# Dense layer -> Convolution layer

纯用MLP，可学习的参数太多了

-平移不变性（translation invariance）：不管检测对象出现在图像中的哪个位置，神经网络的前面几层应该对相同的图像区域具有相似的反应，即为“平移不变性”。

-局部性（locality）：神经网络的前面几层应该只探索输入图像中的局部区域，而不过度在意图像中相隔较远区域的关系，这就是“局部性”原则。最终，可以聚合这些局部特征，以在整个图像级别进行预测。

使用了卷积以后，模型卷积层的参数不再和输入输出的大小相关

WMF\_week2有具体方法和代码

# RNN

把一个变长的序列表示成一个固定长的序列，保留其时序信息

WMF\_week4有具体方法和代码

# Evaluation Metrics

如何衡量一个模型的好坏

## Model Metrics

-在监督学习内，通过最小化损失函数来训练模型

loss是广泛使用的指标

-有很多其他的指标

模型specific：分类的进度、目标检测的mAP

business specific：营收，inference latency推理延迟

-选择模型通过多种指标

## case：Displaying Ads

## Metrics for Classification

-Accuracy：预测等于真实/整个样本数量

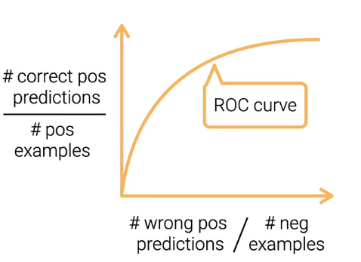
-Precision：预测正类正确的/预测正类的数量

-Recall：预测正类正确的/样本内正类数量

-F1：平衡精度和召回（做调和平均2pr/(p+r)）

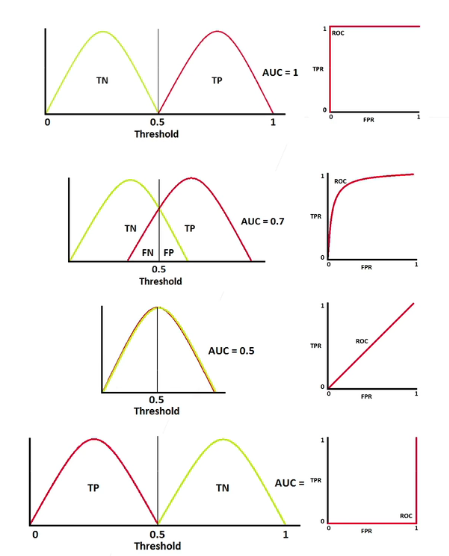
## AUC & ROC of Ads

-AUC是在ROC曲线下面的面积，衡量模型能区分类别的几率

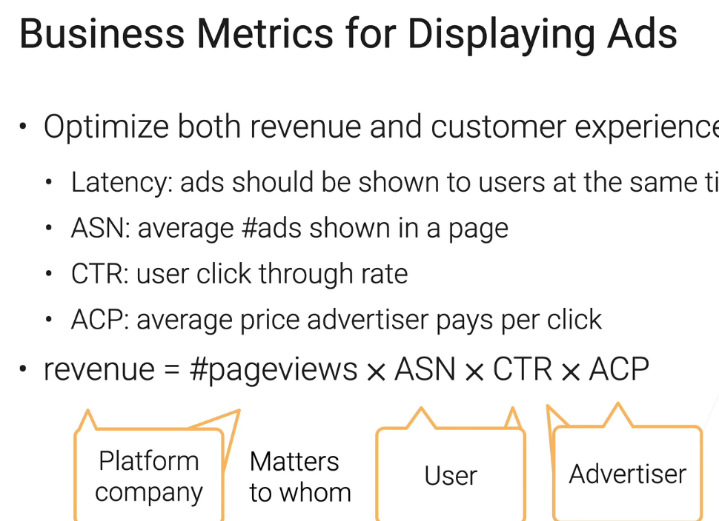


选择一个变量，当y\_hat > 时为正类，否则为负类

曲线上每个点表示不同的



AUC=0.5表示特别糟糕，AUC=1表示非常好

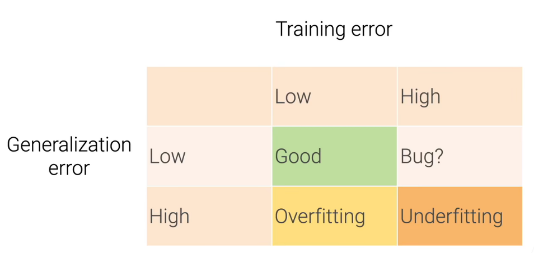


优化收入和消费者体验

# underfiting & overfiting

-training error ：模型在训练数据上的错误率

-generalization error：模型在新的数据上的错误率

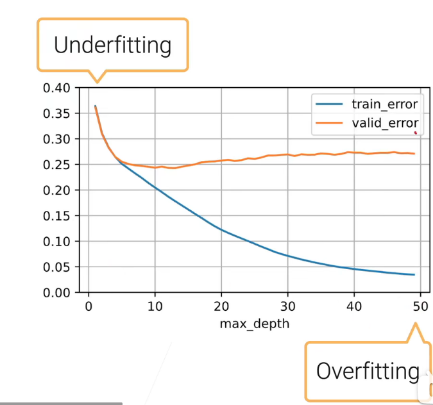


## Model Complexity

-能够拟合各种各样函数的能力

-比较难比较非常不同的两个模型之间的复杂度

-如果模型表达能力能够匹配数据集，那么泛化误差会比较好



maxdepth在10左右的模型较好

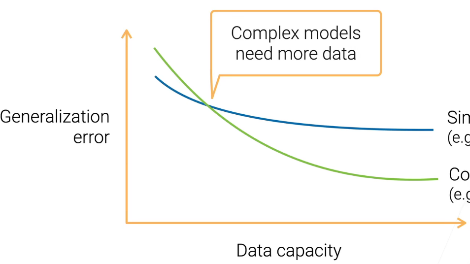
## Data Complexity

-各种因素影响

时序结构、空间结构

多样性：类别

-很难比较两个不同类型数据集的复杂度



随着数据容量的增长

简单模型已经达到了模型表达能力的上限

深层模型的泛化误差还在下降

数据复杂度和模型复杂度是一个要相互匹配的过程

Model Selection：

-选择一个适合你的你的数据的复杂度的模型

-最小化泛化误差

-考虑商业因素（延迟）

-从一个模型族开始，不断选择合适的超参数（将模型复杂度增加去拟合更复杂的数据）

## Summery

-我们关心泛化误差

-模型复杂度：拟合多种函数的能力

-数据复杂度：信息的富裕度

-模型选择：匹配模型和数据复杂度

# Model Validation

-在测试数据集上近似泛化误差（只能使用一次的数据集）

-验证数据集（可以被使用多次）

-训练数据的一部分

-很多时候用test acc，严格意义上是验证数据集

## Hold on Validation

-将数据分为“train”和“vaild”

-通常随机选择n%的样本作为验证集

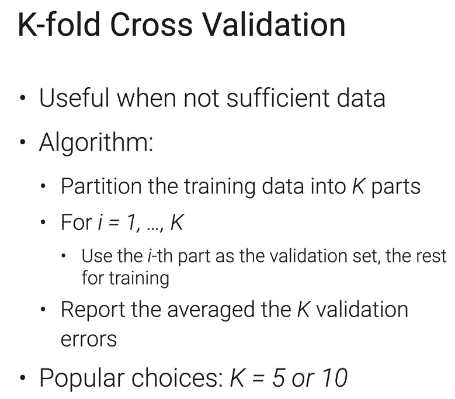
## Split non I.I.D data

-随机划分可能不行

-序列数据：房价、股价

-例子属于同一个组：人脸识别（同一个人脸）

-不平衡的数据（有的类多，有的类少）



K折交叉验证

Common Mistakes

-90%特别好的ML结果是因为Bug

-验证集被污染（有重叠）

-验证集有来源于训练集的数据

-数据来源有冗余

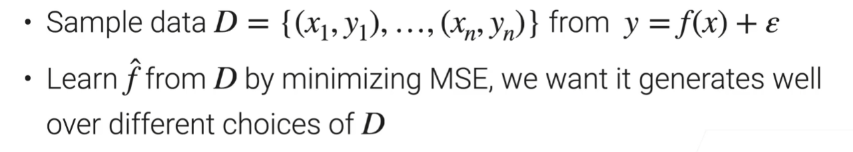
-信息泄露

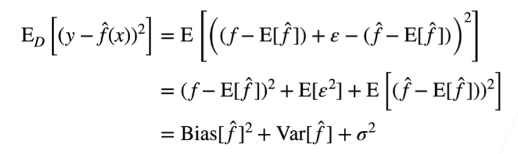
-验证集的数据已经出现过

# Bias & Variance

-在统计学习里面，通常会用偏差和反差来衡量一个模型

## Bias & Variance Decomposition：





模型的泛化误差可以写成这三项

## Reduce Bias & Variance

-Reduce bias

-一个更复杂的模型

-两个方法（boosting、stacking）

-Reduce Variance

-一个更简单的模型

-正则：L2，L1正则

-两个方法（bagging，stacking）

-Reduce

-提升数据质量

-集成学习：boosting、stacking、bagging

-使用多个模型来提升训练的性能

# Bagging

## Bagging -Bootstrap AGGregratING

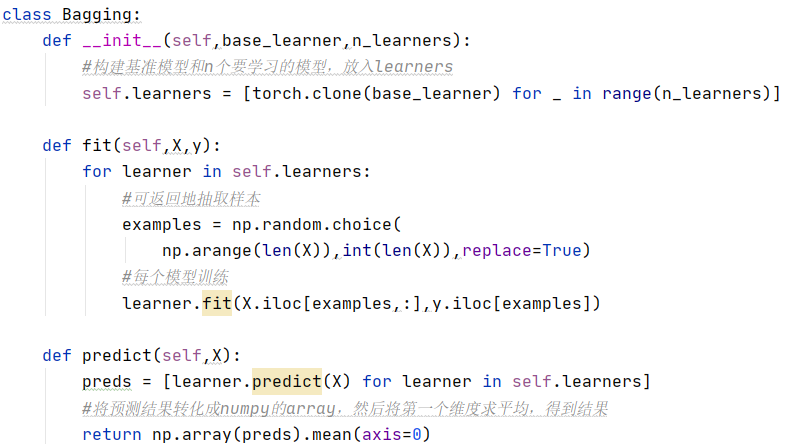
-Bagging 平行训练n个基本模型

-将n个模型的结果收集起来做平均（regression）或majority voting（classification），得到结果

-每个模型在Bootstrap采样的数据上训练

-假设有m个样本，可放回的随机拿m个样本（可能有重复样本）

-大概1-1/e=63%的样本会被拿到，剩下的可以作为验证集

code：

## Unstable Learning

-Bagging减少方差，特别对不稳定的模型

# Boosting：

-Boosting是结合弱的模型将其变成强的模型

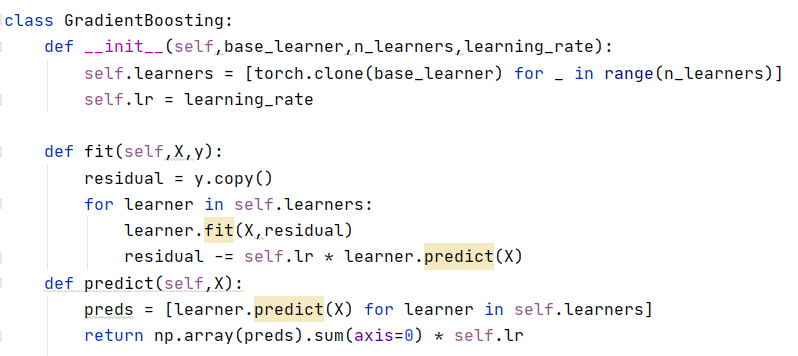
-降低偏差

-顺序地学习n个弱的模型，在第i步：

-训练弱模型hi，评估误差

-根据当前误差重新采样，关注错误样本

-AdaBoost、gradiant boosting（弱的模型是在拟合标号上的残差）

代码：

# Stacking：

刷榜利器

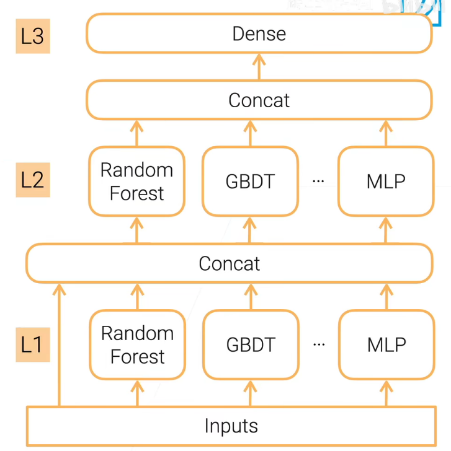
-将多个基准模型放在一起去降低方差

-小模型可以有多个不同的模型类别（在同样的数据上训练不一样的模型）

-将所有模型的输出concat起来，进入一个全连接层输出

-经常在竞赛中使用这个技术

Multi-layer stacking

-多层叠加基准模型去降低偏差

-L2层的模型提取的是L1层的输出

-K折交叉bagging可以减少过拟合

# Model Tuning

## Manual Hyperparameter Tuning

-从一个比较好的基线开始，选一个工具包，这个工具包里面有一些还不错的参数（不见得是最好的，但是会是一个不错的开始点），论文相关就看论文里面的超参数选取

-调一个值，重新训练你的模型

-把前面的步骤重复很多次，得到一些insights关于：

-哪个超参数比较重要

-模型对这些超参数的敏感程度是什么样

-哪个区间比较好

-需要很好的实验管理（做好笔记）

-保存下你的训练日志和超参数，可用作分享、比较和重制

-最简单的方法是把log存到txt文件内，把关键指标保存在excel里

-越来越多的工具可以帮忙：TensorBoard、weight&bias

-重复一个实验是非常难的，它和以下因素有关：

-环境（硬件&库）

-代码

-随机性（随机种子）

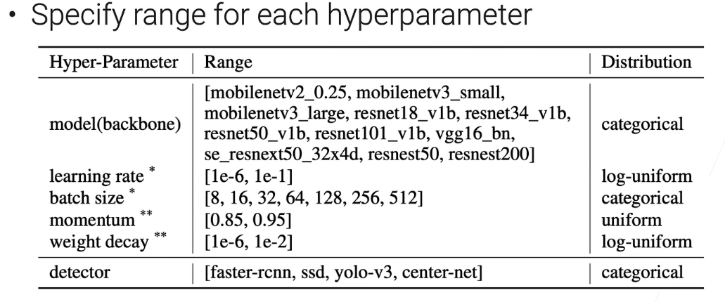
## Automated Hyperparameter Tuning

-在机器学习领域计算开销一直在下降，但是人的成本在增加

-用算法训练1000次，90%的可能打败一个数据科学家

# HPO algorithms

用算法选择超参数



-搜索空间可以以指数级增长

-需要小心地设计空间去提升效率

## HPO algorithms：Black-box or Multi-fidelity

-Black-box：每次训练任务当成一个黑盒（挑一组超参数扔进去，返回精度、误差等衡量指标）

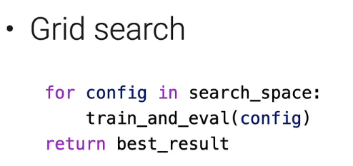
-Multi-fidelity（近似的任务对超参数排序）：修改训练任务加速搜索

-在下采样的数据集上训练

-减少模型大小（例如更少的层、通道）

-早期停止坏的配置

## Two most common HPO strategies

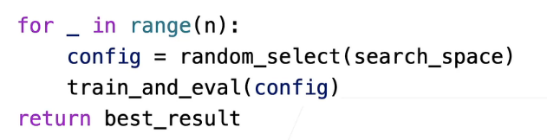


-所有的组合都能评估一遍

-保证找到最佳结果

-维度诅咒（搜索空间特别大的时候，几乎不可能完成）

**Random Search（没有更好的想法之前，第一个试试随机搜索）**



-尝试随机组合

-比Grid search要更有效率

**Bayesian Optimization（BO）**

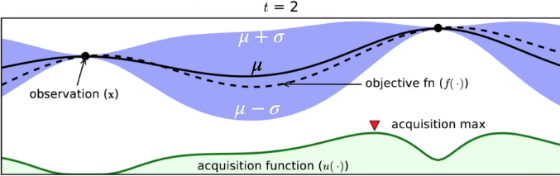
-BO：学习一个从超参数到精度评估指标的函数

-在选下一个超参数的时候，会根据当前的评估

-Surrogate model

-拟合超参数到目标函数的模型

-可以选择回归模型：随机森林、高斯过程等

-Acquisition function

-获取最大意思是最不置信和预测目标可能性比较高

-下一个要采样的超参数是获取函数最大的点

-协调探索和开发

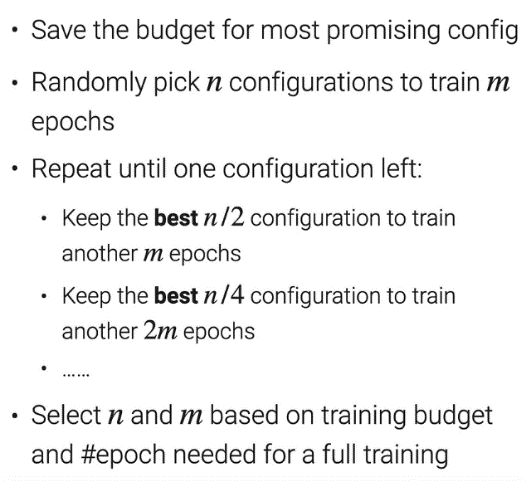
-BO的限制

-在开始阶段和随机搜索差不多

-搜索过程是时序的

-到底从哪里开始比随机搜索要好？

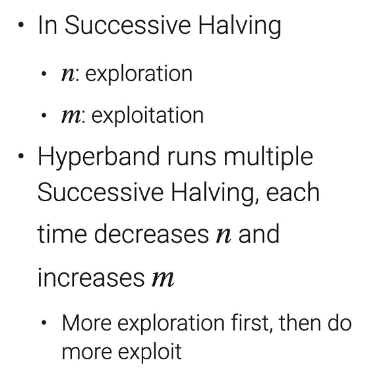
**Successive Halving**



不靠谱的少点资源，靠谱的多点资源，每次淘汰掉一半

缺点：n和m不好取

**Hyberband**



SH的升级，更合理更有效率的选取n，m

## Summery

-Black-box HPO：grid/random search ,Bayesian optimization

-Multi-fidelity HPO：Successive Halving , Hyberband

-在实践中，从随机搜索开始

-关心相关数据集的最好的超参数组合

# Neural Aechitecture Search（NAS）

-一个神经网络有各种各样的超参数

-拓扑结构：resnet，mobilenet，layers

-具体的层：核大小，卷积层的通道数，全连接层或循环神经网络的隐藏输出

-NAS自动设计神经网络

-如何具体化NN的搜索空间

-如何在搜索空间内搜索

-如何衡量搜索的好坏

## The one shot approach

-结合学习模型架构和超参数

-构建和训练一个模型表达多种架构

-评估候选结构

-只关心候选的排名

-使用近似指标：训练一些轮次查看他们的精度

-重新训练最有潜力的架构，得到精度

-Differentiable Architecture Search（将选择模型作为可学习的参数，通过softmax进行子路的选择）

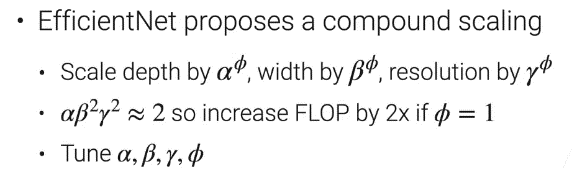
## Scaling CNNs

-一个CNN可以被三种方法调大：

-更深：更多层

-更宽：更多输出通道

-更大的输入：增加输入分辨率



# Deep Network Tuning

-DL是一个编程语言从数据中提取特征

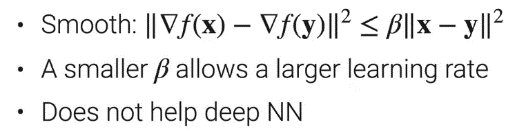
-具体的值是根据数据得到的

-整个语言是可导的

-有非常多的设计模式，从层到整个结构

## Batch Normalization

-对数据进行标准化，使损失函数更加平滑

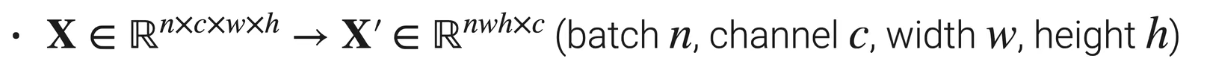
梯度改变不大，可以用更大的学习率

-BN也作用于中间层

-提升平滑性，让训练更加容易

将批量归一化拆解成四步：

-Reshape 输入X成2D（对于2D的输入不改变）



-Normalization 对它的每一列进行标准化

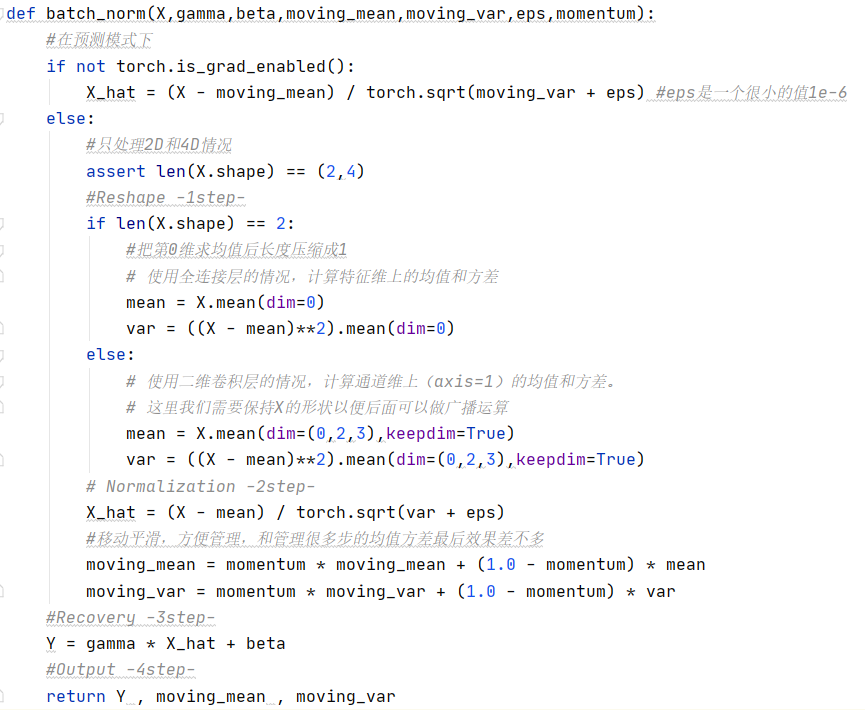


-Recovery 若是允许偏差大一些，可以进行还原操作

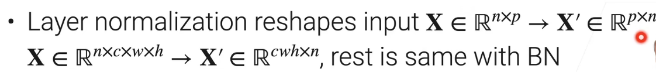
是一个可学习的参数

-Output 先还原成Y，然后得到批量归一化的输出

代码：



## Layer Normalization



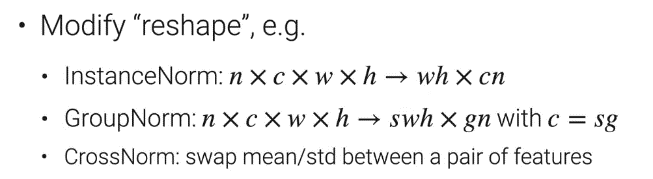
接下来的步骤和BN一样

-标准化每个样本，到当前的时间步骤

-只关心样本内的值

-在Transformer里比较受欢迎

## More Normalization：



-改变“标准化”：白化（标准化加PCA）

-改变“还原”：两个可学习参数换成MLP

-对权重或梯度做标准化

# Transfer Learning

-Motivation

-很难对每一个任务都做很大的数据集

-在深度学习中很受欢迎，DNNs是数据饥饿和训练成本高著称

-Approaches

-特征抽取（用之前的网络替代人工抽特征的步骤）

-在相关的任务上使用的模型，在另一个任务直接用

-微调预训练好的模型

Transferring knowledge：  
-CV存在很多标好的大规模数据集，特别是分类

-我们希望从这些大的数据集上迁移知识到你的任务去

## Pre-trained Models

-一个神经网络可以分成：

-一个特征抽取器（encoder）：将原始的像素转化成在语义空间内线性可分的特征

-一个线性分类器（decoder）：做决定

-预训练模型

-训练在足够大的、普遍的数据集上的神经网络

-特征提取器可能有良好的泛化能力、

## Fine-Tuning techniques

-构建一个新的模型，和预训练的模型架构一样

-初始化新模型：

-用预训练的模型的权重值初始化编码器

-随机初始化解码器

-小区域地开始优化参数

-学习率和训练轮次可以少一点

-限制搜索空间

## Freeze Bottom Layers

-神经网络学习层次化的特征

-底层学习的特征泛化性比较好

-高层的特征向标号的语义特征靠近

-固定住下面的一些层，只调上面的层，是不错的选择