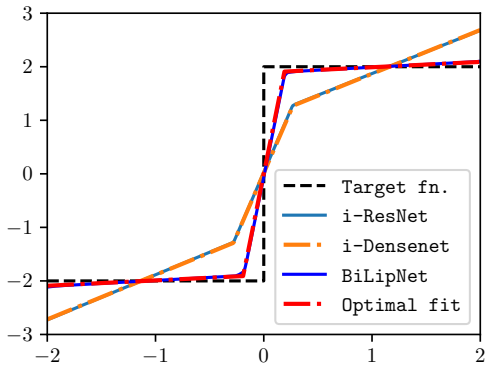
Monotone, Bi-Lipschitz, and Polyak-Łojasiewicz Networks

Ruigang Wang [1] **Krishnamurthy (Dj) Dvijotham** [2] **Ian R. Manchester** [1]

Abstract

This paper presents a new *bi-Lipschitz* invertible neural network, the BiLipNet, which has the ability to smoothly control both its *Lipschitzness* (output sensitivity to input perturbations) and *inverse Lipschitzness* (input distinguishability from different outputs). The second main contribution is a new scalar-output network, the PLNet, which is a composition of a BiLipNet and a quadratic potential. We show that PLNet satisfies the Polyak-Łojasiewicz condition and can be applied to learn non-convex surrogate losses with a unique and efficiently-computable global minimum. The central technical element in these networks is a novel invertible residual layer with certified strong monotonicity and Lipschitzness, which we compose with orthogonal layers to build the BiLipNet. The certification of these properties is based on incremental quadratic constraints, resulting in much tighter bounds than can be achieved with spectral normalization. Moreover, we formulate the calculation of the inverse of a BiLipNet – and hence the minimum of a PLNet – as a series of three-operator splitting problems, for which fast algorithms can be applied.



Model	inv. Lip. (↓)	Lip. (↑)	loss (↓)
i-ResNet	0.80	4.69	0.2090
i-DenseNet	0.82	4.66	0.2091
BiLipNet	**0.11**	**9.97**	**0.0685**
Best Possible	0.10	10.0	0.0677

Figure 1. Fitting a step function, which is not Lipschitz, with certified $(0.1, 10)$-Lipschitz models. Compared to the analytically-computed optimum, the proposed BiLipNet achieves much tighter bounds than models based on spectral normalization.

1. Introduction

In many applications, it is desirable to learn neural networks with *certified input-output behaviors*, i.e., certain properties that are guaranteed by design. For example, Lipschitz-bounded networks have proven to be beneficial for stabilizing of generative adversarial network (GAN) training (Arjovsky et al., 2017; Gulrajani et al., 2017), certifying robustness against adversarial attacks (Tsuzuku et al., 2018; Singla & Feizi, 2021; Zhang et al., 2021; Araujo et al., 2023; Wang & Manchester, 2023) and robust reinforcement learning (Russo & Proutiere, 2021; Barbara et al., 2024).

Another input-output property – *invertibility* has received much attention in the deep learning literature since the introduction of *normalizing flows* (Dinh et al., 2015) for probability-density learning. Invertible neural networks have been applied in applications such as generative modeling (Dinh et al., 2017; Kingma & Dhariwal, 2018), probabilistic inference (Bauer & Mnih, 2019; Ward et al., 2019; Louizos & Welling, 2017), solving inverse problems (Ardizzone et al., 2018) and uncertainty estimation (Liu et al., 2020). A common way to construct invertible networks is to compose invertible affine transformations with more sophisticated invertible layers, including coupling flows (Dinh et al., 2017; Kingma & Dhariwal, 2018), auto-regressive models (Huang et al., 2018; De Cao et al., 2020; Ho et al., 2019), invertible residual layers (Chen et al., 2019; Behrmann et al., 2019), monotone networks (Ahn et al., 2022), and neural ordinary differential equations (Grathwohl et al., 2019), see also in the surveys (Papamakarios et al., 2021; Kobyzev et al., 2020).

[1]Australian Centre for Robotics, School of Aerospace, Mechanical and Mechatronic Engineering, The University of Sydney, Sydney, NSW 2006, Australia. [2]Google DeepMind. Correspondence to: Ruigang Wang <ruigang.wang@sydney.edu.au>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

However, (Behrmann et al., 2021) observed that commonly-used invertible networks suffer from exploding inverses and are thus prone to becoming numerically non-invertible. This observation motivates the input-output property of *bi-Lipschitzness*. A layer $\mathcal{F} : \mathsf{R}^n \to \mathsf{R}^n$ is said to be bi-Lipschitz with bound of (μ, ν), or simply (μ, ν)-Lipschitz, if the following inequalities hold for all $x, x^0 \in \mathsf{R}^n$:

$$\mu\|x - x^0\| \leq \|\mathcal{F}(x) - \mathcal{F}(x^0)\| \leq \nu\|x - x^0\|,$$

where $\|\cdot\|$ is the Euclidean norm. The bound ν controls the output sensitivity to input perturbations while μ controls the input distinguishability from different outputs (Liu et al., 2020). We call μ as the *inverse Lipschitz* bound of $\mathcal{F}$ as $\mathcal{F}^{-1}$ exists and is $1/\mu$-Lipschitz. The ratio $\tau := \nu/\mu$ is called *distortion* (Liang et al., 2023), which is the upper bound of the condition number of the Jacobian matrix of $\mathcal{F}$. A larger distortion implies more expressive flexibility in the model.

In this paper we argue that the bi-Lipschitz property is also useful for learning of surrogate loss (or reward) functions. Given some input/output pairs of a loss function, the objective is to learn a function which matches the observed data and is "easy to optimize" in some sense. This problem appears in many areas, including Q-learning with continuous action spaces, see e.g. (Gu et al., 2016; Amos et al., 2017; Ryu et al., 2019), offline data-driven optimization (Grudzien et al., 2024), learning reward models in inverse reinforcement learning (Arora & Doshi, 2021), and data-driven surrogate losses for engineering process optimization (Cozad et al., 2014; Misener & Biegler, 2023). An important contribution was the input convex neural network (ICNN) (Amos et al., 2017). However, the requirement of input convexity could be too strong in many applications.

1.1. Contributions

- We propose a novel strongly monotone and Lipschitz residual layer of the form $\mathcal{F}(x) = \mu x + \mathcal{H}(x)$. For the nonlinear block $\mathcal{H}$, we introduce a new architecture – *feed-through network* (FTN), which takes a multi-layer perceptron (MLP) as its backbone and adds connections from each hidden layer to the input and output variables. For deep networks, this architecture can improve the model expressivity without suffering from vanishing gradients.
- We parameterize FTNs with certified *strong monotonicity* (which implies inverse Lipschitzness) and Lipschitzness for $\mathcal{F}$ via the integral quadratic constraint (IQC) framework (Megretski & Rantzer, 1997) and the Cayley transform.
- By composing strongly-monotone and Lipschitz FTN layers with orthogonal affine layers we obtain the *BiLipNet*, a new network architecture with smoothly-parameterized bi-Lipschitz bounds.
- We formulate the model inversion $\mathcal{F}^{-1}$ as a three-operator splitting problem, which admits a numerically efficient solver (Davis & Yin, 2017).
- We introduce a new scalar-output network $f : \mathsf{R}^n \to \mathsf{R}$, which we call a *Polyak-Łojasiewicz* network (or PLNet) since it satisfies the condition of the same name (Polyak, 1963; Lojasiewicz, 1963). It consists of a bi-Lipschitz network composed with a quadratic potential, and automatically satisfies favourable properties for surrogate loss learning, in particular existence of a unique global optimum which is efficiently computable.

1.2. Related work

Bi-Lipschitz invertible layer. In literature, there are two types of invertible layers closely related to our models. The first is the invertible residual layer $\mathcal{F}(x) = x + \mathcal{H}(x)$ (Chen et al., 2019; Behrmann et al., 2019), where the nonlinear block $\mathcal{H}$ is a shallow network with Lipschitz bound of $c < 1$. In (Perugachi-Diaz et al., 2021), $\mathcal{H}$ is further extended to a deep MLP. It is easy to show that $\mathcal{F}$ is $(1 - c)$-inverse Lipschitz and $(1 + c)$-Lipschitz. In both cases, the Lipschitz regularization is via spectral normalization (Miyato et al., 2018), which we observe to be very conservative (see Figure 1). Alternatively, a bi-Lipschitz layer can be defined by an implicit equation (Lu et al., 2021; Ahn et al., 2022). However, these require an iterative solver for both the forward and inverse model inference. In contrast, our model has an explicit forward pass and iterative solution is only required for the inverse.

IQC-based Lipschitz estimation and training. In (Fazlyab et al., 2019), the IQC framework of (Megretski & Rantzer, 1997) was first applied to obtain accurate Lipschitz bound estimation of deep networks with slope-restricted activations. It was later pointed out by (Wang et al., 2022) that IQC for Lipschitzness (Fazlyab et al., 2019) is Shor's relaxation of a "Rayleigh quotient" quadratically constrained quadratic programming (QCQP). Direct (i.e. unconstrained) parameterizations based on IQC were were proposed in (Revay et al., 2020) for deep equilibrium networks, in (Araujo et al., 2023) for residual networks, for deep MLPs and CNNs in (Wang & Manchester, 2023), and recurrent models in (Revay et al., 2023). It was pointed out by (Havens et al., 2023) that many recent Lipschitz model parameterizations (Meunier et al., 2022; Prach & Lampert, 2022; Araujo et al., 2023; Wang & Manchester, 2023) are special cases of (Revay et al., 2020). In a recent work (Pauli et al., 2024), the IQC-based Lipschitz estimation was recently extended to more general activations such as GroupSort and MaxMin. All of these are for one-sided (upper) Lipschitzness, whereas our work applies the IQC framework for monotonicity and bi-Lipschitzness.

Bi-Lipschitz networks for learning-based surrogate optimization. (Liang et al., 2023) uses Bi-Lipschitz networks to learn a surrogate constraint set while our work focuses on surrogate loss learning. Both works take distortion bound as an important regularization technique. The difference is that the distortion estimation in (Liang et al., 2023) is based on data samples while our work offers certified and smoothly-parameterized distortion bounds.

2. Preliminaries

We give some definitions for a mapping $\mathcal{F} : \mathsf{R}^n \to \mathsf{R}^n$.

Definition 2.1. $\mathcal{F}$ is said to be μ**-strongly monotone** with $\mu > 0$ if for all $x, x^0 \in \mathsf{R}^n$ we have

$$\langle \mathcal{F}(x) - \mathcal{F}(x^0), x - x^0 \rangle \geq \mu \|x - x^0\|^2,$$

where $\langle \cdot, \cdot \rangle$ is the Euclidean inner product: $\langle a, b \rangle = a^\top b$. $\mathcal{F}$ is **monotone** if the above condition holds for $\mu = 0$.

Definition 2.2. $\mathcal{F}$ is said to be ν**-Lipschitz** with $\nu > 0$ if

$$\|\mathcal{F}(x) - \mathcal{F}(x^0)\| \leq \nu \|x - x^0\|, \quad \forall x, x^0 \in \mathsf{R}^n.$$

$\mathcal{F}$ is said to be μ**-inverse Lipschitz** with $\mu > 0$ if

$$\|\mathcal{F}(x) - \mathcal{F}(x^0)\| \geq \mu \|x - x^0\|, \quad \forall x_1, x_2 \in \mathsf{R}^n.$$

$\mathcal{F}$ is said to be **bi-Lipschitz** with $\nu \geq \mu > 0$, or simply (μ, ν)-Lipschitz, if it is μ-inverse Lipschitz and ν-Lipschitz.

For any (μ, ν)-Lipschitz mapping $\mathcal{F}$, its inverse $\mathcal{F}^{-1}$ is well-defined and $(1/\nu, 1/\mu)$-Lipschitz (Yeh, 2006). By the Cauchy–Schwarz inequality, strong monotonicity implies inverse Lipschitzness, see Figure 2. A notable difference between monotonicity and bi-Lipschitzness is their composition behaviour. Given two bi-Lipschitz mappings $\mathcal{F}_1, \mathcal{F}_2$, their composition $\mathcal{F} = \mathcal{F}_2 \circ \mathcal{F}_1$ is also bi-Lipschitz with bound of $(\mu_1\mu_2, \nu_1\nu_2)$ where (μ_1, ν_1) and (μ_2, ν_2) are the bi-Lipschitz bounds of $\mathcal{F}_1$ and $\mathcal{F}_2$, respectively. However, given two strongly monotone $\mathcal{F}_1, \mathcal{F}_2$ with monotonicity bounds μ_1, μ_2, the composition $\mathcal{F} = \mathcal{F}_2 \circ \mathcal{F}_1$ does *not* need to be strongly monotone. However, it is still $\mu_1\mu_2$-inverse Lipschitz. To quantify the flexibility of bi-Lipschitz maps, we introduce the following:

Definition 2.3. $\mathcal{F}$ satisfies a **distortion bound** τ with $\tau \geq 1$ if $\mathcal{F}$ is (μ, ν)-Lipschitz with $\tau = \nu/\mu$.

For an invertible affine mapping $\mathcal{F}(x) = Px + q$, the condition number of P is a distortion bound. An orthogonal mapping (i.e., $P^\top P = I$) has the smallest possible model distortion $\tau = 1$. Distortion bounds satisfy a composition property, i.e., if $\mathcal{F}_1, \mathcal{F}_2$ have distortion bounds of τ_1, τ_2, then $\mathcal{F}_2 \circ \mathcal{F}_1$ satisfies a distortion bound of $\tau_1\tau_2$. Both $\mathcal{F}$ and $\mathcal{F}^{-1}$ have the same distortion.

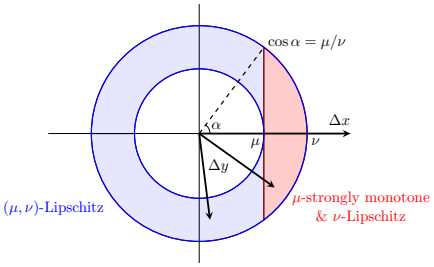


Figure 2. This figure depicts the possible ranges of $\Delta y = F(x')$ $F(x)$ on R^2 for a given $\Delta x = x' \quad x$. The ring (blue area) is for (μ, ν)-Lipschitz F while the half moon (red area) is for a μ-strongly monotone and ν-Lipschitz F. The largest angle between Δx and Δy satisfies $\cos\alpha = \tau^{-1}$ with $\tau = \nu/\mu$ as the distortion.

Surrogate loss learning. Let $\mathcal{D}$ be a dataset containing finite samples of $x_i \in \mathsf{R}^n$ and $y_i = \mathsf{f}(x_i) \in \mathsf{R}$ where f is an unknown loss function. The task is to learn a surrogate loss $\hat{f}$ from $\mathcal{D}$, i.e., $\hat{f} = \arg\min_{f \in \mathfrak{F}} \mathsf{E}_{(x,y) \quad \mathcal{D}} \left[(f(x) - y)^2\right]$ where $\mathfrak{F}$ is the model set (e.g. neural networks). In many applications, it is highly desirable that each $f \in \mathfrak{F}$ has a unique and efficiently-computable global minimum. An important model class is the input convex neural network (ICNN) (Amos et al., 2017). Since $f \in \mathfrak{F}$ is convex w.r.t x, then any local minimum is a global minimum. Moreover, there exists a rich literature for convex optimization. Although convexity is more favourable for downstream optimization problems, it might be a very stringent requirement for fitting the dataset $\mathcal{D}$. In this work we aim to construct a model set $\mathfrak{F}$ such that every $f \in \mathfrak{F}$ does not need to be convex but still poses those favourable properties for optimization. In Section 5, we will show that the construction of such model set relies on bi-Lipschitz neural networks.

3. Monotone and bi-Lipschitz Networks

In this section we first present the construction of μ-strongly monotone and ν-Lipschitz residual layers of the form $\mathcal{F}(x) = \mu x + \mathcal{H}(x)$. We then construct bi-Lipschitz networks by deep composition of the new monotone and Lipschitz layers with orthogonal linear layers.

3.1. Feed-through network

For the nonlinear block $\mathcal{H}$, we introduce a network architecture, called *feed-through network* (FTN), which takes an MLP as its backbone and then connects each hidden layer to input and output variables, see Figure 3. To be specific, the residual layer $\mathcal{F}(x) = \mu x + \mathcal{H}(x)$ can be written as

$$\begin{aligned} z_k &= \sigma(W_k z_{k-1} + U_k x + b_k), \; z_0 = 0 \\ y &= \mu x + \sum_{k=1}^{L} Y_k z_k + b_y \end{aligned} \tag{1}$$

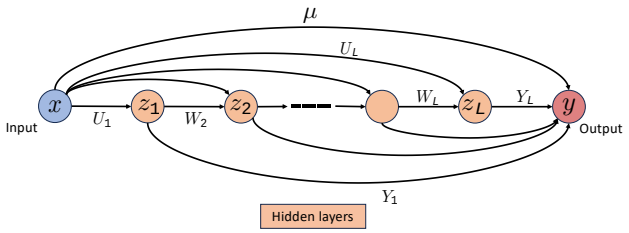


Figure 3. The proposed invertible residual network $F(x) = \mu x + H(x)$ where the nonlinear block H is a feed-through network, whose hidden layers are directly connected to the input and output.

where $z_k \in \mathbb{R}^{m_k}$ are the hidden variables, U_k, W_k, Y_k and b_k, b_y are the learnable weights and biases, respectively. Throughout the paper we assume that the activation σ is a scalar nonlinearity with slope restricted in $[0, 1]$, which is satisfied (possibly with rescaling) by common activation functions such as ReLU, tanh, and sigmoid.

Remark 3.1. FTN contains both short paths $x \to z_i \to y$ preventing vanishing gradients and long paths $x \to z_i \to \cdots \to z_j \to y$ improving model expressivity (see Figure 3).

3.2. SDP conditions for monotonicity and Lipschitzness

The first step towards our parameterization is to establish strong monotonicity and Lipschitzness for $\mathcal{F}$ via semidefinite programming (SDP) conditions. For this, we rewrite $\mathcal{F}$ in a compact form:

$$z = \sigma(Wz + Ux + b), \quad y = \mu x + Yz + b_y \tag{2}$$

where $z = \begin{bmatrix} z_1^\top & \cdots & z_L^\top \end{bmatrix}^\top$, $b = \begin{bmatrix} b_1^\top & \cdots & b_L^\top \end{bmatrix}^\top$, and

$$W = \begin{bmatrix} 0 & & & \\ W_2 & 0 & & \\ & \ddots & \ddots & \\ & & W_L & 0 \end{bmatrix}, \quad U = \begin{bmatrix} U_1 \\ U_2 \\ \vdots \\ U_L \end{bmatrix},$$
$$Y = \begin{bmatrix} Y_1 & Y_2 & \cdots & Y_L \end{bmatrix}.$$

Theorem 3.2. *$\mathcal{F}$ is μ-strongly monotone and ν-Lipschitz if there exists a $\Lambda \in \mathbb{D}_+^m$, where $\mathbb{D}_+^m$ is the set of positive diagonal matrices, such that the following conditions hold:*

$$Y = U^\top \Lambda, \quad 2\Lambda - \Lambda W - W^\top \Lambda \succeq \frac{2}{\gamma} Y^\top Y \tag{3}$$

where $\gamma = \nu - \mu > 0$.

Remark 3.3. The above conditions are obtained by applying the IQC theory (Megretski & Rantzer, 1997) to (2).

3.3. Model parameterization

Let Θ be the set of all $\theta = \{U, W, Y, \Lambda\}$ such that Condition (3) holds. Since it is generally not scalable to train a model with SDP constraints, we instead construct a *direct parameterization*, i.e. both unconstrained and complete:

Definition 3.4. A **direct parameterization** of a constraint set Θ is a surjective differentiable mapping $\mathcal{M} : \mathbb{R}^N \to \Theta$, i.e. for any $\phi \in \mathbb{R}^N$ we have $\mathcal{M}(\phi) \in \Theta$, and the image of $\mathbb{R}^N$ maps onto Θ, i.e. $\mathcal{M}(\mathbb{R}^N) = \Theta$.

A direct parameterization allows us to replace a constrained optimization over $\theta \in \Theta$ with an unconstrained optimization over $\phi \in \mathbb{R}^N$ without loss of generality. This enables use of standard first-order optimization algorithms such as SGD or ADAM (Kingma & Ba, 2015).

We now construct a direct parameterization for FTNs satisfying (3). Here we present the main ideas, see Appendix A for full details. First, we introduce the free parameters

$$\phi = \{F^p, F^q\} \cup \{d_k, F_k^a, F_k^b\}_{1 \le k \le L}$$

where $F^p \in \mathbb{R}^{n \times n}$, $F^q \in \mathbb{R}^{m \times n}$, $d_k \in \mathbb{R}^{m_k}$, $F_k^a \in \mathbb{R}^{m_k \times m_k}$ and $F_k^b \in \mathbb{R}^{m_{k-1} \times m_k}$ with $m_0 = 0$. Then, we compute some intermediate variables $\Psi_k = \operatorname{diag}(e^{d_k})$ and

$$\begin{bmatrix} A_k^\top \\ B_k^\top \end{bmatrix} = \operatorname{Cayley}\left(\begin{bmatrix} F_k^a \\ F_k^b \end{bmatrix}\right), \quad \begin{bmatrix} P \\ Q \end{bmatrix} = \operatorname{Cayley}\left(\begin{bmatrix} F^p \\ F^q \end{bmatrix}\right)$$

where $\operatorname{Cayley} : \mathbb{R}^{n \times p} \to \mathbb{R}^{n \times p}$ with $n \ge p$ is defined by

$$J = \operatorname{Cayley}\left(\begin{bmatrix} G \\ H \end{bmatrix}\right) := \begin{bmatrix} (I+Z)(I-Z)^{-1} \\ -2H(I+Z)^{-1} \end{bmatrix} \tag{4}$$

with $Z = G^\top - G + H^\top H$. It can be verified that $J^\top J = I$ for any $G \in \mathbb{R}^{p \times p}$ and $H \in \mathbb{R}^{(n-p) \times p}$. Note that P will not be used for further weight construction as its purpose is to ensure that $Q^\top Q \preceq I$. Next we set

$$V_k = 2B_k A_k^\top, \quad S_k = A_k Q_k - B_k Q_{k-1}$$

where $Q = \begin{bmatrix} Q_1^\top & \cdots & Q_L^\top \end{bmatrix}^\top$ and $B_1 = 0$, $Q_0 = 0$. Finally, we construct the weights in (1) as:

$$\begin{aligned} U_k &= \sqrt{2\gamma}\Psi_k^{-1} S_k, \; W_k = \Psi_k^{-1} V_k \Psi_{k-1}, \\ Y_k &= \sqrt{\frac{\gamma}{2}} S_k^\top \Psi_k, \; \Lambda_k = \frac{1}{2}\Psi_k^2. \end{aligned} \tag{5}$$

Proposition 3.5. *The model parameterization $\mathcal{M}$ defined in (5) is a direct parameterization for the set Θ, i.e. all models (1) satisfying Condition (3).*

This means that we can learn the free parameter ϕ using first-order methods without any loss of model expressivity.

The construction is now done, but we note that Ψ_k is shared between layers k and $k+1$. To have a modular implementation, we introduce new variables $\hat{z} = \Psi z$ and bias $\hat{b} = \Psi b$ with $\Psi = \operatorname{diag}(\Psi_1, \Psi_2, \ldots, \Psi_L)$. Then, (2) can be rewritten as follows (see Appendix A)

$$\hat{z} = \hat{\sigma}\big(V\hat{z} + \sqrt{2\gamma} S x + \hat{b}\big), \; y = \mu x + \sqrt{\gamma/2} S^\top \hat{z} + b_y \tag{6}$$

where $\hat{\sigma}(x) := \Psi\sigma\left(\Psi^{-1}x\right)$ is a $(0,1)$-Lipschitz layer with learnable scaling Ψ, the weights S, V can be written as

$$S = \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_L \end{bmatrix}, \quad V = \begin{bmatrix} 0 & & & \\ V_2 & 0 & & \\ & \ddots & \ddots & \\ & & V_L & 0 \end{bmatrix}. \tag{7}$$

Number of free parameters. Consider an L-layer FTN (1) where each layer has the same width, i.e. $m_k = d$. The bi-Lipschitz network based on spectral normalization (Liu et al., 2020) has $2Ld^2$ free parameters while our model size is $(3L+1)d^2 + Ld$. Since the Ld^2 term dominates for deep and wide networks, our model has roughly 1.5 times as many parameter as the model from (Liu et al., 2020).

3.4. Bi-Lipschitz networks

We construct bi-Lipschitz networks (referred as BiLipNets) by composing strongly monotone and Lipschitz layers,

$$\mathcal{G} = \mathcal{O}_{K+1} \circ \mathcal{F}_K \circ \mathcal{O}_K \circ \mathcal{F}_{K-1} \circ \cdots \circ \mathcal{O}_2 \circ \mathcal{F}_1 \circ \mathcal{O}_1 \tag{8}$$

where $\mathcal{O}_k(x) = P_k x + q_k$ with $P_k^\top P_k = I$ is an orthogonal layer and $\mathcal{F}_k$ is a μ_k-strongly monotone and ν_k-Lipschitz layer (6). By the composition rule, the above BiLipNet is (μ,ν)-Lipschitz with $\mu = \prod_{k=1}^{L}\mu_k$ and $\nu = \prod_{k=1}^{L}\nu_k$. The orthogonal matrix P can be parameterized via the Cayley transformation (4) or Householder transformation (Singla et al., 2022). Since the distortion of $\mathcal{O}_k$ is 1, it can improve network expressivity without increasing model distortion.

In some applications, e.g., normalising flows (Dinh et al., 2015; Papamakarios et al., 2021), we need to compute the inverse of $\mathcal{G}$, which can be done in a backward manner:

$$\mathcal{G}^{-1}(y) = \mathcal{O}_1^{-1} \circ \mathcal{F}_1^{-1} \circ \cdots \circ \mathcal{O}_K^{-1} \circ \mathcal{F}_K^{-1} \circ \mathcal{O}_{K+1}^{-1}(y), \tag{9}$$

where $\mathcal{O}_k$ has an explicit inverse $\mathcal{O}_k^{-1}(y) = P_k^\top(y - q_k)$. Computing the inverse $\mathcal{F}_k^{-1}(y)$ requires an iterative solver, which will be addressed in Section 4.

Partially bi-Lipschitz networks. A neural network $\tilde{\mathcal{G}} : \mathbb{R}^n \times \mathbb{R}^l \to \mathbb{R}^n$ is said to be *partially bi-Lipschitz* if for any fixed value of $p \in \mathbb{R}^l$, the mapping $y = \tilde{\mathcal{G}}(x;p)$ is (μ,ν)-Lipschitz from x to y. We can construct such mappings via $\tilde{\mathcal{G}}(x;p) = \mathcal{G}_{h(p)}(x)$ where $\mathcal{G}$ is a(μ,ν)-Lipschitz network for any free parameter $\phi \in \mathbb{R}^N$ and $h : p \to \phi$ is a new learnable function. Since the dimension of ϕ is often very high, a practical approach is to make ϕ partially depend on p. For instance, we can learn p-dependent bias via an MLP while the weight matrices of $\mathcal{G}$ is independent of p.

4. Model inverse via operator splitting

In this section we give an efficient algorithm to compute $\mathcal{F}^{-1}(y)$ where $\mathcal{F}$ is a μ-strongly monotone and ν-Lipschitz layer (6). First, we write its model inverse $\mathcal{F}^{-1}$ as

$$\begin{aligned} \hat{z} &= \hat{\sigma}\left(\left(V - \frac{\gamma}{\mu}SS^\top\right)\hat{z} + b_z\right) \\ x &= \frac{1}{\mu}(y - b_y - \sqrt{\gamma/2}S^\top\hat{z}) \end{aligned} \tag{10}$$

with $b_z = \sqrt{2\gamma}/\mu S(y - b_y) + \hat{b}$. Both $\mathcal{F}$ and $\mathcal{F}^{-1}$ can be treated as special cases of deep equilibrium networks (Bai et al., 2019; Winston & Kolter, 2020; Revay et al., 2020) or implicit networks (El Ghaoui et al., 2021). The difference is that $\mathcal{F}$ has an explicit formula due to the strictly lower-triangular V while $\mathcal{F}^{-1}$ is an implicit equation as $SS^\top$ is a full matrix. A natural question for (10) is its **well-posedness**, i.e., for any $y \in \mathbb{R}^n$, does there exists a unique $\hat{z} \in \mathbb{R}^m$ satisfying (10)?

Proposition 4.1. *$\mathcal{F}^{-1}$ is well-posed if V, S are given by (7).*

Certain classes of equilibrium networks were solved via two-operator splitting problems (Winston & Kolter, 2020; Revay et al., 2020). We follow a similar strategy, but our structure admits a three-operator splitting, see Proposition 4.2 with background in Appendix B. To state the result, we first recall the following fact from (Li et al., 2019). For the monotone and 1-Lipschitz activation $\hat{\sigma}$, there exists a proper convex function $f : \mathbb{R}^n \to \mathbb{R}$ satisfying $\hat{\sigma}(\cdot) = \mathbf{prox}_f^1(\cdot)$ with

$$\mathbf{prox}_f(x) = \arg\min_{z \in \mathbb{R}^n} \frac{1}{2}\|x - z\|^2 + \alpha f(z).$$

A list of f for popular activations is given in Appendix B.1.

Proposition 4.2. *Finding a solution $\hat{z} \in \mathbb{R}^m$ to (10) is equivalent to finding a zero to the three-operator splitting problem $0 \in \mathcal{A}(z) + \mathcal{B}(z) + \mathcal{C}(z)$ where $\mathcal{A}, \mathcal{B}, \mathcal{C}$ are monotone operators defined by*

$$\mathcal{A}(z) = (I - V)z - b_z, \ \mathcal{B}(z) = \partial f(z), \ \mathcal{C}(z) = \frac{\gamma}{\mu}SS^\top z$$

where f satisfies $\hat{\sigma}(\cdot) = \mathbf{prox}_f^1(\cdot)$.

For three-operator problems, the Davis-Yin splitting algorithm (**DYS**) (Davis & Yin, 2017) can be applied, obtaining the following fixed-point iteration:

$$\begin{aligned} z^{k+1/2} &= \mathbf{prox}_f(u^k) \\ u^{k+1/2} &= 2z^{k+1/2} - u^k \\ z^{k+1} &= R_{\mathcal{A}}(u^{k+1/2} - \alpha\mathcal{C}(z^{k+1/2})) \\ u^{k+1} &= u^k + z^{k+1} - z^{k+1/2} \end{aligned} \tag{11}$$

where $R_{\mathcal{A}}(v) = ((1+\alpha)I - \alpha V)^{-1}(v + \alpha b_z)$. Since V is strictly lower triangular, we can solve $R_{\mathcal{A}}(v)$ using forward substitution. Furthermore, we can show that (11) is guaranteed to converge with $\alpha \in \left(0, \frac{1}{1}\right)$, where τ is the model distortion.

5. Polyak-Łojasiewicz Networks

We call a network $f : \mathsf{R}^n \to \mathsf{R}$ a *Polyak-Łojasiewicz (PL) network*, or PLNet for short, if it satisfies the following PL condition (Polyak, 1963; Lojasiewicz, 1963):

$$\frac{1}{2}\|\nabla_x f(x)\|^2 \geq m(f(x) - \min_x f(x)), \; \forall x \in \mathsf{R}^n, \quad (12)$$

where $m > 0$. The PL condition is significant in optimization since it is weaker than convexity, but still implies that gradient methods converge to a global minimum with a linear rate (Karimi et al., 2016), making PLNet a promising candidate for learning a surrogate loss models.

Proposition 5.1. *If $\mathcal{G}$ is μ-inverse Lipschitz, then*

$$f(x) = \frac{1}{2}\|\mathcal{G}(x)\|^2 + c, \quad c \in \mathsf{R} \quad (13)$$

is a PLNet with $m = \mu^2$.

Remark 5.2. We can further relax the quadratic assumption: $f(x) = h\big(\mathcal{G}(x)\big)$ is a PLNet if $h : \mathsf{R}^n \to \mathsf{R}$ is strongly convex (Karimi et al., 2016).

Remark 5.3. For parametric optimization problem, one can learn a surrogate loss via $f(x; p) = 1/2\|\tilde{\mathcal{G}}(x; p)\|^2 + c$ where $p \in \mathsf{R}^m$ is the problem-specific parameter and $\tilde{\mathcal{G}}$ is a partially bi-Lipschitz network.

Remark 5.4. Any sub-level set $\mathsf{L} = \{x : f(x) < \alpha\}$ with $\alpha > c$ is homeomorphic to a unit ball, making PLNets suitable for neural Lyapunov functions (Wilson, 1967). Applications of PLNets to learning Lyapunov stable neural dynamics can be found in (Cheng et al., 2024).

Computing global optimum of a PLNet. If f takes the form (13) and $\mathcal{G}$ is bi-Lipschitz network (8), then f has a unique global optimum $x^? = \mathcal{G}^{-1}(0)$ with $\mathcal{G}^{-1}$ given by (9). This can be efficiently computed by analytical inversion of orthogonal layers and applying the DYS algorithm (11) to monotone and Lipschitz layers.

Limitations of gradient descent for finding global optimum. An alternative way to compute the global optimum $x^?$ is the standard gradient descent (GD) method $x^{k+1} = x^k - \alpha\nabla_x f(x^k)$. If $\nabla_x f$ is L-Lipschitz, then the above GD solver with $\alpha = 1/L$ has a linear global convergence rate of $1 - m/L$ with $m = \mu^2$ (Karimi et al., 2016). However, this method has two drawbacks. First, the gradient function $\nabla_x f$ may not be globally Lipschitz, see Example 5.5. Secondly, even if a global Lipschitz bound exists, it is generally hard to estimate.

Example 5.5. Consider a scalar function $f(x) = 0.5g^2(x)$ with $g(x) = 2x + \sin x$, which satisfies the PL condition. Note that $\partial f/\partial x = (2 + \cos x)(2x + \sin x)$ is not globally Lipschitz due to the term $2x \cos x$.

6. Experiments

Here we present experiments which explore the expressive quality of the proposed models, regularisation via model distortion, and performance of the DYS solution method. Code is available at `https://github.com/acfr/PLNet`.

6.1. Uncertainty quantification via neural Gaussian process

It was shown in (Liu et al., 2020) that accurate uncertainty quantification of neural network models depends on a model's ability to quantify the distance of a test example from the training data. This *distance-awareness* can be achieved with bi-Lipschitz residual layers $\mathcal{F}(x) = x + \mathcal{H}(x)$ and a Gaussian process output layer. In (Liu et al., 2020) this is achieved by imposing Lipschitz bound of $0 < c < 1$ for $\mathcal{H}$ via spectral normalization. The resulting model is called Spectral-normalized Neural Gaussian Process (SNGP). In this section we examine the benefits of using the proposed BiLipNet in place of spectrally-normalized layers.

Toy example. Using the two-moon dataset, we compare our (μ, ν)-Lipschitz network to an SNGP using a 3-layer i-ResNet under the same bi-Lipschitz constraints, i.e., $\mu = (1 - c)^3$ and $\nu = (1 + c)^3$, see Figure 4. For the lower-distortion case (i.e., small $c = 0.1$), SNGP fails to completely separate the train and out-of-distribution (OOD) data due to its loose Lipschitz bound. Our model can distinguish the OOD examples from training dataset and the uncertainty surface is close to the SNGP with much higher distortion ($c = 0.9$). As the model distortion increases, our model can have an uncertainty surface very close the dataset. The uncertainty surface of SNGP does not change much from $c = 0.1$ to $c = 0.9$, see Figure 4 and additional results in Appendix D.2.

CIFAR-10/100. For image datasets, the SNGP model in (Liu et al., 2020) contains three bi-Lipschitz components, each with four residual layers of the form $x + \mathcal{H}(x)$ where $\mathcal{H}$ is constructed to be c-Lipschitz using spectral normalization. To ensure certifiable bi-Lipschitzness, we modify the SNGP model by choosing $c \in (0, 1)$ and removing batch normalization from $\mathcal{H}$ since it may re-scale a layer's spectral norm in unexpected ways (Liu et al., 2023). The results of SNGP with batch normalization can be found in Appendix D.2. Our BiLipNet model has a similar architecture as SNGP except replacing the bi-Lipschitz components with the proposed (μ, ν)-Lipschitz network (8). To ensure both models have the same bi-Lipschitz bound, we choose $\mu = (1 - c)^4$ and $\nu = (1 + c)^4$.

Table 1 reports the results of SNGP and BiLipNet under different bounds $c = 0.95, 0.65, 0.35$. For CIFAR-10 dataset, our model uniformly outperforms SNGP on both clean

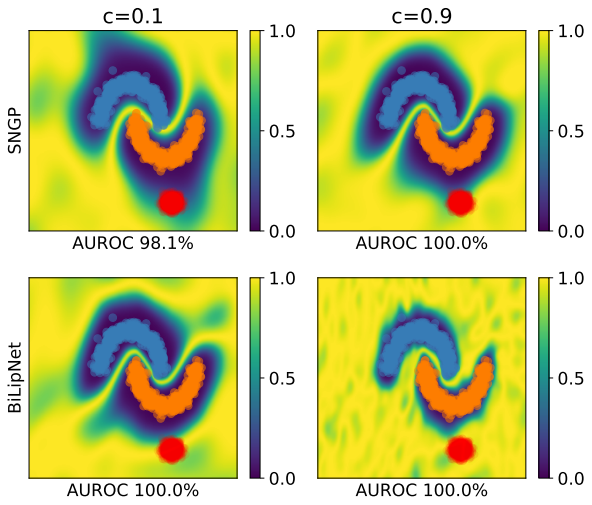


Figure 4. Predictive uncertainty of different NGPs with the same bi-Lipschitz bound. The points from dark blue and regions are classified as in-domain distribution and OOD data, respectively. Light blue and orange points (different colors indicate different labels) are training samples from the two-moon dataset. The red points are ODD test examples. For the case with small distortion, our model can still distinguish the train and OOD data, achieving similar results of SNGP with large distortion.

and corrupted data, i.e., it achieves higher accuracy (about $10 \sim 20\%$ improvement), lower expected calibration error (ECE) and negative log liklihood (NLL). Similar conclusion also holds for CIFAR-100 on accuracy and NLL, though our model has sightly higher ECE.

As with the previous toy example, our model with small distortion ($\tau = 18.6$ for $c = 0.35$) achieves better accuracy than SNGP with large distortion ($\tau = 2.3 \times 10^6$ for $c = 0.95$). Thus, we observe that our parameterization is much more expressive for a given distortion bound.

6.2. Surrogate loss learning

We explore the PLNet's performance with the Rosenbrock function $r(x, y) = 1/200(x-1)^2 + 0.5(y-x^2)^2$ and its higher-dimensional generalizations. The Rosenbrock function is a classical test problem for optimization, since it is non-convex but a unique global minimum point $(1, 1)$, at which the Hessian is poorly-conditioned. We also consider the sum of the Rosenbrock function and a 2D sine wave function, which still has a unique global minimum at $(1, 1)$ while having many local minima, see Appendix D.1.

We learned models of the form (13) where $\mathcal{G}$ is parameterized by MLP, i-ResNet (Behrmann et al., 2019), i-DenseNet (Perugachi-Diaz et al., 2021) and the proposed BiLipNet (8). We also trained the ICNN, a scalar-output model which is convex w.r.t. inputs (Amos et al., 2017).

From Figure 7, we have the following observations. The unconstrained MLP can achieve small test errors. However, it has many local minima near the valley $y = x^2$. This phenomena is more easily visible for the Rosenbrock+Sine case but also occurs in the plain Rosenbrock case. The ICNN model has a unique global minimum but the fitting error is large as its sub-level sets are convex. For i-DenseNet, the sub-level sets become mildly non-convex but their bi-Lipschitz bound is quite conservative, so they do not capture the overall shape. In contrast, our proposed BiLipNet is more flexible and captures the non-convex shape while maintaining a unique global minimum. We note that in the Rosenbrock+Sine case, the BiLipNet surrogate has errors of similar magnitude to the MLP, but remains "easily optimizable", i.e. it satisfies the PL condition and has a unique global minimum. Additional results are in Appendix D.2.

Partial PLNet. We also fit a parameterized Rosenbrock function $r(x, y; p)$ using partial PLNet with p-dependent biases (see Remark 5.3). The results in Figure 8 indicate that the approach can be effective even if only bias terms are modified by the external parameter p, and not weights.

High-dimensional case. We now turn to scalability of the approach to higher-dimensional problems and analyse convergence of the DYS method for computing the global minimum. We apply the approach to a N=20-dimensional version of the Rosenbrock function:

$$R(x) = \frac{1}{N-1} \sum_{i=1}^{N-1} r(x_i, x_{i+1}) \tag{14}$$

which has a global minimum of zero at $x = (1, 1, ..., 1)$ but is non-convex and has spurious local minima (Kok & Sandrock, 2009). We sample 10K training points uniformly over $[-2, 2]^{20}$. Note that, in contrast to the 2D example above, this is very sparse sampling of 20-dimensional space.

A comparison of train and test error vs model distortion is shown in Figure 5. It can be seen that our proposed BiLipNet model achieves far better fits than iResNet (Behrmann et al., 2021) and iDenseNet (Perugachi-Diaz et al., 2021), which can not achieve small training error for any value of the distortion parameter. Furthermore, for our network, the distortion parameter appears to act as an effective regularizer. Note that the best test error occurs after training error drops to near zero ($\sim 10^{-8}$) but distortion is still relatively small.

Solver comparison. Given the surrogate loss function learned by BiLipNet, we now compare methods to compute the location of its global minimum. In Figure 6 we compare the proposed DYS solver to the forward step method (**FSM**), see, e.g., (Ryu & Boyd, 2016). Specifically, the inverse $x = \mathcal{F}^{-1}(y)$ with $\mathcal{F}$ as a μ-strongly monotone and ν-Lipschitz

Method	c	Accuracy (↑) Clean	Accuracy (↑) Corrupted	ECE (↓) Clean	ECE (↓) Corrupted	NLL (↓) Clean	NLL (↓) Corrupted
CIFAR-10							
SNGP	0.95	76.7 ± 0.629	58.7 ± 1.000	0.057 ± 0.007	0.079 ± 0.006	0.682 ± 0.015	1.199 ± 0.041
	0.65	72.5 ± 1.500	54.7 ± 1.778	0.058 ± 0.006	0.078 ± 0.006	0.797 ± 0.046	1.303 ± 0.057
	0.35	62.7 ± 0.334	52.3 ± 0.721	0.069 ± 0.010	0.065 ± 0.006	1.055 ± 0.010	1.356 ± 0.018
BiLipNet	0.95	86.2 ± 0.250	70.8 ± 0.469	0.020 ± 0.003	0.052 ± 0.005	0.423 ± 0.006	0.895 ± 0.020
	0.65	86.7 ± 0.129	72.8 ± 0.592	0.015 ± 0.005	0.047 ± 0.009	0.400 ± 0.006	0.830 ± 0.024
	0.35	84.5 ± 0.184	72.6 ± 0.216	0.010 ± 0.002	0.052 ± 0.004	0.457 ± 0.002	0.827 ± 0.008
CIFAR-100							
SNGP	0.95	36.9 ± 1.656	25.5 ± 1.406	0.131 ± 0.010	0.068 ± 0.005	2.493 ± 0.068	3.073 ± 0.069
	0.65	33.0 ± 0.481	24.3 ± 0.749	0.117 ± 0.006	0.068 ± 0.003	2.683 ± 0.015	3.140 ± 0.048
	0.35	26.5 ± 1.630	19.3 ± 1.296	0.101 ± 0.016	0.056 ± 0.010	3.020 ± 0.062	3.406 ± 0.073
BiLipNet	0.95	51.0 ± 0.480	35.8 ± 0.397	0.230 ± 0.006	0.137 ± 0.007	2.064 ± 0.024	2.718 ± 0.014
	0.65	55.2 ± 0.426	39.2 ± 0.495	0.225 ± 0.004	0.137 ± 0.005	1.887 ± 0.021	2.576 ± 0.022
	0.35	54.4 ± 0.438	41.1 ± 0.200	0.194 ± 0.008	0.126 ± 0.009	1.876 ± 0.031	2.447 ± 0.016

Table 1. Results for SNGP and BiLipNet on CIFAR-10/100, averaged over 5 seeds. To ensure bi-Lipschitz bounds, batch normalization is removed from SNGP. BiLipNet uniformly significantly outperforms SNGP in term of accuracy on both clean and corrupted data.

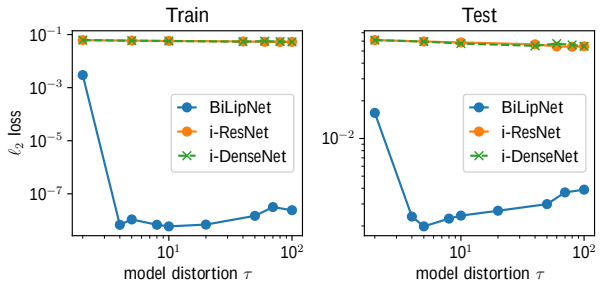


Figure 5. Surrogate loss learning for 20-dimensional Rosenbrock function. Comparison of training and test error vs model distortion for PLNet with different bi-Lipschitz models.

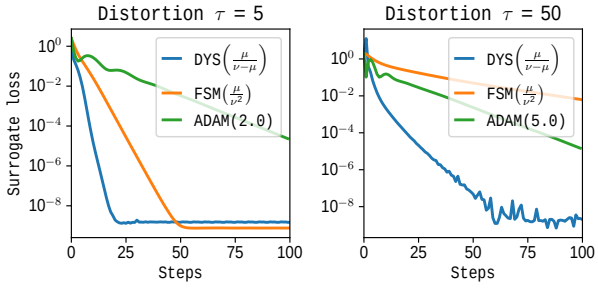


Figure 6. Solver comparison for finding the global minimum of a PLNet. We try a range of rates $[0.1, 0.5, 1.0, 2.0, 5.0]$ for ADAM and present the best result. The proposed back solve method with DYS algorithm (11) converges much faster than ADAM applied to f or back solve method with FSM algorithm (15).

layer can be computed via

$$x^{k+1} = x^k - \alpha(\mathcal{F}(x^k) - y) \tag{15}$$

which has a convergence rate of $1 - \mu^2/\nu^2$ if $\alpha = \mu/\nu^2$. We also consider a commonly used gradient-based method – ADAM (Kingma & Ba, 2015) applied directly to the surrogate loss. We take two values of the distortion parameter: $\tau = 5$ (optimal) and $\tau = 50$. In both cases, the proposed DYS method converges significantly faster than the alternatives, and the results illustrate an additional benefit of regularising via distortion, besides improving the test error: the $\tau = 5$ case converges significantly faster than $\tau = 50$.

At the computed point $x^? = \mathcal{G}^{-1}(0)$ for $\tau = 5$, the true function (14) takes a value of $R(x^?) = 0.041$. This is more than an order of magnitude better than the smallest value of $R(x)$ over the training data, which ranged over $[0.475, 6.532]$, indicating that PLNets have a useful "implicit bias" and do not simply interpolate the training data.

7. Conclusion

This paper has introduced a new bi-Lipschitz network architecture, the BiLipNet, and a new scalar-output network, the PLNet which satisfies the Polyak-Łojasiewicz condition, and is hence "easily optimizable".

The core technical contribution is a new layer-type: the "feed-through" layer, which has certified bounds for strong monotonicity and Lipschitzness. By composing with orthogonal layers we obtain a bi-Lipschitz network structure (BiLipNet) which has much tighter bounds than existing bi-Lipschitz residual networks based on spectral normalization. The PLNet composes a BiLipNet with a quadratic output layer, and guarantees unique global minimum which is efficiently computable.

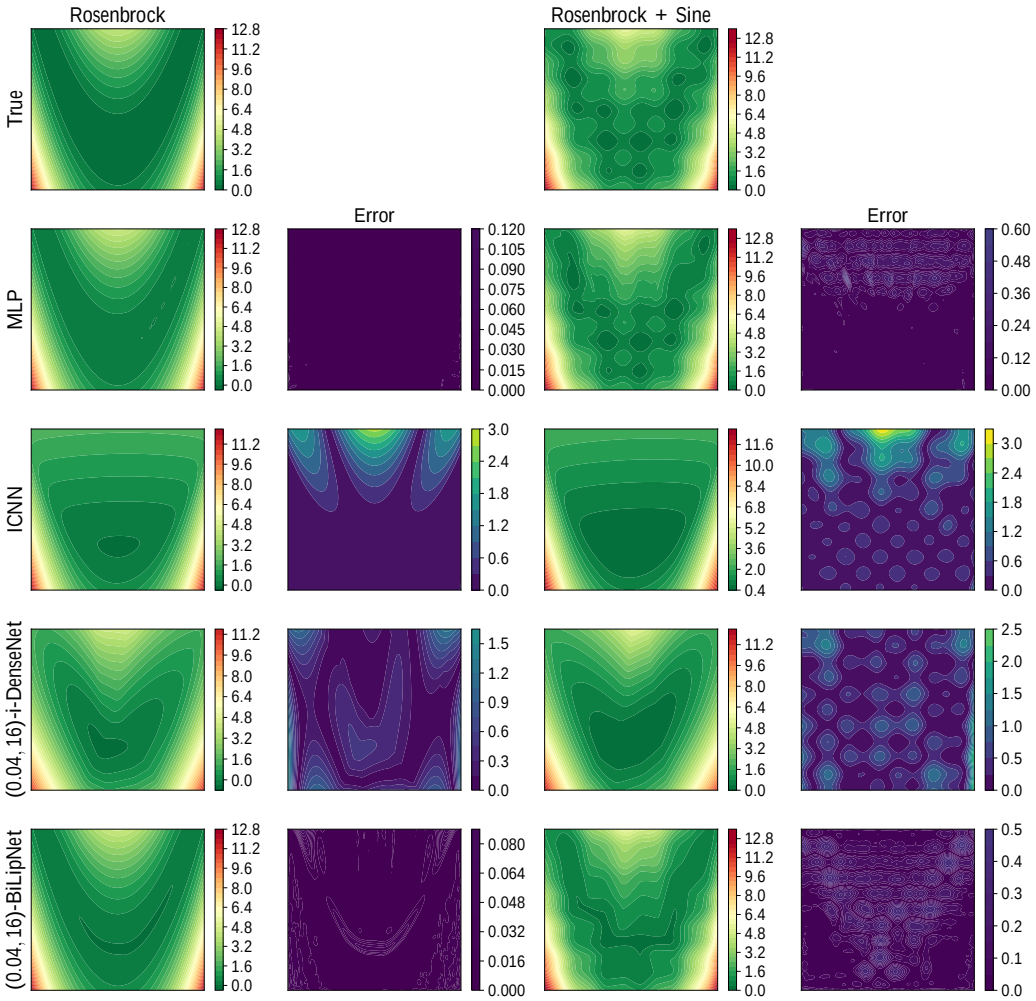


Figure 7. Learning a surrogate loss for the Rosenbrock and Rosenbrock+Sine functions, which is non-convex and has many local minima. The first row contains the true functions while the remaining rows show learned functions and errors for various surrogate loss models.

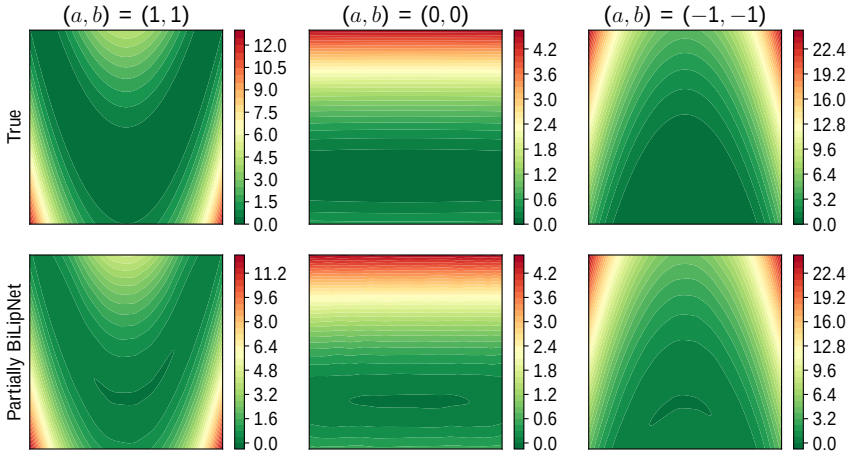


Figure 8. Learning a parameterized Rosenbrock function $r(x, y; a, b) = 1/200(x \quad a)^2 + 0.5(y \quad bx^2)^2$ via a partial PLNet.

Impact Statement

There are many application domains in which the trustworthiness of machine learning is a live topic of debate and raises important and challenging questions. The goal of this paper is to advance the sub-field of machine learning methods which have mathematically-certified properties. In particular, in this paper one application is uncertainty quantification. We hope that a positive impact of our paper and others like it will be to the development of ML methods that can better satisfy societal expectations of trustworthiness and transparency.

We are not aware of any potentially significant negative impacts that are particularly associated with this line of research (models with certified properties).

References

Ahn, B., Kim, C., Hong, Y., and Kim, H. J. Invertible monotone operators for normalizing flows. *Advances in Neural Information Processing Systems*, 35:16836–16848, 2022.

Amos, B., Xu, L., and Kolter, J. Z. Input convex neural networks. In *International Conference on Machine Learning (ICML)*, pp. 146–155. PMLR, 2017.

Araujo, A., Havens, A. J., Delattre, B., Allauzen, A., and Hu, B. A unified algebraic perspective on Lipschitz neural networks. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023.

Ardizzone, L., Kruse, J., Rother, C., and Köthe, U. Analyzing inverse problems with invertible neural networks. In *International Conference on Learning Representations (ICLR)*, 2018.

Arjovsky, M., Chintala, S., and Bottou, L. Wasserstein generative adversarial networks. In *International conference on machine learning (ICML)*, pp. 214–223. PMLR, 2017.

Arora, S. and Doshi, P. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, 297:103500, 2021.

Bai, S., Kolter, J. Z., and Koltun, V. Deep equilibrium models. In *Advances in Neural Information Processing Systems*, pp. 690–701, 2019.

Barbara, N. H., Wang, R., and Manchester, I. R. On robust reinforcement learning with lipschitz-bounded policy networks. *arXiv preprint arXiv:2405.11432*, 2024.

Bauer, M. and Mnih, A. Resampled priors for variational autoencoders. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 66–75. PMLR, 2019.

Behrmann, J., Grathwohl, W., Chen, R. T., Duvenaud, D., and Jacobsen, J.-H. Invertible residual networks. In *International conference on machine learning (ICML)*, pp. 573–582. PMLR, 2019.

Behrmann, J., Vicol, P., Wang, K.-C., Grosse, R., and Jacobsen, J.-H. Understanding and mitigating exploding inverses in invertible neural networks. In *International Conference on Artificial Intelligence and Statistics*, pp. 1792–1800. PMLR, 2021.

Chen, R. T., Behrmann, J., Duvenaud, D. K., and Jacobsen, J.-H. Residual flows for invertible generative modeling. *Advances in Neural Information Processing Systems*, 32, 2019.

Cheng, J., Wang, R., and Manchester, I. R. Learning stable and passive neural differential equations. *arXiv preprint arXiv:2404.12554*, 2024.

Coleman, C., Narayanan, D., Kang, D., Zhao, T., Zhang, J., Nardi, L., Bailis, P., Olukotun, K., Ré, C., and Zaharia, M. Dawnbench: An end-to-end deep learning benchmark and competition. *Training*, 100(101):102, 2017.

Cozad, A., Sahinidis, N. V., and Miller, D. C. Learning surrogate models for simulation-based optimization. *AIChE Journal*, 60(6):2211–2227, 2014.

Davis, D. and Yin, W. A three-operator splitting scheme and its optimization applications. *Set-valued and variational analysis*, 25:829–858, 2017.

Davis, T. A. *Direct methods for sparse linear systems*. SIAM, 2006.

De Cao, N., Aziz, W., and Titov, I. Block neural autoregressive flow. In *Uncertainty in artificial intelligence*, pp. 1263–1273. PMLR, 2020.

Dinh, L., Krueger, D., and Bengio, Y. Nice: Non-linear independent components estimation. *ICLR Workshop Track*, 2015.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using real NVP. In *International Conference on Learning Representations (ICLR)*, 2017.

El Ghaoui, L., Gu, F., Travacca, B., Askari, A., and Tsai, A. Implicit deep learning. *SIAM Journal on Mathematics of Data Science*, 3(3):930–958, 2021.

Fazlyab, M., Robey, A., Hassani, H., Morari, M., and Pappas, G. Efficient and accurate estimation of Lipschitz constants for deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 11427–11438, 2019.

Grathwohl, W., Chen, R. T., Bettencourt, J., and Duvenaud, D. Scalable reversible generative models with free-form continuous dynamics. In *International Conference on Learning Representations (ICLR)*, 2019.

Grudzien, K., Uehara, M., Levine, S., and Abbeel, P. Functional graphical models: Structure enables offline data-driven optimization. In *International Conference on Artificial Intelligence and Statistics*, pp. 2449–2457. PMLR, 2024.

Gu, S., Lillicrap, T., Sutskever, I., and Levine, S. Continuous deep Q-learning with model-based acceleration. In *International conference on machine learning (ICML)*, pp. 2829–2838. PMLR, 2016.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. Improved training of Wasserstein GANs. *Advances in neural information processing systems*, 30, 2017.

Havens, A. J., Araujo, A., Garg, S., Khorrami, F., and Hu, B. Exploiting connections between Lipschitz structures for certifiably robust deep equilibrium models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Helfrich, K., Willmott, D., and Ye, Q. Orthogonal recurrent neural networks with scaled Cayley transform. In *International Conference on Machine Learning (ICML)*, pp. 1969–1978. PMLR, 2018.

Ho, J., Chen, X., Srinivas, A., Duan, Y., and Abbeel, P. Flow++: Improving flow-based generative models with variational dequantization and architecture design. In *International Conference on Machine Learning (ICML)*, pp. 2722–2730. PMLR, 2019.

Huang, C.-W., Krueger, D., Lacoste, A., and Courville, A. Neural autoregressive flows. In *International Conference on Machine Learning (ICML)*, pp. 2078–2087. PMLR, 2018.

Karimi, H., Nutini, J., and Schmidt, M. Linear convergence of gradient and proximal-gradient methods under the Polyak-Łojasiewicz condition. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2016, Riva del Garda, Italy, September 19-23, 2016, Proceedings, Part I 16*, pp. 795–811. Springer, 2016.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Kingma, D. P. and Dhariwal, P. Glow: Generative flow with invertible 1x1 convolutions. *Advances in neural information processing systems*, 31, 2018.

Kobyzev, I., Prince, S. J., and Brubaker, M. A. Normalizing flows: An introduction and review of current methods. *IEEE transactions on pattern analysis and machine intelligence*, 43(11):3964–3979, 2020.

Kok, S. and Sandrock, C. Locating and Characterizing the Stationary Points of the Extended Rosenbrock Function. *Evolutionary Computation*, 17(3):437–453, 09 2009.

Li, J., Fang, C., and Lin, Z. Lifted proximal operator machines. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4181–4188, 2019.

Li, J., Li, F., and Todorovic, S. Efficient riemannian optimization on the stiefel manifold via the Cayley transform. In *International Conference on Learning Representations (ICLR)*, 2020.

Liang, E., Chen, M., and Low, S. Low complexity homeomorphic projection to ensure neural-network solution feasibility for optimization over (non-) convex set. In *International conference on machine learning (ICML)*. PMLR, 2023.

Liu, J., Lin, Z., Padhy, S., Tran, D., Bedrax Weiss, T., and Lakshminarayanan, B. Simple and principled uncertainty estimation with deterministic deep learning via distance awareness. *Advances in Neural Information Processing Systems*, 33:7498–7512, 2020.

Liu, J. Z., Padhy, S., Ren, J., Lin, Z., Wen, Y., Jerfel, G., Nado, Z., Snoek, J., Tran, D., and Lakshminarayanan, B. A simple approach to improve single-model deep uncertainty via distance-awareness. *Journal of Machine Learning Research*, 24(42):1–63, 2023.

Lojasiewicz, S. A topological property of real analytic subsets. *Coll. du CNRS, Les équations aux dérivées partielles*, 117(87-89):2, 1963.

Louizos, C. and Welling, M. Multiplicative normalizing flows for variational bayesian neural networks. In *International Conference on Machine Learning (ICML)*, pp. 2218–2227. PMLR, 2017.

Lu, C., Chen, J., Li, C., Wang, Q., and Zhu, J. Implicit normalizing flows. In *International Conference on Learning Representations (ICLR)*, 2021.

Megretski, A. and Rantzer, A. System analysis via integral quadratic constraints. *IEEE Transactions on Automatic Control*, 42(6):819–830, 1997.

Meunier, L., Delattre, B. J., Araujo, A., and Allauzen, A. A dynamical system perspective for Lipschitz neural networks. In *International Conference on Machine Learning (ICML)*, pp. 15484–15500. PMLR, 2022.

Misener, R. and Biegler, L. Formulating data-driven surrogate models for process optimization. *Computers & Chemical Engineering*, 179:108411, 2023.

Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2018.

Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., and Lakshminarayanan, B. Normalizing flows for probabilistic modeling and inference. *The Journal of Machine Learning Research*, 22(1):2617–2680, 2021.

Pauli, P., Havens, A., Araujo, A., Garg, S., Khorrami, F., Allgöwer, F., and Hu, B. Novel quadratic constraints for extending LipSDP beyond slope-restricted activations. In *International Conference on Learning Representations (ICLR)*, 2024.

Perugachi-Diaz, Y., Tomczak, J., and Bhulai, S. Invertible densenets with concatenated Lipswish. *Advances in Neural Information Processing Systems*, 34:17246–17257, 2021.

Polyak, B. Gradient methods for minimizing functionals (in russian). *USSR Computational Mathematics and Mathematical Physics*, 3(4):643–653, 1963.

Prach, B. and Lampert, C. H. Almost-orthogonal layers for efficient general-purpose Lipschitz networks. In *European Conference on Computer Vision*, pp. 350–365. Springer, 2022.

Rantzer, A. On the Kalman—Yakubovich—Popov lemma. *Systems & control letters*, 28(1):7–10, 1996.

Revay, M., Wang, R., and Manchester, I. R. Lipschitz bounded equilibrium networks. *arXiv preprint arXiv:2010.01732*, 2020.

Revay, M., Wang, R., and Manchester, I. R. Recurrent equilibrium networks: Flexible dynamic models with guaranteed stability and robustness. *IEEE Transactions on Automatic Control*, 2023.

Russo, A. and Proutiere, A. Towards optimal attacks on reinforcement learning policies. In *2021 American Control Conference (ACC)*, pp. 4561–4567. IEEE, 2021.

Ryu, E. K. and Boyd, S. Primer on monotone operator methods. *Appl. comput. math*, 15(1):3–43, 2016.

Ryu, M., Chow, Y., Anderson, R., Tjandraatmadja, C., and Boutilier, C. CAQL: Continuous action Q-learning. In *International Conference on Learning Representations (ICLR)*, 2019.

Singla, S. and Feizi, S. Skew orthogonal convolutions. In *International Conference on Machine Learning (ICML)*, pp. 9756–9766. PMLR, 2021.

Singla, S., Singla, S., and Feizi, S. Improved deterministic l2 robustness on CIFAR-10 and CIFAR-100. In *International Conference on Learning Representations (ICLR)*, 2022.

Trockman, A. and Kolter, J. Z. Orthogonalizing convolutional layers with the Cayley transform. In *International Conference on Learning Representations (ICLR)*, 2021.

Tsuzuku, Y., Sato, I., and Sugiyama, M. Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks. In *Advances in neural information processing systems*, pp. 6541–6550, 2018.

Wang, R. and Manchester, I. Direct parameterization of Lipschitz-bounded deep networks. In *International Conference on Machine Learning (ICML)*, pp. 36093–36110. PMLR, 2023.

Wang, Z., Prakriya, G., and Jha, S. A quantitative geometric approach to neural-network smoothness. *Advances in Neural Information Processing Systems*, 35:34201–34215, 2022.

Ward, P. N., Smofsky, A., and Bose, A. J. Improving exploration in soft-actor-critic with normalizing flows policies. *ICML Workshop on Invertible Neural Networks and Normalizing Flows,*, 2019.

Wilson, F. W. The structure of the level surfaces of a Lyapunov function. *Journal of Differential Equations*, 3(3): 323–329, 1967.

Winston, E. and Kolter, J. Z. Monotone operator equilibrium networks. *Advances in neural information processing systems*, 33:10718–10728, 2020.

Yeh, J. *Real analysis: theory of measure and integration second edition*. World Scientific Publishing Company, 2006.

Zhang, B., Cai, T., Lu, Z., He, D., and Wang, L. Towards certifying l-infinity robustness using neural networks with l-inf-dist neurons. In *International Conference on Machine Learning*, pp. 12368–12379. PMLR, 2021.

A. Model Parameterization

A model parameterization is a mapping $\mathcal{M} : \phi \to \theta$ where $\phi \in \mathsf{R}^N$ is a free learnable parameter while θ includes the model weights $U \in \mathsf{R}^{m\ \ n}, W \in \mathsf{R}^{m\ \ m}, Y \in \mathsf{R}^{n\ \ m}$ and IQC multiplier $\Lambda \in \mathsf{D}_+^m$ with n, m as the dimensions of the input and hidden units, respectively. The aim of this section is to construct a parameterization such that the large-scale SDP constraint (3) holds, i.e., $Y = U^\top \Lambda$ and

$$H = 2\Lambda - W^\top \Lambda - \Lambda W = \begin{bmatrix} 2\Lambda_1 & -W_2^\top \Lambda_2 & & & \\ -\Lambda_2 W_2 & 2\Lambda_2 & -W_3^\top \Lambda_3 & & \\ & \ddots & \ddots & \ddots & \\ & & -\Lambda_{L\ 1} W_{L\ 1} & 2\Lambda_{L\ 1} & -W_L^\top \Lambda_L \\ & & & -\Lambda_L W_L & 2\Lambda_L \end{bmatrix} \geq \frac{2}{\gamma} Y^\top Y \tag{16}$$

Since $H \succeq 0$ has band structure, it can be represented by $H = XX^\top$ (Davis, 2006). Moreover, from Lemma 3 of (Rantzer, 1996) we have that any U, Y satisfying $Y = U^\top \Lambda$ and $XX^\top \succeq {}^2Y^\top Y$ can be represent by

$$U = \sqrt{\gamma/2}\Lambda\ {}^1XQ, \quad Y = \sqrt{\gamma/2}Q^\top X^\top \tag{17}$$

where $Q \in \mathsf{R}^{m\ \ n}$ with $QQ^\top \preceq I$. The remaining task is to find X such that $H = XX^\top$ has the same sparse structure as (16), which was solved by (Wang & Manchester, 2023). For self-contained purpose, we provide detail construction as follows. First, we further parameterize $X = \Psi P$, where $\Psi = \mathrm{diag}(\Psi_1, \ldots, \Psi_L)$ with $\Psi_k \in \mathsf{D}_+^{m_k}$ and

$$P = \begin{bmatrix} A_1 & & & \\ -B_2 & A_2 & & \\ & \ddots & \ddots & \\ & & -B_L & A_L \end{bmatrix}.$$

By comparing $H = \Psi P P^\top \Psi$ with (16) we have

$$H_{kk} = \Psi_k(B_k B_k^\top + A_k A_k^\top)\Psi_k = 2\Lambda_k, \quad H_{k\ 1;k} = -\Psi_k B_k A_{k\ 1}^\top = -\Lambda_k W_k,$$

which further leads to

$$\Psi_k^2 = 2\Lambda_k, \quad B_k B_k^\top + A_k A_k^\top = I, \quad W_k = 2\Psi_{k\ }{}^1B_k A_{k\ 1}^\top \Psi_{k\ 1} \quad k = 1, \ldots, L, \tag{18}$$

with $B_1 = 0$. We have converted the large-scale SDP constraint (16) into many simple and small-scale constraints such as

$$\Psi_k^2 = 2\Lambda_k, \quad R_k R_k^\top = I, \quad QQ^\top \preceq I \tag{19}$$

with $R_k = \begin{bmatrix} B_k & A_k \end{bmatrix}$, which further can be easily parameterized via the Cayley transformation (4), see Section 3.3. The Cayley transformation has been applied to construct orthogonal layers (Helfrich et al., 2018; Li et al., 2020; Trockman & Kolter, 2021) and 1-Lipschitz Sandwich layer (Wang & Manchester, 2023).

An equivalent model representation. The model weights U, Y, W defined in (5) can be rewritten as $U = \sqrt{2\gamma}\Psi\ {}^1S$, $Y = \sqrt{\gamma/2}S^\top \Psi\ {}^1$ and $W = \Psi\ {}^1W\Psi$ with

$$S = \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_L \end{bmatrix} = \begin{bmatrix} A_1 Q_1 \\ A_2 Q_2 - B_2 Q_1 \\ \vdots \\ A_L Q_L - B_L Q_{L\ 1} \end{bmatrix}, \quad V = \begin{bmatrix} 0 & & & \\ V_2 & 0 & & \\ & \ddots & \ddots & \\ & & V_L & 0 \end{bmatrix} = \begin{bmatrix} 0 & & & \\ 2B_2 A_1^\top & 0 & & \\ & \ddots & \ddots & \\ & & 2B_L A_{L\ 1}^\top & 0 \end{bmatrix} \tag{20}$$

where $Q = \begin{bmatrix} Q_1^\top & \cdots & Q_L^\top \end{bmatrix}^\top$. Then, the network (2) can be written as

$$z = \sigma(\Psi\ {}^1V\Psi z + \sqrt{2\gamma}\Psi\ {}^1Sx + b), \quad y = \mu x + \sqrt{\gamma/2}S^\top \Psi z. \tag{21}$$

By introducing the new hidden state $\hat{z} = \Psi z$ and bias $\hat{b} = \Psi b$, we obtain an equivalent form:

$$\hat{z} = \hat{\sigma}\big(V\hat{z} + \sqrt{2\gamma}Sx + \hat{b}\big), \quad y = \mu x + \sqrt{\gamma/2}S^\top \hat{z} + b_y. \tag{22}$$

This representation is useful for computing the model inverse via monotone operator splitting, see Appendix B. We now give a lemma which will be used later for proving some propositions.

Lemma A.1. *For the matrices V, S defined in (20) we have*

$$2I - V - V^\top \succeq 0, \quad 2I - SS^\top \succeq 0. \tag{23}$$

Proof. First, we have

$$2I - (V + V^\top) = 2\begin{bmatrix} I & -A_1B_2^\top & & \\ -B_2A_1^\top & I & -A_2B_3^\top & \\ & -B_3^\top A_2 & \ddots & \ddots \\ & & \ddots & \ddots \end{bmatrix} \succeq 0$$

where the inequality is obtained by sequentially applying the fact $A_kA_k^\top + B_kB_k^\top = I$ and Schur complement to the top diagonal block. For the inequality on S, we have

$$\begin{aligned} 2I - SS^\top &= 2I - PQQ^\top P^\top \succeq 2I - PP^\top \\ &= 2I - \begin{bmatrix} A_1 & & & \\ -B_2 & A_2 & & \\ & \ddots & \ddots & \\ & & -B_L & A_L \end{bmatrix}\begin{bmatrix} A_1 & & & \\ -B_2 & A_2 & & \\ & \ddots & \ddots & \\ & & -B_L & A_L \end{bmatrix}^\top = \begin{bmatrix} I & A_1B_2^\top & & \\ B_2A_1^\top & I & A_2B_3^\top & \\ & B_3^\top A_2 & \ddots & \ddots \\ & & \ddots & \ddots \end{bmatrix} \succeq 0. \end{aligned}$$

Similarly, the last inequality can be established by sequentially applying the Schur complement to the top diagonal block. □

B. Monotone Operator Splitting for Computing Model Inverse

Inspired by (Winston & Kolter, 2020; Revay et al., 2020), we try to compute $x = \mathcal{F}^{-1}(y)$ via an operator splitting method. We first present some background of monotone operator theory based on the survey (Ryu & Boyd, 2016), and then reformulate the model inverse as a three-operator splitting problem.

B.1. Monotone operator

An *operator* is a set-valued or single-valued map defined by a subset of the space $\mathcal{A} \subseteq \mathsf{R}^n \times \mathsf{R}^n$; we use the notation $\mathcal{A}(x) = \{y \mid (x, y) \in \mathcal{A}\}$. For example, the affine operator is defined by $\mathcal{L}(x) = \{(x, Wx + b) \mid x \in \mathsf{R}^n\}$. Another important example is the subdifferential operator $\partial f = \{(x, \partial f(x))\}$ for a proper function $f : \mathsf{R}^n \to \mathsf{R} \cup \{\infty\}$ with $f(z) = \infty$ for $z \notin \mathbf{dom}\, f$, where $\partial f(x) = \{g \in \mathsf{R}^n \mid f(y) \geq f(x) + \langle y - x, g\rangle, \forall y \in \mathsf{R}^n\}$. An operator $\mathcal{A}$ has a Lipschitz bound of L if $\|u - v\| \leq L\|x - y\|$ for all $(x, u), (y, v) \in \mathcal{A}$. It is *non-expansive* if $L = 1$ and *contractive* if $L < 1$. $\mathcal{A}$ is *strongly monotone* with $m > 0$ if

$$\langle u - v, x - y\rangle \geq m\|x - y\|, \quad \forall (x, u), (y, v) \in \mathcal{A}. \tag{24}$$

If the above inequality holds for $m = 0$, we call $\mathcal{A}$ a monotone operator. Similarly, $\mathcal{A}$ is said to be *inverse monotone* with ρ if $\langle u - v, x - y\rangle \geq \rho\|u - v\|$, $\forall (x, u), (y, v) \in \mathcal{A}$. An operator is called *maximal monotone* if no other monotone operator strictly contains it. The linear operator $\mathcal{L}$ is m-strongly monotone if $W + W^\top \succeq 2mI$, and ρ-inverse monotone if $W + W^\top \succeq 2\rho W^\top W$. A subdifferential ∂f is maximal monotone if and only if f is a convex closed proper (CCP) function. Here are some basic operations for operators:

- the operator sum $\mathcal{A} + \mathcal{B} = \{(x, y + z) \mid (x, y) \in \mathcal{A},\ (x, z) \in \mathcal{B}\}$;
- the composition $\mathcal{A}\mathcal{B} = \{(x, z) \mid \exists y \text{ s.t. } (x, y) \in \mathcal{A}, (y, z) \in \mathcal{B}\}$;

Table 2. A list of common activation functions and their associated convex proper $f(z)$ whose proximal operator is $\sigma(x)$ (Revay et al., 2020). For $z \notin \mathbf{dom}\, f$, we have $f(z) = \infty$. In the case of Softplus activation, $\mathrm{Li}_s(z)$ is the polylogarithm function.

Activation	$\sigma(x)$	Convex $f(z)$	$\mathbf{dom}\, f$
ReLu	$\max(x, 0)$	0	$[0, \infty)$
LeakyReLu	$\max(x, 0.01x)$	$\frac{99}{2}\min(z, 0)^2$	$\mathbb{R}$
Tanh	$\tanh(x)$	$\frac{1}{2}\left[\ln(1-z^2) + z\ln\left(\frac{1+z}{1-z}\right) - z^2\right]$	$(-1, 1)$
Sigmoid	$1/(1+e^{-x})$	$z\ln z + (1-z)\ln(1-z) - \frac{z^2}{2}$	$(0, 1)$
Arctan	$\arctan(x)$	$-\ln(\lvert\cos z\rvert) - \frac{z^2}{2}$	$(-1, 1)$
Softplus	$\ln(1+e^x)$	$-\mathrm{Li}_2(e^z) - i\pi z - z^2/2$	$(0, \infty)$

- the inverse operator $\mathcal{A}^{-1} = \{(y, x) \mid (x, y) \in \mathcal{A}\}$;
- the *resolvent* operator $R_{\mathcal{A}} = (I + \alpha\mathcal{A})^{-1}$ with $\alpha > 0$;
- the *Cayley* operator $C_{\mathcal{A}} = 2R_{\mathcal{A}} - I$.

Note that the resolvent and Cayley operators are non-expansive for any maximal monotone $\mathcal{A}$, and are contractive if $\mathcal{A}$ is strongly monotone. For a linear operator $\mathcal{L}$ we have $R_{\mathcal{L}}(x) = (I + \alpha W)^{-1}(x - \alpha b)$. For a subdifferential operator ∂f, its resolvent is $R_{\partial f}(x) = \mathbf{prox}_f(x) := \arg\min_z 1/2\|x - z\| + \alpha f(z)$, which is also called the *proximal operator*.

Activation as proximal operator. As shown in (Li et al., 2019; Revay et al., 2020), many popular slope-restricted scalar activation functions can also be treated as proximal operators. To be specific, if $\sigma : \mathbb{R} \to \mathbb{R}$ is slope-restricted in $[0, 1]$, then there exists a convex proper function f such that $\sigma(\cdot) = \mathbf{prox}_f^1(\cdot)$. For self-contained purpose, we provide a list of common activations and their associated convex proper functions in Table 2, which can also be found in (Revay et al., 2020; Li et al., 2019).

B.2. Operator splitting

Many optimization problems (e.g. convex optimization) can be formulated as one of finding a zero of an appropriate monotone operator $\mathcal{F}$, i.e., find $x \in \mathbb{R}^n$ such that $0 \in \mathcal{F}(x)$. Note that x is a solution if and only if it is a fixed point $x = \mathcal{T}(x)$ with $\mathcal{T} = I - \alpha\mathcal{F}$ for any nonzero $\alpha \in \mathbb{R}$. The corresponding fixed point iteration is $x^{k+1} = \mathcal{T}(x^k) = x^k - \alpha\mathcal{F}(x^k)$. If $\mathcal{F}$ is m-strongly monotone and L-Lipschitz, then this iteration converges by choosing $\alpha \in (0, 2m/L^2)$. The optimal convergence rate is $1 - (m/L)^2$, given by $\alpha = m/L^2$.

If $\mathcal{F}$ contains some non-smooth components, we then split $\mathcal{F}$ into two or three maximal operators:

$$\text{two-operator splitting problem:} \quad 0 \in \mathcal{A}(x) + \mathcal{B}(x) \tag{25}$$

$$\text{three-operator splitting problem:} \quad 0 \in \mathcal{A}(x) + \mathcal{B}(x) + \mathcal{C}(x) \tag{26}$$

where $\mathcal{A}, \mathcal{B}$, and $\mathcal{C}$ are maximal monotone. The main benefit of such splitting is that the resolvent or Cayley operators for individual operator are easy to evaluate, which further leads to more computationally efficient algorithms. For two-operator splitting problem, some popular algorithms include

- forward-backward splitting (FBS) $x = R_{\mathcal{B}}(I - \alpha\mathcal{A})(x)$

- forward-backward-forward splitting (FBFS) $x = ((I - \alpha\mathcal{A})R_B(I - \alpha\mathcal{A}) + \alpha\mathcal{A})(x)$
- Peaceman-Rachford splitting (PRS) $z = C_A C_B(z),\ x = R_B(z)$
- Douglas-Rachford splitting (DRS) $z = (1/2I + 1/2C_A C_B)(z),\ x = R_B(z)$

where the corresponding fixed-point iterations, the choices of hyper-parameter α and convergence results can be found in (Ryu & Boyd, 2016). For three-operator splitting problem, the Davis-Yin splitting (DYS) (Davis & Yin, 2017) can be expressed by $z = \mathcal{T}(z),\ x = R_B(z)$ where $\mathcal{T} = C_A(C_B - \alpha\mathcal{C}R_B) - \alpha\mathcal{C}R_B$.

B.3. Operator splitting perspective for $\mathcal{F}$ [1]

As shown in the proof of Proposition 4.2, By applying the forward-backward splitting with parameter $\alpha = 1$, we can compute the solution z via the following iteration:

$$z^{k+1} = R_B(z^k - \widehat{\mathcal{A}}(z^k)) = \hat{\sigma}\left(\left(V - \gamma/\mu SS^\top\right)z^k + b_z\right).$$

It is worth pointing out that the above iteration may not converge for the choice of $\alpha = 1$. In practice we often use more stable and faster two-operator splitting algorithms (e.g., PRS or DRS), see (Winston & Kolter, 2020; Revay et al., 2020). In this work, the motivation for further decomposing the monotone operator $\widehat{\mathcal{A}}$ into two monotone operators $\mathcal{A}, \mathcal{C}$ is that R_A is a large-scale linear equation with nice sparse structure while $R_{\widehat{A}}$ is dense due to the full weight matrix in $\mathcal{C}$.

Fixed-point iteration. We now apply the DYS algorithm from (Davis & Yin, 2017) to $0 \in \mathcal{A}(z) + \mathcal{B}(z) + \mathcal{C}(z)$, resulting in the following fixed-point iteration:

$$\begin{aligned} z^{k+1=2} &= R_B(u^k) = \mathbf{prox}_f(u^k) \\ u^{k+1=2} &= 2z^{k+1=2} - u^k \\ z^{k+1} &= R_A(u^{k+1=2} - \alpha\mathcal{C}(z^{k+1=2})) \\ u^{k+1} &= u^k + z^{k+1} - z^{k+1=2} \end{aligned} \tag{27}$$

where the third line is a large-scale sparse linear equation of the form

$$\begin{bmatrix} (1+\alpha)I & & & \\ -\alpha V_{21} & (1+\alpha)I & & \\ & \ddots & \ddots & \\ & & -\alpha V_{L;L\ 1} & (1+\alpha)I \end{bmatrix} \begin{bmatrix} z_1^{k+1} \\ z_2^{k+1} \\ \vdots \\ z_L^{k+1} \end{bmatrix} = u^{k+1=2} + \alpha\left(b_z - \frac{\gamma}{\mu}SS^\top z^{k+1=2}\right).$$

By introducing $v^{k+1=2} = b_z - \gamma/\mu SS^\top z^{k+1=2}$, we have

$$z_0^{k+1} = 0, \quad z_l^{k+1} = \frac{\alpha}{1+\alpha}\left(V_{l;l\ 1}z_{l\ 1}^{k+1} + v_l^{k+1=2}\right) + \frac{1}{1+\alpha}u_l^{k+1=2}, \quad l = 1, \dots, L. \tag{28}$$

Convergence range for the hyper-parameter α. From the previous paragraph, we know that (11) is equivalent to the FPI (27). From Theorem 1.1 of (Davis & Yin, 2017), we have that (27) converges for any $\alpha \in (0, 2\beta)$ with β as the inverse-monotone bound of $\mathcal{C}$. From Lemma A.1 we have $2I \succeq S^\top S$ and

$$\frac{2\gamma}{\mu}SS^\top \succeq \frac{\gamma}{\mu}S(S^\top S)S^\top = \frac{\mu}{\gamma}(\gamma/\mu SS^\top)^2 = 2\beta(\gamma/\mu SS^\top)^2$$

i.e., $\mathcal{C}(z)$ is inverse monotone with $\beta = \mu/(2\gamma)$. Therefore, (11) converges for any $\alpha \in (0, \mu/\gamma)$. Since $\gamma = \nu - \mu$ and $\tau = \nu/\mu$, we then obtain the convergence range in term of model distortion τ, i.e., $\alpha \in (0, 1/(\tau - 1))$. Larger α often implies faster convergence rate, see Figure 9.

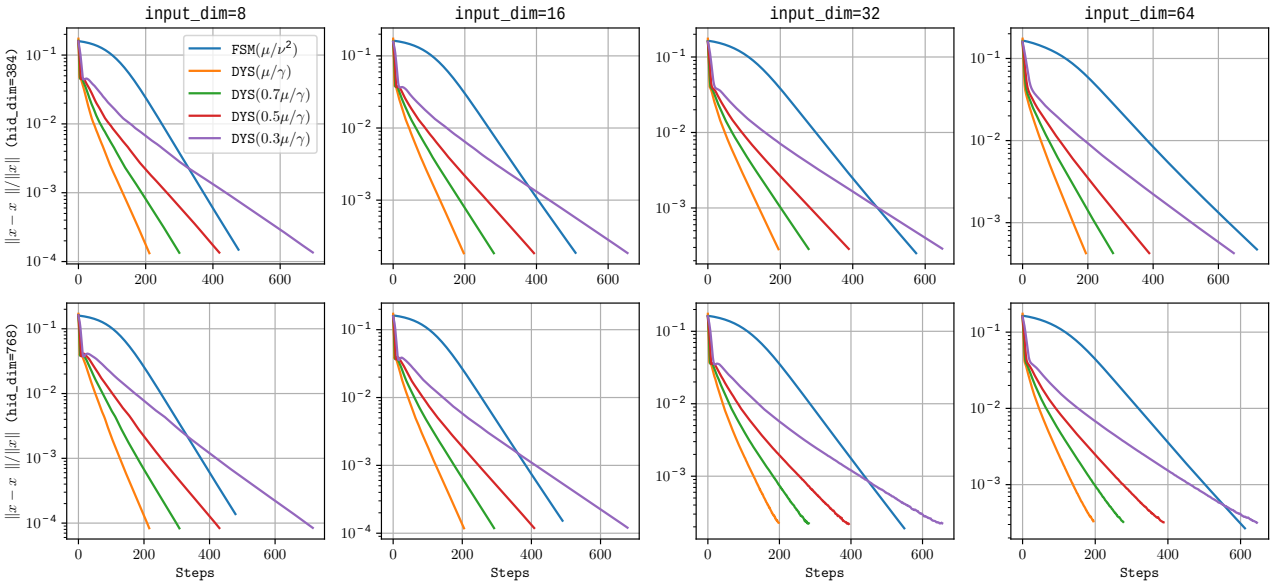


Figure 9. Solver comparison for computing the inverse of random μ-monotone and ν-Lipschitz layers with different input and hidden unit dimensions. We obverse that DYS (27) converges faster for lager α. If α is close to the bound μ/γ with $\gamma = \nu \quad \mu$, DYS converges much faster rate than FSM (15) with hyper-parameter $\alpha = \mu/\nu^2$, which achieves its best convergence rate (Ryu & Boyd, 2016).

C. Proofs

C.1. Proof of Theorem 3.2

We consider the neural network $\mathcal{H} : x \to \tilde{y}$ defined by

$$v = Wz + Ux + b, \quad z = \sigma(v), \quad \tilde{y} = Yz + b_y. \tag{29}$$

Since $\mathcal{F}(x) = \mu x + \mathcal{H}(x)$, then $\mathcal{F}$ is μ-strongly monotone and ν-Lipschitz if $\mathcal{H}$ is monotone and γ-Lipschitz with $\gamma = \nu - \mu$.

For any pair of solutions $s_1 = (x_1, v_1, z_1, \tilde{y}_1)$ and $s_2 = (x_2, v_2, z_2, \tilde{y}_2)$, their difference $\Delta s = s_1 - s_2$ satisfies

$$\Delta v = W\Delta z + U\Delta x, \quad \Delta z = J\ (v_1, v_2)\Delta v, \quad \Delta\tilde{y} = Y\Delta z \tag{30}$$

where J is a diagonal matrix with $[J\ ]_{ii} \in [0, 1]$ since σ is an elementwise activation with slope restricted in $[0, 1]$. For any $\Lambda \in \mathsf{D}_+^m$ we have

$$\langle \Delta v - \Delta z, \Lambda\Delta z\rangle = \Delta v^\top (I - J\ )\Lambda J\ \Delta_v \geq 0, \quad \forall \Delta v \in \mathsf{R}^m. \tag{31}$$

Based on (30), (31) and Condition (3) we have

$$\begin{aligned}
\langle \Delta x, \Delta\tilde{y}\rangle - \langle \Delta v - \Delta z, \Lambda\Delta z\rangle &= \langle \Delta x, Y\Delta z\rangle - \langle (W - I)\Delta z + U\Delta x, \Lambda\Delta z\rangle \\
&= \langle \Delta x, Y\Delta z\rangle - \left\langle \Delta x, U^\top\Lambda\Delta z\right\rangle + \langle (I - W)\Delta z, \Lambda\Delta z\rangle \\
&= \frac{1}{2}\Delta z^\top \left(\Lambda(I - W) + (I - W^\top)\Lambda\right)\Delta z \geq \|Y\Delta z\|^2 \geq 0,
\end{aligned}$$

which further implies $\langle \Delta x, \Delta y\rangle - \mu\|\Delta x\|^2 \geq \langle \Delta v - \Delta z, \Lambda\Delta z\rangle \geq 0$. Thus, $\mathcal{H}$ is monotone. We can use the similar technique to derive the Lipschitz bound of $\mathcal{H}$. Firstly we have

$$\begin{aligned}
\gamma\|\Delta x\|^2 - \frac{1}{\gamma}\|\Delta\tilde{y}\|^2 - 2\langle \Delta v - \Delta z, \Lambda\Delta z\rangle &= \gamma\|\Delta x\|^2 - \frac{1}{\gamma}\|\Delta\tilde{y}\|^2 + 2\langle (I - W)\Delta z, \Lambda\Delta z\rangle - 2\langle U\Delta x, \Lambda\Delta z\rangle \\
&= \gamma\|\Delta x\|^2 - 2\langle \Delta x, \Delta\tilde{y}\rangle - \frac{1}{\gamma}\|\Delta\tilde{y}\|^2 + \Delta z^\top(2\Lambda - \Lambda W - W^\top\Lambda)\Delta z \\
&\geq \gamma\|\Delta x\|^2 - 2\langle \Delta x, \Delta\tilde{y}\rangle - \frac{1}{\gamma}\|\Delta\tilde{y}\|^2 + \frac{2}{\gamma}\|Y\Delta z\|^2 = \left\|\sqrt{\gamma}\Delta x - \frac{1}{\sqrt{\gamma}}\Delta\tilde{y}\right\|^2.
\end{aligned}$$

Due to (31) we can further obtain $\gamma^2\|\Delta x\|^2 \geq \|\Delta\tilde{y}\|^2$, i.e., $\mathcal{H}$ is γ-Lipschitz.

C.2. Proof of Proposition 3.5

Sufficient part: (5) $\Rightarrow$ (3). From (17) we have that $Y = U^\top\Lambda$. We check the inequality part of (3) as follows:

$$2\Lambda - W^\top\Lambda - \Lambda W = \Psi PP^\top\Psi = XX^\top \succeq XQQ^\top X^\top = \frac{2}{\gamma}Y^\top Y.$$

Necessary part: (3) $\Rightarrow$ (5). Since $H \succeq 0$ has band structure, then it can be decomposed into $H = XX^\top$ where X has the following block lower triangular structure (Davis, 2006):

$$X = \begin{bmatrix} X_{11} & & & \\ X_{21} & X_{22} & & \\ & \ddots & \ddots & \\ & & X_{L;L\ 1} & X_{LL} \end{bmatrix}.$$

For this special case, a way to construct X from Λ, W and further computation of the free parameters $d, F_k^a, F_k^b, F^q, F^?$ can be found in (Wang & Manchester, 2023). Finally, we need to show that $XX^\top \succeq 2/\gamma Y^\top Y$ is equivalent to $Y = \sqrt{\gamma/2}Q^\top X^\top$ for some $QQ^\top \preceq I$, which can be directly followed by Lemma 3 of (Rantzer, 1996).

C.3. Proof of Proposition 4.1

From Lemma A.1 we have

$$2I - (V - \gamma/\mu SS^\top) - (V - \gamma/\mu SS^\top)^\top = 2I - V - V^\top + 2\gamma/\mu SS^\top \succeq 2\gamma/\mu SS^\top \succeq 0. \tag{32}$$

Then, the equilibrium network (10) is well-posed by Theorem 1 of (Revay et al., 2020).

C.4. Proof of Proposition 4.2

We first show that $0 \in \mathcal{A}(z) + \mathcal{B}(z) + \mathcal{C}(z)$ is a monotone operator splitting problem. It is obvious that $\mathcal{B}, \mathcal{C}$ are maximal monotone operators. From Lemma A.1 we have $(I - V) + (I - V)^\top \succeq 0$, i.e. $\mathcal{A}$ is also monotone. Then, we show that the above operator splitting problem shares the same set of equilibrium points with the model inverse (10). First, we rewrite it into a two-operator splitting problem $0 \in \widehat{\mathcal{A}}(z) + \mathcal{B}(z)$ where $\widehat{\mathcal{A}} = \mathcal{A} + \mathcal{C}$. By applying the forward-backward splitting with parameter $\alpha = 1$, we can compute the solution z via the following iteration:

$$z^{k+1} = R_B(z^k - \widehat{\mathcal{A}}(z^k)) = \mathbf{prox}_f^1\left(z^k - \left(I - V + \gamma/\mu SS^\top\right)z^k + b_z\right) = \hat{\sigma}\left(\left(V - \gamma/\mu SS^\top\right)z^k + b_z\right).$$

Thus, any solution $z^?$ of the equilibrium network (10) is also an equilibrium point of the above iteration.

C.5. Proof of Proposition 5.1

First, we have $\nabla f(x) = G^\top(x)\mathcal{G}(x)$ where $G(x) = \nabla\mathcal{G}(x)$ satisfies $\|G(x)\| \geq \mu$. Then, the PL inequality holds for f with $m = \mu^2$, i.e.,

$$\frac{1}{2}\|\nabla f(x)\|^2 = \frac{1}{2}\mathcal{G}(x)^\top G(x)^\top G(x)\mathcal{G}(x) \geq \frac{\mu^2}{2}\|\mathcal{G}(x)\|^2 = \mu^2(f(x) - f^?). \tag{33}$$

D. Experiments

D.1. Training details

We choose ReLU as our default activation and use ADAM (Kingma & Ba, 2015) with one-cycle linear learning rate (Coleman et al., 2017) except the NGP case which SGD with piecewise constant scheduling. For the NGP case, we use the cross entropy loss while the L2 loss is used for the rest of the examples. We found that it can improve the model training by enforcing $Q^\top Q = I$, which can be done by fixing $F^p = 0$. Dataset and model architectures are described as follows.

1D Step function. The target function is a step function

$$f(x) = \begin{cases} 2, & x > 0 \\ -2, & x < 0 \end{cases}$$

which is monotone and 0-Lipschitz everywhere except the singularity point $x = 0$. We try to fit this curve with $(0.1, 10)$-Lipschitz models. The optimal fit is a linear piecewise continuous function with slope of 10 near $x = 0$ and slope of 0.1 near $x = \pm 2$. We take 1000 random samples from $[-2, 2]$ for training. Our model (BiLipNet) is an one-layer residual network $\mathcal{F}(x) = \mu x + \mathcal{H}(x)$ where $\mathcal{H}$ has 8 hidden layers of width 32, giving the model 15.8K parameters. We compare to i-ResNet (Chen et al., 2019) and i-DenseNet (Perugachi-Diaz et al., 2021), where the nonlinear block $\mathcal{H}$ has 2 and 4 hidden layers, respectively. For those two models, we test for depth from 2 to 8 with proper hidden width (so that they has similar amount of parameters). And the empirical Lipschitz bound is computed via finite difference over the test data. As shown in Figure 1, our model achieves much tighter bounds than other models.

Neural Gaussian process. We take 1000 two-moon data points as training data and 1000 Gaussian samples with mean $(1.3, -1.8)$ and variance $(0.02, 0.01)$ as OOD data. For all models, we use fixed input weight to mapping the 2D input into 128D hidden space, then perform hidden space transformation using bi-Lipschitz models, and finally add a Gaussian process as the output layer. SNGP uses 3 residual layer $x + \mathcal{H}(x)$ where the Lipschitz bound of $\mathcal{H}$ is $c < 1$. BiLipNet has one monotone and Lipschitz layer with two orthogonal layer, i.e., $K = 1$ for (8). The nonlinear block $\mathcal{H}$ of our model has 6 hidden layers with width of 32. Both models are chosen to have the same amount of parameters, roughly $233K$.

CIFAR-10/100 datasets. We first adopt the SNGP model from (Liu et al., 2020) and make some modifications as follows.

- SNGP contains three bi-Lipschitz components with each including four residual layers of the form $x + \mathcal{H}(x)$. It used spectral norm bound $c = 6$ for the weights inside $\mathcal{H}$, which means that the bi-Lipschitz property may not hold. To provide a certified guarantee of bi-Lipschitzness we need $c \in (0, 1)$. We tried three values of c: 0.35, 0.65 and 0.95. Since the Lipschitz bounds are $\mu = (1 - c)^4$ and $\nu = (1 + c)^4$, a larger c implies a more expressive SNGP model.
- We ran the SNGP with/without batch normalization for the bi-Lipschitz components. As pointed out in (Liu et al., 2023), the batch normalization may re-scale a layer's spectral norm in unexpected ways. So there is no theoretical guarantee on bi-Lipschitz property when batch normalization is applied.
- Training the original SNGP takes about 95% GPU memory of an Nvidia RTX3090. With the same number of parameters, our model needs more GPU memory as it uses the approach from (Trockman & Kolter, 2021) to perform the Cayley transform of convolution operators, which involves FFT and inverse FFT. In order to use a single GPU to train both models, we reduce the width of SNGP so that it has a similar amount of parameters as our model ($\sim$ 14M).

Our model has a similar structure to SNGP except that we replace their bi-Lipschitz components with our proposed bi-Lipschitz networks. Note that there is no batch normalization inside our bi-Lipschitz networks. All models are trained for 200 epochs using the mini-batch stochastic gradient descent (SGD) method with batch size of 256. We adjust the learning rate based on a piecewise constant schedule.

2D Rosenbrock function. The true function is a Rosenbrock function defined by

$$r(x, y) = \frac{1}{200}(x - 1)^2 + \frac{1}{2}\left(y - x^2\right)^2.$$

Note that we use a scaling factor of $1/200$ for the classic Rosenbrock function. The above function is non-convex but has one minimum at $(1, 1)$. We also consider the combination of the above Rosenbrock function with the following 2D Sine function:

$$s(x, y) = 0.25(\sin(8(x - 1) - \pi/2) + \sin(8(y - 1) - \pi/2) + 2).$$

In this case $r(x, y) + s(x, y)$ still has a unique global minimum at $(1, 1)$. But there are many local minima. We take 5K random training samples from the domain $[-2, -2] \times [-1, 3]$. The proposed BiLipNet contains two monotone and Lipschitz layers (i.e., $K = 2$ for (8)). The nonlinear block $\mathcal{H}$ has 4 hidden layers of width 128. The model size is roughly 16K. The ICNN model has 8 hidden layers with width of 180. The MLP has hidden units of $[128, 256, 256, 512]$. We trained i-ResNet and i-DenseNet with different depth and width such that the total amount of parameters is comparable with BiLipNet.

Parametric Rosenbrock function. We consider the following parametric Rosenbrock function

$$r(x, y; p) = \frac{1}{200}(x - a)^2 + \frac{1}{2}(y - bx^2)^2, \quad p = (a, b) \in [-1, 1]^2.$$

We take 10K random training data. The partially BiLipNet contains 3 orthogonal layers, and 2 monotone and Lipschitz layers (the $\mathcal{H}$ block of each layer has 4 hidden layer with width 128). The bias term of each orthogonal layer is produced by an MLP with hidden units of $[64, 128, 2]$ while the bias for those hidden units inside the $\mathcal{H}$ block is generated by an MLP of $[64, 128, 256, 512]$. The model's bi-Lipschitz bound is chosen to be $(0.04, 16)$. The resulting model size is 604K.

ND Rosenbrock function. We also consider the N-dimensional (with $N = 20$) Rosenbrock function:

$$R(x) = \frac{1}{N-1} \sum_{i=1}^{N} r(x_i, x_{i+1})$$

which is non-convex and has a unique global minimum at $(1, 1, \ldots, 1)$. Besides it also has many local minima. We take 10K random samples over the domain $[-2, 2]^{20}$ and do training with batch size of 200. Note that the data size is very small compared to the dimension. We then use 500K samples for testing. BiLipNet has two monotone and Lipschitz layers (i.e., $K = 2$ for (8)) where each layer has a nonlinear block $\mathcal{H}$ with 8 hidden layer of width 256 (model size $\sim$ 2.1M). For the i-ResNet/i-DenseNet, we try different depths from 2 to 10 and observe that depth of 5 yields slightly better results. The width of hidden layer is chosen so that it has a similar amount of parameters as BiLipNet.

D.2. Extra results

Some extra results for the bi-Lipschitz models on two-moon and CIFAR-10/100 datasets are shown in Figure 10 and Table 3, respectively. Figure 11 depicts the additional results on surrogate loss learning.

Method	c	Accuracy (↑) Clean	Accuracy (↑) Corrupted	ECE (↓) Clean	ECE (↓) Corrupted	NLL (↓) Clean	NLL (↓) Corrupted
CIFAR-10							
SNGP-BN	0.95	94.7 ± 0.079	73.0 ± 0.461	0.017 ± 0.002	0.127 ± 0.010	0.166 ± 0.004	0.991 ± 0.054
	0.65	94.1 ± 0.159	72.3 ± 0.561	0.016 ± 0.000	0.116 ± 0.005	0.182 ± 0.005	0.985 ± 0.029
	0.35	92.3 ± 0.260	70.4 ± 0.800	0.008 ± 0.003	0.095 ± 0.007	0.231 ± 0.006	0.995 ± 0.031
BiLipNet	0.95	86.2 ± 0.250	70.8 ± 0.469	0.020 ± 0.003	0.052 ± 0.005	0.423 ± 0.006	0.895 ± 0.020
	0.65	86.7 ± 0.129	72.8 ± 0.592	0.015 ± 0.005	0.047 ± 0.009	0.400 ± 0.006	0.830 ± 0.024
	0.35	84.5 ± 0.184	72.6 ± 0.216	0.010 ± 0.002	0.052 ± 0.004	0.457 ± 0.002	0.827 ± 0.008
CIFAR-100							
SNGP-BN	0.95	72.3 ± 0.513	44.8 ± 0.470	0.071 ± 0.006	0.091 ± 0.006	1.042 ± 0.018	2.476 ± 0.025
	0.65	67.8 ± 1.006	41.5 ± 0.916	0.117 ± 0.007	0.092 ± 0.002	1.231 ± 0.035	2.573 ± 0.036
	0.35	61.9 ± 0.741	37.0 ± 0.660	0.158 ± 0.006	0.098 ± 0.006	1.510 ± 0.029	2.760 ± 0.043
BiLipNet	0.95	51.0 ± 0.480	35.8 ± 0.397	0.230 ± 0.006	0.137 ± 0.007	2.064 ± 0.024	2.718 ± 0.014
	0.65	55.2 ± 0.426	39.2 ± 0.495	0.225 ± 0.004	0.137 ± 0.005	1.887 ± 0.021	2.576 ± 0.022
	0.35	54.4 ± 0.438	41.1 ± 0.200	0.194 ± 0.008	0.126 ± 0.009	1.876 ± 0.031	2.447 ± 0.016

Table 3. Results for SNGP-BN (SNGP with batch normalization) and BiLipNet (*without* batch normalization) on CIFAR-10/100, averaged over 5 seeds. As pointed out by (Liu et al., 2023), the batch normalization may rescale a layer's spectral norm in unexpected ways. So there is no theoretical guarantee on bi-Lipschitz property for SNGP-BN. This may offer it extra expressive power, leading to performance improvement in both clean and corrupted accuracy for a large distortion models (i.e. $c = 0.95$). For models with low distortion (i.e. $c = 0.35$), BiLipNet has better accuracy for the corrupted dataset.

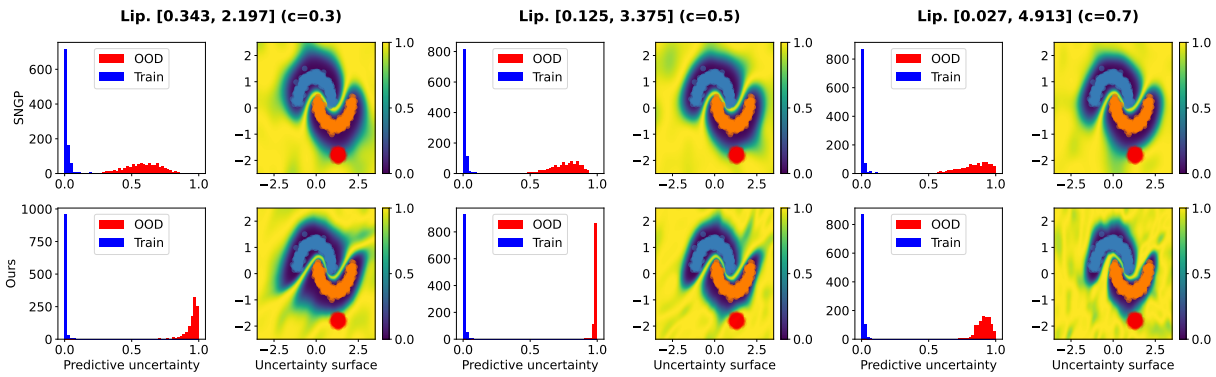


Figure 10. Uncertainty qualification via neural Gaussian process with different bi-Lipschitz bound specifications.

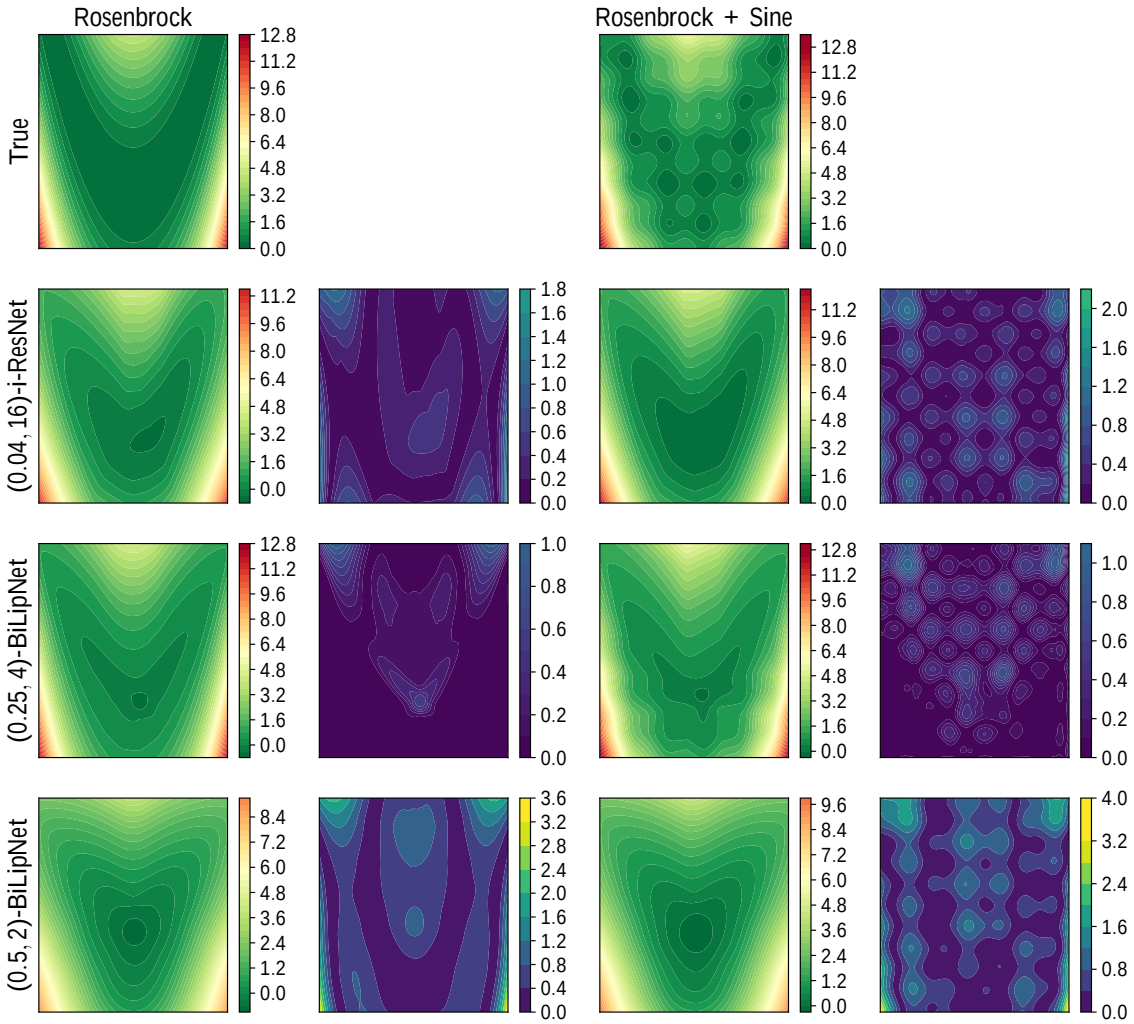


Figure 11. Additional results for Learning a surrogate loss for the Rosenbrock and Rosenbrock + Sine functions. The first row contains the true functions while the remaining rows show learned functions and errors for various surrogate loss models. Our model has the flexibility of capturing the non-convex sub-level sets, but can also fit smoothed representations by reducing the distortion parameter.