

## Week 1: ML Strategy

### Why ML Strategy

- Why do we need an ML strategy? Suppose we have our cat classifier from the last class and it has 90% accuracy but this is not good enough for our application. There is a number of things we can do, listed below. However, some of these ideas will take a long time (i.e. collect more data). Assuming we don't have months to waste, wouldn't it be nice to have quick and effective ways to evaluate which of these ideas are worth trying and which ones are not? That is what we are going to learn in this class.

### Motivating example



#### Ideas:

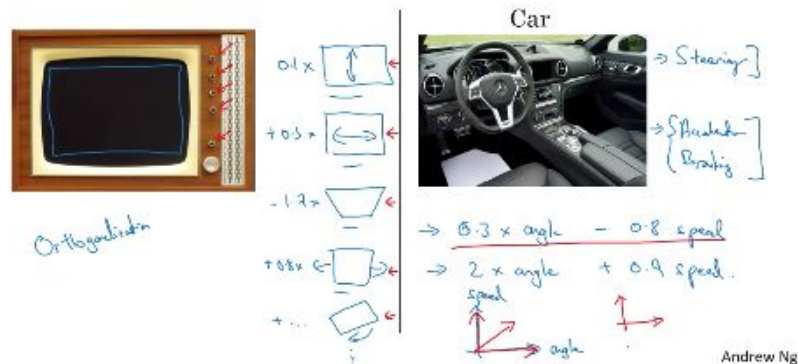
- Collect more data ←
- Collect more diverse training set
- Train algorithm longer with gradient descent
- Try Adam instead of gradient descent
- Try bigger network
- Try smaller network
- Try dropout
- Add  $L_2$  regularization
- Network architecture
  - Activation functions
  - # hidden units
  -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

Andrew Ng

### Orthogonalization

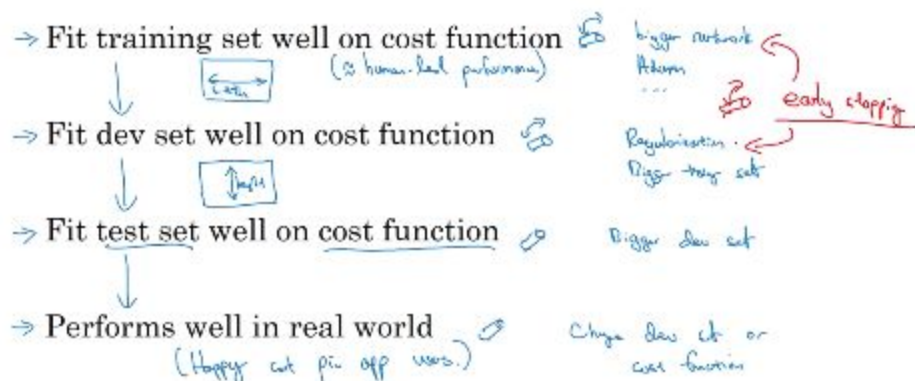
- Orthogonalization is the art of knowing/designing one 'knob' that tunes one thing. Another way of saying it is an action controlling one dimension. In the example below we have examples of 'bad' knobs where a knob controls more than one dimension. This makes it harder for the user to operate the TV or the car.

## TV tuning example



- How does this relate to machine learning? It is relevant because during each ML phase we have a knob or set of knobs that we can use to make our algorithm better. In the 4 stages of ML we have:
  - Fit training set well with cost function and the relevant knobs to use here are a bigger network or an optimization function like Adam.
  - Fit dev set well with cost function and the relevant knobs are regularization or a bigger training set
  - Fit test set well with cost function and the relevant knob is a bigger dev set
  - Model performs well in the real world and the relevant knobs are changing the dev set or the cost function.

## Chain of assumptions in ML

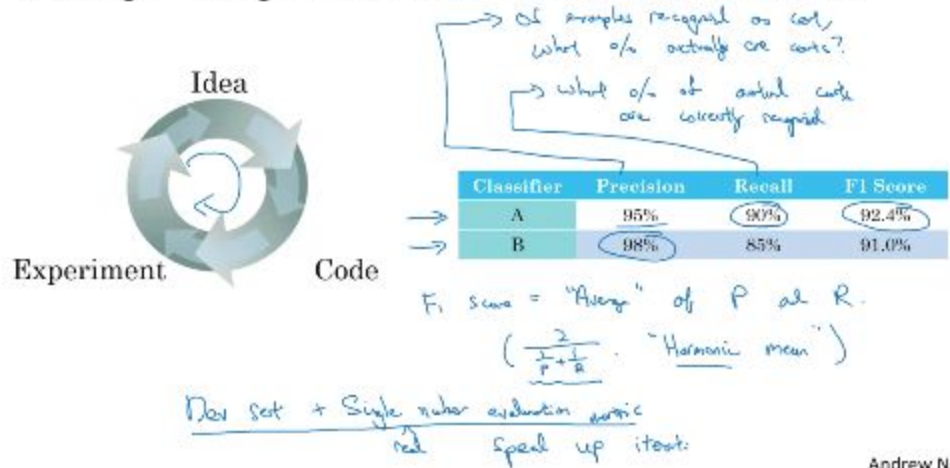


## Setting up the goal: Single number evaluation metric

- ML is an iterative process so we are trying different ideas, experimenting with them and then refining them. In some cases it's kind of uncertain what to do next. In the classifier example above, we don't know what is the path to follow if we are only evaluating precision and recall. Precision is defined as 'of all the examples recognized as cats, what

% of those examples are actually cats?' and recall is defined as 'what % of the actual cats were correctly recognized?'.

## Using a single number evaluation metric



A way to solve this problem is to use a single number evaluation metric. In this case we used the F1 score which can be informally defined as the average of P and R and whose formula is  $2/(1/P + 1/R)$ . This is also known as the harmonic mean of P and R. So what we have is that using a single number evaluation metric can speed up our iterations.

- Here's another example. In this example we have a classifier and its error performance across different geographies. Having different algorithms and many geo's makes it hard to decide which classifier is better. Adding the average gives us a way to have a single metric to evaluate each algorithm and select the one with the lowest error margin.

## Another example

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%

### Setting up the goal: Satisficing and optimizing metric

- We already looked at how we can have one metric to make a decision on which algorithm is better for the task. This time we are going to add a wrinkle: below is an example where we have accuracy (which we want to maximize) and running time (that we care about). What do we do here? In this case we pick one metric to maximize (the

accuracy) and another to satisfy (meaning that we are up to a certain level and then we don't care). We then say we want to maximize/optimize accuracy subject to a running time  $\leq 100$  ms. So we are saying accuracy is the *optimizing* metric and running time is the *satisficing* metric. In general we say that for N metrics we care about we pick 1 optimizing metric and N-1 satisficing metrics.

## Another cat classification example

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

$\text{Cost} = \text{accuracy} - 0.5 \times \text{Running Time}$   
 Maximize Accuracy  
 Subject to Running Time  $\leq 100$  ms.  
 N metrics: 1 optimizing  
 N-1 satisficing

Wakewords / Trigger words  
 Alexa, OK Google -  
 Hey Siri, rishabhdev  
 你好百度

accuracy.  
 #false positive  
 Maximize accuracy.  
 s.t.  $\leq 1$  false positive  
 every 24 hours.

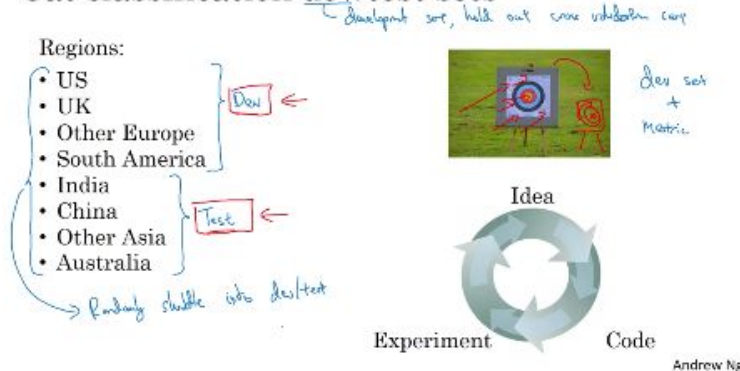
Andrew Ng

The example on the right about the trigger word for alexa-like devices maximizes accuracy of 'waking up' when hearing the trigger word and satisficing a false positive condition (1 false positive every 24 hours). Accuracy is then the optimizing metric and # of false positives is the satisficing metric.

## Train/Dev/Test set distributions

- The ML process calls for training different models in the training set, evaluating them and picking one using the dev set. Finally we take the selected one and evaluate it using the test set.

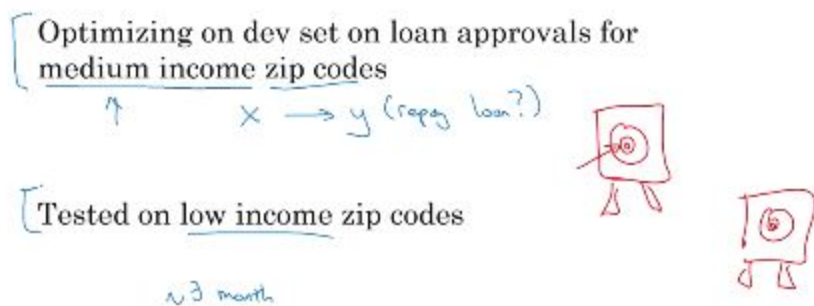
## Cat classification dev/test sets



In the example above, one approach could be to have a dev test with the first 4 geo's and the test set with the last 4 regions but that's a bad idea because the sets come from different distributions; they should be from the same distribution. Changing the distribution is like moving the target we are trying to hit. The recommendation is to randomly shuffle the data into the dev and test set.

- Another example is shown below. In this case, a team spent 3 months working on a dev test on loan approvals for medium income zip codes. It was trying to predict  $y$  (will the loan be repaid) if  $x$  (the income level of the zip code). Then it tried to test on low income zip codes and, predictably, the classifier didn't work well.

### True story (details changed)



Andrew Ng

- The guideline then is choose a dev set and test set that reflects data we expect to have in the future and we consider important to do well on. The dev set and the test set have to come from the same distribution.

### 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.

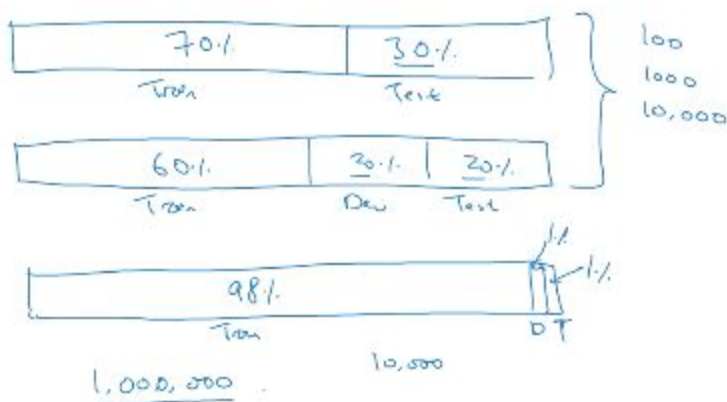


Andrew Ng

### Size of the dev and test sets

- The old way of splitting data (when we had 100, 1,000 or 10,000 examples) was to have a division of 70% train/30% test or 60% train/20% dev/20% test. In the deep learning era it is very common to have much bigger example sets. For cases like, say 1 M examples, that an appropriate distribution is 98% training/1% dev & 1% test.

## Old way of splitting data

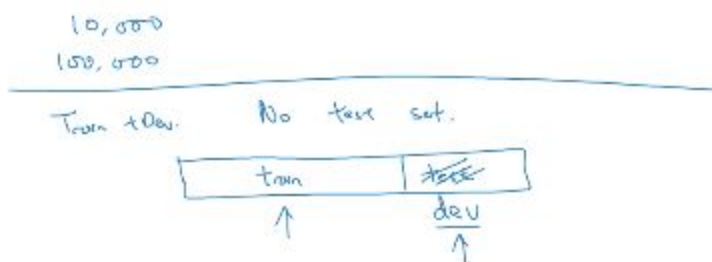


Andrew Ng

- The rule of thumb is to set the size of the test set to be big enough to give us high confidence in the overall performance of our system.

## Size of test set

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



## When to change dev/test sets & metrics

- When can we change our metric? Let's consider the example below. Let's assume we have 2 cat-classifying algorithms and choice A has a 3% error and choice B has a 5% error. However, choice A is showing users pornographic images which is not acceptable. Here we are better served by selecting a new metric to evaluate our 2 algorithms. We can use an equation like this one:  $error = 1/m_{dev} \sum_{i=1}^m I(y_{pred}^{(i)} \neq y^{(i)})$ . I will give us either 0 or 1. We can take into account the porn problem by adding a weight  $w$  and we rewrite the



equation as follows:  $error = 1/m_{dev} \sum_{i=1}^m w^{(i)} I(y_{pred}^{(i)} \neq y^{(i)})$ . Weight  $w=1$  if image is non pornographic and  $w=10$  if  $w$  is pornographic..

## Cat dataset examples

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

→ Metric: classification error

Algorithm A: 3% error → pornographic

✓ Algorithm B: 5% error

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

$I\{y_{pred}^{(i)} \neq y^{(i)}\}$  predicted value (0/1)

Andre

The takeaway is then that if our metric is not ranking the algorithms in the correct preference then it's time to define a new evaluation metric.

- We have 2 steps in such a case: first we define the metric to evaluate our algorithms and then in the 2nd step we figure out how to do well in that metric.

## Orthogonalization for cat pictures: anti-porn

→ 1. So far we've only discussed how to define a metric to evaluate classifiers. ← Place target 30

→ 2. Worry separately about how to do well on this metric. ← Am (shot at target)

$$J = \frac{1}{\sum w^{(i)}} \sum_{i=1}^m w^{(i)} \mathcal{L}(y^{(i)}, y_{pred}^{(i)})$$



- Another example is below. In this example we are training our algorithms in dev/test set that has very well defined pictures. However, the user images are not as good and are kind of blurry and our algorithm does not perform well there. This is an indication that we need to change our metric and/or our dev/test set.

## Another example

Algorithm A: 3% error

✓ Algorithm B: 5% error ←

→ Dev/test



→ User images



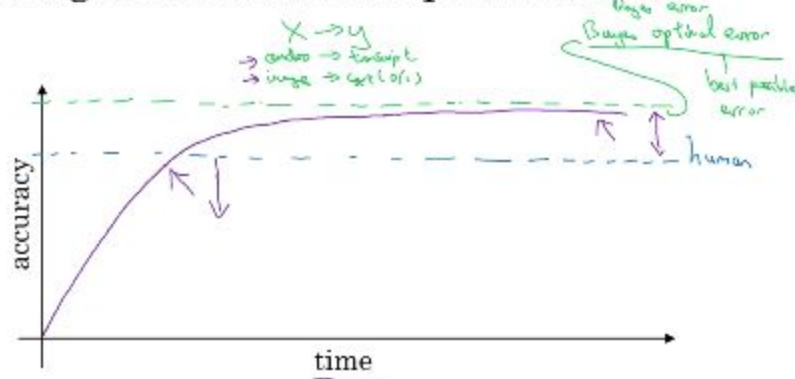
If doing well on your metric + dev/test set does not correspond to doing well on your application, change your metric and/or dev/test set.

Andrew Ng

## Comparing to human-level performance: Why human-level performance?

- It has become common to compare ML algorithm performance to human level performance because now algorithm performance is comparable to the humans. What happens is that as you work on a problem accuracy increases and passess human performance but cannot get to the Bayes optimal error (best possible error for an  $X \rightarrow Y$  function). One possible reason for that is that there might not be a lot of difference between human-level performance and the best possible error.

## Comparing to human-level performance



- We have some tools to help us when ML performance is less than human performance. These tools are:
  - Get labeled data from humans
  - Gain insights from manual error analysis
  - Better analysis of bias/variance



## Why compare to human-level performance

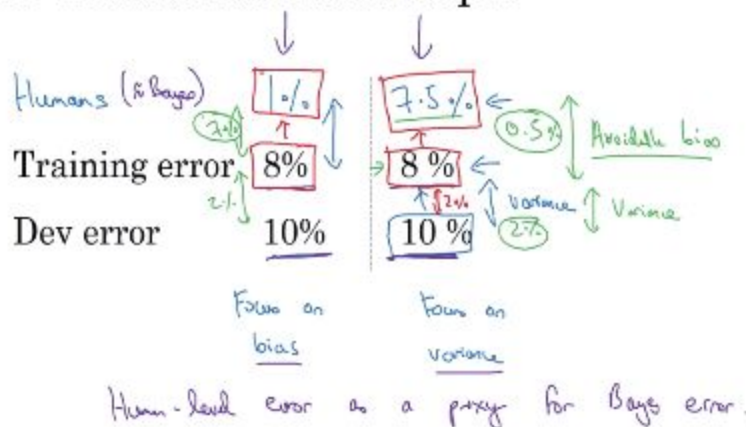
Humans are quite good at a lot of tasks. So long as ML is worse than humans, you can:

- 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.

### Comparing to human-level performance: Avoidable Bias

- Human-level performance is useful in helping us decide how to improve our algorithm. In the example below first we have human-level performance of 1% and a training error of 8%. There is a big difference between the 2. We can use human-level performance as a proxy for Bayes error so it is desirable to reduce our training error to make it closer to 1%; we then focus on techniques to reduce bias. On the 2nd example human-level performance is very close to the training error so we instead will concentrate in reducing the difference between the training error and the dev error; we then focus on techniques to reduce variance.

### Cat classification example



### Understanding human-level performance

- Let's see how we can define human-level error more precisely. Below are different human level errors, which one is the one we should choose as 'human-level' error? The last one because that's the best one a human system can do and, therefore, the Bayes error has to be  $< 0.5\%$ . Maybe for another purpose we can use 1% as the human-level

error. The takeaway is then we should be clear about what is our purpose in defining human-level 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 ←

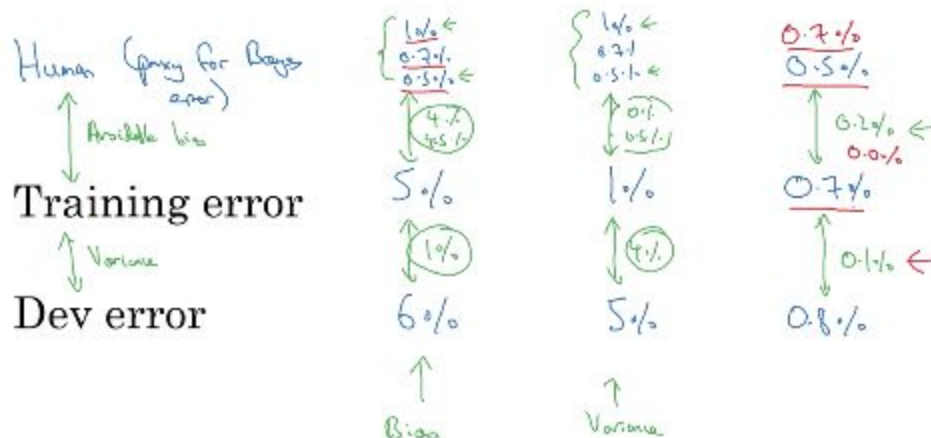
Bayes error  $\leq$  0.5%

What is "human-level" error?

Andrew Ng

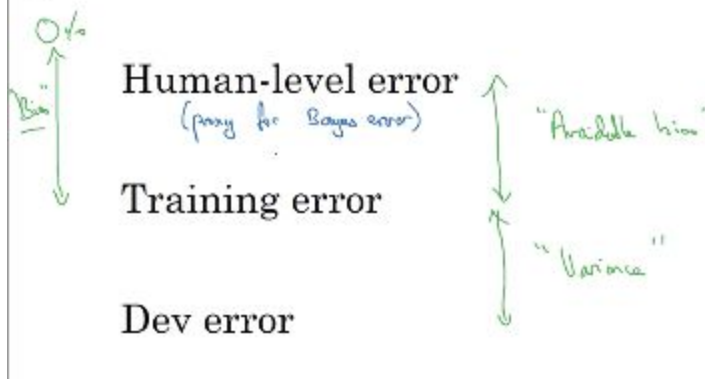
- Below we have a couple of progressive examples where our model is becoming better and in both cases it is clear what we have to concentrate on (either bias or variance). However, on the 3rd example we are approaching human-level performance it becomes hard to tease out the effects of variance or bias. In this 3rd example our algorithm is already doing very well and the difference between all 3 metrics is close and the decision on what to do would depend on the human level performance metric we choose. If we choose 0.7% then we concentrate on variance. If we choose 0.5% we concentrate on bias.

## Error analysis example



- Summarizing, having an estimate of human level performance gives us an estimate of Bayes error, which allows us to decide on whether we need to reduce bias or variance.

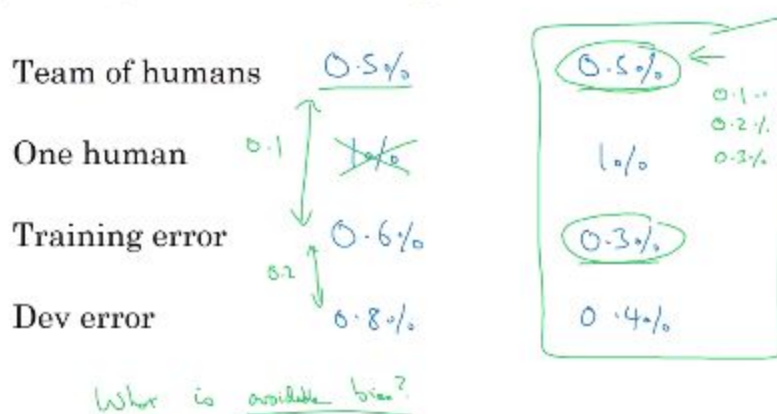
## Summary of bias/variance with human-level performance



### Surpassing human-level performance

- What do we do when we have an algorithm that surpasses human-level performance? Consider the 2 examples below. The first example is relatively easy as we can use the team of humans performance as our Bayes error so we have 0.1% avoidable bias and 0.2% variance. However, the 2nd example is harder: our training error is already doing better than what humans can do, what do we do next? In this example we don't have enough info on what to do next and our tools are not helping us.

### Surpassing human-level performance



Andrew Ng

- Some areas where ML models have exceeded human-level performance are: online advertising, product recommendations, logistics, loan approvals, speech recognition, some image recognition tasks and some narrow medical tasks.

## Problems where ML significantly surpasses human-level performance

- • Online advertising
- • Product recommendations
- • Logistics (predicting transit time)
- • Loan approvals

Structured data  
Not human perception  
Lots of data


{  
- Speech recognition  
- Some image recognition  
- Medical  
- BCG, Skype, ...

Andrew Ng


### Improving your model performance

- We are going to summarize everything we learned into a set of guidelines. We start with two assumptions: we can fit the training set pretty well (which means we have low avoidable bias) and the training set generalizes well to the dev/test set. Each assumption has a set of 'knobs' we can use.

## The two fundamental assumptions of supervised learning

1. You can fit the training set pretty well. 

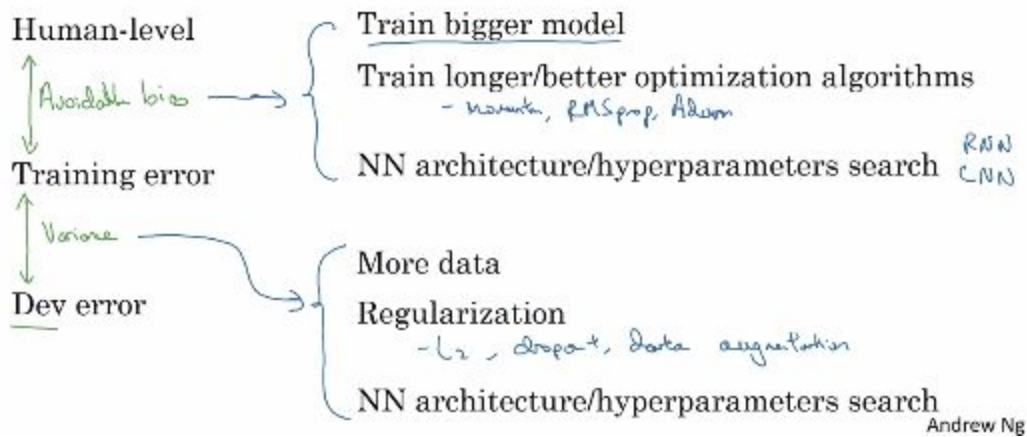
~ Avoidable bias

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

~ Variance

- What are our tools for each situation? To reduce avoidable bias we can train a bigger model, train longer/better optimization algorithms (Adam, RMSProp, etc.) or use a different NN architecture or hyperparameters search. To reduce variance we can use regularization, we can get more data or we can use a different NN architecture or hyperparameters search.

## Reducing (avoidable) bias and variance



Bias and variance are kind of easy concepts to teach but mastering them is an art so we need lots of practice.