Rob Lindsay **Assignment 3** **Machine Learning 6341**

EXECUTIVE SUMMARY

Data

This report serves to explain how different parameters for Artificial Neural Networks produce vastly different classification results. The data that was used was found in the SeoulBikeData.csv. The data contains hourly information on environment variables as well as seasonal and holiday data. The number of bikes that were rented over the course of each hour is what we will derive the classification target off of.

With the exception of the initial challenges faced with special characters in the column titles and the column titles not aligning with the data, both of which were addressed in Assignment 1, the data was exceptionally clean. Using similar data manipulation techniques from the first assignment, target encoding was used over one hot encoding for the ANN model.

Classification Target

For the classification model, I decided to go with the median value of the rented bikes for my train set. Having a balanced classification would allow me to choose Accuracy as my indication for how well each classification model was performing. Using the median as a target value reduced my concern about making sure that there was a representative distribution in my train and test sets as well, although this can easily be accomplished throughout the code.

Classification Model - ANN

I utilized the Keras library to build out my ANNs. Primary hyperparameters consisted of the following:

- Batch Size: [25,75,150,250,500]
- Epochs: [10,50,200,500,1000]
- First Hidden Layer: [3,5,7,14,21]
- Second Hidden Layer: [1*,3,5,7,14] *1 used to mimic the use of just one hidden layer

I utilized relu for all hidden layer activation functions but did choose to explore the use of tanh for my final activation function. For the compiler, I choose to use the following settings: (optimizer='adam', loss='binary_crossentropy', metrics=['accuracy']).

Other experiments would include the use of a customized grid search model through the use of for loops, the development of a basic ensemble that compared poor learners to high quality learners, and the use of GridSearchCV in combination with an EarlyStopping callback that utilized Tensor Flows functionality. The Early Stopping callback function has clear time saving advantages allowing me to perform the GridsearchCV with 5-fold cross validation while adding just 18% to my total run time when compared to the non CV version.

Train, Validation, Test Sets

I split my data into a 60/30/10 split .

Non-Cross Validation Model With 'sigmoid' Activation Function

My initial efforts began by developing a gridsearch model utilizing for loops. In my code, it is *FunctionFindBestParams*. I quickly realized that adding depth to each for loop could significantly increase the time that it took to run through variations of the model, with low values for batch size and high values for Epochs having the largest effects on time costs. Running through my custom gridsearch with all hyperparameters took a total of 3 hours and 27 minutes. Looking at the accuracy results shows quite a bit of variation through the first 200 combinations, but a consolidation pattern from 300 onward.

Figure 1 Accuracy Results for Non-Cross Validation Model

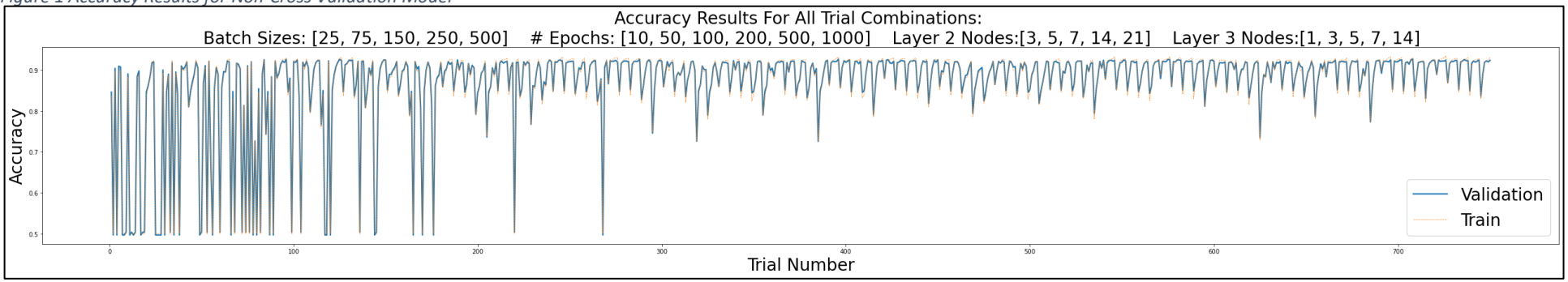


Figure 2 For Loop Design

- Hidden Layer 3
  - Hidden Layer 2
    - Batch Size
      - Epoch Size

With an understanding of how the for loops were running(Figure 2), our Accuracy results(Figure 1) seems to imply that low node values for the first two layers had a higher probability of being a bad estimator. However, once we appear to exceed a threshold around the 275th combination, we can see a far more consistent pattern in which there is some overfitting at our top accuracy training values but also a general pattern of generalization for the minor dips as well(Figure 3).

Figure 3 Zoomed in Accuracy

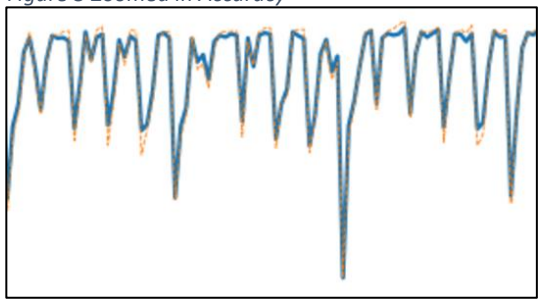


Figure 4 Best Performing Non-CV Model

- Batch Size: [~~25,75,~~150,~~250,500~~]
- Epochs: [~~10,50,200,500~~,1000]
- First Hidden Layer: [~~3,5,7,~~14,~~21~~]
- Second Hidden Layer: [~~1,3,5,7~~,14]

The top performer had a training accuracy of .929 with a validation accuracy dipping to .92656. The parameters that generated this performance are seen to the lower left (Figure 4). However, the exceptional performance came at a cost. As discussed earlier, high value Epochs are one of the contributing factors to increased time costs. At 1000 epochs, it is not surprising to see that the time cost associated with this test was at 28.7 seconds. This was not nearly as bad as the 150 second ranges for trainings of batch sizes of 25 but it raised the question: ***Could we establish nearly as effective models that were more time friendly?*** The answer was yes, and I created multiple visualizations to help see. One such visualization function that I created, *PlotTimePerAccuracyCostGroups*, allowed me to establish a window for accuracy (generally set to .5% but could be adjusted) and time (generally set to 10 seconds but could be adjusted) to generate potential alternatives and comparing them to the top performer. A sample subplot of the visualization can be seen in Figure 5 with our top performer annotated as well as two alternatives. However, the *PlotTimePerAccuracyCostGroups* visualization did not provide the clarity into how the hyperparameters actually effected accuracy or time

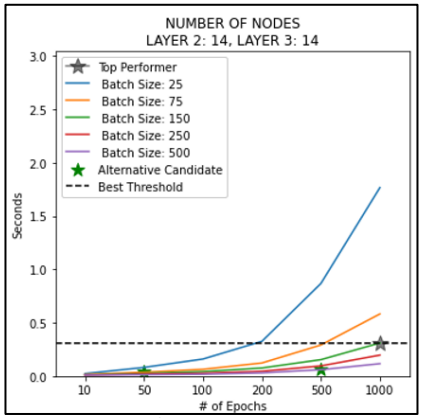


Figure 5 Seconds Per Accuracy Point Non-CV Model

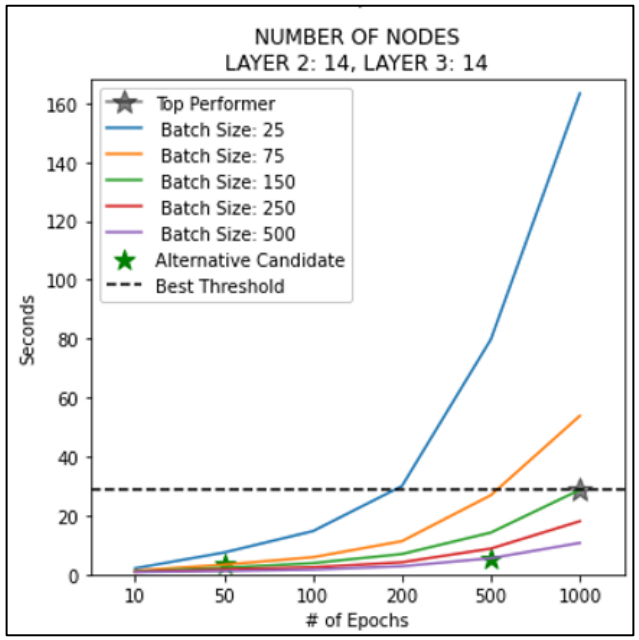


Figure 6 PlotTimeCost Subplot
Non-CV Model

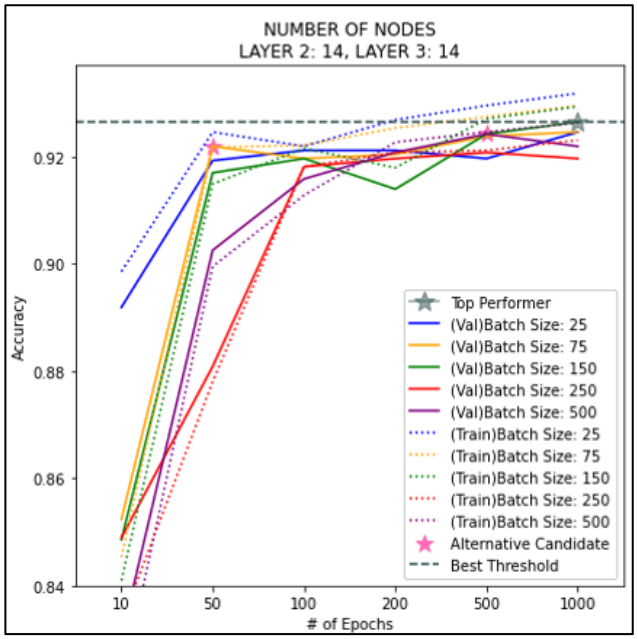


Figure 7 PlotGroups Subplot
Non-CV Model

individually, it merely reflected the ratio. To get a better feel for this, I created *PlotGroups* (Firgure 6) and *PlotTimeCost* (Figure 7) as seen in the subplots to the left. Through the use of subplots which were grouped by Hidden Layer Node features, I found these two visualization approaches to be more effective in describing what was actually going on as the hyperparameters were being manipulated. By utilizing the stars to identify the alternative candidates, I could quickly glance over all of the subplots as in Figure 8 to get a feal for which hyperparameter groupings could lead to the good results for this data set. The *PlotGroups* really did a good job of painting the picture for how much more consistent the train and test results became as you increased epoch size and node size.

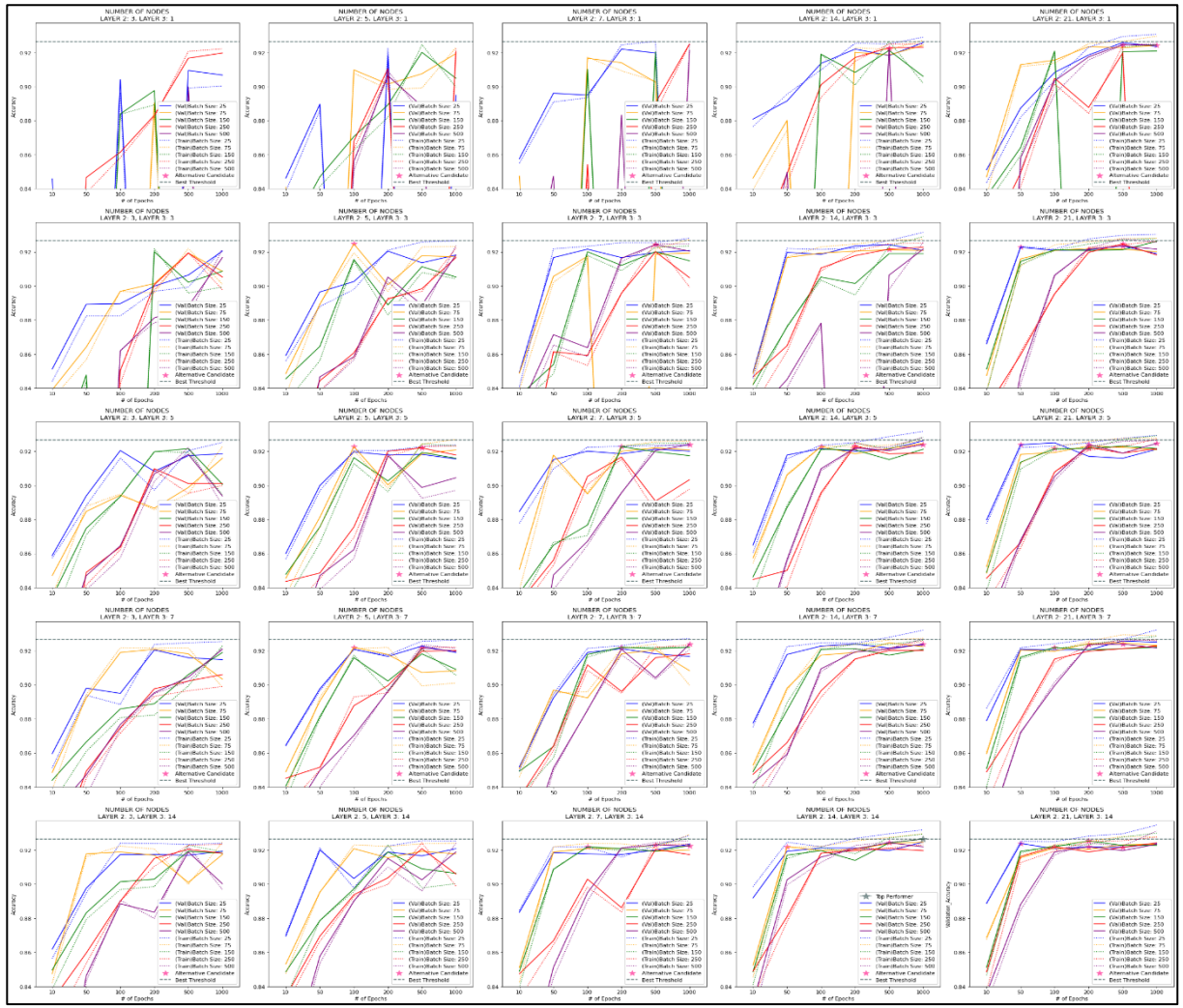


Figure 8 PlotGroups Full Plot
Non-CV Model

Non-Cross Validation Model With 'tanh' Activation Function

Figure 9 Accuracy Results For tanh Activation Function

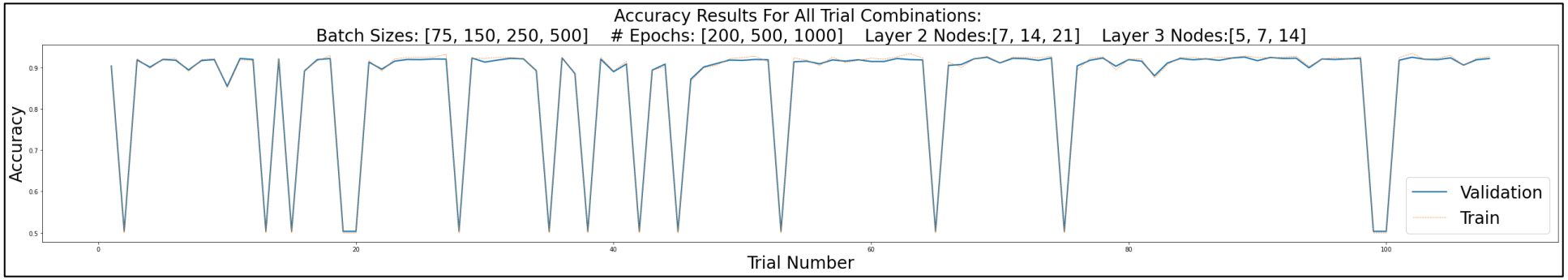
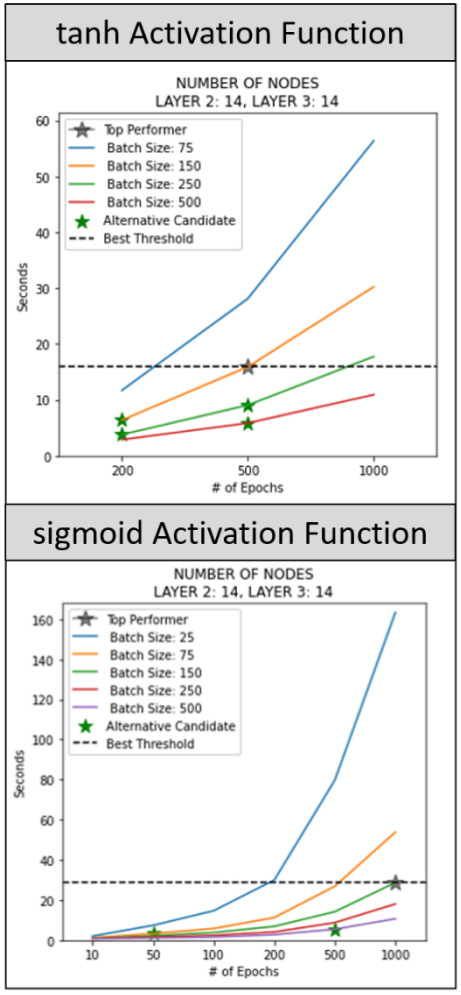


Figure 10 PlotTimeCost Subplot Comparison

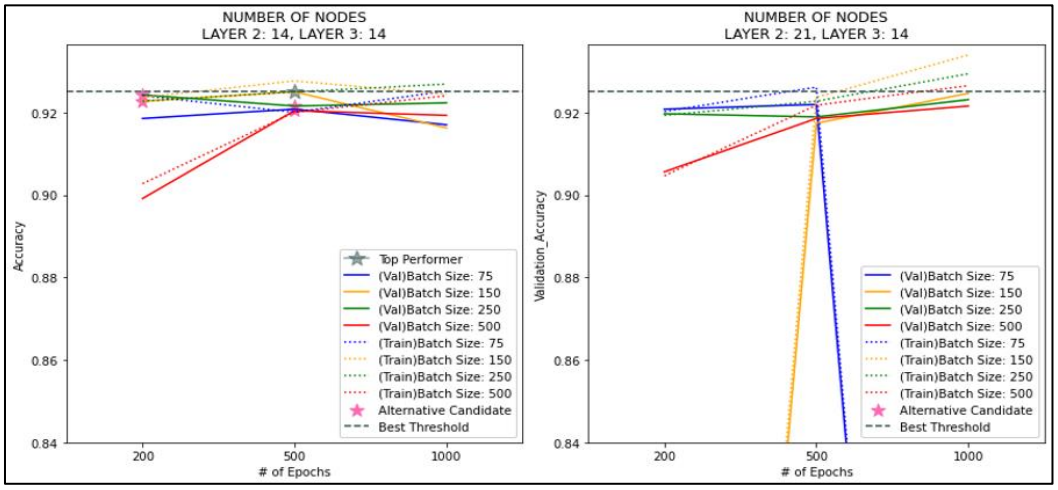


To experiment with the tanh activation function, I focused in on a smaller subset of hyperparameters as I wasn't expecting a significant change in speed and really wanted to just see if there was a change in accuracy. The accuracy results in Figure 9 surprised me. There was significant variation in accuracy even with larger hidden layer node sizes. Figure 11 highlights how surprising it was to see a steep drop-off despite increasing node size.

My suspicions that the time costs were similar were mostly confirmed. Figure 10 shows that they were pretty consistent between the two functions (note that the colors do not correspond from one plot to the other, as batch size of 25 was not included in the tanh experiment). When looking across the 9 different subplots for the experiment, the general trend seen in Figure 10 holds.

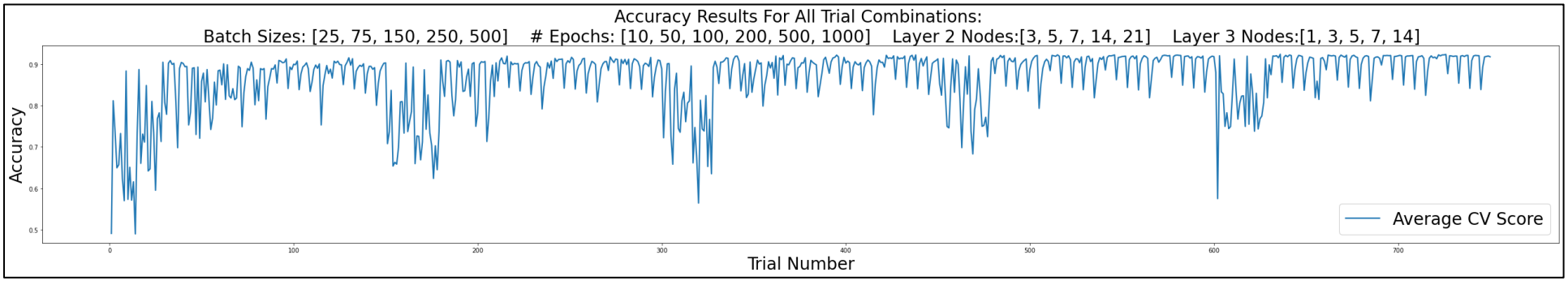
I did find it interesting that both experiments produced the best results with the same hyperparameter. I suspect that this is because the final activation function was the only thing that changed. It would be interesting to see if we would find an alternative grouping had the 'relu' activation function been changed to another function.

Figure 11 PlotGroups Subplot for tanh



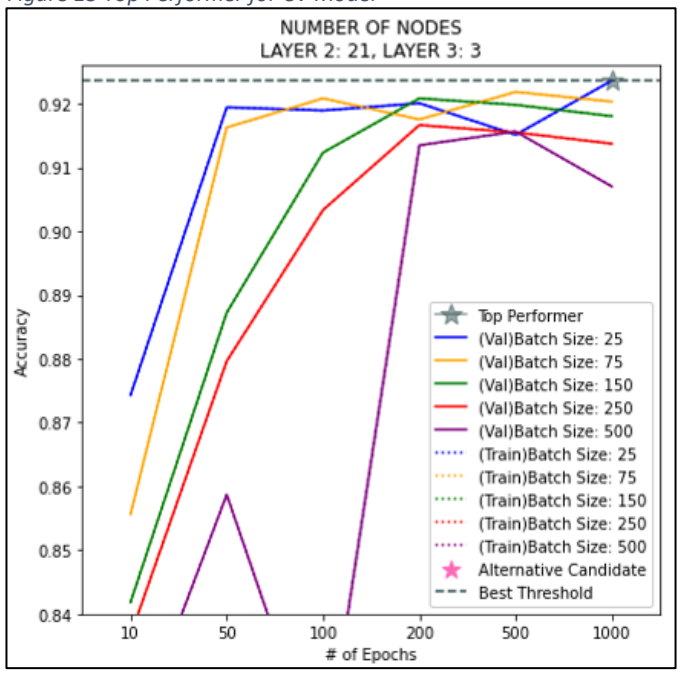
5 Fold Cross Validation Model With 'sigmoid' Activation Function

Figure 12 Accuracy Results For CV Model



For the Cross Validation model, I returned to the full spectrum of hyperparameters while combining my train and validation sets. When I first started running the cross-fold validation it was taking an extremely long time and I quickly realized that I needed to explore what else the Keras package had to offer. One of the features that I found came int the form of a Callback in which I was able to employ *EarlyStopping*, offering a variety of ways to create early stopping during the training phase of each fold. I specifically chose to monitor "loss" with a minimum delta of "1e-2" and a patience of "20". Although I never ran the full GridsearchCV without the EarlyStopping feature, there were clear advantages to using it as I was getting stoppage as early as 21 epochs. This could be evidence that I need to consider reducing my loss value or that there were some combinations that just didn't require much effort to come to a settling point. It would be interesting to further play with these and the other parameters offered through *EarlyStopping* to see how much the results changed but this was not something that I explored.

Figure 13 Top Performer for CV Model



I was once again surprised to see a dip in the far right section of the model, but after the initial buildup, none of the dips were as low as the 50% range that we were seeing in our original model. This shows the effect of each ANN iteration being different and how the Cross Validation provides a better picture as to how the hyperparameter combinations perform on average.

One thing that would potentially need to change with employment of CV is how I measured the potential alternatives. Using the established thresholds of .5% and 10 seconds yielded only a single alternative. Increasing the threshold to 20 seconds yielded 36 alternatives. This is not too surprising given that we are now functioning off of a 5-fold cross validation, but I was hoping that having the EarlyStopping in play would provide more alternatives at the original thresholds. A quick glance through the output of the cross validation doesn't indicate that the early stopping was taking place at one end of the spectrum or the other as I see several early stops in the 92% range where the epoch count is above 100 and several in the 48% range where the epoch count is 21. Considering that our top performer was functioning at a batch size of 25 and the increased time associated with this batch size stretched its time cost all the way to 49 seconds, it shows that your thresholds may need to change depending on what approach you are taking.

Comparing the Different ANN Model Outcomes

With a validation accuracy of 92.66%, the non-CV version of the ANN model was the top performer. The tanh and CV approaches were not far off with validation accuracies of 92.5% and 92.3%. Given that the scores are so close and the fact that the Cross Validation model has proven to generalize well over its 5 folds, this is the model that I would go with as it should continue to generalize well on new data. This would lead to a final test accuracy of 91.55% on our held out test data(Figure 14). It is important to once again point out that these numbers can change from one iteration to the next.

Figure 14 Test Accuracy for CV Model

```
TEST DATA RESULTS :  Sigmoid ANN CV=5
                     Epoch: 500
                     Batch: 25
                     HiddenLayer2: 21
                     HiddenLayer3: 3
****************************************
          Accuracy :  0.9155251141552512
Balanced accuracy :  0.9149080474761967
         Precision :  0.9036402569593148
            Recall :  0.9356984478935698
               AUC :  0.9629972609886526
       Sensitivity :  0.9356984478935698
       Specificity :  0.8941176470588236
```

Comparing the Previous Classification Models from Assignment 2

The ANN model appears to give similar results to what we were getting from our Decision Tree and SVM models in the previous assignment with a range of test results falling between 91% and 92.8% for our top performing models. My preference would still be for the CrossValidation Decision Tree model built off of Cost Complexity Pruning with a test result of 92.5%. This decision tree should generalize well and provide the benefit of being tangible and easy to interpret. However, as discussed below, I would like to see the results of an ensemble comprised of all three approaches to see if any further improvement could be captured.

Exploring Other Outcomes

One thing that I was able to experiment with was an Ensemble approach. One of the reasons that I identified Alternatives was with an intention to see if an ensemble of alternatives could provide performance improvement. My initial attempt to use all of the high performance, time friendly alternatives didn't provide the improvement I was hoping for with a final test accuracy of 91.3%(Figure 15). Thinking that perhaps it didn't improve because the models were having difficulty with similar observations, I reran the ensemble with "poor performers" – taking hyperparameter combinations that provided validation accuracy levels between 60% and 80%. This approach proved to be less accurate than the previous though with a final test accuracy of just 81%(Figure 16).

Another approach could be to create an ensemble that stretches across multiple classifiers. For instance, we could utilize our Decision Tree and SVM models along with the GridsearchCV ANN model developed in this assignment with the hope that the models vary enough on their misclassified observations to create a stronger pooled outcome. However, I did not explore this approach.

Figure 15 Confusion Matrix for Alternatives

```
Train Ensemble Accuracy: 0.923896499238965
Train Confuision Matrix:
 [[2419  207]
 [ 193 2437]]
Validation Ensemble Accuracy: 0.921613394216134
Validation Confuision Matrix:
 [[1223  100]
 [ 106 1199]]
Test Ensemble Accuracy: 0.91324200913242
Test Confuision Matrix:
 [[386  39]
 [ 37 414]]
```

Figure 16 Confusion Matrix for Poor Ensemble

```
Train Ensemble Accuracy: 0.821917808219178
Train Confuision Matrix:
 [[2096  530]
 [ 406 2224]]
Validation Ensemble Accuracy: 0.8420852359208524
Validation Confuision Matrix:
 [[1106  217]
 [ 198 1107]]
Test Ensemble Accuracy: 0.8105022831050228
Test Confuision Matrix:
 [[339  86]
 [ 80 371]]
```