Hackathon2

Mason Lien

The objective of this hackathon was to investigate how changing architecture of a the fully connected layer, activation functions, or even the optimizer yields differing performance of prediction accuracies. The dataset we used was the MNIST, where the training dataset was split into 90% (1688) and validation 10% (188). The batch size was 32 kept at 32 to understand how changing number of neurons and adding additional layers affected the model performance. The final Dense layer contained 10 neurons associated to the 10 numbers. Each of the 10 neurons is associated to a number 0-9, the neuron that has the highest value is ultimately the neuron that gives the predicted number.

Model 1 used sequential with one dense layer at 100 neurons with activation relu, that fed into the last dense layer of 10 neurons to give final prediction. The optimizer was adam and the epochs was set to 5. The training loss dropped significantly after first iteration. The training and validation accuracy almost identical at 0.901. A good start to accuracy but can do better with a few alterations to the model.

Model2 used sequential with two dense layers at 128 neurons with activation relu, that fed into the last dense layer of 10 neurons. The optimizer was adam and epochs was set to 5. Adding an additional dense layer improved the training and validation accuracy to 92% and 93% respectively. This increases the number of parameters in the model, which we must be careful of due to over and underfit issues. In this model the extra layer adds depth and complexity. If the model were to be overfit it would perform very well on the training data but poorly on the test data because it basically memorized the training data, and the goal of a robust model is to generalize well on data that it has not seen prior.

Model3
Same parameters as model except I changed the activation function to sigmoid instead of relu. Relu is more poplar than sigmoid due to its simplicity, and it is fast to use in training. One problem with sigmoid is that the gradient goes to zero if the input is either very large or very small, which when this goes to zero, gradient descent has slow convergence. The advantage of relu over sigmoid is the derivative of relu is faster to compute, which makes a significant difference to training and inference time for convnets. This is due to if the input is negative it goes to zero, where positive inputs are passed. Overall the training accuracy was around 89% where validation was 90%.