Project midterm report – Safe & Constrained AI

Hippolyte Wallaert
Institute for Computational & Mathematical Engineering
Stanford University
`hippowal@stanford.edu`

Abstract

This project investigates the training dynamics and loss landscapes of differentiable optimization layers, specifically focusing on cvxpy and Theseus layers, as means to enforce constraints within neural networks. By comparing these hard-constraint approaches against traditional soft penalty techniques, the study aims to identify meaningful patterns that can inform the design of more effective training strategies and enhance the learning abilities of these layers. The experimental framework evaluates these dynamics across multiple neural network backbones, including MLP and Transformer architectures, using synthetic QC-QP optimization datasets. A comprehensive suite of metrics is tracked to analyze learning behaviors, including gradient signal-to-noise ratio, Jacobian effective rank, and Hessian-based loss landscape sharpness. Ultimately, the findings are presented through a modular, web-based visual interface designed to facilitate the diagnostic analysis of the complex optimization landscapes induced by differentiable layers.

1 Project Overview & Scope Refinement

As a reminder, the goal of this project is to study the training dynamics of differentiable optimization layers Agrawal et al. [2019], Pineda et al. [2023], and hopefully identify meaningful patterns that help understand the loss landscape induced by such layers. These experiments will help design better training strategies for such layers, summarizing the findings in clear visuals to communicate the results effectively. These better training strategies will hopefully help enhance the learning abilities of such layers, which are already known to have universal approximation guarantees (at least in simple cases Min and Azizan [2025]).

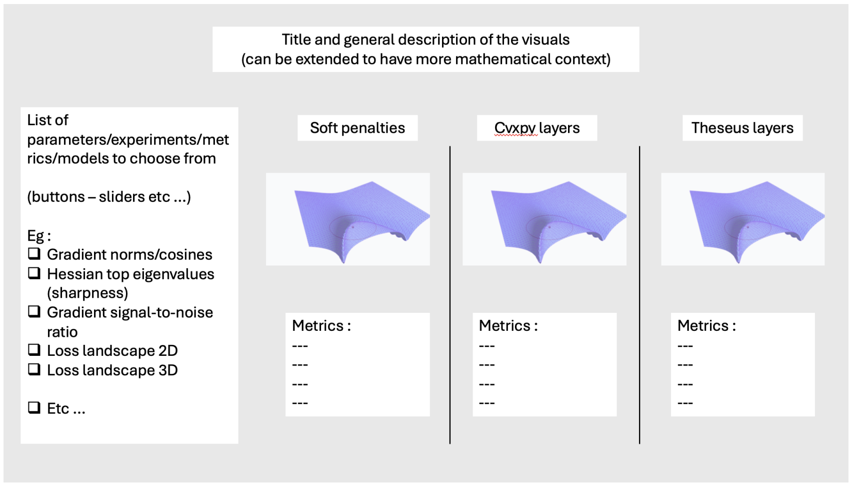
The experiments are mainly focused on comparing the training dynamics of differentiable optimization layers against soft penalty techniques, which are the competing strategy to enforce (soft) constraints in neural networks. The experiments will be conducted on cvxpy layers and Theseus layers, the latter being a generalization of the repair layer developed in Chu et al. [2026]. A detailed derivation of this result is given in the appendix.

The format of the final visuals will be a website as mentioned in the project proposal. The current plan is to develop a similar interface to LossLens Xie et al. [2024]. A sketch of the interface is given below in Figure 1 (and is fully open to comments and improvements !). This organization will allow to quickly gather together all the different experiment results in a simple and modular way. A summary of current and targeted experiments is given in the following section.

2 Current and targeted experiments

2.1 Constraints enforcement techniques

First, recall the problem formulation we are focusing on. The inputs are $x \in \mathcal{X} \subset \mathbb{R}^d$ and the outputs are $y \in \mathcal{Y} \subset \mathbb{R}^n$. The problem is formalized as :


Title and general description of the visuals
(can be extended to have more mathematical context)

List of parameters/experiments/metrics/models to choose from

(buttons – sliders etc ...)

Eg :
- ❑ Gradient norms/cosines
- ❑ Hessian top eigenvalues (sharpness)
- ❑ Gradient signal-to-noise ratio
- ❑ Loss landscape 2D
- ❑ Loss landscape 3D
- ❑ Etc ...

Soft penalties	Cvxpy layers	Theseus layers
Metrics : --- --- --- ---	Metrics : --- --- --- ---	Metrics : --- --- --- ---

Figure 1: Draft visual for the final experiments visualisation platform

$$\min_{\theta} \mathbb{E}_{x\sim\mathcal{X}}[\ell(\theta;x)] \quad \text{s.t.} \quad y \in \mathcal{C}_x,\ \forall x \in \mathcal{X} \tag{1}$$

where $\mathcal{C}_x = \{y \in \mathcal{Y} \mid l_x \leq g_x(y) \leq u_x\}$.

Notations are borrowed from Chu et al. [2026].

As mentioned above, the experiments conducted focus on two different types of constraint enforcement techniques :

1. **Soft penalty loss** : the architecture consists of a given neural network backbone (of any architecture) and directly enforces the constraints through the loss function :
$$\ell_{\text{soft}}(\theta;x) = \ell(\theta;x) + \mu_u \|\operatorname{ReLU}(g(\hat{y}) - u)\|_2 + \mu_\ell \|\operatorname{ReLU}(\ell - g(\hat{y}))\|_2 \tag{2}$$
This strategy offers no strict guarantee of constraint satisfaction at inference time.
2. **Hard constraints through repair layers** : a simple way to enforce hard constraints is to incorporate directly a differentiable repair layer after the neural model in the architecture and let the gradients back-propagate through this repair layer. Different architectures have been developed for these repair layers, including Amos and Kolter [2021], Donti et al. [2021], Min and Azizan [2025] or Chu et al. [2026]. An example of this general framework is given by Figure 2.

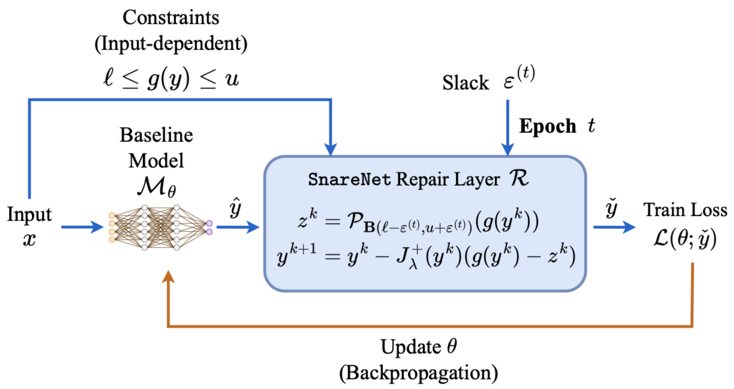

Constraints
(Input-dependent)
$\ell \leq g(y) \leq u$
Slack $\varepsilon^{(t)}$
Epoch t
Baseline Model $\mathcal{M}_\theta$
Input x
$\hat{y}$
SnareNet Repair Layer $\mathcal{R}$
$z^k = \mathcal{P}_{\mathbf{B}(\ell-\varepsilon^{(t)}, u+\varepsilon^{(t)})}(g(y^k))$
$y^{k+1} = y^k - J_\lambda^+(y^k)(g(y^k) - z^k)$
$\check{y}$
Train Loss $\mathcal{L}(\theta;\check{y})$
Update θ
(Backpropagation)

Figure 2: SnareNet architecture

Since the goal of this project is to target differentiable layers in the most general setting, we chose not to focus on specific repair layers like the ones described above. It is actually possible to rewrite these repair layers as specific cases of general convex (cf. cvxpy layers) or nonlinear (cf. Theseus layers) differentiable layers. Hence our work will focus on such implementations of repair layers instead. We acknowledge that this choice maybe not be optimal from the pure purpose of enforcing constraints in neural networks, especially since direct implementations without any general abstractions will be faster to run. However targeting this layers extends considerably the scope of these experiments as these layers are widely used across many applications, which may not be the case for the architectures mentioned above yet.

2.2 Neural networks backbones

As mentioned in the previous section, these constraints enforcement techniques can be incorporated into any neural network architecture, which possibly makes the study of their training dynamics highly dependent of the chosen backbone. Hence this work will try to investigate different neural networks backbones in order to reach robust conclusions. Current implementation and experiments are limited to MLP architectures with various widths and depths, but a complementary study will include transformer architectures as neural network backbones.

2.3 Adaptative relaxation

One significant finding introduced in Chu et al. [2026] consists in using adaptative relaxation in constraints formulations to enable better learning abilities of the hard constrained methods. This work will try to investigate the benefits of this technique under the lens of training dynamics to identify the core advantages of this technique over traditional hard constraint enforcing patterns.

2.4 Metrics and analysis of training dynamics

More importantly, we define clearly the aspects and metrics of training dynamics that our project targets. The goal is to hopefully identify specific behaviors proper to such differentiable layers that could help develop specified techniques to train and leverage them optimally. An extensive list of the metrics and quantities investigated currently is (from basic to more advanced):

1. Any loss related quantity — which is still useful to evaluate the performance of given training configurations
2. Constraint violations quantities (number/magnitude of violations) — which is also necessary to evaluate the performance in our problem
3. Gradient metrics :
   (a) Gradient cosine similarity between loss components in the soft penatly setting
   (b) Gradient signal-to-noise ratio
   (c) Gradient noise scale — to hopefully identify useful rules for choosing efficient batch sizes as done in McCandlish et al. [2018]
   (d) Jacobian effective rank — to identify potential signal collapse in hard constraint settings
4. Loss landscape metrics :
   (a) Hessian top eigenvalues — to evaluate loss sharpness which happens to be critical both for understanding edge of stability behaviors Cohen et al. [2022]
   (b) Plotting the loss landscape using most significant directions — following the techniques developed in Li et al. [2018]

2.5 Datasets

Currently experiments are conducted on synthetic optimization problems, mainly randomly generated QC-QP problems so that all three different methods can be applied. The datasets have been directly recovered from the experimental code of Chu et al. [2026].

Ideally experiments would extend to more practical settings, like basic robotics control problems or optimal powerflow instances, which are the most common applications of constrained neural networks. However this has not been explored yet in this project.

3 Preliminary results

First, significant work has been conducted to refine the scope of this project and properly set up the plan for all the experiments. This led to the current formulation organized around the comparison of soft penalty methods against cvxpy and Theseus layers. The generality of these experiments is considered very interesting and the incorporation of existing repair strategies within this framework (as developed in the Appendix) has been part of this preliminary work.

This work has been followed by a detailed literature review to identify the most relevant metrics to track in order to investigate the learning dynamics of these architectures. Additional reading has been conducted to compare existing frameworks to display similar experiments on visual platforms, leading to the visual proposal mentioned in Figure 1.

The codebase has been (nearly) completely set up and experiments have started using basic MLP backbone architectures.

The code of the project is accessible at the following repository : Github.

4 Work continuation

Further experiments have to be conducted to identify more precisely interesting learning behaviors. The major steps remaining in terms of implementation are :

1. Add properly cvxpy layers to the experiments, as the previous code has been discovered to be incorrect and needs to be updated
2. Add loss visualisation experiments based on the work done in Li et al. [2018]

Once these steps are completed, the core of the work consists in launching the most relevant experiments and iterate until satisfactory results are obtained.

Finally the last significant implementation step will be the development of the final visual platform to display all the experiment results together. The design investigation and choices have already been done so only the practical implementation remains. Hopefully the current experimental setup has been developed in the most organized possible way, so incorporating the results into the website will be relatively simple.

References

A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and Z. Kolter. Differentiable Convex Optimization Layers, Oct. 2019. URL `http://arxiv.org/abs/1910.12430`. arXiv:1910.12430 [cs].

B. Amos and J. Z. Kolter. OptNet: Differentiable Optimization as a Layer in Neural Networks, Dec. 2021. URL `http://arxiv.org/abs/1703.00443`. arXiv:1703.00443 [cs].

Y.-C. Chu, A. Boukas, and M. Udell. SnareNet: Flexible Repair Layers for Neural Networks with Hard Constraints, Feb. 2026. URL `http://arxiv.org/abs/2602.09317`. arXiv:2602.09317 [cs].

J. M. Cohen, S. Kaur, Y. Li, J. Z. Kolter, and A. Talwalkar. Gradient Descent on Neural Networks Typically Occurs at the Edge of Stability, Nov. 2022. URL `http://arxiv.org/abs/2103.00065`. arXiv:2103.00065 [cs].

P. L. Donti, D. Rolnick, and J. Z. Kolter. DC3: A learning method for optimization with hard constraints, Apr. 2021. URL `http://arxiv.org/abs/2104.12225`. arXiv:2104.12225 [cs].

H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein. Visualizing the Loss Landscape of Neural Nets, Nov. 2018. URL `http://arxiv.org/abs/1712.09913`. arXiv:1712.09913 [cs].

S. McCandlish, J. Kaplan, D. Amodei, and O. D. Team. An Empirical Model of Large-Batch Training, Dec. 2018. URL `http://arxiv.org/abs/1812.06162`. arXiv:1812.06162 [cs].

Y. Min and N. Azizan. HardNet: Hard-Constrained Neural Networks with Universal Approximation Guarantees, Oct. 2025. URL `http://arxiv.org/abs/2410.10807`. arXiv:2410.10807 [cs].

L. Pineda, T. Fan, M. Monge, S. Venkataraman, P. Sodhi, R. T. Q. Chen, J. Ortiz, D. DeTone, A. Wang, S. Anderson, J. Dong, B. Amos, and M. Mukadam. Theseus: A Library for Differentiable Nonlinear Optimization, Jan. 2023. URL `http://arxiv.org/abs/2207.09442`. arXiv:2207.09442 [cs].

T. Xie, J. Chen, Y. Yang, C. Geniesse, G. Shi, A. Chaudhari, J. K. Cava, M. W. Mahoney, T. Perciano, G. H. Weber, and R. Maciejewski. LossLens: Diagnostics for Machine Learning through Loss Landscape Visual Analytics, Dec. 2024. URL `http://arxiv.org/abs/2412.13321`. arXiv:2412.13321 [cs].

A SnareNet repair layer as a general Theseus layer

SnareNet Recall from Figure 2 the formulation of the repair layer $\mathcal{R}$. A neural model produces an output $\hat{y}$ and the goal of the repair layer is to modify it to satisfy the constraints. This is done by solving the nonlinear equation for y:

$$g(y) = z \tag{3}$$

where $z = \mathcal{P}_{\mathtt{B}(\ell,u)}(g(\hat{y}))$ is the box-projection of $g(\hat{y})$ on the constraints $\mathcal{C}_x = \{y \in \mathcal{Y} \mid l_x \leq g_x(y) \leq u_x\}$. We refer to Chu et al. [2026] for more detailed explanations of the notations. This nonlinear equation is solved using Newton's method and Levenberg-Marquardt regularization which gives the following update rule for an iterate y^k:

$$y^{k+1} = y^k - \underbrace{(J_g(y^k)^\top J_g(y^k) + \lambda I)^{-1} J_g(y^k)^\top}_{J_\lambda^\dagger(y^k)}(g(y^k) - z^k) \tag{4}$$

where $z^k = \mathcal{P}_{\mathtt{B}(\ell,u)}(g(y^k))$ is the target image point and J_g denotes the Jacobian of the constraint function. The regularization term $\lambda||y - y^k||^2$ ensures that the update remains local to the linearization point.

Note that we omit purposely the adaptative relaxation mechanism in this analysis, as it does not change the equivalence we are proving and would just increase the notations complexity.

Theseus layers The Theseus library generalizes this process by defining a Differentiable Nonlinear Least Squares (DNLS) layer. Given an input θ, some cost functions c_i and cost weights w_i, Theseus minimizes the objective :

$$S(\theta) = \frac{1}{2}\sum_i ||w_i c_i(\theta^i)||^2 \tag{5}$$

where θ_i designates the subset of variables on which the i-th constraint depends [1]. We can adapt this formulation to match SnareNet's formulation :

- Rewrite the input parameter θ to be $\hat{y}$. We also ignore any sparsity consideration and drop the i exponent.
- Write the cost functions to be the constraint violations at every indices : $c_i(y) = g_i(y) - z_i$.
- Make the constraint violations to be equivalent, thus setting the weights w_i to 1.

With these new notations, the Theseus layer can be expressed as the following mapping :

$$\mathcal{R}_{\text{Theseus}} : \ \hat{y} \mapsto \arg\min_y \frac{1}{2}\sum_i \|g_i(y) - z_i\|^2 =: \check{y} \tag{6}$$

Its internal Levenberg-Marquardt optimizer computes an update Δ by solving the following linearized system:

$$(J^\top J + \lambda I)\Delta = J^\top r \tag{7}$$

where $r = g(y) - z$ is the residual vector and J is the Jacobian of the residuals. The Theseus update Δ becomes mathematically equivalent to the SnareNet repair step.

To be more precise, the Theseus layer is only equivalent to the SnareNet repair step if the cost function is updated at every iteration to account for the update of z^k. This can be implemented in Theseus layers by providing an `end_iter_callback` to the optimizer.

[1]This detail enables to leverage sparse solvers in the direct implementation of the layer, which is very useful in some particular cases where the constraints only depend on small subsets of the general input variable.

Note on preconditioning It follows from this analysis that SnareNet could be enhanced by using different weights w_i to account for different scales in constraints formulation. Indeed, using more advanced weighting than forcing $w_i = 1$ could theoretically help managing groups of constraints with different magnitudes. This can be seen as reformulating the least squares problem in a different norm than the standard Euclidian norm, namely a W-norm where W is a diagonal positive matrix. In this case the update would be written :

$$(J^\top w^\top w J + \lambda I)\Delta = J^\top w^\top r \tag{8}$$