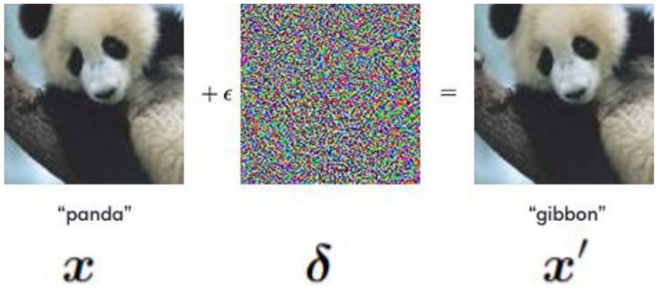
NCVX: A General-Purpose Optimization Solver for Constrained Machine and Deep Learning

Buyun Liang

Apr 7, 2023

University of Minnesota

Motivation: Trustworthy AI



Maximize loss function

$$\boxed{\max_{\boldsymbol{x}'} \ell\left(\boldsymbol{y}, f_{\boldsymbol{\theta}}(\boldsymbol{x}')\right)}$$

$$\text{s.t.} \ \boxed{d\left(\boldsymbol{x}, \boldsymbol{x}'\right) \leq \varepsilon}, \quad \boxed{\boldsymbol{x}' \in [0,1]^n}$$

Allowable perturbation with radius **ε**

Valid image constraints

Minimize robustness radius

$$\boxed{\min_{\boldsymbol{x}'} \ d\left(\boldsymbol{x}, \boldsymbol{x}'\right)}$$

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

Change the predicted class

Valid image constraints

Motivation: AI for science & engineering

Bayonne Bridge

Topology optimization results

Topology optimization

$$\boxed{\min_{\boldsymbol{x},\boldsymbol{u}} \boldsymbol{u}^\intercal \boldsymbol{K}(\boldsymbol{x})\boldsymbol{u}}$$

$$\text{s.t. } \boldsymbol{K}(\boldsymbol{x})\boldsymbol{u} = \boldsymbol{f}, \sum_{i\in\Omega} x_i \le v_0, \boldsymbol{x} \in \{0,1\}^n$$

Minimize compliance:
load bearing structure

Hard physical constraints:
Hooke's law

Materials budgets

Valid grid constraints:
combinatorial constraints

Deep image prior

$$\min_{\boldsymbol{\theta},\boldsymbol{u}} \boldsymbol{u}^\intercal \boldsymbol{K}(\boldsymbol{G_\theta}(\boldsymbol{\beta}))\boldsymbol{u}$$

$$\text{s.t. } \boldsymbol{K}(\boldsymbol{G_\theta}(\boldsymbol{\beta}))\boldsymbol{u} = \boldsymbol{f}, \sum_{i\in\Omega} [\boldsymbol{G_\theta}(\boldsymbol{\beta})]_i \le v_0, \boldsymbol{G_\theta}(\boldsymbol{\beta}) \in \{0,1\}^n$$

Image Credit:
https://www.comsol.com/blogs/finding-a-structures-best-design-with-topology-optimization

Existing Software packages

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: First general-purpose solver for constrained DL problems

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

Key Algorithm[1]

GRANSO

Nonconvex, nonsmooth, constrained

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

Exact penalty function

$$\phi(x;\mu) = \mu f(x) + v(x).$$

Penalty sequential quadratic programming (P-SQP)

$$\min_{d\in\mathbb{R}^n,\, s\in\mathbb{R}^p} \quad \mu\left(f(x_k) + \nabla f(x_k)^{\mathsf{T}} d\right) + e^{\mathsf{T}} s + \frac{1}{2} d^{\mathsf{T}} H_k d$$

$$\text{s.t.} \quad c(x_k) + \nabla c(x_k)^{\mathsf{T}} d \le s, \quad s \ge 0,$$

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.

Key Algorithm[1]

Set $H_0 := I$ and $\mu := \mu_0$
Set $\phi(\cdot)$ as the penalty function given in (2) using $f(\cdot)$ and $c(\cdot)$
Set $\nabla\phi(\cdot)$ and $v(\cdot)$ as the associated gradient (4) and violation function (3)
Evaluate $\phi_0 := \phi(x_0;\mu)$, $\nabla\phi_0 := \nabla\phi(x_0;\mu)$, and $v_0 := v(x_0)$
for $k = 0, 1, 2, \ldots$ **do**
  $[d_k, \hat{\mu}] :=$ **sqp_steering_strategy**(x_k, H_k, μ)
  if $\hat{\mu} < \mu$ **then**
    // Penalty parameter has been lowered by steering; update current iterate
    Set $\mu := \hat{\mu}$
    Reevaluate $\phi_k := \phi(x_k;\mu)$, $\nabla\phi_k := \nabla\phi(x_k;\mu)$, and $v_k := v(x_k)$
  end if
  $\boxed{[x_{k+1}, \phi_{k+1}, \nabla\phi_{k+1}, v_{k+1}] := \textbf{inexact\_linesearch}(x_k, \phi_k, \nabla\phi_k, d_k, \phi(\cdot), \nabla\phi(\cdot))}$
  Compute $d_\diamond$ via (12) and (13)
  if $\|d_\diamond\|_2 < \tau_\diamond$ and $v_{k+1} < \tau_v$ **then**
    // Stationarity and feasibility sufficiently attained; terminate successfully
    break
  end if
  Set H_{k+1} using BFGS update formula
end for

Advantages

- **Reliable step-size rule**
- **Principled stopping criterion**

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.

Limitations of 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

Ref: **Buyun Liang**, Tim Mitchell, Ju Sun. NCVX: A General-Purpose Optimization Solver for Constrained Machine and Deep Learning. In Neural Information Processing Systems (NeurIPS) Workshop on Optimization for Machine Learning (OPT 2022).

NCVX PyGRANSO: Advantages

NCVX

1) Auto-Differentiation

https://ncvx.org/

Orthogonal Dictionary Learning (ODL)

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

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)
```

Demo 2: PyGRANSO for ODL

No Analytical gradients

Ref: **Buyun Liang**, Tim Mitchell, Ju Sun. NCVX: A General-Purpose Optimization Solver for Constrained Machine and Deep Learning. In Neural Information Processing Systems (NeurIPS) Workshop on Optimization for Machine Learning (OPT 2022).

NCVX PyGRANSO: Advantages

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: 17.56377601623535s
```

Ref: **Buyun Liang**, Tim Mitchell, Ju Sun. NCVX: A General-Purpose Optimization Solver for Constrained Machine and Deep Learning. In Neural Information Processing Systems (NeurIPS) Workshop on Optimization for Machine Learning (OPT 2022).

NCVX PyGRANSO: Advantages

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)}
```

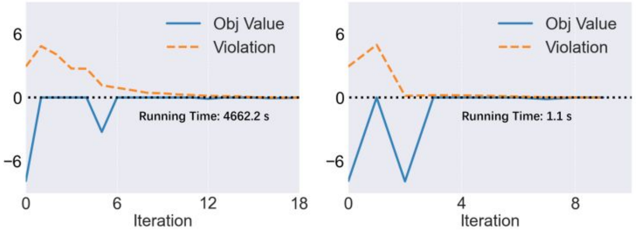
Higher order tensor input

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

Ref: **Buyun Liang**, Tim Mitchell, Ju Sun. NCVX: A General-Purpose Optimization Solver for Constrained Machine and Deep Learning. In Neural Information Processing Systems (NeurIPS) Workshop on Optimization for Machine Learning (OPT 2022).

Practical & Crucial Techniques

1. Constraint-folding



(a) n box constraints **(b)** folded constraints

Reduce # constraints

- Reduce cost of QP in the SQP

Equality into non-negative inequality

$$h_j(\boldsymbol{x}) = 0 \iff |h_j(\boldsymbol{x})| \leq 0$$

Inequality into nonnegative inequality

$$c_i(\boldsymbol{x}) \leq 0 \iff \max\{c_i(\boldsymbol{x}), 0\} \leq 0$$

All non-negative inequalities into one

$$\mathcal{F}(|h_1(\boldsymbol{x})|, \cdots, |h_i(\boldsymbol{x})|, \max\{c_1(\boldsymbol{x}), 0\}, \cdots, \max\{c_j(\boldsymbol{x}), 0\}) \leq 0,$$

$\mathcal{F} : \mathbb{R}_+^{i+j} \mapsto \mathbb{R}_+ \ (\mathbb{R}_+ = \{\alpha : \alpha \geq 0\})$ Can be any function satisfying $\mathcal{F}(\boldsymbol{z}) = 0 \Longrightarrow \boldsymbol{z} = \mathbf{0}$

Ref: Hengyue Liang, **Buyun Liang**, Le Peng, Ying Cui, Tim Mitchell, Ju Sun. *Optimization and Optimizers for Adversarial Robustness*. Under review at International Journal of Computer Vision (IJCV).

Practical & Crucial Techniques

2. Two-stage optimization

Numerical methods may converge to **poor lo** **minima** for NCVX problems

Idea: **different random initializations**

Stage 1 (selecting the best initialization)
R different random initialization;
k iterations

Stage 2 (optimization)
$x^{(*,0)}$ until the stopping criterion is met

Algorithm Selection of $x^{(*,k)}$ and $x^{(*,0)}$ in the two-stage process

Require: Initialization $x^{(r,0)}$ and the corresponding intermediate optimization results $x^{(r,k)}$.

```
if Any x^(r,k) is feasible then
    Set x^(*,k) to be the feasible x^(r,k)'s with the least objective value.
else
    Set x^(*,k) to be the x^(r,k) with the least constraint violation.
end if
Set x^(*,0) corresponds to x^(*,k) found.
return x^(*,k) and x^(*,0).
```

Ref: Hengyue Liang, **Buyun Liang**, Le Peng, Ying Cui, Tim Mitchell, Ju Sun. *Optimization and Optimizers for Adversarial Robustness*. Under review at International Journal of Computer Vision (IJCV).

Practical & Crucial Techniques

3. Numerical Re-scaling

Large amount of constraints: searching direction biased towards staying feasible

Rescaling

$$\min_{\boldsymbol{x}',t} \; t$$
$$\text{s.t.} \; \max_{i \neq y} f^i_{\boldsymbol{\theta}}(\boldsymbol{x}') \geq f^y_{\boldsymbol{\theta}}(\boldsymbol{x}')$$
$$d(\boldsymbol{x}, \boldsymbol{x}') \leq t \,, \; \boxed{\boldsymbol{x}' \in [0,1]^n}$$

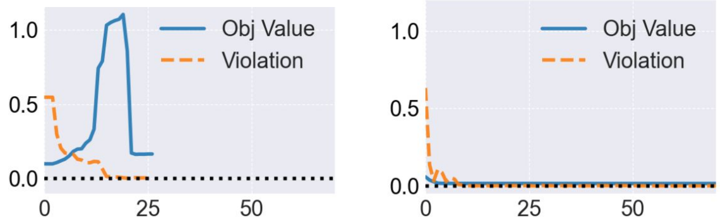
Folding scale: $O(\sqrt{n})$

$$\Rightarrow$$

$$\boxed{\min_{\boldsymbol{x}',t} \; t \cdot \sqrt{n}}$$

Rescaling objective

$$\text{s.t.} \; \max_{i \neq y} f^i_{\boldsymbol{\theta}}(\boldsymbol{x}') \geq f^y_{\boldsymbol{\theta}}(\boldsymbol{x}')$$
$$d(\boldsymbol{x}, \boldsymbol{x}') \leq t \,, \; \boldsymbol{x}' \in [0,1]^n$$



(a) $\min \; t$ **(b)** $\min \; t \cdot \sqrt{n}$

Ref: Hengyue Liang, **Buyun Liang**, Le Peng, Ying Cui, Tim Mitchell, Ju Sun. *Optimization and Optimizers for Adversarial Robustness*. Under review at International Journal of Computer Vision (IJCV).

Practical & Crucial Techniques

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

$$\min_{\boldsymbol{x}'} \; d\left(\boldsymbol{x}, \boldsymbol{x}'\right)$$
$$\text{s.t. } \max_{i \neq y} f_{\boldsymbol{\theta}}^{i}(\boldsymbol{x}') \geq f_{\boldsymbol{\theta}}^{y}(\boldsymbol{x}')\,, \; \boldsymbol{x}' \in [0,1]^n$$

4. Reformulation to accelerate convergence

Metric d	ℓ_1	ℓ_2	ℓ_∞	Others
max	$\ell\left(\boldsymbol{y}, f_\theta\left(\boldsymbol{x}'\right)\right)$	$\ell\left(\boldsymbol{y}, f_\theta\left(\boldsymbol{x}'\right)\right)$	$\ell\left(\boldsymbol{y}, f_\theta\left(\boldsymbol{x}'\right)\right)$	$\ell\left(\boldsymbol{y}, f_\theta\left(\boldsymbol{x}'\right)\right)$
s.t.	$\left\Vert\boldsymbol{x}' - \boldsymbol{x}\right\Vert_1 \leq \varepsilon$	$\left\Vert\boldsymbol{x}' - \boldsymbol{x}\right\Vert_2 \leq \varepsilon$	$\Vert\mathbf{max}\{\boldsymbol{x}' - \boldsymbol{x} - \varepsilon\mathbf{1}, 0\}\Vert_2 \leq 0$	$d\left(\boldsymbol{x}', \boldsymbol{x}\right) \leq \varepsilon$
	$\Vert\mathbf{max}\{\mathbf{concat}\left(-\boldsymbol{x}',\ \boldsymbol{x}' - \mathbf{1}\right),\ \mathbf{0}\}\Vert_2 \leq 0$			
min	$\mathbf{1}^\intercal \boldsymbol{t}$	$\left\Vert\boldsymbol{x}' - \boldsymbol{x}\right\Vert_2$	t	$d\left(\boldsymbol{x}', \boldsymbol{x}\right)$
s.t.	$\Vert\mathbf{max}\{\mathbf{concat}(\boldsymbol{x}' - \boldsymbol{x} - \boldsymbol{t}, -\boldsymbol{x}' + \boldsymbol{x} - \boldsymbol{t}),\ \mathbf{0}\}\Vert_2 \leq 0$	Not Applicable	$\Vert\mathbf{max}\{\mathbf{concat}(\boldsymbol{x}' - \boldsymbol{x} - t\mathbf{1}, -\boldsymbol{x}' + \boldsymbol{x} - t\mathbf{1}),\ \mathbf{0}\}\Vert_2 \leq 0$	Not Applicable
	$\max_{i \neq y} f_{\boldsymbol{\theta}}^{i}(\boldsymbol{x}') \geq f_{\boldsymbol{\theta}}^{y}(\boldsymbol{x}')$			
	$\Vert\mathbf{max}\{\mathbf{concat}\left(-\boldsymbol{x}',\ \boldsymbol{x}' - \mathbf{1}\right),\ \mathbf{0}\}\Vert_2 \leq 0$			

Ref: Hengyue Liang, **Buyun Liang**, Le Peng, Ying Cui, Tim Mitchell, Ju Sun. *Optimization and Optimizers for Adversarial Robustness*. Under review at International Journal of Computer Vision (IJCV).

Example: Adversarial Robustness

Maximize loss function

$$\boxed{\max_{\boldsymbol{x}'} \ell\left(\boldsymbol{y}, f_{\boldsymbol{\theta}}(\boldsymbol{x}')\right)}$$

$$\text{s. t.} \ \boxed{d\left(\boldsymbol{x}, \boldsymbol{x}'\right) \leq \varepsilon}, \quad \boxed{\boldsymbol{x}' \in [0,1]^n}$$

Allowable perturbation with radius ε

Valid image constraints

Minimize robustness radius

$$\boxed{\min_{\boldsymbol{x}'} \ d\left(\boldsymbol{x}, \boldsymbol{x}'\right)}$$

$$\text{s. t.} \ \boxed{\max_{i \neq y} f^i_{\boldsymbol{\theta}}(\boldsymbol{x}') \geq f^y_{\boldsymbol{\theta}}(\boldsymbol{x}')}, \ \boxed{\boldsymbol{x}' \in [0,1]^n}$$

Change the predicted class

Valid image constraints

```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)
```

Example: Adversarial Robustness

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

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

Standard Lp norm

			APGD			PWCF(ours)			Square
Dataset	**Metric (ε)**	**Clean**	**CE**	**M**	**CE+M**	**CE**	**M**	**CE+M**	**M**
CIFAR-10	$\ell_1(12)$	73.29	0.97	<u>0.00</u>	0.00	17.93	0.01	0.01	2.28
	$\ell_2(0.5)$	94.61	81.81	81.06	80.92	81.99	<u>81.02</u>	80.87	87.9
	$\ell_\infty(0.03)$	90.81	69.44	<u>67.71</u>	67.33	88.71	68.20	68.17	71.6
ImageNet-100	$\ell_2(4.7)$	75.04	<u>42.44</u>	44.06	40.86	42.50	43.52	40.60	63.1
	$\ell_\infty(0.016)$	75.04	<u>46.78</u>	47.54	45.20	73.92	47.72	47.72	59.9

NCVX performs strongly and comparable to SOTA as a general solver

Ref: Hengyue Liang, **Buyun Liang**, Le Peng, Ying Cui, Tim Mitchell, Ju Sun. *Optimization and Optimizers for Adversarial Robustness*. Under review at International Journal of Computer Vision (IJCV).

Example: Adversarial Robustness

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

Perceptual distance

$$d(\boldsymbol{x}, \boldsymbol{x}') \doteq \|\phi(\boldsymbol{x}) - \phi(\boldsymbol{x}')\|_2$$
$$\text{where} \quad \phi(\boldsymbol{x}) \doteq [\ \widehat{g}_1(\boldsymbol{x}), \ldots, \widehat{g}_L(\boldsymbol{x})\ ]$$

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

	cross-entropy loss		**margin loss**	
Method	**Viol. (%) ↓**	**Att. Succ. (%) ↑**	**Viol. (%) ↓**	**Att. Succ. (%) ↑**
Fast-LPA	73.8	3.54	41.6	56.8
LPA	**0.00**	80.5	**0.00**	97.0
PPGD	5.44	25.5	**0.00**	38.5
PWCF (ours)	0.62	93.6	**0.00**	100

NCVX can handle general-form distances

Ref: Hengyue Liang, **Buyun Liang**, Le Peng, Ying Cui, Tim Mitchell, Ju Sun. *Optimization and Optimizers for Adversarial Robustness*. Under review at International Journal of Computer Vision (IJCV).

Summary

A solver for constrained deep learning problems

- Auto-differentiation
- GPU support
- Tensor variable support

Practical techniques to speed up

- Constraint folding (into a single one)
- Two stage-optimization
- Objective and constraint rescaling
- Reformulation

Next steps

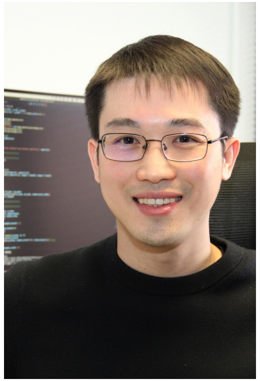
- Autoscaling
- Stochastic Optimization

Thanks to

Hengyue Liang

Ryan de Vera

Prof. Ying Cui



Prof. Qizhi He

Prof. Tim Mitchell

Prof. Ju Sun

References

- **Buyun Liang**, Wenjie Zhang, Hengyue Liang, Tim Mitchell, Ying Cui, Ju Sun. *NCVX: A General-Purpose Optimization Solver for Machine Learning, and Practical Techniques*. In preparation for IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI).
- **Buyun Liang**, Tim Mitchell, Ju Sun. *NCVX: A General-Purpose Optimization Solver for Constrained Machine and Deep Learning.* In Neural Information Processing Systems (NeurIPS) Workshop on Optimization for Machine Learning (OPT 2022).
- Hengyue Liang, **Buyun Liang**, Le Peng, Ying Cui, Tim Mitchell, Ju Sun. *Optimization and Optimizers for Adversarial Robustness*. Under review at International Journal of Computer Vision (IJCV).
- Hengyue Liang, **Buyun Liang**, Le Peng, Ying Cui, Tim Mitchell, Ju Sun. *Implications of Solution Patterns on Adversarial Robustness*. In Computer Vision and Pattern Recognition (CVPR) Workshop of Adversarial Machine Learning on Computer Vision (Art of Robustness).
- Hengyue Liang, **Buyun Liang**, Ying Cui, Tim Mitchell, Ju Sun. *Optimization for Robustness Evaluation beyond ℓ_p Metrics*. In IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2023) & Neural Information Processing Systems (NeurIPS) Workshop on Optimization for Machine Learning (OPT 2022).

Thank you!

Key Algorithm[1]

Nonconvex, nonsmooth, constrained

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

Exact penalty function

$$\phi(x;\mu) = \mu f(x) + v(x).$$

Penalty sequential quadratic programming (P-SQP)

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

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.

Key Algorithm[1]

Corresponding dual

$$\max_{\lambda \in \mathbb{R}^p} \quad \mu f(x_k) + c(x_k)^\mathsf{T}\lambda - \frac{1}{2}(\mu \nabla f(x_k) + \nabla c(x_k)\lambda)^\mathsf{T} H_k^{-1}(\mu \nabla f(x_k) + \nabla c(x_k)\lambda)$$
$$\text{s.t.} \quad 0 \leq \lambda \leq e, \tag{8}$$

Primal solution (recovered from dual solution): searching direction

$$d_k = -H_k^{-1}(\mu \nabla f(x_k) + \nabla c(x_k)\lambda_k). \tag{9}$$

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.

Key Algorithm[1]

GRANSO

Linear model of constraint violation

$$l(d; x_k) := \| \max\{c(x_k) + \nabla c(x_k)^{\mathsf{T}} d, 0\}\|_1$$

Corresponding reduction

$$\begin{aligned} l_\delta(d; x_k) &:= l(0; x_k) - l(d; x_k) \\ &= v(x_k) - \| \max\{c(x_k) + \nabla c(x_k)^{\mathsf{T}} d, 0\}\|_1 \end{aligned}$$

Procedure 1 $[d_k, \mu_{\text{new}}]$ = **sqp_steering_strategy**(x_k, H_k, μ)

Input:

Current iterate x_k and BFGS Hessian approximation H_k
Current value of the penalty parameter μ

Constants:

Values $c_v \in (0, 1)$ and $c_\mu \in (0, 1)$

Output:

Search direction d_k
Penalty parameter $\mu_{\text{new}} \in (0, \mu]$

Solve QP (8) using $\mu_{\text{new}} := \mu$ to obtain search direction d_k from (9)
if $l_\delta(d_k; x_k) < c_v v(x_k)$ **then**
  Solve (8) using $\mu = 0$ to obtain reference direction $\tilde{d}_k$ from (9)
  while $l_\delta(d_k; x_k) < c_v l_\delta(\tilde{d}_k; x_k)$ **do**
    $\mu_{\text{new}} := c_\mu \mu_{\text{new}}$
    Solve QP (8) using $\mu := \mu_{\text{new}}$ to obtain search direction d_k from (9)
  end while
end if

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.

Key Algorithm[1]

Gradient from l most recent iterates

$$G := [\nabla f(x_{k+1-l}) \cdots \nabla f(x_k)]$$

$$J_i := [\nabla c_i(x_{k+1-l}) \cdots \nabla c_i(x_k)], \quad i \in \{1, \ldots, p\}$$

Augmented QP

$$\max_{\sigma \in \mathbb{R}^l, \lambda \in \mathbb{R}^{pl}} \quad \sum_{i=1}^{p} c_i(x_k) e^\mathsf{T} \lambda_i - \frac{1}{2} \begin{bmatrix} \sigma \\ \lambda \end{bmatrix}^\mathsf{T} [G, J_1, \ldots, J_p]^\mathsf{T} H_k^{-1} [G, J_1, \ldots, J_p] \begin{bmatrix} \sigma \\ \lambda \end{bmatrix}$$

$$\text{s.t.} \quad 0 \leq \lambda_i \leq e, \quad e^\mathsf{T}\sigma = \mu, \ \sigma \geq 0. \tag{12}$$

Primal solution: termination condition

$$d_\diamond = H_k^{-1} [G, J_1, \ldots, J_p] \begin{bmatrix} \sigma \\ \lambda \end{bmatrix}$$

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.

Key Algorithm[1]

GRANSO

Augmented QP

$$\max_{\sigma \in \mathbb{R}^l, \lambda \in \mathbb{R}^{pl}} \quad \sum_{i=1}^{p} c_i(x_k) e^{\mathsf{T}} \lambda_i - \frac{1}{2} \begin{bmatrix} \sigma \\ \lambda \end{bmatrix}^{\mathsf{T}} [G, J_1, \ldots, J_p]^{\mathsf{T}} H_k^{-1} [G, J_1, \ldots, J_p] \begin{bmatrix} \sigma \\ \lambda \end{bmatrix}$$

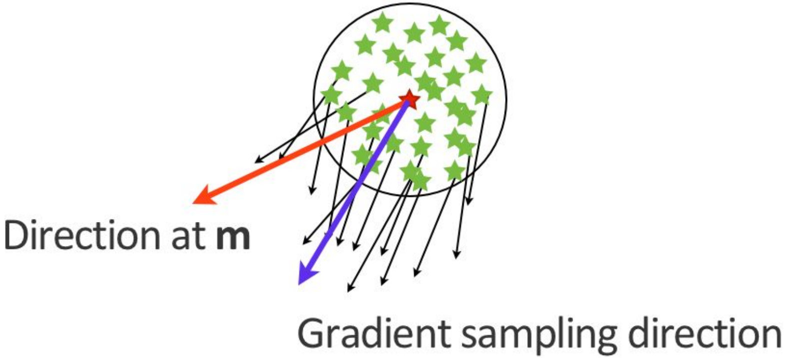
$$\text{s.t.} \quad 0 \le \lambda_i \le e, \quad e^{\mathsf{T}} \sigma = \mu, \quad \sigma \ge 0. \tag{12}$$

Stationarity based on (approximate) gradient sampling

$$G_k := \begin{bmatrix} \nabla f(x^k) & \nabla f(x^{k,1}) & \cdots & \nabla f(x^{k,m}) \end{bmatrix}$$

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

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



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.

Key Algorithm[1]

Set $H_0 := I$ and $\mu := \mu_0$
Set $\phi(\cdot)$ as the penalty function given in (2) using $f(\cdot)$ and $c(\cdot)$
Set $\nabla\phi(\cdot)$ and $v(\cdot)$ as the associated gradient (4) and violation function (3)
Evaluate $\phi_0 := \phi(x_0;\mu)$, $\nabla\phi_0 := \nabla\phi(x_0;\mu)$, and $v_0 := v(x_0)$
for $k = 0, 1, 2, \ldots$ **do**
  $[d_k, \hat{\mu}] :=$ **sqp_steering_strategy**(x_k, H_k, μ)
  if $\hat{\mu} < \mu$ **then**
    // Penalty parameter has been lowered by steering; update current iterate
    Set $\mu := \hat{\mu}$
    Reevaluate $\phi_k := \phi(x_k;\mu)$, $\nabla\phi_k := \nabla\phi(x_k;\mu)$, and $v_k := v(x_k)$
  end if
  $[x_{k+1}, \phi_{k+1}, \nabla\phi_{k+1}, v_{k+1}] :=$ **inexact_linesearch**$(x_k, \phi_k, \nabla\phi_k, d_k, \phi(\cdot), \nabla\phi(\cdot))$
  Compute $d_\diamond$ via (12) and (13)
  if $\|d_\diamond\|_2 < \tau_\diamond$ and $v_{k+1} < \tau_v$ **then**
    // Stationarity and feasibility sufficiently attained; terminate successfully
    break
  end if
  Set H_{k+1} using BFGS update formula
end for

Advantages

- **Reliable step-size rule**
- **Principled stopping criterion**

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.

Limitations of 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