**Evaluating a Learning Algorithm**

Table of Contents

[Evaluating a Hypothesis 3](#_Toc106495235)

[Errors vs Misclassifications 4](#_Toc106495236)

[Model Selection 5](#_Toc106495237)

[-Fold Cross Validation 6](#_Toc106495238)

[Bias and Variance 7](#_Toc106495239)

[Regularization Parameter 8](#_Toc106495240)

[Learning Curves 10](#_Toc106495241)

[Conclusions 12](#_Toc106495242)

[Neural Networks 13](#_Toc106495243)

Suppose we are using **regularized linear regression** to make predictions about housing prices. As a reminder, the cost function for this would be:

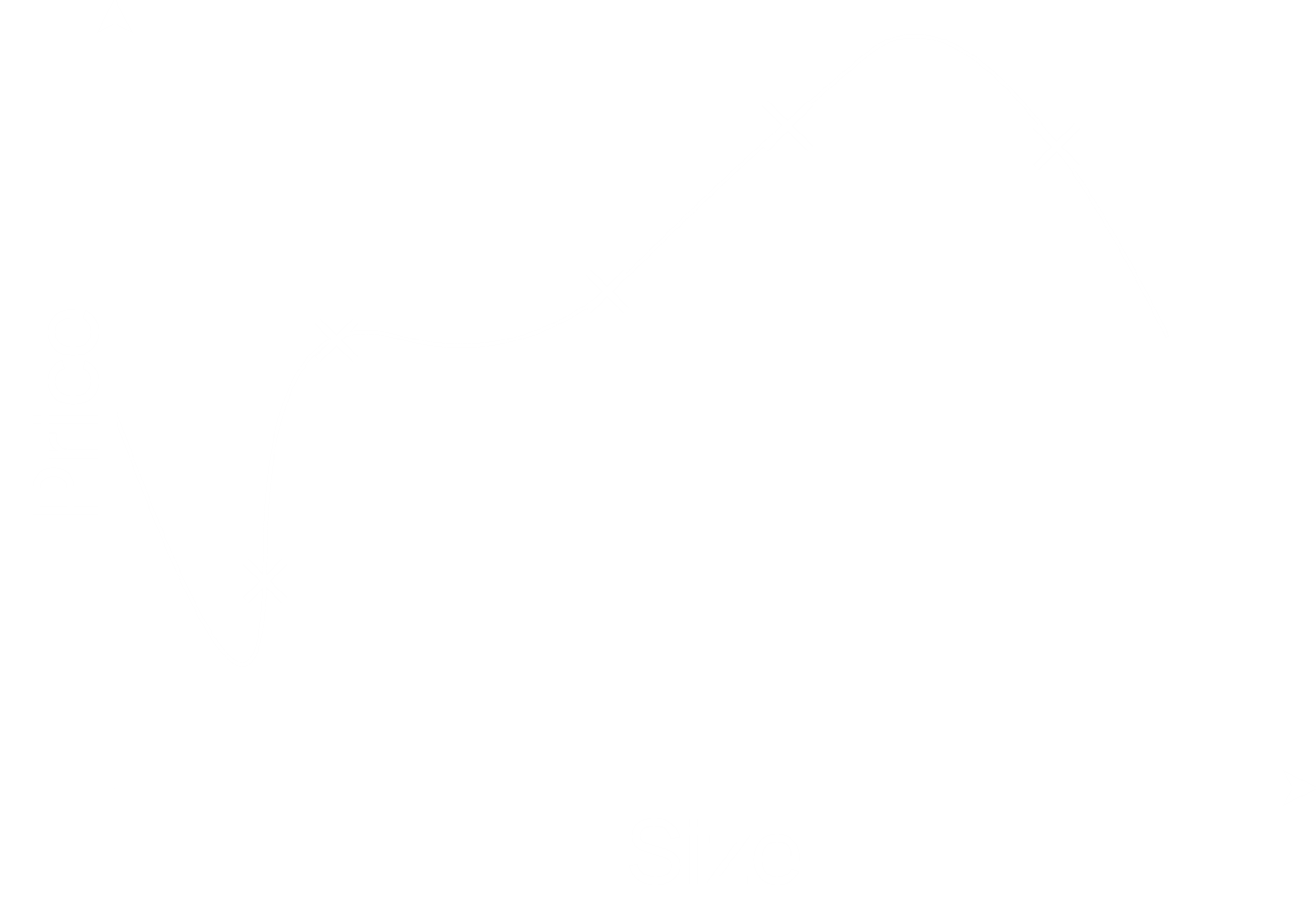
When testing our hypothesis with a new set of input values, suppose we find that there are unacceptably **high errors**. At this stage, there are several things we can try:

* Get more training data
* Use fewer features
* Use more features
* Add polynomial features
* Decrease
* Increase

To figure out which of these steps we need to take, we can run **diagnostics**, which will tell us about what is and is not working in our algorithm. Diagnostics can take some time to implement, but they are definitely beneficial, since they will allow us to avoid huge amounts of wasted time. For example, if we are working with housing prices and decide we need more information about the land, we may have to spend months collecting that data. If we were wrong about needing more features, something diagnostics could have told us, then we will have wasted several months of our time.

## Evaluating a Hypothesis

We have previously seen the problems of **overfitting** and **underfitting** in a graphical manner. For example, the graph below exhibits the overfitting problem.



Unfortunately, such graphical representations can only be used when dealing with a single feature. As we move towards more and more features, this is no longer possible.

A typical method to evaluate a hypothesis is to **split the data** into a **training** and a **testing** set. For samples, we could perhaps use of the samples for the training set, containing samples, while the other of the samples is left for the testing set, containing samples. It is best to randomly shuffle the samples before splitting them into the two subsets.

Once we have our training and testing sets, we can train our algorithm using the training set alone, not allowing it to see the testing set. Afterwards, we can use the inputs from the testing set to make **predictions** and compare the predictions with the corresponding results from the testing set. This will tell us the amount of error the predictions have with respect to the testing set.

If our model exhibits very low error with the training set and very high error with the testing set, this is an indication that our model is **overfitting** the training data and thus failing to generalize.

## Errors vs Misclassifications

For **classification problems** specifically, there is an alternative method to evaluate the hypothesis, which is to measure the percentage of **misclassifications** the model makes. For example, out of test samples, if the model misclassifies of them, then the model can be said to have error.

However, this method has one issue. Regardless of where the actual data point is, the method will report that there either is an error or there isn’t. It will not take into account how bad the error was, i.e. whether the error was actually quite close to the true data point or whether the error was far off. This information is provided by a normal error calculation, i.e. calculate the distance between the error point and the ground truth.

## Model Selection

We now know how to evaluate our model, so let’s say we take a bunch of models and start trying them out.

10.

We split our samples into a training set and a testing set, train each of the models based on the training set and evaluate them using the testing set. Finally, we choose the model which gives us the best accuracy on the testing set. Easy enough right?

The problem is, our model selection process is now technically **overfitting the testing set**. It is picking the model which best fits the chosen testing set. Now, we have no choice but to go through this process, but we need to recognize that the accuracy we will get for general cases will still not be as high as the one we will see with the testing set. So how do we figure out what the accuracy will be for **general cases**?

This is where the **validation set**, also called the cross-validation set, comes in. Instead of splitting our data into a training and a testing set, we can split it into training, validation and testing sets. We can then train all our models on the training set, but instead of choosing the one which performs best on the testing set, we choose the one that performs best on the validation set. Once we have picked a model, we run it on the testing set to get the final error, which we can report as the error our model will make on general cases.

A typical ratio to split the data would be for the training set, for the validation set and for the testing set.

We can also use this same process when we are choosing other **hyperparameters** such as , , etc.

## -Fold Cross Validation

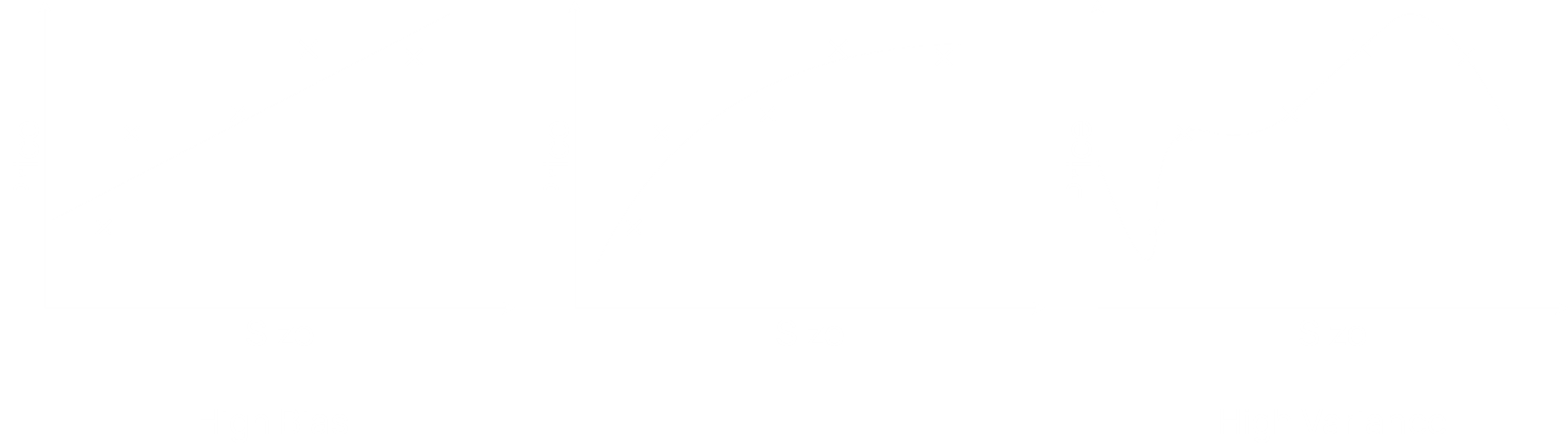
For the keen eyed, there is a further problem in the above process. What if we use the validation set as the test set and the test set as the validation set? More generally, different runs will use different sections of the data as the test and validation sets. Won’t this change the outcome?

The solution to this is to use **-Fold Cross Validation**. Essentially, we split our data into a number of sections. Suppose we have sections. Afterwards, we choose sections for training and section for testing. We then test our model and get some error on the test set. Next, we choose a different section as the test set. We still have 1 section as the test set and 9 sections as the training set. We test the model again and get a different error on the test set. In this manner, we keep going until we have all 10 combinations and 10 different error results. We then present the average error as the error for our model.

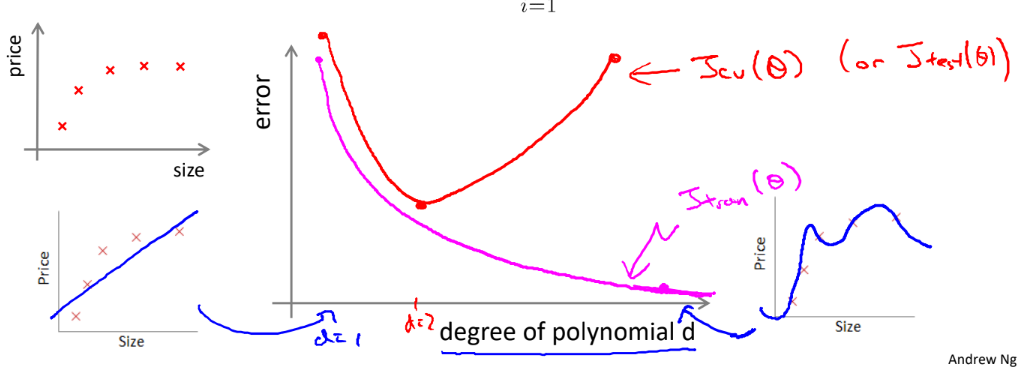
Another issue we need to keep an eye on here is how much the error values for the different combinations **varies**. If the average accuracy is 80%, the highest accuracy is 81% and the lowest one is 79%, then the variance is of 1%, which is good. However, if we are getting very large variance values, this indicates that the performance of our model will vary wildly depending on what data we provide to it. This indicates that our model is perhaps not a good one for general data.

## Bias and Variance

Bias and variance are two of the most common problems that cause poor results for our models. We have already seen both of these issues graphically, as shown below:



We can now examine these problems with reference to training and test sets. Note that from here on out, the variable will be used to denote the **degrees** of polynomials in our model. Thus, a lower value of may give us high bias while a higher value of may give us high variance.

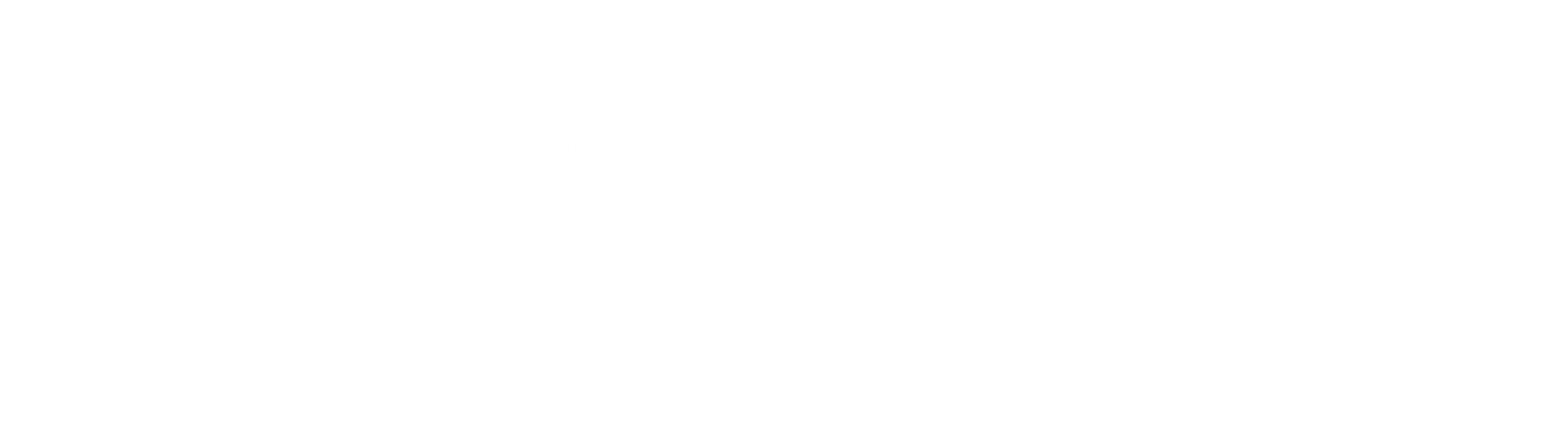


As can be seen in the graph above, as the value of increases, the cost for the **training set** decreases exponentially. This is because the models overfit the training set more and more. However, since they are overfitting, they fail to generalize to the **validation** **set**. The cost for the validation set decrease towards the beginning as the models begin to take on the general shape of the data. At one point, we reach a turning point, which is the perfect place to stop. If we keep increasing the value of , the models will begin to overfit and the cost for the validation set will increase again.

From this same graph, we can also understand whether we have a high bias problem or a high variance problem. Towards the beginning of the graph, where the errors for both the validation and the training sets were high, we had a **high bias** problem. Towards the end of the graph, where the error for the validation set was high but the error for the training set was low, we had a **high variance** problem.

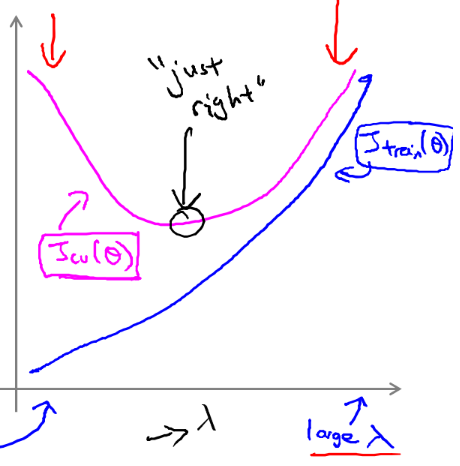
### Regularization Parameter

When figuring out the regularization parameter, , we will find a similar trend. If the value of is too large, we will suffer from **high bias**. If it is too small, we will suffer from **high variance**.



To choose the right value of , we will again define multiple models, each with a different value of . However, in this case, when calculating the errors for each of the training, validation and test sets, we will not take the **regularization term** into account. Thus, the equations will look like this:

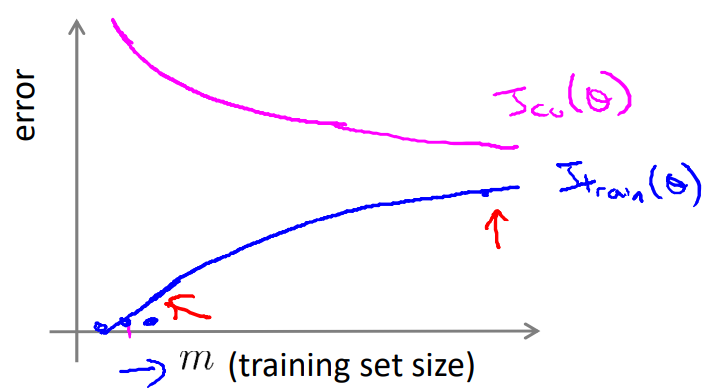
As we keep **increasing** the value of , the **training set error** will keep increasing. Remember that we are taking an overfitted model and using to force it to generalize, which is why this happens. The error for the **testing set** will behave as before, being high at the overfitted state, decreasing to a minimum point, and increasing again as the model becomes underfitted.



## Learning Curves

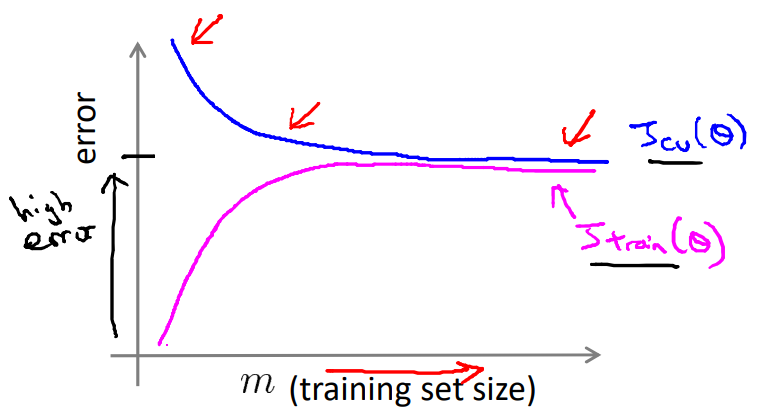
**Learning Curves** are a diagnostic tool that helps us identify problems in our model. It is simply a graph of the number of samples we have against the amount of error in the results. We plot the graphs for both the training set and the test set.

Consider the **ideal situation** first, where the model is more or less correct.



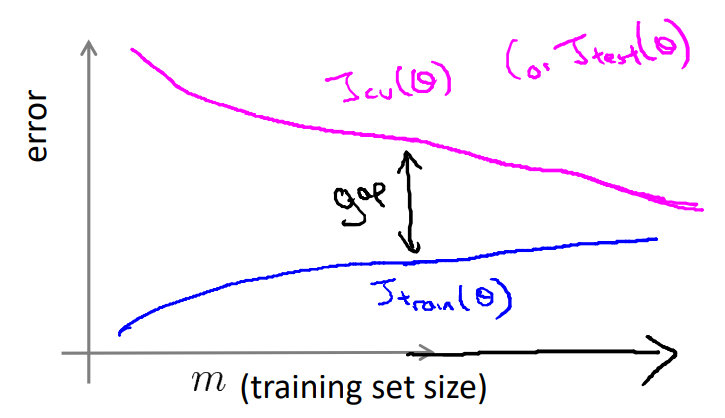
With very few samples, the training set will almost perfectly fit the data, which is why it will have very little error. However, the model will not generalize, which is why the test set will have a huge error. As we add more data, the model will be able to generalize. This will cause the test error to decrease, but the training error will increase, since it will not overfit the training data. No matter how many samples we add though, the test error will always be greater than the training error.

Now suppose we have a model with **high bias**, meaning the model does not fit the data well. For the test set, the error will quickly flatten and remain high, because the model is just wrong. Even for the training set, the error will rise quickly and get to a high value before flattening out. Again, all this proves is that the model itself is incorrect. The error values for the training and testing set will be very close in this case.



This graph tells us something important though. If we have a model with high bias, increasing the amount of training samples we have is not going to help.

Finally, consider the case in which the model has **high variance**. For such a model, the training set error will not increase much as we add more samples. This is because the model will just fit to that data. Interestingly though, the test set error will keep decreasing as we add more and more data. Towards the start, the model failed to generalize, which is why we had a larger error on the test set. This is in fact how we identify the high variance problem, the fact that we have a low error on the training set but a high error on the test set. However, as we add more data, the model is adjusting to all the data there is. The test set will have less and less new data on which the model can make mistakes. Think of it like this, if we give the model all the data there is in the world, the test set will have no error, because even though the model is overfitted, it has overfitted to all of the data there is.



Thus, in the case of high variance, getting more data will help us.

## Conclusions

Based on the things we have learnt above, let us revisit the choices we have when our model is not performing well and re-examine which steps we should take in which situation.

* Get more training data – We now know that this will help only in the case of high variance and not in the case of high bias.
* Use fewer features – This will help in the case of high variance, since the model will be able to generalize.
* Use more features – This will help in the case of high bias, since the model will be able to better fit the data.
* Use polynomial features – This is similar to adding more features and thus helps with high bias.
* Decrease – This will result in less regularization, which will help fix high bias.
* Increase – This will result in more regularization, which will help with high variance.

## Neural Networks

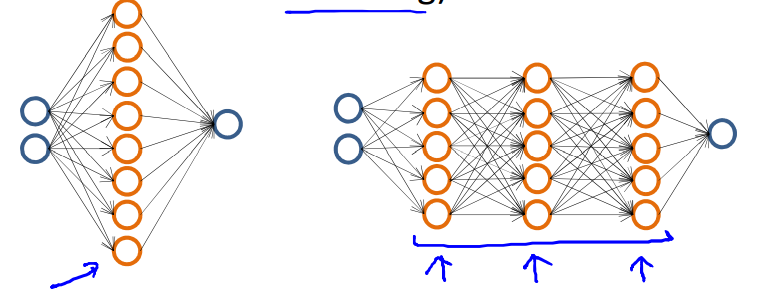
So, how does all of this come into play when dealing with a **neural network**.

One option when modelling a neural network is to use a **small** one, such as the one below:



Such a neural network will have **fewer parameters**, which makes it more prone to **underfitting**. However, they are less computationally expensive.

Alternatively, we may choose a **large** neural network, either will many neurons in a few layers or with many layers with fewer neurons.



Such a neural network has **more parameters** and is thus likely to suffer from **overfitting**. However, they are more computationally expensive.

Generally though, it is better to use a larger neural network and address overfitting using regularization. This is because we are then able to properly work with complex models. The computational expense is becoming less and less of a problem over time as devices become more powerful.

More specifically, it is usually better to use **deeper** networks, with many layers. This is again to deal with complex models. Consider that we want a face recognition application. In this case, the lower layers would do things like detect edges of the face, the middle layers would perhaps create parts of the face, such as the eye or the nose, and the later layers would combine things to create the entire face. Since the later layers are doing more complex work, increasing the number of layers will distribute the work more and make the model more accurate.

