# Deep Learning with Nontrivial Constraints

**Ju Sun**
McKnight Land-Grant Professor
Computer Science & Engineering

Dec 10, 2025

**Industrial and Systems Engineering Graduate Seminar**

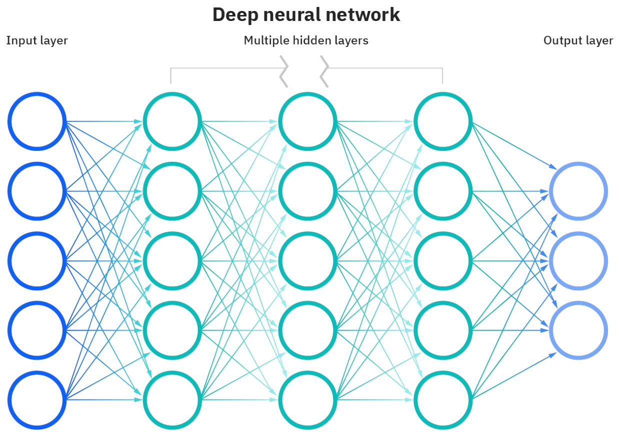
University of Minnesota
Driven to Discover℠

# Research in the group

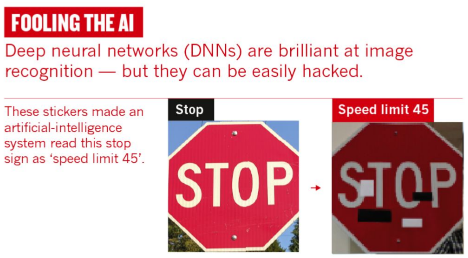
GLOVEX

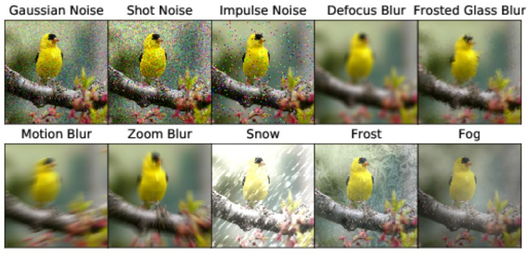
https://glovex.umn.edu/

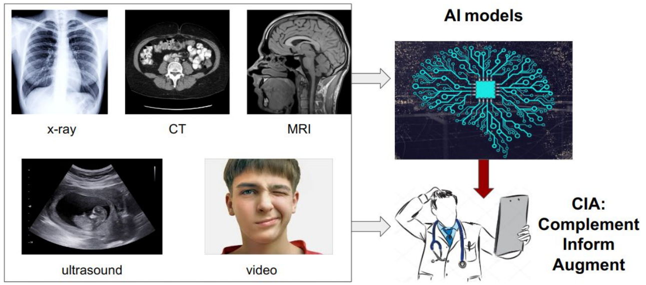
(Machine) **Learning**, (Numerical) **Optimization**, (Computer) **Vision**, healthcar**E**, + **X**

# Our research themes

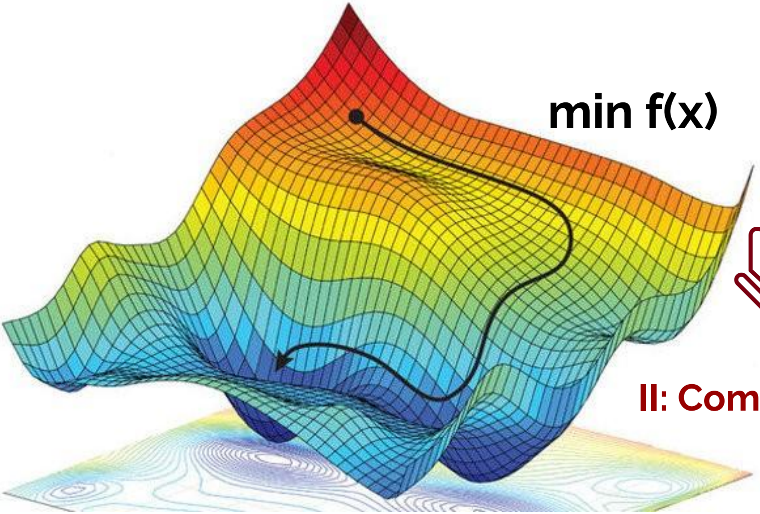



## I: Trustworthy AI

## III: AI for Healthcare

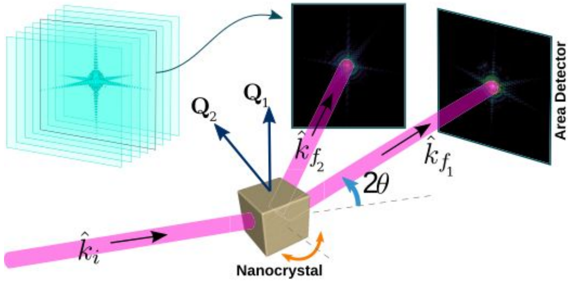


$$\min f(x) \quad \text{s.t.} \quad g(x) <= 0$$

NCVX

https://ncvx.org/

## II: Computation for AI




## IV: AI for Science and Engineering

# Robustness & Deployability of Deep Learning Models

## First general-purpose, reliable solver for robust evaluation

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

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

### Reliability

- SOTA methods RobustBench A standardized benchmark for adversarial robustness
  No stopping criterion (only use maxit); step size scheduler
- PWCF (ours)
  Principled line-search criterion and termination condition

### Generality

- SOTA methods
  Can mostly only handle several lp metrics (l1,l2,linf)
- PWCF (ours)
  Any differentiable metrics and both min and max forms
  E.g., perceptual distance

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

## Selective predictions for safety and deployability

predictor $f : \mathcal{X} \to \mathbb{R}^K$

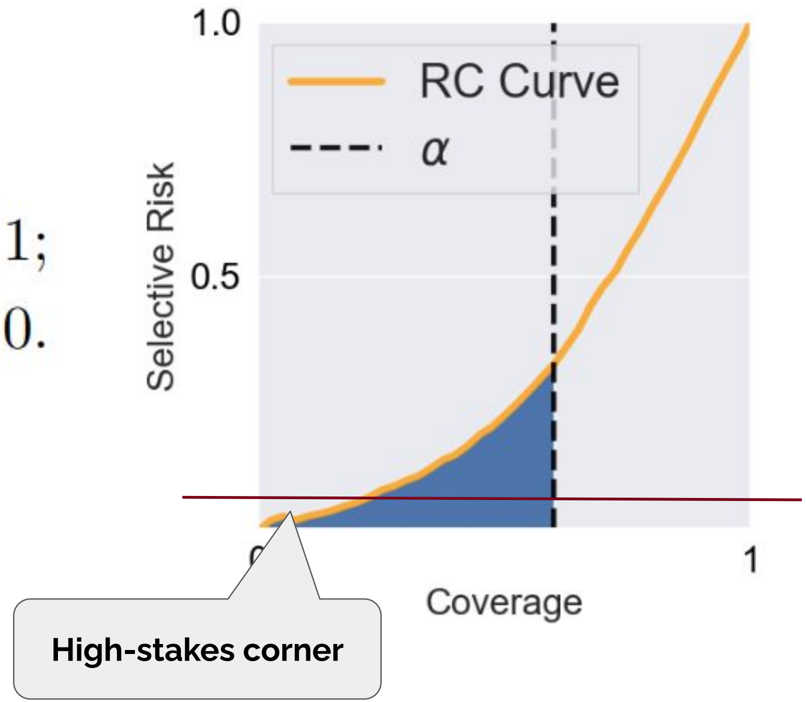
selector $g : \mathcal{X} \to \{0, 1\}$

$$(f, g)(\boldsymbol{x}) \triangleq \begin{cases} f(\boldsymbol{x}) & \text{if } g(\boldsymbol{x}) = 1; \\ \boxed{\text{abstain} \quad \text{if } g(\boldsymbol{x}) =} 0. \end{cases}$$

No prediction on uncertain samples and defer them to humans

$$g_\gamma(\boldsymbol{x}) = \mathbb{1}[s(\boldsymbol{x}) > \gamma]$$

Typically, selection by thresholding prediction confidence



# Computation for Deep Learning with Nontrivial Constraints

PyGRANSO

NCVX PyGRANSO Documentation

Search the docs ...

Introduction

Installation

Sett

Exa

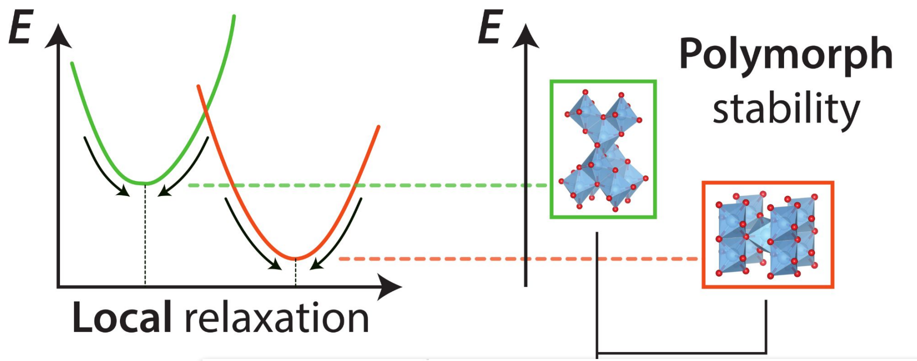
## Home

https://ncvx.org/

NCVX

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

- **First general-purpose solver for <u>hard-constrained</u> deep learning problems**
- **Recently updated to be compatible with PyTorch 2.8**




Stabilit
respe
phase se

Settings

Examples

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

Buyun Liang, Tim Mitchell, Ju Sun

## Enabling

### Robustness evaluation

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

$$\text{s. t. } \boldsymbol{x}' \in \Delta(\boldsymbol{x}) = \{\boldsymbol{x}' \in [0,1]^n : d(\boldsymbol{x}, \boldsymbol{x}') \leq \varepsilon\}$$

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

- **(W-CSTR-T) Weak explicit constraints informed by phase transition**: For each composition, the energy/atom of any non-ground-state polymorph is greater than that of the ground-state polymorph
- **(W-CSTR-S) Weak explicit constraints informed by phase separation**: For each chemical space, the energy/atom of any TUS material is above the lower convex envelope in the composition-energy space

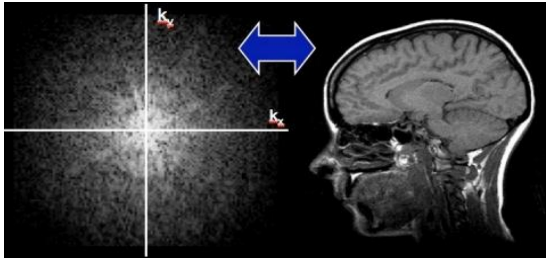
$$\min_{\boldsymbol{\theta}} \frac{1}{N} \sum\nolimits_{i=1}^{N} \ell(\boldsymbol{y}_i, f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)) + \Omega(\boldsymbol{\theta}) \quad \text{s. t.} \quad f_{\boldsymbol{\theta}}(\boldsymbol{x}_p) \geq f_{\boldsymbol{\theta}}(\boldsymbol{x}_q) \; \forall \, (p, q) \in \mathcal{O}_{\text{W-CSTR-T}},$$

$$\text{s. t. } \frac{\sum_{i=1}^{N} \mathbb{1}\{y_i = +1\} \mathbb{1}\{f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) > t\}}{\sum_{i=1}^{N} \mathbb{1}\{y_i = +1\}} \geq \alpha$$

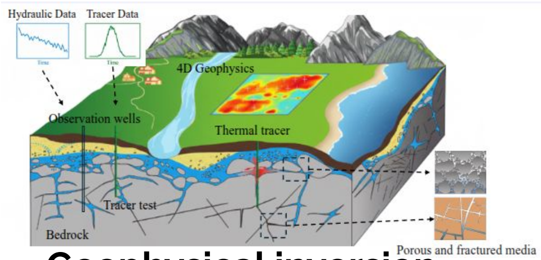
Yash, Le, Zhong, Buyun

# Generative Models for Scientific/Engineering Inverse Modeling

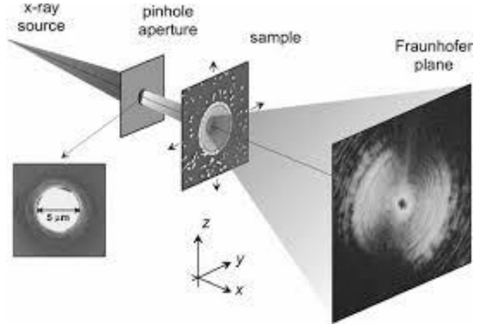
(inverse problems, reconstruction, data assimilation, inverse design/control, conditional generation, estimation …)




**MRI reconstruction**




**Geophysical inversion**




**Coherent diffraction imaging**


**Inverse shape control**

Given $\boldsymbol{y} \approx f(\boldsymbol{x})$, recover $\boldsymbol{x}$

$$\boxed{\min_{\boldsymbol{x}} \; \ell(\boldsymbol{y}, f(\boldsymbol{x})) + \Omega(\boldsymbol{x})}$$

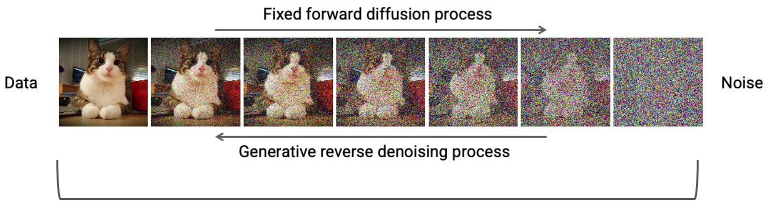
**Challenges:**

- Linear vs. **nonlinear** f
- Unconstrained vs. **constrained** (e.g., PDE (x, y) = 0 )
- Explicit vs. **implicit** f

# A **plug-in principle** for leveraging **pretrained** deep generative models to solve IM

object-only datasets $\{\boldsymbol{x}_i\}_{i=1,\ldots,N}$

**Distribution learning** via deep generative models




$\mathcal{G}_{\boldsymbol{\theta}}$

**Plug in pretrained** deep generative priors

$$\boxed{\min_{\boldsymbol{z}} \; \mathcal{L}(\boldsymbol{z}) \doteq \ell(\boldsymbol{y}, \mathcal{A} \circ \mathcal{G}_{\boldsymbol{\theta}}(\boldsymbol{z})) + \Omega \circ \mathcal{G}_{\boldsymbol{\theta}}(\boldsymbol{z})}$$

**DMPlug: A Plug-in Method for Solving Inverse Problems with Diffusion Models**

Hengkang Wang, Xu Zhang, Taihui Li, Yuxiang Wan, Tiancong Chen, Ju Sun

NeurIPS'24, with **domain-specific** priors

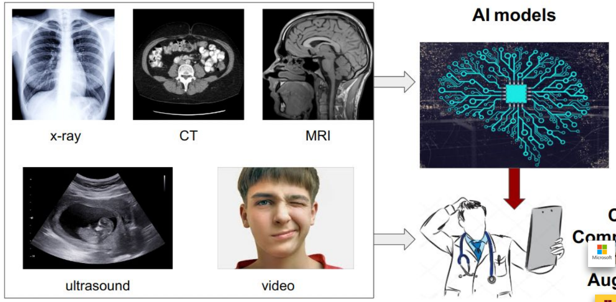
**Saving Foundation Flow-Matching Priors for Inverse Problems**

Yuxiang Wan, Ryan Devera, Wenjie Zhang, Ju Sun

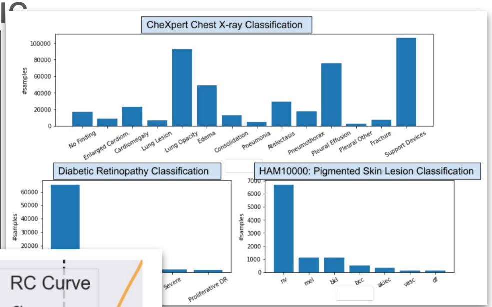
Submitted to ICLR'25, with **foundation** priors (e.g., stable diffusion models)

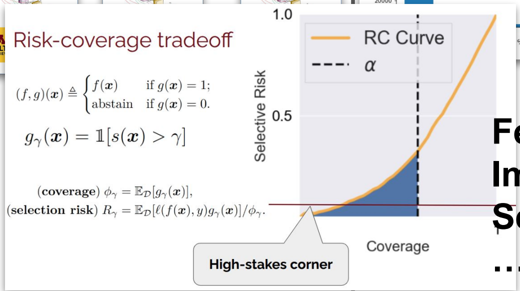
Challenge: Foundation generative models are powerful because **they're not specific**

# Applied and basic research in AI for Healthcare



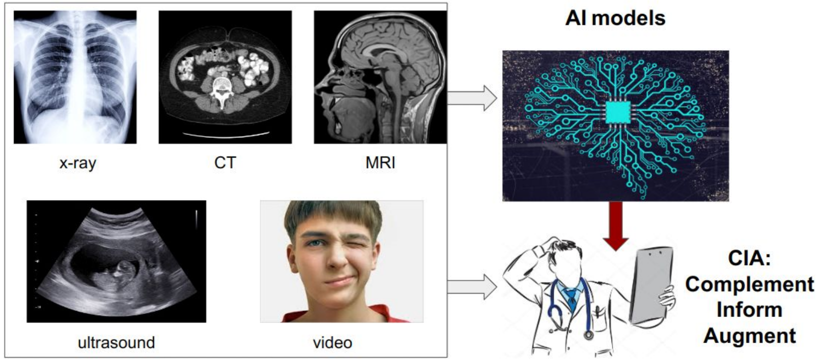

AI-based diagnostic & prognostic models

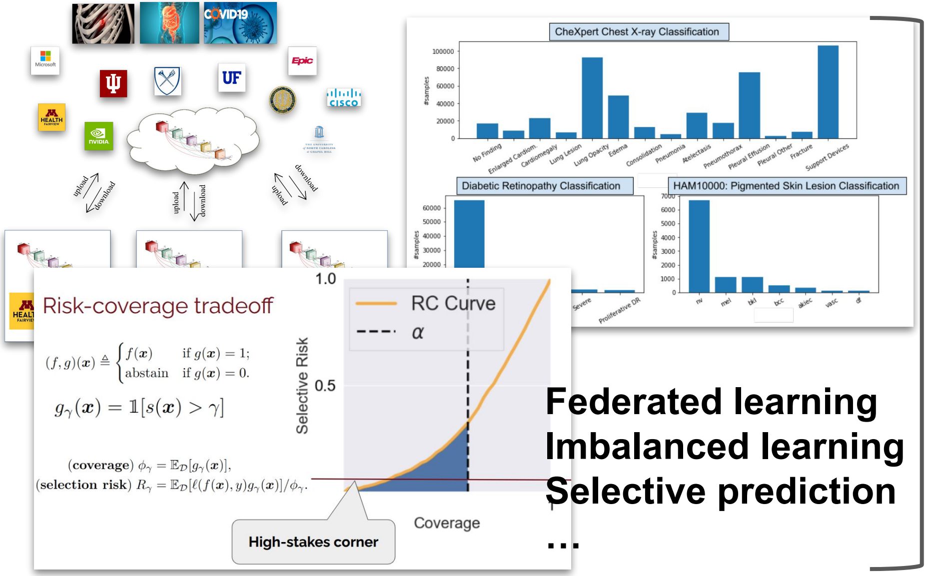




**Federated learning**
**Imbalanced learning**
**Selective prediction**
**…**

Basic research to address data sparsity, data imbalance, and AI safety in healthcare (& beyond)




AI-based diagnostic models



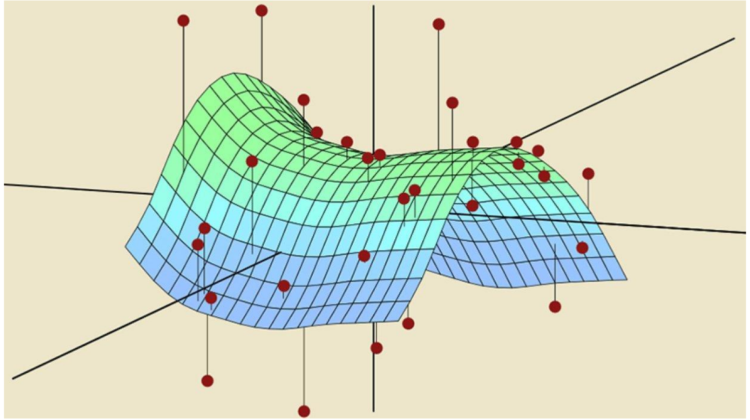

Basic research to address data sparsity, data imbalance, and AI safety in healthcare (& beyond)

**Supported by 3+ (4th forthcoming) active NIH R01 grants**

# Thanks to our funders

# Practical Optimization for Deep Learning with Nontrivial Constraints

# Three fundamental questions in DL



- **Approximation**: is it powerful, i.e., the $\mathcal{H}$ large enough for all possible weights?
- **Optimization**: how to solve
$$\min_{\boldsymbol{w}_i's, \boldsymbol{b}_i's} \frac{1}{n}\sum_{i=1}^{n} \ell\left[\boldsymbol{y}_i, \{\mathsf{NN}\left(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_k, b_1, \ldots, b_k\right)\}\left(\boldsymbol{x}_i\right)\right]$$
- **Generalization**: does the learned NN work well on "similar" data?

# Isn't it solved?

## Base class

CLASS torch.optim.Optimizer(*params*, *defaults*) [SO

Base class for all optimizers.

• WARNING

Parameters need to be specified as collections
consistent between runs. Examples of objects t
and iterators over values of dictionaries.

Parameters:

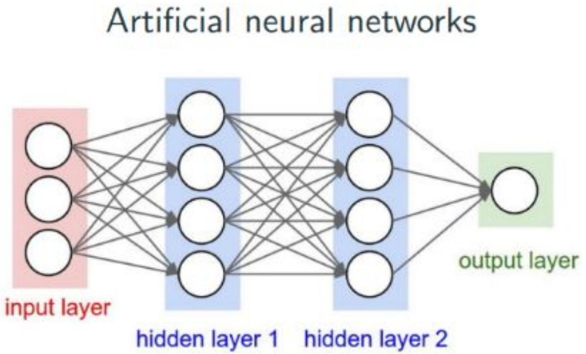
- **params** (*iterable*) – an iterable of to
  Tensors should be optimized.
- **defaults** – (dict): a dict containing d
  when a parameter group doesn't spe

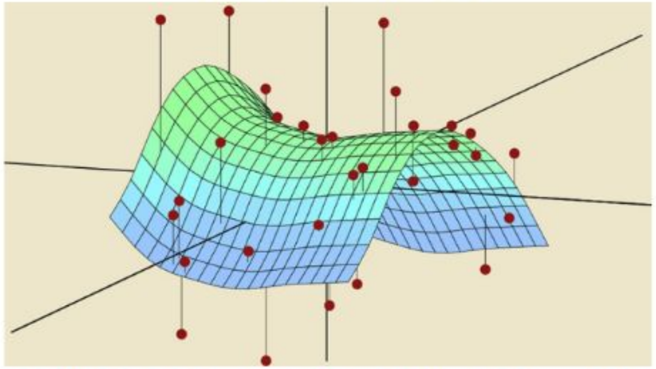
## Algorithms

| | |
|---|---|
| Adadelta | Implements Adadelta algorithm. |
| Adagrad | Implements Adagrad algorithm. |
| Adamax | Implements Adamax algorithm (a variant of Adam based on infinity norm). |
| ASGD | Implements Averaged Stochastic Gradient Descent. |
| LBFGS | Implements L-BFGS algorithm, heavily inspired by minFunc. |
| NAdam | Implements NAdam algorithm. |
| RAdam | Implements RAdam algorithm. |

lgorithm

# When DL meets constraints



used to approximate nonlinear functions

**Unconstrained optimization**

$$\min_{\boldsymbol{w}_i's, \boldsymbol{b}_i's} \frac{1}{n} \sum_{i=1}^{n} \ell \left[\boldsymbol{y}_i, \left\{\mathsf{NN}\left(\boldsymbol{w}_1, \dots, \boldsymbol{w}_k, b_1, \dots, b_k\right)\right\}\left(\boldsymbol{x}_i\right)\right]$$

$$\min_{\boldsymbol{x}} \ f(\boldsymbol{x})$$

**“Solved”**

**Constrained optimization**

$$\min_{\boldsymbol{x}} \ f(\boldsymbol{x}) \quad \text{s. t.} \ \ g(\boldsymbol{x}) \le \mathbf{0}$$

**largely “unsolved”**

# This talk is about GAPS

**Constrained optimization**

$$\min_{\boldsymbol{x}} \ f(\boldsymbol{x}) \quad \text{s.t.} \ \ g(\boldsymbol{x}) \le \mathbf{0}$$

**largely "unsolved"**


An imaginary chat between a PhD student working in deep learning (**DLP**) and a PhD student working in optimization (**OP**)

DLP: Man, I've solved a constrained DL problem recently
OP: Oh, that's a hard problem
DLP: Really? I actually did it
OP: How?
DLP: My problem is $\min_x \ f(x)$, s.t. $g(x) \le 0$. I put $g(x)$ as a penalty and then call ADAM
OP: Are you sure it works?
DLP: Yes, the performance is improved and my paper is published at ICML
OP: Why don't you try augmented Lagrangian methods?
DLP: No implementation in Pytorch. Is it possible we work out some theory about my method?
OP: I think it's hard. It's not convex

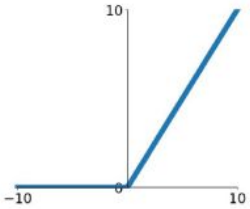
# Outline

**Constrained deep learning: CDL**

- **What, how, and why for CDL**
- No good solvers for CDL yet
- Granso and PyGranso
- PyGranso in action
- Outlook

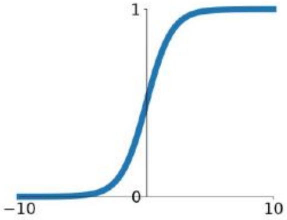
# DL with simple constraints

## Embedding constraints into DL models




**ReLU**
(Rectified Linear Unit)

**Nonnegativity**




**Sigmoid**

**[0, 1]**

$$z \mapsto \left[ \frac{e^{z_1}}{\sum_j e^{z_j}}, \ldots, \frac{e^{z_p}}{\sum_j e^{z_j}} \right]^{\mathsf{T}}$$

**Softmax**

**Nonnegativity and summed to 1**

# DL with nontrivial constraints

- Robustness evaluation
- Imbalanced learning
- Topology optimization
- Contrastive learning
- Neuro-symbolic AI

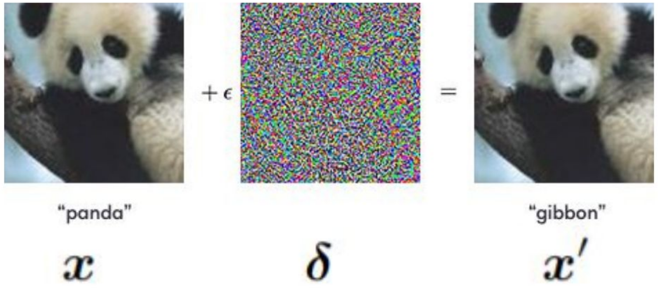
**Deep Learning with Nontrivial Constraints: Methods and Applications**

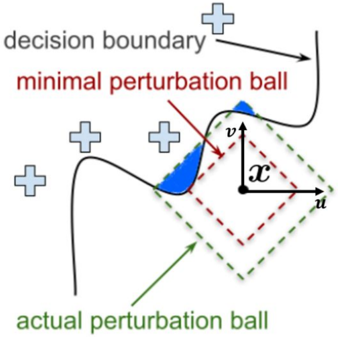
**Chuan He**[1], **Ryan Devera**[1], **Wenjie Zhang**[1], **Ying Cui**[2], **Zhaosong Lu**[3] and **Ju Sun**[1]

[1]Computer Science and Engineering, University of Minnesota
[2]Industrial Engineering and Operations Research, University of California, Berkeley
[3]Industrial and Systems Engineering, University of Minnesota

{he000233, dever120, zhan7867}@umn.edu, yingcui@berkeley.edu, {zhaosong, jusun}@umn.edu

# Robustness evaluation (RE)




Maximize loss function

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

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

Allowable perturbation      Valid image

Minimize robustness radius

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

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

Change the predicted class      Valid image

**Ref** Optimization and Optimizers for Adversarial Robustness. Liang, H., Liang, B., Peng, L., Cui, Y., Mitchell, T., & Sun, J. https://arxiv.org/abs/2303.13401

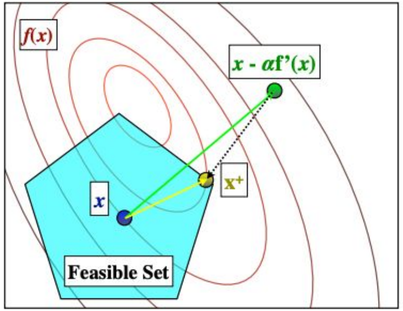
# Projected gradient descent (PGD) for RE

$$\min_{\mathbf{x}\in\mathcal{Q}} f(\mathbf{x})$$

**Step size**

$$\mathbf{x}_{k+1} = P_{\mathcal{Q}}\Big(\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)\Big)$$

$$P_{\mathcal{Q}}(\mathbf{x}_0) = \arg\min_{\mathbf{x}\in\mathcal{Q}} \frac{1}{2}\|\mathbf{x}-\mathbf{x}_0\|_2^2$$ **Projection operator**




**Key hyperparameters:**
(1) step size
(2) iteration number

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

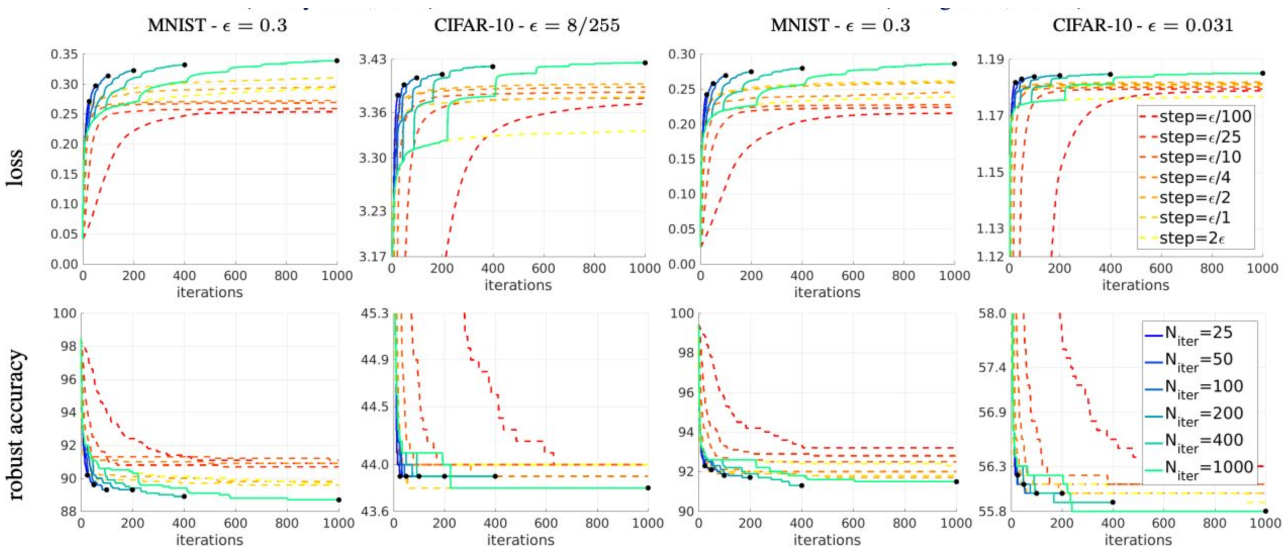
$$\text{s.t. } d\left(\boldsymbol{x}, \boldsymbol{x}'\right) \le \varepsilon\,, \quad \boldsymbol{x}' \in [0,1]^n$$

**Algorithm 1** APGD

**Input:** $f$, $S$, $x^{(0)}$, $\eta$, $N_{\text{iter}}$, $W = \{w_0, \ldots, w_n\}$
**Output:** $x_{\max}$, $f_{\max}$
$x^{(1)} \leftarrow P_S\left(x^{(0)} + \eta\nabla f(x^{(0)})\right)$
$f_{\max} \leftarrow \max\{f(x^{(0)}), f(x^{(1)})\}$
$x_{\max} \leftarrow x^{(0)}$ **if** $f_{\max} \equiv f(x^{(0)})$ **else** $x_{\max} \leftarrow x^{(1)}$
**for** $k = 1$ **to** $N_{\text{iter}}-1$ **do**
  $z^{(k+1)} \leftarrow P_S\left(x^{(k)} + \eta\nabla f(x^{(k)})\right)$
  $x^{(k+1)} \leftarrow P_S\Big(x^{(k)} + \alpha(z^{(k+1)} - x^{(k)})$
    $+(1-\alpha)(x^{(k)} - x^{(k-1)})\Big)$
  **if** $f(x^{(k+1)}) > f_{\max}$ **then**
    $x_{\max} \leftarrow x^{(k+1)}$ and $f_{\max} \leftarrow f(x^{(k+1)})$
  **end if**
  **if** $k \in W$ **then**
    **if** Condition 1 **or** Condition 2 **then**
      $\eta \leftarrow \eta/2$ and $x^{(k+1)} \leftarrow x_{\max}$
    **end if**
  **end if**
**end for**


**Ref** https://angms.science/doc/CVX/CVX_PGD.pdf
https://www.cs.ubc.ca/~schmidtm/Courses/5XX-S20/S5.pdf
Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. Croce, F., Hein, M., ICML 2020
https://arxiv.org/pdf/2003.01690.pdf

# Problem with PGD




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

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

**Tricky to set:**
**iteration number & step size**
**i.e., tricky to decide where to stop**

**Ref** Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. Croce, F., Hein, M., ICML 2020
https://arxiv.org/pdf/2003.01690.pdf

# RE: penalty methods for complicated d

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

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

$$d(\boldsymbol{x}, \boldsymbol{x}') \doteq \|\phi(\boldsymbol{x}) - \phi(\boldsymbol{x}')\|_2$$

where $\phi(\boldsymbol{x}) \doteq [\ \widehat{g}_1(\boldsymbol{x}), \ldots, \widehat{g}_L(\boldsymbol{x})\ ]$

**perceptual distance**

**Projection onto the constraint is complicated**

**Penalty methods**

$$\max_{\widetilde{\mathbf{x}}} \quad \mathcal{L}(f(\widetilde{\mathbf{x}}), y) - \lambda \max\left(0, \|\phi(\widetilde{\mathbf{x}}) - \phi(\mathbf{x})\|_2 - \epsilon\right)$$

Solve it for each fixed $\lambda$ and then increase $\lambda$

**Algorithm 2** Lagrangian Perceptual Attack (LPA)

**procedure** LPA(classifier network $f(\cdot)$, LPIPS distance $d(\cdot,\cdot)$, input $\mathbf{x}$, label $y$, bound $\epsilon$)
 $\lambda \leftarrow 0.01$
 $\widetilde{\mathbf{x}} \leftarrow \mathbf{x} + 0.01 * \mathcal{N}(0,1)$ ▷ initialize perturbations with random Gaussian noise
 **for** $i$ in $1, \ldots, S$ **do** ▷ we use $S = 5$ iterations to search for the best value of $\lambda$
  **for** $t$ in $1, \ldots, T$ **do** ▷ $T$ is the number of steps
   $\Delta \leftarrow \nabla_{\widetilde{\mathbf{x}}} \left[\mathcal{L}(f(\widetilde{\mathbf{x}}), y) - \lambda \max\left(0, d(\widetilde{\mathbf{x}}, \mathbf{x}) - \epsilon\right)\right]$ ▷ take the gradient of (5)
   $\hat{\Delta} = \Delta / \|\Delta\|_2$ ▷ normalize the gradient
   $\eta = \epsilon * (0.1)^{t/T}$ ▷ the step size $\eta$ decays exponentially
   $m \leftarrow d(\widetilde{\mathbf{x}}, \widetilde{\mathbf{x}} + h\hat{\Delta})/h$ ▷ $m \approx$ derivative of $d(\widetilde{\mathbf{x}}, \cdot)$ in the direction of $\hat{\Delta}$; $h = 0.1$
   $\widetilde{\mathbf{x}} \leftarrow \widetilde{\mathbf{x}} + (\eta/m)\hat{\Delta}$ ▷ take a step of size $\eta$ in LPIPS distance
  **end for**
  **if** $d(\widetilde{\mathbf{x}}, \mathbf{x}) > \epsilon$ **then**
   $\lambda \leftarrow 10\lambda$ ▷ increase $\lambda$ if the attack goes outside the bound
  **end if**
 **end for**
 $\widetilde{\mathbf{x}} \leftarrow \text{PROJECT}(d, \widetilde{\mathbf{x}}, \mathbf{x}, \epsilon)$
 **return** $\widetilde{\mathbf{x}}$
**end procedure**

**Ref** Perceptual adversarial robustness: Defense against unseen threat models. Laidlaw, C., Singla, S., & Feizi, S. https://arxiv.org/abs/2006.12655

# Problem with penalty methods

| | 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 |

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

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

**LPA, Fast-LPA**: penalty methods **PPGD**: Projected gradient descent

**PWCF**, an optimizer with a principled stopping criterion on **stationarity & feasibility**

Penalty methods tend to encounter **large constraint violation** (i.e., infeasible solution, known in optimization theory) or **suboptimal solution**

**Ref** Optimization and Optimizers for Adversarial Robustness. Liang, H., Liang, B., Peng, L., Cui, Y., Mitchell, T., & Sun, J. arXiv preprint arXiv:2303.13401.

Unreliable optimization ==
Unreliable RE

# Outline

**Constrained deep learning: CDL**

- What, how, and why for CDL
- **No good solvers for CDL yet**
- Granso and PyGranso
- PyGranso in action
- Outlook

# DL frameworks

JAX: Autograd and XLA

mxnet

PyTorch

TensorFlow

**For unconstrained DL problems**

# Convex optimization solvers and frameworks

CVX RESEARCH

AMPL


**Modeling languages**

IBM CPLEX

GUROBI OPTIMIZATION

**SDPT3 - a MATLAB software package for semidefinite-quadratic-linear programming**

K. C. Toh, R. H. Tütüncü, and M. J. Todd.

**TFOCS: Templates for First-Order Conic Solvers**

**Solvers**

**Not for DL**, which involves NCVX optimization

Note: Gurobi can handle certain NCVX problems

# Manifold optimization

Manopt.jl

Geomstats

$\mathcal{T}_p\mathcal{G}$

geoopt

**McTorch Lib, a manifold optimization library for deep learning**

---

Only for **differentiable manifolds constraints**

# General constrained optimization

KNITRO

COIN|OR

IPOPT

ensmallen
flexible C++ library for efficient numerical optimization

GENO

**Interior-point methods**

**Augmented Lagrangian methods**

Cooper

TensorFlow Constrained Optimization (TFCO)

**Lagrangian-method-based constrained optimization**

# Specialized ML packages

Problem-specific solvers that **cannot be easily extended** to new formulations

# Outline

**Constrained deep learning: CDL**

- What, how, and why for CDL
- No good solvers for CDL yet
- **Granso and PyGranso**
- PyGranso in action
- Outlook

# Issues with typical CDL methods

**projected gradient descent**

$$\min_{\mathbf{x}\in\mathcal{Q}} f(\mathbf{x})$$

$$\mathbf{x}_{k+1} = P_{\mathcal{Q}}\Big(\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)\Big)$$

**Issue: no principled stopping criterion/step size rules**

**penalty methods**

$$\min_{\boldsymbol{x}} \ f(\boldsymbol{x}) \quad \text{s.t.} \ g(\boldsymbol{x}) \leq \mathbf{0}$$

$$\min_{\boldsymbol{x}} \ f(\boldsymbol{x}) + \lambda \max(0, g(\boldsymbol{x}))$$

Solved with increasing $\lambda$ sequence

**Issue: infeasible solution**

**Lagrangian method**

$$\min_{\boldsymbol{x}} \max_{\boldsymbol{\lambda}\geq 0} \ f(\boldsymbol{x}) + \boldsymbol{\lambda}^\mathsf{T} g(\boldsymbol{x})$$

**Idea: alternating minimize $\boldsymbol{x}$ and maximize $\lambda$ via gradient descent**

**Issues**

- Infeasible solution
- Slow convergence

**Want**

- **Feasible & stationary solution**
- **Reasonable speed**

# Principled answers to these questions

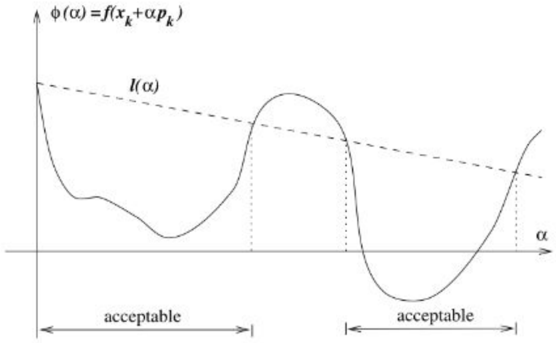
- **Feasible & stationary solution**

  **Stationarity and feasibility check: KKT condition**

- **Reasonable speed**

  **Line search**

- **A hidden problem: nonsmoothness**




Armijo (Sufficient Decrease) Condition

# Key algorithm

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

**Nonconvex, nonsmooth, constrained**

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

**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

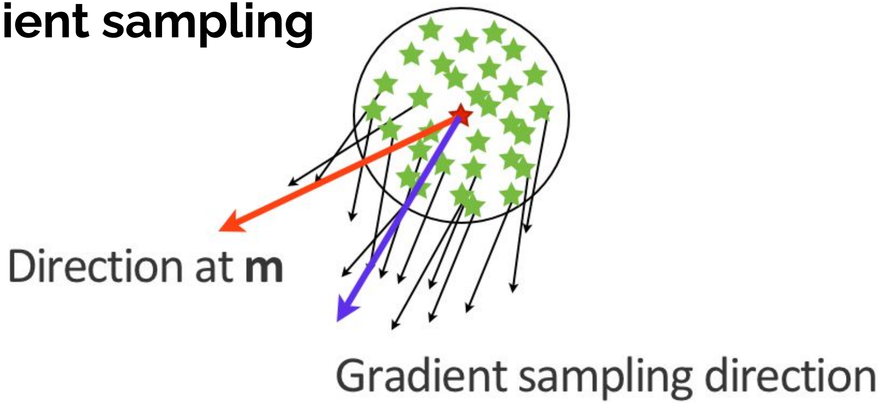
**Steering strategy for the penalty parameter**

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

**Stationarity based on (approximate) gradient sampling**

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

$$\min_{\lambda \in \mathbb{R}^{m+1}} \ \frac{1}{2}\|G_k \lambda\|_2^2$$
$$\text{s.t. } \mathbb{1}^T \lambda = 1, \ \lambda \geq 0$$



# Key take-away

GRANSO

- Principled stopping criterion and line search, to obtain a **solution with certificate** (stationarity & feasibility check)
- Quasi-newton style method for fast convergence, i.e., **reasonable speed and high-precision solution**

**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**

# GRANSO meets PyTorch

GRANSO + PyTorch

PyGRANSO

NCVX PyGRANSO Documentation

Search the docs ...

- Introduction
- Installation
- Settings
- Examples

## Home

NCVX

## NCVX Package

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

**First general-purpose solver for constrained DL problems**

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

Buyun Liang, Tim Mitchell, Ju Sun

# Outline

**Constrained deep learning: CDL**

- What and how for CDL
- Why CDL
- No good solvers for CDL yet
- Granso and PyGranso
- **PyGranso in action**
- Outlook

# Example 1: Support Vector Machine (SVM)

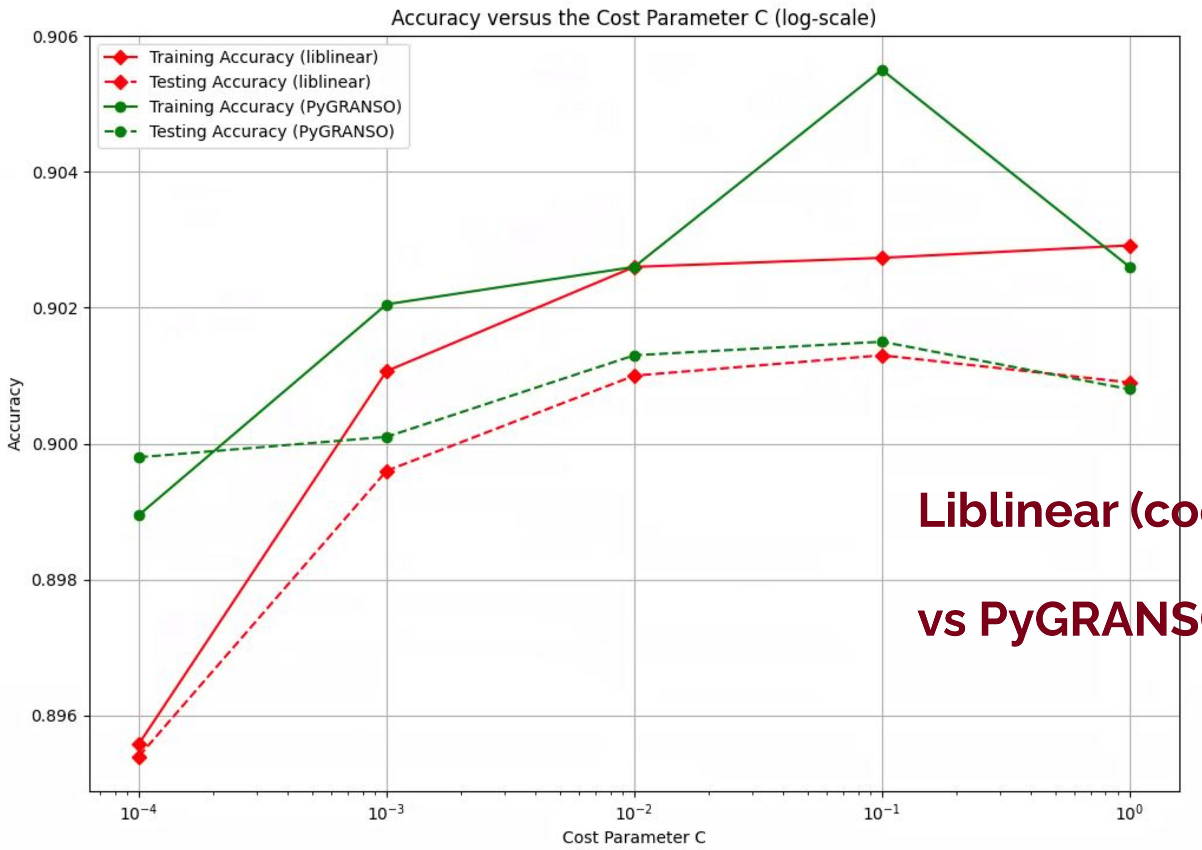
**Soft-margin SVM**

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

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

```
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)
```

# Binary classification (odd vs even digits) on MNIST dataset




**Liblinear (coordinate descent)**

**vs PyGRANSO**

# Example 2: Robustness eval. —min formulation

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

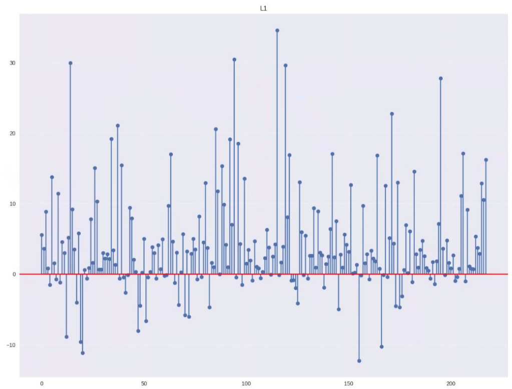
```python
def comb_fn(X_struct):
    # obtain optimization variables
    x_prime = X_struct.x_prime
    # objective function
    f = d(x,x_prime)
    # inequality constraints
    ci = pygransoStruct()
    f_theta_all = f_theta(x_prime)
    fy = f_theta_all[:,y] # true class output
    # output execpt 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)
```

**CIFAR10 dataset**

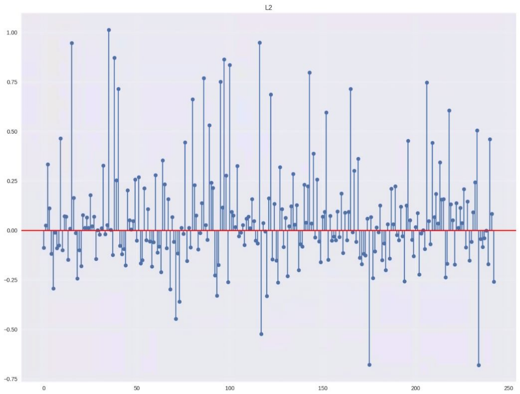
**Compared with FAB [iterative constraint linearization + projected gradient]**
[https://github.com/fra31/auto-attack](https://github.com/fra31/auto-attack)

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

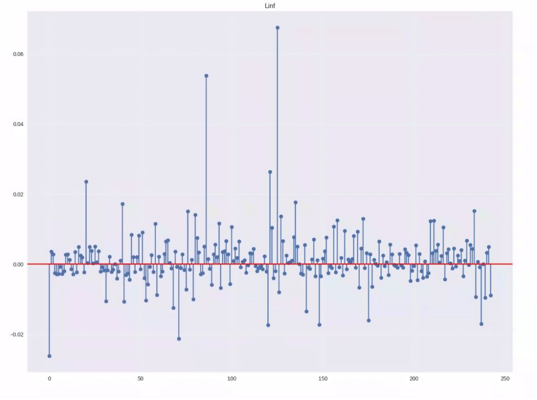
X-axis: image index; Y-axis: PyGRANSO radius - FAB radius




**L1 attack**




**L2 attack**




**Linf attack**

# Many others

https://ncvx.org/

NCVX PyGRANSO Documentation

Search the docs ...

- Introduction
- Installation
- Settings
- Examples
  - Rosenbrock
  - Eigenvalue Optimization
  - Dictionary Learning
  - Nonlinear Feasibility Problem
  - Sphere Manifold
  - Trace Optimization
  - Robust PCA
  - Generalized LASSO
  - Logistic Regression
  - LeNet5
  - Perceptual Attack
  - Orthogonal RNN
- Highlights

## Home

NCVX

### NCVX Package

**NCVX (NonConVeX)** is a user-friendly and scalable python software package targeting general nonsmooth NCVX problems with nonsmooth constraints. **NCVX** is being developed by GLOVEX at the Department of Computer Science & Engineering, University of Minnesota, Twin Cities.

The initial release of **NCVX** contains the solver **PyGRANSO**, a PyTorch-enabled port of GRANSO incorporating auto-differentiation, GPU acceleration, tensor input, and support for new QP solvers. As a highlight, **PyGRANSO** can solve general constrained deep learning problems, the first of its kind.

PyGRANSO

# Our current developments

- PyGranso: deterministic SQP solver
- More scalable deterministic solvers (exact penalty, single-loop ALM, etc)
- Algorithm foundations for stochastic objective with stochastic constraints/numerous constraints
- Scalable solvers for stochastic objective with stochastic constraints/numerous constraints

# Closing

NCVX

## Deep Learning with Nontrivial Constraints: Methods and Applications

Chuan He[1], Ryan Devera[1], Wenjie Zhang[1], Ying Cui[2], Zhaosong Lu[3] and Ju Sun[1]

[1]Computer Science and Engineering, University of Minnesota
[2]Industrial Engineering and Operations Research, University of California, Berkeley
[3]Industrial and Systems Engineering, University of Minnesota
{he000233, dever120, zhan7867}@umn.edu, yingcui@berkeley.edu, {zhaosong, jusun}@umn.edu

PyGRANSO

NCVX PyGRANSO Documentation

Search the docs ...

Introduction
Installation
Settings
Examples

Home

NCVX

NCVX Package

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

First general-purpose solver for constrained DL problems

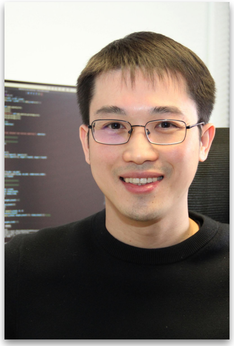
**NCVX: A General-Purpose Optimization Solver for Constrained Machine and Deep Learning**

Buyun Liang, Tim Mitchell, Ju Sun

# Thanks to all contributors

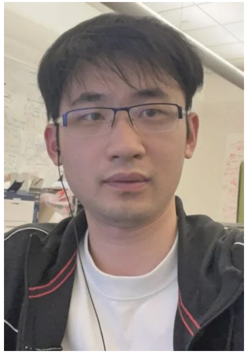
**Prof. Tim Mitchell**
(CS, Queens Col.)

**Prof. Ying Cui**
(IEOR, UC Berkely)



**Prof. Qizhi He**
(CSGE, UMN)

**Prof. Zhaosong Lu**
(CSGE, UMN)



**Dr. Chuan He**
(CS&E, UMN)

# Thanks to all contributors

**Buyun Liang**
(CSE U Penn;
Formally, CS&E, UMN)

**Ryan de Vera**
(CS&E, UMN)

**Hengyue Liang**
(ECE, UMN)

**Wenjie Zhang**
(CS&E, UMN)

**Yash Travadi**
(CS&E, UMN)

**Le Peng**
(CS&E, UMN)