CPSC 8430 -Project1

Fangjian Li

February 2021

The codes are available at: https://github.com/FangjianLi/Deep-learning-Project1-Image-Classification-with-DNN-CNN

1 Deep VS Shallow

1.1 Simulate a function

Two DNN models with same number of parameters are built in this section to compare their performances. The codes are in the file *proj1-1 Simulated function_DNN.ipynb*

1.1.1 DNN models

Inspired by the homework slides, the two models used here are shown in Fig. 2. For hidden layers, the relu activation function is used for both models. The number of parameters of the models are obtained via counting *tf.trainable_variables()*. It is shown that the number of parameter is 571 for both of the models.

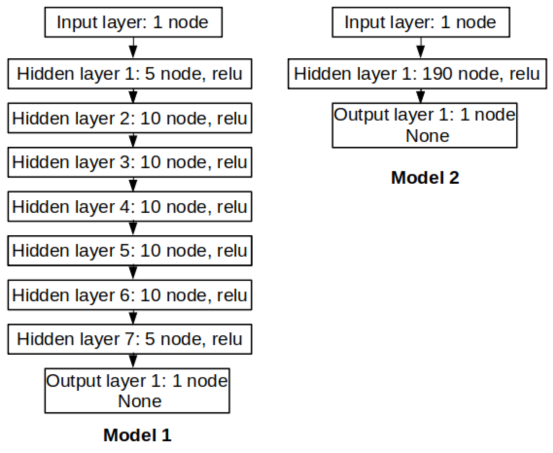

Input layer: 1 node
Hidden layer 1: 5 node, relu
Hidden layer 2: 10 node, relu
Hidden layer 3: 10 node, relu
Hidden layer 4: 10 node, relu
Hidden layer 5: 10 node, relu
Hidden layer 6: 10 node, relu
Hidden layer 7: 5 node, relu
Output layer 1: 1 node None
Model 1
Input layer: 1 node
Hidden layer 1: 190 node, relu
Output layer 1: 1 node None
Model 2

Figure 1: Two DNN models with same number of parameters

1.1.2 Training process

We choose a nonlinear function, $y = 2\cos(x) * x + 2$. The plot of function is shown in Fig. 2.

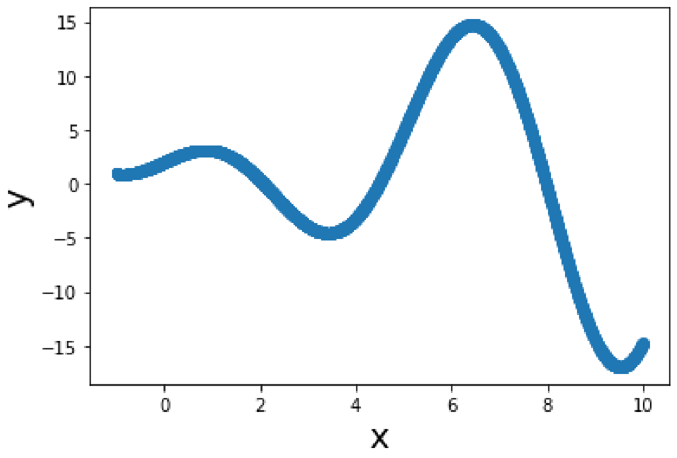


Figure 2: The function $y = 2\cos(x) * x + 2$

The configurations of the neural network training process are summarized in Table 2. It should be noticed that the training configuration of these two DNN models are same.

Table 1: DNN training configuration

Optimizer	Adam	Learning rate	0.005
# of epochs	2000	loss function	$MSE(y, y_{\text{pred}})$

where the loss function is defined as the mean squared error (MSE) between the ground-truth output y with the predicted y_{pred}.

1.1.3 Training results

The plot of loss versus training epochs is plotted in Fig. 3.

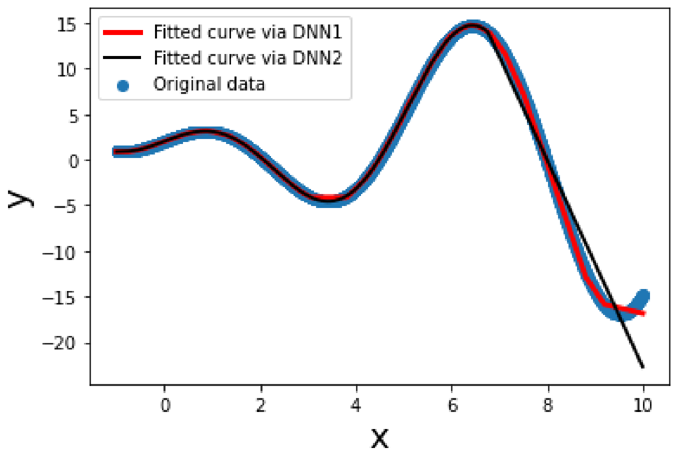


Figure 3: Loss versus training epochs

As can be seen, comparing to the DNN2 (shallow one), DNN1 (deeper one) has faster loss minimization. In addition, DNN1 can achieve lower loss. In the aspect of the loss minimization, the deeper model is better. The prediction performances of the two models are shown in Fig. 4.

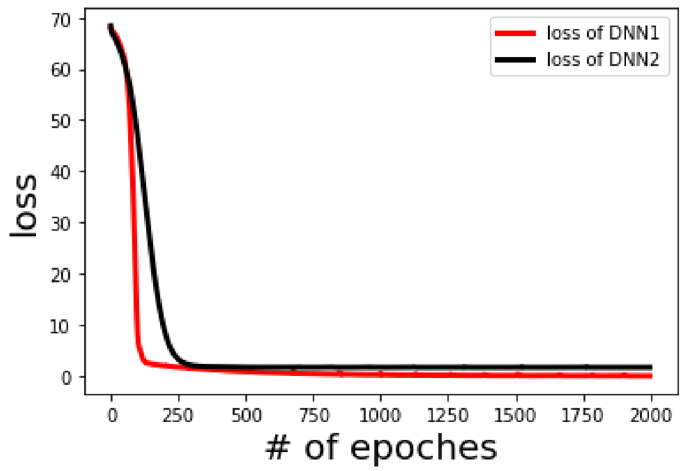


Figure 4: Prediction performance

As can be seen, comparing to the DNN2 (shallow one), DNN1 (deeper one) has better prediction performance. In conclusion, for this function simulation task, the deeper DNN can achieve better performance in both loss minimization and prediction.

1.2 Train on actual task

Corresponding to this requirement, we first use two DNN models and MNIST data set. The codes are in the file *proj1-1 MINST_DNN.ipynb*

1.2.1 DNN models

We use the similar DNN structures in section 1.1. The only difference is the number of nodes in each layers. The detailed structure is shown in Fig. 5. It is shown that the number of parameter is 280610 for model1 and 280645 for model2.

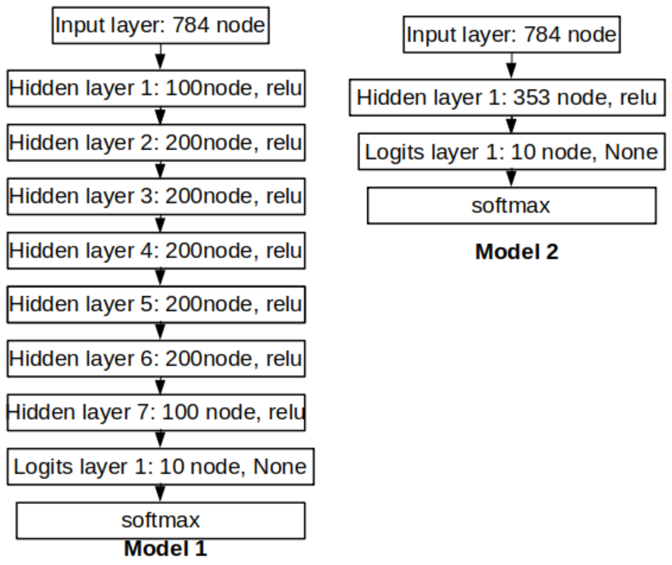

Input layer: 784 node
Hidden layer 1: 100node, relu
Hidden layer 2: 200node, relu
Hidden layer 3: 200node, relu
Hidden layer 4: 200node, relu
Hidden layer 5: 200node, relu
Hidden layer 6: 200node, relu
Hidden layer 7: 100 node, relu
Logits layer 1: 10 node, None
softmax
Model 1
Input layer: 784 node
Hidden layer 1: 353 node, relu
Logits layer 1: 10 node, None
softmax
Model 2

Figure 5: Two DNN models with similar number of parameters

1.2.2 Training process

The configurations of the neural network training process are summarized in Table 2. It should be noticed that the training configuration of these two CNN models are same.

Table 2: CNN training configuration

Optimizer	Adam	Learning rate	1e-4
# of epochs	5	loss function	cross entropy(y, y_{pred})
batch size	64	# of iteration	4297

where the loss function is defined as the cross entropy between the ground-truth label y with the predicted label y_{pred}. The accuracy is also defined as the the percentage that the DNN has the correct predicted labels.

1.2.3 Training results

The plot of loss versus training epochs is plotted in Fig. 6.

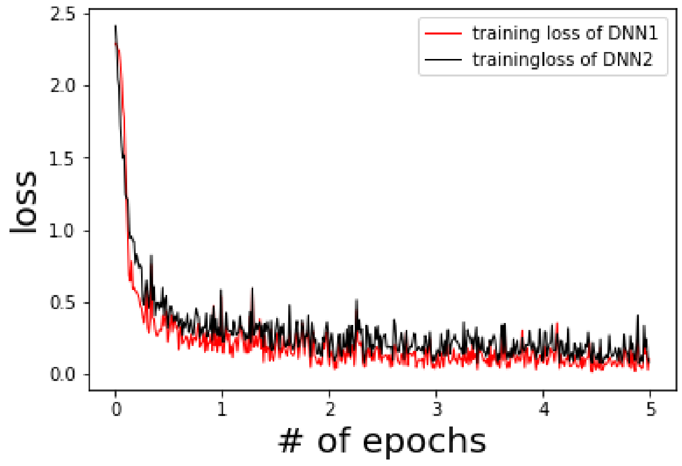
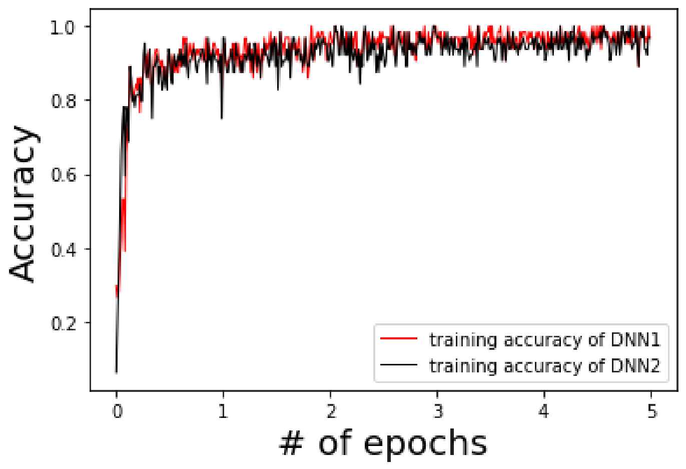
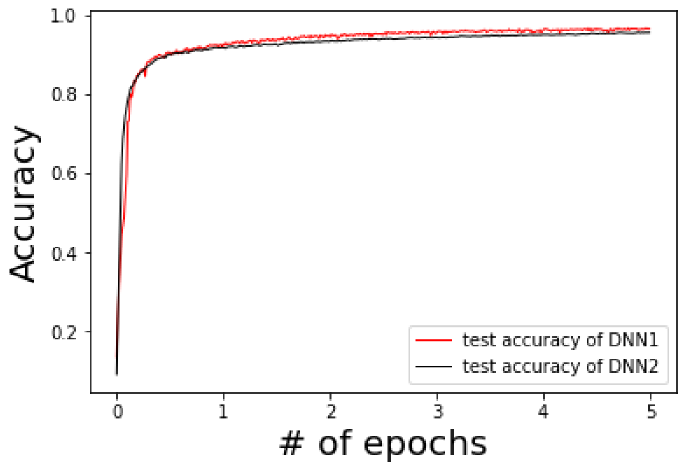


Figure 6: training Loss versus training epochs

As can be seen, comparing to the DNN2 (shallow one), DNN1 (deeper one) has faster loss minimization. In addition, DNN1 can achieve lower loss. In the aspect of the loss minimization, the deeper model is better. The training accuracy of the two models are shown in Fig. 7a.



(a) Training data accuracy



(b) Test data accuracy

Figure 7: Training accuracy of DNN

As can be seen, comparing to the DNN2 (shallow one), DNN1 (deeper one) has (slightly) lower progress in accuracy improvement. In conclusion, the difference between these two DNN models are not much.

Out of curiosity, the test data accuracy is also plotted in Fig. 7b. As can be seen, the test data accuracy has the smoother progress than the training data accuracy. The DNN1 (Deeper) is slightly better than DNN2 (shallow).

1.3 Train on actual task (Bonus)

The Two CNN models are also used here to train on the MNIST dataset. The codes are in the file: *proj1-1 MINST_CNN.ipynb*

1.3.1 CNN models

The structure of the models are shown in Fig. 8. As can be seen, comparing to model 1, the model 2 has one more convectional layer and one more max pooling layer. It is expected that the model 2 should have better performance.

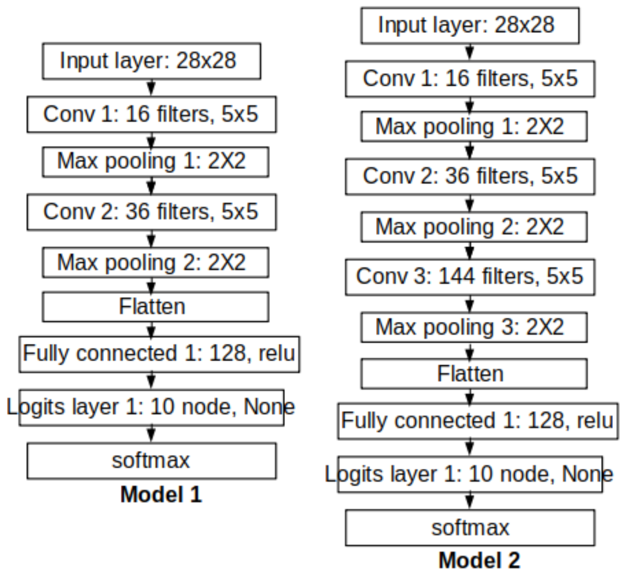


Figure 8: Two DNN models with similar number of parameters

1.3.2 Training process

The configurations of the neural network training process are summarized in Table 3. It should be noticed that the training configuration of these two CNN models are same.

Table 3: CNN training configuration

Optimizer	Adam	Learning rate	1e-4
# of epochs	5	loss function	cross entropy(y, y_{pred})
batch size	64	# of iteration	4297

where the loss function is defined as the cross entropy between the ground-truth label y with the predicted label y_{pred}. The accuracy is also defined as the the percentage that the CNN has the correct predicted labels.

1.3.3 Training results

The plot of loss versus training epochs is plotted in Fig. 9.

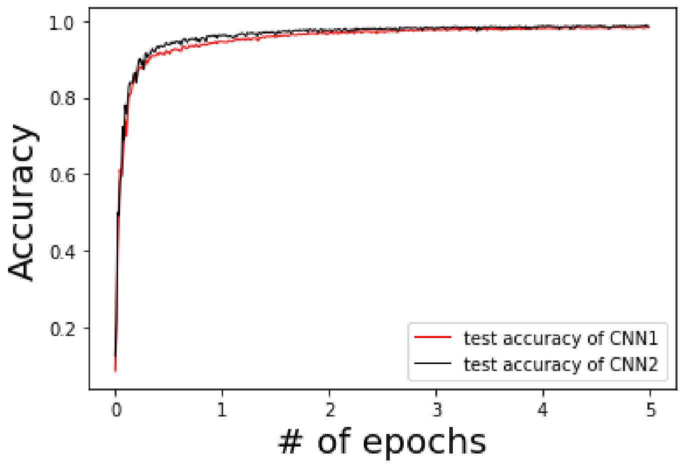
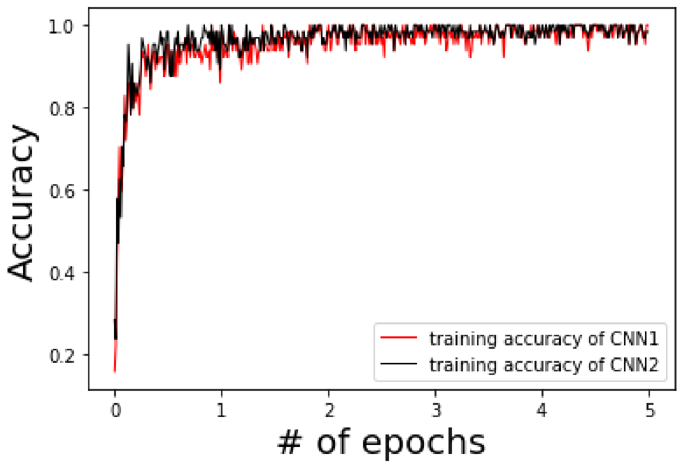
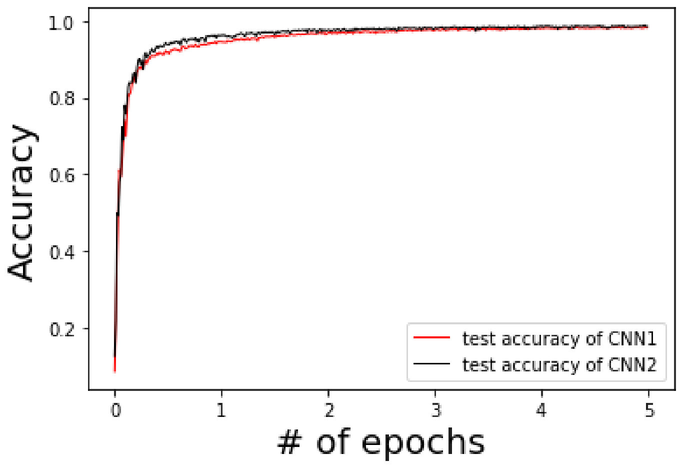


Figure 9: training Loss versus training epochs

As can be seen, comparing to the CNN1, CNN2 has faster loss minimization. In addition, CNN2 can achieve lower loss. In the aspect of the loss minimization, the CNN2 is better. The training accuracy of the two models are shown in Fig. 10a.



(a) Training data accuracy



(b) Test data accuracy

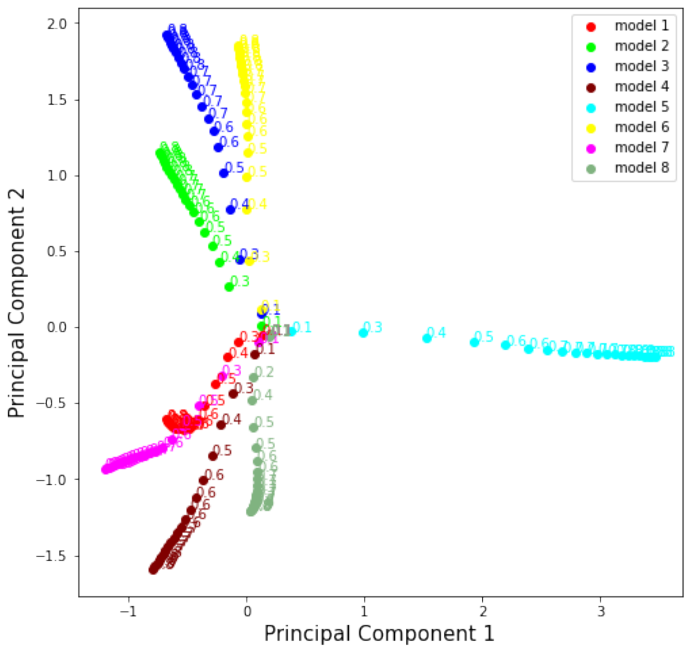
Figure 10: Training and test accuracy of CNN

As can be seen, comparing to the CNN1, CNN2 has (slightly) better accuracy than CNN1 in both training dataset and testing dataset. We can conclude that the deeper model (CNN2) has (slightly) better performance in loss minimization and prediction accuracy.

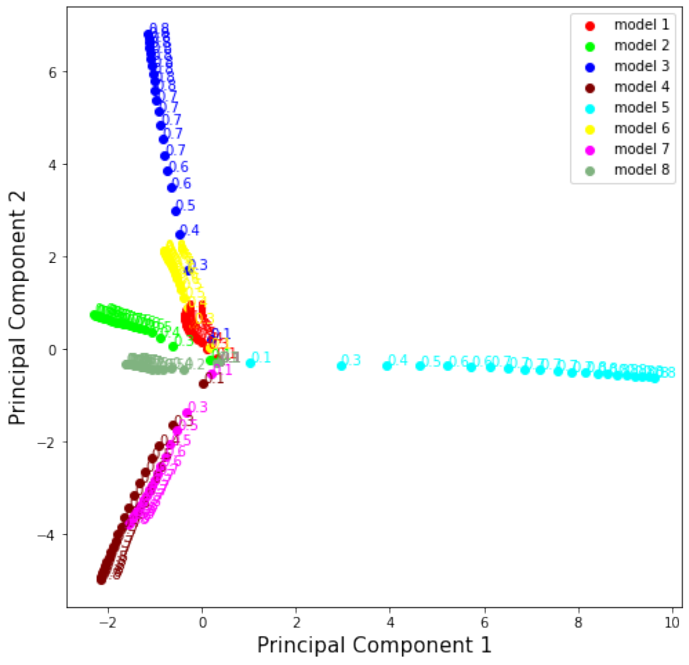
2 Optimization

2.1 Visualize the optimization process

The same DNN1 model from Section 1.2.1 is adopted in this task. The weights of the second hidden layer and the whole model have been recorded for every 3 epochs and conduct the training process for 8 times. Within each training event, we have 21 training epochs. Other training setups are same as Section 1.2.2. The PCA is used to analyze the weight updates with respect to the different training epochs and training events. The PCA plot for the weights of second hidden layer is shown in Fig. 11a. The PCA plot for the weights of the whole DNN model is shown in Fig. 11b. (The codes are in the file: *proj1-2 Weights PCA.ipynb*.)



(a) PCA for second layer weights



(b) PCA for the whole model

Figure 11: PCA analysis of the DNN model (test accuracy is annotated)

As can be seen, different training events of the same DNN model can generate different PCA weight plots. We can find that these 8 training events start from the similar PCA weight point and diverge to different directions as training goes on. In addition, the relative position of PCA plot for the individual layer weights is similar to the one for the whole model weights.

2.2 Observe gradient norm during training

A small DNN network is built to train with the simulated function shown in Fig. 4. The configuration is shown in Table. 4. The plot of gradient norm to iterations and the loss to iterations are shown in Fig. 12. (The codes are in *proj1-2 Optimization_gradient_function.ipynb.*)

Table 4: DNN training configuration

Hidden layer size	10x10x10	Activation function	relu
loss func before 3e4 iters	MSE	loss func after 3e4 iters	1e-5 gradient norm
Optimizer	Adam	Learning rate	1e-4
batch size	64	# of iteration	4297

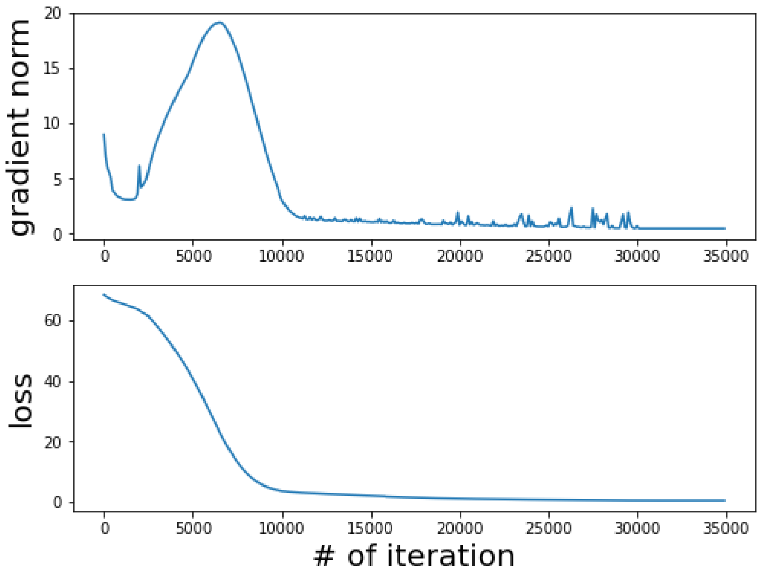


Figure 12: gradient norm to iterations and the loss to iterations

As can be seen from the above plots, the trends of the gradient norm and loss with respect to the number of iterations are different in the beginning. However, both the gradient norm and loss converge almost to zero in the end. Changing the loss function from MSEloss to gradient norm in the end can help to converge the gradient norm.

2.3 What happens when gradient is almost zero

As can be seen from the above plots, the gradient norm come to almost zero after 30000 iterations. In order to further minimize the gradient norm, at 30000 iteration, the loss function has been changed from MSEloss to the 1e-5*gradient norm. The Hessian matrix is first calculated based on the function in Tensorflow. The eigenvalues of the Hessian are found based on the function in Numpy. Minimal ratio can be calculated as the portion of the eigenvalues that are equal to zero. The *argmin* function is used to find the iteration with the smallest gradient norm in each training. The corresponding loss and minimal ratio are recorded for each training time. The plots of the loss versus minimal ratio for 100 training times is shown in Fig. 13. (The codes are in *proj1-2 Optimization_gradient_function.ipynb.*)

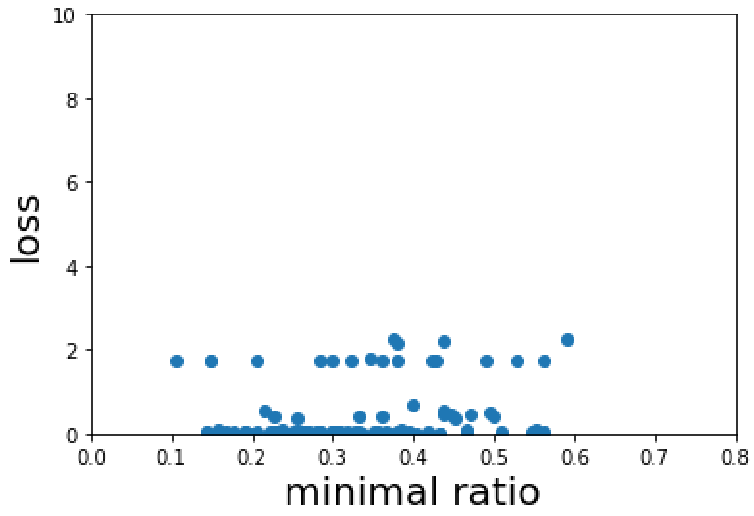


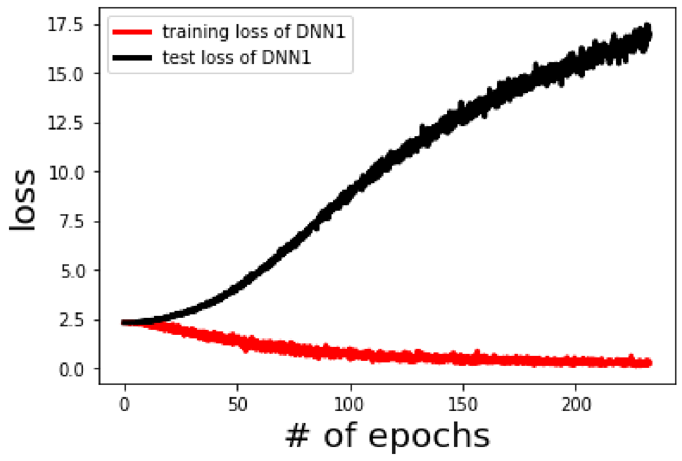
Figure 13: gradient norm to iterations and the loss to iterations

As can be seen, the minimal distribution of the data points are quite converged. The minimal ratio is around 0.2 to 0.6 and the loss is around 0 to 2. The loss is quite large in this case, that might because a shallow network is used (210 weights), the training performance itself is not good enough.

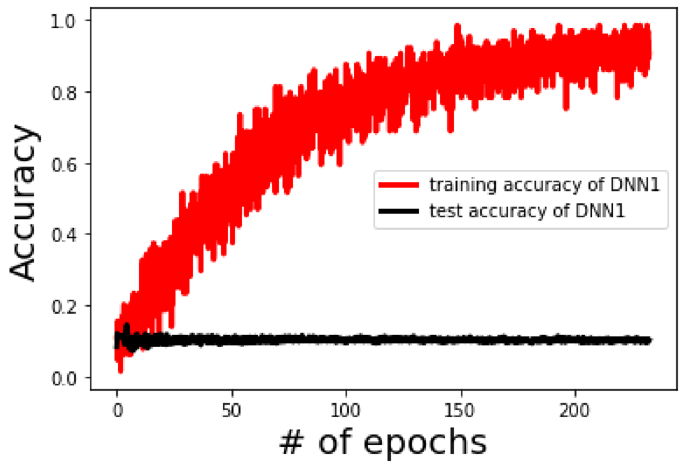
3 Generalization

3.1 Can network fit random labels?

The DNN1 structure used in Section 1.2.1 is used here to train the MINST data. The training process and setup are same as the model training in Section 1.2.2 except that the label in the training data set has been shuffled and the training epochs has been extended to 233. The results are shown in Fig. 14. (The codes are in the file *proj1-3_MINST_DNN_Shuffled_label.ipynb.*)



(a) Training and test loss



(b) Training and test accuracy

Figure 14: Training results of DNN with shuffled MINST labels

As can be seen from the above plots, the loss decreases and training accuracy increases in the training process. It shows that the DNN training algorithm works. However, the test loss drops a lot and test accuracy drops a little bit as the training goes on. It make senses as the rationality of the data is destroyed by the shuffled labels.

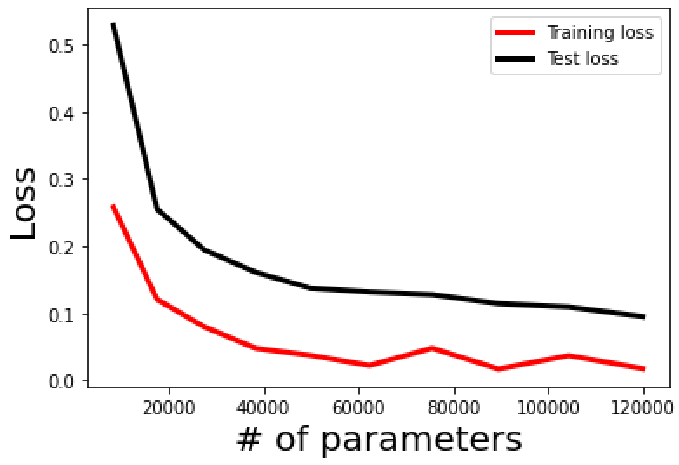
3.2 Number of parameters v.s. Generalization

Ten DNN models are built to train the DNN. The input layer, output layer and training process are same as the DNN in Section 1.2.2 except that the number of epoches has been extended to 9. These ten DNN models have save structure with 5 hidden layers and relu activation function. The only difference is the number of hidden nodes in hidden layers, which is shown in Table 5. (The codes are shown in file *proj1-3_MINST_DNN_Ten_Models.ipynb*.)

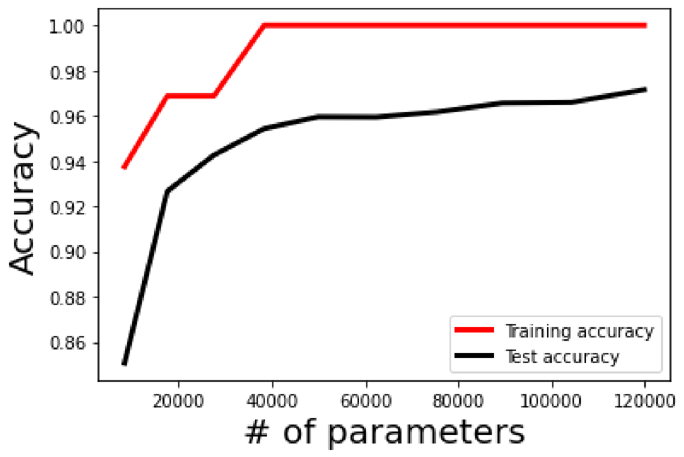
The training results with respect to the number of parameters are shown in Fig. 15

Table 5: Hidden layer configuration

DNN1	$(10,10,10,10,10)$	DNN2	$(20,20,20,20,20)$
DNN3	$(30,30,30,30,30)$	DNN4	$(40,40,40,40,40)$
DNN5	$(50,50,50,50,50)$	DNN6	$(60,60,60,60,60)$
DNN7	$(70,70,70,70,70)$	DNN6	$(80,80,80,80,80)$
DNN9	$(90,90,90,90,90)$	DNN10	$(100,100,100,100,100)$



(a) Training and test loss



(b) Training and test accuracy

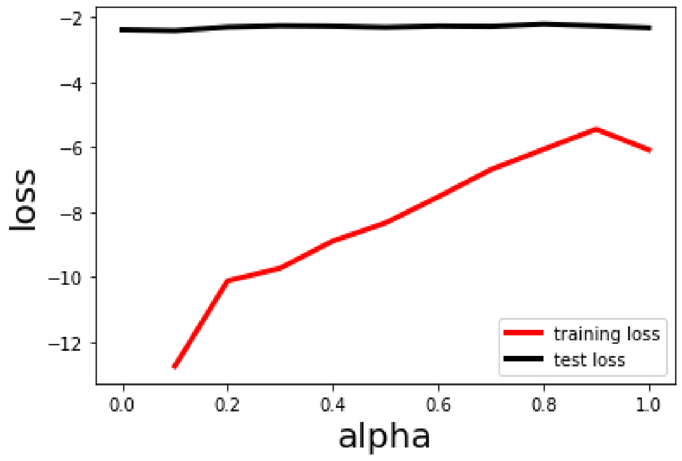
Figure 15: Training results versus the number of parameters

As can be seen from the above plots, in general, larger number of parameters in neural network can have better training performance in the aspects of both loss minimization and accuracy. However, the performance increments caused by the the increment of the parameter numbers becomes smaller.

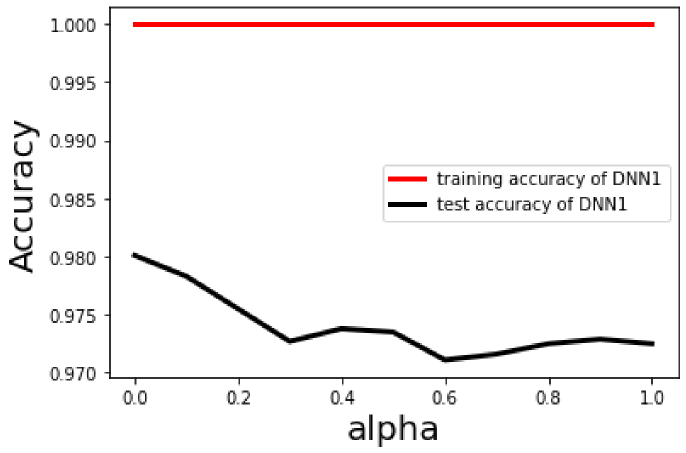
3.3 Flatness v.s. Generalization - part1

I use the built the model m1 and model m2 as same as DNN1 model in Section 1.2.1. They also share the same training configuration as Section 1.2.2 except the different batch size and the number of epochs has been extended to 100. The m1 has the batch size equal to 64. The m2 has the batch size equal to 1024. Eight more models have been built based on the linear interpolation between m1 and m2. The corresponding batch size is equal to $(1-\alpha)*64+\alpha*1024$, where

α is the linear interpolation ratio. The plots of loss and accuracy with respect to linear interpolation ratio is shown in Fig. 16. (The codes are shown in the file: *proj1-3_MINST_DNN_different batch.ipynb*.)



(a) Training and test loss (loss in log scale)



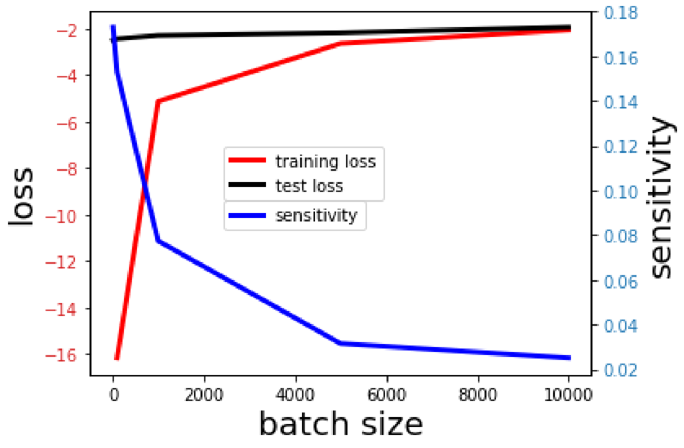
(b) Training and test accuracy

Figure 16: Training results versus the linear interpolation ratio

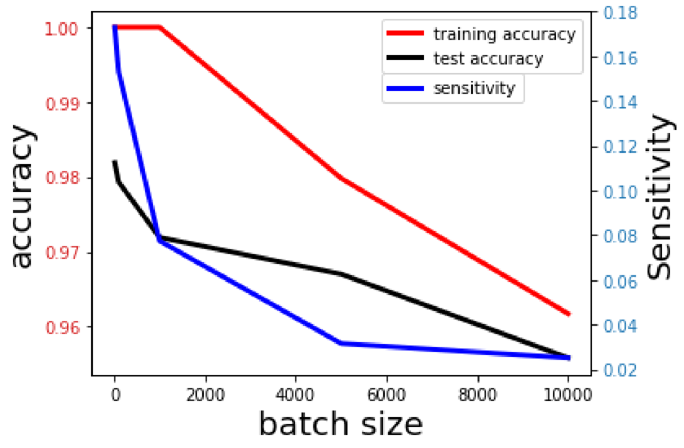
As can be seen from the above plots, the smaller batch can achieve better accuracy comparing to the larger batch. In addition, although the accuracy can become almost 1 for the training set, the accuracy for test set cannot be that high.

3.4 Flatness v.s. Generalization - part2

In this part, by using the same DNN1 model shown in Section 1.2.1, we change the batch size from 10 to 10000 to find its effects on loss, accuracy, and model sensitivity. The training process is same as Section 1.2.1, except that each model is trained for 100 epochs. The sensitivity of the model is defined as the Frobenius norm of gradients of loss to input. The results are shown in Fig. 17. (The codes are shown in the file: *proj1-3_MINST_DNN_sensitivity.ipynb*.)



(a) Training and test loss with sensitivity(loss in log scale)



(b) Training and test accuracy with sensitivity

Figure 17: Training results versus the batch size

As can be seen from the above plots, the smaller batch size can achieve better performance in the aspect of loss mimimization, accuracy, and sensitivity. The sensitivity of the model might be a good indication to show the performance of the model.