

## Why ML Strategy?

There are TOO MANY ideas to try, which is impractical. Thus, **Strategies** are needed.

- Collect more diverse data.
- Try different regularizations.
- Train more epochs.
- Change the architecture of neural networks.
- Train with different optimizers.
- Try bigger /smaller networks.

## Orthogonalization

Orthogonalization or orthogonality is a system design property that assures that modifying an instruction or a component of an algorithm will not create or propagate side effects to other components of the system. It becomes easier to verify the algorithms independently from one another, it reduces testing and development time. 就像是 Debug 虛擬板必須把一塊複雜的電路板分成子模組，在分別對各子模組 Debug。最要不得的是 Couple 在一起的 Bug.

When a supervised learning system is designed, these are the 4 assumptions that need to be true and orthogonal.

1. Fit training set well in cost function ( $\approx$  human-level performance)
  - ↓ - If it doesn't fit well, the use of a bigger neural network or switching to a better optimization algorithm might help.
2. Fit development set well on cost function
  - ↓ - If it doesn't fit well, regularization or using bigger training set might help.
3. Fit test set well on cost function
  - ↓ - If it doesn't fit well, the use of a bigger development set might help
4. Performs well in real world
  - If it doesn't perform well, the development test set is not set correctly or the cost function is not evaluating the right thing.

Early Stopping simultaneously affects two things 1. and 2. which couples two issues.

## 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

		Actual class $y$	
		1	0
Predict class $\hat{y}$	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?

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

### Recall

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

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

Let's compare 2 classifiers A and B used to evaluate if there are cat images:

Classifier	Precision (p)	Recall (r)
A	95%	90%
B	98%	85%

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.

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

*Another Example*

For each algorithm, there are too many numbers to evaluate. So, we need a single value to help us make decisions. Eg, use **Average** to evaluate.

## 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

optimizing →  
satisficing: As long as it's below 100 ms, then I don't care.

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 metrics. 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:

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

It's not always easy to combine all the evaluations/numbers we care about into a single real number! In fact, I would argue it's hard. Eg, when generating disparity map, there are occlusions, disparity, confidence. What is the single value that can represent these three numbers.

Another example: Let's say I want to build a TriggerWord system.

Alexa  
OK Google  
Hi Siri  
你好 百度

Thus, we want to maximize accuracy. Optimizing metric  
And one false positive every 24 hours is allowed. Satisficing metric

# Train / Dev / Test Sets Distribution

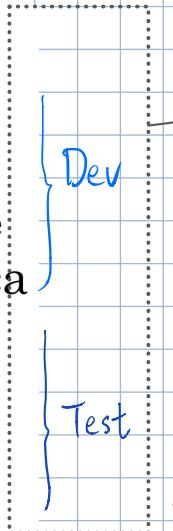
aka, hold-out cross validation set

## Cat Classification Dev / Test Sets

The cat classification APP is applied to the following regions:

Regions:

- US
- UK
- Other Europe
- South America
- India
- China
- Other Asia
- Australia



→ This is a bad choice, because Dev and Test sets are coming from different distributions.

→ The better way is to randomly shuffle the whole dataset into dev/test set.

These two must come from the Same Distribution.

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

Size of the Dev and Test Sets:

Train	Test
70 %	30 %

Set the dev set to be big enough to detect differences in algorithm/models you're trying out.

Train	Dev	Test
60 %	20 %	20 %

100  
1000  
10<sup>5</sup>

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

Train	DT
98 %	1   1

10<sup>6</sup>

First Cat dataset example:

The Metric: cat v.s. non-cat classification error

Algorithm A: 3% error + Pomographic  
Algorithm B: 5% error

Dev + Metric : Prefer A

App / Users : Prefer B

In this case, the old evaluation metric **can't** be used.  
The new evaluation metric could be **redefined** as:

$$\frac{1}{\sum_i \text{weight}(i)} \times \sum_{i=1}^{m_{\text{Dev}}} \text{weight}(i) \sum_{j=1}^{y^{(i)}} \mathbb{I}\{y_{\text{pred}}^{(i)} \neq y^{(i)}\},$$

$$\text{weight}(i) = \begin{cases} 1, & \text{if } x^{(i)} \text{ is non-pom.} \\ 10, & \text{if } x^{(i)} \text{ is pom.} \end{cases}$$

Orthogonalization for cat pictures: anti-porn

Two Steps:

1. Define a metric to evaluate classifiers.  
*Plan target*
2. Worry separately about how to do well on this metric.  
*Shoot at target*

Second Cat dataset example:

The Metric: cat v.s. non-cat classification error

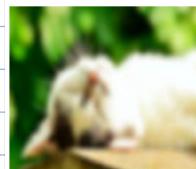
Algorithm A: 3% error

Algorithm B: 5% error

Development  
Dev/test



Application  
User images



The model works well on all the data in development phase, but performs badly in application. There are many reasons to this problem. Eg, users' data is low resolution.

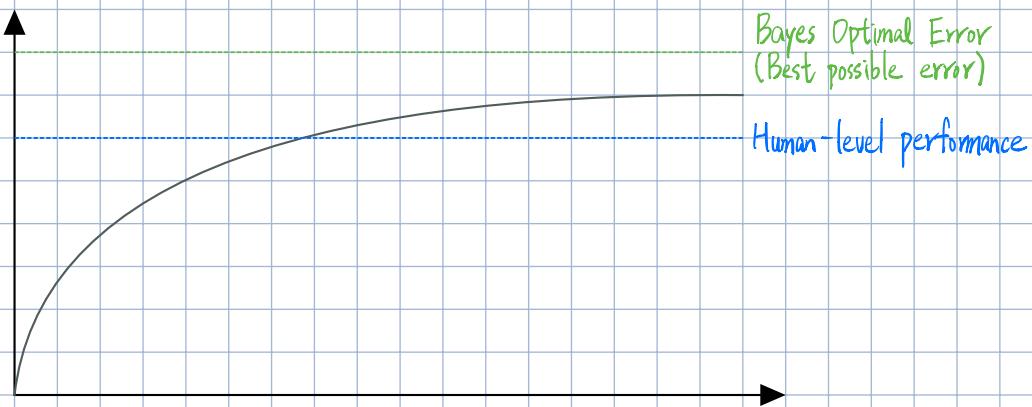
In this case, change the metric and/or dev/test set, such as, including low-resolution imagery in the Dev/Test set.

## Comparing to Human-level Performance

Why Human-level Performance?

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

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



The perfect accuracy may not be 100% due to noises in data. Such as noises in audio or in image are so big that it's just impossible to tell what's in the data.

Avoidable Bias: The difference between Bayes Error and the training error.

Cat Classification example:

Human Error ( $\approx$  Bayes Error):

1%

8%

Bias

Training Error:

7.5%

8%

Variance

Dev Error:

10%

10%

Avoidable Bias = 0.5%

Focus on reducing  
Bias

Focus on reducing  
Variance

Use human-level error as a proxy for Bayes Error

# 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

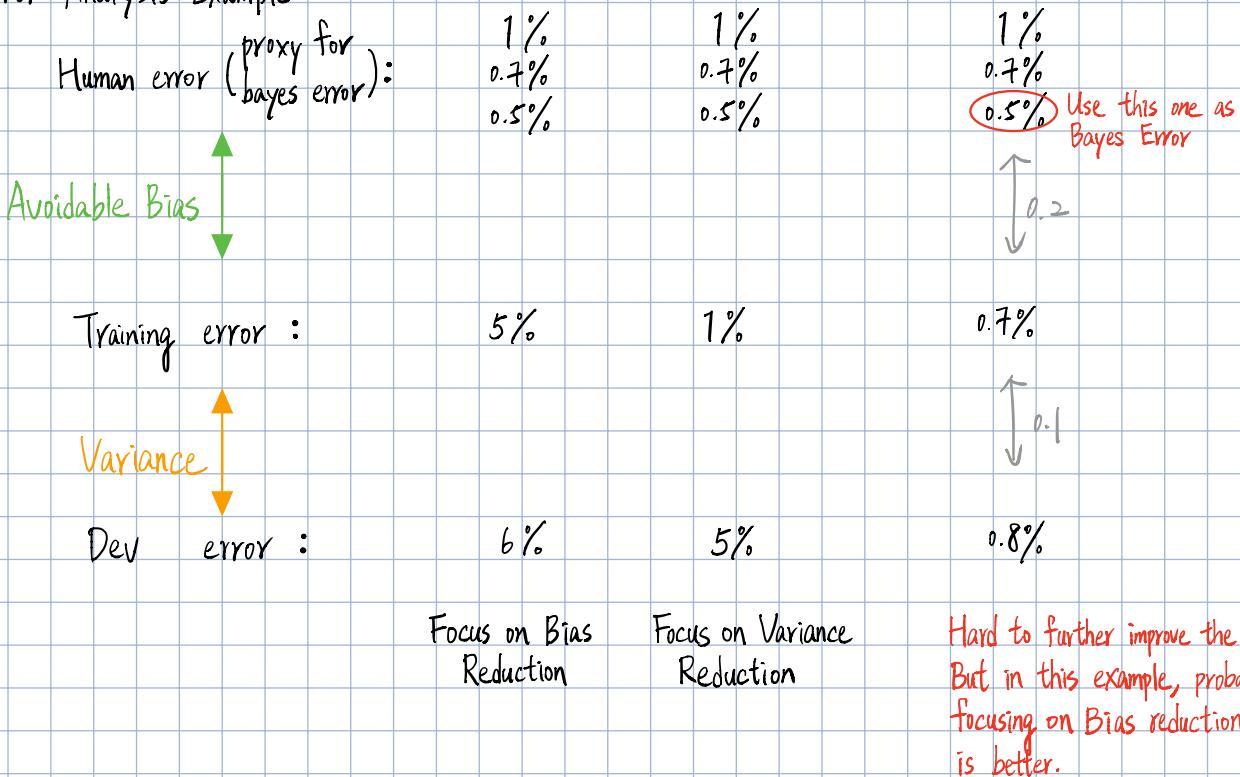


Theoretically the bayes error could be  $\leq 0.5$ .

But it doesn't mean I can't deploy my APP if the error of my APP is bigger than 0.5. Good enough is good enough.

What is “human-level” error?

## Error Analysis Example:



## Surpassing Human-level Performance

Team of humans error:	0.5%	0.5%
One human error:	1%	1%
Training error:	0.6%	0.3%
Dev error:	0.8%	0.4%

What's avoidable Bias?  $0.1\% (0.6 - 0.5)$

Assuming the Bayes Optimal Error is 0.1%, it's becoming harder to further improve the model when the model surpasses human-level performance.

It sounds like it's science fiction. There are already many problems where ML

significantly surpasses human-level performance. However, it's harder for Natural Perception Tasks, such as:

Such as:

- Online advertising
- Product recommendation
- Logistics
- Loan approvals

Notice these are all Structured Data.

- Computer Vision
- Speech Recognition
- Natural Language Problem.

# The two fundamental assumptions of supervised learning

1. You can fit the training set pretty well.
2. The training set performance generalizes pretty well to the dev/test set.

# Reducing (avoidable) bias and variance

