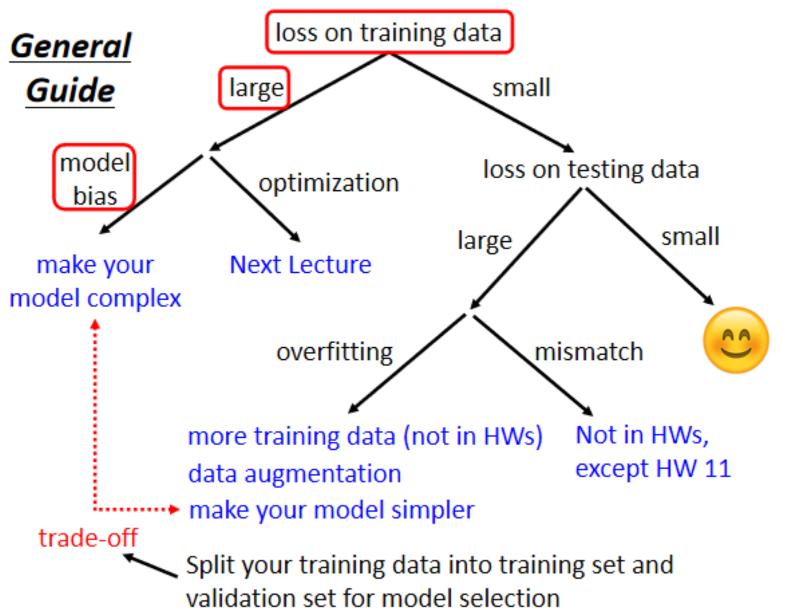
7月11日

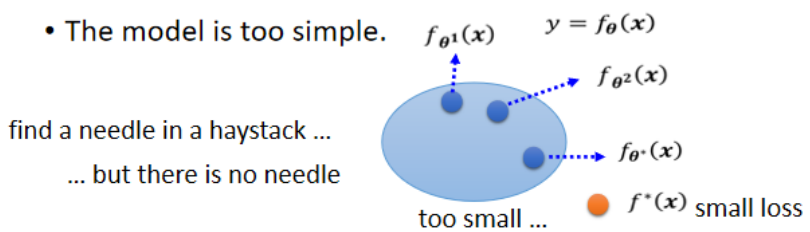
一、李宏毅2021春机器学习课程第二节：机器学习任务攻略

如何做的更好?



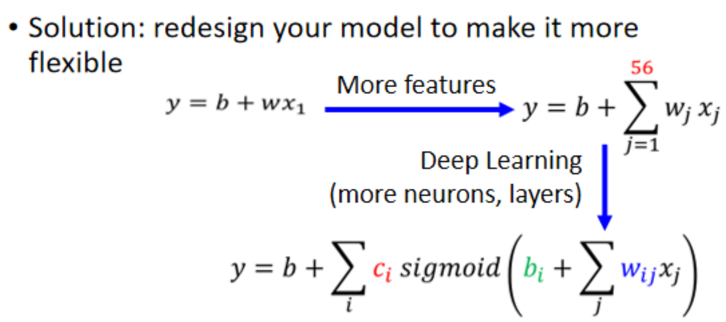
如果在Kaggle上的结果不满意的话，第一件事情就是**检查你的training data的loss**。如果你发现你的模型在**training data的loss很大**，说明它**在训练集上面也没有训练好**，这边有两个可能的原因，第一个是model的bias。

Model bias



问题原因：你的**model太过简单**，function的set太小了，这个function的set中没有包含任何一个function，可以让我们的loss变低，**即可以让loss变低的function，不在你的model可以描述的范围内。**

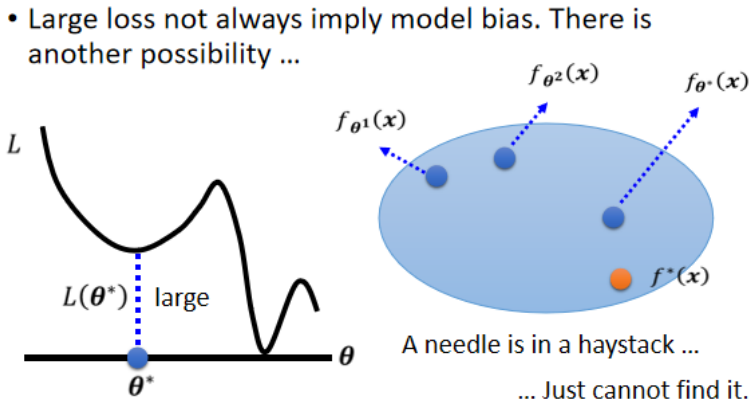
用个比喻来说：这就好像是我们想**大海捞针，但针根本就不在海里，所以任何努力都是徒劳。**



解决方法：**重新设计一个model，给你的model更大的弹性**，举例来说，你可以增加你输入的features，也可以使用Deep Learning，增加网络的层数和复杂度。

但是并不是training的时候，loss大就代表一定是model bias，你可能会遇到另外一个问题，还有可能是**optimization做得不好。**

Optimization Issue

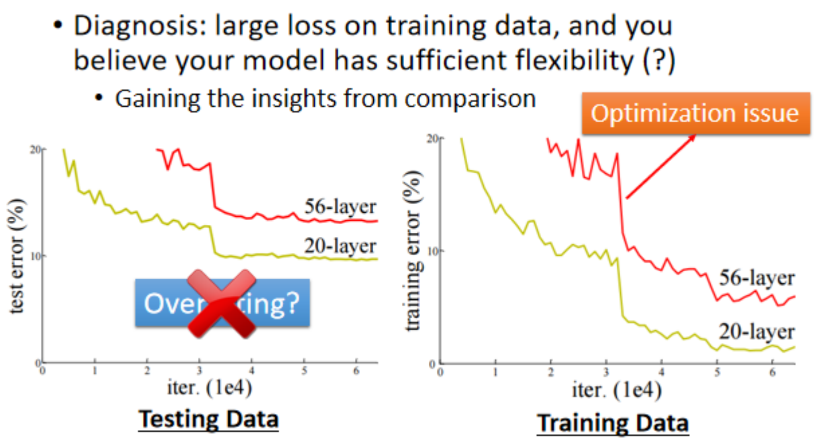


问题原因：你可能会卡在**local minima**的地方，没有办法找到一个真的可以让loss很低的参数就停下了。

用个比喻来说：这就好像是我们想**大海捞针，针确实在海里，但是我们却没有办法把针捞起来。**

那么**training data的loss不够低的时候,到底是model bias,还是optimization的问题呢?**

一个建议判断的方法，就是你可以**通过比较不同的模型，来得知你的model现在到底够不够大**：



举一个例子，这一个实验是从residual network那篇paper里面摘录出来的 (http://arxiv.org/abs/1512.03385)。

这里想测2个networks，一个20层，一个56层，训练之后发现**20层的loss比较低，56层的loss反而比较高，但这个不是overfitting，并不是所有的结果不好，都叫做overfitting。**

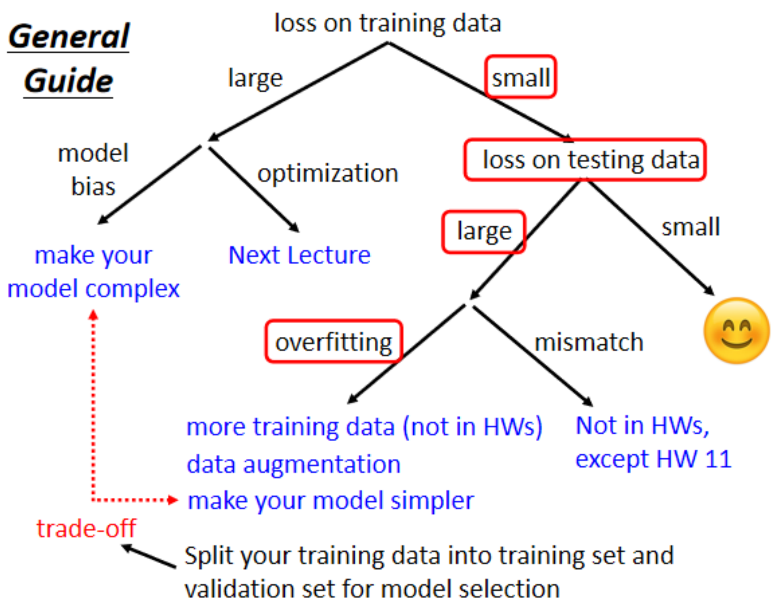
你要**检查一下训练集上面的结果**，发现在训练集上，56层的network loss就比20层的network loss高了，**这代表56层的network，它的optimization没有做好。**之所以能下这个结论，是因为理论上56层的network一定可以做到20层的network能做到的事情（它只要前20层的参数,跟这个20层的network一样，剩下36层什么事都不做）。

Start from shallower networks

一个小建议：**看到一个你从来没有做过的问，也许你可以先跑一些比较小的，比较浅的network，甚至用一些不是deep learning的方法**，比如说 linear model，比如说support vector machine，它们可能是比较容易做Optimize的，比较不会有optimization失败的问题。先有个概念说，这些简单的model，到底可以得到什么样的loss，这样也就有了一个**参考的基准点。**

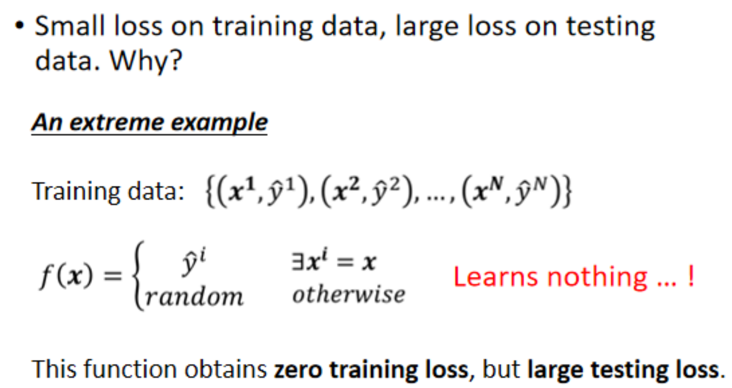
> If deeper networks do not obtain smaller loss on training data，then there is optimization issue.

解决方法：更换Optimization的策略，在SGD上加momentum，改用其他策略等等，下节课会具体讲到。



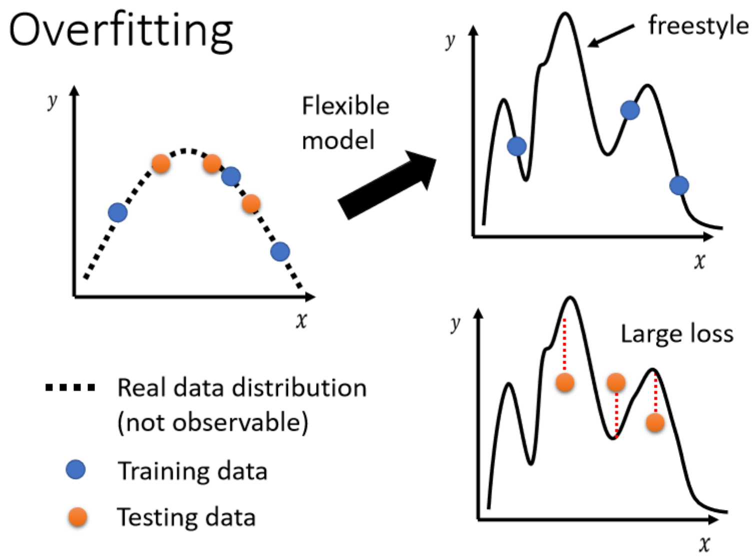
假设你现在经过一番努力，已经可以让你的training data的loss很小了，那接下来就可以看看testing data loss的情况，**如果是training的loss小，testing的loss大，这个有可能是真的遇到overfitting问题了。**

Overfitting



问题描述：举一个比较极端的例子，假如我们有一个一无是处的function：如果今天x当做输入的时候，我们就去比对这个x有没有出现在训练集里面，如果x出现在训练集里面，就把它对应的ŷ当做输出，如果x没有出现在训练集里面，就输出一个随机的值。

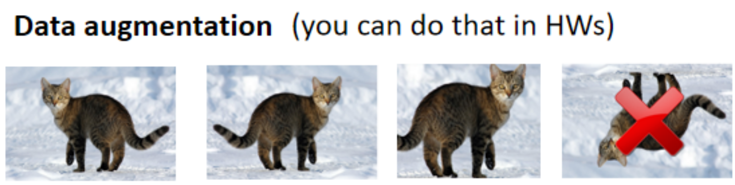
那你可以想像这个function啥事也没有干，但是**在training的data上，它的loss可是0呢！**可是在testing data上面，它的loss会变得很大，因为**它其实什么都没有预测。**



如果你的model它的自由度很大的话，它可以产生非常奇怪的曲线，导致训练集上的结果好，但是测试集上的loss很大。

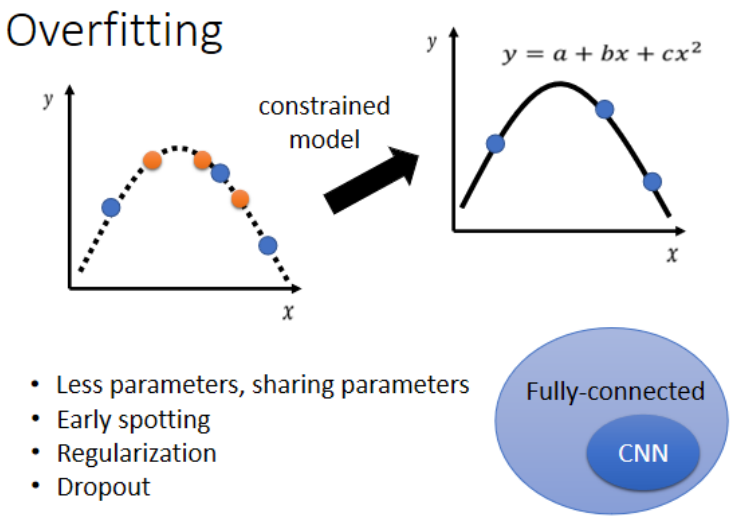
解决方法：

1. 第一个方向是，往往也是最有效的方向，那就是**增加你的训练集**。但是人工搜集训练集往往成本很高，可以使用**data augmentation**技术，注意**很少看到有人把影像上下颠倒**当作augmentation，也就是说你使用这个技术必须要是**reasonable**的，并不是随意的。



2. 第二个方向是，对你的模型进行一定的限制，让其不要有那么大的弹性。那你可能会问我怎么会知道**要用多constrain的model才会好呢**，这就**取决与你对这个问题的理解，对于数据产生背后原理的理解**。

那么又有哪些方法可以给model制造限制呢？

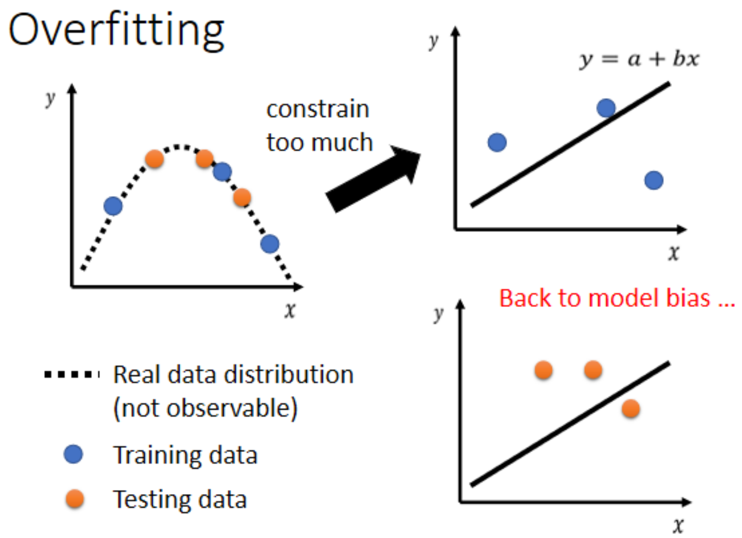


1. **给它比较少的参数**，如果是deep learning的话，就给它**比较少的神经元的数目。**或者是你可以让model共用参数，你可以让一些参数有一样的数值。我们之前讲的network的架构，叫做fully-connected network，那fully-connected network其实是一个比较有弹性的架构，而CNN是一个

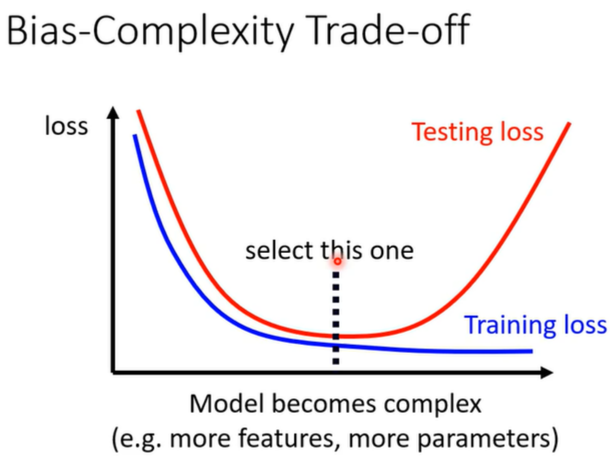
比较有限制的架构，它是针对影像的特性，来限制模型的弹性，就是因为CNN给了比较大的限制，所以CNN在影像上反而会做得比较好。

2. **用比较少的features**，本来给三天的资料，改成用给两天的资料，其实结果就好了一些。
3. 采用**Early stopping**，[(17条消息) 深度学习技巧之Early Stopping（早停法）*df19900725的博客-CSDN博客*early stopping]，基本含义是在训练中计算模型在验证集上的表现，**当模型在验证集上的表现开始下降的时候，停止训练**，这样就能避免继续训练导致过拟合的问题。
4. **Regularization**，[机器学习之正则化（Regularization） - Acjx - 博客园 (cnblogs.com)]，**在代价函数中加入惩罚项，对于太过复杂的模型进行惩罚。**
5. **Dropout**，[(17条消息) 深度学习中Dropout原理解析*Microstrong-CSDN博客*dropout]，Dropout说的简单一点就是：我们在前向传播的时候，**让某个神经元的激活值以一定的概率p停止工作**，这样可以使模型泛化性更强，因为它**不会太依赖某些局部的特征。**

但是我们也不要给模型太多的限制，不然我们又会回到model bias的问题。

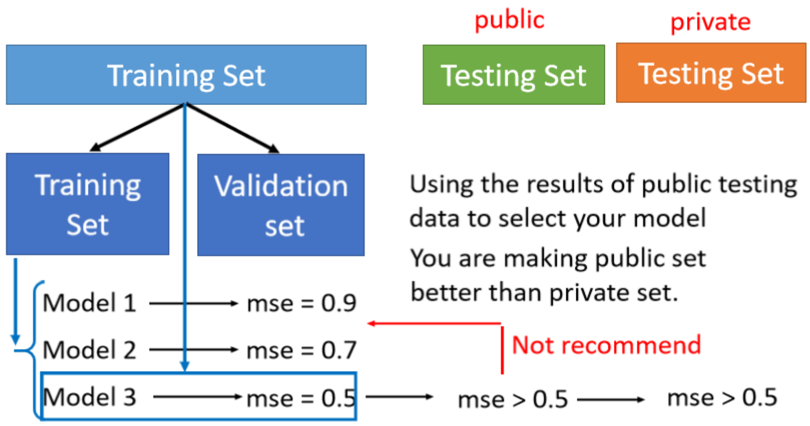


所以要选择**既不简单也不复杂**的模型：

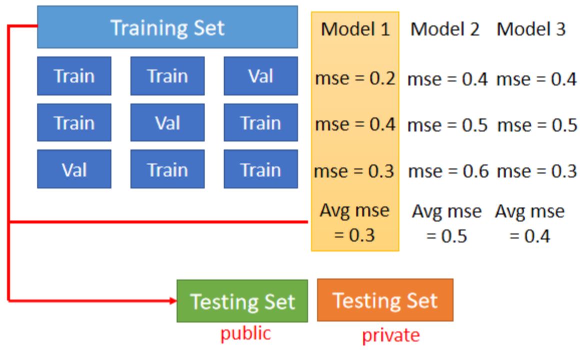


Cross Validation

把Training的资料分成两半，一部分叫作**Training Set**，另一部分用作**Validation Set**，**在Training Set上训练出来的模型，在Validation Set上面去衡量它们的分数**，最后根据Validation Set上面的分数，去挑选结果，这样就可以很大程度上避免在public上面结果很好，但是在private上面结果很差的情况。



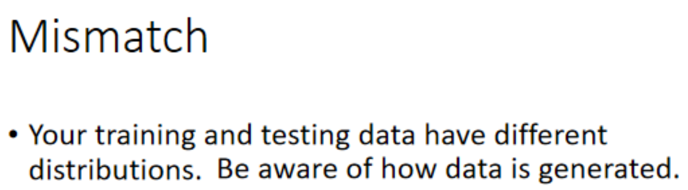
但是这边会有一个问题，就是怎么分Training Set和Validation Set呢，一般就是随机分的，但是如果担心分到很奇怪的Validation Set导致结果很差，那么推荐使用N-fold Cross Validation的方法。



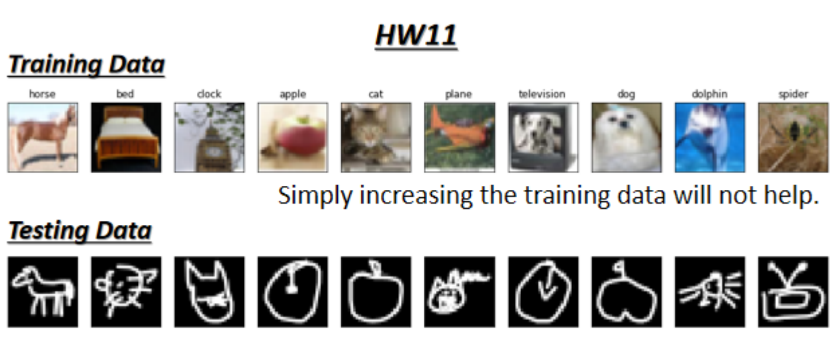
N-fold Cross Validation：[[深度概念]·K-Fold 交叉验证 (Cross-Validation)的理解与应用 - 小宋是呢 - 博客园 (cnblogs.com)] 就是你先**把你的训练集切成N等份**，在这个例子里面我们切成三等份，切完以后，你拿其中**一份当作Validation Set**，**另外两份当Training Set**，然后这件事情你要**重复三次**。

然后接下来你有三个模型，你不知道哪一个是好的，你就把这三个模型在这三个setting下，在这三个Training跟Validation的data set上面，通通跑过一次，**然后把这三个模型，在这三种状况的结果都平均起来**，再看看谁的结果最好。

Mismatch



mismatch的原因跟overfitting其实不一样，一般的overfitting，你可以用搜集更多的资料来克服，但是**mismatch意思是说，你今天的训练集跟测试集，它们的分布是不一样的**。就比如HW11中的情况：

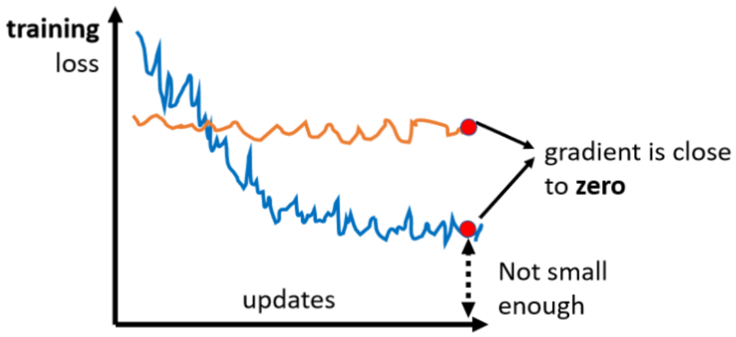


二、李宏毅2021春机器学习课程第2.1节：局部最小值(local minima)与鞍点(saddle point)

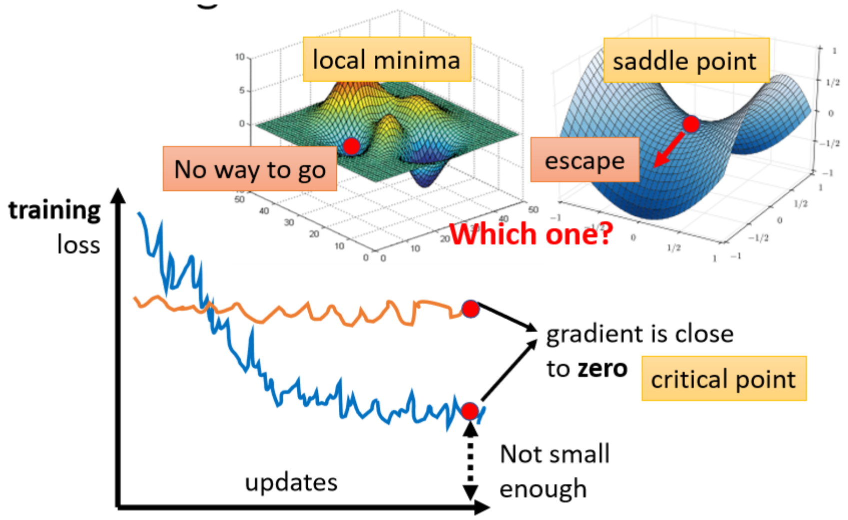
Critical Point

我们常常在做Optimization的时候发现，**随着参数不断update，loss不会再下降**，但是我们对这个loss仍然不满意，有时候我们甚至会发现一开始我们的模型就训练不起来。

过去常见的一个猜想，是因为我们现在走到了一个地方，**这个地方参数对loss的微分为0**，这时gradient descent就没有办法再更新参数了，所以loss当然就不会再下降了。



gradient为0的点，统称为**critical point**，这又包括了**局部最小值（local minima）**，**局部最大值（local maxima）**和**鞍点（saddle point）**这三种情况。



与此同时，当我们的训练受到阻碍的时候，我们其实想要知道我们的训练到底是卡在local minima，还是卡在saddle point。这是因为如果卡在了saddle point，那么我们其实是还有路可以走的，只要找到这条路的方向，就可以将我们的loss进一步减小。

Distinguish local minima and saddle point

要判断我们当前停下的位置是local minima还是saddle point，就需要知道我们当前在error surface中所处的位置，换言之，就需要知道当前位置loss function的形状。loss function本身一般十分复杂，毕竟我们的网络就十分复杂，但我们并不需要知道整个loss function的完整形状，我们**只需要知道我们当前所处区域附近大致的loss function形状**就好了，这就需要用到泰勒级数近似的方法。

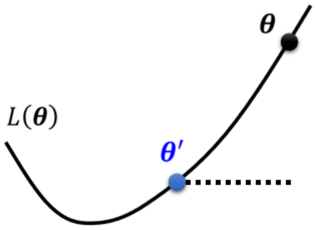
这里需要用到一点**微积分跟线性代数**的知识：

Tayler Series Approximation：泰勒级数近似，在θ'附近的loss function可以近似为下图所示的式子。

Tayler Series Approximation

$L(\boldsymbol{\theta})$ around $\boldsymbol{\theta} = \boldsymbol{\theta}'$ can be approximated below

$$L(\boldsymbol{\theta}) \approx L(\boldsymbol{\theta}') + (\boldsymbol{\theta} - \boldsymbol{\theta}')^T \boldsymbol{g} + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}')^T H (\boldsymbol{\theta} - \boldsymbol{\theta}')$$



在这个式子中:

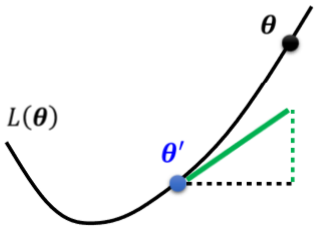
- 第一项是$L(\theta')$，因为当θ'逼近θ的时候，$L(\theta)$与$L(\theta')$其实相差不大。
- 第二项是$(\theta-\theta')^T g$，g**是一个向量，这个**g**就是我们的gradient**，也就是**L的一次微分**，这个**gradient会被用来弥补**θ'**跟**θ**之间的虽然很小但仍然存在的一点点差距**，当然，这个弥补还不够完整，所以我们还需要第三项。

$L(\boldsymbol{\theta})$ around $\boldsymbol{\theta} = \boldsymbol{\theta}'$ can be approximated below

$$L(\boldsymbol{\theta}) \approx L(\boldsymbol{\theta}') + \boxed{(\boldsymbol{\theta} - \boldsymbol{\theta}')^T \boldsymbol{g}} + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}')^T H (\boldsymbol{\theta} - \boldsymbol{\theta}')$$

Gradient $\boldsymbol{g}$ is a *vector*

$$\boldsymbol{g} = \nabla L(\boldsymbol{\theta}') \qquad \boldsymbol{g}_i = \frac{\partial L(\boldsymbol{\theta}')}{\partial \boldsymbol{\theta}_i}$$



- 第三项跟Hessian有关，这个H叫做Hessian，它是一个矩阵，**H里面放的是L的二次微分**，H_{ij}的值就是用

  θ_i对L做微分，再用θ_j对L做微分，总共两次微分的结果。

$L(\boldsymbol{\theta})$ around $\boldsymbol{\theta} = \boldsymbol{\theta}'$ can be approximated below

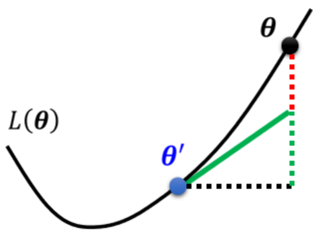
$$L(\boldsymbol{\theta}) \approx L(\boldsymbol{\theta}') + \boxed{(\boldsymbol{\theta} - \boldsymbol{\theta}')^T \boldsymbol{g}} + \boxed{\frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}')^T H (\boldsymbol{\theta} - \boldsymbol{\theta}')}$$

Gradient $\boldsymbol{g}$ is a *vector*

$$\boldsymbol{g} = \nabla L(\boldsymbol{\theta}') \qquad \boldsymbol{g}_i = \frac{\partial L(\boldsymbol{\theta}')}{\partial \boldsymbol{\theta}_i}$$

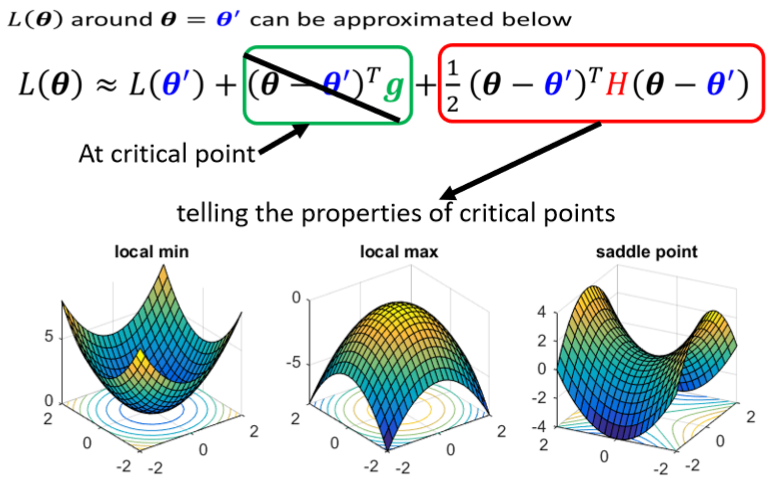
Hessian H is a *matrix*

$$H_{ij} = \frac{\partial^2}{\partial \boldsymbol{\theta}_i \partial \boldsymbol{\theta}_j} L(\boldsymbol{\theta}')$$

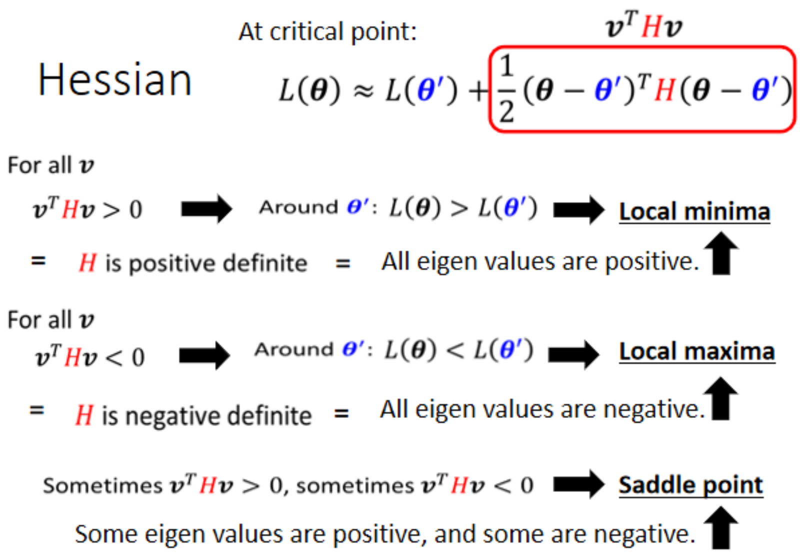


根据Hessian来判断当前的"地貌"：

当我们走到了一个critical point，意味着gradient为0，也就是绿色的这一项完全不见了：



此时的loss function可以被近似为$L(\theta')$，再加上红色的这项，也就是之前提到的二次微分项。我们可以根据第三项，也就是红色的这一项，来判断当前所处的地形。



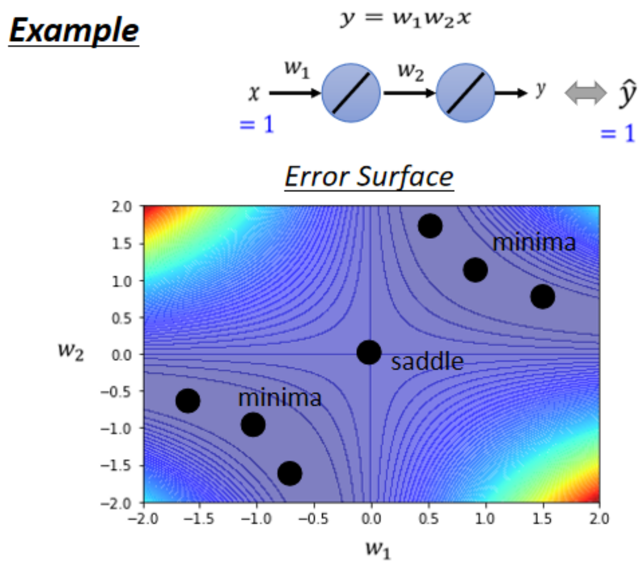
为方便起见，这里**把$(\theta - \theta')$用v这个向量来表示**，那么就有以下三种情况：

1. 对任何可能的v，$v^T Hv$都大于0，这就意味着 $L(\theta) > L(\theta')$。 也就是说在θ'附近，$L(\theta)$都大于$L(\theta')$，**代表$L(\theta')$是附近的一个最低点**，所以它是**local minima**。
2. 对任何可能的v，$v^T Hv$都小于0，这就意味着 $L(\theta) < L(\theta')$。 也就是说在θ'附近，$L(\theta)$都小于$L(\theta')$，**代表$L(\theta')$是附近的一个最高点**，所以它是**local maxima**。
3. 对于不同的v，$v^T Hv$有时小于0，有时大于0，这就说明**在$L(\theta')$附近有的地方高而有的地方低**，所以它是**saddle point**。

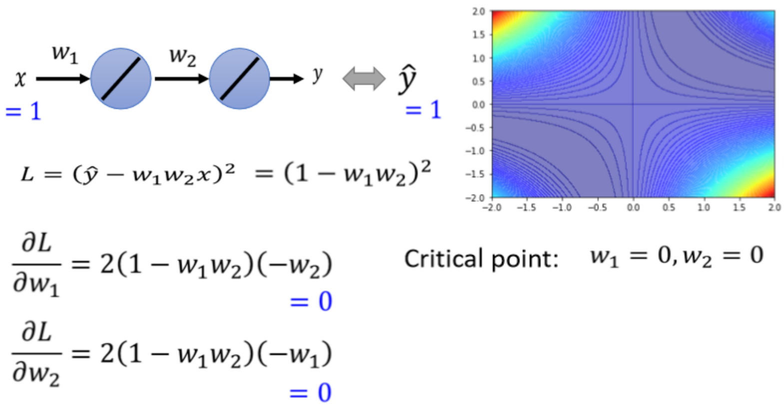
当然把所有可能的v都代入式子中测试是不现实的，所有这里有更简单的方法来确定$v^T Hv$的值与0的关系：如果对所有可能的v而言，$v^T Hv$都大于0，那这种矩阵叫做**正定矩阵(positive definite)**，**正定矩阵所有的特征值(eigen value)都是正的**。所以实际上我们只需要看H的特征值就可以进行判断了：

1. **矩阵H所有特征值都是正的**，则当前处于**local minima**。
2. **矩阵H所有特征值都是负的**，则当前处于**local maxima**。
3. **矩阵H特征值有正有负**，则当前处于**saddle point**。

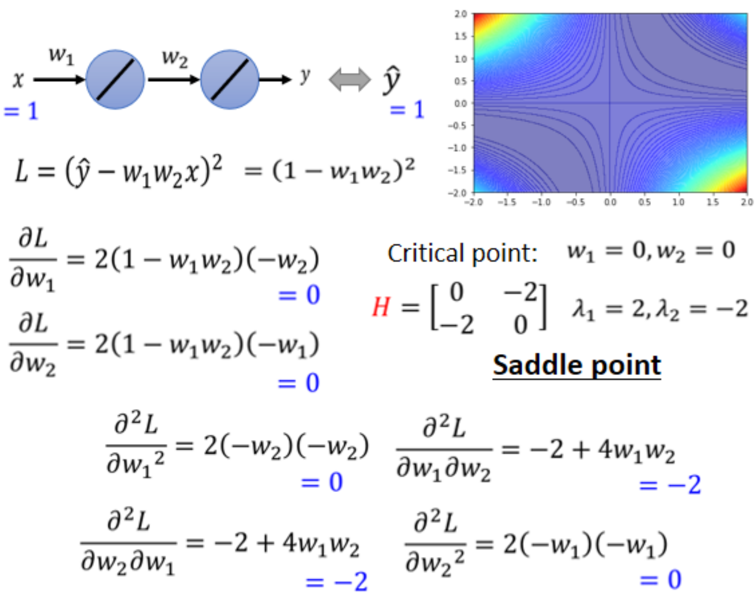
举一个具体的例子来看可能更加直观，假设我们有一个十分简单的网络，它将输入的数据乘上w_1后输入到下一个神经元，在下一个神经元乘上w_2后就直接输出了。假设我们希望输入是1的时候，最终的输出要尽可能逼近1。由于这个网络只有两个参数 w_1和w_2，我们可以很容易画出**error surface**如下：



从图中可以看到，黑点所标识的地方就是一些**critical point**，这里原点处是一个**saddle point**，左下角和右上角分别是一些**local minima。**假设我们要通过计算的方式来判断原点这个critical point的类型，首先可以写出loss function，由于我们希望输入为1时输出逼近1，最终我们得到的L如下图所示，分别求w_1对L的微分以及w_2对L的微分，得到的实际上就是gradient，令这个gradient为0，则得到$w_1 w_2 = 1$或者$w_1 = 0, w_2 = 0$，这也就是我们要找的critical point点的位置。



我们就选定原点来看，接下来需要判断这个critical point究竟是local minima，local maxima，还是saddle point。这时按我们之前所说的，就需要先计算出H矩阵，再算出其特征值，计算的结果如下所示，可以看到H矩阵有**+2和-2一正一负两个特征值**，这就说明我们的原点是一个**saddle point。**



Now escape from saddle point!

At critical point: $L(\boldsymbol{\theta}) \approx L(\boldsymbol{\theta}') + \boxed{\overbrace{\frac{1}{2}(\boldsymbol{\theta}-\boldsymbol{\theta}')^T H(\boldsymbol{\theta}-\boldsymbol{\theta}')}^{\boldsymbol{v}^T H \boldsymbol{v}}}$

Sometimes $\boldsymbol{v}^T H \boldsymbol{v} > 0$, sometimes $\boldsymbol{v}^T H \boldsymbol{v} < 0$ ➡ Saddle point

H may tell us parameter update direction!

$\boldsymbol{u}$ is an eigen vector of H

λ is the eigen value of $\boldsymbol{u}$

$\lambda < 0$

➡ $\underbrace{\boldsymbol{u}^T H \boldsymbol{u}}_{<0} = \boldsymbol{u}^T(\lambda \boldsymbol{u}) = \underbrace{\lambda \|\boldsymbol{u}\|^2}_{<0}$

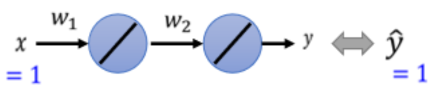
$$L(\boldsymbol{\theta}) \approx L(\boldsymbol{\theta}') + \frac{1}{2}\overbrace{(\boldsymbol{\theta}-\boldsymbol{\theta}')}^{\boldsymbol{u}}{}^T H \overbrace{(\boldsymbol{\theta}-\boldsymbol{\theta}')}^{\boldsymbol{u}} \Rightarrow L(\boldsymbol{\theta}) < L(\boldsymbol{\theta}')$$

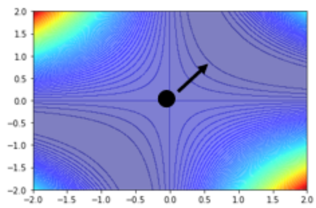
$\boldsymbol{\theta} - \boldsymbol{\theta}' = \boldsymbol{u}$　　$\boldsymbol{\theta} = \boldsymbol{\theta}' + \boldsymbol{u}$　　Decrease L

通过之前的H矩阵，我们还能够找到逃离saddle point的方向。

具体方法：找出**负的特征值**，**再找出它对应的特征向量**，**用这个特征向量去加上θ'就可以找到一个新的点**，**这个点的loss会比原来的loss低。**

还是来看刚刚那个具体的例子，原点是一个critical point，它的Hessian中有一个负的特征值为-2，取这个特征值对应的其中一个特征向量$u\begin{bmatrix}1\\1\end{bmatrix}$，那我们其实只要**顺著这个$u$的方向去更新我们的参数**，**就可以找到一个比saddle point的loss还要更低的点**，也就实现了逃离saddle point的目标。





$L = (\hat{y} - w_1 w_2 x)^2 = (1 - w_1 w_2)^2$

$$\frac{\partial L}{\partial w_1} = 2(1 - w_1 w_2)(-w_2)$$

$$\frac{\partial L}{\partial w_2} = 2(1 - w_1 w_2)(-w_1)$$

Critical point: $w_1 = 0, w_2 = 0$

$H = \begin{bmatrix} 0 & -2 \\ -2 & 0 \end{bmatrix}$　$\lambda_1 = 2, \lambda_2 = -2$

Saddle point

$\lambda_2 = -2$　Has eigenvector　$\boldsymbol{u} = \begin{bmatrix}1\\1\end{bmatrix}$

Update the parameter along the direction of $\boldsymbol{v}_2$

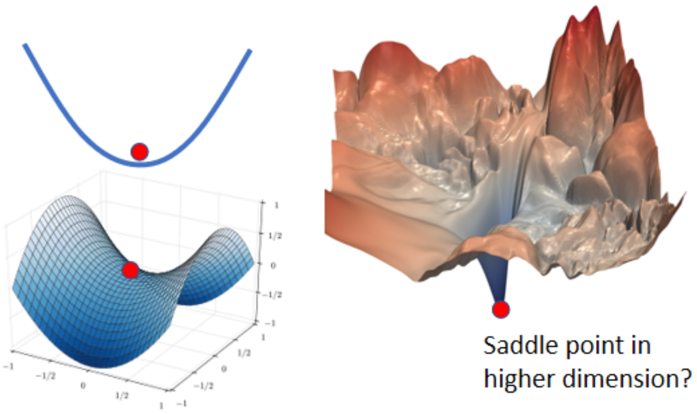
You can escape the saddle point and decrease the loss.

(this method is seldom used in practice)

但这里也有一个需要注意的点，**实际上，我们几乎不会真的把Hessian算出来**，因为这里既有二次微分，又有矩阵特征值计算，整体的运算量非常大。所以在实际过程中一般会使用一些其他的方法来逃离saddle point，具体的方法后面再说，这里我们只需要知道，卡在saddle point并不可怕，我们仍有方法找到逃离它的正确方向。

《三体3：死神永生》中高维碎片带来的启发

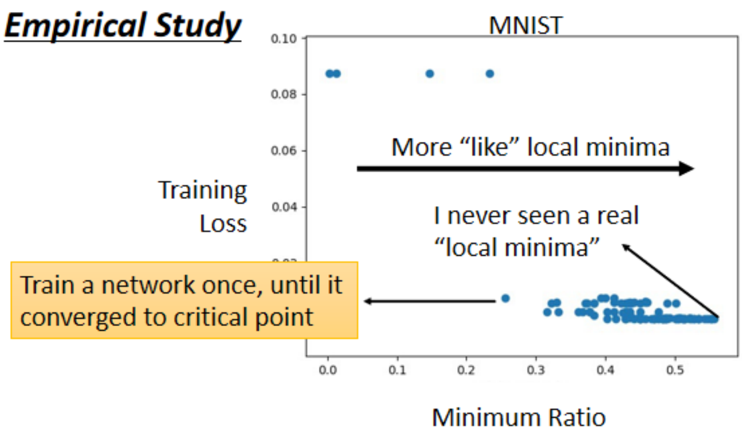
“魔法师”狄奥伦娜可以通过进入四维碎片，从在三维空间中看来完全封闭的石棺中取出圣杯，甚至还放入了一串新鲜的葡萄。这启发了我们，从三维的空间来看，已经没有路可以走了，但**在高维的空间中仍然是有路可以走的**，**error surface会不会也是这样的呢？**



When you have lots of parameters, perhaps local minima is rare?

今天我们在训练一个network的时候，我们的参数往往动辄百万千万以上，所以**我们的error surface其实是在一个非常高的维度中**，既然有这么高的维度，会不会根本就有非常多的路可以走呢？既然有非常多的路可以走，会不会local minima，其实根本上就很少呢？

实际的一些实验也证明了这个猜想，在下面图上，越往右代表我们的critical point越像local minima，**但是它们都没有真的变成local minima**，就算是**在最极端的情况下，我们仍然有负的特征值存在**，也就是说还有办法让loss下降。



$$\text{Minimum ratio} = \frac{\text{Number of } \textbf{Positive} \text{ Eigen values}}{\text{Number of Eigen values}}$$

所以实际上**local minima并没有那么常见，多数的时候我们都只是卡在了一个saddle point。**