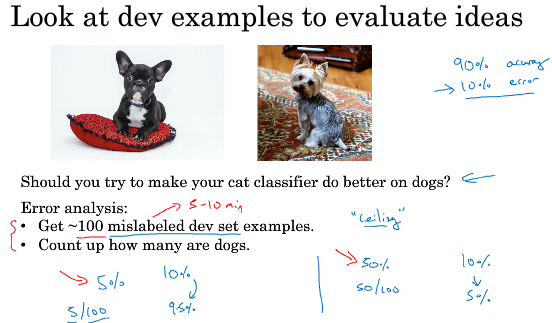
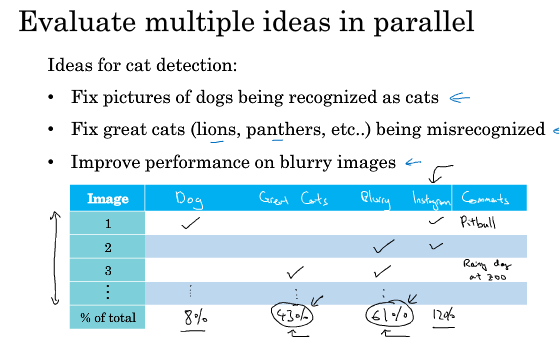
第二章 机器学习ML策略（二）

* 1. 进行误差分析

如果你想得到一个训练算法来做人类可以做的任务，而且你所训练的算法还没有达到人类的效果，你需要手动地检查算法中的错误，来得到你下一步该做什么，这个过程叫做错误分析。举个例子，猫分类系统，在验证集上正确率为90%，比想象中的结果差很多。错误里面有很多把狗看成了猫，有人建议是应该侧重点在狗身上，收集更多狗的图片或设计针对狗特有的特征之类的，为了使猫分类器在狗上表现更好，问题是你是否应该继续，并且开始一个项目侧重于狗的问题。错误分析的目的就是来判断到底值不值得这么做，首先拿大约100张分类错误的验证集图片并进行手动检测，只需要数一数看有多少张，验证集中标错的样本实际上是狗的图片，现在假设事实证明在验证集中分错的100张样本里有5%是狗的图片，也就是说验证集中分错的100张中有5张是狗，这意味着在这100张图片中，特指你分错的这100张，即使完全解决狗的问题，也只在这100张中多分对了5张。那么可以合理的判断出，在狗上面做改进没有多大作用。而如果发现有50张都是狗图，那么如果真的解决了狗的问题，错误率可能就从10%降到5%，这将是有意义的了。



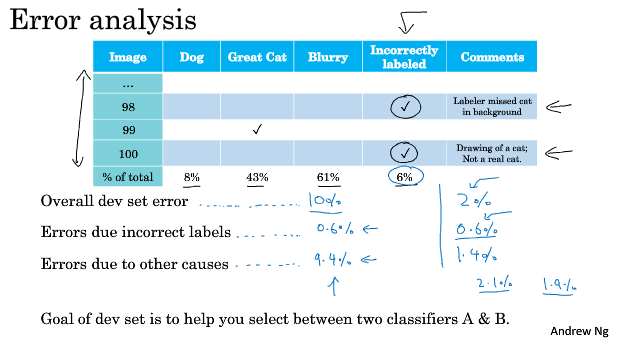
在机器学习中，有时候我们会认为手动操作和使用太多人工判断是不入流的，但是在构建应用系统时这个简单的计数过程，也就是错误分析，可以节省你很多时间，决定什么因素是最重要的，并且在哪个方向上做调整是最有效的。例如，在验证集中选出一部分数据，制作一个电子表格，将其中错误的样本错误分类的原因进行分类，看这个错误原因占的比重是多少，发现很多分类错误都由某个原因造成，并计算不同类别中的误判个数，在这个过程中，你也可能提出新的错误类别，通常有助于判断优先级，或者给新方向的灵感。



* 1. 清楚标注错误的数据

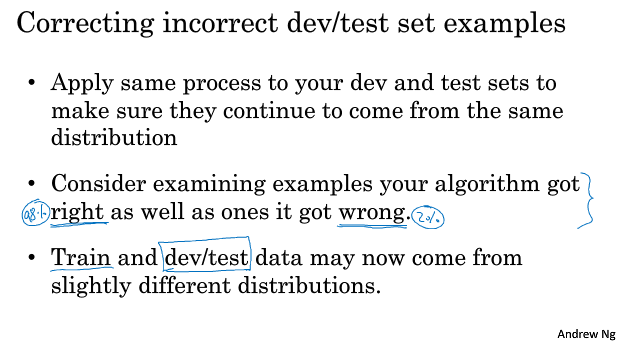
在做错误分类时，有时会发现开发集中有些是错误标签的（y值错误），对于那些错误的标签是不是都需要改正呢。深度学习算法对训练集中的随机错误有很好的鲁棒性，只要错误够随机，置之不理是可以的，但是对系统性的错误就没那么robust了，随机和近似随机误差对大多数深度学习算法来说，不成问题。

前面讨论的是训练集标间出错，那如果是开发集或者测试集有标记出错的而例子呢。一般建议在做误差分析的时候，在电子表上添加一个额外的列，统计标签Y错误的例子数，统计100个错误例子，其中你的分类器的输出和开发集的标签不一致，因为标签标错了，而不是你分类分错了。当统计出其他错误类型的百分比后，也可以统计标签标错的类型百分比。如图：



那么问题来了，是否值得修复这6%标记出错的例子，建议是如果这些错误严重影响了你在开发集上评估算法的能力，就应该花时间去修正错误的标签，反之，用开发集评估成本偏差的能力，就不用花时间解决。如果要修正开发集数据，手动重新检查标签，并尝试修正一些标签，有一些指导原则：

1. 不管用什么修正手段，都要同时作用到开发集和测试集上，因为开发集和测试集必须来自相同的分布。
2. 主要同时检验算法判断正确和判断错误的例子（这里指的是0和1），要检查算法出错的例子很容易，只需要看看那些例子是否需要修正，但还有可能有些例子，算法没判断对，也需要修正。如果只检查算法判断错的例子，对算法的偏差估计可能会变大，算法不公平。通常不这么做，因为如果你的分类器挺好，那么判断错的次数比判断正确的次数要少得多。
3. 如果要进入到一个开发集和测试集去修正这里的部分标签，可能会也可能不会对训练集做一样的事，前面说过，修正训练集中的标签其实相对没那么重要。



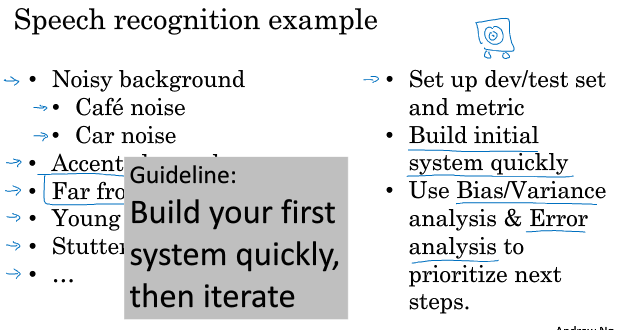
在训练算法时候，通常需要更多的人工误差分析，更多的人类见解来构建这些系统。亲自统计错误数量可以真的帮你找到需要优先处理的任务。

* 1. 快速搭建你的第一个系统，并进行迭代

应该尽快建立你的第一个系统原型，并且快速迭代。如果你正在考虑建立一个新的语音识别系统，可以走很多方向，考虑很多事情，比如特定的技术可以让语音识别系统对嘈杂的背景更加健壮，有一些方法可以在处理带口音的时候更加健壮，还有特定的问题和麦克风与人距离很远有关，就是所谓的远场语音识别，以及儿童的来自发音的挑战，还有比较倾向于使用的词汇……

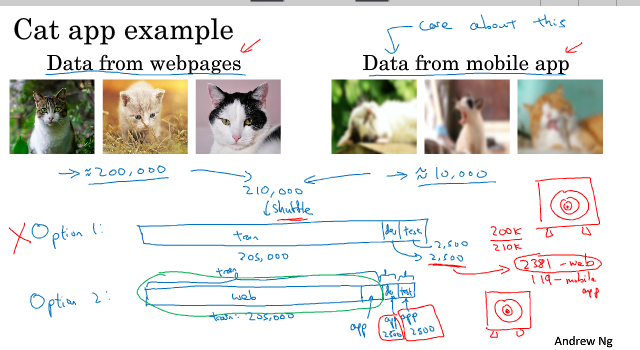
更普遍地来说，对于几乎所有的机器学习应用，可能有50个不同的方向去研究，每个方向都是合理的，会使你的系统更好，但挑战是如何从中选择需要关注的问题。所以就应该快速建立一个开发/测试集以及度量指标，这就决定了你的目标在哪儿。如果你发现出错了你始终可以稍后移动目标。然后我建议你快速建立一个初步的简单的机器学习系统，找到训练集，训练它然后看(结果)，开始观察并理解你的系统，对开发/训练集如何，价值和度量指标是多少，当你构建了初步的系统，你就可以用偏差/方差分析。

如果你研究的是一个比较老的话题，那么可以参考文献等一开始就建立一个比较复杂的模型，否则的话就应该先快速建立一个，可以帮你找到改善系统优先要处理的方向。

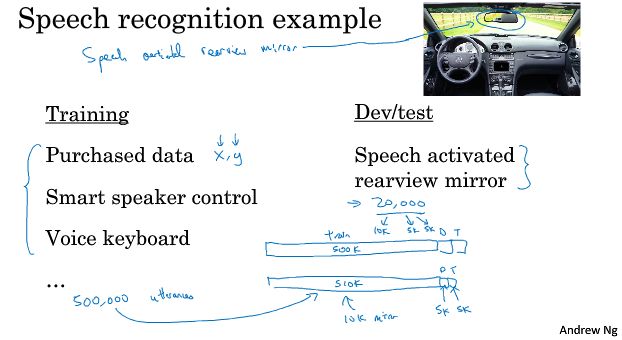


* 1. 在不同的划分上进行训练并测试

深度学习算法对训练数据的胃口很大，当你收集到足够多的数据构成训练集时，算法效果最好。这导致很多团队将能找到的任何数据都塞进训练集，只为有更多的训练数据，即使有些甚至很多这种数据，来自于与开发集和测试集不同的分布。针对这种情况告诉一些好的实践方法，比如，一个移动应用识别猫图，数据可能来自两处，用户自己上传的非专业的可能糊了的照片，以及网上下载的分辨率高的照片。但是你真正期待的是最终系统处理移动应用上传的图片的分布的表现，现在就比较为难了，因为你的数据集比较小，只有一小部分来自于用户上传，另外数据多的都是下载的。方案：将这些数据集放在一起，随机混合，得到训练/开发/测试集。



这样的数据分配有其优点和缺点，优点是这样一来你的训练/开发/测试集，都来自于同一分布，易于管理；而缺点，一个巨大的缺点是，仔细看看你的开发集它有2,500个样本，但是大部分来自网页图片的分布，而不是你真正关心的来自移动应用图片的分布。请记住设置开发集的目的是，告诉你的团队该往哪里瞄准，现在你正在瞄准的目标，正在花费大量的时间优化的是，网页图片的分布，而这其实不是你想要的。所以我建议不要采用方案一，因为它设置的开发集使你的团队致力于，优化一个与你实际关心的目标，并不相同的数据分布。方案二：将所有网络图片放入训练集，如果需要，再加入500张手机上传图，然后对于开发集和测试集都是手机图，这样划分的好处在于现在你瞄准的目标就是你想要处理的目标。缺点在于，现在的训练集分布和开发集测试集分布不一样，但事实证明，这样划分，在长期能给你带来更好的系统性能。



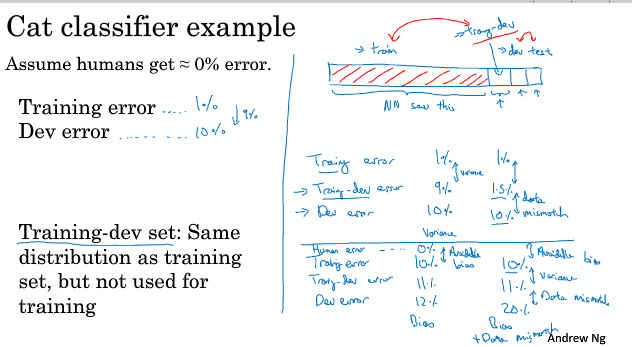
  如果让你的训练集和开发集测试集来自不同的分布，这样你就可以有更多的训练数据，将改善你的学习算法。

* 1. 不匹配数据划分的偏差和方差

通过估计学习算法的偏差和方差，能帮你确定下一步工作的优先级，但当你的训练集、开发集和测试集来自不同的分布时，偏差和方差的分析方法也会相应变化。

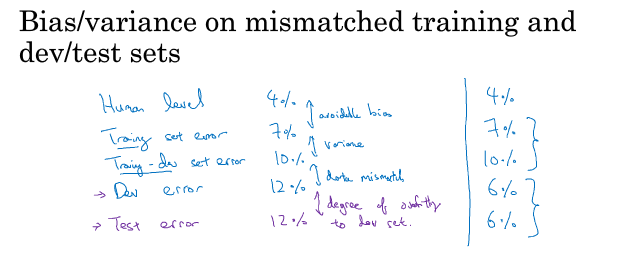
首先你在判断train error 1%和dev error 10%时，首先算法只见过训练集数据，没见过开发集数据，第二，开发集来自不同的分布，同时改变了这两件事情，很难确认这9%的误差有多少是因为算法没看过开发集中的数据导致的，有有多少是分布不同。为了弄清楚哪种影响更大，需要新定义一组数据，叫做训练-开发集(training-dev set)，这是一个新的数据子集，我们要让它与训练集拥有同样的数据分布，但你不用直接拿它来训练你的网络。

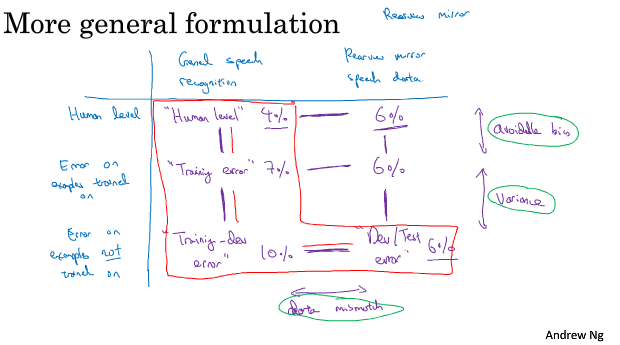
之前我们已经建立了训练集、开发集和测试集，开发集和测试集属于同分布，训练集数据属于不同分布，我们要做的是将训练集随机混淆(shuffle)，取出一小块数据作为训练-开发集，如同开发集与测试集分布相同，训练集与训练-开发集也遵循相同分布。区别在于 现在你只需要用着一部分训练集训练你的网络，不会让神经网络在训练-开发集上跑反向传播。为了进行误差分析，你要做的是看到是分类器在训练集上的误差，训练-开发集误差和开发集误差。



假设有以下情况：train 1%，train-dev 9%，dev 10%，对比前两者可以发现，这是方差问题，过拟合了，因为两者数据来自相同的分布，说明神经网络在训练集中表现很好，但无法泛化到相同分布的train-dev中。又比如：train 1%，train-dev 1.5%，dev 10%，现在方差问题就很小了，说明这就是数据不匹配的问题。换种情况：贝叶斯0%，train 10%，train-dev 11%，dev 12%，说明存在偏差问题了。贝叶斯0%，train 10%，train-dev 11%，dev 20%，就说明有两个问题了，偏差问题和数据不匹配的问题。

总结我们要看的是人类水平误差，训练集误差，训练-开发集误差，开发集误差，取决于这些误差之间差距有多大。如果后面再加上测试集，那么跟dev的差距就说明了对开发集过拟合的程度。如果差距很大，就说明需要一个更大的开发集，并且开发集和测试集来自同一分布。



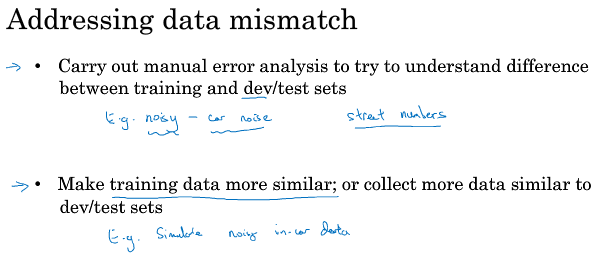


上图则是将我们要考虑的误差放入一个表中，横轴代表数据集分布，第一列为其他来源的数据，第二列为后视镜语音测试数据，第一行人类行为水平，第二行，在训练过的数据上的误差，第三行，没有训练过的数据上的误差。根据我们队数据集的划分已经是否在这些数据上进行训练，就可以把贝叶斯误差，train误差，train-dev误差，dev误差和test误差放入表中。并且两两之间的比较可以直接发现问题，告诉你下一步改进的方向是什么。

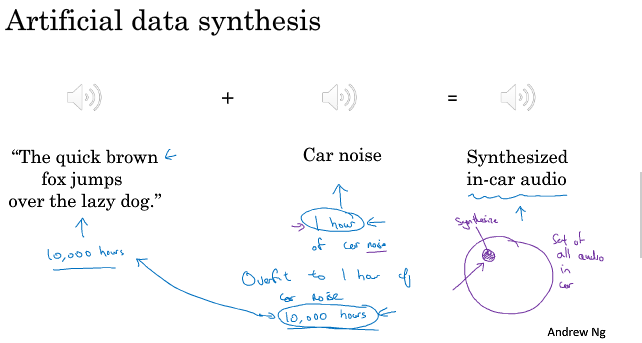
具体来说训练与开发集测试集不同分布的数据，可以为训练提供更多的数据，帮助学习算法获得更好的性能，但是除了偏差和方差问题，现在你又有了新的潜在问题：数据不匹配，解决数据失配问题有什么好办法呢？老实说实际上并没有很好的，至少是非常系统的处理数据失配的方法，但是有一些事情你可以试试。

* 1. 定位数据不匹配

数据不匹配的问题没有系统的解决方案，但是可以做一些尝试。比如亲自做误差分析尝试了解训练集和开发集的具体差异，为了避免对测试集过拟合，要做误差分析，应该只人工看开发集而不是测试集。或者也可以收集更多类似你的开发集和测试集的数据。这些不是系统的改进过程，不保证一定取得进展，但这种人工力量可以一起尝试收集更多和真正场合相似的数据。



如果你的目标是让你的训练集更接近开发集，应该怎么做？人工合成数据，比如汽车背景噪音下录得的数据，可以把清晰的音频跟噪声进行合成，可以快速的制造出更多的数据。人工合成数据有一个潜在的问题，可能对噪音过拟合。



总而言之，如果你发现你的数据存在不匹配的问题，建议做误差分析，或看看训练集开发集，试图找出并了解这两个数据分布到底有什么不同，看看是否有办法收集到更多看起来像开发集的数据作训练。人工合成确实是有效的，特别是现在在语音识别系统已经有成效，但当你使用人工合成数据的时候，一定要记住，有可能从所有可能性的空间里只选了很小一部分去模拟数据。

* 1. 迁移学习 transfer learning

神经网络有一个厉害点就是，可以从一个任务中学习得到知识，并将这些知识运用到另一个独立的任务中。

具体来说，在训练模型的第一阶段，当你在做图像识别的任务时，你训练了所有常用的神经网络的参数，所有权值，所有层，然后这个模型现在能够做出图像识别的预测。训练得出了这样一个神经网络之后，要实现迁移学习，你现在需要把数据集X和Y设定为放射影像数据。现在的Y是你想要预测的诊断结果，为最后一层神经网络创建一个新的随机初始化的权重，使用新建的这个输出层来进行放射结果的诊断，所以你要做的是初始化最后一层的权值，我们叫这个为和。然后重新在新的数据集上训练这个神经网络，就是新的放射数据集上，随机初始化，重新训练新的神经网络。

如果你只有一个小的放射数据集，你可以只重新训练最后一层的权值或倒数第二层，就是和，同时保留其它所有参数。如果你有足够的数据，你也可以重新训练神经网络的其余所有层所有参数。如果你重新训练所有参数，那么这个在图像识别数据的初期训练阶段，有时称为预训练，因为你用之前训练图像识别的网络去预先初始化后者预先训练神经网络的权重。然后你更新权重，在放射科数据上训练，这个过程叫微调fine tuning。根据你拥有的数据集规模，你可能仅对你的神经网络中新建的几层进行训练，或者你也可以重新训练其中更多的几层神经网络。

迁移学习什么时候有意义？当你在你的被迁移的模型中拥有海量数据，而你在需要解决的问题上拥有相对较少的数据时，迁移学习适用。反过来，迁移学习就没有多少意义了。继续总结，迁移学习必须是你尝试把任务B中的任务做好时才会有效，即通常情况下，你要把这个拥有更少数据的任务做好时。迁移学习在应用于任务A实际上拥有比任务B更少的数据的时候， 在这样的情况下，你并不能得到很大的增益。

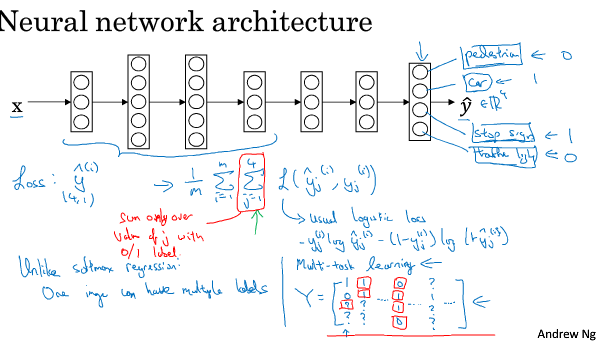
迁移学习，一种你把从一个任务中学到的信息迁移到另外一个不同的任务中的方式。

* 1. 多任务学习

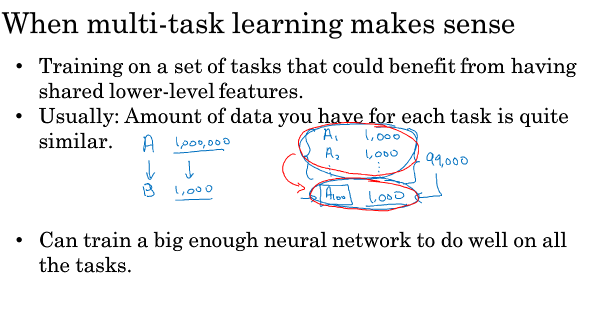
还有另外一种关于从多个任务中学习的版本叫做多任务学习，这是一种当你尝试同时从多个任务中学习的方式，而不是序列化的先从一个任务学习，然后迁移到另外一个任务的学习。

在迁移学习中，步骤是单向的，从A学到的知识迁移到B，多任务学习中，是同时开始学习的，试图让单个神经网络同时做几件事情，希望每个人物都能帮到其他任务。假设你正在建造一辆自动驾驶汽车，然后你的车将需要检测几个不同的事物如行人、其他车辆、停车标志， 还有交通信号灯和其他的东西。如果这个图像作为输入,如x(i),而输出并不是一个标签y(i)而是四个，跟softmax很类似，不同的地方时softmax是区分几个类别属于哪一类，从而表明是那一类的就是1，其余为0，也是一个列向量。而多任务学习是为了说明有哪些东西哪些类存在，所以每一个样本不仅一个标签，可能有两个或者多个，比如说明一张图里面既有人也有交通标志。所以多任务学习中，Y同样是个（n，m），n代表了同时学习几个任务，比如判断是否有人，是否有车，是否有交通标志……代价函数为：

和softmax回归区别是,不像softmax回归,它将单个标签分配给单个示例这一个图像可以有多个标签。如果你建立了一个神经网络，并且试图最小化loss，做的就是多任务学习，因为现在建立的单个神经网络观察每个图，然后解决四个问题，它试图告诉你每个图像中的4个对象。或者你可以做的是训练4个独立的神经网络，而不是训练一个网络做4件事。但是在不同输出之间，神经网络前面的特征可以共享，那么你会发现训练一个神经网络做4件事的结果比训练4个完全独立的神经网络的结果要好。



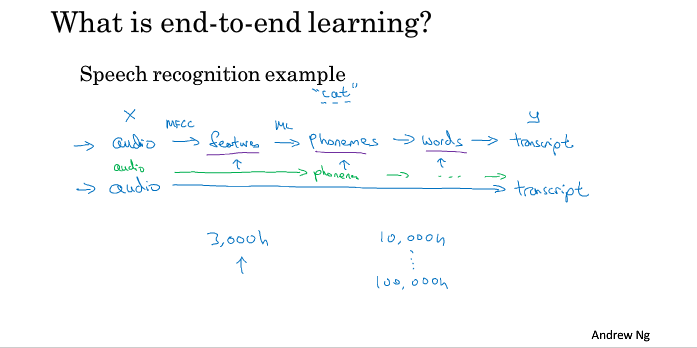
上面举的例子看起来每个样本都要全部标签，但实际上也可以是部分标签，神经网络同样可以完成多任务的工作。这里部分标签的意思是，比如并不知道这张图有没有人，用？来代替，不是0和1，那么仍然可以参与训练，我们只对有标签的值求和，？的部分就忽略。

多任务学习什么时候有意义：满足下图中的三点。其中第二点不是绝对正确的准则，通常会看如果你专注于单项任务，如果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。研究员Rich Carona,多年前发现如果神经网络不够大，多任务学习与单项训练相比会损害准确率。但如果你训练足够大的神经网络,那多任务学习，应该不会或很少影响性能。与你分别单独训练不同任务相比，它实际上能够提高性能。

总结一下，多任务学习能训练一个神经网络来完成多任务，这可以使你的性能比单独执行任务时更好。请注意一点，实际上迁移学习比多任务学习用得更多。所以如果你想解决一个机器学习问题，但你有一个相对较小得数据集，迁移学习可以帮助你。当你有个相关的问题有更大的数据集，你可以用来训练你的神经网络，然后迁移到小数据集的问题上因此,迁移学习在今天使用得很多。也有很多多任务学习但是多任务学习与迁移学习相比要少。但是有一个例外就是计算机视觉物体检测。我在那看到了很多训练神经网络用来检测不同物体的应用。这比训练单独的神经网络来检测物体更有效。但我认为迁移学习和多任务学习都以类似方式提出，实际上我看到迁移学习比多任务学习应用更多。我想这是因为很难找到这么多不同的任务要单独在单个神经网络上训练。再次,用计算机视觉的物体检测是最为显著的例外。

* 1. 什么是端到端的深度学习

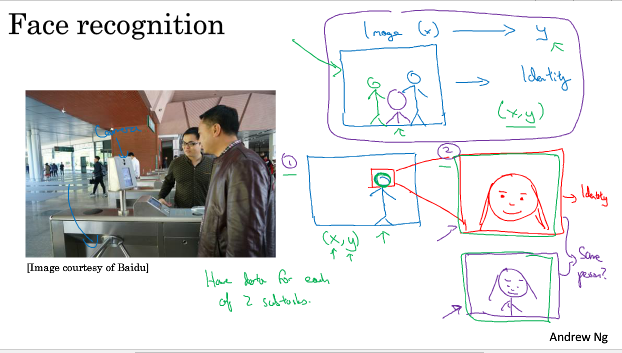
End-to-end learning端到端的学习，就是有一些数据处理系统或者是由多个学习阶段组成的学习系统，端到端的深度学习可以捕获所有阶段，用单个神经网络进行替换，运行速度更快。例如语音识别，目的是接收片段c，将其转化为对应的脚本Y。传统的语音识别分为好几个阶段，首先提取音频的一些特征，一些人工设计的音频特征。提到到低层次的特征之后，应用机器学习算法，查找音素（声音的基本单位），然后将把音素串在一起，形成单词，再把单词合在一起组成音频的脚本。与上面的分阶段不同，端到端的学习可以直接训练一个庞大的神经网络，只需要输入音频片段，然后直接得到脚本输出。从最新的训练数据，直接学习到从x到y的映射，中间绕过了很多步骤。端到端学习面临的最大一个挑战就是需要庞大的数据才能取得好的效果。如果你有中等大小的数据集，可以采取折中的方式，输入音频，绕过特种证，只学习输出神经网络的音素，然后继续其他阶段。



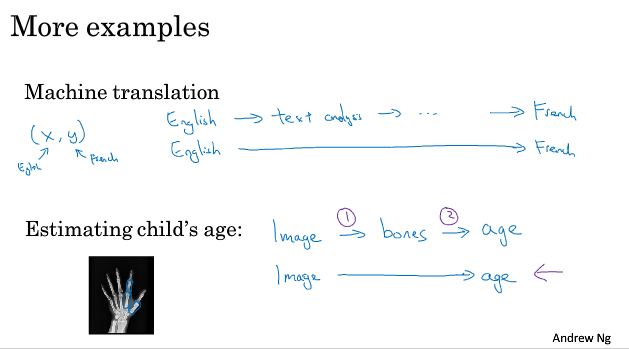
另一个例子，门禁人脸识别，由百度的研究者林元庆开发，如何构建这样一个系统呢？有人接近旋转门，摄像头就能捕捉到一张图像，可以试着学习一个函数，直接建立图像x和个人Y的映射，事实证明，这不是最好的方法，其中一个问题就是，人可以从不同的方向接近旋转门，并且靠近的距离也不一样，从而在一张图片上，脸的位置和脸的大小都各种各样。因此，门禁应用实际上不只是采集原始影像，把它喂给神经网络，试图推断一个人的身份。箱单，还多了一个步骤。首先运行一个软件来检测此人的脸，探测器计算这个人脸的位置，检测到位置之后，放大脸的这部分图片，进行裁剪，使人脸位于图像中心。然后再送进神经网络，试着学习估计这个人的身份。研究者们发现比起一步学到所有的东西更加有效的是，把这个问题拆分为两个步骤：首先找到脸的位置，再推断这是谁的脸。第二种方法使得学习算法（事实上有两个学习算法）得以解决两个比较简单的任务，从而总体上获得更好地效果。

实际上第二步的训练方式是，当你训练网络时，输入两张图像，网络就会告诉你，这两张图像是不是同一个人。所以你先有所有员工的文件，再那检测器拍的图像与所有员工的进行比较，找出对应的那个人。为什么使用两个阶段的方法好呢，有两个原因。第一因为这两个子问题都比原问题简单的多，更重要的是，有很多的数据可以供这两个子阶段使用。特别是你可以获得很多数据用于人脸检测，即第一个子问题:检查一张图片，找到人脸在这个图片中的位置，所以有很多数据，很多有标签的数据（X, Y)其中X是一张图片，而y代表人脸在这个图片中的位置。所以你能够建立一个神经网络很好的处理第一个子问题。同样的，有很多数据可以用于第二个子问题。

  然而，如果你尝试着让一个模型同时学习所有的步骤，只能获取很少数据（X，Y）。其中X是如图所示的旋转栅图片，Y是人的身份。因为没有足够的数据去解决端到端的学习问题，却有足够的数据去解决两个子问题，所以在实际应用中，把整个问题分成两个子问题来解决会比完全端到端的深度学习方案得到更好的性能。尽管如果你有足够的数据，也许端到端的方案会更加好，但是目前来说在实际应用中这不是最好的方法。



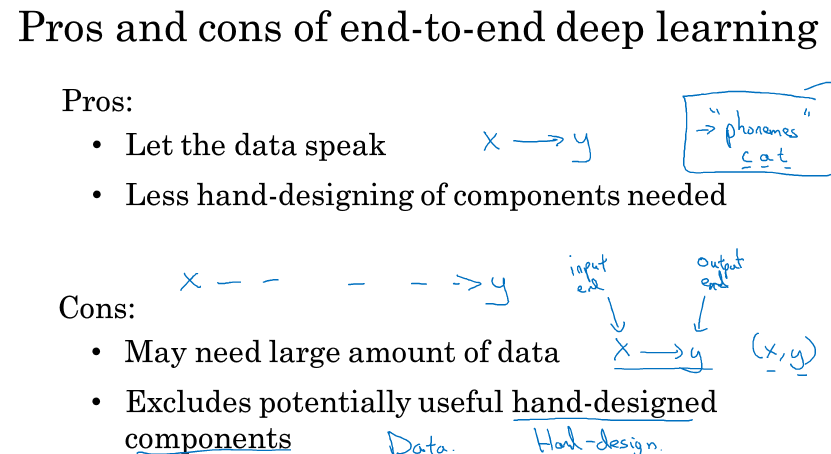
让我们来看看另一些例子以机器翻译作为例子，传统上机器翻译系统也是一个非常冗长和复杂的流程，首先获取，比如说，英文的文本然后做文本分析。简单来说,从文本中抽取一组特征等，然后经过很多步骤后，最终，比如说，得到一个从英文到法文的文本翻译。对于机器翻译来说，你可以获取很多英文和法文对应的文本对，所以端到端深度学习在机器翻译领域非常有效。那是因为今天，收集大量（X,Y）英文和法文对应的文本对的数据集，英文和法文对应的文本对的数据集是可行的。在这个例子里，端到端的深度学习效果很好。最后一个例子,假设你想通过查看儿童手掌的x光图片来估计这个儿童的年龄。事实上，通过手掌的x光图片估计儿童的年龄的典型应用场景。儿科医生使用这个工具去评估一个儿童是否健康成长。如果使用非端到端的方法解决这个问题，是从一个图片中分割出或识别出骨骼，就是尝试着识别这个骨头部分在哪里，另外一个骨头部分在哪里，还有一个骨头部分在哪里等等。基于不同骨骼的长度，你可以大概从表中查找到儿童手掌的平均骨骼长度，然后用这个信息来估计这个儿童的年龄，这样的方法实际上很有效。相反,如果你想直接从图片联系到孩子的年龄,你需要很多数据，而且基于我的了解，这种方法现今不太可行，就是因为没有足够的数据去支持端到端的训练。相反，你能够想象其实我们可以把这个问题分成两步。第一步是相对简单的方法，也许你不需要特别多的数据，也许你不需要特别多的X光图片数据去分割出骨骼。然后第二步，收集统计儿童手掌的信息，你也能基于不多的数据，得到比较准确的评估。所以，这种多步的方法看起来很有前景，可能比端到端的方法更可行。除非你可以获取支持端到端学习的更多的数据。所以，如果端到端深度学习可行，可以非常有效和简化系统，且不用建立很多手工设计的单个组件。但是它不是万能的，它并不总是有效。



* 1. 是否要使用端到端的深度学习

假设你正在搭建一个机器学习系统，要决定是否使用端到端学习，优点：第一点就是端到端学习真正地让数据发挥主导作用，所以如果你有足够的(X, Y)的数据，那么不论那个能最好地将X映射到Y的函数是什么样子，只要你有足够大的神经网络。顺利的话，神经网络都能拟合出来通过单纯的使用机器学习方法。你的神经网络能够更好从输入的X->Y中，学习到数据内在的统计学特性，而不是被迫去反映人的先见。使用端到端深度学习的第二个好处就是，所需的人类动手设计的组件变的更少了，所以这可以简化你的设计工作流程。意味着你不需要话大量的时间去动手设计特征，手工设计这些中间表示形式。

缺点就是第一，端到端学习需要大量的数据，所以为了使用机器学习直接得出X到Y的映射，你或许需要大量的(X，Y)数据。X是你的端到端学习的输入端，而Y是你的输出端。所以，你需要有关输入输出端的所有的(X，Y)数据，去训练你的系统。另一个缺点就是它排除了一些具有潜在用途的手工设计组件，但是如果你没有足够的数据，那么你的学习算法就不能够洞悉你数据中的规律，如果你的训练集很小，手工设计的组件的确是一条可行之路，去将人工知识融入到算法中。



学习算法有两个主要的知识来源一个是数据，另一个是你手工设计的东西。这些东西可以是算法中的组件，或者是特征，或者是其他的东西，所以如果你有很多很多的数据，手工设计就没那么重要。它排除了可能潜在有用的手工设计的成分，如果设计良好，手工设计的成分会带来有很大的帮助。

