的前提下，不同类型的模型总能表现出不同的性能。这实际上就是模型优化成本。从这个角度的反方向来考虑，同一模型也总能找到一种比较适合的优化算法。所以确定了模型后选择一个适合模型的优化算法也是非常重要的手段。

**17.4. 训练角度.** 很多时候我们会把优化和训练放一起。但这里我们分开来讲，主要是为了强调充分的训练。在越大规模的数据集或者模型上，诚然一个好的优化算法总能加速收敛。但你在未探索到模型的上限之前，永远不知道训练多久算训练完成。所以在改善模型上充分训练永远是最必要的过程。充分训练的含义不仅仅只是增大训练轮数。有效的学习率衰减和正则同样是充分训练中非常必要的手段。

## 18. 如何快速构建有效初始模型？

构建一个有效的初始模型能帮助我们快速了解数据的质量和确定模型构建的方向。构建一个良好的初始模型，一般需要注意如下几点：

1、了解“对手”。这里的“对手”通常是指数据，我们在得到数据时，第一步是需要了解数据特点和使用场合。了解数据特点能帮助我们快速定位如何进行建模。确定使用场合能帮助我们进一步确定模型需要优化的方向。数据特点一般需要了解例如数据集规模、训练集和验证集是否匹配、样本的分布是否均匀、数据是否存在缺失值等等。

2、站在巨人肩膀上。根据数据特点，我们通常能匹配到一个现有比较优秀的模型。这类模型都通常能在类似数据上表现出一个比较不错的性能。

3、一切从简。初始模型的作用在于迅速了解数据质量和特点，所以模型的性能通常不需要达到很高，模型复杂度也不需要很高。例如，做图像分类时，我们在使用预训练模型时，不需要一开始就使用例如 ResNet152 这类模型巨大，复杂度过高的模型。这在数据量较小时，很容易造成过拟合而导致出现我们对数据产生一些误导性的判断，此外也增加了额外训练构建时间。所以使用更小更简单的模型以及损失函数来试探数据是相比更明智的选择。

4、总比瞎猜强。构建模型的意义在于建立一个高效的模型，虽然初始模型我们不对性能做过高的要求。但前提在于必须要比随机猜测好，不然构建模型的意义就不存在了。

5、解剖模型。一旦确定了一个初始模型时，无论你对该模型多熟悉，当其面对一批新数据时，你永远需要重新去认识这个模型，因为你永远不确定模型内部到底发生了些什么。解剖模型一般需要在训练时注意误差变化、注意训练和验证集的差异；出现一些 NAN 或者 INF 等情况时，需要打印观察内部输出，确定问题出现的时间和位置；在完成训练后，需要测试模型的输出是否正确合理，以确认评价指标是否符合该数据场景。无论使用任何一种模型，我们都不能把它当做黑盒去看待。

## 19. 如何通过模型重新观察数据？

对于这个问题，与其说如何做，倒不如说这个问题是用来强调这样做的重要性。如何重新观察数据其实不难，而是很多读者，会忽略这一项过程的重要性。

通过模型重新观察数据，不仅能够让我们了解模型情况，也能让我们对数据质量产生进一步的理解。目前深度学习在监督学习领域成就是非常显著的。监督学习需要依赖大量的人为标注，人为标注很难确定是否使用的数据中是否存在错误标注或者漏标注等问题。这无论是哪种情况都会影响我们对模型的判断。所以通过模型重新验证数据质量是非常重要的一步。很多初学者，通常会忽略这一点，而导致出现对模型的一些误判，严重时甚至会影响整个建模方向。此外，对于若出现一些过拟合的情况，我们也可以通过观察来了解模型。例如分类任务，样本严重不平衡时，模型全预测到了一边时，其正确率仍然很高，但显然模型已经出现了问题。

## 20. 如何解决数据不匹配问题？

**20.1. 如何定位数据不匹配？** 数据不匹配问题是个不容易定位和解决的问题。这个问题出现总会和模型过拟合表现很相似，即在训练集上能体现非常不错的性能，但在测试集上表现总是差强人意但区别在于如果遇到是数据不匹配的问题，通常在用一批和训练集有看相同或者相似分布的数据上仍然能取得不错的成绩。但很多时候，当测试集上结果表现很差时，很多初学者可能会直接将问题定位在模型过拟合上，最后对模型尝试各种方法后，性能却始终不能得到有效提升。当遇到这种情况时，建议先定位出是否存在数据不匹配的问题。最简单的验证方式就是可以从训练集中挑选出一部分数据作为验证集，重新划分后训练和验证模型表现。

**20.2. 举例常见几个数据不匹配的场景？** 例如设计款识别物体的 app 时，实际场景的图片均来自于手机拍摄，而训练集确是来自于网上各类抓取下来的图片。例如在图像去噪、去模糊、去雾、超分辨率等图像处理场景时，由于大量数据的难以获取，因此都会采用人为假设合成的图像进行训练，这时候应用到实际场景中也容易出现不匹配的问题

**20.3. 如何解决数据不匹配问题？** 数据不匹配是个很难有固定方法来解决的问题。这里提供几条供参考的途径：1、收集更多符合实际场最需要的数据。这似乎是最简单但也最难方式 2、对结果做错误分析。找出数据集中出错的数据和正确数据之间的特点和区别，这对你无论是进行后续模型的分析或者是数据的处理提供非常有效的思路。注意，这里的数据集包括训练集和测试集 3、数据集增强。数据集增强并不意味着数据集越大越好，其目的是丰富数据的分布以适应更多的变化当遇到数据不匹配时，对数据处理般可以有两种方式。其一，合成或处理更多接近需要的数据特点。其二，对所有数据包括实际场景数据都进行处理，将所有数据都统一到另一个分布上，统一出一种新的特点。

**20.4. 如何提高深度学习系统的性能.** 当我们要试图提高深度学习系统的性能时，目前我们大致可以从三方面考虑：

- 1、提高模型的结构，比如增加神经网络的层数，或者将简单的神经元单位换成复杂的 LSTM 神经元，比如在自然语言处理领域内，利用 LSTM 模型挖掘语法分析的优势。
- 2、改进模型的初始化方式，保证早期梯度具有某些有益的性质，或者具备大量的稀疏性，或者利用线性代数原理的优势。
- 3、选择更强大的学习算法，比如对度梯度更新的方式，也可以是采用除以先前梯度 L2 范数来更新所有参数，甚至还可以选用计算代价较大的二阶算法。







## 1. 超参数概念

**1.1. 什么是超参数，参数和超参数的区别？** 区分两者最大的一点就是是否通过数据来进行调整，模型参数通常是有数据来驱动调整，超参数则不需要数据来驱动，而是在训练前或者训练中人为的进行调整的参数。例如卷积核的具体核参数就是指模型参数，这是有数据驱动的。而学习率则是人为来进行调整的超参数。这里需要注意的是，通常情况下卷积核数量、卷积核尺寸这些也是超参数，注意与卷积核的核参数区分。

**1.2. 神经网络中包含哪些超参数？** 通常可以将超参数分为三类：网络参数、优化参数、正则化参数。

网络参数：可指网络层与层之间的交互方式（相加、相乘或者串接等）、卷积核数量和卷积核尺寸、网络层数（也称深度）和激活函数等。

优化参数：一般指学习率（learning rate）、批样本数量（batch size）、不同优化器的参数以及部分损失函数的可调参数。

正则化：权重衰减系数，丢弃比率（dropout）

**1.3. 为什么要进行超参数调优？** 本质上，这是模型优化寻找最优解和正则项之间的关系。网络模型优化调整的目的是为了寻找到全局最优解（或者相比更好的局部最优解），而正则项又希望模型尽量拟合到最优。两者通常情况下，存在一定的对立，但两者的目标是一致的，即最小化期望风险。模型优化希望最小化经验风险，而容易陷入过拟合，正则项用来约束模型复杂度。所以如何平衡两者之间的关系，得到最优或者较优的解就是超参数调整优化的目的。

### 1.4. 14.2.4 超参数的重要性顺序.

- 首先，**学习率，损失函数上的可调参数**。在网络参数、优化参数、正则化参数中最重要的超参数可能就是学习率了。学习率直接控制着训练中网络梯度更新的量级，直接影响着模型的**有效容限能力**；损失函数上的可调参数，这些参数通常情况下需要结合实际的损失函数来调整，大部分情况下这些参数也能很直接的影响到模型的**的有效容限能力**。这些损失一般可分成三类，第一类辅助损失结合常见的损失函数，起到辅助优化特征表达的作用。例如度量学习中的 Center loss，通常结合交叉熵损失伴随一个权重完成一些特定的任务。这种情况下一般建议辅助损失值不高于或者不低于交叉熵

损失值的两个数量级；第二类，多任务模型的多个损失函数，每个损失函数之间或独立或相关，用于各自任务，这种情况取决于任务之间本身的相关性，目前笔者并没有一个普适的经验由于提供参考；第三类，独立损失函数，这类损失通常会在特定的任务有显著性的效果。例如 RetinaNet 中的 focal loss，其中的参数  $\alpha$ ,  $\gamma$ ，对最终的效果会产生较大的影响。这类损失通常论文中会给出特定的建议值。

- 其次，**批样本数量，动量优化器（Gradient Descent with Momentum）的动量参数**。批样本决定了数量梯度下降的方向。过小的批数量，极端情况下，例如 batch size 为 1，即每个样本都去修正一次梯度方向，样本之间的差异越大越难以收敛。若网络中存在批归一化（batchnorm），batch size 过小则更难以收敛，甚至垮掉。这是因为数据样本越少，统计量越不具有代表性，噪声也相应的增加。而过大的 batch size，会使得梯度方向基本稳定，容易陷入局部最优解，降低精度。一般参考范围会取在 [1:1024] 之间，当然这个不是绝对的，需要结合具体场景和样本情况；动量衰减参数 是计算梯度的指数加权平均数，并利用该值来更新参数，设置为 0.9 是一个常见且效果不错的选择；
- 最后，**Adam 优化器的超参数、权重衰减系数、丢弃法比率（dropout）和网络参数**。在这里说明下，这些参数重要性放在最后并不等价于这些参数不重要。而是表示这些参数在大部分实践中**不建议过多尝试**，例如 Adam 优化器中的  $\beta_1$ ,  $\beta_2$ ,  $\epsilon$ ，常设为 0.9、0.999、 $10^{-8}$  就会有不错的表现。权重衰减系数通常会有个建议值，例如 0.0005，使用建议值即可，不必过多尝试。dropout 通常会在全连接层之间使用防止过拟合，建议比率控制在 [0.2,0.5] 之间。使用 dropout 时需要特别注意两点：一、在 RNN 中，如果直接放在 memory cell 中，循环会放大噪声，扰乱学习。一般会建议放在输入和输出层；二、不建议 dropout 后直接跟上 batchnorm，dropout 很可能影响 batchnorm 计算统计量，导致方差偏移，这种情况下会使得推理阶段出现模型完全垮掉的极端情况；网络参数通常也属于超参数的范围内，通常情况下增加网络层数能增加模型的容限能力，但模型真正有效的容限能力还和样本数量和质量、层之间的关系等有关，所以一般情况下会选择先固定网络层数，调优到一定阶段或者有大量的硬件资源支持可以在网络深度上进行进一步调整。

### 如何影响模型

超参数	容量	原因	注意事项
学习率	调至最优，提升有效容量	过高或者过低的学习率，都会由于优化失败而导致降低模型有效容限	学习率最优点，在训练的不同时间点都可能变化，所以需要一套有效的学习率衰减策略

如何影响模型			
超参数	容量	原因	注意事项
损失函数部分超参数	调至最优，提升有效容量	损失函数超参数大部分情况都会可能影响优化，不合适的超参数会使即便是对目标优化非常合适的损失函数同样难以优化模型，降低模型有效容限。	对于部分损失函数超参数其变化会对结果十分敏感，而有些则并不会太影响。在调整时，建议参考论文的推荐值，并在该推荐值数量级上进行最大最小值调试该参数对结果的影响。
批样本数量	过大过小，容易降低有效容量	大部分情况下，选择适合自身硬件容量的批样本数量，并不会对模型容限造成。	在一些特殊的目标函数的设计中，如何选择样本是很可能影响到模型的有效容限的，例如度量学习（metric learning）中的 N-pair loss。这类损失因为需要样本的多样性，可能会依赖于批样本数量。
丢弃法	比率降低会提升模型的容量	较少的丢弃参数意味着模型参数量的提升，参数间适应性提升，模型容量提升，但不一定能提升模型有效容限	
权重衰减系数	调至最优，提升有效容量	权重衰减可以有效的起到限制参数变化的幅度，起到一定的正则作用	
优化器动量	调至最优，可能提升有效容量	动量参数通常用来加快训练，同时更容易跳出极值点，避免陷入局部最优解。	
模型深度	同条件下，深度增加，模型容量提升	同条件下，下增加深度意味着模型具有更多的参数，更强的拟合能力。	同条件下，深度越深意味着参数越多，需要的时间和硬件资源也越高。
卷积核尺寸	尺寸增加，模型容量提升	增加卷积核尺寸意味着参数量的增加，同条件下，模型参数也相应的增加。	

### 1.5. 14.2.5 部分超参数如何影响模型性能？

超参数	建议范围	注意事项
初始学习率	SGD: [1e-2, 1e-1] momentum: [1e-3, 1e-2] Adagrad: [1e-3, 1e-2] Adadelta: [1e-2, 1e-1] RMSprop: [1e-3, 1e-2] Adam: [1e-3, 1e-2] Adamax: [1e-3, 1e-2] Nadam: [1e-3, 1e-2]	这些范围通常是指从头开始训练的情况。若是微调，初始学习率可在降低一到两个数量级。
损失函数部分超参数	多个损失函数之间，损失值之间尽量相近，不建议超过或者低于两个数量级	这是指多个损失组合的情况，不一定完全正确。单个损失超参数需结合实际情况。
批样本数量	[1:1024]	当批样本数量过大（大于 6000）或者等于 1 时，需要注意学习策略或者内部归一化方式的调整。
丢弃概率	[0, 0.5]	
权重衰减系数	[0, 1e-4]	
卷积核尺寸	[7x7], [5x5], [3x3], [1x1], [7x1, 1x7]	

### 1.6. 14.2.6 部分超参数合适的范围.

## 2. 14.3 网络训练中的超参调整策略

**2.1. 14.3.1 如何调试模型？** 在讨论如何调试模型之前，我们先来纠正一个误区。通常理解如何调试模型的时候，我们想到一系列优秀的神经网络模型以及调试技巧。但这里需要指出的是数据才是模型的根本，如果有一批质量优秀的数据，或者说你能将数据质量处理的很好的时候，往往比挑选或者设计模型的收益来的更大。那在这之后才是模型的设计和挑选以及训练技巧上的事情。

1、探索和清洗数据。探索数据集是设计算法之前最为重要的一步，以图像分类为例，我们需要重点知道给定的数据集样本类别和各类别样本数量是否平衡，图像之间是否存在跨域问题（例如网上爬取的图像通常质量各异，存在噪声）。若是类别数远远超过类别样本数（比如类别 10000，每个类别却只有 10 张图像），那通常的方法可能效果并不显著，这时候 few-shot learning 或者对数据集做进一步增强可能是你比较不错的选择。再如目标检测，待检测目标在

数据集中的尺度范围是对检测器的性能有很大影响的部分。因此重点是检测大目标还是小目标、目标是否密集完全取决于数据集本身。所以，探索和进一步清洗数据集一直都是深度学习中最重要的一步。这是很多新手通常会忽略的一点。

2、探索模型结果。探索模型的结果，通常是需要对模型在验证集上的性能进行进一步的分析，这是如何进一步提升模型性能很重要的步骤。将模型在训练集和验证集都进行结果的验证和可视化，可直观的分析出模型是否存在较大偏差以及结果的正确性。以图像分类为例，若类别间样本数量很不平衡时，我们需要重点关注少样本类别在验证集的结果是否和训练集的出入较大，对出错类别可进一步进行模型数值分析以及可视化结果分析，进一步确认模型的行为。

3、监控训练和验证误差。首先很多情况下，我们忽略代码的规范性和算法撰写正确性验证，这点上容易产生致命的影响。在训练和验证都存在问题时，首先请确认自己的代码是否正确。其次，根据训练和验证误差进一步追踪模型的拟合状态。若训练数据集很小，此时监控误差则显得格外重要。确定了模型的拟合状态对进一步调整学习率的策略的选择或者其他有效超参数的选择则会更得心应手。

4、反向传播数值的计算，这种情况通常适合自己设计一个新操作的情况。目前大部分流行框架都已包含自动求导部分，但并不一定是完全符合你的要求的。验证求导是否正确的方式是比较自动求导的结果和有限差分计算结果是否一致。所谓有限差分即导数的定义，使用一个极小的值近似导数。

$$f'(x_0) = \lim_{n \rightarrow 0} \frac{\Delta y}{\Delta x} = \lim_{n \rightarrow 0} \frac{f(x_0 + \Delta x - f(x_0))}{\Delta x}$$

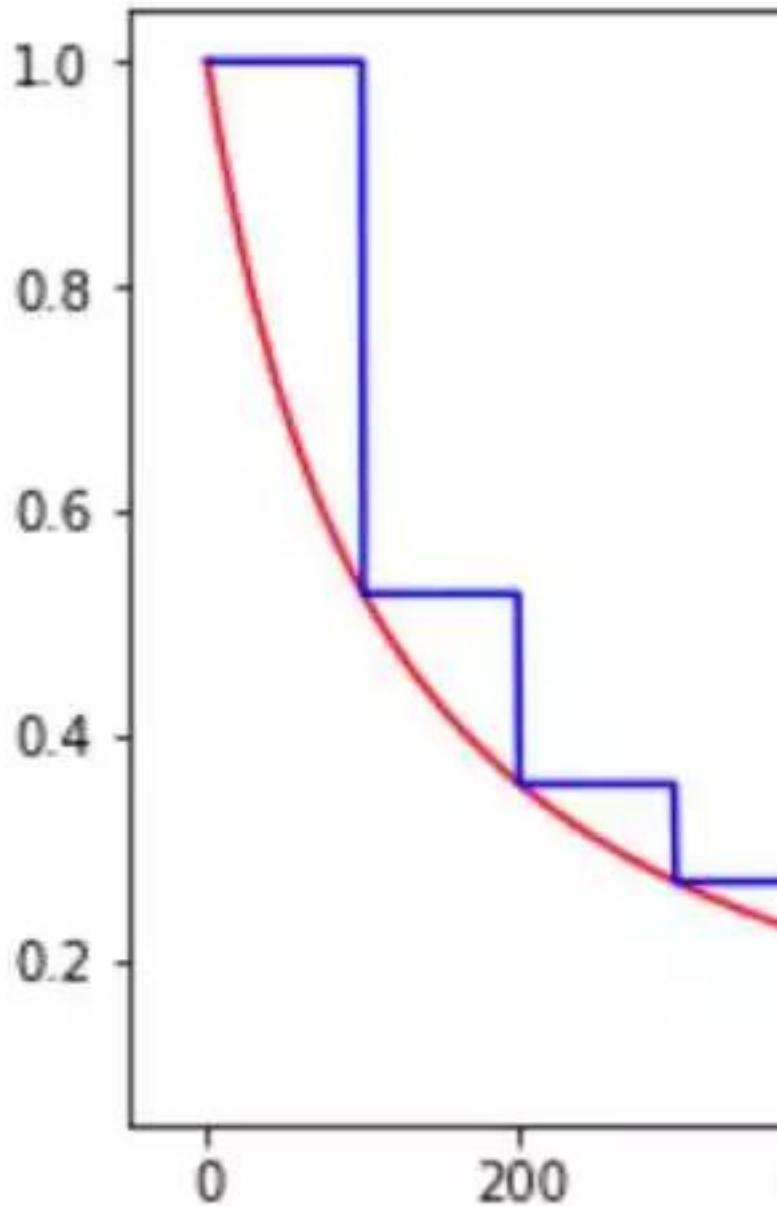
**2.2. 14.3.2 为什么要学习率调整？** 学习率可以说是模型训练最为重要的超参数。通常情况下，一个或者一组优秀的学习率既能加速模型的训练，又能得到一个较优甚至最优的精度。过大或者过小的学习率会直接影响到模型的收敛。我们知道，当模型训练到一定程度的时候，损失将不再减少，这时候模型的一阶梯度接近零，对应 Hessian 矩阵通常是两种情况，一、正定，即所有特征值均为正，此时通常可以得到一个局部极小值，若这个局部极小值接近全局最小则模型已经能得到不错的性能了，但若差距很大，则模型性能还有待于提升，通常情况下后者在训练初最常见。二，特征值有正有负，此时模型很可能陷入了鞍点，若陷入鞍点，模型性能表现就很差。以上两种情况在训练初期以及中期，此时若仍然以固定的学习率，会使模型陷入左右来回的震荡或者鞍点，无法继续优化。所以，学习率衰减或者增大能帮助模型有效的减少震荡或者逃离鞍点。

**2.3. 14.3.3 学习率调整策略有哪些？** 通常情况下，大部分学习率调整策略都是衰减学习率，但有时若增大学习率也同样起到奇效。这里结合 TensorFlow 的内置方法来举例。

### 1、`exponential_decay` 和 `natural_exp_decay`

#### 4、`inverse_time_decay`

```
[] inverse_time_decay(learning_rate, global_step, decay_steps, decay_rate, staircase=False, name=None)
```



逆时衰减,这种方式和指数型类似。如图,

#### 5、`cosine_decay`

**2.4. 14.3.4 极端批样本数量下，如何训练网络？** 极端批样本情况一般是指 batch size 为 1 或者 batch size 在 6000 以上的情况。这两种情况，在使用不合理的情况下都会导致模型最终性能无法达到最优甚至是崩溃的情况。

在目标检测、分割或者 3D 图像等输入图像尺寸较大的场景，通常 batch size 会非常小。而在 14.2.4 中，我们已经讲到这种情况会导致梯度的不稳定以及 batchnorm 统计的不准确。针对梯度不稳定的问题，通常不会太致命，若训练中发现梯度不稳定导致性能的严重降低时可采用累计梯度的策略，即每次计算完不反向更新，而是累计多次的误差后进行一次更新，这是一种在内存有限情况下实现有效梯度更新的一个策略。batch size 过小通常对 batchnorm 的影响是最大的，若网络模型中存在 batchnorm，batch size 若只为 1 或者 2 时会对训练结果产生非常大的影响。这时通常有两种策略，一、若模型使用了预训练网络，可冻结预训练网络中 batchnorm 的模型参数，有效降低 batch size 引起的统计量变化的影响。二、在网络不是过深或者过于复杂时可直接移除 batchnorm 或者使用 groupnorm 代替 batchnorm，前者不多阐释，后者是有 FAIR 提出的一种用于减少 batch 对 batchnorm 影响，其主要策略是先将特征在通道上进行分组，然后在组内进行归一化。即归一化操作上完全与 batch size 无关。这种 groupnorm 的策略被证实在极小批量网络训练上能达到较优秀的性能。当然这里也引入里 group 这个超参数，一般情况下建议不宜取 group 为 1 或者各通道单独为组的 group 数量，可结合实际网络稍加调试。

为了降低训练时间的成本，多机多卡的分布式系统通常会使用超大的 batch size 进行网络训练。同样的在 14.2.4 中，我们提到了超大 batch size 会带来梯度方向过于一致而导致的精度大幅度降低的问题。这时通常可采用层自适应速率缩放 (LARS) 算法。从理论认知上将，batch size 增大会减少反向传播的梯度更新次数，但为了达到相同的模型效果，需要增大学习率。但学习率一旦增大，又会引起模型的不收敛。为了解决这一矛盾，LARS 算法就在各层上自适应的计算一个本地学习率用于更新本层的参数，这样能有效的提升训练的稳定性。目前利用 LARS 算法，腾讯公司使用 65536 的超大 batch size 能将 ResNet50 在 ImageNet 在 4 分钟完成训练，而谷歌使用 32768 的 batch size 使用 TPU 能将该时间缩短至 2 分钟。

### 3. 14.4 合理使用预训练网络

**3.1. 14.4.1 什么是微调 (fine-tune)。** 微调 (fine-tune)，顾名思义指稍微调整参数即可得到优秀的性能，是迁移学习的一种实现方式。微调和从头训练 (train from scratch) 的本质区别在于模型参数的初始化，train from scratch 通常指对网络各类参数进行随机初始化（当然随机初始化也存在一定技巧），随机初始化模型通常不具有任何预测能力，通常需要大量的数据或者特定域的数据进行从零开始的训练，这样需要训练到优秀的模型通常是稍困难的。而微调的网络，网络各类参数已经在其他数据集（例如 ImageNet 数据集）完成较好调整的，具备了较优秀的表达能力。因此，我们只需要以较小的学习速率在自己所需的数据集领域进行学习

即可得到较为优秀的模型。微调通常情况下，无须再重新设计网络结构，预训练模型提供了优秀的结构，只需稍微修改部分层即可。在小数据集上，通常微调的效果比从头训练要好很多，原因在于数据量较小的前提下，训练更多参数容易导致过度拟合。

**3.2. 14.4.2 微调有哪些不同方法？** 以图像分类为例，通常情况下由于不同数据集需要的类别数不同，我们需要修改网络的输出顶层。这种情况下有两种微调方式：

- 不冻结网络模型的任何层，对最后的改动层使用较大的学习率，对未改动层以较小的学习率进行训练全模型训练，进行多轮训练即可。即一步完成训练。
- 冻除了顶部改动层以外的所有层参数，即不对冻结部分的层进行参数训练更新，进行若干轮的微调训练后，放开顶部层以下的若干层或者全部放开所有层的参数，再次进行若干轮训练即可。即分多步训练。

以上两种都属于微调。目前由于存在大量优秀的预训练模型，如何确定哪个模型适合自己的任务并能得到最佳性能需要花大量的时间探索。此时，上述的前者是种不错训练方式，你无须进行过多分步的操作。而当探索到一个比较适合的模型时，你不妨可以再次重新尝试下以第二种方式进行训练，或许能得到相比于前者稍高些的性能，因为小数据集上调整过多的参数过拟合的机率也会增大，当然这并不是绝对的。

**3.3. 14.4.3 微调先冻结底层，训练顶层的原因？** 14.12 中第二种冻结多步训练的方式。首先冻除了顶部改动层以外的所有层参数，对顶层进行训练，这个过程可以理解为顶层的域适应训练，主要用来训练适应模型的现有特征空间，防止顶层糟糕的初始化，对已经具备一定表达能力的层的干扰和破坏，影响最终的性能。之后，在很多深度学习框架教程中会使用放开顶层往下一半的层数，继续进行微调。这样的好处在于越底层的特征通常是越通用的特征，越往上其整体的高层次语义越完备，这通过感受野很容易理解。所以，若预训练模型的数据和微调训练的数据语义差异越大（例如 ImageNet 的预模型用于医学图像的训练），那越往顶层的特征语义差异就越大，因此通常也需要进行相应的调整。

#### 3.4. 14.4.4 不同的数据集特性下如何微调？

- 数据集数据量少，数据和原数据集类似。这是通常做法只需修改最后的输出层，训练即可，训练过多参数容易过拟合。
- 数据集数据量少，数据和原数据集差异较大。由于数据差异较大，可以在完成输出顶层的微调后，微调顶层往下一半的层数，进行微调。
- 数据集数据量大，数据与原数据集差异较大。这种情况下，通常已经不需要用预训练模型进行微调，通常直接重新训练即可。
- 数据集数据量大，数据与原数据类似。这时预训练模型的参数是个很好的初始化，可利用预训练模型放开所有层以较小的学习率微调即可。

**3.5. 14.4.4 目标检测中使用预训练模型的优劣？** 目标检测中无论是一阶段的 YOLO、SSD 或者 RetinaNet 还是二阶段的 Faster R-CNN、R-FCN 和 FPN 都是基于 ImageNet 上预训练好的分类模型。

优势在于：

1、正如大部分微调的情况一样，使用预训练网络已拥有优秀的语义特征，能有效的加快训练速度；

2、其次，对于大部分二阶段的模型来说，并未实现严格意义上的完全端对端的训练，所以使用预训练模型能直接提取到语义特征，能使两个阶段的网络更容易实现模型的优化。

劣势在于，分类模型和检测模型之间仍然存在一定任务上的差异：

1、分类模型大部分训练于单目标数据，对同时进行多目标的捕捉能力稍弱，且不关注目标的位置，在一定程度上让模型损失部分空间信息，这对检测模型通常是不利的；

2、域适应问题，若预训练模型（ImageNet）和实际检测器的使用场景（医学图像，卫星图像）差异较大时，性能会受到影响；

3、使用预训练模型就意味着难以自由改变网络结构和参数限制了应用场合。

**3.6. 14.4.5 目标检测中如何从零开始训练 (train from scratch) ?** 结合 FAIR 相关的研究，我们可以了解目标检测和其他任务从零训练模型一样，只要拥有足够的数据以及充分而有效的训练，同样能训练出不亚于利用预训练模型的检测器。这里我们提供如下几点建议：

1、数据集不大时，同样需要进行数据集增强。

2、预训练模型拥有更好的初始化，train from scratch 需要更多的迭代次数以及时间训练和优化检测器。而二阶段模型由于并不是严格的端对端训练，此时可能需要更多的迭代次数以及时间，而一阶段检测模型训练会相对更容易些（例如 DSOD 以 ScratchDet 及）。

3、目标检测中 train from scratch 最大的问题还是 batch size 过小。所以可采取的策略是增加 GPU 使用异步 batchnorm 增大 batch size，若条件限制无法使用更多 GPU 时，可使用 groupnorm 代替 batchnorm

4、由于分类模型存在对多目标的捕捉能力弱以及对物体空间位置信息不敏感等问题，可借鉴 DetNet 训练一个专属于目标检测的模型网络，增强对多目标、尺度和位置拥有更强的适应性。

#### 4. 14.5 如何改善 GAN 的性能

优化 GAN 性能通常需要在如下几个方面进行

- 设计或选择更适合目的代价函数。
- 添加额外的惩罚。
- 避免判别器过度自信和生成器过度拟合。

- 更好的优化模型的方法。
- 添加标签明确优化目标。

### GAN 常用训练技巧

- 输入规范化到 (-1, 1) 之间，最后一层的激活函数使用 tanh (BEGAN 除外)
- 使用 wassertein GAN 的损失函数，
- 如果有标签数据的话，尽量使用标签，也有人提出使用反转标签效果很好，另外使用标签平滑，单边标签平滑或者双边标签平滑
- 使用 mini-batch norm，如果不用 batch norm 可以使用 instance norm 或者 weight norm
- 避免使用 RELU 和 pooling 层，减少稀疏梯度的可能性，可以使用 leakrelu 激活函数
- 优化器尽量选择 ADAM，学习率不要设置太大，初始 1e-4 可以参考，另外可以随着训练进行不断缩小学习率，
- 给 D 的网络层增加高斯噪声，相当于是一种正则 ## 14.6 AutoML

**4.1. 14.6.1 什么是 AutoML ?.** 目前一个优秀的机器学习和深度学习模型，离不开这几个方面：

- 一、优秀的数据预处理；
- 二、合适的模型结构和功能；
- 三、优秀的训练策略和超参数；
- 四、合适的后处理操作；
- 五、严格的结果分析。

这几方面都对最终的结果有着举足轻重的影响，这也是目前的数据工程师和学者们的主要工作。但由于这每一方面都十分繁琐，尤其是在构建模型和训练模型上。而大部分情况下，这些工作有无须过深专业知识就能使用起来。所以 AutoML 主要的作用就是来帮助实现高效的模型构建和超参数调整。例如深度学习网络的架构搜索、超参数的重要性分析等等。当然 AutoML 并不简单的进行暴力或者随机的搜索，其仍然需要机器学习方面的知识，例如贝叶斯优化、强化学习、元学习以及迁移学习等等。目前也有些不错的 AutoML 工具包，例如 Alex Honchar 的 Hyperopt、微软的 NNI、Autokeras 等。

目前 AutoML 已经成为最新的研究热点，有兴趣的可以参考 [AutoML literature](#)。

**4.2. 14.6.2 自动化超参数搜索方法有哪些 ?.** 目前自动化搜索主要包含网格搜索，随机搜索，基于模型的超参优化

网格搜索：

通常当超参数量较少的时候，可以使用网格搜索法。即列出每个超参数的大致候选集合。利用这些集合进行逐项组合优化。在条件允许的情况下，重复进行网格搜索会当优秀，当然每

基于模型的超参优化：

有别于上述两种的搜索策略，基于模型的超参调优问题转化为了优化问题。直觉上会考虑是否进行一个可导建模，然后利用梯度下降进行优化。但不幸的是我们的超参数通常情况下是离散的，而且其计算代价依旧很高。

基于模型的搜索算法，最常见的就是贝叶斯超参优化。有别于的网格搜索和随机搜索独立于前几次搜索结果的搜索，贝叶斯则是利用历史的搜索结果进行优化搜索。其主要有四部分组成，1. 目标函数，大部分情况下就是模型验证集上的损失。2、搜索空间，即各类待搜索的超参数。3、优化策略，建立的概率模型和选择超参数的方式。4、历史的搜索结果。首先对搜索空间进行一个先验性的假设猜想，即假设一种选择超参的方式，然后不断的优化更新概率模型，最终的目标是找到验证集上误差最小的一组超参数。

**4.3. 14.6.3 什么是神经网络架构搜索 (NAS).** 2015 至 2017 年间，是 CNN 网络设计最兴盛的阶段，大多都是由学者人工设计的网络结构。这个过程通常会很繁琐。其主要原因在于对不同模块组件的组成通常是个黑盒优化的问题，此外，在不同结构超参数以及训练超参数的选择优化上非凸优化问题，或者是个混合优化问题，既有离散空间又有连续空间。NAS (Neural Architecture Search) 的出现就是为了解决如何通过机器策略和自动化的方式设计出优秀高效的网络。而这种策略通常不是统一的标准，不同的网络结合实际的需求通常会有不同的设计，比如移动端的模型会在效率和精度之间做平衡。目前，NAS 也是 AUTOML 中最重要的部分。NAS 通常会分为三个方面，搜索空间（在哪搜索），搜索策略（如何搜索）及评价预估。

- 搜索空间，即在哪搜索，定义了优化问题所需变量。不同规模的搜索空间的变量其对于的难度也是不一样的。早期由于网络结构以及层数相对比较简单，参数量较少，因此会更多的使用遗传算法等进化算法对网络的超参数和权重进行优化。深度学习发展到目前，模型网络结构越来越复杂，参数量级越来越庞大，这些进化算法已经无法继续使用。但若我们先验给定一些网络结构和超参数，模型的性能已经被限制在给定的空间，此时搜索的空间已变得有限，所以只需对复杂模型的架构参数和对应的超参数进行优化即可。
- 搜索策略，即如何搜索，定义了如何快速、准确找到最优的网络结构参数配置的策略。常见的搜索方法包括：随机搜索、贝叶斯优化、强化学习、进化算法以及基于模型的搜索算法。其中主要代表为 2017 年谷歌大脑的使用强化学习的搜索方法。
- 评价预估，定义了如何高效对搜索的评估策略。深度学习中，数据规模往往是庞大的，模型要在如此庞大的数据规模上进行搜索，这无疑是非常耗时的，对优化也会造成非常大的困难，所以需要一些高效的策略做近似的评估。这里一般会有如下三种思路：

一、使用些低保真的训练集来训练模型。低保真在实际中可以用不同的理解，比如较少的迭代次数，用一小部分数据集或者保证结构的同时减少通道数等。这些方法都可以在测试优化

### NASNet 的 RNN 控制器

2、对 RNN 控制进行优化，先验性地将各种尺寸和类型的卷积和池化层加入到搜索空间内，用预测一个卷积模块代替原先预测一层卷积。如图，控制器 RNN 不在预测单个卷积内的超参数组成，而是对一个模块内的每一个部分进行搜索预测，搜索的空间则限定在如下这些操作中：

- identity • 1x3 then 3x1 convolution • 1x7 then 7x1 convolution • 3x3 dilated convolution • 3x3 average pooling • 3x3 max pooling • 5x5 max pooling • 7x7 max pooling
- 1x1 convolution • 3x3 convolution • 3x3 depthwise-separable conv • 5x5 depthwise-separable conv • 7x7 depthwise-separable conv

在模块内的连接方式上也提供了 element-wise addition 和 concatenate 两种方式。NASNet 的搜索方式和过程对 NAS 的一些后续工作都具有非常好的参考借鉴意义。

**4.5. 14.6.5 网络设计中，为什么卷积核设计尺寸都是奇数.** 我们发现在很多大部分网络设计时都会使用例如 3x3/5x5/7x7 等奇数尺寸卷积核，主要原因有两点：

- 保证像素点中心位置，避免位置信息偏移
- 填充边缘时能保证两边都能填充，原矩阵依然对称

### 4.6. 14.6.6 网络设计中，权重共享的形式有哪些，为什么要权重共享. 权重共享的形式：

- 深度学习中，权重共享最具代表性的就是卷积网络的卷积操作。卷积相比于全连接神经网络参数大大减少；
- 多任务网络中，通常为了降低每个任务的计算量，会共享一个骨干网络。
- 一些相同尺度下的结构化递归网络

权重共享的好处：

权重共享一定程度上能增强参数之间的联系，获得更好的共性特征。同时很大程度上降低了网络的参数，节省计算量和计算所需内存（当然，结构化递归并不节省计算量）。此外权重共享能起到很好正则的作用。正则化的目的是为了降低模型复杂度，防止过拟合，而权重共享则正好降低了模型的参数和复杂度。

因此一个设计优秀的权重共享方式，在降低计算量的同时，通常会较独享网络有更好的效果。

权重共享不仅在人工设计 (human-invented) 的网络结构中有简化参数，降低模型复杂度的作用，在神经网络搜索 (NAS) 的网络结构中可以使得 child model 的计算效率提升，使得搜索过程可以在单卡 GPU 上复现，见 Efficient NAS([ENAS](#))。





## 第十五章 异构计算，GPU 和框架选型指南

深度学习训练和推理的过程中，会涉及到大量的向量（vector），矩阵（matrix）和张量（tensor）操作，通常需要大量的浮点计算，包括高精度（在训练的时候）和低精度（在推理和部署的时候）。GPU，作为一种通用可编程的加速器，最初设计是用来进行图形处理和渲染功能，但是从 2007 年开始，英伟达（NVIDIA）公司提出了第一个可编程通用计算平台（GPU），同时提出了 CUDA 框架，从此开启了 GPU 用于通用计算的新纪元。此后，不计其数的科研人员和开发者，对各种不同类型的算法用 CUDA 进行（部分）改写，从而达到几倍到数百倍的加速效果。尤其是在机器学习，特别是深度学习的浪潮来临后，GPU 加速已经是各类工具实现的基本底层构架之一。本章里，会简单介绍 GPU 的基本架构，性能指标，框架选择等等和深度学习相关的内容。

**0.1. 15.1 什么是异构计算？** 异构计算是基于一个更加朴素的概念，“异构现象”，也就是不同计算平台之间，由于硬件结构（包括计算核心和内存），指令集和底层软件实现等方面的不同而有着不同的特性。异构计算就是使用结合了两个或者多个不同的计算平台，并进行协同运算。比如，比较常见的，在深度学习和机器学习中已经比较成熟的架构：CPU 和 GPU 的异构计算；此外还有比较新的 Google 推出的协处理器（TPU），根据目的而定制的 ASIC，可编程的 FPGA 等也都是现在在异构计算中使用比较多的协处理器。而，本章中会着重介绍和深度学习共同繁荣的图形加算器，也就是常说的 GPU。

**0.2. 15.2 什么是 GPU？** GPU，就如名字所包含的内容，原本开发的目的是为了进行计算机图形渲染，而减少对于 CPU 的负载。由于图像的原始特性，也就是像素间的独立性，所以 GPU 在设计的时候就遵从了“单指令流多数据流（SIMD）”架构，使得同一个指令（比如图像的某种变换），可以同时在多一个像素点上进行计算，从而得到比较大的吞吐量，才能使得计算机可以实时渲染比较复杂的 2D/3D 场景。在最初的应用场景里，GPU 并不是作为一种通用计算平台出现的，直到 2007 年左右，一家伟大的公司将 GPU 带到通用计算的世界里，使得其可以在相对比较友好的编程环境（CUDA/OpenCL）里加速通用程序成了可能。从此之后，GPU 通用计算，也就是 GPU 就成了学界和工业界都频繁使用的技术，在深度学习爆发的年代里，GPU 成了推动这股浪潮非常重要的力量。

我们可以从图中读出两点信息：

1. 在同一个系列里面，价格和性能大体上成正比。但后发布的型号性价比更高，例如 980 TI 和 1080 TI。
2. GTX 1000 系列比 900 系列在性价比上高出 2 倍左右。

如果大家继续比较 GTX 较早的系列，也可以发现类似的规律。据此，我们推荐大家在能力范围内尽可能买较新的 GPU。

对于 RTX 系列，新增了 Tensor Cores 单元及支持 FP16，使得显卡的可选择范围更加多元。

#### 0.5.2. 15.5.2 购买建议. 首先给出一些总体的建议：

性价比高但较贵：RTX 2070, GTX 1080 Ti

性价比高又便宜：RTX 2060, GTX 1060 (6GB)

当使用数据集 > 250GB：GTX Titan X (Maxwell), NVIDIA Titan X Pascal 或 NVIDIA Titan Xp

没有足够的钱：GTX 1060 (6GB)

几乎没有钱，入门级：GTX 1050 Ti (4GB)

做 Kaggle 比赛：RTX 2070、GTX 1060 (6GB) 适用于任何“正常”比赛，GTX 1080 Ti (预算足够可以选择 RTX 2080 Ti) 用于“深度学习竞赛”

计算机视觉研究员：RTX 2080 Ti (涡轮散热或水冷散热较好，方便后期增加新的显卡)  
如果网络很深可以选择 Titan RTX

一名 NLP 研究人员：RTX 2080 Ti，并使用 FP16 来训练

搭建一个 GPU 集群：这个有点复杂，另做探讨。

刚开始进行深度学习研究：从 RTX 2060 或 GTX 1060 (6GB) 开始，根据你下一步兴趣 (入门，Kaggle 比赛，研究，应用深度学习) 等等，再进行选择。目前，RTX 2060 和 GTX 1060 都比较合适入门的选择。

想尝试下深度学习，但没有过多要求：GTX 1050 ti (4 或 2GB)

目前独立 GPU 主要有 AMD 和 Nvidia 两家厂商。其中 Nvidia 在深度学习布局较早，对深度学习框架支持更好。因此，目前大家主要会选择 Nvidia 的 GPU。

Nvidia 有面向个人用户（例如 GTX 系列）和企业用户（例如 Tesla 系列）的两类 GPU。这两类 GPU 的计算能力相当。然而，面向企业用户的 GPU 通常使用被动散热并增加了内存校验，从而更适合数据中心，并通常要比面向个人用户的 GPU 贵上 10 倍。

如果你是拥有 100 台机器以上的大公司用户，通常可以考虑针对企业用户的 Nvidia Tesla 系列。如果你是拥有 10 到 100 台机器的实验室和中小公司用户，预算充足的情况下可以考虑 Nvidia DGX 系列，否则可以考虑购买如 Supermicro 之类的性价比比较高的服务器，然后再购买安装 GTX 系列的 GPU。

<b>Architecture</b>	x86_64	ppc64le
<b>Distribution</b>	Fedora	OpenSUSE
	SLES	Ubuntu
<b>Version</b>	16.04	14.04
<b>Installer Type</b> 	runfile (local)	deb (local)
	cluster (local)	deb (network)

## Download Installers for Linux Ubuntu 16.04 x86\_64

The base installer is available for download below.

There is 1 patch available. This patch requires the base installer to be installed first.

### » Base Installer

[Download \(1\)](#)

#### Installation Instructions:

1. Run `sudo  
sh cuda\_8.0.61\_375.26\_linux.run`
2. Follow the command-line prompts

! [cuda9.0] (./img/ch15/cuda9.0.png)

2. 命令行中进入到 cuda 所在的位置，授予运行权限：

```
cuda8.0: sudo chmod +x cuda_8.0.61_375.26_linux.run
```

```
cuda9.0: sudo chmod +x cuda_9.0.176_384.81_linux.run
```

3. 执行命令安装 cuda：

```
cuda8.0: sudo sh cuda_8.0.61_375.26_linux.run
```

```
cuda9.0: sudo sh cuda_9.0.176_384.81_linux.run
```

之后命令之后下面就是安装步骤，cuda8.0 和 cuda9.0 几乎一致：

- 首先出现 cuda 软件的版权说明，可以直接按 q 键跳过阅读
- Do you accept the previously read EULA? accept/decline/quit: **accept**
- Install NVIDIA Accelerated Graphics Driver for Linux-x86\_64 384.81? (y)es/(n)o/(q)uit:**no**
- Install the CUDA 9.0 Toolkit? (y)es/(n)o/(q)uit:**yes**
- Enter Toolkit Location [ default is /usr/local/cuda-9.0 ]: 直接按 enter 键即可
- Do you want to install a symbolic link at /usr/local/cuda? (y)es/(n)o/(q)uit:**yes**
- Install the CUDA 9.0 Samples? (y)es/(n)o/(q)uit:**yes**

以上步骤基本就是 cuda 的安装步骤。

#### 4. 安装 cudnn

cudnn 是 Nvidia 的专门针对深度学习的加速库。。。

0.6.3. 15.6.3 本机安装还是使用 docker ?.

0.6.4. 15.6.4 GPU 驱动问题.

### 0.7. 15.7 框架选择.

0.7.1. 15.7.1 主流框架比较. (一个大表格比较)

0.7.2. 15.7.2 框架详细信息.

- Tensorflow

Tensorflow 是 Google 于 2015 年开源的基于数据流编程的深度学习框架，得益于 Google 强大的技术实力和品牌背书，目前 Tensorflow 发展迅猛，其用户量远远超过其它框架用户。

优点：

1. 由谷歌开发、维护，因此可以保障支持、开发的持续性

2. 巨大、活跃的社区

3. 网络训练的低级、高级接口

4. 「TensorBoard」是一款强大的可视化套件，旨在跟踪网络拓扑和性能，使调试更加简单

5. TensorFlow 不仅支持深度学习，还有支持强化学习和其他算法的工具缺点：

6. 计算图是纯 Python 的, 因此速度较慢
7. 图构造是静态的, 意味着图必须先被「编译」再运行
  - PyTorch pytorch 是 Facebook 于 2017 年才推出的深度学习框架, 相对于其它框架, 算是比较晚的了, 但是这个同时也是优势, 在设计的时候就会避免很多之前框架的问题, 所以一经推出, 就收到大家极大的欢迎优点:

1. 接口简洁且规范, 文档齐全, 和 python 无缝结合,
2. 社区非常活跃, 开源实现较多
3. 提供动态计算图 (意味着图是在运行时生成的), 允许你处理可变长度的输入和输出, 例如, 在使用 RNN 时非常有用
4. 易于编写自己的图层类型, 易于在 GPU 上运行
5. 「TensorBoard」缺少一些关键功能时, 「Losswise」可以作为 Pytorch 的替代品

缺点:

1. 模型部署相对其它框架稍有劣势, 不过后续的 pytorch1.0 版本应该会有很大改善, 和 caffe2 合并后, caffe2 的优秀的模型部署能力可以弥补这个不足 2. 3.

相关资源链接:

1. 官网教程:<https://pytorch.org/tutorials/>
2. 基于 pytorch 的开源项目汇总:<https://github.com/bharathgs/Awesome-pytorch-list>
- 3.

- Keras Keras 是一个更高级、对用户最友好的 API, 具有可配置的后端, 由 Google Brain 团队成员 Francis Chollet 编写和维护优点:

1. 提供高级 API 来构建深度学习模型, 使其易于阅读和使用
2. 编写规范的文档
3. 大型、活跃的社区
4. 位于其他深度学习库 (如 Theano 和 TensorFlow, 可配置) 之上
5. 使用面向对象的设计, 因此所有内容都被视为对象 (如网络层、参数、优化器等)。所有模型参数都可以作为对象属性进行访问缺点:

6. 由于用途非常普遍, 所以在性能方面比较欠缺
7. 与 TensorFlow 后端配合使用时会出现性能问题 (因为并未针对其进行优化), 但与 Theano 后端配合使用时效果良好
8. 不像 TensorFlow 或 PyTorch 那样灵活
  - Sonnet

- Caffe

caffe 是第一个主流产品级深度学习库，于 2014 年由 UC Berkeley 发布开源优点：

1. 简单网络结构无需编写代码，可快速实现
2. 漂亮的 Matlab 和 Python 接口
3. 完全由 c++ 编程实现，部署方便

缺点：

1. 不灵活。在 Caffe 中，每个节点被当做一层，因此如果你想要一种新的层类型，你需要定义完整的前向、后向和梯度更新过程。这些层是网络的构建模块，你需要在无穷无尽的列表中进行选择。（相反，在 TensorFlow 中，每个节点被做一个张量运算例如矩阵相加、相乘或卷积。你可以轻易地定义一个层作为这些运算的组合。因此 TensorFlow 的构建模块更小巧，允许更灵活的模块化。）
2. 需要大量的非必要冗长代码。如果你希望同时支持 CPU 和 GPU，你需要为每一个实现额外的函数。你还需要使用普通的文本编辑器来定义你的模型。真令人头疼！几乎每个人都希望程序化地定义模型，因为这有利于不同组件之间的模块化。有趣的是，Caffe 的主要架构师现在在 TensorFlow 团队工作
3. 专一性。仅定位在计算机视觉（但做得很不错）
4. 不是以 Python 编写！如果你希望引入新的变动，你需要在 C++ 和 CUDA 上编程（对于更小的变动，你可以使用它的 Python 和 Matlab 接口）
5. 糟糕的文档
6. 安装比较困难！有大量的依赖包

- Caffe2

- MxNet MxNet 是 dmlc 社区推出的深度学习框架，MXNet 由学术界发起，包括数个顶尖大学的多个学科的研究人员的贡献，在 2017 年被亚马逊指定为官方框架。mxnet 的最知名的优点就是其对多 GPU 的支持和扩展性强，其优秀的性能使之在工业界占有一席之地，在 amazon 支持之后，其文档和开发进度明显好很多。除了高可扩展性，MXNet 还提供混合编程模型（命令式和声明式），同时兼容多种编程语言（包括 Python、C ++、R、Scala、Julia、Matlab 和 JavaScript）的代码，目前主要在推 python 高层接口 gluon

优点：

1. 多 GPU 支持好，扩展性强，支持多种编程语言接口，主要是由华人团队开发，中文社区活跃，中文文档资源和课程丰富
  2. 针对两大热门领域推出 gluoncv 和 gluonNLP 模块，复现经典论文，达到 State-of-the-art，接口设计简单，文档齐全，拿来就可以用
- 缺点：
1. 现在 mxnet 官方社区主要在推 gluon 接口，接口稍有混乱，坑较多，入手门槛稍高
  2. 偏小众，经典网络和项目的开源实现相对于 tensorflow 和 pytorch 还是比较少，很多还是需要自己手动实现相关资源链接：

1. 官方教程: <http://mxnet.incubator.apache.org> 提供有快速入门教程和详细文档说明
2. 中文教程: <http://zh.gluon.ai/> 官方的中文教程, 此课程有对应的中文版视频, 主要由李沐大神讲课
3. 中文论坛: <https://discuss.gluon.ai/> 官方发中文论坛, mxnet 的主要作者都在这里, 论坛比较活跃, 可及时得到作者的回答
4. 基于 mxnet 的开源项目实现: <https://github.com/chinakook/Awesome-MXNet> 这里主要列举了 mxnet 在各个领域的项目的开源实现

- CNTK
- PaddlePaddle
- 其他国内自主开发开源框架

#### 0.7.3. 15.7.3 哪些框架对于部署环境友好 ?.

- Tensorflow Serving
- ONNX 标准
- TensorRT
- ONNPACK
- Clipper

#### 0.7.4. 15.7.4 移动平台的框架如何选择 ?.

- Tensorflow Lite
- Caffe2

### 0.8. 15.8 其他.

#### 0.8.1. 15.8.1 多 GPU 环境的配置 .

- Tensorflow
- PyTorch

#### 0.8.2. 15.8.2 是不是可以分布式训练 ?.

#### 0.8.3. 15.8.3 可以在 SPARK 环境里训练或者部署模型吗 ?.

#### 0.8.4. 15.8.4 怎么进一步优化性能 ?.

- TVM
- nGraph

#### 0.8.5. 15.8.5 TPU 和 GPU 的区别 ?.

#### 0.8.6. 15.8.6 未来量子计算对于深度学习等 AI 技术的影响 ?.

---

**0.9. 15.1 GPU 购买指南.** 深度学习训练通常需要大量的计算资源。GPU 目前是深度学习最常使用的计算加速硬件。相对于 CPU 来说, GPU 更便宜且计算更加密集。一方面, 相同计算能力的 GPU 的价格一般是 CPU 价格的十分之一。另一方面, 一台服务器通常可以搭载

8 块或者 16 块 GPU。因此，GPU 数量可以看作是衡量一台服务器的深度学习计算能力的一个标准。

#### 0.9.1. 15.1.1 如何选择 GPU.

0.9.2. 15.1.2 GPU 的主要性能指标. 在选择 GPU 时，首先要考虑的第一个 GPU 性能问题是是什么呢：是否为 cuda 核心？时钟速度多大？内存大小多少？这些都不是，对于深度学习性能而言，最重要的特征是内存带宽（memory bandwidth）。简而言之：GPU 针对内存带宽进行了优化，但同时牺牲了内存访问时间（延迟）。CPU 的设计恰恰相反：如果涉及少量内存（例如几个数字相乘  $(3 * 6 * 9)$ ），CPU 可以快速计算，但是对于大量内存（如矩阵乘法  $(A * B * C)$ ）则很慢。由于内存带宽的限制，当涉及大量内存的问题时，GPU 快速计算的优势往往会受到限制。当然，GPU 和 CPU 之间还有更复杂的区别，关于为何 GPU 如此适用于处理深度学习问题，另做探讨。

所以如果你想购买一个快速的 GPU，首先要关注的是 GPU 的带宽（bandwidth）。

0.9.3. 15.1.3 整机配置. 通常，我们主要用 GPU 做深度学习训练。因此，不需要购买高端的 CPU。至于整机配置，尽量参考网上推荐的中高档的配置就好。不过，考虑到 GPU 的功耗、散热和体积，我们在整机配置上也需要考虑以下三个额外因素。

1. 机箱体积。GPU 尺寸较大，通常考虑较大且自带风扇的机箱。
2. 电源。购买 GPU 时需要查一下 GPU 的功耗，例如 50W 到 300W 不等。购买电源要确保功率足够，且不会过载机房的供电。
3. 主板的 PCIe 卡槽。推荐使用 PCIe 3.0 16x 来保证充足的 GPU 到主内存的带宽。如果搭载多块 GPU，要仔细阅读主板说明，以确保多块 GPU 一起使用时仍然是 16x 带宽。注意，有些主板搭载 4 块 GPU 时会降到 8x 甚至 4x 带宽。

#### 0.9.4. 15.1.4 小结.

- 在预算范围之内，尽可能买较新的 GPU。
- 整机配置需要考虑到 GPU 的功耗、散热和体积。

**0.10. 15.2 框架选型.** 目前常用的框架有 tensorflow, keras, pytorch, mxnet 等等，各个框架的优缺点在此简单介绍：

#### 0.10.1. 15.2.1 常用框架简介.

1. tensorflow: tensorflow 由于有 google 的强大背书，加上其优秀的分布式设计，丰富的教程资源和论坛，工业部署方便，基本很多人都是从 tensorflow 入门的优点：google 的强大背书，分布式训练，教程资源丰富，常见问题基本都可以在互联网中找到解决办法，工业部署方便缺点：接口混乱，官方文档不够简洁，清晰，
2. keras: keras 是一种高层编程接口，其可以选择不同的后端，比如 tensorflow, therao 等等优点：接口简洁，上手快，文档好，资源多缺点：封装的太好了导致不理解其技术细节

3. pytorch: PyTorch 是一个开源的 Python 机器学习库, 基于 Torch, 从官方 1.0 版本开始已经完美结合 caffe2, 主要应用于人工智能领域, 如自然语言处理。它最初由 Facebook 的人工智能研究团队开发. 优点: 文档清晰, 兼容 NumPy 的张量计算, 基于带基自动微分系统的深度神经网络, 由 Facebook 开发维护, 常见 model 都有 pytorch 复现版缺点: 工业部署稍弱 (但是合并 caffe2 后支持全平台部署)
4. mxnet mxnet 是 dmlc 社区推出的深度学习框架, 在 2017 年被亚马逊指定为官方框架优点: 支持多种语言, 代码设计优秀, 省显存, 华人团队开发, 中文社区活跃, 官方复现经典论文推出 gluoncv 和 gluonNLP 模块, 非常方便, 拿来就可以用。缺点: 现在 mxnet 官方社区主要在推 gluon 接口, 接口稍有混乱, 坑较多, 入手门槛稍高
5. caffe: 目前很多做深度学习比较早的大厂基本都是在用 caffe, 因为在 2013-2015 年基本就是 caffe 的天下, 并且 caffe 的代码设计很优秀, 基本所有代码都被翻了很多遍了, 被各种分析, 大厂基本都是魔改 caffe, 基于 caffe 来进行二次开发, 所在目前在很多大厂还是在使用 caffe 优点: 资源丰富, 代码容易理解, 部署方便缺点: 入门门槛高, 文档较少

框架选型总结: 1. 新手入门, 首推 pytorch, 上手快, 资源丰富, 官方文档写的非常好 (<https://pytorch.org/tutorials/>) 2. 目前工业部署, tensorflow 是首选, 资源丰富, 并且在分布式训练这一块基本一家独大 3. mxnet 的 gluon 接口有比较丰富的中文资源(教程: zh.gluon.ai, 论坛: discuss.gluon.ai), gluoncv 模块(<https://gluon-cv.mxnet.io>), gluonNLP 模块(<https://gluon-nlp.mxnet.io>)

**0.11. 15.3 模型部署.** 我们一般都是通过 python 或者其他语言来编码训练模型, 然后基于后端来进行部署一般的框架都有自身的部署框架, 比如 tensorflow, pytorch, caffe2, mxnet 等等有一些框架是专门做推理部署使用的, 比如 (1)tensorRT (2)TVM (3)ONNX

## 相关文献

- [1] Aston Zhang, Mu Li, Zachary C. Lipton, and Alex J. Smola. 《动手学深度学习》附录购买 GPU, 2019. [2] Tim Dettmers. Which GPU(s) to Get for Deep Learning: My Experience and Advice for Using GPUs in Deep Learning, 2019.







## CHAPTER 16

# 第十六章 NLP

Markdown Revision 1;

Date: 2018/11/14

Editor: 盛泳潘-电子科技大学;何建宏-学生

Contact: shengyp2011@163.com;Bonopengate@gmail.com

### 1. 16.0 NLP 发展史简述

50 多年来 NLP 的历史发展可以分为三个浪潮，前两波以理性主义和经验主义的形式出现，为当前的深度学习浪潮铺平了道路。NLP 的深层学习革命的主要支柱是：(1) 语言嵌入实体的分布式表征，(2) 由于嵌入而产生的语义泛化，(3) 自然语言的大跨度深序列建模，(4) 能够从低到高表示语言层次的分层网络，以及 (5) 解决许多联合 NLP 问题的端对端深度学习方法。

**1.1. 第一个浪潮：理性主义.** 在第一个浪潮中，NLP 的实验持续了很长一段时间，可以追溯到 20 世纪 50 年代。1950 年，阿兰·图灵提出了图灵测试，以评估计算机表现出与人类无法区分的智能行为的能力。这项测试是基于人类和计算机之间的自然语言对话，旨在生成类似人类的反应。1954 年，George-IBM 实验产出了能够将 60 多个俄语句子翻译成英语的 first 机器翻译系统。

这些方法是基于这样一种信念，即人类思维中的语言知识是由泛型继承提前进行的，而这种信念，在大约 1960 年至 1980 年代后期，占据了 NLP 的大部分研究中的主导地位。这些方法被称为理性主义方法 (Church 2007)。理性主义方法在 NLP 中的主导地位主要是由于诺姆·乔姆斯基 (Noam Chomsky) 关于先天语言结构的论点被广泛接受以及他对 N-grams 方法的批评 (Chomsky 1957)。理性主义者一般假设语言的关键部分在出生时就被硬连接到大脑中，作为人类遗传遗传的一部分，因此他们试图设计手工制作的规则，将知识和推理机制纳入智能 NLP 系统。直到 20 世纪 80 年代，最著名的成功的 NLP 系统，如为模拟 Rogerian psychotherapist 的 ELIZA 系统和为了规则化真实世界信息为规则本体的 MARGIE 系统，都是基于复杂的手写规则。

这一时期恰逢以专家知识工程为特点的早期智能的早期发展，即领域专家根据其所掌握的（非常狭窄的）应用领域的知识设计计算机程序 (Nilsson 1982; Winston 1993)。专家们使

用符号逻辑规则设计了这些程序，这些规则基于对这些知识的仔细表征和工程。这些以知识为基础的智能系统往往通过检测“Head”或最重要的参数，并就每种特殊情况采取特定的解决办法，而这在解决狭义问题方面往往是有效的。这些“Head”参数由人类专家预先确定，使“tail”参数和案例不受影响。由于缺乏学习能力，他们有必要将解决方案推广到新的情况和领域。这一时期的典型方法是专家系统所提供的证据，这是一个模拟人类专家决策能力的计算机系统。这种系统旨在通过知识推理来解决复杂的问题（Nilsson 1982）。第一个专家系统建立于1970年代，然后在1980年代推广。使用的主要“算法”是以“if-then-else”为形式的推断规则（Jackson 1998）。这些智能系统的主要优点是其在进行逻辑推理方面（有限）能力的透明度和可解释性。像NLP系统，如ELIZA和MARGIE，一般专家系统在早期使用手工制作的专家知识，这往往是有效的狭隘的问题，虽然推理无法处理不确定性，是普遍存在的实际应用。

同样，语音识别研究和系统设计，这又是另一个长期存在的NLP和反智能挑战，在这个理性主义时代，主要基于专家知识工程的范式，如elegantly analyzed in (Church and Mercer 1993)。在1970年代和1980年代初，专家系统的语音识别方法相当流行（Reddy 1976; Zue 1985）。然而，研究人员敏锐地认识到，缺乏从数据中学习和处理推理不确定性的能力，导致了接下来描述的第二波语音识别、NLP和对于文本的人工智能浪潮也走向失败。

**1.2. 第二波浪潮：经验主义。**第二波NLP浪潮的特点是利用语料库数据以及基于（浅层）机器学习、统计学等来利用这些数据（Manning and Schütze 1999）。由于许多自然语言的结构和理论都被贬低或抛弃，而倾向于数据驱动的方法，这个时代发展的主要方法被称为经验或务实的方法（Church and Mercer 1993; Church 2014）。NLP的一个主要会议甚至被命名为“自然语言处理的经验方法（Empirical Methods in Natural Language Processing）（EMNLP）”，最直接地反映了NLP研究人员在那个时代对经验方法的强烈积极情绪。

与理性主义方法相反，经验方法认为人类的思维只是从关联、模式识别和泛化的常规操作开始。丰富的感官输入需要使大脑学习自然语言的详细结构。经验主义盛行于1920年至1960年间，自1990年以来一直在兴起。NLP的早期经验方法主要是开发生成模型，如隐马尔可夫模型（HMM）（Baum and Petrie 1966），IBM翻译模型（Brown et al. 1993），和head-driven parsing模型（Collins 1997），以发现大型语料库的规律性。自1990年代后期以来，在各种NLP任务中，歧视性模式已成为事实上的做法。NLP的典型判别模型和方法包括最大熵模型（ratnaparkhi 1997）、支持向量机（Vapnik 1998）、条件随机场（Lafferty et al. 2001）、最大相互信息和最小区分器错误（He et al. 2008）还有感知器（Collins 2002）。

在这种经验主义时代中，NLP与同样的智能方法如语音识别和计算机视觉是平行的。这是在明确的证据表明，学习和感知能力对复杂的智能系统至关重要，但在前一波流行的专家系统中却不存在。例如，当DARPA开始对自动驾驶提出重大挑战时，大多数车辆随后依赖于基

于知识的智能智能。正如语音识别和 NLP 一样，自动驾驶和计算机视觉研究人员意识到基于知识的范式的局限性，因为机器学习需要进行不确定性处理和泛化能力。

在第二波浪潮中，NLP 的经验主义和语音识别是基于数据密集型机器学习的，我们现在称之为“shallow”，因为在下一节中描述的第三波浪潮中，数据的多层或“deep”表征通常缺乏抽象结构。在机器学习中，在第一次浪潮中，研究人员不需要考虑构造精确规则，为知识为基础的 NLP 和语音系统。相反，他们把重点放在统计模型 (Bishop 2006; Murphy 2012) 或作为一个基本引擎的简单的神经网络 (Bishop 1995)。然后，他们使用足够的训练数据进行自动学习或“tune (调整)”系统的参数，使它们能够处理不确定性，并尝试从一个条件泛化到另一个条件，从一个领域泛化到另一个领域。机器学习的关键算法和方法包括 EM (期望最大化)、贝叶斯网络、支持向量机、决策树以及神经网络的反向传播算法。

一般来说，基于机器学习的 NLP、语音和其他智能系统的性能比早期的基于知识的智能系统要好得多。成功的例子包括语音识别 (Jelinek 1998)，面部识别 (Viola and Jones 2004)，实体识别 (Fei-Fei and Perona 2005)，手写字体识别 (Plamondon and Srihari 2000)，以及机器翻译 (Och 2003)。

在语音识别方面，从 20 世纪 80 年代初到 2010 年前后近 30 年，利用基于 HMM 与高斯混合模型相结合的统计生成模型，以及其推广的各种版本 (Baker et al. 2009a, b; Deng and O’Shaughnessy 2003; Rabiner and Juang 1993) 的统计生成模式。泛化 HMM 的许多版本都是基于统计和神经网络的隐动态模型 (Deng 1998; Bridle et al. 1998; Deng and Yu 2007)。前者采用 EM 和 switching extended Kalman filter 算法学习模型参数 (Ma and Deng 2004; Lee et al. 2004)，后者采用反向传播 (Picone et al. 1999)，两者都广泛地利用多个潜在层表示法进行语音分析的生成过程。将这种“深度”生成过程转化为端到端过程的对应方案，导致了深度学习的工业化成功 (Deng et al. 2010, 2013; Hinton et al. 2012)，从而形成了第三波浪潮的驱动力。

**1.3. 第三波浪潮：深度学习.** 在第二波浪潮中开发的 NLP 系统，包括语音识别、语言理解和机器翻译，表现得比在第一波浪潮时更好，鲁棒性更高，但它们远远没有达到人的水平，而这留下了很多需求。除了少数例外，NLP 的（浅层）机器学习模型通常没有足够的容量来吸收大量的训练数据。此外，学习算法、方法和基础设施也都不够强大。所有这一切都在几年前发生了变化，而这导致了第三波 NLP 浪潮，这股浪潮是由深层机器学习或深度学习的新范式推动的 (Bengio 2009; Deng and Yu 2014; LeCun et al. 2015; Goodfellow et al. 2016)。

深度学习起源于人工神经网络，它可以被看作是受生物神经系统启发的细胞类型的级联模型。随着反向传播算法的出现 (Rumelhart et al. 1986)，90 年代对深度神经网络的训练引起了广泛关注。在没有大量训练数据和没有适当的设计和学习范式的情况下，在神经网络训练过程中，学习信号随着层数（或更严格的信用分配深度）在层层传播时呈指数形式消失，使

得调整深层神经网络特别是递归的版本的连接权重变得异常艰难。Hinton 等人 (2006) 克服了这个问题，使用无人监督的预训练模型来进行学习有用的特征探测器。然后，通过监督学习进一步训练网络，对标记数据进行分类。因此，可以学习使用低维表征的方式来学习高维的表征的分布。这项开创性的工作标志着神经网络的复兴。此后提出和发展了各种网络结构，包括 Deep Belief 网络 (Hinton et al. 2006)、堆积自编码器 (Vincent et al. 2010)、深层玻尔兹曼机 (Hinton and Salakhutdinov 2012)、深度卷积神经网络 (Krizhevsky et al. 2012)，深层堆积网络 (Deng et al. 2012)，和深层 Q-networks (Mnih et al. 2015)。深度学习自 2010 年以来已成功地应用于实际智能领域的实际任务，包括语音识别 (Yu et al. 2010; Hinton et al. 2012)，图像识别 (Krizhevsky et al. 2012; He et al. 2016)，以及 NLP 绝大多数领域。

其中由于微软公司在工业化上的成功，以及愈来愈高的准确率等迹象，这些 2010-2011 年语音识别的惊人成功预示着 NLP 的第三波浪潮和人工智能的到来。随着深度学习在语音识别方面取得成功，计算机视觉 (Krizhevsky et al. 2012) 和机器翻译 (Bahdanau et al. 2015) 被类似的深度学习范式所取代。特别是，虽然 Bengio 等人在 2001 的工作，在 2011 年就开发了强大的神经词嵌入技术 (Bengio et al. 2001)，但由于大数据的可用性和更快的计算，它直到 10 多年后才被证明在一个大规模和实际有用规模上才能够实际有用 (Mikolov et al. 2013)。此外，许多其他现实世界的 NLP 应用，如图像字幕 (Karpathy and Fei-Fei 2015; Fang et al. 2015; Gan et al. 2017)，视觉问题回答 (Fei-Fei and Perona 2016)，语音理解系统 (Mesnil et al. 2013)，网络搜索 (Huang et al. 2013b) 和推荐系统由于深度学习而取得成功，此外还有许多非 NLP 任务，包括药物发现和药理学、客户关系管理、推荐系统、手势识别、医学信息、广告投放、医学图像分析、机器人、自动驾驶车辆、纸板和电子游戏 (例如 Atari, Go, Poker, and the latest, DOTA2) 等。详情请参阅维基上的深度学习领域。

在更多基于文本的应用领域中，机器翻译可能受到深度学习的影响最大。从 NLP 第二波浪潮中发展起来的浅层——统计机器翻译开始看起的话，目前在实际应用中最好的机器翻译系统是基于深神经网络的。例如，谷歌在 2016 年 9 月宣布了其转向神经机器翻译的阶段，两个月后微软也发布了类似的声明。Facebook 已经进行了大约一年的机器神经网络翻译的转换工作，到 2017 年 8 月它已经完全将这个系统部署成功。

在口语理解和对话系统领域，深度学习也正在产生巨大影响。目前流行的技术以多种方式维护和扩展了第二波时代浪潮中发展起来的统计方法。与经验（浅层）机器学习方法一样，深度学习也是基于数据密集型方法，以降低手工制作规则的成本，对噪声环境下的语音识别错误和语言理解错误具有很强的鲁棒性，并利用决策过程和强化学习的力量来设计对话策略，例如 (Gasic et al. 2017; Dhingra et al. 2017)。与早期的方法相比，深度神经网络模型和表征方法更强大，它们使端到端学习成为可能。然而，深度学习也没有解决可解释性和领域泛化问题。

将深度学习应用于 NLP 问题方面的最近的两个重要技术突破是序列到序列学习 (Sutskevar et al. 2014) 和注意力机制建模 (Bahdanau et al. 2015)，以及最近的 BERT 模型 (Jacob

el al.2018)。序列到序列学习引入了一个强大的学习范式，即使用递归神经网络以端到端的方式进行编码和解码。注意力机制建模最初是为了克服编码一个长序列的难度而开发的，后来的持续发展又扩展了它的能力，提供了两个任意序列的高度可塑对齐能力，而其两个可以同时学习神经网络参数。而 BERT 则是实现了双向建模获取以得到更好的语言表征能力。序列到序列学习和注意力机制的关键概念在基于统计学习和词局部表征的最佳系统上提高了基于分布式单词嵌入的神经机器翻译的性能，而 BERT 更重要的意义是双向获取同一文段的高维意义。在这一成功之后，这些概念也被成功地应用到许多其他与 NLP 相关的任务中，如图像字幕 (Karpathy and Fei-Fei 2015; Devlin et al. 2015)、语音识别 (Chorowski et al. 2015)、一次性学习、句法分析、唇读、文本理解、摘要以及问答系统等。撇开他们巨大的经验成功不谈，基于神经网络的深度学习模型往往比早期浪潮中的传统机器学习模型更简单、更容易设计。在许多应用中，在端到端的任务中，模型的所有部分都同时进行深度学习，从特征抽取到预测。导致神经网络模型相对简单的另一个因素是，相同的模型构建块（即不同类型的层）通常在许多不同的应用中使用。为多种任务使用相同的构建块，这种方法使得模型更容易迁移到其它任务和数据上。此外，谷歌等公司还开发了软件工具包，以便更快、更有效地实现这些模型。由于以上这些原因，神经网络在数据量大而且基于云的方式上，是更常用的。

尽管深度学习在重塑语音、图像和视频的处理方面被证明是有效的，而且具有它的革命性，但在将深度学习与基于文本的 NLP 相结合方面的有效性并不那么明确，尽管它在一些实用的 NLP 任务中取得了经验上的成功。在语音、图像和视频处理中，深度学习通过直接从原始数据学习规律来解决语义差距问题。然而，在 NLP 中，人们提出了更强的理论和结构化模型，即语音、语法和语义，来提取理解和生成自然语言的基本机制，这些机制与神经网络不那么容易兼容。与语音、图像和视频信号相比，从文本数据中学习的神经表征可以对自然语言提供同样直接的见解，但是这个也不够直接。因此，将神经网络，特别是那些具有复杂层次结构的神经网络应用于 NLP，已成为 NLP 和深度学习社区中最活跃的领域，近年来取得了非常显著的进展 (Deng 2016; Manning and Socher 2017; Jacob et al.2018)。

2. 16.1 如何理解序列到序列模型？
3. 16.2 序列到序列模型有什么限制吗？
4. 16.3 如果不采用序列到序列模型，可以考虑用其它模型方法吗？
  5. 16.4 如何理解词向量？
  6. 16.5 词向量哪家好？
7. 16.6 解释一下注意力机制的原理？
8. 16.7 注意力机制是不是适用于所有场景呢？它的鲁棒性如何？
9. 16.8 怎么将原有的模型加上注意力机制呢？
10. 16.9 通俗地解释一下词法分析是什么？有什么应用场景？
11. 16.10 深度学习中的词法分析有哪些常见模型呢？
12. 16.11 通俗地解释一下知识图谱是什么？有什么应用场景？
  13. 16.12 深度学习中的知识图谱有哪些常见模型呢？
  14. 16.13 深度学习中的机器翻译有哪些常见模型呢？
15. 16.14 机器翻译的通俗实现以及部署过程是怎样的呢？
16. 16.15 通俗地解释一下文本情感分析是什么？常见的应用场景是？
  17. 16.16 最常用的情感分析模型是什么呢？如何快速部署呢？
18. 16.17 通俗地解释一下问答系统？它涵盖哪些领域？常见的应用场景是？
  19. 16.18 常见的问答系统模型是什么？如何快速部署呢？
  20. 16.19 图像文字生成是什么？它的技术原理是什么？
21. 16.20 常见的图像文字生成模型是什么？
22. 16.21 NLP 的无监督学习发展动态是怎样的？有哪些领域在尝试无监督学习？
23. 16.22 NLP 和强化学习的结合方式是怎样的？有哪些方向在尝试强化学习？
  24. 16.23 NLP 和元学习？元学习如何能够和 NLP 结合起来？
25. 16.24 能说一下各自领域最常用且常见的基准模型有哪些吗？





## 模型压缩及移动端部署

深度神经网络在人工智能的应用中，包括语音识别、计算机视觉、自然语言处理等各方面，在取得巨大成功的同时，这些深度神经网络需要巨大的计算开销和内存开销，严重阻碍了资源受限下的使用。本章总结了模型压缩、加速一般原理和方法，以及在移动端如何部署。

**0.1. 模型压缩理解.** 模型压缩是指利用数据集对已经训练好的深度模型进行精简，进而得到一个轻量且准确率相当的网络，压缩后的网络具有更小的结构和更少的参数，可以有效降低计算和存储开销，便于部署在受限的硬件环境中。

**0.2. 为什么需要模型压缩和加速？** (1) 随着 AI 技术的飞速发展，越来越多的公司希望在自己的移动端产品中注入 AI 能力。

(2) 对于在线学习和增量学习等实时应用而言，如何减少含有大量层级及结点的大型神经网络所需要的内存和计算量显得极为重要。

(3) 模型的参数在一定程度上能够表达其复杂性，相关研究表明，并不是所有的参数都在模型中发挥作用，部分参数作用有限、表达冗余，甚至会降低模型的性能。

(4) 复杂的模型固然具有更好的性能，但是高额的存储空间、计算资源消耗是使其难以有效的应用在各硬件平台上的重要原因。

(5) 智能设备的流行提供了内存、CPU、能耗和宽带等资源，使得深度学习模型部署在智能移动设备上变得可行。

(6) 高效的深度学习方法可以有效的帮助嵌入式设备、分布式系统完成复杂工作，在移动端部署深度学习有很重要的意义。

必要性	首先是资源受限，其次在许多网络结构中，如 VGG-16 网络，参数数量 1 亿 3 千多万，占用 500MB 空间，需要进行 309 亿次浮点运算才能完成一次图像识别任务。
必要性	首先是资源受限，其次在许多网络结构中，如 VGG-16 网络，参数数量 1 亿 3 千多万，占用 500MB 空间，需要进行 309 亿次浮点运算才能完成一次图像识别任务。
可行性	模型的参数在一定程度上能够表达其复杂性，相关研究表明，并不是所有的参数都在模型中发挥作用，部分参数作用有限、表达冗余，甚至会降低模型的性能。论文提出，很多的深度神经网络仅仅使用很少一部分（5%）权值就足以预测剩余的权值。该论文还提出这些剩下的权值甚至可以直接不用被学习。也就是说，仅仅训练一小部分原来的权值参数就有可能达到和原来网络相近甚至超过原来网络的性能（可以看作一种正则化）。
最终目的	最大程度的减小模型复杂度，减少模型存储需要的空间，也致力于加速模型的训练和推测

### 0.3. 17.3 模型压缩的必要性及可行性.

**0.4. 17.4 目前有哪些深度学习模型压缩方法？** 目前深度学习模型压缩方法主要分为更精细化模型设计、模型裁剪、核的稀疏化、量化、低秩分解、迁移学习等方法，而这些方法又可分为前端压缩和后端压缩。

对比项目	前端压缩	后端压缩
含义	不会改变原始网络结构的压缩技术	会大幅度上改变原始网络结构的压缩技术
主要方法	知识蒸馏、紧凑的模型结构设计、滤波器层面的剪枝	低秩近似、未加限制的剪枝、参数量化、二值化
实现难度	较简单	较难
是否可逆	可逆	不可逆
成熟应用	剪枝	低秩近似、参数量化
待发展应用	知识蒸馏	二值网络

#### 0.4.1. 17.4.1 前端压缩和后端压缩对比.

**0.4.2. 17.4.2 网络剪枝.** 深度学习模型因其稀疏性，可以被裁剪为结构精简的网络模型，具体包括结构性剪枝与非结构性剪枝。

#### 0.4.5. 17.4.5 前端压缩. (1) 知识蒸馏

一个复杂模型可由多个简单模型或者强约束条件训练得到。复杂模型特点是性能好，但其参数量大，计算效率低。小模型特点是计算效率高，但是其性能较差。知识蒸馏是让复杂模型学习到的知识迁移到小模型当中，使其保持其快速的计算速度前提下，同时拥有复杂模型的性能，达到模型压缩的目的。(2) 紧凑的模型结构设计

紧凑的模型结构设计主要是对神经网络卷积的方式进行改进，比如使用两个 3x3 的卷积替换一个 5x5 的卷积、使用深度可分离卷积等方式降低计算参数量。目前很多网络基于模块化设计思想，在深度和宽度两个维度上都很大，导致参数冗余。因此有很多关于模型设计的研究，如 SqueezeNet、MobileNet 等，使用更加细致、高效的模型设计，能够很大程度的减少模型尺寸，并且也具有不错的性能。

#### (3) 滤波器层面的剪枝

滤波器层面的剪枝属于非结构化剪枝，主要是对较小的权重矩阵整个剔除，然后对整个神经网络进行微调。此方式由于剪枝过于粗放，容易导致精度损失较大，而且部分权重矩阵中会存留一些较小的权重造成冗余，剪枝不彻底。具体操作是在训练时使用稀疏约束（加入权重的稀疏正则项，引导模型的大部分权重趋向于 0）。完成训练后，剪去滤波器上的这些 0。

优点是简单，缺点是剪得不干净，非结构化剪枝会增加内存访问成本。

#### 0.4.6. 17.4.6 后端压缩. (1) 低秩近似

在卷积神经网络中，卷积运算都是以矩阵相乘的方式进行。对于复杂网络，权重矩阵往往非常大，非常消耗存储和计算资源。低秩近似就是用若干个低秩矩阵组合重构大的权重矩阵，以此降低存储和计算资源消耗。

事项	特点
优点	可以降低存储和计算消耗；一般可以压缩 2-3 倍；精度几乎没有损失；
缺点	模型越复杂，权重矩阵越大，利用低秩近似重构参数矩阵不能保证模型的性能；超参数的数量随着网络层数的增加呈线性变化趋势，例如中间层的特征通道数等等。随着模型复杂度的提升，搜索空间急剧增大。

事项	特点
优点	可以降低存储和计算消耗；一般可以压缩 2-3 倍；精度几乎没有损失；
缺点	模型越复杂，权重矩阵越大，利用低秩近似重构参数矩阵不能保证模型的性能；超参数的数量随着网络层数的增加呈线性变化趋势，例如中间层的特征通道数等等。随着模型复杂度的提升，搜索空间急剧增大。

#### (2) 未加限制的剪枝

完成训练后，不加限制地剪去那些冗余参数。

事项	特点
优点	保持模型性能不损失的情况下，减少参数量 9-11 倍；剔除不重要的权重，可以加快计算速度，同时也可以提高
缺点	极度依赖专门的运行库和特殊的运行平台，不具有通用性；压缩率过大时，破坏性能；

事项	特点
优点	保持模型性能不损失的情况下，减少参数量 9-11 倍；剔除不重要的权重，可以加快计算速度，同时也可以提高
缺点	极度依赖专门的运行库和特殊的运行平台，不具有通用性；压缩率过大时，破坏性能；

#### (3) 参数量化

神经网络的参数类型一般是 32 位浮点型，使用较小的精度代替 32 位所表示的精度。或者是将多个权重映射到同一数值，权重共享。**量化其实是一种权值共享的策略**。量化后的权值张量是一个高度稀疏的有很多共享权值的矩阵，对非零参数，我们还可以进行定点压缩，以获得更高的压缩率。

事项	特点
优点	模型性能损失很小，大小减少 8-16 倍；
缺点	压缩率大时，性能显著下降；依赖专门的运行库，通用性较差；
举例	二值化网络：XNORnet [13], ABCnet with Multiple Binary Bases [14], Bin-net with High-Order Residual Quantization [15], Bi-Real Net [16]；三值化网络：Ternary weight networks [17], Trained Ternary Quantization [18]；

W1-A8 或 W2-A8 量化：Learning Symmetric Quantization [19]；INT8 量化：TensorFlow-lite [20]，TensorRT [21]；其他（非线性）：Intel INQ [22]，log-net，CNNPack [23] 等；原文：[https://blog.csdn.net/baidu\\_31437863/article/details/84474847](https://blog.csdn.net/baidu_31437863/article/details/84474847) || 总结 | 最为典型就是二值网络、XNOR 网络等。其主要原理就是采用 1bit 对网络的输入、权重、响应进行编码。减少模型大小的同时，原始网络的卷积操作可以被 bit-wise 运算代替，极大提升了模型的速度。但是，如果原始网络结果不够复杂（模型描述能力），由于二值网络会较大程度降低模型的表达能力。因此现阶段有相关的论文开始研究 n-bit 编码方式成为 n 值网络或者多值网络或者变 bit、组合 bit 量化来克服二值网络表达能力不足的缺点。|

#### (4) 二值网络

相对量化更为极致，对于 32bit 浮点型数用 1bit 二进制数-1 或者 1 表示，可大大减小模型尺寸。

事项	特点
优点	网络体积小，运算速度快，有时可避免部分网络的 overfitting
缺点	二值神经网络损失的信息相对于浮点精度是非常大；粗糙的二值化近似导致训练时模型收敛速度非常慢

#### (5) 三值网络

(出自《深度神经网络压缩与加速综述》)

0.4.8. 17.4.7 总体压缩效果评价指标有哪些?. 网络压缩评价指标包括运行效率、参数压缩率、准确率. 与基准模型比较衡量性能提升时, 可以使用提升倍数 (speedup) 或提升比例 (ratio)。

评价指标      特点

准确率	目前, 大部分研究工作均会测量 Top-1 准确率, 只有在 ImageNet 这类大型数据集上才会只用 Top-5 准确率。
参数压缩率	统计网络中所有可训练的参数, 根据机器浮点精度转换为字节 (byte) 量纲, 通常保留两位有效数字以作对比。
运行效率	可以从网络所含浮点运算次数 (FLOP)、网络所含乘法运算次数 (MULTS) 或随机实验测得的网络平均运行时间来评估。

网络结构	TOP1 准确率/%	参数量/M	CPU 运行时间/ms
MobileNet V1	70.6	4.2	123
ShuffleNet(1.5)	69.0	2.9	-
ShuffleNet(x2)	70.9	4.4	-
MobileNet V2	71.7	3.4	80
MobileNet V2(1.4)	74.7	6.9	149

0.4.9. 17.4.8 几种轻量化网络结构对比.

0.4.10. 17.4.9 网络压缩未来研究方向有哪些?. 网络剪枝、网络精馏和网络分解都能在一定程度上实现网络压缩的目的. 回归到深度网络压缩的本质目的上, 即提取网络中的有用信息, 以下是一些值得研究和探寻的方向. (1) 权重参数对结果的影响度量. 深度网络的最终结果是由全部的权重参数共同作用形成的, 目前, 关于单个卷积核/卷积核权重的重要性的度量仍然是比较简单的方式, 尽管文献 [14] 中给出了更为细节的分析, 但是由于计算难度大, 并不实用. 因此, 如何通过更有效的方式来近似度量单个参数对模型的影响, 具有重要意义. (2) 学生网络结构的构造. 学生网络的结构构造目前仍然是由人工指定的, 然而, 不同的学生网络结构的训练难度不同, 最终能够达到的效果也有差异. 因此, 如何根据教师网络结构设计合理的网络结构在精简模型的条件下获取较高的模型性能, 是未来的一个研究重点. (3) 参数重建的硬件架构支持. 通过分解网络可以无损地获取压缩模型, 在一些对性能要求高的场景中是非常重要的. 然而, 参数的重建步骤会拖累预测阶段的时间开销, 如何通过硬件的支持加速这一重建过程, 将是未来的一个研究方向. (4) 任务或使用场景层面的压缩. 大型网络通常是在量级较大的数据集上训练完成的, 比如, 在 ImageNet 上训练的模型具备对 1 000 类物体的分类, 但在一些具体场景的应用中, 可能仅需要一个能识别其中几类的小型模型. 因此, 如何从一个全功能的网络压缩得到部分功能的子网络, 能够适应很多实际应用场景的需求. (5) 网络压缩效用的评价. 目前, 对各类深度网络压缩算法的评价是比较零碎的, 倾重于和被压缩的大型网络在参数量和运行时间上的比较. 未来的研究可以从提出更加泛化的压缩评价标准出发, 一方面平衡运行速度和模型大小.

以上 3 步即是 TensorRT 对于所部署的深度学习网络的优化和重构，根据其优化和重构策略，第一和第二步适用于所有的网络架构，但是第三步则对于含有 Inception 结构的神经网络加速效果最为明显。

Tips: 想更好地利用 TensorRT 加速网络推断，可在基础网络中多采用 Inception 模型结构，充分发挥 TensorRT 的优势。

条件	方法
若训练的网络模型包含 TensorRT 支持的操作	1、对于 Caffe 与 TensorFlow 训练的模型，若包含的操作都是 TensorRT 支持的，则可以直接由 TensorRT 优化重构 2、对于 MXnet, PyTorch 或其他框架训练的模型，若包含的操作都是 TensorRT 支持的，可以采用 TensorRT API 重建网络结构，并间接优化重构； 1、TensorFlow 模型可通过 <code>tf.contrib.tensorrt</code> 转换，其中不支持的操作会保留为 TensorFlow 计算节点； 2、不支持的操作可通过 Plugin API 实现自定义并添加进 TensorRT 计算图； 3、将深度网络划分为两个部分，一部分包含的操作都是 TensorRT 支持的，可以转换为 TensorRT 计算图。另一部则采用其他框架实现，如 MXnet 或 PyTorch；
若训练的网络模型包含 TensorRT 不支持的操作	

#### 0.5.3. 17.5.3 TensorRT 如何优化重构模型？

0.5.4. 17.5.4 TensorRT 加速效果如何？以下是在 TitanX (Pascal) 平台上，TensorRT 对大型分类网络的优化加速效果：

Network	Precision	Framework/GPU:Tit	Time(Batch=8,unit:ms)	Top1	Val.Acc.(ImageNet-1k)
				X	
Resnet50	fp32	TensorFlow	24.1		0.7374
Resnet50	fp32	MXnet	15.7		0.7374
Resnet50	fp32	TRT4.0.1	12.1		0.7374
Resnet50	int8	TRT4.0.1	6		0.7226
Resnet101	fp32	TensorFlow	36.7		0.7612
Resnet101	fp32	MXnet	25.8		0.7612

Network	Precision	Framework/GPU	Time(Batch=8,unit:ms)	Top1 Val.Acc.(ImageNet-1k)
Resnet101	fp32	TRT4.0.1	19.3	0.7612
Resnet101	int8	TRT4.0.1	9	0.7574

### 0.6. 17.6 影响神经网络速度的 4 个因素 (再稍微详细一点).

1. FLOPs(FLOPs 就是网络执行了多少 multiply-adds 操作);
2. MAC(内存访问成本);
3. 并行度 (如果网络并行度高, 速度明显提升);
4. 计算平台 (GPU, ARM)

**0.7. 17.7 压缩和加速方法如何选择 ?.** 1 ) 对于在线计算内存存储有限的应用场景或设备, 可以选择参数共享和参数剪枝方法, 特别是二值量化权值和激活、结构化剪枝. 其他方法虽然能够有效的压缩模型中的权值参数, 但无法减小计算中隐藏的内存大小 (如特征图). 2 ) 如果在应用中用到的紧性模型需要利用预训练模型, 那么参数剪枝、参数共享以及低秩分解将成为首要考虑的方法. 相反地, 若不需要借助预训练模型, 则可以考虑紧性滤波设计及知识蒸馏方法. 3 ) 若需要一次性端对端训练得到压缩与加速后模型, 可以利用基于紧性滤波设计的深度神经网络压缩与加速方法. 4 ) 一般情况下, 参数剪枝, 特别是非结构化剪枝, 能大大压缩模型大小, 且不容易丢失分类精度. 对于需要稳定的模型分类的应用, 非结构化剪枝成为首要选择. 5 ) 若采用的数据集较小时, 可以考虑知识蒸馏方法. 对于小样本的数据集, 学生网络能够很好地迁移教师模型的知识, 提高学生网络的判别性. 6 ) 主流的 5 个深度神经网络压缩与加速算法相互之间是正交的, 可以结合不同技术进行进一步的压缩与加速. 如: 韩松等人 [ 3 0 ] 结合了参数剪枝和参数共享; 温伟等人 [ 6 4 ] 以及 A l v a r e z 等人 [ 8 5 ] 结合了参数剪枝和低秩分解. 此外对于特定的应用场景, 如目标检测, 可以对卷积层和全连接层使用不同的压缩与加速技术分别处理.

参考《深度神经网络压缩与加速综述》

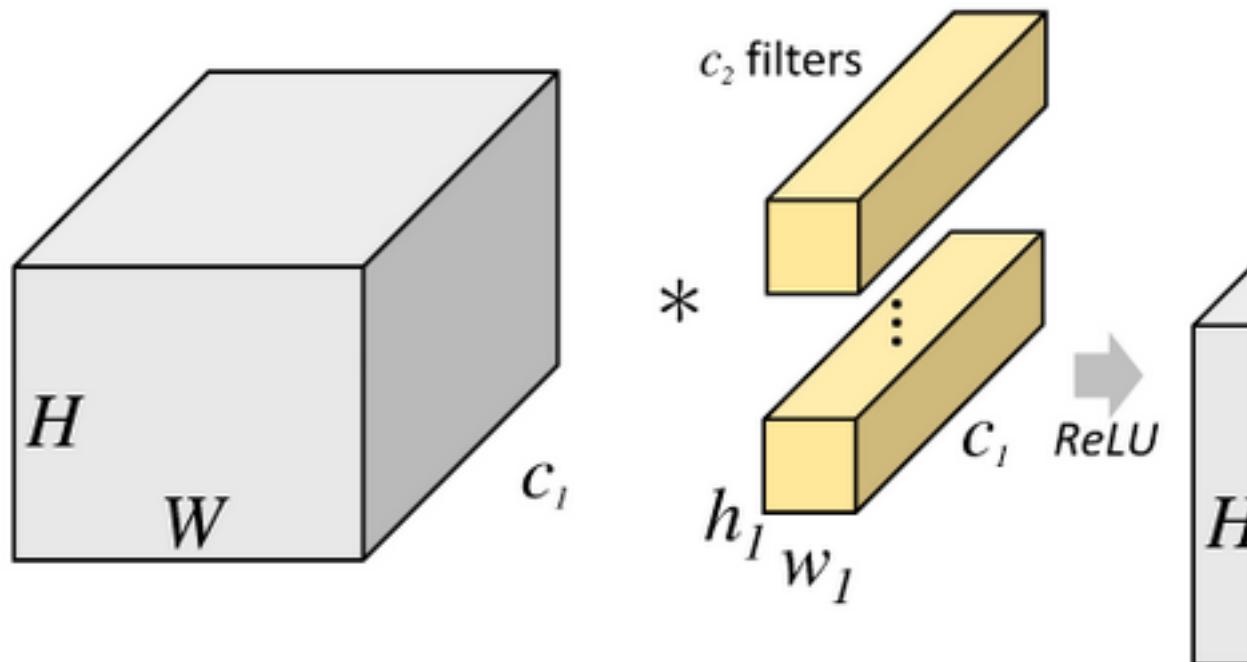
### 0.8. 17.8 改变网络结构设计为什么回实现模型压缩、加速 ?.

0.8.1. 17.8.1 *Group convolution*. Group convolution 最早出现在 AlexNet 中, 是为了解决单卡显存不够, 将网络部署到多卡上进行训练而提出. Group convolution 可以减少单个卷积  $1/g$  的参数量. 如何计算的呢?

假设

- 输入特征的维度为  $H * W * C_1$ ;
- 卷积核的维度为  $H_1 * W_1 * C_1$ , 共  $C_2$  个;
- 输出特征的维度为  $H_1 * W_1 * C_2$  .

传统卷积计算方式如下：

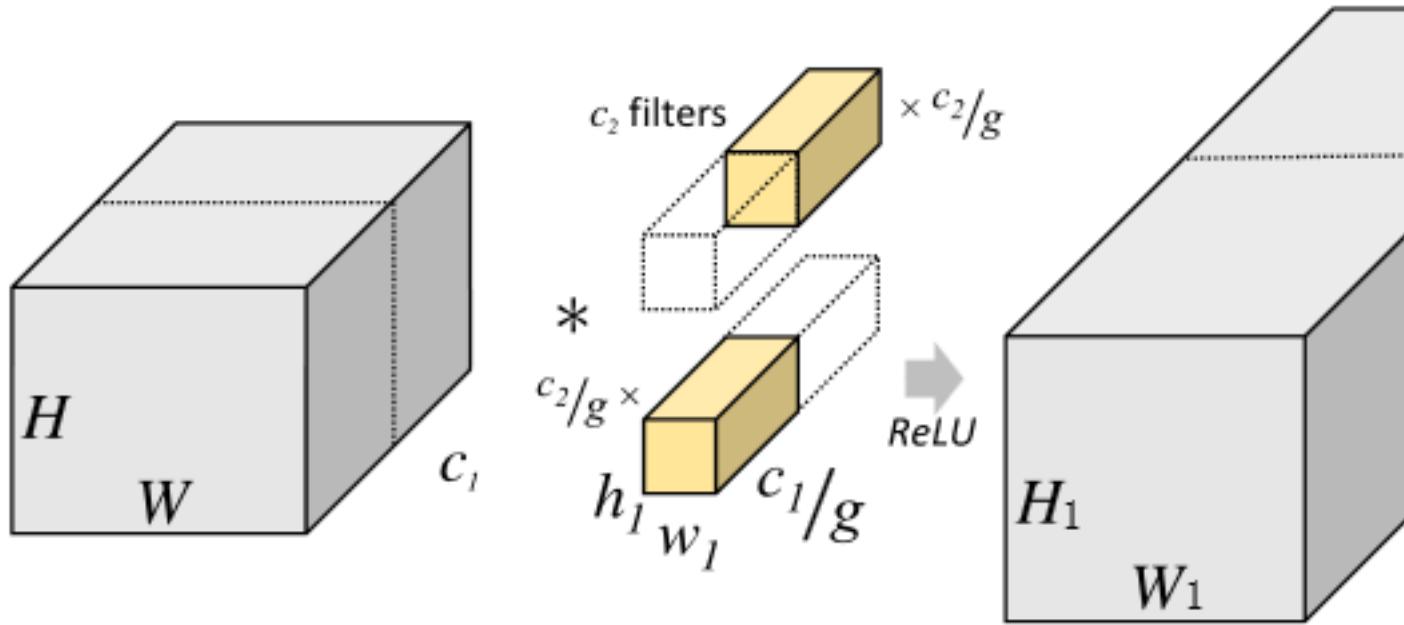


传统卷积运算量为：

$$A = H * W * h1 * w1 * c1 * c2$$

Group convolution 是将输入特征的维度  $c_1$  分成  $g$  份，每个 group 对应的 channel 数为  $c_1/g$ ，特征维度  $H * W * c_1/g$ ；，每个 group 对应的卷积核的维度也相应发生改变为  $h_1 * w_1 * c_1/g$ ，共  $c_2/g$  个；每个 group 相互独立运算，最后将结果叠加在一起。

Group convolution 计算方式如下：



Group convolution 运算量为：

$$B = H * W * h_1 * w_1 * c_1/g * c_2/g * g$$

Group 卷积相对于传统卷积的运算量：

$$\frac{B}{A} = \frac{H * W * h_1 * w_1 * c_1/g * c_2/g * g}{H * W * h_1 * w_1 * c_1 * c_2} = \frac{1}{g}$$

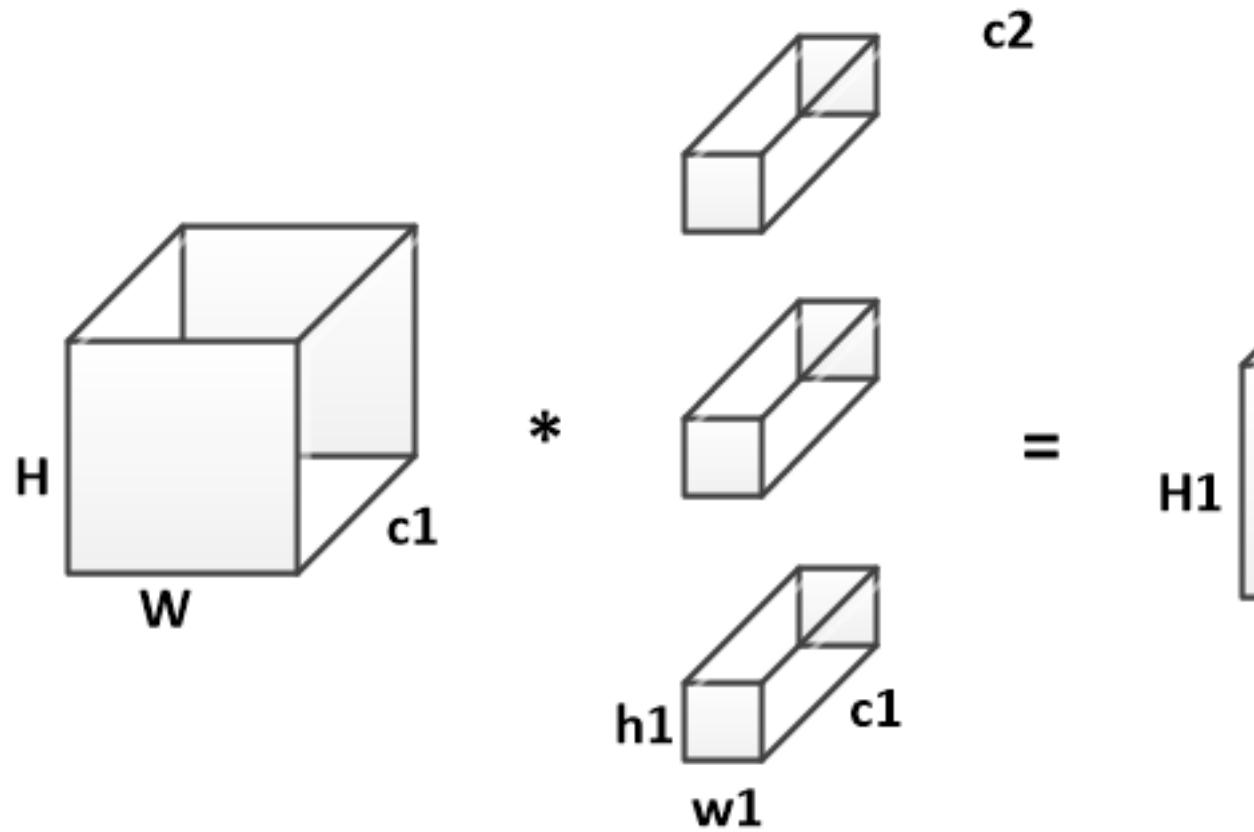
由此可知：group 卷积相对于传统卷积减少了  $1/g$  的参数量。

0.8.2. 17.8.2. *Depthwise separable convolution.* Depthwise separable convolution 是由 depthwise conv 和 pointwise conv 构成。

depthwise conv(DW) 有效减少参数数量并提升运算速度。但是由于每个 feature map 只被一个卷积核卷积，因此经过 DW 输出的 feature map 不能只包含输入特征图的全部信息，而且特征之间的信息不能进行交流，导致“信息流通不畅”。

pointwise conv(PW) 实现通道特征信息交流，解决 DW 卷积导致“信息流通不畅”的问题。假设输入特征的维度为  $H * W * c_1$ ；卷积核的维度为  $h_1 * w_1 * c_1$ ，共  $c_2$  个；输出特征的维度为  $H_1 * W_1 * c_2$ 。

传统卷积计算方式如下：

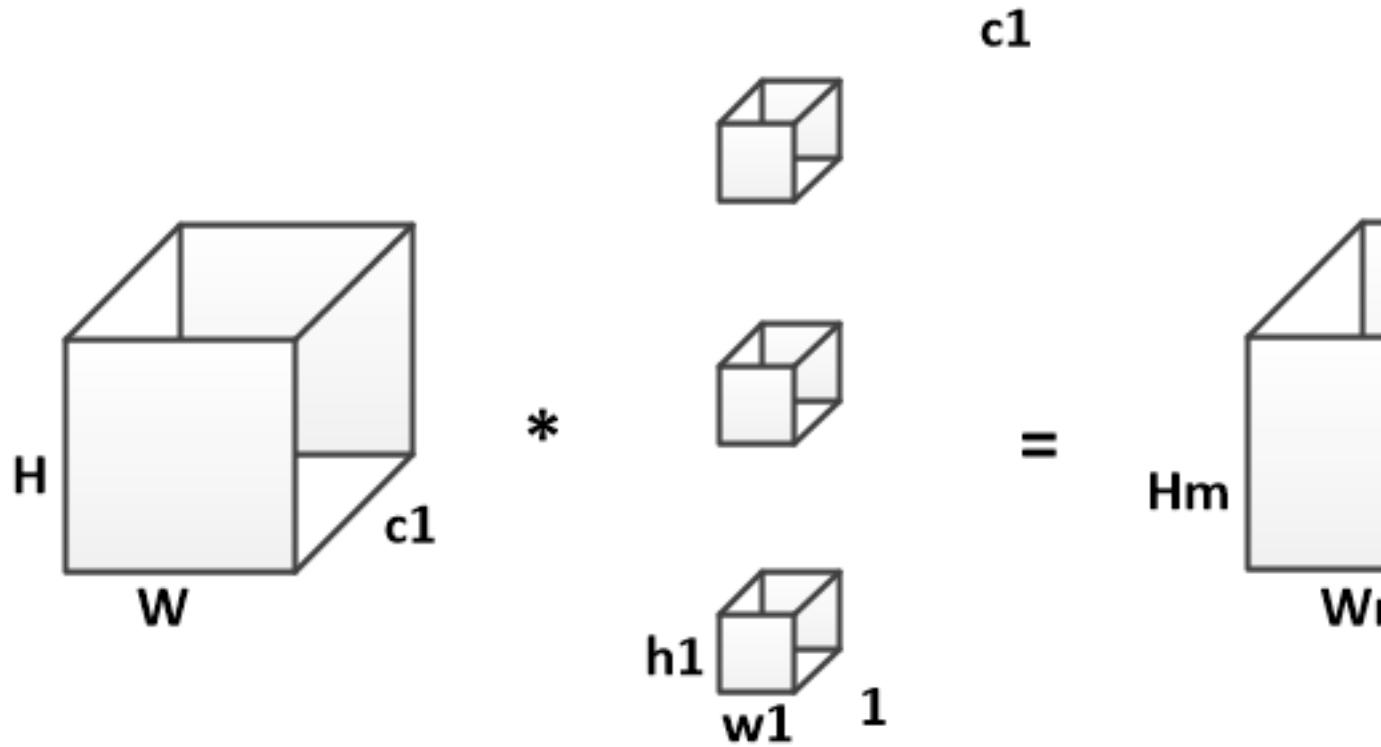


### Standard convolution

传统卷积运算量为：

$$A = H * W * h1 * w1 * c1 * c2$$

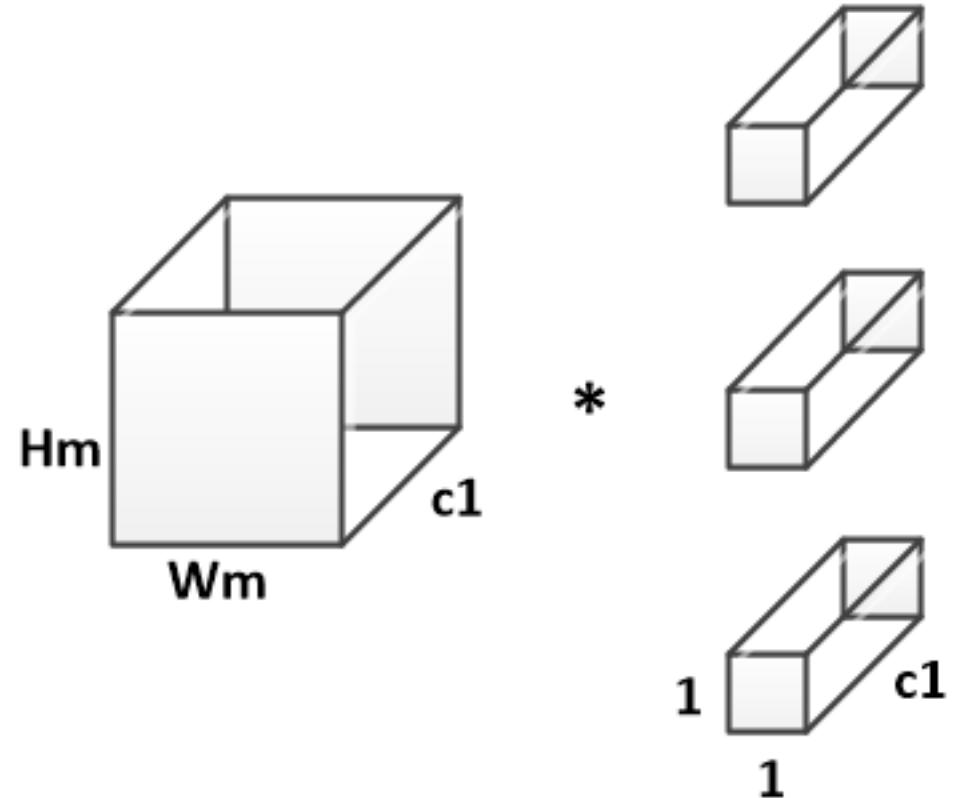
DW 卷积的计算方式如下：



## Depthwise convolution

DW 卷积运算量为：

$$B_D W = H * W * h1 * w1 * 1 * c1$$



## Pointwise conv

PW 卷积的计算方式如下:

$$B_{PW} = H_m * W_m * 1 * 1 * c_1 * c_2$$

Depthwise separable convolution 运算量为:

$$B = B_D W + B_{PW}$$

Depthwise separable convolution 相对于传统卷积的运算量: \$\$\frac{B}{A} = \frac{H \* W \* h\_1 \* w\_1 \* 1 \* c\_1 + H\_m \* W\_m \* 1 \* 1 \* c\_1 \* c\_2}{H \* W \* h\_1 \* w\_1 \* c\_1 \* c\_2} = \frac{1}{c\_2} + \frac{1}{h\_1 \* w\_1}\$\$ Depthwise separable convolution

0.8.3. 17.8.3 输入输出的 channel 相同时, MAC 最小. 卷积层的输入和输出特征通道数相等时 MAC 最小, 此时模型速度最快。

假设 feature map 的大小为  $h * w$ , 输入通道  $c_1$ , 输出通道  $c_2$ 。

已知:

$$FLOPs = B = h * w * c_1 * c_2 \Rightarrow c_1 * c_2 = \frac{B}{h * w}$$

$$MAC = h * w * (c_1 + c_2) + c_1 * c_2$$

$$\Rightarrow MAC \geq 2 * h * w \sqrt{\frac{B}{h * w}} + \frac{B}{h * w}$$

根据均值不等式得到  $(c_1 - c_2)^2 \geq 0$ , 等式成立的条件是  $c_1 = c_2$ , 也就是输入特征通道数和输出特征通道数相等时, 在给定 FLOPs 前提下, MAC 达到取值的下界。

**0.8.4. 17.8.4 减少组卷积的数量. 过多的 group 操作会增大 MAC, 从而使模型速度变慢**  
 由以上公式可知, group 卷积相比与传统的卷积可以降低计算量, 提高模型的效率; 如果在相同的 FLOPs 时, group 卷积为了满足 FLOPs 会是使用更多 channels, 可以提高模型的精度。但是随着 channel 数量的增加, 也会增加 MAC。

FLOPs:

$$B = \frac{h * w * c_1 * c_2}{g}$$

MAC:

$$MAC = h * w * (c_1 + c_2) + \frac{c_1 * c_2}{g}$$

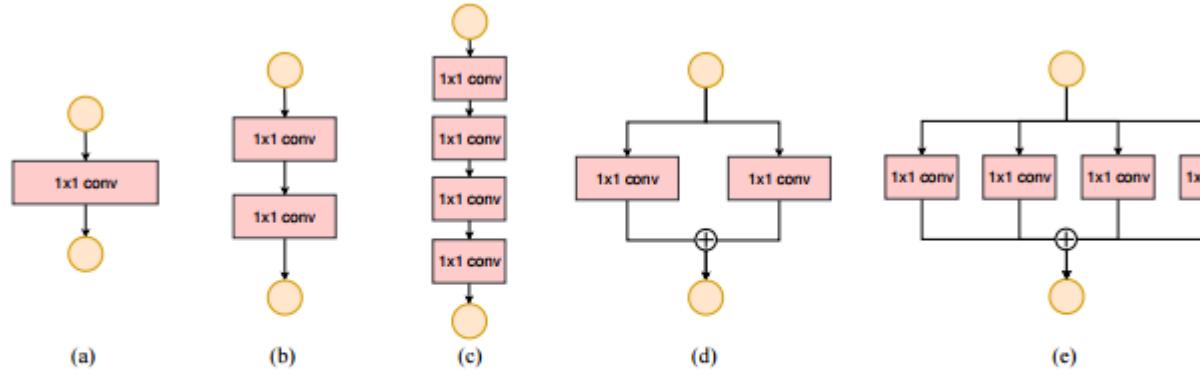
由 MAC, FLOPs 可知:

$$MAC = h * w * c_1 + \frac{B * g}{c_1} + \frac{B}{h * w}$$

当 FLOPs 固定 (B 不变) 时, g 越大, MAC 越大。

**0.8.5. 17.8.5 减少网络碎片化程度 (分支数量). 模型中分支数量越少, 模型速度越快**  
 此结论主要是由实验结果所得。

以下为网络分支数和各分支包含的卷积数目对神经网络速度的影响。



Appendix Fig. 1: Building blocks used in experiments for guideline 3. (a) 1-fragment. (b) 2-fragment-series. (c) 4-fragment-series. (d) 2-fragment-parallel. (e) 4-fragment-parallel.

实验中使用的基本网络结构，分别将它们重复 10 次，然后进行实验。实验结果如下：

由实验结果可知，随着网络分支数量的增加，神经网络的速度在降低。网络碎片化程度对 GPU 的影响效果明显，对 CPU 不明显，但是网络速度同样在降低。

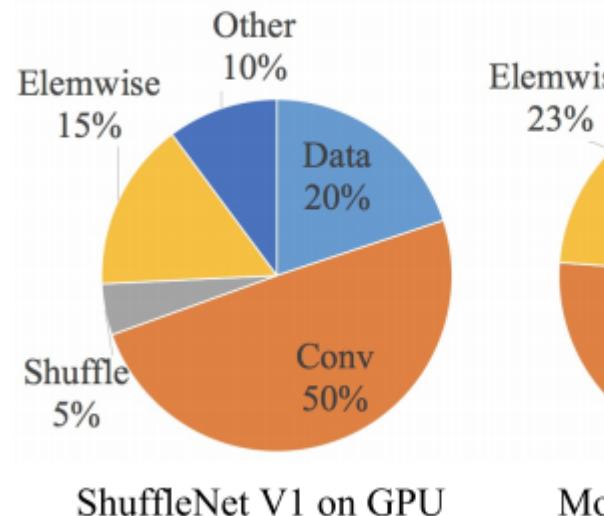
#### 0.8.6. 17.8.7 减少元素级操作。元素级操作所带来的时问消耗也不能忽视

ReLU，Tensor 相加，Bias 相加的操作，分离卷积（depthwise convolution）都定义为元素级操作。

FLOPs 大多数是对于卷积计算而言的，因为元素级操作的 FLOPs 相对要低很多。但是过的元

Table 3: Validation results for the proposed method. The validation error and the number of FLOPs is the same for all methods.

素级操作也会带来时间成本。ShuffleNet 作者对 ShuffleNet v1 和 MobileNet v2 的几种层操作的

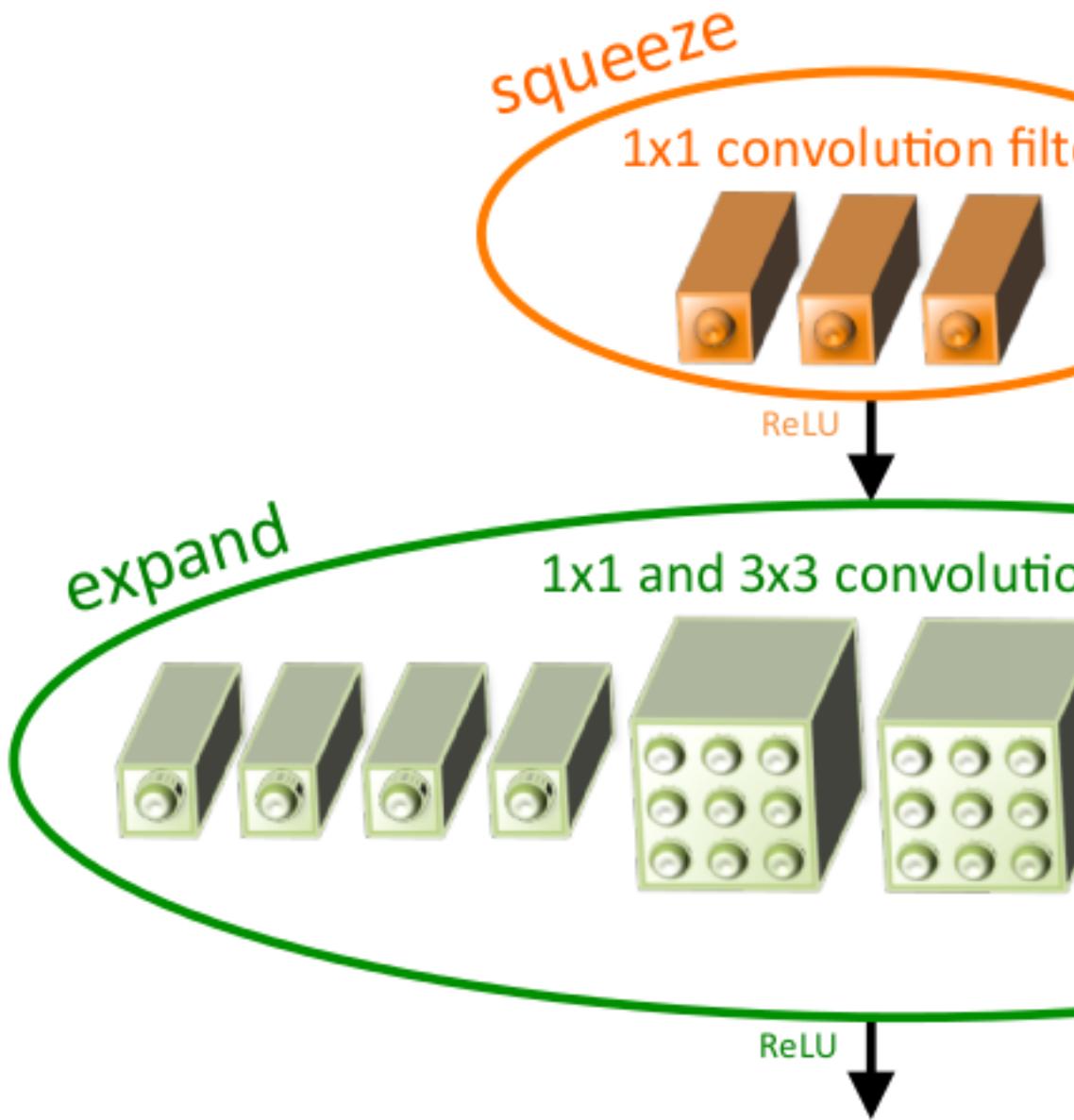


时间消耗做了分析，发现元素级操作对于网络速度的影响也很大。

### 0.9. 17.9 常用的轻量级网络有哪些？

0.9.1. 17.9.1 SqueezeNet. SqueezeNet 出自 F. N. Iandola, S. Han 等人发表的论文《SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5MB model size》，作者在保证精度不损失的同时，将原始 AlexNet 压缩至原来的 510 倍。

#### 1.1 设计思想在网络结构设计方面主要采取以下三种方式：  
 \* 用  $1 \times 1$  卷积核替换  $3 \times 3$  卷积  
 \* 理论上一个  $1 \times 1$  卷积核的参数是一个  $3 \times 3$  卷积核的  $1/9$ ，可以将模型尺寸压缩 9 倍。  
 \* 减小  $3 \times 3$  卷积的输入通道数  
 \* 根据上述公式，减少输入通道数不仅可以减少卷积的运算量，而且输入通道数与输出通道数相同时还可以减少 MAC。  
 \* 延迟降采样  
 \* 分辨率越大的输入能够提供更多特征的信息，有利于网络的训练判断，延迟降采样可以提高网络精度。  
 #### 1.2 网络架构 SqueezeNet 提出一种多分支结构——fire model，其中是由 Squeeze 层和 expand 层构成。Squeeze 层是由  $s_1$  个  $1 \times 1$  卷积组成，主要是通过  $1 \times 1$  的卷积降低 expand 层的输入维度；expand 层利用  $e_1$  个  $1 \times 1$  和  $e_3$  个  $3 \times 3$  卷积构成多分支结构提取输入特征，以此提高网络的精度（其中



$e1 = e3 = 4 * s1$ )。



CNN architecture	Computation cost
AlexNet	240M
AlexNet	8.5M
AlexNet	4.8M
AlexNet	4.8M
SqueezeNet (ours)	4.8M
SqueezeNet (ours)	4.8M
SqueezeNet (ours)	4.8M

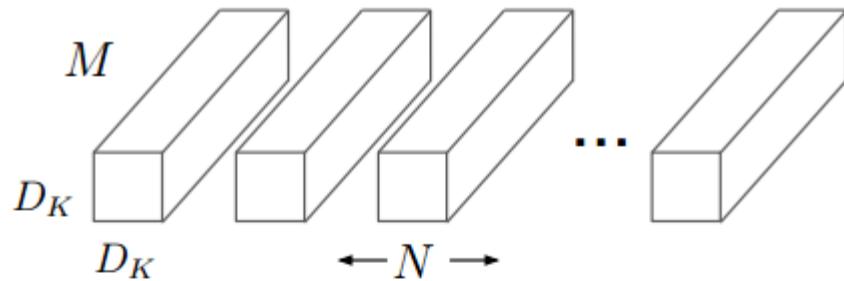
### 1.3 实验结果. 不同压缩方法在 ImageNet 上的对比实验结果

由实验结果可知, SqueezeNet 不仅保证了精度, 而且将原始 AlexNet 从 240M 压缩至 4.8M, 压缩 50 倍, 说明此轻量级网络设计是可行。

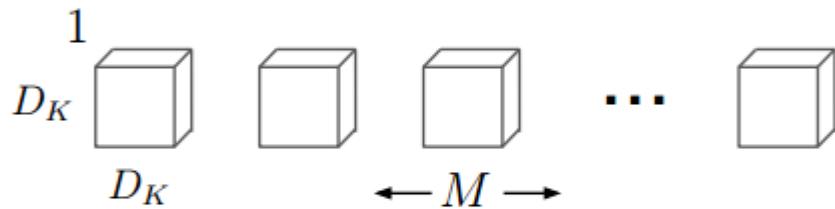
0.9.2. 17.9.2 MobileNet. MobileNet 是 Google 团队于 CVPR-2017 的论文《MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications》中针对手机等嵌入式设备提出的一种轻量级的深层神经网络, 该网络结构在 VGG 的基础上使用 DW+PW 的组合, 在保证不损失太大精度的同时, 降低模型参数量。#### 2.1 设计思想 \* 采用深度可分离卷积代替传统卷积 \* 采用 DW 卷积在减少参数数量的同时提升运算速度。但是由于每个 feature map 只被一个卷积核卷积, 因此经过 DW 输出的 feature map 不能只包含输入特征图的全部信息, 而且特征之间的信息不能进行交流, 导致“信息流通不畅”。\* 采用 PW 卷积实现通道特征信息交流, 解决 DW 卷积导致“信息流通不畅”的问题。\* 使用 stride=2 的卷积替换 pooling \* 直接在卷积时利用 stride=2 完成了下采样, 从而节省了需要再去用 pooling 再去进行一次下采样的时间, 可以提升运算速度。同时, 因为 pooling 之前需要一个 stride=1 的 conv, 而与 stride=2 conv 的计算量想比要高近 4 倍(个人理解)。#### 2.2 网络架构 \* DW

conv 和 PW conv MobileNet 的网络架构主要是由 DW conv 和 PW conv 组成，相比于传统卷积可以降低  $\frac{1}{N} + \frac{1}{Dk}$  倍的计算量。

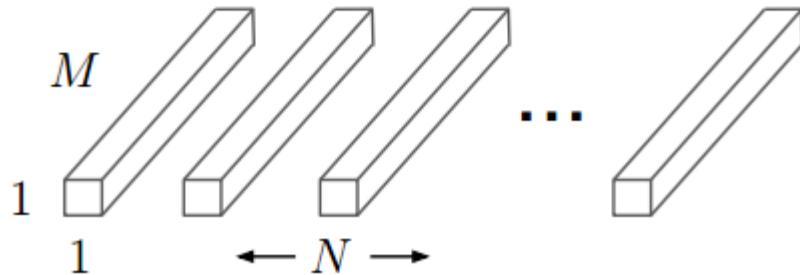
标准卷积与 DW conv 和 PW conv 如图所示：



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



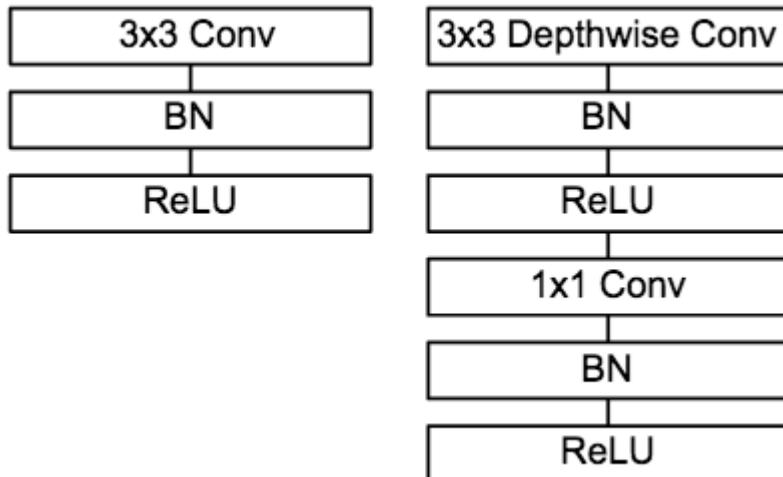
(c)  $1 \times 1$  Convolutional Filters called Pointwise Convolution in the context of MobileNet

<http://blog.csdn.net/u011995719>

深度可分离卷

$$\frac{D_K \cdot D_K \cdot M \cdot D_F}{D_K \cdot D_K \cdot \dots} = \frac{1}{N} + \frac{1}{D_K^2}$$

积与传统卷积运算量对比：



网络结构：

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Inp
Conv / s2	$3 \times 3 \times 3 \times 32$	224x224x3
Conv dw / s1	$3 \times 3 \times 32$ dw	112x112x32
Conv / s1	$1 \times 1 \times 32 \times 64$	112x112x64
Conv dw / s2	$3 \times 3 \times 64$ dw	112x112x64
Conv / s1	$1 \times 1 \times 64 \times 128$	112x112x128
Conv dw / s1	$3 \times 3 \times 128$ dw	112x112x128
Conv / s1	$1 \times 1 \times 128 \times 128$	112x112x128
Conv dw / s2	$3 \times 3 \times 128$ dw	112x112x128
Conv / s1	$1 \times 1 \times 128 \times 256$	112x112x256
Conv dw / s1	$3 \times 3 \times 256$ dw	112x112x256
Conv / s1	$1 \times 1 \times 256 \times 256$	112x112x256
Conv dw / s2	$3 \times 3 \times 256$ dw	112x112x256
Conv / s1	$1 \times 1 \times 256 \times 512$	112x112x512
$5 \times$ Conv dw / s1	$3 \times 3 \times 512$ dw	112x112x512
	$1 \times 1 \times 512 \times 512$	112x112x512
Conv dw / s2	$3 \times 3 \times 512$ dw	112x112x512
Conv / s1	$1 \times 1 \times 512 \times 1024$	70x70x1024
Conv dw / s2	$3 \times 3 \times 1024$ dw	70x70x1024
Conv / s1	$1 \times 1 \times 1024 \times 1024$	70x70x1024
Avg Pool / s1	Pool $7 \times 7$	70x70x1024
FC / s1	$1024 \times 1000$	1000
Softmax / s1	Classifier	1000

- MobileNets 的架构

**Table 4. Depthwise Separable vs Full Convolution MobileNet**

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
<b>Conv MobileNet</b>	<b>71.7%</b>	4866	29.3
<b>MobileNet</b>	<b>70.6%</b>	569	4.2

<http://blog.csdn.net>

### 2.3 实验结果.

由上表可知，使用相同的结构，深度可分离卷积虽然准确率降低 1%，但是参数量减少了 6/7。

0.9.3. 17.9.3 *MobileNet-v2*. MobileNet-V2 是 2018 年 1 月公开在 arXiv 上论文《Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation》，是对 MobileNet-V1 的改进，同样是一个轻量化卷积神经网络。#### 3.1 设计思想 \* 采用 Inverted residuals \* 为了保证网络可以提取更多的特征，在 residual block 中第一个 1\*1 Conv 和 3\*3 DW Conv 之前进行通道扩充 Linear bottlenecks \* 为了避免 Relu 对特征的破坏，在 residual block 的 Eltwise sum 之前的那个 1\*1 Conv 不再采用 Relu \* stride=2 的 conv 不使用 shot-cut, stride=1 的 conv 使用 shot-cut

### 3.2 网络架构.

- Inverted residuals

ResNet 中 Residuals block 先经过 1\*1 的 Conv layer，把 feature map 的通道数降下来，再经过 3\*3 Conv layer，最后经过一个 1\*1 的 Conv layer，将 feature map 通道数再“扩张”回去。即采用先压缩，后扩张的方式。而 inverted residuals 采用先扩张，后压缩的方式。

MobileNet 采用 DW conv 提取特征，由于 DW conv 本身提取的特征数就少，再经过传

统 residuals block 进行“压缩”，此时提取的特征数会更少，因此 inverted residuals 对

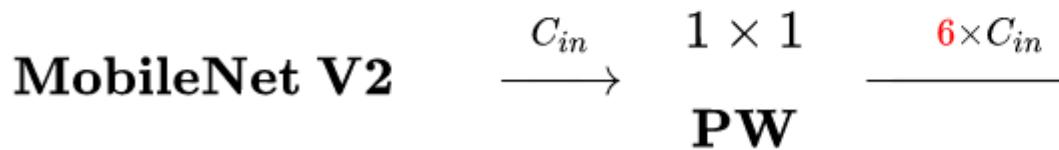
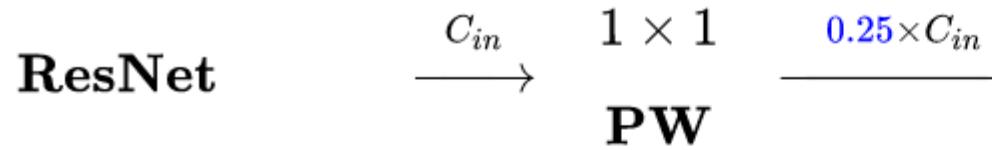


其进行“扩张”，保证网络可以提取更多的特征。

- Linear bottlenecks

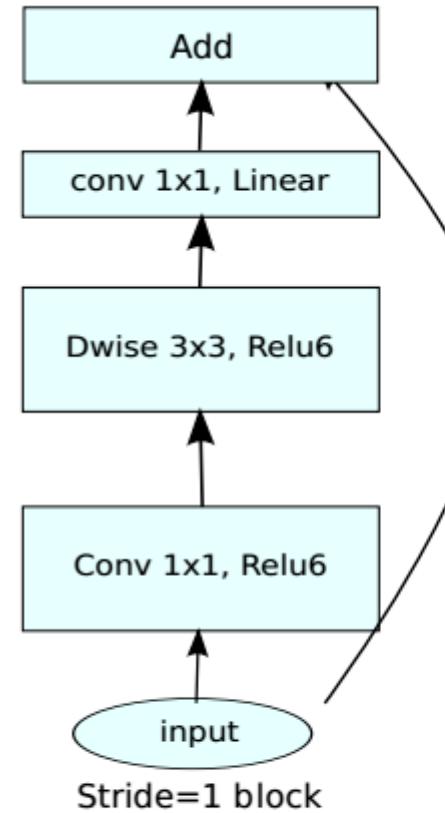
ReLU 激活函数会破坏特征。ReLU 对于负的输入，输出全为 0，而本来 DW conv 特征通

道已经被“压缩”，再经过 ReLu 的话，又会损失一部分特征。采用 Linear，目的是防止



Relu 破坏特征。

- shortcut



(d) Mob

stride=2 的 conv 不使用 shot-cut, stride=1 的 conv 使用 shot-cut

- 网络架构

Input	Operator	<i>t</i>	<i>c</i>	<i>n</i>	<i>s</i>
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$28^2 \times 64$	bottleneck	6	96	3	+
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times k$	conv2d 1x1	-	k	-	-

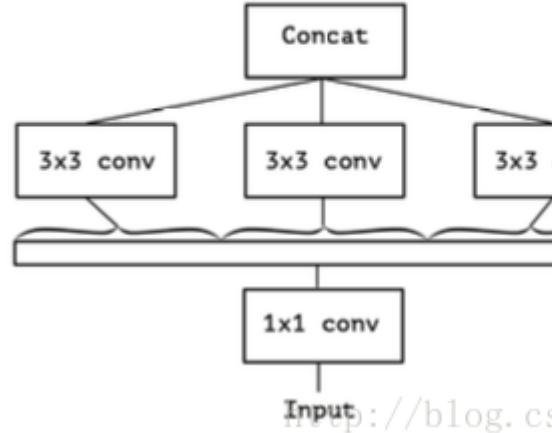
0.9.4. 17.9.4 Xception. Xception 是 Google 提出的, arXiv 的 V1 于 2016 年 10 月公开《Xception: Deep Learning with Depthwise Separable Convolutions》, Xception 是对 Inception v3 的另一种改进, 主要是采用 depthwise separable convolution 来替换原来 Inception v3 中的卷积操作。#### 4.1 设计思想 \* 采用 depthwise separable convolution 来替换原来 Inception v3 中的卷积操作

与原版的 Depth-wise convolution 有两个不同之处: \* 第一个: 原版 Depth-wise convolution, 先逐通道卷积, 再  $1 \times 1$  卷积; 而 Xception 是反过来, 先  $1 \times 1$  卷积, 再逐通道卷积; 第二个: 原版 Depth-wise convolution 的两个卷积之间是不带激活函数的, 而 Xception 在经过  $1 \times 1$  卷积之后会带上一个 Relu 的非线性激活函数;

4.2 网络架构. feature map 在空间和通道上具有一定的相关性, 通过 Inception 模块和非线性激活函数实现通道之间的解耦。增多  $3 \times 3$  的卷积的分支的数量, 使它与  $1 \times 1$  的卷积的输出通

道数相等，此时每个  $3 \times 3$  的卷积只作用与一个通道的特征图上，作者称之为“极致的 Inception”（Extream Inception）模块，这就是 Xception 的基本模块。

Figure 3. A strictly equivalent reformulation of the Inception module.



（Extream Inception）”模块，这就是 Xception 的基本模块。

0.9.5. 17.9.5 *ShuffleNet-v1*. ShuffleNet 是 Face++ 团队提出的，晚于 MobileNet 两个月在 arXiv 上公开《ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices》用于移动端前向部署的网络架构。ShuffleNet 基于 MobileNet 的 group 思想，将卷积操作限制到特定的输入通道。而与之不同的是，ShuffleNet 将输入的 group 进行打散，从而保证每个卷积核的感受野能够分散到不同 group 的输入中，增加了模型的学习能力。####  
 5.1 设计思想 \* 采用 group conv 减少大量参数 \* roup conv 与 DW conv 存在相同的“信息流通不畅”问题 \* 采用 channel shuffle 解决上述问题 \* MobileNet 中采用 PW conv 解决上述问题，SheffleNet 中采用 channel shuffle \* 采用 concat 替换 add 操作 \* avg pooling 和 DW conv( $s=2$ ) 会减小 feature map 的分辨率，采用 concat 增加通道数从而弥补分辨率减小而带来信息的损失

5.2 网络架构. MobileNet 中  $1 \times 1$  卷积的操作占据了约 95% 的计算量, 所以作者将  $1 \times 1$  也更

Table 2.

Type
Conv $1 \times 1$
Conv DW $3 \times$
Conv $3 \times 3$
Fully Connected

改为 group 卷积, 使得相比 MobileNet 的计算量大大减少。

group 卷积与 DW 存在同样使“通道信息交流不畅”的问题, MobileNet 中采用 PW conv 解决上述问题, SheffleNet 中采用 channel shuffle。



Figure 2: ShuffleNet Unit  
ShuffleNet unit with pointwise convolution with stride = 2.

ShuffleNet 的 shuffle 操作如图所示

avg pooling 和 DW conv( $s=2$ ) 会减小 feature map 的分辨率, 采用 concat 增加通道数从而弥补分辨率减小而带来信息的损失; 实验表明: 多多使用通道(提升通道的使用率), 有助于提高小模



Figure 1: Channel shuffle with two stacked convolution layers. a) two stacked convolution layers to the input channels within the group when GConv2 takes data from different groups using channel shuffle.

网络结构：

Layer	Output size	KSize	Stride	Repeat	$g = 1$
Image	$224 \times 224$				3
Conv1	$112 \times 112$	$3 \times 3$	2	1	24
MaxPool	$56 \times 56$	$3 \times 3$	2		
Stage2	$28 \times 28$		2	1	144
	$28 \times 28$		1	3	144
Stage3	$14 \times 14$		2	1	288
	$14 \times 14$		1	7	288
Stage4	$7 \times 7$		2	1	576
	$7 \times 7$		1	3	576
GlobalPool	$1 \times 1$	$7 \times 7$			
FC					1000
Complexity					143M

0.9.6. 17.9.6 *ShuffleNet-v2*. huffleNet-v2 是 Face++ 团队提出的《ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design》，旨在设计一个轻量级但是保证精度、速度的深度网络。#### 6.1 设计思想 \* 文中提出影响神经网络速度的 4 个因素：\* a. FLOPs(FLOPs 就是网络执行了多少 multiply-adds 操作) \* b. MAC(内存访问成本) \* c. 并行度 (如果网络并行度高，速度明显提升) \* d. 计算平台 (GPU, ARM) \* ShuffleNet-v2 提出了 4 点网络结构设计策略：\* G1. 输入输出的 channel 相同时，MAC 最小 \* G2. 过度的组卷积会增加 MAC \* G3. 网络碎片化会降低并行度 \* G4. 元素级运算不可忽视

6.2 网络结构. depthwise convolution 和瓶颈结构增加了 MAC，用了太多的 group，跨层连接中的 element-wise Add 操作也是可以优化的点。所以在 shuffleNet V2 中增加了几种新特性。

所谓的 channel split 其实就是将通道数一分为 2, 化成两分支来代替原先的分组卷积结构(G2), 并且每个分支中的卷积层都是保持输入输出通道数相同(G1), 其中一个分支不采取任何操作减少基本单元数(G3), 最后使用了 concat 代替原来的 element-wise add, 并且后面不加 ReLU 直接(G4), 再加入 channel shuffle 来增加通道之间的信息交流。对于下采样层, 在这一层中对通道数进行翻倍。在网络结构的最后, 即平均值池化层前加入一层 1x1 的卷积层来进一步的混合特征。

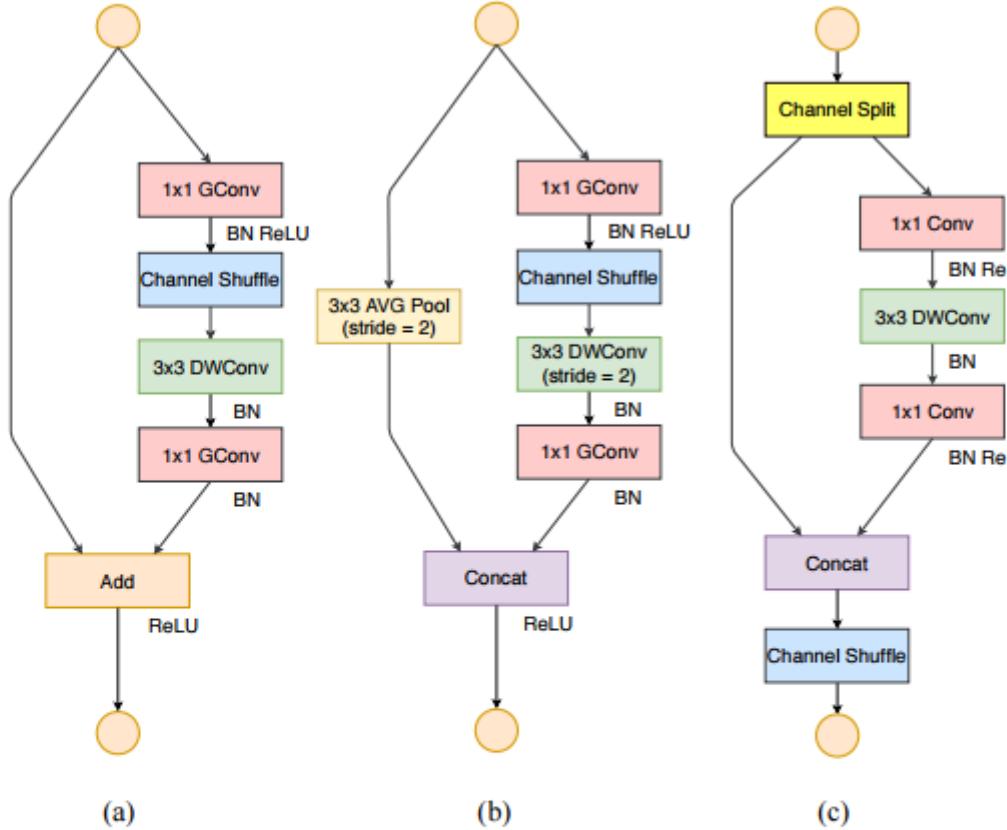


Fig. 3: Building blocks of ShuffleNet v1 [15] and this work. (a) the general building block; (b) the ShuffleNet unit for spatial down sampling; (c) our unit for spatial down sampling ( $2 \times$ ). **DWConv**: depthwise group convolution.

征。

网络结构

Layer	Output size	KSize	Stride	Repeat	Output channels			
					0.5×	1×	1.5×	2×
Image	224×224				3	3	3	3
Conv1	112×112	3×3	2					
MaxPool	56×56	3×3	2	1	24	24	24	24
Stage2	28×28 28×28		2 1	1 3	48	116	176	244
Stage3	14×14 14×14		2 1	1 7	96	232	352	488
Stage4	7×7 7×7		2 1	1 3	192	464	704	976
Conv5	7×7	1×1	1	1	1024	1024	1024	2048
GlobalPool	1×1	7×7						
FC					1000	1000	1000	1000
FLOPs					41M	146M	299M	591M
# of Weights					1.4M	2.3M	3.5M	7.4M

Table 5: Overall architecture of ShuffleNet v2, for four different levels of complexities.

6.4 ShuffleNet-v2 具有高精度的原因.

- 由于高效，可以增加更多的 channel，增加网络容量
- 采用 split 使得一部分特征直接与下面的 block 相连，特征复用 (DenseNet)

### 0.10. 17.10 现有移动端开源框架及其特点.

0.10.1. 17.10.1 NCNN. 1、开源时间：2017 年 7 月

2、开源用户：腾讯优图

3、GitHub 地址：<https://github.com/Tencent/ncnn>

4、特点：

- 1) NCNN 考虑了手机端的硬件和系统差异以及调用方式，架构设计以手机端运行为主要原则。
- 2) 无第三方依赖，跨平台，手机端 CPU 的速度快于目前所有已知的开源框架（以开源时间为参照对象）。
- 3) 基于 ncnn，开发者能够将深度学习算法轻松移植到手机端高效执行，开发出人工智能 APP。

5、功能：

- 1、NCNN 支持卷积神经网络、多分支多输入的复杂网络结构，如 vgg、googlenet、resnet、squeezenet 等。
- 2、NCNN 无需依赖任何第三方库。
- 3、NCNN 全部使用 C/C++ 实现，以及跨平台的 cmake 编译系统，可轻松移植到其他系统和设备上。
- 4、汇编级优化，计算速度极快。使用 ARM NEON 指令集实现卷积层，全连接层，池化层等大部分 CNN 关键层。
- 5、精细的数据结构设计，没有采用需消耗大量内存的通常框架——im2col + 矩阵乘法，使得内存占用极低。
- 6、支持多核并行计算，优化 CPU 调度。
- 7、整体库体积小于 500K，可精简到小于 300K。
- 8、可扩展的模型设计，支持 8bit 量化和半精度浮点存储。
- 9、支持直接内存引用加载网络模型。
- 10、可注册自定义层实现并扩展。

## 6、NCNN 在 Android 端部署示例

- 1) 选择合适的 Android Studio 版本并安装。
- 2) 根据需求选择 NDK 版本并安装。
- 3) 在 Android Studio 上配置 NDK 的环境变量。
- 4) 根据自己需要编译 NCNN sdk

```
mkdir build-android cd build-android cmake -DCMAKE_TOOLCHAIN_FILE=$ANDROID_NDK/build/cmake/
```

安装完成之后，install 下有 include 和 lib 两个文件夹。

备注：

ANDROID\_ABI 是架构名字，"armeabi-v7a" 支持绝大部分手机硬件

ANDROID\_ARM\_NEON 是否使用 NEON 指令集，设为 ON 支持绝大部分手机硬件

ANDROID\_PLATFORM 指定最低系统版本，"android-14" 就是 android-4.0

- 5) 进行 NDK 开发。

Facebook 开源高性能内核库 QNNPACK <https://baijiahao.baidu.com/s?id=1615725346726413945&wfr=spider&for=pc>  
[http://www.sohu.com/a/272158070\\_610300](http://www.sohu.com/a/272158070_610300)

支持移动端深度学习的几种开源框架 <https://blog.csdn.net/zchang81/article/details/74280019>

0.10.3. 17.10.3 Prestissimo. 1、开源时间：2017 年 11 月

2、开源用户：九言科技

3、GitHub 地址：<https://github.com/in66-dev/In-Prestissimo>

4、功能特点：

### 基础功能

- 支持卷积神经网络，支持多输入和多分支结构
- 精炼简洁的 API 设计，使用方便
- 提供调试接口，支持打印各个层的数据以及耗时
- 不依赖任何第三方计算框架，整体库体积 500K 左右（32 位约 400k，64 位约 600k）
- 纯 C++ 实现，跨平台，支持 android 和 ios
- 模型为纯二进制文件，不暴露开发者设计的网络结构

### 极快的速度

- 大到框架设计，小到汇编书写上全方位的优化，iphone7 上跑 SqueezeNet 仅需 26ms（单线程）
- 支持浮点 (float) 和整型 (int) 两种运算模式，float 模式精度与 caffe 相同，int 模式运算速度快，大部分网络用 int 的精度便已经足够
- 以巧妙的内存布局提升 cpu 的 cache 命中率，在中低端机型上性能依然强劲
- 针对 float-arm32, float-arm64, int-arm32, int-arm64 四个分支均做了细致的优化，保证 arm32 位和 arm64 位版本都有非常好的性能

### SqueezeNet-v1.1 测试结果

Note: 手机测试性能存在一定的抖动，连续多次运算取平均时间

Note: 像华为 mate8, mate9, Google nexus 6 虽然是 64 位的 CPU，但测试用的是 32 位的库，因此 cpu 架构依然写 arm-v7a

CPU 架构	机型	CPU	ncnn (4 线程)	mdl	Prestissimo_float
arm-v7a	小米 2	高通 APQ8064 1.5GHz	185 ms	370 ms	184 ms
arm-v7a	小米 2s	四核骁龙 APQ8064 Pro 1.7GHz	166 ms	-	136 ms
arm-v7a	红米 Note 4x	骁龙 625 四核 2.0GHz	124 ms	306 ms	202 ms
arm-v7a	Google Nexus 6	骁龙 805 四核 2.7GHz	84 ms	245 ms	103 ms
arm-v7a	Vivo x6d	联发科 MT6752 1.7GHz	245 ms	502 ms	370 ms
arm-v7a	华为 Mate 8	海思麒麟 950 4 大 4 小 2.3GHz 1.8GHz	75 ms	180 ms	95 ms

CPU 架构	机型	CPU	ncnn (4 线程)	mdl	Prestissimo
arm-v7a	华为 Mate 9	海思麒麟 960 4 大 4 小 2.4GHz 1.8GHz	61 ms	170 ms	-
arm-v8	iphone7	Apple A10 Fusion 2.34GHz	-	-	-

### 未开放特性

- 多核并行加速 (多核机器可以再提升 30%-100% 的速度)
- depthwise 卷积运算 (支持 mobilenet)
- 模型压缩功能, 压缩后的模型体积可缩小到 20% 以下
- GPU 运算模式 (Android 基于 opengl es 3.1, ios 基于 metal)

### 同类框架对比

框架	caffe	tensorflow	mdl-android	mdl-ios	ncnn	CoreML	Prestissimo
计算硬件	cpu	cpu	cpu	gpu	cpu	gpu	cpu (gpu 版)
计算速度	慢	慢	慢	很快	很快	极快	极快
库大小	大	较大	中等	小	小	小	小
兼容性	好	好	好	限 ios8 以上	很好	仅支持 ios11	很好
模型支持度	很好	好	-	差 (仅限指定模型)	较好	-	中等 (当前版本不支持)

### 使用方法-模型转换

绝影支持的是私有的模型文件格式, 需要把 caffe 训练出来的模型转换为.prestissimo 格式, 模型转换工具为 `caffe2Prestissimo.out`。`caffe2Prestissimo.out` 依赖 protobuf 3.30。将 `XXX.prototxt` 和 `YYY.caffemodel` 转化为 Prestissimo 模型 `ZZZ.prestissimo`: (得到) `./caffe2Prestissimo.out XXX.prototxt YYY.caffemodel ZZZ.prestissimo`

0.10.4. 17.10.4 MDL (mobile-deep-learning). 1、开源时间: 2017 年 9 月 (已暂停更新)

2、开源用户: 百度

3、GitHub 地址: <https://github.com/allonli/mobile-deep-learning>

4、功能特点:

- 一键部署, 脚本参数就可以切换 ios 或者 android
- 支持 iOS gpu 运行 MobileNet、squeezene
- 已经测试过可以稳定运行 MobileNet、GoogLeNet v1、squeezene、ResNet-50 模型
- 体积极小, 无任何第三方依赖。纯手工打造。
- 提供量化函数, 对 32 位 float 转 8 位 uint 直接支持, 模型体积量化后 4M 上下
- 与 ARM 相关算法团队线上线下多次沟通, 针对 ARM 平台会持续优化
- NEON 使用涵盖了卷积、归一化、池化所有方面的操作

MDL 框架主要包括：模型转换模块（MDL Converter）、模型加载模块（Loader）、网络管理模块（Net）、矩阵运算模块（Gemm）及供 Android 端调用的 JNI 接口层（JNI Interfaces）。

其中，模型转换模块主要负责将 Caffe 模型转为 MDL 模型，同时支持将 32bit 浮点型参数量化为 8bit 参数，从而极大地压缩模型体积；模型加载模块主要完成模型的反量化及加载校验、网络注册等过程，网络管理模块主要负责网络中各层 Layer 的初始化及管理工作；MDL 提供了供 Android 端调用的 JNI 接口层，开发者可以通过调用 JNI 接口轻松完成加载及预测过程。

## 6、MDL 的性能及兼容性

- 体积 armv7 300k+
- 速度 iOS GPU mobilenet 可以达到 40ms、squeezenet 可以达到 30ms

MDL 从立项到开源，已经迭代了一年多。移动端比较关注的多个指标都表现良好，如体积、功耗、速度。百度内部产品线在应用前也进行过多次对比，和已开源的相关项目对比，MDL 能够在保证速度和能耗的同时支持多种深度学习模型，如 mobilenet、googlenet v1、squeezenet 等，且具有 iOS GPU 版本，squeezenet 一次运行最快可以达到 3-40ms。

### 同类框架对比

框架 Caffe2TensorFlowNCNNMDL(CPU)MDL(GPU) 硬件 CPUCPUCPUCPUGPU 速度 慢慢快快极快体积大大小小兼容 Android&iOSAndroid&iOSAndroid&iOSAndroid&iOSiOS 与支持 CNN 的移动端框架对比，MDL 速度快、性能稳定、兼容性好、demo 完备。

### 兼容性

MDL 在 iOS 和 Android 平台均可以稳定运行，其中 iOS10 及以上平台有基于 GPU 运算的 API，性能表现非常出色，在 Android 平台则是纯 CPU 运行。高中低端机型运行状态和手机百度及其他 App 上的覆盖都有绝对优势。

MDL 同时也支持 Caffe 模型直接转换为 MDL 模型。

0.10.5. 17.10.5 Paddle-Mobile. 1、开源时间：持续更新，已到 3.0 版本

2、开源用户：百度

3、GitHub 地址：<https://github.com/PaddlePaddle/paddle-mobile>

4、功能特点：

### 功能特点

- 高性能支持 ARM CPU
- 支持 Mali GPU
- 支持 Adreno GPU
- 支持苹果设备的 GPU Metal 实现

### 1. 配置模型部署文件 (.yml)

模型部署文件详细描述了需要部署的模型以及生成库的信息，MACE 根据该文件最终生成对应的库文件。

### 2. 编译 MACE 库

编译 MACE 的静态库或者动态库。

### 3. 转换模型

将 TensorFlow 或者 Caffe 的模型转为 MACE 的模型。

#### 4.1. 部署

根据不同使用目的集成 Build 阶段生成的库文件，然后调用 MACE 相应的接口执行模型。

#### 4.2. 命令行运行

MACE 提供了命令行工具，可以在命令行运行模型，可以用来测试模型运行时间，内存占用和正确性。

#### 4.3. Benchmark

MACE 提供了命令行 benchmark 工具，可以细粒度的查看模型中所涉及的所有算子的运行时间。

7、MACE 在哪些角度进行了优化？

**MACE** 专为移动端异构计算平台优化的神经网络计算框架。主要从以下的角度做了专门的优化：

- 性能
- 代码经过 NEON 指令，OpenCL 以及 Hexagon HVX 专门优化，并且采用 [Winograd 算法](#)来进行卷积操作的加速。此外，还对启动速度进行了专门的优化。
- 功耗
- 支持芯片的功耗管理，例如 ARM 的 big.LITTLE 调度，以及高通 Adreno GPU 功耗选项。
- 系统响应
- 支持自动拆解长时间的 OpenCL 计算任务，来保证 UI 渲染任务能够做到较好的抢占调度，从而保证系统 UI 的相应和用户体验。
- 内存占用
- 通过运用内存依赖分析技术，以及内存复用，减少内存的占用。另外，保持尽量少的外部依赖，保证代码尺寸精简。
- 模型加密与保护
- 模型保护是重要设计目标之一。支持将模型转换成 C++ 代码，以及关键常量字符混淆，增加逆向的难度。
- 硬件支持范围

其组件包括：

- TensorFlow 模型 (TensorFlow Model)：保存在磁盘中的训练模型。
- TensorFlow Lite 转化器 (TensorFlow Lite Converter)：将模型转换成 TensorFlow Lite 文件格式的项目。
- TensorFlow Lite 模型文件 (TensorFlow Lite Model File)：基于 FlatBuffers，适配最大速度和最小规模的模型。

6、移动端开发步骤：

Android Studio 3.0, SDK Version API26, NDK Version 14

- 步骤:1. 将此项目导入到 Android Studio:<https://github.com/tensorflow/tensorflow/tree/master/tensorflow/contrib/lite>
2. 下载移动端的模型(model)和标签数据(labels):[https://storage.googleapis.com/download.tensorflow.org/models/tflite/mobilenet\\_v1\\_224\\_android\\_quant\\_2017\\_11\\_08.zip](https://storage.googleapis.com/download.tensorflow.org/models/tflite/mobilenet_v1_224_android_quant_2017_11_08.zip)
3. 下载完成解压 mobilenet\_v1\_224\_android\_quant\_2017\_11\_08.zip 文件得到一个 xxx.tflite 和 labes.txt 文件，分别是模型和标签文件，并且把这两个文件复制到 assets 文件夹下。
4. 构建 app, run.....

17.7.9 TensorFlow Lite 和 TensorFlow Mobile 的区别？

- TensorFlow Lite 是 TensorFlow Mobile 的进化版。
- 在大多数情况下，TensorFlow Lite 拥有较小的二进制大小，更少的依赖以及更好的性能。
- 相比 TensorFlow Mobile 是对完整 TensorFlow 的裁减，TensorFlow Lite 基本就是重新实现了。从内部实现来说，在 TensorFlow 内核最基本的 OP, Context 等数据结构，都是新的。从外在表现来说，模型文件从 PB 格式改成了 FlatBuffers 格式，TensorFlow 的 size 有大幅度优化，降至 300K，然后提供一个 converter 将普通 TensorFlow 模型转化成 TensorFlow Lite 需要的格式。因此，无论从哪方面看，TensorFlow Lite 都是一个新的实现方案。

0.10.9. 17.10.9 *PocketFlow*. 1、开源时间：2018 年 9 月

2、开源用户：腾讯

3、GitHub 地址：<https://github.com/Tencent/PocketFlow>

4、简介：

全球首个自动模型压缩框架

一款面向移动端 AI 开发者的自动模型压缩框架，集成了当前主流的模型压缩与训练算法，结合自研超参数优化组件实现了全程自动化托管式的模型压缩与加速。开发者无需了解具体算法细节，即可快速地将 AI 技术部署到移动端产品上，实现了自动托管式模型压缩与加速，实现用户数据的本地高效处理。

5、框架介绍

开发者将未压缩的原始模型作为 PocketFlow 框架的输入，同时指定期望的性能指标，例如模型的压缩和/或加速倍数；在每一轮迭代过程中，超参数优化组件选取一组超参数取值组合，之后模型压缩/加速算法组件基于该超参数取值组合，对原始模型进行压缩，得到一个压缩后的候选模型；基于对候选模型进行性能评估的结果，超参数优化组件调整自身的模型参数，并选取一组新的超参数取值组合，以开始下一轮迭代过程；当迭代终止时，PocketFlow 选取最优的超参数取值组合以及对应的候选模型，作为最终输出，返回给开发者用作移动端的模型部署。

### 6、PocketFlow 如何实现模型压缩与加速？

具体地，PocketFlow 通过下列各个算法组件的有效结合，实现了精度损失更小、自动化程度更高的深度学习模型的压缩与加速：

- a) 通道剪枝 (channel pruning) 组件：在 CNN 网络中，通过对特征图中的通道维度进行剪枝，可以同时降低模型大小和计算复杂度，并且压缩后的模型可以直接基于现有的深度学习框架进行部署。在 CIFAR-10 图像分类任务中，通过对 ResNet-56 模型进行通道剪枝，可以实现 2.5 倍加速下分类精度损失 0.4%，3.3 倍加速下精度损失 0.7%。
- b) 权重稀疏化 (weight sparsification) 组件：通过对网络权重引入稀疏性约束，可以大幅度降低网络权重中的非零元素个数；压缩后模型的网络权重可以以稀疏矩阵的形式进行存储和传输，从而实现模型压缩。对于 MobileNet 图像分类模型，在删去 50% 网络权重后，在 ImageNet 数据集上的 Top-1 分类精度损失仅为 0.6%。
- c) 权重量化 (weight quantization) 组件：通过对网络权重引入量化约束，可以降低用于表示每个网络权重所需的比特数；团队同时提供了对于均匀和非均匀两大类量化算法的支持，可以充分利用 ARM 和 FPGA 等设备的硬件优化，以提升移动端的计算效率，并为未来的神经网络芯片设计提供软件支持。以用于 ImageNet 图像分类任务的 ResNet-18 模型为例，在 8 比特定点量化下可以实现精度无损的 4 倍压缩。
- d) 网络蒸馏 (network distillation) 组件：对于上述各种模型压缩组件，通过将未压缩的原始模型的输出作为额外的监督信息，指导压缩后模型的训练，在压缩/加速倍数不变的前提下均可以获得 0.5%-2.0% 不等的精度提升。
- e) 多 GPU 训练 (multi-GPU training) 组件：深度学习模型训练过程对计算资源要求较高，单个 GPU 难以在短时间内完成模型训练，因此团队提供了对于多机多卡分布式训练的全面支持，以加快使用者的开发流程。无论是基于 ImageNet 数据的 Resnet-50 图像分类模型还是基于 WMT14 数据的 Transformer 机器翻译模型，均可以在一个小时训练完毕。[1]

## 参考文献

[1] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Jiezhang Cao, Qingyao Wu, Junzhou Huang, Jinhui Zhu, 「Discrimination-aware Channel Pruning for Deep Neural Networks」, In Proc. of the 32nd Annual Conference on Neural Information Processing Systems, NIPS '18, Montreal, Canada, December 2018.

[2] Jiaxiang Wu, Weidong Huang, Junzhou Huang, Tong Zhang, 「Error Compensated Quantized SGD and its Applications to Large-scale Distributed Optimization」, In Proc. of the 35th International Conference on Machine Learning, ICML'18, Stockholm, Sweden, July 2018.

0.10.10. 17.10.10 其他几款支持移动端深度学习的开源框架. <https://blog.csdn.net/zchang81/article/details/74280>

0.10.11. 17.10.11 MDL、NCNN 和 TFLite 比较. 百度-MDL 框架、腾讯-NCNN 框架和谷歌 TFLite 框架比较。

	MDL	NCNN	TFLite
代码质量	中	高	很高
跨平台	√	√	√
支持 caffe 模型	√	√	×
支持 TensorFlow 模型	×	×	√
CPU NEON 指令优化	√	√	√
GPU 加速	√	×	×

相同点:

- 只含推理 (inference) 功能，使用的模型文件需要通过离线的方式训练得到。
- 最终生成的库尺寸较小，均小于 500kB。
- 为了提升执行速度，都使用了 ARM NEON 指令进行加速。
- 跨平台，iOS 和 Android 系统都支持。

不同点:

- MDL 和 NCNN 均是只支持 Caffe 框架生成的模型文件，而 TFLite 则毫无意外的只支持自家大哥 TensorFlow 框架生成的模型文件。
- MDL 支持利用 iOS 系统的 Metal 框架进行 GPU 加速，能够显著提升在 iPhone 上的运行速度，达到准实时的效果。而 NCNN 和 TFLite 还没有这个功能。

## 0.11. 17.11 移动端开源框架部署.

0.11.1. 17.8.1 以 NCNN 为例. 部署步骤

0.11.2. 17.8.2 以 QNNPACK 为例. 部署步骤

0.11.3. 17.8.4 在 Android 手机上使用 MACE 实现图像分类.

#### 0.11.4. 17.8.3 在 Android 手机上使用 PaddleMobile 实现图像分类. 编译 paddle-mobile 库

1) 编译 Android 能够使用的 CPP 库: 编译 Android 的 paddle-mobile 库, 可选择使用 Docker 编译和 Ubuntu 交叉编译, 这里介绍使用 Ubuntu 交叉编译 paddle-mobile 库。

注: 在 Android 项目, Java 代码调用 CPP 代码, CPP 的函数需要遵循一定的命名规范, 比如 Java\_ 包名 \_ 类名 \_ 对应的 Java 的方法名。

目前官方提供了 5 个可以给 Java 调用的函数, 该代码在:paddle-mobile/src/jni/paddle\_mobile\_jni.cpp, 如果想要让这些函数能够在自己的包名下的类调用, 就要修改 CPP 的函数名称修改如下:

```
 JNIEXPORT jboolean JNICALL Java_com_baidu_paddle_PML_load(JNIEnv *env,
 jclass thiz, jstring modelPath) { ANDROIDLOGI("load invoked"); bool optimize = true;
 return getPaddleMobileInstance()->Load(jstring2cppstring(env, modelPath), optimize); }
```

笔者项目的包名为 com.example.paddlemobile1, 在这个包下有一个 ImageRecognition.java 的程序来对应这个 CPP 程序, 那么修改 load 函数如下:

```
 JNIEXPORT jboolean JNICALL Java_com_example_paddlemobile1_ImageRecognition_load(JNIEnv *
*env, jclass thiz, jstring modelPath) { ANDROIDLOGI("load invoked"); bool optimize =
true; return getPaddleMobileInstance()->Load(jstring2cppstring(env, modelPath), optimize);
}
```

#### 使用 Ubuntu 交叉编译 paddle-mobile 库

1、下载和解压 NDK。

```
 wget https://dl.google.com/android/repository/android-ndk-r17b-linux-x86_64.zip
 unzip android-ndk-r17b-linux-x86_64.zip
```

2、设置 NDK 环境变量, 目录是 NDK 的解压目录。

```
 export NDK_ROOT="/home/test/paddlepaddle/android-ndk-r17b"
```

设置好之后, 可以使用以下的命令查看配置情况。

```
root@test:/home/test/paddlepaddle# echo $NDK_ROOT
/home/test/paddlepaddle/android-ndk-r17b
```

3、安装 cmake, 需要安装较高版本的, 笔者的 cmake 版本是 3.11.2。

下载 cmake 源码

```
 wget https://cmake.org/files/v3.11/cmake-3.11.2.tar.gz
```

解压 cmake 源码

```
 tar -zxvf cmake-3.11.2.tar.gz
```

进入到 cmake 源码根目录，并执行 bootstrap。

```
cd cmake-3.11.2  
.bootstrap
```

最后执行以下两条命令开始安装 cmake。

```
make  
make install
```

安装完成之后，可以使用 cmake --version 是否安装成功。

```
root@test:/home/test/paddlepaddle# cmake --version  
cmake version 3.11.2
```

```
CMake suite maintained and supported by Kitware (kitware.com/cmake).
```

4、克隆 paddle-mobile 源码。

```
git clone https://github.com/PaddlePaddle/paddle-mobile.git
```

5、进入到 paddle-mobile 的 tools 目录下，执行编译。

```
cd paddle-mobile/tools/  
sh build.sh android
```

(可选) 如果想编译针对某一个网络编译更小的库时，可以在命令后面加上相应的参数，如下：

```
sh build.sh android googlenet
```

6、最后会在 paddle-mobile/build/release/arm-v7a/build 目录下生产 paddle-mobile 库。

```
[] root@test:/home/test/paddlepaddle/paddle-mobile/build/release/arm-v7a/build ls libpaddle-  
mobile.so
```

libpaddle-mobile.so 就是我们在开发 Android 项目的时候使用到的 paddle-mobile 库。

### 创建 Android 项目

1、首先使用 Android Studio 创建一个普通的 Android 项目，包名为 com.example.paddlemobile1

2、在 main 目录下创建两个 assets/paddle\_models 文件夹，这个文件夹存放 PaddleFluid 训练好的预测模型。PaddleMobile 支持量化模型，使用模型量化可以把模型缩小至原来的四分之一，如果使用量化模型，那加载模型的接口也有修改一下，使用以下的接口加载模型：

```
[] public static native boolean loadQualified(String modelDir);
```

3、在 main 目录下创建一个 jniLibs 文件夹，这个文件夹是存放 CPP 编译库的，在本项目中就存放上一部分编译的 libpaddle-mobile.so

4、在 Android 项目的配置文件夹中加上权限声明，因为我们要使用到读取相册和使用相机，所以加上以下的权限声明：

```
[] <uses-permission android:name="android.permission.CAMERA" /> <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" /> <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

5、修改 `activity_main.xml` 界面，修改成如下：

```
[<?xml version="1.0" encoding="utf-8"?> <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android" xmlns:app="http://schemas.android.com/apk/res-auto" xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent" android:layout_height="match_parent" tools:context=".MainActivity"> <LinearLayout android:id="@+id/btn_ll" android:layout_alignParentBottom="true" android:layout_width="match_parent" android:layout_height="wrap_content" android:orientation="horizontal"> <Button android:id="@+id/use_photo" android:layout_weight="1" android:layout_width="0dp" android:layout_height="wrap_content" android:text="相册" /> <Button android:id="@+id/start_camera" android:layout_weight="1" android:layout_width="0dp" android:layout_height="wrap_content" android:text="拍照" /> </LinearLayout> <TextView android:layout_above="@+id/btn_ll" android:id="@+id/result_text" android:textSize="16sp" android:layout_width="match_parent" android:hint="预测结果会在这里显示" android:layout_height="100dp" /> <ImageView android:layout_alignParentTop="true" android:layout_above="@+id/result_text" android:id="@+id/show_image" android:layout_width="match_parent" android:layout_height="match_parent" /> </RelativeLayout>]
```

6、创建一个 `ImageRecognition.java` 的 Java 程序，这个程序的作用就是调用 `paddle-mobile/src/jni/` 的函数，对应的是里面的函数。目前支持以下几个接口。

```
[package com.example.paddlemobile1;
public class ImageRecognition { // set thread num public static native void setThread(int threadCount);
    //Load seperated parameters public static native boolean load(String modelDir);
    // load qualified model public static native boolean loadQualified(String modelDir);
    // Load combined parameters public static native boolean loadCombined(String modelPath, String paramPath);
    // load qualified model public static native boolean loadCombinedQualified(String modelPath, String paramPath);
    // object detection public static native float[] predictImage(float[] buf, int[] ddims);}
```

```
// predict yuv image public static native float[] predictYuv(byte[] buf, int imgWidth, int  
imgHeight, int[] ddims, float[] meanValues);
```

```
// clear model public static void clear(); }
```

7、然后编写一个 `PhotoUtil.java` 的工具类。

```
[ ] package com.example.paddlemobile1;
```

```
import android.app.Activity; import android.content.Context; import android.content.Intent;  
import android.database.Cursor; import android.graphics.Bitmap; import android.graphics.BitmapFactory;  
import android.net.Uri; import android.os.Build; import android.provider.MediaStore; import  
android.support.v4.content.FileProvider; import android.util.Log;
```

```
import java.io.File; import java.io.IOException; import java.util.Arrays;
```

```
public class PhotoUtil { // start camera public static Uri start_camera(Activity activity,  
int requestCode) { Uri imageUri; // save image in cache path File outputImage =  
new File(activity.getExternalCacheDir(), "out_image.jpg"); try { if (outputImage.exists())  
{ outputImage.delete(); } outputImage.createNewFile(); } catch (IOException e) {  
e.printStackTrace(); } if (Build.VERSION.SDK_INT >= 24) { // compatible with Android  
7.0 or over imageUri = FileProvider.getUriForFile(activity, "com.example.paddlemobile1",  
outputImage); } else { imageUri = Uri.fromFile(outputImage); } // set system cam-  
era Action Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE); // set  
save photo path intent.putExtra(MediaStore.EXTRA_OUTPUT, imageUri); // set photo  
quality, min is 0, max is 1 intent.putExtra(MediaStore.EXTRA_VIDEO_QUALITY, 0);  
activity.startActivityForResult(intent, requestCode); return imageUri; }
```

```
// get picture in photo public static void use_photo(Activity activity, int requestCode){
```

```
Intent intent = new Intent(Intent.ACTION_PICK); intent.setType("image/*"); activity.startActivityForResult(intent,  
requestCode); }
```

```
// get photo from Uri public static String get_path_from_URI(Context context, Uri uri)  
{ String result; Cursor cursor = context.getContentResolver().query(uri, null, null, null, null);  
if (cursor == null) { result = uri.getPath(); } else { cursor.moveToFirst(); int idx = cur-  
sor.getColumnIndex(MediaStore.Images.ImageColumns.DATA); result = cursor.getString(idx);  
cursor.close(); } return result; }
```

```
// Compress the image to the size of the training image, and change RGB public static  
float[] getScaledMatrix(Bitmap bitmap, int desWidth, int desHeight) { float[] dataBuf =  
new float[3 * desWidth * desHeight]; int rIndex; int gIndex; int bIndex; int[] pixels = new  
int[desWidth * desHeight]; Bitmap bm = Bitmap.createScaledBitmap(bitmap, desWidth,  
desHeight, false); bm.getPixels(pixels, 0, desWidth, 0, 0, desWidth, desHeight); int j = 0; int
```

```

k = 0; for (int i = 0; i < pixels.length; i++) { int clr = pixels[i]; j = i / desHeight; k = i % desWidth;
rIndex = j * desWidth + k; gIndex = rIndex + desHeight * desWidth; bIndex = gIndex + desHeight * desWidth;
dataBuf[rIndex] = (float) ((clr & 0x00ff0000) >> 16) - 148; dataBuf[gIndex] = (float) ((clr & 0x0000ff00) >> 8) - 148;
dataBuf[bIndex] = (float) ((clr & 0x000000ff) >> 0) - 148;
} if (bm.isRecycled()) { bm.recycle(); } return dataBuf; }
// compress picture public static Bitmap getScaleBitmap(String filePath) {
BitmapFactory.Options opt = new BitmapFactory.Options(); opt.inJustDecodeBounds = true; BitmapFactory.decodeFile(filePath, opt);
int bmpWidth = opt.outWidth; int bmpHeight = opt.outHeight;
int maxSize = 500;
// compress picture with inSampleSize opt.inSampleSize = 1; while (true) { if (bmpWidth / opt.inSampleSize < maxSize || bmpHeight / opt.inSampleSize < maxSize) { break; }
opt.inSampleSize *= 2; } opt.inJustDecodeBounds = false; return BitmapFactory.decodeFile(filePath, opt); } }

```

- `start_camera()` 方法是启动相机并返回图片的 URI。
- `use_photo()` 方法是打开相册，获取到的图片 URI 在回到函数中获取。
- `get_path_from_URI()` 方法是把图片的 URI 转换成绝对路径。
- `getScaledMatrix()` 方法是把图片压缩成跟训练时的大小，并转换成预测需要用的数据格式浮点数组。
- `getScaleBitmap()` 方法是对图片进行等比例压缩，减少内存的支出。

8、最后修改 `MainActivity.java`，修改如下：

- `load_model()` 方法是加载预测模型的。
- `clear_model()` 方法是清空预测模型的。
- `copy_file_from_asset()` 方法是把预测模型复制到内存卡上。
- `predict_image()` 方法是预测图片的。
- `get_max_result()` 方法是获取概率最大的预测结果。
- `request_permissions()` 方法是动态请求权限的。

因为使用到图像加载框架 Glide，所以要在 `build.gradle` 加入以下的引用。

```
implementation 'com.github.bumptech.glide:glide:4.3.1'
```

8、最后运行项目，选择图片预测就会得到结果。

### 0.12. 17.9 移动端开源框架部署疑难. 增加常见的几个问题

知识蒸馏(Distillation)相关论文阅读(1)——Distilling the Knowledge in a Neural Network  
(以及代码复现)





## 第十八章 \_ 后端架构选型、离线及实时计算

Markdown Revision 1;

Date: 2018/11/11

Editor: 梁志成

Contact: superzhicheng@foxmail.com

### 1. 18.1 为什么需要分布式计算？

在这个数据爆炸的时代，产生的数据量不断地在攀升，从 GB,TB,PB,ZB. 挖掘其中数据的价值也是企业在不断地追求的终极目标。但是要想对海量的数据进行挖掘，首先要考虑的就是海量数据的存储问题，比如 Tb 量级的数据。

谈到数据的存储，则不得不说的是磁盘的数据读写速度问题。早在上个世纪 90 年代初期，普通硬盘的可以存储的容量大概是 1G 左右，硬盘的读取速度大概为 4.4MB/s. 读取一张硬盘大概需要 5 分钟时间，但是如今硬盘的容量都在 1TB 左右了，相比扩展了近千倍。但是硬盘的读取速度大概是 100MB/s。读完一个硬盘所需要的时间大概是 2.5 个小时。所以如果是基于 TB 级别的数据进行分析的话，光硬盘读取完数据都要好几天了，更谈不上计算分析了。那么该如何处理大数据的存储，计算分析呢？

一个很简单的减少数据读写时间的方法就是同时从多个硬盘上读写数据，比如，如果我们有 100 个硬盘，每个硬盘存储 1% 的数据，并行读取，那么不到两分钟就可以完成之前需要 2.5 小时的数据读写任务了。这就是大数据中的分布式存储的模型。当然实现分布式存储还需要解决很多问题，比如硬件故障的问题，使用多台主机进行分布式存储时，若主机故障，会出现数据丢失的问题，所以有了副本机制：系统中保存数据的副本。一旦有系统发生故障，就可以使用另外的副本进行替换（著名的 RAID 冗余磁盘阵列就是按这个原理实现的）。其次比如一个很大的文件如何进行拆分存储，读取拆分以后的文件如何进行校验都是要考虑的问题。比如我们使用 Hadoop 中的 HDFS 也面临这个问题，只是框架给我们实现了这些问题的解决办法，开发中开发者不用考虑这些问题，底层框架已经实现了封装。

同样假如有一个 10TB 的文件，我们要统计其中某个关键字的出现次数，传统的做法是遍历整个文件，然后统计出关键字的出现次数，这样效率会特别特别低。基于分布式存储以后，数据被分布式存储在不同的服务器上，那么我们就可以使用分布式计算框架（比如

Github 地址: <https://github.com/apache/mahout>

**2.4. 18.2.4 Spark MLlib.** MLlib(Machine Learnig lib) 【4】是 Spark 对常用的机器学习算法的实现库，同时包括相关的测试和数据生成器。

MLlib 是 MLBase 一部分，其中 MLBase 分为四部分：MLlib、MLI、ML Optimizer 和 MLRuntime。- ML Optimizer 会选择它认为最适合的已经在内部实现好了的机器学习算法和相关参数，来处理用户输入的数据，并返回模型或别的帮助分析的结果；- MLI 是一个进行特征抽取和高级 ML 编程抽象的算法实现的 API 或平台；- MLlib 是 Spark 实现一些常见的机器学习算法和实用程序，包括分类、回归、聚类、协同过滤、降维以及底层优化，该算法可以进行可扩充；MLRuntime 基于 Spark 计算框架，将 Spark 的分布式计算应用到机器学习领域。

MLlib 主要包含三个部分：- 底层基础：包括 Spark 的运行库、矩阵库和向量库 - 算法库：包含广义线性模型、推荐系统、聚类、决策树和评估的算法 - 实用程序：包括测试数据的生成、外部数据的读入等功能

### 架构图

MLlib 目前支持 4 种常见的机器学习问题：分类、回归、聚类和协同过滤，MLlib 在 Spark 整个生态系统中的位置如图下图所示。

### 生态系统位置

**2.5. 18.2.5 Ray.** Ray 【5】是加州大学伯克利分校实时智能安全执行实验室 (RISE-Lab) 的研究人员针对机器学习领域开发的一种新的分布式计算框架，该框架旨在让基于 Python 的机器学习和深度学习工作负载能够实时执行，并具有类似消息传递接口 (MPI) 的性能和细粒度。

增强学习的场景，按照原理定义，因为没有预先可用的静态标签信息，所以通常需要引入实际的目标系统（为了加快训练，往往是目标系统的模拟环境）来获取反馈信息，用做损失/收益判断，进而完成整个训练过程的闭环反馈。典型的步骤是通过观察特定目标系统的状态，收集反馈信息，判断收益，用这些信息来调整参数，训练模型，并根据新的训练结果产出可用于调整目标系统的行为 Action，输出到目标系统，进而影响目标系统状态变化，完成闭环，如此反复迭代，最终目标是追求某种收益的最大化（比如对 AlphoGo 来说，收益是赢得一盘围棋的比赛）。

在这个过程中，一方面，模拟目标系统，收集状态和反馈信息，判断收益，训练参数，生成 Action 等等行为可能涉及大量的任务和计算（为了选择最佳 Action，可能要并发模拟众多可能的行为）。而这些行为本身可能也是千差万别的异构的任务，任务执行的时间也可能长短不一，执行过程有些可能要求同步，也有些可能更适合异步。

另一方面，整个任务流程的 DAG 图也可能是动态变化的，系统往往可能需要根据前一个环节的结果，调整下一个环节的行为参数或者流程。这种调整，可能是目标系统的需要（比如在自动驾驶过程中遇到行人了，那么我们可能需要模拟计算刹车的距离来判断该采取的行动是刹车还是拐弯，而平时可能不需要这个环节），也可能是增强学习特定训练算法的需要（比如根据多个并行训练的模型的当前收益，调整模型超参数，替换模型等等）。

此外，由于所涉及到的目标系统可能是具体的，现实物理世界中的系统，所以对时效性也可能是有强要求的。举个例子，比如你想要实现的系统是用来控制机器人行走，或者是用来打视频游戏的。那么整个闭环反馈流程就需要在特定的时间限制内完成（比如毫秒级别）。

总结来说，就是增强学习的场景，对分布式计算框架的任务调度延迟，吞吐量和动态修改 DAG 图的能力都可能有很高的要求。按照官方的设计目标，Ray 需要支持异构计算任务，动态计算链路，毫秒级别延迟和每秒调度百万级别任务的能力。

Ray 的目标问题，主要是在类似增强学习这样的场景中所遇到的工程问题。那么增强学习的场景和普通的机器学习，深度学习的场景又有什么不同呢？简单来说，就是对整个处理链路流程的时效性和灵活性有更高的要求。

Ray 框架优点 - 海量任务调度能力 - 毫秒级别的延迟 - 异构任务的支持 - 任务拓扑图动态修改的能力

Ray 没有采用中心任务调度的方案，而是采用了类似层级（hierarchy）调度的方案，除了一个全局的中心调度服务节点（实际上这个中心调度节点也是可以水平拓展的），任务的调度也可以在具体的执行任务的工作节点上，由本地调度服务来管理和执行。与传统的层级调度方案，至上而下分配调度任务的方式不同的是，Ray 采用了至下而上的调度策略。也就是说，任务调度的发起，并不是先提交给全局的中心调度器统筹规划以后再分发给次级调度器的。而是由任务执行节点直接提交给本地的调度器，本地的调度器如果能满足该任务的调度需求就直接完成调度请求，在无法满足的情况下，才会提交给全局调度器，由全局调度器协调转发给有能力满足需求的另外一个节点上的本地调度器去调度执行。

架构设计一方面减少了跨节点的 RPC 开销，另一方面也能规避中心节点的瓶颈问题。当然缺点也不是没有，由于缺乏全局的任务视图，无法进行全局规划，因此任务的拓扑逻辑结构也就未必是最优的了。

## 架构图

## 任务调度图

Ray 架构现状：- API 层以上的部分还比较薄弱，Core 模块核心逻辑仍需时间打磨。- 国内目前除了蚂蚁金服和 RISELab 有针对性的合作以外，关注程度还很低，没有实际的应用实例看到，整体来说还处于比较早期的框架构建阶段。

Github 地址: <https://github.com/ray-project/ray>

**2.6. 18.2.6 Spark stream.** 随着大数据的发展，人们对大数据的处理要求也越来越高，原有的批处理框架 MapReduce 适合离线计算，却无法满足实时性要求较高的业务，如实时推荐、用户行为分析等。Spark Streaming 是建立在 Spark 上的实时计算框架，通过它提供的丰富的 API、基于内存的高速执行引擎，用户可以结合流式、批处理和交互式查询应用。

Spark 是一个类似于 MapReduce 的分布式计算框架，其核心是弹性分布式数据集，提供了比 MapReduce 更丰富的模型，可以在快速在内存中对数据集进行多次迭代，以支持复杂的数据挖掘算法和图形计算算法。Spark Streaming 【6】是一种构建在 Spark 上的实时计算框架，它扩展了 Spark 处理大规模流式数据的能力。

Spark Streaming 的优势在于： - 能运行在 100+ 的结点上，并达到秒级延迟。 - 使用基于内存的 Spark 作为执行引擎，具有高效和容错的特性。 - 能集成 Spark 的批处理和交互查询。 - 为实现复杂的算法提供和批处理类似的简单接口。

#### Spark Streaming 架构图

Spark Streaming 把实时输入数据流以时间片  $\Delta t$ （如 1 秒）为单位切分成块。Spark Streaming 会把每块数据作为一个 RDD，并使用 RDD 操作处理每一小块数据。每个块都会生成一个 Spark Job 处理，最终结果也返回多块。

#### Spark Streaming 基本原理图

正如 Spark Streaming 最初的目标一样，它通过丰富的 API 和基于内存的高速计算引擎让用户可以结合流式处理，批处理和交互查询等应用。因此 Spark Streaming 适合一些需要历史数据和实时数据结合分析的应用场合。当然，对于实时性要求不是特别高的应用也能完全胜任。另外通过 RDD 的数据重用机制可以得到更高效的容错处理。

**2.7. 18.2.7 Horovod.** Horovod 【7】是 Uber 开源的又一个深度学习工具，它的发展吸取了 Facebook 「一小时训练 ImageNet 论文」与百度 Ring Allreduce 的优点，可为用户实现分布式训练提供帮助。

Horovod 支持通过用于高性能并行计算的低层次接口 – 消息传递接口 (MPI) 进行分布式模型训练。有了 MPI，就可以利用分布式 Kubernetes 集群来训练 TensorFlow 和 PyTorch 模型。

分布式 TensorFlow 的参数服务器模型 (parameter server paradigm) 通常需要对大量样板代码进行认真的实现。但是 Horovod 仅需要几行。下面是一个分布式 TensorFlow 项目使用 Horovod 的示例：

```
import tensorflow as tf
```

```
import horovod.tensorflow as hvd
# Initialize Horovod
hvd.init()
# Pin GPU to be used to process local rank (one GPU per process)
config = tf.ConfigProto()
config.gpu_options.visible_device_list = str(hvd.local_rank())
# Build model...
loss = ...
opt = tf.train.AdagradOptimizer(0.01)
# Add Horovod Distributed Optimizer
opt = hvd.DistributedOptimizer(opt)
# Add hook to broadcast variables from rank 0 to all other processes during
# initialization.
hooks = [hvd.BroadcastGlobalVariablesHook(0)]
# Make training operation
train_op = opt.minimize(loss)
# The MonitoredTrainingSession takes care of session initialization,
# restoring from a checkpoint, saving to a checkpoint, and closing when done
# or an error occurs.
with tf.train.MonitoredTrainingSession(checkpoint_dir=“/tmp/train_logs” ,
                                         config=config,
                                         hooks=hooks) as mon_sess:
    while not mon_sess.should_stop():
        # Perform synchronous training.
        mon_sess.run(train_op)
```

在该示例中，粗体文字指进行单个 GPU 分布式项目时必须做的改变：

- hvd.init() 初始化 Horovod。
- config.gpu\_options.visible\_device\_list = str(hvd.local\_rank()) 向每个 TensorFlow 流程分配一个 GPU。
- opt=hvd.DistributedOptimizer(opt) 使用 Horovod 优化器包裹每一个常规 TensorFlow 优化器，Horovod 优化器使用 ring-allreduce 平均梯度。

- `hvd.BroadcastGlobalVariablesHook(0)` 将变量从第一个流程向其他流程传播，以实现一致性初始化。如果该项目无法使用 `MonitoredTrainingSession`，则用户可以运行 `hvd.broadcast_global_variables(0)`。

之后，可以使用 `mpirun` 命令使该项目的多个拷贝在多个服务器中运行：

```
$ mpirun -np 16 -x LD_LIBRARY_PATH -H
server1:4,server2:4,server3:4,server4:4 python train.py
```

`mpirun` 命令向四个节点分布 `train.py`，然后在每个节点的四个 GPU 上运行 `train.py`。  
Github 地址：<https://github.com/uber/horovod>

**2.8. 18.2.8 BigDL.** BigDL 【9】是一种基于 Apache Spark 的分布式深度学习框架。它可以无缝的直接运行在现有的 Apache Spark 和 Hadoop 集群之上。BigDL 的设计吸取了 Torch 框架许多方面的知识，为深度学习提供了全面的支持；包括数值计算和高级神经网络；借助现有的 Spark 集群来运行深度学习计算，并简化存储在 Hadoop 中的大数据集的数据加载。

BigDL 优点：- 丰富的深度学习支持。模拟 Torch 之后，BigDL 为深入学习提供全面支持，包括数字计算（通过 Tensor）和高级神经网络；此外，用户可以使用 BigDL 将预先训练好的 Caffe 或 Torch 模型加载到 Spark 程序中。- 极高的性能。为了实现高性能，BigDL 在每个 Spark 任务中使用英特尔 MKL 和多线程编程。因此，在单节点 Xeon（即与主流 GPU 相当）上，它比开箱即用开源 Caffe，Torch 或 TensorFlow 快了数量级。- 有效地横向扩展。BigDL 可以通过利用 Apache Spark（快速分布式数据处理框架），以及高效实施同步 SGD 和全面减少 Spark 的通信，从而有效地扩展到“大数据规模”上的数据分析

BigDL 缺点：- 对机器要求高 jdk7 上运行性能差在 CentOS 6 和 7 上，要将最大用户进程增加到更大的值（例如 514585）；否则，可能会看到错误，如“无法创建新的本机线程”。  
- 训练和验证的数据会加载到内存，挤占内存

BigDL 满足的应用场景：- 直接在 Hadoop/Spark 框架下使用深度学习进行大数据分析（即将数据存储在 HDFS、HBase、Hive 等数据库上）；- 在 Spark 程序中/工作流中加入深度学习功能；- 利用现有的 Hadoop/Spark 集群来运行深度学习程序，然后将代码与其他的应用场景进行动态共享，例如 ETL（Extract、Transform、Load，即通常所说的数据抽取）、数据仓库（data warehouse）、功能引擎、经典机器学习、图表分析等。

**2.9. 18.2.9 Petastorm.** Petastorm 是一个由 Uber ATG 开发的开源数据访问库。这个库可以直接基于数 TB Parquet 格式的数据集进行单机或分布式训练和深度学习模型评估。Petastorm 支持基于 Python 的机器学习框架，如 Tensorflow、Pytorch 和 PySpark，也可以直接用在 Python 代码中。

### 深度学习集群

即使是在现代硬件上训练深度模型也很耗时，而且在很多情况下，很有必要在多台机器上分配训练负载。典型的深度学习集群需要执行以下几个步骤：- 一台或多台机器读取集中式或本地数据集。- 每台机器计算损失函数的值，并根据模型参数计算梯度。在这一步通常会使用 GPU。- 通过组合估计的梯度（通常由多台机器以分布式的方式计算得出）来更新模型系数。

通常，一个数据集是通过连接多个数据源的记录而生成的。这个由 Apache Spark 的 Python 接口 PySpark 生成的数据集稍后将被用在机器学习训练中。Petastorm 提供了一个简单的功能，使用 Petastorm 特定的元数据对标准的 Parquet 进行了扩展，从而让它可以与 Petastorm 兼容。有了 Petastorm，消费数据就像在 HDFS 或文件系统中创建和迭代读取对象一样简单。Petastorm 使用 PyArrow 来读取 Parquet 文件。

将多个数据源组合到单个表格结构中，从而生成数据集。可以多次使用相同的数据集进行模型训练和评估。

### 深度学习集群

为分布式训练进行分片在分布式训练环境中，每个进程通常负责训练数据的一个子集。一个进程的数据子集与其他进程的数据子集正交。Petastorm 支持将数据集的读时分片转换为正交的样本集。

Petastorm 将数据集的非重叠子集提供给参与分布式训练的不同机器

本地缓存 Petastorm 支持在本地存储中缓存数据。当网络连接速度较慢或带宽很昂贵时，这会派上用场。

Github 地址：<https://github.com/uber/petastorm>

**2.10. 18.2.10 TensorFlowOnSpark.** TensorFlowOnSpark【10】为 Apache Hadoop 和 Apache Spark 集群带来可扩展的深度学习。通过结合深入学习框架 TensorFlow 和大数据框架 Apache Spark、Apache Hadoop 的显着特征，TensorFlowOnSpark 能够在 GPU 和 CPU 服务器集群上实现分布式深度学习。

满足的应用场景：为了利用 TensorFlow 在现有的 Spark 和 Hadoop 集群上进行深度学习。而不需要为深度学习设置单独的集群。

### 架构图

### 运行流程图

优点：- 轻松迁移所有现有的 TensorFlow 程序，<10 行代码更改；- 支持所有 TensorFlow 功能：同步/异步训练，模型/数据并行，推理和 TensorBoard；- 服务器到服务器的直接通

**4.2. 18.4.2 实时流计算过程.** 我们以热卖产品的统计为例，看下传统的计算手段： - 将用户行为、log 等信息清洗后保存在数据库中。 - 将订单信息保存在数据库中。 - 利用触发器或者协程等方式建立本地索引，或者远程的独立索引。 - join 订单信息、订单明细、用户信息、商品信息等等表，聚合统计 20 分钟内热卖产品，并返回 top-10。 - web 或 app 展示。

这是一个假想的场景，但假设你具有处理类似场景的经验，应该会体会到这样一些问题和难处： - 水平扩展问题 (scale-out) 显然，如果是一个具有一定规模的电子商务网站，数据量都是很大的。而交易信息因为涉及事务，所以很难直接舍弃关系型数据库的事务能力，迁移到具有更好的 scale-out 能力的 NoSQL 数据库中。

那么，一般都会做 sharding。历史数据还好说，我们可以按日期来归档，并可以通过批处理式的离线计算，将结果缓存起来。但是，这里的要求是 20 分钟内，这很难。 - 性能问题这个问题，和 scale-out 是一致的，假设我们做了 sharding，因为表分散在各个节点中，所以我们需要多次入库，并在业务层做聚合计算。

问题是，20 分钟的时间要求，我们需要入库多少次呢？10 分钟呢？5 分钟呢？实时呢？

而且，业务层也同样面临着单点计算能力的局限，需要水平扩展，那么还需要考虑一致性的问题。所以，到这里一切都显得很复杂。 - 业务扩展问题

假设我们不仅仅要处理热卖商品的统计，还要统计广告点击、或者迅速根据用户的访问行为判断用户特征以调整其所见的信息，更加符合用户的潜在需求等，那么业务层将会更加复杂。也许你有更好的办法，但实际上，我们需要的是一种新的认知：这个世界发生的事，是实时的。所以我们需要一种实时计算的模型，而不是批处理模型。我们需要的这种模型，必须能够处理很大的数据，所以要有很好的 scale-out 能力，最好是，我们都不需要考虑太多一致性、复制的问题。

那么，这种计算模型就是实时计算模型，也可以认为是流式计算模型。现在假设我们有了这样的模型，我们就可以愉快地设计新的业务场景： - 转发最多的微博是什么？ - 最热卖的商品有哪些？ - 大家都在搜索的热点是什么？ - 我们哪个广告，在哪个位置，被点击最多？或者说，我们可以问： 这个世界，在发生什么？

最热的微博话题是什么？我们以一个简单的滑动窗口计数的问题，来揭开所谓实时计算的神秘面纱。假设，我们的业务要求是：统计 20 分钟内最热的 10 个微博话题。

解决这个问题，我们需要考虑： - 数据源

这里，假设我们的数据，来自微博长连接推送的话题。 - 问题建模

我们认为的话题是 # 号扩起来的话题，最热的话题是此话题出现的次数比其它话题都要多。比如： @foreach\_break : 你好, # 世界 #, 我爱你, # 微博 #。“世界”和“微博”就是话题。

- 计算引擎采用 storm

- 定义时间

时间的定义是一件很难的事情，取决于所需的精度是多少。根据实际，我们一般采用 tick 来表示时刻这一概念。在 storm 的基础设施中，executor 启动阶段，采用了定时器来触发“过了一段时间”这个事件。如下所示：

```
(defn setup-ticks! [worker executor-data]
  (let [storm-conf (:storm-conf executor-data)
        tick-time-secs (storm-conf TOPOLOGY-TICK-TUPLE-FREQ-SECS)
        receive-queue (:receive-queue executor-data)
        context (:worker-context executor-data)]
    (when tick-time-secs
      (if (or (system-id? (:component-id executor-data))
              (and (= false (storm-conf TOPOLOGY-ENABLE-MESSAGE-TIMEOUTS))
                   (= :spout (:type executor-data))))
          (log-message "Timeouts disabled for executor " (:component-id executor-data) ":" (:executor-id executor-data))
          (schedule-recurring
            (:user-timer worker)
            tick-time-secs
            tick-time-secs
            (fn []
              (disruptor/publish
                receive-queue
                [[nil (TupleImpl. context [tick-time-secs] Constants/SYSTEM_TASK_ID Constants/SYSTEM_TICK_STREAM_ID)
                  )))))))))
```

之前的博文中，已经详细分析了这些基础设施的关系，不理解的童鞋可以翻看前面的文章。每隔一段时间，就会触发这样一个事件，当流的下游的 bolt 收到一个这样的事件时，就可以选择是增量计数还是将结果聚合并发送到流中。bolt 如何判断收到的 tuple 表示的是“tick”呢？负责管理 bolt 的 executor 线程，从其订阅的消息队列消费消息时，会调用到 bolt 的 execute 方法，那么，可以在 execute 中这样判断：

```
public static boolean isTick(Tuple tuple) {
    return tuple != null
        && Constants.SYSTEM_COMPONENT_ID .equals(tuple.getSourceComponent())
        && Constants.SYSTEM_TICK_STREAM_ID.equals(tuple.getSourceStreamId());
}
```

Topology

```

String spoutId = "wordGenerator";
String counterId = "counter";
String intermediateRankerId = "intermediateRanker";
String totalRankerId = "finalRanker";
// 这里，假设TestWordSpout就是我们发送话题tuple的源
builder.setSpout(spoutId, new TestWordSpout(), 5);
// RollingCountBolt的时间窗口为9秒钟，每3秒发送一次统计结果到下游
builder.setBolt(counterId, new RollingCountBolt(9, 3), 4).fieldsGrouping(spoutId, new F
// IntermediateRankingsBolt，将完成部分聚合，统计出top-n的话题
builder.setBolt(intermediateRankerId, new IntermediateRankingsBolt(TOP_N), 4).fieldsGro
    "obj"));
// TotalRankingsBolt，将完成完整聚合，统计出top-n的话题
builder.setBolt(totalRankerId, new TotalRankingsBolt(TOP_N)).globalGrouping(intermediat
```

```

上面的topology设计如下：

! [] (./img/18-4-3.png)

将聚合计算与时间结合起来

前文，我们叙述了**tick**事件，回调中会触发**bolt**的**execute**方法，那可以这么做：

```

RollingCountBolt: @Override public void execute(Tuple tuple) { if (TupleUtils.isTick(tuple))
{ LOG.debug("Received tick tuple, triggering emit of current window counts"); // tick 来了,
将时间窗口内的统计结果发送，并让窗口滚动 emitCurrentWindowCounts(); } else { // 常规
tuple, 对话题计数即可 countObjAndAck(tuple); } }
// obj 即为话题，增加一个计数 count++ // 注意，这里的速度基本取决于流的速度，
可能每秒百万，也可能每秒几十。 // 内存不足？ bolt 可以 scale-out. private void countOb
jAndAck(Tuple tuple) { Object obj = tuple.getValue(0); counter.incrementCount(obj); collec
tor.ack(tuple); }

```

```

IntermediateRankingsBolt & TotalRankingsBolt:
    public final void execute(Tuple tuple, BasicOutputCollector collector) {
        if (TupleUtils.isTick(tuple)) {
            getLogger().debug("Received tick tuple, triggering emit of current rankings");
            // 将聚合并排序的结果发送到下游
            emitRankings(collector);
        }
        else {
            // 聚合并排序
            updateRankingsWithTuple(tuple);
        }
    }
}

```

其中，IntermediateRankingsBolt 和 TotalRankingsBolt 的聚合排序方法略有不同：

IntermediateRankingsBolt 的聚合排序方法：

```

@Override
void updateRankingsWithTuple(Tuple tuple) {
    // 这一步，将话题、话题出现的次数提取出来
    Rankable rankable = RankableObjectWithFields.from(tuple);
    // 这一步，将话题出现的次数进行聚合，然后重排序所有话题
    super.getRankings().updateWith(rankable);
}

```

TotalRankingsBolt 的聚合排序方法：

```

@Override
void updateRankingsWithTuple(Tuple tuple) {
    // 提出来自IntermediateRankingsBolt的中间结果
    Rankings rankingsToBeMerged = (Rankings) tuple.getValue(0);
    // 聚合并排序
    super.getRankings().updateWith(rankingsToBeMerged);
    // 去0，节约内存
    super.getRankings().pruneZeroCounts();
}

```

而重排序方法比较简单粗暴，因为只求前 N 个，N 不会很大：

```
private void rerank() {
```

```

Collections.sort(rankedItems);
Collections.reverse(rankedItems);
}

```

### 结语

下图可能就是我们想要的结果，我们完成了  $t_0 - t_1$  时刻之间的热点话题统计。

## 5. 18.5 如何进行离线计算？

## 6. 18.6 如何使用分布式框架提高模型训练速度？

## 7. 18.7 深度学习分布式计算框架如何在移动互联网中应用？

## 8. 18.8 如何在个性化推荐中应用深度学习分布式框架？

## 9. 18.9 如何评价个性化推荐系统的效果？

**9.1. 18.9.1 准确率与召回率 (Precision & Recall).** 准确率和召回率是广泛用于信息检索和统计学分类领域的两个度量值，用来评价结果的质量。其中精度是检索出相关文档数与检索出的文档总数的比率，衡量的是检索系统的查准率；召回率是指检索出的相关文档数和文档库中所有的相关文档数的比率，衡量的是检索系统的查全率。

一般来说，Precision 就是检索出来的条目（比如：文档、网页等）有多少是准确的，Recall 就是所有准确的条目有多少被检索出来了。

正确率、召回率和 F 值是在鱼龙混杂的环境中，选出目标的重要评价指标。不妨看看这些指标的定义先：

正确率 = 提取出的正确信息条数 / 提取出的信息条数

召回率 = 提取出的正确信息条数 / 样本中的信息条数

两者取值在 0 和 1 之间，数值越接近 1，查准率或查全率就越高。

F 值 = 正确率 \* 召回率 \* 2 / (正确率 + 召回率) (F 值即为正确率和召回率的调和平均值)

不妨举这样一个例子：某池塘有 1400 条鲤鱼，300 只虾，300 只鳖。现在以捕鲤鱼为目的。撒一大网，逮着了 700 条鲤鱼，200 只虾，100 只鳖。那么，这些指标分别如下：

正确率 =  $700 / (700 + 200 + 100) = 70\%$

召回率 =  $700 / 1400 = 50\%$

F 值 =  $70\% * 50\% * 2 / (70\% + 50\%) = 58.3\%$

不妨看看如果把池子里的所有的鲤鱼、虾和鳖都一网打尽，这些指标又有何变化：

b 越大，表示查准率的权重越大。

**9.4. 18.9.4 平均正确率 (Average Precision).** 平均正确率表示不同查全率的点上的正确率的平均。







## 第十八章后端架构选型及应用场景

### 1. 18.1 为什么需要分布式计算？

在这个数据爆炸的时代，产生的数据量不断地在攀升，从 GB,TB,PB,ZB. 挖掘其中数据的价值也是企业在不断地追求的终极目标。但是要想对海量的数据进行挖掘，首先要考虑的就是海量数据的存储问题，比如 Tb 量级的数据。

谈到数据的存储，则不得不说的是磁盘的数据读写速度问题。早在上个世纪 90 年代初期，普通硬盘的可以存储的容量大概是 1G 左右，硬盘的读取速度大概为 4.4MB/s. 读取一张硬盘大概需要 5 分钟时间，但是如今硬盘的容量都在 1TB 左右了，相比扩展了近千倍。但是硬盘的读取速度大概是 100MB/s。读完一个硬盘所需要的时间大概是 2.5 个小时。所以如果是基于 TB 级别的数据进行分析的话，光硬盘读取完数据都要好几天了，更谈不上计算分析了。那么该如何处理大数据的存储，计算分析呢？

一个很简单的减少数据读写时间的方法就是同时从多个硬盘上读写数据，比如，如果我们有 100 个硬盘，每个硬盘存储 1% 的数据，并行读取，那么不到两分钟就可以完成之前需要 2.5 小时的数据读写任务了。这就是大数据中的分布式存储的模型。当然实现分布式存储还需要解决很多问题，比如硬件故障的问题，使用多台主机进行分布式存储时，若主机故障，会出现数据丢失的问题，所以有了副本机制：系统中保存数据的副本。一旦有系统发生故障，就可以使用另外的副本进行替换（著名的 RAID 冗余磁盘阵列就是按这个原理实现的）。其次比如一个很大的文件如何进行拆分存储，读取拆分以后的文件如何进行校验都是要考虑的问题。比如我们使用 Hadoop 中的 HDFS 也面临这个问题，只是框架给我们实现了这些问题的解决办法，开发中开发者不用考虑这些问题，底层框架已经实现了封装。

同样假如有一个 10TB 的文件，我们要统计其中某个关键字的出现次数，传统的做法是遍历整个文件，然后统计出关键字的出现次数，这样效率会特别特别低。基于分布式存储以后，数据被分布式存储在不同的服务器上，那么我们就可以使用分布式计算框架（比如 MapReduce,Spark 等）来进行并行计算（或者说是分布式计算），即：每个服务器上分别统计自己存储的数据中关键字出现的次数，最后进行一次汇总，那么假如数据分布在 100 台服务器上，即同时 100 台服务器同时进行关键字统计工作，效率一下子可以提高几十倍。

Github 地址: <https://github.com/apache/mahout>

**2.4. 18.2.4 Spark MLlib.** MLlib(Machine Learnig lib) 【4】是 Spark 对常用的机器学习算法的实现库，同时包括相关的测试和数据生成器。

MLlib 是 MLBase 一部分，其中 MLBase 分为四部分：MLlib、MLI、ML Optimizer 和 MLRuntime。

- ML Optimizer 会选择它认为最适合的已经在内部实现好了的机器学习算法和相关参数，来处理用户输入的数据，并返回模型或别的帮助分析的结果；
- MLI 是一个进行特征抽取和高级 ML 编程抽象的算法实现的 API 或平台；
- MLlib 是 Spark 实现一些常见的机器学习算法和实用程序，包括分类、回归、聚类、协同过滤、降维以及底层优化，该算法可以进行可扩充；
- MLRuntime 基于 Spark 计算框架，将 Spark 的分布式计算应用到机器学习领域。

MLlib 主要包含三个部分：

- 底层基础：包括 Spark 的运行库、矩阵库和向量库
- 算法库：包含广义线性模型、推荐系统、聚类、决策树和评估的算法
- 实用程序：包括测试数据的生成、外部数据的读入等功能

### 架构图

MLlib 目前支持 4 种常见的机器学习问题：分类、回归、聚类和协同过滤，MLlib 在 Spark 整个生态系统中的位置如图下图所示。

### 生态系统位置

**2.5. 18.2.5 Ray.** Ray 【5】是加州大学伯克利分校实时智能安全执行实验室 (RISE-Lab) 的研究人员针对机器学习领域开发的一种新的分布式计算框架，该框架旨在让基于 Python 的机器学习和深度学习工作负载能够实时执行，并具有类似消息传递接口 (MPI) 的性能和细粒度。

增强学习的场景，按照原理定义，因为没有预先可用的静态标签信息，所以通常需要引入实际的目标系统（为了加快训练，往往是目标系统的模拟环境）来获取反馈信息，用做损失/收益判断，进而完成整个训练过程的闭环反馈。典型的步骤是通过观察特定目标系统的状态，收集反馈信息，判断收益，用这些信息来调整参数，训练模型，并根据新的训练结果产出可用于调整目标系统的行为 Action，输出到目标系统，进而影响目标系统状态变化，完成闭环，如此反复迭代，最终目标是追求某种收益的最大化（比如对 AlphoGo 来说，收益是赢得一盘围棋的比赛）。

在这个过程中，一方面，模拟目标系统，收集状态和反馈信息，判断收益，训练参数，生成 Action 等等行为可能涉及大量的任务和计算（为了选择最佳 Action，可能要并发模拟众多可能的行为）。而这些行为本身可能也是千差万别的异构的任务，任务执行的时间也可能长短不一，执行过程有些可能要求同步，也有些可能更适合异步。

另一方面，整个任务流程的 DAG 图也可能是动态变化的，系统往往可能需要根据前一个环节的结果，调整下一个环节的行为参数或者流程。这种调整，可能是目标系统的需要（比如在自动驾驶过程中遇到行人了，那么我们可能需要模拟计算刹车的距离来判断该采取的行动是刹车还是拐弯，而平时可能不需要这个环节），也可能是增强学习特定训练算法的需要（比如根据多个并行训练的模型的当前收益，调整模型超参数，替换模型等等）。

此外，由于所涉及到的目标系统可能是具体的，现实物理世界中的系统，所以对时效性也可能是有强要求的。举个例子，比如你想要实现的系统是用来控制机器人行走，或者是用来打视频游戏的。那么整个闭环反馈流程就需要在特定的时间限制内完成（比如毫秒级别）。

总结来说，就是增强学习的场景，对分布式计算框架的任务调度延迟，吞吐量和动态修改 DAG 图的能力都可能有很高的要求。按照官方的设计目标，Ray 需要支持异构计算任务，动态计算链路，毫秒级别延迟和每秒调度百万级别任务的能力。

Ray 的目标问题，主要是在类似增强学习这样的场景中所遇到的工程问题。那么增强学习的场景和普通的机器学习，深度学习的场景又有什么不同呢？简单来说，就是对整个处理链路流程的时效性和灵活性有更高的要求。

Ray 框架优点：

- 海量任务调度能力
- 毫秒级别的延迟
- 异构任务的支持
- 任务拓扑图动态修改的能力

Ray 没有采用中心任务调度的方案，而是采用了类似层级（hierarchy）调度的方案，除了一个全局的中心调度服务节点（实际上这个中心调度节点也是可以水平拓展的），任务的调度也可以在具体的执行任务的工作节点上，由本地调度服务来管理和执行。与传统的层级调度方案，至上而下分配调度任务的方式不同的是，Ray 采用了至下而上的调度策略。也就是说，任务调度的发起，并不是先提交给全局的中心调度器统筹规划以后再分发给次级调度器的。而是由任务执行节点直接提交给本地的调度器，本地的调度器如果能满足该任务的调度需求就直接完成调度请求，在无法满足的情况下，才会提交给全局调度器，由全局调度器协调转发给有能力满足需求的另外一个节点上的本地调度器去调度执行。

正如 Spark Streaming 最初的目标一样，它通过丰富的 API 和基于内存的高速计算引擎让用户可以结合流式处理，批处理和交互查询等应用。因此 Spark Streaming 适合一些需要历史数据和实时数据结合分析的应用场合。当然，对于实时性要求不是特别高的应用也能完全胜任。另外通过 RDD 的数据重用机制可以得到更高效的容错处理。

**2.7. 18.2.7 Horovod.** Horovod 【7】是 Uber 开源的又一个深度学习工具，它的发展吸取了 Facebook 「一小时训练 ImageNet 论文」与百度 Ring Allreduce 的优点，可为用户实现分布式训练提供帮助。

Horovod 支持通过用于高性能并行计算的低层次接口 – 消息传递接口 (MPI) 进行分布式模型训练。有了 MPI，就可以利用分布式 Kubernetes 集群来训练 TensorFlow 和 PyTorch 模型。

分布式 TensorFlow 的参数服务器模型 (parameter server paradigm) 通常需要对大量样板代码进行认真的实现。但是 Horovod 仅需要几行。下面是一个分布式 TensorFlow 项目使用 Horovod 的示例：

```
import tensorflow as tf
import horovod.tensorflow as hvd
# Initialize Horovod
hvd.init()
# Pin GPU to be used to process local rank (one GPU per process)
config = tf.ConfigProto()
config.gpu_options.visible_device_list = str(hvd.local_rank())
# Build model...
loss = ...
opt = tf.train.AdagradOptimizer(0.01)
# Add Horovod Distributed Optimizer
opt = hvd.DistributedOptimizer(opt)
# Add hook to broadcast variables from rank 0 to all other processes during
# initialization.
hooks = [hvd.BroadcastGlobalVariablesHook(0)]
# Make training operation
train_op = opt.minimize(loss)
# The MonitoredTrainingSession takes care of session initialization,
# restoring from a checkpoint, saving to a checkpoint, and closing when done
# or an error occurs.
```

```

with tf.train.MonitoredTrainingSession(checkpoint_dir= “/tmp/train_logs” ,
config=config,
hooks=hooks) as mon_sess:
while not mon_sess.should_stop():
# Perform synchronous training.
mon_sess.run(train_op)

```

在该示例中，粗体文字指进行单个 GPU 分布式项目时必须做的改变：

- hvd.init() 初始化 Horovod。
- config.gpu\_options.visible\_device\_list = str(hvd.local\_rank()) 向每个 TensorFlow 流程分配一个 GPU。
- opt=hvd.DistributedOptimizer(opt) 使用 Horovod 优化器包裹每一个常规 TensorFlow 优化器，Horovod 优化器使用 ring-allreduce 平均梯度。
- hvd.BroadcastGlobalVariablesHook(0) 将变量从第一个流程向其他流程传播，以实现一致性初始化。如果该项目无法使用 MonitoredTrainingSession，则用户可以运行 hvd.broadcast\_global\_variables(0)。

之后，可以使用 mpirun 命令使该项目的多个拷贝在多个服务器中运行：

```
$ mpirun -np 16 -x LD_LIBRARY_PATH -H
server1:4,server2:4,server3:4,server4:4 python train.py
```

mpirun 命令向四个节点分布 train.py，然后在每个节点的四个 GPU 上运行 train.py。

Github 地址：<https://github.com/uber/horovod>

**2.8. 18.2.8 BigDL.** BigDL 【9】是一种基于 Apache Spark 的分布式深度学习框架。它可以无缝的直接运行在现有的 Apache Spark 和 Hadoop 集群之上。BigDL 的设计吸取了 Torch 框架许多方面的知识，为深度学习提供了全面的支持；包括数值计算和高级神经网络；借助现有的 Spark 集群来运行深度学习计算，并简化存储在 Hadoop 中的大数据集的数据加载。

BigDL 优点：

- 丰富的深度学习支持。模拟 Torch 之后，BigDL 为深入学习提供全面支持，包括数字计算（通过 Tensor）和高级神经网络；此外，用户可以使用 BigDL 将预先训练好的 Caffe 或 Torch 模型加载到 Spark 程序中。
- 极高的性能。为了实现高性能，BigDL 在每个 Spark 任务中使用英特尔 MKL 和多线程编程。因此，在单节点 Xeon（即与主流 GPU 相当）上，它比开箱即用开源 Caffe, Torch 或 TensorFlow 快了数量级。

### 3.2. 18.3.2 实时流计算过程.

我们以热卖产品的统计为例，看下传统的计算手段：

- 将用户行为、log 等信息清洗后保存在数据库中.
- 将订单信息保存在数据库中.
- 利用触发器或者协程等方式建立本地索引，或者远程的独立索引.
- join 订单信息、订单明细、用户信息、商品信息等等表，聚合统计 20 分钟内热卖产品，并返回 top-10.
- web 或 app 展示.

这是一个假想的场景，但假设你具有处理类似场景的经验，应该会体会到这样一些问题和难处：

- 水平扩展问题（scale-out）显然，如果是一个具有一定规模的电子商务网站，数据量都是很大的。而交易信息因为涉及事务，所以很难直接舍弃关系型数据库的事务能力，迁移到具有更好的 scale-out 能力的 NoSQL 数据库中。

那么，一般都会做 sharding。历史数据还好说，我们可以按日期来归档，并可以通过批处理式的离线计算，将结果缓存起来。但是，这里的要求是 20 分钟内，这很难。

- 性能问题这个问题，和 scale-out 是一致的，假设我们做了 sharding，因为表分散在各个节点中，所以我们需要多次入库，并在业务层做聚合计算。

问题是，20 分钟的时间要求，我们需要入库多少次呢？10 分钟呢？5 分钟呢？实时呢？

而且，业务层也同样面临着单点计算能力的局限，需要水平扩展，那么还需要考虑一致性的问题。所以，到这里一切都显得很复杂。

#### • 业务扩展问题

假设我们不仅仅要处理热卖商品的统计，还要统计广告点击、或者迅速根据用户的访问行为判断用户特征以调整其所见的信息，更加符合用户的潜在需求等，那么业务层将会更加复杂。也许你有更好的办法，但实际上，我们需要的是一种新的认知：这个世界发生的事，是实时的。所以我们需要一种实时计算的模型，而不是批处理模型。我们需要的这种模型，必须能够处理很大的数据，所以要有很好的 scale-out 能力，最好是，我们都不需要考虑太多一致性、复制的问题。

那么，这种计算模型就是实时计算模型，也可以认为是流式计算模型。现在假设我们有了这样的模型，我们就可以愉快地设计新的业务场景：

- 转发最多的微博是什么？
- 最热卖的商品有哪些？
- 大家都在搜索的热点是什么？
- 我们哪个广告，在哪个位置，被点击最多？

或者说，我们可以问：这个世界，在发生什么？

最热的微博话题是什么？我们以一个简单的滑动窗口计数的问题，来揭开所谓实时计算的神秘面纱。假设，我们的业务要求是：统计 20 分钟内最热的 10 个微博话题。

解决这个问题，我们需要考虑：

- 数据源

这里，假设我们的数据，来自微博长连接推送的话题。

- 问题建模

我们认为的话题是 # 号扩起来的话题，最热的话题是此话题出现的次数比其它话题都要多。比如：@foreach\_break : 你好, # 世界 #, 我爱你, # 微博 #。“世界”和“微博”就是话题。

- 计算引擎采用 storm
- 定义时间

时间的定义是一件很难的事情，取决于所需的精度是多少。根据实际，我们一般采用 tick 来表示时刻这一概念。在 storm 的基础设施中，executor 启动阶段，采用了定时器来触发“过了一段时间”这个事件。如下所示：

```
(defn setup-ticks! [worker executor-data]
  (let [storm-conf (:storm-conf executor-data)
        tick-time-secs (storm-conf TOPOLOGY-TICK-TUPLE-FREQ-SECS)
        receive-queue (:receive-queue executor-data)
        context (:worker-context executor-data)]
    (when tick-time-secs
      (if (or (system-id? (:component-id executor-data))
              (and (= false (storm-conf TOPOLOGY-ENABLE-MESSAGE-TIMEOUTS))
                   (= :spout (:type executor-data))))
          (log-message "Timeouts disabled for executor " (:component-id executor-data) ":" (:executer
          (schedule-recurring
            (:user-timer worker)
            tick-time-secs
            tick-time-secs
            (fn []
              (disruptor/publish
                receive-queue
                [[nil (TupleImpl. context [tick-time-secs] Constants/SYSTEM_TASK_ID Constants/SYSTEM
                ])])))))))))))
```

将聚合并计算与时间结合起来前文，我们叙述了 tick 事件，回调中会触发 bolt 的 execute 方法，那可以这么做：

RollingCountBolt:

```

@Override
public void execute(Tuple tuple) {
    if (TupleUtils.isTick(tuple)) {
        LOG.debug("Received tick tuple, triggering emit of current window counts");
        // tick来了，将时间窗口内的统计结果发送，并让窗口滚动
        emitCurrentWindowCounts();
    }
    else {
        // 常规tuple，对话题计数即可
        countObjAndAck(tuple);
    }
}

// obj即为话题，增加一个计数 count++
// 注意，这里的速度基本取决于流的速度，可能每秒百万，也可能每秒几十。
// 内存不足？bolt可以scale-out.
private void countObjAndAck(Tuple tuple) {
    Object obj = tuple.getValue(0);
    counter.incrementCount(obj);
    collector.ack(tuple);
}

// 将统计结果发送到下游
private void emitCurrentWindowCounts() {
    Map<Object, Long> counts = counter.getCountsThenAdvanceWindow();
    int actualWindowLengthInSeconds = lastModifiedTracker.secondsSinceOldestModification();
    lastModifiedTracker.markAsModified();
    if (actualWindowLengthInSeconds != windowLengthInSeconds) {
        LOG.warn(String.format(WINDOW_LENGTH_WARNING_TEMPLATE, actualWindowLengthInSeconds, w));
    }
    emit(counts, actualWindowLengthInSeconds);
}

```

```

IntermediateRankingsBolt & TotalRankingsBolt:
    public final void execute(Tuple tuple, BasicOutputCollector collector) {
        if (TupleUtils.isTick(tuple)) {
            getLogger().debug("Received tick tuple, triggering emit of current rankings");
            // 将聚合并排序的结果发送到下游
            emitRankings(collector);
        }
        else {
            // 聚合并排序
            updateRankingsWithTuple(tuple);
        }
    }
}

```

其中，IntermediateRankingsBolt 和 TotalRankingsBolt 的聚合排序方法略有不同：

IntermediateRankingsBolt 的聚合排序方法：

```

@Override
void updateRankingsWithTuple(Tuple tuple) {
    // 这一步，将话题、话题出现的次数提取出来
    Rankable rankable = RankableObjectWithFields.from(tuple);
    // 这一步，将话题出现的次数进行聚合，然后重排序所有话题
    super.getRankings().updateWith(rankable);
}

```

TotalRankingsBolt 的聚合排序方法：

```

@Override
void updateRankingsWithTuple(Tuple tuple) {
    // 提出来自IntermediateRankingsBolt的中间结果
    Rankings rankingsToBeMerged = (Rankings) tuple.getValue(0);
    // 聚合并排序
    super.getRankings().updateWith(rankingsToBeMerged);
    // 去0，节约内存
    super.getRankings().pruneZeroCounts();
}

```

而重排序方法比较简单粗暴，因为只求前 N 个，N 不会很大：

```
private void rerank() {
```

#### 4. 18.4 如何进行离线计算？

相对于实时计算，有充裕的时间进行运算挖掘

##### 应用场景

- 用户流失预警系统
- 基于用户购买的挽回系统
- 用户特征和规则提取系统
- 数据分析系统
- 用户画像系统

##### 流程

- 数据采集
- 数据预处理
- 数据建模
- ETL
- 数据导出
- 工作流调度

##### 4.1. 18.4.1 数据采集.

Flume 收集服务器日志到 hdfs

type=taildir taildir可以监控一个目录，也可以用一个正则表达式匹配文件名进行实时收集  
taildir=spooldir + exec + 支持断点续传

```
agent1.sources = source1
agent1.sinks = sink1
agent1.channels = channel1
```

```
agent1.sources.source1.type = TAILDIR
agent1.sources.source1.positionFile = /var/log/flume/taildir_position.json
agent1.sources.source1.filegroups = f1 f2
```

# 监控文件内容的改变

```
agent1.sources.source1.filegroups.f1 = /usr/local/nginx/logs/example.log
```

# 监控生成的文件

```
agent1.sources.source1.filegroups.f1 = /usr/local/nginx/logs/*.log.*  
  
agent1.sources.source1.interceptors = i1  
agent1.sources.source1.interceptors.i1.type = host  
agent1.sources.source1.interceptors.i1.hostHeader = hostname  
  
# 配置sink组件为hdfs  
  
agent1.sinks.sink1.type = hdfs  
agent1.sinks.sink1.hdfs.path=  
hdfs://node-1:9000/weblog/flume-collection/%y-%m-%d/%H-%M_%hostname  
  
# 指定文件名前缀  
  
agent1.sinks.sink1.hdfs.filePrefix = access_log  
  
# 指定每批下沉数据的记录条数  
  
agent1.sinks.sink1.hdfs.batchSize= 100  
agent1.sinks.sink1.hdfs.fileType = DataStream  
agent1.sinks.sink1.hdfs.writeFormat =Text  
  
# 指定下沉文件按1G大小滚动  
  
agent1.sinks.sink1.hdfs.rollSize = 1024*1024*1024  
  
# 指定下沉文件按1000000条数滚动  
  
agent1.sinks.sink1.hdfs.rollCount = 1000000  
  
# 指定下沉文件按30分钟滚动  
  
agent1.sinks.sink1.hdfs.rollInterval = 30
```

```
# agent1.sinks.sink1.hdfs.round = true

# agent1.sinks.sink1.hdfs.roundValue = 10

# agent1.sinks.sink1.hdfs.roundUnit = minute

agent1.sinks.sink1.hdfs.useLocalTimeStamp = true

# 使用memory类型channel

agent1.channels.channel1.type = memory
agent1.channels.channel1.capacity = 500000
agent1.channels.channel1.transactionCapacity = 600

# Bind the source and sink to the channel

agent1.sources.source1.channels = channel1
agent1.sinks.sink1.channel = channel1
```

#### 4.2. 18.4.2 数据预处理.

数据预处理编程技巧

- 对于本次分析无利用的数据通常采用逻辑删除建立标记位通过 01 或者 true false 表示数据是否有效
- 对于最后一个字段不固定的情况可以采用动态拼接的方式
- 静态资源过滤
  - js css img (静态数据) 只关心真正请求页面的 (index.html)
  - data (动态数据)
- 在 mr 中, 如果涉及小且频繁使用的数据, 如何优化?
- 每次都从数据库查询效率极低
- 可以通过数据结构保存在内存中方便查询一般在 setup 方法中进行 初始化操作
- 关于 mr 程序输出文件名
  - part-r-00000 表示是 reducetask 的输出

- part-m-00000 表示是 maptask 的输出

### 4.3. 18.4.3 数据建模.

- 维度建模

专门适用于 OLAP 的设计模式存在着两种类型的表：事实表维度表

- 事实表：主题的客观度量能够以记录主题为准信息多不精准
- 维度表：看问题分析问题的角度信息精但是不全可跟事实表关系
- 维度建模三种常见模型
- 星型模型一个事实表带多个维度表维度之间没关系数仓发展建立初期（一个主题）
- 雪花模型一个事实表带多个维度表维度之间可以继续关系维度不利于维护少用
- 星座模型多个事实表带多个维度有些维度可以共用数仓发展后期（多个主题）

不管什么模型，在数仓中，一切有利于数据分析即可为，不用考虑数据冗余性和其他设计规范。

- 模块设计-维度建模

在本项目中，因为分析主题只有一个（网站流量日志），所有采用星型模型事实表—> 对应清洗完之后的数据维度表—> 来自于提前通过工具生成维度表范围要横跨事实表分析维度点击流模型属于业务模型数据既不是事实表也不是维度表是为了后续计算某些业务指标方便而由业务指定

- 宽表：为了分析，把原来表中某些字段属性提取出来，构成新的字段也称之为明细表

窄表：没有扩宽的表原始表宽表数据来自于窄表 insert（宽）+select（窄）总结：hive 中，有几种方式可以创建出带有数据的表？

- create+load data 创建表加载数据（内部表）
- create +external +location 创建外部表指定数据路径
- create+insert+select 表的数据来自于后面查询语句返回的结果
- create+select 创建的表结构和数据来自于后面的查询语句

```
# -- hive内置解析url的函数
```

```
parse_url_tuple (url,host path,query,queryvalue)
```

```
# -- 通常用于把后面的表挂接在左边的表之上 返回成为一个新表
```

```
a LATERAL VIEW b
LATERAL VIEW
```

```
create table t_ods_tmp_referurl as SELECT a.* ,b.* FROM ods_weblog_origin a L
```

- group by 语法限制

```
“‘ select count(*) as pvs from ods_weblog_detail t where datestr=‘20130918’
group by t.hour
select t.hour,count(*) as pvs from ods_weblog_detail t where datestr=‘20130918’
group by t.hour
# - 在有 group by 的语句中，出现在 select 后面的字段要么是分组的字段要么是被聚合函数包围的字段。解决： select t.day,t.hour,count(*) as pvs from
ods_weblog_detail t where datestr=‘20130918’ group by t.day,t.hour; “‘
```

#### 4.4. 18.4.4 ETL.

##### 宽表生成

- 生成 ods+url 解析表

```
create table t_ods_tmp_referurl as
SELECT a.* ,b.* 
FROM ods_weblog_origin a
LATERAL VIEW parse_url_tuple(regexp_replace(http_referer, "\\"", ""), 'HOST', 'PATH',
```

- 生成 ods+url+date 解析表

```
create table t_ods_tmp_detail as
select b.* ,substring(time_local,0,10) as daystr,
substring(time_local,12) as tmstr,
substring(time_local,6,2) as month,
substring(time_local,9,2) as day,
substring(time_local,11,3) as hour
From t_ods_tmp_referurl b;
```

- 综合

```
create table ods_weblog_detail(
valid          string, --有效标识
```

```

remote_addr      string, --来源IP
remote_user      string, --用户标识
time_local       string, --访问完整时间
daystr           string, --访问日期
timestr          string, --访问时间
month            string, --访问月
day               string, --访问日
hour              string, --访问时
request          string, --请求的url
status            string, --响应码
body_bytes_sent  string, --传输字节数
http_referer     string, --来源url
ref_host          string, --来源的host
ref_path          string, --来源的路径
ref_query         string, --来源参数query
ref_query_id     string, --来源参数query的值
http_user_agent   string --客户终端标识
)
partitioned by(datestr string);

insert into table ods_weblog_detail partition(datestr='20130918')
select c.valid,c.remote_addr,c.remote_user,c.time_local,
substring(c.time_local,0,10) as daystr,
substring(c.time_local,12) as tmstr,
substring(c.time_local,6,2) as month,
substring(c.time_local,9,2) as day,
substring(c.time_local,12,2) as hour,
c.request,c.status,c.body_bytes_sent,c.http_referer,c.ref_host,c.ref_path,c.ref_query,c.ref_query_id
from
(select a.* ,b.*
from ods_weblog_origin a
LATERAL view
parse_url_tuple(regexp_replace(a.http_referer,"\"",""),'HOST','PATH','QUERY','QUERY_ID')b

```

**DML 分析**

- 计算该处理批次（一天）中的各小时 pvs

```
select
t.month,t.day,t.hour,count(*)
from ods_weblog_detail t
where t.datestr='20130918'
group by t.month,t.day,t.hour;
```

- 计算每天的 pvs

```
select t.month,t.day,count(*) from ods_weblog_detail t where t.datestr='20130918' gr
```

```
select a.month,a.day,sum(a.pvs)
from
(
    select
        t.month as month,t.day as day,t.hour as hour,count(*) as pvs
    from ods_weblog_detail t
    where t.datestr='20130918'
    group by t.month,t.day,t.hour
) a
group by a.month,a.day;
```

统计每小时各来访 url 产生的 pvs

```
select
t.day,t.hour,t.http_referer,t.ref_host,count(*)
from ods_weblog_detail t
where datestr='20130918'
group by t.day,t.hour,t.http_referer,t.ref_host
having t.ref_host is not null;
```

- 统计每小时各来访 host 的产生的 pv 数并排序

```
select
t.month,t.day,t.hour,t.ref_host,count(*) as pvs
from ods_weblog_detail t
where datestr='20130918'
group by t.month,t.day,t.hour,t.ref_host
having t.ref_host is not null
```

```
order by t.hour asc ,pvs desc;
```

按照时间维度，统计一天内各小时产生最多pvs的来源 (host) topN(分组Top)

```
select
```

```
a.month,a.day,a.hour,a.host,a.pvs,a.rmp
```

```
from
```

```
(
```

```
select
```

```
t.month as month,t.day as day,t.hour as hour,t.ref_host as host,count(*) as pvs,
```

```
row_number()over(partition by concat(t.month,t.day,t.hour) order by pvs desc) rmp
```

```
from ods_weblog_detail t
```

```
where datestr='20130918'
```

```
group by t.month,t.day,t.hour,t.ref_host
```

```
having t.ref_host is not null
```

```
order by hour asc ,pvs desc
```

```
)a
```

```
where a.rmp < 4;
```

统计今日所有来访者平均请求的页面数。

```
select count(*)/count(distinct remote_addr) from ods_weblog_detail where datestr='20130918'
```

```
select
```

```
sum(a.pvs)/count(a.ip)
```

```
from
```

```
(
```

```
select
```

```
t.remote_addr as ip,count(*) as pvs
```

```
from ods_weblog_detail t
```

```
where t.datestr='20130918'
```

```
group by t.remote_addr
```

```
) a;
```

- 统计每日最热门的页面 top10

```
select
```

```
t.request,count(*) as counts
```

```
from ods_weblog_detail t
```

```
where datestr='20130918'  
group by t.request  
order by counts desc  
limit 10;  
    • 每日新访客  
  
select  
today.ip  
from  
(  
    select distinct t.remote_addr as ip  
    from ods_weblog_detail t  
) today  
left join history  
on today.ip=history.ip  
where history.ip is null;  
    • 查询今日所有回头访客及其访问次数 (session)  
  
select  
remote_addr,count(session) as cs  
from ods_click_stream_visit  
where datestr='20130918'  
group by remote_addr  
having cs >1;  
    • 人均访问频次  
  
select  
count(session)/count(distinct remote_addr)  
from ods_click_stream_visit  
where datestr='20130918';  
    • 级联查询自 join  
  
select  
rn.step as rnstep,rn.numbs as rnnumbs,rr.step as rrstep,rr.numbs as rrnumbs  
from dw_oute_nums rn  
inner join  
dw_oute_nums rr;
```

```

# -- 绝对转化

select
a.rrstep,a.rrnumbs/a.rnnumbs
from
(
    select
        rn.step as rnstep,rn.numbs as rnnumbs,rr.step as rrstep,rr.numbs as rrnumbs
        from dw_oute_nums rn
        inner join dw_oute_nums rr
)a
where a.rnstep='step1';

# -- 相对转化

select
tmp.rrstep as step,tmp.rrnumbs/tmp.rnnumbs as leakage_rate
from
(
    select rn.step as rnstep,rn.numbs as rnnumbs,rr.step as rrstep,rr.numbs as rrnumbs from
        inner join
        dw_oute_nums rr
) tmp
where cast(substr(tmp.rnstep,5,1) as int)=cast(substr(tmp.rrstep,5,1) as int)-1;

```

#### 4.5. 18.4.5 数据导出.

Sqoop 可以对 HDFS 文件进行导入导出到关系型数据库 Sqoop 工作机制是将导入或导出命令翻译成 mapreduce 程序来实现。在翻译出的 mapreduce 中主要是对 inputformat 和 outputformat 进行定制 sqoop 实际生产环境中关于 mysql 地址尽量不要使用: localhost 可用 ip 或者域名代替

##### 导入

- mysql—>hdfs 导入的文件分隔符为逗号
- mysql—>hive

- 需要先复制表结构到 hive 再向表中导入数据
- 导入的文件分隔符为 ‘01’
- sqoop 中增量导入的判断是通过上次导入到某个列的某个值来标识的，这个值由用户自己维护，一般企业中选择不重复且自增长的主键最多，自增长的时间也可以。

```
# 导入mysql表到hdfs
```

```
bin/sqoop import \
--connect jdbc:mysql://node-1:3306/userdb \
--username root \
--password 123 \
--target-dir /sqoopresult \
--table emp --m 1
```

```
# 支持条件导入数据
```

```
bin/sqoop import \
--connect jdbc:mysql://node-1:3306/userdb \
--username root \
--password 123 \
--where "id > 1202" \
--target-dir /sqoopresult/t1 \
--table emp --m 1
```

```
# 将关系型数据的表结构复制到hive中
```

```
bin/sqoop create-hive-table \
--connect jdbc:mysql://node-1:3306/userdb \
--table emp_add \
--username root \
--password 123 \
--hive-table default.emp_add_sp
```

```
# 从关系数据库导入文件到hive中
```

```
bin/sqoop import \
--connect jdbc:mysql://node-1:3306/userdb \
--username root \
--password 123 \
--table emp_add \
--hive-table default.emp_add_sp \
--hive-import \
--m 1
```

# 增量导入

```
bin/sqoop import \
--connect jdbc:mysql://node-1:3306/userdb \
--username root \
--password 123 \
--table emp_add \
--target-dir '/user/hive/warehouse/emp_add_sp' \
--incremental append \
--check-column id \
--last-value 1205 \
--fields-terminated-by '\001' \
--m 1
```

## 导出

- hdfs 导出到 mysql
- 要先在 mysql 中手动创建对应的表结构

# hdfs文件导出到mysql

```
bin/sqoop export \
--connect jdbc:mysql://node-1:3306/userdb \
--username root \
--password 123 \
--table employee \
```

```
--export-dir /hivedata/employee.txt \
--fields-terminated-by '\001'
```

#### 4.6. 18.4.6 工作流调度.

azkaban 工作流程

- 配置 job 文件（注意文件的第一行头信息）
- 把 job 配置连同其他资源一起打成.zip 压缩包
- 页面上创建工程 project
- 上传.zip 压缩包
- execute/schedule

### 5. 18.5 如何设计一个人机交互系统？

**5.1. 18.5.1 什么是人机交互系统？** 人机交互系统 (Human-computer interaction, 简称 HCI) 是研究人与计算机之间通过相互理解的交流与通信，在最大程度上为人们完成信息管理，服务和处理等功能，使计算机真正成为人们工作学习的和谐助手的一门技术科学。

通俗来讲，就是指人与计算机之间使用某种对话语言，以一定的交互方式，为完成确定任务的人与计算机之间的信息交换过程。

目前工业界落地的产品包括阿里巴巴集团的云小蜜、天猫精灵；百度的 UNIT、小度；小米的小爱同学；京东的叮咚智能音箱等。

#### 【产品图】

人机交互发展阶段大致可分为四个阶段

- 第一代人机交互技术：基于键盘和字符显示器的交互阶段
- 第二代人机交互技术：基于鼠标和图形显示器的交互阶段
- 第三代人机交互技术：基于多媒体技术的交互阶段
- 第四代人机交互技术：人机自然交互与通信

本章节重点介绍第四代人机交互技术，传统人机交互模型主要组成部分包括：

1. 多模态输入/输出：多模态输入/输出是第四代人机交互与通信的主要标志之一。多模态输入包括键盘、鼠标、文字、语音、手势、表情、注视等多种输入方式；而多模态输出包括文字、图形、语音、手势、表情等多种交互信息
2. 智能接口代理：智能接口代理是实现人与计算机交互的媒介
3. 视觉获取：视觉系统主要用于实时获取外部视觉信息
4. 视觉合成：使人机交互能够在一个仿真或虚拟的环境中进行，仿佛现实世界中人与人之间的交互

**5.5. 18.5.5 什么是指代消解？如何指代消解？** 指代消解，广义上说，就是在篇章中确定代词指向哪个名词短语的问题。按照指向，可以分为回指和预指。回指就是代词的先行语在代词前面，预指就是代词的先行语在代词后面。按照指代的类型可以分为三类：人称代词、指示代词、有定描述、省略、部分 - 整体指代、普通名词短语。

1. 主谓关系 SBV (subject-verb)
2. 状中结构 ADV (adverbial)
3. 核心 HED (head)
4. 动宾关系 VOB (verb-object)
5. 标点 WP
6. nhd: 疾病, v: 动词, vn: 名动词, nbx: 自定义专有名词, rzv: 谓词性指示代词

例子：

输入1：感冒/nhd 可以/v 投保/vn 相互保/nbx 吗/y ?

输入2：那/rzv 癌症/nhd 呢/y ?

输出：癌症可以投保相互保吗？

输入1：相互保很好

输入2：这个产品比e生保好在哪？

输出：相互保比e生保好在哪？

输入1：相互保、e生保都是医疗险吗

输入2：前者能报销啥

输出：相互保能报销啥

输入1：相互保、e生保都是医疗险吗

输入2：第一种能报销啥

输出：相互保能报销啥

输入3：第二种呢

输出：e生保能报销啥

分析：

感冒 --(SBV)--> 投保

可以 --(ADV)--> 投保

投保 --(HED)--> ##核心##

相互保 --(VOB)--> 投保

吗 --(RAD)--> 投保

github 地址: <https://github.com/nlpaueb/deep-relevance-ranking>

- **N-grams**

**\*\*N-grams\*\***是机器学习中**NLP**处理中的一个较为重要的语言模型，常用来做句子相似度比较，模

如果你是一个玩**LOL**的人，那么当我说“正方形打野”、“你是真的皮”，“你皮任你皮”这些词时，

**N-grams** 正是基于这样的想法，它的第一个特点是某个词的出现依赖于其他若干个词，第二个特点是我们获得的信息越多，预测越准确。我想说，我们每个人的大脑中都有一个 N-gram 模型，而且是在不断完善和训练的。我们的见识与经历，都在丰富着我们的阅历，增强着我们的联想能力。**N-grams** 模型是一种语言模型 (Language Model, LM)，语言模型是一个基于概率的判别模型，它的输入是一句话（单词的顺序序列），输出是这句话的概率，即这些单词的联合概率 (joint probability)。

**\*\*N-grams 中的概率计算\*\***

假设我们有一个有 n 个词组成的句子  $S = (w_1, w_2, \dots, w_n)$ ，如何衡量它的概率呢？让我们假设，

$$p(S) = p(w_1 w_2 \dots w_n) = p(w_1)p(w_2|w_1)\dots p(w_n|w_{n-1} \dots w_1)$$

这个衡量方法有两个缺陷：

- **参数空间过大**，概率  $p(w_n|w_{n-1} \dots w_1)$  的参数有  $O(n)$  个。
- **数据稀疏严重**，词同时出现的情况可能没有，组合阶数高时尤其明显。

为了解决第一个问题，我们引入**马尔科夫假设 (Markov Assumption)**：一个词的出现仅与它之前的若干个词有关。

$$p(w_1 w_n) = p(w_i w_{i+1} \dots w_n)$$

如果一个词的出现仅依赖于它前面出现的一个词，那么我们就称之为 **Bi-gram**:  $p(S) = p(w_1 w_2 \dots w_n) = p(w_1)p(w_2|w_1)\dots p(w_n|w_{n-1})$

```
w 2
w n
)=p(w 1
)p(w 2
w 1
) p(w n
w n-1
```

)  
*\* \*Tri - gram \* \**

$p(S) = p(w_1$

$w_2$

$w_n$

$) = p(w_1$

$) p(w_2$

$w_1$

$) p(w_n$

$w_{n-1}$

$w_{n-2}$

) N-gram 的 N\$ 可以取很高, 然而现实中一般 bi-gram 和 tri-gram 就够用了。

那么, 如何计算其中的每一项条件概率  $p(w_n w_{n-1} w_2 w_1)$  答案是极大似然估计 (Maximum Likelihood Estimation, MLE), 即数频数: \$\$ p(w\_n

$w_{n-1}$

$) = C(w_{n-1}$

$) C(w_{n-1}$

$w_n$

$) \$\$$

$\$\$ p(w_n$

$w_{n-1}$

$w_{n-2}$

$) = C(w_{n-2}$

$w_{n-1}$

$) C(w_{n-2}$

$w_{n-1}$

$w_n$

)

$\$\$$

$\$\$ p(w_n$

$w_{n-1}$

$w_2$

$w_1$

$) = C(w_1$

容易统计，“I”出现了 3 次，“I am”出现了 2 次，因此能计算概率：

$$p(amI) = 32$$

同理，还能计算出如下概率：

$$p(I) = 0.67 \\ p(do I) = 0.33 \\ p(Sam am) = 0.5 \\ p(not do) = 1 \\ p(Sam) = 0.5 \\ p(like not) = 1$$

**5.10. 18.5.10 如何评估人机交互系统的效果？** 人机交互系统可通过有效问题数量、推荐 Top1 答案的准确率、推荐 Top3 答案的准确率、有效问题响应的准确率、知识覆盖率为指标衡量其效果。

| 类型     | 指标        | 收集方式   |
|--------|-----------|--------|
| 问答评估指标 | 有效问题数     | 日志抽样统计 |
| 问答评估指标 | Top1 准确率  | 日志抽样统计 |
| 问答评估指标 | Top3 准确率  | 日志抽样统计 |
| 问答评估指标 | 有效问题响应准确率 | 日志抽样统计 |
| 问答评估指标 | 知识覆盖率     | 日志抽样统计 |

| 类型   | 指标      | 计算公式                                                             |
|------|---------|------------------------------------------------------------------|
| 总体指标 | 请求量占比   | FAQ、任务型、寒暄机器人的请求量/总请求量                                           |
| 总体指标 | 总用户量    | FAQ、任务型、寒暄引擎的独立访客数                                               |
| 总体指标 | 解决率     | (点击解决 button 量 + 不点击) / FAQ 咨询重量 = 1 - 点击未解决 button 量 / FAQ 咨询重量 |
| 总体指标 | 未解决率    | 点击解决 button / FAQ 总咨询量 = 1 - 解决率                                 |
| 总体指标 | 转人工率    | FAQ 转人工量 / FAQ 总咨询量                                              |
| 总体指标 | 任务达成率   | 业务办理成功量 / 业务办理请求量                                                |
| 总体指标 | 平均会话时长  | FAQ、任务、向量对话时长 / 总用户数                                             |
| 总体指标 | 平均对话轮数  | 总对话论数 / 总用户数                                                     |
| 总体指标 | 断点问题分析  | 转空客问题                                                            |
| 总体指标 | 无应答问题分析 | 机器人返回兜底话术的问题                                                     |
| 总体指标 | 热点问题分析  | 问题咨询总量 TopN                                                      |

## 6. 18.6 如何设计个性化推荐系统？

**6.1. 18.6.1 什么是个性化推荐系统？** 个性化推荐系统就是根据用户的历史，社交关系，兴趣点，上下文环境等信息去判断用户当前需要或潜在感兴趣的内容的一类应用。

大数据时代，我们的生活的方方面面都出现了信息过载的问题：电子商务、电影或者视频网站、个性化音乐网络电台、社交网络、个性化阅读、基于位置的服务、个性化邮件、个性

– 对曝光不足的做平滑，曝光充分的影响不大

#### 6.6. 18.6.6 用户画像.

- 用户标签
- 统计方法
- 用户 feeds 内行为，标签计数（点击率），缺点：无法加入更多特征，不方便后续优化
- 基于机器学习的方法
- 对用户长期兴趣建模
- LR 模型
- 用户标签作为特征

#### 6.7. 18.6.7 GBDT 粗排.

- 为什么需要粗排？
- 快速筛选高质量的候选集
- 方便利用在线实时反馈特征
- 如何做粗排的特征设计？
- 特征要相对稠密
- 如何选择合适的算法模型？
- lightgbm
- xgboost
- lightgbm 比 xgboost 速度更快；在线预测时，线程更安全

#### 6.8. 18.6.8 在线 FM 精排.

- 为什么需要在线学习？
- feeds 内容更新快
- 用户兴趣会随时间变化
- 排序模型需要快速反应用户的兴趣变化
- FM 模型

\$\$

\$\$

$$() = 1 / (1 + \exp())$$

- 采用 FTRL(Follow The Regularized Leader) 更新模型
- 算法概述

- 算法特性及优缺点

在线学习，实时性高；可以处理大规模稀疏数据；有大规模模型参数训练能力；根据不同的特征特征学习率。

FTRL-Proximal 工程实现上的 tricks：

- 1.saving memory

方案1) Poisson Inclusion: 对某一维度特征所来的训练样本，以p的概率接受并更新模型

方案2) Bloom Filter Inclusion: 用bloom filter从概率上做某一特征出现k次才更新。

2. 浮点数重新编码

1) 特征权重不需要用32bit或64bit的浮点数存储，存储浪费空间。

2) 16bit encoding，但是要注意处理rounding技术对regret带来的影响(注：python可以尝

3. 训练若干相似 model

1) 对同一份训练数据序列，同时训练多个相似的model。

2) 这些model有各自独享的一些feature，也有一些共享的feature。

3) 出发点：有的特征维度可以是各个模型独享的，而有的各个模型共享的特征，可以用同

- 4.Single Value Structure

1) 多个model公用一个feature存储（例如放到cbase或redis中），各个model都更新这个共

2) 对于某一个model，对于他所训练的特征向量的某一维，直接计算一个迭代结果并与旧值

5. 使用正负样本的数目来计算梯度的和（所有的 model 具有同样的 N 和 P）

%E5%90%8E%E7%AB%AF%E6%9E%B6%E6%9E%84%E9%80%89%E5%9E%8B%E5%8F%8

6-2-1-1.jpg %E5%90%8E%E7%AB%AF%E6%9E%B6%E6%9E%84%E9%80%89%E5%9E%8B%E5%8

6-2-1-1.bb

- 6.subsampling Training Data

1) 在实际中，CTR远小于50%，所以正样本更加有价值。通过对训练数据集进行subampling

2) 正样本全部采（至少有一个广告被点击的query数据），负样本使用一个比例r采样（完

3) 解决办法：训练的时候，对样本再乘一个权重。权重直接乘到loss上面，从而梯度也会

- 适合场景

点击率模型

- 案例

<https://www.kaggle.com/jiweiliu/ftrl-starter-code/output>

[https://github.com/Angel-ML/angel/blob/master/docs/algo/ftrl\\_lr\\_spark.md](https://github.com/Angel-ML/angel/blob/master/docs/algo/ftrl_lr_spark.md)

- 如何选择精排特征？

- 新增特征需保证已有特征索引不变

推荐任务是一个极端的多类分类问题。这个预测问题的实质，是基于用户 (U) 和语境 (C)，在给定的时间  $t$  精确地从库 (V) 中上百万的视频类 ( $i$ ) 中，对特定的视频观看 ( $W_t$ ) 情况进行分类。

