Class6_VAE

February 6, 2025

[1]:
```
import torch
from torch import nn
from torch import optim
from torch.utils.data import Dataset, DataLoader

import matplotlib.pyplot as plt

import torchvision
from torchvision import datasets
from torchvision import transforms
```

[2]:
```
device = "cuda" if torch.cuda.is_available() else "cpu"
print(device)
```

```
cuda
```

[17]:
```
class VAE(nn.Module):
    def __init__(self, input_dim, latent_dim):
        super(VAE, self).__init__()
        # Encoder
        self.en_ln1 = nn.Linear(input_dim, 400)
        self.en_mu = nn.Linear(400, latent_dim)
        self.en_logvar = nn.Linear(400, latent_dim)

        # Decoder
        self.de_ln1 = nn.Linear(latent_dim, 400)
        self.de_ln2 = nn.Linear(400, input_dim)

        # Activation
        self.relu = nn.ReLU()

    def encode(self, x):
        x = self.relu(self.en_ln1(x))
        mu = self.en_mu(x)
        logvar = self.en_logvar(x) # Keep its positive
        return mu, logvar

    def reparameterize(self, mu, logvar):
```

```
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)
        return mu + std * eps

    def decode(self, z):
        z = self.relu(self.de_ln1(z))
        z = torch.sigmoid(self.de_ln2(z))
        return z

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

def custom_loss(syn_x, og_x, mu, logvar):
    # BCE = torch.nn.functional.binary_cross_entropy(syn_x, og_x,␣
 ↪reduction="mean")
    MSE = torch.nn.functional.mse_loss(syn_x, og_x)
    logvar = torch.clamp(logvar, min=-10)
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp()) / (28*28*512)
    return MSE + KLD
```

[10]:
```
training_dataset = datasets.MNIST(
    r"./dataset", train=True, download=False,
    transform=transforms.Compose([transforms.ToTensor()])
    )
# testing_dataset = datasets.MNIST(r"./dataset", train=False, download=True,␣
 ↪transform=transforms.ToTensor())
```

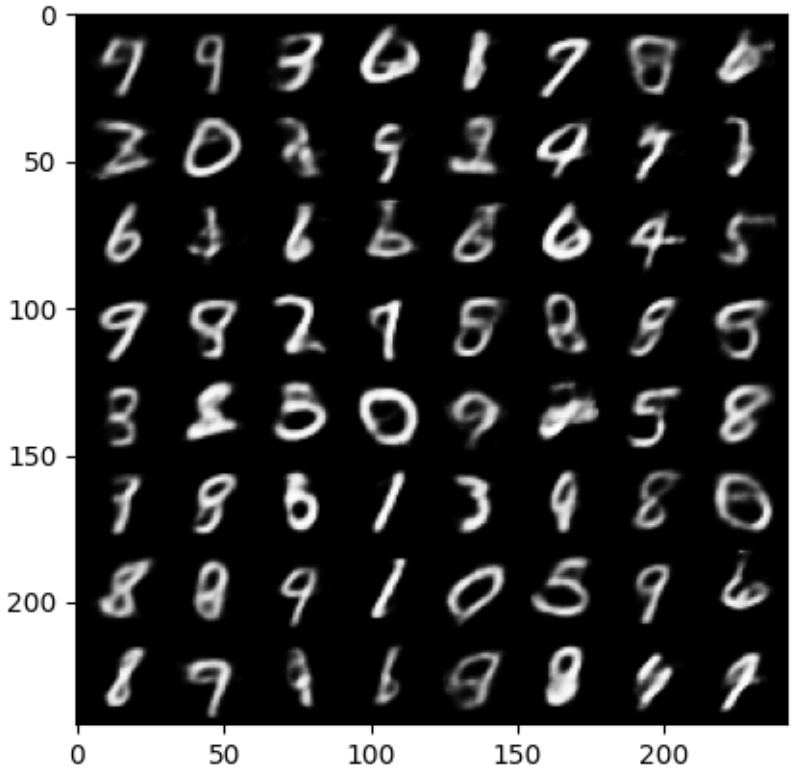
[11]:
```
class ImageDataset(Dataset):
    def __init__(self, input_dataset):
        super().__init__()
        self.input_dataset = input_dataset.to(torch.float)

    def __len__(self):
        return len(self.input_dataset)

    def __getitem__(self, idx):
        return self.input_dataset[idx]

training_loader = DataLoader(ImageDataset(training_dataset.data),␣
 ↪batch_size=512, shuffle=True)
```

[21]:
```
input_dim = len(torch.flatten(training_dataset.data[0]))
vae_model = VAE(input_dim, 20).to(device)
optimizer = optim.Adam(vae_model.parameters(), lr=1e-3)
```

[22]:
```
vae_model.train()

for ep in range(1, 101):
    n = 0
    last_mean_loss = 0
    for im_batch in training_loader:
        # origin_input = torch.flatten(im_batch, start_dim=1)
        origin_input = im_batch.view(-1, input_dim).to(device) / 255
        optimizer.zero_grad()
        outputs, mu, logvar = vae_model(origin_input)
        loss = custom_loss(outputs, origin_input, mu, logvar)
        loss.backward()
        optimizer.step()
        last_mean_loss += loss.item()
        n += 1

    if ep%10 == 0:
        print("lasted loss:", last_mean_loss/n)
```

```
lasted loss: 0.04071540739071571
lasted loss: 0.03894379256703591
lasted loss: 0.03816944851650525
lasted loss: 0.03767615130518453
lasted loss: 0.03733125622621027
lasted loss: 0.03709859325219009
lasted loss: 0.036844006992118844
lasted loss: 0.03662660320178937
lasted loss: 0.036495851431736503
lasted loss: 0.03636061114464271
```

[24]:
```
vae_model.eval()
with torch.no_grad():
    # Sample random points in latent space
    z = torch.randn(64, 20).to(device)
    sample = vae_model.decode(z).cpu()

    # Visualize some generated images
    sample = sample.view(64, 1, 28, 28)
    grid_img = torchvision.utils.make_grid(sample, nrow=8)
    plt.imshow(grid_img.permute(1, 2, 0))
    plt.show()
```


0
50
100
150
200
0
50
100
150
200

[ ]: