

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

```
106 def test(epoch):
107     model.eval()
108     test_loss = 0
109     with torch.no_grad():
110         for i, (data, label) in enumerate(test_loader):
111             count = 0
112             if i == 0:
113                 for k in range(10):
114                     for j, l in enumerate(label):
115                         if k == l:
116                             if k == 0 and count == 0:
117                                 chosen = torch.stack([data[j]], 1)
118                                 = print(k)
119                                 count += 1
120                             else:
121                                 tmp = torch.stack([data[j]], 1)
122                                 = print(2, chosen)
123                                 chosen = torch.cat([chosen, tmp])
124                                 = print(k)
125                                 count += 1
126                                 if count == 2:
127                                     count = 0
128                                     break
129
130             recon_batch, mu, logvar, z = model(chosen)
131             z_ = torch.empty(90, 20)
132             for i in range(10):
133                 z_[i*9] = z[i*2]
134                 z_[i*9+8] = z[i*2+1]
135                 tmp = torch.add(-z[i*2], z[i*2+1])
136                 for j in range(1, 8):
137                     z_[i * 9 + j] = z_[i*9] + j/8 * tmp
138
139             recon = model.decode(z_)
140             save_image(recon.view(90, 1, 28, 28).cpu(), 'results/test.png', nrow=9)
```

Code for different digit pairs

```
133 z_d = torch.empty(90, 20)
134 for i in range(9):
135     z_d[i*9] = z[i*2+1]
136     z_d[i*9+8] = z[i*2+2]
137     tmp = torch.add(-z[i*2+1], z[i*2+2])
138     for j in range(1, 8):
139         z_d[i * 9 + j] = z_d[i*9] + j/8 * tmp
140 z_d[81] = z[-1]
141 z_d[89] = z[0]
142 tmp = torch.add(-z[-1], z[0])
143 for j in range(1, 8):
144     z_d[81 + j] = z_d[81] + j / 8 * tmp
145
146 recon_d = model.decode(z_d)
147 save_image(recon_d.view(90, 1, 28, 28).cpu(), 'results/test_d.png', nrow=9)
148
149 test_loss /= len(test_loader.dataset)
150 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>

```
1 from __future__ import print_function
2 import argparse
3 import torch
4 import torch.utils.data
5 from torch import nn, optim
6 from torch.nn import functional as F
7 from torchvision import datasets, transforms
8 from torchvision.utils import save_image
9
10
11 parser = argparse.ArgumentParser(description='VAE MNIST Example')
12 parser.add_argument('--batch-size', type=int, default=128, metavar='N',
13                     help='input batch size for training (default: 128)')
14 parser.add_argument('--epochs', type=int, default=1, metavar='N',
15                     help='number of epochs to train (default: 10)')
16 parser.add_argument('--no-cuda', action='store_true', default=False,
17                     help='enables CUDA training')
18 parser.add_argument('--seed', type=int, default=1, metavar='S',
19                     help='random seed (default: 1)')
20 parser.add_argument('--log-interval', type=int, default=10, metavar='N',
21                     help='how many batches to wait before logging training status')
22 args = parser.parse_args()
23 args.cuda = not args.no_cuda and torch.cuda.is_available()
24
25 torch.manual_seed(args.seed)
26
27 device = torch.device("cuda" if args.cuda else "cpu")
28
29 kwargs = {'num_workers': 1, 'pin_memory': True} if args.cuda else {}
30 train_loader = torch.utils.data.DataLoader(
31     datasets.MNIST('./data', train=True, download=True,
32                  transform=transforms.ToTensor()),
33     batch_size=args.batch_size, shuffle=True, **kwargs)
34 test_loader = torch.utils.data.DataLoader(
35     datasets.MNIST('./data', train=False, transform=transforms.ToTensor()),
36     batch_size=args.batch_size, shuffle=True, **kwargs)
37
38 class VAE(nn.Module):
39     def __init__(self):
40         super(VAE, self).__init__()
41
42         self.fc1 = nn.Linear(784, 400)
43         self.fc21 = nn.Linear(400, 20)
44         self.fc22 = nn.Linear(400, 20)
45         self.fc3 = nn.Linear(20, 400)
46         self.fc4 = nn.Linear(400, 784)
47
48     def encode(self, x):
49         h1 = F.relu(self.fc1(x))
50         return self.fc21(h1), self.fc22(h1)
51
52     def reparameterize(self, mu, logvar):
53         std = torch.exp(0.5*logvar)
54         eps = torch.randn_like(std)
55         return eps.mul(std).add_(mu)
56
57     def decode(self, z):
58         h3 = F.relu(self.fc3(z))
59         return torch.sigmoid(self.fc4(h3))
60
61     def forward(self, x):
62         mu, logvar = self.encode(x.view(-1, 784))
63         z = self.reparameterize(mu, logvar)
64         return self.decode(z), mu, logvar, z
65
66 model = VAE().to(device)
67 optimiser = optim.Adam(model.parameters(), lr=1e-3)
```

```

72 # Reconstruction + KL divergence losses summed over all elements and batch
73 def loss_function(recon_x, x, mu, logvar):
74     BCE = F.binary_cross_entropy(recon_x, x.view(-1, 784), reduction='sum')
75     KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
76
77     return BCE + KLD
78
79
80 def train(epoch):
81     model.train()
82     train_loss = 0
83     for batch_idx, (data, label) in enumerate(train_loader):
84         data = data.to(device)
85         optimizer.zero_grad()
86         recon_batch, mu, logvar, z = model(data)
87         loss = loss_function(recon_batch, data, mu, logvar)
88         loss.backward()
89         train_loss += loss.item()
90         optimizer.step()
91
92     print('====> Epoch: {} Average loss: {:.4f}'.format(
93         epoch, train_loss / len(train_loader.dataset)))
94

```

```

95 def test(epoch):
96     model.eval()
97     test_loss = 0
98     with torch.no_grad():
99         for i, (data, label) in enumerate(test_loader):
100             count = 0
101             if i == 0:
102                 for k in range(10):
103                     for j, l in enumerate(label):
104                         if k == 1:
105                             if k == 0 and count == 0:
106                                 chosen = torch.stack([data[j]], 1)
107                                 # print(k)
108                                 count += 1
109                             else:
110                                 tmp = torch.stack([data[j]], 1)
111                                 # print(2, chosen)
112                                 chosen = torch.cat([chosen, tmp])
113                                 # print(k)
114                                 count += 1
115                                 if count == 2:
116                                     count = 0
117                                     break
118
119             recon_batch, mu, logvar, z = model(chosen)
120             z_ = torch.empty(90, 20)
121             for i in range(10):
122                 z_[i*9] = z[i*9]
123                 z_[i*9+9] = z[i*9+1]
124                 tmp = torch.add(-z[i*9], z[i*9+1])
125                 for j in range(1, 9):
126                     z_[i * 9 + j] = z_[i*9] + j/8 * tmp
127
128             recon = model.decode(z_)
129             save_image(recon.view(90, 1, 28, 28).cpu(), 'results/test.png', nrow=9)
130

```

```

131 z_d = torch.empty(90, 20)
132 for i in range(9):
133     z_d[i*9] = z[i*9+1]
134     z_d[i*9+1] = z[i*9+2]
135     tmp = torch.add(-z[i*9+1], z[i*9+2])
136     for j in range(1, 9):
137         z_d[i * 9 + j] = z_d[i*9] + j/8 * tmp
138
139 z_d[0] = z[-1]
140 z_d[9] = z[0]
141 tmp = torch.add(-z[-1], z[0])
142 for j in range(1, 9):
143     z_d[0] + j] = z_d[0] + j / 8 * tmp
144
145 recon_d = model.decode(z_d)
146 save_image(recon_d.view(90, 1, 28, 28).cpu(), 'results/test_d.png', nrow=9)
147
148 test_loss /= len(test_loader.dataset)
149 print('====> Test set loss: {:.4f}'.format(test_loss))
150

```

```

151 if __name__ == '__main__':
152     for epoch in range(1, args.epochs + 1):
153         train(epoch)
154         test(0)
155

```