1.1

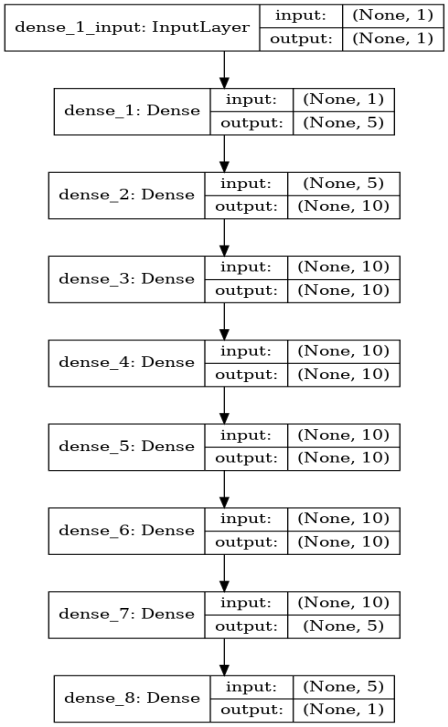
This part is solved by the two files:

First_Funtion_WithThreeModels.ipynb

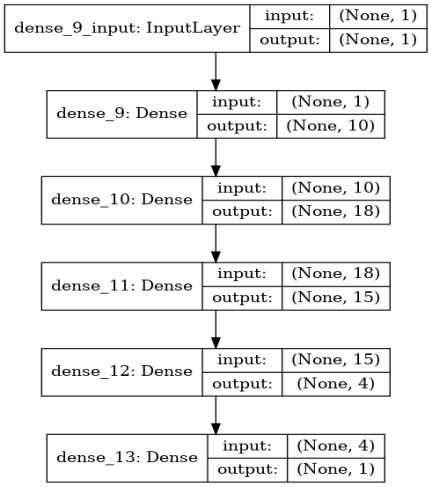
Second_Funtion_WithThreeModels.ipynb

- **Simulate a Function:**
    - **Describe the models you use, including the number of parameters (at least two models) and the function you use.**

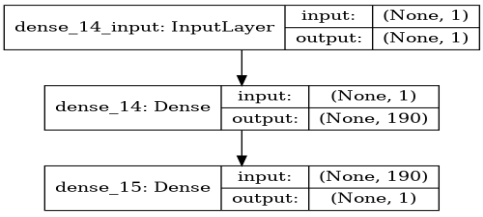
The shape and the number of parameters of the three models I used for this problem are listed below.



Model1, parameters number: 571



Model2, parameters number: 572



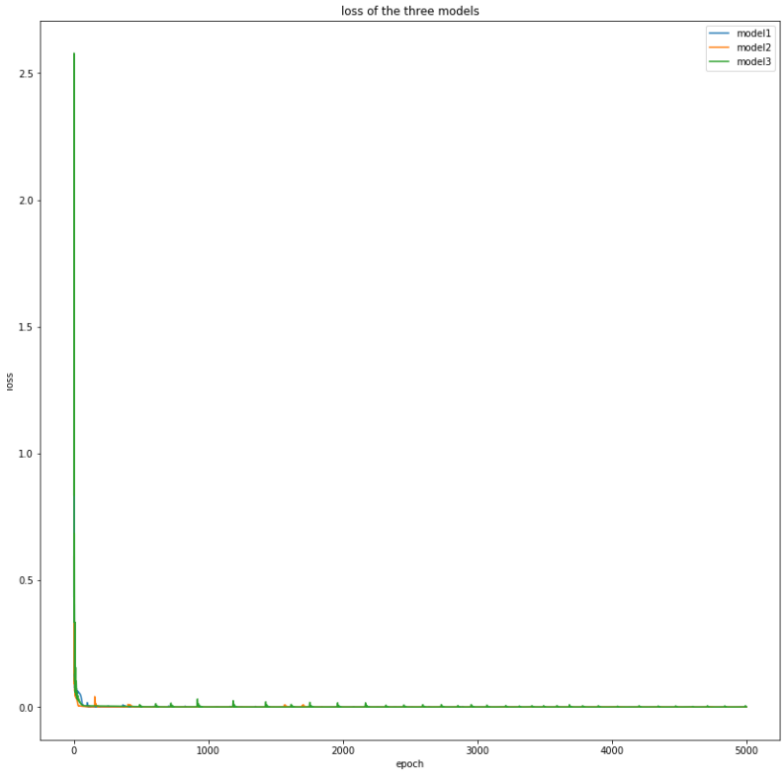
Model3, parameters number: 571

The first function I used for simulation is y = sin(pi x)/(pi x) + sin(0.5x).

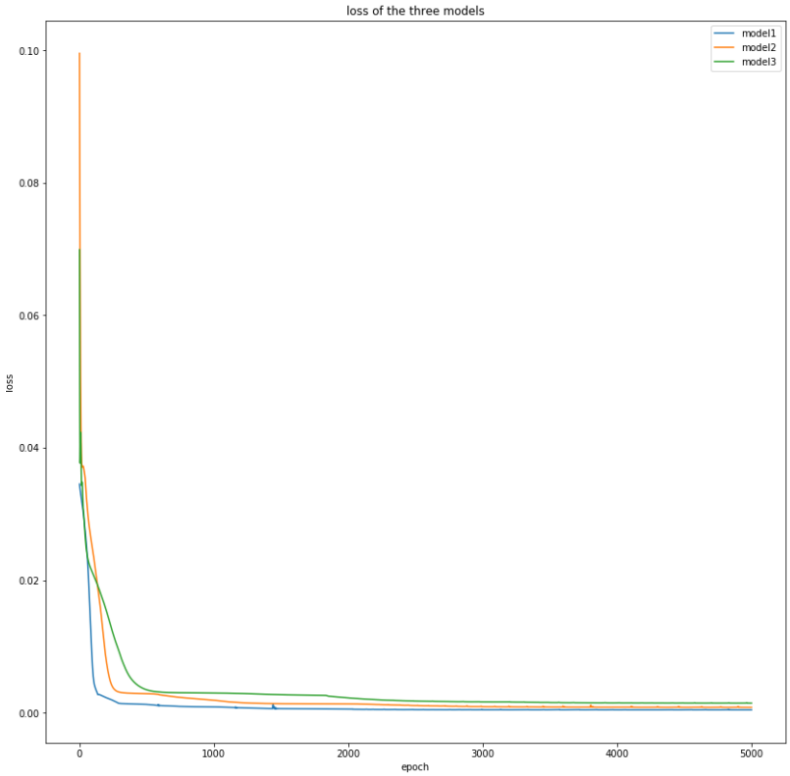
The second function I used for simulation is y = sin(pi x)/(pi x) + 0.7x^2.

- **In one chart, plot the training loss of all models.**

The loss of the models for the first function is shown below.

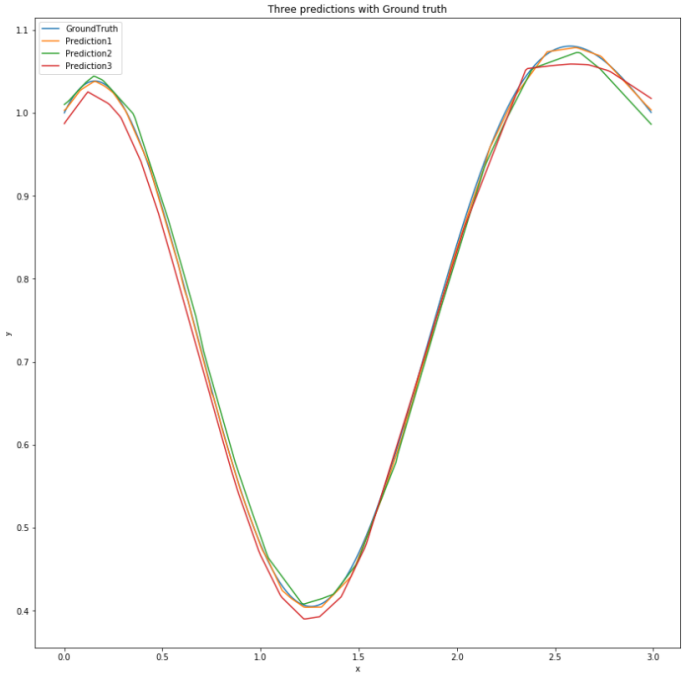


The loss of the models for the second function is shown below.

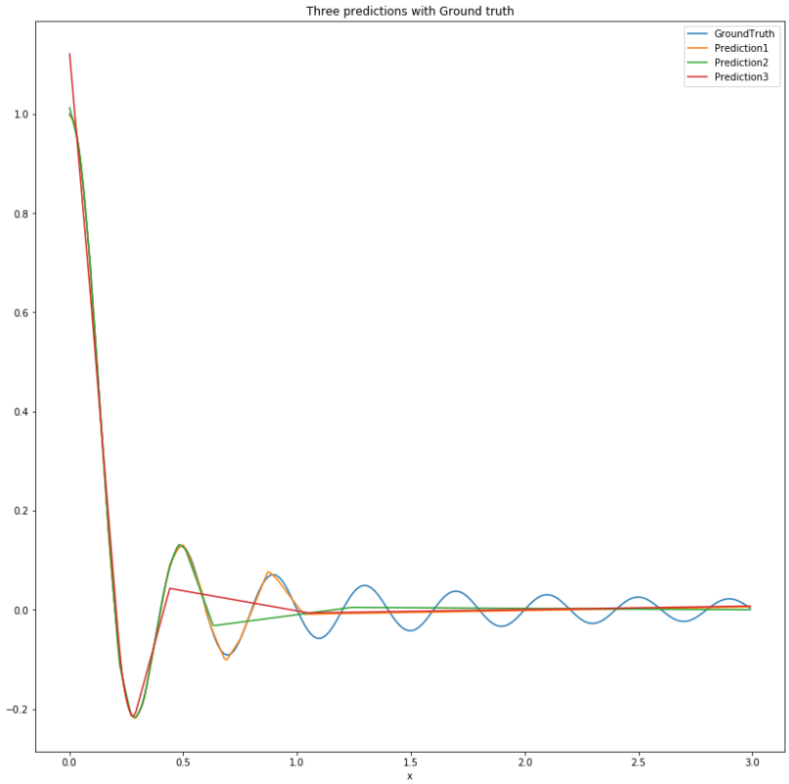


- **In one graph, plot the predicted function curve of all models and the ground-truth function curve.**

The predicted functions for the first function are shown below.



The predicted functions for the second function are shown below.



- **Comment on your results.**

We can see that the performance of the first model is the best among the three models. The first one is the model with the largest number of layers. So we can see that if the parameters cannot be updated with enough layers, the performance of the model will not be good, even the number of parameters are very big.

1.2

- **Train on Actual Tasks:**

This part is solved by the file:

CNN.ipynb

- **Describe the models you use and the task you chose.**

Model 1:

The model I used is a CNN model with two convolutional layers. For the first one: filters=16, kernel_size=5, activation=tf.nn.relu.

And for the second one: filters=36, kernel_size=5, activation=tf.nn.relu.

Model 2:

CNN with two convolutional layers. For the first one: filters=16, kernel_size=5, activation=tf.nn.relu.

And for the second one: filters=64, kernel_size=5, activation=tf.nn.relu.

Model 3:

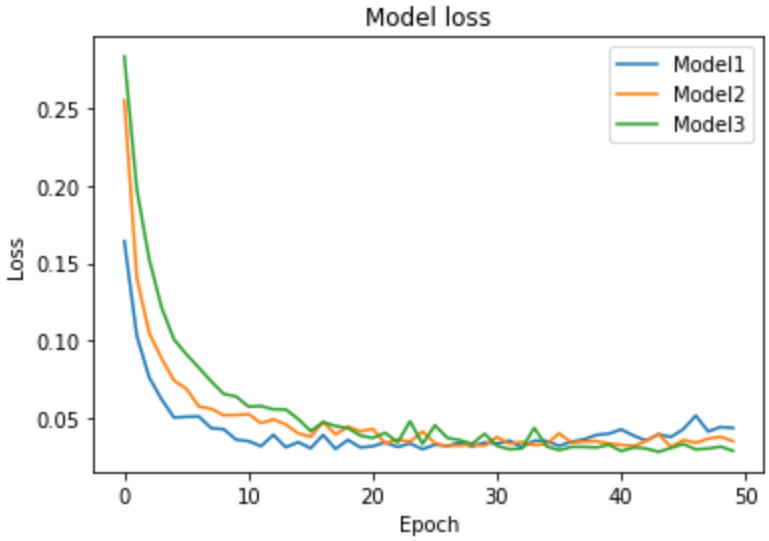
CNN with three convolutional layers. For the first layer: filters=16, kernel_size=5, activation=tf.nn.relu.

The second layer: filters=16, kernel_size=5, activation=tf.nn.relu.

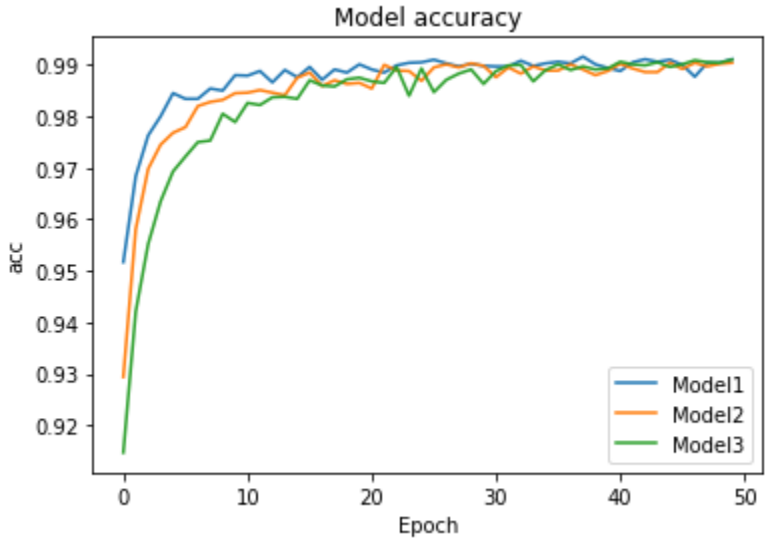
And the third layer: filters=16, kernel_size=5, activation=tf.nn.relu.

The task is MNIST.

- **In one chart, plot the training loss of all models.**



- **In one chart, plot the training accuracy.**



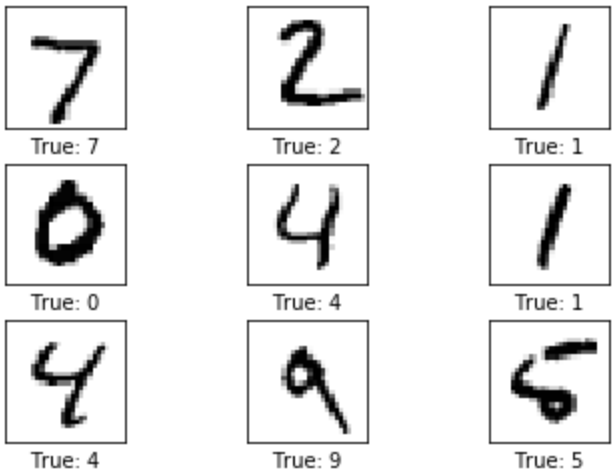
- **Comment on your results.**

We can see different CNN models with different architectures have different performance on MNIST dataset. However, we can see that the all the three CNN models have good performance on this task because that the three accuracies are all very high. So we can know that CNN can perform well in MNIST dataset.

2

This problem is solved by the file:

CNN_Mnist.ipynb



- **Visualize the optimization process.**
    - **Describe your experiment settings. (The cycle you record the model parameters, optimizer, dimension reduction method, etc.)**

The cycle: Collect the weights every 3 epochs, and train 8 times.

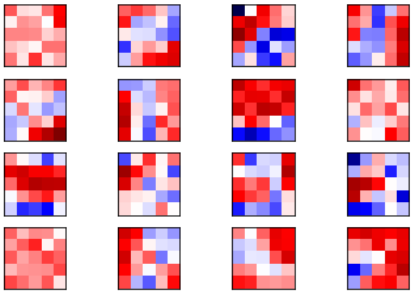
Optimizer: I was using Adam Optimizer with a learning rate of 0.01.

Reducing method: Reduce the dimension of weights to 2 by PCA.

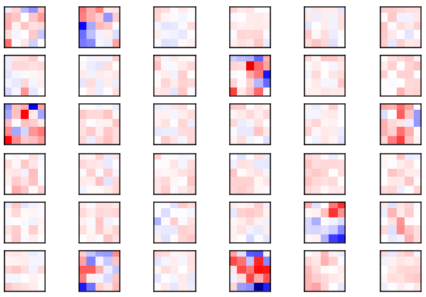
- **Train the model for 8 times, selecting the parameters of any one layer and whole model and plot them on the figures separately.**

The weights visualization:

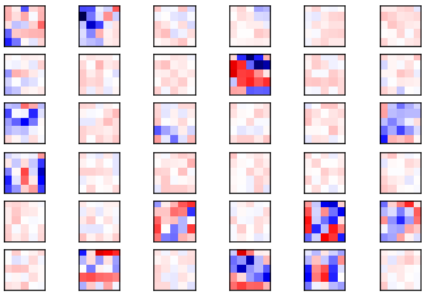
In [89]:
```
plot_conv_weights(weights=weights_conv1)
```



In [90]:
```
plot_conv_weights(weights=weights_conv2, input_channel=0)
```



In [91]:
```
plot_conv_weights(weights=weights_conv2, input_channel=1)
```

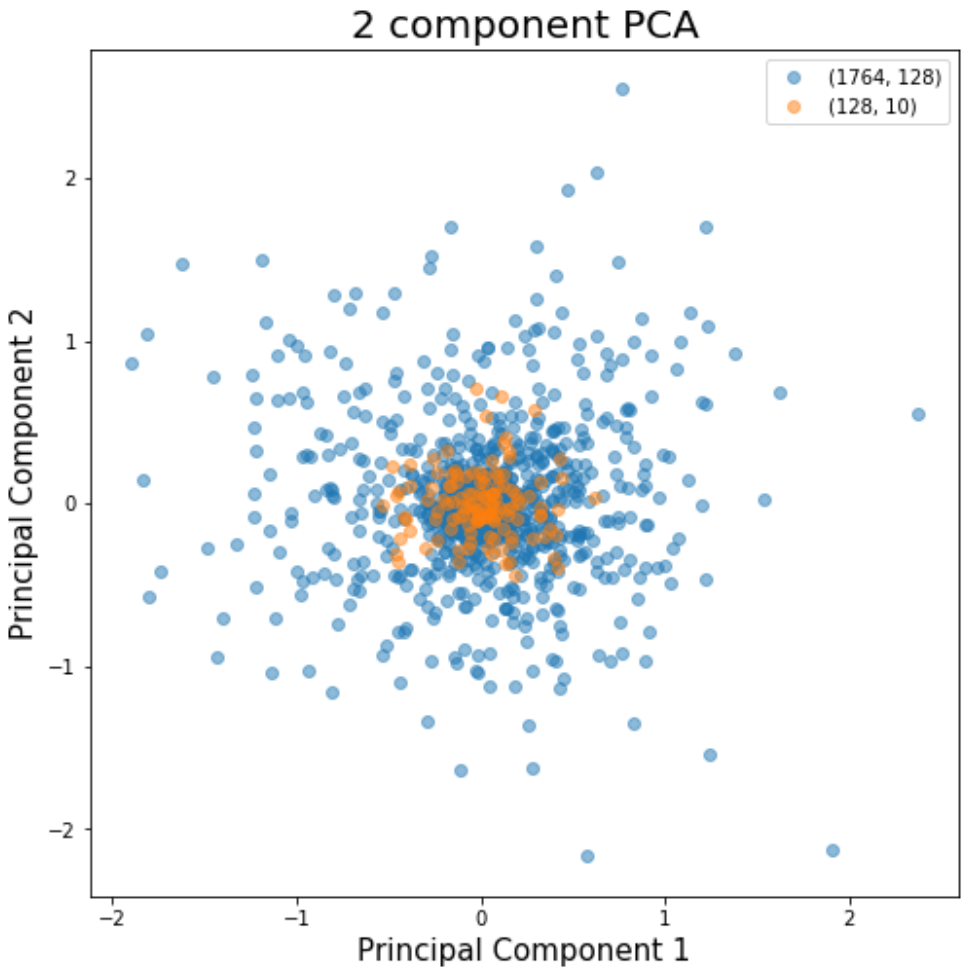


This is the weights when using Adam Optimizer.

PCA:

```
In [92]: plot_fc_weights(weights_list=[weights_fc1, weights_fc_out])
```

```
(1764, 128)
(128, 10)
```

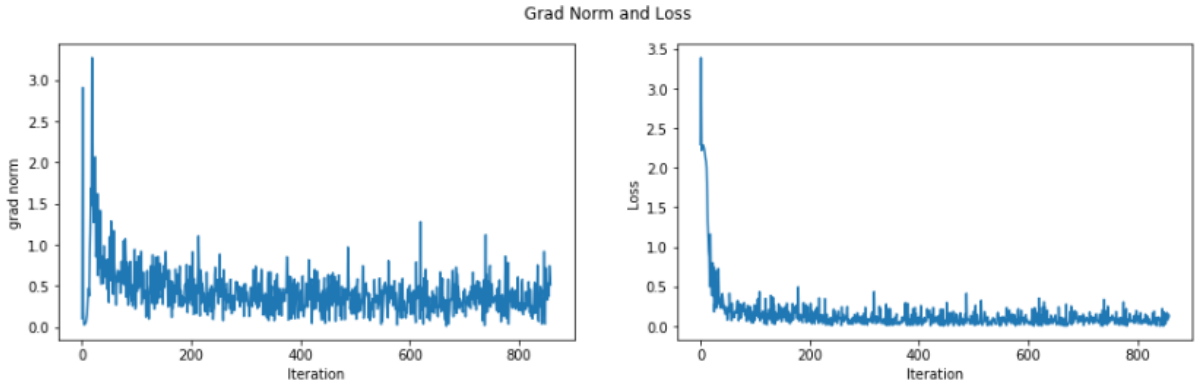


This is the PCA visualization when using Adam Optimizer.

- **Comment on your result.**

We can see the weights changing during the training for each of the three layers. We can see the filters number and the convolution processes during training. And for the reduction of the dimension of weights to 2, we can see that PCA can deduct the dimension of the weights to 2.

- **Observe gradient norm during training.**
  - **Plot one figure which contain gradient norm to iterations and the loss to iterations.**



- Comment your result.

We can see the change of the gradient norm with the number of iterations. The gradient is decreasing seeing from the whole image. And the gradient norm was getting zero sometimes. We can find out the changing processes of the gradient norm along the training steps.

- **What happens when gradient is almost zero?**
    - **State how you get the weight which gradient norm is zero and how you define the minimal ratio.**

How to get the weight which gradient norm is zero:

First, train the network with original loss function. And then change the objective function to gradient norm and keep training.

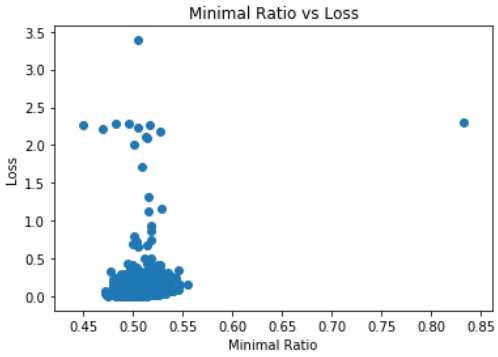
How to get minimal ratio:

Compute $H(L(\theta_{norm=0}))$ (hessian matrix), and then find its eigenvalues. The proportion of the eigenvalues which are greater than zero is the minimal ratio.

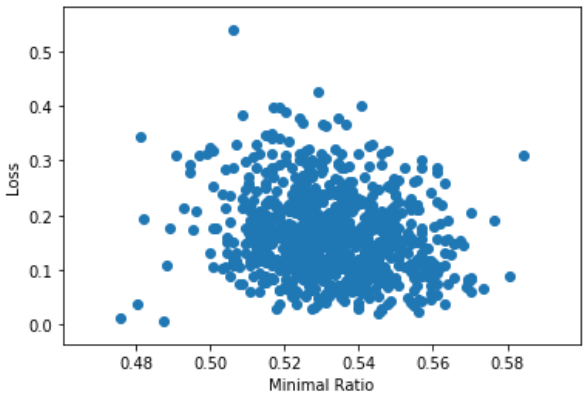
- **Train the model for 100 times. Plot the figure of minimal ratio to the loss.**

When using normal object function for training:

```
Out[96]: Text(0.5, 1.0, 'Minimal Ratio vs Loss')
```

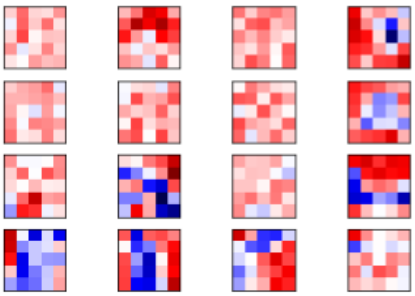


When using gradient norm as object function for training:

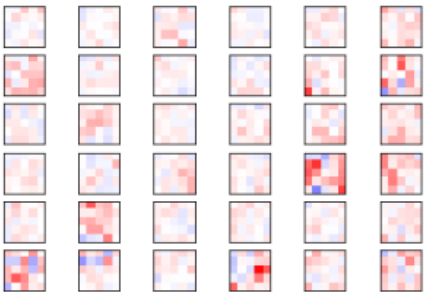


And the weights when gradient norm is zero (as small as possible):

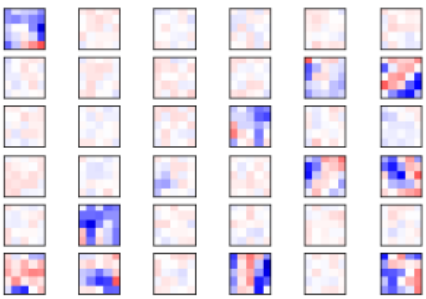
In [20]:
```
plot_conv_weights(weights=weights_conv1)
```



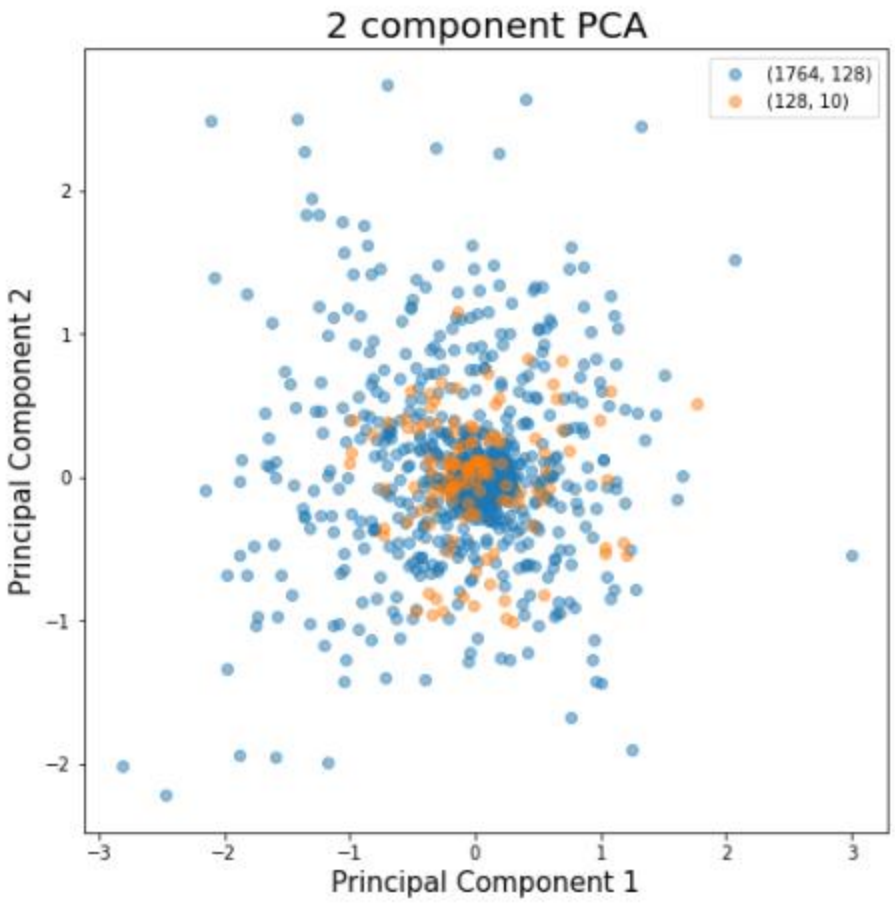
In [21]:
```
plot_conv_weights(weights=weights_conv2, input_channel=0)
```



In [22]:
```
plot_conv_weights(weights=weights_conv2, input_channel=1)
```



And the PCA at this time:



- **Comment your result.**

We can find the zero-gradient norm by change objective function. And Minimal ratio is defined as the proportion of eigenvalues greater than zero. When the gradient norm is zero, the gradient is reaching the local minimum or the local maximum, and at this time, the optimizer will try to jump out from the local minimum and heading to the global minimum, finding out the final weights.

3

3.1

This problem is solved by file:

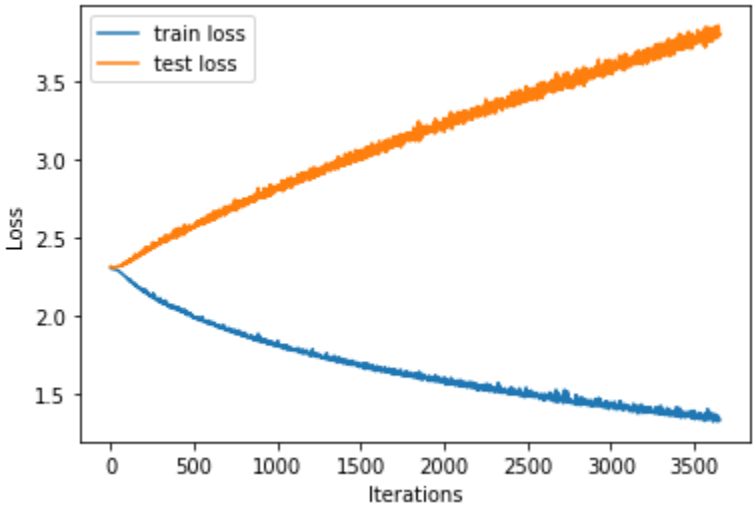
fit_random_labels.ipynb

- **Can network fit random labels?**
    - **Describe your settings of the experiments. (e.g. which task, learning rate, optimizer)**

The task I was using is MNIST dataset. And a two-layers CNN model is applied for this problem.

The optimizer I was using is Adam Optimizer with a learning rate of 0.001.

- **Plot the figure of the relationship between training and testing, loss and epochs.**



We can see the testing loss and the training loss during the training process. Because the model we are training is using the wrong dataset with wrong labels, the model cannot predict the testing set correctly. We can see that the model after training can fit the training set but cannot fit the testing set.

3.2

This problem is solved by file:

Number_of_parameters_vs_Generalization.ipynb

- **Number of parameters v.s. Generalization**
    - **Describe your settings of the experiments. (e.g. which task, the 10 or more structures you choose)**

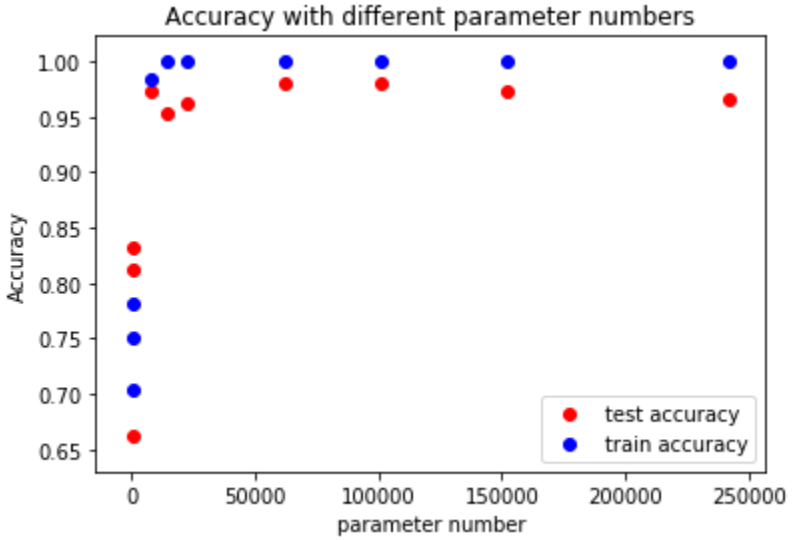
The task is MNIST dataset.

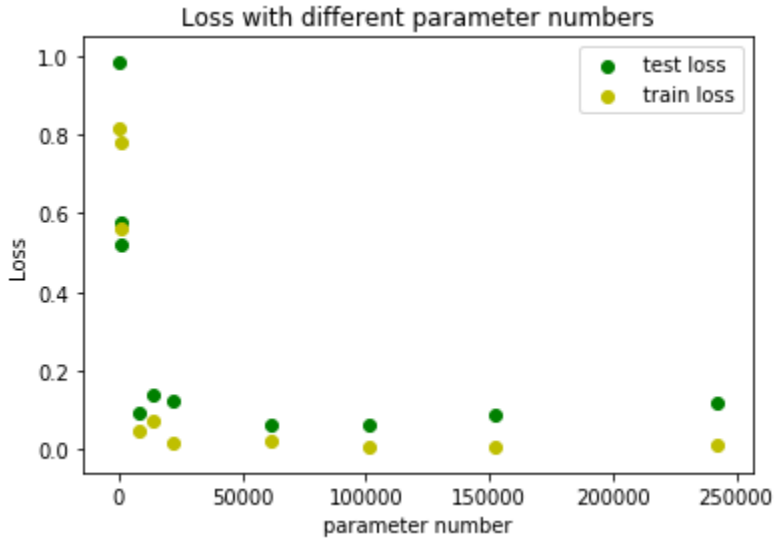
10 models:

All the ten models are CNN models, and they have different number of parameters.

1. 1-layer CNN with 152102 parameters.
2. 1-layer CNN with 750 parameters.
3. 1-layer CNN with 236 parameters.
4. 1-layer CNN with 14014 parameters.
5. 1-layer CNN with 494 parameters.
6. 1-layer CNN with 101274 parameters.
7. 2-layer CNN with 61982 parameters.
8. 1-layer CNN with 21098 parameters.
9. 2-layer CNN with 8028 parameters.
10. 2-layer CNN with 242062 parameters.

- Plot the figures of both training and testing, loss and accuracy to the number of parameters.





- **Comment your result.**

We can see that when the number of parameters is small, the larger the number of parameters is, the better the performance of the model will be. And when the number of parameters is large enough, the number of parameters will has minimal influence on the performance of the model. This is because when the model is complex enough, it is easy to get good weights for the problem, so the number of parameters will matter less. And, when the number of parameters is getting larger, the testing accuracy (or loss) and training accuracy (or loss) will have larger difference. This is because when the number of parameters of the model is getting larger, the model is more likely to overfit.

3.2

- **Flatness v.s. Generalization**
    - **Part 1:**

This problem is solved by file:

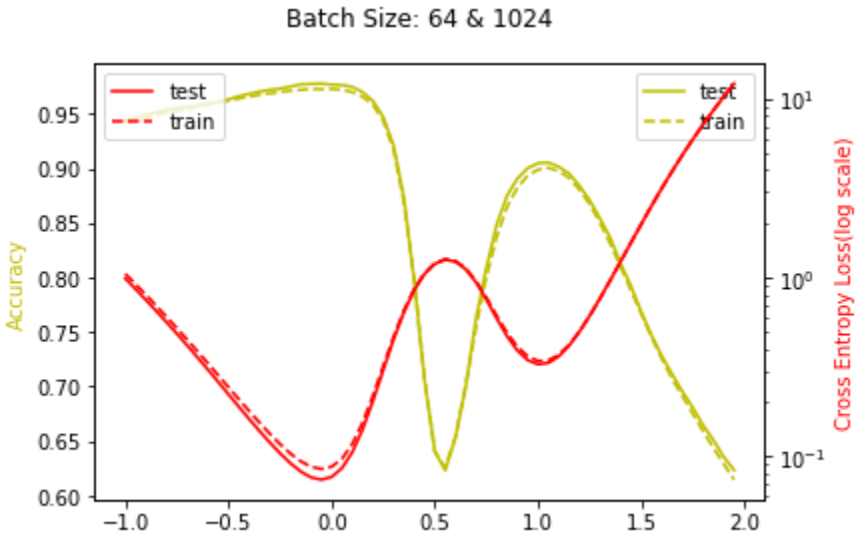
Flatness_vs_Generalization-part1.ipynb

- **Describe the settings of the experiments (e.g. which task, what training approaches)**

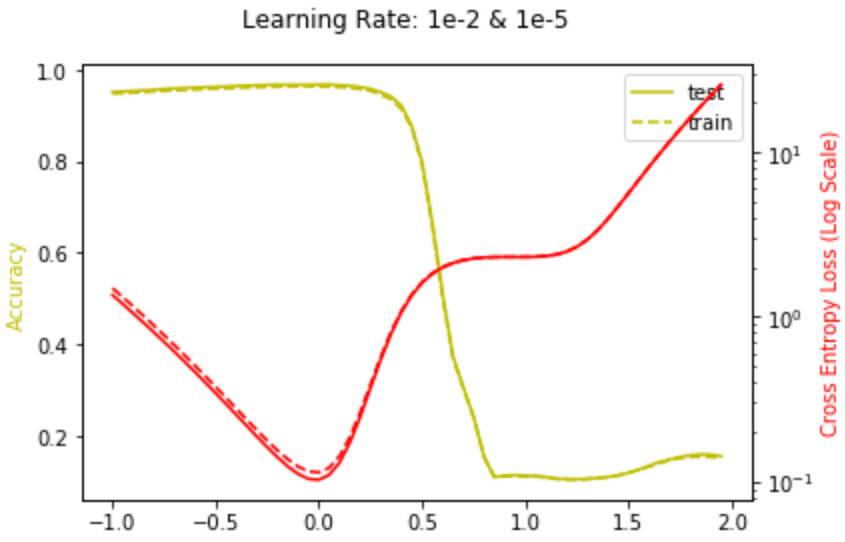
The task is MNIST dataset. The approach is using two CNN models with different settings, and then take out their weights and bias and store them. And in the third model, I combine the weights and bias with different proportions, getting the new weights and bias for the new model.

- **Plot the figures of both training and testing, loss and accuracy to the number of interpolation ratio.**

When using different batch size for the two models:



When using different learning rates for the two models:



- Comment your result.

We can see that when combining the two models' weights and bias, use them for a new training, we will still get the best performance with a certain proportion of the interpolation ratio. So we can find that the accuracy of our model will reach the best at some points, and the performance will be worse at some points. This is all depends on the interpolation ratio for combining the formal two model's weights and biases.

- **Part 2:**

This problem is solved by file:

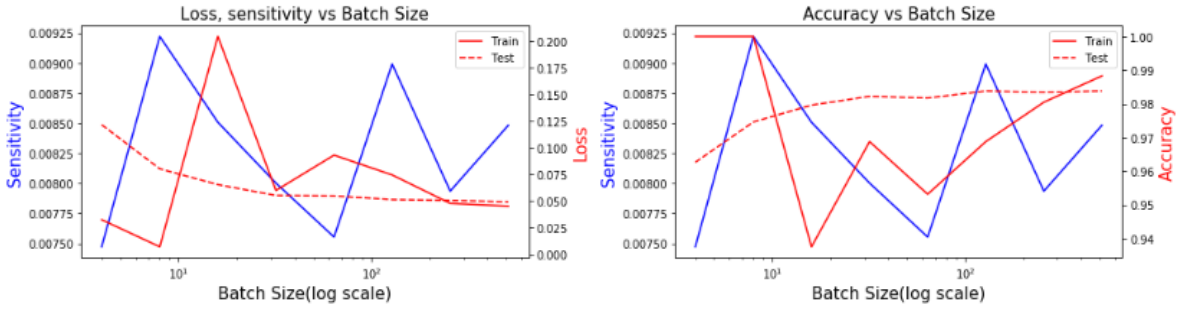
Flatness_vs_Generalization-part2.ipynb

- **Describe the settings of the experiments (e.g. which task, what training approaches)**

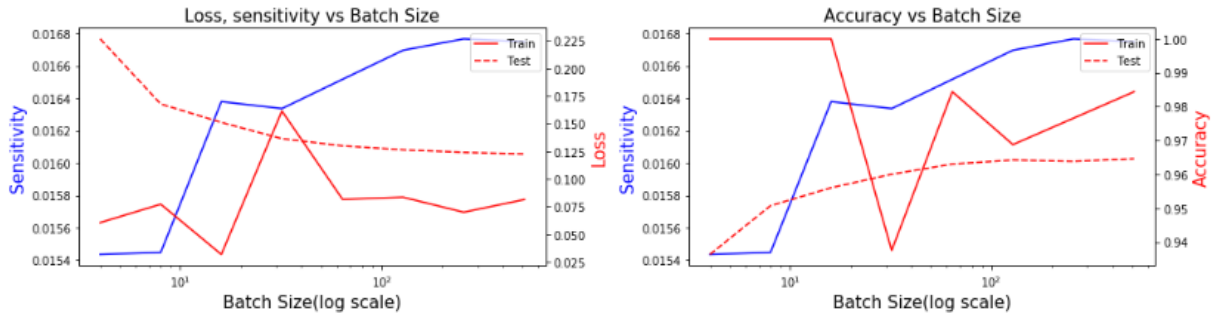
The task is MNIST dataset. I was using five models for this problem. The first three are using different batch size for comparison, and the last two are using different learning rates for comparison. The first three models are 1 CNN and 2 DNN models. And the two last models are 1 DNN and 1 CNN models.

- **Plot the figures of both training and testing, loss and accuracy, sensitivity to your chosen variable.**

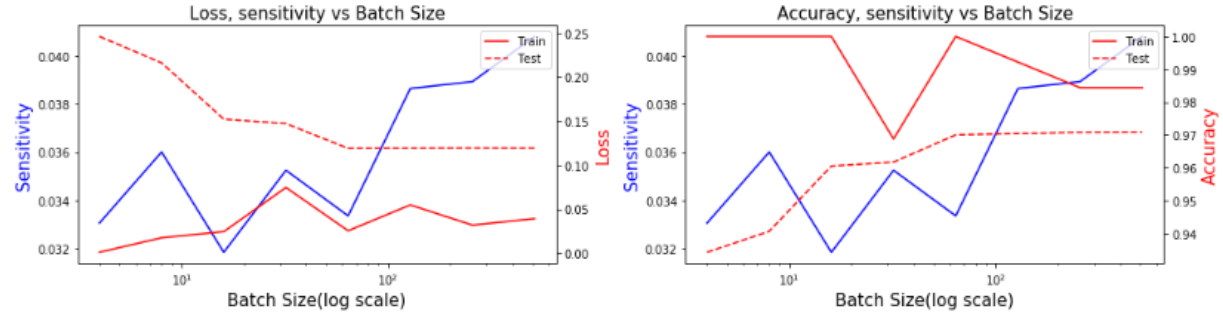
First mode:



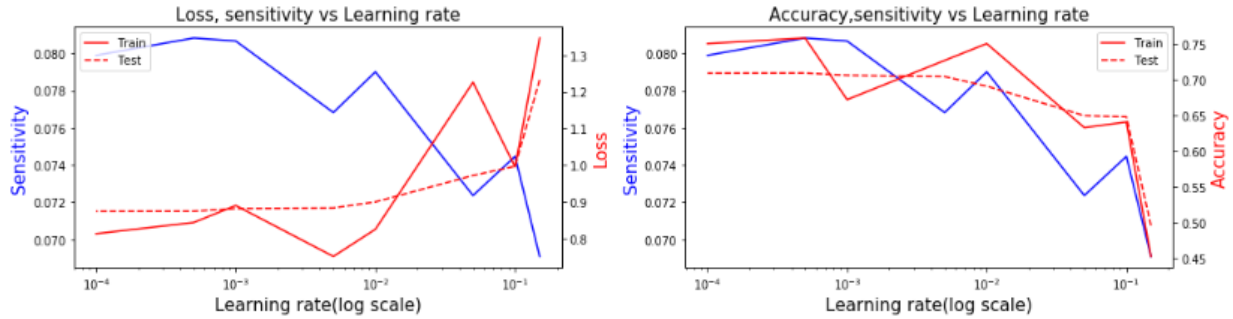
Second model:



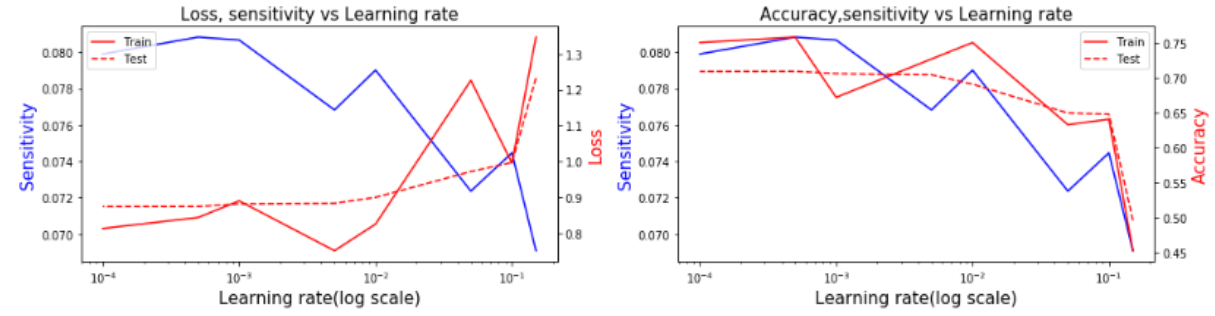
Third model:



Forth model:



Fifth model:



- **Comment your result.**

We can see different performance and sensitivity of the model with different batch sizes and learning rates. We can find out that the learning rate and the batch size has impacts on the performance and the sensitivity. There is a generalization gap when using large-batch (LB) methods (instead of small-batch (SB) methods) for training deep learning models.

The reasons (maybe more than these):

LB methods lack the explorative properties of SB methods and tend to zoom-in on the minimizer closest to the initial point.

SB and LB methods converge to qualitatively different minimizers with differing generalization properties (i.e. SB converges to flat minimizer, LB converges to sharp minimizer).