CS 559 - Fall 2021
Homework #6

Krishna Garg
kgarg8@uic.edu

1 Default Hyperparameters

Optimizer: Adam

Weight Decay: 0

Batch size: 100

Learning rate: 1e-3

Scheduler: StepLR

Scheduler step size: 1

Scheduler gamma: 0.7

Loss: CrossEntropyLoss

2 Network

2-layer CNN: Each layer of Conv2d is followed by Relu activation function, followed by Max Pooling layer & 2 layers each of Dropout and Fully connected layer. The inputs are 3-channel images of shape (200 * 200) and the output is one of the 9 classes corresponding to each image. The training dataset has 72k images and the testing dataset contains 18k images.

3 What worked and what didn't work

3.1 80-20 split with default hyperparameters

Didn't work - 100% training accuracy but 87.40% testing accuracy in 1-2 epochs

3.2 60-20-20 split with default hyperparameters

Didn't work - 100% training accuracy but 87.40% accuracy on both val and test datasets in 1-2 epochs

3.3 Tuning the optimizer Adam, with weight_decay (L2 regularization)

3.3.1 weight_decay = 0.003

Didn't work, gave same results

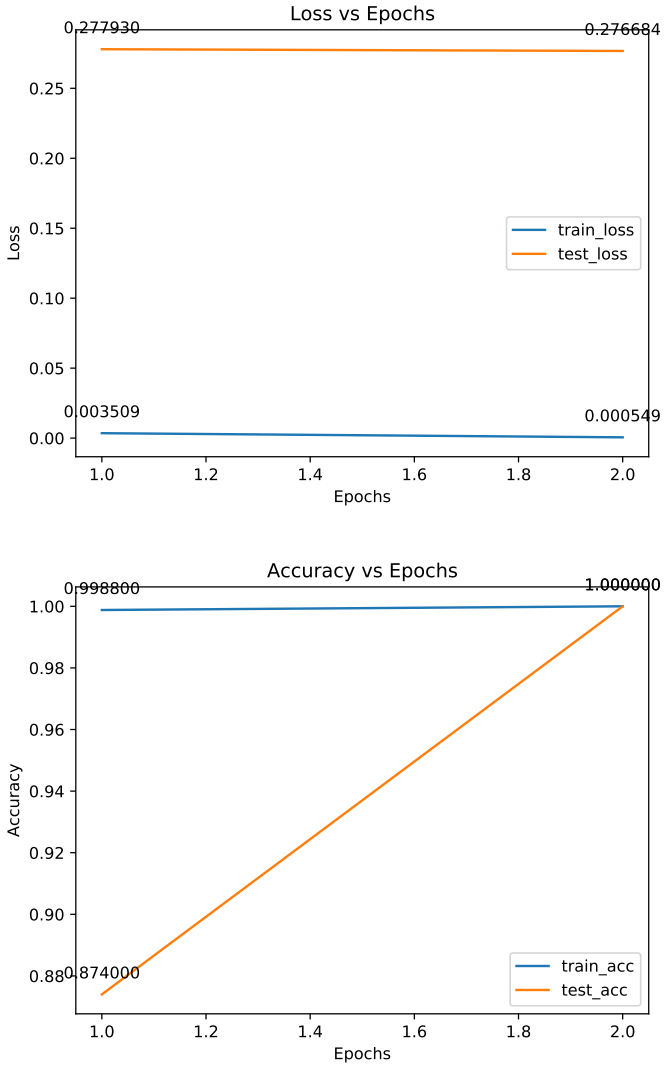

Figure 1: Plots: (a) Loss vs Epochs, (b) Accuracy vs Epochs for both training and testing sets

3.3.2 weight_decay = 0.03

Worked, gave 100% accuracy on all datasets. The L2 regularization added helped to overcome the overfitting problem in the above cases and led to build better generalization capability of the model.

Code - 0601-656377418-Garg.py

```
# Code heavily inspired from:
    https://github.com/pytorch/examples/blob/master/mnist/main.py

# training
import argparse, torch, torch.nn as nn, torch.nn.functional as F,
    torch.optim as optim, glob, shutil, random, numpy as np, math
from torchvision import datasets, transforms
from torch.optim.lr_scheduler import StepLR

# Uncomment to create separate folders for train/ test
# random.seed(112)
# for item in ['Circle', 'Square', 'Octagon', 'Heptagon', 'Nonagon',
    'Star', 'Hexagon', 'Pentagon', 'Triangle']:
#    mylist = [f for f in glob.glob('output/{}*'.format(item))]
#    random.shuffle(mylist)
#    for filename in mylist[:8000]:
#       shutil.copy(filename, 'train_original/images/')
#    for filename in mylist[8000:]:
#       shutil.copy(filename, 'test_original/images/')

class Net(nn.Module):
   def __init__(self):
      super(Net, self).__init__()
      self.conv1 = nn.Conv2d(3, 32, 3, 1)
      self.conv2 = nn.Conv2d(32, 64, 3, 1)
      self.dropout1 = nn.Dropout(0.25)
      self.dropout2 = nn.Dropout(0.5)
      self.fc1   = nn.Linear(614656, 128)
      self.fc2   = nn.Linear(128, 9)

   def forward(self, x):
      x = self.conv1(x)
      x = F.relu(x)
      x = self.conv2(x)
      x = F.relu(x)
      x = F.max_pool2d(x, 2)
      x = self.dropout1(x)
      x = torch.flatten(x, 1)
      x = self.fc1(x)
      x = F.relu(x)
      x = self.dropout2(x)
      x = self.fc2(x)
      return x

def train(args, model, device, train_loader, optimizer, epoch):
   model.train()
   tot_loss = 0
   correct = 0
   for batch_idx, (data, target) in enumerate(train_loader):
      data, target = data.to(device), target.to(device)
      optimizer.zero_grad()
      output = model(data)
      loss = torch.nn.CrossEntropyLoss()(output, target)
      loss.backward()
      optimizer.step()

      pred  = output.argmax(dim=1, keepdim=True)
      correct += pred.eq(target.view_as(pred)).sum().item()

      tot_loss = tot_loss + loss.item()
      if batch_idx % args.log_interval == 0:
         print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f},
             Accuracy: {:.2f}%'.format(
```

```
            epoch, batch_idx * len(data), len(train_loader.dataset),
            100. * batch_idx / len(train_loader),
                tot_loss/(batch_idx+1),
                100.0*correct/((batch_idx+1)*args.batch_size)))

    loss = tot_loss / len(train_loader)
    acc = 100.0 * correct / (len(train_loader) * args.batch_size)
    print('End of Epoch: {}'.format(epoch))
    print('Training Loss: {:.6f}, Training Accuracy:
        {:.2f}%'.format(loss, acc))

def test(args, model, device, test_loader):
    model.eval()
    tot_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output    = model(data)
            tot_loss  += torch.nn.CrossEntropyLoss()(output, target).item()
                # sum up batch loss
            pred      = output.argmax(dim=1, keepdim=True) # get the index
                of the max log-probability
            correct   += pred.eq(target.view_as(pred)).sum().item()

    loss = tot_loss / len(test_loader)
    acc = 100.0 * correct / (len(test_loader) * args.test_batch_size)
    print('Test Loss: {:.6f}, Test Accuracy: {:.2f}%'.format(loss, acc))
    return loss, acc

def main():
    # Training settings
    parser = argparse.ArgumentParser(description='PyTorch MNIST Example')
    parser.add_argument('--batch-size', type=int, default=100,
        help='input batch size for training (default: 64)')
    parser.add_argument('--test-batch-size', type=int, default=1000,
        help='input batch size for testing (default: 1000)')
    parser.add_argument('--epochs', type=int, default=14, help='number of
        epochs to train (default: 14)')
    parser.add_argument('--lr', type=float, default=1e-3, help='learning
        rate (default: 1.0)')
    parser.add_argument('--gamma', type=float, default=0.7,
        help='Learning rate step gamma (default: 0.7)')
    parser.add_argument('--seed', type=int, default=112, help='random
        seed (default: 112)')
    parser.add_argument('--log-interval', type=int, default=100,
        help='how many batches to wait before logging training status')
    parser.add_argument('--save-model', action='store_true',
        default=True, help='For Saving the current Model')
    args = parser.parse_args()
    print(args.save_model)

    random.seed(args.seed)
    np.random.seed(args.seed)
    torch.manual_seed(args.seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False

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

    transform=transforms.Compose([transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))])
    dataset1  = datasets.ImageFolder('train_original/',
        transform=transform)
```

```
    dataset2  = datasets.ImageFolder('test_original/',
        transform=transform)
    train_loader = torch.utils.data.DataLoader(dataset1,
        batch_size=args.batch_size, shuffle=True)
    test_loader = torch.utils.data.DataLoader(dataset2,
        batch_size=args.test_batch_size)

    model = Net().to(device)
    optimizer = optim.Adam(model.parameters(), lr=args.lr,
        weight_decay=0.03)

    scheduler = StepLR(optimizer, step_size=1, gamma=args.gamma)
    for epoch in range(1, args.epochs + 1):
       train(args, model, device, train_loader, optimizer, epoch)
       test_loss, test_acc = test(args,model, device, test_loader)
       scheduler.step()

       if args.save_model: # save new model after every epoch
          torch.save(model.state_dict(), "0602-656377418-Garg.pt")

       if test_loss < 1e-4 or math.isclose(test_acc, 100.0):
          break

if __name__ == '__main__':
    main()
```

Code - 0603-656377418-Garg.py

```
# inference
import torch, torch.nn as nn, torch.nn.functional as F, random, numpy as
    np
from torchvision import datasets, transforms

class Net(nn.Module):
    def __init__(self):
       super(Net, self).__init__()
       self.conv1 = nn.Conv2d(3, 32, 3, 1)
       self.conv2 = nn.Conv2d(32, 64, 3, 1)
       self.dropout1 = nn.Dropout(0.25)
       self.dropout2 = nn.Dropout(0.5)
       self.fc1   = nn.Linear(614656, 128)
       self.fc2   = nn.Linear(128, 9)

    def forward(self, x):
       x = self.conv1(x)
       x = F.relu(x)
       x = self.conv2(x)
       x = F.relu(x)
       x = F.max_pool2d(x, 2)
       x = self.dropout1(x)
       x = torch.flatten(x, 1)
       x = self.fc1(x)
       x = F.relu(x)
       x = self.dropout2(x)
       x = self.fc2(x)
       return x

test_batch_size = 100
saved_model_path = '0602-656377418-Garg.pt'
device       = torch.device("cuda" if torch.cuda.is_available() else
    "cpu")
checkpoint   = torch.load(saved_model_path, map_location=device)
model        = Net().to(device)
model.load_state_dict(checkpoint)
```

```
model.eval()

transform = transforms.Compose([transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))])
test_dataset = datasets.ImageFolder('test_original/',
    transform=transform)
test_loader = torch.utils.data.DataLoader(test_dataset,
    batch_size=test_batch_size)

tot_loss = 0
correct = 0
with torch.no_grad():
   for data, target in test_loader:
      data, target = data.to(device), target.to(device)
      output    = model(data)
      tot_loss  += torch.nn.CrossEntropyLoss()(output, target).item() #
          sum up batch loss
      pred      = output.argmax(dim=1, keepdim=True) # get the index of
          the max log-probability
      correct   += pred.eq(target.view_as(pred)).sum().item()

print('Test Loss: {:.6f}, Test Accuracy: {:.2f}%'.format(
   tot_loss/(len(test_loader)),
       100.0*correct/(len(test_loader)*test_batch_size)))
```