Spring 2023 Introduction to Deep Learning

Homework Assignment 4

Due date: April 25 2023

Problem (Build LeNet for colorful image classification). In this problem, you are asked to train and test a neural network for ***entire*** CIFAR-10 colorful image dataset. Some information of the network is as follows:

- Its structure is **modified LeNet**. You can check the 4th slide in Lecture 10 for details.
- An incomplete code has been given. You can fill it or re-write all the codes by yourself.

Performance Requirement and Submission:

- The test accuracy should achieve above 50%
- You need to submit **three** results: 1) network without dropout/batch normalization, 2) network with one additional dropout layer and 3) network with one additional batch normalization. Compare the results in your submission.
- Submission should include your source codes and screen snapshot of your train and test accuracy, plus the training time

Suggestion for hyperparameter setting (not necessary to follow): Check the default setting in the code. You are allowed to change them

About dataset loading: Check the default setting in the code. You are allowed to change them

Reminding: You can check PyTorch *torch.nn* to find the packed Batch Normalization and Dropout layer if you would like to use.

Model w/o dropout and batch normalization :-

```
class LeNet(nn.Module):
    def __init__(self, num_classes=10, grayscale=False):
        super(LeNet, self).__init__()
        self.grayscale = grayscale
        self.num_classes = num_classes

        if self.grayscale:
            in_channels = 1
        else:
            in_channels = 3

        self.conv1 = nn.Sequential(OrderedDict([
            ('c1',nn.Conv2d(in_channels,6,kernel_size=(5,5))),
            ('relu1',nn.ReLU()),
            ('s1',nn.MaxPool2d(kernel_size=(2,2),stride=2))
        ]))

        self.conv2 = nn.Sequential(OrderedDict([
            ('c2',nn.Conv2d(6,16,kernel_size=(5,5))),
            ('relu2',nn.ReLU()),
            ('s2',nn.MaxPool2d(kernel_size=(2,2),stride=2))
        ]))

        self.conv3 = nn.Sequential(OrderedDict([
            ('c3',nn.Conv2d(16,120,kernel_size=(5,5))),
            ('relu3',nn.ReLU()),
        ]))

        self.flatten = nn.Flatten()

        self.fc1 = nn.Sequential(OrderedDict([
            ('f5',nn.Linear(120,84)),
            ('relu5',nn.ReLU())
        ]))

        self.fc2 = nn.Sequential(OrderedDict([
            ('f6',nn.Linear(84,num_classes)),
            ('sig6',nn.LogSoftmax(dim=-1))
        ]))


    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)

        x = self.flatten(x)

        x = self.fc1(x)
        out = self.fc2(x)
        return out
```

Model w/o dropout and batch normalization Training:-

```
def Net():
    time0 = time.time()

    # Training settings
    batch_size = 128
    epochs = 5
    lr = 0.05
    no_cuda = False
    save_model = False

    use_cuda = not no_cuda and torch.cuda.is_available()
    torch.manual_seed(100)

    device = torch.device("cuda" if use_cuda else "cpu")

    trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform_train)
    train_loader = torch.utils.data.DataLoader(trainset, batch_size=128, shuffle=True)

    testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform_test)
    test_loader = torch.utils.data.DataLoader(testset, batch_size=100, shuffle=False)

    model = LeNet().to(device)
    optimizer = optim.SGD(model.parameters(), lr=lr, momentum=0.9, weight_decay=5e-4)

    for epoch in range(1, epochs + 1):
        train( model, device, train_loader, optimizer, epoch)
        test( model, device, test_loader)

    if (save_model):
        torch.save(model.state_dict(),"cifar_lenet.pt")

    time1 = time.time()
    print ('Traning and Testing total excution time is: %s seconds ' % (time1-time0))
```

```
Net()
```

```
Train Epoch: 5 [47360/50000 (95%)]      Loss: 1.432482
Train Epoch: 5 [48640/50000 (97%)]      Loss: 1.323309
Train Epoch: 5 [31200/50000 (100%)]     Loss: 1.412259

Test set: Average loss: 1.3730, Accuracy: 5189/10000 (52%)

Traning and Testing total excution time is: 82.6780858039856 seconds
```

To compare and observe the difference a batch normalization and dropout layers make, we first run a base model. We see that with 5 epochs we achieve a accuracy of 54% in about 83 sec.

Model w/ dropout :-

```
class LeNet_Dropout(nn.Module):
    def __init__(self, num_classes=10, grayscale=False):
        super(LeNet_Dropout, self).__init__()
        self.grayscale = grayscale
        self.num_classes = num_classes

        if self.grayscale:
            in_channels = 1
        else:
            in_channels = 3

        self.conv1 = nn.Sequential(OrderedDict([
            ('c1',nn.Conv2d(in_channels,6,kernel_size=(5,5))),
            ('relu1',nn.ReLU()),
            ('s1',nn.MaxPool2d(kernel_size=(2,2),stride=2))
        ]))

        self.conv2 = nn.Sequential(OrderedDict([
            ('c2',nn.Conv2d(6,16,kernel_size=(5,5))),
            ('relu2',nn.ReLU()),
            ('s2',nn.MaxPool2d(kernel_size=(2,2),stride=2))
        ]))

        self.conv3 = nn.Sequential(OrderedDict([
            ('c3',nn.Conv2d(16,120,kernel_size=(5,5))),
            ('relu3',nn.ReLU()),
        ]))

        self.flatten = nn.Flatten()

        self.dropout = nn.Dropout(0.5)

        self.fc1 = nn.Sequential(OrderedDict([
            ('f5',nn.Linear(120,84)),
            ('relu5',nn.ReLU())
        ]))

        self.fc2 = nn.Sequential(OrderedDict([
            ('f6',nn.Linear(84,num_classes)),
            ('sig6',nn.LogSoftmax(dim=-1))
        ]))

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.flatten(x)
        x = self.dropout(x)
        x = self.fc1(x)
        out = self.fc2(x)
        return out
```

Model w/ dropout Training :-

```
def Net_Dropout():
    time0 = time.time()

    # Training settings
    batch_size = 128
    epochs = 10
    lr = 0.05
    no_cuda = False
    save_model = False

    use_cuda = not no_cuda and torch.cuda.is_available()
    torch.manual_seed(100)

    device = torch.device("cuda" if use_cuda else "cpu")

    trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform_train)
    train_loader = torch.utils.data.DataLoader(trainset, batch_size=128, shuffle=True)

    testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform_test)
    test_loader = torch.utils.data.DataLoader(testset, batch_size=100, shuffle=False)

    model = LeNet_Dropout().to(device)
    optimizer = optim.SGD(model.parameters(), lr=lr, momentum=0.9, weight_decay=5e-4)

    for epoch in range(1, epochs + 1):
        train( model, device, train_loader, optimizer, epoch)
        test( model, device, test_loader)

    if (save_model):
        torch.save(model.state_dict(),"cifar_lenet.pt")

    time1 = time.time()
    print ('Traning and Testing total excution time is: %s seconds ' % (time1-time0))


Net_Dropout()
```

```
Train Epoch: 5 [31200/50000 (100%)]     Loss: 1.537211

Test set: Average loss: 1.3654, Accuracy: 5096/10000 (51%)

Train Epoch: 6 [0/50000 (0%)]   Loss: 1.647001
Train Epoch: 10 [48640/50000 (97%)]     Loss: 1.473060
Train Epoch: 10 [31200/50000 (100%)]    Loss: 1.539141

Test set: Average loss: 1.3678, Accuracy: 5308/10000 (53%)

Traning and Testing total excution time is: 158.8505666255951 seconds
```

After adding one layer of dropout between the last two hidden layers, we observe that training takes longer than base model and we also see a drop in accuracy. A dropout layer turns an input to 0 at probability of p, allowing us to combat overfitting. However, on the other hand this means it takes longer for the model to train to almost the same amount of accuracy as the base model.

Model w/ batch normalization :-

```
class LeNet_Normalization(nn.Module):
    def __init__(self, num_classes=10, grayscale=False):
        super(LeNet_Normalization, self).__init__()
        self.grayscale = grayscale
        self.num_classes = num_classes

        if self.grayscale:
            in_channels = 1
        else:
            in_channels = 3

        self.conv1 = nn.Sequential(OrderedDict([
            ('c1',nn.Conv2d(in_channels,6,kernel_size=(5,5))),
            ('relu1',nn.ReLU()),
            ('BatchNorm1',nn.BatchNorm2d(6)), # Normalize After Activation
            ('s1',nn.MaxPool2d(kernel_size=(2,2),stride=2))
        ]))

        self.conv2 = nn.Sequential(OrderedDict([
            ('c2',nn.Conv2d(6,16,kernel_size=(5,5))),
            ('relu2',nn.ReLU()),
            ('s2',nn.MaxPool2d(kernel_size=(2,2),stride=2))
        ]))

        self.conv3 = nn.Sequential(OrderedDict([
            ('c3',nn.Conv2d(16,120,kernel_size=(5,5))),
            ('relu3',nn.ReLU()),
        ]))

        self.flatten = nn.Flatten()

        self.fc1 = nn.Sequential(OrderedDict([
            ('f5',nn.Linear(120,84)),
            ('relu5',nn.ReLU())
        ]))

        self.fc2 = nn.Sequential(OrderedDict([
            ('f6',nn.Linear(84,num_classes)),
            ('sig6',nn.LogSoftmax(dim=-1))
        ]))

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.flatten(x)
        x = self.fc1(x)
        out = self.fc2(x)
        return out
```

Model w/ batch normalization Training:-

```
def Net_Normalization():
    time0 = time.time()

    # Training settings
    batch_size = 128
    epochs = 5
    lr = 0.05
    no_cuda = False
    save_model = False

    use_cuda = not no_cuda and torch.cuda.is_available()
    torch.manual_seed(100)

    device = torch.device("cuda" if use_cuda else "cpu")

    trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform_train)
    train_loader = torch.utils.data.DataLoader(trainset, batch_size=128, shuffle=True)

    testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform_test)
    test_loader = torch.utils.data.DataLoader(testset, batch_size=100, shuffle=False)

    model = LeNet_Normalization().to(device)
    optimizer = optim.SGD(model.parameters(), lr=lr, momentum=0.9, weight_decay=5e-4)

    for epoch in range(1, epochs + 1):
        train( model, device, train_loader, optimizer, epoch)
        test( model, device, test_loader)

    if (save_model):
        torch.save(model.state_dict(),"cifar_lenet.pt")

    time1 = time.time()
    print ('Traning and Testing total excution time is: %s seconds ' % (time1-time0))
```

```
Net_Normalization()
```

```
Train Epoch: 5 [47360/50000 (95%)]      Loss: 1.205444
Train Epoch: 5 [48640/50000 (97%)]      Loss: 1.201012
Train Epoch: 5 [31200/50000 (100%)]     Loss: 1.562015

Test set: Average loss: 1.1681, Accuracy: 5907/10000 (59%)

Traning and Testing total excution time is: 81.09823203086853 seconds
```

When we add only one layer of batch normalization, we observe that we achieve higher accuracy in less time than the base model. Thus, batch normalization after activation of convolution layer can help reduce training time of the model, and even help with the accuracy of the model.