# ECE 685D HW4

Zanwen Fu (zf93)

Nov 7 2025

## Submission Instructions:

1. Upload your Jupyter Notebook (.ipynb file).
2. Export all outputs and necessary derivations as a PDF or HTML (preferred) and upload them.

### LLM policy:

The use of large language models (LLMs) is allowed for this assignment, but if you use them, you **must disclose how.** ChatGPT was used to evaluate the answers.

## 1 Problem 1: Sparse Encoding for Denoising (25 pts)

Consider an auto-encoding like this:

$$\min_{D} \frac{1}{T} \sum_{t=1}^{T} \min_{h^{(t)}} \frac{1}{2} \|x^{(t)} - Dh^{(t)}\|_2^2 + \lambda \|h^{(t)}\|_1.$$


Figure 1: Over-complete auto-encoder with non-linear encoding and linear decoding modules. LASSO ($L_1$) regularization is used to enforce meaningful feature learning.

We will use MNIST dataset. Let the dimension of $h$ be 1.5 times that of your input and $f(x)$ defined by your own conjecture. Let X be a batch sampled from MNIST, taking $X + \mathcal{N}(0, \sigma^2 I)$ ($\sigma^2$ of your choice, and you can clip the pixrls of noisy images between 0 and 1 to stabilize training) as the model input, and our goal is to obtain its output $\widehat{X} \approx X$. Use the default training split for your auto-encoder. You should plot

(a) 5 input-output pairs $(X + \mathcal{N}(0, \sigma^2 I), \widehat{X})$ using the data in the test set.
**Answer: Please refer to the jupyter notebook below**

(b) The top 5 dictionary vectors in $D$ whose corresponding intensity $|h_i|$ are the largest.
**Answer: Please refer to the jupyter notebook below**

(c) Plot the mean square error (MSE) between noiseless image $X$ and predicted image $\widehat{X}$ as a function of the noise level $\sigma^2$. You can do this for about 5 values of $\sigma^2$ between 0 and 1.

**Answer: Please refer to the jupyter notebook below**

Hint: Using MLP for this may make the visualization easier.

# 2 Problem 2: Probabilistic PCA (35pts)

Let $x_1, \cdots, x_N \in \mathbb{R}^d$ be $N$ independent observations from

$$x_j = \underbrace{Wz_j + \mu}_{\text{signal}} + \underset{\text{noise}}{\epsilon_j}, \quad \text{where} \quad \begin{bmatrix} z_j \\ \epsilon_j \end{bmatrix} \overset{\text{i.i.d.}}{\sim} \mathcal{N}\left(0, \begin{bmatrix} I_q & 0 \\ 0 & \sigma^2 I_d \end{bmatrix}\right).$$

That is, $(z_j)$ are from the Gaussian distribution $\mathcal{N}(\mu, C)$, where the covariance matrix is $C = WW^\top + \sigma^2 I_d$. We target to estimate the parameters

$$\mu \in \mathbb{R}^d, \quad W \in \mathbb{R}^{d\times q}, \quad \text{and} \quad \sigma \in (0, \infty)$$

by maximum likelihood estimation (MLE). Here the dimension $q < \min\{d, N\}$ is known.

(a) (5pts) Write the likelihood function $\mathcal{L}(\mu, W, \sigma^2)$ given $x_1, \cdots, x_N$, and derive the MLE $\widehat{\mu}$ of $\mu$.

**Answer(a)** Given that each observation $x_j \in \mathbb{R}^d$ follows

$$x_j \sim \mathcal{N}(\mu, C), \quad \text{where } C = WW^\top + \sigma^2 I_d,$$

and assuming independence across $N$ samples, the likelihood function is

$$\mathcal{L}(\mu, W, \sigma^2) = \prod_{j=1}^{N} \frac{1}{(2\pi)^{d/2}|C|^{1/2}} \exp\left(-\frac{1}{2}(x_j - \mu)^\top C^{-1}(x_j - \mu)\right).$$

Taking the logarithm gives the log-likelihood (up to additive constants):

$$\ell(\mu, W, \sigma^2) = -\frac{N}{2}\log|C| - \frac{1}{2}\sum_{j=1}^{N}(x_j - \mu)^\top C^{-1}(x_j - \mu).$$

To find the MLE of $\mu$, differentiate $\ell$ with respect to $\mu$ and set to zero:

$$\frac{\partial \ell}{\partial \mu} = C^{-1}\left(\sum_{j=1}^{N} x_j - N\mu\right) = 0.$$

Hence, the maximum likelihood estimator of $\mu$ is

$$\boxed{\hat{\mu} = \frac{1}{N}\sum_{j=1}^{N} x_j.}$$

(b) (10pts) Define the sample covariance matrix of $x_1, \cdots, x_N$ as

$$S = \frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})(x_i - \overline{x})^\top,$$

where $\overline{x} = N^{-1}\sum_{i=1}^{N} x_i \in \mathbb{R}^d$ is the mean of $x_1, \cdots, x_N$. Prove that $\mathcal{L}(\mu, W, \sigma^2)$ is minimized only if

$$C^{-1}W = C^{-1}SC^{-1}W.$$

*Remark.* You may use the following fact: for any positive definite matrices $A, X \in \mathbb{R}^{d\times d}$,

$$\frac{\mathrm{d}}{\mathrm{d}X}\log\det(X) = X^{-1}, \quad \text{and} \quad \frac{\mathrm{d}}{\mathrm{d}X}\mathrm{Tr}(AX^{-1}) = -X^{-1}AX^{-1}.$$

**Answer (b)** Using the result from part (a), the MLE of $\mu$ is

$$\hat{\mu} = \bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i.$$

Substituting this into the log-likelihood (and omitting additive constants), we obtain

$$\mathcal{L}(\mu, W, \sigma^2) \equiv \mathcal{L}(W, \sigma^2) = \frac{N}{2}\left(\log|C| + \mathrm{Tr}(C^{-1}S)\right),$$

where

$$C = WW^\top + \sigma^2 I_d, \qquad S = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})(x_i - \bar{x})^\top.$$

To find the stationary point with respect to $W$, we compute the differential:

$$d\mathcal{L} = \frac{N}{2}\Big(\mathrm{Tr}(C^{-1}dC) + \mathrm{Tr}(S\, d(C^{-1}))\Big).$$

Using the given matrix calculus facts

$$\frac{d}{dX}\log\det(X) = X^{-1}, \qquad \frac{d}{dX}\mathrm{Tr}(AX^{-1}) = -X^{-1}AX^{-1},$$

we have

$$d\mathcal{L} = \frac{N}{2}\mathrm{Tr}\big[(C^{-1} - C^{-1}SC^{-1})\, dC\big].$$

Since $C = WW^\top + \sigma^2 I_d$, its differential is

$$dC = dW\, W^\top + W\, dW^\top.$$

Substituting this into the above expression gives

$$\begin{aligned} d\mathcal{L} &= \frac{N}{2}\mathrm{Tr}\big[(C^{-1} - C^{-1}SC^{-1})(dW\, W^\top + W\, dW^\top)\big] \\ &= N\,\mathrm{Tr}\Big[\big((C^{-1} - C^{-1}SC^{-1})W\big)^\top dW\Big]. \end{aligned}$$

Therefore, the gradient with respect to $W$ is

$$\frac{\partial\mathcal{L}}{\partial W} = N\big(C^{-1} - C^{-1}SC^{-1}\big)W.$$

Setting this derivative to zero gives the first-order condition:

$$\big(C^{-1} - C^{-1}SC^{-1}\big)W = 0,$$

which simplifies to

$$C^{-1}W = C^{-1}SC^{-1}W.$$

Hence, we have shown that $\mathcal{L}(\mu, W, \sigma^2)$ is minimized only if

$$\boxed{C^{-1}W = C^{-1}SC^{-1}W.}$$

(c) (10pts) Assume $W$ is a minimizer, and $W = QDV^\top$ the compact SVD of $W$, i.e. $Q \in \mathbb{R}^{d\times q}, V \in \mathbb{R}^{q\times q}$ are two column orthogonal matrices, and $D \in \mathbb{R}^{q\times q}$ is a nonnegative diagonal matrix. Show that

$$SQ = Q(D^2 + \sigma^2 I_d).$$

Using this formula to argue that all potential minimizer $W$ has the form of

$$W = U_q \left(\Lambda_q - \sigma^2 I_d\right)^{1/2} R,$$

where the columns of $U_q$ are $q$ distinct eigenvectors of $S$, $\Lambda$ is a diagonal matrix with corresponding eigenvalues and $R \in \mathbb{R}^{q\times q}$ is any orthogonal matrix.

**Answer (c)** From part (b), we have the first-order condition

$$C^{-1}W = C^{-1}SC^{-1}W.$$

Multiplying both sides by $C$ gives the equivalent expression

$$W = SC^{-1}W. \tag{1}$$

Let the compact SVD of $W$ be

$$W = QDV^\top,$$

where $Q \in \mathbb{R}^{d\times q}$ and $V \in \mathbb{R}^{q\times q}$ are column-orthonormal matrices, and $D = \mathrm{diag}(d_1, \ldots, d_q)$ is a nonnegative diagonal matrix.

Then,

$$WW^\top = QD^2Q^\top,$$

and hence the covariance matrix

$$C = WW^\top + \sigma^2 I_d = Q(D^2 + \sigma^2 I_q)Q^\top + \sigma^2(I_d - QQ^\top).$$

The inverse of $C$ is therefore

$$C^{-1} = Q(D^2 + \sigma^2 I_q)^{-1}Q^\top + \frac{1}{\sigma^2}(I_d - QQ^\top).$$

Now compute $C^{-1}W$:

$$\begin{aligned} C^{-1}W &= \left[Q(D^2 + \sigma^2 I_q)^{-1}Q^\top + \tfrac{1}{\sigma^2}(I_d - QQ^\top)\right]QDV^\top \\ &= Q(D^2 + \sigma^2 I_q)^{-1}Q^\top QDV^\top \quad (\text{since } (I_d - QQ^\top)Q = 0) \\ &= Q(D^2 + \sigma^2 I_q)^{-1}DV^\top. \end{aligned}$$

Substitute this into equation (1):

$$QDV^\top = S\,Q(D^2 + \sigma^2 I_q)^{-1} DV^\top.$$

Right-multiplying by $V$ and cancelling $D$ (for nonzero singular values) gives

$$Q = SQ(D^2 + \sigma^2 I_q)^{-1},$$

which implies

$$SQ = Q(D^2 + \sigma^2 I_q). \tag{2}$$

Equation (2) shows that the columns of $Q$ are eigenvectors of $S$ with corresponding eigenvalues

$$\lambda_i = d_i^2 + \sigma^2.$$

Let the eigen-decomposition of $S$ be $S = U\Lambda U^\top$, and let $U_q$ denote the $q$ eigenvectors associated with the $q$ largest eigenvalues $\Lambda_q = \mathrm{diag}(\lambda_1, \ldots, \lambda_q)$.
Then, from $\lambda_i = d_i^2 + \sigma^2$, we have

$$D^2 = \Lambda_q - \sigma^2 I_q, \qquad D = (\Lambda_q - \sigma^2 I_q)^{1/2}.$$

Since $W = QDV^\top$, and $Q = U_q$, any minimizer must have the form

$$\boxed{W = U_q \left(\Lambda_q - \sigma^2 I_q\right)^{1/2} R,}$$

where $R = V^\top$ is an arbitrary orthogonal $q \times q$ matrix.

(d) (10pts) Show that the MLE of $\sigma^2$ is given by

$$\widehat{\sigma}^2 = \frac{1}{d-q} \sum_{j=q+1}^{d} \lambda_j(S), \tag{2.1}$$

where $\lambda_1(S) \geq \lambda_2(S) \geq \cdots \geq \lambda_d(S)$ are eigenvalues of $S$, including repetitions according to algebraic multiplicity and sorted in decreasing order. Derive the MLE $\widehat{W}$ of $W$ using (c).
**Answer (d)** Using the result from part (c), the eigen-decomposition of the sample covariance matrix $S$ is

$$S = U\Lambda U^\top, \qquad \Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_d), \quad \lambda_1 \geq \cdots \geq \lambda_d.$$

Any maximum likelihood estimator (MLE) of $W$ has the form

$$\hat{W} = U_q(\Lambda_q - \sigma^2 I_q)^{1/2} R,$$

where $U_q$ contains the $q$ principal eigenvectors of $S$, $\Lambda_q = \mathrm{diag}(\lambda_1, \ldots, \lambda_q)$, and $R$ is an arbitrary orthogonal matrix.

With this $\hat{W}$, we have

$$\hat{W}\hat{W}^\top = U_q(\Lambda_q - \sigma^2 I_q)U_q^\top,$$

so

$$C = \hat{W}\hat{W}^\top + \sigma^2 I_d = U_q \Lambda_q U_q^\top + \sigma^2(I_d - U_q U_q^\top).$$

Thus, $C$ and $S$ share the same eigenvectors, and the eigenvalues of $C$ are

$$c_i = \begin{cases} \lambda_i, & i = 1, \ldots, q, \\ \sigma^2, & i = q+1, \ldots, d. \end{cases}$$

From part (b), the log-likelihood (up to additive constants) is

$$\mathcal{L}(\sigma^2) = \frac{N}{2}\Big( \log|C| + \mathrm{Tr}(C^{-1}S) \Big).$$

Using the eigenvalues of $C$, we have

$$\log|C| = \sum_{i=1}^{q} \log \lambda_i + (d-q)\log \sigma^2,$$

and

$$\mathrm{Tr}(C^{-1}S) = \sum_{i=1}^{d} \frac{\lambda_i}{c_i} = \sum_{i=1}^{q} \frac{\lambda_i}{\lambda_i} + \sum_{i=q+1}^{d} \frac{\lambda_i}{\sigma^2} = q + \frac{1}{\sigma^2} \sum_{i=q+1}^{d} \lambda_i.$$

Hence (ignoring constants independent of $\sigma^2$),

$$\mathcal{L}(\sigma^2) = \frac{N}{2} \left[ (d-q)\log \sigma^2 + \frac{1}{\sigma^2} \sum_{i=q+1}^{d} \lambda_i \right].$$

Differentiate with respect to $\sigma^2$:

$$\frac{d\mathcal{L}}{d\sigma^2} = \frac{N}{2} \left[ \frac{d-q}{\sigma^2} - \frac{1}{(\sigma^2)^2} \sum_{i=q+1}^{d} \lambda_i \right].$$

Setting this derivative to zero yields

$$(d-q)\sigma^2 = \sum_{i=q+1}^{d} \lambda_i,$$

so the MLE of $\sigma^2$ is

$$\boxed{\hat{\sigma}^2 = \frac{1}{d-q} \sum_{j=q+1}^{d} \lambda_j(S).}$$

This estimator corresponds to the average of the smallest $(d-q)$ eigenvalues of the sample covariance matrix $S$.

# 3 Problem 3: Gaussian-Bernoulli Restricted Boltzmann Machines (40pts)

Let the energy function of Gaussian-Bernoulli RBM take the form

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

(a) Use the definition $p(v,h) = e^{-E(x,h)}/Z$ to derive $p(v_i = x|h)$ and $p(h_j = 1|v)$. You may leave $p(v_i = x|h)$ in an integral form once you cancel the terms involving $\sum_j a_j h_j$.

**Answer (a)**

Given the energy function of the Gaussian–Bernoulli RBM:

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

the joint probability is defined as:

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

1. Derivation of $p(v_i = x \mid h)$:

Conditioning on $h$, the term $\sum_j \alpha_j h_j$ is constant with respect to $v$ and can be dropped. The joint over $v$ given $h$ factorizes as:

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

Hence, for each visible unit $v_i$:

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

Recognizing this as a Gaussian distribution, we obtain:

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

(Equivalently, one may leave it in the unnormalized exponential or integral form above.)

2. Derivation of $p(h_j = 1 \mid v)$:

Now fix $v$. The relevant part of the exponent involving $h$ is:

$$\log p(v,h) = \sum_j h_j \left(\alpha_j + \sum_i W_{ij} \frac{v_i}{\sigma_i}\right) + (\text{terms independent of } h).$$

Thus the conditional over $h$ factorizes as:

$$p(h \mid v) = \prod_j p(h_j \mid v),$$

and each hidden unit follows a Bernoulli distribution:

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

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the logistic sigmoid function.

(b) Train a Gaussian-Bernouilli RBM over the Fashion MNIST dataset using contrastive divergence minimization (see the lecture notes). Use the standard training and testing split available in Pytorch. Use learning rate 0.001 and batch size 128. Train the model over 25 epochs for $M = \{10, 50, 100, 250\}$, where $M$ is the dimension of the hidden weights $W$ and report the mean squared reconstruction error for the test dataset. A template container Bernoulli-Bernoulli has been provided to you. You may use it as a start.

**Answer: Please refer to the jupyter notebook below**

# Q1

November 7, 2025

## 0.1 Q1 Sparse Encoding for Denoising

[9]:
```
import random, numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
import matplotlib.pyplot as plt

if torch.backends.mps.is_available():
    device = torch.device("mps")
else:
    device = torch.device("cpu")
seed = 1337
torch.manual_seed(seed); np.random.seed(seed); random.seed(seed)

# Hyperparameters
IMG_H, IMG_W = 28, 28
INPUT_DIM = IMG_H * IMG_W
H_DIM = int(1.5 * INPUT_DIM)
LAMBDA_L1 = 1e-3               # L1 on codes
LR = 1e-3
BATCH_SIZE = 128
EPOCHS = 10
TRAIN_SIGMA = 0.3

# Data
tfm = transforms.ToTensor()  # maps to [0,1]
train_ds = datasets.MNIST(root="./data", train=True, transform=tfm,␣
 ↪download=True)
test_ds  = datasets.MNIST(root="./data", train=False, transform=tfm,␣
 ↪download=True)
train_loader = DataLoader(train_ds, batch_size=BATCH_SIZE, shuffle=True,␣
 ↪num_workers=2, pin_memory=True)
test_loader  = DataLoader(test_ds,  batch_size=BATCH_SIZE, shuffle=False,␣
 ↪num_workers=2, pin_memory=True)
```

```
def add_noise(x, sigma):
    if sigma == 0:
        return x
    z = x + sigma * torch.randn_like(x)
    return torch.clamp(z, 0.0, 1.0)
```

[10]:
```
class Encoder(nn.Module):
    def __init__(self, in_dim=INPUT_DIM, h_dim=H_DIM):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(in_dim, 1024),
            nn.ReLU(inplace=True),
            nn.Linear(1024, h_dim),
        )
    def forward(self, x_flat):
        return self.net(x_flat)

class LinearDecoder(nn.Module):
    """
    Explicit dictionary D: x_hat = h @ D
    D has shape [h_dim, in_dim]. (Each row is a 'dictionary atom' reshaped to␣
 ↪28x28.)
    """
    def __init__(self, h_dim=H_DIM, in_dim=INPUT_DIM):
        super().__init__()
        # Initialize near-orthogonal for stability
        self.D = nn.Parameter(torch.empty(h_dim, in_dim))
        nn.init.xavier_uniform_(self.D)

    def forward(self, h):
        # h: [B, h_dim]  ->  x_hat_flat: [B, in_dim]
        return h @ self.D

class SparseDenoisingAE(nn.Module):
    def __init__(self):
        super().__init__()
        self.enc = Encoder(INPUT_DIM, H_DIM)
        self.dec = LinearDecoder(H_DIM, INPUT_DIM)
    def forward(self, x):
        # x: [B,1,28,28] -> flatten
        x_flat = x.view(x.size(0), -1)
        h = self.enc(x_flat)                    # nonlinear encoding
        x_hat_flat = self.dec(h)                # explicit linear decoding
        x_hat = x_hat_flat.view(-1, 1, 28, 28)
        return x_hat, h

model = SparseDenoisingAE().to(device)
```

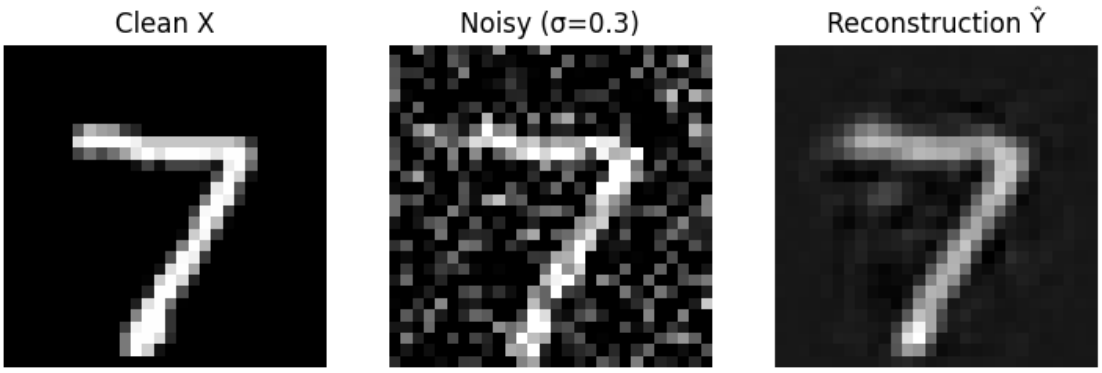
```
opt = torch.optim.Adam(model.parameters(), lr=LR)
```

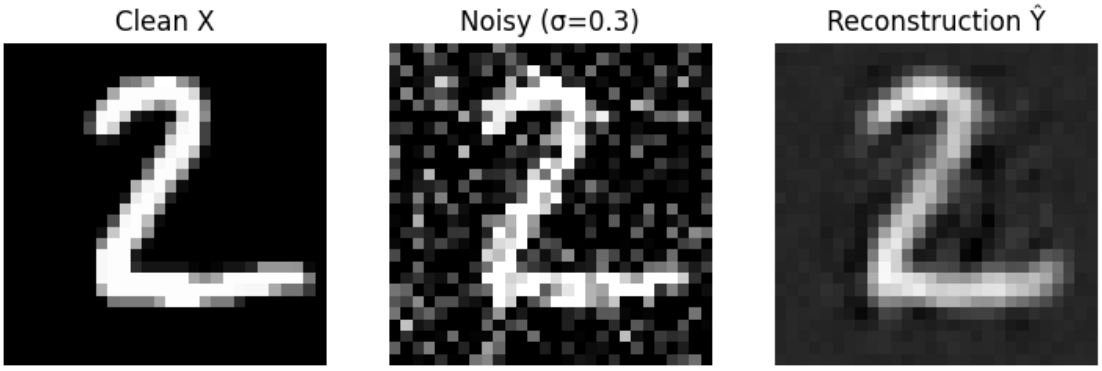
[11]:
```
def mse_loss(a, b):
    return F.mse_loss(a, b, reduction="mean")

history = {"train_loss": []}

model.train()
for epoch in range(1, EPOCHS+1):
    running = 0.0
    for x, _ in train_loader:
        x = x.to(device)
        x_noisy = add_noise(x, TRAIN_SIGMA)

        x_hat, h = model(x_noisy)

        recon = 0.5 * mse_loss(x_hat, x)              # 1/2 ||x - D h||^2
        l1 = LAMBDA_L1 * h.abs().mean()               #   ||h||_1 (mean over␣
 ↪batch for scale)
        loss = recon + l1

        opt.zero_grad()
        loss.backward()
        opt.step()

        running += loss.item() * x.size(0)

    epoch_loss = running / len(train_loader.dataset)
    history["train_loss"].append(epoch_loss)
    print(f"Epoch {epoch:02d}/{EPOCHS}  loss={epoch_loss:.6f}")
```

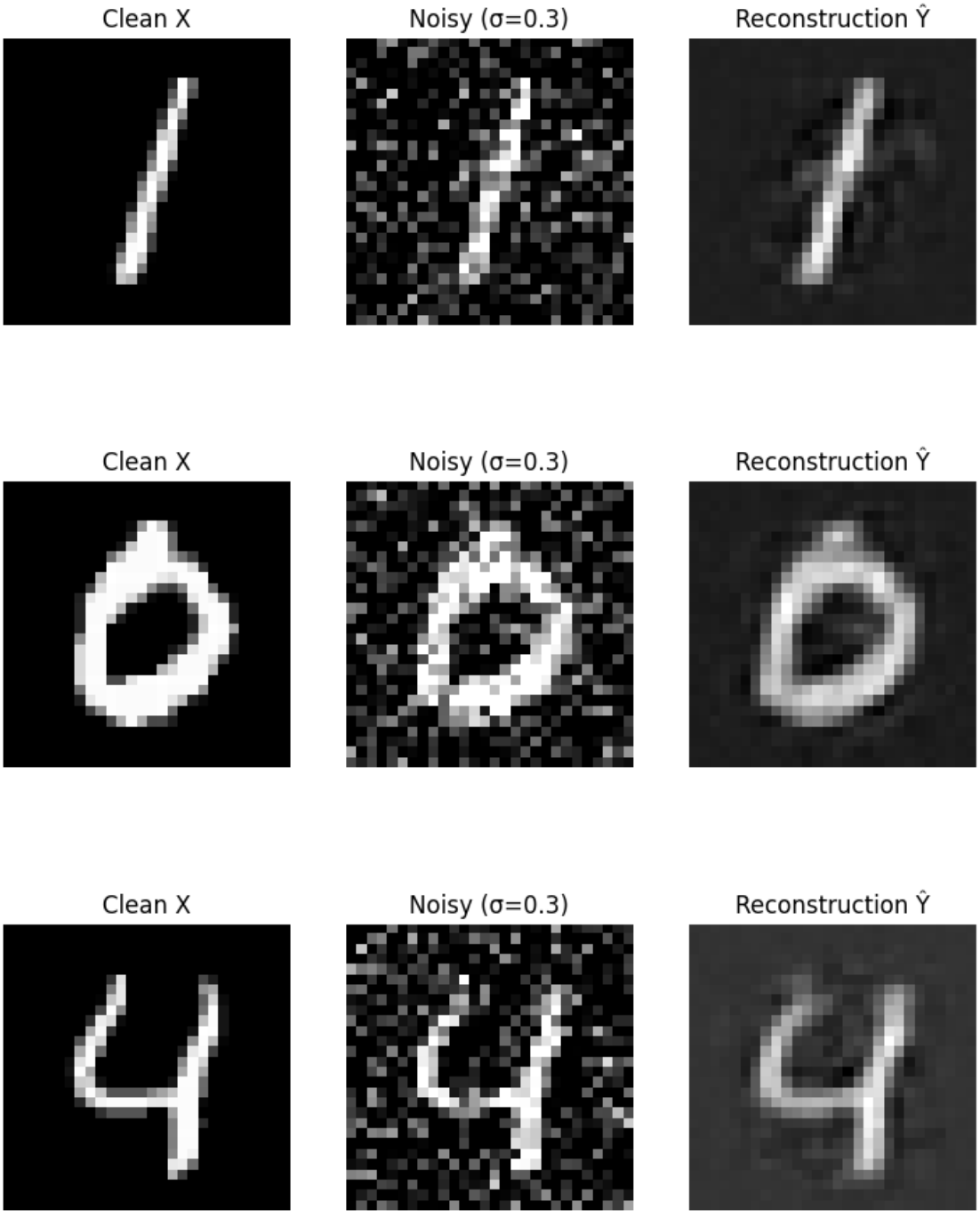
```
/Users/zanwenfu/miniforge3/envs/ece685/lib/python3.11/site-
packages/torch/utils/data/dataloader.py:692: UserWarning: 'pin_memory' argument
is set as true but not supported on MPS now, device pinned memory won't be used.
  warnings.warn(warn_msg)
```

```
Epoch 01/10  loss=0.009012
Epoch 02/10  loss=0.005926
Epoch 03/10  loss=0.005568
Epoch 04/10  loss=0.005364
Epoch 05/10  loss=0.005210
Epoch 06/10  loss=0.005101
Epoch 07/10  loss=0.004994
Epoch 08/10  loss=0.004901
Epoch 09/10  loss=0.004836
Epoch 10/10  loss=0.004775
```

[12]:
```
model.eval()
n_show = 5
sigma_display = TRAIN_SIGMA

with torch.no_grad():
    x, _ = next(iter(test_loader))
    x = x.to(device)
    x_noisy = add_noise(x, sigma_display)
    x_hat, _ = model(x_noisy)

def show_triplet(clean, noisy, recon, idx):
    fig = plt.figure(figsize=(9,3))
    plt.subplot(1,3,1); plt.imshow(clean[idx,0].cpu(), cmap="gray"); plt.
 ↪axis("off"); plt.title("Clean X")
    plt.subplot(1,3,2); plt.imshow(noisy[idx,0].cpu(), cmap="gray"); plt.
 ↪axis("off"); plt.title(f"Noisy ( ={sigma_display})")
    plt.subplot(1,3,3); plt.imshow(recon[idx,0].cpu(), cmap="gray"); plt.
 ↪axis("off"); plt.title("Reconstruction Ŷ")
    plt.show()

for i in range(n_show):
    show_triplet(x, x_noisy, x_hat, i)
```

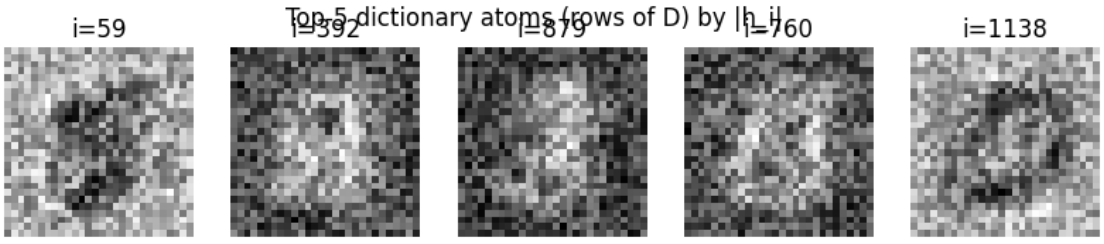





[13]:
```
model.eval()
with torch.no_grad():
    x_sample, _ = test_ds[random.randrange(len(test_ds))]
```

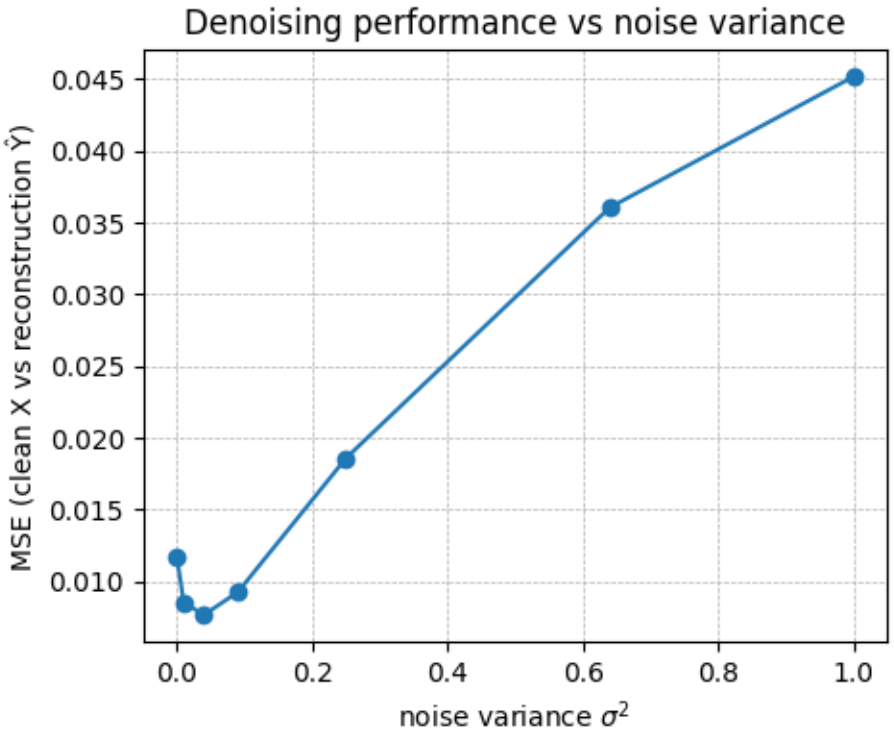
```
    x_sample = x_sample.unsqueeze(0).to(device)
    x_noisy_s = add_noise(x_sample, sigma_display)
    x_hat_s, h_s = model(x_noisy_s)            # h_s: [1, H_DIM]
    h_abs = h_s[0].abs().detach().cpu().numpy()
    topk_idx = np.argsort(-h_abs)[:5]          # top-5 |h_i|

D = model.dec.D.detach().cpu()                 # [H_DIM, INPUT_DIM]

fig = plt.figure(figsize=(10,2))
for j, idx in enumerate(topk_idx, 1):
    atom = D[idx].view(28,28).numpy()
    plt.subplot(1,5,j); plt.imshow(atom, cmap="gray"); plt.axis("off"); plt.
 ↪title(f"i={int(idx)}")
plt.suptitle("Top-5 dictionary atoms (rows of D) by |h_i|"); plt.show()
```




[14]:
```
def eval_mse_at_sigma(sigma):
    model.eval()
    total, count = 0.0, 0
    with torch.no_grad():
        for x, _ in test_loader:
            x = x.to(device)
            x_noisy = add_noise(x, sigma)
            x_hat, _ = model(x_noisy)
            total += F.mse_loss(x_hat, x, reduction="sum").item()
            count += x.numel()
    return total / count  # average per-pixel MSE

sigmas = [0.0, 0.1, 0.2, 0.3, 0.5, 0.8, 1.0]
mse_vals = [eval_mse_at_sigma(s) for s in sigmas]

for s, v in zip(sigmas, mse_vals):
    print(f" ={s:.2f}  MSE={v:.6f}")

plt.figure(figsize=(5,4))
plt.plot([s*s for s in sigmas], mse_vals, marker="o")
plt.xlabel(r"noise variance $\sigma^2$")
```

```
plt.ylabel("MSE (clean X vs reconstruction Ŷ)")
plt.title("Denoising performance vs noise variance")
plt.grid(True, linestyle="--", linewidth=0.5)
plt.show()
```

```
=0.00  MSE=0.011618
=0.10  MSE=0.008486
=0.20  MSE=0.007650
=0.30  MSE=0.009242
=0.50  MSE=0.018580
=0.80  MSE=0.036046
=1.00  MSE=0.045155
```



# Q3

November 7, 2025

## 0.1 Q3: Gaussian-Bernoulli Restricted Boltzmann Machines

[8]:
```
%matplotlib inline

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

if torch.backends.mps.is_available():
    device = torch.device("mps")
elif torch.cuda.is_available():
    device = torch.device("cuda")
else:
    device = torch.device("cpu")
```

[9]:
```
batch_size = 128

transform = transforms.ToTensor()  # keeps pixels in [0,1]

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

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

```
    transform=transform
)

train_loader = torch.utils.data.DataLoader(
    train_dataset, batch_size=batch_size, shuffle=True, drop_last=True
)

test_loader = torch.utils.data.DataLoader(
    test_dataset, batch_size=batch_size, shuffle=False, drop_last=False
)

n_vis = 28 * 28
print("Visible dimension:", n_vis)
```

```
Visible dimension: 784
```

[10]:
```
class GaussianBernoulliRBM(nn.Module):
    def __init__(self, n_vis, n_hid, k=1, sigma_val=1.0):
        super().__init__()
        self.n_vis = n_vis
        self.n_hid = n_hid
        self.k = k

        self.W = nn.Parameter(0.01 * torch.randn(n_vis, n_hid))
        self.v_bias = nn.Parameter(torch.zeros(n_vis))
        self.h_bias = nn.Parameter(torch.zeros(n_hid))

        # fixed sigma_i
        sigma = torch.full((n_vis,), float(sigma_val))
        self.register_buffer("sigma", sigma)

    def hidden_prob(self, v):
        v_scaled = v / self.sigma
        return torch.sigmoid(v_scaled @ self.W + self.h_bias)

    def sample_h(self, v):
        p_h = self.hidden_prob(v)
        return torch.bernoulli(p_h), p_h

    def visible_mean(self, h):
        # mean of p(v|h)
        return self.v_bias + self.sigma * (h @ self.W.t())

    def sample_v(self, h):
        mean = self.visible_mean(h)
        # gaussian sample used only in CD:
        v_sample = mean + self.sigma * torch.randn_like(mean)
```

```
        return v_sample, mean

    def free_energy(self, v):
        v_centered = (v - self.v_bias) / self.sigma
        vis_term = 0.5 * (v_centered ** 2).sum(dim=1)
        hidden_lin = (v / self.sigma) @ self.W + self.h_bias
        hidden_term = F.softplus(hidden_lin).sum(dim=1)
        return vis_term - hidden_term

    def gibbs_k(self, v0):
        v = v0
        for _ in range(self.k):
            h, _ = self.sample_h(v)
            v, _ = self.sample_v(h)
        return v
```

[11]:
```
def make_optimizer(rbm, lr=1e-3):
    return optim.Adam(rbm.parameters(), lr=lr)


def train_rbm(rbm, train_loader, epochs=25, lr=1e-3, verbose=True):
    rbm.train()
    opt = make_optimizer(rbm, lr=lr)

    for epoch in range(1, epochs + 1):
        total_loss = 0.0
        n_batches = 0

        for batch, _ in train_loader:
            batch = batch.view(batch.size(0), -1).to(device)

            v0 = batch
            # CD-1 negative sample
            vk = rbm.gibbs_k(v0).detach()

            # Free energy difference loss:
            loss = (rbm.free_energy(v0) - rbm.free_energy(vk)).mean()

            opt.zero_grad()
            loss.backward()
            opt.step()

            total_loss += loss.item()
            n_batches += 1

        if verbose:
            print(f"Epoch {epoch:3d} | Loss: {total_loss / n_batches:.6f}")
```

```python
@torch.no_grad()
def reconstruction_mse(rbm, data_loader):
    rbm.eval()
    total_se = 0.0
    total_pix = 0

    for batch, _ in data_loader:
        v0 = batch.view(batch.size(0), -1).to(device)
        # one-step reconstruction:
        h_prob = rbm.hidden_prob(v0)
        h_sample = torch.bernoulli(h_prob)
        v_mean = rbm.visible_mean(h_sample)

        # clip to [0,1] since inputs are in [0,1]
        v_mean = torch.clamp(v_mean, 0.0, 1.0)

        se = F.mse_loss(v_mean, v0, reduction="sum").item()
        total_se += se
        total_pix += v0.numel()

    return total_se / total_pix
```

[13]:
```python
hidden_sizes = [10, 50, 100, 250]
results = {}

for M in hidden_sizes:
    print(f"\n=== Training Gaussian-Bernoulli RBM with {M} hidden units ===")
    rbm = GaussianBernoulliRBM(n_vis=n_vis, n_hid=M, k=1).to(device)
    train_rbm(rbm, train_loader, epochs=25, lr=1e-3, verbose=True)

    test_mse = reconstruction_mse(rbm, test_loader)
    results[M] = (rbm, test_mse)
    print(f"Test reconstruction MSE (M={M}): {test_mse:.6f}")
```

```
=== Training Gaussian-Bernoulli RBM with 10 hidden units ===
Epoch   1 | Loss: -356.074600
Epoch   2 | Loss: -357.745492
Epoch   3 | Loss: -357.957886
Epoch   4 | Loss: -358.860389
Epoch   5 | Loss: -363.493862
Epoch   6 | Loss: -366.127349
Epoch   7 | Loss: -368.254419
Epoch   8 | Loss: -369.593607
Epoch   9 | Loss: -370.454461
Epoch  10 | Loss: -371.040498
```

```
Epoch  11 | Loss: -371.680281
Epoch  12 | Loss: -372.164262
Epoch  13 | Loss: -372.635438
Epoch  14 | Loss: -373.036469
Epoch  15 | Loss: -373.548179
Epoch  16 | Loss: -374.044232
Epoch  17 | Loss: -374.464832
Epoch  18 | Loss: -374.769481
Epoch  19 | Loss: -374.864906
Epoch  20 | Loss: -375.050380
Epoch  21 | Loss: -375.182436
Epoch  22 | Loss: -375.378754
Epoch  23 | Loss: -375.499984
Epoch  24 | Loss: -375.549753
Epoch  25 | Loss: -375.674578
Test reconstruction MSE (M=10): 0.040334

=== Training Gaussian-Bernoulli RBM with 50 hidden units ===
Epoch   1 | Loss: -362.264392
Epoch   2 | Loss: -370.176814
Epoch   3 | Loss: -372.621070
Epoch   4 | Loss: -374.271910
Epoch   5 | Loss: -375.609468
Epoch   6 | Loss: -376.197354
Epoch   7 | Loss: -376.752186
Epoch   8 | Loss: -377.346155
Epoch   9 | Loss: -377.621582
Epoch  10 | Loss: -377.776280
Epoch  11 | Loss: -378.138925
Epoch  12 | Loss: -378.319652
Epoch  13 | Loss: -378.468719
Epoch  14 | Loss: -378.617254
Epoch  15 | Loss: -378.826147
Epoch  16 | Loss: -378.967096
Epoch  17 | Loss: -379.263967
Epoch  18 | Loss: -379.218247
Epoch  19 | Loss: -379.355548
Epoch  20 | Loss: -379.495095
Epoch  21 | Loss: -379.520778
Epoch  22 | Loss: -379.552484
Epoch  23 | Loss: -379.766672
Epoch  24 | Loss: -379.620055
Epoch  25 | Loss: -379.663063
Test reconstruction MSE (M=50): 0.030755

=== Training Gaussian-Bernoulli RBM with 100 hidden units ===
Epoch   1 | Loss: -364.918366
Epoch   2 | Loss: -372.728431
```

```
Epoch   3 | Loss: -375.095759
Epoch   4 | Loss: -376.374486
Epoch   5 | Loss: -377.022250
Epoch   6 | Loss: -377.427818
Epoch   7 | Loss: -377.667513
Epoch   8 | Loss: -377.811758
Epoch   9 | Loss: -378.094511
Epoch  10 | Loss: -378.234024
Epoch  11 | Loss: -378.588990
Epoch  12 | Loss: -378.912865
Epoch  13 | Loss: -378.913644
Epoch  14 | Loss: -379.148234
Epoch  15 | Loss: -379.289983
Epoch  16 | Loss: -379.371517
Epoch  17 | Loss: -379.403906
Epoch  18 | Loss: -379.706743
Epoch  19 | Loss: -379.794621
Epoch  20 | Loss: -379.716007
Epoch  21 | Loss: -379.988178
Epoch  22 | Loss: -379.991094
Epoch  23 | Loss: -379.963454
Epoch  24 | Loss: -380.146145
Epoch  25 | Loss: -380.133716
Test reconstruction MSE (M=100): 0.029753

=== Training Gaussian-Bernoulli RBM with 250 hidden units ===
Epoch   1 | Loss: -368.476898
Epoch   2 | Loss: -375.153443
Epoch   3 | Loss: -376.126392
Epoch   4 | Loss: -376.835517
Epoch   5 | Loss: -377.165322
Epoch   6 | Loss: -377.627375
Epoch   7 | Loss: -377.872236
Epoch   8 | Loss: -378.267568
Epoch   9 | Loss: -378.733642
Epoch  10 | Loss: -378.805659
Epoch  11 | Loss: -379.191563
Epoch  12 | Loss: -379.210033
Epoch  13 | Loss: -379.271008
Epoch  14 | Loss: -379.397386
Epoch  15 | Loss: -379.466476
Epoch  16 | Loss: -379.658684
Epoch  17 | Loss: -379.674851
Epoch  18 | Loss: -379.690834
Epoch  19 | Loss: -379.943882
Epoch  20 | Loss: -379.757107
Epoch  21 | Loss: -380.099984
Epoch  22 | Loss: -379.923438
```

```
Epoch  23 | Loss: -380.098474
Epoch  24 | Loss: -379.985596
Epoch  25 | Loss: -380.178429
Test reconstruction MSE (M=250): 0.029339
```

[ ]:
```python
best_M = min(results, key=lambda m: results[m][1])
rbm_best = results[best_M][0]
print(f"Best hidden size by test MSE: M = {best_M}")

rbm_best.eval()

# Take a small batch from the test set
data, _ = next(iter(test_loader))
data = data[:32]
v0 = data.view(data.size(0), -1).to(device)

with torch.no_grad():
    # h|v
    h_prob = rbm_best.hidden_prob(v0)
    h_sample = torch.bernoulli(h_prob)
    # mean of p(v|h)
    v_mean = rbm_best.visible_mean(h_sample)
    v_mean = torch.clamp(v_mean, 0.0, 1.0)

orig_grid = make_grid(data, padding=0)
recon_grid = make_grid(v_mean.view_as(data).cpu(), padding=0)

plt.figure(figsize=(8, 4))

plt.subplot(1, 2, 1)
plt.title("Original (FashionMNIST)")
plt.axis("off")
plt.imshow(orig_grid.permute(1, 2, 0).numpy(), cmap="gray")

plt.subplot(1, 2, 2)
plt.title(f"Reconstruction (M={best_M})")
plt.axis("off")
plt.imshow(recon_grid.permute(1, 2, 0).numpy(), cmap="gray")

plt.show()
```

```
Best hidden size by test MSE: M = 250
```

Original (FashionMNIST)

Reconstruction (M=250)