Question 1

For the neural network with 2 hidden layers, the output is computed as following:

First hidden layer: $\vec{x} * W_{0,1} + \overrightarrow{b_1}$

Second hidden layer is: $\left(\vec{x} * W_{0,1} + \overrightarrow{b_1}\right) * W_{1,2} + \overrightarrow{b_2}$

$$\vec{x} * W_{0,1} * W_{1,2} + \overrightarrow{b_1} * W_{1,2} + \overrightarrow{b_2}$$

Output layer: $\left(\vec{x} * W_{0,1} * W_{1,2} + \overrightarrow{b_1} * W_{1,2} + \overrightarrow{b_2}\right) * W_{2,3} + \overrightarrow{b_3}$

$$\vec{x} * W_{0,1} * W_{1,2} * W_{2,3} + \overrightarrow{b_1} * W_{1,2} * W_{2,3} + \overrightarrow{b_2} * W_{2,3} + \overrightarrow{b_3}$$

For the second neural network, the output is

$$\vec{x} * \widetilde{W} + \tilde{b}$$

In order for two networks to be identical,

$$\widetilde{W} = W_{0,1} * W_{1,2} * W_{2,3}$$

$$\tilde{b} = \overrightarrow{b_1} * W_{1,2} * W_{2,3} + \overrightarrow{b_2} * W_{2,3} + \overrightarrow{b_3}$$

$\vec{x}$ and $\vec{b}$ are row vector. W is matrix, where (i, j)-th entry is the weight from node i in previous layer to node j in next layer.

Question 2

Three networks perform differently because they have different number of hidden layers and for each hidden layer, the number of nodes are different. From function view of neural network, those layers are equivalent to adding up different functions and forming a function to approximate true function. The formed function approximates the true function with different accuracy. From manifold view of neural network, those networks map input to different dimensions. In different dimensions, it is different how input with same class score are separable from each other. What's more, the number of parameters and configuration of networks are different, which constrains how complex features the network can learn.

Overall, the network with 14-100-40-4 performs best with correctness as high as 0.99 for both training data and testing data.

These are the reasons why this setting works best. One is that the hidden layer has 100 nodes which is much more than the hidden nodes in other networks. With 100 nodes in the hidden layer, it can be viewed as adding 100 functions to approximate the true function. The true function can be well approximated. In the hidden layer with 100 nodes, the input data with 14 dimensions is mapped to 100 dimension, which is much larger than the original dimension. It is easier to separate data with low dimension in higher dimension.

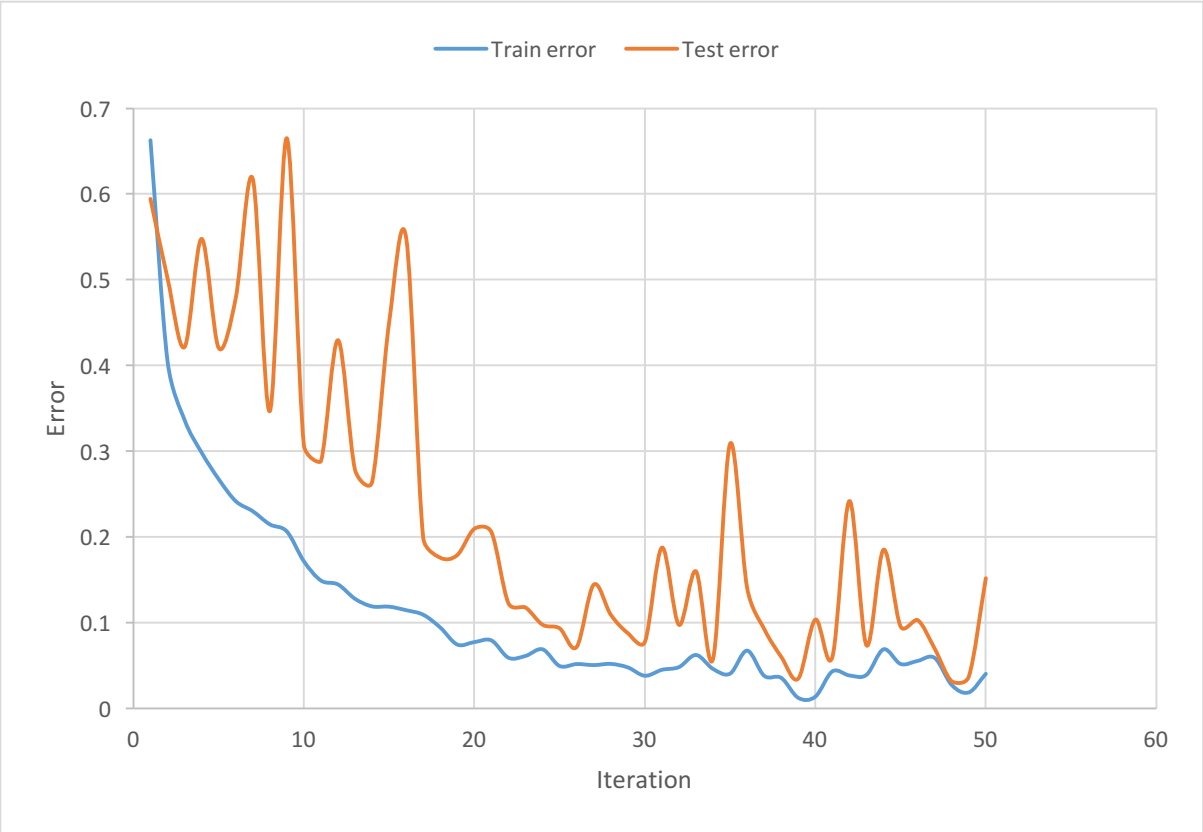
Another reason is that network with 14-100-40-4 is less likely to suffer from vanishing gradient descent problem as it contains fewer layers compared to the other two networks. Moreover, network with 14-100-40-4 allows us to search larger weight and bias space. For deep network, the output quickly overflows when weights are greater than one. This constrains the weight space and bias space the network can be trained on. It is more likely

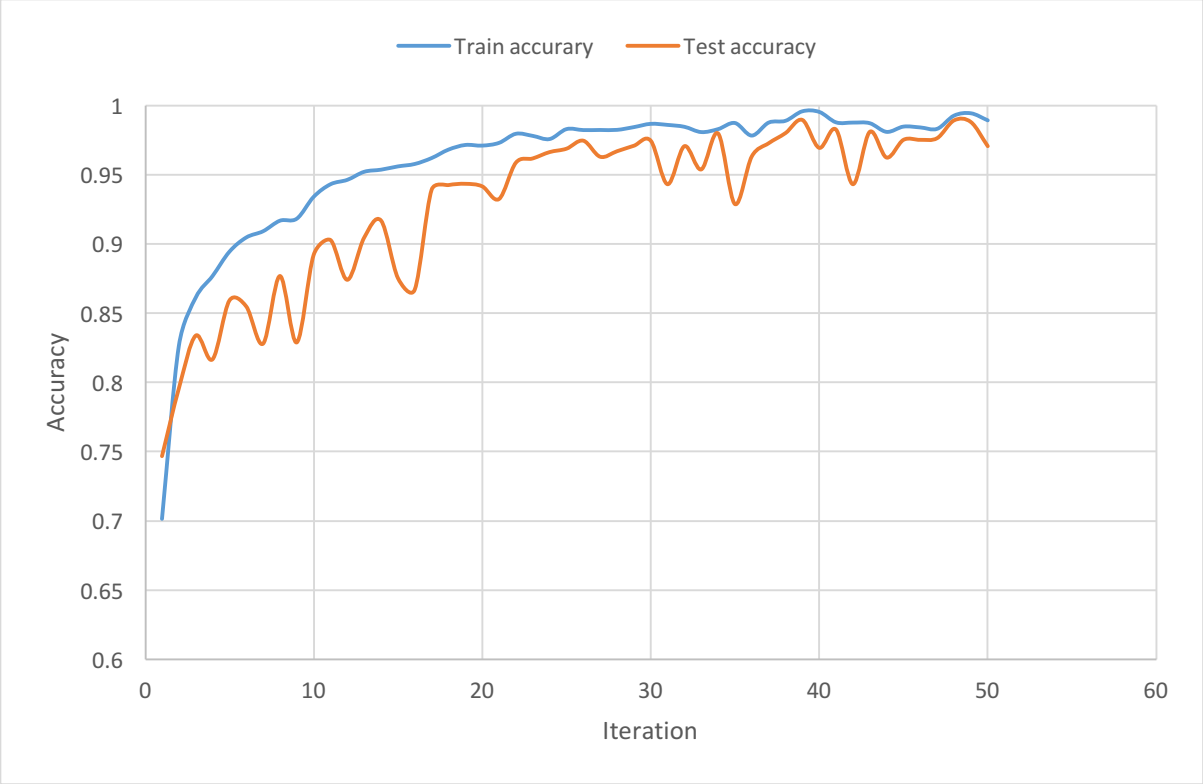
to find global minimum with larger weight and bias space. Therefore, network with 14-100-40-4 is more likely to perform better.

Anomalies

1. The output becomes nan if the weight is large. This is because the output overflows. To solve the problem, I used small weights and learning rate.
2. Local minimum is another problem when training the network. To overcome the problem, I used momentum when calculating gradients.
3. Test error sometimes increases with number of iterations. This is problem of overfitting. To resolve the problem, we can allocate an additional validation set.
4. When calculating the loss function, nan is returned. This is because softmax can be zero for some values in hot vector and log(0) is not defined. To overcome this problem, I used softmax = np.max(softmax, np.exp(-100)) to avoid softmax becomes zero. This will not affect calculating gradient because the error is not used to calculate it. This, however, will affect the absolute value of error. I used both accuracy and error as an indication of performance.
5. When calculating the softmax, the value overflows. I used stable softmax.
6. For network with 14-14*28-4, the error changes very slowly. This is because the gradient gets smaller and smaller when we move backward (as weight is less than 1). Therefore, the change in weights after each iteration is small and error decreases very slowly. I tried batch processing and momentum to alleviate the problem.

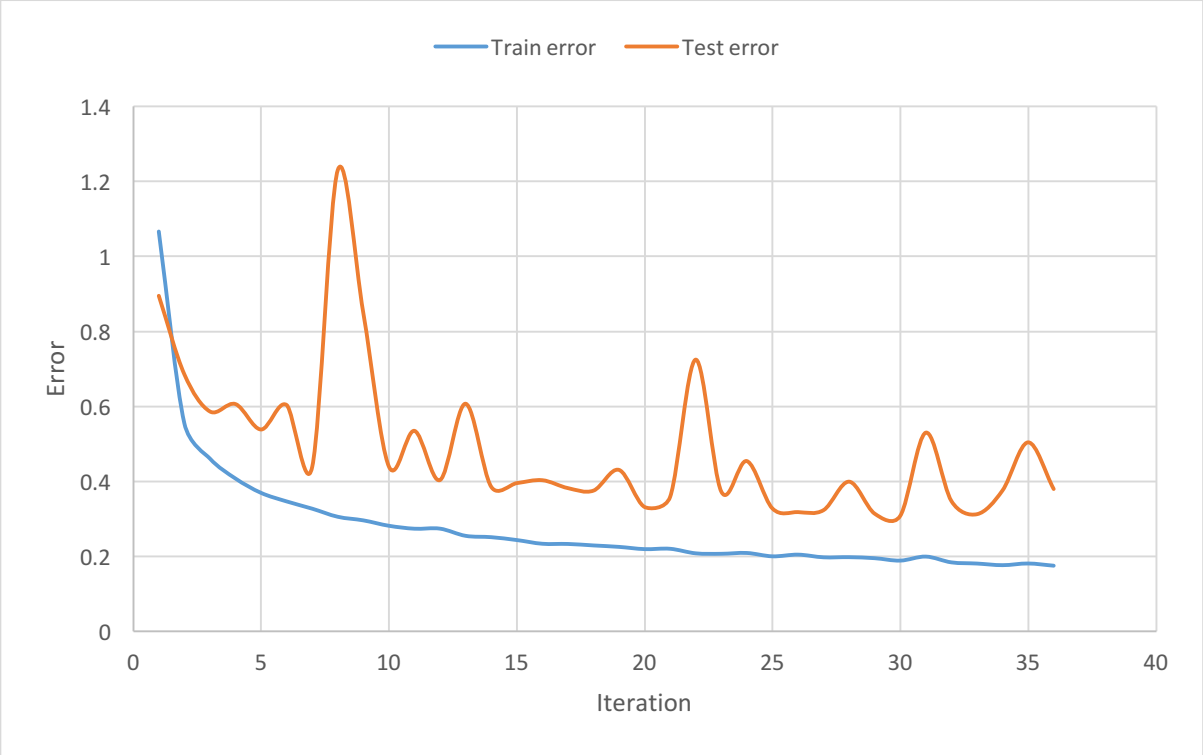
14-100-40-4

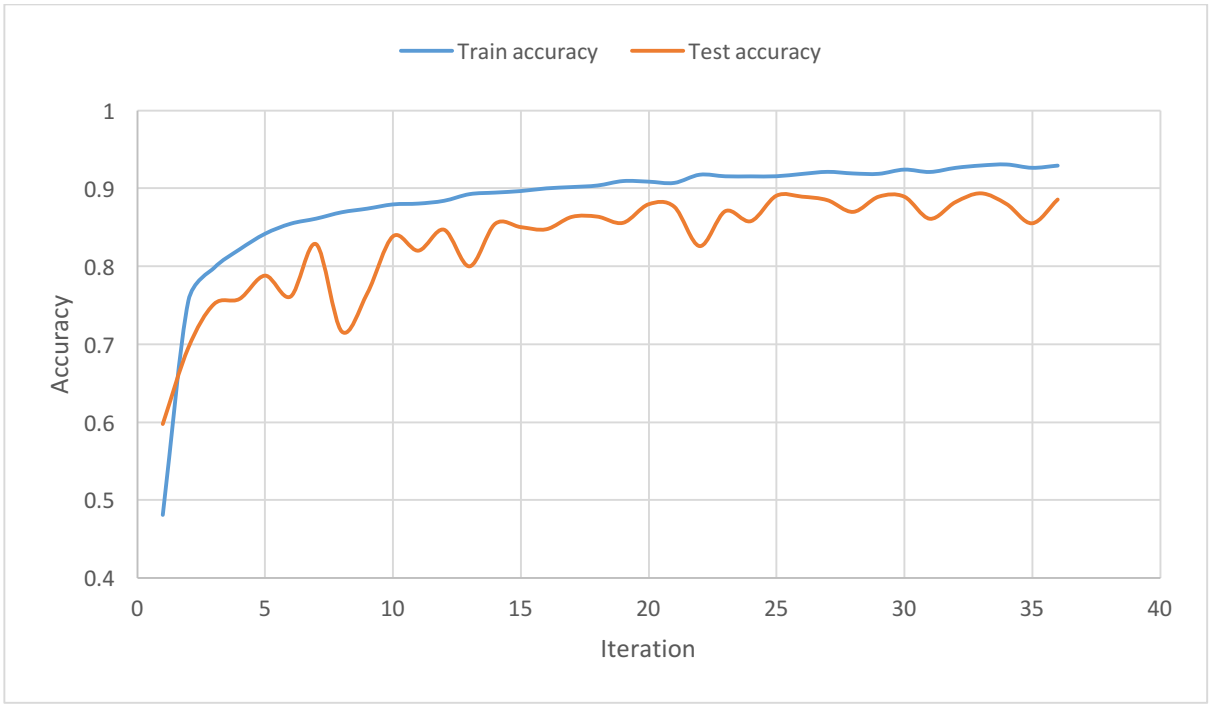




Accuracy used in this chart.
Accuracy = Number of correctly classified items / total number of items

14-28*6-4




Train accuracy
Test accuracy
Accuracy
1
0.9
0.8
0.7
0.6
0.5
0.4
0
5
10
15
20
25
30
35
40
Iteration