Deep Learning with Constraints and Nonsmoothness

Ju Sun
Computer Science & Engineering
University of Minnesota

Thanks to

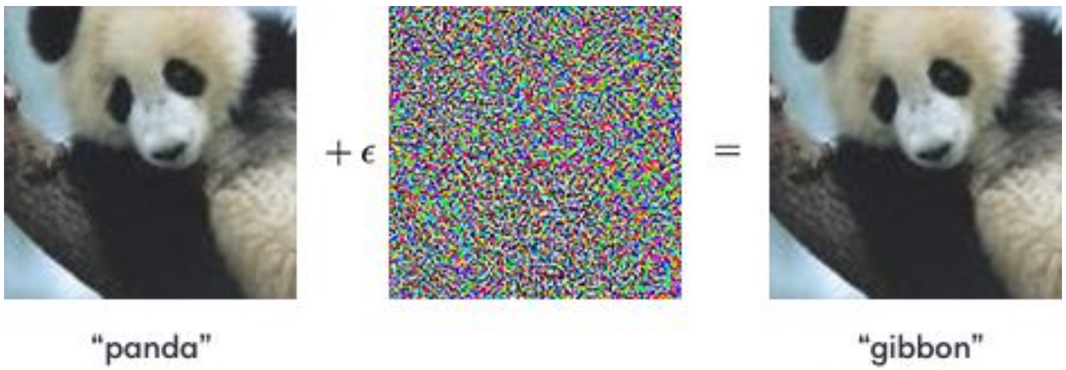
Buyun Liang
(Master, CS&E, UMN)

Hengyue Liang
(PhD, ECE, UMN)

Prof. Ying Cui
(ISYE, UMN)

Dr. Tim Mitchell
(MPI Magdeburg → Queens College, CUNY)

A quick example



$\boldsymbol{x}$ $\quad$ $\boldsymbol{\delta}$ $\quad$ $\boldsymbol{x}'$

$f_{\boldsymbol{\theta}}$: deep learning model

$$\max_{\boldsymbol{x}'} \quad \ell(\boldsymbol{y}, f_{\boldsymbol{\theta}}(\boldsymbol{x}'))$$ ⟸ Try to change the label

$$\text{s.t.} \quad d(\boldsymbol{x}, \boldsymbol{x}') \leq \epsilon$$ ⟸ Small perturbation

$$\boldsymbol{x}' \in [0, 1]^n$$ ⟸ Still a valid image

A quick example

f_θ : deep learning model

How to solve

$$\max_{x'} \quad \ell(y, f_\theta(x'))$$
$$\text{s.t.} \quad d(x, x') \leq \epsilon$$
$$x' \in [0,1]^n$$

Tensorflow VS Pytorch

Unconstrained optimization only

$$d = \ell_1, \ell_2, \ell_\infty$$

Projected gradient methods

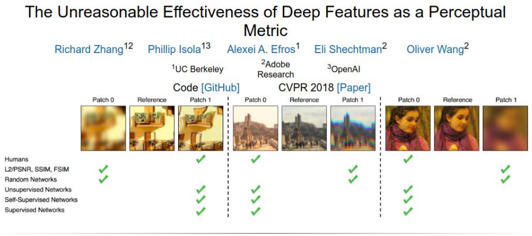
ROBUSTBENCH

A standardized benchmark for adversarial robustness

Simple d only

https://robustbench.github.io/

$$d(x, x') = \|g(x) - g(x')\| \text{ ???}$$



Perceptual metrics

A quick example

NCVX https://ncvx.org/

f_θ : deep learning model

$$\max_{x'} \quad \ell(y, f_\theta(x'))$$

$$\text{s.t.} \quad d(x, x') \leq \epsilon$$

```python
def comb_fn(X_struct):
    # obtain optimization variables
    x_prime = X_struct.x_prime
    # objective function
    f = loss_func(y, f_theta(x_prime))
    # inequality constraint
    ci = pygransoStruct()
    ci.c1 = d(x,x_prime) - epsilon
    # equality constraint
    ce = None
    return [f,ci,ce]
# specify optimization variable (tensor)
var_in = {"x_prime": list(x.shape)}
# pygranso main algorithm
soln = pygranso(var_in,comb_fn)
```

```
PyGRANSO: A PyTorch-enabled port of GRANSO with auto-differentiation
Version 1.2.0
Licensed under the AGPLv3, Copyright (C) 2021-2022 Tim Mitchell and Buyun Liang

Problem specifications:
 # of variables                     :   3072
 # of inequality constraints        :      1
 # of equality constraints          :      0

     | <--- Penalty Function --> |                | Total Violation | <--- Line Search ---> | <- Stationarity ->
Iter |    Mu    |     Value      |   Objective    |   Ineq   |  Eq  | SD | Evals |     t    | Grads |   Value
   0 | 1.000000 |  0.50909392739 |  0.50909392739 | 0.000000 |   -  | -  |     1 | 0.000000 |     1 | 0.752374
   1 | 1.000000 |  0.13139580076 |  0.13139580076 | 0.000000 |   -  | S  |     1 | 1.000000 |     1 | 0.569790
   2 | 1.000000 | -0.01353464659 | -0.11231957917 | 0.098785 |   -  | S  |     1 | 1.000000 |     1 | 0.364019
   3 | 1.000000 | -0.03305865013 | -0.23057790915 | 0.197519 |   -  | S  |     2 | 0.500000 |     1 | 0.391431
   4 | 1.000000 | -0.07512002132 | -0.36057738209 | 0.285457 |   -  | S  |     2 | 0.500000 |     1 | 0.436613
   5 | 1.000000 | -0.11627463660 | -0.49942430204 | 0.383150 |   -  | S  |     1 | 1.000000 |     1 | 0.349554
   6 | 1.000000 | -0.17664891011 | -0.58140452016 | 0.404756 |   -  | S  |     1 | 1.000000 |     1 | 0.484603
   7 | 1.000000 | -0.29911013557 | -0.68329983929 | 0.384190 |   -  | S  |     1 | 1.000000 |     1 | 0.357295
   8 | 1.000000 | -0.35472121343 | -0.75604941979 | 0.401328 |   -  | S  |     1 | 1.000000 |     1 | 0.760431
   9 | 1.000000 | -0.46871129466 | -0.80585740448 | 0.337146 |   -  | S  |     1 | 1.000000 |     1 | 1.010051
  10 | 1.000000 | -0.50872108700 | -0.88639971711 | 0.377679 |   -  | S  |     1 | 1.000000 |     1 | 0.548706
  11 | 1.000000 | -0.62616245130 | -0.88869577749 | 0.262533 |   -  | S  |     1 | 1.000000 |     1 | 0.638738
  12 | 1.000000 | -0.64267769247 | -0.93742006917 | 0.294742 |   -  | S  |     1 | 1.000000 |     1 | 0.870128
  13 | 1.000000 | -0.74731841034 | -0.98236757218 | 0.235049 |   -  | S  |     1 | 1.000000 |     1 | 0.606221
  14 | 1.000000 | -0.79218506008 | -1.00000000000 | 0.207815 |   -  | S  |     2 | 0.500000 |     1 | 0.514724
  15 | 1.000000 | -0.83822978425 | -1.00000000000 | 0.161770 |   -  | S  |     1 | 1.000000 |     1 | 0.260043
  16 | 1.000000 | -0.85468479749 | -0.94460724778 | 0.089922 |   -  | S  |     1 | 1.000000 |     1 | 1.303794
  17 | 1.000000 | -0.91321142003 | -0.99960270319 | 0.086391 |   -  | S  |     1 | 1.000000 |     1 | 0.601452
  18 | 1.000000 | -0.92405471150 | -1.00000000000 | 0.075945 |   -  | S  |     3 | 0.250000 |     1 | 0.292231
  19 | 1.000000 | -0.95860778204 | -1.00000000000 | 0.041392 |   -  | S  |     1 | 1.000000 |     1 | 0.106619

     | <--- Penalty Function --> |                | Total Violation | <--- Line Search ---> | <- Stationarity ->
Iter |    Mu    |     Value      |   Objective    |   Ineq   |  Eq  | SD | Evals |     t    | Grads |   Value
  20 | 1.000000 | -0.97443979631 | -0.98763548475 | 0.013196 |   -  | S  |     1 | 1.000000 |     1 | 1.048798
  21 | 1.000000 | -0.98244681692 | -1.00000000000 | 0.017553 |   -  | S  |     3 | 0.250000 |     1 | 0.052289
  22 | 1.000000 | -0.99989348389 | -0.99989348389 | 0.000000 |   -  | S  |     2 | 2.000000 |     1 | 0.409106
  23 | 1.000000 | -1.00000000000 | -1.00000000000 | 0.000000 |   -  | S  |     3 | 0.250000 |     1 | 0.000000

Optimization results:
F = final iterate, B = Best (to tolerance), MF = Most Feasible

   F |          |                | -1.00000000000 | 0.000000 |   -  |    |       |          |       |
   B |          |                | -1.00000000000 | 0.000000 |   -  |    |       |          |       |
  MF |          |                | -1.00000000000 | 0.000000 |   -  |    |       |          |       |

Iterations:              23
Function evaluations:    34
PyGRANSO termination code: 0 --- converged to stationarity and feasibility tolerances.
```

Motivation

Example 1: Robustness of deep models

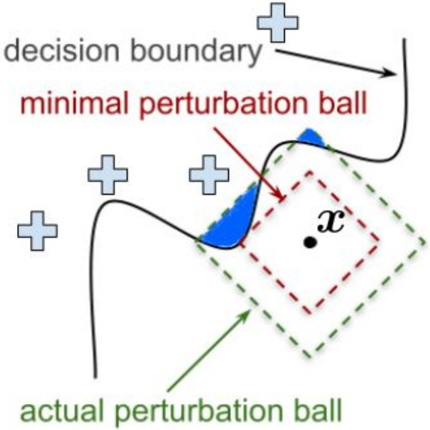
f_θ : deep learning model

Robustness in adversarial settings

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

Robustness limit of deep models **(safety radius)**

$$\min_{\boldsymbol{x}'} \quad d(\boldsymbol{x}, \boldsymbol{x}')$$
$$\text{s. t.} \quad \max_{\ell \neq c} f_{\boldsymbol{\theta}}^{\ell}(\boldsymbol{x}') \geq f_{\boldsymbol{\theta}}^{c}(\boldsymbol{x}')$$
$$\boldsymbol{x}' \in [0,1]^n$$



RobustBench
A standardized benchmark for adversarial robustness

https://robustbench.github.io/

- SOTA PGD-based methods only for $d = \ell_1, \ell_2, \ell_\infty$
- Hand-optimized step-size schedule

Example 2: Physics-informed neural networks (PINNs)

$$f\left(\mathbf{x}; \frac{\partial u}{\partial x_1}, \ldots, \frac{\partial u}{\partial x_d}; \frac{\partial^2 u}{\partial x_1 \partial x_1}, \ldots, \frac{\partial^2 u}{\partial x_1 \partial x_d}; \ldots; \boldsymbol{\lambda}\right) = 0, \quad \mathbf{x} \in \Omega,$$

PDE

$$\mathcal{B}(u, \mathbf{x}) = 0 \quad \text{on} \quad \partial\Omega$$

Boundary Condition

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{T}) = \underbrace{w_f}_{\text{penalty parameters}} \overbrace{\mathcal{L}_f(\boldsymbol{\theta}; \mathcal{T}_f)}^{\text{objective}} + \underbrace{w_b}_{\text{penalty parameters}} \overbrace{\mathcal{L}_b(\boldsymbol{\theta}; \mathcal{T}_b)}^{\text{constraint}}$$

$$\mathcal{L}_f(\boldsymbol{\theta}; \mathcal{T}_f) = \frac{1}{|\mathcal{T}_f|} \sum_{\mathbf{x} \in \mathcal{T}_f} \left\| f\left(\mathbf{x}; \frac{\partial \hat{u}}{\partial x_1}, \ldots, \frac{\partial \hat{u}}{\partial x_d}; \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_1}, \ldots, \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_d}; \ldots; \boldsymbol{\lambda}\right) \right\|_2^2$$

$$\mathcal{L}_b(\boldsymbol{\theta}; \mathcal{T}_b) = \frac{1}{|\mathcal{T}_b|} \sum_{\mathbf{x} \in \mathcal{T}_b} \|\mathcal{B}(\hat{u}, \mathbf{x})\|_2^2,$$

- Auto-differentiation replaces finite-difference (mesh-free)
- Potential for **efficiently** solving high-dimensional problems
- **SOTA: penalty methods (recently Lagrangian methods)**

Ref: Lu, Lu, et al. "DeepXDE: A deep learning library for solving differential equations." SIAM Review 63.1 (2021): 208-228.

Observations

- People try to avoid complicated constraints

- If constraints cannot be avoided, people will do naive things — penalty methods (e.g., Lagrangian methods)

General-purpose solvers

Solvers	Nonconvex	Nonsmooth	Differentiable manifold constraints	General smooth constraint	Specific constrained ML problem
SDPT3, Gurobi, Cplex, TFOCS, CVX(PY), AMPL, YALMIP	❌	✔	❌	❌	❌
PyTorch, Tensorflow	✔	✔	❌	❌	❌
(Py)manopt, Geomstats, McTorch, Geoopt, GeoTorch	✔	✔	✔	❌	❌
KNITRO, IPOPT, GENO, ensmallen	✔	✔	✔	✔	❌
Scikit-learn, MLib, Weka	✔	✔	❌	❌	✔

NCVX PyGRANSO

https://ncvx.org/

Key algorithm

GRANSO

http://www.timmitchell.com/software/GRANSO/

Nonconvex, nonsmooth, constrained

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

Exact penalty method

$$\phi(x;\mu) = \mu f(x) + v(x) \qquad v(x) = \|\max\{c(x), 0\}\|_1 = \sum_{i\in\mathcal{P}_x} c_i(x) \quad \text{where } \mathcal{P}_x = \{i \in \{1,\dots,p\} : c_i(x) > 0\}$$

Penalty sequential quadratic programming (P-SQP)

$$\begin{aligned} \min_{d\in\mathbb{R}^n,\ s\in\mathbb{R}^p} \quad & \mu(f(x_k) + \nabla f(x_k)^\mathsf{T} d) + e^\mathsf{T} s + \frac{1}{2} d^\mathsf{T} H_k d \\ \text{s.t.} \quad & c(x_k) + \nabla c(x_k)^\mathsf{T} d \le s, \quad s \ge 0, \end{aligned}$$

Ref: ***Curtis, Frank E., 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 (2017): 148-181.*

Algorithm highlights

Steering strategy for the penalty parameter

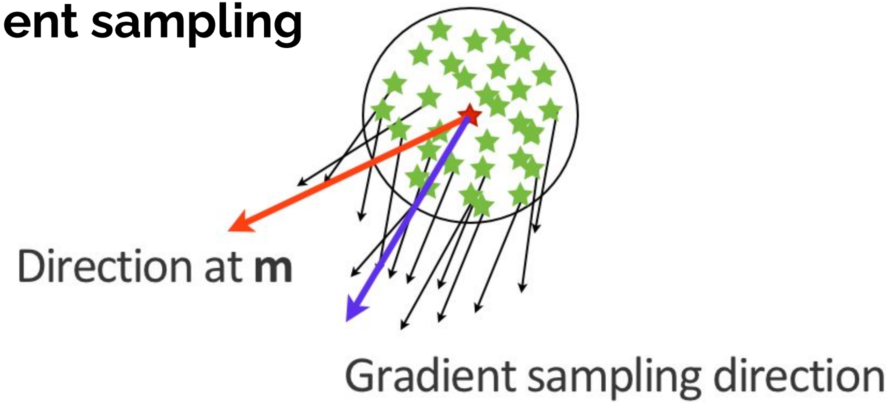
If feasibility improvement is insufficient : $l_\delta(d_k; x_k) < c_v v(x_k)$

Stationarity based on (approximate) gradient sampling

$$G_k := [\nabla f(x^k) \quad \nabla f(x^{k,1}) \quad \cdots \quad \nabla f(x^{k,m})]$$

$$\min_{\lambda \in \mathbb{R}^{m+1}} \ \frac{1}{2}\|G_k \lambda\|_2^2$$

$$\text{s.t. } \mathbb{1}^T \lambda = 1, \ \lambda \geq 0$$



Limitations of GRANSO

GRANSO

http://www.timmitchell.com/software/GRANSO/

```
% Gradient of inner product with respect to A
f_grad        = imag((conj(Bty)*Cx.')/(y'*x));
f_grad        = f_grad(:);

% Gradient of inner product with respect to A
ci_grad       = real((conj(Bty)*Cx.')/(y'*x));
ci_grad       = ci_grad(:);
```

analytical gradients required

```
p             = size(B,2);
m             = size(C,1);
X             = reshape(x,p,m);
```

vector variables only

Lack of Auto-Differentiation

Lack of GPU Support

No native support of tensor variables

⇒ impossible to do deep learning with GRANSO

NCVX PyGRANSO

NCVX https://ncvx.org/

1) Auto-Differentiation

Orthogonal Dictionary Learning (ODL)

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

Analytical gradients

```
function[f,fg,ci,cig,ce,ceg]=comb_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,comb_fn);
```

Demo 1: GRANSO for ODL

```
def comb_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]} # define variable
soln = pygranso(var_in,comb_fn)
```

No Analytical gradients

Demo 2: PyGRANSO for ODL

NCVX PyGRANSO

NCVX https://ncvx.org/

2) GPU acceleration for large scale problems

Orthogonality-constrained RNN

GPU: ~7.2 s for 100 iter

```
PyGRANSO: A PyTorch-enabled port of GRANSO with auto-differentiation
Version 1.2.0
Licensed under the AGPLv3, Copyright (C) 2021-2022 Tim Mitchell and Buyun Liang

Problem specifications:
 # of variables                     :   48010
 # of inequality constraints        :       0
 # of equality constraints          :       1

Limited-memory mode enabled with size = 20.
NOTE: limited-memory mode is generally NOT
recommended for nonsmooth problems.

    | <--- Penalty Function --> |                | Total Violation | <--- Line Search ---> | <- Stationarity -> |
Iter|    Mu    |     Value      |   Objective    | Ineq |    Eq    | SD | Evals |     t    | Grads |   Value    |
   0| 100.0000 | 231.110993915  | 2.31110993915  |  -   | 2.20e-14 | -  |     1 | 0.000000 |     1 | 70.02796   |
  10| 2.781284 | 6.15638768205  | 1.83942004642  |  -   | 1.040438 | S  |     3 | 4.000000 |     1 | 0.087806   |
  20| 1.077526 | 2.42602824202  | 1.85198233657  |  -   | 0.430468 | S  |     1 | 1.000000 |     1 | 0.009331   |
  30| 0.785517 | 1.62062600991  | 1.84414672987  |  -   | 0.172018 | S  |     3 | 4.000000 |     1 | 0.070584   |
  40| 0.785517 | 1.49341762439  | 1.86152922129  |  -   | 0.031155 | S  |     1 | 1.000000 |     1 | 0.008798   |
  50| 0.785517 | 1.45863093500  | 1.84741128366  |  -   | 0.007458 | S  |     1 | 1.000000 |     1 | 0.021082   |
  60| 0.375710 | 0.65961089292  | 1.74086384511  |  -   | 0.005551 | S  |     1 | 1.000000 |     1 | 0.060758   |
  70| 0.375710 | 0.61368640026  | 1.62370589875  |  -   | 0.003644 | S  |     1 | 1.000000 |     1 | 0.004978   |
  80| 0.221853 | 0.34727899398  | 1.53948397613  |  -   | 0.005740 | S  |     3 | 4.000000 |     1 | 0.060639   |
  90| 0.221853 | 0.33322515702  | 1.49579349213  |  -   | 0.001379 | S  |     3 | 0.250000 |     1 | 0.007120   |
 100| 0.221853 | 0.32795759434  | 1.47195631688  |  -   | 0.001399 | S  |     1 | 1.000000 |     1 | 0.020357   |

F = final iterate, B = Best (to tolerance), MF = Most Feasible
Optimization results:

   F|          |                | 1.47195631688  |  -   | 0.001399 |    |       |          |       |            |
   B|          |                | 2.31110993915  |  -   | 2.20e-14 |    |       |          |       |            |
  MF|          |                | 2.31110993915  |  -   | 2.20e-14 |    |       |          |          |       |            |

Iterations:              100
Function evaluations:    148
PyGRANSO termination code: 4 --- max iterations reached.

Total Wall Time: 7.24015998840332s
```

CPU: ~17.6 s for 100 iter

```
PyGRANSO: A PyTorch-enabled port of GRANSO with auto-differentiation
Version 1.2.0
Licensed under the AGPLv3, Copyright (C) 2021-2022 Tim Mitchell and Buyun Liang

Problem specifications:
 # of variables                     :   48010
 # of inequality constraints        :       0
 # of equality constraints          :       1

Limited-memory mode enabled with size = 20.
NOTE: limited-memory mode is generally NOT
recommended for nonsmooth problems.

    | <--- Penalty Function --> |                | Total Violation | <--- Line Search ---> | <- Stationarity -> |
Iter|    Mu    |     Value      |   Objective    | Ineq |    Eq    | SD | Evals |     t    | Grads |   Value    |
   0| 100.0000 | 234.459429436  | 2.32459429436  |  -   | 2.000000 | -  |     1 | 0.000000 |     1 | 84.45258   |
  10| 3.815204 | 7.61543767313  | 1.40009010433  |  -   | 2.273808 | S  |     2 | 2.000000 |     1 | 0.035876   |
  20| 1.478088 | 3.36426539638  | 1.18836657672  |  -   | 1.607755 | S  |     2 | 2.000000 |     1 | 0.039562   |
  30| 1.077526 | 2.36764743644  | 0.91175234571  |  -   | 1.385210 | S  |     3 | 4.000000 |     1 | 0.122148   |
  40| 0.572642 | 1.74741425646  | 0.89840965347  |  -   | 1.232947 | S  |     2 | 2.000000 |     1 | 0.031523   |
  50| 0.338139 | 1.47480385659  | 0.89839455270  |  -   | 1.171021 | S  |     1 | 1.000000 |     1 | 0.022276   |
  60| 0.221853 | 1.34061810155  | 0.87445955456  |  -   | 1.146617 | S  |     3 | 4.000000 |     1 | 0.027327   |
  70| 0.117902 | 1.23339816968  | 0.81824319678  |  -   | 1.136926 | S  |     2 | 2.000000 |     1 | 0.029037   |
  80| 0.077355 | 1.19367524620  | 0.80101161581  |  -   | 1.131713 | S  |     1 | 1.000000 |     1 | 0.005670   |
  90| 0.062658 | 1.17924531645  | 0.77380656707  |  -   | 1.130760 | S  |     1 | 1.000000 |     1 | 0.007646   |
 100| 0.029969 | 1.15301795796  | 0.76665514964  |  -   | 1.130042 | S  |     2 | 2.000000 |     1 | 0.003602   |

F = final iterate, B = Best (to tolerance), MF = Most Feasible
Optimization results:

   F|          |                | 0.76665514964  |  -   | 1.130042 |    |       |          |       |            |
  MF|          |                | 0.76665514964  |  -   | 1.130042 |    |       |          |       |            |

Iterations:              100
Function evaluations:    182
PyGRANSO termination code: 4 --- max iterations reached.

Total Wall Time: 17.563776016235355s
```

NCVX PyGRANSO

NCVX https://ncvx.org/

3) General Tensor Variables

```
var_in = {"x1": [1], "x2": [1]}
```

Scalar input

```
# objective function
f = (8 * abs(x1**2 - x2) + (1 - x1)**2)
```

```
var_in = {"q": [n,1]}
```

Vector input

```
# objective function
qtY = q.T @ Y
f = 1/m * torch.norm(qtY, p = 1)
```

```
var_in = {"M": [d1,d2],"S": [d1,d2]}
```

Matrix inputs

```
# objective function
f = torch.norm(M, p = 'nuc') + eta * torch.norm(S, p = 1)
```

```
var_in = {"x_tilde": list(inputs.shape)}
```

Higher order tensor input

```
adv_inputs = X_struct.x_tilde
epsilon = eps
logits_outputs = model(adv_inputs)
f = -torch.nn.functional.cross_entropy(logits_outputs,labels)
```

Example 1: Support Vector Machine (SVM)

Soft-margin SVM

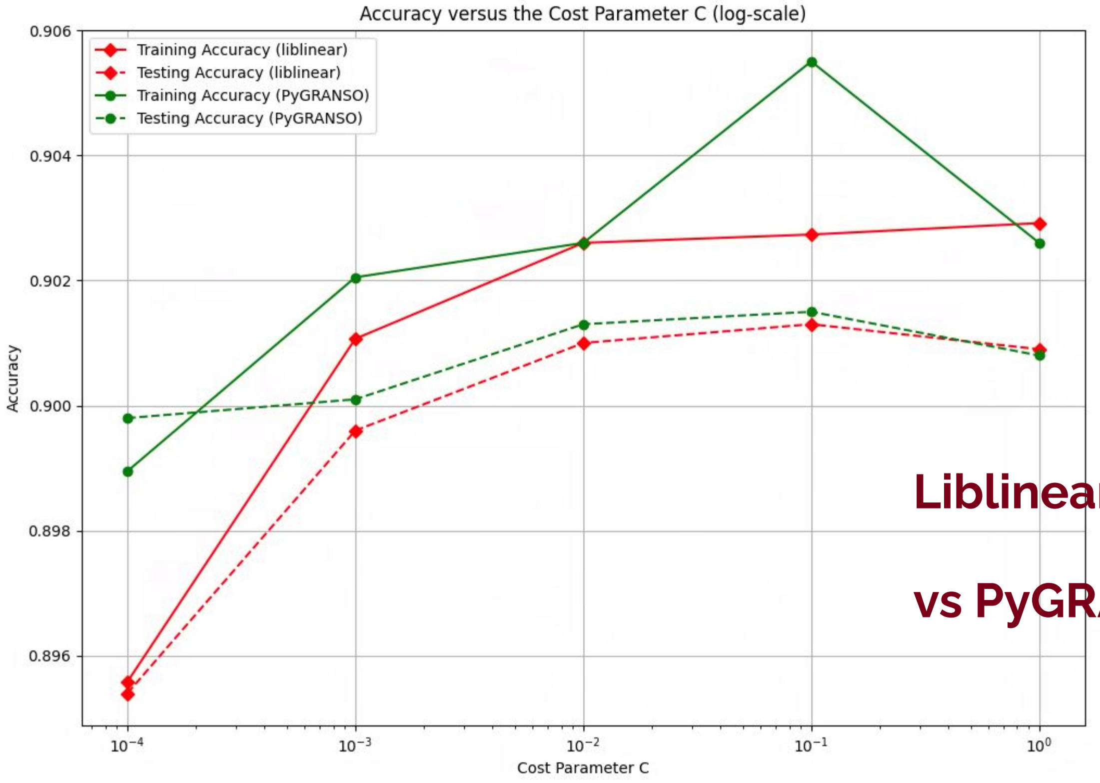
$$\min_{\boldsymbol{w},b,\zeta} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\zeta_i$$

$$\text{s.t.} \quad y_i\left(\boldsymbol{w}^\intercal \boldsymbol{x}_i + b\right) \geq 1 - \zeta_i,\ \zeta_i \geq 0\ \ \forall i = 1, \ldots, n$$

```
def comb_fn(X_struct):
    # obtain optimization variables
    w = X_struct.w
    b = X_struct.b
    zeta = X_struct.zeta
    # objective function
    f = 0.5*w.T@w + C*torch.sum(zeta)
    # inequality constraints
    ci = pygransoStruct()
    ci.c1 = 1 - zeta - y*(x@w+b)
    ci.c2 = -zeta
    # equality constraint
    ce = None
    return [f,ci,ce]
# specify optimization variables
var_in = {"w": [d,1], "b": [1,1], "zeta": [n,1]}
# pygranso main algorithm
soln = pygranso(var_in,comb_fn)
```

Support Vector Machine

Binary classification (odd vs even digits) on MNIST dataset



Liblinear (coordinate descent)

vs PyGRANSO

Example 2: Orthogonal RNN

```python
def comb_fn(rnn_model):
    # obtain weights of recurrent kernel
    W_hh = list(rnn_model.parameters())[1]
    # objective function
    f = CrossEntropyLoss(rnn_model(x), y)
    # inequality constraint
    ci = None
    # equality constraints
    ce = pygransoStruct()
    ce.c1 = W_hh.T@W_hh-torch.eye(hidden_size)
    ce.c2 = torch.det(W_hh) - 1
    return [f,ci,ce]
# optimization variables is inside "rnn_model" (torch.nn)
# pygranso main algorithm
soln = pygranso(rnn_model,comb_fn)
```

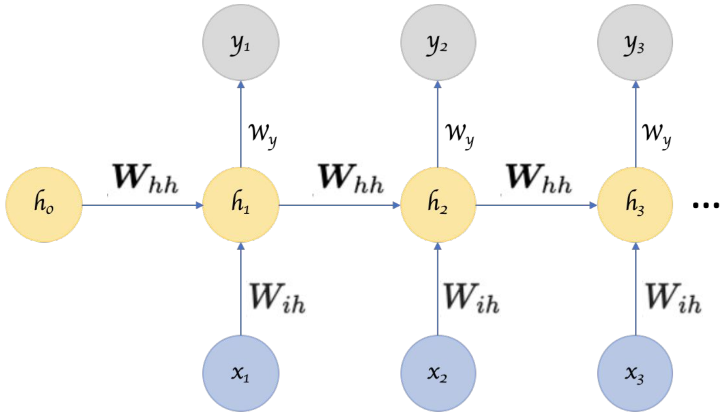
$$\min_{\theta} \left(f_\theta(x), y\right)$$

$$\boxed{\text{s.t.} \quad \boldsymbol{W}_{hh}^\intercal \boldsymbol{W}_{hh} = \boldsymbol{I}; \quad \det(\boldsymbol{W}_{hh}) = 1}$$

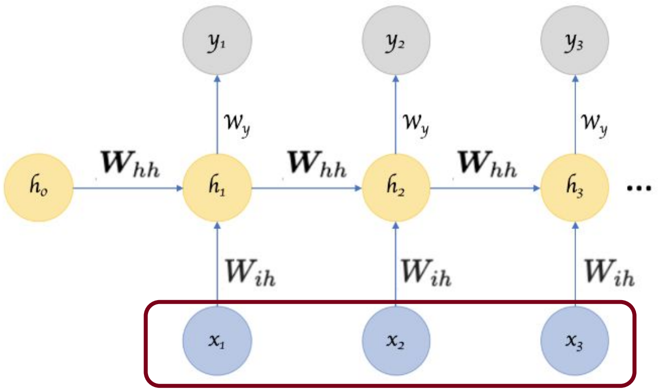
$\boldsymbol{W}_{hh}$ is from the weights of recurrent kernel (subvector of θ)

Each layer computes the following function

$$h_t = \tanh(W_{ih}x_t + b_{ih} + W_{hh}h_{t-1} + b_h h)$$



Orthogonal RNN



1st, 2nd and 3rd row of the 28X28 image

Compared with:

GeoTorch: A gradient-based manifold optimization method

https://github.com/Lezcano/geotorch

	Train accuracy	Test accuracy
GeoTorch	93.75%	88.60%
PyGRANSO	94.80%	89.10%

Example 3: Robustness—max formulation

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

```python
def comb_fn(X_struct):
    # obtain optimization variables
    x_prime = X_struct.x_prime
    # objective function
    f = loss_func(y, f_theta(x_prime))
    # inequality constraints
    ci = pygransoStruct()
    ci.c1 = d(x,x_prime) - epsilon
    ci.c2 = -x_prime
    ci.c3 = x_prime-1
    # equality constraint
    ce = None
    return [f,ci,ce]
# specify optimization variable (tensor)
var_in = {"x_prime": list(x.shape)}
# pygranso main algorithm
soln = pygranso(var_in,comb_fn)
```

Robustness: max formulation

$$\min_{\boldsymbol{\theta}} \ \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})} \max_{\substack{\boldsymbol{x}': d(\boldsymbol{x},\boldsymbol{x}') \le \epsilon \\ \boldsymbol{x}' \in [0,1]^n}} \ell(\boldsymbol{y}, f_{\boldsymbol{\theta}}(\boldsymbol{x}'))$$

robust accuracy:

lower means more effective attack

Dataset	Model - Attack	Clean	APGD		PWCF(ours)		Square	APGD+
			CE	M	CE	M	M	PWCF
CIFAR10	P_1 [63] - $\ell_1(12)$	73.3	0.96	0.00	28.6	0.00	2.28	**0.00**
	WRN-70-16 [64] - $\ell_2(0.5)$	94.7	81.8	81.1	81.8	81.0	87.9	**80.8**
	WRN-70-16 [64] - $\ell_\infty(0.03)$	90.8	69.4	68.0	73.6	72.8	71.6	**67.1**
ImageNet100	PAT-Alex [12] - $\ell_2(4.7)$	75.0	42.7	44.0	42.8	44.5	63.1	**40.9**
	PAT-Alex [12] - $\ell_\infty(0.016)$	75.0	48.0	48.2	56.6	48.8	59.9	**45.2**

Results taken from: Hengyue Liang, Buyun Liang, Ying Cui, Tim Mitchell, Ju Sun. On Optimization and Optimizers in Adversarial Robustness (tentative). In submission to IEEE Transactions on Pathology Analysis and Machine Intelligence, 2022.

Example 4: Robustness—min formulation

$$\begin{aligned} \min_{\boldsymbol{x}'} \quad & d(\boldsymbol{x}, \boldsymbol{x}') \\ \text{s.t.} \quad & \max_{\ell \neq c} f_{\boldsymbol{\theta}}^{\ell}(\boldsymbol{x}') \geq f_{\boldsymbol{\theta}}^{c}(\boldsymbol{x}') \\ & \boldsymbol{x}' \in [0,1]^n \end{aligned}$$

```python
def comb_fn(X_struct):
    # obtain optimization variables
    x_prime = X_struct.x_prime
    # objective function
    f = d(x,x_prime)
    # inequality constraints
    ci = pygransoStruct()
    f_theta_all = f_theta(x_prime)
    fy = f_theta_all[:,y] # true class output
    # output execpt true class
    fi = torch.hstack((f_theta_all[:,:y],f_theta_all[:,y+1:]))
    ci.c1 = fy - torch.max(fi)
    ci.c2 = -x_prime
    ci.c3 = x_prime-1
    # equality constraint
    ce = None
    return [f,ci,ce]
# specify optimization variable (tensor)
var_in = {"x_prime": list(x.shape)}
# pygranso main algorithm
soln = pygranso(var_in,comb_fn)
```

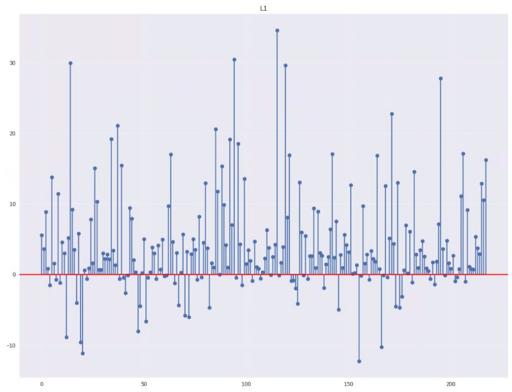
$$\min_{\boldsymbol{x}'} \quad d(\boldsymbol{x}, \boldsymbol{x}')$$
$$\text{s.t.} \quad \max_{\ell \neq c} f_{\boldsymbol{\theta}}^{\ell}(\boldsymbol{x}') \geq f_{\boldsymbol{\theta}}^{c}(\boldsymbol{x}')$$
$$\boldsymbol{x}' \in [0, 1]^n$$

Robustness: min formulation

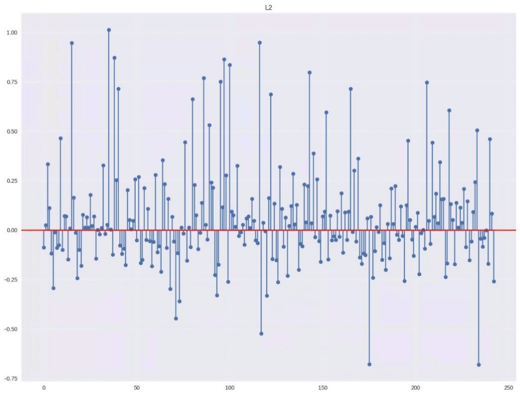
CIFAR10 dataset

Compared with FAB [iterative constraint linearization + projected gradient]
https://github.com/fra31/auto-attack

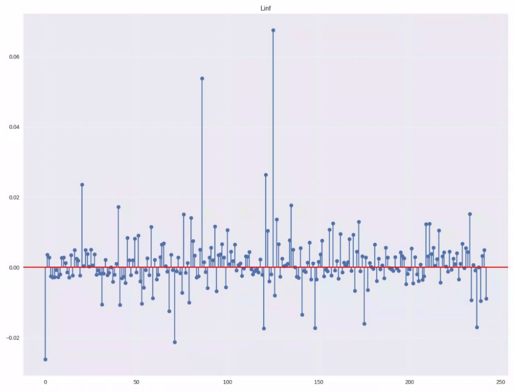
X-axis: image index; Y-axis: PyGRANSO radius - FAB radius



L1 attack



L2 attack



Linf attack

Practical & crucial tricks

- **Constraint folding**

Equality into non-negative inequality $c(x) = 0 \iff |c(x)| \leq 0$

Inequality into nonnegative inequality $c(x) \leq 0 \iff \max(c(x), 0) \leq 0$

All non-negative inequalities into one

$$c_1(x) \leq 0, \ldots, c_k(x) \leq 0 \iff \|[c_1(x), \ldots, c_k(x)]\|_2 \leq 0$$

- **Reduce # constraints**
  - Reduce cost of QP in the SQP
  - Reduce cost of AD

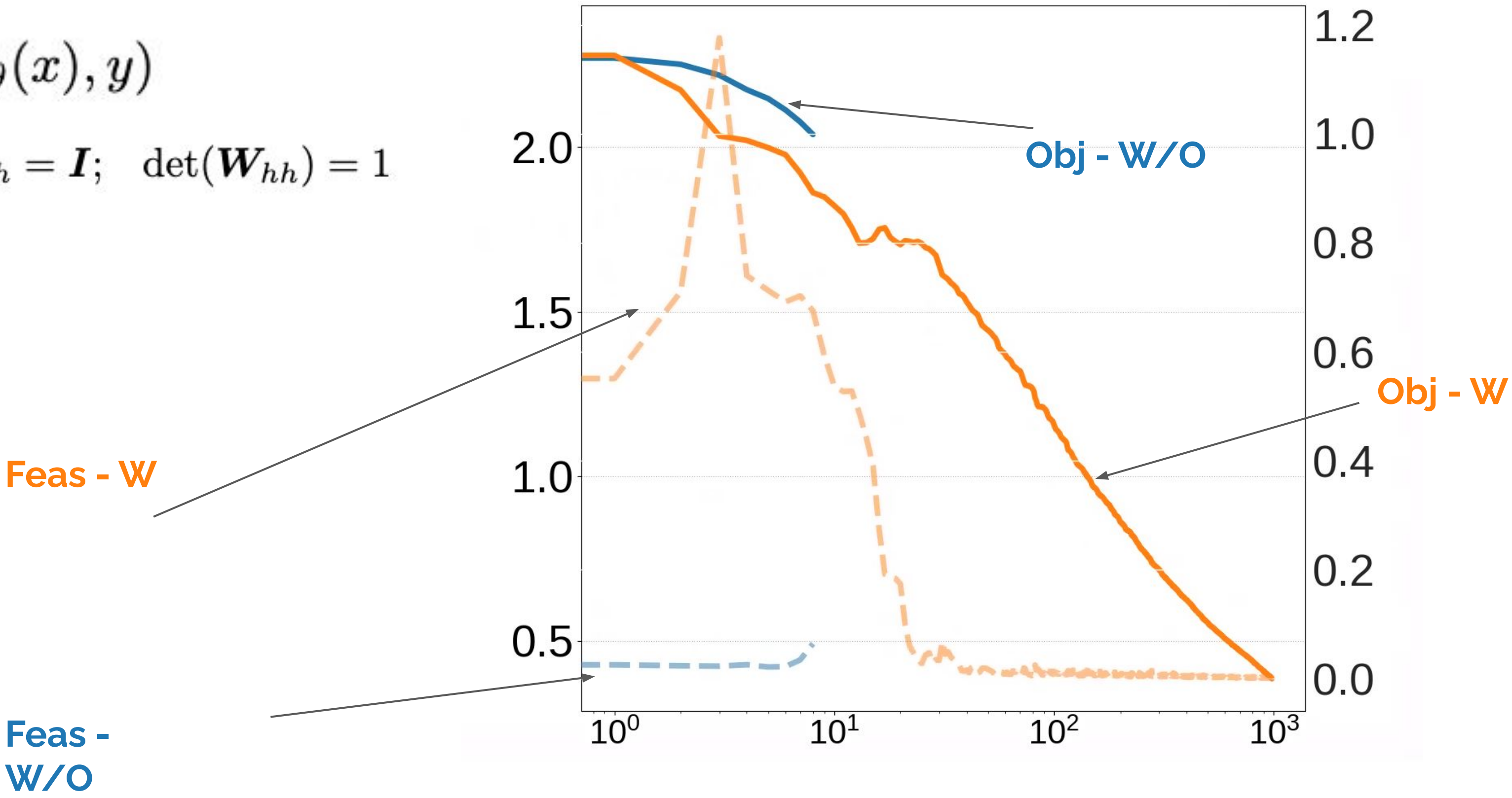
Enabled by NCVX's ability to handle nonsmoothness

Practical & crucial tricks

- **Constraint folding**

$$\min_{\theta} \left(f_\theta(x), y\right)$$

$$\text{s.t.} \;\; \boldsymbol{W}_{hh}^{\intercal}\boldsymbol{W}_{hh} = \boldsymbol{I}; \quad \det(\boldsymbol{W}_{hh}) = 1$$



Practical & crucial tricks

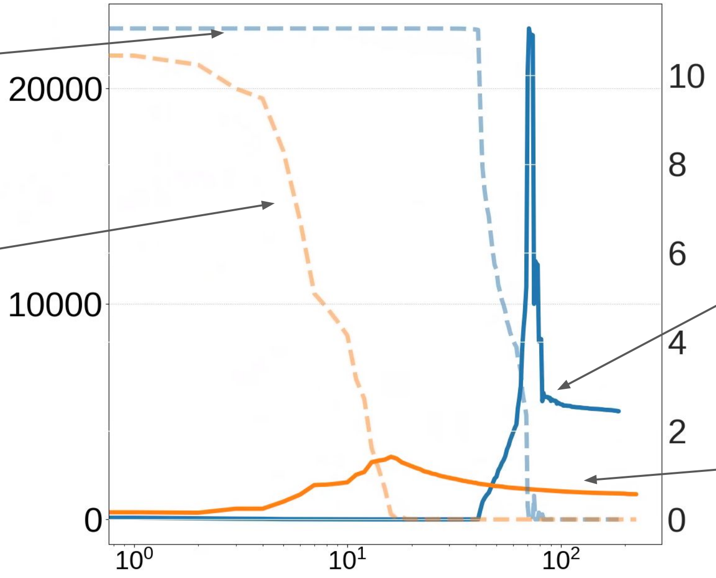
- **Rescaling**

$$\min_{\boldsymbol{x}'} \ ||\boldsymbol{x}'||_1 \quad \text{s.t.} \ \max_{i \neq y} f_{\boldsymbol{\theta}}^i(\boldsymbol{x}') \geq f_{\boldsymbol{\theta}}^y(\boldsymbol{x}') \quad \boxed{\boldsymbol{x}' \in [0,1]^n}$$

Folding scale: $O(\sqrt{n})$

$$\Rightarrow \quad \min_{\boldsymbol{x}'} \ \boxed{\frac{||\boldsymbol{x}'||_1}{\sqrt{n}}} \quad \text{s.t.} \ \max_{i \neq y} f_{\boldsymbol{\theta}}^i(\boldsymbol{x}') \geq f_{\boldsymbol{\theta}}^y(\boldsymbol{x}') \quad \boldsymbol{x}' \in [0,1]^n$$

Rescaling objective

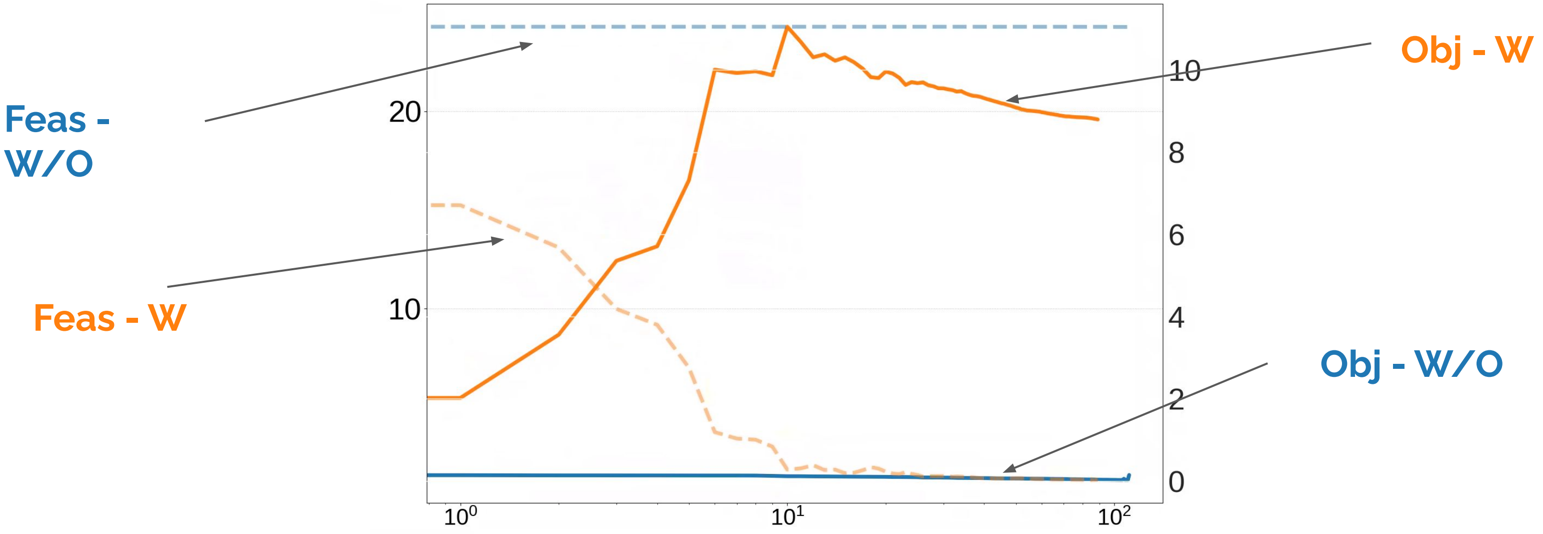


Practical & crucial tricks

- **Reformulation**

$$\begin{aligned} \min_{\boldsymbol{x}'} \quad & \|\boldsymbol{x} - \boldsymbol{x}'\|_\infty \\ \text{s.t.} \quad & \max_{\ell \neq c} f^\ell_{\boldsymbol{\theta}}(\boldsymbol{x}') \geq f^c_{\boldsymbol{\theta}}(\boldsymbol{x}') \\ & \boldsymbol{x}' \in [0,1]^n \end{aligned} \quad \Longrightarrow \quad \begin{aligned} \min_{\boldsymbol{x}'} \quad & t \\ \text{s.t.} \quad & |x_i - x'_i| \leq t \ \forall\, i \\ & \max_{\ell \neq c} f^\ell_{\boldsymbol{\theta}}(\boldsymbol{x}') \geq f^c_{\boldsymbol{\theta}}(\boldsymbol{x}') \\ & \boldsymbol{x}' \in [0,1]^n \end{aligned}$$

Folding

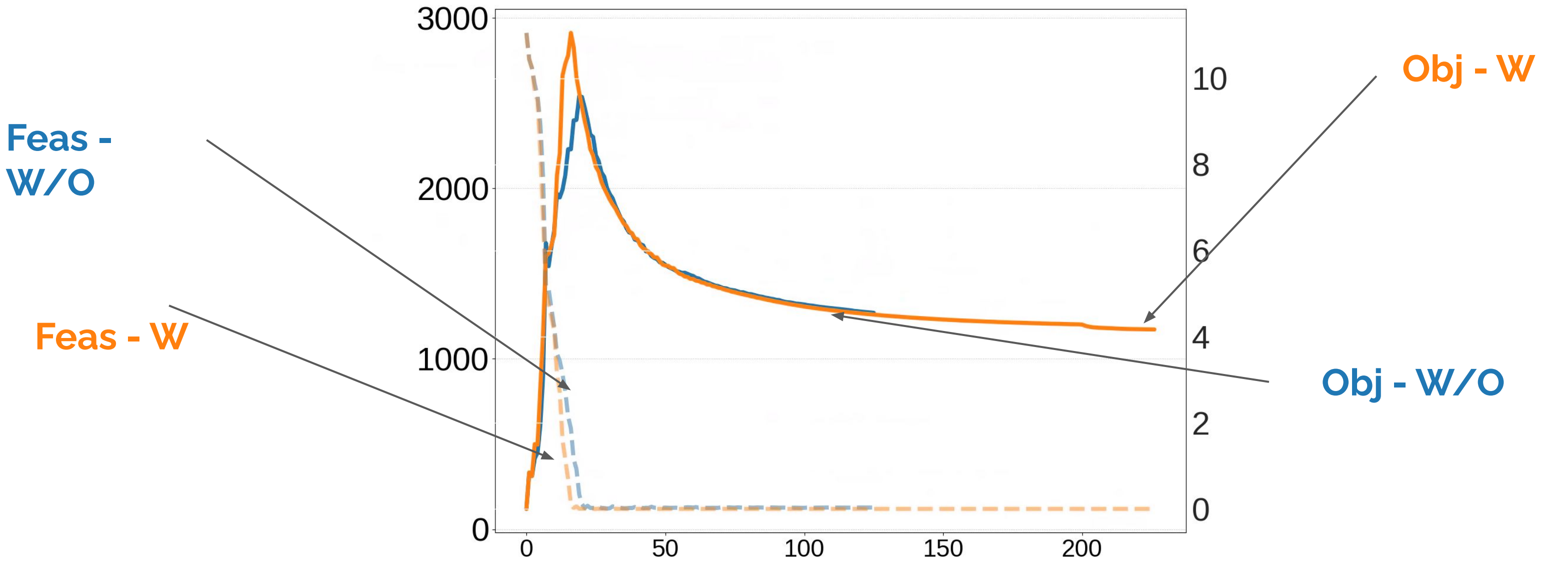


Practical & crucial tricks

- **Fold constraints into DNNs**

$$\begin{aligned} \min_{\boldsymbol{x}'} \quad & \|\boldsymbol{x} - \boldsymbol{x}'\|_\infty \\ \text{s. t.} \quad & \max_{\ell \neq c} f_{\boldsymbol{\theta}}^{\ell}(\boldsymbol{x}') \geq f_{\boldsymbol{\theta}}^{c}(\boldsymbol{x}') \\ & \boxed{\boldsymbol{x}' \in [0,1]^n} \end{aligned}$$

```
def forward(self, x):
    if self.use_clamp_input:
        x = torch.clamp(x, 0, 1)
    return self.model(x)
```



Summary

NCVX https://ncvx.org/

A solver for constrained, nonsmooth deep learning problems

- Auto-differentiation
- GPU support
- Tensor variable support

Practical tricks to speed up

- Constraint folding (into a single one)
- Objective and constraint rescaling
- Reformulation
- Build constraints into DNNs

Next steps

- Autoscaling
- Stochastic objective

References

https://ncvx.org/

- **NCVX: A User-Friendly and Scalable Package for Nonconvex Optimization in Machine Learning**.
  Buyun Liang, Tim Mitchell, Ju Sun. 2021 https://arxiv.org/abs/2111.13984
- **NCVX: A General-Purpose Optimization Solver for Machine Learning, and Practical Tricks.**
  Buyun Liang, Tim Mitchell, Ying Cui, Ju Sun. In submission to IEEE Transactions on Pattern Analysis and Machine Intelligence, 2022.
- **On Optimization and Optimizers in Adversarial Robustness (tentative)**.
  Hengyue Liang, Buyun Liang, Ying Cui, Tim Mitchell, Ju Sun. In submission to IEEE Transactions on Pattern Analysis and Machine Intelligence, 2022.