Score-Based Point Cloud Denoising Development Track

Kunho Kim
KAIST
kaist984@kaist.ac.kr

Gyeongwon Jeong
KAIST
jgyw0910@kaist.ac.kr

Abstract

This paper reproduce Score-Based Point Cloud Denoising [1]. Denoising the noisy point cloud is crucial to many 3D vision application including autonomous driving and robotics. The original paper [1] introduced the concept of a score of the distribution model of noisy point clouds, and proposed the score-based denoising algorithm using gradient ascent. The biggest challenge experienced during implementation was the difficulty of debugging according to the long training time. Finally, we achieved almost similar but slightly inferior results compared to the authors' model, and especially our unsupervised model shows a slightly better result than authors'. Code is available at: https://github.com/Soulmates2/Score-Based-Point-Cloud-Denoising

1. Introduction

Point clouds are 3D sampled points on continuous surfaces, captured by scanning sensors. It is widely used in the 3D research field. However, due to limitations of scanning sensors, point clouds tend to be noisy. Noise in point clouds adversely affects several downstream tasks such as rendering and reconstruction, thus point cloud denoising is essentially needed in the real world.

The purpose of point cloud denoising is to create a noise-free point cloud from a noisy point cloud. While classical point cloud denoising methods have been mostly optimization-based that reiled on geometric priors [6], recent studies have emerged to denoise point clouds using deep learning. The most representative deep-learning-based denoising method is to predict the displacement between each point in the noisy point cloud and each point in the clean point cloud using a neural network [7]. However, displacement-based methods have a problem of causing shrinkage and outliers due to inaccurate predictions of noise displacement. To address this problem, authors of Score-Based Point Cloud Denoising [1] introduced the concept of a *score* of the distribution model of noisy point clouds. They construct the model that predicts the score of the distribution, and perform gradient ascent using the predicted score to denoise the point cloud; a detailed description will be given in the next section.

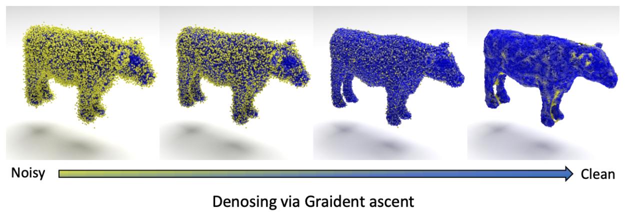


Figure 1. An illustration of score-based point cloud denoising. Performing gradient ascent with estimated score denoises the point cloud.

This report reproduce Score-Based Point Cloud Denoising. We implemented the whole pipeline of denoising network with libraries "pytorch", "pytorch3d", "pytorch-cluster" and "point_cloud_utils". We borrowed datasets, hyper-parameter setting and basic skeleton code of denoising model from authors.

We have four key challenges in reproducing the original work: (1) Traning time of the model is too long (almost 40 hours), so it is difficult to debug our code within a deadline. (2) The detailed architecture of the model is not explained in the paper, and it can be known only by looking at the authors' code. (3) Some datasets (PC-Net) required for testing and visualization codes are not provided. (4) For PC-Net dataset, authors' code even not reproduce the point-to-mesh distance results in their paper. For (2), we re-drawn the entire pipeline of the model ourselves by referring to the author's code, and implemented our model based on it.

We use the same experimental setup as the original paper. We use PU-Net [2] dataset for training and testing our model, and PC-Net [3] dataset for testing only. We evaluate our model using two metrics: Chamfer distance (CD) [4] and point-to-mesh distance (P2M) [5]. Our model shows results that are almost similar to the author's model, but slightly inferior. Especially for unsupervised learning, our model shows a slightly better result than the authors' model.

In summary, our achievements are:

- Implemented the whole code for original paper with a very basic skeleton code,
- Achieved almost similar but slightly inferior performance compared to the authors' model.

2. Method Summary

We have a noisy point cloud $\boldsymbol{X} = \{\boldsymbol{x}_i\}_{i=1}^N$ and its underlying noise-free point cloud $\boldsymbol{Y} = \{\boldsymbol{y}_i\}_{i=1}^N$. Our model is trained to predict a noise-free point cloud $\boldsymbol{Y}$ from a noisy point cloud $\boldsymbol{X}$.

2.1. Overview

A *noise-free* point cloud can be represented as a set of samples from some 3D distribution function $p : \mathbb{R}^3 \to \mathbb{R}$. Also, the distribution of the *noisy* point cloud can be seen as the convolution of the noise-free distribution p and some noise model $n : \mathbb{R}^3 \to \mathbb{R}$, expressed as follows:

$$(p * n)(\boldsymbol{x}) = \int_{\boldsymbol{s}\in\mathbb{R}^3} p(\boldsymbol{s})n(\boldsymbol{x}-\boldsymbol{s})\mathrm{d}\boldsymbol{s}. \tag{1}$$

We assume that the noise model $n(\boldsymbol{x})$ is continuous and has a unique maximum value at $\boldsymbol{x} = \boldsymbol{0}$. Under this assumption, the noise-free surface is the *mode* of the noisy distribution $p * n$. Hence we can get the noise-free point cloud by moving noisy points toward the mode of $p * n$, and this can be done by performing gradient ascent on $\log[(p * n)(\boldsymbol{x})]$, the log probability function of $p * n$.

However, the biggest problem here is that $p * n$ is unknown at test time. One important observation to tackle this problem is that in order to perform gradient ascent, it is not necessary to know $p*n$, but only the gradient of its log probability function $\nabla_{\boldsymbol{x}} \log[(p * n)(\boldsymbol{x})]$. We call this gradient a *score*. We build a neural network that estimates the score, and use it to denoise the noisy point cloud by performing gradient ascent.

Section 2.2 describes the structure of a score estimation network, and section 2.3 explains the objective function for training the score estimation network. Section 2.4 describes the gradient ascent algorithm for denoising a noise point cloud using the estimated score.

2.2. The Score Estimation Network

The score estimation network estimates the score $\mathcal{S}_i(\boldsymbol{x}) := \nabla_{\boldsymbol{x}} \log[(p*n)(\boldsymbol{x})]$ for any 3D coordinate $\boldsymbol{x} \in \mathbb{R}^3$ near the noisy point cloud $\boldsymbol{X} = \{\boldsymbol{x}_i\}_{i=1}^N$. It consists of two units: a *feature extraction unit* and a *score estimation unit*.

The feature extraction unit is a neural network that learns pointwise features from the input noisy point cloud $\boldsymbol{X}$. We adopt the structure of feature extraction networks of other denoising models [8], which consists of a stack of densely connected dynamic graph convolutional layers [9]. Let $\boldsymbol{h}_i$ denote the learned feature for point $\boldsymbol{x}_i$.

The score estimation unit takes some 3D coordinate $\boldsymbol{x} \in \mathbb{R}^3$ near $\boldsymbol{x}_i \in \boldsymbol{X}$ and the point $\boldsymbol{x}_1$'s feature $\boldsymbol{h}_i$. The score of the point $\boldsymbol{x}$ is estimated as follows:

$$\mathcal{S}_i(\boldsymbol{x}) = \mathrm{Score}(\boldsymbol{x} - \boldsymbol{x}_i, \boldsymbol{h}_i), \tag{2}$$

where $\mathrm{Score}(\cdot)$ is a multi-layer perceptron (MLP).

2.3. The Training Objective Function

In this section, we describe the objective function for training the score estimation network. The ground truth score $\boldsymbol{s}(\boldsymbol{x})$ for some point $\boldsymbol{x} \in \mathbb{R}^3$ is defined as follows:

$$\boldsymbol{s}(\boldsymbol{x}) = \frac{1}{k} \sum_{y_i \in k\mathrm{NN}(\boldsymbol{x},\boldsymbol{Y})} (\boldsymbol{y}_i - \boldsymbol{x}), \tag{3}$$

where $\boldsymbol{Y} = \{\boldsymbol{y}_i\}_{i=1}^N$ is the ground truth noise-free point cloud and $k\mathrm{NN}(\boldsymbol{x}, \boldsymbol{Y})$ is $\boldsymbol{x}$'s k-nearest neighborhood in $\boldsymbol{Y}$. Intuitively, $\boldsymbol{s}(\boldsymbol{x})$ is a vector from $\boldsymbol{x}$ to the surface of the noise-free point cloud $\boldsymbol{Y}$.

We define the training objective function $\mathcal{L}^{(i)}$ for each point $\boldsymbol{x}_i \in \boldsymbol{X}$ as follows:

$$\mathcal{L}^{(i)} = \mathbb{E}_{\boldsymbol{x}\in\mathcal{N}(\boldsymbol{x}_i)} \left[\|\boldsymbol{s}(\boldsymbol{x}) - \mathcal{S}_i((x))\|_2^2 \right], \tag{4}$$

where $\mathcal{N}(\boldsymbol{x}_i)$ is a distribution in $\mathbb{R}^3$ which concentrated in the neighborhood of $\boldsymbol{x}_i$. The final objective function $\mathcal{L}$ is just an average of each point's objective function:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N \mathcal{L}^{(i)}. \tag{5}$$

2.4. The Score-Based Denoising Algorithm

Since $\mathcal{S}_i(\boldsymbol{x})$ is trained to estimate the vector from $\boldsymbol{x}$ to the surface of ground truth point cloud $\boldsymbol{Y}$, we can solely use $\mathcal{S}_i$ to denoise $\boldsymbol{x}_i$. However, to improve robustness and reduce the bias, we use the ensemble score function:

$$\mathcal{E}_i(\boldsymbol{x}) = \frac{1}{k} \sum_{\boldsymbol{x}_j \in k\mathrm{NN}(\boldsymbol{x}_i)} \mathcal{S}_j(\boldsymbol{x}), \tag{6}$$

where $k\mathrm{NN}(\boldsymbol{x}_i)$ is $\boldsymbol{x}_i$'s k-nearest neighborhood in $\boldsymbol{X}$. Note that this k may be different from the k of equation (3).

Using the ensemble score function, denoise the point cloud $\boldsymbol{X}$ using the following gradient ascent rule:

$$\begin{aligned} &\boldsymbol{x}_i^{(t)} = \boldsymbol{x}_i^{(t-1)} + \alpha_t \mathcal{E}_i(\boldsymbol{x}_i^{(t-1)}),\ t = 1, 2, \ldots, T, \\ &\boldsymbol{x}_i^{(0)} = \boldsymbol{x}_i,\ \boldsymbol{x}_i \in \boldsymbol{X}, \end{aligned} \tag{7}$$

	# Points	10K (Sparse)						50K (Dense)					
	Noise	1%		2%		3%		1%		2%		3%	
Dataset	Model	CD	P2M	CD	P2M	CD	P2M	CD	P2M	CD	P2M	CD	P2M
PU	Authors (1)	2.521	0.463	3.686	1.074	4.708	1.942	0.716	0.150	1.288	0.566	1.928	1.041
	Authors (Unsup.)	3.107	0.888	4.675	1.829	7.225	3.762	0.918	0.265	2.439	1.411	5.303	3.841
	Authors (2)	2.549	0.486	3.656	1.090	4.837	2.123	0.714	0.152	1.251	0.543	1.918	1.037
	Ours	2.578	0.528	3.829	1.194	5.053	2.242	0.751	0.176	1.556	0.775	2.321	1.398
	Ours(Unsup.)	3.000	0.779	4.582	1.688	7.574	4.042	0.928	0.260	2.304	1.290	3.489	2.162
PC	Authors (1)	3.369	0.830	5.132	1.195	6.776	1.941	1.066	0.177	1.659	0.354	2.494	0.657
	Authors (2)	3.332	1.224	5.113	1.826	6.787	2.958	1.055	0.307	1.641	0.581	2.473	1.102
	Ours	3.208	1.198	5.185	1.785	6.960	2.857	1.053	0.279	1.908	0.649	2.829	1.119

Table 1. Comparison among authors and ours. (1) represents the result of paper, and (2) represent the result of CD and P2M is multiplied by 10^4

where T is the number of gradient ascent steps, and $\{\alpha_t\}_{t=1}^{T}$ is the step size sequence satisfying the following criteria: $\{\alpha_t\}$ is a decreasing sequence towards 0. This criteria are necessary to ensure convergence and avoid over-denoising. Finally, $\{\boldsymbol{x}_i^{(T)}\}_{i=1}^{N}$ is a denoised point cloud.

3. Implementation Details

We have implemented whole score estimation network (including feature extraction unit and score estimation unit), loss function, score-based gradient ascent algorithm, and some other stuff including preprocessing, evaluation code, and all codes for ablation studies. On the other hand, we borrowed dataset, hyper-parameter setting and basic skeleton code (including class name and parameter name of functions) of denoising model from authors [1]. We also used some external libraries, which are listed in Acknowledgments section below.

For dataset, first we normalize point clouds into the unit sphere. Then these point cloud are perturbed by Gaussian noise with randomly selected standard deviation from 0.5% to 2%. Data augmentation was applied through random scale and random rotation.

For the feature extraction unit, 4 densely connected edge convolution layers are stacked with fully-connected layers in between.

For the score estimation unit, point coordinate and shape latent are forwarded into stacked first convolution layer, 4 Resnet convolution block, and last convolution layer.

For the loss function, we just followed the equation in section 2.3. We use $k = 4$ for equation (3), and $\mathcal{N}(\boldsymbol{x}_i)$ in equation (4) is implemented by finding k-nearest neighborhood of $\boldsymbol{x}_i$ in $\boldsymbol{X}$ with $k = 32$.

For denoising, we first divide the whole point cloud into 1000 patches in order to be good at denoise without being shape-dependent. Centers of the patches are determined by the farthest point sampling. Each patch is then fed into the score estimation network, and we calculate the ensemble score (equation (6)) for each point with $k = 4$. We use $T = 30$, and $\alpha_t = 0.2 \times 0.95^t$ for the step size sequence for gradient ascent.

4. Experimental Results

4.1. Experiment Setup

Datasets For training, we use only PU-Net train dataset (40 shapes) with resolutions 10K, 30K and 50K. Each point cloud of dataset is sampled by Poisson disk sampling from each mesh file. As mentiond in Section 5, we apply transformations to the data.

For evaluation, as in the original paper, we use PU-Net test dataset (20 shapes) and PC-Net test dataset (10 shapes) provided by the author. Just like the train dataset, each point cloud of test dataset is sampled by Poisson disk sampling.

Authors use Paris-rue-Madame dataset for visual evaluation, but we did not use this dataset because of visualization issue.

Baseline In Table 1, Authors(1) represents the evaluation results described in the paper. Authors(2) represents the evaluation results with best checkpoint trained by us with author's code. We set Authors(2) as the baseline for a more accurate comparison since there is difference from paper.

Evaluation Metrics We use Chamfer distance (CD) and point-to-mesh distance (P2M). Before computing the metrics, we normalize the denoised point cloud into the unit sphere because the size of point clouds varies.

4.2. Quantitative Results

We use PU-Net test dataset (20 shapes) and PC-Net test dataset (10 shapes) with resolutions 10K and 50K provided by the author. These dataset are perturbed by isotropic Gaussian noise with standard deviation from 1% to 3%. As presented in Table 1, our implementation is slightly inferior

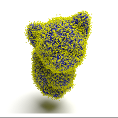
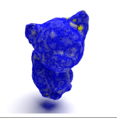
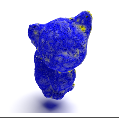
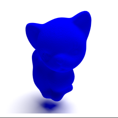
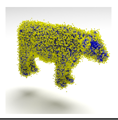
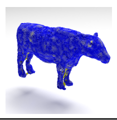
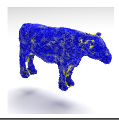
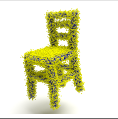
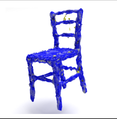
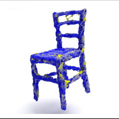
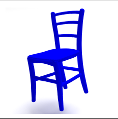
	Noisy	Authors	Ours	Clean
Kitten				
Cow				
Chair				

Figure 2. Visualization comparison of denoising results under Gaussian noise. The farther away points are from the surface, the more yellow.

to the baseline. Since model training takes 40 hours, we tried to debug each module but this was the best result.

Our model is trained with only Gaussian noise. For testing its generalizability, we wanted to use a different noise type — simulated LiDAR noise. The authors provided us these LiDAR dataset but it does not have clean data, only noisy data. Therefore, we could not evaluate quantitative evaluation for LiDAR noise.

We also conducted an experiment on unsupervised learning. Equation (4) was replaced by:

$$\mathcal{L}_{unsup} = \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_{\boldsymbol{x_j}\in\mathrm{NN}(\boldsymbol{x}_i)}\left[\|\mathbb{E}_{\boldsymbol{x_j}\in\mathrm{NN}(\boldsymbol{x}_i,\boldsymbol{X})}[\boldsymbol{x_j}-\boldsymbol{x_i}]-\mathcal{S}_i((x_j))\|_2^2\right]. \tag{8}$$

4.3. Qualitative Results

We use Mistuba renderer for visualization. The yellow color indicates the noisy points away from surface and the blue color indicates the points underlying surface. It also represents its reconstruction error measured by point-to-mesh distance introduced in Section 4.1. Our results show that score-based denoising achieve good performance.

4.4. Ablation Studies

(1) Score-based denoising algorithm We replace the gradient ascent rule by directly adding without step size α_t. Then, predicted score is added to the input coordinates x_i

$$y_i = x_i + \mathcal{E}_i(x_i). \tag{9}$$

(2) Neighborhood-covering training objective We replace the training objective by this:

$$\mathcal{L}^{(i)} = ||s(x_i) - \mathcal{S}_i(x_i)||_2^2, \tag{10}$$

which consider only the position of x_i. While the original training objective considers the neighborhood of x_i.

(3) Ensemble score function We replace the ensemble of 4 score functions with the single score function $S_i(x)$.

As shown in Table 2, all the components affect to the denoising performance. For the first and second parts of the ablation studies, evaluation result is too high compared to authors' result.

Dataset: PU		10K (Sparse)					
Noise		1%		2%		3%	
Ablation		CD	P2M	CD	P2M	CD	P2M
Authors	**(1)**	3.237	0.994	5.241	2.258	7.471	4.049
	(1)+iter.	3.237*	0.994*	5.241*	2.258*	6.073	2.953
	(2)	4.726	2.188	5.740	2.748	5.976	3.036
	(3)	2.522	0.471	4.021	1.280	6.872	3.497
	Full	**2.521**	**0.463**	**3.686**	**1.074**	**4.708**	**1.942**
Ours	**(1)**	7.504	4.596	12.45	8.866	10.49	6.978
	(1)+iter.	7.504*	4.596*	12.45*	8.866*	10.49*	6.978*
	(2)	9.932	6.714	14.24	10.54	26.43	20.94
	(3)	2.580	0.530	4.235	1.475	5.396	2.497
	Full	**2.578**	**0.528**	**3.829**	**1.194**	**5.053**	**2.242**

Table 2. Ablation studies. (*) represents that achieves the best performance with only 1 iteration. CD and P2M is multiplied by 10^4.

4.5. Beyond Denoising: Upsampling via Denoising

We asked about the upsampling setup via email, but the person who replied to the email was not the author, and he said he did not know. In addition, we could not obtain dataset for upsampling, so this experiment is infeasible.

5. Conclusion

We re-implement almost every part of Score-Based Point Cloud Denoising [1]. Our model showed little low but similar performance compared to the author's model, and showed better performance for unsupervised learning.

Acknowledgments

Kunho Kim implemented the score estimation unit and training/test pipeline, and Gyeongwon Jeong implemented the feature extraction unit and gradient ascent algorithm. There is no external collaborator. We borrowed dataset, hyper-parameter setting and rendering code from author. To evaluate point-to-mesh distance, we use libraries "point_cloud_utils" and "pytorch3d". We use library "pytorch-cluster" to use farthest point sampling function to merge denoised patches.

References

[1] Luo Shitong and Hu Wei. Score-Based Point Cloud Denoising. ICCV 2021.

[2] Yu Lequan, Li Xianzhi, Fu Chi-Wing, Cohen-Or Daniel, and Heng Pheng-Ann. PU-Net: Point Cloud Upsampling Network. CVPR 2018.

[3] Rakotosaona Marie-Julie, La Barbera Vittorio, Guerrero Paul, Mitra Niloy J and Ovsjanikov Maks. POINT-CLEANNET: Learning to denoise and remove outliers from dense point clouds. Computer Graphics Forum, 2020.

[4] Fan Haoqiang, Su Hao and Guibas Leonidas. A Point Set Generation Network for 3D Object Reconstruction from a Single Image. https://arxiv.org/abs/1612.0060. 2016.

[5] Ravi Nikhila, Reizenstein Jeremy, Novotny David, Gordon Taylor, Lo Wan-Yen, Johnson Justin and Gkioxari Georgia. Accelerating 3D Deep Learning with PyTorch3D. https://arxiv.org/abs/2007.08501. 2020

[6] Julie Digne and Carlo De Franchis. The bilateral filter for point clouds. Image Processing On Line, 7:278–287, 2017.

[7] Marie-Julie Rakotosaona, Vittorio La Barbera, Paul Guerrero, Niloy J Mitra, and Maks Ovsjanikov. Pointcleannet: Learning to denoise and remove outliers from dense point clouds. In Computer Graphics Forum, volume 39, pages 185–203. Wiley Online Library, 2020.

[8] Shitong Luo and Wei Hu. Differentiable manifold reconstruction for point cloud denoising. In Proceedings of the 28th ACM International Conference on Multimedia, pages 1330–1338, 2020.

[9] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. ACM Transactions on Graphics (TOG), 38(5):1–12, 2019.