Course 3：结构化机器学习项目（Structuring Machine Learning Projects）

3.1 ML strategy

ML strategy

当最初得到一个深度神经网络模型时，希望从很多方面来对它进行优化，例如：
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…

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
机器学习监督式学习模型大致分成四个独立的“功能”：

- Fit training set well on cost function ，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
- Fit dev set well on cost function，优化验证集可以通过正则化，采用更多训练样本来实现
- Fit test set well on cost function，优化测试集可以通过使用更多的验证集样本来实现
- Performs well in real world，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

每一种“功能”对应不同的调节方法，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。即 early stopping 同时影响两个“功能”，不具有独立性、正交性

3.2 Setting Up your Goal

Single number evaluation metric

准确率与召回率（Precision & Recall）
准确率和召回率是广泛用于信息检索和统计学分类领域的两个度量值,用来评价结果的质量。其中精度是检索出相关文档数与检索出的文档总数的比率，衡量的是检索系统的查准率；召

回率是指检索出的相关文档数和文档库中所有的相关文档数的比率，衡量的是检索系统的查全率。
一般来说，Precision 就是检索出来的条目（比如：文档、网页等）有多少是准确的，Recall 就是所有准确的条目有多少被检索出来了。
正确率、召回率和 F 值是在鱼龙混杂的环境中，选出目标的重要评价指标。不妨看看这些指标的定义先：

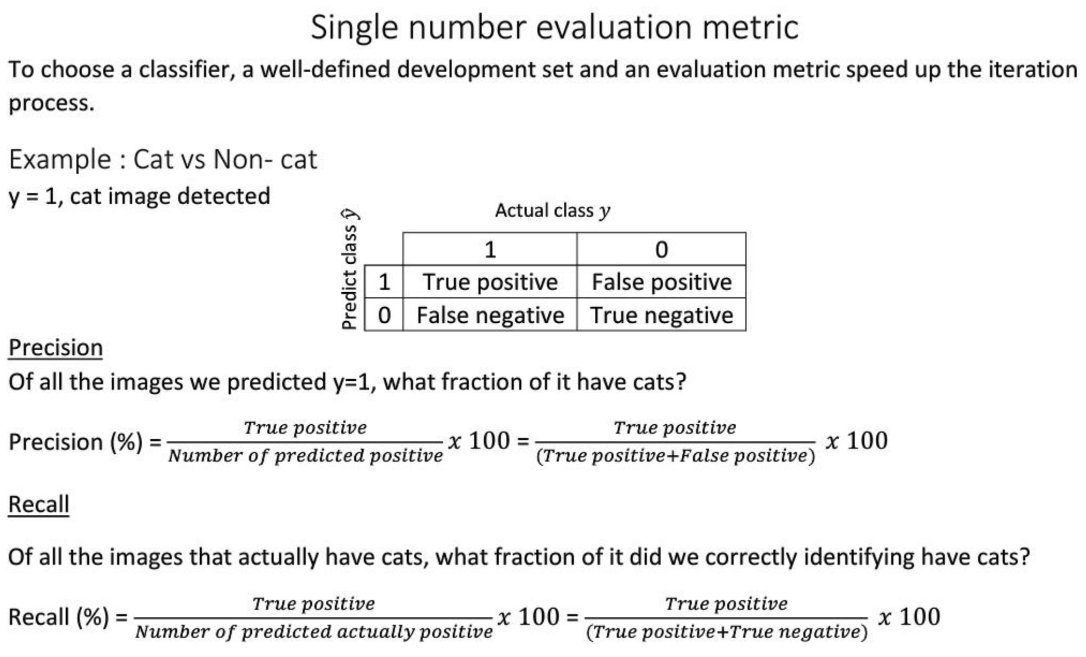
1. 正确率 = 提取出的正确信息条数 / 提取出的信息条数
2. 召回率 = 提取出的正确信息条数 / 样本中的信息条数

两者取值在 0 和 1 之间，数值越接近 1，查准率或查全率就越高。

3. F 值 = 正确率 * 召回率 * 2 / (正确率 + 召回率) （F 值即为正确率和召回率的调和平均值）

不妨举这样一个例子：某池塘有 1400 条鲤鱼，300 只虾，300 只鳖。现在以捕鲤鱼为目的。撒一大网，逮着了 700 条鲤鱼，200 只虾，100 只鳖。那么，这些指标分别如下：
正确率 = 700 / (700 + 200 + 100) = 70%
召回率 = 700 / 1400 = 50%
F 值 = 70% * 50% * 2 / (70% + 50%) = 58.3%
不妨看看如果把池子里的所有的鲤鱼、虾和鳖都一网打尽，这些指标又有何变化：
正确率 = 1400 / (1400 + 300 + 300) = 70%
召回率 = 1400 / 1400 = 100%

F 值 = 70% * 100% * 2 / (70% + 100%) = 82.35%



Single number evaluation metric

To choose a classifier, a well-defined development set and an evaluation metric speed up the iteration process.

Example : Cat vs Non- cat

y = 1, cat image detected

Predict class $\hat{y}$ \ Actual class y	1	0
1	True positive	False positive
0	False negative	True negative

Precision

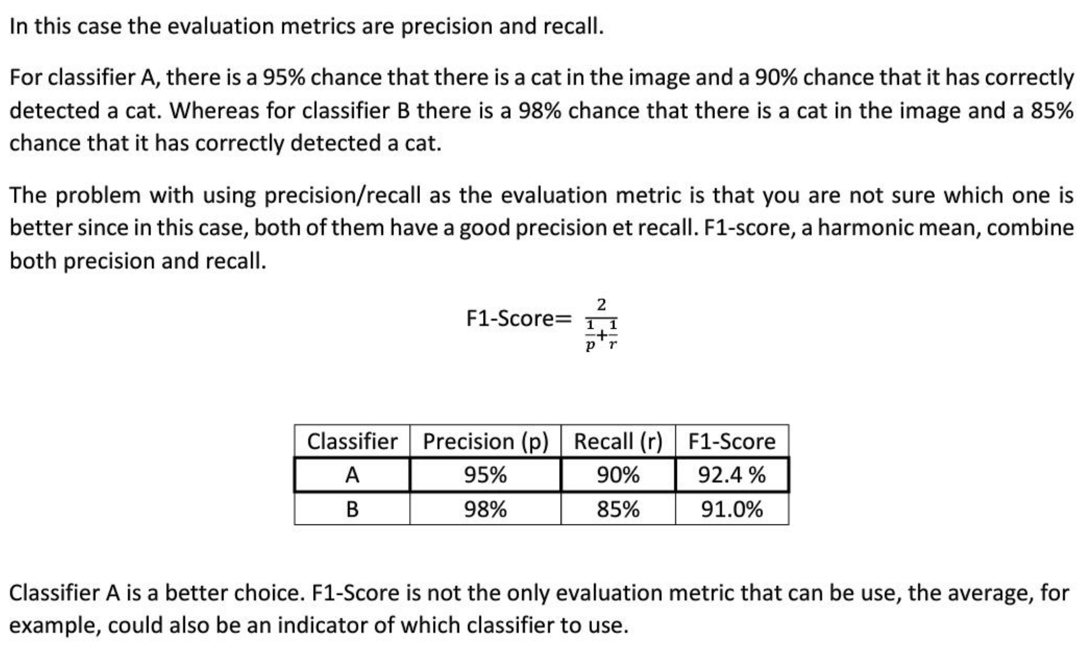
Of all the images we predicted y=1, what fraction of it have cats?

Precision (%) = $\frac{True\ positive}{Number\ of\ predicted\ positive}\ x\ 100$ = $\frac{True\ positive}{(True\ positive+False\ positive)}\ x\ 100$

Recall

Of all the images that actually have cats, what fraction of it did we correctly identifying have cats?

Recall (%) = $\frac{True\ positive}{Number\ of\ predicted\ actually\ positive}\ x\ 100$ = $\frac{True\ positive}{(True\ positive+True\ negative)}\ x\ 100$



In this case the evaluation metrics are precision and recall.

For classifier A, there is a 95% chance that there is a cat in the image and a 90% chance that it has correctly detected a cat. Whereas for classifier B there is a 98% chance that there is a cat in the image and a 85% chance that it has correctly detected a cat.

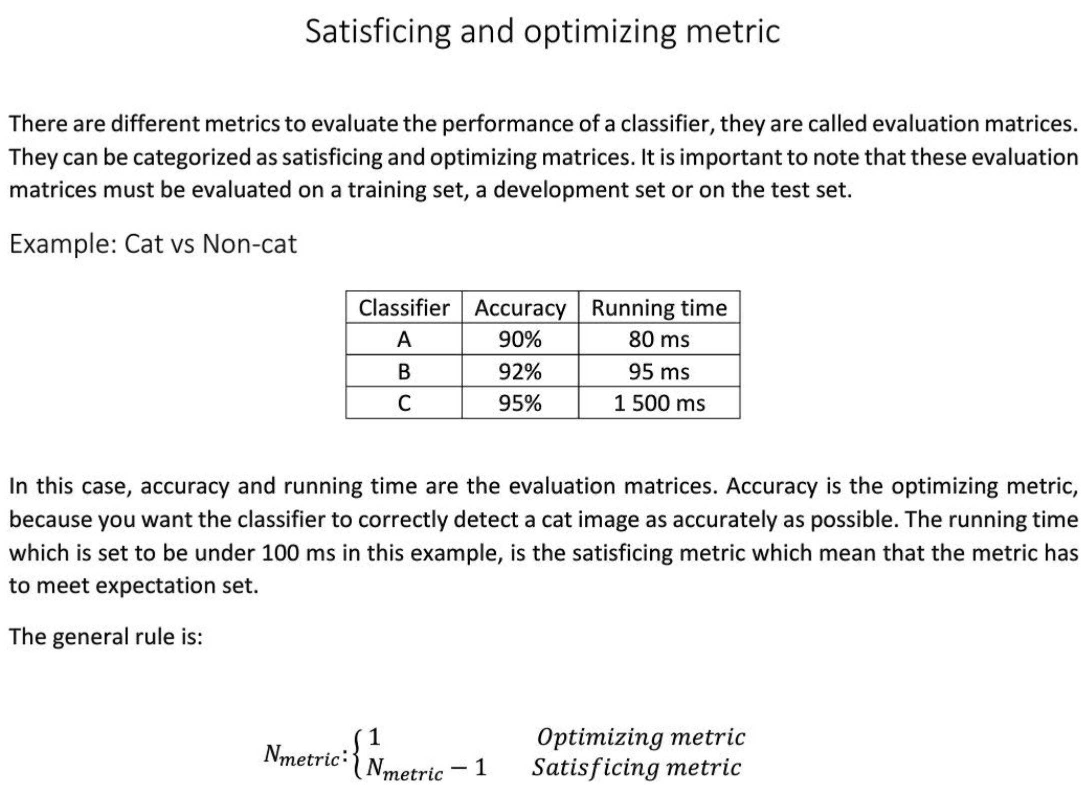
The problem with using precision/recall as the evaluation metric is that you are not sure which one is better since in this case, both of them have a good precision et recall. F1-score, a harmonic mean, combine both precision and recall.

$$\text{F1-Score} = \frac{2}{\frac{1}{p}+\frac{1}{r}}$$

Classifier	Precision (p)	Recall (r)	F1-Score
A	95%	90%	92.4 %
B	98%	85%	91.0%

Classifier A is a better choice. F1-Score is not the only evaluation metric that can be use, the average, for example, could also be an indicator of which classifier to use.

Satisficing and optimizing metrics



Satisficing and optimizing metric

There are different metrics to evaluate the performance of a classifier, they are called evaluation matrices. They can be categorized as satisficing and optimizing matrices. It is important to note that these evaluation matrices must be evaluated on a training set, a development set or on the test set.

Example: Cat vs Non-cat

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

In this case, accuracy and running time are the evaluation matrices. Accuracy is the optimizing metric, because you want the classifier to correctly detect a cat image as accurately as possible. The running time which is set to be under 100 ms in this example, is the satisficing metric which mean that the metric has to meet expectation set.

The general rule is:

$$N_{metric}:\begin{cases}1 & Optimizing\ metric\\ N_{metric}-1 & Satisficing\ metric\end{cases}$$

Accuracy 和 Running time 这两个性能不太合适综合成单值评价指标。可以将 Accuracy 作为优化指标（Optimizing metic），Running time 作为满意指标（Satisficing metic）。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，模型 C 不满足阈值条件，剔除；模型 B 相比较模型 A 而言，Accuracy 更高，性能更好

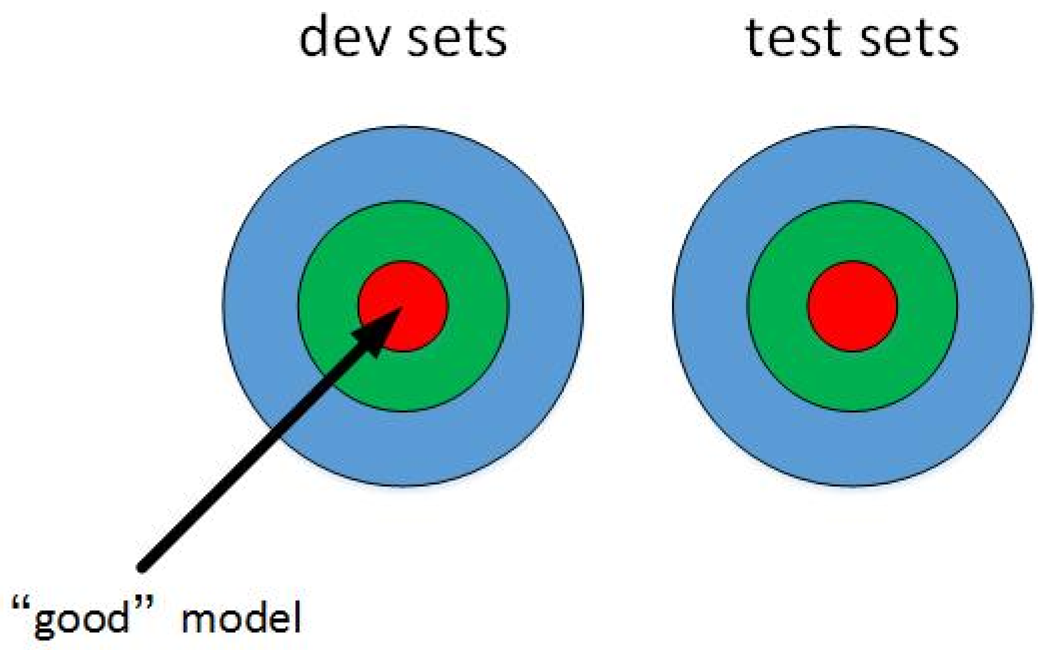
如果要考虑 N 个指标，则选择一个指标为优化指标，其他 N-1 个指标都是满足指标：

Train/dev/test distributions

训练、开发、测试集选择设置的一些规则和意见：

- ✓ 训练、开发、测试集的设置会对产品带来非常大的影响；
- ✓ 在选择开发集和测试集时要使二者来自同一分布，且从所有数据中随机选取；
- ✓ 所选择的开发集和测试集中的数据，要与未来想要或者能够得到的数据类似，即模型数据和未来数据要具有相似性；
- ✓ 设置的测试集只要足够大，使其能够在过拟合的系统中给出高方差的结果就可以，也许 10000 左右的数目足够；
- ✓ 设置开发集只要足够使其能够检测不同算法、不同模型之间的优劣差异就可以，百万大数据中 1%的大小就足够；

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，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。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，结果肯定无法射中 test sets 上的靶心位置



- 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%
- 没有 dev sets 的情况下，Train/test sets 的比例设为 70%/30%
- 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%

When to change dev/test sets and metrics
假设有两个猫的图片的分类器:

- 评估指标: 分类错误率
- 算法 A: 3% 错误率
- 算法 B： 5% 错误率

初始的评价标准是错误率， A 更好一些。实际使用时发现算法 A 会通过一些色情图片，但是 B 没有。从用户的角度来说，更 倾向选择 B 模型，虽然 B 的错误率高一些。这时候需要改变之前只使用错误率作为评价标准，考虑新的情况进行改变。如增 加色情图片的权重，增加其代价

假设开始的评估指标如下:

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

该评估指标对色情图片和非色情图片一视同仁

修改的方法，在其中加入权重 $w^{(i)}$ ：

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

$$w^{(i)} = \begin{cases} 1, & x^{(i)} \text{ is non} - \text{porn} \\ 10 \text{ or } 100, & x^{(i)} \text{ is porn} \end{cases}$$

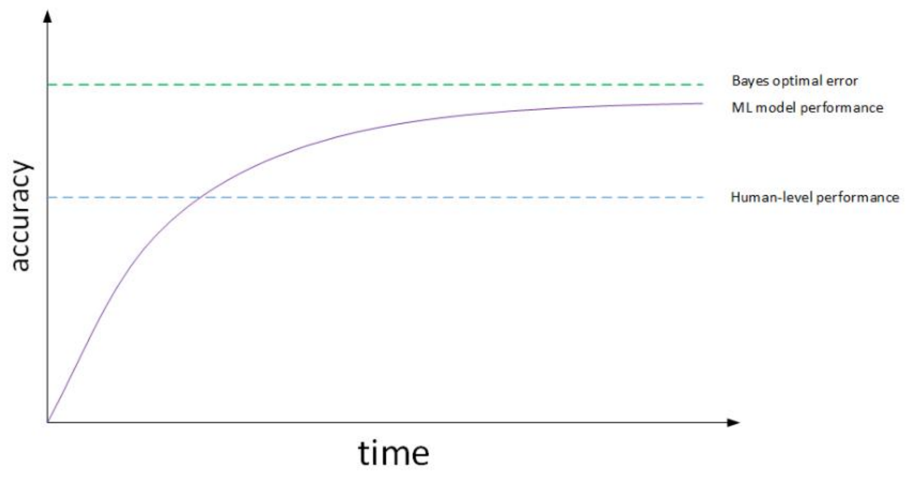
通过设置权重，当算法将色情图片分类为猫时，误差项会快速变大

概括来说，机器学习可分为两个过程:

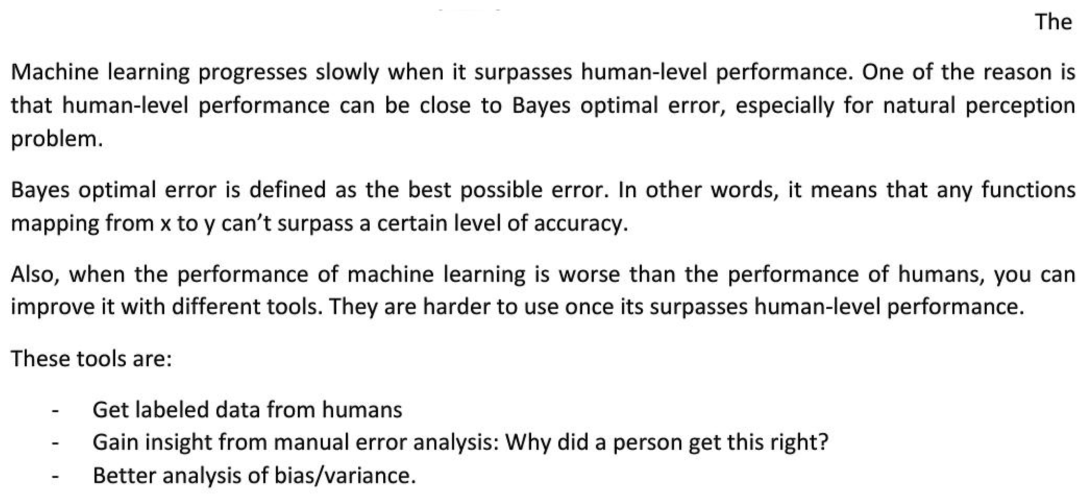
- Define a metric to evaluate classifiers
- How to do well on this metric
  第一步是找靶心，第二步是通过训练，射中靶心。但是在训练的过程中可能会根据实际情况改变算法模型的评价标准，进 行动态调整，如果评估指标无法正确评估算法的排名，则需要重新定义一个新的评估指标

3.3 Comparing to Human-level Performance

Why human-level performance?



贝叶斯最优错误率（Bayes optimal error），理论上任何模型都不能超过它，即没有任何办法设计出一个到的函数，让它能够超过一定的准确度，bayes optimal error 代表了最佳表现。



The

Machine learning progresses slowly when it surpasses human-level performance. One of the reason is that human-level performance can be close to Bayes optimal error, especially for natural perception problem.

Bayes optimal error is defined as the best possible error. In other words, it means that any functions mapping from x to y can't surpass a certain level of accuracy.

Also, when the performance of machine learning is worse than the performance of humans, you can improve it with different tools. They are harder to use once its surpasses human-level performance.

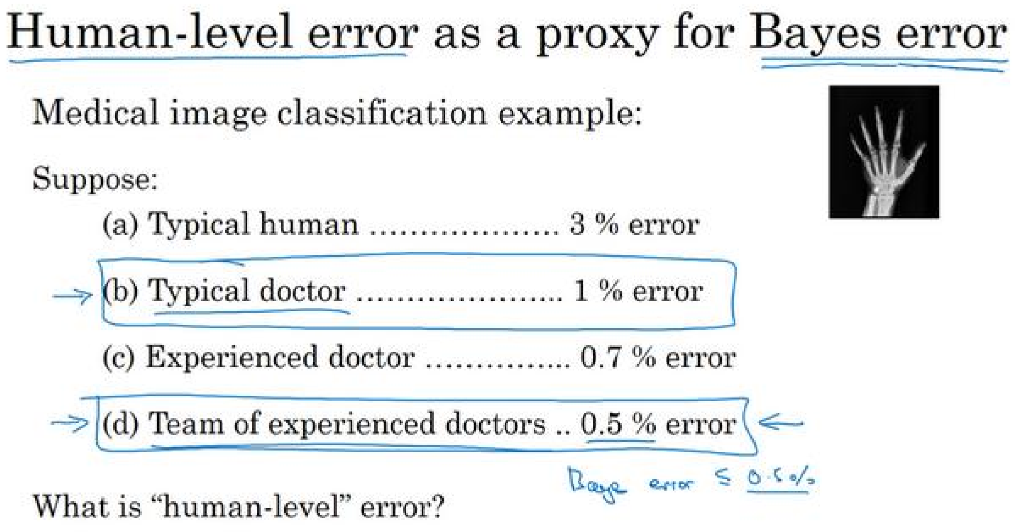
These tools are:

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

可避免偏差（Avoidable bias）

贝叶斯错误率或者对贝叶斯错误率的估计和训练错误率之间的差值称为可避免偏差
理论上是不可能超过贝叶斯错误率的，除非过拟合
训练错误率和开发错误率之前的差值，说明算法在方差问题上还有多少改善空间

Understanding human-level performance



Human-level error as a proxy for Bayes error

Medical image classification example:

Suppose:

(a) Typical human .... 3 % error

→ $\boxed{\text{(b) Typical doctor .... 1 \% error}}$

(c) Experienced doctor .... 0.7 % error

→ $\boxed{\text{(d) Team of experienced doctors .. 0.5 \% error}}$ ←

Bayes error ≤ 0.5%

What is "human-level" error?

在减小误诊率的背景下，人类水平误差在这种情形下应定义为：0.5% error。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

如果在为了部署系统或者做研究分析的背景下，也许超过一名普通医生即可，即人类水平误差在这种情形下应定义为：1% error

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。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。

Scenario A

In this case, the choice of human-level performance doesn't have an impact. The avoidable bias is between 4%-4.5% and the variance is 1%. Therefore, the focus should be on bias reduction technique.

Scenario B

In this case, the choice of human-level performance doesn't have an impact. The avoidable bias is between 0%-0.5% and the variance is 4%. Therefore, the focus should be on variance reduction technique.

Scenario C

In this case, the estimate for Bayes error has to be 0.5% since you can't go lower than the human-level performance otherwise the training set is overfitting. Also, the avoidable bias is 0.2% and the variance is 0.1%. Therefore, the focus should be on bias reduction technique.

Summary of bias/variance with human-level performance

- Human - level error – proxy for Bayes error
- If the difference between human-level error and the training error is bigger than the difference between the training error and the development error. The focus should be on bias reduction technique
- If the difference between training error and the development error is bigger than the difference between the human-level error and the training error. The focus should be on variance reduction technique

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

基本假设：

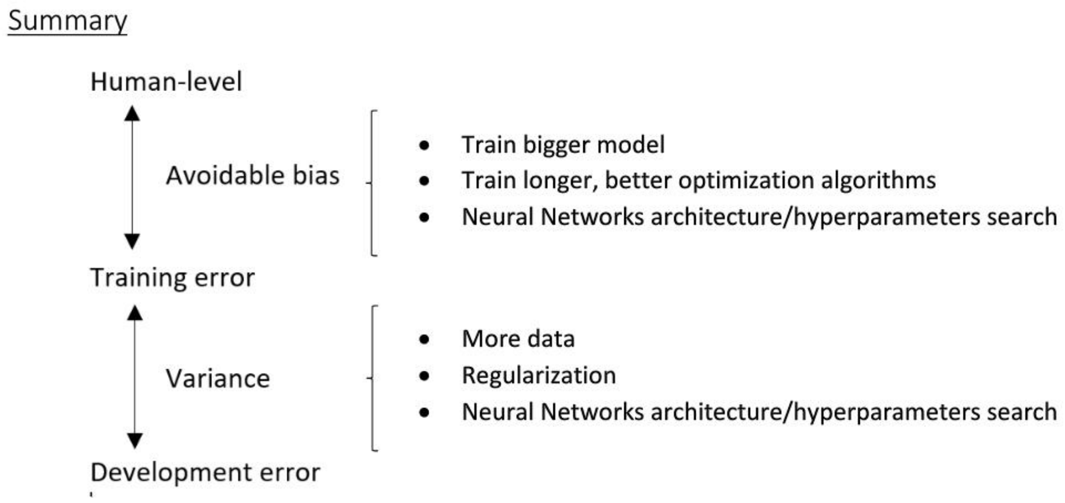
- 模型在训练集上有很好的表现；
- 模型推广到开发和测试集啥也有很好的表现

减少可避免偏差

- 训练更大的模型
- 训练更长时间、训练更好的优化算法（Momentum、RMSprop、Adam）
- 寻找更好的网络架构（RNN、CNN）、寻找更好的超参数

减少方差

- 收集更多的数据
- 正则化（L2、dropout、数据增强）
- 寻找更好的网络架构（RNN、CNN）、寻找更好的超参数



- Train bigger model
- Train longer, better optimization algorithms
- Neural Networks architecture/hyperparameters search

- More data
- Regularization
- Neural Networks architecture/hyperparameters search

3.4 Error analysis

人工检查一下算法

进行错误分析，应该找一组错误样本，可能在开发集或者测试集，观察错误标记的样本，看看假阳性（false positives）和假阴性（false negatives），统计属于不同错误类型的错误数量。在这个过程中，可能会得到启发，归纳出新的错误类型，通过统计不同错误标记类型的百分比，可以发现哪些问题需要优先解决

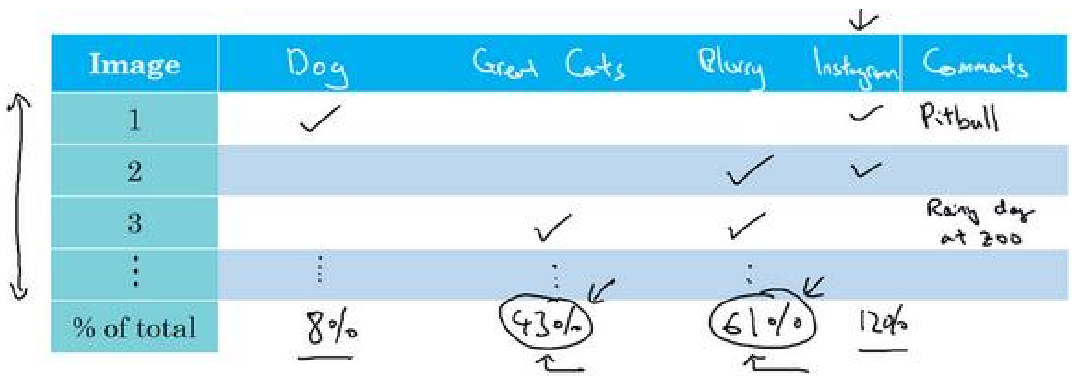


Image	Dog	Great Cats	Blurry	Instagram	Comments
1	✓			✓	Pitbull
2			✓	✓	
3		✓	✓		Rainy day at zoo
⋮	⋮	⋮	⋮		
% of total	8%	43%	61%	12%	

Cleaning up Incorrectly labeled data

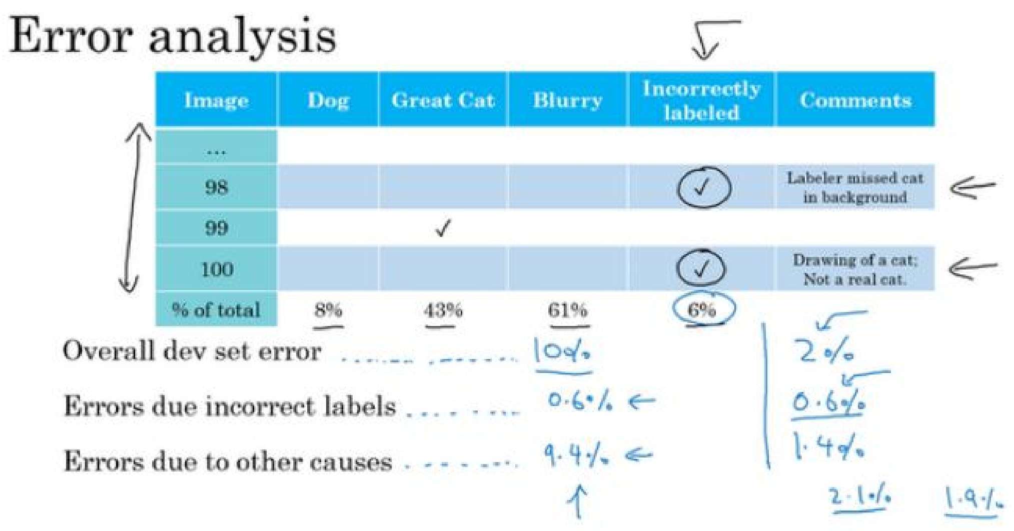
训练集：深度学习算法对于训练集中的随机错误是相当健壮的（robust）。只要这些错误样本离随机错误不太远，有时可能做标记的人没有注意或者不小心，按错键了，如果错误足够随机，放着这些错误不管可能也没问题，而不要花太多时间修复它们，只要总数据集足够大，实际错误率可能不会太高
深度学习算法对随机误差很健壮，但对系统性的错误没那么健壮。如果做标记的人一直把白色的狗标记成猫，那就成问题。因为分类器学习之后，会把所有白色的狗都分类为猫。但随机错误或近似随机错误，对于大多数深度学习算法来说不成问题

开发集和测试集有标记出错的样本：在错误分析时，添加一个额外的列，统计标签 y=1 错误的样本数。统计因为标签错误所占的百分比，解释为什么学习算法做出和数据集的标记不一样的预测

是否值得修正 6%标记出错的样本：

- 如果标记错误严重影响了在开发集上评估算法的能力，应该去花时间修正错误的标签
- 如果没有严重影响到用开发集评估成本偏差的能力，不应该花时间去处理

看 3 个数字来确定是否值得去人工修正标记出错的数据：
- 看整体的开发集错误率，系统达到了 90%整体准确度，10%错误率，应该看错误标记引起的错误的数量或者百分比。6%的错误来自标记出错，10%的 6%是 0.6%，剩下的占 9.4%，是其他原因导致的，比如把狗误认为猫，大猫图片。即有 9.4%错误率需要集中精力修正，而标记出错导致的错误是总体错误的一小部分而已，应该看其他原因导致的错误
- 错误率降到了 2%，但总体错误中的 0.6%还是标记出错导致的。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%

2.1%　1.9%

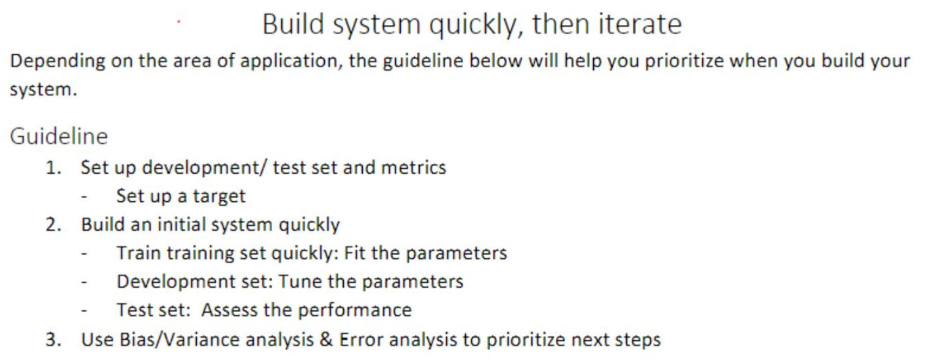
如果决定要去修正开发集数据，手动重新检查标签，并尝试修正一些标签，这里还有一些额外的方针和原则需要考虑：

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，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，希望算法能够推广到测试集上，这样能够更高效的在来自同一分布的开发集和测试集上迭代
2. 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。可以让一个人来仔细检查这些标签，但必须同时检查开发集和测试集
3. 要同时检验算法判断正确和判断错误的样本，如果只修正算法出错的样本，算法的偏差估计可能会变大，会让算法有一点不公平的优势

修正训练集中的标签相对没那么重要，如果训练集来自稍微不同的分布，对于这种情况学习算法其实相当健壮，通常是一件很合理的事情
几个建议：

4. 构造实际系统时，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
5. 搭建机器学习系统时，花时间亲自检查数据非常值得，可以帮你找到需要优先处理的任务，然后确定应该优先尝试哪些想法，或者哪些方向

Build your first system quickly, then iterate



Build system quickly, then iterate

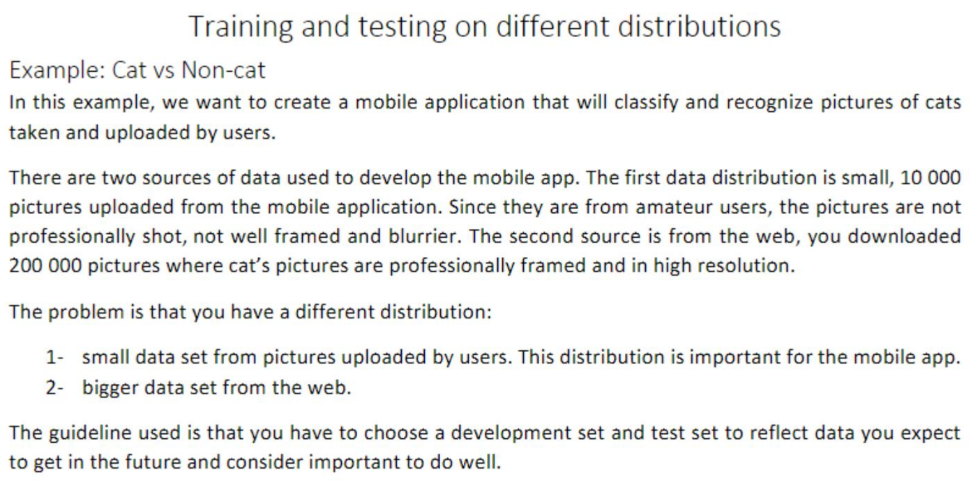
Depending on the area of application, the guideline below will help you prioritize when you build your system.

Guideline

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

3.5 Mismatched training and Dev/Test set

Training and testing on different distributions



Training and testing on different distributions

Example: Cat vs Non-cat

In this example, we want to create a mobile application that will classify and recognize pictures of cats taken and uploaded by users.

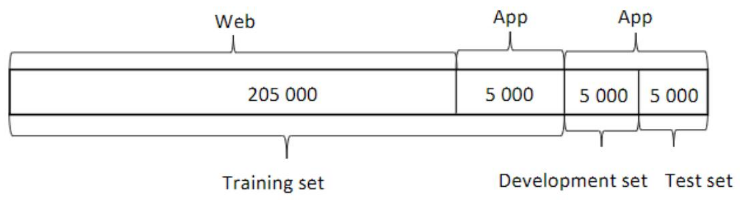
There are two sources of data used to develop the mobile app. The first data distribution is small, 10 000 pictures uploaded from the mobile application. Since they are from amateur users, the pictures are not professionally shot, not well framed and blurrier. The second source is from the web, you downloaded 200 000 pictures where cat's pictures are professionally framed and in high resolution.

The problem is that you have a different distribution:

1- small data set from pictures uploaded by users. This distribution is important for the mobile app.
2- bigger data set from the web.

The guideline used is that you have to choose a development set and test set to reflect data you expect to get in the future and consider important to do well.

The data is split as follow:



The advantage of this way of splitting up is that the target is well defined.

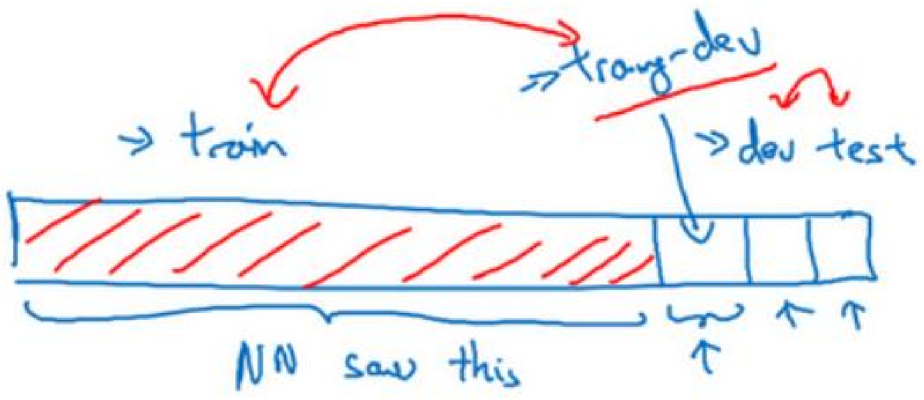
The disadvantage is that the training distribution is different from the development and test set distributions. However, this way of splitting the data has a better performance in long term.

Bias and Variance with mismatched data distributions

如何分析误差是由于数据不匹配造成还是由于 variance， bias 比较大造成

- 算法只见过训练集数据，没见过开发集数据（方差）
- 开发集数据来自不同的分布

随机打散训练集，分出一部分训练集作为训练-开发集（training-dev），训练集、训练-开发集来自同一分布



只在训练集训练神经网络，不让神经网络在训练-开发集上跑后向传播。为了进行误差分析，应该看分类器在训练集上的误差、训练-开发集上的误差、开发集上的误差

- 假设训练误差是 1%，训练-开发集上的误差是 9%，开发集误差是 10%，存在方差，因为训练-开发集的错误率是在和训练集来自同一分布的数据中测得的，尽管神经网络在训练集中表现良好，但无法泛化到来自相同分布的训练-开发集
- 假设训练误差为 1%，训练-开发误差为 1.5%，开发集错误率 10%。方差很小，当转到开发集时错误率大大上升，是数据不匹配的问题
- 如果训练集误差是 10%，训练-开发误差是 11%，开发误差为 12%，人类水平对贝叶斯错误率的估计大概是 0%，存在可避免偏差问题
- 如果训练集误差是 10%，训练-开发误差是 11%，开发误差是 20%，有两个问题

  可避免偏差问题

  数据不匹配问题

Human level 4%和 Training error 7%衡量了可避免偏差大小，Training error 7%和 Training-dev error 10%衡量了方差大小，Training-dev error 10%和 Dev/Test dev 6%衡量了数据不匹配问题的大小

Bias and variance with mismatched data distributions

Example: Cat classifier with mismatch data distribution

When the training set is from a different distribution than the development and test sets, the method to analyze bias and variance changes.

	Classification error (%)					
	Scenario A	Scenario B	Scenario C	Scenario D	Scenario E	Scenario F
Human (proxy for Bayes error)	0	0	0	0	0	4
Training error	1	1	1	10	10	7
Training-development error	-	9	1.5	11	11	10
Development error	10	10	10	12	20	6
Test error	-	-	-	-	-	6

Scenario A

If the development data comes from the same distribution as the training set, then there is a large variance problem and the algorithm is not generalizing well from the training set.

However, since the training data and the development data come from a different distribution, this conclusion cannot be drawn. There isn't necessarily a variance problem. The problem might be that the development set contains images that are more difficult to classify accurately.

When the training set, development and test sets distributions are different, two things change at the same time. First of all, the algorithm trained in the training set but not in the development set. Second of all, the distribution of data in the development set is different.

It's difficult to know which of these two changes what produces this 9% increase in error between the training set and the development set. To resolve this issue, we define a new subset called training-development set. This new subset has the same distribution as the training set, but it is not used for training the neural network.

Scenario B

The error between the training set and the training- development set is 8%. In this case, since the training set and training-development set come from the same distribution, the only difference between them is the neural network sorted the data in the training and not in the training development. The neural network is not generalizing well to data from the same distribution that it hadn't seen before

Therefore, we have really a variance problem.

Scenario C

In this case, we have a mismatch data problem since the 2 data sets come from different distribution.

Scenario D

In this case, the avoidable bias is high since the difference between Bayes error and training error is 10 %.

Scenario E

In this case, there are 2 problems. The first one is that the avoidable bias is high since the difference between Bayes error and training error is 10 % and the second one is a data mismatched problem.

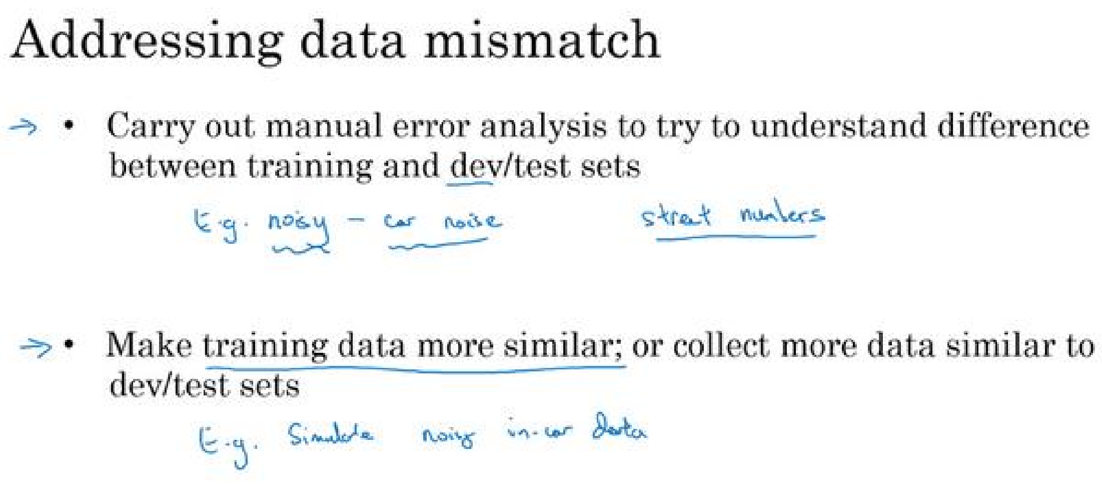
Scenario F

Development should never be done on the test set. However, the difference between the development set and the test set gives the degree of overfitting to the development set.

General formulation

$$\begin{tikzcd}
\text{Bayes error} \arrow[d, leftrightarrow, "\text{Avoidable Bias}"] \\
\text{Training set error} \arrow[d, leftrightarrow, "\text{Variance}"] \\
\text{Development - Training set error} \arrow[d, leftrightarrow, "\text{Data mismatch}"] \\
\text{Development set error} \arrow[d, leftrightarrow, "\text{Degree of overfitting to the development set}"] \\
\text{Test set error}
\end{tikzcd}$$

Addressing data mismatch



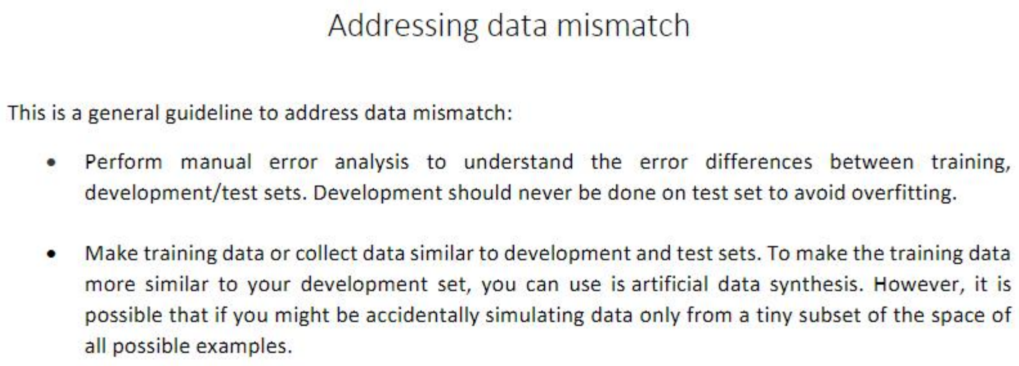
Addressing data mismatch

→ • 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



Addressing data mismatch

This is a general guideline to address data mismatch:

- Perform manual error analysis to understand the error differences between training, development/test sets. Development should never be done on test set to avoid overfitting.
- Make training data or collect data similar to development and test sets. To make the training data more similar to your development set, you can use is artificial data synthesis. However, it is possible that if you might be accidentally simulating data only from a tiny subset of the space of all possible examples.

3.6 Learning from multiple tasks

迁移学习

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，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）

Transfer Learning

Transfer learning refers to using the neural network knowledge for another application.

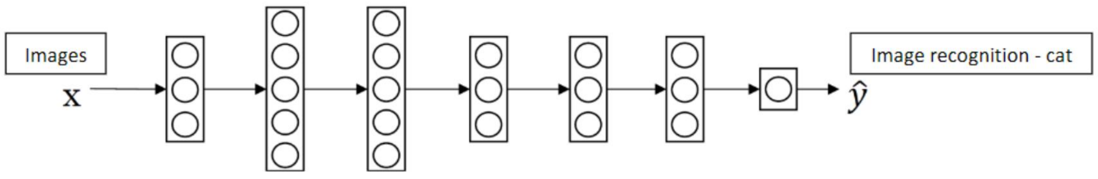
When to use transfer learning

- Task A and B have the same input x
- A lot more data for Task A than Task B
- Low level features from Task A could be helpful for Task B

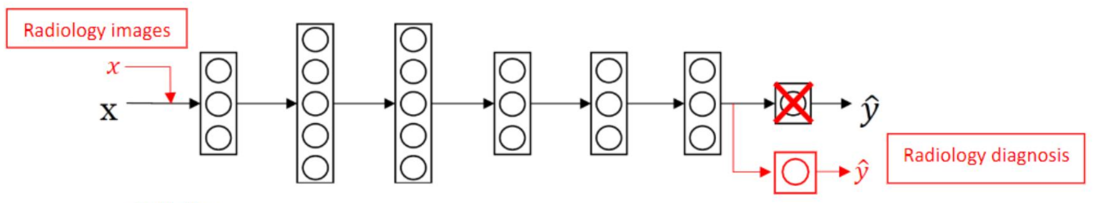
Example 1: Cat recognition - radiology diagnosis

The following neural network is trained for cat recognition, but we want to adapt it for radiology diagnosis. The neural network will learn about the structure and the nature of images. This initial phase of training on image recognition is called pre-training, since it will pre-initialize the weights of the neural network. Updating all the weights afterwards is called fine-tuning.

For cat recognition
Input x: image
Output y – 1: cat, 0: no cat



Radiology diagnosis
Input x: Radiology images – CT Scan, X-rays
Output y :Radiology diagnosis – 1: tumor malign, 0: tumor benign



Guideline

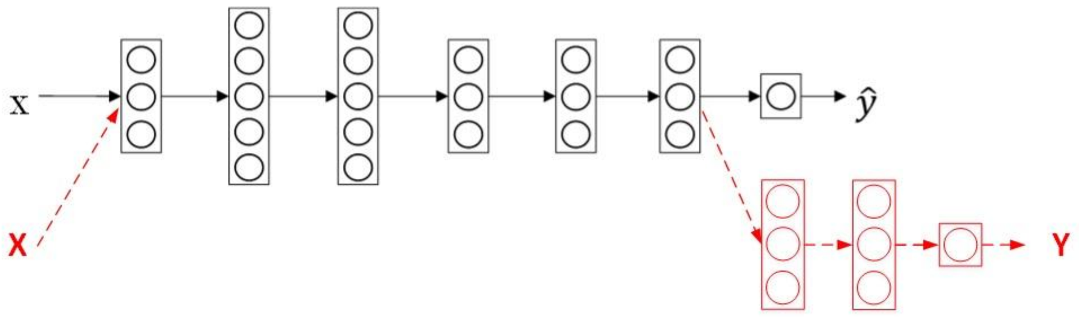
- Delete last layer of neural network
- Delete weights feeding into the last output layer of the neural network
- Create a new set of randomly initialized weights for the last layer only
- New data set (x, y)

- 如果需要构建新模型的样本数量较少，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
- 如果样本数量足够多，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解，因为样本对模 型的影响最大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。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，可以去掉输出层后再增加额外一些神经层



多任务学习（Multi-task learning）

在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，希望每个任务都能帮到其他所有任务

Multi-task learning 与 Softmax regression 的区别在于：
Multi-task learning 是 multiple labels 的，即输出向量 y 可以有多个元素为 1
Softmax regression 是 single label 的，即输出向量 y 只有一个元素为 1
多任务学习当三件事为真时有意义的：

- 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
- 如果每个任务的数据量很接近，这个准则没那么绝对，不一定对
- 想要从多任务学习得到很大性能提升，其他任务加起来必须要有比单个任务大得多的数据量
- 多任务学习会降低性能的唯一情况是神经网络还不够大。但如果可以训练一个足够大的神经网络，多任务学习肯定不会或者很少会降低性能

在实践中，多任务学习的使用频率要低于迁移学习。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。不过在计算机视觉领域，物体检测这个例子是最显著的例外情况

Example: Simplified autonomous vehicle

The vehicle has to detect simultaneously several things: pedestrians, cars, road signs, traffic lights, cyclists, etc. We could have trained four separate neural networks, instead of train one to do four tasks. However, in this case, the performance of the system is better when one neural network is trained to do four tasks than training four separate neural networks since some of the earlier features in the neural network could be shared between the different types of objects.

The input $x^{(i)}$ is the image with multiple labels

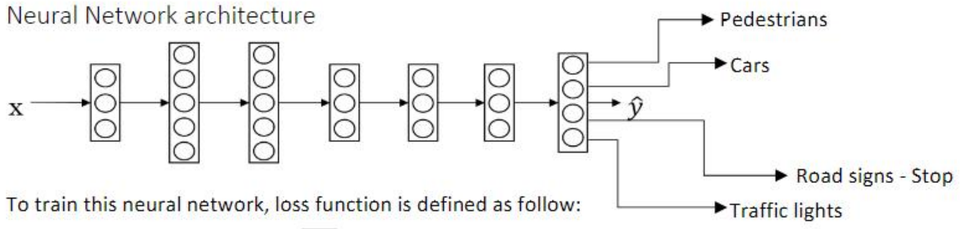
The output $y^{(i)}$ has 4 labels which are represents:

$$y^{(i)} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \begin{array}{l} \text{Pedestrians} \\ \text{Cars} \\ \text{Road signs - Stop} \\ \text{Traffic lights} \end{array}$$

$$\downarrow$$

$$Y = \begin{bmatrix} | & | & | & | \\ y^{(1)} & y^{(2)} & y^{(3)} & y^{(4)} \\ | & | & | & | \end{bmatrix} \qquad \begin{array}{l} Y = (4, m) \\ Y = (4,1) \end{array}$$

Neural Network architecture



To train this neural network, loss function is defined as follow:

$$-\frac{1}{m}\sum_{i=1}^{m}\boxed{\sum_{j=1}^{4}}\left(y_j^{(i)}\log\left(\hat{y}_j^{(i)}\right)+\left(1-y_j^{(i)}\right)\log\left(1-\hat{y}_j^{(i)}\right)\right)$$

Also, the cost can be compute such as it is not influenced by the fact that some entries are not labeled. Example:

$$Y = \begin{bmatrix} 1 & 0 & ? & ? \\ 0 & 1 & ? & 0 \\ 0 & 1 & ? & 1 \\ ? & 0 & 1 & 0 \end{bmatrix}$$

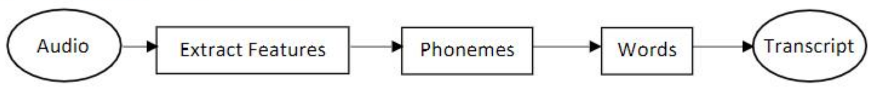
End-to-end deep learning

What is end-to-end deep learning

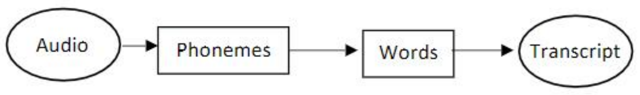
End-to-end deep learning is the simplification of a processing or learning systems into one neural network.

Example - Speech recognition model

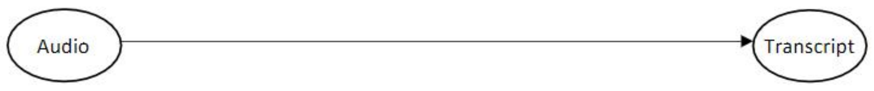
The traditional way - small data set



The hybrid way - medium data set



The End-to-End deep learning way – large data set



End-to-end deep learning cannot be used for every problem since it needs a lot of labeled data. It is used mainly in audio transcripts, image captures, image synthesis, machine translation, steering in self-driving cars, etc.

Whether to use end-to-end deep learning

Before applying end-to-end deep learning, you need to ask yourself the following question: Do you have enough data to learn a function of the complexity needed to map x and y?

Pro:

- Let the data speak
  - By having a pure machine learning approach, the neural network will learn from x to y. It will be able to find which statistics are in the data, rather than being forced to reflect human preconceptions.

- Less hand-designing of components needed
  - It simplifies the design work flow.

Cons:

- Large amount of labeled data
  - It cannot be used for every problem as it needs a lot of labeled data.

- Excludes potentially useful hand-designed component
  - Data and any hand-design's components or features are the 2 main sources of knowledge for a learning algorithm. If the data set is small than a hand-design system is a way to give manual knowledge into the algorithm.