CS498 Applied Machine Learning Assignment #8

Students:

nidiaib2, Nidia Bucarelli

sunnyk2, Sunny Katiyar

wangx2, Wang Xiang

May 14, 2020

Spring 2020

EXERCISE 17.1.

Source: https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

Architecture:

```
class Net(nn.Module):
 def __init__(self):
  super(Net, self).__init__()
  self.conv1 = nn.Conv2d(1, 10, 5)
  self.pool = nn.MaxPool2d(2, 2)
  self.conv2 = nn.Conv2d(10, 20, 5)
  self.fc1 = nn.Linear(320, 120)
  self.fc2 = nn.Linear(120, 84)
  self.fc3 = nn.Linear(84, 10)
 def forward(self, x):
  x = self.pool(F.relu(self.conv1(x)))
  x = self.pool(F.relu(self.conv2(x)))
  x = x.view(-1, 320)
  x = F.relu(self.fc1(x))
  x = F.relu(self.fc2(x))
  x = self.fc3(x)
  return x
```

- Optimizer: SDG
- Loss: Cross Entropy
- Batch_size: 32
- Accuracy in Test Set: 98%

EXERCISE 17.2.

Part a.

- Epochs= 30
- Batch_size= 124
- No data augmentation
- learning_rate= 0.001
- Loss= Cross Entropy
- Optimizer= SGD

RESULTS:

- THE TRAINING ACCURACY IS: 93.19%;
- THE TESTING ACCURACY IS: 93.71%;

When adding momentum using ADAGRAD (with lr=0.01), the accuracy improved. Momentum helps us overcome local minimum, and thus by finding the global minimum, it can lead to a higher accuracy. Higher accuracy was achieved in earlier epochs.

Epoch 0 (training):

- THE TRAINING ACCURACY IS: 95.38%;
- THE TESTING ACCURACY IS: 98.17%;

Part b.

- Epochs= 30
- Batch_size= 124
- No data augmentation
- learning_rate= 0.001
- Loss = Cross Entropy
- Optimizer= SGD

Architecture Modification:

- Dropout after Conv1 = 0.30
- Dropout after Conv2= 0.30

RESULTS:

- THE TRAINING ACCURACY IS: 94.14%;
- THE TESTING ACCURACY IS: 95.17%;

When adding dropouts, it's a regularization method, and allows us to prevent overfitting when the neural network structure is relatively complicated compared to data size.

Part c.

- Epochs= 13
- Batch_size= 124
- No data augmentation
- learning_rate= 0.001
- Loss= Cross Entropy
- Optimizer= Adam

Architecture Modifications:

- Dropout after Conv1 = 0.30
- Dropout after Con2= 0.30
- Batch Normalization after Conv3
- Batch Normalization after Conv4

RESULTS:

- THE TRAINING ACCURACY IS: 99.95%;
- THE TESTING ACCURACY IS: 99.07%;

Best accuracy is achieved when batch normalization is added to control for input distribution and Adam is used as optimizer.

HW8

May 15, 2020

```
In [0]: import h5py
        import numpy as np
        from random import randint
        import time
        import requests
        import matplotlib.pyplot as plt
        import cv2
        # from sklearn.ensemble.forest import _generate_unsampled_indices
        # from sklearn.ensemble import BaggingClassifier
        from sklearn.ensemble import RandomForestClassifier
        import h5py
        import numpy as np
        from random import randint
        import torch
        import torch.nn as nn
        import torchvision
        import torchvision.transforms as transforms
        import torch.nn.functional as F
        import torch.optim as optim
        from torch.autograd import Variable
        import time
```

0.0.1 1 Problem 17.1

References https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html https://nextjournal.com/gkoehler/pytorch-mnist

1.1 Loading the Dataset and Normalizing MNIST

```
In [0]: transform = transforms.Compose(
        [transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))])
        trainset = torchvision.datasets.MNIST(root='./data', train=True,
        download=True, transform=transform)
        trainloader = torch.utils.data.DataLoader(trainset, batch_size=32,
        shuffle=True, num_workers=2)
        testset = torchvision.datasets.MNIST(root='./data', train=False,
        download=True, transform=transform)
```

```
        testloader = torch.utils.data.DataLoader(testset, batch_size=32,
        shuffle=False, num_workers=2)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./data/MNIST/raw/tra


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


Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./data/MNIST/raw/tra


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


Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw/t10k


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


Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw/t10k




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


Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw
Processing...
Done!


/pytorch/torch/csrc/utils/tensor_numpy.cpp:141: UserWarning: The given NumPy array is not write
```

1.2 Define a Convolutional Neural Network

```
In [0]: class Net(nn.Module):
          def __init__(self):
            super(Net, self).__init__()
            self.conv1 = nn.Conv2d(1, 10, 5)
            self.pool = nn.MaxPool2d(2, 2)
            self.conv2 = nn.Conv2d(10, 20, 5)
            self.fc1 = nn.Linear(320, 120)
            self.fc2 = nn.Linear(120, 84)
```

```
            self.fc3 = nn.Linear(84, 10)
          def forward(self, x):
            x = self.pool(F.relu(self.conv1(x)))
            x = self.pool(F.relu(self.conv2(x)))
            x = x.view(-1, 320)
            x = F.relu(self.fc1(x))
            x = F.relu(self.fc2(x))
            x = self.fc3(x)
            return x
        net = Net()
```

1.3 Define a loss function

```
In [0]: criterion = nn.CrossEntropyLoss()
        optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

1.4 Train the network on the training data

```
In [0]: for epoch in range(5): # loop over the dataset multiple times
          running_loss = 0.0
          for i, data in enumerate(trainloader, 0):
            # get the inputs; data is a list of [inputs, labels]
            inputs, labels = data
            # zero the parameter gradients
            optimizer.zero_grad()
            # forward + backward + optimize
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            # print statistics
            running_loss += loss.item()
            if i % 512 == 511: # print every 512 mini-batches
              print('[%d, %5d] loss: %.3f' %
                (epoch + 1, i + 1, running_loss / 2000))
              running_loss = 0.0
        print('Finished Training')
```

```
[1,   512] loss: 0.111
[1,  1024] loss: 0.066
[1,  1536] loss: 0.046
[2,   512] loss: 0.033
[2,  1024] loss: 0.030
[2,  1536] loss: 0.026
[3,   512] loss: 0.021
[3,  1024] loss: 0.022
[3,  1536] loss: 0.020
[4,   512] loss: 0.017
```

```
[4,  1024] loss: 0.016
[4,  1536] loss: 0.016
[5,   512] loss: 0.014
[5,  1024] loss: 0.015
[5,  1536] loss: 0.013
Finished Training
```

```
In [0]: PATH = './MNIST.pth'
        torch.save(net.state_dict(), PATH)
```

1.5 Test the network on the testing data

```
In [0]: n_epochs = 5
        test_losses = []
        test_counter = [i*len(trainloader.dataset) for i in range(n_epochs + 1)]
        net.eval()
        test_loss = 0
        correct = 0
        with torch.no_grad():
          for data, target in testloader:
            output = net(data)
            test_loss += F.nll_loss(output, target, size_average=False).item()
            pred = output.data.max(1, keepdim=True)[1]
            correct += pred.eq(target.data.view_as(pred)).sum()
        test_loss /= len(testloader.dataset)
        test_losses.append(test_loss)
        print('\nTest set: Accuracy: {}/{} ({:.0f}%)\n'.format(
        correct, len(testloader.dataset),
        100. * correct / len(testloader.dataset)))
```

```
/usr/local/lib/python3.6/dist-packages/torch/nn/_reduction.py:43: UserWarning: size_average and
  warnings.warn(warning.format(ret))



Test set: Accuracy: 9848/10000 (98%)
```

0.0.2 2 Problem 17.2

2.1 Loading the Dataset MNIST

```
In [0]: ##Load MNIST DATA

        # url= 'https://drive.google.com/uc?export=download&id=1Mvkkzx-XtOCTgvCCLMwO_FWHpF3C_Y
        # r = requests.get(url, allow_redirects=True)
        # open('MNISTdata.hdf5', 'wb').write(r.content)
```

```python
# MNIST_DATA=h5py.File('MNISTdata.hdf5', 'r')
# x_train= np.float32(MNIST_DATA['x_train'][:])
# y_train= np.int32(np.array(MNIST_DATA['y_train'][:,0]))
# x_test= np.float32(MNIST_DATA['x_test'][:])
# y_test=np.int32(np.array(MNIST_DATA['y_test'][:,0]))

batch_size= 124

train_dataset= torchvision.datasets.MNIST(root= './', train= True, transform= transfor
test_dataset=torchvision.datasets.MNIST(root= './', transform= transforms.ToTensor(), 


train_loader= torch.utils.data.DataLoader(dataset= train_dataset, batch_size=batch_siz
test_loader= torch.utils.data.DataLoader(dataset= test_dataset, batch_size=batch_size,
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./MNIST/raw/train-i


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


Extracting ./MNIST/raw/train-images-idx3-ubyte.gz to ./MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./MNIST/raw/train-l


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


Extracting ./MNIST/raw/train-labels-idx1-ubyte.gz to ./MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./MNIST/raw/t10k-ima


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


Extracting ./MNIST/raw/t10k-images-idx3-ubyte.gz to ./MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ./MNIST/raw/t10k-lab


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


Extracting ./MNIST/raw/t10k-labels-idx1-ubyte.gz to ./MNIST/raw
Processing...
Done!
```

2.2 Define a Convolutional Neural Network

In [0]:
```python
class network(nn.Module):
    def __init__(self):
        super(network, self).__init__()
        self.conv1 = nn.Conv2d(1,20,5,stride=1,padding=0)
        # self.dropout1 = nn.Dropout(0.30)
        self.MaxPool1= nn.MaxPool2d(2, stride=2)


        self.conv2 = nn.Conv2d(20,50,5,stride=1, padding=0)
        # self.dropout2 = nn.Dropout(0.30)
        self.MaxPool2= nn.MaxPool2d(2, stride=2)


        self.conv3 = nn.Conv2d(50,500,4,stride=1, padding=0)
        # self.conv3_bn= nn.BatchNorm2d(num_features=500,track_running_stats=False)

        self.conv4 = nn.Conv2d(500,10,1,stride=1, padding=0)
        # self.conv4_bn= nn.BatchNorm2d(num_features=10,track_running_stats=False)


        # self.dropoutfc= nn.Dropout(0.30)
        # self.fc3 = nn.Linear(500,10)

    def forward(self, x):
        # x= self.MaxPool1(self.dropout1(self.conv1(x)))
        # x= self.MaxPool2(self.dropout2(self.conv2(x)))

        x= self.MaxPool1((self.conv1(x)))
        x= self.MaxPool2((self.conv2(x)))
        # x= self.conv3_bn((self.conv3(x)))
        x= self.conv3(x)
        x= F.relu(x)
        x= self.conv4(x)
        # x= self.conv4_bn(x)

        # print (x.shape)
        # print (x.flatten().shape)
        return (x.flatten().reshape(x.shape[0],10))
```

In [0]:
```python
device= torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model= network().to(device)

num_epochs=30
learning_rate= 0.001
TrainLoss=[]
Train_Accuracy=[]
```

```
scheduler_step_size=5
gamma_scheduler=0.1
```

2.3 Define a loss function

```
In [0]: # optimizer= optim.Adagrad(model.parameters(), lr=0.01)
        optimizer= optim.SGD(model.parameters(), lr=learning_rate)
        #scheduler= torch.optim.lr_scheduler.StepLR(optimizer,step_size=scheduler_step_size, g
        criterion= nn.CrossEntropyLoss()
```

2.4 Train the network on the training data

```
In [0]: for epoch in range(num_epochs):
            model.train()
            time1= time.time()
           # scheduler.step()
            total=0
            correct=0

            for images,labels in train_loader:
                images= images.to(device)
                labels= labels.to(device)

                ##FORWARD PASS
                output= model(images)
                loss= criterion(output,labels)
                TrainLoss.append(loss.item())

                ##ACCURACY
                y_prediction= output.data.max(1)[1]
                total += labels.size(0)
                correct += (float((y_prediction.eq(labels)).sum()))


                ##BACKWARD
                optimizer.zero_grad()
                loss.backward()
                optimizer.step()

            Accuracy_Train= 100*(correct)/total


            time2= time.time()
            print('AT EPOCH:', epoch, ' THE TRAINING ACCURACY IS:', Accuracy_Train, '%', 'time
```

2.5 Test the network on the testing data

```
In [0]:
```

```
        ##TEST DATA
        model.eval()
        test_Accuracy= []
        total= 0
        correct=0
        TestLoss= []

        for images,labels in test_loader:
            images= images.to(device)
            labels= labels.to(device)
            output= model(images)
            y_prediction= output.data.max(1)[1]
            loss_test= criterion(output,labels)
            TestLoss.append(loss_test.item())

            total += labels.size(0)
            correct += (float((y_prediction.eq(labels)).sum()))*100

        Accuracy_Test= (correct)/total

        print ('The Accuracy in Test Set is:', Accuracy_Test, 'Mean Test Loss: ', np.mean(T
```

```
AT EPOCH: 0  THE TRAINING ACCURACY IS: 23.131666666666668 % time: 4.251698017120361
The Accuracy in Test Set is: 29.86 Mean Test Loss:  2.2762233828320917
AT EPOCH: 1  THE TRAINING ACCURACY IS: 36.24333333333333 % time: 4.232872724533081
The Accuracy in Test Set is: 42.91 Mean Test Loss:  2.2434597898412636
AT EPOCH: 2  THE TRAINING ACCURACY IS: 49.89333333333333 % time: 4.215153694152832
The Accuracy in Test Set is: 56.6 Mean Test Loss:  2.1915572602071878
AT EPOCH: 3  THE TRAINING ACCURACY IS: 61.74 % time: 4.184246063232422
The Accuracy in Test Set is: 65.38 Mean Test Loss:  2.0920480945963917
AT EPOCH: 4  THE TRAINING ACCURACY IS: 68.08666666666667 % time: 4.185768365859985
The Accuracy in Test Set is: 69.94 Mean Test Loss:  1.8622394653014194
AT EPOCH: 5  THE TRAINING ACCURACY IS: 71.61333333333333 % time: 4.212546348571777
The Accuracy in Test Set is: 75.13 Mean Test Loss:  1.3870188836698178
AT EPOCH: 6  THE TRAINING ACCURACY IS: 77.295 % time: 4.210388660430908
The Accuracy in Test Set is: 80.27 Mean Test Loss:  0.9010405981982196
AT EPOCH: 7  THE TRAINING ACCURACY IS: 81.24833333333333 % time: 4.191043853759766
The Accuracy in Test Set is: 83.58 Mean Test Loss:  0.6528593629230688
AT EPOCH: 8  THE TRAINING ACCURACY IS: 83.98833333333333 % time: 4.189209461212158
The Accuracy in Test Set is: 85.48 Mean Test Loss:  0.5369231185795348
AT EPOCH: 9  THE TRAINING ACCURACY IS: 85.87833333333333 % time: 4.260404586791992
The Accuracy in Test Set is: 86.82 Mean Test Loss:  0.4708290168164689
AT EPOCH: 10  THE TRAINING ACCURACY IS: 87.0 % time: 4.242781162261963
The Accuracy in Test Set is: 88.07 Mean Test Loss:  0.4266984008344603
AT EPOCH: 11  THE TRAINING ACCURACY IS: 87.935 % time: 4.203297853469849
The Accuracy in Test Set is: 88.68 Mean Test Loss:  0.3963590997420711
AT EPOCH: 12  THE TRAINING ACCURACY IS: 88.57833333333333 % time: 4.2004804611206055
```

```
The Accuracy in Test Set is: 89.36 Mean Test Loss:  0.3724690511546753
AT EPOCH: 13  THE TRAINING ACCURACY IS: 89.12666666666667 % time: 4.184293985366821
The Accuracy in Test Set is: 89.57 Mean Test Loss:  0.35608190385463795
AT EPOCH: 14  THE TRAINING ACCURACY IS: 89.60666666666667 % time: 4.1846911907196045
The Accuracy in Test Set is: 90.15 Mean Test Loss:  0.3371802564296458
AT EPOCH: 15  THE TRAINING ACCURACY IS: 90.025 % time: 4.201342821121216
The Accuracy in Test Set is: 90.45 Mean Test Loss:  0.324088429917156
AT EPOCH: 16  THE TRAINING ACCURACY IS: 90.36 % time: 4.254647970199585
The Accuracy in Test Set is: 90.95 Mean Test Loss:  0.31001562744746974
AT EPOCH: 17  THE TRAINING ACCURACY IS: 90.68166666666667 % time: 4.25424337387085
The Accuracy in Test Set is: 91.29 Mean Test Loss:  0.2994886639493483
AT EPOCH: 18  THE TRAINING ACCURACY IS: 90.99833333333333 % time: 4.208758354187012
The Accuracy in Test Set is: 91.47 Mean Test Loss:  0.2889221763169324
AT EPOCH: 19  THE TRAINING ACCURACY IS: 91.23333333333333 % time: 4.195350646972656
The Accuracy in Test Set is: 91.56 Mean Test Loss:  0.28127906393305757
AT EPOCH: 20  THE TRAINING ACCURACY IS: 91.57 % time: 4.1824212074279785
The Accuracy in Test Set is: 92.12 Mean Test Loss:  0.2701680617016039
AT EPOCH: 21  THE TRAINING ACCURACY IS: 91.805 % time: 4.186009645462036
The Accuracy in Test Set is: 92.33 Mean Test Loss:  0.2604909411597031
AT EPOCH: 22  THE TRAINING ACCURACY IS: 92.01666666666667 % time: 4.200885772705078
The Accuracy in Test Set is: 92.46 Mean Test Loss:  0.2536887782453387
AT EPOCH: 23  THE TRAINING ACCURACY IS: 92.28666666666666 % time: 4.2011168003082275
The Accuracy in Test Set is: 92.75 Mean Test Loss:  0.24622060682762553
AT EPOCH: 24  THE TRAINING ACCURACY IS: 92.44666666666667 % time: 4.212292194366455
The Accuracy in Test Set is: 92.87 Mean Test Loss:  0.2394483265363508
AT EPOCH: 25  THE TRAINING ACCURACY IS: 92.66333333333333 % time: 4.220020532608032
The Accuracy in Test Set is: 93.1 Mean Test Loss:  0.23229993810808217
AT EPOCH: 26  THE TRAINING ACCURACY IS: 92.79666666666667 % time: 4.187253713607788
The Accuracy in Test Set is: 93.32 Mean Test Loss:  0.22612452758821072
AT EPOCH: 27  THE TRAINING ACCURACY IS: 93.05666666666667 % time: 4.184278964996338
The Accuracy in Test Set is: 93.43 Mean Test Loss:  0.21979578091176571
AT EPOCH: 28  THE TRAINING ACCURACY IS: 93.22833333333334 % time: 4.195254564285278
The Accuracy in Test Set is: 93.54 Mean Test Loss:  0.21559509319931636
AT EPOCH: 29  THE TRAINING ACCURACY IS: 93.38 % time: 4.232873201370239
The Accuracy in Test Set is: 93.81 Mean Test Loss:  0.20760673690403317
```

In [0]:
```
# torch.save(model,'/content/drive/My Drive/CS-498 Applied Machine Learning/HW8/Nidia/
torch.save({'state_dict': model.state_dict()}, '/content/drive/My Drive/CS-498 Applied
```