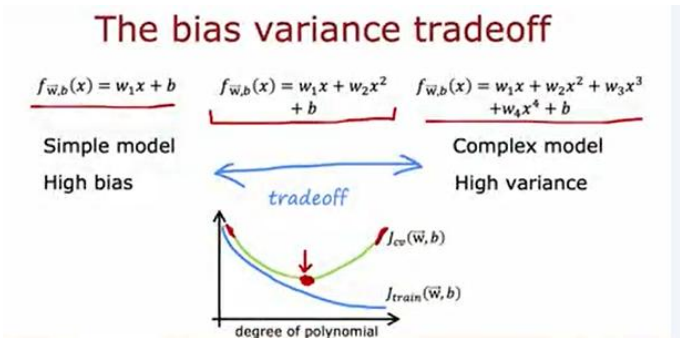
Bias/variance and neural networks

Was seen that high bias or high variance are both bad in the sense that they hurt the performance of your algorithm. One of the reasons that neural networks have been so successful is because your networks, together with the idea of big data or hopefully having large data sets. It's given us a new way of new ways to address both high bias and high variance.

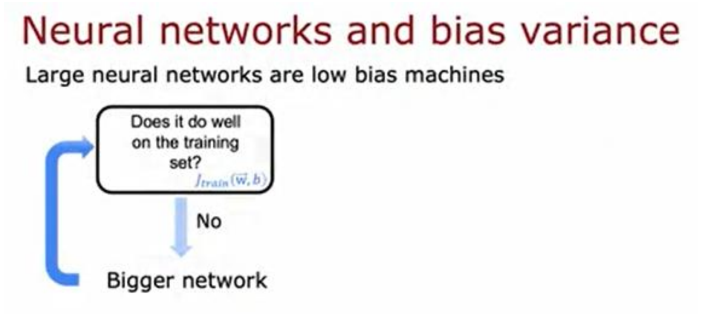
Let's take a look. You saw that if you're fitting different order polynomial is to a data set, then if you were to fit a linear model like this on the left. You have a pretty simple model that can have high bias whereas you were to fit a complex model, then you might suffer from high variance. And there's this tradeoff between bias and variance, and in our example it was choosing a second order polynomial that helps you make a tradeoff and pick a model with lowest possible cross validation error.



The bias variance tradeoff
$f_{\vec{w},b}(x) = w_1x + b$
$f_{\vec{w},b}(x) = w_1x + w_2x^2 + b$
$f_{\vec{w},b}(x) = w_1x + w_2x^2 + w_3x^3 + w_4x^4 + b$
Simple model
High bias
tradeoff
Complex model
High variance
$J_{cv}(\vec{w},b)$
$J_{train}(\vec{w},b)$
degree of polynomial

And so before the days of neural networks, machine learning engineers talked a lot about this bias variance tradeoff in which you have to balance the complexity that is the degree of polynomial. Or the regularization parameter longer to make bias and variance both not be too high. And if you hear machine learning engineers talk about the bias variance tradeoff.

This is what they're referring to where if you have too simple a model, you have high bias, too complex a model high variance. And you have to find a tradeoff between these two bad things to find probably the best possible outcome. But it turns out that neural networks offer us a way out of this dilemma of having to tradeoff bias and variance with some caveats. And it turns out that large neural networks when trained on small term moderate sized datasets are low bias machines.

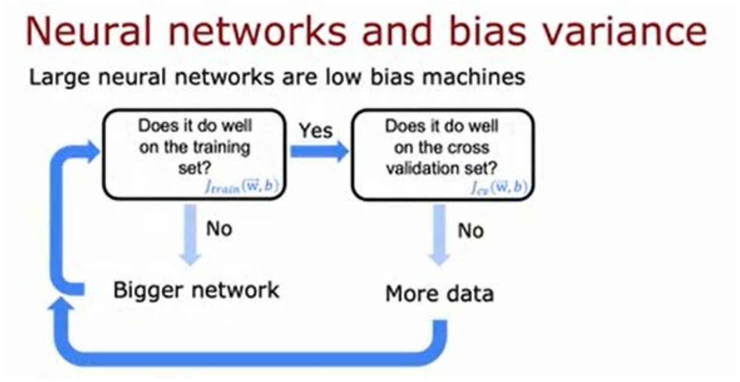


Neural networks and bias variance
Large neural networks are low bias machines
Does it do well on the training set?
$J_{train}(\vec{w},b)$
No
Bigger network

And what I mean by that is, if you make your neural network large enough, you can almost always fit your training set well. So long as your training set is not enormous. And what this means is this gives us a new recipe to try to reduce bias or reduce variance as needed without needing to really trade off between the two of them.

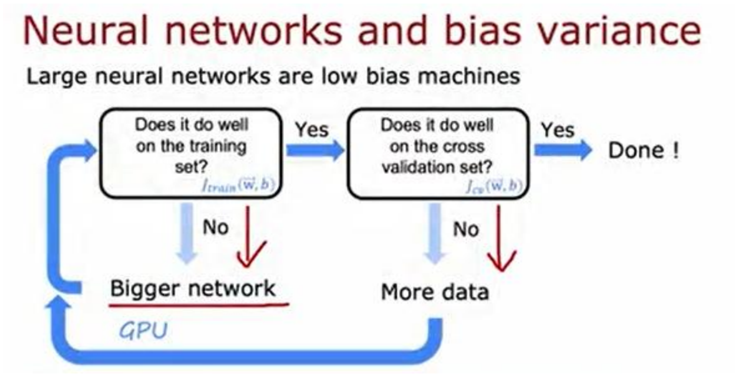
So let me share with you a simple recipe that isn't always applicable. But if it applies can be very powerful for getting an accurate model using a neural network which is first train your algorithm on your training set and then asked does it do well on the training set. So measure Jtrain and see if it is high and by high, I mean for example, relative to human level performance or some baseline level of performance and if it is not doing well then you have a high bias problem, high trains error.

And one way to reduce bias is to just use a bigger neural network and by bigger neural network, I mean either more hidden layers or more hidden units per layer.


Neural networks and bias variance
Large neural networks are low bias machines
Does it do well on the training set?
$J_{train}(\vec{w},b)$
Yes
Does it do well on the cross validation set?
$J_{cv}(\vec{w},b)$
No
Bigger network
No
More data

And you can then keep on going through this loop and make your neural network bigger and bigger until it does well on the training set. Meaning that achieves the level of error in your training set that is roughly comparable to the target level of error you hope to get to, which could be human level performance. After it does well on the training set, so the answer to that question is yes. You then ask does it do well on the cross validation set?

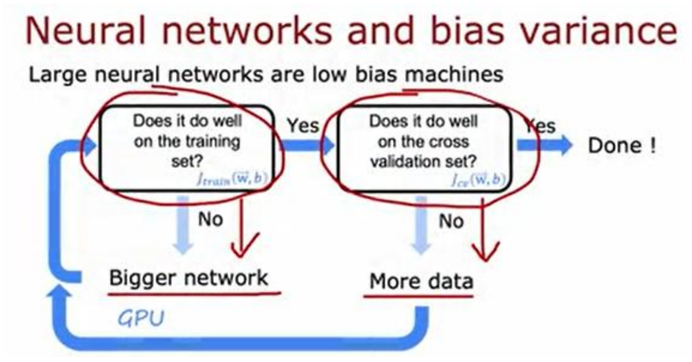
In other words, does it have high variance and if the answer is no, then you can conclude that the algorithm has high variance because it doesn't want to train set does not do on the cross validation set. So that big gap in Jcv and Jtrain indicates you probably have a high variance problem, and if you have a high variance problem, then one way to try to fix it is to get more data. To get more data and go back and retrain the model and just double-check, do you just want the training set?


Neural networks and bias variance
Large neural networks are low bias machines
Does it do well on the training set?
$J_{train}(\vec{w},b)$
Yes
Does it do well on the cross validation set?
$J_{cv}(\vec{w},b)$
Yes
Done !
No
Bigger network
GPU
No
More data

If not, have a bigger network, or it does see if it does when the cross validation set and if not get more data. And if you can keep on going around and around and around this loop until eventually it does well in the cross validation set. Then you're probably done because now you have a model that does well on the cross validation set and hopefully will also generalize to new examples as well.

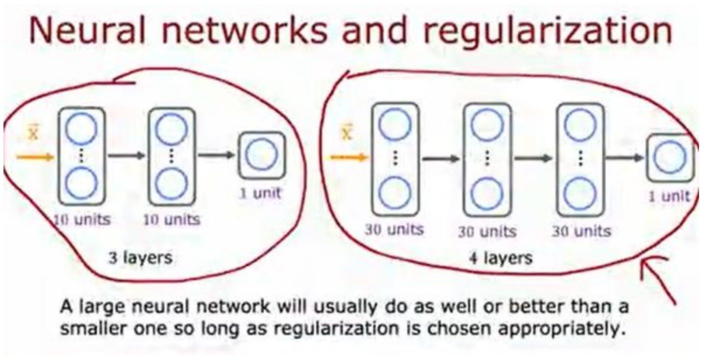
Now, of course there are limitations of the application of this recipe training bigger neural network doesn't reduce bias but at some point it does get computationally expensive. That's why the rise of neural networks has been really assisted by the rise of very fast computers, including especially GPUs or graphics processing units. Hardware traditionally used to speed up computer graphics, but it turns out has been very useful for speeding on neural networks as well.

But even with hardware accelerators beyond a certain point, the neural networks are so large, it takes so long to train, it becomes infeasible.


Neural networks and bias variance
Large neural networks are low bias machines
Does it do well on the training set?
$J_{train}(\vec{w}, b)$
Yes
Does it do well on the cross validation set?
$J_{cv}(\vec{w}, b)$
Yes
Done !
No
No
Bigger network
More data
GPU

And then of course the other limitation is more data. Sometimes you can only get so much data, and beyond a certain point it's hard to get much more data. But I think this recipe explains a lot of the rise of deep learning in the last several years, which is for applications where you do have access to a lot of data. Then being able to train large neural networks allows you to eventually get pretty good performance on a lot of applications.

One thing that was implicit in this slide that may not have been obvious is that as you're developing a learning algorithm, sometimes you find that you have high bias, in which case you do things like increase your neural network. But then after you increase your neural network you may find that you have high variance, in which case you might do other things like collect more data. And during the hours or days or weeks, you're developing a machine learning algorithm at different points, you may have high bias or high variance.

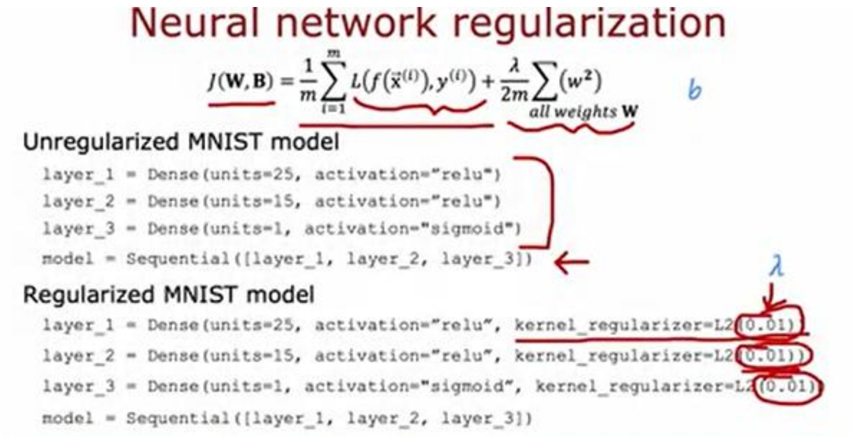

Neural networks and regularization
$\vec{x}$
10 units
10 units
1 unit
3 layers
$\vec{x}$
30 units
30 units
30 units
1 unit
4 layers
A large neural network will usually do as well or better than a smaller one so long as regularization is chosen appropriately.

And it can change but it's depending on whether your algorithm has high bias or high variance at that time. Then that can help give guidance for what you should be trying next. When you train your neural network, one thing that people have asked me before is, hey Andrew, what if my neural network is too big? Will that create a high variance problem? It turns out that a large neural network with well-chosen regularization, well usually do as well or better than a smaller one.

And so for example, if you have a small neural network like this, and you were to switch to a much larger neural network like this, you would think that the risk of overfitting goes up significantly. But it turns out that if you were to regularize this larger neural network appropriately, then this larger neural network usually will do at least as well or better than the smaller one.

So long as the regularization has chosen appropriately. So another way of saying this is that it almost never hurts to go to a larger neural network so long as you regularized appropriately with one caveat, which is that when you train the larger neural network, it does become more computational e expensive. So the main way it hurts, it will slow down your training and your inference process and very briefly to regularize a neural network.

This is what you do if the cost function for your neural network is the average loss and so the loss here could be squared error or logistic loss. Then the regularization term for a neural network looks like pretty much what you'd expect is lambda over two m times the sum of w squared where this is a sum over all weights W in the neural network and similar to regularization for linear regression and logistic regression, we usually don't regularize the parameters be in the neural network although in practice it makes very little difference whether you do so or not.


Neural network regularization

$$\underline{J(\mathbf{W},\mathbf{B})} = \underline{\frac{1}{m}\sum_{i=1}^{m} \underbrace{L(f(\vec{x}^{(i)}),y^{(i)})}} + \underbrace{\frac{\lambda}{2m}\sum_{\text{all weights } \mathbf{W}}(w^2)} \qquad b$$

Unregularized MNIST model

```
layer_1 = Dense(units=25, activation="relu")
layer_2 = Dense(units=15, activation="relu")
layer_3 = Dense(units=1, activation="sigmoid")
model = Sequential([layer_1, layer_2, layer_3])
```

Regularized MNIST model

```
layer_1 = Dense(units=25, activation="relu", kernel_regularizer=L2(0.01))
layer_2 = Dense(units=15, activation="relu", kernel_regularizer=L2(0.01))
layer_3 = Dense(units=1, activation="sigmoid", kernel_regularizer=L2(0.01))
model = Sequential([layer_1, layer_2, layer_3])
```

And the way you would implement regularization in tensorflow is recall that this was the code for implementing an unregulated Rised handwritten digit classification model. We create three layers like so with a number of fitting units activation And then create a sequential model with the three layers.

If you want to add regularization then you would just add this extra term colonel regularize A equals l. two and then 0.01 where that's the value of longer in terms of though actually lets you choose different values of lambda for different layers although for simplicity you can choose the same value of lambda for all the weights and all of the different layers as follows. And then this will allow you to implement regularization in your neural network.

So to summarize two Takeaways, I hope you have from this video are one. It hardly ever hurts to have a larger neural network so long as you regularize appropriately. one caveat being that having a larger neural network can slow down your algorithm. So maybe that's the one way it hurts, but it shouldn't hurt your algorithm's performance for the most part and in fact it could even help it significantly. And second so long as your training set isn't too large. Then a neural network, especially large neural network is often a low bias machine.

It just fits very complicated functions very well, which is why when I'm training neural networks, I find that I'm often fighting variance problems rather than bias problems, at least if the neural network is large enough. So the rise of deep learning has really changed the way that machine learning practitioners think about bias and variance.

Having said that even when you're training a neural network measuring bias and variance and using that to guide what you do next is often a very helpful thing to do. So that's it for bias and variance. Let's go on to the next video. We will take all the ideas we've learned and see how they fit in to the development process of machine learning systems. And I hope that we'll tie a lot of these pieces together to give you practical advice for how to quickly move forward in the development of your machine learning systems