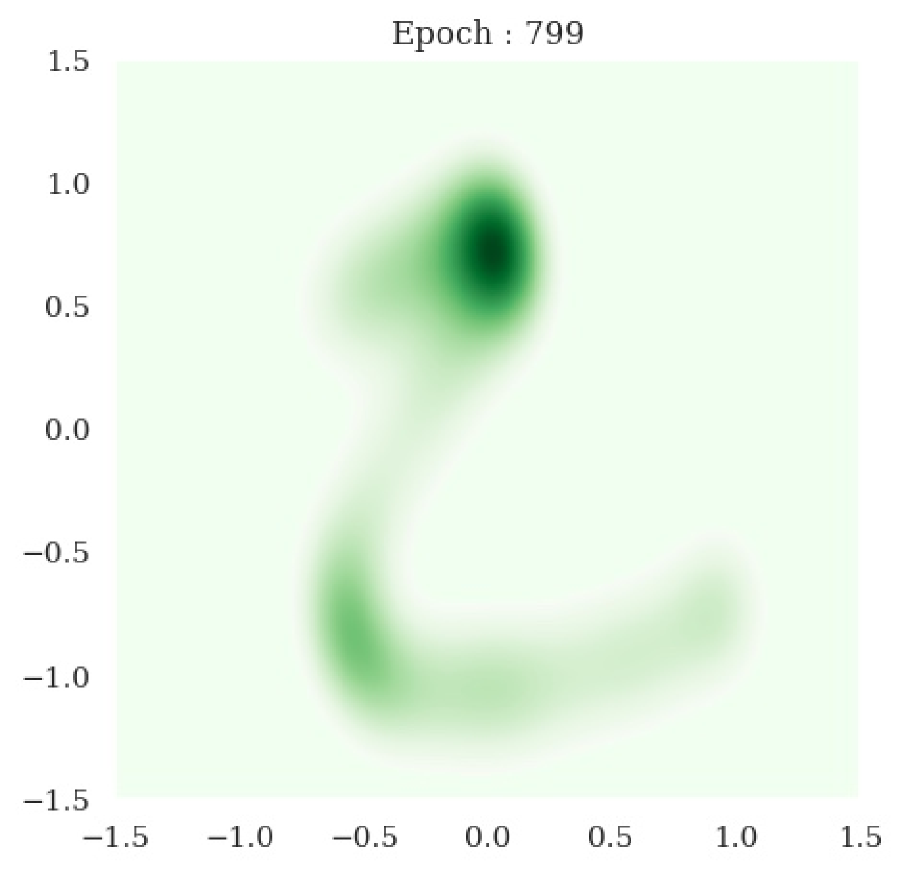
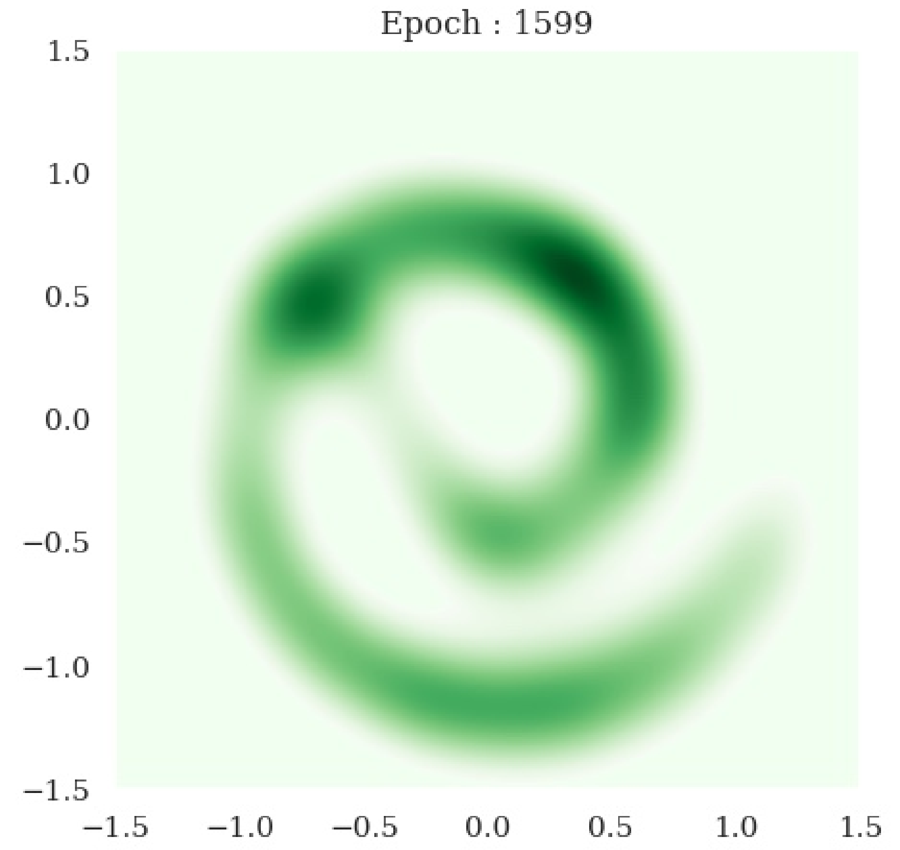
```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, TensorDataset, DataLoader
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

"""
Step 0 : Define training configurations
"""

batch_size = 64
learning_rate = 5e-4
num_epochs = 4000
reg_coeff = 500
device = "cuda:0" if torch.cuda.is_available() else "cpu"

"""
Step 1 : Define custom dataset
"""

def make_swiss_roll(n_samples=2000, noise = 1.0, dimension = 2, a = 20, b = 5):
    """
    Generate 2D swiss roll dataset
    """
    t = 2 * np.pi * np.sqrt(np.random.uniform(0.25,4,n_samples))

    X = 0.1 * t * np.cos(t)
    Y = 0.1 * t * np.sin(t)

    errors = 0.025 * np.random.multivariate_normal(np.zeros(2), np.eye(dimension), size = n_samples)
    X += errors[:, 0]
    Y += errors[:, 1]
    return np.stack((X, Y)).T

def show_data(data, title):
    """
    Plot the data distribution
    """
    sns.set(rc={'axes.facecolor': 'honeydew', 'figure.figsize': (5.0, 5.0)})
    plt.figure(figsize = (5, 5))
    plt.rc('text', usetex = False)
    plt.rc('font', family = 'serif')
    plt.rc('font', size = 10)

    g = sns.kdeplot(x=data[:, 0], y=data[:, 1], fill=True, thresh=0.1, levels=1000, cmap="Greens")

    g.grid(False)
    plt.margins(0, 0)
    plt.xlim(-1.5,1.5)
    plt.ylim(-1.5,1.5)
    plt.title(title)
```

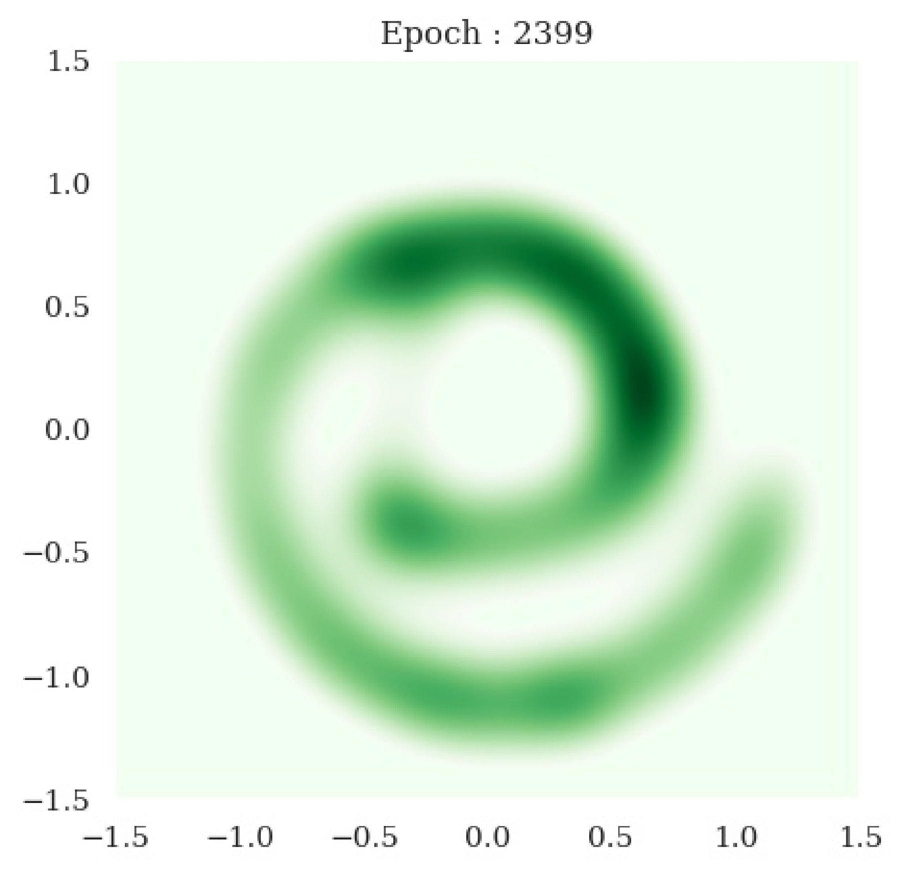
```
    plt.show()


"""
Step 2 : Define custom dataset and dataloader.
"""

class SwissRollDataset(Dataset) :
    def __init__(self, data) :
        super().__init__()
        self.data = torch.from_numpy(data)

    def __len__(self) :
        return len(self.data)

    def __getitem__(self, idx) :
        return self.data[idx]


data = make_swiss_roll()
dataset = SwissRollDataset(data)
loader = DataLoader(dataset, batch_size = batch_size, shuffle = True)


sns.set(rc={'axes.facecolor': 'honeydew', 'figure.figsize': (5.0, 5.0)})
plt.figure(figsize = (5, 5))
plt.rc('text', usetex = False)
plt.rc('font', family = 'serif')
plt.rc('font', size = 10)

g = sns.kdeplot(x=data[:, 0], y=data[:, 1], fill=True, thresh=0.1, levels=1000, cmap="Greens")

g.grid(False)
plt.margins(0, 0)
plt.xlim(-1.5,1.5)
plt.ylim(-1.5,1.5)
plt.title('p_data')
plt.savefig('swiss_roll_true.png')
plt.show()


"""
Step 3 : Implement models
"""

class Generator(nn.Module):
    def __init__(self, width = 32):
        super().__init__()
        self.layer1 = nn.Linear(1, width)
        self.layer2 = nn.Linear(width, 2)

    def forward(self, x):
        x = F.tanh(self.layer1(x))
```

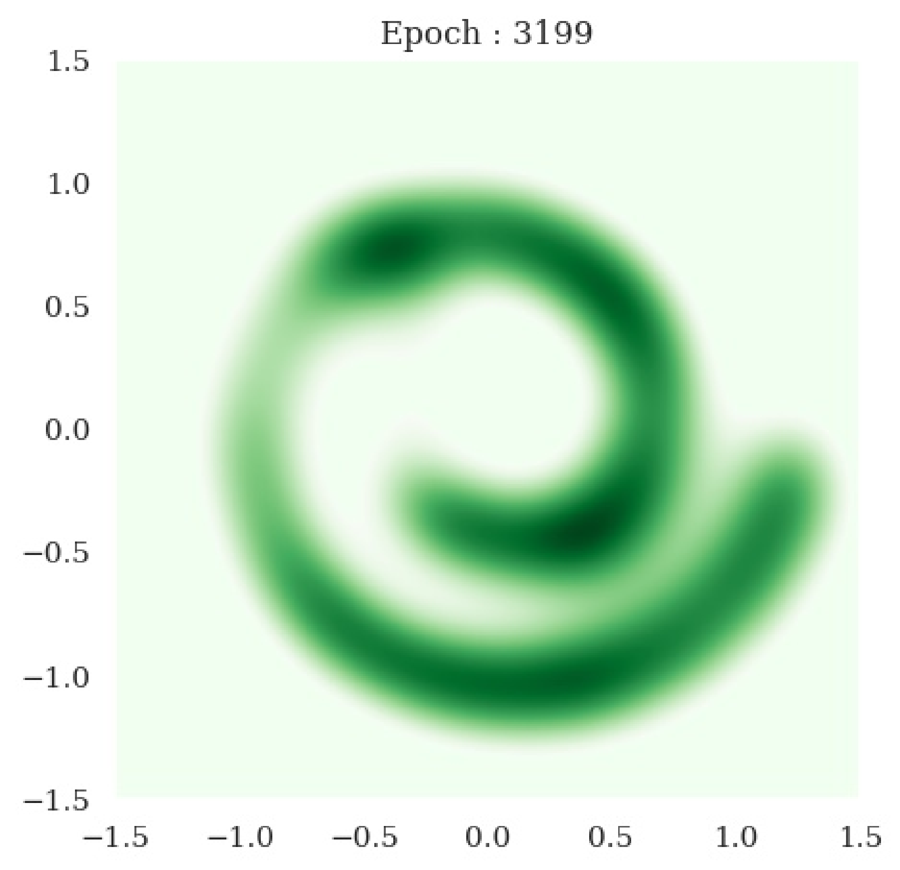
```
        x = self.layer2(x)

        return x


class Discriminator(nn.Module):
    def __init__(self, width=128):
        super().__init__()
        self.layer1 = nn.Linear(2,width)
        self.layer2 = nn.Linear(width, width)
        self.layer3 = nn.Linear(width, 1)

    def forward(self, x):
        x = F.tanh(self.layer1(x))
        x = F.tanh(self.layer2(x))
        x = F.sigmoid(self.layer3(x))

        return x



"""
Step 4 : Train models
"""
D = Discriminator().to(device)
G = Generator().to(device)

D_optimizer = torch.optim.Adam(D.parameters(), lr = learning_rate)
G_optimizer = torch.optim.Adam(G.parameters(), lr = learning_rate)


for epoch in range(num_epochs) :
    for batch_idx, x in enumerate(loader) :

        D_optimizer.zero_grad()

        x = x.view(x.shape[0], -1).to(torch.float32).to(device)

        D_real_loss = torch.mean(torch.log(D(x)))
        Z = torch.randn(batch_size,1).to(device)
        D_fake_loss = torch.mean(torch.log(1-D(G(Z))))

        D_loss = -(D_real_loss + D_fake_loss)

        D_loss.backward()
        D_optimizer.step()

        D_optimizer.zero_grad()
        G_optimizer.zero_grad()

        Z = torch.randn(batch_size,1).to(device)
        G_loss = torch.mean(torch.log(D(G(Z))))
        G_loss = -1 * G_loss
```

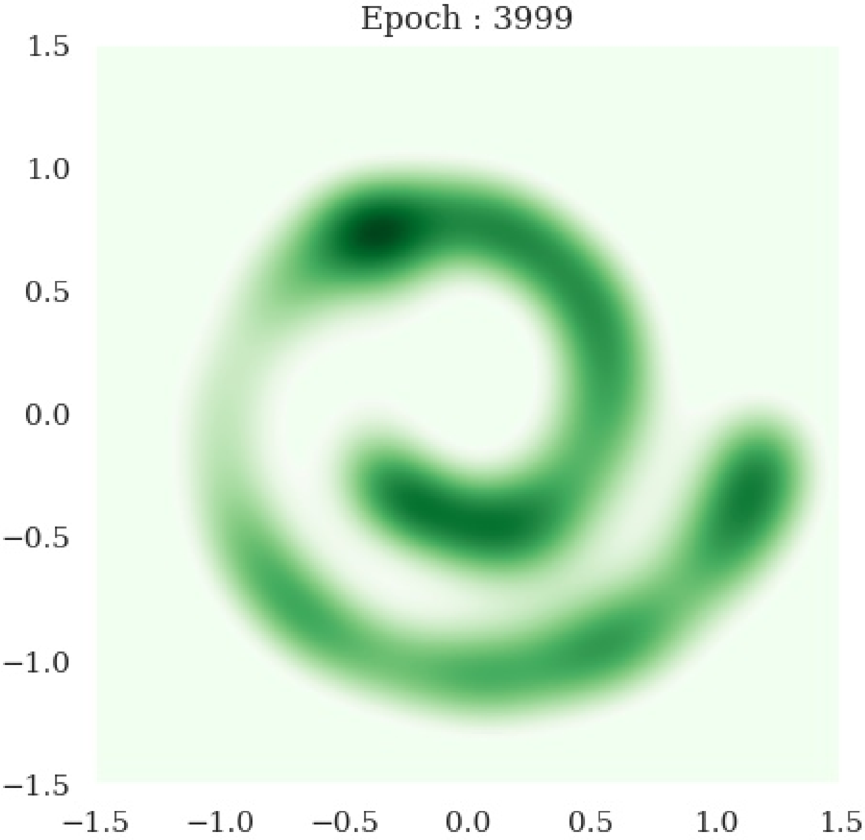
```
        G_loss.backward()
        G_optimizer.step()




    # Visualize the intermediate result
    if (epoch+1) % (num_epochs // 5) == 0:
        print(epoch)
        with torch.no_grad():
            Z = torch.randn(2000,1).to(device)
            viz = G(Z)
            viz = viz.cpu().numpy()


            sns.set(rc={'axes.facecolor': 'honeydew', 'figure.figsize': (5.0, 5.0)})
            plt.figure(figsize = (5, 5))
            plt.rc('text', usetex = False)
            plt.rc('font', family = 'serif')
            plt.rc('font', size = 10)

            g = sns.kdeplot(x=viz[:, 0], y=viz[:, 1], fill=True, thresh=0.1, levels=1000, cmap="Greens")

            g.grid(False)
            plt.margins(0, 0)
            plt.xlim(-1.5,1.5)
            plt.ylim(-1.5,1.5)
            plt.title(f"Epoch : {epoch}")
            plt.show()


with torch.no_grad():
    Z = torch.randn(2000,1).to(device)
    viz = G(Z)
    viz = viz.cpu().numpy()

    sns.set(rc={'axes.facecolor': 'honeydew', 'figure.figsize': (5.0, 5.0)})
    plt.figure(figsize = (5, 5))
    plt.rc('text', usetex = False)
    plt.rc('font', family = 'serif')
    plt.rc('font', size = 10)

    g = sns.kdeplot(x=viz[:, 0], y=viz[:, 1], fill=True, thresh=0.1, levels=1000, cmap="Greens")

    g.grid(False)
    plt.margins(0, 0)
    plt.xlim(-1.5,1.5)
    plt.ylim(-1.5,1.5)
    plt.title(f"Epoch : {epoch}")
    plt.show()
```


Epoch : 799
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


Epoch : 1599
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


Epoch : 2399
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


Epoch : 3199
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


Epoch : 3999
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5