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}")
```

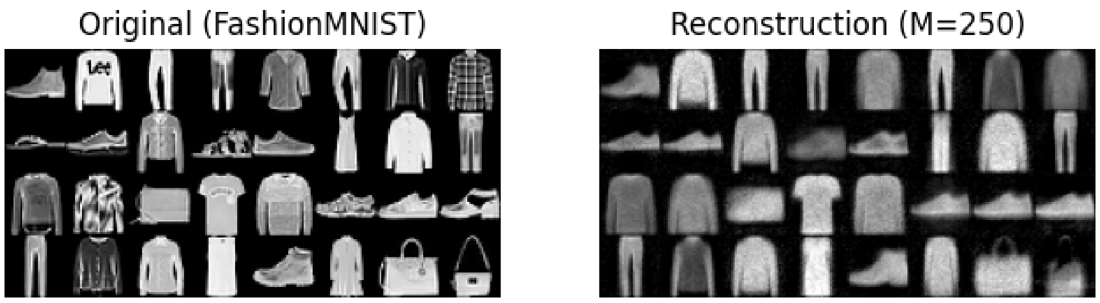
```python
@torch.no_grad()
def reconstruction_mse(rbm, data_loader):
    rbm.eval()
    total_se = 0.0
    total_pix = 0

    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
```

[ ]:
```
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)