Supervised Fitting of Geometric Primitives to 3D Point Clouds – Supplementary Material

Lingxiao Li*[1] Minhyuk Sung*[1] Anastasia Dubrovina[1] Li Yi[1] Leonidas Guibas[1,2]
[1]Stanford University [2]Facebook AI Research

S.1. Numerical Stability Control

In our differentiable model estimator, we are solving two linear algebra problems: homogeneous least square and unconstrained least square. In both problems, numerical stability issues can occur.

We solve the homogeneous least square using SVD to find $\mathbf{v}$, the right singular vector corresponding to the smallest singular value. However, when backpropagating the gradient through SVD [1, 2], the gradient value goes to infinity when the singular values of the input matrix are not all distinct. In our case, such an issue happens only when the output segment (decided by membership matrix $\hat{\mathbf{W}}$) becomes degenerate. For instance, when fitting a plane to a segment via SVD, non-distinct singular values correspond to the case where the points in the segment with significant weights concentrate on a *line* or a *single point*. Hence if we get good segmentation by minimizing the segmentation loss (Section 3.3 in the paper), then such degenerate cases should not happen. Thus, we handle the issue by simply bounding the gradient in the following way. We implemented a custom SVD layer following [1, 2], and when computing $K_{ij} = \frac{1}{\sigma_i - \sigma_j}$ in Equation 13 of [1] where σ_i, σ_j are singular values, we instead use $K_{ij} = \frac{1}{\text{sign}(\sigma_i - \sigma_j)\max(|\sigma_i - \sigma_j|, \epsilon)}$ for $\epsilon = 10^{-10}$.

When solving the unconstrained least square using Cholesky factorization, numerical unstability can happen even when the segmentation is correct, but the type used in the estimator does not match with the segment. For instance, when fitting a sphere to a segment that is almost a *flat plane*, the optimal sphere is the one with center at infinity. To deal with such a singular case (as well as cases when the segments are degenerate), we add a l_2-regularizer to the formulation (Equation 7 in the paper) and solve instead

$$\min_{\mathbf{c}\in\mathbb{R}^3} \|\text{diag}(\mathbf{w})(\mathbf{X}\mathbf{c} - \mathbf{y})\|^2 + \lambda\|\mathbf{c}\|^2, \qquad (1)$$

with $\lambda = 10^{-8}$. Even with such a modification, Cholesky factorization can still become unstable when the condition number of $\text{diag}(\mathbf{w})\mathbf{X}$ is too large, where the condition number of a matrix is defined to be the ratio of its largest singular value over its smallest singular value. To deal with this, we trivialize the least square problem when the condition number is larger than 10^5 by setting $\mathbf{X} = \mathbf{0}$ to prevent gradient flow.

*equal contribution

S.2. Training Details

We use the default hyperparameters for training PointNet++ [3] with a batch size of 16, initial learning rate 10^{-3}, and staircase learning decay 0.7. All neural network models in the experiments are trained for 100 epochs, using Adam optimizer. The longest experiment (SPFN and its ablation studies) took 50 hours to train on a single Titan Xp GPU, although the decay of the total loss was not substantial after 50 epochs. We will release our source code and include a link to the code in the final version.

S.3. DPPN Architecture

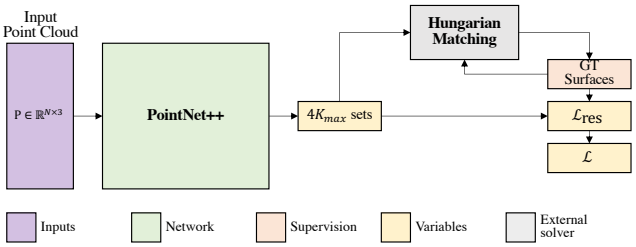

Figure S1: DPPN architecture.

The output of DPPN is simply a collection of $4K_{\text{max}}$ primitives including K_{max} planes, K_{max} spheres, K_{max} cylinders, and K_{max} cones. In order to compare with SPFN outputs, as a post-processing step, we construct auxiliary membership matrix $\hat{\mathbf{W}}$ by assigning each input point to the closest primitive among the $4K_{\text{max}}$ predicted primitives. Similarly, we construct per-point type matrix $\hat{\mathbf{T}}$ by assigning the type of each point to be the type of its closest primitive. The numbers reported in Table 1 in the paper are computed in the same evaluation pipeline as in SPFN after such post-processing step.

Ind	Method	Seg. (Mean IoU)	Primitive Type (%)	Point Normal (°)	Primitive Axis (°)	$f\mathbf{S}_k g$ Residual Mean	$f\mathbf{S}_k g$ Residual Std.	$f\mathbf{S}_k g$ Coverage $\epsilon = 0.01$	$f\mathbf{S}_k g$ Coverage $\epsilon = 0.02$	$\mathbf{P}$ Coverage $\epsilon = 0.01$	$\mathbf{P}$ Coverage $\epsilon = 0.02$
1	SPFN (Row 13 in Table 1)	77.14	96.93	8.66	1.51	0.011	0.131	86.63	91.64	88.31	96.30
2	SPFN, $64k$ test input	**77.29**	**97.27**	**8.50**	1.49	**0.010**	**0.126**	**87.03**	**91.87**	**89.01**	**96.42**
3	SPFN, w/ outliers	72.38	95.94	9.67	1.97	0.015	0.147	82.57	88.78	79.75	88.44
4	SPFN, $K_{\text{max}} = 48$	76.30	96.55	8.69	**1.39**	0.011	0.134	85.77	90.52	88.09	95.42

Table S1: Results of additional experiments described in Section S.5. First row is the same as row 13 in Table 1 in the paper. See Section 4.2 in the paper for the details of evaluation metrics. Lower is better in 3-5$^{\text{th}}$ metrics, and higher is better in the rest.

S.4. Primitive Correspondences

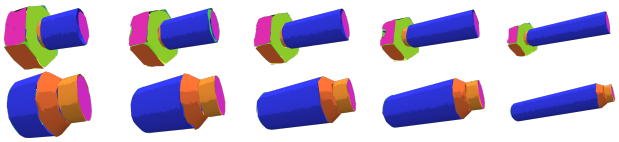

Figure S2: Output primitives of SPFN. Colors indicating column indices in $\hat{\mathbf{W}}$ are consistent across shapes in the same category.

Tulsiani *et al*. [5] and Sung *et al*. [4] introduced a type of neural networks capable of discovering correspondences across different inputs without direct supervision. Notice in SPFN, changing the ordering of the columns in $\hat{\mathbf{W}}$ does not affect the loss. Despite such ambiguity, SPFN implicitly learns a preferred order such that the primitives represented by the same columns in $\hat{\mathbf{W}}$ in different shapes appear to be similar, resulting in rich correspondence information for primitives from different shapes (Figure S2). These results provide insight into the possible design variations for the same category of shapes.

S.5. Additional Experiments

To further study the capability of SPFN, we have conducted the following additional experiments.

PointNet++ [3] used in our architecture has a limitation of handling high resolution point clouds during training time due to the increase of memory consumption. However, it is also known that PointNet++ is robust to the change of the resolution of point clouds at test time (See Section 3.3 in [3]). Hence, we can consider processing high resolution input point clouds in the test time by training the network with lower resolution point clouds. In Table S1, row 2 shows the results of testing $64k$ point clouds with the same SPFN model in Table 1 in the paper (trained with $8k$ point clouds), and it exhibits a slight improvement in nearly all metrics. We also assessed SPFN by adding not only noise in the inputs (as described in Section 4.1 in the paper) but also outliers. Row 3 describes the results when we add 10% outliers, which are uniformly sampled in space outside of the central cube $[-0.5, 0.5]^3$, in both training and test data. The results show a little drop but still comparable performance. Lastly, we also investigated how robust SPFN can be if we change the maximum number of primitives, K_{max}. Row 4 illustrated the results when training the network with $K_{\text{max}} = 48$, and we observed no substantial difference in performance.

References

[1] Catalin Ionescu, Orestis Vantzos, and Cristian Sminchisescu. Matrix backpropagation for deep networks with structured layers. 2015. 1

[2] Catalin Ionescu, Orestis Vantzos, and Cristian Sminchisescu. Training deep networks with structured layers by matrix backpropagation. *CoRR*, abs/1509.07838, 2015. 1

[3] Charles Ruizhongtai Qi, Ly Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NIPS*, 2017. 1, 2

[4] Minhyuk Sung, Hao Su, Ronald Yu, and Leonidas Guibas. Deep functional dictionaries: Learning consistent semantic structures on 3D models from functions. In *NIPS*, 2018. 2

[5] Shubham Tulsiani, Hao Su, Leonidas J. Guibas, Alexei A. Efros, and Jitendra Malik. Learning shape abstractions by assembling volumetric primitives. In *CVPR*, 2017. 2