NCVX: A General-Purpose Optimization Solver for Constrained Machine and Deep Learning

author names withheld

Under Review for OPT 2022

Abstract

Imposing explicit constraints are new and increasingly trendy in deep learning, stimulated by, e.g., trustworthy AI that performs robust optimization over complicated perturbation sets and scientific applications that need to respect physical laws and constraints. However, it can be hard to reliably solve constrained deep learning problems without optimization expertise. Existing deep learning frameworks do not admit constraints. General-purpose optimization packages can handle constraints but do not perform auto-differentiation and have trouble dealing with nonsmoothness. In this paper, we introduce a new software package called `NCVX`, whose initial release contains the solver `PyGRANSO`, a PyTorch-enabled general-purpose optimization package for constrained machine and deep learning problems, the first of its kind. `NCVX` inherits auto-differentiation, GPU acceleration, and tensor variables from PyTorch, is built on freely available and widely used open-source frameworks. `NCVX` is available at https://ncvx.org, with detailed documentation and numerous examples from machine/deep learning and other fields.

1. Introduction

Mathematical optimization is an indispensable modeling and computational tool for all science and engineering fields, especially for machine learning. To date, researchers have developed numerous foolproof techniques and user-friendly solvers and modeling languages for convex (CVX) problems, such as SDPT3 [63], Gurobi [27], Cplex [13], TFOCS [5], CVX(PY) [21, 26], AMPL [23], YALMIP [43]. These developments have substantially lowered the barrier of CVX optimization for non-experts. However, practical problems, especially from machine and deep learning, are often nonconvex (NCVX), and possibly also nonsmooth (NSMT) and constrained (CSTR).

There are methods and packages handling NCVX problems in restricted settings: PyTorch [53] and TensorFlow [1] can solve large-scale NCVX, NSMT problems without constraints. CSTR problems can be heuristically turned into penalty forms, thus making the constraints implicit, but this may not produce feasible solutions for the original problems. When the constraints are simple, structured methods such as projected (sub)gradient and Frank-Wolfe [58] can be used. When the constraints are differentiable manifolds, one can consider manifold optimization methods and packages, e.g., (Py)manopt [6, 64], Geomstats [50], McTorch [48], and Geoopt [38]. For general CSTR problems, KNITRO [55] and IPOPT [70] implement interior-point methods, while ensmallen [17] and GENO [41] rely on augmented Lagrangian methods. However, moving beyond smooth (SMT) constraints, both of these families of methods, at best, handle only special types of NSMT constraints. Finally, packages specialized for machine learning, e.g., scikit-learn [54], MLib [49] and Weka [68], often use problem-specific solvers that cannot be easily extended to new formulations.

© .

2. The GRANSO and NCVX packages

GRANSO[1] is among the first optimization packages that can handle general NCVX, NSMT, CSTR problems [18]:

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}), \quad \text{s.t. } c_i(\boldsymbol{x}) \leq 0, \ \forall\, i \in \mathcal{I}; \quad c_i(\boldsymbol{x}) = 0, \ \forall\, i \in \mathcal{E}. \tag{1}$$

Here, the objective f and constraint functions c_i's are only required to be almost everywhere continuously differentiable. GRANSO is based on quasi-Newton updating with sequential quadratic programming (BFGS-SQP) and has the following advantages: (1) **unified treatment of NCVX problems**: no need to distinguish CVX vs NCVX and SMT vs NSMT problems, similar to typical nonlinear programming packages; (2) **reliable step-size rule**: specialized methods for NSMT problems, such as subgradient and proximal methods, often entail tricky step-size tuning and require the expertise to recognize the structures [58], while GRANSO chooses step sizes adaptively via a gold-standard line search; (3) **principled stopping criterion**: GRANSO stops its iteration by checking a theory-grounded stationarity condition for NMST problems, whereas specialized methods are usually stopped when reaching ad-hoc iteration caps.

However, GRANSO users must derive gradients analytically[2] and then provide code for these computations, a process which is often error-prone in machine learning and impractical for modern deep learning. Furthermore, as part of the MATLAB software ecosystem, GRANSO is generally not compatible with existing popular machine learning frameworks—mostly in Python and R—and users' own existing toolchains. To overcome these issues and facilitate both high performance and ease of use in machine and deep learning, we introduce a new software package called NCVX, whose initial release contains the solver PyGRANSO, a PyTorch-enabled port of GRANSO with several new key features: (1) auto-differentiation of all gradients, which is a main feature to make PyGRANSO user-friendly; (2) support for both CPU and GPU computations for improved hardware acceleration and massive parallelism; (3) support for general tensor variables including vectors and matrices, as opposed to the single vector of concatenated optimization variables that GRANSO uses; (4) integrated support for OSQP [59] and other QP solvers for respectively computing search directions and the stationarity measure on each iteration. OSQP generally outperforms commercial QP solvers in terms of scalability and speed. All of these enhancements are crucial for solving large-scale machine/deep learning problems. NCVX, licensed under the AGPL version 3, is built entirely on freely available and widely used open-source frameworks; see https://ncvx.org for documentation and examples.

3. Usage examples: dictionary learning and neural perceptual attack

In order to make NCVX friendly to non-experts, we strive to keep the user input minimal. The user is only required to specify the optimization variables (names and dimensions of variables) and define the objective and constraint functions. Here, we briefly demonstrate the usage of PyGRANSO solver on a couple of machine learning problems.

1. http://www.timmitchell.com/software/GRANSO/
2. GRANSO is implemented in MATLAB and does not support auto-differentiation, although recent versions of MATLAB have included primitive auto-differentiation functionalities.

Orthogonal Dictionary Learning (ODL, [3]) One hopes to find a "transformation" $\boldsymbol{q} \in \mathbb{R}^n$ to sparsify a data matrix $\boldsymbol{Y} \in \mathbb{R}^{n \times m}$:

$$\min_{\boldsymbol{q} \in \mathbb{R}^n} \; f(\boldsymbol{q}) \doteq \frac{1}{m} \|\boldsymbol{q}^\intercal \boldsymbol{Y}\|_1, \quad \text{s.t. } \|\boldsymbol{q}\|_2 = 1, \tag{2}$$

where the constraint $\|\boldsymbol{q}\|_2 = 1$ is to avoid the trivial solution $\boldsymbol{q} = \boldsymbol{0}$. Problem (2) is NCVX, NSMT, and CSTR: nonsmoothness comes from the objective, and nonconvexity comes from the constraint. Demos 1 & 2 show the implementations of ODL in `GRANSO` and `PyGRANSO`, respectively. Note that the analytical gradients of the objective and constraint functions are not required in `PyGRANSO`.

```
function[f,fg,ci,cig,ce,ceg]=fn(q)
   f = 1/m*norm(q'*Y, 1);%obj
   fg = 1/m*Y*sign(Y'*q);%obj grad
   ci = [];cig = [];%no ineq constr
   ce = q'*q - 1; % eq constr
   ceg = 2*q; % eq constr grad
end
soln = granso(n,fn);
```

Demo 1: `GRANSO` for ODL

```
def fn(X_struct):
   q = X_struct.q
   f = 1/m*norm(q.T@Y, p=1) # obj
   ce = pygransoStruct()
   ce.c1 = q.T@q - 1 # eq constr
   return [f,None,ce]
var_in = {"q": [n,1]}# def variable
soln = pygranso(var_in, fn)
```

Demo 2: `PyGRANSO` for ODL

Neural Perceptual Attack (NPA, [40]) The CSTR deep learning problem, NPA, is shown below:

$$\max_{\widetilde{\boldsymbol{x}}} \; \mathcal{L}\left(f\left(\widetilde{\boldsymbol{x}}\right), y\right), \quad \text{s.t. } d\left(\boldsymbol{x}, \widetilde{\boldsymbol{x}}\right) = \|\phi\left(\boldsymbol{x}\right) - \phi\left(\widetilde{\boldsymbol{x}}\right)\|_2 \leq \epsilon. \tag{3}$$

Here, $\boldsymbol{x}$ is an input image, and the goal is to find its perturbed version $\widetilde{\boldsymbol{x}}$ that is perceptually similar to $\boldsymbol{x}$ (encoded by the constraint) but can fool the classifier f (encoded by the objective). The loss $\mathcal{L}(\cdot, \cdot)$ is the margin loss used in Laidlaw et al. [40]. Both f in the objective and ϕ in the constraint are deep neural networks with ReLU activations, making both the objective and constraint functions NSMT and NCVX. The $d\left(\boldsymbol{x}, \widetilde{\boldsymbol{x}}\right)$ distance is called the Learned Perceptual Image Patch Similarity (LPIPS) [40, 72]. Demo 3 is the `PyGRANSO` example for solving problem (3). Note that the codes for data loading, model specification, loss function, and LPIPS distance are not included here. It is almost impossible to derive analytical subgradients for problem (3), and thus the auto-differentiation feature in `PyGRANSO` is necessary for solving it.

```
def comb_fn(X_struct):
   adv_inputs = X_struct.x_tilde
   f = MarginLoss(model(adv_inputs),labels) # obj
   ci = pygransoStruct()
   ci.c1 = lpips_dists(adv_inputs) - 0.5 # ineq constr. bound eps=0.5
   return [f,ci,None] # No eq constr
var_in = {"x_tilde": list(inputs.shape)} # define variable
soln = pygranso(var_in,comb_fn)
```

Demo 3: `PyGRANSO` for NPA

4. Constrained deep learning applications

In this section, we highlight 4 families of constrained deep learning problems with highly nontrivial, often nonsmooth, constraints. These constraints cannot be easily built into the underlying neural networks. PyGRANSO can directly solve these problems, and detailed codes and tutorials are available on https://ncvx.org/examples.

4.1. Robustness of deep learning models

In visual recognition, deep neural networks (DNNs) are not robust against perturbations that are easily discounted by human perception either adversarially constructed or naturally occurring [25, 32]. To formalize robustness, one popular way is the *adversarial loss* defined as [46]

$$\max_{\boldsymbol{x}'} \ell\left(\boldsymbol{y}, f_{\boldsymbol{\theta}}(\boldsymbol{x}')\right) \quad \text{s.t. } \boldsymbol{x}' \in \Delta(\boldsymbol{x}) = \{\boldsymbol{x}' \in [0,1]^n : d\left(\boldsymbol{x}, \boldsymbol{x}'\right) \leq \epsilon\}, \tag{4}$$

where f_θ is the DNN model, and $\Delta(\boldsymbol{x})$ is the set of allowable perturbations. Early works assume that $\Delta(\boldsymbol{x})$ is the ℓ_p norm ball intersected with the natural image box, i.e., $\{\boldsymbol{x}' \in [0,1]^n : \|\boldsymbol{x} - \boldsymbol{x}'\|_p \leq \epsilon\}$, where $p = 1, 2, \infty$ are popular choices [25, 46]. To capture visually realistic perturbations other than ℓ_p norm balls, recent work has also modeled nontrivial transformations [32], which induce non-ℓ_p metrics. For empirical robustness evaluation (RE), solving Eq. (4) leads to the worst perturbations that could fool $f_{\boldsymbol{\theta}}$.

An alternative formalism of robustness is the *robustness radius* (or minimum distortion radius), defined as the minimal level of perturbation that can cause $f_{\boldsymbol{\theta}}$ to change its predicted class:

$$\min_{\boldsymbol{x}' \in [0,1]^d} d\left(\boldsymbol{x}, \boldsymbol{x}'\right) \quad \text{s.t. } \max_{i \neq y} f^i_{\boldsymbol{\theta}}(\boldsymbol{x}') \geq f^y_{\boldsymbol{\theta}}(\boldsymbol{x}'). \tag{5}$$

Solving Eq. (5) produces not only a minimally distorted perturbation $\boldsymbol{x}'$, but also a robustness radius, which makes it another popular choice for RE [14, 15]. For small and restricted $f_{\boldsymbol{\theta}}$ and selected d, Eq. (5) can be solved exactly by mixed integer programming [7, 37, 62]. For general $f_{\boldsymbol{\theta}}$ and selected d, lower bounds of the robust radius can be computed [45, 66, 67, 71]. But in general, Eq. (5) is heuristically solved via gradient-based methods or iterative linearization [8, 14, 29, 51, 56, 57, 61].

Eqs. (4) and (5) are difficult constrained problems, especially when d is sophisticated—necessary for modeling realistic perturbations [22, 32, 33, 39, 69]. Most popular solvers for them rely on explicit projections onto simple sets, and they will not work when d is a non-ℓ_p metric. Also, previous work has shown that the quality of the solution using these handcrafted methods is sensitive to key hyperparameters: e.g., step-size schedule and iteration budget [9, 15]. With PyGRANSO, we can reliably solve Eqs. (4) and (5) with general d's with minimal hyperparameter tuning.

4.2. Neural structural optimization

Designing physical structures such as bridges, optical devices and airplanes often boils down to structural optimization, a fundamental problems in physical science and engineering in general [10, 11, 34]. A primitive version of structural optimization takes the form:

$$\min_{\boldsymbol{x}, \boldsymbol{u}} \boldsymbol{u}^\intercal \boldsymbol{K}(\boldsymbol{x}) \boldsymbol{u} \quad \text{s.t. } \boldsymbol{K}(\boldsymbol{x})\boldsymbol{u} = \boldsymbol{f}, \boldsymbol{V}(\boldsymbol{x}) = v_0, \boldsymbol{x} \in \{0,1\}^d, \tag{6}$$

where $\boldsymbol{x} \in \mathbb{R}^d$ is the vectorized design variable that indicates where the material should be put to form the structure, $\boldsymbol{u} \in \mathbb{R}^n$ is the displacement vector, $\boldsymbol{K}$ is the global stiffness matrix, $\boldsymbol{f}$ is

the vector of external force. For the constraints, $\text{s.t.}\ \boldsymbol{K}(\boldsymbol{x})\boldsymbol{u} = \boldsymbol{f}$ encodes the physical laws (e.g., Hooke's law), $\boldsymbol{V}(\boldsymbol{x}) = v_0$ limits the amount of material to be used.

Recent work [34] introduces the deep image prior (DIP) [65] idea into Eq. (6): $\boldsymbol{x}$ is reparametrized as $\boldsymbol{x} = g_{\boldsymbol{\theta}}(\boldsymbol{\beta})$, where $g_{\boldsymbol{\theta}}$ is a deep neural network parametrized by $\boldsymbol{\theta}$, and $\boldsymbol{\beta}$ is a fixed random input. DIP is know to encourage spatial continuity of the design—which is not explicitly encoded in Eq. (6). Also, the overparametrization in DIP tends to ease the global optimization. Relaxing the integer constraint $\boldsymbol{x} \in \{0,1\}^d$ into a box constraint $\boldsymbol{x} \in [0,1]^d$ as is typically done in the field, we arrive at

$$\min_{\boldsymbol{\theta},\boldsymbol{u}} \boldsymbol{u}^\intercal \boldsymbol{K}(g_{\boldsymbol{\theta}}(\boldsymbol{\beta}))\boldsymbol{u} \quad \text{s.t.}\ \boldsymbol{K}(g_{\boldsymbol{\theta}}(\boldsymbol{\beta}))\boldsymbol{u} = \boldsymbol{f}, \boldsymbol{V}(g_{\boldsymbol{\theta}}(\boldsymbol{\beta})) = v_0, g_{\boldsymbol{\theta}}(\boldsymbol{\beta}) \in [0,1]^d. \tag{7}$$

In order to solve it with PyTorch, recent work transforms Eq. (7) into an unconstrained problem [34], which includes 1) eliminating $\boldsymbol{u}$ from the physical constraint by solving the linear system $\boldsymbol{K}(g_{\boldsymbol{\theta}}(\boldsymbol{\beta}))\boldsymbol{u} = \boldsymbol{f}$; 2) enforcing the couple of constraints $\boldsymbol{V}(g_{\boldsymbol{\theta}}(\boldsymbol{\beta})) = \boldsymbol{V}_0$ and $g_{\boldsymbol{\theta}}(\boldsymbol{\beta}) \in [0,1]^d$ via reparametrization, binary search, and implicit differentiation. By contrast, using `PyGRANSO` we can directly solve Eq. (7) without any of the problem-specific tricks. For realistic design problems, $\boldsymbol{x}$ needs to be discrete-valued (can be more than 2 values in multi-material design), and the constraint $\boldsymbol{K}(g_{\boldsymbol{\theta}}(\boldsymbol{\beta}))\boldsymbol{u} = \boldsymbol{f}$ becomes nonlinear—$\boldsymbol{K}$ becomes a nonlinear operator acting on $\boldsymbol{u}$ [10, 11]. `PyGRANSO` can easily handle these general cases also, whereas the tricks used in [34] cannot be generalized.

4.3. Orthogonal recurrent neural networks (RNNs)

The exploding and vanishing gradient problem is common in RNNs, and it could occur whenever the recurrent kernel does not have unit eigenvalues [2, 4, 28, 31, 42]. Recent work proposes imposing orthogonality constraint on the associated weight matrix directly [30, 42]:

$$\min_{\boldsymbol{\theta}} \mathcal{L}\left(f_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y}\right) \quad \text{s.t.}\ \boldsymbol{W}_{hh}^\intercal(\boldsymbol{\theta})\boldsymbol{W}_{hh}(\boldsymbol{\theta}) = \boldsymbol{I}, \det \boldsymbol{W}_{hh}(\boldsymbol{\theta}) = 1, \tag{8}$$

where $f_{\boldsymbol{\theta}}$ is the RNN parameterized by $\boldsymbol{\theta}$, and $\boldsymbol{W}_{hh}$ is the recurrent kernel of the RNN (i.e., a subvector of $\boldsymbol{\theta}$). Since the constraints define the famous special orthogonal group which is a smooth manifold, manifold optimization methods can be developed to solve Eq. (8) [30, 42]. But it entails heavy mathematics from differential geometry, and hence is impractical for non-experts to implement and build on (there could be additional constraints in real applications). Again, `PyGRANSO` is able to directly handle the nonlinear constraints, whether the user recognizes or not that the constraint forms the special orthogonal group.

4.4. Knowledge-aware machine learning (KAML)

KAML concerns learning that incorporates prior knowledge, e.g., physical laws as constraints. As an example, it can take the general form of minimizing a loss subject to PDE constraints [20, 47]:

$$\begin{aligned} \min_{u(\boldsymbol{x})} \quad & \mathcal{L}(u(\boldsymbol{x})) \\ \text{s.t.} \quad & f\left(\boldsymbol{x}; \frac{\partial u}{\partial x_1}, \ldots, \frac{\partial u}{\partial x_d}; \frac{\partial^2 u}{\partial x_1 \partial x_1}, \ldots, \frac{\partial^2 u}{\partial x_1 \partial x_d}; \ldots\right) = 0, \quad \boldsymbol{x} \in \Omega \\ & \mathcal{B}\left(u, \boldsymbol{x}\right) = 0, \quad \boldsymbol{x} \in \partial\Omega \end{aligned} \tag{9}$$

Here, Ω and $\partial\Omega$ denote the domain and its boundary respectively, and $\mathcal{B}(\cdot)$ represents the boundary and/or initial conditions. If the loss is a constant, this reduces to solving the PDE problem about $u(\boldsymbol{x})$. In supervised learning scenarios, the loss could take the form of $1/N \cdot \sum_{i=1}^{N} \ell(\boldsymbol{y}_i, u(\boldsymbol{x}_i))$, where $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}$ is the training set and u is the predictor to be learned from the training set.

To solve Eq. (9), one can parametrize $u(\boldsymbol{x})$ as a neural network, i.e., $u(\boldsymbol{x}; \boldsymbol{\theta})$. This is natural in modern supervised learning, and is called physics-informed neural networks (PINNs) in the numerical PDE community [12, 16, 20, 35, 44, 52]. In numerical PDEs, classical methods use finite-difference approximations for all the partial derivatives, whereas the PINN idea directly works with continuous functions that allow partial derivatives to be computed via auto-differentiation. This "mesh-free" nature of PINNs holds the promise for high-precision solutions even for high-dimensional PDEs.

The state-of-the-art methods for solving the DNN-parametrized version of Eq. (9) use penalty methods, Lagrangian methods, and augmented Lagrangian methods [16, 20, 44, 47], which often involve delicate tuning of multiple hyperparameters and could lead to infeasible solutions. By contrast, `PyGRANSO` stops iterations by rigorous check of constraint violation and stationarity.

5. Roadmap

Although `NCVX`, with the `PyGRANSO` solver, already has many powerful features, we plan to further improve it by adding several major components: (1) **symmetric rank one (SR1)**: SR1, another major type of quasi-Newton methods, allows less stringent step-size search and tends to help escape from saddle points faster by taking advantage of negative curvature directions [19]; (2) **stochastic algorithms**: in machine learning, computing with large-scale datasets often involves finite sums with huge number of terms, calling for stochastic algorithms for reduced per-iteration cost and better scalability [60]; (3) **conic programming (CP)**: semidefinite programming and second-order cone programming, special cases of CP, are abundant in machine learning, e.g., kernel machines [73]; (4) **minimax optimization (MMO)**: MMO is an emerging modeling technique in machine learning, e.g., generative adversarial networks (GANs) [24] and multi-agent reinforcement learning [36].

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[2] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *International conference on machine learning*, pages 1120–1128. PMLR, 2016.

[3] Yu Bai, Qijia Jiang, and Ju Sun. Subgradient descent learns orthogonal dictionaries. *arXiv preprint arXiv:1810.10702*, October 2018.

[4] Nitin Bansal, Xiaohan Chen, and Zhangyang Wang. Can we gain more from orthogonality regularizations in training deep networks? *Advances in Neural Information Processing Systems*, 31, 2018.

[5] Stephen R. Becker, Emmanuel J. Candès, and Michael C. Grant. Templates for convex cone problems with applications to sparse signal recovery. *Mathematical programming computation*, 3(3):165, 2011.

[6] Nicolas Boumal, Bamdev Mishra, Pierre-Antoine Absil, and Rodolphe Sepulchre. Manopt, a Matlab toolbox for optimization on manifolds. *Journal of Machine Learning Research*, 15 (42):1455–1459, 2014.

[7] Rudy Bunel, P Mudigonda, Ilker Turkaslan, P Torr, Jingyue Lu, and Pushmeet Kohli. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, 21(2020), 2020.

[8] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. *arXiv:1608.04644*, August 2016.

[9] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, Aleksander Madry, and Alexey Kurakin. On evaluating adversarial robustness. *arXiv:1902.06705*, February 2019.

[10] Aaditya Chandrasekhar and Krishnan Suresh. Tounn: topology optimization using neural networks. *Structural and Multidisciplinary Optimization*, 63(3):1135–1149, 2021.

[11] Aaditya Chandrasekhar, Saketh Sridhara, and Krishnan Suresh. Auto: a framework for automatic differentiation in topology optimization. *Structural and Multidisciplinary Optimization*, 64(6):4355–4365, 2021.

[12] Di Chen, Yiwei Bai, Wenting Zhao, Sebastian Ament, John M Gregoire, and Carla P Gomes. Deep reasoning networks: Thinking fast and slow. *arXiv preprint arXiv:1906.00855*, 2019.

[13] IBM ILOG Cplex. V12. 1: User's manual for CPLEX. *International Business Machines Corporation*, 46(53):157, 2009.

[14] Francesco Croce and Matthias Hein. Minimally distorted adversarial examples with a fast adaptive boundary attack. In *International Conference on Machine Learning*, pages 2196–2205. PMLR, 2020.

[15] Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *International conference on machine learning*, pages 2206–2216. PMLR, 2020.

[16] Salvatore Cuomo, Vincenzo Schiano Di Cola, Fabio Giampaolo, Gianluigi Rozza, Maizar Raissi, and Francesco Piccialli. Scientific machine learning through physics-informed neural networks: Where we are and what's next. *arXiv preprint arXiv:2201.05624*, 2022.

[17] Ryan R. Curtin, Marcus Edel, Rahul Ganesh Prabhu, Suryoday Basak, Zhihao Lou, and Conrad Sanderson. The ensmallen library for flexible numerical optimization. *Journal of Machine Learning Research*, 22(166):1–6, 2021.

[18] Frank E Curtis, Tim Mitchell, and Michael L Overton. A BFGS-SQP method for nonsmooth, nonconvex, constrained optimization and its evaluation using relative minimization profiles. *Optimization Methods and Software*, 32(1):148–181, 2017.

[19] Yann N. Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 2933–2941, Cambridge, MA, USA, 2014. MIT Press.

[20] 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. *arXiv preprint arXiv:2009.07330*, 2020.

[21] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[22] Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. Exploring the landscape of spatial robustness. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1802–1811. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/engstrom19a.html.

[23] David M. Gay. The AMPL modeling language: An aid to formulating and solving optimization problems. In Mehiddin Al-Baali, Lucio Grandinetti, and Anton Purnama, editors, *Numerical analysis and optimization*, pages 95–116. Springer International Publishing, Cham, 2015. ISBN 978-3-319-17689-5.

[24] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[25] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *CoRR*, abs/1412.6572, 2015.

[26] Michael Grant, Stephen Boyd, and Yinyu Ye. CVX: Matlab software for disciplined convex programming. http://cvxr.com/cvx, 2008.

[27] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2021. URL https://www.gurobi.com.

[28] Mehrtash Harandi and Basura Fernando. Generalized backpropagation,\'{E} tude de cas: Orthogonality. *arXiv preprint arXiv:1611.05927*, 2016.

[29] Matthias Hein and Maksym Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. *arXiv:1705.08475*, May 2017.

[30] Kyle Helfrich, Devin Willmott, and Qiang Ye. Orthogonal recurrent neural networks with scaled cayley transform. In *International Conference on Machine Learning*, pages 1969–1978. PMLR, 2018.

[31] Mikael Henaff, Arthur Szlam, and Yann LeCun. Recurrent orthogonal networks and long-memory tasks. In *International Conference on Machine Learning*, pages 2034–2042. PMLR, 2016.

[32] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, 2019.

[33] Hossein Hosseini and Radha Poovendran. Semantic adversarial examples. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1614–1619, 2018.

[34] Stephan Hoyer, Jascha Sohl-Dickstein, and Sam Greydanus. Neural reparameterization improves structural optimization. *arXiv preprint arXiv:1909.04240*, 2019.

[35] Dou Huang, Haoran Zhang, Xuan Song, and Ryosuke Shibasaki. Differentiable projection for constrained deep learning. *arXiv preprint arXiv:2111.10785*, 2021.

[36] Chi Jin, Praneeth Netrapalli, and Michael Jordan. What is local optimality in nonconvex-nonconcave minimax optimization? In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4880–4889. PMLR, 13–18 Jul 2020.

[37] Guy Katz, Clark Barrett, David Dill, Kyle Julian, and Mykel Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. *arXiv:1702.01135*, February 2017.

[38] Max Kochurov, Rasul Karimov, and Serge Kozlukov. Geoopt: Riemannian optimization in PyTorch. *arXiv preprint arXiv:2005.02819*, May 2020.

[39] Cassidy Laidlaw and Soheil Feizi. Functional adversarial attacks. *Advances in neural information processing systems*, 32, 2019.

[40] Cassidy Laidlaw, Sahil Singla, and Soheil Feizi. Perceptual adversarial robustness: Defense against unseen threat models. *arXiv preprint arXiv:2006.12655*, June 2020.

[41] Sören Laue, Matthias Mitterreiter, and Joachim Giesen. GENO – GENeric Optimization for classical machine learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[42] Mario Lezcano-Casado and David Martınez-Rubio. Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group. In *International Conference on Machine Learning*, pages 3794–3803. PMLR, 2019.

[43] Johan Lofberg. YALMIP : a toolbox for modeling and optimization in MATLAB. In *2004 IEEE International Conference on Robotics and Automation (IEEE Cat. No.04CH37508)*, pages 284–289. IEEE, 2004. doi: 10.1109/cacsd.2004.1393890.

[44] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. Deepxde: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021.

[45] Zhaoyang Lyu, Ching-Yun Ko, Zhifeng Kong, Ngai Wong, Dahua Lin, and Luca Daniel. Fastened crown: Tightened neural network robustness certificates. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5037–5044, 2020.

[46] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[47] Levi McClenny and Ulisses Braga-Neto. Self-adaptive physics-informed neural networks using a soft attention mechanism. *arXiv preprint arXiv:2009.04544*, 2020.

[48] Mayank Meghwanshi, Pratik Jawanpuria, Anoop Kunchukuttan, Hiroyuki Kasai, and Bamdev Mishra. McTorch, a manifold optimization library for deep learning. *arXiv preprint arXiv:1810.01811*, October 2018.

[49] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Zaharia, and Ameet Talwalkar. MLlib: Machine learning in Apache Spark. *Journal of Machine Learning Research*, 17(34):1–7, 2016.

[50] Nina Miolane, Nicolas Guigui, Alice Le Brigant, Johan Mathe, Benjamin Hou, Yann Thanwerdas, Stefan Heyder, Olivier Peltre, Niklas Koep, Hadi Zaatiti, Hatem Hajri, Yann Cabanes, Thomas Gerald, Paul Chauchat, Christian Shewmake, Daniel Brooks, Bernhard Kainz, Claire Donnat, Susan Holmes, and Xavier Pennec. Geomstats: A Python package for Riemannian geometry in machine learning. *Journal of Machine Learning Research*, 21(223):1–9, 2020.

[51] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. *arXiv:1511.04599.*, November 2015.

[52] Yatin Nandwani, Abhishek Pathak, and Parag Singla. A primal dual formulation for deep learning with constraints. *Advances in Neural Information Processing Systems*, 32, 2019.

[53] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

[54] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[55] Gianni Pillo and Massimo Roma. *Large-Scale Nonlinear Optimization*, volume 83 of *Nonconvex Optimization and Its Applications*. Springer Science & Business Media, Boston, MA, 2006.

[56] Maura Pintor, Fabio Roli, Wieland Brendel, and Battista Biggio. Fast minimum-norm adversarial attacks through adaptive norm constraints. *Advances in Neural Information Processing Systems*, 34, 2021.

[57] Jérôme Rony, Luiz G Hafemann, Luiz S Oliveira, Ismail Ben Ayed, Robert Sabourin, and Eric Granger. Decoupling direction and norm for efficient gradient-based l2 adversarial attacks and defenses. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4322–4330, 2019.

[58] Suvrit Sra, Sebastian Nowozin, and Stephen J. Wright. *Optimization for Machine Learning*. The MIT Press, Cambridge, MA, 2012.

[59] Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. OSQP: an operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 2020. doi: 10.1007/s12532-020-00179-2.

[60] Ruoyu Sun. Optimization for deep learning: theory and algorithms. *arXiv preprint arXiv:1912.08957*, December 2019.

[61] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[62] Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. *arXiv:1711.07356*, November 2017.

[63] Kim-Chuan Toh, Michael J. Todd, and Reha H. Tütüncü. SDPT3—a Matlab software package for semidefinite programming, Version 1.3. *Optimization Methods and Software*, 11(1-4): 545–581, 1999.

[64] James Townsend, Niklas Koep, and Sebastian Weichwald. Pymanopt: A Python toolbox for optimization on manifolds using automatic differentiation. *Journal of Machine Learning Research*, 17(137):1–5, 2016.

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

[66] Lily Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane Boning, and Inderjit Dhillon. Towards fast computation of certified robustness for relu networks. In *International Conference on Machine Learning*, pages 5276–5285. PMLR, 2018.

[67] Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, Dong Su, Yupeng Gao, Cho-Jui Hsieh, and Luca Daniel. Evaluating the robustness of neural networks: An extreme value theory approach. *arXiv preprint arXiv:1801.10578*, 2018.

[68] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, Burlington, MA, third edition, 2011.

[69] Eric Wong, Frank R. Schmidt, and J. Zico Kolter. Wasserstein adversarial examples via projected sinkhorn iterations. *arXiv:1902.07906*, February 2019.

[70] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1): 25–57, 2006. doi: 10.1007/s10107-004-0559-y.

[71] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. *Advances in neural information processing systems*, 31, 2018.

[72] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 586–595, 2018.

[73] Richard Y. Zhang, Cédric Josz, and Somayeh Sojoudi. Conic optimization for control, energy systems, and machine learning: Applications and algorithms. *Annual Reviews in Control*, 47: 323–340, 2019.