CS498 AML HW11

Huiyun Wu (hwu63)

Yidi Yang (yyang160)

Part 1. Show grid of 10 x 9 for a single MNIST digit- decoded images.

Part2 : Show a grid of 10 x 9 for different digits - decoded images.

Page 3: Important lines of your code.

Code for same digit pairs

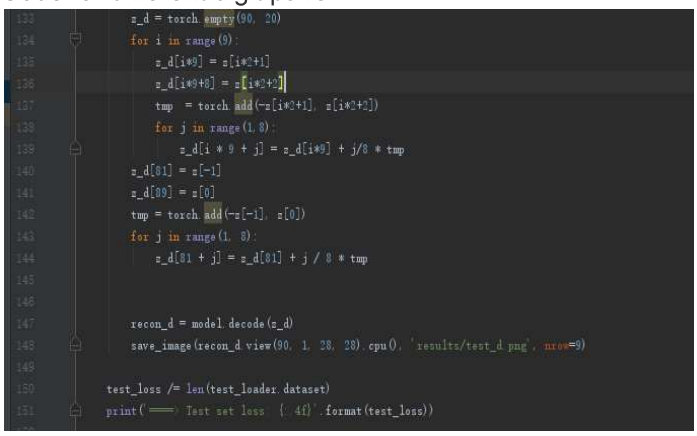
```
def test(epoch):
    model.eval()
    test_loss = 0
    with torch.no_grad():
        for i, (data, label) in enumerate(test_loader):

            count = 0
            if i == 0:
                for k in range(10):
                    for j,l in enumerate(label):
                            if k == l:
                                if k == 0 and count == 0 :
                                    chosen = torch.stack([data[j]], 1)
                                    # print(k)
                                    count += 1
                                else:
                                    tmp = torch.stack([data[j]], 1)
                                    # print(2, chosen)
                                    chosen = torch.cat([chosen, tmp])
                                    # print(k)
                                    count += 1
                                    if count == 2 :
                                        count = 0
                                        break

        recon_batch, mu, logvar, z = model(chosen)
        z_ = torch.empty(90, 20)
        for i in range(10):
            z_[i*9] = z[i*2]
            z_[i*9+8] = z[i*2+1]
            tmp = torch.add(-z[i*2], z[i*2+1])
            for j in range(1,8):
                z_[i * 9 + j] = z_[i*9] + j/8 * tmp

        recon = model.decode(z_)
        save_image(recon.view(90, 1, 28, 28).cpu(), 'results/test.png', nrow=9)
```

Code for different digit pairs



```
        z_d = torch.empty(90, 20)
        for i in range(9):
            z_d[i*9] = z[i*2+1]
            z_d[i*9+8] = z[i*2+2]
            tmp = torch.add(-z[i*2+1], z[i*2+2])
            for j in range(1,8):
                z_d[i * 9 + j] = z_d[i*9] + j/8 * tmp
        z_d[81] = z[-1]
        z_d[89] = z[0]
        tmp = torch.add(-z[-1], z[0])
        for j in range(1, 8):
            z_d[81 + j] = z_d[81] + j / 8 * tmp


        recon_d = model.decode(z_d)
        save_image(recon_d.view(90, 1, 28, 28).cpu(), 'results/test_d.png', nrow=9)

    test_loss /= len(test_loader.dataset)
    print('====> Test set loss: {:.4f}'.format(test_loss))
```

Page 4 - onwards: Your entire code.

Code reference: https://github.com/pytorch/examples/blob/master/vae/main.py

```python
from __future__ import print_function
import argparse
import torch
import torch.utils.data
from torch import nn, optim
from torch.nn import functional as F
from torchvision import datasets, transforms
from torchvision.utils import save_image


parser = argparse.ArgumentParser(description='VAE MNIST Example')
parser.add_argument('--batch-size', type=int, default=128, metavar='N',
                    help='input batch size for training (default: 128)')
parser.add_argument('--epochs', type=int, default=1, metavar='N',
                    help='number of epochs to train (default: 10)')
parser.add_argument('--no-cuda', action='store_true', default=False,
                    help='enables CUDA training')
parser.add_argument('--seed', type=int, default=1, metavar='S',
                    help='random seed (default: 1)')
parser.add_argument('--log-interval', type=int, default=10, metavar='N',
                    help='how many batches to wait before logging training status')
args = parser.parse_args()
args.cuda = not args.no_cuda and torch.cuda.is_available()

torch.manual_seed(args.seed)

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

kwargs = {'num_workers': 1, 'pin_memory': True} if args.cuda else {}
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=True, download=True,
                   transform=transforms.ToTensor()),
    batch_size=args.batch_size, shuffle=True, **kwargs)
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=False, transform=transforms.ToTensor()),
    batch_size=args.batch_size, shuffle=True, **kwargs)

class VAE(nn.Module):
    def __init__(self):
        super(VAE, self).__init__()

        self.fc1 = nn.Linear(784, 400)
        self.fc21 = nn.Linear(400, 20)
        self.fc22 = nn.Linear(400, 20)
        self.fc3 = nn.Linear(20, 400)
        self.fc4 = nn.Linear(400, 784)

    def encode(self, x):
        h1 = F.relu(self.fc1(x))
        return self.fc21(h1), self.fc22(h1)

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5*logvar)
        eps = torch.randn_like(std)
        return eps.mul(std).add_(mu)

    def decode(self, z):
        h3 = F.relu(self.fc3(z))
        return torch.sigmoid(self.fc4(h3))

    def forward(self, x):
        mu, logvar = self.encode(x.view(-1, 784))
        z = self.reparameterize(mu, logvar)
        return self.decode(z), mu, logvar, z


model = VAE().to(device)
optimizer = optim.Adam(model.parameters(), lr=1e-3)
```

```python
# Reconstruction + KL divergence losses summed over all elements and batch
def loss_function(recon_x, x, mu, logvar):
    BCE = F.binary_cross_entropy(recon_x, x.view(-1, 784), reduction='sum')
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())

    return BCE + KLD


def train(epoch):
    model.train()
    train_loss = 0
    for batch_idx, (data, label) in enumerate(train_loader):
        data = data.to(device)
        optimizer.zero_grad()
        recon_batch, mu, logvar, z = model(data)
        loss = loss_function(recon_batch, data, mu, logvar)
        loss.backward()
        train_loss += loss.item()
        optimizer.step()

    print('====> Epoch: {} Average loss: {:.4f}'.format(
          epoch, train_loss / len(train_loader.dataset)))
```

```python
def test(epoch):
    model.eval()
    test_loss = 0
    with torch.no_grad():
        for i, (data, label) in enumerate(test_loader):

            count = 0
            if i == 0:
                for k in range(10):
                    for j,l in enumerate(label):
                            if k == l:
                                if k == 0 and count == 0 :
                                    chosen = torch.stack([data[j]], 1)
                                    # print(k)
                                    count += 1
                                else:
                                    tmp = torch.stack([data[j]], 1)
                                    # print(2, chosen)
                                    chosen = torch.cat([chosen, tmp])
                                    # print(k)
                                    count += 1
                                    if count == 2 :
                                        count = 0
                                        break

            recon_batch, mu, logvar, z = model(chosen)
            z_ = torch.empty(90, 20)
            for i in range(10):
                z_[i*9] = z[i*2]
                z_[i*9+8] = z[i*2+1]
                tmp = torch.add(-z[i*2], z[i*2+1])
                for j in range(1,8):
                    z_[i * 9 + j] = z_[i*9] + j/8 * tmp

            recon = model.decode(z_)
            save_image(recon.view(90, 1, 28, 28).cpu(), 'results/test.png', nrow=9)
```

```python
            z_d = torch.empty(90, 20)
            for i in range(9):
                z_d[i*9] = z[i*2+1]
                z_d[i*9+8] = z[i*2+2]
                tmp = torch.add(-z[i*2+1], z[i*2+2])
                for j in range(1,8):
                    z_d[i * 9 + j] = z_d[i*9] + j/8 * tmp
            z_d[81] = z[-1]
            z_d[89] = z[0]
            tmp = torch.add(-z[-1], z[0])
            for j in range(1, 8):
                z_d[81 + j] = z_d[81] + j / 8 * tmp


            recon_d = model.decode(z_d)
            save_image(recon_d.view(90, 1, 28, 28).cpu(), 'results/test_d.png', nrow=9)

        test_loss /= len(test_loader.dataset)
        print('====> Test set loss: {:.4f}'.format(test_loss))
```

```python
if __name__ == "__main__":
    for epoch in range(1, args.epochs + 1):
        train(epoch)
    test(0)
```