SNARENET: FLEXIBLE REPAIR LAYERS FOR NEURAL NETWORKS WITH HARD CONSTRAINTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural networks are increasingly used as fast surrogate models across various domains, but unconstrained predictions can violate physical, operational, or safety requirements. We propose SnareNet, a feasibility-controlled architecture to learn mappings whose outputs must satisfy input-dependent constraints. SnareNet appends a differentiable repair layer that navigates in the constraint map's range space, steering iterates toward feasibility and producing a repaired output that satisfies constraints to a user-specified tolerance. We stabilize end-to-end training by adaptive relaxation, a new training paradigm that snares the neural network at initialization and shrinks it into the feasible set, enabling early exploration and strict feasibility later in training. On optimization learning and trajectory planning benchmarks, SnareNet consistently attains improved objective quality while satisfying constraints more reliably than prior work, and it is the first to enforce *non-convex* constraints at medium-to-high precision robustly across instances.

1 INTRODUCTION

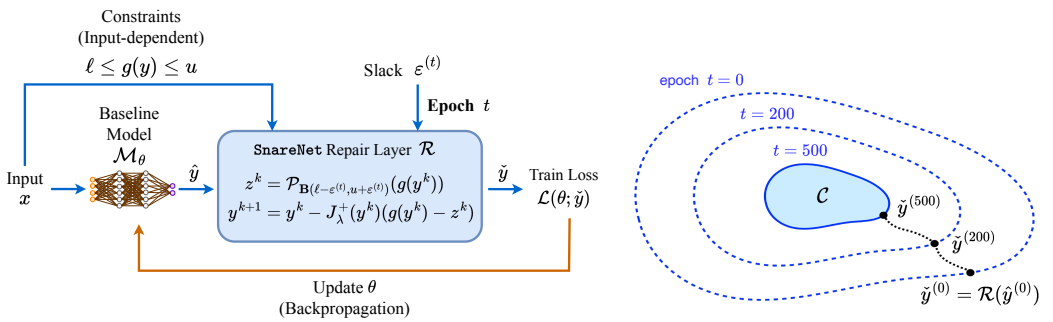
Deep learning models have emerged as powerful tools across diverse applications, from computational biology (Jumper et al., 2021; Angermueller et al., 2016) and drug discovery (Gómez-Bombarelli et al., 2018; Wu et al., 2018) to robotics (Levine et al., 2016) and autonomous systems (Grigorescu et al., 2020). Their success stems from an ability to learn complex patterns from data and make accurate predictions in high-dimensional spaces. Neural networks (NNs) are increasingly being deployed as fast surrogate models that can replace or supplement traditional computational methods. For instance, in domains where the same type of problem must be solved frequently in real time (e.g., power systems operation, (Pan, 2021); robotic control, (Williams et al., 2017)) or where individual solutions require expensive computation (e.g., weather forecasting, (Lam et al., 2023; Bonev et al., 2025); fluid dynamics simulations, (Kochkov et al., 2021)), NNs offer orders-of-magnitude speedups while maintaining competitive accuracy. Standard NN architectures deliver unconstrained outputs. While simple constraints like simplex constraints and coordinate-wise bounds can be readily enforced through architecture design, it is challenging to enforce complex constraints, even for linear constraints. Research on *constrained neural networks* is growing in the past few years (Amos & Kolter, 2017; Lu et al., 2021; Iftakher et al., 2025), driven by several compelling needs.

First, many critical applications require constrained outputs to ensure validity and safety. In learning-based control policies, outputs must satisfy safety constraints and physical actuator limits to prevent catastrophic failures (Ames et al., 2016). In applications such as optimal power flow in electrical grids, outputs must satisfy operational constraints including power balance equations and capacity limits to deliver an executable solution (Pan et al., 2020). Violating these constraints can lead to solutions that are not only suboptimal but potentially dangerous or nonsensical in practice.

Secondly, constraints provide a mechanism to inject domain knowledge and inductive biases into NNs, which can potentially improve generalization, particularly when training data is limited. In hierarchical time series forecasting (Cini et al., 2023; Stratigakos et al., 2022; Rangapuram et al., 2021), aggregation constraints across levels of the hierarchy ensures coherent forecasts. By encoding known structural properties of the problem into the network architecture, we can reduce the effective hypothesis space and regularize the learning process, leading to improved performance and robustness.

The most straightforward approach to impose constraints on NNs penalizes constraint violations in the loss function (Platt & Barr, 1987; Zhang & Constantinides, 1992; Márquez-Neila et al., 2017), known

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) SnareNet is equipped with an iterative repair layer.

(b) Adaptive relaxation training paradigm.

Figure 1: SnareNet's architecture design and training paradigm.

as *soft constraint methods*. However, this approach provides no guarantees on constraint satisfaction at inference time. Typical gradient-based training with soft constraints produces constraint violations on the order of 10^{-2} or worse (Donti et al., 2021), leading to physically nonsensical or operationally infeasible solutions. Perhaps more surprisingly, our numerical experiments in Appendix G reveal that soft constraint training can be counterproductive even as a warm-start strategy. These observations underscore the necessity for hard constraint enforcement mechanisms that provide both feasibility and efficient training.

In this paper, we introduce SnareNet (Figure 1), a novel framework to enforce nonlinear, input-dependent constraints on outputs of NNs while preserving their approximation capabilities and enabling end-to-end training. Our key contributions are:

- **Adaptive relaxation training paradigm:** We introduce a novel training strategy that progressively tightens constraint satisfaction during training (Figure 1b), improving optimality achieved by the final model while maintaining strict feasibility at the end of training.
- **Feasibility control even for non-convex constraints:** SnareNet enables users to explicitly control the desired level of constraint satisfaction, allowing for empirical trade-offs between strict feasibility and solution quality. SnareNet is the first repair layer to consistently achieve tight feasibility even on non-convex constraints.
- **Broad applicability and robustness:** We demonstrate SnareNet's effectiveness on optimization learning tasks and neural control policies. SnareNet consistently produces feasible solutions with better objective values compared to state-of-the-art baselines.

2 Constrained Neural Networks

The fundamental task in deep learning is to learn complex input-output mappings $\Phi : \mathcal{X} \to \mathcal{Y}$ from data. Given input $x \in \mathcal{X}$, the deep learning model $\mathcal{M}_\theta : \mathcal{X} \to \mathcal{Y}$ parametrized by weights θ is expected to produce an output $\hat{y} = \mathcal{M}_\theta(x)$ that approximates $y = \Phi(x)$. These deep learning models are typically trained by minimizing an empirical risk over a finite training dataset $\mathcal{X}_{\text{train}} \subset \mathcal{X}$ defined by $\mathcal{L}(\theta) = \frac{1}{|\mathcal{X}_{\text{train}}|} \sum_{x \in \mathcal{X}_{\text{train}}} \ell(\theta; x)$, where $\ell(\theta; x)$ denotes a suitable loss function measuring the discrepancy between predictions and targets for input x under model parameter θ. For simplicity, we consider the setting where $\mathcal{X} \subset \mathbb{R}^d$ and $\mathcal{Y} \subset \mathbb{R}^n$ admit vector representations.

In this paper, we consider the task to impose input-dependent (i.e., x-dependent) constraints on the output y, which take the form:

$$\ell_x \le g_x(y) \le u_x, \quad \text{(abbrev. } \ell \le g(y) \le u) \tag{1}$$

where $g : \mathbb{R}^n \to \mathbb{R}^m$ is differentiable, $\ell, u \in (\mathbb{R} \cup \{\pm\infty\})^m$ are lower and upper bounds respectively, and all g, ℓ, u are parametrized by input x. We suppress the dependence of g, ℓ, u on x throughout the paper for notational simplicity. The formulation in (1) encompasses equality constraints when $\ell_i = u_i$ for index i, as well as one-sided constraints when $\ell_i = -\infty$ or $u_i = \infty$. Throughout the paper, we assume the feasible set $\mathcal{C} := \{y \in \mathcal{Y} \mid \ell \le g(y) \le u\}$ is non-empty for any $x \in \mathcal{X}$.

Soft-Constraint Methods. To encourage the model $\mathcal{M}_\theta$ to produce feasible outputs, early approaches penalize constraint violations of the output[1] $\hat{y} = \hat{y}(\theta, x) := \mathcal{M}_\theta(x)$ in the loss function:

$$\ell_{\text{soft}}(\theta; x) = \ell(\theta; x) + \mu_u \|\text{ReLU}(g(\hat{y}) - u)\|^2 + \mu_\ell \|\text{ReLU}(\ell - g(\hat{y}))\|^2, \tag{2}$$

where $\mu_u > 0$ and $\mu_\ell > 0$ are penalty weights that determine the strength of the penalty, and $\text{ReLU}(\cdot) := \max(0, \cdot)$ is applied elementwise. The loss in (2) is also called a *soft loss* and allows the use of standard unconstrained optimization techniques during training. This soft constraint methodology is commonly used for data-parameterized constrained optimization (Van Hentenryck, 2025) and for solving partial differential equations (Raissi et al., 2019; Dener et al., 2020). While penalty methods are straightforward to implement and broadly applicable, models trained with soft loss generally violate constraints on unseen problem instances. Strict constraint satisfaction requires setting μ_u and μ_ℓ to infinity (or a very large number), which however makes the soft loss infinitely ill-conditioned and hard to solve (Rathore et al., 2024).

Hard-Constraint Methods. To impose the constraints (1) on the output, a more sophisticated approach appends a *repair module*[2] $\mathcal{R} : \mathbb{R}^n \to \mathbb{R}^n$ that maps any potentially infeasible output $\hat{y}$ to a feasible one $\check{y}$. We use $\hat{y} = \mathcal{M}_\theta(x)$, pronounced "y-hat", to denote the approximate solution from the baseline model, which is potentially infeasible; and we use $\check{y} = \mathcal{R}(\hat{y})$, pronounced "y-check", to denote the solution after applying the repair module, which is enforced to be feasible. Two paradigms have appeared to couple repair modules to a baseline NN, post-processing and end-to-end training.

Post-processing approaches apply the repair module only during inference, but do not allow differentiable training for model parameters θ. The simplest post-processing approach is projection onto the feasible set. Given any test instance x_{test} and a trained model $\mathcal{M}_\theta$, the repair module projects the baseline output onto the feasible set, i.e.,

$$\mathcal{R}(\hat{y}(\theta, x_{\text{test}})) := \underset{y \in \mathcal{C}}{\arg\min} \|y - \mathcal{M}_\theta(x_{\text{test}})\|^2. \tag{3}$$

When $\mathcal{C}$ is convex, problem (3) is a convex program that typically can be reliably solved by existing optimization solvers. When $\mathcal{C}$ is non-convex, optimization solvers still reliably find feasible (but not always optimal) solutions to (3). Alternatively, Liang et al. (2024) propose to use an auxiliary deep learning model that learns a homeomorphic mapping from the feasible set to the unit ball. However, their method needs binary search to project an infeasible solution to the feasible set, which precludes end-to-end training of the combined system. Diffusion-based samplers (Ding et al., 2025) recover feasibility by generating and selecting among multiple candidates, yielding only probabilistic feasibility guarantees. Post-processing can lead to critical problems as demonstrated in Grontas et al. (2025, Appendix C.4): models trained without active constraints may diverge on unbounded objectives. Even when they converge, solution quality may be poor after projection since the baseline model fails to anticipate the repair operation.

Trainable layers enable end-to-end approaches that activate the repair layer during both training and inference, allowing gradients of model parameters θ to flow through the repair operation. The baseline model $\mathcal{M}_\theta$ can therefore adapt to the repair operation. To enable backpropagation, the repair layer must use only operations that are differentiable almost everywhere. OptNet (Amos & Kolter, 2017) proposes differentiable convex optimization problem as a layer, for which the gradients are calculated from the KKT conditions inU backward pass. Under convex constraints, the convex projection problem (3) can be appended as a repair layer, albeit at the cost of expensive training. State-of-the-art lightweight trainable repair layers typically employ a fixed number of iterative updates derived from classical optimization algorithms, such as DC3 (Donti et al., 2021), FSNet (Nguyen & Donti, 2025), ΠInet (Grontas et al., 2025), and ENFORCE (Lastrucci & Schweidtmann, 2025). DC3 and FSNet both move an initial prediction toward the feasible region via gradient descent on the distance to feasibility. ΠInet decomposes an affine feasible set as the intersection of two convex sets in lifted dimension and applies Douglas-Rachford algorithm, in which the computational efficiency is improved by restricting the framework to linear and box constraints. ENFORCE handles only equality constraints through

[1] We denote the output by $\hat{y}$ for simplicity when the dependence on θ and x is clear from the context.

[2] We borrow the word "repair" from the term *repair layer* in Van Hentenryck (2025), but we call $\mathcal{R}$ a *layer* only if it is trainable. The same concept has also been referred to as a *projection layer* (Min & Azizan, 2024; Grontas et al., 2025; Liang et al., 2024) or a *feasibility layer* (Li et al., 2023; Nguyen & Donti, 2025) in the literature when the repair module is trainable.

iterative linearization and projection. While iterative repair layers improve constraint satisfaction, backpropagation through these iterations requires intensive GPU memory. In contrast, HardNet (Min & Azizan, 2024) uses a closed-form repair layer that guarantees exact feasibility up to machine precision and enables efficient backpropagation, but HardNet is limited to linear constraints and often produces suboptimal solutions.

Key Challenges. A trainable repair layer must reach feasibility using only differentiable operations. Differentiability can be achieved by unrolling iterative updates, but repair layers that use too many iterations generate large unrolled computational graphs and yield slow training. Convex optimization layers (Amos & Kolter, 2017) avoid unrolling, but cannot handle non-convex constraints. Designing an efficient repair layer for non-convex constraints remains an open challenge.

3 Novel Preimage Perspective From Linear Constraints

We develop a new interpretation of the closed-form repair layer for full row rank linear constraints proposed by HardNet (Min & Azizan, 2024). Our preimage perspective interprets the inequality feasibility problem as a more tractable equation solving problem. Our preimage perspective suggests a natural generalization to nonlinear and nonconvex constraints.

3.1 Closed-Form Repair Layer for Linear Constraints and Preimage Perspective

HardNet (Min & Azizan, 2024) shows that iterative repair layers are unnecessary by introducing a closed-form repair layer that strictly enforces all linear constraints $\ell \le Ay \le u$ when $A \in \mathbb{R}^{m\times n}$ has full row rank:

$$\mathcal{R}(\hat{y}) := \hat{y} + A^{+}\boldsymbol{\delta}(A\hat{y};\ell,u), \quad \text{where } \boldsymbol{\delta}(z;\ell,u) := \mathrm{ReLU}(\ell - z) - \mathrm{ReLU}(z-u) \tag{4}$$

and A^+ is the pseudoinverse of A. The *correction vector* $\boldsymbol{\delta}(z;\ell,u)$ corrects z to the box $\mathbf{B}(\ell,u) := \{z \in \mathbb{R}^m \mid \ell_i \le z_i \le u_i,\ \forall i \in [m]\}$ by elementwise projection since

$$\mathcal{P}_{\mathbf{B}(\ell,u)}(z) := \begin{cases} z_i, & \text{if } \ell_i \le z_i \le u_i; \\ \ell_i, & \text{if } z_i \le \ell_i; \\ u_i, & \text{if } z_i \ge u_i; \end{cases} \quad = z + \boldsymbol{\delta}(z;\ell,u). \tag{5}$$

Observe that the preimage of $\mathbf{B}(\ell,u)$ under the map A is exactly the feasible set $\mathcal{C}$:

$$\mathcal{C} = \{y \in \mathbb{R}^n \mid \ell \le Ay \le u\} = \{y \in \mathbb{R}^n \mid Ay \in \mathbf{B}(\ell,u)\} = \text{Preimage of } \mathbf{B}(\ell,u) \text{ under } A.$$

We interpret HardNet through the preimage: HardNet projects the image vector $A\hat{y}$ onto the box $\mathbf{B}(\ell,u)$ and chooses a particular feasible $\check{y} = \mathcal{R}(\hat{y})$ that lies in the preimage of $\mathcal{P}_{\mathbf{B}(\ell,u)}(A\hat{y})$. That is, we can *reformulate the inequality feasibility problem as a tractable equation solving problem*: $A\check{y} = \mathcal{P}_{\mathbf{B}(\ell,u)}(A\hat{y})$ using box projector; and moreover, the bix projector admits a closed-form expression (5). When A has full row rank, HardNet solves for $\check{y}$ by updating $\hat{y}$ with the vector $A^+\boldsymbol{\delta}(A\hat{y};\ell,u)$ that is pulled back from the correction $\boldsymbol{\delta}(A\hat{y};\ell,u)$ in the constraint space by the pseudoinverse A^+ (Figure 2a). Linearity of A ensures $A^+\boldsymbol{\delta}(A\hat{y};\ell,u)$ serves as a feasibility correction:

$$A\mathcal{R}(\hat{y}) = A(\hat{y} + A^{+}\boldsymbol{\delta}(A\hat{y};\ell,u)) = A\hat{y} + \boldsymbol{\delta}(A\hat{y};\ell,u) = \mathcal{P}_{\mathbf{B}(\ell,u)}(A\hat{y}).$$

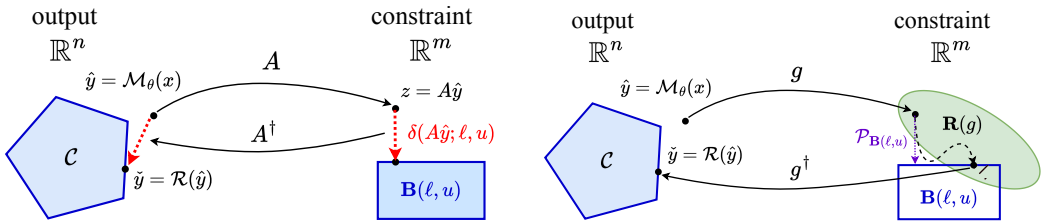
3.2 The challenge of non-linear constraints

The above reformulation that uses the box projector to convert an inequality feasibility problem into an equation-solving problem does not directly extend to nonlinear constraints. Example 1 illustrates the challenge:

Example 1. Constrain $y \in \mathbb{R}^2$ to the intersection of two disks of radius $3/2$, centered at $(\pm 1, 0)$:

$$g(y) = \begin{bmatrix} g_1(y) \\ g_2(y) \end{bmatrix} := \begin{bmatrix} (y_1+1)^2 + y_2^2 \\ (y_1-1)^2 + y_2^2 \end{bmatrix} \le u := \begin{bmatrix} 9/4 \\ 9/4 \end{bmatrix}.$$

Now, the infeasible prediction $\hat{y} = (-1, 0)$ violates the second constraint but satisfies the first one as $g_1(\hat{y}) = 0$ and $g_2(\hat{y}) = 4$. The box projection of $g(\hat{y})$ is $\mathcal{P}_{\mathbf{B}(-\infty,u)}(g(\hat{y})) = (0, \frac{9}{4})$. However, there is *no* point $y \in \mathbb{R}^2$ such that $g(y) = (0, \frac{9}{4})$ since the system of equations $g_1(y_1, y_2) = (y_1+1)^2 + y_2^2 = 0$ and $g_2(y_1, y_2) = (y_1 - 1)^2 + y_2^2 = \frac{9}{4}$ has no solution and the preimage of $(0, \frac{9}{4})$ under g is empty.



(a) Linear $g(y) = Ay$ with full row rank A

(b) Nonlinear $g(y)$

Figure 2: Preimage perspective to reformulate inequality feasibility problems to tractable equation solving problems. HardNet corrects an infeasible prediction $\hat{y}$ by a correction pulled back from the constraint space to the output space (Figure 2b). SnareNet finds a path to approach the intersection $\mathbf{R}(g) \cap \mathbf{B}(\ell, u)$ (Figure 2b).

Example 1 reveals the fundamental difficulty: the *joint numerical range* $\mathbf{R}(g) := \{g(y) \in \mathbb{R}^m \mid y \in \mathbb{R}^n\}$ of non-linear g is typically a proper subset of $\mathbb{R}^m$. Hence, the projection of $g(\hat{y})$ onto $\mathbf{B}(\ell, u)$ may have empty preimage under g. To guarantee feasibility, we must move towards the intersection $\mathbf{R}(g) \cap \mathbf{B}(\ell, u)$, which characterizes points that are both (i) in the range of g (ensuring a preimage exists) and (ii) within the feasible box (ensuring constraint satisfaction). See Figure 2b.

4 SnareNet: Design and Analysis

This section introduces SnareNet (Figure 1), which efficiently finds a feasible point satisfying (1).

4.1 Repair Layer: Regularized Adaptive Newton Update

Given $z \in \mathbf{R}(g) \cap \mathbf{B}(\ell, u)$, Newton's method can find a feasible $\check{y} \in \mathcal{C}$ by solving the non-linear system $g(y) = z$. SnareNet uses $z^k = \mathcal{P}_{\mathbf{B}(\ell,u)}(g(y^k))$ as the initial RHS at each Newton iteration:

$$y^{k+1} = \arg\min_y \|J_g(y^k)(y - y^k) + g(y^k) - z^k\|^2 = y^k - J_g(y^k)^+(g(y^k) - z^k), \tag{6}$$

where $J_g(y) \in \mathbb{R}^{m \times n}$ is the Jacobian of g. When $g = A$ has full row rank, z^k must admit a preimage, Newton's method converges in one iteration, and the update (6) reduces to (4) in HardNet. The linearization in Newton's method (6) approximates g only locally and thus requires safeguards to ensure convergence. SnareNet uses Levenberg–Marquardt (LM) regularization, replacing (6) by

$$y^{k+1} = \arg\min_y \|J_g(y^k)(y - y^k) + g(y^k) - z^k\|^2 + \lambda\|y - y^k\|^2 = y^k - J_\lambda^\dagger(y^k)(g(y^k) - z^k), \tag{7}$$

where $J_\lambda^\dagger(y^k) := [J_g(y^k)^\top J_g(y^k) + \lambda I]^{-1} J_g(y^k)^\top$. The regularization term $\lambda\|y - y^k\|^2$ with $\lambda > 0$ ensures the new point is close to y^k, at which g is linearized. SnareNet terminates when the update magnitude or constraint violation falls below a prescribed tolerance or the iteration limit is reached.

4.2 Training Paradigm: Adaptive Relaxation

Strict constraints enforcement during early-stage training can harm performance. Min & Azizan (2024) observes that immediate enforcement prevents models from reaching better-optimized final states, and solves the problem by disabling the repair layer and using soft loss (2) in the first few epochs, also called "soft-epochs". Nguyen & Donti (2025) show that randomly initialized networks produce large violations that force the repair layer to work overtime and worsen the final solution.

SnareNet instead uses a new *adaptive relaxation* (AdaRel) training paradigm. In the beginning of training, we repair the approximate solution $\hat{y}$ towards a relaxed feasible set $\mathcal{C}_{\varepsilon^{(t)}} := \{y \in \mathbb{R}^n \mid \ell - \varepsilon^{(t)} \le g(y) \le u + \varepsilon^{(t)}\}$ parametrized by a slack $\varepsilon^{(t)} \ge 0$ at epoch t. Our experiments use a slack $\varepsilon^{(t)}$ linearly decaying to zero over a preset decay horizon, allowing the model to explore a broader solution space initially while gradually tightening the constraints. We set $\varepsilon^{(t)} = 0$ for the last few training epochs to ensure exact feasibility. At epoch t, we implement AdaRel by replacing z^k in (7) with $z^k := \mathcal{P}_{\mathbf{B}(\ell - \varepsilon^{(t)}, u + \varepsilon^{(t)})}(g(y^k))$.

4.3 Analysis through the Lens of Preconditioning

SnareNet's update (7) is equivalent to *preconditioned gradient descent* (PGD) on constraint violation $F(y) = \frac{1}{2}\|g(y) - \mathcal{P}_{\mathbf{B}(\ell,u)}(g(y))\|^2$ with preconditioner $P_\lambda(y) := (J_g(y)^\top J_g(y) + \lambda I)^{-1}$ for $\lambda > 0$. Proposition A.1 shows that $F(y)$ is differentiable if g is and $\nabla F(y) = J_g(y)^\top (g(y) - \mathcal{P}_{\mathbf{B}(\ell,u)}(g(y)))$. The convergence guarantees of SnareNet are grounded in PGD update and properties of F such as convexity and Polyak-Lojasiewicz (PL) condition. Theorem 4.1 demonstrates our preconditioner $P_\lambda(y)$ enables a fast linear convergence rate for full row rank linear constraints. Theorem 4.2 justifies asymptotic feasibility under convex constraints. SnareNet minimizes constraint violation by PGD for non-convex constraints.

Theorem 4.1 (Linear Convergence for Full Row Rank Linear Constraints)**.** *Suppose $\mathcal{C} := \{y \in \mathbb{R}^n \mid \ell \leq Ay \leq u\}$ is non-empty and A has full row rank with minimum singular value $\sigma_{\min} > 0$. Then SnareNet with $\lambda > 0$ converges linearly to a feasible point at rate $F(y^k) \leq \left(1 - \frac{\sigma_{\min}^2}{\sigma_{\min}^2 + \lambda}\right)^k F(y^0)$.*

Theorem 4.1 suggests that a smaller λ leads to faster convergence and $\lambda = 0$ matches one step convergence as in HardNet. However, our empirical results (Figure 4) show that a small λ can lead to feasible solutions with worse optimality.

Theorem 4.2 (Convergence for Convex Constraints)**.** *Suppose $\mathcal{C} := \{y \in \mathbb{R}^n \mid h(y) \leq 0,\ \ell_A \leq Ay \leq u_A\}$ is non-empty, where $h : \mathbb{R}^n \to \mathbb{R}^{m_h}$ is a twice differentiable convex function with L-Lipschitz continuous Jacobian, $A \in \mathbb{R}^{m_A \times n}$, and $\ell_A, u_A \in \mathbb{R}^{m_A}$. Let $y^0 \in \mathbb{R}^n$ be the initial point, $\mathcal{D} := \{y \in \mathbb{R}^n \mid F(y) \leq F(y^0)\}$ be the sublevel set, and $D := \sup_{y \in \mathcal{D}} \inf_{y^* \in \mathcal{C}} \|y - y^*\|$. Suppose $\sigma_{\max}^2(J_h(y)) + \sigma_{\max}^2(A) \leq \sigma^2$ for all $y \in \mathcal{D}$ and $\lambda > 0$ satisfies $\lambda^2 - 2\sqrt{2F(y^0)}L\lambda - \frac{L^2 F(y^0)}{8} \geq 0$. Then SnareNet converges to a feasible point at rate $F(y^k) \leq \frac{F(y^0)}{1 + \frac{F(y^0)}{2D^2(\sigma^2+\lambda)}k}$.*

Remark 4.3 (Guarantees on Non-Convex Constraints). SnareNet's update satisfies the descent property (23), paralleling the sufficient decrease of gradient descent (GD). Hence, as in non-convex GD analysis, SnareNet converges to a stationary point of F, and to a feasible point if stronger assumptions are imposed on F (e.g., a PL condition). We omit this case, as such assumptions are restrictive and the result follows from standard non-convex optimization theory.

4.4 Computational Remarks

The primary limitation of SnareNet, shared by all extant repair layers in the literature, is the computational cost of repair: applying $J_\lambda^\dagger(y^k)$ with a direct (factor-solve) method costs $\mathcal{O}(nm^2 + m^3)$ (for $m < n$) using the push-through identity $[J^\top J + \lambda I]^{-1} J^\top = J^\top [JJ^\top + \lambda I]^{-1}$. SnareNet's training cost grows with tighter feasibility tolerances and larger m and/or n. However, compared to competitors, SnareNet strikes a balance between efficient repair and computational overhead. Cheaper methods like DC3 fail to enforce constraints consistently across instances, while existing robust methods like OptNet require significantly longer training time and work only on convex constraints. Our experiments show that SnareNet consistently ensures a medium-to-high feasibility tolerance, even for *non-convex* constraints, while training faster than OptNet. SnareNet is more efficient on less constrained problems as it uses fewer repair iterations (see Appendix H).

As an alternative to direct inversion, iterative methods like conjugate gradient can find an approximate solution for high-dimensional problems using cheaper iterations, but often require more iterations to converge and may even diverge. These methods do not form the Jacobian, but require only vector-Jacobian products (VJP), computed efficiently using automatic differentiation. Empirically, we observe divergence of SnareNet with iterative methods, so our experiments use a direct solver.

5 Experiments

We demonstrate the effectiveness of SnareNet on optimization learning and neural control policies[3]. Experiments run on a server with two 64-core AMD EPYC 7763 @ 2.45 GHz, 1 TB RAM, and NVIDIA A6000 GPUs unless otherwise specified.

[3]Code is available at `https://anonymous.4open.science/r/SnareNet-26B6/README.md`

5.1 Optimization Learning

Optimization learning (Van Hentenryck, 2025) task seeks a fast surrogate neural solver for a family of optimization problems parametrized by input $x \in \mathcal{X}$:

$$\min_{y \in \mathbb{R}^n} f_x(y) \;\text{ subject to }\; \ell_x \leq g_x(y) \leq u_x, \tag{8}$$

Problems of the form (8) can be non-linear and non-convex, and can be slow to solve with traditional optimization solvers. The goal of optimization learning is to learn a model $\mathcal{M}_\theta$ that approximates the solution map $\Phi : \mathcal{X} \to \mathbb{R}^n$, which maps the instance parameter x to optimal solution $y^\star = \Phi(x)$ of (8). In particular, $\mathcal{M}_\theta$ must produce a feasible solution for all $x \in \mathcal{X}$.

Families of Optimization Problems. We consider three families: non-convex linearly-constrained programs (NCLPs) and convex/non-convex quadratically constrained quadratic programs (QCQPs).

$$\textbf{(NCLP)} \quad \begin{array}{ll} \min\limits_{y \in \mathbb{R}^n} & \frac{1}{2} y^T Q y + p^T \sin(y) \\ \text{s.t.} & Ay \leq b,\; Cy = x, \end{array} \qquad \textbf{(QCQP)} \quad \begin{array}{ll} \min\limits_{y \in \mathbb{R}^n} & \frac{1}{2} y^T Q y + p^T y \\ \text{s.t.} & y^T H_i y + g_i^T y \leq h_i,\; i = 1, \ldots, m_{\text{ineq}} \\ & Cy = x, \end{array}$$

where $x \in \mathbb{R}^{m_{\text{eq}}}$ is the input and the other problem data are constant within the family. Each family consists of 10000 problem instances and is split into train/valid/test set in the ratio 8:1:1.

Baselines. We compare SnareNet to four baselines: OptNet (Amos & Kolter, 2017), DC3 (Donti et al., 2021), and HardNet (Min & Azizan, 2024) achieve feasibility using end-to-end trainable repair layers; and HProj (Liang et al., 2024) uses post-processing with a learning-based projection module. Further details of each baseline appear in Appendix B.

Evaluation. Table 1 summarizes the six evaluation metrics in our experiments. Each subpart in Figure 3 follows the same 2-by-3 layout. All experiments are run for 5 random seeds, over which the metrics are averaged. The optimality gap is assessed relative to SciPy's SLSQP solver (Virtanen et al., 2020) on NCLPs and Gurobi (Gurobi Optimization, LLC, 2024) on convex QCQPs. We evaluate optimality by objective values on non-convex QCQPs due to long traditional solver time. The complete test results are provided in Appendix C.

Table 1: Six evaluation metrics where $\mathrm{gmean}(\cdot)$ denotes the geometric mean.

	Optimality Gap / Objective	**Inequality Violation**	**Equality Violation**
Geometric Mean	$\underset{x \in \mathcal{X}_{\text{test}}}{\mathrm{gmean}}(f_x(\hat{y}) - f_x^\star) \,/\, \underset{x \in \mathcal{X}_{\text{test}}}{\mathrm{mean}} f_x(\hat{y})$	$\underset{x \in \mathcal{X}_{\text{test}}}{\mathrm{gmean}}\, \underset{j \in [n_{\text{ineq}}]}{\mathrm{gmean}}(j\text{-th inequality/equality violation})$	
Maximum	$\max\limits_{x \in \mathcal{X}_{\text{test}}} (f_x(\hat{y}) - f_x^\star) \,/\, \max\limits_{x \in \mathcal{X}_{\text{test}}} f_x(\hat{y})$	$\max\limits_{x \in \mathcal{X}_{\text{test}}} \max\limits_{j \in [n_{\text{ineq}}]} (j\text{-th inequality/equality violation})$	

Feasibility Control and Low Variation. Figure 3 evaluates trained neural solvers on a separate test set for all three families[4]. DC3 and HProj show larger seed-to-seed variability in inequality-constraint violations and optimality gap and may fail to ensure feasibility across all test instances, especially for QCQPs. HardNet exhibits larger variability in maximum optimality gap. Our SnareNet shows less variability and successfully controls feasibility up to the specified tolerance.

Better Optimality. Figure 3 also shows that SnareNet achieves optimality gaps at least one order of magnitude smaller than all baseline methods except for OptNet while maintaining feasibility within the prescribed feasibility tolerance `tol`. However, OptNet does not apply to non-convex constraints and requires much longer training to achieve similar performance on convex constraints. HProj exhibits substantial variation in optimality gap across random seeds[5]because its projection step is applied post-training, which can degrade optimality.

[4]Each run for NCLPs/convex QCQPs/non-convex QCQPs was terminated after 1500/1500/2000 epochs or 16/24/24 hours.

[5]The results of HProj on non-convex QCQPs are averaged over 3 of 5 random seeds (2 were excluded due to numerical instability and outlier objective values of ≈ 5000, respectively.

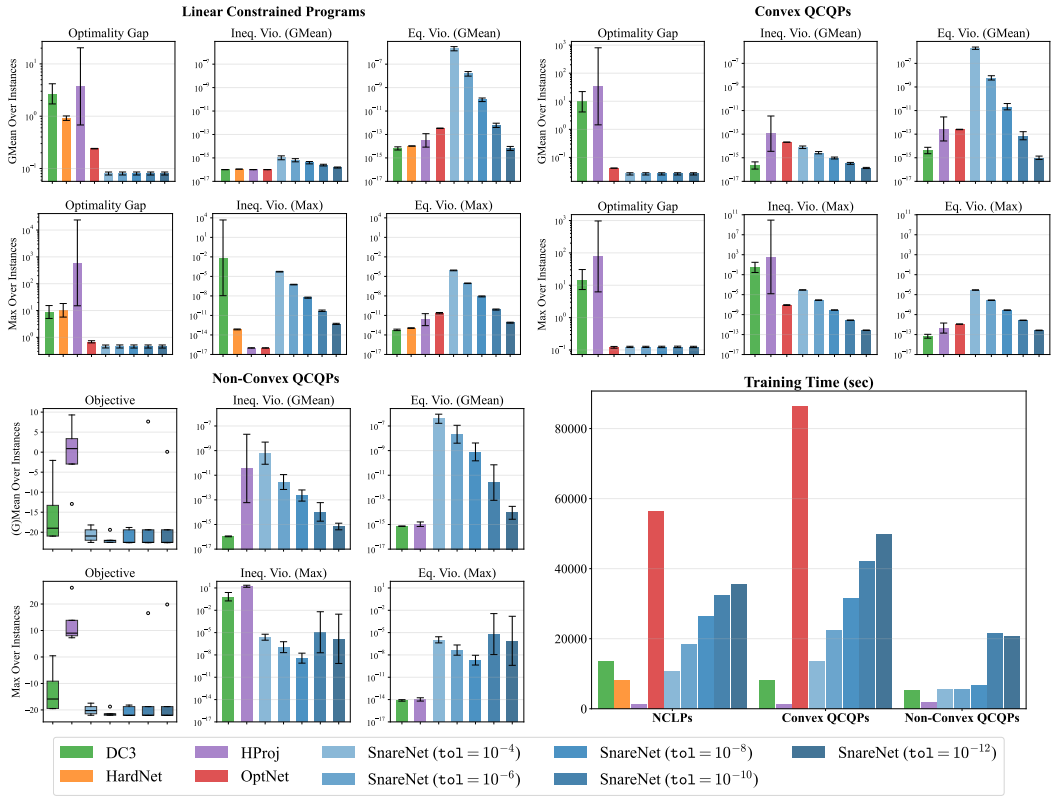


Figure 3: Evaluation metrics on 833 test instances of NCLPs and Convex/Non-Convex QCQPs and the average training time. Black error bars indicate the standard deviation across 5 random seeds.

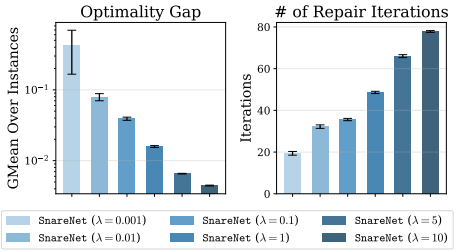


Figure 4: Optimality gap and repair iterations on 833 test instances of NCLPs.

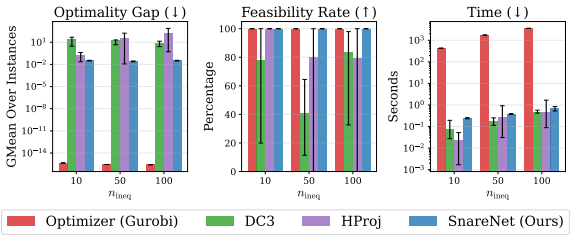


Figure 5: Performance on convex QCQPs with various number of inequality constraints.

λ Improves Optimality Even for Linear Constraints. SnareNet specializes to HardNet when $\lambda = 0$. Larger $\lambda > 0$ requires more iterations to enforce feasibility but can improve the optimality gap by orders of magnitude. See Figure 4.

SnareNet Handles Many Constraints. HardNet requires full row rank linear constraints and is thus limited to at most $2n$ constraints. SnareNet overcomes this limitation through LM-regularization. Figure 5 presents test results on convex QCQPs with $n_{\text{ineq}} \in \{10, 50, 100\}$ inequality constraints. SnareNet consistently produces feasible solutions within tolerance 10^{-4} across all test instances. In contrast, DC3 achieves feasibility on only 80% of test instances on average and exhibits high sensitivity to random initialization. HProj shows substantial variation in feasibility rates for problems with $n_{\text{ineq}} = 50$ and 100 inequality constraints. Inference time for the classical solver Gurobi is significantly slower than neural solvers and scales rapidly with the number of inequality constraints.

Adaptive Relaxation beats Soft Epochs. Table 2 summarizes an ablation study on the warm-up strategy used in HardNet and SnareNet with different λ values. Each model was trained for 1000 epochs in total, with the warm-up strategy applied for the first 500 epochs. The feasibility tolerance of SnareNet was set to 10^{-8} for computational efficiency. We make three observations: 1) Warm-up strategy is required to achieve better optimality. 2) Adaptive relaxation (AdaRel) reduces both the maximum and geometric mean of the optimality gap compared to other warm up strategies. 3) The improvement of AdaRel over soft epochs is more pronounced for smaller λ (e.g., HardNet $\lambda = 0$).

Table 2: Ablation on warm-up strategies for NCLPs. Values are mean $\pm$ std across 5 random seeds.

Method	Warm-up	Max Opt. Gap.	GMean Opt. Gap.	Max Ineq. Error	Gmean Ineq. Error	Max Eq. Error	Gmean Eq. Error
HardNet (i.e. $\lambda = 0$)	None	$(1.66 \pm 0.35) \times 10^{1}$	8.61 ± 1.62	$(2.14 \pm 0.39) \times 10^{-13}$	$(2.72 \pm 0.53) \times 10^{-16}$	$(2.53 \pm 0.71) \times 10^{-13}$	$(2.22 \pm 0.57) \times 10^{-14}$
HardNet (i.e. $\lambda = 0$)	500 Soft Epochs	9.34 ± 3.40	1.73 ± 0.58	$(5.88 \pm 1.42) \times 10^{-14}$	$(1.03 \pm 0.00) \times 10^{-16}$	$(9.55 \pm 1.07) \times 10^{-14}$	$(8.50 \pm 0.39) \times 10^{-15}$
HardNet (i.e. $\lambda = 0$)	500 AdaRel	9.29 ± 8.41	1.01 ± 0.25	$(5.41 \pm 1.10) \times 10^{-14}$	$(1.04 \pm 0.00) \times 10^{-16}$	$(9.04 \pm 0.99) \times 10^{-14}$	$(8.54 \pm 0.28) \times 10^{-15}$
SnareNet ($\lambda = 0.001$)	None	9.52 ± 3.34	6.15 ± 1.75	$(5.67 \pm 1.08) \times 10^{-9}$	$(2.09 \pm 1.67) \times 10^{-15}$	$(8.23 \pm 1.24) \times 10^{-9}$	$(3.24 \pm 1.00) \times 10^{-10}$
SnareNet ($\lambda = 0.001$)	500 Soft Epochs	3.25 ± 1.55	$(5.84 \pm 1.93) \times 10^{-1}$	$(2.05 \pm 0.28) \times 10^{-9}$	$(1.22 \pm 0.03) \times 10^{-16}$	$(8.14 \pm 1.14) \times 10^{-9}$	$(3.59 \pm 3.57) \times 10^{-10}$
SnareNet ($\lambda = 0.001$)	500 AdaRel	2.23 ± 1.03	$(3.57 \pm 1.01) \times 10^{-1}$	$(3.36 \pm 2.20) \times 10^{-9}$	$(1.23 \pm 0.04) \times 10^{-16}$	$(6.70 \pm 2.17) \times 10^{-9}$	$(4.08 \pm 1.15) \times 10^{-10}$
SnareNet ($\lambda = 0.1$)	None	5.63 ± 0.49	4.49 ± 0.53	$(1.04 \pm 0.63) \times 10^{-8}$	$(2.90 \pm 0.80) \times 10^{-15}$	$(2.00 \pm 0.61) \times 10^{-8}$	$(2.26 \pm 1.04) \times 10^{-9}$
SnareNet ($\lambda = 0.1$)	500 Soft Epochs	$(1.90 \pm 0.27) \times 10^{-1}$	$(3.91 \pm 0.44) \times 10^{-2}$	$(1.46 \pm 0.21) \times 10^{-8}$	$(3.23 \pm 0.16) \times 10^{-16}$	$(2.16 \pm 0.31) \times 10^{-8}$	$(4.10 \pm 0.94) \times 10^{-10}$
SnareNet ($\lambda = 0.1$)	500 AdaRel	$(2.11 \pm 0.25) \times 10^{-1}$	$(3.79 \pm 0.28) \times 10^{-2}$	$(1.39 \pm 0.37) \times 10^{-8}$	$(3.30 \pm 0.13) \times 10^{-16}$	$(2.08 \pm 0.51) \times 10^{-8}$	$(4.08 \pm 1.26) \times 10^{-10}$

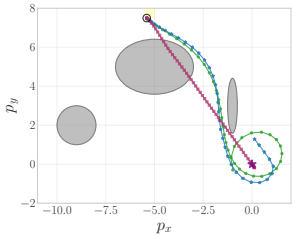
5.2 Neural Control Policies

Neural control policies use deep neural networks to learn mappings from system states x to control actions u, either as standalone policies or to enhance traditional control methods. In this experiment, we train a neural network policy $\pi_\theta(x)$ to control a unicycle system, avoiding obstacle collisions by enforcing safety constraints (Min & Azizan, 2024; Tayal et al., 2024). The neural policy is constructed as the sum of a nominal controller $\pi_{\text{nom}}(x)$, designed without obstacle awareness, and a learned correction term $\mathcal{M}_\theta(x)$: $\pi_\theta(x) := \pi_{\text{nom}}(x) + \mathcal{M}_\theta(x)$. We use *control barrier functions* (CBFs) (Ames et al., 2019) defined in Appendix E to parametrize the safe set. Collision-free trajectories are guaranteed by enforcing

$$\nabla h_j(x)^T (F(x) + G\pi_\theta(x)) \geq -\alpha h_j(x)$$

for each obstacle with CBF h_j and some $\alpha > 0$. The number of constraints scales linearly with the number of obstacles. HardNet can avoid at most two obstacles as it requires full row rank constraints and the system has only two controls, whereas SnareNet handles arbitrarily many obstacles. Figure 6 shows the learned neural policies and the detailed evaluation statistics can be found in Appendix E. Across all test instances, SnareNet achieves a 99.4% feasibility rate, compared to DC3's 64.9% feasibility rate, demonstrating the effectiveness of SnareNet in enforcing safety constraints in neural control policies. Since we use a higher order CBF, a trajectory may violate a constraint without visually colliding with any obstacle, e.g., DC3 trajectory in Figure 6a.

(a) Initialization box 1



(b) Initialization box 2

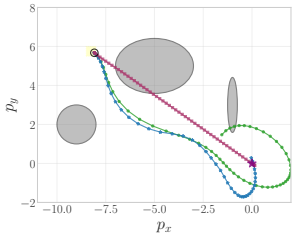




Figure 6: Simulated trajectories from neural policies.

6 Conclusion

We introduce SnareNet, a framework to enforce input-dependent constraints on the output of neural networks with feasibility guarantees. Our empirical evaluation demonstrates that SnareNet consistently produces feasible solutions with better objective values, even for *non-convex* constraints. Our key innovations include new perspective to reformulate an inequality feasibility problem as a more tractable equation solving problem, regularized Newton updates for efficient and stable convergence, and adaptive relaxation that progressively tightens the constraints.

AI USE STATEMENT

In this work, we used generative AI tools to polish the writing of the paper and to format the manuscript for the ICLR template. For the theoretical results, generative AI tools were used to motivate proof ideas, but all proofs were written by the authors. For the code, generative AI tools generated the plotting code for the figures and assisted in editing the SnareNet implementation under close supervision; the authors reviewed and checked every edit to the code of the core repair layer. We did not use generative AI tools for literature search, and the data generation code was adapted by the authors from existing repositories of prior works. We have reviewed all AI-assisted work, and we take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This paper develops learning-based methods for safety-critical and operationally constrained applications. It does not involve human subjects, personally identifiable data, or crowdsourcing, and we are not aware of ethical concerns that must be specifically highlighted here.

REPRODUCIBILITY STATEMENT

The experimental setup, datasets, and baselines are described in Section 5, Appendix B, and Appendix E; pseudocode for SnareNet is given in Algorithm 1; complete proofs of our theoretical results, with all assumptions stated, are in Appendix A. Anonymized source code is linked in Section 5.

REFERENCES

Aaron D Ames, Xiangru Xu, Jessy W Grizzle, and Paulo Tabuada. Control barrier function based quadratic programs for safety critical systems. *IEEE Transactions on Automatic Control*, 62(8): 3861–3876, 2016.

Aaron D. Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada. Control barrier functions: Theory and applications, 2019. URL https://arxiv.org/abs/1903.11199.

Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International conference on machine learning*, pp. 136–145. PMLR, 2017.

Christof Angermueller, Tanel Pärnamaa, Leopold Parts, and Oliver Stegle. Deep learning for computational biology. *Molecular systems biology*, 12(7):878, 2016.

Boris Bonev, Thorsten Kurth, Ankur Mahesh, Mauro Bisson, Jean Kossaifi, Karthik Kashinath, Anima Anandkumar, William D Collins, Michael S Pritchard, and Alexander Keller. Fourcastnet 3: A geometric approach to probabilistic machine-learning weather forecasting at scale. *arXiv preprint arXiv:2507.12144*, 2025.

Andrea Cini, Danilo Mandic, and Cesare Alippi. Graph-based time series clustering for end-to-end hierarchical forecasting. *arXiv preprint arXiv:2305.19183*, 2023.

Alp Dener, Marco Andres Miller, Randy Michael Churchill, Todd Munson, and Choong-Seock Chang. Training neural networks under physical constraints using a stochastic augmented lagrangian approach, 2020. URL https://arxiv.org/abs/2009.07330.

Shutong Ding, Yimiao Zhou, Ke Hu, Xi Yao, Junchi Yan, Xiaoying Tang, and Ye Shi. Diffusion-based learning framework for constrained nonconvex optimization with weighted bootstrapped refinement. *arXiv preprint arXiv:2502.10330*, 2025.

Priya L Donti, David Rolnick, and J Zico Kolter. Dc3: A learning method for optimization with hard constraints. *arXiv preprint arXiv:2104.12225*, 2021.

Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of field robotics*, 37(3):362–386, 2020.

Panagiotis D Grontas, Antonio Terpin, Efe C Balta, Raffaello D'Andrea, and John Lygeros. Pinet: Optimizing hard-constrained neural networks with orthogonal projection layers. *arXiv preprint arXiv:2508.10480*, 2025.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. URL `https://www.gurobi.com`.

Ashfaq Iftakher, Rahul Golder, Bimol Nath Roy, and MM Faruque Hasan. Physics-informed neural networks with hard nonlinear equality and inequality constraints. *Computers & Chemical Engineering*, pp. 109418, 2025.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *nature*, 596(7873):583–589, 2021.

Dmitrii Kochkov, Jamie A Smith, Ayya Alieva, Qing Wang, Michael P Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21):e2101784118, 2021.

Remi Lam, Alvaro Sanchez-Gonzalez, Matthew Willson, Peter Wirnsberger, Meire Fortunato, Ferran Alet, Suman Ravuri, Timo Ewalds, Zach Eaton-Rosen, Weihua Hu, et al. Learning skillful medium-range global weather forecasting. *Science*, 382(6677):1416–1421, 2023.

Giacomo Lastrucci and Artur M Schweidtmann. ENFORCE: Nonlinear constrained learning with adaptive-depth neural projection. *arXiv preprint arXiv:2502.06774*, 2025.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

Meiyi Li, Soheil Kolouri, and Javad Mohammadi. Learning to solve optimization problems with hard linear constraints. *IEEE Access*, 11:59995–60004, 2023.

Enming Liang, Minghua Chen, and Steven H. Low. Homeomorphic projection to ensure neural-network solution feasibility for constrained optimization. *Journal of Machine Learning Research*, 25(329):1–55, 2024. URL `http://jmlr.org/papers/v25/23-1577.html`.

Lu Lu, Raphael Pestourie, Wenjie Yao, Zhicheng Wang, Francesc Verdugo, and Steven G Johnson. Physics-informed neural networks with hard constraints for inverse design. *SIAM Journal on Scientific Computing*, 43(6):B1105–B1132, 2021.

Youngjae Min and Navid Azizan. Hardnet: Hard-constrained neural networks with universal approximation guarantees. *arXiv preprint arXiv:2410.10807*, 2024.

Pablo Márquez-Neila, Mathieu Salzmann, and Pascal Fua. Imposing hard constraints on deep networks: Promises and limitations, 2017. URL `https://arxiv.org/abs/1706.02025`.

Hoang T Nguyen and Priya L Donti. Fsnet: Feasibility-seeking neural network for constrained optimization with guarantees. *arXiv preprint arXiv:2506.00362*, 2025.

Xiang Pan. Deepopf: deep neural networks for optimal power flow. In *Proceedings of the 8th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, pp. 250–251, 2021.

Xiang Pan, Tianyu Zhao, Minghua Chen, and Shengyu Zhang. Deepopf: A deep neural network approach for security-constrained dc optimal power flow. *IEEE Transactions on Power Systems*, 36(3):1725–1735, 2020.

John Platt and Alan Barr. Constrained differential optimization. In *Neural information processing systems*, 1987.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

Syama Sundar Rangapuram, Lucien D Werner, Konstantinos Benidis, Pedro Mercado, Jan Gasthaus, and Tim Januschowski. End-to-end learning of coherent probabilistic forecasts for hierarchical time series. In *International Conference on Machine Learning*, pp. 8832–8843. PMLR, 2021.

Pratik Rathore, Weimu Lei, Zachary Frangella, Lu Lu, and Madeleine Udell. Challenges in training pinns: A loss landscape perspective. *arXiv preprint arXiv:2402.01868*, 2024.

Akylas Stratigakos, Dennis van Der Meer, Simon Camal, and Georges Kariniotakis. End-to-end learning for hierarchical forecasting of renewable energy production with missing values. In *2022 17th International Conference on Probabilistic Methods Applied to Power Systems (PMAPS)*, pp. 1–6. IEEE, 2022.

Manan Tayal, Bhavya Giri Goswami, Karthik Rajgopal, Rajpal Singh, Tejas Rao, Jishnu Keshavan, Pushpak Jagtap, and Shishir Kolathaya. A collision cone approach for control barrier functions, 2024. URL `https://arxiv.org/abs/2403.07043`.

Pascal Van Hentenryck. Optimization learning. *arXiv preprint arXiv:2501.03443*, 2025.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M Rehg, Byron Boots, and Evangelos A Theodorou. Information theoretic mpc for model-based reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 1714–1721. IEEE, 2017.

Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S Pappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530, 2018.

Shengwei Zhang and AG Constantinides. Lagrange programming neural networks. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 39(7):441–452, 1992.

A Convergence Analysis of SnareNet

The convergence analysis of SnareNet is based on the observation that SnareNet's repair layer update with $\lambda > 0$ is equivalent to preconditioned gradient descent (PGD) on the constraint violation function defined as

$$F(y) = \frac{1}{2}\|g(y) - \mathcal{P}_{\mathbf{B}(\ell,u)}(g(y))\|^2. \tag{9}$$

Appendix A.1 establishes the analytical properties of F that are crucial for the convergence analysis. The convergence analysis for linear and convex constraints follows in Appendix A.2 and Appendix A.3, respectively.

A.1 Analytical Properties of F

The function F is twice differentiable almost everywhere, despite the presence of the seemingly non-smooth projection operator $\mathcal{P}_{\mathbf{B}(\ell,u)}$. Moreover, when $g(y) = Ay$ is linear with $A \in \mathbb{R}^{m\times n}$, the function F is convex. See Proposition A.1.

Proposition A.1 (Differentiability & Convexity)**.** *Let $g : \mathbb{R}^n \to \mathbb{R}^m$ be a twice differentiable function. Then $F(y)$ defined in* (9) *is twice differentiable almost everywhere and its gradient is given by*

$$\nabla F(y) = J_g(y)^\top (g(y) - \mathcal{P}_{\mathbf{B}(\ell,u)}(g(y))). \tag{10}$$

In particular, if $g = A \in \mathbb{R}^{m\times n}$ is linear, the constraint violation function $F(y)$ is convex.

Proof. Observe that $F(y) = R(g(y))$ a composition of g and a separable function $R : \mathbb{R}^m \to \mathbb{R}$ defined by

$$R(z) := \frac{1}{2}\|z - \mathcal{P}_{\mathbf{B}(\ell,u)}(z)\|^2 = \frac{1}{2}\sum_{i=1}^{m}(z_i - \mathcal{P}_{[\ell_i,u_i]}(z_i))^2, \tag{11}$$

where $\mathcal{P}_{[\ell_i,u_i]}$ being the one-dimensional projection operator onto the interval $[\ell_i, u_i]$.

- **Almost everywhere twice differentiability.** It suffices to show that $r_{[\ell,u]}(a) := \frac{1}{2}(a - \mathcal{P}_{[\ell,u]}(a))^2$ with $a \in \mathbb{R}$ is twice differentiable almost everywhere since $R(z) = \sum_{i=1}^{m} r_{[\ell_i,u_i]}(z_i)$ is the sum of such functions and $F(y) = R(g(y))$ is a composition of R and a twice differentiable g. The piecewise definition of $r(a)$ and its derivatives $r'(a)$ and $r''(a)$ are

$$r_{[\ell,u]}(a) = \begin{cases} \frac{1}{2}(a-\ell)^2, & \text{if } a < \ell; \\ 0, & \text{if } \ell \le a \le u; \\ \frac{1}{2}(a-u)^2, & \text{if } a > u, \end{cases} \qquad r'_{[\ell,u]}(a) = \begin{cases} a-\ell, & \text{if } a < \ell; \\ 0, & \text{if } \ell \le a \le u; \\ a-u, & \text{if } a > u, \end{cases}$$

$$r''_{[\ell,u]}(a) = \begin{cases} 0, & \text{if } \ell < a < u; \\ 1, & \text{if } a < \ell \text{ or } a > u. \end{cases}$$

  Note that $r'_{[\ell,u]}(a) = a - \mathcal{P}_{[\ell,u]}(a)$ is continuous everywhere, and the second derivative $r''_{[\ell,u]}(a)$ is well-defined for all $a \in \mathbb{R}$ except at the two points $a = \ell$ and $a = u$. Thus, $r_{[\ell,u]}(a)$ is twice differentiable almost everywhere, and so is $F(y)$.

- **Gradient expression.** Note that $\nabla R(z) = z - \mathcal{P}_{\mathbf{B}(\ell,u)}(z)$ since $r'_{[\ell,u]}(a) = a - \mathcal{P}_{[\ell,u]}(a)$. The gradient of $F(y)$ follows by the chain rule:

$$\nabla F(y) = J_g(y)^\top \nabla R(g(y)) = J_g(y)^\top (g(y) - \mathcal{P}_{\mathbf{B}(\ell,u)}(g(y))).$$

- **Convexity of F when g is linear.** Observe that $R(z) = \sum_{i=1}^{m} r_{[\ell_i,u_i]}(z_i)$ is a sum of convex functions since each $r_{[\ell_i,u_i]}$ is convex. Thus, R is convex. As $F(y) = R(Ay)$ is a composition of a convex function R and a linear function Ay, F is convex.

□

Using expression (10) of $\nabla F(y)$, SnareNet's repair update (7) with $\lambda > 0$ can be expressed as

$$\begin{aligned} y^{k+1} &= y^k - (J_g(y^k)^\top J_g(y^k) + \lambda I)^{-1} J_g(y^k)^\top (g(y^k) - \mathcal{P}_{\mathbf{B}(\ell,u)}(g(y^k))) \\ &= y^k - (J_g(y^k)^\top J_g(y^k) + \lambda I)^{-1} \nabla F(y^k). \end{aligned} \tag{12}$$

Equation (12) can be interpreted as a PGD step on $F(y)$ with preconditioner $P_k := (J_g(y^k)^\top J_g(y^k) + \lambda I)^{-1}$. To show the convergence under the choice P_k, we first establish an upper bound on $F(y)$ in Lemma A.2 that is analogous to the quadratic upper bound for smooth functions, and then show that F satisfies the Polyak-Lojasiewicz (PL) condition under the preconditioned norm when $J_g(y)$ has full-row rank in Lemma A.3. Finally, we prove the convergence of SnareNet in Theorem 4.1 and Theorem 4.2 for linear and nonlinear constraints, respectively.

Lemma A.2 (Upper bound on F)**.** *Let $g : \mathbb{R}^n \to \mathbb{R}^m$ be a twice differentiable function with an L_g-Lipschitz continuous Jacobian. Then for any $x, y \in \mathbb{R}^n$, the constraint violation function $F(y)$ defined in* (9) *satisfies the bound*

$$\begin{aligned} F(y) \le F(x) + \langle \nabla F(x), y - x\rangle + \frac{1}{2}(y-x)^\top J_g(x)^\top J_g(x)(y-x) \\ + \frac{L_g}{2}\left[\|\nabla R(g(x))\| + \|J_g(x)(y-x)\| + \frac{L_g}{4}\|y-x\|^2\right]\|y-x\|^2. \end{aligned} \tag{13}$$

In particular, if $g(y) = Ay$ is linear with $A \in \mathbb{R}^{m\times n}$, the bound in (13) *reduces to*

$$F(y) \le F(x) + \langle \nabla F(x), y - x\rangle + \frac{1}{2}(y-x)^\top A^\top A(y-x). \tag{14}$$

Proof. We first show that $R(z)$ defined in (11) is 1-smooth, or equivalently, its gradient $\nabla R(z) = z - \mathcal{P}_{\mathbf{B}(\ell,u)}(z)$ is 1-Lipschitz continuous: for any $z, z' \in \mathbb{R}^m$, we have

$$\begin{aligned} \|\nabla R(z) - \nabla R(z')\|^2 &= \|(z - \mathcal{P}_{\mathbf{B}(\ell,u)}(z)) - (z' - \mathcal{P}_{\mathbf{B}(\ell,u)}(z'))\|^2 \\ &= \|z - z'\|^2 - 2\langle \mathcal{P}_{\mathbf{B}(\ell,u)}(z) - \mathcal{P}_{\mathbf{B}(\ell,u)}(z'), z - z'\rangle + \|\mathcal{P}_{\mathbf{B}(\ell,u)}(z) - \mathcal{P}_{\mathbf{B}(\ell,u)}(z')\|^2 \\ &\le \|z - z'\|^2 - \|\mathcal{P}_{\mathbf{B}(\ell,u)}(z) - \mathcal{P}_{\mathbf{B}(\ell,u)}(z')\|^2 \\ &\qquad\qquad \text{(by firmly non-expansiveness of } \mathcal{P}_{\mathbf{B}(\ell,u)}) \\ &\le \|z - z'\|^2. \end{aligned}$$

Apply the 1-smoothness of $R(z)$ to upper bound $F(y) = R(g(y))$ by

$$\begin{aligned} F(y) = R(g(y)) &\le R(g(x)) + \langle \nabla R(g(x)), g(y) - g(x)\rangle + \frac{1}{2}\|g(y) - g(x)\|^2 \\ &= F(x) + \langle \nabla R(g(x)), g(y) - g(x)\rangle + \frac{1}{2}\|g(y) - g(x)\|^2. \end{aligned} \tag{15}$$

Let $e_x(y) := g(y) - g(x) - J_g(x)(y-x)$ denote the error of linear approximation. Using the fundamental theorem of calculus and the L_g-Lipschitz continuity of J_g, the error can be bounded by

$$\|e_x(y)\| = \left\|\int_0^1 [J_g(x + t(y-x)) - J_g(x)](y-x)\,dt\right\| \le \int_0^1 L_g t\|y-x\|^2\,dt = \frac{L_g}{2}\|y-x\|^2. \tag{16}$$

Cauchy-Schwartz inequality and (16) imply

$$\begin{aligned} \langle \nabla R(g(x)), g(y) - g(x)\rangle &= \langle \nabla R(g(x)), g(y) - g(x) - J_g(x)(y-x)\rangle + \langle \nabla R(g(x)), J_g(x)(y-x)\rangle \\ &\le \|\nabla R(g(x))\|\|g(y) - g(x) - J_g(x)(y-x)\| + \langle J_g(x)^\top \nabla R(g(x)), y - x\rangle \\ &\le \frac{L_g\|\nabla R(g(x))\|}{2}\|y-x\|^2 + \langle \nabla F(x), y - x\rangle. \end{aligned} \tag{17}$$

Equation (16) also implies

$$\begin{aligned} \|g(y) - g(x)\|^2 &= \|e_x(y) + J_g(x)(y-x)\|^2 \\ &= \|J_g(x)(y-x)\|^2 + 2\langle J_g(x)(y-x), e_x(y)\rangle + \|e_x(y)\|^2 \\ &\le (y-x)^\top J_g(x)^\top J_g(x)(y-x) + L_g\|J_g(x)(y-x)\|\|y-x\|^2 + \frac{L_g^2}{4}\|y-x\|^4. \end{aligned} \tag{18}$$

Combine (15), (17), and (18) to conclude (13). When $g(y) = A(y)$ is linear, the Jacobian of g is constant $J_g(y) = A$ for all $y \in \mathbb{R}^n$ and thus $L_g = 0$. The bound reduces to (14). □

Lemma A.3 (PL-condition in preconditioned norm)**.** *Suppose $J_g(y)$ has full-row rank with minimum singular value $\sigma_{\min}(J_g(y)) > 0$ and $\mathcal{C} := \{y \in \mathbb{R}^n \mid \ell \le g(y) \le u\}$ is non-empty. The constraint violation function $F(y)$ defined in* (9) *satisfies the Polyak-Lojasiewicz (PL) condition under the norm $\|\cdot\|_{(J_g(y)^\top J_g(y) + \lambda I)^{-1}}$:*

$$\frac{1}{2}\|\nabla F(y)\|^2_{(J_g(y)^\top J_g(y)+\lambda I)^{-1}} \ge \frac{\sigma^2_{\min}(J_g(y))}{\sigma^2_{\min}(J_g(y)) + \lambda}[F(y) - \min_{y\in\mathbb{R}^n} F(y)] = \frac{\sigma^2_{\min}(J_g(y))}{\sigma^2_{\min}(J_g(y)) + \lambda}F(y).$$

Proof. Let $d(y) := g(y) - \mathcal{P}_{\mathbf{B}}(g(y))$. Since $\nabla F(y) = J_g(y)^\top d(y)$, the gradient under preconditioned norm is

$$\|\nabla F(y)\|^2_{(J_g(y)^\top J_g(y)+\lambda I)^{-1}} = d(y)^\top J_g(y)[J_g(y)^\top J_g(y) + \lambda I]^{-1} J_g(y)^\top d(y).$$

Let $\sigma_i(J_g(y))$ denote the i-th singular value of $J_g(y)$. Then the singular values of $J_g(y)[J_g(y)^\top J_g(y) + \lambda I]^{-1} J_g(y)^\top$ are $\frac{\sigma_i^2(J_g(y))}{\sigma_i^2(J_g(y))+\lambda}$ for $i = 1, \ldots, m$ with minimum $\frac{\sigma^2_{\min}(J_g(y))}{\sigma^2_{\min}(J_g(y))+\lambda}$. Since $F(y) = \frac{1}{2}\|d(y)\|^2$ and non-empty $\mathcal{C}$ guarantees $\min_{y\in\mathbb{R}^n} F(y) = 0$, we conclude

$$\frac{1}{2}\|\nabla F(y)\|^2_{(J_g(y)^\top J_g(y)+\lambda I)^{-1}} \geq \frac{\sigma^2_{\min}(J_g(y))}{2(\sigma^2_{\min}(J_g(y)) + \lambda)}\|d(y)\|^2 = \frac{\sigma^2_{\min}(J_g(y))}{\sigma^2_{\min}(J_g(y)) + \lambda} F(y).$$

□

A.2 Proof of Theorem 4.1

Proof. Denote $H_\lambda := A^\top A + \lambda I$. Then $y^{k+1} = y^k - H_\lambda^{-1}\nabla F(y^k)$ and $A^\top A \prec H_\lambda$. Substitute $y = y^{k+1}$ and $x = y^k$ into (14) to obtain

$$\begin{aligned} F(y^{k+1}) &\leq F(y^k) + \langle \nabla F(y^k), y^{k+1} - y^k\rangle + \frac{1}{2}(y^{k+1} - y^k)^\top A^\top A(y^{k+1} - y^k) \\ &\leq F(y^k) + \langle \nabla F(y^k), y^{k+1} - y^k\rangle + \frac{1}{2}(y^{k+1} - y^k)^\top H_\lambda(y^{k+1} - y^k) \\ &= F(y^k) - \nabla F(y^k)^\top H_\lambda^{-1}\nabla F(y^k) + \frac{1}{2}\nabla F(y^k)^\top H_\lambda^{-1}\nabla F(y^k) \\ &\leq F(y^k) - \frac{1}{2}\|\nabla F(y^k)\|^2_{H_\lambda^{-1}}, \end{aligned} \tag{19}$$

proving the descent property for SnareNet's update. Since A has full row rank, Lemma A.3 guarantees $F(y)$ satisfies PL-condition and thus

$$F(y^{k+1}) \leq F(y^k) - \frac{\sigma^2_{\min}}{\sigma^2_{\min} + \lambda} F(y^k) = \left(1 - \frac{\sigma^2_{\min}}{\sigma^2_{\min} + \lambda}\right) F(y^k).$$

□

A.3 Proof of Theorem 4.2

Proof. Observe that F is convex under the convex constraints represented by $\mathcal{C} := \{y \in \mathbb{R}^n \mid h(y) \leq 0,\ \ell_A \leq Ay \leq u_A\}$ since F takes the form

$$F(y) := \frac{1}{2}\|\max(0, h(y))\|^2 + \frac{1}{2}\|Ay - \mathcal{P}_{\mathbf{B}(\ell_A, u_A)}(Ay)\|^2$$

and both $\|\max(0, h(y))\|^2$ and $\|Ay - \mathcal{P}_{\mathbf{B}(\ell_A,u_A)}(Ay)\|^2$ are convex functions of y. Using SnareNet's notation, the constraint function g, the bounds ℓ, u, and the Jacobian J_g are (defined) as

$$g(y) := \begin{bmatrix} h(y) \\ Ay \end{bmatrix},\ \ell := \begin{bmatrix} -\infty \\ \ell_A \end{bmatrix},\ u := \begin{bmatrix} 0 \\ u_A \end{bmatrix},\ J_g(y) := \begin{bmatrix} J_h(y) \\ A \end{bmatrix}.$$

The Jacobian of g is L-Lipschitz continuous since $J_h(y)$ is L-Lipschitz continuous and A is constant.

Now, we show that our choice of λ guarantees $y^k \in \mathcal{D}$ for all $k \geq 0$ and also the descent property $F(y^{k+1}) \leq F(y^k)$. We prove by induction. Note that $y^0 \in \mathcal{D}$ by definition. Suppose $y^k \in \mathcal{D}$ by induction hypothesis. Substitute $y = y^{k+1}$ and $x = y^k$ into (13) to obtain

$$\begin{aligned} F(y^{k+1}) \leq F(y^k) + \langle \nabla F(y^k), y^{k+1} - y^k\rangle + \tfrac{1}{2}(y^{k+1} - y^k)^\top J_g(y^k)^\top J_g(y^k)(y^{k+1} - y^k) \\ + \tfrac{L_g}{2}\left[\|\nabla R(g(y^k))\| + \|J_g(y^k)(y^{k+1} - y^k)\| + \tfrac{L_g}{4}\|y^{k+1} - y^k\|^2\right]\|y^{k+1} - y^k\|^2. \end{aligned} \tag{20}$$

Since $y^k \in \mathcal{D}$, we have $\|\nabla R(g(y^k))\| = \|g(y^k) - \mathcal{P}_{\mathbf{B}(\ell,u)}(g(y^k))\| = \sqrt{2F(y^k)} \leq \sqrt{2F(y^0)} =: M$ and thus

$$\|J_g(y^k)(y^{k+1} - y^k)\| \leq \left\|J_g(y^k)(J_g(y^k)^\top J_g(y^k) + \lambda I)^{-1} J_g(y^k)^\top\right\| \left\|\nabla R(g(y^k))\right\| \leq M, \quad (21)$$

$$\|y^{k+1} - y^k\|^2 = \left\|(J_g(y^k)^\top J_g(y^k) + \lambda I)^{-1} J_g(y^k)^\top\right\|^2 \left\|\nabla R(g(y^k))\right\|^2 \leq \tfrac{M^2}{4\lambda}. \quad (22)$$

Let $H_k := J_g(y^k)^\top J_g(y^k) + \lambda I$. Then SnareNet's update reads $y^{k+1} = y^k - H_k^{-1}\nabla F(y^k)$ and (20) becomes

$$\begin{aligned} F(y^{k+1}) &\leq F(y^k) + \langle \nabla F(y^k), y^{k+1} - y^k\rangle + \tfrac{1}{2}(y^{k+1} - y^k)^\top J_g(y^k)^\top J_g(y^k)(y^{k+1} - y^k) \\ &\quad + \left(L_g M + \tfrac{L_g^2 M^2}{32\lambda}\right)\|y^{k+1} - y^k\|^2 \qquad \text{(by (21) and (22))} \\ &\leq F(y^k) + \langle \nabla F(y^k), y^{k+1} - y^k\rangle + \tfrac{1}{2}(y^{k+1} - y^k)^\top H_k(y^{k+1} - y^k) \\ &\qquad \text{(by } \lambda^2 - 2L_g M\lambda - \tfrac{L_g^2 M^2}{16} \geq 0) \\ &= F(y^k) - \tfrac{1}{2}\|\nabla F(y^k)\|^2_{H_k^{-1}}, \qquad (23) \end{aligned}$$

which guarantees $F(y^{k+1}) \leq F(y^k)$ and thus $y^{k+1} \in \mathcal{D}$. The claim that $y^k \in \mathcal{D}$ for all $k \geq 0$ is proved by induction and the descent property is established.

Next, we prove the sublinear convergence using the convexity of F and the descent property (23). Since F is convex and the feasible set $\mathcal{C}$ is non-empty, we have $\min_{y\in\mathbb{R}^n} F(y) = 0$ and

$$F(y^k) = F(y^k) - \min_{y\in\mathbb{R}^n} F(y) \leq \|\nabla F(y^k)\|_{H_k^{-1}} \|y^k - \mathcal{P}_{\mathcal{C}}(y^k)\|_{H_k}. \quad (24)$$

To bound $\|y^k - \mathcal{P}_{\mathcal{C}}(y^k)\|_{H_k}$, observe that $\|H_k\| \leq \sigma^2_{\max}(J_g(y^k)) + \lambda \leq \sigma^2_{\max}(J_h(y^k)) + \sigma^2_{\max}(A) + \lambda \leq \sigma^2 + \lambda$ since $y^k \in \mathcal{D}$, and thus

$$\|y^k - \mathcal{P}_{\mathcal{C}}(y^k)\|^2_{H_k} \leq \|H_k\|\|y^k - \mathcal{P}_{\mathcal{C}}(y^k)\|^2 \leq (\sigma^2 + \lambda)D^2. \quad (25)$$

Combine (23), (24), and (25) to obtain

$$F(y^{k+1}) \leq F(y^k) - \frac{F(y^k)^2}{2D^2(\sigma^2 + \lambda)}. \quad (26)$$

If $F(y^{k+1}) = 0$, the desired bound follows immediately. Otherwise, (26) implies

$$\frac{1}{F(y^{k+1})} - \frac{1}{F(y^k)} \geq \frac{F(y^k) - F(y^{k+1})}{F(y^k)F(y^{k+1})} \geq \frac{F(y^k)}{2D^2(\sigma^2 + \lambda)F(y^{k+1})} \geq \frac{1}{2D^2(\sigma^2 + \lambda)}.$$

The telescoping sum yields

$$\frac{1}{F(y^{k+1})} \geq \frac{1}{F(y^0)} + \frac{k+1}{2D^2(\sigma^2 + \lambda)},$$

which implies the desired sublinear convergence bound. □

B Details of Baselines

This section provides the details of each baselines we compared to in Section 5.

- DC3 (Donti et al., 2021) strictly enforces equality constraints by predicting free variables and solving dependent variables from equality constraints. The inequality constraint violation are iteratively reduced by a fixed number of gradient descent steps on the free variables.
- HardNet (Min & Azizan, 2024) uses soft-epochs and strictly enforces linear constraints by (4). We compare to HardNet on NCLPs.
- OptNet (Amos & Kolter, 2017) appends a differentiable convex projection problem (3) as a repair layer. We compare to OptNet on convex constrained problems: NCLPs and convex QCQPs.
- HProj (Liang et al., 2024) is a post-processing method with two stages training: the first stage trains a homeomorphic map and the second stage trains a soft-constraint NN. At inference time, HProj uses the learned homeomorphic map and bisection to project onto the feasible set.

C TABLES FOR ALL TEST RESULTS IN SECTION 5.1

This section includes all the evaluation results on test instances for experiments in Section 5.1. For ease of navigation, Table 3 summarizes the correspondence between figures presented in the main text and their associated results tables organized by problem class. Columns "# of Ineq. Violations" and "# of Eq. Violations" are counted with threshold 10^{-4}.

Table 3: Correspondence between tables, problem classes, and figures in main paper.

Tables	Problem Class	Figures
Table 4	NCLP	Figure 3
Table 5	Convex QCQP	Figure 3
Table 6	Non-Convex QCQP	Figure 3
Table 7	NCLP	Figure 4
Table 8	Convex QCQP (10 inequality constraints)	Figure 5
Table 9	Convex QCQP (50 inequality constraints)	Figure 5
Table 10	Convex QCQP (100 inequality constraints)	Figure 5

Table 4: Evaluation metrics on the NCLPs test set.

Method	Max Opt. Gap	GMean Opt. Gap	Max Ineq. Error	GMean Ineq. Error	# Ineq Violations	Max Eq. Error	GMean Eq. Error	# Eq Violations	Test Time (s)
DC3	8.81 ± 5.68	2.68 ± 1.35	$(7.16 \pm 16.00) \times 10^{-3}$	$(1.00 \pm 0.00) \times 10^{-16}$	$(2.40 \pm 5.37) \times 10^{-4}$	$(6.03 \pm 1.61) \times 10^{-14}$	$(6.46 \pm 2.27) \times 10^{-15}$	0.00	0.186 ± 0.074
HardNet	$(1.02 \pm 0.47) \times 10^{1}$	$(9.17 \pm 0.99) \times 10^{-1}$	$(7.09 \pm 1.19) \times 10^{-14}$	$(1.10 \pm 0.01) \times 10^{-16}$	0.00	$(1.13 \pm 0.14) \times 10^{-13}$	$(1.04 \pm 0.03) \times 10^{-14}$	0.00	0.116 ± 0.035
HProj	$(5.97 \pm 12.04) \times 10^{2}$	3.72 ± 7.12	0.00	0.00	0.00	$(2.24 \pm 4.49) \times 10^{-12}$	$(3.12 \pm 5.58) \times 10^{-14}$	0.00	0.049 ± 0.029
OptNet	$(6.83 \pm 0.70) \times 10^{-1}$	$(2.40 \pm 0.03) \times 10^{-1}$	0.00	0.00	0.00	$(2.24 \pm 0.57) \times 10^{-11}$	$(3.39 \pm 0.16) \times 10^{-13}$	0.00	1.634 ± 0.249
SnareNet ($\texttt{tol} = 10^{-4}$)	$(4.57 \pm 0.62) \times 10^{-1}$	$(8.03 \pm 0.58) \times 10^{-2}$	$(5.29 \pm 0.67) \times 10^{-5}$	$(1.02 \pm 0.40) \times 10^{-15}$	0.00	$(8.34 \pm 1.19) \times 10^{-5}$	$(2.08 \pm 1.01) \times 10^{-6}$	0.00	0.501 ± 0.076
SnareNet ($\texttt{tol} = 10^{-6}$)	$(4.58 \pm 0.64) \times 10^{-1}$	$(8.03 \pm 0.58) \times 10^{-2}$	$(5.54 \pm 0.63) \times 10^{-7}$	$(6.34 \pm 2.06) \times 10^{-16}$	0.00	$(8.72 \pm 0.99) \times 10^{-7}$	$(1.49 \pm 0.78) \times 10^{-8}$	0.00	0.869 ± 0.112
SnareNet ($\texttt{tol} = 10^{-8}$)	$(4.57 \pm 0.64) \times 10^{-1}$	$(8.03 \pm 0.58) \times 10^{-2}$	$(5.36 \pm 1.09) \times 10^{-9}$	$(3.91 \pm 0.94) \times 10^{-16}$	0.00	$(8.37 \pm 1.48) \times 10^{-9}$	$(9.11 \pm 2.89) \times 10^{-11}$	0.00	0.972 ± 0.152
SnareNet ($\texttt{tol} = 10^{-10}$)	$(4.57 \pm 0.64) \times 10^{-1}$	$(8.03 \pm 0.58) \times 10^{-2}$	$(5.25 \pm 1.43) \times 10^{-11}$	$(2.44 \pm 0.43) \times 10^{-16}$	0.00	$(8.00 \pm 1.66) \times 10^{-11}$	$(6.02 \pm 2.51) \times 10^{-13}$	0.00	1.379 ± 0.272
SnareNet ($\texttt{tol} = 10^{-12}$)	$(4.56 \pm 0.64) \times 10^{-1}$	$(8.03 \pm 0.58) \times 10^{-2}$	$(5.05 \pm 1.16) \times 10^{-13}$	$(1.51 \pm 0.17) \times 10^{-16}$	0.00	$(7.72 \pm 1.52) \times 10^{-13}$	$(6.42 \pm 2.35) \times 10^{-15}$	0.00	1.646 ± 0.605

Table 5: Evaluation metrics on the convex QCQPs test set.

Method	Max Opt. Gap	GMean Opt. Gap	Max Ineq. Error	GMean Ineq. Error	# Ineq Violations	Max Eq. Error	GMean Eq. Error	# Eq Violations	Test Time (s)
DC3	$(1.50 \pm 1.10) \times 10^{1}$	9.60 ± 8.14	2.89 ± 4.08	$(2.22 \pm 2.16) \times 10^{-16}$	0.70 ± 1.08	$(4.62 \pm 4.23) \times 10^{-14}$	$(4.12 \pm 2.79) \times 10^{-15}$	0.00	0.191 ± 0.088
HProj	$(7.78 \pm 17.18) \times 10^{1}$	$(3.41 \pm 7.57) \times 10^{1}$	$(3.58 \pm 8.01) \times 10^{2}$	$(1.10 \pm 2.47) \times 10^{-13}$	10.00 ± 22.36	$(2.01 \pm 4.41) \times 10^{-12}$	$(2.77 \pm 6.08) \times 10^{-13}$	0.00	0.260 ± 0.374
OptNet	$(1.21 \pm 0.10) \times 10^{-1}$	$(4.15 \pm 0.01) \times 10^{-2}$	$(8.09 \pm 2.07) \times 10^{-8}$	$(2.10 \pm 0.03) \times 10^{-14}$	0.00	$(1.25 \pm 0.13) \times 10^{-11}$	$(2.51 \pm 0.16) \times 10^{-13}$	0.00	3.044 ± 0.203
SnareNet ($\texttt{tol} = 10^{-4}$)	$(1.24 \pm 0.06) \times 10^{-1}$	$(2.60 \pm 0.26) \times 10^{-2}$	$(8.63 \pm 0.94) \times 10^{-5}$	$(7.74 \pm 1.97) \times 10^{-15}$	0.00	$(8.33 \pm 0.87) \times 10^{-5}$	$(1.93 \pm 0.45) \times 10^{-6}$	0.00	0.807 ± 0.093
SnareNet ($\texttt{tol} = 10^{-6}$)	$(1.25 \pm 0.06) \times 10^{-1}$	$(2.60 \pm 0.27) \times 10^{-2}$	$(7.91 \pm 0.98) \times 10^{-7}$	$(2.68 \pm 0.64) \times 10^{-15}$	0.00	$(7.64 \pm 0.97) \times 10^{-7}$	$(5.92 \pm 2.64) \times 10^{-9}$	0.00	1.356 ± 0.135
SnareNet ($\texttt{tol} = 10^{-8}$)	$(1.24 \pm 0.07) \times 10^{-1}$	$(2.60 \pm 0.26) \times 10^{-2}$	$(8.15 \pm 0.85) \times 10^{-9}$	$(9.52 \pm 1.98) \times 10^{-16}$	0.00	$(7.88 \pm 0.82) \times 10^{-9}$	$(2.13 \pm 1.23) \times 10^{-11}$	0.00	1.780 ± 0.183
SnareNet ($\texttt{tol} = 10^{-10}$)	$(1.25 \pm 0.07) \times 10^{-1}$	$(2.60 \pm 0.26) \times 10^{-2}$	$(7.56 \pm 0.70) \times 10^{-11}$	$(3.35 \pm 0.70) \times 10^{-16}$	0.00	$(7.31 \pm 0.72) \times 10^{-11}$	$(7.35 \pm 5.48) \times 10^{-14}$	0.00	2.231 ± 0.454
SnareNet ($\texttt{tol} = 10^{-12}$)	$(1.24 \pm 0.06) \times 10^{-1}$	$(2.60 \pm 0.26) \times 10^{-2}$	$(7.38 \pm 0.99) \times 10^{-13}$	$(1.36 \pm 0.17) \times 10^{-16}$	0.00	$(7.06 \pm 0.95) \times 10^{-13}$	$(9.93 \pm 3.87) \times 10^{-16}$	0.00	2.603 ± 0.956

Table 6: Evaluation metrics on the non-convex QCQPs test set.

Method	Max Obj. Value	Mean Obj. Value	Max Ineq. Error	GMean Ineq. Error	# Ineq Violations	Max Eq. Error	GMean Eq. Error	# Eq Violations	Test Time (s)
DC3	-12.70 ± 9.39	-15.25 ± 8.99	$(6.66 \pm 10.43) \times 10^{-1}$	$(1.09 \pm 0.09) \times 10^{-16}$	0.12 ± 0.12	$(7.71 \pm 2.16) \times 10^{-15}$	$(7.28 \pm 0.36) \times 10^{-16}$	0.00	0.172 ± 0.059
HProj	12.86 ± 8.93	-0.47 ± 9.22	$(1.69 \pm 0.70) \times 10^{1}$	$(3.55 \pm 6.15) \times 10^{-11}$	5.85 ± 8.50	$(1.05 \pm 0.70) \times 10^{-14}$	$(1.05 \pm 0.60) \times 10^{-15}$	0.00	0.438 ± 0.139
SnareNet ($\texttt{tol} = 10^{-4}$)	-19.98 ± 1.89	-20.61 ± 1.81	$(2.40 \pm 1.89) \times 10^{-6}$	$(6.19 \pm 10.78) \times 10^{-10}$	0.00	$(1.07 \pm 0.84) \times 10^{-6}$	$(3.87 \pm 2.92) \times 10^{-7}$	0.00	0.124 ± 0.059
SnareNet ($\texttt{tol} = 10^{-6}$)	-21.16 ± 1.37	-21.72 ± 1.32	$(1.10 \pm 2.27) \times 10^{-7}$	$(2.81 \pm 5.40) \times 10^{-12}$	0.00	$(4.48 \pm 9.07) \times 10^{-8}$	$(2.14 \pm 4.40) \times 10^{-8}$	0.00	0.124 ± 0.020
SnareNet ($\texttt{tol} = 10^{-8}$)	-20.61 ± 1.98	-21.18 ± 1.91	$(3.68 \pm 4.05) \times 10^{-9}$	$(2.25 \pm 1.54) \times 10^{-13}$	0.00	$(1.99 \pm 2.19) \times 10^{-9}$	$(7.84 \pm 8.85) \times 10^{-10}$	0.00	0.141 ± 0.028
SnareNet ($\texttt{tol} = 10^{-10}$)	-13.66 ± 16.95	-15.89 ± 13.23	$(1.11 \pm 2.49) \times 10^{-5}$	$(1.05 \pm 0.95) \times 10^{-14}$	0.00	$(6.28 \pm 14.03) \times 10^{-6}$	$(2.55 \pm 2.97) \times 10^{-12}$	0.00	0.616 ± 1.049
SnareNet ($\texttt{tol} = 10^{-12}$)	-13.01 ± 18.41	-17.40 ± 9.87	$(1.47 \pm 3.28) \times 10^{-6}$	$(6.86 \pm 3.82) \times 10^{-16}$	0.00	$(7.90 \pm 17.66) \times 10^{-7}$	$(8.91 \pm 8.89) \times 10^{-15}$	0.00	0.651 ± 1.117

Table 7: Evaluation metrics on the NCLP test set.

Method	Obj. Value	Max Opt. Gap	GMean Opt. Gap	Max Ineq. Error	GMean Ineq. Error	# Ineq Violations	Max Eq. Error	GMean Eq. Error	# Eq Violations	Test Time (s)
SnareNet ($\lambda = 0.001$)	-11.19 ± 0.29	2.92 ± 1.73	$(4.28 \pm 2.66) \times 10^{-1}$	$(4.07 \pm 1.46) \times 10^{-9}$	$(1.72 \pm 0.14) \times 10^{-16}$	0.00	$(6.57 \pm 2.36) \times 10^{-9}$	$(1.87 \pm 0.97) \times 10^{-10}$	0.00	0.28 ± 0.12
SnareNet ($\lambda = 0.01$)	-11.59 ± 0.01	$(4.28 \pm 0.46) \times 10^{-1}$	$(7.90 \pm 1.00) \times 10^{-2}$	$(5.35 \pm 0.88) \times 10^{-9}$	$(2.71 \pm 0.51) \times 10^{-16}$	0.00	$(7.88 \pm 1.14) \times 10^{-9}$	$(7.74 \pm 3.91) \times 10^{-11}$	0.00	0.76 ± 0.74
SnareNet ($\lambda = 0.1$)	-11.64 ± 0.00	$(2.18 \pm 0.12) \times 10^{-1}$	$(3.86 \pm 0.22) \times 10^{-2}$	$(1.40 \pm 0.22) \times 10^{-8}$	$(3.58 \pm 0.04) \times 10^{-16}$	0.00	$(2.08 \pm 0.33) \times 10^{-8}$	$(4.29 \pm 0.82) \times 10^{-10}$	0.00	0.61 ± 0.26
SnareNet ($\lambda = 10$)	-11.67 ± 0.00	$(1.87 \pm 0.16) \times 10^{-2}$	$(4.29 \pm 0.06) \times 10^{-3}$	$(8.65 \pm 0.46) \times 10^{-8}$	$(6.38 \pm 0.16) \times 10^{-16}$	0.00	$(1.20 \pm 0.06) \times 10^{-7}$	$(4.98 \pm 0.61) \times 10^{-9}$	0.00	2.19 ± 1.24
SnareNet ($\lambda = 1$)	-11.66 ± 0.00	$(6.99 \pm 0.74) \times 10^{-2}$	$(1.57 \pm 0.05) \times 10^{-2}$	$(1.96 \pm 0.08) \times 10^{-8}$	$(5.01 \pm 0.09) \times 10^{-16}$	0.00	$(3.21 \pm 0.13) \times 10^{-8}$	$(6.27 \pm 0.93) \times 10^{-10}$	0.00	1.31 ± 0.32
SnareNet ($\lambda = 5$)	-11.67 ± 0.00	$(2.74 \pm 0.08) \times 10^{-2}$	$(6.34 \pm 0.06) \times 10^{-3}$	$(5.07 \pm 0.25) \times 10^{-8}$	$(6.78 \pm 0.12) \times 10^{-16}$	0.00	$(7.52 \pm 0.39) \times 10^{-8}$	$(3.19 \pm 0.24) \times 10^{-9}$	0.00	2.22 ± 1.17

Table 8: Evaluation metrics on convex QCQP with 10 inequality constraints test set.

Method	Max Opt. Gap	GMean Opt. Gap	GMean Ineq. Error	# Ineq Violations	Max Eq. Error	GMean Eq. Error	# Eq Violations	Test Time (s)
Optimizer (Gurobi)	1.42×10^{-14}	4.48×10^{-16}	0.00	0.00	5.43×10^{-10}	1.06×10^{-14}	0.00	428.44
DC3	$(7.96 \pm 11.33) \times 10^{1}$	$(2.21 \pm 1.94) \times 10^{1}$	$(3.52 \pm 7.87) \times 10^{-10}$	0.97 ± 1.88	$(1.48 \pm 1.11) \times 10^{-13}$	$(1.40 \pm 0.94) \times 10^{-14}$	0.00	0.08 ± 0.07
HProj	$(2.40 \pm 1.99) \times 10^{-1}$	$(1.60 \pm 1.42) \times 10^{-1}$	0.00	0.00	$(1.08 \pm 0.92) \times 10^{-13}$	$(1.41 \pm 0.98) \times 10^{-14}$	0.00	0.02 ± 0.02
SnareNet (Ours)	$(1.74 \pm 0.08) \times 10^{-1}$	$(3.16 \pm 0.29) \times 10^{-2}$	$(4.78 \pm 0.92) \times 10^{-15}$	0.00	$(5.13 \pm 1.28) \times 10^{-5}$	$(6.35 \pm 1.68) \times 10^{-7}$	0.00	0.25 ± 0.01

Table 9: Evaluation metrics on convex QCQP with 50 inequality constraints test set.

Method	Max Opt. Gap	GMean Opt. Gap	GMean Ineq. Error	# Ineq Violations	Max Eq. Error	GMean Eq. Error	# Eq Violations	Test Time (s)
Optimizer (Gurobi)	7.11×10^{-15}	2.74×10^{-16}	1.01×10^{-16}	0.00	1.58×10^{-8}	3.89×10^{-13}	0.00	1742.79
DC3	$(3.37 \pm 1.78) \times 10^{1}$	$(1.59 \pm 0.76) \times 10^{1}$	$(2.39 \pm 4.72) \times 10^{-13}$	5.33 ± 5.07	$(7.63 \pm 5.12) \times 10^{-14}$	$(5.42 \pm 3.01) \times 10^{-15}$	0.00	0.18 ± 0.08
HProj	$(7.78 \pm 17.18) \times 10^{1}$	$(3.41 \pm 7.57) \times 10^{1}$	$(1.10 \pm 2.47) \times 10^{-13}$	10.00 ± 22.36	$(2.01 \pm 4.41) \times 10^{-12}$	$(2.77 \pm 6.08) \times 10^{-13}$	0.00	0.26 ± 0.37
SnareNet (Ours)	$(1.25 \pm 0.06) \times 10^{-1}$	$(2.60 \pm 0.27) \times 10^{-2}$	$(2.68 \pm 0.64) \times 10^{-15}$	0.00	$(7.64 \pm 0.97) \times 10^{-7}$	$(5.92 \pm 2.64) \times 10^{-9}$	0.00	0.39 ± 0.01

Table 10: Evaluation metrics on convex QCQP with 100 inequality constraints test set.

Method	Max Opt. Gap	GMean Opt. Gap	GMean Ineq. Error	# Ineq Violations	Max Eq. Error	GMean Eq. Error	# Eq Violations	Test Time (s)
Optimizer (Gurobi)	7.11×10^{-15}	2.57×10^{-16}	1.04×10^{-16}	0.00	1.57×10^{-7}	7.90×10^{-13}	0.00	3563.30
DC3	$(1.14 \pm 0.66) \times 10^{1}$	6.86 ± 4.63	$(1.42 \pm 0.91) \times 10^{-16}$	0.65 ± 1.37	$(1.76 \pm 1.27) \times 10^{-14}$	$(1.70 \pm 0.89) \times 10^{-15}$	0.00	0.48 ± 0.08
HProj	$(2.35 \pm 5.22) \times 10^{2}$	$(1.53 \pm 3.40) \times 10^{2}$	$(5.40 \pm 12.08) \times 10^{-11}$	20.07 ± 44.86	$(1.83 \pm 4.03) \times 10^{-12}$	$(3.04 \pm 6.73) \times 10^{-13}$	0.00	0.44 ± 0.69
SnareNet (Ours)	$(1.74 \pm 0.08) \times 10^{-1}$	$(3.16 \pm 0.29) \times 10^{-2}$	$(1.88 \pm 0.22) \times 10^{-15}$	0.00	$(7.77 \pm 0.54) \times 10^{-7}$	$(2.80 \pm 0.97) \times 10^{-9}$	0.00	0.68 ± 0.12

D Additional Experiments: Comparison with FSNet

This section compares SnareNet with FSNet (Nguyen & Donti, 2025), a recent feasibility-seeking method that also targets hard feasibility under general constraints, on the three benchmark families in Section 5.1. Both methods use the same feasibility tolerance 10^{-4}. We report the same evaluation metrics as in Appendix C, where the numbers of inequality and equality violations are counted with threshold 10^{-4}. Tables 11 to 13 summarize the results.

Relation between FSNet and SnareNet. Both FSNet and SnareNet perform gradient-based updates on the same constraint violation function F defined in Section 4.3. FSNet takes steepest descent steps with a line search on the step size and without preconditioning, whereas SnareNet's update is equivalent to preconditioned gradient descent (PGD). FSNet's linear convergence guarantee assumes the Polyak-Lojasiewicz (PL) condition on F, which is a strong assumption for general constraints. Under the same PL assumption, the descent property (23) established for SnareNet also yields linear convergence (see also Remark 4.3).

Results. Across all three families, FSNet does not reliably enforce feasibility at tolerance 10^{-4}: its maximum inequality and equality errors are of order 10^{-3} to 10^{-2}, and its numbers of inequality and equality violations are much higher than those of SnareNet, which has no violations on any test instance. In particular, FSNet violates nearly all equality constraints in every family: 46.56 of 50 on NCLPs, 46.55 of 50 on convex QCQPs, and 9.66 of 10 on non-convex QCQPs. FSNet attains a smaller maximum optimality gap on NCLPs and convex QCQPs and lower objective values on non-convex QCQPs, but these values are attained at infeasible outputs and are therefore not directly comparable with those of SnareNet. FSNet has a shorter test time on NCLPs and convex QCQPs since its iterations do not solve a linear system, and a comparable test time on non-convex QCQPs.

Table 11: Evaluation metrics of FSNet and SnareNet on the NCLPs test set.

Method	Max Opt. Gap	GMean Opt. Gap	Max Ineq. Error	GMean Ineq. Error	# Ineq Violations	Max Eq. Error	GMean Eq. Error	# Eq Violations	Test Time (s)
FSNet (`tol` $= 10^{-4}$)	$(1.74 \pm 0.10) \times 10^{-1}$	$(1.21 \pm 0.06) \times 10^{-1}$	$(3.38 \pm 0.15) \times 10^{-3}$	$(1.41 \pm 0.24) \times 10^{-16}$	0.39 ± 0.42	$(5.87 \pm 0.70) \times 10^{-3}$	$(6.01 \pm 0.40) \times 10^{-4}$	46.56 ± 0.40	0.070 ± 0.007
SnareNet (`tol` $= 10^{-4}$)	$(4.57 \pm 0.62) \times 10^{-1}$	$(8.03 \pm 0.58) \times 10^{-2}$	$(5.29 \pm 0.67) \times 10^{-5}$	$(1.02 \pm 0.40) \times 10^{-15}$	0.00	$(8.34 \pm 1.19) \times 10^{-5}$	$(2.08 \pm 1.01) \times 10^{-6}$	0.00	0.501 ± 0.076

Table 12: Evaluation metrics of FSNet and SnareNet on the convex QCQPs test set.

Method	Max Opt. Gap	GMean Opt. Gap	Max Ineq. Error	GMean Ineq. Error	# Ineq Violations	Max Eq. Error	GMean Eq. Error	# Eq Violations	Test Time (s)
FSNet (`tol` $= 10^{-4}$)	$(1.04 \pm 0.05) \times 10^{-1}$	$(3.01 \pm 0.60) \times 10^{-2}$	$(5.88 \pm 0.51) \times 10^{-3}$	$(1.35 \pm 0.41) \times 10^{-15}$	4.12 ± 0.64	$(7.17 \pm 1.33) \times 10^{-3}$	$(6.27 \pm 0.71) \times 10^{-4}$	46.55 ± 0.51	0.144 ± 0.050
SnareNet (`tol` $= 10^{-4}$)	$(1.24 \pm 0.06) \times 10^{-1}$	$(2.60 \pm 0.26) \times 10^{-2}$	$(8.63 \pm 0.94) \times 10^{-5}$	$(7.74 \pm 1.97) \times 10^{-15}$	0.00	$(8.33 \pm 0.87) \times 10^{-5}$	$(1.93 \pm 0.45) \times 10^{-6}$	0.00	0.807 ± 0.093

Table 13: Evaluation metrics of FSNet and SnareNet on the non-convex QCQPs test set.

Method	Max Obj. Value	Mean Obj. Value	Max Ineq. Error	GMean Ineq. Error	# Ineq Violations	Max Eq. Error	GMean Eq. Error	# Eq Violations	Test Time (s)
FSNet (`tol` $= 10^{-4}$)	-22.45 ± 0.02	-22.96 ± 0.01	$(1.08 \pm 0.45) \times 10^{-2}$	$(1.04 \pm 0.72) \times 10^{-13}$	10.63 ± 1.60	$(1.00 \pm 0.40) \times 10^{-2}$	$(1.34 \pm 0.37) \times 10^{-3}$	9.66 ± 0.09	0.119 ± 0.010
SnareNet (`tol` $= 10^{-4}$)	-19.98 ± 1.89	-20.61 ± 1.81	$(2.40 \pm 1.89) \times 10^{-6}$	$(6.19 \pm 10.78) \times 10^{-10}$	0.00	$(1.07 \pm 0.84) \times 10^{-6}$	$(3.87 \pm 2.92) \times 10^{-7}$	0.00	0.124 ± 0.059

E Neural Control Policies Experiment Details

The system state is $x(t) = (p_x(t), p_y(t), \theta(t), v(t), w(t)) \in \mathbb{R}^5$ at time t, where p_x, p_y are the coordinates, θ is the heading angle, v is the linear velocity, and w is the angular velocity. The control $u(t) = (a(t), \alpha(t)) \in \mathbb{R}^2$ includes linear acceleration a and angular acceleration α. The control changes the system state by the linear dynamics: $\dot{x} = F(x) + Gu$, where $F : \mathbb{R}^5 \to \mathbb{R}^5$ and $G \in \mathbb{R}^{5\times 2}$ are

$$F(x) = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ w \\ 0 \\ 0 \end{bmatrix} \quad \text{and} \quad G(x) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Our neural policy is trained by minimizing a quadratic cost over a finite time horizon:

$$\Delta t \sum_{i=0}^{n_t} x(t_i)^T Q x(t_i) + c\|\pi_\theta(x(t_i))\|^2,$$

Algorithm 1 Repair layer in SnareNet

1: **assume** $\hat{y} = \mathcal{M}_\theta(x)$ and constraints $\ell \leq g(y) \leq u$
2: **function** $\mathcal{R}(\hat{y}, \lambda, \varepsilon)$
3: **init** $y^0 = \hat{y}$
4: **for** $k = 0, 1, 2, \ldots$ until convergence **do**
5: **compute** $z^k := \mathcal{P}_{\mathbf{B}(\ell-\varepsilon, u+\varepsilon)}(g(y^k))$
6: **update** y^{k+1} using Equation (7) with λ
7: **end for**
8: **return** $\breve{y} = y^{k+1}$
9: **end function**

where $Q = \text{diag}(100, 100, 0, 0.1, 0.1)$ penalizes deviations from the target position $(0, 0)$ and velocity, $c = 0.1$ weights the control effort, and $n_t = 10$ steps for trajectories. This objective encourages trajectories to reach the origin $(0, 0)$ with minimal control effort.

For an elliptical obstacle centered at (c_x, c_y) with axes (r_x, r_y), we use the higher order CBF $h(x) = \dot{h}_e(x) + \kappa h_e(x)$ with $\kappa > 0$, (Min & Azizan, 2024) in our experiments, where

$$h_e(x) = \big(\frac{c_x - p_x + \ell\cos\theta}{r_x}\big)^2 + \big(\frac{c_y - p_y + \ell\sin\theta}{r_y}\big)^2 - 1.$$

For training, we uniformly sample initial states from two boxes in the state space, differing only in position: Box 1 with positions $(p_x, p_y) \in [-5.0, -5.5] \times [7.5, 8.0]$ and Box 2 with positions $(p_x, p_y) \in [-8.5, -8.0] \times [5.5, 6.0]$. Both boxes share the same heading angle range $\theta \in [-\frac{\pi}{4}, -\frac{\pi}{8}]$ and zero initial velocities ($v = w = 0$).

Table 14 shows the detailed evaluation statistics: SnareNet achieves a 99.4% feasibility rate on the test set compared to DC3's 64.9% feasibility rate, demonstrating the effectiveness of SnareNet in enforcing safety constraints in neural control policies.

Table 14: Evaluation statistics on 84 CBF test instances.

Method	Dataset	Objective	# Feasible (1e-4)	Max Ineq. Error	GMean Ineq. Error
DC3	Box 1	3656	58	6.555	6.785×10^{-15}
SnareNet	Box 1	4060	84	4.21×10^{-8}	1.763×10^{-14}
DC3	Box 2	4269	51	4.17	1.913×10^{-14}
SnareNet	Box 2	5226	83	0.7483	4.038×10^{-14}

F Algorithms of SnareNet

Algorithm 1 provides the pseudocode for SnareNet's repair layer. The training algorithm of SnareNet is summarized in Algorithm 2.

G Soft Constraint Training Can Be Counterproductive

Soft constraint training is commonly employed as a warm-up strategy for hard-constrained neural networks. However, as shown in Figure 7, this approach can be counterproductive. While HardNet achieves a lower optimality gap after 1000 soft epochs than after 500, this reduction does not persist: once hard constraint training begins, the optimality gap increases sharply, negating any apparent progress. Additional soft constraint epochs constitute an inefficient use of computational resources, as the gains from soft constraint training do not transfer to improved final model performance.

Algorithm 2 Training paradigm of SnareNet

function TRAIN($\mathcal{X}_{\text{train}}$)
  init neural network $\mathcal{M}_\theta : \mathbb{R}^d \to \mathbb{R}^n$ and $\lambda \geq 0$
  init constraint violation $\varepsilon_x^{(0)}$ for $x \in \mathcal{X}_{\text{train}}$
  set a relaxation parameter schedule $\{\varepsilon_x^{(t)}\}_{t=1}^T$
  for epoch $t = 1, 2, \ldots, T$ **do**
    for mini-batch $\mathcal{B} \subset \mathcal{X}_{\text{train}}$ **do**
      compute approximate solution $\hat{y}_i = \mathcal{M}_\theta(x_i)$ for all $x_i \in \mathbf{B}$
      repair to relaxed feasible solution $\check{y}_i = \mathcal{R}(\hat{y}_i, \lambda, \varepsilon_{x_i}^{(t)})$ for all $x_i \in \mathbf{B}$
      compute batch loss $\mathcal{L}_\mathcal{B}(\theta)$
      update θ using $\nabla_\theta \mathcal{L}_\mathcal{B}(\theta)$
    end for
  end for
end function

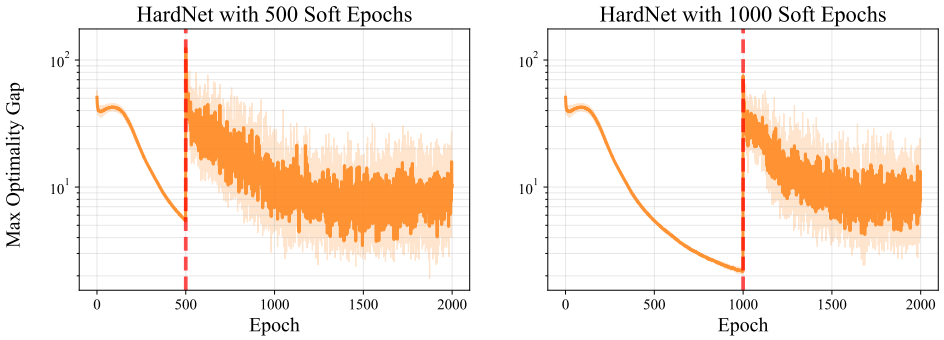


Figure 7: Maximum optimality gap over all NCP validation instances for HardNet trained with soft constraints for 500 and 1000 epochs and hard constraints for the rest.

H Scaling of Computational Resources

Table 15 shows the training time, memory usage, and number of repair iterations taken for SnareNet on non-convex programs (NCPs) of varying sizes. The experiments were run on an Intel 6426Y CPU and an NVIDIA L40S GPU, constrained to 16, 16, and 32 GB of GPU memory per run for 100, 500, and 1000 constraints, respectively. Note that SnareNet may use less computational resources for larger problems under a fixed number of constraints since the problems can be less constrained and require fewer repair iterations to achieve feasibility. For example, the 500 and 1000 variable problems with 100 constraints are cheaper (both in time and memory) than the 100 variable problem with 100 constraints. Table 15 indicates that SnareNet is more suitable for problems with $m \ll n$.

Table 15: Total training time (sec) / memory usage / maximum repair iterations on NCPs of varying sizes.

	100 constraints (50 eq. + 50 ineq.)	500 constraints (50 eq. + 450 ineq.)	1000 constraints (50 eq. + 950 ineq.)
100 var	3674.4s / 2.3GB / 32	3496.9s / 4.4GB / 100	5832.5s / 4.9GB / 100
500 var	869.1s / 1.3GB / 3	6493.8s / 11.6GB / 11	29301.7s / 23.8GB / 30
1000 var	1107.3s / 1.3GB / 3	2724.5s / 3.3GB / 3	>86400.0s / 31.1GB / 8