**Sandya Rani Prasadam**

**Git Hub:** [**https://github.com/sandya-33/Deep-learning-hw1**](https://github.com/sandya-33/Deep-learning-hw1)

**1-1 Simulate a function:**

I have used 3 models. I used a batch size of 64 for all the models. And I used 20% of train set to do validation.

**Hyper parameters**:

Learning rate: 0.01

Weight\_decay: 0.005

Batch size: 64

Loss function: nn.CrossEntropyLoss()

Optimization function: SGD

Epochs= 100

Model 1:

* 2d convolutional layer apply max pool(relu)
* 2d convolutional layer apply max pool(relu)
* 2d convolutional layer apply max pool(relu)
* 2d dense layer ->relu
* 2d dense layer ->relu
* 2d dense layer-> dropout
* 2d dense layer-> dropout
* 2d dense layer

Number of parameters: 581866

Model 2:

* 2d convolutional layer apply->relu
* 2d convolutional layer ->relu ->max pool
* 2d convolutional layer apply->relu
* 2d convolutional layer ->relu ->max pool
* drpoout
* 2d convolutional layer apply->relu
* 2d convolutional layer ->relu ->max pool
* Dropout

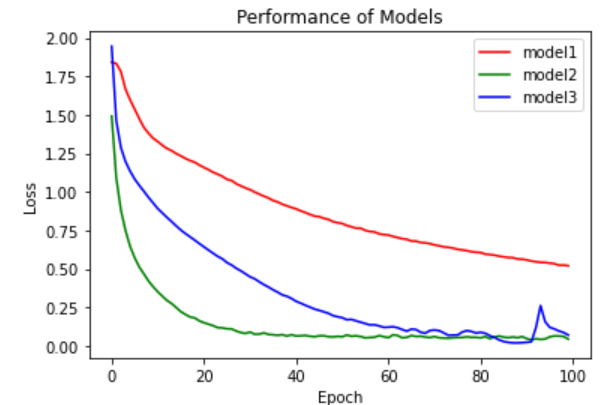
Number of parameters: 5853170

Model 3:

* 2d convolutional layer apply
* 2d convolutional layer ->max pool
* 2d convolutional layer apply
* 2d convolutional layer ->max pool
* 2d dense layer -> relu
* 2d dense layer

Number of parameters: 278752

**Training loss of all models**



1-1-2

**Ground truth curve** model 1:red, model2: blue, model 3: green actual: black



**Findings**:

I found that model 2 is performing well when compared to all the other models. And model 1 is not performing well. Mode1 2’s definition is divided into 3 blocks and have more convolutional layers compared to other two structures. We can see that at one point, between 0, 20 the loss drop is more compared to other two layers.

**1-2: Visualize the optimization process**

Model:

* 2d convolutional layer apply max pool(relu)
* 2d convolutional layer apply max pool(relu)
* 2d convolutional layer apply max pool(relu)
* 2d dense layer ->relu
* 2d dense layer ->relu
* 2d dense layer-> dropout
* 2d dense layer-> dropout
* 2d dense layer

Number of parameters: 581866

**Hyper parameters:**

Learning rate: 0.01

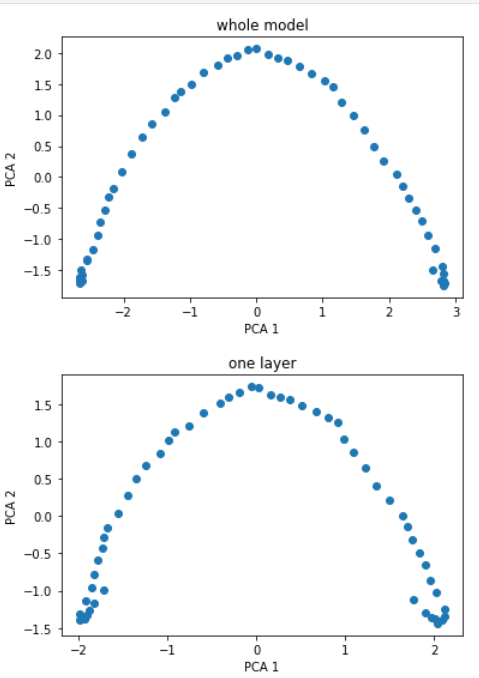
Weight\_decay: 0.005

Batch size: 64

Loss function: nn.CrossEntropyLoss()

Optimization function: SGD

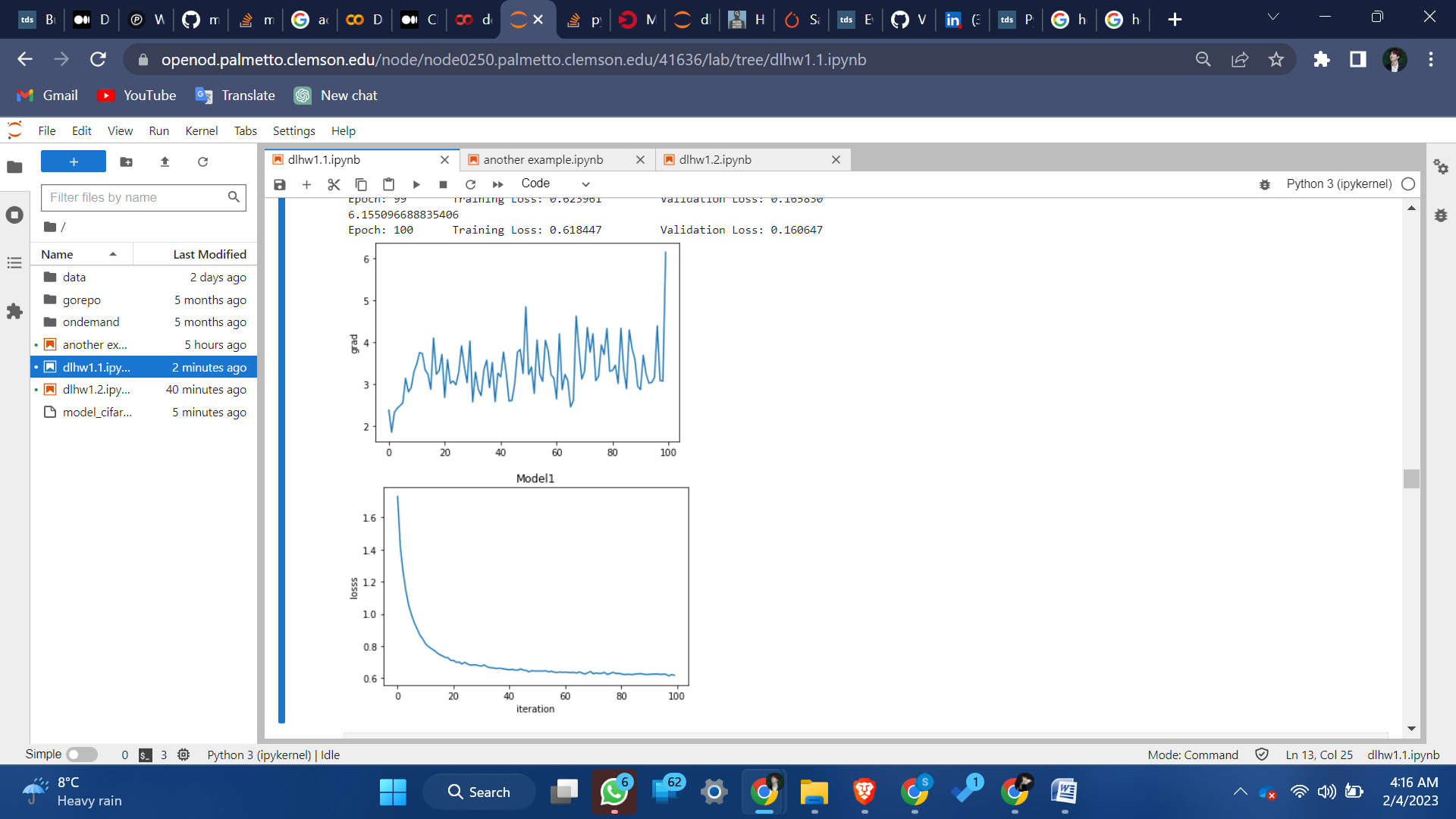
Epochs= 20



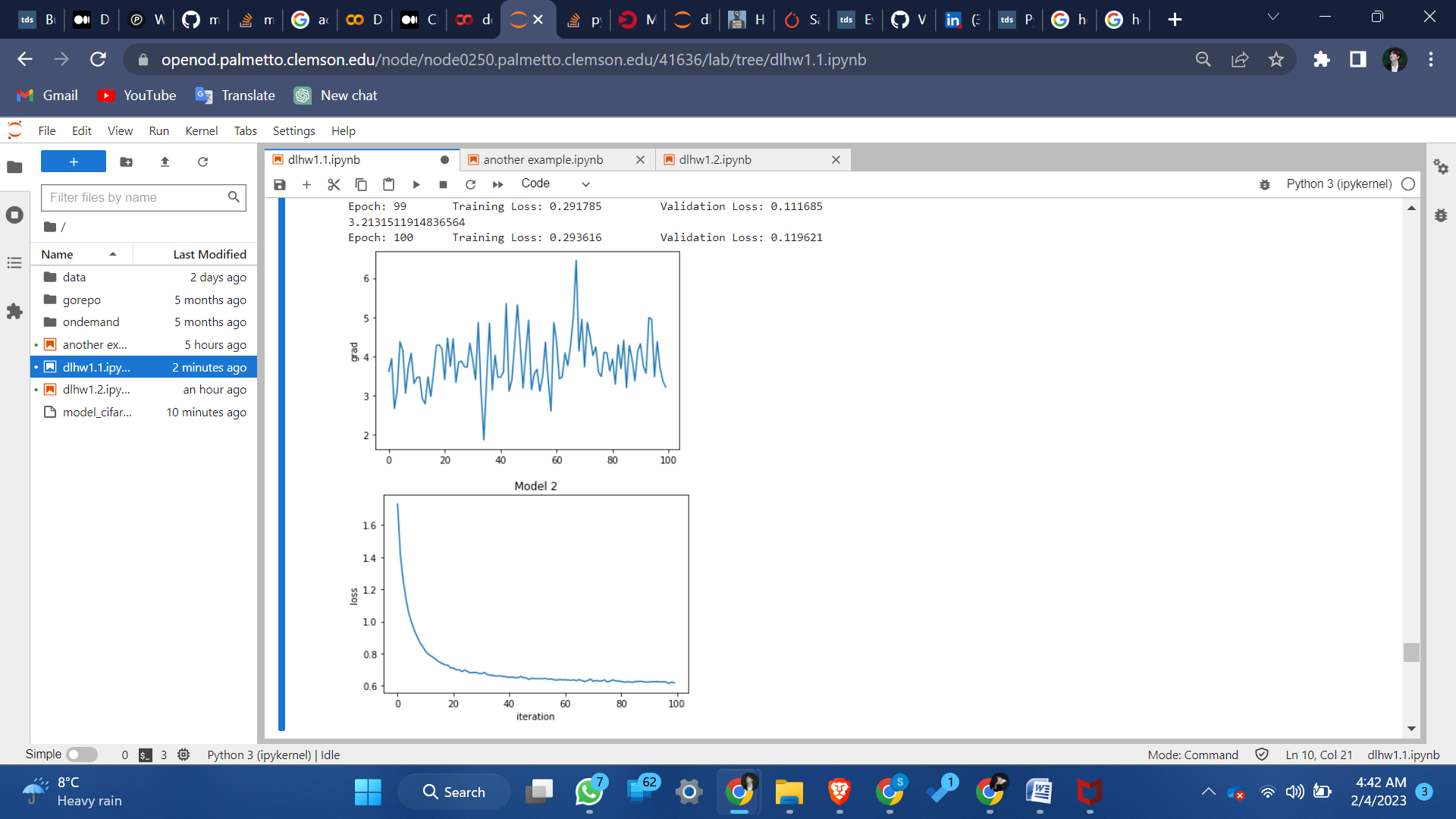
This graph represents the weights collected in 8 cycles and then applied pca algorithm to reduce the weights dimensions. There is not much change in the second graph to the first graph. Only some points are added. We can see that weights in the second graph changed at the edges of the graph.

1-2 **Loss and Gradient norm**

**Model-1**

****

**Model-2**

**`**

The graph is plotted as epoch vs loss and epoch vs gradient norm. The number of epoch taken are 100. We can say that in terms of loss both the models are nearly the same. Both the models are converged and trained. Between 60 and 80 epochs the loss and the grad norm are significantly decreased and the value is close to 0.

**1-2: when gradient is almost zero**

**1-3: Training and testing loss after shuffling the data**

**Model:**

* 2d convolutional layer apply max pool(relu)
* 2d convolutional layer apply max pool(relu)
* 2d convolutional layer apply max pool(relu)
* 2d dense layer ->relu
* 2d dense layer ->relu
* 2d dense layer-> dropout
* 2d dense layer-> dropout
* 2d dense layer

Number of parameters: 581866

**Hyper parameters:**

Learning rate: 0.001

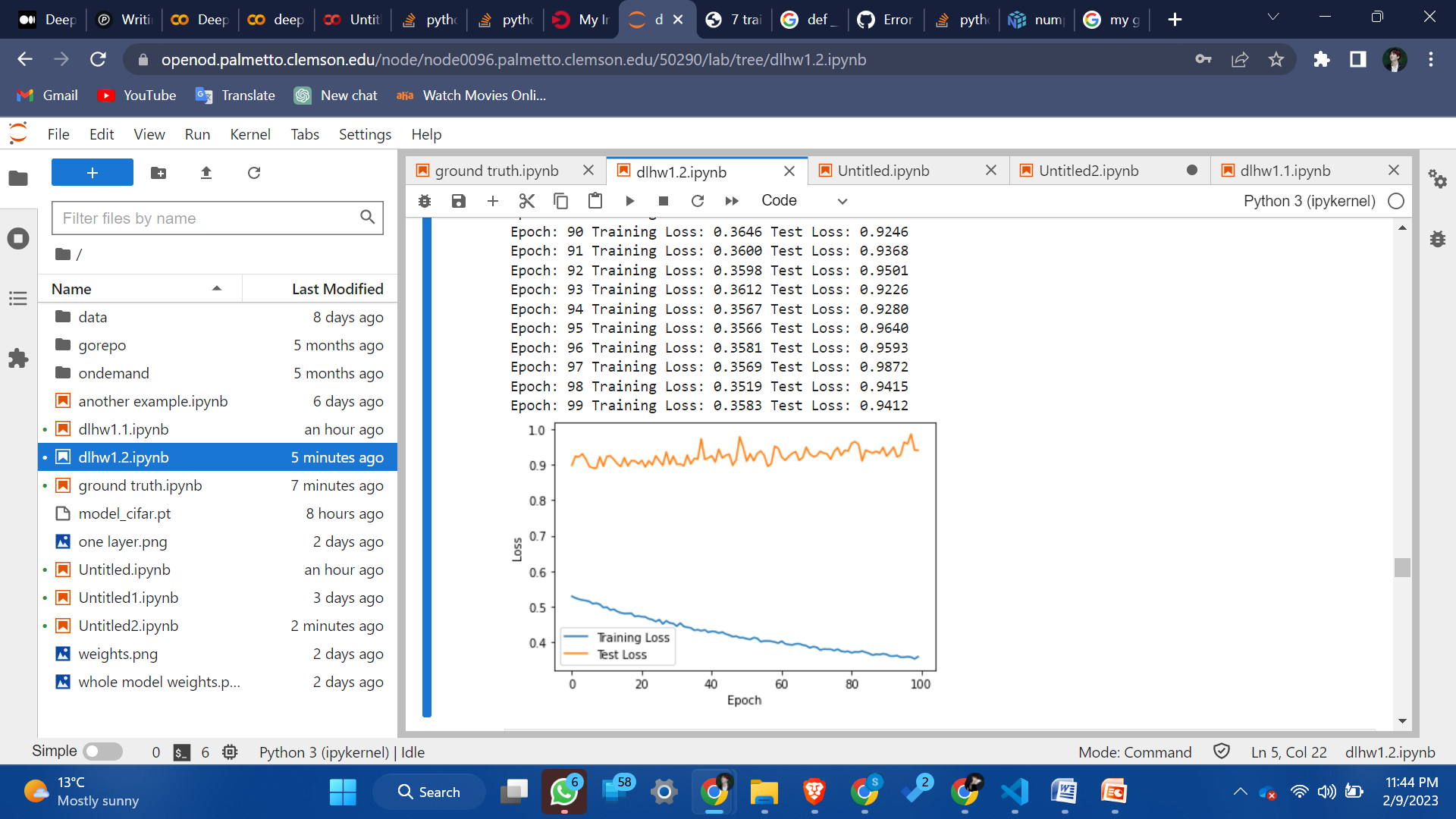
Weight\_decay: 0.005

Batch size: 64

Loss function: nn.CrossEntropyLoss()

Optimization function: SGD

Epochs= 100

****

The above graph is based on a model which is given with shuffled labels. That means we gave the model wrong data. So it is understandable that the training loss decreased when testing loss increased over the number of epochs. As the training process adopts itself for the data given.

**1-3 Number of parameters v.s. Generalization**

I used the Cifar-10 dataset for this part.

**Model:**

* 3 dense layer-> relu

**Hyper parameters:**

Learning rate: 0.001

Weight\_decay: 0.005

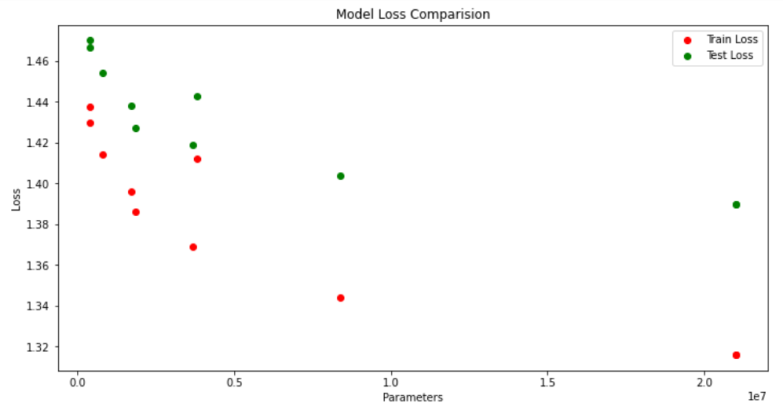
Batch size: 64

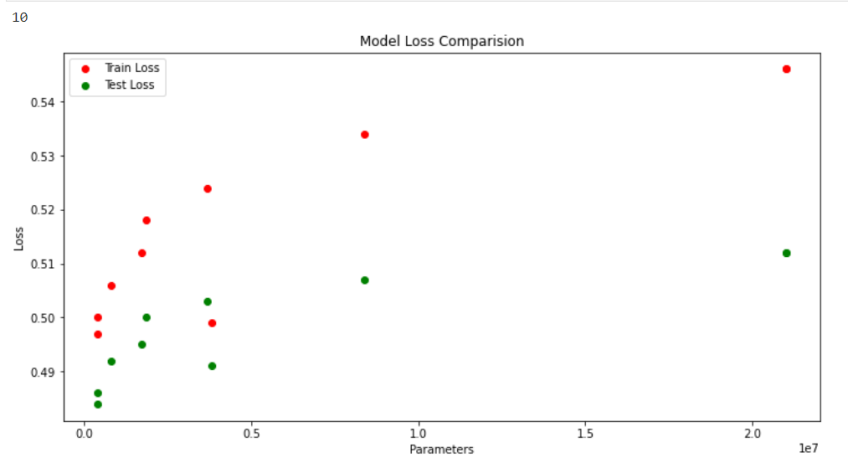
Loss function: nn.CrossEntropyLoss()

Optimization function: SGD

Epochs= 100

I changed the number of parameters, inputs, outputs rates and created 10 similar models.

****

****

We can see from the graphs that as the numbers of parameters are increased, train loss and test loss are decreasing and accuracy is increasing. But the difference between training and test points increased. This is due to the over fitting of the model for the data. It is a good thing that as the number of parameters is increasing the loss is decreasing and accuracy is increasing. But a good model should have less difference between train and test loss/accuracy.

**1-3 Flatness vs Generalization**

**Part 1:**

**Interpolation ratio**

For this part I took 2 similar models with different learning rate.

Model 1:

* 2d convolutional layer apply max pool(relu)
* 2d convolutional layer apply max pool(relu)
* 2d convolutional layer apply max pool(relu)
* 2d dense layer ->relu
* 2d dense layer ->relu
* 2d dense layer-> dropout
* 2d dense layer-> dropout
* 2d dense layer

Number of parameters: 581866

Learning rate: 0.001

Weight\_decay: 0.005

Batch size: 64

Loss function: nn.CrossEntropyLoss()

Optimization function: SGD

Epochs= 100

Model 2:

* 2d convolutional layer apply max pool(relu)
* 2d convolutional layer apply max pool(relu)
* 2d convolutional layer apply max pool(relu)
* 2d dense layer ->relu
* 2d dense layer ->relu
* 2d dense layer-> dropout
* 2d dense layer-> dropout
* 2d dense layer

Number of parameters: 581866

**Hyper parameters:**

Learning rate: 0.0001

Weight\_decay: 0.005

Loss function: nn.CrossEntropyLoss()

Optimization function: SGD

Epochs= 100

To find the interpolation ratio:

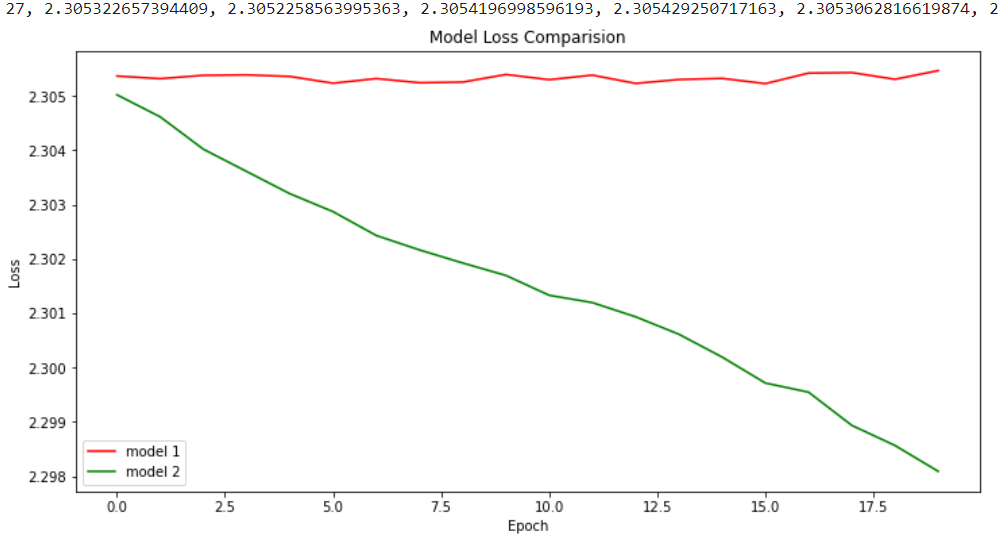
1. train two models, record number of parameters of both the models.

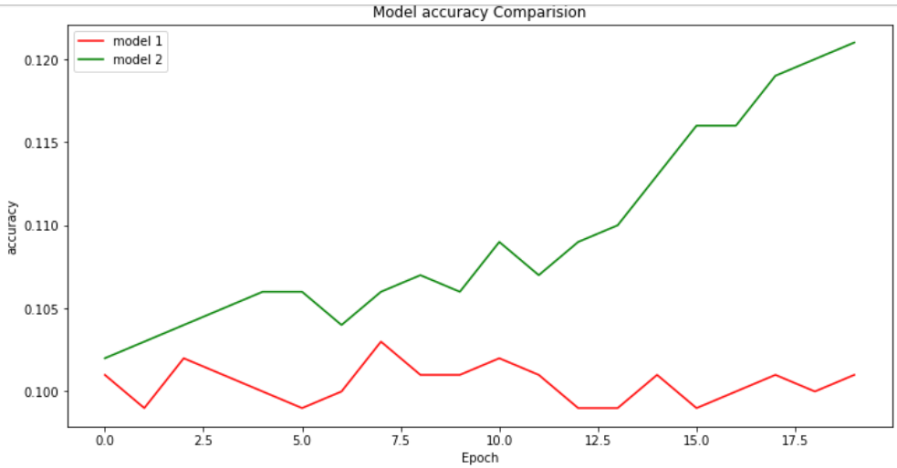
2. We then use the formula

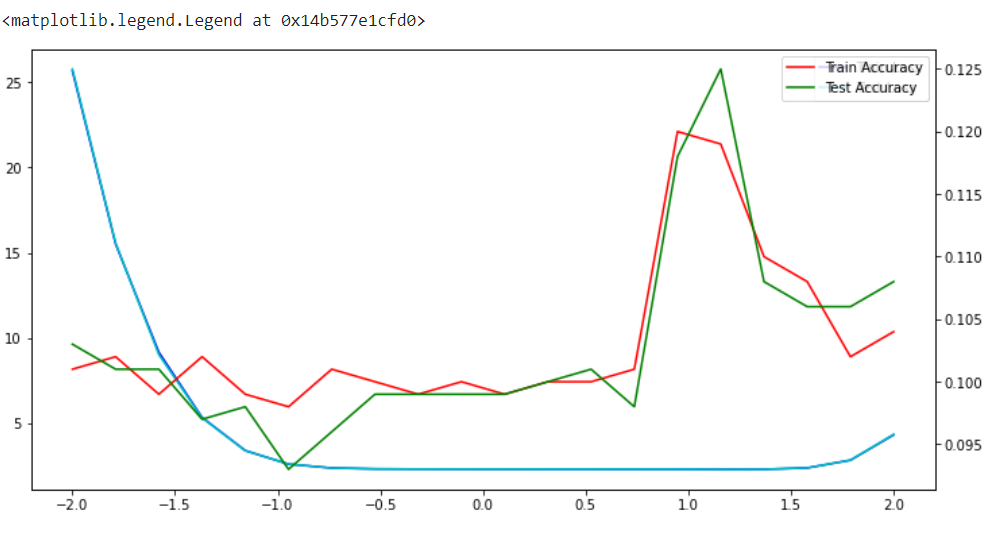
theta = (1-alpha[i])\*m1\_param + alpha[i]\* m2\_param to get the interpolation ratio.

3. We then get a new model using the theta value.

4. record loss and accuracy of the new model.





****

Interpolation is like guessing with the data or knowledge we have. From the graph we can see that both test loss and train loss decrased but then increased after the alpha value of 1.5. From the graph we can say that alpha value between 1.0 and 1.5 are ideal.

**Part 2:**

**Sensitivity**

* 2d convolutional layer apply max pool(relu)
* 2d convolutional layer apply max pool(relu)
* 2d convolutional layer apply max pool(relu)
* 2d dense layer ->relu
* 2d dense layer ->relu
* 2d dense layer-> dropout
* 2d dense layer-> dropout
* 2d dense layer

**Hyper parameters:**

Learning rate: 0.0001

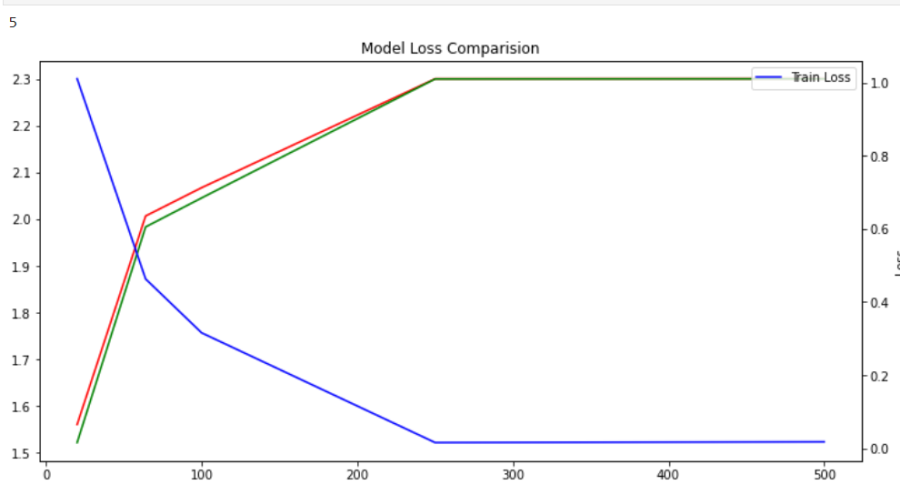
Weight\_decay: 0.005

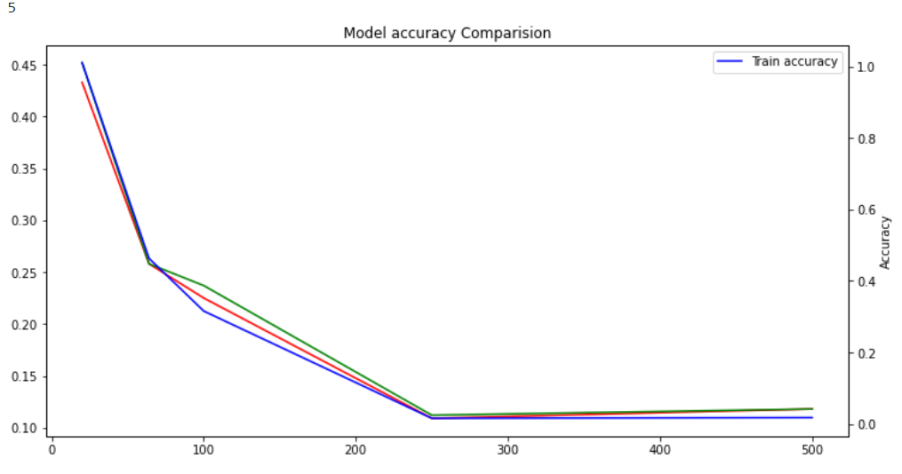
Loss function: nn.CrossEntropyLoss()

Optimization function: SGD

Epochs= 100

Then I created 5 different models with five batch sizes such as 100, 250, 500, 5000, 10000. Then find sensitivity of all the models and plot batch size vs train loss/test loss/sensitivity And batch size vs train accuracy/test accuracy/sensitivity

****

****

From the graphs we can say that sensitivity decreases with the increase in batch size. At some point between 0 and 100(i.e.,64) is an optimal batch size, where sensitivity started decreasing and there is a significant increase in accuracy and decrease in loss at that point.