Machine Learning Strategy

1.Why ML Strategy

当最初得到一个深度神经网络模型时，可能希望从很多方面对它进行优化：
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 L2 regularization
Network architecture: Activation functions, #hidden units…

可选择的方法很多，也很复杂、繁琐。盲目选择、尝试不仅耗费时间而且可能收效甚微。因此，使用快速、有效的策略来优化机器学习模型是非常必要的。

2.Orthogonalization

机器学习中有许多参数、超参数需要调试。通过每次**只调试一个参数，保持其它参数不变**，而得到的模型某一性能改变是一种最常用的调参策略，称之为**正交化方法**（Orthogonalization）。

Orthogonalization的核心在于每次调试一个参数只会影响模型的某一个性能，而不会影响其它功能。也就是说彼此之间是互不影响的，是正交的，这也是Orthogonalization名称的由来。这种方法能够更快更有效地进行机器学习模型的调试和优化。

对应到机器学习监督式学习模型中，可以大致分成四个独立的“功能”，每个“功能”对应一些可调节的唯一的旋钮。四个“功能”如下：
Fit training set well on cost function
Fit dev set well on cost function
Fit test set well on cost function
Performs well in real world

其中，第一条**优化训练集**可以通过使用**更复杂NN，使用Adam等优化算法**来实现；
第二条**优化验证集**可以通过**正则化，采用更多训练样本**来实现；
第三条**优化测试集**可以通过**使用更多的验证集样本**来实现；
第四条**提升实际应用模型可以通过更换验证集，使用新的cost function来实现。**概括来说，每一种“功能”对应不同的调节方法。而这些调节方法只会对应一个“功能”，是正交的。

early stopping在模型功能调试中并不推荐使用。
因为early stopping在提升验证集性能的同时降低了训练集的性能。也就是说**early stopping**同时影响两个“功能”，**不具有独立性、正交性。**

3. Single number evaluation metric

构建、优化机器学习模型时，单值评价指标非常必要。有了量化的单值评价指标后，就能根据这一指标比较不同超参数对应的模型的优劣，从而选择最优的那个模型。

举个例子，比如有A和B两个模型，它们的准确率（Precision）和召回率（Recall）分别如下：

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

如果只看Precision的话，B模型更好。如果只看Recall的话，A模型更好。
实际应用中，通常使用单值评价指标F1 Score来评价模型的好坏。F1 Score综合了Precision和Recall的大小，计算方法如下：

$$F1 = \frac{2 \cdot P \cdot R}{P + R}$$

然后得到了A和B模型各自的F1 Score：

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

从F1 Score来看，A模型比B模型更好一些。通过引入单值评价指标F1 Score，很方便对不同模型进行比较。

除了F1 Score之外，还可以使用**平均值**作为单值评价指标来对模型进行评估。
如下图所示，A, B, C, D, E, F六个模型对不同国家样本的错误率不同，可以计算其平均性能，然后选择平均错误率最小的那个模型（C模型）。

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

4.Satisficing and Optimizing metic

有时候，要把所有的性能指标都综合在一起，构成单值评价指标是比较困难的。
解决办法是，可以把某些性能作为**优化指标（Optimizing metic），寻求最优化值**；而某些性能作为**满意指标（Satisficing metic），只要满足阈值**就行了。

举个猫类识别的例子，有A，B，C三个模型，各个模型的Accuracy和Running time如下表中所示：

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

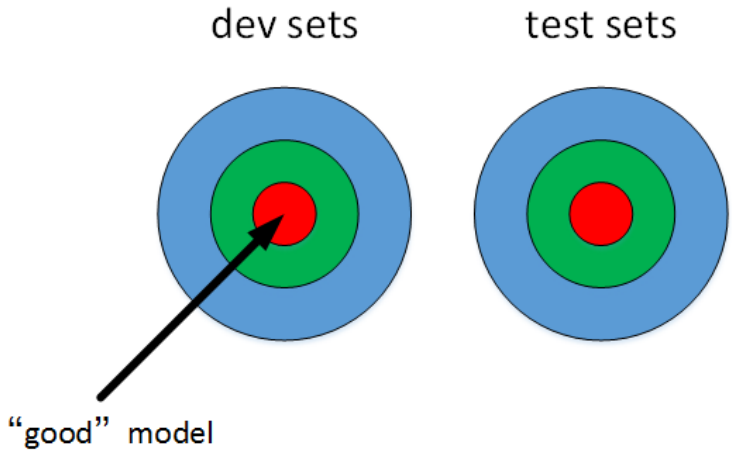
Accuracy和Running time这两个性能不太合适综合成单值评价指标。因此，可以将Accuracy作为优化指标（Optimizing metic），将Running time作为满意指标（Satisficing metic）。也就是说，给Running time设定一个阈值，在其满足阈值的情况下，选择Accuracy最大的模型。如果设定Running time必须在100ms以内，那么很明显，模型C不满足阈值条件，首先剔除；模型B相比较模型A而言，Accuracy更高，性能更好。

概括来说，性能指标（Optimizing metic）是需要优化的，越优越好；而满意指标（Satisficing metic）只要满足设定的阈值就好了。

5. Train/dev/test distributions

Train/dev/test sets如何设置对机器学习的模型训练非常重要，合理设置能够大大提高模型训练效率和模型质量。

原则上应该尽量保证dev sets和test sets来源于同一分布且都反映了实际样本的情况。如果dev sets和test sets不来自同一分布，那么从dev sets上选择的“最佳”模型往往不能够在test sets上表现得很好。这就好比在dev sets上找到最接近一个靶的靶心的箭，但是test sets提供的靶心却远远偏离dev sets上的靶心，结果这支肯定无法射中test sets上的靶心位置。



6. Size of the dev and test sets

当样本数量不多（小于一万）的时候，通常将Train/dev/test sets的比例设为60%/20%/20%，在没有dev sets的情况下，Train/test sets的比例设为70%/30%。当样本数量很大（百万级别）的时候，通常将相应的比例设为98%/1%/1%或者99%/1%。

对于dev sets数量的设置，应该遵循的准则是通过dev sets能够检测不同算法或模型的区别，以便选择出更好的模型。

对于test sets数量的设置，应该遵循的准则是通过test sets能够反映出模型在实际中的表现。

实际应用中，可能只有train/dev sets，而没有test sets。这种情况也是允许的，只要算法模型没有对dev sets过拟合。但是，条件允许的话，最好是有test sets，实现无偏估计。

7. When to change dev/test sets and metrics

算法模型的评价标准有时候需要根据实际情况进行动态调整，目的是让算法模型在实际应用中有更好的效果。

举个猫类识别的例子。初始的评价标准是错误率，算法A错误率为3%，算法B错误率为5%。显然，A更好一些。但是，实际使用时发现算法A会通过一些色情图片，但是B没有出现这种情况。从用户的角度来说，他们可能更倾向选择B模型，虽然B的错误率高一些。这时候，就需要改变之前单纯只是使用错误率作为评价标准，而考虑新的情况进行改变。例如增加色情图片的权重，增加其代价。

原来的cost function：

$$J = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$

更改评价标准后的cost function： J=

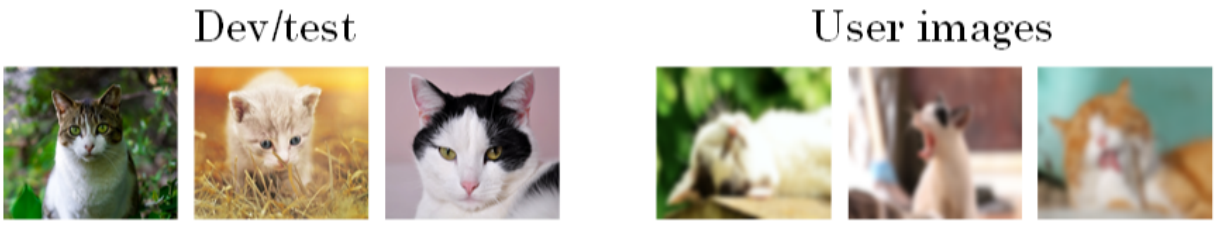
$$\frac{1}{w^{(i)}}\sum_{i=1}^{m} w^{(i)} L(\hat{y}^{(i)}, y^{(i)})$$

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

概括来说，机器学习可分为两个过程：
Define a metric to evaluate classifiers
How to do well on this metric

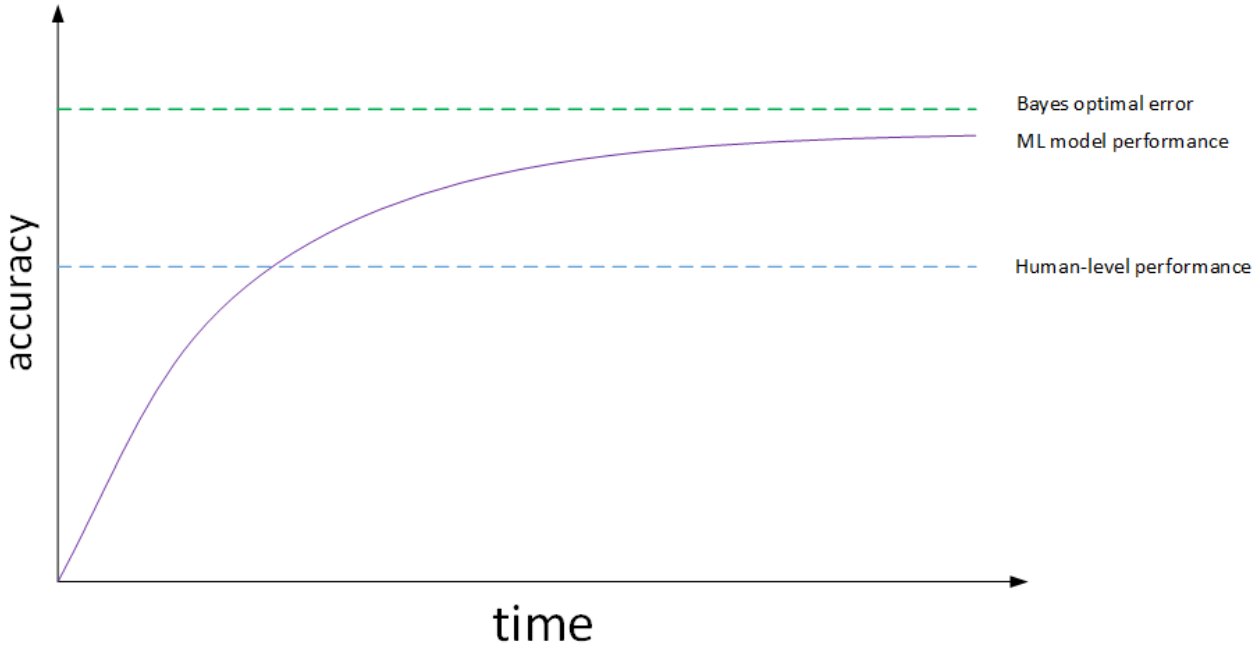
也就是说，第一步是找靶心，第二步是通过训练，射中靶心。
但是在训练的过程中可能会根据实际情况改变算法模型的评价标准，进行动态调整。

另外一个需要动态改变评价标准的情况是dev/test sets与实际使用的样本分布不一致。比如猫类识别样本图像分辨率差异。



8. Why human-level performance

机器学习模型的表现通常会跟人类水平表现作比较，如下图所示：



图中，横坐标是训练时间，纵坐标是准确性。机器学习模型经过训练会不断接近human-level performance甚至超过它。但是，超过human-level performance之后，准确性会上升得比较缓慢，最终不断接近理想的最优情况，我们称之为bayes optimal error。理论上任何模型都不能超过它，**bayes optimal error代表了最佳表现。**

实际上，human-level performance在某些方面有不俗的表现。例如图像识别、语音识别等领域，人类是很擅长的。所以，让机器学习模型性能不断接近human-level performance非常必要也做出很多努力：
Get labeled data from humans.
Gain insight from manual error analysis: Why did a person get this right?
Better analysis of bias/variance.

9. Avoidable bias

实际应用中，要看human-level error，training error和dev error的相对值。例如猫类识别的例子中，如果human-level error为1%，training error为8%，dev error为10%。由于training error与human-level error相差7%，dev error与training error只相差2%，所以目标是尽量在训练过程中减小training error，即减小偏差bias。如果图片很模糊，肉眼也看不太清，human-level error提高到7.5%。这时，由于training error与human-level error只相差0.5%，dev error与training error只相差2%，所以目标是尽量在训练过程中减小dev error，即方差variance。这是相对而言的。

对于物体识别这类CV问题，human-level error是很低的，很接近理想情况下的bayes optimal error。因此，上面例子中的1%和7.5%都可以近似看成是两种情况下对应的bayes optimal error。实际应用中，我们一般会用human-level error代表bayes optimal error。

通常，**把training error与human-level error之间的差值称为bias**，也称作avoidable bias；**把dev error与training error之间的差值称为variance。**根据bias和variance值的相对大小，可以知道算法模型是否发生了欠拟合或者过拟合。

10. Understanding human-level performance

human-level performance能够代表bayes optimal error。但是，human-level performance如何定义呢？举个医学图像识别的例子，不同人群的error有所不同：
Typical human : 3% error
Typical doctor : 1% error
Experienced doctor : 0.7% error
Team of experienced doctors : 0.5% error

不同人群他们的错误率不同。一般来说，将表现最好的那一组，即Team of experienced doctors作为human-level performance。那么，这个例子中，human-level error就为0.5%。但是实际应用中，不同人可能选择的human-level performance基准是不同的，这会带来一些影响。

假如该模型training error为0.7%，dev error为0.8。如果选择Team of experienced doctors，即human-level error为0.5%，则bias比variance更加突出。如果选择Experienced doctor，即human-level error为0.7%，则variance更加突出。也就是说，选择什么样的human-level error，有时候会影响bias和variance值的相对变化。当然这种情况一般只会在模型表现很好，接近bayes optimal error的时候出现。越接近bayes optimal error，模型越难继续优化，因为这时候的human-level performance可能是比较模糊难以准确定义的。

11. Surpassing human-level performance

对于自然感知类问题，例如视觉、听觉等，机器学习的表现不及人类。但是在很多其它方面，机器学习模型的表现已经超过人类了，包括：
Online advertising
Product recommendations
Logistics(predicting transit time)
Loan approvals

实际上，机器学习模型超过human-level performance是比较困难的。但是只要提供足够多的样本数据，训练复杂的神经网络，模型预测准确性会大大提高，很有可能接近甚至超过human-level performance
当算法模型的表现超过human-level performance时，很难再通过人的直觉来解决如何继续提高算法模型性能的问题。

12. Improving your model performance

提高机器学习模型性能主要要解决两个问题：avoidable bias和variance。
training error与human-level error之间的差值反映的是avoidable bias，dev error与training error之间的差值反映的是variance。

解决avoidable bias的常用方法包括：
Train bigger model
Train longer/better optimization algorithms: momentum, RMSprop, Adam
NN architecture/hyperparameters search

解决variance的常用方法包括：
More data
Regularization: L2, dropout, data augmentation
NN architecture/hyperparameters search

13.Carrying out error analysis

对已经建立的机器学习模型进行错误分析（error analysis）十分必要，而且有针对性地、正确地进行error analysis更加重要。

举个例子，猫类识别问题，已经建立的模型的错误率为10%。为了提高正确率，发现该模型会将一些狗类图片错误分类成猫。一种常规解决办法是扩大狗类样本，增强模型对够类（负样本）的训练。但是，这一过程可能会花费几个月的时间，耗费这么大的时间成本到底是否值得呢？也就是说扩大狗类样本，重新训练模型，对提高模型准确率到底有多大作用？这时候就需要进行error analysis，帮助做出判断。

方法很简单，可以从分类错误的样本中统计出狗类的样本数量。根据狗类样本所占的比重，判断这一问题的重要性。假如狗类样本所占比重仅为5%，即使花费几个月的时间扩大狗类样本，提升模型对其识别率，改进后的模型错误率最多只会降低到9.5%。相比之前的10%，并没有显著改善。

把这种性能限制称为ceiling on performance。相反，假如错误样本中狗类所占比重为50%，那么改进后的模型错误率有望降低到5%，性能改善很大。因此，值得去花费更多的时间扩大狗类样本。

这种error analysis虽然简单，但是能够避免花费大量的时间精力去做一些对提高模型性能收效甚微的工作，可以专注解决影响模型正确率的主要问题，十分必要。

这种error analysis可以同时评估多个影响模型性能的因素，通过各自在错误样本中所占的比例来判断其重要性。例如，猫类识别模型中，可能有以下几个影响因素：
Fix pictures of dogs being recognized as cats
Fix great cats(lions, panthers, etc…) being misrecognized
Improve performance on blurry images

通常来说，比例越大，影响越大，越应该花费时间和精力着重解决这一问题。这种error analysis让改进模型更加有针对性，从而提高效率。

14.Cleaning up incorrectly labeled data

监督式学习中，训练样本有时候会出现输出y标注错误的情况，即incorrectly labeled examples。如果这些label标错的情况是随机性的（random errors），DL算法对其包容性是比较强的，即健壮性好，一般可以直接忽略，无需修复。然而，如果是系统错误（systematic errors），这将对DL算法造成影响，降低模型性能。

刚才说的是训练样本中出现incorrectly labeled data，如果是dev/test sets中出现incorrectly labeled data，怎么办

方法很简单，利用上节内容介绍的error analysis，统计dev sets中所有分类错误的样本中incorrectly labeled data所占的比例。根据该比例的大小，决定是否需要修正所有incorrectly labeled data，还是可以忽略。举例说明，若：
Overall dev set error: 10%
Errors due incorrect labels: 0.6%
Errors due to other causes: 9.4%

上面数据表明Errors due incorrect labels所占的比例仅为0.6%，占dev set error的6%，而其它类型错误占dev set error的94%。因此，这种情况下，可以忽略incorrectly labeled data。

如果优化DL算法后，出现下面这种情况： Overall dev set error: 2%
Errors due incorrect labels: 0.6%
Errors due to other causes: 1.4%

上面数据表明Errors due incorrect labels所占的比例依然为0.6%，但是却占dev set error的30%，而其它类型错误占dev set error的70%。因此，这种情况下，incorrectly labeled data不可忽略，需要手动修正。

dev set的主要作用是在不同算法之间进行比较，选择错误率最小的算法模型。但是，如果有incorrectly labeled data的存在，当不同算法错误率比较接近的时候，无法仅仅根据Overall dev set error准确指出哪个算法模型更好，必须修正incorrectly labeled data。

关于修正incorrect dev/test set data，有几条建议：
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

15.Build your first system quickly then iterate

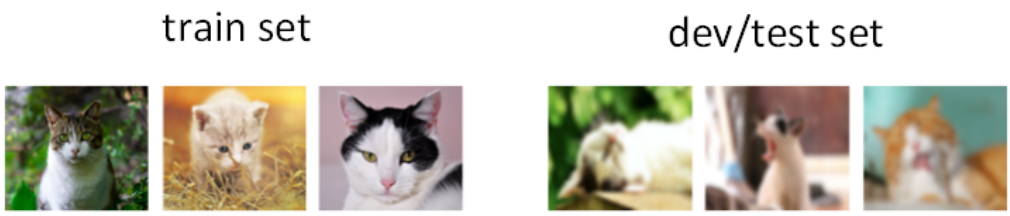
如何构建一个机器学习应用模型。
先快速构建第一个简单模型，然后再反复迭代优化。

Set up dev/test set and metric
Build initial system quickly
Use Bias/Variance analysis & Error analysis to prioritize next steps

16. Training and testing on different distribution

当train set与dev/test set不来自同一个分布的时候，该如何解决这一问题，构建准确的机器学习模型

以猫类识别为例，train set来自于网络下载（webpages），图片比较清晰；dev/test set来自用户手机拍摄（mobile app），图片比较模糊。假如train set的大小为200000，而dev/test set的大小为10000，显然train set要远远大于dev/test set。



虽然dev/test set质量不高，但是模型最终主要应用在对这些模糊的照片的处理上。面对train set与dev/test set分布不同的情况，有两种解决方法。

第一种方法是**将train set和dev/test set完全混合**，然后在随机选择一部分作为train set，另一部分作为dev/test set。例如，混合210000例样本，然后随机选择205000例样本作为train set，2500例作为dev set，2500例作为test set。这种做法的优点是实现train set和dev/test set分布一致，缺点是dev/test set中webpages图片所占的比重比mobile app图片大得多。例如dev set包含2500例样本，大约有2381例来自webpages，只有119例来自mobile app。这样，dev set的算法模型对比验证，仍然主要由webpages决定，实际应用的mobile app图片所占比重很小，达不到验证效果。因此，这种方法并不是很好。

第二种方法是将**原来的train set和一部分dev/test set组合当成train set，剩下的dev/test set分别作为dev set和test set。**例如，200000例webpages图片和5000例mobile app图片组合成train set，剩下的2500例mobile app图片作为dev set，2500例mobile app图片作为test set。其关键在于dev/test set全部来自于mobile app。这样保证了验证集最接近实际应用场合。这种方法较为常用，而且性能表现比较好。

17.Bias and Variance with mismatched data distributions

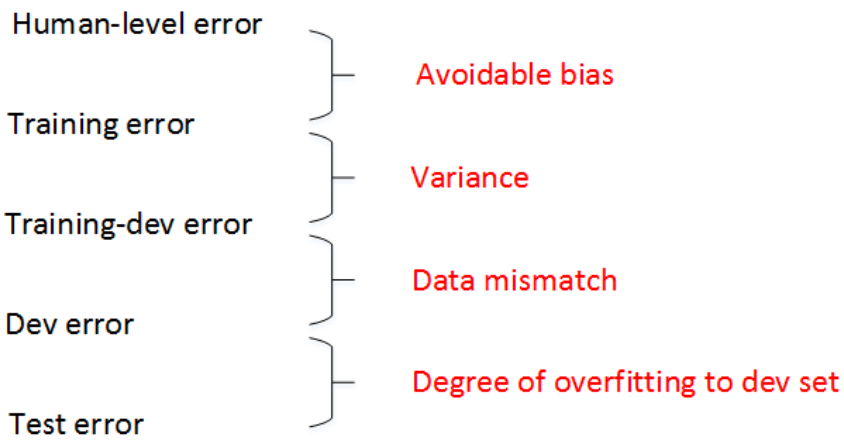
根据human-level error、training error和dev error的相对值可以判定是否出现了bias或者variance。但是，需要注意的一点是，如果train set和dev/test set来源于不同分布，则无法直接根据相对值大小来判断。
例如某个模型human-level error为0%，training error为1%，dev error为10%。根据我们之前的理解，显然该模型出现了variance。但是，training error与dev error之间的差值9%可能来自算法本身（variance），也可能来自于样本分布不同。比如dev set都是很模糊的图片样本，本身就难以识别，跟算法模型关系不大。因此不能简单认为出现了variance。

在可能伴有**train set与dev/test set分布不一致**的情况下，**定位是否出现variance的方法是设置train-dev set。**
train-dev set的定义是：“Same distribution as training set, but not used for training.”也就是说，从原来的train set中分割出一部分作为train-dev set，train-dev set不作为训练模型使用，而是与dev set一样用于验证。

这样，就有training error、training-dev error和dev error三种error。其中，**training error与training-dev error的差值反映了variance；training-dev error与dev error的差值反映了data mismatch problem**，即样本分布不一致。

举例说明，如果training error为1%，training-dev error为9%，dev error为10%，则variance问题比较突出。如果training error为1%，training-dev error为1.5%，dev error为10%，则data mismatch problem比较突出。通过引入train-dev set，能够比较准确地定位出现了variance还是data mismatch。

总结一下human-level error、training error、training-dev error、dev error以及test error之间的差值关系和反映的问题：



一般情况下，human-level error、training error、training-dev error、dev error以及test error的数值是递增的，但是也会出现dev error和test error下降的情况。这主要可能是因为训练样本比验证/测试样本更加复杂，难以训练。

18.Addressing data mismatch

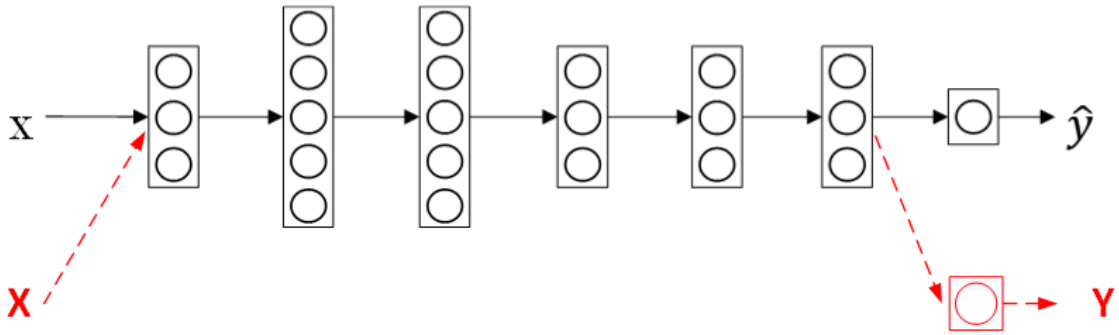
关于如何解决train set与dev/test set样本分布不一致的问题，有两条建议：
Carry out manual error analysis to try to understand difference between training dev/test sets
Make training data more similar; or collect more data similar to dev/test sets

为了让train set与dev/test set类似，可以使用人工数据合成的方法（artificial data synthesis）。
例如说话人识别问题，实际应用场合（dev/test set）是包含背景噪声的，而训练样本train set很可能没有背景噪声。为了让train set与dev/test set分布一致，可以在train set上人工添加背景噪声，合成类似实际场景的声音。这样会让模型训练的效果更准确。但是，需要注意的是，不能给每段语音都增加同一段背景噪声，这样会出现对背景噪音的过拟合，效果不佳。这就是人工数据合成需要注意的地方。

19.Transfer learning

深度学习非常强大的一个功能之一就是有时候可以将已经训练好的模型的一部分知识（网络结构）直接应用到另一个类似模型中去。比如已经训练好一个猫类识别的神经网络模型，那么可以直接把该模型中的一部分网络结构应用到使用X光片预测疾病的模型中去。这种学习方法被称为**迁移学习**（Transfer Learning）。

如果已经有一个训练好的神经网络，用来做图像识别。现在，想要构建另外一个通过X光片进行诊断的模型。迁移学习的做法是无需重新构建新的模型，而是利用之前的神经网络模型，只改变样本输入、输出以及输出层的权重系数$W^{[L]}$，$b^{[L]}$。也就是说对新的样本(X,Y)，**重新训练输出层权重系数**$W^{[L]}$，$b^{[L]}$，而其它层所有的权重系数$W^{[l]}$，$b^{[l]}$保持不变。

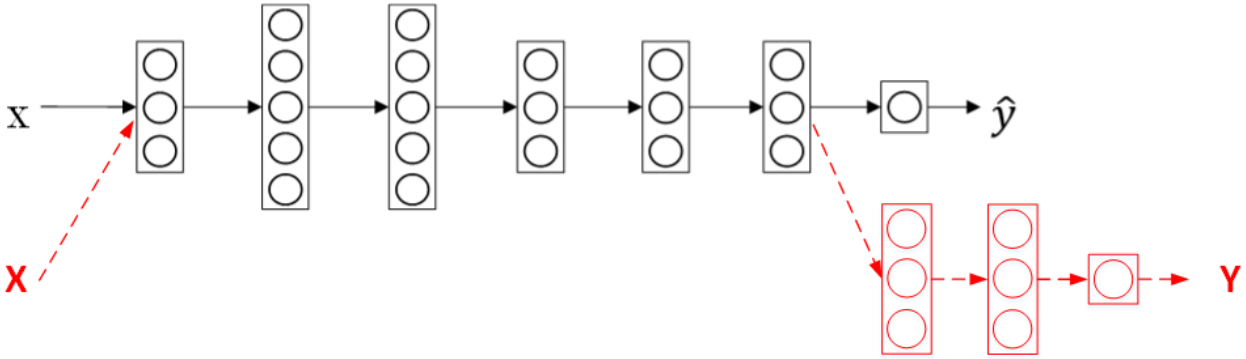


迁移学习，重新训练权重系数，如果需要构建新模型的样本数量较少，那么可以只训练输出层的权重系数$W^{[L]}$，$b^{[L]}$，保持其它层所有的权重系数$W^{[l]}$，$b^{[l]}$不变。这种做法相对来说比较简单。**如果样本数量足够多，那么也可以只保留网络结构，重新训练所有层的权重系数。**这种做法使得模型更加精确，因为毕竟样本对模型的影响最大。选择哪种方法通常由数据量决定。

如果重新训练所有权重系数，初始$W^{[l]}$，$b^{[l]}$由之前的模型训练得到，这一过程称为pre-training。之后，不断调试、优化$W^{[l]}$，$b^{[l]}$的过程称为fine-tuning。pre-training和fine-tuning分别对应上图中的黑色箭头和红色箭头。

迁移学习之所以能这么做的原因是，神经网络浅层部分能够检测出许多图片固有特征，例如图像边缘、曲线等。使用之前训练好的神经网络部分结果有助于我们更快更准确地提取X光片特征。二者处理的都是图片，而图片处理是有相同的地方，第一个训练好的神经网络已经实现如何提取图片有用特征了。 因此，即便是即将训练的第二个神经网络样本数目少，仍然可以根据第一个神经网络结构和权重系数得到健壮性好的模型。

迁移学习可以保留原神经网络的一部分，再添加新的网络层。具体问题，具体分析，可以去掉输出层后再增加额外一些神经层。



总体来说，迁移学习的应用场合主要包括三点：
Task A and B have the same input x.
You have a lot more data for Task A than Task B.
Low level features from A could be helpful for learning B.

20.Multi-task learning

多任务学习（multi-task learning）就是构建神经网络同时执行多个任务。这跟二元分类或者多元分类都不同，**多任务学习类似将多个神经网络融合在一起，用一个网络模型来实现多种分类效果。**如果有C个，那么输出y的维度是$(C,1)$。例如汽车自动驾驶中，需要实现的多任务为行人、车辆、交通标志和信号灯。如果检测出汽车和交通标志，则y为：

$$y=[0110]$$

多任务学习模型的cost function为：

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

其中，j表示任务下标，总有c个任务。对应的loss function为：

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

值得一提的是，Multi-task learning与Softmax regression的区别在于Softmax regression是single label的，即输出向量y只有一个元素为1；而Multi-task learning是multiple labels的，即输出向量y可以有多个元素为1。

多任务学习是使用单个神经网络模型来实现多个任务。实际上，也可以分别构建多个神经网络来实现。但是，**如果各个任务之间是相似问题（例如都是图片类别检测），则可以使用多任务学习模型。**另外，多任务学习中，可能存在训练样本Y某些label空白的情况，这并不影响多任务模型的训练。

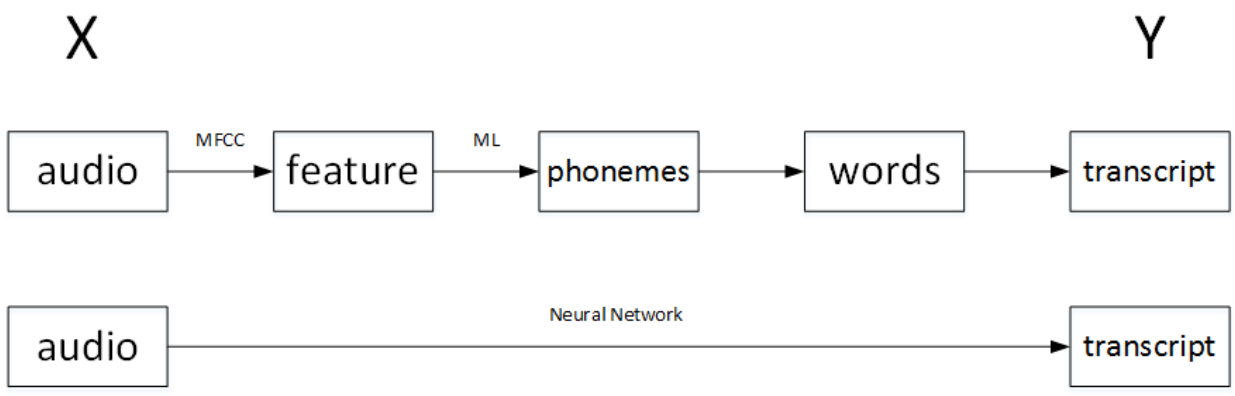
总体来说，多任务学习的应用场合主要包括三点：
Training on a set of tasks that could benefit from having shared lower-level features.
Usually: Amount of data you have for each task is quite similar.
Can train a big enough neural network to do well on all the tasks.

迁移学习和多任务学习在实际应用中，迁移学习使用得更多一些。

21.What is end-to-end deep learning

端到端（end-to-end）深度学习就是将所有不同阶段的数据处理系统或学习系统模块组合在一起，用一个单一的神经网络模型来实现所有的功能。它将所有模块混合在一起，只关心输入和输出。

以语音识别为例，传统的算法流程和end-to-end模型的区别如下：



如果训练样本足够大，神经网络模型足够复杂，那么end-to-end模型性能比传统机器学习分块模型更好。实际上，end-to-end让神经网络模型内部去自我训练模型特征，自我调节，增加了模型整体契合度。

22.Whether to use end-to-end deep learning

end-to-end深度学习有优点也有缺点。

优点：
Let the data speak
Less hand-designing of components needed

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