机器学习中的范数规则化

今天聊聊机器学习中出现的非常频繁的问题：过拟合与规则化。我们先简单的来理解下常用的 L0、L1、L2 和核范数规则化。最后聊一下规则化项参数的选择问题。

监督机器学习问题无非就是“minimize your error while regularizing your parameters”，也就是在规则化参数的同时最小化误差。最小化误差是为了让我们的模型拟合我们的训练数据，而规则化参数是防止我们的模型过分拟合我们的训练数据。多么简约的哲学啊！因为参数太多，会导致我们的模型复杂度上升，容易过拟合，也就是我们的训练误差会很小。但训练误差小并不是我们的最终目标，我们的目标是希望模型的测试误差小，也就是能准确的预测新的样本。所以，我们需要保证模型“简单”的基础上最小化训练误差，这样得到的参数才具有好的泛化性能（也就是测试误差也小），而模型“简单”就是通过规则函数来实现的。另外，规则项的使用还可以约束我们的模型的特性。这样就可以将人对这个模型的先验知识融入到模型的学习当中，强行地让学习到的模型具有人想要的特性，例如稀疏、低秩、平滑等等。要知道，有时候人的先验是非常重要的。前人的经验会让你少走很多弯路，对机器学习也是一样，如果被我们人稍微点拨一下，它肯定能更快的学习相应的任务。只是由于人和机器的交流目前还没有那么直接的方法，目前这个媒介只能由规则项来担当了。

还有几种角度来看待规则化的。规则化符合奥卡姆剃刀(Occam's razor)原理，它的思想很平易近人：在所有可能选择的模型中，我们应该选择能够很好地解释已知数据并且十分简单的模型。从贝叶斯估计的角度来看，规则化项对应于模型的先验概率。民间还有个说法就是，规则化是结构风险最小化策略的实现，是在经验风险上加一个正则化项(regularizer)或惩罚项(penalty term)。

一般来说，监督学习可以看做最小化下面的目标函数：

$$\omega^* = \arg\min_{\omega} \sum_i L(y_i, f(x_i;\omega)) + \lambda\Omega(\omega)$$

其中，第一项 $L(y_i,f(x_i;w))$ 衡量我们的模型（分类或者回归）对第 i 个样本的预测值 $f(x_i;w)$和真实的标签 y_i 之前的误差。因为我们的模型是要拟合我们的训练样本的嘛，所以我们要求这一项最小，也就是要求我们的模型尽量的拟合我们的训练数据。但正如上面说言，我们不仅要保证训练误差最小，我们更希望我们的模型测试误差小，所以我们需要加上第二项，也就是对参数 w 的规则化函数 Ω(w)去约束我们的模型尽量的简单。

如果你在机器学习浴血奋战多年，你会发现，机器学习的大部分带参模型都和这个不但形似，而且神似。是的，其实大部分无非就是变换这两项而已。对于第一项 Loss 函数，如果是 Square loss，那就是最小二乘了；如果是 Hinge Loss，那就是著名的 SVM 了；如果是 exp-Loss，那就是牛逼的 Boosting 了；如果是 log-Loss，那就是 Logistic

Regression 了；还有等等。不同的 loss 函数，具有不同的拟合特性，这个也得就具体问题具体分析的。但这里，我们先不究 loss 函数的问题，我们把目光转向“规则项 Ω(w)”。

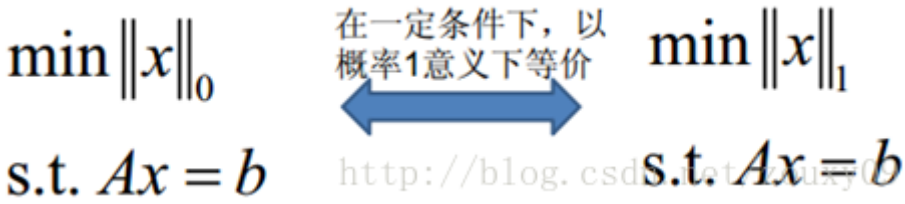
规则化函数 Ω(w)也有很多种选择，一般是模型复杂度的单调递增函数，模型越复杂，规则化值就越大。比如，规则化项可以是模型参数向量的范数。然而，不同的选择对参数 w 的约束不同，取得的效果也不同，但我们在论文中常见的都聚集在：零范数、一范数、二范数、迹范数、Frobenius 范数和核范数等等。

一、L0 范数与 L1 范数

L0 范数是指向量中非 0 的元素的个数。如果我们用 L0 范数来规则化一个参数矩阵 W 的话，就是希望 W 的大部分元素都是 0。换句话说，让参数 W 是稀疏的。看到了“稀疏”二字，大家都应该从当下风风火火的“压缩感知”和“稀疏编码”中醒悟过来，原来用的漫山遍野的“稀疏”就是通过这玩意来实现的。但你又开始怀疑了，是这样吗？看到的 papers 世界中，稀疏不是都通过 L1 范数来实现吗？脑海里是不是到处都是$||W||_1$影子呀！几乎是抬头不见低头见。没错，这就是这节的题目把 L0 和 L1 放在一起的原因，因为他们有着某种不寻常的关系。那我们再来看看 L1 范数是什么？它为什么可以实现稀疏？为什么大家都用 L1 范数去实现稀疏，而不是 L0 范数呢？

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）。现在我们来分析下这个价值一个亿的问题：为什么 L1 范数会使权值稀疏？有人可能会这样给你回答“它是 L0 范数的最优凸近似”。实际上，还存在一个更美的回答：任何的规则化算子，如果他在 $W_i=0$ 的地方不可微，并且可以分解为一个“求和”的形式，那么这个规则化算子就可以实现稀疏。这说是这么说，W 的 L1 范数是绝对值，|w|在 w=0 处是不可微，但这还是不够直观。这里因为我们需要和 L2 范数进行对比分析。所以关于 L1 范数的直观理解，请待会看看第二节。

对了，上面还有一个问题：既然 L0 可以实现稀疏，为什么不用 L0，而要用 L1 呢？个人理解一是因为 L0 范数很难优化求解（NP 难问题），二是 L1 范数是 L0 范数的最优凸近似，而且它比 L0 范数要容易优化求解。所以大家才把目光和万千宠爱转于 L1 范数。



OK，来个一句话总结：L1 范数和 L0 范数可以实现稀疏，L1 因具有比 L0 更好的优化求解特性而被广泛应用。

好，到这里，我们大概知道了 L1 可以实现稀疏，但我们会想呀，为什么要稀疏？让我们的参数稀疏有什么好处呢？这里扯两点：

1）特征选择(Feature Selection)：

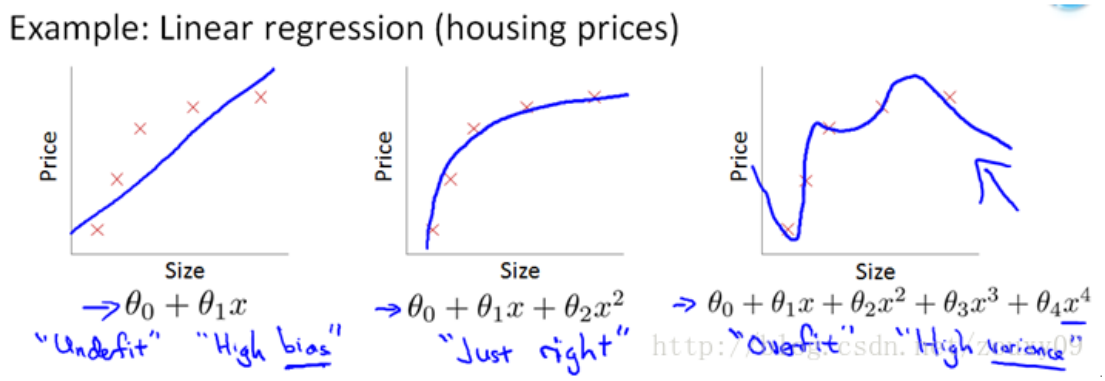
大家对稀疏规则化趋之若鹜的一个关键原因在于它能实现特征的自动选择。一般来说，x_i 的大部分元素（也就是特征）都是和最终的输出 y_i 没有关系或者不提供任何信息的，在最小化目标函数的时候考虑 x_i 这些额外的特征，虽然可以获得更小的训练误差，但在预测新的样本时，这些没用的信息反而会被考虑，从而干扰了对正确 y_i 的预测。稀疏规则化算子的引入就是为了完成特征自动选择的光荣使命，它会学习地去掉这些没有信息的特征，也就是把这些特征对应的权重置为 0。

2）可解释性(Interpretability)：

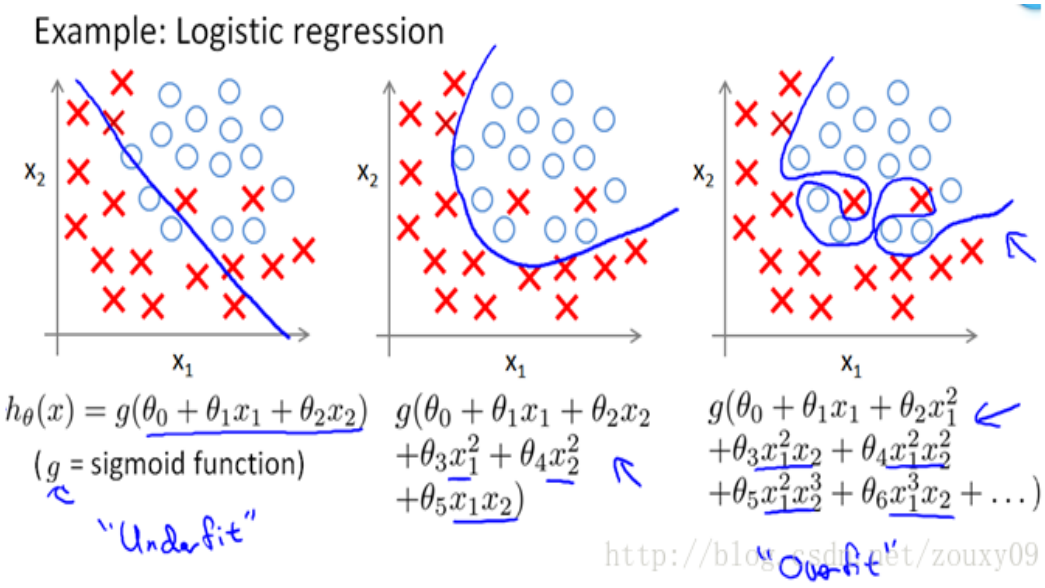
另一个青睐于稀疏的理由是，模型更容易解释。例如患某种病的概率是 y，然后我们收集到的数据 x 是 1000 维的，也就是我们需要寻找这 1000 种因素到底是怎么影响患上这种病的概率的。假设我们这个是个回归模型：$y=w_1*x_1+w_2*x_2+\ldots+w_{1000}*x_{1000}+b$（当然了，为了让 y 限定在[0,1]的范围，一般还得加个 Logistic 函数）。通过学习，如果最后学习到的 w*就只有很少的非零元素，例如只有 5 个非零的 w_i，那么我们就有理由相信，这些对应的特征在患病分析上面提供的信息是巨大的，决策性的。也就是说，患不患这种病只和这 5 个因素有关，那医生就好分析多了。但如果 1000 个 w_i 都非 0，医生面对这 1000 种因素，累觉不爱。

二、L2 范数

除了 L1 范数，还有一种更受宠幸的规则化范数是 L2 范数: $||W||_2$。它也不逊于 L1 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”。这用的很多吧，因为它的强大功效是改善机器学习里面一个非常重要的问题：过拟合。至于过拟合是什么，上面也解释了，就是模型训练时候的误差很小，但在测试的时候误差很大，也就是我们的模型复杂到可以拟合到我们的所有训练样本了，但在实际预测新的样本的时候，糟糕的一塌糊涂。通俗的讲就是应试能力很强，实际应用能力很差。擅长背诵知识，却不懂得灵活利用知识。例如下图所示（来自 Ng 的 course）：



上面的图是线性回归，下面的图是 Logistic 回归，也可以说是分类的情况。从左到右分别是欠拟合（underfitting，也称 High-bias）、合适的拟合和过拟合（overfitting，也称 High variance）三种情况。可以看到，如果模型复杂（可以拟合任意的复杂函数），它可以让我们的模型拟合所有的数据点，也就是基本上没有误差。对于回归来说，就是我们的函数曲线通过了所有的数据点，如上图右。对分类来说，就是我们的函数曲线要把所有的数据点都分类正确，如下图右。这两种情况很明显过拟合了。



那现在到我们非常关键的问题了，为什么 L2 范数可以防止过拟合？回答这个问题之前，我们得先看看 L2 范数是个什么东西。

L2 范数是指向量各元素的平方和然后求平方根。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，这里是有很大的区别的哦。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。为什么越小的参数说明模型越简单？我也不懂，我的理解是：限制了参数很小，实际上就限制了多项式某些分量的影响很小（看上面线性回归的模型的那个拟合的图），这样就相当于减少参数个数。其实我也不太懂，希望大家可以指点下。

这里也一句话总结下：通过 L2 范数，我们可以实现了对模型空间的限制，从而在一定程度上避免了过拟合。L2 范数的好处是什么呢？

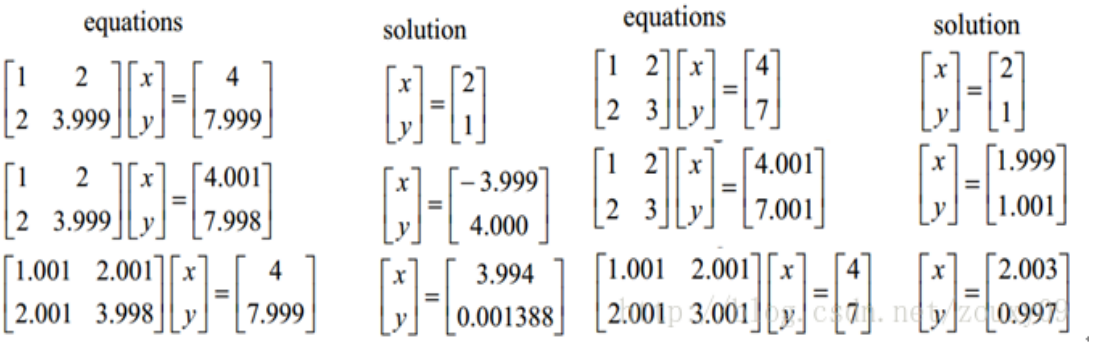
1）学习理论的角度：

从学习理论的角度来说，L2 范数可以防止过拟合，提升模型的泛化能力。

2）优化计算的角度：

从优化或者数值计算的角度来说，L2 范数有助于处理 condition number 不好的情况下矩阵求逆很困难的问题。哎，等等，这 condition number 是啥？

这里聊聊优化问题。优化有两大难题，一是：局部最小值，二是：ill-condition 病态问题。前者就不说了，大家都懂吧，我们要找的是全局最小值，如果局部最小值太多，那我们的优化算法就很容易陷入局部最小而不能自拔，这很明显不是观众愿意看到的剧情。那下面我们来聊聊 ill-condition。ill-condition 对应的是 well-condition。那他们分别代表什么？假设我们有个方程组 AX=b，我们需要求解 X。如果 A 或者 b 稍微的改变，会使得 X 的解发生很大的改变，那么这个方程组系统就是 ill-condition 的，反之就是 well-condition 的。我们具体举个例子吧：



咱们先看左边的那个。第一行假设是我们的 AX=b，第二行我们稍微改变下 b，得到的 x 和没改变前的差别很大，看到吧。第三行我们稍微改变下系数矩阵 A，可以看到结果的变化也很大。换句话来说，这个系统的解对系数矩阵 A 或者 b 太敏感了。又因为一般我们的系数矩阵 A 和 b 是从实验数据里面估计得到的，所以它是存在误差的，如果我们的系统对这个误差是可以容忍的就还好，但系统对这个误差太敏感了，以至于我们的解的误差更大，那这个解就太不靠谱了。所以这个方程组系统就是 ill-conditioned 病态的，不正常的，不稳定的，有问题的，哈哈。这清楚了吧。右边那个就叫 well-condition 的系统了。

还是再啰嗦一下吧，对于一个 ill-condition 的系统，我的输入稍微改变下，输出就发生很大的改变，这不好啊，这表明我们的系统不能实用啊。你想想看，例如对于一个回归问题 y=f(x)，我们是用训练样本 x 去训练模型 f，使得 y 尽量输出我们期待的值，例如 0。那假如我们遇到一个样本 x’，这个样本和训练样本 x 差别很小，面对他，系统本应该输出和上面的 y 差不多的值的，例如 0.00001，最后却给我输出了一个 0.9999，这很明显不对呀。就好像，你很熟悉的一个人脸上长了个青春痘，你就不认识他了，那你大脑就太差劲了，哈哈。所以如果一个系统是 ill-conditioned 病态的，我们就会对它的结果产生怀疑。那到底要相信它多少呢？我们得找个标准来衡量吧，因为有些系统的病没那么重，它的结果还是可以相信的，不能一刀切吧。终于回来了，上面的 condition number 就是拿来衡量 ill-condition 系统的可信度的。condition number 衡量的是输入发生微小变化的时候，输出会发生多大的变化。也就是系统对微小变化的敏感度。condition number 值小的就是 well-conditioned 的，大的就是 ill-conditioned 的。

如果方阵 A 是非奇异的，那么 A 的 conditionnumber 定义为：

$$\kappa(A) = \| A \| \ \| A^{-1} \|$$

也就是矩阵 A 的 norm 乘以它的逆的 norm。所以具体的值是多少，就要看选择的 norm 是什么了。如果方阵 A 是奇异的，那么 A 的 condition number 就是正无穷大了。实际上，每一个可逆方阵都存在一个 condition number。但如果要计算它，我们需要先知道这个方阵的 norm（范数）和 Machine Epsilon（机器的精度）。为什么要范数？范数就相当于衡量一个矩阵的大小，我们知道矩阵是没有大小的，当上面不是要衡量一个矩阵 A 或者向量 b 变化的时候，我们的解 x 变化的大小吗？所以肯定得要有一个东西来度量矩阵和向量的大小吧？对了，他就是范数，表示矩阵大小或者向量长度。经过比较简单的证明，对于 AX=b，我们可以得到以下的结论：

$$\frac{\|\Delta x\|}{\|x\|} \le \|A\| \cdot \|A^{-1}\| \cdot \frac{\|\Delta b\|}{\|b\|} \qquad \frac{\|\Delta x\|}{\|x\|} \le K(A) \cdot \frac{\|\Delta b\|}{\|b\|} \qquad \frac{\|\Delta x\|}{\|x + \Delta x\|} \le K(A) \frac{\|\Delta A\|}{\|A\|}$$

也就是我们的解 x 的相对变化和 A 或者 b 的相对变化是有像上面那样的关系的，其中 k(A)的值就相当于倍率，看到了吗？相当于 x 变化的界。

对 condition number 来个一句话总结：condition number 是一个矩阵（或者它所描述的线性系统）的稳定性或者敏感度的度量，如果一个矩阵的 condition number 在 1 附近，那么它就是 well-conditioned 的，如果远大于 1，那么它就是 ill-conditioned 的，如果一个系统是 ill-conditioned 的，它的输出结果就不要太相信了。

好了，对这么一个东西，已经说了好多了。对了，我们为什么聊到这个的了？回到第一句话：从优化或者数值计算的角度来说，L2 范数有助于处理 condition number 不好的情况下矩阵求逆很困难的问题。因为目标函数如果是二次的，对于线性回归来说，那实际上是有解析解的，求导并令导数等于零即可得到最优解为：

$$\hat{\mathbf{w}} = \left(X^T X\right)^{-1} X^T \mathbf{y}$$

然而，如果当我们的样本 X 的数目比每个样本的维度还要小的时候，矩阵 X^TX 将会不是满秩的，也就是 X^TX 会变得不可逆，所以 w*就没办法直接计算出来了。或者更确切地说，将会有无穷多个解（因为我们方程组的个数小于未知数的个数）。也就是说，我们的数据不足以确定一个解，如果我们从所有可行解里随机选一个的话，很可能并不是真正好的解，总而言之，我们过拟合了。

但如果加上 L2 规则项，就变成了下面这种情况，就可以直接求逆了：

$$w^* = \left(X^T X + \lambda I\right)^{-1} X^T y$$

这里面，专业点的描述是：要得到这个解，我们通常并不直接求矩阵的逆，而是通过解线性方程组的方式（例如高斯消元法）来计算。考虑没有规则项的时候，也就是 λ=0 的情况，如果矩阵 X^TX 的 condition number 很大的话，解线性方程组就会在数值上相当不稳定，而这个规则项的引入则可以改善 condition number。

另外，如果使用迭代优化的算法，condition number 太大仍然会导致问题：它会拖慢迭代的收敛速度，而规则项从优化的角度来看，实际上是将目标函数变成 λ-strongly convex（λ 强凸）的了。哎哟哟，这里又出现个 λ 强凸，啥叫 λ 强凸呢？

当 f 满足：

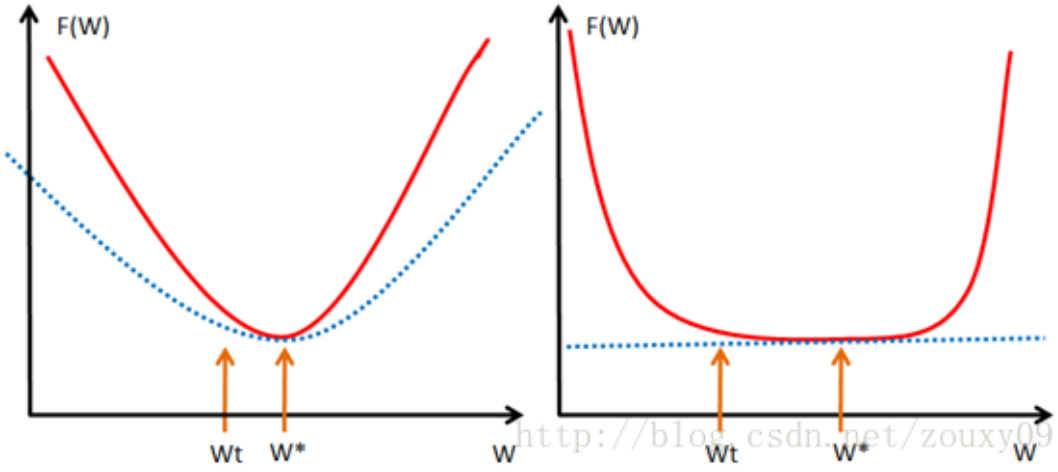
$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\lambda}{2} \| y - x \|^2$$

时，我们称 f 为 λ-stronglyconvex 函数，其中参数 λ>0。当 λ=0 时退回到普通 convex 函数的定义。

在直观的说明强凸之前，我们先看看普通的凸是怎样的。假设我们让 f 在 x 的地方做一阶泰勒近似（一阶泰勒展开忘了吗？f(x)=f(a)+f'(a)(x-a)+o(||x-a||).）：

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + o(\| y - x \|)$$

直观来讲，convex 性质是指函数曲线位于该点处的切线，也就是线性近似之上，而 strongly convex 则进一步要求位于该处的一个二次函数上方，也就是说要求函数不要太“平坦”而是可以保证有一定的“向上弯曲”的趋势。专业点说，就是 convex 可以保证函数在任意一点都处于它的一阶泰勒函数之上，而 strongly convex 可以保证函数在任意一点都存在一个非常漂亮的二次下界 quadratic lower bound。当然这是一个很强的假设，但是同时也是非常重要的假设。可能还不好理解，那我们画个图来形象的理解下。



大家一看到上面这个图就全明白了吧。我们取我们的最优解 w*的地方。如果我们的函数 f(w)，见左图，也就是红色那个函数，都会位于蓝色虚线的那根二次函数之上，

这样就算 w_t 和 w*离的比较近的时候，$f(w_t)$和 f(w*)的值差别还是挺大的，也就是会保证在我们的最优解 w*附近的时候，还存在较大的梯度值，这样我们才可以在比较少的迭代次数内达到 w*。但对于右图，红色的函数 f(w)只约束在一个线性的蓝色虚线之上，假设是如右图的很不幸的情况（非常平坦），那在 w_t 还离我们的最优点 w*很远的时候，我们的近似梯度$(f(w_t)-f(w^*))/(w_t-w^*)$就已经非常小了，在 w_t 处的近似梯度$\partial f/\partial w$ 就更小了，这样通过梯度下降 $w_{t+1}=w_t-\alpha*(\partial f/\partial w)$，我们得到的结果就是 w 的变化非常缓慢，像蜗牛一样，非常缓慢的向我们的最优点 w*爬动，那在有限的迭代时间内，它离我们的最优点还是很远。

所以仅仅靠 convex 性质并不能保证在梯度下降和有限的迭代次数的情况下得到的点 w 会是一个比较好的全局最小点 w*的近似点（插个话，有地方说，实际上让迭代在接近最优的地方停止，也是一种规则化或者提高泛化性能的方法）。正如上面分析的那样，如果 f(w)在全局最小点 w*周围是非常平坦的情况的话，我们有可能会找到一个很远的点。但如果我们有“强凸”的话，就能对情况做一些控制，我们就可以得到一个更好的近似解。至于有多好嘛，这里面有一个 bound，这个 bound 的好坏也要取决于 strongly convex 性质中的常数 α 的大小。看到这里，不知道大家学聪明了没有。如果要获得 strongly convex 怎么做？最简单的就是往里面加入一项$(\alpha/2)*\|w\|^2$。

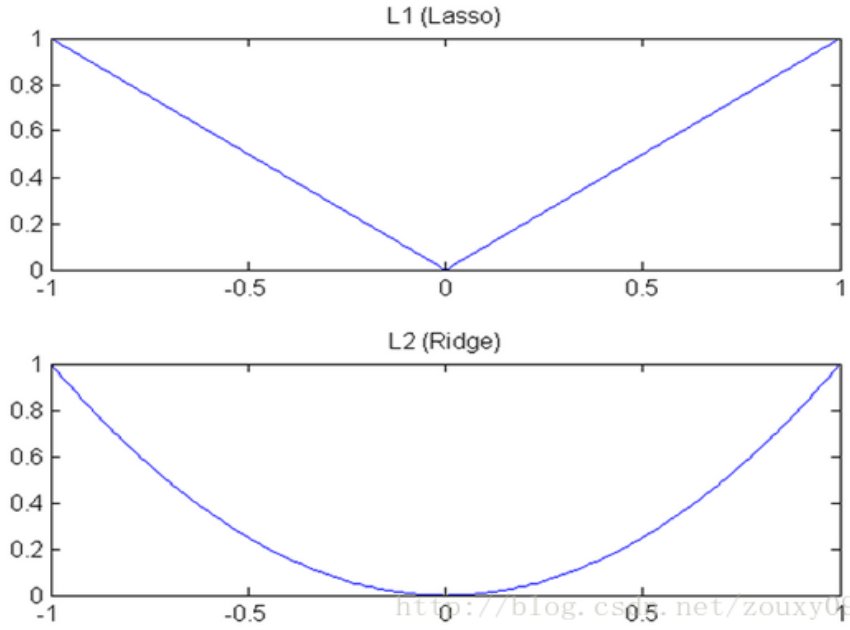
呃，讲个 strongly convex 花了那么多的篇幅。实际上，在梯度下降中，目标函数收敛速率的上界实际上是和矩阵 X^TX 的 condition number 有关，X^TX 的 condition number 越小，上界就越小，也就是收敛速度会越快。

这一个优化说了那么多的东西。还是来个一句话总结吧：L2 范数不但可以防止过拟合，还可以让我们的优化求解变得稳定和快速。

好了，这里兑现上面的承诺，来直观的聊聊 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？我看到的有两种几何上直观的解析：

1）下降速度：

我们知道，L1 和 L2 都是规则化的方式，我们将权值参数以 L1 或者 L2 的方式放到代价函数里面去。然后模型就会尝试去最小化这些权值参数。而这个最小化就像一个下坡的过程，L1 和 L2 的差别就在于这个“坡”不同，如下图：L1 就是按绝对值函数的“坡”下降的，而 L2 是按二次函数的“坡”下降。所以实际上在 0 附近，L1 的下降速度比 L2 的下降速度要快。所以会非常快得降到 0。不过我觉得这里解释的不太中肯，当然了也不知道是不是自己理解的问题。



L1 在江湖上人称 Lasso，L2 人称 Ridge。不过这两个名字还挺让人迷糊的，看上面的图片，Lasso 的图看起来就像 ridge，而 ridge 的图看起来就像 lasso。

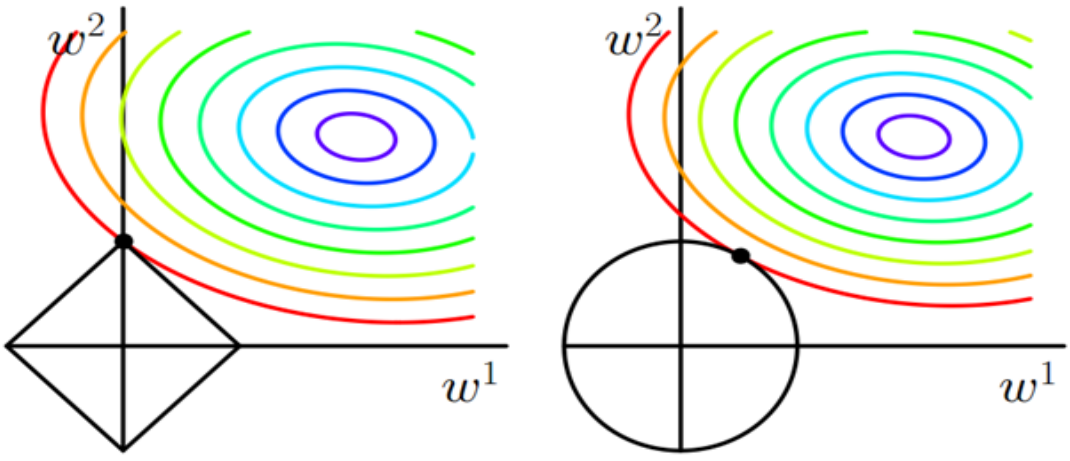
2）模型空间的限制：

实际上，对于 L1 和 L2 规则化的代价函数来说，我们可以写成以下形式：

$$Lasso: \quad \min_w \frac{1}{n} \| y - X w \|^2, \quad s.t. \| w \|_1 \le C$$

$$Ridge: \quad \min_w \frac{1}{n} \| y - X w \|^2, \quad s.t. \| w \|_2 \le C$$

也就是说，我们将模型空间限制在 w 的一个 L1-ball 中。为了便于可视化，我们考虑两维的情况，在(w1, w2)平面上可以画出目标函数的等高线，而约束条件则成为平面上半径为 C 的一个 norm ball 。等高线与 norm ball 首次相交的地方就是最优解：



(a) ℓ_1-ball meets quadratic function. ℓ_1-ball has corners. It's very likely that the meet-point is at one of the corners.

(b) ℓ_2-ball meets quadratic function. ℓ_2-ball has no corner. It is very unlikely that the meet-point is on any of axes.

可以看到，L1-ball 与 L2-ball 的不同就在于 L1 在和每个坐标轴相交的地方都有“角”出现，而目标函数的测地线除非位置摆得非常好，大部分时候都会在角的地方相交。注意到在角的位置就会产生稀疏性，例如图中的相交点就有 w1=0，而更高维的时候（想象一下三维的 L1-ball 是什么样的？）除了角点以外，还有很多边的轮廓也是既有很大的概率成为第一次相交的地方，又会产生稀疏性。

相比之下，L2-ball 就没有这样的性质，因为没有角，所以第一次相交的地方出现在具有稀疏性的位置的概率就变得非常小了。这就从直观上来解释了为什么 L1-regularization 能产生稀疏性，而 L2-regularization 不行的原因了。

因此，一句话总结就是：L1 会趋向于产生少量的特征，而其他的特征都是 0，而 L2 会选择更多的特征，这些特征都会接近于 0。Lasso 在特征选择时候非常有用，而 Ridge 就只是一种规则化而已。

三、核范数

核范数$||W||_*$是指矩阵奇异值的和，英文称呼叫 Nuclear Norm。这个相对于上面火热的 L1 和 L2 来说，可能大家就会陌生点。那它是干嘛用的呢？霸气登场：约束 Low-Rank（低秩）。OK，OK，那我们得知道 Low-Rank 是啥？用来干啥的？

我们先来回忆下线性代数里面“秩”到底是啥？举个简单的例子吧：

$$\begin{cases} x_1 - x_2 + x_3 = 5 \\ x_1 + x_2 + x_3 = 7 \\ 2x_1 - 2x_2 + 2x_3 = 14 \end{cases}.$$

对上面的线性方程组，第一个方程和第二个方程有不同的解，而第 2 个方程和第 3 个方程的解完全相同。从这个意义上说，第 3 个方程是“多余”的，因为它没有带来任何的信息量，把它去掉，所得的方程组与原来的方程组同解。为了从方程组中去掉多余的方程，自然就导出了“矩阵的秩”这一概念。

还记得我们怎么手工求矩阵的秩吗？为了求矩阵 A 的秩，我们是通过矩阵初等变换把 A 化为阶梯型矩阵，若该阶梯型矩阵有 r 个非零行，那 A 的秩 rank(A)就等于 r。从物理意义上讲，矩阵的秩度量的就是矩阵的行列之间的相关性。如果矩阵的各行或列是线性无关的，矩阵就是满秩的，也就是秩等于行数。回到上面线性方程组来说吧，因为线性方程组可以用矩阵描述嘛。秩就表示了有多少个有用的方程了。上面的方程组有 3 个方程，实际上只有 2 个是有用的，一个是多余的，所以对应的矩阵的秩就是 2 了。

既然秩可以度量相关性，而矩阵的相关性实际上有带有了矩阵的结构信息。如果矩阵之间各行的相关性很强，那么就表示这个矩阵实际可以投影到更低维的线性子空间，也就是用几个向量就可以完全表达了，它就是低秩的。所以我们总结的一点就是：如果

矩阵表达的是结构性信息，例如图像、用户-推荐表等等，那么这个矩阵各行之间存在这一定的相关性，那这个矩阵一般就是低秩的。

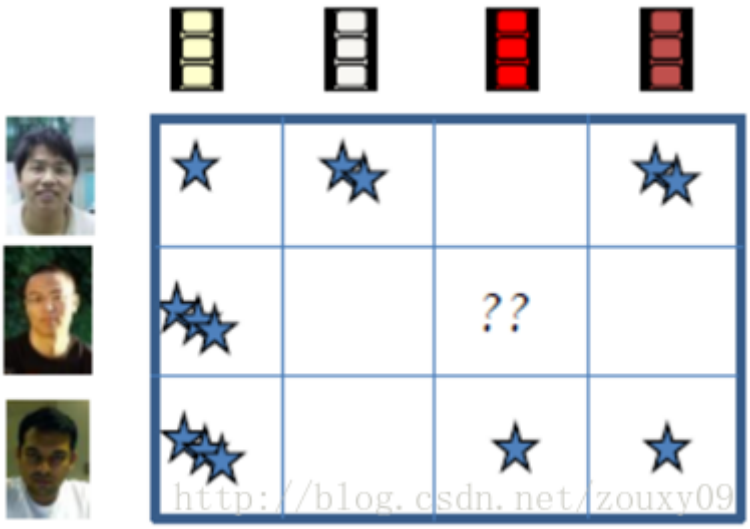
如果 X 是一个 m 行 n 列的数值矩阵，rank(X)是 X 的秩，假如 rank (X)远小于 m 和 n，则我们称 X 是低秩矩阵。低秩矩阵每行或每列都可以用其他的行或列线性表出，可见它包含大量的冗余信息。利用这种冗余信息，可以对缺失数据进行恢复，也可以对数据进行特征提取。

好了，低秩有了，那约束低秩只是约束 rank(w)呀，和我们这节的核范数有什么关系呢？他们的关系和 L0 与 L1 的关系一样。因为 rank()是非凸的，在优化问题里面很难求解，那么就需要寻找它的凸近似来近似它了。对，你没猜错，rank(w)的凸近似就是核范数$\|W\|_*$.

好了，到这里，我也没什么好说的了，因为我也是稍微翻看了下这个东西，所以也还没有深入去看它。但我发现了这玩意还有很多很有意思的应用，下面我们举几个典型的吧。

1）矩阵填充(Matrix Completion)：

我们首先说说矩阵填充用在哪。一个主流的应用是在推荐系统里面。我们知道，推荐系统有一种方法是通过分析用户的历史记录来给用户推荐的。例如我们在看一部电影的时候，如果喜欢看，就会给它打个分，例如 3 颗星。然后系统，例如 Netflix 等知名网站就会分析这些数据，看看到底每部影片的题材到底是怎样的？针对每个人，喜欢怎样的电影，然后会给对应的用户推荐相似题材的电影。但有一个问题是：我们的网站上面有非常多的用户，也有非常多的影片，不是所有的用户都看过说有的电影，不是所有看过某电影的用户都会给它评分。假设我们用一个“用户-影片”的矩阵来描述这些记录，例如下图，可以看到，会有很多空白的地方。如果这些空白的地方存在，我们是很难对这个矩阵进行分析的，所以在分析之前，一般需要先对其进行补全。也叫矩阵填充。



那到底怎么填呢？如何才能无中生有呢？每个空白的地方的信息是否蕴含在其他已有的信息之上了呢？如果有，怎么提取出来呢？Yeah，这就是低秩生效的地方了。这叫低秩矩阵重构问题，它可以用如下的模型表述：已知数据是一个给定的 m*n 矩阵 A，如果其中一些元素因为某种原因丢失了，我们能否根据其他行和列的元素，将这些元素恢复？当然，如果没有其他的参考条件，想要确定这些数据很困难。但如果我们已知 A 的秩 rank(A)<<m 且 rank(A)<<n，那么我们可以通过矩阵各行(列)之间的线性相关将丢失的元素求出。你会问，这种假定我们要恢复的矩阵是低秩的，合理吗？实际上是十分合理的，比如一个用户对某电影评分是其他用户对这部电影评分的线性组合。所以，通过低秩重构就可以预测用户对其未评价过的视频的喜好程度。从而对矩阵进行填充。

2）鲁棒 PCA：

主成分分析，这种方法可以有效的找出数据中最“主要"的元素和结构，去除噪音和冗余，将原有的复杂数据降维，揭示隐藏在复杂数据背后的简单结构。我们知道，最简单的主成分分析方法就是 PCA 了。从线性代数的角度看，PCA 的目标就是使用另一组基去重新描述得到的数据空间。希望在这组新的基下，能尽量揭示原有的数据间的关系。这个维度即最重要的“主元"。PCA 的目标就是找到这样的“主元”，最大程度的去除冗余和噪音的干扰。

鲁棒主成分分析（Robust PCA）考虑的是这样一个问题：一般我们的数据矩阵 X 会包含结构信息，也包含噪声。那么我们可以将这个矩阵分解为两个矩阵相加，一个是低秩的（由于内部有一定的结构信息，造成各行或列间是线性相关的），另一个是稀疏的（由于含有噪声，而噪声是稀疏的），则鲁棒主成分分析可以写成以下的优化问题：

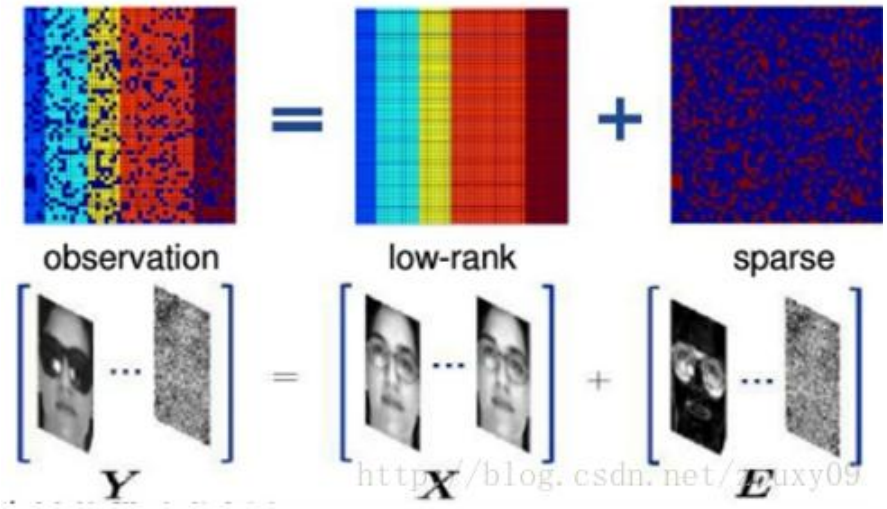
$$\min_{A,E} rank(A)+\lambda||E||_0 \quad s.t\ X=A+E$$

与经典 PCA 问题一样，鲁棒 PCA 本质上也是寻找数据在低维空间上的最佳投影问题。对于低秩数据观测矩阵 X，假如 X 受到随机（稀疏）噪声的影响，则 X 的低秩性就会破坏，使 X 变成满秩的。所以我们就需要将 X 分解成包含其真实结构的低秩矩阵和稀疏噪声矩阵之和。找到了低秩矩阵，实际上就找到了数据的本质低维空间。那有了 PCA，为什么还有这个 Robust PCA 呢？Robust 在哪？因为 PCA 假设我们的数据的噪声是高斯的，对于大的噪声或者严重的离群点，PCA 会被它影响，导致无法正常工作。而 Robust PCA 则不存在这个假设。它只是假设它的噪声是稀疏的，而不管噪声的强弱如何。

由于 rank 和 L0 范数在优化上存在非凸和非光滑特性，所以我们一般将它转换成求解以下一个松弛的凸优化问题：

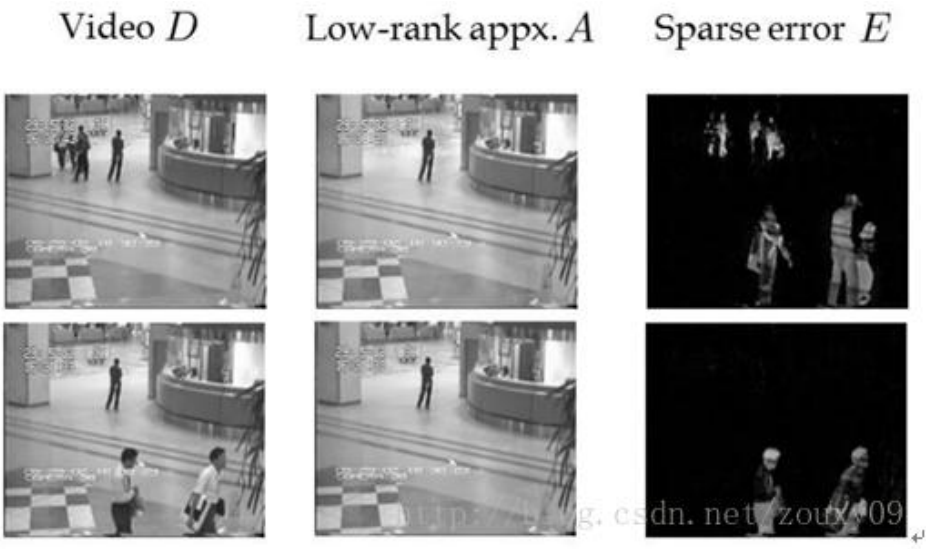
$$\min_{A,E} ||A||_*+\lambda||E||_1 \quad s.t\ X=A+E$$

说个应用吧。考虑同一副人脸的多幅图像，如果将每一副人脸图像看成是一个行向量，并将这些向量组成一个矩阵的话，那么可以肯定，理论上，这个矩阵应当是低秩的。但是，由于在实际操作中，每幅图像会受到一定程度的影响，例如遮挡，噪声，光照变化，平移等。这些干扰因素的作用可以看做是一个噪声矩阵的作用。所以我们可以把我们的同一个人脸的多个不同情况下的图片各自拉长一列，然后摆成一个矩阵，对这个矩阵进行低秩和稀疏的分解，就可以得到干净的人脸图像（低秩矩阵）和噪声的矩阵了（稀疏矩阵），例如光照，遮挡等等。



3）背景建模：

背景建模的最简单情形是从固定摄相机拍摄的视频中分离背景和前景。我们将视频图像序列的每一帧图像像素值拉成一个列向量，那么多个帧也就是多个列向量就组成了一个观测矩阵。由于背景比较稳定，图像序列帧与帧之间具有极大的相似性，所以仅由背景像素组成的矩阵具有低秩特性；同时由于前景是移动的物体，占据像素比例较低，故前景像素组成的矩阵具有稀疏特性。视频观测矩阵就是这两种特性矩阵的叠加，因此，可以说视频背景建模实现的过程就是低秩矩阵恢复的过程。



4）变换不变低秩纹理（TILT）：

以上章节所介绍的针对图像的低秩逼近算法，仅仅考虑图像样本之间像素的相似性，却没有考虑到图像作为二维的像素集合，其本身所具有的规律性。事实上，对于未加旋转的图像，由于图像的对称性与自相似性，我们可以将其看做是一个带噪声的低秩矩阵。当图像由端正发生旋转时，图像的对称性和规律性就会被破坏，也就是说各行像素间的线性相关性被破坏，因此矩阵的秩就会增加。

低秩纹理映射算法(TransformInvariant Low-rank Textures，TILT)是一种用低秩性与噪声的稀疏性进行低秩纹理恢复的算法。它的思想是通过几何变换 τ 把 D 所代表的图像区域校正成正则的区域，如具有横平竖直、对称等特性，这些特性可以通过低秩性来进行刻画。

低秩的应用非常多，大家有兴趣的可以去找些资料深入了解下。

四、规则化参数的选择

现在我们回过头来看看我们的目标函数：

$$w^* = \arg\min_w \sum_i L(y_i, f(x_i; w)) + \lambda\Omega(w)$$

里面除了 loss 和规则项两块外，还有一个参数 λ。它也有个霸气的名字，叫 hyper-parameters（超参）。你不要看它势单力薄的，它非常重要。它的取值很大时候会决定我们的模型的性能，事关模型生死。它主要是平衡 loss 和规则项这两项的，λ 越大，就表示规则项要比模型训练误差更重要，也就是相比于要模型拟合我们的数据，我们更希望我们的模型能满足我们约束的 Ω(w)的特性。反之亦然。举个极端情况，例如 λ=0 时，就没有后面那一项，代价函数的最小化全部取决于第一项，也就是集全力使得输出和期待输出差别最小，那什么时候差别最小啊，当然是我们的函数或者曲线可以经过所有的点了，这时候误差就接近 0，也就是过拟合了。它可以复杂的代表或者记忆所有这些样本，但对于一个新来的样本泛化能力就不行了。毕竟新的样本会和训练样本有差别的嘛。

那我们真正需要什么呢？我们希望我们的模型既可以拟合我们的数据，又具有我们约束它的特性。只有它们两者的完美结合，才能让我们的模型在我们的任务上发挥强大的性能。所以如何讨好它，是非常重要。在这点上，大家可能深有体会。还记得你复现了很多论文，然后复现出来的代码跑出来的准确率没有论文说的那么高，甚至还差之万里。这时候，你就会怀疑，到底是论文的问题，还是你实现的问题？实际上，除了这两个问题，我们还需要深入思考另一个问题：论文提出的模型是否具有 hyper-parameters？论文给出了它们的实验取值了吗？经验取值还是经过交叉验证的取值？这个问题是逃不掉的，因为几乎任何一个问题或者模型都会具有 hyper-parameters，只是有时候它是隐藏着的，你看不到而已，但一旦你发现了，证明你俩有缘，那请试着去修改下它吧，有可能有“奇迹”发生哦。

回到问题本身。我们选择参数 λ 的目标是什么？我们希望模型的训练误差和泛化能力都很强。这时候，你有可能还反映过来，这不是说我们的泛化性能是我们的参数 λ 的函数吗？那我们为什么按优化那一套，选择能最大化泛化性能的 λ 呢？Oh，sorry to tell you that，因为泛化性能并不是 λ 的简单的函数！它具有很多的局部最大值！而且它的搜索空间很大。所以大家确定参数的时候，一是尝试很多的经验值，这和那些在这个领域摸爬打滚的大师是没得比的。当然了，对于某些模型，大师们也整理了些调参经验给我们。例如 Hinton 大哥的那篇 A Practical Guide to Training RestrictedBoltzmann Machines 等等。还有一种方法是通过分析我们的模型来选择。怎么做呢？就是在训练之前，我们大概计算下这时候的 loss 项的值是多少？Ω(w)的值是多少？然后针对他们的比例来确定我们的 λ，这种启发式的方法会缩小我们的搜索空间。另外一种最常见的方法就是交叉验证 Cross validation 了。先把我们的训练数据库分成几份，然后取一部分做训练集，一部分做测试集，然后选择不同的 λ 用这个训练集来训练 N 个模型，然后用这个测试集来测试我们的模型，取 N 模型里面的测试误差最小对应的 λ 来作为我们最终的 λ。如果我们的模型一次训练时间就很长了，那么很明显在有限的时间内，我们只能测试非常少的 λ。例如假设我们的模型需要训练 1 天，这在深度学习里面是家常便饭了，然后我们有一个星期，那我们只能测试 7 个不同的 λ。这就让你遇到最好的 λ 那是上辈子积下来的福气了。那有什么方法呢？两种：一是尽量测试 7 个比较靠谱的 λ，或者说 λ 的搜索空间我们尽量广点，所以一般对 λ 的搜索空间的选择一般就是 2 的多少次方了，从-10 到 10 啊什么的。但这种方法还是不大靠谱，最好的方法还是尽量让我们的模型训练的时间减少。例如假设我们优化了我们的模型训练，使得我们的训练时间减少到 2 个小时。那么一个星期我们就可以对模型训练 7*24/2=84 次，也就是说，我们可以在 84 个 λ 里面寻找最好的 λ。这让你遇见最好的 λ 的概率就大多了吧。这就是为什么我们要选择优化也就是收敛速度快的算法，为什么要用 GPU、多核、集群等来进行模型训练、为什么具有强大计算机资源的工业界能做很多学术界也做不了的事情（当然了，大数据也是一个原因）的原因了。