CS 4331/5331 – Generative AI
Homework 3

Department of Computer Science
Whitacre College of Engineering
Texas Tech University

Due by Friday, Oct. 31, 11:59pm via Canvas

You are encouraged to work in pairs. Each team member must understand the entire solution, and clearly indicate their individual contributions (who did what) at the top of the first page of the submitted assignment.

The use of generative AI to solve this assignment—especially for writing code—is strictly prohibited. Any violation will result in failing the course. Please feel free to reach out to the course staff with any questions. Gaining hands-on experience is an essential part of the learning process.

1 Generative Adversarial Networks (100 pts)

This assignment focuses on understanding GAN training dynamics. You will work with the provided Generator and Discriminator implementations. Your tasks will focus on implementing modular training functions, experimenting with different data distributions, and observing the effect of different update strategies.

1.1 Training the Discriminator (20 points)

The function `train_discriminator()` performs one training step for the discriminator D. Its purpose is to help D learn to correctly identify real data from the true distribution and fake data produced by the generator G.

Note

In this step, we want to update only the discriminator D, not the generator G. Therefore, when generating fake samples with G, we use `detach()` to prevent gradients from flowing back into G.
Example: Suppose you have two neural networks, A and B, and you want to update only B. If x is an input to A, then

$$y = A(x).\text{detach}()$$

can be used as input to B. This ensures that when computing the loss and backpropagating through B, the parameters of A remain unchanged.

The training procedure follows these steps:

1. Sample a batch of **real data** from the true data distribution.
2. Generate a batch of **fake data** using the generator G, and apply `detach()` so that only D is updated.
3. Compute the discriminator's predictions $D(\text{real})$ and $D(\text{fake})$.
4. Compute the discriminator loss using the expression introduced in the lectures:

$$\mathcal{L}_D = -\mathbb{E}_{x\sim p_{\text{data}}}[\log D(x)] - \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))].$$

5. Backpropagate this loss and update the discriminator's parameters.

To-Do: Complete the missing parts in `train_discriminator()` marked as "To-Do".

1.2 Training the Generator (20 points)

The function `train_generator()` performs one training step for the generator G. Its purpose is to help G produce samples that are increasingly similar to the real data, so that the discriminator D cannot distinguish them from true samples.

Unlike in the discriminator step, here we *do not detach* the generated samples. We want gradients to flow from D back into G so that G can improve its parameters based on how well it fools D.

The training procedure follows these steps:

1. Generate a batch of fake data using the generator G (without detaching).
2. Compute the discriminator's prediction $D(\text{fake})$ on these generated samples.
3. Compute the generator loss using the expression introduced in the lectures:

$$\mathcal{L}_G = -\mathbb{E}_{z\sim p_z}[\log D(G(z))].$$

4. Backpropagate the loss and update the generator's parameters using gradient descent.

To-Do: Complete the missing parts in `train_generator()` marked as "To-Do".

1.3 Run the Main File and Save Output (20 points)

After completing `train_discriminator()` and `train_generator()`, you will run the GAN training pipeline and save the outputs. This step helps you observe how well the generator learns different distributions.

1. Open `main.py` and locate the line where the GAN is trained:

```
losses = train_gan(device, num_steps=num_steps, plot_every=plot_every,
                   mode="mixture", batch_size=256)
```

2. To run the GAN on a different real data mode, change the `mode` argument. The available modes are defined in `sample.py` (`"gaussian"` and `"mixture"`). You may inspect the supported distributions in `sample.py`.
3. Run `main.py` for each mode. Save the generated plots and include them in your submission.
4. Briefly describe your observations for each mode, commenting on how well the generator captures the structure of the true distribution.

To-Do: Edit the `mode` argument in `train_gan()` and run experiments on both `"gaussian"` and `"mixture"` distributions, save the plots, and write a short analysis of the results.

1.4 Complex Real Data Distribution (20 points)

In this task, you will extend the set of real data distributions to study how the generator behaves with more complex data.

1. Implement a new mode in `sample_real_data()`:
   - Add `mode="mixture_4"`, a mixture of four Gaussian clusters located at the corners of a square (e.g., coordinates `[±2, ±2]`).
2. Train and evaluate the GAN with this new mode:
   (a) Run two experiments using different Gaussian variances:
      - Small variance: clusters are tight and well-separated.
      - Large variance: clusters overlap and form a more continuous distribution.
   (b) Visualize the generator's output for both settings.
   (c) Compare and discuss how the generator's learned samples differ between the two variance settings.

To-Do: Implement the new data distribution in `sample_real_data()`, provide plots of the generated samples for both variance settings and write a short comparison describing the differences.

1.5 Training Loop Experiment (20 points)

In this task, you will study how the relative update frequency of the discriminator affects GAN training and stability.

1. Modify the GAN training loop so that the discriminator is updated only **once per generator update**.
2. Retrain the GAN for each of the real data modes: `"gaussian"`, `"mixture"` and `"mixture_4"`.
3. Compare the results to the original setting where the discriminator is updated twice per generator update.
4. Observe the effect on:
   - Generator performance.
   - Training stability (e.g., mode collapse, oscillations in loss).

To-Do: Make the modifications to the GAN training loop in `train.py`, provide plots for each mode and write a short explanation of how reducing discriminator updates affected generator performance and training stability.

Deliverables

- Loss and frame-by-frame plots for each mode in problem 1.3 (4 plots).
- Observations for problem 1.3.
- Loss and frame-by-frame plots for each setting in new mode in problem 1.4 (4 plots).
- Observations for problem 1.4
- Loss and frame-by-frame plots for each mode in problem 1.5 (8 plots).
- Observations for problem 1.5
- Well-documented code (with comments).

Files

1. `GAN.py` — Contains the generator and discriminator models.
2. `train.py` — Contains the function to train the GAN.
3. `sample.py` — Contains the function used to sample real data.
4. `plotting.py` — Contains the functions for the visualizations.
5. `main.py` — Contains the main code to run everything. Everything should run only using `python main.py`.