CS498 AML,AMO HW9

Shrashti Singhal, Ankush Singhal

TOTAL POINTS

130 / 130

QUESTION 1

1 Same Digits 45 / 45

✓ - **0 pts** **Correct**

- **45 pts** Wrong.
- **20 pts** Plainly interpolated in image space.
- **20 pts** Attempted but failed to train

QUESTION 2

2 Different Digits 45 / 45

✓ - **0 pts** **Correct**

- **45 pts** Wrong
- **20 pts** Plainly interpolated in image space
- **20 pts** Attempted but failed to train
- **20 pts** Wrong interpolation illustrated

QUESTION 3

3 Code 10 / 10

✓ - **0 pts** **Correct**

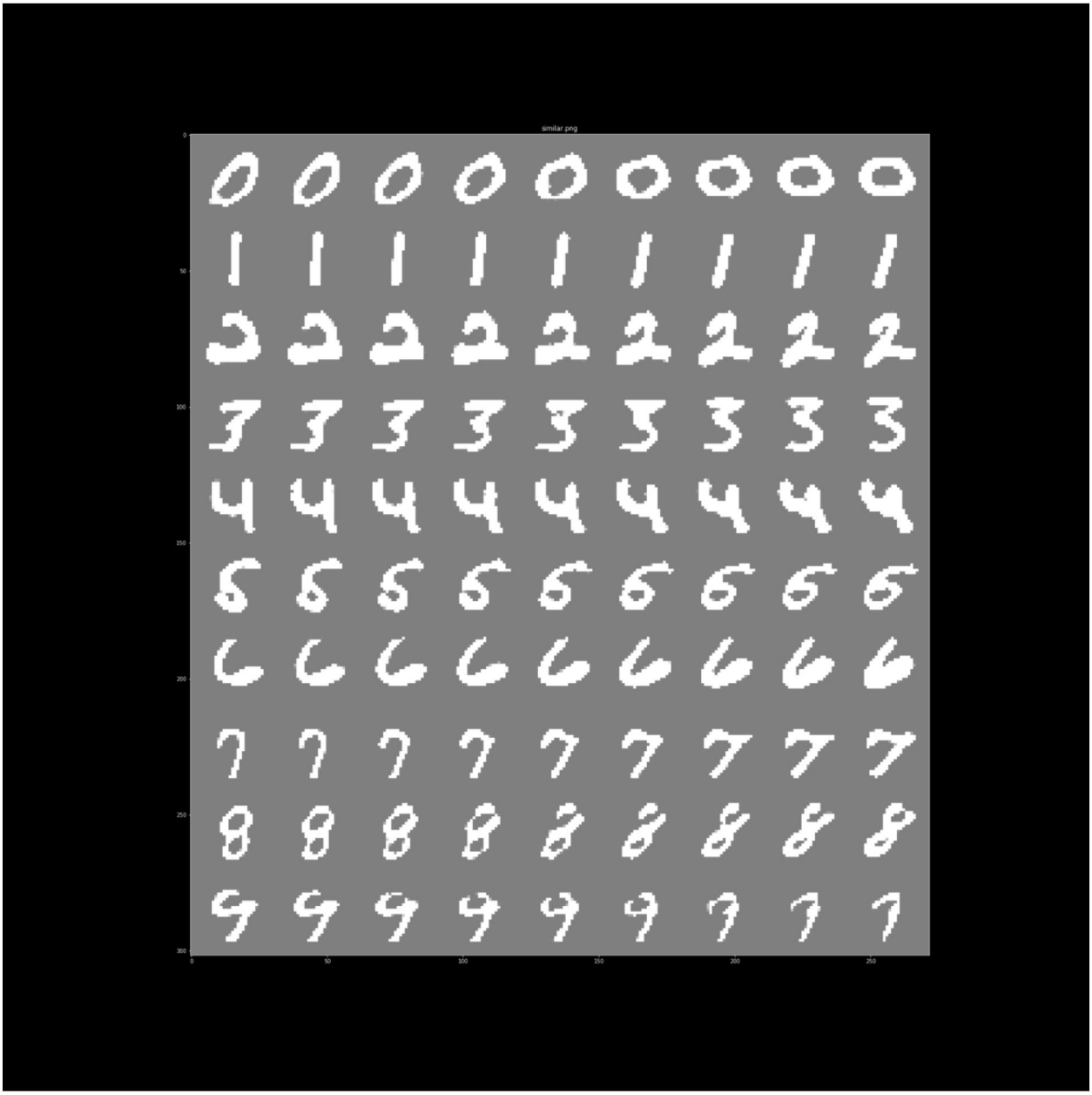
QUESTION 4

4 Late Penalty 30 / 30

✓ - **0 pts** **not late**

- **5 pts** 1 day late
- **10 pts** 2 days late
- **15 pts** 3 days late
- **20 pts** 4 days late
- **25 pts** 5 days late
- **30 pts** 6+ days late

1) **Same digit interpolates**



1 Same Digits **45 / 45**

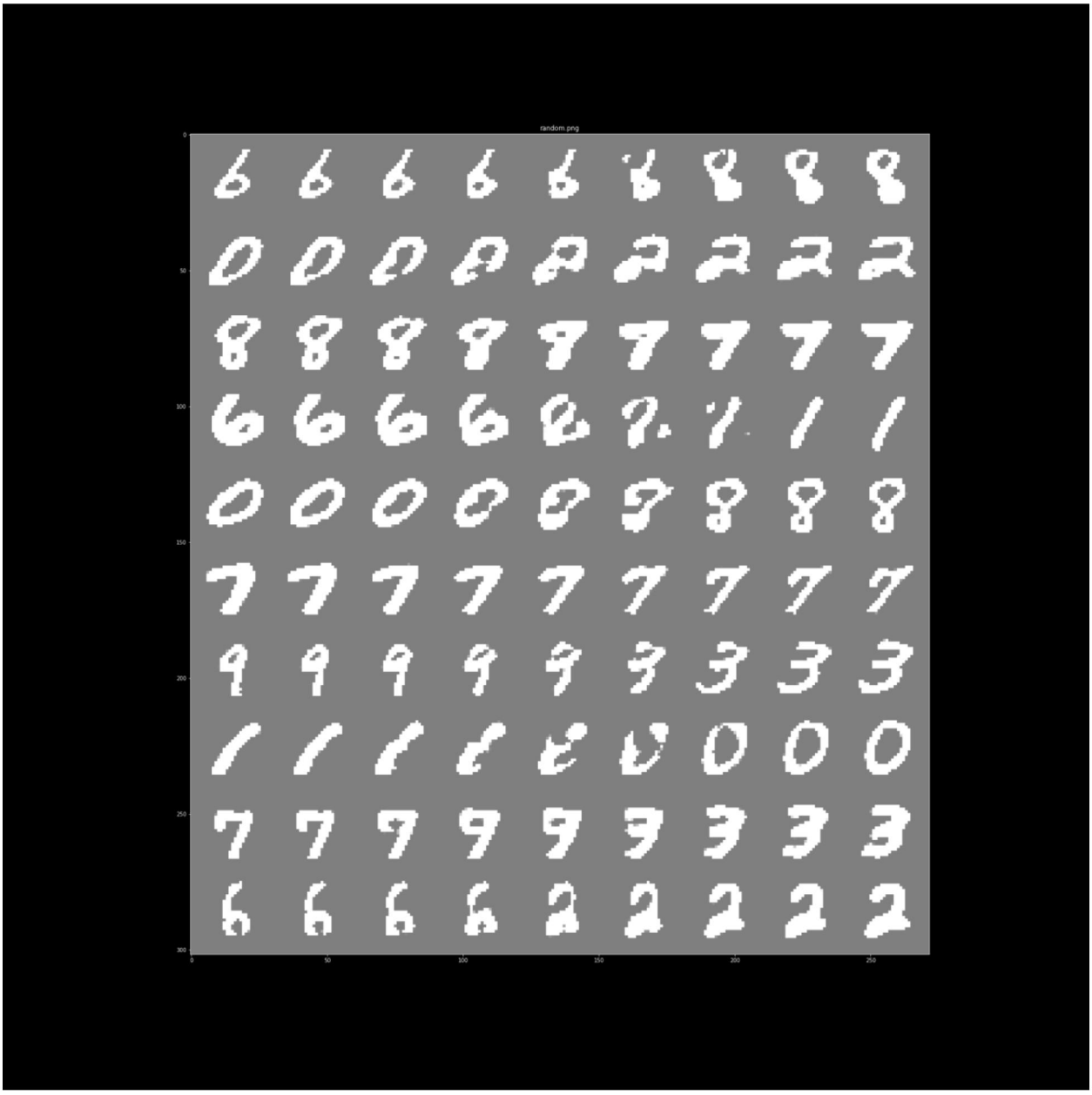
✓ **- 0 pts Correct**

- 45 pts Wrong.

- 20 pts Plainly interpolated in image space.

- 20 pts Attempted but failed to train

2) Different digit interpolates



2 Different Digits 45 / 45

✓ **- 0 pts** **Correct**

- 45 pts Wrong

- 20 pts Plainly interpolated in image space

- 20 pts Attempted but failed to train

- 20 pts Wrong interpolation illustrated

3) Code

```
import torch
%matplotlib inline
import matplotlib.pyplot as plt
from torch import nn, optim
from torch.autograd import Variable
from torch.nn import functional as F
from torchvision.utils import save_image
import torch.utils.data
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
from torchvision import utils
import numpy as np

torch.rand(5, 3)

!mkdir hw9_data

from torchvision import datasets, transforms

## YOUR CODE HERE ##
transformations = transforms.Compose([transforms.ToTensor(),transforms.Normalize((0.1307,),
(0.3081,))])
mnist_train = datasets.MNIST('./hw9_data', download=True, train=True, transform=transformations)
mnist_test = datasets.MNIST('./hw9_data', download=True, train=False, transform=transformations)

## YOUR CODE HERE ##
```

```
train_loader = DataLoader(mnist_train, batch_size=32, shuffle=True, num_workers=4)

test_loader = DataLoader(mnist_test, batch_size=32, shuffle=True, num_workers=4)



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

        # ENCODER
        self.fc1 = nn.Linear(784, 400)
        self.relu = nn.ReLU()
        self.fc21 = nn.Linear(400, 20)  # mu layer
        self.fc22 = nn.Linear(400, 20)  # logvariance layer

        # DECODER
        self.fc3 = nn.Linear(20, 400)
        self.fc4 = nn.Linear(400, 784)
        self.sigmoid = nn.Sigmoid()

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

    def reparameterize(self, mu: Variable, logvar: Variable) -> Variable:
        if self.training:
            std = logvar.mul(0.5).exp_()
            eps = Variable(std.data.new(std.size()).normal_())
            return eps.mul(std).add_(mu)
```

```
        else:
            return mu

    def decode(self, z: Variable) -> Variable:
        h3 = self.relu(self.fc3(z))
        return self.sigmoid(self.fc4(h3))

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

model = VAE()
optimizer = optim.Adam(model.parameters(), lr=1e-3)
BATCH_SIZE = 32
LOG_INTERVAL = 100
EPOCHS = 10
SEED = 1

def loss_function(recon_x, x, mu, logvar) -> Variable:
    BCE = F.binary_cross_entropy(recon_x, x.view(-1, 784))
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    KLD /= BATCH_SIZE * 784
    return BCE + KLD

def train(epoch):
    # toggle model to train mode
    model.train()
```

```
    train_loss = 0
    for batch_idx, (data, _) in enumerate(train_loader):
        data = Variable(data)
        optimizer.zero_grad()
        recon_batch, mu, logvar = model(data)
        loss = loss_function(recon_batch, data, mu, logvar)
        loss.backward()
        train_loss += loss.item()
        optimizer.step()
        if batch_idx % LOG_INTERVAL == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader),
                loss.item() / len(data)))

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


def test(epoch):
            model.eval()
            test_loss = 0
            for i, (data, _) in enumerate(test_loader):
                        data = Variable(data, volatile=True)
                        recon_batch, mu, logvar = model(data)
                        test_loss += loss_function(recon_batch, data, mu, logvar).item()
                        if i == 0:
                                    n = min(data.size(0), 8)
                                    comparison = torch.cat([data[:n], recon_batch.view(BATCH_SIZE, 1, 28, 28)[:n]])
                                    save_image(comparison.data.cpu(), 'mnist/reconstruction_' + str(epoch) +
'.png', nrow=n)
```

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


for epoch in range(1, EPOCHS + 1):
    train(epoch)
    test(epoch)


    sample = Variable(torch.randn(64, 20))
    sample = model.decode(sample).cpu()


    save_image(sample.data.view(64, 1, 28, 28),'results/sample_' + str(epoch) + '.png')


def create_interpolates(A, B, model):
    fig = plt.figure(figsize=(28, 28))
    recon_batch, mu_a, logvar_a = model.forward(A)
    recon_batch, mu_b, logvar_b = model.forward(B)
    sample_A = model.reparameterize(mu_a, logvar_a)
    sample_B = model.reparameterize(mu_b, logvar_b)


    flattened_images = torch.zeros((9, 20))
    images = torch.zeros((9, 28, 28))
    difference = sample_B - sample_A
    for i in range(9):
        flattened_images[i] = torch.add(sample_A.detach(), i/8, difference.detach())
        images[i] = model.decode(flattened_images[i]).view(28, 28)
    return images


def show_mnist_batch(images_batch, title):
```

```
    grid = utils.make_grid(images_batch.view(90, 1, 28, 28), nrow=9)
    utils.save_image(grid, title)
    plt.imshow(grid.detach().numpy().transpose((1, 2, 0)))
    plt.title(title)
    plt.savefig(title)


similar_pairs = {}
for _, (x, y) in enumerate(test_loader):
  for i in range(len(y)):
    if y[i].item() not in similar_pairs:
      similar_pairs[y[i].item()] = []
    if len(similar_pairs[y[i].item()])<2:
      similar_pairs[y[i].item()].append(x[i])

  done = True
  for i in range(10):
    if i not in similar_pairs or len(similar_pairs[i])<2:
      done = False

  if done:
    break

# similar_pairs[i] contains two images indexed at 0 and 1 that have images of the digit i


## YOUR CODE HERE ##
all_interpolates_list = []
for i in range(10):
    all_interpolates_list.append(create_interpolates(similar_pairs[i][0], similar_pairs[i][1], model))
```

```
show_mnist_batch(torch.cat(all_interpolates_list), "similar.png")


random_pairs = {}
for _, (x, y) in enumerate(test_loader):
  # Make sure the batch size is greater than 20
  for i in range(10):
    random_pairs[i] = []
    random_pairs[i].append(x[2*i])
    random_pairs[i].append(x[2*i+1])
  break


# random_pairs[i] contains two images indexed at 0 and 1 that are chosen at random
## YOUR CODE HERE ##
all_interpolates_list = []
for i in range(10):
    all_interpolates_list.append(create_interpolates(random_pairs[i][0], random_pairs[i][1], model))
show_mnist_batch(torch.cat(all_interpolates_list), "random.png")
```

3 Code 10 / 10

✓ - 0 pts Correct

4 Late Penalty 30 / 30

- ✓ **- 0 pts not late**
- **- 5 pts** 1 day late
- **- 10 pts** 2 days late
- **- 15 pts** 3 days late
- **- 20 pts** 4 days late
- **- 25 pts** 5 days late
- **- 30 pts** 6+ days late