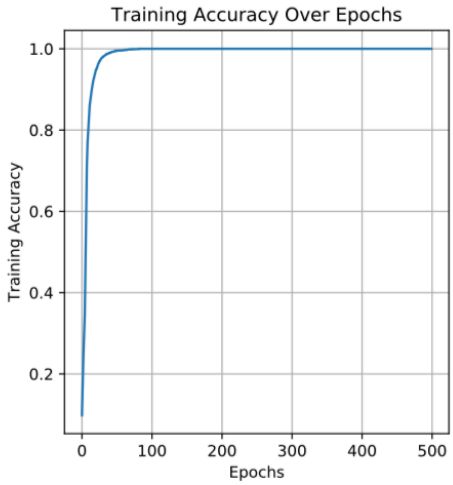
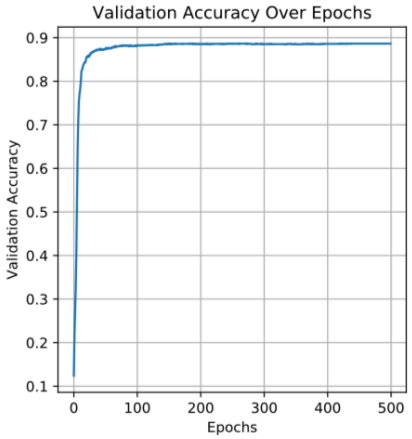
Andrew Robbertz alrobbertz@wpi.edu
CS 4341 A18
09/08/2018

MINST Data Classification Using Artificial Neural Networks

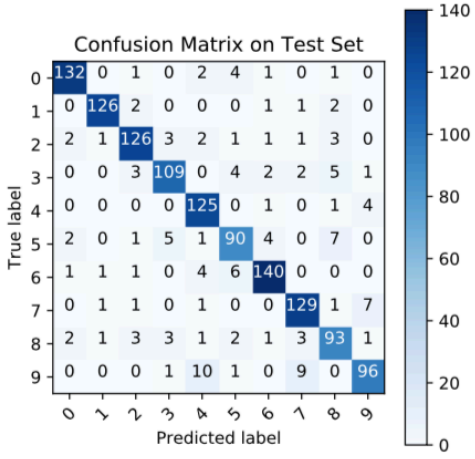
Artificial Neural Network for Classifying Data Without Dropout

Splitting the dataset with 80% of the data in the training set and saving 20% in the test set, the following results were found training over 500 epochs and a batch size of 512.





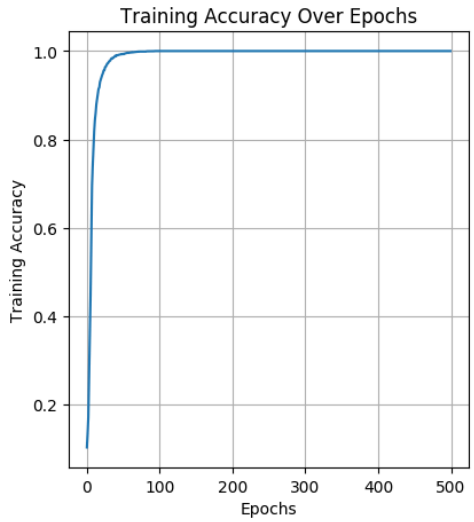
Test Accuracy reaches 100% after the first 100 epochs, while validation accuracy plateaus at just under 90%. This suggests that we are not learning anything new after the first 100 epochs, and all after are not completely necessary.

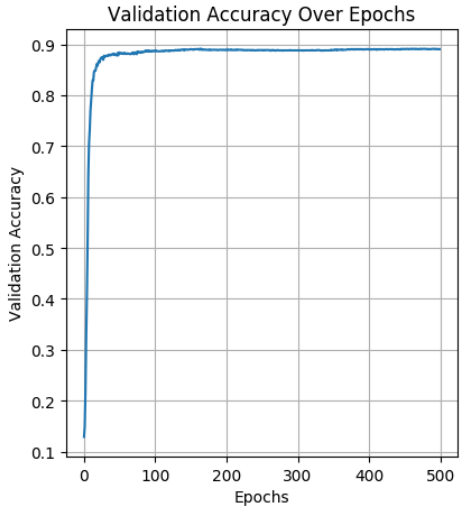


The test set reported 88.3% accuracy, which is sufficiently close to the validation accuracy after it plateaus. The difference may be caused partially by overfitting, as well as any bias in the dataset that emphasizes certain numbers more than others.

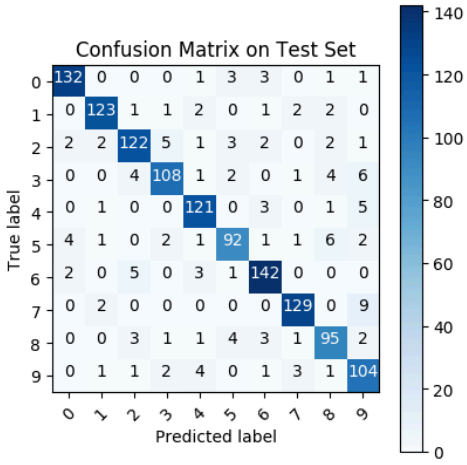
Artificial Neural Network for Classifying Data with Dropout

Splitting the dataset with 80% of the data in the training set and saving 20% in the test set, the following results were found training over 500 epochs and a batch size of 512.





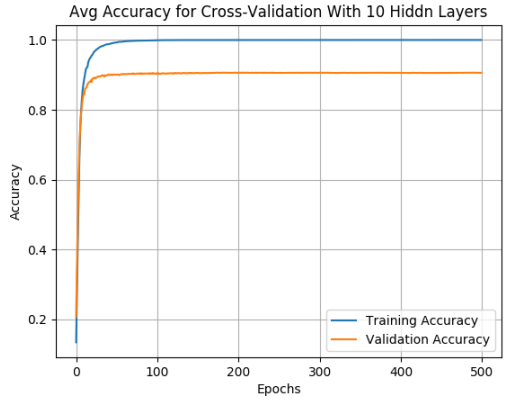
With 20% dropout on the input later, test accuracy still reaches 100% after the first 100 epochs, and validation accuracy still plateaus at just under 90% although it does get slightly closer. Still we are not learning much after the first 100 epochs, and could run significantly fewer epochs.

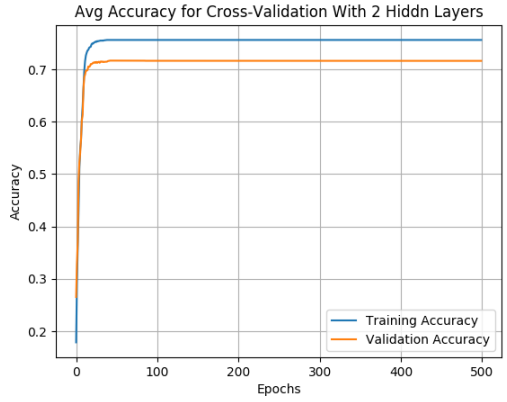


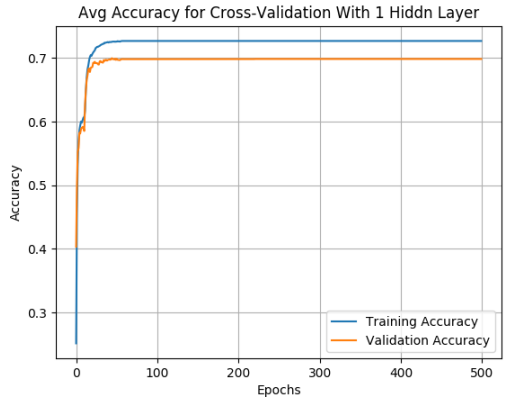
The test set reported 89.4% accuracy, which is slightly better than our test accuracy without dropout. This suggests that the network was likely overfitting data in the training and validation sets to a small degree. The difference is so small though that it may be coincidental. The confusion matrix is noticeably better with dropout than without, also suggesting that we may have been overfitting.

Evaluating ANN Performance with Varying Hidden Layers

I first evaluated the network using 3-fold cross-validation on a network with 10 hidden layers, and then repeated for 2 and 1 hidden layers. As a baseline I tested each number of layers with 500 epochs and batch size of 512, same as above. The following results were found tracking training and validation accuracy over each fold.



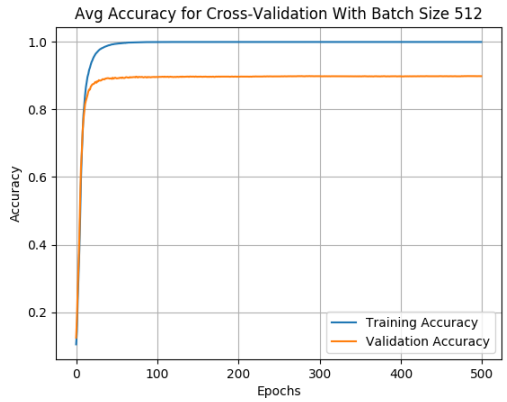


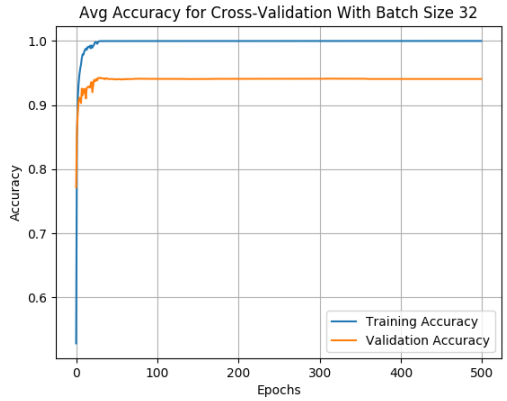


As seen in the data, the model with 10 hidden layers clearly outperforms the other two, with the average accuracy being about 20% higher in both cases. Note the differences in scale for each of the figures. The model with 2 hidden layers still slightly outperforms the single layer model as expected. The 10-layer model's success can simply be attributed to the fact that it has exponentially more connections and weights than the other two to parameterize the data.

Evaluating ANN Performance with Varying Batch Sizes

I evaluated the network using the 10-layer model and 3-fold cross-validation as shown in the previous test with 500 epochs and varying the batch size between 32 and 512.

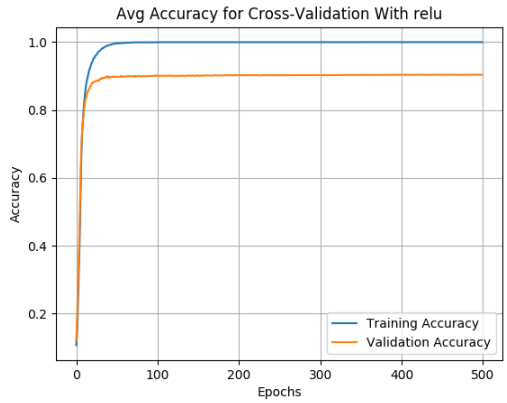


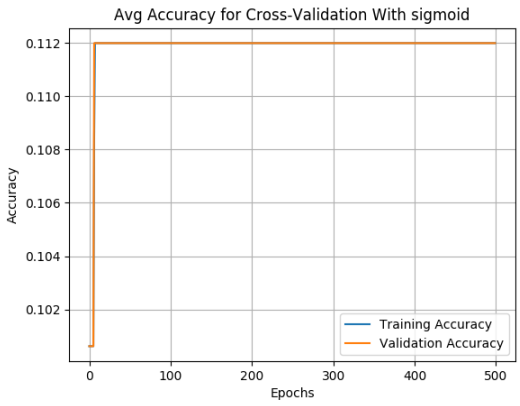


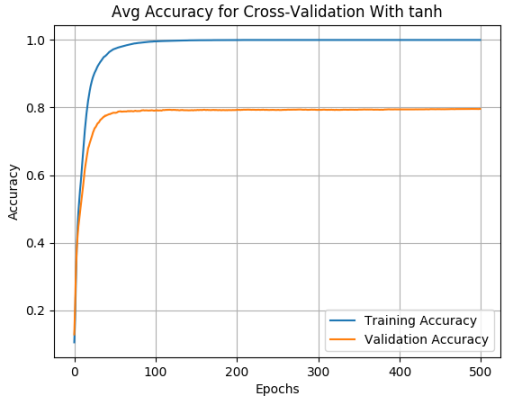
The test with batch size 512 is essentially the same as the first figure for varying layer depth as expected. However, the test for batch size 32 yielded slightly higher accuracy across the validation sets. This is not surprising, as smaller batch sizes do help to fine-tune the network better than larger batches. This can also be seen as both training and validation accuracies plateau earlier with a batch size of 32 than with 512. The main downside of running smaller batches is that computation time increases significantly because more batches are needed per each epoch. The performance gains of about 5% across epochs is non-trivial, however it may not be worthwhile to decrease batch size on sufficiently large datasets where computation time will be impacted most.

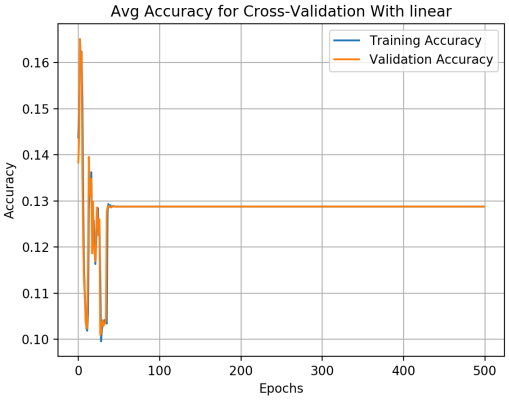
Evaluating ANN Performance with Varying Activation Functions

All previous tests were done using the relu activation function in the input and hidden layers. It was not clear to me how slightly varying activation functions can have such a large impact on ANN performance, therefore I did one final experiment to see how the accuracy might vary. I tested the same model with 10 hidden layers, 500 epochs, and batch size of 512 using the relu, tanh, sigmoid, and linear activation functions built into Keras to see how accuracy might change. I tested all three using 3-fold cross-validation, same as in the previous two tests.









Not surprisingly, relu has the best accuracy, many thanks to Professor Heffernan for recommending we start with relu. The tanh activation function seems to work relatively well, although not as well as relu. Both the sigmoid and linear activation functions have atrocious accuracy, and I'm not sure whether this is due to the activation function itself, or another part of the model that is disturbed by using a different function. Something that is especially curious is the initial jumps in accuracy for the linear activation function. This seems like it would be characteristic of the linear activation function itself, as it's derivative is always 1 so the weights would jump significantly.