

## 第一章 数学基础

### 1.1 向量和矩阵

- 1.1.1 标量、向量、矩阵、张量之间的联系
- 1.1.2 张量与矩阵的区别
- 1.1.3 矩阵和向量相乘结果
- 1.1.4 向量和矩阵的范数归纳
- 1.1.5 如何判断一个矩阵为正定

### 1.2 导数和偏导数

- 1.2.1 导数偏导计算
- 1.2.2 导数和偏导数有什么区别？

### 1.3 特征值和特征向量

- 1.3.1 特征值分解与特征向量
- 1.3.2 奇异值与特征值有什么关系

### 1.4 概率分布与随机变量

- 1.4.1 机器学习为什么要使用概率
- 1.4.2 变量与随机变量有什么区别
- 1.4.3 随机变量与概率分布的联系
- 1.4.4 离散型随机变量和概率质量函数
- 1.4.5 连续型随机变量和概率密度函数
- 1.4.6 举例理解条件概率
- 1.4.7 联合概率与边缘概率联系区别
- 1.4.8 条件概率的链式法则
- 1.4.9 独立性和条件独立性

### 1.5 常见概率分布

- 1.5.1 Bernoulli分布
- 1.5.2 高斯分布
- 1.5.3 何时采用正态分布
- 1.5.4 指数分布
- 1.5.5 Laplace 分布
- 1.5.6 Dirac分布和经验分布

### 1.6 期望、方差、协方差、相关系数

- 1.6.1 期望
- 1.6.2 方差
- 1.6.3 协方差
- 1.6.4 相关系数

### 参考文献

## 第二章 机器学习基础

### 2.1 基本概念

- 2.1.1 大话理解机器学习本质
- 2.1.2 什么是神经网络
- 2.1.3 各种常见算法图示
- 2.1.4 计算图的导数计算
- 2.1.5 理解局部最优与全局最优
- 2.1.5 大数据与深度学习之间的关系

### 2.2 机器学习学习方式

- 2.2.1 监督学习
- 2.2.2 非监督式学习
- 2.2.3 半监督式学习
- 2.2.4 弱监督学习
- 2.2.5 监督学习有哪些步骤

### 2.8 分类算法

- 2.8.1 常用分类算法的优缺点？
- 2.8.2 分类算法的评估方法

- 2.8.3 正确率能很好的评估分类算法吗
- 2.8.4 什么样的分类器是最好的
- 2.9 逻辑回归
  - 2.9.1 回归划分
  - 2.9.2 逻辑回归适用性
  - 2.9.3 逻辑回归与朴素贝叶斯有什么区别
  - 2.9.4 线性回归与逻辑回归的区别
- 2.10 代价函数
  - 2.10.1 为什么需要代价函数
  - 2.10.2 代价函数作用原理
  - 2.10.3 为什么代价函数要非负
  - 2.10.4 常见代价函数
  - 2.10.5 为什么用交叉熵代替二次代价函数
- 2.11 损失函数
  - 2.11.1 什么是损失函数
  - 2.11.2 常见的损失函数
  - 2.11.3 逻辑回归为什么使用对数损失函数
  - 2.11.4 对数损失函数是如何度量损失的
- 2.12 梯度下降
  - 2.12.1 机器学习中为什么需要梯度下降
  - 2.12.2 梯度下降法缺点
  - 2.12.3 梯度下降法直观理解
  - 2.12.4 梯度下降法算法描述
  - 2.12.5 如何对梯度下降法进行调优
  - 2.12.6 随机梯度和批量梯度区别
  - 2.12.7 各种梯度下降法性能比较
- 2.14 线性判别分析 (LDA)
  - 2.14.1 LDA思想总结
  - 2.14.2 图解LDA核心思想
  - 2.14.3 二类LDA算法原理
  - 2.14.4 LDA算法流程总结
  - 2.14.5 LDA和PCA区别
  - 2.14.6 LDA优缺点
- 2.15 主成分分析 (PCA)
  - 2.15.1 主成分分析 (PCA) 思想总结
  - 2.15.2 图解PCA核心思想
  - 2.15.3 PCA算法推理
  - 2.15.4 PCA算法流程总结
  - 2.15.5 PCA算法主要优缺点
  - 2.15.6 降维的必要性及目的
  - 2.15.7 KPCA与PCA的区别
- 2.16 模型评估
  - 2.16.1 模型评估常用方法?
  - 2.16.2 误差、偏差和方差有什么区别和联系
  - 2.16.3 经验误差与泛化误差
  - 2.16.4 图解欠拟合、过拟合
  - 2.16.5 如何解决过拟合与欠拟合
  - 2.16.6 交叉验证的主要作用
  - 2.16.7 理解k折交叉验证
  - 2.16.8 混淆矩阵
  - 2.16.9 错误率及精度
  - 2.16.10 查准率与查全率
  - 2.16.11 ROC与AUC
  - 2.16.12 如何画ROC曲线
  - 2.16.13 如何计算TPR, FPR
  - 2.16.14 如何计算AUC
  - 2.16.15 为什么使用Roc和Auc评价分类器
  - 2.16.16 直观理解AUC

- 2.16.17 代价敏感错误率与代价曲线
  - 2.16.18 模型有哪些比较检验方法
  - 2.16.19 为什么使用标准差
  - 2.16.20 类别不平衡产生原因
  - 2.16.21 常见的类别不平衡问题解决方法
  - 2.17 决策树
    - 2.17.1 决策树的基本原理
    - 2.17.2 决策树的三要素?
    - 2.17.3 决策树学习基本算法
    - 2.17.4 决策树算法优缺点
    - 2.17.5 熵的概念以及理解
    - 2.17.6 信息增益的理解
    - 2.17.7 剪枝处理的作用及策略
  - 2.18 支持向量机
    - 2.18.1 什么是支持向量机
    - 2.18.2 支持向量机能解决哪些问题
    - 2.18.3 核函数特点及其作用
    - 2.18.4 SVM为什么引入对偶问题
    - 2.18.5 如何理解SVM中的对偶问题
    - 2.18.7 常见的核函数有哪些
    - 2.18.9 SVM主要特点
    - 2.18.10 SVM主要缺点
    - 2.18.11 逻辑回归与SVM的异同
  - 2.19 贝叶斯分类器
    - 2.19.1 图解极大似然估计
    - 2.19.2 极大似然估计原理
    - 2.19.3 贝叶斯分类器基本原理
    - 2.19.4 朴素贝叶斯分类器
    - 2.19.5 举例理解朴素贝叶斯分类器
    - 2.19.6 半朴素贝叶斯分类器
  - 2.20 EM算法
    - 2.20.1 EM算法基本思想
    - 2.20.2 EM算法推导
    - 2.20.3 图解EM算法
    - 2.20.4 EM算法流程
  - 2.21 降维和聚类
    - 2.21.1 图解为什么会产生维数灾难
    - 2.21.2 怎样避免维数灾难
    - 2.21.3 聚类和降维有什么区别与联系
    - 2.21.4 有哪些聚类算法优劣衡量标准
    - 2.21.5 聚类和分类有什么区别
    - 2.21.6 不同聚类算法特点性能比较
    - 2.21.7 四种常用聚类方法之比较
    - 2.21.8 k-means聚类算法
    - 2.21.9 层次聚类算法
    - 2.21.10 SOM聚类算法
    - 2.21.11 FCM聚类算法
    - 2.21.12 四种聚类算法试验
- 参考文献
- ### 第三章 深度学习基础
- 3.1 基本概念
    - 3.1.1 神经网络组成?
    - 3.1.2 神经网络有哪些常用模型结构?
    - 3.1.3 如何选择深度学习开发平台?
    - 3.1.4 为什么使用深层表示?
    - 3.1.5 为什么深层神经网络难以训练?
    - 3.1.6 深度学习和机器学习有什么不同?
  - 3.2 网络操作与计算

- 3.2.1 前向传播与反向传播?
- 3.2.2 如何计算神经网络的输出?
- 3.2.3 如何计算卷积神经网络输出值?
- 3.2.4 如何计算 Pooling 层输出值输出值?
- 3.2.5 实例理解反向传播
- 3.2.6 神经网络更“深”有什么意义?
- 3.3 超参数
  - 3.3.1 什么是超参数?
  - 3.3.2 如何寻找超参数的最优值?
  - 3.3.3 超参数搜索一般过程?
- 3.4 激活函数
  - 3.4.1 为什么需要非线性激活函数?
  - 3.4.2 常见的激活函数及图像
  - 3.4.3 常见激活函数的导数计算?
  - 3.4.4 激活函数有哪些性质?
  - 3.4.5 如何选择激活函数?
  - 3.4.6 使用 ReLU 激活函数的优点?
  - 3.4.7 什么时候可以用线性激活函数?
  - 3.4.8 怎样理解 Relu (< 0 时) 是非线性激活函数?
  - 3.4.9 Softmax 定义及作用
  - 3.4.10 Softmax 函数如何应用于多分类?
  - 3.4.11 交叉熵代价函数定义及其求导推导
  - 3.4.12 为什么 Tanh 收敛速度比 Sigmoid 快?
- 3.5 Batch\_Size
  - 3.5.1 为什么需要 Batch\_Size?
  - 3.5.2 Batch\_Size 值的选择
  - 3.5.3 在合理范围内, 增大Batch\_Size 有何好处?
  - 3.5.4 盲目增大 Batch\_Size 有何坏处?
  - 3.5.5 调节 Batch\_Size 对训练效果影响到底如何?
- 3.6 归一化
  - 3.6.1 归一化含义?
  - 3.6.2 为什么要归一化?
  - 3.6.3 为什么归一化能提高求解最优解速度?
  - 3.6.4 3D 图解未归一化
  - 3.6.5 归一化有哪些类型?
  - 3.6.6 局部响应归一化作用
  - 3.6.7 理解局部响应归一化
  - 3.6.8 什么是批归一化 (Batch Normalization)
  - 3.6.9 批归一化 (BN) 算法的优点
  - 3.6.10 批归一化 (BN) 算法流程
  - 3.6.11 批归一化和群组归一化比较
  - 3.6.12 Weight Normalization 和 Batch Normalization 比较
  - 3.6.13 Batch Normalization 在什么时候用比较合适?
- 3.7 预训练与微调(fine tuning)
  - 3.7.1 为什么无监督预训练可以帮助深度学习?
  - 3.7.2 什么是模型微调 fine tuning
  - 3.7.3 微调时候网络参数是否更新?
  - 3.7.4 fine-tuning 模型的三种状态
- 3.8 权重偏差初始化
  - 3.8.1 全都初始化为 0
  - 3.8.2 全都初始化为同样的值
  - 3.8.3 初始化为小的随机数
  - 3.8.4 用 [Math Processing Error] 校准方差
  - 3.8.5 稀疏初始化(Sparse Initialazation)
  - 3.8.6 初始化偏差
- 3.9 学习率
  - 3.9.1 学习率的作用
  - 3.9.2 学习率衰减常用参数有哪些

- 3.9.3 分段常数衰减
- 3.9.4 指数衰减
- 3.9.5 自然指数衰减
- 3.9.6 多项式衰减
- 3.9.7 余弦衰减
- 3.12 Dropout 系列问题
  - 3.12.1 为什么要正则化?
  - 3.12.2 为什么正则化有利于预防过拟合?
  - 3.12.3 理解dropout正则化
  - 3.12.4 dropout率的选择
  - 3.12.5 dropout有什么缺点?
- 3.13 深度学习中常用的数据增强方法?
- 3.14 如何理解 Internal Covariate Shift?

参考文献

## 第四章 经典网络解读

- 4.1 LeNet-5
  - 4.1.1 模型介绍
  - 4.1.2 模型结构
  - 4.1.3 模型特性
- 4.2 AlexNet
  - 4.2.1 模型介绍
  - 4.2.2 模型结构
  - 4.2.3 模型特性
- 4.3 ZFNet
  - 4.3.1 模型介绍
  - 4.3.2 模型结构
  - 4.3.3 模型特性
- 4.4 Network in Network
  - 4.4.1 模型介绍
  - 4.4.2 模型结构
  - 4.4.3 模型特点
- 4.5 VGGNet
  - 4.5.1 模型介绍
  - 4.5.2 模型结构
  - 4.5.3 模型特性
- 4.6 GoogLeNet
  - 4.6.1 模型介绍
  - 4.6.2 模型结构
  - 4.6.3 模型特性
- Restnet
- Densenet
- 4.7 为什么现在的CNN模型都是在GoogleNet、VGGNet或者AlexNet上调整的?

参考文献

## 第五章 卷积神经网络 (CNN)

- 5.1 卷积神经网络的组成层
  - 5.1.1 输入层
  - 5.1.2 卷积层
  - 5.1.3 激活层
  - 5.1.4 池化层
  - 5.1.5 全连接层
- 5.2 卷积在图像中有什么直观作用
- 5.3 卷积层有哪些基本参数?
- 5.4 卷积核有什么类型?
- 5.5 二维卷积与三维卷积有什么区别?
- 5.7 有哪些池化方法?
- 5.8 [Math Processing Error]卷积作用?
- 5.9 卷积层和池化层有什么区别?
- 5.10 卷积核是否一定越大越好?

- 5.11 每层卷积是否只能用一种尺寸的卷积核?
- 5.12 怎样才能减少卷积层参数量?
- 5.13 在进行卷积操作时,必须同时考虑通道和区域吗?
- 5.14 采用宽卷积的好处有什么?
- 5.15 理解转置卷积与棋盘效应
  - 5.15.1 标准卷积
  - 5.15.2 转置卷积
  - 5.15.3 棋盘效应
- 5.16 卷积神经网络的参数设置
- 5.17 提高卷积神经网络的泛化能力
- 5.18 卷积神经网络在不同领域的应用
  - 5.18.1 联系
  - 5.18.2 区别
- 5.19 卷积神经网络凸显共性的方法?
  - 5.19.1 局部连接
  - 5.19.2 权值共享
  - 5.19.3 池化操作
- 5.20 全连接、局部连接、全卷积与局部卷积
- 5.21 局部卷积的应用
- 5.22 NetVLAD池化 (贡献者: 熊楚原-中国人民大学)

参考文献

## 第六章 循环神经网络(RNN)

- 6.1 为什么需要RNN?
- 6.2 图解RNN基本结构
  - 6.2.1 基本的单层网络结构
  - 6.2.2 图解经典RNN结构
  - 6.2.3 vector-to-sequence结构
  - 6.2.4 sequence-to-vector结构
  - 6.2.5 Encoder-Decoder结构
  - 6.2.6 以上三种结构各有怎样的应用场景
  - 6.2.7 图解RNN中的Attention机制
- 6.3 RNNs典型特点?
- 6.4 CNN和RNN的区别 ?
- 6.5 RNNs和FNNs有什么区别?
- 6.6 RNNs训练和传统ANN训练异同点?
- 6.7 为什么RNN 训练的时候Loss波动很大
- 6.8 标准RNN前向输出流程
- 6.9 BPTT算法推导
- 6.9 RNN中为什么会出现梯度消失?
- 6.10 如何解决RNN中的梯度消失问题?
- 6.11 LSTM
  - 6.11.1 LSTM的产生原因
  - 6.11.2 图解标准RNN和LSTM的区别
  - 6.11.3 LSTM核心思想图解
  - 6.11.4 LSTM流行的变体
- 6.12 LSTMs与GRUs的区别
- 6.13 RNNs在NLP中典型应用?
- 6.13 常见的RNNs扩展和改进模型
  - 6.13.1 Simple RNNs(SRNs)
  - 6.13.2 Bidirectional RNNs
  - 6.13.3 Deep RNNs
  - 6.13.4 Echo State Networks (ESNs)
  - 6.13.4 Gated Recurrent Unit Recurrent Neural Networks
  - 6.13.5 Bidirectional LSTMs
  - 6.13.6 Stacked LSTMs
  - 6.13.7 Clockwork RNNs(CW-RNNs)
  - 6.13.8 CNN-LSTMs

参考文献

## 第七章 生成对抗网络

### 7.1 GAN基本概念

- 7.1.1 如何通俗理解GAN?
- 7.1.2 GAN的形式化表达
- 7.1.3 GAN的目标函数是什么?
- 7.1.4 GAN的目标函数和交叉熵有什么区别?
- 7.1.5 GAN的Loss为什么降不下去?
- 7.1.6 生成式模型、判别式模型的区别?
- 7.1.7 什么是mode collapsing?
- 7.1.8 如何解决mode collapsing?

### 7.2 GAN的生成能力评价

- 7.2.1 如何客观评价GAN的生成能力?
- 7.2.2 Inception Score
- 7.2.3 Mode Score
- 7.2.4 Kernel MMD (Maximum Mean Discrepancy)
- 7.2.5 Wasserstein distance
- 7.2.6 Fréchet Inception Distance (FID)
- 7.2.7 1-Nearest Neighbor classifier
- 7.2.8 其他评价方法

### 7.3 其他常见的生成式模型有哪些?

- 7.3.1 什么是自回归模型: pixelRNN与pixelCNN?
- 7.3.2 什么是VAE?

### 7.4 GAN的改进与优化

- 7.4.1 如何生成指定类型的图像——条件GAN
- 7.4.2 CNN与GAN——DCGAN
- 7.4.3 如何理解GAN中的输入随机噪声?
- 7.4.4 GAN为什么容易训练崩溃?
- 7.4.5 WGAN如何解决训练崩溃问题?
- 7.4.6 WGAN-GP: 带有梯度正则的WGAN
- 7.4.7 LSGAN
- 7.4.8 如何尽量避免GAN的训练崩溃问题?

### 7.3 GAN的应用 (图像翻译)

- 7.3.1 什么是图像翻译?
- 7.3.2 有监督图像翻译: pix2pix
- 7.3.3 其他图像翻译的tricks
- 7.3.4 如何生成高分辨率图像和高分辨率视频?
- 7.3.5 有监督的图像翻译的缺点?
- 7.3.6 无监督图像翻译: CycleGAN
- 7.3.7 多领域的无监督图像翻译: StarGAN

### 7.4 GAN的应用 (文本生成)

- 7.4.1 GAN为什么不适合文本任务?
- 7.4.2 seqGAN用于文本生成

### 7.5 GAN在其他领域的应用

- 7.5.1 数据增广
- 7.5.2 图像超分辨与图像补全
- 7.5.3 语音领域

## 参考文献

## 第八章 目标检测

### 8.1 基本概念

- 8.1.1 什么是目标检测?
- 8.1.2 目标检测要解决的核心问题?
- 8.1.3 目标检测算法分类?
- 8.1.4 目标检测有哪些应用?

### 8.2 Two Stage目标检测算法

- 8.2.1 R-CNN
- 8.2.2 Fast R-CNN
- 8.2.3 Faster R-CNN
- 8.2.4 R-FCN

8.2.5 FPN
8.2.6 Mask R-CNN
8.3 One Stage目标检测算法
8.3.1 SSD
8.3.2 DSSD
8.3.3 YOLOv1
8.3.4 YOLOv2
8.3.5 YOLO9000
8.3.6 YOLOv3
8.3.7 RetinaNet
8.3.8 RFBNet
8.3.9 M2Det
8.4 人脸检测
8.4.1 目前主要有人脸检测方法分类?
8.4.2 如何检测图片中不同大小的人脸?
8.4.3 如何设定算法检测最小人脸尺寸?
8.4.4 如何定位人脸的位置?
8.4.5 如何通过一个人脸的多个框确定最终人脸框位置?
8.4.6 基于级联卷积神经网络的人脸检测 (Cascade CNN)
8.4.7 基于多任务卷积神经网络的人脸检测 (MTCNN)
8.4.8 Facebox
8.5 目标检测的技巧汇总
8.6 目标检测的常用数据集
8.6.1 PASCAL VOC
8.6.2 MS COCO
8.6.3 Google Open Image
8.6.4 ImageNet
TODO
参考文献

# 第一章 数学基础

深度学习通常又需要哪些数学基础？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、优化理论、信息论等等。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。

## 1.1 向量和矩阵

### 1.1.1 标量、向量、矩阵、张量之间的联系

#### 标量 (scalar)

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。

#### 向量 (vector)

一个向量表示一组有序排列的数。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 $X$ 的第一个元素是 $X_1$ ，第二个元素是 $X_2$ ，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）。

## 矩阵 (matrix)

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 $A$ 。

## 张量 (tensor)

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“A”。张量 $A$ 中坐标为 $(i, j, k)$ 的元素记作 $A_{(i,j,k)}$ 。

## 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。

## 1.1.2 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。
- 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。

## 1.1.3 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：

$$a_{ik} * b_{kj} = c_{ij} \quad (1.3-1)$$

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 $n \times 1$ 的矩阵。

## 1.1.4 向量和矩阵的范数归纳

### 向量的范数(norm)

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：

- 向量的1范数：向量的各个元素的绝对值之和，上述向量 $\vec{a}$ 的1范数结果就是：29。

$$\|\vec{x}\|_1 = \sum_{i=1}^N |x_i| \quad (1)$$

- 向量的2范数：向量的每个元素的平方和再开平方根，上述 $\vec{a}$ 的2范数结果就是：15。

$$\|\vec{x}\|_2 = \sqrt{\sum_{i=1}^N |x_i|^2} \quad (2)$$

- 向量的负无穷范数：向量的所有元素的绝对值中最小的：上述向量 $\vec{a}$ 的负无穷范数结果就是：5。

$$\|\vec{x}\|_{-\infty} = \min |x_i| \quad (3)$$

- 向量的正无穷范数：向量的所有元素的绝对值中最大的：上述向量 $\vec{a}$ 的正无穷范数结果就是：10。

$$\|\vec{x}\|_{+\infty} = \max |x_i| \quad (4)$$

- 向量的p范数：

$$L_p = \|\vec{x}\|_p = \sqrt[p]{\sum_{i=1}^N |x_i|^p} \quad (5)$$

## 矩阵的范数

定义一个矩阵 $A = [-1, 2, -3; 4, -6, 6]$ 。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

$$\|A\|_p := \sup_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p} \quad (6)$$

当向量取不同范数时, 相应得到了不同的矩阵范数。

- 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), 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。

$$\|A\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^m |a_{ij}| \quad (7)$$

- 矩阵的2范数:** 矩阵 $A^T A$ 的最大特征值开平方根, 上述矩阵 $A$ 的2范数得到的最终结果是: 10.0623。

$$\|A\|_2 = \sqrt{\lambda_{\max}(A^T A)} \quad (8)$$

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

- 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), 上述矩阵 $A$ 的行范数先得到[6, 16], 再取最大的最终结果就是: 16。

$$\|A\|_\infty = \max_{1 \leq i \leq m} \sum_{j=1}^n |a_{ij}| \quad (9)$$

- 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。
- 矩阵的L0范数:** 矩阵的非0元素的个数, 通常用它来表示稀疏, L0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。
- 矩阵的L1范数:** 矩阵中的每个元素绝对值之和, 它是L0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。
- 矩阵的F范数:** 矩阵的各个元素平方之和再开平方根, 它通常也叫做矩阵的L2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。

$$\|A\|_F = \sqrt{\left(\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2\right)} \quad (10)$$

- 矩阵的L21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L1范数 (也可认为是向量的1范数), 很容易看出它是介于L1和L2之间的一种范数, 上述矩阵 $A$ 最终结果就是: 17.1559。

- 矩阵的 p 范数

$$\|A\|_p = \sqrt[p]{\left( \sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^p \right)}$$
 (11)

### 1.1.5 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。

## 1.2 导数和偏导数

### 1.2.1 导数偏导计算

**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

$$v = \frac{s}{t}$$
 (12)

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

$$\bar{v} = \frac{\Delta s}{\Delta t} = \frac{s(t_0 + \Delta t) - s(t_0)}{\Delta t}$$
 (13)

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：

$$v(t_0) = \lim_{\Delta t \rightarrow 0} \bar{v} = \lim_{\Delta t \rightarrow 0} \frac{\Delta s}{\Delta t} = \lim_{\Delta t \rightarrow 0} \frac{s(t_0 + \Delta t) - s(t_0)}{\Delta t}$$
 (14)

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

$$\lim_{\Delta x \rightarrow 0} \frac{\Delta y}{\Delta x} = \lim_{\Delta x \rightarrow 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}$$
 (15)

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

$$\lim_{\Delta x \rightarrow 0} \frac{f(x_0 + \Delta x, y_0) - f(x_0, y_0)}{\Delta x} = A \quad (16)$$

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 $\frac{\partial z}{\partial x}|_{y=y_0}^{x=x_0}$ 或 $\frac{\partial f}{\partial x}|_{y=y_0}^{x=x_0}$ 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。

## 1.2.2 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。

## 1.3 特征值和特征向量

### 1.3.1 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 $v$ 是方阵 $A$ 的特征向量，将一定可以表示成下面的形式：

$$Av = \lambda v \quad (17)$$

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

$$A = Q \sum Q^{-1} \quad (18)$$

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。

### 1.3.2 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：

$$(A^T A)V = \lambda V \quad (19)$$

这里 $V$ 就是上面的右奇异向量，另外还有：

$$\sigma_i = \sqrt{\lambda_i}, u_i = \frac{1}{\sigma_i} A \mu_i \quad (20)$$

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 $m, n$ ) 个的奇异值来近似描述矩阵，即部分奇异值分解：

$$A_{m \times n} \approx U_{m \times r} \sum_{r \times r} V_{r \times n}^T \quad (21)$$

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 $A$ 。

## 1.4 概率分布与随机变量

### 1.4.1 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。

### 1.4.2 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%。

### 1.4.3 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。

#### 1.4.4 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
- $\forall x \in X, 0 \leq P(x) \leq 1$ .
- $\sum_{x \in X} P(x) = 1$ . 我们把这一条性质称之为 归一化的(normalized)

#### 1.4.5 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。
- $\forall x \in X, p(x) \geq 0$ . 注意, 我们并不要求  $p(x) \leq 1$ , 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。
- $\int p(x)dx = 1$ , 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  $\delta x$  的无线小的区域内的概率为  $p(x)\delta x$ . 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  $\int_a^b p(x)dx$ .

#### 1.4.6 举例理解条件概率

条件概率公式如下:

$$P(A|B) = P(A \cap B)/P(B) \quad (22)$$

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。

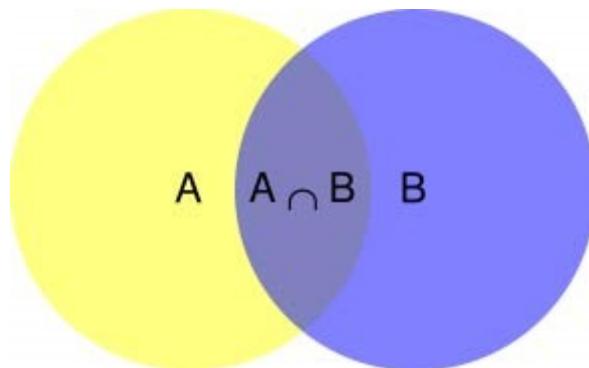


图1.1 条件概率文氏图示意

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：** $P(\text{女}|\text{女}) = P(\text{女女})/P(\text{女})$ ，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 $P(\text{女女})$ 为 $1/4$ ， $P(\text{女}) = 1 - P(\text{男男}) = 3/4$ ，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。

## 1.4.7 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。

## 1.4.8 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

$$P(AB) = P(B|A)P(A) \quad (23)$$

推广

$$P(ABC) = P(C|AB)P(B|A)P(A) \quad (24)$$

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

$$P(A_1 A_2 \dots A_n) = P(A_n | A_1 A_2 \dots A_{n-1})P(A_{n-1} | A_1 A_2 \dots A_{n-2}) \dots P(A_2 | A_1)P(A_1) = P(A_1) \prod_{i=2}^n P(A_i | A_1 A_2 \dots A_{i-1}) \quad (25)$$

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。

## 1.4.9 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 $Z$ ，

$$P(X, Y|Z) \neq P(X|Z)P(Y|Z) \quad (26)$$

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

$$X \perp Y | Z \iff P(X, Y | Z) = P(X|Z)P(Y|Z) \quad (27)$$

$X$ 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。

## 1.5 常见概率分布

### 1.5.1 Bernoulli分布

**Bernoulli分布**是单个二值随机变量分布, 单参数 $\phi \in [0, 1]$ 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:

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

**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 $p_k = 1 - \vec{1}^T \vec{p}$ .

**适用范围:** 伯努利分布适合对**离散型**随机变量建模.

### 1.5.2 高斯分布

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

$$N(x; \mu, \sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x - \mu)^2\right) \quad (28)$$

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:

$$N(x; \mu, \sigma^2) = \sqrt{\frac{1}{2\pi}} \exp\left(-\frac{1}{2}x^2\right) \quad (29)$$

对概率密度函数高效求值:

$$N(x; \mu, \beta^{-1}) = \sqrt{\frac{\beta}{2\pi}} \exp\left(-\frac{1}{2}\beta(x - \mu)^2\right) \quad (30)$$

其中,  $\beta = \frac{1}{\sigma^2}$ 通过参数 $\beta \in (0, \infty)$ 来控制分布精度。

### 1.5.3 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  $\Sigma$ :

$$N(\vec{x}; \vec{\mu}, \Sigma) = \sqrt{\frac{1}{(2\pi)^n \det(\Sigma)}} \exp\left(-\frac{1}{2}(\vec{x} - \vec{\mu})^T \Sigma^{-1} (\vec{x} - \vec{\mu})\right) \quad (31)$$

对多位正态分布概率密度高效求值:

$$N(\vec{x}; \vec{\mu}, \vec{\beta}^{-1}) = \sqrt{\det(\vec{\beta})} (2\pi)^n \exp\left(-\frac{1}{2}(\vec{x} - \vec{\mu})^T \vec{\beta} (\vec{x} - \vec{\mu})\right) \quad (32)$$

此处,  $\vec{\beta}$  是一个精度矩阵。

### 1.5.4 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:

$$p(x; \lambda) = \lambda I_{x \geq 0} \exp(-\lambda x) \quad (33)$$

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。

### 1.5.5 Laplace 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

$$\text{Laplace}(x; \mu; \gamma) = \frac{1}{2\gamma} \exp\left(-\frac{|x - \mu|}{\gamma}\right) \quad (34)$$

### 1.5.6 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. Dirac 分布的狄拉克  $\delta$  函数(也称为**单位脉冲函数**) 定义如下:

$$p(x) = \delta(x - \mu), x \neq \mu \quad (35)$$

$$\int_a^b \delta(x - \mu) dx = 1, a < \mu < b \quad (36)$$

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

$$\hat{p}(\vec{x}) = \frac{1}{m} \sum_{i=1}^m \delta(\vec{x} - \vec{x}^{(i)}) \quad (37)$$

, 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  $\delta$  函数适合对**连续型** 随机变量的经验分布.

## 1.6 期望、方差、协方差、相关系数

## 1.6.1 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:  $E(\sum_{k=1}^n a_k x_k + c) = \sum_{k=1}^n a_k E(x_k) + c$
- 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 $E(xy) = E(x)E(y)$ 。

## 1.6.2 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：

$$Var(x) = E((x - E(x))^2) \quad (38)$$

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,  $Var(ax + by) = a^2 Var(x) + b^2 Var(y)$

## 1.6.3 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：

$$Cov(x, y) = E((x - E(x))(y - E(y))) \quad (39)$$

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:

$$Cov\left(\sum_{i=1}^m a_i x_i, \sum_{j=1}^n b_j y_j\right) = \sum_{i=1}^m \sum_{j=1}^n a_i b_j Cov(x_i y_j) \quad (40)$$

- 3) 特殊情况:

$$Cov(a + bx, c + dy) = bdCov(x, y) \quad (41)$$

## 1.6.4 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：

$$Corr(x, y) = \frac{Cov(x, y)}{\sqrt{Var(x)Var(y)}} \quad (42)$$

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。

## 参考文献

[1] Ian, Goodfellow, Yoshua, Bengio, Aaron...深度学习[M], 人民邮电出版, 2017

[2] 周志华.机器学习[M].清华大学出版社, 2016.

[3] 同济大学数学系.高等数学（第七版）[M], 高等教育出版社, 2014.

[4] 盛骤, 试式干, 潘承毅等编. 概率论与数理统计（第4版）[M], 高等教育出版社, 2008

# 第二章 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。

## 2.1 基本概念

### 2.1.1 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。

### 2.1.2 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。

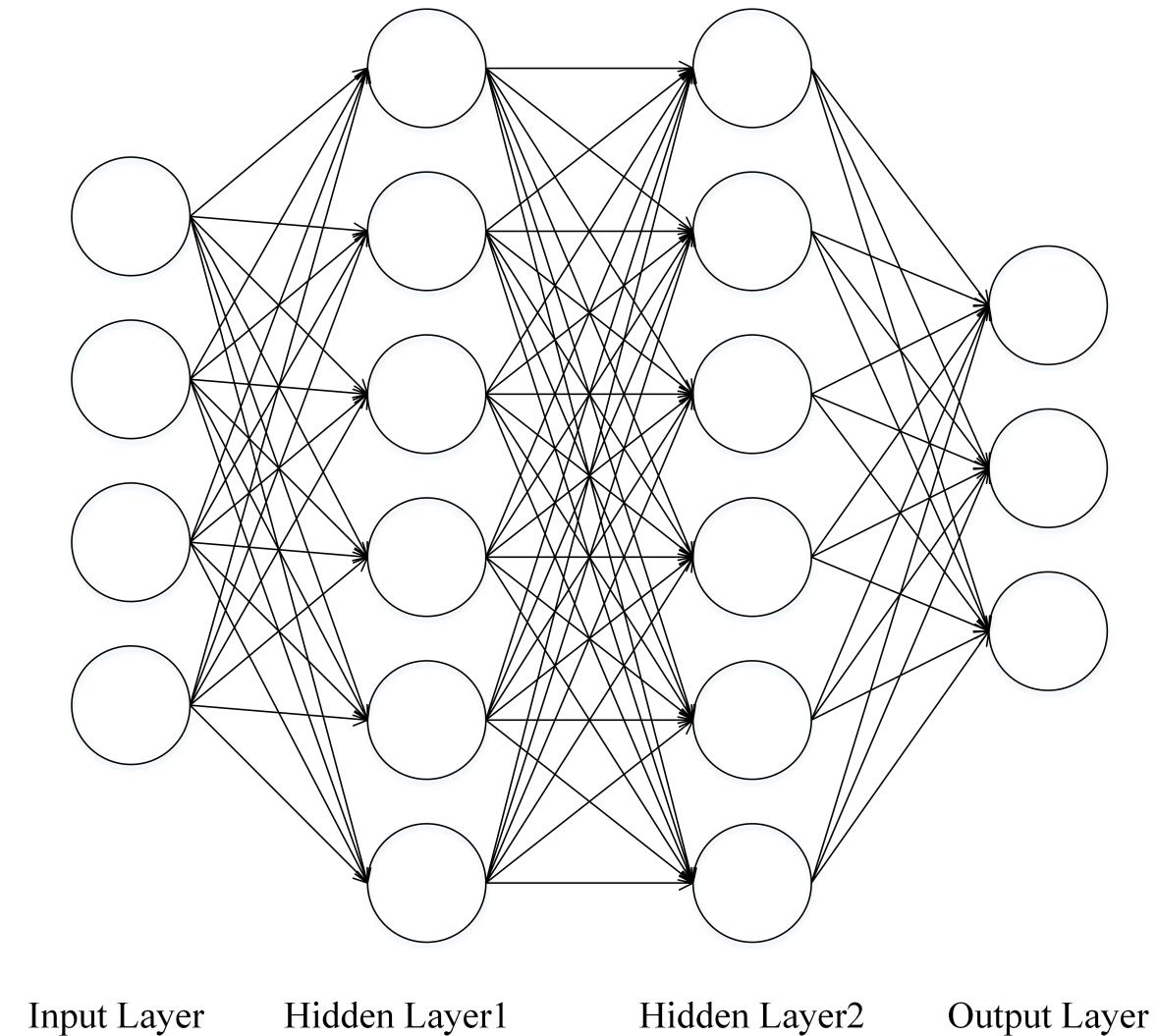
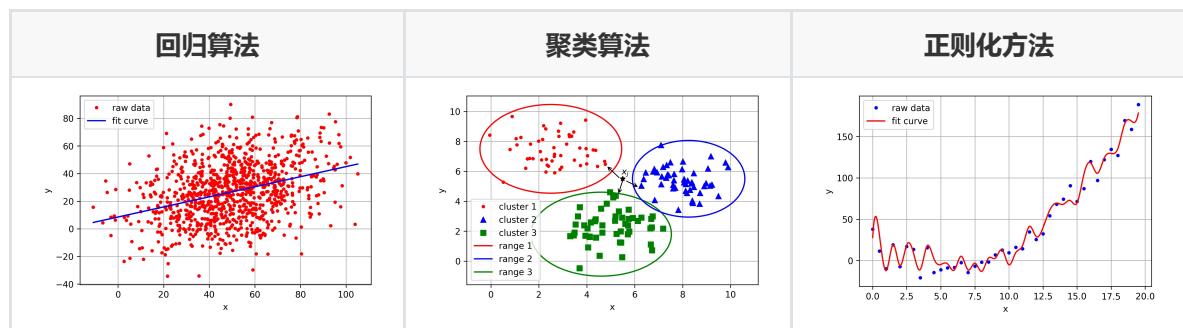


图2-1 神经网络系统

### 2.1.3 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。



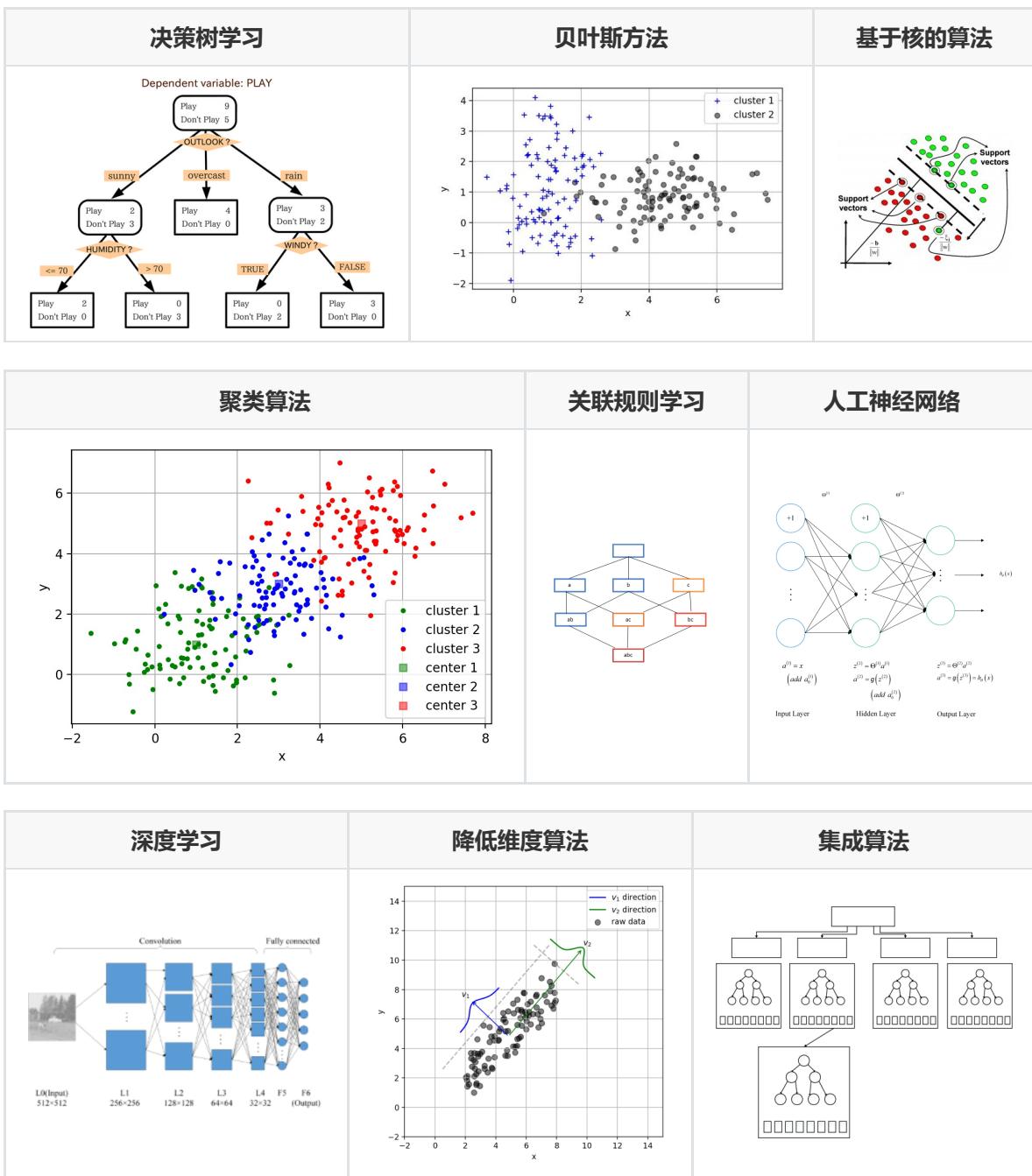


图2-2 各种常见算法图示

## 2.1.4 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

$$\frac{dz}{dt} = \frac{\partial z}{\partial u} \cdot \frac{du}{dt} + \frac{\partial z}{\partial v} \cdot \frac{dv}{dt} \quad (43)$$

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：

$$f(x) = x^2, g(x) = 2x + 1 \quad (44)$$

则：

$$f[g(x)]' = 2[g(x)] \times g'(x) = 2[2x + 1] \times 2 = 8x + 4 \quad (45)$$

## 2.1.5 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寻找到的，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。

## 2.1.5 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。

## 2.2 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。

## 2.2.1 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。

## 2.2.2 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。

## 2.2.3 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。

## 2.2.4 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。

## 2.2.5 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，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结果。

# 2.8 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。

## 2.8.1 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
Bayes 贝叶斯分类法	1) 所需估计的参数少，对于缺失数据不敏感。 2) 有着坚实的数学基础，以及稳定的分类效率。	1) 需要假设属性之间相互独立，这往往并不成立。（喜欢吃番茄、鸡蛋，却不喜欢吃番茄炒蛋）。 2) 需要知道先验概率。 3) 分类决策存在错误率。
Decision Tree决策树	1) 不需要任何领域知识或参数假设。 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，得到可行且效果较好的结果。 5) 能够同时处理数据型和常规性属性。	1) 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) 易于过拟合。 3) 忽略属性之间的相关性。 4) 不支持在线学习。
SVM支持向量机	1) 可以解决小样本下机器学习的问题。 2) 提高泛化性能。 3) 可以解决高维、非线性问题。超高维文本分类仍受欢迎。 4) 避免神经网络结构选择和局部极小的问题。	1) 对缺失数据敏感。 2) 内存消耗大，难以解释。 3) 运行和调参略烦人。
KNN K近邻	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)； 4) 准确度高，对数据没有假设，对outlier不敏感；	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。
Logistic Regression 逻辑回归	1) 速度快。 2) 简单易于理解，直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) 如果想要一个概率框架，动态调整分类阀值。	特征处理复杂。需要归一化和较多的特征工程。
Neural Network 神经网络	1) 分类准确率高。 2) 并行处理能力强。 3) 分布式存储和学习能力强。 4) 鲁棒性较强，不易受噪声影响。	1) 需要大量参数（网络拓扑、阀值、阈值）。 2) 结果难以解释。 3) 训练时间过长。

算法	优点	缺点
Adaboosting	1) adaboost是一种有很高精度的分类器。 2) 可以使用各种方法构建子分类器，Adaboost算法提供的是框架。 3) 当使用简单分类器时，计算出的结果是可以理解的。而且弱分类器构造极其简单。 4) 简单，不用做特征筛选。 5) 不用担心overfitting。	对outlier比较敏感

## 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

		预测类别			
		Yes	No	总计	
实际类别	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	

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。

- 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,  $error\ rate = (FP+FN)/(P+N)$ , 对某一个实例来说, 分对与分错是互斥事件, 所以  $accuracy = 1 - error\ rate$ 。

- 3) 灵敏度 (sensitivity)

$sensitivity = TP/P$ , 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)

$specificity = TN/N$ , 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)

$precision = TP/(TP+FP)$ , 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,  $recall = TP/(TP+FN) = TP/P = sensitivity$ , 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:  $F1 = \frac{2 \times precision \times recall}{precision + recall}$ 。

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F1 (macro-averaging) 两种指标。

- (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, 而微平均F1平等考虑文档集中的每一个文档, 所以它的值受到常见类别的影响比较大。

- 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, 是以灵敏度 (真阳性率) 为纵坐标, 以1减去特异性 (假阳性率) 为横坐标绘制的性能评价曲线。可以将不同模型对同一数据集的ROC曲线绘制在同一笛卡尔坐标系中, ROC曲线越靠近左上角, 说明其对应模型越可靠。也可以通过ROC曲线下面的面积 (Area Under Curve, AUC) 来评价模型, AUC越大, 模型越可靠。

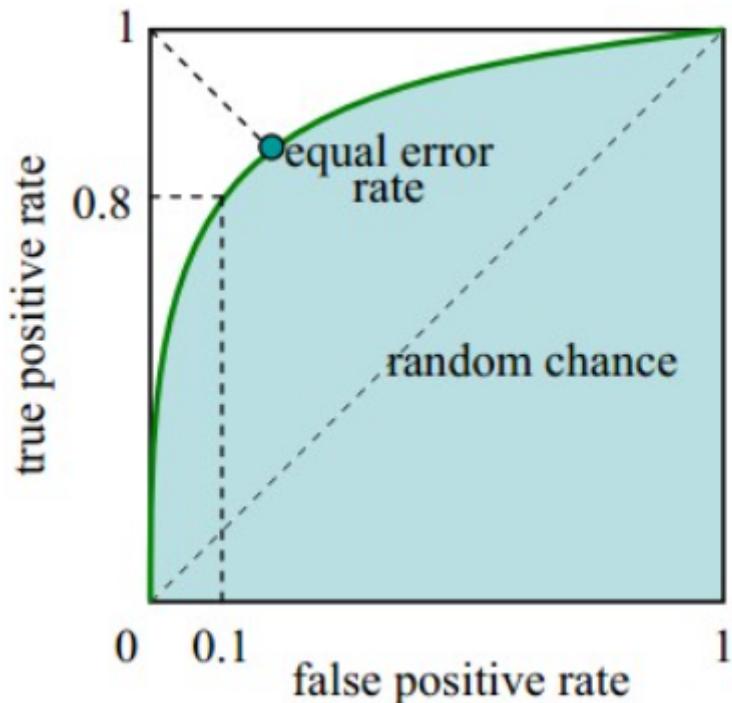


图2-3 ROC曲线

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。

### 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。

### 2.8.4 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。

## 2.9 逻辑回归

### 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。

## 2.9.2 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。

## 2.9.3 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。

## 2.9.4 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。

## 2.10 代价函数

### 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。

### 2.10.2 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。

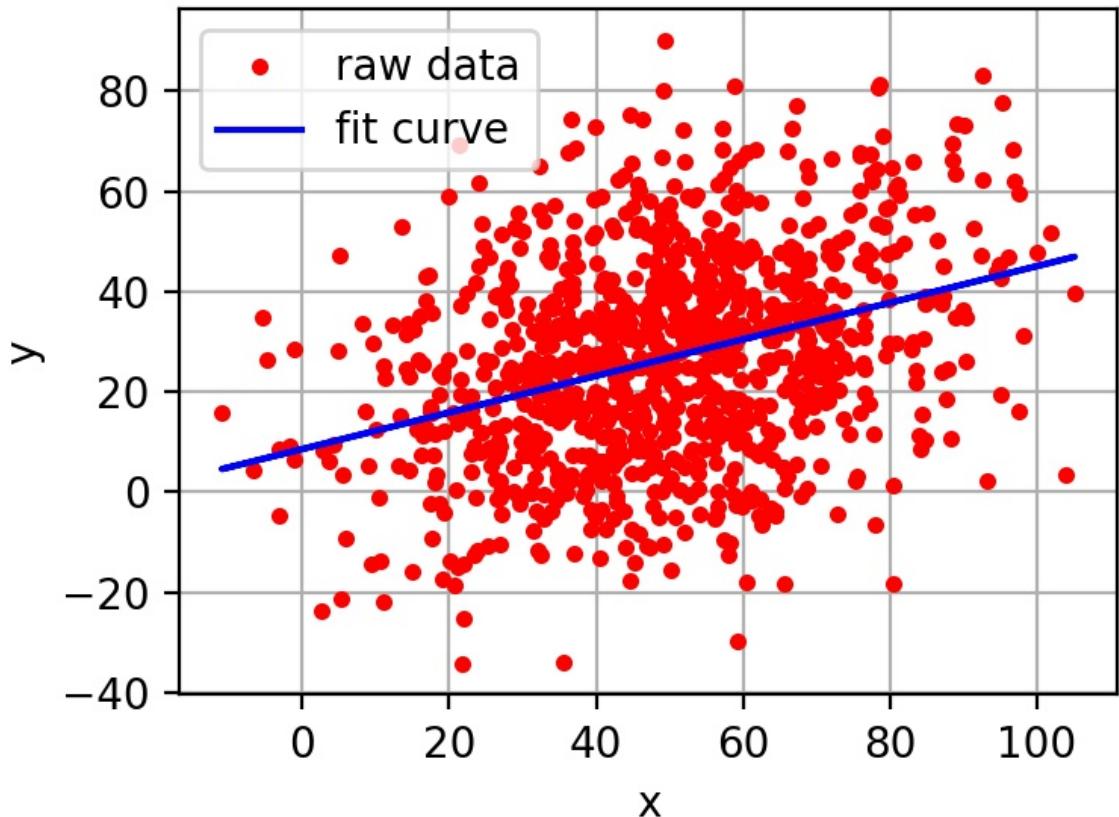


图2-4  $h(x) = A + Bx$ 函数示意图

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：

$$J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)})^2 \quad (46)$$

**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。

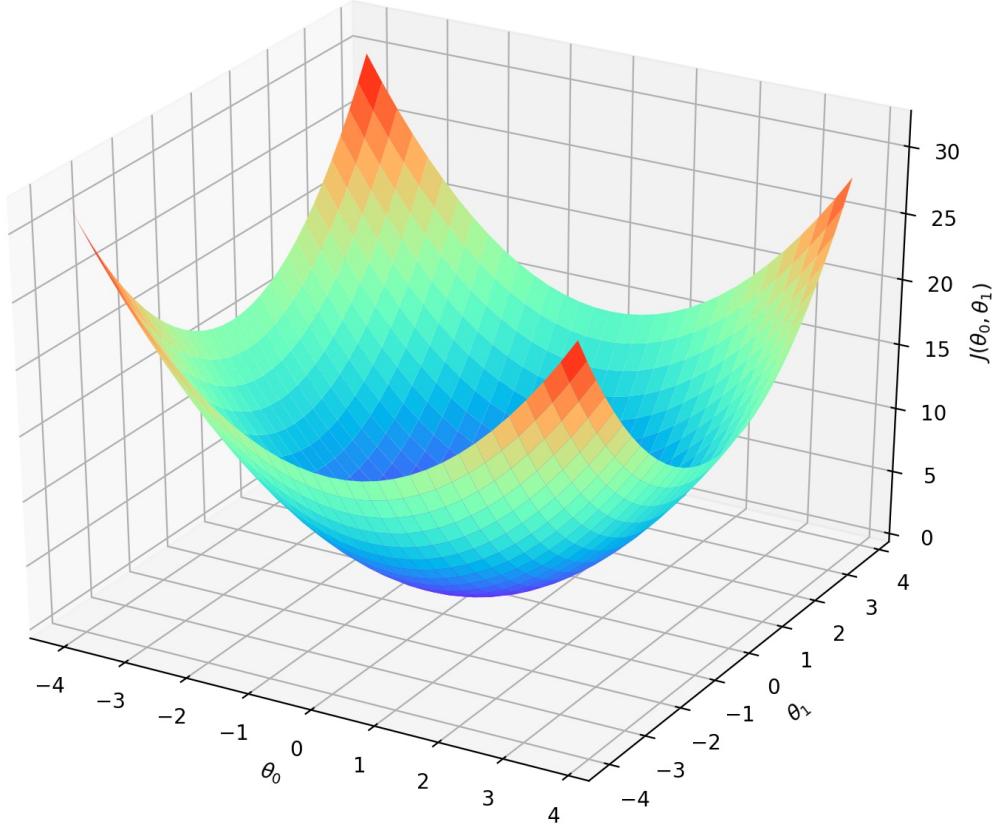


图2-5 代价函数三维图像

### 2.10.3 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。

### 2.10.4 常见代价函数

(1) 二次代价函数 (quadratic cost) :

$$J = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2 \quad (47)$$

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：

$$J = \frac{(y - a)^2}{2} \quad (48)$$

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：

$$\frac{\partial J}{\partial b} = (a - y)\sigma'(z) \quad (49)$$

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：

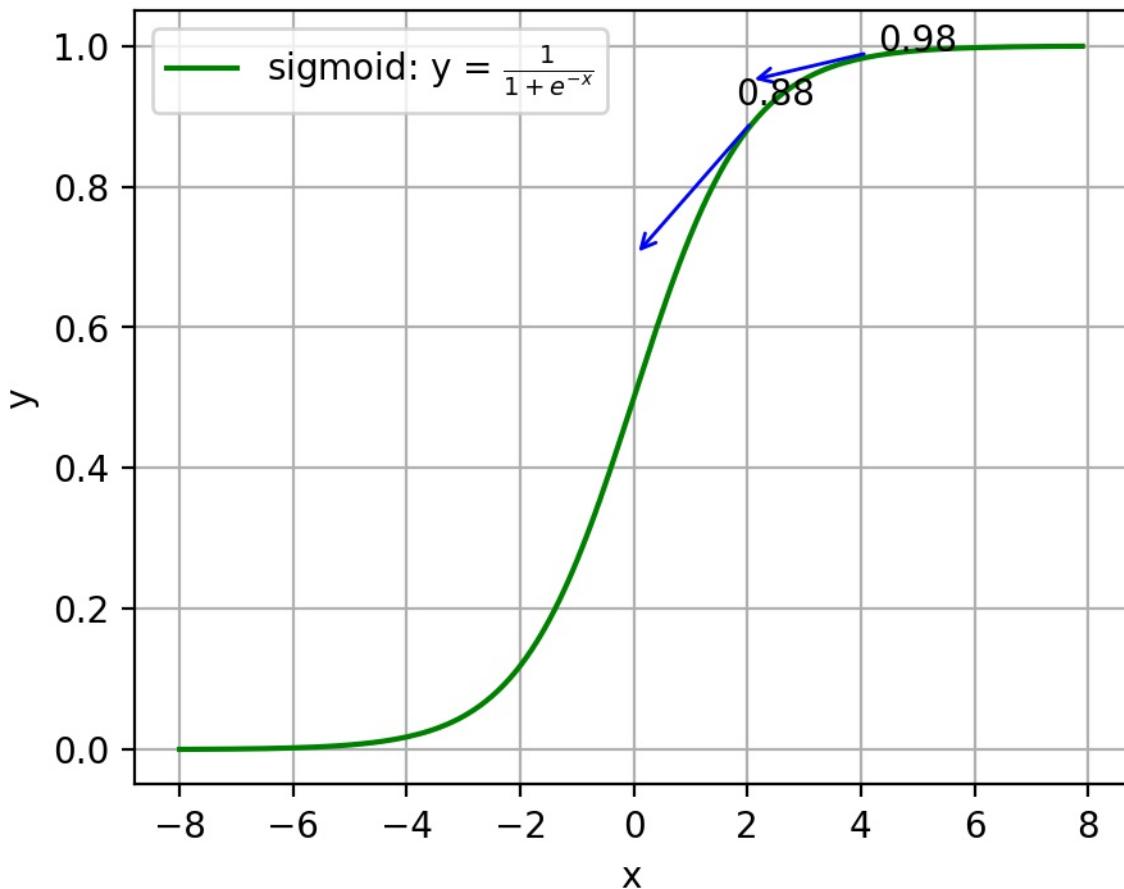


图2-6 sigmoid函数曲线

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) :

$$J = -\frac{1}{n} \sum_x [y \ln a + (1 - y) \ln (1 - a)] \quad (50)$$

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：

$$\frac{\partial J}{\partial w_j} = \frac{1}{n} \sum_x x_j (\sigma(z) - y), \quad \frac{\partial J}{\partial b} = \frac{1}{n} \sum_x (\sigma(z) - y) \quad (51)$$

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()`。

## 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：

$$\frac{\partial J}{\partial w_j} = \frac{1}{n} \sum_x x_j (\sigma(z) - y), \quad \frac{\partial J}{\partial b} = \frac{1}{n} \sum_x (\sigma(z) - y) \quad (52)$$

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。

## 2.11 损失函数

### 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。

### 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。

$$L(Y, f(x)) = \begin{cases} 1, & Y \neq f(x) \\ 0, & Y = f(x) \end{cases} \quad (53)$$

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：

$$L(Y, f(x)) = \begin{cases} 1, & |Y - f(x)| \geq T \\ 0, & |Y - f(x)| < T \end{cases} \quad (54)$$

#### (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：

$$L(Y, f(x)) = |Y - f(x)| \quad (55)$$

#### (3) 平方损失函数

$$L(Y, f(x)) = \sum_N (Y - f(x))^2 \quad (56)$$

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

$$L(Y, P(Y|X)) = -\log P(Y|X) \quad (57)$$

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。

#### (6) 指数损失函数

指数损失函数的标准形式为：

$$L(Y, f(x)) = \exp(-Yf(x)) \quad (58)$$

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：

$$L(y) = \max(0, 1 - ty) \quad (59)$$

统一的形式：

$$L(Y, f(x)) = \max(0, Yf(x)) \quad (60)$$

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

$$\min_{w, b} \sum_{i=1}^N (1 - y_i(wx_i + b)) + \lambda \|w\|^2 \quad (61)$$

上式相似于下式

$$\frac{1}{m} \sum_{i=1}^N l(wx_i + by_i) + \|w\|^2 \quad (62)$$

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

### 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

$$P(y = 1|x; \theta) = \frac{1}{1 + e^{-\theta^T x}} \quad (63)$$

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：

$$P(X = n) = \begin{cases} 1 - p, & n = 0 \\ p, & n = 1 \end{cases} \quad (64)$$

其似然函数为：

$$L(\theta) = \prod_{i=1}^m P(y = 1|x_i)^{y_i} P(y = 0|x_i)^{1-y_i} \quad (65)$$

对数似然函数为：

$$\begin{aligned}\ln L(\theta) &= \sum_{i=1}^m [y_i \ln P(y=1|x_i) + (1-y_i) \ln P(y=0|x_i)] \\ &= \sum_{i=1}^m [y_i \ln P(y=1|x_i) + (1-y_i) \ln(1-P(y=1|x_i))]\end{aligned}\quad (66)$$

对数函数在单个数据点上的定义为：

$$cost(y, p(y|x)) = -y \ln p(y|x) - (1-y) \ln(1-p(y|x)) \quad (67)$$

则全局样本损失函数为：

$$cost(y, p(y|x)) = - \sum_{i=1}^m [y_i \ln p(y_i|x_i) + (1-y_i) \ln(1-p(y_i|x_i))] \quad (68)$$

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。

## 2.11.4 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：

$$P(x_i; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x_i - \mu)^2}{2\sigma^2}\right) \quad (69)$$

(3) 其联合概率为：

$$\begin{aligned}P(x_1, x_2, \dots, x_n; \mu, \sigma) &= \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x_1 - \mu)^2}{2\sigma^2}\right) \\ &\times \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x_2 - \mu)^2}{2\sigma^2}\right) \times \dots \times \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x_n - \mu)^2}{2\sigma^2}\right)\end{aligned}\quad (70)$$

对上式取自然对数，可得：

$$\begin{aligned}\ln(P(x_1, x_2, \dots, x_n; \mu, \sigma)) &= \ln\left(\frac{1}{\sigma\sqrt{2\pi}}\right) - \frac{(x_1 - \mu)^2}{2\sigma^2} \\ &+ \ln\left(\frac{1}{\sigma\sqrt{2\pi}}\right) - \frac{(x_2 - \mu)^2}{2\sigma^2} + \dots + \ln\left(\frac{1}{\sigma\sqrt{2\pi}}\right) - \frac{(x_n - \mu)^2}{2\sigma^2}\end{aligned}\quad (71)$$

根据对数定律，上式可以化简为：

$$\begin{aligned}\ln(P(x_1, x_2, \dots, x_n; \mu, \sigma)) &= -n \ln(\sigma) - \frac{n}{2} \ln(2\pi) \\ &- \frac{1}{2\sigma^2} [(x_1 - \mu)^2 + (x_2 - \mu)^2 + \dots + (x_n - \mu)^2]\end{aligned}\quad (72)$$

然后求导为：

$$\frac{\partial \ln(P(x_1, x_2, \dots, x_n; \mu, \sigma))}{\partial \mu} = \frac{n}{\sigma^2} [\mu - (x_1 + x_2 + \dots + x_n)] \quad (73)$$

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：

$$\mu = \frac{x_1 + x_2 + \dots + x_n}{n} \quad (74)$$

同理，可计算 $\sigma$ 。

## 2.12 梯度下降

---

### 2.12.1 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。

### 2.12.2 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。

### 2.12.3 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：

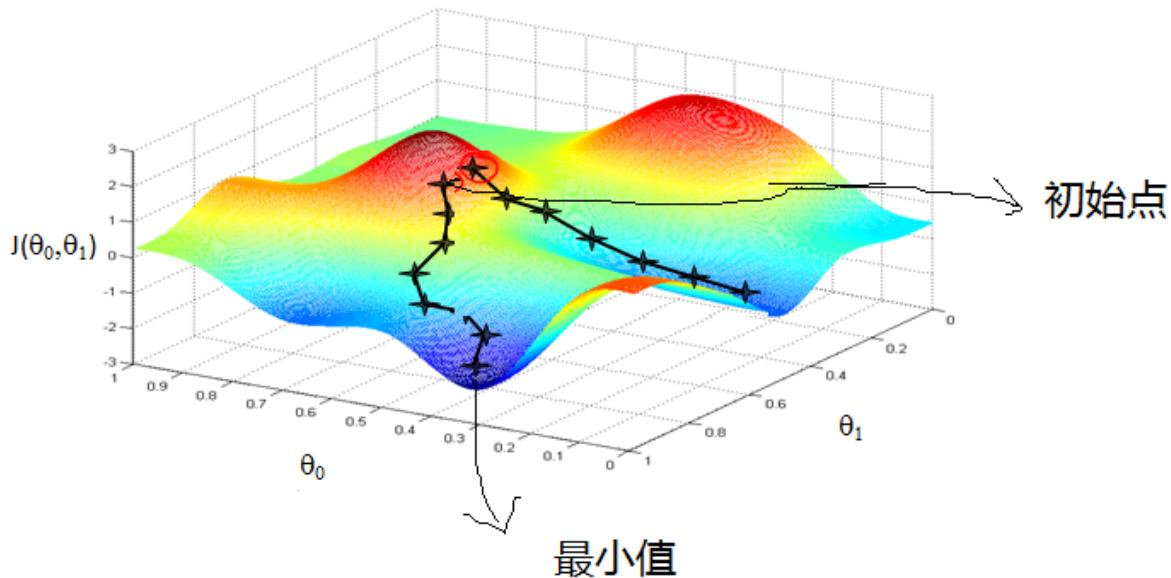


图2.7 梯度下降法经典图示

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，一直走到我们觉得已经到了山脚。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)；
- (3) 得到全局最优解或者接近全局最优解。

## 2.12.4 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：

$$h_{\theta}(x_1, x_2, \dots, x_n) = \theta_0 + \theta_1 x_1 + \dots + \theta_n x_n \quad (75)$$

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：

$$J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{2m} \sum_{j=0}^m (h_{\theta}(x_0^{(j)}, x_1^{(j)}, \dots, x_n^{(j)}) - y_j)^2 \quad (76)$$

- (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:

$$\frac{\partial}{\partial \theta_i} J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{2m} \sum_{j=0}^m (h_\theta(x_0^{(j)}, x_1^{(j)}, \dots, x_n^{(j)}) - y_j)^2 \quad (77)$$

2) 计算当前位置下降的距离。

$$\varphi_i = \alpha \frac{\partial}{\partial \theta_i} J(\theta_0, \theta_1, \dots, \theta_n) \quad (78)$$

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:

$$\theta_i = \theta_i - \alpha \frac{\partial}{\partial \theta_i} J(\theta_0, \theta_1, \dots, \theta_n) \quad (79)$$

$$\theta_i = \theta_i - \alpha \frac{1}{m} \sum_{j=0}^m (h_\theta(x_0^{(j)}, x_1^{(j)}, \dots, x_n^{(j)}) - y_j) x_i^{(j)} \quad (80)$$

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。

## 2.12.5 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, 根据损失函数的最小值, 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。

## 2.12.6 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 (BDG) 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:

$$h_\theta(x_0, x_1, \dots, x_n) = \theta_0 x_0 + \theta_1 x_1 + \dots + \theta_n x_n \quad (81)$$

损失函数为:

$$J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{2m} \sum_{j=0}^m (h_\theta(x_0^j, x_1^j, \dots, x_n^j) - y^j)^2 \quad (82)$$

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：

$$\frac{\partial}{\partial \theta_i} J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{m} \sum_{j=0}^m (h_\theta(x_0^j, x_1^j, \dots, x_n^j) - y^j) x_i^j \quad (83)$$

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 $\theta_i$ ：

$$\theta_i = \theta_i - \frac{1}{m} \sum_{j=0}^m (h_\theta(x_0^j, x_1^j, \dots, x_n^j) - y^j) x_i^j \quad (84)$$

c) 从上式可以注意到，它得到的虽然是一个全局最优解，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，

$$J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{m} \sum_{j=0}^m (y^j - h_\theta(x_0^j, x_1^j, \dots, x_n^j))^2 = \frac{1}{m} \sum_{j=0}^m cost(\theta, (x^j, y^j)) \quad (85)$$

b) 对每个参数 $\theta$ 按梯度方向更新 $\theta$ ：

$$\theta_i = \theta_i + (y^j - h_\theta(x_0^j, x_1^j, \dots, x_n^j)) \quad (86)$$

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	<ul style="list-style-type: none"> <li>a) 采用所有数据来梯度下降。</li> <li>b) 批量梯度下降法在样本量很大的时候，训练速度慢。</li> </ul>
随机梯度 下降	<ul style="list-style-type: none"> <li>a) 随机梯度下降用一个样本来梯度下降。</li> <li>b) 训练速度很快。</li> <li>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。</li> <li>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。</li> </ul>

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：

$$\theta_i = \theta_i - \alpha \sum_{j=t}^{t+n-1} (h_\theta(x_0^j, x_1^j, \dots, x_n^j) - y^j) x_i^j \quad (87)$$

## 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：

	<b>BGD</b>	<b>SGD</b>	<b>Mini-batch GD</b>	<b>Online GD</b>
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。

## 2.14 线性判别分析 (LDA)

### 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。

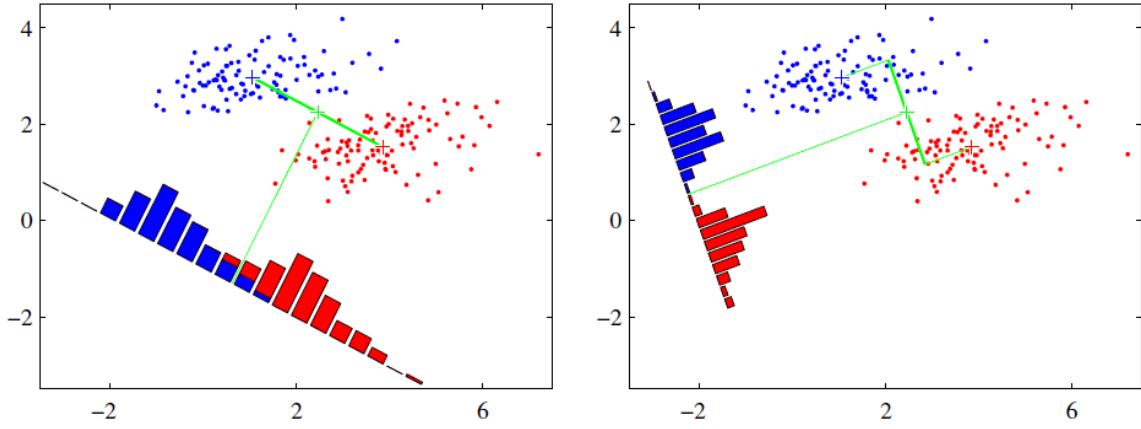
LDA分类思想简单总结如下：

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。

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”。

### 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。

### 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；

$\sum_j (j = 0, 1)$  为第  $j$  类样本的协方差矩阵。

其中

$$u_j = \frac{1}{N_j} \sum_{\mathbf{x} \in X_j} \mathbf{x} (j = 0, 1), \quad \sum_j = \sum_{\mathbf{x} \in X_j} (\mathbf{x} - u_j)(\mathbf{x} - u_j)^T (j = 0, 1) \quad (88)$$

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  $\mathbf{w}^T \sum_0 \mathbf{w}, \mathbf{w}^T \sum_1 \mathbf{w}$  尽量小，最小化  $\mathbf{w}^T \sum_0 \mathbf{w} - \mathbf{w}^T \sum_1 \mathbf{w}$ 。

定义

类内散度矩阵

$$S_w = \sum_0 + \sum_1 = \sum_{\mathbf{x} \in X_0} (\mathbf{x} - u_0)(\mathbf{x} - u_0)^T + \sum_{\mathbf{x} \in X_1} (\mathbf{x} - u_1)(\mathbf{x} - u_1)^T \quad (89)$$

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

$$\arg \max_w J(\mathbf{w}) = \frac{\|\mathbf{w}^T u_0 - \mathbf{w}^T u_1\|_2^2}{\mathbf{w}^T \sum_0 \mathbf{w} + \mathbf{w}^T \sum_1 \mathbf{w}} = \frac{\mathbf{w}^T (u_0 - u_1)(u_0 - u_1)^T \mathbf{w}}{\mathbf{w}^T (\sum_0 + \sum_1) \mathbf{w}} = \frac{\mathbf{w}^T S_b \mathbf{w}}{\mathbf{w}^T S_w \mathbf{w}} \quad (90)$$

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  $\mathbf{w}$ 。

## 2.14.4 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  $\bar{D} = \{(p_1, y_1), (p_2, y_2), \dots, (p_m, y_m)\}$

## 2.14.5 LDA和PCA区别

异同点	LDA	PCA
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. 两者都假设数据符合高斯分布；	
不同点	有监督的降维方法；	无监督的降维方法；
	降维最多降到 $k-1$ 维；	降维多少没有限制；
	可以用于降维，还可以用于分类；	只用于降维；
	选择分类性能最好的投影方向；	选择样本点投影具有最大方差的方向；
	更明确，更能反映样本间差异；	目的较为模糊；

## 2.14.6 LDA优缺点

优缺点	简要说明
优点	1. 可以使用类别的先验知识； 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，其目的更明确，更能反映样本间的差异；
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而不是均值时，降维效果不好； 4. LDA可能过度拟合数据。

## 2.15 主成分分析 (PCA)

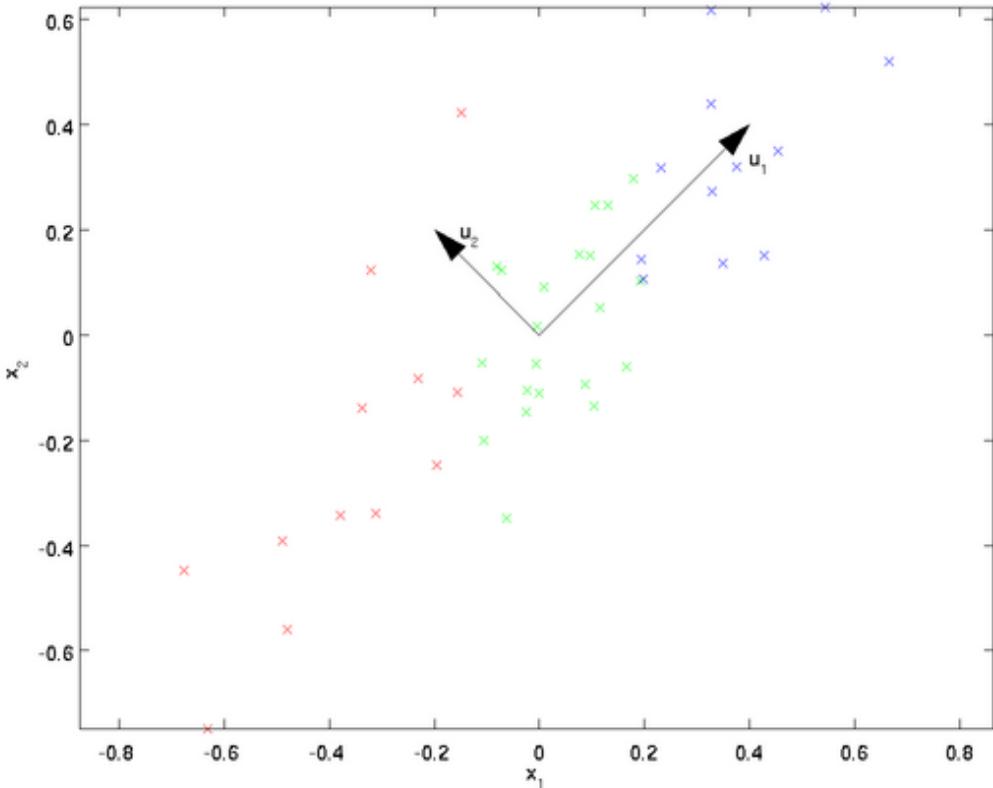
### 2.15.1 主成分分析 (PCA) 思想总结

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。

### 2.15.2 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。

### 2.15.3 PCA算法推理

下面以基于最小投影距离为评价指标推导:

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。

如果用  $z^{(i)}$  去恢复  $x^{(i)}$ , 则得到的恢复数据为  $\hat{x}^{(i)} = \sum_{j=1}^{n'} z_j^{(i)} w_j = Wz^{(i)}$ , 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  $\sum_{i=1}^m \|\hat{x}^{(i)} - x^{(i)}\|_2^2$ 。对此式进行推导, 可得:

$$\begin{aligned}
& \sum_{i=1}^m \|\hat{x}^{(i)} - x^{(i)}\|_2^2 = \sum_{i=1}^m \|Wz^{(i)} - x^{(i)}\|_2^2 \\
&= \sum_{i=1}^m (Wz^{(i)})^T (Wz^{(i)}) - 2 \sum_{i=1}^m (Wz^{(i)})^T x^{(i)} + \sum_{i=1}^m (x^{(i)})^T x^{(i)} \\
&= \sum_{i=1}^m (z^{(i)})^T (z^{(i)}) - 2 \sum_{i=1}^m (z^{(i)})^T x^{(i)} + \sum_{i=1}^m (x^{(i)})^T x^{(i)} \\
&= - \sum_{i=1}^m (z^{(i)})^T (z^{(i)}) + \sum_{i=1}^m (x^{(i)})^T x^{(i)} \\
&= -tr \left( W^T \left( \sum_{i=1}^m x^{(i)} (x^{(i)})^T \right) W \right) + \sum_{i=1}^m (x^{(i)})^T x^{(i)} \\
&= -tr (W^T X X^T W) + \sum_{i=1}^m (x^{(i)})^T x^{(i)}
\end{aligned} \tag{91}$$

在推导过程中，分别用到了  $\bar{x}^{(i)} = Wz^{(i)}$ ，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， $\sum_{i=1}^m (x^{(i)})^T x^{(i)}$  是一个常量。最小化  $\sum_{i=1}^m \|\hat{x}^{(i)} - x^{(i)}\|_2^2$  又可等价于

$$\underbrace{\arg \min_W}_{W} - tr (W^T X X^T W) \text{ s.t. } W^T W = I \tag{92}$$

利用拉格朗日函数可得到

$$J(W) = -tr (W^T X X^T W) + \lambda (W^T W - I) \tag{93}$$

对  $W$  求导，可得  $-XX^T W + \lambda W = 0$ ，也即  $XX^T W = \lambda W$ 。 $XX^T$  是  $n'$  个特征向量组成的矩阵， $\lambda$  为  $XX^T$  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。

## 2.15.4 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  $D' = (z^{(1)}, z^{(2)}, \dots, z^{(n')})$ 。

主要步骤如下：

1. 对所有的样本进行中心化， $x^{(i)} = x^{(i)} - \frac{1}{m} \sum_{j=1}^m x^{(j)}$ 。
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。

## 2.15.5 PCA算法主要优缺点

优缺点	简要说明
优点	1. 仅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，可消除原始数据成分间的相互影响的因素。 3. 计算方法简单，主要运算是特征值分解，易于实现。
缺点	1. 主成分各个特征维度的含义具有一定的模糊性，不如原始样本特征的解释性强。 2. 方差小的非主成分也可能含有对样本差异的重要信息，因降维丢弃可能对后续数据处理有影响。

## 2.15.6 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在数据中明确的显示出来；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。

## 2.15.7 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：

$$\sum_{i=1}^m x^{(i)} \left( x^{(i)} \right)^T W = \lambda W \quad (94)$$

其映射为

$$\sum_{i=1}^m \phi \left( x^{(i)} \right) \phi \left( x^{(i)} \right)^T W = \lambda W \quad (95)$$

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。

## 2.16 模型评估

### 2.16.1 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

### 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”。
- Error = Bias + Variance + 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。

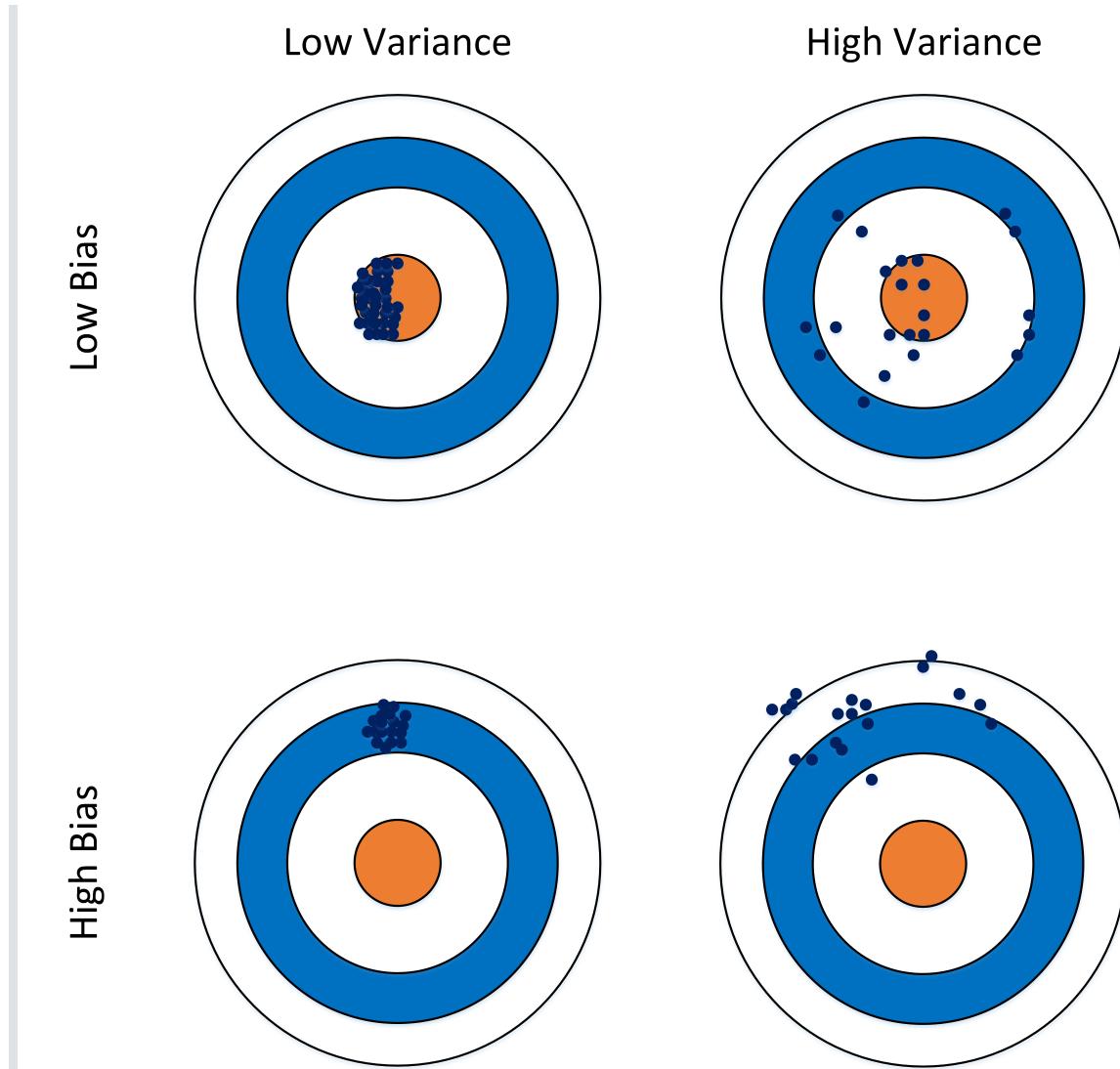
**对于Bias：**

- 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。



### 2.16.3 经验误差与泛化误差

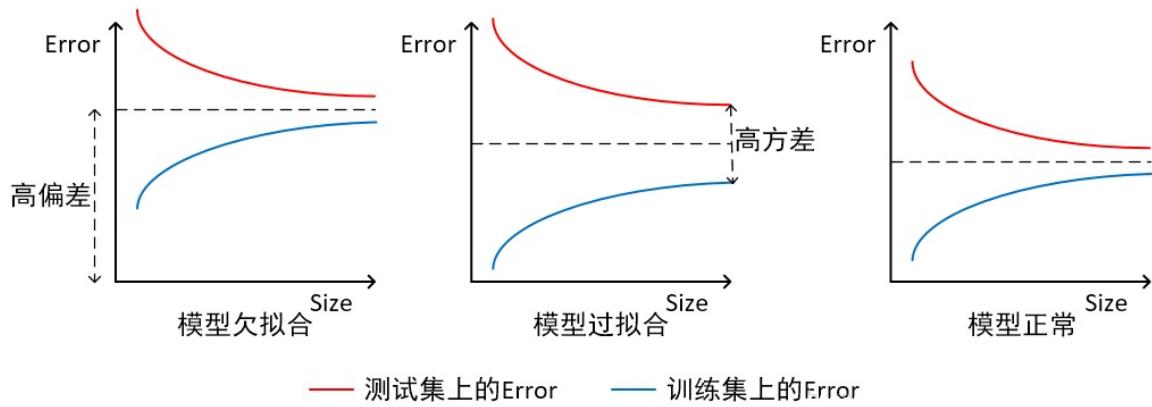
经验误差 (empirical error)：也叫训练误差 (training error)，模型在训练集上的误差。

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”。

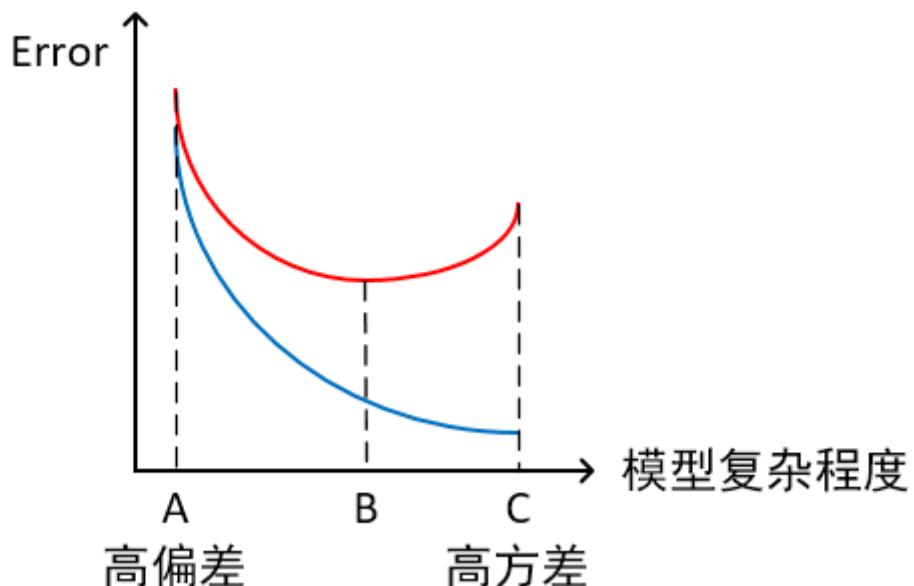
### 2.16.4 图解欠拟合、过拟合

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



## 2. 横轴为模型复杂程度，纵轴为误差



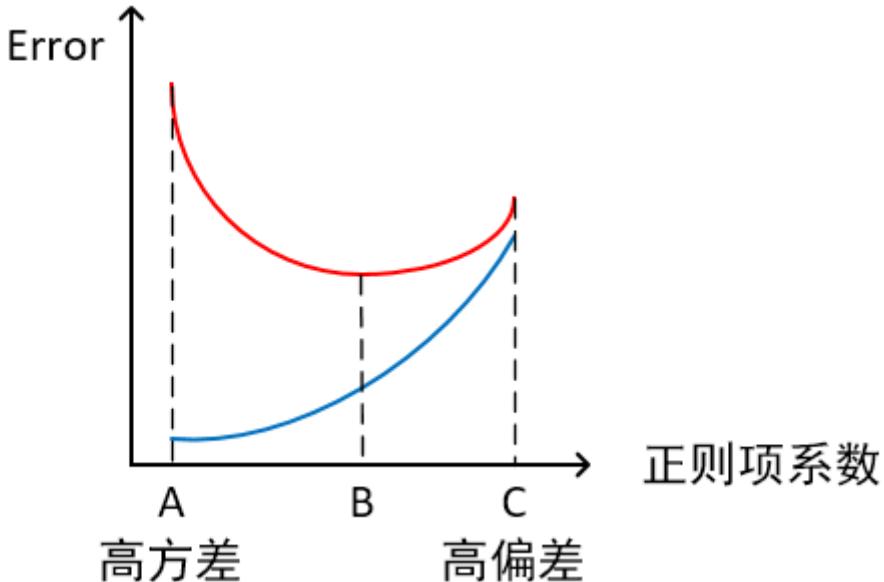
红线为测试集上的Error, 蓝线为训练集上的Error

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打折扣，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。

## 2.16.5 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，其实质就是线性模型，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。

## 2.16.6 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。

## 2.16.7 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. 在每种情况下，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。

## 2.16.8 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)

## 2.16.9 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。

## 2.16.10 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：

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

**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。

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

**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。

## 2.16.11 ROC与AUC

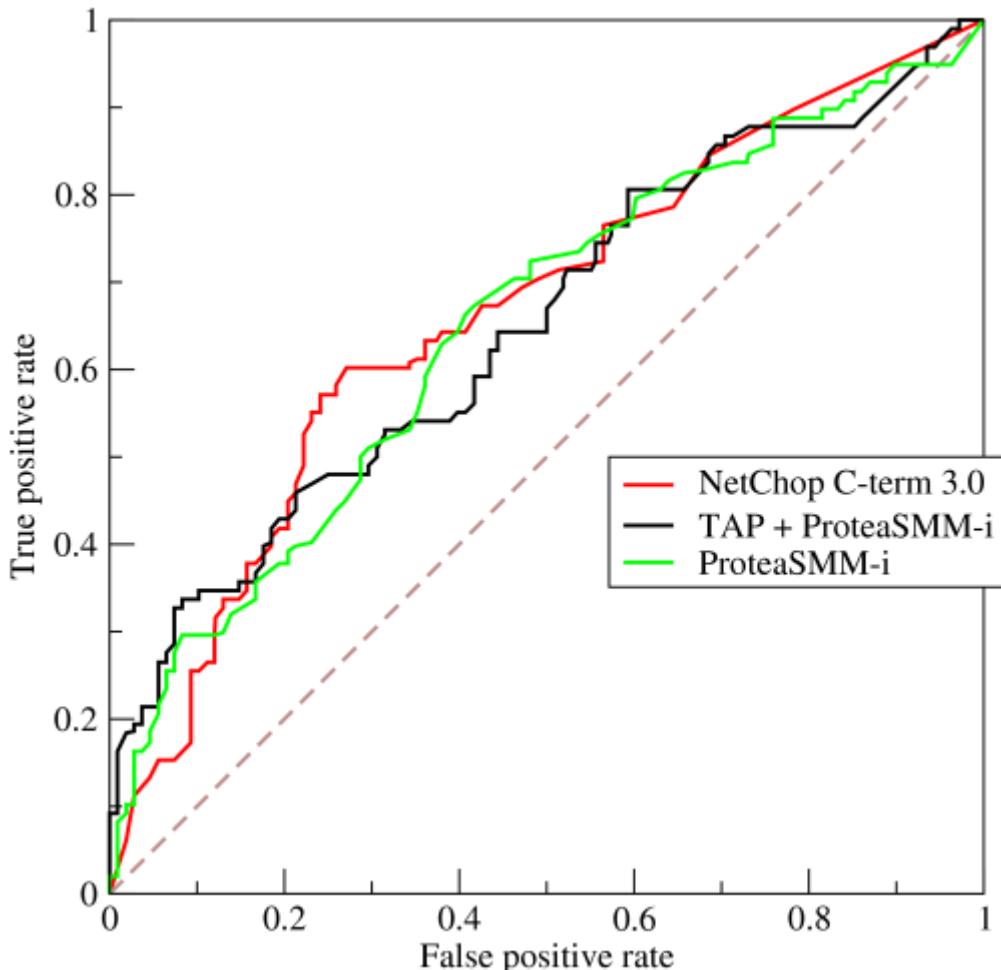
ROC全称是“受试者工作特征” (Receiver Operating Characteristic)。

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

$$TPR = \frac{TP}{TP+FN}, FPR = \frac{FP}{FP+TN} \quad (96)$$

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。

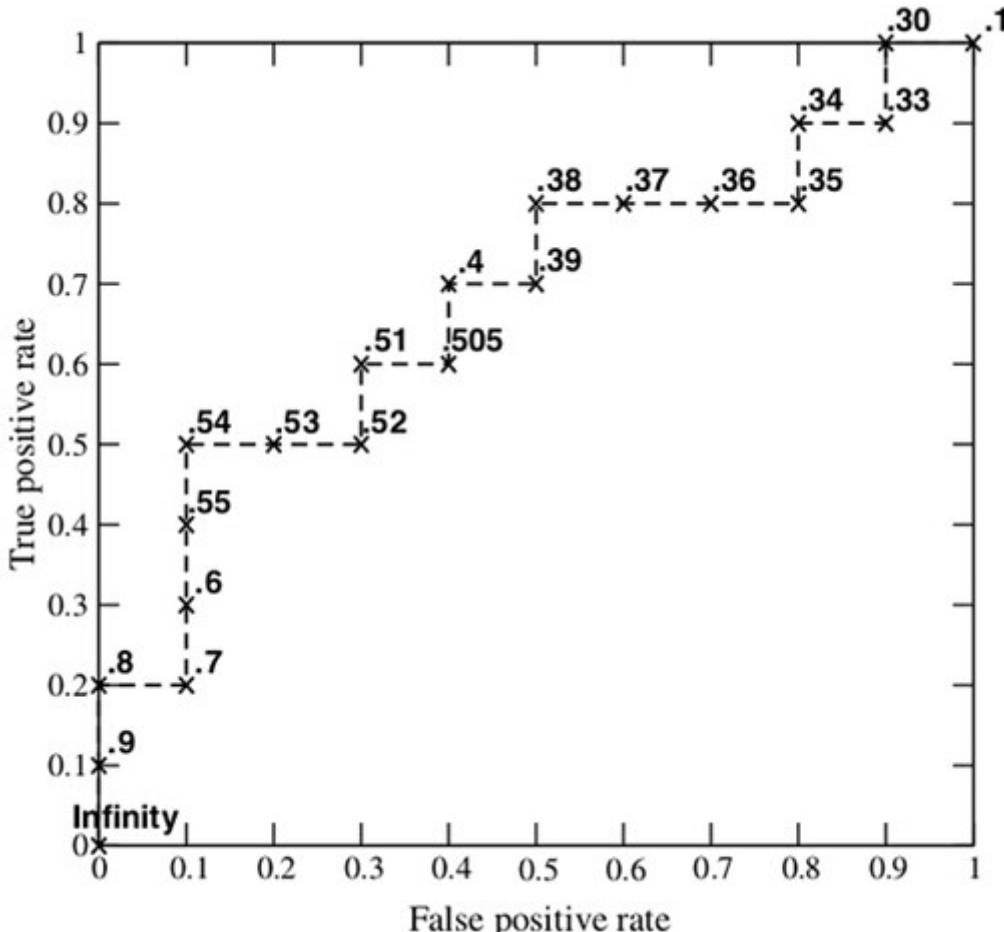
ROC曲线所覆盖的面积称为AUC (Area Under Curve), 可以更直观的判断学习器的性能, AUC越大则性能越好。

## 2.16.12 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。



## 2.16.13 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 $\text{score} \geq 0.1$ ，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：

$$\text{TPR} = \text{TP}/(\text{TP}+\text{FN}) = 1; \quad \text{FPR} = \text{FP}/(\text{TN}+\text{FP}) = 1;$$

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：

$$\text{TPR} = \text{TP}/(\text{TP}+\text{FN}) = 1; \quad \text{FPR} = \text{FP}/(\text{TN}+\text{FP}) = 0.5;$$

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：

$$\text{TPR} = \text{TP}/(\text{TP}+\text{FN}) = 0.5; \quad \text{FPR} = \text{FP}/(\text{TN}+\text{FP}) = 0.5;$$

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。

## 2.16.14 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。

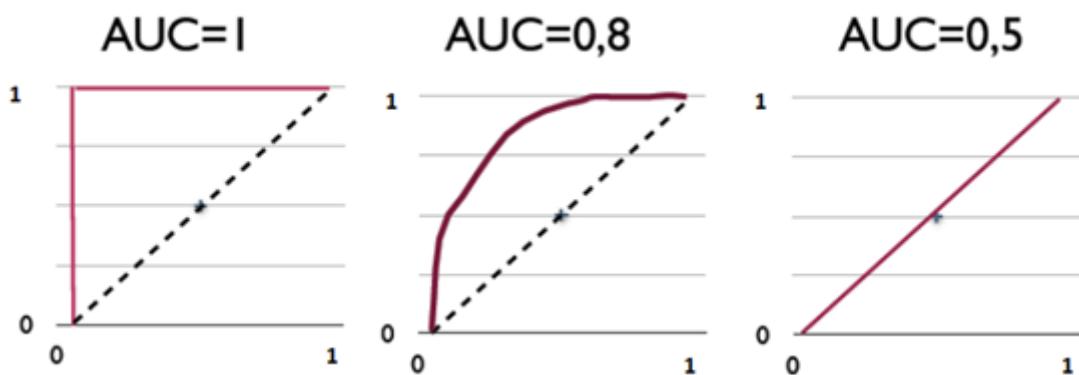
## 2.16.15 为什么使用Roc和Auc评价分类器

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。

## 2.16.16 直观理解AUC

下图展现了三种AUC的值：

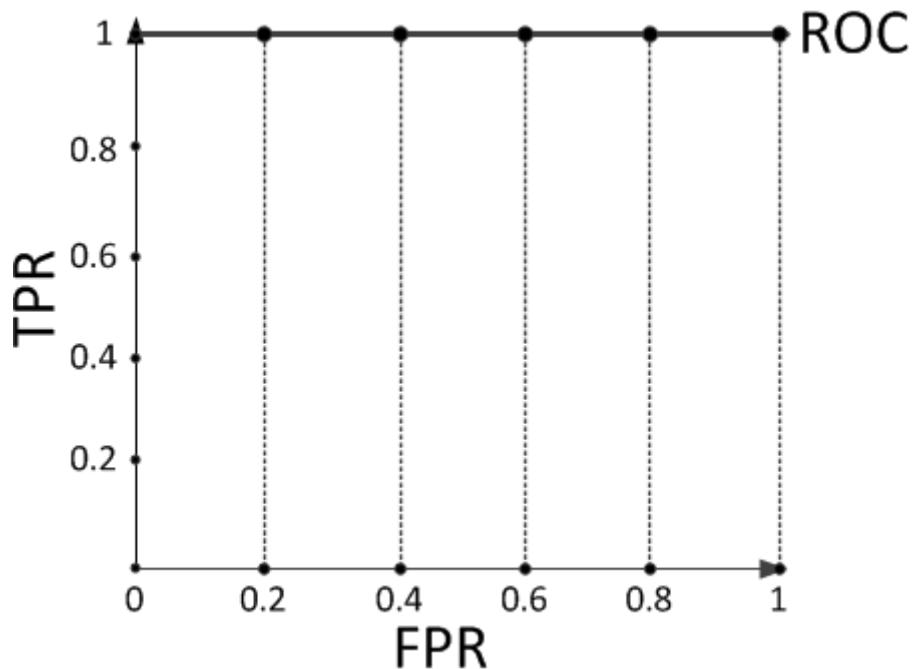


AUC是衡量二分类模型优劣的一种评价指标，表示正例排在负例前面的概率。其他评价指标有精确度、准确率、召回率，而AUC比这三者更为常用。

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。

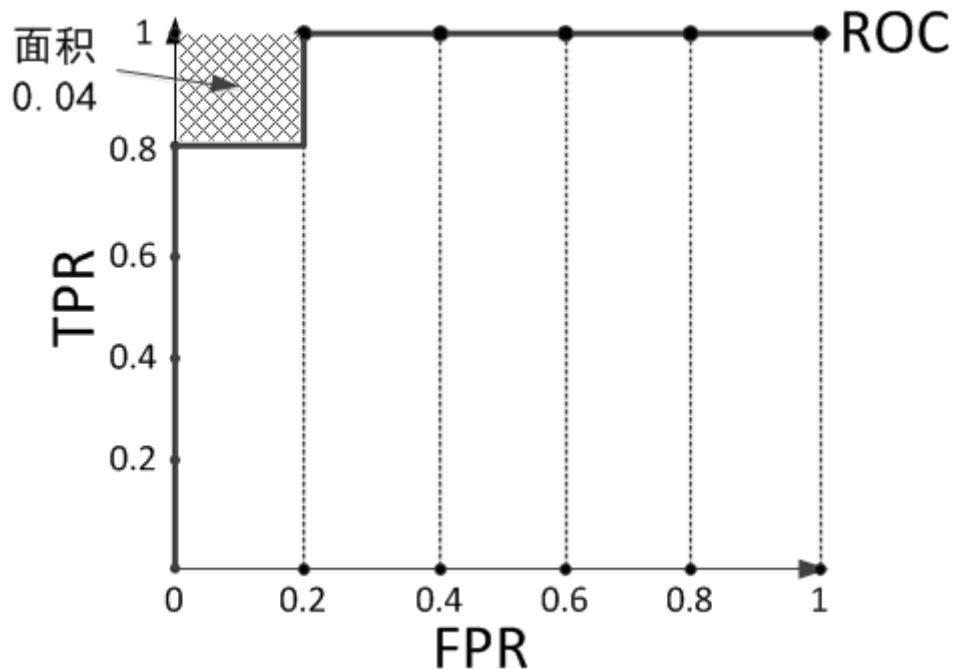
举例：

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：



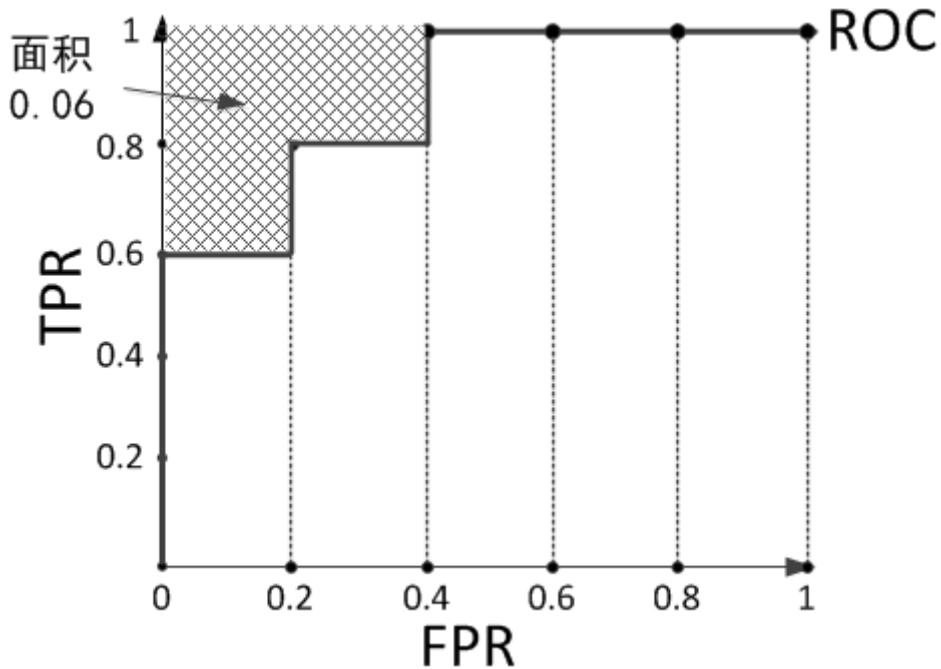
描点方式按照样本预测结果的得分高低从左至右开始遍历。从原点开始，每遇到1便向y轴正方向移动y轴最小步长1个单位，这里是 $1/5=0.2$ ；每遇到0则向x轴正方向移动x轴最小步长1个单位，这里也是0.2。不难看出，上图的AUC等于1，印证了正例排在负例前面的概率的确为100%。

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 $0.2 \times 0.2 \times 3 = 0.12$ 。

### 2.16.17 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：

真实类别	预测类别	
	第 0 类	第 1 类
第 0 类	0	$Cost_{01}$
第 1 类	$Cost_{10}$	0

当判断正确的时候，值为0，不正确的时候，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：

$$E(f; D; cost) = \frac{1}{m} \left( \sum_{x_i \in D^+} (f(x_i) \neq y_i) \times Cost_{01} + \sum_{x_i \in D^-} (f(x_i) \neq y_i) \times Cost_{10} \right) \quad (97)$$

$D^+$ 、 $D^-$ 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：

$$P(+)\text{Cost} = \frac{p * Cost_{01}}{p * Cost_{01} + (1 - p) * Cost_{10}} \quad (98)$$

其中 $p$ 是样本为正例的概率。

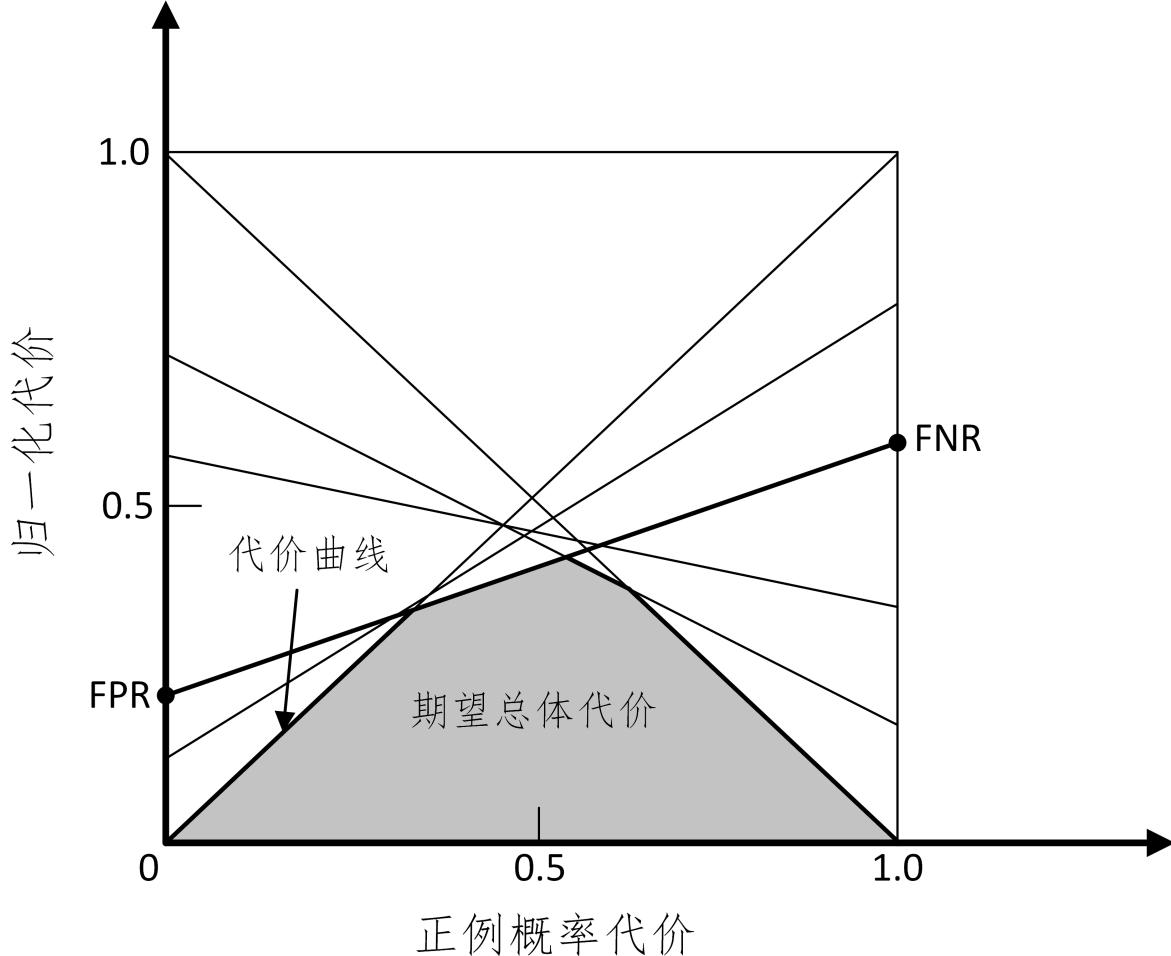
代价曲线纵轴维[0,1]的归一化代价：

$$Cost_{norm} = \frac{FNR * p * Cost_{01} + FNR * (1 - p) * Cost_{10}}{p * Cost_{01} + (1 - p) * Cost_{10}} \quad (99)$$

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。



### 2.16.18 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。

### 2.16.19 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。

## 2.16.20 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。

# 2.17 决策树

## 2.17.1 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。

## 2.17.2 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。

## 2.17.3 决策树学习基本算法

### 机器学习算法 1 决策树学习基本算法

输入：训练集 $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$ ；属性集 $A = \{a_1, a_2, \dots, a_d\}$

输出：以 $node$ 为根节点的一棵决策树

```
1: function TreeGenerate(D, A)
2:     生成节点node
3:     if D中的样本全属于同一类别C then
4:         将node标记为C类叶节点;return
5:     end if
6:     if A = ∅ OR D中样本再A上取值相同 then
7:         将node标记为叶节点，其类别标记为D中样本类最多的类;return
8:     end if
9:     从A中选择最优划分属性 $a_*$ 
10:    for  $a_*$ 的每一个值 $a_*^v$  do
11:        为node生成一个分支;令 $D_v$ 表示D中在 $a_*$ 上取值为 $a_*^v$ 的样本子集;
12:        if  $D_v$ 为空 then 将分支节点标记为叶节点，其类别标记为D中样本最多的类;return
13:        else 以TreeGenerate( $D_v, A \ {a_*}$ )为分支节点
14:        end if
15:    end for
16: end function
```

## 2.17.4 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是太好。
- 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。

## 2.17.5 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：

$$H(X) = - \sum_{i=1}^n p_i \log_2 p_i \quad (100)$$

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 $|C_k|$ 为类别为k的样本个数， $|D|$ 为样本总数。样本集合D的熵为：

$$H(D) = - \sum_{k=1}^k \frac{|C_k|}{|D|} \log_2 \frac{|C_k|}{|D|} \quad (101)$$

## 2.17.6 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：

$$g(D, A) = H(D) - H(D|A) \quad (102)$$

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:

$$\text{信息增益比} = \text{惩罚参数} \times \text{信息增益}$$

信息增益比本质: 在信息增益的基础之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。

## 2.17.7 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。

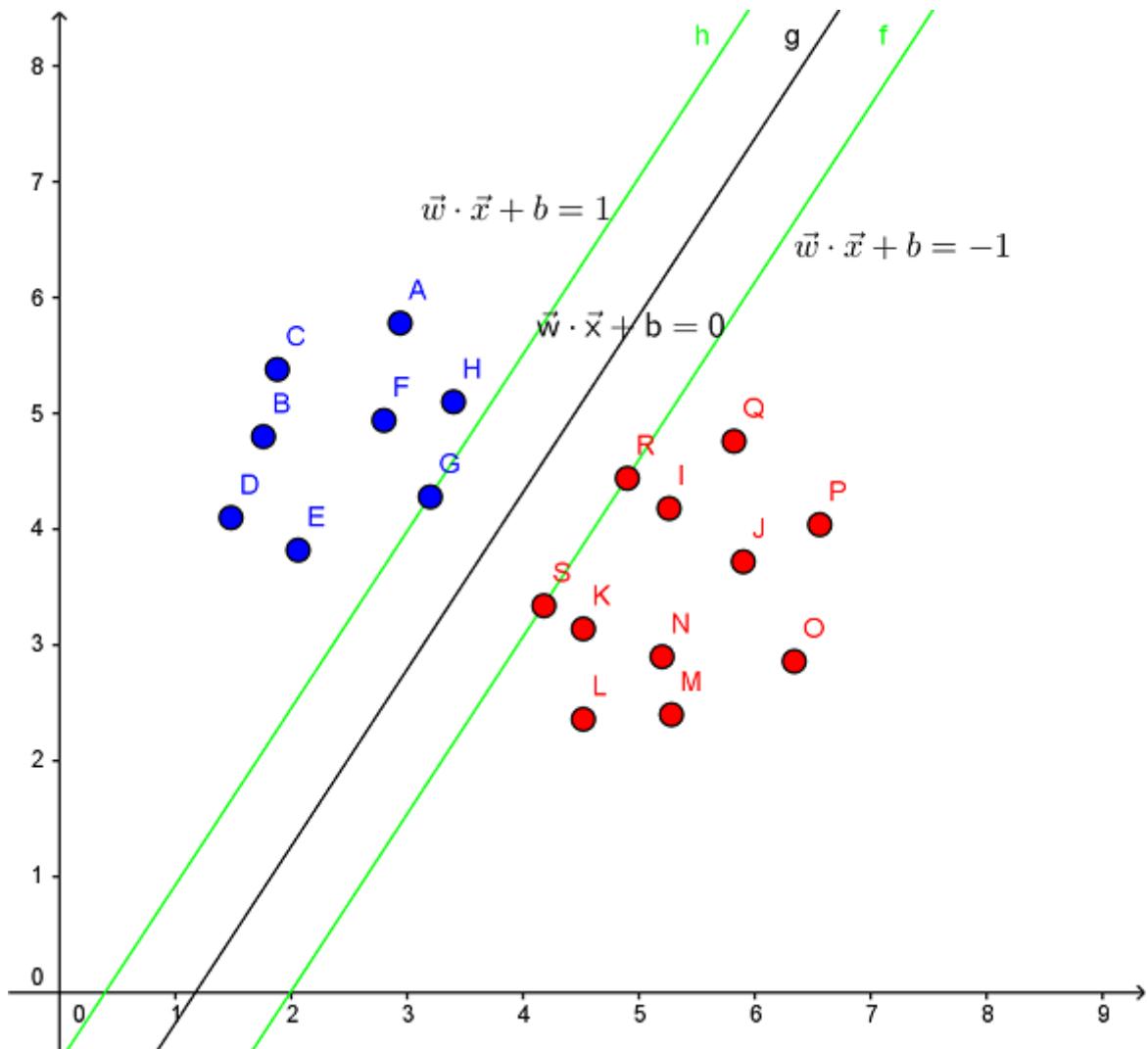
## 2.18 支持向量机

### 2.18.1 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；

## 2.18.2 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。

## 2.18.3 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中的形式和参数。
- 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。

## 2.18.4 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。

## 2.18.5 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

$$\min_{w,b} \frac{1}{2} \|w\|^2 \quad (103) \quad (1)$$

$$s.t. y_i(w^T x_i + b) \geq 1, i = 1, 2, \dots, m. \quad (104)$$

**step 1.** 转化问题：

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

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， $\alpha_i (1 - y_i(w^T x_i + b))$ 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：

$$\max_{\alpha_i \geq 0} \min_{w,b} \left\{ \frac{1}{2} \|w\|^2 + \sum_{i=1}^m \alpha_i (1 - y_i(w^T x_i + b)) \right\} \quad (105)$$

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？

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

若方程组(3)无解，则v是问题(1)的一个下界。若(3)有解，则

$$\forall \alpha > 0, \min_{\mathbf{w}, b} \left\{ \frac{1}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^m \alpha_i (1 - y_i(\mathbf{w}^T \mathbf{x}_i + b)) \right\} < v \tag{106}$$

由逆否命题得：若

$$\exists \alpha > 0, \min_{\mathbf{w}, b} \left\{ \frac{1}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^m \alpha_i (1 - y_i(\mathbf{w}^T \mathbf{x}_i + b)) \right\} \geq v \tag{107}$$

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

$$\max_{\alpha_i \geq 0} \min_{\mathbf{w}, b} \left\{ \frac{1}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^m \alpha_i (1 - y_i(\mathbf{w}^T \mathbf{x}_i + b)) \right\} \tag{108}$$

**step 3.** 令

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^m \alpha_i (1 - y_i(\mathbf{w}^T \mathbf{x}_i + b)) \tag{109}$$

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

$$p^* = \frac{1}{2} \|\mathbf{w}^*\|^2 \geq L(\mathbf{w}^*, b, \mathbf{a}) \geq \min_{\mathbf{w}, b} L(\mathbf{w}, b, \mathbf{a}) \tag{110}$$

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

此时，取最大值即可求得好的下界，即

$$\max_{\alpha_i \geq 0} \min_{\mathbf{w}, b} L(\mathbf{w}, b, \mathbf{a}) \tag{111}$$

## 2.18.7 常见的核函数有哪些

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	$k(x, y) = \exp(-\sigma(x^k - y^k)^2)$	

核函数	$K(x_i, x_j) = \tanh(ax^t y + c)$	$\tanh$ 为双曲正切函数, $a > 0, c < 0$
-----	-----------------------------------	------------------------------------

## 2.18.9 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：
  - ①增、删非支持向量样本对模型没有影响；
  - ②支持向量样本集具有一定的鲁棒性；
  - ③有些成功的应用中，SVM方法对核的选取不敏感
- (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。

## 2.18.10 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。

## 2.18.11 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：

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m [y^i \log h_\theta(x^i) + (1 - y^i) \log(1 - h_\theta(x^i))] \quad (112)$$

支持向量机的目标函数：

$$L(w, n, a) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^n \alpha_i (y_i (w^T x_i + b) - 1) \quad (113)$$

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！！！\*\*

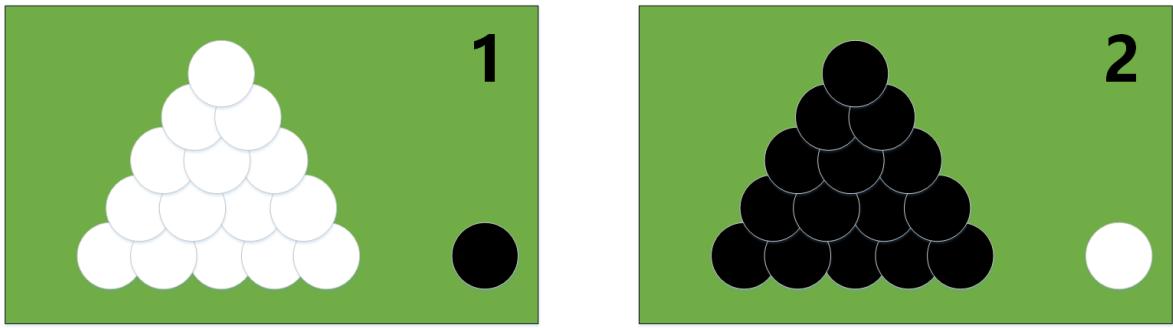
### 6、SVM自带结构风险最小化，LR则是经验风险最小化。

7、SVM会用核函数而LR一般不用核函数。

## 2.19 贝叶斯分类器

### 2.19.1 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”。

## 2.19.2 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：

$$D = \vec{x}_1, \vec{x}_2, \dots, \vec{x}_n \quad (114)$$

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。

$$l(\vec{\theta}) = p(D|\vec{\theta}) = p(\vec{x}_1, \vec{x}_2, \dots, \vec{x}_n | \vec{\theta}) = \prod_{i=1}^n p(\vec{x}_i | \vec{\theta}) \quad (115)$$

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：

$$\hat{\vec{\theta}} = d(D) = \arg \max_{\vec{\theta}} l(\vec{\theta}) \quad (116)$$

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

## 2.19.3 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 $\mathbf{x}$ ，它属于哪一类呢？

计算步骤如下：

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

step 2. 通过比较所有的 $P(c_i|\mathbf{x})$ ，得到样本 $\mathbf{x}$ 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：

$$P(c_i|\mathbf{x}) = \frac{P(\mathbf{x}|c_i)P(c_i)}{P(\mathbf{x})}. \quad (117)$$

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。

## 2.19.4 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：

$$P(\mathbf{x}|c_i) = P(x_1, x_2, \dots, x_d|c_i) \quad (118)$$

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：

$$P(x_1, x_2, \dots, x_d|c_i) = \prod_{j=1}^d P(x_j|c_i) \quad (119)$$

这样的话，我们就可以很容易地推出相应的判定准则了：

$$h_{nb}(\mathbf{x}) = \arg \max_{c_i \in Y} P(c_i) \prod_{j=1}^d P(x_j|c_i) \quad (120)$$

### 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 $P(x_j|c_i)$

$$P(x_j|c_i) = \frac{P(x_j, c_i)}{P(c_i)} \approx \frac{\#(x_j, c_i)}{\#(c_i)} \quad (121)$$

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 $\mu$ 和方差 $\sigma$ ，然后计算 $x_j$ 在这个分布中的概率密度 $P(x_j|c_i)$ 。

## 2.19.5 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	0.242	0.267	不

编号	色泽	根蒂	敲声	纹理	脐部	触感	密度	含糖率	好瓜
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

$$P(\text{好瓜} = \text{是}) = \frac{8}{17} = 0.471 \quad (122)$$

$$P(\text{好瓜} = \text{否}) = \frac{9}{17} = 0.529 \quad (123)$$

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）

$$P_{\text{青绿}|\text{是}} = P(\text{色泽} = \text{青绿} | \text{好瓜} = \text{是}) = \frac{3}{8} = 0.375 \quad (124)$$

$$P_{\text{青绿}|\text{否}} = P(\text{色泽} = \text{青绿} | \text{好瓜} = \text{否}) = \frac{3}{9} \approx 0.333 \quad (125)$$

$$P_{\text{蜷缩}|\text{是}} = P(\text{根蒂} = \text{蜷缩} | \text{好瓜} = \text{是}) = \frac{5}{8} = 0.625 \quad (126)$$

$$P_{\text{蜷缩}|\text{否}} = P(\text{根蒂} = \text{蜷缩} | \text{好瓜} = \text{否}) = \frac{3}{9} = 0.333 \quad (127)$$

$$P_{\text{浊响}|\text{是}} = P(\text{敲声} = \text{浊响} | \text{好瓜} = \text{是}) = \frac{6}{8} = 0.750 \quad (128)$$

$$P_{\text{浊响}|\text{否}} = P(\text{敲声} = \text{浊响} | \text{好瓜} = \text{否}) = \frac{4}{9} \approx 0.444 \quad (129)$$

$$P_{\text{清晰}|\text{是}} = P(\text{纹理} = \text{清晰} | \text{好瓜} = \text{是}) = \frac{7}{8} = 0.875 \quad (130)$$

$$P_{\text{清晰}|\text{否}} = P(\text{纹理} = \text{清晰} | \text{好瓜} = \text{否}) = \frac{2}{9} \approx 0.222 \quad (131)$$

$$P_{\text{凹陷}|\text{是}} = P(\text{脐部} = \text{凹陷} | \text{好瓜} = \text{是}) = \frac{6}{8} = 0.750 \quad (132)$$

$$P_{\text{凹陷}|\text{否}} = P(\text{脐部} = \text{凹陷} | \text{好瓜} = \text{否}) = \frac{2}{9} \approx 0.222 \quad (133)$$

$$P_{\text{硬滑}|\text{是}} = P(\text{触感} = \text{硬滑} | \text{好瓜} = \text{是}) = \frac{6}{8} = 0.750 \quad (134)$$

$$P_{\text{硬滑|否}} = P(\text{触感} = \text{硬滑} | \text{好瓜} = \text{否}) = \frac{6}{9} \approx 0.667 \quad (135)$$

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

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

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

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

于是有

$$P(\text{好瓜} = \text{是}) \times P_{\text{青绿|是}} \times P_{\text{蜷缩|是}} \times P_{\text{浊响|是}} \times P_{\text{清晰|是}} \times P_{\text{凹陷|是}} \quad (136)$$

$$\times P_{\text{硬滑|是}} \times p_{\text{密度: } 0.697|\text{是}} \times p_{\text{含糖: } 0.460|\text{是}} \approx 0.063 \quad (137)$$

(138)

$$P(\text{好瓜} = \text{否}) \times P_{\text{青绿|否}} \times P_{\text{蜷缩|否}} \times P_{\text{浊响|否}} \times P_{\text{清晰|否}} \times P_{\text{凹陷|否}} \quad (139)$$

$$\times P_{\text{硬滑|否}} \times p_{\text{密度: } 0.697|\text{否}} \times p_{\text{含糖: } 0.460|\text{否}} \approx 6.80 \times 10^{-5} \quad (140)$$

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”。

## 2.19.6 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：

$$P(\mathbf{x}|c_i) = \prod_{j=1}^d P(x_j|c_i, \text{pa}_j) \quad (141)$$

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

$$P(x_j|c_i, \text{pa}_j) = \frac{P(x_j, c_i, \text{pa}_j)}{P(c_i, \text{pa}_j)} \quad (142)$$

## 2.20 EM算法

### 2.20.1 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。

## 2.20.2 EM算法推导

对于 $m$ 个样本观察数据 $x = (x^1, x^2, \dots, x^m)$ ，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：

$$\theta = \arg \max_{\theta} \sum_{i=1}^m \log P(x^{(i)}; \theta) \quad (143)$$

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

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

由于上式不能直接求出 $\theta$ ，采用缩放技巧：

$$\sum_{i=1}^m \log \sum_{z^{(i)}} P(x^{(i)}, z^{(i)}; \theta) = \sum_{i=1}^m \log \sum_{z^{(i)}} Q_i(z^{(i)}) \frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \quad (144)$$

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

$$\geq \sum_{i=1}^m \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \quad (145)$$

上式用到了Jensen不等式：

$$\log \sum_j \lambda_j y_j \geq \sum_j \lambda_j \log y_j, \lambda_j \geq 0, \sum_j \lambda_j = 1 \quad (146)$$

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：

$$\frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} = c, c \text{ 为常数} \quad (147)$$

由于 $Q_i(z^{(i)})$ 是一个分布，所以满足

$$\sum_z Q_i(z^{(i)}) = 1 \quad (148)$$

综上，可得：

$$Q_i(z^{(i)}) = \frac{P(x^{(i)}, z^{(i)}; \theta)}{\sum_z P(x^{(i)}, z^{(i)}; \theta)} = \frac{P(x^{(i)}, z^{(i)}; \theta)}{P(x^{(i)}; \theta)} = P(z^{(i)} | x^{(i)}; \theta) \quad (149)$$

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：

$$\arg \max_{\theta} \sum_{i=1}^m \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \quad (150)$$

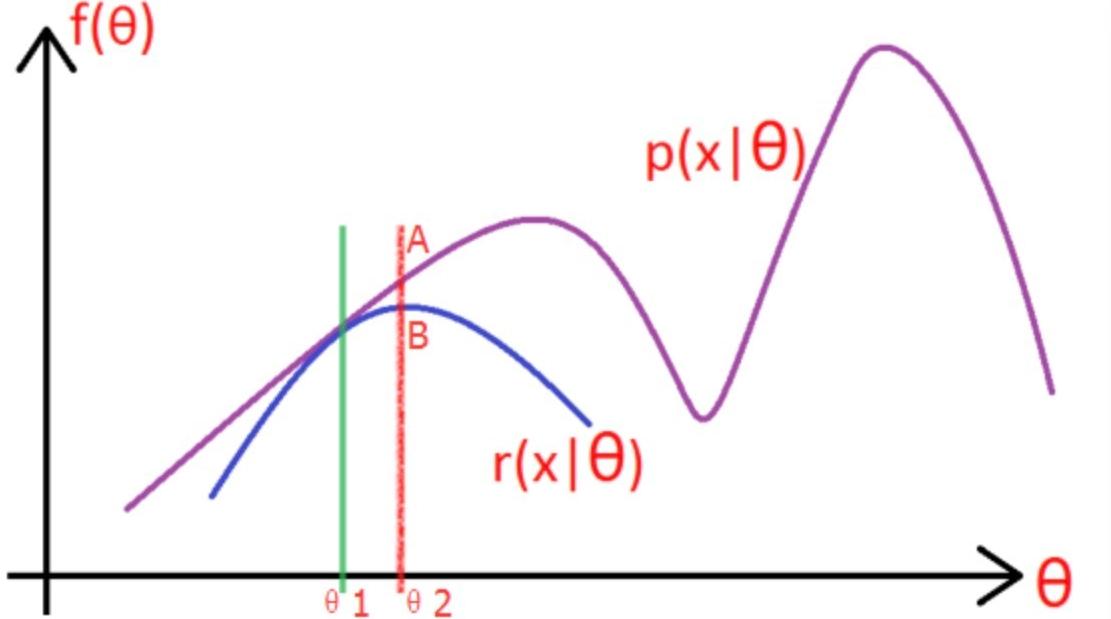
简化得：

$$\arg \max_{\theta} \sum_{i=1}^m \sum_{z^{(i)}} Q_i(z^{(i)}) \log P(x^{(i)}, z^{(i)}; \theta) \quad (151)$$

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。

### 2.20.3 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 $r \leq p$ .

### 2.20.4 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：

$$Q_i(z^{(i)}) = P(z^{(i)}|x^{(i)}, \theta^j) \quad (152)$$

$$L(\theta, \theta^j) = \sum_{i=1}^m \sum_{z^{(i)}} P(z^{(i)}|x^{(i)}, \theta^j) \log P(x^{(i)}, z^{(i)}; \theta) \quad (153)$$

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

$$\theta^{j+1} = \arg \max_{\theta} L(\theta, \theta^j) \quad (154)$$

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 $\theta$ 。

## 2.21 降维和聚类

### 2.21.1 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：

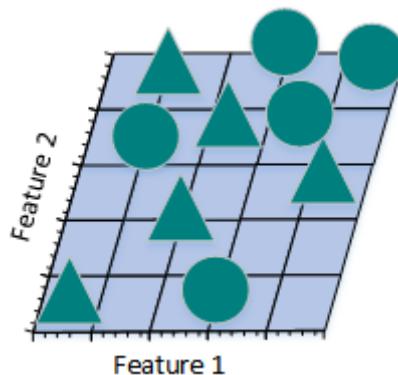


图2.21.1.b

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：

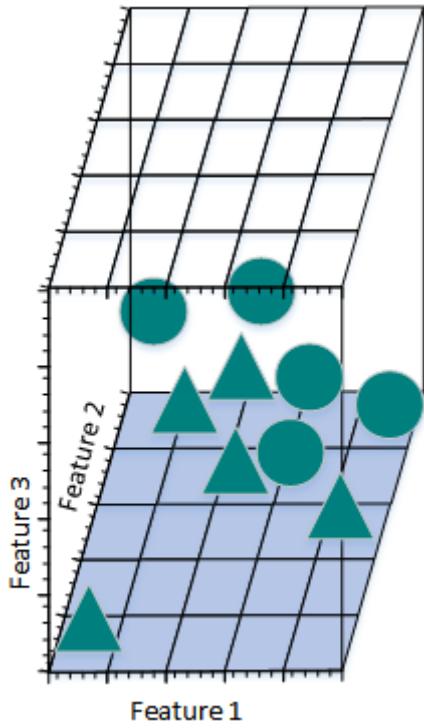


图2.21.1.c

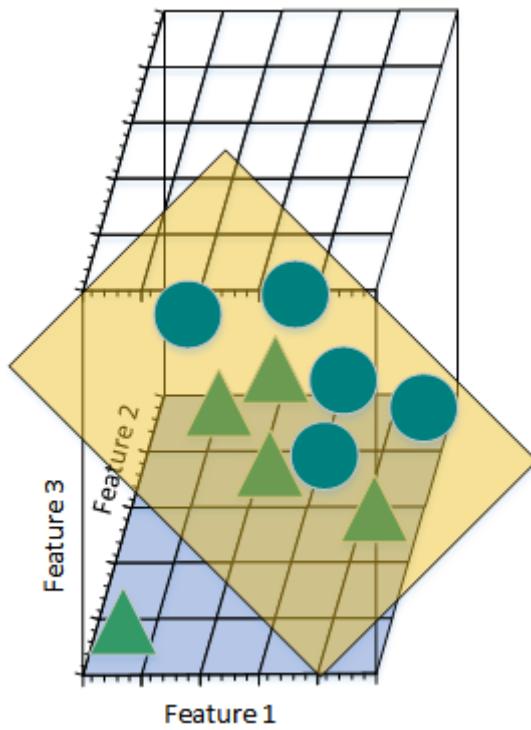


图2.21.1.d

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？

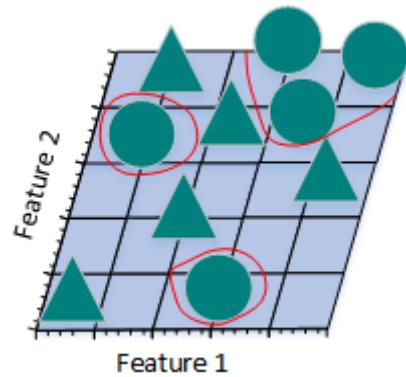


图2.21.1.e

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。

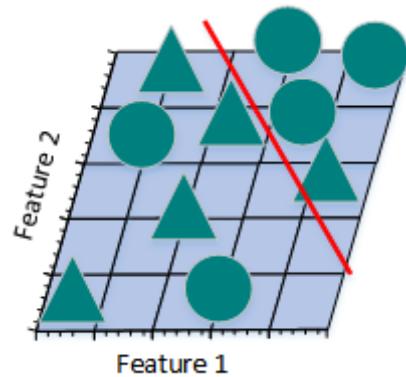
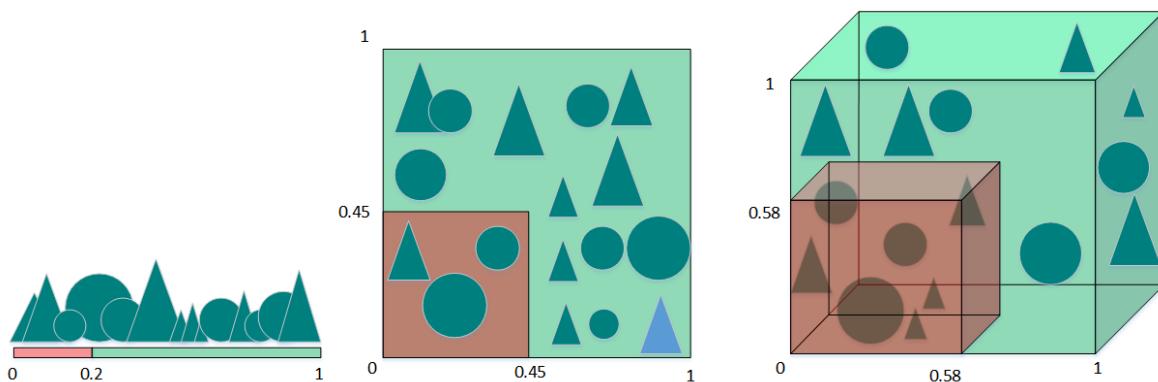


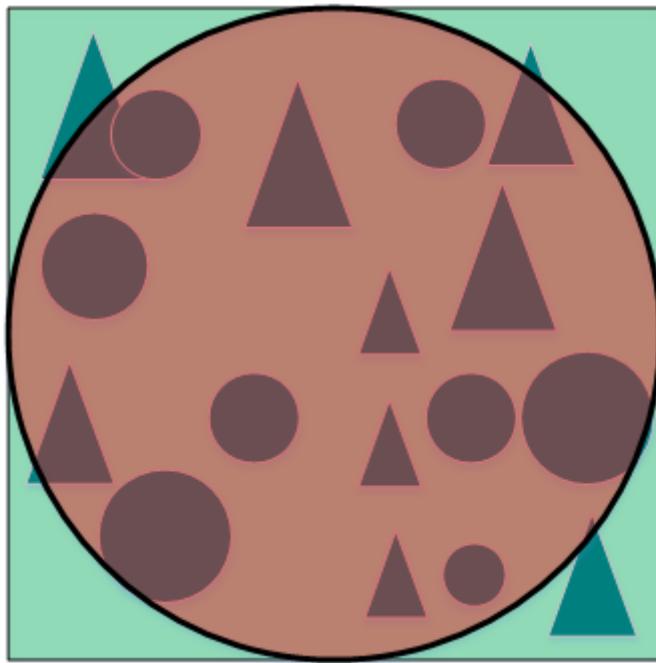
图2.21.1.f

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。

## 2.21.2 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

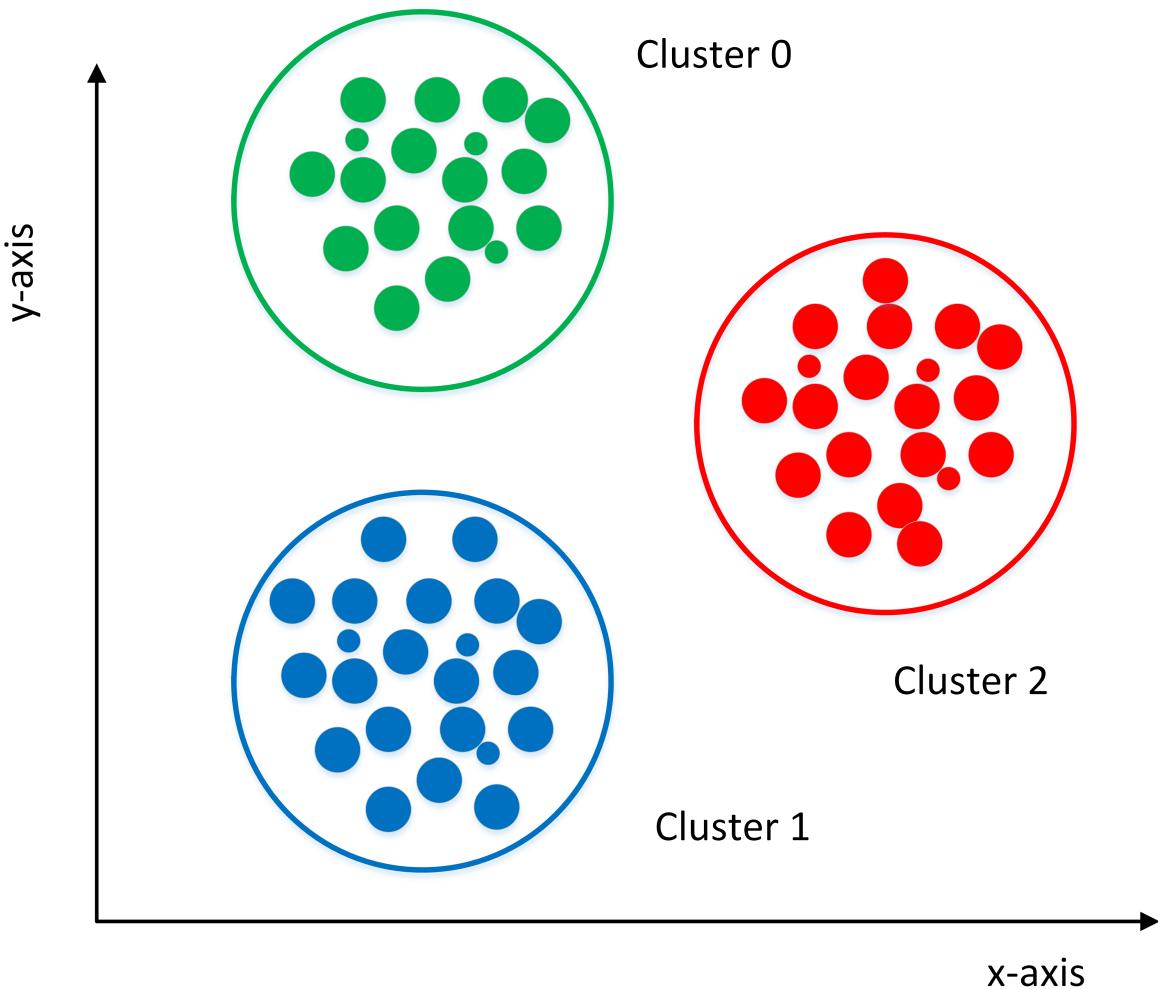
奇异值分解简化数据、拉普拉斯特征映射

Lasso缩减系数法、小波分析法、

## 2.21.3 聚类和降维有什么区别与联系

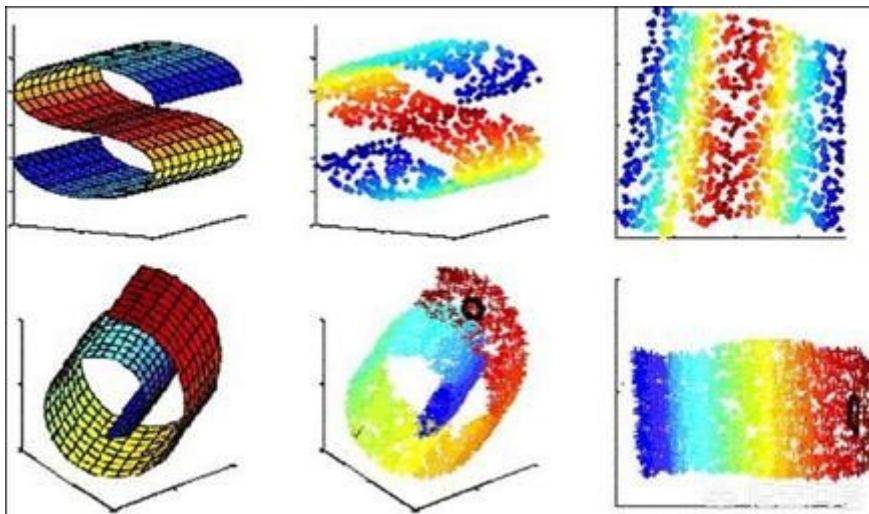
聚类用于找寻数据内在的分布结构，既可以作为一个单独的过程，比如异常检测等等。也可作为分类等其他学习任务的前驱过程。聚类是标准的无监督学习。

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。

## 2.21.4 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。

## 2.21.5 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。

## 2.21.6 不同聚类算法特点性能比较

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

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
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

## 2.21.7 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。

## 2.21.8 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：

$$E = \sum_{i=1}^k \sum_{p \in C_i} \|p - m_i\|^2 \quad (155)$$

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；

## 2.21.9 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。

## 2.21.10 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。

## 2.21.11 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：

$$\begin{cases} \sum_{i=1}^c u_{ij} = 1 & \forall j \\ u_{ij} \in [0, 1] & \forall i, j \\ \sum_{j=1}^n u_{ij} > 0 & \forall i \end{cases} \quad (156)$$

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：

$$(min) J_m(U, V) = \sum_{j=1}^n \sum_{i=1}^c u_{ij}^m d_{ij}^2(x_j, v_i) \quad (157)$$

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。

## 2.21.12 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： $avg = \frac{1}{k} \sum_{i=1}^k \frac{m_i}{n_i}$ 。

## 参考文献

---

- [1] Goodfellow I, Bengio Y, Courville A. Deep learning[M]. MIT press, 2016.
- [2] 周志华. 机器学习[M]. 清华大学出版社, 2016.
- [3] Michael A. Nielsen. "Neural Networks and Deep Learning", Determination Press, 2015.
- [4] Suryansh S. Gradient Descent: All You Need to Know, 2018.
- [5] 刘建平. 梯度下降小结,EM算法的推导, 2018
- [6] 杨小兵. 聚类分析中若干关键技术的研究[D]. 杭州：浙江大学, 2005.
- [7] XU Rui, Donald Wunsch 1 1. survey of clustering algorithm[J]. IEEE. Transactions on Neural Networks, 2005, 16(3): 645-67 8.
- [8] YI Hong, SAM K. Learning assignment order of instances for the constrained k-means clustering algorithm[J]. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 2009, 39 (2): 568-574.
- [9] 贺玲, 吴玲达, 蔡益朝. 数据挖掘中的聚类算法综述[J]. 计算机应用研究, 2007, 24(1):10-13.
- [10] 孙吉贵, 刘杰, 赵连宇. 聚类算法研究[J]. 软件学报, 2008, 19(1): 48-61.
- [11] 孔英会, 苑津莎, 张铁峰等. 基于数据流管理技术的配变负荷分类方法研究. 中国国际供电会议, CICED2006.
- [12] 马晓艳, 唐雁. 层次聚类算法研究[J]. 计算机科学, 2008, 34(7): 34-36.
- [13] FISHER R A. Iris Plants Database <https://www.ics.uci.edu/ml/MLRepository.html>, Authorized license.
- [14] Quinlan J R. Induction of decision trees[J]. Machine learning, 1986, 1(1): 81-106.
- [15] Breiman L. Random forests[J]. Machine learning, 2001, 45(1): 5-32.

## 第三章 深度学习基础

---

## 3.1 基本概念

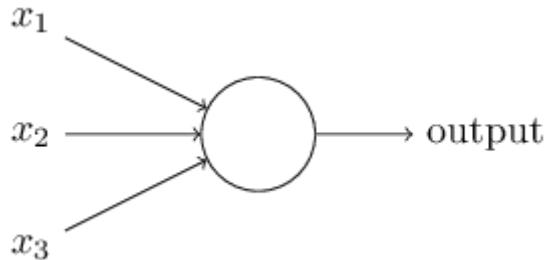
### 3.1.1 神经网络组成?

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：

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

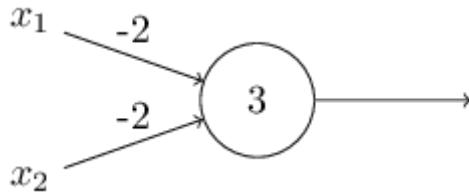
假如把感知机想象成一个加权投票机制，比如3位评委给一个歌手打分，打分分别为4分、1分、-3分，这3位评分的权重分别是1、3、2，则该歌手最终得分为 $4 \times 1 + 1 \times 3 + (-3) \times 2 = 1$ 。按照比赛规则，选取的 threshold 为 3，说明只有歌手的综合评分大于3时，才可顺利晋级。对照感知机，该选手被淘汰，因为：

$$\sum_i w_i x_i < \text{threshold} = 3, \text{output} = 0 \quad (158)$$

用 $-b$ 代替 threshold，输出变为：

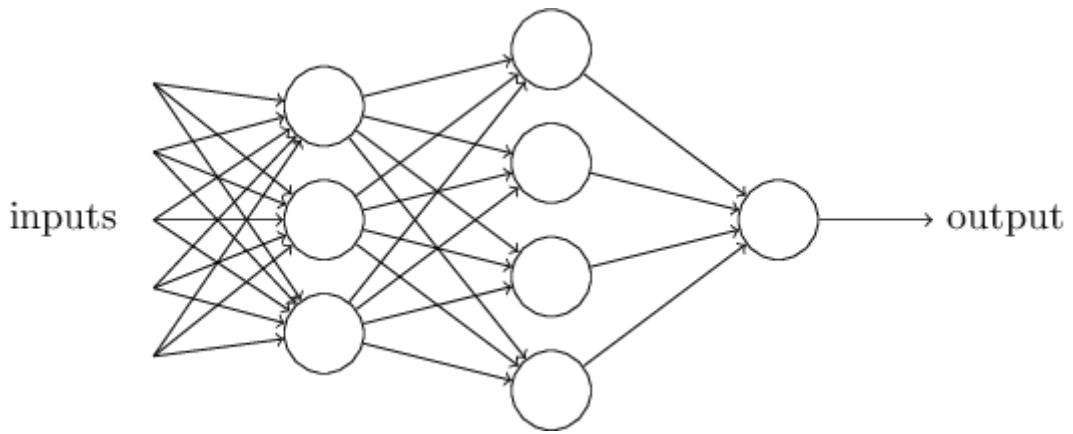
$$\text{output} = \begin{cases} 0, & \text{if } \mathbf{w} \cdot \mathbf{x} + b \leq 0 \\ 1, & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \end{cases}$$

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：



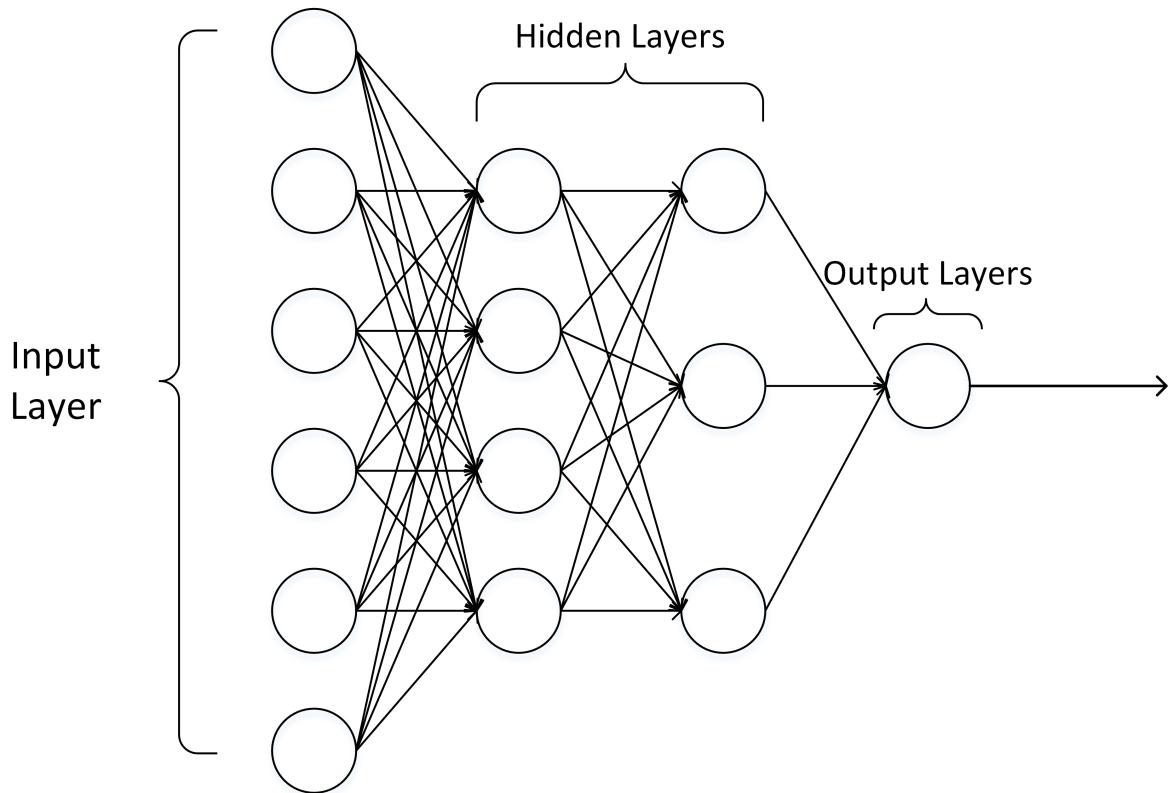
当输入为 0, 1 时，感知机输出为  $0 \times (-2) + 1 \times (-2) + 3 = 1$ 。

复杂一些的感知机由简单的感知机单元组合而成：

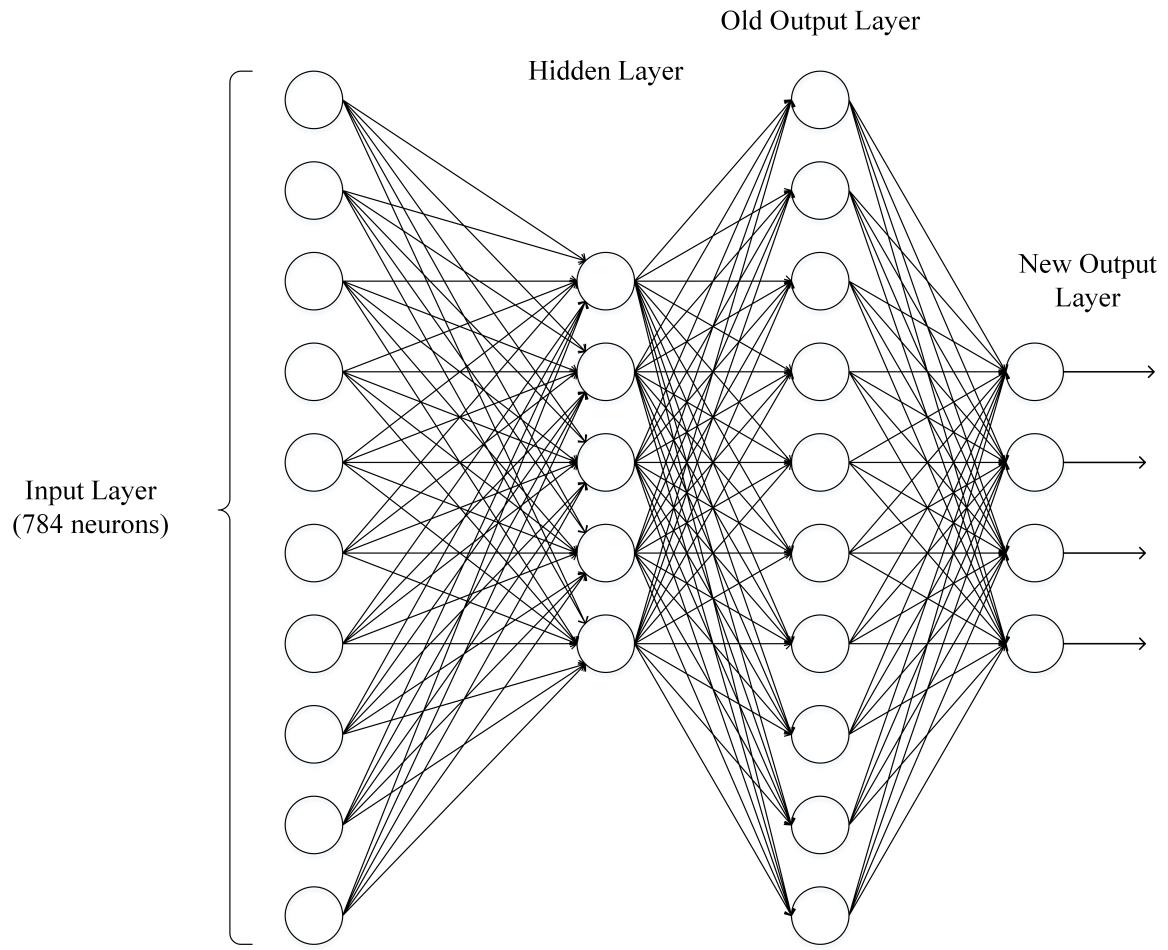


### 多层感知机

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：



### 3.1.2 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:

# A mostly complete chart of Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org

Backfed Input Cell

Input Cell

Noisy Input Cell

Hidden Cell

Probabilistic Hidden Cell

Spiking Hidden Cell

Output Cell

Match Input Output Cell

Recurrent Cell

Memory Cell

Different Memory Cell

Kernel

Convolution or Pool

Perceptron (P)

Feed Forward (FF)

Radial Basis Network (RBF)

Recurrent Neural Network (RNN)

Long / Short Term Memory (LSTM)

Gated Recurrent Unit (GRU)

Auto Encoder (AE)

Variational AE (VAE)

Denoising AE (DAE)

Sparse AE (SAE)

Markov Chain (MC)

Hopfield Network (HN)

Boltzmann Machine (BM)

Restricted BM (RBM)

Deep Belief Network (DBN)

Deep Convolutional Network (DCN)

Deconvolutional Network (DN)

Deep Convolutional Inverse Graphics Network (DCIGN)

Generative Adversarial Network (GAN)

Liquid State Machine (LSM)

Extreme Learning Machine (ELM)

Echo State Network (ESN)

Deep Residual Network (DRN)

Kohonen Network (KN)

Support Vector Machine (SVM)

Neural Turing Machine (NTM)

## 3.1.3 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。

### **3.1.4 为什么使用深层表示？**

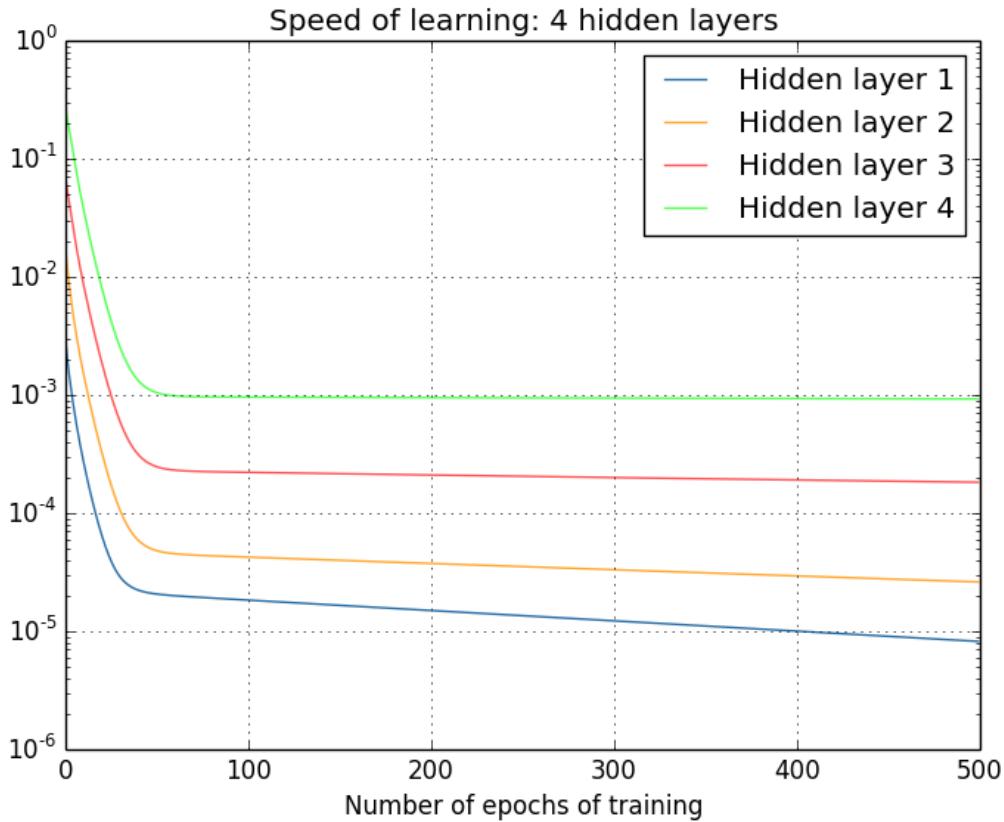
1. 深度神经网络是一种特征递进式的学习算法，浅层的神经元直接从输入数据中学习一些低层次的简单特征，例如边缘、纹理等。而深层的特征则基于已学习到的浅层特征继续学习更高级的特征，从计算机的角度学习深层的语义信息。
2. 深层的网络隐藏单元数量相对较少，隐藏层数目较多，如果浅层的网络想要达到同样的计算结果则需要指数级增长的单元数量才能达到。

### **3.1.5 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：



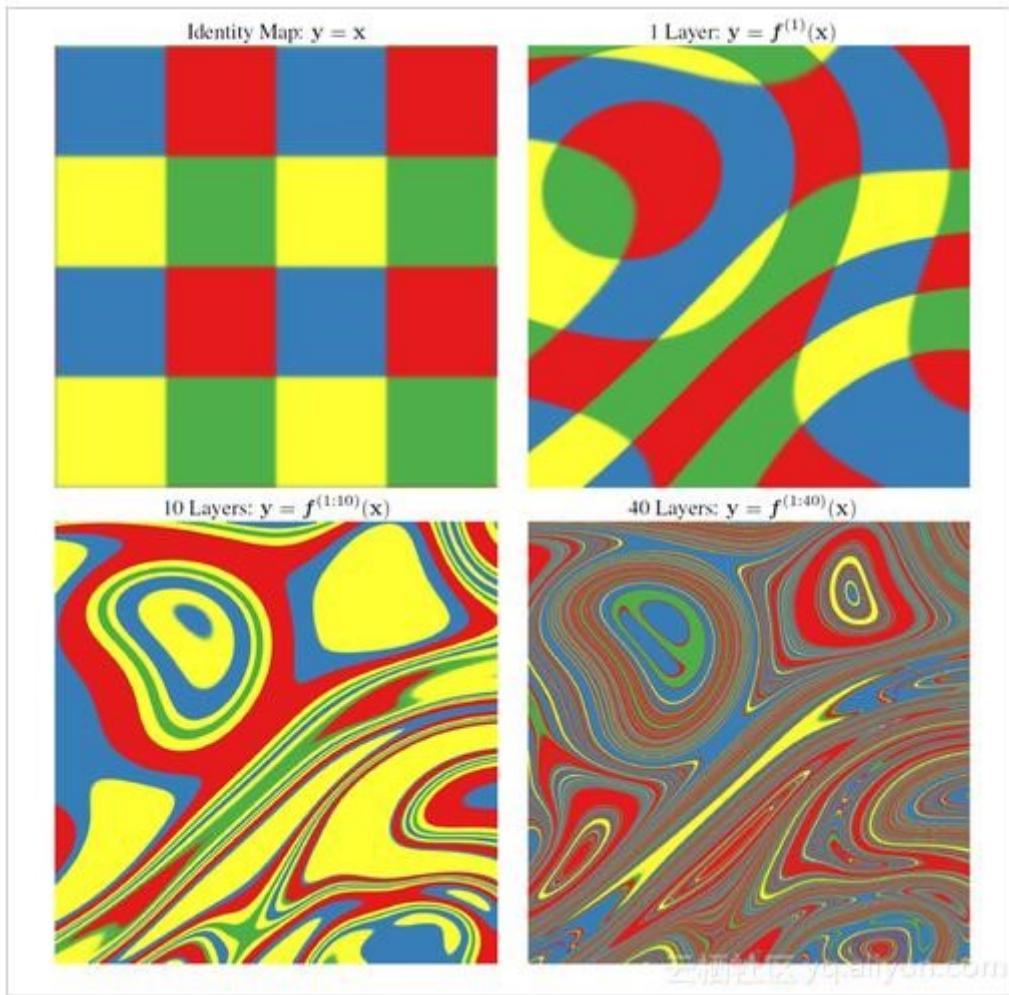
## 2. 梯度爆炸

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：



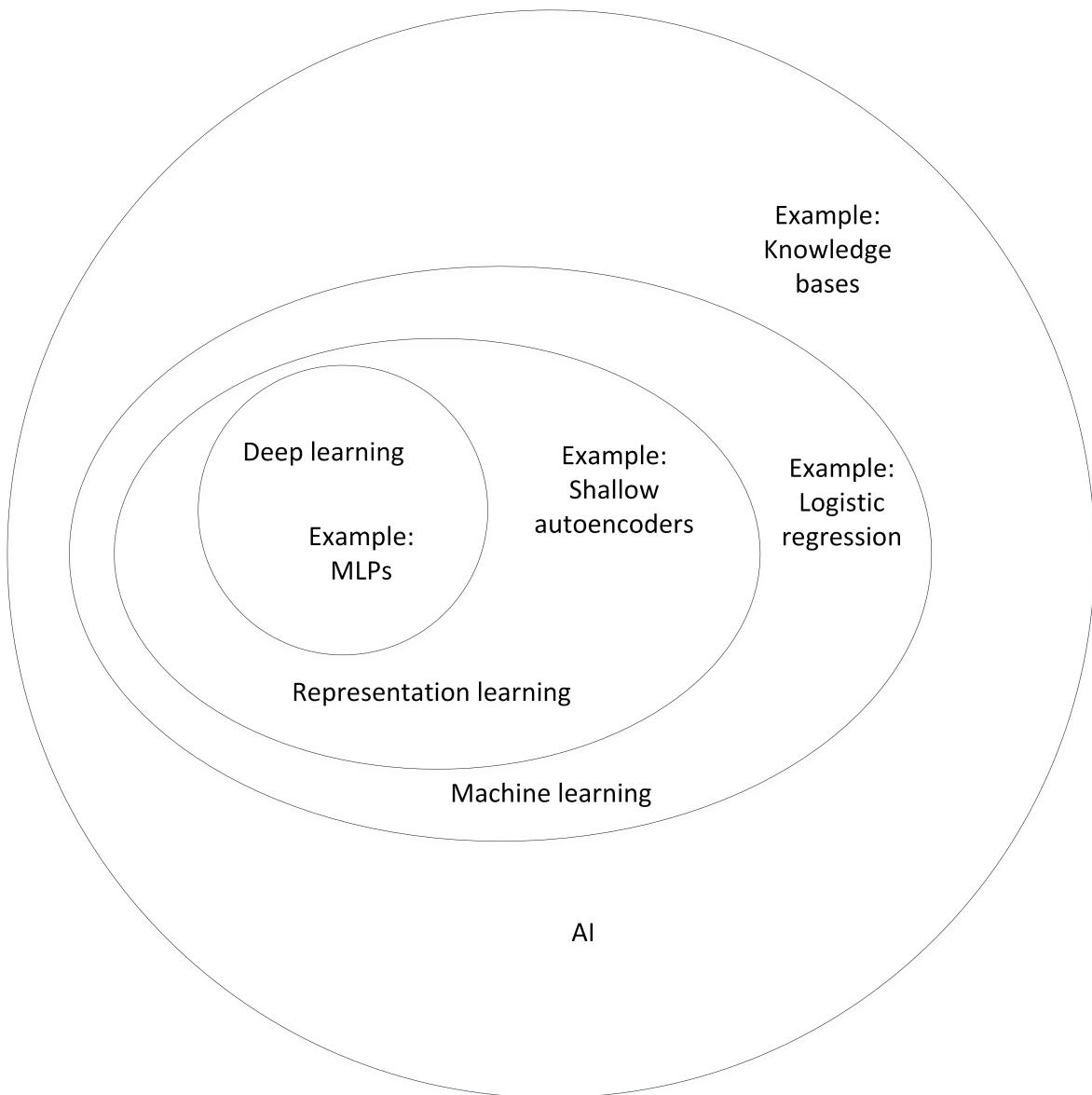
随着深度的增加，输入空间（左上角所示）会在输入空间中的每个点处被扭曲成越来越细的单丝，只有一个与细丝正交的方向影响网络的响应。沿着这个方向，网络实际上对变化变得非常敏感。

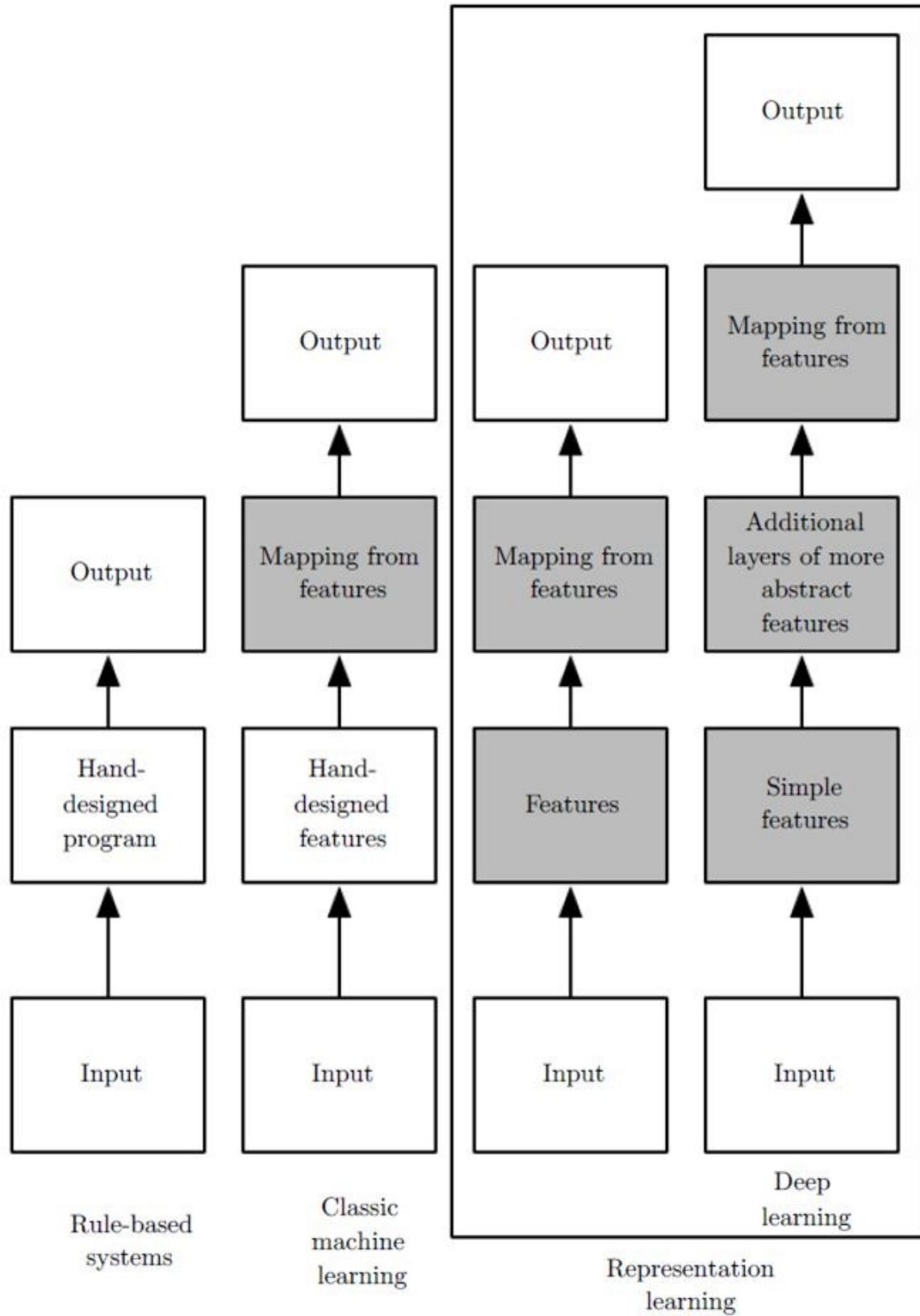
### 3.1.6 深度学习和机器学习有什么不同？

**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。



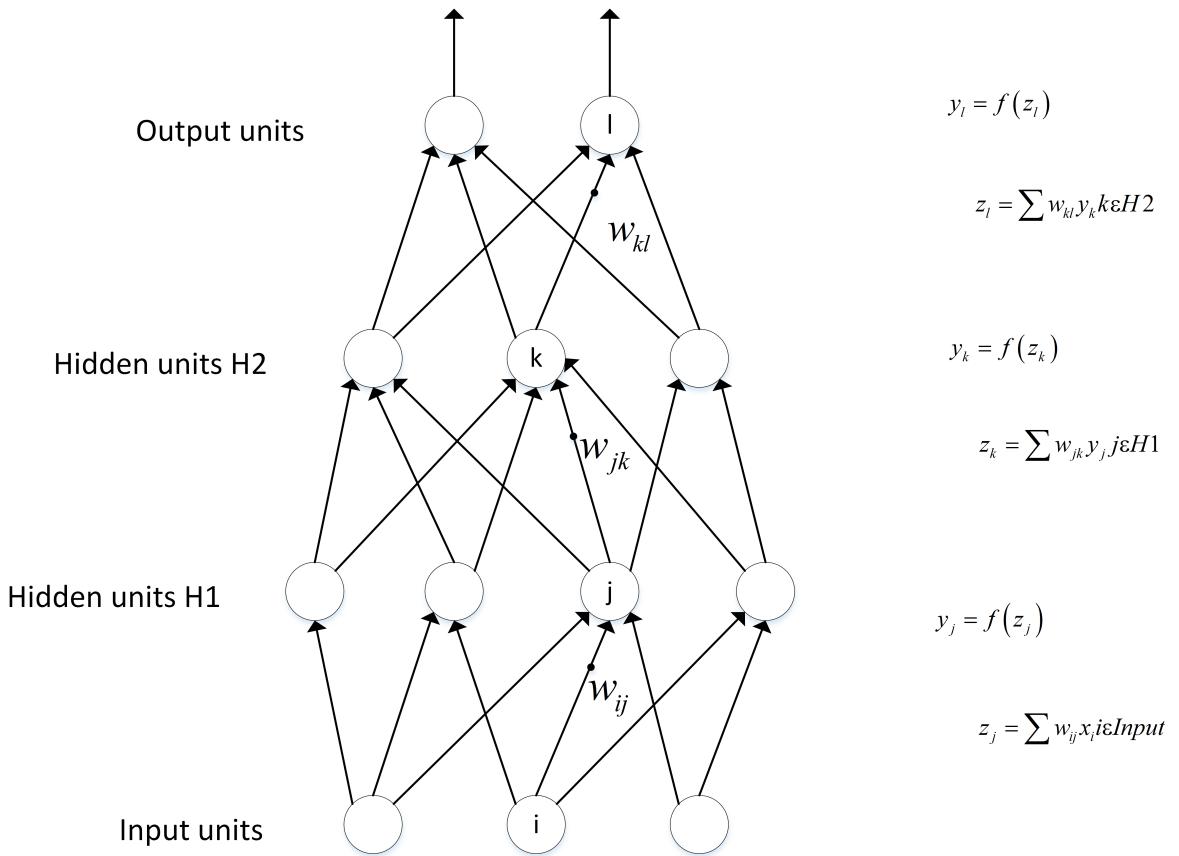


## 3.2 网络操作与计算

### 3.2.1 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。

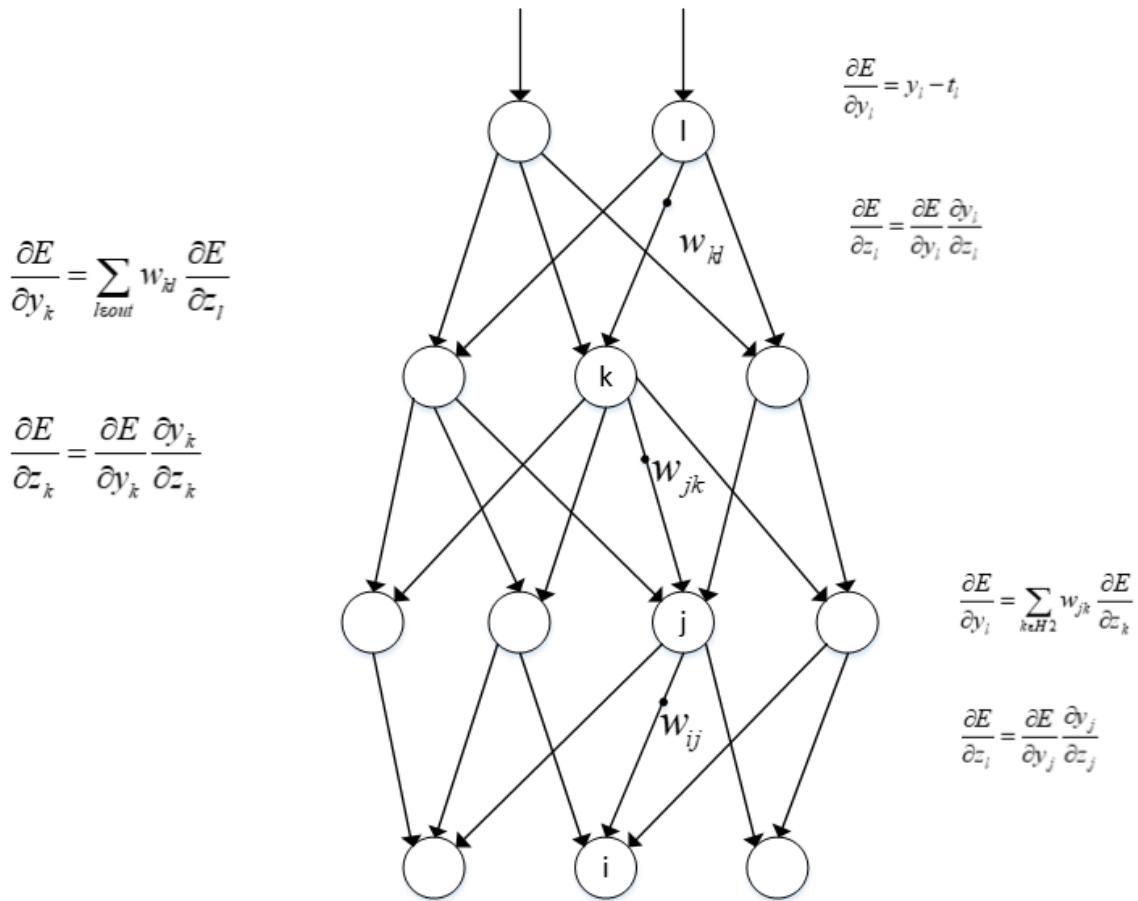
**前向传播**



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。

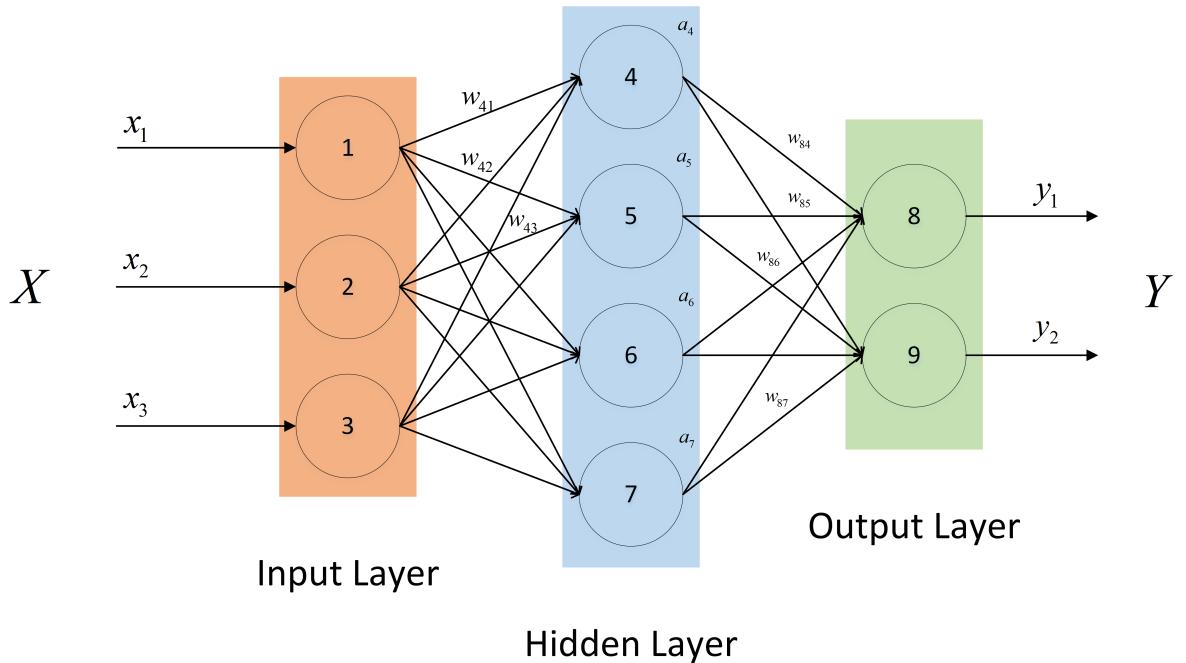
### 反向传播



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。

### 3.2.2 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  $x_1, x_2, x_3$ 。

$$a_4 = \sigma(w^T \cdot a) = \sigma(w_{41}x_4 + w_{42}x_2 + w_{43}x_3 + w_{4b}) \quad (159)$$

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  $y_1$ ：

$$y_1 = \sigma(w^T \cdot a) = \sigma(w_{84}a_4 + w_{85}a_5 + w_{86}a_6 + w_{87}a_7 + w_{8b}) \quad (160)$$

其中  $w_{8b}$  是节点 8 的偏置项。

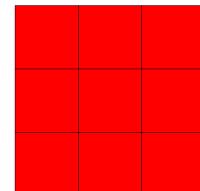
同理，我们还可以计算出  $y_2$ 。这样输出层所有节点的输出值计算完毕，我们就得到了在输入向量  $x_1, x_2, x_3, x_4$  时，神经网络的输出向量  $y_1, y_2$ 。这里我们也看到，输出向量的维度和输出层神经元个数相同。

### 3.2.3 如何计算卷积神经网络输出值？

假设有一个  $5*5$  的图像，使用一个  $3*3$  的 filter 进行卷积，想得到一个  $3*3$  的 Feature Map，如下所示：

1	1	1	0	0
0	1	1	0	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

1	0	1
0	1	0
1	0	1



Bias=0

Image 5\*5

filter 3\*3

Feature map 3\*3

$x_{i,j}$  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：

$$a_{i,j} = f\left(\sum_{m=0}^2 \sum_{n=0}^2 w_{m,n} x_{i+m, j+n} + w_b\right) \quad (161)$$

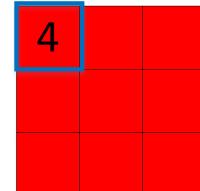
当步长为 1 时，计算 feature map 元素  $a_{0,0}$  如下：

$$\begin{aligned} a_{0,0} &= f\left(\sum_{m=0}^2 \sum_{n=0}^2 w_{m,n} x_{0+m, 0+n} + w_b\right) = \text{relu}(w_{0,0}x_{0,0} + w_{0,1}x_{0,1} + w_{0,2}x_{0,2} + w_{1,0}x_{1,0} + \\ &\quad w_{1,1}x_{1,1} + w_{1,2}x_{1,2} + w_{2,0}x_{2,0} + w_{2,1}x_{2,1} + w_{2,2}x_{2,2}) \\ &= 1 + 0 + 1 + 0 + 1 + 0 + 0 + 0 + 1 \\ &= 4 \end{aligned} \quad (162)$$

其计算过程图示如下：

1	1	1	0	0
0	1	1	0	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

1	0	1
0	1	0
1	0	1



Bias=0

Image 5\*5

filter 3\*3

Feature map 3\*3

以此类推，计算出全部的 Feature Map。

1	1	1	0	0
0	1	1	0	0
0	0	$\frac{1}{x_1}$	$\frac{1}{x_0}$	$\frac{1}{x_1}$
0	0	$\frac{1}{x_0}$	$\frac{1}{x_1}$	$\frac{0}{x_0}$
0	1	$\frac{1}{x_1}$	$\frac{0}{x_0}$	$\frac{0}{x_1}$

4	3	4
2	4	3
2	3	4

Image

Convolved  
Feature

当步幅为 2 时，Feature Map 计算如下

1	1	1	0	0
0	1	1	0	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

1	0	1
0	1	0
1	0	1

4	4
2	4

Bias=0

Image 5\*5

filter 3\*3

Feature map 2\*2

注：图像大小、步幅和卷积后的Feature Map 大小是有关系的。它们满足下面的关系：

$$W_2 = (W_1 - F + 2P)/S + 1 \quad (163)$$

$$H_2 = (H_1 - F + 2P)/S + 1$$

其中  $W_2$ ，是卷积后 Feature Map 的宽度； $W_1$  是卷积前图像的宽度； $F$  是 filter 的宽度； $P$  是 Zero Padding 数量，Zero Padding 是指在原始图像周围补几圈 0，如果  $P$  的值是 1，那么就补 1 圈 0； $S$  是步幅； $H_2$  卷积后 Feature Map 的高度； $H_1$  是卷积前图像的高度。

举例：假设图像宽度  $W_1 = 5$ ，filter 宽度  $F = 3$ ，Zero Padding  $P = 0$ ，步幅  $S = 2$ ， $Z$  则

$$W_2 = (W_1 - F + 2P)/S + 1 = (5 - 3 + 0)/2 + 1 = 2 \quad (164)$$

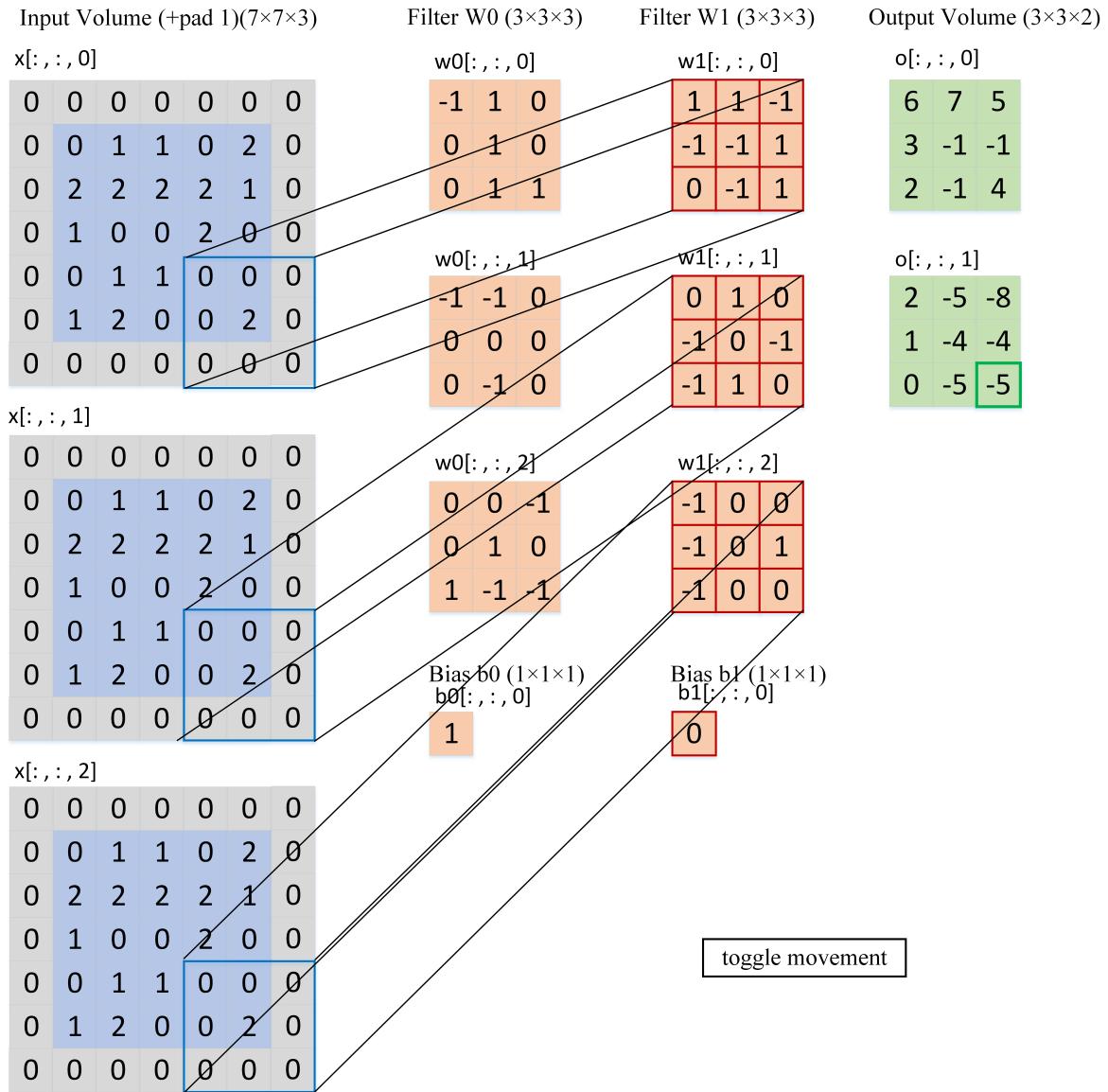
说明 Feature Map 宽度是 2。同样，我们也可以计算出 Feature Map 高度也是 2。

如果卷积前的图像深度为  $D$ ，那么相应的 filter 的深度也必须为  $D$ 。深度大于 1 的卷积计算公式：

$$a_{i,j} = f(\sum_{d=0}^{D-1} \sum_{m=0}^{F-1} \sum_{n=0}^{F-1} w_{d,m,n} x_{d,i+m,j+n} + w_b) \quad (165)$$

其中， $D$  是深度； $F$  是 filter 的大小； $w_{d,m,n}$  表示 filter 的第  $d$  层第  $m$  行第  $n$  列权重； $a_{d,i,j}$  表示 feature map 的第  $d$  层第  $i$  行第  $j$  列像素；其它的符号含义前面相同，不再赘述。

每个卷积层可以有多个 filter。每个 filter 和原始图像进行卷积后，都可以得到一个 Feature Map。卷积后 Feature Map 的深度(个数)和卷积层的 filter 个数相同。下面的图示显示了包含两个 filter 的卷积层的计算。 $7 \times 7 \times 3$  输入，经过两个  $3 \times 3 \times 3$  filter 的卷积(步幅为 2)，得到了  $3 \times 3 \times 2$  的输出。图中的 Zero padding 是 1，也就是在输入元素的周围补了一圈 0。

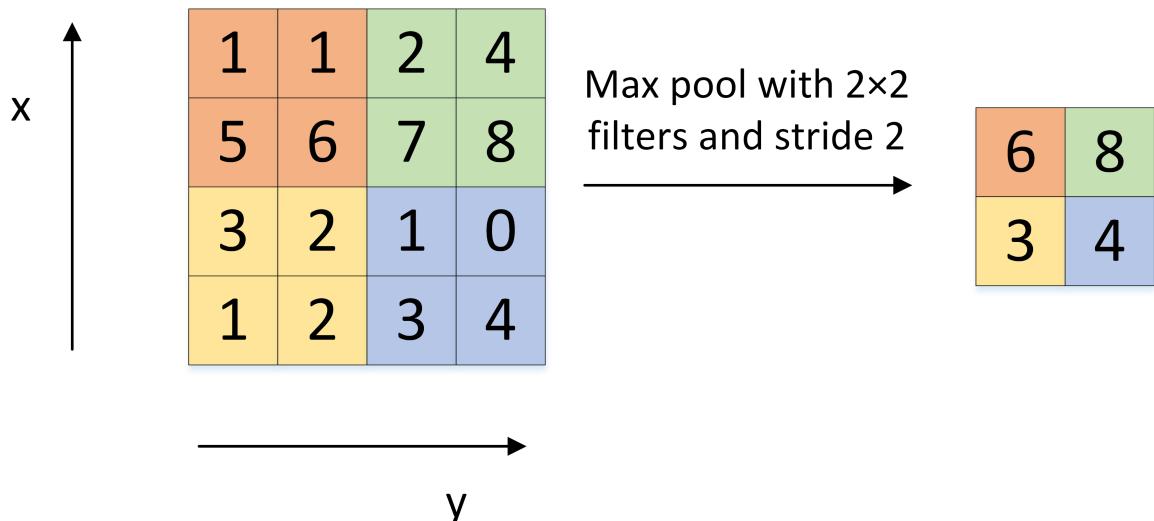


以上就是卷积层的计算方法。这里面体现了局部连接和权值共享：每层神经元只和上一层部分神经元相连(卷积计算规则)，且 filter 的权值对于上一层所有神经元都是一样的。对于包含两个  $3 \times 3 \times 3$  的 filter 的卷积层来说，其参数数量仅有  $(3 \times 3 \times 3 + 1) * 2 = 56$  个，且参数数量与上一层神经元个数无关。与全连接神经网络相比，其参数数量大大减少了。

### 3.2.4 如何计算 Pooling 层输出值输出值？

Pooling 层主要的作用是下采样，通过去掉 Feature Map 中不重要的样本，进一步减少参数数量。Pooling 的方法很多，最常用的是 Max Pooling。Max Pooling 实际上就是在  $n \times n$  的样本中取最大值，作为采样后的样本值。下图是  $2 \times 2$  max pooling：

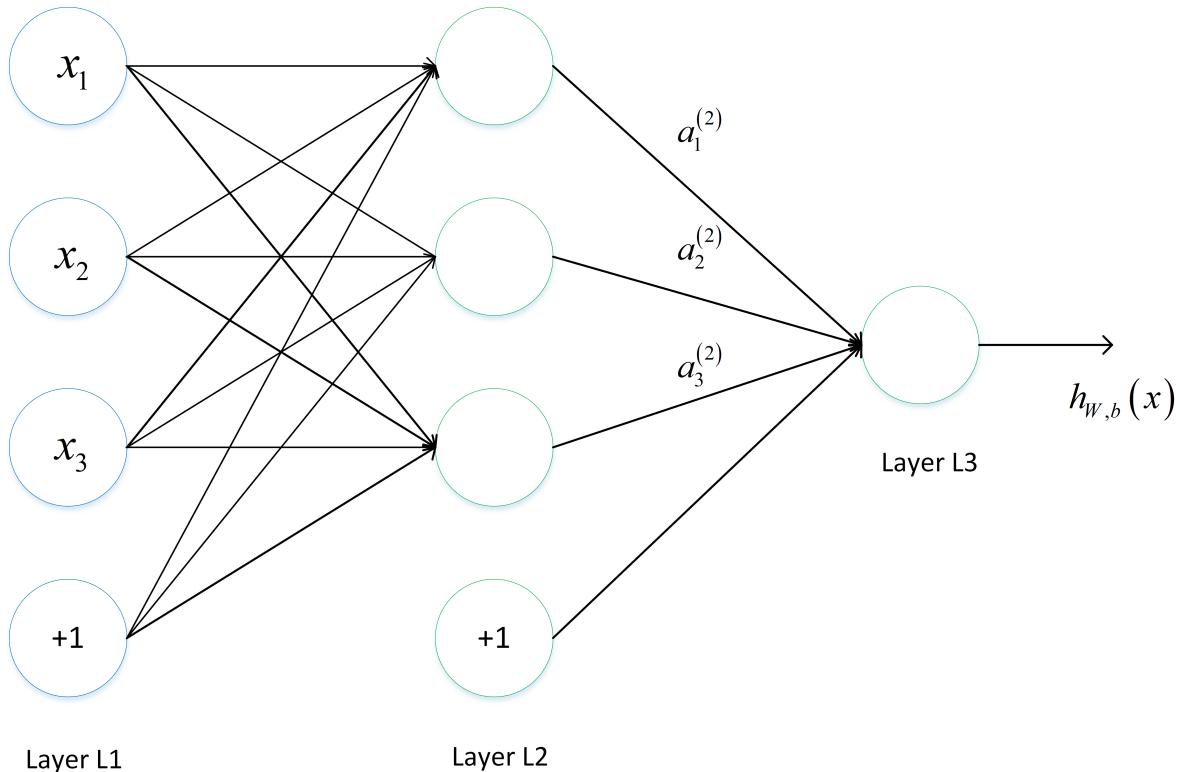
## Single depth slice



除了 Max Pooling 之外，常用的还有 Average Pooling —— 取各样本的平均值。  
对于深度为  $D$  的 Feature Map，各层独立做 Pooling，因此 Pooling 后的深度仍然为  $D$ 。

### 3.2.5 实例理解反向传播

一个典型的三层神经网络如下所示：

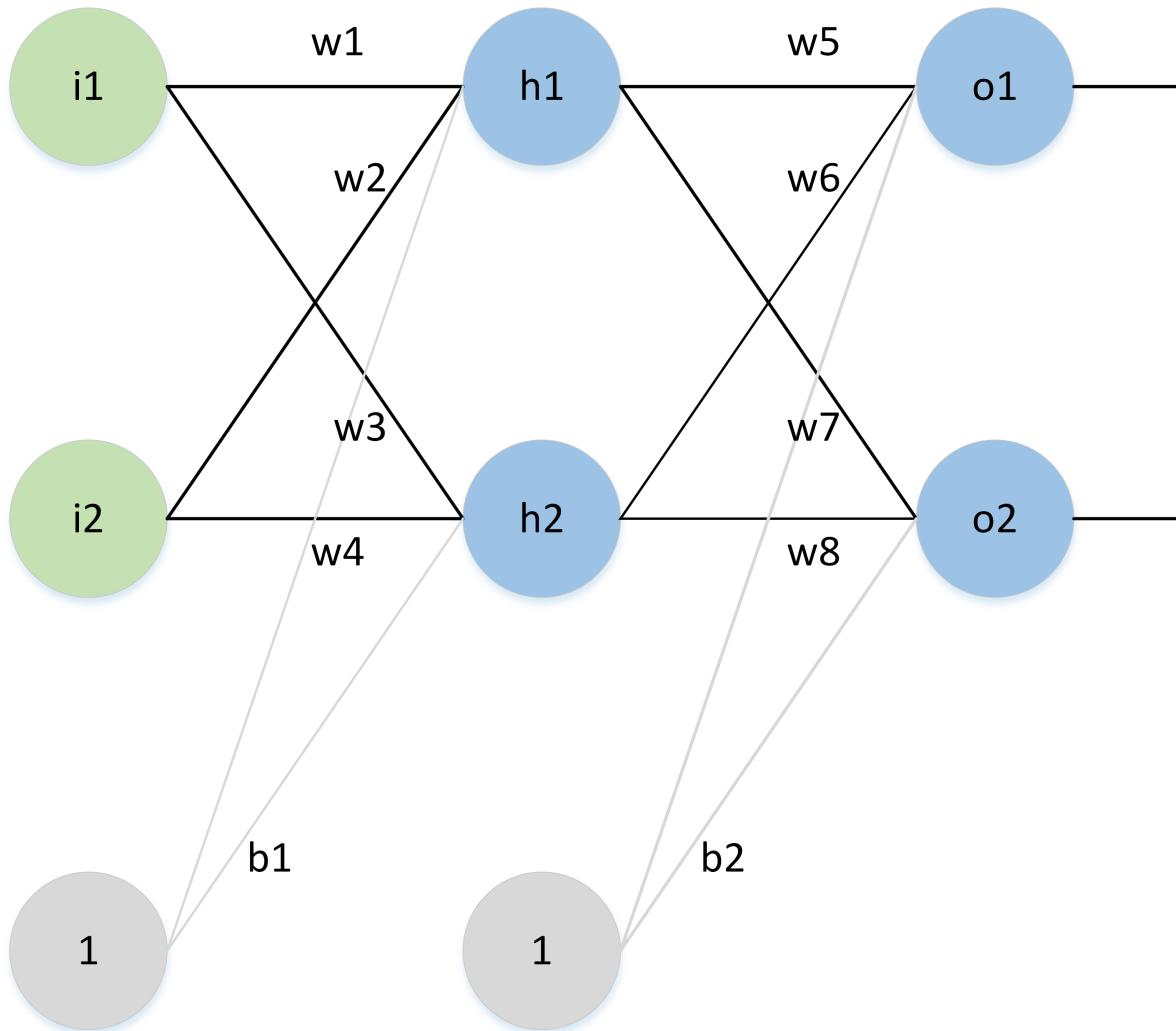


其中 Layer  $L_1$  是输入层，Layer  $L_2$  是隐含层，Layer  $L_3$  是输出层。

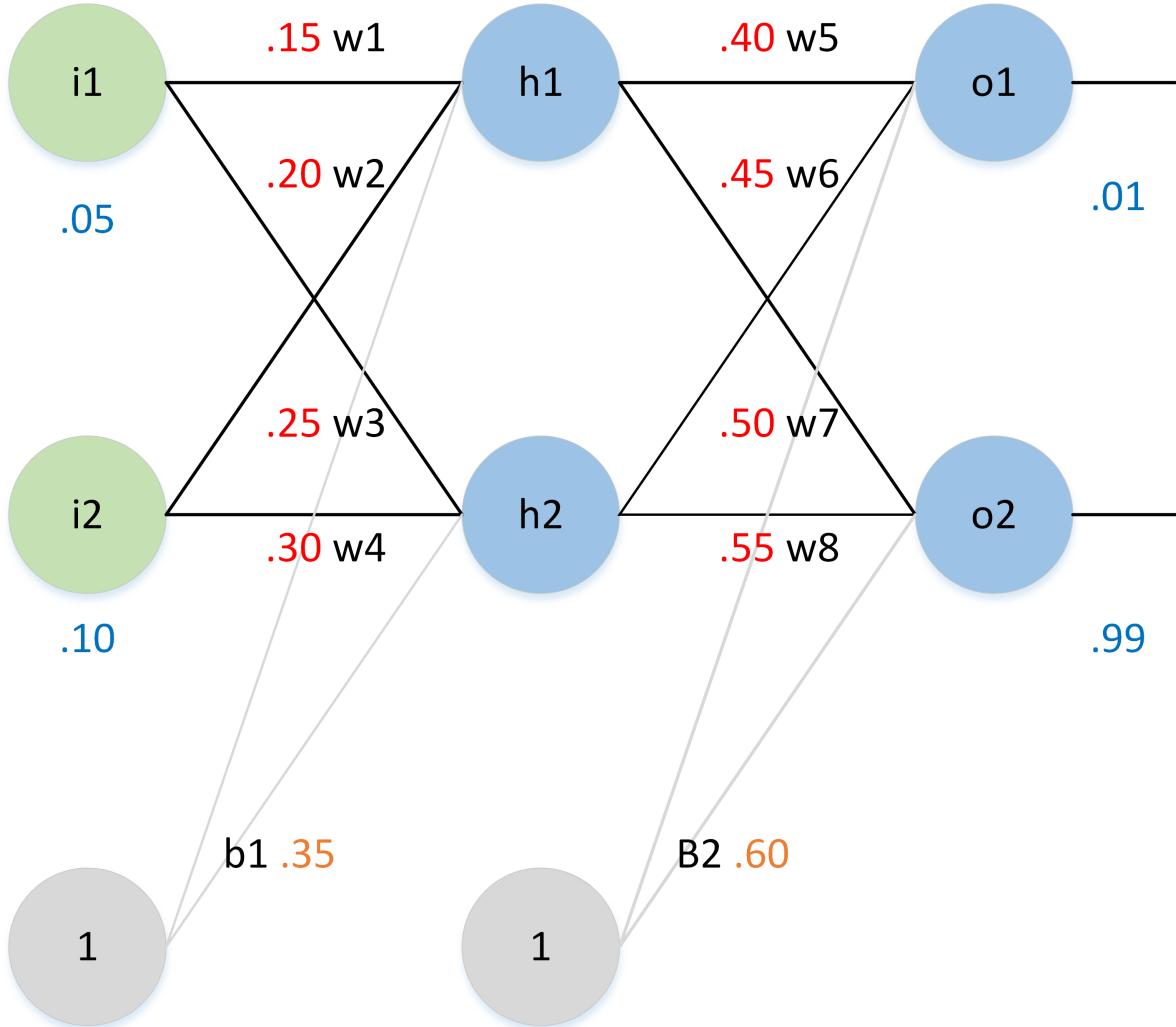
假设输入数据集为  $D = x_1, x_2, \dots, x_n$ ，输出数据集为  $y_1, y_2, \dots, y_n$ 。

如果输入和输出是一样，即为自编码模型。如果原始数据经过映射，会得到不同于输入的输出。

假设有如下的网络层：



输入层包含神经元  $i_1, i_2$ , 偏置  $b_1$ ; 隐含层包含神经元  $h_1, h_2$ , 偏置  $b_2$ , 输出层为  $o_1, o_2$ ,  $w_i$  为层与层之间连接的权重, 激活函数为  $sigmoid$  函数。对以上参数取初始值, 如下图所示:



其中：

- 输入数据  $i_1 = 0.05, i_2 = 0.10$
- 输出数据  $o_1 = 0.01, o_2 = 0.99$ ;
- 初始权重  
 $w_1 = 0.15, w_2 = 0.20, w_3 = 0.25, w_4 = 0.30, w_5 = 0.40, w_6 = 0.45, w_7 = 0.50, w_8 = 0.55$
- 目标：给出输入数据  $i_1, i_2$  (0.05和0.10)，使输出尽可能与原始输出  $o_1, o_2$ ，(0.01和0.99)接近。

### 前向传播

1. 输入层 --> 输出层

计算神经元  $h_1$  的输入加权和：

$$\begin{aligned} net_{h1} &= w_1 * i_1 + w_2 * i_2 + b_1 * 1 & (166) \\ net_{h1} &= 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775 \end{aligned}$$

神经元  $h_1$  的输出  $o_1$ ：(此处用到激活函数为 sigmoid 函数)：

$$out_{h1} = \frac{1}{1 + e^{-net_{h1}}} = \frac{1}{1 + e^{-0.3775}} = 0.593269992 \quad (167)$$

同理，可计算出神经元  $h_2$  的输出  $o_1$ ：

$$out_{h2} = 0.596884378 \quad (168)$$

2. 隐含层-->输出层：

计算输出层神经元  $o_1$  和  $o_2$  的值：

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1 \quad (169)$$

$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967 \quad (170)$$

$$out_{o1} = \frac{1}{1 + e^{-net_{o1}}} = \frac{1}{1 + e^{1.105905967}} = 0.75136079 \quad (171)$$

这样前向传播的过程就结束了，我们得到输出值为  $[0.75136079, 0.772928465]$ ，与实际值  $[0.01, 0.99]$  相差还很远，现在我们对误差进行反向传播，更新权值，重新计算输出。

## 反向传播

### 1. 计算总误差

总误差：(这里使用Square Error)

$$E_{total} = \sum \frac{1}{2} (target - output)^2 \quad (172)$$

但是有两个输出，所以分别计算  $o1$  和  $o2$  的误差，总误差为两者之和：

$$E_{o1} = \frac{1}{2} (target_{o1} - out_{o1})^2 = \frac{1}{2} (0.01 - 0.75136507)^2 = 0.274811083.$$

$$E_{o2} = 0.023560026.$$

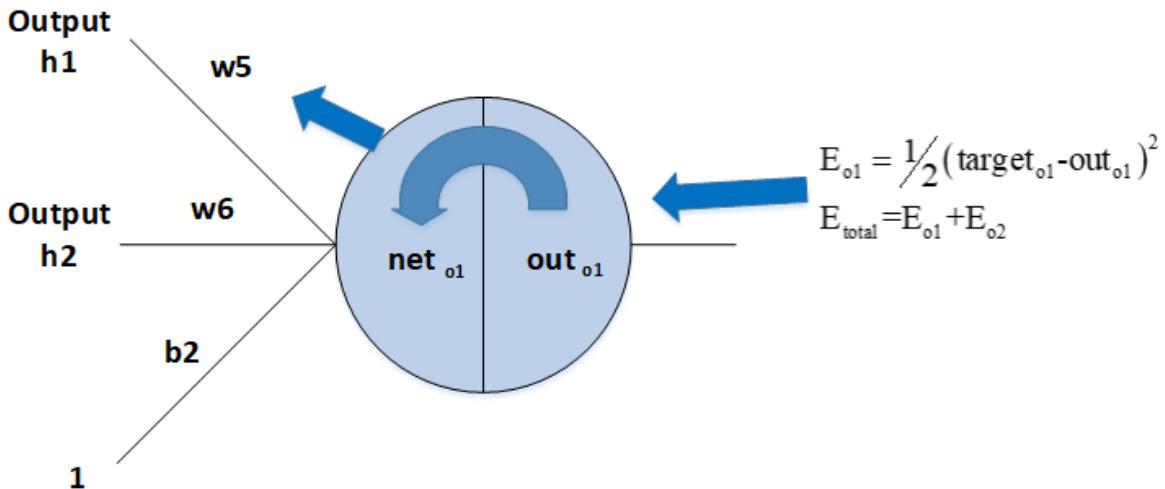
$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109.$$

### 2. 隐含层 --> 输出层的权值更新：

以权重参数  $w5$  为例，如果我们想知道  $w5$  对整体误差产生了多少影响，可以用整体误差对  $w5$  求偏导求出：(链式法则)

$$\frac{\partial E_{total}}{\partial w5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w5} \quad (173)$$

下面的图可以更直观的看清楚误差是怎样反向传播的：



## 3.2.6 神经网络更“深”有什么意义？

前提：在一定范围内。

- 在神经元数量相同的情况下，深层网络结构具有更大容量，分层组合带来的是指数级的表达空间，能够组合成更多不同类型的子结构，这样可以更容易地学习和表示各种特征。
- 隐藏层增加则意味着由激活函数带来的非线性变换的嵌套层数更多，就能构造更复杂的映射关系。

## 3.3 超参数

### 3.3.1 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。

### 3.3.2 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LIGO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。

### 3.3.3 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。

## 3.4 激活函数

### 3.4.1 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?

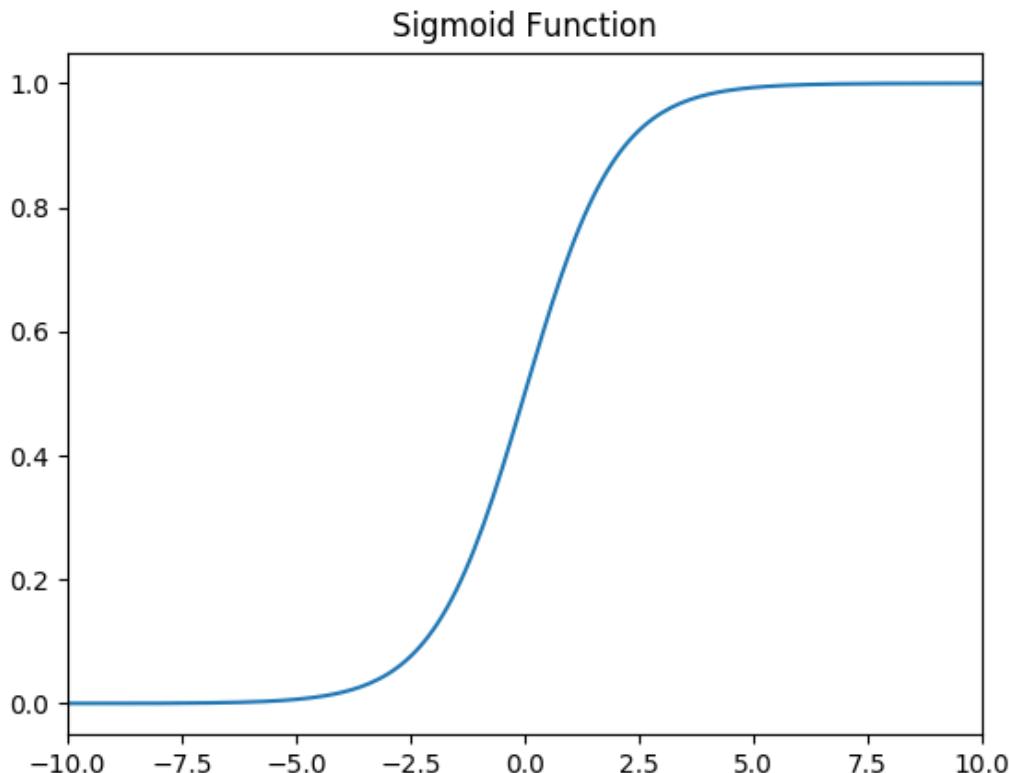
1. 假若网络中全部是线性部件，那么线性的组合还是线性，与单独一个线性分类器无异。这样就做不到用非线性来逼近任意函数。
2. 使用非线性激活函数，以便使网络更加强大，增加它的能力，使它可以学习复杂的事物，复杂的表单数据，以及表示输入输出之间非线性的复杂的任意函数映射。使用非线性激活函数，能够从输入输出之间生成非线性映射。

## 3.4.2 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:

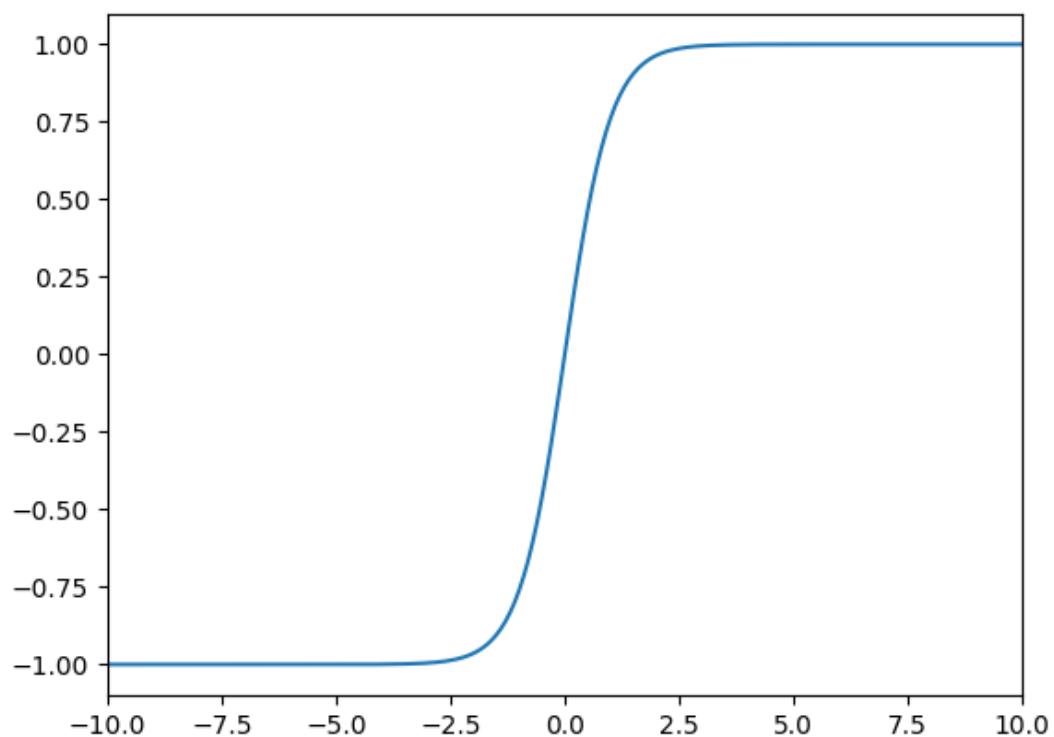


### 2. tanh激活函数

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:

Tanh Function

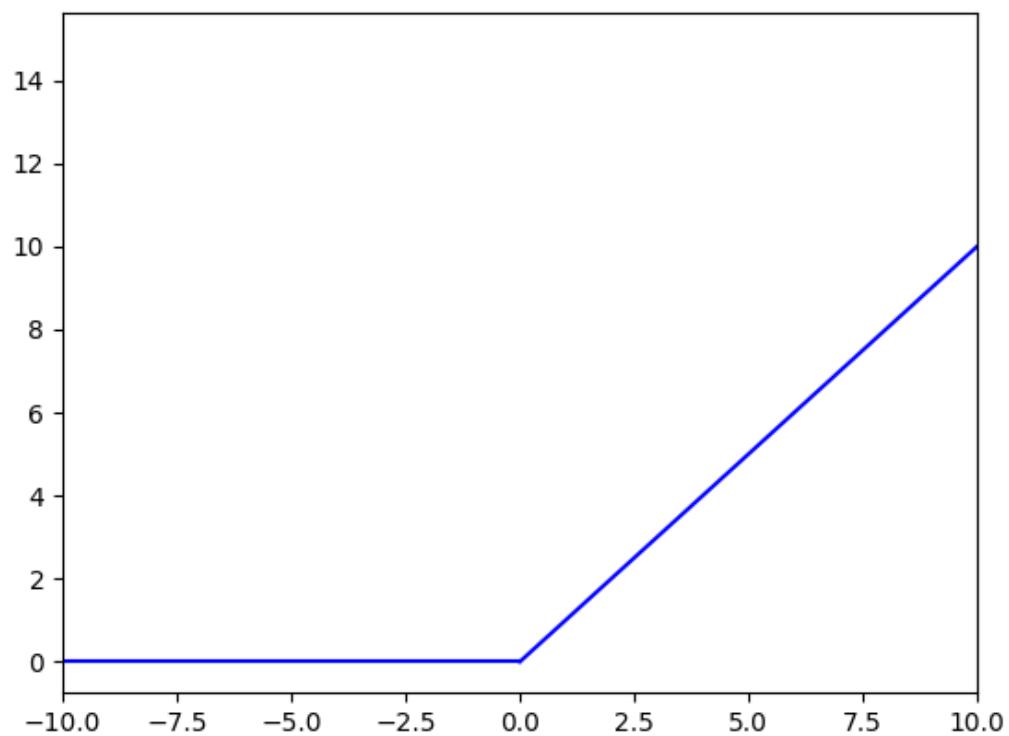


### 3. Relu激活函数

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

函数图像如下:

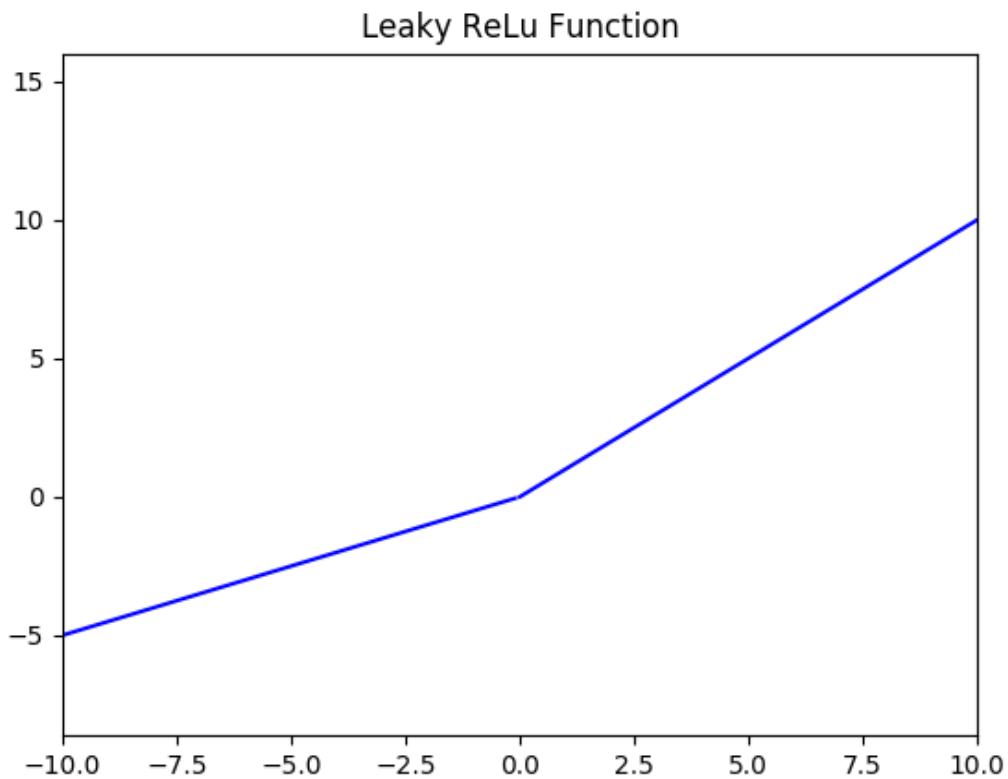
Relu Function



### 4. Leak Relu 激活函数

函数定义为:  $f(x) = \begin{cases} ax, & x < 0 \\ x, & x \geq 0 \end{cases}$ , 值域为  $(-\infty, +\infty)$ 。

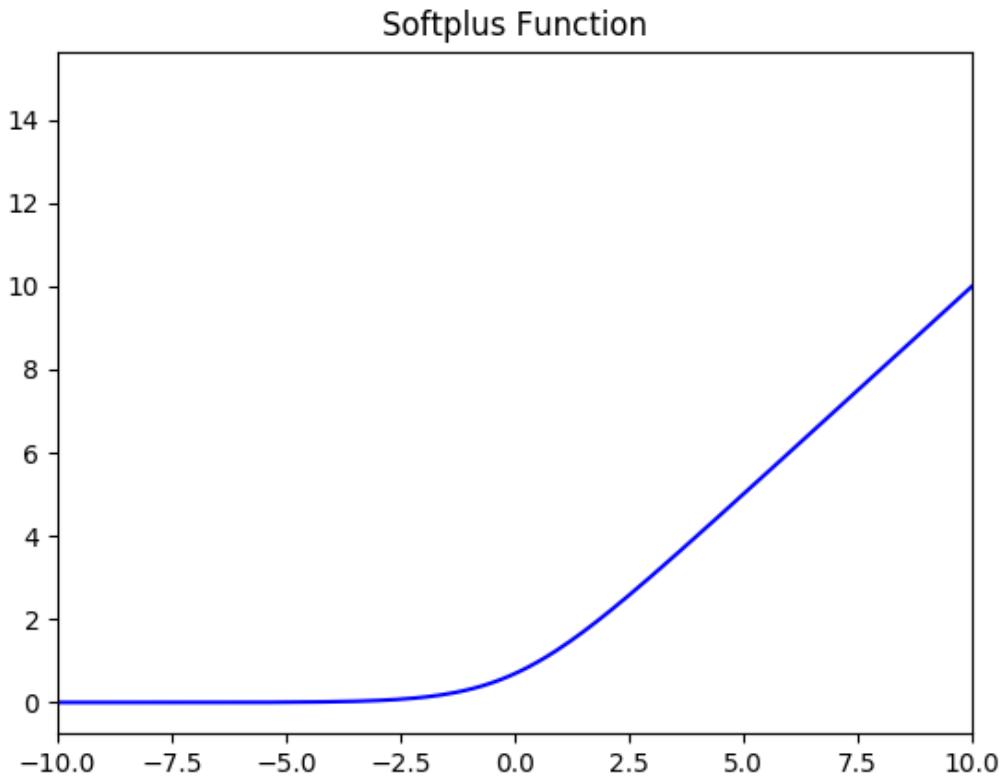
图像如下 ( $a = 0.5$ ) :



## 5. SoftPlus 激活函数

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

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。

### 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 激活函数	$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = \frac{1}{1+e^{-x}} \left(1 - \frac{1}{1+e^{-x}}\right) = f(x)(1 - f(x))$	当 $x =$ $f'($ $x =$ $f'($
Tanh 激活函数	$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$f'(x) = -(tanh(x))^2$	当 $x =$ $f'($ $x =$ $f'($
Relu 激活函数	$f(x) = \max(0, x)$	$c(u) = \begin{cases} 0, & x < 0 \\ 1, & x > 0 \\ undefined, & x = 0 \end{cases}$	通常 时, 数

### 3.4.4 激活函数有哪些性质?

1. 非线性：当激活函数是线性的，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.  $f(x) \approx x$ ：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。

### 3.4.5 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。

### 3.4.6 使用 ReLu 激活函数的优点？

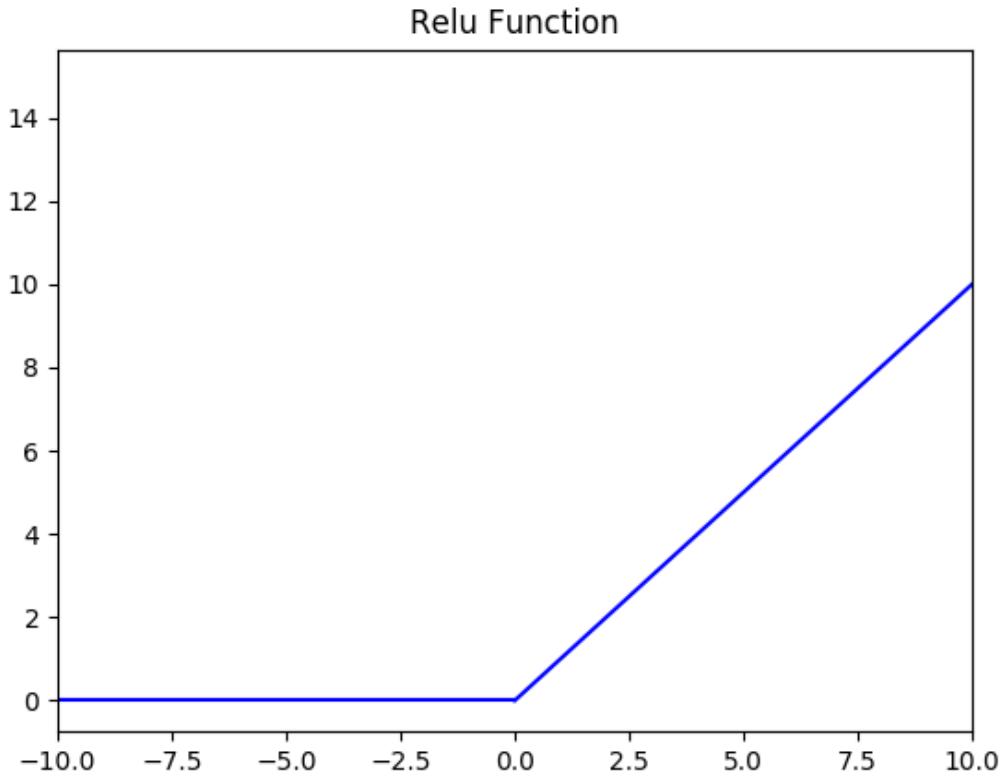
1. 在区间变动很大的情况下，ReLu 激活函数的导数或者激活函数的斜率都会远大于 0，在程序实现就是一个 if-else 语句，而 sigmoid 函数需要进行浮点四则运算，在实践中，使用 ReLu 激活函数神经网络通常会比使用 sigmoid 或者 tanh 激活函数学习的更快。
2. sigmoid 和 tanh 函数的导数在正负饱和区的梯度都会接近于 0，这会造成梯度弥散，而 Relu 和 Leaky ReLu 函数大于 0 部分都为常数，不会产生梯度弥散现象。
3. 需注意，Relu 进入负半区的时候，梯度为 0，神经元此时不会训练，产生所谓的稀疏性，而 Leaky ReLu 不会产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

### 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。

### 3.4.8 怎样理解 Relu (< 0 时) 是非线性激活函数？

Relu 激活函数图像如下：



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。

### 3.4.9 Softmax 定义及作用

Softmax 是一种形如下式的函数：

$$P(i) = \frac{\exp(\theta_i^T x)}{\sum_{k=1}^K \exp(\theta_k^T x)} \quad (174)$$

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：

$$P(i) = \frac{1}{1 + \exp(-\theta_i^T x)} \quad (175)$$

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于 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。

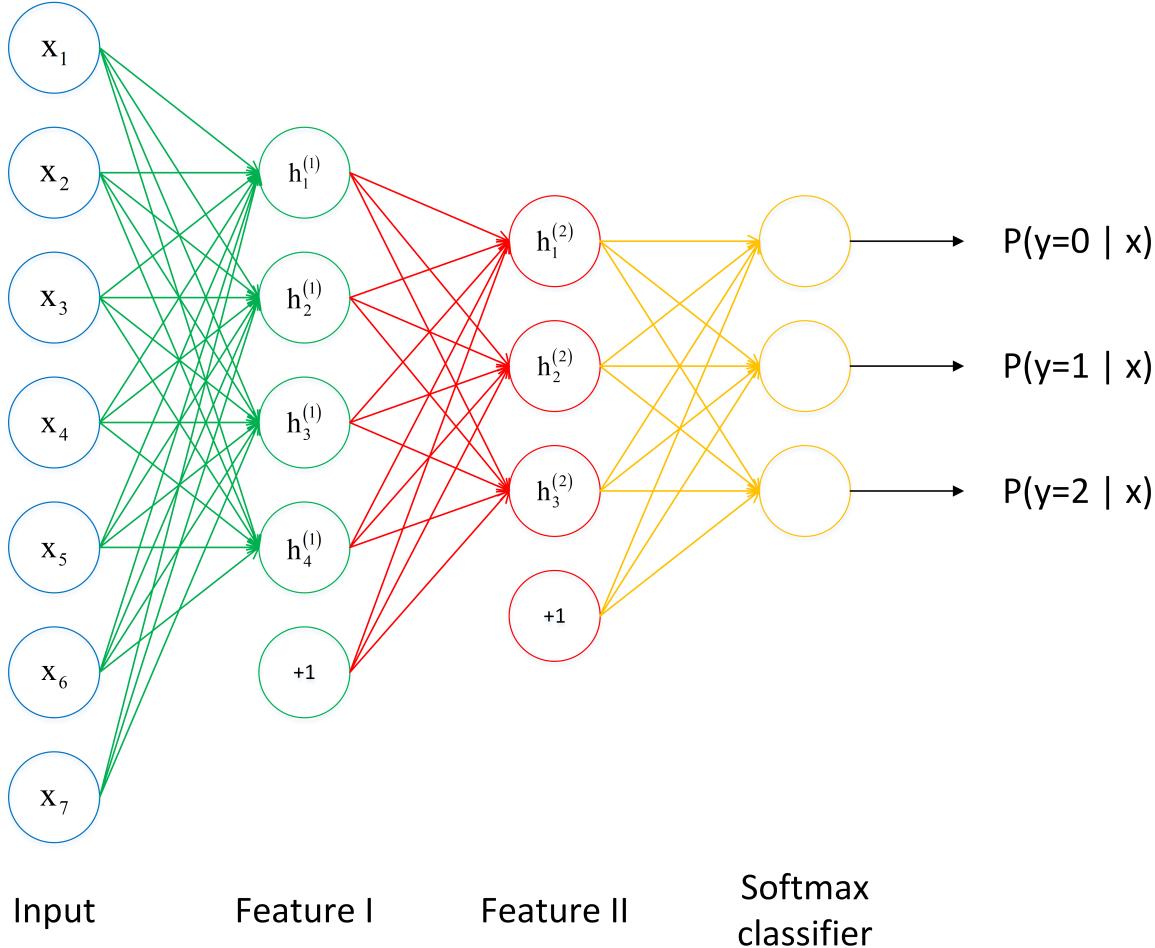
### 3.4.10 Softmax 函数如何应用于多分类?

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!

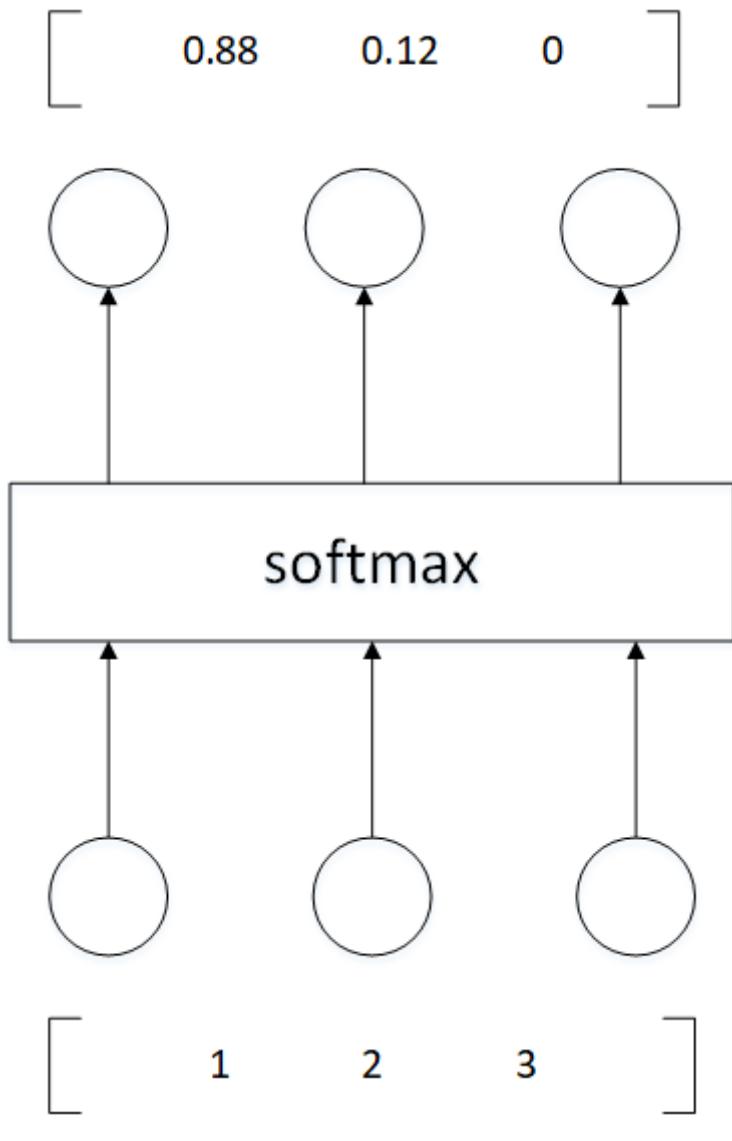
假设我们有一个数组,  $V_i$  表示  $V$  中的第  $i$  个元素, 那么这个元素的 softmax 值就是

$$S_i = \frac{e^{V_i}}{\sum_j e^{V_j}} \quad (176)$$

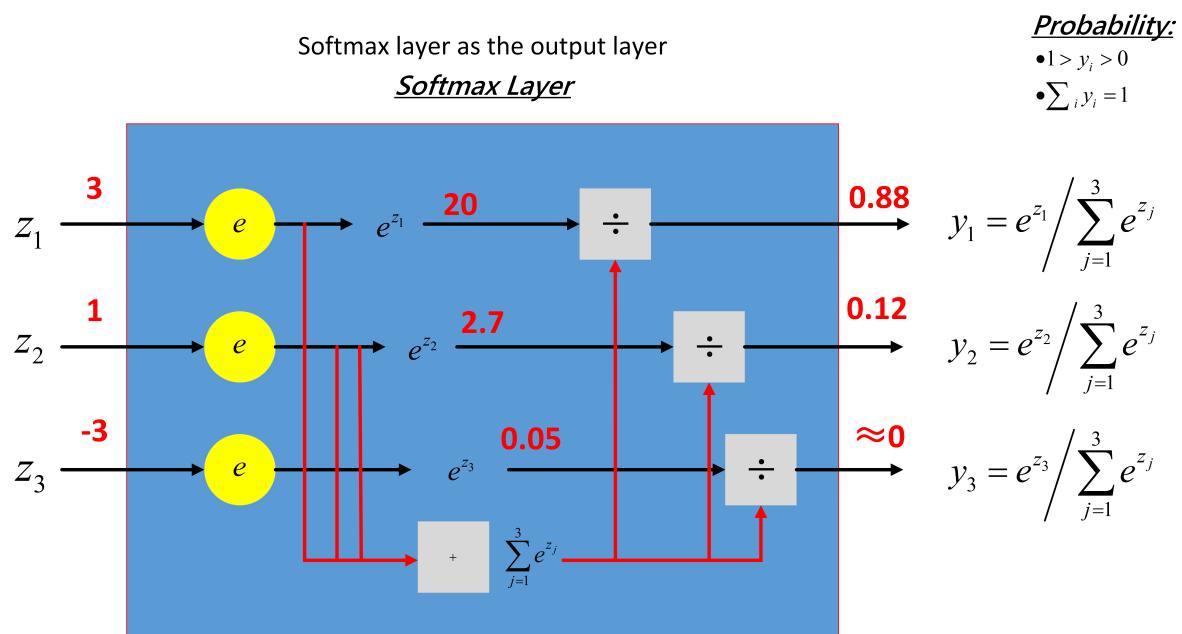
从下图看, 神经网络中包含了输入层, 然后通过两个特征层处理, 最后通过 softmax 分析器就能得到不同条件下的概率, 这里需要分成三个类别, 最终会得到  $y = 0, y = 1, y = 2$  的概率值。



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！

### 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  $a \approx 0$ 。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  $-\ln(1 - a) \approx 0$ 。反之,  $y = 1$  而  $a \approx 1$ 。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:

$$\frac{\partial C}{\partial w_j} = -\frac{1}{n} \sum \frac{\partial}{\partial w_j} [y \ln a + (1 - y) \ln(1 - a)] \quad (238)$$

$$= -\frac{1}{n} \sum \frac{\partial}{\partial a} [y \ln a + (1 - y) \ln(1 - a)] * \frac{\partial a}{\partial w_j} \quad (239)$$

$$= -\frac{1}{n} \sum \left( \frac{y}{a} - \frac{1 - y}{1 - a} \right) * \frac{\partial a}{\partial w_j} \quad (240)$$

$$= -\frac{1}{n} \sum \left( \frac{y}{\sigma(z)} - \frac{1 - y}{1 - \sigma(z)} \right) \frac{\partial \sigma(z)}{\partial w_j} \quad (241)$$

$$= -\frac{1}{n} \sum \left( \frac{y}{\sigma(z)} - \frac{1 - y}{1 - \sigma(z)} \right) \sigma'(z) x_j \quad (242)$$

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

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

这是一个优美的公式。它告诉我们权重学习的速度受到  $\sigma(z) - y$ , 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  $\sigma'(z)$  导致的学习缓慢。当我们使用交叉熵的时候,  $\sigma'(z)$  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:

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

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

### 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。

## 3.5 Batch\_Size

### 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。

### 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  $\text{Batch\_Size}$  为全数据集或者  $\text{Batch\_Size} = 1$  都有各自缺点，可不可以选择一个适中的  $\text{Batch\_Size}$  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。

### 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  $\text{Batch\_Size}$  越大，其确定的下降方向越准，引起训练震荡越小。

### 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.  $\text{Batch\_Size}$  增大到一定程度，其确定的下降方向已经基本不再变化。

### 3.5.5 调节 Batch\_Size 对训练效果影响到底如何？

1.  $\text{Batch\_Size}$  太小，模型表现效果极其糟糕(error飙升)。
2. 随着  $\text{Batch\_Size}$  增大，处理相同数据量的速度越快。
3. 随着  $\text{Batch\_Size}$  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， $\text{Batch\_Size}$  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。

## 3.6 归一化

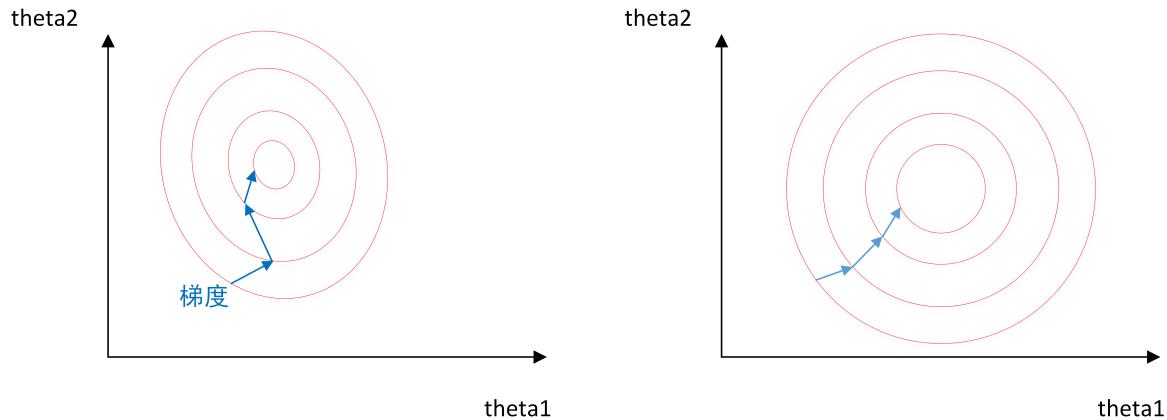
### 3.6.1 归一化含义？

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。

### 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。

### 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。

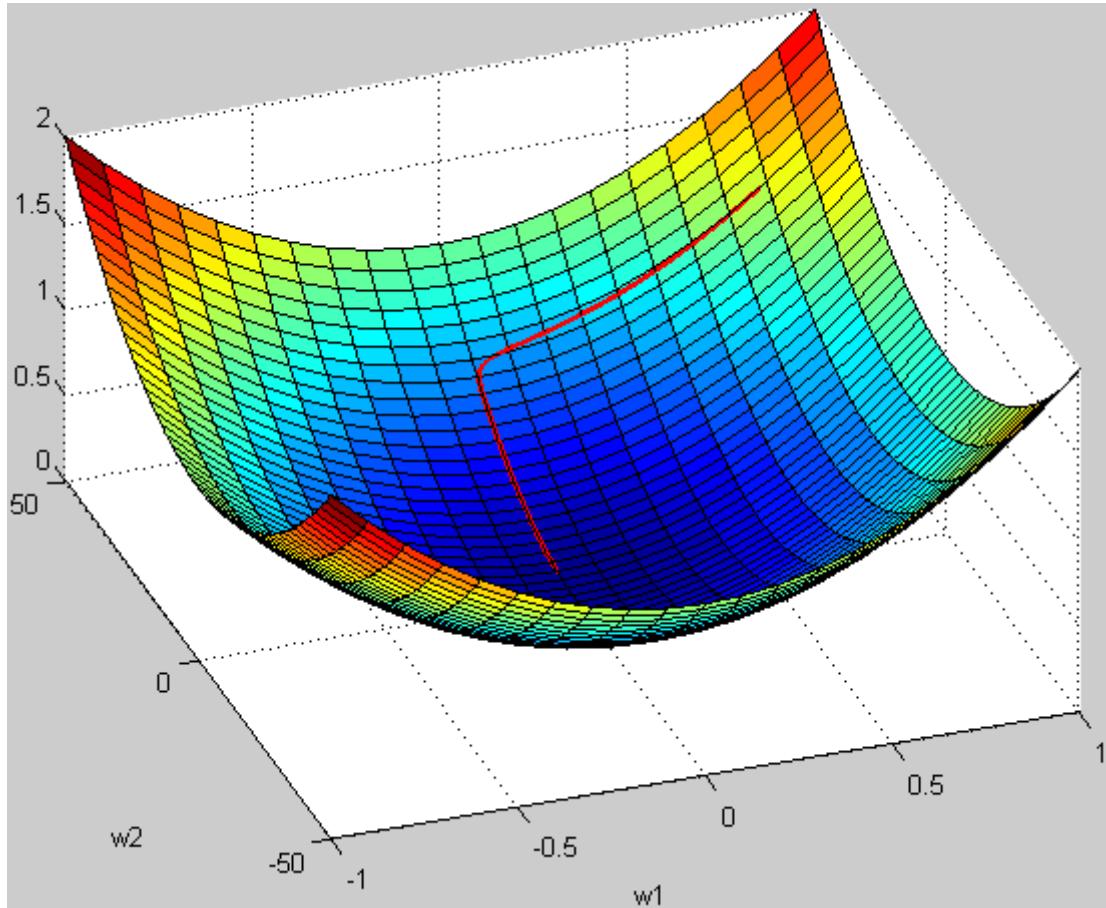
因此如果机器学习模型使用梯度下降法求最优解时，归一化往往非常有必要，否则很难收敛甚至不能收敛。

### 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。



### 3.6.5 归一化有哪些类型?

#### 1. 线性归一化

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \quad (177)$$

适用范围：比较适用于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

$$x' = \frac{x - \mu}{\sigma} \quad (178)$$

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。

### 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。

### 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：

$$b_{x,y}^i = a_{x,y}^i / \left( k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a_{x,y}^j)^2 \right)^\beta \quad (179)$$

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。

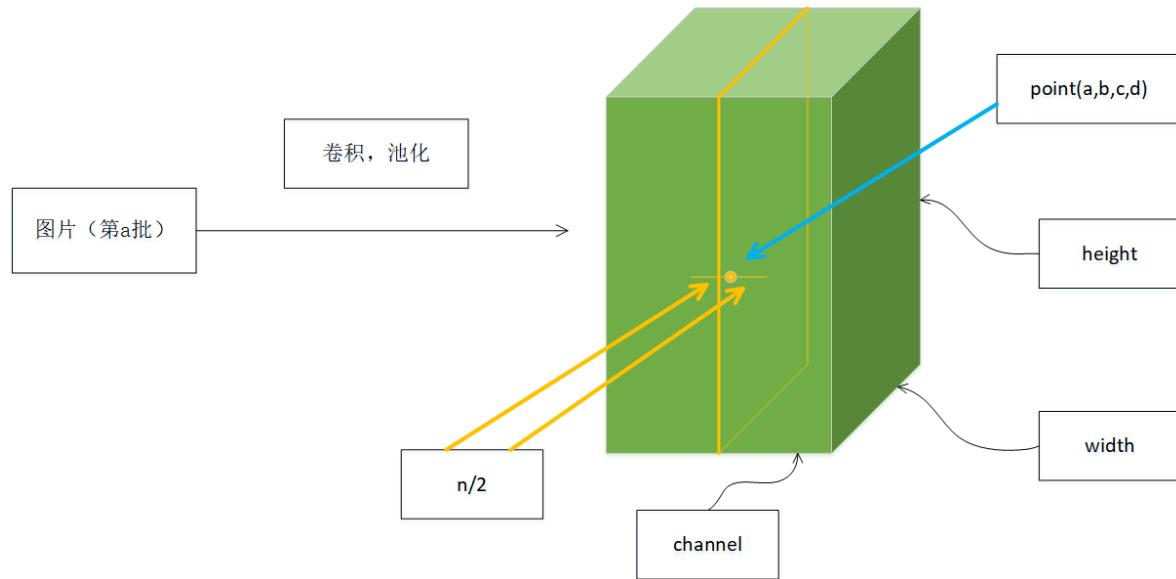
2)  $a_{x,y}^i$  表示在这个输出结构中的一个位置  $[a, b, c, d]$ ，可以理解成在某一张图中的某一个通道下的某个高度和某个宽度位置的点，即第  $a$  张图的第  $d$  个通道下的高度为  $b$  宽度为  $c$  的点。

3)  $N$ ：论文公式中的  $N$  表示通道数 (channel)。

4)  $a, n/2, k$  分别表示函数中的 input, depth\_radius, bias。参数  $k, n, \alpha, \beta$  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：



### 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)。

### 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。

### 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

$$\mu_\beta = \frac{1}{m} \sum_{i=1}^m (x_i) \quad (180)$$

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

$$\sigma_\beta^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_\beta)^2 \quad (181)$$

3. 归一化处理，得到

$$\hat{x}_i = \frac{x_i + \mu_\beta}{\sqrt{\sigma_\beta^2} + \epsilon} \quad (182)$$

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，对经过上面归一化处理得到的数据进行重构，得到

$$y_i = \gamma \hat{x}_i + \beta \quad (183)$$

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  $\sigma_\beta^2$  的无偏估计得到。

### 3.6.11 批归一化和群组归一化比较

名称	特点
批量归一化 (Batch Normalization, 以下简称 BN)	可让各种网络并行训练。但是，批量维度进行归一化会带来一些问题——批量统计估算不准确导致批量变小时，BN 的误差会迅速增加。在训练大型网络和将特征转移到计算机视觉任务中（包括检测、分割和视频），内存消耗限制了只能使用小批量的 BN。
群组归一化 Group Normalization (简称 GN)	GN 将通道分成组，并在每组内计算归一化的均值和方差。GN 的计算与批量大小无关，并且其准确度在各种批量大小下都很稳定。
比较	在 ImageNet 上训练的 ResNet-50 上，GN 使用批量大小为 2 时的错误率比 BN 的错误率低 10.6%；当使用典型的批量时，GN 与 BN 相当，并且优于其他标归一化变体。而且，GN 可以自然地从预训练迁移到微调。在进行 COCO 中的目标检测和分割以及 Kinetics 中的视频分类比赛中，GN 可以胜过其竞争对手，表明 GN 可以在各种任务中有效地取代强大的 BN。

### 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。

### 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 应作用在非线性映射前。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。

## 3.7 预训练与微调(fine tuning)

### 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。

### 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。

### 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。

### 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。

## 3.8 权重偏差初始化

### 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。

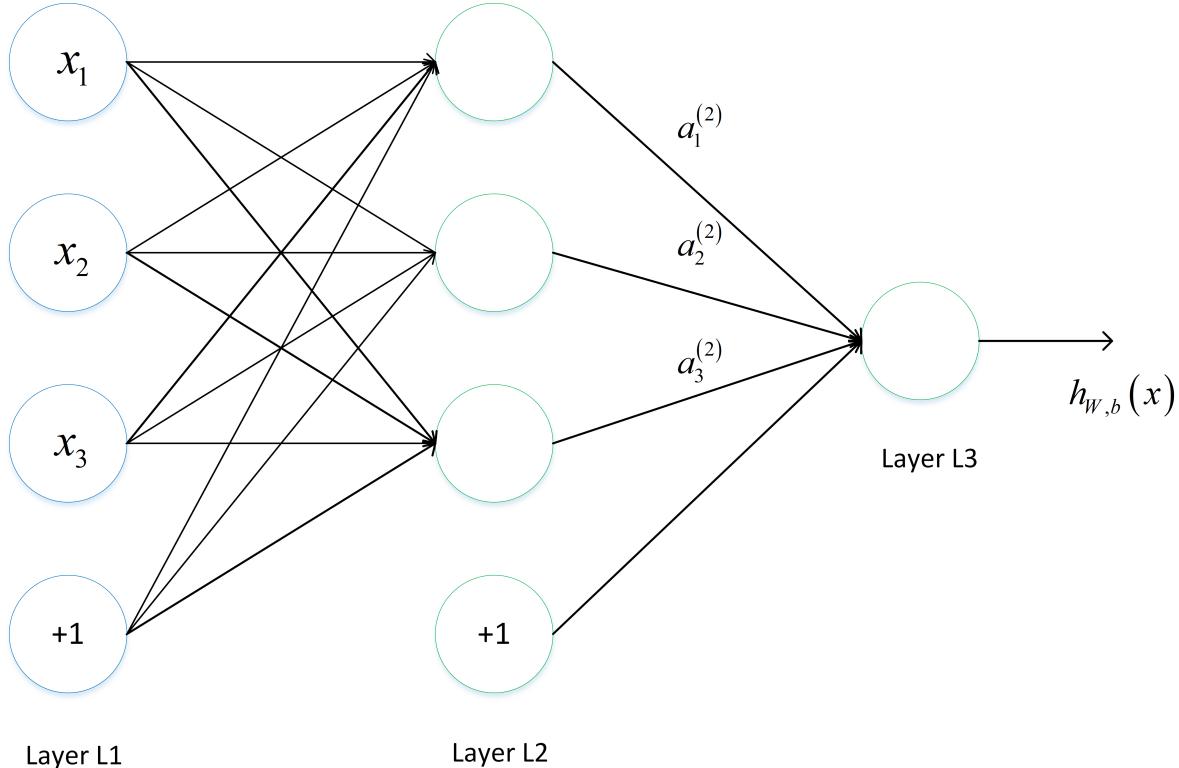
形象化理解：在神经网络中考虑梯度下降的时候，设想你在爬山，但身处直线形的山谷中，两边是对称的山峰。由于对称性，你所在之处的梯度只能沿着山谷的方向，不会指向山峰；你走了一步之后，情况依然不变。结果就是你只能收敛到山谷中的一个极大值，而走不到山峰上去。

### 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：

$$a_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}) \quad (184)$$

$$a_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}) \quad (185)$$

$$a_3^{(2)} = f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)}) \quad (186)$$

$$h_{W,b}(x) = a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)}) \quad (187)$$

$$xa_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)})a_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}) \quad (188)$$

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  $a_1 = a_2 = a_3 = \dots$ ，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

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) = a_j^{(l)} \delta_i^{(l+1)} \frac{\partial}{\partial b_i^{(l)}} J(W, b; x, y) = \delta_i^{(l+1)} \quad (189)$$

$\delta$  的计算公式

$$\delta_i^{(l)} = \left( \sum_{j=1}^{s_{t+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)}) \quad (190)$$

如果用的是 sigmoid 函数

$$f'(z_i^{(l)}) = a_i^{(l)} (1 - a_i^{(l)}) \quad (191)$$

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）。

### 3.8.3 初始话为小的随机数

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，神经元在一开始都是随机的、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  $W = 0.01 * np.random.randn(D, H)$ ，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。

### 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。

### 3.8.5 稀疏初始化(Sparse Initialazation)

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。

### 3.8.6 初始话偏差

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.

## 3.9 学习率

### 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。

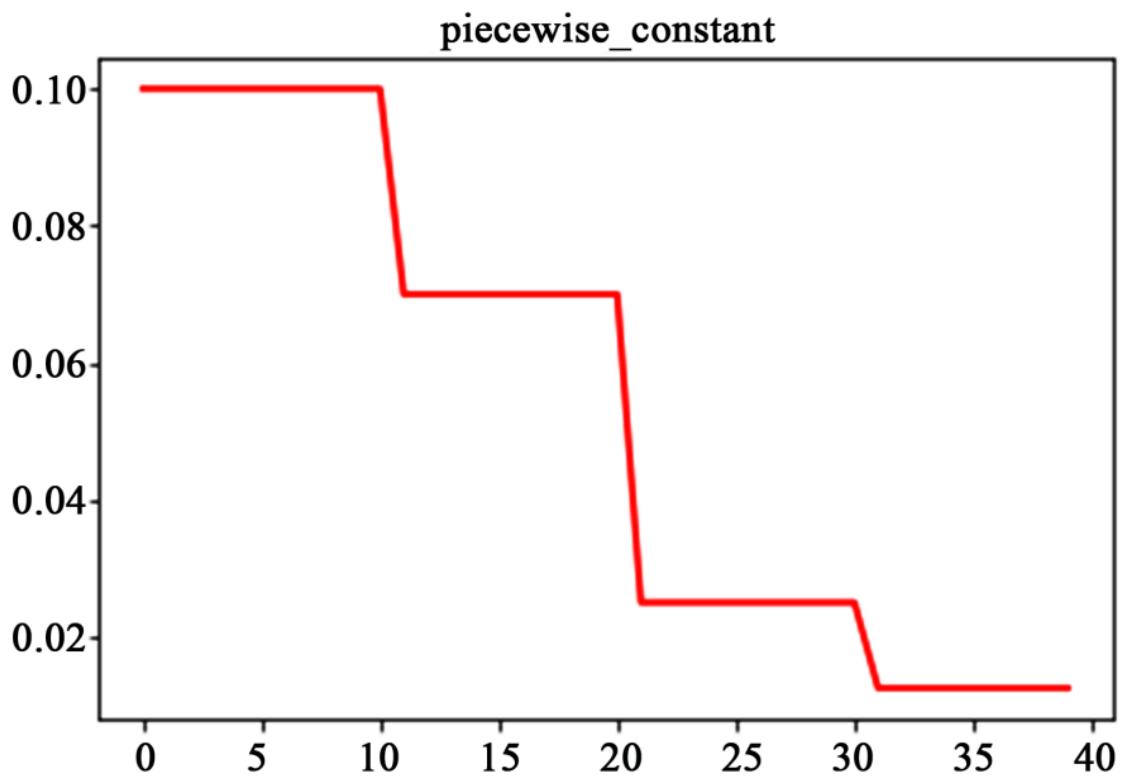
在模型优化中，常用到的几种学习率衰减方法有：分段常数衰减、多项式衰减、指数衰减、自然指数衰减、余弦衰减、线性余弦衰减、噪声线性余弦衰减

### 3.9.2 学习率衰减常用参数有哪些

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
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

### 3.9.3 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。

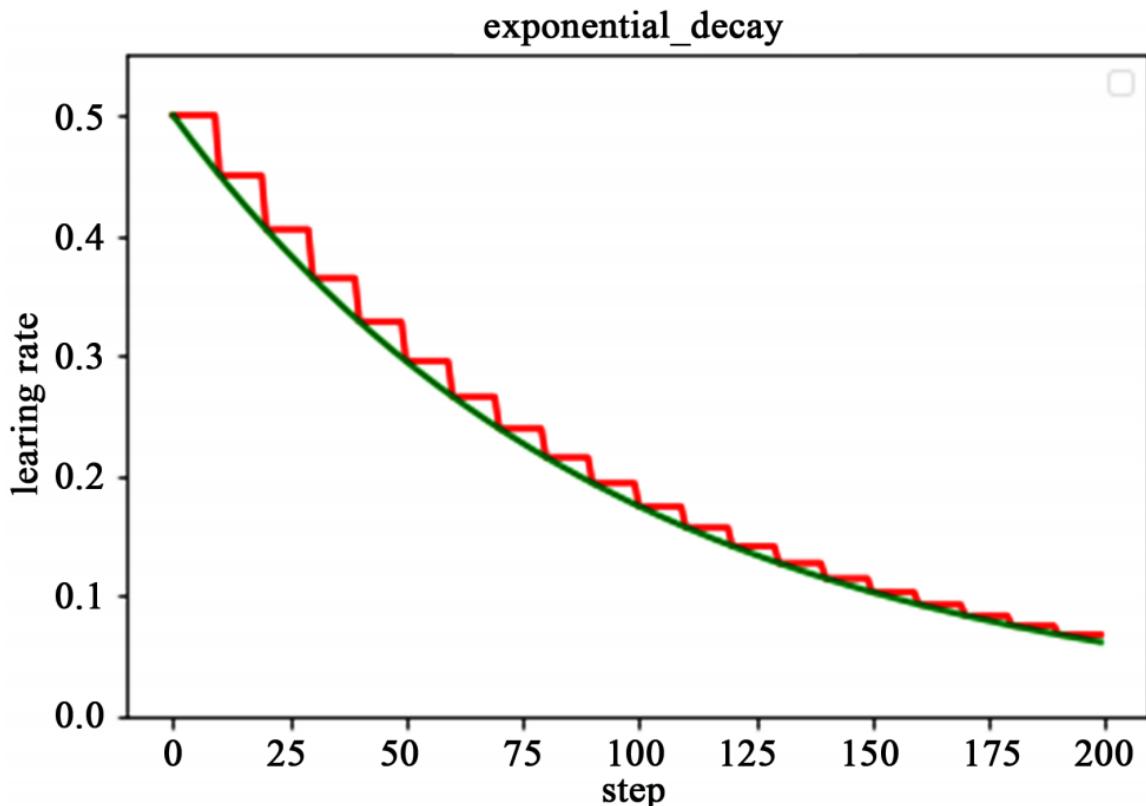


### 3.9.4 指数衰减

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：

$$\text{decayed\_learning\_rate} = \text{learning\_rate} * \text{decay\_rate}^{\frac{\text{global\_step}}{\text{decay\_steps}}} \quad (192)$$

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。

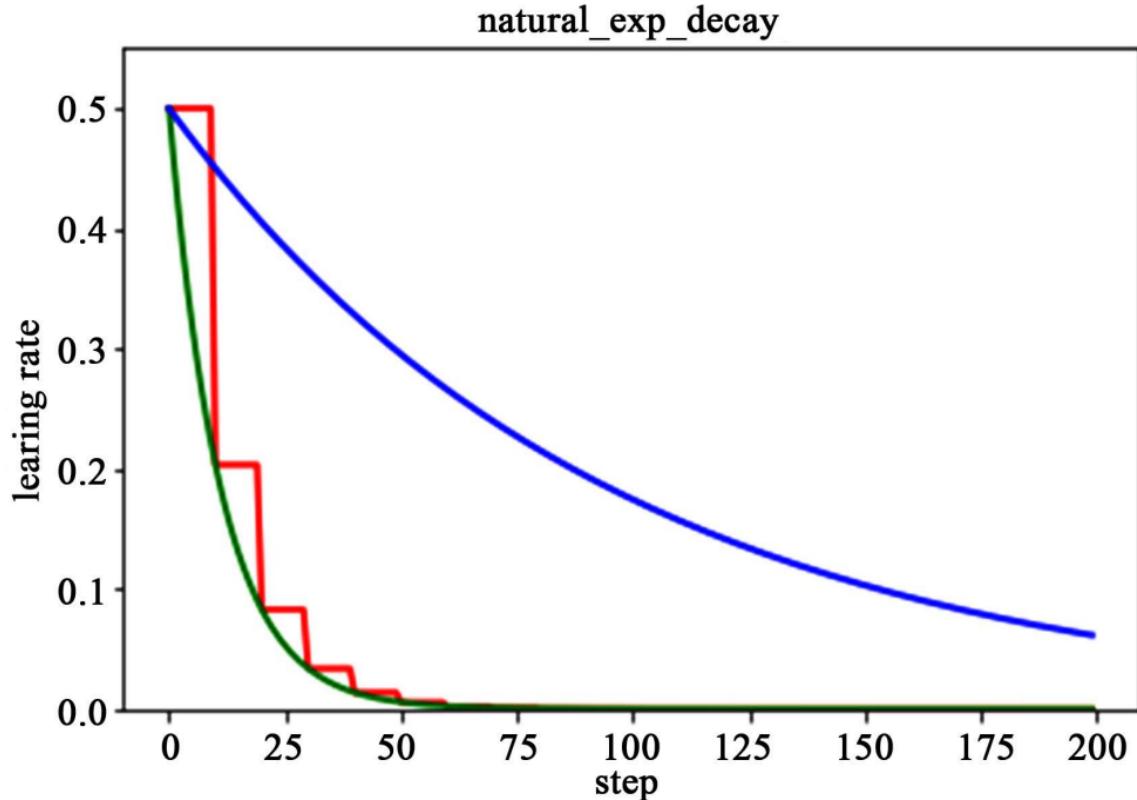


### 3.9.5 自然指数衰减

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

$$\text{decayed\_learning\_rate} = \text{learning\_rate} * e^{\frac{-\text{decay\_rate}}{\text{global\_step}}} \quad (193)$$

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。



### 3.9.6 多项式衰减

应用多项式衰减的方式进行更新学习率，这里会给定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。

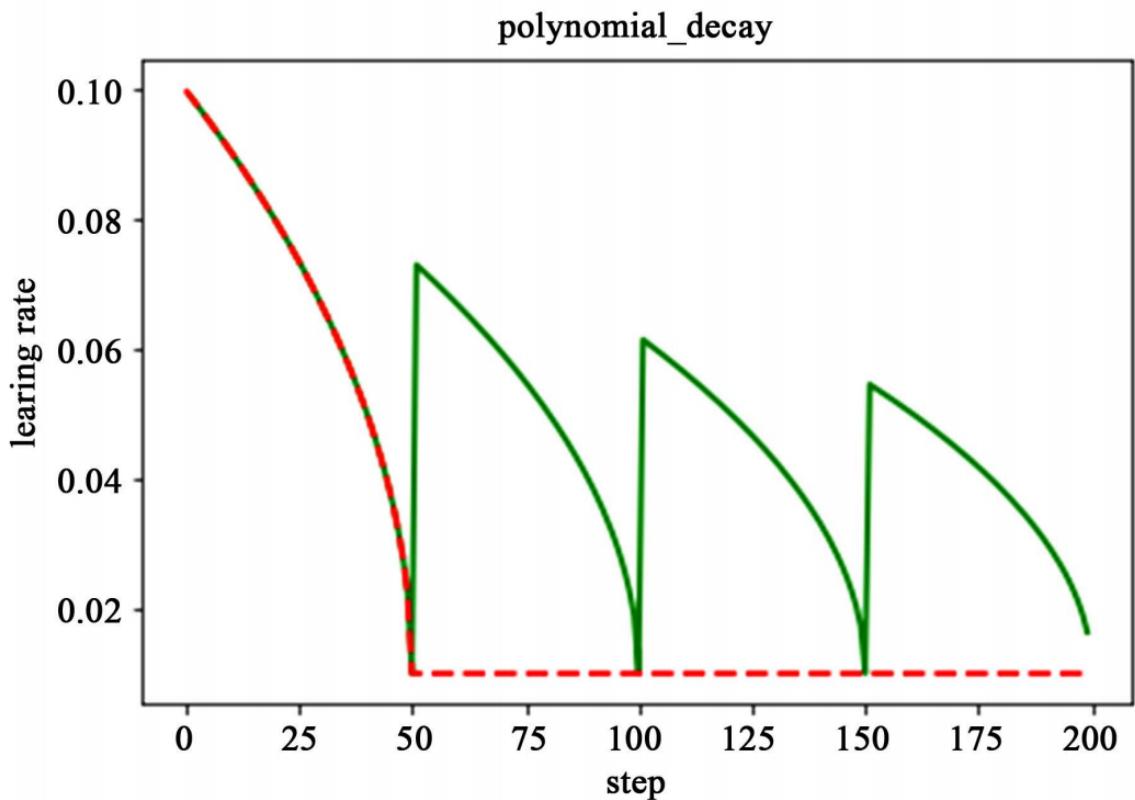
$$\text{global\_step} = \min(\text{global\_step}, \text{decay\_steps}) \quad (194)$$

$$\begin{aligned} \text{decayed\_learning\_rate} &= (\text{learning\_rate} - \text{end\_learning\_rate}) * \left(1 - \frac{\text{global\_step}}{\text{decay\_steps}}\right)^{\text{power}} \\ &+ \text{end\_learning\_rate} \end{aligned} \quad (195)$$

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。

$$\text{decay\_steps} = \text{decay\_steps} * \text{ceil}\left(\frac{\text{global\_step}}{\text{decay\_steps}}\right) \quad (196)$$

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。



### 3.9.7 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：

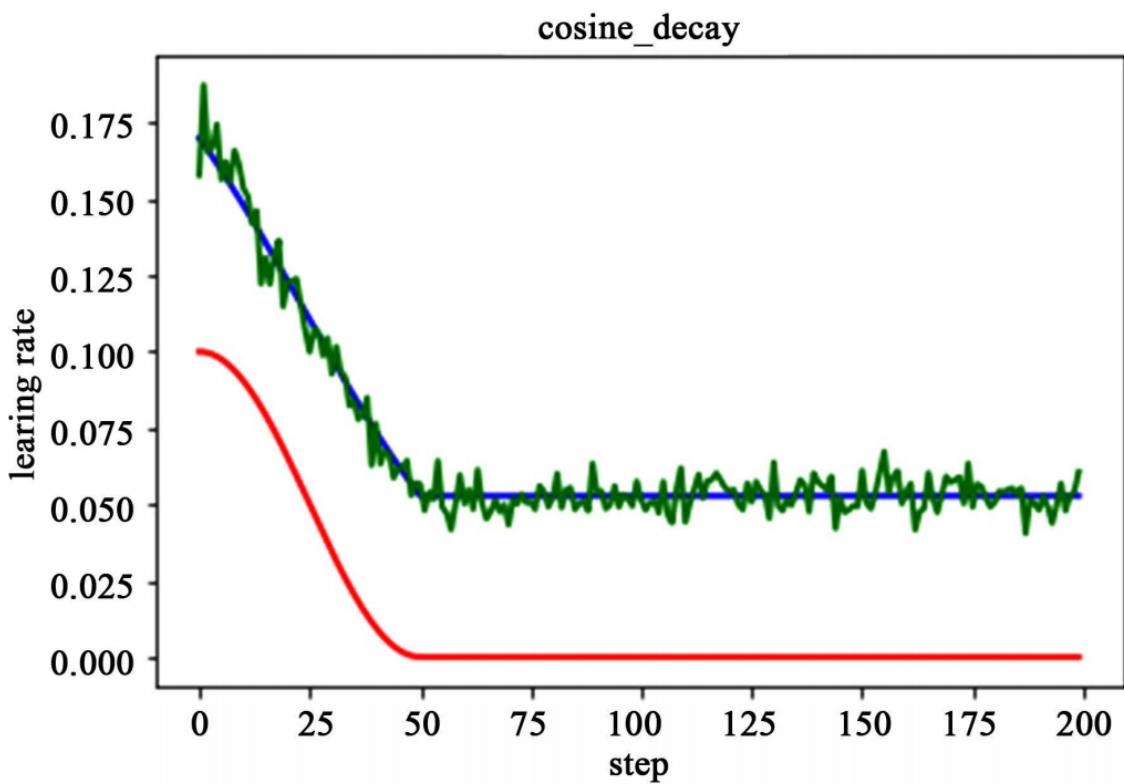
$$global\_step = \min(global\_step, decay\_steps) \quad (197)$$

$$cosine\_decay = 0.5 * \left( 1 + \cos \left( \pi * \frac{global\_step}{decay\_steps} \right) \right) \quad (198)$$

$$decayed = (1 - \alpha) * cosine\_decay + \alpha \quad (199)$$

$$decayed\_learning\_rate = learning\_rate * decayed \quad (200)$$

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。

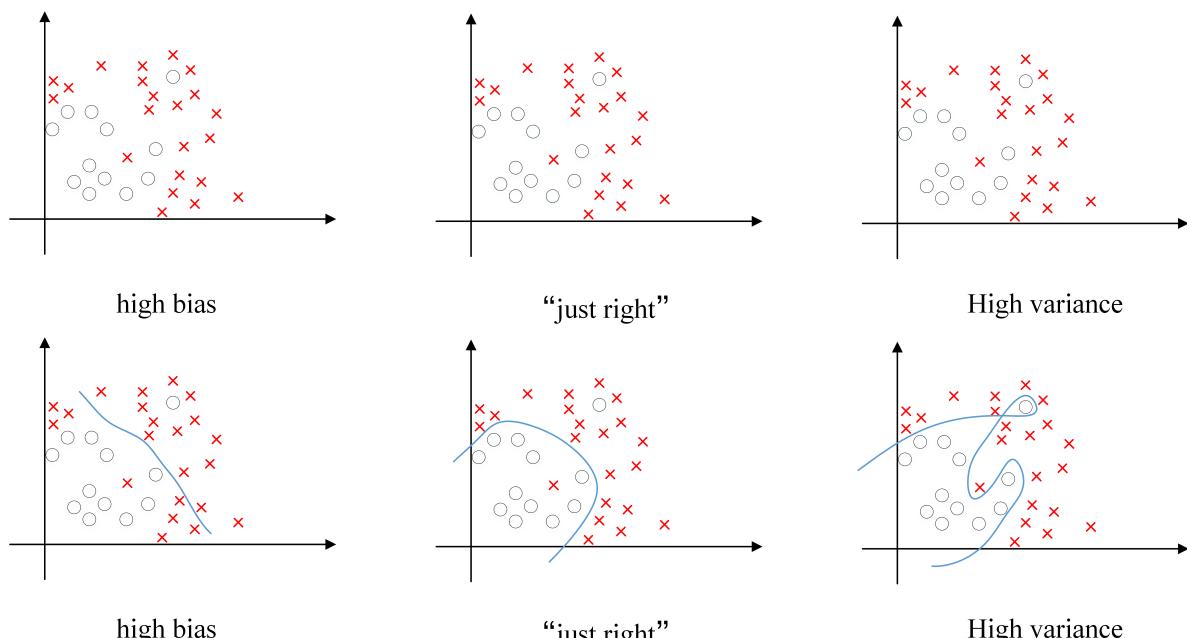


## 3.12 Dropout 系列问题

### 3.12.1 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。
- 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。

### 3.12.2 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。

### 3.12.3 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的结果实它会压缩权重，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。

### 3.12.4 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  $1/p$ 。

### 3.12.5 dropout有什么缺点？

dropout一大缺点就是代价函数不再被明确定义，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每次迭代后都会下降，因为我们所优化的代价函数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。

## 3.13 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；

## 3.14 如何理解 Internal Covariate Shift？

---

(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？其中最重要的原因是，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。

## 参考文献

- [1] Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain.[J]. Psychological Review, 1958, 65(6):386-408.
- [2] Duvenaud D , Rippel O , Adams R P , et al. Avoiding pathologies in very deep networks[J]. Eprint Arxiv, 2014:202-210.
- [3] Rumelhart D E, Hinton G E, Williams R J. Learning representations by back-propagating errors[J]. Cognitive modeling, 1988, 5(3): 1.
- [4] Hecht-Nielsen R. Theory of the backpropagation neural network[M]//Neural networks for perception. Academic Press, 1992: 65-93.
- [5] Felice M. Which deep learning network is best for you?| CIO[J]. 2017.
- [6] Conneau A, Schwenk H, Barrault L, et al. Very deep convolutional networks for natural language processing[J]. arXiv preprint arXiv:1606.01781, 2016, 2.
- [7] Ba J, Caruana R. Do deep nets really need to be deep?[C]//Advances in neural information processing systems. 2014: 2654-2662.
- [8] Nielsen M A. Neural networks and deep learning[M]. USA: Determination press, 2015.
- [9] Goodfellow I, Bengio Y, Courville A. Deep learning[M]. MIT press, 2016.
- [10] 周志华. 机器学习[M]. 清华大学出版社, 2016.

- [11] Kim J, Kwon Lee J, Mu Lee K. Accurate image super-resolution using very deep convolutional networks[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 1646-1654.
- [12] Chen Y, Lin Z, Zhao X, et al. Deep learning-based classification of hyperspectral data[J]. IEEE Journal of Selected topics in applied earth observations and remote sensing, 2014, 7(6): 2094-2107.
- [13] Domhan T, Springenberg J T, Hutter F. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves[C]//Twenty-Fourth International Joint Conference on Artificial Intelligence. 2015.
- [14] Maclaurin D, Duvenaud D, Adams R. Gradient-based hyperparameter optimization through reversible learning[C]//International Conference on Machine Learning. 2015: 2113-2122.
- [15] Srivastava R K, Greff K, Schmidhuber J. Training very deep networks[C]//Advances in neural information processing systems. 2015: 2377-2385.
- [16] Bergstra J, Bengio Y. Random search for hyper-parameter optimization[J]. Journal of Machine Learning Research, 2012, 13(Feb): 281-305.
- [17] Ngiam J, Khosla A, Kim M, et al. Multimodal deep learning[C]//Proceedings of the 28th international conference on machine learning (ICML-11). 2011: 689-696.
- [18] Deng L, Yu D. Deep learning: methods and applications[J]. Foundations and Trends® in Signal Processing, 2014, 7(3-4): 197-387.
- [19] Erhan D, Bengio Y, Courville A, et al. Why does unsupervised pre-training help deep learning? [J]. Journal of Machine Learning Research, 2010, 11(Feb): 625-660.
- [20] Dong C, Loy C C, He K, et al. Learning a deep convolutional network for image super resolution[C]//European conference on computer vision. Springer, Cham, 2014: 184-199.
- [21] 郑泽宇, 梁博文, 顾思宇. TensorFlow: 实战Google深度学习框架 (第2版) [M].电子工业出版社,2018.
- [22] 焦李成. 深度学习优化与识别[M].清华大学出版社,2017.
- [23] 吴岸城. 神经网络与深度学习[M].电子工业出版社,2016.
- [24] Wei, W.G.H., Liu, T., Song, A., et al. (2018) An Adaptive Natural Gradient Method with Adaptive Step Size in Multi-layer Perceptrons. Chinese Automation Congress, 1593-1597.
- [25] Y Feng, Y Li. An Overview of Deep Learning Optimization Methods and Learning Rate Attenuation Methods[J]. Hans Journal of Data Mining, 2018, 8(4), 186-200.

# 第四章 经典网络解读

## 4.1 LeNet-5

### 4.1.1 模型介绍

LeNet-5是由*LeCun*提出的一种用于识别手写数字和机器印刷字符的卷积神经网络（Convolutional Neural Network, CNN）<sup>[1]</sup>，其命名来源于作者*LeCun*的名字，5则是其研究成果的代号，在LeNet-5之前还有LeNet-4和LeNet-1鲜为人知。LeNet-5阐述了图像中像素特征之间的相关性能够由参数共享的卷积操作所提取，同时使用卷积、下采样（池化）和非线性映射这样的组合结构，是当前流行的大多数深度图像识别网络的基础。

### 4.1.2 模型结构

图4.1 LeNet-5网络结构图

如图4.1所示，LeNet-5一共包含7层（输入层不作为网络结构），分别由2个卷积层、2个下采样层和3个连接层组成，网络的参数配置如表4.1所示，其中下采样层和全连接层的核尺寸分别代表采样范围和连接矩阵的尺寸（如卷积核尺寸中的“ $5 \times 5 \times 1/1, 6$ ”表示核大小为 $5 \times 5 \times 1$ 、步长为1且核个数为6的卷积核）。

表4.1 LeNet-5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 $C_1$	$32 \times 32 \times 1$	$5 \times 5 \times 1/1, 6$	$28 \times 28 \times 6$	$(5 \times 5 \times 1 + 1) \times 6$
下采样层 $S_2$	$28 \times 28 \times 6$	$2 \times 2/2$	$14 \times 14 \times 6$	$(1 + 1) \times 6^*$
卷积层 $C_3$	$14 \times 14 \times 6$	$5 \times 5 \times 6/1, 16$	$10 \times 10 \times 16$	$1516^*$
下采样层 $S_4$	$10 \times 10 \times 16$	$2 \times 2/2$	$5 \times 5 \times 16$	$(1 + 1) \times 16$
卷积层 $C_5^*$	$5 \times 5 \times 16$	$5 \times 5 \times 16/1, 120$	$1 \times 1 \times 120$	$(5 \times 5 \times 16 + 1) \times 120$
全连接层 $F_6$	$1 \times 1 \times 120$	$120 \times 84$	$1 \times 1 \times 84$	$(120 + 1) \times 84$
输出层	$1 \times 1 \times 84$	$84 \times 10$	$1 \times 1 \times 10$	$(84 + 1) \times 10$

\* 在LeNet中，下采样操作和池化操作类似，但是在得到采样结果后会乘以一个系数和加上一个偏置项，所以下采样的参数个数是 $(1 + 1) \times 6$ 而不是零。

\*  $C_3$  卷积层可训练参数并未直接连接  $S_2$  中所有的特征图 (Feature Map) , 而是采用如图4.2所示的采样特征方式进行连接 (稀疏连接) , 生成的16个通道特征图中分别按照相邻3个特征图、相邻4个特征图、非相邻4个特征图和全部6个特征图进行映射, 得到的参数个数计算公式为  $6 \times (25 \times 3 + 1) + 6 \times (25 \times 4 + 1) + 3 \times (25 \times 4 + 1) + 1 \times (25 \times 6 + 1) = 1516$ , 在原论文中解释了使用这种采样方式原因包含两点: 限制了连接数不至于过大 (当年的计算能力比较弱); 强制限定不同特征图的组合可以使映射得到的特征图学习到不同的特征模式。

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X			X X	X		X		X X X	X	X	X	X	X	X	
1	X X			X	X X				X X	X X	X X	X X	X X	X X	X X	
2	X X X				X X X		X X X			X	X X X	X X	X X	X X	X X	
3		X X X			X X X X		X X X X			X	X X	X X	X X	X X	X X	
4		X X X			X X X X		X X X X	X X X X		X X	X X	X X	X X	X X	X X	
5		X X X			X X X X		X X X X X	X X X X X		X X	X X	X X	X X	X X	X X	

图4.2  $S_2$  与  $C_3$  之间的特征图稀疏连接

\*  $C_5$  卷积层在图4.1中显示为全连接层, 原论文中解释这里实际采用的是卷积操作, 只是刚好在  $5 \times 5$  卷积后尺寸被压缩为  $1 \times 1$ , 输出结果看起来和全连接很相似。

### 4.1.3 模型特性

- 卷积网络使用一个3层的序列组合: 卷积、下采样 (池化) 、非线性映射 (LeNet-5最重要的特性, 奠定了目前深层卷积网络的基础)
- 使用卷积提取空间特征
- 使用映射的空间均值进行下采样
- 使用  $tanh$  或  $sigmoid$  进行非线性映射
- 多层神经网络 (MLP) 作为最终的分类器
- 层间的稀疏连接矩阵以避免巨大的计算开销

## 4.2 AlexNet

### 4.2.1 模型介绍

AlexNet是由Alex Krizhevsky提出的首个应用于图像分类的深层卷积神经网络, 该网络在2012年ILSVRC (ImageNet Large Scale Visual Recognition Competition) 图像分类竞赛中以15.3%的top-5测试错误率赢得第一名<sup>[2]</sup>。AlexNet使用GPU代替CPU进行运算, 使得在可接受的时间范围内模型结构能够更加复杂, 它的出现证明了深层卷积神经网络在复杂模型下的有效性, 使CNN在计算机视觉中流行开来, 直接或间接地引发了深度学习的热潮。

### 4.2.2 模型结构

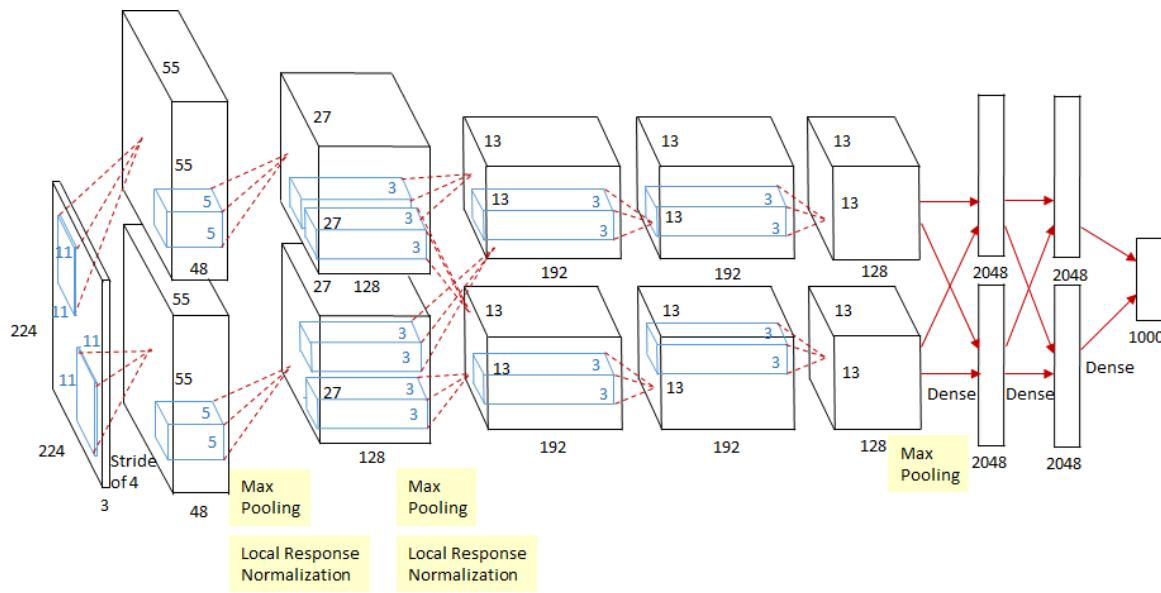


图4.3 AlexNet网络结构图

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 Response 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
卷积层 $C_1^*$	$224 \times 224 \times 3$	$11 \times 11 \times 3/4, 48(\times 2_{GPU})$	$55 \times 55 \times 48(\times 2_{GPU})$
下采样层 $S_{max}^*$	$55 \times 55 \times 48(\times 2_{GPU})$	$3 \times 3/2(\times 2_{GPU})$	$27 \times 27 \times 48(\times 2_{GPU})$
卷积层 $C_2$	$27 \times 27 \times 48(\times 2_{GPU})$	$5 \times 5 \times 48/1, 128(\times 2_{GPU})$	$27 \times 27 \times 128(\times 2_{GPU})$
下采样层 $S_{max}$	$27 \times 27 \times 128(\times 2_{GPU})$	$3 \times 3/2(\times 2_{GPU})$	$13 \times 13 \times 128(\times 2_{GPU})$
卷积层 $C_3^*$	$13 \times 13 \times 128 \times 2_{GPU}$	$3 \times 3 \times 256/1, 192(\times 2_{GPU})$	$13 \times 13 \times 192(\times 2_{GPU})$
卷积层 $C_4$	$13 \times 13 \times 192(\times 2_{GPU})$	$3 \times 3 \times 192/1, 192(\times 2_{GPU})$	$13 \times 13 \times 192(\times 2_{GPU})$
卷积层 $C_5$	$13 \times 13 \times 192(\times 2_{GPU})$	$3 \times 3 \times 192/1, 128(\times 2_{GPU})$	$13 \times 13 \times 128(\times 2_{GPU})$
下采样层 $S_{max}$	$13 \times 13 \times 128(\times 2_{GPU})$	$3 \times 3/2(\times 2_{GPU})$	$6 \times 6 \times 128(\times 2_{GPU})$
全连接层 $F_6^*$	$6 \times 6 \times 128 \times 2_{GPU}$	$9216 \times 2048(\times 2_{GPU})$	$1 \times 1 \times 2048(\times 2_{GPU})$
全连接层 $F_7$	$1 \times 1 \times 2048 \times 2_{GPU}$	$4096 \times 2048(\times 2_{GPU})$	$1 \times 1 \times 2048(\times 2_{GPU})$
全连接层 $F_8$	$1 \times 1 \times 2048 \times 2_{GPU}$	$4096 \times 1000$	$1 \times 1 \times 1000$

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。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。

## 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

## 4.3 ZFNet

### 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Zeiler，而且Clarifai对ZFNet的改动比较小，所以通常认为是ZFNet获得了冠军）<sup>[3-4]</sup>。ZFNet实际上是微调（fine-tuning）了的AlexNet，并通过反卷积（Deconvolution）的方式可视化各层的输出特征图，进一步解释了卷积操作在大型网络中效果显著的原因。

### 4.3.2 模型结构

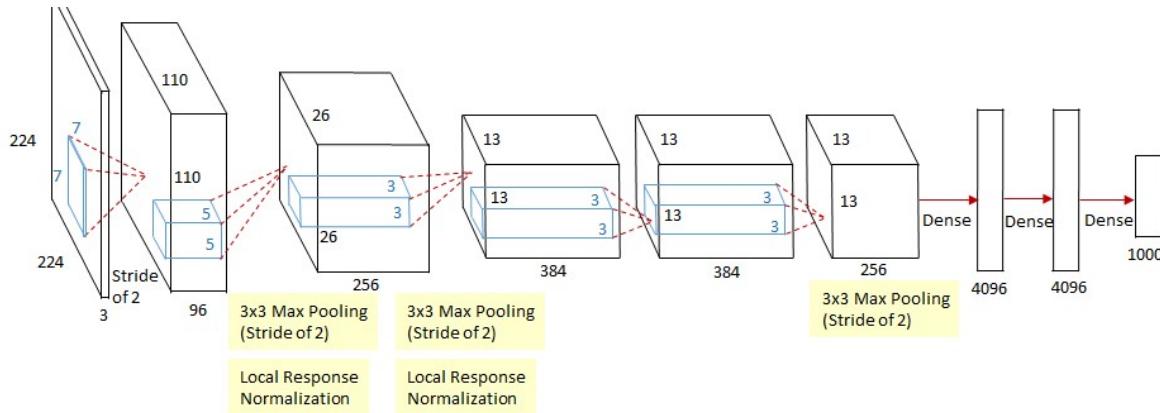
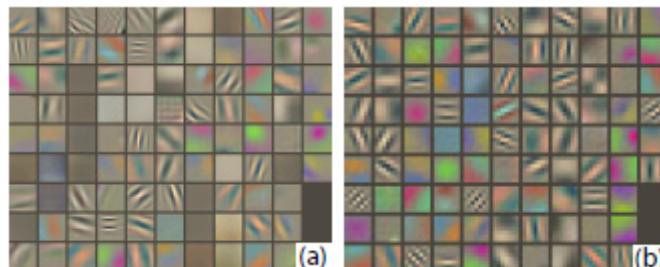


图4.4 ZFNet网络结构图（原始结构图与AlexNet风格结构图）

如图4.4所示，ZFNet与AlexNet类似，都是由8层网络组成的卷积神经网络，其中包含5层卷积层和3层全连接层。两个网络结构最大的不同在于，ZFNet第一层卷积采用了 $7 \times 7 \times 3/2$ 的卷积核替代了AlexNet中第一层卷积核 $11 \times 11 \times 3/4$ 的卷积核。图4.5中ZFNet相比于AlexNet在第一层输出的特征图中包含更多中间频率的信息，而AlexNet第一层输出的特征图大多是低频或高频的信息，对中间频率特征的缺失导致后续网络层次如图4.5 (c) 能够学习到的特征不够细致，而导致这个问题的根本原因在于AlexNet在第一层中采用的卷积核和步长过大。



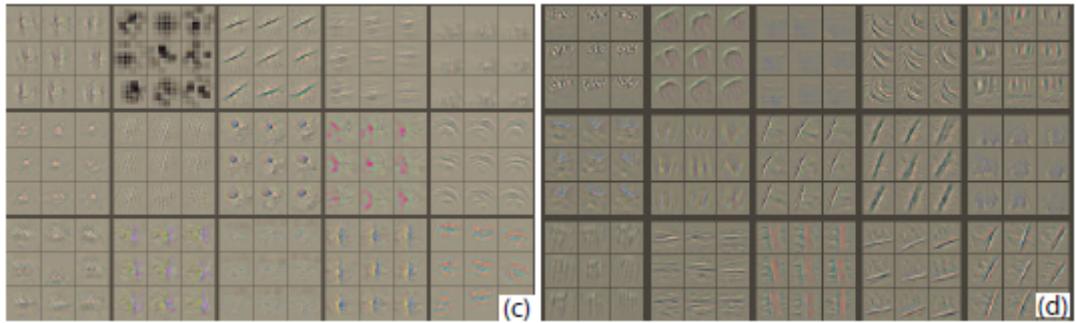


图4.5 (a) ZFNet第一层输出的特征图 (b) AlexNet第一层输出的特征图 (c) AlexNet第二层输出的特征图 (d) ZFNet第二层输出的特征图

表4.3 ZFNet网络参数配置

Table 4.3 ZFNet Network Parameter Configuration

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。

### 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 Network in Network

### 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。

### 4.4.2 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
局部全连接层 $L_{11}^*$	$32 \times 32 \times 3$	$(3 \times 3) \times 16/1$	$30 \times 30 \times 16$	$(3 \times 3 \times 3 + 1) \times 16$
全连接层 $L_{12}^*$	$30 \times 30 \times 16$	$16 \times 16$	$30 \times 30 \times 16$	$((16 + 1) \times 16)$
局部全连接层 $L_{21}$	$30 \times 30 \times 16$	$(3 \times 3) \times 64/1$	$28 \times 28 \times 64$	$(3 \times 3 \times 16 + 1) \times 64$
全连接层 $L_{22}$	$28 \times 28 \times 64$	$64 \times 64$	$28 \times 28 \times 64$	$((64 + 1) \times 64)$
局部				

全连接层 $L_{31}$	$28 \times 28 \times 64$ 输入尺寸	$(3 \times 3) \times 100/1$ 核尺寸	$26 \times 26 \times 100$ 输出尺寸	$(3 \times 3 \times 64 + 1) \times 100$ 参数个数
全连接层 $L_{32}$	$26 \times 26 \times 100$	$100 \times 100$	$26 \times 26 \times 100$	$((100 + 1) \times 100)$
全局平均采样 $GAP^*$	$26 \times 26 \times 100$	$26 \times 26 \times 100/1$	$1 \times 1 \times 100$	0

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 $30 \times 30$  ( $\frac{32-3k+1}{1 stride} = 30$ )

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。

### 4.4.3 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。
- 使用全局平均池化来代替最后一个全连接层，能够有效地减少参数量（没有可训练参数），同时池化用到了整个特征图的信息，对空间信息的转换更加鲁棒，最后得到的输出结果可直接作为对应类别的置信度。

## 4.5 VGGNet

### 4.5.1 模型介绍

VGGNet是由牛津大学视觉几何小组 (Visual Geometry Group, VGG) 提出的一种深层卷积网络结构，他们以7.32%的错误率赢得了2014年ILSVRC分类任务的亚军（冠军由GoogLeNet以6.65%的错误率夺得）和25.32%的错误率夺得定位任务 (Localization) 的第一名 (GoogLeNet错误率为26.44%)<sup>[5]</sup>，网络名称VGGNet取自该小组名缩写。VGGNet是首批把图像分类的错误率降低到10%以内模型，同时该网络所采用的 $3 \times 3$ 卷积核的思想是后来许多模型的基础，该模型发表在2015年国际学习表征会议 (International Conference On Learning Representations, ICLR) 后至今被引用的次数已经超过1万4千余次。

### 4.5.2 模型结构

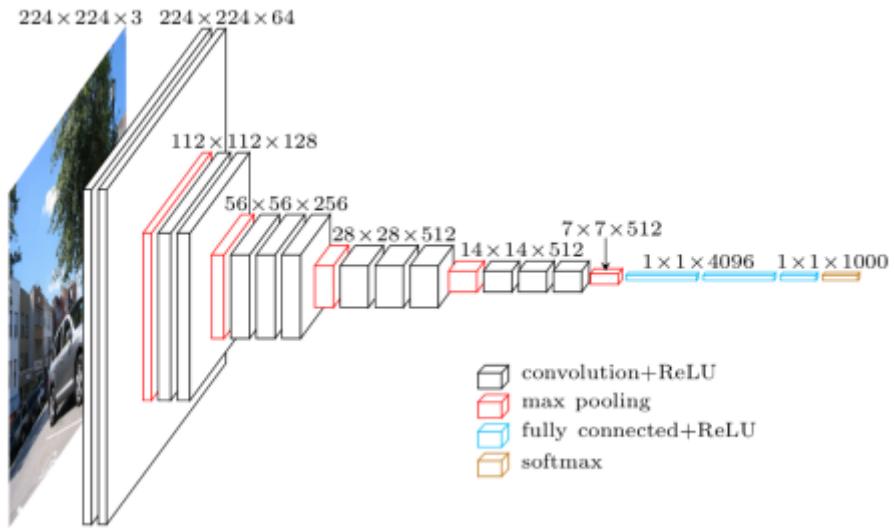


图 4.7 VGG16网络结构图

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 $3 \times 3$ 卷积即可与 $5 \times 5$ 卷积拥有相同感受视野），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 $C_{11}$	$224 \times 224 \times 3$	$3 \times 3 \times 64/1$	$224 \times 224 \times 64$	$(3 \times 3 \times 3 + 1) \times 64 = 576$
卷积层 $C_{12}$	$224 \times 224 \times 64$	$3 \times 3 \times 64/1$	$224 \times 224 \times 64$	$(3 \times 3 \times 64 + 1) \times 64 = 1152$
下采样层 $S_{max1}$	$224 \times 224 \times 64$	$2 \times 2/2$	$112 \times 112 \times 64$	0
卷积层 $C_{21}$	$112 \times 112 \times 64$	$3 \times 3 \times 128/1$	$112 \times 112 \times 128$	$(3 \times 3 \times 64 + 1) \times 128 = 1152$
卷积层 $C_{22}$	$112 \times 112 \times 128$	$3 \times 3 \times 128/1$	$112 \times 112 \times 128$	$(3 \times 3 \times 128 + 1) \times 128 = 1152$
下采样层 $S_{max2}$	$112 \times 112 \times 128$	$2 \times 2/2$	$56 \times 56 \times 128$	0
卷积层 $C_{31}$	$56 \times 56 \times 128$	$3 \times 3 \times 256/1$	$56 \times 56 \times 256$	$(3 \times 3 \times 128 + 1) \times 256 = 1152$
卷积层 $C_{32}$	$56 \times 56 \times 256$	$3 \times 3 \times 256/1$	$56 \times 56 \times 256$	$(3 \times 3 \times 256 + 1) \times 256 = 1152$
卷积层 $C_{33}$	$56 \times 56 \times 256$	$3 \times 3 \times 256/1$	$56 \times 56 \times 256$	$(3 \times 3 \times 256 + 1) \times 256 = 1152$
下采样层 $S_{max3}$	$56 \times 56 \times 256$	$2 \times 2/2$	$28 \times 28 \times 256$	0
卷积层 $C_{41}$	$28 \times 28 \times 256$	$3 \times 3 \times 512/1$	$28 \times 28 \times 512$	$(3 \times 3 \times 256 + 1) \times 512 = 1152$
卷积层 $C_{42}$	$28 \times 28 \times 512$	$3 \times 3 \times 512/1$	$28 \times 28 \times 512$	$(3 \times 3 \times 512 + 1) \times 512 = 1152$
卷积层 $C_{43}$	$28 \times 28 \times 512$	$3 \times 3 \times 512/1$	$28 \times 28 \times 512$	$(3 \times 3 \times 512 + 1) \times 512 = 1152$
下采样层 $S_{max4}$	$28 \times 28 \times 512$	$2 \times 2/2$	$14 \times 14 \times 512$	0
卷积层 $C_{51}$	$14 \times 14 \times 512$	$3 \times 3 \times 512/1$	$14 \times 14 \times 512$	$(3 \times 3 \times 512 + 1) \times 512 = 1152$
卷积层 $C_{52}$	$14 \times 14 \times 512$	$3 \times 3 \times 512/1$	$14 \times 14 \times 512$	$(3 \times 3 \times 512 + 1) \times 512 = 1152$
卷积层 $C_{53}$	$14 \times 14 \times 512$	$3 \times 3 \times 512/1$	$14 \times 14 \times 512$	$(3 \times 3 \times 512 + 1) \times 512 = 1152$

网络层	输入尺寸	核尺寸	输出尺寸	参数个数
下采样层 $S_{max5}$	$14 \times 14 \times 512$	$2 \times 2/2$	$7 \times 7 \times 512$	0
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 $FC_3$	$1 \times 4096$	$4096 \times 1000$	$1 \times 1000$	$(4096 + 1) \times 1000$

### 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。

## 4.6 GoogLeNet

### 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( $Inception_{v1-v4}$ )。

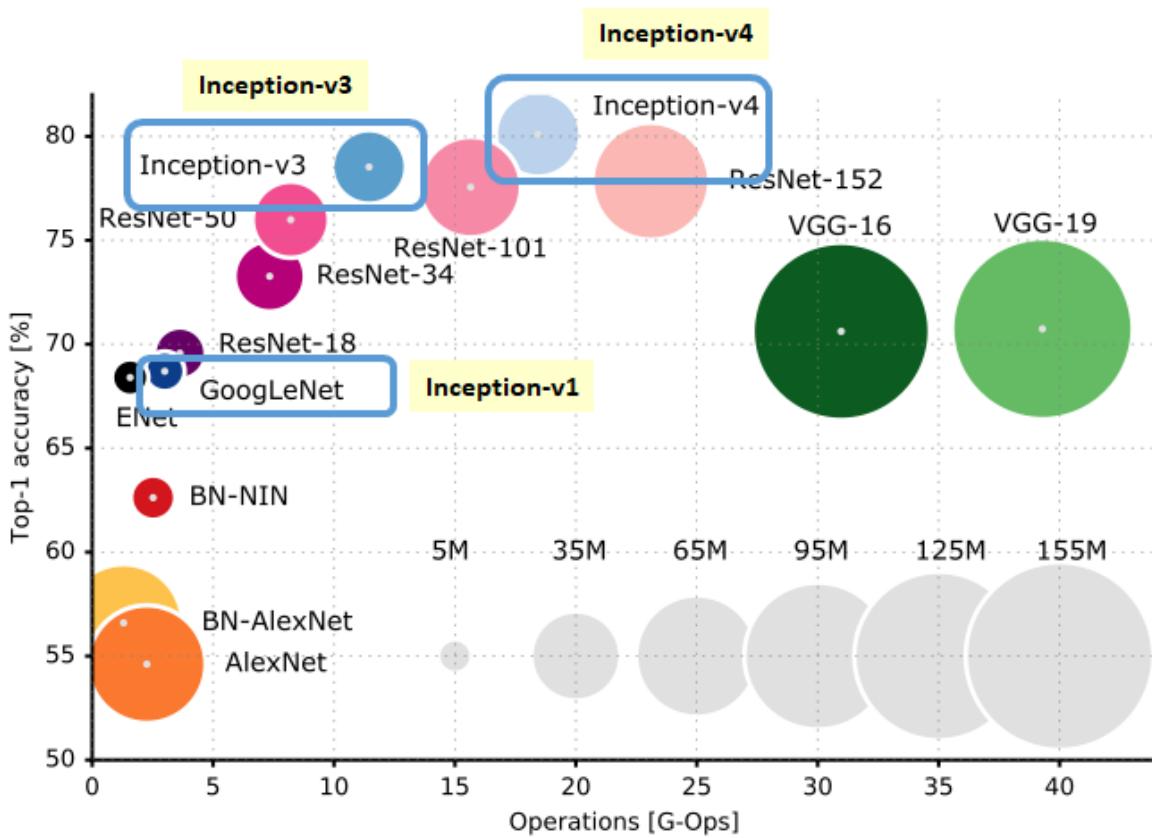


图 4.8 Inception 性能比较图

#### 4.6.2 模型结构

## GoogLeNet

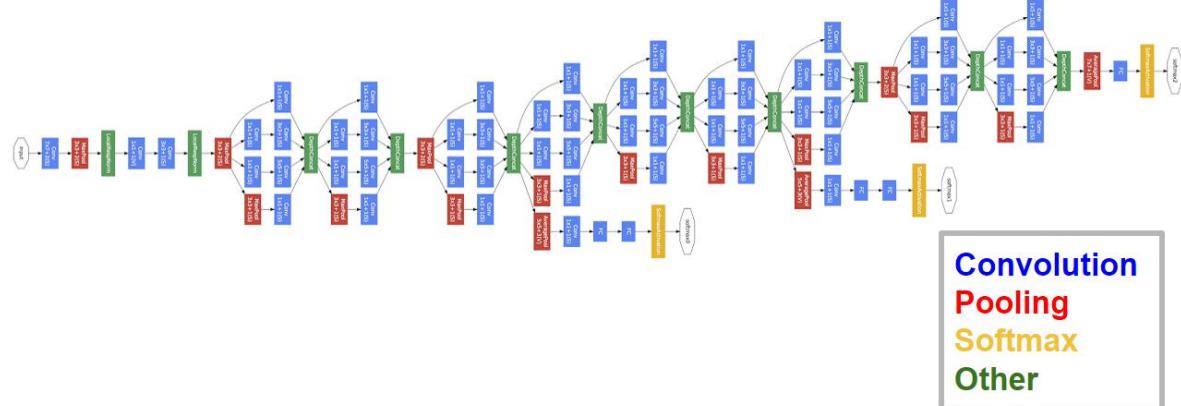
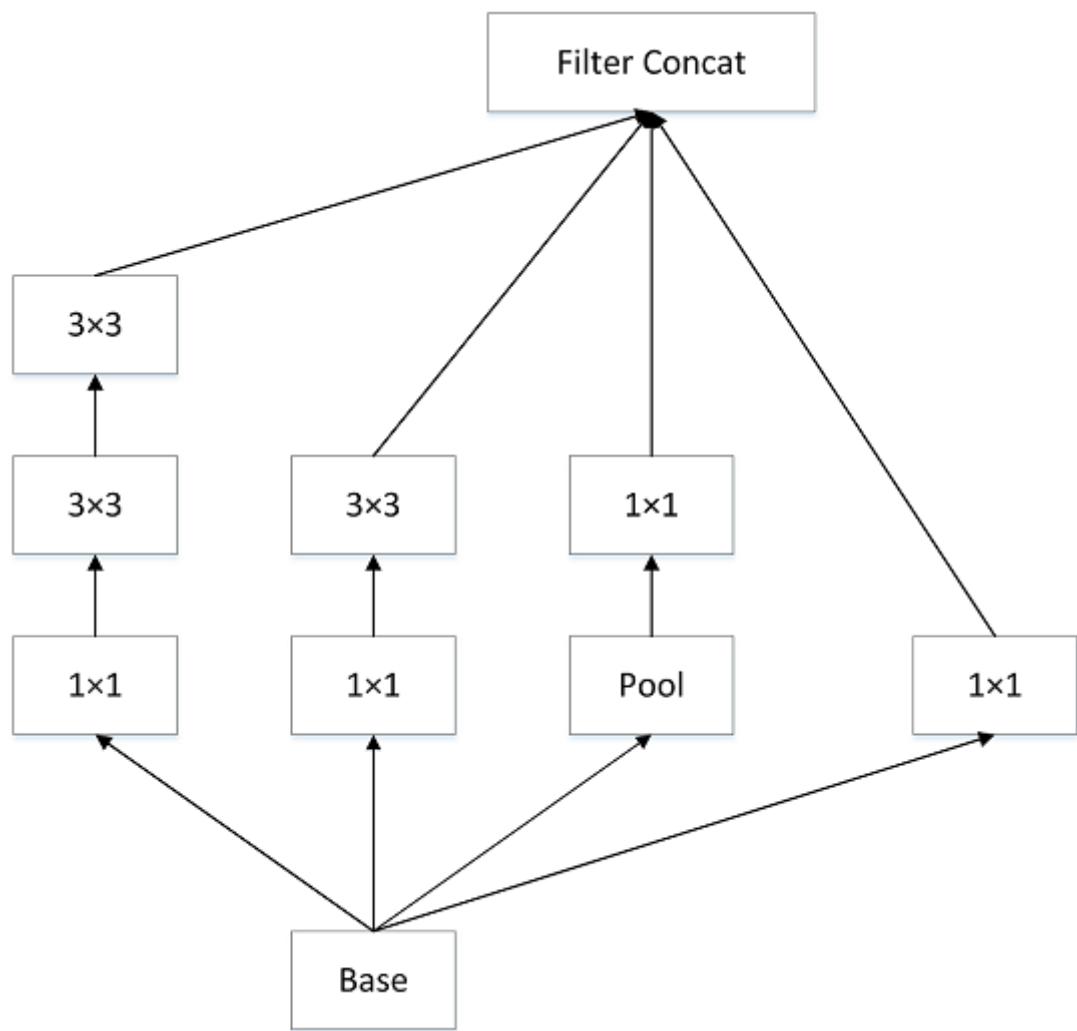


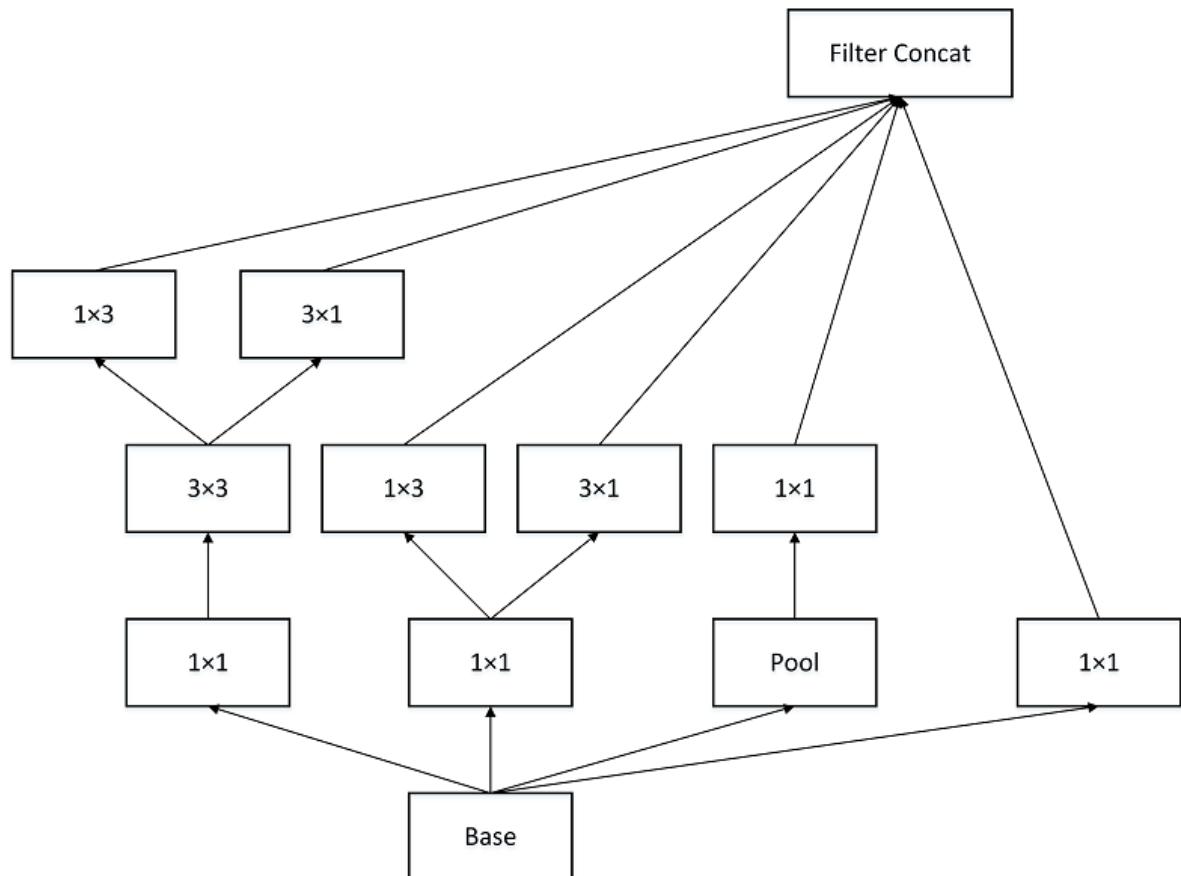
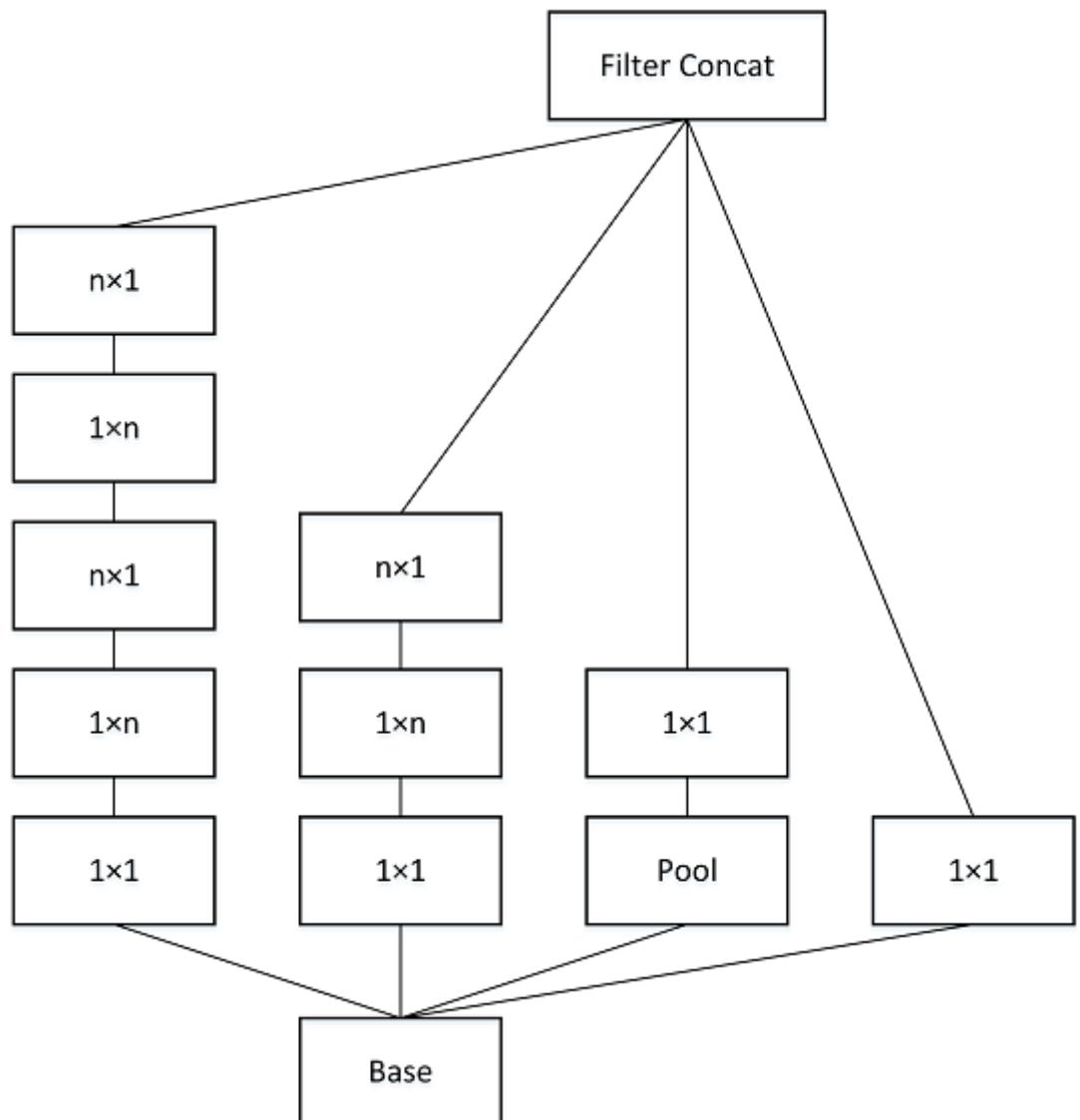
图 4.9 GoogLeNet 网络结构图

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<sub>v1</sub>*在同一层中采用不同的卷积核，并对卷积结果进行合并；*Inception<sub>v2</sub>*组合不同卷积核的堆叠形式，并对卷积结果进行合并；*Inception<sub>v3</sub>*则在*v<sub>2</sub>*基础上进行深度组合的尝试；*Inception<sub>v4</sub>*结构相比于前面的版本更加复杂，子网络中嵌套着子网络。

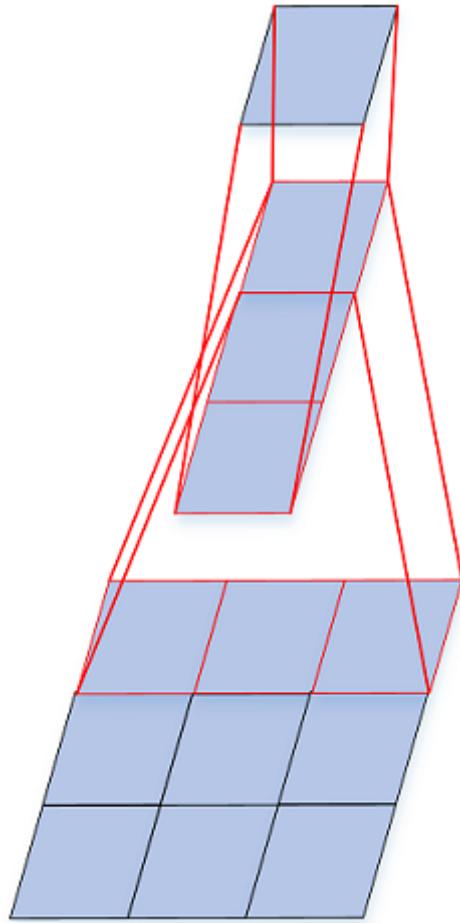
*Inception<sub>v1</sub>*

*Inception<sub>v2</sub>*

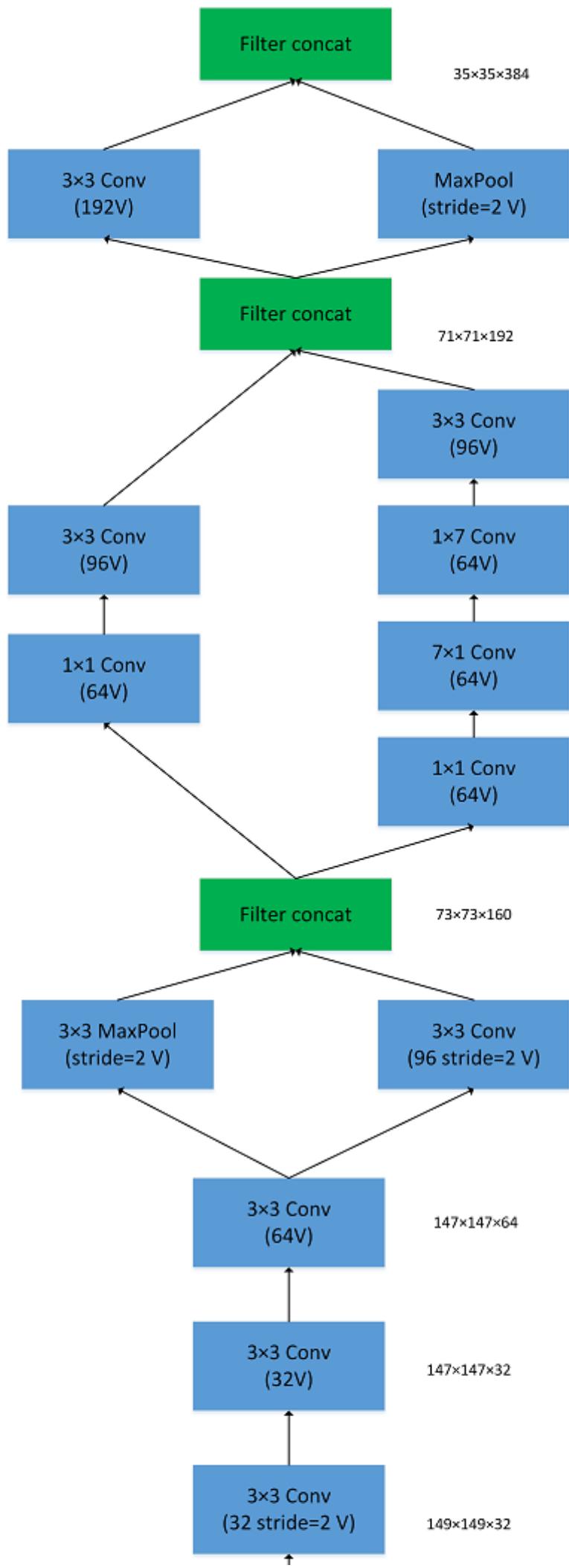




*Inception<sub>v3</sub>*



*Inception<sub>v4</sub>*



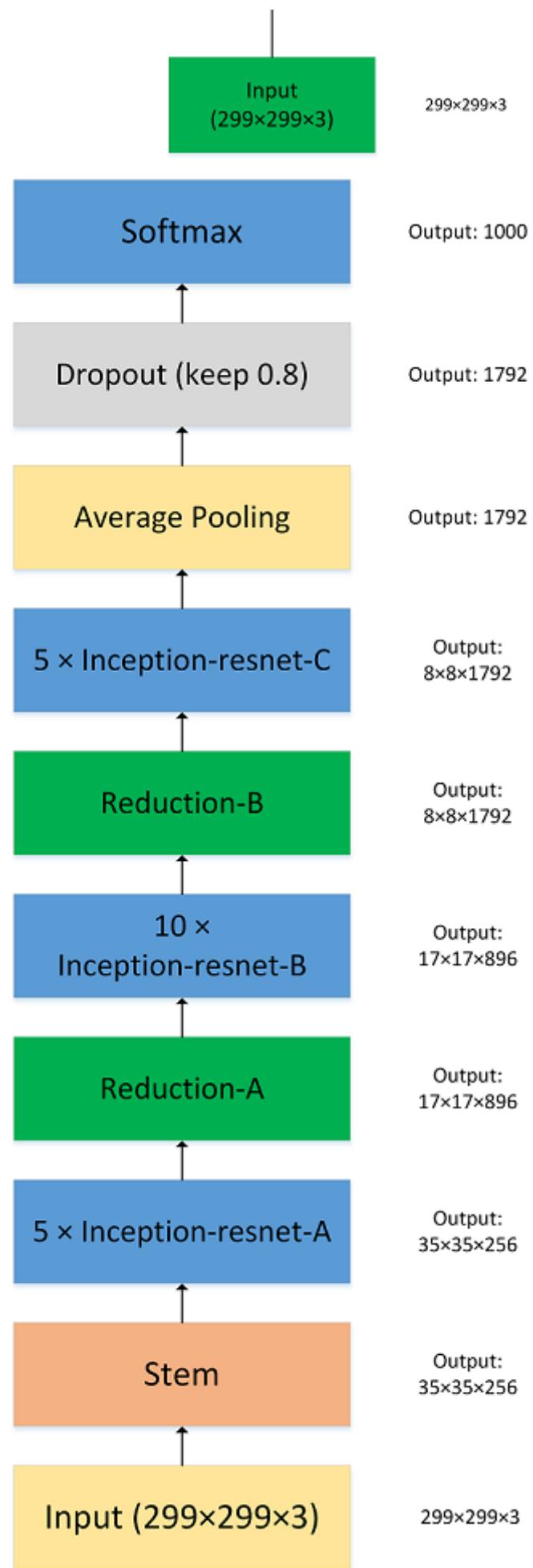


图 4.10 Inception<sub>v1-4</sub>结构图

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 $C_{42}$				