RETHINKING SAMPLING IN 3D POINT CLOUD GENERATIVE ADVERSARIAL NETWORKS SUPPLEMENTARY MATERIAL

Anonymous authors
Paper under double-blind review

1 TRAINING DETAILS

For WGAN training, we set the gradient penalty coefficient $\lambda_{gp} = 1$. In each iteration, the discriminator gets updated ten times while the generator gets updated one time ($n_{critic} = 10$). The latent vector $z \in \mathbb{R}^{512}$ is sampled from a standard normal distribution $\mathcal{N}(0,\ I)$. We used the Adam optimizer for updating all the generator and discriminator networks with a learning rate 10^{-4} and other coefficients of $\beta_1 = 0.5$ and $\beta_2 = 0.999$. We train all the GANs for 6000 epochs on chair dataset and 1500 epochs on multi-category dataset. We observe all GANs converge at the end of training.

Our networks are implemented using PyTorch. We use one NVIDIA Titan-Xp to train a GAN model. We will release our code to facilitate research in this field.

2 NETWORK ARCHITECTURES

2.1 DISCRIMINATORS

PointNet Discriminator For all our experiments, we use a fixed PointNet architecture as below. Note that mix pooling will double the feature dimension while max pooling and average pooling keep it unchanged. For each FC layer in the MLPs except the last one before the final output, we use LeakyRELU as the activation function. To constrain the range of the discriminator output, we use a Sigmoid activation at the end, which we find helpful for stabilizing the training in our experiments.

$$\begin{array}{llll} & \text{MLP}([3, 64, 128, 1024]) & \rightarrow \text{ Max/Avg/Mix-Pooling}() & \rightarrow\ C \in \mathbb{R}^{N\times 1024/1024/2048} \\ \rightarrow & \text{MLP}([1024/1024/2048, 512, 1]) & \rightarrow \text{ Sigmoid}() & \end{array}$$

Attention-Max/Mix The network structures are shown below.

$$\begin{array}{llll} \text{MLP}([3, 32, 64]) & \rightarrow F \in \mathbb{R}^{N\times 64} & \rightarrow \text{MLP}([64, 32]) & \rightarrow G \in \mathbb{R}^{N\times 32} \\ & \rightarrow F \in \mathbb{R}^{N\times 64} & \rightarrow \text{MLP}([64, 32]) & \rightarrow H \in \mathbb{R}^{N\times 32} \\ & \rightarrow F \in \mathbb{R}^{N\times 64} & \rightarrow \text{MLP}([64, 64]) & \rightarrow K \in \mathbb{R}^{N\times 64} \\ F + \omega \text{SoftMax}(GH^T)K & \rightarrow \text{MLP}([64, 256, 1024]) & \rightarrow \text{Max/Mix-Pooling}() & \\ & \rightarrow \text{MLP}([1024/2048, 512, 1]) & \rightarrow \text{Sigmoid}() & \end{array}$$

$\omega \in \mathbb{R}$ is a learnable weight to balance between the original feature F and the feature $GH^T K$ from self attention unit.

Original attention implementation in PU-GAN(Li et al., 2019) PU-GAN(Li et al., 2019) performs an additional early fusion after the first MLP, basically it performs max pooling to obtain $T = \text{Max}(F) \in \mathbb{R}^{1\times 64}$ and per-point concatenates T to F forming $F' = [F, \text{tile}(T)]$. Other than replacing F by F', it is almost same to our implementation. We modify the dimensionality of its final feature after the second max pooling to be 1024 for a fair comparison to other discriminators. Since this structure leverages two pooling layers, it is unfair to compare it with other discriminators and hard to see the effect of its self-attention unit, which is why we introduce our PointNet-Attention.

PointNet++ To speed up the training, we use a slightly simplified version of PointNet++(Qi et al., 2017) with the architecture shown below. Same to the original implementation, we use LeakyReLU and batch normalization for each FC layer in the set abstraction layers (SA), and only LeakyReLU for the FC layers in the final MLP.

$$\begin{aligned}
& \text{SA}(512, 0.1, [3, 64, 64, 128]) && \rightarrow \text{Max-Pooling}() \\
\rightarrow\ & \text{SA}(128, 0.2, [128 + 3, 128, 256, 256]) && \rightarrow \text{Max-Pooling}() \\
\rightarrow\ & \text{GlobalSA}([256 + 3, 256, 512, 1024]) && \rightarrow \text{Max-Pooling}() \\
\rightarrow\ & \text{MLP}([1024, 512, 1]) && \rightarrow \text{Sigmoid}()
\end{aligned}$$

DGCNN To speed up the training, we use a slightly simplified version of DGCNN(Qi et al., 2017) with the architecture shown below. Same to the original implementation, we use LeakyReLU and batch normalization for each convolutional layer in the EdgeConv layers.

$$\begin{aligned}
& \text{EdgeConv}([6, 64]) && \rightarrow \text{Max-Pooling}() \\
\rightarrow\ & \text{EdgeConv}([64 \times 2, 64]) && \rightarrow \text{Max-Pooling}() \\
\rightarrow\ & \text{EdgeConv}([64 \times 2, 128]) && \rightarrow \text{Max-Pooling}() \\
\rightarrow\ & \text{EdgeConv}([128 \times 2, 256]) && \rightarrow \text{Max-Pooling}() \\
\rightarrow\ & \text{EdgeConv}([512, 1024]) && \rightarrow \text{Max-Pooling}() \\
\rightarrow\ & \text{MLP}([1024 \times 2, 512, 256, 1]) && \rightarrow \text{Sigmoid}()
\end{aligned}$$

2.2 GENERATORS

We used the officially released code of TreeGAN(Shu et al., 2019) and GraphCNN-GAN(Valsesia et al., 2018). We implement the FC generator and the deformation generator with the architectures shown below. For each FC layer except the last one in the second MLP, we use LeakyReLU as the activation function.

Note that our ground truth data are normalized point clouds with a zero center and a unit length diagonal. We hence use a Sigmoid activation at the last layer. And we translate the Sigmoid output by $(-0.5, -0.5, -0.5)$ to produce the final point cloud.

FC Generator The structure of FC generator is shown below:

$$\text{MLP}([512, 512, 512, 512, 2048, 2048 \times 3]) \rightarrow \text{Sigmoid}() - (0.5, 0.5, 0.5)$$

Deformation Generator In our experiments, we use a unit sphere as our template surface, because all of our real point clouds are sampled from closed surfaces. In addition to a latent code, we input 2048 uniformly sampled points on the unit sphere to our deformation generator. We randomly generate the points for each point cloud simply by normalizing random variables drawn from 3-dimension normal distribution into unit vectors(Muller, 1959). We add a batch normalization layer to the last FC layer in the first MLP, which we find important for its generation quality.

$$\text{MLP}([3 + 512, 512, 512, 512, 512]) \rightarrow \text{MLP}([512, 64, 3]) \rightarrow \text{Sigmoid}() - (0.5, 0.5, 0.5)$$

3 MORE RESULTS OF POINTNET-BASED DISCRIMINATOR VARIANTS

In Table.1, we provide more results of other variants of PointNet-based discriminators.

PointNet-Max-2048 Note that PointNet-Max discriminator generates a 1024-D feature after max-pooling while PointNet-Mix discriminator doubles the size to 2048-D due to the concatenation of max and average pooling features. This difference affects the weights of the following MLP layers. For a completely fair comparison, we also experiment with a PointNet-Max-2048 discriminator which has the following network structure:

$$\text{MLP}([3, 64, 128, 2048]) \rightarrow \text{Max-Pooling}() \rightarrow \text{MLP}([2048, 512, 1]) \rightarrow \text{Sigmoid}().$$

To highlight the difference, in Table 1, we change the name of our previous PointNet-Max discriminator in the Table 3 of the main paper from Max to Max-1024. When paired with FC generator, comparing to PointNet-Max-1024, PointNet-Max-2048 shares a very similar performance. It gets slightly worse on the chair dataset except for on COV-EMD metric while outperforming PointNet-Max-1024 slightly on the multi-category dataset. Using both 2048-D feature, PointNet-Max-2048 is stil far behind PointNet-Mix, which again demonstrates the advantage of Mix-Pooling over Max-Pooling.

PointNet-Attention-Mix and original PU-GAN Discriminator(Li et al., 2019) In the Table 3 of the main paper, we show that PointNet-Attention can improve the performance on sampling-related metrics comparing to PointNet-Max, which means self-attention module does capture the point density distribution though its overall generation quality is worse than PointNet-Mix. Note that PointNet-Attention also leverages a max pooling before outputting scores, it would be interesting to know whether replacing the max pooling by mix pooling can further improve its performance. In Table.1, we denote the two variants as Attention-Max and Attention-Mix, correspondingly. We find that PointNet-Attention-Mix actually performs worse than PointNet-Attention-Max. We argue that the mix-pooling's density awareness comes from average pooling, which computes the center of mass in the feature space and hence is aware to certain baised sampling, but self-attention module degrades the density awareness in the average feature. In PointNet-Attention-Mix, the mix pooling sees a weighted sum of the per-point feature F and a blended feature GH^TK from self-attention unit. Different with averaging, the learned correlation coefficients between the points GH^T no longer maintains the information of point density distribution, and that's why leveraging a mix-pooling in this case doesn't help enforcing a more uniform point density.

Our implementation of PointNet-Attention is different with the original implementation in PU-GAN(Li et al., 2019), which leverages two max pooling layers with one at an early stage and one at a later stage. We experiment with the original implementation in PU-GAN(Li et al., 2019) and find that it outperforms our Attention-Max/Mix however is still fall behind PointNet-Mix by a large margin.

Dataset	Generator	Pooling	FPD-Mix ↓	FPD-Max↓	FGD↓	MMD-E↓	MMD-C↓	COV-E↑	COV-C↑
Chair	FC	Max-1024	1.571	0.211	7.030	0.1017	0.00164	23.56	72.75
	FC	**Max-2048**	1.638	0.224	7.926	0.1697	0.00144	42.26	70.98
	FC	Mix	**0.184**	**0.209**	**2.124**	**0.0674**	0.00196	73.64	74.96
	FC	Attention-Max	0.635	0.672	4.971	0.1156	0.00160	68.92	70.54
	FC	**Attention-Mix**	0.759	0.814	5.532	0.1167	0.00167	69.36	68.04
	FC	**Attention**	0.582	0.602	4.945	0.1178	0.00164	70.98	71.72
	Deform	Max-1024	0.913	0.268	5.602	0.0908	0.00201	68.5	72.61
	Deform	Mix	0.534	0.373	2.836	0.0695	0.00200	**76.29**	**75.11**
	Deform	Attention-Max	0.696	0.755	2.987	0.1141	0.00160	68.77	69.36
	Deform	**Attention-Mix**	0.792	0.817	3.010	0.1141	**0.00157**	65.97	68.33
Multi -Cat	FC	Max-1024	1.553	0.354	6.981	0.0842	0.00153	35.16	64.16
	FC	**Max-2048**	1.415	0.306	6.226	0.1253	0.00128	53.80	**74.10**
	FC	Mix	**0.255**	**0.285**	2.550	**0.0656**	0.00184	**73.5**	72.16
	FC	Attention-Max	0.414	0.453	4.234	0.1188	0.00134	72.19	69.63
	FC	**Attention-Mix**	0.531	0.581	4.555	0.1135	0.00139	69.95	68.93
	FC	**Attention**	0.442	0.486	4.202	0.1191	0.001527	73.4	74.08
	Deform	Max-1024	1.072	0.633	3.845	0.0799	0.00179	62.5	64.5
	Deform	Mix	0.614	0.349	2.451	0.0670	0.00191	70.83	68.83
	Deform	Attention-Max	0.616	0.720	2.531	0.1113	0.00141	72.04	69.60
	Deform	**Attention-Mix**	0.447	0.492	**2.120**	0.1085	**0.00132**	73.12	70.56

Table 1: **Evaluating more variants of PointNet-based discriminators.** Here the new discriminators, Max-2048, Attention-Mix, and Attention(Li et al., 2019), are in bold.

4 FPD IMPLEMENTATION

We choose not to use the checkpoint of FPD metric in the released code(Shu, 2019) of TreeGAN(Shu et al., 2019) but train our own FPD-Max/Mix. We found that, the released code (Shu, 2019) implements a PointNet-Max network containing a spatial transformer network for feature extraction. We empirically found that the spatial transformer network, which learns to rotate the real point clouds in ModelNet40(Wu et al., 2015), can lead to a very large variance in the FPD scores of generated point clouds. So, we remove the spatial transformer network from the PointNet feature extractor and stick to a vanilla PointNet in our FPD-Mix/Max implementation.

Here we compare the original FPD and our FPD-Max, both of which use PointNet-Max features of the generated point clouds from a deformation generator. This deformation generator is trained using a PointNet-Mix discriminator. We always evaluate FPD using a set of 10K samples. To see the variance between two closed checkpoints, we use one checkpoint at epoch 5990 and the other at

	Mean@6000	Std@6000	Relative Std@6000	Mean@5990	Mean Diff	Relative Mean Diff
Original FPD(Shu, 2019)	3.814	0.2642	6.93%	1.910	1.9039	49.91%
FPD-Max	0.318	0.00655	2.06%	0.3082	0.0106	3.33%

Table 2: **Comparison between our FPD-Max and original FPD**: the first three columns show the mean, the standard deviation, and the relative standard deviation of the FPDs of the 5 different sets of generated samples using the checkpoint at epoch 6000. The fourth column shows the mean FPD of the generated samples using the checkpoint at epoch 5990. The firth and the sixth columns show the absolute and relative differences between the mean FPDs for checkpoint 6000 and 5990.

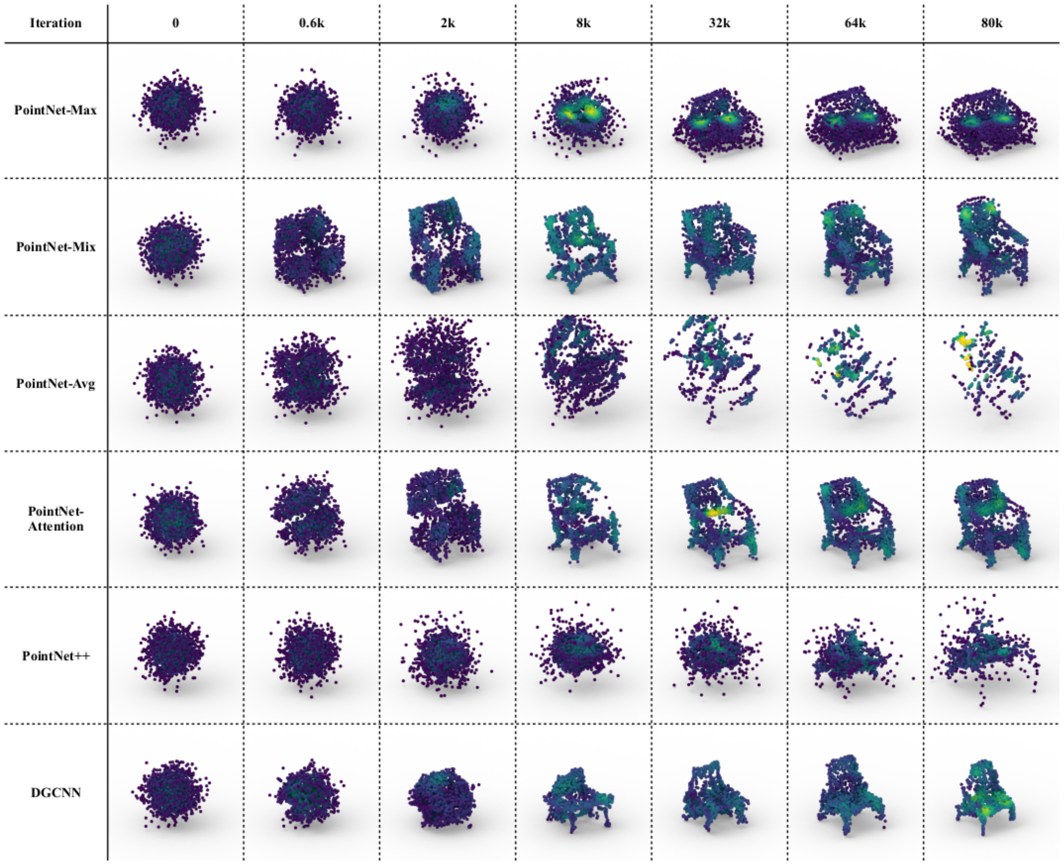

Figure 1: **Visualization of the evolution of one learnable point cloud for each discriminator during no generation trainings.** We color-code the local point density that ranges from sparse (dark blue) to dense (light yellow).

epoch 6000 in this experiment. To obtain the variance of different sets of samples, we generate 5 sets of 10K samples for the checkpoint at epoch 6000 using different random seeds.

The comparison is shown in Table 2. Our FPD-Max has a lower relative standard deviation, which is defined as the ratio of the standard deviation to the absolute value of the mean. Also, given that our GAN is almost converged at epoch 6000, our FPD-Max only relatively changes 3.33% from epoch 5990 to epoch 6000 while the original FPD changes 49.91%. The larger variance and the significant change indicate that the original FPD is not a stable metric for evaluating generated point clouds.

5 Visualization of Learnable Point Clouds during No Generator Training

In Figure 1, we show the evolution of one learnable point cloud during its training process for each discriminator. The result indicates that only PointNet-Max/Mix/Attention are qualified as good teachers for point cloud generation. Compared to PointNet-Max, PointNet-Mix/Attention give gradients to every point, resulting in a faster convergence. Also, PointNet-Mix/Attention generates a more uniform shape comparing to PointNet-Max.

For point cloud density visualization used in Figure 1 and Figure 2 of the main paper , point densities are estimated using a linear kernel density estimation method with a bandwidth 0.1. Basically, the local density of a point is proportional to the number of points within its ball neighbourhood with a radius of 0.1.

References

Ruihui Li, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. Pu-gan: a point cloud upsampling adversarial network. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 7203–7212, 2019.

Mervin E Muller. A note on a method for generating points uniformly on n-dimensional spheres. *Communications of the ACM*, 2(4):19–20, 1959.

Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, pp. 5099–5108, 2017.

Dong Wook Shu. Treegan. `https://github.com/seowok/TreeGAN`, 2019. [Online; accessed 22-November-2019].

Dong Wook Shu, Sung Woo Park, and Junseok Kwon. 3d point cloud generative adversarial network based on tree structured graph convolutions. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3859–3868, 2019.

Diego Valsesia, Giulia Fracastoro, and Enrico Magli. Learning localized generative models for 3d point clouds via graph convolution. 2018.

Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1912–1920, 2015.