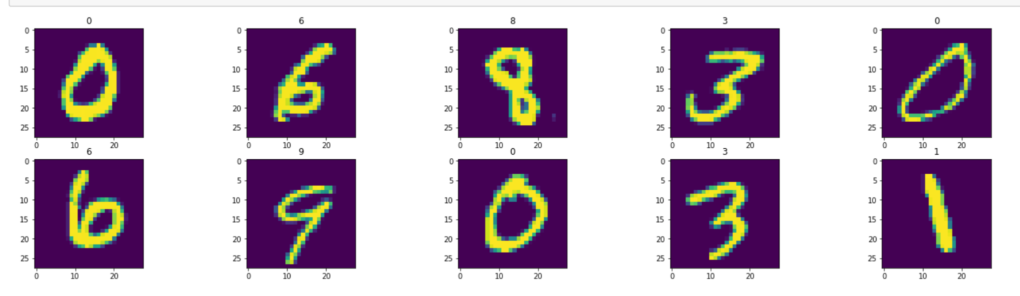
In [1]:
```python
import torch
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

## 1. Data Loading and Preprocessing

In [2]:
```python
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize((0.5,), (0.5,)),
                                ])

train_set=torchvision.datasets.MNIST(root='./mnist',train=True,download=True,transform=transform)
train_loader = torch.utils.data.DataLoader(train_set, batch_size=32, shuffle=True)
test_set = torchvision.datasets.MNIST('./mnist', download=True, train=False, transform=transform)
test_loader = torch.utils.data.DataLoader(test_set, batch_size=32,shuffle=False)
```

## 2. Sanity Check to verify whether labels are named correctly or not

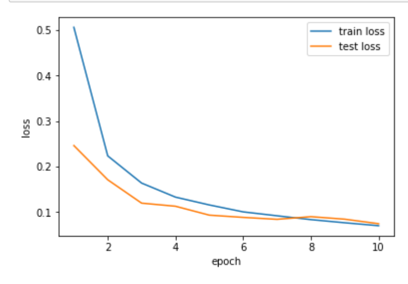
In [3]:
```python
x,y=iter(train_loader).next()
fig=plt.figure(figsize=(25,6))
for i in range(1,11):
    ax=fig.add_subplot(2,5,i)
    ax.imshow((x[i-1]/2+0.5).squeeze())
    ax.set_title(y[i-1].item())
```

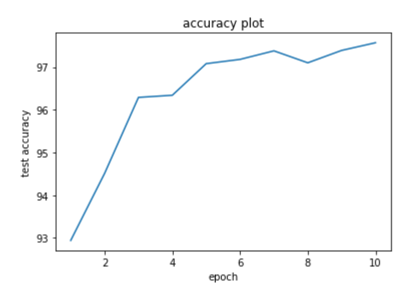



## 3.1 Multi Layer Perceptron Model- 1

In [4]:
```python
#https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html#sphx-glr-beginner-blitz-cifar10-tutorial-py

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1=nn.Linear(784,256)
        self.fc2=nn.Linear(256,128)
        self.fc3=nn.Linear(128,10)
        self.dropout=nn.Dropout(0.2)
    def forward(self,x):
        x=x.view(-1,784)
        x=F.relu(self.fc1(x))
        x=self.dropout(x)
        x=F.relu(self.fc2(x))
        x=self.fc3(x)
        return x
```

In [5]:
```python
model=Model()
criterion=nn.CrossEntropyLoss()
optimizer=optim.SGD(model.parameters(),lr=0.01,momentum=0.6)

epochs=10
train_loss_epoch=[]
test_loss_epoch=[]
test_acc_epoch=[]
best_loss=np.inf
for epoch in range(epochs):
    train_loss=0.0
    model.train()
    for i,data in enumerate(train_loader):
        inputs,labels=data
        optimizer.zero_grad()#zero graident buffers
        outputs=model(inputs)
        loss=criterion(outputs,labels)
        loss.backward() # backpropagate
        optimizer.step()# update weights
        train_loss+=loss.item()
    train_loss_epoch.append(train_loss/len(train_loader))

    model.eval()
    test_loss=0.0
    correct_count=0
    for i,data in enumerate(test_loader):
        inputs,labels=data
        outputs=model(inputs)
        _,predicted = torch.max(outputs, 1)
        correct_count+=(predicted==labels).sum().item()
        loss=criterion(outputs,labels)
        test_loss+=loss.item()
    if best_loss>test_loss:
        torch.save(model,'best_model.pth')
        best_loss=test_loss
    test_loss_epoch.append(test_loss/len(test_loader))
    test_acc_epoch.append(correct_count*100/len(test_loader.dataset))


plt.plot(range(1,epochs+1),train_loss_epoch,label='train loss')
plt.plot(range(1,epochs+1),test_loss_epoch,label='test loss')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend()
plt.show()
plt.plot(range(1,epochs+1),test_acc_epoch)
plt.xlabel('epoch')
plt.ylabel('test accuracy')
plt.title('accuracy plot')
plt.show()
```




## 3.2 checking how model is performing on different classes

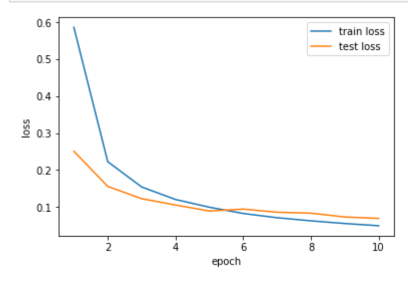
In [6]:
```python
model=torch.load('best_model.pth')
model.eval()
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
for data in test_loader:
    inputs, labels = data
    outputs = model(inputs)
    _, predicted = torch.max(outputs, 1)
    c = (predicted == labels)
    for i in range(len(labels)):
        label = labels[i]
        class_correct[label] += c[i].item()
        class_total[label] += 1


for i in range(10):
    print('Accuracy of %5s : %2d %%' % (i, 100 * class_correct[i] / class_total[i]))
```

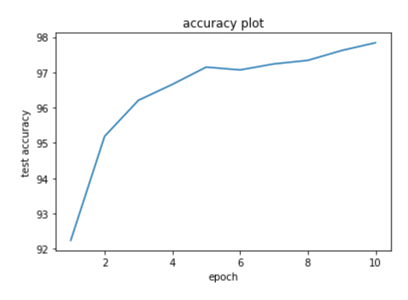
```
Accuracy of     0 : 98 %
Accuracy of     1 : 99 %
Accuracy of     2 : 97 %
Accuracy of     3 : 98 %
Accuracy of     4 : 98 %
Accuracy of     5 : 95 %
Accuracy of     6 : 97 %
Accuracy of     7 : 95 %
Accuracy of     8 : 97 %
Accuracy of     9 : 96 %
```

## 4. Multi Layer Perceptron Model- 2

In [7]:
```python
class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1=nn.Linear(784,512)
        self.fc2=nn.Linear(512,256)
        self.fc3=nn.Linear(256,128)
        self.fc4=nn.Linear(128,10)
        self.dropout=nn.Dropout(0.2)
    def forward(self,x):
        x=x.view(-1,784)
        x=F.relu(self.fc1(x))
        x=F.relu(self.fc2(x))
        x=self.dropout(x)
        x=F.relu(self.fc3(x))
        x=self.fc4(x)

        return x
```

In [8]:
```python
model=Model()
criterion=nn.CrossEntropyLoss()
optimizer=optim.SGD(model.parameters(),lr=0.01,momentum=0.6)

epochs=10
train_loss_epoch=[]
test_loss_epoch=[]
test_acc_epoch=[]
best_loss=np.inf
for epoch in range(epochs):
    train_loss=0.0
    model.train()
    for i,data in enumerate(train_loader):
        inputs,labels=data
        optimizer.zero_grad()#zero graident buffers
        outputs=model(inputs)
        loss=criterion(outputs,labels)
        loss.backward() # backpropagate
        optimizer.step()# update weights
        train_loss+=loss.item()
    train_loss_epoch.append(train_loss/len(train_loader))

    model.eval()
    test_loss=0.0
    correct_count=0
    for i,data in enumerate(test_loader):
        inputs,labels=data
        outputs=model(inputs)
        _,predicted = torch.max(outputs, 1)
        correct_count+=(predicted==labels).sum().item()
        loss=criterion(outputs,labels)
        test_loss+=loss.item()
    if best_loss>test_loss:
        torch.save(model,'best_model.pth')
        best_loss=test_loss
    test_loss_epoch.append(test_loss/len(test_loader))
    test_acc_epoch.append(correct_count*100/len(test_loader.dataset))


plt.plot(range(1,epochs+1),train_loss_epoch,label='train loss')
plt.plot(range(1,epochs+1),test_loss_epoch,label='test loss')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend()
plt.show()
plt.plot(range(1,epochs+1),test_acc_epoch)
plt.xlabel('epoch')
plt.ylabel('test accuracy')
plt.title('accuracy plot')
plt.show()
```




## 4.2 checking how model is performing on different classes

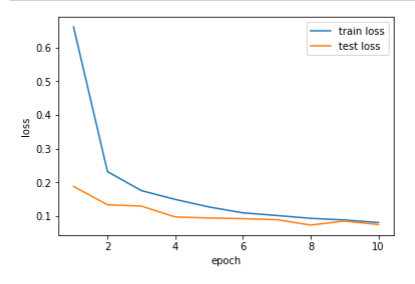
In [9]:
```python
model=torch.load('best_model.pth')
model.eval()
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
for data in test_loader:
    inputs, labels = data
    outputs = model(inputs)
    _, predicted = torch.max(outputs, 1)
    c = (predicted == labels)
    for i in range(len(labels)):
        label = labels[i]
        class_correct[label] += c[i].item()
        class_total[label] += 1


for i in range(10):
    print('Accuracy of %5s : %2d %%' % (i, 100 * class_correct[i] / class_total[i]))
```

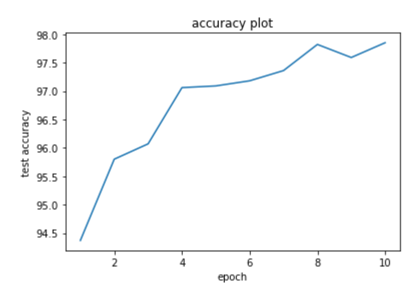
```
Accuracy of     0 : 99 %
Accuracy of     1 : 98 %
Accuracy of     2 : 98 %
Accuracy of     3 : 98 %
Accuracy of     4 : 97 %
Accuracy of     5 : 97 %
Accuracy of     6 : 97 %
Accuracy of     7 : 95 %
Accuracy of     8 : 98 %
Accuracy of     9 : 96 %
```

## 5. Multi Layer Perceptron Model- 3

In [10]:
```python
class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1=nn.Linear(784,512)
        self.fc2=nn.Linear(512,256)
        self.fc3=nn.Linear(256,128)
        self.fc4=nn.Linear(128,56)
        self.fc5=nn.Linear(56,28)
        self.fc6=nn.Linear(28,10)
        self.dropout=nn.Dropout(0.2)
        self.batchnorm=nn.BatchNorm1d(128)
    def forward(self,x):
        x=x.view(-1,784)
        x=F.relu(self.fc1(x))
        x=self.dropout(x)
        x=F.relu(self.fc2(x))
        x=F.relu(self.fc3(x))
        x=self.batchnorm(x)
        x=F.relu(self.fc4(x))
        x=self.dropout(x)
        x=F.relu(self.fc5(x))
        x=self.fc6(x)

        return x
```

In [11]:
```python
model=Model()
criterion=nn.CrossEntropyLoss()
optimizer=optim.SGD(model.parameters(),lr=0.01,momentum=0.6)

epochs=10
train_loss_epoch=[]
test_loss_epoch=[]
test_acc_epoch=[]
best_loss=np.inf
for epoch in range(epochs):
    train_loss=0.0
    model.train()
    for i,data in enumerate(train_loader):
        inputs,labels=data
        optimizer.zero_grad()#zero graident buffers
        outputs=model(inputs)
        loss=criterion(outputs,labels)
        loss.backward() # backpropagate
        optimizer.step()# update weights
        train_loss+=loss.item()
    train_loss_epoch.append(train_loss/len(train_loader))

    model.eval()
    test_loss=0.0
    correct_count=0
    for i,data in enumerate(test_loader):
        inputs,labels=data
        outputs=model(inputs)
        _,predicted = torch.max(outputs, 1)
        correct_count+=(predicted==labels).sum().item()
        loss=criterion(outputs,labels)
        test_loss+=loss.item()
    if best_loss>test_loss:
        torch.save(model,'best_model.pth')
        best_loss=test_loss
    test_loss_epoch.append(test_loss/len(test_loader))
    test_acc_epoch.append(correct_count*100/len(test_loader.dataset))


plt.plot(range(1,epochs+1),train_loss_epoch,label='train loss')
plt.plot(range(1,epochs+1),test_loss_epoch,label='test loss')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend()
plt.show()
plt.plot(range(1,epochs+1),test_acc_epoch)
plt.xlabel('epoch')
plt.ylabel('test accuracy')
plt.title('accuracy plot')
plt.show()
```




## 5.2 checking how model is performing on different classes

In [12]:
```python
model=torch.load('best_model.pth')
model.eval()
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
for data in test_loader:
    inputs, labels = data
    outputs = model(inputs)
    _, predicted = torch.max(outputs, 1)
    c = (predicted == labels)
    for i in range(len(labels)):
        label = labels[i]
        class_correct[label] += c[i].item()
        class_total[label] += 1


for i in range(10):
    print('Accuracy of %5s : %2d %%' % (i, 100 * class_correct[i] / class_total[i]))
```

```
Accuracy of     0 : 99 %
Accuracy of     1 : 99 %
Accuracy of     2 : 97 %
Accuracy of     3 : 98 %
Accuracy of     4 : 96 %
Accuracy of     5 : 97 %
Accuracy of     6 : 98 %
Accuracy of     7 : 97 %
Accuracy of     8 : 96 %
Accuracy of     9 : 97 %
```