Trevor Klinkenberg

CS 461 Intro to A.I.

Program 4 Project report

For my A.I. I wrote a simple tensor flow system with the addition of numpy and pandas for clearer output reports. I started out by manually splitting the data into three sets of data with a 70/15/15 split; the training data, the test data and the validation data. This ensures that I was never including my validation or test data into my training set, giving a more accurate representation of the A.I.’s performance.

The training data is first converted from CSV to a tensor to be fed into the A.I., this is further segregated into batches of 64, for faster and more accurate computation. This data is fed into the model and trained through the batched data and a set of epochs. Once the A.I. has finished training, the test data is loaded and converted from CSV to a tensor in the same method as the training data. The A.I. is then evaluated against this test data and reports its test loss and accuracy against this data it has not seen. Finally, the A.I. is evaluated using the validation set using the same conversions as the training and test set. This set was not introduced until I had settled on the model I observed had the best performance.

For my first model I created a network of 2 hidden layers of 10 nodes each with ReLU conversion and a final output layer with a sigmoid output. This model used the adam optimizer and binary crossentropy for computing the loss and back propagation. The model was then trained with 15 epochs. This model produced a final training accuracy of 97% and a test accuracy of 98%.

The first super parameter I decided to change was the optimizer function. The SGD produced a 96% training and 98% test results, The Nadam converged midway to 97% in the training epochs and produced a 98% test result. The RMSprop algorithm converged as quick as the Nadam but only produced 96% training result. I decided to go with the Nadam optimizer since this seemed to have produced the highest output with the fewest epochs.

Next I decided to adjust the formulas on my hidden layers. I changed the 2 hidden layers to a soft max function, this produced a lower training result of 96% and a test result of 97%. I changed these layers to linear and the model converged to 97% after only 5 epochs, this did still produce a 98% in the training set however. Lastly I chose to try the tanh function. This model converged after 4 epochs to 96% training output and 97% with the test output. I settled on using a combination of relu softmax and sigmoid to have the model converge later and still produce a 98% in the testing data.

My next change was on the amount of epochs and nodes in my model. I tripled the nodes in my hidden layers to 30 and doubled my epochs to 30. This produced an extremely early convergence to roughly 97% after the first 4 epochs and still produced a 98% in the test data. I reduced the epochs to 10, set the softmax layer to 30, and the relu layer to 15. This produced similar results of 97% on the training data and 97.7% on the test data. I swapped the node amounts of the layers so that the softmax had 15 and the relu had 30, this produced a later convergence to 96.7% on the training and 98.2% on the test set. Ultimately I settled on a model with 15 softmax nodes, 25 relu nodes and a final layer of 1 sigmoid node, trained on 20 epochs. This produced a later convergence of 96.8% training and 98.3% test results.

Finally I combined all this together and added my validation data. The superparameters and model I settled on had the following: 3 hidden layers with 15 softmax nodes, 25 relu nodes and 1 sigmoid output node. The model was trained on 20 epochs with a Nadam optimizer and binary\_crossentropy function for calculating loss and backpropagation. This A.I. produced a training result of 96.9%, a test result of 98.3% and a validation result of 98.9% accuracy. I believe this is the highest performing network I could create using the smallest set of resources.

This project was a great introduction to using the common set of tools for training A.I. for real world applications. Using tensor flow was extremely easy to pick up and create an A.I., the documentation had great tutorials and resources on how to create an A.I. that was highly functional. My main suggestion for this project would maybe be to change the data that we are training our A.I. with. Most alterations to the model did not seem to have a significant impact in the overall results of the A.I. I do not know if that this is because the data was “easy” for the A.I. to learn off of or if there just was not enough data to make a significant impact. I believe the rule of thumb is to create and A.I. that is greater then 80% for acceptable losses and greater then 90% for the highest accuracy models. With that being said most of my changes still resulted in an A.I. tat produced higher then 90% results in all cases. It would be nice to see a training set where there was a larger variance depending on the algorithms and models used. Additionally, I think it would be a much more fun project if we could have trained our models using image recognition. The MNIST data is a great easy start that I believe anyone could pick up and you can show sample identifier results with this data. In addition, the MNIST data has more categories to separate the data into instead of the binary “pulsar/not pulsar” that this project entailed.

Overall, I had an enjoyable time creating this A.I. and can see myself using this knowledge for future use to create my own personal A.I. for other purposes.