1. **损失函数**

损失函数（loss function）是用来估量你模型的预测值f(x)与真实值Y的不一致程度，它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。

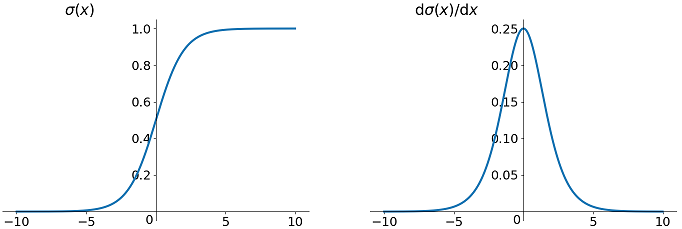
1. **深度神经网络（DNN）损失函数和激活函数的选择**

如果不用激励函数（相当于激励函数是f(x)=x），在这种情况下，每一层的输出都是上一层的线性函数，无论神经网络有多少层，输出都是输入的线性组合，这与一个隐藏层的效果相当（这种情况就是多层感知机MPL）。

对比激活函数Sigmoid、ReLu，tanh

**Sigmoid函数**

https://img-blog.csdn.net/20170903152554267?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGlsdTkxNg==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

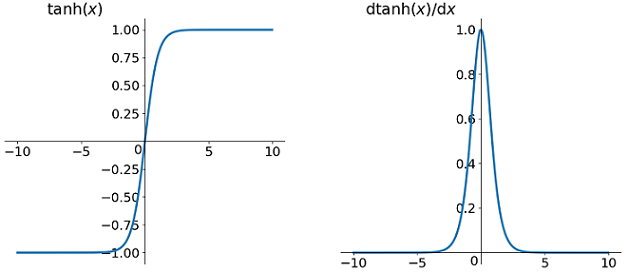


但是Sigmoid函数有3大缺点：

* 容易出现梯度消失
* 输出不是zero-centered
* 幂运算相对耗时

**tanh函数**

https://img-blog.csdn.net/20170903160024954



优点

* 全程可导；输出区间为-1到1；解决了zero-centered的输出问题

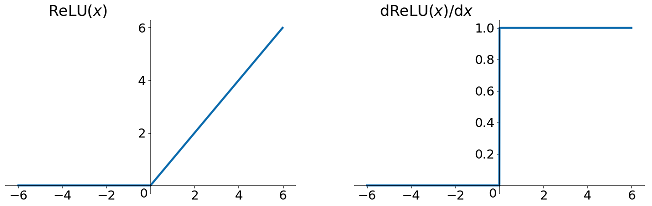
缺点

* 梯度消失的问题和幂运算的问题仍然存在

**ReLU函数**

ReLU函数(Rectified Linear Units)其实就是一个取最大值函数，注意这并不是全区间可导的，但是我们可以取次梯度(subgradient)

https://img-blog.csdn.net/20170903160343431



优点

* 解决了梯度消失的问题 (在正区间)
* 计算速度非常快，只需要判断输入是否大于0
* 收敛速度远快于sigmoid和tanh
* Relu会使一部分神经元的输出为0，这样就造成了网络的稀疏性，并且减少了参数的相互依存关系，缓解了过拟合问题的发生

缺点

* 输出不是zero-centered
* Dead ReLU Problem

Dead ReLU Problem指的是某些神经元可能永远不会被激活，导致相应的参数永远不能被更新。有两个主要原因可能导致这种情况产生: (1) 非常不幸的参数初始化，这种情况比较少见 (2) 学习速率太高导致在训练过程中参数更新太大，不幸使网络进入这种状态。解决方法是可以采用Xavier初始化方法，以及避免将学习速率设置太大或使用adagrad等自动调节学习速率的算法。

尽管存在这两个问题，ReLU目前仍是最常用的激活函数。

在深度神经网络（DNN）反向传播算法(BP)中，我们对DNN的前向反向传播算法的使用做了总结。其中使用的损失函数是均方差，而激活函数是Sigmoid. MSE损失+Sigmoid激活函数的问题. Sigmoid的这个曲线意味着在大多数时候，我们的梯度变化值很小，导致我们的W,b更新到极值的速度较慢，也就是我们的算法收敛速度较慢。

交叉熵损失+Sigmoid改进收敛速度

https://ask.qcloudimg.com/http-save/yehe-1557966/rdwaww6k2m.png?imageView2/2/w/1620

对数似然损失+softmax进行分类输出:

对于用于分类的softmax激活函数，对应的损失函数一般都是用对数似然函数，即：

https://ask.qcloudimg.com/http-save/yehe-1557966/u8vhh0652z.png?imageView2/2/w/1620

什么是梯度爆炸和梯度消失呢？简单理解，就是在反向传播的算法过程中，由于我们使用了是矩阵求导的链式法则，有一大串连乘，如果连乘的数字在每层都是小于1的，则梯度越往前乘越小，导致梯度消失，而如果连乘的数字在每层都是大于1的，则梯度越往前乘越大，导致梯度爆炸。

上面我们对DNN损失函数和激活函数做了详细的讨论，重要的点有：

* 如果使用sigmoid激活函数，则交叉熵损失函数一般肯定比均方差损失函数好
* 如果是DNN用于分类，则一般在输出层使用softmax激活函数和对数似然损失函数
* ReLU激活函数对梯度消失问题有一定程度的解决，尤其是在CNN模型中

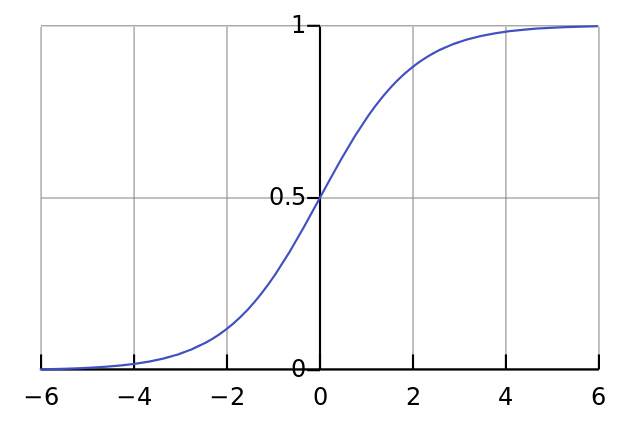
1. **SGD中S（Stochastic）代表什么**

随机选取一个样本来学习

梯度下降的优化框架有三种：批量梯度下降（全），随机梯度下降（一），小批量梯度下降（mini）。它们不同之处在于每次学习（更新模型参数）使用的样本个数，每次更新使用不同的样本会导致每次学习的准确性和学习时间不同

1. **logistic函数和softmax函数**

logistic函数/sigmod函数



softmax函数经常用在神经网络的最后一层，作为输出层，进行多分类。

Logistic具体针对的是二分类问题，而softmax解决的是多分类问题，因此从这个角度也可以理解logistic函数是softmax函数的一个特例。

对于Softmax回归当分类数目k=2的时候，Softmax将会退化为Logistic回归

logistic regression+MSE到底哪里有欠缺？梯度消失（sigmod函数的导数的性质）

logistic和cross-entropy,关于交叉熵

1. **标量、向量、张量之间的联系**

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。使用粗体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 ）

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. **张量与矩阵的区别？**
2. 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的。
3. 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。
4. 张量可以用3×3 矩阵形式来表达。4 表示标量的数和表示矢量的三维数组也可分别看作1×1，1×3 的矩阵。
5. **向量和矩阵的范数归纳**

范数（Norms）被用于度量矩阵的大小，或者相应地，度量向量的长度。范数是一个函数，它将 R^mxn（或 R^n）映射到 R。形式地说：

定义 1：任何函数满足 || · ||: R^m×n → R 和下列性质，则称为一个范数：

非负性：|| A ||≥0；|| A ||=0 当且仅当 A=0；

三角不等律：|| A+B ||≤|| A ||+|| B ||；

标量乘法律：|| αA ||=|α| || A ||，α∈R。

可以很容易地证明以下两个性质：

|| A ||=|| -A ||

| || A ||-|| B || | ≤ || A-B ||

第二个性质被称为倒三角型不等式。

定义一个矩阵A=[-1 2 -3；4 -6 6]；

* 矩阵的1范数：矩阵每一列上的元素绝对值先求和，再从中取最大值（列和最大）。上述矩阵A的1范数就是9；
* 矩阵的2范数：矩阵A^TA的最大特征值开平方根，上述矩阵A的2范数就是：10.0623；
* 矩阵的无穷范数：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)，上述矩阵的结果就是：16；

二范数相信大家在本科学线代的时候就已经被灌输了“用来度量向量长度”、“用来度量向量空间中两个点的距离”这两个典型意义。

那么在机器学习中，二范数主要有什么重要的应用呢？

二范数会削弱强特征，增强弱特征

1. **如何判断一个矩阵为正定？**

在凸优化中要用到，再细点就是在泰勒展开式的基础上判断一个函数是不是凸函数。矩阵的正定就相当于实数是否大于0，但问题是矩阵不是实数啊，放心世界上聪敏人多的是，聪敏人就想办法变成实数，即 x^{T}Hx ，称二次型（结果是个实数，这就好与0比较了），当二次型对世界上所有的每一个 x 有x^{T}Hx>0，则H正定，但是世界上的x有无穷多，不可能一一试下去，所以就另辟捷径算H的特征值，当H中的最小特征值都大于0，则H为正定。H的正定乍一看没多大用处，但是对判断一个自变量很多的函数是否为凸函数特别灵，H大于0则该函数为凸函数，这时候H就是hessian矩阵。

考虑矩阵的特征值

若所有特征值均不小于零，则称为半正定

若所有特征值均大于零，则称为正定

1. **特征值分解与特征向量**

如果把矩阵看作是运动，对于运动而言，最重要的当然就是运动的速度和方向，

* 特征值就是运动的速度
* 特征向量就是运动的方向

既然运动最重要的两方面都被描述了，特征值、特征向量自然可以称为运动（即矩阵）的特征

* 特征向量是一个向量，当在它上面应用线性变换时其方向保持不变

特征值、特征向量的应用：

控制系统，特征值 = 1 的，系统最终会趋于稳定：

图片压缩：

比如说，有下面这么一副512×512的图片（方阵才有特征值，所以找了张正方形的图），

1. **奇异值与特征值有什么关系？**

矩阵可以认为是一种线性变换，如果将这种线性变换放在几何意义上，则他的作用效果和基的选择有关

以Ax = b为例，x是m维向量，b是n维向量，m,n可以相等也可以不相等，表示矩阵可以将一个向量线性变换到另一个向量，这样一个线性变换的作用可以包含旋转、缩放和投影三种类型的效应。

奇异值分解将一个矩阵原本混合在一起的三种作用效果，分解出来了

特征值分解其实是对旋转缩放两种效应的归并, 求特征向量和特征值的过程，我们找到了这样一组基，在这组基下，矩阵的作用效果仅仅是存粹的缩放。

特征值分解和奇异值分解都是给一个矩阵(线性变换)找一组特殊的基，特征值分解找到了特征向量这组基，在这组基下该线性变换只有缩放效果。而奇异值分解则是找到另一组基，这组基下线性变换的旋转、缩放、投影三种功能独立地展示出来了。

又因为有投影效应的矩阵不是方阵，没有特征值，所以奇异值分解可以适用于所有矩阵，但特征值分解就仅仅适用于方阵了。

1. **机器学习为什么要使用概率？**

概率论是用于表示不确定性声明的数学框架。它不仅提供了量化不确定性的方法，也提供了用于导出新的不确定性声明的公理。

概率论使我们能够作出不确定的声明以及在不确定性存在的情况下进行推理，而信息论使我们能够量化概率分布中的不确定性的总量。

1. **常见概率分布？**
2. **举例理解条件概率**

P（A|B）=P(AB)/P(B)

1. **条件概率的链式法则**

2个事件同时发生的概率：

P(a, b) = P(a | b) \* P(b)

其中：P(a, b)表示 a和b事件同时发生的概率， P(a | b)是一个条件概率，表示在b事件发生的条件下，a发生的概率

3个事件的概率链式调用：

P(a, b, c) = P(a | b, c) \* P(b, c) = P(a | b, c) \* P(b | c) \* P(c)

推广到N个事件，概率链式法则长这样：

P(X1, X2, ... Xn) = P(X1 | X2, X3 ... Xn) \* P(X2 | X3, X4 ... Xn) ... P(Xn-1 | Xn) \* P(Xn)

1. **独立性和条件独立性**

独立性：

在概率论中，独立性是指随机变量的分布不因知道其它随机变量的值而改变。在机器学习中，我们通常都会对数据做这样的假设。例如，我们会假设训练样本是从某一底层空间独立提取；并且假设样例i的标签独立于样例j(i≠j)的特性。

从数学角度来说，随机变量X独立于Y，当：

* P(X)=P(X|Y)

注意，上式没有标明X,Y的取值，也就是说该公式对任意X,Y可能的取值均成立。

利用等式(2)，很容易可以证明如果X对Y独立，那么Y也独立于X。当X和Y相互独立时，记为X⊥Y。

对于随机变量X和Y的独立性，有一个等价的数学公式：

* P(X,Y)=P(X)P(Y)

我们有时也会讨论条件独立，就是当我们当我们知道一个随机变量（或者更一般地，一组随机变量）的值时，那么其它随机变量之间相互独立。正式地，我们说“给定Z，X和Y条件独立”，如果：

* P(X|Z)=P(X|Y,Z)

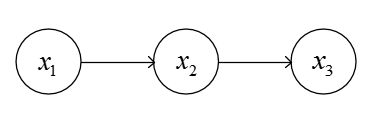
或者等价的：

P(X,Y|Z)=P(X|Z)P(Y|Z)

x1和x2相互独立

https://pic2.zhimg.com/80/v2-63035da666e9b498f3543a9de586986a_hd.jpg

在有向图模型中，若两个变量之间没有连接并且没有共同的祖先，则相互独立。条件独立性：x1和x3在给定x2时条件独立



https://pic1.zhimg.com/80/v2-0fe42f48b0f7030ff33ff85d0cb7249d_hd.jpg

意味着可对联合分布做因式分解变为条件分布的乘积并减少冗余，从而用更少的参数描述条件概率分布。

在有向图模型中，任何变量与其对应的马尔科夫覆盖是条件独立的。

1. **期望、方差、协方差、相关系数总结**

* 数学期望

数学期望是随机变量的重要特征之一,随机变量X的数学期望记为E(X),E(X)是X的算术平均的近似值,数学期望表示了X的平均值大小。

当X为离散型随机变量时,并且其分布律为 P(X=xk) ＝ pk   ,其中k=1,2,…,n；则数学期望[image](https://images0.cnblogs.com/blog/383728/201312/05170453-bdc59180ac7e4e0cb3d91125de4cdfed.png)（要求绝对收敛）.当X为连续型随机变量时,设其概率密度为f(x),则数学期望为[image](https://images0.cnblogs.com/blog/383728/201312/05170455-362c90f2dc5840ac909058d9f2235d21.png)（要求绝对收敛）.

* 方差

数学期望给出了随机变量的平均大小,现实生活中我们还经常关心随机变量的取值在均值周围的散布程度,而方差就是这样的一个数字特征。

设X是随机变量,并且E{[X-E(X)2]}存在,则称它为X的方差,记为D(X)。

当X为离散型时,D(x) = [image](https://images0.cnblogs.com/blog/383728/201312/05170502-98480c1a46db4b4bad9f26db7c131624.png).

当X为连续型时,D(x) = [image](https://images0.cnblogs.com/blog/383728/201312/05170505-50e7558dbdab43a49660ee387dadc96d.png).

方差的算术平方根[image](https://images0.cnblogs.com/blog/383728/201312/05170511-e27260356493422ea89bc4547b4d23ed.png)为X的标准差。

另外,D(X) = E{[X-E(X)2]} 经过化解可得D(X) = E(X2) – [E(X)] .我们一般计算的时候常用这个式子。

* 协方差

对于二维的随机变量(X,Y)，我们还要讨论它们的相互关系,协方差就是一个这样的数字特征。

因为E{[X-E(X)][Y-E[Y]]} = E(XY) – E(X)E(Y).

又当X,Y相互独立的时候E(XY) = E(X)E(Y).这意味着若E{[X-E(X)][Y-E[Y]]} ≠ 0 ,则X与Y是存在一定关系的。我们把E{[X-E(X)][Y-E[Y]]} 称为随机变量X与Y的协方差。记为Cov(X,Y).即：Cov(X,Y) = E{[X-E(X)][Y-E[Y]]}

* 相关系数

协方差在某种意义上是表示了两个随机变量间的关系,但是Cov(X,Y)的取值大小与X,Y的量纲有关,不方便分析,所以为了避免这一点,我们用X,Y的标准化随机变量来讨论。

我们称[image](https://images0.cnblogs.com/blog/383728/201312/05170806-7ac88c6e93904958a0ab68fe2b8f0c0d.png)为随机变量X与Y的相关系数,记为[image](https://images0.cnblogs.com/blog/383728/201312/05170809-418f3461098d4ea8958922eac2107dbd.png)(无量纲)。

其中[image](https://images0.cnblogs.com/blog/383728/201312/05170816-2f36ee8ce8c24b6ca60538eec2d81c95.png)为X,Y的协方差即Cov(X,Y),D(X),D(Y)分别是X,Y的方差且D(X)>0，D(Y)>0。

关于相关系数，我们有下面的性质：

|[image](https://images0.cnblogs.com/blog/383728/201312/05170821-7e6bf11030ed4f10b48d915a3260b4be.png)| ≤ 1

|[image](https://images0.cnblogs.com/blog/383728/201312/05170828-9e209e6121f742009ea6768fc5525c2c.png)| = 1 的充要条件是X 与 Y 以概率 1 存在线性关系，即 P{Y = a +bX} = 1, a,b是常数。

若[image](https://images0.cnblogs.com/blog/383728/201312/05170842-c2324cf2469f42cfaeabebc686c05e91.png) = 0,则说明X,Y不相关并且X与Y不存在线性关系。

若随机变量X,Y相互独立，则[image](https://images0.cnblogs.com/blog/383728/201312/05170859-23aaeb71d23b41f3ade82248cebd331c.png)= 0，即X,Y不相关。

注意：两个不相关的随机变量，不一定相互独立,有一特殊情况是,当随机变量X,Y服从二维正态分布的时候,独立与不相关等价。

不相关只能说明X与Y不存在线性关系。

独立说明X与Y既不存在线性关系,也不存在非线性关系。

第二章 机器学习基础

1. **各种常见算法:监督学习、非监督学习、半监督学习、弱监督学习？**

* 监督学习（supervised learning）:

已知数据和其对应的标签，训练一个算法，将输入数据映射到标签的过程

已知一些图片是狗，一些图片不是狗，那么训练一个算法，当一个新的图片输入算法的时候算法告诉我们这张图片是不是狗

* 无监督学习(unsupervised learning)

已知数据不知道任何标签，按照一定的偏好，训练一个智能算法，将所有的数据映射到多个不同标签的过程。所谓的按照一定的偏好，是比如特征空间距离最近，等人们认为属于一类的事物应具有的一些特点。举个例子，猪和鸵鸟混杂在一起，算法会测量高度，发现动物们主要集中在两个高度，一类动物身高一米左右，另一类动物身高半米左右，那么算法按照就近原则，75厘米以上的就是高的那类也就是鸵鸟，矮的那类是第二类也就是猪，当然这里也会出现身材矮小的鸵鸟和身高爆表的猪会被错误的分类

* 半监督学习(semi supervised learning)

已知数据和部分数据对应的标签，有一部分数据的标签未知，训练一个智能算法，学习已知标签和未知标签的数据，将输入数据映射到标签的过程。半监督通常是一个数据的标注非常困难，比如说医院的检查结果，医生也需要一段时间来判断健康与否，可能只有几组数据知道是健康还是非健康，其他的只有数据不知道是不是健康。

* 弱监督学习(weakly supervised learning)：

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，如果我们知道一幅图，告诉你图上有一只猪，然后需要你把猪在哪里，猪和背景的分界在哪里找出来，那么这就是一个已知若标签，去学习强标签的弱监督学习问题。

对于目标检测任务，图像分类的标签相比物体的bounding box是一种弱监督的标注，对于语义分割任务，image-level的标签和物体的bounding box相比pixel-level（像素层面）的标签则是一种弱监督的标注。

1. **监督学习有哪些步骤**

监督学习（Supervised Learning）

监督学习是使用已知正确答案的示例来训练网络的。想象一下，我们可以训练一个网络，让其从照片库中（其中包含你父母的照片）识别出你父母的照片。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，我们要浏览你的照片（数据集），确定所有有你父母的照片，并对其进行标注，从而开始此过程。然后我们将把整堆照片分成两堆。我们将使用第一堆来训练网络（训练数据），而通过第二堆来查看模型在选择我们父母照片操作上的准确程度（验证数据）。

等到数据集准备就绪后，我们就会将照片提供给模型。在数学上，我们的目标就是在深度网络中找到一个函数，这个函数的输入是一张照片，而当你的父母不在照片中时，其输出为0，否则输出为1。

此步骤通常称为分类任务。在这种情况下，我们进行的通常是一个结果为yes or no的训练，但事实是，监督学习也可以用于输出一组值，而不仅仅是0或1。例如，我们可以训练一个网络，用它来输出一个人偿还信用卡贷款的概率，那么在这种情况下，输出值就是0到100之间的任意值。这些任务我们称之为回归。

步骤2：训练

为了继续该过程，模型可通过以下规则（激活函数）对每张照片进行预测，从而决定是否点亮工作中的特定节点。这个模型每次从左到右在一个层上操作——现在我们将更复杂的网络忽略掉。当网络为网络中的每个节点计算好这一点后，我们将到达亮起（或未亮起）的最右边的节点（输出节点）。

既然我们已经知道有你父母的照片是哪些图片，那么我们就可以告诉模型它的预测是对还是错。然后我们会将这些信息反馈（feed back）给网络。

该算法使用的这种反馈，就是一个量化“真实答案与模型预测有多少偏差”的函数的结果。这个函数被称为成本函数（cost function），也称为目标函数（objective function），效用函数（utility function）或适应度函数（fitness function）。然后，该函数的结果用于修改一个称为反向传播（backpropagation）过程中节点之间的连接强度和偏差，因为信息从结果节点“向后”传播。

我们会为每个图片都重复一遍此操作，而在每种情况下，算法都在尽量最小化成本函数。

其实，我们有多种数学技术可以用来验证这个模型是正确还是错误的，但我们常用的是一个非常常见的方法，我们称之为梯度下降（gradient descent）。

步骤3：验证

一旦我们处理了第一个堆栈中的所有照片，我们就应该准备去测试该模型。我们应充分利用好第二堆照片，并使用它们来验证训练有素的模型是否可以准确地挑选出含有你父母在内的照片。

步骤4：使用

最后，一旦你有了一个准确的模型，你就可以将该模型部署到你的应用程序中。你可以将模型定义为API调用，例如ParentsInPicture(photo)，并且你可以从软件中调用该方法，从而导致模型进行推理并给出相应的结果。

1. **分类网络和回归的区别？**

loss function不同，分类的损失函数一般用交叉熵这种，而回归的损失函数一般用类似均方误差（MSE,mean squared error）

举几个例子:

1. Logistic Regression 和 Linear Regression：

* Linear Regression： 输出一个标量 wx+b，这个值是连续值，所以可以用来处理回归问题
* Logistic Regression：把上面的 wx+b 通过 sigmoid 函数映射到(0,1)上，并划分一个阈值, 大于阈值的分为一类，小于等于分为另一类，可以用来处理二分类问题
* 更进一步：对于N分类问题，则是先得到N组w值不同的 wx+b，然后归一化，比如用softmax 函数，最后变成N个类上的概率，可以处理多分类问题

2. Support Vector Regression 和 Support Vector Machine:

* SVR：输出 wx+b，即某个样本点到分类面的距离，是连续值，所以是回归模型
* SVM：把这个距离用 sign(·) 函数作用，距离为正(在超平面一侧)的样本点是一类，为负的是另一类，所以是分类模型

3. Naive Bayes 用于分类 和 回归:

* 用于分类：y是离散的类别，所以得到离散的 p(y|x)，给定 x ，输出每个类上的概率
* 用于回归：对上面离散的 p(y|x)求期望 ΣyP(y|x)，就得到连续值。但因为此时y本身是连续的值，所以最地道的做法是，得到连续的概率密度函数p(y|x)，然后再对y求期望。

4. 前馈神经网络(如 CNN 系列) 用于 分类 和 回归:

* 用于回归：最后一层有m个神经元，每个神经元输出一个标量，m个神经元的输出可以看做向量 v，现全部连到一个神经元上，则这个神经元输出 wv+b，是一个连续值，可以处理回归问题，跟上面 Linear Regression 思想一样
* 用于N分类：现在这m个神经元最后连接到 N 个神经元，就有 N 组w值不同的 wv+b，同理可以归一化（比如用 softmax ）变成 N个类上的概率（补充一下，如果不用 softmax，而是每个 wx+b 用一个 sigmoid，就变成多标签问题，跟多分类的区别在于，样本可以被打上多个标签）

5. 循环神经网络(如 RNN 系列) 用于分类 和 回归：

* 用于回归 和 分类： 跟 CNN 类似，输出层的值 y = wv+b，可做分类可做回归，只不过区别在于，RNN 的输出跟时间有关，即输出的是 {y(t), y(t+1),...}序列

1. **常用分类算法的优缺点？**

* 决策树

一种启发式算法，核心是在决策树各个节点上应用信息增益等准则来选取特征，进而递归地构造决策树

优点：

1. 计算复杂度不高，易于理解和解释，可以理解决策树所表达的意义；

2. 数据预处理阶段比较简单，且可以处理缺失数据；

3. 能够同时处理数据型和分类型属性，且可对有许多属性的数据集构造决策树；

4. 是一个白盒模型，给定一个观察模型，则根据所产生的决策树很容易推断出相应的逻辑表达式；

5. 在相对短的时间内能够对大数据集合做出可行且效果良好的分类结果。

6. 可以对有许多属性的数据集构造决策树。

缺点：

1. 对于那些各类别样本数目不一致的数据，信息增益的结果偏向于那些具有更多数值的属性；

2. 对噪声数据较为敏感；

3. 容易出现过拟合问题；

4. 忽略了数据集中属性之间的相关性；

5.处理缺失数据时的困难

* KNN算法

一种惰性分类方法，从训练集中找出k个最接近测试对象的训练对象，再从这k个训练对象中找出居于主导的类别，将其赋给测试对象。

优点：

1. 简单有效，容易理解和实现；

2. 重新训练的代价较低（类别体系的变化和训练集的变化）；

3. 计算时间和空间线性于训练集的规模；

4. 错误率渐进收敛于贝叶斯错误率，可作为贝叶斯的近似；

5. 适合处理多模分类和多标签分类问题；

6. 对于类域的交叉或重叠较多的待分类样本集较为适合；

缺点：

1. 是懒散学习方法，比一些积极学习的算法要慢；

2. 计算量比较大，需对样本点进行剪辑；

3. 对于样本不平衡的数据集效果不佳，可采用加权投票法改进；

4. k值的选择对分类效果有很大影响，较小的话对噪声敏感，需估计最佳k值。

5.可解释性不强，计算量大。

* 朴素贝叶斯算法

贝叶斯分类器的分类原理是利用各个类别的先验概率，再利用贝叶斯公式及独立性假设计算出属性的类别概率以及对象的后验概率，即该对象属于某一类的概率，选择具有最大后验概率的类作为该对象所属的类别。

优点：

1. 数学基础坚实，分类效率稳定，容易解释；

2. 所需估计的参数很少，对缺失数据不太敏感；

3. 无需复杂的迭代求解框架，适用于规模巨大的数据集。

缺点：

1. 属性之间的独立性假设往往不成立（可考虑用聚类算法先将相关性较大的属性进行聚类）；

2. 需要知道先验概率，分类决策存在错误率。

* SVM算法

对于两类线性可分学习任务，SVM找到一个间隔最大的超平面将两类样本分开，最大间隔能够保证该超平面具有最好的泛化能力。

优点：

1. 可以解决小样本情况下的ML问题；

2. 可以提高泛化性能；

3. 可以解决高维问题，避免维数灾难；

4. 可以解决非线性问题；

5. 可以避免神经网络结构选择和局部极小点问题。

参数C和g的选择对分类性能的影响：

C是惩罚系数，C越大，交叉validation高，容易过学习；

g是核函数的到达0的速率，g越小，函数下降快，交叉validation高，也容易造成过学习。

缺点：

1. 对缺失数据敏感；

2. 对非线性问题没有通用解决方案，必须谨慎选择kernel function来处理。

* Logistic回归算法

二项logistic回归模型是一种分类模型，由条件概率分布P(Y|X)表示，形式为参数化的logistic分布。这里随机变量X取值为实数，随机变量Y取值为1或0。可以通过有监督的方法来估计模型参数。

优点：

1. 计算代价不高，易于理解和实现；

2. 适用于数值型和分类型数据。

缺点：

1. 容易欠拟合；

2. 分类精度可能不高。

* 人工神经网络

优点：

1. 分类的准确度高，并行分布处理能力强，分布存储及学习能力强；

2. 对噪声神经有较强的鲁棒性和容错能力，能充分逼近复杂的非线性关系，具备联想记忆的功能等。

缺点：

1. 神经网络需要大量的参数，如网络拓扑结构、权值和阈值的初始值；

2. 不能观察之间的学习过程，输出结果难以解释，会影响到结果的可信度和可接受程度；

3. 学习时间过长,甚至可能达不到学习的目的。

以下是我这些年总结的指南训练集有多大？

* 如果你的训练集很小，高偏差/低方差的分类器（如朴素贝叶斯）比低偏差/高方差的分类器（如K近邻或Logistic回归）更有优势，因为后者容易过拟合。但是随着训练集的增大，高偏差的分类器并不能训练出非常准确的模型，所以低偏差/高方差的分类器会胜出（它们有更小的渐近误差）。你也可以从生成模型与鉴别模型的区别来考虑它们。
* 某些分类器的优势，朴素贝叶斯(Naive Bayes, NB)超级简单，就像做一些数数的工作。如果条件独立假设成立的话，NB将比鉴别模型（如Logistic回归）收敛的更快，所以你只需要少量的训练数据。即使条件独立假设不成立，NB在实际中仍然表现出惊人的好。如果你想做类似半监督学习，或者是既要模型简单又要性能好，NB值得尝试。
* Logistic回归(Logistic Regression, LR) LR有很多方法来对模型正则化。比起NB的条件独立性假设，LR不需要考虑样本是否是相关的。与决策树与支持向量机（SVM）不同，NB有很好的概率解释，且很容易利用新的训练数据来更新模型（使用在线梯度下降法）。如果你想要一些概率信息（如，为了更容易的调整分类阈值，得到分类的不确定性，得到置信区间），或者希望将来有更多数据时能方便的更新改进模型，LR是值得使用的。
* 决策树（Decision Tree, DT）DT容易理解与解释（对某些人而言——不确定我是否也在他们其中）。DT是非参数的，所以你不需要担心野点（或离群点）和数据是否线性可分的问题（例如，DT可以轻松的处理这种情况：属于A类的样本的特征x取值往往非常小或者非常大，而属于B类的样本的特征x取值在中间范围）。DT的主要缺点是容易过拟合，这也正是随机森林（Random Forest, RF）（或者Boosted树）等集成学习算法被提出来的原因。此外，RF在很多分类问题中经常表现得最好（我个人相信一般比SVM稍好），且速度快可扩展，也不像SVM那样需要调整大量的参数，所以最近RF是一个非常流行的算法。
* 支持向量机（Support Vector Machine, SVM）很高的分类正确率，对过拟合有很好的理论保证，选取合适的核函数，面对特征线性不可分的问题也可以表现得很好。SVM在维数通常很高的文本分类中非常的流行。由于较大的内存需求和繁琐的调参，我认为RF已经开始威胁其地位了。

1. **正确率能很好的评估分类算法吗？分类算法的评估方法？**

AUC/ROC就是用于分类的性能度量标准

ROC曲线：接收者操作特征曲线（receiver operating characteristic curve），是反映敏感性和特异性连续变量的综合指标，roc曲线上每个点反映着对同一信号刺激的感受性

* 横坐标：1-Specificity，伪正类率(False positive rate， FPR)，预测为正但实际为负的样本占所有负例样本的比例；
* 纵坐标：Sensitivity，真正类率(True positive rate， TPR)，预测为正且实际为正的样本占所有正例样本的比例。

AUC (Area Under Curve) 被定义为ROC曲线下的面积，显然这个面积的数值不会大于1。又由于ROC曲线一般都处于y=x这条直线的上方，所以AUC的取值范围一般在0.5和1之间。

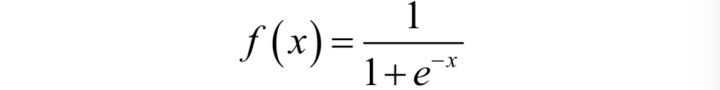
* AUC = 1，是完美分类器，采用这个预测模型时，存在至少一个阈值能得出完美预测。绝大多数预测的场合，不存在完美分类器。
* 0.5 < AUC < 1，优于随机猜测。这个分类器（模型）妥善设定阈值的话，能有预测价值。
* AUC = 0.5，跟随机猜测一样（例：丢铜板），模型没有预测价值。
* AUC < 0.5，比随机猜测还差；但只要总是反预测而行，就优于随机猜测。

1. **理解逻辑回归**

logistic回归由Cox在1958年提出[1]，它的名字虽然叫回归，但这是一种二分类算法，并且是一种线性模型。由于是线性模型，因此在预测时计算简单，在某些大规模分类问题，如广告点击率预估（CTR）上得到了成功的应用。如果你的数据规模巨大，而且要求预测速度非常快，则非线性核的SVM、神经网络等非线性模型已经无法使用，此时logistic回归是你为数不多的选择。

logistic回归源于一个非常朴素的想法：对于二分类问题，能否直接预测出一个样本 属于正样本的概率值？

x的取值范围可以是(−∞ ,+∞ )，现在想想，哪些函数能够将一个(−∞ ,+∞ )之内的实数值变换到区间[0,1]？



机器学习中被广为使用的logistic函数，也叫sigmoid函数，它有一个迷人的性质，单调增，并且定义域是(−∞ ,+∞ )，值域是(0,1)

logistic回归预测的是样本属于某一类的概率，样本的类别标签为离散的1或者0，因此不适合直接用欧氏距离误差来定义损失函数，这里通过最大似然估计来确定参数。

除了梯度下降法这种一阶优化技术，还可以使用牛顿法及其变种，如BFGS算法。

**总结**

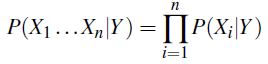
logistic回归是一种二分类算法，它用logistic函数预测出一个样本属于正样本的概率值。预测时，并不需要真的用logistic函数映射，而只需计算一个线性函数，因此是一种线性模型。训练时，采用了最大似然估计，优化的目标函数是一个凸函数，因此能保证收敛到全局最优解。虽然有概率值，但logistic回归是一种判别模型而不是生成模型，因为它并没有假设样本向量x所服从的概率分布，即没有对p(x, y)建模，而是直接预测类后验概率p(y|x)的值。

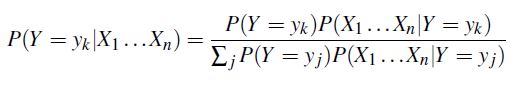
推广到多类

logistic回归只能用于二分类问题，将它进行推广可以得到处理多类分类问题的softmax回归，思路类似，采用指数函数进行变换，然后做归一化。这种变换在神经网络尤其是深度学习中被广为使用，对于多分类问题，神经网络的最后一层往往是softmax层（不考虑损失函数层，它只在训练时使用）。

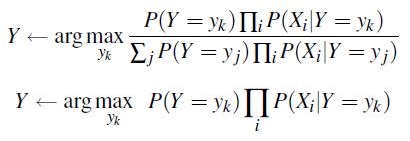
1. **逻辑回归与朴素贝叶斯有什么区别？**

* Naive Bayes是一个生成模型，在计算P(y|x)之前，先要从训练数据中计算P(x|y)和P(y)的概率，从而利用贝叶斯公式计算P(y|x)。
* Logistic Regression是一个判别模型，它通过在训练数据集上最大化判别函数P(y|x)学习得到，不需要知道P(x|y)和P(y)。
* Naive Bayes是建立在条件独立假设基础之上的，设特征X含有n个特征属性（X1，X2，...Xn），那么在给定Y的情况下，X1，X2，...Xn是条件独立的





如果要判定X属于哪一类，可以这样计算：

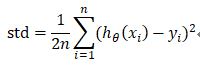


* Logistic Regression的限制则要宽松很多，如果数据满足独立假设，Logistic Regression能够取得非常好的效果；当数据不满度条件独立假设时，Logistic Regression仍然能够通过调整参数让模型最大化的符合数据的分布，从而训练得到在现有数据集下的一个最优模型。
* 当数据集比较小的时候，应该选用Naive Bayes，为了能够取得很好的效果，数据的需求量为O(log n)
* 当数据集比较大的时候，应该选用Logistic Regression，为了能够取得很好的效果，数据的需求量为O( n)
* Naive Bayes运用了比较严格的条件独立假设，为了计算P(y|x)，我们可以利用统计的方法统计数据集中P(x|y)和P(y)出现的次数，从而求得P(x|y)和P(y)。因而其所需的数据量要小一些，为O(log n).
* Logistic Regression在计算时，是在整个参数空间进行线性搜索的，需要的数据集就更大，为O( n)

1. **常见代价函数？**

代价函数是学习模型优化时的目标函数或者准则，通过最小化代价函数来优化模型。

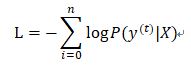
* 均方差代价函数



在线性回归模型里面提出来的

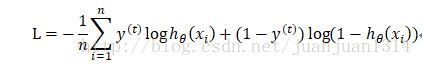
* 对数损失函数

对数似然作为代价函数是在RNN中

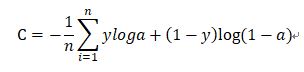


表示真实目标在数据集中的条件概率的负对数。其意义在于，在很多预测目标概率的模型中，将最大概率对应的类型作为输出类型，因此，真实目标的预测概率越高，分类越准确，学习的目标是真实目标的预测概率最大化。而概率是小于1的，其对数值小于0，且对数是单调递增的，因此，当负对数最小化，就等同于对数最大化，概率最大化。

逻辑回归中的代价函数实际上就是对数似然的特殊表示的方式：



* 交叉熵



交叉熵在神经网络中基本都用交叉熵作为代价函数

对数似然函数来作为softmax回归的代价函数

输出层神经元是sigmoid函数，可以采用交叉熵代价函数。而深度学习中更普遍的做法是将softmax作为最后一层，此时常用的代价函数是对数似然函数。

对数似然函数与softmax的组合和交叉熵与sigmoid函数的组合非常相似。对数似然函数在二分类时可以化简为交叉熵代价函数的形式

1. **为什么用交叉熵代替二次代价函数**

MSE和sigmoid函数的性质，导致σ′(z)在z取大部分值时会很小（如下图标出来的两端，几近于平坦），这样会使得w和b更新非常慢

交叉熵，当误差大的时候，权重更新就快，当误差小的时候，权重的更新就慢。这是一个很好的性质

1. **什么是损失函数？ 常见的损失函数？**

Loss Function 是定义在单个样本上的，算的是一个样本的误差。

Cost Function 是定义在整个训练集上的，是所有样本误差的平均，也就是损失函数的平均。

Object Function（目标函数 ）定义为：Cost Function + 正则化项。

The loss function computes the error for a single training example; the cost function is the average of the loss funcitons of the entire training set.

1. **逻辑回归为什么使用对数损失函数？**

对数损失在单个数据点上的定义为：

Cost (y,p(y|x)) = −y lnp(y|x)−(1−y)ln(1−p(y|x))

Cost (y,p(y|x))=−y ln⁡p(y|x)−(1−y)ln⁡(1−p(y|x))

全体样本的损失函数则可表达为：

cost(y,p(y|x))=−∑i=1m[yilnp(yi|xi)+(1−yi)ln(1−p(yi|xi))]

cos⁡t(y,p(y|x))=−∑i=1m[yiln⁡p(yi|xi)+(1−yi)ln⁡(1−p(yi|xi))]

可以看到，这个对数损失函数与上面的极大似然估计的对数似然函数本质上是等价的。所以逻辑回归直接采用对数损失函数来求参数，实际上与采用极大似然估计来求参数是一致的。

1. **对数损失函数是如何度量损失的？**

（1）有一类概率型的目标函数，例如逻辑回归来解决二分类问题，假设其目标函数为p(x)，可以简单理解为样本x归属到某一类别的概率。

（2）根据最大似然估计的理论，优化目标是使得P(X)=p(x1)(1-p(x2))p(x3)，最大化（这里假设x1和x3是正例，x2是负例，由于目标函数是求正例的概率，所以1-p(x)自然就是负例的概率）。

（3）乘法表达式求极值比较麻烦，所以最好想办法转化成加法表达式。最自然的想法是两边取对数，把等式右边转化为加法表达式。由于对数单调增，那么求P(X)的最大值的问题，可以转化为求logP(X) 的最大值的问题。

（4）求logP(X)的最大值，其实就是求-logP(X)的最小值。这个-logP(X)其实就是所谓的log loss了。

（5）log loss本身的求解可以使用梯度下降等各种方法。log loss只代表了一个从原始的loss到log形式loss的转化过程。

1. **机器学习中为什么需要梯度下降？梯度下降法缺点？如何对梯度下降法进行调优？**

梯度下降是机器学习中用来求最小值的算法，它被广泛应用于像逻辑回归、线性回归和神经网络的模型中。

梯度下降法是一种迭代型的优化算法，根据初始点在每一次迭代的过程中选择下降法方向，进而改变需要修改的参数，梯度下降法的详细过程如下：

Start at a random point

Repeat

Determine a descent direction

Choose a step size

Update

Until stopping criterion is satisfied

1. **梯度下降、随机梯度下降与批梯度下降算法之间的比较**

这三种算法都用于反向传播的优化损失函数算法。在每轮迭代中更新一次权重w，根据多次迭代，最终无限的靠近我们预期的权重最优值。

* 梯度下降算法：

(1) 如果数据集比较小，完全可以采用全数据集(Full Batch Learning)的形式，采用全数据有两个好处：

a. 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。

b. 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。

(2) 但是梯度下降也存在一下这些缺点：

a. 梯度下降算法并不能保证被优化函数达到全局最优解，只有当损失函数为凸函数时,梯度下降算法才能保证达到全局最优解，因此不能保证一定达到全局最优，受限于损失函数是否为凸函数。

b. 梯度下降算法的另外一个问题是计算时间太长，因为要在全部训练数据上最小化损失，在海量训练数据下，这样是十分耗时。

* 随机梯度下降算法:

为了加速训练的过程，可以使用随机梯度下降算法(stochastic gradient descent-SGD)。随机梯度下降算法也是称为"在线学习"。

(1) 这个算法优化的不是在全部训练数据上的损失函数，而是在每轮迭代中，随机优化某一条训练数据上的损失函数，这样每一轮参数的更新速度大大加快。

(2) 但是带来了如下问题：

在某一条数据上损失函数更小并不代表在全部数据上的损失函数更小，于是使用随机梯度下降优化得到的神经网络甚至可能无法达到全局最优。

* 批梯度下降算法：

为了综合梯度下降算法和随机梯度下降算法的优缺点，在实际应用中一般采用这两个算法的这种----->每次计算一小部分训练数据的损失函数。批梯度下降算法（Mini-batches Learning）在深度学习很多算法的反向传播算法中非常常用。这一小部分训练数据也称为一个batch，因此也引入了batch\_size的概念，batch\_size顾名思义就是来度量每一个batch中实例的个数。

(1) 引入batch有很好的优势：

a. 通过矩阵运算，每次在一个batch上优化神经网络参数并不会比单个数据慢太多。

b. 每次使用一个batch可以大大减小收敛所需要的迭代次数，同时可以使收敛到的结果更加接近梯度下降的效果。

但是批梯度下降算法带来一个问题，那就是如何选取最优的batch\_size?

(2) 可不可以选择一个适中的 Batch\_Size 值呢？

当然可以，这就是批梯度下降法(Mini-batches Learning)。因为如果数据集足够充分，那么用一半(甚至少得多)的数据训练算出来的梯度与用全部数据训练出来的梯度是几乎一样的。

(3) 在合理范围内，增大 Batch\_Size 有何好处？

a. 内存利用率提高了，大矩阵乘法的并行化效率提高。

b. 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。

c. 在一定范围内，一般来说 Batch\_Size 越大，其确定的下降方向越准，引起训练震荡越小。

(4) 盲目增大 Batch\_Size 有何坏处？

a. 内存利用率提高了，但是内存容量可能撑不住了。

b. 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。

c. 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。

1. **决策树**

* 决策树的基本原理

决策树（decision tree）是一个树结构（可以是二叉树或非二叉树）。

其每个非叶节点表示一个特征属性上的测试，每个分支代表这个特征属性在某个值域上的输出，而每个叶节点存放一个类别。

使用决策树进行决策的过程就是从根节点开始，测试待分类项中相应的特征属性，并按照其值选择输出分支，直到到达叶子节点，将叶子节点存放的类别作为决策结果。

决策树模型核心是下面几部分：

1. 结点和有向边组成
2. 结点有内部结点和叶结点俩种类型
3. 内部结点表示一个特征，叶节点表示一个类

* 决策树的三要素？

决策树学习算法包括三部分：特征选择、树的生成和树的剪枝

* 决策树算法优缺点

优点：

得出结果的过程易于理解，比神经网络等黑箱操作好理解，计算量较小，比其他算法计算的较快

缺点：

很容易造成过拟合，需要采用剪枝操作，当类别太多时，会很复杂

* 剪枝处理的作用及策略？

剪枝是决策树算法对付过拟合的主要手段。因为决策树在划分过程中为了尽可能正确分类训练样本，可能会不断重复，产生过拟合现象。即可能会为了分类正确，将具有某一些属性的样本强行划分到某一类中，而这些样本本不应该属于这一类，只是在这个数据集中属于了这一类，不具有普遍的意义，这样就差生了过拟合。为了降低过拟合的风险可以主动去掉一些分支。剪枝可以分为后剪枝和预剪枝，预剪枝在决策树生成过程中进行，后剪枝在决策树生成完后进行。预剪枝可能产生欠拟合风险，后剪枝效果更好，但是训练时间要长一些。

预剪枝（pre-pruning）：预剪枝就是在构造决策树的过程中，先对每个结点在划分前进行估计，若果当前结点的划分不能带来决策树模型泛华性能的提升，则不对当前结点进行划分并且将当前结点标记为叶结点。

后剪枝（post-pruning）：后剪枝就是先把整颗决策树构造完毕，然后自底向上的对非叶结点进行考察，若将该结点对应的子树换为叶结点能够带来泛华性能的提升，则把该子树替换为叶结点。

* 熵的概念以及理解，信息增益的理解

信息熵是度量样本集合纯度最常用的指标。假定当前样本集合D中第k类样本所占的比例为https://private.codecogs.com/gif.latex?p_k%28k%3D1%2C2%2C%5Ccdots%20%2C%7Cy%7C%29,则D的信息熵为

Ent(D)=-\sum_{k=1}^{|y|}p_klog_2p_k（1）

https://private.codecogs.com/gif.latex?Ent%28D%20%29，的值越小，则D的纯度越高

假定离散属性a有V个可能的取值\left \{ a^1,a^2,\cdots ,a^V \right \},若使用a来对样本集D进行划分，则会产生V个分支节点，其中第个分支节点包含了D中所有在属性a上取值a^v为的样本D^v，记为.根据（1）式考虑到不同的分支节点包含的样本数不同，可得用属性a对样本集D进行划分所获得的信息增益为：

Gain(D,a)=Ent(D)-\sum_{v=1}^{V}{}\frac{D^v}{D}Ent(D^v)

一般而言，信息增益越大，说明使用属性a来进行划分所获得的纯度提升越大，因此所找的属性为a^*=\underset{a\in A}{argmax}Gain(D,a)

ID3算法就是采用信息增益作为特征选择的标准

在上边所介绍的信息增益准则，它对可取值较多的属性有所偏好，为减少这种偏好所带来的不利影响，C4.5决策树算法不直接使用信息增益，而是使用增益率来选择最优划分属性。信息增益率定义为：

Gain\_ratio(D,a)=\frac{Gain(D,a)}{IV(a)}

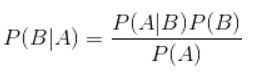
其中

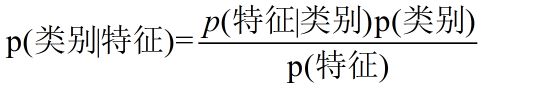
IV(a)=-\sum_{v=1}^{V}\frac{|D^v|}{|D|}log_2\frac{|D^v|}{|D|}

称为属性a的固有值，属性a的可能取值数目越多，则IV(a)的值通常会越大。

1. **贝叶斯**

贝叶斯分类是一类分类算法的总称，这类算法均以贝叶斯定理为基础，故统称为贝叶斯分类。而朴素朴素贝叶斯分类是贝叶斯分类中最简单，也是常见的一种分类方法。





朴素贝叶斯分类有朴素一词的来源，朴素贝叶斯算法是假设各个特征之间相互独立

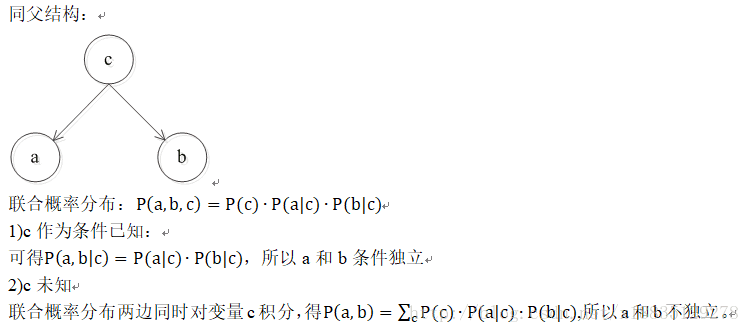
* 极大似然估计

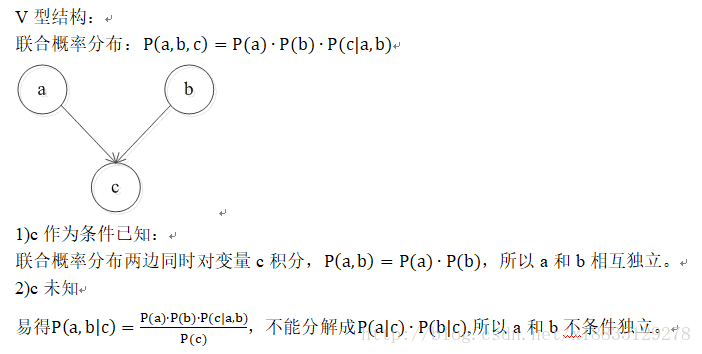
1. 得求解最大似然估计的一般过程为:
2. 写出似然函数
3. 如果无法直接求导的话，对似然函数取对数求导数
4. 求解模型中参数的最优值

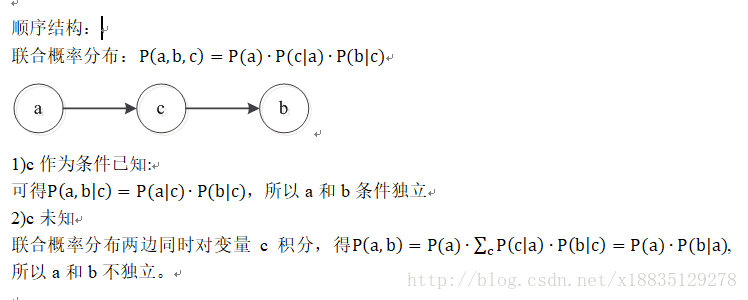
* 朴素与半朴素贝叶斯分类器

朴素贝叶斯分类器的一个重要假定：分类对应的各个属性间是相互独立的，然而在现实应用中，这个往往难以做到，那怎么办呢？很简单，适当考虑一部分属性间的相互依赖关系，这种放松后的分类称为半朴素贝叶斯分类，其中最常用的策略：假定每个属性仅依赖于其他最多一个属性，称其依赖的这个属性为其超父属性，这种关系称为：独依赖估计（ODE）

* 贝叶斯网三种典型结构







1. **前向传播与反向传播？**

反向传播法（BackPropagation），就是一个链式求导法则反复用

1. **什么是超参数？ 如何寻找超参数的最优值？ 超参数搜索一般过程？**

基础概念

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，在机器学习过程中需要对超参数进行优化，给学习器选择一组最优超参数，以提高学习的性能和效果。比如，树的数量或树的深度，学习率（多种模式）以及k均值聚类中的簇数等都是超参数。

与超参数区别的概念是参数，它是模型训练过程中学习到的一部分，比如回归系数，神经网络权重等。简单的描述参数是模型训练获得的，超参数是人工配置参数（本质上是参数的参数，每次改变超参数，模型都要重新训练）。

调参问题

在深度神经网络中，超参数的调整是一项必备技能，通过观察在训练过程中的监测指标如损失loss和准确率来判断当前模型处于什么样的训练状态，及时调整超参数以更科学地训练模型能够提高资源利用率。在本研究中使用了以下超参数，下面将分别介绍并总结了不同超参数的调整规则。

（1）学习率

学习率（learning rate或作lr）是指在优化算法中更新网络权重的幅度大小。学习率可以是恒定的、逐渐降低的，基于动量的或者是自适应的。不同的优化算法决定不同的学习率。当学习率过大则可能导致模型不收敛，损失loss不断上下震荡；学习率过小则导致模型收敛速度偏慢，需要更长的时间训练。通常lr取值为[0.01,0.001,0.0001]

（2）批次大小batch\_size

批次大小是每一次训练神经网络送入模型的样本数，在卷积神经网络中，大批次通常可使网络更快收敛，但由于内存资源的限制，批次过大可能会导致内存不够用或程序内核崩溃。bath\_size通常取值为[16,32,64,128]

（3）优化器optimizer

目前Adam是快速收敛且常被使用的优化器。随机梯度下降(SGD)虽然收敛偏慢，但是加入动量Momentum可加快收敛，同时带动量的随机梯度下降算法有更好的最优解，即模型收敛后会有更高的准确性。通常若追求速度则用Adam更多。

（4）迭代次数

迭代次数是指整个训练集输入到神经网络进行训练的次数，当测试错误率和训练错误率相差较小时，可认为当前迭代次数合适；当测试错误率先变小后变大时则说明迭代次数过大了，需要减小迭代次数，否则容易出现过拟合。

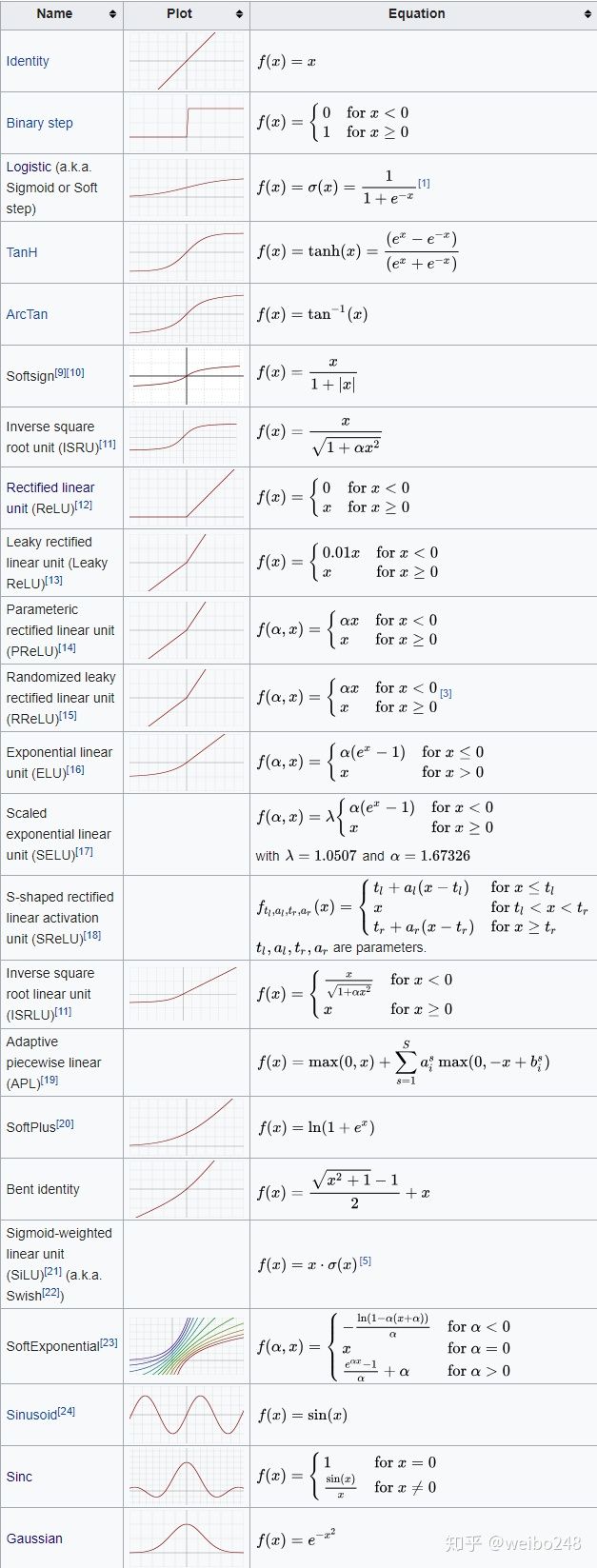
（5）激活函数

在神经网络中，激活函数不是真的去激活什么，而是用激活函数给神经网络加入一些非线性因素，使得网络可以更好地解决较为复杂的问题。比如有些问题是线性可分的，而现实场景中更多问题不是线性可分的，若不使用激活函数则难以拟合非线性问题，测试时会有低准确率。所以激活函数主要是非线性的，如sigmoid、tanh、relu。sigmoid函数通常用于二分类，但要防止梯度消失，故适合浅层神经网络且需要配备较小的初始化权重，tanh函数具有中心对称性，适合于有对称性的二分类。在深度学习中，relu是使用最多的激活函数，简单又避免了梯度消失。

常见超参数搜索算法：

* 网格搜索
* 随机搜索
* 启发式搜索

1. **常见的激活函数及图像? 常见激活函数的导数计算？**



1. **如何选择激活函数？**

现在我们已经了解了这么多的激活函数，接下来就需要分析在哪种情况下应该使用哪种激活函数了。激活函数好或坏，不能凭感觉定论。然而，根据问题的性质，我们可以为神经网络更快更方便地收敛作出更好的选择。

* 用于分类器时，Sigmoid函数及其组合通常效果更好。
* 由于梯度消失问题，有时要避免使用sigmoid和tanh函数。
* ReLU函数是一个通用的激活函数，目前在大多数情况下使用。
* 如果神经网络中出现死神经元，那么PReLU函数就是最好的选择。
* 请记住，ReLU函数只能在隐藏层中使用。
* 一点经验：你可以从ReLU函数开始，如果ReLU函数没有提供最优结果，再尝试其他激活函数。

1. **使用ReLu激活函数的优点？**

修正线性单元(Rectified linear unit，ReLU）

ReLU有以下优势：

* 对于线性函数而言，ReLU的表达能力更强，尤其体现在深度网络中；
* 而对于非线性函数而言，ReLU由于非负区间的梯度为常数，因此不存在梯度消失问题(Vanishing Gradient Problem)，使得模型的收敛速度维持在一个稳定状态。

1. **Softmax函数如何应用于多分类?**

Sigmoid + cross-entropy follows the Bernoulli distribution, while softmax + log-likelihood follows the multinomial distribution with one observation (which is a multiclass version of the Bernoulli).

For binary classification problems, the softmax function outputs two values (between 0 and 1 and sum up to 1), to represent the probabilities of each class.

While the sigmoid function outputs one value between 0 and 1, to represent the probability of one class (so the probability of the other class is just 1-p).

Sigmoid+互信息输出结果是伯努利分布（注：P(y\_1|X), P(y\_2|X),...,P(y\_n|X)）

而Softmax输出的是多项分布（注：P(y\_1, y\_2,...,y\_n|X)）

对于二值分类问题，Softmax输出两个值，这两个值相加为1

对于Sigmoid来说，也输出两个值，不过没有可加性，两个值各自是0到1的某个数，对于一个值p来说，1-p是它对应的另一个概率。

例如：如果我们预测某个东西是或者不是，那么我们可以这样：

输出(0, 1)代表“是”，输出(1, 0)代表“否”

Softmax可能输出(0.3, 0.7)，代表算法认为“是”的概率是0.7，“否”的概率是0.3，相加为1

Sigmoid的输出可能是(0.4, 0.8)，它们相加不为1，解释来说就是Sigmoid认为输出第一位为1的概率是0.4，第一位不为1的概率是0.6（1-p），第二位为1的概率是0.8，第二位不为1的概率是0.2。

Geoff Hinton covered exactly this topic in his coursera course on neural nets. The problem with sigmoids is that as you reach saturation (values get close to 1 or 0), the gradients vanish. This is detrimental to optimization speed. Softmax doesn't have this problem, and in fact if you combine softmax with a cross entropy error function the gradients are just (z-y), as they would be for a linear output with least squares error.

1. **Batch\_Size**

Batch\_Size（批尺寸）是机器学习中一个重要参数

批梯度下降法（Mini-batches Learning）。因为如果数据集足够充分，那么用一半（甚至少得多）的数据训练算出来的梯度与用全部数据训练出来的梯度是几乎一样的。

* 在合理范围内，增大 Batch\_Size 有何好处？

内存利用率提高了，大矩阵乘法的并行化效率提高。

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。

在一定范围内，一般来说 Batch\_Size 越大，其确定的下降方向越准，引起训练震荡越小。

* 盲目增大 Batch\_Size 有何坏处？

内存利用率提高了，但是内存容量可能撑不住了。

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。

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。

* 运行结果与上文分析相印证：

Batch\_Size 太小，算法在 200 epoches 内不收敛。

随着 Batch\_Size 增大，处理相同数据量的速度越快。

随着 Batch\_Size 增大，达到相同精度所需要的 epoch 数量越来越多。

由于上述两种因素的矛盾， Batch\_Size 增大到某个时候，达到时间上的最优。

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。

既然 Full Batch Learning 并不适用大数据集，那么走向另一个极端怎么样？

所谓另一个极端，就是每次只训练一个样本，即 Batch\_Size = 1。这就是在线学习（Online Learning）。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。使用在线学习，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

1. **归一化? 归一化含义？为什么要归一化? 为什么归一化能提高求解最优解速度？**

* 数据归一化/标准化（二者还是有所区别的）是机器学习/数据挖掘的一项基础工作，是数据预处理的重要一步。样本各个特征往往具有不同的分布/取值范围，通过归一化将各个维度的特征值映射到相同区间，使得各特征值具有相同量纲，处于同一数量级

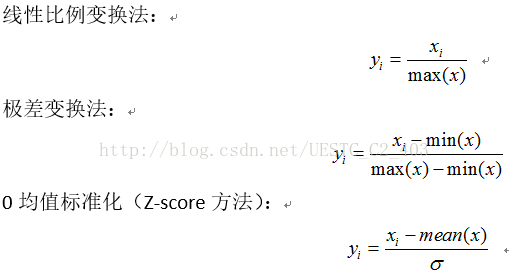
如果本来各维的量纲是相同的，最好不要做归一化，以尽可能多地保留信息。

如果本来各维的量纲是不同的，就需要先对各维分别归一化。

* 归一化是机器学习中必不可少的一步，只要样本各维量纲不一致，我们就需要对数据集进行归一化，不管训练什么机器学习模型，最好都先对数据集进行归一化，这是因为，归一化不仅可以提高模型正确率（针对不具有伸缩不变性的机器学习算法），还可以加快模型收敛，提升训练速度。

于是损失函数的等高线类似椭圆形状，最优解的寻优过程如下图所示，可以发现梯度下降方向只要不在椭圆对称轴上，这一步损失函数几乎就不会减小，这会导致震荡，使得收敛很慢，甚至不收敛。

此时损失函数等高线类似圆形形状，最优解的寻优过程如下图所示，可以发现归一化后，寻优过程变得平滑，收敛速度得到提高，加快模型训练速度。



1. **Weight Normalization和Batch Normalization**

Weight Normalization和Batch Normalization都属于参数重写（Re-parameterization）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：

* 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minbatch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，minibtach的均值和方差，效率低且占内存）。
* 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Trainingset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。

* 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。

normalization按大类分可以分三类：

一是对参数做归一化，二是对输出特征做归一化。归一化的手段可以多种多样，最基本的是instance级别的 l2 norm，也可以是batchnorm，也可以是其他的。bn是使用batchnorm手段的 特征归一化。目前是训练深度网络的必备利器ln是使用一种类似bn的方法对特征归一化，在rnn模型里效果较好wn是使用l2 norm的参数归一化此外，还有一种cosine norm的方法，同时对参数和特征做l2 norm归一化，似乎效果也不错最后，对参数做batch norm不知道会怎么样

1. **为什么无监督预训练可以帮助深度学习？**

最近深度学习框架中比如：Deep Belief Networks，Stacks of Auto-Encoder variants. 都基本基于先通过无监督的预训练，然后再通过有监督的fine tune，达到了最好的效果。

深度网络存在以下缺点：

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。（深层网络意味着特征比较多，机器学习里面临多特征：1.多样本 2.规则化 3.特征选择）

2. 多层神经网络参数优化是个高阶非凸优化问题，常收敛较差的局部解

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个网络参数的初始值。  无监督学习--->参数初始值；监督学习--->fine-tuning，即训练有标注样本。经过预训练最终能得到比较好的局部最优解。

我们为什么要做这种无监督预训练？反向传播(back-propgation)在多层次模型上到底有什么问题呢？其实反向传播算法的问题在于当我们对每一层做梯度计算(gradient computation)的时候，随着层次的深度，它的值会变得越来越小，我们称之为梯度消失问题(gradient vanishing problem). 所以，在一般情况下，当我们对深度模型的底层计算梯度的时候它的值已经变得非常小(或已接近于0)。类似的问题在训练RNN(Recurrent Neural Network)的时候也会存在，我们会在后面给予介绍。当然，也很容易想到，虽然梯度在底层上的变化很小，但如果有“足够多”的数据，我们也可以在无预训练的情况下训练出有效的模型。

1. **权重偏差初始化，全都初始化为0，全都初始化为同样的值，初始化为小的随机数**

1） 全部初始化为零；

但是将权重w全部初始化为零，那么每一层所学到的参数都是一样的，因为它们的梯度一样，所以在反向传播的过程中，每一层的神经元也是相同的。因此会导致代价函数在开始的一段时间内，明显下降，但是一段时间以后，停止继续下降

2） 初始化为相同的随机数

其实将权重w初始化为相同的随机数和全部初始化为零是一样的，都会导致同样的问题

3） 初始化为较小的随机数；

权重参数随机初始化为服从均值为零和方差为1的高斯分布函数，开始模型可以很好的运行一段时间，但是随着时间增加，前向传递时，方差开始减少，梯度也开始向零靠近，会导致Gradient Vanishing。特别地，当激活函数为sigmoid时，梯度接近0.5；当激活函数为时tanh，梯度接近0。

4） 初始化为较大的随机数

反向传播时，倒数趋于零，梯度也会消失。此外，权重较大且当输入也很大时，如果使用sigmoid做激活函数，会使输出趋向于0和1，会导致更多问题

1. **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

1. **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。

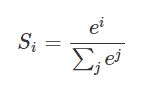
1. **初始化偏差**

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.

1. **Softmax, Softmax定义及作用, Softmax推导**

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，Vi表示V中的第i个元素，那么这个元素的softmax值就是



1. **理解One Hot Encodeing原理及作用？**

数据预处理之独热编码（One-Hot Encoding）

独热编码即 One-Hot 编码，又称一位有效编码，其方法是使用N位状态寄存器来对N个状态进行编码，每个状态都由他独立的寄存器位，并且在任意时候，其中只有一位有效

为什么使用one-hot编码来处理离散型特征?

* 使用one-hot编码，将离散特征的取值扩展到了欧式空间，离散特征的某个取值就对应欧式空间的某个点。
* 将离散特征通过one-hot编码映射到欧式空间，是因为，在回归，分类，聚类等机器学习算法中，特征之间距离的计算或相似度的计算是非常重要的，而我们常用的距离或相似度的计算都是在欧式空间的相似度计算，计算余弦相似性，基于的就是欧式空间。
* 将离散型特征使用one-hot编码，确实会让特征之间的距离计算更加合理。比如，有一个离散型特征，代表工作类型，该离散型特征，共有三个取值，不使用one-hot编码，其表示分别是x\_1 = (1), x\_2 = (2), x\_3 = (3)。两个工作之间的距离是，(x\_1, x\_2) = 1, d(x\_2, x\_3) = 1, d(x\_1, x\_3) = 2。那么x\_1和x\_3工作之间就越不相似吗？显然这样的表示，计算出来的特征的距离是不合理。那如果使用one-hot编码，则得到x\_1 = (1, 0, 0), x\_2 = (0, 1, 0), x\_3 = (0, 0, 1)，那么两个工作之间的距离就都是sqrt(2).即每两个工作之间的距离是一样的，显得更合理。
* 对离散型特征进行one-hot编码是为了让距离的计算显得更加合理
* 将离散型特征进行one-hot编码的作用，是为了让距离计算更合理，但如果特征是离散的，并且不用one-hot编码就可以很合理的计算出距离，那么就没必要进行one-hot编码，比如，该离散特征共有1000个取值，我们分成两组，分别是400和600,两个小组之间的距离有合适的定义，组内的距离也有合适的定义，那就没必要用one-hot 编码
* 离散特征进行one-hot编码后，编码后的特征，其实每一维度的特征都可以看做是连续的特征。就可以跟对连续型特征的归一化方法一样，对每一维特征进行归一化。比如归一化到[-1,1]或归一化到均值为0,方差为1

1. **引申：如何做归一化**

处理离散型特征和连续型特征并存的情况，如何做归一化：

* 需要归一化情况：

拿到获取的原始特征，必须对每一特征分别进行归一化，比如，特征A的取值范围是[-1000,1000]，特征B的取值范围是[-1,1].

如果使用logistic回归，w1\*x1+w2\*x2，因为x1的取值太大了，所以x2基本起不了作用。

所以，必须进行特征的归一化，每个特征都单独进行归一化。

* 连续型特征归一化的常用方法：

Rescale bounded continuous features: All continuous input that are bounded, rescale them to [-1, 1] through x = (2x - max - min)/(max - min).线性放缩到[-1,1]

Standardize all continuous features: All continuous input should be standardized and by this I mean, for every continuous feature, compute its mean (u) and standard deviation (s) and do x = (x - u)/s.放缩到均值为0，方差为1

有些情况不需要进行特征的归一化：

基于树的方法是不需要进行特征的归一化，例如随机森林，bagging 和 boosting等。

基于参数的模型或基于距离的模型，都是要进行特征的归一化。

1. **常用的优化器有哪些 ?**

首先来看一下梯度下降最常见的三种变形 BGD，SGD，MBGD，这三种形式的区别就是取决于我们用多少数据来计算目标函数的梯度，这样的话自然就涉及到一个 trade－off，即参数更新的准确率和运行时间

* SGD with Momentum

如果数据是稀疏的，就用自适用方法，即 Adagrad, Adadelta, RMSprop, Adam。

RMSprop, Adadelta, Adam 在很多情况下的效果是相似的。

Adam 就是在 RMSprop 的基础上加了 bias-correction 和 momentum，

随着梯度变的稀疏，Adam 比 RMSprop 效果会好。

整体来讲，Adam 是最好的选择。

很多论文里都会用 SGD，没有 momentum 等。SGD 虽然能达到极小值，但是比其它算法用的时间长，而且可能会被困在鞍点。

如果需要更快的收敛，或者是训练更深更复杂的神经网络，需要用一种自适应的算法。

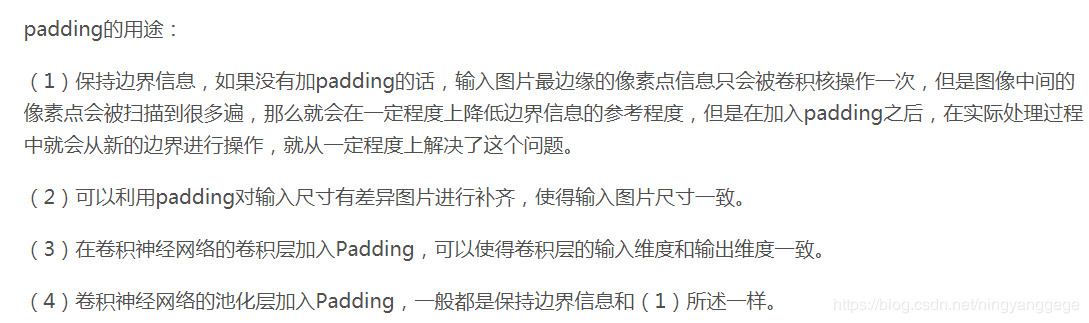
1. **Dropout 系列问题 , dropout率的选择**

数据量小的时候，dropout效果不好，数据量大了，dropout效果好

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。

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）

1. **Padding 系列问题**





1. **卷积神经网络的组成层**

卷积神经网络分成三个主要部分，分别为：卷积层（convolution layer），池化层（pooling layer） 和 全连接层 （fully connected layer）

1. **卷积层和池化层有什么区别？**

* 卷积层：用来进行特征的提取
* 池化层：对输入的特征图进行压缩，一方面使特征图变小，简化网络计算复杂度；一方面进行特征压缩，提取主要特征

在卷积神经网络中，我们经常会碰到池化操作，而池化层往往在卷积层后面，通过池化来降低卷积层输出的特征向量，同时改善结果（不易出现过拟合）

最常见的池化操作为平均池化mean pooling和最大池化max pooling：

平均池化：计算图像区域的平均值作为该区域池化后的值。

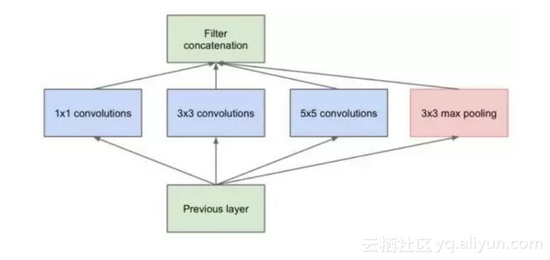
最大池化：选图像区域的最大值作为该区域池化后的值。

1. **卷积核一定越大越好？**

AlexNet中用到了一些非常大的卷积核，比如11×11、5×5卷积核，之前人们的观念是，卷积核越大，receptive field（感受野）越大，看到的图片信息越多，因此获得的特征越好。虽说如此，但是大的卷积核会导致计算量的暴增，不利于模型深度的增加，计算性能也会降低。于是在VGG（最早使用）、Inception网络中，利用2个3×3卷积核的组合比1个5×5卷积核的效果更佳，同时参数量（3×3×2+1 VS 5×5×1+1）被降低，因此后来3×3卷积核被广泛应用在各种模型中。

1. **每层卷积只能用一种尺寸的卷积核？ Inception结构**

传统的层叠式网络，基本上都是一个个卷积层的堆叠，每层只用一个尺寸的卷积核，例如VGG结构中使用了大量的3×3卷积层。事实上，同一层feature map可以分别使用多个不同尺寸的卷积核，以获得不同尺度的特征，再把这些特征结合起来，得到的特征往往比使用单一卷积核的要好，谷歌的GoogLeNet，或者说Inception系列的网络，就使用了多个卷积核的结构：



如上图所示，一个输入的feature map分别同时经过1×1、3×3、5×5的卷积核的处理，得出的特征再组合起来，获得更佳的特征。但这个结构会存在一个严重的问题：参数量比单个卷积核要多很多，如此庞大的计算量会使得模型效率低下。

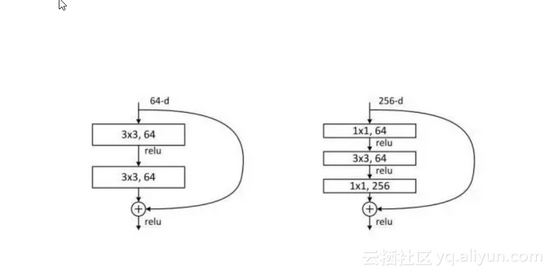
1. **怎样才能减少卷积层参数量？**

1×1卷积核也被认为是影响深远的操作，往后大型的网络为了降低参数量都会应用上1×1卷积核

256维的输入直接经过一个3×3×256的卷积层，输出一个256维的feature map，那么参数量为：256×3×3×256 = 589,824

256维的输入先经过一个1×1×64的卷积层，再经过一个3×3×64的卷积层，最后经过一个1×1×256的卷积层，输出256维，参数量为：256×1×1×64 + 64×3×3×64 + 64×1×1×256 = 69,632。足足把第一种操作的参数量降低到九分之一

1. **越深的网络就越难训练吗？ Resnet残差网络**



ResNet skip connection

传统的卷积层层叠网络会遇到一个问题，当层数加深时，网络的表现越来越差，很大程度上的原因是因为当层数加深时，梯度消散得越来越严重，以至于反向传播很难训练到浅层的网络。为了解决这个问题，何凯明大神想出了一个“残差网络”，使得梯度更容易地流动到浅层的网络当中去，而且这种“skip connection”能带来更多的好处，

1. **卷积操作时必须同时考虑通道和区域吗？**

Xception网络就是基于以上的问题发明而来。我们首先对每一个通道进行各自的卷积操作，有多少个通道就有多少个过滤器。得到新的通道feature maps之后，这时再对这批新的通道feature maps进行标准的1×1跨通道卷积操作。这种操作被称为 “ DepthWise convolution”缩写“DW”。

这种操作是相当有效的，在imagenet 1000类分类任务中已经超过了InceptionV3的表现，而且也同时减少了大量的参数，我们来算一算，假设输入通道数为3，要求输出通道数为256，两种做法：

1.直接接一个3×3×256的卷积核，参数量为：3×3×3×256 = 6,912

2.DW操作，分两步完成，参数量为：3×3×3 + 3×1×1×256 = 795，又把参数量降低到九分之一

因此，一个depthwise操作比标准的卷积操作降低不少的参数量，同时论文中指出这个模型得到了更好的分类效果。

1. **窄卷积和宽卷积，采用宽卷积的好处有什么？**

矩阵的中部使用3x3的滤波器没有问题，在矩阵的边缘该怎么办呢？左上角的元素没有顶部和左侧相邻的元素，该如何滤波呢？解决的办法是采用补零法（zero-padding）。所有落在矩阵范围之外的元素值都默认为0。这样就可以对输入矩阵的每一个元素做滤波了，输出一个同样大小或是更大的矩阵。补零法又被称为是宽卷积，不使用补零的方法则被称为窄卷积。

当滤波器长度相对输入向量的长度较大时，你会发现宽卷积很有用，或者说很有必要。

在tensorflow的实现中，conv2d的padding=‘SAME',默认的就是宽卷积的方式，会进行补零。padding = ’VALID'就是窄卷积的方式。

但是存在两个缺点：

1.卷积后的矩阵越变越小（如果卷积层100层，每一层都缩小最终得到的将是很小的图片）

2.输入矩阵（左）边缘像素（绿阴影）只被计算过一次，而中间像素（红阴影）被卷积计算多次，意味着丢失图像角落信息。

为了解决这两个问题，就对输入图像进行padding，即填充像素

1. **如何得到卷积层输出的深度？**

卷积层输出的深度==卷积核的个数

1. **激活函数通常放在卷积神经网络的那个操作之后？**

全连接层之后i

model.add(keras.layers.Flatten())

model.add(keras.layers.Dense(500, activation='relu'))

model.add(keras.layers.Dense(10, activation='softmax'))

1. **如何理解最大池化层有几分缩小？**

主要会做降采样（downsampling）

池化操作一般由两种，一种是Ava Pooling ，一种max Pooling

同样地采用一个2\*2的filter,max pooling是在每一个区域中寻找最大值，这里的stride=2,最终在原特征图中提取主要特征得到右图。

（Avy pooling现在不怎么用了，方法是对每一个2\*2的区域元素求和，再除以4，得到主要特征），而一般的filter取2\*2,最大取3\*3,stride取2，压缩为原来的1/4.

注意：这里的pooling操作是特征图缩小，有可能影响网络的准确度，因此可以通过增加特征图的深度来弥补（这里的深度变为原来的2倍）

1. **不同卷积后图像大小计算？**

像素宽度：W（Width）

填充大小：P（Padding）

卷积核大小：K（Kernel-size）

步长大小：S（stride）

卷积后所得feature map尺寸大小计算公式如下：

W_{n+1}=\left (W_{n}+2*P-K \right )/S+1

补充：

1.Padding的作用用于解决图像边缘信息损失的问题；

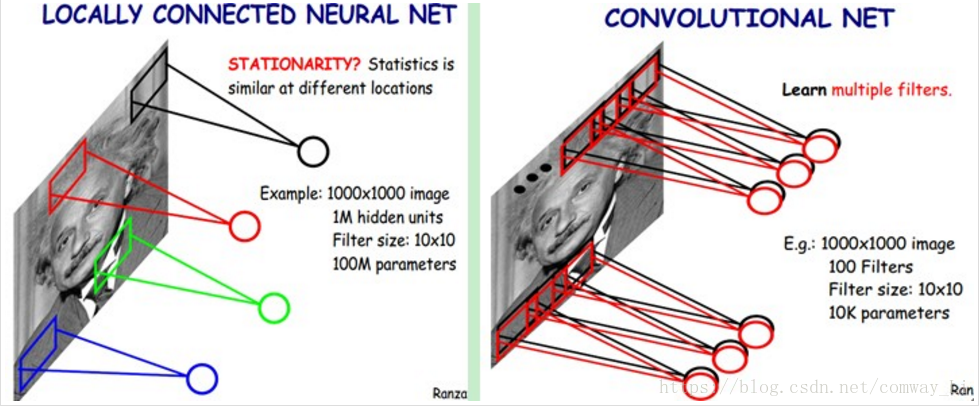
2.计算卷积后map尺寸时若不为整数则向下取整，而计算pooling后尺寸时则向上取整。

1. **局部连接 && 权值共享**

所谓的权值共享就是说，给一张输入图片，用一个filter去扫这张图, filter里面的数就叫权重，这张图每个位置是被同样的filter扫的，所以权重是一样的，也就是共享。

共享的角度来说：权重共享即filter的值共享

每个神经元仅与输入神经元的一块区域连接，这块局部区域称作感受野（receptive field）。在图像卷积操作中，即神经元在空间维度（spatial dimension，即上图示例H和W所在的平面）是局部连接。



1. **如何解决训练样本少的问题**

首先，数据量不够，你就把1个当8、9、10······个用，裁剪裁剪，变换变换，翻转翻转。

其次，学习特征不够，人工先验来凑。

人工增加训练集的大小. 通过平移, 翻转, 加噪声等方法从已有数据中创造出一批"新"的数据.

Regularization. 数据量比较小会导致模型过拟合, 使得训练误差很小而测试误差特别大. 通过在Loss Function 后面加上正则项可以抑制过拟合的产生. 缺点是引入了一个需要手动调整的hyper-parameter. 详见

这也是一种正则化手段. 不过跟以上不同的是它通过随机将部分神经元的输出置零来实现.

Unsupervised Pre-training. 用Auto-Encoder或者RBM的卷积形式

一层一层地做无监督预训练,最后加上分类层做有监督的Fine-Tuning.

1. **什么样的样本集不适合用深度学习?**

* 数据集太小，数据样本不足时，深度学习相对其它机器学习算法，没有明显优势；
* 数据集没有局部相关特性，目前深度学习表现比较好的领域主要是图像／语音／自然语言处理等领域，这些领域的一个共性是局部相关性。图像中像素组成物体，语音信号中音位组合成单词，文本数据中单词组合成句子，这些特征元素的组合一旦被打乱，表示的含义同时也被改变。对于没有这样的局部相关性的数据集，不适于使用深度学习算法进行处理。

1. **对所有优化问题来说,有没有可能找到比已知算法更好的算法?**

没有免费的午餐定理：

对于训练样本（黑点），不同的算法A/B在不同的测试样本（白点）中有不同的表现，这表示：对于一个学习算法A，若它在某些问题上比学习算法 B更好，则必然存在一些问题，在那里B比A好。 也就是说：对于所有问题，无论学习算法A多聪明，学习算法 B多笨拙，它们的期望性能相同。

但是：没有免费午餐定理假设所有问题出现几率相同，实际应用中，不同的场景，会有不同的问题分布，所以，在优化算法时，针对具体问题进行分析，是算法优化的核心所在

1. **何为共线性, 跟过拟合有啥关联?**

共线性：多变量线性回归中，变量之间由于存在高度相关关系而使回归估计不准确。

共线性会造成冗余，导致过拟合。

解决方法：排除变量的相关性／加入权重正则。

1. **广义线性模型是怎被应用在深度学习中?**

机器学习中常见的广义线性模型（GLM）：这种模型是把自变量的线性预测函数当作因变量的估计值。在机器学习中，有很多模型都是基于广义线性模型的，比如传统的线性回归模型，最大熵

A Statistical View of Deep Learning (I): Recursive GLMs

深度学习从统计学角度，可以看做递归的广义线性模型。

广义线性模型相对于经典的线性模型(y=wx+b)，核心在于引入了连接函数g(.)，形式变为：y=g−1(wx+b)。

深度学习时递归的广义线性模型，神经元的激活函数，即为广义线性模型的链接函数。逻辑回归（广义线性模型的一种）的Logistic函数即为神经元激活函数中的Sigmoid函数，很多类似的方法在统计学和神经网络中的名称不一样，容易引起初学者（这里主要指我）的困惑。

1. **造成梯度消失的原因?**

神经网络的训练中，通过改变神经元的权重，使网络的输出值尽可能逼近标签以降低误差值，训练普遍使用BP算法，核心思想是，计算出输出与标签间的损失函数值，然后计算其相对于每个神经元的梯度，进行权值的迭代。

梯度消失会造成权值更新缓慢，模型训练难度增加。造成梯度消失的一个原因是，许多激活函数将输出值挤压在很小的区间内，在激活函数两端较大范围的定义域内梯度为0。造成学习停止

1. **权值初始化方法有哪些**

xavier高斯初始化，Glorot正态分布初始化方法，也称作Xavier正态分布初始化，参数由0均值，标准差为sqrt(2 / (fan\_in + fan\_out))的正态分布产生，

权值初始化的方法主要有：常量初始化（constant）、高斯分布初始化（gaussian）、positive\_unitball初始化、均匀分布初始化（uniform）、xavier初始化、msra初始化、双线性初始化（bilinear）

1. **启发式优化算法中，如何避免陷入局部最优解？**

这些算法的性能有两部分组成，一部分是exploration，另外一部分是exploitation。也可以分别对应于global search和local search。我们都知道，local search的方法是比价欧容易陷入局部最优的，而global的方法是不会对局部最优敏感的。

启发式算法中，局部最优值的陷入无法避免。启发式，本质上是一种贪心策略，这也在客观上决定了不符合贪心规则的更好（或者最优）解会错过。简单来说，避免陷入局部最优就是两个字：随机。具体实现手段上，可以根据所采用的启发式框架来灵活地加入随机性。比如遗传里面，可以在交叉变异时，可以在控制人口策略中，也可以在选择父本母本样本时；禁忌里面，可以在禁忌表的长度上体现，也可以在解禁策略中使用，等等。这些都要结合具体问题特定的算例集，需要反复尝试摸索才行。参数的敏感性是一个问题，建议不要超过3个参数，参数越不敏感越好。不同算例集用不同种子运行多次（100次左右才有统计意义），统计平均性能即可。需注意全局的随机重启通常来说不是一个好办法，因为等于主动放弃之前搜索结果，万不得已不要用，或者就是不用。三个原则应该把握：越随机越好；越不随机越好；二者平衡最好。

1. **凸优化中如何改进GD方法以防止陷入局部最优解**

在对函数进行凸优化时，如果使用导数的方法（如：梯度下降法/GD，牛顿法等）来寻找最优解，有可能陷入到局部最优解而非全局最优解。

为了防止得到局部最优，可以对梯度下降法进行一些改进，防止陷入局部最优。

但是请注意，这些方法只能保证以最大的可能找到全局最优，无法保证100%得到全局最优。

* incremental GD/stochastic GD

在GD中，是需要遍历所有的点之后才计算w的变化的；但是，在stochastic GD中，每输入一个点，就根据该点计算下一步的w，这样，不仅可以从batch training变成online training方法，而且每次是按照单点的最优方向而不是整体的最优方向前进，从而相当于在朝目标前进的路上多拐了好多弯，有可能逃出局部最优。

* momentum方法

momentum相当与记忆住上一次的更新。在每次的更新中，都要加一个k倍的上一次更新量。这样，也不再是按照标准路线前进，每次的步骤都容易受到上一次的影响，从而可能会逃出局部最优。另外，也会加大步长，从而加快收敛。

1. **常见的损失函数？**

* 回归损失函数：

回归的损失函数比较少，常见的有MSE,MAE,RMSE,也比较好理解，就是预测值和真实值直接的差距最小。

分类损失函数：

分类损失函数的种类比较多，常见的分类算法，logistic regression, SVM, adaboost

* log损失函数（逻辑回归）
* 指数损失函数（adaboost）：
* hinge损失函数（SVM)
* 回归损失：均方误差/平方损失/L2 损失
* 平均绝对误差/L1 损失
* 交叉熵损失/负对数似然：这是分类问题中最常见的设置。随着预测概率偏离实际标签，交叉熵损失会逐渐增加。

1. **如何进行特征选择（feature selection）？**

评价准则分为五类：距离度量（Distance Measure）、信息增益度量（Information Gain Measure）、依赖性度量（Dependence Measure）、一致性度量（Consistency Measure）和分类器错误率度量（Classifier Error Rate Measure）。

（1）距离度量：距离度量一般认为是差异性或者分离性的度量，常用的距离度量方法有欧式距离等。对于一个二元分类问题，对于两个特征f1和f2，如果特征f1引起的两类条件概率差异大于特征f2，则认为特征f1优于特征f2。

（2）信息增益度量：特征f的信息增益定义为使用特征f的先验不确定性与期望的后验不确性之间的差异。若特征f1的信息增益大于特征f2的信息增益，则认为特征f1优于特征f2。

（3）依赖性度量：依赖性度量又称为相关性度量（Correlation Measure）、通常可采用皮尔逊相关系数（Pearson correlation coefficient）来计算特征f与类别C之间的相关度，若特征f1与类别C之间的相关性大于特征f2与类别C之间的相关性，则认为特征f1优于特征f2。同样也可以计算得到属性与属性之间的相关度，属性与属性之间的相关性越低越好。

（4）一致性度量：假定两个样本，若它们的特征值相同，且所属类别也相同，则认为它们是一致的：否则，则称它们不一致。一致性常用不一致率来衡量，其尝试找出与原始特征集具有一样辨别能力的最小的属性子集。

（5）分类器错误率度量：该度量使用学习器的性能作为最终的评价阈值。它倾向于选择那些在分类器上表现较好的子集。

以上5种度量方法中，距离度量（Distance Measure）、信息增益度量（Information Gain Measure）、依赖性度量（Dependence Measure）、一致性度量（Consistency Measure）常用于过滤式（filter）；分类器错误率度量(Classifier Error Rate Measure)则用于包裹式(wrapper）。

提到特征选择的动机首先要说下维灾难（the curse of dimensionality）,用个图(图片来自wiki)来形象的说明维灾难：

所谓的维灾难就是当特征维度超过一定界限后，分类器的性能随着特征维度的增加反而下降（而且维度越高训练模型的时间开销也会越大）。导致分类器下降的原因往往是因为这些高纬度特征中含有无关特征和冗余特征，因此特征选择的主要目的是去除特征中的无关特征和冗余特征：

1. **梯度消失/梯度爆炸原因，以及解决方法？**

神经网络的训练过程通常分为两个阶段：前向传播和反向传播。

* 预训练加微调

此方法来自Hinton在2006年发表的一篇论文，Hinton为了解决梯度的问题，提出采取无监督逐层训练方法，其基本思想是每次训练一层隐节点，训练时将上一层隐节点的输出作为输入，而本层隐节点的输出作为下一层隐节点的输入，此过程就是逐层“预训练”（pre-training）；在预训练完成后，再对整个网络进行“微调”（fine-tunning）。Hinton在训练深度信念网络（Deep Belief Networks中，使用了这个方法，在各层预训练完成后，再利用BP算法对整个网络进行训练。此思想相当于是先寻找局部最优，然后整合起来寻找全局最优，此方法有一定的好处，但是目前应用的不是很多了）

* 梯度剪切、正则

梯度剪切这个方案主要是针对梯度爆炸提出的，其思想是设置一个梯度剪切阈值，然后更新梯度的时候，如果梯度超过这个阈值，那么就将其强制限制在这个范围之内。这可以防止梯度爆炸。

* 另外一种解决梯度爆炸的手段是采用权重正则化（weithts regularization）比较常见的是l1 l1l1正则，和l2 l2l2正则，在各个深度框架中都有相应的API可以使用正则化，比如在tensorflow tensorflowtensorflow中，若搭建网络的时候已经设置了正则化参数，则调用以下代码可以直接计算出正则损失：
* relu、leakrelu、elu等激活函数
* Batchnorm是深度学习发展以来提出的最重要的成果之一了，目前已经被广泛的应用到了各大网络中，具有加速网络收敛速度，提升训练稳定性的效果，Batchnorm本质上是解决反向传播过程中的梯度问题。batchnorm全名是batch normalization，简称BN，即批规范化，通过规范化操作将输出信号x规范化保证网络的稳定性。

batchnorm就是通过对每一层的输出规范为均值和方差一致的方法，消除了w ww带来的放大缩小的影响，进而解决梯度消失和爆炸的问题，或者可以理解为BN将输出从饱和区拉倒了非饱和区。

1. **深度学习为什么不用二阶优化**

* 时间复杂度：使用二阶方法通常需要直接计算或者近似估计 Hessian 矩阵，这部分的时间损耗使得其相比一阶方法在收敛速度上带来的优势完全被抵消；
* 考虑cost 和样本数量的关系。在相同计算时间（计算资源）下，使用少数样本估计梯度从而进行多次迭代，和使用全部样本进行单次迭代，哪一种效果更好？ 类似的，使用少数样本估计的梯度进行多次迭代，和使用少数样本估计的Hessian的L-BFGS方法， 哪个效果更好？ 实际中，在相同的时间内使用更多的样本信息所得的效果比使用较少样本更好。对二阶优化算法来说，同样的计算时间中，BFGS只能在一个mini-batch上迭代一次，而SGD能在n个mini-batch上进行迭代，后者取得的cost improvement 比前者更好。这时候，使用二阶方法是不值得的

二阶优化方法目前还不适用于深度学习训练中，主要存在问题是：

1. 最重要的问题是二阶方法的计算量大，训练较慢。

2. 求导不易，实现比SGD这类一阶方法复杂。

3. 另外其优点在深度学习中无法展现出来，主要是二阶方法能够更快地求得更高精度的解，这在浅层模型是有益的，但是在神经网络这类深层模型中对参数的精度要求不高，相反 相对而言不高的精度对模型还有益处，能够提高模型的泛化能力。 当然，二阶优化方法也有优点，在凸优化中，训练较SGD这类方法更为稳定更为平滑，不用调参

1. **怎样优化你的深度学习系统？**

这里有一些方法，可以使用预训练模型来减少你的拟合时间和提高你的准确性：

1.研究理想的预训练结构：了解迁移学习的好处，或浏览一些强大的CNN架构。考虑那些看起来不太合适，但是共享特征的领域。

2.使用较小的学习率：因为预训练的权重通常比随机初始化的权重要好！你在这里的选择取决于学习环境和预训练的进展情况，在不同的epochs上检测误差，了解你离收敛有多近。

3.使用dropout：与Ridge和LASSO正则化回归模型一样，没有一个最优的α适合所有的模型，它是一个超级参数，取决于你的具体问题，必须进行测试。从更大的变化开始，就和上面的学习率一样。

4.限制权值大小：我们可以限制某些层的权值的最大范数(绝对值)，以泛化我们的模型。

5.不要动第一层：神经网络的第一个隐藏层倾向于捕捉通用的和可解释的特征，如形状、曲线或交互，这些特征通常与领域相关。我们通常最好是别动这些，重点优化其他的层。这可能意味着添加隐藏层，所以先着急。

6.修改输出层：用一个新的激活函数和适合你的领域的输出大小替换模型默认值。然而，不要把自己局限于最明显的解决方案上。虽然MNIST可能看起来想要10个输出类，但是一些数字有共同的变化，12-16个类可能会更好地解决这些变化，并提高模型性能！正如上面的提示一样，越接近输出，深度学习模型应该越来越多地进行修改和定制。

1. **为什么要设置单一数字评估指标？**

单一评估指标：就是指当遇到上面的这种情况时，我们只需要从查准率和查全率中选择一个进行评价即可，可以根据不同指标的不同特性来选择适合于自己系统的指标。如果想同时兼顾查准率和查全率时，可以使用F1 score。通过上表可以发现，通过F1 score我们可以很快就挑选出分类器A要优于分类器B。

1. **满足和优化指标（Satisficing and optimizing metrics**）

假设你创建一个硬件设备，这个设备通过麦克风监听用户说话，当发现说特定的“唤醒词（wakeword）”，则会唤醒系统。比如，亚马逊的Echo监听“Alexa”，苹果的Siri监听“Hey Siri”，Android监听“Okay Google”，百度的应用监听“Hello Baidu”。你会关心假正例的比率（the false positive rate）——没有说唤醒词的情况下系统被唤醒的频率，同时也会关心假反例的比率（the false negative rate）——当某个人说了唤醒词它唤醒失败的频率。这样一个系统的合理性能应该是在每24个小时的操作不超过一个假正例的情况下（满足指标），最小化假反例比率（优化指标）。

1. **怎样划分训练/开发/测试集 ？**

* 一般需要将样本分成独立的三部分训练集（train set），开发集（develop set）和测试集（test set）。其中训练集用来估计模型，开发集用来确定网络结构或者控制模型复杂程度的参数，而测试集则检验最终选择最优的模型的性能如何。一个典型的划分是训练集占总样本的50％，而其它各占25％，三部分都是从样本中随机抽取。
* 在大数据时代的机器学习或者深度学习领域中，如果还是按照传统的数据划分方式不是十分合理，因为测试集和验证集用于评估模型和选择模型，所需要的数据量和传统的数据量差不多，但是由于收集到的数据远远大于传统机器学习时代的数据量，所以占的比例也就要缩小。比如我们拥有1000000，这么多的数据，训练集：验证集：测试集=98:1:1。如果是两类，也就是相同的道理。如果你的训练和测试数据来自相同的分布1.打乱并将您的数据分成训练/开发/测试集
* Ng建议训练/开发/测试拆分约70％/ 15％/ 15％。

1. **设置评估指标的意义？**

评估指标

评估指标很多，我们应该选择一个能跟业务指标波动一致的评估指标，这样通过观察评估指标就能判断模型效果，可以大大提高模型迭代效率。

评估指标用于反应模型效果，在预测问题中，要评估模型的效果，就需要将模型预测结果 f(x) ​和真真实标注 Y​ 进行比较，评估指标定义为f(x)和Y​的函数。

score = metric(f(x),Y)

* 精确率(Precision）是指在所有系统判定的“真”的样本中，确实是真的的占比，就是TP/(TP+FP)。
* 召回率（Recall）是指在所有确实为真的样本中，被判为的“真”的占比，就是TP/(TP+FN)。
* TPR（True Positive Rate）的定义，跟Recall一样。
* FPR（False Positive Rate），又被称为“Probability of False Alarm”，就是所有确实为“假”的样本中，被误判真的样本，或者FP/(FP+TN)
* F1值是为了综合考量精确率和召回率而设计的一个指标，一般公式为取P和R的harmonic mean:2PrecisionRecall/(Precision+Recall)。
* ROC=Receiver Operating Characteristic，是TPR vs FPR的曲线；与之对应的是Precision-Recall Curve，展示的是Precision vs Recall的曲线。

1. **什么是可避免偏差？**

Avoidable bias (可避免偏差)

可避免偏差这个词说明了有一些别的偏差，或者错误率有个无法超越的最低水平，那就是说如果贝叶斯错误率是 7.5%。你实际上并不想得到低于该级别的错误率，所以你不会说你的训练错误率是 8%，然后 8%就衡量了例子中的偏差大小。你应该说，可避免偏差可能在0.5%左右，或者 0.5%是可避免偏差的指标。而这个 2%是方差的指标，所以要减少这个 2%比减少这个 0.5%空间要大得多。而在左边的例子中，这 7%衡量了可避免偏差大小，而 2%衡量了方差大小。所以在左边这个例子里，专注减少可避免偏差可能潜力更大。

对于某些任务如计算机视觉上，人类能够做到的水平和贝叶斯误差相差不远。（这里贝叶斯误差指最好的分类器的分类误差，也就是说没有分类器可以做到100% 正确）。这里将人类水平误差近似为贝叶斯误差。

左边的例子：8% 与 1% 差距较大

主要专注于减少偏差，即减少训练集误差和人类水平误差之间的差距，来提高模型性能。

右边的例子：8% 与 7.5% 接近

主要专注于减少方差，即减少开发集误差和测试集误差之间的差距，来提高模型性能。

1. **什么是TOP5错误率？**

imagenet图像通常有1000个可能的类别，对每幅图像你可以猜5次结果(即同时预测5个类别标签)，当其中有任何一次预测对了，结果都算对，当5次全都错了的时候，才算预测错误，这时候的分类错误率就叫top5错误率.

Top-1 error 的意思是：假如模型预测某张动物图片（一只猫）的类别，且模型只输出1个预测结果，那么这一个结果正好能猜出来这个动物是只猫的概率就是Top-1正确率。猜出来的结果不是猫的概率则成为Top-1错误率。简单来说就是模型猜错的概率。Top-5 error 的意思是：假如模型预测某张动物图片（还是刚才那只猫），但模型会输出来5个预测结果，那么这五个结果中有猫这个分类的概率成为Top-5正确率，相反，预测输出的这五个结果里没有猫这个分类的概率则成为Top-5错误率。一般来说，Top-1和Top-5错误率越低，模型的性能也就越好。且Top-5 error 在数值上会比Top-1 error 的数值要小，毕竟从1个结果猜对的几率总会比从5个结果里猜对的几率要小嘛！

1. **可避免偏差、几大错误率之间的关系？**

13.28 怎样选取可避免偏差及贝叶斯错误率？

13.29 怎样减少方差？

13.30 贝叶斯错误率的最佳估计

13.31 举机器学习超过单个人类表现几个例子？

13.32 如何改善你的模型？

13.33 理解误差分析

13.34 为什么值得花时间查看错误标记数据？

13.35 快速搭建初始系统的意义？

13.36 为什么要在不同的划分上训练及测试？

13.37 如何解决数据不匹配问题？

13.38 梯度检验注意事项？

13.39 什么是随机梯度下降？

13.40 什么是批量梯度下降？

13.41 什么是小批量梯度下降？

13.42 怎么配置mini-batch梯度下降

13.43 局部最优的问题

13.44 提升算法性能思路

1. **有哪些超参数，为超参数选择合适的范围，如何搜索超参数？**

超参数是我们控制我们模型结构、功能、效率等的 调节旋钮，具体有哪些呢：

* learning rate
* epochs(迭代次数，也可称为 num of iterations)
* num of hidden layers(隐层数目)
* num of hidden layer units(隐层的单元数/神经元数)
* activation function(激活函数)
* batch-size(用mini-batch SGD的时候每个批量的大小)
* optimizer(选择什么优化器，如SGD、RMSProp、Adam)
* 用诸如RMSProp、Adam优化器的时候涉及到的β1，β2等等

人工的超参数选择过程，我们也可以采取类似参数搜索的办法，来提高效率，如果进行人工试错的方式，会非常浪费时间。

* 超参数搜索过程：
* 将数据集分为训练集，验证集及测试集。
* 选择模型性能评价指标
* 用训练集对模型进行训练
* 在验证集上对模型进行参数进行搜索，用性能指标评价参数好坏
* 选出最优参数

常见超参数搜索算法：

* 网格搜索
* 随机搜索
* 启发式搜索

网格搜索：

网格搜索是在所有候选的参数选择中，通过循环遍历，尝试每一种可能性，表现最好的参数就是最终的结果（暴力搜索）。

随机搜索：

随机搜索（random search）是利用随机数去求函数近似的最优解的方法，区别于网格搜索的暴力搜索方式。

原理：在一定的区间内，不断随机地而不是有倾向性产生随机点，并计算其约束函数和目标函数的值，对满足约束条件的点，逐个比较其目标函数的值，将坏的点抛弃，保留好的点，最后便得到最优解的近似解。

启发式搜索：

启发式搜索(Heuristically Search)又称为有信息搜索(Informed Search)，它是利用问题拥有的启发信息来引导搜索，达到减少搜索范围、降低问题复杂度的目的，这种利用启发信息的搜索过程称为启发式搜索。

原理：在状态空间中的搜索对每一个搜索的位置进行评估，得到最好的位置，再从这个位置进行搜索直到目标。这样可以省略大量无谓的搜索路径，提高了效率。在启发式搜索中，对位置的估价是十分重要的。采用了不同的估价可以有不同的效果。

启发式搜索有模拟退火算法(SA)、遗传算法(GA)、列表搜索算法(ST)、进化规划(EP)、进化策略(ES)、蚁群算法(ACA)、人工神经网络(ANN)...等。

1. **What’s the trade-off between bias and variance?**

* Bias 是由于你使用的学习算法过度简单地拟合结果或者错误地拟合结果导致的错误。它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，即算法本身的拟合能力。Bias 可能会导致模型欠拟合，使其难以具有较高的预测准确性，也很难将你的知识从训练集推广到测试集。
* Variance 是由于你使用的学习算法过于复杂而产生的错误。它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。反应预测的波动情况。Variance 过高会导致算法对训练数据的高纬度变化过于敏感，这样会导致模型过度拟合数据。从而你的模型会从训练集里带来太多噪音，这会对测试数据有一定的好处。
* Bias-Variance 的分解，本质上是通过在基础数据集中添加偏差、方差和一点由噪声引起的不可约误差，来分解算法上的学习误差。从本质上讲，如果你使模型更复杂并添加更多变量，你将会失去一些 Bias 但获得一些 Variance，这就是我们所说的权衡（tradeoff）。这也是为什么我们在建模的过程中，不希望这个模型同时拥有高的偏差和方差。

1. **How is KNN different from k-means clustering?**

问题3: KNN和 k-means 聚类由什么不同？

K-Nearest Neighbors是一种监督分类算法，而 k-means聚类是一种无监督的聚类算法。 虽然这些机制起初可能看起来相似，但这实际上意味着为了使K-Nearest Neighbors工作，你需要标记数据，以便将未标记的点分类（因此是最近邻居部分）。 K均值聚类仅需要一组未标记的点和阈值：算法将采用未标记的点并逐渐学习如何通过计算不同点之间的距离的平均值将它们聚类成组。

这里的关键区别在于，KNN需要标记点，因此是有监督的学习，而k-means不是，因此是无监督学习。

1. **ROC**

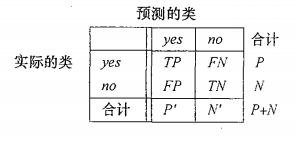
ROC曲线的作用：

1.较容易地查出任意界限值时的对类别的识别能力

2.选择最佳的诊断界限值。ROC曲线越靠近左上角,试验的准确性就越高。最靠近左上角的ROC曲线的点是错误最少的最好阈值，其假阳性和假阴性的总数最少。

3.两种或两种以上不同诊断试验对算法性能的比较。在对同一种算法的两种或两种以上诊断方法进行比较时，可将各试验的ROC曲线绘制到同一坐标中，以直观地鉴别优劣，靠近左上角的ROC曲线所代表的受试者工作最准确。亦可通过分别计算各个试验的ROC曲线下的面积(AUC)进行比较，哪一种试验的AUC最大，则哪一种试验的诊断价值最佳。

ROC曲线是根据一系列不同的二分类方式（分界值或决定阈），以真阳性率TPR（灵敏度）为纵坐标，假阳性率FPR（1-特异度）为横坐标绘制的曲线。



TPR = TP/P 即召回率公式

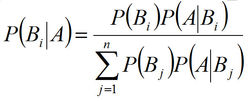
FPR = FP/N 即1-specificity

ROC曲线是以FPR为横坐标，以TPR为纵坐标，以概率为阈值来度量模型正确识别正实例的比例与模型错误的把负实例识别成正实例的比例之间的权衡，TPR的增加必定以FPR的增加为代价，ROC曲线下方的面积是模型准确率的度量所以根据ROC曲线定义可知，绘制ROC要求模型必须能返回监测元组的类预测概率，根据概率对元组排序和定秩，并使正概率较大的在顶部，负概率较大的在底部进行画图

1. **What is Bayes’ Theorem? How is it useful in a machine learning context?**

什么是贝叶斯定理？它在机器学习环境中如何有用?

贝叶斯定理描述了当你不能准确知悉一个事物的本质时，你可以依靠与事物特定本质相关的事件出现的多少去判断其本质属性的概率。 它给出了已知先验知识下事件的后验概率。



1. **Explain the difference between L1 and L2 regularization.**

问题8：L1、L2正则之间有什么不同？

* L2正则，对应的是加入2范数，使得对权重进行衰减，从而达到惩罚损失函数的目的，防止模型过拟合。保留显著减小损失函数方向上的权重，而对于那些对函数值影响不大的权重使其衰减接近于0。相当于加入一个gaussian prior。
* L1正则 对应得失加入1范数，同样可以防止过拟合。它会产生更稀疏的解，即会使得部分权重变为0，达到特征选择的效果。

一般回归分析中回归w表示特征的系数，从上式可以看到正则化项是对系数做了处理（限制）。L1正则化和L2正则化的说明如下：

* L1正则化是指权值向量w中各个元素的绝对值之和，通常表示为||w||\_1
* L2正则化是指权值向量w中各个元素的平方和然后再求平方根（可以看到Ridge回归的L2正则化项有平方符号），通常表示为||w||\_2​

相同点：都用于避免过拟合

不同点：L1可以让一部分特征的系数缩小到0，从而间接实现特征选择。所以L1适用于特征之间有关联的情况。

L2让所有特征的系数都缩小，但是不会减为0，它会使优化求解稳定快速。所以L2适用于特征之间没有关联的情况

1. **第一类误差和第二类误差有什么区别？**

第一类误差指的是假正率，第二类指的是假负率。简单来说，第一类误差意味着假设为真的情况下，作出了拒绝原假设的一种错误推断。第二类误差意味着假设为假的情况下，做出了接受原假设的一种错误判断。

举个例子：第一类误差，你误判一个男的他怀孕了。第二类误差，你误判了一位其实已经怀孕的女子没怀孕。

1. **概率和似然有什么区别？**

| **符号** | **含义** |
| --- | --- |
| O | 观测值 |
| θ | 随机过程中的参数 |
| θ^ | 参数的估计 |
| P(O|θ) | 概率 |
| L(θ|O) | 释然 |

“概率”描述了给定模型参数后，描述结果的合理性，而不涉及任何观察到的数据。

抛一枚均匀的硬币，拋20次，问15次拋得正面的可能性有多大？ 这里的可能性就是”概率”，均匀的硬币就是给定参数θ=0.5，“拋20次15次正面”是观测值O。求概率P(H=15|θ=0.5)=？的概率。

“似然”描述了给定了特定观测值后，描述模型参数是否合理。

拋一枚硬币，拋20次，结果15次正面向上，问其为均匀的可能性？ 这里的可能性就是”似然”，“拋20次15次正面”为观测值O为已知，参数θ=?并不知道，求L(θ|H=15)=P(H=15|θ=0.5)的最大化下的θ 值。

1. **如何对决策树进行剪枝？**

判断的标准就是看划分前后的泛华性能是否有提升，也就是如果划分后泛华性能有提升，则划分；否则，不划分。

对比预剪枝和后剪枝，能够发现，后剪枝决策树通常比预剪枝决策树保留了更多的分支，一般情形下，后剪枝决策树的欠拟合风险小，泛华性能往往也要优于预剪枝决策树。但后剪枝过程是在构建完全决策树之后进行的，并且要自底向上的对树中的所有非叶结点进行逐一考察，因此其训练时间开销要比未剪枝决策树和预剪枝决策树都大得多。

剪枝是在决策树中，为了降低模型的复杂度，提高决策树模型的预测精度，去除预测能力较弱的分支后所发生的现象。修剪可以自下而上和自上而下进行，方法包括减少错误修剪和成本复杂度修剪。

减少错误修剪可能是最简单的版本：替换每个节点。如果不降低预测精度，则保持修剪。虽然很简单，但这种启发式方法实际上非常接近于一种可以最大限度地优化准确性的方法。

1. **什么是F1数，怎么使用它？**

F1分数是衡量模型性能的指标。它是模型精度和召回的加权平均值，结果趋向于1是最好的，结果趋向于0是最差的。你可以在分类测试中使用它，而真正的否定并不重要。

1. **逻辑回归**

逻辑回归（Logistic Regression）是一种用于解决二分类（0 or 1）问题的机器学习方法，用于估计某种事物的可能性。比如某用户购买某商品的可能性，某病人患有某种疾病的可能性，以及某广告被用户点击的可能性等。 注意，这里用的是“可能性”，而非数学上的“概率”，logisitc回归的结果并非数学定义中的概率值，不可以直接当做概率值来用。该结果往往用于和其他特征值加权求和，而非直接相乘。

那么逻辑回归与线性回归是什么关系呢？

逻辑回归（Logistic Regression）与线性回归（Linear Regression）都是一种广义线性模型（generalized linear model）。逻辑回归假设因变量 y 服从伯努利分布，而线性回归假设因变量 y 服从高斯分布。 因此与线性回归有很多相同之处，去除Sigmoid映射函数的话，逻辑回归算法就是一个线性回归。可以说，逻辑回归是以线性回归为理论支持的，但是逻辑回归通过Sigmoid函数引入了非线性因素，因此可以轻松处理0/1分类问题。

从上图可以看到sigmoid函数是一个s形的曲线，它的取值在[0, 1]之间，在远离0的地方函数的值会很快接近0或者1。它的这个特性对于解决二分类问题十分重要

在线性回归中，最常用的是均方误差(Mean squared error)，

在逻辑回归中，最常用的是代价函数是交叉熵(Cross Entropy)，交叉熵是一个常见的代价函数，在神经网络中也会用到。

ROC曲线绘制采用不同分类阈值的TPR和FPR，降低分类阈值会将更多的样本判为正类别，从而增加FP和TP的个数。为了绘制ROC曲线，需要使用不同的分类阈值多次评估回归模型，很麻烦。有一种基于排序的高效算法可以为我们提供此类信息，这种算法称为曲线下的面积(AUV,area under roc curve)。

详解ROC和AUC

ROC曲线的横轴为FPR，越低越好，纵轴为TPR，越高越好，故如果有两个不同的模型，曲线位于左上方的模型优于曲线位于右下方的模型，这一点可以拿曲线的面积(AUV)来量化。完美的分类为TPR=1，FPR=0；ROC曲线过(0,0)和(1,1)点

AUC = 1，是完美分类器，采用这个预测模型时，不管设定什么阈值都能得出完美预测。绝大多数预测的场合，不存在完美分类器。

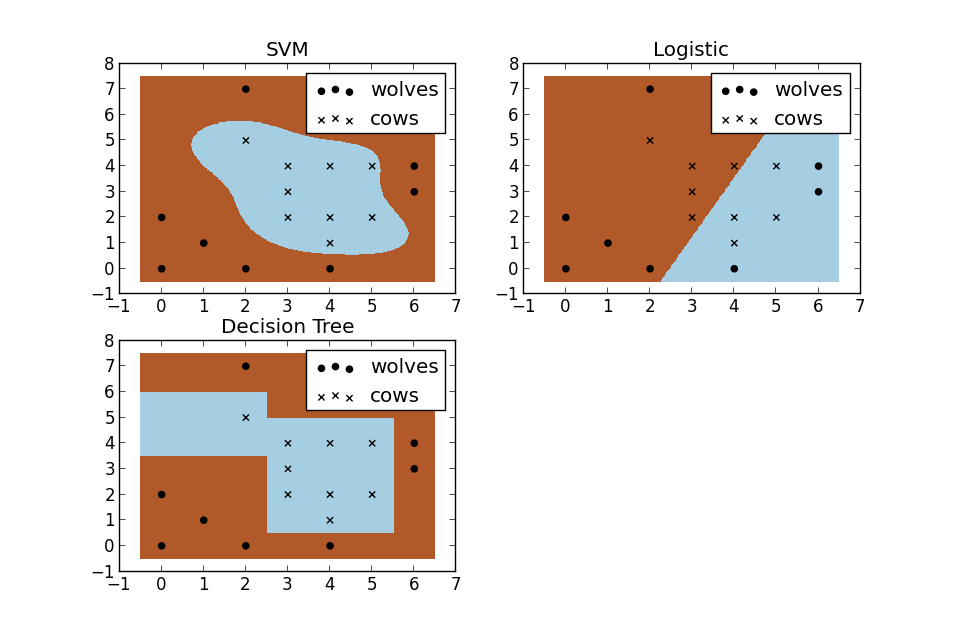
0.5 < AUC < 1，优于随机猜测。这个分类器（模型）妥善设定阈值的话，能有预测价值。

AUC = 0.5，跟随机猜测一样（例：丢铜板），模型没有预测价值。

AUC < 0.5，比随机猜测还差；但只要总是反预测而行，就优于随机猜测。

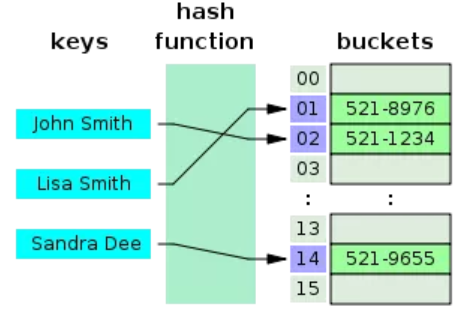
1. **什么是核技巧，有什么用处？**

核技巧使用核函数，确保在高维空间不需要明确计算点的坐标，而是计算数据的特征空间中的内积。这使其具有一个很有用的属性：更容易的计算高维空间中点的坐标。许多算法都可以表示称这样的内积形式，使用核技巧可以保证低维数据在高维空间中运用算法进行计算。



1. **描述哈希表**

哈希表是一种产生关联数组的数据结构。 通过使用散列函数将键映射到某些值。 它们通常用于数据库索引等任务



1. **请简要介绍下SVM**

SVM，全称是support vector machine，中文名叫支持向量机。SVM是一个面向数据的分类算法，它的目标是为确定一个分类超平面，从而将不同的数据分隔开。

它是一种二类分类模型，其基本模型定义为特征空间上的间隔最大的线性分类器，其学习策略便是间隔最大化，最终可转化为一个凸二次规划问题的求解。

1. **在k-means或kNN，我们常用欧氏距离来计算最近的邻居之间的距离，有时也用曼哈顿距离，请对比下这两种距离的差别**。

https://img-my.csdn.net/uploads/201211/20/1353398777_7638.png

欧氏距离，最常见的两点之间或多点之间的距离表示法，又称之为欧几里得度量

欧氏距离虽然很有用，但也有明显的缺点。它将样品的不同属性（即各指标或各变量量纲）之间的差别等同看待，这一点有时不能满足实际要求。

曼哈顿距离，我们可以定义曼哈顿距离的正式意义为L1-距离或城市区块距离，也就是在欧几里得空间的固定直角坐标系上两点所形成的线段对轴产生的投影的距离总和。例如在平面上，坐标（x1, y1）的点P1与坐标（x2, y2）的点P2的曼哈顿距离为：https://img-my.csdn.net/uploads/201211/20/1353398955_7627.png，要注意的是，曼哈顿距离依赖座标系统的转度，而非系统在座标轴上的平移或映射。当坐标轴变动时，点间的距离就会不同。

通俗来讲，想象你在曼哈顿要从一个十字路口开车到另外一个十字路口，驾驶距离是两点间的直线距离吗？显然不是，除非你能穿越大楼。而实际驾驶距离就是这个“曼哈顿距离”，这也是曼哈顿距离名称的来源， 同时，曼哈顿距离也称为城市街区距离(City Block distance)。

1. **CNN的卷积核是单层的还是多层的？**

一般而言，深度卷积网络是一层又一层的。层的本质是特征图, 存贮输入数据或其中间表示值。一组卷积核则是联系前后两层的网络参数表达体, 训练的目标就是每个卷积核的权重参数组。

描述网络模型中某层的厚度，通常用名词通道channel数或者特征图feature map数。不过人们更习惯把作为数据输入的前层的厚度称之为通道数（比如RGB三色图层称为输入通道数为3），把作为卷积输出的后层的厚度称之为特征图数。

卷积核(filter)一般是3D多层的，除了面积参数, 比如3x3之外, 还有厚度参数H（2D的视为厚度1). 还有一个属性是卷积核的个数N。

卷积核的厚度H, 一般等于前层厚度M(输入通道数或feature map数). 特殊情况M > H。

卷积核的个数N, 一般等于后层厚度(后层feature maps数，因为相等所以也用N表示)。

卷积核通常从属于后层，为后层提供了各种查看前层特征的视角，这个视角是自动形成的。

卷积核厚度等于1时为2D卷积，也就是平面对应点分别相乘然后把结果加起来，相当于点积运算.

卷积的意思就是把一个区域，不管是一维线段，二维方阵，还是三维长方块，全部按照卷积核的维度形状，从输入挖出同样维度形状, 对应逐点相乘后求和，浓缩成一个标量值也就是降到零维度，作为输出到一个特征图的一个点的值. 这个很像渔夫收网。

可以比喻一群渔夫坐一个渔船撒网打鱼，鱼塘是多层水域，每层鱼儿不同。

船每次移位一个stride到一个地方，每个渔夫撒一网，得到收获，然后换一个距离stride再撒，如此重复直到遍历鱼塘。

A渔夫盯着鱼的品种，遍历鱼塘后该渔夫描绘了鱼塘的鱼品种分布；

B渔夫盯着鱼的重量，遍历鱼塘后该渔夫描绘了鱼塘的鱼重量分布；

还有N-2个渔夫，各自兴趣各干各的；

最后得到N个特征图，描述了鱼塘的一切！

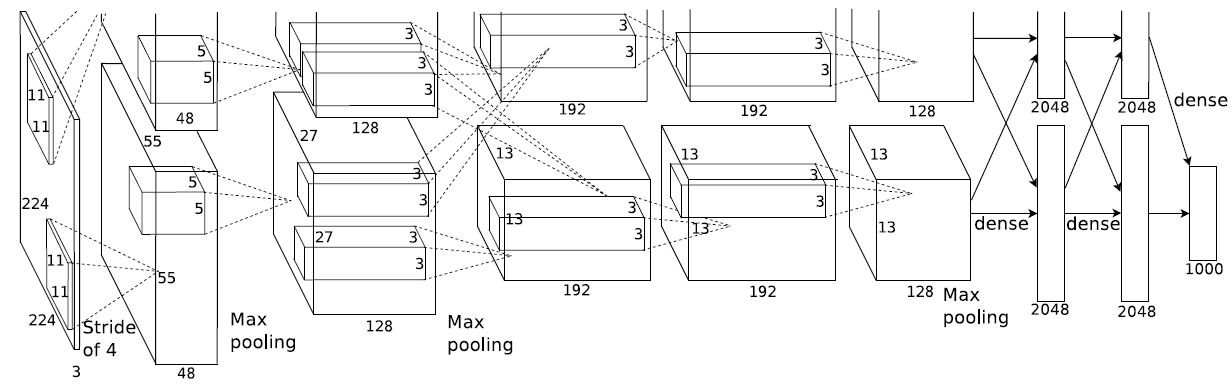
2D卷积表示渔夫的网就是带一圈浮标的渔网，只打上面一层水体的鱼；

3D卷积表示渔夫的网是多层嵌套的渔网，上中下层水体的鱼儿都跑不掉；

1x1卷积可以视为每次移位stride，甩钩钓鱼代替了撒网；

下面解释一下特殊情况的 M > H：

实际上，除了输入数据的通道数比较少之外，中间层的feature map数很多，这样中间层算卷积会累死计算机（鱼塘太深，每层鱼都打，需要的鱼网太重了）。所以很多深度卷积网络把全部通道/特征图划分一下，每个卷积核只看其中一部分（渔夫A的渔网只打捞深水段，渔夫B的渔网只打捞浅水段）。这样整个深度网络架构是横向开始分道扬镳了，到最后才又融合。这样看来，很多网络模型的架构不完全是突发奇想，而是是被参数计算量逼得。特别是现在需要在移动设备上进行AI应用计算(也叫推断), 模型参数规模必须更小, 所以出现很多减少握手规模的卷积形式, 现在主流网络架构大都如此。比如AlexNet：



1. **Logistic Regression:**

即在逻辑回归模型中，我们最大化似然函数和最小化对数似然损失函数实际上是等价的。

L1L1范数：是指向量中各个元素绝对值之和，也有个美称叫“稀疏规则算子”（Lasso regularization）。那么，参数稀疏 有什么好处呢？

一个关键原因在于它能实现 特征的自动选择。一般来说，大部分特征 xixi和输出 yiyi 之间并没有多大关系。在最小化目标函数的时候考虑到这些额外的特征 xixi，虽然可以获得更小的训练误差，但在预测新的样本时，这些没用的信息反而会干扰了对正确 yiyi 的预测。稀疏规则化算子的引入就是为了完成特征自动选择的光荣使命，它会学习地去掉这些没有信息的特征，也就是把这些特征对应的权重置为0。

L2L2范数：它有两个美称，在回归里面，有人把有它的回归叫“岭回归”（Ridge Regression），有人也叫它“权值衰减”(weight decay)。

它的强大之处就是它能 解决过拟合 问题。我们让 L2L2 范数的规则项 ||w||2||w||2 最小，可以使得 ww 的每个元素都很小，都接近于0，但与 L1L1 范数不同，它不会让它等于0，而是接近于0，这里还是有很大区别的。而越小的参数说明模型越简单，越简单的模型则越不容易产生过拟合现象。咦，你为啥说越小的参数表示的模型越简单呢？ 其实我也不知道，我也是猜，可能是因为参数小，对结果的影响就小了吧。

虽然逻辑回归能够用于分类，不过其本质还是线性回归。它仅在线性回归的基础上，在特征到结果的映射中加入了一层sigmoid函数（非线性）映射，即先把特征线性求和，然后使用sigmoid函数来预测。

线性回归在整个实数域内敏感度一致，而分类范围，需要在[0,1]之内。而逻辑回归就是一种减小预测范围，将预测值限定为[0,1]间的一种回归模型，其回归方程与回归曲线如下图所示。逻辑曲线在z=0时，十分敏感，在z>>0或z<<0处，都不敏感，将预测值限定为(0,1)。

逻辑回归跟最大熵模型没有本质区别。逻辑回归是最大熵对应类别为二类时的特殊情况，也就是当逻辑回归类别扩展到多类别时，就是最大熵模型。

逻辑回归跟最大熵模型没有本质区别。逻辑回归是最大熵对应类别为二类时的特殊情况

指数簇分布的最大熵等价于其指数形式的最大似然。

二项式分布的最大熵解等价于二项式指数形式(sigmoid)的最大似然；

多项式分布的最大熵等价于多项式分布指数形式(softmax)的最大似然

1. **overfitting怎么解决？**

dropout、regularization、batch normalization

* 正则化(Regularization)

L2正则化：目标函数中增加所有权重w参数的平方之和, 逼迫所有w尽可能趋向零但不为零. 因为过拟合的时候, 拟合函数需要顾忌每一个点, 最终形成的拟合函数波动很大, 在某些很小的区间里, 函数值的变化很剧烈, 也就是某些w非常大. 为此, L2正则化的加入就惩罚了权重变大的趋势.

L1正则化：目标函数中增加所有权重w参数的绝对值之和, 逼迫更多w为零(也就是变稀疏. L2因为其导数也趋0, 奔向零的速度不如L1给力了). 大家对稀疏规则化趋之若鹜的一个关键原因在于它能实现特征的自动选择。一般来说，xi的大部分元素（也就是特征）都是和最终的输出yi没有关系或者不提供任何信息的，在最小化目标函数的时候考虑xi这些额外的特征，虽然可以获得更小的训练误差，但在预测新的样本时，这些没用的特征权重反而会被考虑，从而干扰了对正确yi的预测。稀疏规则化算子的引入就是为了完成特征自动选择的光荣使命，它会学习地去掉这些无用的特征，也就是把这些特征对应的权重置为0。

* 随机失活(dropout)

在训练的运行的时候，让神经元以超参数p的概率被激活(也就是1-p的概率被设置为0), 每个w因此随机参与, 使得任意w都不是不可或缺的, 效果类似于数量巨大的模型集成。

* 逐层归一化(batch normalization)

这个方法给每层的输出都做一次归一化(网络上相当于加了一个线性变换层), 使得下一层的输入接近高斯分布. 这个方法相当于下一层的w训练时避免了其输入以偏概全, 因而泛化效果非常好.

LR和SVM的联系与区别

【理论功底】

主要考察对机器学习模型的理解，

选择性提问（如果遇到面试者的研究方向自己 不了解但感兴趣，会很欣善，可以趁机学习一个）这块儿的问题会比较细碎，都是我自己 說\思考这松{背书是没用的，这里任何一个点我都可以绐你展开问下去、，

1. 过拟合欠拟合（举几个例子让判断下，顺便问问交叉验证的目的、超参数捜索方法' EarlyStopping)、L1正则和L2的做法、

正则化背后的思想（顺便问问BatchNorm、 Covariance Shift )、

L1正则产生稀疏解原理、

逻辑回归为何线性模型（顺便问问LR如何解决低维不可分、从图模型角度看LR和朴素贝叶斯和无监督）、

几种参数估计方法

MLE/MAP/贝叶斯的联系和区别、

简单说下SVM的支持向量（顺便问问KKT条件、为何对 偶、核的通俗理解）、

GBDTM机森林能否并行（顺便问问bagging boosting )、

生成模型判别模型举个例子

聚类方法的掌握（顺便问问Kmeans的EM推导思路、谱聚类和 Graph-cu的理解）、

梯度下降类方法和牛顿类方法的区别（顺便问问Adanx， L-BFGS的思路）、

半监督的思想（顺便问问一些特定半监督算法是如何利用用无标签数据的、从MAP角度看半监督）、

常见的分类模型的评价指标（顺便问问交叉熵、R0C如何绘制、AUC的物理含义、类别不均衡样本）

2. CNN中卷积操作和卷积核作用、

maxpooling作用、卷积层与全连接层的联系、梯度爆炸和消失的概念（顺便问问神经网结权值初始化的方法、为何能减缓梯度爆炸消失、CNN中有哪些解决办法、LSTM何解决的、如何梯度裁剪、

dropouts如何用在RNN系列网结中、 dropouts止过拟合）、

为何卷积可以用在图像/语音/语句上（顺便问问channel在不同类型 数据源中的含义）

【代码能力】

主要考察实现算法和优化代码的能力，

我一般会先看面试者的githiib repo (如果 简历给出来），看其代码冈格、架构能力，如果设有github,

我会避免问典型的应试题，而是问一些我本人从实际问题中抽象出的小算法题，比如：

1. 给出节点的矩阵和边的矩眸，求路径和最大的路径（来源于Viterbi算法，本质就是个动态 规划），至少给个思路和伪代码（顺便聊聊前向传播和反向传播）

2. 给出一数组，数组元素是pair儿，表示\_个有向无环图的<父亲节点，孩子节点> ,用最优 的方法，将其变成一个新的有序数组，数组元素是该有向无环图所有节点，

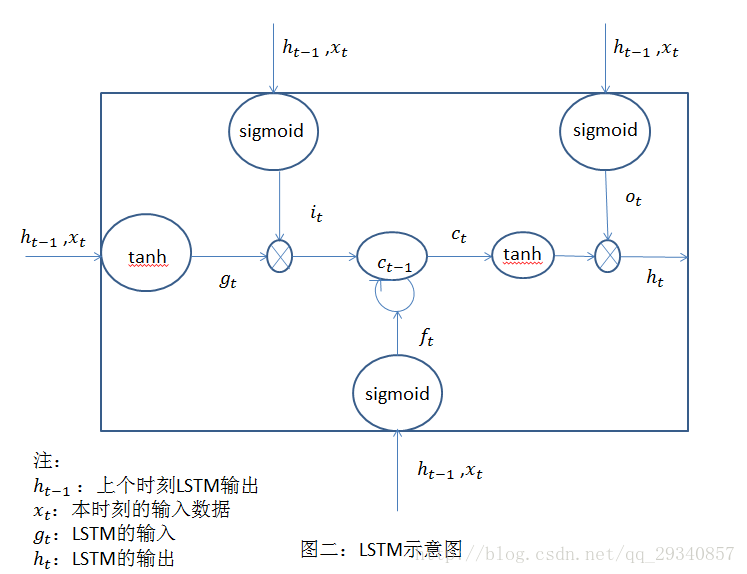
数组的有序性 体现在：父亲节点亥子节点前面（来源于贝叶斯网结实现时的小trick )

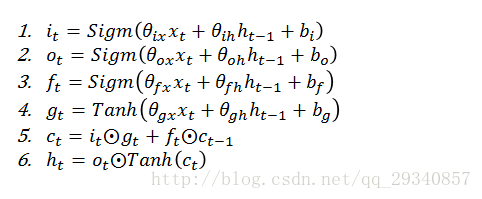
【项目能力】主要考察解决实际问题的思路、填坑能力，这部分最考验面试言功底，要能从面 试者浮夸的插述中寻找有意义的点，并一步步深挖。另外很多dirty work(数据预处理、文本清

1. **RNN中梯度消失的解决方案（LSTM）**

度裁剪（Clipping Gradient）

LSTM（Long Short-Term Memory）





LSTM（Long Short-Term Memory）

一定程度上模仿了长时记忆，相比于梯度裁剪，最大的优点就是，自动学习在什么时候可以将error反向传播，自动控制哪些是需要作为记忆存储在LSTM cell中。一般长时记忆模型包括写入，读取，和忘记三个过程对应到LSTM中就变成了input\_gate,output\_gate,forget\_gate,三个门，范围在0到1之间，相当于对输入输出进行加权的学习，利用大量数据来自动学习加权的参数（即学习了哪些错误可以用BP更新参数），

1. **L1、L2范数**

* L1范数是指向量中各个元素绝对值之和
* L2范数是指向量各元素的平方和然后求平方根

说起L1、L2范数，大家会立马想到这是机器学习中常用的正则化方法，一般添加在损失函数后面，可以看作是损失函数的惩罚项

L1和L2的作用如下：

* L1正则化可以产生稀疏权值矩阵，即产生一个稀疏模型，可以用于特征选择；一定程度上可以防止过拟合
* L2正则化可以防止模型过拟合

理解L1范数

理解L1范数，主要需要理解两个问题。第一是L1产生稀疏矩阵的作用，第二是为什么L1可以产生稀疏模型。

稀疏模型与特征选择

稀疏矩阵指的是很多元素为0、只有少数元素是非零值的矩阵。以线性回归为例，即得到的线性回归模型的大部分系数都是0，这表示只有少数特征对这个模型有贡献，从而实现了特征选择。总而言之，稀疏模型有助于进行特征选择。

L1会趋向于产生少量的特征，而其它特征都是0。

L2会选择更多的特征，这些特征都会趋近于0。L1在特征选择时非常有用，而L2只是一种防止过拟合的方法。在所有特征中只有少数特征起重要作用的情况下，选择L1范数比较合适，因为它能自动选择特征。而如果所有特征中，大部分特征都能起作用，而且起的作用很平均，那么使用L2范数也许更合适。

当对损失函数加上L2正则化以后, 都要乘以一个小于1的因子,系数减少

1. **什么是批标准化 (Batch Normalization)**

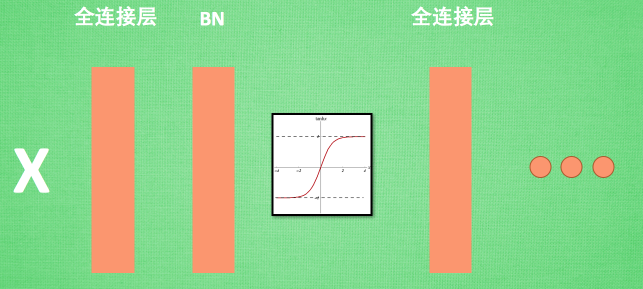
Batch Normalization, 批标准化, 和普通的数据标准化类似, 是将分散的数据统一的一种做法, 也是优化神经网络的一种方法

　　BN的基本思想其实相当直观：因为深层神经网络在做非线性变换前的激活输入值（就是那个x=WU+B，U是输入）随着网络深度加深或者在训练过程中，其分布逐渐发生偏移或者变动，之所以训练收敛慢，一般是整体分布逐渐往非线性函数的取值区间的上下限两端靠近（对于Sigmoid函数来说，意味着激活输入值WU+B是大的负值或正值），所以这导致反向传播时低层神经网络的梯度消失，这是训练深层神经网络收敛越来越慢的本质原因，而BN就是通过一定的规范化手段，把每层神经网络任意神经元这个输入值的分布强行拉回到均值为0方差为1的标准正态分布，其实就是把越来越偏的分布强制拉回比较标准的分布，这样使得激活输入值落在非线性函数对输入比较敏感的区域，这样输入的小变化就会导致损失函数较大的变化，意思是这样让梯度变大，避免梯度消失问题产生，而且梯度变大意味着学习收敛速度快，能大大加快训练速度

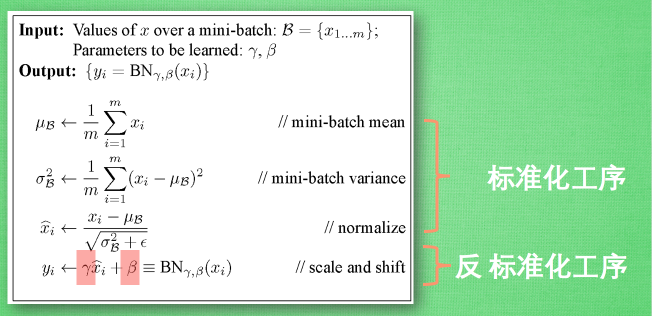
对于每个隐层神经元，把逐渐向非线性函数映射后向取值区间极限饱和区靠拢的输入分布强制拉回到均值为0方差为1的比较标准的正态分布，使得非线性变换函数的输入值落入对输入比较敏感的区域，以此避免梯度消失问题。

BN 添加位置

Batch normalization 的 batch 是批数据, 把数据分成小批小批进行 stochastic gradient descent. 而且在每批数据进行前向传递 forward propagation 的时候, 对每一层都进行 normalization 的处理,



BN 算法：



我们引入一些 batch normalization 的公式. 这三步就是我们在刚刚一直说的 normalization 工序, 但是公式的后面还有一个反向操作, 将 normalize 后的数据再扩展和平移. 原来这是为了让神经网络自己去学着使用和修改这个扩展参数 gamma, 和 平移参数 β, 这样神经网络就能自己慢慢琢磨出前面的 normalization 操作到底有没有起到优化的作用, 如果没有起到作用, 我就使用 gamma 和 belt 来抵消一些 normalization 的操作.

BatchNorm的好处

* 不仅仅极大提升了训练速度，收敛过程大大加快
* 还能增加分类效果，一种解释是这是类似于Dropout的一种防止过拟合的正则化表达方式，所以不用Dropout也能达到相当的效果
* 调参过程也简单多了，对于初始化要求没那么高，而且可以使用大的学习率等

1. **SVM**
2. **机器学习分类模型评价指标**

**混淆矩阵:**  
TP：预测为1，预测正确，即实际1

FP：预测为1，预测错误，即实际0

FN：预测为0，预测错确，即实际1

TN：预测为0，预测正确即，实际0

**准确率:**

准确率的定义是预测正确的结果占总样本的百分比

准确率=(TP+TN)/(TP+TN+FP+FN)

**精准率（Precision）**又叫查准率，它是针对预测结果而言的，它的含义是在所有被预测为正的样本中实际为正的样本的概率，意思就是在预测为正样本的结果中，我们有多少把握可以预测正确，

精准率=TP/(TP+FP)

**召回率**

召回率（Recall）又叫查全率，它是针对原样本而言的，它的含义是在实际为正的样本中被预测为正样本的概率，其公式如下：

精准率=TP/(TP+FN)

精准率和召回率的关系，F1分数

**F1分数**

但通常，如果想要找到二者之间的一个平衡点，我们就需要一个新的指标：F1分数。F1分数同时考虑了查准率和查全率，让二者同时达到最高，取一个平衡。F1分数的公式为 = 2\*查准率\*查全率 / (查准率 + 查全率)。我们在图中看到的平衡点就是F1分数得来的结果。

灵敏度，特异度，真正率，假正率

在正式介绍ROC/AUC之前，我们还要再介绍两个指标，这两个指标的选择也正是ROC和AUC可以无视样本不平衡的原因。这两个指标分别是：灵敏度和（1-特异度），也叫做真正率（TPR）和假正率（FPR）。

灵敏度（Sensitivity） = TP/(TP+FN)

特异度（Specificity） = TN/(FP+TN)

其实我们可以发现灵敏度和召回率是一模一样的，只是名字换了而已。

由于我们比较关心正样本，所以需要查看有多少负样本被错误地预测为正样本，所以使用（1-特异度），而不是特异度。

真正率（TPR） = 灵敏度 = TP/(TP+FN)

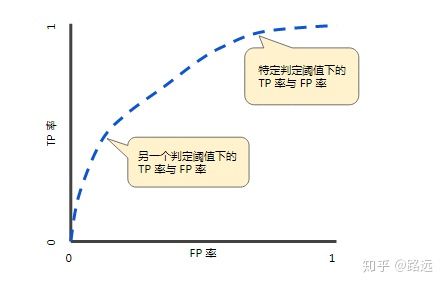
假正率（FPR） = 1- 特异度 = FP/(FP+TN)

下面是真正率和假正率的示意，我们发现TPR和FPR分别是基于实际表现1和0出发的，也就是说它们分别在实际的正样本和负样本中来观察相关概率问题。正因为如此，所以无论样本是否平衡，都不会被影响。还是拿之前的例子，总样本中，90%是正样本，10%是负样本。我们知道用准确率是有水分的，但是用TPR和FPR不一样。这里，TPR只关注90%正样本中有多少是被真正覆盖的，而与那10%毫无关系，同理，FPR只关注10%负样本中有多少是被错误覆盖的，也与那90%毫无关系，所以可以看出：如果我们从实际表现的各个结果角度出发，就可以避免样本不平衡的问题了，这也是为什么选用TPR和FPR作为ROC/AUC的指标的原因。

**ROC（接受者操作特征曲线）**

ROC（Receiver Operating Characteristic）曲线，又称接受者操作特征曲线。该曲线最早应用于雷达信号检测领域，用于区分信号与噪声。后来人们将其用于评价模型的预测能力，ROC曲线是基于混淆矩阵得出的。

ROC曲线中的主要两个指标就是真正率和假正率，上面也解释了这么选择的好处所在。其中横坐标为假正率（FPR），纵坐标为真正率（TPR），下面就是一个标准的ROC曲线图。



**AUC（曲线下的面积）**

为了计算 ROC 曲线上的点，我们可以使用不同的分类阈值多次评估逻辑回归模型，但这样做效率非常低。幸运的是，有一种基于排序的高效算法可以为我们提供此类信息，这种算法称为曲线下面积（Area Under Curve）。

比较有意思的是，如果我们连接对角线，它的面积正好是0.5。对角线的实际含义是：随机判断响应与不响应，正负样本覆盖率应该都是50%，表示随机效果。ROC曲线越陡越好，所以理想值就是1，一个正方形，而最差的随机判断都有0.5，所以一般AUC的值是介于0.5到1之间的。

AUC的一般判断标准

0.5 - 0.7：效果较低，但用于预测股票已经很不错了

0.7 - 0.85：效果一般

0.85 - 0.95：效果很好

0.95 - 1：效果非常好，但一般不太可能

* 1. 计算图的导数计算图解？ 37

线性判别分析（LDA）思想总结 39

2.29 图解LDA核心思想 39

2.30 二类LDA算法原理？ 40

2.30 LDA算法流程总结？ 41

2.31 LDA和PCA区别？ 41

2.32 LDA优缺点？ 41

2.33 主成分分析（PCA）思想总结 42

2.34 图解PCA核心思想 42

2.35 PCA算法推理 43

2.36 PCA算法流程总结 44

2.37 PCA算法主要优缺点 45

2.38 降维的必要性及目的 45

2.39 KPCA与PCA的区别？ 46

2.40 模型评估 47

2.40.1模型评估常用方法？ 47

2.40.2 经验误差与泛化误差 47

2.40.3 图解欠拟合、过拟合 48

2.40.4 如何解决过拟合与欠拟合？ 49

2.40.5 交叉验证的主要作用？ 50

2.40.6 k折交叉验证？ 50

2.40.7 混淆矩阵 50

2.40.8 错误率及精度 51

2.40.9 查准率与查全率 51

2.40.10 ROC与AUC 52

2.40.11 如何画ROC曲线？ 53

2.40.12 如何计算TPR，FPR？ 54

2.40.13 如何计算Auc？ 56

2.40.14 为什么使用Roc和Auc评价分类器？ 56

2.40.15 直观理解AUC 56

2.40.16 代价敏感错误率与代价曲线 57

2.40.17 模型有哪些比较检验方法 59

2.40.18 偏差与方差 59

2.40.19 为什么使用标准差？ 60

2.40.20 点估计思想 61

2.40.21 点估计优良性原则？ 61

1. **支持向量机**

支持向量机（support vector machines, SVM）是一种二分类模型，它的基本模型是定义在特征空间上的间隔最大的线性分类器，间隔最大使它有别于感知机；SVM还包括核技巧，这使它成为实质上的非线性分类器。SVM的的学习策略就是间隔最大化，可形式化为一个求解凸二次规划的问题，也等价于正则化的合页损失函数的最小化问题。SVM的的学习算法就是求解凸二次规划的最优化算法

* 支持向量机解决的问题？

SVM本身是一个二值分类器

* 核函数作用？常见的核函数有哪些？

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

性核函数：线性核，主要用于线性可分的情况，我们可以看到特征空间到输入空间的维度是一样的，其参数少速度快，对于线性可分数据，其分类效果很理想，因此我们通常首先尝试用线性核函数来做分类，看看效果如何，如果不行再换别的

多项式核函数：多项式核函数可以实现将低维的输入空间映射到高纬的特征空间，但是多项式核函数的参数多，当多项式的阶数比较高的时候，核矩阵的元素值将趋于无穷大或者无穷小，计算复杂度会大到无法计算。

高斯（RBF）核函数：高斯径向基函数是一种局部性强的核函数，其可以将一个样本映射到一个更高维的空间内，该核函数是应用最广的一个，无论大样本还是小样本都有比较好的性能，而且其相对于多项式核函数参数要少，因此大多数情况下在不知道用什么核函数的时候，优先使用高斯核函数。

1. 如果特征的数量大到和样本数量差不多，则选用LR或者线性核的SVM；
2. 如果特征的数量小，样本的数量正常，则选用SVM+高斯核函数；
3. 如果特征的数量小，而样本的数量很大，则需要手工添加一些特征从而变成第一种情况。

* 对偶问题
* 理解支持向量回归

* 软间隔与正则化 73
* SVM主要特点及缺点？

SVM有如下主要几个特点：

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

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

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

(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。支持向量机之所以成为目前最常用，效果最好的分类器之一，在于其优秀的泛化能力，这是是因为其本身的优化目标是结构化风险最小，而不是经验风险最小，因此，通过margin的概念，得到对数据分布的结构化描述，因此减低了对数据规模和数据分布的要求。SVM也并不是在任何场景都比其他算法好，对于每种应用，最好尝试多种算法，然后评估结果。如SVM在邮件分类上，还不如逻辑回归、KNN、bayes的效果好。

(10)  它基于结构风险最小化原则，这样就避免了过学习问题，泛化能力强

(11)  它是一个凸优化问题，因此局部最优解一定是全局最优解的优点。

(12)  泛华错误率低，分类速度快，结果易解释

不足之处：

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

(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.

什么是贝叶斯错误率

什么是贝叶斯最优错误率

EM算法解决问题及实现流程

为什么会产生维数灾难？

怎样避免维数灾难

聚类和降维有什么区别与联系？

GBDT和随机森林的区别

四种聚类方法之比较

第三章 深度学习基础 88

3.1 基本概念 88

3.1.1 神经网络组成？ 88

3.1.2 神经网络有哪些常用模型结构？ 90

3.1.3 如何选择深度学习开发平台？ 92

3.1.4 为什么使用深层表示 92

3.1.5 为什么深层神经网络难以训练？ 93

3.1.6 深度学习和机器学习有什么不同 94

3.2 网络操作与计算

1. **CNN的卷积核是单层的还是多层的？**

一般而言，深度卷积网络是一层又一层的。层的本质是特征图, 存贮输入数据或其中间表示值。一组卷积核则是联系前后两层的网络参数表达体, 训练的目标就是每个卷积核的权重参数组。

描述网络模型中某层的厚度，通常用名词通道channel数或者特征图feature map数。不过人们更习惯把作为数据输入的前层的厚度称之为通道数（比如RGB三色图层称为输入通道数为3），把作为卷积输出的后层的厚度称之为特征图数。

卷积核(filter)一般是3D多层的，除了面积参数, 比如3x3之外, 还有厚度参数H（2D的视为厚度1). 还有一个属性是卷积核的个数N。

卷积核的厚度H, 一般等于前层厚度M(输入通道数或feature map数). 特殊情况M > H。

卷积核的个数N, 一般等于后层厚度(后层feature maps数，因为相等所以也用N表示)。

卷积核通常从属于后层，为后层提供了各种查看前层特征的视角，这个视角是自动形成的。

卷积核厚度等于1时为2D卷积，对应平面点相乘然后把结果加起来，相当于点积运算；

卷积核厚度大于1时为3D卷积，每片分别平面点求卷积，然后把每片结果加起来，作为3D卷积结果；1x1卷积属于3D卷积的一个特例，有厚度无面积, 直接把每片单个点乘以权重再相加。

归纳之，卷积的意思就是把一个区域，不管是一维线段，二维方阵，还是三维长方块，全部按照卷积核的维度形状，对应逐点相乘再求和，浓缩成一个标量值也就是降到零维度，作为下一层的一个feature map的一个点的值

可以比喻一群渔夫坐一个渔船撒网打鱼，鱼塘是多层水域，每层鱼儿不同。

船每次移位一个stride到一个地方，每个渔夫撒一网，得到收获，然后换一个距离stride再撒，如此重复直到遍历鱼塘。

A渔夫盯着鱼的品种，遍历鱼塘后该渔夫描绘了鱼塘的鱼品种分布；

B渔夫盯着鱼的重量，遍历鱼塘后该渔夫描绘了鱼塘的鱼重量分布；

还有N-2个渔夫，各自兴趣各干各的；

最后得到N个特征图，描述了鱼塘的一切

2D卷积表示渔夫的网就是带一圈浮标的渔网，只打上面一层水体的鱼；

3D卷积表示渔夫的网是多层嵌套的渔网，上中下层水体的鱼儿都跑不掉；

1x1卷积可以视为每次移位stride，甩钩钓鱼代替了撒网；

下面解释一下特殊情况的 M > H：

实际上，除了输入数据的通道数比较少之外，中间层的feature map数很多，这样中间层算卷积会累死计算机（鱼塘太深，每层鱼都打，需要的鱼网太重了）。所以很多深度卷积网络把全部通道/特征图划分一下，每个卷积核只看其中一部分（渔夫A的渔网只打捞深水段，渔夫B的渔网只打捞浅水段）。这样整个深度网络架构是横向开始分道扬镳了，到最后才又融合。这样看来，很多网络模型的架构不完全是突发奇想，而是是被参数计算量逼得。特别是现在需要在移动设备上进行AI应用计算(也叫推断), 模型参数规模必须更小, 所以出现很多减少握手规模的卷积形式, 现在主流网络架构大都如此。

1. **什么是卷积？**

对图像（不同的数据窗口数据）和滤波矩阵（一组固定的权重：因为每个神经元的多个权重固定，所以又可以看做一个恒定的滤波器filter）做内积（逐个元素相乘再求和）的操作就是所谓的『卷积』操作，也是卷积神经网络的名字来源。

1. **什么是CNN的池化pool层？**

池化，简言之，即取区域平均或最大

1. **deep learning（rnn、cnn）调参的经验**

 参数初始化

1) uniform均匀分布初始化：

传统的初始化权重问题是用标准正态分布（均值为0，方差为1）随机初始化

w = np.random.uniform(low=-scale, high=scale, size=[n\_in,n\_out])

把权值与偏置进行 均匀分布的初始化。用min 与 max 来控制它们的的上下 限，默认为（0，1）

2) Xavier初始法，适用于普通激活函数(tanh,sigmoid)：scale = np.sqrt(3/n)

适用于普通激活函数 (tanh,sigmoid)：stdev = np.sqrt(n)

3) He初始化，适用于ReLU：scale = np.sqrt(6/n)

4) normal高斯分布初始化：w = np.random.randn(n\_in,n\_out) \* stdev

# stdev为高斯分布的标准差，均值设为0

5) svd初始化：对RNN有比较好的效果。

 数据预处理方式

1） 0均值（zero-center） 是最常用的预处理方法，就是把数据的每一维-每一维的均值，这样数据就变成0均值的了。在numpy中，这个操作可以写成：X -= np.mean(X, axis = 0)。对于图片来讲，我们可以更简单地对所有pixel减去同一个均值（如 X -= np.mean(X)），当然也可以对RGB三个通道分别减均值。

2）归一化(Normalization) 是指将数据归一化到相同的尺度。通常有两种归一化的方法。第一种是0均值以后的数据的每一维/每一维的标准差（X /= np.std(X, axis = 0)）；另一种是将数据归一化到每一维的最大最小值为1和-1。这种归一化只适用于当你认为数据的不同维度应该具有相同的重要性时。对于图片来说，不同像素的尺度是基本一致的（0-255），因此我们并不需要对它进行归一化操作。

3） PCA和白化（PCA whitening） 是另一种形式的预处理方法。首先我们将数据变成0均值的，然后计算数据的协方差矩阵来得到数据不同维度之间的相关性：

 训练技巧

1） 要做梯度归一化,即算出来的梯度除以minibatch size

2）clip c(梯度裁剪): 限制最大梯度,其实是value = sqrt(w1^2+w2^2….),如果value超过了阈值,就算一个衰减系系数,让value的值等于阈值: 5,10,15

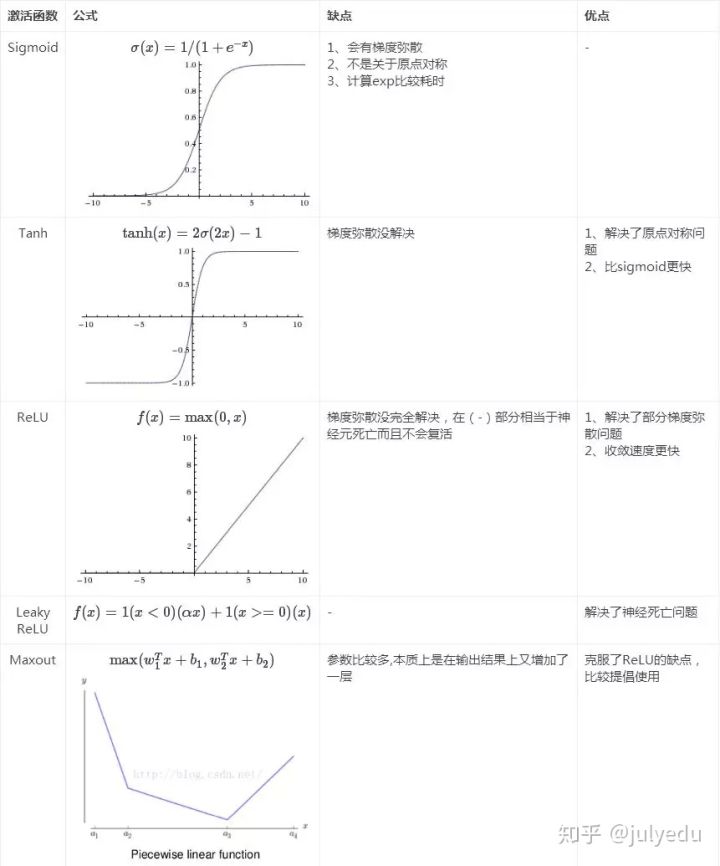
3）dropout对小数据防止过拟合有很好的效果,值一般设为0.5,小数据上dropout+sgd在我的大部分实验中，效果提升都非常明显.因此可能的话，建议一定要尝试一下。 dropout的位置比较有讲究, 对于RNN,建议放到输入->RNN与RNN->输出的位置.

4） adam,adadelta等,在小数据上,我这里实验的效果不如sgd, sgd收敛速度会慢一些，但是最终收敛后的结果，一般都比较好。如果使用sgd的话,可以选择从1.0或者0.1的学习率开始,隔一段时间,在验证集上检查一下,如果cost没有下降,就对学习率减半. 我看过很多论文都这么搞,我自己实验的结果也很好. 当然,也可以先用ada系列先跑,最后快收敛的时候,更换成sgd继续训练.同样也会有提升.据说adadelta一般在分类问题上效果比较好，adam在生成问题上效果比较好。

5） 除了gate之类的地方,需要把输出限制成0-1之外,尽量不要用sigmoid,可以用tanh或者relu之类的激活函数.1. sigmoid函数在-4到4的区间里，才有较大的梯度。之外的区间，梯度接近0，很容易造成梯度消失问题。2. 输入0均值，sigmoid函数的输出不是0均值的。

6) rnn的dim和embdding size,一般从128上下开始调整. batch size,一般从128左右开始调整.batch size合适最重要,并不是越大越好

1. **激活函数**



1. **请问人工神经网络中为什么ReLu要好过于tanh和sigmoid function？**

第一，采用sigmoid等函数，算激活函数时（指数运算），计算量大，反向传播求误差梯度时，求导涉及除法和指数运算，计算量相对大，而采用Relu激活函数，整个过程的计算量节省很多。

第二，对于深层网络，sigmoid函数反向传播时，很容易就会出现梯度消失的情况（在sigmoid接近饱和区时，变换太缓慢，导数趋于0，这种情况会造成信息丢失），这种现象称为饱和，从而无法完成深层网络的训练。而ReLU就不会有饱和倾向，不会有特别小的梯度出现。

第三，Relu会使一部分神经元的输出为0，这样就造成了网络的稀疏性，并且减少了参数的相互依存关系，缓解了过拟合问题的发生（以及一些人的生物解释balabala）。当然现在也有一些对relu的改进，比如prelu，random relu等，在不同的数据集上会有一些训练速度上或者准确率上的改进，具体的大家可以找相关的paper看。

1. **为什么LSTM模型中既存在sigmoid又存在tanh两种激活函数，而不是选择统一一种sigmoid或者tanh？这样做的目的是什么？**

sigmoid 用在了各种gate上，产生0~1之间的值，这个一般只有sigmoid最直接了

tanh 用在了状态和输出上，是对数据的处理，这个用其他激活函数或许也可以

1. **如何解决RNN梯度爆炸和弥散的问题？**

就是当梯度大于一定阈值的的时候，将它截断为一个较小的数。

解决梯度弥散的问题，我们介绍了两种方法。第一种方法是将随机初始化，改为一个有关联的矩阵初始化。第二种方法是使用ReLU（Rectified Linear Units）代替sigmoid函数。ReLU的导数不是0就是1.因此，神经元的梯度将始终为1，而不会当梯度传播了一定时间之后变小。

1. **什麽样的资料集不适合用深度学习？**

（1）数据集太小，数据样本不足时，深度学习相对其它机器学习算法，没有明显优势

（2）数据集没有局部相关特性，目前深度学习表现比较好的领域主要是图像／语音／自然语言处理等领域，这些领域的一个共性是局部相关性。图像中像素组成物体，语音信号中音位组合成单词，文本数据中单词组合成句子，这些特征元素的组合一旦被打乱，表示的含义同时也被改变。对于没有这样的局部相关性的数据集，不适于使用深度学习算法进行处理。举个例子：预测一个人的健康状况，相关的参数会有年龄、职业、收入、家庭状况等各种元素，将这些元素打乱，并不会影响相关的结果。

1. **如何解决梯度消失和梯度膨胀？**

（1）梯度消失：

根据链式法则，如果每一层神经元对上一层的输出的偏导乘上权重结果都小于1的话，那么即使这个结果是0.99，在经过足够多层传播之后，误差对输入层的偏导会趋于0

可以采用ReLU激活函数有效的解决梯度消失的情况，也可以用Batch Normalization解决这个问题。

（2）梯度膨胀

根据链式法则，如果每一层神经元对上一层的输出的偏导乘上权重结果都大于1的话，在经过足够多层传播之后，误差对输入层的偏导会趋于无穷大

可以通过激活函数来解决，或用Batch Normalization解决这个问题。

在深度多层感知机网络中，梯度爆炸会引起网络不稳定，最好的结果是无法从训练数据中学习，而最坏的结果是出现无法再更新的 NaN 权重值。

梯度爆炸导致学习过程不稳定。—《深度学习》，2016。

在循环神经网络中，梯度爆炸会导致网络不稳定，无法利用训练数据学习，最好的结果是网络无法学习长的输入序列数据。

如何确定是否出现梯度爆炸？

训练过程中模型梯度快速变大。

训练过程中模型权重变成 NaN 值。

训练过程中，模型损失变成 NaN。

训练过程中，每个节点和层的误差梯度值持续超过 1.0。

1. **请简述神经网络的发展史**

sigmoid会饱和，造成梯度消失。于是有了ReLU

ReLU负半轴是死区，造成梯度变0。于是有了LeakyReLU，PReLU

强调梯度和权值分布的稳定性，由此有了ELU，以及较新的SELU

太深了，梯度传不下去，于是有了highway

干脆连highway的参数都不要，直接变残差，于是有了ResNet

强行稳定参数的均值和方差，于是有了BatchNorm

在梯度流中增加噪声，于是有了 Dropout

RNN梯度不稳定，于是加几个通路和门控，于是有了LSTM

RNN主要用于自然语言处理(Natural Language Processing)领域，用于处理序列到序列的问题。普通RNN会遇到梯度爆炸和梯度消失的问题。所以现在在NLP领域，一般会使用LSTM模型。在最近的机器翻译领域，Attention作为一种新的手段，也被引入进来

LSTM简化一下，有了GRU

GAN的JS散度有问题，会导致梯度消失或无效，于是有了WGAN

WGAN对梯度的clip有问题，于是有了WGAN-GP

1. **如何修复梯度爆炸问题？**

有很多方法可以解决梯度爆炸问题，本节列举了一些最佳实验方法。

（1） 重新设计网络模型

在深度神经网络中，梯度爆炸可以通过重新设计层数更少的网络来解决。

使用更小的批尺寸对网络训练也有好处。

在循环神经网络中，训练过程中在更少的先前时间步上进行更新（沿时间的截断反向传播，truncated Backpropagation through time）可以缓解梯度爆炸问题。

（2）使用 ReLU 激活函数

在深度多层感知机神经网络中，梯度爆炸的发生可能是因为激活函数，如之前很流行的 Sigmoid 和 Tanh 函数。

使用 ReLU 激活函数可以减少梯度爆炸。采用 ReLU 激活函数是最适合隐藏层的新实践。

（3）使用长短期记忆网络

在循环神经网络中，梯度爆炸的发生可能是因为某种网络的训练本身就存在不稳定性，如随时间的反向传播本质上将循环网络转换成深度多层感知机神经网络。

使用长短期记忆（LSTM）单元和相关的门类型神经元结构可以减少梯度爆炸问题。

采用 LSTM 单元是适合循环神经网络的序列预测的最新最好实践。

（4）使用梯度截断（Gradient Clipping）

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

处理梯度爆炸有一个简单有效的解决方案：如果梯度超过阈值，就截断它们。

——《Neural Network Methods in Natural Language Processing》，2017.

具体来说，检查误差梯度的值是否超过阈值，如果超过，则截断梯度，将梯度设置为阈值。

梯度截断可以一定程度上缓解梯度爆炸问题（梯度截断，即在执行梯度下降步骤之前将梯度设置为阈值）。

——《深度学习》，2016.

在 Keras 深度学习库中，你可以在训练之前设置优化器上的 clipnorm 或 clipvalue 参数，来使用梯度截断。

默认值为 clipnorm=1.0 、clipvalue=0.5。详见：https://keras.io/optimizers/。

（5）使用权重正则化（Weight Regularization）

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

对循环权重使用 L1 或 L2 惩罚项有助于缓解梯度爆炸。

——On the difficulty of training recurrent neural networks，2013.

在 Keras 深度学习库中，你可以通过在层上设置 kernel\_regularizer 参数和使用 L1 或 L2 正则化项进行权重正则化。

1. **在神经网络中，有哪些办法防止过拟合？**
2. Dropout
3. 加L1/L2正则化
4. BatchNormalization
5. 网络bagging
6. **CNN是什么，CNN关键的层有哪些？**

其关键层有：

1. 输入层，对数据去均值，做data augmentation等工作
2. 卷积层，局部关联抽取feature
3. 激活层，非线性变化
4. 池化层，下采样
5. 全连接层，增加模型非线性
6. 高速通道，快速连接
7. BN层，缓解梯度弥散
8. **请简述应当从哪些方向上思考和解决深度学习中出现的的over fitting问题？**

如果模型的训练效果不好，可先考察以下几个方面是否有可以优化的地方。

(1)选择合适的损失函数（choosing proper loss ）

神经网络的损失函数是非凸的，有多个局部最低点，目标是找到一个可用的最低点。非凸函数是凹凸不平的，但是不同的损失函数凹凸起伏的程度不同，例如下述的平方损失和交叉熵损失，后者起伏更大，且后者更容易找到一个可用的最低点，从而达到优化的目的。

Square Error（平方损失）

Cross Entropy（交叉熵损失）

(2)选择合适的Mini-batch size

采用合适的Mini-batch进行学习，使用Mini-batch的方法进行学习，一方面可以减少计算量，一方面有助于跳出局部最优点。因此要使用Mini-batch。更进一步，batch的选择非常重要，batch取太大会陷入局部最小值，batch取太小会抖动厉害，因此要选择一个合适的batch size。

(3)选择合适的激活函数（New activation function）

使用激活函数把卷积层输出结果做非线性映射，但是要选择合适的激活函数。

- Sigmoid函数是一个平滑函数，且具有连续性和可微性，它的最大优点就是非线性。但该函数的两端很缓，会带来猪队友的问题，易发生学不动的情况，产生梯度弥散。

- ReLU函数是如今设计神经网络时使用最广泛的激活函数，该函数为非线性映射，且简单，可缓解梯度弥散。

(4)选择合适的自适应学习率（apdative learning rate）

- 学习率过大，会抖动厉害，导致没有优化提升

- 学习率太小，下降太慢，训练会很慢

(5)使用动量（Momentum）

在梯度的基础上使用动量，有助于冲出局部最低点。

如果以上五部分都选对了，效果还不好，那就是产生过拟合了，可使如下方法来防止过拟合，分别是

- 1.早停法（earyly stoping）。早停法将数据分成训练集和验证集，训练集用来计算梯度、更新权重和阈值，验证集用来估计误差，若训练集误差降低但验证集误差升高，则停止训练，同时返回具有最小验证集误差的连接权和阈值。

- 2.权重衰减（Weight Decay）。到训练的后期，通过衰减因子使权重的梯度下降地越来越缓。

- 3.Dropout。Dropout是正则化的一种处理，以一定的概率关闭神经元的通路，阻止信息的传递。由于每次关闭的神经元不同，从而得到不同的网路模型，最终对这些模型进行融合。

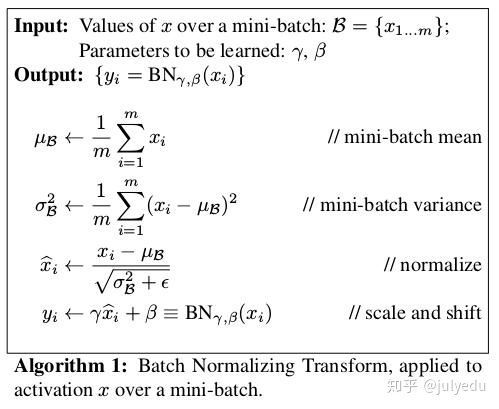
- 4.调整网络结构（Network Structure）。

1. **对于分类任务，通常使用交叉熵损失函数,回归任务中那样使用均方差**
2. **请写出Batch Normalization的计算方法及其应用**

批量归一化（Batch Normalization简称BN，其中，Normalization是数据标准化或归一化、规范化，Batch可以理解为批量，加起来就是批量标准化。解决在训练过程中中间层数据分布发生改变的问题，以防止梯度消失或爆炸、加快训练速度

与激活函数层、卷积层、全连接层、池化层一样，BN(Batch Normalization)也属于网络的一层。

BN的本质原理：在网络的每一层输入的时候，又插入了一个归一化层，也就是先做一个归一化处理（归一化至：均值0、方差为1），然后再进入网络的下一层。不过归一化层可不像我们想象的那么简单，它是一个可学习、有参数（γ、β）的网络层。



输入：输入数据x1..xm（这些数据是准备进入激活函数的数据）

计算过程中可以看到,

1.求数据均值；

2.求数据方差；

3.数据进行标准化（个人认为称作正态化也可以）

4.训练参数γ，β

5.输出y通过γ与β的线性变换得到新的值

因为BN保证每一层的输入分布稳定，这一点本身可以使得训练加速，而且另一方面它也可以帮助减少梯度消失和梯度爆炸的现象。

1. **损失函数**

损失函数（loss function）是用来估量你模型的预测值f(x)与真实值Y的不一致程度，它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。

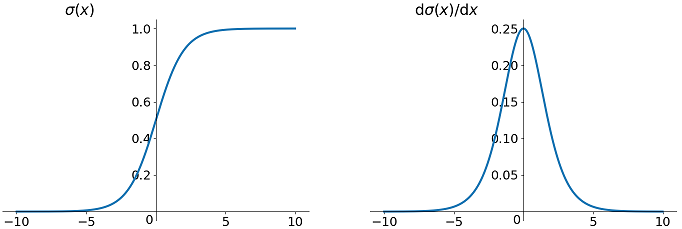
1. **深度神经网络（DNN）损失函数和激活函数的选择**

如果不用激励函数（相当于激励函数是f(x)=x），在这种情况下，每一层的输出都是上一层的线性函数，无论神经网络有多少层，输出都是输入的线性组合，这与一个隐藏层的效果相当（这种情况就是多层感知机MPL）。

对比激活函数Sigmoid、ReLu，tanh

Sigmoid函数

https://img-blog.csdn.net/20170903152554267?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGlsdTkxNg==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

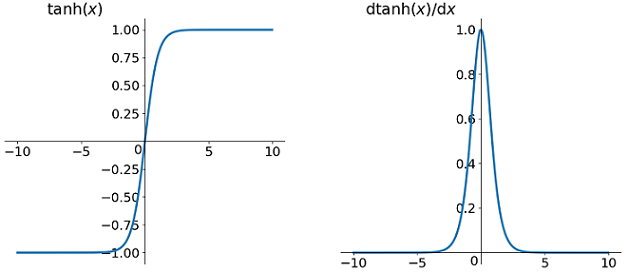


但是Sigmoid函数有3大缺点：

1. 容易出现梯度消失
2. 输出不是zero-centered
3. 幂运算相对耗时

tanh函数:

https://img-blog.csdn.net/20170903160024954



优点

全程可导；输出区间为-1到1；解决了zero-centered的输出问题。

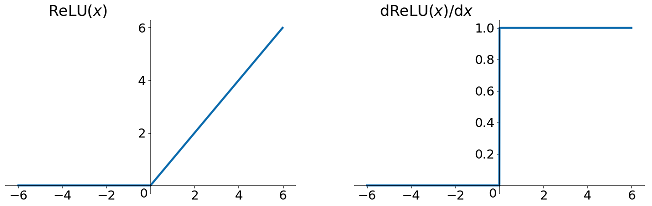
缺点

梯度消失的问题和幂运算的问题仍然存在。

ReLU函数:

ReLU函数(Rectified Linear Units)其实就是一个取最大值函数，注意这并不是全区间可导的，但是我们可以取次梯度(subgradient)。

https://img-blog.csdn.net/20170903160343431



优点

解决了梯度消失的问题 (在正区间)

计算速度非常快，只需要判断输入是否大于0

收敛速度远快于sigmoid和tanh

Relu会使一部分神经元的输出为0，这样就造成了网络的稀疏性，并且减少了参数的相互依存关系，缓解了过拟合问题的发生

缺点

输出不是zero-centered

Dead ReLU Problem

Dead ReLU Problem指的是某些神经元可能永远不会被激活，导致相应的参数永远不能被更新。有两个主要原因可能导致这种情况产生: (1) 非常不幸的参数初始化，这种情况比较少见 (2) 学习速率太高导致在训练过程中参数更新太大，不幸使网络进入这种状态。解决方法是可以采用Xavier初始化方法，以及避免将学习速率设置太大或使用adagrad等自动调节学习速率的算法。

尽管存在这两个问题，ReLU目前仍是最常用的激活函数。

在深度神经网络（DNN）反向传播算法(BP)中，我们对DNN的前向反向传播算法的使用做了总结。其中使用的损失函数是均方差，而激活函数是Sigmoid. MSE损失+Sigmoid激活函数的问题. Sigmoid的这个曲线意味着在大多数时候，我们的梯度变化值很小，导致我们的W,b更新到极值的速度较慢，也就是我们的算法收敛速度较慢。

交叉熵损失+Sigmoid改进收敛速度

https://ask.qcloudimg.com/http-save/yehe-1557966/rdwaww6k2m.png?imageView2/2/w/1620

对数似然损失+softmax进行分类输出:

对于用于分类的softmax激活函数，对应的损失函数一般都是用对数似然函数，即：

https://ask.qcloudimg.com/http-save/yehe-1557966/u8vhh0652z.png?imageView2/2/w/1620

什么是梯度爆炸和梯度消失呢？简单理解，就是在反向传播的算法过程中，由于我们使用了是矩阵求导的链式法则，有一大串连乘，如果连乘的数字在每层都是小于1的，则梯度越往前乘越小，导致梯度消失，而如果连乘的数字在每层都是大于1的，则梯度越往前乘越大，导致梯度爆炸。

上面我们对DNN损失函数和激活函数做了详细的讨论，重要的点有：

1）如果使用sigmoid激活函数，则交叉熵损失函数一般肯定比均方差损失函数好；

2）如果是DNN用于分类，则一般在输出层使用softmax激活函数和对数似然损失函数；

3）ReLU激活函数对梯度消失问题有一定程度的解决，尤其是在CNN模型中。

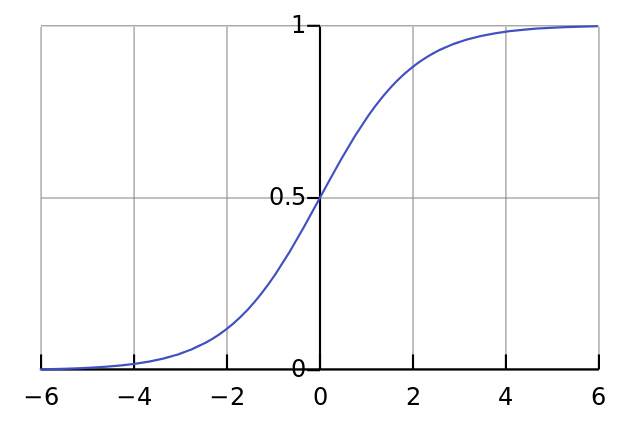
1. **SGD中S（Stochastic）代表什么**

随机选取一个样本来学习

梯度下降的优化框架有三种：批量梯度下降（全），随机梯度下降（一），小批量梯度下降（mini）。它们不同之处在于每次学习（更新模型参数）使用的样本个数，每次更新使用不同的样本会导致每次学习的准确性和学习时间不同

1. **logistic函数和softmax函数**

logistic函数



softmax函数经常用在神经网络的最后一层，作为输出层，进行多分类。

Logistic具体针对的是二分类问题，而softmax解决的是多分类问题，因此从这个角度也可以理解logistic函数是softmax函数的一个特例。

对于Softmax回归当分类数目k=2的时候，Softmax将会退化为Logistic回归

logistic regression+MSE到底哪里有欠缺

logistic和cross-entropy,关于交叉熵

第一章 数学基础

1. **标量、向量、张量之间的联系**

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。使用粗体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 ） 。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. **张量与矩阵的区别？**
2. 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的。
3. 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。
4. 张量可以用3×3 矩阵形式来表达。4 表示标量的数和表示矢量的三维数组也可分别看作1×1，1×3 的矩阵。
5. **向量和矩阵的范数归纳**

范数（Norms）被用于度量矩阵的大小，或者相应地，度量向量的长度。范数是一个函数，它将 R^mxn（或 R^n）映射到 R。形式地说：

定义 1：任何函数满足 || · ||: R^m×n → R 和下列性质，则称为一个范数：

非负性：|| A ||≥0；|| A ||=0 当且仅当 A=0；

三角不等律：|| A+B ||≤|| A ||+|| B ||；

标量乘法律：|| αA ||=|α| || A ||，α∈R。

可以很容易地证明以下两个性质：

|| A ||=|| -A ||

| || A ||-|| B || | ≤ || A-B ||

第二个性质被称为倒三角型不等式。

定义一个矩阵A=[-1 2 -3；4 -6 6]；

矩阵的1范数：矩阵每一列上的元素绝对值先求和，再从中取最大值（列和最大）。上述矩阵A的1范数就是9；

矩阵的2范数：矩阵$A^TA$的最大特征值开平方根，上述矩阵A的2范数就是：10.0623；

矩阵的无穷范数：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)，上述矩阵的结果就是：16；

二范数相信大家在本科学线代的时候就已经被灌输了“用来度量向量长度”、“用来度量向量空间中两个点的距离”这两个典型意义。那么在机器学习中，二范数主要有什么重要的应用呢？

二范数会削弱强特征，增强弱特征

1. **如何判断一个矩阵为正定？**

考虑矩阵的特征值。

* 若所有特征值均不小于零，则称为半正定。
* 若所有特征值均大于零，则称为正定。

正定、半正定矩阵的直觉代表一个向量经过它的变化后的向量与其本身的夹角小于等于90度。

1. **特征值分解与特征向量**

特征向量是一个向量，当在它上面应用线性变换时其方向保持不变

* 特征值就是运动的速度
* 特征向量就是运动的方向

1. **奇异值与特征值有什么关系？**

矩阵可以认为是一种线性变换，如果将这种线性变换放在几何意义上，则他的作用效果和基的选择有关

以Ax = b为例，x是m维向量，b是n维向量，m,n可以相等也可以不相等，表示矩阵可以将一个向量线性变换到另一个向量，这样一个线性变换的作用可以包含旋转、缩放和投影三种类型的效应。

奇异值分解将一个矩阵原本混合在一起的三种作用效果，分解出来了

特征值分解其实是对旋转缩放两种效应的归并, 求特征向量和特征值的过程，我们找到了这样一组基，在这组基下，矩阵的作用效果仅仅是存粹的缩放。

特征值分解和奇异值分解都是给一个矩阵(线性变换)找一组特殊的基，特征值分解找到了特征向量这组基，在这组基下该线性变换只有缩放效果。而奇异值分解则是找到另一组基，这组基下线性变换的旋转、缩放、投影三种功能独立地展示出来了。

又因为有投影效应的矩阵不是方阵，没有特征值，所以奇异值分解可以适用于所有矩阵，但特征值分解就仅仅适用于方阵了。

1. **机器学习为什么要使用概率？**

概率论是用于表示不确定性声明的数学框架。它不仅提供了量化不确定性的方法，也提供了用于导出新的不确定性声明的公理。

概率论使我们能够作出不确定的声明以及在不确定性存在的情况下进行推理，而信息论使我们能够量化概率分布中的不确定性的总量。

1. **常见概率分布？**



1. **举例理解条件概率**

P（A|B）=P(AB)/P(B)

1. **条件概率的链式法则**

2个事件同时发生的概率：

P(a, b) = P(a | b) \* P(b)

其中：P(a, b)表示 a和b事件同时发生的概率， P(a | b)是一个条件概率，表示在b事件发生的条件下，a发生的概率

3个事件的概率链式调用：

P(a, b, c) = P(a | b, c) \* P(b, c) = P(a | b, c) \* P(b | c) \* P(c)

推广到N个事件，概率链式法则长这样：

P(X1, X2, ... Xn) = P(X1 | X2, X3 ... Xn) \* P(X2 | X3, X4 ... Xn) ... P(Xn-1 | Xn) \* P(Xn)

1. **独立性和条件独立性**

独立性

在概率论中，独立性是指随机变量的分布不因知道其它随机变量的值而改变。在机器学习中，我们通常都会对数据做这样的假设。例如，我们会假设训练样本是从某一底层空间独立提取；并且假设样例i的标签独立于样例j(i≠j)的特性。

从数学角度来说，随机变量X独立于Y，当：

P(X)=P(X|Y)

注意，上式没有标明X,Y的取值，也就是说该公式对任意X,Y可能的取值均成立。）

利用等式(2)，很容易可以证明如果X对Y独立，那么Y也独立于X。当X和Y相互独立时，记为X⊥Y。

对于随机变量X和Y的独立性，有一个等价的数学公式：

P(X,Y)=P(X)P(Y)

我们有时也会讨论条件独立，就是当我们当我们知道一个随机变量（或者更一般地，一组随机变量）的值时，那么其它随机变量之间相互独立。正式地，我们说“给定Z，X和Y条件独立”，如果：

P(X|Z)=P(X|Y,Z)

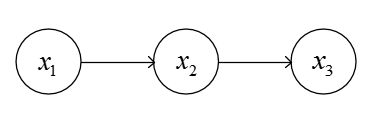
或者等价的：

P(X,Y|Z)=P(X|Z)P(Y|Z)

x1和x2相互独立

https://pic2.zhimg.com/80/v2-63035da666e9b498f3543a9de586986a_hd.jpg

在有向图模型中，若两个变量之间没有连接并且没有共同的祖先，则相互独立。条件独立性：x1和x3在给定x2时条件独立



https://pic1.zhimg.com/80/v2-0fe42f48b0f7030ff33ff85d0cb7249d_hd.jpg

意味着可对联合分布做因式分解变为条件分布的乘积并减少冗余，从而用更少的参数描述条件概率分布。

在有向图模型中，任何变量与其对应的马尔科夫覆盖是条件独立的。

1. **期望、方差、协方差、相关系数总结**

1：数学期望

数学期望是随机变量的重要特征之一,随机变量X的数学期望记为E(X),E(X)是X的算术平均的近似值,数学期望表示了X的平均值大小。

当X为离散型随机变量时,并且其分布律为 P(X=xk) ＝ p^k ,其中k=1,2,…,n；则数学期望[image](https://images0.cnblogs.com/blog/383728/201312/05170453-bdc59180ac7e4e0cb3d91125de4cdfed.png)（要求绝对收敛）.当X为连续型随机变量时,设其概率密度为f(x),则数学期望为[image](https://images0.cnblogs.com/blog/383728/201312/05170455-362c90f2dc5840ac909058d9f2235d21.png)（要求绝对收敛）.

2: 方差

数学期望给出了随机变量的平均大小,现实生活中我们还经常关心随机变量的取值在均值周围的散布程度,而方差就是这样的一个数字特征。

设X是随机变量,并且E{[X-E(X)2]}存在,则称它为X的方差,记为D(X)。

* 当X为离散型时,D(x) = [image](https://images0.cnblogs.com/blog/383728/201312/05170502-98480c1a46db4b4bad9f26db7c131624.png).
* 当X为连续型时,D(x) = [image](https://images0.cnblogs.com/blog/383728/201312/05170505-50e7558dbdab43a49660ee387dadc96d.png).

方差的算术平方根[image](https://images0.cnblogs.com/blog/383728/201312/05170511-e27260356493422ea89bc4547b4d23ed.png)为X的标准差。

另外,D(X) = E{[X-E(X)2]} 经过化解可得 D(X) = E(X2) – [E(X)]^2 .我们一般计算的时候常用这个式子。

3： 协方差

对于二维的随机变量(X,Y)，我们还要讨论它们的相互关系,协方差就是一个这样的数字特征。

因为E{[X-E(X)][Y-E[Y]]} = E(XY) – E(X)E(Y).

又当X,Y相互独立的时候E(XY) = E(X)E(Y).这意味着若E{[X-E(X)][Y-E[Y]]} ≠ 0 ,则X与Y是存在一定关系的。我们把E{[X-E(X)][Y-E[Y]]} 称为随机变量X与Y的协方差。记为Cov(X,Y).即：Cov(X,Y) = E{[X-E(X)][Y-E[Y]]}

4.相关系数

协方差在某种意义上是表示了两个随机变量间的关系,但是Cov(X,Y)的取值大小与X,Y的量纲有关,不方便分析,所以为了避免这一点,我们用X,Y的标准化随机变量来讨论。

我们称[image](https://images0.cnblogs.com/blog/383728/201312/05170806-7ac88c6e93904958a0ab68fe2b8f0c0d.png)为随机变量X与Y的相关系数,记为[image](https://images0.cnblogs.com/blog/383728/201312/05170809-418f3461098d4ea8958922eac2107dbd.png)(无量纲)。

其中[image](https://images0.cnblogs.com/blog/383728/201312/05170816-2f36ee8ce8c24b6ca60538eec2d81c95.png)为X,Y的协方差即Cov(X,Y),D(X),D(Y)分别是X,Y的方差且D(X)>0，D(Y)>0。

关于相关系数，我们有下面的性质：

|[image](https://images0.cnblogs.com/blog/383728/201312/05170821-7e6bf11030ed4f10b48d915a3260b4be.png)| ≤ 1

|[image](https://images0.cnblogs.com/blog/383728/201312/05170828-9e209e6121f742009ea6768fc5525c2c.png)| = 1 的充要条件是X 与 Y 以概率 1 存在线性关系，即 P{Y = a +bX} = 1, a,b是常数。

若[image](https://images0.cnblogs.com/blog/383728/201312/05170842-c2324cf2469f42cfaeabebc686c05e91.png)= 0,则说明X,Y不相关并且X与Y不存在线性关系。

若随机变量X,Y相互独立，则[image](https://images0.cnblogs.com/blog/383728/201312/05170859-23aaeb71d23b41f3ade82248cebd331c.png) = 0，即X,Y不相关。

注意：两个不相关的随机变量，不一定相互独立,有一特殊情况是,当随机变量X,Y服从二维正态分布的时候,独立与不相关等价。

不相关只能说明X与Y不存在线性关系。

独立说明X与Y既不存在线性关系,也不存在非线性关系。

第二章 机器学习基础

1. **各种常见算法:监督学习、非监督学习、半监督学习、弱监督学习？**

* 监督学习（supervised learning）:

已知数据和其对应的标签，训练一个算法，将输入数据映射到标签的过程

已知一些图片是狗，一些图片不是狗，那么训练一个算法，当一个新的图片输入算法的时候算法告诉我们这张图片是不是狗

* 无监督学习(unsupervised learning)

已知数据不知道任何标签，按照一定的偏好，训练一个智能算法，将所有的数据映射到多个不同标签的过程。所谓的按照一定的偏好，是比如特征空间距离最近，等人们认为属于一类的事物应具有的一些特点。举个例子，猪和鸵鸟混杂在一起，算法会测量高度，发现动物们主要集中在两个高度，一类动物身高一米左右，另一类动物身高半米左右，那么算法按照就近原则，75厘米以上的就是高的那类也就是鸵鸟，矮的那类是第二类也就是猪，当然这里也会出现身材矮小的鸵鸟和身高爆表的猪会被错误的分类

* 半监督学习(semi supervised learning)

已知数据和部分数据对应的标签，有一部分数据的标签未知，训练一个智能算法，学习已知标签和未知标签的数据，将输入数据映射到标签的过程。半监督通常是一个数据的标注非常困难，比如说医院的检查结果，医生也需要一段时间来判断健康与否，可能只有几组数据知道是健康还是非健康，其他的只有数据不知道是不是健康。

* 弱监督学习(weakly supervised learning)：

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，如果我们知道一幅图，告诉你图上有一只猪，然后需要你把猪在哪里，猪和背景的分界在哪里找出来，那么这就是一个已知若标签，去学习强标签的弱监督学习问题。

对于目标检测任务，图像分类的标签相比物体的bounding box是一种弱监督的标注，对于语义分割任务，image-level的标签和物体的bounding box相比pixel-level（像素层面）的标签则是一种弱监督的标注。

1. **监督学习有哪些步骤**

监督学习（Supervised Learning）

监督学习是使用已知正确答案的示例来训练网络的。想象一下，我们可以训练一个网络，让其从照片库中（其中包含你父母的照片）识别出你父母的照片。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，我们要浏览你的照片（数据集），确定所有有你父母的照片，并对其进行标注，从而开始此过程。然后我们将把整堆照片分成两堆。我们将使用第一堆来训练网络（训练数据），而通过第二堆来查看模型在选择我们父母照片操作上的准确程度（验证数据）。

等到数据集准备就绪后，我们就会将照片提供给模型。在数学上，我们的目标就是在深度网络中找到一个函数，这个函数的输入是一张照片，而当你的父母不在照片中时，其输出为0，否则输出为1。

此步骤通常称为分类任务。在这种情况下，我们进行的通常是一个结果为yes or no的训练，但事实是，监督学习也可以用于输出一组值，而不仅仅是0或1。例如，我们可以训练一个网络，用它来输出一个人偿还信用卡贷款的概率，那么在这种情况下，输出值就是0到100之间的任意值。这些任务我们称之为回归。

步骤2：训练

为了继续该过程，模型可通过以下规则（激活函数）对每张照片进行预测，从而决定是否点亮工作中的特定节点。这个模型每次从左到右在一个层上操作——现在我们将更复杂的网络忽略掉。当网络为网络中的每个节点计算好这一点后，我们将到达亮起（或未亮起）的最右边的节点（输出节点）。

既然我们已经知道有你父母的照片是哪些图片，那么我们就可以告诉模型它的预测是对还是错。然后我们会将这些信息反馈（feed back）给网络。

该算法使用的这种反馈，就是一个量化“真实答案与模型预测有多少偏差”的函数的结果。这个函数被称为成本函数（cost function），也称为目标函数（objective function），效用函数（utility function）或适应度函数（fitness function）。然后，该函数的结果用于修改一个称为反向传播（backpropagation）过程中节点之间的连接强度和偏差，因为信息从结果节点“向后”传播。

我们会为每个图片都重复一遍此操作，而在每种情况下，算法都在尽量最小化成本函数。

其实，我们有多种数学技术可以用来验证这个模型是正确还是错误的，但我们常用的是一个非常常见的方法，我们称之为梯度下降（gradient descent）。

步骤3：验证

一旦我们处理了第一个堆栈中的所有照片，我们就应该准备去测试该模型。我们应充分利用好第二堆照片，并使用它们来验证训练有素的模型是否可以准确地挑选出含有你父母在内的照片。

步骤4：使用

最后，一旦你有了一个准确的模型，你就可以将该模型部署到你的应用程序中。你可以将模型定义为API调用，例如ParentsInPicture(photo)，并且你可以从软件中调用该方法，从而导致模型进行推理并给出相应的结果。

1. **分类网络和回归的区别？**

loss function不同，分类的损失函数一般用交叉熵这种，而回归的损失函数一般用类似均方误差（MSE,mean squared error）

举几个例子:

* Logistic Regression 和 Linear Regression：

1. Linear Regression： 输出一个标量 wx+b，这个值是连续值，所以可以用来处理回归问题
2. Logistic Regression：把上面的 wx+b 通过 sigmoid 函数映射到(0,1)上，并划分一个阈值, 大于阈值的分为一类，小于等于分为另一类，可以用来处理二分类问题
3. 更进一步：对于N分类问题，则是先得到N组w值不同的 wx+b，然后归一化，比如用softmax 函数，最后变成N个类上的概率，可以处理多分类问题

* Support Vector Regression 和 Support Vector Machine:

1. SVR：输出 wx+b，即某个样本点到分类面的距离，是连续值，所以是回归模型
2. SVM：把这个距离用 sign(·) 函数作用，距离为正(在超平面一侧)的样本点是一类，为负的是另一类，所以是分类模型

* Naive Bayes 用于分类 和 回归:

1. 用于分类：y是离散的类别，所以得到离散的 p(y|x)，给定 x ，输出每个类上的概率
2. 用于回归：对上面离散的 p(y|x)求期望 ΣyP(y|x)，就得到连续值。但因为此时y本身是连续的值，所以最地道的做法是，得到连续的概率密度函数p(y|x)，然后再对y求期望。参考

* 前馈神经网络(如 CNN 系列) 用于分类和回归:

1. 用于回归：最后一层有m个神经元，每个神经元输出一个标量，m个神经元的输出可以看做向量 v，现全部连到一个神经元上，则这个神经元输出 wv+b，是一个连续值，可以处理回归问题，跟上面 Linear Regression 思想一样
2. 用于N分类：现在这m个神经元最后连接到 N 个神经元，就有 N 组w值不同的 wv+b，同理可以归一化（比如用 softmax ）变成 N个类上的概率（补充一下，如果不用 softmax，而是每个 wx+b 用一个 sigmoid，就变成多标签问题，跟多分类的区别在于，样本可以被打上多个标签）

* 循环神经网络(如 RNN 系列) 用于分类 和 回归：

1. 用于回归 和 分类： 跟 CNN 类似，输出层的值 y = wv+b，可做分类可做回归，只不过区别在于，RNN 的输出跟时间有关，即输出的是 {y(t), y(t+1),...}序列

1. **常用分类算法的优缺点？**

* 决策树

一种启发式算法，核心是在决策树各个节点上应用信息增益等准则来选取特征，进而递归地构造决策树。

优点：

1. 计算复杂度不高，易于理解和解释，可以理解决策树所表达的意义；

2. 数据预处理阶段比较简单，且可以处理缺失数据；

3. 能够同时处理数据型和分类型属性，且可对有许多属性的数据集构造决策树；

4. 是一个白盒模型，给定一个观察模型，则根据所产生的决策树很容易推断出相应的逻辑表达式；

5. 在相对短的时间内能够对大数据集合做出可行且效果良好的分类结果。

6.  可以对有许多属性的数据集构造决策树。

缺点：

1. 对于那些各类别样本数目不一致的数据，信息增益的结果偏向于那些具有更多数值的属性；

2. 对噪声数据较为敏感；

3. 容易出现过拟合问题；

4. 忽略了数据集中属性之间的相关性；

5.处理缺失数据时的困难

* KNN算法

一种惰性分类方法，从训练集中找出k个最接近测试对象的训练对象，再从这k个训练对象中找出居于主导的类别，将其赋给测试对象。

优点：

1. 简单有效，容易理解和实现；

2. 重新训练的代价较低（类别体系的变化和训练集的变化）；

3. 计算时间和空间线性于训练集的规模；

4. 错误率渐进收敛于贝叶斯错误率，可作为贝叶斯的近似；

5. 适合处理多模分类和多标签分类问题；

6. 对于类域的交叉或重叠较多的待分类样本集较为适合；

缺点：

1. 是懒散学习方法，比一些积极学习的算法要慢；

2. 计算量比较大，需对样本点进行剪辑；

3. 对于样本不平衡的数据集效果不佳，可采用加权投票法改进；

4. k值的选择对分类效果有很大影响，较小的话对噪声敏感，需估计最佳k值。

5.可解释性不强，计算量大。

* 朴素贝叶斯算法

贝叶斯分类器的分类原理是利用各个类别的先验概率，再利用贝叶斯公式及独立性假设计算出属性的类别概率以及对象的后验概率，即该对象属于某一类的概率，选择具有最大后验概率的类作为该对象所属的类别。

优点：

1. 数学基础坚实，分类效率稳定，容易解释；

2. 所需估计的参数很少，对缺失数据不太敏感；

3. 无需复杂的迭代求解框架，适用于规模巨大的数据集。

缺点：

1. 属性之间的独立性假设往往不成立（可考虑用聚类算法先将相关性较大的属性进行聚类）；

2. 需要知道先验概率，分类决策存在错误率。

* SVM算法

对于两类线性可分学习任务，SVM找到一个间隔最大的超平面将两类样本分开，最大间隔能够保证该超平面具有最好的泛化能力。

优点：

1. 可以解决小样本情况下的ML问题；

2. 可以提高泛化性能；

3. 可以解决高维问题，避免维数灾难；

4. 可以解决非线性问题；

5. 可以避免神经网络结构选择和局部极小点问题。

参数C和g的选择对分类性能的影响：

C是惩罚系数，C越大，交叉validation高，容易过学习；

g是核函数的到达0的速率，g越小，函数下降快，交叉validation高，也容易造成过学习。

缺点：

1. 对缺失数据敏感；

2. 对非线性问题没有通用解决方案，必须谨慎选择kernel function来处理。

* Logistic回归算法

二项logistic回归模型是一种分类模型，由条件概率分布P(Y|X)表示，形式为参数化的logistic分布。这里随机变量X取值为实数，随机变量Y取值为1或0。可以通过有监督的方法来估计模型参数。

优点：

1. 计算代价不高，易于理解和实现；

2. 适用于数值型和分类型数据。

缺点：

1. 容易欠拟合；

2. 分类精度可能不高。

* 人工神经网络

优点：

1. 分类的准确度高，并行分布处理能力强，分布存储及学习能力强；

2. 对噪声神经有较强的鲁棒性和容错能力，能充分逼近复杂的非线性关系，具备联想记忆的功能等。

缺点：

1. 神经网络需要大量的参数，如网络拓扑结构、权值和阈值的初始值；

2. 不能观察之间的学习过程，输出结果难以解释，会影响到结果的可信度和可接受程度；

3. 学习时间过长,甚至可能达不到学习的目的。

4. 算法对初始种群的选择有一定的依赖性，能够结合一些启发算法进行改进。

以下是我这些年总结的指南训练集有多大？

* 如果你的训练集很小，高偏差/低方差的分类器（如朴素贝叶斯）比低偏差/高方差的分类器（如K近邻或Logistic回归）更有优势，因为后者容易过拟合。但是随着训练集的增大，高偏差的分类器并不能训练出非常准确的模型，所以低偏差/高方差的分类器会胜出（它们有更小的渐近误差）。你也可以从生成模型与鉴别模型的区别来考虑它们。
* 某些分类器的优势，朴素贝叶斯(Naive Bayes, NB)超级简单，就像做一些数数的工作。如果条件独立假设成立的话，NB将比鉴别模型（如Logistic回归）收敛的更快，所以你只需要少量的训练数据。即使条件独立假设不成立，NB在实际中仍然表现出惊人的好。如果你想做类似半监督学习，或者是既要模型简单又要性能好，NB值得尝试。
* Logistic回归(Logistic Regression, LR) LR有很多方法来对模型正则化。比起NB的条件独立性假设，LR不需要考虑样本是否是相关的。与决策树与支持向量机（SVM）不同，NB有很好的概率解释，且很容易利用新的训练数据来更新模型（使用在线梯度下降法）。如果你想要一些概率信息（如，为了更容易的调整分类阈值，得到分类的不确定性，得到置信区间），或者希望将来有更多数据时能方便的更新改进模型，LR是值得使用的。
* 决策树（Decision Tree, DT）DT容易理解与解释（对某些人而言——不确定我是否也在他们其中）。DT是非参数的，所以你不需要担心野点（或离群点）和数据是否线性可分的问题（例如，DT可以轻松的处理这种情况：属于A类的样本的特征x取值往往非常小或者非常大，而属于B类的样本的特征x取值在中间范围）。DT的主要缺点是容易过拟合，这也正是随机森林（Random Forest, RF）（或者Boosted树）等集成学习算法被提出来的原因。此外，RF在很多分类问题中经常表现得最好（我个人相信一般比SVM稍好），且速度快可扩展，也不像SVM那样需要调整大量的参数，所以最近RF是一个非常流行的算法。
* 支持向量机（Support Vector Machine, SVM）很高的分类正确率，对过拟合有很好的理论保证，选取合适的核函数，面对特征线性不可分的问题也可以表现得很好。SVM在维数通常很高的文本分类中非常的流行。由于较大的内存需求和繁琐的调参，我认为RF已经开始威胁其地位了。

1. **正确率能很好的评估分类算法吗？分类算法的评估方法？**

AUC/ROC就是用于分类的性能度量标准

* ROC:

横轴FPR：1-TNR，1-Specificity，FPR越大，预测正类中实际负类越多。

纵轴TPR：Sensitivity(正类覆盖率)，TPR越大，预测正类中实际正类越多。

理想目标：TPR=1，FPR=0，即图中(0,1)点，故ROC曲线越靠拢(0,1)点，越偏离45度对角线越好，Sensitivity、Specificity越大效果越好。

AUC (Area Under Curve) 被定义为ROC曲线下的面积，显然这个面积的数值不会大于1。又由于ROC曲线一般都处于y=x这条直线的上方，所以AUC的取值范围一般在0.5和1之间。使用AUC值作为评价标准是因为很多时候ROC曲线并不能清晰的说明哪个分类器的效果更好，而作为一个数值，对应AUC更大的分类器效果更好

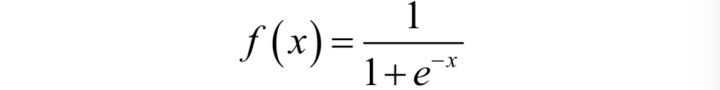
* AUC = 1，是完美分类器，采用这个预测模型时，存在至少一个阈值能得出完美预测。绝大多数预测的场合，不存在完美分类器。
* 0.5 < AUC < 1，优于随机猜测。这个分类器（模型）妥善设定阈值的话，能有预测价值。
* AUC = 0.5，跟随机猜测一样（例：丢铜板），模型没有预测价值。
* AUC < 0.5，比随机猜测还差；但只要总是反预测而行，就优于随机猜测。

1. **理解逻辑回归**

logistic回归由Cox在1958年提出[1]，它的名字虽然叫回归，但这是一种二分类算法，并且是一种线性模型。由于是线性模型，因此在预测时计算简单，在某些大规模分类问题，如广告点击率预估（CTR）上得到了成功的应用。如果你的数据规模巨大，而且要求预测速度非常快，则非线性核的SVM、神经网络等非线性模型已经无法使用，此时logistic回归是你为数不多的选择。

logistic回归源于一个非常朴素的想法：对于二分类问题，能否直接预测出一个样本 属于正样本的概率值？

x的取值范围可以是(−∞ ,+∞ )，现在想想，哪些函数能够将一个(−∞ ,+∞ )之内的实数值变换到区间[0,1]？



机器学习中被广为使用的logistic函数，也叫sigmoid函数，它有一个迷人的性质，单调增，并且定义域是(−∞ ,+∞ )，值域是(0,1)

logistic回归预测的是样本属于某一类的概率，样本的类别标签为离散的1或者0，因此不适合直接用欧氏距离误差来定义损失函数，这里通过最大似然估计来确定参数。

除了梯度下降法这种一阶优化技术，还可以使用牛顿法及其变种，如BFGS算法。

总结

logistic回归是一种二分类算法，它用logistic函数预测出一个样本属于正样本的概率值。预测时，并不需要真的用logistic函数映射，而只需计算一个线性函数，因此是一种线性模型。训练时，采用了最大似然估计，优化的目标函数是一个凸函数，因此能保证收敛到全局最优解。虽然有概率值，但logistic回归是一种判别模型而不是生成模型，因为它并没有假设样本向量x所服从的概率分布，即没有对p(x, y)建模，而是直接预测类后验概率p(y|x)的值。

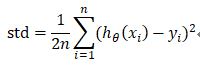
推广到多类

logistic回归只能用于二分类问题，将它进行推广可以得到处理多类分类问题的softmax回归，思路类似，采用指数函数进行变换，然后做归一化。这种变换在神经网络尤其是深度学习中被广为使用，对于多分类问题，神经网络的最后一层往往是softmax层（不考虑损失函数层，它只在训练时使用）。

1. **逻辑回归与朴素贝叶斯有什么区别？**
2. Naive Bayes是一个生成模型，在计算P(y|x)之前，先要从训练数据中计算P(x|y)和P(y)的概率，从而利用贝叶斯公式计算P(y|x)。
3. Logistic Regression是一个判别模型，它通过在训练数据集上最大化判别函数P(y|x)学习得到，不需要知道P(x|y)和P(y)。逻辑斯特回归实际上是用线性回归模型的预测结果去逼近后验概率的逻辑发生比
4. Naive Bayes是建立在条件独立假设基础之上的，设特征X含有n个特征属性（X1，X2，...Xn），那么在给定Y的情况下，X1，X2，...Xn是条件独立的。
5. Logistic Regression的限制则要宽松很多，如果数据满足条件独立假设，Logistic Regression能够取得非常好的效果；当数据不满度条件独立假设时，Logistic Regression,仍然能够通过调整参数让模型最大化的符合数据的分布，从而训练得到在现有数据集下的一个最优模型。
6. 当数据集比较小的时候，应该选用Naive Bayes，为了能够取得很好的效果，数据的需求量为O(log n),当数据集比较大的时候，应该选用Logistic Regression，为了能够取得很好的效果，数据的需求量为O( n),Naive Bayes运用了比较严格的条件独立假设，为了计算P(y|x)，我们可以利用统计的方法统计数据集中P(x|y)和P(y)出现的次数，从而求得P(x|y)和P(y)。因而其所需的数据量要小一些，为O(log n). Logistic Regression在计算时，是在整个参数空间进行线性搜索的，需要的数据集就更大，为O( n).
7. **常见代价函数？**

代价函数是学习模型优化时的目标函数或者准则，通过最小化代价函数来优化模型。

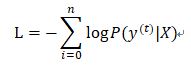
* 均方差代价函数



在线性回归模型里面提出来的

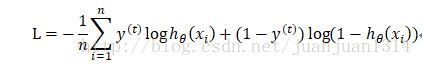
* 对数损失函数

对数似然作为代价函数是在RNN中

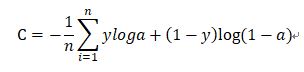


表示真实目标在数据集中的条件概率的负对数。其意义在于，在很多预测目标概率的模型中，将最大概率对应的类型作为输出类型，因此，真实目标的预测概率越高，分类越准确，学习的目标是真实目标的预测概率最大化。而概率是小于1的，其对数值小于0，且对数是单调递增的，因此，当负对数最小化，就等同于对数最大化，概率最大化。

逻辑回归中的代价函数实际上就是对数似然的特殊表示的方式：



* 交叉熵



交叉熵在神经网络中基本都用交叉熵作为代价函数

对数似然函数常用来作为softmax回归的代价函数，

然后输出层神经元是sigmoid函数，可以采用交叉熵代价函数。而深度学习中更普遍的做法是将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

1. **为什么用交叉熵代替二次代价函数**

MSE和sigmoid函数的性质，导致σ′(z)在z取大部分值时会很小（如下图标出来的两端，几近于平坦），这样会使得w和b更新非常慢

交叉熵，当误差大的时候，权重更新就快，当误差小的时候，权重的更新就慢。这是一个很好的性质

1. **什么是损失函数？ 常见的损失函数？**

Loss Function 是定义在单个样本上的，算的是一个样本的误差。

Cost Function 是定义在整个训练集上的，是所有样本误差的平均，也就是损失函数的平均。

Object Function（目标函数 ）定义为：Cost Function + 正则化项。

The loss function computes the error for a single training example; the cost function is the average of the loss funcitons of the entire training set.

1. **逻辑回归为什么使用对数损失函数？**

对数损失在单个数据点上的定义为：

Cost (y,p(y|x)) = −y lnp(y|x)−(1−y)ln(1−p(y|x))

Cost (y,p(y|x))=−y ln⁡p(y|x)−(1−y)ln⁡(1−p(y|x))

全体样本的损失函数则可表达为：

cost(y,p(y|x))=−∑i=1m[yilnp(yi|xi)+(1−yi)ln(1−p(yi|xi))]

cos⁡t(y,p(y|x))=−∑i=1m[yiln⁡p(yi|xi)+(1−yi)ln⁡(1−p(yi|xi))]

可以看到，这个对数损失函数与上面的极大似然估计的对数似然函数本质上是等价的。所以逻辑回归直接采用对数损失函数来求参数，实际上与采用极大似然估计来求参数是一致的。

1. **对数损失函数是如何度量损失的？**

（1）有一类概率型的目标函数，例如逻辑回归来解决二分类问题，假设其目标函数为p(x)，可以简单理解为样本x归属到某一类别的概率。

（2）根据最大似然估计的理论，优化目标是使得P(X)=p(x1)(1-p(x2))p(x3)，最大化（这里假设x1和x3是正例，x2是负例，由于目标函数是求正例的概率，所以1-p(x)自然就是负例的概率）。

（3）乘法表达式求极值比较麻烦，所以最好想办法转化成加法表达式。最自然的想法是两边取对数，把等式右边转化为加法表达式。由于对数单调增，那么求P(X)的最大值的问题，可以转化为求logP(X) 的最大值的问题。

（4）求logP(X)的最大值，其实就是求-logP(X)的最小值。这个-logP(X)其实就是所谓的log loss了。

（5）log loss本身的求解可以使用梯度下降等各种方法。log loss只代表了一个从原始的loss到log形式loss的转化过程。

1. **机器学习中为什么需要梯度下降？梯度下降法缺点？如何对梯度下降法进行调优？**

梯度下降是机器学习中用来求最小值的算法，它被广泛应用于像逻辑回归、线性回归和神经网络的模型中。

梯度下降法是一种迭代型的优化算法，根据初始点在每一次迭代的过程中选择下降法方向，进而改变需要修改的参数，梯度下降法的详细过程如下：

Start at a random point

Repeat:  
 Determine a descent direction

Choose a step size

Update

Until stopping criterion is satisfied

1. **梯度下降、随机梯度下降与批梯度下降算法之间的比较**

这三种算法都用于反向传播的优化损失函数算法。在每轮迭代中更新一次权重w，根据多次迭代，最终无限的靠近我们预期的权重最优值。

* 梯度下降算法：

(1) 如果数据集比较小，完全可以采用全数据集(Full Batch Learning)的形式，采用全数据有两个好处：

a. 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。

b. 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。

(2) 但是梯度下降也存在一下这些缺点：

a. 梯度下降算法并不能保证被优化函数达到全局最优解，只有当损失函数为凸函数时,梯度下降算法才能保证达到全局最优解，因此不能保证一定达到全局最优，受限于损失函数是否为凸函数。

b. 梯度下降算法的另外一个问题是计算时间太长，因为要在全部训练数据上最小化损失，在海量训练数据下，这样是十分耗时。

* 随机梯度下降算法:

为了加速训练的过程，可以使用随机梯度下降算法(stochastic gradient descent-SGD)。随机梯度下降算法也是称为"在线学习"。

(1) 这个算法优化的不是在全部训练数据上的损失函数，而是在每轮迭代中，随机优化某一条训练数据上的损失函数，这样每一轮参数的更新速度大大加快。

(2) 但是带来了如下问题：

在某一条数据上损失函数更小并不代表在全部数据上的损失函数更小，于是使用随机梯度下降优化得到的神经网络甚至可能无法达到全局最优。

* 批梯度下降算法：

为了综合梯度下降算法和随机梯度下降算法的优缺点，在实际应用中一般采用这两个算法的这种----->每次计算一小部分训练数据的损失函数。批梯度下降算法（Mini-batches Learning）在深度学习很多算法的反向传播算法中非常常用。这一小部分训练数据也称为一个batch，因此也引入了batch\_size的概念，batch\_size顾名思义就是来度量每一个batch中实例的个数。

(1) 引入batch有很好的优势：

a. 通过矩阵运算，每次在一个batch上优化神经网络参数并不会比单个数据慢太多。

b. 每次使用一个batch可以大大减小收敛所需要的迭代次数，同时可以使收敛到的结果更加接近梯度下降的效果。

但是批梯度下降算法带来一个问题，那就是如何选取最优的batch\_size?

(2) 可不可以选择一个适中的 Batch\_Size 值呢？

当然可以，这就是批梯度下降法(Mini-batches Learning)。因为如果数据集足够充分，那么用一半(甚至少得多)的数据训练算出来的梯度与用全部数据训练出来的梯度是几乎一样的。

(3) 在合理范围内，增大 Batch\_Size 有何好处？

a. 内存利用率提高了，大矩阵乘法的并行化效率提高。

b. 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。

c. 在一定范围内，一般来说 Batch\_Size 越大，其确定的下降方向越准，引起训练震荡越小。

(4) 盲目增大 Batch\_Size 有何坏处？

a. 内存利用率提高了，但是内存容量可能撑不住了。

b. 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。

c. 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。

1. **决策树**

决策树的基本原理:

决策树（decision tree）是一个树结构（可以是二叉树或非二叉树）。其每个非叶节点表示一个特征属性上的测试，每个分支代表这个特征属性在某个值域上的输出，而每个叶节点存放一个类别。使用决策树进行决策的过程就是从根节点开始，测试待分类项中相应的特征属性，并按照其值选择输出分支，直到到达叶子节点，将叶子节点存放的类别作为决策结果。

决策树模型核心是下面几部分：

* 结点和有向边组成
* 结点有内部结点和叶结点俩种类型
* 内部结点表示一个特征，叶节点表示一个类

决策树的三要素？

决策树学习算法包括三部分：特征选择、树的生成和树的剪枝

决策树算法优缺点:

优点：

得出结果的过程易于理解，比神经网络等黑箱操作好理解，计算量较小，比其他算法计算的较快

缺点：

很容易造成过拟合，需要采用剪枝操作，当类别太多时，会很复杂

剪枝处理的作用及策略？

剪枝是决策树算法对付过拟合的主要手段。因为决策树在划分过程中为了尽可能正确分类训练样本，可能会不断重复，产生过拟合现象。即可能会为了分类正确，将具有某一些属性的样本强行划分到某一类中，而这些样本本不应该属于这一类，只是在这个数据集中属于了这一类，不具有普遍的意义，这样就差生了过拟合。为了降低过拟合的风险可以主动去掉一些分支。剪枝可以分为后剪枝和预剪枝，预剪枝在决策树生成过程中进行，后剪枝在决策树生成完后进行。预剪枝可能产生欠拟合风险，后剪枝效果更好，但是训练时间要长一些。

* 预剪枝（pre-pruning）：预剪枝就是在构造决策树的过程中，先对每个结点在划分前进行估计，若果当前结点的划分不能带来决策树模型泛华性能的提升，则不对当前结点进行划分并且将当前结点标记为叶结点。
* 后剪枝（post-pruning）：后剪枝就是先把整颗决策树构造完毕，然后自底向上的对非叶结点进行考察，若将该结点对应的子树换为叶结点能够带来泛华性能的提升，则把该子树替换为叶结点。

熵的概念以及理解，信息增益的理解

信息熵是度量样本集合纯度最常用的指标。假定当前样本集合D中第k类样本,所占的比例为https://private.codecogs.com/gif.latex?p_k%28k%3D1%2C2%2C%5Ccdots%20%2C%7Cy%7C%29,则D的信息熵为

Ent(D)=-\sum_{k=1}^{|y|}p_klog_2p_k

https://private.codecogs.com/gif.latex?Ent%28D%20%29，的值越小，则D的纯度越高

假定离散属性a有V个可能的取值\left \{ a^1,a^2,\cdots ,a^V \right \},若使用a来对样本集D进行划分，则会产生V个分支节点，其中第个分支节点包含了D中所有在属性a上取值a^v为的样本D^v，记为.根据（1）式考虑到不同的分支节点包含的样本数不同，可得用属性a对样本集D进行划分所获得的信息增益为：

Gain(D,a)=Ent(D)-\sum_{v=1}^{V}{}\frac{D^v}{D}Ent(D^v)

一般而言，信息增益越大，说明使用属性a来进行划分所获得的纯度提升越大，因此所找的属性为a^*=\underset{a\in A}{argmax}Gain(D,a)

ID3算法就是采用信息增益作为特征选择的标准

在上边所介绍的信息增益准则，它对可取值较多的属性有所偏好，为减少这种偏好所带来的不利影响，C4.5决策树算法不直接使用信息增益，而是使用增益率来选择最优划分属性。信息增益率定义为：

Gain\_ratio(D,a)=\frac{Gain(D,a)}{IV(a)}

其中

IV(a)=-\sum_{v=1}^{V}\frac{|D^v|}{|D|}log_2\frac{|D^v|}{|D|}

称为属性a的固有值，属性a的可能取值数目越多，则IV(a)的值通常会越大。

1. **支持向量机**

支持向量机（support vector machines, SVM）是一种二分类模型，它的基本模型是定义在特征空间上的间隔最大的线性分类器，间隔最大使它有别于感知机；SVM还包括核技巧，这使它成为实质上的非线性分类器。SVM的的学习策略就是间隔最大化，可形式化为一个求解凸二次规划的问题，也等价于正则化的合页损失函数的最小化问题。SVM的的学习算法就是求解凸二次规划的最优化算法

* 支持向量机解决的问题？

SVM本身是一个二值分类器

* 核函数作用？常见的核函数有哪些？

非线性映射是SVM方法的理论基础,SVM利用内积核函数代替向高维空间的非线性映射,性核函数：线性核，主要用于线性可分的情况，我们可以看到特征空间到输入空间的维度是一样的，其参数少速度快，对于线性可分数据，其分类效果很理想，因此我们通常首先尝试用线性核函数来做分类，看看效果如何，如果不行再换别的

多项式核函数：多项式核函数可以实现将低维的输入空间映射到高纬的特征空间，但是多项式核函数的参数多，当多项式的阶数比较高的时候，核矩阵的元素值将趋于无穷大或者无穷小，计算复杂度会大到无法计算。

高斯（RBF）核函数：高斯径向基函数是一种局部性强的核函数，其可以将一个样本映射到一个更高维的空间内，该核函数是应用最广的一个，无论大样本还是小样本都有比较好的性能，而且其相对于多项式核函数参数要少，因此大多数情况下在不知道用什么核函数的时候，优先使用高斯核函数。

* 如果特征的数量大到和样本数量差不多，则选用LR或者线性核的SVM；
* 如果特征的数量小，样本的数量正常，则选用SVM+高斯核函数；
* 如果特征的数量小，而样本的数量很大，则需要手工添加一些特征从而变成第一种情况。
* SVM主要特点及缺点？

SVM有如下主要几个特点：

1. 非线性映射是SVM方法的理论基础,SVM利用内积核函数代替向高维空间的非线性映射；
2. 对特征空间划分的最优超平面是SVM的目标,最大化分类边际的思想是SVM方法的核心；
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小样本学习方法。它基本上不涉及概率测度及大数定律等,因此不同于现有的统计方法。从本质上看,它避开了从归纳到演绎的传统过程,实现了高效的从训练样本到预报样本的“转导推理”,大大简化了通常的分类和回归等问题。
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. 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。
7. 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。
8. SVM在小样本训练集上能够得到比其它算法好很多的结果。支持向量机之所以成为目前最常用，效果最好的分类器之一，在于其优秀的泛化能力，这是是因为其本身的优化目标是结构化风险最小，而不是经验风险最小，因此，通过margin的概念，得到对数据分布的结构化描述，因此减低了对数据规模和数据分布的要求。SVM也并不是在任何场景都比其他算法好，对于每种应用，最好尝试多种算法，然后评估结果。如SVM在邮件分类上，还不如逻辑回归、KNN、bayes的效果好。
9. 它基于结构风险最小化原则，这样就避免了过学习问题，泛化能力强
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。
11. 泛华错误率低，分类速度快，结果易解释

不足之处：

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，而是使用了简单的naive bayes分类器，或者是使用逻辑回归模型分类。

1. 用SVM解决多分类问题存在困难

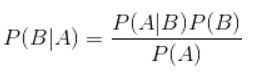
经典的支持向量机算法只给出了二类分类的算法，而在数据挖掘的实际应用中，一般要解决多类的分类问题。可以通过多个二类支持向量机的组合来解决。主要有一对多组合模式、一对一组合模式和SVM决策树；再就是通过构造多个分类器的组合来解决。主要原理是克服SVM固有的缺点，结合其他算法的优势，解决多类问题的分类精度。如：与粗集理论结合，形成一种优势互补的多类问题的组合分类器。

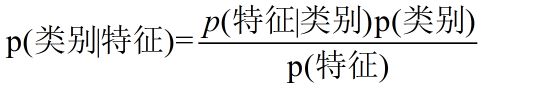
1. 对缺失数据敏感，对参数和核函数的选择敏感

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.

1. **贝叶斯**

贝叶斯分类是一类分类算法的总称，这类算法均以贝叶斯定理为基础，故统称为贝叶斯分类。而朴素朴素贝叶斯分类是贝叶斯分类中最简单，也是常见的一种分类方法。



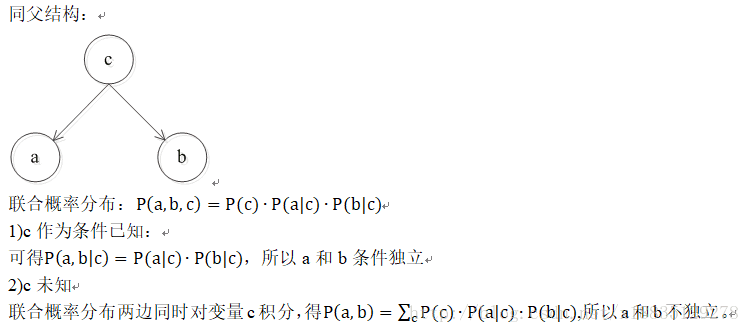


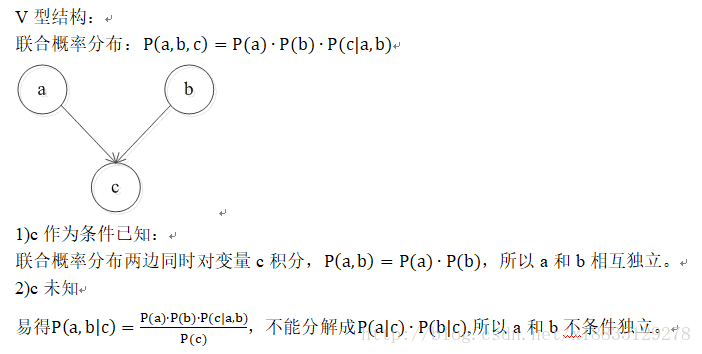
朴素贝叶斯分类有朴素一词的来源，朴素贝叶斯算法是假设各个特征之间相互独立

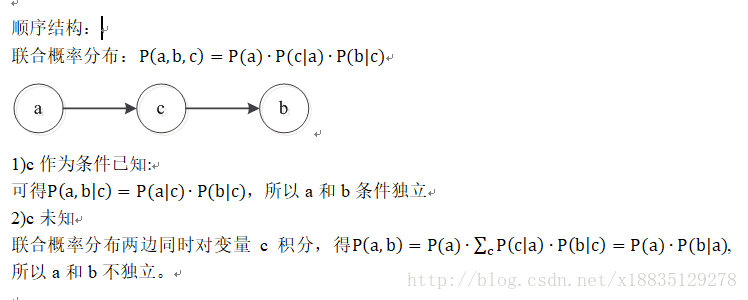
* 朴素与半朴素贝叶斯分类器

朴素贝叶斯分类器的一个重要假定：分类对应的各个属性间是相互独立的，然而在现实应用中，这个往往难以做到，那怎么办呢？很简单，适当考虑一部分属性间的相互依赖关系，这种放松后的分类称为半朴素贝叶斯分类，其中最常用的策略：假定每个属性仅依赖于其他最多一个属性，称其依赖的这个属性为其超父属性，这种关系称为：独依赖估计（ODE）

* 贝叶斯网三种典型结构







1. **前向传播与反向传播？**

反向传播法（BackPropagation），就是一个链式求导法则反复用

1. **什么是超参数？ 如何寻找超参数的最优值？ 超参数搜索一般过程？**

基础概念

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，在机器学习过程中需要对超参数进行优化，给学习器选择一组最优超参数，以提高学习的性能和效果。比如，树的数量或树的深度，学习率（多种模式）以及k均值聚类中的簇数等都是超参数。

与超参数区别的概念是参数，它是模型训练过程中学习到的一部分，比如回归系数，神经网络权重等。简单的描述参数是模型训练获得的，超参数是人工配置参数（本质上是参数的参数，每次改变超参数，模型都要重新训练）。

调参问题

在深度神经网络中，超参数的调整是一项必备技能，通过观察在训练过程中的监测指标如损失loss和准确率来判断当前模型处于什么样的训练状态，及时调整超参数以更科学地训练模型能够提高资源利用率。在本研究中使用了以下超参数，下面将分别介绍并总结了不同超参数的调整规则。

（1）学习率

学习率（learning rate或作lr）是指在优化算法中更新网络权重的幅度大小。学习率可以是恒定的、逐渐降低的，基于动量的或者是自适应的。不同的优化算法决定不同的学习率。当学习率过大则可能导致模型不收敛，损失loss不断上下震荡；学习率过小则导致模型收敛速度偏慢，需要更长的时间训练。通常lr取值为[0.01,0.001,0.0001]

（2）批次大小batch\_size

批次大小是每一次训练神经网络送入模型的样本数，在卷积神经网络中，大批次通常可使网络更快收敛，但由于内存资源的限制，批次过大可能会导致内存不够用或程序内核崩溃。bath\_size通常取值为[16,32,64,128]

（3）优化器optimizer

目前Adam是快速收敛且常被使用的优化器。随机梯度下降(SGD)虽然收敛偏慢，但是加入动量Momentum可加快收敛，同时带动量的随机梯度下降算法有更好的最优解，即模型收敛后会有更高的准确性。通常若追求速度则用Adam更多。

（4）迭代次数

迭代次数是指整个训练集输入到神经网络进行训练的次数，当测试错误率和训练错误率相差较小时，可认为当前迭代次数合适；当测试错误率先变小后变大时则说明迭代次数过大了，需要减小迭代次数，否则容易出现过拟合。

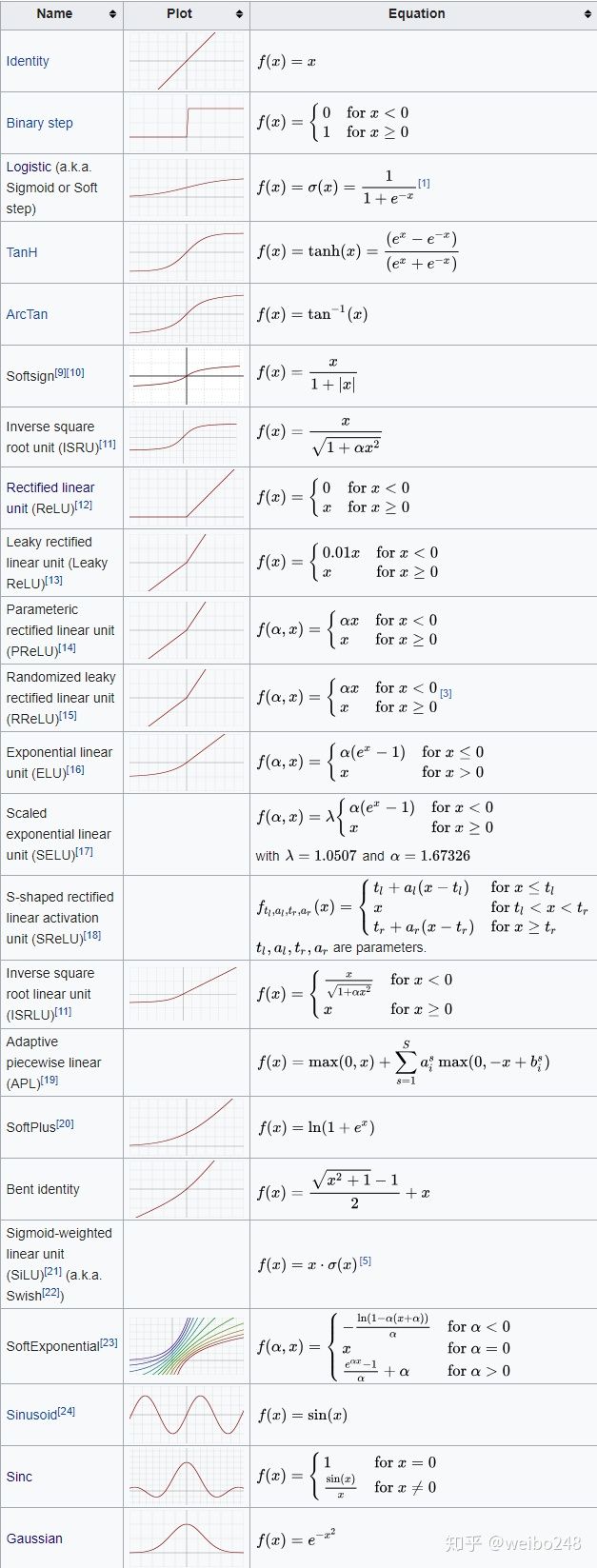
（5）激活函数

在神经网络中，激活函数不是真的去激活什么，而是用激活函数给神经网络加入一些非线性因素，使得网络可以更好地解决较为复杂的问题。比如有些问题是线性可分的，而现实场景中更多问题不是线性可分的，若不使用激活函数则难以拟合非线性问题，测试时会有低准确率。所以激活函数主要是非线性的，如sigmoid、tanh、relu。sigmoid函数通常用于二分类，但要防止梯度消失，故适合浅层神经网络且需要配备较小的初始化权重，tanh函数具有中心对称性，适合于有对称性的二分类。在深度学习中，relu是使用最多的激活函数，简单又避免了梯度消失。

常见超参数搜索算法：

* 网格搜索
* 随机搜索
* 启发式搜索

1. **常见的激活函数及图像? 常见激活函数的导数计算？**



1. **如何选择激活函数？**

现在我们已经了解了这么多的激活函数，接下来就需要分析在哪种情况下应该使用哪种激活函数了。激活函数好或坏，不能凭感觉定论。然而，根据问题的性质，我们可以为神经网络更快更方便地收敛作出更好的选择。

用于分类器时，Sigmoid函数及其组合通常效果更好。

由于梯度消失问题，有时要避免使用sigmoid和tanh函数。

ReLU函数是一个通用的激活函数，目前在大多数情况下使用。

如果神经网络中出现死神经元，那么PReLU函数就是最好的选择。

请记住，ReLU函数只能在隐藏层中使用。

一点经验：你可以从ReLU函数开始，如果ReLU函数没有提供最优结果，再尝试其他激活函数。

1. **使用ReLu激活函数的优点？**

修正线性单元(Rectified linear unit，ReLU）

ReLU有以下优势：

* 对于线性函数而言，ReLU的表达能力更强，尤其体现在深度网络中；
* 而对于非线性函数而言，ReLU由于非负区间的梯度为常数，因此不存在梯度消失问题(Vanishing Gradient Problem)，使得模型的收敛速度维持在一个稳定状态。

1. **Softmax函数如何应用于多分类?**

Sigmoid + cross-entropy follows the Bernoulli distribution, while softmax + log-likelihood follows the multinomial distribution with one observation (which is a multiclass version of the Bernoulli).

For binary classification problems, the softmax function outputs two values (between 0 and 1 and sum up to 1), to represent the probabilities of each class.

While the sigmoid function outputs one value between 0 and 1, to represent the probability of one class (so the probability of the other class is just 1-p).

Sigmoid+互信息输出结果是伯努利分布（注：P(y\_1|X), P(y\_2|X),...,P(y\_n|X)）

而Softmax输出的是多项分布（注：P(y\_1, y\_2,...,y\_n|X)）

对于二值分类问题，Softmax输出两个值，这两个值相加为1

对于Sigmoid来说，也输出两个值，不过没有可加性，两个值各自是0到1的某个数，对于一个值p来说，1-p是它对应的另一个概率。

例如：如果我们预测某个东西是或者不是，那么我们可以这样：

输出(0, 1)代表“是”，输出(1, 0)代表“否”

Softmax可能输出(0.3, 0.7)，代表算法认为“是”的概率是0.7，“否”的概率是0.3，相加为1

Sigmoid的输出可能是(0.4, 0.8)，它们相加不为1，解释来说就是Sigmoid认为输出第一位为1的概率是0.4，第一位不为1的概率是0.6（1-p），第二位为1的概率是0.8，第二位不为1的概率是0.2。

Geoff Hinton covered exactly this topic in his coursera course on neural nets. The problem with sigmoids is that as you reach saturation (values get close to 1 or 0), the gradients vanish. This is detrimental to optimization speed. Softmax doesn't have this problem, and in fact if you combine softmax with a cross entropy error function the gradients are just (z-y), as they would be for a linear output with least squares error.

nkorslund ( https://www.reddit.com/r/MachineLearning/comments/32iyt9/question\_comparison\_between\_softmax\_and\_sigmoid/ )

这个回答提到Hinton在coursera的课提到这个课题了，很可惜我没上过这门课（不过这门课正在准备2016年9月份重开，https://www.coursera.org ）。Hinton认为当Sigmoid函数的某个输出接近1或者0的时候，就会产生梯度消失，严重影响优化速度，而Softmax没有这个问题。

1. **Batch\_Size**

Batch\_Size（批尺寸）是机器学习中一个重要参数

批梯度下降法（Mini-batches Learning）。因为如果数据集足够充分，那么用一半（甚至少得多）的数据训练算出来的梯度与用全部数据训练出来的梯度是几乎一样的。

* 在合理范围内，增大 Batch\_Size 有何好处？

内存利用率提高了，大矩阵乘法的并行化效率提高。

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。

在一定范围内，一般来说 Batch\_Size 越大，其确定的下降方向越准，引起训练震荡越小。

* 盲目增大 Batch\_Size 有何坏处？

内存利用率提高了，但是内存容量可能撑不住了。

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。

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。

* 运行结果与上文分析相印证：

Batch\_Size 太小，算法在 200 epoches 内不收敛。

随着 Batch\_Size 增大，处理相同数据量的速度越快。

随着 Batch\_Size 增大，达到相同精度所需要的 epoch 数量越来越多。

由于上述两种因素的矛盾， Batch\_Size 增大到某个时候，达到时间上的最优。

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。

既然 Full Batch Learning 并不适用大数据集，那么走向另一个极端怎么样？

所谓另一个极端，就是每次只训练一个样本，即 Batch\_Size = 1。这就是在线学习（Online Learning）。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。使用在线学习，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

1. **归一化? 归一化含义？为什么要归一化? 为什么归一化能提高求解最优解速度？**

* 数据归一化/标准化（二者还是有所区别的）是机器学习/数据挖掘的一项基础工作，是数据预处理的重要一步。样本各个特征往往具有不同的分布/取值范围，通过归一化将各个维度的特征值映射到相同区间，使得各特征值具有相同量纲，处于同一数量级

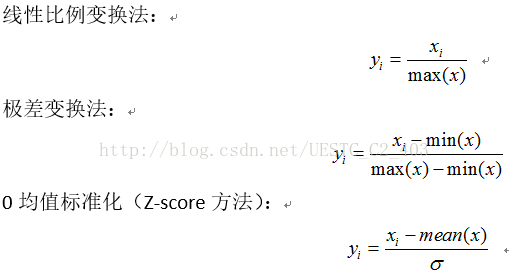
如果本来各维的量纲是相同的，最好不要做归一化，以尽可能多地保留信息。

如果本来各维的量纲是不同的，就需要先对各维分别归一化。

* 归一化是机器学习中必不可少的一步，只要样本各维量纲不一致，我们就需要对数据集进行归一化，不管训练什么机器学习模型，最好都先对数据集进行归一化，这是因为，归一化不仅可以提高模型正确率（针对不具有伸缩不变性的机器学习算法），还可以加快模型收敛，提升训练速度。

于是损失函数的等高线类似椭圆形状，最优解的寻优过程如下图所示，可以发现梯度下降方向只要不在椭圆对称轴上，这一步损失函数几乎就不会减小，这会导致震荡，使得收敛很慢，甚至不收敛。

此时损失函数等高线类似圆形形状，最优解的寻优过程如下图所示，可以发现归一化后，寻优过程变得平滑，收敛速度得到提高，加快模型训练速度。



1. **Weight Normalization和Batch Normalization**

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minbatch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，minibatch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。

normalization按大类分可以分三类，一是对参数做归一化，二是对输出特征做归一化。归一化的手段可以多种多样，最基本的是instance级别的 l2 norm，也可以是batchnorm，也可以是其他的。bn是使用batchnorm手段的 特征归一化。目前是训练深度网络的必备利器ln是使用一种类似bn的方法对特征归一化，在rnn模型里效果较好wn是使用l2 norm的参数归一化此外，还有一种cosine norm的方法，同时对参数和特征做l2 norm归一化，似乎效果也不错最后，对参数做batchnorm不知道会怎么样

1. **为什么无监督预训练可以帮助深度学习？**

最近深度学习框架中比如：Deep Belief Networks，Stacks of Auto-Encoder variants. 都基本基于先通过无监督的预训练，然后再通过有监督的fine tune，达到了最好的效果。

深度网络存在以下缺点：

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。（深层网络意味着特征比较多，机器学习里面临多特征：1.多样本 2.规则化 3.特征选择）

2. 多层神经网络参数优化是个高阶非凸优化问题，常收敛较差的局部解

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个网络参数的初始值。  无监督学习--->参数初始值；监督学习--->fine-tuning，即训练有标注样本。经过预训练最终能得到比较好的局部最优解。

我们为什么要做这种无监督预训练？反向传播(back-propgation)在多层次模型上到底有什么问题呢？其实反向传播算法的问题在于当我们对每一层做梯度计算(gradient computation)的时候，随着层次的深度，它的值会变得越来越小，我们称之为梯度消失问题(gradient vanishing problem). 所以，在一般情况下，当我们对深度模型的底层计算梯度的时候它的值已经变得非常小(或已接近于0)。类似的问题在训练RNN(Recurrent Neural Network)的时候也会存在，我们会在后面给予介绍。当然，也很容易想到，虽然梯度在底层上的变化很小，但如果有“足够多”的数据，我们也可以在无预训练的情况下训练出有效的模型。

1. **权重偏差初始化，全都初始化为0，全都初始化为同样的值，初始化为小的随机数**

1） 全部初始化为零；

但是将权重w全部初始化为零，那么每一层所学到的参数都是一样的，因为它们的梯度一样，所以在反向传播的过程中，每一层的神经元也是相同的。因此会导致代价函数在开始的一段时间内，明显下降，但是一段时间以后，停止继续下降

2） 初始化为相同的随机数

其实将权重w初始化为相同的随机数和全部初始化为零是一样的，都会导致同样的问题

3） 初始化为较小的随机数；

权重参数随机初始化为服从均值为零和方差为1的高斯分布函数，开始模型可以很好的运行一段时间，但是随着时间增加，前向传递时，方差开始减少，梯度也开始向零靠近，会导致Gradient Vanishing。特别地，当激活函数为sigmoid时，梯度接近0.5；当激活函数为时tanh，梯度接近0。

4） 初始化为较大的随机数

反向传播时，倒数趋于零，梯度也会消失。此外，权重较大且当输入也很大时，如果使用sigmoid做激活函数，会使输出趋向于0和1，会导致更多问题

1. **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

1. **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。

1. **初始化偏差**

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.

1. **理解One Hot Encodeing原理及作用？**

数据预处理之独热编码（One-Hot Encoding）

独热编码即 One-Hot 编码，又称一位有效编码，其方法是使用N位状态寄存器来对N个状态进行编码，每个状态都由他独立的寄存器位，并且在任意时候，其中只有一位有效

为什么使用one-hot编码来处理离散型特征?

1. 使用one-hot编码，将离散特征的取值扩展到了欧式空间，离散特征的某个取值就对应欧式空间的某个点。
2. 将离散特征通过one-hot编码映射到欧式空间，是因为，在回归，分类，聚类等机器学习算法中，特征之间距离的计算或相似度的计算是非常重要的，而我们常用的距离或相似度的计算都是在欧式空间的相似度计算，计算余弦相似性，基于的就是欧式空间。
3. 将离散型特征使用one-hot编码，确实会让特征之间的距离计算更加合理。比如，有一个离散型特征，代表工作类型，该离散型特征，共有三个取值，不使用one-hot编码，其表示分别是x\_1 = (1), x\_2 = (2), x\_3 = (3)。两个工作之间的距离是，(x\_1, x\_2) = 1, d(x\_2, x\_3) = 1, d(x\_1, x\_3) = 2。那么x\_1和x\_3工作之间就越不相似吗？显然这样的表示，计算出来的特征的距离是不合理。那如果使用one-hot编码，则得到x\_1 = (1, 0, 0), x\_2 = (0, 1, 0), x\_3 = (0, 0, 1)，那么两个工作之间的距离就都是sqrt(2).即每两个工作之间的距离是一样的，显得更合理,对离散型特征进行one-hot编码是为了让距离的计算显得更加合理。
4. 将离散型特征进行one-hot编码的作用，是为了让距离计算更合理，但如果特征是离散的，并且不用one-hot编码就可以很合理的计算出距离，那么就没必要进行one-hot编码，比如，该离散特征共有1000个取值，我们分成两组，分别是400和600,两个小组之间的距离有合适的定义，组内的距离也有合适的定义，那就没必要用one-hot 编码
5. 离散特征进行one-hot编码后，编码后的特征，其实每一维度的特征都可以看做是连续的特征。就可以跟对连续型特征的归一化方法一样，对每一维特征进行归一化。比如归一化到[-1,1]或归一化到均值为0,方差为1
6. **如何做归一化**

处理离散型特征和连续型特征并存的情况，如何做归一化：

* 需要归一化情况：

拿到获取的原始特征，必须对每一特征分别进行归一化，比如，特征A的取值范围是[-1000,1000]，特征B的取值范围是[-1,1].

如果使用logistic回归，w1\*x1+w2\*x2，因为x1的取值太大了，所以x2基本起不了作用。

所以，必须进行特征的归一化，每个特征都单独进行归一化。

* 连续型特征归一化的常用方法：

Rescale bounded continuous features: All continuous input that are bounded, rescale them to [-1, 1] through x = (2x - max - min)/(max - min).线性放缩到[-1,1]

Standardize all continuous features: All continuous input should be standardized and by this I mean, for every continuous feature, compute its mean (u) and standard deviation (s) and do x = (x - u)/s.放缩到均值为0，方差为1

* 有些情况不需要进行特征的归一化：

基于树的方法是不需要进行特征的归一化，例如随机森林，bagging 和 boosting等。

基于参数的模型或基于距离的模型，都是要进行特征的归一化。

1. **常用的优化器有哪些 ?**

首先来看一下梯度下降最常见的三种变形 BGD，SGD，MBGD，这三种形式的区别就是取决于我们用多少数据来计算目标函数的梯度，这样的话自然就涉及到一个 trade－off，即参数更新的准确率和运行时间

SGD with Momentum

如果数据是稀疏的，就用自适用方法，即 Adagrad, Adadelta, RMSprop, Adam。

RMSprop, Adadelta, Adam 在很多情况下的效果是相似的。

Adam 就是在 RMSprop 的基础上加了 bias-correction 和 momentum，

随着梯度变的稀疏，Adam 比 RMSprop 效果会好。

整体来讲，Adam 是最好的选择。

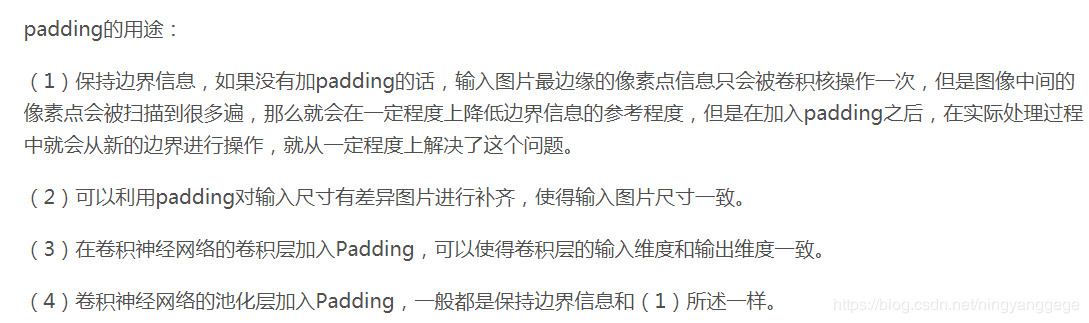
很多论文里都会用 SGD，没有 momentum 等。SGD 虽然能达到极小值，但是比其它算法用的时间长，而且可能会被困在鞍点。

如果需要更快的收敛，或者是训练更深更复杂的神经网络，需要用一种自适应的算法。

1. **Dropout 系列问题 , dropout率的选择**

* 数据量小的时候，dropout效果不好，数据量大了，dropout效果好
* 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。
* 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）

1. **Padding 系列问题**





1. **卷积神经网络的组成层**

卷积神经网络分成三个主要部分，分别为：卷积层（convolution layer），池化层（pooling layer） 和 全连接层 （fully connected layer）

1. **卷积层和池化层有什么区别？**

* 卷积层：用来进行特征的提取
* 池化层：对输入的特征图进行压缩，一方面使特征图变小，简化网络计算复杂度；一方面进行特征压缩，提取主要特征

在卷积神经网络中，我们经常会碰到池化操作，而池化层往往在卷积层后面，通过池化来降低卷积层输出的特征向量，同时改善结果（不易出现过拟合）

最常见的池化操作为平均池化mean pooling和最大池化max pooling：

平均池化：计算图像区域的平均值作为该区域池化后的值。

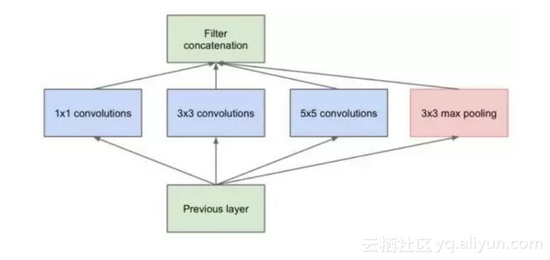
最大池化：选图像区域的最大值作为该区域池化后的值。

1. **卷积核一定越大越好？**

AlexNet中用到了一些非常大的卷积核，比如11×11、5×5卷积核，之前人们的观念是，卷积核越大，receptive field（感受野）越大，看到的图片信息越多，因此获得的特征越好。虽说如此，但是大的卷积核会导致计算量的暴增，不利于模型深度的增加，计算性能也会降低。于是在VGG（最早使用）、Inception网络中，利用2个3×3卷积核的组合比1个5×5卷积核的效果更佳，同时参数量（3×3×2+1 VS 5×5×1+1）被降低，因此后来3×3卷积核被广泛应用在各种模型中。

1. **每层卷积只能用一种尺寸的卷积核？– Inception结构**

传统的层叠式网络，基本上都是一个个卷积层的堆叠，每层只用一个尺寸的卷积核，例如VGG结构中使用了大量的3×3卷积层。事实上，同一层feature map可以分别使用多个不同尺寸的卷积核，以获得不同尺度的特征，再把这些特征结合起来，得到的特征往往比使用单一卷积核的要好，谷歌的GoogLeNet，或者说Inception系列的网络，就使用了多个卷积核的结构：



如上图所示，一个输入的feature map分别同时经过1×1、3×3、5×5的卷积核的处理，得出的特征再组合起来，获得更佳的特征。但这个结构会存在一个严重的问题：参数量比单个卷积核要多很多，如此庞大的计算量会使得模型效率低下。

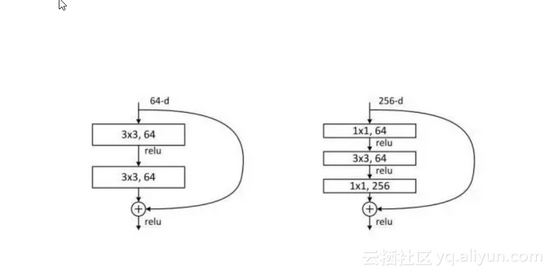
1. **怎样才能减少卷积层参数量？**

1×1卷积核也被认为是影响深远的操作，往后大型的网络为了降低参数量都会应用上1×1卷积核

256维的输入直接经过一个3×3×256的卷积层，输出一个256维的feature map，那么参数量为：256×3×3×256 = 589,824

256维的输入先经过一个1×1×64的卷积层，再经过一个3×3×64的卷积层，最后经过一个1×1×256的卷积层，输出256维，参数量为：256×1×1×64 + 64×3×3×64 + 64×1×1×256 = 69,632。足足把第一种操作的参数量降低到九分之一.

1. **越深的网络就越难训练吗？– Resnet残差网络**



ResNet skip connection

传统的卷积层层叠网络会遇到一个问题，当层数加深时，网络的表现越来越差，很大程度上的原因是因为当层数加深时，梯度消散得越来越严重，以至于反向传播很难训练到浅层的网络。为了解决这个问题，何凯明大神想出了一个“残差网络”，使得梯度更容易地流动到浅层的网络当中去，而且这种“skip connection”能带来更多的好处，

1. **卷积操作时必须同时考虑通道和区域吗？**

Xception网络就是基于以上的问题发明而来。我们首先对每一个通道进行各自的卷积操作，有多少个通道就有多少个过滤器。得到新的通道feature maps之后，这时再对这批新的通道feature maps进行标准的1×1跨通道卷积操作。这种操作被称为 “ DepthWise convolution”缩写“DW”。

这种操作是相当有效的，在imagenet 1000类分类任务中已经超过了InceptionV3的表现，而且也同时减少了大量的参数，我们来算一算，假设输入通道数为3，要求输出通道数为256，两种做法：

1.直接接一个3×3×256的卷积核，参数量为：3×3×3×256 = 6,912

2.DW操作，分两步完成，参数量为：3×3×3 + 3×1×1×256 = 795，又把参数量降低到九分之一！

因此，一个depthwise操作比标准的卷积操作降低不少的参数量，同时论文中指出这个模型得到了更好的分类效果。

1. **窄卷积和宽卷积，采用宽卷积的好处有什么？**

矩阵的中部使用3x3的滤波器没有问题，在矩阵的边缘该怎么办呢？左上角的元素没有顶部和左侧相邻的元素，该如何滤波呢？解决的办法是采用补零法（zero-padding）。所有落在矩阵范围之外的元素值都默认为0。这样就可以对输入矩阵的每一个元素做滤波了，输出一个同样大小或是更大的矩阵。补零法又被称为是宽卷积，不使用补零的方法则被称为窄卷积。

当滤波器长度相对输入向量的长度较大时，你会发现宽卷积很有用，或者说很有必要。

在tensorflow的实现中，conv2d的padding=‘SAME',默认的就是宽卷积的方式，会进行补零。padding = ’VALID'就是窄卷积的方式。

但是存在两个缺点：

1.卷积后的矩阵越变越小（如果卷积层100层，每一层都缩小最终得到的将是很小的图片）

2.输入矩阵（左）边缘像素（绿阴影）只被计算过一次，而中间像素（红阴影）被卷积计算多次，意味着丢失图像角落信息。

为了解决这两个问题，就对输入图像进行padding，即填充像素

1. **如何得到卷积层输出的深度？**

卷积层输出的深度==卷积核的个数

1. **激活函数通常放在卷积神经网络的那个操作之后？**

全连接层之后i

model.add(keras.layers.Flatten())

model.add(keras.layers.Dense(500, activation='relu'))

model.add(keras.layers.Dense(10, activation='softmax'))

1. **如何理解最大池化层有几分缩小？**

主要会做降采样（downsampling）

池化操作一般由两种，一种是Ava Pooling ，一种max Pooling

同样地采用一个2\*2的filter,max pooling是在每一个区域中寻找最大值，这里的stride=2,最终在原特征图中提取主要特征得到右图。

（Avy pooling现在不怎么用了，方法是对每一个2\*2的区域元素求和，再除以4，得到主要特征），而一般的filter取2\*2,最大取3\*3,stride取2，压缩为原来的1/4.

注意：这里的pooling操作是特征图缩小，有可能影响网络的准确度，因此可以通过增加特征图的深度来弥补（这里的深度变为原来的2倍）

1. **不同卷积后图像大小计算？**

像素宽度：W（Width）

填充大小：P（Padding）

卷积核大小：K（Kernel-size）

步长大小：S（stride）

卷积后所得feature map尺寸大小计算公式如下：

W_{n+1}=\left (W_{n}+2*P-K \right )/S+1

补充：

1.Padding的作用用于解决图像边缘信息损失的问题；

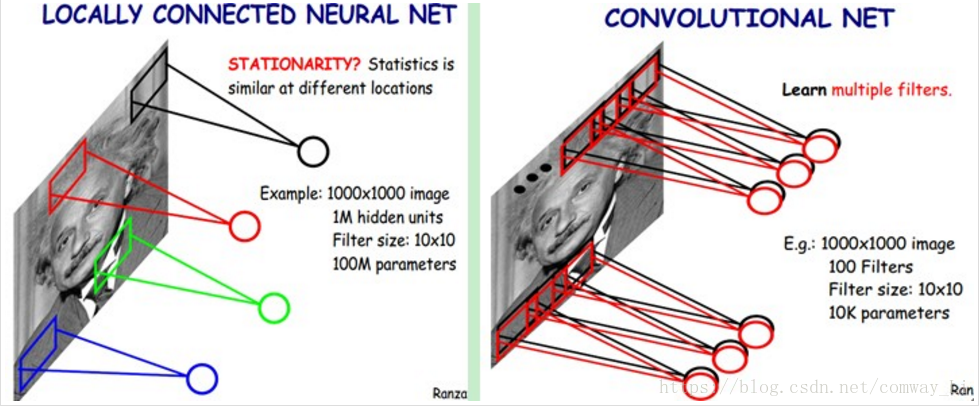
2.计算卷积后map尺寸时若不为整数则向下取整，而计算pooling后尺寸时则向上取整。

1. **局部连接 && 权值共享**

所谓的权值共享就是说，给一张输入图片，用一个filter去扫这张图, filter里面的数就叫权重，这张图每个位置是被同样的filter扫的，所以权重是一样的，也就是共享。

共享的角度来说：权重共享即filter的值共享

每个神经元仅与输入神经元的一块区域连接，这块局部区域称作感受野（receptive field）。在图像卷积操作中，即神经元在空间维度（spatial dimension，即上图示例H和W所在的平面）是局部连接。



1. **如何解决训练样本少的问题**

* 首先，数据量不够，你就把1个当8、9、10······个用，裁剪裁剪，变换变换，翻转翻转。
* 其次，学习特征不够，人工先验来凑。
* 人工增加训练集的大小. 通过平移, 翻转, 加噪声等方法从已有数据中创造出一批"新"的数据.
* Regularization: 数据量比较小会导致模型过拟合, 使得训练误差很小而测试误差特别大. 通过在Loss Function 后面加上正则项可以抑制过拟合的产生. 缺点是引入了一个需要手动调整的hyper-parameter.

这也是一种正则化手段. 不过跟以上不同的是它通过随机将部分神经元的输出置零来实现.

* Unsupervised Pre-training. 用Auto-Encoder或者RBM的卷积形式

一层一层地做无监督预训练,最后加上分类层做有监督的Fine-Tuning.

1. **什么样的样本集不适合用深度学习?**

* 数据集太小，数据样本不足时，深度学习相对其它机器学习算法，没有明显优势；
* 数据集没有局部相关特性，目前深度学习表现比较好的领域主要是图像／语音／自然语言处理等领域，这些领域的一个共性是局部相关性。图像中像素组成物体，语音信号中音位组合成单词，文本数据中单词组合成句子，这些特征元素的组合一旦被打乱，表示的含义同时也被改变。对于没有这样的局部相关性的数据集，不适于使用深度学习算法进行处理。

1. **对所有优化问题来说,有没有可能找到比已知算法更好的算法?**

没有免费的午餐定理：

对于训练样本（黑点），不同的算法A/B在不同的测试样本（白点）中有不同的表现，这表示：对于一个学习算法A，若它在某些问题上比学习算法 B更好，则必然存在一些问题，在那里B比A好。 也就是说：对于所有问题，无论学习算法A多聪明，学习算法 B多笨拙，它们的期望性能相同。

但是：没有免费午餐定理假设所有问题出现几率相同，实际应用中，不同的场景，会有不同的问题分布，所以，在优化算法时，针对具体问题进行分析，是算法优化的核心所在

1. **何为共线性, 跟过拟合有啥关联?**

共线性：多变量线性回归中，变量之间由于存在高度相关关系而使回归估计不准确。

共线性会造成冗余，导致过拟合。

解决方法：排除变量的相关性／加入权重正则。

1. **广义线性模型是怎被应用在深度学习中?**

机器学习中常见的广义线性模型（GLM）：这种模型是把自变量的线性预测函数当作因变量的估计值。在机器学习中，有很多模型都是基于广义线性模型的，比如传统的线性回归模型，最大熵

A Statistical View of Deep Learning (I): Recursive GLMs

深度学习从统计学角度，可以看做递归的广义线性模型。

广义线性模型相对于经典的线性模型(y=wx+b)，核心在于引入了连接函数g(.)，形式变为：y=g−1(wx+b)。

深度学习时递归的广义线性模型，神经元的激活函数，即为广义线性模型的链接函数。逻辑回归（广义线性模型的一种）的Logistic函数即为神经元激活函数中的Sigmoid函数，很多类似的方法在统计学和神经网络中的名称不一样，容易引起初学者（这里主要指我）的困惑。

1. **造成梯度消失的原因?**

神经网络的训练中，通过改变神经元的权重，使网络的输出值尽可能逼近标签以降低误差值，训练普遍使用BP算法，核心思想是，计算出输出与标签间的损失函数值，然后计算其相对于每个神经元的梯度，进行权值的迭代。

梯度消失会造成权值更新缓慢，模型训练难度增加。造成梯度消失的一个原因是，许多激活函数将输出值挤压在很小的区间内，在激活函数两端较大范围的定义域内梯度为0。造成学习停止

1. **权值初始化方法有哪些**

权值初始化的方法主要有：常量初始化（constant）、高斯分布初始化（gaussian）、positive\_unitball初始化、均匀分布初始化（uniform）、xavier初始化、msra初始化、双线性初始化（bilinear）

1. **启发式优化算法中，如何避免陷入局部最优解？**

这些算法的性能有两部分组成，一部分是exploration，另外一部分是exploitation。也可以分别对应于global search和local search。我们都知道，local search的方法是比价欧容易陷入局部最优的，而global的方法是不会对局部最优敏感的。

启发式算法中，局部最优值的陷入无法避免。启发式，本质上是一种贪心策略，这也在客观上决定了不符合贪心规则的更好（或者最优）解会错过。简单来说，避免陷入局部最优就是两个字：随机。具体实现手段上，可以根据所采用的启发式框架来灵活地加入随机性。比如遗传里面，可以在交叉变异时，可以在控制人口策略中，也可以在选择父本母本样本时；禁忌里面，可以在禁忌表的长度上体现，也可以在解禁策略中使用，等等。这些都要结合具体问题特定的算例集，需要反复尝试摸索才行。参数的敏感性是一个问题，建议不要超过3个参数，参数越不敏感越好。不同算例集用不同种子运行多次（100次左右才有统计意义），统计平均性能即可。需注意全局的随机重启通常来说不是一个好办法，因为等于主动放弃之前搜索结果，万不得已不要用，或者就是不用。三个原则应该把握：越随机越好；越不随机越好；二者平衡最好。

1. **凸优化中如何改进GD方法以防止陷入局部最优解**

在对函数进行凸优化时，如果使用导数的方法（如：梯度下降法/GD，牛顿法等）来寻找最优解，有可能陷入到局部最优解而非全局最优解。

为了防止得到局部最优，可以对梯度下降法进行一些改进，防止陷入局部最优。

但是请注意，这些方法只能保证以最大的可能找到全局最优，无法保证100%得到全局最优。

* incremental GD/stochastic GD

在GD中，是需要遍历所有的点之后才计算w的变化的；但是，在stochastic GD中，每输入一个点，就根据该点计算下一步的w，这样，不仅可以从batch training变成online training方法，而且每次是按照单点的最优方向而不是整体的最优方向前进，从而相当于在朝目标前进的路上多拐了好多弯，有可能逃出局部最优。

* momentum方法

momentum相当与记忆住上一次的更新。在每次的更新中，都要加一个k倍的上一次更新量。这样，也不再是按照标准路线前进，每次的步骤都容易受到上一次的影响，从而可能会逃出局部最优。另外，也会加大步长，从而加快收敛。

1. **常见的损失函数？**

* 回归损失函数：

回归的损失函数比较少，常见的有MSE,MAE,RMSE,也比较好理解，就是预测值和真实值直接的差距最小。

分类损失函数：

分类损失函数的种类比较多，常见的分类算法，logistic regression, SVM, adaboost

* log损失函数（逻辑回归）
* 指数损失函数（adaboost）：
* hinge损失函数（SVM)
* 回归损失：均方误差/平方损失/L2 损失
* 平均绝对误差/L1 损失
* 交叉熵损失/负对数似然：这是分类问题中最常见的设置。随着预测概率偏离实际标签，交叉熵损失会逐渐增加。

1. **如何进行特征选择（feature selection）？**

评价准则分为五类：距离度量（Distance Measure）、信息增益度量（Information Gain Measure）、依赖性度量（Dependence Measure）、一致性度量（Consistency Measure）和分类器错误率度量（Classifier Error Rate Measure）。

1. 距离度量：距离度量一般认为是差异性或者分离性的度量，常用的距离度量方法有欧式距离等。对于一个二元分类问题，对于两个特征f1和f2，如果特征f1引起的两类条件概率差异大于特征f2，则认为特征f1优于特征f2。
2. 信息增益度量：特征f的信息增益定义为使用特征f的先验不确定性与期望的后验不确性之间的差异。若特征f1的信息增益大于特征f2的信息增益，则认为特征f1优于特征f2。
3. 依赖性度量：依赖性度量又称为相关性度量（Correlation Measure）、通常可采用皮尔逊相关系数（Pearson correlation coefficient）来计算特征f与类别C之间的相关度，若特征f1与类别C之间的相关性大于特征f2与类别C之间的相关性，则认为特征f1优于特征f2。同样也可以计算得到属性与属性之间的相关度，属性与属性之间的相关性越低越好。
4. 一致性度量：假定两个样本，若它们的特征值相同，且所属类别也相同，则认为它们是一致的：否则，则称它们不一致。一致性常用不一致率来衡量，其尝试找出与原始特征集具有一样辨别能力的最小的属性子集。
5. 分类器错误率度量：该度量使用学习器的性能作为最终的评价阈值。它倾向于选择那些在分类器上表现较好的子集。

以上5种度量方法中，距离度量（Distance Measure）、信息增益度量（Information Gain Measure）、依赖性度量（Dependence Measure）、一致性度量（Consistency Measure）常用于过滤式（filter）；分类器错误率度量(Classifier Error Rate Measure)则用于包裹式(wrapper）。

提到特征选择的动机首先要说下维灾难（the curse of dimensionality）,用个图(图片来自wiki)来形象的说明维灾难：

所谓的维灾难就是当特征维度超过一定界限后，分类器的性能随着特征维度的增加反而下降（而且维度越高训练模型的时间开销也会越大）。导致分类器下降的原因往往是因为这些高纬度特征中含有无关特征和冗余特征，因此特征选择的主要目的是去除特征中的无关特征和冗余特征.

1. **梯度消失/梯度爆炸原因，以及解决方法？**

神经网络的训练过程通常分为两个阶段：前向传播和反向传播。

* 预训练加微调

此方法来自Hinton在2006年发表的一篇论文，Hinton为了解决梯度的问题，提出采取无监督逐层训练方法，其基本思想是每次训练一层隐节点，训练时将上一层隐节点的输出作为输入，而本层隐节点的输出作为下一层隐节点的输入，此过程就是逐层“预训练”（pre-training）；在预训练完成后，再对整个网络进行“微调”（fine-tunning）。Hinton在训练深度信念网络（Deep Belief Networks中，使用了这个方法，在各层预训练完成后，再利用BP算法对整个网络进行训练。此思想相当于是先寻找局部最优，然后整合起来寻找全局最优，此方法有一定的好处，但是目前应用的不是很多了）

* 梯度剪切、正则

梯度剪切这个方案主要是针对梯度爆炸提出的，其思想是设置一个梯度剪切阈值，然后更新梯度的时候，如果梯度超过这个阈值，那么就将其强制限制在这个范围之内。这可以防止梯度爆炸。

* 另外一种解决梯度爆炸的手段是采用权重正则化（weithts regularization）比较常见的是L1正则，和L2正则，在各个深度框架中都有相应的API可以使用正则化，比如在tensorflow tensorflowtensorflow中，若搭建网络的时候已经设置了正则化参数，则调用以下代码可以直接计算出正则损失：
* relu、leak-relu、elu等激活函数
* Batchnorm是深度学习发展以来提出的最重要的成果之一了，目前已经被广泛的应用到了各大网络中，具有加速网络收敛速度，提升训练稳定性的效果，Batchnorm本质上是解决反向传播过程中的梯度问题。batchnorm全名是batch normalization，简称BN，即批规范化，通过规范化操作将输出信号x规范化保证网络的稳定性。

batchnorm就是通过对每一层的输出规范为均值和方差一致的方法，消除了w带来的放大缩小的影响，进而解决梯度消失和爆炸的问题，或者可以理解为BN将输出从饱和区拉倒了非饱和区。

1. **深度学习为什么不用二阶优化**

* 时间复杂度：使用二阶方法通常需要直接计算或者近似估计 Hessian 矩阵，这部分的时间损耗使得其相比一阶方法在收敛速度上带来的优势完全被抵消；
* 考虑cost 和样本数量的关系。在相同计算时间（计算资源）下，使用少数样本估计梯度从而进行多次迭代，和使用全部样本进行单次迭代，哪一种效果更好？ 类似的，使用少数样本估计的梯度进行多次迭代，和使用少数样本估计的Hessian的L-BFGS方法， 哪个效果更好？ 实际中，在相同的时间内使用更多的样本信息所得的效果比使用较少样本更好。对二阶优化算法来说，同样的计算时间中，BFGS只能在一个mini-batch上迭代一次，而SGD能在n个mini-batch上进行迭代，后者取得的cost improvement 比前者更好。这时候，使用二阶方法是不值得的

二阶优化方法目前还不适用于深度学习训练中，主要存在问题是：

1. 最重要的问题是二阶方法的计算量大，训练较慢。

2. 求导不易，实现比SGD这类一阶方法复杂。

3. 另外其优点在深度学习中无法展现出来，主要是二阶方法能够更快地求得更高精度的解，这在浅层模型是有益的，但是在神经网络这类深层模型中对参数的精度要求不高，相反 相对而言不高的精度对模型还有益处，能够提高模型的泛化能力。 当然，二阶优化方法也有优点，在凸优化中，训练较SGD这类方法更为稳定更为平滑，不用调参

1. **怎样优化你的深度学习系统？**

这里有一些方法，可以使用预训练模型来减少你的拟合时间和提高你的准确性：

1. 研究理想的预训练结构：了解迁移学习的好处，或浏览一些强大的CNN架构。考虑那些看起来不太合适，但是共享特征的领域。
2. 使用较小的学习率：因为预训练的权重通常比随机初始化的权重要好！你在这里的选择取决于学习环境和预训练的进展情况，在不同的epochs上检测误差，了解你离收敛有多近。
3. 使用dropout：与Ridge和LASSO正则化回归模型一样，没有一个最优的α适合所有的模型，它是一个超级参数，取决于你的具体问题，必须进行测试。从更大的变化开始，就和上面的学习率一样。
4. 限制权值大小：我们可以限制某些层的权值的最大范数(绝对值)，以泛化我们的模型。
5. 不要动第一层：神经网络的第一个隐藏层倾向于捕捉通用的和可解释的特征，如形状、曲线或交互，这些特征通常与领域相关。我们通常最好是别动这些，重点优化其他的层。这可能意味着添加隐藏层，所以先着急。
6. 修改输出层：用一个新的激活函数和适合你的领域的输出大小替换模型默认值。然而，不要把自己局限于最明显的解决方案上。虽然MNIST可能看起来想要10个输出类，但是一些数字有共同的变化，12-16个类可能会更好地解决这些变化，并提高模型性能！正如上面的提示一样，越接近输出，深度学习模型应该越来越多地进行修改和定制。
7. **为什么要设置单一数字评估指标？**

单一评估指标：就是指当遇到上面的这种情况时，我们只需要从查准率和查全率中选择一个进行评价即可，可以根据不同指标的不同特性来选择适合于自己系统的指标。如果想同时兼顾查准率和查全率时，可以使用F1 score。通过上表可以发现，通过F1 score我们可以很快就挑选出分类器A要优于分类器B。

1. **满足和优化指标（Satisficing and optimizing metrics**）

假设你创建一个硬件设备，这个设备通过麦克风监听用户说话，当发现说特定的“唤醒词（wakeword）”，则会唤醒系统。比如，亚马逊的Echo监听“Alexa”，苹果的Siri监听“Hey Siri”，Android监听“Okay Google”，百度的应用监听“Hello Baidu”。你会关心假正例的比率（the false positive rate）——没有说唤醒词的情况下系统被唤醒的频率，同时也会关心假反例的比率（the false negative rate）——当某个人说了唤醒词它唤醒失败的频率。这样一个系统的合理性能应该是在每24个小时的操作不超过一个假正例的情况下（满足指标），最小化假反例比率（优化指标）。

1. **怎样划分训练/开发/测试集 ？**

* 一般需要将样本分成独立的三部分训练集（train set），开发集（develop set）和测试集（test set）。其中训练集用来估计模型，开发集用来确定网络结构或者控制模型复杂程度的参数，而测试集则检验最终选择最优的模型的性能如何。一个典型的划分是训练集占总样本的50％，而其它各占25％，三部分都是从样本中随机抽取。
* 在大数据时代的机器学习或者深度学习领域中，如果还是按照传统的数据划分方式不是十分合理，因为测试集和验证集用于评估模型和选择模型，所需要的数据量和传统的数据量差不多，但是由于收集到的数据远远大于传统机器学习时代的数据量，所以占的比例也就要缩小。比如我们拥有1000000，这么多的数据，训练集：验证集：测试集=98:1:1。如果是两类，也就是相同的道理。如果你的训练和测试数据来自相同的分布1.打乱并将您的数据分成训练/开发/测试集
* Ng建议训练/开发/测试拆分约70％/ 15％/ 15％。

1. **设置评估指标的意义？**

评估指标

评估指标很多，我们应该选择一个能跟业务指标波动一致的评估指标，这样通过观察评估指标就能判断模型效果，可以大大提高模型迭代效率。

评估指标用于反应模型效果，在预测问题中，要评估模型的效果，就需要将模型预测结果 f(x) ​和真真实标注 Y​ 进行比较，评估指标定义为f(x)和Y​的函数。

score = metric(f(x),Y)

* 精确率(Precision）是指在所有系统判定的“真”的样本中，确实是真的的占比，就是TP/(TP+FP)。
* 召回率（Recall）是指在所有确实为真的样本中，被判为的“真”的占比，就是TP/(TP+FN)。
* TPR（True Positive Rate）的定义，跟Recall一样。
* FPR（False Positive Rate），又被称为“Probability of False Alarm”，就是所有确实为“假”的样本中，被误判真的样本，或者FP/(FP+TN)
* F1值是为了综合考量精确率和召回率而设计的一个指标，一般公式为取P和R的harmonic mean:2PrecisionRecall/(Precision+Recall)。
* ROC=Receiver Operating Characteristic，是TPR vs FPR的曲线；与之对应的是Precision-Recall Curve，展示的是Precision vs Recall的曲线。

1. **什么是可避免偏差？**

Avoidable bias (可避免偏差)

可避免偏差这个词说明了有一些别的偏差，或者错误率有个无法超越的最低水平，那就是说如果贝叶斯错误率是 7.5%。你实际上并不想得到低于该级别的错误率，所以你不会说你的训练错误率是 8%，然后 8%就衡量了例子中的偏差大小。你应该说，可避免偏差可能在0.5%左右，或者 0.5%是可避免偏差的指标。而这个 2%是方差的指标，所以要减少这个 2%比减少这个 0.5%空间要大得多。而在左边的例子中，这 7%衡量了可避免偏差大小，而 2%衡量了方差大小。所以在左边这个例子里，专注减少可避免偏差可能潜力更大。

对于某些任务如计算机视觉上，人类能够做到的水平和贝叶斯误差相差不远。（这里贝叶斯误差指最好的分类器的分类误差，也就是说没有分类器可以做到100% 正确）。这里将人类水平误差近似为贝叶斯误差。

左边的例子：8% 与 1% 差距较大

主要专注于减少偏差，即减少训练集误差和人类水平误差之间的差距，来提高模型性能。

右边的例子：8% 与 7.5% 接近

主要专注于减少方差，即减少开发集误差和测试集误差之间的差距，来提高模型性能。

1. **什么是TOP5错误率？**

imagenet图像通常有1000个可能的类别，对每幅图像你可以猜5次结果(即同时预测5个类别标签)，当其中有任何一次预测对了，结果都算对，当5次全都错了的时候，才算预测错误，这时候的分类错误率就叫top5错误率.

Top-1 error 的意思是：假如模型预测某张动物图片（一只猫）的类别，且模型只输出1个预测结果，那么这一个结果正好能猜出来这个动物是只猫的概率就是Top-1正确率。猜出来的结果不是猫的概率则成为Top-1错误率。简单来说就是模型猜错的概率。Top-5 error 的意思是：假如模型预测某张动物图片（还是刚才那只猫），但模型会输出来5个预测结果，那么这五个结果中有猫这个分类的概率成为Top-5正确率，相反，预测输出的这五个结果里没有猫这个分类的概率则成为Top-5错误率。一般来说，Top-1和Top-5错误率越低，模型的性能也就越好。且Top-5 error 在数值上会比Top-1 error 的数值要小，毕竟从1个结果猜对的几率总会比从5个结果里猜对的几率要小嘛！

1. **有哪些超参数，为超参数选择合适的范围，如何搜索超参数？**

超参数是我们控制我们模型结构、功能、效率等的 调节旋钮，具体有哪些呢：

* learning rate
* epochs(迭代次数，也可称为 num of iterations)
* num of hidden layers(隐层数目)
* num of hidden layer units(隐层的单元数/神经元数)
* activation function(激活函数)
* batch-size(用mini-batch SGD的时候每个批量的大小)
* optimizer(选择什么优化器，如SGD、RMSProp、Adam)
* 用诸如RMSProp、Adam优化器的时候涉及到的β1，β2等等

人工的超参数选择过程，我们也可以采取类似参数搜索的办法，来提高效率，如果进行人工试错的方式，会非常浪费时间。

* 超参数搜索过程
* 将数据集分为训练集，验证集及测试集。
* 选择模型性能评价指标
* 用训练集对模型进行训练
* 在验证集上对模型进行参数进行搜索，用性能指标评价参数好坏
* 选出最优参数

常见超参数搜索算法：

* 网格搜索
* 随机搜索
* 启发式搜索

网格搜索：

网格搜索是在所有候选的参数选择中，通过循环遍历，尝试每一种可能性，表现最好的参数就是最终的结果（暴力搜索）。

随机搜索：

随机搜索（random search）是利用随机数去求函数近似的最优解的方法，区别于网格搜索的暴力搜索方式。

原理：在一定的区间内，不断随机地而不是有倾向性产生随机点，并计算其约束函数和目标函数的值，对满足约束条件的点，逐个比较其目标函数的值，将坏的点抛弃，保留好的点，最后便得到最优解的近似解。

启发式搜索：

启发式搜索(Heuristically Search)又称为有信息搜索(Informed Search)，它是利用问题拥有的启发信息来引导搜索，达到减少搜索范围、降低问题复杂度的目的，这种利用启发信息的搜索过程称为启发式搜索。

原理：在状态空间中的搜索对每一个搜索的位置进行评估，得到最好的位置，再从这个位置进行搜索直到目标。这样可以省略大量无谓的搜索路径，提高了效率。在启发式搜索中，对位置的估价是十分重要的。采用了不同的估价可以有不同的效果。

启发式搜索有模拟退火算法(SA)、遗传算法(GA)、列表搜索算法(ST)、进化规划(EP)、进化策略(ES)、蚁群算法(ACA)、人工神经网络(ANN)...等。

1. **What’s the trade-off between bias and variance?**

* Bias 是由于你使用的学习算法过度简单地拟合结果或者错误地拟合结果导致的错误。它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，即算法本身的拟合能力。Bias 可能会导致模型欠拟合，使其难以具有较高的预测准确性，也很难将你的知识从训练集推广到测试集。
* Variance 是由于你使用的学习算法过于复杂而产生的错误。它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。反应预测的波动情况。Variance 过高会导致算法对训练数据的高纬度变化过于敏感，这样会导致模型过度拟合数据。从而你的模型会从训练集里带来太多噪音，这会对测试数据有一定的好处。
* Bias-Variance 的分解，本质上是通过在基础数据集中添加偏差、方差和一点由噪声引起的不可约误差，来分解算法上的学习误差。从本质上讲，如果你使模型更复杂并添加更多变量，你将会失去一些 Bias 但获得一些 Variance，这就是我们所说的权衡（tradeoff）。这也是为什么我们在建模的过程中，不希望这个模型同时拥有高的偏差和方差。

1. **How is KNN different from k-means clustering?**

问题3: KNN和 k-means 聚类由什么不同？

K-Nearest Neighbors是一种监督分类算法，而 k-means聚类是一种无监督的聚类算法。 虽然这些机制起初可能看起来相似，但这实际上意味着为了使K-Nearest Neighbors工作，你需要标记数据，以便将未标记的点分类（因此是最近邻居部分）。 K均值聚类仅需要一组未标记的点和阈值：算法将采用未标记的点并逐渐学习如何通过计算不同点之间的距离的平均值将它们聚类成组。

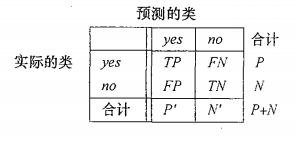
这里的关键区别在于，KNN需要标记点，因此是有监督的学习，而k-means不是，因此是无监督学习。

1. **ROC**

ROC曲线的作用：

* 较容易地查出任意界限值时的对类别的识别能力
* 选择最佳的诊断界限值。ROC曲线越靠近左上角,试验的准确性就越高。最靠近左上角的ROC曲线的点是错误最少的最好阈值，其假阳性和假阴性的总数最少。
* 两种或两种以上不同诊断试验对算法性能的比较。在对同一种算法的两种或两种以上诊断方法进行比较时，可将各试验的ROC曲线绘制到同一坐标中，以直观地鉴别优劣，靠近左上角的ROC曲线所代表的受试者工作最准确。亦可通过分别计算各个试验的ROC曲线下的面积(AUC)进行比较，哪一种试验的AUC最大，则哪一种试验的诊断价值最佳。

ROC曲线是根据一系列不同的二分类方式（分界值或决定阈），以真阳性率TPR（灵敏度）为纵坐标，假阳性率FPR（1-特异度）为横坐标绘制的曲线。



TPR = TP/P 即召回率公式

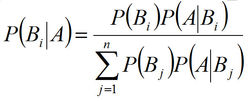
FPR = FP/N 即1-specificity

ROC曲线是以FPR为横坐标，以TPR为纵坐标，以概率为阈值来度量模型正确识别正实例的比例与模型错误的把负实例识别成正实例的比例之间的权衡，TPR的增加必定以FPR的增加为代价，ROC曲线下方的面积是模型准确率的度量所以根据ROC曲线定义可知，绘制ROC要求模型必须能返回监测元组的类预测概率，根据概率对元组排序和定秩，并使正概率较大的在顶部，负概率较大的在底部进行画图

1. **What is Bayes’ Theorem? How is it useful in a machine learning context?**

什么是贝叶斯定理？它在机器学习环境中如何有用?

贝叶斯定理描述了当你不能准确知悉一个事物的本质时，你可以依靠与事物特定本质相关的事件出现的多少去判断其本质属性的概率。 它给出了已知先验知识下事件的后验概率。



1. **Explain the difference between L1 and L2 regularization.**

问题8：L1、L2正则之间有什么不同？

* L2正则，对应的是加入2范数，使得对权重进行衰减，从而达到惩罚损失函数的目的，防止模型过拟合。保留显著减小损失函数方向上的权重，而对于那些对函数值影响不大的权重使其衰减接近于0。相当于加入一个gaussian prior。
* L1正则 对应得失加入1范数，同样可以防止过拟合。它会产生更稀疏的解，即会使得部分权重变为0，达到特征选择的效果。

一般回归分析中回归w表示特征的系数，从上式可以看到正则化项是对系数做了处理（限制）。L1正则化和L2正则化的说明如下：

* L1正则化是指权值向量w中各个元素的绝对值之和，通常表示为||w||\_1
* L2正则化是指权值向量w中各个元素的平方和然后再求平方根（可以看到Ridge回归的L2正则化项有平方符号），通常表示为||w||\_2

相同点：都用于避免过拟合

不同点：L1可以让一部分特征的系数缩小到0，从而间接实现特征选择。所以L1适用于特征之间有关联的情况, L2让所有特征的系数都缩小，但是不会减为0，它会使优化求解稳定快速。所以L2适用于特征之间没有关联的情况

1. **第一类误差和第二类误差有什么区别？**

第一类误差指的是假正率，第二类指的是假负率。简单来说，第一类误差意味着假设为真的情况下，作出了拒绝原假设的一种错误推断。第二类误差意味着假设为假的情况下，做出了接受原假设的一种错误判断。

举个例子：第一类误差，你误判一个男的他怀孕了。第二类误差，你误判了一位其实已经怀孕的女子没怀孕。

1. **概率和似然有什么区别？**

| **符号** | **含义** |
| --- | --- |
| O | 观测值 |
| θ | 随机过程中的参数 |
| θ^ | 参数的估计 |
| P(O|θ) | 概率 |
| L(θ|O) | 释然 |

“概率”描述了给定模型参数后，描述结果的合理性，而不涉及任何观察到的数据。

抛一枚均匀的硬币，拋20次，问15次拋得正面的可能性有多大？ 这里的可能性就是”概率”，均匀的硬币就是给定参数θ=0.5，“拋20次15次正面”是观测值O。求概率P(H=15|θ=0.5)=？的概率。

“似然”描述了给定了特定观测值后，描述模型参数是否合理。

拋一枚硬币，拋20次，结果15次正面向上，问其为均匀的可能性？ 这里的可能性就是”似然”，“拋20次15次正面”为观测值O为已知，参数θ=?并不知道，求L(θ|H=15)=P(H=15|θ=0.5)的最大化下的θ 值。

1. **如何对决策树进行剪枝？**

判断的标准就是看划分前后的泛华性能是否有提升，也就是如果划分后泛华性能有提升，则划分；否则，不划分。

对比预剪枝和后剪枝，能够发现，后剪枝决策树通常比预剪枝决策树保留了更多的分支，一般情形下，后剪枝决策树的欠拟合风险小，泛华性能往往也要优于预剪枝决策树。但后剪枝过程是在构建完全决策树之后进行的，并且要自底向上的对树中的所有非叶结点进行逐一考察，因此其训练时间开销要比未剪枝决策树和预剪枝决策树都大得多。

剪枝是在决策树中，为了降低模型的复杂度，提高决策树模型的预测精度，去除预测能力较弱的分支后所发生的现象。修剪可以自下而上和自上而下进行，方法包括减少错误修剪和成本复杂度修剪。

减少错误修剪可能是最简单的版本：替换每个节点。如果不降低预测精度，则保持修剪。虽然很简单，但这种启发式方法实际上非常接近于一种可以最大限度地优化准确性的方法。

1. **什么是F1数，怎么使用它？**

F1分数是衡量模型性能的指标。它是模型精度和召回的加权平均值，结果趋向于1是最好的，结果趋向于0是最差的。你可以在分类测试中使用它，而真正的否定并不重要

1. **逻辑回归**

逻辑回归（Logistic Regression）是一种用于解决二分类（0 or 1）问题的机器学习方法，用于估计某种事物的可能性。比如某用户购买某商品的可能性，某病人患有某种疾病的可能性，以及某广告被用户点击的可能性等。 注意，这里用的是“可能性”，而非数学上的“概率”，logisitc回归的结果并非数学定义中的概率值，不可以直接当做概率值来用。该结果往往用于和其他特征值加权求和，而非直接相乘。

那么逻辑回归与线性回归是什么关系呢？

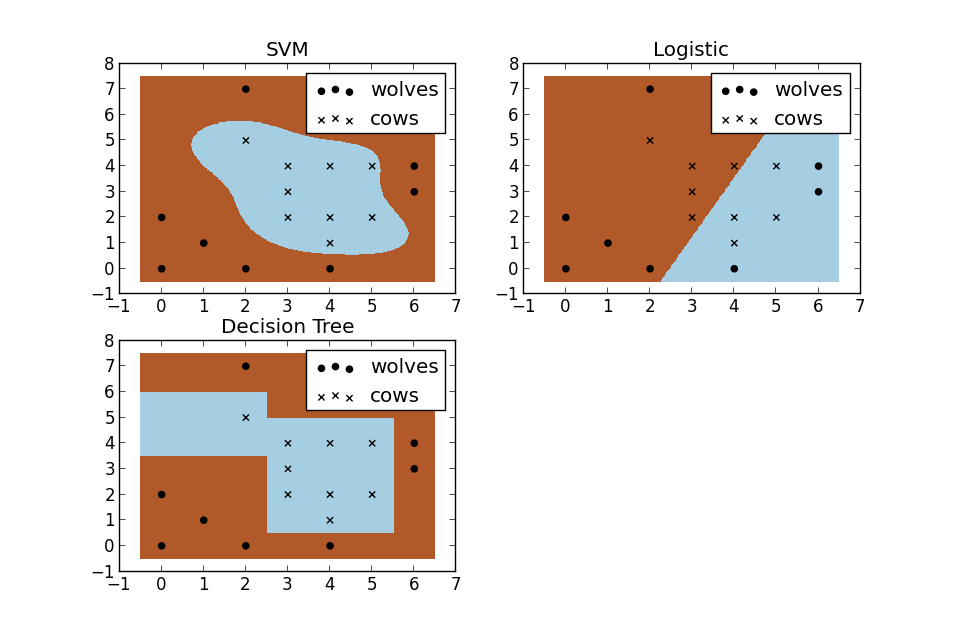
逻辑回归（Logistic Regression）与线性回归（Linear Regression）都是一种广义线性模型（generalized linear model）。逻辑回归假设因变量 y 服从伯努利分布，而线性回归假设因变量 y 服从高斯分布。 因此与线性回归有很多相同之处，去除Sigmoid映射函数的话，逻辑回归算法就是一个线性回归。可以说，逻辑回归是以线性回归为理论支持的，但是逻辑回归通过Sigmoid函数引入了非线性因素，因此可以轻松处理0/1分类问题。

从上图可以看到sigmoid函数是一个s形的曲线，它的取值在[0, 1]之间，在远离0的地方函数的值会很快接近0或者1。它的这个特性对于解决二分类问题十分重要

* 在线性回归中，最常用的是均方误差(Mean squared error)，
* 在逻辑回归中，最常用的是代价函数是交叉熵(Cross Entropy)，交叉熵是一个常见的代价函数，在神经网络中也会用到。

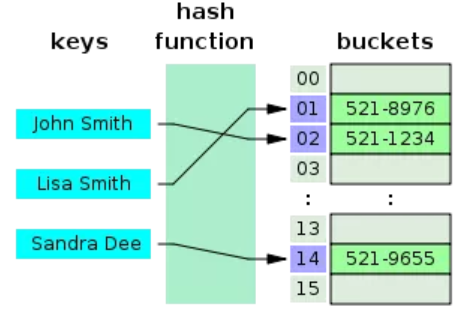
1. **什么是核技巧，有什么用处？**

核技巧使用核函数，确保在高维空间不需要明确计算点的坐标，而是计算数据的特征空间中的内积。这使其具有一个很有用的属性：更容易的计算高维空间中点的坐标。许多算法都可以表示称这样的内积形式，使用核技巧可以保证低维数据在高维空间中运用算法进行计算。



1. **描述哈希表**

哈希表是一种产生关联数组的数据结构。 通过使用散列函数将键映射到某些值。 它们通常用于数据库索引等任务



1. **请简要介绍下SVM**

SVM，全称是support vector machine，中文名叫支持向量机。SVM是一个面向数据的分类算法，它的目标是为确定一个分类超平面，从而将不同的数据分隔开。

它是一种二类分类模型，其基本模型定义为特征空间上的间隔最大的线性分类器，其学习策略便是间隔最大化，最终可转化为一个凸二次规划问题的求解。

1. **在k-means或kNN，我们常用欧氏距离来计算最近的邻居之间的距离，有时也用曼哈顿距离，请对比下这两种距离的差别**。

https://img-my.csdn.net/uploads/201211/20/1353398777_7638.png

* 欧氏距离，最常见的两点之间或多点之间的距离表示法，又称之为欧几里得度量

欧氏距离虽然很有用，但也有明显的缺点。它将样品的不同属性（即各指标或各变量量纲）之间的差别等同看待，这一点有时不能满足实际要求。

* 曼哈顿距离，我们可以定义曼哈顿距离的正式意义为L1-距离或城市区块距离，也就是在欧几里得空间的固定直角坐标系上两点所形成的线段对轴产生的投影的距离总和。例如在平面上，坐标（x1,y1）的点P1与坐标（x2,y2）的点P2的曼哈顿距离为：https://img-my.csdn.net/uploads/201211/20/1353398955_7627.png，要注意的是，曼哈顿距离依赖座标系统的转度，而非系统在座标轴上的平移或映射。当坐标轴变动时，点间的距离就会不同。

通俗来讲，想象你在曼哈顿要从一个十字路口开车到另外一个十字路口，驾驶距离是两点间的直线距离吗？显然不是，除非你能穿越大楼。而实际驾驶距离就是这个“曼哈顿距离”，这也是曼哈顿距离名称的来源， 同时，曼哈顿距离也称为城市街区距离(City Block distance)。

1. **CNN的卷积核是单层的还是多层的？**

一般而言，深度卷积网络是一层又一层的。层的本质是特征图, 存贮输入数据或其中间表示值。一组卷积核则是联系前后两层的网络参数表达体, 训练的目标就是每个卷积核的权重参数组。

描述网络模型中某层的厚度，通常用名词通道channel数或者特征图feature map数。不过人们更习惯把作为数据输入的前层的厚度称之为通道数（比如RGB三色图层称为输入通道数为3），把作为卷积输出的后层的厚度称之为特征图数。

卷积核(filter)一般是3D多层的，除了面积参数, 比如3x3之外, 还有厚度参数H（2D的视为厚度1). 还有一个属性是卷积核的个数N。

卷积核的厚度H, 一般等于前层厚度M(输入通道数或feature map数). 特殊情况M > H。

卷积核的个数N, 一般等于后层厚度(后层feature maps数)因为相等所以也用N表示)。

卷积核通常从属于后层，为后层提供了各种查看前层特征的视角，这个视角是自动形成的。

卷积核厚度等于1时为2D卷积，也就是平面对应点分别相乘然后把结果加起来，相当于点积运算.

卷积的意思就是把一个区域，不管是一维线段，二维方阵，还是三维长方块，全部按照卷积核的维度形状，从输入挖出同样维度形状, 对应逐点相乘后求和，浓缩成一个标量值也就是降到零维度，作为输出到一个特征图的一个点的值. 这个很像渔夫收网。

可以比喻一群渔夫坐一个渔船撒网打鱼，鱼塘是多层水域，每层鱼儿不同。

船每次移位一个stride到一个地方，每个渔夫撒一网，得到收获，然后换一个距离stride再撒，如此重复直到遍历鱼塘。

A渔夫盯着鱼的品种，遍历鱼塘后该渔夫描绘了鱼塘的鱼品种分布；

B渔夫盯着鱼的重量，遍历鱼塘后该渔夫描绘了鱼塘的鱼重量分布；

还有N-2个渔夫，各自兴趣各干各的；

最后得到N个特征图，描述了鱼塘的一切！

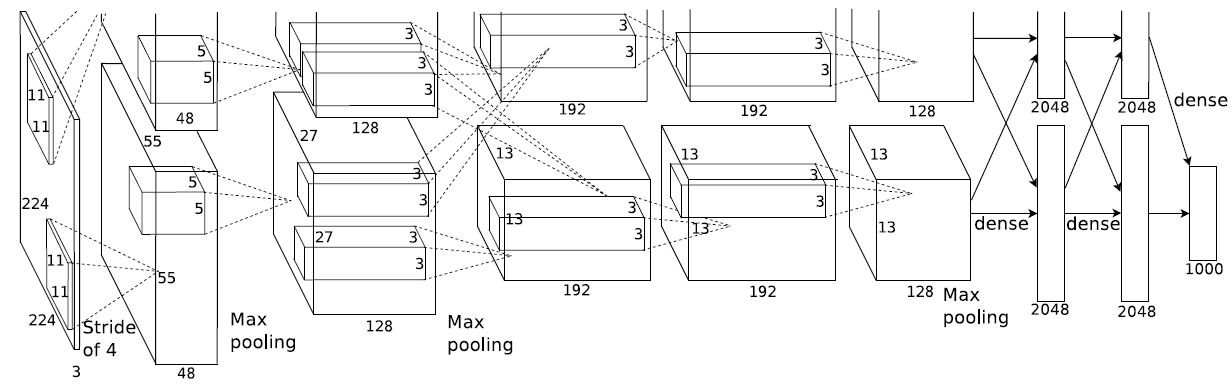
2D卷积表示渔夫的网就是带一圈浮标的渔网，只打上面一层水体的鱼；

3D卷积表示渔夫的网是多层嵌套的渔网，上中下层水体的鱼儿都跑不掉；

1x1卷积可以视为每次移位stride，甩钩钓鱼代替了撒网；

下面解释一下特殊情况的 M > H：

实际上，除了输入数据的通道数比较少之外，中间层的feature map数很多，这样中间层算卷积会累死计算机（鱼塘太深，每层鱼都打，需要的鱼网太重了）。所以很多深度卷积网络把全部通道/特征图划分一下，每个卷积核只看其中一部分（渔夫A的渔网只打捞深水段，渔夫B的渔网只打捞浅水段）。这样整个深度网络架构是横向开始分道扬镳了，到最后才又融合。这样看来，很多网络模型的架构不完全是突发奇想，而是是被参数计算量逼得。特别是现在需要在移动设备上进行AI应用计算(也叫推断), 模型参数规模必须更小, 所以出现很多减少握手规模的卷积形式, 现在主流网络架构大都如此。比如AlexNet：



1. **Logistic Regression:**

* 在逻辑回归模型中，我们最大化似然函数和最小化对数似然损失函数实际上是等价的。

L1范数：是指向量中各个元素绝对值之和，也有个美称叫“稀疏规则算子”（Lasso regularization）。那么，参数稀疏 有什么好处呢？

一个关键原因在于它能实现 特征的自动选择。一般来说，大部分特征 xixi和输出 yiyi 之间并没有多大关系。在最小化目标函数的时候考虑到这些额外的特征 xixi，虽然可以获得更小的训练误差，但在预测新的样本时，这些没用的信息反而会干扰了对正确 yiyi 的预测。稀疏规则化算子的引入就是为了完成特征自动选择的光荣使命，它会学习地去掉这些没有信息的特征，也就是把这些特征对应的权重置为0。

L2范数：它有两个美称，在回归里面，有人把有它的回归叫“岭回归”（Ridge Regression），有人也叫它“权值衰减”(weight decay)。

它的强大之处就是它能 解决过拟合 问题。我们让 L2 范数的规则项 ||w||2|最小，可以使得 w 的每个元素都很小，都接近于0，但与 L1 范数不同，它不会让它等于0，而是接近于0，这里还是有很大区别的。而越小的参数说明模型越简单，越简单的模型则越不容易产生过拟合现象。咦，你为啥说越小的参数表示的模型越简单呢？ 其实我也不知道，我也是猜，可能是因为参数小，对结果的影响就小了吧。

虽然逻辑回归能够用于分类，不过其本质还是线性回归。它仅在线性回归的基础上，在特征到结果的映射中加入了一层sigmoid函数（非线性）映射，即先把特征线性求和，然后使用sigmoid函数来预测。

线性回归在整个实数域内敏感度一致，而分类范围，需要在[0,1]之内。而逻辑回归就是一种减小预测范围，将预测值限定为[0,1]间的一种回归模型，其回归方程与回归曲线如下图所示。逻辑曲线在z=0时，十分敏感，在z>>0或z<<0处，都不敏感，将预测值限定为(0,1)。

逻辑回归跟最大熵模型没有本质区别。逻辑回归是最大熵对应类别为二类时的特殊情况，也就是当逻辑回归类别扩展到多类别时，就是最大熵模型。

逻辑回归跟最大熵模型没有本质区别。逻辑回归是最大熵对应类别为二类时的特殊情况

指数簇分布的最大熵等价于其指数形式的最大似然。

二项式分布的最大熵解等价于二项式指数形式(sigmoid)的最大似然；

多项式分布的最大熵等价于多项式分布指数形式(softmax)的最大似然

1. **overfitting怎么解决？**

dropout、regularization、batch normalization

* 正则化(Regularization)

L2正则化：目标函数中增加所有权重w参数的平方之和, 逼迫所有w尽可能趋向零但不为零. 因为过拟合的时候, 拟合函数需要顾忌每一个点, 最终形成的拟合函数波动很大, 在某些很小的区间里, 函数值的变化很剧烈, 也就是某些w非常大. 为此, L2正则化的加入就惩罚了权重变大的趋势.

L1正则化：目标函数中增加所有权重w参数的绝对值之和, 逼迫更多w为零(也就是变稀疏. L2因为其导数也趋0, 奔向零的速度不如L1给力了). 大家对稀疏规则化趋之若鹜的一个关键原因在于它能实现特征的自动选择。一般来说，xi的大部分元素（也就是特征）都是和最终的输出yi没有关系或者不提供任何信息的，在最小化目标函数的时候考虑xi这些额外的特征，虽然可以获得更小的训练误差，但在预测新的样本时，这些没用的特征权重反而会被考虑，从而干扰了对正确yi的预测。稀疏规则化算子的引入就是为了完成特征自动选择的光荣使命，它会学习地去掉这些无用的特征，也就是把这些特征对应的权重置为0。

* 随机失活(dropout)

在训练的运行的时候，让神经元以超参数p的概率被激活(也就是1-p的概率被设置为0), 每个w因此随机参与, 使得任意w都不是不可或缺的, 效果类似于数量巨大的模型集成。

* 逐层归一化(batch normalization)

这个方法给每层的输出都做一次归一化(网络上相当于加了一个线性变换层), 使得下一层的输入接近高斯分布. 这个方法相当于下一层的w训练时避免了其输入以偏概全, 因而泛化效果非常好.

1. **LR和SVM的联系与区别**
2. **deep learning（rnn、cnn）调参的经验**

* 参数初始化

1. uniform均匀分布初始化：

传统的初始化权重问题是用标准正态分布（均值为0，方差为1）随机初始化

w = np.random.uniform(low=-scale, high=scale, size=[n\_in,n\_out])

把权值与偏置进行 均匀分布的初始化。用min 与 max 来控制它们的的上下 限，默认为（0，1）

1. Xavier初始法，适用于普通激活函数(tanh,sigmoid)：scale = np.sqrt(3/n)

适用于普通激活函数 (tanh,sigmoid)：stdev = np.sqrt(n)

1. He初始化，适用于ReLU：scale = np.sqrt(6/n)
2. normal高斯分布初始化：w = np.random.randn(n\_in,n\_out) \* stdev

# stdev为高斯分布的标准差，均值设为0

1. svd初始化：对RNN有比较好的效果。

* 数据预处理方式
  1. 0均值（zero-center） 是最常用的预处理方法，就是把数据的每一维-每一维的均值，这样数据就变成0均值的了。在numpy中，这个操作可以写成：X -= np.mean(X, axis = 0)。对于图片来讲，我们可以更简单地对所有pixel减去同一个均值（如 X -= np.mean(X)），当然也可以对RGB三个通道分别减均值。

2）归一化(Normalization) 是指将数据归一化到相同的尺度。通常有两种归一化的方法。第一种是0均值以后的数据的每一维/每一维的标准差（X /= np.std(X, axis = 0)）；另一种是将数据归一化到每一维的最大最小值为1和-1。这种归一化只适用于当你认为数据的不同维度应该具有相同的重要性时。对于图片来说，不同像素的尺度是基本一致的（0-255），因此我们并不需要对它进行归一化操作。

3） PCA和白化（PCA whitening） 是另一种形式的预处理方法。首先我们将数据变成0均值的，然后计算数据的协方差矩阵来得到数据不同维度之间的相关性：

* 训练技巧
  1. 要做梯度归一化,即算出来的梯度除以minibatch size

2）clip c(梯度裁剪): 限制最大梯度,其实是value = sqrt(w1^2+w2^2….),如果value超过了阈值,就算一个衰减系系数,让value的值等于阈值: 5,10,15

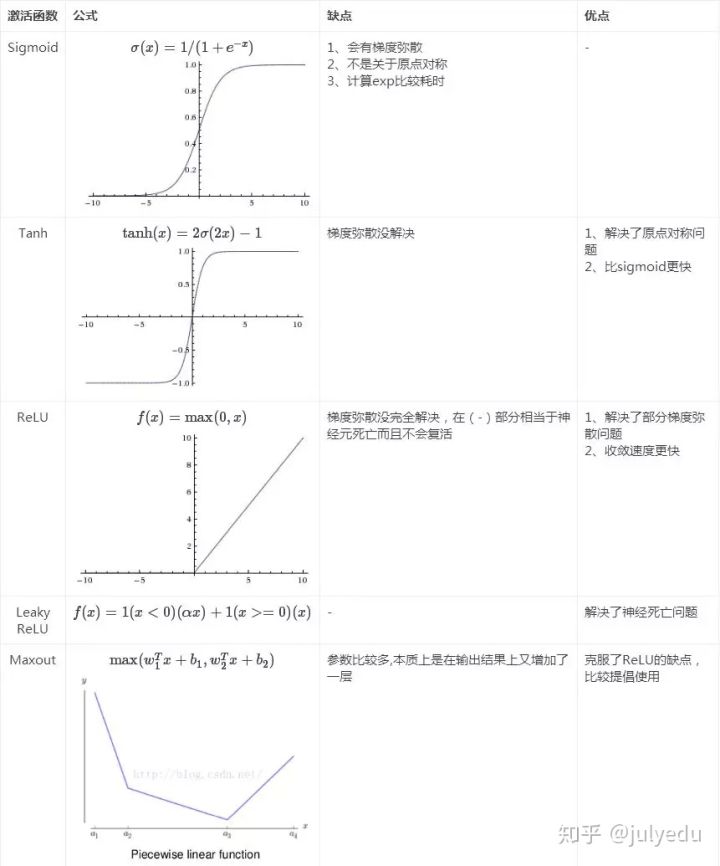
3）dropout对小数据防止过拟合有很好的效果,值一般设为0.5,小数据上dropout+sgd在我的大部分实验中，效果提升都非常明显.因此可能的话，建议一定要尝试一下。 dropout的位置比较有讲究, 对于RNN,建议放到输入->RNN与RNN->输出的位置.

1. adam,adadelta等,在小数据上,我这里实验的效果不如sgd, sgd收敛速度会慢一些，但是最终收敛后的结果，一般都比较好。如果使用sgd的话,可以选择从1.0或者0.1的学习率开始,隔一段时间,在验证集上检查一下,如果cost没有下降,就对学习率减半. 我看过很多论文都这么搞,我自己实验的结果也很好. 当然,也可以先用ada系列先跑,最后快收敛的时候,更换成sgd继续训练.同样也会有提升.据说adadelta一般在分类问题上效果比较好，adam在生成问题上效果比较好。
2. 除了gate之类的地方,需要把输出限制成0-1之外,尽量不要用sigmoid,可以用tanh或者relu之类的激活函数.1. sigmoid函数在-4到4的区间里，才有较大的梯度。之外的区间，梯度接近0，很容易造成梯度消失问题。2. 输入0均值，sigmoid函数的输出不是0均值的。
3. rnn的dim和embdding size,一般从128上下开始调整. batch size,一般从128左右开始调整.batch size合适最重要,并不是越大越好
4. **CNN特点**

CNN抓住此共性的手段主要有四个：局部连接／权值共享／池化操作／多层次结构。

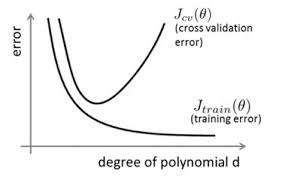
局部连接使网络可以提取数据的局部特征；权值共享大大降低了网络的训练难度，一个Filter只提取一个特征，在整个图片（或者语音／文本） 中进行卷积；池化操作与多层次结构一起，实现了数据的降维，将低层次的局部特征组合成为较高层次的特征，从而对整个图片进行表示

1. 激活函数



1. **overfitting怎么解决？**

dropout、regularization、batch normalizatin



其直观的表现如下图所示，随着训练过程的进行，模型复杂度增加，在training data上的error渐渐减小，但是在验证集上的error却反而渐渐增大——因为训练出来的网络过拟合了训练集, 对训练集外的数据却不work, 这称之为泛化(generalization)性能不好。泛化性能是训练的效果评价中的首要目标，没有良好的泛化，就等于南辕北辙, 一切都是无用功。过拟合是泛化的反面，好比乡下快活的刘姥姥进了大观园会各种不适应，但受过良好教育的林黛玉进贾府就不会大惊小怪。实际训练中, 降低过拟合的办法一般如下：

* 正则化(Regularization)

L2正则化：目标函数中增加所有权重w参数的平方之和, 逼迫所有w尽可能趋向零但不为零. 因为过拟合的时候, 拟合函数需要顾忌每一个点, 最终形成的拟合函数波动很大, 在某些很小的区间里, 函数值的变化很剧烈, 也就是某些w非常大. 为此, L2正则化的加入就惩罚了权重变大的趋势.

L1正则化：目标函数中增加所有权重w参数的绝对值之和, 逼迫更多w为零(也就是变稀疏. L2因为其导数也趋0, 奔向零的速度不如L1给力了). 大家对稀疏规则化趋之若鹜的一个关键原因在于它能实现特征的自动选择。一般来说，xi的大部分元素（也就是特征）都是和最终的输出yi没有关系或者不提供任何信息的，在最小化目标函数的时候考虑xi这些额外的特征，虽然可以获得更小的训练误差，但在预测新的样本时，这些没用的特征权重反而会被考虑，从而干扰了对正确yi的预测。稀疏规则化算子的引入就是为了完成特征自动选择的光荣使命，它会学习地去掉这些无用的特征，也就是把这些特征对应的权重置为0。

* 随机失活(dropout)

在训练的运行的时候，让神经元以超参数p的概率被激活(也就是1-p的概率被设置为0), 每个w因此随机参与, 使得任意w都不是不可或缺的, 效果类似于数量巨大的模型集成。

* 逐层归一化(batch normalization)

这个方法给每层的输出都做一次归一化(网络上相当于加了一个线性变换层), 使得下一层的输入接近高斯分布. 这个方法相当于下一层的w训练时避免了其输入以偏概全, 因而泛化效果非常好.

* 提前终止(early stopping)

理论上可能的局部极小值数量随参数的数量呈指数增长, 到达某个精确的最小值是不良泛化的一个来源. 实践表明, 追求细粒度极小值具有较高的泛化误差。这是直观的，因为我们通常会希望我们的误差函数是平滑的, 精确的最小值处所见相应误差曲面具有高度不规则性, 而我们的泛化要求减少精确度去获得平滑最小值, 所以很多训练方法都提出了提前终止策略. 典型的方法是根据交叉叉验证提前终止: 若每次训练前, 将训练数据划分为若干份, 取一份为测试集, 其他为训练集, 每次训练完立即拿此次选中的测试集自测. 因为每份都有一次机会当测试集, 所以此方法称之为交叉验证. 交叉验证的错误率最小时可以认为泛化性能最好, 这时候训练错误率虽然还在继续下降, 但也得终止继续训练了.

1. **LR和SVM的联系与区别**

1、LR和SVM都可以处理分类问题，且一般都用于处理线性二分类问题（在改进的情况下可以处理多分类问题）

2、两个方法都可以增加不同的正则化项，如l1、l2等等。所以在很多实验中，两种算法的结果是很接近的。

区别：

1、LR是参数模型，SVM是非参数模型。

2、从目标函数来看，区别在于逻辑回归采用的是logistical loss，SVM采用的是hinge loss，这两个损失函数的目的都是增加对分类影响较大的数据点的权重，减少与分类关系较小的数据点的权重。

3、SVM的处理方法是只考虑support vectors，也就是和分类最相关的少数点，去学习分类器。而逻辑回归通过非线性映射，大大减小了离分类平面较远的点的权重，相对提升了与分类最相关的数据点的权重。

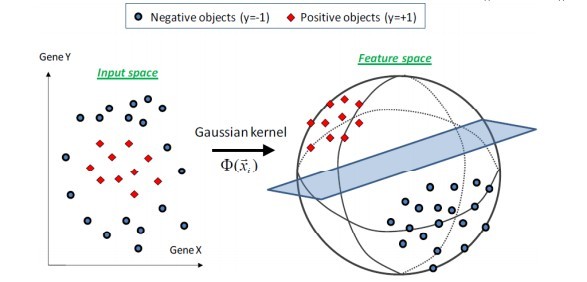
4、逻辑回归相对来说模型更简单，好理解，特别是大规模线性分类时比较方便。而SVM的理解和优化相对来说复杂一些，SVM转化为对偶问题后,分类只需要计算与少数几个支持向量的距离,这个在进行复杂核函数计算时优势很明显,能够大大简化模型和计算。

5、logic 能做的 svm能做，但可能在准确率上有问题，svm能做的logic有的做不了。

1. **说说你知道的核函数**

通常人们会从一些常用的核函数中选择（根据问题和数据的不同，选择不同的参数，实际上就是得到了不同的核函数），例如：

* 多项式核，显然刚才我们举的例子是这里多项式核的一个特例（R = 1，d = 2）。虽然比较麻烦，而且没有必要，不过这个核所对应的映射实际上是可以写出来的，该空间的维度是，其中 是原始空间的维度。https://img-my.csdn.net/uploads/201304/03/1364958204_2877.jpg
* 高斯核，https://img-my.csdn.net/uploads/201304/03/1364958259_8460.jpg这个核就是最开始提到过的会将原始空间映射为无穷维空间的那个家伙。不过，如果选得很大的话，高次特征上的权重实际上衰减得非常快，所以实际上（数值上近似一下）相当于一个低维的子空间；反过来，如果选得很小，则可以将任意的数据映射为线性可分——当然，这并不一定是好事，因为随之而来的可能是非常严重的过拟合问题。不过，总的来说，通过调控参数，高斯核实际上具有相当高的灵活性，也是使用最广泛的核函数之一。下图所示的例子便是把低维线性不可分的数据通过高斯核函数映射到了高维空间：



* 线性核，https://img-my.csdn.net/uploads/201304/03/1364958354_7262.jpg这实际上就是原始空间中的内积。这个核存在的主要目的是使得“映射后空间中的问题”和“映射前空间中的问题”两者在形式上统一起来了(意思是说，咱们有的时候，写代码，或写公式的时候，只要写个模板或通用表达式，然后再代入不同的核，便可以了，于此，便在形式上统一了起来，不用再分别写一个线性的，和一个非线性的)。

1. **LR与线性回归的区别与联系**

LR工业上一般指Logistic Regression(逻辑回归)而不是Linear Regression(线性回归). LR在线性回归的实数范围输出值上施加sigmoid函数将值收敛到0~1范围, 其目标函数也因此从差平方和函数变为对数损失函数, 以提供最优化所需导数（sigmoid函数是softmax函数的二元特例, 其导数均为函数值的f\*(1-f)形式）。请注意, LR往往是解决二元0/1分类问题的, 只是它和线性回归耦合太紧, 不自觉也冠了个回归的名字(马甲无处不在). 若要求多元分类,就要把sigmoid换成大名鼎鼎的softmax了。

个人感觉逻辑回归和线性回归首先都是广义的线性回归，其次经典线性模型的优化目标函数是最小二乘，而逻辑回归则是似然函数，另外线性回归在整个实数域范围内进行预测，敏感度一致，而分类范围，需要在[0,1]。逻辑回归就是一种减小预测范围，将预测值限定为[0,1]间的一种回归模型，因而对于这类问题来说，逻辑回归的鲁棒性比线性回归的要好。

逻辑回归的模型本质上是一个线性回归模型，逻辑回归都是以线性回归为理论支持的。但线性回归模型无法做到sigmoid的非线性形式，sigmoid可以轻松处理0/1分类问题。

1. **谈谈判别式模型和生成式模型？**

* 判别方法：由数据直接学习决策函数 Y = f（X），或者由条件分布概率 P（Y|X）作为预测模型，即判别模型。
* 生成方法：由数据学习联合概率密度分布函数 P（X,Y）,然后求出条件概率分布P(Y|X)作为预测的模型，即生成模型。

由生成模型可以得到判别模型，但由判别模型得不到生成模型。

常见的判别模型有：K近邻、SVM、决策树、感知机、线性判别分析（LDA）、线性回归、传统的神经网络、逻辑斯蒂回归、boosting、条件随机场

常见的生成模型有：朴素贝叶斯、隐马尔可夫模型、高斯混合模型、文档主题生成模型（LDA）、限制玻尔兹曼机

1. **L1和L2的区别**

L1范数（L1 norm）是指向量中各个元素绝对值之和，也有个美称叫“稀疏规则算子”（Lasso regularization）。

比如 向量A=[1，-1，3]， 那么A的L1范数为 |1|+|-1|+|3|=5.

简单总结一下就是：

L1范数: 为x向量各个元素绝对值之和。

L2范数: 为x向量各个元素平方和的1/2次方，L2范数又称欧几里德范数或者Frobenius范数

Lp范数: 为x向量各个元素绝对值p次方和的1/p次方.

在支持向量机学习过程中，L1范数实际是一种对于成本函数求解最优的过程，因此，L1范数正则化通过向成本函数中添加L1范数，使得学习得到的结果满足稀疏化，从而方便人类提取特征。

L1范数可以使权值稀疏，方便特征提取。

L2范数可以防止过拟合，提升模型的泛化能力。

L1和L2的差别，为什么一个让绝对值最小，一个让平方最小，会有那么大的差别呢？看导数一个是1一个是w便知, 在靠进零附近, L1以匀速下降到零, 而L2则完全停下来了. 这说明L1是将不重要的特征(或者说, 重要性不在一个数量级上)尽快剔除, L2则是把特征贡献尽量压缩最小但不至于为零. 两者一起作用, 就是把重要性在一个数量级(重要性最高的)的那些特征一起平等共事(简言之, 不养闲人也不要超人)。

1. **LSTM结构推导，为什么比RNN好？**

推导forget gate，input gate，cell state， hidden information等的变化；因为LSTM有进有出且当前的cell informaton是通过input gate控制之后叠加的，RNN是叠乘，因此LSTM可以防止梯度消失或者爆炸

1. **拼写检查,根据谷歌一员工写的文章显示，Google的拼写检查基于贝叶斯方法。请说说的你的理解，具体Google是怎么利用贝叶斯方法，实现"拼写检查"的功能**。

用户输入一个单词时，可能拼写正确，也可能拼写错误。如果把拼写正确的情况记做c（代表correct），拼写错误的情况记做w（代表wrong），那么"拼写检查"要做的事情就是：在发生w的情况下，试图推断出c, https://img-blog.csdn.net/20141112230745208.换言之：已知w，然后在若干个备选方案中，找出可能性最大的那个c，也就是求的最大值。

    而根据贝叶斯定理，有：

https://img-blog.csdn.net/20141112230845578

由于对于所有备选的c来说，对应的都是同一个w，所以它们的P(w)是相同的，因此我们只要最大化:  
https://img-blog.csdn.net/20141112231004421

* P(c)表示某个正确的词的出现"概率"，它可以用"频率"代替。如果我们有一个足够大的文本库，那么这个文本库中每个单词的出现频率，就相当于它的发生概率。某个词的出现频率越高，P(c)就越大。比如在你输入一个错误的词“Julw”时，系统更倾向于去猜测你可能想输入的词是“July”，而不是“Jult”，因为“July”更常见。
* P(w|c)表示在试图拼写c的情况下，出现拼写错误w的概率。为了简化问题，假定两个单词在字形上越接近，就有越可能拼错，P(w|c)就越大。举例来说，相差一个字母的拼法，就比相差两个字母的拼法，发生概率更高。你想拼写单词July，那么错误拼成Julw（相差一个字母）的可能性，就比拼成Jullw高（相差两个字母）。值得一提的是，一般把这种问题称为“编辑距离”，

1. **归一化的类型**

1）线性归一化

      这种归一化方法比较适用在数值比较集中的情况。这种方法有个缺陷，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。实际使用中可以用经验常量值来替代max和min

2）标准差标准化

　　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，其转化函数为：

　　其中μ为所有样本数据的均值，σ为所有样本数据的标准差

3）非线性归一化

     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。需要根据数据分布的情况，决定非线性函数的曲线，比如log(V, 2)还是log(V, 10)等

1. **哪些机器学习算法不需要做归一化处理？**

概率模型不需要归一化，因为它们不关心变量的值，而是关心变量的分布和变量之间的条件概率，如决策树、rf。而像adaboost、svm、lr、KNN、KMeans之类的最优化问题就需要归一化。

1. **对于树形结构为什么不需要归一化？**

答：数值缩放，不影响分裂点位置。因为第一步都是按照特征值进行排序的，排序的顺序不变，那么所属的分支以及分裂点就不会有不同。对于线性模型，比如说LR，我有两个特征，一个是(0,1)的，一个是(0,10000)的，这样运用梯度下降时候，损失等高线是一个椭圆的形状，这样我想迭代到最优点，就需要很多次迭代，但是如果进行了归一化，那么等高线就是圆形的，那么SGD就会往原点迭代，需要的迭代次数较少。

另外，注意树模型是不能进行梯度下降的，因为树模型是阶跃的，阶跃点是不可导的，并且求导没意义，所以树模型（回归树）寻找最优点事通过寻找最优分裂点完成的。

1. **逻辑斯特回归为什么要对特征进行离散化**

在工业界，很少直接将连续值作为逻辑回归模型的特征输入，而是将连续特征离散化为一系列0、1特征交给逻辑回归模型，这样做的优势有以下几点：

* 离散特征的增加和减少都很容易，易于模型的快速迭代；
* 稀疏向量内积乘法运算速度快，计算结果方便存储，容易扩展；
* 离散化后的特征对异常数据有很强的鲁棒性：比如一个特征是年龄>30是1，否则0。如果特征没有离散化，一个异常数据“年龄300岁”会给模型造成很大的干扰；
* 逻辑回归属于广义线性模型，表达能力受限；单变量离散化为N个后，每个变量有单独的权重，相当于为模型引入了非线性，能够提升模型表达能力，加大拟合；
* 离散化后可以进行特征交叉，由M+N个变量变为M\*N个变量，进一步引入非线性，提升表达能力；
* 特征离散化后，模型会更稳定，比如如果对用户年龄离散化，20-30作为一个区间，不会因为一个用户年龄长了一岁就变成一个完全不同的人。当然处于区间相邻处的样本会刚好相反，所以怎么划分区间是门学问；
* 特征离散化以后，起到了简化了逻辑回归模型的作用，降低了模型过拟合的风险。

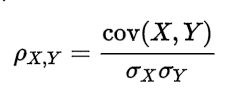
1. **熵**

香农Claude E. Shannon引入信息（熵），将其定义为离散随机事件的出现概率。一个系统越是有序，信息熵就越低；反之，一个系统越是混乱，信息熵就越高。所以说，信息熵可以被认为是系统有序化程度的一个度量。熵的定义很简单，即用来表示随机变量的不确定性。

1. **协方差和相关性有什么区别？**

相关性是协方差的标准化格式。协方差本身很难做比较。例如：如果我们计算工资（$）和年龄（岁）的协方差，因为这两个变量有不同的度量，所以我们会得到不能做比较的不同的协方差。

https://img-blog.csdn.net/20171204144855400



我们计算相关性来得到一个介于-1和1之间的值，就可以忽略它们各自不同的度量。

1. **线性分类器与非线性分类器的区别以及优劣**

线性和非线性是针对，模型参数和输入特征来讲的；比如输入x，模型y=a\*x+a\*x^2那么就是非线性模型，如果输入是x和X^2则模型是线性的。

线性分类器可解释性好，计算复杂度较低，不足之处是模型的拟合效果相对弱些。

非线性分类器效果拟合能力较强，不足之处是数据量不足容易过拟合、计算复杂度高、可解释性不好。

常见的线性分类器有：LR,贝叶斯分类，单层感知机、线性回归

常见的非线性分类器：决策树、RF、GBDT、多层感知机

SVM两种都有（看线性核还是高斯核）

1. **数据的逻辑存储结构（如数组，队列，树等）对于软件开发具有十分重要的影响，试对你所了解的各种存储结构从运行速度、存储效率和适用场合等方面进行简要地分析。 数据结构/算法**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 运行速度 | 存储效率 | 存储效率 |
| 数组 | 快 | 高 | 比较适合进行查找操作，还有像类似于矩阵等的操作 |
| 链表 | 较快 | 较高 | 比较适合增删改频繁操作，动态的分配内存 |
| 队列 | 较快 | 较高 | 比较适合进行任务类等的调度 |
| 栈 | 一般 | 较高 | 比较适合递归类程序的改写 |
| 二叉树（树） | 较快 | 一般 | 一切具有层次关系的问题都可用树来描述 |
| 图 | 一般 | 一般 | 除了像最小生成树、最短路径、拓扑排序等经典用途。还被用于像神经网络等人工智能领域等等 |

1. **贝叶斯公式**

贝叶斯公式可以直接根据条件概率的定义直接推出。即因为P(A,B) = P(A)P(B|A) = P(B)P(A|B)，所以P(A|B) = P(A)P(B|A) / P(B)

1. **说下红黑树的五个性质。**

红黑树，一种二叉查找树，但在每个结点上增加一个存储位表示结点的颜色，可以是Red或Black。

通过对任何一条从根到叶子的路径上各个结点着色方式的限制，红黑树确保没有一条路径会比其他路径长出俩倍，因而是接近平衡的。

红黑树，作为一棵二叉查找树，满足二叉查找树的一般性质。下面，来了解下 二叉查找树的一般性质。

二叉查找树，也称有序二叉树（ordered binary tree），或已排序二叉树（sorted binary tree），是指一棵空树或者具有下列性质的二叉树：

* 若任意节点的左子树不空，则左子树上所有结点的值均小于它的根结点的值；
* 若任意节点的右子树不空，则右子树上所有结点的值均大于它的根结点的值；
* 任意节点的左、右子树也分别为二叉查找树。
* 没有键值相等的节点（no duplicate nodes）。

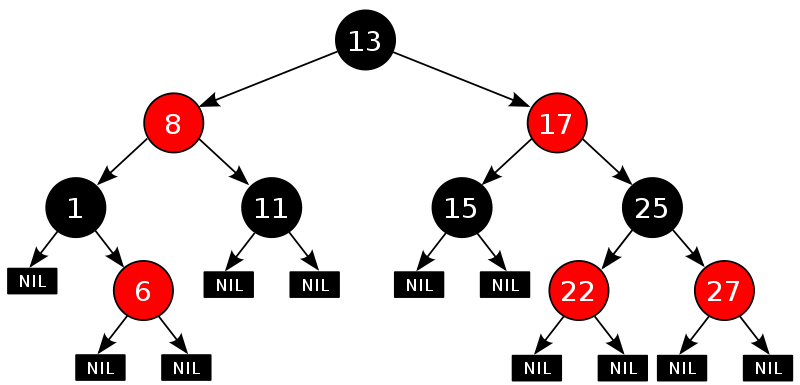
因为一棵由n个结点随机构造的二叉查找树的高度为lgn，所以顺理成章，二叉查找树的一般操作的执行时间为O(lgn)。但二叉查找树若退化成了一棵具有n个结点的线性链后，则这些操作最坏情况运行时间为O(n)。

红黑树虽然本质上是一棵二叉查找树，但它在二叉查找树的基础上增加了着色和相关的性质使得红黑树相对平衡，从而保证了红黑树的查找、插入、删除的时间复杂度最坏为O(log n)。

但它是如何保证一棵n个结点的红黑树的高度始终保持在logn的呢？这就引出了红黑树的5个性质：

* 每个结点要么是红的要么是黑的。
* 根结点是黑的。
* 每个叶结点（叶结点即指树尾端NIL指针或NULL结点）都是黑的。
* 如果一个结点是红的，那么它的两个儿子都是黑的。
* 对于任意结点而言，其到叶结点树尾端NIL指针的每条路径都包含相同数目的黑结点。

正是红黑树的这5条性质，使一棵n个结点的红黑树始终保持了logn的高度，从而也就解释了上面所说的“红黑树的查找、插入、删除的时间复杂度最坏为O(log n)”这一结论成立的原因。

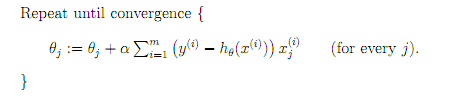


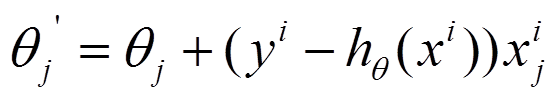
1. **激活函数**

常用的非线性激活函数有sigmoid、tanh、relu等等，前两者sigmoid/tanh比较常见于全连接层，后者relu常见于卷积层

sigmod函数，是逻辑斯蒂回归的压缩函数，它的性质是可以把分隔平面压缩到[0,1]区间一个数（向量），在线性分割平面值为0时候正好对应sigmod值为0.5，大于0对应sigmod值大于0.5、小于0对应sigmod值小于0.5；0.5可以作为分类的阀值；exp的形式最值求解时候比较方便，用相乘形式作为logistic损失函数，使得损失函数是凸函数；不足之处是sigmod函数在y趋于0或1时候有死区，控制不好在bp形式传递loss时候容易造成梯度弥撒。

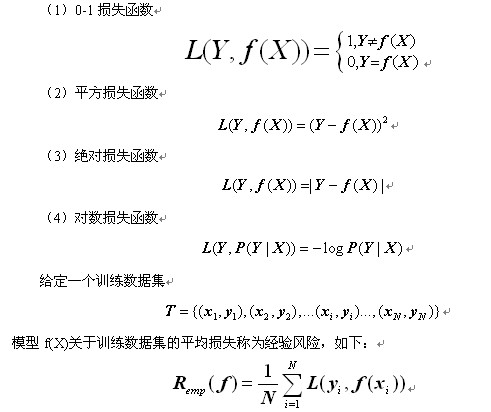
1. **梯度下降和随机梯度下降**





i是样本编号下标，j是样本维数下标，m为样例数目，n为特征数目。所以更新一个θj只需要一个样本就可以

1. **说说常见的损失函数？**



HashMap基于Hashtable实现，不同之处在于HashMap是非同步的，并且允许null，即null value和null key，Hashtable则不允许null，

此外，记住一点：hashmap/hashset等凡是带有hash字眼的均基于hashtable实现，没带hash字眼的如set/map均是基于红黑树实现，

1. **以下哪些方法不可以直接来对文本分类？**

A、Kmeans

B、决策树

C、支持向量机

D、KNN

正确答案: A分类不同于聚类。

A：Kmeans是聚类方法，典型的无监督学习方法。分类是监督学习方法，BCD都是常见的分类方法。

1. **关于logit 回归和SVM 不正确的是（A）**
2. Logit回归本质上是一种根据样本对权值进行极大似然估计的方法，而后验概率正比于先验概率和似然函数的乘积

logit仅仅是最大化似然函数，并没有最大化后验概率，更谈不上最小化后验概率。A错误

B. Logit回归的输出就是样本属于正类别的几率，可以计算出概率，正确

C. SVM的目标是找到使得训练数据尽可能分开且分类间隔最大的超平面，应该属于结构风险最小化。

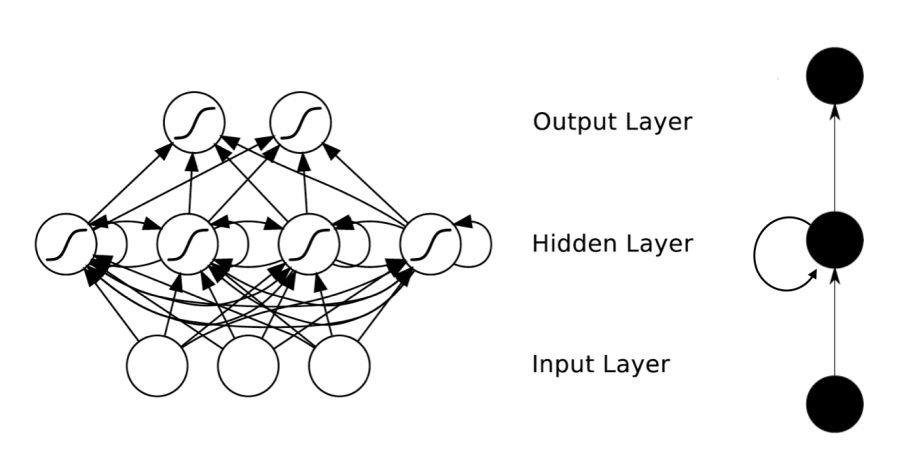
D. SVM可以通过正则化系数控制模型的复杂度，避免过拟合。

Logit回归可以用于预测事件发生概率的大小，SVM目标是结构风险最小化，SVM可以有效避免模型过拟合。

极大似然估计就是各个相互独立的样本后验概率的乘积吧~是错在最小化~

逻辑回归的目标是最大化似然函数，而不是最大或最小化后验概率

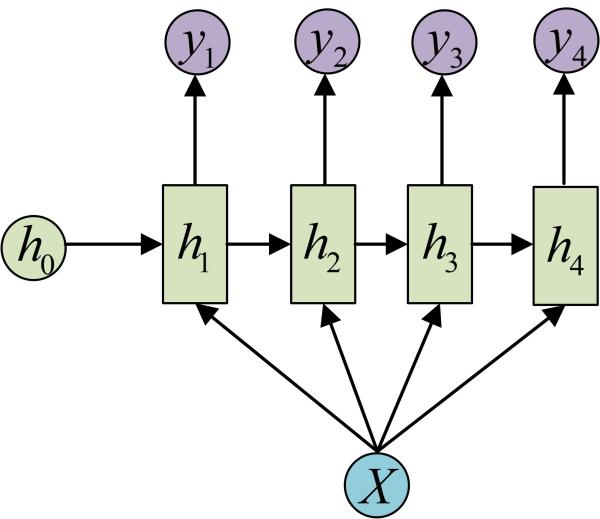
1. **什么是RNN？**





隐藏层之间的节点不再无连接而是有连接的，并且隐藏层的输入不仅包括输入层的输出还包括上一时刻隐藏层的输出

在这个间隔不断增大时，RNN 会丧失学习到连接如此远的信息的能力。



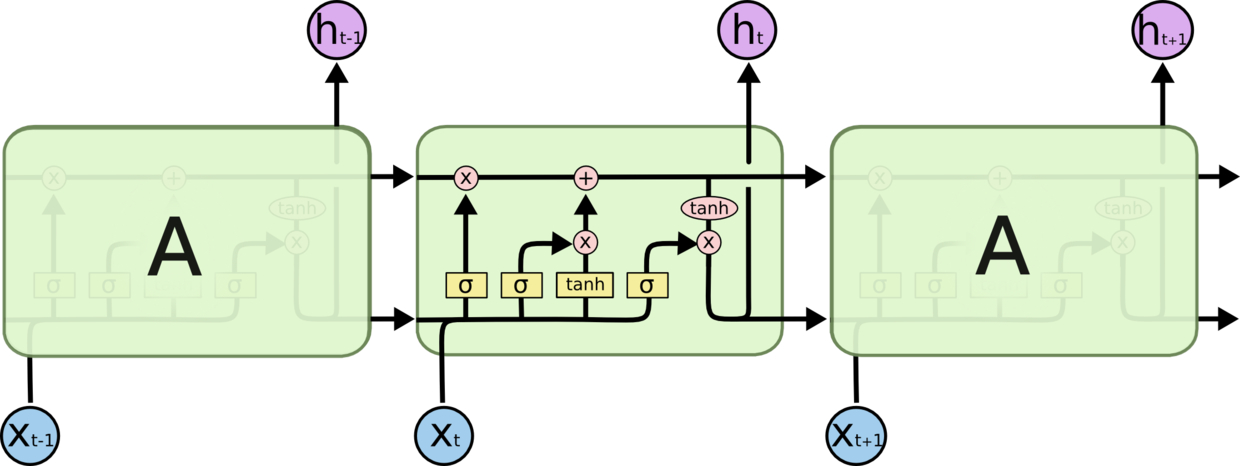
1. **如何解决RNN梯度爆炸和弥散的问题？**

为了解决梯度爆炸问题，Thomas Mikolov首先提出了一个简单的启发性的解决方案，就是当梯度大于一定阈值的的时候，将它截断为一个较小的数。

解决梯度弥散的问题，我们介绍了两种方法。第一种方法是将随机初始化改为一个有关联的矩阵初始化。第二种方法是使用ReLU（Rectified Linear Units）代替sigmoid函数。ReLU的导数不是0就是1.因此，神经元的梯度将始终为1，而不会当梯度传播了一定时间之后变小

1. **LSTM 网络**

Long Short Term 网络,一般就叫做 LSTM,是一种 RNN 特殊的类型，可以学习长期依赖信息。LSTM和基线RNN并没有特别大的结构不同，但是它们用了不同的函数来计算隐状态。LSTM的“记忆”我们叫做细胞/cells，你可以直接把它们想做黑盒，这个黑盒的输入为前状态和当前输入。这些“细胞”会决定哪些之前的信息和状态需要保留/记住，而哪些要被抹去。实际的应用中发现，这种方式可以有效地保存很长时间之前的关联信息。



* RNN引入了循环的概念，但是在实际过程中却出现了初始信息随时间消失的问题，即长期依赖（Long-Term Dependencies）问题，所以引入了LSTM。
* LSTM：因为LSTM有进有出且当前的cell informaton是通过input gate控制之后叠加的，RNN是叠乘，因此LSTM可以防止梯度消失或者爆炸的变化是关键

1. **标准化与归一化的区别？**

归一化方法：

1、把数变为（0，1）之间的小数主要是为了数据处理方便提出来的，把数据映射到0～1范围之内处理，更加便捷快速。

把有量纲表达式变为无量纲表达式 归一化是一种简化计算的方式，即将有量纲的表达式，经过变换，化为无量纲的表达式，成为纯量。

标准化方法：数据的标准化是将数据按比例缩放，使之落入一个小的特定区间。由于信用指标体系的各个指标度量单位是不同的，为了能够将指标参与评价计算，需要对指标进行规范化处理，通过函数变换将其数值映射到某个数值区间。

归一化，一般的方法是 (x-min(x))/(max(x)-min(x))

标准化，一般方法是 (x-mean(x))/std(x)

如果采用标准化，不改变样本在这两个维度上的分布，则左图还是会保持二维分布的一个扁平性；而采用归一化则会在不同维度上对数据进行不同的伸缩变化（归一区间，会改变数据的原始距离，分布，信息），使得其呈类圆形。虽然这样样本会失去原始的信息，但这防止了归一化前直接对原始数据进行梯度下降类似的优化算法时最终解被数值大的特征所主导。归一化之后，各个特征对目标函数的影响权重是一致的。这样的好处是在提高迭代求解的精度。

1. **为什么LSTM模型中既存在sigmoid又存在tanh两种激活函数？**

二者目的不一样

* sigmoid 用在了各种gate上，产生0~1之间的值，这个一般只有sigmoid最直接
* tanh 用在了状态和输出上，是对数据的处理，这个用其他激活函数或许也可以

**问题 页码 难度级 笔记**

**第1章　特征工程**

**为什么需要对数值类型的特征做归一化？ 002 ★☆☆☆☆**

**怎样处理类别型特征？ 004 ★★☆☆☆**

**什么是组合特征？如何处理高维组合特征？ 006 ★★☆☆☆**

**怎样有效地找到组合特征？ 009 ★★☆☆☆**

**有哪些文本表示模型？它们各有什么优缺点？ 011 ★★☆☆☆**

**如何缓解图像分类任务中训练数据不足带来的问题？ 016 ★★☆☆☆**

**Word2Vec是如何工作的？它和隐狄利克雷模型有什么区别与联系？ 013 ★★★☆☆**

**第2章　模型评估**

**准确率的局限性。 022 ★☆☆☆☆**

**精确率与召回率的权衡。 023 ★☆☆☆☆**

**平方根误差的“意外”。 025 ★☆☆☆☆**

**什么是ROC曲线？ 027 ★☆☆☆☆**

**为什么要进行在线A/B测试？ 037 ★☆☆☆☆**

**如何进行线上A/B测试？ 038 ★☆☆☆☆**

**过拟合和欠拟合具体是指什么现象？ 045 ★☆☆☆☆**

**如何绘制ROC曲线？ 028 ★★☆☆☆**

**如何计算AUC？ 030 ★★☆☆☆**

**为什么在一些场景中要使用余弦相似度而不是欧氏距离？ 033 ★★☆☆☆**

**18**

**如何划分实验组和对照组？ 038 ★★☆☆☆**

**模型评估过程中的验证方法及其优缺点。 040 ★★☆☆☆**

**能否说出几种降低过拟合和欠拟合风险的方法？ 046 ★★☆☆☆**

**ROC曲线相比P-R曲线有什么特点？ 030 ★★★☆☆**

**余弦距离是否是一个严格定义的距离？ 034 ★★★☆☆**

**自助法采样在极限情况下会有多少数据从未被选择过？ 041 ★★★☆☆**

**超参数有哪些调优方法？ 043 ★★★☆☆**

**第3章　经典算法**

**逻辑回归相比线性回归，有何异同？ 058 ★★☆☆☆**

**决策树有哪些常用的启发函数？ 062 ★★☆☆☆**

**线性可分的两类点在SVM分类超平面上的投影仍然线性可分吗？ 051 ★★★☆☆**

**证明存在一组参数使得高斯核SVM的训练误差为0。 054 ★★★☆☆**

**加入松弛变量的SVM的训练误差可以为0吗？ 056 ★★★☆☆**

**用逻辑回归处理多标签分类任务的一些相关问题。 059 ★★★☆☆**

**如何对决策树进行剪枝？ 067 ★★★☆☆**

**训练误差为0的SVM分类器一定存在吗？ 055 ★★★★☆**

**第4章　降维**

**从最大方差的角度定义PCA的目标函数并给出求解方法。 074 ★★☆☆☆**

**从回归的角度定义PCA的目标函数并给出对应的求解方法。 078 ★★☆☆☆**

**线性判别分析的目标函数以及求解方法。 083 ★★☆☆☆**

**线性判别分析与主成分分析的区别与联系 086 ★★☆☆☆**

**第5章　非监督学习**

**K均值聚类算法的步骤是什么？ 093 ★★☆☆☆**

**高斯混合模型的核心思想是什么？它是如何迭代计算的？ 103 ★★☆☆☆**

**K均值聚类的优缺点是什么？如何对其进行调优？ 094 ★★★☆☆**

**针对K均值聚类的缺点，有哪些改进的模型？ 097 ★★★☆☆**

**自组织映射神经网络是如何工作的？它与K均值算法有何区别？ 106 ★★★☆☆**

**怎样设计自组织映射神经网络并设定网络训练参数？ 109 ★★★☆☆**

**以聚类算法为例，如何区分两个非监督学习算法的优劣？ 111 ★★★☆☆**

**证明K均值聚类算法的收敛性。 099 ★★★★☆**

**第6章　概率图模型**

**写出图6.1（a）中贝叶斯网络的联合概率分布。 118 ★☆☆☆☆**

**写出图6.1（b）中马尔可夫网络的联合概率分布。 119 ★☆☆☆☆**

**解释朴素贝叶斯模型的原理，并给出概率图模型表示。 121 ★★☆☆☆**

**解释最大熵模型的原理，并给出概率图模型表示。 122 ★★☆☆☆**

**常见的主题模型有哪些？试介绍其原理。 133 ★★☆☆☆**

**如何确定LDA模型中的主题个数？ 136 ★★☆☆☆**

**常见的概率图模型中，哪些是生成式的，哪些是判别式的？ 125 ★★★☆☆**

**如何对中文分词问题用隐马尔可夫模型进行建模和训练？ 128 ★★★☆☆**

**如何用主题模型解决推荐系统中的冷启动问题？ 137 ★★★☆☆**

**最大熵马尔可夫模型为什么会产生标注偏置问题？如何解决？ 129 ★★★★☆**

**第7章　优化算法**

**有监督学习涉及的损失函数有哪些？ 142 ★☆☆☆☆**

**训练数据量特别大时经典梯度法存在的问题，如何改进？ 155 ★☆☆☆☆**

**机器学习中哪些是凸优化问题？哪些是非凸优化问题？ 145 ★★☆☆☆**

**无约束优化问题的求解。 148 ★★☆☆☆**

**随机梯度下降法失效的原因。 158 ★★☆☆☆**

**如何验证求目标函数梯度功能的正确性？ 152 ★★★☆☆**

**随机梯度下降法的一些变种。 160 ★★★☆☆**

**L1正则化使得模型参数具有稀疏性的原理是什么？ 164 ★★★☆☆**

**第8章　采样**

**如何编程实现均匀分布随机数生成器？ 174 ★☆☆☆☆**

**19**

**简述MCMC采样法的主要思想。 185 ★☆☆☆☆**

**举例说明采样在机器学习中的应用。 172 ★★☆☆☆**

**简单介绍几种常见的MCMC采样法。 186 ★★☆☆☆**

**MCMC采样法如何得到相互独立的样本？ 187 ★★☆☆☆**

**简述一些常见的采样方法的主要思想和具体操作。 176 ★★★☆☆**

**如何对高斯分布进行采样？ 180 ★★★☆☆**

**如何对贝叶斯网络进行采样？ 190 ★★★☆☆**

**当训练集中正负样本不均衡时，如何处理数据以更好地训练分类模型？ 194 ★★★☆☆**

**第9章　前向神经网络**

**写出常用激活函数及其导数。 207 ★☆☆☆☆**

**神经网络训练时是否可以将参数全部初始化为0？ 217 ★☆☆☆☆**

**多层感知机表示异或逻辑时最少需要几个隐层？ 200 ★★☆☆☆**

**为什么Sigmoid和Tanh激活 函数会导致梯度消失的现象？ 208 ★★☆☆☆**

**写出多层感知机的平方误差和交叉熵损失函数。 212 ★★☆☆☆**

**解释卷积操作中的稀疏交互和参数共享及其作用。 223 ★★☆☆☆**

**一个隐层需要多少隐节点能够实现包含n元输入的任意布尔函数？ 203 ★★★☆☆**

**多个隐层实现包含n元输入的任意布尔函数最少需要多少个节点和网络层？ 205 ★★★☆☆**

**ReLU系列的激活函数的优点是什么？他们有什么局限性以及如何改进？ 209 ★★★☆☆**

**平方误差损失函数和交叉熵损失函数分别适合什么场景？ 214 ★★★☆☆**

**为什么Dropout可以抑制过拟合？简述它的工作原理和实现？ 218 ★★★☆☆**

**批量归一化的基本动机与原理是什么？在卷积神经网络中如何使用？ 220 ★★★☆☆**

**常用的池化操作有哪些？池化的作用是什么？ 225 ★★★☆☆**

**卷积神经网络如何用于文本分类任务？ 227 ★★★☆☆**

**ResNet的提出背景和核心理论是什么？ 230 ★★★☆☆**

**根据损失函数推导各层参数更新的梯度计算公式。 212 ★★★★☆**

**第10章　循环神经网络**

**循环神经网络与前馈神经网络相比有什么特点？ 236 ★☆☆☆☆**

**循环神经网络为什么会出现梯度消失或梯度爆炸？有哪些改进方案？ 238 ★★☆☆☆**

**LSTM是如何实现长短期记忆功能的？ 243 ★★☆☆☆**

**什么是Seq2Seq模型？它有哪些优点？ 247 ★★☆☆☆**

**在循环神经网络中能否使用ReLU作为激活函数？ 241 ★★★☆☆**

**LSTM里各模块分别使用什么激活函数？可以用其它的激活函数吗？ 245 ★★★☆☆**

**Seq2Seq模型在解码时有哪些常用的方法？ 249 ★★★☆☆**

**Seq2Seq模型引入注意力机制是为了解决什么问题？为什么选用双向循环神经模型？ 251 ★★★★☆**

**第11章　强化学习**

**强化学习中有哪些基本概念？ 258 ★☆☆☆☆**

**从价值迭代来考虑，如何找到图中马里奥的一条最优路线？ 260 ★★☆☆☆**

**从策略迭代来考虑，如何找到图中马里奥的一条最优路线？ 261 ★★☆☆☆**

**什么是深度强化学习？它与传统的强化学习有什么不同？ 264 ★★★☆☆**

**在智能体与环境的交互中，什么是探索和利用？如何平衡探索与利用？ 272 ★★★☆☆**

**什么是策略梯度下降？与传统Q-learning有什么不同？有什么优势？ 268 ★★★★☆**