**KNN**

KNN算法核心：假如A在1类中，那么如果B与A在某些特征上非常接近，那么B很有可能也在1类中

KNN 应用实例：

1.判断0-9的手写体

2.用于约会网站的配对效果。根据一些人的特征判断对这个人的喜欢程度。例如提取一个人的特征如下：

1. 每个星期的飞行里程数
2. 每个星期的冰激凌消费数
3. 每个星期电子游戏所占的百分比

根据这些特征，将对一个人的喜爱程度分为3类：特别喜爱，喜爱和讨厌

优点：精度高，对异常值不敏感。

缺点：计算复杂度高，空间复杂度高

**决策树：**

构造过程：通过某种算法（ID3，C4.5等）利用样本的特征的取值将样本构建成一棵树。这棵树的枝干就是特征值的取值。树的叶子就是最后的分类。树的节点就是在取一个特征值的情况下样本变成了一个什么样子。

优点：计算复杂度不高，计算结果易于理解，可以处理不相关特征的数据。

缺点：可能会产生过度匹配的问题。只能拥有标称型数据，即数据必须离散化

这里需要明白一个概念就是：信息增益。信息增益是指使用某个特征对信息进行划分后，数据信息量划分前后的变化。而决策树总是想通过能取得最大的信息增益的特征来对树进行划分。而计算信息增益的算法有很多。所以构建决策树的算法也有很多。

决策树的构造是一个很耗时的任务，所以每次构造完决策树以后可以将他序列化然后存到硬盘里，这样就不用每次再来构造了。所以构造好的决策树其实是一个确定的模型。当我们一进行预测时，只需要遍历这个树就可以了。但是决策树构造的过程是一个不确定的过程。因为他每次都需要先测量数据的不一致性，也就是熵来寻找最优化的方案来划分数据集。所以随着数据量的增加，熵是变化的。即划分数据的方案是变化的，那么决策树就是变化的，预测结果也会相应的发生变化。

决策树有个很大的问题就是过度匹配问题，即决策树可能会产生过多的数据集划分。为了消除这个问题，我们就需要修建决策树，合并相伶的无法产生大量信息增益的叶节点。

应用实例：  
医生通过眼球的状况，近视的程度等等来给近视患者推荐适合他们的隐形眼镜（软材料，硬材料和不适合佩戴）。

**3.朴素贝叶斯**

优点：在数据量较小的情况下仍然起作用。

缺点：对输入数据的准备方式较为敏感

朴素贝叶斯是基于俩个假设的：

1.特征之间是独立的。例如一个特征是：delicious 后边出现的词，显然bacon要比shit 更加合理，但是朴素贝叶斯理论会认为特征相互独立。

2.理论认为所有的特征同等重要。

朴素贝叶斯有俩种实现方式：

1.贝努力模型的实现。这种实现是不考虑文档中词出现的频率。

2.另一种实现是多项式实现，这种实现是要考虑词出现的频率

用朴素贝叶斯对文档进行分析的第一个步骤就是词向量的构建。这个东西其实做起来也很简单。举个例子：假如我们想要通过某一些词对用户评论进行分类，来确定用户评论是否是恶意的辱骂。现在我先把一些可能涉及到辱骂的词构建一个数据字典。例如

Mot = {stupid, con, conard, penis ……} 那么当我拿到要给文档需要对这个文档的评论进行分类时，我想创建一个和数据字典一样大的数组[0,0,0,0] 。假如我在文档里面遇到了一个stupid这个词，那么我就把数组对应与stupid 位置的0 改成1。如果我们用的是多项式模型来实现朴素贝叶斯，那么当我第二次遇到stupid的时候我需要把改成2 。当然如果用伯努利那就无所谓了。当我遍历完这个文本，就可得到一个mot每个字符是否出现的向量[]。例如stupid,con,connar,penis 分别出现了1 2 0 2 次，那么这个向量就应该是[1 2 0 2]。那么这个向量我们就叫做词向量。

第二个步骤就是通过计算概率来对某个预测事件进行分组。并且我们可以得到这个事件落到这组的概率。计算公式如下：

P (|w) = , 这个东西其实一点都不高端，就是大学学过的条件概率的公式。通俗的讲就是说 w 是我们已知的预测事件的特征值的集合，问题就是假如我们在w情况下，我们把事件分到类i的概率。右面就是转化成的条件概率。

因为我们前面假设过朴素贝叶斯是基于特征值相互独立的。所以

= \* \* \* …\* 每个代表一个事件的第i个特征。 所以右面的公式的每一项其实都是可以算出来的。那么左边就是可解的。

朴素贝叶斯经常会遇到想下溢出的问题即 由于数值太小，计算机直接取0. 对于这种问题解决办法是取对数然后相加。

朴素贝叶斯应用实例：

1.用于文本的分析： 例如评论是否是恶意的辱骂，垃圾邮件的过滤，根据社交网络上要的某些言语推测某个人性格，兴趣爱好，喜欢的异性的特点等等

**4. Logistics 回归**

Logistics 是个线性回归这种回归用于把点分成2份。我们用0表示第一份，用1表示第二份。假如我们想用一个函数来进行分类。记为y = f(x) 。 当我给定一个x的值y = 1 或者 y=0 . 那么这样就通过这个函数将x分类1类或者2类。

自然的我们就想到了这个函数。,但是我们有要想让这个函数连续，而不是出现瞬间跳跃的情况。因为那种跳跃的情况很难处理。所以我们选择了Sigmoid 函数来进行处理。函数的图像如下：

如果我们放大横坐标的标尺。可以把这个函数看作是直接跳跃上去的。现在我们有了能够进行0 ，1 分类的函数。即我们知道了f（x）。 但是x我们输入什么呢？这里我们就要用到Logistics回归。首先它是一种线性回归。实线性关系的表达能力非常强大，每个特征对结果的影响强弱可以由前面的参数体现，而且每个特征变量可以首先映射到一个函数，然后再参与线性计算。这样就可以表达特征与结果之间的非线性关系。



例如我们用x1 表示特征1 。 x2 表示特征2 。ceta 表示权重。Logistics回归的根本问题其实就是怎样获得每个ceta的值。即每个特征的权重。

那这个就是通过大量的样本训练得到的。 所以机器学习的过程就是不断地就成每个ceta的值，知道他与现实情况无线的接近。那么，判断他与现实情况是否接近依据什么呢？

依据的公式如下：

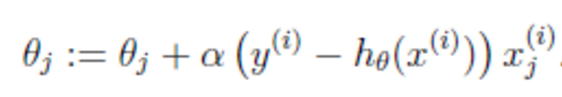


依据的就是J(ceta) 。 举个例子，假如x1 代表房子的颜色，x2代表房子的大小，y 代表房子的真实价格。 H(x) 代表我们给ceta1和ceta2取值以后的函数。然后我们按照这个函数把所有样本的房子的颜色x1和房子的大小x2输入到H(x) 中，那么我们就可以得到一个H(x)的值。最后求方差即可。所以这个机器学习的过程其实就是我们不断的对ceta进行复制的过程。最后我们找到一组保证J(ceta)最小的ceta值即可。那么问题来了，ceta可以有有无数个取值，我到底取哪一个呢？

所以在ceta取值这里才是这个算法的精华所在。现在人们最常用的取值方法有以下几种：  
（1）梯度下降法

（2）最小二乘法

梯度下降法就是改变θ的值，使得J(θ)按梯度下降的方向进行减少。梯度方向由J(θ)对θ的偏导数确定，由于求的是极小值，因此梯度方向是偏导数的反方向。结果为



注意这里的ceta 是有方向的。为了理解梯度下降法举个简单的例子：假如你在直升机上，然后把你空投到一个山区里，空投的地点是随机的，但是你的任务是到山脚。步骤如下：

（1）空投以后你随机的落到一座大山的某个地方，这就相当于我们给ceta赋值。

（2）然后你从任何一个方向都可以走，但如果你不傻，要下山的话肯定是要向下走。那么cata也是他先确定一个方向，然后试探一下如果这个方向走一步，值是否减小。他往减少最多的方向走。

（3）你在大山里，终于走到了山脚，可能你觉得是山脚，其实如果你翻越另一座山，还有更低的山脚。Ceta也是，通过不断的试探，下降。最终他可能求得的不是最小值，而是局部极小值。

另一种直接计算结果的方法是最小二乘法，将训练特征表示为X矩阵，结果表示成y向量，仍然是线性回归模型，误差函数不变。那么θ可以直接由下面公式得出



应用实例：

因为这是一个线性的模型。所以这个模型最好用于各种特征值是相互独立的，例如房价。如果对房价是否跌进行预测。房价基本就可以表示成一个线性模型，例如

房价受房子面积，房子颜色，房子地段等等的影响，但是这些影响相互没有什么关系。

**支持向量机：**

首先支持向量机像决策树一样是一种分类算法，并且这种分类算法是经过严格数学推倒的严谨的分类算法

这里只为了搞清楚他是怎么回事。数学的推倒全部省略。

SVM是为了干嘛呢？他是为了找到一个线性分类器，而线性分类器可以干嘛？他可以在一个N维空间中找到一个

超平面，例如：

wTx+b=0

那这个超平面就可以把n维空间上的点进行分类。

如果是线性可分的，例如一个2维空间就可以用一条直线来进行分开，三维那就需要一个平面，N维那就需要一个n-1维的东西来分，这个n-1维的东西就是我刚才提到的超平面。现在我们想象一个美好的世界。即所有的超平面都是线性可分的。那么我们总能通过某种方法找到一个超平面把这些点分开。但是能分开的这些点绝对不止一个超平面，那么那么多超平面我们使用哪个把他们分开呢？即我们怎么找到最优化的超平面。显然，最优化的超平面就是让超平面离距离他最近的那些点的距离最远。

显然长的直线比短的直线要更好

而距离他们最近的这些点就叫做支持向量。怎么找那个超平面呢 。首先就是先对支持向量内每个点到直线进行的距离进行求值，最后将这些点的距离相加就是总的距离，那么我们的任务就是让这个总距离最小。求这个总距离比较复杂，用到了著名的拉格朗日乘子。这里就省略了，因为我也不是很会，而且即使弄懂了因为步骤特别多过几天又忘了所以收益不大，。然后就求出了这个超平面。

等等，问题其实还没有解决，因为刚才我们是基于一个假设的，即：空间上的样本是线性可分的。

难道一个空间上的点总是存在超平面把空间上的点进行分类么？

答案是否定的，好那么问题来了？既然不存在那SVM有个鸟用？不要急。我们有方法来对N为空间的点进行映射，映射到更高维就存在超平面了。为啥？我也不知道，反正可以经过严格的数学公式证明的。但是如果直接进行映射，那个维度增长的有点吓人啊。有多吓人

例如：要保证3维空间总是找到超平面，那么得投射到19唯的空间里。这尼玛不是都比么?这个时候Kernel核函数就来了。这个东西非常牛逼，他可以先在低维运算，运算完了在投射到高维，这样数据量就不是很大了。那kernel核函数是啥？这个就是根据经验的得到的函数我们拿着直接用就好了。所以我们总能找到这样一个超平面来对空间的点进行分类了。

常见的kernel函数如下:

线性内核，多项式内核，径向基内核（RBF），sigmoid核

Linear核：主要用于线性可分的情形。参数少，速度快

RBF核：主要用于线性不可分的情形。参数多，分类结果非常依赖于参数

吴恩达老师对于使用核函数总结如下：

1. 如果Feature的数量很大，跟样本数量差不多，这时候选用LR或者是Linear Kernel的SVM

2. 如果Feature的数量比较小，样本数量一般，不算大也不算小，选用SVM+Gaussian Kernel

3. 如果Feature的数量比较小，而样本数量很多，需要手工添加一些feature变成第一种情况

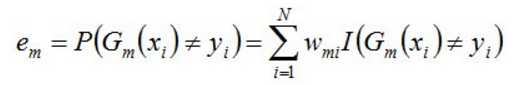
到这里为止，支持向量机就搞定了一大半，当然剩下没有明白的就是那些数学证明了。

**Adaboosting：**

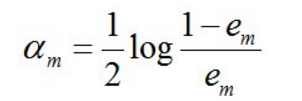
Adaboosting：我觉得Adaboosting严格的说不是一个机器学习的算法，他的想法很简单就是用多个弱分类器组成一个强分类器。通俗地来讲就是三个臭皮匠胜过一个诸葛亮。但是臭皮匠要想胜过诸葛亮那么他们得相互配合。否则10个臭皮匠也胜不过诸葛亮

Adaboosting的算法如下：

1.用一个弱分类器对样本进行分类。假如对样本进行分类后得到的误差值为pi。因为臭皮匠其实也有聪明和笨之分，所以在臭皮匠分类的时候我们会更加倾向于聪明的臭皮匠得到的分类结果，那怎么知道臭皮匠是聪明还是笨呢？误差值小的显然就是聪明的臭皮匠。所以我们要让聪明的臭皮匠得到的分类器的权值大一点。权值的计算方法如下：



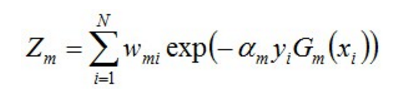
由误差率计算弱分类器m在最后强分类器的权重：



2.那么臭皮匠怎么相互配合呢？他们配合的方式就是修改样本点的权重，假如臭皮匠m用他的臭方法对样本点进行了分类。当分类完以后他发现有很多点没有分对，那么他把这些点的权重修改掉。在初始情况下，每个点的权重都是1/N, 其中N就是样本里数据点的个数。那么臭皮匠把自己没有搞定的点的权重增加以后告诉下一个臭皮匠这些点我没有搞定，你来搞定。下一个臭皮匠又会尝试搞定这些难搞定的点。所以臭皮匠赢诸葛亮的策略就是尽可能让更多的臭皮匠聚焦那些难搞定的点。说不定有哪一个灵光一闪就搞定了。所以臭皮匠之间用的分类方法一定是不一样的。

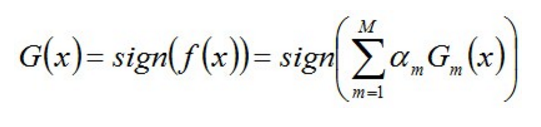
更新每个点的权重的计算方法如下：





3.最后利用误差率得到每个臭皮匠的分类器在总的分类器的权重，最后强分类器就是臭皮匠得到的这些弱分类器的加权平均

如下：



优点：

　　1)adaboost是一种有很高精度的分类器

　　2)可以使用各种方法构建子分类器，adaboost算法提供的是框架

　　3)当使用简单分类器时，计算出的结果是可以理解的。而且弱分类器构造极其简单

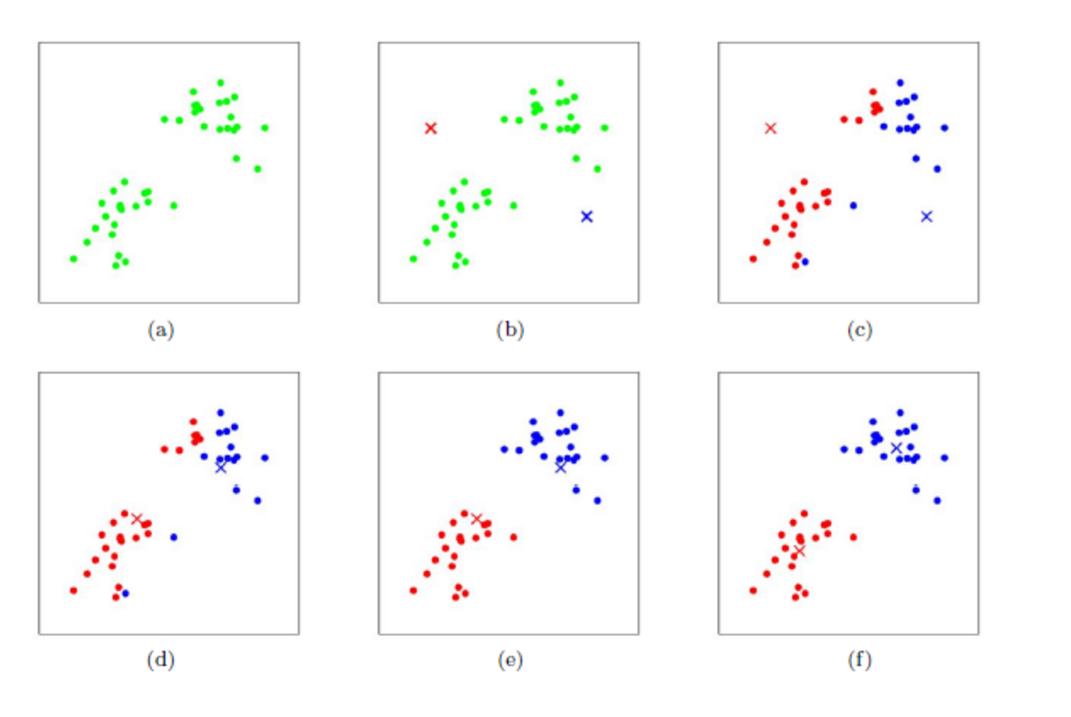
　　4)简单，不用做特征筛选

　　5)不用担心overfitting！

**K-Means 算法**

首先这是一个聚类算法，即我们预先是不给出样本点的类别的，然后样本通过某个聚类算法把和自己一样的样本聚在一起形成一个分类，这就是聚类。

聚类算法最近点就是给星空中的星星进行分类了。为了说明聚类算法，我们也用这个例子来说明。



首先我们给出一共要分成多少个类。假如绿色的是天空中的星星，我们要把星星分成俩个类。K-mean 算法最核心的是类的质心。一个类的质心就是对这个类里面所有星星的坐标求平均。假如我们想把这些星星分成俩类，我们随机选俩个质心。即b图中等的红点和蓝点。

K-mean 对所有样本的分类标准是样本离那个类的质心近他就归到那个类。所有如果将b的红点和蓝点作为质心的话c图就是所有点的分类结果。当我们进行了一次分类后，再对类里所有样本求质心，就会得到新的质心。如图d。然后利用新的质心再对所有的样本进行分类，如图e。重复上面来个步骤，发现质心不在移动。此时我们就认为分类完毕。

那有个问题？会不会质心一直在变，那永远不会分类完毕。这种情况是不存在的。有严格的数学公式证明过这个绝对是收敛的。

K mean 算法优缺点：

优点：

收敛速度快

如果数据比较好的话，聚类效果还不错

容易理解

缺点：

需要预先知道k的值 。对异常值敏感

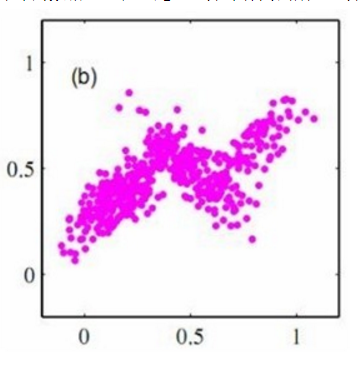
**混合高斯模型：**

混合高斯模型是一个比较难理解的模型，用到了大量的数学知识。为了理解混合高斯模型，需要先来理解。单一高斯模型。其实我们对一维的单一高斯模型是非常熟悉的。因为他就是非常著名的正态分布。自然界的很多东西都遵循正态分布，例如身高，考试成绩等等。而单一高斯模型就是多维的正态分布而已。公式如下：



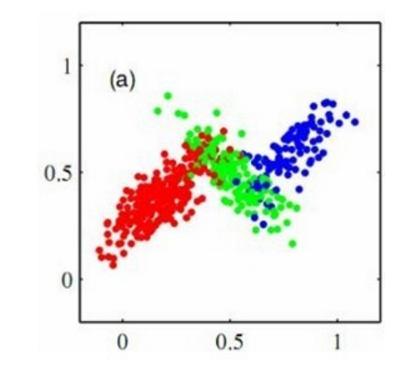
如果用正态分布对数据进行分类，其实就是简单的给定一个阈值。例如0.5，高于这个阈值的是一类，低于这个阈值的又是一类，这个很简单。

混合高斯模型是什么意思呢？先来看一个图：



这个图是无法用单一高斯模型来进行拟合的。单高斯分布模型在二维空间应该近似于椭圆，在三维空间上近似于椭球。遗憾的是在很多分类问题中，属于同一类别的样本点并不满足“椭圆”分布的特性。这就引入了高斯混合模型。

高斯混合模型通俗地来讲就是用多个单一高斯模型来模拟一个图形。例如这个图像可以用三个单一高斯模型来拟合。如下：



而在拟合过程中每一个单一高斯模型就代表一个类。所以对于一个给定的点，我们可以计算他在每一个单一高斯模型的概率，然后比较这些概率。其中最大的单一高斯模型就是这个点的分类。

所以高斯混合模型最大的问题就是已知样本点，怎么来拟合成一个高斯混合模型。拟合的过程非常的数学，用的方法是EM方法来不断的尝试值。直到尝试到最好的值。EM算法通俗的解释如下：

如果要优化的目标函数有俩个参数，例如L(W,B) ，那么我们先固定W，调整B，使的L最小，然后在固定B1，调整W，是的L最小，如此反复迭代知道收敛。

最后就可以拟合成三个单一高斯分布。

其实仔细看起来GMM和K-Mean 非常的相似。因为他们都是实现聚类，只不过用的聚类算法不一样，那为什么要用GMM呢？看起来那么复杂。这需要先了解下统计方法学的

概率函数和非概率函数

例如y = f(x)

对于非概率函数主要我给定一个x就能找到一个确定的y

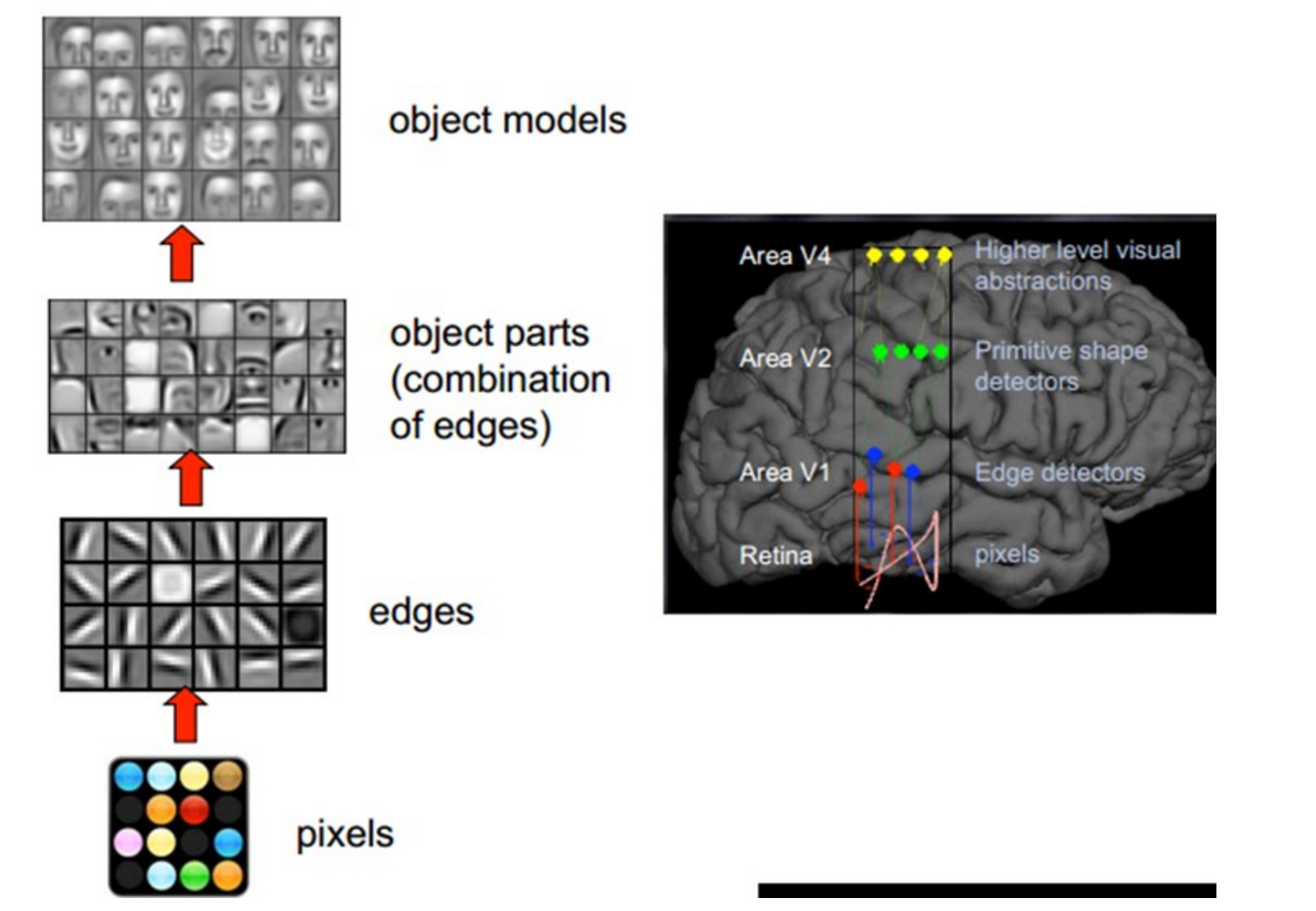
而对于概率函数给定一个x我只能告诉你等于y的概率

所以比较GMM和K-Mean。 GMM用的是概率函数，而K-mean用的是非概率函数。GMM会告诉用户这个点被分到了哪个类，并且还会得到分到每个类的概率。这个概率其实非常有用，举个例子：假如你在街上看到一条狗，既像朋友的狗又像隔壁老王家的狗。假如概率分别是51%和49%。如果此时你选择你家的狗，即使你得到了一个分类，那也是一个风险很大的分类。为了避免这种风险很大的分类。如果我们用GMM算法，那么我们可以吧这些风险很大的分类先提取出来，在利用其它的特征来判断这些点的分类。例如你可以给老王打个电话问问狗在不在他家，顺便问问老王在不在你家，这样岂不是一举两得。但是KNN就办不到。

正是由于GMM这个特征，他被广泛的应用于图像识别。语音识别和人脸识别等领域。

**深度学习**

首先深度学习是基于人工神经网络实现的。而这俩个算法的灵感都在于人大脑处理信息的方式。所以在了解这俩个算法之前，我们必须先了解一下人的大脑处理信息的方式：

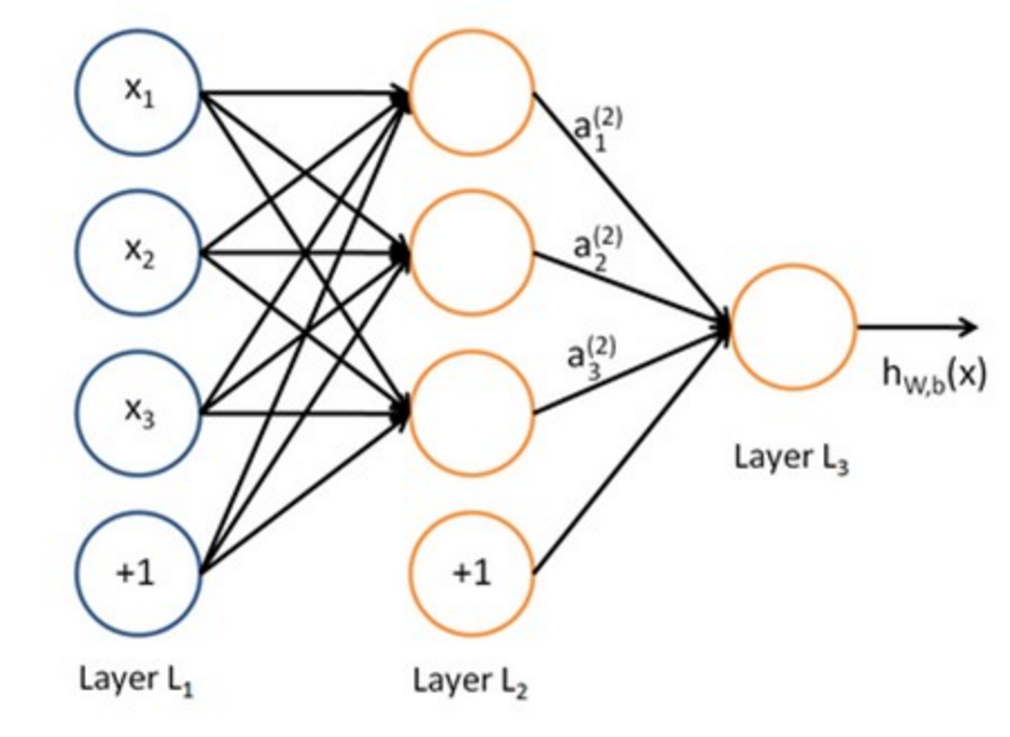


我们拿人脸来举例子，首先就是视网膜上会呈现出人脸的颜色，轮廓等各个特征。这些就像是图像输入到计算机的2进制代码。然后大脑会对这些特征进行进一步抽象，例如抽象出人脸的颜色，鼻子的轮廓，眼睛的轮廓等等。然后大脑会进一步根据颜色轮廓等信息在来判断这是人脸还是狗脸。至于大脑有多少神经细胞负责抽象，抽象的具体过程是什么？人类现在也没有搞清楚。但是大脑的分层处理模式是经过证实的：

当人类的感官系统受到外界的刺激后，会在相应的神经细胞上产生电信号，这些电信号逐步分层转化，最后到大脑皮层形成一个抽象的概念。例如图上：

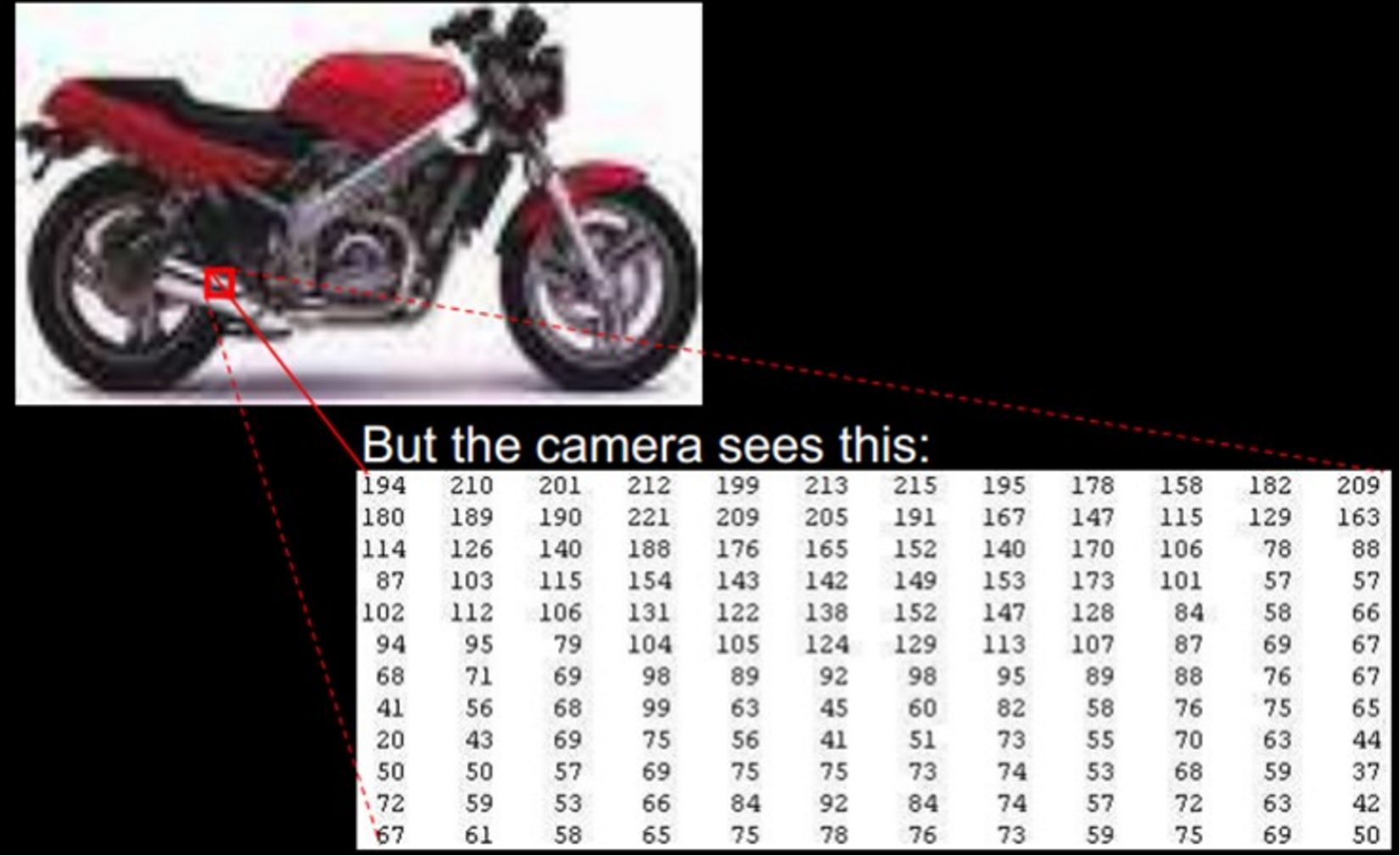
当我们受到视觉信号后，这些视觉信号会激活蓝色的神经细胞。蓝色的神经细胞处理信号后又会绿色的神经细胞。。。。最后传到黄色的神经细胞（大脑皮层）后，形成对信号的抽象。

有了这个启发，人类就开发了人工神经网络。但是这个比较弱鸡。因为想比大脑的分层处理模型，他只有三层如下：

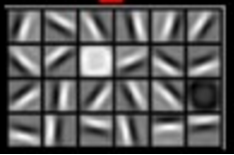


（1）

LayerL1是输入层，LayerL2是隐层，LayerL3是输出层。这里LayerL2有点难以让人理解。注意这个隐层不是对信息进行分类的。而是对信息进行组合。以一种其他方式对信息进行重组。我们先不管这个组合的方式。先看看他为什么要进行信息的重新组合。例子如下：



例如我们给一个摩托车拍照，拍照后我们会得这样一个2进制表。如果把这个2进制表的每一个像素点呈现到图像上可能是这样的：



从这个东西猜出来摩托车是不可能的。但是如果你通过某种方式在隐层里面将上面的2进制点进行重组，那么你可能就会得到如下结果：

那么这时你可能就会猜出来他是一个摩托车了。那这就是隐层的作用。即我们可能面对一堆输入的数据进行分类时，会发现特别困难。当我们通过隐层对数据进行重新组合后，再把这些重新组合的数据输入到分类器里发现此时就非常容易分类。这其实也正是符合人类的一个事物的认知。例如，如果你拿到一个拼图，如果拼图都拆分成了一个个的小块，那么你根据这些一个个小块来预测最后拼成图的图形是很困难的。但是如果你先拼好了几部分，然后拿着拼好的几大块再来预测最后的图形是不是就相对简单了点。

观察图（1），我们会发现隐层有a1，a2，a3 。根据上面的解释你可能就会明白a1 ，a2，a3分别代表什么意思了？a1 即我们按照a1的方式对数据进行重组，a2 ， a3 同理。那这种方式又是一种什么方式了？举个简单的例子，假如我们这个神经网络用来识别猫和狗。 a1负责对身体进行重组，a2负责对脸进行重组，a3对腿进行重组，拿到这三个数据后我们在根据猫和狗的身体，脸和腿的特征进行分类。所以a1，a2和a3像极了大脑里面的神经元，例如有的神经元处理颜色，有的神经元处理轮廓灯，他们各司其职，然后将信息输入到大脑皮层进行抽象。

那现在回到刚才的问题，那这些神经元（a1,a2,a3）是通过一种什么样的方式来对信息进行重组变换的？回到刚才a1神经元，假如这个神经元负责识别脸，我们拿到一个猫的脸的label，然后我们利用logistics回归来拟合猫的脸。这样我们就可以的到logistics的参数。最后我们把隐层得到的数据发送给分类器进行分类即可，这里分类器可以是SVN，可以是boosting等。那既然隐层是通过logistics回归的到的，那么他总有误差，那么如何让误差到最小呢？采用的是back propagation的方式进行，简单来讲就是采用迭代的算法来训练整个网络，随机设定初值，计算当前网络的输出，然后根据当前输出和label之间的差去改变前面各层的参数，直到收敛（整体是一个梯度下降法）

那现在我们就明白了人工神经网络。但是人工神经网络是优缺点的：

第一：比较容易过拟合，还是刚才的例子一个图像可能本来没有身子，但是a2非要把几个简单的图像拟合成一个身子，那这就是过拟合

第二：他的误差校对太繁琐，如果有多层，你根本不知道参数有因为哪个隐层出现错误而导致结果和label有偏差。所以人工神经网络的隐层越少越准确

第三：人工神经网络只能实现分类功能，不能实现聚类的功能，这个和人脑是背道而驰的。

基于这些缺点，强大的深度学习出现了。要理解深度学习和人工神经网络的区别必须先看个故事：

1995 年前后，Bruno Olshausen和 David Field 两位学者任职 Cornell University，他们试图同时用生理学和计算机的手段，双管齐下，研究视觉问题。

他们收集了很多黑白风景照片，从这些照片中，提取出400个小碎片，每个照片碎片的尺寸均为 16x16 像素，不妨把这400个碎片标记为 S[i], i = 0,.. 399。接下来，再从这些黑白风景照片中，随机提取另一个碎片，尺寸也是 16x16 像素，不妨把这个碎片标记为 T。

他们提出的问题是，如何从这400个碎片中，选取一组碎片，S[k], 通过叠加的办法，合成出一个新的碎片，而这个新的碎片，应当与随机选择的目标碎片 T，尽可能相似，同时，S[k] 的数量尽可能少。用数学的语言来描述，就是：

Sum\_k (a[k] \* S[k]) --> T, 其中 a[k] 是在叠加碎片 S[k] 时的权重系数。

为解决这个问题，Bruno Olshausen和 David Field 发明了一个算法，稀疏编码（Sparse Coding）。

稀疏编码是一个重复迭代的过程，每次迭代分两步：

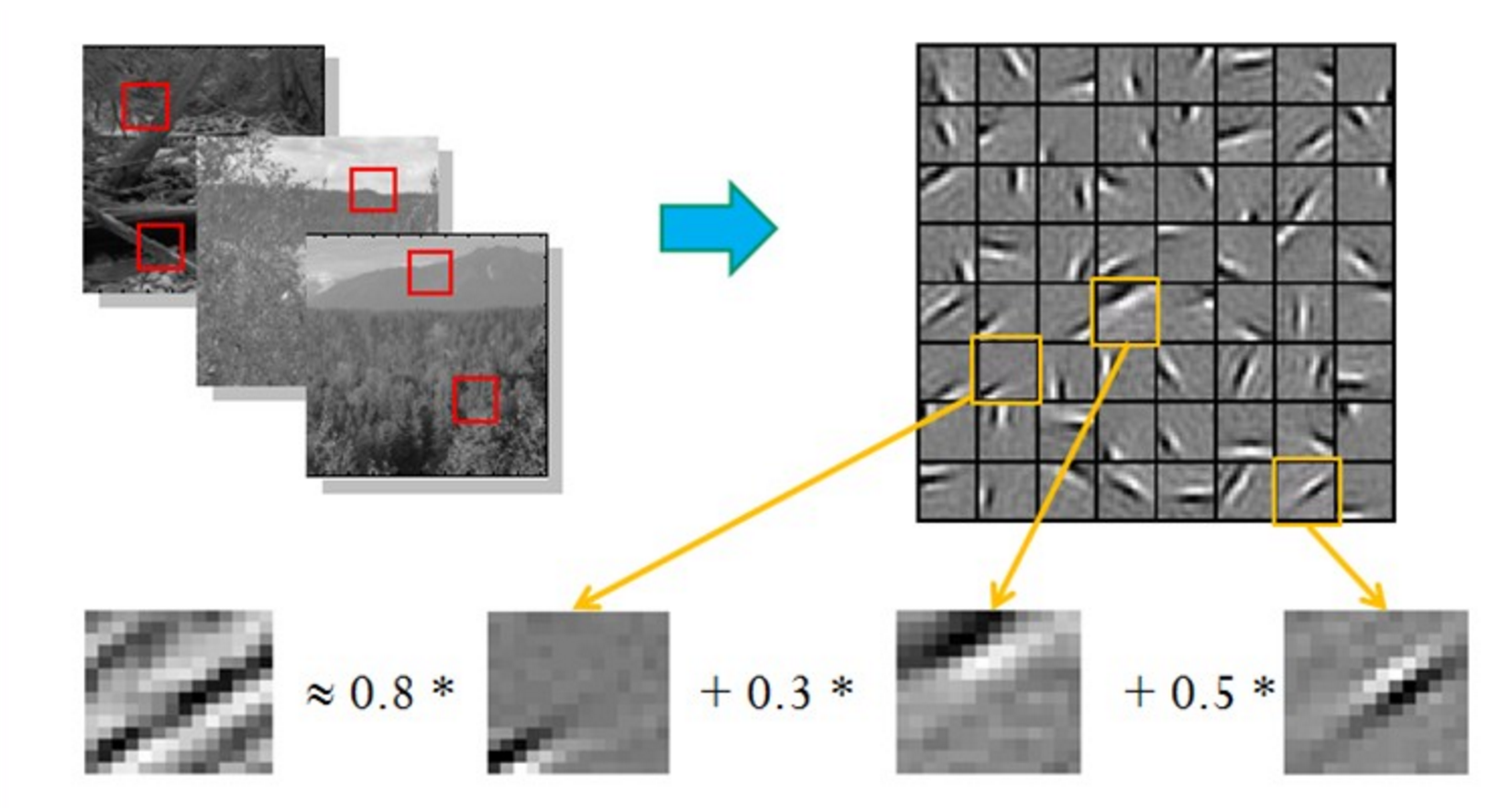
1）选择一组 S[k]，然后调整 a[k]，使得Sum\_k (a[k] \* S[k]) 最接近 T。

2）固定住 a[k]，在 400 个碎片中，选择其它更合适的碎片S’[k]，替代原先的 S[k]，使得Sum\_k (a[k] \* S’[k]) 最接近 T。

经过几次迭代后，最佳的 S[k] 组合，被遴选出来了。令人惊奇的是，被选中的 S[k]，基本上都是照片上不同物体的边缘线，这些线段形状相似，区别在于方向。

Bruno Olshausen和 David Field 的算法结果，与 David Hubel 和Torsten Wiesel 的生理发现，不谋而合！

也就是说，复杂图形，往往由一些基本结构组成。比如下图：一个图可以通过用64种正交的edges（可以理解成正交的基本结构）来线性表示。比如样例的x可以用1-64个edges中的三个按照0.8,0.3,0.5的权重调和而成。而其他基本edge没有贡献，因此均为0 。



另外，大牛们还发现，不仅图像存在这个规律，声音也存在。他们从未标注的声音中发现了20种基本的声音结构，其余的声音可以由这20种基本结构合成。

这个故事告诉我们，图像或者声音他们的复杂特征可以由仅仅几个简单特征正交得到。那这就很好了。因为基于这个特征我们可以进行无监督学习。

深度学习的思想是基于神经网络的，但是深度学习又解决了神经网络的刚才几个问题。深度学习的过程大致如下：

1）使用自下上升非监督学习（就是从底层开始，一层一层的往顶层训练）：

采用无标定数据（有标定数据也可）分层训练各层参数，这一步可以看作是一个无监督训练过程，是和传统神经网络区别最大的部分（这个过程可以看作是feature learning过程）：

具体的，先用无标定数据训练第一层，训练时先学习第一层的参数（这一层可以看作是得到一个使得输出和输入差别最小的三层神经网络的隐层），由于模型capacity的限制以及稀疏性约束，使得得到的模型能够学习到数据本身的结构，从而得到比输入更具有表示能力的特征；在学习得到第n-1层后，将n-1层的输出作为第n层的输入，训练第n层，由此分别得到各层的参数；

2）自顶向下的监督学习（就是通过带标签的数据去训练，误差自顶向下传输，对网络进行微调）：

基于第一步得到的各层参数进一步fine-tune整个多层模型的参数，这一步是一个有监督训练过程；第一步类似神经网络的随机初始化初值过程，由于DL的第一步不是随机初始化，而是通过学习输入数据的结构得到的，因而这个初值更接近全局最优，从而能够取得更好的效果；所以deep learning效果好很大程度上归功于第一步的feature learning过程。

当所有层训练完后，Hinton使用wake-sleep算法进行调优。

将除最顶层的其它层间的权重变为双向的，这样最顶层仍然是一个单层神经网络，而其它层则变为了图模型。向上的权重用于“认知”，向下的权重用于“生成”。然后使用Wake-Sleep算法调整所有的权重。让认知和生成达成一致，也就是保证生成的最顶层表示能够尽可能正确的复原底层的结点。比如顶层的一个结点表示人脸，那么所有人脸的图像应该激活这个结点，并且这个结果向下生成的图像应该能够表现为一个大概的人脸图像。Wake-Sleep算法分为醒（wake）和睡（sleep）两个部分。

1）wake阶段：认知过程，通过外界的特征和向上的权重（认知权重）产生每一层的抽象表示（结点状态），并且使用梯度下降修改层间的下行权重（生成权重）。也就是“如果现实跟我想象的不一样，改变我的权重使得我想象的东西就是这样的”。

2）sleep阶段：生成过程，通过顶层表示（醒时学得的概念）和向下权重，生成底层的状态，同时修改层间向上的权重。也就是“如果梦中的景象不是我脑中的相应概念，改变我的认知权重使得这种景象在我看来就是这个概念”。

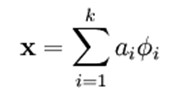
深度学习的概念介绍完了，我们再来看下稀疏编码。

**9.2、Sparse Coding稀疏编码**

       如果我们把输出必须和输入相等的限制放松，同时利用线性代数中基的概念，即O = a1\*Φ1 + a2\*Φ2+….+ an\*Φn， Φi是基，ai是系数，我们可以得到这样一个优化问题：

Min |I – O|，其中I表示输入，O表示输出。

       通过求解这个最优化式子，我们可以求得系数ai和基Φi，这些系数和基就是输入的另外一种近似表达。

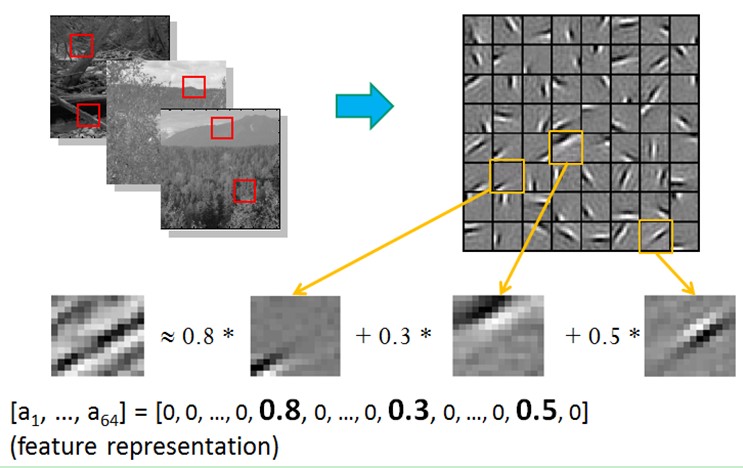


       因此，它们可以用来表达输入I，这个过程也是自动学习得到的。如果我们在上述式子上加上L1的Regularity限制，得到：

Min |I – O| + u\*(|a1| + |a2| + … + |an |)

        这种方法被称为Sparse Coding。通俗的说，就是将一个信号表示为一组基的线性组合，而且要求只需要较少的几个基就可以将信号表示出来。“稀疏性”定义为：只有很少的几个非零元素或只有很少的几个远大于零的元素。要求系数 ai 是稀疏的意思就是说：对于一组输入向量，我们只想有尽可能少的几个系数远大于零。选择使用具有稀疏性的分量来表示我们的输入数据是有原因的，因为绝大多数的感官数据，比如自然图像，可以被表示成少量基本元素的叠加，在图像中这些基本元素可以是面或者线。同时，比如与初级视觉皮层的类比过程也因此得到了提升（人脑有大量的神经元，但对于某些图像或者边缘只有很少的神经元兴奋，其他都处于抑制状态）。

         稀疏编码[**算法**](http://lib.csdn.net/base/datastructure)是一种无监督学习方法，它用来寻找一组“超完备”基向量来更高效地表示样本数据。虽然形如主成分分析技术（PCA）能使我们方便地找到一组“完备”基向量，但是这里我们想要做的是找到一组“超完备”基向量来表示输入向量（也就是说，基向量的个数比输入向量的维数要大）。超完备基的好处是它们能更有效地找出隐含在输入数据内部的结构与模式。然而，对于超完备基来说，系数ai不再由输入向量唯一确定。因此，在稀疏编码算法中，我们另加了一个评判标准“稀疏性”来解决因超完备而导致的退化（degeneracy）问题。（[详细过程请参考：UFLDL Tutorial稀疏编码](http://deeplearning.stanford.edu/wiki/index.php/%E7%A8%80%E7%96%8F%E7%BC%96%E7%A0%81)）



       比如在图像的Feature Extraction的最底层要做Edge Detector的生成，那么这里的工作就是从Natural Images中randomly选取一些小patch，通过这些patch生成能够描述他们的“基”，也就是右边的8\*8=64个basis组成的basis，然后给定一个test patch, 我们可以按照上面的式子通过basis的线性组合得到，而sparse matrix就是a，下图中的a中有64个维度，其中非零项只有3个，故称“sparse”。

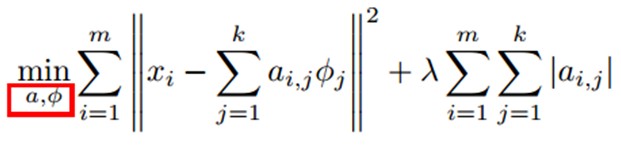
       这里可能大家会有疑问，为什么把底层作为Edge Detector呢？上层又是什么呢？这里做个简单解释大家就会明白，之所以是Edge Detector是因为不同方向的Edge就能够描述出整幅图像，所以不同方向的Edge自然就是图像的basis了……而上一层的basis组合的结果，上上层又是上一层的组合basis……（就是上面第四部分的时候咱们说的那样）

       Sparse coding分为两个部分：

**1）Training阶段：**给定一系列的样本图片[x1, x 2, …]，我们需要学习得到一组基[Φ1, Φ2, …]，也就是字典。

       稀疏编码是k-means算法的变体，其训练过程也差不多（EM算法的思想：如果要优化的目标函数包含两个变量，如L(W, B)，那么我们可以先固定W，调整B使得L最小，然后再固定B，调整W使L最小，这样迭代交替，不断将L推向最小值。EM算法可以见我的博客：“[从最大似然到EM算法浅解](http://blog.csdn.net/zouxy09/article/details/8537620)”）。

       训练过程就是一个重复迭代的过程，按上面所说，我们交替的更改a和Φ使得下面这个目标函数最小。



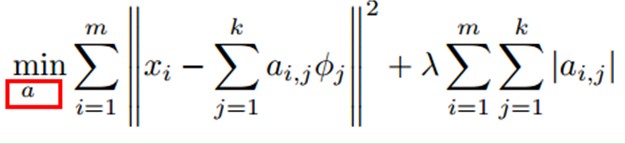
      每次迭代分两步：

a）固定字典Φ[k]，然后调整a[k]，使得上式，即目标函数最小（即解LASSO问题）。

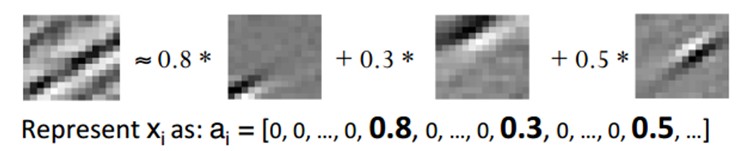
b）然后固定住a [k]，调整Φ [k]，使得上式，即目标函数最小（即解凸QP问题）。

      不断迭代，直至收敛。这样就可以得到一组可以良好表示这一系列x的基，也就是字典。

**2）Coding阶段：**给定一个新的图片x，由上面得到的字典，通过解一个LASSO问题得到稀疏向量**a**。这个稀疏向量就是这个输入向量x的一个稀疏表达了。



例如：



想了解更多关于深度学习的信息，请看我github TensorFlow目录里面的项目

机器学习领域常见的几种降维算法

主成分分析算法(PCA) :

Principal Component Analysis(PCA)是最常用的线性降维方法，它的目标是通过某种线性投影，将高维的数据映射到低维的空间中表示，并期望在所投影的维度上数据的方差最大，以此使用较少的数据维度，同时保留住较多的原数据点的特性

PCA追求的是在降维之后能够最大化保持数据的内在信息，并通过衡量在投影方向上的数据方差的大小来衡量该方向的重要性。但是这样投影以后对数据的区分作用并不大，反而可能使得数据点揉杂在一起无法区分。这也是PCA存在的最大一个问题，这导致使用PCA在很多情况下的分类效果并不好

Linear Discriminant Analysis (也有叫做Fisher Linear Discriminant)是一种有监督的（supervised）线性降维算法。与PCA保持数据信息不同，LDA是为了使得降维后的数据点尽可能地容易被区分！

1、同类的数据点尽可能的接近（within class）

2、不同类的数据点尽可能的分开（between class）

Locally linear embedding（LLE）[1] 是一种非线性降维算法，它能够使降维后的数据较好地保持原有流形结构。LLE可以说是流形学习方法最经典的工作之一。很多后续的流形学习、降维方法都与LLE有密切联系。

Laplacian Eigenmaps[1] 看问题的角度和LLE有些相似，也是用局部的角度去构建数据之间的关系。

它的直观思想是希望相互间有关系的点（在图中相连的点）在降维后的空间中尽可能的靠近。Laplacian Eigenmaps可以反映出数据内在的流形结构。