practical advice

- lets say you have a model that's your first approach to a problem. it works, ok not great at all. what do you do?
  - try a smaller/larger set of features
  - get more training data
  - use different features
  - run gradient iteration for more iterations/run newton's method
  - try different hyperparameters. (λ, C etc)
  - try a different model, like SVM, linear regression, logistic regression.
- that is a lot of things to do → relying more on luck than anything else. what's a better way?
  - run a diagnostic to see what the problem is instead of just trying a bunch of different stuff.
- underfitting/overfitting problem (bias/variance tradeoff)
  - if training error is much lower than validation/test error, then it might be a case of overfitting
  - if you plot test error as a function of number of training examples and you see that it is rapidly going down, then it is likely to help to get more training examples. on the other hand if you see a flattening out, more training examples may not help.
  - training error usually grows as a function of number of examples so if you have high training error and if you get more and more examples, that's not likely to make things better.
- in case of high variance (overfitting):
  - get more training examples
  - use a smaller set of features
- in case of high bias (underfitting):
  - try a larger set of features
  - try a different set of features
- further questions:
  - is the algorithm converging? (gradient descent, newton's method etc). lets say we use SVM and BLR (bayesian linear regression).
  - are you optimizing the right function? for example, in spam classification you might care more about marking non-spam incorrectly than marking spam incorrectly. you can choose the weights in your cost function to reflect this.
  - if svm training cost is higher than blr training cost but svm does better, it might be that the optimization objective is wrong. if svm training cost is lower than blr training cost and svm does better, it might be that the algorithm used for optimization is not the right one, or we need to let it converge further.
- as a summary, look at this slide:

Diagnostics tell you what to try next

Bayesian logistic regression, implemented with gradient descent.

Fixes to try:

– Try getting more training examples.	Fixes high variance.
– Try a smaller set of features.	Fixes high variance.
– Try a larger set of features.	Fixes high bias.
– Try email header features.	Fixes high bias.
– Run gradient descent for more iterations.	Fixes optimization algorithm.
– Try Newton's method.	Fixes optimization algorithm.
– Use a different value for λ.	Fixes optimization objective.
– Try using an SVM.	Fixes optimization objective.

- Also,

Debugging an RL algorithm

The controller given by θ_{RL} performs poorly.

Suppose that:

1. The helicopter simulator is accurate.
2. The RL algorithm correctly controls the helicopter (in simulation) so as to minimize $J(\theta)$.
3. Minimizing $J(\theta)$ corresponds to correct autonomous flight.

Then: The learned parameters θ_{RL} should fly well on the actual helicopter.

Diagnostics:

1. If θ_{RL} flies well in simulation, but not in real life, then the problem is in the simulator. Otherwise:
2. Let θ_{human} be the human control policy. If $J(\theta_{human}) < J(\theta_{RL})$, then the problem is in the reinforcement learning algorithm. (Failing to minimize the cost function J.)
3. If $J(\theta_{human}) \geq J(\theta_{RL})$, then the problem is in the cost function. (Maximizing it doesn't correspond to good autonomous flight.)

Andrew Y. Ng

- It is up to you to figure out your own diagnostics for your machine learning problems.
- *Even if an algorithm is working well, it is super important to dig in and understand why it is working well → especially if its a system you are spending a lot of time maintaining and you have deployed it to production. If you're doing research on it, you better understand why it works well.*
- If you've got a big machine learning system (like a self driving car) with a bunch of ML components, how do you tell where the error is coming from? one common approach is to plug in ground truth into each component and see where the errors are.
- Ablative analysis is where lets say you have a bunch of features and you have a accuracy %. Now, remove one feature at a time and see how much the accuracy drops. This tells how sort of how much discriminative power each feature is contributing.
- "coming up with and implementing diagnostics and doing ablative analysis and other things to understand the problem better is often time very ver well spent." - andrew ng