In [1]:
```
import torch
import numpy as np
import torch.nn as nn
import matplotlib.pyplot as plt
from torchvision.datasets import MNIST
from torch.utils.data import DataLoader
import torchvision.transforms as transforms
from mpl_toolkits.axes_grid1 import ImageGrid
from torchvision.utils import save_image, make_grid
```

In [2]:
```
# create a transofrm to apply to each datapoint
transform = transforms.Compose([transforms.ToTensor()])

# download the MNIST datasets
path = '~/datasets'
train_dataset = MNIST(path, transform=transform, download=True)
test_dataset  = MNIST(path, transform=transform, download=True)

# create train and test dataloaders
batch_size = 100
train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=batch_size, shuffle=False)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

In [3]:
```
class VAE(nn.Module):

    def __init__(self, input_dim=784, hidden_dim=400, latent_dim=200, device=device):
        super(VAE, self).__init__()

        # encoder
        self.encoder = nn.Sequential(
            nn.Linear(input_dim, hidden_dim),
            nn.LeakyReLU(0.2),
            nn.Linear(hidden_dim, latent_dim),
            nn.LeakyReLU(0.2)
            )

        # latent mean and variance
        self.mean_layer = nn.Linear(latent_dim, 2)
        self.logvar_layer = nn.Linear(latent_dim, 2)

        # decoder
        self.decoder = nn.Sequential(
            nn.Linear(2, latent_dim),
            nn.LeakyReLU(0.2),
            nn.Linear(latent_dim, hidden_dim),
            nn.LeakyReLU(0.2),
            nn.Linear(hidden_dim, input_dim)
            )

    def encode(self, x):
        x = self.encoder(x)
        mean, logvar = self.mean_layer(x), self.logvar_layer(x)
        return mean, logvar

    def reparameterization(self, mean, logvar):
        epsilon = torch.randn_like(logvar).to(device)
        z = mean + (logvar/2).exp()*epsilon
        return z

    def decode(self, x):
        return self.decoder(x)

    def forward(self, x):
        mean, logvar = self.encode(x)
        z = self.reparameterization(mean, logvar)
        x_hat = self.decode(z)
        return x_hat, mean, logvar
```

In [13]:
```
model = VAE().to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

In [14]:
```
def loss_function(x, x_hat, mean, log_var):
    reproduction_loss = nn.functional.mse_loss(x_hat, x, reduction='sum')
    KLD = - 0.5 * torch.sum(1+ log_var - mean.pow(2) - log_var.exp())
```

```
    return reproduction_loss + KLD
```

In [15]:
```
def train(model, optimizer, epochs, device):
    model.train()
    for epoch in range(epochs):
        overall_loss = 0
        for batch_idx, (x, _) in enumerate(train_loader):
            x = x.view(batch_size, -1).to(device)

            optimizer.zero_grad()

            x_hat, mean, log_var = model(x)
            loss = loss_function(x, x_hat, mean, log_var)

            overall_loss += loss.item()

            loss.backward()
            optimizer.step()

        print("\tEpoch", epoch + 1, "\tAverage Loss: ", overall_loss/(batch_idx*batch_size))
    return overall_loss

train(model, optimizer, epochs=50, device=device)
```

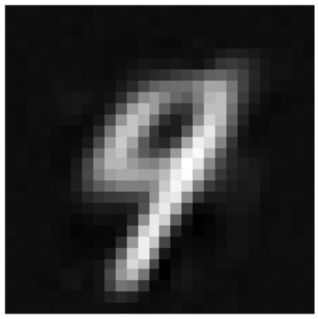
```
        Epoch 1         Average Loss:  44.90734841679492
        Epoch 2         Average Loss:  40.757268009345005
        Epoch 3         Average Loss:  39.486435697679724
        Epoch 4         Average Loss:  38.7257742233149
        Epoch 5         Average Loss:  38.27964989663762
        Epoch 6         Average Loss:  37.90731650325412
        Epoch 7         Average Loss:  37.615150050702994
        Epoch 8         Average Loss:  37.331490452022905
        Epoch 9         Average Loss:  37.19050968329377
        Epoch 10        Average Loss:  37.024223127412874
        Epoch 11        Average Loss:  36.90037159914962
        Epoch 12        Average Loss:  36.727448339191625
        Epoch 13        Average Loss:  36.58070171069621
        Epoch 14        Average Loss:  36.553968732066465
        Epoch 15        Average Loss:  36.421423943062656
        Epoch 16        Average Loss:  36.34540449903485
        Epoch 17        Average Loss:  36.21433060227332
        Epoch 18        Average Loss:  36.1550144079651
        Epoch 19        Average Loss:  36.08352418011138
        Epoch 20        Average Loss:  35.9948438722741
        Epoch 21        Average Loss:  35.93922505119209
        Epoch 22        Average Loss:  35.8742404210468
        Epoch 23        Average Loss:  35.848164046196786
        Epoch 24        Average Loss:  35.71230819269094
        Epoch 25        Average Loss:  35.709727039369
        Epoch 26        Average Loss:  35.68782412826717
        Epoch 27        Average Loss:  35.61055909425866
        Epoch 28        Average Loss:  35.57203330828073
        Epoch 29        Average Loss:  35.51085223826821
        Epoch 30        Average Loss:  35.470594671946735
        Epoch 31        Average Loss:  35.42248072960142
        Epoch 32        Average Loss:  35.40860261487244
        Epoch 33        Average Loss:  35.31653475600611
        Epoch 34        Average Loss:  35.29057205938935
        Epoch 35        Average Loss:  35.28282936975037
        Epoch 36        Average Loss:  35.24929938569491
        Epoch 37        Average Loss:  35.216343539688545
        Epoch 38        Average Loss:  35.16881060719689
        Epoch 39        Average Loss:  35.121357393344375
        Epoch 40        Average Loss:  35.10692839302483
        Epoch 41        Average Loss:  35.096852032847714
        Epoch 42        Average Loss:  35.08559472493218
        Epoch 43        Average Loss:  34.998794373271856
        Epoch 44        Average Loss:  34.98728417398137
        Epoch 45        Average Loss:  35.02352690274807
        Epoch 46        Average Loss:  34.88520460940761
        Epoch 47        Average Loss:  34.962183530167465
        Epoch 48        Average Loss:  34.86952581695404
        Epoch 49        Average Loss:  34.86015463190604
        Epoch 50        Average Loss:  34.8394119186274
2086880.7739257812
```

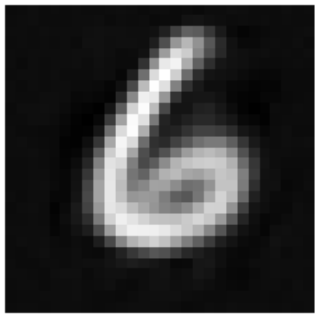
Out[15]:

In [16]:
```
def generate_digit(z1, z2):
    z_sample = torch.tensor([[z1, z2]], dtype=torch.float).to(device)
```

```
    x_decoded = model.decode(z_sample)
    digit = x_decoded.detach().cpu().reshape(28, 28) # reshape vector to 2d array
    plt.figure(figsize = (2.5,2.5))
    plt.imshow(digit, cmap='gray')
    plt.axis('off')
    plt.show()

generate_digit(0.8, -0.9), generate_digit(-0.2, 1.0)
```





Out[16]: (None, None)

In [ ]: