我突然有一种很神奇的想法：  
 某一天晚上，我开始思考，为什么我知道老师写在黑板上的那一笔就是数字1，然后回想起幼儿园还不知道小学的时候，当时心里虽然没有办法理解为什么，但是直接告诉自己“以后这就是数字1”。仔细想了一下，发现当时已经对很多事物都有了识别的能力，比如至少已经知道纸是纸，笔是笔，所以才能按照老师的要求不断在纸上面书写数字1 。接着，我就开始思考婴儿是如何知道认知一个东西，幻想一个婴儿在地上爬行的过程中，遇到了一个脸盆，之后他拿起了脸盆，摔击、敲打等，后来就明白脸盆是一个整体，也就是所谓的“一个东西”，然后拿着脸盆环绕着观察就能记住脸盆，之后再知道这个名字与具体发音，就能像成人一样认知这个脸盆。对于具体事物的认知，还要考虑事物之间的组合，比如一个黑点，在脸上它可能叫做雀斑，在纸上它可能叫做一滴墨。对于小时候吃到的豆腐乳，我是很迟之后才知道这个东西的名字，对于豆腐乳的认知，我们需要视觉（视觉上的一个整体），辅以味觉、触觉、嗅觉等，不过除了视觉之外都是辅助信息。在吃的过程中，将豆腐乳戳破，然后就会发现，原来还可以分为豆腐乳皮等等。之后在遇到其他东西，结合举一反三的能力，就会明白什么是皮，继而甚至能够自己创造一个带有皮的词汇。

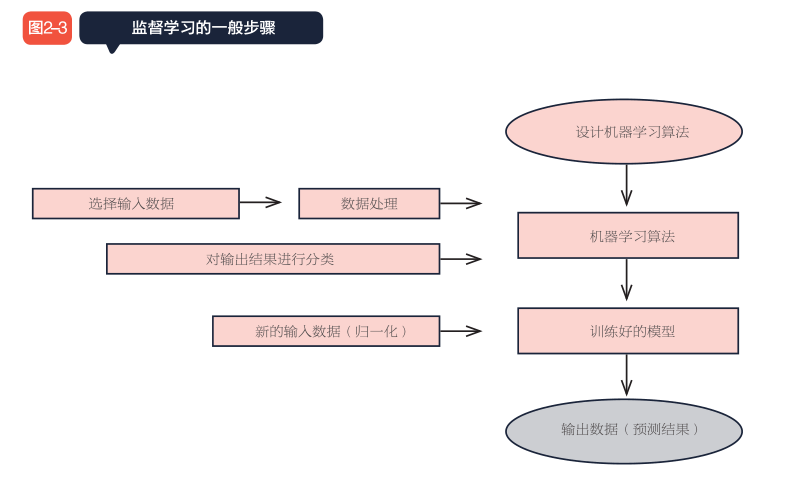
所以机器学习的关键在于如何让机器能够明白“一个东西”，它需要知道怎么样才能算是一个“整体”。但是这是对具体事物的认知，认知的过程中要结合对事物的组合，就像上面举例的黑点一样。对于中文中“皮”的概念，需要抽象上的举一反三能力（认知大量同种特征事物之后对相同的特征的一种概括能力）。

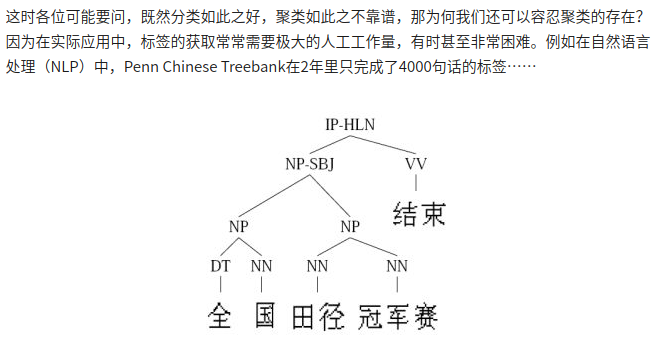
虽然我不清楚现在机器学习的算法具体如何高大上，但是当当从数据的获取上，我就明白了，婴儿可以拿着盆环顾来认知，但是如果机器仅仅通过二维图像来认知最后实现结果绝对不具有对事物的普遍认知能力，如果能够建立所有事物的3D一般模型，或许能有不错的效果。人脸识别已经开始3D建模。

想起来这个观点，我在大一的“新生研讨课”上就提到过，作为最后的课程报告交给魏晓勇老师（好像是他吧），但是我的分数和同学们相比却低了一些，因为这种情况让我并没有将自己的这个想法真正放在心上，直到今天开始又一次很认真地思考。

人处理问题是图形化的，机器处理问题是数学化的

机器学习又分为监督学习和非监督学习，机器学习的本质就是寻找数据特征与自定义标签之间的关系。监督学习，是指给出的训练数据已经带有了标签。非监督学习，是指训练数据不带有标签，通过大量的数据让机器自己总结经验判断对错的学习，需要聚类。半监督学习，是指少量训练数据带有标签，大部分数据不应该带有标签。隐藏在半监督学习下的基本规律在于：数据的分布必然不是完全随机的，通过一些有标签数据的局部特征，以及更多没标签数据的整体分布，就可以得到可以接受甚至是非常好的分类结果。





目前，市场应用较多的是监督学习，因为目前机器自己判断是十分不靠谱的，但是创建自定义标签是件复杂的事情。机器学习较多的应用于分类、回归、聚类等问题。最后收敛程度代表最后的效果

生物可识别特征： 指纹、脸、体型与走姿、声纹、视网膜、掌型、虹膜、静脉(通过静脉识别仪取得个人手掌静脉分布图)

**信息：**Shanon认为信息是用来消除随机不确定性的东西，信息可以定义如下：

182127101856815

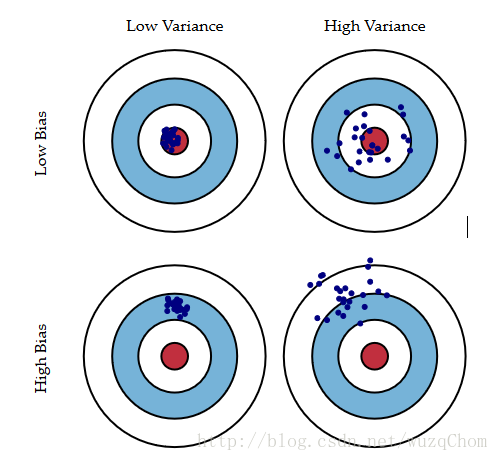
　　I(x)用来表示随机变量的信息，p(xi)指是当xi发生时的概率。

　　熵是用来度量不确定性的，当熵越大，X=xi的不确定性越大，反之越小。对于机器学习中的分类问题而言，熵越大即这个类别的不确定性更大，反之越小。

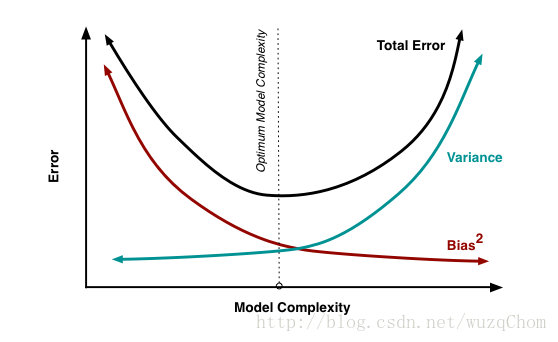
**没有免费的午餐定理 NFL：**（No Free Lunch Theorem）的意思应该从经济学的角度来理解，即不管做什么事情一定会存在成本。在机器学习中，经常用于指出分类算法的优异程度需要指定一个特定的问题来进行比较，比如对于人的面部识别相对于对于人的手的识别，所需要选择的关键特征一定不同。但是这并不是对“存在一种算法能够对所有事物进行学习”的反对，因为就好比人能够认知很多事物并且认知的准确度很高，但是这种准确度并不是100%，记得识别模糊图像的时候，人的识别率比机器的识别率低。

**线性回归**的英文是Regression toward the mean，直译就是"回归到均值"。各种数据都有其平均值，比如人类的身高。高的人大概率孩子也高，矮的人大概率孩子也矮，但是高的人身高每增加一个单位，其孩子可能只能增加半个单位；矮的人身高每减少一个单位，其孩子可能只会减少半个单位。也就是子代的平均高度会向着一个中心趋势靠拢，这就是回归到均值。回归问题等价于函数拟合，线性回归也就是用线性的函数去拟合所有观测数据，利用最小二乘法使得误差（预测值和真实值差值的平方和）最小化。对于新的观测数据，使用该函数计算得到一个预测值。

任何依赖于数据的东西，分类、聚类或者深度学习的效果，基本还要依赖于数据的好的程度，也就是如何标注数据或者如何筛选数据进行训练。在机器学习上，存在偏差与方差这两个概念。偏差认为是单个模型的学习能力，而方差则描述的是同一个学习算法在不同数据集的不稳定性，具体可以结合下面的例子来实现：



偏差描述的是算法的预测的平均值和真实值的关系，图中的红色位置就是真实值所在位置，蓝色的点是算法每次预测的值。 偏差越高则离红色部分越远，而方差越大则算法每次的预测之间的波动会比较大。

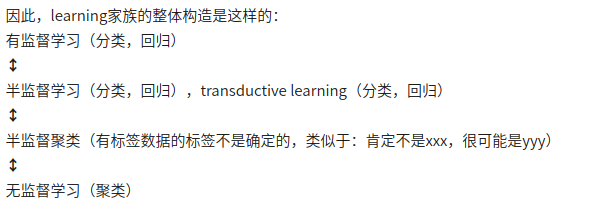
偏差方差窘境：  


偏差与方差两者是冲突的，假如学习算法训练不足时，此时学习器的拟合能力不够强，此时数据的扰动不会对结果产生很大的影响（可以想象成由于训练的程度不够，此时学习器指学习到了一些所有的数据都有的一些特征），这个时候偏差主导了算法的泛化能力。随着训练的进行，学习器的拟合能力逐渐增强，变差逐渐减小，但此时不同通过数据学习得到的学习器就可能会有较大的偏差，即此时的方差会主导模型的泛化能力。若学习进一步进行，学习器就可能学到数据集所独有的特征，而这些特征对于其它的数据是不适用的，这个时候就是发生了过拟合的想象。

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）。

如果数据集很小，比如通过狼人杀视频（国内的视频都是顶尖局）或者论坛上面的记录（日本那边存在狼人杀的游戏论坛）来训练机器判断狼人杀中的角色，就需要借助一些理论来辅助判断，狼人杀本质是一个博弈论+心理学游戏，越是顶尖的局，心理学的影响就会越小。

**Transductive Learning（直推学习）：** 训练数据少量标记，并且假设未标记数据就是最终用来测试的数据。学习的目的就是为了在这些数据上获取最佳的适应。

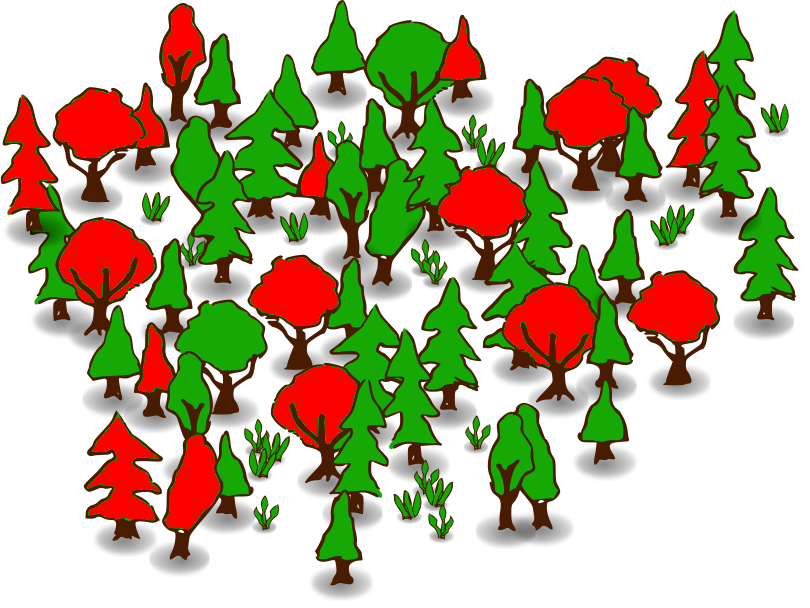


**集成学习：**集成学习通过建立几个模型组合的来解决单一预测问题。它的工作原理是生成多个分类器/模型，各自独立地学习和作出预测。这些预测最后结合成单预测，因此优于任何一个单分类的做出预测。

**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。

**决策树（Decision Tree, DT）**：无监督学习中的产物，按照训练数据提供的概率，得到表示状态变化的决策树。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。

**随机森林RF**：  
随机森林中有许多的分类树。我们要将一个输入样本进行分类，我们需要将输入样本输入到每棵树中进行分类。打个形象的比喻：森林中召开会议，讨论某个动物到底是老鼠还是松鼠，每棵树都要独立地发表自己对这个问题的看法，也就是每棵树都要投票。该动物到底是老鼠还是松鼠，要依据投票情况来确定，获得票数最多的类别就是森林的分类结果。森林中的每棵树都是独立的，99.9%不相关的树做出的预测结果涵盖所有的情况，这些预测结果将会彼此抵消。少数优秀的树的预测结果将会超脱于芸芸“噪音”，做出一个好的预测。将若干个弱分类器的分类结果进行投票选择，从而组成一个强分类器，这就是随机森林bagging的思想（关于bagging的一个有必要提及的问题：bagging的代价是不用单棵决策树来做预测，具体哪个变量起到重要作用变得未知，所以bagging改进了预测准确率但损失了解释性。）。



**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。

**神经网络 NN：**

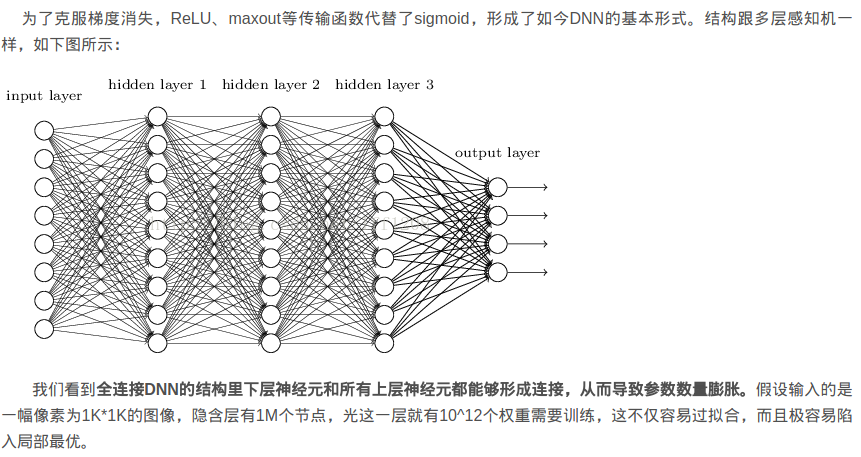
  神经网络技术起源于上世纪五、六十年代，当时叫感知机（perceptron），包含有输入层、输出层和一个隐藏层。输入的特征向量通过隐藏层变换到达输出层，由输出层得到分类结果。但早期的单层感知机存在一个严重的问题——它对稍微复杂一些的函数都无能为力（如异或操作）。直到上世纪八十年代才被Hition、Rumelhart等人发明的多层感知机克服，就是具有多层隐藏层的感知机。

      多层感知机可以摆脱早期离散传输函数的束缚，使用sigmoid或tanh等连续函数模拟神经元对激励的响应，在训练算法上则使用Werbos发明的反向传播BP算法。这就是现在所说的神经网络NN。

**神经网络的层数直接决定了它对现实的刻画能力**——利用每层更少的神经元拟合更加复杂的函数。但问题出现了——**随着神经网络层数的加深，优化函数越来越容易陷入局部最优解，并且这个“陷阱”越来越偏离真正的全局最优**。利用有限数据训练的深层网络，性能还不如较浅层网络。同时，另一个不可忽略的问题是随着网络层数增加，“**梯度消失**”现象更加严重。（具体来说，我们常常使用sigmoid作为神经元的输入输出函数。对于幅度为1的信号，在BP反向传播梯度时，每传递一层，梯度衰减为原来的0.25。层数一多，梯度指数衰减后低层基本上接受不到有效的训练信号。）

**深度学习的热潮。具体是利用预训练的方式缓解了局部最优解的问题**

DNN： 深度神经网络

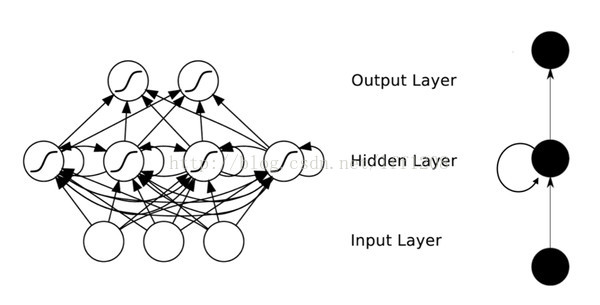


CNN： 卷积神经网络

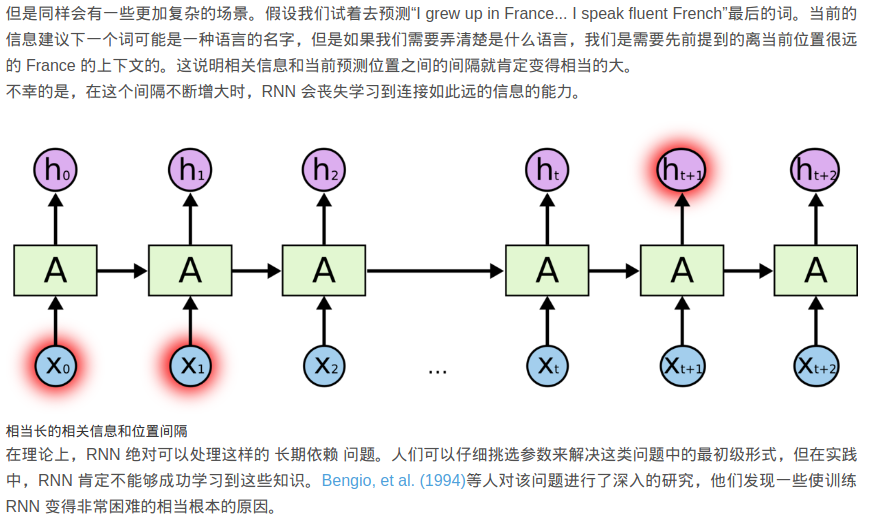
由于图像中存在固有的局部模式（如人脸中的眼睛、鼻子、嘴巴等），所以将图像处理和神将网络结合引出卷积神经网络CNN。CNN是通过卷积核将上下层进行链接，同一个卷积核在所有图像中是共享的，图像通过卷积操作后仍然保留原先的位置关系。

RNN： 循环神经网络

RNN用于处理时间序列上变化的内容，比如狼人杀游戏的进行。在普通的全连接网络或CNN中，每层神经元的信号只能向上一层传播，样本的处理在各个时刻独立，因此又被成为前向神经网络(Feed-forward Neural Networks)。而在RNN中，神经元的输出可以在下一个时间段直接作用到自身，即第i层神经元在m时刻的输入，除了(i-1)层神经元在该时刻的输出外，还包括其自身在(m-1)时刻的输出。



但是为了解决RNN的长期依赖问题，后面提出了LSTM、Attention等。



听说训练RNN和LSTM硬件成本有点高。

LSTM网络（长短期记忆RNN）：

增加了“忘记门”，之后升级出现了“重置门”、“更新门”。

RNN+Attention 注意力模型：

相关网址： <https://blog.csdn.net/tg229dvt5i93mxaq5a6u/article/details/78422216>

深度学习里的Attention model其实模拟的是人脑的注意力模型，举个例子来说，当我们观赏一幅画时，虽然我们可以看到整幅画的全貌，但是在我们深入仔细地观察时，其实眼睛聚焦的就只有很小的一块，这个时候人的大脑主要关注在这一小块图案上，也就是说这个时候人脑对整幅图的关注并不是均衡的，是有一定的权重区分的。这就是深度学习里的Attention Model的核心思想。

“回顾过去”的更好方式是使用注意力模型将过去编码向量汇总到语境矢量 CT中。

消耗的硬件资源传闻和LSTM相比少不少。

DRNN 深度循环神经网络

fastText：

fasttext是facebook 2016年开源的一个词向量与文本分类工具。是words2vec

衍生出来。fastText的学习速度比较快，效果还不错。fastText适用与分类类别非常大而且数据集足够多的情况，当分类类别比较小或者数据集比较少的话，很容易过拟合。

如果你用的底层数学库不一样，而这些底层数学库的速度不同。比如说Atlas和MKL，一般MKL完胜。就会导致执行速度的不同。

深度学习嘛离不开CUDA和cuDNN 两种加速

编程语言：C/C++ Python

浏览器中的开源学习库：deeplearnjs

TensorFlow、MXNet、Theano、Torch、Caffe和Microsoft CNTK

<https://deeplearning4j.org/cn/compare-dl4j-torch7-pylearn#caffe>

<https://chenrudan.github.io/blog/2015/11/18/comparethreeopenlib.html>

有限自动机加上栈很搭。

**PyTorch**

Facebook于2017年1月开放了Torch的Python API ― [PyTorch](https://github.com/pytorch/pytorch" \t "_blank)的源代码。PyTorch 支持动态计算图，让您能处理长度可变的输入和输出，而这在RNN应用和其他一些情形中很有帮助。CMU 的 DyNet 和 PFN 的 Chainer 框架也支持动态计算图。

**caffe**

网友倾向于linux中使用这个深度学习框架。

cuDNN全称CUDA Deep Neural Network library，是nvidia专门针对深度神经网络设计的一套GPU计算加速库，被广泛用于各种深度学习框架。

**DL4J**

Deeplearning4j基于JVM，与Hadoop和[Spark](https://deeplearning4j.org/cn/spark.html)集成，可使用任意数量的[GPU](./gpu)或[CPU](./native)运行