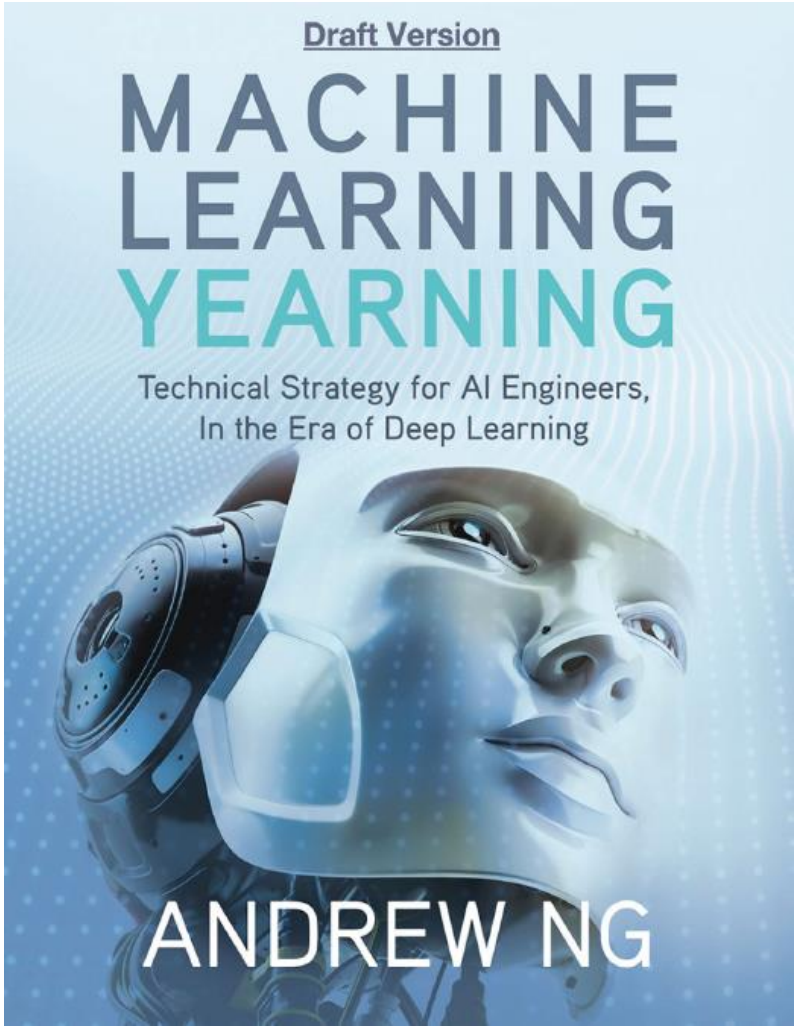
Machine Learning Yearning



- 作者 Andrew Ng

Machine Learning Yearning

- 关于作者

  Andrew Ng（中文名：吴恩达）

  - 华裔美国人
  - 斯坦福大学计算机科学系和电子工程系教授(Adjunct)
  - 人工智能和机器学习领域国际最权威的学者
  - Coursera 和 deeplearning.ai 创始人

Machine Learning Yearning

- 关于 **Andrew Ng的Google Brain项目**

  所开发的人工神经网络通过观看一周YouTube视频，自主学会识别哪些是关于“**猫**“的视频。这个案例为人工智能领域翻开崭新一页。

Building High-level Features Using Large Scale Unsupervised Learning

Quoc V. Le QUOCLE@CS.STANFORD.EDU
Marc'Aurelio Ranzato RANZATO@GOOGLE.COM
Rajat Monga RAJATMONGA@GOOGLE.COM
Matthieu Devin MDEVIN@GOOGLE.COM
Kai Chen KAICHEN@GOOGLE.COM
Greg S. Corrado GCORRADO@GOOGLE.COM
Jeff Dean JEFF@GOOGLE.COM
Andrew Y. Ng ANG@CS.STANFORD.EDU

ICML 2012

Abstract

We consider the problem of building high-level, class-specific feature detectors from only unlabeled data. For example, is it pos-

1. Introduction

The focus of this work is to build *high-level*, class-specific feature detectors from *unlabeled* images. For instance, we would like to understand if it is possible to build a face detector from only unlabeled images. This approach is inspired by the neuroscientific conjecture

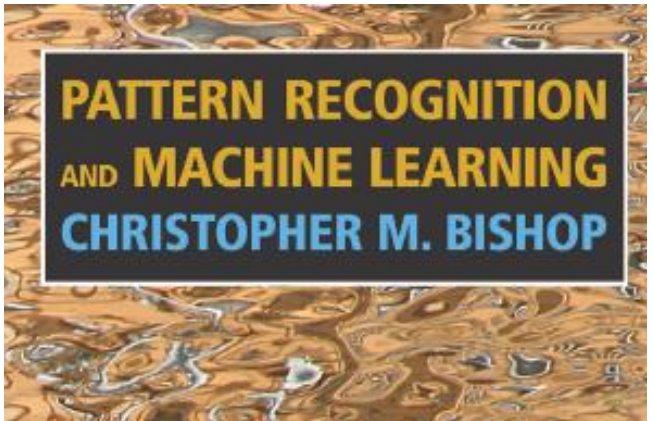
Machine Learning Yearning

- **所需知识储备**

  熟悉监督学习(supervised learning)概念，即使用标注(labeled)的训练样本(x,y) 来学习一个从 x 映射到 y 的函数。

  监督学习算法主要包括线性回归(linear regression)、对数回归(logistic regression，又译作逻辑回归)和神经网络(neural network)。虽然机器学习的形式有许多种，但当前具备实用价值的大部分机器学习算法都来自于监督学习。

  后续将经常提及**神经网络**（也被人们称为“**深度学习**”），但只需对这个概念有基础的了解便可。

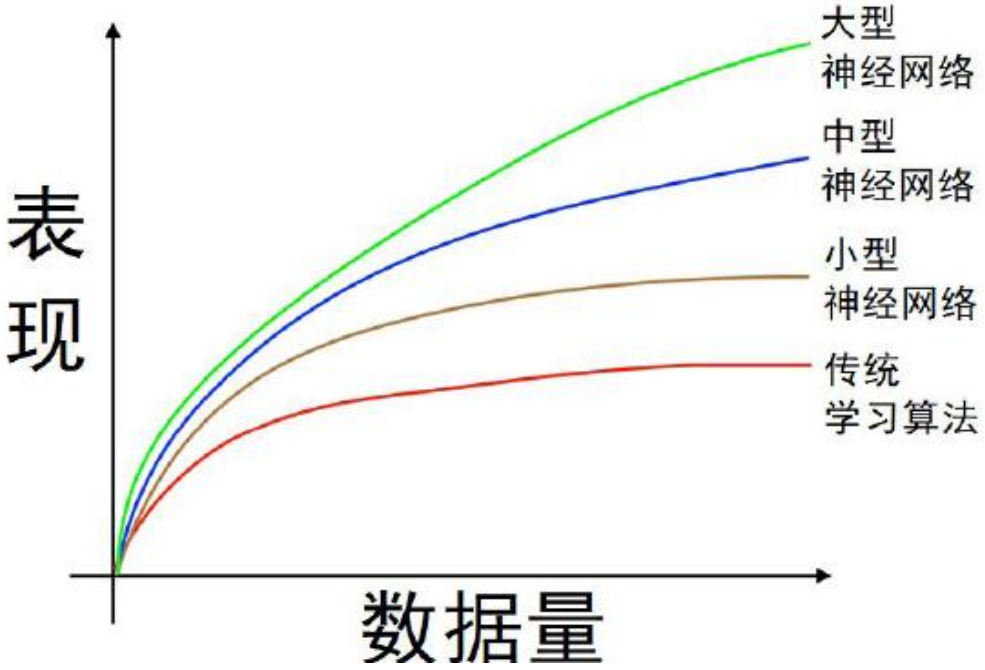


Machine Learning Yearning

- **规模驱动机器学习发展**

  关于**深度学习**（**神经网络**）的一些想法在几十年前就有了，那为什么它们到现在才流行起来了呢？

  - 数据可用性（data availability）：大数据时代……
  - 计算规模（computational scale）：GPU并行计算……



Lectures contents

- 设置开发集与测试集
- 基础误差分析
- 偏差与方差
- 学习曲线

- 与人类表现水平对比
- 在不同的分布上训练与测试
- 端到端深度学习
- 根据组件进行误差分析

Lectures contents

- **设置开发集与测试集**
- 基础误差分析
- 偏差与方差
- 学习曲线

- 与人类表现水平对比
- 在不同的分布上训练与测试
- 端到端深度学习
- 根据组件进行误差分析

设置开发集与测试集

- **案例——建立猫咪图片创业公司**

你正在建立一家创业公司，产品目标是**开发一款App**为猫咪爱好者们提供数不尽的猫咪图片，你打算应用**神经网络（neural network）** 技术来构建一套计算机视觉系统，通过该系统**来识别图片中的猫**。

作为开始，团队已经在不同的网站下载了含有猫的图片（正样本，又译作正例），以及不含猫的图片（负样本，又译作反例），从而得到了一个巨型的数据集。团队将数据集按照 70% / 30% 的比例划分为训练集（training set）和测试集（test set），并且使用这些数据构建出了一个在训练集和测试集上均表现良好的猫咪图片分类器（classifier）。

可当你将这个分类器（classifier）部署到移动应用中时，却发现它的性能相当之差！

设置开发集与测试集

- **案例——建立猫咪图片初创公司**

这究竟是什么原因导致的呢？

你会发现，从网站上下载下来作为训练集的图片与用户上传的图片有较大的区别——用户上传的图片大部分是使用手机拍摄的，此类图片往往分辨率较低，且模糊不清，采光也不够理想。但由于用来进行训练和测试的数据集图片均取自网站，这就导致了算法不能够很好地泛化（generalize）到我们所关心的手机图片的实际分布（actual distribution）情况上。



vs

设置开发集与测试集

- **我们通常认为:**

- 训练集（training set）用于运行学习算法；

- 开发集（development set）用于调整参数、选择特征、以及对学习算法作出其它决定。有时也称为“留出交叉验证集”（**hold-out cross validation set**）；

- 测试集（test set）用于评估算法的性能，但不会据此改变学习算法或参数；

在定义了**开发集**和**测试集**后，你的团队将可以尝试许多的想法，比如调整学习算法的参数来探索哪些参数的使用效果最好。开发集和测试集能够帮助你的团队快速检测算法性能。换而言之，开发集和测试集的使命就是 **引导** 你的团队对机器学习系统做出最重要的改变 。

设置开发集与测试集

- **回顾案例——建立猫咪图片初创公司**

  这究竟是什么原因导致的呢？

正确的做法是：合理地选择开发集和测试集，使之能够代表将来实际数据的情况，并期望算法能够运行良好。

尽可能地选择你最终期望算法能够正确处理的样本作为测试集，而不是随便选择一个你恰好拥有的训练集样本。

设置开发集与测试集

- **Tip：开发集和测试集应该服从同一分布**

根据公司的核心市场分布情况，你将猫咪 app 的图像数据划分为“美国”、“中国”、“印度”和“其它地区”四个区域。在设立开发集和测试集时，可以尝试将“美国”和“印度”的数据归于开发集，而“中国”和“其它地区”的数据归于测试集。也就是说我们可以随机地将其中两个区域的数据分配给开发集，另外两个区域的数据分配给测试集。

这样做对吗？

当然不对！一旦定义好了开发集和测试集，你的团队将专注于提升开发集的性能表现，这就要求开发集能够体现核心任务：使算法在四个地区都表现优异，而不仅仅是其中的两个。

设置开发集与测试集

- **Tip：开发集和测试集应该服从同一分布**
- 举个例子，假设你的团队开发了一套能在开发集上运行性能良好，却在测试集上效果不佳的系统。如果此时开发集和测试集的分布相同，那么你就能清楚地明白问题：算法在开发集上过拟合了（overfit）。解决方案显然是去获取更多开发集数据。
- 但是如果开发集和测试集**服从不同的分布**，解决方案就不那么明确了。此时可能存在以下一种或者多种情况：
  - 算法在开发集上过拟合
  - 测试集比开发集更难进行预测
  - 测试集不一定更难预测，只是它与开发集性质不相同（即不属于同一分布）。这种情况，大量针对开发集性能的改进工作将会是徒劳的。

设置开发集与测试集

- **开发集和测试集应该有多大?**

开发集的规模应该尽可能的大，至少要能够区分出你所尝试的不同算法之间的性能差异。例如如果分类器 A 的准确率为 90.0%，而分类器 B 的准确率为 90.1%，那么使用仅含有 100个样本的开发集将无法检测出这 0.1% 的差异。

一个样本容量仅为 100 的开发集，规模太小了。通常来说，开发集的规模应该在 1,000 到 10,000 个样本数据之间，而当开发集样本容量为 10,000 时，你将很有可能检测到这 0.1% 的性能提升。

设置开发集与测试集

- **开发集和测试集应该有多大？**

那么测试集的大小又该如何确定呢？

它的规模应该大到使你能够对整体系统的性能进行一个高度可信的评估。

设置开发集与测试集

- **使用单值评估指标进行优化**

单值评估指标（single-number evaluation metric）：待你在开发集（或测试集）上运行分类器之后，它将返回单个数值-------分类准确率就是其中的一种！

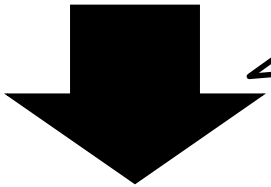
相比之下，**查准率**（Precision，又译作精度）和 **查全率**（Recall，又译作召回率） 的组合并不能作为单值评估指标，因为它给出了两个值来对你的分类器进行评估。多值评估指标提高了在算法之间进行优劣比较的难度，假设你的算法表现如下：

Classifier	Precision	Recall
A	95%	90%
B	98%	85%

设置开发集与测试集

- 使用单值评估指标进行优化

Classifier	Precision	Recall
A	95%	90%
B	98%	85%



F1 分数被定义为Precision和Recall的调和平均数

Classifier	Precision	Recall	F1 score
A	95%	90%	**92.4%**
B	98%	85%	**91.0%**

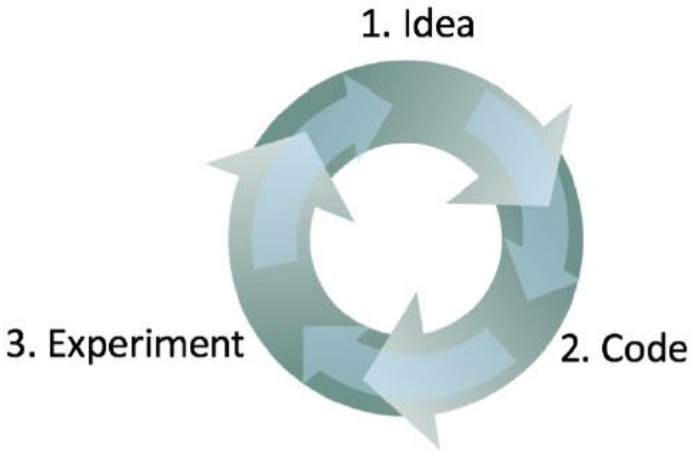
设置开发集与测试集

- **通过开发集和度量指标加速迭代**

  建立一个机器学习系统时，往往的做法是：

  1. 尝试一些关于系统构建的想法（idea）。
  2. 使用代码（code）实现想法。
  3. 根据实验（experiment） 结果判断想法是否行得通。

  （第一个想到的点子一般都行不通！）在此基础上学习总结，从而产生新的想法，并保持这一迭代过程。



设置开发集与测试集

- **何时修改开发集、测试集和指标**

每当开展一个新项目时，一般尽快选好开发集和测试集，因为这可以帮团队制定一个明确的目标，例如要求团队在不到一周（一般不会更长）的时间内给出一个初始的开发集、测试集和指标，提出一个不太完美的方案并迅速执行，这比起花过多的时间去思考要好很多。

如果你渐渐发现初始的开发集、测试集和指标设置与期望目标有一定差距，那就尽快想办法去改进它们。有**三个主要原因**可能导致开发集/评估指标错误地将分类器A排在分类器B前面……

设置开发集与测试集

- **何时修改开发集、测试集和指标**

原因1：你需要处理的实际数据的分布和开发集/测试集数据的分布情况不同

假设你的初始开发集和测试集中主要是成年猫的图片，然而你在 app 上发现用户上传的更多是小猫的图片，这就导致了开发集和测试集的分布与你需要处理数据的实际分布情况不同。在这种情况下，需要更新你的开发集和测试集，使之更具代表性。

设置开发集与测试集

- **何时修改开发集、测试集和指标**

原因2：算法在开发集上过拟合了

在开发集上反复评估某个想法会导致算法在开发集上“过拟合”。当你完成开发后，应该在测试集上评估你的系统。如果你发现算法在开发集上的性能比测试集好得多，则表明你很有可能在开发集上过拟合了。在这种情况下，你需要获取一个新的开发集。

设置开发集与测试集

- 何时修改开发集、测试集和指标

原因3：该指标不是项目应当优化的目标

Classifier	Precision	Recall	F1 score
A	95%	90%	**92.4%**
B	98%	85%	**91.0%**

2 / ((1/Precision) + (1/Recall)).

某些实际应用：如医学图像诊断

赋予P和R不同的权值

$$F_\beta = \frac{(1+\beta^2)\text{Precision} \cdot \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}}$$

设置开发集与测试集

- **何时修改开发集、测试集和指标**

Don't worry!

在项目中改变开发集、测试集或者指标是很常见的。一个初始的开发集、测试集和指标能够帮助团队进行快速迭代，当你发现它们对团队的导向不正确时，不要担心，你只需要对其进行修改并确保团队能够了解接下来的新方向。

设置开发集与测试集

- **小结**

- 被选择作为开发集和测试集的数据分布应当尽可能一致，并且应当与你未来计划获取并对其进行良好处理的数据有着相同的分布。
- 为你的团队选择一个单值评估指标进行优化。当需要考虑多项目标时，不妨将它们整合到一个表达式里（比如对多个误差指标取平均），或者设定满意度指标和优化指标。
- 机器学习是一个高度迭代的过程：在出现最终令人满意的方案之前，你可能要尝试很多想法。拥有开发集、测试集和单值评估指标可以帮助你快速评估一个算法，从而加速迭代进程。
- 当你要探索一个全新的应用时，尽可能在一周内建立你的开发集、测试集和评估指标。

设置开发集与测试集

- **小结**

- 开发集的规模应当大到能够检测出算法精度的细微改变，但也不需要太大；测试集的规模应该大到能够使你能对系统的最终性作出一个充分的估计。
- 当开发集和评估指标对于团队已经不能提供一个正确的导向时，尽快修改它们：
  ① 如果算法在开发集上过拟合，则需要获取更多的开发集数据。
  ② 如果开发集与测试集的数据分布和实际数据分布不同，则需要获取新的开发集和测试集。
  ③ 如果评估指标无法对最重要的任务目标进行度量，则需要修改评估指标。

Lectures contents

- 设置开发集与测试集
- **基础误差分析**
- 偏差与方差
- 学习曲线

- 与人类表现水平对比
- 在不同的分布上训练与测试
- 端到端深度学习
- 根据组件进行误差分析

基础误差分析

- **案例——如何改进我们的算法？**

在使用猫咪 app 时，你注意到它将一些狗的图片误分类为猫了，因为有些狗的样子的确很像猫！

团队中有人建议，加入第三方软件来帮助系统更好地处理狗的样本，但这需要一个月的时间去完成。面对团队成员高涨的热情，你会允许他们这样做吗？

基础误差分析

- **案例——如何改进我们的算法？**

在执行这项计划前，建议你先预估一下该任务能提升多少系统精度。这样你就能够更加理性地判断是否值得花一个月的时间做这件事，还是将这段时间用于其它任务。

具体而言，你可以这样：

1. 收集 100 个开发集中被误分类的样本，即造成系统误差的样本。
2. 人为查看这些样本，并计算其中狗的比例。

查看误分类样本的这一过程称为误差分析!

在上面的例子中，如果只有 5% 误分类的图像是狗，那么无论你在狗的问题上做多少的算法改进，最终都不会消除超过原有的 5% 误差 . 也即是说 5% 是该计划所能起到帮助的“上限”（最大可能值）。

基础误差分析

- **案例——如何改进我们的算法？**

相反，如果你发现 50% 的误分类图像是狗，那就可以自信地说这个项目将效果明显，它可以将相对误差减少 50%。

具体地，如果整个系统当前的精度为 90%（对应误差为 10%），前者最多能将精度提升到 90.5% （对应误差下降到9.5%， 改进了原有 10% 误差其中的5%）。后者最多能将精度提升到 95% （对应误差下降到5%， 改进了原有10% 误差其中的50%）！

基础误差分析

- **所谓“误差分析”**

误差分析（Error Analysis） 指的是检查被算法误分类的开发集样本的过程，以便帮助你找到造成这些误差的原因。这将协助你确定各个项目的优先级（就像上面的例子所提到的那样）并且获得探索新方向的灵感，我们将会在之后再讨论这些内容。

基础误差分析

- **在误差分析时并行评估多个想法**

对于改进猫检测器，你的团队有一些想法

- 修正算法将狗误分类为猫的问题。
- 修正算法将大型猫科动物（比如狮子、黑豹等等,下面用大猫代指）误分类为家猫的问题。
- 改善系统在模糊图像上的表现。

上述的想法都可以并行进行评估，即创建表格如下：

图像	狗	大猫	模糊	备注
1	✔			罕见美国比特犬
2			✔	
3		✔	✔	狮子；雨天在动物园拍摄的图片
4		✔		树木后的美洲豹
占全体比例	25%	50%	50%	

基础误差分析

- 在误差分析时并行评估多个想法

图像	狗	大猫	模糊	备注
1	✔			罕见美国比特犬
2			✔	
3		✔	✔	狮子；雨天在动物园拍摄的图片
4		✔		树木后的美洲豹
…	…	…	…	…
占全体比例	8%	43%	61%	

所以，你应该可以判断出，对于解决狗的误分类问题项目最多可以改进 8% 的误差，处理大猫和模糊类则可以改进更多。因此你将更有可能挑选后两者之一进行处理。如果你的团队有足够的人力并行处理多个方向，则可以要求一部分成员处理大猫类别，另一部分成员处理模糊类别。

基础误差分析

- **清洗误标注的开发集和测试集样本**

在进行误差分析时，你可能会注意到一些开发集的样本被误标注（mislabeled ）了……

此处的“误标注”指的是图像在使用算法处理前，已经被负责标注的人员进行了错误的标注，也就是说，某个样本 (x,y) 的分类标签（label）y 的值并不正确。如果你不确定这些被误标注的图片是否起着关键作用，可以添加一个类别来跟踪记录误标注样本的比例

图像	狗	大猫	模糊	误标注	备注
…					
98				✔	标注者忽略了背景中的猫
99		✔			
100				✔	猫的画像；非真猫
占全体比例	8%	43%	61%	6%	

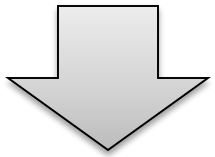
基础误差分析

- **清洗误标注的开发集和测试集样本**

  那么这个时候，需要修正开发集中的标签吗？

  举个例子，假设你的分类器表现如下：

  - 开发集整体精度……90% （10% 整体误差）
  - 误标注样本造成的误差0.6% （6% 开发集误差）
  - 其它原因造成的误差…9.4% （94% 开发集误差）

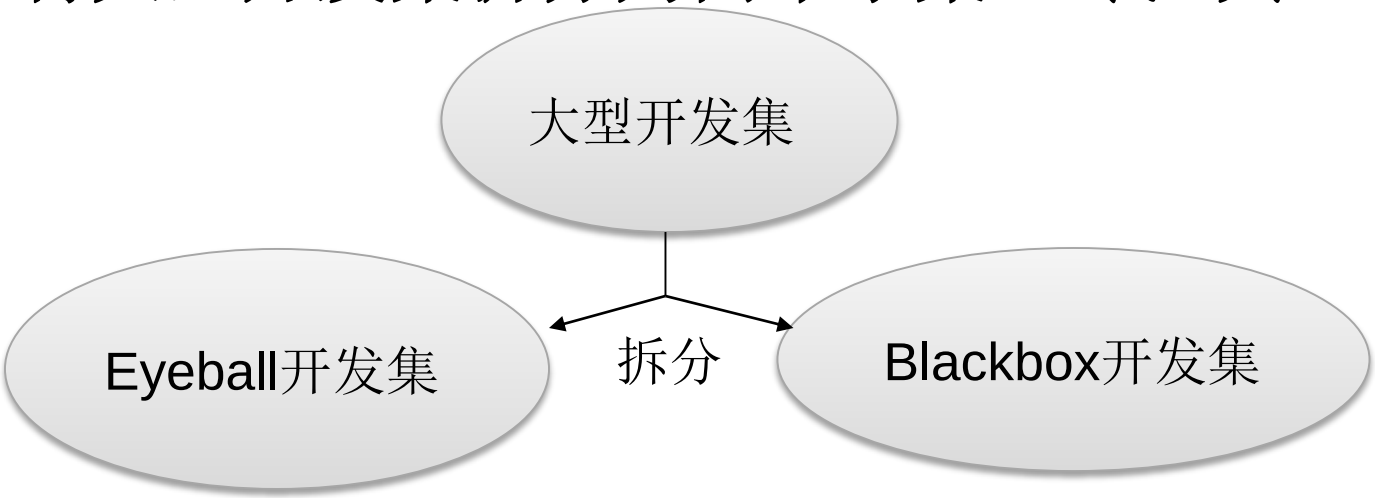


  假设通过不断地改进猫分类器，达到以下性能：

  - 开发集整体精度…… 98.0% （2.0% 整体误差）
  - 误标注样本造成的误差0.6% （30% 开发集误差）
  - 其它原因造成的误差…1.4% （70% 开发集误差）

基础误差分析

- **将大型开发集拆分为两个子集，专注其一**



为什么我们要把开发集明确分为 Eyeball 开发集和 Blackbox 开发集呢？因为当你在 Eyeball开发集中建立了对样本的直观认识之后，可使得Eyeball开发集更快地过拟合。 将开发集明确地分为 Eyeball 和 Blackbox 开发两个子集将很有帮助，它使你了解在人为的误差分析过程中 Eyeball 开发集何时开始发生过拟合。

基础误差分析

- **Eyeball集和Blackbox集应当多大？**

Eyeball 开发集应该大到能够让你对算法的主要错误类别有所察觉。如果你正在处理一项实际情况中人类表现良好的任务（例如识别图像中的猫），下面是一些粗略的指导方案：

- 如果在 Eyeball 开发集上只出现 10 次误判，这个开发集就有点小了。若只有 10 个错误样本，很难准确估计不同错误类别的影响。但如果数据非常少且无法提供更多 Eyeball开发集样本时，聊胜于无，这将有助于确立项目的优先级。
- 如果分类器在 Eyeball 开发集样本上出现了约 20 次误判，你将可以开始大致了解主要的误差来源。
- 如果有约 50 个错误样本，你将能够比较好地了解主要的误差来源。
- 如果有约 100 个错误样本，你将会非常清楚主要的误差来源。我见过有人手动分析更多的错误样本——有时候多达500个。只要你有足够多的数据，这将是无害的!

基础误差分析

- **Eyeball集和Blackbox集应当多大？**

Blackbox 开发集该有多大呢？我们先前提到，开发集有约 1000-10000 个样本是正常的 。完善一下该陈述，一个有 1000-10000 个样本的 Blackbox 开发集通常会为你提供足够的数据去调超参和选择模型，即使数据再多一些也无妨。而含有 100 个样本的 Blackbox 开发集虽然比较小，但仍然是有用的。

如果开发集较小，那么你可能没有足够的数据将其分成足够大的 Eyeball 开发集和 Blackbox开发集来满足目的。相反，你的整个开发集可能需要用作 Eyeball 开发集——即，你将手动检查所有的开发集数据。

基础误差分析

- **小结**

- 使用误差分析法去帮助你识别出最有前景的方向，并据此不断迭代改进你的算法。
- 通过手动检查约 100 个被算法错误分类的开发集样本来执行误差分析，并计算主要的错误类别。使用这些信息来确定优先修正哪种类型的错误。
- 考虑将开发集分为人为检查的 Eyeball 开发集和非人为检查的 Blackbox 开发集。如果在Eyeball 开发集上的性能比在 Blackbox 开发集上好很多，说明你已过拟合 Eyeball 开发集，下一步应该考虑为其获取更多数据。
- Eyeball 开发集应该足够大，以便于算法有足够多的错误分类样本供你分析。而对大多数应用来说，含有1000-10000个样本的 Blackbox 开发集已足够。
- 如果你的开发集不够大，无法按照这种方式进行拆分，那么就使用 Eyeball 开发集来执行人工误差分析、模型选择和调超参。

Lectures contents

- 设置开发集与测试集
- 基础误差分析
- **偏差与方差**
- 学习曲线

- 与人类表现水平对比
- 在不同的分布上训练与测试
- 端到端深度学习
- 根据组件进行误差分析

偏差与方差

- **是不是数据越多效果越好？**

拥有更多的数据是无害的，然而它并不总是如我们期望的那样有帮助。有时获取更多的数据可能是在浪费时间。那么，何时才应该添加数据呢？

机器学习中有两个主要的误差来源：**偏差（Bias）**和**方差（Variance）**。理解它们将协助你决定是否该添加数据，并依此合理安排时间去执行其它的策略来提升性能。

偏差与方差

- **所谓偏差与方差**

偏差（Bias）：是指算法在**训练集**上的错误率。在本例中，它是 15%。我们非正式地将它作为算法的偏差（Bias）。

方差（Variance）：是指算法在**开发集（或测试集）**上的表现比训练集上差多少。我们非正式地将它作为算法的方差（Variance）。

假设你希望构建一个误差为 5% 的猫识别器，而目前的训练集错误率为 15%，开发集错误率为 16%。

偏差为15%

方差为16%-15%=1%

偏差与方差

- **所谓偏差与方差**

一些学习算法的优化能解决误差来源的第一个部分——偏差，并且提高算法在训练集上的性能。

而另一些优化能解决第二个部分——方差，并帮助算法从训练集到开发/测试集上更好地泛化。

为了选择最有成效的改变方向，了解二者哪一方更需解决是很有用的。

偏差与方差

- **偏差和方差举例**

思考一下，我们的“猫分类”任务目标：一个“理想的”分类器（比如人类）在这个任务中能够取得近乎完美的表现。

假设你的算法表现如下：

训练错误率 = 1%

开发错误率 = 11%

这其中存在什么问题呢？根据前一章的定义，我们估计偏差为 1%，方差为 10%（=11%-1%）。

因此，它有一个很高的方差（ high variance ）。虽然分类器的训练误差非常低，但是并没有成功泛化到开发集上。这也被叫做过拟合（ overfitting ）。

偏差与方差

- **偏差和方差举例**

假设你的算法表现如下：

训练错误率 = 15%

开发错误率 = 16%

我们估计偏差为 15%，方差为 1%。该分类器的错误率为 15%，没有很好地拟合训练集，但它在开发集上的误差不比在训练集上的误差高多少。因此，该分类器具有较高的偏差（ high bias ），而方差较低。我们称该算法是欠拟合（ underfitting ）的。

偏差与方差

- **偏差和方差举例**

又如你的算法表现如下：

训练错误率 = 15%

开发错误率 = 30%

我们估计偏差为 15%，方差为 15%。该分类器有高偏差和高方差（ high bias and high variance ）：它在训练集上表现得很差，因此有较高的偏差，而它在开发集上表现更差，因此方差同样较高。由于该分类器同时过拟合和欠拟合，过拟合/欠拟合术语很难准确应用于此。

偏差与方差

- **偏差和方差举例**

最后假设你的算法表现如下：

训练错误率 = 0.5%

开发错误率 = 1%

该分类器效果很好，它具有低偏差和低方差。恭喜获得这么好的表现！

偏差与方差

- **与最优错误率比较**

在我们的“猫咪识别”案例中，“理想”错误率——即一个“最优”分类器应该达到的值——接近 0%。在几乎所有情况下，人类总是可以识别出图片中的猫。因此，我们希望机器也能够有这样优秀的表现。

----因此可定义“最优错误率”为0%

若换作其他问题，难度则更大：假设你正在构建一个语音识别系统，并发现 14% 的音频片段背景噪声太多，或者十分难以理解，导致即使是人类也无法识别出所说的内容。在这种情况下，即使是“最优”的语音识别系统也可能约有 14% 的误差。

----因此可定义“最优错误率”为14%

偏差与方差

- **与最优错误率比较**

对于最佳错误率远超零的状况，有一个对算法误差（在开发集上）更详细的分解如下：

误差=最优错误率+可避免偏差 +方差

最优错误率（也可称为“不可避免偏差”）：例如14%，假设我们决定，即使是世界上最好的语音系统，仍会有 14% 的误差。我们可以认为14%是学习算法的偏差“不可避免”的部分。

可避免偏差：即训练错误率（例如15%）和最优误差率之间的差值（=1%）。

方差：即开发错误和训练错误之间的差值（同之前）。

偏差与方差

- **处理偏差和方差**

简单说来，如果具有较高的可避免偏差，那么加大模型的规模（例如通过添加层/神经元数量来增加神经网络的大小）。

如果具有较高的方差，说明算法泛化存在问题，那么增加训练集的数据量。

偏差与方差

- **减少可避免偏差的常用技术(重要)**

加大模型规模（例如神经元/层的数量）：这项技术能够使算法更好地拟合训练集，从而减少偏差。当你发现这样做会增大方差时，通过加入正则化可以抵消方差的增加。

根据误差分析结果修改输入特征：假设误差分析结果鼓励你增加额外的特征，从而帮助算法消除某个特定类别的误差。这些新的特征对处理偏差和方差都有所帮助。理论上，添加更多的特征将增大方差；当这种情况发生时，你可以加入正则化来抵消方差的增加。

减少或者去除正则化（L2 正则化，L1 正则化，dropout）：这将减少可避免偏差，但会增大方差。

修改模型架构（比如神经网络架构）使之更适用于你的问题：这将同时影响偏差和方差。

偏差与方差

- **减少方差的常用技术（解决过拟合手段）**

添加更多的训练数据：这是最简单可靠的处理方差的策略，只要有大量的数据和对应的计算能力来处理他们。

加入正则化（L2 正则化，L1 正则化，dropout）：这项技术可以降低方差，但却增大了偏差。

加入提前终止（例如根据开发集误差提前终止梯度下降）：这项技术可以降低方差但却增大了偏差。提前终止（Early stopping）有点像正则化理论，一些学者认为它是正则化技术之一。

通过特征选择减少输入特征的数量和种类：这种技术或许有助于解决方差问题，但也可能增加偏差。当你的训练集很小的时候，特征选择是非常有用的。

减小模型规模（比如神经元/层的数量）：谨慎使用。这种技术可以减少方差，同时可能增加偏差。

偏差与方差

- 小结
  - 什么是偏差？
  - 什么是不可避免偏差？
  - 什么是可避免偏差？
  - 减少可避免偏差的常用技术（4种）？
  - 减少方差（解决过拟合）的常用技术（5种）？

Lectures contents

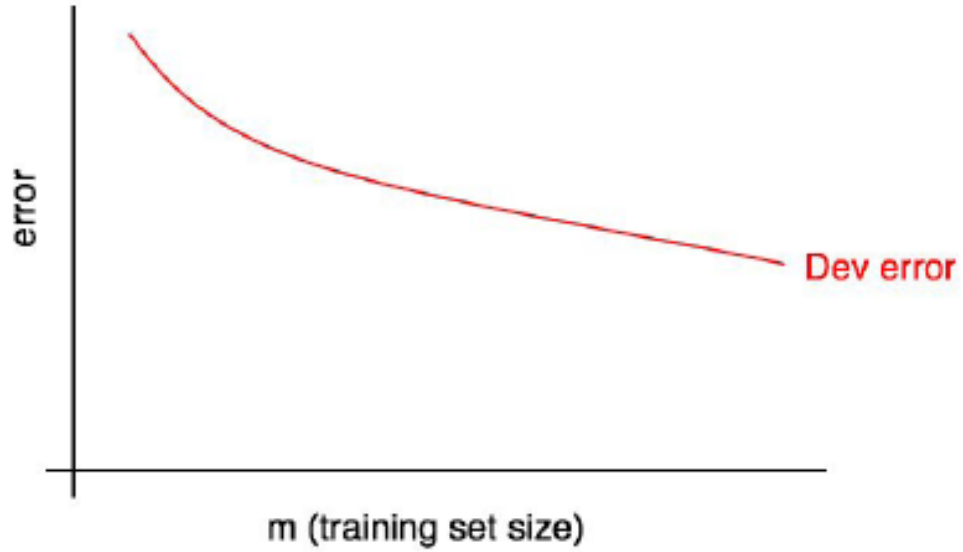
- 设置开发集与测试集
- 基础误差分析
- 偏差与方差
- **学习曲线**

- 与人类表现水平对比
- 在不同的分布上训练与测试
- 端到端深度学习
- 根据组件进行误差分析

学习曲线

- **一项更有帮助的技术：绘制学习曲线**

学习曲线可以将开发集的误差与训练集样本的数量进行关联比较。想要绘制出它，你需要设置不同大小的训练集运行算法。假设有 1000 个样本，你可以选择在规模为 100、200、300 ...1000 的样本集中分别运行算法，接着便能得到开发集误差随训练集大小变化的曲线。下图是一个例子：

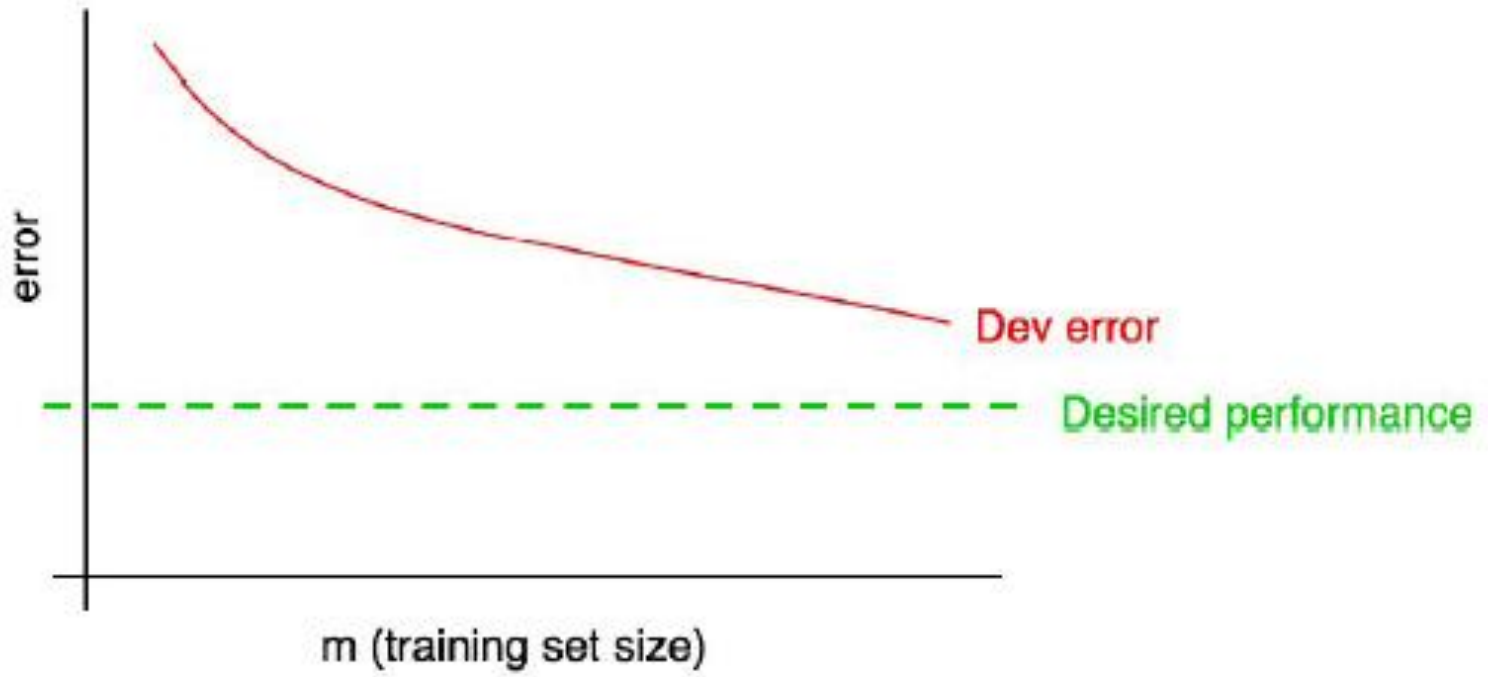


随着训练集大小的增加，开发集误差应该降低！

学习曲线

- **一项更有帮助的技术：绘制学习曲线**

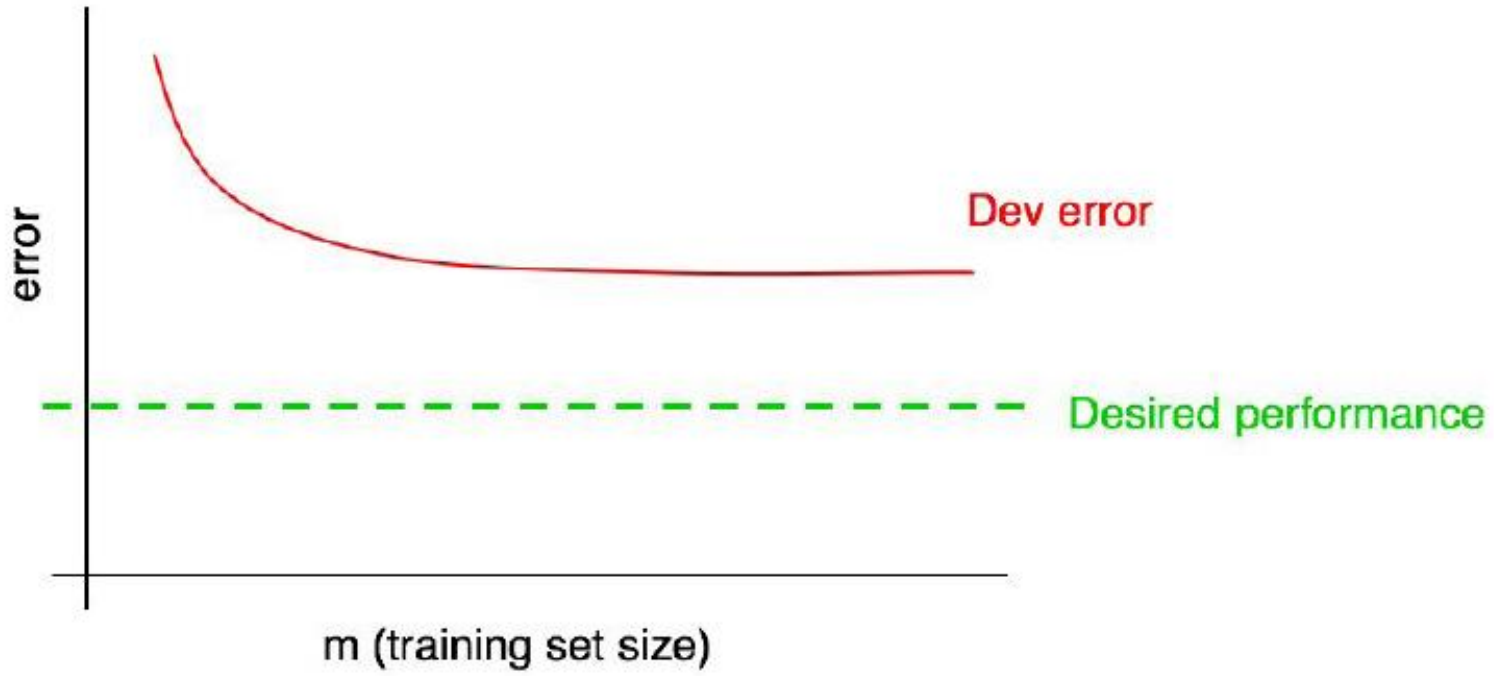
我们通常会设置一些“期望错误率”，并希望学习算法最终能够达到该值。例如：



学习曲线

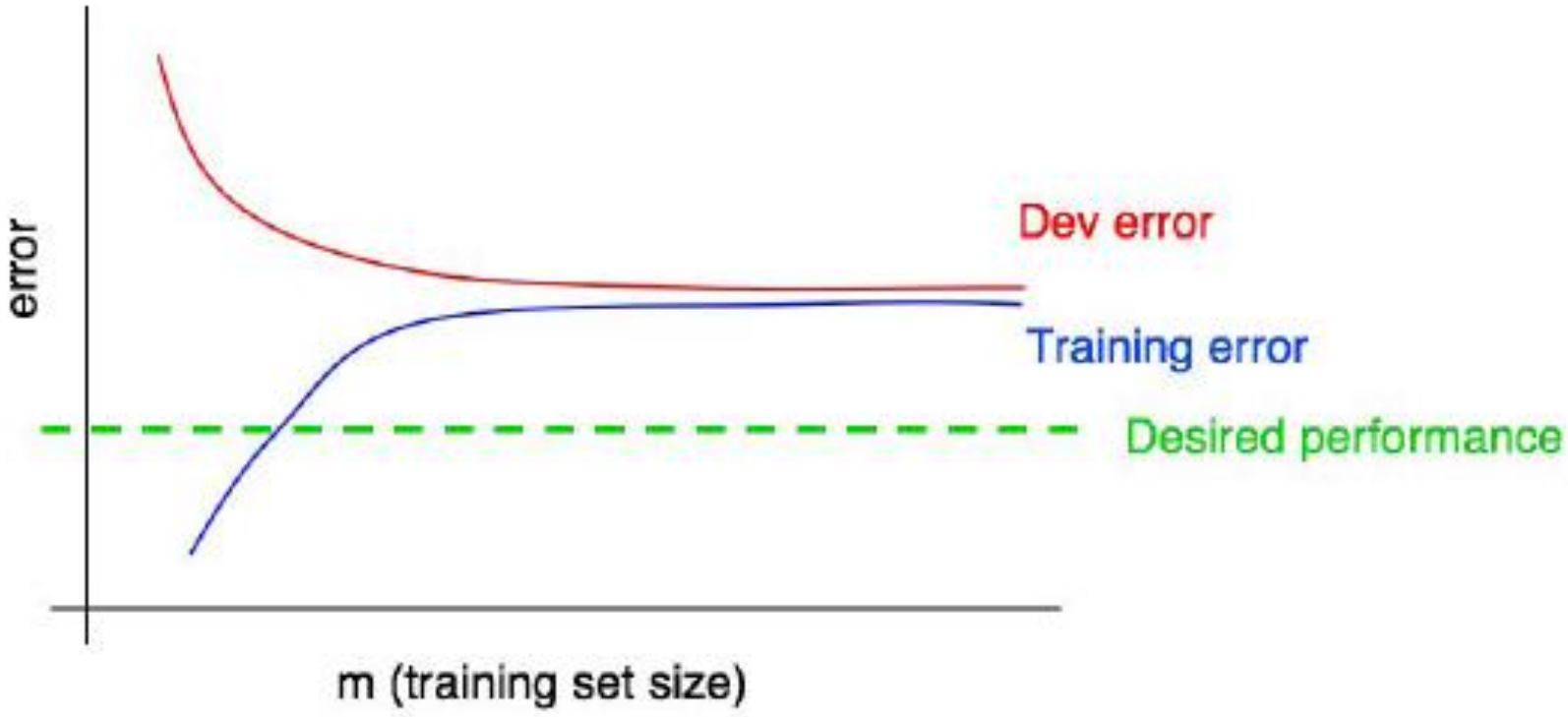
- **一项更有帮助的技术：绘制学习曲线**

又例如：



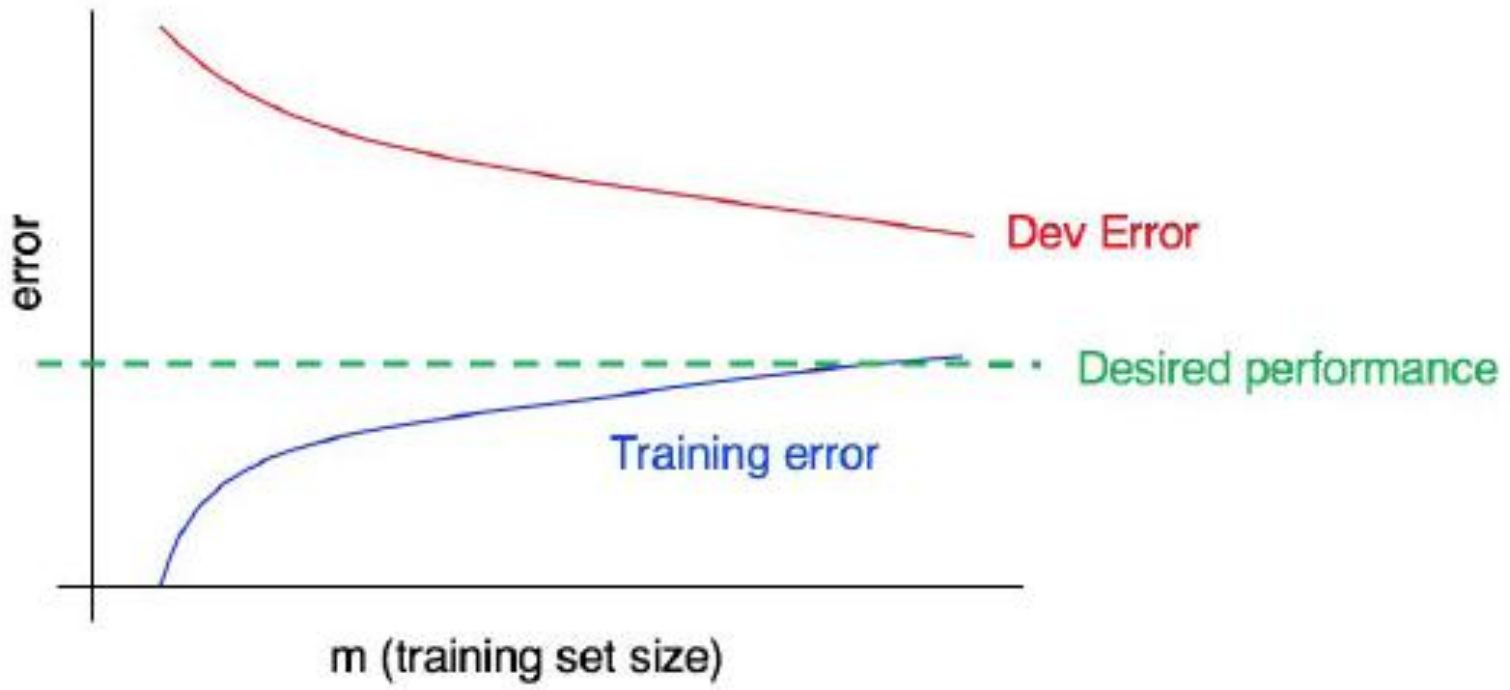
学习曲线

- 解读学习曲线：高偏差



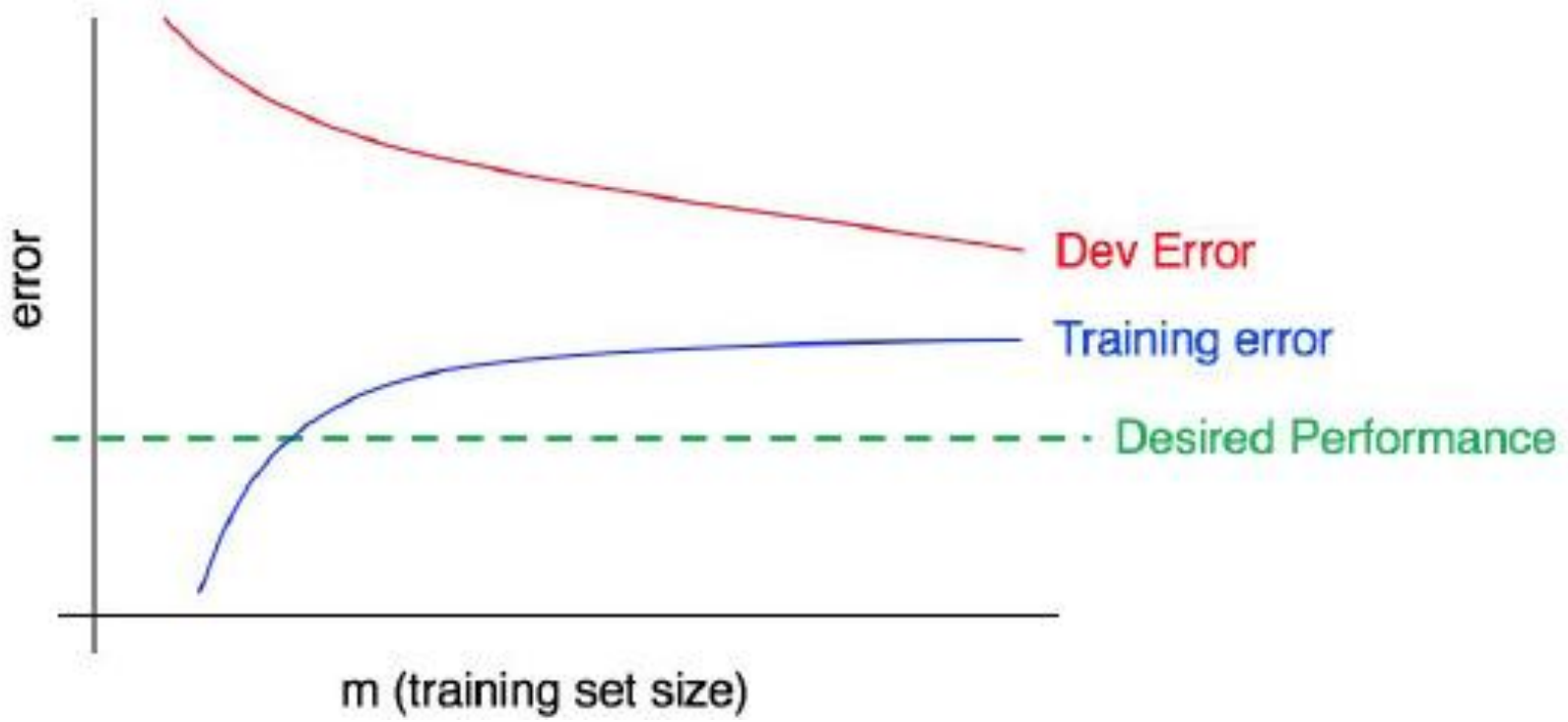
学习曲线

- 解读学习曲线：其他情况？



学习曲线

- 解读学习曲线：其他情况？



学习曲线

- **解读学习曲线**

最后提一点，绘制一个学习曲线的成本可能非常高：例如，你可能需要训练 10 个模型，其中样本规模可以是 1000 个，然后是 2000 个，一直到 10000 个。使用小数据集训练模型比使用型数据集要快得多。因此，你可以用 1000、2000、4000、6000 和 10000 个样本来训练模型，而不是像上面那样将训练集的大小均匀地间隔在一个线性的范围内。这仍然可以让你对学习曲线的变化趋势有一个清晰的认识。正因为绘制学习曲线的高成本，这种技术（非均匀间隔）只有在训练所有额外模型所需的**计算成本很重要**时才有意义。

Lectures contents

- 设置开发集与测试集
- 基础误差分析
- 偏差与方差
- 学习曲线

- **与人类表现水平对比**
- **在不同的分布上训练与测试**
- **端到端深度学习**
- **根据组件进行误差分析**

End of Unit 1