CPSC 8430: Deep Learning – Homework 1

Sadegh Sadeghi Tabas

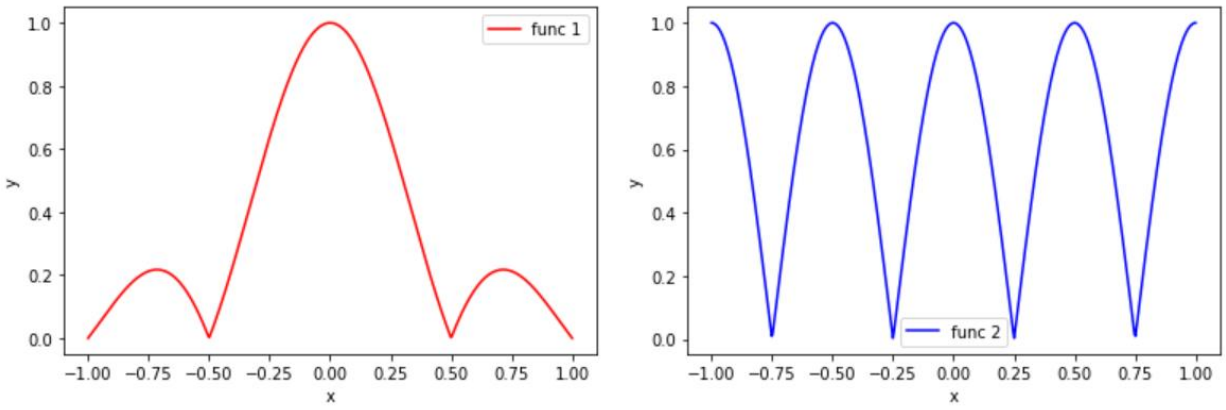
GitHub: https://github.com/sadeghitabas/CPSC8430-DeepLearning.git

1.1. Simulate a Function:

In this assignment, I have chosen two non-liner functions (single input – single output) and generated 500 data points to be used in our training with different DNN models. **Three** different DNN models constructed to be trained and simulate the mentioned functions. Number of parameters kept same for all three models (1387 parameters). The models descriptions and the functions equations presented as follows:

Function 1:	abs(sin(5πx))/(5πx))
Function 2:	abs(cos(2πx))
Model 1:	
	1- Dense layer: Units=150, Activation function: ReLU **2- Dropout layer:** keep_prob = 0.5 **3- Dense layer:** Units=20, Activation function: ReLU **4- Dropout layer:** keep_prob = 0.5 **5- Dense layer 3:** Units=5, Activation function: ReLU **6- Logits:** Units=1, **Activation function:** None **Total parameters =** 3607 **Hyperparameters:** Learning rate: 0.01 Optimizer: Gradient Descent Optimizer Dropout: keep probability=0.5
Model 2:	
	1- Dense layer 1: Units=15, Activation function: ReLU **2- Dense layer 2:** Units=20, Activation function: ReLU **3- Dropout layer:** keep_prob = 0.5 **4- Dense layer 3:** Units=25, Activation function: ReLU **5- Dense layer 4:** Units=40, Activation function: ReLU **6- Dropout layer:** keep_prob = 0.5 **7- Dense layer 5:** Units=29, Activation function: ReLU **8- Dense layer 6:** Units=10, Activation function: ReLU **9- Dense layer 7:** Units=4, Activation function: ReLU **10- Dense layer 8:** Units=2, Activation function: ReLU **11- Logits:** Units=1, **Activation function:** None **Total parameters = 3607** **Hyperparameters:** Learning rate: 0.075 Optimizer: Gradient Descent Optimizer Dropout: keep probability=0.5
Model 3:	
	Dense layer 1: Units=50, Activation function: ReLU **Dropout layer:** keep_prob = 0.5 **Dense layer 2:** Units=45, Activation function: ReLU **Dense layer 3:** Units=15, Activation function: ReLU

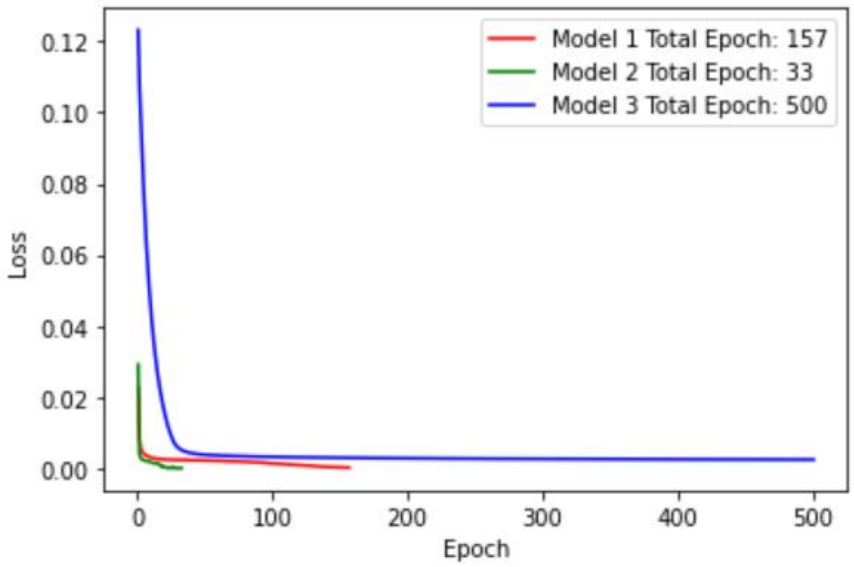
	Dense layer 4: Units=12, Activation function: ReLU **Dropout layer**: keep_prob = 0.5 **Dense layer 5**: Units=10, Activation function: ReLU **Dense layer 6:** Units=5, Activation function: ReLU **Logits:** Units=1, **Activation function:** None **Total parameters =** 3606 **Hyperparameters**: Learning rate: 0.0004 Optimizer: Gradient Descent Optimizer Dropout: keep probability=0.5



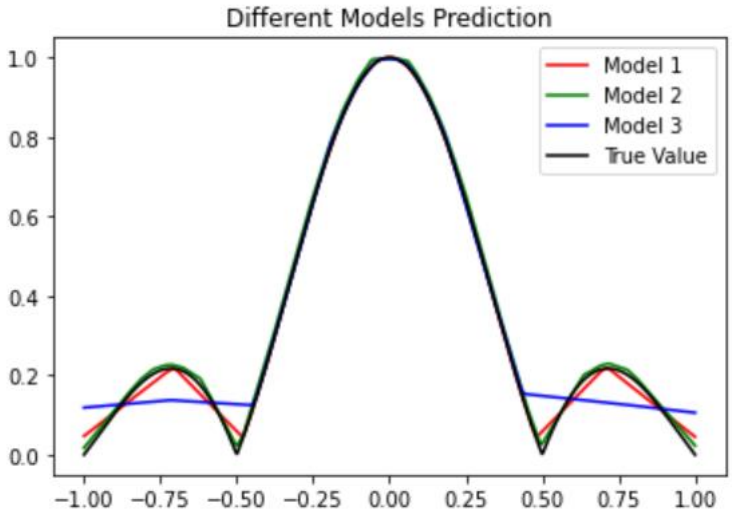
1.2.1. Simulation of Function 1

All the models will converge after reaching maximum epochs or when the model stops learning as the loss will be almost equals to zero. For reducing overfitting of the models, I have used dropout regularization which will also improve generalization of DNN.

As you can see in the below figure the employed models converged at different number of epochs; this number is included in the graph.



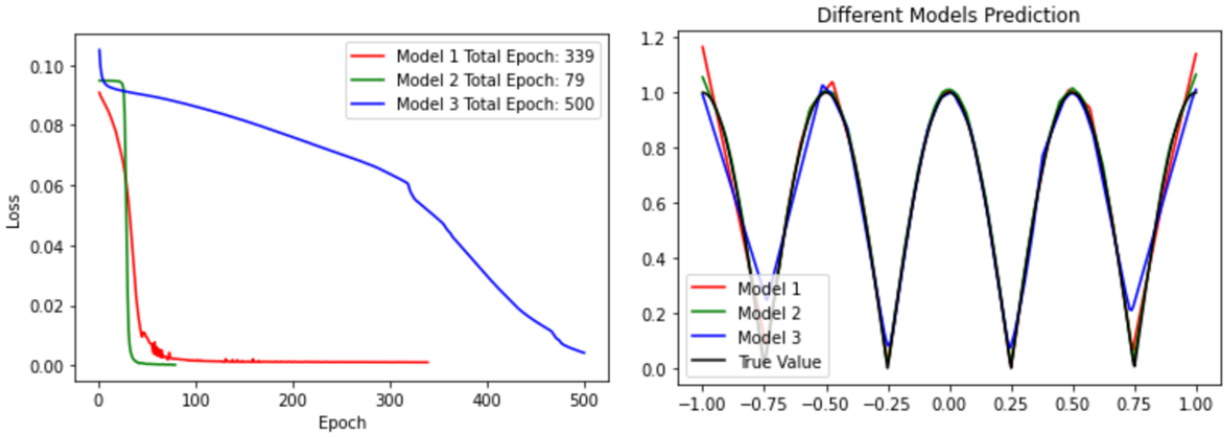
Also, the models prediction as well as the ground-truth is provided in the below figure:



Based on the above figure, the model 3 does not show a good result comparing with the other two. But model 2 converged quickly and indicated lower losses.

1.2.2. Simulation of Function 2

Next, different models employed to simulate the second function and the result is presented in the below figures.



As it can be seen in the above figure model 2 is converged quickly with a lower loss and model 3 after 500 iteration still need more iteration to be converged.

Based on the results of modeling of the mentioned functions, the model 2 outperformed the other two model in simulating function 1 and 2, while the model 3 showed worst results among others. Although all three models have same number of parameters, but the structure is different, and it indicates that the structure of the model plays a pivotal role in the modeling performance.

1.2. Train on Actual Tasks: MNIST

In the second part, I have chosen MNIST dataset of handwritten digits which consists of 55000 training set, 10000 test-set and 5000 validation set. I have trained 2 different DNN models and 1 CNN model on MNIST dataset.

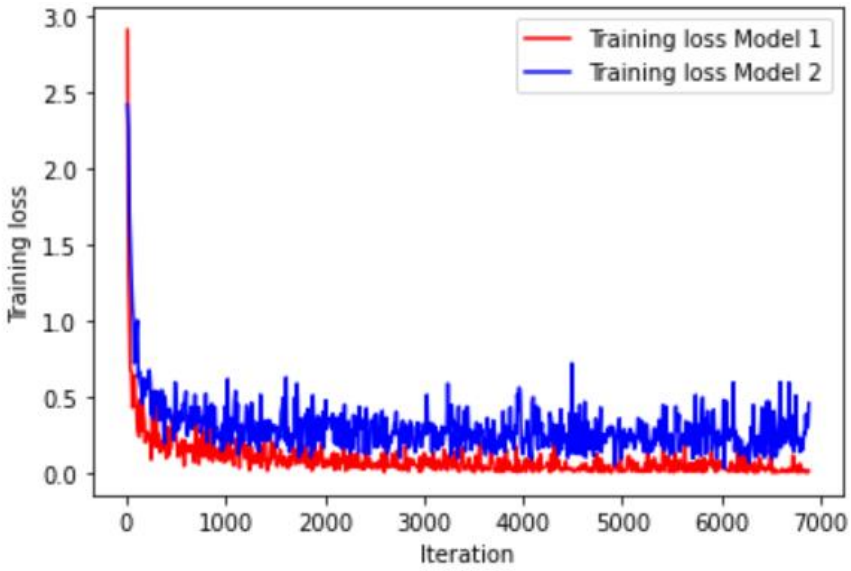
Below table presents the different models structure used to simulate the MNIST dataset.

Dataset	
	MNIST Train-set size: 55000 Validation-set size: 5000 Test-set size: 10000
Model 1:	DNN
	1- Dense layer: Units=784, Activation function: ReLU **2- Dropout layer:** keep_prob = 0.5 **3- Dense layer:** Units=256, Activation function: ReLU **4- Logits:** Units=10, **Activation function:** softmax **Hyperparameters:** Learning rate: 0.015 Optimizer: Adam optimizer Weights initialization: Random normal function with standard deviation=0.0999 Bias initialization: constant value=0.1 Dropout: keep probability=0.5
Model 2:	DNN
	1- Dense layer 1: Units=784, Activation function: ReLU **2- Logits:** Units=10, **Activation function:** softmax **Hyperparameters:** Learning rate: 0.015 Optimizer: Adam optimizer Weights initialization: Random normal function with standard deviation=0.0999 Bias initialization: constant value=0.1 Dropout: keep probability=0.5
Model 2:	CNN
	1- Convolutional layer 1: filters=36, kernel size=5, activation=relu **2- Max pooling:** 2 x 2 **3- Convolutional layer 2:** filters=36, kernel size=5, activation=relu **4- Max pooling:** 2 x 2 **5- Convolutional layer 3:** filters=36, kernel size=5, activation=relu **6- Max pooling:** 2 x 2 **7- Dropout:** keep_prob=0.75 **8- Flattening:** Number of outputs=576 **9- Dense layer 1**: Neurons=576, Activation function: ReLU **10- Dropout:** keep_prob=0.75

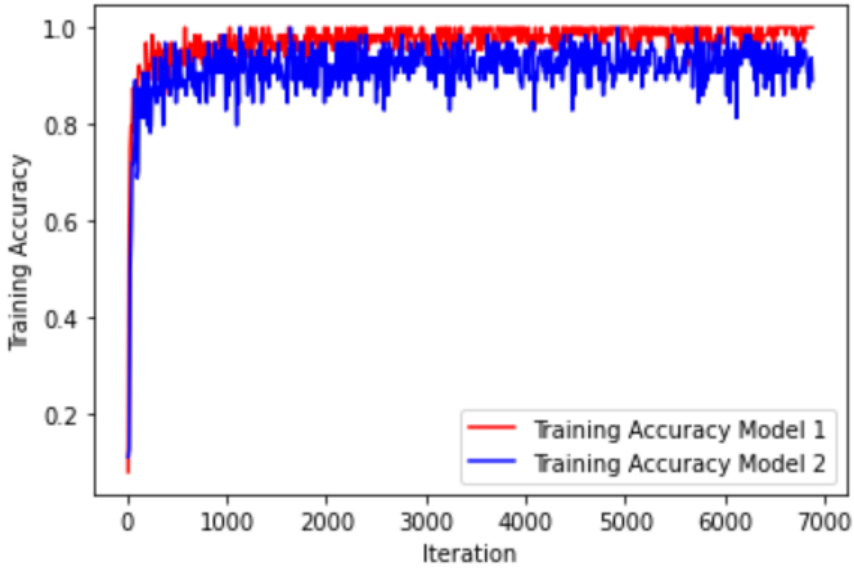
	11- Logits: Neurons=10, Activation function: softmax **Hyperparameters:** Learning rate: 0.001 Optimizer: Adam optimizer Weights initialization: Random normal function with standard deviation=0.0999 Bias initialization: constant value=0.1 Dropout: keep probability=0.5

Training the models: the above 3 models has been trained 8 epochs with a batch size of 64; there are 55000 training labels in the dataset.

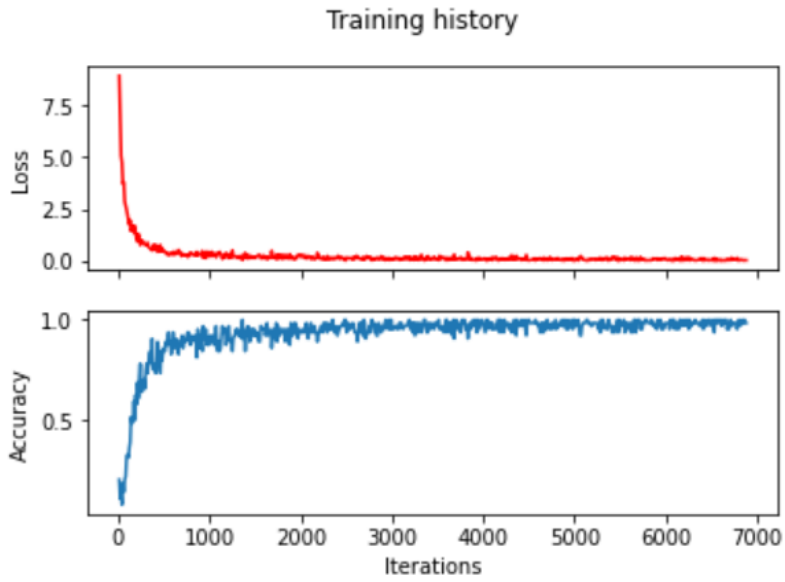
DNNs training loss vs iteration results is provided in the below graph:



DNNs training accuracy vs iteration results is provided in the below graph:



CNN training loss and accuracy vs iteration is provided in the below graph:



After training all 3 models (2 DNNs and 1 CNN) for 8 epochs, the accuracy results from the test dataset which consists of 10000 images presented as follows:

Model 1: 97.42% (9742 / 10000)

Model 2: 92.82% (9282 / 10000)

CNN model: 98.23% (9823 / 10000)

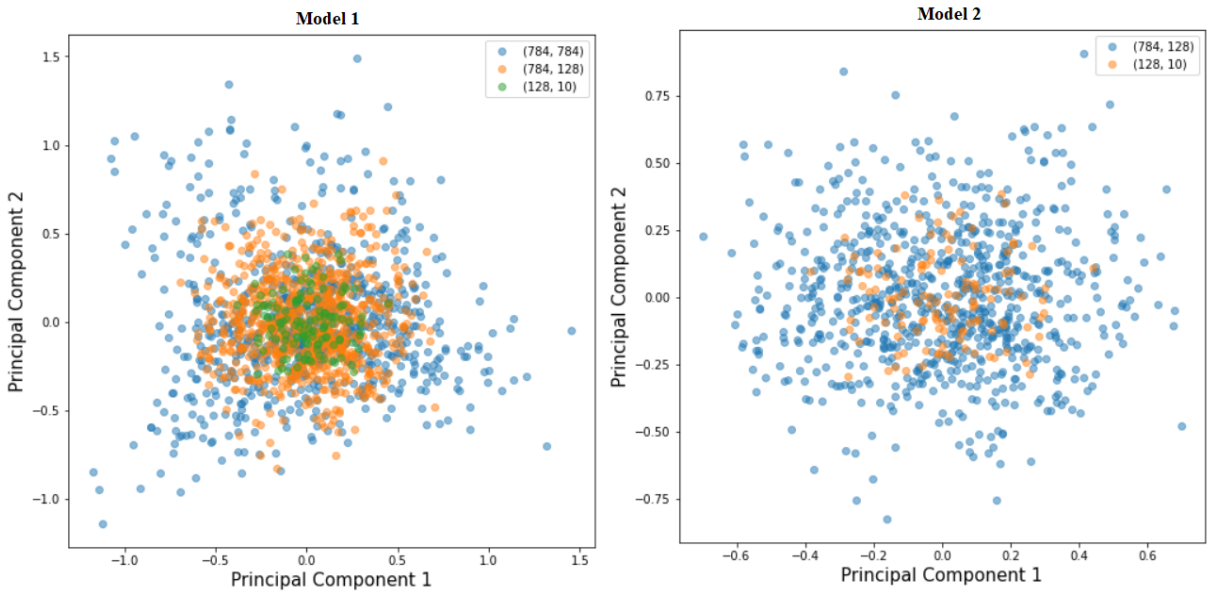
DNN Model 1, performed well on classifying the images on the test dataset with accuracy of

97.42%. On the other hand, as we expected the CNN model showed the best performance on classifying the images.

2. Optimization

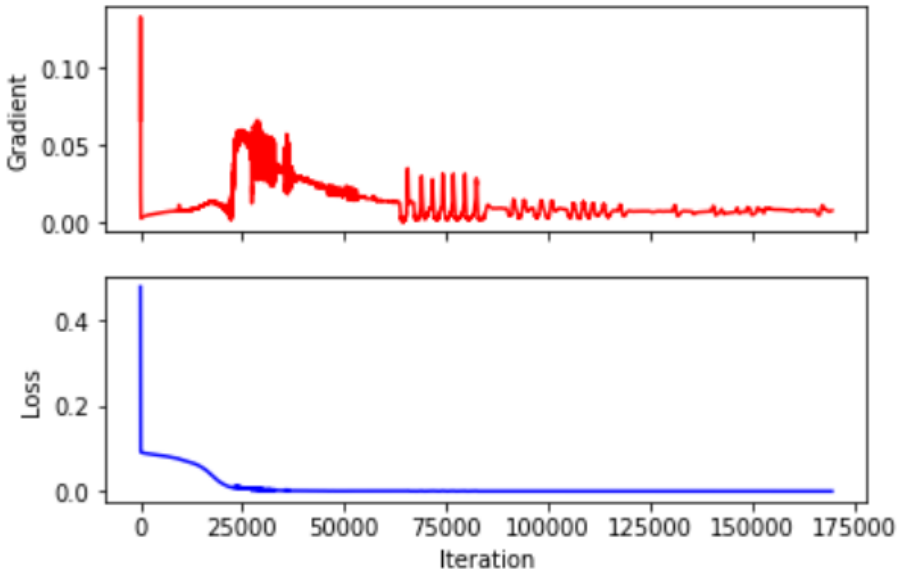
2.1. Visualize the optimization process

For the DNN model on the task of MNIST dataset the weights collected every 3 epochs, and model trained 8 times. DNN model learning rate set to 0.015 and Adam optimizer selected as Optimizer method. The dimension of weights reduced to 2 by PCA. The results presented in the below figures for layer 2 and 3 and the whole model.

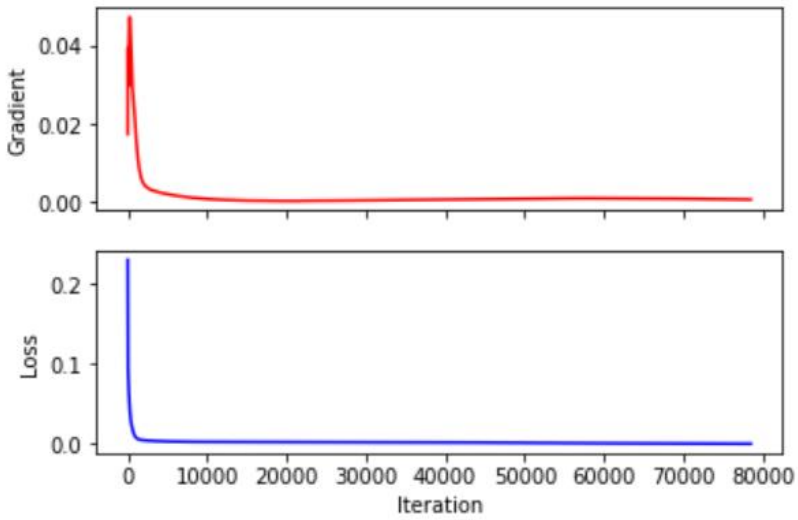


2.2. Visualize the optimization process

Gradient norm for Model 1 – Function 1:



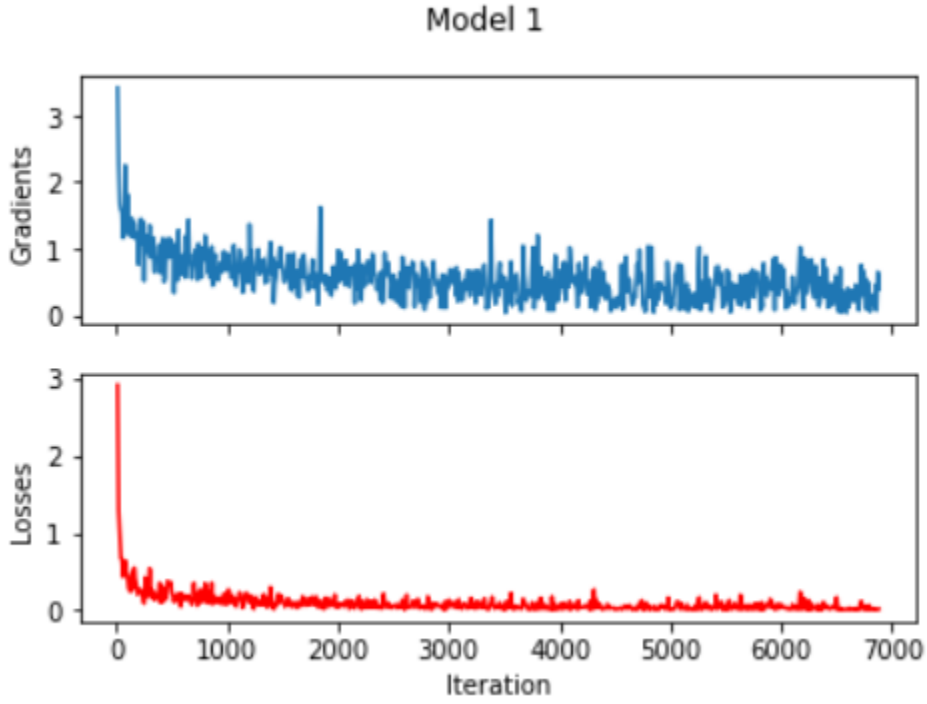
Gradient norm for Model 1 – Function 2:

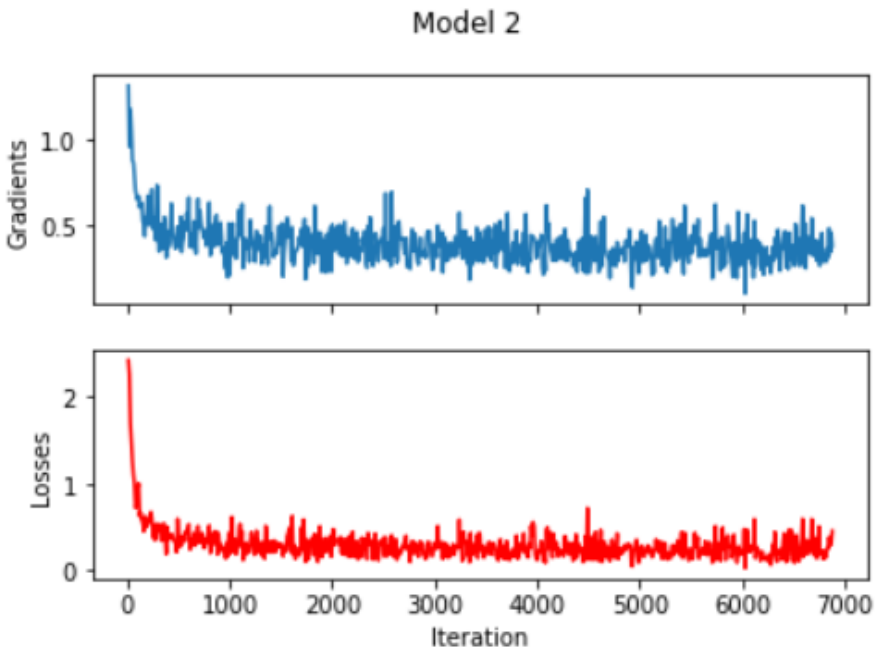


The Model 1- function 1 is trained and converged but there is a slight increase around iteration 25000 in the loss that we can see the associated variation in the gradient norm. As the loss is too less for the Model 1, gradient norm is also significantly smaller.

Train on actual task: MNIST

In addition, below figures show the gradients and losses vs iteration. Based on the presented figures we can say that gradient tell the direction for moving towards the minima. When the gradient is almost zero, we should be at the minima. By changing the objective function to gradient norm or by using the second ordered optimization method like Levenberg-Marquardt algorithm or Newton's method, we can get the gradient norm which will almost be zero. Minimal ratio is defined as the proportion of eigenvalues of hessian matrix which are greater than zero.

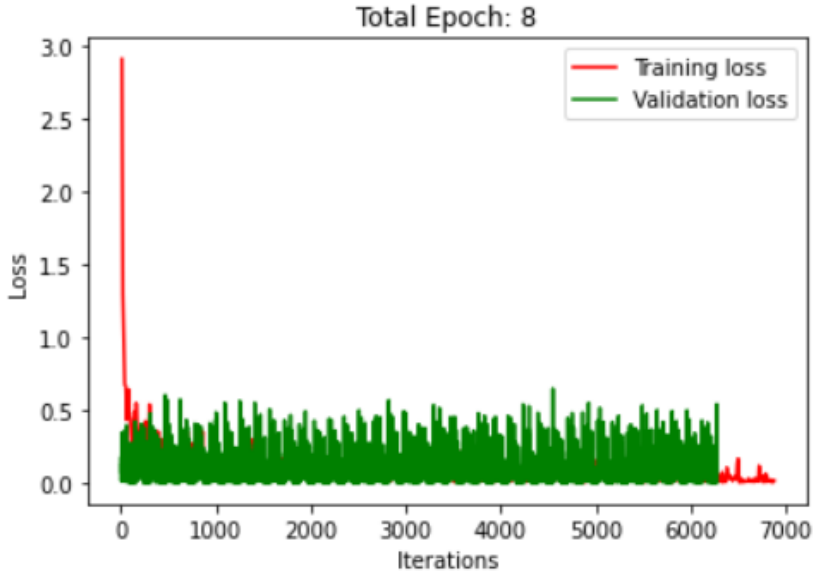




2.3. Generalization

2.3.1 Can network fit random labels?

I've used the MNIST – Model 1 to investigate if the network can fit with random labels. To do so I shuffled labels before training, and I employed Adam optimizer. Batch size = 64 is selected for both train and test. The results is presented in the below figure.



2.3.2. Number of parameters v.s. Generalization

2.3.3. Flatness v.s. Generalization

The above two parts took a lot of time and unfortunately, I was not able to provide the results for them