Programming Assignment: Diffusion Model Practice

Instructor: Prof. Yi MA, Prof. Yanchao YANG

Teaching Assistant: Yi ZHANG, Pei ZHOU

July 2024

1 Preliminary

This project is a straightforward implementation of the diffusion model, which has gained significant attention in recent years. After completing this project, you will have a comprehensive understanding of the principles underlying diffusion models. Please refer to the following papers which demonstrate the details of diffusion model.

- Song, Yang, et al. "Score-based generative modeling through stochastic differential equations." arXiv preprint arXiv:2011.13456 (2020).
- Ho, Jonathan, Ajay Jain, and Pieter Abbeel. "Denoising diffusion probabilistic models." Advances in neural information processing systems 33 (2020): 6840-6851.

In brief, the diffusion process incrementally introduces small amounts of Gaussian noise to the data $\boldsymbol{x}_0 \sim p_{data}$ at each timestep t. This can be mathematically represented as the following stochastic differential equation:

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + f(\boldsymbol{x}, t)\Delta t + g(t)\boldsymbol{w} \quad \text{where} \quad \boldsymbol{w} \sim \mathcal{N}(0, \Delta t \boldsymbol{I}) \quad \text{and} \quad \boldsymbol{x}_0 \sim p_{data} \tag{1}$$

The diffusion process can be analytically resolved, with $\boldsymbol{x}_t$ being normally distributed as $\boldsymbol{x}_t \sim \mathcal{N}(\alpha_t \boldsymbol{x}_0, \sigma_t^2 \boldsymbol{I})$. Ultimately, the data is transformed into standard Gaussian noise.

In the diffusion model framework, a neural network $score_{\boldsymbol{\theta}}$ is trained to approximate score $\nabla \log p_t(\boldsymbol{x}_t)$ by marginalizing over the conditional score $\nabla \log p_t(\boldsymbol{x}_t|\boldsymbol{x}_0)$. In practice, the optimization objective is expressed as:

$$\boldsymbol{\theta} = \arg\min_{\boldsymbol{\theta}} \quad \mathbb{E}_{\boldsymbol{x}_0 \sim p_{\text{data}}, \boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})} \left| w(t)[score_\theta(\boldsymbol{x}_t, t) + \frac{\boldsymbol{\epsilon}}{\sigma_t}] \right|^2, \tag{2}$$

where $\boldsymbol{x}_t = \alpha_t \boldsymbol{x}_0 + \sigma_t \boldsymbol{\epsilon}$ and loss weight $w(t) = \sigma_t$.

Starting from standard Gaussian noise, the diffusion model reverses this diffusion process to generate data via the following update:

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + [-f(\boldsymbol{x}, t) + g(t)^2 \nabla \log p_t(\boldsymbol{x}_t)]\Delta t + g(t)\mathbf{w}, \quad \text{where} \quad \boldsymbol{w} \sim \mathcal{N}(0, \Delta t \boldsymbol{I}) \quad \text{and} \quad \boldsymbol{x}_0 \sim \mathcal{N}(0, \boldsymbol{I}), \tag{3}$$

which is also regarded as a denoising process.

2 Implementation and Experiements

Read and understand the basic principle of diffusion model, and then conduct the following experiments to implement the data visualization and diffusion model algorithms and test the sampling performance. In this section, you are required to complete the missing code segments in the specified areas to make the code

```
##### Question 1: You need to implement these lines by yourself!! Around 3 lines (not necessarily)  #####



##### End of your implementation #####
```

Figure 1: An example illustrating the areas that require completion.

executable. An example of the specified areas to be completed is provided in Figure 1. The programming assignment is available at https://colab.research.google.com/drive/1SCi403lSBJc_z-xHuQ6kNDacQ3yLtHph?usp=sharing. You can copy it and run it in your own workspace.

1. Obtain 10,000 sampling points from a Gaussian distribution and a Mixture of Gaussian distributions, respectively. Then, create a scatter plot for each batch of data points. If executed correctly, the plots should appear similar to Figure 2. For a detailed understanding of Mixture models, please refer to https://en.wikipedia.org/wiki/Mixture_model.

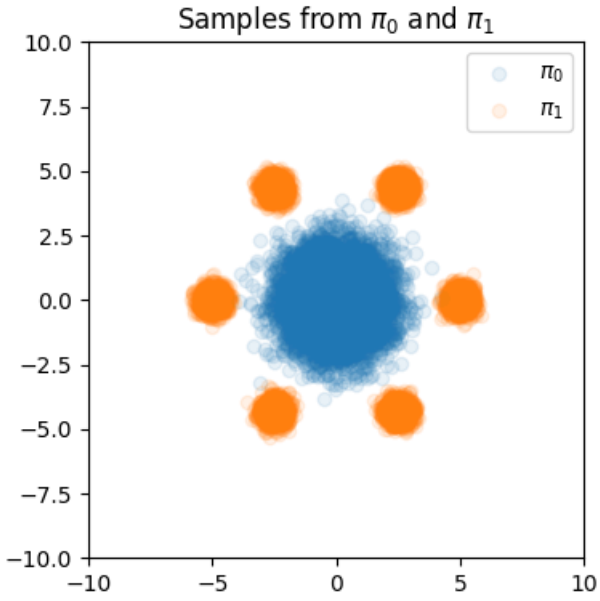

Figure 2: Scatter plots of data points sampled from a Gaussian distribution and a Mixture of Gaussian distributions.

2. Write the `forward` function of a Multilayer Perceptron (MLP) network, ensuring that all components within the `__init__` function are included in the `forward` function. The implementation should successfully output a tensor with dimensions $[10, 2]$.

3. Complete the denoising steps within the `sample_ode` function of the `VPSDE` class, and loss design in the `train` function of the same class, following the formulas provided in the preliminary section.

4. Instantiate a diffusion model and an optimizer (such as the Adam optimizer), ensuring that the training script can be executed successfully. Assign the name **flow** to the diffusion model to allow error-free execution of subsequent code. If the setup is correct, you should be able to observe a loss curve similar to the one shown in Figure 3.

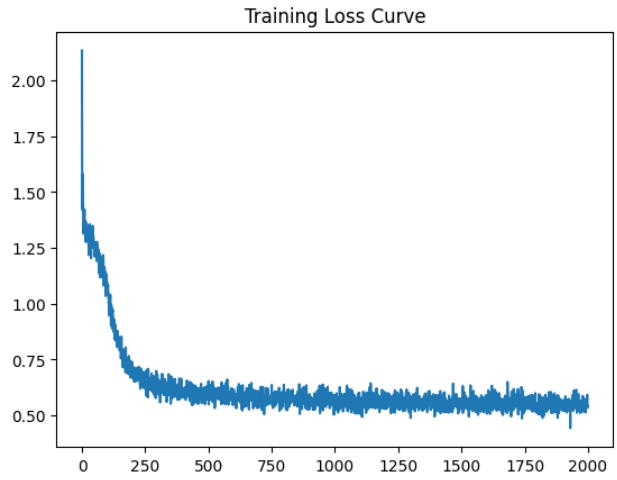

Figure 3: Training loss curve

5. Sample 1000 trajectories from the well-trained diffusion model, which is assigned to the variable **traj**. Utilize these sampled trajectories to plot the scatter plot of the endpoints of the trajectories, as well as the full trajectories in 2D space. Upon successful execution, you should be able to observe figures similar to those shown in Figure 4.

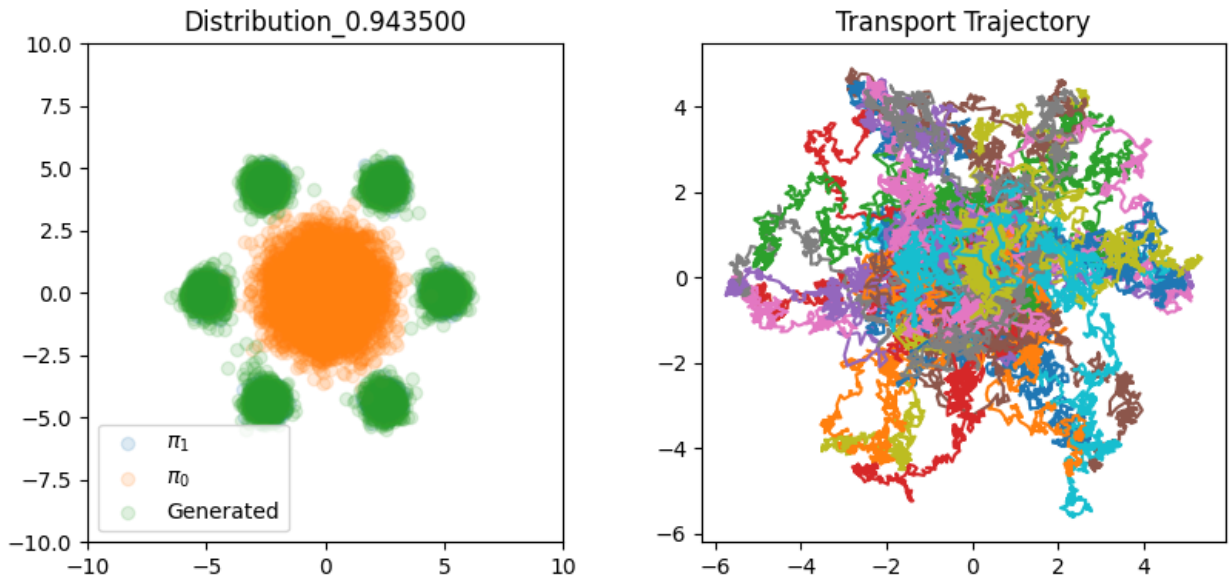

Figure 4: Left: The endpoints of the trajectories alongside the predefined distribution. Right: Sampled full trajectories in 2D space.