CPSC 8810: Deep Learning – Homework 1

by - Abhijeet Amle

***Abstract*—In this paper, I have covered the results of simulating a function on DNN models also the results for the DNN and CNN models of the MNIST handwritten digits.**

I. Introduction

In this assignment, for regression problem I have chosen two non-liner functions with 200 data points each; $\cos 5\pi x$ and $(\sin 5\pi x)/(5\pi x)$, trained both the functions with 3 different DNN models keeping the number of parameters for each model equals to 1271. For second part, training on actual task, 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.

II. Homework 1-1

A. Simulating a Function

I have trained 3 different models with same number of parameters 1271 each until convergence. All the models will converge after reaching maximum epochs or when the model stops learning as the loss will be almost equals to zero (loss will stop reducing further). For reducing overfitting of the models, I have used dropout regularization which will also improve generalization of DNN.

Function 1:

Below are the models used for $\cos 5\pi x$.

Model 1:

Dense layer 1: Neurons=100, Activation function: ReLU

Dropout layer: keep_prob = 0.8

Dense layer 2: Neurons=10, Activation function: ReLU

Dense layer 3: Neurons=5, Activation function: ReLU

Logits: Neurons=1, Activation function: None

Total parameters = 1271

Hyperparameters used:

Learning rate: 0.093

Optimizer: Gradient Descent Optimizer

Dropout: keep probability=0.5(during training)

Model 2:

Dense layer 1: Neurons=10, Activation function: ReLU

Dense layer 2: Neurons=10, Activation function: ReLU

Dropout layer: keep_prob = 0.5

Dense layer 3: Neurons=10, Activation function: ReLU

Dense layer 4: Neurons=50, Activation function: ReLU

Dropout layer: keep_prob = 0.5

Dense layer 5: Neurons=6, Activation function: ReLU

Dense layer 6: Neurons=10, Activation function: ReLU

Dense layer 7: Neurons=5, Activation function: ReLU

Dense layer 8: Neurons=7, Activation function: ReLU

Logits: Neurons=1, Activation function: None

Total parameters = 1271

Hyperparameters used:

Learning rate: 0.075

Optimizer: Gradient Descent Optimizer

Dropout: keep probability=0.5(during training)

Model 3:

Dense layer 1: Neurons=10, Activation function: ReLU

Dropout layer: keep_prob = 0.5

Dense layer 2: Neurons=44, Activation function: ReLU

Dense layer 3: Neurons=6, Activation function: ReLU

Dense layer 4: Neurons=25, Activation function: ReLU

Dropout layer: keep_prob = 0.5

Dense layer 5: Neurons=10, Activation function: ReLU

Dense layer 6: Neurons=5, Activation function: ReLU

Logits: Neurons=1, Activation function: None

Total parameters = 1270

Hyperparameters used:

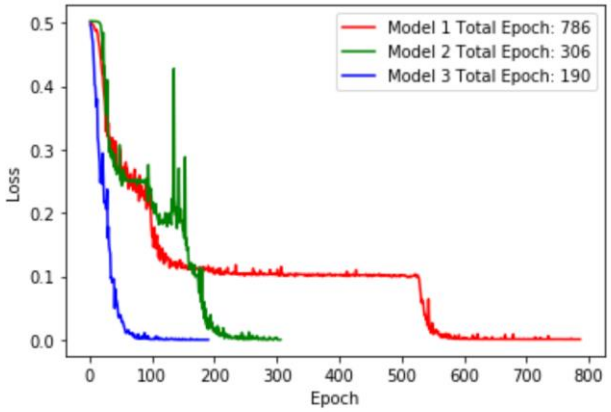
Learning rate: 0.0004

Optimizer: Gradient Descent Optimizer

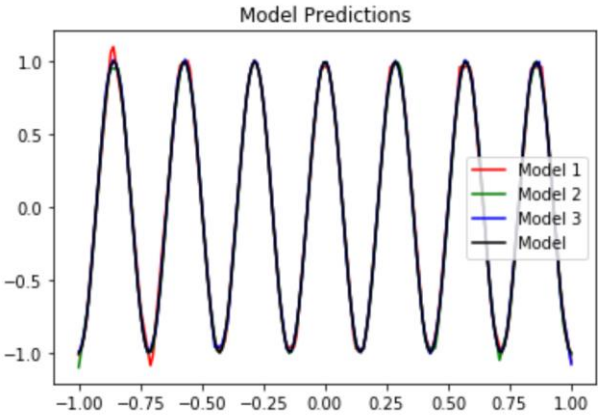
Dropout: keep probability=0.5(during training)

All 3 models are trained until convergence on the same condition which is mentioned earlier. All 3 models converged at

different number of epochs; this number is included in the graph of loss vs epochs as below:



Below graph is for model predictions and the ground-truth:



Conclusion:

Model 1 with 3 hidden layers converged after training for 786 epochs.

Model 2 with 8 hidden layers converged after training for 306 epochs.

Model 3 with 6 hidden layers converged after training for 190 epochs.

Function 2:

Below are the models used for $(\sin 5\pi x)/(5\pi x)$

Model 1:

Dense layer 1: Neurons=100, Activation function: ReLU

Dropout layer: keep_prob = 0.5

Dense layer 2: Neurons=10, Activation function: ReLU

Dropout layer: keep_prob = 0.5

Dense layer 3: Neurons=5, Activation function: ReLU

Logits: Neurons=1, Activation function: None

Total parameters = 1271

Hyperparameters used:

Learning rate: 0.085

Optimizer: Gradient Descent Optimizer

Dropout: keep probability=0.5(during training)

Model 2:

Dense layer 1: Neurons=10, Activation function: ReLU

Dense layer 2: Neurons=10, Activation function: ReLU

Dropout layer: keep_prob = 0.5

Dense layer 3: Neurons=10, Activation function: ReLU

Dropout layer: keep_prob = 0.5

Dense layer 4: Neurons=50, Activation function: ReLU

Dropout layer: keep_prob = 0.5

Dense layer 5: Neurons=6, Activation function: ReLU

Dense layer 6: Neurons=10, Activation function: ReLU

Dense layer 7: Neurons=5, Activation function: ReLU

Dense layer 8: Neurons=7, Activation function: ReLU

Logits: Neurons=1, Activation function: None

Total parameters = 1271

Hyperparameters used:

Learning rate: 0.055

Optimizer: Gradient Descent Optimizer

Dropout: keep probability=0.5(training)

Model 3:

Dense layer 1: Neurons=10, Activation function: ReLU

Dropout layer: keep_prob = 0.5

Dense layer 2: Neurons=44, Activation function: ReLU

Dense layer 3: Neurons=6, Activation function: ReLU

Dense layer 4: Neurons=25, Activation function: ReLU

Dropout layer: keep_prob = 0.5

Dense layer 5: Neurons=10, Activation function: ReLU

Dense layer 6: Neurons=5, Activation function: ReLU

Logits: Neurons=1, Activation function: None

Total parameters = 1270

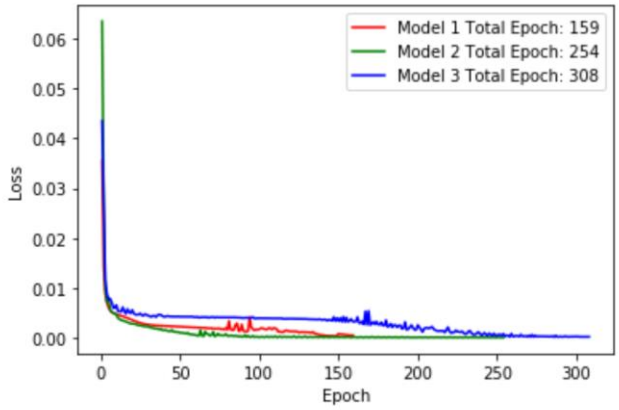
Hyperparameters used:

Learning rate: 0.0004

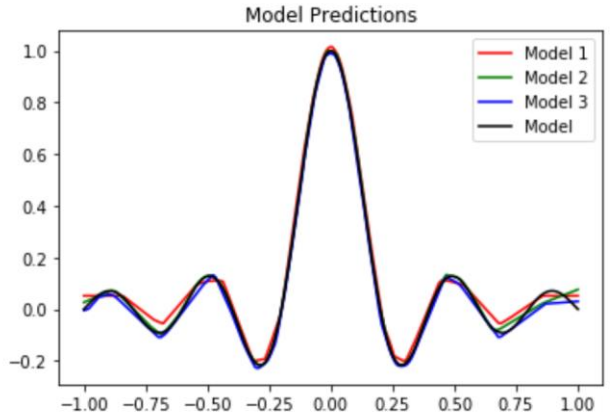
Optimizer: Gradient Descent Optimizer

Dropout: keep probability=0.5(during training)

All 3 models are trained until convergence on the same condition. All 3 models converged at different number of epochs; this number is included in the graph of loss vs epochs as below:



Below graph is for model predictions and the ground-truth:



Conclusion:

Model 1 with 3 hidden layers converged after training for 159 epochs.

Model 2 with 8 hidden layers converged after training for 254 epochs.

Model 3 with 6 hidden layers converged after training for 308 epochs.

B. Train on actual task: MNIST

I have trained 3 models on MNIST, 2 are DNN models and 1 is CNN model.

Below are the DNN models for MNIST dataset.

Model 1:

Dense layer 1: Neurons=784, Activation function: ReLU

Dropout layer: keep_prob = 0.6

Dense layer 2: Neurons=128, Activation function: ReLU

Logits: Neurons=10, Activation function: softmax

Hyperparameters used:

Learning rate: 0.0009

Optimizer: Adam optimizer

Weights initialization: Random normal function with standard deviation=0.0999

Bias initialization: constant value=0.1

Dropout: keep probability=0.5(during training)

Model 2: Model 2 is shallow network with only one hidden layer of 10 neurons.

dense_1: logits	input:	(None, 784)
	output:	(None, 10)

Hyperparameters used:

Activation Function: softmax

Learning Rate: 0.0015

Optimizer: Adam optimizer

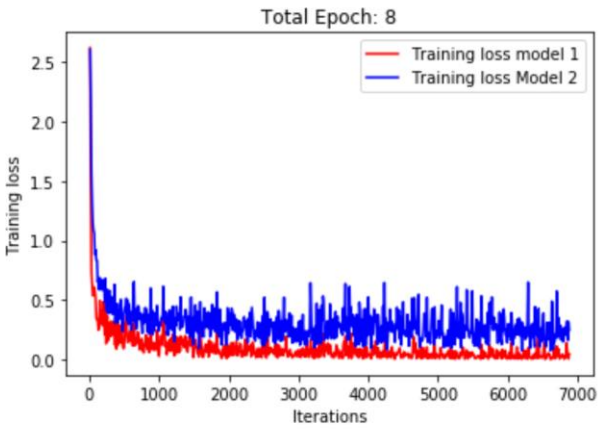
Weights initialization: Random normal function with standard deviation=0.0999

Bias initialization: constant value=0.1

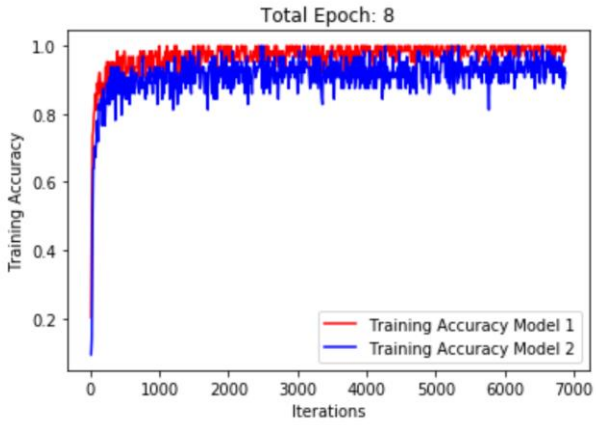
Dropout: keep probability=0.5(during training)

Training the models: I have trained these 2 models for 8 epochs with a batch size of 64; there are 55000 training labels in the dataset.

Below is the graph for training loss vs total iterations:



Below is the graph for training accuracy vs total iterations:



Below is the CNN model for MNIST dataset.

Convolutional layer 1: filters=36, kernel size=5, activation=relu

Max pooling: 2 x 2

Convolutional layer 2: filters=36, kernel size=5, activation=relu

Max pooling: 2 x 2

Convolutional layer 3: filters=36, kernel size=5, activation=relu

Max pooling: 2 x 2

Dropout: keep_prob=0.75

Flattening: Number of outputs=576

Dense layer 1: Neurons=576, Activation function: ReLU

Dropout: keep_prob=0.75

Logits: Neurons=10, Activation function: softmax

Hyperparameters used:

Learning rate: 0.001

Optimizer: Adam optimizer

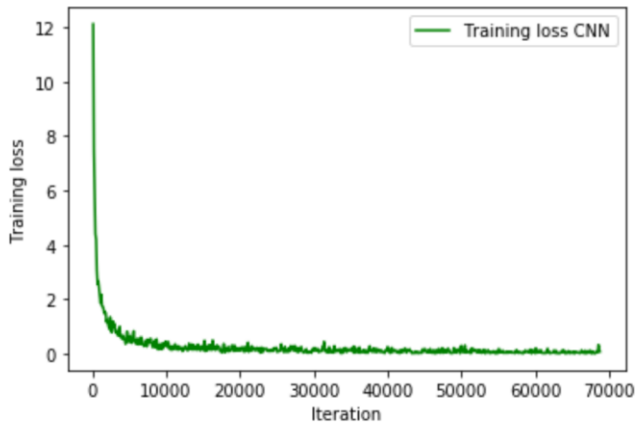
Weights initialization: Random normal function with standard deviation=0.0999

Bias initialization: constant value=0.1

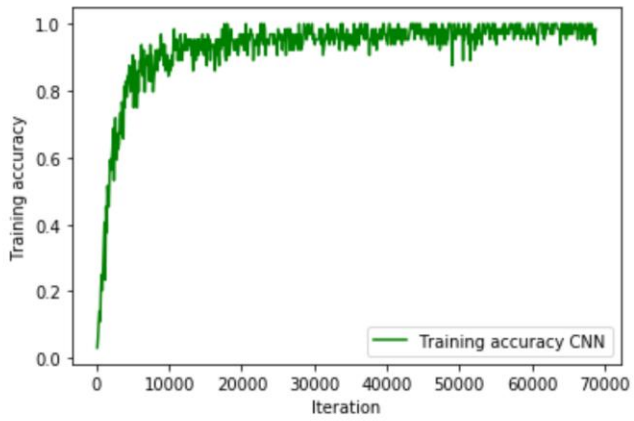
Dropout: keep probability=0.5(during training)

Training the models: I have trained these 2 models for 8 epochs with a batch size of 64; there are 55000 training labels in the dataset.

Below is the graph for training loss vs total iterations:



Below is the graph for training accuracy vs total iterations:



Conclusion:

After training all 3 models (2 DNN and 1 CNN) for 8 epochs, below are the accuracy results from the test dataset which consists of 10000 images.

Model 1: 97.55 %

Model 2: 92.76 %

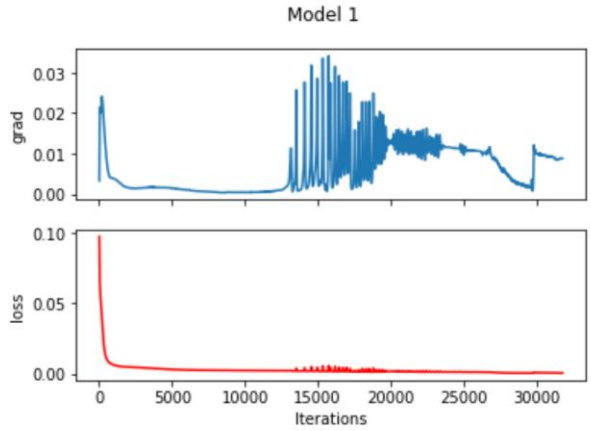
CNN model: 98.04 %

Model 2, despite being only a single layer model performed well on classifying the images on the test dataset with accuracy of 92.76%

III. HOMEWORK 1-2

A. *Simulating a Function*

Gradient norm for sin function Model 1:



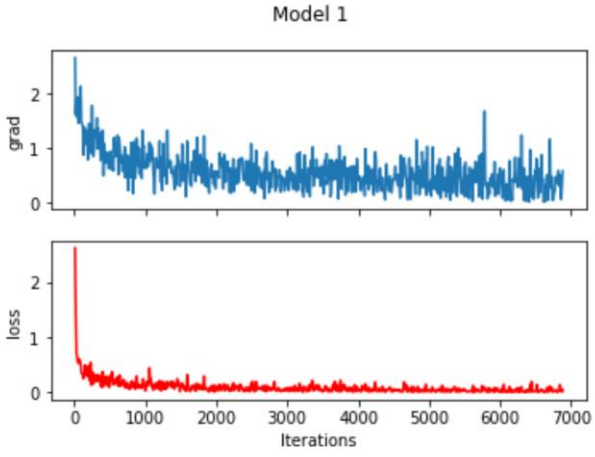
Conclusion:

The Model 1 is trained until convergence. As there is a slight increase around 15000 iterations in the loss during training, we can see a relevant variation in the gradient norm. As the loss is too less for the Model 1, gradient norm is also significantly smaller.

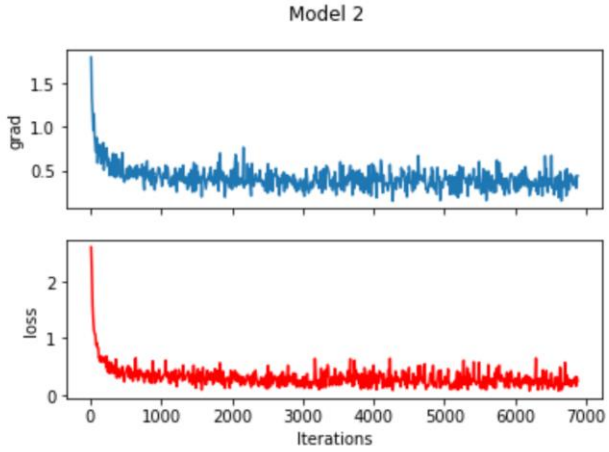
B. *Train on actual task: MNIST*

The model is trained for 8 epochs.

Below is the gradient norm for DNN Model 1:



Below is the gradient norm for Model 2:



Gradient will 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.

Visualizing the optimization process:

Used Model 1 for MNIST dataset.

Dense layer 1: Neurons=784, Activation function: ReLU

Dropout layer: keep_prob = 0.6

Dense layer 2: Neurons=128, Activation function: ReLU

Logits: Neurons=10, Activation function: softmax

Hyperparameters used:

Learning rate: 0.0009

Optimizer: Adam optimizer

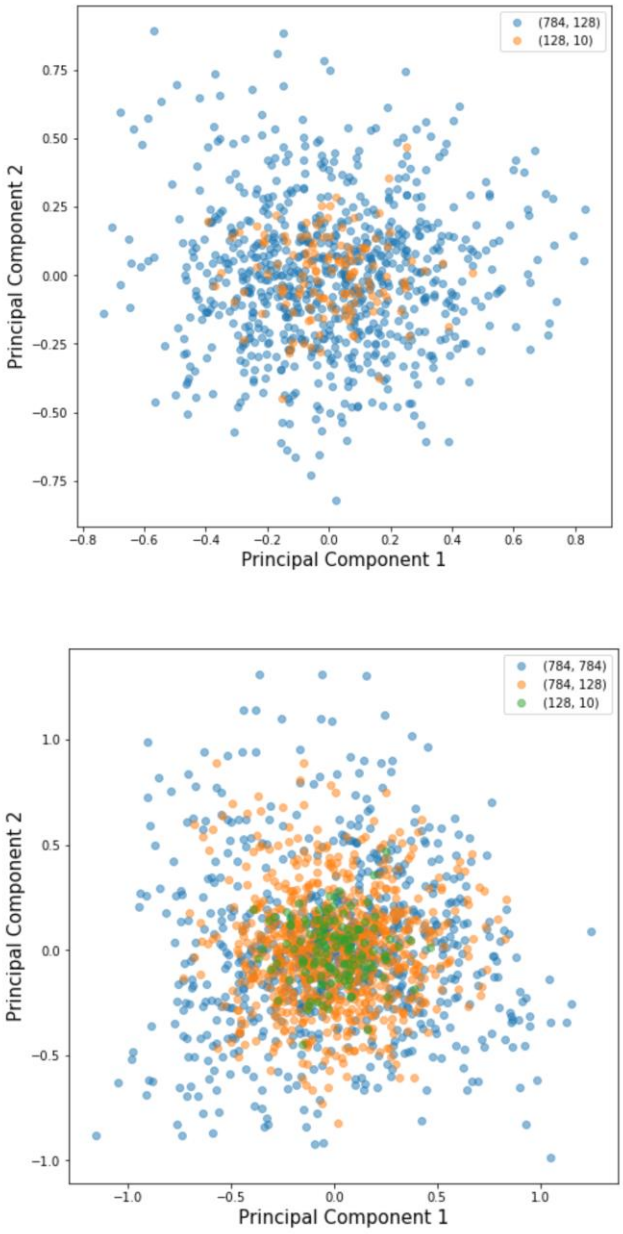
Weights initialization: Random normal function with standard deviation=0.0999

Bias initialization: constant value=0.1

Dropout: keep probability=0.5(during training)

Dimension reduction method: Principle Component Analysis

Total epochs trained = 8



PCA for whole model.

IV. HOMEWORK 1-3

Fitting random labels to the network: MNIST – Model 1

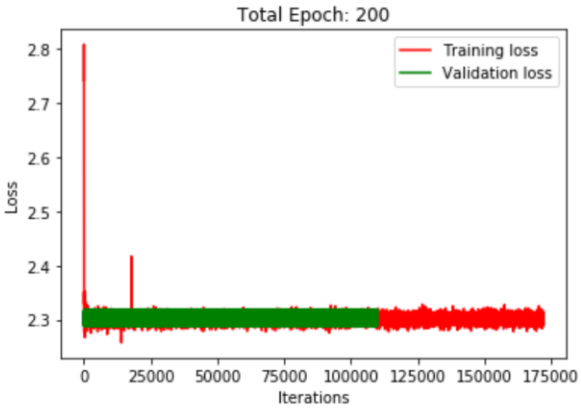
Shuffled labels before training.

Learning rate: 0.0009

Optimizer: Adam optimizer

Batch size: 64 for both training and testing

Total epoch: 200



REFERENCES

[1] https://github.com/CpSc8810/DeepLearning/blob/master/Tutorial/2_tf_cnn_classification.ipynb

[2] https://colab.research.google.com/drive/1JMuqstQmiM4L7Bf9mzEl68c1NANTmTRC