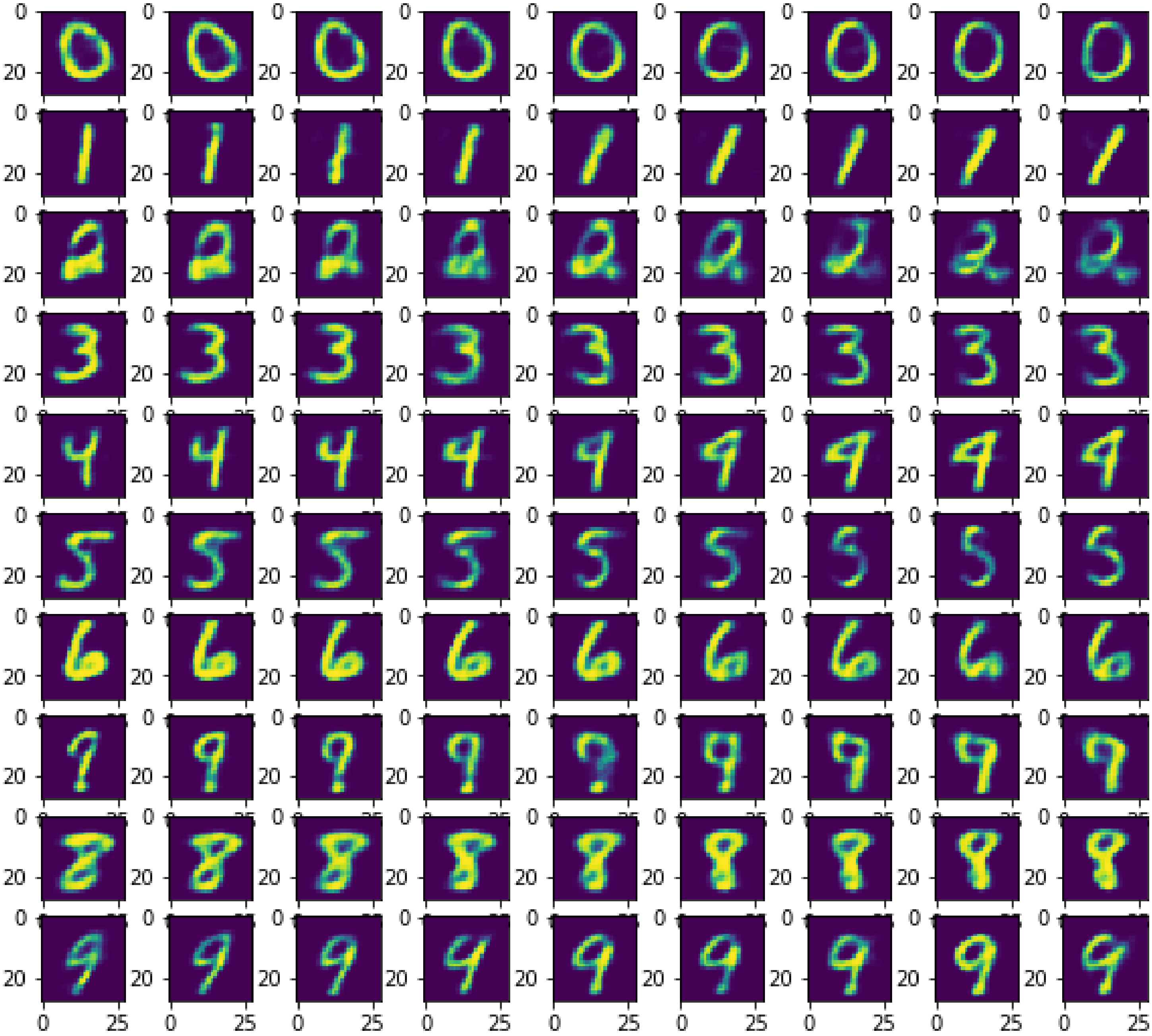
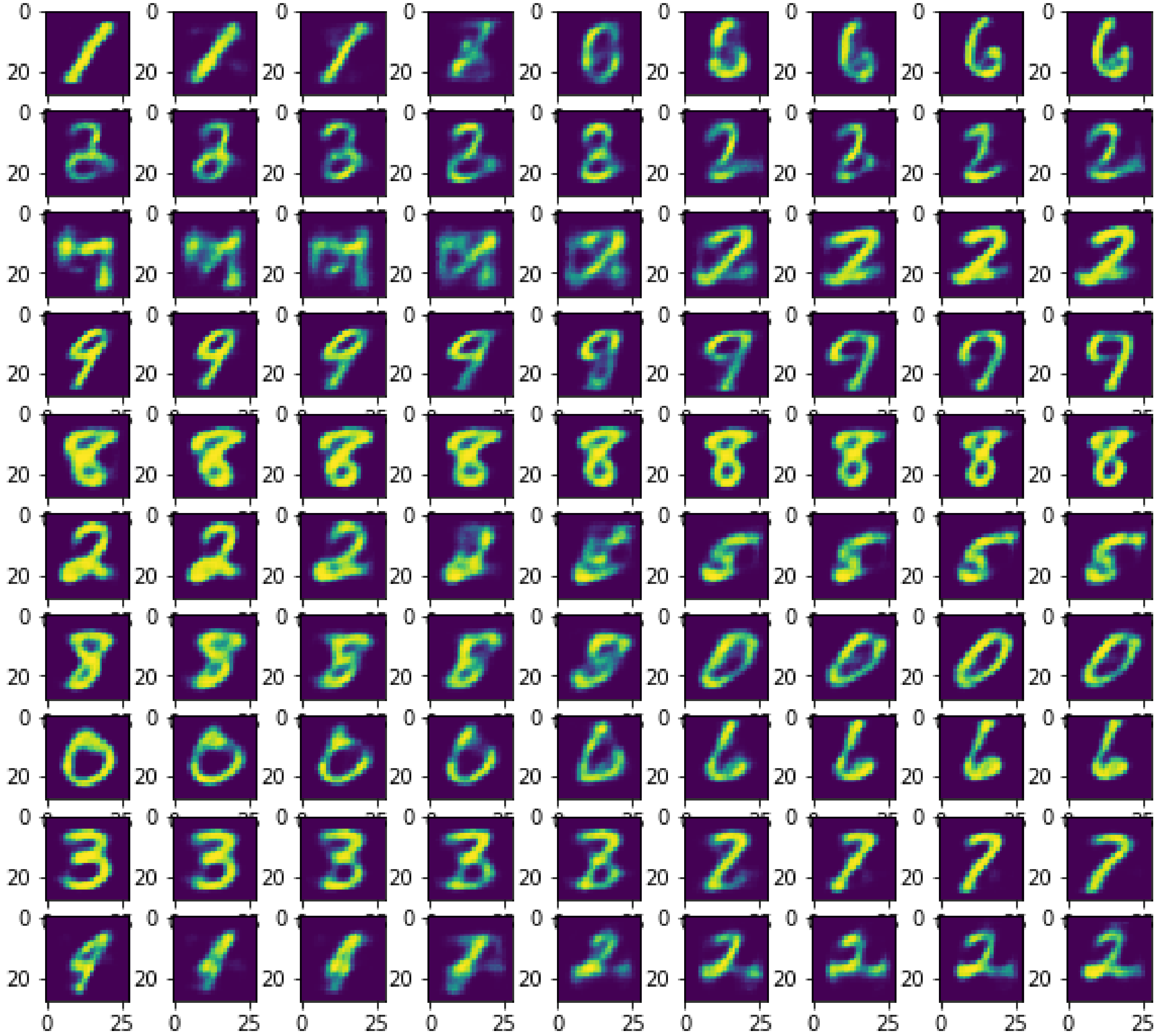
Same digit interpolates:



Different digit interpolation



Code:

```
#Part 1 – Datasets and Dataloaders
import torch
!mkdir hw9_data

from torchvision import datasets, transforms
root = 'hw9_data'
transformations = transforms.ToTensor()
mnist_train = datasets.MNIST(root, train=True, download=True, transform=transformations)
mnist_test = datasets.MNIST(root, train=False, download=True, transform=transformations)

from torch.utils.data import DataLoader

BATCH_SIZE=32
kwargs = {'num_workers':1, 'pin_memory':True}
train_loader = DataLoader(mnist_train, batch_size=BATCH_SIZE, shuffle=True, **kwargs)
test_loader = DataLoader(mnist_test, batch_size=BATCH_SIZE, shuffle=True, **kwargs)
```

```
#part 2-encoder and decoder:
from torch import nn
from torch.nn import functional as F
inputPixel = 784
outputs = 400

class Encoder(nn.Module):
  def __init__(self, latent_dim):
    super(Encoder, self).__init__()
    self.fc1 = nn.Linear(inputPixel,outputs)
    self.relu = nn.ReLU()
    self.fc21 = nn.Linear(outputs, latent_dim)
    self.fc22 = nn.Linear(outputs, latent_dim)

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

class Decoder(nn.Module):
  def __init__(self, latent_dim):
    super(Decoder, self).__init__()
    self.relu = nn.ReLU()
    self.fc3 = nn.Linear(latent_dim, outputs)
    self.fc4 = nn.Linear(outputs, inputPixel)
    self.sigmoid = nn.Sigmoid()
```

```
def forward(self,x):
  h3 = self.relu(self.fc3(x))
  return self.sigmoid(self.fc4(h3))
```

```
#part 3: TGraining and loss functions:
def sample(mu, logvar):
    std = logvar.mul(0.5).exp_()
    eps  =std.data.new(std.size()).normal_()
    return eps.mul(std).add_(mu)
def vae_loss(x, x_hat, mu, logvar):
    BCE = F.binary_cross_entropy(x, x_hat.view(-1, inputPixel))
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    KLD /= BATCH_SIZE * inputPixel
    return BCE + KLD
from torch import optim

encoder = Encoder(latent_dim = 32)
encoder.cuda()
decoder = Decoder(latent_dim = 32)
decoder.cuda()
params = list(encoder.parameters())+list(decoder.parameters())
optimizer = optim.Adam(params, lr=1e-3)
import tensorflow as tf
def train(encoder, decoder, train_loader, optimizer, num_epochs = 10):
    train_loss = 0
    for batch_idx, (data, _) in enumerate(train_loader):
        data = data.view(data.shape[0],-1)
        data = data.cuda()
        optimizer.zero_grad()
        mu, logvar = encoder.forward(data)
        z = sample(mu, logvar)
        recon_x = decoder.forward(z)

        loss = vae_loss(recon_x, data, mu, logvar)

        loss.backward()

        train_loss += loss

        optimizer.step()

        if batch_idx % 10 == 0:

            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(

                num_epochs, batch_idx * len(data), len(train_loader.dataset),

                100. * batch_idx / len(train_loader),

                loss / len(data)))

        print('====> Epoch: {} Average loss: {:.4f}'.format(

              num_epochs, train_loss / len(train_loader.dataset)))

train(encoder, decoder, train_loader, optimizer, num_epochs = 10)
```

```
#part 4 Visulaizing the VAE output:
import matplotlib.pyplot as plt
from torchvision import utils
import numpy as np

def create_interpolates(A, B, encoder, decoder):
    muA, logvarA = encoder.forward(A)
    zA = sample(muA, logvarA)
    recon_a = decoder.forward(zA)
    muB, logvarB = encoder.forward(B)
    zB = sample(muB, logvarB)
    recon_b = decoder.forward(zB)
    result = [recon_a.detach().cpu().reshape([28,28])]
    muDiff = (muB-muA)/7
    muTemp = muA
    for i in range(7):
        muTemp += muDiff
        zTemp = sample(muTemp, logvarA)
        result.append(decoder.forward(zTemp).detach().cpu().reshape([28,28]))
    result.append(recon_b.detach().cpu().reshape([28,28]))
    return result
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
allP = []
for j in range(10):
    A, B = similar_pairs[j][0], similar_pairs[j][1]
    A = A.view(A.shape[0],-1)
    B = B.view(B.shape[0],-1)
    A = A.cuda()
    B = B.cuda()
    allP.append(create_interpolates(A, B, encoder, decoder))

fig = plt.figure(figsize=(10,9))
for i in range(90):
    fig.add_subplot(10,9,i+1)
    plt.imshow(allP[i//9][i%9])
fig.show()
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
allP = []
for j in range(10):
    A, B = random_pairs[j][0], random_pairs[j][1]
    A = A.view(A.shape[0],-1)
    B = B.view(B.shape[0],-1)
    A = A.cuda()
    B = B.cuda()
    allP.append(create_interpolates(A, B, encoder, decoder))

fig = plt.figure(figsize=(10,9))
for i in range(90):
    fig.add_subplot(10,9,i+1)
    plt.imshow(allP[i//9][i%9])
fig.show()
```