Problem 1.2

In [87]:
```
%matplotlib inline

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim
import torch.optim as optim

import torchvision
import torchvision.transforms as transforms
from torchvision.utils import make_grid

import matplotlib.pyplot as plt
import numpy as np
```

In [88]:
```
# Load data
batch_size = 128
train_set, test_set, train_loader, test_loader = {},{},{},{}
transform = transforms.Compose(
    [transforms.ToTensor()])

train_set = torchvision.datasets.FashionMNIST(root='../data', train=True, download=Tru
test_set = torchvision.datasets.FashionMNIST(root='../data', train=False, download=Tru

train_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_size, shuffle=T
test_loader = torch.utils.data.DataLoader(test_set, batch_size=batch_size, shuffle=Fal


device = 'cpu'
```

In [89]:
```
# set up model architecture
class RestrBoltzMachiNet(nn.Module):
    """Restricted Boltzmann Machine for generating MNIST images."""

    def __init__(self, D: int, F: int, k: int):
        super(RestrBoltzMachiNet, self).__init__()
        self.m = F
        self.W = nn.Parameter(torch.randn(F, D)* 1e-2) # Initialized from Normal(mean=
        self.c = nn.Parameter(torch.zeros(D)) # Initialized as 0.0
        self.b = nn.Parameter(torch.zeros(F)) # Initilaized as 0.0
        self.k = k


    def sample_x(self, mu, sigma):
        """Sample from a normal distribution defined by a given parameter.

           Args:
                mu: Mean of the normal distribution.
                sigma: Standard deviation of the normal distribution.

           Returns:
               bern_sample: Sample from Bernoulli(p)
        """
```

```
        norm_sample = torch.normal(mu, sigma)
        return norm_sample

    def sample_h(self, p):
        """Sample from a bernoulli distribution defined by a given parameter.

           Args:
                p: Parameter of the bernoulli distribution.

           Returns:
               bern_sample: Sample from Bernoulli(p)
        """

        bern_sample = p.bernoulli()
        return bern_sample


    def P_h_x(self, x):
        """Returns the conditional P(h|x). (Slide 9, Lecture 14)

        Args:
            x: The parameter of the conditional h|x.

        Returns:
            ph_x: probability of hidden vector being element-wise 1.
        """

        ph_x = torch.sigmoid(F.linear(x, self.W, self.b)) # n_batch x F
        return ph_x


    def P_x_h(self, h):
        """Returns the conditional P(x|h). (Slide 9, Lecture 14)

        Args:
            h: The parameter of the conditional x|h.

        Returns:
            px_h: probability of visible vector being element-wise 1.
        """

        px_h_mu = torch.sigmoid(F.linear(h, self.W.t(), self.c)) # n_batch x D
        px_h_sigma = torch.sigmoid(F.linear(h, self.W.t(), self.c)) # n_batch x D
        return px_h_mu, px_h_sigma


    def free_energy(self, x):
        """Returns the Average F(x) free energy. (Slide 11, Lecture 14)."""

        vbias_term = x.mv(self.c) # n_batch x 1
        wv_b = F.linear(x, self.W, self.b) # n_batch x F
        hidden_term = F.softplus(wv_b).sum(dim=1) # n_batch x 1
        return (-hidden_term - vbias_term).mean() # 1 x 1


    def forward(self, x):
        """Generates x_negative using MCMC Gibbs sampling starting from x."""

        x_negative = x
```

```
        for _ in range(self.k):

            ## Step 1: Sample h from previous iteration.
            # Get the conditional prob h|x
            phx_k = self.P_h_x(x_negative)
            # Sample from h|x
            h_negative = self.sample_h(phx_k)

            ## Step 2: Sample x using h from step 1.
            # Get the conditional proba x|h
            pxh_k_mu, pxh_k_sigma = self.P_x_h(h_negative)
            # Sample from x|h
            x_negative = self.sample_x(pxh_k_mu, pxh_k_sigma)

        return x_negative, pxh_k_mu, pxh_k_sigma
```

In [99]:

```
# build training loop
def train(model, device, train_loader, optimizer, epoch):

    train_loss = 0
    model.train()
    m = model.m

    for batch_idx, (data, target) in enumerate(train_loader):

        # torchvision provides us with normalized data, s.t. input is in [0,1]
        data = data.view(data.size(0),-1) # flatten the array: Converts n_batchx1x28x2
        data = data.bernoulli()
        data = data.to(device)

        optimizer.zero_grad()

        x_tilde, _, _ = model(data)
        x_tilde = x_tilde.detach()

        loss = model.free_energy(data) - model.free_energy(x_tilde)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()

        if (batch_idx+1) % (len(train_loader)//4) == 0:
            print(f"M={m}; Epoch={epoch}, {round(100. * batch_idx / len(train_loader))
    print("\n")

def test(model, device, test_loader):

    model.eval()
    criterion = nn.MSELoss()
    test_loss = 0

    with torch.no_grad():
        for data, target in test_loader:
            data = data.view(data.size(0),-1)
            data = data.bernoulli()
            data = data.to(device)
            xh_k,_,_ = model(data)
            loss = criterion(data, xh_k)
            test_loss += loss.item() # sum up batch loss
```

```
    test_loss = (test_loss*batch_size)/len(test_loader.dataset)
    print(f"M={model.m}; Test MSE = {test_loss}")

# set up optimizer/scheduler
def make_optimizer(optimizer_name, model, **kwargs):
    if optimizer_name=='Adam':
        optimizer = optim.Adam(model.parameters(),lr=kwargs['lr'])
    elif optimizer_name=='SGD':
        optimizer = optim.SGD(model.parameters(),lr=kwargs['lr'],momentum=kwargs.get('
                              weight_decay=kwargs.get('weight_decay', 0.))
    else:
        raise ValueError('Not valid optimizer name')
    return optimizer

def make_scheduler(scheduler_name, optimizer, **kwargs):
    if scheduler_name=='MultiStepLR':
        scheduler = optim.lr_scheduler.MultiStepLR(optimizer,milestones=kwargs['milest
    else:
        raise ValueError('Not valid scheduler name')
    return scheduler
```

In [100…

```
# training setup

optimizer_name = 'Adam'
scheduler_name = 'MultiStepLR'
num_epochs = 25
lr = 0.001
```

In [101…

```
# run training
device = torch.device(device)

rbms = [RestrBoltzMachiNet(D=28*28, F=10, k=10).to(device),
       RestrBoltzMachiNet(D=28*28, F=50, k=10).to(device),
       RestrBoltzMachiNet(D=28*28, F=100, k=10).to(device),
       RestrBoltzMachiNet(D=28*28, F=250, k=10).to(device)]

i = 0
for rbm in rbms:
    optimizer = make_optimizer(optimizer_name, rbm, lr=lr)
    scheduler = make_scheduler(scheduler_name, optimizer, milestones=[5], factor=0.1)

    print("Beginning training...")
    for epoch in range(1, num_epochs + 1):
        print(f"Epoch {epoch} of {num_epochs}")
        train(rbm, device, train_loader, optimizer, epoch)
        scheduler.step()

    rbms[i] = rbm
    i += 1
```

```
Beginning training...
Epoch 1 of 25
M=10; Epoch=1, 100% complete

Epoch 2 of 25
M=10; Epoch=2, 100% complete

Epoch 3 of 25
M=10; Epoch=3, 100% complete

Epoch 4 of 25
M=10; Epoch=4, 100% complete

Epoch 5 of 25
M=10; Epoch=5, 100% complete

Epoch 6 of 25
M=10; Epoch=6, 100% complete

Epoch 7 of 25
M=10; Epoch=7, 100% complete

Epoch 8 of 25
M=10; Epoch=8, 100% complete

Epoch 9 of 25
M=10; Epoch=9, 100% complete

Epoch 10 of 25
M=10; Epoch=10, 100% complete

Epoch 11 of 25
M=10; Epoch=11, 100% complete

Epoch 12 of 25
M=10; Epoch=12, 100% complete

Epoch 13 of 25
M=10; Epoch=13, 100% complete

Epoch 14 of 25
M=10; Epoch=14, 100% complete

Epoch 15 of 25
M=10; Epoch=15, 100% complete

Epoch 16 of 25
M=10; Epoch=16, 100% complete

Epoch 17 of 25
M=10; Epoch=17, 100% complete

Epoch 18 of 25
M=10; Epoch=18, 100% complete

Epoch 19 of 25
M=10; Epoch=19, 100% complete

Epoch 20 of 25
M=10; Epoch=20, 100% complete
```

```
Epoch 21 of 25
M=10; Epoch=21, 100% complete

Epoch 22 of 25
M=10; Epoch=22, 100% complete

Epoch 23 of 25
M=10; Epoch=23, 100% complete

Epoch 24 of 25
M=10; Epoch=24, 100% complete

Epoch 25 of 25
M=10; Epoch=25, 100% complete

Beginning training...
Epoch 1 of 25
M=50; Epoch=1, 100% complete

Epoch 2 of 25
M=50; Epoch=2, 100% complete

Epoch 3 of 25
M=50; Epoch=3, 100% complete

Epoch 4 of 25
M=50; Epoch=4, 100% complete

Epoch 5 of 25
M=50; Epoch=5, 100% complete

Epoch 6 of 25
M=50; Epoch=6, 100% complete

Epoch 7 of 25
M=50; Epoch=7, 100% complete

Epoch 8 of 25
M=50; Epoch=8, 100% complete

Epoch 9 of 25
M=50; Epoch=9, 100% complete

Epoch 10 of 25
M=50; Epoch=10, 100% complete

Epoch 11 of 25
M=50; Epoch=11, 100% complete

Epoch 12 of 25
M=50; Epoch=12, 100% complete

Epoch 13 of 25
M=50; Epoch=13, 100% complete

Epoch 14 of 25
M=50; Epoch=14, 100% complete

Epoch 15 of 25
```

```
M=50; Epoch=15, 100% complete

Epoch 16 of 25
M=50; Epoch=16, 100% complete

Epoch 17 of 25
M=50; Epoch=17, 100% complete

Epoch 18 of 25
M=50; Epoch=18, 100% complete

Epoch 19 of 25
M=50; Epoch=19, 100% complete

Epoch 20 of 25
M=50; Epoch=20, 100% complete

Epoch 21 of 25
M=50; Epoch=21, 100% complete

Epoch 22 of 25
M=50; Epoch=22, 100% complete

Epoch 23 of 25
M=50; Epoch=23, 100% complete

Epoch 24 of 25
M=50; Epoch=24, 100% complete

Epoch 25 of 25
M=50; Epoch=25, 100% complete

Beginning training...
Epoch 1 of 25
M=100; Epoch=1, 100% complete

Epoch 2 of 25
M=100; Epoch=2, 100% complete

Epoch 3 of 25
M=100; Epoch=3, 100% complete

Epoch 4 of 25
M=100; Epoch=4, 100% complete

Epoch 5 of 25
M=100; Epoch=5, 100% complete

Epoch 6 of 25
M=100; Epoch=6, 100% complete

Epoch 7 of 25
M=100; Epoch=7, 100% complete

Epoch 8 of 25
M=100; Epoch=8, 100% complete

Epoch 9 of 25
M=100; Epoch=9, 100% complete
```

```
Epoch 10 of 25
M=100; Epoch=10, 100% complete

Epoch 11 of 25
M=100; Epoch=11, 100% complete

Epoch 12 of 25
M=100; Epoch=12, 100% complete

Epoch 13 of 25
M=100; Epoch=13, 100% complete

Epoch 14 of 25
M=100; Epoch=14, 100% complete

Epoch 15 of 25
M=100; Epoch=15, 100% complete

Epoch 16 of 25
M=100; Epoch=16, 100% complete

Epoch 17 of 25
M=100; Epoch=17, 100% complete

Epoch 18 of 25
M=100; Epoch=18, 100% complete

Epoch 19 of 25
M=100; Epoch=19, 100% complete

Epoch 20 of 25
M=100; Epoch=20, 100% complete

Epoch 21 of 25
M=100; Epoch=21, 100% complete

Epoch 22 of 25
M=100; Epoch=22, 100% complete

Epoch 23 of 25
M=100; Epoch=23, 100% complete

Epoch 24 of 25
M=100; Epoch=24, 100% complete

Epoch 25 of 25
M=100; Epoch=25, 100% complete

Beginning training...
Epoch 1 of 25
M=250; Epoch=1, 100% complete

Epoch 2 of 25
M=250; Epoch=2, 100% complete

Epoch 3 of 25
M=250; Epoch=3, 100% complete

Epoch 4 of 25
M=250; Epoch=4, 100% complete
```

```
Epoch 5 of 25
M=250; Epoch=5, 100% complete

Epoch 6 of 25
M=250; Epoch=6, 100% complete

Epoch 7 of 25
M=250; Epoch=7, 100% complete

Epoch 8 of 25
M=250; Epoch=8, 100% complete

Epoch 9 of 25
M=250; Epoch=9, 100% complete

Epoch 10 of 25
M=250; Epoch=10, 100% complete

Epoch 11 of 25
M=250; Epoch=11, 100% complete

Epoch 12 of 25
M=250; Epoch=12, 100% complete

Epoch 13 of 25
M=250; Epoch=13, 100% complete

Epoch 14 of 25
M=250; Epoch=14, 100% complete

Epoch 15 of 25
M=250; Epoch=15, 100% complete

Epoch 16 of 25
M=250; Epoch=16, 100% complete

Epoch 17 of 25
M=250; Epoch=17, 100% complete

Epoch 18 of 25
M=250; Epoch=18, 100% complete

Epoch 19 of 25
M=250; Epoch=19, 100% complete

Epoch 20 of 25
M=250; Epoch=20, 100% complete

Epoch 21 of 25
M=250; Epoch=21, 100% complete

Epoch 22 of 25
M=250; Epoch=22, 100% complete

Epoch 23 of 25
M=250; Epoch=23, 100% complete

Epoch 24 of 25
M=250; Epoch=24, 100% complete
```

```
Epoch 25 of 25
M=250; Epoch=25, 100% complete
```

In [102…

```
for rbm in rbms:
    test(rbm, device, test_loader)
```

```
M=10; Test MSE = 0.2886427856445313
M=50; Test MSE = 0.29383039016723633
M=100; Test MSE = 0.29727467880249026
M=250; Test MSE = 0.3012441173553467
```

Problem 2.1

In [172…

```
%matplotlib inline

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import random_split

import torchvision
import torchvision.transforms as transforms
from torchvision.utils import make_grid

import matplotlib.pyplot as plt
import numpy as np
import os
from tsne import *
```

In [159…

```
# labels for later
master_labels = ["T-shirt", "Pants", "Pullover" , "Dress", "Coat", "Sandal", "Shirt",

# prepare data loaders
batch_size = 128

train_set,test_set,train_loader,test_loader = {},{},{},{}
transform = transforms.Compose(
    [transforms.ToTensor()])

train_set = torchvision.datasets.FashionMNIST(root='../data', train=True, download=Tru
test_set = torchvision.datasets.FashionMNIST(root='../data', train=False, download=Tru
train_set, val_set = random_split(train_set, [int(0.8 * len(train_set)), len(train_set

train_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_size, shuffle=T
val_loader = torch.utils.data.DataLoader(val_set, batch_size=batch_size, shuffle=True,
test_loader = torch.utils.data.DataLoader(test_set, batch_size=batch_size, shuffle=Fal

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

In [160…

```
class ConditioNet(nn.Module):
    def __init__(self, n_class, n_in, n_hid, z_dim):
        super(ConditioNet, self).__init__()

        self.enc1 = nn.Linear(n_in + n_class, n_hid)
```

```
        self.enc2m = nn.Linear(n_hid, z_dim)
        self.enc2s = nn.Linear(n_hid, z_dim)
        self.dec1 = nn.Linear(z_dim, n_hid)
        self.dec2 = nn.Linear(n_hid, n_in)

    def encode(self, x):
        h1 = F.relu(self.enc1(x))
        return self.enc2m(h1), self.enc2s(h1)

    def reparameterize(self, mu, logvar):
        stdev = torch.exp(0.5*logvar)
        eps = torch.randn_like(stdev)
        return mu + eps*stdev

    def decode(self, z):
        h3 = F.relu(self.dec1(z))
        return torch.sigmoid(self.dec2(h3))

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

In [161…

```
def loss_function(recon_x, x, mu, logvar):
    BCE = F.binary_cross_entropy(recon_x, x, reduction='sum') # BCE = -Negative Log-Li
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp()) # KL Divergence b/w
    return BCE + KLD
```

In [162…

```
def train_cVAE(model, device, train_loader, optimizer, epoch):
    train_loss = 0
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data = data.view(data.size(0),-1)
        target_onehot = torch.zeros(data.shape[0], 10)
        target_onehot[range(data.shape[0]), target] = 1
        data_adj = torch.cat((data, target_onehot), dim=1)

        data = data.to(device)
        data_adj = data_adj.to(device)

        optimizer.zero_grad()
        output, mu, logvar = model(data_adj)
        loss = loss_function(output, data, mu, logvar)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()
        if batch_idx % (len(train_loader)//2) == 0:
            print('Train({})[{:.0f}%]: Loss: {:.4f}'.format(
                epoch, 100. * batch_idx / len(train_loader), train_loss/(batch_idx+1))
    return train_loss

def test_cVAE(model, device, test_loader, epoch):
    model.eval()
    test_loss = 0
    with torch.no_grad():
        for data, target in test_loader:
            data = data.view(data.size(0),-1)
            target_onehot = torch.zeros(data.shape[0], 10)
            target_onehot[range(data.shape[0]), target] = 1
```

```
            data_adj = torch.cat((data, target_onehot), dim=1)

            data = data.to(device)
            data_adj = data_adj.to(device)

            output, mu, logvar = model(data_adj)
            loss = loss_function(output, data, mu, logvar)
            test_loss += loss.item() # sum up batch loss
    test_loss = (test_loss*batch_size)/len(test_loader.dataset)
    print('Test({}): Loss: {:.4f}'.format(
        epoch, test_loss))
    return test_loss

def make_optimizer(optimizer_name, model, **kwargs):
    if optimizer_name=='Adam':
        optimizer = optim.Adam(model.parameters(),lr=kwargs['lr'])
    elif optimizer_name=='SGD':
        optimizer = optim.SGD(model.parameters(),lr=kwargs['lr'],momentum=kwargs['mome
    else:
        raise ValueError('Not valid optimizer name')
    return optimizer
```

In [ ]:

```
# set up model training

optimizer_name = 'Adam'

val_num_epochs = 10
train_num_epochs = 10
n_class = 10
n_in = 28*28
n_hids = list(range(100, 801, 200))
z_dims = list(range(5, 51, 25))
lrs = list(range(-4, 0, 1))

device = torch.device(device)


# if present, load tested hyperparameters
if os.path.exists("hyperparameters.pth"):
    hyperparams = torch.load("hyperparameters.pth")
else:
    hyperparams = torch.empty(0,4)


# grid search for best n_hid, z_dim
i = 0
for n_hid in n_hids:
    for z_dim in z_dims:
        for lr_init in lrs:

            # skip iteration if already done
            i += 1
            row_match = (hyperparams[:, 0] == n_hid) & (hyperparams[:, 1] == z_dim) &
            if torch.any(row_match):
                continue

            # set up model
            lr = 10**lr_init
            cvae = ConditioNet(n_class, n_in, n_hid, z_dim).to(device)
```

```
            optimizer = make_optimizer(optimizer_name, cvae, lr=lr)

            # train model
            print(f'Combination {i} of {len(n_hids) * len(z_dims) * len(lrs)}: n_hid =

            cvae.train()
            for epoch in range(1, val_num_epochs + 1):
                train_cVAE(cvae, device, train_loader, optimizer, epoch)

            # generate validation loss
            cvae.eval()
            val_loss = 0
            with torch.no_grad():
                for data, target in val_loader:
                    data = data.view(data.size(0),-1)
                    target_onehot = torch.zeros(data.shape[0], 10)
                    target_onehot[range(data.shape[0]), target] = 1
                    data_adj = torch.cat((data, target_onehot), dim=1)

                    data = data.to(device)
                    data_adj = data_adj.to(device)

                    output, mu, logvar = cvae(data_adj)
                    loss = loss_function(output, data, mu, logvar)
                    val_loss += loss.item() # sum up batch loss
            val_loss = (val_loss*batch_size)/len(val_loader.dataset)

            # save model, if best
            if not os.path.exists("best_cVAE_weights.pth"):
                torch.save(cvae.state_dict(), "best_cVAE_weights.pth")
            elif val_loss < torch.min(hyperparams[:,3]):
                torch.save(cvae.state_dict(), "best_cVAE_weights.pth")

            # store model, validation loss, hyperparameters
            # save hyperparameters
            cur_hyperparams = torch.Tensor([n_hid, z_dim, lr_init, val_loss]).unsqueez
            hyperparams = torch.cat((hyperparams, cur_hyperparams))
            torch.save(hyperparams, "hyperparameters.pth")

print("Search complete!")
```

In [239…

```
# load best model
hyperparams = torch.load("hyperparameters.pth")
best_ind = torch.argmin(hyperparams[:,3])
n_hid_best, z_dim_best, lr_best = [float(elem) for elem in hyperparams[best_ind, :3]]
print(f'Best parameters: n_hid = {n_hid_best}, z_dim = {z_dim_best}, lr = {10**lr_best

best_cvae = ConditioNet(int(n_class), int(n_in), int(n_hid_best), int(z_dim_best))
best_cvae.load_state_dict(torch.load("best_cVAE_weights.pth"))
```

```
Best parameters: n_hid = 700.0, z_dim = 30.0, lr = 0.001
```

Out[239]:

```
<All keys matched successfully>
```

In [240…

```
# set up encoded dataset

# Labels for later
master_labels = ["T-shirt", "Pants", "Pullover", "Dress", "Coat", "Sandal", "Shirt", '

# Load the FashionMNIST dataset
```

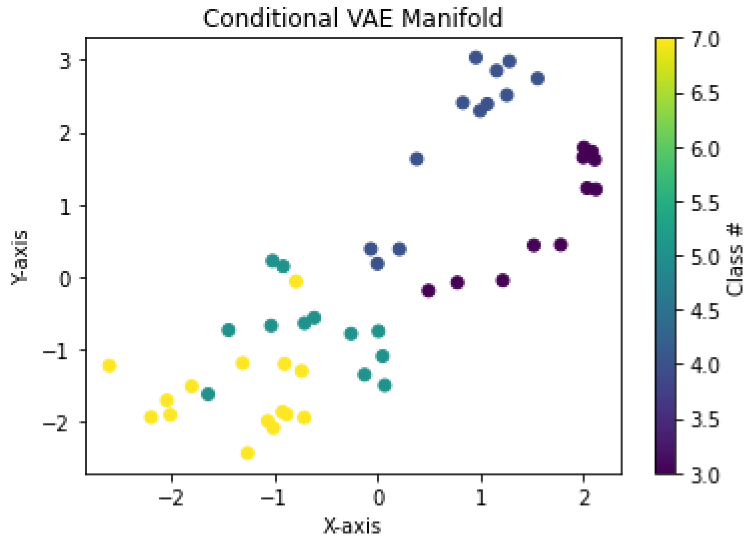
```
dataset = torchvision.datasets.FashionMNIST(root='../data', train=True, download=True,

# set up data loader
# extract data
_, (data_big, target_big) = next(enumerate(test_loader))

data = torch.empty(0,1,28,28)
target = torch.empty(0)

for label in ["Dress", "Coat", "Sandal", "Sneaker"]:
    # subset to label
    # pick first ten elements
    label_num = master_labels.index(label)
    inds = torch.nonzero(target_big == label_num).squeeze()[:64]
    data_sub = data_big[inds,:,:,:]
    target_sub = target_big[inds]

    # stack on tensor
    data = torch.cat((data, data_sub))
    target = torch.cat((target, target_sub))

# encode to latent representations
data = data.view(data.size(0),-1)
target_onehot = torch.zeros(data.shape[0], 10)
target_onehot[range(data.shape[0]), target.long()] = 1
data_adj = torch.cat((data, target_onehot), dim=1)
data_enc, _, _ = best_cvae(data_adj)

# convert latent space to 2-D representation
xys = tsne(data_enc.detach().numpy(), initial_dims=784)
```

```
Preprocessing the data using PCA...
Computing pairwise distances...
Computing P-values for point 0 of 50...
Mean value of sigma: 6.383087
Iteration 10: error is 10.974558
Iteration 20: error is 10.748424
Iteration 30: error is 10.252363
Iteration 40: error is 10.134036
Iteration 50: error is 9.481507
Iteration 60: error is 9.613097
Iteration 70: error is 9.717016
Iteration 80: error is 9.697520
Iteration 90: error is 9.838638
Iteration 100: error is 9.660642
Iteration 110: error is 1.088582
Iteration 120: error is 0.974139
Iteration 130: error is 0.900128
Iteration 140: error is 0.789879
Iteration 150: error is 0.669124
Iteration 160: error is 0.566064
Iteration 170: error is 0.470720
Iteration 180: error is 0.416331
Iteration 190: error is 0.350189
Iteration 200: error is 0.307640
Iteration 210: error is 0.289938
Iteration 220: error is 0.277827
Iteration 230: error is 0.267311
Iteration 240: error is 0.252877
Iteration 250: error is 0.247531
Iteration 260: error is 0.243413
Iteration 270: error is 0.241728
Iteration 280: error is 0.240054
Iteration 290: error is 0.238260
Iteration 300: error is 0.233619
Iteration 310: error is 0.232529
Iteration 320: error is 0.231605
Iteration 330: error is 0.230151
Iteration 340: error is 0.228995
Iteration 350: error is 0.228585
Iteration 360: error is 0.228260
Iteration 370: error is 0.227735
Iteration 380: error is 0.225981
Iteration 390: error is 0.221541
Iteration 400: error is 0.218063
Iteration 410: error is 0.216597
Iteration 420: error is 0.216387
Iteration 430: error is 0.216237
Iteration 440: error is 0.216118
Iteration 450: error is 0.215990
Iteration 460: error is 0.215795
Iteration 470: error is 0.215515
Iteration 480: error is 0.215182
Iteration 490: error is 0.214758
Iteration 500: error is 0.214203
Iteration 510: error is 0.213661
Iteration 520: error is 0.213073
Iteration 530: error is 0.212270
Iteration 540: error is 0.211228
Iteration 550: error is 0.209968
Iteration 560: error is 0.208196
```

```
Iteration 570: error is 0.205702
Iteration 580: error is 0.202326
Iteration 590: error is 0.197139
Iteration 600: error is 0.187681
Iteration 610: error is 0.190025
Iteration 620: error is 0.110890
Iteration 630: error is 0.057002
Iteration 640: error is 0.040258
Iteration 650: error is 0.035163
Iteration 660: error is 0.033650
Iteration 670: error is 0.033171
Iteration 680: error is 0.033055
Iteration 690: error is 0.033027
Iteration 700: error is 0.033018
Iteration 710: error is 0.033016
Iteration 720: error is 0.033015
Iteration 730: error is 0.033014
Iteration 740: error is 0.033014
Iteration 750: error is 0.033014
Iteration 760: error is 0.033014
Iteration 770: error is 0.033014
Iteration 780: error is 0.033014
Iteration 790: error is 0.033014
Iteration 800: error is 0.033014
Iteration 810: error is 0.033014
Iteration 820: error is 0.033014
Iteration 830: error is 0.033014
Iteration 840: error is 0.033014
Iteration 850: error is 0.033014
Iteration 860: error is 0.033014
Iteration 870: error is 0.033014
Iteration 880: error is 0.033014
Iteration 890: error is 0.033014
Iteration 900: error is 0.033014
Iteration 910: error is 0.033014
Iteration 920: error is 0.033014
Iteration 930: error is 0.033014
Iteration 940: error is 0.033014
Iteration 950: error is 0.033014
Iteration 960: error is 0.033014
Iteration 970: error is 0.033014
Iteration 980: error is 0.033014
Iteration 990: error is 0.033014
Iteration 1000: error is 0.033014
```

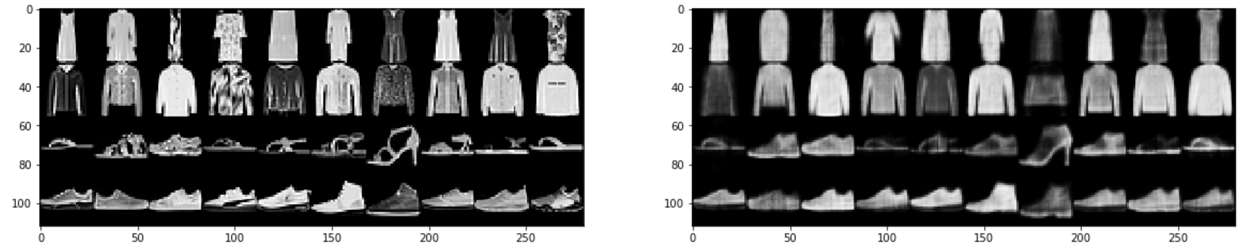
In [241…

```python
# plot manifold
plt.scatter(xys[:, 0], xys[:, 1], c=target, cmap='viridis', marker='o')

# Add labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Conditional VAE Manifold')

# Add colorbar
cbar = plt.colorbar()
cbar.set_label("Class #")

# Show the plot
plt.show()
```



In [233…

```
# generate 10 images per class (dress, coat, sandal, sneaker)

def show(img1, img2):
    npimg1 = img1.cpu().numpy()
    npimg2 = img2.cpu().numpy()

    fig, axes = plt.subplots(1,2, figsize=(20,10))
    axes[0].imshow(np.transpose(npimg1, (1,2,0)), interpolation='nearest')
    axes[1].imshow(np.transpose(npimg2, (1,2,0)), interpolation='nearest')
    fig.show()

data_big, target_big = next(iter(test_loader))

data = torch.empty(0,1,28,28)
target = torch.empty(0)

for label in ["Dress", "Coat", "Sandal", "Sneaker"]:
    # subset to label
    # pick first ten elements
    label_num = master_labels.index(label)
    inds = torch.nonzero(target_big == label_num).squeeze()[:10]
    data_sub = data_big[inds,:,:,:]
    target_sub = target_big[inds]

    # stack on tensor
    data = torch.cat((data, data_sub))
    target = torch.cat((target, target_sub))

data_size = data.size()
data = data.view(data.size(0),-1)
target_onehot = torch.zeros(data.shape[0], 10)
target_onehot[range(data.shape[0]), target.long()] = 1
data_adj = torch.cat((data, target_onehot), dim=1)

data_adj = data_adj.to(device)

output, _, _ = best_cvae(data_adj)
output = output.detach()

show(make_grid(data.reshape(data_size), padding=0, nrow=10), make_grid(output.reshape(
```



Problem 2.2

In [179…

```
# define model architecture

class UnconditioNet(nn.Module):
    def __init__(self, n_in, n_hid, z_dim):
        super(UnconditioNet, self).__init__()

        self.enc1 = nn.Linear(n_in, n_hid)
        self.enc2m = nn.Linear(n_hid, z_dim)
        self.enc2s = nn.Linear(n_hid, z_dim)
        self.dec1 = nn.Linear(z_dim, n_hid)
        self.dec2 = nn.Linear(n_hid, n_in)

    def encode(self, x):
        h1 = F.relu(self.enc1(x))
        return self.enc2m(h1), self.enc2s(h1)

    def reparameterize(self, mu, logvar):
        stdev = torch.exp(0.5*logvar)
        eps = torch.randn_like(stdev)
        return mu + eps*stdev

    def decode(self, z):
        h3 = F.relu(self.dec1(z))
        return torch.sigmoid(self.dec2(h3))

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

In [180…

```
# define training/testing functions

def train_VAE(model, device, train_loader, optimizer, epoch):
    train_loss = 0
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data = data.view(data.size(0),-1)
        data = data.to(device)

        optimizer.zero_grad()
        output, mu, logvar = model(data)
        loss = loss_function(output, data, mu, logvar)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()
        if batch_idx % (len(train_loader)//2) == 0:
```

```
            print('Train({})[{:.0f}%]: Loss: {:.4f}'.format(
                epoch, 100. * batch_idx / len(train_loader), train_loss/(batch_idx+1)))
    return train_loss

def test_VAE(model, device, test_loader, epoch):
    model.eval()
    test_loss = 0
    with torch.no_grad():
        for data, target in test_loader:
            data = data.view(data.size(0),-1)
            data = data.to(device)

            output, mu, logvar = model(data)
            loss = loss_function(output, data, mu, logvar)
            test_loss += loss.item() # sum up batch loss
    test_loss = (test_loss*batch_size)/len(test_loader.dataset)
    print('Test({}): Loss: {:.4f}'.format(
        epoch, test_loss))
    return test_loss
```

In [182…

```
# set up model training

optimizer_name = 'Adam'
train_num_epochs = 10
n_in = 28*28

# load best hyperparameters from cVAE
hyperparams = torch.load("hyperparameters.pth")
best_ind = torch.argmin(hyperparams[:,3])
n_hid, z_dim, lr_init = [int(elem) for elem in hyperparams[best_ind, :3]]
lr = 10**lr_init

# define model
vae = UnconditioNet(n_in, n_hid, z_dim).to(device)
optimizer = make_optimizer(optimizer_name, vae, lr=lr)

# train model
if not os.path.exists("best_VAE_weights.pth"):
    vae.train()
    for epoch in range(1, train_num_epochs + 1):
        train_VAE(vae, device, train_loader, optimizer, epoch)
    torch.save(vae.state_dict(), "best_VAE_weights.pth")
else:
    vae = UnconditioNet(int(n_in), int(n_hid_best), int(z_dim_best))
    vae.load_state_dict(torch.load("best_VAE_weights.pth"))

# test model
test_VAE(vae, device, test_loader, epoch)
```

```
Train(1)[0%]: Loss: 70462.5234
Train(1)[50%]: Loss: 39079.2938
Train(1)[100%]: Loss: 36685.3317
Train(2)[0%]: Loss: 33409.1875
Train(2)[50%]: Loss: 33190.3864
Train(2)[100%]: Loss: 32970.8460
Train(3)[0%]: Loss: 32344.4297
Train(3)[50%]: Loss: 32347.6265
Train(3)[100%]: Loss: 32188.3758
Train(4)[0%]: Loss: 31282.3691
Train(4)[50%]: Loss: 31885.5240
Train(4)[100%]: Loss: 31787.3562
Train(5)[0%]: Loss: 31286.6758
Train(5)[50%]: Loss: 31577.1630
Train(5)[100%]: Loss: 31546.4689
Train(6)[0%]: Loss: 31323.7559
Train(6)[50%]: Loss: 31412.9422
Train(6)[100%]: Loss: 31377.9458
Train(7)[0%]: Loss: 30531.8203
Train(7)[50%]: Loss: 31286.3584
Train(7)[100%]: Loss: 31269.9615
Train(8)[0%]: Loss: 31965.9844
Train(8)[50%]: Loss: 31170.0682
Train(8)[100%]: Loss: 31175.4792
Train(9)[0%]: Loss: 31353.0176
Train(9)[50%]: Loss: 31173.5927
Train(9)[100%]: Loss: 31100.7907
Train(10)[0%]: Loss: 31239.4883
Train(10)[50%]: Loss: 31049.2272
Train(10)[100%]: Loss: 31036.9186
Test(10): Loss: 31224.8554
```

Out[182]:

```
31224.85535
```

In [242…

```python
# set up encoded dataset

# Labels for later
master_labels = ["T-shirt", "Pants", "Pullover", "Dress", "Coat", "Sandal", "Shirt", '

# Load the FashionMNIST dataset
dataset = torchvision.datasets.FashionMNIST(root='../data', train=True, download=True,

# set up data loader
# extract data
_, (data_big, target_big) = next(enumerate(test_loader))

data = torch.empty(0,1,28,28)
target = torch.empty(0)

for label in ["Dress", "Coat", "Sandal", "Sneaker"]:
    # subset to label
    # pick first ten elements
    label_num = master_labels.index(label)
    inds = torch.nonzero(target_big == label_num).squeeze()[:64]
    data_sub = data_big[inds,:,:,:]
    target_sub = target_big[inds]

    # stack on tensor
    data = torch.cat((data, data_sub))
    target = torch.cat((target, target_sub))
```

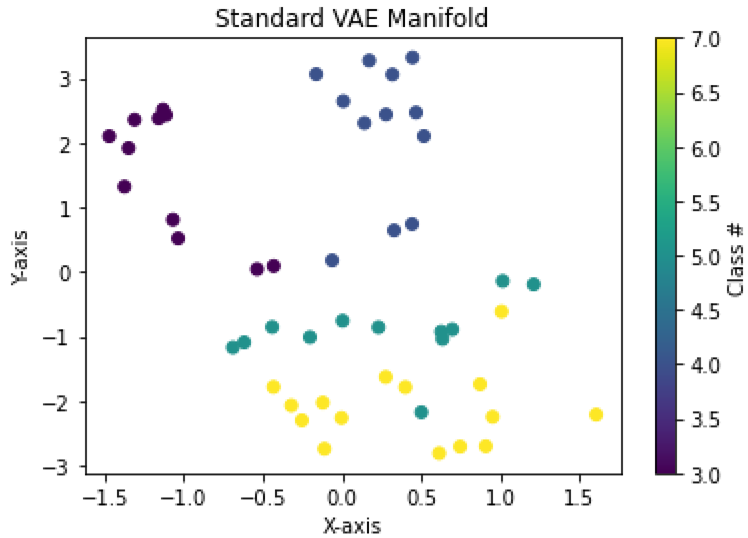
```
# encode to latent representations
data = data.view(data.size(0),-1)
data_adj = data
data_enc, _, _ = vae(data_adj)

# convert latent space to 2-D representation
xys_vae = tsne(data_enc.detach().numpy(), initial_dims=784)
```

```
Preprocessing the data using PCA...
Computing pairwise distances...
Computing P-values for point 0 of 50...
Mean value of sigma: 6.354944
Iteration 10: error is 10.027535
Iteration 20: error is 10.160423
Iteration 30: error is 9.962111
Iteration 40: error is 11.193229
Iteration 50: error is 10.240972
Iteration 60: error is 11.062531
Iteration 70: error is 9.978937
Iteration 80: error is 10.732067
Iteration 90: error is 11.169940
Iteration 100: error is 10.373302
Iteration 110: error is 1.092721
Iteration 120: error is 0.927085
Iteration 130: error is 0.866500
Iteration 140: error is 0.794036
Iteration 150: error is 0.694102
Iteration 160: error is 0.626317
Iteration 170: error is 0.569957
Iteration 180: error is 0.530376
Iteration 190: error is 0.503395
Iteration 200: error is 0.456229
Iteration 210: error is 0.398342
Iteration 220: error is 0.330634
Iteration 230: error is 0.320761
Iteration 240: error is 0.299345
Iteration 250: error is 0.274778
Iteration 260: error is 0.256822
Iteration 270: error is 0.239303
Iteration 280: error is 0.226879
Iteration 290: error is 0.217101
Iteration 300: error is 0.214251
Iteration 310: error is 0.213062
Iteration 320: error is 0.212251
Iteration 330: error is 0.209922
Iteration 340: error is 0.207859
Iteration 350: error is 0.204048
Iteration 360: error is 0.203233
Iteration 370: error is 0.202255
Iteration 380: error is 0.201709
Iteration 390: error is 0.200751
Iteration 400: error is 0.198218
Iteration 410: error is 0.196753
Iteration 420: error is 0.194952
Iteration 430: error is 0.193146
Iteration 440: error is 0.191432
Iteration 450: error is 0.189831
Iteration 460: error is 0.189567
Iteration 470: error is 0.189309
Iteration 480: error is 0.189024
Iteration 490: error is 0.188665
Iteration 500: error is 0.188429
Iteration 510: error is 0.188247
Iteration 520: error is 0.188014
Iteration 530: error is 0.187735
Iteration 540: error is 0.187395
Iteration 550: error is 0.186956
Iteration 560: error is 0.186460
```

```
Iteration 570: error is 0.185845
Iteration 580: error is 0.185163
Iteration 590: error is 0.184263
Iteration 600: error is 0.183250
Iteration 610: error is 0.182068
Iteration 620: error is 0.180346
Iteration 630: error is 0.177694
Iteration 640: error is 0.173538
Iteration 650: error is 0.165998
Iteration 660: error is 0.152458
Iteration 670: error is 0.065144
Iteration 680: error is 0.055504
Iteration 690: error is 0.040701
Iteration 700: error is 0.036699
Iteration 710: error is 0.035352
Iteration 720: error is 0.034855
Iteration 730: error is 0.034706
Iteration 740: error is 0.034641
Iteration 750: error is 0.034620
Iteration 760: error is 0.034609
Iteration 770: error is 0.034605
Iteration 780: error is 0.034604
Iteration 790: error is 0.034604
Iteration 800: error is 0.034604
Iteration 810: error is 0.034604
Iteration 820: error is 0.034604
Iteration 830: error is 0.034604
Iteration 840: error is 0.034604
Iteration 850: error is 0.034604
Iteration 860: error is 0.034604
Iteration 870: error is 0.034604
Iteration 880: error is 0.034604
Iteration 890: error is 0.034604
Iteration 900: error is 0.034604
Iteration 910: error is 0.034604
Iteration 920: error is 0.034604
Iteration 930: error is 0.034604
Iteration 940: error is 0.034604
Iteration 950: error is 0.034604
Iteration 960: error is 0.034604
Iteration 970: error is 0.034604
Iteration 980: error is 0.034604
Iteration 990: error is 0.034604
Iteration 1000: error is 0.034604
```

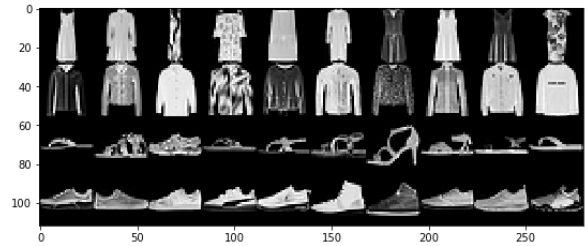
In [243…

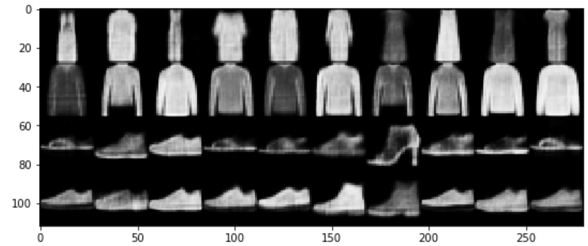
```
# plot manifold
plt.scatter(xys_vae[:, 0], xys_vae[:, 1], c=target, cmap='viridis', marker='o')

# Add labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Standard VAE Manifold')

# Add colorbar
cbar = plt.colorbar()
cbar.set_label("Class #")

# Show the plot
plt.show()
```



In [235…

```
# generate 10 images per class (dress, coat, sandal, sneaker)

def show(img1, img2):
    npimg1 = img1.cpu().numpy()
    npimg2 = img2.cpu().numpy()

    fig, axes = plt.subplots(1,2, figsize=(20,10))
    axes[0].imshow(np.transpose(npimg1, (1,2,0)), interpolation='nearest')
    axes[1].imshow(np.transpose(npimg2, (1,2,0)), interpolation='nearest')
    fig.show()

data_big, target_big = next(iter(test_loader))

data = torch.empty(0,1,28,28)
target = torch.empty(0)

for label in ["Dress", "Coat", "Sandal", "Sneaker"]:
    # subset to label
    # pick first ten elements
    label_num = master_labels.index(label)
    inds = torch.nonzero(target_big == label_num).squeeze()[:10]
    data_sub = data_big[inds,:,:,:]
    target_sub = target_big[inds]

    # stack on tensor
    data = torch.cat((data, data_sub))
    target = torch.cat((target, target_sub))

data_size = data.size()
data = data.view(data.size(0),-1)
data_adj = data

data_adj = data_adj.to(device)

output, _, _ = vae(data_adj)
output = output.detach()

show(make_grid(data.reshape(data_size), padding=0, nrow=10), make_grid(output.reshape(
```





Interpretation: Comparing the two VAEs, the manifold in the conditional variant is noticeably more continuous than in the standard version. There is also more mixing between members of different classes, indicating we can get smoother transformations between them.