Learning Smooth Neural Functions via Lipschitz Regularization

HSUEH-TI DEREK LIU, University of Toronto, Canada
FRANCIS WILLIAMS, NVIDIA, USA
ALEC JACOBSON, University of Toronto & Adobe Research, Canada
SANJA FIDLER, University of Toronto & NVIDIA, Canada
OR LITANY, NVIDIA, USA

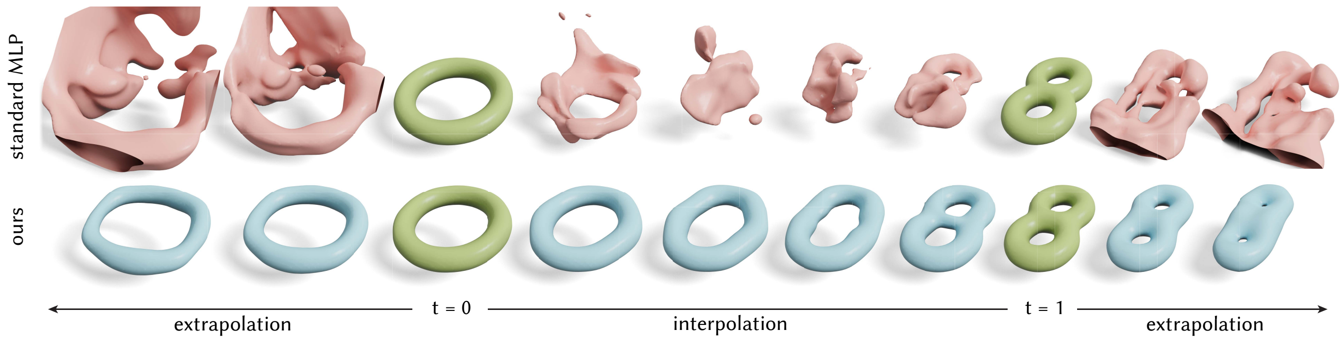


Fig. 1. We fit neural networks to the signed distance field of a torus when the latent code $t = 0$ and a double torus when $t = 1$ (green). Our Lipschitz multilayer perceptron (MLP) achieves smooth interpolation and extrapolation results (blue) when changing t, while the standard MLP fails (red).

Neural implicit fields have recently emerged as a useful representation for 3D shapes. These fields are commonly represented as neural networks which map latent descriptors and 3D coordinates to implicit function values. The latent descriptor of a neural field acts as a deformation handle for the 3D shape it represents. Thus, smoothness with respect to this descriptor is paramount for performing shape-editing operations. In this work, we introduce a novel regularization designed to encourage smooth latent spaces in neural fields by penalizing the upper bound on the field's Lipschitz constant. Compared with prior Lipschitz regularized networks, ours is computationally fast, can be implemented in four lines of code, and requires minimal hyperparameter tuning for geometric applications. We demonstrate the effectiveness of our approach on shape interpolation and extrapolation as well as partial shape reconstruction from 3D point clouds, showing both qualitative and quantitative improvements over existing state-of-the-art and non-regularized baselines.

ACM Reference Format:
Hsueh-Ti Derek Liu, Francis Williams, Alec Jacobson, Sanja Fidler, and Or Litany. 2022. Learning Smooth Neural Functions via Lipschitz Regularization. *ACM Trans. Graph.* 1, 1 (May 2022), 13 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Authors' addresses: Hsueh-Ti Derek Liu, University of Toronto, Canada, hsuehtil@cs.toronto.edu; Francis Williams, NVIDIA, USA, francis@fwilliams.info; Alec Jacobson, University of Toronto & Adobe Research, Canada, jacobson@cs.toronto.edu; Sanja Fidler, University of Toronto & NVIDIA, Canada, fidler@cs.toronto.edu; Or Litany, NVIDIA, USA, or.litany@gmail.com.

0730-0301/2022/5-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Neural Fields have become a popular representation for shapes in geometric learning tasks. A neural field is an implicit function encoded as a neural network which maps input 3D coordinates to scalar values (for example signed distances). In many tasks, these networks are conditioned on an additional *shape latent code* which is learned from a large corpus of shapes and acts as knob to deform the shape encoded by the neural field. Thus, smoothness with respect to the latent descriptor of a neural field is a desirable property to encourage well behaved deformations.

There are many traditional ways to encourage a function to posses some notion of smoothness. However we find that such classical approaches are not applicable to obtaining a smooth latent space of neural fields. For example in Fig. 2, we minimize the Dirichlet energy defined over the latent space, but the neural network still possesses non-smooth behavior outside the training set.

In this work, we focus on encouraging smoothness with respect to the latent parameter of a neural field. Since neural fields are continuous by construction, we use the *Lipschitz bound* as a metric for smoothness of the latent space. This notion of smoothness is defined over the entire space. Thus, it encourages smoothness even away from the training set (Fig. 1). While Lipschitz constrained networks have been proposed before (see Sec. 2), they are not readily applicable to geometric applications. In particular, they require predetermining the Lipschitz bound, which is not known in advance and highly input dependent (see Fig. 3). Therefore, to use prior Lipschitz architectures one has to perform extensive per-shape hyperparameter tuning to find a reasonable Lipschitz constant.

We therefore propose a novel *smoothness* regularizer to minimize a learned Lipschitz bound on the latent vector of a neural field. Our method is extremely simple and effective: one only needs to add

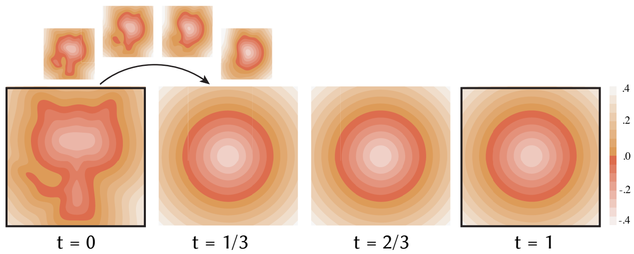


Fig. 2. We fit a multilayer perceptron to fit the signed distance functions (SDFs) of a cat shape and a circle at $t = 0$ and $t = 1$ respectively. In addition, we minimize the Dirichlet energy of t at $t = 1/3, 2/3$. While the network finds a smooth solution at those sample time steps, it still has non-uniform change beyond the samples, such as between $0 \leq t \leq 1/3$.

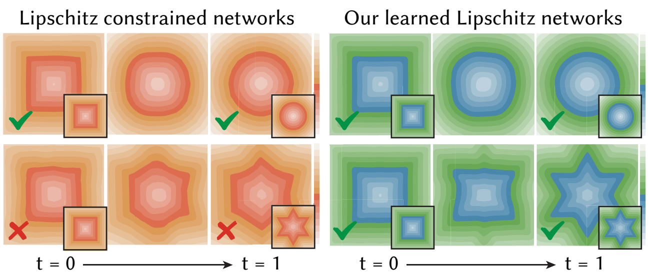


Fig. 3. Different tasks require different Lipschitz constants. We manually specify the same Lipschitz constant on the top two interpolation examples. One of them is sufficient (top left) to fit the ground truth (black), but the other one is not (bottom left). In contrast, our method learns a suitable Lipschitz constant for each task (right) with the same hyperparameter.

a weight normalization layer and augment the loss function with a simple regularization term encouraging small Lipschitz. Unlike previous approaches, our method can perform high quality deformations on latent spaces learned with as few as two shapes. We demonstrate the effectiveness of our method on the tasks of shape interpolation and extrapolation (Sec. 5.2), robustness to adversarial inputs (Sec. 5.1), and shape completion from partial point-clouds (Sec. 5.3), in which we oupterform past methods both qualitatively and quantitatively.

2 RELATED WORK

We focus our discussion on how learning-based methods encourage smoothness and methods similar in methodology. For an overview on neural fields, please refer to [Xie et al. 2021].

Geometric Regularizations. Many existing approaches rely on classic measures to encourage smoothness in neural 3D mesh processing. Several methods (e.g., [Hertz et al. 2020; Liu et al. 2019; Wang et al. 2018]) use Laplacian regularization which penalizes the difference between a vertex and the center of mass of its 1-ring neighbors. Kato et al. [2018] encourage smoothness by encouraging flat dihedral angle between adjacent faces. Hertz et al. [2020]; Wang et al. [2018] penalize edge-lengths and their variance. Rakotosaona and Ovsjanikov [2020] define an isometry regularization in character deformation to obtain area preserving interpolation. Many techniques

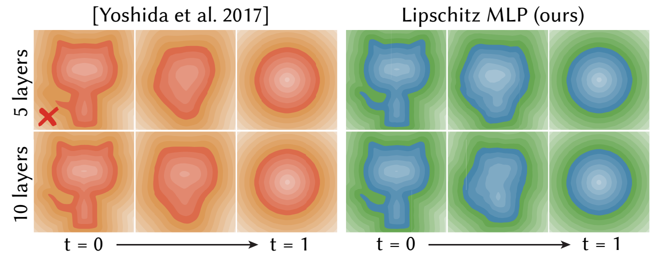


Fig. 4. The spectral norm regularization proposed by Yoshida and Miyato [2017] is more sensitive to the number of layers. Therefore, using the same α on a 5-layer and a 10-layer MLP leads to different effects (red). In contrast, our regularization (blue) leads to more consistent results.

(e.g., [Chen et al. 2019]) even use a mixture of these regularizations. However, these regularizations often require the input being a manifold triangle mesh. In other representations such as neural fields, regularizations of the level set surface are difficult to be defined. Previous works also introduced other geometric regularizations for encouraging different properties, such as [Gropp et al. 2020; Williams et al. 2021]. But we exclude our discussion on those techniques because they are not directly related to the smoothness of a network function.

Network Regularizations. Given input samples, one can differentiate through a network to obtain derivative information of the network output with respect to these inputs. Then we can encourage smoothness at these input samples by penalizing the norm of the Jacobian [Drucker and Le Cun 1991; Gulrajani et al. 2017; Hoffman et al. 2019; Jakubovitz and Giryes 2018; Varga et al. 2017] or the Hessian [Moosavi-Dezfooli et al. 2019]. If differentiating through the network is undesirable, Elsner et al. [2021] propose to penalize the difference in the output based on the input similarity. These techniques are effective in obtaining smooth solutions at training samples, but they have no guarantee to obtain a smooth function beyond them. On the contrary, it may even promote non-smooth behavior by squeezing function changes to locations without training samples (see Fig. 2).

In lieu of this, one should use regularization techniques that do not depend on the input to a network, such as penalizing L2 norm [Tihonov 1963] or L1 norm [Tibshirani 1996] of the weight matrices. Other training techniques can also be used to regularize the network, such as early-stopping [Ulyanov et al. 2018; Williams et al. 2019], dropout [Srivastava et al. 2014], and learning rate decay [Li et al. 2019b]. Applying these techniques can alleviate overfitting, but how they relate to the smoothness of a network function remains an open problem. In our experiments, they produce less smooth results compared to our method (see Table 1).

Lipschitz Regularizations. The Lipschitz constant of neural networks has attracted huge attention because of its applications in robustness against adversarial attacks [Li et al. 2019a; Oberman and Calder 2018], better generalization [Yoshida and Miyato 2017], and Wasserstein generative adversarial networks [Arjovsky et al. 2017]. Several techniques have been proposed to precisely constraint the Lipschitz constant of a network. Miyato et al. [2018] normalize the weight matrices by dividing each weight matrix by its largest eigenvalue. This *spectral normalization* enforces a neural network to be 1-Lipschitz, a neural network with Lipschitz bound 1, under the L2 norm. Gouk et al. [2021] rely on different weight normalization methods to constrain the Lipschitz bound under L1 and L-infinity norms. Anil et al. [2019]; Cissé et al. [2017] obtain 1-Lipschitz networks by orthonormalizing each weight matrix. Strictly constraining the Lipschitz constant of a network is not always desirable because it may lead to undesired behavior in the optimization [Gulrajani et al. 2017; Rosca et al. 2020]. In response, Terjék [2020] propose a regularization to softly encourage a network to be c-Lipschitz. However, these Lipschitz constrained networks often fail to achieve the prescribed Lipschitz bound because the estimated Lipschitz bound is not tight due to the ignorance of activation functions. Several papers complement this subject by proposing more accurate methods to estimate the true Lipschitz constant, such as [Jordan and Dimakis 2020; Virmaux and Scaman 2018; Weng et al. 2018]. Anil et al. [2019] propose a new activation function based on sorting to tighten the estimated Lipschitz bound. Unfortunately, the above-mentioned Lipschitz constrained networks require to know the target Lipschitz constant beforehand. This makes them difficult to be deployed to geometry applications because a good Lipschitz constant is unknown, thus leading to extensive hyperparameter tuning (see Fig. 3).

This inspires some Lipschitz-like regularizations, such as the spectral norm regularization which penalizes the largest eigenvalue of each weight matrix [Yoshida and Miyato 2017]. They also show that adding regularization improves the generalizability and adversarial robustness. But this regularization does not incorporate the fact that the Lipschitz constant grows exponentially with respect to the depth of the network. In practice, it causes difficulties in hyperparameter tuning because changing the number of layers requires to also change the weight on the regularization (see Fig. 4).

3 BACKGROUND IN LIPSCHITZ NETWORKS

A neural network f_θ with parameter θ is called *Lipschitz continuous* if there exist a constant $c \geq 0$ such that

$$\underbrace{\|f_\theta(\mathrm{t}_0) - f_\theta(\mathrm{t}_1)\|_p}_{\text{change in the output}} \leq c \underbrace{\|\mathrm{t}_0 - \mathrm{t}_1\|_p}_{\text{change in the input}} \tag{1}$$

for all possible inputs $\mathrm{t}_0, \mathrm{t}_1$ under a p-norm of choice. The parameter c is called the *Lipschitz constant*. Intuitively, this constant c bounds how fast this function f_θ can change.

As pointed out by several previous papers mentioned in Sec. 2, the Lipschitz bound c of an fully-connected network with 1-Lipschitz activation functions (e.g., ReLU) can be estimated via

$$c = \prod_{i=1}^{L} \|\mathrm{W}_i\|_p, \tag{2}$$

where W_i is the weight matrix at layer i and L denotes the number of layers. This estimate is a loose upper bound due to the ignorance of activation functions, but in practice, optimizing this upper bound is still effective (i.e. [Miyato et al. 2018]).

In the past years, different ways of controlling the Lipschitz bound of a network have been studied. A dominant strategy is to perform weight normalization. For instance, if one wants to enforce the network to be 1-Lipschitz $c = 1$, then one can achieve this by normalizing the weight such that $\|\mathrm{W}_i\|_p = 1$ after each gradient step during training. The normalization scheme depends on the choice of different matrix p-norms:

$$\|\mathrm{M}\|_2 = \sigma_{\max}(\mathrm{M}), \tag{3}$$

$$\|\mathrm{M}\|_1 = \max_j \sum_i |m_{ij}|, \quad \|\mathrm{M}\|_\infty = \max_i \sum_j |m_{ij}|, \tag{4}$$

where $\sigma_{\max}(\mathrm{M})$ denotes the maximum eigenvalue of M. Thus, when $p = 2$, weight normalization consists of rescaling the weight matrix based on its maximum eigenvalue. Popular techniques include spectral normalization based on the power iteration [Miyato et al. 2018] and the Björck Orthonormalization [Anil et al. 2019; Björck and Bowie 1971]. When $p = \infty$ ($p = 1$), weights normalization is simply scaling individual rows (columns) to have a maximum absolute row (column) sum smaller than a prescribed bound.

These matrix norms are also related to each other. Let M be a matrix with size m-by-n, its 2-norm is bounded by its 1-norm and ∞-norm in the following relationships

$$\frac{1}{\sqrt{n}}\|\mathrm{M}\|_\infty \leq \|\mathrm{M}\|_2 \leq \sqrt{m}\|\mathrm{M}\|_\infty \tag{5}$$

$$\frac{1}{\sqrt{m}}\|\mathrm{M}\|_1 \leq \|\mathrm{M}\|_2 \leq \sqrt{n}\|\mathrm{M}\|_1. \tag{6}$$

This implies that optimizing the Lipschitz bound under a particular choice of norms will effectively optimize the bound measured by the other norms. One could also consider the entry-wise matrix norm $\|\mathrm{M}\|_{p,q}$ (see [Horn and Johnson 2012]). But we leave the exploration of the most effective strategy as future work.

4 METHOD

Throughout we use $f_\theta(\mathrm{x}, \mathrm{t})$ to denote the forward model of an implicit shape parameterized by a neural network, namely a mapping from a tuple of a location $\mathrm{x} \in \mathbb{R}^d$ in d-dimensional space and a latent code $\mathrm{t} \in \mathbb{R}^{|\mathrm{t}|}$ to $\mathbb{R}$. f_θ has parameters $\theta = \{\mathrm{W}_i, \mathrm{b}_i\}$ containing weights W_i and biases b_i of each layer i.

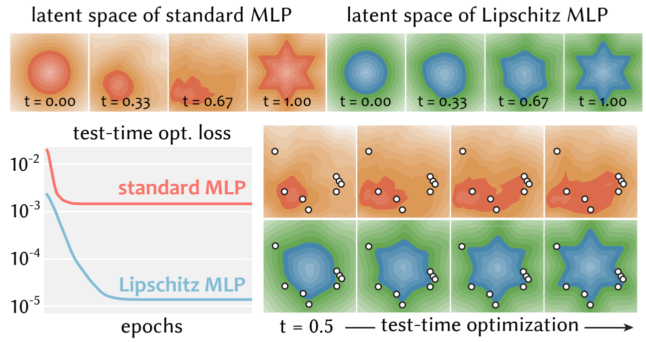


Fig. 5. We demonstrate the importance of smooth latent space on a toy test-time reconstruction of a 2D point cloud. We train a standard MLP and our Lipschitz MLP to interpolate a circle ($t = 0$) to a star ($t = 1$). Then we sample points on the zero-isoline of the star (white points) and optimize the latent code to fit those points (with initial t = 0.5). Our smooth latent space enables latent optimization to find the correct solution (bottom blue) while the non-smooth one suffers from poor local minimum (bottom red).

Our goal is to train a neural network that is smooth with respect to its latent code t. This property is important for shape editing and in applications requiring a well-structured latent space in which a small change in t results in a small change to the output (see Fig. 5).

A straightforward idea is to augment the loss function with some smoothness regularizations, such as the Dirichlet energy. Specifically, one would draw a bunch of samples points t_j in the latent space and turn the original loss function $\mathcal{L}$ into

$$\mathcal{J}(\theta) = \mathcal{L}(\theta) + \alpha \sum_j \| \frac{\partial f_\theta}{\partial t}(x, t_j) \|^2. \tag{7}$$

Although being effective in encouraging a smooth neural field f_θ with respect to the change in latent code at the sampled locations t_j, it often results in non-smooth behavior elsewhere. For instance, in Fig. 2, we apply the Dirichlet regularization to a toy task: interpolating between two neural SDFs conditioned on two latent codes, $t = 0$ and $t = 1$, with 1D latent dimension. In this example, we minimize the Dirichlet energy at $t = 1/3, 2/3$. The network is able to find a perfectly smooth (constant) solution at the sampled ts, but it squeezes all the changes at the very beginning and results in non-smooth behavior $0 < t < 1/3$. This issue is even more troublesome when the latent dimension is large and sampling densely is intractable. A more desirable approach is to guarantee smoothness for *all* possible latent inputs without the need to densely sample the latent space.

Our main idea is to define the smoothness energy solely based on network parameters (i.e. weights of a neural network) regardless of the inputs. One promising solution is to encourage *Lipschitz continuity* with respect to the inputs, in our case the latent code t, and use its Lipschitz constant c as a proxy for smoothness. Specifically, we want the network to satisfy

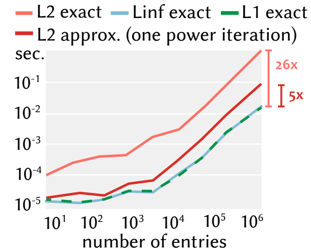
$$\|f_\theta(x, t_0) - f_\theta(x, t_1)\|_p \leq c \; \|t_0 - t_1\|_p \tag{8}$$

for all possible combinations of x, t_0, t_1. As the upper bound of the Lipschitz constant $c = \prod_i \|W_i\|_p$ only depends on the weight matrices W_i, c is independent to the choice of inputs. Therefore, by decreasing c, one guarantees smoothness everywhere even beyond the training set (see Fig. 1).

To decrease the Lipschitz constant and encourage smoothness, we present a new regularization. The key idea is to treat the Lipschitz constant of a network as a learnable parameter and minimize it, instead of a pre-determined value (e.g., [Miyato et al. 2018]). There are many possible ways one can formulate such a regularization and there is not a single formulation that is uniformly the best. We first present our recommended solution and defer the comparison with alternative formulations in Sec. 4.2.

4.1 Lipschitz Multilayer Perceptron

The first question is to choose a p-norm to measure the Lipschitz constant Eq. (8). In our case, we have no restriction on the choice of p-norm. We solely want to have a small Lipschitz constant to encourage smooth behavior with respect to the change of the latent code t. Thus, we simply choose the matrix ∞-norm due to its efficiency (see the inset). But if applications require other choices, our approach is also applicable.



After determining the matrix norm, our method only requires two simple modifications to a standard fully-connected network: adding a weight normalization to each fully connected layer, parameterized by a learnable Lipschitz variable, and a regularization term to encourage an overall small Lipschitz constant that is minimized together with the task loss function.

4.1.1 Weight Normalization Layer. Our weight normalization shares the same spirit as the other weight normalization methods for accelerating training [Salimans and Kingma 2016] and generalization ability [Huang et al. 2018]. But the key difference is that our normalization is based on the Lipschitz constant of a layer, which is more suitable for obtaining a smooth network.

We augment each layer of an MLP $y = \sigma(W_i x + b_i)$ with a Lipschitz weight normalization layer given a trainable Lipschitz bound c_i for layer i

$$y = \sigma(\widehat{W}_i x + b_i), \quad \widehat{W}_i = \textit{normalization}(W_i, \textit{softplus}(c_i)), \tag{9}$$

where the $\textit{softplus}(c_i) = \ln(1 + e^{c_i})$ is a reparameterization designed to avoid infeasible negative Lipschitz bounds. In most of our cases $c_i \approx \textit{softplus}(c_i)$ because c_i is often a large positive number. Due to our choice of using ∞-norm, this normalization is efficient and simple: we scale each row of W_i to have the absolute value row-sum less than or equal to $\textit{softplus}(c_i)$. If one of the rows already has the absolute value row-sum smaller than $\textit{softplus}(c_i)$, then no scaling is performed. With this normalization layer, even if the raw weight matrix W_i has a Lipschitz constant greater than $\textit{softplus}(c_i)$, this normalization can still guarantee the Lipschitz constant is bounded by $\textit{softplus}(c_i)$. Therefore, we never clip the weights during training.

Implementation. Our method can be implemented in a few lines of code. Given the weight matrix `Wi` and the per-layer Lipschitz upper bound `ci`, the normalization layer can be implemented in JAX [Bradbury et al. 2018] as

```
import jax.numpy as jnp
def normalization(Wi, softplus_ci): # L-inf norm
  absrowsum = jnp.sum(jnp.abs(Wi), axis=1)
  scale = jnp.minimum(1.0, softplus_ci/absrowsum)
  return Wi * scale[:,None]
```

and each layer of the Lipschitz MLP is simply

```
y = sigma(normalization(Wi, softplus(ci))*x + bi)
```

where `sigma` denotes the activation function and `softplus` is the built-in *softplus* function in JAX.

Although being efficient, using our Lipschitz weight normalization will still increase the training time. For example, in the 2D interpolation task (such as Fig. 3), adding our normalization slows down the training from 265.83 epochs per second down to 229.95 epochs per second.

However, incorporating our regularization will not influence the performance during test time because one can explicitly construct the normalized weight matrix $\widehat{W}_i$ by clipping the weight matrix W_i

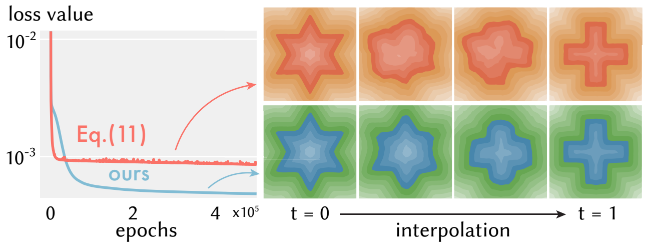


Fig. 6. Our method converges to a smoother result compared to the k-Lipschitz architecture described in [Anil et al. 2019] (see Eq. (11)). We use the same α for both networks because we both define the regularization as the raw Lipschitz constant of the network.

with the learned constant c_i. Then, one can use the vanilla MLP with the normalized weights $\widehat{W}_i$ as their final model.

4.1.2 Our Lipschitz Regularization. The second ingredient is to augment the original loss function $\mathcal{L}$ with a *Lipschitz regularization.* Our Lipschitz regularization is defined simply as the Lipschitz bound of the network. But instead of directly defining on the weight matrices, we define it on the parameterized per-layer Lipschitz bounds $\mathit{softplus}(c_i)$ in the normalization layer Eq. (9). Specifically, we augment the original loss function $\mathcal{L}$ with a Lipschitz term as

$$\boxed{\mathcal{J}(\theta, C) = \mathcal{L}(\theta) + \alpha \prod_{i=1}^{l} \mathit{softplus}(c_i)} \tag{10}$$

where we use $C = \{c_i\}$ to denote the collection of per-layer Lipschitz constants c_i used in the weight normalization. As mentioned in Eq. (2), the product of per-layer Lipschitz constants is the Lipschitz bound of the network.

4.2 Comparison with Alternatives

There are many ways one can implement and formulate a Lipschitz regularization. In this section, we compare our formulation with alternative formulations. As the amount of regularization α will influence the analysis, we perform parameter sweeping for each formulation independently and compare their best set-ups.

One solution is to design a regularization based on the architecture of the k-Lipschitz networks, such as the one suggested by Anil et al. [2019]. Specifically, Anil et al. [2019] constrain all the layers to be 1-Lipschitz and multiply the final layer with a constant k to make it k-Lipschitz. A possible formulation to make it learnable is to simply treat the k as the Lipschitz regularization term

$$\mathcal{J}(\theta, k) = \mathcal{L}(\theta) + \alpha k \tag{11}$$

However, we struggle to use this formulation in Eq. (11) to find a good local minimum even for the simple 2D interpolation task (see Fig. 6). Moreover, when one switches to other types of activations, the result is even worse because the distributive property of per-layer scaling no longer holds.

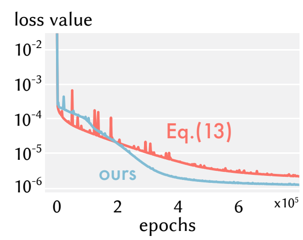
Another alternative is the formulation by Yoshida and Miyato [2017] which defines a Lipschitz-like regularization as the summation of squared Lipschitz bounds of each layer. Generalizing the definition in [Yoshida and Miyato 2017] to p-norms gives us

$$\mathcal{J}(\theta) = \mathcal{L}(\theta) + \alpha \sum_{i=1}^{l} \|W_i\|_p^2 \tag{12}$$

Although this formulation can effectively find a smooth solution given a good α, finding a good α is not easy with this formulation. This formulation fails to capture the exponential growth of the Lipschitz constant with respect to the network depth (see Eq. (2)). In practice, it implies that the formulation Eq. (12) proposed by Yoshida and Miyato [2017] requires a different α when we change the depth of the network. In Fig. 4, we use the spectral norm for the method by Yoshida and Miyato [2017] and show that the same α results in different behavior for networks with different capacities. In contrast, our method leads to a more consistent behavior with the same α.

Another alternative is to define the Lipschitz regularization directly on the weight matrices, without the normalization layer.

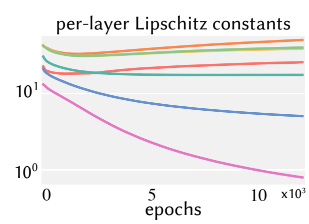
$$\mathcal{J}(\theta) = \mathcal{L}(\theta) + \alpha \prod_{i=1}^{l} \|W_i\|_\infty \tag{13}$$



This approach works equally well as our method on narrower networks, but it converges slower on wider ones. We suspect that this is because the ∞-norm only depends on a single row of the weight matrix. So on a wider network, this formulation requires more epochs to penalize its parameters. In the inset, we show the convergence of a 2-layer MLP with 1024 neurons each layer on 2D interpolation.

Another tempting solution is to consider the log of the Lipschitz bound to turn the product in Eq. (10) into a summation

$$\mathcal{J}(\theta, C) = \mathcal{L}(\theta) + \alpha \sum_{i=1}^{l} \log(\mathit{softplus}(c_i)) \tag{14}$$



However, this makes the regularization unbounded because log goes to negative infinity when one of the Lipschitz constants approaches zero. In practice, this implies the tendency to continue penalizing the layer with a smaller Lipschitz constant. In a few cases, we did not observe the Lipschitz constants to converge. In the inset, we visualize the per-layer Lipschitz constants of a network trained on the ShapeNet [Chang et al. 2015], where the layers are ordered in rainbow colors. We can observe that one of the Lipschitz constants continues to decrease (purple) even after a week of training.

4.3 Comparison with Weight Decay

Our Lipschitz regularization can be perceived as a variant of the weight decay regularization, such as the Tikhonov (L2) regularization [Tihonov 1963] and Lasso (L1) [Tibshirani 1996]. These weight decay methods are often used to avoid overfitting and improve generalization ability. However, it is unclear what their relationships are with respect to the smoothness of the network. As a result,

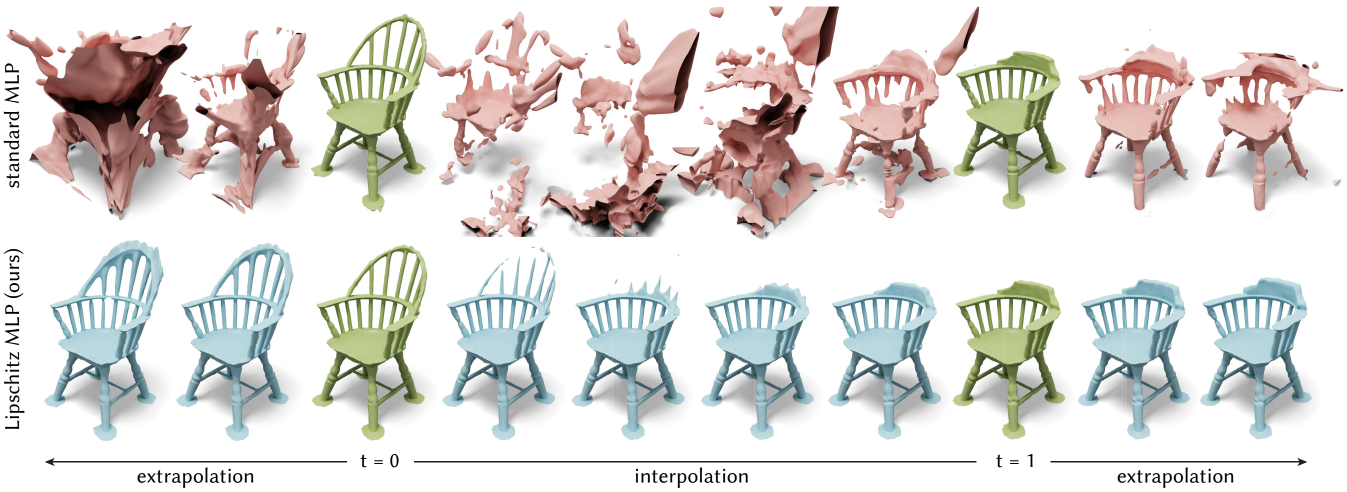


Fig. 7. Our Lipschitz regularization is applicable to different implicit representations. We fit an occupancy network [Mescheder et al. 2019a] to two shapes (green) at $t = 0$ and $t = 1$ with (blue) and without (red) our Lipschitz regularization. Our method generates smoother interpolation/extrapolation results.

Table 1. We compute the squared norm of the Jacobian matrix J via back-propagation. We report the average and the maximum value of the $\|J\|^2$ for all training data and show that our Lipschitz regularization achieves a smoother solution compared to other weight decay methods.

Metrics	Ours	L2	L1	Vanilla
mean $\|J\|^2$	**1.009**	1.020	1.016	1.021
max $\|J\|^2$	**9.419**	21.181	17.361	23.658

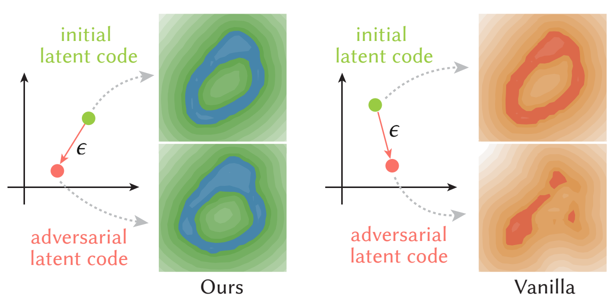
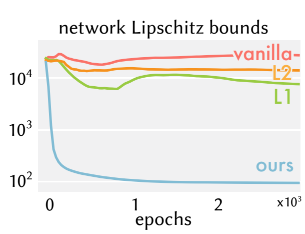


Fig. 8. We perform a (virtual) adversarial attack which perturbs the latent code with the fast gradient sign method [Goodfellow et al. 2015]. Vanilla AE is vulnerable to the attack so the SDF of "0" is completely destroyed. In contrast, our Lipschitz regularized network is more robust to the attack.

networks trained using weight decay are less smooth (measured by Lipschitz constant) compared to the network trained with our Lipschitz regularization (see the inset).



Our Lipschitz regularization also leads to a smoother network compared to other weight decay measured by a popular metric, square Jacobian norm. To verify this, we train autoencoders (AE) to reconstruct the MNIST digits represented as signed distance functions. In Table 1, our Lipschitz AE leads to smaller Jacobian norms compared to the vanilla AE, the L1 regularized AE, and the L2 regularized AE. We provide experimental details in App. A.6.

Besides weight decay, there are other types of regularization that are not defined on network weights, such as adding noise [Poole et al. 2014] and Dropout [Srivastava et al. 2014]. These methods can complement our approach, such as [Gouk et al. 2021]. We leave the study on mixing and matching these regularizations as future work. For a more comprehensive discussion, please refer to a survey on regularization [Moradi et al. 2020].

5 EXPERIMENTS

Our regularization encourages a fully connected network to output Lipschitz continuous functions and is therefore applicable to different tasks that favor smooth solutions. In this section, we examine at the effectiveness of our approach in improving the robustness of a network, shape interpolation, and test-time optimization.

5.1 Adversarial Robustness

Adversarial attacks are small, structured changes made to a network's input signal that cause a significant change in output [Szegedy et al. 2014]. As been previously shown, Lipschitz continuous networks can improve robustness against adversarial attacks [Li et al. 2019a]. Here we demonstrate that our proposed regularization can serve that purpose. To that end we train an AE to reconstruct the signed distance functions of MNIST digits from their input image. We then adversarially perturb the latent code as described in Fig. 8, and show that Lipschitz MLP is more robust to adversarial perturbations than a standard one. We quantitatively evaluate the robustness against this type of latent adversarial attack on all the MNIST digits. A standard AE results in an average 0.06 and maximum 0.34 difference in the signed distance value. In contrast, our Lipschitz AE is

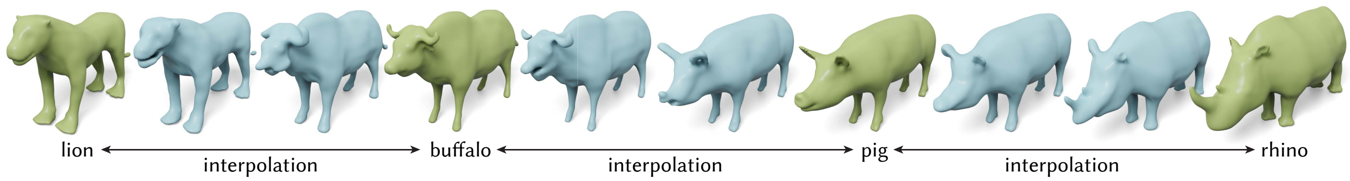


Fig. 9. Our method only encourages smooth interpolation. Thus our method cannot extract high-level information, such as semantics, from only a handful of shapes. Therefore, when we interpolate between animal shapes (green), the interpolated results (blue) may not be realistic animals.

Fig. 10. Our method enables smooth interpolation between few training shapes. By training our model on three examples (green), we can generate high-quality novel shapes by interpolating latent codes of training shapes.

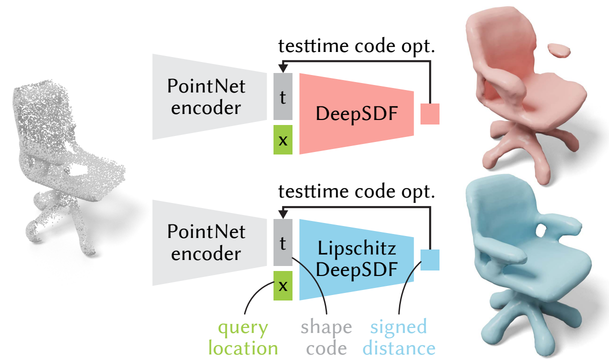


Fig. 11. Encouraging smoothness with respect to the change in latent code encourages the latent space to be more compact. In the application of test time optimization, our method yields a better reconstruction (blue) given partial observations (left).

more robust with only an average 0.03 and maximum 0.16 difference. We refer readers to App. A.6 for details about this experiment.

5.2 Few-Shot Shape Interpolation & Extrapolation

Shape interpolation is a fundamental task and several classic methods exist, such as [Solomon et al. 2015] and [Kilian et al. 2007]. When shapes are mapped from a latent descriptor to 3D via a decoder, interpolation through latent space traversal often requires training on abundant data so that the latent space is well structured. Our regularization can aid shape interpolation and extrapolation in given only sparse training shapes. In Fig. 13, we provide several 3D SDF interpolation examples trained on only two shapes. Our method is also applicable to other implicit representations. In Fig. 7, we evaluate our method to interpolate the *occupancy* [Mescheder et al. 2019a] which assigns each point in $\mathbb{R}^3$ a binary value $[0, 1]$ representing whether it is inside or outside.

5.3 Reconstruction with Test Time Optimization

Autoencoders are popular in reconstructing the full shape from a partial point cloud. However, simply forward passing the partial points through the AE often outputs unsatisfying results. A common way to resolve this issue is to further optimize the latent code of the partial point cloud during test time [Gurumurthy and Agrawal 2019]. Despite being effective, test time optimization is very sensitive to parameters in the optimization (e.g., initialization) and suffers from bad local minima. Duggal et al. [2022] even propose a dedicated method aiming for resolving this issue of test-time optimization.

Table 2. We quantitatively evaluate the test time optimization. Given a ground truth point cloud from the test set, we delete the right-half of the point to obtain a partial point cloud and then we perform test-time optimization to reconstruct the full shape back. We report the Chamfer distance and Hausdorff distance between the ground truth point cloud and our reconstructed full shape averaged across the test set.

Metrics	Lipschitz DeepSDF	DeepSDF
average Chamfer distance	**0.0013**	0.0343
average Hausdorff distance	**0.1270**	0.3441

We discover that our Lipschitz regularization can complement the research in stabilizing test time optimization. Simply by adding our Lipschitz regularization to the vanilla autoencoder set-up, we can encourage a smoother latent manifold and stabilize the test time optimization. In Fig. 11 and Table 2, we show that we achieve a better reconstruction result both qualitatively and quantitatively. Our method can complement the method based on training additional networks, such as adding a discriminator [Duggal et al. 2022] or a generative adversarial network [Gurumurthy and Agrawal 2019]. But we leave them as future work.

6 CONCLUSION & FUTURE WORK

Our regularization encourages fully connected networks to have a small Lipschitz constant. Our regularization is defined on a loose upper bound of the true Lipschitz constant. Using a tighter estimate would benefit applications that require more precise control. Our shape interpolation behaves similarly to linear interpolation. Incorporating the Wasserstein metric into our smoothness measure could encourage shape interpolation to behave more like *optimal transport*. Furthermore, encouraging learning high-level structural information from few examples would aid to the experiment on few shot shape interpolation (see Fig. 9). Our Lipschitz regularization is a generic technique to encourage smooth neural network solutions. However, our experiments are mainly conducted on neural implicit geometry tasks. We would be interested in applying our regularization to other tasks beyond geometry processing.

ACKNOWLEDGMENTS

Our research is funded in part by NSERC Discovery (RGPIN2017−05235, RGPAS−2017−507938), New Frontiers of Research Fund (NFRFE−201), the Ontario Early Research Award program, the Canada Research Chairs Program, a Sloan Research Fellowship, the DSI Catalyst Grant program and gifts by Adobe Systems. Special thanks to Otman Benchekroun, Alex Evans, James Lucas, Oded Stein, Towaki Takikawa, and Jackson Wang for inspiring early discussions. We thank Silvia Sellán and Frank Shen for assisting code release and experiments. We also want to thank all the artists for sharing a rich variety of 3D models to facilitate our research.

REFERENCES

Cem Anil, James Lucas, and Roger B. Grosse. 2019. Sorting Out Lipschitz Function Approximation. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 291–301.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein generative adversarial networks. In *International conference on machine learning*. PMLR, 214–223.

Åke Björck and Clazett Bowie. 1971. An iterative algorithm for computing the best estimate of an orthogonal matrix. *SIAM J. Numer. Anal.* 8, 2 (1971), 358–364.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018. *JAX: composable transformations of Python+NumPy programs.* http://github.com/google/jax

Angel X. Chang, Thomas A. Funkhouser, Leonidas J. Guibas, Pat Hanrahan, Qi-Xing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. 2015. ShapeNet: An Information-Rich 3D Model Repository. *CoRR* abs/1512.03012 (2015).

Wenzheng Chen, Huan Ling, Jun Gao, Edward J. Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. 2019. Learning to Predict 3D Objects with an Interpolation-based Differentiable Renderer. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 9605–9616.

Moustapha Cissé, Piotr Bojanowski, Edouard Grave, Yann N. Dauphin, and Nicolas Usunier. 2017. Parseval Networks: Improving Robustness to Adversarial Examples. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017 (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, 854–863.

Harris Drucker and Yann Le Cun. 1991. Double backpropagation increasing generalization performance. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, Vol. 2. IEEE, 145–150.

Shivam Duggal, Zihao Wang, Wei-Chiu Ma, Sivabalan Manivasagam, Justin Liang, Shenlong Wang, and Raquel Urtasun. 2022. Mending Neural Implicit Modeling for 3D Vehicle Reconstruction in the Wild. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 1900–1909.

Tim Elsner, Moritz Ibing, Victor Czech, Julius Nehring-Wirxel, and Leif Kobbelt. 2021. Intuitive Shape Editing in Latent Space. *CoRR* abs/2111.12488 (2021). arXiv:2111.12488

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010 (JMLR Proceedings, Vol. 9)*, Yee Whye Teh and D. Mike Titterington (Eds.). JMLR.org, 249–256.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.).

Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael J. Cree. 2021. Regularisation of neural networks by enforcing Lipschitz continuity. *Mach. Learn.* 110, 2 (2021), 393–416.

Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. 2020. Implicit Geometric Regularization for Learning Shapes. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 3789–3799.

Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. 2017. Improved Training of Wasserstein GANs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 5767–5777.

Swaminathan Gurumurthy and Shubham Agrawal. 2019. High Fidelity Semantic Shape Completion for Point Clouds Using Latent Optimization. In *IEEE Winter Conference on Applications of Computer Vision, WACV 2019, Waikoloa Village, HI, USA, January 7-11, 2019*. IEEE, 1099–1108.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, 1026–1034.

Amir Hertz, Rana Hanocka, Raja Giryes, and Daniel Cohen-Or. 2020. Deep geometric texture synthesis. *ACM Trans. Graph.* 39, 4 (2020), 108.

Judy Hoffman, Daniel A. Roberts, and Sho Yaida. 2019. Robust Learning with Jacobian Regularization. *CoRR* abs/1908.02729 (2019). arXiv:1908.02729

Roger A Horn and Charles R Johnson. 2012. *Matrix analysis*. Cambridge university press.

Lei Huang, Xianglong Liu, Bo Lang, Adams Wei Yu, Yongliang Wang, and Bo Li. 2018. Orthogonal Weight Normalization: Solution to Optimization Over Multiple Dependent Stiefel Manifolds in Deep Neural Networks. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, Sheila A. McIlraith and Kilian Q. Weinberger (Eds.). AAAI Press, 3271–3278.

Daniel Jakubovitz and Raja Giryes. 2018. Improving DNN Robustness to Adversarial Attacks Using Jacobian Regularization. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XII (Lecture Notes in Computer Science, Vol. 11216)*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.). Springer, 525–541.

Matt Jordan and Alexandros G. Dimakis. 2020. Exactly Computing the Local Lipschitz Constant of ReLU Networks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.).

Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. 2018. Neural 3D Mesh Renderer. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. Computer Vision Foundation / IEEE Computer Society, 3907–3916.

Martin Kilian, Niloy J. Mitra, and Helmut Pottmann. 2007. Geometric modeling in shape space. *ACM Trans. Graph.* 26, 3 (2007), 64.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.).

Qiyang Li, Saminul Haque, Cem Anil, James Lucas, Roger B. Grosse, and Jörn-Henrik Jacobsen. 2019a. Preventing Gradient Attenuation in Lipschitz Constrained Convolutional Networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 15364–15376.

Yuanzhi Li, Colin Wei, and Tengyu Ma. 2019b. Towards Explaining the Regularization Effect of Initial Large Learning Rate in Training Neural Networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 11669–11680.

Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. 2019. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 7708–7717.

Lars M. Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. 2019a. Occupancy Networks: Learning 3D Reconstruction in Function Space. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 4460–4470.

Lars M. Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. 2019b. Occupancy Networks: Learning 3D Reconstruction in Function Space. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 4460–4470.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. 2018. Spectral Normalization for Generative Adversarial Networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Jonathan Uesato, and Pascal Frossard. 2019. Robustness via Curvature Regularization, and Vice Versa. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 9078–9086.

Reza Moradi, Reza Berangi, and Behrouz Minaei. 2020. A survey of regularization strategies for deep models. *Artificial Intelligence Review* 53, 6 (2020), 3947–3986.

Adam M. Oberman and Jeff Calder. 2018. Lipschitz regularized Deep Neural Networks converge and generalize. *CoRR* abs/1808.09540 (2018). arXiv:1808.09540

Jeong Joon Park, Peter Florence, Julian Straub, Richard A. Newcombe, and Steven Lovegrove. 2019. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 165–174.

Ben Poole, Jascha Sohl-Dickstein, and Surya Ganguli. 2014. Analyzing noise in autoencoders and deep networks. *CoRR* abs/1406.1831 (2014). arXiv:1406.1831

Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. 2017. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 77–85.

Marie-Julie Rakotosaona and Maks Ovsjanikov. 2020. Intrinsic Point Cloud Interpolation via Dual Latent Space Navigation. In *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 12347)*, Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (Eds.). Springer, 655–672.

Mihaela Rosca, Theophane Weber, Arthur Gretton, and Shakir Mohamed. 2020. A case for new neural network smoothness constraints. *CoRR* abs/2012.07969 (2020). arXiv:2012.07969

Tim Salimans and Diederik P. Kingma. 2016. Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett (Eds.). 901.

Justin Solomon, Fernando de Goes, Gabriel Peyré, Marco Cuturi, Adrian Butscher, Andy Nguyen, Tao Du, and Leonidas J. Guibas. 2015. Convolutional wasserstein distances: efficient optimal transportation on geometric domains. *ACM Trans. Graph.* 34, 4 (2015), 66:1–66:11.

Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15, 1 (2014), 1929–1958.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.).

Dávid Terjék. 2020. Adversarial Lipschitz Regularization. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Robert Tibshirani. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)* 58, 1 (1996), 267–288.

Andrei Nikolajevits Tihonov. 1963. Solution of incorrectly formulated problems and the regularization method. *Soviet Math.* 4 (1963), 1035–1038.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. 2018. Deep image prior. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 9446–9454.

Dániel Varga, Adrián Csiszárik, and Zsolt Zombori. 2017. Gradient regularization improves accuracy of discriminative models. *arXiv preprint arXiv:1712.09936* (2017).

Aladin Virmaux and Kevin Scaman. 2018. Lipschitz regularity of deep neural networks: analysis and efficient estimation. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (Eds.). 3839–3848.

Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. 2018. Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XI (Lecture Notes in Computer Science, Vol. 11215)*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.). Springer, 55–71.

Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane S. Boning, and Inderjit S. Dhillon. 2018. Towards Fast Computation of Certified Robustness for ReLU Networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018 (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer G. Dy and Andreas Krause (Eds.). PMLR, 5273–5282.

Francis Williams, Teseo Schneider, Cláudio T. Silva, Denis Zorin, Joan Bruna, and Daniele Panozzo. 2019. Deep Geometric Prior for Surface Reconstruction. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 10130–10139.

Francis Williams, Matthew Trager, Joan Bruna, and Denis Zorin. 2021. Neural Splines: Fitting 3D Surfaces With Infinitely-Wide Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, 9949–9958.

Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. 2021. Neural Fields in Visual Computing and Beyond. *CoRR* abs/2111.11426 (2021).

Yuichi Yoshida and Takeru Miyato. 2017. Spectral Norm Regularization for Improving the Generalizability of Deep Learning. *CoRR* abs/1705.10941 (2017). arXiv:1705.10941

A EXPERIMENTAL & IMPLEMENTATION DETAILS

For all our experiments, we initialize the per-layer Lipschitz constant $c_i = \|W_i\|_\infty$ in Eq. (10) as the Lipschitz constant of the initial weight matrix. The weight matrices are initialized with the method by [Glorot and Bengio 2010] if the activation is tanh and with the method by He et al. [2015] if the activation is ReLU or its variants. In the following subsections, we present details of individual experiments in the main text.

A.1 3D Neural Implicit Interpolation

We use the DeepSDF architecture [Park et al. 2019] to design the interpolation experiment of 3D neural SDFs (Fig. 1, 10, 9). The inputs to our network are the query location in $\mathbb{R}^3$ and the latent code t. Our network consists of 5 hidden layers of 256 neurons with the tanh activation. We multiply the input point x with one hundred $100x$ to avoid the possibility that the network smoothness in the latent code is bounded by spatial smoothness. The last layer is a linear layer outputting the signed distance value at the input query location. The training shapes are normalized to the bounding box between 0 and 1. We use the mean square error (MSE) as our loss function $\mathcal{L}(\theta)$ for our baseline model. We augment the MSE loss with our Lipschitz regularization Eq. (10) (with $\alpha = 10^{-6}$) to evaluate the influence of our method. We compute the MSE loss by sampling 10^5 points where 40% of them are on the surface, 40% are near the surface, and 20% are drawn from uniformly sampling the bounding box. We use the Adam optimizer [Kingma and Ba 2015] with its default parameters presented and learning rate 10^{-4}.

For our experiment on occupancy interpolation [Mescheder et al. 2019b] (Fig. 7), we use the same architecture as our SDF experiments and append a *sigmoid* function right before the output. Our loss function is the cross-entropy loss with and without our Lipschitz regularization (with $\alpha = 10^{-6}$).

In these interpolation experiments, we pre-determine the latent codes for each shape we want to interpolate. For example Fig. 1, we minimize the loss with respect to the SDF of a torus when $t = 0$ and with the double torus when $t = 1$. Similarly, we also use $t = 0, 1$ in our occupancy interpolation Fig. 7. In Fig. 10, the three sets of code for the green shapes are [0,0], [1,0], [0.5, 0.866]. In Fig. 9, the latent code for the four shapes are set to be one-hot vectors, such as [1,0,0,0].

A.2 2D Neural Implicit Interpolation

The experiments on interpolating 2D neural implicits are similar to the above-mentioned 3D interpolation ones App. A.1. We minimize the MSE loss with Adam and use the DeepSDF architecture. The only difference is that the network is smaller and our training data is sampled uniformly on the 2D space. In Fig. 2, 3, we use a MLP with 5 hidden layers of 64 neurons with ReLU activation. Specifically, we set the Dirichlet regularization Eq. (7) to be 10^{-4} in Fig. 2. We manually set the Lipschitz constant per layer to be 1.4 in Fig. 3. For our Lipschitz regularization, we set $\alpha = 3 \times 10^{-6}$.

A.3 Toy 2D test time optimization

In Fig. 5, we present a toy test time optimization to demonstrate the importance of having a smooth latent space. We use the same training set-up as App. A.2 to train our interpolation networks. During test-time, we initialize the latent code to be $t = 0.5$ and we randomly sample 8 points on the iso-line of the star shape and minimize the square distance at these sample points. Intuitively, if the loss is minimized, these points will lie on the zero iso-line of the optimized SDF. In Fig. 5, we show the optimization using Adam. We also tried to optimize the code using SGD, but we only notice a small difference in this toy set-up.

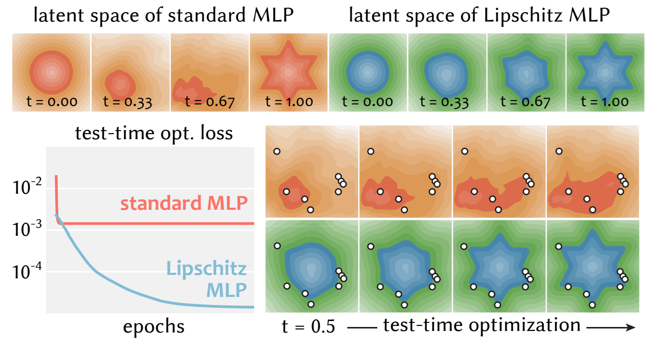


Fig. 12. In this simple toy example, test-time optimization using SGD gives us similar result compared to the one optimized with Adam (see Fig. 5).

A.4 Test time optimization

We evaluate test time optimization on the chair category of the ShapeNet dataset [Chang et al. 2015], which contains 4746 shapes. Our network architecture follows the baseline autoencoder model in [Duggal et al. 2022]. The encoder is a PointNet [Qi et al. 2017]. Specifically, the input to our PointNet is a point location p_i in $\mathbb{R}^3$. We pass this point p_i to a fully-connected network of size [3, 256, 512] with tanh activation to transform this point to a feature vector z_i of size 512. This fully-connected network is used to independently process each point p_i in the point cloud. Thus, after this step, we obtain a n-by-512 feature matrix where n is the number of points. We perform a max-pooling for this feature matrix to obtain a feature vector z_{global} of size 512, encoding the global information of the point cloud. We then concatenate this global feature vector z_{global} is with the feature vector of individual point feature z_i. After concatenation, we pass this local-global feature $[z_i, z_{\text{global}}]$ to the second fully-connected network of size [1024, 512, 256] with tanh activation. Similar to the previous step, we use this shared network to process each point independently. Thus, after processing all the points, we will obtain a n-by-256 feature matrix. We then perform another max-pooling to turn this n-by-256 feature matrix into a 256 global feature vector. To prevent the latent codes from diverging to an arbitrarily vector with large magnitude, apply a sigmoid function to the global feature vector ensure each latent code lies between 0 and 1. We then treat this as the final feature representation of the point cloud after the encoding process.

After the encoding process, our decoder is a DeepSDF [Park et al. 2019] which takes the query location in $\mathbb{R}^3$ and the 256 latent vector from the encoder as its input, and then outputs the signed distance value. The decoder has size [259, 1024, 1024, 1024, 512, 256, 128, 1] with the leaky ReLU activation. Similar to App. A.1, we minimize the

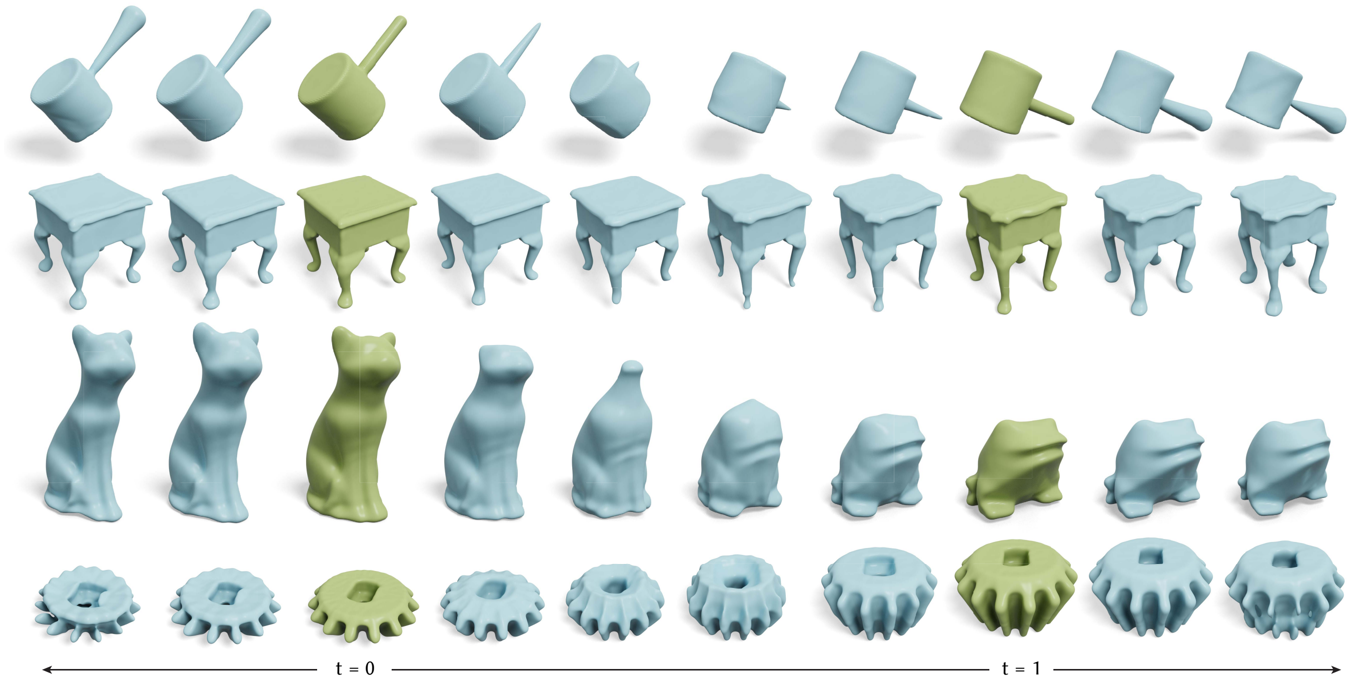


Fig. 13. We fit neural networks to the signed distance field of a shape when the latent code $t = 0$ and another shape when $t = 1$ (green). Our Lipschitz regularization encourages smooth interpolation and extrapolation (blue) even when trained on only a pair of shapes.

MSE loss with Adam and set the learning rate to be 10^{-4}. To evaluate our method, we add a Lipschitz regularization with $\alpha = 10^{-11}$ to the decoder during training (see Fig. 11).

For the test time optimization, given a partial point cloud, we initialize the latent code by passing through our PointNet encoder. With this code, we pass each point in the point cloud to the decoder and minimize the square SDF value. Intuitively, we want the point on the partial point cloud to lie on the zero iso-surface. In addition, we also augment an Eikonal term [Gropp et al. 2020] weighted by $1e-2$ to encourage the output to be an SDF-like function. We minimize this loss (square SDF and an Eikonal loss) by changing the latent code parameter (the parameter before applying the sigmoid) during test time with Adam with a learning rate 10^{-4} until converged.

A.5 Alternative Lipschitz Regularizations

We evaluate our method against the method proposed in [Anil et al. 2019] (Eq. (11)) and another alternative mentioned in Eq. (13) on 2D interpolation tasks App. A.2. We use a 5 layer ReLU MLP with 64 neurons on each hidden layer. Because these approaches are all defined on the Lipschitz bound of the network, we use the same $\alpha = 10^{-6}$ for a fair comparison.

We also compare against the method by Yoshida and Miyato [2017] on 2D interpolation. We use 5 and 10 layers ReLU MLP with 64 neurons on each hidden layer respectively. We use $\alpha = 10^{-5}$ for the method by Yoshida and Miyato [2017] and $\alpha = 10^{-6}$ for our regularization.

In Eq. (14), we evaluate it on a large network trained on the ShapeNet [Chang et al. 2015]. We notice that if the task is simple, such as 2D interpolation, whether to take a log result in similar performance when we have a good α. But when evaluating on large experiments with large networks, minimizing the log of the Lipschitz bound Eq. (14) may start to have issues on convergence. In this experiment specifically, we use the same training set-up as App. A.4 and we use $\alpha = 10^{-6}$.

A.6 MNIST Implicit Autoencoder

The experiments presented in Sec. 4.3 and Sec. 5.1 are evaluated on the MNIST dataset (60000 hand-written digits) represented in 28-by-28 SDF images. Our autoencoder uses two MLPs as our encoder and decoder. Our encoder has size [784, 256, 128, 64, 32] with leaky ReLU activation. It takes the image of an MNIST digit (in SDF form) as the input and outputs a latent code with dimension 32. Similar to App. A.4, we then apply a sigmoid function on the output to ensure the actual latent code lies between 0 and 1. The inputs to our decoder are the latent code and the position in the image space. It outputs the SDF value at the location. Our decoder has dimension [35, 128, 128, 128, 1] with the sorting activation [Anil et al. 2019]. We multiply the input position by 100 to avoid the possibility that the network is constrained by spatial smoothness We use Adam with a learning rate 10^{-4} to minimize the MSE loss evaluated on the 28-by-28 regular 2D grid. For each regularization (L1, L2, and ours), we perform parameter sweeping on log scale and report the best one in terms of test accuracy. Specifically, we use 10^{-7} for both the L1 and L2 regularization, and 10^{-6} for our Lipschitz regularization.

To construct an adversarial perturbation in the latent space, we fist obtain the initial latent code t_i of a valid MNIST digit by passing

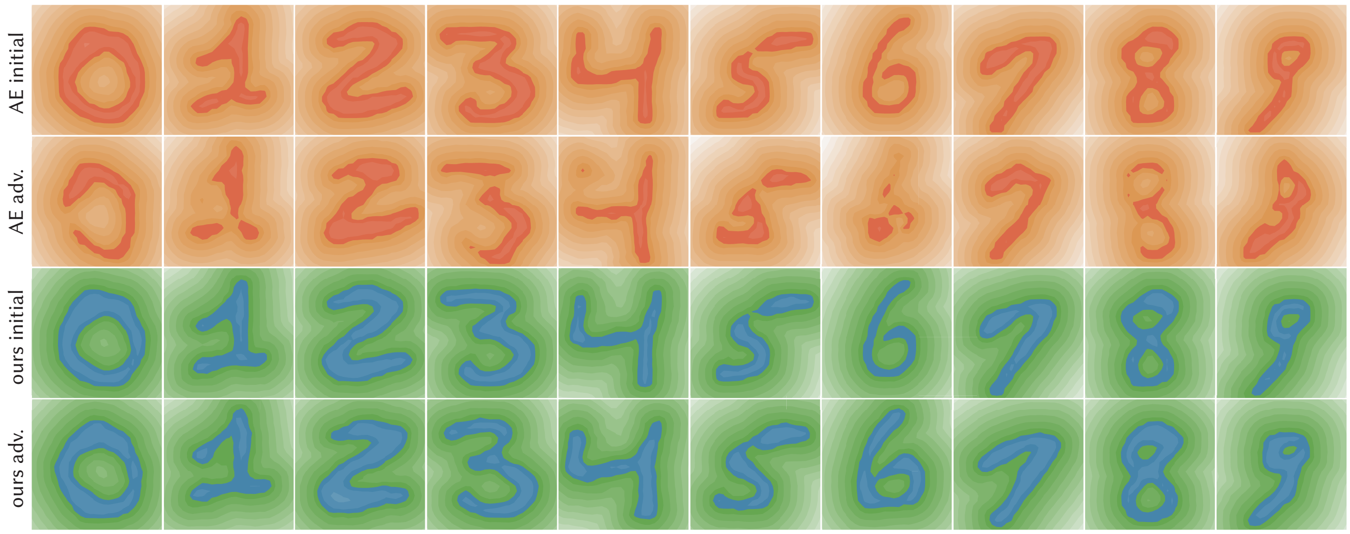


Fig. 14. We perform adversarial attacks in the latent space, the same setup as Fig. 8. We can observe that Vanilla AE is vulnerable to the attack so the initial SDFs (first row) are completely destroyed after adversarial perturbations (second row). In contrast, our Lipschitz regularized network is more robust to the attack (third and the fourth rows).

an image I_i from the training/testing set to our encoder, then we follow the *fast gradient signed method* proposed in [Goodfellow et al. 2015] to compute the adversarial perturbation. Specifically, we set the loss function $\mathcal{J}$ to be squared L2 pixel difference between the input MNIST digit I_i and the decoded image output by the network $f_\theta(\mathsf{t})$ using another code t. Then the adversarial latent code is constructed by

$$\mathsf{t}_{\text{adv.}} = \mathsf{t}_i + \epsilon \ \text{sign}\Big(\frac{\partial \mathcal{J}(\mathsf{I}_i, f_\theta(\mathsf{t}))}{\partial \mathsf{t}}\Big) \tag{15}$$

with predetermined small magnitude $\epsilon = 0.05$. Then the adversarial MNIST digits can be obtained by visualizing the output of the network with the adversarial code $f_\theta(\mathsf{t}_{\text{adv.}})$.

B RELATIONSHIP WITH WEIGHT NORMALIZATION

Weight Normalization is a reparameterization technique proposed by Salimans and Kingma [2016] to accelerate the training process. The key idea is to parameterize the weight matrix W with a trainable matrix V and a trainable scaling factor g

$$\mathsf{W} = g \times \frac{\mathsf{V}}{\|\mathsf{V}\|} \tag{16}$$

where the matrix V is normalized to have unit norm and g is the scaling factor that controls the magnitude of W. This reparameterization is similar to our weight normalization layer in Sec. 4.1.1, but with a different norm. However, the key difference is that this reparameterization along is insufficient to guarantee smoothness. In the first row of Fig. 15, we can observe that solely with the method by Salimans and Kingma [2016] still results in non-smooth interpolation. This is because there is no regularization to encourage small g. In the second row of Fig. 15, we demonstrate the flexibility of our method that we can apply our Lipschitz regularization to encourage small g under the reparameterization in [Salimans and Kingma 2016] and successfully lead to smooth interpolation results.

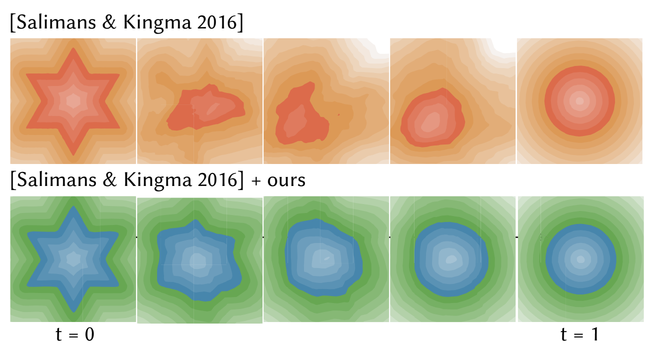


Fig. 15. Our Lipschitz regularization can complement other reparameterization schemes, such as the *weight normalization* by Salimans and Kingma [2016]. We show that solely with weight normalization is insuffcent for obtain smooth results (top row), but our method can be used jointly with [Salimans and Kingma 2016] to obtain smooth interpolation (bottom row).

C COMPARISON WITH SPECTRAL NORMALIZATION

Spectral Normalization proposed by Miyato et al. [2018] is a method to constrain the Lipschitz bound of a network. As discussed in Sec. 2, these Lipschitz constrained networks are sensitive to the choice of the Lipschitz bound. In most geometry applications, a good choice of bound is unknown, thus it requires extensive hyperparameter tuning. In Fig. 16, we show that the results of Lipschitz constrained networks change dramatically when increasing the bounds. Our method, instead, results in smoother change with better results when playing with our regularization parameter α.

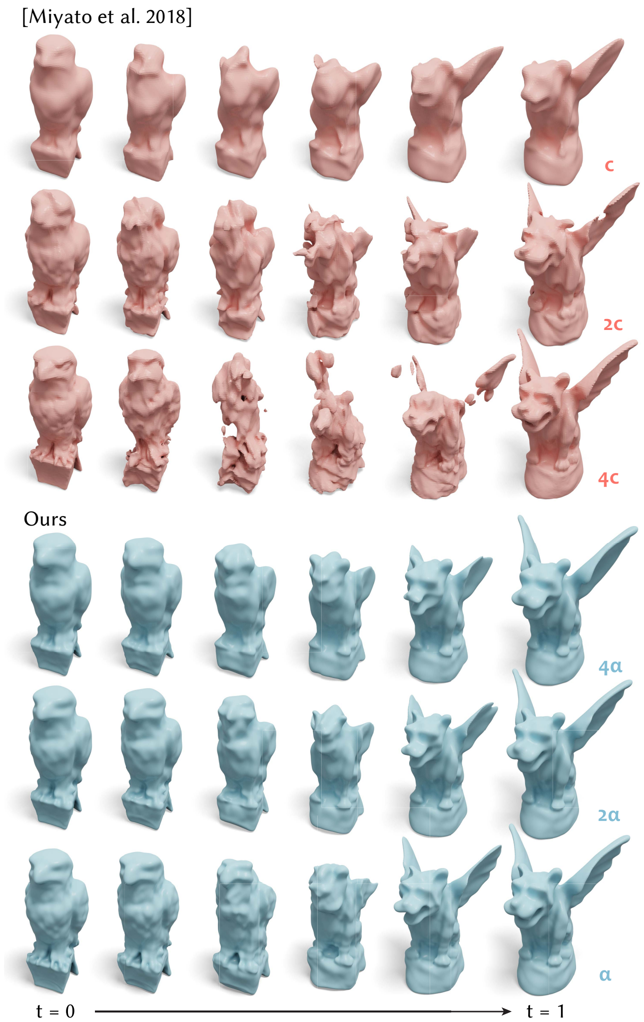


Fig. 16. Lipschitz constrained networks, such as [Miyato et al. 2018], are sensitive to the change of the prescribed Lipschitz bound. We can observe a dramatic change from too smooth (1st row) to too non-smooth (3rd row) with a minor logarithmic scaling of the initial Lipschitz bound c. In contrast, our method (blue) is more robust with respect to the change of our regularization weight α.