8810 Deep Learning Homework 1

Song Liao

March 3, 2020

All the code and images can be found at: `https://github.com/songer12345/cpsc_8810`

1 Deep vs Shallow

1.1 Simulate a Function

Requirement:

Train two different DNN models with same amount of parameters
Compare training process by showing loss in each epoch
Visualize ground-truth and predictions

In this part, I use three DNN models with totally similar parameters, each model and layer are:

Model 1:	1-190-1	571
Model 2:	1-10-18-15-4-1	572
Model 3:	1-5-10-10-10-10-10-5-1	571

Each model contains an input layer, several hidden layers which are dense layers.

I use two functions: First function is $y = \frac{sin(5\pi x)}{5\pi x}$, and the training loss and prediction are shown in Fig.1 and Fig.2.

Second function is: $y = sgn(sin(5/pix))$, and the loss and prediction are shown in Fig.3 and Fig.4.

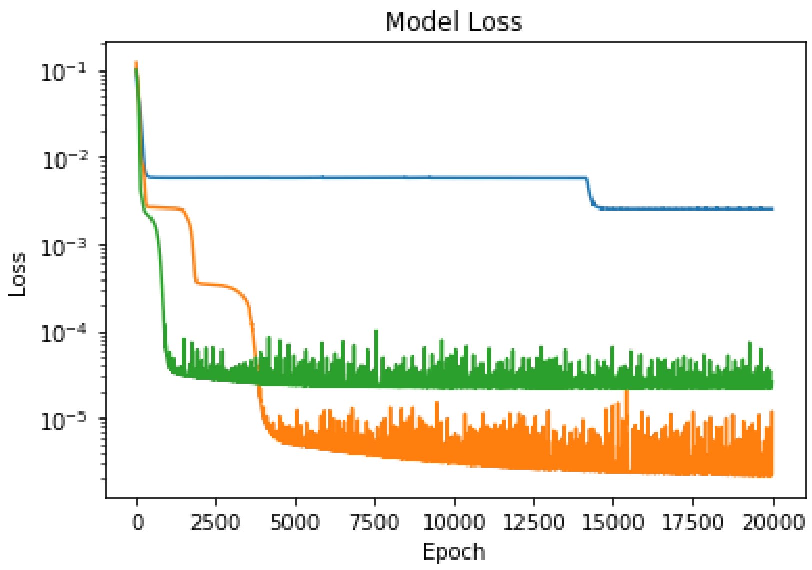

Figure 1: Model loss of function 1

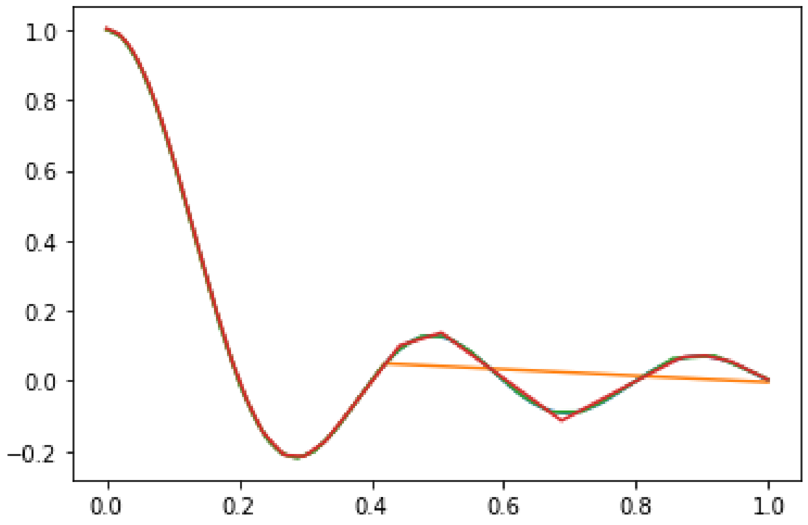

Figure 2: Model prediction of function 1

From the loss we could find that the model 1 has biggest loss and model 3 has smallest loss at last. This may because model 1 only have 1 hidden layer, and model 2 and model 3 have more hidden layers so they are more possible to get better result.

In the prediction graph, the model 1 doesn't fit well, and this is corresponding to the loss doesn't decrease and the model doesn't learn. The model 2 and model 3 has better loss, and because our loss function is to decrease the distance of output layer and y, so lower loss means prediction is closer to ground-truth and they have better prediction. The model three has lowest loss and best

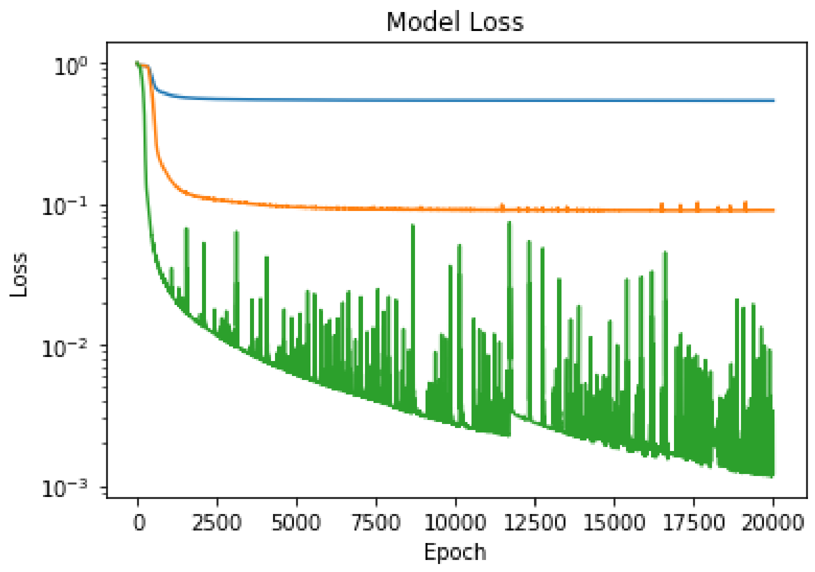

Figure 3: Model loss of function 2

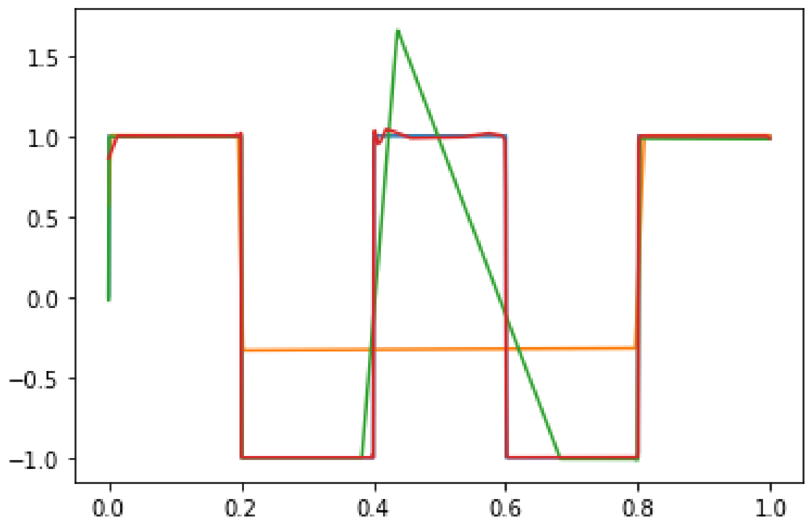

Figure 4: Model prediction of function 2

prediction.

To decrease the overfitting in my model, I use the dropout layer after input layer.

1.2 Train on Actual Tasks

In this task, I use the MNIST dataset with two CNN models. Then I visualize the training process with loss and accuracy. First model I use CNN with 1 hidden layer and 16 dimensions, and second model has 2 hidden layers with both 16 dimensions. The loss of two models are in Fig.5 and the training accuracy as well as testing accuracy are in Fig.6.

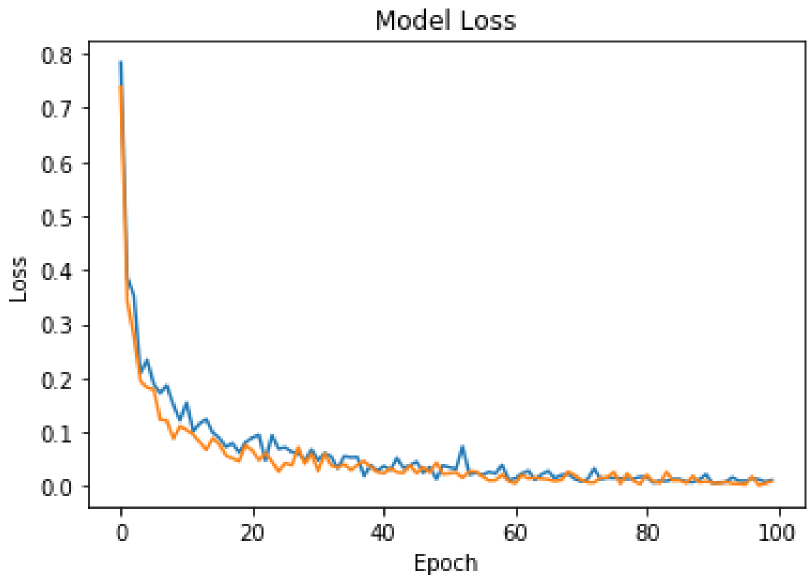

Figure 5: CNN loss on MNIST

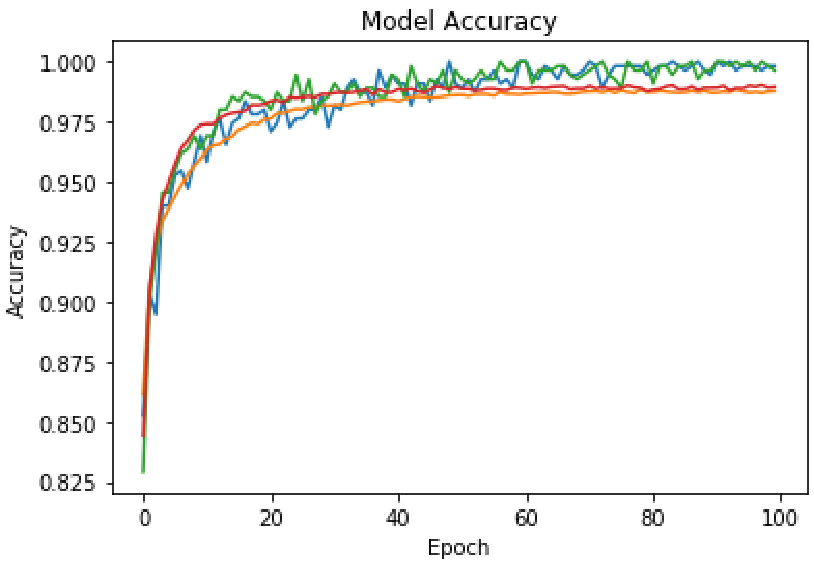

Figure 6: CNN accuracy on MNIST

From the loss, we can find that CNN model with 2 hidden layers are faster to reduce loss and get a better loss. The 2 layers model also has higher accuracy.

2 Optimization

2.1 Visualize optimization process

Requirement:

Collect weights of the model every n epochs.
Also collect the weights of the model of different training events.
Record the accuracy (loss) corresponding to the collected parameters.
Plot the above results on a figure.

In this task, I train a DNN model with three hidden layers with 128, 128, 10 units. First, I use the ”get_weights_variable” function to get the weights of each layer. Then I train the model and store these weights as well as accuracy every 3 epoch. For each model, I run 18 epoches so I would get 6 results. After that, I reshape all the weights to 1 dimension and combine them. For example, first layer has 764*128 parameters and second layer has 128*128 parameters, so the weights are [784, 128] and [128, 128], I reshape them to [1,784*128] and [1,128*128] and combine them as [1, 784*128+128*128]. At last, the vector is processed with PCA and result is a [1,2] vector. Then I draw this result of all 6 epoches and 8 times. The first layer result is shown in Fig.7 and whole model is in Fig.8 .

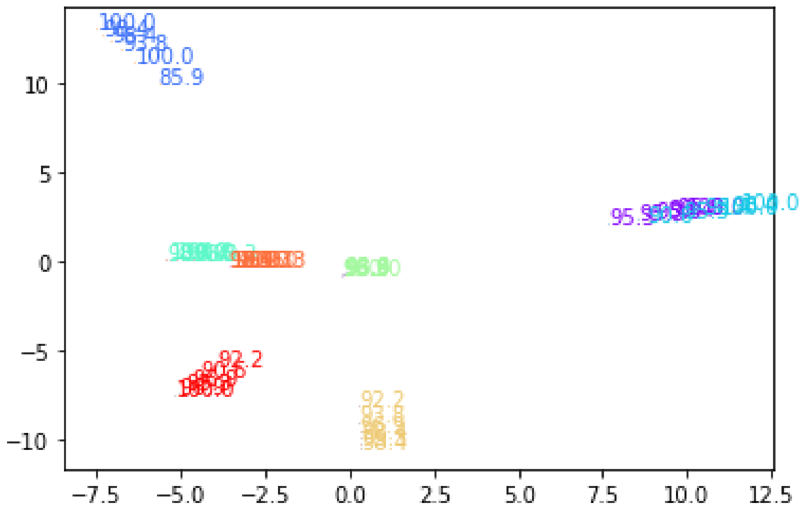

Figure 7: Single layer PCA

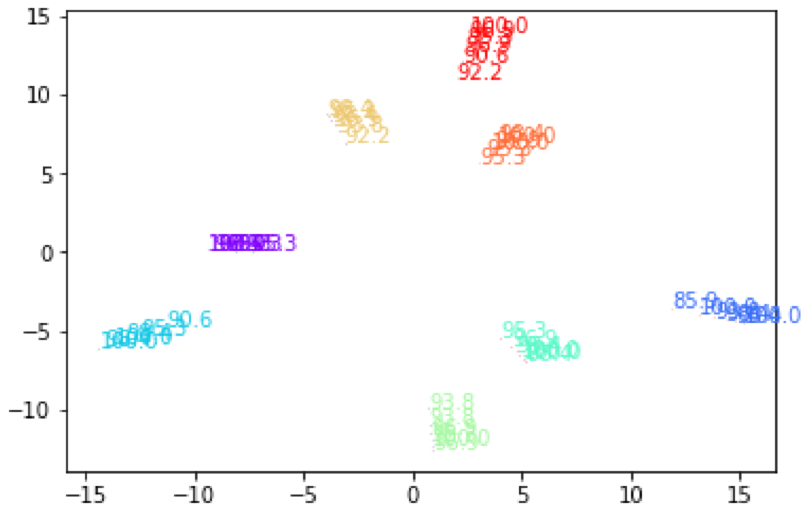
Figure 8: Whole model PCA

2.2 Observe gradient norm during training

In this task, I train two DNN models. For the first function $y = \frac{sin(5\pi x)}{5\pi x}$, the model is pretty simple with 1 layer and 128 units.. The gradient of each layer is defined as from loss to each layer. Then I record the gradient of each layer and add the gradient together. When calculating, I add the square of each value of vector and take the square root. I train the model for 700 iterations. The model gradient is in Fig.9 and loss is shown in Fig.10.

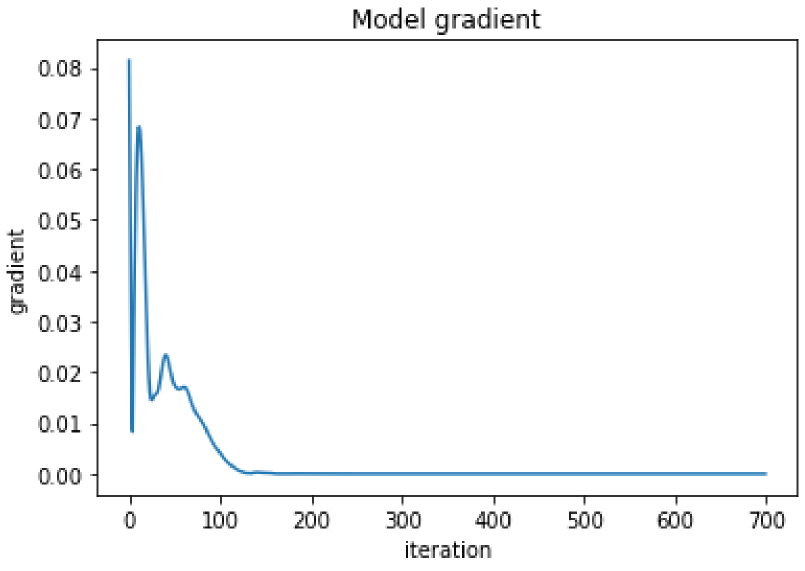

Figure 9: Gradient of model

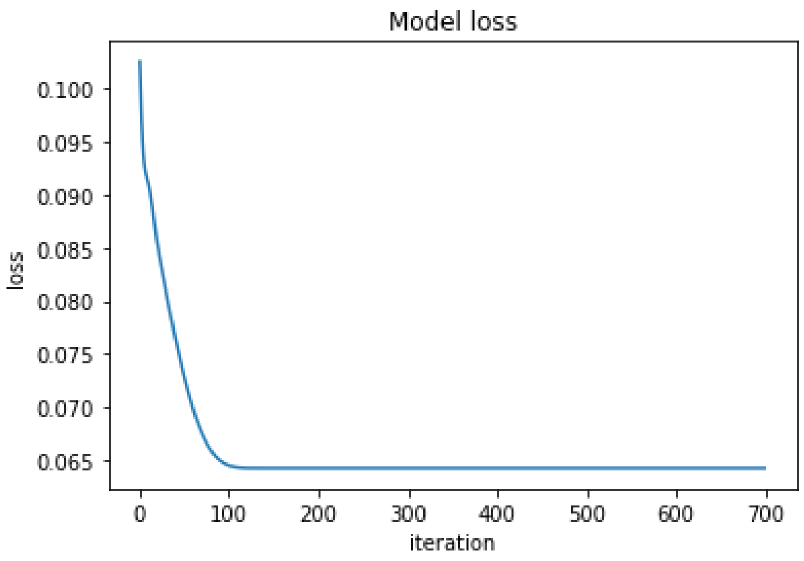

Figure 10: Loss of model

For the second model for MNIST dataset, there are three layers with 128, 128, 10 units. I train the model for 17500 iterations. The model gradient is in Fig.11 and loss is shown in Fig.12.

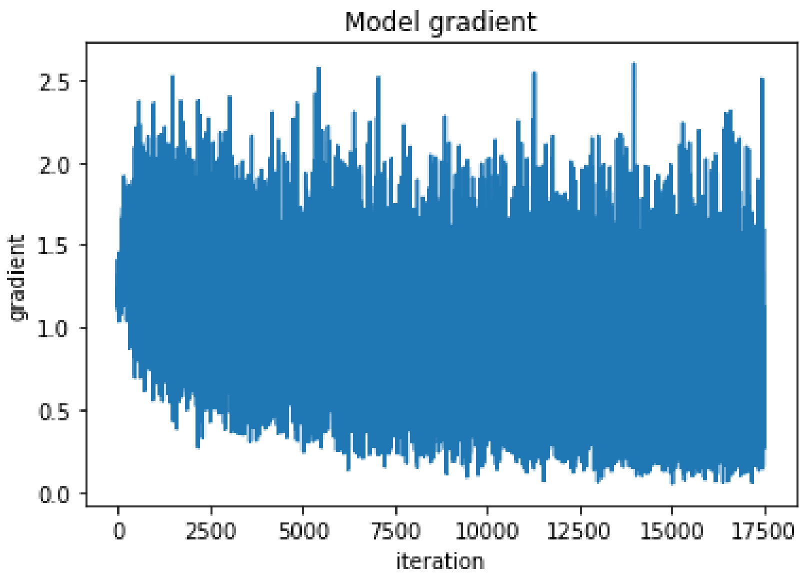

Figure 11: Gradient of model

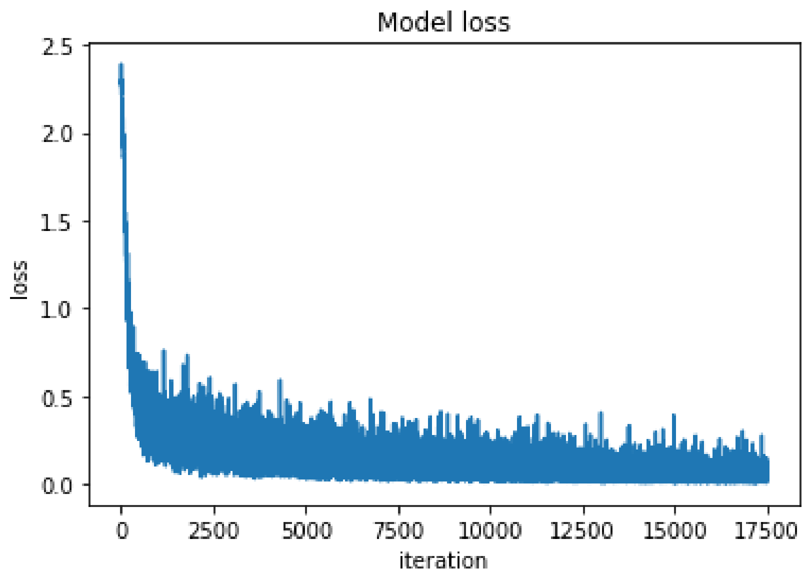

Figure 12: Loss of model

2.3 What happens when gradient is almost zero

In this task, I use the function $y = \frac{sin(5\pi x)}{5\pi x}$ and train a DNN model with 128, 128, 10 units. First, I need to reach the point where gradient norm is zero. I train the model with objective function as loss function. From last task I find the function is convergence with 1000 iterations, so I train the model 1000 iterations and the model gradient is near to zero. Then I change the objective function to gradient and train 1000 iterations to make gradient lower. I choose the 1950 iteration to compute the minimal ratio. When running mode, I saved the weights of each layer for every iteration, and then get the loss of iteration 1950. Then I sample 100 weights from 1900 to 2000 iterations around 1950 and get their loss. The minimal ratio is the proportion that loss is greater than iteration 1950. I train the model 100 times and each time has 2000 iterations and calculate the minimal ration. The loss and minimal ratio is shown in Fig.13.

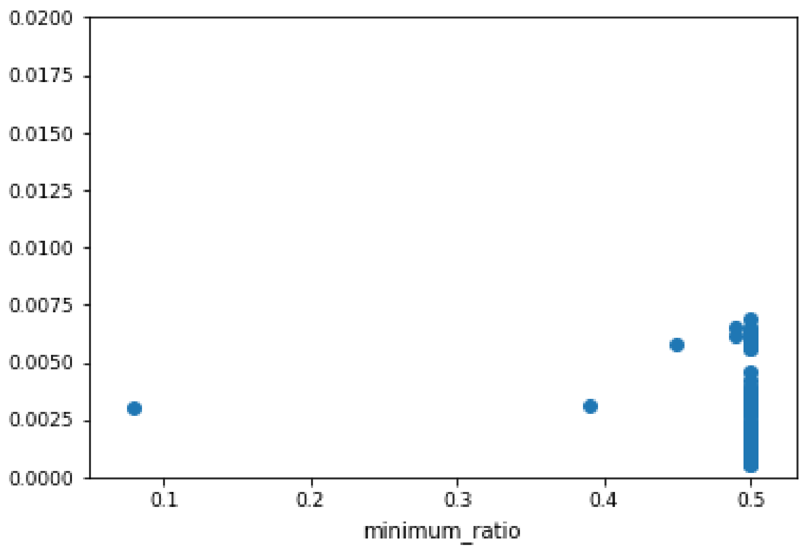

Figure 13: Loss and minimal ratio

3 Generation

3.1 Can network fit random labels?

Requirement:

Train on MNIST or CIFAR-10
Randomly shuffle the label before training.
Try to fit the network with these random labels.

In this task, I train a DNN model with 3 hidden layers with all 256 nodes. The dataset is MNIST. While training, I use "random.shuffle" function to shuffle the label and train. The result is shown in Fig.14. From the result we can find that for a random label, even the model can train well on the training data, it cannot do well on testing data.

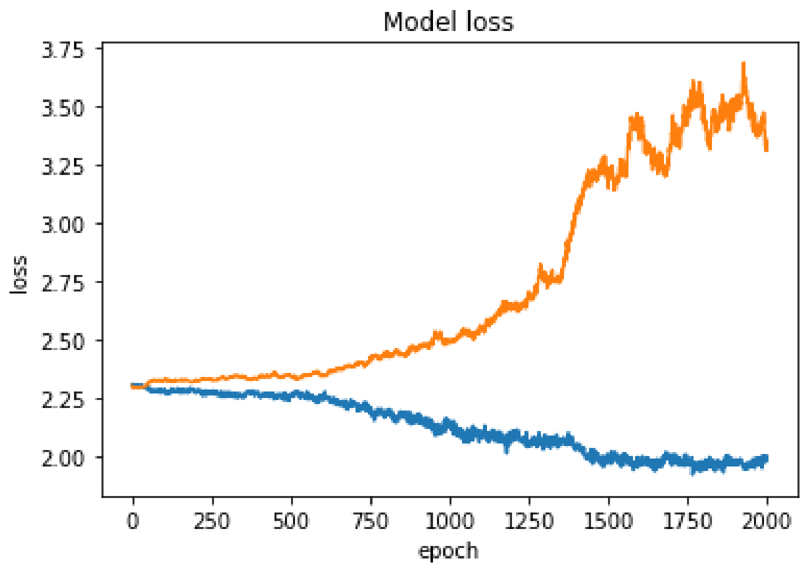

Figure 14: Loss of random labels

3.2 Number of parameters v.s. Generalization

Requirement:
Train on MNIST or CIFAR-10
At least 10 similar-structured models with different amount of parameters
Record both training and testing, loss and accuracy

In this task, I train eleven models with 3 hidden layers but different parameters. The nodes of each model are:
models0=[128, 128, 128]
models1=[128, 128, 256]
models2=[128, 256, 256]
models3=[256, 256, 256]
models4=[256, 256, 512]
models5=[256, 512, 512]
models6=[512, 512, 512]
models7=[512, 512, 1024]
models8=[512, 1024, 1024]
models9=[1024, 1024, 1024]
models10=[1024, 1024, 2048]
And the parameters of each model is range from 13,4794 to 397,3130. For each model, I train the model with 10000 iterations and test it. The loss of training and testing are in Fig.15 and Fig.16.

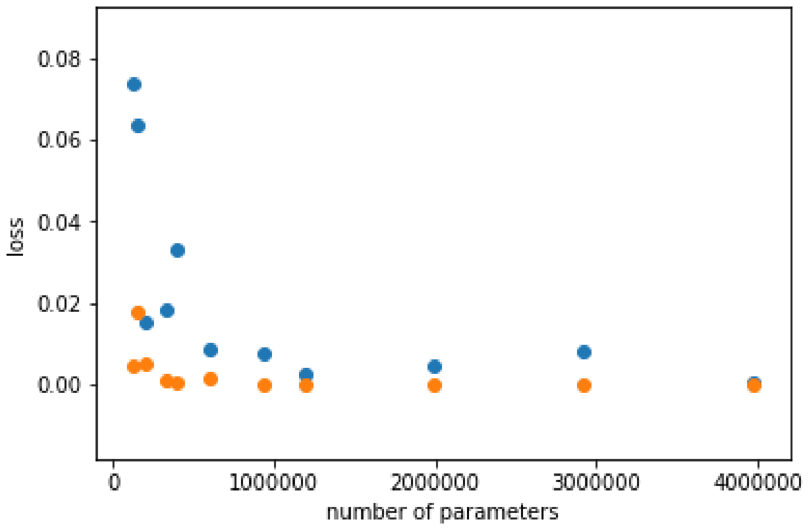

Figure 15: Loss with different parameters

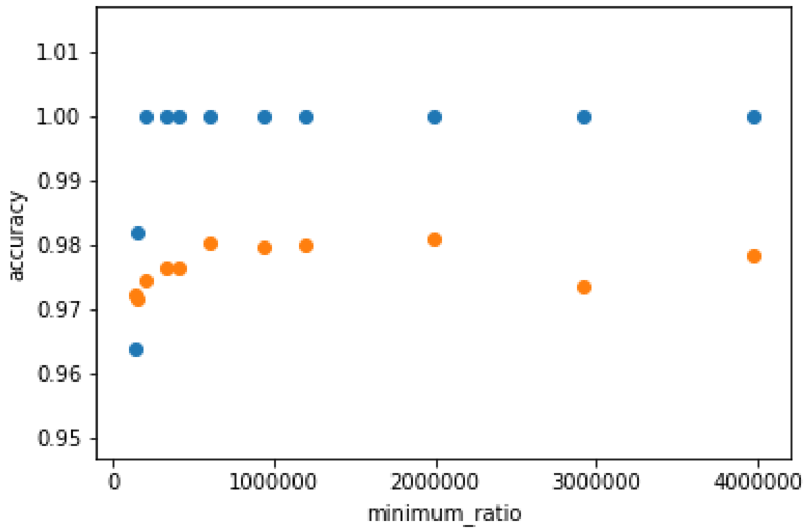

Figure 16: Accuracy with different parameters

From the loss and accuracy graphs we can find: at first stage, the loss decreases and accuracy increases with the increase of parameters number. But after that, even parameter increase a lot, the loss and accuracy don't change too much. It means the model result doesn't always increase with parameter number, so it is useless to only add layers and parameters.

3.3 Flatness v.s. Generalization

3.3.1 Linear interpolation

Requirement:

Train two models (m1 and m2) with different training approach. (e.g. batch size 64 and 1024)

Record the loss and accuracy of the model which is linear interpolation between m1 and m2.

In this task, I train two models with different batch size first. One is 64 and another is 1024. After training, I record all weights of each layer of two models. Then I add each layer weights with different alpha and assign it to third model layer. While assigning, first we need to construction the model and initialize it. Then we get weights of each layer and assign it with previous weights. The weights include kernel parameters and bias parameters. After assigning, we run the third model with training data and testing data. The alpha is set from -1 to 2 with 0.5 interval. The result is in Fig.17.

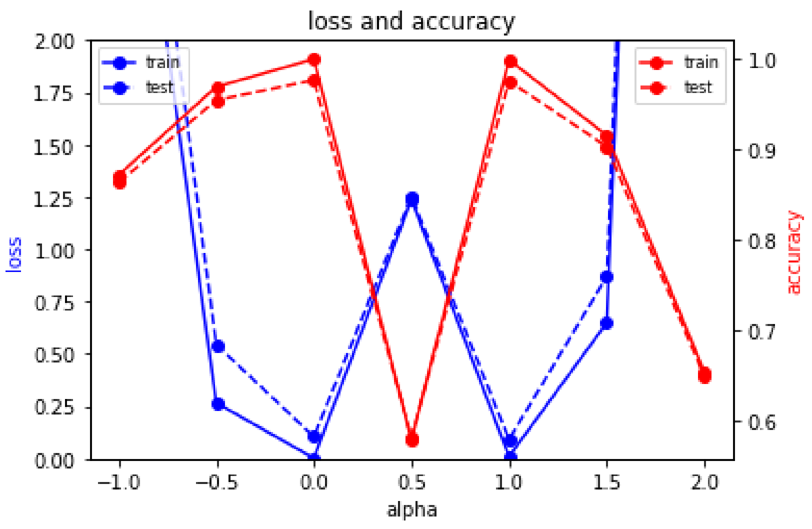

Figure 17: Loss and accuracy with interpolation

When the alpha is 0 or 1, the third model is totally same as first or second model, so the model is trained and it should have different result on training data and testing data. But for other alpha, the model is a new model and not trained with these data, so the training data and testing data are both new data for model and it should perform similar. The reason why graph is not entirely same as we thought may be the training data and testing data are different, so the result can be part of random.

3.3.2 Sensitivity

Requirement:

Train at least 5 models with different training approach.

Record the loss and accuracy of all models.

Record the sensitivity of all models.

In this task, I train 5 models with different batch size as 10, 33, 100, 333, 1000. Then I define the gradient as from loss to input. While training, I record the loss and accuracy as well as gradient . After training, I calculate the Frobenius norm of gradient as sensitivity. The result of loss with sensitivity is in Fig.18 and accuracy with sensitivity is in Fig.19.

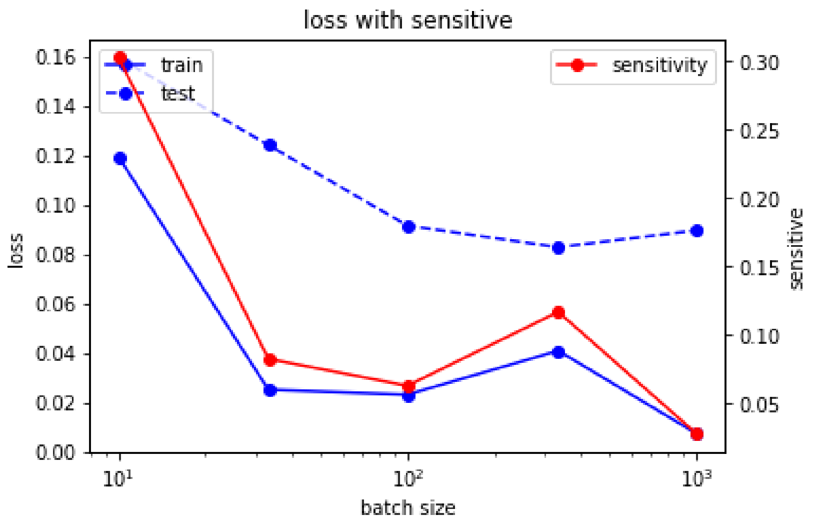

Figure 18: Loss with sensitivity

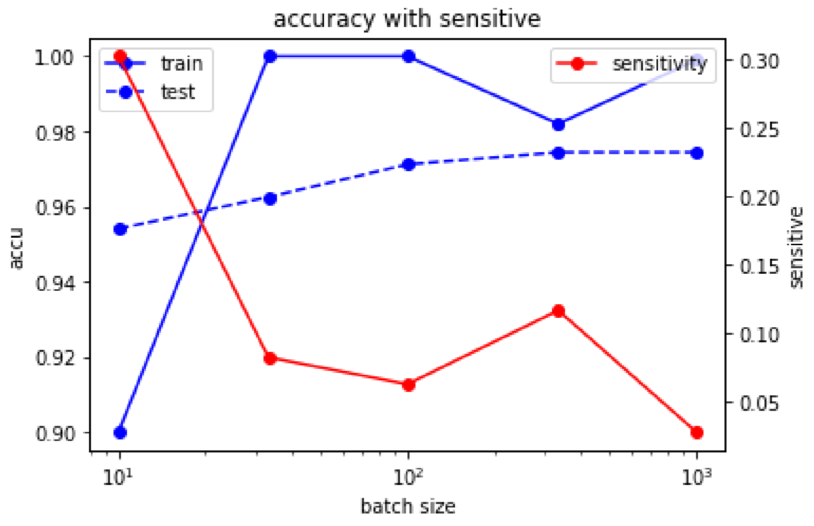

Figure 19: Accuracy with sensitivity

From the result we can find that the sensitivity is decreasing with the increase of batch size. So when the batch size is too large, the perform of model is not very well.