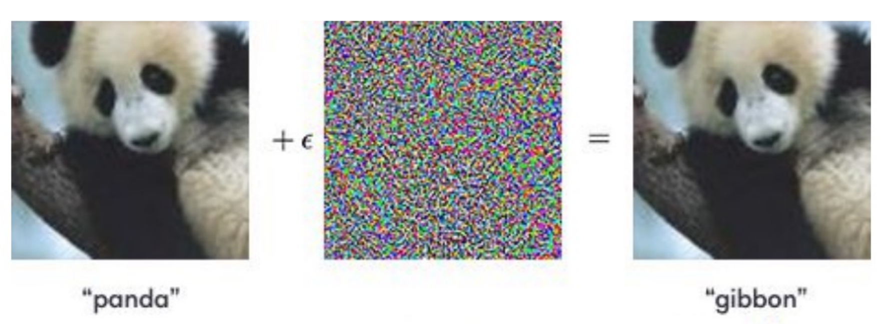
When Deep Learning Meets Nontrivial Constraints

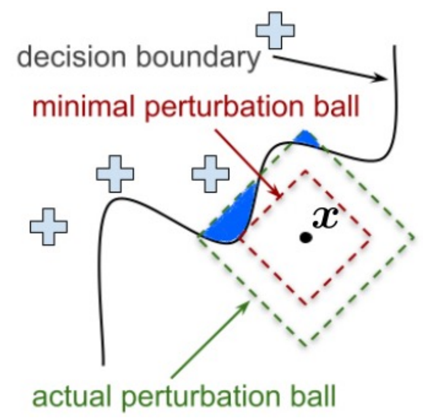
Buyun Liang[1], Wenjie Zhang[1], Ryan de Vera[1], Hengyue Liang[2], Tim Mitchell[3], Ju Sun[1]
[1] Department of Computer Science and Engineering, University of Minnesota
[2] Department of Electrical and Computer Engineering, University of Minnesota
[3] Department of Computer Science, Queens College, City University of New York

1. Motivating examples & methods

(Constrained deep learning: CDL)

1.1 Robustness evaluation





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

Maximum adversarial loss

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

Minimum distortion radius

- **Projected gradient descent**

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

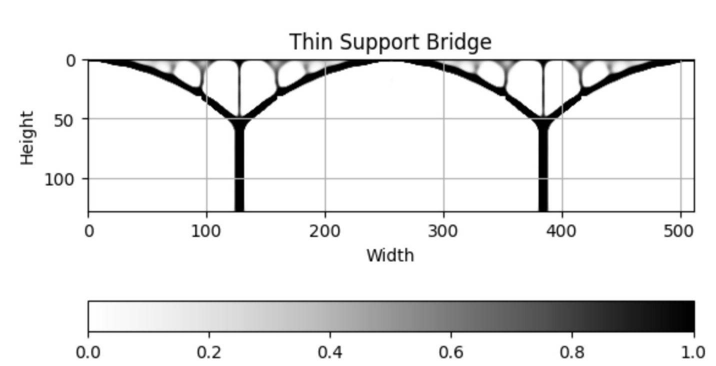
- **Penalty method**

  Problem: large **constraint violation** or **suboptimal** solution

1.2 Neural Topology Optimization

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

Neural structural optimization



Solution from *PyGRANSO* (ours)

Cons of SOTA **unconstrained optimization methods**:

- **Solving linear systems** to eliminate the physical constraint
- Use **problem specific technique** to handle design constraints
- Cannot handle **discrete-valued optimization variables**

1.3 Other problems

- **Lagrangian methods for imbalanced learning: infeasible solution, slow convergence**
- **Augmented Lagrangian methods for PINNs: infeasible solution**
- **First-order solver for PINNs: low quality solution**

2. No good solvers for CDL yet

Solvers or modeling languages	Nonconvex	Nonsmooth	Differentiable manifold constraints	General smooth constraint	Specific constrained ML problem	General CDL
PyTorch, Tensorflow, JAX, MXNet	✓	✓	✗	✗	✗	✗
CVX, AMPL, YALMIP, SDPT3, Cplex, Gurobi*, SDPT3, TFOCS	✗	✓	✗	✗	✗	✗
(Py)manopt, Geomstats, McTorch, Geoopt	✓	✓	✓	✗	✗	✗
KNITRO, IPOPT, GENO, ensmallen, TFCO, Cooper	✓	✓	✓	✓	✗	✗
Scikit-learn, MLib, Weka	✓	✓	✗	✗	✓	✗

3. A solver for constrained optimization

- **Principled answers to issues in CDL methods**

  Stationarity & **feasibility** check: KKT condition
  Line search methods
  Gradient-sampling-based idea for nonsmoothness

- **A principled solver: GRANSO**

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} f(\boldsymbol{x}), s.t. \; c_i(\boldsymbol{x}) \le 0, \forall i \in \zeta; c_j(\boldsymbol{x}) = 0, \forall j \in \xi$$

Nonconvex, nonsmooth, constrained

Keep advantages:

Principled stopping criterion and **line search**
⇒ obtain a solution with certificate
BFGS-Sequential quadratic programming
⇒ reasonable speed and high-precision solution

Problems:

Lack of **auto-differentiation**
Lack of **GPU** Support
No native support of **tensor** variables

⇒ impossible to do deep learning with GRANSO!

4. NCVX PyGRANSO

First general-purpose solver for CDL

Advantages:

Auto-differentiation; GPU Support; support of **tensor** variables

Auto-Differentiation

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

Orthogonal dictionary learning

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

GRANSO

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

PyGRANSO

General Tensor Variables

```
var_in = {"M":[d1,d2],"S":[d1,d2]}
# objective function
f = torch.norm(M, p='nuc') + eta * torch.norm(S, p = 1)
```

Matrix input

```
var_in = {"x_tilde":list(inputs.shape)}
adv_inputs = X_struct.x_tilde
epsilon = eps
logits_outputs = model(adv_inputs)
f = -torch.nn.functional.cross_entropy(logits_outputs,labels)
```

Higher order tensor input

Constraint-folding

Reduce # of constraints: reduce the cost of QP in the SQP

$$h_j(\boldsymbol{x}) = 0 \Leftrightarrow |h_j(\boldsymbol{x})| \le 0, \quad \textbf{Equality Constraint}$$

$$c_i(\boldsymbol{x}) \le 0 \Leftrightarrow \max\{c_i(\boldsymbol{x}), 0\} \le 0, \quad \textbf{Inequality Constraint}$$

$$\mathcal{F}(|h_1(\boldsymbol{x})|, \cdots, |h_j(\boldsymbol{x})|, \max\{c_1(\boldsymbol{x}), 0\} \cdots, \max\{c_i(\boldsymbol{x}), 0\}) \le 0$$

Constrained Deep Learning Applications

See ncvx.org for detailed examples for CDL!

Ref: [1] Liang, B., Mitchell, T., & Sun, J. (2022). NCVX: A general-purpose optimization solver for constrained machine and deep learning. In OPT 2022: Optimization for Machine Learning (NeurIPS 2022 Workshop). [2] Liang, H., Liang, B., Peng, L., Cui, Y., Mitchell, T., & Sun, J. (2023). Optimization and Optimizers for Adversarial Robustness. arXiv preprint arXiv:2303.13401..
[3] Liang, H., Liang, B., Cui, Y., Mitchell, T., & Sun, J. (2022). Optimization for robustness evaluation beyond ℓp metrics. In OPT 2022: Optimization for Machine Learning (NeurIPS 2022 Workshop).