

目录

第一周 机器学习策略（1）	2
1.1 为什么是 ML 策略?	2
1.2 正交化.....	3
1.3 单一数字评估指标.....	5
1.4 满足和优化指标.....	7
1.5 训练/开发/测试集划分.....	9
1.6 开发集和测试集的大小.....	12
1.7 什么时候该改变开发/测试集和指标?	13
1.8 为什么是人的表现?	16
1.9 可避免偏差.....	18
1.10 理解人的表现.....	19
1.11 超过人的表现.....	21
1.12 改善你的模型的表现.....	23
第二周：机器学习策略（2）	24
2.1 进行误差分析.....	24
2.2 清除标注错误的数据.....	26
2.3 快速搭建你的第一个系统，并进行迭代.....	29
2.4 使用来自不同分布的数据进行训练和测试.....	30
2.5 数据分布不匹配时的偏差与方差的分析	32
2.6 处理数据不匹配问题.....	34
2.7 迁移学习（Transfer learning）	37
2.8 多任务学习（Multi-task learning）	39
2.9 什么是端到端的深度学习？（What is end-to-end deep learning?）	42
2.10 是否要使用端到端的深度学习？（Whether to use end-to-end learning?）	45

第一周 机器学习策略（1）

1.1 为什么是 ML 策略？

如何构建你的机器学习项目也就是说机器学习的策略。我希望通过这门课程你们能够学到如何更快速高效地优化你的机器学习系统。

Motivating example



Ideas:

- Collect more data ←
- Collect more diverse training set
- Train algorithm longer with gradient descent
- Try Adam instead of gradient descent
- Try bigger network
- Try smaller network
- Try dropout
- Add L_2 regularization
- Network architecture
 - Activation functions
 - # hidden units
 - ...

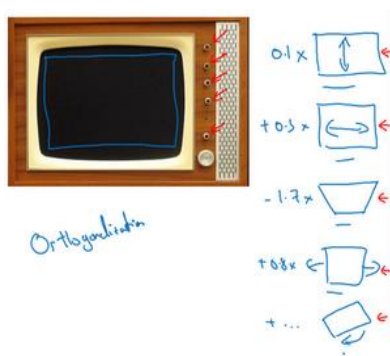
Andrew Ng

我们从一个启发性的例子开始讲，假设正在调试你的猫分类器，经过一段时间的调整，系统达到了 90% 准确率，但对应用程序来说还不够好。我们可能有很多想法去改善系统，比如，我们去收集更多的训练数据；或者可能训练集的多样性还不够，应该收集更多不同姿势的猫咪图片，或者更多样化的反例集；或者想再用梯度下降训练算法，训练久一点；或者想尝试用一个完全不同的优化算法，比如 **Adam** 优化算法；或者尝试使用规模更大或者更小的神经网络；或者想试试 **dropout** 或者 L_2 正则化；或者想修改网络的架构，比如修改激活函数，改变隐藏单元的数目之类的方法。

当尝试优化一个深度学习系统时，通常可以有很多想法可以去试，问题在于如果做出了错误的选择，完全有可能白费 6 个月的时间，往错误的方向前进，在 6 个月之后才意识到这方法根本不管用。比如，我见过一些团队花了 6 个月时间收集更多数据，却在 6 个月之后发现，这些数据几乎没有改善他们系统的性能。所以如果有快速有效的方法能够判断哪些想法是靠谱的，甚至提出新的想法，判断哪些是值得一试的想法，哪些是可以放心舍弃的。在这门课程中，可以学习一些策略，一些分析机器学习问题的方法，可以朝着最有希望的方向前进。

1.2 正交化

搭建建立机器学习系统的挑战之一是，可以尝试和改变的东西太多太多了，比如说，有那么多的超参数可以调。但那些效率很高的机器学习专家有个特点，他们对于要调整什么来达到某个效果，非常清楚，这个步骤我们称之为正交化，让我告诉你是什么意思吧。



这是一张老式电视图片，有很多旋钮可以用来调整图像的各种性质，可能有一个旋钮用来调图像垂直方向的高度，另外有一个旋钮用来调图像宽度，还有一个旋钮用来调整图像左右偏移，还有一个旋钮用来调图像旋转角度之类的。电视设计师花了大量时间设计电路，那时通常都是模拟电路来确保每个旋钮都有相对明确的功能，如一个旋钮来调整高度，一个旋钮调整宽度，一个旋钮调整梯形角度，以此类推。

想象一下，如果有一个旋钮调的是 $0.1x$ 表示图像高度， $+0.3x$ 表示图像宽度， $-1.7x$ 表示梯形角度， $+0.8x$ 表示图像在水平轴上的坐标之类的。如果调整其中一个旋钮，那么图像的高度、宽度、梯形角度、平移位置全部都会同时改变，那几乎不可能把电视调好，让图像显示在区域正中。所以在这种情况下，正交化指的是电视设计师设计这样的旋钮，使得每个旋钮都只调整一个性质，这样调整电视图像就容易得多，就可以把图像调到正中。

那么这与机器学习有什么关系呢？要弄好一个监督学习系统，通常需要调系统的旋钮。这要求确保四件事情：①通常必须确保系统在训练集上得到的结果不错，所以训练集上的表现必须通过某种评估，达到能接受的程度。②在训练集上表现不错之后，希望系统也能在开发集上有好的表现。③希望系统在测试集上也有好的表现。④希望系统在测试集上系统的成本函数在实际使用中表现令人满意，比如说，希望这些猫图片应用的用户满意。

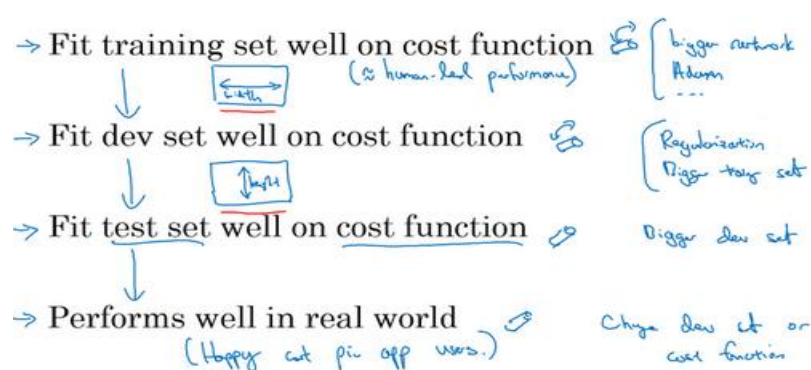
如果算法在成本函数上不能很好地拟合训练集，此时应该设置一个旋钮，这样可以调整算法，让它很好地拟合训练集，所以用来调试的旋钮是可能训练更大的网络，或者可以切换到更好的优化算法，比如 **Adam** 优化算法等等。

如果算法在开发集上做的不好，它在训练集上做得很好，但开发集不行，此时应该有一组正则化的旋钮可以调节。类比到电视，就是现在调好了电视的宽度，如果图像的高度不太对，就需要一个不同的旋钮来调节电视图像的高度，然后希望这个旋钮尽量不会影响到电视的宽度。增大训练集可以是另一个可用的旋钮，它可以帮助算法更好地归纳开发集的规律，现在调好了电视图像的高度和宽度。

如果系统在开发集上做的很好，但测试集上做得不好呢？如果是这样，那么需要调的旋钮，可能是更大的开发集。因为如果它在开发集上做的不错，但测试集不行这可能意味着你对开发集过拟合了，你需要往回退一步，使用更大的开发集。

最后如果它在测试集上做得很好，但无法给你的猫图片应用用户提供良好的体验，这意味着需要回去改变开发集或成本函数。因为如果根据某个成本函数，系统在测试集上做的很好，但它无法反映算法在现实世界中的表现，这意味着要么开发集分布设置不正确，要么成本函数测量的指标不对。

Chain of assumptions in ML



我们很快会逐一讲到这些例子，以后会详细介绍这些特定的旋钮，现在只需要对正交化过程有个概念。我们要非常清楚，到底是四个问题中的哪一个，知道可以调节哪些不同的东西尝试解决那个问题。

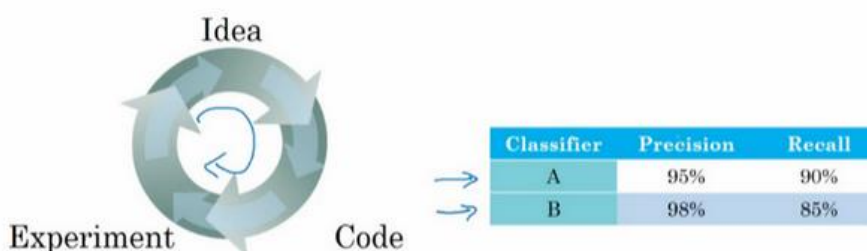
当我训练神经网络时，我一般不用 **early stopping**，这个技巧也还不错，很多人都这么干。但个人而言，我觉得早期停止有点难以分析，因为这个旋钮会同时影响你对训练集的拟合，因为如果你用 **early stopping**，那么对训练集的拟合就不太好，但它同时也用来改善开发集的表现，所以这个旋钮没那么正交化。因为它同时影响两件事情，就像一个旋钮同时影响电视图像的宽度和高度。

在机器学习中，如果可以观察系统，然后说这一部分是错的，它在训练集上做的不好、在开发集上做的不好、它在测试集上做的不好，或者它在测试集上做的不错，但在现实世界中不好，这就很好。必须弄清楚到底是什么地方出问题了，然后我们刚好有对应的旋钮，刚好可以解决那个问题，那个限制了机器学习系统性能的问题。这就是我们这周和下周要讲到的，如何诊断出系统性能瓶颈到底在哪。还有找到你可以用的一组特定的旋钮来调整你的系统，来改善它特定方面的性能，我们开始详细讲讲这个过程吧。

1.3 单一数字评估指标

(Single number evaluation metric) 无论是调整超参数，或者是尝试不同的学习算法，或者在搭建机器学习系统时尝试不同手段，如果一个单实数评估指标，进展会快得多，它可以快速判断新尝试比之前好还是差。

应用机器学习是一个非常经验性的过程，我们通常有一个想法，编程序，跑实验，看看效果如何，然后使用这些实验结果来改善想法，然后继续走这个循环，不断改进算法。比如说对于猫分类器，之前搭建了某个分类器A，通过改变超参数，还有改变训练集等手段，现在训练出来了一个新的分类器B，所以评估分类器的一个合理方式是观察它的查准率 (precision) 和查全率 (recall)。



查准率: 在分类器标记为猫的例子中，有多少真的是猫。所以如果分类器A有 95% 的查准率，这意味着分类器说这图有猫的时候，有 95% 的机会真的是猫。

查全率: 对于所有真猫的图片，你的分类器正确识别出了多少百分比。实际为猫的图片中，有多少被系统识别出来？如果分类器A查全率是 90%，这意味着对于所有的图像，比如说开发集都是真的猫图，分类器A准确地分辨出了其中的 90%。

事实证明，查准率和查全率往往需要折中，两个指标都要顾及到。我们希望的效果是，当分类器说某个东西是猫的时候，很大机会它真的是一只猫，但对于所有是猫的图片，也希望系统能够将大部分分类为猫，所以用查准率和查全率来评估分类器是比较合理的。

但使用查准率和查全率作为评估指标的时候，如果分类器A在查全率上表现更好，分类器B在查准率上表现更好，就无法判断哪个分类器更好。如果有很多分类器，就这两个评估指标而言，很难去快速地选择较好的分类器，所以需要找到一个新的评估指标，这个新评估指标能够结合查准率和查全率。结合查准率和查全率的方法是 F_1 分数。非正式的，可以认为这是查准率 P 和查全率 R 的平均值， $F_1 = \frac{2}{\frac{1}{P} + \frac{1}{R}}$ ，这个函数叫做查准率 P 和查全率 R 的调和平均数 (harmonic mean)。

$$F_1 \text{ score} = \text{"Average" of } P \text{ and } R. \\ \left(\frac{2}{\frac{1}{P} + \frac{1}{R}} \right) \text{ "Harmonic mean"}$$

但在这个例子中可以马上看出，分类器A的 F_1 分数更高。假设 F_1 分数是结合查准率和查全率的合理方式，你可以快速选出分类器A，淘汰分类器B。

Classifier	Precision	Recall	F1 Score
A	<u>95%</u>	<u>90%</u>	<u>92.4%</u>
B	<u>98%</u>	85%	91.0%

我发现很多机器学习团队都有一个定义明确的开发集用来测量查准率和查全率，再加上这样一个单一数值评估指标，能快速判断分类器A或者分类器B更好。再者，迭代速度也会很快，它可以加速改进机器学习算法的迭代过程。

我们来看另一个例子，假设在开发一个猫应用来服务四地区的爱猫人士，美国、中国、印度还有世界其他地区。我们假设两个分类器在来自四个地区的数据中得到了不同的错误率，比如算法A在美国用户上传的图片中达到了 3%错误率等等。

Algorithm	US	China	India	Other
A	<u>3%</u>	7%	5%	9%
B	5%	6%	5%	10%
C	2%	3%	4%	5%
D	5%	8%	7%	2%
E	4%	5%	2%	4%
F	7%	11%	8%	12%

所以跟踪一下，分类器在不同市场和四地区中的表现应该是有用的，但是通过跟踪四个数字，很难扫一眼这些数值就快速判断算法A或算法B哪个更好。所以在这个例子中，除了跟踪分类器在四个不同的地理大区的表现，还可以算一算平均值。假设平均表现是一个合理的单实数评估指标，通过计算平均值就可以快速判断。看起来算法C的平均错误率最低，然后可以继续用算法C。

Algorithm	US	China	India	Other	Average
A	<u>3%</u>	7%	5%	9%	6%
B	5%	6%	5%	10%	6.5%
C	2%	3%	4%	5%	<u>3.5%</u>
D	5%	8%	7%	2%	5.25%
E	4%	5%	2%	4%	3.75%
F	7%	11%	8%	12%	9.5%

1.4 满足和优化指标

（Satisficing and optimizing metrics）要把顾及到的所有事情组合成单实数评估指标有时并不容易，在那些情况里，设立满足和优化指标是很重要的。

假设我们已经决定看重猫分类器的分类准确度，但除了准确度之外，还需要考虑运行时间，就是需要多长时间来分类一张图。分类器A需要 80 毫秒，B需要 95 毫秒，C需要 1500 毫秒来分类图像。

Classifier	Accuracy	Running time
A	90%	80ms
B	92%	95ms
C	95%	1,500ms

Handwritten notes on the table: "optimizing" with an arrow pointing to Accuracy, and "satisficing" with an arrow pointing to Running time. A blue circle highlights the 92% accuracy for classifier B, and a blue arrow points to the 95ms running time for classifier B.

Handwritten formula: $Cost = accuracy - 0.5 \times runningTime$

Handwritten constraints: $maximize\ accuracy$
 $subject\ to\ runningTime \leq 100\ ms.$

Handwritten summary: $N\ metrics:$
 1 optimizing
 $N-1\ satisficing$

我们可以这么做，将准确度和运行时间组合成一个整体评估指标。比如说，总体成本是 $cost = accuracy - 0.5 \times runningTime$ ，这种组合方式可能太刻意，只用这样的公式来组合准确度和运行时间，两个数值的线性加权求和。

我们还可以做其他事情，就是选择一个分类器，能够最大限度提高准确度，但必须满足运行时间要求，就是图像分类所需的时间必须小于等于 100 毫秒。所以在这种情况下，我们就说**准确度是一个优化指标（Optimizing metric）**，因为我们想要准确度尽可能准确，所以最大化优化指标。**运行时间是满足指标（Satisficing metric），意思是它必须足够好，它只需要小于 100 毫秒**，达到之后即可，我们不在乎这指标有多好。所以这是一个相当合理的权衡方式，或者说将准确度和运行时间结合起来的方式。实际情况是，只要运行时间少于 100 毫秒，用户就不会在乎运行时间是 100 毫秒还是 50 毫秒，甚至更快。在这种情况下分类器 B 最好，因为在所有的运行时间都小于 100 毫秒的分类器中，它的准确度最好。

更一般地说，如果要考虑 N 个指标，有时选择其中一个指标作为优化指标是合理的，尽量优化那个指标，然后剩下 $N - 1$ 个都是满足指标，意味着只要它们达到一定阈值即可，例如运行时间快于 100 毫秒，而不在乎它超过那个门槛之后的表现。

这里是另一个例子，假设我们正在构建一个系统来检测唤醒语（**trigger word detection**），这指的是语音控制设备。比如亚马逊 **Echo**，你会说“**Alexa**”，或者用“**Okay Google**”来唤醒谷歌设备，或者对于苹果设备，你会说“**Hey Siri**”，或者对于某些百度设备，我们用“你好百度”唤醒。

Wakewords / Trigger words
 Alexa, OK Google,
 Hey Siri, nǐ hǎo bǎi dù
你好百度

这些唤醒词可以唤醒语音控制设备，然后监听我们想说的话。所以我们一定要考虑触发字检测系统的准确性（优化指标），当有人说出其中一个触发词时，有多大概率可以唤醒设备。也可能需要顾及假阳性（**false positive**）的数量，就是没有人在说这个触发词时，它被随机唤醒的概率有多大？所以这种情况下，组合这两种评估指标的合理方式可能是最大化精确度。所以当某人说出唤醒词时，设备被唤醒的概率最大化，满足指标，比如说，必须满足 24 小时内最多只能有 1 次假阳性。所以在这种情况下，准确度是优化指标，然后每 24 小时发生一次假阳性是满足指标，你只要每 24 小时最多有一次假阳性就满足了。

accuracy.
 #false positive

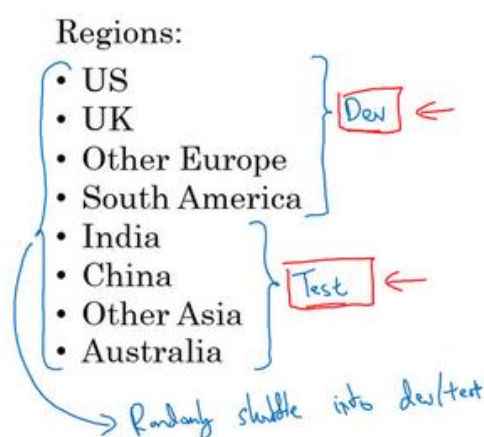
maximize accuracy.
 s.t. ≤ 1 false positive
every 24 hours.

总结一下，如果需要顾及多个指标，比如说，有一个优化指标，你想尽可能优化的，然后还有一个或多个满足指标，需要满足的，需要达到一定的门槛。现在就有一个全自动的方法，在观察多个成本大小时，选出“最好的”那个。现在这些评估指标必须是在训练集或开发集或测试集上计算或求出来的。所以还需要做一件事，就是设立训练集、开发集，还有测试集。在下一个视频里，我想和大家分享一些如何设置训练、开发和测试集的指导方针，我们下一个视频继续。

1.5 训练/开发/测试集划分

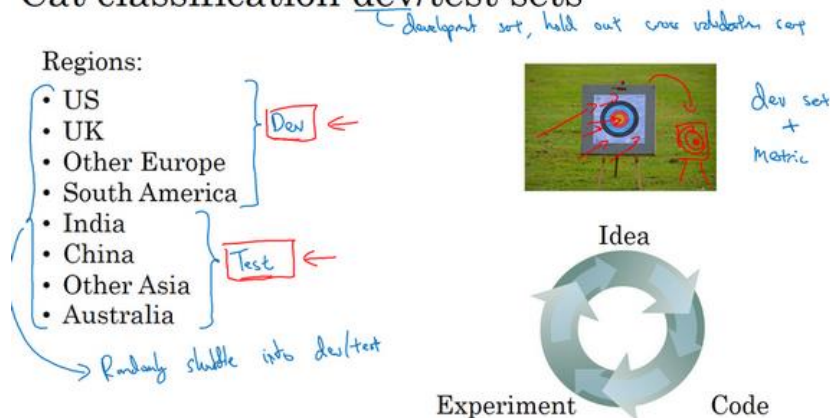
(Train/dev/test distributions) 设立训练集，开发集和测试集的方式大大影响在建立机器学习应用方面取得进展的速度。同样的团队，即使是大公司里的团队，在设立这些数据集的方式，真的会让团队的进展变慢而不是加快，我们看看应该如何设立这些数据集，让团队效率最大化。

在这个视频中，我讨论如何设立开发集和测试集，开发（dev）集也叫做（development set），有时称为保留交叉验证集（hold out cross validation set）。现在，举个例子，我们要开发一个猫分类器，然后在这些区域里运营，美国、英国、其他欧洲国家，南美洲、印度、中国，其他亚洲国家和澳大利亚，那么应该如何设立开发集和测试集呢？



其中一种做法是，可以选择前 4 个区域，但也可以是随机选的区域，然后说来自这四个区域的数据构成开发集，然后其他四个区域（后四个），也可以随机选择 4 个，这些数据构成测试集。事实证明，这个想法非常糟糕，因为这个例子中，开发集和测试集来自不同的分布。我么想设立开发集加上一个单实数评估指标，这就是像是定下目标，然后告诉团队要瞄准的靶心，因为一旦建立了这样的开发集和指标，团队就可以快速迭代，尝试不同的想法，跑实验，可以很快地使用开发集和指标去评估不同分类器，然后尝试选出最好的那个。所以机器学习团队一般都很擅长使用不同方法去逼近目标，然后不断迭代，不断逼近靶心。

Cat classification dev/test sets

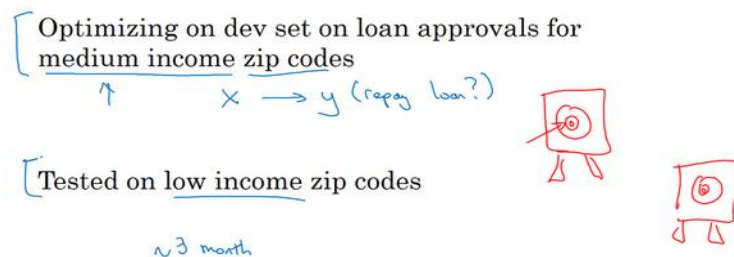


Andrew Ng

在左边的例子中，设立开发集和测试集时存在一个问题，团队可能会花上几个月时间在开发集上迭代优化，结果发现，当在测试集上测试系统时，来自这四个国家或者说下面这四个地区的数据（即测试集数据）和开发集里的数据可能差异很大，所以可能会收获“意外惊喜”，并发现花了那么多个月的时间去针对开发集优化，在测试集上的表现却不佳。所以，如果开发集和测试集来自不同的分布，就像你设了一个目标，让团队花几个月尝试逼近靶心，结果在几个月工作之后发现，测试的时候，你说“等等，我要把目标移到这里”，然后团队可能会说“好吧，为什么你让我们花那么多个月的时间去逼近那个靶心，然后突然间你可以把靶心移到不同的位置？”。所以，**为了避免这种情况，我建议的是将所有数据随机洗牌，放入开发集和测试集，开发集和测试集都有来自八个地区的数据，并且开发集和测试集都来自同一分布，这分布就是将所有数据混在一起。**

这里是另一个例子，这是个真实的故事，但有一些细节变了。有一个机器学习团队，花了好几个月在开发集上优化，开发集里面有中等收入邮政编码的贷款审批数据。那么具体的机器学习问题是，输入 x 为贷款申请，你是否可以预测输出 y ， y 是他们有没有还贷能力？所以这系统能帮助银行判断是否批准贷款。所以开发集来自贷款申请，这些贷款申请来自中等收入邮政编码，**zip code** 就是美国的邮政编码。但是在这上面训练了几个月之后，团队突然决定要在，低收入邮政编码数据上测试一下。当然了，这个分布数据里面中等收入和低收入邮政编码数据是很不一样的，而且他们花了大量时间针对前面那组数据优化分类器，导致系统在后面那组数据中效果很差。所以这个特定团队实际上浪费了 3 个月的时间，不得不退回去重新做很多工作。

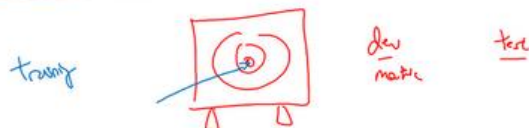
True story (details changed)



这里实际发生的是，这个团队花了三个月瞄准一个目标，三个月后经理突然问：“你们试试瞄准那个目标如何？”，这新目标位置完全不同，所以这件事对于这个团队来说非常崩溃。

Guideline

Choose a dev set and test set to reflect data you expect to get in the future and consider important to do well on.



所以我建议你们在设立开发集和测试集时，要选择这样的开发集和测试集，能够反映你未来会得到数据，认为很重要的数据，必须得到好结果的数据，特别是，这里的开发集和测试集可能来自同一个分布。所以不管你未来会得到什么样的数据，一旦你的算法效果不错，要尝试收集类似的数据，而且，不管那些数据是什么，都要随机分配到开发集和测试集上。因为这样，你才能将瞄准想要的目标，让你的团队高效迭代来逼近同一个目标，希望最好是同一个目标。

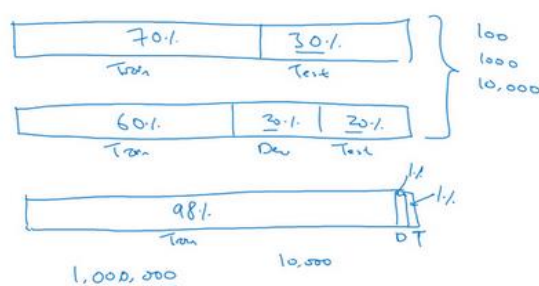
我们还没提到如何设立训练集，我们会在之后的视频里谈谈如何设立训练集，但这个视频的重点在于，设立开发集以及评估指标，真的就定义了你瞄准的目标。我们希望通过在同一分布中设立开发集和测试集，你就可以瞄准你所希望的机器学习团队瞄准的目标。而设立训练集的方式则会影响你逼近那个目标有多快，但我们可以在另一个讲座里提到。我知道有一些机器学习团队，他们如果能遵循这个方针，就可以省下几个月的工作，所以我希望这些方针也能帮到你们。

接下来，实际上你的开发集和测试集的规模，如何选择它们的大小，在深度学习时代也在变化，我们会在下一个视频里提到这些内容。

1.6 开发集和测试集的大小

(Size of dev and test sets) 在上一个视频中你们知道了你的开发集和测试集为什么必须来自同一分布，但它们规模应该多大？在深度学习时代，设立开发集和测试集的方针也在变化，我们来看看一些最佳做法。

Old way of splitting data

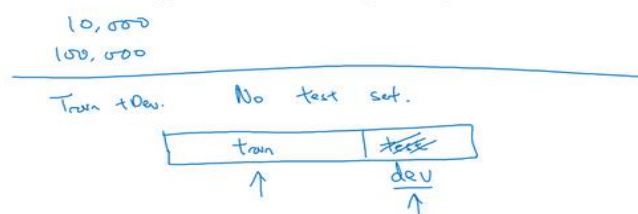


在机器学习中，把全部数据用 70/30 比例分成训练集和测试集。或者如果必须设立训练集、开发集和测试集，60%训练集，20%开发集，20%测试集。在机器学习的早期，这样分是相当合理的，特别是以前的数据集大小要小得多。所以如果总共有 100 个样本，这样 70/30 或者 60/20/20 分的经验法则是相当合理的。如果有几千个样本或者有一万个样本，这些做法也还是合理的。

在现代机器学习中，我们更习惯操作规模大得多的数据集，比如说有 1 百万个训练样本，这样分可能更合理，98%作为训练集，1%开发集，1%测试集，用D和T缩写来表示开发集和测试集。因为如果有 1 百万个样本，1%就是 10,000 个样本，这对于开发集和测试集来说可能已经够了。而且因为深度学习算法对数据的胃口很大，我们可以看到那些有海量数据集的问题，有更高比例的数据划分到训练集里，那么测试集呢？要记住，测试集的目的是完成系统开发之后，测试集可以帮你评估投产系统的性能。方针就是，令你的测试集足够大，能够以高置信度评估系统整体性能。所以除非需要对最终投产系统有一个很精确的指标，一般来说测试集不需要上百万个例子。对于应用程序有 10,000 个例子就足够给出性能指标了。

Size of test set

→ Set your test set to be big enough to give high confidence in the overall performance of your system.



测试集的目的是评估最终的成本偏差，只需要设立足够大的测试集，可以用来这么评估就行了，可能只需要远远小于总体数据量的 30%。

1.7 什么时候该改变开发/测试集和指标？

我们已经学过如何设置开发集和评估指标，就像是把目标定在某个位置，让团队瞄准。但有时候在项目进行中，可能意识到目标的位置放错了，此时我们应该移动目标。

来看一个例子，假设我们在构建一个猫分类器，试图找到很多猫的照片，向爱猫人士用户展示，我们决定使用的指标是分类错误率。算法A和B分别有 3% 错误率和 5% 错误率，所以算法A似乎做得更好。但我们实际试一下这些算法，假设算法A由于某些原因，把很多色情图像分类成猫了，此时如果部署算法A，那么用户在看到猫图的同时，因错误率有 3%，用户也会看到一些色情图像，这是完全不能接受的。相比之下，算法B有 5% 的错误率，这样分类器就得到较少的图像，但它不会推送色情图像。所以算法B实际上是一个更好的算法，因为它不让任何色情图像通过。

Cat dataset examples

Metric + Dev : Prefer A
You/users : Prefer B.

→ Metric: classification error

Algorithm A: 3% error → pornographic

✓ Algorithm B: 5% error

$$\left\{ \begin{array}{l} \text{Error: } \frac{1}{\sum w^{(i)}} \sum_{i=1}^{m_{dev}} w^{(i)} I\{y_{pred}^{(i)} \neq y^{(i)}\} \\ \rightarrow w^{(i)} = \begin{cases} 1 & \text{if } x^{(i)} \text{ is non-porn} \\ 10 & \text{if } x^{(i)} \text{ is porn} \end{cases} \end{array} \right.$$

在这个例子中，算法A在评估指标上做得更好，它的错误率达到 3%，但实际上是个糟糕的算法。在这种情况下，评估指标加上开发集都倾向于选择算法A，因为算法A的错误率较低。但用户更倾向于使用算法B，因为它不会将色情图像分类为猫。当这种情况发生时，评估指标无法正确衡量算法之间的优劣排序时，原来的指标错误地预测算法A是更好的算法就发出了信号，应该改变评估指标，或者要改变开发集或测试集。在这种情况下，分类错误率指标可以写成这样：

$$Error = \frac{1}{m_{dev}} \sum_{i=1}^{m_{dev}} I\{y_{pred}^{(i)} \neq y^{(i)}\}$$

m_{dev} 是开发集数，用 $y_{pred}^{(i)}$ 表示预测值，其值为 0 或 1， I 这符号表示一个函数，统计出里面这个表达式为真的样本数，所以这个公式就统计了分类错误的样本。这个评估指标的问题在于，它对色情和非色情图片一视同仁，但其实我们希望分类器不会错误标记色情图像。一个修改评估指标的方法是， $\frac{1}{m_{dev}}$ 与 $\sum_{i=1}^{m_{dev}} I\{y_{pred}^{(i)} \neq y^{(i)}\}$ 之间加个权重项 $w^{(i)}$ ，即：

$$Error = \frac{1}{m_{dev}} \sum_{i=1}^{m_{dev}} w^{(i)} I\{y_{pred}^{(i)} \neq y^{(i)}\}$$

Error:

$$\frac{1}{m_{dev}} \sum_{i=1}^{m_{dev}} w^{(i)} I\{y_{pred}^{(i)} \neq y^{(i)}\}$$

↖ predicted value (0/1)

$$w^{(i)} = \begin{cases} 1 & \text{if } x^{(i)} \text{ is non-porn} \\ 10 & \text{if } x^{(i)} \text{ is porn} \end{cases}$$

其中如果图片 $x^{(i)}$ 不是色情图片，则 $w^{(i)} = 1$ 。如果 $x^{(i)}$ 是色情图片， $w^{(i)}$ 可能就是 10 甚至 100，这样你赋予了色情图片更大的权重，让算法将色情图分类为猫图时，错误率快速变大。这个例子里，把色情图片分类成猫这一错误的惩罚权重加大 10 倍。

如果你希望得到归一化常数，在技术上，就是 $w^{(i)}$ 对所有 i 求和，这样错误率仍然在 0 和 1 之间，即：

$$Error = \frac{1}{\sum w^{(i)}} \sum_{i=1}^{m_{dev}} w^{(i)} I\{y_{pred}^{(i)} \neq y^{(i)}\}$$

Error:

$$\frac{1}{\sum w^{(i)}} \sum_{i=1}^{m_{dev}} w^{(i)} I\{y_{pred}^{(i)} \neq y^{(i)}\}$$

↖ predicted value (0/1)

$$\rightarrow w^{(i)} = \begin{cases} 1 & \text{if } x^{(i)} \text{ is non-porn} \\ 10 & \text{if } x^{(i)} \text{ is porn} \end{cases}$$

加权的细节并不重要，实际上要使用这种加权，我们必须自己过一遍开发集和测试集，在开发集和测试集里，自己把色情图片标记出来，这样才能使用这个加权函数。但结论是，如果评估指标无法正确评估好算法的排名，那么就需要花时间定义一个新的评估指标。评估指标的意义在于，准确分辨两个分类器，哪一个更适合实际应用。

我们再讲一个例子，假设两个猫分类器A和B，错误率分别是 3%和 5%，测试集来自于网上下载的图片，这些是高质量，取景框很专业的图像，但当部署到手机应用时，算法作用到用户上传的图片时，那些图片取景不专业，没有把猫完整拍下来，或者猫的表情很古怪，也许图像很模糊，当实际测试算法时，发现算法B表现其实更好。

Another example

Algorithm A: 3% error

✓ Algorithm B: 5% error ←

→ Dev/test



→ User images



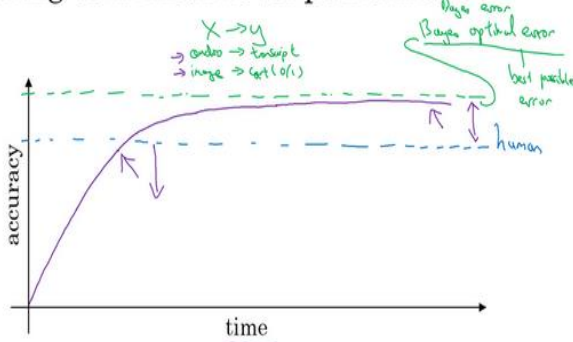
If doing well on your metric + dev/test set does not correspond to doing well on your application, change your metric and/or dev/test set.

这是另一个指标和开发集测试集出问题的例子，问题在于，做评估用的是很漂亮的高分辨率的开发集和测试集，图片取景很专业。但用户真正关心的是，他们上传的图片能不能被正确识别。那些图片可能是没那么专业的照片，有点模糊，取景很业余。所以方针是，如果在当前开发集或者开发集和测试集分布中表现很好，但实际应用程序，真正关注的地方表现不好，那么就需要修改指标或者开发测试集。换句话说，如果发现开发测试集都是这些高质量图像，但在开发测试集上做的评估无法预测应用实际的表现，因为应用处理的是低质量图像，那么就应该改变开发测试集，让数据更能反映实际需要处理好的数据。

1.8 为什么是人的表现？

过去的几年里，更多的机器学习团队一直在讨论如何比较机器学习系统和人类的表现，为什么呢？我认为有两个主要原因，首先是因为深度学习系统的进步，机器学习算法变得更好了，在许多机器学习的应用领域已经有算法可以威胁到人类的表现了。其次，事实证明，当试图让机器做人类能做的事情时，可以精心设计机器学习系统的工作流程，让工作流程效率更高，所以在这些场合，比较人类和机器，或让机器模仿人类的行为是很自然的。

Comparing to human-level performance



在很多机器学习任务中，一些团队研究一个问题，当开始往人类水平努力时，进展是很快，但是过了一段时间，当这个算法表现比人类更好时，进展和精确度的提升就变得更慢了。在超越人类水平之后，它还可以变得更好，但性能增速，准确度上升的速度这个斜率，会变得越来越平缓，我们都希望能达到理论最佳性能水平。随着时间的推移，当继续训练算法时，可能模型越来越大，数据越来越多，但是性能无法超过某个理论上限，这个上限就是贝叶斯最优错误率（**Bayes optimal error**），它一般认为是理论上可能达到的最优错误率，就是说没有任何办法设计出一个 x 到 y 的函数，让它能够超过一定的准确度。

例如，对于语音识别来说，如果 x 是音频片段，有些音频就是这么嘈杂，基本不可能知道说的是什么，所以完美的准确率可能不是 100%。或者对于猫图识别来说，也许一些图像非常模糊，不管是人类还是机器，都无法判断该图片中是否有猫。所以，完美的准确度可能不是 100%。而贝叶斯最优错误率有时写作 **Bayesian error**，即省略 **optimal**，就是从 x 到 y 映射的理论最优函数，永远不会被超越。

事实证明，机器学习的进展往往相当快，直到超越人类的表现之前一直很快，当你超越人类的表现时，有时进展会变慢。我认为有两个原因：一是，人类水平在很多任务中离贝叶斯最优错误率已经不远了，人们非常擅长看图像，分辨里面有没有猫或者听写音频。所以，当超越人类的表现之后也许没有太多的空间继续改善了。二是，只要算法表现比人类的表现更差，那么实际上可以使用某些工具来提高性能。一旦你超越了人类的表现，这些工具就没那么好用了。

Why compare to human-level performance

Humans are quite good at a lot of tasks. So long as ML is worse than humans, you can:

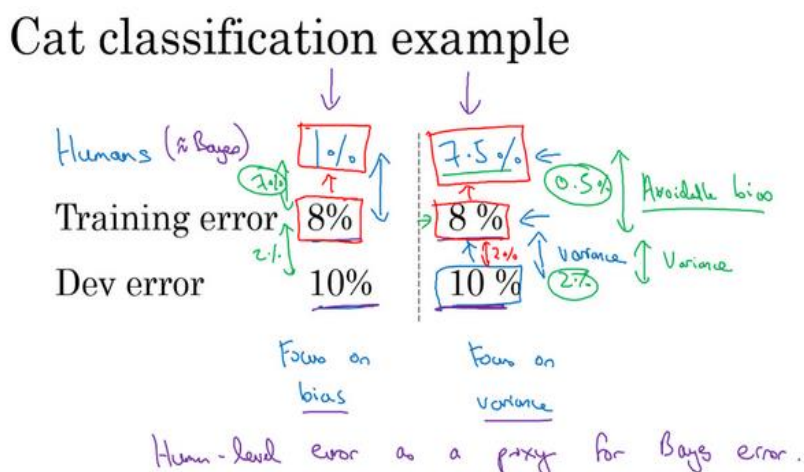
- - Get labeled data from humans. (x, y)
- - Gain insight from manual error analysis:
Why did a person get this right?
- - Better analysis of bias/variance.

比如，对于人类相当擅长的任务，包括看图识别事物，听写音频，或阅读语言，人类一般很擅长处理这些自然数据。对于人类擅长的任务，只要机器学习算法的表现比人类差，就可以从让人帮你标记数据，这样你就有更多的数据可以喂给学习算法；下周我们会讨论，人工错误率分析，但只要人类的表现比任何其他算法都要好，就可以让人类看看你算法处理的例子，知道错误出在哪里，并尝试了解为什么人能做对，算法做错；也可以更好地分析偏差和方差。但是只要你的算法仍然比人类糟糕，你就有这些重要策略可以改善算法。而一旦你的算法做得比人类好，这三种策略就很难利用了。所以这可能是另一个和人类表现比较的好处，特别是在人类做得很好的任务上。

1.9 可避免偏差

我们讨论过，希望学习算法能在训练集上表现良好，但有时实际上并不想做得太好。我们得知道人类水平的表现是怎样的，可以明确算法在训练集上的表现应该有多好，或者有多不好。我们经常使用猫分类器来做例子，比如人类具有近乎完美的准确度，假设人类水平的错误是 1%。在这种情况下，如果学习算法达到 8% 的训练错误率和 10% 的开发错误率，算法在训练集上的表现和人类水平的表现有很大差距，说明算法对训练集的拟合并不好。所以从减少偏差和方差的工具这个角度看，在这种情况下，我们应该会把重点放在减少偏差上，比如说训练更大的神经网络，或者跑久一点梯度下降。

现在我们看看同样的训练错误率和开发错误率，假设人类的表现不是 1%，而是 7.5%，也许因为数据集中的图像非常模糊，分辨率很低。在这种情况下，我们知道算法在训练集上的表现还好，它只是比人类的表现差一点点（0.5%）。此时我们应该减少学习算法的方差，也许可以试试正则化，让开发错误率更接近训练错误率。



总结一下，在这两种情况下，具有同样的训练错误率和开发错误率，我们如何决定专注于减少偏差的策略或者减少方差的策略。左边的例子中，假设贝叶斯错误率是 1%，那 8% 的训练错误率真的很高，应该可以把它降到 1%，此时减少偏差可能更有效。右边的例子中，如果贝叶斯错误率是 7.5%，这里我们使用人类水平错误率来替代贝叶斯错误率，但训练错误率是 8%，没有太多改善的空间了，不能继续减少训练错误率了，我们也不会希望它比 7.5% 好得多。而这边，训练误差和开发误差之间有更多的改进空间，可以将这个 2% 的差距缩小一点，使用减少方差的手段应该可行，比如正则化，或者收集更多的训练数据。

我们把贝叶斯错误率和训练错误率之间的差值称可避免偏差（**Avoidable bias**），希望一直提高训练集表现，直到接近贝叶斯错误率，但实际上也不希望做到比贝叶斯错误率更好，这理论上是不可能超过贝叶斯错误率的，除非过拟合。而这个训练错误率和开发错误率之前的差值，就大概说明算法在方差问题上还有多少改善空间。

在这个例子中，当理解了人类水平错误率，理解了对贝叶斯错误率的估计，就可以在不同的场景中专注于不同的策略，使用避免偏差策略还是避免方差策略。

1.10 理解人的表现

使用人类水平表现（**Understanding human-level performance**）这个词的定义，可以推动机器学习项目的进展。在上一节中，3 我们用过这个词“人类水平错误率”用来估计贝叶斯误差，那是理论最低的错误率，任何函数不管是现在还是将来，能够到达的最低值。我们先记住这点，然后看看医学图像分类例子。

Human-level error as a proxy for Bayes error

Medical image classification example:

Suppose:

- (a) Typical human 3 % error
 - (b) Typical doctor 1 % error
 - (c) Experienced doctor 0.7 % error
 - (d) Team of experienced doctors .. 0.5 % error ←
- Bayes error \leq 0.5%*



What is “human-level” error?

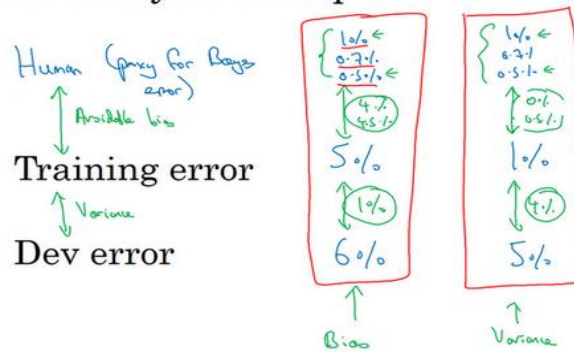
假设我们要观察这样的放射科图像，然后作出分类诊断，假设一个普通的未经训练的人员，在此任务上达到 3% 的错误率；普通的放射科医生，能达到 1% 的错误率；经验丰富的医生错误率为 0.7%；还有一队经验丰富的医生，让他们都看看这个图像，然后他们讨论后达成共识的意见有 0.5% 的错误率，此时人类水平错误率是 3%，1%，0.7% 还是 0.5% 呢？

如果我们想要替代或估计贝叶斯错误率，那么经验丰富的医生讨论后可以达到 0.5% 的错误率，根据定义我们知道贝叶斯错误率应该小于等于 0.5%。因为也许更有经验的医生能做得更好，所以比 0.5% 好一点。但是我们知道最优错误率不能高于 0.5%，此时我们就可以用 0.5% 估计贝叶斯错误率，将人类水平定义为 0.5%。

本节的要点是，在定义人类水平错误率时，要弄清楚目标所在，如果要表明可以超越单个人类，那么就有理由在某些场合部署系统，也许这个定义是合适的。但是如果您的目标是替代贝叶斯错误率，那么这个定义（经验丰富的医生团队——0.5%）才合适。

要了解为什么这个很重要，我们来看一个错误率分析的例子。比方说，在医学图像诊断例子中，训练错误率是 5%，开发错误率是 6%，我们把人类水平的表现看成是贝叶斯错误率的替代品，可能会用 1% 或 0.7% 或 0.5%。同时也回想一下，前面视频中的定义，贝叶斯错误率和训练错误率差值衡量了可避免偏差，训练误差与开发误差之间的差值可以衡量或估计学习算法的方差问题有多严重。所以在这个第一个例子中，无论做出哪些选择，可避免偏差大概是 4%，如果你取 1% 就是 4%，如果取 0.5% 就是 4.5%，而训练误差与开发误差之间的差值是 1%。所以在这个例子中，不管怎么定义人类水平错误率，可避免偏差是 4% 和 4.5%，这明显都比方差大。所以在这种情况下，应该专注于减少偏差的，如训练更大的网络。

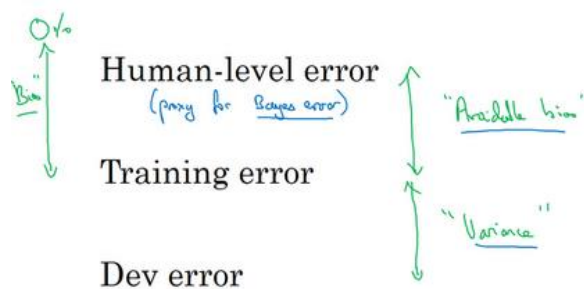
Error analysis example



现在来看看第二个例子，比如训练错误率是 1%，开发错误率是 5%，这其实也不怎么重要，人类水平表现是 1%或 0.7%还是 0.5%。因为不管使用哪一个定义，测量可避免偏差就是 0%到 0.5%之间。而训练错误率和开发错误率之前的差距是 4%，比任何一种可避免偏差都大。所以应该主要使用减少方差的工具，比如正则化或者去获取更大的训练集。

什么时候真正有效呢？就是比如你的训练错误率是 0.7%，所以你现在已经做得很好了，你的开发错误率是 0.8%。在这种情况下，你用 0.5%来估计贝叶斯错误率关系就很大。因为在这种情况下，你测量到的可避免偏差是 0.2%，这是你测量到的方差问题 0.1%的两倍，这表明也许偏差和方差都存在问题。但是，可避免偏差问题更严重。在这个例子中 0.5%，就是对贝叶斯错误率的最佳估计，因为一群人类医生可以实现这一目标。如果用 0.7 代替贝叶斯错误率，你测得的可避免偏差基本上是 0%，那就可能忽略可避免偏差了。实际上应该试试能不能在训练集上做得更好。

Summary of bias/variance with human-level performance



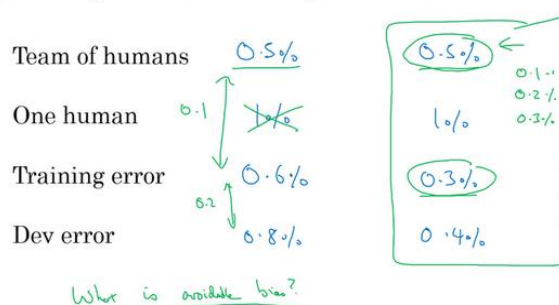
总结一下我们讲到的，如果想理解偏差和方差，那么在人类可以做得很好的任务中，可以估计人类水平的错误率，可以使用人类水平错误率来估计贝叶斯错误率。所以到贝叶斯错误率估计值的差距，可以了解可避免偏差问题有多大，而训练错误率和开发错误率之间的差距告诉你方差上的问题有多大，算法是否能够从训练集泛化推广到开发集。

回顾一下，对人类水平有大概的估计可以对贝叶斯错误率进行估计，这样可以更快地作出决定是否应该专注于减少算法的偏差，或者减少算法的方差。这个决策技巧通常很有效，直到系统性能开始超越人类，那么对贝叶斯错误率的估计就不再准确了，但这些技巧还是可以帮你做出明确的决定。

1.11 超过人的表现

我们讨论过机器学习进展，会在接近或者超越人类水平的时候变得越来越慢。我们举例谈谈为什么会这样。

Surpassing human-level performance



假设有一个问题，一组人类专家充分讨论之后，达到 0.5%的错误率，单个人类专家错误率是 1%，然后训练算法有 0.6%的训练错误率，0.8%的开发错误率。在这种情况下，可避免偏差是 0.1%，方差是 0.2%。和减少可避免偏差比较起来，减少方差可能空间更大。

再来看一个别的例子，一个人类专家团和单个人类专家的表现和以前一样，但算法可以得到 0.3%训练错误率，还有 0.4%开发错误率。现在，可避免偏差是什么呢？现在其实很难回答，基于本例中给出的信息，实际上没有足够的信息来判断优化算法是应该专注减少偏差还是减少方差，这样取得进展的效率就会降低。还有，如果错误率已经比一群充分讨论后的人类专家更低，那么依靠人类直觉去判断算法还能往什么方向优化就很难了。所以在这个例子中，一旦错误率超过这个 0.5%的门槛，要进一步优化机器学习问题就没有明确的选项和前进的方向了。这并不意味着不能取得进展，只是现有的一些工具帮助你指明方向的工具就没那么好用了。

Problems where ML significantly surpasses human-level performance

- - Online advertising
 - - Product recommendations
 - - Logistics (predicting transit time)
 - - Loan approvals
- Handwritten notes on the left:
- Structured data
 - Not natural perception
 - lots of data
- Handwritten notes on the right (enclosed in a bracket):
- Speech recognition
 - Some image recognition
 - Medical
 - ECG, Skin cancer, ...


现在，机器学习有很多问题已经可以大大超越人类水平（**Surpassing human-level performance**）了。例如，①网络广告：估计某个用户点击广告的可能性；②提出产品建议：向你推荐电影或书籍之类的任务；③物流预测：从A到B开车需要多久，或者预测快递车从A到B需要多少时间；④或者预测某人偿还贷款的能力，这样就能判断是否批准这人的贷款。我想这些问题都是今天的机器学习远远超过了单个人类的表现。

请注意这四个例子，所有这四个例子都是从结构化数据中学习得来的，这里可能有个数据库记录用户点击的历史，购物历史数据库，或者从A到B需要多长时间的数据库，以前的贷款申请及结果的数据库，这些并不是自然感知问题，这些不是计算机视觉问题，或语音识别，或自然语言处理任务。人类在自然感知任务中往往表现非常好，所以有可能对计算机来说，在自然感知任务的表现要超越人类要更难一些。


最后，这些问题中，机器学习团队都可以访问大量数据，所以比如说，那四个应用中，最好的系统看到的数据量可能比任何人类能看到的都多，所以这样就相对容易得到超越人类水平的系统。现在计算机可以检索那么多数据，它可以比人类更敏锐地识别出数据中的统计规律。除了这些，今天已经有语音识别系统超越人类水平了，还有一些计算机视觉任务，一些图像识别任务，计算机已经超越了人类水平。但由于人类对这种自然感知任务非常擅长，我想计算机达到那种水平要难得多。还有一些医疗方面的任务，比如阅读 **ECG** 或诊断皮肤癌，或者某些特定领域的放射科读图任务，这些任务计算机做得非常好了，也许超越了单个人类的水平。

1.12 改善你的模型的表现

我们已经学过正交化，如何设立开发集和测试集，用人类水平错误率来估计贝叶斯错误率以及如何估计可避免偏差和方差。我们现在把它们全部组合起来写成一套指导方针，如何提高学习算法性能的指导方针。所以想要让一个监督学习算法达到实用，基本上希望或者假设可以完成两件事情。首先，算法对训练集的拟合很好，做到可避免偏差很低。第二，在训练集中做得很好，然后推广到开发集和测试集也很好，方差不是太大。

1. You can fit the training set pretty well. 

~ Avoidable bias

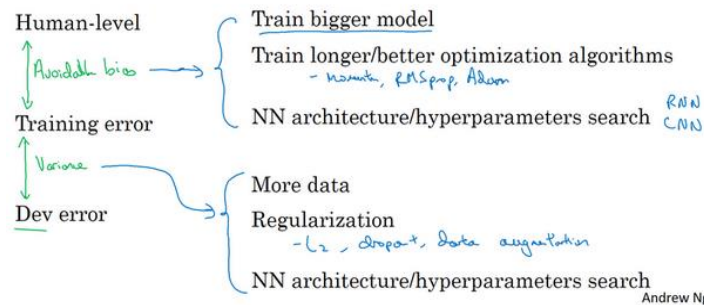
2. The training set performance generalizes pretty well to the dev/test set. 

~ Variance

在正交化的精神下，这里有第一组旋钮，可以修正可避免偏差问题，比如训练更大的网络或者训练更久。第二组旋钮用来处理方差问题，比如正则化或者收集更多训练数据。

总结一下前几段视频我们见到的步骤，如果想提升机器学习系统的性能，建议先看看训练错误率和贝叶斯错误率估计值之间的距离，了解可避免偏差有多大。换句话说就是对训练集的优化还有多少空间。然后看看开发错误率和训练错误率之间的距离，就知道方差问题有多大。换句话说就是做多少努力让算法表现能够从训练集推广到开发集。

Reducing (avoidable) bias and variance



减少可避免偏差，我们可以试试这样的策略：比如使用规模更大的模型，这样算法在训练集上的表现会更好，或者训练更久。使用更好的优化算法，比如说加入 **momentum** 或者 **RMSprop**，或者 **Adam**。还可以试试寻找更好的神经网络架构，或者说更好的超参数。这些手段包罗万有，也可以改变激活函数，改变层数或者隐藏单位数，虽然这么做可能会让模型规模变大，或者试用其他模型，其他架构，如循环神经网络和卷积神经网络。


减少方差，我们可以试试这样的策略：可以收集更多数据，因为收集更多数据去训练可以更好地推广到系统看不到的开发集数据，也可以尝试正则化，包括 **L2** 正则化，**dropout** 正则化或者我们在之前课程中提到的数据增强，同时你也可以试用不同的神经网络架构，超参数搜索。

第二周：机器学习策略（2）

2.1 进行误差分析

（Carrying out error analysis）如果我们希望学习算法能够胜任人类能做的任务，但它还没有达到人类的表现，那么人工检查一下算法犯的的错误，也许可以了解接下来应该做什么。这个过程称为错误分析，我们从一个例子开始讲吧。

Look at dev examples to evaluate ideas



90% accuracy
→ 10% error

Should you try to make your cat classifier do better on dogs? ←

Error analysis:

- Get ~100 mislabeled dev set examples.
- Count up how many are dogs.

→ 5% 10%
5/100 95%

“catting”

→ 50% 10%
50/100 5%

假设正在调试猫分类器，然后取得 90% 准确率，10% 错误率，在开发集上做到这样，距离希望的目标还有很远。你的同事看了一下算法分类出错的例子，注意到算法将一些狗分类为猫，所以你的队友建议，针对狗的图片优化算法。试想一下，为了让猫分类器在狗图上做的更好，让算法不再将狗分类成猫，设计一些只处理狗的算法。但问题在于，应不应该去开始做一个项目专门处理狗？这可能需要花费几个月的时间才能让算法在狗图片上犯更少的错误，这值得吗？这里有个错误分析流程，可以很快知道这个方向是否值得努力。

首先，收集一下比如说 100 个错误标记的开发集样本，然后手动检查，一次只看一个，看看开发集里有多少错误标记的样本是狗。假设 100 个错误标记样本中只有 5% 是狗，就是说有 5 个是狗。这意味着即使完全解决了狗的问题，也只能修正这 100 个错误中的 5 个。换句话说，如果在狗的问题上花了很多时间，那么最多只能将错误率从 10% 下降到 9.5%。错误率相对下降了 5%（总体下降了 0.5%，100 的错误样本，错误率为 10%，则样本为 1000），那就是 10% 下降到 9.5%。此时可以确定这样花时间不好，因为这样分析给出了一个优化上限。在机器学习中，有时我们称之为性能上限，最好能有多少帮助。

但现在假设我们观察一下这 100 个错误标记的开发集样本，有 50 张图都是狗，现在花时间去解决狗的问题可能效果就很好。这种情况下，如果真的解决了狗的问题，那么错误率可能就从 10% 下降到 5% 了，此时可以集中精力减少错误标记的狗图的问题。

在机器学习中，有时候我们很鄙视手工操作，但如果要搭建应用系统，那简单的人工统计步骤，错误分析，可以节省大量时间，可以迅速决定什么是最重要的，或者最有希望的方法。

向。实际上，如果观察 100 个错误标记的开发集样本，也许只需要 5 到 10 分钟的时间，亲自看看这 100 个样本，并亲自统计一下有多少是狗。根据结果，看看有没有占到 5%、50% 或者其他东西。这个在 5 到 10 分钟之内就能估计这个方向有多少价值，并且做出更好的决定，是不是把未来几个月的时间投入到解决错误标记的狗图这个问题。

现在我们要描述一下如何使用错误分析来评估某个想法，有时在做错误分析时，也可以同时并行评估几个想法，比如，也许可以改善针对狗图的性能，或者有时候猫科动物，如狮子，豹，猎豹等，它们经常被分类成小猫或者家猫，或者也许因为有些图像模糊，如果能设计出一些系统，能够更好地处理模糊图像。面对这些想法，要用错误分析来进行评估。

Evaluate multiple ideas in parallel

Ideas for cat detection:

- Fix pictures of dogs being recognized as cats ←
- Fix great cats (lions, panthers, etc..) being misrecognized ←
- Improve performance on blurry images ←

建立一个表格，在最左边人工过一遍想分析的图像集，如果观察 100 张图的话可能是从 1 到 100。表格的一列对应要评估的想法，狗的问题，猫科动物的问题，模糊图像的问题。记住在错误分析过程中，就看算法识别错误的开发集样本，如果第一张识别错误的图片是狗图，那么就打个勾。如果第二张照片很模糊，也记一下。如果第三张是在下雨天动物园里的狮子，被识别成猫了，这是大型猫科动物，图片模糊。最后，这组图像过了一遍之后，可以统计这里每个错误类型的百分比。也许检查图像中 8% 是狗，可能 43% 属于大猫，61% 属于模糊。在这个步骤做到一半时，有时可能会发现其他错误类型，比如发现有 Instagram 滤镜，那些花哨的图像滤镜，干扰了分类器。在这种情况下，实际上可以在错误分析途中，增加这样一列，比如多色滤镜 Instagram 滤镜和 Snapchat 滤镜，然后再过一遍，也统计一下那些问题，并确定这个新的错误类型占了多少百分比。

Image	Dog	Great Cats	Blurry	Instagram	Comments
1	✓			✓	Pitbull
2			✓	✓	
3		✓	✓		Rainy day at zoo
⋮	⋮	⋮	⋮	⋮	
% of total	8%	43%	61%		12%

假设在这个样本中，有很多错误来自模糊图片，也有很多错误类型是大猫图片，这个分析的结果不是说一定要处理模糊图片，这个分析不是一个严格的数学公式，告诉我们应该做什么，但它可以说明，不管狗图片或者 Instagram 图片处理得有多好，最多只能取得 8% 或者 12% 的性能提升。而在大猫图片这一类型或者模糊图像，可以做得更好，性能提高的上限空间要大得多。总结一下，进行错误分析，应找一组错误样本，观察错误标记的样本，看看假阳性（false positives）和假阴性（false negatives），通过统计不同错误标记类型占总数的百分比，可以发现哪些问题需要优先解决。

2.2 清除标注错误的数据

(Cleaning up Incorrectly labeled data) 监督学习问题的数据由输入 x 和输出标签 y 构成，如果观察数据发现有些输出标签 y 是错的，你的数据有些标签是错的，是否值得花时间去修正这些标签呢？在猫分类问题中，图片是猫 ($y = 1$)；不是猫 ($y = 0$)。假设看了一些数据样本，发现倒数第二张图片其实不是猫，所以这是标记错误的样本，它表示学习算法输出了错误的 y 值。对于标记错误的样本，就是在训练集或测试集 y 的标签，人类给这部分数据加的标签，实际上是错的，这实际上是一只狗， y 其实应该是 0，也许做标记的人疏忽了。如果发现数据有一些标记错误的样本，此时该怎么办？

Incorrectly labeled examples



首先，我们来考虑训练集，事实证明，深度学习算法对于训练集中的随机错误是相当健壮的 (**robust**)。只要标记出错的样本离随机错误不太远，有时可能做标记的人没有注意或者不小心，按错键了，此时不管这些错误也没问题，而没必要花太多时间修复它们。当然检查一下这些标签，并修正它们也没什么害处，有时候修正这些错误是有价值的，有时候放着不管也可以，只要总数据集总足够大，实际错误率可能不会太高。要注意的是，深度学习算法对随机误差很健壮，但对系统性的错误就没那么健壮了。比如说，如果做标记的人一直把白色的狗标记成猫，那就有问题了。因为分类器学习后会把所有白色的狗都分类为猫。

DL algorithms are quite robust to random errors in the training set.

Systematic errors

现在讨论如果是开发集和测试集中有标记出错的样本呢？如果担心开发集或测试集上标记出错的样本带来的影响，一般做法是在错误分析时，添加一个额外的列来统计标签 $y = 1$ 错误的样本数。比如说，统计一下对 100 个标记出错的样本的影响，其中分类器的输出和开发集的标签不一致，有时对于其中的少数样本，分类器输出和标签不同是因为标签错了，而不是分类器出错。也许在这个样本中，标记的人漏了背景里的一只猫，所以那里打个勾，来表示第 98 个样本标签出错了。也许这张图实际上是猫的画，而不是一只真正的猫，此时我们希望标记数据的人将它标记为 $y = 0$ ，而不是 $y = 1$ ，然后再在那里打个勾。当你统计出其他错误类型的百分比后，还可以统计因为标签错误所占的百分比，开发集里的 y 值是错的，这就解释了为什么学习算法做出和数据集里的标记不一样的预测 1。

现在问题是，是否值得修正这 6% 标记出错的样本呢？如果这些标记错误严重影响了在开发集上评估算法的能力，那么就应该花时间修正错误的标签。但是，如果它们没有严重影响用到用开发集评估成本偏差的能力，那么就没必要花宝贵的时间去处理。

我们常常看 3 个数字来确定是否值得去人工修正标记出错的数据，看看整体的开发集错误率。我们假设我们的系统达到了 90%整体准确度，10%错误率，那么应该看看错误标记引起的错误的数量或者百分比。在这种情况下，6%的错误来自标记出错，10%的 6%是 0.6%，在这种情况下，有 9.4%错误率需要首先集中精力修正，而标记出错只是总体错误的一小部分而已，没必要花时间先行修正，当然如果人手足够时间富裕的话修正也没问题。

Error analysis

Image	Dog	Great Cat	Blurry	Incorrectly labeled	Comments
...					
98				✓	Labeler missed cat in background
99		✓			
100				✓	Drawing of a cat; Not a real cat.
% of total	8%	43%	61%	6%	

现在假设在学习问题上取得了很大进展，错误率不再是 10%了，而降到了 2%，但总体错误中的 0.6%还是标记出错导致的。现在如果检查一组标记出错的开发集图片，开发集数据有 2%标记错误了，那么其中很大一部分（0.6%除以 2% = 30%标签）。这么多错误样本是因为标记出错导致的，此时修正开发集里的错误标签更有价值。

Error analysis

Image	Dog	Great Cat	Blurry	Incorrectly labeled	Comments
...					
98				✓	Labeler missed cat in background
99		✓			
100				✓	Drawing of a cat; Not a real cat.
% of total	8%	43%	61%	6%	

Overall dev set error	10%				2%
Errors due incorrect labels	0.6%				0.6%
Errors due to other causes	9.4%				1.4%

开发集的主要目的是，从两个分类器A和B中选择一个更好的。当测试两个分类器A和B时，在开发集上一个有 2.1%错误率，另一个有 1.9%错误率，现在不能再轻易信任开发集了，因为它无法说明哪个分类器更好，因为 0.6%的错误率是开发集中标记出错导致的。此时应该修正开发集里的错误标签，因为标记出错对算法错误的整体评估标准有严重的影响。

现在如果决定要去修正开发集数据，手动重新检查标签，并尝试修正一些标签，这里还有一些额外的方针和原则需要考虑。首先，不管用什么修正手段，都要同时作用到开发集和测试集上，因为开发和测试集必须来自相同的分布。开发集确定了目标，当击中目标后，我们希望算法能够推广到测试集上，这样才能更高效地在来自同一分布的开发集和测试集上迭代。换句话说，如果打算修正开发集上的部分数据，最好也对测试集做同样的修正以确保它们继续来自相同的分布。

Correcting incorrect dev/test set examples

- Apply same process to your dev and test sets to make sure they continue to come from the same distribution
- Consider examining examples your algorithm got right as well as ones it got wrong.
- Train and dev/test data may now come from slightly different distributions.

其次，要同时检验算法判断正确和判断错误的样本，要检查算法出错的样本很容易，只需要看看那些样本是否需要修正，但还有可能有些样本算法判断正确，那些也需要修正。如果只修正算法出错的样本，对算法的偏差估计可能会变大，这会让算法有一点不公平的优势，我们就需要再次检查出错的样本，但也需要再次检查做对的样本，因为算法有可能因为运气好把某个东西判断对了。在那个特例里，修正那些标签可能会让算法从判断对变成判断错。这第二点不是很容易做，通常也不会这么做。原因是，如果你的分类器很准确，那么判断错的次数比判断正确的次数要少得多，2%出错，98%正确，更容易检查 2%数据上的标签，然而检查 98%数据上的标签要花的时间长得多，通常不这么做，但也是要考虑到。

- Train and dev/test data may now come from slightly different distributions.

最后，如果进入到一个开发集和测试集去修正部分标签，你可能会也可能不会去对训练集做同样的事情，我们讨论过，修正训练集中的标签其实相对没那么重要，只修正开发集和测试集中的标签即可，因为它们通常比训练集小得多，我们不想把所有额外的精力投入到修正大得多的训练集中的标签。

最后与大家分享几点建议：

首先，深度学习研究人员有时会喜欢这样说：“我只是把数据提供给算法，我训练过了，效果拔群”。更多时候，我们把数据喂给算法，然后训练它，并减少人工干预，减少使用人类的见解。但是，在构造实际系统时，通常需要更多的人工错误分析，更多的人类见解来架构这些系统，尽管深度学习的研究人员不愿意承认这点。

其次，不知道为什么，我看一些工程师和研究人员不愿意亲自去看这些样本，也许做这些事情很无聊，坐下来看 100 或几百个样本来统计错误数量，但我经常亲自这么做。当我带领一个机器学习团队时，我想知道它所犯的的错误，我会亲自去看看这些数据，尝试和一部分错误作斗争。我想就因为花了这几分钟，或者几个小时去亲自统计数据，真的可以找到需要优先处理的任务，我发现花时间亲自检查数据非常值得，所以我强烈建议，如果你在搭建你的机器学习系统的话，然后你想确定应该优先尝试哪些想法，或者哪些方向。

2.3 快速搭建你的第一个系统，并进行迭代

(Build your first system quickly, then iterate) 如果正在开发全新的机器学习应用，我通常会提出这样的建议，尽快建立第一个系统原型，然后快速迭代，让我告诉你我的意思吧。如果你正在考虑建立一个新的语音识别系统，其实可以走很多方向，可以优先考虑很多事情。比如，有一些特定的技术，可以让语音识别系统对嘈杂的背景更加健壮，嘈杂的背景可能是说咖啡店的噪音，背景里有很多人在聊天，或者车辆的噪音，高速上汽车的噪音或者其他类型的噪音。有一些方法可以让语音识别系统在处理带口音时更健壮，还有特定的问题和麦克风与说话人距离很远有关，就是所谓的远场语音识别。儿童的语音识别带来特殊的挑战，主要是单词发音方面，还有他们选用的词汇。还有比如说，说话人口吃，或者说了很多无意义的短语，比如“哦”，“啊”之类的。总而言之，可以做很多事情来改进语音识别系统。

Speech recognition example



- | | |
|--|---|
| <ul style="list-style-type: none"> → • Noisy background <ul style="list-style-type: none"> → • Café noise → • Car noise → • Accented speech → • Far from microphone → • Young children's speech → • Stuttering <i>uh, ah, um,...</i> → • ... | <ul style="list-style-type: none"> → • Set up dev/test set and metric • Build initial <u>system quickly</u> • Use <u>Bias/Variance analysis & Error analysis</u> to prioritize next steps. |
|--|---|

一般来说，对于几乎所有的机器学习程序可能会有 50 个不同的方向可以前进，并且每个方向都是相对合理的可以改善系统。但挑战在于，如何选择一个方向集中精力处理。所以如果想搭建全新的机器学习程序，就要快速搭好第一个系统，然后开始迭代。首先，快速设立开发集和测试集还有指标，这样就决定了目标所在，如果目标定错了，之后改也是可以的。但一定要设立某个目标。之后，马上搭好一个机器学习系统原型，找到训练集，训练一下，看看效果，开始理解算法表现如何，在开发集测试集评估指标表现如何。当建立第一个系统后，就可以马上用到之前说的偏差方差分析，错误分析，来确定下一步优先做什么。特别是如果错误分析证明大部分的错误来源是说话人远离麦克风，这对语音识别构成特殊挑战，那么就应集中精力研究这些技术，远场语音识别技术，处理说话人离麦克风很远的情况。

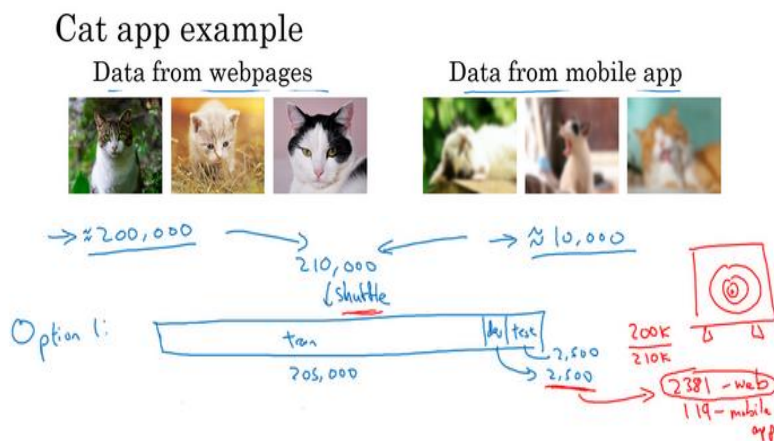
建立这个初始系统的所有意义在于，它可以是一个快速和粗糙的实现 (**quick and dirty implementation**)，有一个学习过的系统，有一个训练过的系统，我们确定偏差方差的范围，就可以知道下一步应该优先做什么，也能够进行错误分析，可以观察一些错误，然后想出所有能走的方向，哪些是实际上最有希望的方向。但是，如果尝试搭建一个人脸识别设备，那么可以从现有大量学术文献为基础出发（因为人脸识别有很多学术文献），一开始就搭建比较复杂的系统。但如果第一次处理某个新问题，建议不要把第一个系统弄得太复杂，反而应该构建一些快速而粗糙的系统，然后来找到改善系统要优先处理的方向。

2.4 使用来自不同分布的数据进行训练和测试

(Training and testing on different distributions) 深度学习对训练数据的胃口很大，收集到足够多带标签的数据构成训练集，算法效果最好，这导致很多团队用尽一切办法收集数据。在深度学习时代，越来越多的团队都用来自开发和测试集分布不同的数据来训练，这里有一些微妙的地方，下面介绍一些最佳做法来处理训练集和测试集存在差异的情况。

假设在开发一个手机应用，应用要识别用户上传的图片是不是猫。现有两个数据来源，一个是真正关心的数据分布，来自用户上传的数据，这些照片一般业余，取景不太好，有些甚至很模糊；另一个数据来源可以用爬虫程序挖掘网页直接下载，这些照片取景专业、高分辨率、拍摄专业的猫图片。现在的困境是：应用的用户数还不多，只收集到 10,000 张用户上传的照片，通过爬虫挖掘网页可以下载到海量（20 万张）猫图，这又不是我们真正关心的数据分布，此时要怎么做呢？

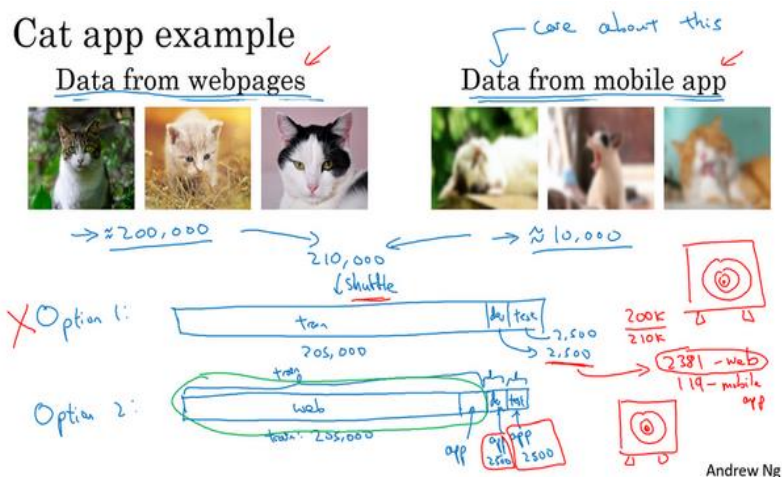
做法 1：将两组数据合并在一起，这样就有 21 万张照片，之后把这 21 万张照片随机分配到训练、开发和测试集中，假设开发集和测试集各包含 2500 个样本，训练集有 20,5000 个样本。如此设立数据集有一些好处，也有坏处。好处在于，训练集、开发集和测试集都来自同一分布，这样更好管理。但坏处在于，开发集的 2500 个样本中，只有 119 张图来自手机上传，很多图片（数学期望值是： $2500 \times \frac{200k}{210k} = 2381$ ）都来自网页下载，可那并不是我们真正关心的数据分布，真正要处理的是来自手机的照片。



要记住，设立开发集的目的是告诉团队去瞄准的目标，而瞄准目标的方式，大部分精力都用在优化网页下载的图片，这其实不是我们想要的。所以不建议使用做法 1，因为这样设立开发集就是告诉团队，针对不同于实际关心的数据分布去优化，所以不要这么做。

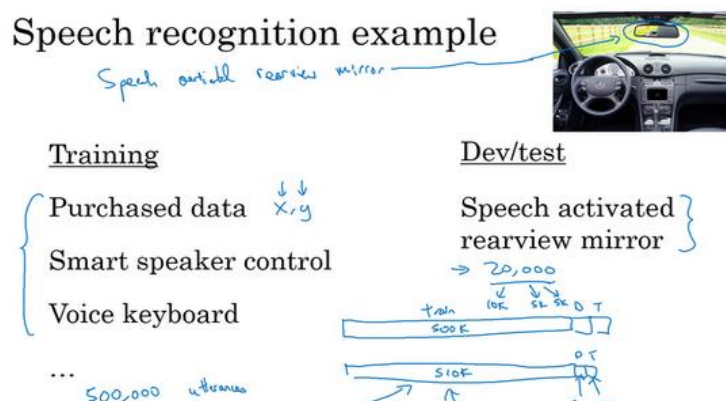
做法 2：训练集还是 205,000 张图片，其中 200,000 张图片是来自网页下载，5000 张来自手机上传的图片。对于开发集和测试集都是来自手机图，开发集是 2500 张来自应用的图片，测试集也是 2500 张来自应用的图片。这样将数据分成训练集、开发集和测试集的好处在于，现在瞄准的目标就是想要处理的目标，开发集包含的数据全部来自手机上传，这是真

正关心的图片分布。我们试试搭建一个学习系统，让系统在处理手机上传图片分布时效果良好。缺点在于，现在训练集分布和开发集、测试集分布并不一样。但事实证明，这样把数据分成训练、开发和测试集，在长期能有更好的系统性能。我们以后会讨论一些特殊的技巧，可以处理训练集的分布和开发集和测试集分布不一样的情况。



再来看另一个样本，假设正在开发一个全新的产品，一个语音激活汽车后视镜。具体而言就是造一个后视镜，你可以和后视镜对话，只需要说：“亲爱的后视镜，请帮我找到最近的加油站的路线”，然后后视镜就会处理这个请求。这实际上是一个真正的产品，假设现在要研制这个产品，那么怎么收集数据去训练这个产品语言识别模块呢？

现在有很多来自其他语音识别应用的数据，它们并不是来自语音激活后视镜的数据。我们来看如何分配训练集、开发集和测试集。举例来说，对于来自其他语音识别应用的数据，也许收集了 500,000 段录音，但这些对于开发集和测试集数据集小得多，因为用户要查询导航信息或试图找到通往各个地方的路线，这个数据集可能会有很多街道地址，这个数据的分布和左边大不一样，但这真的是需要关心的数据，因为这些数据是产品必须处理好的，所以就应该把它设成开发和测试集。



此时我们可以设立训练集有 500,000 段语音，开发集和测试集包含 10,000 段语音，是从实际的语音激活后视镜收集的。或者换种方式，将 20,000 段来自语音激活后视镜的录音拿一半放在训练集里，那么训练集是 51 万段语音，包括来自那里的 50 万段语音，还有来自后视镜的 1 万段语音，然后开发集和测试集各自有 5000 段语音。

2.5 数据分布不匹配时的偏差与方差的分析

（Bias and Variance with mismatched data distributions）估计学习算法的偏差和方差真的可以确定接下来应该优先做的方向，但是，当训练集来自和开发集、测试集不同分布时，分析偏差和方差的方式可能不一样，我们来看为什么。

我们用猫分类器为例，人类在这个任务上能做到几乎完美，所以贝叶斯错误率或者说贝叶斯最优错误率几乎是 0%。要进行错误率分析，通常要看训练误差，也要看看开发集的误差。比如说，在这个样本中，训练集误差是 1%，开发集误差是 10%，如果开发集和训练集来自一样的分布，你可能会说，这里存在很大的方差问题，算法不能很好的从训练集出发泛化，它处理训练集很好，但处理开发集就突然间效果很差了。

Cat classifier example

Assume humans get $\approx 0\%$ error.

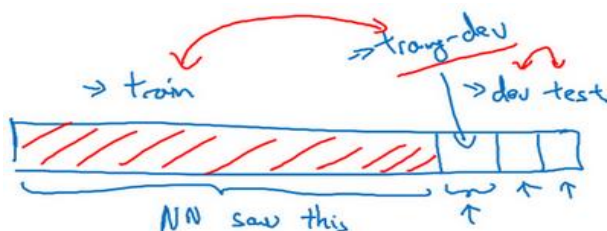
Training error 1%
Dev error 10%

如果训练数据和开发数据来自不同的分布，就不能下这个结论了。特别是算法在开发集上做得不错，可能因为训练集很容易识别，都是高分辨率图片，很清晰的图像，但开发集要难以识别得多。所以也许软件没有方差问题，这只不过反映了开发集包含更难准确分类的图片。所以这个分析的问题在于，当看训练误差，再看开发误差，有两件事变了。首先算法只见过训练集数据，没见过开发集数据。第二，开发集数据来自不同的分布，而且因为同时改变了两件事情，很难确认这增加的 9% 误差率有多少是因为算法没看到开发集中的数据导致的，这是问题方差的部分，有多少是因为开发集数据本身就是不一样。

为了分辨清楚两个因素的影响，定义一组新的数据是有意义的，我们称之为训练-开发集，所以这是一个新的数据子集。我们从训练集的分布里挖出来，但不会用来训练网络。

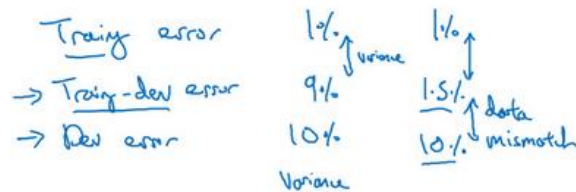
Training-dev set: Same distribution as training set, but not used for training

我们已经设立过这样的训练集、开发集和测试集了，并且开发集和测试集来自相同的分布，但训练集来自不同的分布。我们要做的是随机打散训练集，然后分出一部分训练集作为训练-开发集（training-dev），就像开发集和测试集来自同一分布，训练集、训练-开发集也来自同一分布。

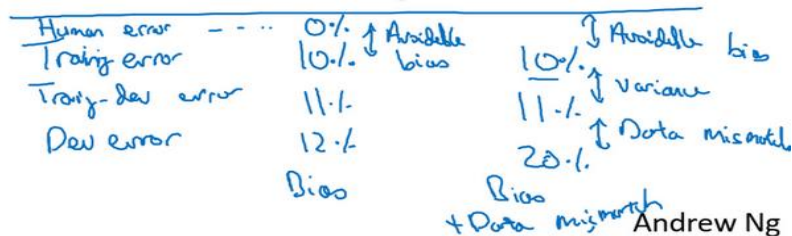


但不同的地方是，现在只在训练集训练神经网络，不会让神经网络在训练-开发集上跑后向传播。为了进行误差分析，我们首先看看分类器在训练集上的误差，其次是训练-开发集上的误差，还有开发集上的误差。比如说训练误差是 1%，训练-开发集上误差是 9%，开发集误差是 10%，和以前一样。此时可以得到结论，当从训练数据变到训练-开发集数据时，错误率真的上升了很多。而训练数据和训练-开发数据的差异在于，神经网络能看到第一部分数据并直接在上面做了训练，但没有在训练-开发集上直接训练，这就说明算法存在方差问题，因为训练-开发集和训练集来自同一分布的。

我们来看一个不同的样本，假设训练误差为 1%，训练-开发误差为 1.5%，开发集错误率为 10%。现在方差问题就很小了，因为算法从见过的训练数据转到训练-开发集数据，神经网络还没有看到的数据，错误率只上升了一点点。但转到开发集时，错误率就上升了，这是数据不匹配的问题。因为学习算法没有直接在训练-开发集或者开发集训练过，但是这两个数据集来自不同的分布。但不管算法在学习什么，它在训练-开发集上做的很好，但开发集上做的不好，我们称之为数据不匹配的问题。



我们再来看几个样本，假设训练误差是 10%，训练-开发误差是 11%，开发误差为 12%，要记住，人类水平对贝叶斯错误率的估计大概是 0%，如果得到了这种等级的表现，那就真的存在偏差问题了，因为算法做的比人类水平差很多。



最后一个例子，如果训练集错误率是 10%，训练-开发错误率是 11%，开发错误率是 20%，这其实有两个问题。第一，可避免偏差很大，人类能做到接近 0% 错误率，但算法在训练集上错误率为 10%。这里方差似乎很小，但数据不匹配问题很大。

总结一下，我们要看的关键数据是人类水平错误率，训练集错误率，训练-开发集错误率（分布和训练集一样，但没有直接在上面训练）。根据这些错误率之间差距有多大，可以大概知道，可避免偏差、方差、数据不匹配的问题各自有多大。

2.6 处理数据不匹配问题

（Addressing data mismatch）如果训练集来自和开发测试集不同的分布，如果错误分析显示有一个数据不匹配问题该怎么办？这个问题没有完全系统的解决方案，但可以尝试一些事情。为了避免对测试集过拟合，要做错误分析，应该人工去看开发集而不是测试集。

Addressing data mismatch

- • Carry out manual error analysis to try to understand difference between training and dev/test sets

E.g. noisy - car noise street numbers

- • Make training data more similar; or collect more data similar to dev/test sets

E.g. Simulate noisy in-car data

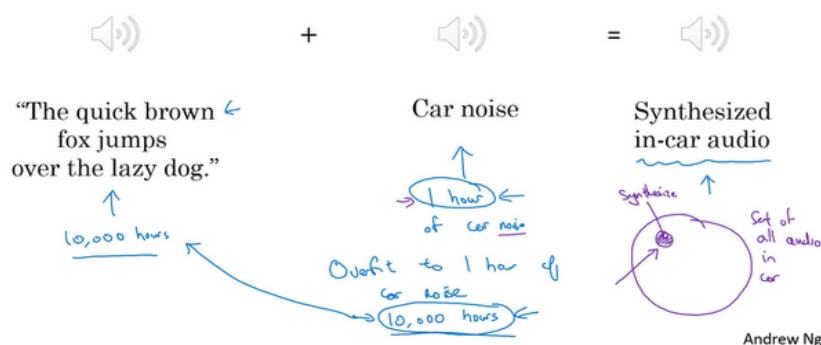
作为一个具体的例子，如果开发一个语音激活的后视镜应用，可能要听一下来自开发集的样本，尝试弄清楚开发集和训练集到底有什么不同。比如说，可能会发现很多开发集样本噪音很多，有很多汽车噪音，这是开发集和训练集差异之一；也许还会发现其他错误，车子语言激活后视镜，它可能经常识别错误街道号码，因为那里有很多导航请求都有街道地址，所以得到正确的街道号码很重要。当了解开发集误差的性质时，开发集有可能跟训练集不同或者更难识别，那么可以尝试把训练数据变得更像开发集，收集更多类似开发集和测试集的数据。如果发现车辆背景噪音是主要的错误来源，那么可以模拟车辆噪声数据。如果发现很难识别街道号码，可以收集更多人们说数字的音频数据，加到训练集里。

具体而言，可以利用的一种技术是人工合成数据（**artificial data synthesis**）。在解决汽车噪音问题的场合，要建立语音识别系统。也许实际上没那么多在汽车背景噪音下录得的音频，或者在高速公路背景噪音下录的音频，但可以人工合成。比如我们有大量清晰的音频，不带车辆背景噪音，比如“The quick brown fox jumps over the lazy dog”，这可能是训练集里的一段音频，这个句子在 AI 测试中经常使用，因为这个短句包含了从 a 到 z 所有字母。之后，也可以收集一段汽车噪音，再把两个音频片段放到一起，就合成出“the quick brown fox jumps over the lazy dog”（带有汽车噪声），这是一个相对简单的音频合成例子。

通过人工数据合成，可以快速制造更多的训练数据，就像真的在车里录的那样，那就不需要花时间实际出去收集数据，比如说在实际行驶中的车子，录下上万小时的音频。所以，如果错误分析显示应该尝试让数据听起来更像在车里录的，那么人工合成那种音频，然后喂给机器学习算法，这样做是合理的。

人工数据合成有一个潜在问题，比如说，在安静的背景里录 10,000 小时音频，然后只录了一小时车辆背景噪音，将这 1 小时汽车噪音回放 10,000 次，并叠加到安静的背景下录的 10,000 小时音频。如果这么做了，人听起来这个音频没什么问题。但是有一个风险，有可能学习算法对这 1 小时汽车噪音过拟合，因为只模拟了全部数据空间的一小部分。

Artificial data synthesis



对于人耳来说，这些音频听起来没什么问题，因为一小时的车辆噪音对人耳来说，听起来和其他任意一小时车辆噪音是一样的。但有可能从这整个空间很小的一个子集出发合成数据，神经网络最后可能对这一小时汽车噪音过拟合。我不知道以较低成本收集 10,000 小时的汽车噪音是否可行，这样可以不用一遍又一遍地回放那 1 小时汽车噪音，这样就有 10,000 个小时永不重复的汽车噪音来叠加到 10,000 小时安静背景下录得的永不重复的语音。这是可以做的，但不保证能做。但是使用 10,000 小时永不重复的汽车噪音，而不是 1 小时重复学习，算法有可能取得更好的性能。

Car recognition:



这里有人工合成数据的另一个例子，假设在研发无人驾驶汽车，我们希望检测出这样的车，然后用这样的框包住它。很多人都讨论过的一个思路是，为什么不用计算机合成图像来模拟成千上万的车辆呢？事实上，这里有几张车辆照片（下图后两张图片），其实是用计算机合成的，我想这个合成是相当逼真的，通过这样合成图片可以训练出一个相当不错的计算机视觉系统来检测车子。

Artificial data synthesis

Car recognition:



Andrew Ng

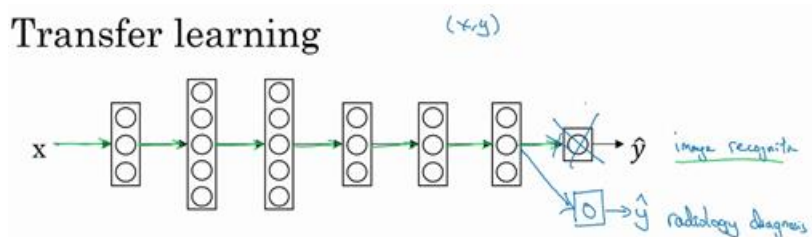
不幸的是，过拟合情况也会在这里出现，比如这是所有车的集合，如果只合成这些车中很小的子集，对于人眼来说也许这样合成图像没什么问题，但学习算法可能会对合成的这一个小子集过拟合。比如我们找到一个电脑游戏，里面车辆渲染的画面很逼真，那么就可以截图，得到数量巨大的汽车图片数据集。如果这个游戏只有 20 辆独立的车辆，那么这游戏看起来还行，但现实世界里车辆的设计可不只 20 种，如果用这 20 量独特的车合成的照片去训练系统，那么神经网络很可能对这 20 辆车过拟合，但人类很难分辨出来。即使这些图像看起来很逼真，可能真的只用了所有可能出现的车辆的很小的子集。

总而言之，如果你认为存在数据不匹配问题，我建议做错误分析，或者看看训练集和开发集，试图找出这两个数据分布到底有什么不同，然后看看是否有办法收集更多看起来像开发集的数据作训练。我们谈到其中一种办法是人工数据合成，人工数据合成确实有效。我们已经看到人工数据合成显著提升了已经非常好的语音识别系统的表现，所以这是可行的。但当使用人工数据合成时，一定要谨慎，要记住有可能从所有可能性的空间只选了很小一部分去模拟数据。

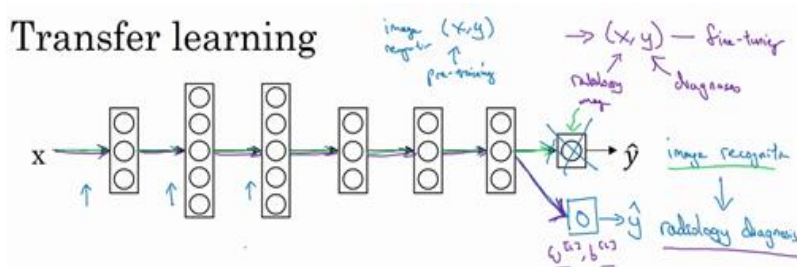
2.7 迁移学习

深度学习中，最强大的理念之一就是，有时神经网络可以从一个任务中习得知识，并将这些知识应用到另一个独立的任务中。例如，已经训练好一个识别猫的神经网络，然后使用部分习得的知识去更好地阅读 x 射线扫描图，这就是迁移学习（**Transfer learning**）。

假设已经训练好一个图像识别的神经网络，并在 (x, y) 对上训练，其中 x 是图像， y 是某些对象，图像是猫、狗、鸟或其他东西。如果把这个神经网络拿来，然后让它适应或者说迁移，在不同任务中学到的知识，比如放射科诊断（**radiodiagnosis**），用于阅读 X 射线扫描图。我们可以做的是把神经网络最后的输出层拿走，也删掉进入到最后一层的权重，然后为最后一层重新赋予随机权重，然后让它在放射诊断数据上训练。

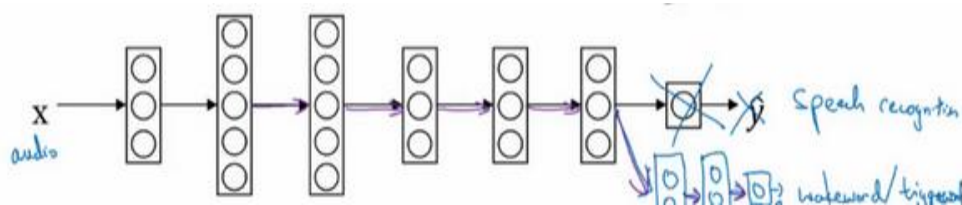


具体来说，当进行图像识别任务训练时，可以训练神经网络的所有参数，权重，层，然后得到一个能够做图像识别预测的网络。之后要实现迁移学习，就把数据集换成新的 (x, y) 对，现在变成放射科图像， y 是想要预测的诊断，而我们要做的就是初始化最后一层的权重，称为 $w^{[L]}$ 和 $b^{[L]}$ 随机初始化。现在，我们在这个新的放射科数据集上重新训练网络，如果放射科数据集很小，可能只需要重新训练最后一层的权重，就是 $w^{[L]}$ 和 $b^{[L]}$ ，并保持其他参数不变。如果数据足够多，可以重新训练神经网络中剩下的所有层的参数，此时在图像识别数据的初期训练阶段，称为**预训练（pre-training）**，因为用图像识别数据去预先初始化，或者预训练神经网络的权重。然后，更新所有权重，在放射科数据上训练，这个过程叫**微调（fine tuning）**。

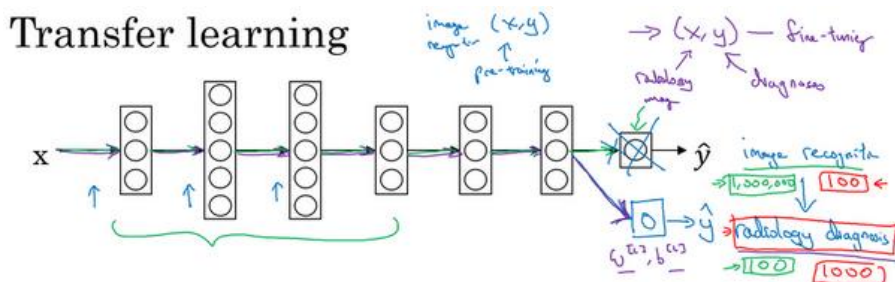


把图像识别中学到的知识迁移到放射科诊断，这么做为什么有效果呢？因为有很多低层次特征，比如说边缘检测、曲线检测、阳性对象检测（**positive objects**），从非常大的图像识别数据库中习得这些能力可能有助于算法在放射科诊断中做得更好，算法学到了很多结构信息，图像形状信息，其中一些知识可能会很有用，所以学会了图像识别，它就可能学到足够多的信息，可以了解不同图像的组成部分是怎样的，这些知识有可能帮助放射科诊断网络学习更快一些，或者需要更少的学习数据。

这里是另一个例子，假设已经训练出一个语音识别系统， x 是音频输入， y 是听写文本，此语音识别系统可以输出听写文本。现在如果想搭建一个“唤醒词”或“触发词”检测系统，所谓唤醒词或触发词就是我们说的一句话，可以唤醒家里的语音控制设备，比如说“Alexa”可以唤醒一个亚马逊 Echo 设备，或说“OK Google”来唤醒 Google 设备，说“Hey Siri”来唤醒苹果设备，说“你好百度”唤醒一个百度设备。要做到这点，可能需要去掉神经网络的最后一层，然后加入新的输出节点，但有时不只加入一个新节点，而是加入几个新层，然后把唤醒词检测问题的标签 y 喂进去训练。我们可能只需要重新训练网络的新层，也许重新训练神经网络中更多的层，这取决于有多少数据。



迁移学习什么时候是有意义的呢？迁移学习起作用的场合是，在迁移来源问题中有很多数据，但迁移目标问题没有那么多数据。例如，假设图像识别任务中有 1 百万个样本，数据相当多，可以学习低层次特征，可以在神经网络的前面几层学到如何识别很多有用的特征。但是对于放射科任务，也许只有一百个样本，数据很少，所以从图像识别训练中学到的很多知识可以迁移，并且真正加强放射科识别任务的性能，即使放射科数据很少。



从数据量很多的问题迁移到数据量相对小的问题，这是有意义的。但反过来的话，迁移学习可能就没有意义了。比如，用 100 张图训练图像识别系统，有 100 甚至 1000 张图用于训练放射科诊断系统，100 甚至 1000 张图用于训练放射科诊断系统的每个样本价值比 100 张图训练图像识别系统要大得多，至少就建立性能良好的放射科系统而言是这样。

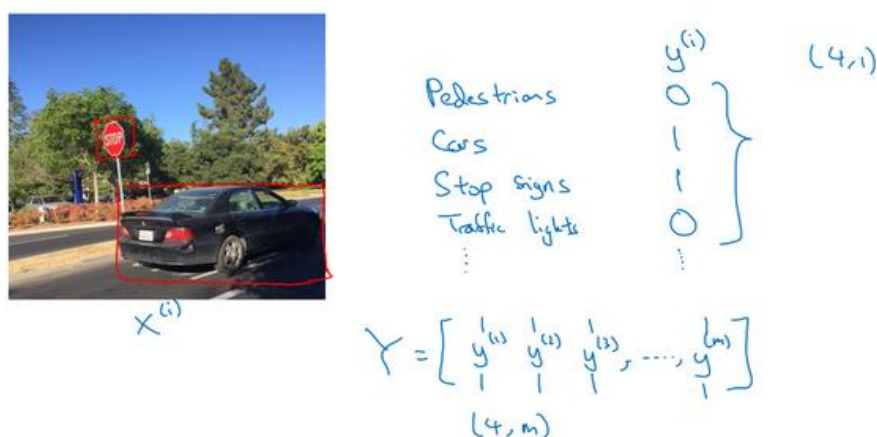
总结一下，什么时候迁移学习是有意义的？如果想从任务A学习并迁移一些知识到任务B，那么当任务A和任务B都有同样的输入 x 时，迁移学习是有意义的。在第一个例子中，A和B的输入都是图像，在第二个例子中，两者输入都是音频。我们希望提高任务B的性能，因为任务B每个数据更有价值，对任务B来说通常任务A的数据量必须大得多，才有帮助，因为任务A里单个样本的价值没有比任务B单个样本价值大，学习完任务A的低层次特征，就可以帮助任务B的学习，那迁移学习更有意义一些。

2.8 多任务学习

迁移学习中的步骤是串行的，从任务A里学习然后迁移到任务B。在多任务学习（Multi-task learning）中，可以同时开始学习，试图让单个神经网络同时做几件事情，希望这里每个任务都能帮到其他所有任务。

我们来看一个例子，假设研发无人驾驶车辆，那么无人驾驶车可能需要同时检测不同的物体，比如检测行人、车辆、停车标志，还有交通灯等各种其他东西。比如在左边这个例子中，图像里有停车标志，有辆车，但没有行人，也没有交通灯。

Simplified autonomous driving example

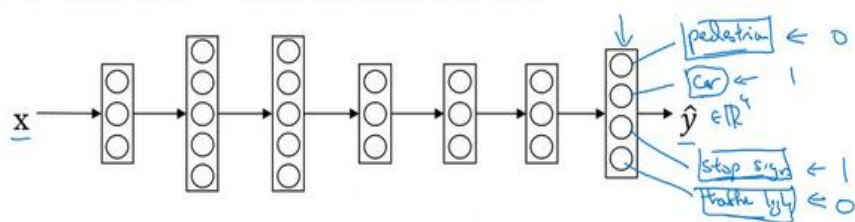


如果这是输入图像 $x^{(i)}$ ，那么这里不再是一个标签 $y^{(i)}$ ，而是有 4 个标签。在这个例子中，没有行人，有一辆车，有一个停车标志，没有交通灯。如果尝试检测其他物体，也许 $y^{(i)}$ 的维数会更高，现在我们就先用 4 个，所以 $y^{(i)}$ 是维度 4×1 的向量。如果从整体来看这个训练集标签和以前类似，我们将训练集的标签水平堆叠起来，像这样 $y^{(1)}$ 一直到 $y^{(m)}$ ：

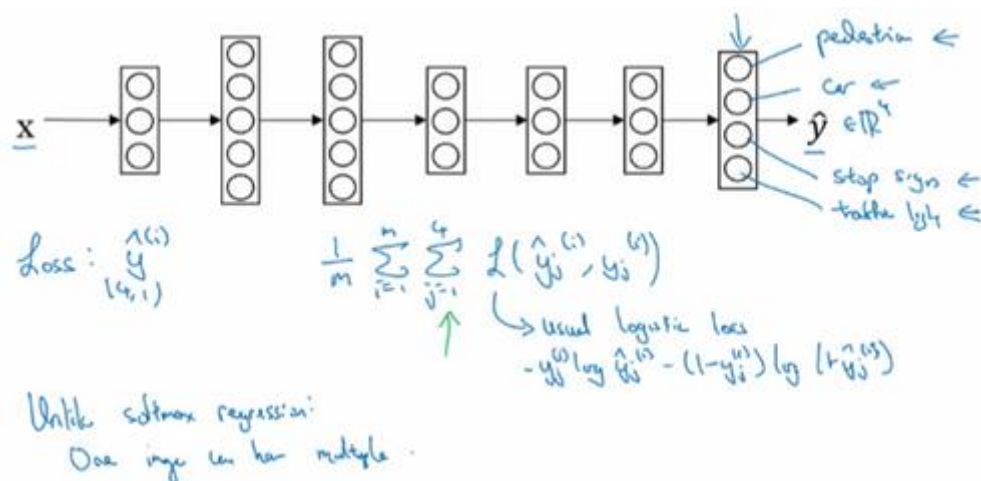
$$Y = \begin{bmatrix} | & | & | & \dots & | \\ y^{(1)} & y^{(2)} & y^{(3)} & \dots & y^{(m)} \\ | & | & | & \dots & | \end{bmatrix}$$

不过现在 $y^{(i)}$ 是 4×1 向量，所以这些都是竖向的列向量，这个矩阵 Y 现在变成 $4 \times m$ 矩阵。而之前，当 y 是单实数时，这就是 $1 \times m$ 矩阵。

Neural network architecture



那么现在可以做的是训练一个神经网络，来预测 y 值。输入 x 到神经网络，输出是一个四维向量 y 。请注意，这里的四个节点，第一个节点是预测图中有没有行人，第二个输出节点预测有没有车，第三个预测有没有停车标志，第四个预测有没有交通灯。



要训练这个神经网络，现在需要定义神经网络的损失函数，对于一个输出 \hat{y} ，是个4维向量，对于整个训练集的平均损失：

$$\frac{1}{m} \sum_{i=1}^m \sum_{j=1}^4 L(\hat{y}_j^{(i)}, y_j^{(i)})$$

$\sum_{j=1}^4 L(\hat{y}_j^{(i)}, y_j^{(i)})$ 这些单个样本预测的损失，所以这就是对四个分量的求和，行人、车、停车标志、交通灯，而这个标志 L 指的是 **logistic 损失**，我们就这么写：

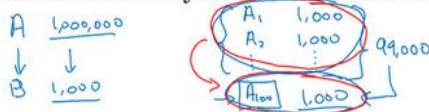
$$L(\hat{y}_j^{(i)}, y_j^{(i)}) = -y_j^{(i)} \log \hat{y}_j^{(i)} - (1 - y_j^{(i)}) \log (1 - \hat{y}_j^{(i)})$$

整个训练集的平均损失和之前分类猫的例子主要区别在于，现在要对 $j = 1$ 到4求和，这与 **softmax** 回归的主要区别在于，与 **softmax** 回归不同，**softmax** 将单个标签分配给单个样本，比如A属于第一类，B属于第二类，C属于第三类。而这张图可以有多个标签，不是说每张图都只是一张行人图片，汽车图片、停车标志图片或者交通灯图片。所以不是只给图片一个标签，而是需要遍历不同类型，然后看看每个类型，那类物体有没有出现在图中。

如果训练了一个神经网络，试图最小化这个成本函数，这就是多任务学习。因为现在是建立单个神经网络，观察每张图，然后解决四个问题，系统会辨别出每张图里有没有这四个物体。另外，我们也可以训练四个不同的神经网络，而不是训练一个网络做四件事情。但神经网络一些早期特征，在识别不同物体时都会用到，然后你发现，训练一个神经网络做四件事情比训练四个独立的神经网络分别做四件事性能更好，这就是多任务学习的力量。

When multi-task learning makes sense

- Training on a set of tasks that could benefit from having shared lower-level features.
- Usually: Amount of data you have for each task is quite similar.



- Can train a big enough neural network to do well on all the tasks.

那么多任务学习什么时候有意义呢？当三件事为真时，它就是有意义的。

第一，如果训练的一组任务，可以共用低层次特征。对于无人驾驶的例子，同时识别交通灯、汽车和行人是有道理的，这些物体有相似的特征，也许能帮你识别停车标志，因为这些都是道路上的特征。

第二，在多任务学习中，通常有更多任务而不仅仅是两个，比如说要完成 100 个任务，尝试同时识别 100 种不同类型的物体。我们可能会发现，每个任务大概有 1000 个样本，所以如果专注加强单个任务的性能，比如我们专注加强第 100 个任务的表现，如果试图单独去做这个最后的任务，但我们只有 1000 个样本去训练这个任务，只有 1000 个样本的训练集，效果可能会很差。而通过在其他 99 项任务的训练，这些加起来可以一共有 99000 个样本，这可能大幅提升算法性能，可以提供很多知识来增强这个任务的性能。如果有对称性，这其他 99 个任务，也许能提供一些数据或提供一些知识来帮到这 100 个任务中的每一个任务。第二点不是绝对正确的，但通常会看的是如果专注于单项任务，想要从多任务学习得到很大性能提升，那么其他任务加起来必须要有比单个任务大得多的数据量。

最后，多任务学习在以下场合更有意义，当我们可以训练一个足够大的神经网络，同时做好所有的工作。多任务学习的替代方法是为每个任务训练一个单独的神经网络，可以训练一个用于行人检测的神经网络，一个用于汽车检测的神经网络，一个用于停车标志检测的神经网络和一个用于交通信号灯检测的神经网络。那么研究员 **Rich Carona** 几年前发现的是什么呢？多任务学习会降低性能的唯一情况，和训练单个神经网络相比性能更低的情况就是神经网络还不够大。如果可以训练一个足够大的神经网络，那么多任务学习肯定不会或者很少会降低性能。

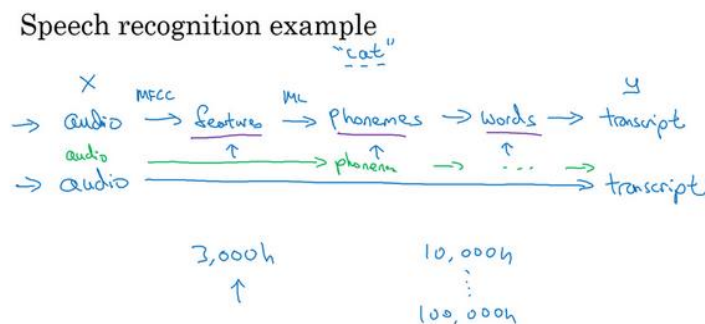
总结一下，多任务学习就是训练一个神经网络来执行许多任务，这可以有更高的性能，比单独完成各个任务更高的性能。但要注意，实际上迁移学习比多任务学习使用频率更高，因为很难找到那么多相似且数据量对等的任务可以用单一神经网络训练。如果想解决一个机器学习问题，但数据集相对较小，那么迁移学习真的能帮到你，找到一个相关问题，其中数据量要大得多，就能以它为基础训练神经网络，然后迁移到这个数据量很少的任务上。

2.9 什么是端到端的深度学习？

深度学习中最令人振奋的最新动态之一就是端到端深度学习（What is end-to-end deep learning?）的兴起，那么端到端学习到底是什么呢？简而言之，以前有一些数据处理系统或者学习系统需要多个阶段的处理。端到端深度学习就是忽略所有这些不同的阶段，用单个神经网络代替它。

我们以语音识别为例，目标是输入 x ，比如说一段音频，然后把它映射到一个输出 y ，就是这段音频的听写文本。传统上，语音识别需要很多阶段。首先会提取一些特征，一些手工设计的音频特征，在提取出一些低层次特征之后，可以应用机器学习算法在音频片段中找到音位，音位是声音的基本单位，比如说“Cat”这个词是三个音节构成的，算法就把这三个音位提取出来，然后将音位串在一起构成独立的词，再将词串起来构成音频片段的听写文本。

What is end-to-end learning?



这种有很多阶段的流水线相比，端到端深度学习做的是，训练一个巨大的神经网络，输入是一段音频，输出直接是听写文本。**AI** 的其中一个有趣的社会学效应是，随着端到端深度学习系统表现开始更好，有一些花了大量时间或者整个事业生涯设计出流水线各个步骤的研究员，都投入到开发这个流水线的功能上去了。而端到端深度学习只需要把训练集拿过来，直接学到了 x 和 y 之间的函数映射，直接绕过了其中很多步骤。对一些学科里的人来说，这点相当难以接受，他们无法接受这样构建 **AI** 系统，因为有些情况，端到端方法完全取代了旧系统，某些投入了多年研究的中间组件也许已经过时了。

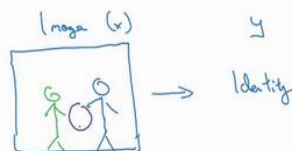
事实证明，端到端深度学习的挑战之一是，可能需要大量数据才能让系统表现良好，比如，只有 3000 小时数据去训练语音识别系统，那么传统的流水线效果真的很好。但当拥有非常大的数据集时，比如 10,000 或者 100,000 小时数据，这样端到端方法突然开始很厉害了，只有大数据集才能让端到端方法真正发出耀眼光芒。如果你的数据量适中，那么也可以用中间件方法，输入还是音频，然后绕过特征提取，直接尝试从神经网络输出音位，然后也可以在其他阶段用，所以这是往端到端学习迈出的小小一步，但还没有到那里。

这张图上是百度的林元庆研究员做的，一个相机会拍下接近门禁的人，如果它认出了那个人，门禁系统就自动打开，让他通过，这样就不需要刷一个 **RFID** 工卡就能进入这个设施，系统已部署在越来越多的中国办公室。

Face recognition



[Image courtesy of Baidu]

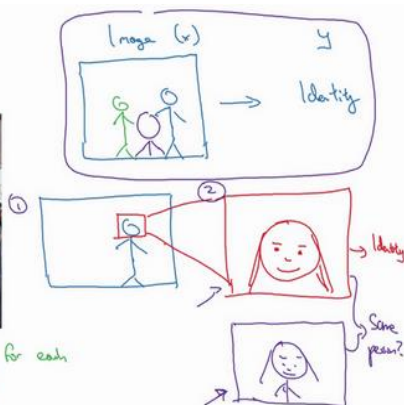


那么，怎么搭建这样的系统呢？第一件事是，看看相机拍到的照片，但也许这是相机照片，有人接近门禁了，这可能是相机拍到的图像 x 。我们可以尝试直接学习图像 x 到人物 y 身份的函数映射，事实证明这不是最好的方法。其中一个问题是，人可以从很多不同的角度接近门禁，他们可能在绿色位置，可能在蓝色位置。有时他们更靠近相机，所以他们看起来更大，有时候他们非常接近相机，那照片中脸就很大了。在实际研制这些门禁系统时，它不是直接将原始照片喂到一个神经网络，试图找出一个人的身份。相反，迄今为止最好的方法似乎是一个多步方法，首先，运行一个软件来检测人脸，第一个检测器找的是人脸位置，检测到人脸，然后放大图像的那部分，并裁剪图像，使人脸居中显示，然后就是这里红线框起来的照片，再喂到神经网络里，让网络去学习那人的身份。

Face recognition



[Image courtesy of Baidu]



研究人员发现，比起一步学习，把这个问题分解成两个更简单的步骤可以得到更好的表现。首先，是弄清楚脸在哪里。第二步是看着脸，弄清楚这是谁。训练第二步的方式，训练网络的方式就是输入两张图片，然后将输入的两张图比较一下，判断是否是同一个人。比如记录了 10,000 个员工 ID，你可以把红色框起来的图像快速比较，看看这张红线内的照片，是不是那 10000 个员工之一，来判断是否应该允许其进入这个设施或者进入这个办公楼。

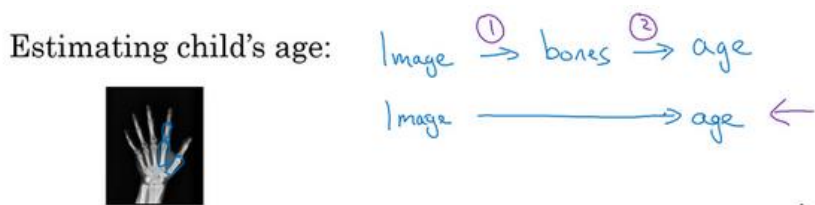
为什么两步法更好呢？实际上有两个原因。第一：解决的两个问题，每个问题实际上要简单得多。第二，两个子任务的训练数据都很多。具体来说，有很多数据可以用于人脸识别训练，对于这里的任务 1 来说，任务就是观察一张图，找出人脸所在的位置，把人脸图像框出来，所以有很多标签数据 (x, y) ，其中 x 是图片， y 是表示人脸的位置，可以建立一个神经网络，很好地处理任务 1。对于任务 2 来说，也有很多数据可用，业界领先的公司拥有数百万张人脸照片，所以输入一张裁剪得很紧凑的照片，比如这张红色照片，他们可以用来观察

两张图片，并试图判断照片里人的身份，确定是否同一个人，所以任务 2 还有很多数据。相比之下，如果想一步到位，这样 (x, y) 的数据对就少得多，其中 x 是门禁系统拍摄的图像， y 是那人的身份，因为没有足够多的数据去解决这个端到端学习问题，却有足够多的数据来解决子问题 1 和子问题 2。

实际上，把这个分成两个子问题，比纯粹的端到端深度学习方法，达到更好的表现。不过如果有足够多的数据来做端到端学习，也许端到端方法效果更好。但在今天的实践中，并不是最好的方法。

我们再来看几个例子，比如机器翻译。传统上，机器翻译系统有一个很复杂的流水线，比如英语机翻得到文本，然后做文本分析，基本上要从文本中提取一些特征之类的，经过很多步骤，最后会将英文文本翻译成法文。因为对于机器翻译来说的确有很多(英文, 法文)的数据对，端到端深度学习在机器翻译领域非常好用，那是因为在今天可以收集 $x - y$ 对的大数据集，就是英文句子和对应的法语翻译。在这个例子中，端到端深度学习效果很好。

最后一个例子，比如希望观察一个孩子手部的 X 光照片，并估计一个孩子的年龄，儿科医生用来判断一个孩子的发育是否正常。



处理这个例子的一个非端到端方法，就是照一张图，然后分割出每一块骨头，所以就是分辨出那段骨头应该在哪里，那段骨头在哪里，那段骨头在哪里。知道了不同骨骼的长度，可以去查表，查到儿童手中骨头的平均长度，然后用它来估计孩子的年龄，所以这种方法实际上很好。相比之下，如果直接从图像去判断孩子的年龄，那么需要大量的数据去直接训练。据我所知，这种做法今天还是不行的，因为没有足够的数据来训练端到端的方式。

我们可以将这个问题分解成两个步骤，第一步是一个比较简单的问题，也许不需要那么多数据，也许不需要许多 X 射线图像来切分骨骼。第二步，收集儿童手部的骨头长度的统计数据，不需要太多数据也能做出相当准确的估计，所以这个多步方法看起来很有希望，也许比端到端方法更有希望，至少直到能获得更多端到端学习的数据之前。

所以端到端深度学习系统是可行的，它表现可以很好，也可以简化系统架构，让你不需要搭建那么多手工设计的单独组件，但它也不是灵丹妙药，并不是每次都能成功。

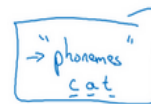
2.10 是否要使用端到端的深度学习？

假设正在搭建一个机器学习系统，是否使用端对端方法，我们来看看端到端深度学习的一些优缺点，这样就可以根据一些准则，判断应用程序是否有希望使用端到端方法。

Pros and cons of end-to-end deep learning

Pros:

- Let the data speak $x \rightarrow y$
- Less hand-designing of components needed

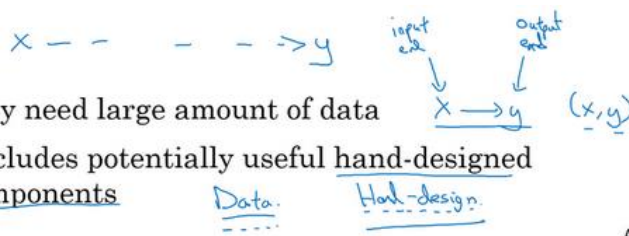


这里是应用端到端学习的一些好处，首先端到端学习真的只是让数据说话。如果有足够多的 (x, y) 数据，那么不管从 x 到 y 最适合的函数映射是什么，如果训练一个足够大的神经网络，希望这个神经网络能自己搞清楚，而使用纯机器学习方法，直接从 x 到 y 输入去训练的神经网络，可能更能够捕获数据中的任何统计信息，而不是被迫引入人类的成见。

例如，在语音识别领域，早期的识别系统有音位概念，就是基本的声音单元，如 **cat** 单词的“cat”的 **Cu-**、**Ah-**和 **Tu-**，我觉得这个音位是人类语言学家生造出来的，用音位描述语言也还算合理。但是不要强迫学习算法以音位为单位思考，这点有时没那么明显。如果让学习算法学习它想学习的任意表示方式，而不是强迫学习算法使用音位作为表示方式，那么其整体表现可能会更好。端到端深度学习的第二个好处就是，所需手工设计的组件更少，这也许能够简化设计工作流程，不需要花太多时间去手工设计功能，手工设计这些中间表示方式。

Cons:

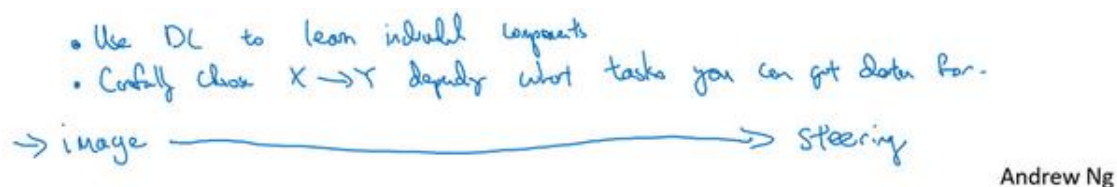
- May need large amount of data
- Excludes potentially useful hand-designed components



这里有一些缺点，首先，直接学到这个 x 到 y 的映射，可能需要大量 (x, y) 数据。我们在以前的视频里看过一个例子，其中可以收集大量任务数据，比如人脸识别，我们可以收集很多数据用来分辨图像中的人脸，当找到一张脸后，也可以找得到很多人脸识别数据。但是对于整个端到端任务，可能只有更少的数据可用。另一个缺点是，它排除了可能有用的手工设计组件。机器学习研究人员一般都很鄙视手工设计的東西，但如果没有很多数据，学习算法就没办法从很小的训练集数据中获得洞察力，所以手工设计组件在这种情况下可能是把人类知识直接注入算法的途径，这总不是一件坏事。学习算法有两个主要的知识来源，一个是数据，另一个是手工设计的任何东西，可能是组件，功能，或者其他东西。当然，有大量数据时，手工设计的東西就不太重要了。所以端到端深度学习的弊端之一是它把可能有用的人工设计的组件排除在外了，精心设计的人工组件可能非常有用，但它们也有可能真的伤害到算法表现。例如，强制算法以音位为单位思考，也许让算法自己找到更好的表示方法更好。所以这是一把双刃剑，但往往好处更多，手工设计的组件往往在训练集更小的时候帮助更大。

还有一个更复杂的例子，怎么造出一辆自己能行驶的车呢？可以做一件事，这不是端到端的深度学习方法，可以把车前方的雷达、激光雷达或者其他传感器的读数看成是输入图像。为了说明起来简单，我们就说拍一张车前方或者周围的照片，然后驾驶要安全的话，必须能检测到附近的车，也需要检测到行人，或检测其他的东西，当然，我们这里提供的是高度简化的例子。

弄清楚其他车的位置之后，就需要计划自己的路线。换句话说，当看到其他车子在哪，行人在哪里，算法需要决定如何摆方向盘在接下来的几秒钟内引导车子的路径。这是道路的俯视图，这是你的车，也许你决定了要走这条路线，那么就需要摆动方向盘到合适的角度，还要发出合适的加速和制动指令，所以从传感器或图像输入到检测行人和车辆，深度学习可以做得很好，但一旦知道其他车辆和行人的位置或者动向，选择一条车要走的路，这通常用的不是深度学习，而是用所谓的运动规划软件完成的。如果你学过机器人课程，你一定知道运动规划，然后决定了车子要走的路径之后，还会有一些其他算法，可以产生精确的决策确定方向盘应该精确地转多少度，油门或刹车上应该用多少力。



这个例子就表明了，如果想使用机器学习或者深度学习来学习某些单独的组件，那么当应用监督学习时，应该仔细选择要学习的 x 到 y 映射类型，这取决于那些任务可以收集数据。相比之下，谈论端到端深度学习方法是很激动人心的，输入图像，直接得出方向盘转角，但是就目前能收集到的数据而言，还有我们今天能够用神经网络学习的数据类型而言，这实际上不是最有希望的方法，或者说这个方法并不是团队想出的最好用的方法。而我认为这种纯粹端到端深度学习方法，其实前景不如这样更复杂的多步方法，因为目前能收集到的数据，还有我们现在训练神经网络的能力是有局限的。