# 提升算法性能可以使用的方法

* 收集更多的数据
* 收集更多的不同的训练集
* 花更多时间用梯度下降法训练算法
* 用Adam代替梯度下降
* 尝试更大的网络
* 尝试更小的网络
* 尝试dropout
* 增加L2正则化
* 尝试不同的网络架构
  + 激活函数
  + 隐藏单元数量
  + ……

# 正交化 --- 前提假设

假定变更系统或算法的一个属性，不会影响系统的其他部分。

对于一个监督型学习系统来说，必须满足以下4个条件并且是正交的：

* Fit training set well on cost function
  + 接近人类水平，否则应该使用更大的网络或更好的优化算法（如adam等）
* Fit development set well on cost function
  + 否则应该加入正则化或使用更大的训练集。
* Fit test set well on cost function
  + 否则应该使用更大的dev集
* Performs well in real world
  + 否则说明测试集与客观现实世界不符，或cost function不对，需要改变

# 尽快建立系统，然后开始迭代

1. Set up development/ test set and metrics
   * Set up a target
2. Build an initial system quickly
   * Train training set quickly: Fit the parameters
   * Development set: Tune the parameters
   * Test set: Assess the performance
3. Use Bias/Variance analysis & Error analysis to prioritize next steps

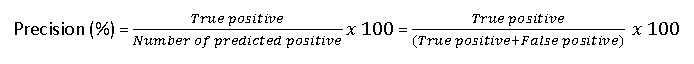
# 建立目标

## 单数字的评价指标

好的dev集和评价指标，可加速迭代过程

### 综合多个评价指标为1个

二分类问题中：

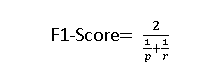


---- 正确的true在被预测为true中的比例



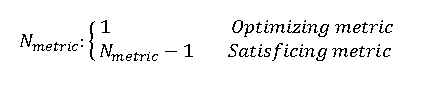
---- 正确的true在实际的true中的比例

在对比算法时，两个分开的指标不方便判断算法的好坏，可以使用F1-score来判断：



F1-score不是唯一的评价指标，可以有其他的很多种。

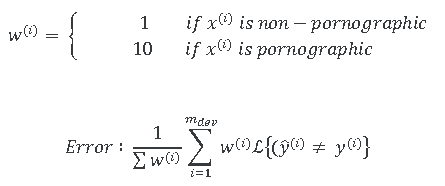
### 满足指标和优化指标



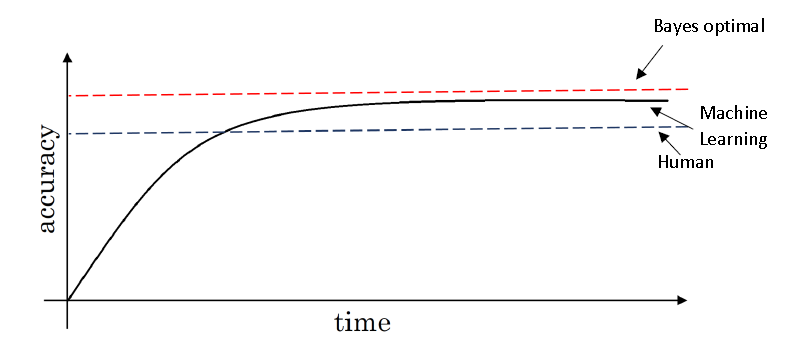
满足=1，不满足=0

## Dev集和测试集

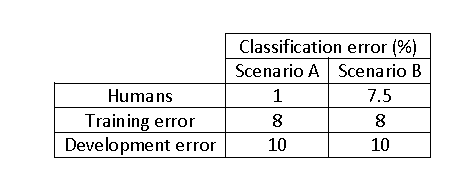
1. dev集和test集必须与未来实际使用的数据是同分布
   1. 采用随机抽取的方式从全部数据中抽样，作为dev和test集
2. Dev集必须足够大，以区别不同算法的好坏，但是不一定要是全部数据的30%、20%
3. Test集必须足够大，以保证足够的置信度，但是不一定要是全部数据的30%、20%
4. 当Dev集和test集存在一些需要排除的错误因素时，基于正交性，可以将目标分两步来考虑：
   1. 单独定义评价函数，用于区分算法的好坏
   2. 单独考虑代价函数，引入权重，放大错误因素的代价。例如：



## 算法可达到的最好情况

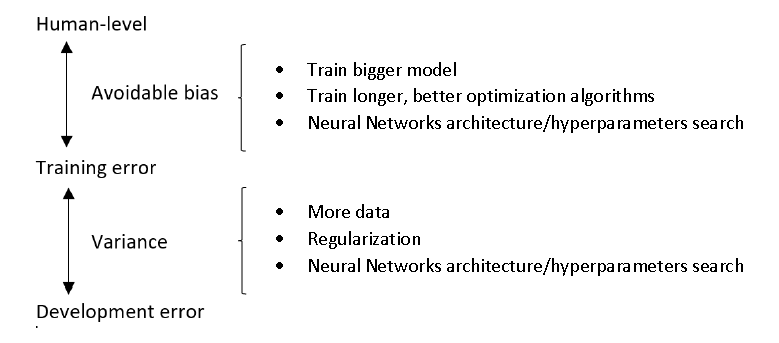


* 算法可到达的最好情况是贝叶斯最优，贝叶斯最优受限与客观（如：噪音、等）
* 部分应用中，机器学习的性能可以超过人类
* 当机器学习算法的性能低于人类水平时，可以考虑用如下工具：
  + 从人类那里获得更多的标识数据
  + 手工分析误差原因：为什么人类能够正确识别？
  + 更好的分析bias和variance
* 当机器学习性能低于人类水平，如果知道了人类水平在哪里，就有可能分辨训练集好坏
  + 人类水平可作为贝叶斯最优的代理
  + 人类最好的水平，作为人类水平
  + Avoidable bias：人类水平与机器学习之间的差距



场景A：avoidable bias = 8-1=7，dev error和training error之间的差2 🡪 training error相对大，重点解决underfit问题 🡪 采用减少bias的技术

场景B：avoidable bias = 8-7.5=0.5，dev error和training error之间的差2 🡪 training error与dev error之间的差异相对大，重点解决overfit问题 🡪 采用减少variance的技术



* 当机器学习算法水平高于人类水平时，缺乏足够信息判断后续应该重点在减少bias还是减少variance

# 分析错误

## 统计不同错误类型

1. 从错误分类的dev集中抽取100个样例，人工分析
2. 记录不同错误类型的数量及简单原因
3. 计算每种错误类型的比例，从比例最高的错误开始解决

## 修正数据中被错误标识的数据

* 深度学习算法对训练集中的**随机**错误是非常鲁棒的，不需要特别修正。系统性错误才需要修正
* 训练集和dev/test集数据的分布，可以轻微的不一致
* 少量了dev集数据标识错误，可以先不处理

# 训练集和dev集/test集失配(mismatch)的处理

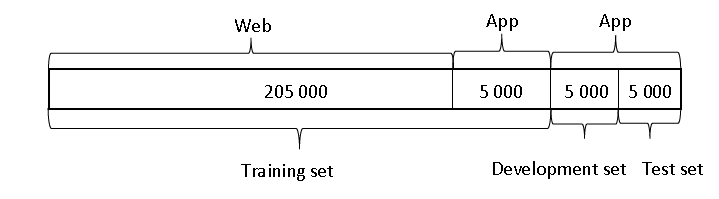
## 什么是失配？

训练集和dev集/test集失配是指：训练集数据和dev集/test集数据分布不一致。

出现失配的常见情况：当不同来源的数据分布不一致，与目标一致的数据量不够大时

## 失配时划分train、dev、test集的原则

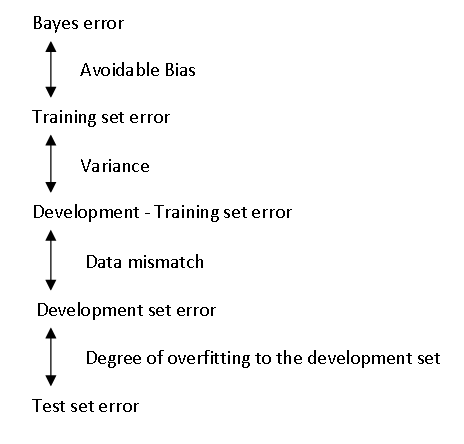
可从目标一致的数据集中抽取dev集/test机数据，保证dev集/test集数据分布是一致的且和目标一致，剩下的数据都放在训练集中。



## 失配带来的bias和variance

为了识别dev集/test集和训练集的差异，是否来自失配，通常在将训练集随机shuffle并分为两部分，一部分进行训练，一部分作为training-dev集。

**Training-dev set**：Same distribution as training set, but not used for training



## 如何解决

没有系统的方法，只有一些指引：

* 人工分析，理解差异原因
* 使得训练集更像dev集/test集：
  + 人工合成数据，注意考虑数据的代表性（是否只是代表了可能数据的一小部分）
  + 收集更多的分布类似dev集/test集的数据

# 多任务学习

## 迁移学习

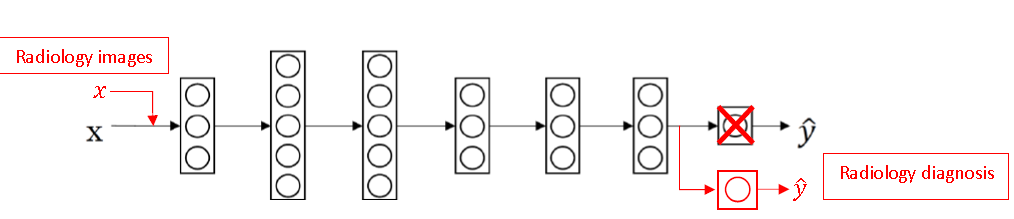
迁移学习：指利用其他应用的网络

什么时候可以迁移学习：任务A 🡪 任务B

* A和B有同样的输入，例如：都是图片、都是语音、等等
* A的数据比B多很多
* A的底层属性可以用于训练B

如何使用：

1. 删除网络的最后一层
2. 删除网络最后输出层的输入权重
3. 为最后一层随机生成一组初始化的权重
4. 输入新的数据（x，y）



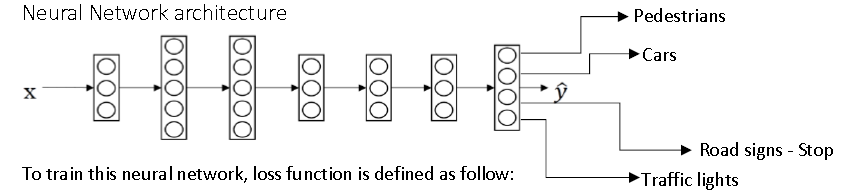
## 多任务学习

多任务学习：指一个神经元网络里同时完成多个任务

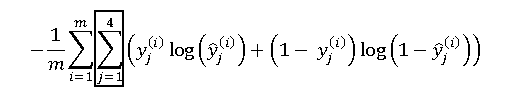
什么时候使用：

1. 一起完成的多个任务，受益于共享多个底层的属性
2. 用于这些任务的数据，通常比较一致
3. 可以训练足够大的网络使得对所有的任务都有好处

例如，在一个视觉网络中，同时训练发现停止标识、前方有障碍物、交通灯等



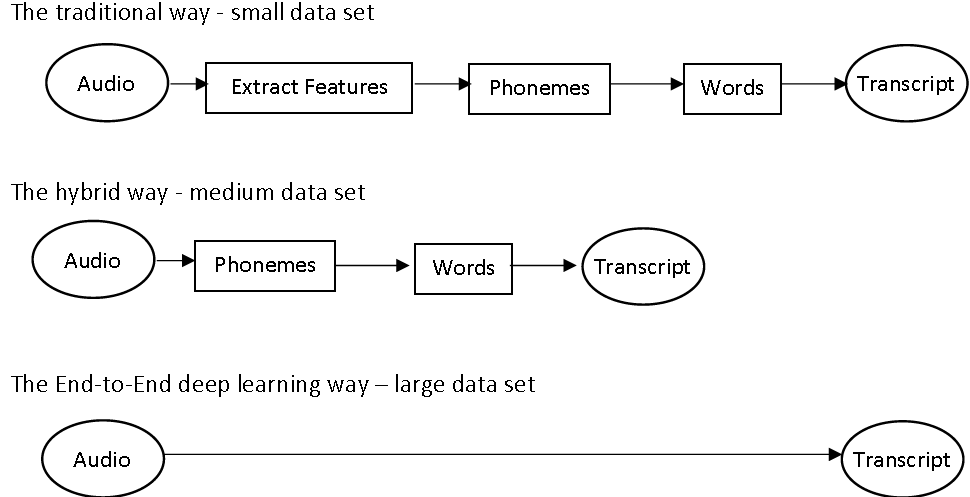
训练这类网络，Loss函数定义改变为：



另外，cost函数也被设计为，即使某些标签为空也不影响cost函数的计算。

# 端到端学习

端到端学习：指将多个学习或处理系统集成到一个里面。例如：语音识别系统



要使用端到端系统，最关键是要有足够多的数据可进行训练。