Exercise 4: Custom Losses in Convolutional Neural Networks on the MNIST dataset

In [1]:
```
import torch
import torchvision
import torchvision.datasets as datasets
import torchvision.transforms as transforms
transform = transforms.Compose([transforms.ToTensor(),
  transforms.Normalize((0.5,), (0.5,))
])
#loading the MNIST dataset
trainset = datasets.MNIST(root='./data', train=True, download=True, transform=tra
trainloader = torch.utils.data.DataLoader(trainset, batch_size=1,
                                          shuffle=True, num_workers=2)
testset = datasets.MNIST(root='./data', train=False,
                                         download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=1,
                                         shuffle=False, num_workers=2)
```

executed in 1.40s, finished 09:42:56 2020-07-16

In [29]:
```
#defining classes for regression
#classes for regression is same as the classes for classification except that the
classes = ('0','1','2','3','4','5','6','7','8','9')
```

executed in 4ms, finished 11:04:19 2020-07-16

In [30]:
```
print(len(trainset),len(testset))
```

executed in 6ms, finished 11:15:13 2020-07-16

```
60000 10000
```

Train test split

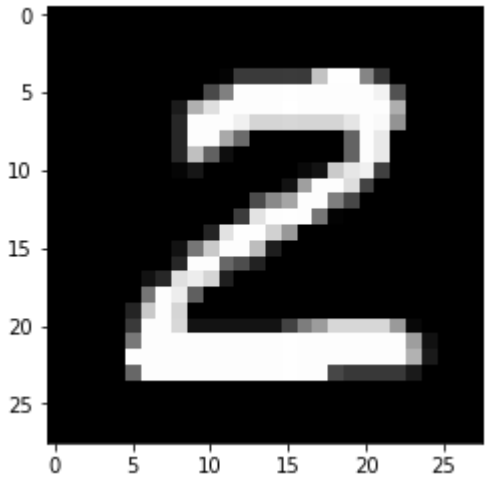
The total size of dataset is 70000

60000 records are used for training

10000 records are used for testing

In [3]:
```
import matplotlib.pyplot as plt
import numpy as np



def imshow(img):
    img = img / 2 + 0.5
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()
#printing an image
dataiter = iter(trainloader)
images, labels = dataiter.next()
imshow(torchvision.utils.make_grid(images))
# print labels
print(' '.join('%5s' % classes[labels[j]] for j in range(1)))
```

executed in 7.80s, finished 09:43:07 2020-07-16



2

Structure

First layer: Convolutional layer, taken in one channel, gives out 6 channels

Second layer: MaxPool layer, dimension does not change

Third layer: Convolutional layer, taken in 6 channels, gives out 16 channels

Fourth layer: MaxPool layer, dimension does not change

Fifth layer: Fully connected layer that takes flattened output of fourth layer

Classification head

Takes output from the fifth layer

First layer in classification head: Fullyconnected layer than takes output from fifth layer and gives out output of dimension 1X84

Second layer in classification head: Fullyconnected layer than takes output from first layer in classif head and gives out output of dimension 40

Regression head

First layer: Fully connected layer than takes output from fifth layer and gives output of size 50

Second layer: Fully connected layer than takes output from first layer in reg head and gives output of size 25

third layer: Fully connected layer than takes output from fifth layer and gives output of size 1

In [7]:
```
import torch.nn as nn
import torch.nn.functional as F

#multi task setting

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(256, 120)
        self.classif1 = nn.Linear(120, 84)
        self.classif2 = nn.Linear(84, 10)

        self.fcreg1 = nn.Linear(120, 50)
        self.fcreg2 = nn.Linear(50,25)
        self.fcreg3 = nn.Linear(25,1)


    def forward(self, x):

        x = self.pool(F.relu(self.conv1(x)))

        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1,256)
        x = F.relu(self.fc1(x))
        classif = F.relu(self.classif1(x))
        classif = self.classif2(classif)

        reg = F.relu(self.fcreg1(x))
        reg = F.relu(self.fcreg2(reg))
        reg = self.fcreg3(reg)

        return [classif,reg]


net = Net()
```

executed in 80ms, finished 09:47:03 2020-07-16

In [8]:
```
import torch.optim as optim
#Stochastic gradient descent
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

executed in 6ms, finished 09:47:05 2020-07-16

In [28]:
```
#
```

executed in 5ms, finished 11:02:01 2020-07-16

In [11]:
```
def custom_loss_crossentropy(x,y):
    log_prob = -1.0 * F.log_softmax(x, 1)
    loss = log_prob.gather(1, y.unsqueeze(1))
    loss = loss.mean()
    return loss
def custom_loss_mse(output, target):
    loss = torch.mean((output - target)**2)
    return loss
class_loss = []
reg_loss = []
loss_list = []
#Running for three epochs
for epoch in range(3):
    print('epoch',epoch)
    running_loss = 0.0
    #Iterating over training dataset
    for i, data in enumerate(trainloader, 0):

        inputs, labels = data

        #zeroing the gradients
        optimizer.zero_grad()
        #Feeding inputs to the model
        outputs = net(inputs)
        #Classification loss is a crossentropy loss
        class_l = custom_loss_crossentropy(outputs[0], labels)
        #Regression loss is a Mean Squared Error loss
        reg_l = custom_loss_mse(outputs[1], int(labels))
        #Loss is the combination of classification loss and regression loss
        loss = class_l+reg_l
        loss_list.append(loss)
        class_loss.append(class_l)
        reg_loss.append(reg_l)
        #Backpropagation of errors
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        if i % 2000 == 1999:
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

print('Finished Training')
```

executed in 32m 12s, finished 10:19:37 2020-07-16

```
epoch 0
[1,  2000] loss: 10.866
[1,  4000] loss: 10.836
[1,  6000] loss: 10.640
[1,  8000] loss: 9.254
[1, 10000] loss: 10.517
[1, 12000] loss: 10.982
[1, 14000] loss: 10.716
[1, 16000] loss: 10.664
[1, 18000] loss: 10.453
[1, 20000] loss: 10.925
[1, 22000] loss: 10.778
```

```
[1, 24000] loss: 10.786
[1, 26000] loss: 10.879
[1, 28000] loss: 11.030
[1, 30000] loss: 10.566
[1, 32000] loss: 10.778
[1, 34000] loss: 10.512
[1, 36000] loss: 10.592
[1, 38000] loss: 10.623
[1, 40000] loss: 10.746
[1, 42000] loss: 10.738
[1, 44000] loss: 10.391
[1, 46000] loss: 10.415
[1, 48000] loss: 10.894
[1, 50000] loss: 10.915
[1, 52000] loss: 10.827
[1, 54000] loss: 10.954
[1, 56000] loss: 10.690
[1, 58000] loss: 10.956
[1, 60000] loss: 10.650
epoch 1
[2,  2000] loss: 10.990
[2,  4000] loss: 10.501
[2,  6000] loss: 10.805
[2,  8000] loss: 10.960
[2, 10000] loss: 10.990
[2, 12000] loss: 9.947
[2, 14000] loss: 9.873
[2, 16000] loss: 10.203
[2, 18000] loss: 10.796
[2, 20000] loss: 10.982
[2, 22000] loss: 10.415
[2, 24000] loss: 10.908
[2, 26000] loss: 10.560
[2, 28000] loss: 10.634
[2, 30000] loss: 10.817
[2, 32000] loss: 10.750
[2, 34000] loss: 10.754
[2, 36000] loss: 10.791
[2, 38000] loss: 10.476
[2, 40000] loss: 10.513
[2, 42000] loss: 10.950
[2, 44000] loss: 10.707
[2, 46000] loss: 10.876
[2, 48000] loss: 10.793
[2, 50000] loss: 10.856
[2, 52000] loss: 10.545
[2, 54000] loss: 10.674
[2, 56000] loss: 10.635
[2, 58000] loss: 10.611
[2, 60000] loss: 10.875
epoch 2
[3,  2000] loss: 10.821
[3,  4000] loss: 10.677
[3,  6000] loss: 10.634
[3,  8000] loss: 10.846
[3, 10000] loss: 10.650
[3, 12000] loss: 10.485
```

```
[3, 14000] loss: 10.935
[3, 16000] loss: 10.893
[3, 18000] loss: 10.448
[3, 20000] loss: 10.446
[3, 22000] loss: 10.812
[3, 24000] loss: 10.820
[3, 26000] loss: 10.694
[3, 28000] loss: 11.010
[3, 30000] loss: 10.817
[3, 32000] loss: 10.645
[3, 34000] loss: 10.541
[3, 36000] loss: 10.905
[3, 38000] loss: 10.672
[3, 40000] loss: 10.804
[3, 42000] loss: 10.882
[3, 44000] loss: 10.925
[3, 46000] loss: 10.688
[3, 48000] loss: 10.814
[3, 50000] loss: 10.745
[3, 52000] loss: 10.841
[3, 54000] loss: 10.655
[3, 56000] loss: 10.704
[3, 58000] loss: 10.718
[3, 60000] loss: 10.588
Finished Training
```

Classification loss

Loss for the final minibatch is 2.3065

Regression loss

Loss for the final minibatch is 8.2815

Total loss for final minibatch is 10.5880

In [27]:

```
print(sum(class_loss[-2000:])/2000)
print(sum(reg_loss[-2000:])/2000)
print(sum(loss_list[-2000:])/2000)
```

executed in 58ms, finished 10:58:32 2020-07-16

```
tensor(2.3065, grad_fn=<DivBackward0>)
tensor(8.2815, grad_fn=<DivBackward0>)
tensor(10.5880, grad_fn=<DivBackward0>)
```

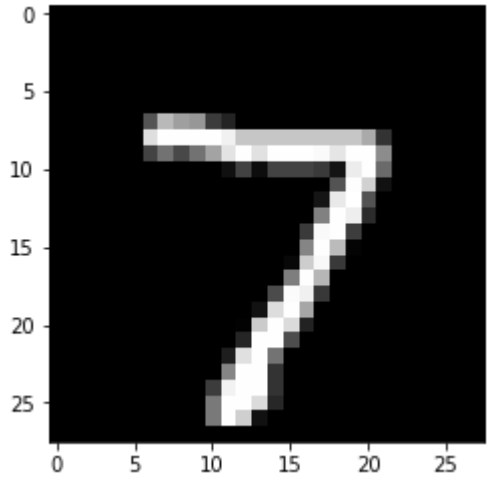
In [ ]:

```

```

executed in 19ms, finished 11:22:55 2020-07-16

In [19]:
```
dataiter = iter(testloader)
images, labels = dataiter.next()

# print images
imshow(torchvision.utils.make_grid(images))
print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in range(1)))
```

executed in 33.2s, finished 10:43:49 2020-07-16




```
GroundTruth:      7
```

In [81]:
```
outputs = net(images)
```

executed in 8ms, finished 22:06:20 2020-07-15

In [82]:
```
_, predicted = torch.max(outputs[0], 1)
print(outputs[1])
print('Predicted: ', ' '.join('%5s' % classes[predicted[j]]
                              for j in range(1)))
```

executed in 9ms, finished 22:06:22 2020-07-15

```
tensor([[-2.9946,  0.3137,  2.1022, -1.1559, -1.6439, -4.4379, -8.3407, 13.123
9,
         -3.4624,  3.2797]], grad_fn=<AddmmBackward>)
Predicted:      7
```

In [23]:
```
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs[0].data, 1)
        predicted2 = outputs[1]
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %d %%' % (
    100 * correct / total))
```

executed in 13.9s, finished 10:55:35 2020-07-16

```
Accuracy of the network on the 10000 test images: 11 %
```

Accuracy for classification

11%

It does not perform good after introducing regression head

Regression loss does not converge much

In [ ]:

In [ ]: