SPSR Parameter determined by Neural Network

Florian Steinschorn* TU Wien
Philipp Erler† TU Wien

Abstract

CR Categories: I.2.6 [Artificial Intelligence]: Learning—Parameter learning;

Keywords: machine learning, neural network, parameter optimization

1 Introduction

In 'Parameter Optimization for Surface Reconstruction' ([Steinschorn et al. 2025]) we tested different parameter optimization methods to find the most accurate and fast way to find optimal parameters for longer-running tasks that cannot be exhaustively tested. One strategy that we did not test is using machine learning to solve this problem. If it is possible to train a neural network to determine close-to-optimal parameters for a given task, then that would certainly be faster than all the other tested solutions.
For this report, we generated training data for the problem of reconstructing meshes from point clouds using Screened Poisson Surface Reconstruction (SPSR). We used ParamILS ([Hutter et al. 2009]) for this, and then tested two different networks to see if this is achievable. We describe our strategy for this, present our results, and discuss the encountered problems.

2 Related Work

The reconstruction algorithm we try to optimize is Screened Poisson Surface Reconstruction by Kazhdan and Hoppe ([Kazhdan and Hoppe 2013]). It is used to generate watertight surfaces from point clouds by first converting them into a vector field, which can then be split up by its sign into inside and outside the model. The border between these two sets is then the surface of the model.
SPSRs advancement over standard Poisson Surface Reconstruction is that it also adds a term to the error function that punishes results that are farther away from the sampling points, therefore resulting in a more accurate fit to the input points. Another difference to PSR is that SPSR uses a different boundary function, which ensures water tightness. This does, however, pose the disadvantage that non-watertight models (like many 3D scans that do not include a bottom scan) can be blown out and connected to the edge of the reconstruction space.

In this report, we show tests using two different neural networks. The first one is PointNet by Qi et. al. ([Qi et al. 2017]). The authors discuss the usage of PointNet to classify point clouds, or label parts of them by partition or semantic differences. PointNet solves the problem of having to be perturbation resistant, due to the nature of the point cloud data, by including a max pooling layer after some point-level feature extraction steps. The global feature descriptor obtained this way is then also used for point-level labeling for segmentation.

The second neural network that we tested is Point Convolution for Surface Reconstruction (Poco) by Boulch and Marlet ([Boulch and Marlet 2022]). In the first step, they generate a latent feature vector for each input point. This information is then later used to determine whether a query point is inside or outside the object. For each query point, a fixed-size neighborhood of input points is selected and their feature vectors are used to decide on the probabilities of the query point being inside or outside the object.

3 Data Generation

To train the neural network, a lot of input data is needed, for our case, this means point clouds and their corresponding optimal parameters. We chose the ABC dataset ([Koch et al. 2019]) for this. To generate 'ground truth' optimal parameters for each point cloud, we used ParamILS ([Hutter et al. 2009]) as described and implemented in ([Steinschorn et al. 2025]) with a starting sample size of 70, a perturbation distance of 0.6 and 5 maximal repetitions.

3.1 Little Big Data

To speed up this resource-expensive step, we used the 'Little Big Data' cluster (LBDv3)[1] provided by the TU Wien dataLAB. This cluster contains 20 worker nodes each equipped with 2 Intel Xeon E5-2650v4 processors, which have 12 cores with hyperthreading. For these 48 available threads, each machine has 256GB of RAM. With this setup, we managed to compute a close-to-optimal parameter set for 1079 point clouds.

While LBDv3 was a great asset, without which we would not have been able to generate that amount of data, we also had some problems with it, which cost us more time than anticipated.

3.1.1 Documentation

The first problem with using LBDv3 was getting started. There is not a lot of documentation aside from basic infrastructure. While there are some example applications, they are all based on Jupyter notebooks and did not apply to our system, which needs to start spark calls from within Python.
A second problem was getting the dashboard to run, which allows the user to monitor running processes and get error messages. While there was some documentation, it did not work. The dataLAB team was fortunately very helpful with this and managed to help us fix this problem.

3.1.2 Environment Updates

There were two instances of environment updates on the LBDv3 cluster, that introduced breaking changes to our setup without being announced or documented.

*e-mail: f.steinschorn@gmail.com
†e-mail: perler@cg.tuwien.ac.at

[1] https://colab.tuwien.ac.at/display/DBD/Big+Data+LBD

At one point, there were some library updates that required code changes on our side. This caused a lot of idle time on the cluster until we figured out what was happening.
Another time, the dashboard mentioned above stopped working. Neither the available documentation nor the provided solution by the dataLAB team has worked since. Because it happened quite late in the data generation process, this did not cause any major issues, since the dashboard was no longer needed for debugging.

3.1.3 Other

Apart from those fixable issues, we also had the problem that our processes would regularly crash without any helpful error messages after many hours of running. We do not know what caused this issue, but it meant that we had to closely monitor the data generation process and regularly restart it.

4 Implementation

In this section, we describe how we used each of the two systems, what we had to change, and how well our implementation worked.

4.1 PointNet

For PointNet we used the Python implementation provided by the authors [2] as a starting point. Very few modifications were necessary to run this code on our data. We modified the classification net by increasing the output vector to length 7 (the number of parameters relevant for SPSR). We had to add a custom data loader for our ABC data including the calculated ground truths. All our changes are available on a fork of the PointNet repository [3].
We kept most settings as they are for the classification configuration, retaining Adam Optimizer and the set learning rates for example, but we changed the loss function to the mean square error between the prediction and the ground truth.
We used 80% of our 1079 point clouds as training data and 20% as test data and trained for 500 epochs.

4.2 Poco

For our tests with the Poco network, we used the implementation by Erler et al. for PPSurf ([Erler et al. 2024]) [4]. Similar to the PointNet implementation, we had to add our own data loader to read ABC data including our calculated ground truths. We modified the model to use the mean square error between prediction and ground truth as a loss function.
We had to modify the way Poco uses query points. The standard behavior is to choose many query points, and for each determine the probability of being inside or outside the model. This means the output of the network is a vector of length 2 per query point. To get only one output vector, instead of multiple query points, we only use one. This point is placed in the center of the scene ([0, 0, 0]). To still get global information, we extended the neighborhood used for this query point to include all available input points. Additionally, we increased the size of the output vector from 2 (inside and outside probability) to 7 (number of parameters for SPSR). All our changes are available on a fork of the PPSurf repository [5].
We used 90% of our 1079 point clouds as training data and 10% as test data and trained for 500 epochs.

[2] https://github.com/fxia22/pointnet.pytorch
[3] https://github.com/thefloff/pointnet.pytorch.spsr_parameters/tree/spsr
[4] https://github.com/cg-tuwien/ppsurf
[5] https://github.com/thefloff/ppsurf_param_opt

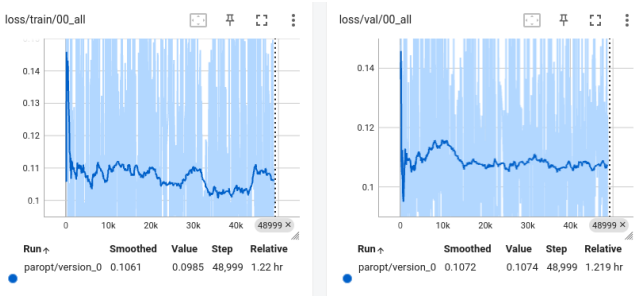

Figure 1: Loss curves during training of the Poco network for, training on the left and validation on the right.

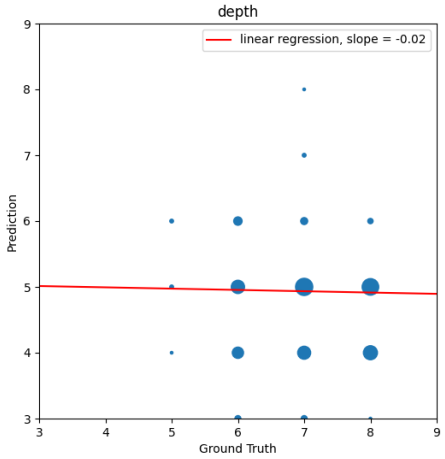

Figure 2: Correlation between ground truth and PointNet predicted outputs for the parameter 'depth'. The size of the dot represents the number of occurrences.

5 Results

To evaluate our results we use the average mean square error (MSE) of the test set, as well as the correlation between ground truths and predictions. This is represented by the slope of the linear regression, which should be 1.0 for a perfect prediction.
Unfortunately, none of our results show strong correlations between ground truth parameters and predicted parameters. Another indication for this, are the loss curves during training, which do not converge. An example of this can be seen in Figure 1. The loss values in this graph is calculated on the normalized feature vector and is therefore smaller than the MSE presented in the following sections.

5.1 PointNet

The results created using the PointNet implementation are the worst of the two networks. It achieved an MSE of 4.5.

It looks like the network managed to learn a generally good output, that outperforms the default parameters of SPSR but did not manage to catch differences between the individual point clouds well. As an example, Figure 2 shows the correlation between the ground truth and the prediction or depth values output by PointNet. A somewhat better result has been achieved for samples_per_node, as can be seen in Figure 3, but even here there is no clear correlation between ground truth and prediction.

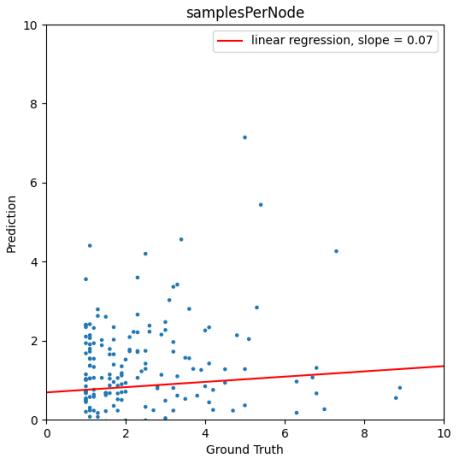

Figure 3: Correlation between ground truth and PointNet predicted outputs for the parameter 'samples_per_node'. The size of the dot represents the number of occurrences.

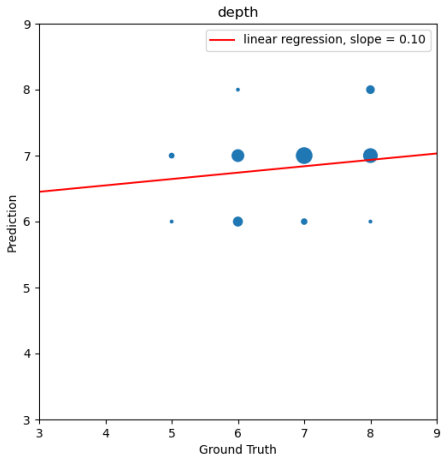

Figure 4: Correlation between ground truth and Poco predicted outputs for the parameter 'depth'. The size of the dot represents the number of occurrences.

5.2 Poco

In general, the achieved predictions by Poco more closely resemble the distribution of input ground truths. There is, however, still no clear correlation for individual point clouds. The achieved MSE of 1.42 is much smaller than for PointNet.

As can be seen in Figure 4, the network correctly learned the general range of good values, but there is no clear correlation between ground truth and prediction. It is a similar situation for samples_per_node, where the network learned a general range of expected values, but does not seem to consider the actual point cloud. Figure 5 shows this and it can be seen that especially higher ground truth values are completely disregarded.

6 Conclusion

We have to conclude, that our approaches did not yield the results we were hoping for. There are a few possible reasons for this.

6.1 Problems with Dataset

Our dataset of 1079 point clouds may be simply too small. Although it would have been great to have more data, generating more optimal parameters for point clouds was not possible due to time and resource constraints.
Another possibility is that the ABC dataset is simply too uniform and the individual point clouds too similar. The dataset consists of only geometric shapes, as can be seen in Figure 6, which shows a random sample of the dataset.

6.2 Problems with Ground Truth Parameters

It is possible that the limits used for generating the ground truth were not optimal. Figure 7 shows the distribution of ground-truth values for 'point_weight'. As can be seen here, a lot of values are the minimum of 2. It might be the case, that the actual optimum for this parameter could be 1 in some cases, therefore causing our ground truths to contain some wrong values. Similarly, 'depth' often causes better results the higher it is chosen, but we limited it to 8, since the computation time for SPSR increases dramatically

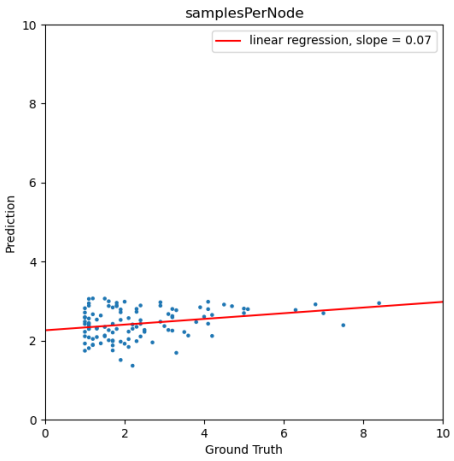

Figure 5: Correlation between ground truth and Poco predicted outputs for the parameter 'samples_per_node'. The size of the dot represents the number of occurrences.

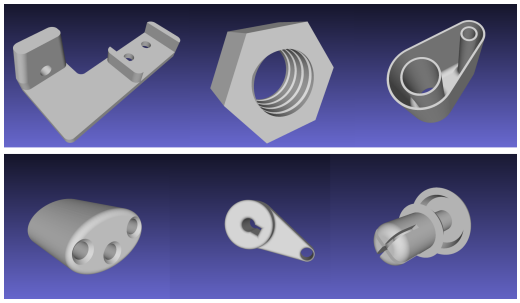

Figure 6: Randomly selected meshes from the ABC dataset.

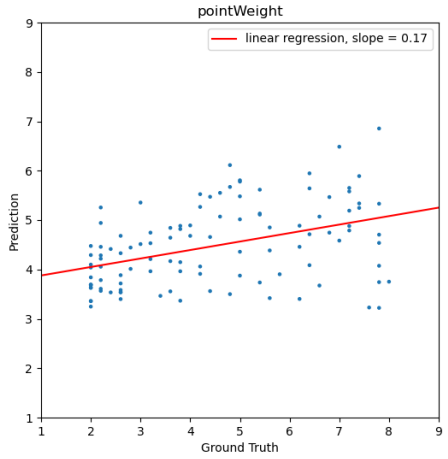

Figure 7: Correlation between ground truth and Poco predicted outputs for the parameter 'point_weight'. The size of the dot represents the number of occurrences.

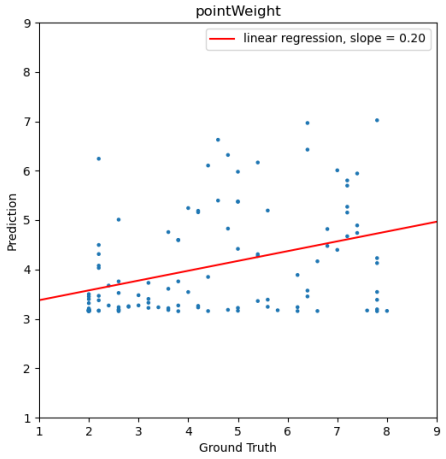

Figure 8: Correlation between ground truth and Poco predicted outputs for the parameter 'point_weight' when only using this parameter in the loss function. The size of the dot represents the number of occurrences.

above that.
It is also necessary to keep in mind that the ParamILS algorithm used when generating the ground truths yields a very good, but not necessarily the best possible parameter configuration. Especially, if some parameters are less important for the quality of the result, they may be farther off from the optimum. The importance of the parameter is not considered in the loss function used in our tests.
Another issue is that all parameters get normalized to the range [0.0, 1.0] during training, but the number of options is different per parameter. So a difference of 1 in 'cgDepth', which has a range of [0, 1] is an error of 1, while a difference of 1 in samples_per_node, with a range of [1, 10] in 0.1 steps, only causes an error of 0.01.

To eliminate the contamination of different scales and the importance of parameters, we also ran Poco with just one parameter used in the loss function. The result of this for the parameter 'point_weight' can be seen in Figure 8. While the slope of the linear regression is slightly steeper, the result is still not satisfying. We also do not get any output value under 3, although there are quite a lot of inputs under 3.

References

BOULCH, A., AND MARLET, R. 2022. Poco: Point convolution for surface reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 6302–6314.

ERLER, P., FUENTES-PEREZ, L., HERMOSILLA, P., GUERRERO, P., PAJAROLA, R., AND WIMMER, M. 2024. Ppsurf: Combining patches and point convolutions for detailed surface reconstruction. *Computer Graphics Forum 43*, 1, e15000.

HUTTER, F., HOOS, H. H., LEYTON-BROWN, K., AND STÜTZLE, T. 2009. Paramils: An automatic algorithm configuration framework. *J. Artif. Int. Res. 36*, 1 (Sept.), 267–306.

KAZHDAN, M., AND HOPPE, H. 2013. Screened poisson surface reconstruction. *ACM Transactions on Graphics (TOG) 32*, 3, 29.

KOCH, S., MATVEEV, A., JIANG, Z., WILLIAMS, F., ARTEMOV, A., BURNAEV, E., ALEXA, M., ZORIN, D., AND PANOZZO, D. 2019. Abc: A big cad model dataset for geometric deep learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

QI, C. R., SU, H., MO, K., AND GUIBAS, L. J., 2017. Pointnet: Deep learning on point sets for 3d classification and segmentation.

STEINSCHORN, F., WIMMER, M., AND ERLER, P., 2025. Parameter optimization for surface reconstruction.