HW4 − Q1

(a) **Feedforward Computation:** Perform the feedforward calculation for the input vector $\mathbf{x} = [\,+1 \;\; -1 \;\; +1\,]^{\mathrm{T}}$. Fill in the following table. Follow the notation used in the slides, *i.e.*, $\mathbf{s}^{(l)}$ is the linear activation, $\mathbf{a}^{(l)} = \underline{h}(\mathbf{s}^{(l)})$, and $\dot{\mathbf{a}}^{(l)} = \underline{\dot{h}}(\mathbf{s}^{(l)})$.

l:	1	2	3
$\mathbf{s}^{(l)}$:	$W^{(1)}\cdot x+b^{(1)}=$ $\begin{bmatrix}1&-2&1\\3&4&-2\end{bmatrix}\begin{bmatrix}1\\-1\\1\end{bmatrix}+\begin{bmatrix}1\\-2\end{bmatrix}=\begin{bmatrix}5\\-5\end{bmatrix}$	$W^{(2)}\cdot a^{(1)}+b^{(2)}=$ $\begin{bmatrix}1&-2\\3&4\end{bmatrix}\begin{bmatrix}5\\0\end{bmatrix}+\begin{bmatrix}1\\0\end{bmatrix}=\begin{bmatrix}6\\15\end{bmatrix}$	$W^{(3)}\cdot a^{(2)}+b^{(3)}=$ $\begin{bmatrix}2&2\\3&-3\\2&1\end{bmatrix}\begin{bmatrix}6\\15\end{bmatrix}+\begin{bmatrix}0\\-4\\-2\end{bmatrix}=\begin{bmatrix}42\\-31\\25\end{bmatrix}$
$\mathbf{a}^{(l)}$:	$\because h(s^{(1)})=\mathrm{ReLU}(s^{(1)})=\max(s^{(1)},0)$ $\mathrm{ReLU}(\begin{bmatrix}5\\-1\end{bmatrix})=\begin{bmatrix}5\\0\end{bmatrix}$	$\mathrm{ReLU}(\begin{bmatrix}6\\15\end{bmatrix})=\begin{bmatrix}6\\15\end{bmatrix}$	$\because \mathrm{softmax}=e^{s_i^{(l)}}/\Sigma_{j=1}^{n} e^{s_j^{(l)}}$ $\mathrm{softmax}(\begin{bmatrix}42\\-31\\25\end{bmatrix})\approx\begin{bmatrix}1\\0\\0\end{bmatrix}$
$\dot{\mathbf{a}}^{(l)}$:	$\because \dot{h}(s^{(1)})=\begin{cases}1, & s^{(1)}>0\\0, & s^{(1)}\le 0\end{cases}$ $\begin{bmatrix}1\\0\end{bmatrix}$	$\begin{bmatrix}1\\1\end{bmatrix}$	(not needed)

(b) **Backpropagation Computation:** Apply standard SGD backpropagation for the input assuming a multi-category cross-entropy loss function and one-hot labeled target: $\mathbf{y} = [\,0 \;\; 0 \;\; 1\,]^{\mathrm{T}}$. Follow the notation used in the slides, *i.e.*, $\delta^{(l)} = \nabla_{\mathbf{s}^{(l)}} C$. Enter the delta values in the table below and provide the updated weights and biases assuming a learning rate $\eta = 0.5$.

$$\because \underline{h}(s_i^{(3)}) = a_i^{(3)} = \mathrm{softmax} = \frac{e^{s_i^{(l)}}}{\Sigma_{j=1}^{n} e^{s_j^{(l)}}}$$

$$C = \text{cross-entropy loss function} = -\Sigma_{i=1}^{c} y_i \log(p_i)$$

$$\therefore \delta^{(3)} = \nabla_{s^{(l)}} C = \frac{\partial C}{\partial s_i^{(3)}} = \Sigma_{j=1}^{c} \frac{\partial C}{\partial a_i^{(3)}} \cdot \frac{\partial a_i^{(3)}}{\partial s_i^{(3)}}$$

$$\text{when } i=k: \frac{\partial a_i^{(3)}}{\partial s_i^{(3)}} = a_i^{(3)}(1-a_i^{(3)})$$

$$i\neq k: \frac{\partial a_i^{(3)}}{\partial s_k^{(3)}} = -a_i^{(3)} a_k^{(3)}$$

$$\frac{\partial C}{\partial a_j^{(3)}} = -\frac{y_i}{a_j^{(3)}} \quad \text{when } y_i = 1, \ \frac{\partial C}{\partial a_j^{(3)}} = -\frac{1}{a_j^{(3)}}$$

$$\therefore \text{when } i = k, \ \frac{\partial C}{\partial s_i^{(3)}} = a_i^{(3)} - y_i$$

$$\text{when } i \neq k, \ \frac{\partial C}{\partial s_i^{(3)}} = 0$$

l:	1	2	3
$\delta^{(l)}$:	$\delta^{(1)} = (W^{(2)})^T \delta^{(2)} \circ a^{(1)} =$ $\begin{bmatrix} 1 & 3 \\ -2 & 4 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \circ \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \end{bmatrix}$	$\delta^{(2)} = (W^{(3)})^T \delta^{(3)} \circ \dot{a}^{(2)} =$ $\begin{bmatrix} 2 & 3 & 2 \\ 2 & -3 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \circ \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$	$\because \delta^{(3)} = a^{(3)} - y$ $\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$
$\mathbf{W}^{(l)}$:	$W^{(1)} - \eta \delta^{(1)} (x)^T =$ $\begin{bmatrix} 1 & -2 & 1 \\ 3 & 4 & -2 \end{bmatrix} - 0.5 \begin{bmatrix} 3 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & -1 & 1 \end{bmatrix}$ $= \begin{bmatrix} -0.5 & -0.5 & -0.5 \\ 3 & 4 & -2 \end{bmatrix}$	$W^{(2)} - \eta \delta^{(2)} (a^{(1)})^T =$ $\begin{bmatrix} 1 & -2 \\ 3 & 4 \end{bmatrix} - 0.5 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 5 & 0 \end{bmatrix} = \begin{bmatrix} 1 & -2 \\ 0.5 & 4 \end{bmatrix}$	$W^{(3)} - \eta \delta^{(3)} (a^{(2)})^T =$ $\begin{bmatrix} 2 & 2 \\ 3 & -3 \\ 2 & 1 \end{bmatrix} - 0.5 \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \begin{bmatrix} 6 & 15 \end{bmatrix} = \begin{bmatrix} -1 & -5.5 \\ 3 & -3 \\ 5 & 8.5 \end{bmatrix}$
$\mathbf{b}^{(l)}$:	$b^{(1)} - \eta \delta^{(1)} =$ $\begin{bmatrix} 1 \\ -2 \end{bmatrix} - 0.5 \begin{bmatrix} 3 \\ 0 \end{bmatrix} = \begin{bmatrix} -0.5 \\ -2 \end{bmatrix}$	$b^{(2)} - \eta \delta^{(2)} =$ $\begin{bmatrix} 1 \\ 0 \end{bmatrix} - 0.5 \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ -0.5 \end{bmatrix}$	$b^{(3)} - \eta \delta^{(3)} =$ $\begin{bmatrix} 0 \\ -4 \\ -2 \end{bmatrix} - 0.5 \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} -0.5 \\ -4 \\ -1.5 \end{bmatrix}$

hw4q2

In [231]:
```
import numpy as np
import pandas as pd
import h5py
import matplotlib.pyplot as plt
```

In [232]:
```
with h5py.File('mnist_traindata.hdf5','r') as f:
    xdata_train=np.array(f['xdata'])
    ydata_train=np.array(f['ydata'])
    y_train=np.zeros(len(ydata_train))
    for i in range(len(ydata_train)):
        if np.argmax(ydata_train[i])==2:
            y_train[i]=1


with h5py.File('mnist_testdata.hdf5','r') as f:
    xdata_test=np.array(f['xdata'])
    ydata_test=np.array(f['ydata'])
    y_test=np.zeros(len(ydata_test))
    for i in range(len(ydata_test)):
        if np.argmax(ydata_test[i])==2:
            y_test[i]=1

weights=[]
bias=0
```

In [233]:
```
def sigmoid(z):
    return 1/(1+np.exp(-z))

def binary_log_loss(W,x,w0,y,reg_type=None,lmd=0.1):
    m=x.shape[0]
    z=np.dot(x,W)+w0
    p=sigmoid(z)

    loss=-np.sum(y*np.log(p)+(1-y)*np.log(1-p))/m

    if reg_type=='l1':
        loss+=(lmd/m)*np.sum(np.abs(W))
    elif reg_type=='l2':
        loss+=(lmd/(2*m))*np.sum(np.square(W))

    return loss
```

In [234]:
```
def plt_accuracy_loss(loss_train,loss_test,accuracy_train,accuracy_test,learni
    plt.figure()
    plt.plot(loss_train,label='loss train')
    plt.plot(loss_test,label='loss test')
    plt.xlabel('Iteration Number')
    plt.ylabel('Log loss')
    plt.title(f'learning rate{learning_rate}:log loss')
    plt.legend()
    plt.show()

    plt.figure()
    plt.plot(accuracy_train,label='accuracy train')
    plt.plot(accuracy_test,label='accuracy test')
    plt.xlabel('Iteration Number')
    plt.ylabel('accuracy')
    plt.title(f'learning rate{learning_rate}:Accuracy')
    plt.legend()
    plt.show()
```

In [235]:
```
def print_table(iter_stop,accuracy_train,loss_train,accuracy_test,loss_test,le
    data={
        'Learning rate':learning_rate,
        'Iter':iter_stop,
        'Accuracy train':accuracy_train,
        'Loss train':loss_train,
        'Accuracy test':accuracy_test,
        'Loss test':loss_test,
    }
    df=pd.DataFrame(data)
    print(df)
```

In [236]:

```
def gradient_descent(W,w0,x_train,x_test,y_train,y_test,learning_rate,max_iter
    m=x_train.shape[0]
    loss_history_train=[]
    loss_history_test=[]
    accuracy_history_train=[]
    accuracy_history_test=[]
    threshold=1e-4
    pre_loss_train=0

    for iter in range(max_iter):
        z_train=np.dot(x_train,W)+w0
        p_train=sigmoid(z_train)

        #draw initial W w0 --- loss and accuracy of test
        if iter==0:
            loss_train=binary_log_loss(W,x_train,w0,y_train,reg_type,lmd)
            loss_history_train.append(loss_train)
            loss_test=binary_log_loss(W,x_test,w0,y_test,reg_type,lmd)
            loss_history_test.append(loss_test)
            z_test=np.dot(x_test,W)+w0
            p_test=sigmoid(z_test)
            predictions_test=np.where(p_test>=0.5,1,0)
            accuracy_test=np.mean(predictions_test==y_test)
            accuracy_history_test.append(accuracy_test)

        dw=np.dot(x_train.T,(p_train-y_train))/m
        db=np.sum(p_train-y_train)/m

        if reg_type=='l1':
            dw+=(lmd/m)*np.sign(W)
        elif reg_type=='l2':
            dw+=(lmd/m)*W

        W-=learning_rate*dw
        w0-=learning_rate*db

        loss_train=binary_log_loss(W,x_train,w0,y_train,reg_type,lmd)
        loss_history_train.append(loss_train)
        loss_test=binary_log_loss(W,x_test,w0,y_test,reg_type,lmd)
        loss_history_test.append(loss_test)

        predictions_train=np.where(p_train>=0.5,1,0)
        accuracy_train=np.mean(predictions_train==y_train)
        accuracy_history_train.append(accuracy_train)

        z_test=np.dot(x_test,W)+w0
        p_test=sigmoid(z_test)
        predictions_test=np.where(p_test>=0.5,1,0)
        accuracy_test=np.mean(predictions_test==y_test)
        accuracy_history_test.append(accuracy_test)

        #print(loss_train-pre_loss_train)
        if np.abs(loss_train-pre_loss_train)<threshold:
            print(f"Model converged at iter {iter}")
            break
        pre_loss_train=loss_train

    plt_accuracy_loss(loss_history_train,loss_history_test,accuracy_history_tr
```

```
    return W,w0,loss_train,loss_test,accuracy_train,accuracy_test,iter
```

In [237]:

```
def logistic_regression(x_train,x_test,y_train,y_test,max_iter,learning_rates,
    iter_stop=[]
    final_accuracy_train=[]
    final_loss_train=[]
    final_accuracy_test=[]
    final_loss_test=[]
    for learning_rate in learning_rates:
        n=x_train.shape[1]
        W=np.zeros(n)
        w0=0
        W,w0,loss_train,loss_test,accuracy_train,accuracy_test,iter=gradient_d
        iter_stop.append(iter)
        final_accuracy_train.append(accuracy_train)
        final_loss_train.append(loss_test)
        final_accuracy_test.append(accuracy_test)
        final_loss_test.append(loss_train)
    print_table(iter_stop,final_accuracy_train,final_loss_train,final_accuracy
```

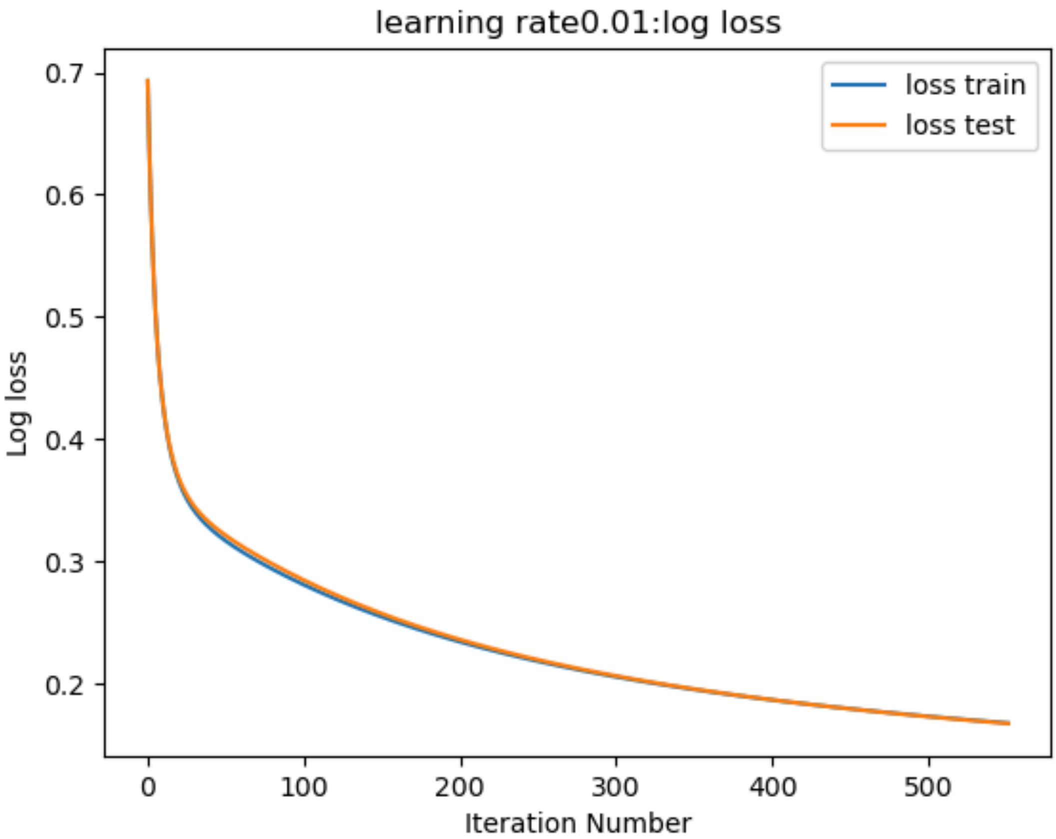
In [238]:

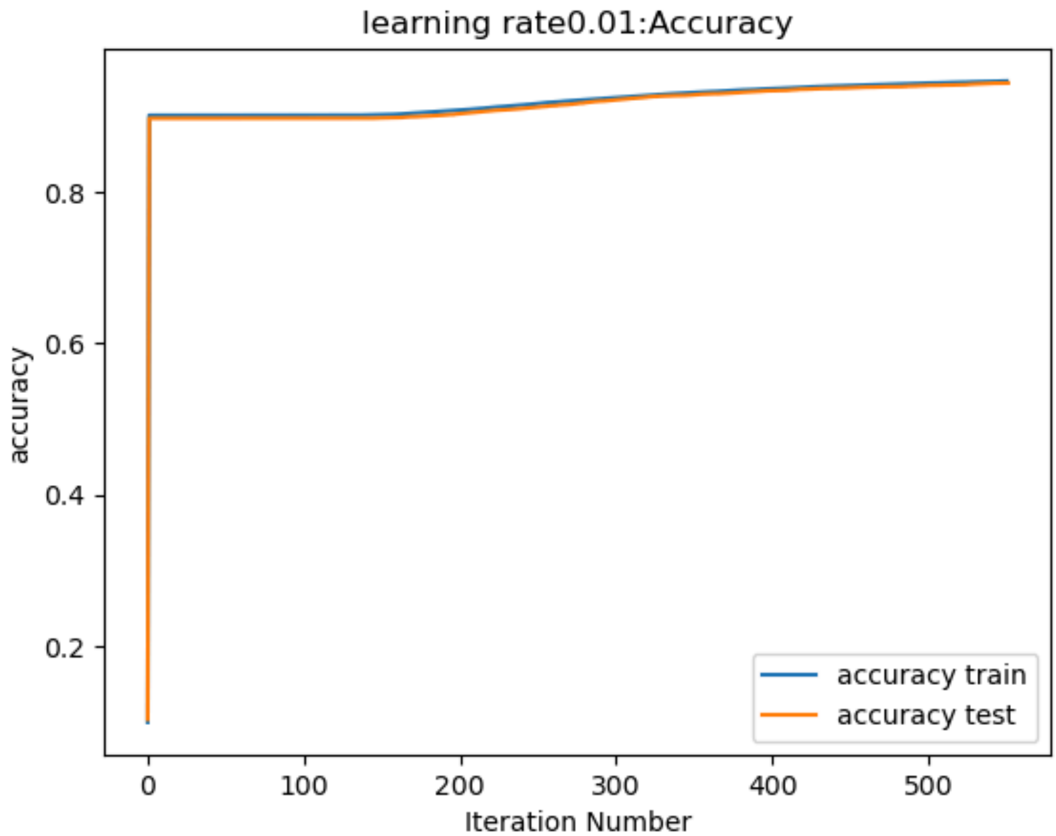
```
learning_rates=[0.01,0.1,1,2,3]
max_iter=1000
lmd=0.01
```

No regulariser

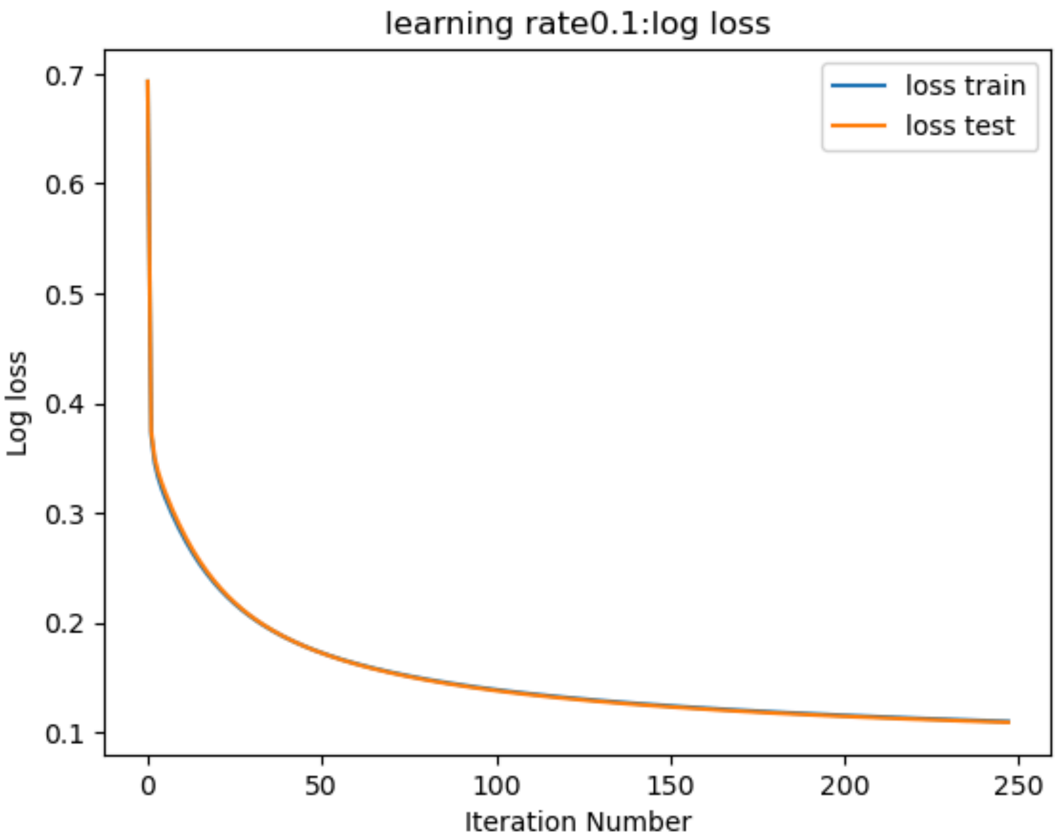
In [239]:
```
logistic_regression(xdata_train,xdata_test,y_train,y_test,max_iter,learning_ra
```

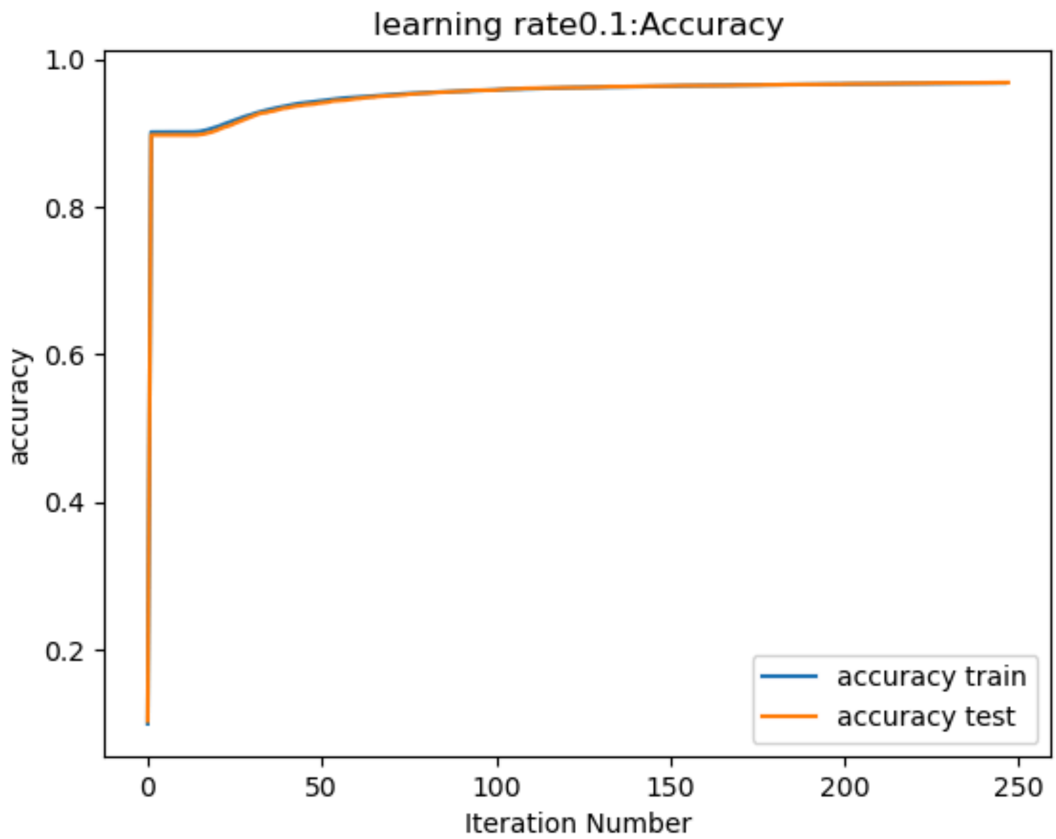
```
Model converged at iter 550
```



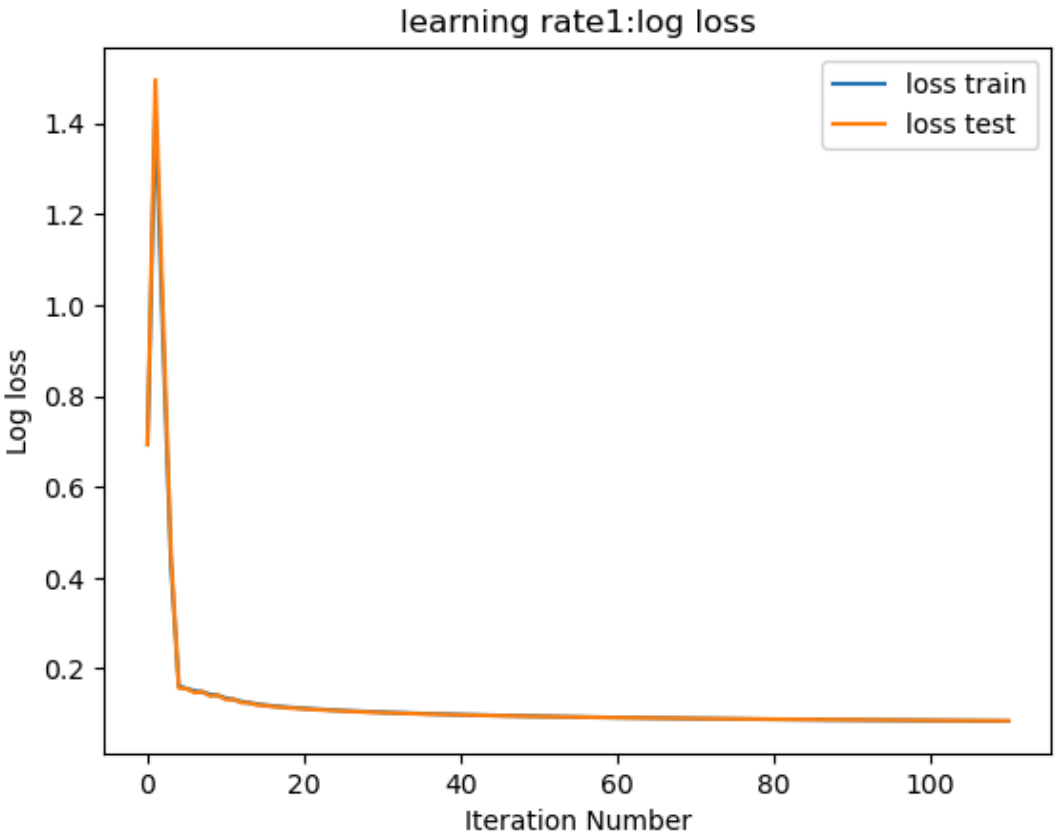


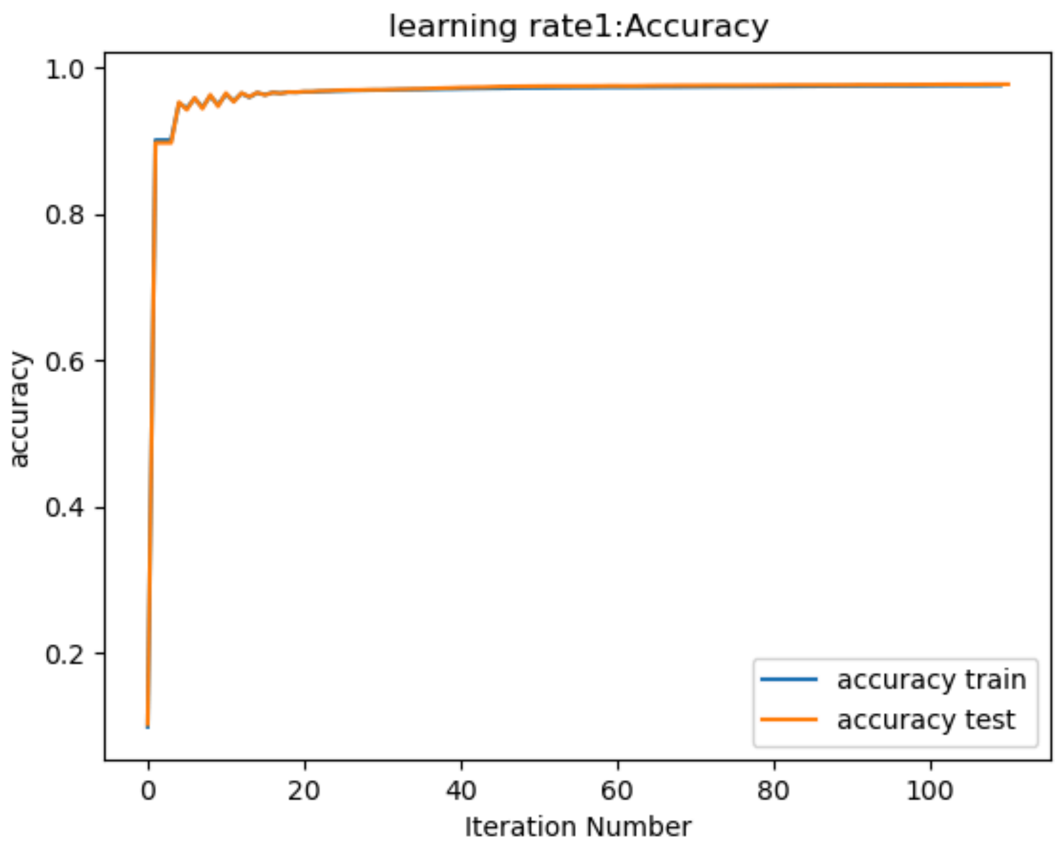
```
Model converged at iter 246
```



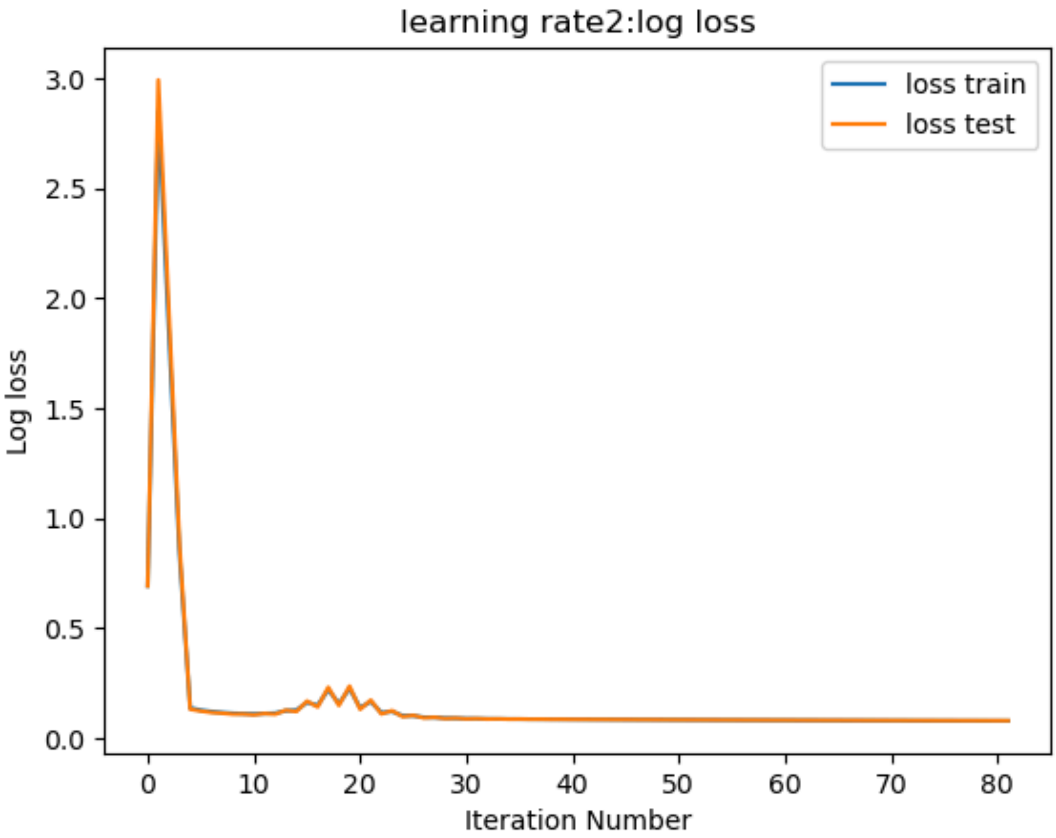


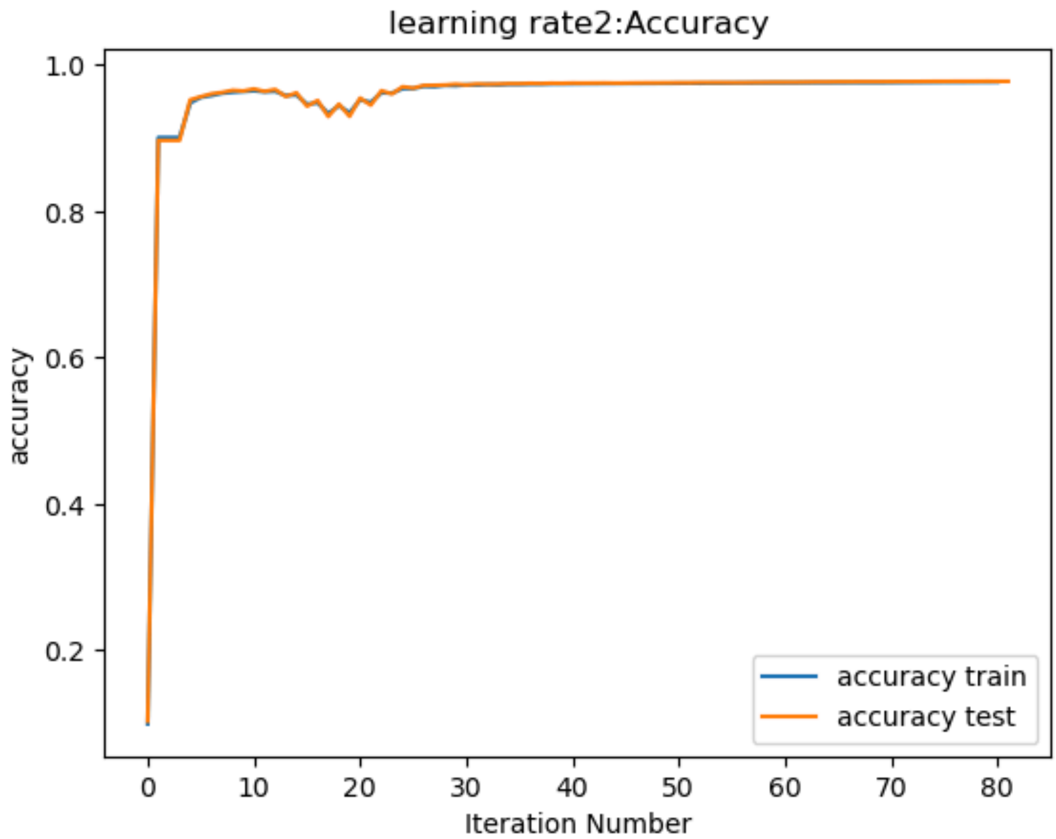

```
Model converged at iter 109
```



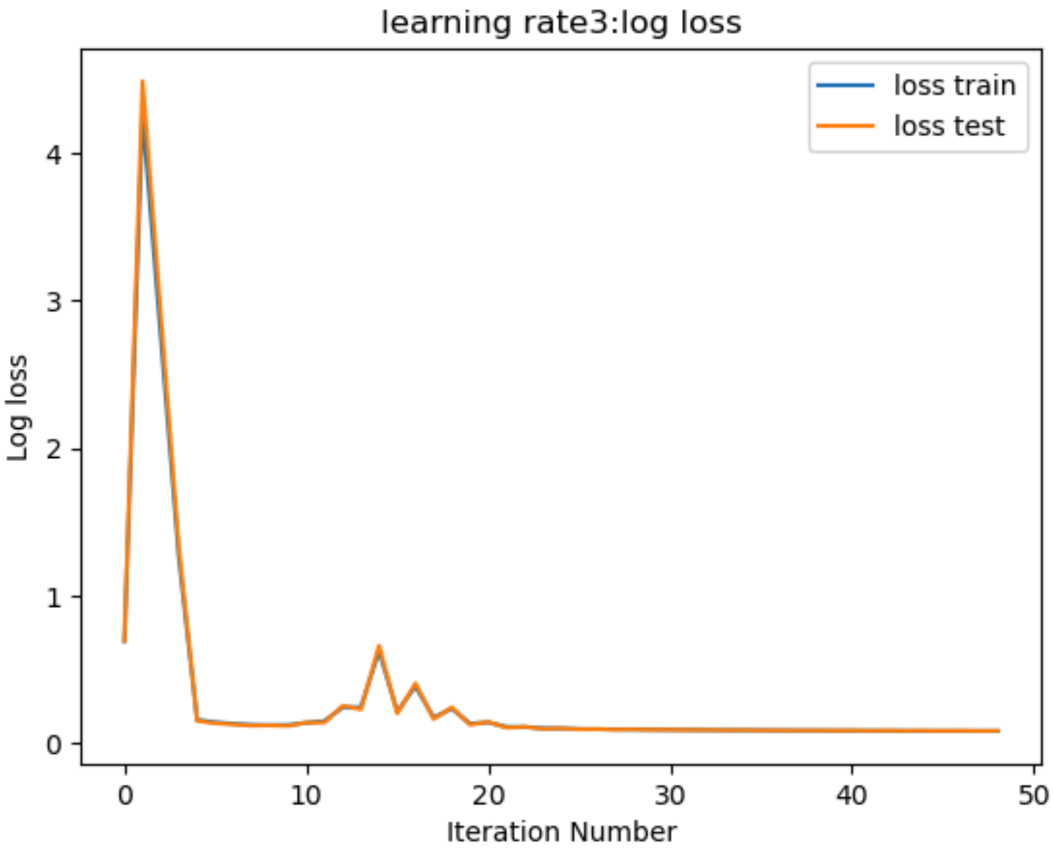


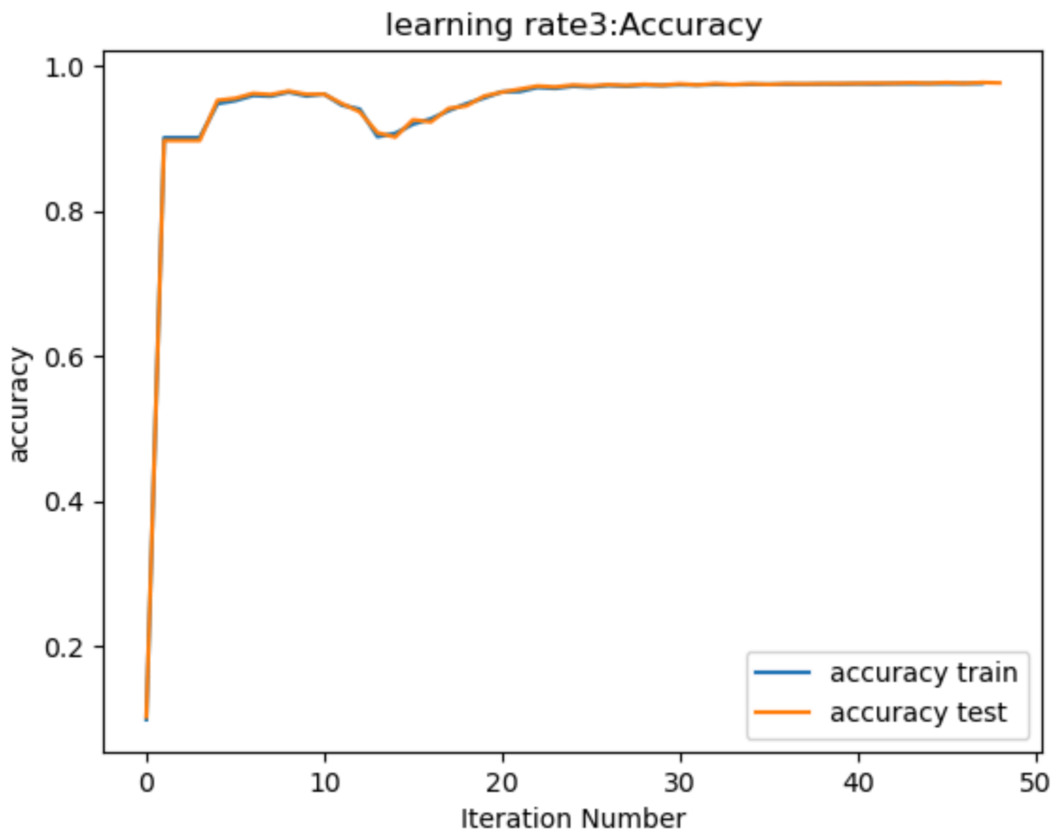

```
Model converged at iter 80
```






```
Model converged at iter 47
```



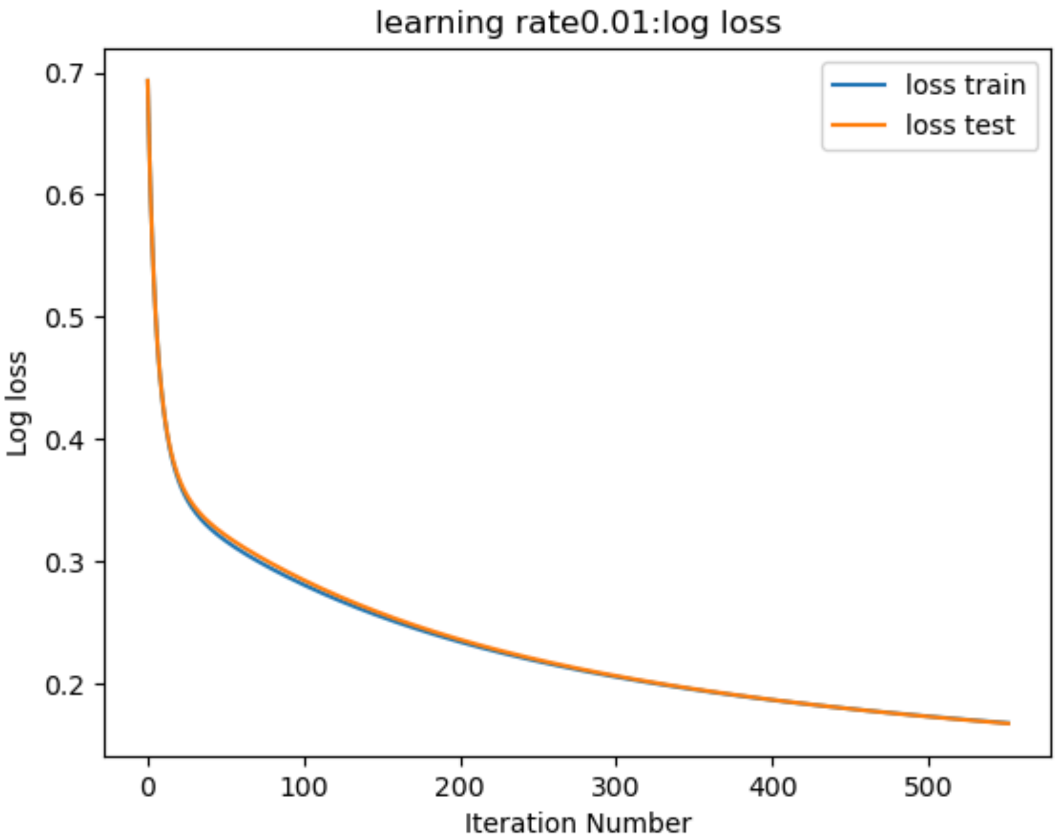


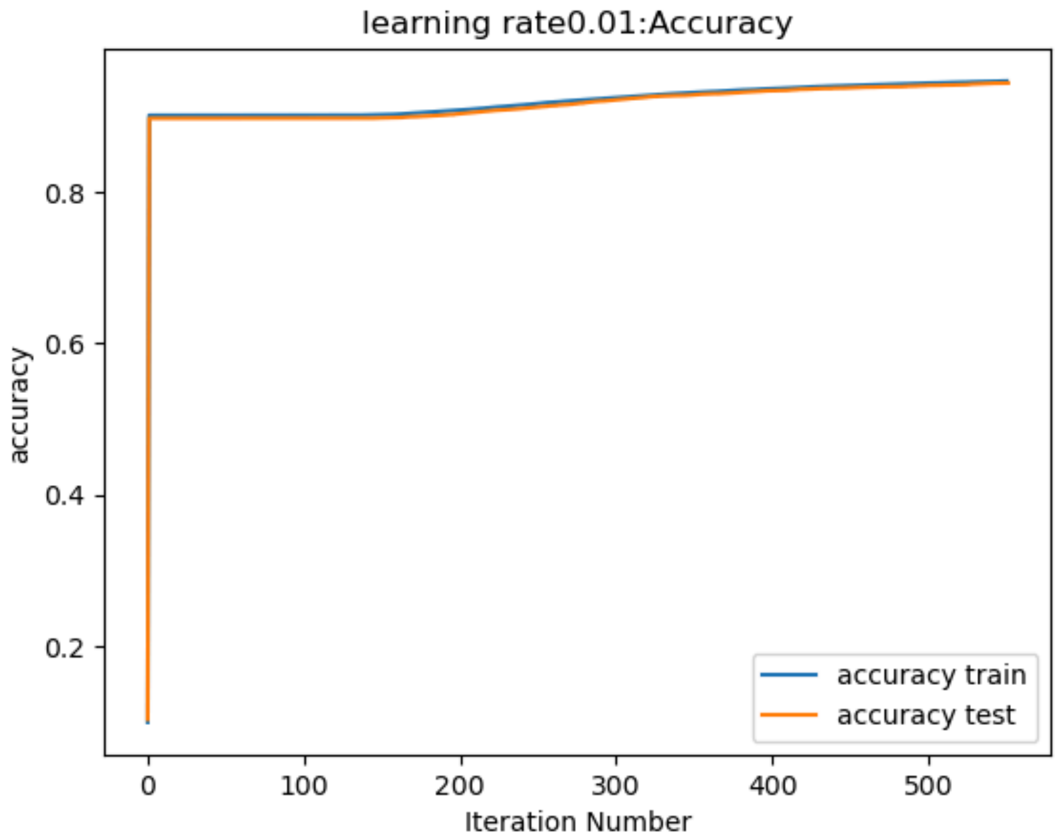
	Learning rate	Iter	Accuracy train	Loss train	Accuracy test	Loss test
0	0.01	550	0.946033	0.167011	0.9435	0.167308
1	0.10	246	0.967400	0.108955	0.9679	0.109749
2	1.00	109	0.975217	0.085216	0.9768	0.084703
3	2.00	80	0.976650	0.081529	0.9775	0.080554
4	3.00	47	0.975333	0.085829	0.9761	0.083927

L1 regularization

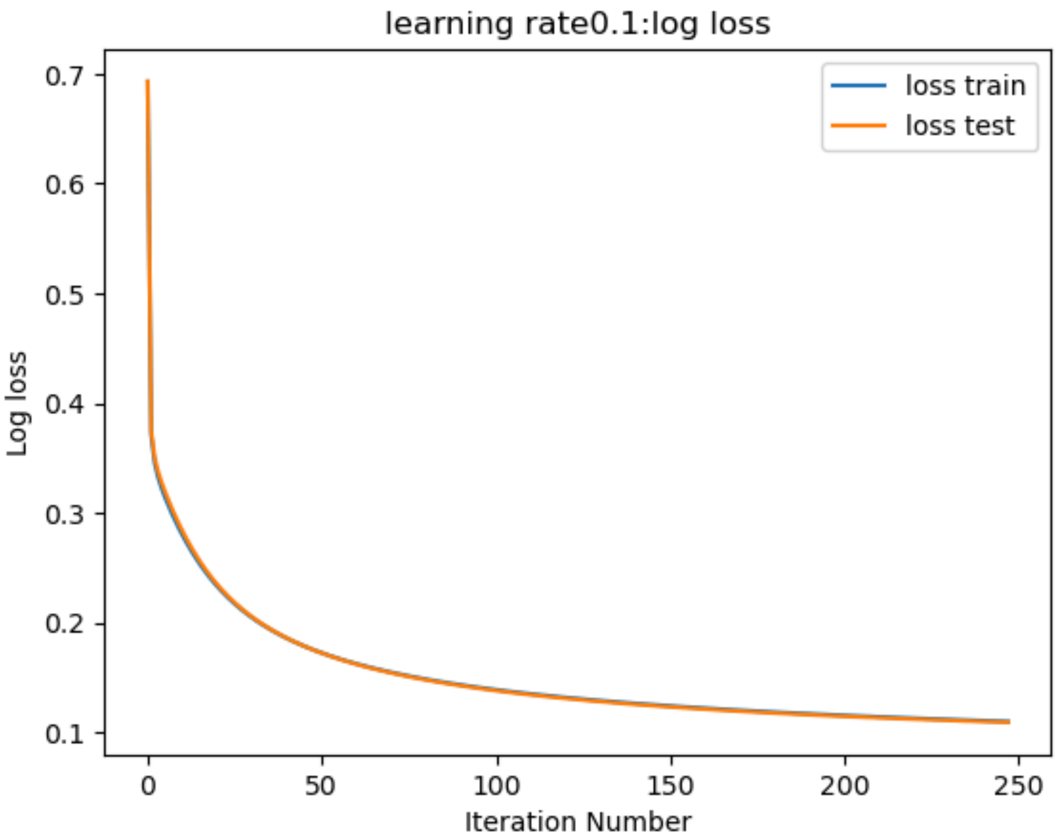
In [240]:
```
logistic_regression(xdata_train,xdata_test,y_train,y_test,max_iter,learning_ra
```

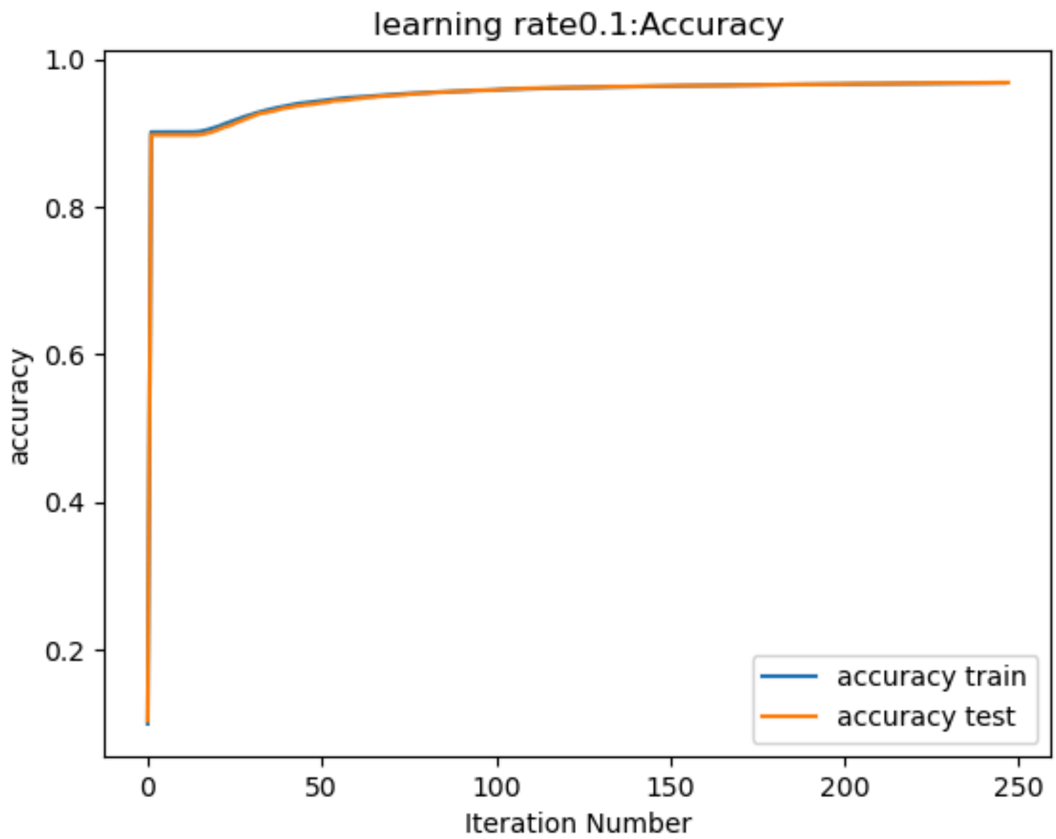
```
Model converged at iter 550
```



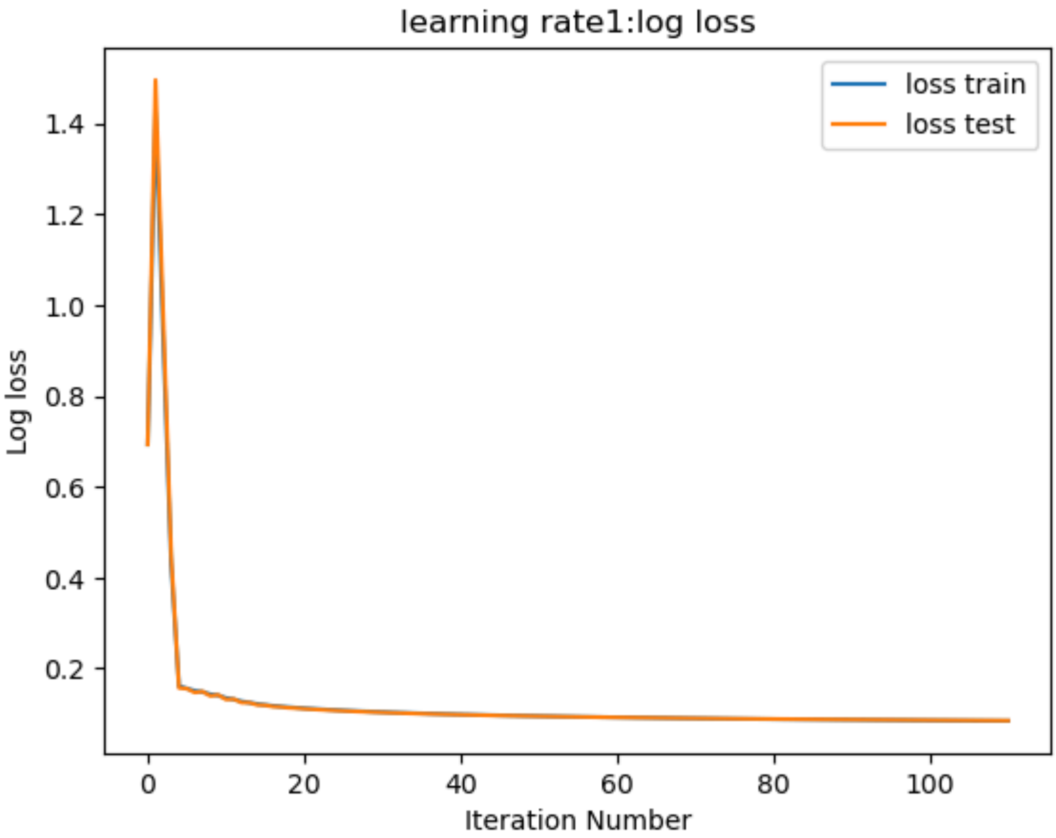


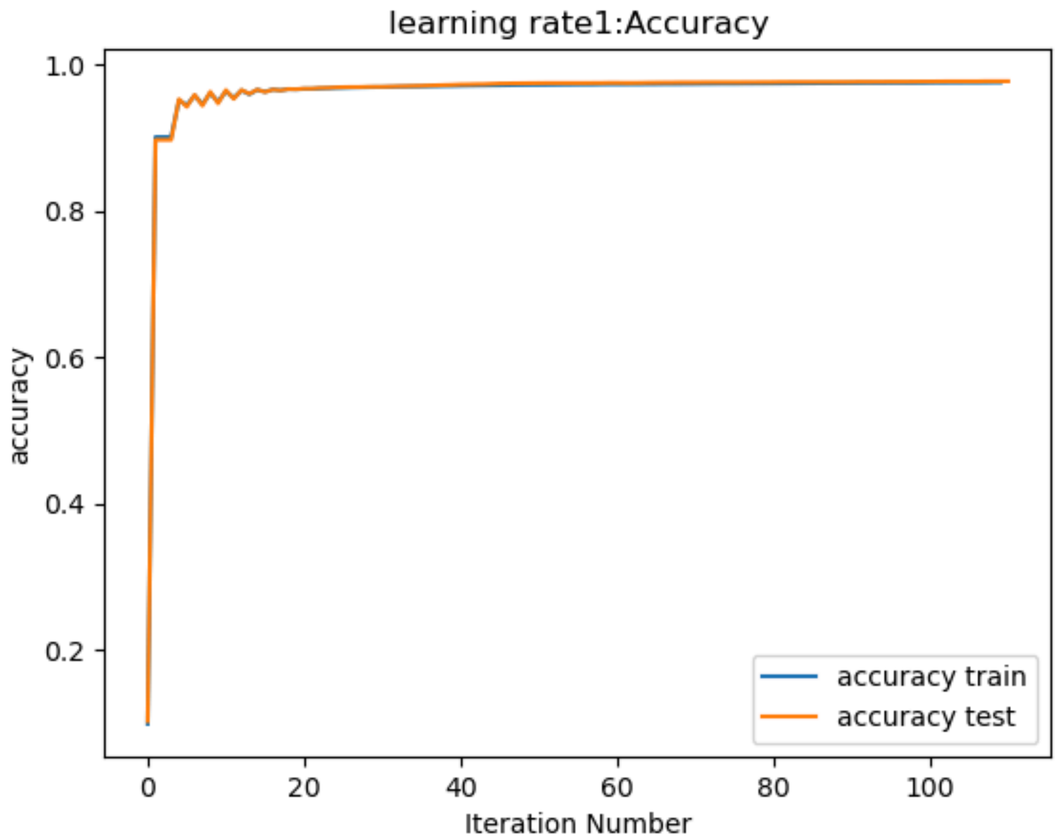
```
Model converged at iter 246
```



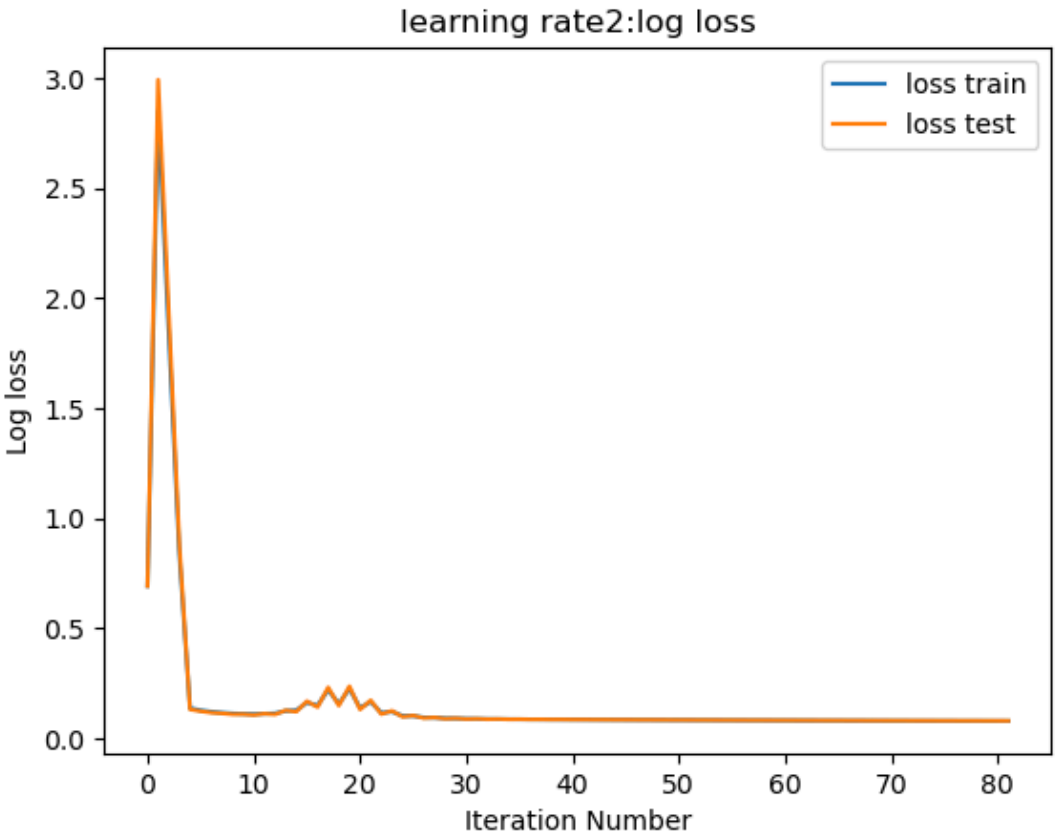


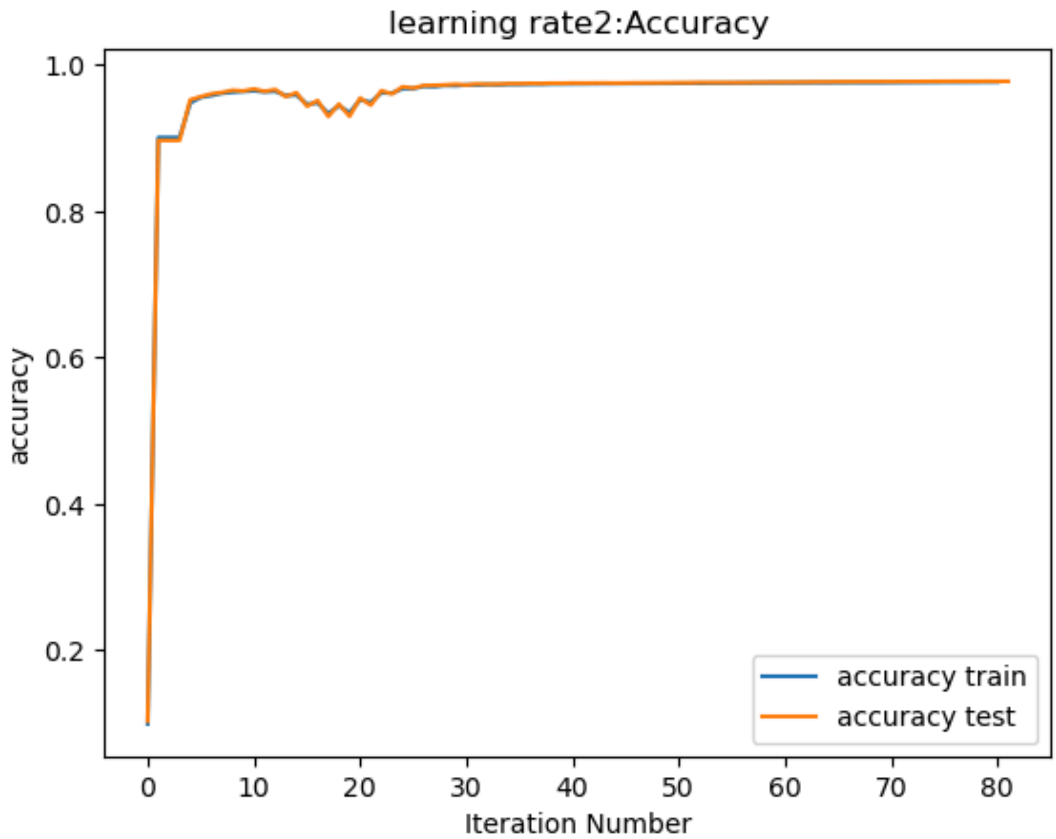

```
Model converged at iter 109
```



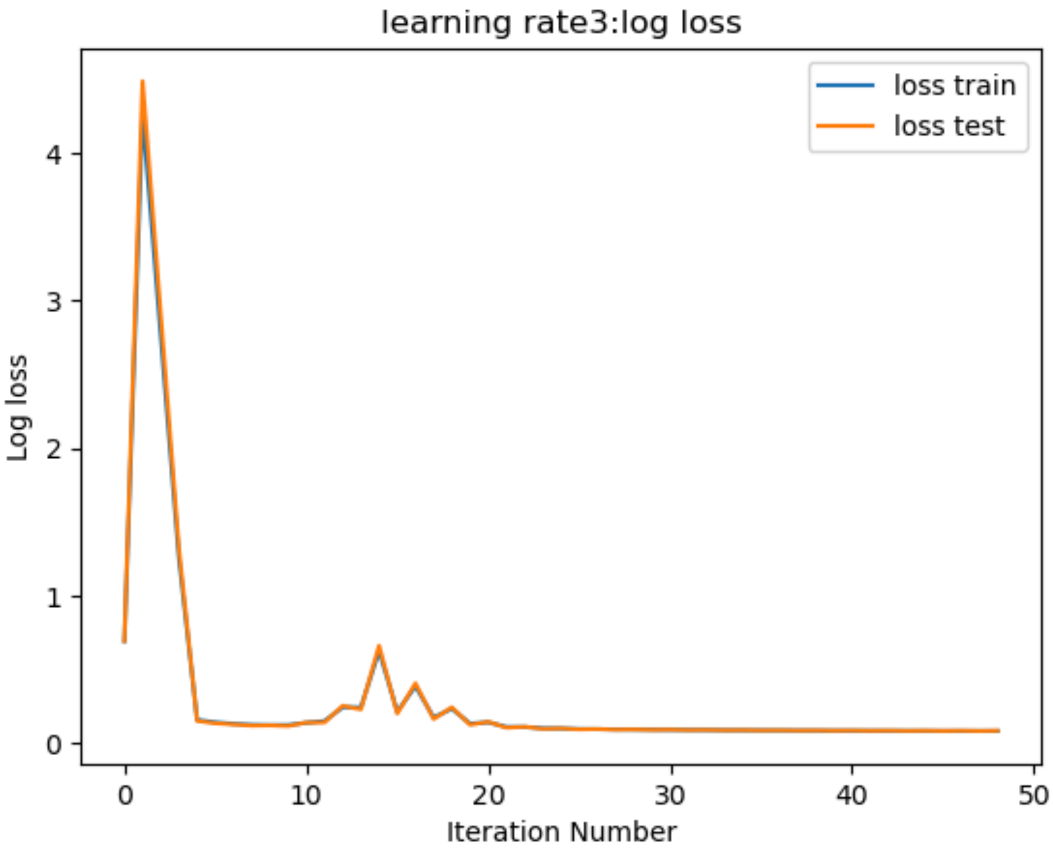


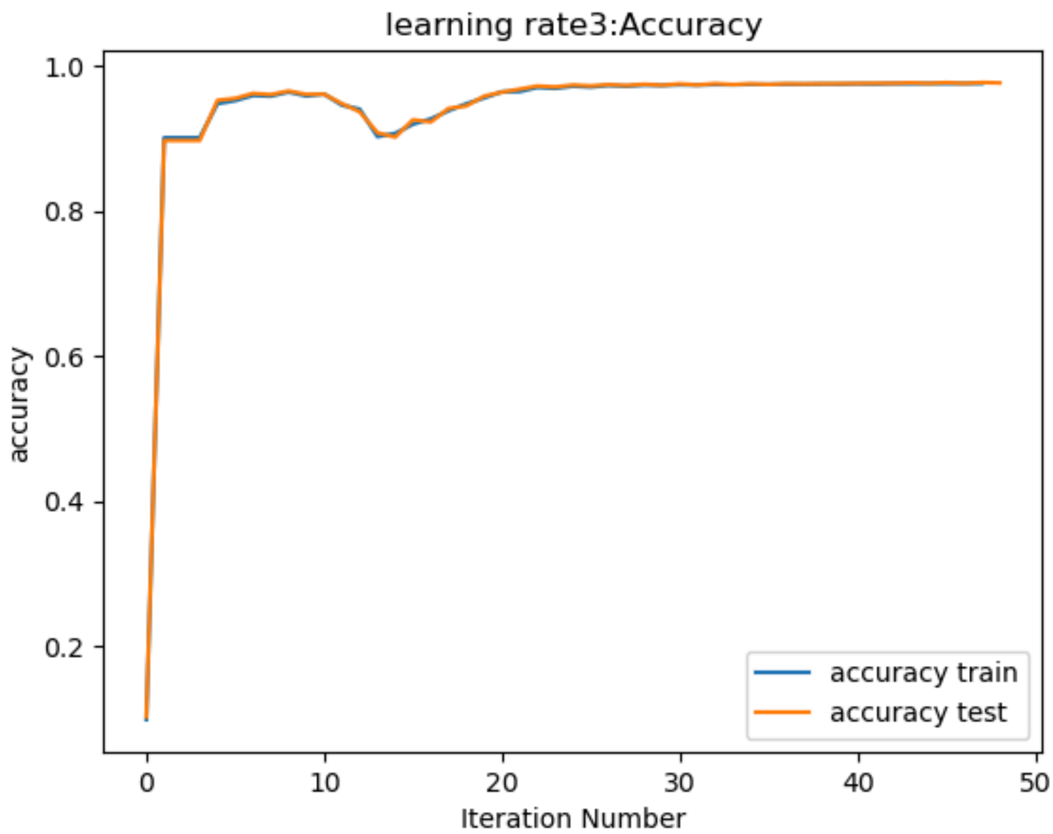

```
Model converged at iter 80
```






```
Model converged at iter 47
```



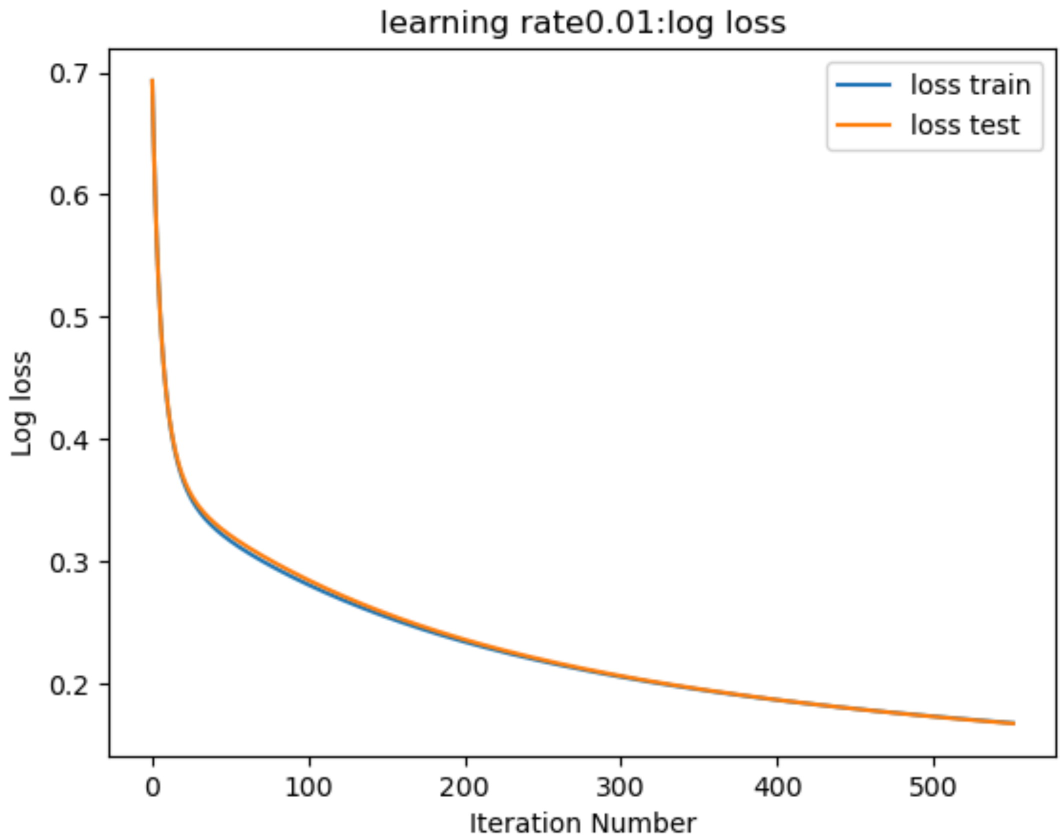


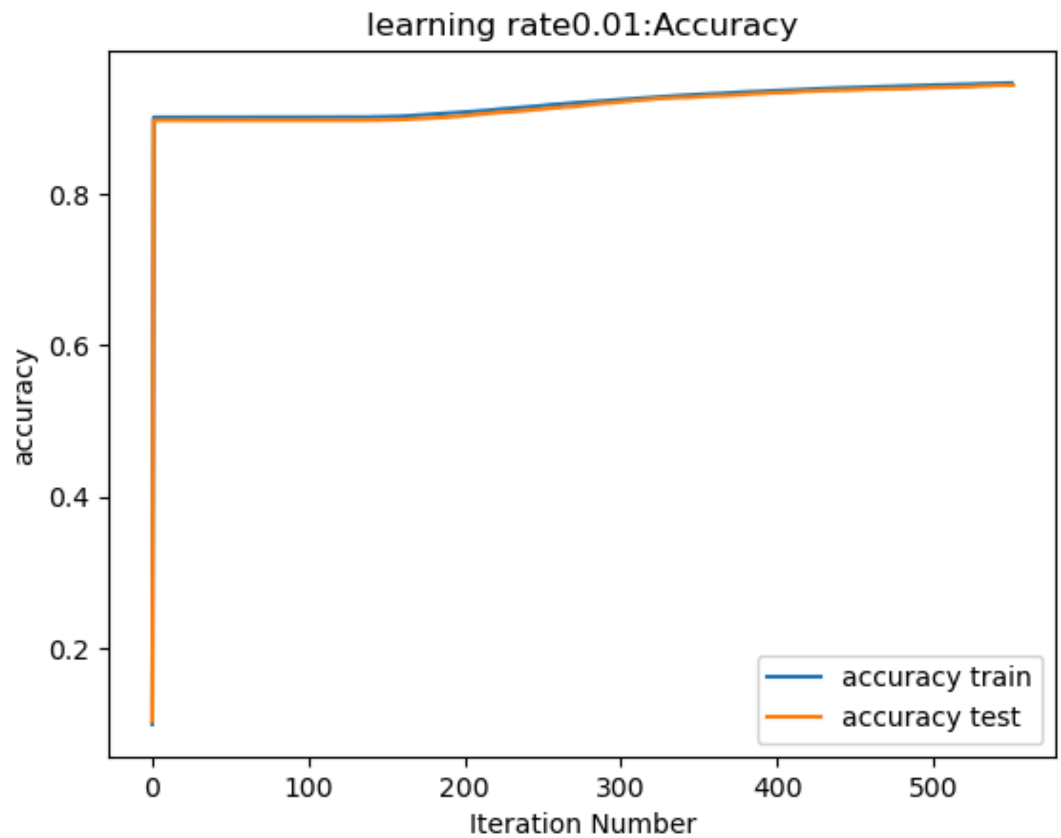
	Learning rate	Iter	Accuracy train	Loss train	Accuracy test	Loss test
0	0.01	550	0.946033	0.167029	0.9435	0.167312
1	0.10	246	0.967400	0.108989	0.9679	0.109755
2	1.00	109	0.975217	0.085270	0.9768	0.084714
3	2.00	80	0.976650	0.081593	0.9775	0.080565
4	3.00	47	0.975333	0.085902	0.9761	0.083939

L2 regularization

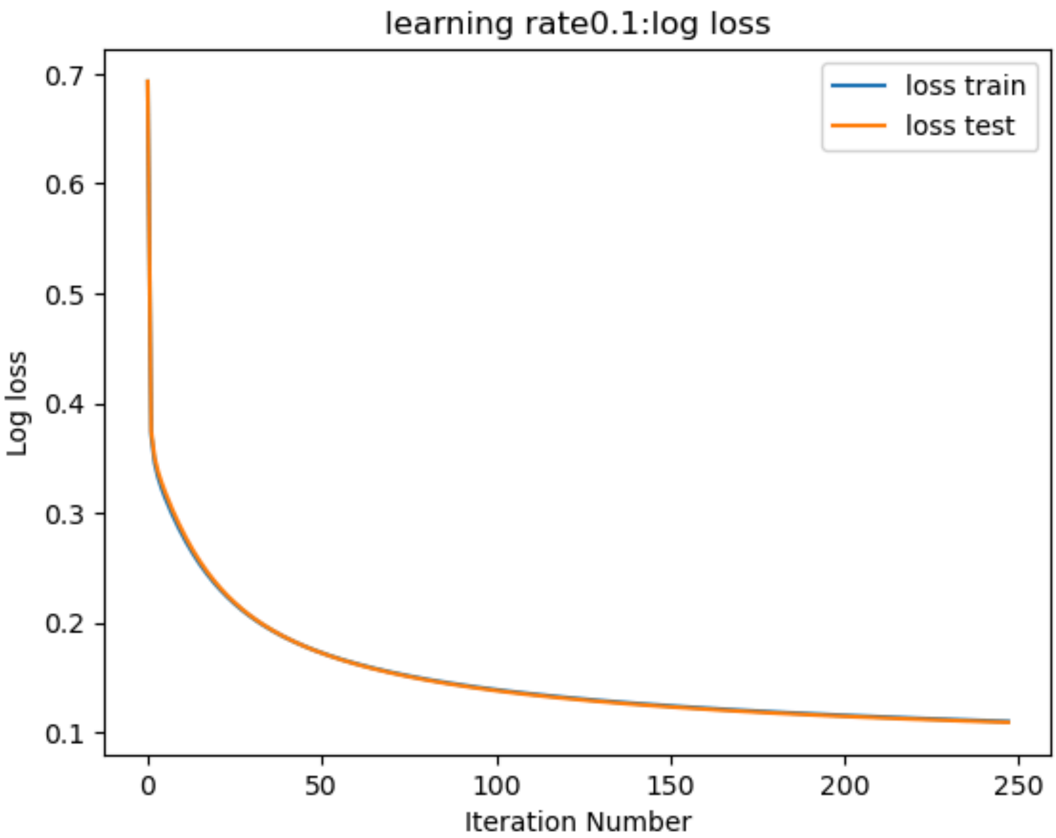
In [241]:
```
logistic_regression(xdata_train,xdata_test,y_train,y_test,max_iter,learning_ra
```

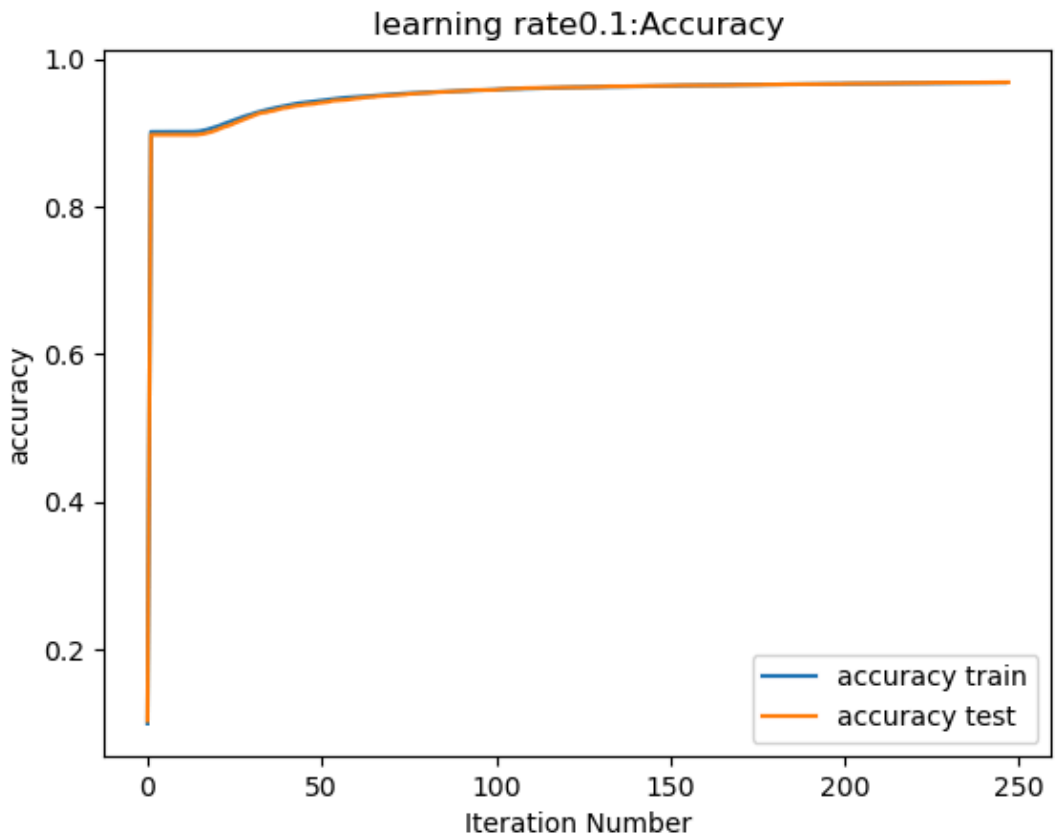
```
Model converged at iter 550
```



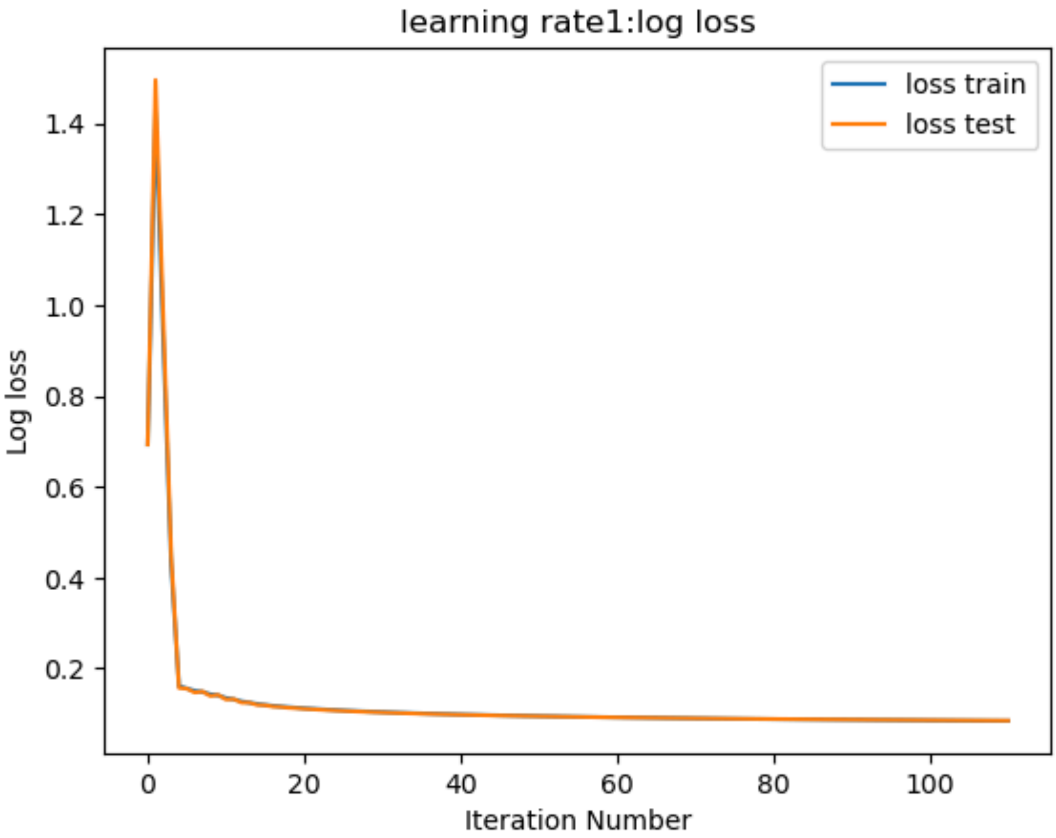


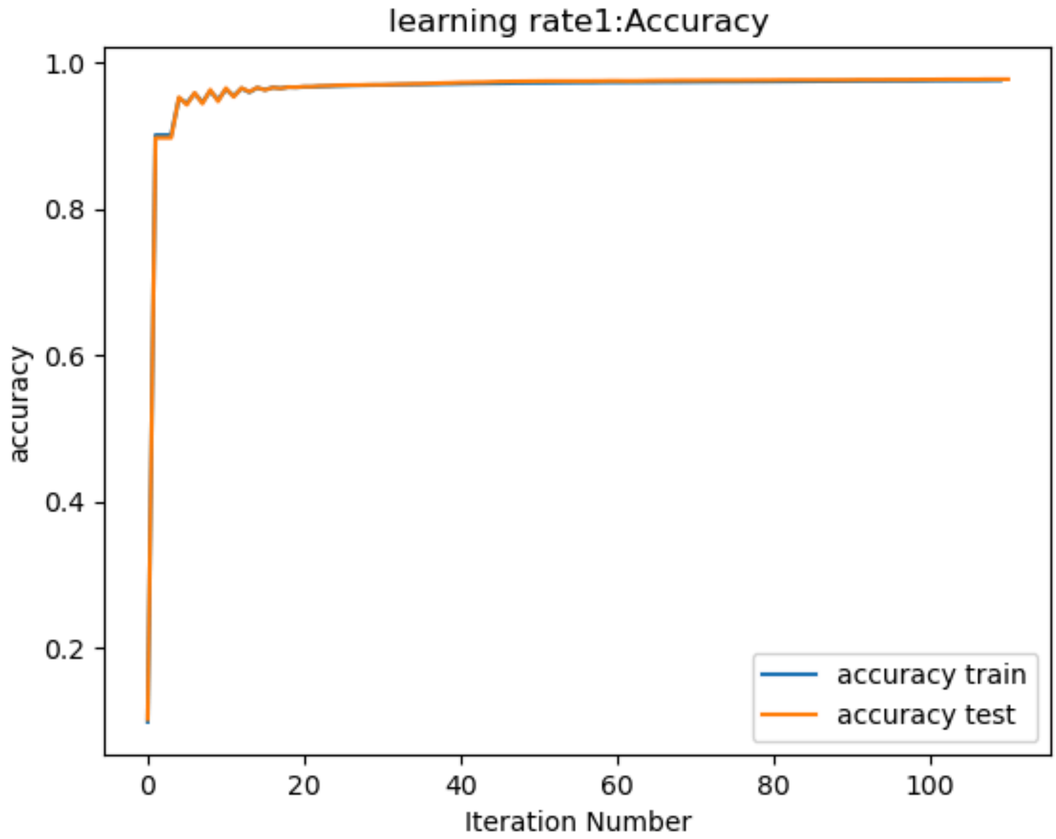
```
Model converged at iter 246
```



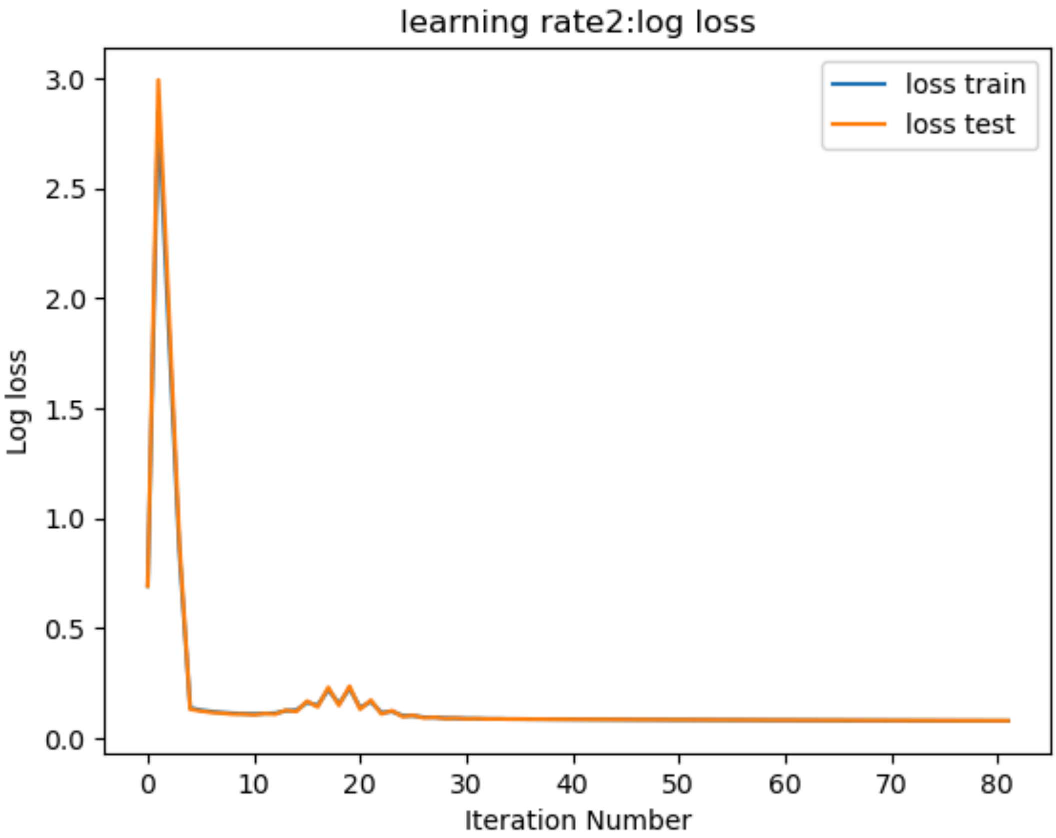


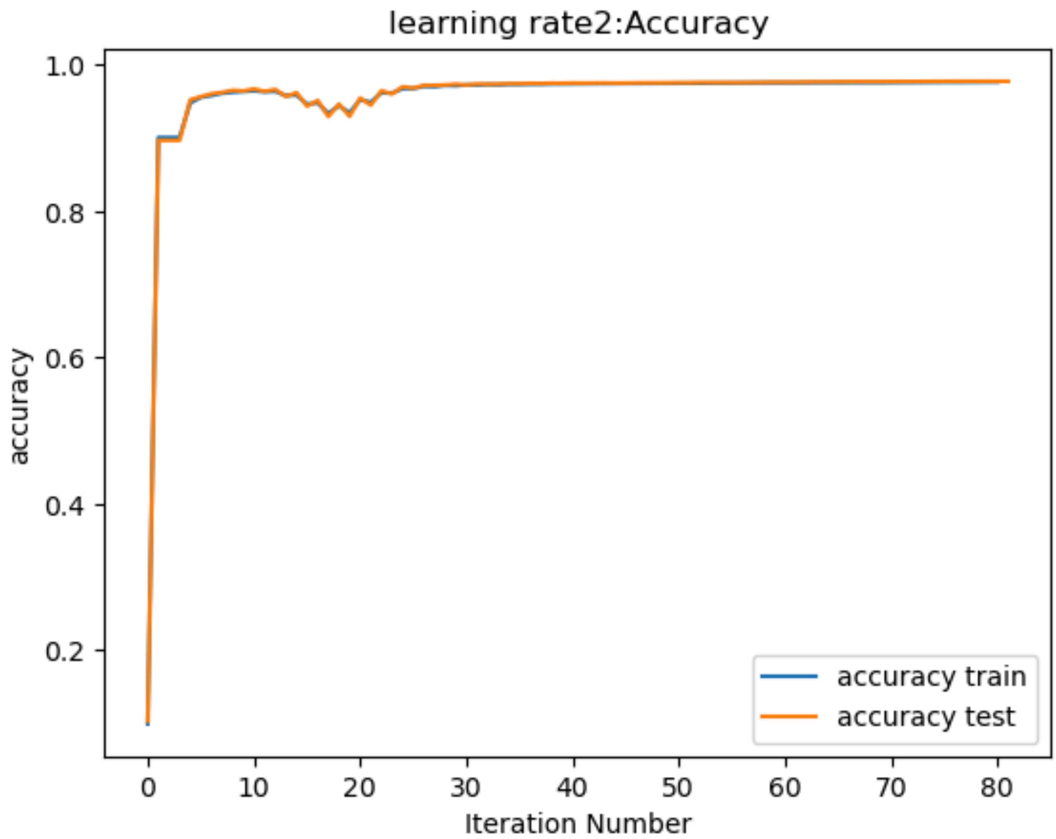

```
Model converged at iter 109
```



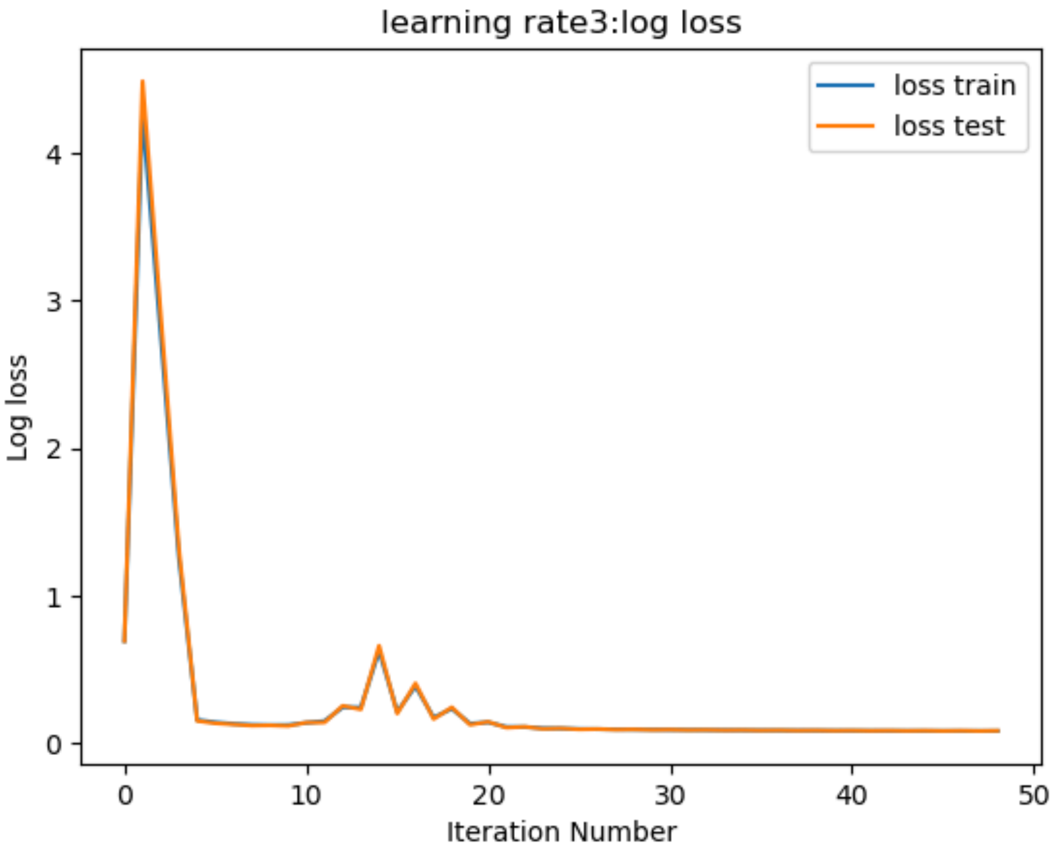


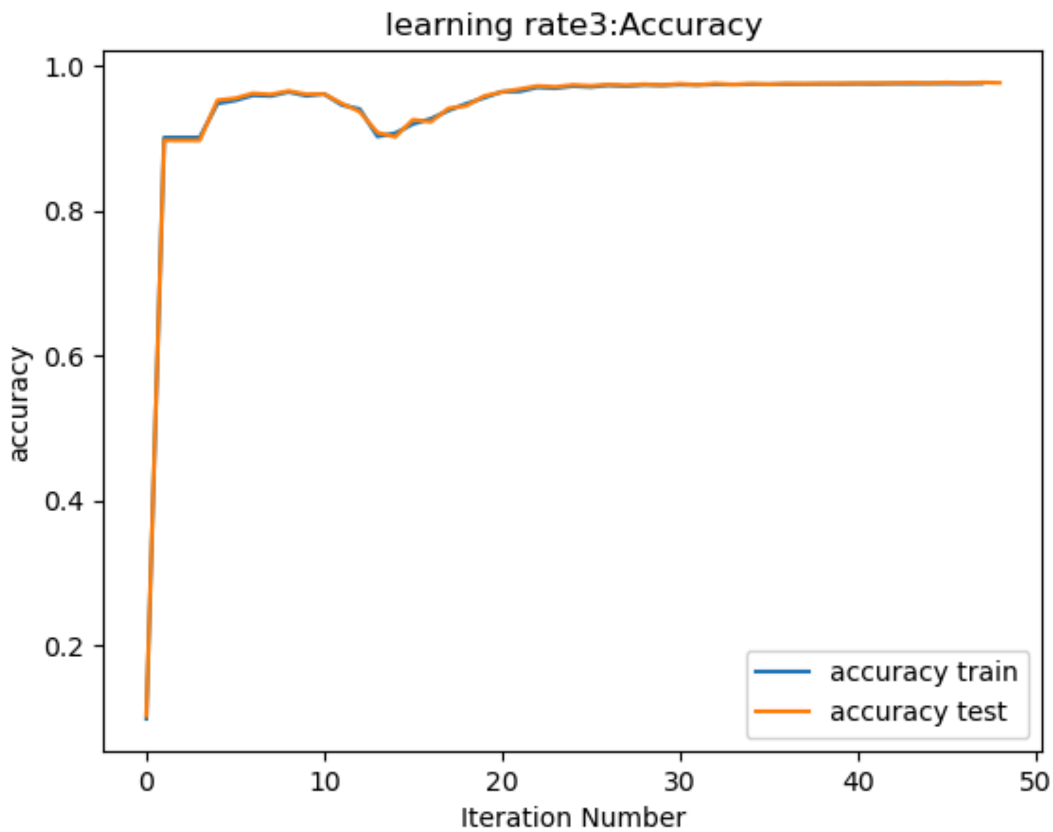

```
Model converged at iter 80
```






```
Model converged at iter 47
```





	Learning rate	Iter	Accuracy train	Loss train	Accuracy test	Loss test
0	0.01	550	0.946033	0.167011	0.9435	0.167308
1	0.10	246	0.967400	0.108957	0.9679	0.109749
2	1.00	109	0.975217	0.085221	0.9768	0.084704
3	2.00	80	0.976650	0.081536	0.9775	0.080555
4	3.00	47	0.975333	0.085837	0.9761	0.083927

Saving weights and bias to hdf5

In [250]:

```
#pick a best result
def gradient_descent1(W,w0,x_train,x_test,y_train,y_test,learning_rate,max_ite
    m=x_train.shape[0]
    threshold=1e-4
    pre_loss_train=0

    for iter in range(max_iter):
        z_train=np.dot(x_train,W)+w0
        p_train=sigmoid(z_train)

        dw=np.dot(x_train.T,(p_train-y_train))/m
        db=np.sum(p_train-y_train)/m

        W-=learning_rate*dw
        w0-=learning_rate*db

        loss_train=binary_log_loss(W,x_train,w0,y_train)

        z_test=np.dot(x_test,W)+w0
        p_test=sigmoid(z_test)
        predictions_test=np.where(p_test>=0.5,1,0)
        accuracy_test=np.mean(predictions_test==y_test)

        if np.abs(loss_train-pre_loss_train)<threshold:
            #print(f"Model converged at iter {iter}")
            break
        pre_loss_train=loss_train
    return W,w0

n=xdata_train.shape[1]
W=np.zeros(n)
w0=0
W,w0=gradient_descent1(W,w0,xdata_train,xdata_test,y_train,y_test,2,max_iter)
```

In [251]:

```
outFile='hw4q2_wb.hd5'
weight_length=784
assert W.shape[0]==weight_length, 'Error: the length is incorrect'

with h5py.File(outFile, 'w') as hf:
    hf.create_dataset('w', data = np.asarray(W))
    hf.create_dataset('b', data = np.asarray(w0))

with h5py.File(outFile,'r') as hf:
    w_copy=hf['w'][:]

np.testing.assert_array_equal(W,w_copy)
```

i. How did you determine a learning rate? What values did you try? What was your final value?

I set a list of learning rates (0.01,0.1,1,1.5,2), let the training model running over each of them. And then comparing the final train and test accuracies in a table to see which one performed the best

ii. Describe the method you used to establish model convergence.

I have set a max number of iterations (= 1000). And if the loss no longer change significantly (the change between two successive iterations less than thethreshold), the iteration stops.

iii. What regularizers did you try? Specifically, how did each impact your model or improve its performance?

I tried no regulariser, L1 and L2 regularisers. However, there was no significant difference in the results among the three methods, possibly because there was no over-fitting

iv. Plot log-loss (i.e., learning curve) of the training set and test set on the same figure. On a separate figure plot the accuracy against iteration number of your model on the training set and test set. Plot each as a function of the iteration number.

Plots have been given above.

v. Clasify each input to the binary output “digit is a 2” using a 0.5 threshold. Compute the final loss and final accuracy for both your training set and test set.

The data is given in the table above.