CSE 574 Assignment 2 Report

Team 66 members information:

Xian Zhou

Yuxuan Zhang

Shuyan Chen

1. Hyper-parameter in Neural Network

1.1 Neural network implementation

1.1.1 Feature selection

The classification models cannot get any difference information from the features that have the same values for all data. So we remove these features in the preprocess step using the following codes.

```
# Feature selection
# Your code here.
feature = []
for i in range(0, 784):
    feature.append(not (all(train_total[:, i] == train_total[0, i])))
feature = np.array(feature)
# We select the column that not all values the same
# feature is a boolean list used for data Feature selection
```

The indices of the features we use are stored in the ‘feature’. It is a boolean list (i.e, a list a True or False) used for data feature selection.

1.1.2 Feedforward Propagation

The feedward path is quite simple. There is only one hidden layer. We use w1 and the input vector (after feature selection, from 28*28 = 784 to 717) and the sigmoid activation function to get hidden node value. Then use similar method to get the output value. There are 10 output class, i.e., 10 digits from 0 to 9. We also need to add bias 1 to the input and hidden layer.

1.1.3 Error function and Backpropagation (Regularization)

The error function we use is the negative log-likelihood error function. We calculate the error for every training data and calculate the mean (i.e., divide n). We also use the p2 norm of parameters to avoid overfitting. The error calculation is the implementation of the following two equations:

$$J(W^{(1)}, W^{(2)}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{l=1}^{k}(y_{il}\ln o_{il} + (1-y_{il})\ln(1-o_{il}))$$

$$\tilde{J}(W^{(1)}, W^{(2)}) = J(W^{(1)}, W^{(2)}) + \frac{\lambda}{2n}\left(\sum_{j=1}^{m}\sum_{p=1}^{d+1}(w_{jp}^{(1)})^2 + \sum_{l=1}^{k}\sum_{j=1}^{m+1}(w_{lj}^{(2)})^2\right)$$

The y_{il} is the ground truth data, which will be a vector with length 10.

```
y_gt = np.zeros((10, 1), float)
y_gt[int(train_label[i, 0]), 0] = 1.0  # 1.0 means probability equals to 1.0
```

The gradient is the derivative of error function w.r.t all weights in w1 and w2. The gradient is calculated using chain rules. The gradient calculation is the implementation of equation (8), (9) and (12). The regularization term is like equation (16), (17). We use the original gradient descent for updating weights. We calculated all the gradient for all training data and calculate the mean (like the following equation). The weights update once after all training data are calculated.

$$\nabla J(W^{(1)}, W^{(2)}) = \frac{1}{n}\sum_{i=1}^{n}\nabla J_i(W^{(1)}, W^{(2)})$$

1.1.4 Neural network prediction

Use the w1 and w2 learned from the training data. Calculate the outputs using the neural network (the forward path). Choose the one with the highest probability as the predicted class label.

1.2 Hyper-parameter selection

The weight parameters can be learned from the training data using gradient descent. There are also hyperparameters that cannot be learned. In our model, the hyperparameters are number of hidden nodes (n_hidden) and the coefficient of regularization (lambda). We need to set those hyperparameters and use the validation data to help to choose the best one.

Because there are 2 hyperparameter, we need to follow the single variable principle in our experiment about hyperparameter selection. That is, we can only change one of them once.

We tried 20 different hidden nodes (from 4 to 80, in the step of 4) and 5 different lambda (0, 1, 5, 10, 20). Because we can only change one variable in the compared experiment, so we have tried 20 * 5 = 100 different combination of n_hidden and lambda. The following are the results of training accuracy (Train_acc(%)), validation accuracy(Validation_acc(%)), test accuracy (Test_acc(%)) and time cost with different hyperparameter.

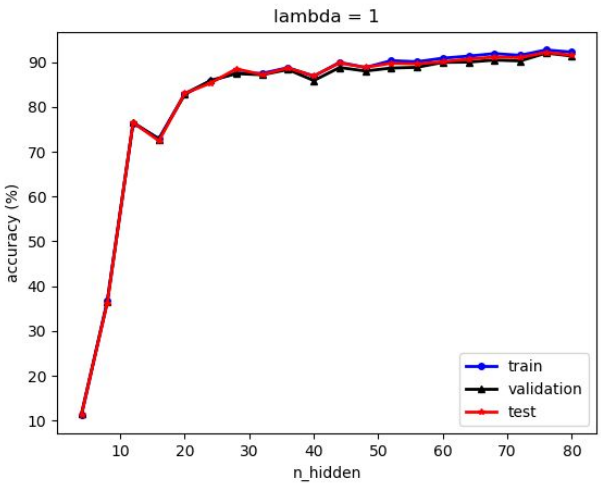
From all of the following result, we find that:

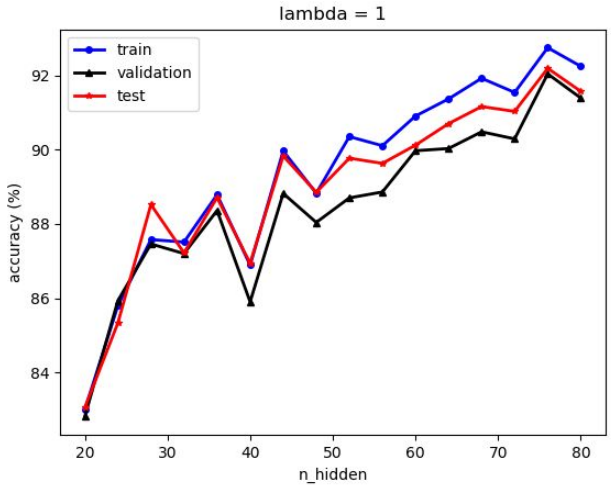
(1) With the same lambda, the train, validation and test accuracy increase with the increase of hidden node number. The reason is that insufficient number of nodes corresponds to insufficient hidden features and don't capable to classify these 10 digits. After the node number more than 40, the improvement of accuracy is tiny. Besides, it seems that the validation and test accuracy will decrease after the hidden node number larger than 76.

(2) We compare the results with all 100 hyperparameters and find that the prediction performance is best when lambda = 1 and hidden node number = 76.

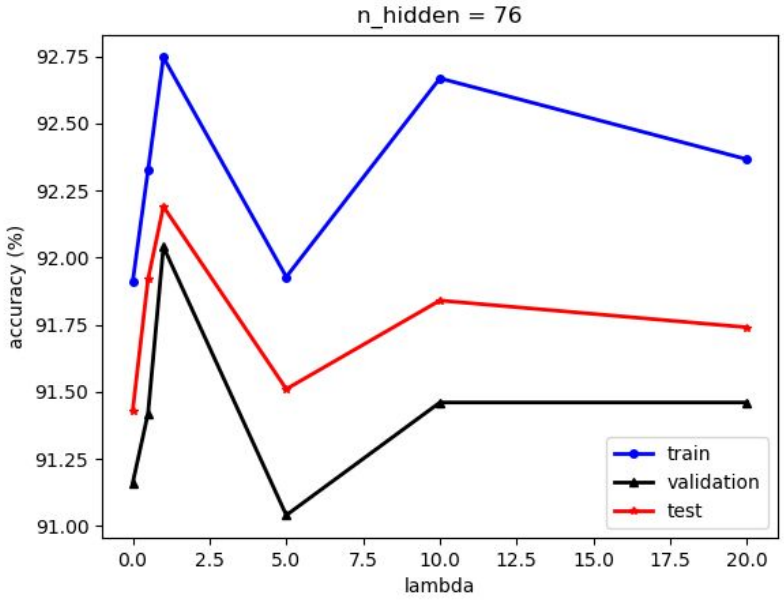
Lambda = 1

n_hidden	Train_acc(%)	Validation_acc(%)	Test_acc(%)	Time(sec)
4.0	11.17	11.57	11.35	133.82368
8.0	36.766	36.47	36.35	350.357272
12.0	76.406	76.55	76.52	330.277699
16.0	72.944	72.81	72.45	841.997008
20.0	82.994	82.81	83.05	313.628841
24.0	85.794	85.93	85.33	419.915115
28.0	87.578	87.46	88.52	620.672108
32.0	87.51	87.2	87.22	467.965487
36.0	88.786	88.36	88.71	567.977046
40.0	86.888	85.9	86.94	488.082938
44.0	89.966	88.82	89.83	454.949393
48.0	88.826	88.04	88.85	500.227723
52.0	90.35	88.7	89.77	506.138477
56.0	90.106	88.86	89.63	552.645623
60.0	90.904	89.97	90.12	503.518797
64.0	91.366	90.03	90.7	545.131033
68.0	91.922	90.48	91.16	535.915329
72.0	91.54	90.29	91.03	583.019739
76.0	92.746	92.04	92.19	671.909886
80.0	92.244	91.39	91.57	624.384654



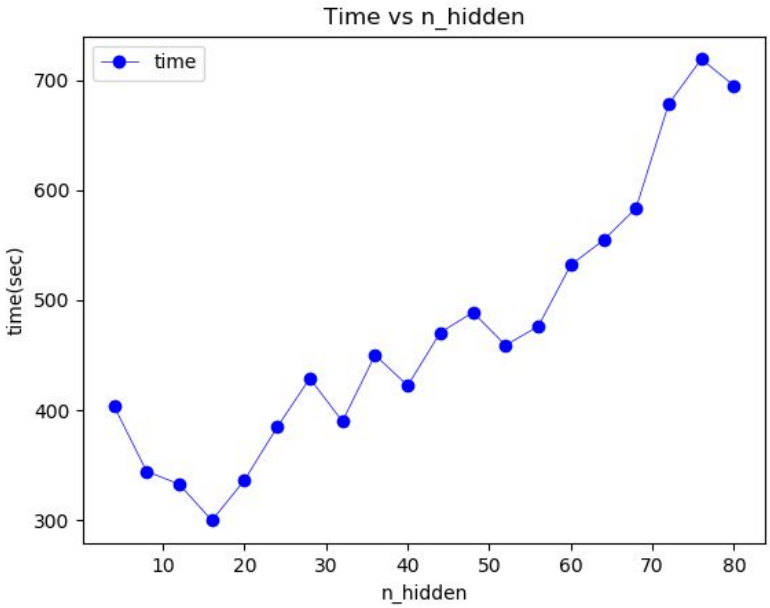


(3) With the same hidden node number (we use n_hidden = 76 because it is a hard work and meaningless to consider all conditions), we can see that both zero regularization and large regularization cannot give us optimal performance. The lambda controls amount of smoothing applied to the network. Lambda = 0 will lead to overfitting, i.e., high accuracy in training data and low accuracy in validation and test accuracy. At the other extreme, a large lambda, will produce very smooth surface and underfitting neural network. In the 6 lambda (0, 0.5, 1, 5, 10, 20), lambda = 1 shows the best performance. (Note: in n_hidden = 76, we also tried lambda = 0.5)



Lambda	Train_acc(%)	Validation_acc(%)	Test_acc(%)	Time(sec)
0.0	91.91	91.16	91.43	691.17538619
0.5	92.326	91.42	91.92	772.0139
1.0	92.746	92.04	92.19	671.909886
5.0	91.928	91.04	91.51	563.039253
10.0	92.668	91.46	91.84	719.08728695
20.0	92.366	91.46	91.74	595.87011504

(4) With the same lambda, the time cost will increase linearly with the increase the number of the hidden node. The time cost will not change with different lambda because the change of lambla will not lead to computational work change.



(All the data obtained of the 100 different hyperparameters are in the supplemental data in the end of this report.)

2: Accuracy of classification method on handwritten digits test data

The classification accuracy using neural network (nnScript, lambda=1, n_hidden = 76) is:

Training accuracy: 92.746%
Validation accuracy: 92.04%
Testing accuracy: 92.19%

The classification accuracy using convolutional neural network (cnnScript, stddev=0.05, bias = 0.05, num_filters1 = 16, num_filters2 = 36, fc_size = 128， filter size =3) is:

Training accuracy: 98.7%
Validation accuracy: 98.7%
Testing accuracy: 96.9%

3: Accuracy of classification method on CelebA dataset

The parameters of neural network running on CelebA dataset:

lambda = 10, n_hidden = 256, n_class = 2, layers = 1.

Accuracy: 50%

Training Time: 12060.21 seconds.

4: Comparison of nn to deepnn in accuracy and training time

4.1 The result of neural network running on CelebA dataset

Using neural network written by ourselves to run on CelebA dataset, the parameters are as follows: lambda = 10, n_hidden = 256, n_class = 2, layers = 1.

Accuracy: 50%

Training Time:12060.21 seconds.

4.2 The result of deep neural network with different layers

Using deep neural network running on CelebA dataset, the parameters are as follows:

learning_rate = 0.0001, training_epochs = 100, batch_size = 100, n_hidden of each layer = 256, n_class = 2, layers = 2, 3, 5, 7.

For 2 layer deep neural network, the accuracy is 80.5829%, and the training time is 418.13 seconds.

For 3 layer deep neural network, the accuracy is 79.296%, and the training time is 384.86 seconds.

For 5 layer deep neural network, the accuracy is 76.5708%, and the training time is 455.65 seconds.

For 7 layer deep neural network, the accuracy is 74.4133%, and the training time is 537.39 seconds.

Compared different layers of deep neural network: With the increment of layers, the accuracy decrease instead of increasing. Because when hidden layers' increment is much more than the sufficient number of layers, it will cause network's overfit to the training set.

4.3 Comparison

The accuracy of deep neural network is much higher than neural network. The training time of deep neural network is also much quicker than neural network. The accuracy of 2 hidden layers' deep neural network is highest.

5: Accuracy and training time of Convolutional Neural Network

5.1 Code structure

5.1.1 Feedforward propagation

In convolutional and pooling layers, we use function ***tf.truncated_normal()*** to generate a random normalized filter and ***tf.constant()*** to generate biases. Then we use ***tf.Variable()*** to set our filter and bias variables into our tensorflow frame. After this, we use ***tf.nn.conv2d()*** to do convolution with 5*5 filters and stride 1, and use ***tf.nn.max_pool()*** to do pooling with 2*2 filters and stride 2. At last, we use ***tf.nn.relu()*** to change output nonlinear. We set 16 filters in first layer and 36 in second layer.

In fully-connected layer, we use ***tf.matmul()*** to multiply input vector with weight. In first fully-connected layer, we use flatten output of convolutional layers as its input and set the node size with 128. In second layer, we set the input node size with 128 and output size with 10.

5.1.2 Backpropagation

We use ***tf.nn.softmax_cross_entropy_with_logits()*** to get cross entropy. Then we use ***tf.reduce_mean()*** to get the mean value of cross entropy which is our error cost. After this, we use ***tf.train.AdamOptimizer()*** to optimize our model with Adam algorithm which is a extended version of gradient descent algorithm. We set the alpha with value 0.0001 which Adam has recommended in his paper.

5.1.3 Convolutional neural network prediction

We use ***session.run(tf.global_variables_initializer())*** to initialize our graph flow with variables and run the session. We use ***data.train.next_batch()*** to get the training data and train these data with our optimizer for 10000 times. Then we test data with this graph flow and choose the one with the highest probability as the predicted class label.

5.2 Time and accuracy of test data with different hyper-parameter

5.2.1 Hyper-parameter selection

In our model, the hyper-parameters are:

- Bias and std parameter which used to generate the filter,size of filter
- Number of filters in convolutional layers
- Number of nodes in fully-connected layers
- Number of layers
- The parameter in gradient descent

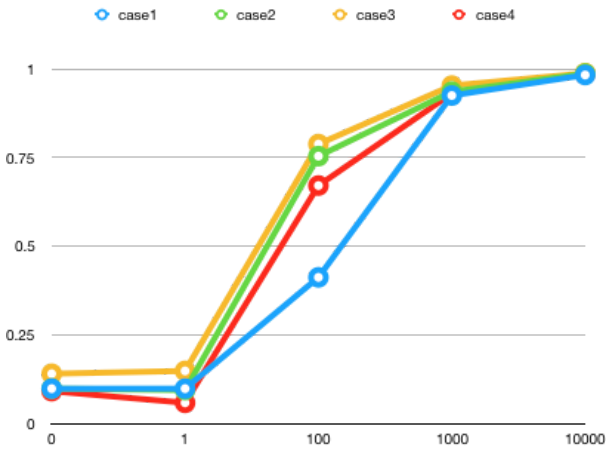
Considering we are more focused on convolutional part and the good parameter values in gradient descent have been given, we change the first two hyper-parameters to see their effect on time and accuracy.

5.2.2 Data and figure

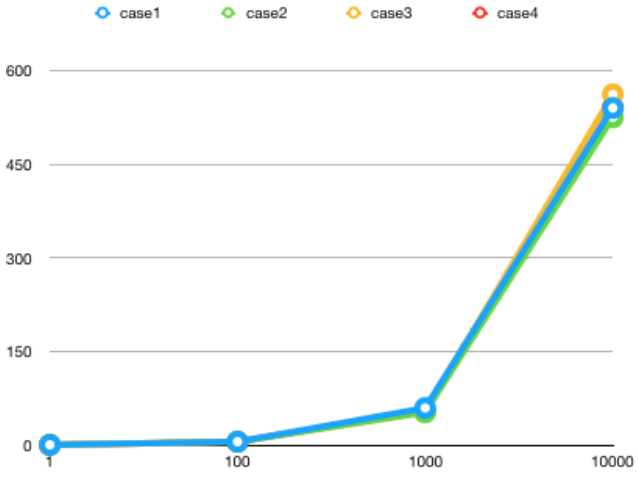
Size and std parameter which used to generate the filter:

Case1: Std = 0.02 Bias = 0.05
Case2: Std = 0.05 Bias = 0.05
Case3: Std = 0.1 Bias = 0.05
Case4: Std = 0.05 Bias = 0.1

Accuracy:



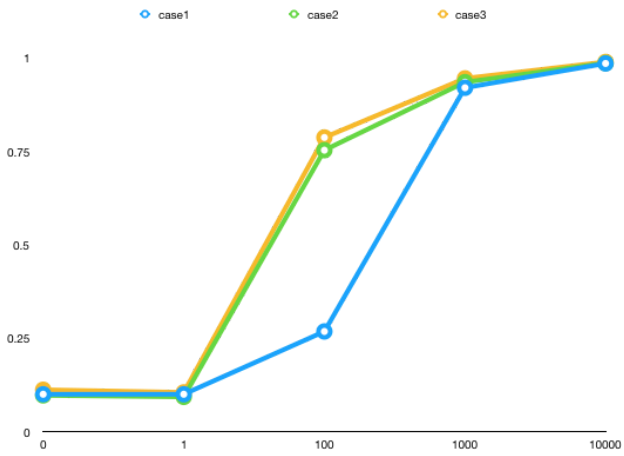
Time(seconds):



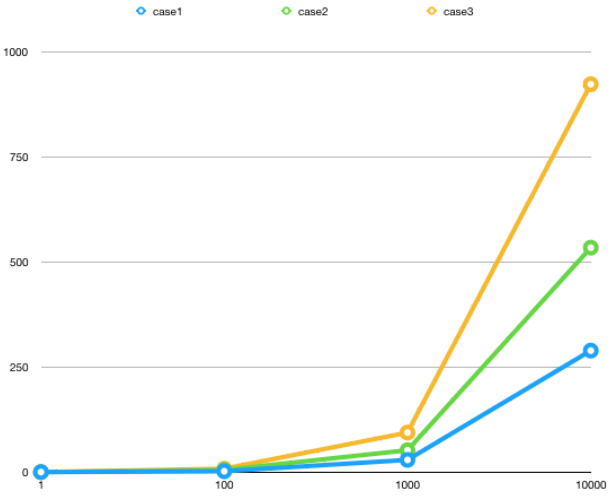
Case1: Std = 0.05 Bias = 0.05 Size = 3

Case2: Std = 0.05 Bias = 0.05 Size = 5
Case3: Std = 0.05 Bias = 0.05 Size = 7

Accuracy:

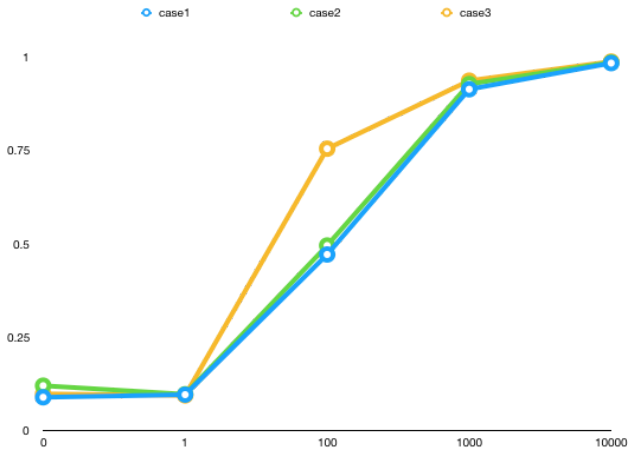


Time(seconds):

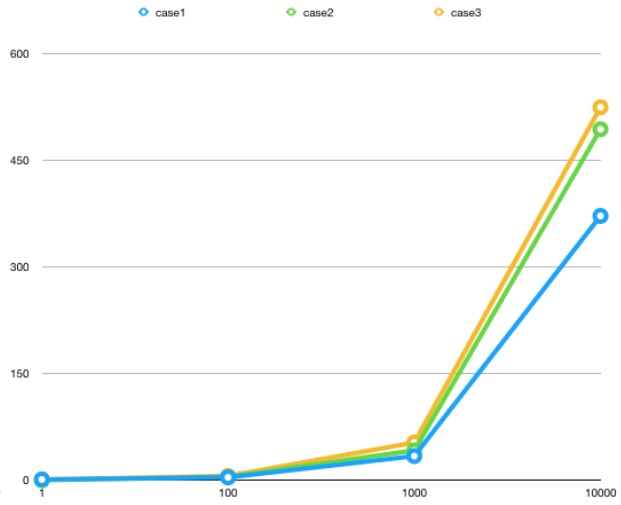


Case1: Std = 0.05 Bias = 0.05 Size = 5 filter1 =4 filter2 =36 fc = 128
Case2: Std = 0.05 Bias = 0.05 Size = 5 filter1 =8 filter2 =36 fc = 128
Case3: Std = 0.05 Bias = 0.05 Size = 5 filter1 =16 filter2 =36 fc = 128

Accuracy:



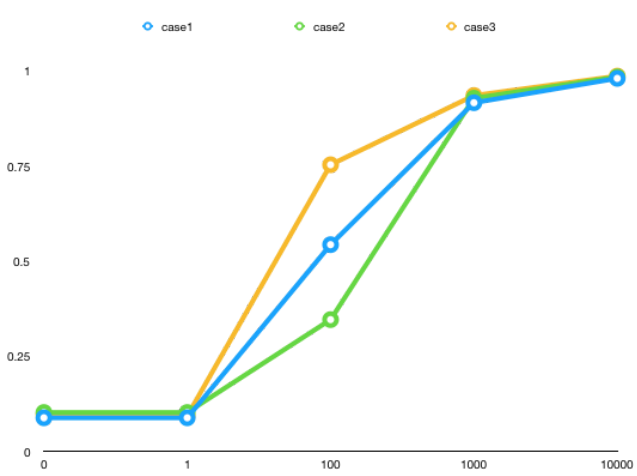
Time(seconds):

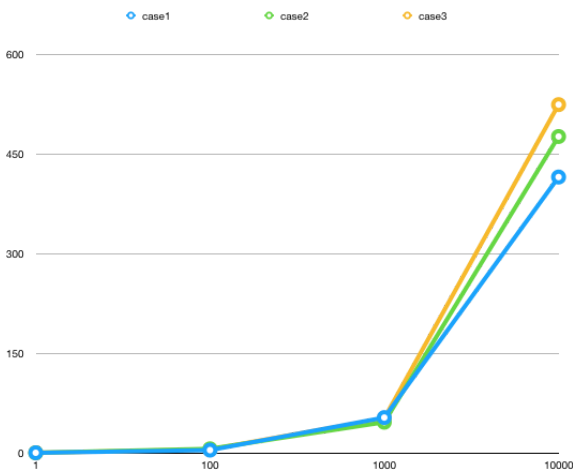


Case1: Std = 0.05 Bias = 0.05 Size = 5 filter1 =16 filter2 =9 fc = 128
Case2: Std = 0.05 Bias = 0.05 Size = 5 filter1 =16 filter2 =18 fc = 128
Case3: Std = 0.05 Bias = 0.05 Size = 5 filter1 =16 filter2 =36 fc = 128

Accuracy:

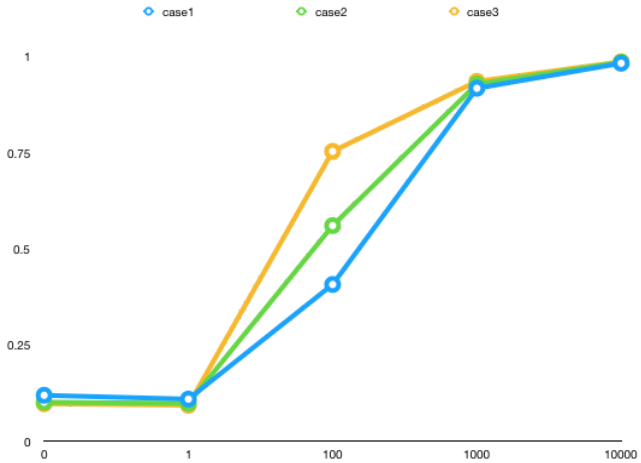
Time(seconds):



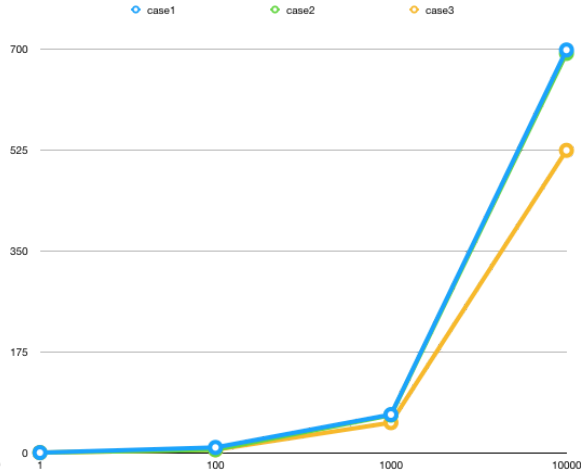


Std = 0.05 Bias = 0.05 Size = 5 filter1 =16 filter2 =36 fc = 32
Std = 0.05 Bias = 0.05 Size = 5 filter1 =16 filter2 =36 fc = 64
Std = 0.05 Bias = 0.05 Size = 5 filter1 =16 filter2 =36 fc = 128

Accuracy:



Time(seconds):



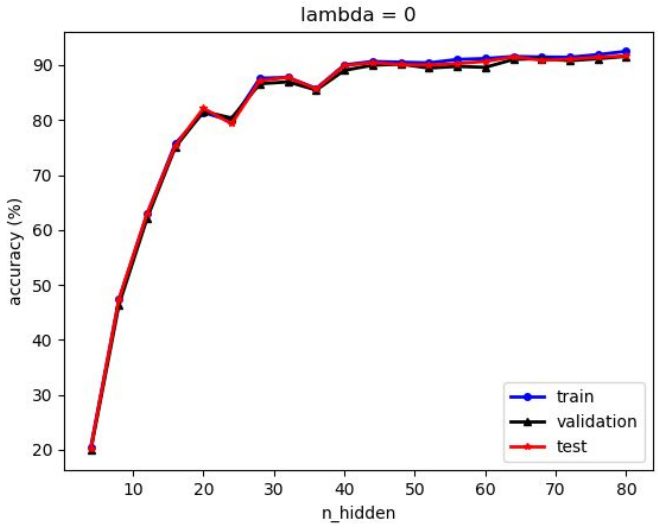
From the data above, we can see that when we run the optimizer above 1000 times, all hyper-parameter will not affect the accuracy too much and accuracy will be more than 90%. We also find that std and bias affect very little both on traning time and accuracy. The size of filter will affect the training time, time goes up with the bigger size. Number of filters will affect the time little when we run the optimizer above 1000 times. More filters will cost more time. Conversely, less node of fully_connected layer will induce the grown up of time.

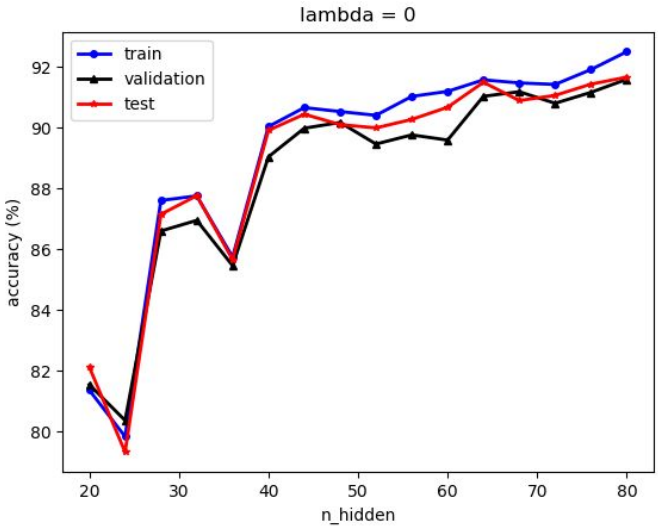
At last, we choose (stddev=0.05, bias = 0.05, num_filters1 = 16, num_filters2 = 36, fc_size = 128, filter size =3) as our parameter.

Supplemental data

Lambda = 0

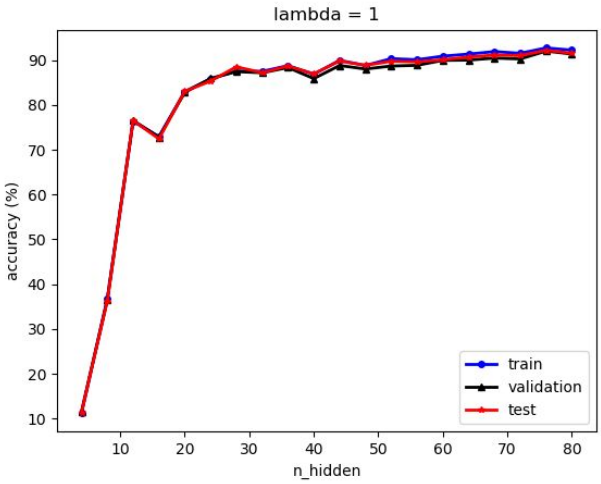
n_hidden	Train_acc(%)	Validation_acc(%)	Test_acc(%)	Time(sec)
4.0	20.454	19.9	20.21	286.22244692
8.0	47.382	46.4	47.4	328.44054699
12.0	62.91	62.14	63.0	304.03582096
16.0	75.716	75.06	75.36	355.25125813
20.0	81.37	81.55	82.13	363.57231522
24.0	79.838	80.37	79.33	432.96767807
28.0	87.604	86.6	87.15	415.85183215
32.0	87.754	86.95	87.76	495.16718221
36.0	85.744	85.46	85.66	492.04575419
40.0	90.034	89.04	89.91	497.04827785
44.0	90.658	89.98	90.44	526.10919285
48.0	90.528	90.17	90.1	424.155756
52.0	90.404	89.46	89.99	469.7942512
56.0	91.026	89.76	90.27	493.94721985
60.0	91.192	89.59	90.67	536.46507716
64.0	91.57	91.03	91.49	548.80284691
68.0	91.47	91.18	90.89	530.2837832
72.0	91.42	90.8	91.06	594.35710788
76.0	91.91	91.16	91.43	691.17538619
80.0	92.498	91.59	91.66	630.4787147

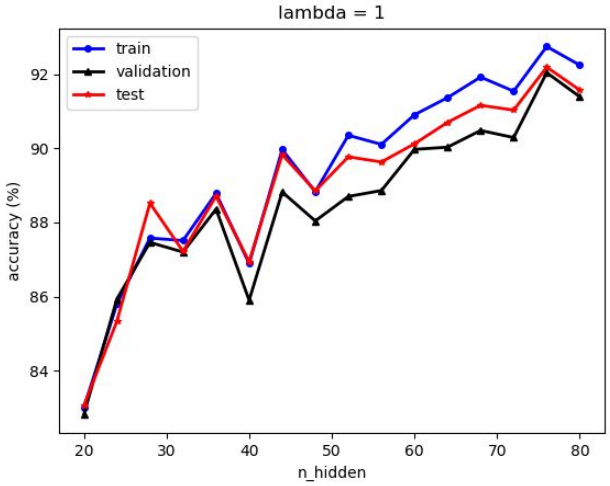




Lambda = 1

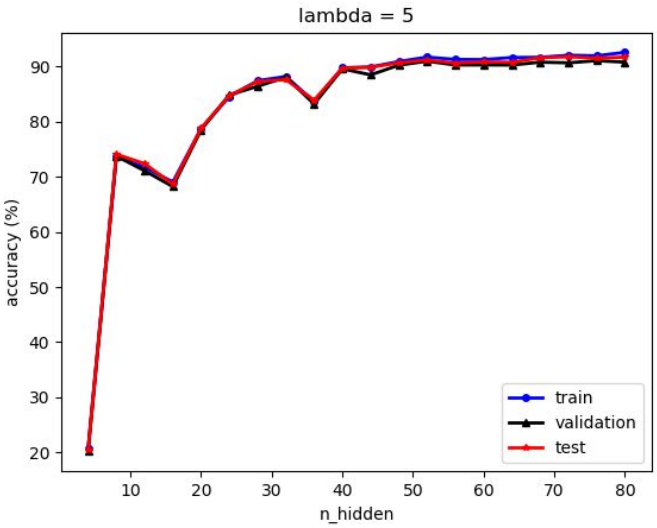
n_hidden	Train_acc(%)	Validation_acc(%)	Test_acc(%)	Time(sec)
4.0	11.17	11.57	11.35	133.82368
8.0	36.766	36.47	36.35	350.357272
12.0	76.406	76.55	76.52	330.277699
16.0	72.944	72.81	72.45	841.997008
20.0	82.994	82.81	83.05	313.628841
24.0	85.794	85.93	85.33	419.915115
28.0	87.578	87.46	88.52	620.672108
32.0	87.51	87.2	87.22	467.965487
36.0	88.786	88.36	88.71	567.977046
40.0	86.888	85.9	86.94	488.082938
44.0	89.966	88.82	89.83	454.949393
48.0	88.826	88.04	88.85	500.227723
52.0	90.35	88.7	89.77	506.138477
56.0	90.106	88.86	89.63	552.645623
60.0	90.904	89.97	90.12	503.518797
64.0	91.366	90.03	90.7	545.131033
68.0	91.922	90.48	91.16	535.915329
72.0	91.54	90.29	91.03	583.019739
76.0	92.746	92.04	92.19	671.909886
80.0	92.244	91.39	91.57	624.384654

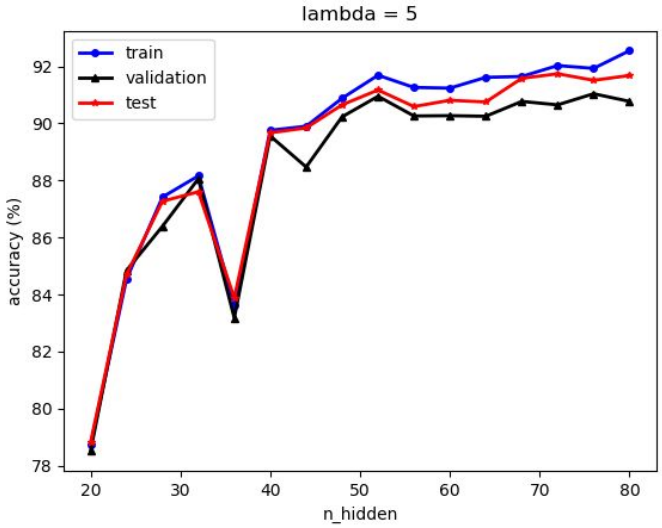




Lambda = 5

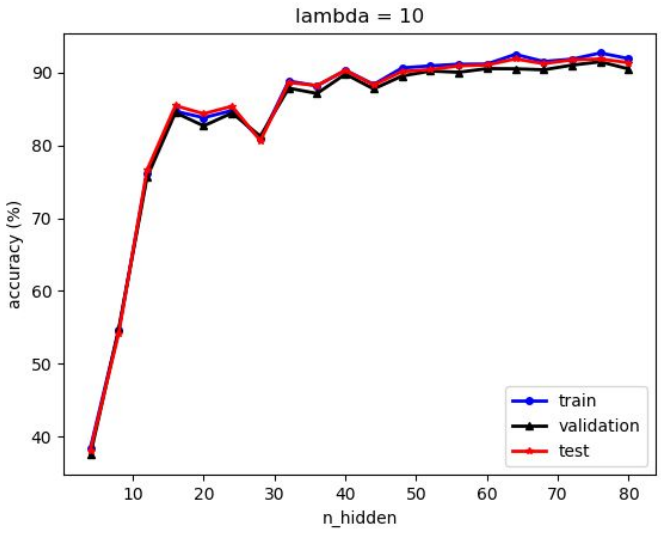
n_hidden	Train_acc(%)	Validation_acc(%)	Test_acc(%)	Time(sec)
4.0	20.66	20.2	20.46	390.57882595
8.0	73.614	73.75	74.08	361.44305205
12.0	71.718	71.07	72.37	353.92166901
16.0	69.036	68.21	68.65	406.20285106
20.0	78.734	78.51	78.81	431.52806401
24.0	84.538	84.84	84.7	469.51299715
28.0	87.414	86.39	87.27	421.78414297
32.0	88.16	88.06	87.6	450.72108293
36.0	83.636	83.18	83.88	521.79470706
40.0	89.756	89.56	89.67	549.40572095
44.0	89.9	88.47	89.84	434.67709422
48.0	90.882	90.23	90.65	520.25629902
52.0	91.688	90.95	91.17	515.31069708
56.0	91.26	90.26	90.59	503.5479219
60.0	91.236	90.27	90.81	531.25872684
64.0	91.614	90.25	90.75	552.68852186
68.0	91.644	90.77	91.58	606.53340912
72.0	92.028	90.65	91.74	635.22455287
76.0	91.928	91.04	91.51	563.039253
80.0	92.55	90.77	91.68	677.84444404

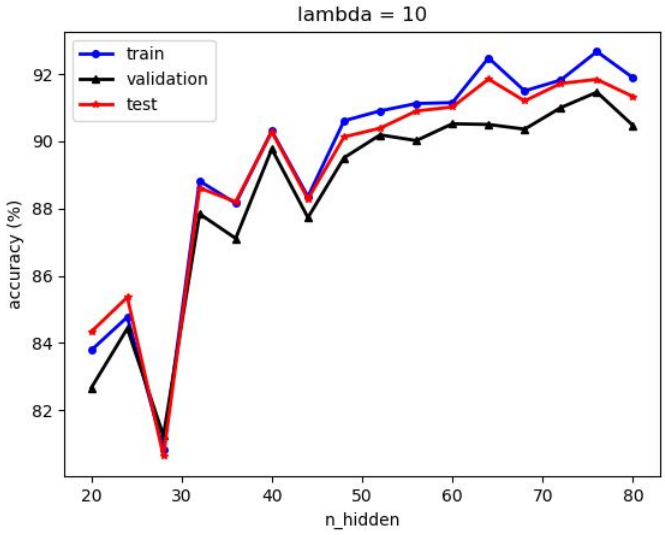




Lambda = 10

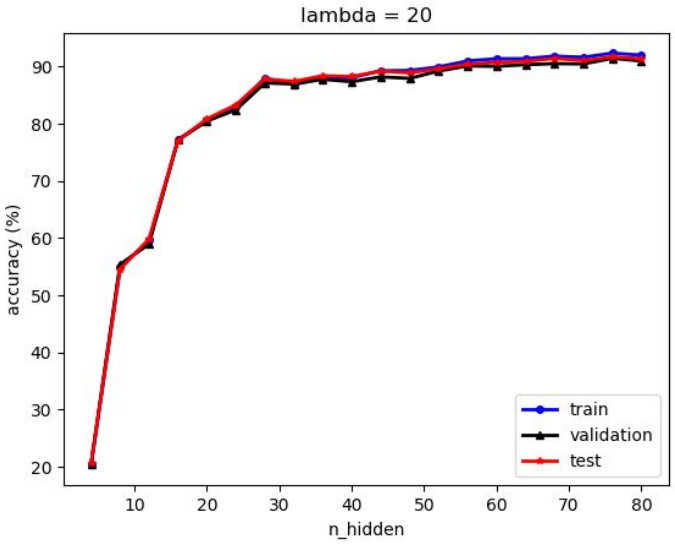
n_hidden	Train_acc(%)	Validation_acc(%)	Test_acc(%)	Time(sec)
4.0	38.448	37.52	38.14	403.70315695
8.0	54.586	54.77	54.22	344.30824399
12.0	76.224	75.62	76.6	332.57885218
16.0	84.672	84.48	85.43	299.65590906
20.0	83.78	82.66	84.33	336.30742407
24.0	84.772	84.43	85.36	384.48001218
28.0	80.836	81.24	80.63	428.82205796
32.0	88.818	87.84	88.61	389.77705121
36.0	88.152	87.11	88.2	449.98017216
40.0	90.306	89.77	90.28	422.28891492
44.0	88.352	87.73	88.28	470.49704409
48.0	90.61	89.51	90.13	488.82004118
52.0	90.904	90.19	90.39	458.9672749
56.0	91.122	90.02	90.9	476.075454
60.0	91.15	90.52	91.02	532.02430582
64.0	92.48	90.5	91.85	554.45478916
68.0	91.498	90.36	91.2	583.72096419
72.0	91.822	91.0	91.72	678.47298574
76.0	92.668	91.46	91.84	719.08728695
80.0	91.89	90.47	91.33	694.98612094

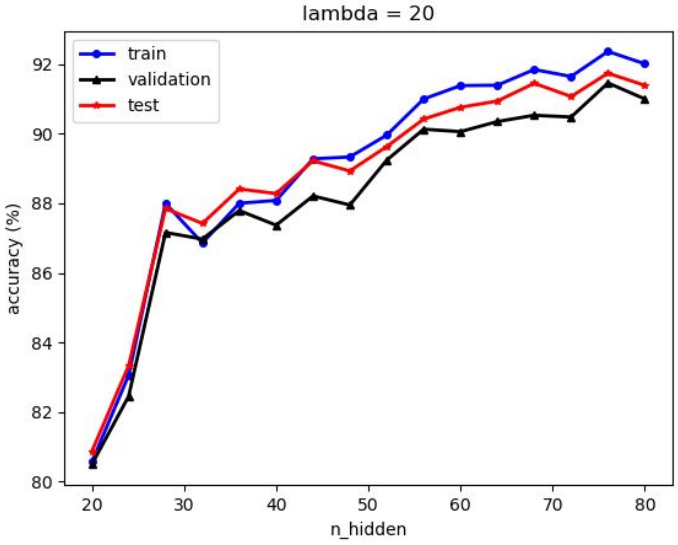




Lambda = 20

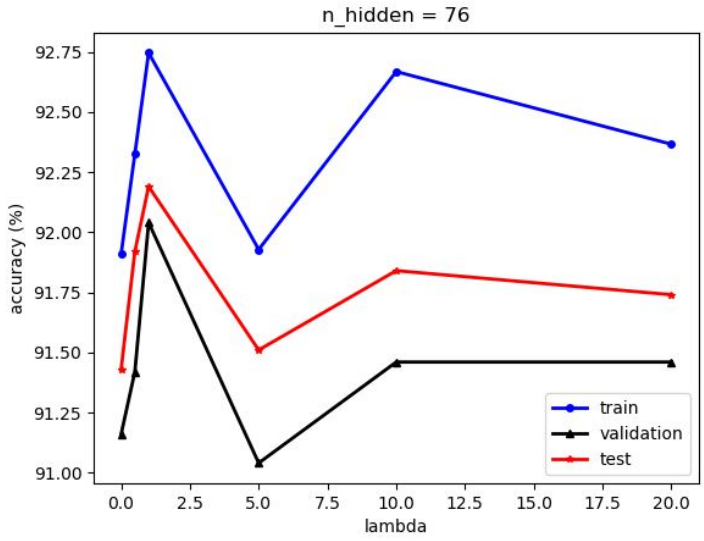
n_hidden	Train_acc(%)	Validation_acc(%)	Test_acc(%)	Time(sec)
4.0	20.346	20.49	20.6	332.38234782
8.0	54.936	55.4	54.53	376.85546303
12.0	59.562	58.96	59.88	390.02833104
16.0	77.24	77.16	77.09	380.75639081
20.0	80.558	80.49	80.86	425.77212906
24.0	83.054	82.45	83.33	483.75870275
28.0	87.996	87.16	87.85	477.01666784
32.0	86.858	86.97	87.42	539.62061787
36.0	88.0	87.79	88.41	549.21238494
40.0	88.082	87.37	88.28	534.93783879
44.0	89.278	88.21	89.22	504.78737593
48.0	89.332	87.95	88.93	614.65196371
52.0	89.96	89.24	89.63	639.04452705
56.0	90.998	90.13	90.42	557.41487074
60.0	91.384	90.06	90.76	600.46598101
64.0	91.394	90.35	90.94	569.08543682
68.0	91.846	90.53	91.45	606.91317487
72.0	91.646	90.48	91.07	598.89119911
76.0	92.366	91.46	91.74	595.87011504
80.0	92.01	91.0	91.39	597.37332201





Regularization in Neural Network

Lambda	Train_acc(%)	Validation_acc(%)	Test_acc(%)	Time(sec)
0.0	91.91	91.16	91.43	691.17538619
0.5	92.326	91.42	91.92	772.0139
1.0	92.746	92.04	92.19	671.909886
5.0	91.928	91.04	91.51	563.039253
10.0	92.668	91.46	91.84	719.08728695
20.0	92.366	91.46	91.74	595.87011504



Time vs n_hidden