VANDERBILT UNIVERSITY

Homework 4

CS 8395 - Special Topics in Deep Learning

Due April 25th, Midnight

1 Generative Adversarial Networks (GANs) [25pt]

In class, we talked about generative models and, in particular, generative adversarial networks (GANs). In this problem, you will implement a simple GAN. In 'hw4_p1.pkl', you will find samples from a 2D distribution shown below in blue.

The idea in GANs is to use an adversarial network that tries to distinguish the points on the manifold of the data from any other points in $\mathbb{R}^d$. Let $\eta(x;\theta_\eta)$: $\mathbb{R}^d \rightarrow [0,1]$ be the discriminator:

$$\min_{\theta_\psi} \max_{\theta_\eta} \frac{1}{N}\sum_i \log(\eta(x_i;\theta_\eta)) + \frac{1}{M}\sum_j \log\left(1-\eta(\hat{x}_j;\theta_\eta)\right), \qquad where\ \hat{x}_j = \psi(z_j;\theta_\psi)$$

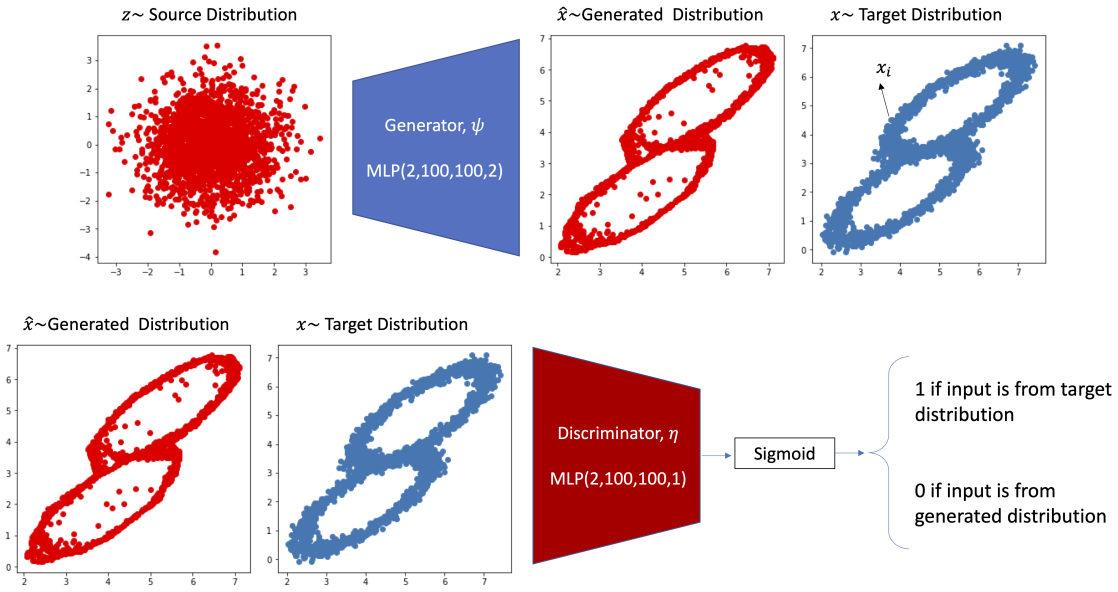

Figure 1: Problem description.

You need to train a neural network, a generator $\psi : \mathbb{R}^2 \rightarrow \mathbb{R}^2$, that receives samples from a 2D normal distribution, i.e., Source distribution, and maps it to the target distribution. Here, you will need to define an adversarial network, $\eta : \mathbb{R}^2 \rightarrow \mathbb{R}$, that is trained to distinguish real samples from the target distribution from the generated ones. The objective function that you need to optimize is shown in Figure 1. Use the following instructions and design a GAN that solves the problem.

1. For your generator, ψ, use a multi-layer perceptron with the following architecture: $2 \rightarrow 100 \rightarrow 100 \rightarrow 2$ and ReLU or LeakyReLU as its activations.
2. For your discriminator, η, use a multi-layer perceptron with the following architecture: $2 \rightarrow 100 \rightarrow 100 \rightarrow 1$ and ReLU or LeakyReLU as its activations.
3. Define two optimizers one per model. Use Adam optimizer with learning rate $lr = 1e{-}4$ for your training.
4. In each epoch, first optimize your discriminator for 10 iterations, and then optimize your generator for $1 - 5$ iterations (your choice).

Report the following items:

1. Generate a 2D grid in the range of 0-8 (using torch.meshgrid), and plot the optimized discriminator's output on this grid using the following code

```
X,Y=torch.meshgrid(torch.linspace(0,8),torch.linspace(0,8))
xgrid=torch.stack([X.reshape(-1),Y.reshape(-1)],1)
discGrid=discriminator(xgrid.to(device))
discGrid=discGrid.detach().cpu().numpy()
plt.scatter(xgrid[:,0],xgrid[:,1],c=discGrid)
```

2. Sample 2,000 points from the 2D normal distribution and send them through your optimized generator. Include the output of the following code in your report.

```
z=torch.randn((2000,2))
xhat=generator(z.to(device))
xhat=xhat.detach().cpu()
fig,ax=plt.subplots(1,2,figsize=(10,5))
ax[0].scatter(z[:,0],z[:,1])
ax[0].set_title('Network\'s input')
ax[1].scatter(xhat[:,0],xhat[:,1])
ax[1].set_title('Network\'s output')
plt.show()
```

2 Diffusion Models [25pt]

In this problem, we will revisit the generative modeling problem in Problem 1, but this time, we will use diffusion models.

To start, we will be using sinusoidal embedding to embed the time variable t into a vector. You can use the following class to do it.

```python
class SinusoidalEmbedding(nn.Module):
    def __init__(self, size: int, scale: float = 1.0):
        super().__init__()
        self.size = size
        self.scale = scale
    def forward(self, x: torch.Tensor):
        x = x * self.scale
        half_size = self.size // 2
        emb = torch.log(torch.Tensor([10000.0])) / (half_size - 1)
        emb = torch.exp(-emb * torch.arange(half_size))
        emb = x.unsqueeze(-1) * emb.unsqueeze(0)
        emb = torch.cat((torch.sin(emb), torch.cos(emb)), dim=-1)
        return emb
    def __len__(self):
        return self.size
```

Use a 128-dimensional time embedding, and define your noise predictor, $\epsilon_\theta(x, t)$ to be a MLP with the following architecture $[2 + 128, 512, 512, 2]$. Use ReLU or LeakyReLU as your activation.

To train your network you will need to implement the following. Let $q(x_0)$ denote the distribution of your input data, then the training is as follows:

Repeat:

- Sample from the dataset, $x_0 \sim q(x_0)$

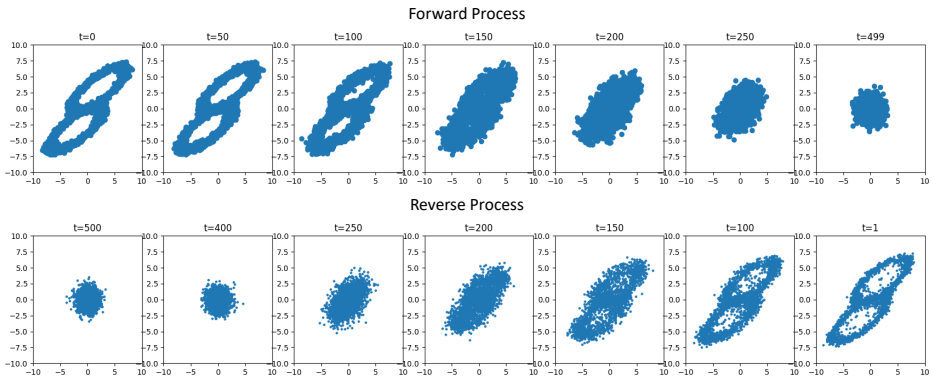

Figure 2: The forward and reverse diffusion process

- Generate time $t \sim uniform(\{1, ..., T\})$
- Sample noise, $\epsilon \sim \mathcal{N}(0, I)$
- Calculate forward diffusion, $x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon$
- Calculate Loss, $l(\theta) = \|\epsilon - \epsilon_\theta(x_t, t)\|^2$
- Perform gradient descent: $\theta \leftarrow \theta - \eta \nabla_\theta l(\theta)$

After training, to perform inference, you will need to do the following:

- Sample noise, $x_T \sim \mathcal{N}(0, I)$
- For $t = T, ..., 1$ do
  - $\epsilon \sim \mathcal{N}(0, I)$ if $t > 1$, else $\epsilon = 0_{2\times 1}$
  - $x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right) + \sqrt{\sigma_t}\epsilon$
- return x_0

Where you can use $T = 500$, $\beta = torch.linspace(1e-3, 1e-1, T) **2$, and recall the definitions:

- $\alpha_t := 1 - \beta_t$
- $\bar{\alpha}_t := \prod_{i=1}^{t} \alpha_i$
- $\sigma_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$

You must provide plots of your forward diffusion and the reverse diffusion similar to Figure 2.

3 Evasion Attacks on Neural Networks [25pts]

In this problem, you are going to download a pre-trained ResNet18 model (pre-trained on ImageNet), and calculate adversarial attacks on the input image of your choice. Here is a good Google Colab example of how you might want to set up your pre-trained model, link.

In what follows, whenever possible you should visualize the attacked image as well as its corresponding predicted class before and after the attack. Please provide the following:

1. Write down the optimization problem corresponding to un-targeted and targeted attacks, assuming the perturbation, δ, must satisfy $\delta \leq 0.05$.
2. Implement and test untargeted FGSM on your input image of choice such that you maximize the network's loss.
3. Implement and test Least Likely FGSM.
4. Implement and test Projected Gradient Descent (PGD).
5. Implement and test Carlini-Wagner (CW) attack on your image.

4 Adversarial training [25pts]

In this assignment, you will train an adversarially robust classifier on MNIST. Throughout the assignment use $\|\delta\|_\infty \leq 0.1$. Please provide the following:

1. Train an MLP classifier on MNIST and report the accuracy on the clean test set. Perform FGSM on the test set and report the accuracy of the model on the attacked images.
2. Train another MLP model on MNIST, but this time use adversarial training. You can use the FGSM attack in your adversarial training. Report the adversarially trained network's performance on clean and attacked test sets.