ELEC 5630 (L1) Assignment 5

3D Generation

Professor: TAN, Ping

Due Dec. 9th, 2025

1 Introduction to Point Cloud Generation

1.1 Point Cloud Generation

Point clouds are a fundamental data structure in 3D computer vision, representing the surface of 3D objects or environments as a collection of points in space. Each point typically has 3D coordinates (x, y, z) and may include additional attributes like color or normal vectors. Generating high-quality, realistic, and diverse point clouds is crucial for applications in virtual reality, robotics, autonomous driving, and 3D content creation. Traditional methods often rely on explicit geometric modeling or reconstruction from multi-view images. However, the discrete and unstructured nature of point clouds presents unique challenges for generative modeling compared to grid-based data like images.

1.2 Diffusion Models

Recently, **Denoising Diffusion Probabilistic Models (DDPMs)** [1], or simply Diffusion Models, have emerged as a leading class of generative models, achieving state-of-the-art results in image and audio synthesis. Diffusion models operate by defining a Markov chain that gradually adds random noise to the data (the **forward diffusion process**) and then learning a reverse Markov chain that reverses this process to generate data from noise (the **reverse generation process**).

The forward process gradually transforms a data sample $\mathbf{x}_0$ (e.g., a point cloud) into a Gaussian noise distribution $\mathbf{x}_T$ over T steps. The reverse process learns a neural network, often a U-Net or a point-cloud-specific architecture like PointNet [2, 3], to predict the noise added at each step, effectively reversing the diffusion and generating a clean sample $\mathbf{x}_0$ from pure noise $\mathbf{x}_T$. This approach has shown remarkable success in capturing complex data distributions and generating high-fidelity samples.

The application of diffusion models to point cloud generation is a vibrant area of research, aiming to leverage their stability and generative power to produce high-resolution and diverse 3D shapes. This project will guide you through implementing a simplified diffusion model for point cloud generation.

In this project, you will be provided with a skeleton code for the point cloud diffusion training pipeline, and you are required to complete the core compo-

nents. Every script you write in this section should be included in the *python/* directory. The file "run.sh" will be executed as the program entry, and you also need to upload the screenshot of the output.

Post questions to Canvas so everybody can share unless the questions are private. Please look at Canvas first if similar questions have been posted.

2 Method

2.1 Objective

In this project, you will learn and implement the foundational principles for generative modeling of point clouds using diffusion models:

1. Understand and implement the core **neural network architecture** suitable for processing and generating point clouds, such as variants of PointNet or PointNet++.
2. Familiarize yourself with and implement the **mathematical principles of the diffusion process** (forward and reverse), including the noise scheduling and loss function.
3. Implement the **sampling (generation) process** to generate novel 3D point clouds from random noise.
4. Conduct a **quantitative evaluation** of the generated point clouds using standard metrics (e.g., Minimum Matching Distance, Chamfer Distance).
5. Implement **visualization techniques** to showcase the quality and diversity of the generated point clouds and the generation process itself.

2.2 Point Cloud Encoding Network (30pts)

The network used in the reverse diffusion process (the noise predictor) must be invariant to the permutation of points, as point clouds are an unordered set of points. The ϵ-prediction network, $\epsilon_\theta(\mathbf{x}_t, t)$, takes a noisy point cloud $\mathbf{x}_t$ and the current time step t as input, and predicts the noise ϵ added at that step.

Implementation Task: You will implement the noise prediction network, ϵ_θ. This typically involves:

- **Point Feature Extraction**: Use an architecture like a simplified PointNet or a series of Multi-Layer Perceptrons (MLPs) applied individually to each point to lift the point coordinates into a high-dimensional feature space.
- **Global Feature Aggregation**: Apply a symmetric function, such as **max pooling**, to aggregate the features of all points, resulting in a **global shape feature**.
- **Feature Combination and Noise Prediction**: Concatenate the global feature and the time embedding t with the per-point features. Use subsequent MLPs to output the predicted noise ϵ for each point.

Ensure your network is designed to maintain the permutation invariance property.

2.3 Forward Diffusion Process Implementation (25pts)

The forward process gradually adds Gaussian noise to the clean point cloud $\mathbf{x}_0$ over T time steps according to a pre-defined variance schedule $\beta_1, \ldots, \beta_T$. The noisy sample $\mathbf{x}_t$ at any time t can be sampled non-recurrently using the formula:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

where $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$.

Implementation Task: You are required to:

- Implement the $\bar{\alpha}_t$ schedule (or α_t and β_t schedules) based on a linear or cosine noise schedule.
- Implement the **forward sampling function** that takes a clean point cloud $\mathbf{x}_0$ and a time step t (or a batch of t's) and returns the noisy point cloud $\mathbf{x}_t$ and the ground truth noise $\boldsymbol{\epsilon}$ for training.

2.4 Reverse Process and Training Objective (25pts)

The training objective is to learn the noise prediction network ϵ_θ by minimizing the simplified loss function, which is the mean squared error (MSE) between the predicted noise and the actual sampled noise:

$$\mathcal{L} = \mathbb{E}_{t\sim[1,T],\mathbf{x}_0,\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[\|\boldsymbol{\epsilon} - \epsilon_\theta(\mathbf{x}_t, t)\|^2\right]$$

where $\mathbf{x}_t$ is sampled from $\mathbf{x}_0$ and $\boldsymbol{\epsilon}$ using the forward process formula. The core generation step is to estimate $\mathbf{x}_{t-1}$ from $\mathbf{x}_t$ by predicting $\mathbf{x}_0$ and then sampling $\mathbf{x}_{t-1}$.

Implementation Task: You need to complete the training pipeline:

- Implement the **training step** that samples a random time step t, computes $\mathbf{x}_t$, calculates the predicted noise $\epsilon_\theta(\mathbf{x}_t, t)$, and computes the MSE loss.
- Implement the **reverse sampling process** (generation) which starts from pure noise $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and iteratively denoises it over T steps using the learned network ϵ_θ. The iterative step typically uses the mean of the learned reverse distribution.

2.5 Visualization and Evaluation (20pts)

After training, you must be able to generate new point clouds and evaluate the model's performance.

Implementation Task:

- **Generation and Visualization**: Implement a function to generate a batch of new point clouds from scratch. Use a visualization library (e.g., Matplotlib 3D, Open3D, Mitsuba [4], or a simple save-to-file utility for external viewers) to display a few generated shapes and **the denoise process**.
- **Evaluation**: Compute the **Minimum Matching Distance (MMD)** metric to evaluate the model's distribution coverage over the ground truth set. The MMD is computed between the set of generated point clouds ($\mathcal{S}_g$) and the set of ground truth point clouds ($\mathcal{S}_r$) as follows:

$$\text{MMD}(\mathcal{S}_g, \mathcal{S}_r) = \frac{1}{|\mathcal{S}_r|} \sum_{Y \in \mathcal{S}_r} \min_{X \in \mathcal{S}_g} D(X, Y)$$

where $D(X, Y)$ is the distance between two individual point clouds X and Y, and you should use the **Chamfer Distance (CD)** for $D(X, Y)$. The Chamfer Distance between two point clouds X and Y is defined as:

$$D_{chamfer}(X, Y) = \sum_{x \in X} \min_{y \in Y} \|x - y\|_2^2 + \sum_{y \in Y} \min_{x \in X} \|x - y\|_2^2$$

3 Submission

You only need to upload **ONE** zip file containing the code, the README file, the results, and the screenshot of your program. **Please do not upload the full dataset. You can prepare an empty folder called *"/data"* as a placeholder.** The code should be primarily in Python format. Implementing another language is also allowed in this assignment. You are allowed to use high-level functions provided in Python libraries (e.g., PyTorch, NumPy).

You should write the necessary information in the README file, including the command to run the script, the expected output (e.g., final loss, Chamfer Distance), and any relevant analysis or discussion of your implementation. The code in the submission must be self-contained and runnable. The script output should be matched with the file in the results folder.

Grading will only consider the completeness of the implementation, not the complexity of the implementation or the final performance of the model. Cite the paper, GitHub repo, or code URL if you use or reference code online. Please keep academic integrity; plagiarism is not tolerated in this course.

4 Tips

You can know more about the best practices of Python via **Zen of Python**.

References

[1] Jonathan Ho, Ajay Jain, and Pieter Abbeel. *Denoising Diffusion Probabilistic Models.* 2020. arXiv: `2006.11239 [cs.LG]`. URL: `https://arxiv.org/abs/2006.11239`.

[2] Charles R. Qi et al. *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation.* 2017. arXiv: `1612.00593 [cs.CV]`. URL: `https://arxiv.org/abs/1612.00593`.

[3] Charles R. Qi et al. *PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space.* 2017. arXiv: `1706.02413 [cs.CV]`. URL: `https://arxiv.org/abs/1706.02413`.

[4] Merlin Nimier-David et al. “Mitsuba 2: A Retargetable Forward and Inverse Renderer”. In: *Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 38.6 (Dec. 2019). DOI: `10.1145/3355089.3356498`.