ECE685D HW5

Zanwen Fu

November 23 2025

LLM Policy: ChatGPT and Gemini 3 were used to evaluate the correctness of answers.

Q1

November 23, 2025

1 Problem 1: GAN

1.1 1.1 Implement a Deep Convolutional GAN (DCGAN)

[1]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
from torchvision.utils import make_grid
import matplotlib.pyplot as plt

if torch.cuda.is_available():
    device = torch.device("cuda")
elif getattr(torch.backends, "mps", None) is not None and torch.backends.mps.
 ↪is_available():
    device = torch.device("mps")
else:
    device = torch.device("cpu")
```

[2]:
```
batch_size = 128

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))  # scale to [-1, 1]
])

train_dataset = datasets.MNIST(
    root="./data",
    train=True,          # *** ONLY TRAIN SPLIT ***
    transform=transform,
    download=True
)

train_loader = DataLoader(
    train_dataset,
    batch_size=batch_size,
    shuffle=True,
```

```
    num_workers=2,
    pin_memory=True if device.type == "cuda" else False
)

len(train_dataset)
```

[2]: 60000

[3]:
```
class Discriminator(nn.Module):
    def __init__(self, p_drop=0.3):
        super().__init__()
        self.net = nn.Sequential(
            # 28x28x1 -> 14x14x64
            nn.Conv2d(1, 64, kernel_size=4, stride=2, padding=1),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout2d(p_drop),

            # 14x14x64 -> 7x7x128
            nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout2d(p_drop),

            # 7x7x128 -> 4x4x256
            nn.Conv2d(128, 256, kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout2d(p_drop),

            # 4x4x256 -> 4x4x512
            nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(512),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout2d(p_drop),
        )

        self.classifier = nn.Linear(4 * 4 * 512, 1)

    def forward(self, x):
        x = self.net(x)
        x = x.view(x.size(0), -1)
        logits = self.classifier(x)
        return logits
```

[4]:
```
class Generator(nn.Module):
    def __init__(self, noise_dim=100, p_drop=0.2):
        super().__init__()
```

```
        self.noise_dim = noise_dim
        self.fc = nn.Sequential(
            nn.Linear(noise_dim, 256 * 7 * 7),
            nn.Dropout(p_drop),
        )

        self.net = nn.Sequential(
            nn.ConvTranspose2d(256, 128, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(True),

            nn.ConvTranspose2d(128, 64, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(True),

            nn.Conv2d(64, 32, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(True),

            nn.Conv2d(32, 1, kernel_size=3, stride=1, padding=1),
            nn.Tanh()
        )

    def forward(self, z):
        x = self.fc(z)
        x = x.view(z.size(0), 256, 7, 7)
        img = self.net(x)
        return img
```

[5]:
```
noise_dim = 100

G = Generator(noise_dim=noise_dim).to(device)
D = Discriminator().to(device)

criterion = nn.BCEWithLogitsLoss()

lr = 2e-4
beta1 = 0.5
beta2 = 0.999

optimizer_D = optim.Adam(D.parameters(), lr=lr, betas=(beta1, beta2))
optimizer_G = optim.Adam(G.parameters(), lr=lr, betas=(beta1, beta2))
```

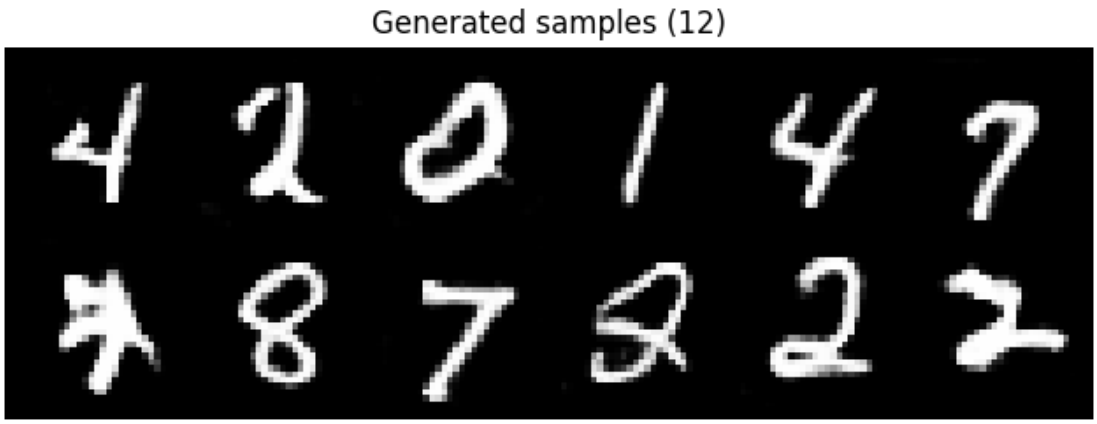
[6]:
```
num_epochs = 20  # adjust as needed

for epoch in range(1, num_epochs + 1):
    G.train(); D.train()
```

```
    running_loss_D = 0.0
    running_loss_G = 0.0

    for real_imgs, _ in train_loader:
        real_imgs = real_imgs.to(device)
        b = real_imgs.size(0)

        real_labels = torch.ones(b, 1, device=device)
        fake_labels = torch.zeros(b, 1, device=device)

        # ----- Train D -----
        optimizer_D.zero_grad()

        logits_real = D(real_imgs)
        loss_real = criterion(logits_real, real_labels)

        z = torch.randn(b, noise_dim, device=device)
        with torch.no_grad():
            fake_imgs = G(z)
        logits_fake = D(fake_imgs)
        loss_fake = criterion(logits_fake, fake_labels)

        loss_D = loss_real + loss_fake
        loss_D.backward()
        optimizer_D.step()

        # ----- Train G -----
        optimizer_G.zero_grad()

        z = torch.randn(b, noise_dim, device=device)
        fake_imgs = G(z)
        logits_fake_for_G = D(fake_imgs)
        D_fake = torch.sigmoid(logits_fake_for_G)

        #  _G = Ez[ log(1 - D(G(z))) ]
        loss_G = torch.log(1 - D_fake + 1e-8).mean()    # this will be   0
        loss_G.backward()
        optimizer_G.step()

        running_loss_D += loss_D.item() * b
        running_loss_G += loss_G.item() * b

    avg_loss_D = running_loss_D / len(train_dataset)
    avg_loss_G = running_loss_G / len(train_dataset)
    print(f"Epoch [{epoch}/{num_epochs}]  Loss_D: {avg_loss_D:.4f}  Loss_G:␣
↪{avg_loss_G:.4f}")
```

```
Epoch [1/20]  Loss_D: 1.0388  Loss_G: -0.4796
```

```
Epoch [2/20]  Loss_D: 1.0796  Loss_G: -0.5082
Epoch [3/20]  Loss_D: 1.0864  Loss_G: -0.5235
Epoch [4/20]  Loss_D: 1.0785  Loss_G: -0.5201
Epoch [5/20]  Loss_D: 1.0815  Loss_G: -0.5186
Epoch [6/20]  Loss_D: 1.0971  Loss_G: -0.5261
Epoch [7/20]  Loss_D: 1.1109  Loss_G: -0.5323
Epoch [8/20]  Loss_D: 1.1197  Loss_G: -0.5293
Epoch [9/20]  Loss_D: 1.1293  Loss_G: -0.5401
Epoch [10/20]  Loss_D: 1.1301  Loss_G: -0.5535
Epoch [11/20]  Loss_D: 1.1537  Loss_G: -0.5482
Epoch [12/20]  Loss_D: 1.1554  Loss_G: -0.5704
Epoch [13/20]  Loss_D: 1.1587  Loss_G: -0.5594
Epoch [14/20]  Loss_D: 1.1599  Loss_G: -0.5623
Epoch [15/20]  Loss_D: 1.1702  Loss_G: -0.5565
Epoch [16/20]  Loss_D: 1.1727  Loss_G: -0.5595
Epoch [17/20]  Loss_D: 1.1757  Loss_G: -0.5752
Epoch [18/20]  Loss_D: 1.1619  Loss_G: -0.5565
Epoch [19/20]  Loss_D: 1.1707  Loss_G: -0.5609
Epoch [20/20]  Loss_D: 1.1639  Loss_G: -0.5649
```

[7]:
```python
G.eval()
with torch.no_grad():
    z = torch.randn(12, noise_dim, device=device)
    gen_imgs = G(z).cpu()

# from [-1, 1] back to [0, 1]
gen_imgs = (gen_imgs + 1) / 2.0
gen_imgs = gen_imgs.clamp(0, 1)

grid = make_grid(gen_imgs, nrow=6, padding=2)

plt.figure(figsize=(8, 4))
plt.imshow(grid.permute(1, 2, 0).squeeze(), cmap="gray")
plt.axis("off")
plt.title("Generated samples (12)")
plt.show()
```



1.2 1.2 GAN as a pre-training framework

[ ]:
```python
class DiscriminatorFeatureExtractor(nn.Module):
    def __init__(self, discriminator):
        super().__init__()
        self.net = discriminator.net  # all conv / BN / dropout blocks

    def forward(self, x):
        x = self.net(x)
        return x.view(x.size(0), -1)  # flatten 4x4x512 -> 8192

feature_extractor = DiscriminatorFeatureExtractor(D).to(device)
feature_extractor.eval()

# freeze all its parameters (we don't train D anymore)
for p in feature_extractor.parameters():
    p.requires_grad = False

# infer feature dimension
example_batch, _ = next(iter(train_loader))
example_batch = example_batch.to(device)
with torch.no_grad():
    feat_example = feature_extractor(example_batch)
feature_dim = feat_example.shape[1]
print("Feature dimension:", feature_dim)
```

```
Feature dimension: 8192
```

[9]:
```python
from torch.utils.data import Subset

# 10% of training set
num_train = len(train_dataset)
```

```
subset_size = num_train // 10  # 6000 for MNIST
indices = torch.randperm(num_train)[:subset_size]
train_subset = Subset(train_dataset, indices)

subset_loader = DataLoader(
    train_subset,
    batch_size=batch_size,
    shuffle=True,
    num_workers=2,
    pin_memory=True if device.type == "cuda" else False
)

# Test set
test_dataset = datasets.MNIST(
    root="./data",
    train=False,
    transform=transform,
    download=True
)

test_loader = DataLoader(
    test_dataset,
    batch_size=batch_size,
    shuffle=False,
    num_workers=2,
    pin_memory=True if device.type == "cuda" else False
)

len(train_subset), len(test_dataset)
```

[9]:
```
(6000, 10000)
```

[10]:
```
class LinearClassifier(nn.Module):
    def __init__(self, feature_dim, num_classes=10):
        super().__init__()
        self.fc = nn.Linear(feature_dim, num_classes)

    def forward(self, features):
        return self.fc(features)

classifier = LinearClassifier(feature_dim).to(device)

criterion_cls = nn.CrossEntropyLoss()
optimizer_cls = optim.Adam(classifier.parameters(), lr=1e-3)
```

[ ]:
```
num_epochs_cls = 10  # classification epochs
```

```
for epoch in range(1, num_epochs_cls + 1):
    # ----- train -----
    classifier.train()
    feature_extractor.eval()

    running_loss = 0.0
    correct = 0
    total = 0

    for images, labels in subset_loader:
        images = images.to(device)
        labels = labels.to(device)

        with torch.no_grad():          # only classifier is trained
            feats = feature_extractor(images)

        optimizer_cls.zero_grad()
        logits = classifier(feats)
        loss = criterion_cls(logits, labels)
        loss.backward()
        optimizer_cls.step()

        running_loss += loss.item() * labels.size(0)
        _, predicted = logits.max(1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    train_loss = running_loss / total
    train_acc = 100.0 * correct / total

    # ----- evaluate on test set -----
    classifier.eval()
    test_correct = 0
    test_total = 0
    with torch.no_grad():
        for images, labels in test_loader:
            images = images.to(device)
            labels = labels.to(device)
            feats = feature_extractor(images)
            logits = classifier(feats)
            _, predicted = logits.max(1)
            test_total += labels.size(0)
            test_correct += (predicted == labels).sum().item()

    test_acc = 100.0 * test_correct / test_total

    print(f"[Epoch {epoch}/{num_epochs_cls}] "
```

```
          f"Train Loss: {train_loss:.4f}  "
          f"Train Acc: {train_acc:.2f}%  "
          f"Test Acc: {test_acc:.2f}%")
```

```
[Epoch 1/10] Train Loss: 0.2346  Train Acc: 93.15%  Test Acc: 97.43%
[Epoch 2/10] Train Loss: 0.0671  Train Acc: 97.88%  Test Acc: 97.76%
[Epoch 3/10] Train Loss: 0.0432  Train Acc: 98.63%  Test Acc: 97.92%
[Epoch 4/10] Train Loss: 0.0291  Train Acc: 99.08%  Test Acc: 97.82%
[Epoch 5/10] Train Loss: 0.0226  Train Acc: 99.40%  Test Acc: 97.90%
[Epoch 6/10] Train Loss: 0.0189  Train Acc: 99.55%  Test Acc: 97.92%
[Epoch 7/10] Train Loss: 0.0123  Train Acc: 99.77%  Test Acc: 98.22%
[Epoch 8/10] Train Loss: 0.0092  Train Acc: 99.92%  Test Acc: 98.20%
[Epoch 9/10] Train Loss: 0.0076  Train Acc: 99.95%  Test Acc: 98.19%
[Epoch 10/10] Train Loss: 0.0064  Train Acc: 99.93%  Test Acc: 98.17%
```

1.3 1.2 Answer:

After training the DCGAN in Part 1.1, I removed the final linear layer of the discriminator and used the remaining convolutional blocks as a fixed feature extractor. These layers were frozen, and I extracted features for 10% of the MNIST training set. On top of these features, I trained a single linear classifier to predict digit labels (0–9). The model was evaluated on the official MNIST test split.

Results | Epoch | Train Acc | Test Acc | | — | ———- | ———- | | 1 | 93.15% | 97.43% | | 5 | 99.40% | 97.90% | | 10 | **99.93%** | **98.17%** |

Discussion The classifier reached nearly 100% training accuracy and consistently achieved around 98% test accuracy. This performance is significantly higher than training a classifier from scratch using only 10% labeled data, indicating that the GAN discriminator learned meaningful visual features during adversarial training. These pretrained features enabled the linear classifier to generalize well despite limited supervision.

Conclusion: GAN-based pretraining provides a strong feature representation that improves downstream classification performance when labeled data is scarce.

ECE685D HW5

Zanwen Fu

November 2025

1 Solution to 2.1

The joint distribution of a Gaussian–Bernoulli RBM is

$$p(v,h) = \frac{1}{Z}\exp\left(\sum_i\sum_j W_{ij}h_j\frac{v_i}{\sigma_i} - \sum_i\frac{(v_i-b_i)^2}{2\sigma_i^2} + \sum_j \alpha_j h_j\right),$$

derived from the energy

$$E(v,h;\theta) = -\left(\sum_i\sum_j W_{ij}h_j\frac{v_i}{\sigma_i} - \sum_i\frac{(v_i-b_i)^2}{2\sigma_i^2} + \sum_j \alpha_j h_j\right).$$

(a) Conditional distribution $p(h_j = 1 \mid v)$

Starting from

$$p(h \mid v) \propto p(v,h),$$

we keep only the terms depending on h:

$$p(h \mid v) \propto \exp\left(\sum_j h_j\left(\alpha_j + \sum_i W_{ij}\frac{v_i}{\sigma_i}\right)\right).$$

Because $h_j \in \{0,1\}$ and hidden units factorize,

$$p(h_j = 1 \mid v) = \frac{\exp\left(\alpha_j + \sum_i W_{ij}\frac{v_i}{\sigma_i}\right)}{1+\exp\left(\alpha_j + \sum_i W_{ij}\frac{v_i}{\sigma_i}\right)}.$$

Thus,

$$p(h_j = 1 \mid v) = \sigma\left(\alpha_j + \sum_i W_{ij}\frac{v_i}{\sigma_i}\right).$$

(b) Conditional distribution $p(v_i \mid h)$

Using Bayes' rule,

$$p(v_i \mid h) \propto p(v_i, h),$$

and keeping only terms involving v_i,

$$p(v_i \mid h) \propto \exp\left(\sum_j W_{ij} h_j \frac{v_i}{\sigma_i} - \frac{(v_i - b_i)^2}{2\sigma_i^2} \right).$$

Let

$$A_i = \sum_j W_{ij} h_j / \sigma_i.$$

Then the exponent becomes

$$\sum_j W_{ij} h_j \frac{v_i}{\sigma_i} - \frac{(v_i - b_i)^2}{2\sigma_i^2} = -\frac{1}{2\sigma_i^2}\Big[(v_i - b_i)^2 - 2\sigma_i^2 A_i v_i\Big].$$

Completing the square yields

$$p(v_i \mid h) \propto \exp\left(-\frac{(v_i - \mu_i)^2}{2\sigma_i^2}\right), \qquad \mu_i = b_i + \sigma_i \sum_j W_{ij} h_j.$$

Therefore,

$$p(v_i \mid h) = \mathcal{N}\left(v_i;\, b_i + \sigma_i \sum_j W_{ij} h_j,\; \sigma_i^2 \right).$$

Integral form (optional, as allowed in the question):

$$p(v_i = x \mid h) = \frac{1}{C(h)} \exp\left(\sum_j W_{ij} h_j \frac{x}{\sigma_i} - \frac{(x - b_i)^2}{2\sigma_i^2} \right),$$

where $C(h)$ is the normalization constant obtained by integrating over x.

Q2

November 23, 2025

1 Problem 2: Gaussian-Bernoulli Restricted Boltzmann Machines

1.1 2.2. GB-RBM on KMNIST

[1]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
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
```

[2]:
```
if torch.cuda.is_available():
    device = torch.device("cuda")
elif getattr(torch.backends, "mps", None) is not None and torch.backends.mps.
 ↪is_available():
    device = torch.device("mps")
else:
    device = torch.device("cpu")

batch_size = 128
transform = transforms.Compose([transforms.ToTensor()])

train_set = torchvision.datasets.KMNIST(
    root='./data', train=True, download=True, transform=transform
)
test_set = torchvision.datasets.KMNIST(
    root='./data', train=False, download=True, transform=transform
)

train_loader = torch.utils.data.DataLoader(
    train_set, batch_size=batch_size, shuffle=True, num_workers=0
)
test_loader = torch.utils.data.DataLoader(
```

```
    test_set, batch_size=batch_size, shuffle=False, num_workers=0
)
```

```
100%|      | 18.2M/18.2M [00:21<00:00, 836kB/s]
100%|      | 29.5k/29.5k [00:00<00:00, 168kB/s]
100%|      | 3.04M/3.04M [00:02<00:00, 1.16MB/s]
100%|      | 5.12k/5.12k [00:00<00:00, 1.75MB/s]
```

[3]:
```
class RBM(nn.Module):
    """Gaussian-Bernoulli RBM (visible Gaussian, hidden Bernoulli)."""

    def __init__(self, D: int, F: int, k: int):
        super().__init__()
        self.W = nn.Parameter(torch.randn(F, D) * 1e-2)  # [F, D]
        self.c = nn.Parameter(torch.zeros(D))            # visible bias
        self.b = nn.Parameter(torch.zeros(F))            # hidden bias
        self.k = k                                       # CD-k steps

    def sample(self, p):
        """Sample Bernoulli with parameter p."""
        return torch.bernoulli(p)

    def sample_gaussian(self, mean):
        """Sample Gaussian N(mean, 1)."""
        return mean + torch.randn_like(mean)

    def P_h_x(self, x):
        """p(h=1 | x) for each hidden unit."""
        # x: [B, D] -> [B, F]
        logits = F.linear(x, self.W, self.b)
        return torch.sigmoid(logits)

    def P_x_h(self, h):
        """Mean of p(x | h) (Gaussian)."""
        # h: [B, F] -> [B, D]
        mean = F.linear(h, self.W.t(), self.c)
        return mean

    def free_energy(self, x):
        """Free energy F(x) for Gaussian-Bernoulli RBM with unit variance."""
        # v-term
        v_term = 0.5 * ((x - self.c) ** 2).sum(dim=1)
        # h-term: sum_j log(1 + exp(b_j + W_j v))
        wx_b = F.linear(x, self.W, self.b)
        h_term = F.softplus(wx_b).sum(dim=1)
        return v_term - h_term
```

```
    def forward(self, x):
        """
        Run CD-k starting from x.
        Returns:
          x_k        : final negative sample
          x_recon_mu : mean of p(x | h_k) (used for reconstruction plots)
        """
        x_k = x
        for _ in range(self.k):
            p_h_given_x = self.P_h_x(x_k)
            h_k = self.sample(p_h_given_x)
            x_mu = self.P_x_h(h_k)
            x_k = self.sample_gaussian(x_mu)
        # last mean (without noise) for nicer reconstructions
        x_recon_mu = self.P_x_h(self.P_h_x(x_k))
        return x_k, x_recon_mu
```

[ ]:
```
def train(model, device, train_loader, optimizer, epoch):
    model.train()
    train_loss = 0.0

    for batch_idx, (data, _) in enumerate(train_loader):
        data = data.view(data.size(0), -1).to(device)
        mean, std = data.mean(), data.std()
        data_norm = (data - mean) / std

        optimizer.zero_grad()

        # CD-k
        x_k, _ = model(data_norm)
        # contrastive divergence estimate of -log p(x)
        loss = (model.free_energy(data_norm) -
                model.free_energy(x_k.detach())).mean()

        loss.backward()
        optimizer.step()

        train_loss += loss.item()
        if (batch_idx + 1) % (len(train_loader) // 2) == 0:
            print('Train({})[{:.0f}%]: Loss: {:.4f}'.format(
                epoch, 100. * batch_idx / len(train_loader),
                train_loss / (batch_idx + 1)))


def test(model, device, test_loader, epoch):
    model.eval()
    test_loss = 0.0
```

```
    with torch.no_grad():
        for data, _ in test_loader:
            data = data.view(data.size(0), -1).to(device)
            mean, std = data.mean(), data.std()
            data_norm = (data - mean) / std

            x_k, _ = model(data_norm)
            loss = (model.free_energy(data_norm) -
                    model.free_energy(x_k)).mean()
            test_loss += loss.item()

    test_loss = (test_loss * batch_size) / len(test_loader.dataset)
    print('Test({}): Loss: {:.4f}'.format(epoch, test_loss))
```

[5]:
```
def show(img1, img2):
    npimg1 = img1.cpu().numpy()
    npimg2 = img2.cpu().numpy()

    fig, axes = plt.subplots(1, 2, figsize=(12, 6))
    axes[0].imshow(np.transpose(npimg1, (1, 2, 0)), interpolation='nearest')
    axes[0].set_title("Original")
    axes[0].axis('off')
    axes[1].imshow(np.transpose(npimg2, (1, 2, 0)), interpolation='nearest')
    axes[1].set_title("Reconstruction")
    axes[1].axis('off')
    plt.show()
```

[6]:
```
def reconstruction_mse(model, device, data_loader):
    model.eval()
    sq_err_sum = 0.0
    n_pixels = 0

    with torch.no_grad():
        for data, _ in data_loader:
            data = data.view(data.size(0), -1).to(device)
            mean, std = data.mean(), data.std()
            data_norm = (data - mean) / std

            _, x_recon_norm = model(data_norm)
            x_recon = x_recon_norm * std + mean

            # MSE over all pixels
            mse_batch = F.mse_loss(x_recon, data, reduction='sum')
            sq_err_sum += mse_batch.item()
            n_pixels += data.numel()
```

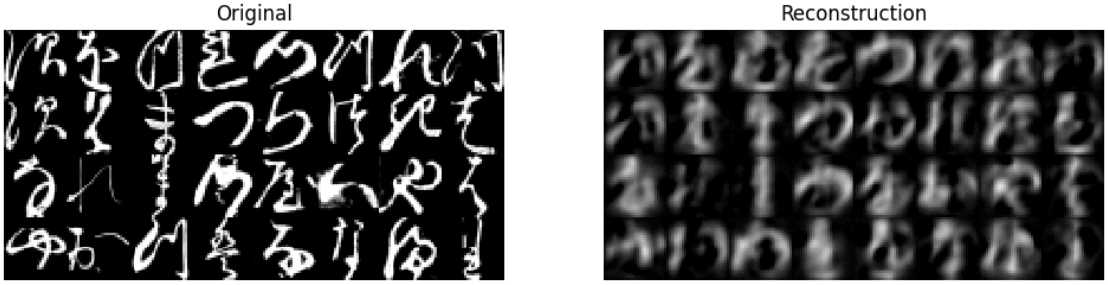
```
    return sq_err_sum / n_pixels
```

[ ]:
```
seed = 42
num_epochs = 25
torch.manual_seed(seed)

M_list = [16, 64, 256]
D = 28 * 28

for M in M_list:
    print("=" * 60)
    print(f"M = {M}")
    rbm = RBM(D=D, F=M, k=5).to(device)
    optimizer = optim.Adam(rbm.parameters(), lr=1e-3, weight_decay=0.0)

    for epoch in range(1, num_epochs + 1):
        train(rbm, device, train_loader, optimizer, epoch)
        test(rbm, device, test_loader, epoch)

    # --- reconstruction plots on train & test ---
    # test set
    data_test, _ = next(iter(test_loader))
    data_test = data_test[:32]
    data_size = data_test.size()
    data_flat = data_test.view(data_test.size(0), -1).to(device)
    mean, std = data_flat.mean(), data_flat.std()
    data_norm = (data_flat - mean) / std
    _, recon_norm = rbm(data_norm)
    recon = (recon_norm * std + mean).clamp(0, 1)

    print(f"Sample reconstructions on test set for M={M}")
    show(make_grid(data_flat.view(data_size), padding=0),
         make_grid(recon.view(data_size), padding=0))

    # train set
    data_train, _ = next(iter(train_loader))
    data_train = data_train[:32]
    data_size_tr = data_train.size()
    data_flat_tr = data_train.view(data_train.size(0), -1).to(device)
    mean_tr, std_tr = data_flat_tr.mean(), data_flat_tr.std()
    data_norm_tr = (data_flat_tr - mean_tr) / std_tr
    _, recon_norm_tr = rbm(data_norm_tr)
    recon_tr = (recon_norm_tr * std_tr + mean_tr).clamp(0, 1)

    print(f"Sample reconstructions on train set for M={M}")
    show(make_grid(data_flat_tr.view(data_size_tr), padding=0),
         make_grid(recon_tr.view(data_size_tr), padding=0))
```

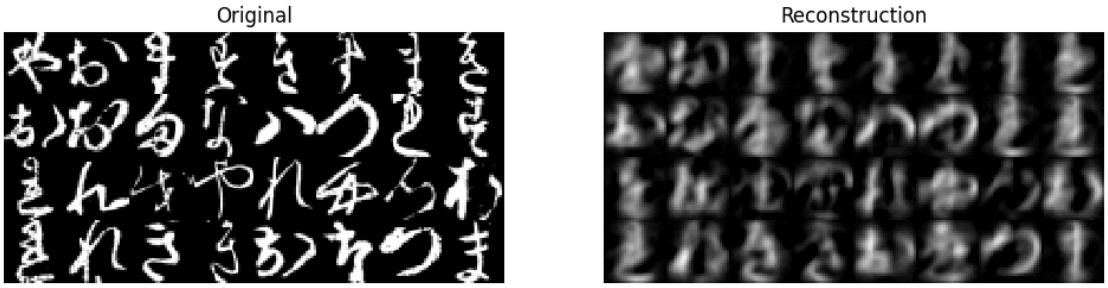
```
    # --- report MSE on train & test ---
    train_mse = reconstruction_mse(rbm, device, train_loader)
    test_mse = reconstruction_mse(rbm, device, test_loader)
    print(f"M={M}: Train reconstruction MSE = {train_mse:.6f}")
    print(f"M={M}: Test  reconstruction MSE = {test_mse:.6f}\n")
```

```
============================================================
M = 16
Train(1)[50%]: Loss: -45.4647
Train(1)[100%]: Loss: -63.3583
Test(1): Loss: -84.6611
Train(2)[50%]: Loss: -97.7533
Train(2)[100%]: Loss: -104.1220
Test(2): Loss: -107.0049
Train(3)[50%]: Loss: -119.5804
Train(3)[100%]: Loss: -122.7830
Test(3): Loss: -118.8012
Train(4)[50%]: Loss: -131.1427
Train(4)[100%]: Loss: -132.9202
Test(4): Loss: -126.7871
Train(5)[50%]: Loss: -138.4676
Train(5)[100%]: Loss: -139.6793
Test(5): Loss: -131.9055
Train(6)[50%]: Loss: -142.7850
Train(6)[100%]: Loss: -143.3994
Test(6): Loss: -133.7487
Train(7)[50%]: Loss: -144.9447
Train(7)[100%]: Loss: -145.4383
Test(7): Loss: -135.3678
Train(8)[50%]: Loss: -146.1320
Train(8)[100%]: Loss: -146.7899
Test(8): Loss: -136.2961
Train(9)[50%]: Loss: -147.7628
Train(9)[100%]: Loss: -148.1618
Test(9): Loss: -136.8142
Train(10)[50%]: Loss: -148.5743
Train(10)[100%]: Loss: -148.8958
Test(10): Loss: -138.1012
Train(11)[50%]: Loss: -149.4128
Train(11)[100%]: Loss: -149.6761
Test(11): Loss: -138.2358
Train(12)[50%]: Loss: -149.8672
Train(12)[100%]: Loss: -150.0321
Test(12): Loss: -138.7551
Train(13)[50%]: Loss: -150.5439
Train(13)[100%]: Loss: -150.6816
Test(13): Loss: -139.2322
```

```
Train(14)[50%]: Loss: -150.6672
Train(14)[100%]: Loss: -151.0996
Test(14): Loss: -138.9385
Train(15)[50%]: Loss: -151.2138
Train(15)[100%]: Loss: -151.3236
Test(15): Loss: -139.7993
Train(16)[50%]: Loss: -151.6756
Train(16)[100%]: Loss: -151.7711
Test(16): Loss: -140.0735
Train(17)[50%]: Loss: -152.2596
Train(17)[100%]: Loss: -152.0033
Test(17): Loss: -140.3256
Train(18)[50%]: Loss: -152.0391
Train(18)[100%]: Loss: -152.1508
Test(18): Loss: -140.5244
Train(19)[50%]: Loss: -152.1177
Train(19)[100%]: Loss: -152.4582
Test(19): Loss: -140.8739
Train(20)[50%]: Loss: -152.5973
Train(20)[100%]: Loss: -152.6743
Test(20): Loss: -140.6840
Train(21)[50%]: Loss: -152.7149
Train(21)[100%]: Loss: -152.6681
Test(21): Loss: -141.2114
Train(22)[50%]: Loss: -152.6525
Train(22)[100%]: Loss: -152.8391
Test(22): Loss: -141.1000
Train(23)[50%]: Loss: -153.1604
Train(23)[100%]: Loss: -153.1296
Test(23): Loss: -141.3030
Train(24)[50%]: Loss: -153.0999
Train(24)[100%]: Loss: -153.3150
Test(24): Loss: -141.8588
Train(25)[50%]: Loss: -153.2821
Train(25)[100%]: Loss: -153.4312
Test(25): Loss: -142.1384
Sample reconstructions on test set for M=16
```

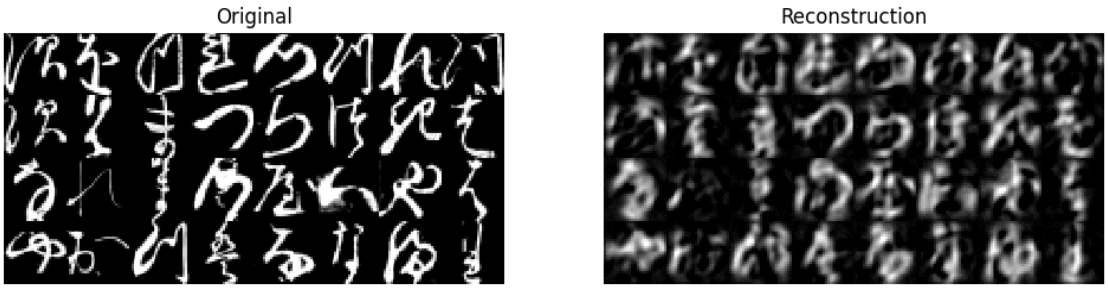


Sample reconstructions on train set for M=16




```
M=16: Train reconstruction MSE = 0.074537
M=16: Test  reconstruction MSE = 0.075677

============================================================
M = 64
Train(1)[50%]: Loss: -84.7132
Train(1)[100%]: Loss: -112.9077
Test(1): Loss: -142.7905
Train(2)[50%]: Loss: -162.6840
Train(2)[100%]: Loss: -168.8877
Test(2): Loss: -167.4687
Train(3)[50%]: Loss: -183.8675
Train(3)[100%]: Loss: -187.4649
Test(3): Loss: -181.4940
Train(4)[50%]: Loss: -196.3436
Train(4)[100%]: Loss: -198.3927
Test(4): Loss: -190.5410
Train(5)[50%]: Loss: -204.8335
Train(5)[100%]: Loss: -205.9769
Test(5): Loss: -196.4883
Train(6)[50%]: Loss: -209.9194
Train(6)[100%]: Loss: -211.1158
Test(6): Loss: -201.4049
Train(7)[50%]: Loss: -214.0671
Train(7)[100%]: Loss: -214.7995
Test(7): Loss: -204.5763
Train(8)[50%]: Loss: -217.2933
Train(8)[100%]: Loss: -217.9169
Test(8): Loss: -207.2948
Train(9)[50%]: Loss: -219.6699
Train(9)[100%]: Loss: -220.1426
Test(9): Loss: -209.2623
Train(10)[50%]: Loss: -221.6876
```

```
Train(10)[100%]: Loss: -222.0221
Test(10): Loss: -211.3252
Train(11)[50%]: Loss: -222.9737
Train(11)[100%]: Loss: -223.4287
Test(11): Loss: -212.9185
Train(12)[50%]: Loss: -224.3263
Train(12)[100%]: Loss: -224.7216
Test(12): Loss: -213.8104
Train(13)[50%]: Loss: -225.7758
Train(13)[100%]: Loss: -225.9939
Test(13): Loss: -214.8146
Train(14)[50%]: Loss: -226.7713
Train(14)[100%]: Loss: -227.0180
Test(14): Loss: -215.5557
Train(15)[50%]: Loss: -227.7376
Train(15)[100%]: Loss: -227.8618
Test(15): Loss: -216.9739
Train(16)[50%]: Loss: -228.5130
Train(16)[100%]: Loss: -228.5964
Test(16): Loss: -217.2990
Train(17)[50%]: Loss: -229.3933
Train(17)[100%]: Loss: -229.3316
Test(17): Loss: -218.5981
Train(18)[50%]: Loss: -229.8372
Train(18)[100%]: Loss: -229.9112
Test(18): Loss: -219.5454
Train(19)[50%]: Loss: -230.3896
Train(19)[100%]: Loss: -230.4914
Test(19): Loss: -220.0290
Train(20)[50%]: Loss: -231.1495
Train(20)[100%]: Loss: -231.3054
Test(20): Loss: -220.6215
Train(21)[50%]: Loss: -231.9496
Train(21)[100%]: Loss: -231.7726
Test(21): Loss: -221.1306
Train(22)[50%]: Loss: -232.2005
Train(22)[100%]: Loss: -232.2095
Test(22): Loss: -221.7639
Train(23)[50%]: Loss: -232.5549
Train(23)[100%]: Loss: -232.7096
Test(23): Loss: -222.5033
Train(24)[50%]: Loss: -233.2687
Train(24)[100%]: Loss: -233.2679
Test(24): Loss: -222.9062
Train(25)[50%]: Loss: -233.6306
Train(25)[100%]: Loss: -233.7370
Test(25): Loss: -223.0435
Sample reconstructions on test set for M=64
```

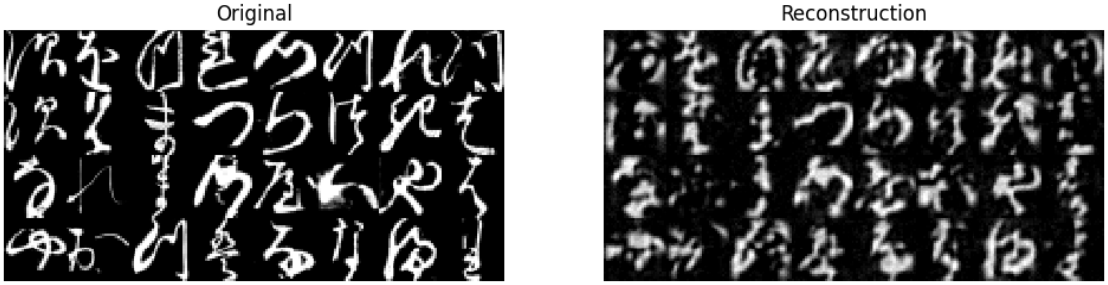


Sample reconstructions on train set for M=64

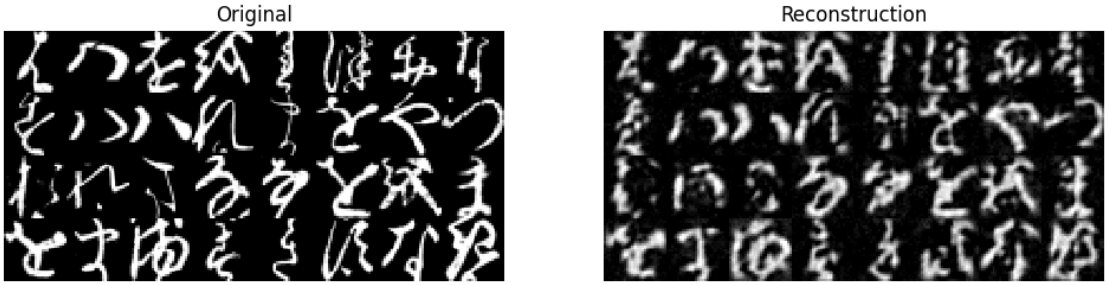

```
M=64: Train reconstruction MSE = 0.059141
M=64: Test  reconstruction MSE = 0.061102

============================================================
M = 256
Train(1)[50%]: Loss: -130.2417
Train(1)[100%]: Loss: -167.0051
Test(1): Loss: -208.0395
Train(2)[50%]: Loss: -229.0704
Train(2)[100%]: Loss: -235.5944
Test(2): Loss: -235.5848
Train(3)[50%]: Loss: -250.8172
Train(3)[100%]: Loss: -253.1058
Test(3): Loss: -247.7673
Train(4)[50%]: Loss: -260.2081
Train(4)[100%]: Loss: -261.7731
Test(4): Loss: -253.8400
Train(5)[50%]: Loss: -266.1993
Train(5)[100%]: Loss: -267.3963
Test(5): Loss: -259.8221
Train(6)[50%]: Loss: -270.9623
Train(6)[100%]: Loss: -271.7264
Test(6): Loss: -263.5669
```

```
Train(7)[50%]: Loss: -274.4663
Train(7)[100%]: Loss: -275.3056
Test(7): Loss: -267.1778
Train(8)[50%]: Loss: -277.4988
Train(8)[100%]: Loss: -278.3500
Test(8): Loss: -270.3924
Train(9)[50%]: Loss: -280.3278
Train(9)[100%]: Loss: -281.1329
Test(9): Loss: -272.9337
Train(10)[50%]: Loss: -283.3339
Train(10)[100%]: Loss: -283.8218
Test(10): Loss: -276.1666
Train(11)[50%]: Loss: -285.6375
Train(11)[100%]: Loss: -286.2558
Test(11): Loss: -278.4577
Train(12)[50%]: Loss: -287.9813
Train(12)[100%]: Loss: -288.5382
Test(12): Loss: -280.7099
Train(13)[50%]: Loss: -290.5384
Train(13)[100%]: Loss: -290.9056
Test(13): Loss: -283.2783
Train(14)[50%]: Loss: -292.3498
Train(14)[100%]: Loss: -292.7403
Test(14): Loss: -284.9780
Train(15)[50%]: Loss: -294.4642
Train(15)[100%]: Loss: -294.7155
Test(15): Loss: -287.3253
Train(16)[50%]: Loss: -295.9334
Train(16)[100%]: Loss: -296.3550
Test(16): Loss: -288.6040
Train(17)[50%]: Loss: -297.8585
Train(17)[100%]: Loss: -298.0873
Test(17): Loss: -290.2294
Train(18)[50%]: Loss: -299.4184
Train(18)[100%]: Loss: -299.8047
Test(18): Loss: -292.1462
Train(19)[50%]: Loss: -300.6133
Train(19)[100%]: Loss: -300.8603
Test(19): Loss: -293.6198
Train(20)[50%]: Loss: -301.9890
Train(20)[100%]: Loss: -302.3077
Test(20): Loss: -294.5501
Train(21)[50%]: Loss: -303.4178
Train(21)[100%]: Loss: -303.4515
Test(21): Loss: -296.6276
Train(22)[50%]: Loss: -304.4737
Train(22)[100%]: Loss: -304.4558
Test(22): Loss: -297.0279
```

```
Train(23)[50%]: Loss: -305.2873
Train(23)[100%]: Loss: -305.4996
Test(23): Loss: -298.1171
Train(24)[50%]: Loss: -306.0818
Train(24)[100%]: Loss: -306.2949
Test(24): Loss: -298.6795
Train(25)[50%]: Loss: -307.2251
Train(25)[100%]: Loss: -307.2296
Test(25): Loss: -300.3207
Sample reconstructions on test set for M=256
```




```
Sample reconstructions on train set for M=256
```




```
M=256: Train reconstruction MSE = 0.057000
M=256: Test  reconstruction MSE = 0.059685
```

Q3

November 23, 2025

1 Problem 3: Two Variational Autoencoders

1.1 3.1 Vanilla VAE

[8]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, transforms

if torch.cuda.is_available():
    device = torch.device("cuda")
elif getattr(torch.backends, "mps", None) is not None and torch.backends.mps.
 ↪is_available():
    device = torch.device("mps")
else:
    device = torch.device("cpu")

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

        self.fc1 = nn.Linear(n_in, n_hid)
        self.fc21 = nn.Linear(n_hid, z_dim)   # mu
        self.fc22 = nn.Linear(n_hid, z_dim)   # logvar
        self.fc3 = nn.Linear(z_dim, n_hid)
        self.fc4 = nn.Linear(n_hid, n_in)

    def encode(self, x, c=None):
        """Encoder forward pass: x -> (mu, logvar).  c is ignored for vanilla␣
 ↪VAE."""
        h = F.relu(self.fc1(x))
        mu = self.fc21(h)
        logvar = self.fc22(h)
        return mu, logvar

    def reparameterize(self, mu, logvar):
        """Implements: z = mu + epsilon * stdev, with epsilon ~ N(0, I)."""
```

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

    def decode(self, z, c=None):
        """Decoder forward pass: z -> reconstruction of x.  c is ignored."""
        h = F.relu(self.fc3(z))
        # FashionMNIST pixels are in [0,1], so we use sigmoid at the output
        x_recon = torch.sigmoid(self.fc4(h))
        return x_recon

    def forward(self, x, c=None):
        # x: (batch_size, 784)
        mu, logvar = self.encode(x, c)
        z = self.reparameterize(mu, logvar)
        x_recon = self.decode(z, c)
        return x_recon, mu, logvar
```

[2]:
```
n_in = 784
n_hid = 400
z_dim = 20
learning_rate = 1e-3
batch_size = 128
num_epochs = 20

# ------------------------
# Data: FashionMNIST + 20% validation split
# ------------------------
transform = transforms.Compose([
    transforms.ToTensor(),
])

full_train_dataset = datasets.FashionMNIST(
    root="./data",
    train=True,
    transform=transform,
    download=True,
)

test_dataset = datasets.FashionMNIST(
    root="./data",
    train=False,
    transform=transform,
    download=True,
)

train_size = int(0.8 * len(full_train_dataset))
```

```
val_size = len(full_train_dataset) - train_size
train_dataset, val_dataset = random_split(full_train_dataset, [train_size,␣
 ↪val_size])

train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

# ------------------------
# Loss: reconstruction + KL
# ------------------------
def vae_loss(recon_x, x, mu, logvar):
    # flatten to (batch, 784)
    x = x.view(x.size(0), -1)
    recon_x = recon_x.view(x.size(0), -1)

    # Binary cross-entropy reconstruction loss (sum over pixels)
    recon_loss = F.binary_cross_entropy(recon_x, x, reduction="sum")

    # KL divergence between q(z|x) and N(0,I)
    # 0.5 * sum(mu^2 + sigma^2 - log(sigma^2) - 1)
    kld = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())

    # average per data point (optional)
    return (recon_loss + kld) / x.size(0)

# ------------------------
# Training loop
# ------------------------
model = VAE(n_in, n_hid, z_dim).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

for epoch in range(1, num_epochs + 1):
    model.train()
    train_loss = 0.0
    for x, labels in train_loader:
        x = x.to(device).view(-1, n_in)  # flatten
        optimizer.zero_grad()
        recon_x, mu, logvar = model(x)   # c is ignored
        loss = vae_loss(recon_x, x, mu, logvar)
        loss.backward()
        optimizer.step()
        train_loss += loss.item() * x.size(0)

    train_loss /= len(train_loader.dataset)

    # Validation
```

```
    model.eval()
    val_loss = 0.0
    with torch.no_grad():
        for x, labels in val_loader:
            x = x.to(device).view(-1, n_in)
            recon_x, mu, logvar = model(x)
            loss = vae_loss(recon_x, x, mu, logvar)
            val_loss += loss.item() * x.size(0)
    val_loss /= len(val_loader.dataset)

    print(f"Epoch {epoch:03d} | train loss: {train_loss:.4f} | val loss:␣
 ↪{val_loss:.4f}")

torch.save(model.state_dict(), "vae_fashionmnist.pt")
print("Saved model weights to vae_fashionmnist.pt")
```

```
100%|      | 26.4M/26.4M [00:01<00:00, 13.9MB/s]
100%|      | 29.5k/29.5k [00:00<00:00, 253kB/s]
100%|      | 4.42M/4.42M [00:00<00:00, 4.84MB/s]
100%|      | 5.15k/5.15k [00:00<00:00, 17.9MB/s]
```

```
Epoch 001 | train loss: 290.3483 | val loss: 264.7819
Epoch 002 | train loss: 258.9247 | val loss: 256.3403
Epoch 003 | train loss: 252.8323 | val loss: 251.4031
Epoch 004 | train loss: 249.5057 | val loss: 248.9714
Epoch 005 | train loss: 247.5360 | val loss: 247.2198
Epoch 006 | train loss: 246.2268 | val loss: 246.2897
Epoch 007 | train loss: 245.2569 | val loss: 245.2917
Epoch 008 | train loss: 244.5115 | val loss: 245.0276
Epoch 009 | train loss: 243.9698 | val loss: 244.3799
Epoch 010 | train loss: 243.5142 | val loss: 244.1185
Epoch 011 | train loss: 243.1693 | val loss: 243.6222
Epoch 012 | train loss: 242.8897 | val loss: 243.4241
Epoch 013 | train loss: 242.5679 | val loss: 243.2843
Epoch 014 | train loss: 242.3468 | val loss: 243.0216
Epoch 015 | train loss: 242.0210 | val loss: 242.7011
Epoch 016 | train loss: 241.8006 | val loss: 242.5665
Epoch 017 | train loss: 241.6295 | val loss: 242.2854
Epoch 018 | train loss: 241.4636 | val loss: 242.1922
Epoch 019 | train loss: 241.2744 | val loss: 242.1374
Epoch 020 | train loss: 241.1012 | val loss: 241.9481
Saved model weights to vae_fashionmnist.pt
```

1.2 3.2 C-VAE

[3]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, transforms

if torch.cuda.is_available():
    device = torch.device("cuda")
elif getattr(torch.backends, "mps", None) is not None and torch.backends.mps.
 ↪is_available():
    device = torch.device("mps")
else:
    device = torch.device("cpu")

class CVAE(nn.Module):
    def __init__(self, n_in, n_hid, z_dim, n_classes):
        super().__init__()
        self.n_in = n_in
        self.n_classes = n_classes

        # encoder: [x, one_hot(c)] -> h -> mu, logvar
        self.fc1 = nn.Linear(n_in + n_classes, n_hid)
        self.fc21 = nn.Linear(n_hid, z_dim)
        self.fc22 = nn.Linear(n_hid, z_dim)

        # decoder: [z, one_hot(c)] -> h -> x_hat
        self.fc3 = nn.Linear(z_dim + n_classes, n_hid)
        self.fc4 = nn.Linear(n_hid, n_in)

    def encode(self, x, c):
        """
        Encoder forward pass.
        x: (B, 784)
        c: (B,) long labels
        """
        one_hot = F.one_hot(c, num_classes=self.n_classes).float()
        inp = torch.cat([x, one_hot], dim=1)
        h = F.relu(self.fc1(inp))
        mu = self.fc21(h)
        logvar = self.fc22(h)
        return mu, logvar

    def reparameterize(self, mu, logvar):
        """Implements: z = mu + epsilon * stdev."""
        std = torch.exp(0.5 * logvar)
```

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

    def decode(self, z, c):
        """
        Decoder forward pass.
        z: (B, z_dim)
        c: (B,) long labels
        """
        one_hot = F.one_hot(c, num_classes=self.n_classes).float()
        inp = torch.cat([z, one_hot], dim=1)
        h = F.relu(self.fc3(inp))
        x_recon = torch.sigmoid(self.fc4(h))  # outputs in [0,1]
        return x_recon

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

[4]:
```
n_in = 784
n_hid = 400
z_dim = 20
n_classes = 10
learning_rate = 1e-3
batch_size = 128
num_epochs = 20

transform = transforms.Compose([
    transforms.ToTensor(),
])

full_train_dataset = datasets.FashionMNIST(
    root="./data",
    train=True,
    transform=transform,
    download=True,
)

test_dataset = datasets.FashionMNIST(
    root="./data",
    train=False,
    transform=transform,
    download=True,
)
```

```
train_size = int(0.8 * len(full_train_dataset))
val_size = len(full_train_dataset) - train_size
train_dataset, val_dataset = random_split(full_train_dataset, [train_size,␣
 ↪val_size])

train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

def vae_loss(recon_x, x, mu, logvar):
    x = x.view(x.size(0), -1)
    recon_x = recon_x.view(x.size(0), -1)

    recon_loss = F.binary_cross_entropy(recon_x, x, reduction="sum")
    kld = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    return (recon_loss + kld) / x.size(0)
```

[5]:
```
model = CVAE(n_in, n_hid, z_dim, n_classes).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

for epoch in range(1, num_epochs + 1):
    model.train()
    train_loss = 0.0
    for x, labels in train_loader:
        x = x.to(device).view(-1, n_in)
        labels = labels.to(device)

        optimizer.zero_grad()
        recon_x, mu, logvar = model(x, labels)
        loss = vae_loss(recon_x, x, mu, logvar)
        loss.backward()
        optimizer.step()

        train_loss += loss.item() * x.size(0)
    train_loss /= len(train_loader.dataset)

    # validation
    model.eval()
    val_loss = 0.0
    with torch.no_grad():
        for x, labels in val_loader:
            x = x.to(device).view(-1, n_in)
            labels = labels.to(device)
            recon_x, mu, logvar = model(x, labels)
            loss = vae_loss(recon_x, x, mu, logvar)
            val_loss += loss.item() * x.size(0)
    val_loss /= len(val_loader.dataset)
```

```
    print(f"Epoch {epoch:03d} | train loss: {train_loss:.4f} | val loss:␣
 ↪{val_loss:.4f}")

# ------------------------
# Save C-VAE weights for later use (required by 3.2)
# ------------------------
torch.save(model.state_dict(), "cvae_fashionmnist.pt")
print("Saved model weights to cvae_fashionmnist.pt")
```

```
Epoch 001 | train loss: 290.3016 | val loss: 262.4978
Epoch 002 | train loss: 257.8125 | val loss: 252.6125
Epoch 003 | train loss: 250.9448 | val loss: 247.8219
Epoch 004 | train loss: 247.2942 | val loss: 244.8680
Epoch 005 | train loss: 245.1321 | val loss: 243.2835
Epoch 006 | train loss: 243.6201 | val loss: 242.0301
Epoch 007 | train loss: 242.4977 | val loss: 241.1477
Epoch 008 | train loss: 241.6783 | val loss: 240.3919
Epoch 009 | train loss: 241.0254 | val loss: 239.9772
Epoch 010 | train loss: 240.4261 | val loss: 239.5868
Epoch 011 | train loss: 239.8918 | val loss: 239.0188
Epoch 012 | train loss: 239.5168 | val loss: 238.7477
Epoch 013 | train loss: 239.0728 | val loss: 238.2041
Epoch 014 | train loss: 238.8118 | val loss: 238.0087
Epoch 015 | train loss: 238.4650 | val loss: 237.5201
Epoch 016 | train loss: 238.2286 | val loss: 237.4653
Epoch 017 | train loss: 238.0273 | val loss: 237.4298
Epoch 018 | train loss: 237.8721 | val loss: 237.0821
Epoch 019 | train loss: 237.6228 | val loss: 236.6978
Epoch 020 | train loss: 237.4128 | val loss: 236.8433
Saved model weights to cvae_fashionmnist.pt
```

1.3 3.3 Manifold Comparison

[ ]:
```
import torch
import torch.nn.functional as F
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import numpy as np

if torch.cuda.is_available():
    device = torch.device("cuda")
elif getattr(torch.backends, "mps", None) is not None and torch.backends.mps.
 ↪is_available():
    device = torch.device("mps")
```

```
else:
    device = torch.device("cpu")

n_in = 784
n_hid = 400
z_dim = 20
n_classes = 10

# ------------------------------------------------
# Load test set
# ------------------------------------------------
transform = transforms.Compose([
    transforms.ToTensor(),
])

test_dataset = datasets.FashionMNIST(
    root="./data",
    train=False,
    transform=transform,
    download=True,
)

test_loader = DataLoader(test_dataset, batch_size=256, shuffle=False)

# ------------------------------------------------
# Load trained models
# ------------------------------------------------
vae = VAE(n_in, n_hid, z_dim).to(device)
vae.load_state_dict(torch.load("vae_fashionmnist.pt", map_location=device))
vae.eval()

cvae = CVAE(n_in, n_hid, z_dim, n_classes).to(device)
cvae.load_state_dict(torch.load("cvae_fashionmnist.pt", map_location=device))
cvae.eval()

# ------------------------------------------------
# Collect mu for all test samples
# ------------------------------------------------
all_mu_vae = []
all_mu_cvae = []
all_labels = []

with torch.no_grad():
    for x, y in test_loader:
        x = x.to(device).view(-1, n_in)
        y = y.to(device)
```

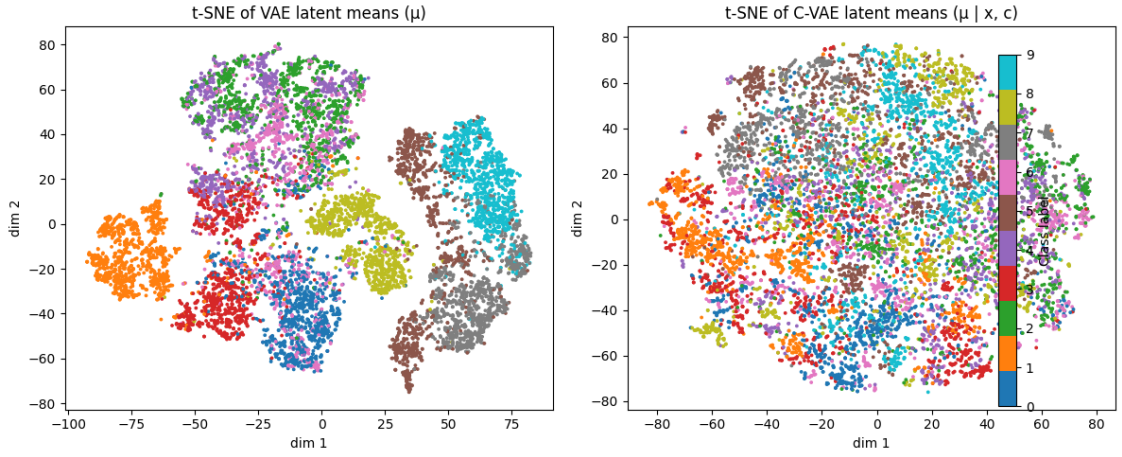
```
        # VAE
        mu_vae, logvar_vae = vae.encode(x)              # c not used
        # C-VAE
        mu_cvae, logvar_cvae = cvae.encode(x, y)

        all_mu_vae.append(mu_vae.cpu())
        all_mu_cvae.append(mu_cvae.cpu())
        all_labels.append(y.cpu())

all_mu_vae = torch.cat(all_mu_vae, dim=0).numpy()
all_mu_cvae = torch.cat(all_mu_cvae, dim=0).numpy()
all_labels = torch.cat(all_labels, dim=0).numpy()

print("Shapes:",
      all_mu_vae.shape, all_mu_cvae.shape, all_labels.shape)
```

```
Shapes: (10000, 20) (10000, 20) (10000,)
```

[11]:
```
# ------------------------------------------------
# t-SNE on mu (separately for VAE and C-VAE)
# ------------------------------------------------
tsne = TSNE(
    n_components=2,
    random_state=42,
)
mu2d_vae = tsne.fit_transform(all_mu_vae)

# For a fair comparison, make a new TSNE instance for cVAE
tsne_c = TSNE(
    n_components=2,
    random_state=42,
)
mu2d_cvae = tsne_c.fit_transform(all_mu_cvae)

# ------------------------------------------------
# Plot and save figures
# ------------------------------------------------
plt.figure(figsize=(12, 5))

# VAE
plt.subplot(1, 2, 1)
scatter1 = plt.scatter(mu2d_vae[:, 0], mu2d_vae[:, 1],
                       c=all_labels, s=3, cmap="tab10")
plt.title("t-SNE of VAE latent means ( )")
plt.xlabel("dim 1")
plt.ylabel("dim 2")
```

```
# C-VAE
plt.subplot(1, 2, 2)
scatter2 = plt.scatter(mu2d_cvae[:, 0], mu2d_cvae[:, 1],
                       c=all_labels, s=3, cmap="tab10")
plt.title("t-SNE of C-VAE latent means (  | x, c)")
plt.xlabel("dim 1")
plt.ylabel("dim 2")

# Common legend
cbar = plt.colorbar(scatter2, ax=plt.gcf().axes, fraction=0.02, pad=0.04)
cbar.set_label("Class label")

plt.tight_layout()
plt.savefig("tsne_vae_vs_cvae_fashionmnist.png", dpi=300)
plt.show()
```

```
/var/folders/w5/_y5qsxd54h7g1_bgp2wv659h0000gn/T/ipykernel_73938/3398934085.py:4
2: UserWarning: This figure includes Axes that are not compatible with
tight_layout, so results might be incorrect.
  plt.tight_layout()
```



1.4 3.3 Answer:

To compare the latent representations learned by the VAE and C-VAE, I passed all Fashion-MNIST test samples through each encoder and collected the posterior means . I then applied t-SNE to project the 20-dimensional latent means into a 2-dimensional manifold and colored the points by class label. The resulting visualizations show clear and consistent differences between the two models.

VAE manifold (left figure). The vanilla VAE produces a latent space where the data points form distinct, well-separated clusters, each corresponding to a different Fashion-MNIST class. Classes such as sneakers, boots, pullovers, and coats occupy their own regions. This separation occurs

because the VAE has no access to the class label. Therefore, in order to reconstruct the input effectively, the encoder is forced to encode class-related information into the latent variable . As a result, class identity becomes one of the dominant organizing factors of the latent space, leading to a structured and interpretable clustering pattern.

C-VAE manifold (right figure). In contrast, the latent space learned by the C-VAE is much more entangled. The clusters almost completely overlap, and no clear class-dependent structure appears. This is expected because the decoder receives the class label explicitly, so the latent variable no longer needs to encode information about the object category. Instead, only captures residual variations within each class (e.g., style, orientation, thickness), while the class identity is handled separately by the conditioning variable. This reduces the pressure on the encoder to organize data by class, and the resulting latent manifold appears more "mixed" and dispersed.

Hypothesis. The difference arises because the VAE must use to encode all information relevant to reconstructing , including class identity. The C-VAE, however, offloads class information to the conditioning variable , allowing to focus on intra-class variations. Consequently, the VAE forms class-separable clusters, whereas the C-VAE forms a more overlapping latent distribution.