FYS-STK4155 - Extra exercise

Kari Lovise Lodsby

December 17, 2021

Abstract

In this exercise, I will look at the bias-variance tradeoff when the complexity of a method used to solve a regression problem (regression to a noisy Franke function) increases. The bias-variance tradeoff arises when the complexity of a model increases: the model gets better at modelling the signals in the data, which decreases bias, but if the complexity is too high, the model will also "learn" the noise, which increases the variance. This trade-off means that higher complexity will not necessarily yield a lower mean-squared error.

The first model I will look at is linear regression, using the common ordinary least squares method and the Ridge method with the polynomial degree as the complexity. The second is deep learning, where I used the number of layers in a neural network as the complexity. Finally, I looked at decision trees with the maximum depth as the measure of complexity. For the linear regression and neural networks, I reused code from projects 1 and 2. For the decision tree, I used sklearn's implementation.

1 Introduction

When we evaluate an algorithm, we wish to minimize its mean-squared error. This error can be decomposed as the sum of the bias and variance. Bias happens when erroneous assumptions in the learning algorithm cause it to miss patterns in the data (underfitting) because the model is not complex enough to pick them up. Variance is a type of error caused by sensitivity to small fluctuations in the training set (overfitting), as the model ends up picking up the noise from the training data in addition to the desired patterns.

There are many algorithms that can be used to solve regression problems. I will look at three methods in this paper: linear regression by ordinary least squares approximation, deep learning by a neural network, and decision trees. I will focus my discussion on decision trees, as I have talked about linear regression and neural networks in previous projects.

A decision tree is a non-supervised learning method for classification and regression. It is performed by breaking down a dataset into smaller and smaller subsets while developing an associated decision tree. This tree is supposed to predict the value of a variable after inferring simple decision rules from the data set. For regression, it can be seen as piecewise constant interpolation.

This type of algorithm has several advantages over neural networks: it is simple to understand and interpret, as the trees can be visualized and the conditions can be explained with simple Boolean logic. The cost of using the tree is logaritmic to the number of data points used to train the tree, so it is not as computationally expensive as deep learning.

On the other hand, the method can create over-complex trees that fail to generalize well. This suggests that decision trees are particularly suspectible to overfitting, which will result in bad mean squared errors for high maximum depths.

2 Description of the method

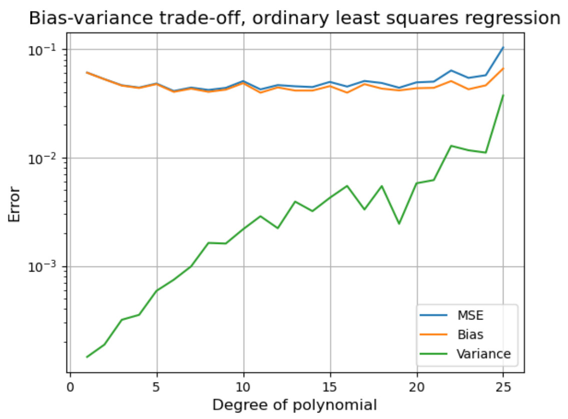
The function I used was the Franke function sampled at 2000 points with a noise factor of 0.2. I fixed the seed so that each run would be the same, and each algorithm would be tested on the same data set.

For the ordinary least squares test, I used a maximum polynomial degree of 25, as the program is fast. For Ridge regression, I limited it to 15 because of its relative slowness.

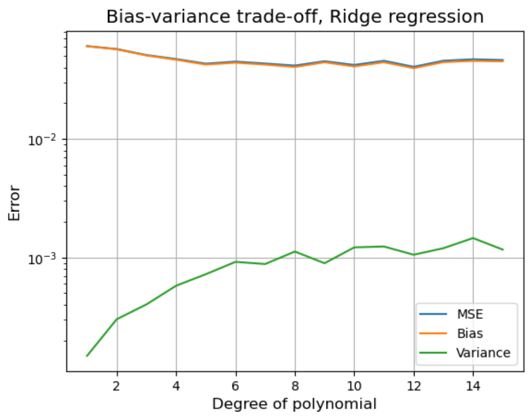
For decision trees, I set the maximum depth to 30. This was probably more than necessary, considering that I had a data set of 2000 points to work with, which limits the number of subsets it could be split into.

For the neural network, I only tried a small number of depths because the program was so slow.

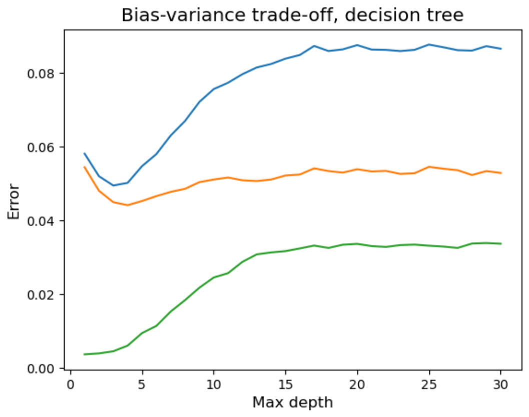
3 Results, discussion and conclusion



For the ordinary least squares method, the variance clearly increases as the polynomial degree increases. The bias decreases until around degree 5, where it seems to flatten out. This makes sense, as OLS is a low-bias method. Interestingly, polynomials of very high degrees had relatively high bias as well. This may be because polynomials of very high degrees are prone to undesirable behaviour in general (such as Runge's phenomenon when attempting to interpolate functions), or because of the specific problem I was looking at. The runtime of this program was fairly quick - it did not take many seconds to test every degree.



Ridge regression is a method that adds some bias in exchange for decreasing variance, which this graph shows. Otherwise, the patterns are similar to those for OLS regression. This method increased the runtime a bit, but not too much.

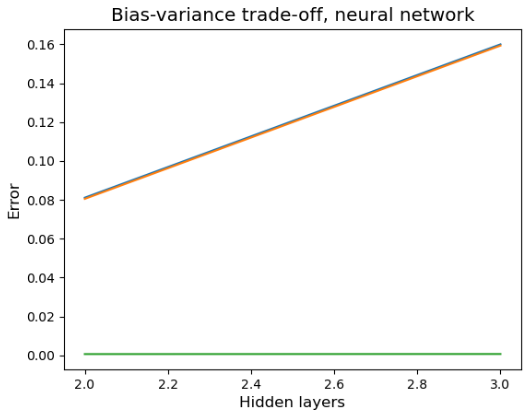


For decision trees, we observe that the MSE decreases up to around three layers, and then increases again before plateauing around 15 layers. I believe the plateau is explained by the fact that the generation of a decision tree involves breaking down a data set into smaller and smaller subsets, and since I only had 2000 data points, it may have reached a point where most ”subsets” were down to a single point and could not be split further, meaning that the variance could not increase further.

A decision tree is a model of low bias and high variance, as it makes almost no assumptions about the target function, and is very susceptible to variance in the data. (Indeed, increasing the noise factor to 0.5 yielded a lot of variance.) I did not experiment with them, but it is possible to use ensemble algorithms (such as bootstrapping aggregation and random forests) to reduce variance at the cost of adding some bias.

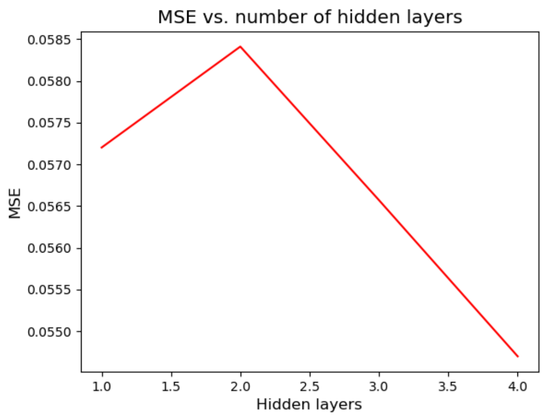
Again, I get the result that the bias also increases again after a certain point, though unlike for OLS, it does not go back to its previous maximum. It is notable that the bias was low in the first place - the effect of the bias increasing again is less noticeable if the data is noisier.

This method had reasonable runtimes. (The color coding is the same as for the linear regression tests, but the legend did not show up in the graph for some reason.)



I tried a deep learning method with three different outputs, but this appears to have yielded little useful information. It also had an extremely long runtime of several hours, so I decided to abandon this attempt.

I decided to use a simpler program that only checks the MSE, to test 2-4 layers instead of 2-3, and cut down the number of data points to 200 to curb the runtimes (though they were still much longer than the other methods). However, it is worth noting that my extreme run times for machine learning include the time to train the neural network. Once the training has been completed, it is possible to save the parameters and run the model, which should be fast and accurate if it was made well.



While it is odd that 1 layer yielded a better result than 2, it seems that the MSE decreases when more layers are added, at least in this range. However, it is consistent with results that say that it is possible to decrease both the bias and variance for neural networks by increasing the complexity.

I do not know what would happen with even more layers, or if changing other network design features would change the curve. The high learning times limited my ability to experiment with this and with network designs in general.

4 Conclusion

For this kind of regression problem, I would just stick to linear regression in most cases because it is fast, easy to implement and reasonably accurate. It also does not suffer from the problem of decision trees being partwise constant.

However, I chose to solve a single problem using a fairly simple model, which plays to the strengths of linear regression. If I was going to look at a big, complex data set, machine learning would probably be good. In this case, the deep learning would be good at picking up the complexities of the data, and the training time (and the time to find a good network design!) is easier to accept if the resulting model will be used for numerous problems instead of just one.