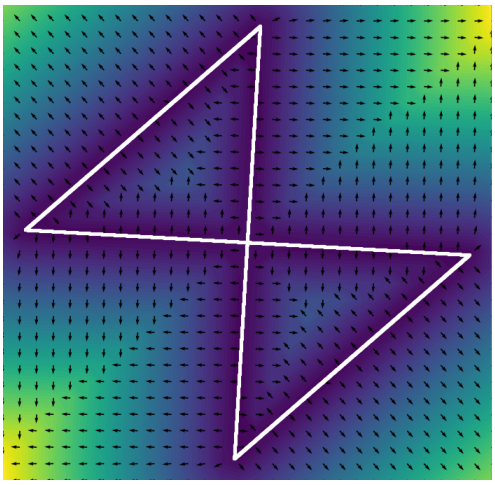
NPM3D Project

Neural Unsigned Distance Fields
for implicit function learning
Authors: Julian Chibane, Aymen Mir and Gerard Pons-Moll



Lucas MARANDAT

March 23, 2023

1 Introduction

1.1 Context

Cloud points research often involves reconstructing continuous and renderable surfaces from unstructured and incomplete 3D point-clouds, which is a fundamental challenge. One promising approach to tackle this problem is implicit function learning (IFL), which involves learning the surface by classifying continuous points as inside or outside the surface. While this approach enables continuous surface evaluation, it requires a closed surface, which is not always practical for objects like cars, wires, or thin details.

1.2 Author's contributions

To address this challenge, the authors propose a new technique called Neural Distance Fields (NDF). Instead of classifying points as inside or outside the surface, NDF uses a neural network to predict the unsigned distance between the point and the surface. With this approach, the authors demonstrate the ability to extend previous approaches to non-closed surfaces, including generating dense point-clouds from sparse ones, computing normals/meshes, and rendering point-clouds as images.

1.3 Scope of this report

In this report, I provide a clear description and explanation of the main contributions made by the authors, which include two algorithms for generating dense point-clouds and computing normals/rendering point-clouds. Additionally, I provide a detailed analysis of the architecture used by the authors, which is based on IF-Nets [CAPM20]. Furthermore, I present the results of various experiments that justify different choices that the authors made or could have made. Throughout this report, the focus is primarily on the reconstruction of point-clouds, which was the main contribution of the authors.

2 Shape Reconstruction

2.1 Limitations of Previous Approaches

Reconstructing surfaces from point clouds is a challenging task. Various methods have been proposed to address this issue, but each has its own limitations:

- *Voxel*-based methods use voxels as input and output. This results in fixed resolution and exponentially increasing memory requirements.
- *Mesh*-based methods deform a mesh from a mean shape to approximate the true shape. However, they are limited to a single topology and do not guarantee surface continuity.
- *Point-clouds*-based methods typically use a combination of points and voxels. However, they still suffer from fixed output resolution limitations.
- *Implicit Function Learning*-based methods (IFL) predict if a given point is inside or outside the shape. Thus, they are limited to closed shapes only.

Unfortunately, none of these approaches can simultaneously handle different topologies, generate high-dimensional point clouds with arbitrary resolution and ensure surface continuity. The authors focus their analysis on the learning-based methods.

2.2 Problem reformulation

In order to tackle the surface reconstruction problem, one approach that has been found to be appealing is the use of distance fields. These fields are represented as functions that assign a distance to every point in space in relation to a specific surface. They enable a concise and efficient representation of geometric shapes and their characteristics. In formal terms, distance fields can be defined as follows:

$$UDF(p, \mathcal{S}) = min_{q \in \mathcal{S}} \|p - q\| \tag{1}$$

Here, $p \in \mathbb{R}^d$ is a point and $\mathcal{S} \subset \mathbb{R}^d$ denotes the surface in question. Finding $UDF_{\mathcal{S}}$ can be challenging, thus previous approaches use the Implicit Function Learning (IFL) methods. The IFL methods consist of a function f which either predicts if a given point is inside or outside a given shape ($f : \mathbb{R}^d \times \mathcal{S} \longrightarrow [0, 1]$) or predicts the signed distance using a Signed Distance Function (SDF) of a point to a given shape, where negative distance means the point is inside the shape ($f : \mathbb{R}^d \times \mathcal{S} \longrightarrow \mathbb{R}$). However, as mentioned before, IFL still lacks the ability to handle non-closed shapes.

To tackle this issue, the authors proposed to reformulate the SDF approach to predict the **unsigned** distance of a point to the surface, namely $UDF_{\mathcal{S}}$. By predicting the distance fields, they did not distinguish the inside or the outside of a shape, thus making the approach compatible with non-closed shapes. The problem can be reformulated as follows:

$$\min_f \mathbb{E}_{p,\mathcal{S}} \left\| f(p, \mathcal{S}) - UDF(p, \mathcal{S}) \right\|_k \tag{2}$$

2.3 Neural Distance Fields (NDF)

The paper's authors used a neural network to predict the unsigned distance of a point p to a given surface $\mathcal{S}$, which they named Neural Distance Fields (NDF). This approach simplifies the search for an optimal f in (2). You can see an illustration of NDF in figure 4.

2.3.1 Architecture

They describe an approach where they first encode a voxelized point cloud using a specific architecture (see fig 1). When a new point is introduced, it undergoes continuous encoding by grid sampling using the encoded voxelized cloud. The resulting encoded point is then passed through a neural network, which predicts the distance between the point and the shape represented by the voxelized point cloud.

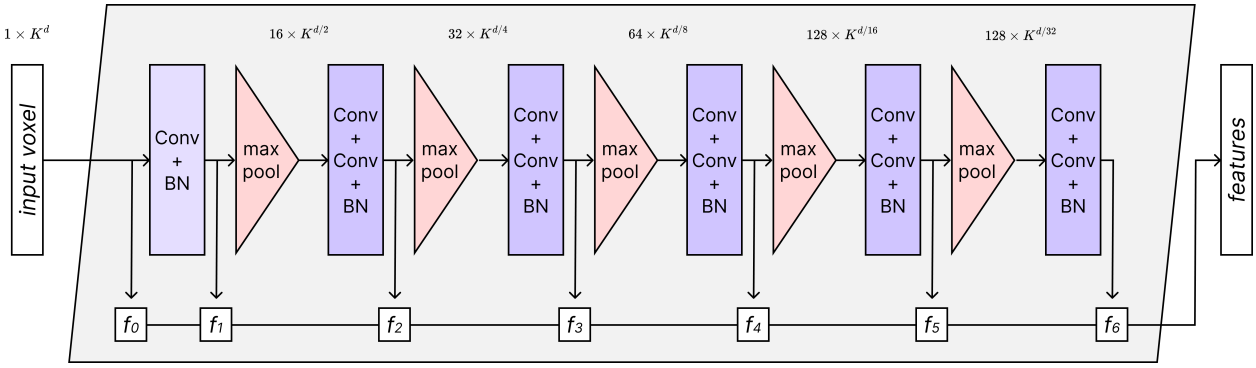


Figure 1: **Encoder** - The method processes a voxelized point cloud and generates 7 features from distinct layers. Each feature has a unique shape and captures distinct information.

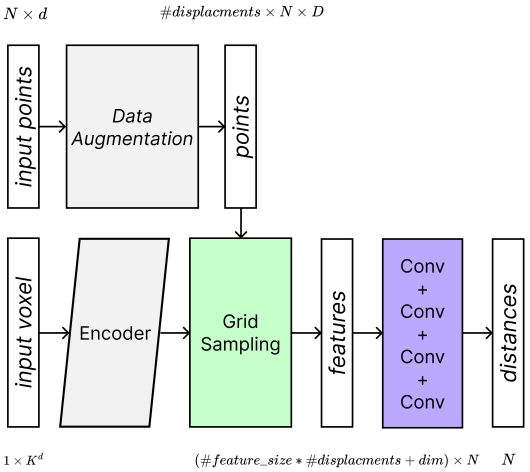
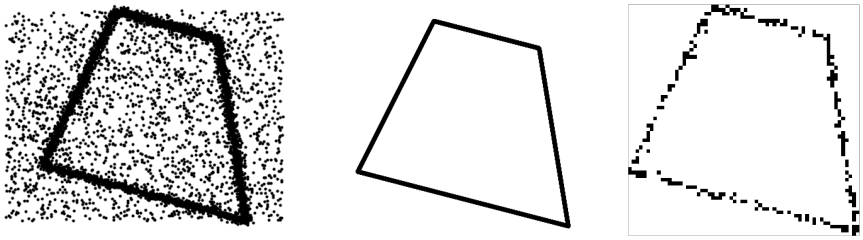


Figure 2: **NDF** (f_θ) - The method involves processing a voxelized point cloud along with additional input points. These input points are augmented by displacing them in multiple directions, such as $\{e_1, e_2, 0, -e_1, -e_2\}$ in 2D. Meanwhile, the voxelized point cloud is encoded using an **Encoder** (see Figure 1). The augmented points are then sampled from the encoded voxelized cloud, resulting in a continuous encoding for each point. Finally, a convolution operation is performed on the encoded points to generate a distance output.

2.3.2 Learning



(a) Cloud-points $\mathcal{P}_i$ (b) Surface $\mathcal{S}_i$ (c) Voxel $\mathcal{V}_i$

Figure 3: **2D Dataset example** ($\mathcal{D}_i$) - To gain a better understanding of the dataset, consider the following example. The left image displays a point cloud generated by sampling around the surface and uniformly inside the space. The middle image depicts the surface that we intend to approximate the distance fields for. The right image shows a voxelized version of the surface, with a resolution of 64 and some noise added to better handle real-world scenarios.

The learning process involves approximating the original objective defined in Equation 2 by sampling through a dataset $\mathcal{D} = \{\mathcal{P}_i, \mathcal{S}_i, \mathcal{V}_i\} i = 1^N$, where $\mathcal{P}_i$ represents a point cloud, $\mathcal{S}_i$ represents the corresponding surface, and $\mathcal{V}_i$ represents its voxelized version (see Figure 3). With this dataset $\mathcal{D}$, the learning loss can be expressed as:

$$\mathcal{L}_\mathcal{D}(\theta) := \sum_{i=1}^{N} \sum_{p \in \mathcal{P}_i} \| f_\theta(p, \mathcal{V}_i) - UDF(p, \mathcal{S}_i) \|_k \tag{3}$$

The authors of the original paper limit the maximum predicted distance to a value of $\delta > 0$, which improves the accuracy of the model's representation of the surface. In this report, both versions will be explored and compared on a simpler task.

2.4 Algorithms

The authors of the paper proposed two primary applications of their Neural Distance Fields (NDF). The first is to densify a sparse point cloud, while the second is to determine the surface level along a given ray, which is useful for visualization purposes. Furthermore, by using the surface level, they demonstrate the ability to approximate surface normals. This presentation focuses on the first algorithm, which was used for the experiments. However, the second algorithm is also described, and the main intuitions behind it are presented.

2.4.1 Dense Point-Clouds

The approach for densifying a sparse point cloud involves first sampling uniform points in a given space. Then, we use the $-\nabla_{p_i} f_\theta(\mathcal{V}i, p_i)$ direction to move each point closer to the surface by minimizing the predicted distance to the surface. If $f_\theta = UDF$, then we should move at a step of $f_\theta(\mathcal{V}_i, p_i)$, which is the distance to the surface. However, in practice, $f_\theta \neq UDF$, so we use an iterative approach. While the original algorithm can be found in [CMPM20], a similar and simpler version is described below. During experiments on simpler tasks, it was found that this simpler version worked just as well, if not better than the one provided in the paper.

Algorithm 1: Dense CPs

Data: Voxelized version of the sparse cloud-points $\mathcal{V}$ and N
Result: Dense cloud-points $\mathcal{P}$ such that $|\mathcal{P}| = N$
$\mathcal{P} := \emptyset$;
$\tilde{\mathcal{P}} \sim Unif(-1, 1, N)$;
while $|\mathcal{P}| < N$ **do**
  $\tilde{\mathcal{P}} \leftarrow \tilde{\mathcal{P}} - f_\theta(\mathcal{V}, \tilde{\mathcal{P}}) \cdot \frac{\nabla_{\tilde{\mathcal{P}}} f_\theta(\mathcal{V}, \tilde{\mathcal{P}})}{\|\nabla_{\tilde{\mathcal{P}}} f_\theta(\mathcal{V}, \tilde{\mathcal{P}})\|}$; `// move points toward the surface`
  $\mathcal{P} \leftarrow \mathcal{P} \cup \{p \in \tilde{\mathcal{P}} | f_\theta(\mathcal{V}, p) < \delta\}$; `// store points which are closer than` δ `to the surface`
  $\tilde{\mathcal{P}} \leftarrow \tilde{\mathcal{P}} + \mathcal{N}(0, \delta)$; `// augment` $\tilde{\mathcal{P}}$ `to explore neighbourhood area`
end

2.4.2 Roots along ray

Finding exact roots along a ray can be challenging, so the authors propose an approximation method with a precision limit of $\delta = \sup_{p \in \mathcal{S}} f_\theta(\mathcal{V}, p)$, which corresponds to the maximum error of NDF on the surface $\mathcal{S}$. The algorithm is described in the original paper [CMPM20]. The main idea is to find the best λ along a given ray r and an initial point p_0 such that $f_\theta(\mathcal{V}, p_0 + \lambda r) < \delta + \epsilon$, where $\epsilon \in \mathbb{R}_{++}$ is the queried precision. This method allows us to approximate the roots along the direction r, and sampling around the root allows for an approximation of the normal of the root.

3 Experiments

The original paper includes the source code for reproducing the experiments, as well as links to the dataset. Although the dataset has a size of over 120Gb, the authors provide a smaller version that is 2Gb in size. Running one epoch of this smaller dataset takes approximately 20 minutes on a dedicated server with an NVIDIA RTX A6000. To conduct a more in-depth analysis of the NDF, I decided to create a lighter dataset consisting of 2D cloud-points. This reduced the epoch time to 30 seconds, while still providing a sufficiently large dataset of 1,000 different cloud-points, each containing 10,000 points.

3.1 Dataset

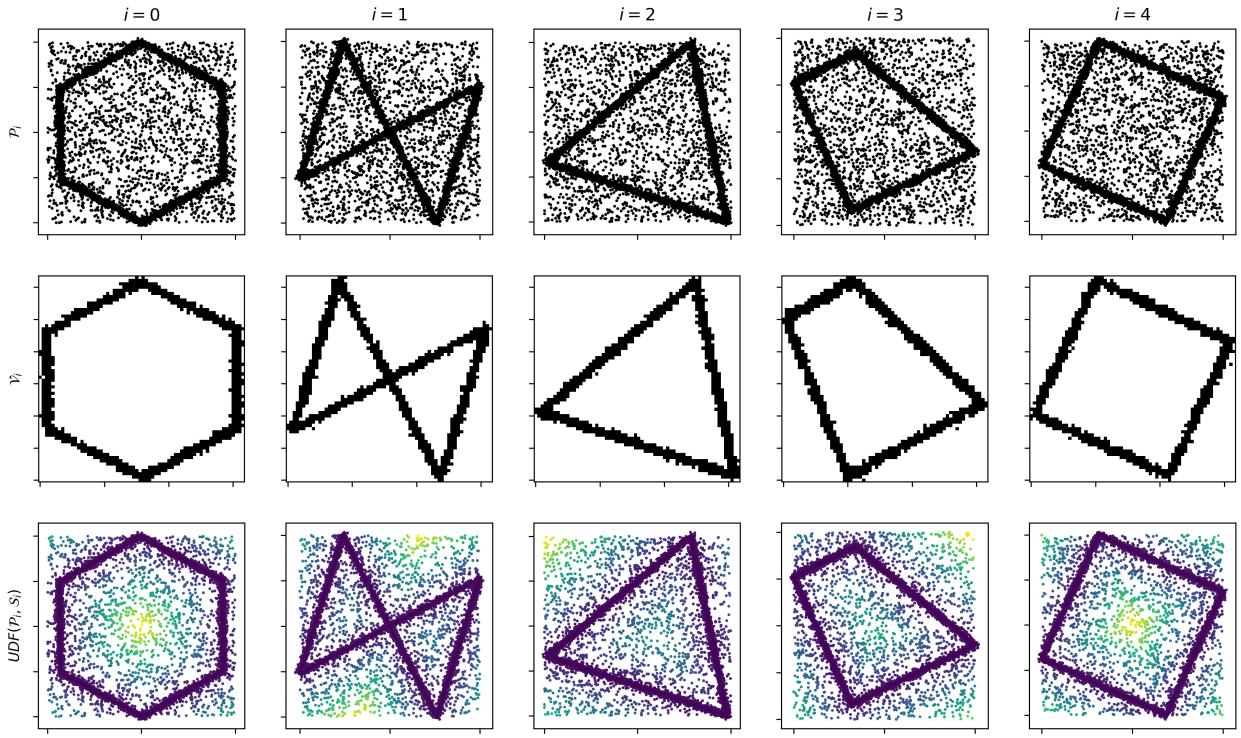


Figure 4: **Dataset** ($\eta_{unif} = .2$) - Consists of five samples of a generated dataset. In this dataset, η_{unif} is the proportion of uniform samples compared to samples around the targeted surface. In the loss function $\mathcal{L}_{\mathcal{D}}(\theta)$ (see Equation 3), $\mathcal{P}_i$ and $\mathcal{V}_i$ have the same meaning as before. The pre-computed UDF of each point $p \in \mathcal{P}_i$ to the surface $\mathcal{S}_i$ is represented by $UDF(\mathcal{P}_i, \mathcal{S}_i)$.

To create a simple yet interesting dataset, I generated random shapes from a set of six possible shapes: square, triangle, complex quadrilateral, trapezoid, pentagon, and hexagon. I then randomly rotated each shape to increase the number of unique shapes.

Using these shapes, I sampled cloud-points using various strategies. To balance between sampling around the surface shape and uniformly, I used a parameter η_{unif}. The authors of the original paper only sampled around the surface of the shape ($\eta_{unif} = 0$).

Additionally, the authors proposed different strategies for sampling around the surface of the shape by varying the variance σ_k to create a point $p \sim \mathcal{S}_i + \mathcal{N}(0, \sigma_k)$, which is drawn from a Gaussian distribution around the surface of the shape. I adopted the same strategy for my dataset.

3.2 Comparisons

To gain a better understanding of the impact of each decision, I conducted 30 different trainings using various generated datasets and loss functions. I then compared the results. Each comparison was conducted on a distinct dataset from the one used for training. The qualitative results were generated using the same initial cloud-points, and the displayed results were chosen randomly from the dataset.

3.2.1 Should we restrict the distance during training by clamping it?

TLDR; Yes, it seems to increase the accuracy.

They originaly implemented clamping tricks in their original loss function to "focus the model's capacity

on representing the vicinity of the surface." Throughout the training process, the clamped loss produced superior results compared to the original loss. For the remainder of the study, we will only compare the clamped version, as it provides more accurate results.

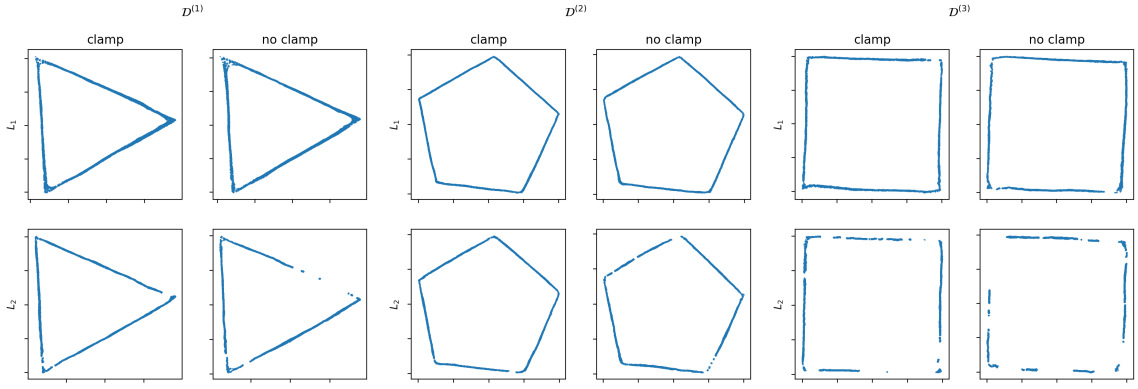


Figure 5: **Qualitative Results** - $\mathcal{D}^{(i)}$ refers to the first randomly selected item from dataset i.

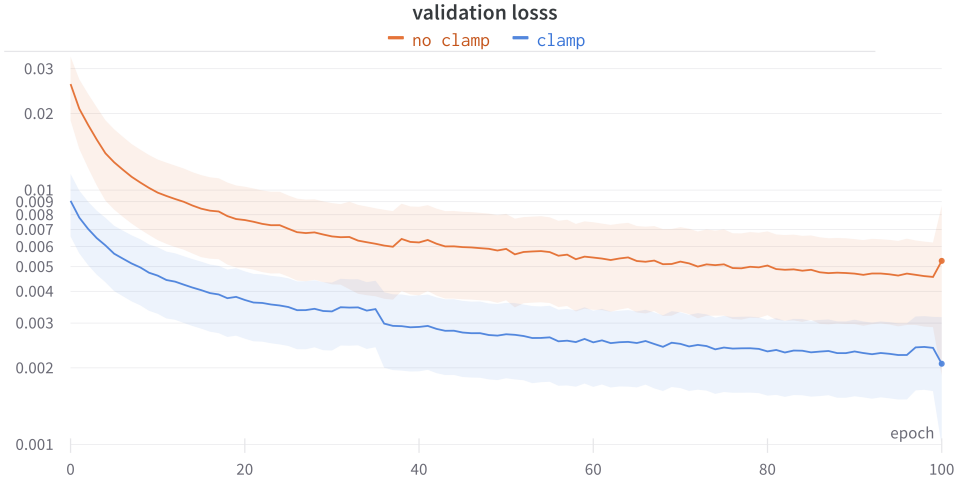


Figure 6: **Quantitative Results** - The plotted area represents the standard error of the runs.

3.2.2 Should we only sample cloud-points around the surface of the shape?

TLDR; Maybe not.

They initially utilized cloud-points sampled only around the surface of the shape. However, this approach seemed to introduce a significant bias, as the NDF did not encounter any examples that could be far from the initial surface. As Algorithm 1 1 use a uniformly sampled cloud-points, we introduce the possibility of some points being far from the surfaces in the training dataset, leading to a substantial performance improvement, as illustrated in the results. Henceforth, we set $\eta_{unif} = 0.2$.

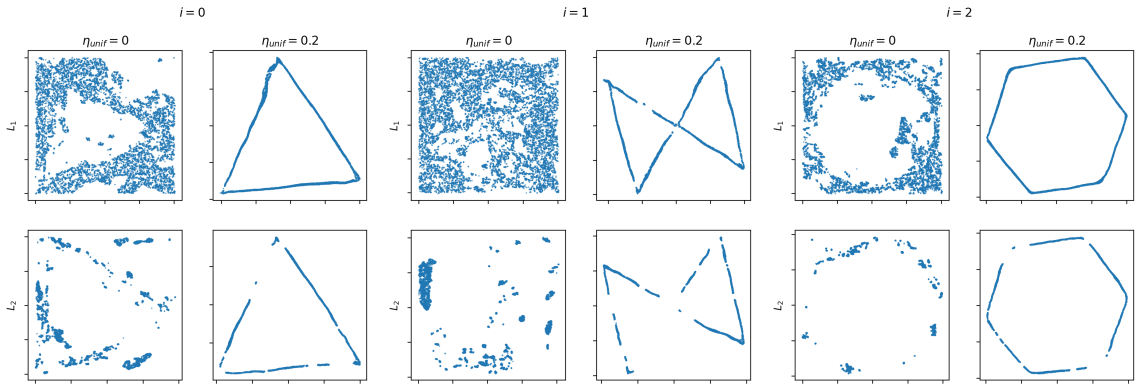


Figure 7: **Qualitative Results** - The index i corresponds to the i^{th} element of an unseen dataset. Note that $\eta_{unif} = 0$ implies that every generated cloud-point is sampled around the surface of the shape, while $\eta_{unif} = 0.2$ means that 20% of the cloud-points are uniformly sampled in the space.

3.2.3 Should we include the ground truth of the surface during training?

TLDR; It could improve accuracy.

The original paper utilized only sampled points in the learning process, which implies that the dataset is generated with a non-zero standard deviation ($\sigma_k > 0$) according to my definition. To investigate the impact of including ground truth in the dataset, I trained the model using two different sets of standard deviations: $\sigma = \{0.02, 0.003\}$ (without any ground truth present in the dataset, similar to the original paper) and $\sigma = \{0.02, 0.003, 0\}$ (with some ground truth included in the dataset). The experiments indicate that including some ground truth in the dataset leads to an improvement in the qualitative results on unseen point-clouds. However, the quantitative results using the ground truth in the dataset are slightly weaker, which could be attributed to the difficulty in predicting exactly zero versus predicting values near zero.

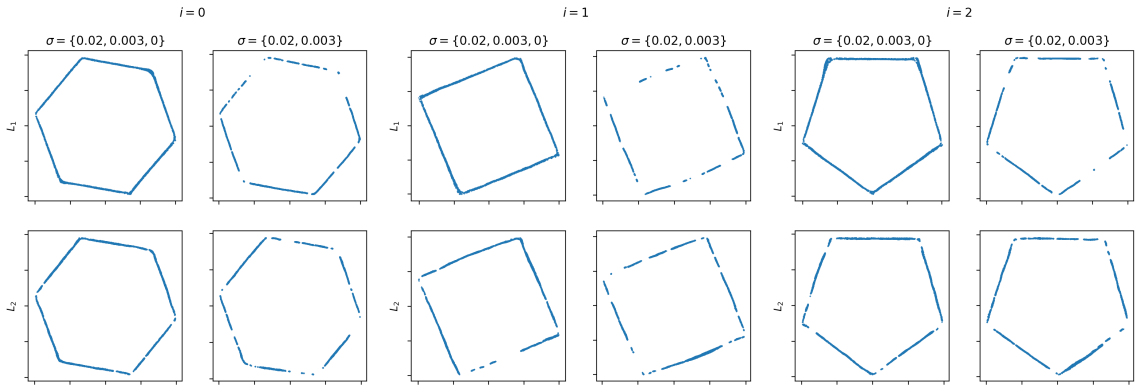


Figure 8: **Qualitative Results** - The index i corresponds to the i^{th} element of an unseen dataset.

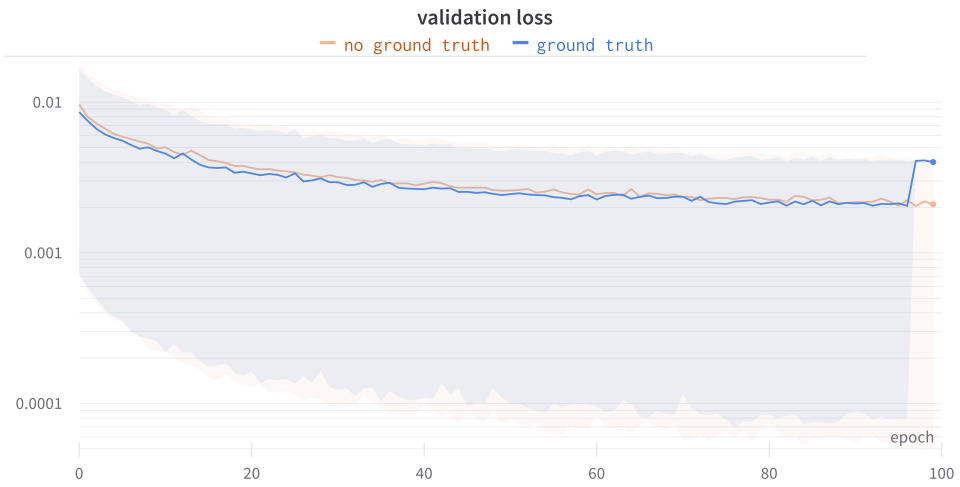
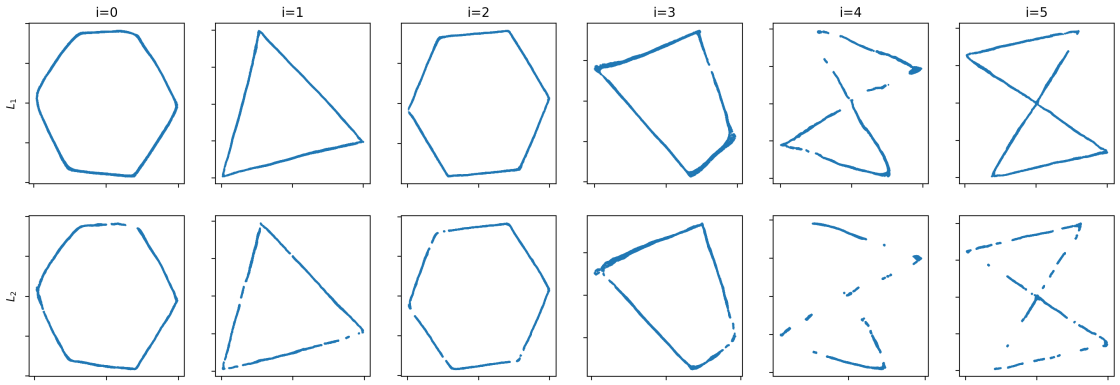


Figure 9: **Quantitative Results** - The plotted area represents the standard error of the runs.

3.2.4 Which metric should we use?

TLDR; Probably the L_1.

Initially, the authors employed the L_1 loss function for training the model, possibly to prevent penalizing the model too harshly for outliers. During the experiments, I tested both L_1 and L_2 loss functions, and while both resulted in similar quantitative results, the former yielded better reconstruction quality, as illustrated in the figure below.

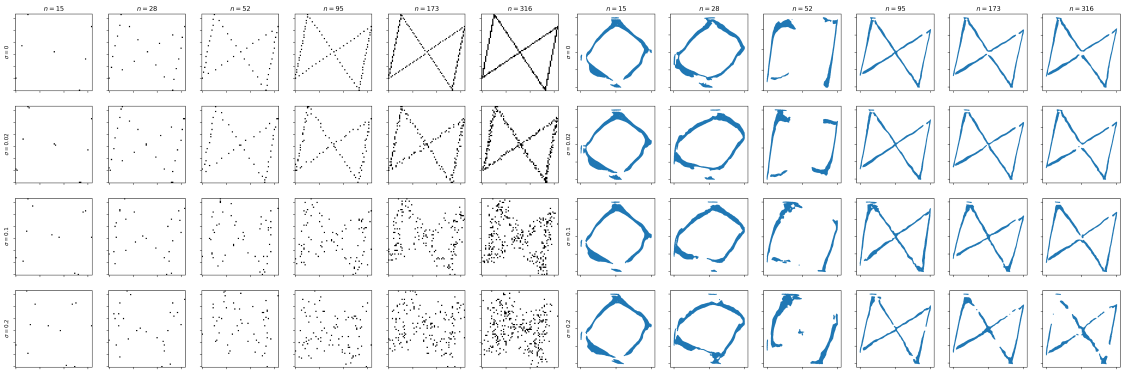


(a)

Figure 10: **Qualitative Results** - The index i corresponds to the i^{th} element of an unseen dataset.

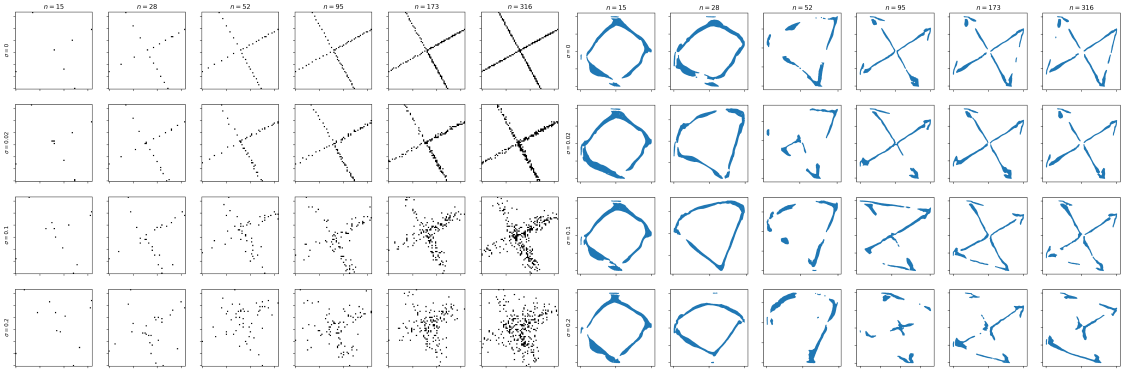
3.3 Performances

In order to evaluate the efficiency of the network, I conducted an experiment using a dataset consisting of 10,000 sparse point clouds, each containing 100 points. The clouds were generated using $\sigma = \{0.8, 0.02, 0.003, 0\}$ and a minimum resolution of 32, with η_{unif} set to 0.2. The network was trained for 60 epochs using the L_1 norm and the clamp tricks. I varied the input sample size and the type of noise applied to the data during testing, and utilized a reconstruction threshold of $\delta = 0.005$.



(a) Input (b) Prediction

Figure 11: **Qualitative Results** - The experiments on a known shape indicate that the network demonstrates promising results in terms of denoising and increasing the number of points in the point cloud.



(a) Input (b) Prediction

Figure 12: **Qualitative Results** - When tested on an unknown shape (in this case, a cross), the network was able to produce a shape that closely resembled the ground truth, despite not being trained on that particular shape.

4 Conclusion

In conclusion, this paper holds great promise, as it delivers state-of-the-art results while overcoming previous limitations. Although the experiments require considerable computation to reproduce, the network demonstrates the ability to handle previously unseen cases, validating the authors' claim of topological adaptation. The questions encountered during the experiments, particularly the use of the authors' pipeline for generating datasets, indicate that there is room for optimization. By re-coding the implementation, I was able to streamline the code and make it more readable. I also provided quantitative and qualitative comparisons of different formulations of the learning process. The network architecture schema I created aims to clarify the original paper and provide an overview of how NDF works.

Although my report raises new questions, such as testing the suggested improvements on the ShapeNet dataset or exploring the effects of different choices (such as the latent dimension or network architecture), I could not address them due to time and resource limitations. It may be worthwhile to investigate adding constraints on the regularity of the output of the network, as this could lead to faster and more effective learning, given that we are approximating the distance fields function.

References

[CAPM20] Julian Chibane, Thiemo Alldieck, and Gerard Pons-Moll. Implicit functions in feature space for 3d shape reconstruction and completion. *arXiv*, March 2020.

[CMPM20] Julian Chibane, Aymen Mir, and Gerard Pons-Moll. Neural unsigned distance fields for implicit function learning. *arXiv*, October 2020.