Why ML Strategy?

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

- Collect more diverse data.
- Train more epochs.
- Train with different optimizers.
- Try bigger/smaller networks.
- Try different regularizations.
- Change the architecture of neural networks.
- Try smaller sets of features.
- Getting additional features.
- Adding polynomial features (x_1^2, x_2^2, x_1x_2, etc)

- Get more training data. fix high variance
- Try smaller sets of features. ~~fix high bias.~~ fix high variance
- Get additional features. fix high bias.
- Add polynomial features. ~~fix high variance~~ fix high bias.
- Decrease λ. fix high bias.
- Increase λ. fix high variance

Is using neural networks equivalent to adding polynomial features automatically? Or in other words, polynomial features can be found automatically by neural networks?

The model makes unacceptably large errors in predictions. What should I do?

I need to evaluate the model. The following explains how to evaluate linear regression model and logistic regression model. First, we need to split data into 70% training set and 30% testing set. m_{train}: No. training examples. m_{test}: No. training examples.

Train/test procedure for linear regression:

Fit w, b by minimizing $J(w,b) = \frac{1}{2m_{train}} \sum_{i=1}^{m_{train}} \left(f(x^{(i)}) - y^{(i)}\right)^2 + \boxed{\frac{\lambda}{2m_{train}} \sum_{j=1}^{n} w_j^2}$

Compute train error: $J_{train}(w,b) = \frac{1}{2m_{train}} \sum_{i=1}^{m_{train}} \left(f(x^{(i)}) - y^{(i)}\right)^2$

Compute test error: $J_{test}(w,b) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} \left(f(x^{(i)}) - y^{(i)}\right)^2$

Notice that the regularization term is only used during training. In evaluation, J_{train} and J_{test} doesn't need it.

Train/test procedure for logistic regression:

Fit w, b by minimizing $J(w,b) = \frac{-1}{m_{train}} \sum_{i=1}^{m_{train}} \left[y^{(i)} \log(f(x^{(i)}) + (1-y^{(i)}) \log(1-f(x^{(i)}))\right] + \boxed{\frac{\lambda}{2m_{train}} \sum_{j=1}^{n} w_j^2}$

Compute train error: $J_{train}(w,b) = \frac{-1}{m_{train}} \sum_{i=1}^{m_{train}} \left[y^{(i)} \log(f(x^{(i)}) + (1-y^{(i)}) \log(1-f(x^{(i)}))\right]$

Compute test error: $J_{test}(w,b) = \frac{-1}{m_{test}} \sum_{i=1}^{m_{test}} \left[y^{(i)} \log(f(x^{(i)}) + (1-y^{(i)}) \log(1-f(x^{(i)}))\right]$

In logistic regression, J_{train} and J_{test} can be used to evaluate a model. But, a better choice would be Confusion matrix.

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電路板，必須把一塊複雜的電路拆分成子模組，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 design, these are the 4 assumptions that needs to be true and orthogonal.

1. Fit training set well in cost function (≈ 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.

Using train/valid/test set in a machine learning procedure is better than using only train/test set is because we want to isolate the development process and the final testing process. So, we use training set to train models while using validation set to select hyperparameters. After the development process, we then use test set to get the more objective generalization error. This is more objective because test set has never been exposed to the development process.

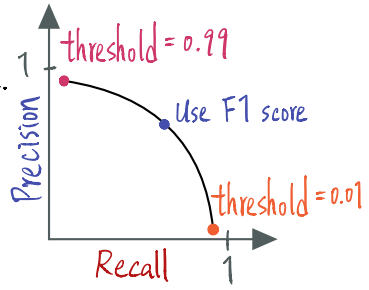
Trading off between Precision and Recall

Given a trained logistic regression model: $0 < f_{w,b}(\vec{x}) < 1$ to predict a rare disease.

if $f(\vec{x}) \geq$ threshold, $\hat{y} = 1$. (disease)

if $f(\vec{x}) <$ threshold, $\hat{y} = 0$. (no disease)

Normally, threshold = 0.5



1. Suppose we want to predict $y=1$ only if very confident. Thus, increasing threshold = 0.7

   It results to higher precision, lower recall.

2. Suppose we want to avoid missing too many cases of a rare disease. Thus, decreasing threshold = 0.3

   It results to lower precision, higher recall.

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	Actual class y = 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?

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

Recall

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

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

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~~ and 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.

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

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 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:

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

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

Cat Classification Dev / Test Sets → aka, hold-out cross validation set

The cat classification APP is applied to the following regions:

Regions:

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

Dev

Test

→ 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%

100

Train	Dev	Test
60%	20%	20%

1000

Train	D	T
98%	1	1

10^5

10^6

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

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

First Cat dataset example:

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

Algorithm A: 3% error + Pornographic
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:

$$\frac{1}{\sum_i w^{(i)}} \sum_{i=1}^{m_{dev}} w^{(i)} \mathcal{L}\{y_{pred}^{(i)} \neq y^{(i)}\},$$

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

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)

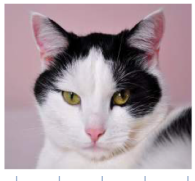
Second Cat dataset example:

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

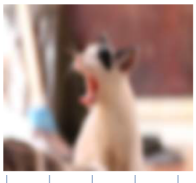
Algorithm A: 3% error
Algorithm B: 5% error

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.

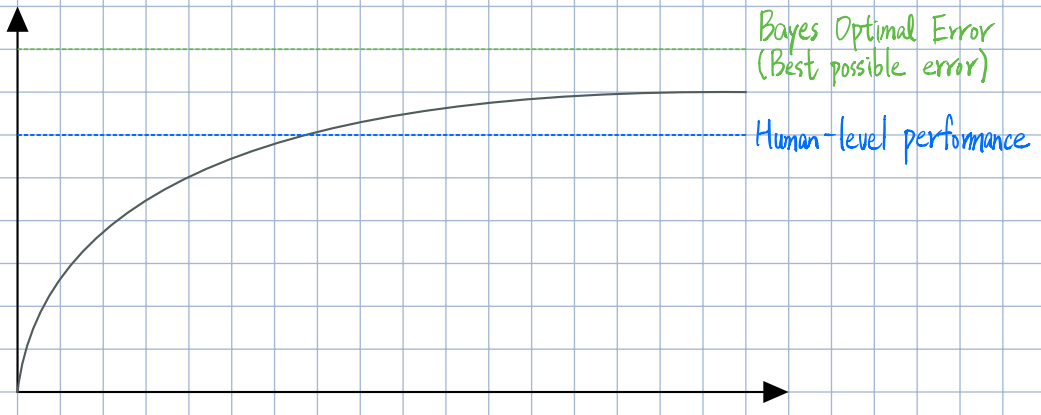
In this case, change the metric and/or dev/test set, such as, including low-resolution imagery in the Dev/Test set.

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 (≈ Bayes Error):	1%	7.5%
Training Error:	8%	8%
Dev Error:	10%	10%
	Bias	Variance
	Focus on reducing Bias	Focus on reducing Variance

Avoidable Bias = 0.5%

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

What is "human-level" error?

Theoretically the bayes error could be ≤ 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.

Error Analysis Example:

Human error (proxy for bayes error):	1%	1%	1%
	0.7%	0.7%	0.7%
	0.5%	0.5%	0.5% (Use this one as Bayes Error)
Avoidable Bias			0.2
Training error :	5%	1%	0.7%
Variance			0.1
Dev error :	6%	5%	0.8%
	Focus on Bias Reduction	Focus on Variance Reduction	Hard to further improve the model. But in this example, probably focusing on Bias reduction is better.

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.

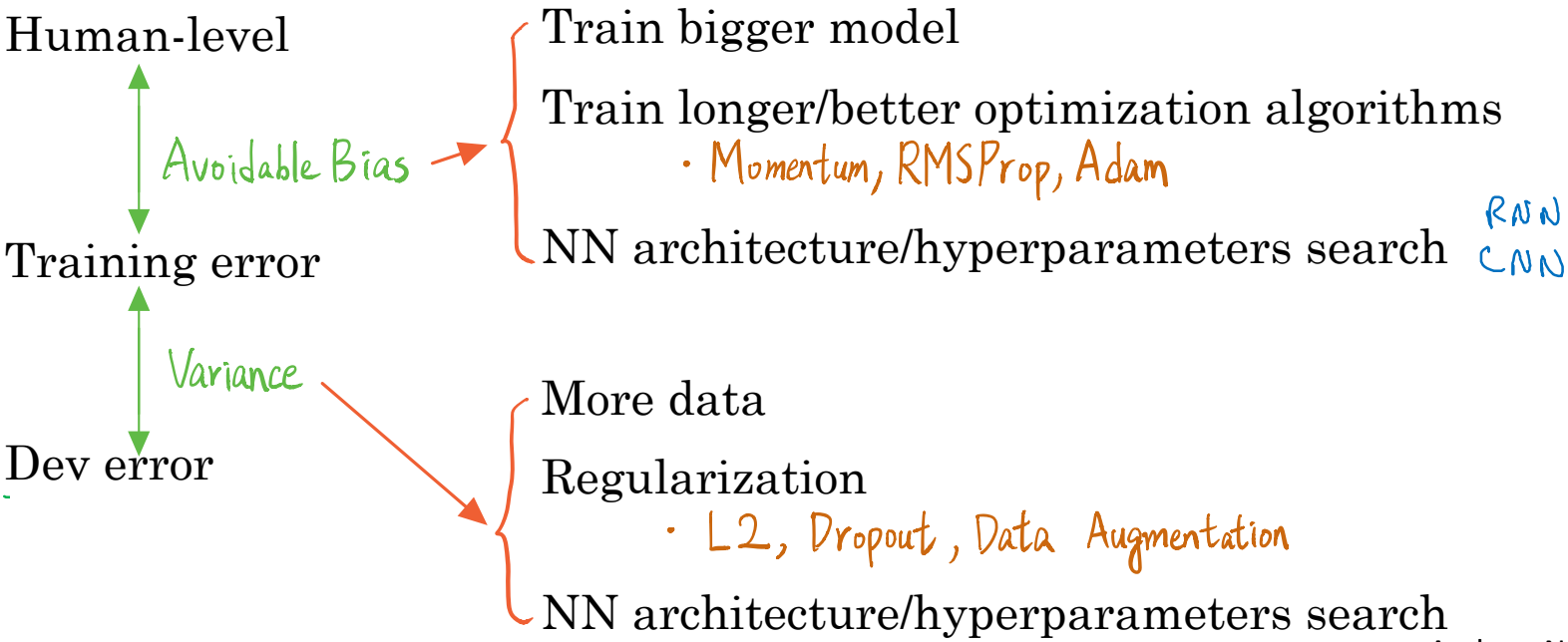
- Computer Vision
- Speech Recognition
- Natural Language Problem

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



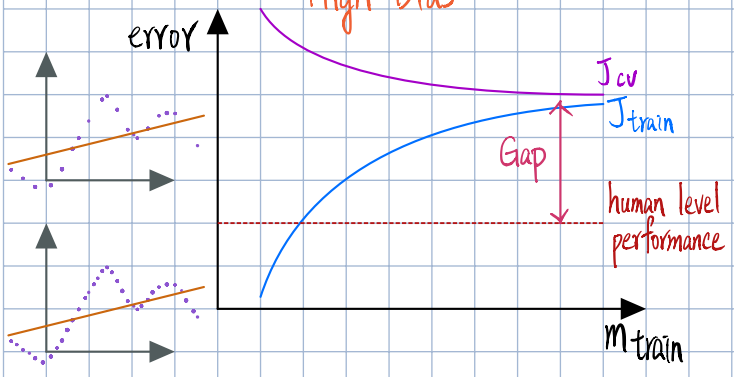
Understanding Learning Curve

J_{train}: training error , J_{cv}: cross validation error. m_{train}: No. of training samples

A Good Model

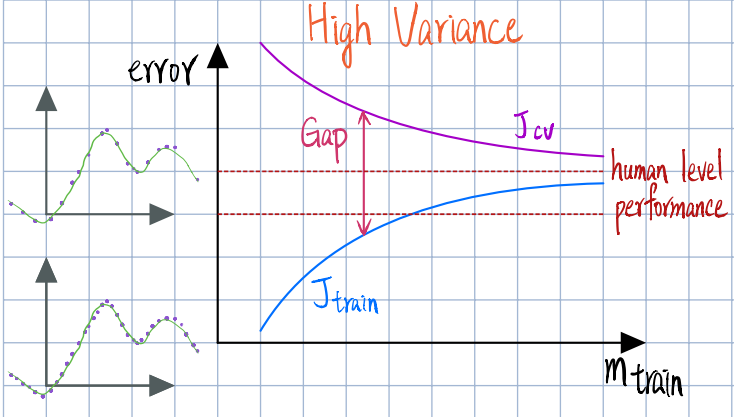
This graph shows the development procedure is going well. First, J_{train} increases when m_{train} increases, which is normal because it's becoming more difficult for a model to fit more data. Second, J_{cv} is decreasing and approaching J_{train}, which means more training data is help the model generalize well. Third, J_{cv} is always higher than J_{train} because the model is fitting the traing set, not validation set.

High Bias



First, J_{train} starts to flatten out when given more training data. The reason is that the model is too simple to grab features out from training data. Thus, throwing more data to it is not helpful. Second, due to the same reason as the first point, J_{cv} is also flattening out. Third, there's a huge gap between the baseline and J_{train}, which indicates this model has high bias. This could be solved by using a more complex model. However, it's important to note that using more data wouldn't help. Stop wasting time on collecting more data.

High Variance



First, J_{train} is unrealistic low due to overfitting of training data when training data is few. Second, when training data is few, there is a huge gap between J_{cv} and J_{train} due to overfitting. Third, when given more data, it's possible that J_{train} is higher or lower than the human level performance. Fourth, when giving more data, J_{train} will continue to go up. But, J_{cv} is hopefully going down. I.e., getting more training data is "likely" to help.

Conclusion: if a learning algorithm has high bias, getting more training data will not help.

if a learning algorithm suffers from high variance, getting more training data is likely to help.