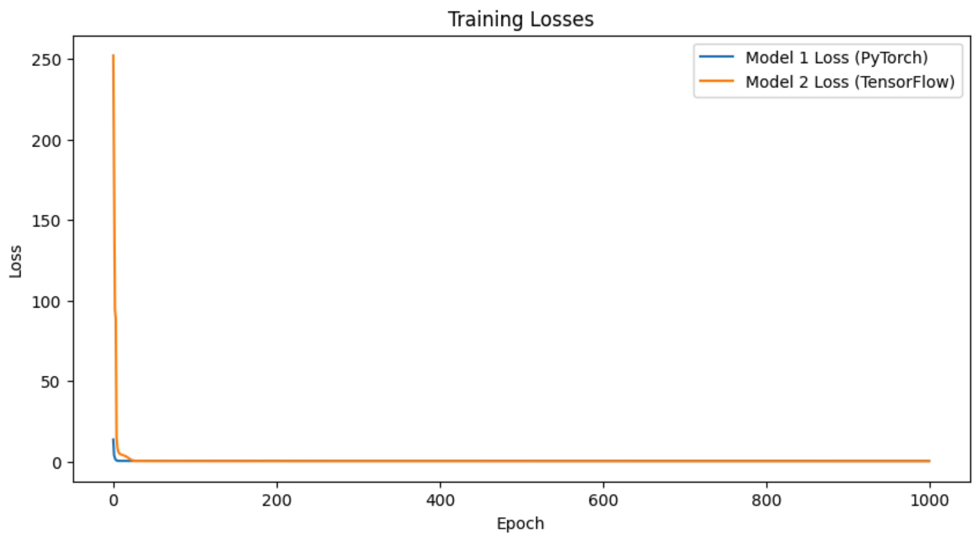
HW1 Report

John Pascoe

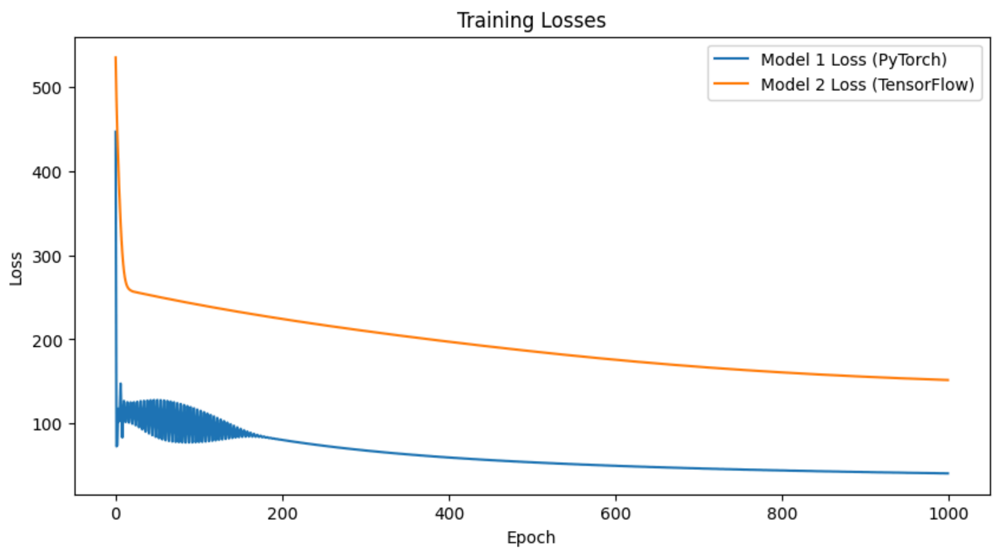
Github: CPSC-8430-Deep-Learning/HW1 at main · jpascoe32fb/CPSC-8430-Deep-Learning (github.com)

1-1 Questions

Simulating a Function

1) The first model I used was a simple DNN with one hidden layer and a total of 31 (1 input + 10 outputs + 10 biases) parameters. The second model was also a simple DNN with directly implemented weights and biases (1*10 + 10 for the first layer and 10*1 + 1 for the second layer) and a custom mean squared error calculation. I used two different functions for analysis: the first being y = sin(x) + 0.5x, and the second being y = cos(x) + 0.5x^2.
2) Training Loss
    a) *Function y = sin(x) + 0.5x*
        i) 

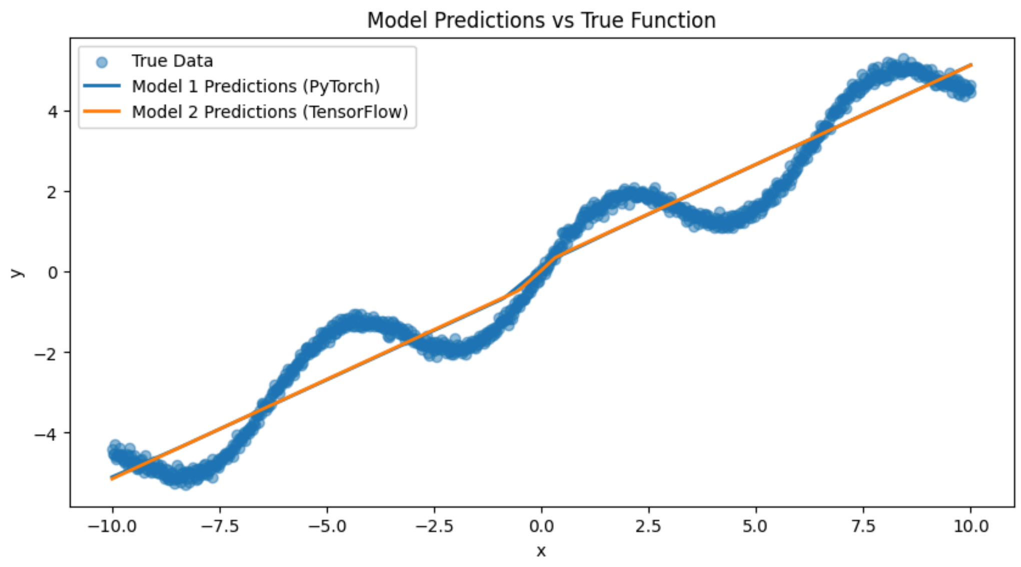
    b) *Function y = cos(x) + 0.5x^2*



i)

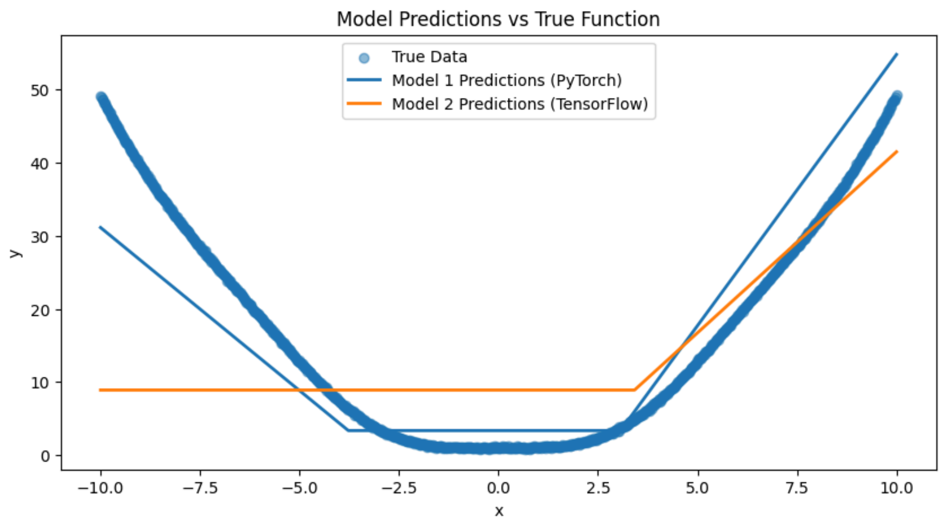
3) Training Accuracy

a) *Function y = sin(x) + 0.5x*



i)

b) *Function y = cos(x) + 0.5x^2*

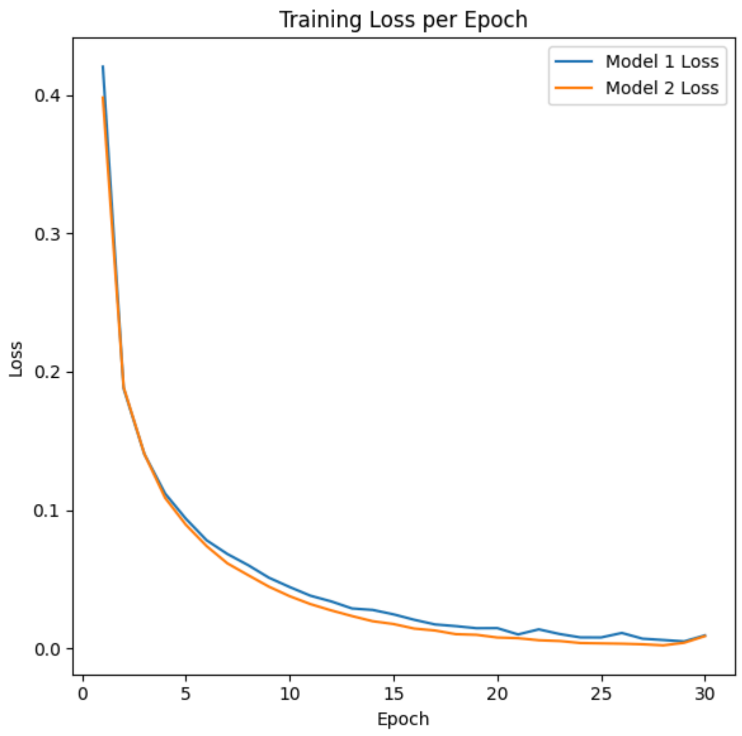


i)

4) Comment on Results
    a) The training loss graphs all indicate that the models very quickly minimized the loss function, with the loss dropping sharply within the first few epochs and flattening out. This indicates convergence for the models loss and they all converge to a low value which suggests good fitting to the training data.
    b) The scatter plot of the true data shows the underlying pattern that both models aimed to learn. Both the models appear to predict the function accurately which is indicated by the lines fitting well to the distribution of true data points. There is only a significant difference in accuracy shown when the function y = cos(x) + 0.5x^2 was used to train the models.

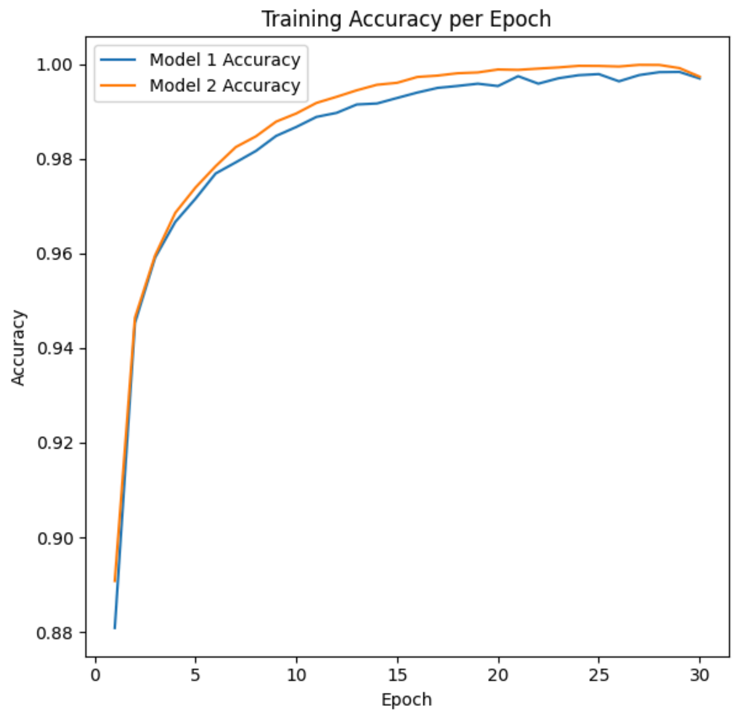
Train on Actual Tasks

1) Model 1 is a DNN model with three layers. This mode has more layers, but fewer neurons per layer compared to model 2. It contains a flattening layer, two dense ReLU layers, and a final dense layer using softmax activation. Model 2 is also a DNN model but it has fewer layers with more neurons per layer than model 1. It contains a flattening layer, one dense ReLU layer, and a final dense softmax activation layer. The task I chose was the MNIST dataset.
2) Training Loss



a)

3) Training Accuracy



a)

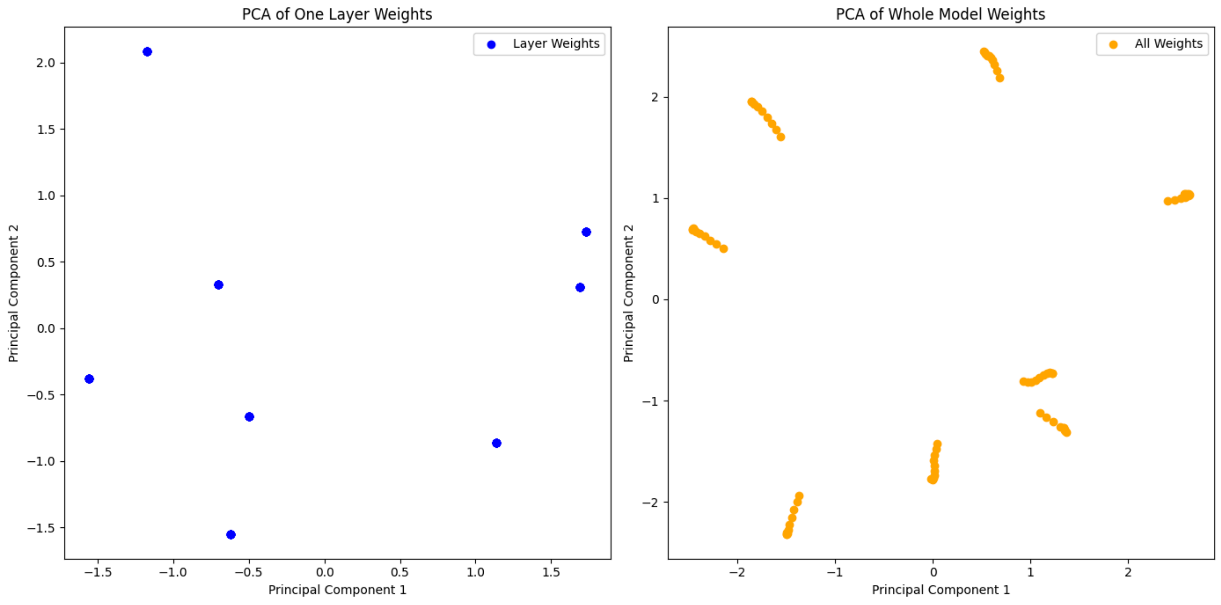
4) Comment on Results

a) Both models show a decreasing trend in training loss as the number of epochs increases, which indicates that both models learned from the training data over time. Model 2, with fewer layers and more neurons per layer, starts with a slightly higher loss but converges to a similar value as model 1 by the end. The accuracy graph shows an increasing trend for both models, which indicates an improvement in the models' performances on the training set as they learned. Model 2 rapidly reached a high accuracy and stayed very close to model 1 throughout the process.

1-2 Questions

Visualize the Optimization Process

1) The model consists of three fully connected layers. ReLU activation functions are used in the hidden layers. The input is a tensor of 200 points linearly spaced between -10 and 10. Each input is transformed by a sine function with an additional 0.5 times 'x' noise added. The model is trained using the Adam optimizer with a learning rate of 0.01 and a loss function of mean squared error. The training ran for 100 epochs. Model parameters were recorded every 3 epochs during training. Two sets of weights were collected: the weights from the first layer and all the weights from the entire model. Principal component analysis was applied to reduce the dimensionality of the collected weights to two principal components. The training and weight collection process was repeated 8 times, which resulted in multiple sets of weights used in the PCA.
2) Figures



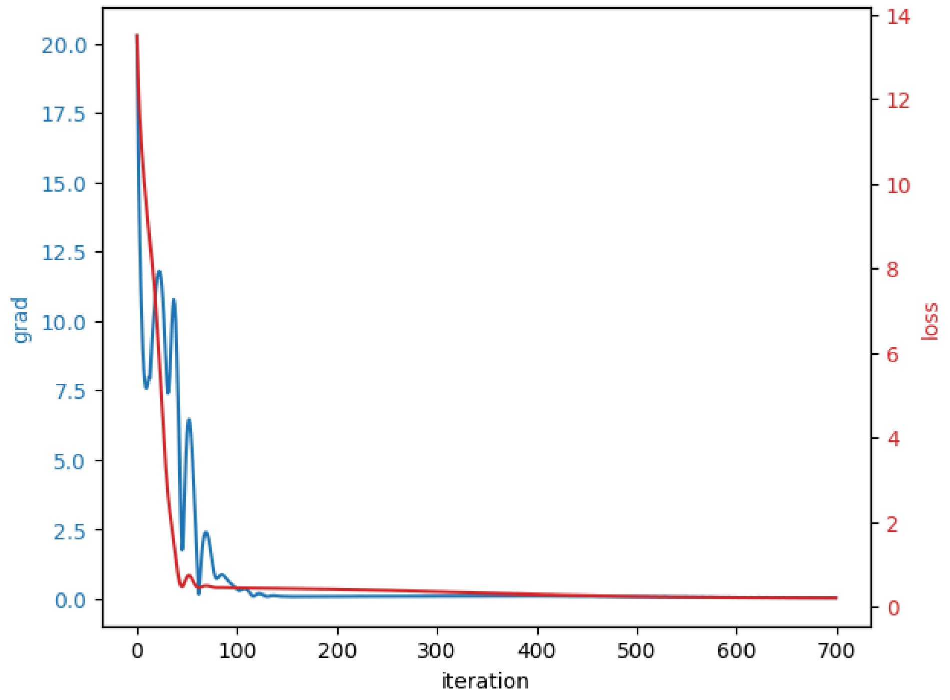
a)

3) The PCA of the first layer's weights shows a distribution of points that seem to cluster along the diagonal axis from bottom left to top right. This suggests that there is some variance in the weights in the first layer, but the variation is predominantly along one axis in the higher dimensional space, which PCA has captured. The PCA of the entire model's weights shows several clusters along the horizontal axis. These clusters could indicate different sets of solutions that the model's weights have converged to during different

training cycles. The multiple clusters suggest a greater diversity in how the model's weights are distributed compared to the individual layer. Overall, the PCA visualization indicates that there's some variance in the weights of the first layer, but the entire model's weights exhibit a more complex structure with distinct groupings.

Observe Gradient Norm During Training

1) Figure



   a)

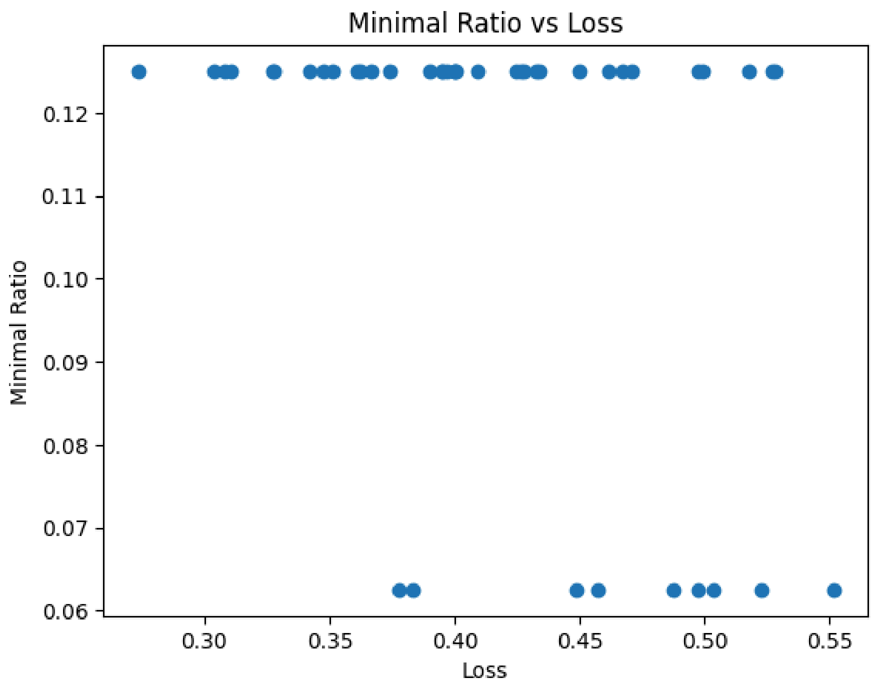
2) Comment on Results
   a) It appears that the training process was effective. The model's loss decreases over the epochs to a low value, which indicates good performance on the training data, and the gradient norm decreases and stabilizes, suggesting that the model is converging to a minimum of the loss function. The plot does not show how the model fits over the data, so further testing is necessary to determine that.

What Happens When Gradient is Almost Zero

1) The small network I utilized for this experiment was trained to fit the function y = sin(x) + 0.5x and then analyzed the situations where the gradient norm of the model's weight was close to zero. During the training process, after each update of the weights through backpropagation, the code calculated the gradient norm of all the model's parameters. This was done by summing the norms of the gradients of each parameter. The gradient norm is a measure of how much the weights are changing. The code then checked

whether this gradient norm was below a certain threshold, which was a small positive value which represents the almost zero condition. If the gradient norm was less than the threshold, it implied that the model's weights are near a local minimum. The minimal ratio was used to see how many weights are close to zero. My code computed this by first creating a concatenated tensor of all the weights in the model. It then checked each weight to see if its absolute value is below a certain weight threshold. The minimal ratio was then calculated as the number of weights below this threshold divided by the total number of weights.

2) Figure



a)

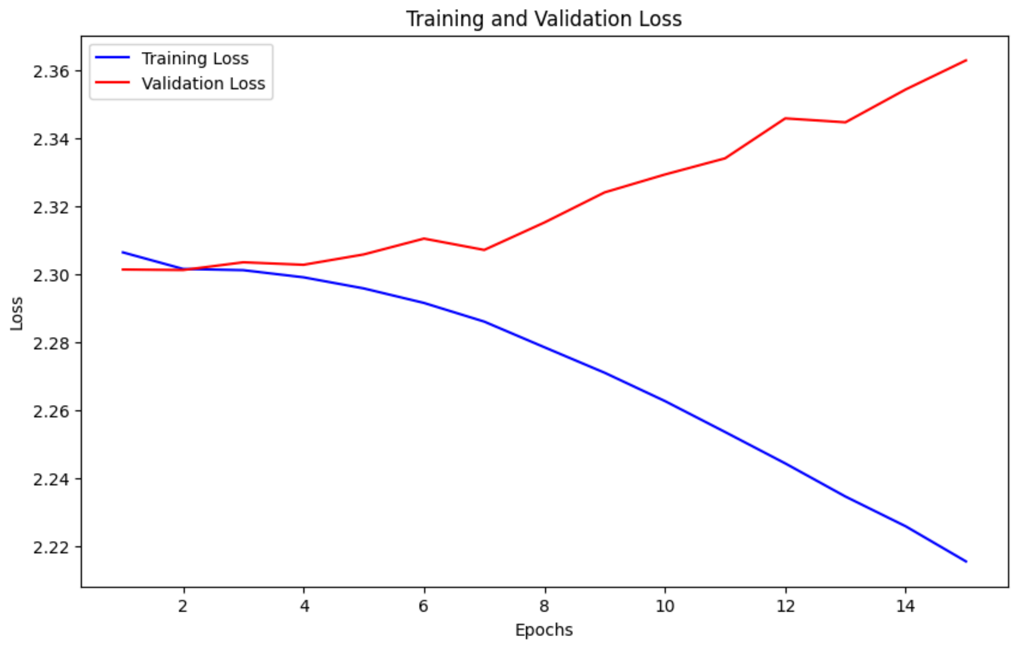
3) Comment on Results

   a) Most of the data points for loss are concentrated in a narrow band of loss values, roughly between 0.3 and 0.55. This suggests that when the gradient norm is small, the loss of the model doesn't vary wildly, which means the training process is somewhat stable. The minimal ratio values are clustered between approximately 0.06 and 0.12. There doesn't seem to be a clear trend of minimal ratio, but the values are relatively low, which suggests that a small proportion of weights are near zero. This means that a small proportion of weights might be contributing to the model reaching a point of near zero gradients.

1-3 Questions

<u>Can Network Fit Random Labels</u>

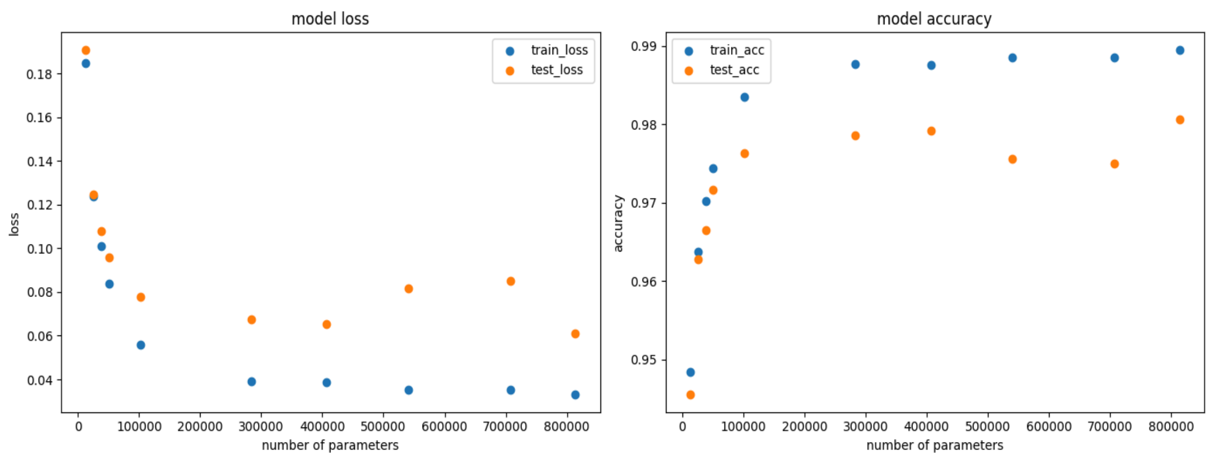
1) The task for this experiment was to train the network model to classify images from the MNIST dataset where the labels are shuffled randomly, which means the model's task is to actually fit to a randomly assigned label rather than learning the true digit represented in the images. This introduced noise into the training and validation process since the labels do not correspond to the actual images. The learning rate is the default learning rate for the Adam optimizer which is 0.001. The Adam optimizer is an adaptive learning rate optimization algorithm designed specifically for training deep neural networks. The model architecture is made up of a flattening layer, a dense ReLU layer, and a dense softmax activation layer.
2) Figure



a)

Number of Parameters vs Generalization

1) In this experiment, the models I used were significantly similar, with the main difference being the size of a single hidden layer. The number of neurons in this hidden layer varied across different models. The models were trained and tasked with the MNIST dataset. The performance of each model was measured in terms of loss and accuracy on both the training and validation data.
2) Figures

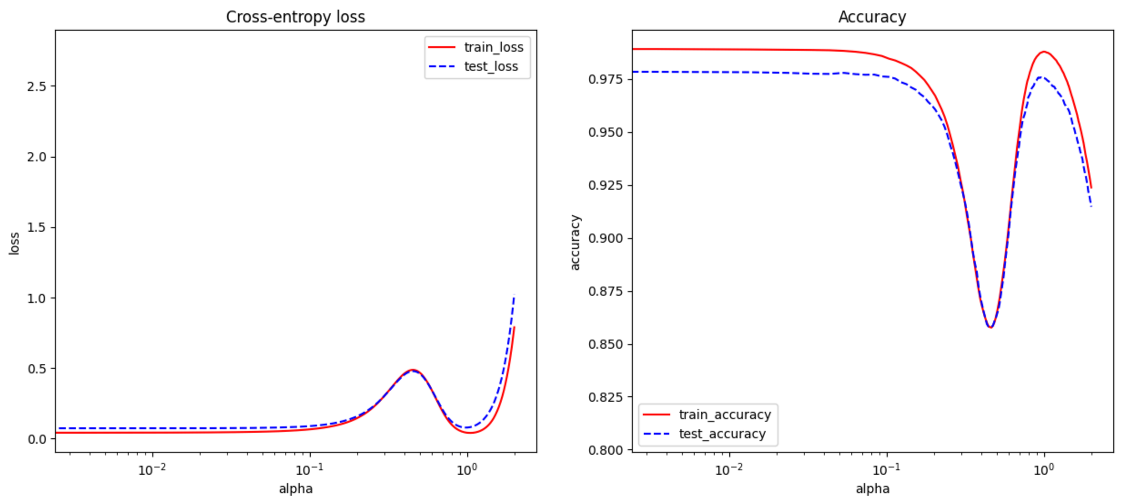


a)

3) Comment on Results
   a) From the graphs, it can be inferred that models with a high number of parameters tend to have better training performance but do not necessarily lead to better generalization. This could be because larger models are more prone to overfitting, which leads to decreased performance on new data.

<u>Flatness vs Generalization</u>

1) Part 1
   a) This experiment was based on two models trained on the MNIST dataset. The models were similar but had different training approaches. The first model was trained with a batch size of 64 and a learning rate of 1e-3. The second model was trained with a batch size of 1024 and a learning rate of 1e-2. Each model was trained for 5 epochs. The Adam optimizer was used for training efficiency.
   b) Figures

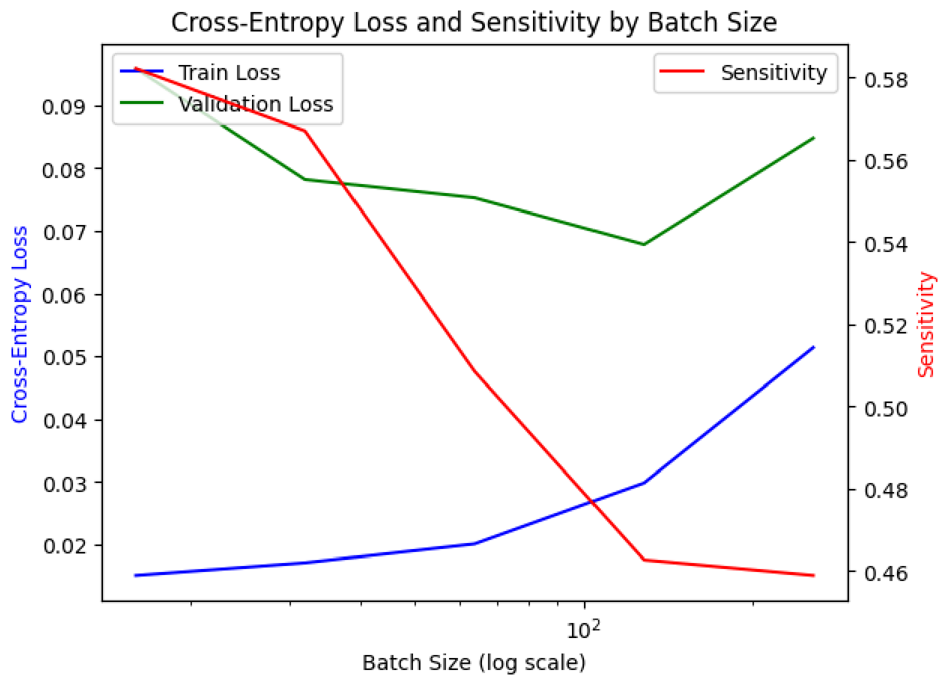


      i)

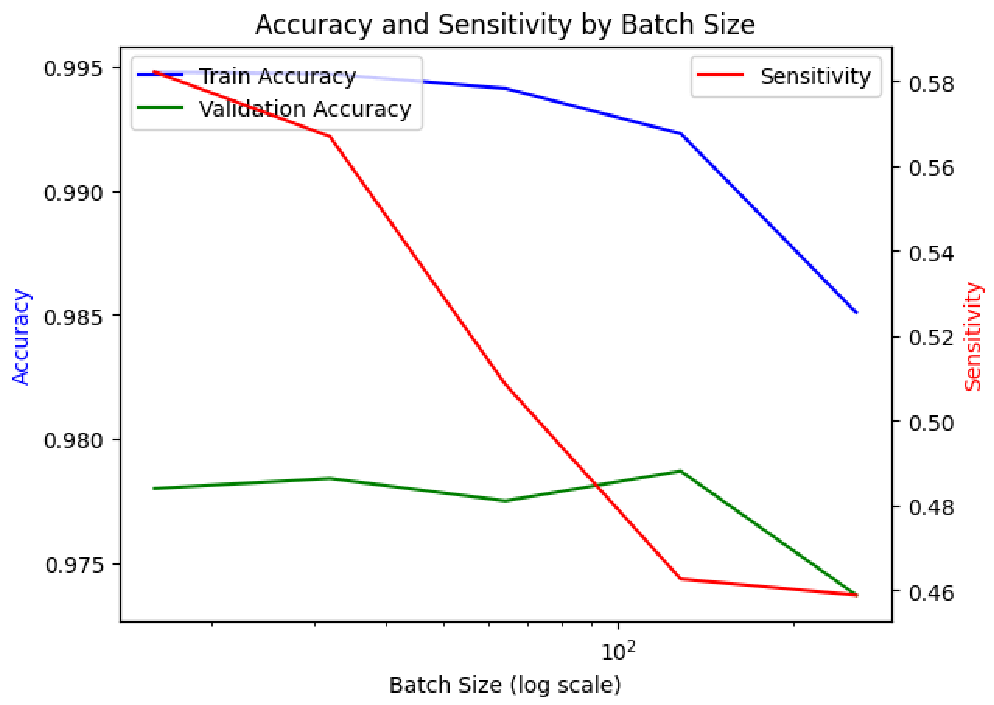
   c) Comment on Results

i) The loss graph shows that both models have a U-shaped curve with a minimum loss at ~alpha 0.1. The training and test losses diverge for alpha values less than 0.01 and greater than 1, which suggests overfitting when moving away from the optimal interpolation between the models. Similarly, the is an inverted U-shape for accuracy, with a peak around the same alpha value where the loss is minimized. The accuracy drops significantly for alpha values less than 0.001 and greater than 1, which coincides with the increase in loss. Both graphs indicate that a blend of both models would result in optimal performance.

2) Part 2
   a) This experiment was based on the task of utilizing the MNIST dataset and five similar models that use different batch sizes during training.
   b) Figures



i)



ii)

c) Comment on Results

i) The results indicate that while larger batch sizes can lead to lower training loss and high training accuracy, they may also cause the model to be more sensitive to input changes and could potentially harm validation performance due to overfitting. The results are also consistent with current understanding in the field: smaller batch sizes tend to provide more noise during training, which can help with generalization, while larger batch sizes provide more accurate estimates of the gradient but can lead to convergence to suboptimal points in the loss landscape. Sensitivity seems to decrease in value as batch size increases. The cross entropy loss increases as the batch size increases suggesting batch size affects the generalization and loss of the model.