Course #3: Structuring Machine Learning Project

Week #1: ML Strategy (1)

Why ML Strategy:-

Let's start by a motivating example:-

→ Let's say you are working on a Cat classifier. After training sometime you see that your model's accuracy is 90%. But this is not good enough for your application.

→ By then you must have a lot of Ideas about improving your accuray. e.g:

(i) Colleding more data.
(ii) Collect more diverse training set.
(iii) Train algorithm longer with Gradient Descent.
(iv) Try different otimization algorithm like **Adam**.
(v) Try bigger or smaller network.
(vi) Try Dropout
(vii) Add L2- regularization.
(viii) Try different Network architecture
ix) Different activation function et.c.

→ There will be differed Ideas in your mind and if you choose poorly, it is quite possible that, you spend six month charging in one direction only to find out that you just did it wrong.

Would it be nice to have no. of strategies that help to choose only useful parameters and discard others.

Orthogonalization :-

→ Having a clear eye about "what to tune" in order to achieve certain amount of accuracy in one thing is called "Orthogonalization".

e.g: take an example of tuning the TV.

The old school TV used to have lot of knobs on them in order to tune lots of thing.

e.g
- (a) ← 1 knob to tune horizontal alignment of picture
- (b) ← 1 knob to tune Vertical " " "
- (c) ← 1 knob to tune trapezodal alignment.
- (d) ← 1 knob to tune circular alignment e.t.c.

Now Imagine if you had a knob which controls the positioning of image like this.

$$0.1xa + 0.3xb - 1.7xc + 0.8xd + \cdots$$

Now if you had a knob like this, it would be impossible to position the image

→ Now orthogonalization refers to: TV designers have designed a system where every function is controlled by one knobs and this makes the tuning easier.

How does this relate to machine learning?

→ In supervised learning you need to tune the knobs of your system to make sure 4 things hold true.

① You have to make sure that, you are atleast doing well on training set. For some applications this means that, it should be comparable to human level performance.

② After doing well on the training set, you hope that, it would also do well on the test.

③ As a result of that, you hope your system will also be applicable to real world examples.

⇒ Now analogous to TV tuning example, you want here knobs or set of knobs to make sure that, your algorithm works well on your training set.

→ So what knobs would do is that, you might train a bigger network or you might switch to better optimization algorithm and so ons.

→ Now if your algorithm is not doing well on the dev set, you have another set of knobs that adds regularization to your algorithm. OR getting a bigger set of data can be controlled by another knob too.

→ First 2 adjustments can be thought of as adjusting width of picture and length respectively.

Now, what if your algorithm doesn't meet the 3rd criteria? what if you do well on the dev set but not on the test set?

→ Then you probably want to tune you "bigger dev set" knob i.e. if its is doing well on the dev set but not on the test set then it means that you have overtuned your dev set and you might wanna change that.

→ Now after all this if it is not delivering the required amount of accuracy on the real world application then either you might want to tune your dev set or change your cost function.

Upon not doing well on the real world application, it means that, either your dev set / test set distribution isn't set correctly or your cost function isn't measuring the right thing.

→ Upon fitting a neural network, it is not a good idea to use early stopping, because it affects two things at a time.

↳ It affects the training accuracy

↳ It also often simultaneously done to improve your dev set performance.

So this knob is not well orthogonalized.

Let's dig deeper into the process of identification of recognizing the bottlenecks of the application and and identify some knobs to in order to rectify this issue.

→ Setting up your goal:-

Single number evaluation metric:-

Whenever you are trying to fit a neural network, it would be nice to have single real number metric ~~with~~ which would tell you, whether your current idea is better or worse than the previous one.

Let's look at an example:-

As we have said repeatedly that, deep learning is an empirical process. So having a single number metric for evaluation can come in handy like this:-

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

Precision → of examples recognized as cats what % actually are cats.

Recall → what % of actual cats are correctly recognized.

→ There is often a trade off between Precision and Recall, and you care about both. You want that, when a classifier says something is a cat, there is a high chance that, it is a cat.

But of all the images that are cats, you want to pull a large fraction of them as cats.

But the problem is that, if your model A is doing well on Precision but model B is doing well on Recall then you are not sure which classifier to go with.

→ In practice you may be not training one classifier may you are training dozen. So instead of using above mentioned 2 metrics, we use a third one which combines them 2. In machine learning literature this is called

"F1 Score" → { Informally it can be thought of as average of Precision and recall

Formally its definition is:-

$$\text{F1 Score} = \frac{2}{\frac{1}{P} + \frac{1}{R}}$$ → Harmonic Mean.

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

So, well defined dev set + single number evaluation metric is a recipe to speed up iterations.

Let's get another example:

e.g: You have trained a classifier for cats. Now you are determined to test its accuracy in different parts of the world's cats. e.g say you test it on cats submitted by Pakistani users, USA users, Indian users, Chinese users e.t.c.

Now if you have a lot of models that you have trained, it would be difficult to deal with all the data. what you should do is take the average of the error for every model and Pick on that basis.

Satisficing and optimizing Metrics:-

It's not always easy to combine all the things you care about in one single row number evaluation metric.

e.g when training a cat classifier you take into consideration the accuracy and also the running time of your model.

Classifier	Accuracy	Running time
A	90%	80 ms
B	92%	95 ms
C	95%	1500 ms

One thing you could do is combine Accuracy and running time into an overall metric ⇒ "Cost".

e.g | Cost = accuracy − 0.5* running time |

May be it seems a bit artificial to have a formula like this. Here is what you can do instead.

→ You might want to choose a classifier that is maximizing the accuracy and satisfy that running time ≤ 100 ms.

→ So in this case accuracy is your optimizing parameter, because you want it optimized and Running time is Satisficing parameter.

→ Now this criterion sounds like a bit legible. So in general if you have N metrics, you might want to pick 1 optimizing parameter and (N-1) satisficing parameters.

One other example:- You are trying to build a trigger word system. Like you make up google assistant by saying "OK Google".

Here your metrics could be like the accuracy which you want to maximize and other one can be the the false positive alarm like how often does it make up even when you haven't said anything.

So here you can set accuracy as your optimizing parameter and no. of false positives ≤ 1 for every 24 hours.

You would want these parameters satisfied on train, test and dev set, let's see how to setup those.

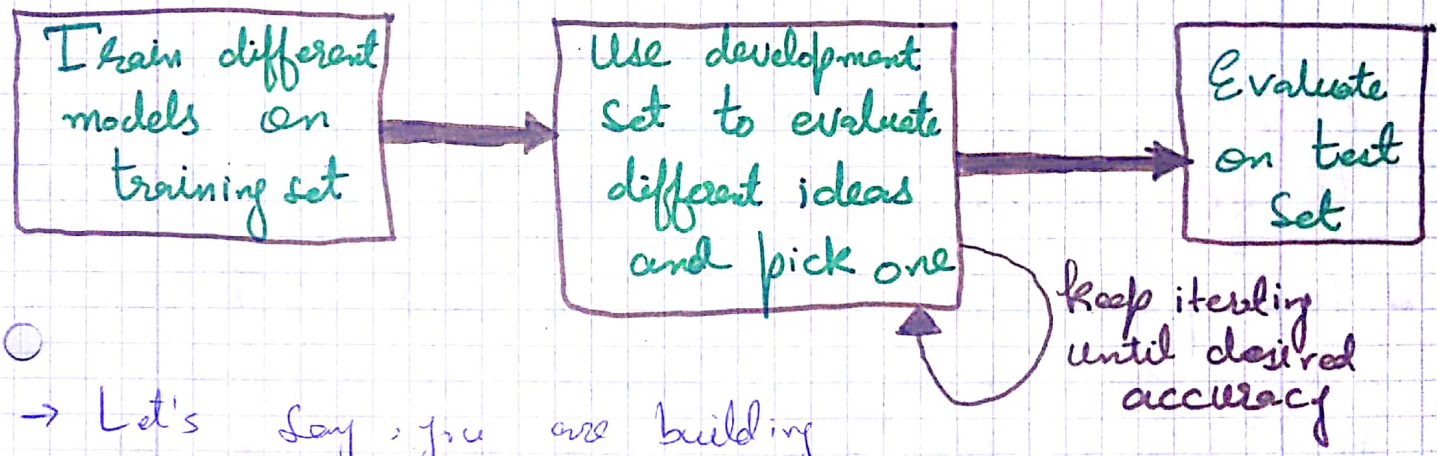
Train/dev/test Distributions:-

- The way you distribute your Train/test/dev set can have a lot of impact on the speed of success towards a trained model.

⇒ Let's see how you can set up your dataset to maximize your team's efficiency.

→ Let's focus on how you develop your development and test sets
  ↳ also called hold out Cross-Validation set.

Usually this is the workflow in machine learning :-

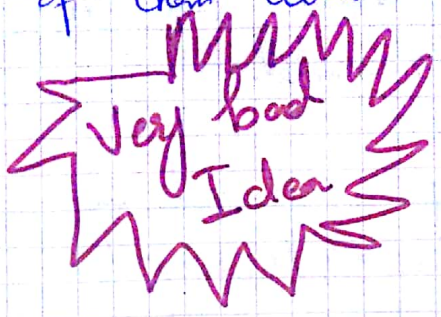


→ Let's say, you are building a classifier and you are operating in the following regions:-

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

Now how do you set up your dev set and test set?

→ One way is that, you say any randomly four countries' data will come into development set and rest of them will come into test set.



→ because your dev and test set come from different distributions.

→ Here is what I mean?

→ Setting up your dev set + your row number evaluation metric, that's like setting your target and telling your team where you think is the bull's eye you want to aim at.

Once you have done that, then you can iterate through different ideas and try to hit that "target".

→ Now problem here is that, you spend month on training your model in order to achieve the goal ("hitting bullseye) only to find out that your test set distribution is different and you fail in getting intended accuracy on test set.

→ What you can do is that, you shuffle the data from all different regions and pick randomly for dev and test set.

This approach will make your distribution same.

Choose a dev set and test set to reflect data you expect to get in future and consider important to do well on.

Should come from same distribution.

Size of dev and test sets:-

Having established the kind of data that should be in dev and test set, now what about their size?

Old way of splitting data:-

70% Test	30% Test

OR

60% Test	20% dev	20% test

In old days this was pretty reasonable, specially when data set sizes were smaller.

→ Now that, in the era of bigdata, where you have millions or billions of data, it would be quite reasonable to classify like this:-

98% Training set	1%	1%

1% → Dev Set; 1% → Training Set

→ because for 1% of one million will be 10,000 and this seems pretty reasonable.

Size of test set:-

Remember the purpose of test set:-
after you finish training your system, and dev. it, the test set evaluates, how good your system is.

So the guideline here is that,

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

→ In some cases it might be okay to not have high confidence in test set. Sometimes it might be okay to just use dev set (if you have it in very large amount). But it is not recommended.

When to change dev/test sets and metrics:-

We have seen that, classifying the data into dev and test sets and identifying a metric is like setting a goal for your team. But some times you realize that, you are following a wrong target and need to change your goal in between.

Let's see an example for that:-

Let's say you are classifying cats and following is the data you recieve.

Metric : Classification Error

Algorithm A : 3 % Error → Seems like it is doing better.

Algorithm B : 5 % Error

→ Let's say you try out different algorithms and access results and you see that **Algorithm A** is for some reason letting through pornographic images.

→ which means although "algorithm A" will classify lots of cats correctly but it will also show pornographic images which is totally unacceptable.

→ But with **Algorithm B** although it is misclassifying some images but it is not letting through pornographic images. So, for your company this is better.

Now, what's happening here is that, algorithm A is preffered by your performance metric and dev set, but You and users prefer algorithm B.

So, when your algorithm is no longer ranked rightly by your performance metric (i.e. mispredicting which algorithm is better in this case algorithm A), then it is a sign that, You should change your evaluation metric or perhaps your dev set/test set.

So, in this case; the misclassification error metric that you are using can be written as follows:-

$$\text{Error} = \frac{1}{m_{dev}} \sum_{i=1}^{M_{dev}} w^{(i)} \left[\left\{ y^{(i)}_{predicted} \neq y^{(i)} \right\} \right]$$

(Either 0 or 1)

gives number of misclassified examples.

→ one way to remove this is adding another factor $w^{(i)}$ ⇒ weight.

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

This way you are giving a much higher weight to misclassified images, so the error term goes much more if the algorithm classify a type of image which you don't want to see "Correctly".

→ If you want this normalization constant that in error formula: $1/m_{dev}$ becomes $1/\sum_i w^{(i)}$ So, this error will still be between zero and 1.

→ For all this you would have to go through your dev and test sets and label images which are pornographic.

Main takeaway:- if your performance metric is not giving you the proper results i.e. not telling you which model would be actually good then change the performance/Evaluation metric or dev/test set.

Orthogonalization for Cat pictures: anti-Porn

1- So, for we've only discussed how to define a metric to evaluate classifiers.

2- Worry seperately about how to do well on this metric i.e. how to make it optimized so that it does its job well.

So machine learning task can be thought to be solved in two distinct steps:-

1- Tune one knob to define the target. i.e. define metric

2- Tune second knob to make sure that you do well on that target. → hit bullseye. i.e. that metric is telling you what you actually want to know.

→ In terms of shoting a target, may be your algorithm is optimizing a cost function that looks like this:-

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

one thing you can do it change it and incorporate these weights or may be you end up changing that normalization constant as well.

→ **Another example** would be that if you are developing your model on images which are very nice and really high resolution than when you will evaluate it on images which are blurry it won't do that well. So, that will be the time to make some changes in dev set.

★ If doing well on your metric + dev/test set doesn't correspond to doing well on your application, change your metric and/or dev/test set.

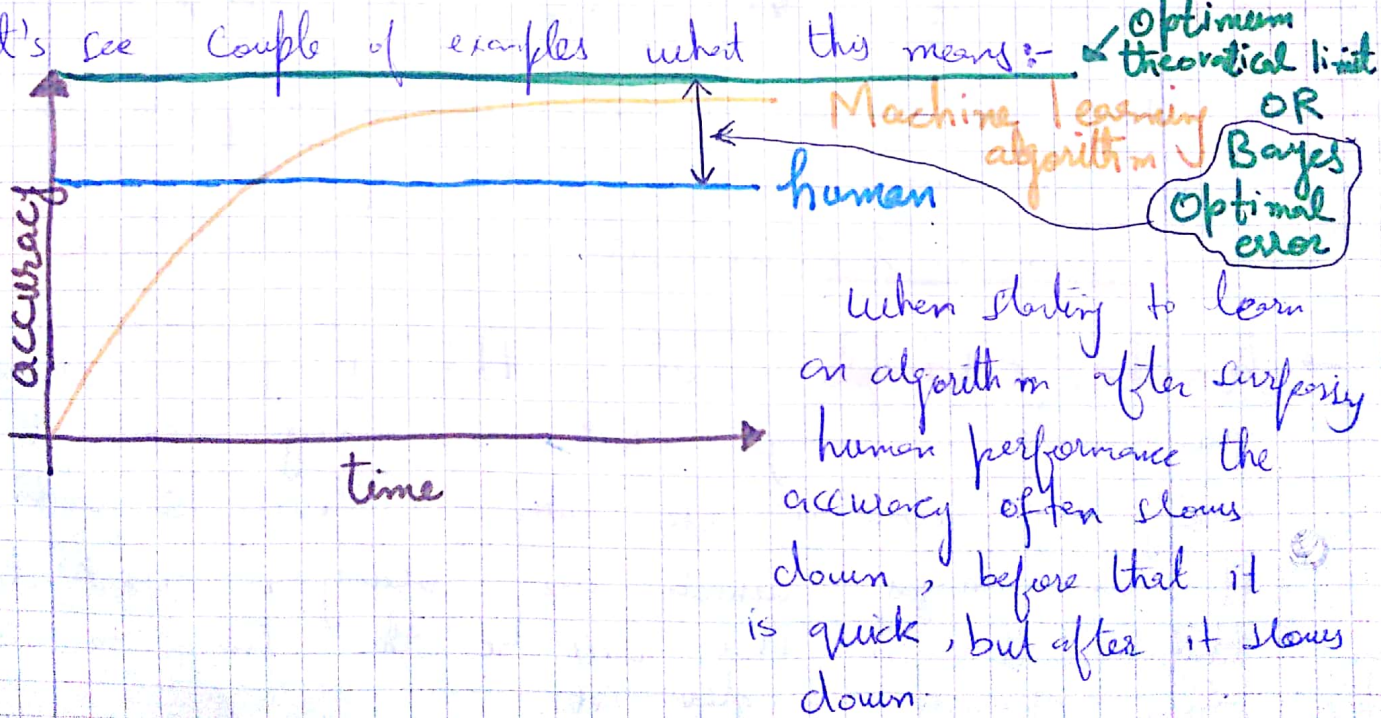
Comparing to human level Performance:-

Why human level Performance:-

There are mainly 2 reasons:-

- With advances in deep learning, suddenly machine learning's algorithms in a lot of application areas have become much more feasible and competitive with human level performance.
- When you are trying to build something via machine learning, the workflow is much more efficient.

Let's see couple of examples what this means:-



When starting to learn an algorithm after surpassing human performance the accuracy often slows down, before that it is quick, but after it slows down.

what happen after is that, performance approaches but never surpasses that theoretical limit and that limit is called **Bayes optimal error**.

Think of it as best possible error, that there is no way for any function mapping from $X \to Y$ to surpass a certain level of accuracy

e.g: when trying to train a speech recognition model, if the clip is so much noisy that there is impossible to tell if the letter is which is perfect is 100, this is in that case when you don't know what is actually there in the transcript.

→ Why performance slows down after surpassing human level performance?

One reason might be :-

- Human level performance for many tasks is not that far from Bayes' optimal error

Second reason might be :-

- As long as your performance is worse than human level performance, then there are actually certain tools you could use to improve performance that are harder to use once you've surpassed human level performance.

Here is what it's mean.

→ The tasks humans are quite good at, so long as ML is worse than humans, you can:

- Get labelled data from humans.
- Gain insight from "manual error analysis".
  i.e: why a person got this right but a machine didn't.
- Get better analysis of bias/variance.

Eeven though you know what is bias/variance, knowing how well humans can do on a task can help you understand better how much you should try to reduce bias and variance.

Let's explain this by an example.

Avoidable bias:-

You want your algorithm to do well on a training set but not "too well". Knowing human level performance can help you, how well you want to do on training set.

Cat classification example:-

Suppose:-

Humans' error : 1%

Training error : 8%

Dev error : 10%

In this example you would need to improve your algorithm performance on Training set. The fact that there is huge difference b/w human performance and training error shows that, your algorithm isn't fitting the Training set well.

In this case focus on improving bias.

→ Now let's say the human's performance is ~~15%~~ 7.5% but the human level error is 7.5% on a different application, but the training and dev error are still the same.

Now in this scenario, may be your algorithm is not doing that much worse. May be you'll focus on reducing the difference b/w training and dev error i.e. focus on reducing variance.

Now in these kinds of scenerios what you do is use "Human level error" as a proxy for Bayes' error (Human ≈ Bayes)

Let's call the difference between Training error and Bayes' error "Avoidable bias"

So, what it means is that, keep improving the model until you do better on Training set and training error (→) approaches to Bayes' error. You can't actually do better than Bayes' error because that's just overfitting.

Difference between training and development error is still a variance problem measure of your algorithm.

The term "Avoidable Bias" refers to that, you can't actually go below a certain level of error measure. Actually you don't actually want to get below that error (Bayes' error).

Understanding human level Performance:-

Human-level error as a proxy for Bayes' error:-

↳ Means what is the best possible error that a model even now or in future can ever achieve.

← Bearing this in mind, let's look at the image classification example

→ Let's say you want to look at radiology Image and make a diagnosis Classification decision.

Suppose:-

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

(b) Typical Doctor ..... 1% error

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

(d) Team of Experienced doctors ..... 0.5% error

Based on this, What is "human level" Error?

So, team of doctors achieve this kind of accuracy (0.5%) and keeping this data in mind, Bayes' error Cannot be ~~less~~ better then 0.5%.

Bayes' Error ≤ 0.5%

- **Main Takeaway:-** It is that to define your purpose and if it is to show that you can surpass a single human and therefore argue for deploying your system in some context then may be **(b)** is the proper definition for human level error. But if you go proxy for Bayes' error then **(d)** is the proper definition.

→ To see why this matters let's look at the error analysis example

Error analysis Example:-

Let's say for medical imaging diagnosis example you have the following data:-

Training Error :- 5 %

Development Error:- 6 %

→ Variance measure

Human level performance :- 1% / 0.7% / 0.5% } Depending upon the application

(Proxy for Bayes' Error)

Avoidable Bias ⇒ whatever human level error you use it will be bigger than variance measure error. So, we will focus on bias reduction technique e.g. train a bigger network

Now Consider another case.

Training Error:- 1 %

Development Error:- 5 %

Now here avoidable bias will be 0 – 0.5%, whereas for variance measure error is 4%.

So second case suggest you should focus on variance reduction techniques such as Regularization OR getting a bigger training set.

→ **But,** where it really matters is that when you have following data.

Training Error :- 0.7 %

Development Error :- 0.8 %

} In this case it really matters that, you use your estimate of Bayes' error to 0.5 %.

Here your avoidable bias has 0.2 % error which is twice as big as variance measure (0.1 %). In this case may both bias and variance are the problems but avoidable bias is much bigger problem.

→ had you used 0.7 % as a human level performance then avoidable bias would've been zero and in that case you [redacted] might've missed. In this case you actually should try to do better on training set.

→ So these example should give us an idea that, as we approach human level it becomes very hard to tease out bias and variance effects. In this case progress on your machine learning algorithms actually becomes much more difficult as you approach human level accuracy.

→ In earlier courses we were measuring training error and were seeing how much it is greater than zero, and just using that to try to understand our "bias". This works fine when your Bayes' error is zero such as recognizing cats. As humans are near perfect for that so, bayes' error is also near perfect for that. But for problems when data is noisy and it is even impossible for humans to do well on them, then your bayes' error will not be zero. In these kind of situations having a better estimate of bayes' error can help you better estimate avoidable bias and variance.

Surpassing human level Performance :- (Human error as proxy for Bayes' Error)

Let's see few things trying to accomplish human level performance.

→ We have discussed, how machine learning progress gets harder as you approach or even surpass human-level performance.

Let's talk over one more example why that's the case :-

You have the following data :-

Team of humans : ..... 0.5 %

One human : ..... 1 %

Training Error : ..... 0.6 %

Dev Error : ..... 0.8 %

Now, in this case, what is the avoidable bias?

In this example it is easier to answer. You are not gonna use "one human" parameter. So, your avoidable bias is at least 0.1% and your variance is 0.2%.

So, may be there's is more to do to reduce your variance than your avoidable bias.

Now let's take a harder example:-

Team of humans: ---- 0.5%

One human: ---------- 1%

Training Error: ------- 0.3%

Dev Error: ---------- 0.4%

Now, what is the avoidable bias? Now, it is much more harder to understand that.

→ It's the fact that your training error is 0.3%, does this mean that you've overfitted by 0.2% or if bayes' error is 0.1% or 0.2% or 0.3% you don't really know.

Based on the information given in this example, you actually don't have enough information to tell that, if you should focus on reducing bias or reducing variance on your algorithm. So, that slows down the efficiency where you should make progress.

→ Moreover if your training accuracy is already better than team of doctors for a certain case then it is also harder to rely just on human intuition. to tell your algorithm what are ways that your algorithm could still improve the performance?

→ So, according to last problem once you surpass 0.5% threshold your options of making progress further are less clear. Doesn't mean that you can't make progress, but some of the tools you have for pointing you in a clear direction just don't work well.

Problems where ML significantly surpasses human-level performance

- Online advertising
- Product Recommendations
- Logistics (Predicting transit time)
- Loans approvals

→ Notice that, all fours of these examples are learning from structured data where you might have a database what might users have clicked on .....

There are not natural perception problems like speech recognition or CV e.t.c

Humans tend to be very good in natural perception tasks.

→ Reason for them to surpass human level performance may be is that, in this Bigdata age computers can process bigdata with ease and find statistical patterns.

Improving your model's performance:-

We have seen so much for this, let's put all of them together to compile guidelines for improving model's performance.

Two fundamental assumptions of supervised learning:-

① You can fit the training set pretty well.
means :- You can achieve low avoidable bias. tuned by one knob such as training a bigger network

② The training set performance generalizes pretty well to the dev/test set.
means :- Variance is not too bad.
tuned by a seperate knob such as regularization or getting more training data.

Reducing (avoidable) bias and variance :-

Human Level :-
Gives Avoidable Bias
Accuracy on training sets.

Training Error :-
Gives Variance Estimate
Meaning how much hardwork to put on for the generalization of your network to dev and test set.

Dev Error :-

For this you could do the following:-

- Train bigger model
- Train longer / better optimization algorithms.
- Different NN architecture / hyperparameter search.
  - ↳ Includ changing activation function to changing no of layers or hidden units.
  - ↳ Try RNNs or CNNs.

For this you could do the following:-

- Get More data
- Regularization L2 - Dropout, data augmentation
- Different NN architecture / hyperparameter search.