In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import h5py
from tqdm import tqdm
from sklearn.utils import shuffle

class Linear():
    def __init__(self, in_features, out_features, act=Non
e, act_bkwd=None):
      self.W=np.random.randn(in_features,out_features)*np.
sqrt(2/out_features)
      self.b=np.zeros(out_features)
      self.activation=act
      self.activation_bkwd=act_bkwd

    def __call__(self, x, *args, **kwargs):
        return x@self.W+self.b

class NumpyMLP():
  def __init__(self, train_file, test_file, activation='re
lu'):
    #PART a
    self.data={}
    self.act=activation
    with h5py.File(train_file, 'r') as hf:
        self.data['x_train']=hf['xdata'][:]
        self.data['x_valid']=self.data['x_train'][50000:]
        self.data['x_train']=self.data['x_train'][:50000]
        self.data['y_train']=hf['ydata'][:]
        self.data['y_valid']=self.data['y_train'][50000:]
        self.data['y_train']=self.data['y_train'][:50000]
    with h5py.File(test_file,'r') as hf:
        self.data['x_test']=hf['xdata'][:]
        self.data['y_test']=hf['ydata'][:]
    print('Train set shapes:', self.data['x_train'].shape,
self.data['y_train'].shape)
    print('Valid set shapes:', self.data['x_valid'].shape,
self.data['y_valid'].shape)
    print('Test set shapes:',self.data['x_test'].shape, se
lf.data['y_test'].shape)

    #PART b & c
    if activation=='relu':
      self.linear1=Linear(in_features=784, out_features=51
2, act=self.relu, act_bkwd=self.relu_backward)
      self.linear2=Linear(in_features=512,out_features=64,
act=self.relu, act_bkwd=self.relu_backward)
      self.linear3=Linear(in_features=64, out_features=32,
act=self.relu, act_bkwd=self.relu_backward)
```

```
      self.linear4=Linear(in_features=32, out_features=10)
    else:
      self.linear1=Linear(in_features=784, out_features=51
2, act=self.tanh, act_bkwd=self.tanh_backward)
      self.linear2=Linear(in_features=512,out_features=64,
act=self.tanh, act_bkwd=self.tanh_backward)
      self.linear3=Linear(in_features=64, out_features=32,
act=self.tanh, act_bkwd=self.tanh_backward)
      self.linear4=Linear(in_features=32, out_features=10)

  def relu(self, x):
    return np.clip(x,a_min=0,a_max=None)

  def relu_backward(self,x):
    x[x<=0]=0
    x[x>0]=1
    return x

  def tanh(self,x):
    return np.tanh(x)

  def tanh_backward(self, x):
    out = [1-np.tanh(sample)**2 for sample in x]
    return np.array(out)

  def softmax(self, x):
    out=[np.exp(sample)/np.sum(np.exp(sample)) for sample
in x]
    return np.array(out)

  def save_weights(self, model_path):
    with h5py.File(model_path, 'w') as hf:
      hf.create_dataset('linear_1.weight', data=self.linea
r1.W)
      hf.create_dataset('linear_1.bias', data=self.linear
1.b)
      hf.create_dataset('linear_2.weight', data=self.linea
r2.W)
      hf.create_dataset('linear_2.bias', data=self.linear
2.b)
      hf.create_dataset('linear_3.weight', data=self.linea
r3.W)
      hf.create_dataset('linear_3.bias', data=self.linear
3.b)
      hf.create_dataset('linear_4.weight', data=self.linea
r4.W)
      hf.create_dataset('linear_4.bias', data=self.linear
4.b)
      hf.create_dataset('activation',data=self.act)

  def load_weights(self,model_path):
```

```
    with h5py.File(model_path,'r') as hf:
      self.linear1.W=hf['linear_1.weight'][:]
      self.linear1.b=hf['linear_1.bias'][:]
      self.linear2.W=hf['linear_2.weight'][:]
      self.linear2.b=hf['linear_2.bias'][:]
      self.linear3.W=hf['linear_3.weight'][:]
      self.linear3.b=hf['linear_3.bias'][:]
      self.linear4.W=hf['linear_4.weight'][:]
      self.linear4.b=hf['linear_4.bias'][:]
      if hf['activation']=='relu':
        self.linear1.activation, self.linear1.activation_b
kwd=self.relu, self.relu_backward
        self.linear2.activation, self.linear2.activation_b
kwd=self.relu, self.relu_backward
        self.linear3.activation, self.linear3.activation_b
kwd=self.relu, self.relu_backward
        self.linear4.activation, self.linear4.activation_b
kwd=self.relu, self.relu_backward
      else:
        self.linear1.activation, self.linear1.activation_b
kwd=self.tanh, self.tanh_backward
        self.linear2.activation, self.linear2.activation_b
kwd=self.tanh, self.tanh_backward
        self.linear3.activation, self.linear3.activation_b
kwd=self.tanh, self.tanh_backward
        self.linear4.activation, self.linear4.activation_b
kwd=self.tanh, self.tanh_backward

  def final_train_test(self, epochs, batch_size=50, lr=1e-
3, save_path='model.hdf5'):
    losses=[]
    train_losses=[]
    valid_losses=[]
    train_accuracy=[]
    valid_accuracy=[]
    steps=[]
    best_val_acc=0.0
    full_x_data=np.vstack((mlp.data['x_train'], mlp.data['
x_valid']))
    full_y_data=np.vstack((mlp.data['y_train'], mlp.data['
y_valid']))
    full_x_data, full_y_data = shuffle(full_x_data,full_y_
data)
    for i in tqdm(range(epochs)):
      if i == int(epochs*0.5):
        lr=lr/2
      if i == int(epochs*0.78):
        lr=lr/2
      steps.append(lr)
      running_loss=0.0
      count=0
```

```
      for j in range(0, full_x_data.shape[0], batch_size):
        x_batch=full_x_data[j:j+batch_size]
        x_batch=x_batch/255.
        noise=np.random.normal(0,0.05,x_batch.shape)
        x_batch=x_batch+noise
        y_batch=full_y_data[j:j+batch_size]

        h_1=self.linear1(x_batch)
        a_1=self.linear1.activation(h_1)
        noise=np.random.normal(0,0.05,a_1.shape)
        a_1+=noise
        h_2=self.linear2(a_1)
        a_2=self.linear2.activation(h_2)
        noise=np.random.normal(0,0.05,a_2.shape)
        a_2+=noise
        h_3=self.linear3(a_2)
        a_3=self.linear3.activation(h_3)
        noise=np.random.normal(0,0.05,a_3.shape)
        # a_3+=noise
        h_4=self.linear4(a_3)
        a_4=self.softmax(h_4)

        loss=-np.sum(y_batch*np.log(a_4))/y_batch.shape[0]
        running_loss+=loss
        count+=batch_size

        delta_L = (a_4-y_batch)/x_batch.shape[0]
        da_3 = self.linear3.activation_bkwd(h_3)
        da_2 = self.linear2.activation_bkwd(h_2)
        da_1 = self.linear1.activation_bkwd(h_1)

        delta_3 = np.multiply(da_3, delta_L@self.linear4.
W.T)
        delta_2 = np.multiply(da_2,delta_3@self.linear3.W.
T)
        delta_1 = np.multiply(da_1,delta_2@self.linear2.W.
T)

        # print(np.multiply(delta_L, a_3).shape)
        # print(a_3.shape, y_batch.shape, delta_L.shape, a
_3.shape, self.linear3.W.shape, a_2.shape, a_1.shape, delt
a_2.shape, self.linear2.W.shape)
        self.linear4.W = self.linear4.W - lr* (a_3.T@delta
_L)/x_batch.shape[0]
        self.linear4.b = self.linear4.b - lr* np.average(d
elta_L, axis=0)

        self.linear3.W = self.linear3.W - lr * (a_2.T@delt
a_3)/x_batch.shape[0]
        self.linear3.b = self.linear3.b - lr * np.average
(delta_3,axis=0)
```

```
        self.linear2.W = self.linear2.W - lr * (a_1.T@delt
a_2)/x_batch.shape[0]
        self.linear2.b = self.linear2.b - lr* np.average(d
elta_2,axis=0)

        self.linear1.W = self.linear1.W - lr * (x_batch.T@
delta_1)/x_batch.shape[0]
        self.linear1.b = self.linear1.b - lr * np.average
(delta_1,axis=0)

      losses.append(running_loss/count)
      y_pred_train=self.softmax(self.linear4(self.linear3.
activation(self.linear3(self.linear2.activation(self.linea
r2(self.linear1.activation(self.linear1(full_x_data/25
5.))))))))
      y_pred_logits=np.argmax(y_pred_train,axis=1)
      y_gt_logits=np.argmax(full_y_data,axis=1)
      train_accuracy.append((y_pred_logits==y_gt_logits).s
um()/y_gt_logits.shape[0])
      train_losses.append(-np.sum(full_y_data*np.log(y_pre
d_train))/y_pred_train.shape[0])
    y_pred_test=self.softmax(self.linear4(self.linear3.act
ivation(self.linear3(self.linear2.activation(self.linear2
(self.linear1.activation(self.linear1(self.data['x_test']/
255.))))))))
    y_pred_test_logits=np.argmax(y_pred_test,axis=1)
    y_gt_test_logits=np.argmax(self.data['y_test'],axis=1)
    print('Final test accuracy:',(y_pred_test_logits==y_gt
_test_logits).sum()/y_gt_test_logits.shape[0])
    return losses, train_losses, valid_losses, train_accur
acy, valid_accuracy, steps


  def train(self, epochs, batch_size=50, lr=1e-3, save_pat
h='model.hdf5'):
    losses=[]
    train_losses=[]
    valid_losses=[]
    train_accuracy=[]
    valid_accuracy=[]
    steps=[]
    best_val_acc=0.0
    for i in tqdm(range(epochs)):
      if i == int(epochs*0.5):
        lr=lr/2
      if i == int(epochs*0.78):
        lr=lr/2
      steps.append(lr)
      running_loss=0.0
      count=0
```

```
      for j in range(0, self.data['x_train'].shape[0], bat
ch_size):
        x_batch=self.data['x_train'][j:j+batch_size]
        x_batch=x_batch/255.
        noise=np.random.normal(0,0.05,x_batch.shape)
        x_batch=x_batch+noise
        y_batch=self.data['y_train'][j:j+batch_size]

        h_1=self.linear1(x_batch)
        a_1=self.linear1.activation(h_1)
        noise=np.random.normal(0,0.05,a_1.shape)
        a_1+=noise
        h_2=self.linear2(a_1)
        a_2=self.linear2.activation(h_2)
        noise=np.random.normal(0,0.05,a_2.shape)
        a_2+=noise
        h_3=self.linear3(a_2)
        a_3=self.linear3.activation(h_3)
        noise=np.random.normal(0,0.05,a_3.shape)
        # a_3+=noise
        h_4=self.linear4(a_3)
        a_4=self.softmax(h_4)

        loss=-np.sum(y_batch*np.log(a_4))/y_batch.shape[0]
        running_loss+=loss
        count+=batch_size

        delta_L = (a_4-y_batch)/x_batch.shape[0]
        da_3 = self.linear3.activation_bkwd(h_3)
        da_2 = self.linear2.activation_bkwd(h_2)
        da_1 = self.linear1.activation_bkwd(h_1)

        delta_3 = np.multiply(da_3, delta_L@self.linear4.
W.T)
        delta_2 = np.multiply(da_2,delta_3@self.linear3.W.
T)
        delta_1 = np.multiply(da_1,delta_2@self.linear2.W.
T)

        # print(np.multiply(delta_L, a_3).shape)
        # print(a_3.shape, y_batch.shape, delta_L.shape, a
_3.shape, self.linear3.W.shape, a_2.shape, a_1.shape, delt
a_2.shape, self.linear2.W.shape)
        self.linear4.W = self.linear4.W - lr* (a_3.T@delta
_L)/x_batch.shape[0]
        self.linear4.b = self.linear4.b - lr* np.average(d
elta_L, axis=0)

        self.linear3.W = self.linear3.W - lr * (a_2.T@delt
a_3)/x_batch.shape[0]
        self.linear3.b = self.linear3.b - lr * np.average
```

```
(delta_3,axis=0)

        self.linear2.W = self.linear2.W - lr * (a_1.T@delt
a_2)/x_batch.shape[0]
        self.linear2.b = self.linear2.b - lr* np.average(d
elta_2,axis=0)

        self.linear1.W = self.linear1.W - lr * (x_batch.T@
delta_1)/x_batch.shape[0]
        self.linear1.b = self.linear1.b - lr * np.average
(delta_1,axis=0)

      losses.append(running_loss/count)
      y_pred_train=self.softmax(self.linear4(self.linear3.
activation(self.linear3(self.linear2.activation(self.linea
r2(self.linear1.activation(self.linear1(self.data['x_train
']/255.))))))))
      y_pred_logits=np.argmax(y_pred_train,axis=1)
      y_gt_logits=np.argmax(self.data['y_train'],axis=1)
      train_accuracy.append((y_pred_logits==y_gt_logits).s
um()/y_gt_logits.shape[0])
      train_losses.append(-np.sum(self.data['y_train']*np.
log(y_pred_train))/y_pred_train.shape[0])
      y_pred_valid=self.softmax(self.linear4(self.linear3.
activation(self.linear3(self.linear2.activation(self.linea
r2(self.linear1.activation(self.linear1(self.data['x_valid
']/255.))))))))
      y_pred_valid_logits=np.argmax(y_pred_valid,axis=1)
      y_gt_valid_logits=np.argmax(self.data['y_valid'],axi
s=1)
      valid_losses.append(-np.sum(self.data['y_valid']*np.
log(y_pred_valid))/y_pred_valid.shape[0])
      valid_accuracy.append((y_pred_valid_logits==y_gt_val
id_logits).sum()/y_gt_valid_logits.shape[0])
      if valid_accuracy[-1]>best_val_acc:
        best_val_acc=valid_accuracy[-1]
        self.save_weights(save_path)
    return losses, train_losses, valid_losses, train_accur
acy, valid_accuracy, steps
```

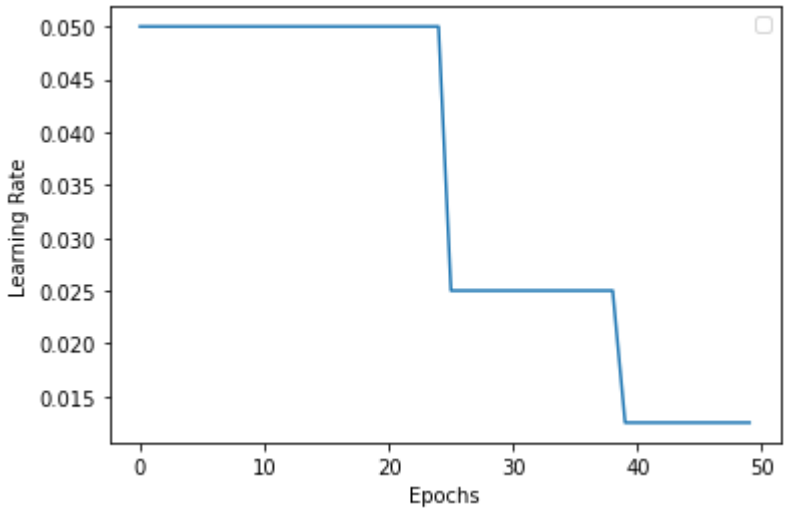
In [ ]:
```
mlp=NumpyMLP('mnist_traindata.hdf5','mnist_testdata.hdf5',
activation='relu')
losses,train_losses, valid_losses, train_accuracy,valid_ac
curacy, lr_steps =mlp.train(epochs=50, lr=5e-2, save_path=
'model_relu_5e-2.hdf5')
valid_accuracy[-1]
```

```
 78%|███████▊  | 39/50 [11:01<03:05, 16.90s/it]

 80%|████████  | 40/50 [11:18<02:48, 16.83s/it]

 82%|████████▏ | 41/50 [11:35<02:31, 16.88s/it]

 84%|████████▍ | 42/50 [11:53<02:17, 17.16s/it]

 86%|████████▌ | 43/50 [12:10<01:59, 17.05s/it]

 88%|████████▊ | 44/50 [12:26<01:41, 16.96s/it]
```

Out[ ]: 0.9573

In [ ]:
```
plt.plot(lr_steps)
plt.xlabel('Epochs')
plt.ylabel('Learning Rate')
plt.legend()
plt.show()
print(set(lr_steps))
```

```
No handles with labels found to put in legend.
```



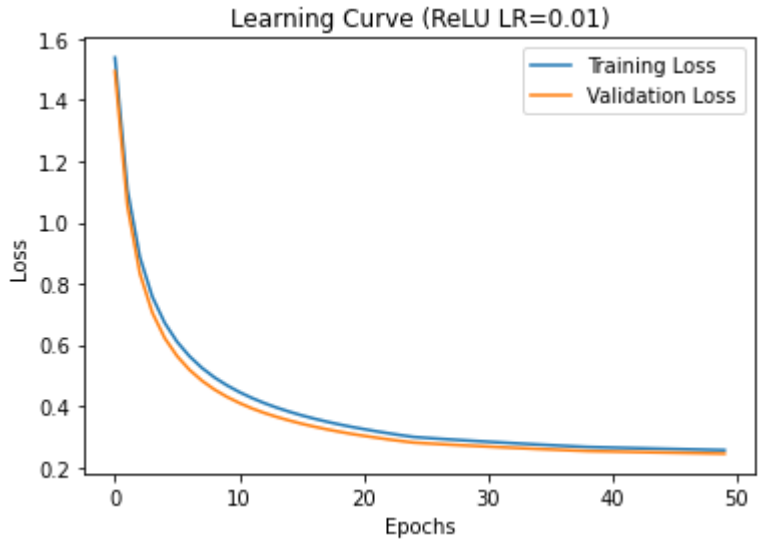

```
{0.05, 0.0125, 0.025}
```

In [ ]:
```
model_relu_lr5e2=[losses,train_losses,valid_losses,train_a
ccuracy, valid_accuracy, lr_steps]
```

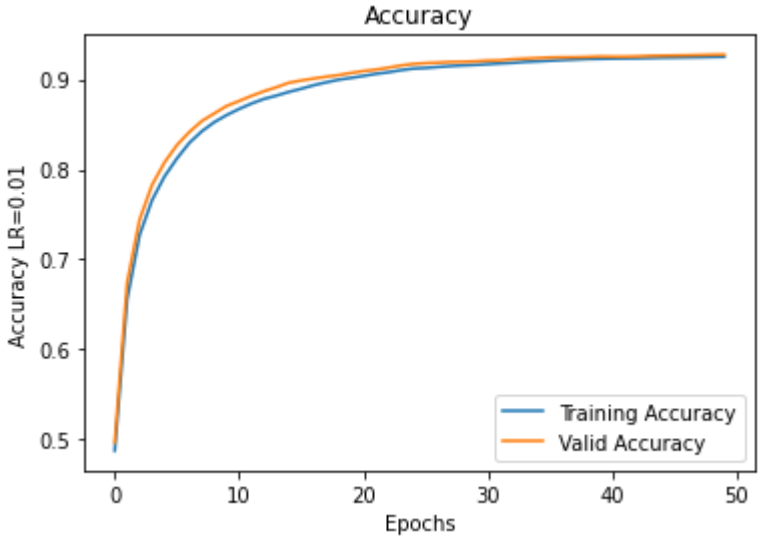
In [ ]:
```
mlp=NumpyMLP('mnist_traindata.hdf5','mnist_testdata.hdf5',
activation='relu')
losses,train_losses, valid_losses, train_accuracy,valid_ac
curacy, lr_steps =mlp.train(epochs=50, lr=1e-2, save_path=
'model_relu_1e-2.hdf5')
valid_accuracy[-1]
```

In [ ]:
```
model_relu_lr1e2=[losses,train_losses,valid_losses,train_a
ccuracy, valid_accuracy, lr_steps]
```

In [ ]:
```
plt.plot(train_losses, label='Training Loss')
plt.plot(valid_losses, label='Validation Loss')
plt.title('Learning Curve (ReLU LR=0.01)')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

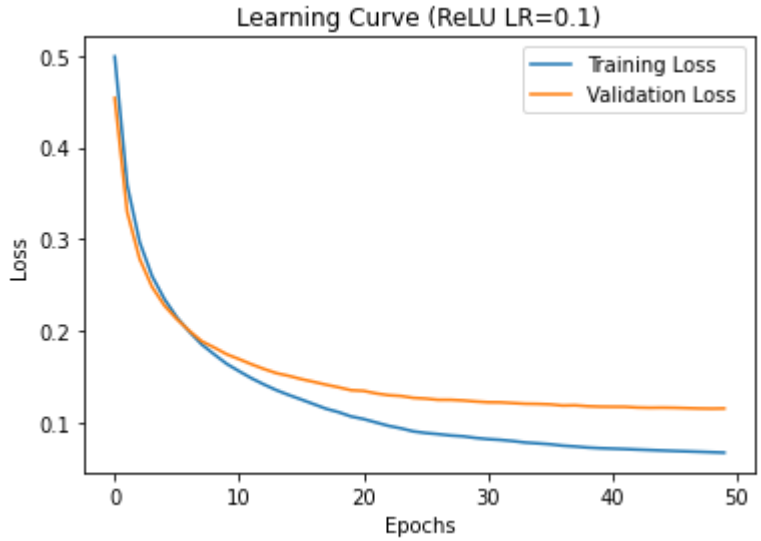


In [ ]:
```
plt.plot(train_accuracy, label='Training Accuracy')
plt.plot(valid_accuracy, label='Valid Accuracy')
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy LR=0.01')
plt.legend()
plt.show()
```

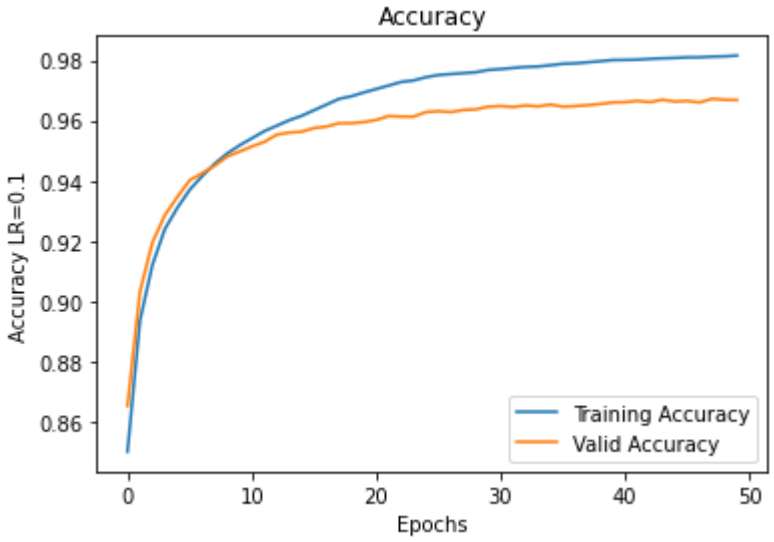


In [ ]:
```
mlp=NumpyMLP('mnist_traindata.hdf5','mnist_testdata.hdf5',
activation='relu')
losses,train_losses, valid_losses, train_accuracy,valid_ac
curacy, lr_steps =mlp.train(epochs=50, lr=1e-1, save_path=
'model_relu_1e-3.hdf5')
model_relu_lr1e1=[losses,train_losses,valid_losses,train_a
ccuracy, valid_accuracy, lr_steps]
valid_accuracy[-1]
```

In [ ]:
```
plt.plot(train_losses, label='Training Loss')
plt.plot(valid_losses, label='Validation Loss')
plt.title('Learning Curve (ReLU LR=0.1)')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [ ]:
```
plt.plot(train_accuracy, label='Training Accuracy')
plt.plot(valid_accuracy, label='Valid Accuracy')
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy LR=0.1')
plt.legend()
plt.show()
```



In [ ]:
```
model_relu_lr1e1=model_relu_lr1e3
```

In [ ]:
```
mlp=NumpyMLP('mnist_traindata.hdf5','mnist_testdata.hdf5',
activation='tanh')
losses,train_losses, valid_losses, train_accuracy,valid_ac
curacy, lr_steps =mlp.train(epochs=50, lr=5e-2, save_path=
'model_tanh_5e-2.hdf5')
valid_accuracy[-1]
```

In [ ]:
```
model_tanh_lr5e2=[losses,train_losses,valid_losses,train_a
ccuracy, valid_accuracy, lr_steps]
```

In [ ]:
```
mlp=NumpyMLP('mnist_traindata.hdf5','mnist_testdata.hdf5',
activation='tanh')
losses,train_losses, valid_losses, train_accuracy,valid_ac
curacy, lr_steps =mlp.train(epochs=50, lr=1e-1, save_path=
'model_tanh_1e-1.hdf5')
valid_accuracy[-1]
```

In [ ]:
```
model_tanh_lr1e1=[losses,train_losses,valid_losses,train_a
ccuracy, valid_accuracy, lr_steps]
```

In [ ]:
```
print(model_relu_lr1e1[-2][-1])
print(model_relu_lr1e2[-2][-1])
print(model_relu_lr5e2[-2][-1])
print(model_tanh_lr1e1[-2][-1])
print(model_tanh_lr5e2[-2][-1])
```

```
0.9668
0.9276
0.9573
0.9599
0.9494
```

In [ ]:
```
mlp=NumpyMLP('mnist_traindata.hdf5','mnist_testdata.hdf5',
activation='relu')
losses,train_losses, valid_losses, train_accuracy,valid_ac
curacy, lr_steps =mlp.final_train_test(epochs=50, lr=1e-1)
```

```
78%|███████▊  | 39/50 [13:16<03:37, 19.73s/it]

80%|████████  | 40/50 [13:35<03:17, 19.76s/it]

82%|████████▏ | 41/50 [13:55<02:57, 19.73s/it]

84%|████████▍ | 42/50 [14:15<02:37, 19.71s/it]

86%|████████▌ | 43/50 [14:35<02:18, 19.84s/it]

88%|████████▊ | 44/50 [14:55<01:58, 19.79s/it]

Final test accuracy: 0.9673
```

ReLU, LR = 0.1

In [92]:
```
settings=['ReLU LR=0.1','ReLU LR=1e-2','ReLU LR=5e-2','Tan
h LR=0.1', 'Tanh LR=5e-2']
for i,model in enumerate([model_relu_lr1e1,model_relu_lr1e
2,model_relu_lr5e2,model_tanh_lr1e1,model_tanh_lr5e2]):
  losses,train_losses, valid_losses, train_accuracy,valid_
accuracy, lr_steps = model
  plt.plot(train_losses, label='Training Loss')
  plt.plot(valid_losses, label='Validation Loss')
  plt.title('Learning Curve ('+settings[i]+')')
  plt.xlabel('Epochs')
  plt.ylabel('Loss')
  plt.legend()
  plt.show()
  plt.figure()
  plt.plot(train_accuracy, label='Training Accuracy')
  plt.plot(valid_accuracy, label='Valid Accuracy')
  plt.title('Accuracy '+settings[i])
  plt.xlabel('Epochs')
  plt.ylabel('Accuracy')
  plt.legend()
  plt.show()
```

Network Config:

Layers = 4 Neurons [512, 64, 32, 10]

Batch size = 16

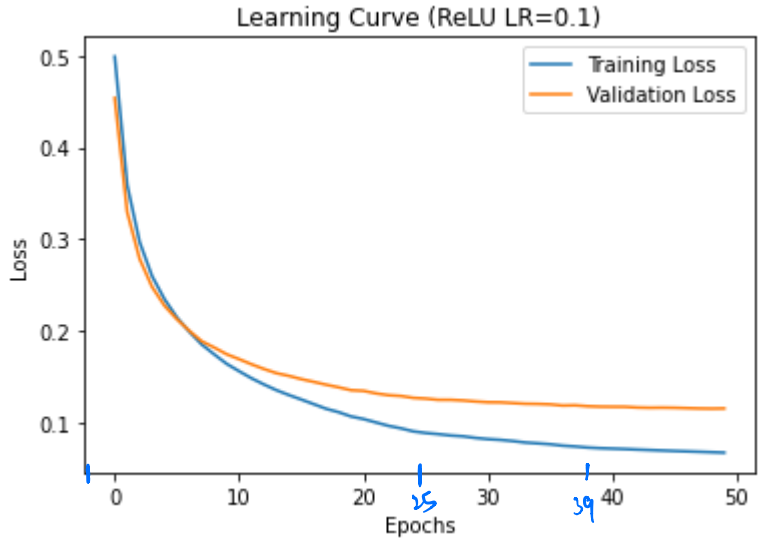
3 Values of LR 0.1, 0.01, 0.05

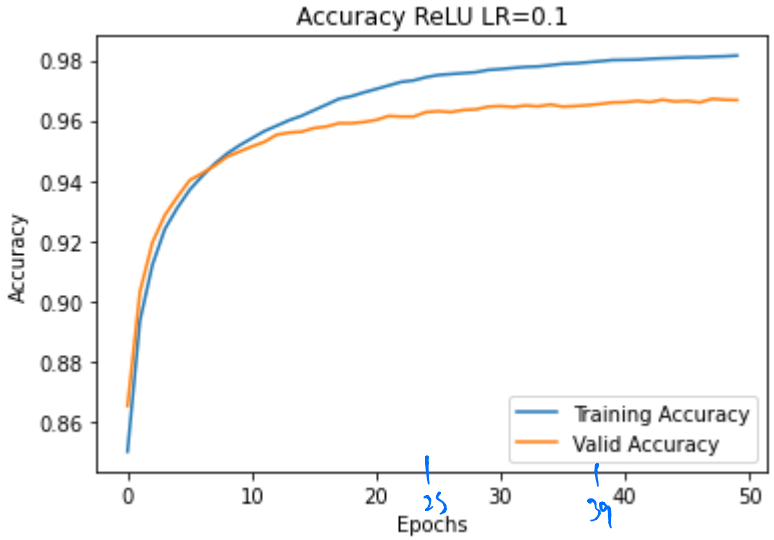
Param Init training

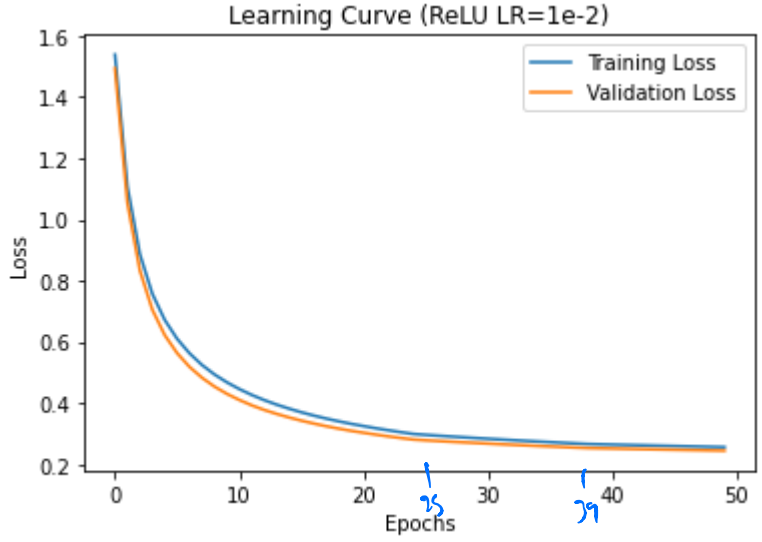
Find Test Acc: 96.73%

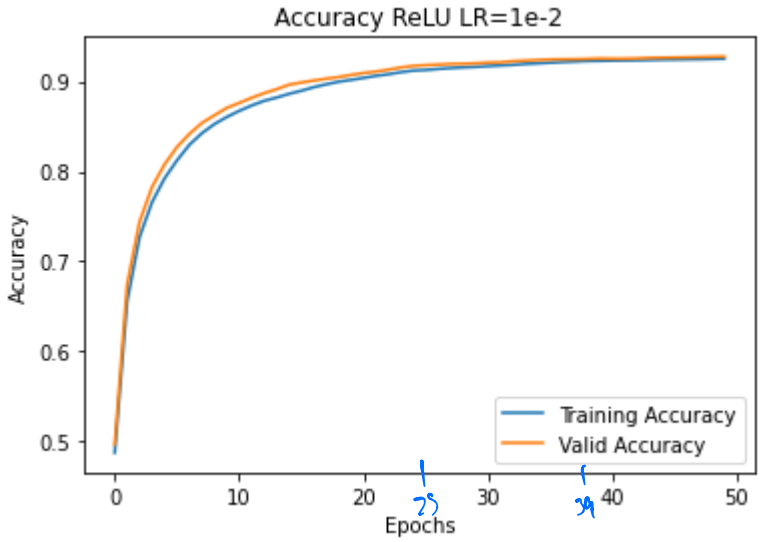
Regularization Technique from “Adversarial Noise Layer: Regularize Neural Network by Adding Noise”, Z. You, et al. CVPR 2018 [https://arxiv.org/abs/1805.08000]

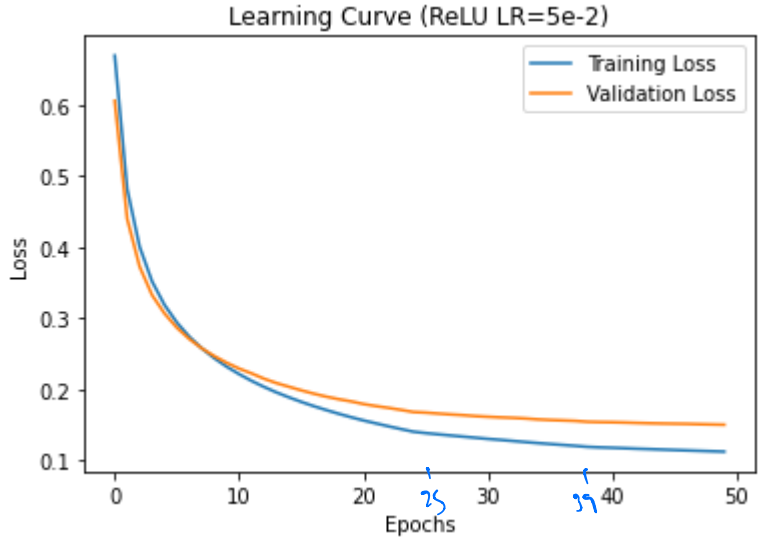
LR halved at epoch 25, 39

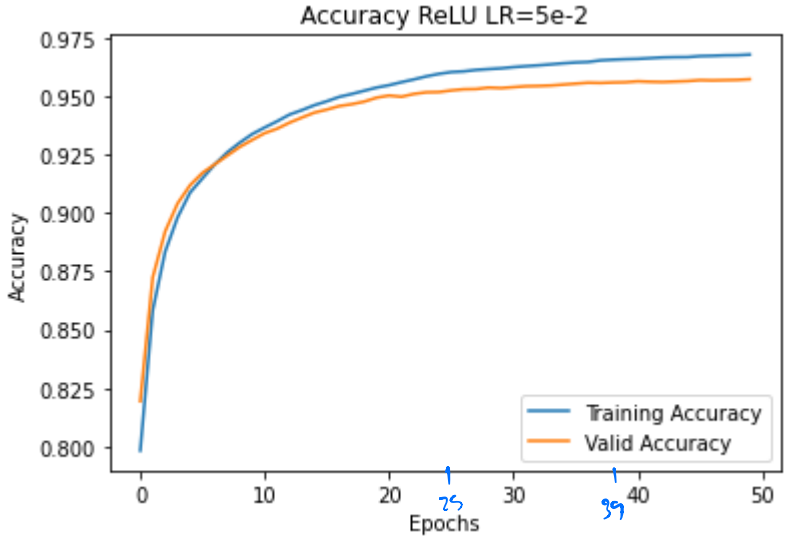


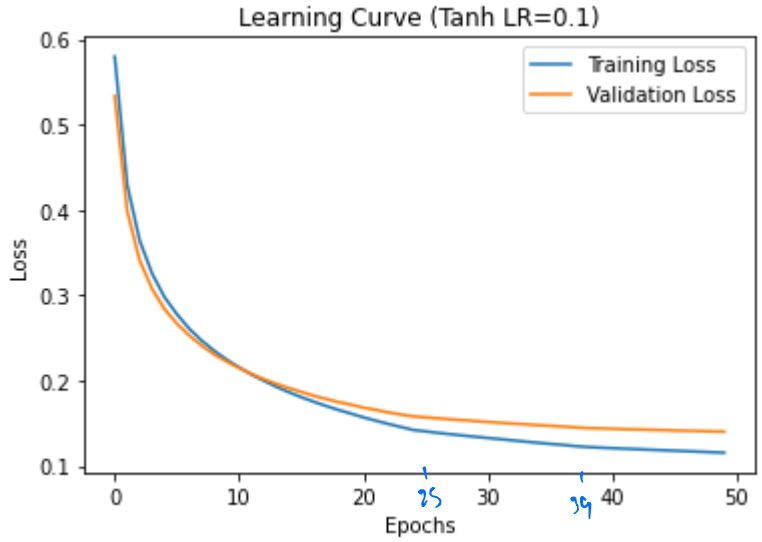


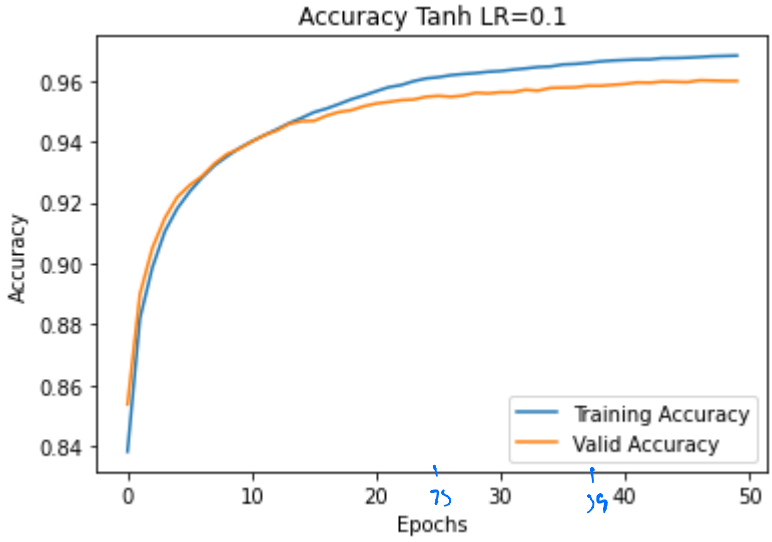


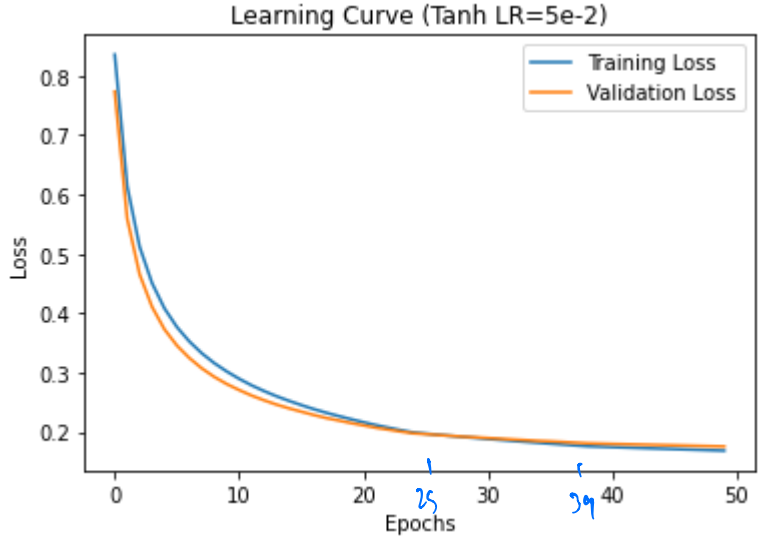

Accuracy ReLU LR=1e-2
Accuracy
Epochs
Training Accuracy
Valid Accuracy

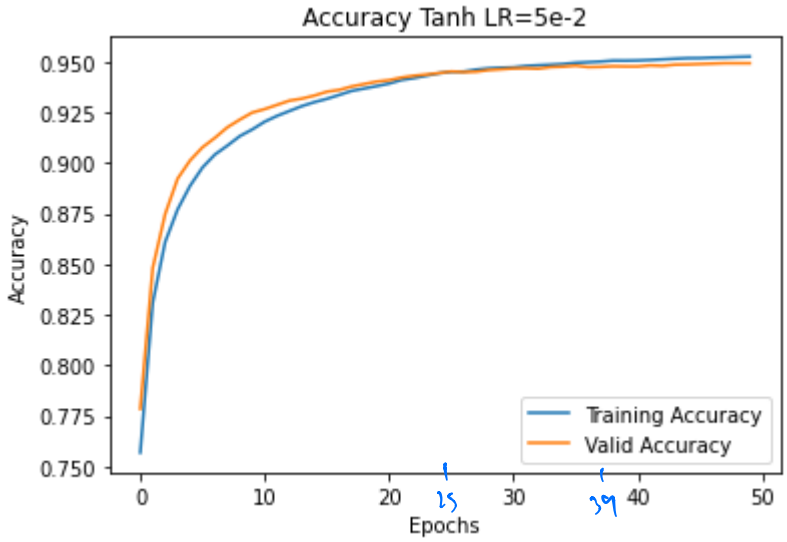

Learning Curve (ReLU LR=5e-2)
Loss
Epochs
Training Loss
Validation Loss


Accuracy ReLU LR=5e-2
Accuracy
Epochs
Training Accuracy
Valid Accuracy


Learning Curve (Tanh LR=0.1)
Training Loss
Validation Loss
Loss
Epochs


Accuracy Tanh LR=0.1
Training Accuracy
Valid Accuracy
Accuracy
Epochs


Learning Curve (Tanh LR=5e-2)
Training Loss
Validation Loss
Loss
Epochs


Accuracy Tanh LR=5e-2
Accuracy
Epochs
Training Accuracy
Valid Accuracy
0.950
0.925
0.900
0.875
0.850
0.825
0.800
0.775
0.750
0
10
20
30
40
50

In [ ]:

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import h5py
from tqdm import tqdm

class Linear():
    def __init__(self, in_features, out_features, act=None,
      self.W=np.random.randn(in_features,out_features)*np.sq
      self.b=np.zeros(out_features)
      self.activation=act
      self.activation_bkwd=act_bkwd

    def __call__(self, x, *args, **kwargs):
        return x@self.W+self.b

class NumpyMLP():
  def __init__(self, train_file, test_file, activation='relu
    #PART a
    self.data={}
    self.act=activation
    with h5py.File(train_file, 'r') as hf:
        self.data['x_train']=hf['xdata'][:]
        self.data['x_valid']=self.data['x_train'][50000:]
        self.data['x_train']=self.data['x_train'][:50000]
        self.data['y_train']=hf['ydata'][:]
        self.data['y_valid']=self.data['y_train'][50000:]
        self.data['y_train']=self.data['y_train'][:50000]
    with h5py.File(test_file,'r') as hf:
        self.data['x_test']=hf['xdata'][:]
        self.data['y_test']=hf['ydata'][:]
    print('Train set shapes:', self.data['x_train'].shape,se
    print('Valid set shapes:', self.data['x_valid'].shape, s
    print('Test set shapes:',self.data['x_test'].shape, self

    #PART b & c
    if activation=='relu':
      self.linear1=Linear(in_features=784, out_features=512,
      self.linear2=Linear(in_features=512,out_features=64, a
      self.linear3=Linear(in_features=64, out_features=32, a
      self.linear4=Linear(in_features=32, out_features=10)
    else:
      self.linear1=Linear(in_features=784, out_features=512,
      self.linear2=Linear(in_features=512,out_features=64, a
      self.linear3=Linear(in_features=64, out_features=32, a
      self.linear4=Linear(in_features=32, out_features=10)

  def relu(self, x):
    return np.clip(x,a_min=0,a_max=None)
```

```
def relu_backward(self,x):
  x[x<=0]=0
  x[x>0]=1
  return x

def tanh(self,x):
  return np.tanh(x)

def tanh_backward(self, x):
  out = [1-np.tanh(sample)**2 for sample in x]
  return np.array(out)

def softmax(self, x):
  out=[np.exp(sample)/np.sum(np.exp(sample)) for sample in
  return np.array(out)

def save_weights(self, model_path):
  with h5py.File(model_path, 'w') as hf:
    hf.create_dataset('linear_1.weight', data=self.linear1
    hf.create_dataset('linear_1.bias', data=self.linear1.b
    hf.create_dataset('linear_2.weight', data=self.linear2
    hf.create_dataset('linear_2.bias', data=self.linear2.b
    hf.create_dataset('linear_3.weight', data=self.linear3
    hf.create_dataset('linear_3.bias', data=self.linear3.b
    hf.create_dataset('linear_4.weight', data=self.linear4
    hf.create_dataset('linear_4.bias', data=self.linear4.b
    hf.create_dataset('activation',data=self.act)

def load_weights(self,model_path):
  with h5py.File(model_path,'r') as hf:
    self.linear1.W=hf['linear_1.weight'][:]
    self.linear1.b=hf['linear_1.bias'][:]
    self.linear2.W=hf['linear_2.weight'][:]
    self.linear2.b=hf['linear_2.bias'][:]
    self.linear3.W=hf['linear_3.weight'][:]
    self.linear3.b=hf['linear_3.bias'][:]
    self.linear4.W=hf['linear_4.weight'][:]
    self.linear4.b=hf['linear_4.bias'][:]
    if hf['activation']=='relu':
      self.linear1.activation, self.linear1.activation_bkw
      self.linear2.activation, self.linear2.activation_bkw
      self.linear3.activation, self.linear3.activation_bkw
      self.linear4.activation, self.linear4.activation_bkw
    else:
      self.linear1.activation, self.linear1.activation_bkw
      self.linear2.activation, self.linear2.activation_bkw
      self.linear3.activation, self.linear3.activation_bkw
      self.linear4.activation, self.linear4.activation_bkw

def train(self, epochs, batch_size=50, lr=1e-3, save_path=
  losses=[]
```

```
train_losses=[]
valid_losses=[]
train_accuracy=[]
valid_accuracy=[]
steps=[]
best_val_acc=0.0
for i in tqdm(range(epochs)):
  if i == int(epochs*0.5):
    lr=lr/2
  if i == int(epochs*0.78):
    lr=lr/2
  steps.append(lr)
  running_loss=0.0
  count=0
  for j in range(0, self.data['x_train'].shape[0], batch
    x_batch=self.data['x_train'][j:j+batch_size]
    x_batch=x_batch/255.
    noise=np.random.normal(0,0.05,x_batch.shape)
    x_batch=x_batch+noise
    y_batch=self.data['y_train'][j:j+batch_size]

    h_1=self.linear1(x_batch)
    a_1=self.linear1.activation(h_1)
    noise=np.random.normal(0,0.05,a_1.shape)
    a_1+=noise
    h_2=self.linear2(a_1)
    a_2=self.linear2.activation(h_2)
    noise=np.random.normal(0,0.05,a_2.shape)
    a_2+=noise
    h_3=self.linear3(a_2)
    a_3=self.linear3.activation(h_3)
    noise=np.random.normal(0,0.05,a_3.shape)
    # a_3+=noise
    h_4=self.linear4(a_3)
    a_4=self.softmax(h_4)

    loss=-np.sum(y_batch*np.log(a_4))/y_batch.shape[0]
    running_loss+=loss
    count+=batch_size

    delta_L = (a_4-y_batch)/x_batch.shape[0]
    da_3 = self.linear3.activation_bkwd(h_3)
    da_2 = self.linear2.activation_bkwd(h_2)
    da_1 = self.linear1.activation_bkwd(h_1)

    delta_3 = np.multiply(da_3, delta_L@self.linear4.W.T
    delta_2 = np.multiply(da_2,delta_3@self.linear3.W.T)
    delta_1 = np.multiply(da_1,delta_2@self.linear2.W.T)

    # print(np.multiply(delta_L, a_3).shape)
    # print(a_3.shape, y_batch.shape, delta_L.shape, a_3
```

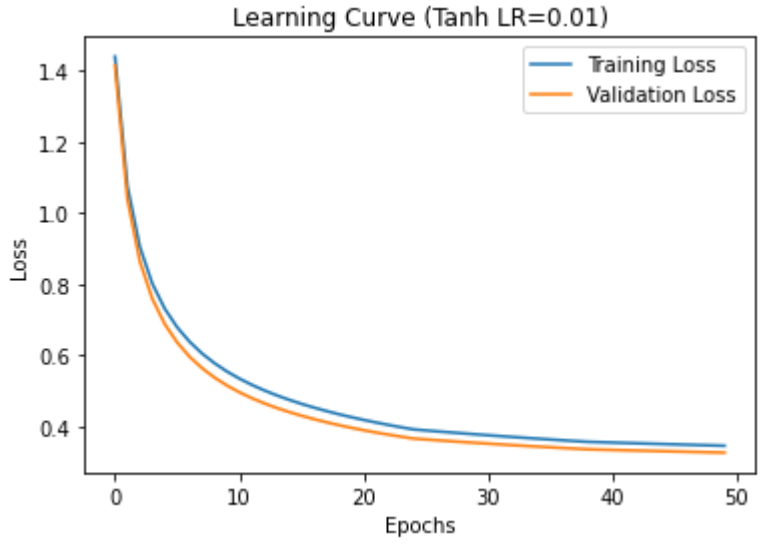
```
        self.linear4.W = self.linear4.W - lr* (a_3.T@delta_L
        self.linear4.b = self.linear4.b - lr* np.average(del

        self.linear3.W = self.linear3.W - lr * (a_2.T@delta_
        self.linear3.b = self.linear3.b - lr * np.average(de

        self.linear2.W = self.linear2.W - lr * (a_1.T@delta_
        self.linear2.b = self.linear2.b - lr* np.average(del

        self.linear1.W = self.linear1.W - lr * (x_batch.T@de
        self.linear1.b = self.linear1.b - lr * np.average(de

      losses.append(running_loss/count)
      y_pred_train=self.softmax(self.linear4(self.linear3.ac
      y_pred_logits=np.argmax(y_pred_train,axis=1)
      y_gt_logits=np.argmax(self.data['y_train'],axis=1)
      train_accuracy.append((y_pred_logits==y_gt_logits).sum
      train_losses.append(-np.sum(self.data['y_train']*np.lo
      y_pred_valid=self.softmax(self.linear4(self.linear3.ac
      y_pred_valid_logits=np.argmax(y_pred_valid,axis=1)
      y_gt_valid_logits=np.argmax(self.data['y_valid'],axis=
      valid_losses.append(-np.sum(self.data['y_valid']*np.lo
      valid_accuracy.append((y_pred_valid_logits==y_gt_valid
      if valid_accuracy[-1]>best_val_acc:
        best_val_acc=valid_accuracy[-1]
        self.save_weights(save_path)
    return losses, train_losses, valid_losses, train_accurac
```

In [2]:
```
mlp=NumpyMLP('mnist_traindata.hdf5','mnist_testdata.hdf5', a
losses,train_losses, valid_losses, train_accuracy,valid_accu
model_tanh_lr1e2=[losses,train_losses,valid_losses,train_acc
valid_accuracy[-1]
```

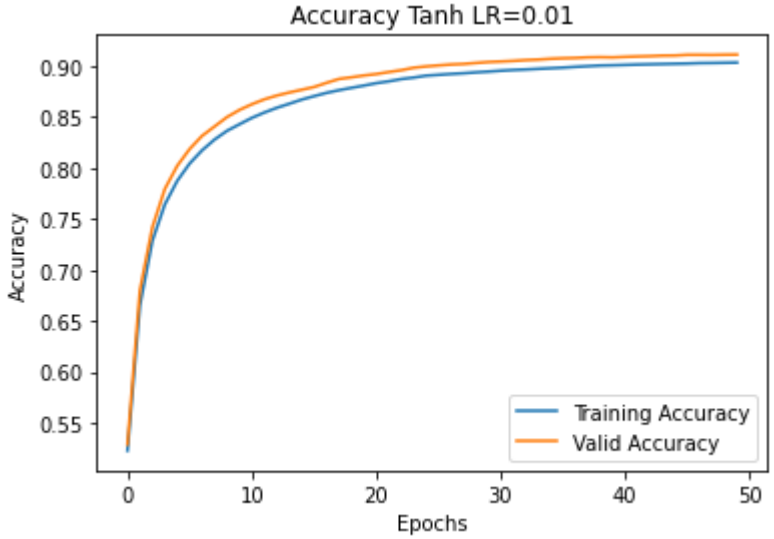
```
  0%|          | 0/50 [00:00<?, ?it/s]

Train set shapes: (50000, 784) (50000, 10)
Valid set shapes: (10000, 784) (10000, 10)
Test set shapes: (10000, 784) (10000, 10)

100%|██████████| 50/50 [15:48<00:00, 18.98s/it]
```

Out[2]:
```
0.9113
```

In [5]:
```
plt.plot(train_losses, label='Training Loss')
plt.plot(valid_losses, label='Validation Loss')
plt.title('Learning Curve (Tanh LR=0.01)')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [8]:
```
plt.plot(train_accuracy, label='Training Accuracy')
plt.plot(valid_accuracy, label='Valid Accuracy')
plt.title('Accuracy Tanh LR=0.01')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



In [11]:
```
valid_accuracy[-1]
```

Out[11]:
```
0.9113
```

In [12]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns
```

In [13]:
```
train_set=torchvision.datasets.FashionMNIST(root='./data', t
test_set=torchvision.datasets.FashionMNIST(root='./data', tr

train_loader=torch.utils.data.DataLoader(train_set, batch_si
test_loader=torch.utils.data.DataLoader(test_set, batch_size
```

```
Downloading http://fashion-mnist.s3-website.eu-central-1.am
azonaws.com/train-images-idx3-ubyte.gz (http://fashion-mnis
t.s3-website.eu-central-1.amazonaws.com/train-images-idx3-u
byte.gz) to ./data/FashionMNIST/raw/train-images-idx3-ubyt
e.gz

HBox(children=(FloatProgress(value=1.0, bar_style='info', m
ax=1.0), HTML(value='')))

Extracting ./data/FashionMNIST/raw/train-images-idx3-ubyte.
gz to ./data/FashionMNIST/raw
Downloading http://fashion-mnist.s3-website.eu-central-1.am
azonaws.com/train-labels-idx1-ubyte.gz (http://fashion-mnis
t.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-u
byte.gz) to ./data/FashionMNIST/raw/train-labels-idx1-ubyt
e.gz

HBox(children=(FloatProgress(value=1.0, bar_style='info', m
ax=1.0), HTML(value='')))

Extracting ./data/FashionMNIST/raw/train-labels-idx1-ubyte.
gz to ./data/FashionMNIST/raw
Downloading http://fashion-mnist.s3-website.eu-central-1.am
azonaws.com/t10k-images-idx3-ubyte.gz (http://fashion-mnis
t.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ub
yte.gz) to ./data/FashionMNIST/raw/t10k-images-idx3-ubyte.g
z

HBox(children=(FloatProgress(value=1.0, bar_style='info', m
ax=1.0), HTML(value='')))
```

```
Extracting ./data/FashionMNIST/raw/t10k-images-idx3-ubyte.g
z to ./data/FashionMNIST/raw
Downloading http://fashion-mnist.s3-website.eu-central-1.am
azonaws.com/t10k-labels-idx1-ubyte.gz (http://fashion-mnis
t.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ub
yte.gz) to ./data/FashionMNIST/raw/t10k-labels-idx1-ubyte.g
z
HBox(children=(FloatProgress(value=1.0, bar_style='info', m
ax=1.0), HTML(value='')))

Extracting ./data/FashionMNIST/raw/t10k-labels-idx1-ubyte.g
z to ./data/FashionMNIST/raw
Processing...
Done!

/usr/local/lib/python3.6/dist-packages/torchvision/datasets
/mnist.py:469: UserWarning: The given NumPy array is not wr
iteable, and PyTorch does not support non-writeable tensor
s. This means you can write to the underlying (supposedly n
on-writeable) NumPy array using the tensor. You may want to
copy the array to protect its data or make it writeable bef
ore converting it to a tensor. This type of warning will be
suppressed for the rest of this program. (Triggered interna
lly at  /pytorch/torch/csrc/utils/tensor_numpy.cpp:141.)
  return torch.from_numpy(parsed.astype(m[2], copy=False)).
view(*s)
```

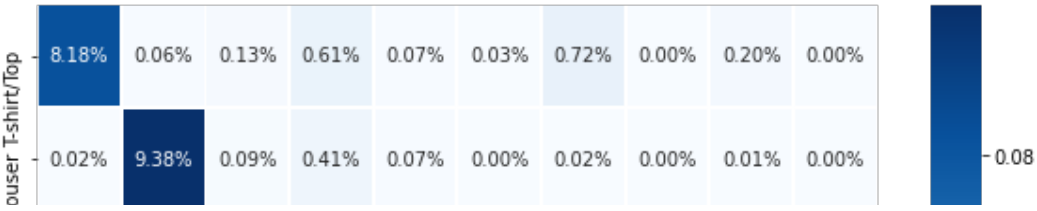
In [14]:
```
output_mapping = {
                  0: "T-shirt/Top",
                  1: "Trouser",
                  2: "Pullover",
                  3: "Dress",
                  4: "Coat",
                  5: "Sandal",
                  6: "Shirt",
                  7: "Sneaker",
                  8: "Bag",
                  9: "Ankle Boot"
                  }
```

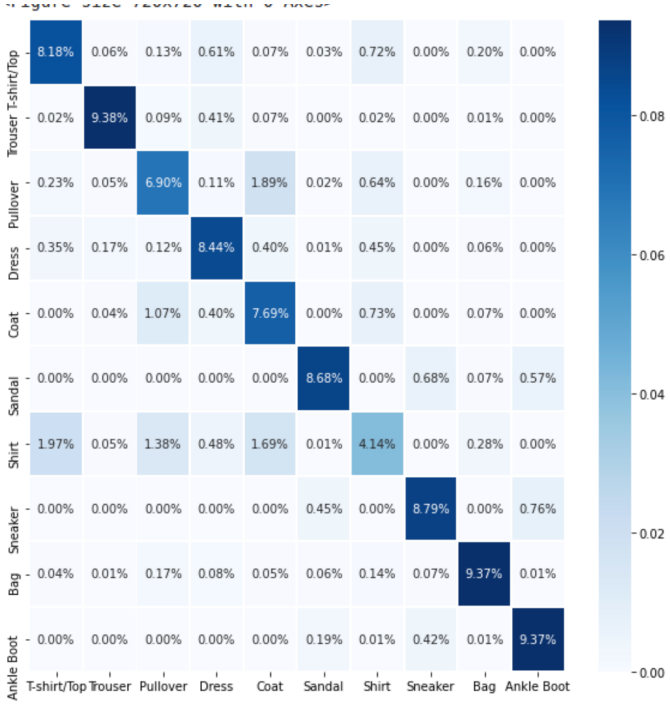
In [67]:
```
class MLP(nn.Module):
    def __init__(self, dropout_rate):
        super(MLP, self).__init__()
        self.net=nn.Sequential(
            nn.Linear(784,100),
            nn.ReLU(),
            nn.Dropout(dropout_rate),
            nn.Linear(100,10)
        )

    def forward(self, x):
```

```
        x=x.view(100,-1)
        return self.net(x)

def train():
    mlp=MLP(dropout_rate=0.3)
    losses=[]
    running_loss=0.0
    counts=0
    criterion=nn.CrossEntropyLoss()
    optimizer=torch.optim.SGD(mlp.parameters(), lr=0.01)
    for i in range(10):
        for x_train, y_train in train_loader:
            optimizer.zero_grad()
            y_pred=mlp(x_train)
            loss=criterion(y_pred,y_train)
            l2_reg=torch.tensor(0.)
            lambda_reg=torch.tensor(0.0001)
            for param in mlp.parameters():
                l2_reg+=torch.norm(param)
            loss+=lambda_reg * l2_reg
            loss.backward()
            optimizer.step()
            running_loss+=loss.item()
            counts+=1
        losses.append(running_loss/counts)
        corrects=0
        batches=0
        preds=[]
        targets=[]
        for x_test,y_test in test_loader:
            with torch.no_grad():
                y_pred=mlp(x_test)
                preds.append(F.softmax(y_pred).max(1,keepdim
                targets.append(y_test.cpu())
                corrects+=F.softmax(y_pred).max(1,keepdim=Tr
            batches+=1
        plt.clf()
        plt.figure(figsize=(10,10))
        print('Epoch %d, Loss %f'%(i, running_loss/counts))
        print('Test Accuracy:%f'%(corrects/(batches*100)))
        targets_test=torch.cat(targets).numpy()
        preds_test=(torch.cat(preds)[:,-1]).numpy()
        cf_matrix=confusion_matrix(targets_test,preds_test)

        hm=sns.heatmap(cf_matrix / np.sum(cf_matrix), annot=
                       fmt='.2%', cmap='Blues', linewidths=.9,
        hm.get_figure().savefig('epoch_%d_confusion.png'%i)
    return cf_matrix, losses

cf_matrix, losses=train()
```

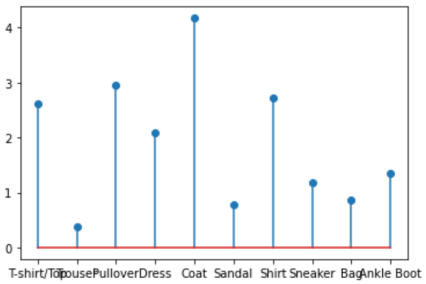
```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.p
```

```
y:44: UserWarning: Implicit dimension choice for softmax ha
s been deprecated. Change the call to include dim=X as an a
rgument.
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.p
y:46: UserWarning: Implicit dimension choice for softmax ha
s been deprecated. Change the call to include dim=X as an a
rgument.
Epoch 0, Loss 1.395633
Test Accuracy:0.654800
Epoch 1, Loss 1.127830
Test Accuracy:0.715300
Epoch 2, Loss 0.999274
Test Accuracy:0.748300
Epoch 3, Loss 0.919806
Test Accuracy:0.763400
Epoch 4, Loss 0.862425
Test Accuracy:0.781300
Epoch 5, Loss 0.818585
Test Accuracy:0.791000
Epoch 6, Loss 0.783801
Test Accuracy:0.794000
Epoch 7, Loss 0.755195
Test Accuracy:0.799700
Epoch 8, Loss 0.731126
Test Accuracy:0.814600
Epoch 9, Loss 0.710433
Test Accuracy:0.809400

<Figure size 432x288 with 0 Axes>

<Figure size 720x720 with 0 Axes>

<Figure size 720x720 with 0 Axes>

<Figure size 720x720 with 0 Axes>

<Figure size 720x720 with 0 Axes>

<Figure size 720x720 with 0 Axes>

<Figure size 720x720 with 0 Axes>

<Figure size 720x720 with 0 Axes>

<Figure size 720x720 with 0 Axes>
```





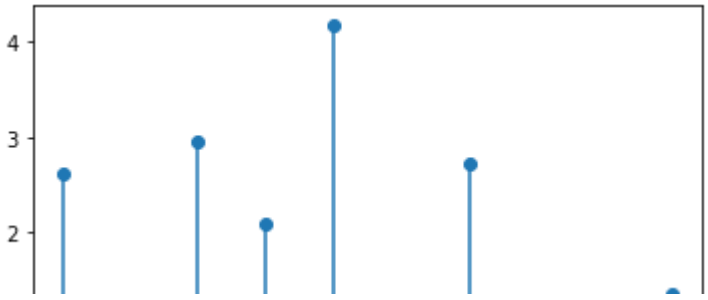

```
Class 0-T-shirt/Top most confused with: Shirt at a rate of ~1.97%
Class 1-Trouser most confused with: Dress at a rate of ~0.17%
Class 2-Pullover most confused with: Shirt at a rate of ~1.3800000000000001%
Class 3-Dress most confused with: T-shirt/Top at a rate of ~0.61%
Class 4-Coat most confused with: Pullover at a rate of ~1.8900000000000001%
Class 5-Sandal most confused with: Sneaker at a rate of ~0.45%
Class 6-Shirt most confused with: Coat at a rate of ~0.73%
Class 7-Sneaker most confused with: Sandal at a rate of ~0.68%
Class 8-Bag most confused with: Shirt at a rate of ~0.28%
Class 9-Ankle Boot most confused with: Sneaker at a rate of ~0.76%
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:10: UserWarning: In Ma
  # Remove the CWD from sys.path while we load stuff.
```




```
Most likely to be confused 2 classes:  Coat,Pullover
```

In [81]:
```
confusions_per_class=[]
for k,v in output_mapping.items():
  m=k
  col=cf_matrix[:,m]
  col=col.copy()
  col[m]=-1
  confusions_per_class.append(np.sum(col[col>0])/np.sum(cf_m
  col=col.tolist()
  print(f'Class {m}-{output_mapping[m]} most confused with:
plt.stem(confusions_per_class)
plt.xticks(range(10),output_mapping.values())
plt.show()
confusions_per_class=np.array(confusions_per_class).argsort(
print('Most likely to be confused 2 classes: ',','.join([out
```

```
Class 0-T-shirt/Top most confused with: Shirt at a rate of
~1.97%
Class 1-Trouser most confused with: Dress at a rate of ~0.1
7%
Class 2-Pullover most confused with: Shirt at a rate of ~1.
3800000000000001%
Class 3-Dress most confused with: T-shirt/Top at a rate of
~0.61%
Class 4-Coat most confused with: Pullover at a rate of ~1.8
9000000000000001%
Class 5-Sandal most confused with: Sneaker at a rate of ~0.
45%
Class 6-Shirt most confused with: Coat at a rate of ~0.73%
Class 7-Sneaker most confused with: Sandal at a rate of ~0.
68%
Class 8-Bag most confused with: Shirt at a rate of ~0.28%
Class 9-Ankle Boot most confused with: Sneaker at a rate of
~0.76%

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.p
y:10: UserWarning: In Matplotlib 3.3 individual lines on a
stem plot will be added as a LineCollection instead of indi
vidual lines. This significantly improves the performance o
f a stem plot. To remove this warning and switch to the new
behaviour, set the "use_line_collection" keyword argument t
o True.
  # Remove the CWD from sys.path while we load stuff.
```



```
Most likely to be confused 2 classes:  Coat,Pullover
```

In [82]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
```

In [83]:
```
train_set=torchvision.datasets.FashionMNIST(root='./data', t
test_set=torchvision.datasets.FashionMNIST(root='./data', tr

train_loader=torch.utils.data.DataLoader(train_set, batch_si
test_loader=torch.utils.data.DataLoader(test_set, batch_size
```

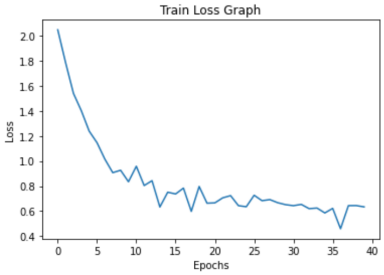
In [84]:
```
output_mapping = {
                  0: "T-shirt/Top",
                  1: "Trouser",
                  2: "Pullover",
                  3: "Dress",
                  4: "Coat",
                  5: "Sandal",
                  6: "Shirt",
                  7: "Sneaker",
                  8: "Bag",
                  9: "Ankle Boot"
                  }
```

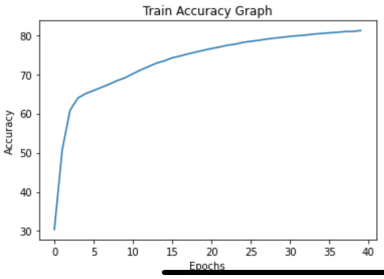
In [85]:
```
class MLP_1(nn.Module):
    def __init__(self):
        super(MLP_1, self).__init__()
        self.linear1=nn.Linear(784,128)
        self.linear2=nn.Linear(128,10)

    def forward(self,x):
        x=self.linear1(x)
        x=F.relu(x)
        x=self.linear2(x)
        return x

class MLP_2(nn.Module):
    def __init__(self):
        super(MLP_2, self).__init__()
        self.linear1=nn.Linear(784,48)
        self.linear2=nn.Linear(48,10)

    def forward(self,x):
```

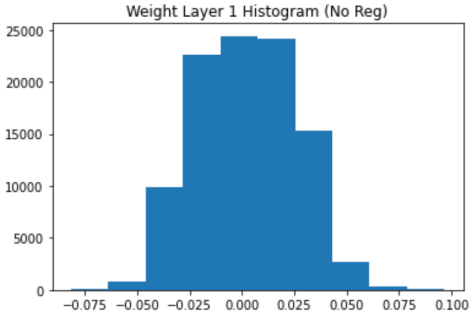
```
        x=self.linear1(x)
        x=F.relu(x)
        x=F.dropout(x,0.2)
        x=self.linear2(x)
        return x
```

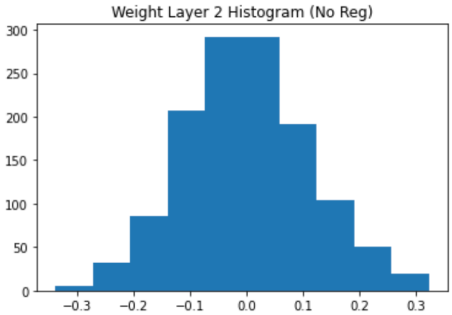
In [99]:
```
def setting_1():
    mlp_1 = MLP_1()
    mlp_1 = mlp_1.cuda()
    count = 0
    loss_func = nn.CrossEntropyLoss()
    optimizer = torch.optim.SGD(mlp_1.parameters(), lr=1e-3)
    losses = []
    accuracy = []
    for epoch in range(40):
        correct = 0
        for images, labels in train_loader:
            optimizer.zero_grad()
            count += labels.size(0)
            input = images.view(-1, 28 * 28)
            input=input.to('cuda')
            outputs = mlp_1.forward(input)
            labels=labels.to('cuda')
            # for param in mlp_1.parameters():
            #     # print(param)
            #     l2_regularization+=torch.norm(param,2)**2
            loss = loss_func(outputs, labels)
            loss.backward()
            optimizer.step()
            predictions = torch.max(outputs, 1)[1]
            correct += (predictions.cpu() == labels.cpu()).s
        losses.append(loss.data)
        accuracy.append(100 * correct / len(train_loader.dat
        print(
            f'Epoch {epoch + 1:02d}, Iteration: {count:5d},

    plt.plot(losses)
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title('Train Loss Graph')
    plt.show()
    plt.figure()
    plt.plot(accuracy)
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.title('Train Accuracy Graph')
    plt.show()
    plt.figure()

    plt.hist(mlp_1.linear1.weight.cpu().detach().numpy().T.f
    plt.title('Weight Layer 1 Histogram (No Reg)')
```

```
Epoch 33, Iteration: 1980000, Loss: 0.6198, Accuracy: 80.015%
Epoch 34, Iteration: 2040000, Loss: 0.6248, Accuracy: 80.262%
Epoch 35, Iteration: 2100000, Loss: 0.5856, Accuracy: 80.412%
Epoch 36, Iteration: 2160000, Loss: 0.6220, Accuracy: 80.580%
Epoch 37, Iteration: 2220000, Loss: 0.4597, Accuracy: 80.720%
Epoch 38, Iteration: 2280000, Loss: 0.6439, Accuracy: 80.903%
Epoch 39, Iteration: 2340000, Loss: 0.6444, Accuracy: 80.948%
Epoch 40, Iteration: 2400000, Loss: 0.6339, Accuracy: 81.160%
```









```
    plt.show()
    plt.figure()
    plt.hist(mlp_1.linear2.weight.cpu().detach().numpy().T.f
    plt.title('Weight Layer 2 Histogram (No Reg)')
    plt.show()
    return mlp_1
mlp_1=setting_1()
```

```
Epoch 01, Iteration: 60000, Loss: 2.0460, Accuracy: 30.405%
Epoch 02, Iteration: 120000, Loss: 1.7835, Accuracy: 50.74
5%
Epoch 03, Iteration: 180000, Loss: 1.5386, Accuracy: 60.83
2%
Epoch 04, Iteration: 240000, Loss: 1.4006, Accuracy: 63.95
5%
Epoch 05, Iteration: 300000, Loss: 1.2380, Accuracy: 65.09
0%
Epoch 06, Iteration: 360000, Loss: 1.1454, Accuracy: 65.86
7%
Epoch 07, Iteration: 420000, Loss: 1.0141, Accuracy: 66.66
8%
Epoch 08, Iteration: 480000, Loss: 0.9068, Accuracy: 67.48
8%
Epoch 09, Iteration: 540000, Loss: 0.9261, Accuracy: 68.39
0%
Epoch 10, Iteration: 600000, Loss: 0.8343, Accuracy: 69.12
0%
```

In [100]:

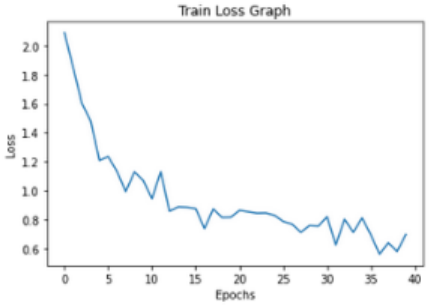
```
def setting_2():
  mlp_2 = MLP_2()
  mlp_2 = mlp_2.cuda()
  count = 0
  loss_func = nn.CrossEntropyLoss()
  optimizer = torch.optim.SGD(mlp_2.parameters(), lr=1e-3)
  losses = []
  accuracy = []
  for epoch in range(40):
      correct = 0
      for images, labels in train_loader:
          optimizer.zero_grad()
          count += 1
          input = images.view(-1, 28 * 28)
          input = input.to('cuda')
          labels = labels.to('cuda')
          l2_regularization = torch.tensor(0.)
          l2_regularization = l2_regularization.to('cuda')

          outputs = mlp_2.forward(input)

          for param in mlp_2.parameters():
              l2_regularization+=torch.norm(param)
```

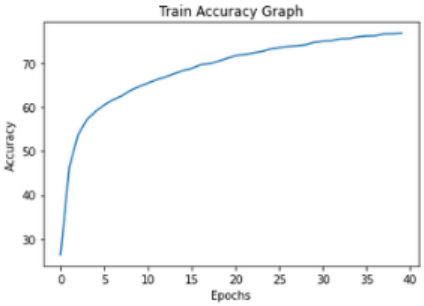
```
            loss = loss_func(outputs, labels)+0.0001*l2_regula
            loss.backward()
            optimizer.step()
            predictions = torch.max(outputs, 1)[1]
            correct += (predictions.cpu() == labels.cpu()).sum
        losses.append(loss.data)
        accuracy.append(100 * correct / len(train_loader.datas
        print(
            f'Epoch {epoch + 1:02d}, Iteration: {count:5d}, Lo

    plt.plot(losses)
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title('Train Loss Graph')
    plt.show()
    plt.figure()
    plt.plot(accuracy)
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.title('Train Accuracy Graph')
    plt.show()
    plt.figure()

    plt.hist(mlp_2.linear1.weight.cpu().detach().numpy().T.fla
    plt.title('Weight Layer 1 Histogram (L2 Reg)')
    plt.show()
    plt.figure()
    plt.hist(mlp_2.linear2.weight.cpu().detach().numpy().T.fla
    plt.title('Weight Layer 2 Histogram (L2 Reg)')
    plt.show()
    return mlp_2
mlp_2=setting_2()
```

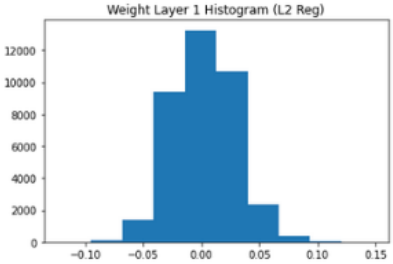
```
Epoch 01, Iteration:   600, Loss: 2.0905, Accuracy: 26.408%
Epoch 02, Iteration:  1200, Loss: 1.8492, Accuracy: 46.372%
Epoch 03, Iteration:  1800, Loss: 1.6025, Accuracy: 53.710%
Epoch 04, Iteration:  2400, Loss: 1.4805, Accuracy: 57.138%
Epoch 05, Iteration:  3000, Loss: 1.2089, Accuracy: 59.028%
Epoch 06, Iteration:  3600, Loss: 1.2353, Accuracy: 60.515%
Epoch 07, Iteration:  4200, Loss: 1.1331, Accuracy: 61.690%
Epoch 08, Iteration:  4800, Loss: 0.9938, Accuracy: 62.582%
Epoch 09, Iteration:  5400, Loss: 1.1297, Accuracy: 63.822%
Epoch 10, Iteration:  6000, Loss: 1.0708, Accuracy: 64.712%
Epoch 11, Iteration:  6600, Loss: 0.9449, Accuracy: 65.512%
Epoch 12, Iteration:  7200, Loss: 1.1306, Accuracy: 66.250%
Epoch 13, Iteration:  7800, Loss: 0.8580, Accuracy: 66.910%
Epoch 14, Iteration:  8400, Loss: 0.8883, Accuracy: 67.632%
Epoch 15, Iteration:  9000, Loss: 0.8855, Accuracy: 68.322%
Epoch 16, Iteration:  9600, Loss: 0.8757, Accuracy: 68.832%
Epoch 17, Iteration: 10200, Loss: 0.7388, Accuracy: 69.653%
Epoch 18, Iteration: 10800, Loss: 0.8739, Accuracy: 69.908%
Epoch 19, Iteration: 11400, Loss: 0.8148, Accuracy: 70.438%
```

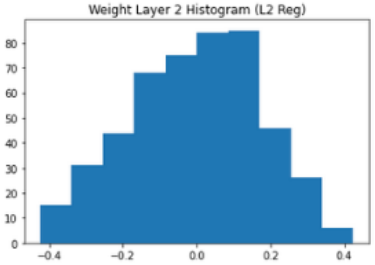
```
  return mlp_2
mlp_2=setting_2()
```

```
Epoch 30, Iteration: 18000, Loss: 0.7548, Accuracy: 74.770%
Epoch 31, Iteration: 18600, Loss: 0.8203, Accuracy: 75.013%
Epoch 32, Iteration: 19200, Loss: 0.6249, Accuracy: 75.138%
Epoch 33, Iteration: 19800, Loss: 0.8023, Accuracy: 75.563%
Epoch 34, Iteration: 20400, Loss: 0.7130, Accuracy: 75.638%
Epoch 35, Iteration: 21000, Loss: 0.8116, Accuracy: 76.013%
Epoch 36, Iteration: 21600, Loss: 0.6957, Accuracy: 76.217%
Epoch 37, Iteration: 22200, Loss: 0.5612, Accuracy: 76.273%
Epoch 38, Iteration: 22800, Loss: 0.6402, Accuracy: 76.667%
Epoch 39, Iteration: 23400, Loss: 0.5803, Accuracy: 76.695%
Epoch 40, Iteration: 24000, Loss: 0.6964, Accuracy: 76.850%
```









In [105]:
```
print(torch.max(mlp_1.linear1.weight), torch.min(mlp_1.linea
print(torch.max(mlp_1.linear2.weight), torch.min(mlp_1.linea

print(torch.max(mlp_2.linear1.weight), torch.min(mlp_2.linea
print(torch.max(mlp_2.linear2.weight), torch.min(mlp_2.linea
```

```
tensor(0.0965, device='cuda:0', grad_fn=<MaxBackward1>) ten
sor(-0.0817, device='cuda:0', grad_fn=<MinBackward1>)
tensor(0.3232, device='cuda:0', grad_fn=<MaxBackward1>) ten
sor(-0.3376, device='cuda:0', grad_fn=<MinBackward1>)
tensor(0.1474, device='cuda:0', grad_fn=<MaxBackward1>) ten
sor(-0.1221, device='cuda:0', grad_fn=<MinBackward1>)
tensor(0.4243, device='cuda:0', grad_fn=<MaxBackward1>) ten
sor(-0.4241, device='cuda:0', grad_fn=<MinBackward1>)
```

With regularization we have a more spread out distribution of weights with a slightly larger range as compared to without regularization

In [106]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import h5py
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import zero_one_loss
import matplotlib.pyplot as plt
```

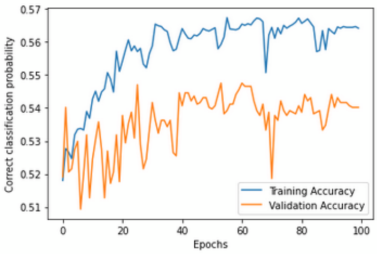
In [111]:
```
class BinaryMLP(nn.Module):
  def __init__(self):
    super(BinaryMLP, self).__init__()

    self.linear1=nn.Linear(20,20)
    self.linear2=nn.Linear(20,1)

  def forward(self, x):
    x=self.linear1(x)
    x=F.relu(x)
    x=self.linear2(x)
    return F.sigmoid(x)
```

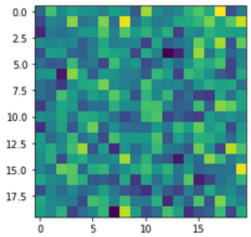
In [124]:
```
def bin_mlp_train():
  mlp=BinaryMLP()
  data_file='binary_random_20fa.hdf5'
  with h5py.File(data_file,'r') as hf:
      human = hf['human'][:]
      machine = hf['machine'][:]
  x_data = np.vstack([human, machine])
  human_y=np.ones(5100).T
  machine_y=np.zeros(5100).T
```

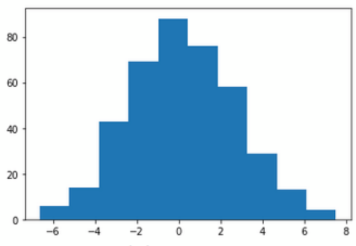
```
    y_data=np.concatenate([human_y,machine_y])
    x_train, x_valid, y_train, y_valid = train_test_split(x_da

    criterion=nn.BCELoss()
    optimizer=torch.optim.Adam(mlp.parameters(),lr=1e-2)

    running_loss=0.0
    count=0
    train_acc=[]
    valid_acc=[]
    for epoch in range(100):
        for i in range(int(x_train.shape[0]/16)):
            optimizer.zero_grad()
            x_batch=torch.Tensor(x_train[i:i+16])
            y_batch=torch.Tensor(y_train[i:i+16])

            y_pred=mlp(x_batch)

            loss=criterion(y_pred,y_batch)
            l2_regularization=torch.tensor(0.)
            for param in mlp.parameters():
                l2_regularization+=torch.norm(param)
            loss+=l2_regularization*0.000001
            loss.backward()
            optimizer.step()

            running_loss+=loss.item()
            count+=16
        with torch.no_grad():
            y_pred_train=mlp.forward(torch.Tensor(x_train))
            y_pred_valid=mlp.forward(torch.Tensor(x_valid))
            train_acc.append((1-zero_one_loss(((y_pred_train>0
            valid_acc.append((1-zero_one_loss(((y_pred_valid >
        print(f'Epoch: {epoch}, Loss:{(running_loss/count)}')
    plt.plot(train_acc, label='Training Accuracy')
    plt.plot(valid_acc, label='Validation Accuracy')
    plt.legend()
    plt.xlabel('Epochs')
    plt.ylabel('Correct classification probability')
    plt.show()
    plt.figure()
    plt.imshow(mlp.linear1.weight.detach().numpy())
    plt.show()
    plt.figure()
    plt.hist(mlp.linear1.weight.detach().numpy().flatten())
    plt.show()
    print(mlp.linear1.weight)
    print(mlp.linear1.bias)
```

In [125]:
```
bin_mlp_train()
```

```
[125] Epoch: 89, Loss:0.013836373590590863
      Epoch: 90, Loss:0.013777933530968517
      Epoch: 91, Loss:0.013715613511085318
      Epoch: 92, Loss:0.013653344861732027
      Epoch: 93, Loss:0.01359189475783268
      Epoch: 94, Loss:0.013531492832310536
      Epoch: 95, Loss:0.013472141254895428
      Epoch: 96, Loss:0.013413820809943038
      Epoch: 97, Loss:0.013356477366864482
      Epoch: 98, Loss:0.013300044079505302
      Epoch: 99, Loss:0.013244450290837092
```







```
/usr/local/lib/python3.6/dist-packages/torch/nn/functional.
py:1625: UserWarning: nn.functional.sigmoid is deprecated.
Use torch.sigmoid instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use t
orch.sigmoid instead.")
/usr/local/lib/python3.6/dist-packages/torch/nn/modules/los
s.py:529: UserWarning: Using a target size (torch.Size([1
6])) that is different to the input size (torch.Size([16,
1])) is deprecated. Please ensure they have the same size.
  return F.binary_cross_entropy(input, target, weight=self.
weight,reduction=self.reduction)
Epoch: 0, Loss:0.038579304024475795
Epoch: 1, Loss:0.03869761327080721
Epoch: 2, Loss:0.03755526756989508
Epoch: 3, Loss:0.03669944084705967
Epoch: 4, Loss:0.03563085684297131
Epoch: 5, Loss:0.03467326655198692
Epoch: 6, Loss:0.033993559366702536
Epoch: 7, Loss:0.03304366951547035
```

A majority of the weights are close to 0 which signifies sparse connections amongst features. Therefore by losing uncorrelated features we would be able to reduce dimensionality and get close to similar accuracy.

In [143]:
```
class CNN1D(nn.Module):
  def __init__(self):
    super(CNN1D,self).__init__()
    self.conv1=nn.Conv1d(20,10,3) #TODO 1: need to fix kerne
    self.conv2=nn.Conv1d(10,1,5)
  def forward(self, x):
    x=self.conv1(x)
    x=self.conv2(x)
    return F.sigmoid(x)
```

Attempts on new architecture

In [ ]:
```
class RNN(nn.Module):
    def __init__(self):
        super(RNN, self).__init__()
        self.hidden_dim = 16
        self.embedding=nn.Embedding(3,self.hidden_dim)
        self.fc = nn.Linear(self.hidden_dim, 1)
        self.fc_out = nn.Linear(20,1)

    def forward(self,x):
        embedded=self.embedding(x)
        out=self.fc(embedded)
        out=self.fc_out(out.squeeze())
        print('out',out.shape)
        return out
```

In [141]:
```
def cnn_1d_train():
  mlp=CNN1D()
```

```
data_file='binary_random_20fa.hdf5'
with h5py.File(data_file,'r') as hf:
    human = hf['human'][:]
    machine = hf['machine'][:]
x_data = np.vstack([human, machine])
human_y=np.ones(5100).T
machine_y=np.zeros(5100).T
y_data=np.concatenate([human_y,machine_y])
x_train, x_valid, y_train, y_valid = train_test_split(x_da

criterion=nn.BCELoss()
optimizer=torch.optim.Adam(mlp.parameters(),lr=1e-2)

running_loss=0.0
count=0
train_acc=[]
valid_acc=[]
for epoch in range(100):
    for i in range(int(x_train.shape[0]/16)):
        optimizer.zero_grad()
        x_batch=torch.Tensor(x_train[i:i+16])
        y_batch=torch.Tensor(y_train[i:i+16])

        y_pred=mlp(x_batch.unsqueeze(-2))

        loss=criterion(y_pred,y_batch)
        l2_regularization=torch.tensor(0.)
        for param in mlp.parameters():
            l2_regularization+=torch.norm(param)
        loss+=l2_regularization*0.000001
        loss.backward()
        optimizer.step()

        running_loss+=loss.item()
        count+=16
    with torch.no_grad():
        y_pred_train=mlp.forward(torch.Tensor(x_train))
        y_pred_valid=mlp.forward(torch.Tensor(x_valid))
        train_acc.append((1-zero_one_loss(((y_pred_train>0
        valid_acc.append((1-zero_one_loss(((y_pred_valid >
    print(f'Epoch: {epoch}, Loss:{(running_loss/count)}')
plt.plot(train_acc, label='Training Accuracy')
plt.plot(valid_acc, label='Validation Accuracy')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Correct classification probability')
plt.show()
```

In [142]:
```
cnn_1d_train()
```

```
---------------------------------------------------------------
---------------
RuntimeError                              Traceback (most r
ecent call last)
<ipython-input-142-5434ef44fbb6> in <module>()
----> 1 cnn_1d_train()

<ipython-input-141-cc214f61211b> in cnn_1d_train()
     24           y_batch=torch.Tensor(y_train[i:i+16])
     25
---> 26           y_pred=mlp(x_batch.unsqueeze(-2))
     27
     28           loss=criterion(y_pred,y_batch)

/usr/local/lib/python3.6/dist-packages/torch/nn/modules/mod
ule.py in _call_impl(self, *input, **kwargs)
    720             result = self._slow_forward(*input, **k
wargs)
    721         else:
--> 722             result = self.forward(*input, **kwargs)
    723         for hook in itertools.chain(
    724                 _global_forward_hooks.values(),

<ipython-input-136-f65d49509a1d> in forward(self, x)
      5     self.conv2=nn.Conv1d(10,1,5)
      6   def forward(self, x):
----> 7     x=self.conv1(x)
      8     x=self.conv2(x)
      9     return F.sigmoid(x)

/usr/local/lib/python3.6/dist-packages/torch/nn/modules/mod
ule.py in _call_impl(self, *input, **kwargs)
    720             result = self._slow_forward(*input, **k
wargs)
    721         else:
--> 722             result = self.forward(*input, **kwargs)
    723         for hook in itertools.chain(
    724                 _global_forward_hooks.values(),

/usr/local/lib/python3.6/dist-packages/torch/nn/modules/con
v.py in forward(self, input)
    255                             _single(0), self.dilati
on, self.groups)
    256         return F.conv1d(input, self.weight, self.bi
as, self.stride,
--> 257                         self.padding, self.dilatio
n, self.groups)
    258
    259

RuntimeError: Given groups=1, weight of size [1, 20, 3], ex
```

```
pected input[16, 1, 20] to have 20 channels, but got 1 chan
```

In [ ]:

```

```