Lab 11: Experiments with variational autoencoders

Aim: To implement a variational autoencoder on the MNIST dataset in order to perform data compression and reconstruction.

Description:

Unlike a regular autoencoder that learns a fixed mapping a VAE learns a distribution over the latent space enabling sampling and generating of new data

1. Latent variable model -

The encoder outputs two parameters for each input mean (μ) and standard deviation (σ) of the latent variable distribution.

$z = \mu + \sigma \odot \varepsilon, \quad \varepsilon \sim N(0,1)$

2. Decoder

Reconstructs the image from latent vector z.

3. Loss function

The VAE optimizes a combined loss:

Loss = Reconstruction Loss + KL Divergence

4. Generative Property

Because VAE learns a smooth latent space, we can sample random latent vectors to generate new images similar to training data.

Input (784)

↓

Encoder :

Dense (512) → Dense (256)

↓

outputs

μ (mean)

log (σ2) (log variance)

↓

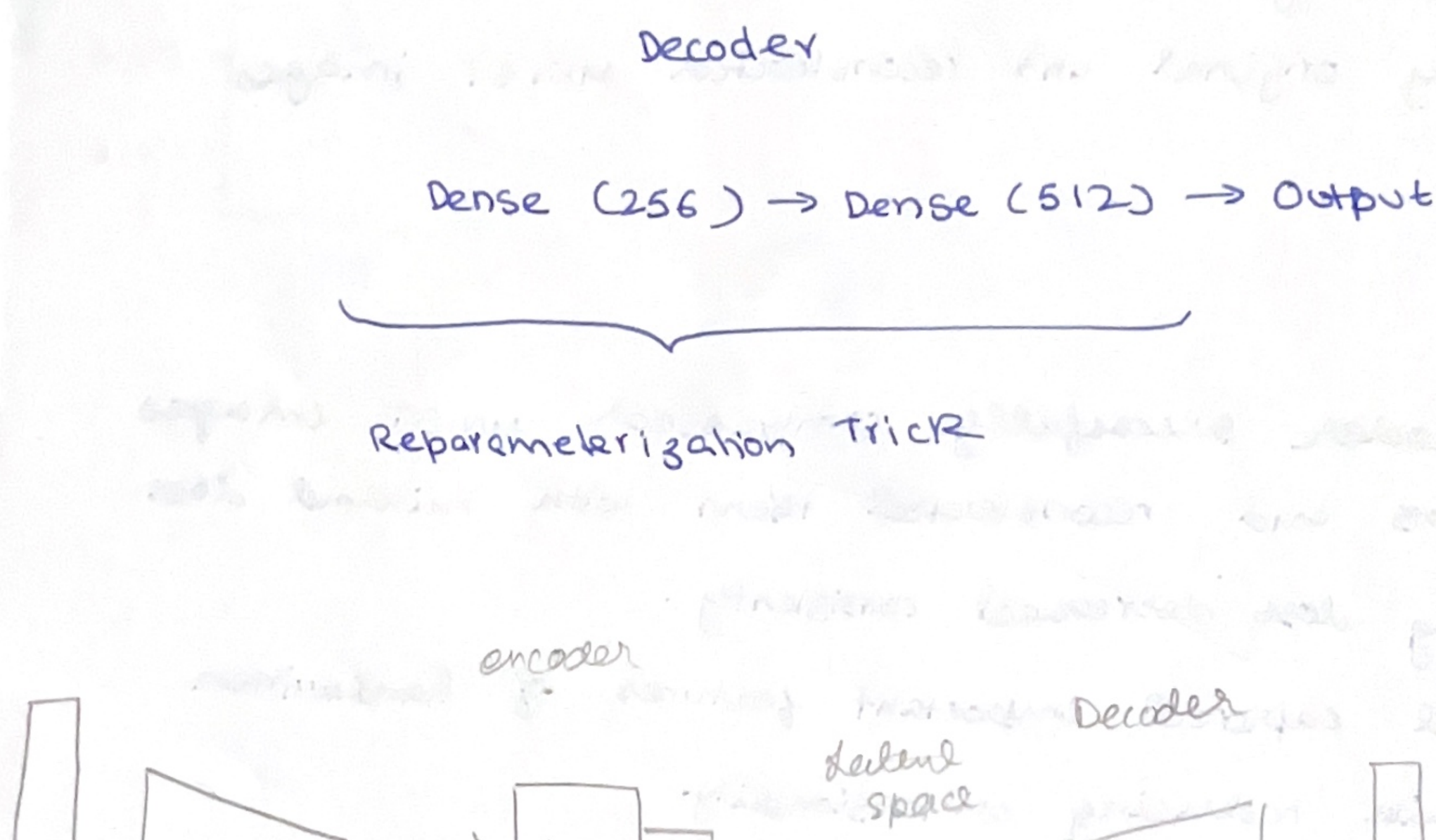
Reparameterization : $z = \mu + \sigma * \epsilon$

↓

Decoder

Dense (256) → Dense (512) → Output (784)

Reparameterization Trick



Reconstruction Loss + KL Divergence

Pseudocode :

1. Import torch, torchvision, matplotlib, etc
2. Load MNIST dataset and normalise to [0,1]
3. Define VAE model

   Encoder: Flatten → Linear layer → outputs μ and log(σ^2)

   Reparameterization

   Decoder

4. Define VAE Loss function

   Reconstruction loss (BCE)

   KL divergence between latent z ~ N(0,1)

5. Train model for given epochs

   Forward pass, compute losses, backpropogate

6. Plot training loss curve.
7. Display original, reconstructed, and generated images

Result:

The variational auto encoder successfully performed compression and probabilistic reconstruction on the MNIST dataset

Output :

Epoch	Training Loss (VAE)
1	166.23
2.	152.45
3.	145.16
5.	132.87
10.	121.34
15	114.89
20	109.76

```python
import os
import math
from typing import Tuple
import torch
from torch import nn, optim
from torch.nn import functional as F
from torchvision import datasets, transforms, utils
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt

# --------------------------
# Config
# --------------------------
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
BATCH_SIZE = 128
LR = 1e-3
EPOCHS = 20
LATENT_DIM = 20
HIDDEN_DIM = 400
DATA_DIR = "./data"
RESULT_DIR = "./results"
SEED = 42

torch.manual_seed(SEED)

os.makedirs(RESULT_DIR, exist_ok=True)

# --------------------------
# Model: simple MLP VAE
# --------------------------
class VAE(nn.Module):
    def __init__(self, input_dim: int = 28*28, hidden_dim: int = HIDDEN_DIM, latent_dim: int = LATENT_DIM):
        super().__init__()
        self.input_dim = input_dim
        # Encoder
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.fc_mu = nn.Linear(hidden_dim, latent_dim)
        self.fc_logvar = nn.Linear(hidden_dim, latent_dim)
        # Decoder
        self.fc_dec1 = nn.Linear(latent_dim, hidden_dim)
        self.fc_dec2 = nn.Linear(hidden_dim, input_dim)

    def encode(self, x: torch.Tensor) -> Tuple[torch.Tensor, torch.Tensor]:
        h1 = F.relu(self.fc1(x))
        mu = self.fc_mu(h1)
        logvar = self.fc_logvar(h1)
        return mu, logvar

    def reparameterize(self, mu: torch.Tensor, logvar: torch.Tensor) -> torch.Tensor:
        # Reparameterization trick
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)
        return mu + eps * std

    def decode(self, z: torch.Tensor) -> torch.Tensor:
        h = F.relu(self.fc_dec1(z))
        recon_logits = self.fc_dec2(h)
        # We'll apply sigmoid later (loss expects probabilities)
        return torch.sigmoid(recon_logits)
```

```
# ----------------------------
def vae_loss(recon_x: torch.Tensor, x: torch.Tensor, mu: torch.Tensor, logvar: torch.Tensor) -> torch.Tensor:
    """
    recon_x: [B, D] in (0,1) because of sigmoid
    x: original flattened images [B, D] in (0,1)
    mu, logvar: [B, latent_dim]
    Loss = BCE(reconstruction) + KL divergence
    """
    # Reconstruction loss (binary cross entropy per pixel)
    BCE = F.binary_cross_entropy(recon_x, x, reduction='sum')  # sum over batch and pixels

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

    return BCE + KLD, BCE, KLD

# ----------------------------
# Data
# ----------------------------
transform = transforms.Compose([
    transforms.ToTensor(),  # gives values in [0,1]
])

train_dataset = datasets.MNIST(DATA_DIR, train=True, download=True, transform=transform)
test_dataset = datasets.MNIST(DATA_DIR, train=False, download=True, transform=transform)

train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True, num_workers=2, pin_memory=True)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False, num_workers=2, pin_memory=True)

# ----------------------------
# Utilities for saving images
# ----------------------------
def save_reconstruction_images(model: VAE, data_loader: DataLoader, epoch: int, device=DEVICE, n=8):
    """Take first batch from loader, produce reconstructions and save side-by-side images."""
    model.eval()
    with torch.no_grad():
        imgs, _ = next(iter(data_loader))
        imgs = imgs.to(device)[:n]
        imgs_flat = imgs.view(imgs.size(0), -1)
        recon_flat, mu, logvar = model(imgs_flat)
        recon = recon_flat.view(-1, 1, 28, 28)
        # concatenate original and reconstructed (alternating)
        comparison = torch.cat([imgs, recon])
        utils.save_image(comparison.cpu(), f"{RESULT_DIR}/reconstruction_epoch_{epoch:03d}.png", nrow=n)
    model.train()

def save_sampled_images(model: VAE, epoch: int, device=DEVICE, n=64):
    """Sample z ~ N(0, I) and decode to images, save grid."""
    model.eval()
    with torch.no_grad():
        z = torch.randn(n, LATENT_DIM).to(device)
        samples = model.decode(z).view(-1, 1, 28, 28)
        utils.save_image(samples.cpu(), f"{RESULT_DIR}/sample_epoch_{epoch:03d}.png", nrow=8)
    model.train()

# ----------------------------
# Training loop
# ----------------------------
def train(model: VAE, train_loader, optimizer, epoch:int, device=DEVICE):
```

```python
def train(model: VAE, train_loader, optimizer, epoch:int, device=DEVICE):
    model.train()
    train_loss = 0.0
    train_bce = 0.0
    train_kld = 0.0
    for batch_idx, (data, _) in enumerate(train_loader):
        data = data.to(device)
        data_flat = data.view(data.size(0), -1)
        optimizer.zero_grad()
        recon_batch, mu, logvar = model(data_flat)
        loss, bce, kld = vae_loss(recon_batch, data_flat, mu, logvar)
        loss.backward()
        optimizer.step()

        train_loss += loss.item()
        train_bce += bce.item()
        train_kld += kld.item()

        if batch_idx % 200 == 0:
            print(f"Train Epoch: {epoch} [{batch_idx * len(data)}/{len(train_loader.dataset)}"
                  f" ({100. * batch_idx / len(train_loader):.0f}%)]\tLoss: {loss.item() / len(data):.4f}")

    avg_loss = train_loss / len(train_loader.dataset)
    avg_bce = train_bce / len(train_loader.dataset)
    avg_kld = train_kld / len(train_loader.dataset)
    print(f"====> Epoch: {epoch} Average loss: {avg_loss:.4f} (BCE: {avg_bce:.4f} | KLD: {avg_kld:.4f})")
    return avg_loss, avg_bce, avg_kld

def test(model: VAE, test_loader, device=DEVICE):
    model.eval()
    test_loss = 0.0
    test_bce = 0.0
    test_kld = 0.0
    with torch.no_grad():
        for data, _ in test_loader:
            data = data.to(device)
            data_flat = data.view(data.size(0), -1)
            recon_batch, mu, logvar = model(data_flat)
            loss, bce, kld = vae_loss(recon_batch, data_flat, mu, logvar)
            test_loss += loss.item()
            test_bce += bce.item()
            test_kld += kld.item()

    avg_loss = test_loss / len(test_loader.dataset)
    avg_bce = test_bce / len(test_loader.dataset)
    avg_kld = test_kld / len(test_loader.dataset)
    print(f"====> Test set loss: {avg_loss:.4f} (BCE: {avg_bce:.4f} | KLD: {avg_kld:.4f})")
    return avg_loss, avg_bce, avg_kld

# ---------------------------
# Run training
# ---------------------------
def main():
    model = VAE().to(DEVICE)
    optimizer = optim.Adam(model.parameters(), lr=LR)

    train_losses = []
    test_losses = []
    train_bces = []
```

File Edit Selection View Go Run ··· CODE

lab8.ipynb | lab11.ipynb | lab12.ipynb | lab14.ipynb | lab15.ipynb

.vscode > PPS > lab11.ipynb > import os

Generate + Code + Markdown | Run All Restart Clear All Outputs | Jupyter Variables Outline ··· Python 3.13.

```
    plt.xlabel('Epoch')
    plt.ylabel('Loss (avg per example)')
    plt.legend()
    plt.title('VAE training loss')
    plt.grid(True)
    plt.savefig(os.path.join(RESULT_DIR, "loss_curve.png"))
    plt.close()

    print(f"Training complete. Results and images are in {RESULT_DIR}")

if __name__ == "__main__":
    main()
```

[3] ✓ 9m 0.1s Python

```
100%|██████████| 9.91M/9.91M [00:25<00:00, 384kB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 81.8kB/s]
100%|██████████| 1.65M/1.65M [00:05<00:00, 290kB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 13.3MB/s]
C:\Users\awast\AppData\Roaming\Python\Python313\site-packages\torch\utils\data\dataloader.py:668: UserWarning: 'pin_memory' argument is set as true but
  warnings.warn(warn_msg)
Train Epoch: 1 [0/60000 (0%)]   Loss: 547.7181
Train Epoch: 1 [25600/60000 (43%)]      Loss: 153.1178
Train Epoch: 1 [51200/60000 (85%)]      Loss: 123.4195
====> Epoch: 1 Average loss: 165.6825 (BCE: 150.3776 | KLD: 15.3050)
====> Test set loss: 128.3353 (BCE: 107.1710 | KLD: 21.1644)
Train Epoch: 2 [0/60000 (0%)]   Loss: 128.4395
Train Epoch: 2 [25600/60000 (43%)]      Loss: 120.0056
Train Epoch: 2 [51200/60000 (85%)]      Loss: 118.3655
====> Epoch: 2 Average loss: 121.8102 (BCE: 99.1239 | KLD: 22.6863)
====> Test set loss: 115.9141 (BCE: 91.8947 | KLD: 24.0194)
Train Epoch: 3 [0/60000 (0%)]   Loss: 115.5623
Train Epoch: 3 [25600/60000 (43%)]      Loss: 114.7992
Train Epoch: 3 [51200/60000 (85%)]      Loss: 104.8220
====> Epoch: 3 Average loss: 114.8986 (BCE: 90.5636 | KLD: 24.3350)
====> Test set loss: 112.1210 (BCE: 86.9325 | KLD: 25.1884)
Train Epoch: 4 [0/60000 (0%)]   Loss: 106.1408
Train Epoch: 4 [25600/60000 (43%)]      Loss: 107.4020
Train Epoch: 4 [51200/60000 (85%)]      Loss: 110.7743
====> Epoch: 4 Average loss: 111.0942 (BCE: 87.0079 | KLD: 24.8863)
====> Test set loss: 110.2379 (BCE: 85.1449 | KLD: 25.0930)
Train Epoch: 5 [0/60000 (0%)]   Loss: 109.5320
Train Epoch: 5 [25600/60000 (43%)]      Loss: 110.8454
Train Epoch: 5 [51200/60000 (85%)]      Loss: 111.4579
====> Epoch: 5 Average loss: 110.1414 (BCE: 84.9979 | KLD: 25.1435)
====> Test set loss: 109.2726 (BCE: 83.6299 | KLD: 25.6426)

Train Epoch: 20 [51200/60000 (85%)]     Loss: 106.2419
====> Epoch: 20 Average loss: 104.1737 (BCE: 78.5096 | KLD: 25.6641)
====> Test set loss: 104.0369 (BCE: 78.7166 | KLD: 25.3203)
Training complete. Results and images are in ./results
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...