通俗来讲，举个例子吧。假如妳想训练出这样一个模型：根据人的身高和体重来预测这个人美还是丑。这是一个简单的二分类问题。

现在想象你面前有一个平面直角坐标系。横轴（x轴）代表人的身高，纵轴（y轴）代表人的体重。

现在咱们拍脑袋决定：咱们的模型应该是线性的（就是一条直线）。通俗来讲，在妳的平面直角坐标系里，妳画一条线，这条线就把美人和丑人分开了。这条线就是我们最终的模型。

现在咱们要开始往平面直角坐标系放数据了。

假设咱只有两组数据（就是说咱们的坐标系里只有两个点）。这两组数据随机组合，一共有3种情况。  
第一种情况：咱们有两个美人的数据  
第二种情况：咱们有两个丑人的数据  
第三种情况：咱们有一美一丑的数据  
无论是哪一种情况，咱们都可以通过一条线把美人和丑人分开。这说明：线性的模型是完全可以shatter两组数据的。

但假如说咱们有四组数据（坐标系里有四个点）。咱们就无法保证线性模型可以完全解释所有数据的可能性了。例如咱们的数据是（180cm, 50kg）=美，（10cm, 10kg）=美，（180cm, 10kg）= 丑还有(10cm, 50kg）=丑。对于这组数据，咱无论怎么画直线，都没有办法把美和丑分开在直线两侧。这说明：线性的模型是没有办法Shatter 4组数据的。

假如说咱们有三组数据，还是总可以通过画线的方式把美/丑分开的。（大家仔细想一想）。

所以，线性模型在这种二维数据的情况下的VC dimension 是3。（因为线性模型最多能Shatter3组数据）

现在，假如说我们突然改变主意了：咱们的模型可以是非线性的。那非线性模型的VC dimension可就高了。想象一下，一条曲线是不是理论上可以把坐标系里的所有美丑都分开？

所以通俗的理解： VC dimension就是某类模型对数据数量的包容性。VC dimension越高，就说明包容性越强。

说了这么多，VC dimension到底有什么用呢？简单来说，VC dimension可以帮助我们选择更好的模型。所谓“更好”的模型，可以理解为风险（risk)更低的模型。

如何估计模型的风险呢？咱们有这个公式：  
真正的风险 < 根据已有数据计算出的风险 +f(VC dimension)

f(VC dimension)是一个以VC dimension为变量的函数。咱们要选择的模型，一定要使f(VC dimension)低，这样真正的风险就会低。风险低的模型就更好。



**目录：**

* 说说历史
* Hoeffding不等式
* Connection to Learning
* 学习可行的两个核心条件
* Effective Number of Hypotheses
* Growth Function
* Break Point与Shatter
* VC Bound
* VC dimension
* 深度学习与VC维
* 小结
* 参考文献

VC维在机器学习领域是一个很基础的概念，它给诸多机器学习方法的可学习性提供了坚实的理论基础，但有时候，特别是对我们工程师而言，SVM，LR，深度学习等可能都已经用到线上了，但却不理解VC维。

这里，在台湾大学[机器学习基石](https://www.coursera.org/course/ntumlone)课程的基础上，我们简单聊聊“VC维的来龙去脉”。我们将解决以下问题：为什么某机器学习方法是可学习的？为什么会有过拟合？拿什么来衡量机器学习模型的复杂度？深度学习与VC维的关系？

**说说历史**

在讲VC维之前，我们不妨来说说VC维的历史。而说起VC维的历史，又不得不提起神经网络，一方面是因为神经网络与VC维的发明过程是交织在一起的，另一方面是由于神经网络乏善可陈的泛化控制方法，深度学习在理论基础上一直被怀疑，甚至神经网络和VC维的代表SVM还一起争风吃醋过好多年。

1943年，模拟神经网络由麦卡洛可（McCulloch）和皮茨（Pitts)提出，他们分析了理想化的人工神经元网络，并且指出了它们进行简单逻辑运算的机制。

1957年，康奈尔大学的实验心理学家弗兰克·罗森布拉特(Rosenblatt)在一台IBM–704计算机上模拟实现了一种他发明的叫作“感知机”（Perceptron）的神经网络模型。神经网络与支持向量机都源自于感知机（Perceptron）。

1962年，罗森布拉特著作：《神经动力学原理：感知机和大脑机制的理论》（Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms）。

1969年，明斯基和麻省理工学院的另一位教授佩普特合作著作：《感知机：计算几何学》（Perceptrons: An Introduction to Computational Geometry)。在书中，明斯基和佩普特证明单层神经网络不能解决XOR（异或）问题。

1971年，V. Vapnik and A. Chervonenkis在论文“On the uniform convergence of relative frequencies of events to their probabilities”中提出**VC维**的概念。

1974年，V. Vapnik提出了结构风险最小化原则。

1974年，沃波斯（Werbos）的博士论文证明了在神经网络多加一层，并且利用“后向传播”（Back-propagation）学习方法，可以解决XOR问题。那时正是神经网络研究的低谷，文章不合时宜。

1982年，在加州理工担任生物物理教授的霍普菲尔德，提出了一种新的神经网络，可以解决一大类模式识别问题，还可以给出一类组合优化问题的近似解。这种神经网络模型后被称为霍普菲尔德网络。

1986年，Rummelhart与McClelland发明了神经网络的学习算法Back Propagation。

1993年，Corinna Cortes和Vapnik等人提出了支持向量机(support vector machine)。神经网络是多层的非线性模型，支持向量机利用核技巧把非线性问题转换成线性问题。

1992~2005年，SVM与Neural network之争，但被互联网风潮掩盖住了。

2006年，Hinton提出神经网络的Deep Learning算法。Deep Learning假设神经网络是多层的，首先用Restricted Boltzmann Machine（非监督学习）学习网络的结构，然后再通过Back Propagation（监督学习）学习网络的权值。

现在，deep learning的应用越来越广泛，甚至已经有超越SVM的趋势。一方面以Hinton，Lecun为首的深度学习派坚信其有效实用性，另一方面Vapnik等统计机器学习理论专家又坚持着理论阵地，怀疑deep learning的泛化界。

**Hoeffding不等式**

Hoeffding不等式是关于一组随机变量均值的概率不等式。 如果X1,X2,⋯,Xn为一组独立同分布的参数为p的伯努利分布随机变量，n为随机变量的个数。定义这组随机变量的均值为：

[average_x_1](http://www.flickering.cn/wp-content/uploads/2015/03/average_x_1.png)

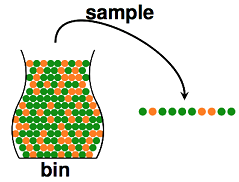
对于任意δ>0, Hoeffding不等式可以表示为

[hoeffding_11](http://www.flickering.cn/wp-content/uploads/2015/03/hoeffding_11.png)

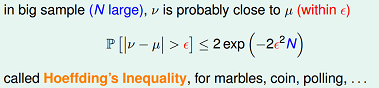
更多请参考:[Hoeffding不等式](http://science.scileaf.com/library/2461)，[集中不等式](http://zh.wikipedia.org/zh-cn/%E9%9B%86%E4%B8%AD%E4%B8%8D%E7%AD%89%E5%BC%8F)

**case示例**：

在统计推断中，我们可以利用样本的统计量(statistic)来推断总体的参数(parameter)，譬如使用样本均值来估计总体期望。如下图所示，我们从罐子里抽球，希望估计罐子里红球和绿球的比例。

[](http://www.flickering.cn/wp-content/uploads/2015/03/bin_sample.png)

直觉上，如果我们有更多的样本(抽出更多的球)，则样本期望ν应该越来越接近总体期望μ。事实上，这里可以用hoeffding不等式表示如下：

[](http://www.flickering.cn/wp-content/uploads/2015/03/bin_sample_hoeffding.png)

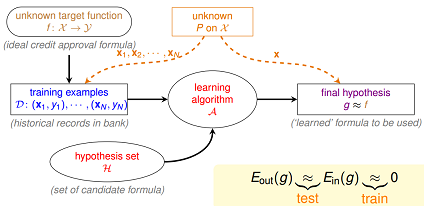
从hoeffding不等式可以看出，当n逐渐变大时，不等式的UpperBound越来越接近0，所以样本期望越来越接近总体期望。

**Connection to Learning**

接下来，我们希望可以将机器学习关联到上一节讨论的hoeffding不等式。

一个基本的机器学习过程如下图所示。其中的概念定义为： f 表示理想的方案(可以是一个函数，也可以是一个分布)，H 是该机器学习方法的假设空间，g 表示我们求解的用来预测的假设，g属于H。

机器学习的过程就是：通过算法A，在假设空间H中，根据样本集D，选择最好的假设作为g。选择标准是 g 近似于 f。

[](http://www.flickering.cn/wp-content/uploads/2015/03/setup_of_the_learning_problem_add_components.png)

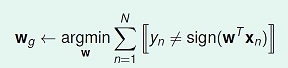
拿[perceptron](http://zh.wikipedia.org/zh/%E6%84%9F%E7%9F%A5%E5%99%A8)来举例。

感知机（perceptron）是一个线性分类器(linear classifiers）。 线性分类器的几何表示：直线、平面、超平面。

perceptron的假设空间，用公式描述，如下所示：

[perceptron_formula](http://www.flickering.cn/wp-content/uploads/2015/03/perceptron_formula.jpg)

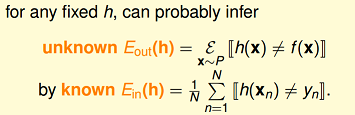
感知器的优化目标如下式所示，w\_g就是我们要求的最好的假设。

[](http://www.flickering.cn/wp-content/uploads/2015/03/perceptron_optim.jpg)

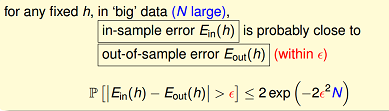
设定两个变量，如下图所示，图中 f(x)表示理想目标函数，h(x)是我们预估得到的某一个目标函数，h(x)是假设空间H中的一个假设。

**Eout(h)**，可以理解为在理想情况下(已知f)，总体(out-of-sample)的损失(这里是0–1 loss)的期望，称作expected loss。

**Ein(h)**，可以理解为在训练样本上(in-of-sample)，损失的期望，称作expirical loss。

[](http://www.flickering.cn/wp-content/uploads/2015/03/learning_hoeffding.png)

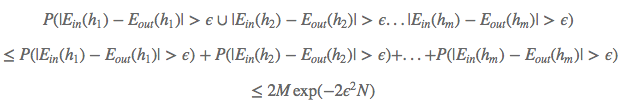
当训练样本量N足够大，且样本是独立同分布的，类比于上面“抽球”的例子，可以通过样本集上的expirical loss Ein(h) 推测总体的expected loss Eout(h)。基于hoeffding不等式，我们得到下面式子：

[](http://www.flickering.cn/wp-content/uploads/2015/03/learning_hoeffding2.png)

根据上面不等式，我们可以推断，当N足够大时，expected loss和expirical loss将非常接近。

注意在上面推导中，我们是针对某一个特定的解h(x)。在我们的假设空间H中，往往有很多个假设函数(甚至于无穷多个)，这里我们先假定H中有M个假设函数。

那么对于整个假设空间，也就是这M个假设函数，可以推导出下面不等式：

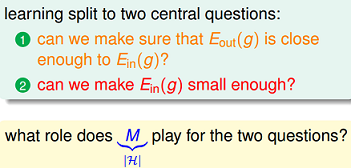
[](http://www.flickering.cn/wp-content/uploads/2015/03/hoeffding_12.png)

上面式子的含义是：在假设空间H中，设定一个较小的ϵ值，任意一个假设h，它的Ein(h)与Eout(h)的差由该值2Mexp(−2ϵ2N)所约束住。注意这个bound值与 “样本数N和假设数M” 密切相关。

**学习可行的两个核心条件**

在往下继续推导前，先看一下**什么情况下Learning是可行的**？

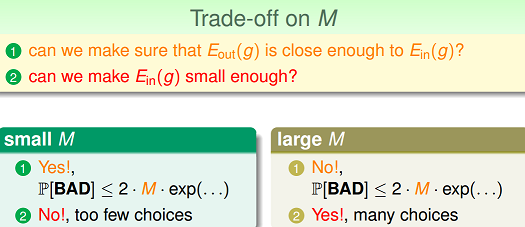
1. 如果假设空间H的size M是有限的，当N足够大时，那么对假设空间中任意一个g，Eout(g)约等于Ein(g)；
2. 利用算法A从假设空间H中，挑选出一个g，使得Ein(g)接近于0，那么[probably approximately correct](http://en.wikipedia.org/wiki/Probably_approximately_correct_learning)而言，Eout(g)也接近为0；

[](http://www.flickering.cn/wp-content/uploads/2015/03/two_central_questions.png)

上面这两个核心条件，也正好对应着test和train这两个过程。train过程希望损失期望(即Ein(g) )尽可能小；test过程希望在真实环境中的损失期望也尽可能小，即Ein(g)接近于Eout(g)。

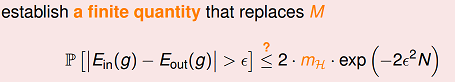
但往往我们更多在关心，如何基于模型的假设空间，利用最优化算法，找到Ein最小的解g。但容易忽视test这个过程，如果让学习可行，不仅仅是要在训练集表现好，在真实环境里也要表现好。

从上述推导出来的不等式，我们看到假设数M 在这两个核心条件中有着重要作用。

[](http://www.flickering.cn/wp-content/uploads/2015/03/trade_off_on_M.png)

M太小，当N足够大时，Ein和Eout比较接近，但如果候选假设集太小，不容易在其中找到一个g，使得Ein(g)约等于0，第二项不能满足。而如果M太大，这时候选集多了，相对容易在其中找到一个g，使得Ein(g)约等于0，但第一项就不能满足了。所以假设空间H的大小M很关键。

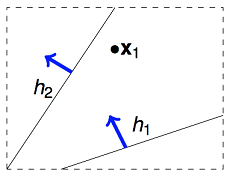
对于一个假设空间，M可能是无穷大的。要能够继续推导下去，那么有一个直观的思路，能否找到一个有限的因子m\_H来替代不等式bound中的M。

[](http://www.flickering.cn/wp-content/uploads/2015/03/finite_quantity.png)

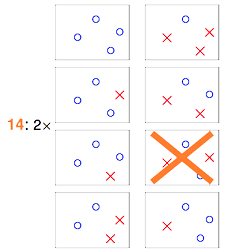
虽说假设空间很大，上述推导里，我们用到了P(h1 or h2 … hm) <= P(h1) + P(h2) + … + P(hm)。但事实上，多个h之间并不是完全独立的，他们是有很大的重叠的，也就是在M个假设中，可能有一些假设可以归为同一类。

下面我们以二维假设空间为例，来解释一下该空间下各假设在确定的训练样本上的重叠性。

举例来说，如果我们的算法要在平面上(二维空间)挑选一条直线方程作为g，用来划分一个点x1。假设空间H是所有的直线，它的size M是无限多的。但是实际上可以将这些直线分为两类，一类是把x1判断为正例的，另一类是把x1判断为负例的。如下图所示：

[](http://www.flickering.cn/wp-content/uploads/2015/03/1point2lines.png)

那如果在平面上有两个数据点x1,x2，这样的话，假设空间H中的无数条直线可以分为4类。那依次类推，3个数据点情况下，H中最多有8类直线。4个数据点，H中最多有14类直线(注意：为什么不是16类直线)。

[](http://www.flickering.cn/wp-content/uploads/2015/03/4points14lines.png)

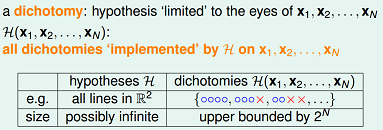
从上面在二维假设空间中的分析，我们可以推测到一个结论，假设空间size M是很大，但在样本集D上，有效的假设函数数目是有限的。接下来我们将继续推导这个有效的假设函数值。

**Effective Number of Hypotheses**

对于这个有效的假设函数值，我们尝试用一个数学定义来说明：

从H中任意选择一个方程h，让这个h对样本集合D进行二元分类，输出一个结果向量。例如在平面里用一条直线对2个点进行二元分类，输出可能为{1,–1}，{–1,1}，{1,1}，{–1,–1}，这样每个输出向量我们称为一个dichotomy。

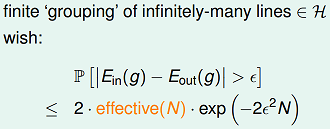
下面是hypotheses与dichotomies的概念对比：

[](http://www.flickering.cn/wp-content/uploads/2015/03/dichotomies.png)

注意到，如果对平面上的4个点来分类，根据前面分析，输出的结果向量只有14种可能，即有14个dichotomies。

如果有N个样本数据，那么有效的假设个数定义为： effective(N) = H作用于样本集D“最多”能产生多少不同的dichotomy。

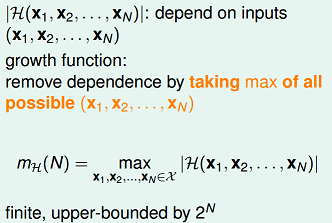
所以有一个直观思路，能否用effective(N)来替换hoeffding不等式中的M。接下来我们来分析下effective(N)。

[](http://www.flickering.cn/wp-content/uploads/2015/03/finite_effective_n.png)

**Growth Function**

H作用于D“最多”能产生多少种不同的dichotomies？这个数量与假设空间H有关，跟数据量N也有关。将H作用于D“最多”能产生的dichotomies数量(即effective(N) )表示为数学符号：max\_H(x1,x2,…,xN)

这个式子又称为“成长函数”(growth function)。在H确定的情况下，growth function是一个与N相关的函数。

[](http://www.flickering.cn/wp-content/uploads/2015/03/growth_function.png)

下图举4个例子，分别计算其growth function：

[](http://www.flickering.cn/wp-content/uploads/2015/03/growth_function_4case.png)

对于第一个例子，positive ray，相当于是正向的射线。该假设空间，作用于1个样本点，可以产生2种dichotomies：(–1)，(+1)。作用于2个样本点，可以产生3种dichotomies：(–1,+1)，(–1,–1)，(+1,+1)。作用于3个样本点，可以产生4种dichotomies。依次类推，可以推导出其成长函数 m\_H(N)=N+1；

求解出m\_H(N)后，那是不是可以考虑用m\_H(N)替换M? 如下所示：

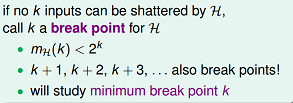
[growth_function_replace_m](http://www.flickering.cn/wp-content/uploads/2015/03/growth_function_replace_m.png)

**Break Point与Shatter**

在进一步推导前，再看两个概念：shatter，break point。

Shatter的概念：当假设空间H作用于N个input的样本集时，产生的dichotomies数量等于这N个点总的组合数2N是，就称：这N个inputs被H给shatter掉了。

要注意到 shatter 的原意是“打碎”，在此指“N个点的所有(碎片般的)可能情形都被H产生了”。所以mH(N)=2N的情形是即为“shatter”。

[](http://www.flickering.cn/wp-content/uploads/2015/03/break_point.png)

对于给定的成长函数m\_H(N)，从N=1出发，N慢慢变大，当增大到k时，出现mH(N)<2k的情形，则我们说k是该成长函数的**break point**。对于任何N > k个inputs而言，H都没有办法再shatter他们了。

举例来说，对于上面的positive ray的例子，因为m\_H(N)=N+1，当N=2时，m\_H(2)<22， 所以它的break point就是2。

**VC Bound**

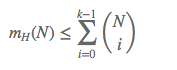
说完break point的概念后，再回到成长函数。

我们将成长函数的上界，设为B(N,k)，意为：maximum possible m\_H(N) when break point = k。

那么我们做一些简单的推导：

* B(2,2)=3。因为break point=2，任意两个点都不能被shatter，m\_H(2)肯定小于22，所以B(2,2)=3。
* B(3,2)=4。因为任意两个点都不能被shatter，那么3个点产生的dichotomies不能超过4，所以B(3,2)=4。
* B(N,1)=1。
* B(N,k)=2N for N < k；B(N,k)=2N–1 for N=k；
* B(4,3)=？去掉其中的一个数据点x4后，考虑到break point=3，余下数据(x1,x2,x3)的dichotomies数目不能超过B(3,3)。当扩展为(x1,x2,x3,x4)时，(x1,x2,x3)上的dichotomies只有部分被重复复制了，设被复制的dichotomies数量为a，未被复制的数量为b。于是有B(3,3) = a+b; B(4,3) = 2*a + b。因为a被复制了，表示x4有两个取值，那么(x1,x2,x3)上的a应该小于等于B(3,2)。所以推导出B(4,3) = 2*a + b <= B(3,3) + B(3,2)。
* 对于任意N>k，类推可以得到，B(N,k) ≤ B(N−1,k)+B(N−1,k−1)

最后利用数学归纳法，可以证明得到下面的bounding function(N>k)：

[](http://www.flickering.cn/wp-content/uploads/2015/03/m_h_n_1.png)

这个式子显然是多项式的，多项式的最高幂次项为：N^(k–1)。

所以我们得到结论：如果break point存在（有限的正整数），生长函数m(N) 是多项式的。

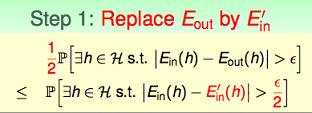
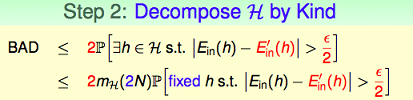
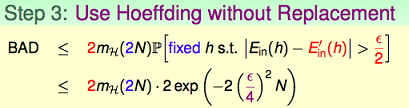
再重复一遍，H作用于数据量为N的样本集D，方程的数量看上去是无穷的，但真正有效(effective)的方程的数量却是有限的，这个数量为m\_H(N)。H中每一个h作用于D都能算出一个Ein来，一共有m\_H(N)个不同的Ein。

OK，到目前为止，关于m\_H(N)的推导结束。回到growth function小节提出的问题，能否用**m\_H(N)直接替换M?**

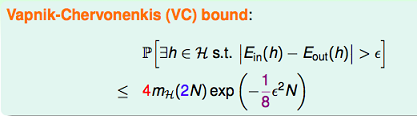
既然得到了m(N)的多项式上界，我们希望对之前的不等式中M 进行替换，用m\_H(N)来替换M。这样替换后，当break point存在时，N足够大时，该上界是有限的。

[replace_vc_bound](http://www.flickering.cn/wp-content/uploads/2015/03/replace_vc_bound.png)

然而直接替换是存在问题的，主要问题是：Ein的可能取值是有限个的，但Eout的可能取值是无限的。可以通过将Eout 替换为验证集(verification set) 的Ein’ 来解决这个问题。 下面是推导过程：

[](http://www.flickering.cn/wp-content/uploads/2015/03/vc_bound_step1.png)[](http://www.flickering.cn/wp-content/uploads/2015/03/vc_bound_step2.png)[](http://www.flickering.cn/wp-content/uploads/2015/03/vc_bound_step3.png)

最后我们得到下面的VC bound:

[](http://www.flickering.cn/wp-content/uploads/2015/03/vc_bound1.png)

关于这个公式的数学推导，我们可以暂且不去深究。我们先看一下这个式子的意义，如果假设空间存在有限的break point，那么m\_H(2N)会被最高幂次为k–1的多项式上界给约束住。随着N的逐渐增大，指数式的下降会比多项式的增长更快，所以此时VC Bound是有限的。更深的意义在于，N足够大时，对H中的任意一个假设h，Ein(h)都将接近于Eout(h)，这表示学习可行的第一个条件是有可能成立的。

**VC dimension**

说了这么多，VC维终于露出庐山真面目了。此概念由Vladimir Vapnik与Alexey Chervonenkis提出。

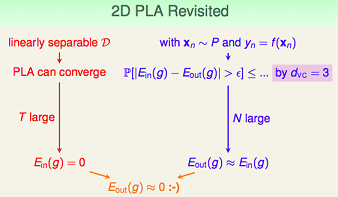
一个假设空间H的**VC dimension**，是这个H最多能够shatter掉的点的数量，记为dvc(H)。如果不管多少个点H都能shatter它们，则dvc(H)=无穷大。还可以理解为：vc-dim就是argmax\_n {growth function=power(2,n)}。

根据定义，可以得到一个明显的结论：

k = d\_vc(H) + 1

根据前面的推导，我们知道VC维的大小：与学习算法A无关，与输入变量X的分布也无关，与我们求解的目标函数f 无关。它只与模型和假设空间有关。

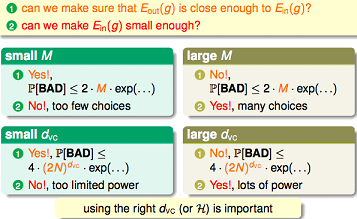
我们已经分析了，对于2维的perceptron，它不能shatter 4个样本点，所以它的VC维是3。此时，我们可以分析下2维的perceptron，如果样本集是线性可分的，perceptron learning algorithm可以在假设空间里找到一条直线，使Ein(g)=0；另外由于其VC维=3，当N足够大的时候，可以推断出：Eout(g)约等于Ein(g)。这样学习可行的两个条件都满足了，也就证明了2维感知器是可学习的。

[](http://www.flickering.cn/wp-content/uploads/2015/03/pla_revised.png)

总结回顾一下，要想让机器学到东西，并且学得好，有2个条件：

* H的d\_vc是有限的，这样VC bound才存在。(good H)；N足够大(对于特定的d\_vc而言)，这样才能保证vc bound不等式的bound不会太大。(good D)
* 算法A有办法在H中顺利的挑选一个使得Ein最小的g。(good A)

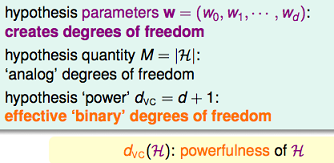
回到最开始提出的学习可行的两个核心条件，尝试用VC维来解释：

[](http://www.flickering.cn/wp-content/uploads/2015/03/m_and_d_vc.png)

从上图可以看出，当VC维很小时，条件1容易满足，但因为假设空间较小，可能不容易找到合适的g 使得Ein(g)约等于0。当VC维很大时，条件2容易满足，但条件1不容易满足，因为VC bound很大。

VC维反映了假设空间H 的强大程度(powerfulness)，VC 维越大，H也越强，因为它可以打散(shatter)更多的点。

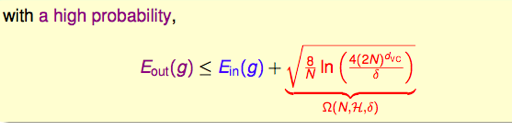
定义模型自由度是，模型当中可以自由变动的参数的个数，即我们的机器需要通过学习来决定模型参数的个数。

[](http://www.flickering.cn/wp-content/uploads/2015/03/degree_of_freedom.png)

一个实践规律：VC 维与假设参数w 的自由变量数目大约相等。dVC = #free parameters。

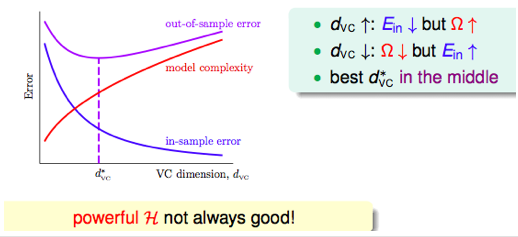
[vc_practical_rule](http://www.flickering.cn/wp-content/uploads/2015/03/vc_practical_rule.png)

我们将原不等式做一个改写，如下图所示：

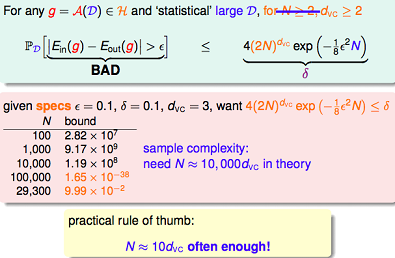
[](http://www.flickering.cn/wp-content/uploads/2015/03/vc_power1.png)

上面式子中的第3项表示模型复杂度。模型越复杂，VC维大，Eout 可能距离Ein 越远。如下图所示，随着d\_vc的上升，E\_in不断降低，而模型复杂度不断上升。

它们的上升与下降的速度在每个阶段都是不同的，因此我们能够寻找一个二者兼顾的，比较合适的d\_vc，用来决定应该使用多复杂的模型。

[](http://www.flickering.cn/wp-content/uploads/2015/03/vc_power2.png)

模型较复杂时(d\_vc 较大)，需要更多的训练数据。 理论上，数据规模N 约等于 10000\*d\_vc（称为采样复杂性，sample complexity）；然而，实际经验是，只需要 N = 10\*d\_vc。 造成理论值与实际值之差如此之大的最大原因是，VC Bound 过于宽松了，我们得到的是一个比实际大得多的上界。

[](http://www.flickering.cn/wp-content/uploads/2015/03/n_practical_rule.png)

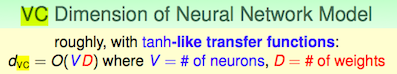
注意在前述讨论中，理想的目标函数为f(x)，error measure用的是“0–1 loss”。如果在unknown target上引入噪声(+noise)，或者用不同的error measure方法，VC theory还有效吗？这里只给出结论，VC theory对于绝大部分假设空间(or 加入噪声)和error度量方法，都是有效的。

除此外，我们为了避免overfit，一般都会加正则项。那加了正则项后，新的假设空间会得到一些限制，此时新假设空间的VC维将变小，也就是同样训练数据条件下，Ein更有可能等于Eout，所以泛化能力更强。这里从VC维的角度解释了正则项的作用。

**深度学习与VC维**

对于神经网络，其VC维的公式为：

dVC = O(VD)，其中V表示神经网络中神经元的个数，D表示weight的个数，也就是神经元之间连接的数目。(注意：此式是一个较粗略的估计，深度神经网络目前没有明确的vc bound)

[](http://www.flickering.cn/wp-content/uploads/2015/03/neural_network_vc_dimension.png)

举例来说，一个普通的三层全连接神经网络：input layer是1000维，hidden layer有1000个nodes，output layer为1个node，则它的VC维大约为O(1000\*1000\*1000)。

可以看到，神经网络的VC维相对较高，因而它的表达能力非常强，可以用来处理任何复杂的分类问题。根据上一节的结论，要充分训练该神经网络，所需样本量为10倍的VC维。如此大的训练数据量，是不可能达到的。所以在20世纪，复杂神经网络模型在out of sample的表现不是很好，容易overfit。

但现在为什么深度学习的表现越来越好。原因是多方面的，主要体现在：

* 通过修改神经网络模型的结构，以及提出新的regularization方法，使得神经网络模型的VC维相对减小了。例如卷积神经网络，通过修改模型结构(局部感受野和权值共享)，减少了参数个数，降低了VC维。2012年的AlexNet，8层网络，参数个数只有60M；而2014年的[GoogLeNet](http://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf)，22层网络，参数个数只有7M。再例如dropout，drop connect，denosing等regularization方法的提出，也一定程度上增加了神经网络的泛化能力。
* 训练数据变多了。随着互联网的越来越普及，相比于以前，训练数据的获取容易程度以及量和质都大大提升了。训练数据越多，Ein越容易接近于Eout。而且目前训练神经网络，还会用到很多data augmentation方法，例如在图像上，剪裁，平移，旋转，调亮度，调饱和度，调对比度等都使用上了。
* 除此外，pre-training方法的提出，GPU的利用，都促进了深度学习。

但即便这样，深度学习的VC维和VC Bound依旧很大，其泛化控制方法依然没有强理论支撑。但是实践又一次次证明，深度学习是好用的。所以VC维对深度学习的指导意义，目前不好表述，有一种思想建议，深度学习应该抛弃对VC维之类概念的迷信，尝试从其他方面来解释其可学习型，例如使用泛函空间（如[Banach Space](http://en.wikipedia.org/wiki/Banach_space)）中的概率论。