Lab 2 & 3

Daniel Mehta

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset
from sklearn.datasets import make_moons
```

Part 1: Implementing a VAE

In [4]:
```
#seed
torch.manual_seed(5504)

# create dataset
theta = np.linspace(0, 2 * np.pi, 100)
radius = 3
x = radius * np.cos(theta)
y = radius * np.sin(theta)

data = np.stack([x, y], axis=1)

dataset = TensorDataset(torch.tensor(data, dtype=torch.float32))
loader = DataLoader(dataset, batch_size=32, shuffle=True)
```

1. Train a VAE on a circular dataset & 2. Visualize the original and reconstructed data

In [6]:
```
#VAE
class VAE(nn.Module):
    def __init__(self):
        super().__init__()
        '''
        self.fc1 =nn.Linear(2,4)
        self.fc_mu = nn.Linear(4,2)
        self.fc_logvar = nn.Linear(4,2)
        self.fc2 = nn.Linear(2,4)
        self.fc3 = nn.Linear(4,2)
```

```python
        '''
        self.fc1 =nn.Linear(2, 16)
        self.fc_mu = nn.Linear(16, 2)
        self.fc_logvar = nn.Linear(16, 2)
        self.fc2 = nn.Linear(2, 16)
        self.fc3 = nn.Linear(16, 2)

    def encode(self, x):
        h = torch.relu(self.fc1(x))
        return self.fc_mu(h), self.fc_logvar(h)
    def reparam(self, mu, logvar):
        std = torch.exp(0.5 * logvar)
        return mu + std * torch.randn_like(std)

    def decode(self, z):
        h = torch.relu(self.fc2(z))
        return self.fc3(h)

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

#loss function
def vae_loss(recon, x, mu, logvar):
    recon_loss= nn.functional.mse_loss(recon, x, reduction='sum')
    kl = -0.5 * torch.sum(1+logvar-mu.pow(2) - logvar.exp())
    return recon_loss + kl
```

In [7]:

```python
#train
def train_vae(vae, loader, epochs=500, lr=0.01, print_every=100):
    optimizer = torch.optim.Adam(vae.parameters(), lr=lr)
    for epoch in range(epochs):
        for (x,) in loader:
            recon, mu, logvar = vae(x)
            loss = vae_loss(recon, x, mu, logvar)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
        if epoch % print_every == 0:
            print(f"Epoch {epoch}, Loss: {loss.item():.4f}")
```

In [8]:

```python
vae = VAE()
train_vae(vae, loader, epochs=500, lr=0.01)
```

```
Epoch 0, Loss: 38.2493
Epoch 100, Loss: 10.4523
Epoch 200, Loss: 10.6384
Epoch 300, Loss: 13.7440
Epoch 400, Loss: 17.8205
```

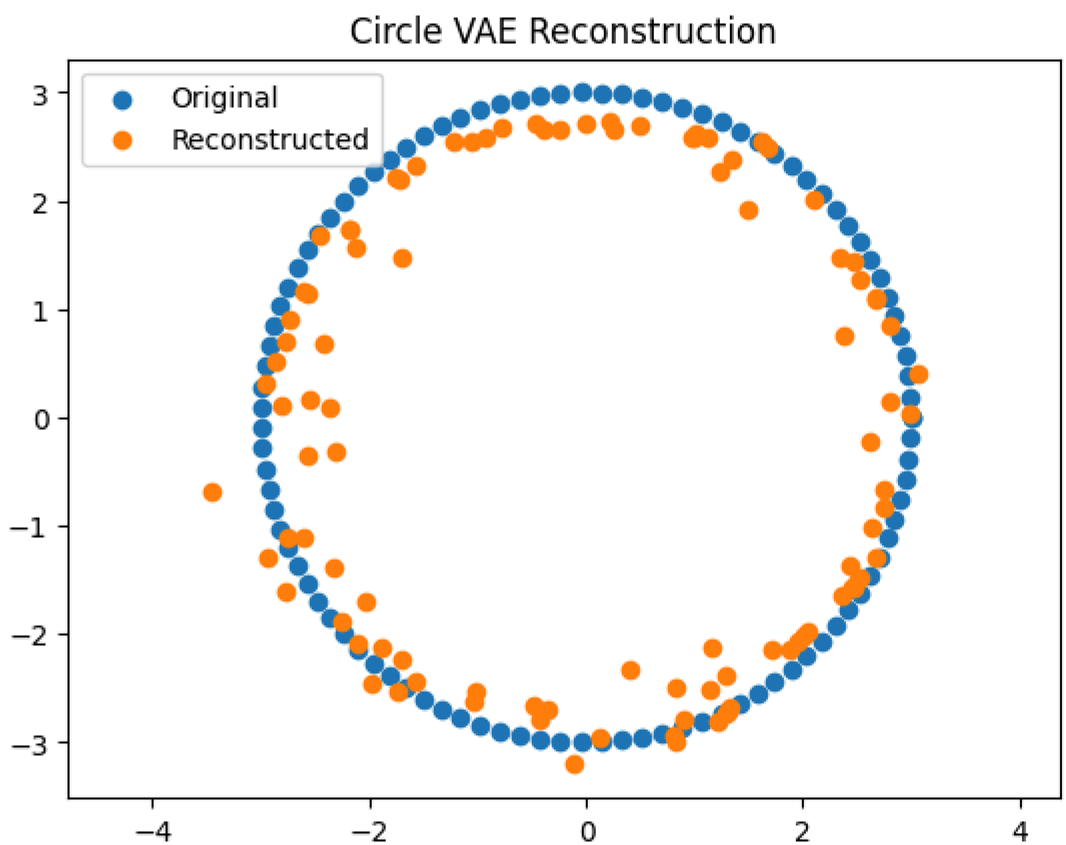
In [9]:

```python
vae.eval()

with torch.no_grad():
    x_tensor = torch.tensor(data, dtype=torch.float32)
    recon, mu, _ = vae(x_tensor)

recon = np.array(recon.cpu().tolist())

plt.scatter(data[:, 0], data[:, 1], label="Original")
plt.scatter(recon[:, 0], recon[:, 1], label="Reconstructed")
plt.legend()
plt.axis("equal")
plt.title("Circle VAE Reconstruction")
plt.show()
```



3. Analyze the VAE's performance by observing reconstruction quality.

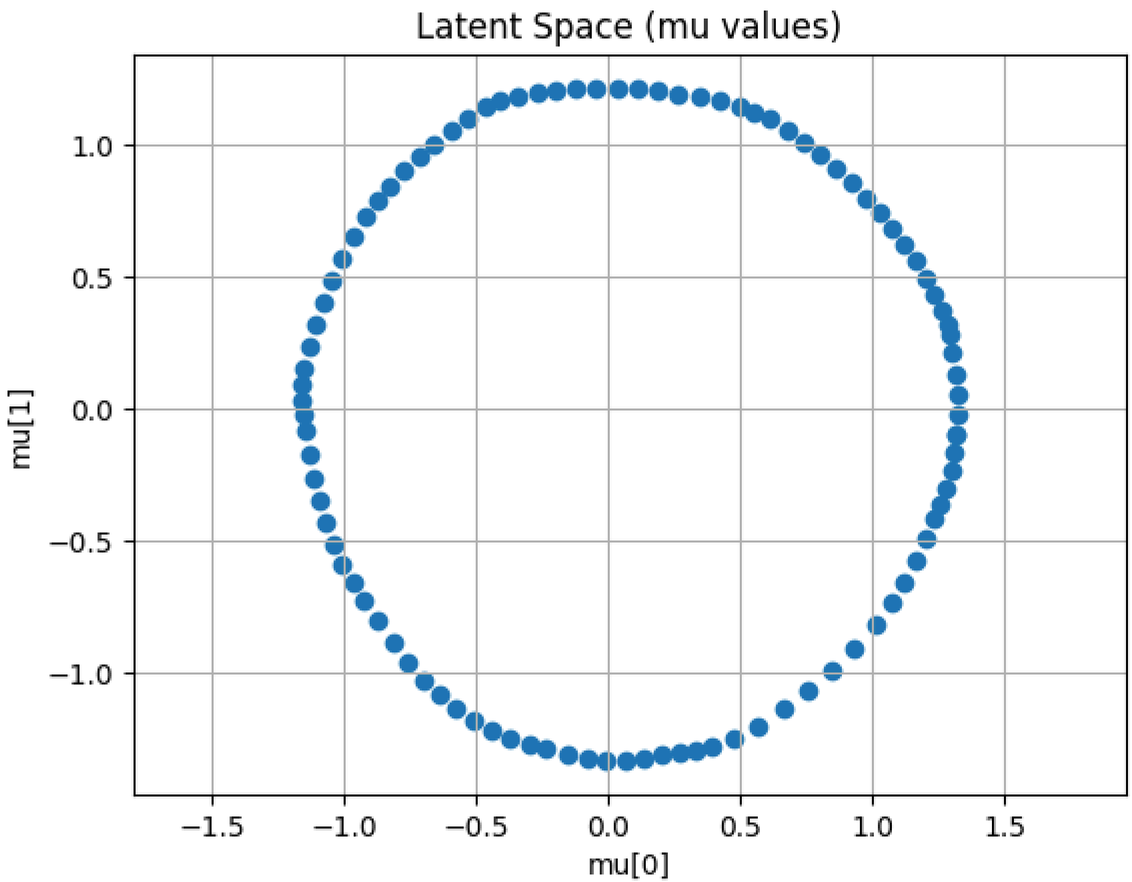
- VAE captures the circlar structure reasonably well
- Reconstructed points are scattered near the original circle
- Some noise is visible due to sampling from the latent distribution
- Overall, I believe the reconstruction quality is reasonable for a basic VAE

Part 2: Exploration and Visualization of Latent Space

1. Modify the code to plot the latent space of the trained VAE by extracting the mu (mean) values for each input point.

In [12]:
```
mu = mu.cpu().numpy()

plt.scatter(mu[:, 0], mu[:, 1])
plt.title("Latent Space (mu values)")
plt.xlabel("mu[0]")
plt.ylabel("mu[1]")
plt.axis("equal")
plt.grid(True)
plt.show()
```



2. Visualize the latent space and observe how the circular dataset is represented in 2D.

- The VAE maps the circlar dataset into nearly a circluar latent space. preserving the inputs structure.
- These points are spread evenly and continously.
- This shows how the model has learned a meaningful and organized latent representation

Part 3: Your Challenge

1. Generate new synthetic data that is different from our circular data.

In [17]:
```
#instead of circular data I generated a synthetic 'two moons'
```

```
X, _ = make_moons(n_samples=100,noise=0.05)
data = X.astype(np.float32)

dataset = TensorDataset(torch.tensor(data))
loader = DataLoader(dataset, batch_size=32,shuffle=True)
```

2. Train and test the reconstructed data

In [19]:
```
vae = VAE()
train_vae(vae, loader, epochs=3000, lr=0.001)
```

```
Epoch 0, Loss: 7.4233
Epoch 100, Loss: 5.4879
Epoch 200, Loss: 4.2822
Epoch 300, Loss: 5.9113
Epoch 400, Loss: 4.1785
Epoch 500, Loss: 5.2701
Epoch 600, Loss: 3.9023
Epoch 700, Loss: 4.2528
Epoch 800, Loss: 2.6073
Epoch 900, Loss: 2.7496
Epoch 1000, Loss: 2.7179
Epoch 1100, Loss: 3.4164
Epoch 1200, Loss: 1.8931
Epoch 1300, Loss: 5.7537
Epoch 1400, Loss: 3.6696
Epoch 1500, Loss: 3.3041
Epoch 1600, Loss: 2.8493
Epoch 1700, Loss: 2.2866
Epoch 1800, Loss: 4.8632
Epoch 1900, Loss: 3.2166
Epoch 2000, Loss: 4.8166
Epoch 2100, Loss: 4.3498
Epoch 2200, Loss: 2.6331
Epoch 2300, Loss: 4.8445
Epoch 2400, Loss: 4.5074
Epoch 2500, Loss: 3.3524
Epoch 2600, Loss: 2.9776
Epoch 2700, Loss: 3.0770
Epoch 2800, Loss: 2.2693
Epoch 2900, Loss: 4.6125
```

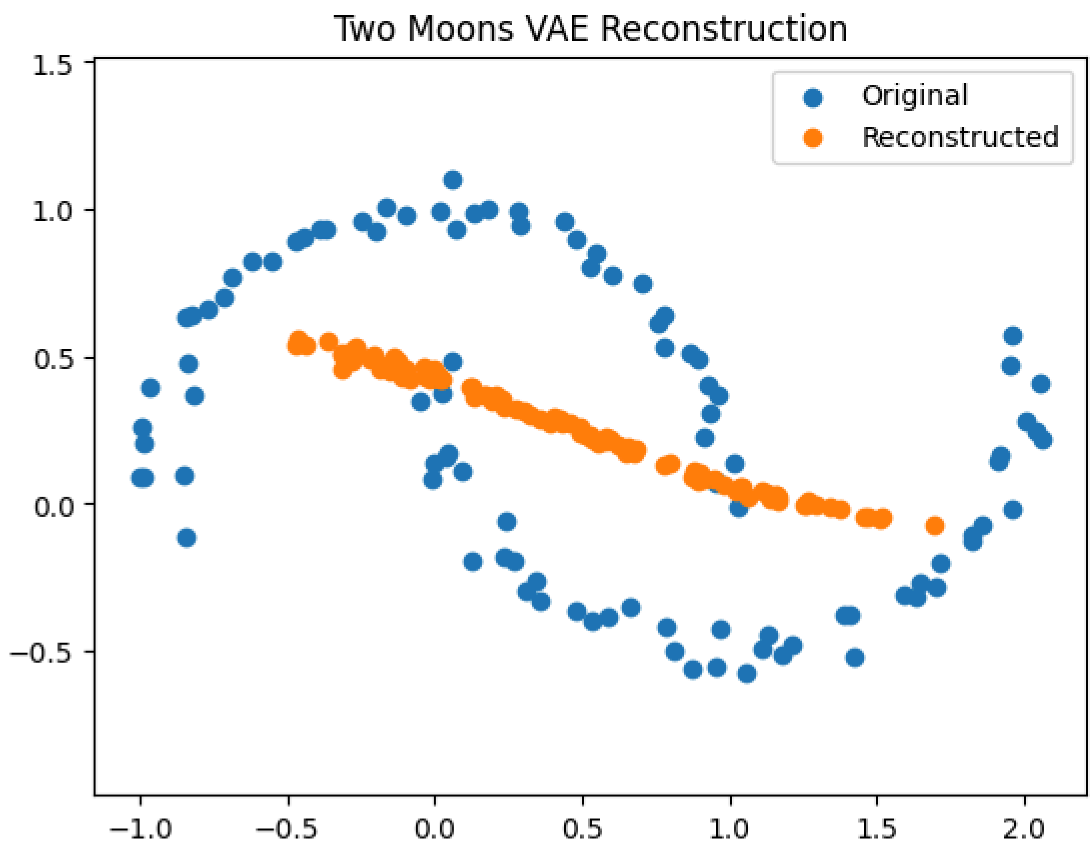
In [20]:
```
vae.eval()

with torch.no_grad():
    x_tensor = torch.tensor(data, dtype=torch.float32)
```

```
    recon, mu, _ = vae(x_tensor)

recon = np.array(recon.cpu().tolist())

plt.scatter(data[:, 0], data[:, 1], label="Original")
plt.scatter(recon[:, 0], recon[:, 1], label="Reconstructed")
plt.legend()
plt.axis("equal")
plt.title("Two Moons VAE Reconstruction")
plt.show()
```



In [ ]: