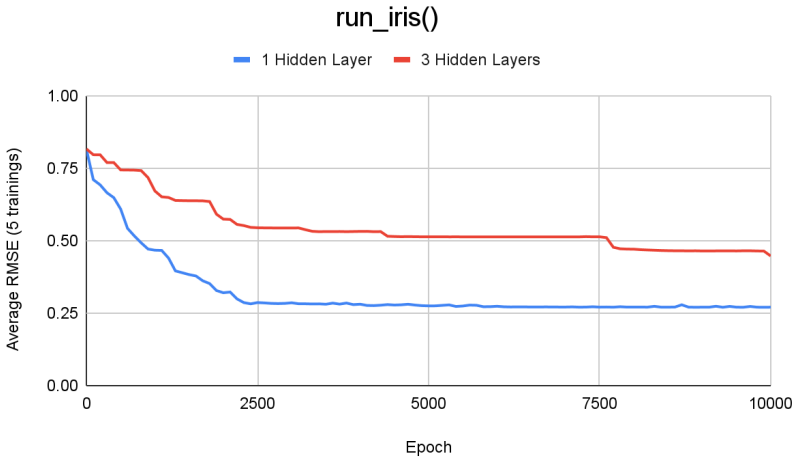
Discussion for results of each run

When running the Neural Network on the iris, sin, and XOR dataset, the RMSE had an overall decrease when training the neural network. I believe this indicates that the neural network is able to successfully update its weights to improve the accuracy of its predictions. When testing the neural network, the RMSE was slightly greater than the final RMSE of the training phase. I think that this suggestions that the neural network is not overfitting, since the RMSE value is not exactly the same as the RMSE for the last training Epoch.

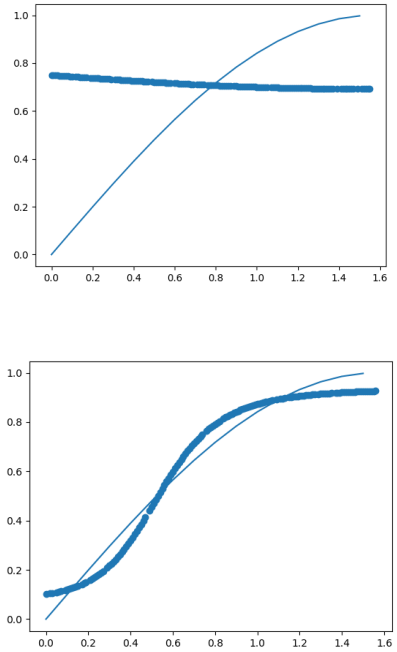
Effect of altering the number of hidden layers, learning rate, and epochs of the network

Number of Hidden Layers:

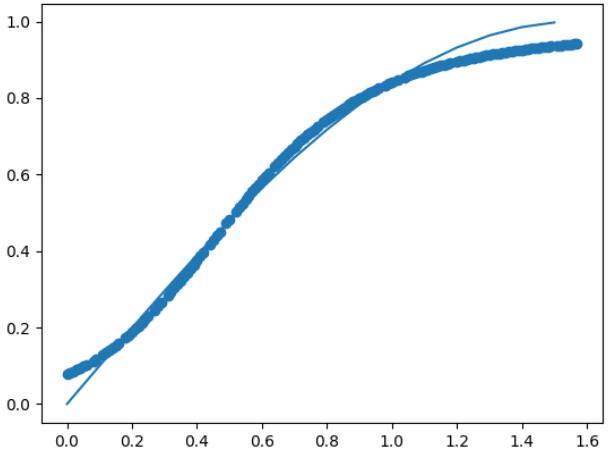


In general, increasing the number of hidden layers did not seem to increase the accuracy of the network's predictions. In fact, it does the opposite (the RMSE increases with the number of hidden layers). According to the assignment, one explanation for this is that the more layers a neural network has, the greater the probability of the network getting "stuck." This phenomenon is more clearly shown for iris dataset, since the decrease in RMSE value when the network has 3 hidden layers "flattens out" much more quickly compared to when the network just has 1 hidden layer.

3 Hidden Layers



1 Hidden Layer

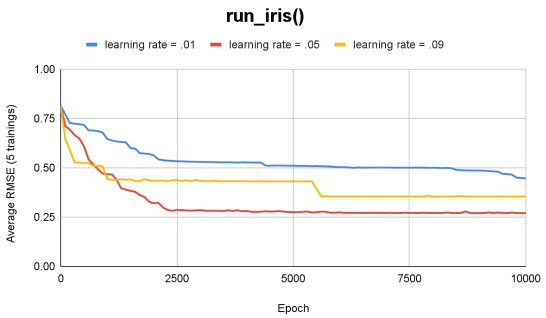


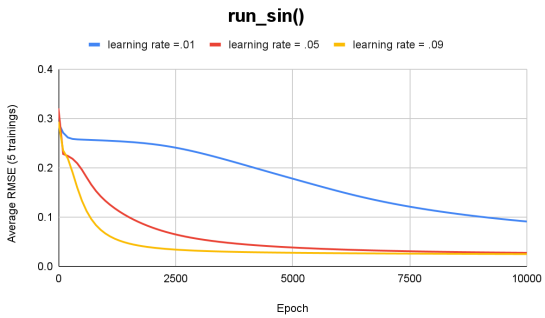
Comparing the output generated by the network when testing the network on the sin dataset seems to support idea that less hidden layers results in greater accuracy. The sin curve generated by this network when it has 3 hidden layers does not resemble the actual sin curve as closely compared to the network when it has just 1 hidden layer.

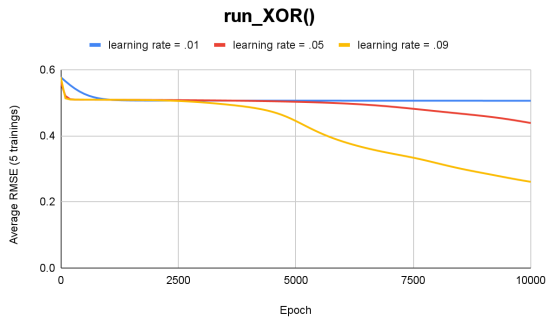
In addition, I noticed that increasing the number of hidden layers increased the time it took to train and test this network. This makes sense, since the number of calculations and and weight adjustments would increase with the number of hidden layers. In the case of the datasets we are testing, it seems like using 1 hidden layer would be more optimal in terms of accuracy and efficiency. I was also thinking about how the datasets our network classifies does not have a large number of input and output nodes. In this case, the hidden layers would be excessive; however, for a dataset with a greater number of input and output nodes, perhaps having more

than 1 hidden layer would be optimal because having more layers (and neurodes) would allow the network to take care of "subfeatures" of the complex features. (The hidden layer figure for the XOR dataset is in the Extra Credit pdf.)

Learning rate:







I found it surprising that for **run_iris()**, the RMSE for a learning rate of .09 was actually higher overall compared to the RMSE for a learning rate of .05. For **run_sin()**, it seems like the RMSE for a learning rate of .09 is just slightly lower than the RMSE for a learning rate of .05. For **run_XOR()**, the RMSE for a learning rate of .05 becomes slightly lower than the RMSE for a learning rate of .01, while the RMSE for a learning rate of .09 became much smaller than the RMSE for the other learning rates. For my graphs, the RMSEs shown are the averages of 5 trainings for 10,000 epochs.

One consistency for both of the graphs is that the RMSE for learning rates of .09 and .05 were lower than the learning rate of .01. One possible reason may be because if the learning rate is too low, the network will not adjust the weights as much compared to a learning rate that is higher. It would be interesting to see whether a really high learning rate would decrease the accuracy of the network (since this may cause the network to adjust the weights too much).

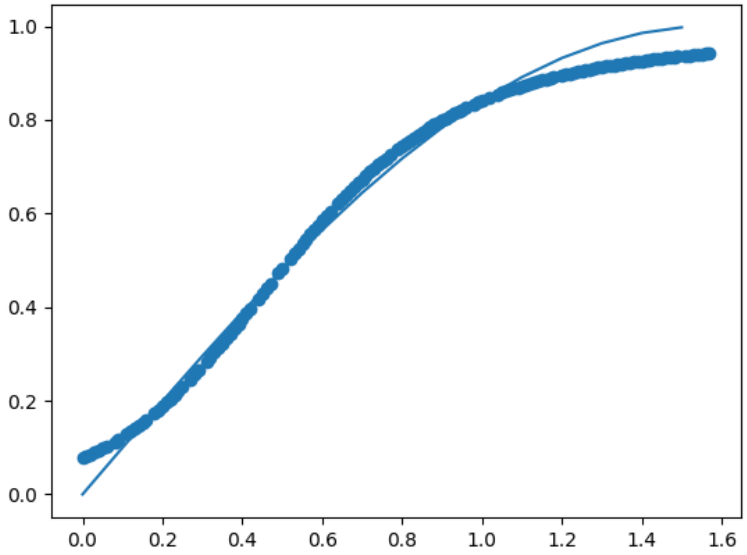
Number of Epochs:

	5,000 Epochs	10,000 Epochs	20,000 Epochs
run_iris()	Average Testing RMSE: 0.385	Average Testing RMSE: 0.390	Average Testing RMSE: 0.259
run_sin()	Average Testing RMSE: 0.0333	Average Testing RMSE: 0.0306	Average Testing RMSE: 0.0301
run_XOR()	Average Testing RMSE: 0.357	Average Testing RMSE: 0.316	Average Testing RMSE: 0.300

For this table, the RMSEs shown are the averages of 5 trainings for 10,000 epochs.The overall trend is as the number of epochs increases for the training phase, the RMSE for the testing phase decreases. One exception is **run_iris()**, where the RMSE for 5,000 epochs is slightly lower than the RMSE for 10,000 epochs. Perhaps if I had increased the number trainings, the RMSE for 5,000 epochs would be higher than the RMSE for 10,000 epochs. Nonetheless, I think that the overall pattern of the accuracy improving for the testing phase as the number of epochs during the training phase increases still holds.

Intuitively, this makes sense since the more practice the network gets with a certain dataset, the more it is able to “refine” its weights to make a more accurate prediction. This does make me wonder whether it is possible to “overtrain” a model by increasing the number of epochs. If the training dataset is large and is representative of the data “population,” maybe increasing the number of epochs will not be too much of an issue. However, I could see how training a network extensively on a small, nonrepresentative dataset would do more harm than good in terms of generalization.

Discussion of where the network succeeded and failed



Graph of the sin curve produced by the test method superimposed on an actual sin graph from 0 to pi/2

This network succeeded in the sense that after training, the network is able to adjust its weights so that it can predict the label corresponding to a feature more accurately over time. Thus, when we are testing the network, its predictions are not arbitrary and are better than the network's initial predictions during training. As depicted in the figure above, while the sin curve generated by the network during testing does not perfectly resemble an actual sin curve, the output is fairly similar.

A shortcoming of this network can also be observed in Figure 1. The beginning of the sin curve produced by the network during testing is not very "linear," and the towards the end, the output the network generates deviates slightly from the actual sin curve.

This could be due to the neural network not being trained equally on all portions of the sin graph. That is to say, the network does not have much experience training with the "beginning" of the sin curve (from about 0.0 - 0.1) and the "end" of the sin curve (from about

1.1 - 1.6). The probability of randomly selecting a point on the “middle portion” of the sin curve (from about 0.1 - 1.1) to be in the training dataset is much greater than selecting a point that is on the “beginning” and end of the sin curve.

One solution to this is to ensure that the training dataset for the network is more representative of the dataset as a whole. This way, the network would be trained in such a way that it is able to generalize better. (While I am talking specifically about the sin dataset in this case, I believe that this solution could also apply to the other datasets we were testing the network on).

Another shortcoming of this network is that if the values of the input neurodes are zero, no training will occur because the weights for the input and neighboring hidden neurodes would be zero. This can be solved by including a bias node, which would provide a constant input to the first hidden layer, which would prevent the weights from being zero.