



清华大学
Tsinghua University

達摩院
ALIBABA DAMO ACADEMY



GLinSAT: The General Linear Satisfiability Neural Network Layer By Accelerated Gradient Descent

Hongtai Zeng¹, Chao Yang², Yanzhen Zhou¹, Cheng Yang², Qinglai Guo^{1*}

¹ State Key Laboratory of Power Systems, Department of Electrical Engineering, Tsinghua University

² Decision Intelligence Lab, Alibaba DAMO Academy

Content

01

Background

02

Methodology

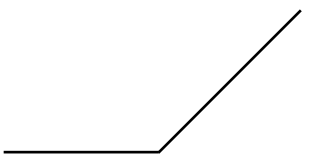
03

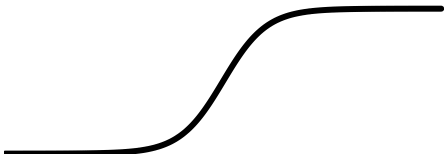
Experiments

1 Background



- ◆ Many researchers want to use neural networks to predict the optimal solution of **constrained decision-making problems** for accelerating solution process.
- ◆ Unfortunately, standard activation layers can only **impose simple constraints** on their outputs.


$$x = \text{ReLU}(c)$$
$$\Leftrightarrow x \geq 0$$


$$x = \text{Sigmoid}(c)$$
$$\Leftrightarrow 0 \leq x \leq 1$$

$$\varphi(c) = \frac{\exp(c)}{\mathbf{1}^T \exp(c)}$$
$$x = \text{Softmax}(c)$$
$$\Leftrightarrow x \geq 0, \mathbf{1}^T x = 1$$

- ◆ How to make neural network outputs **satisfy general linear constraints**?

$$A'_1 x \leq b'_1, A'_2 x = b'_2, A'_3 x \geq b'_3, l' \leq x \leq u'$$

1 Background



清华大学
Tsinghua University

達摩院
ALIBABA DAMO ACADEMY



Possible solutions ...

- ◆ Penalize constraint violation in loss
 - ⇒ Hard to choose penalty coefficient and **violation can be unbounded**
- ◆ Reinforcement learning with hard-coded action space
 - ⇒ **Limited applicable scenarios**
- ◆ Differentiable optimizer based methods
 - ⇒ encounter **dilemma in supported constraint types and efficiency**

1 Background



清华大学
Tsinghua University

達摩院
ALIBABA DAMO ACADEMY



Method	Supporting Constraints	GPU-based	Matrix-factorization-free
Perturbed Optimizer	Combinatorial		
Sinhorn	Double Stochastic Matrix		
SATNet	PSD Matrix with unit diagonals		
CvxpyLayers	Linear & Conic		
OptNet	Linear		
LinSAT	Positive Linear		
GLinSAT (ours)	Linear		

Content

01

Background

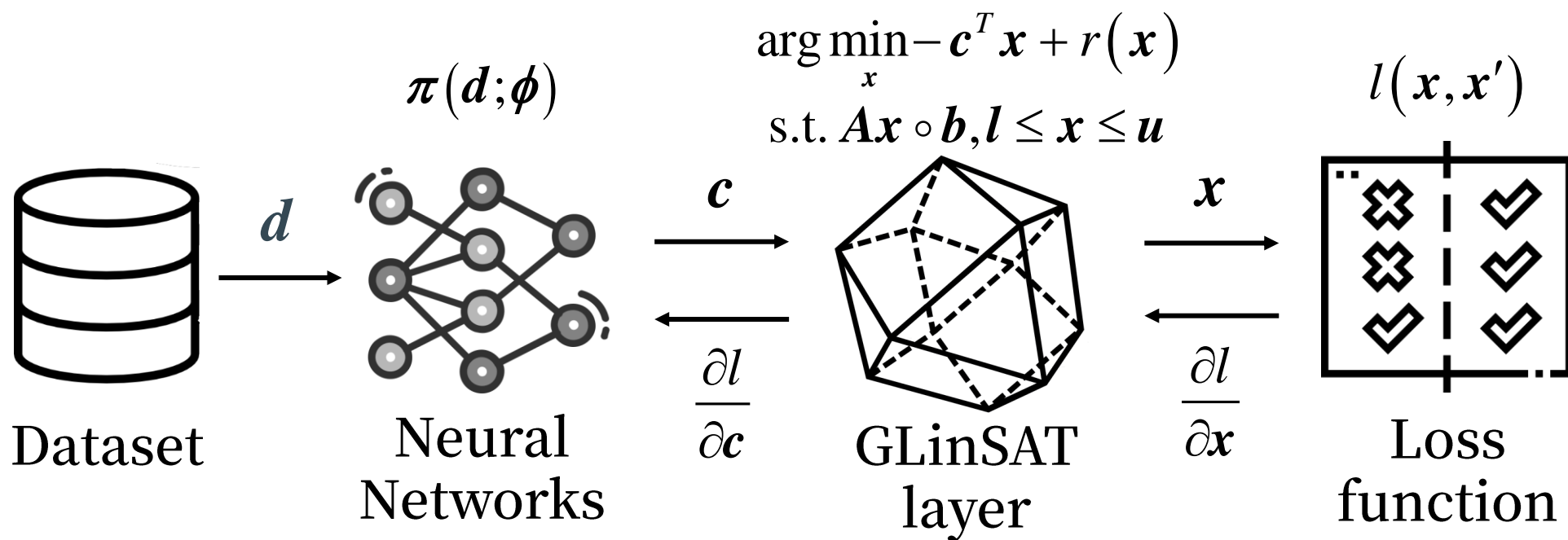
02

Methodology

03

Experiments

2 Methodology



A pipeline that shows how GLinSAT layer works.

2 Methodology



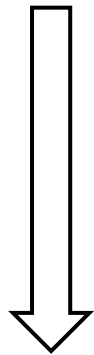
清华大学
Tsinghua University

達摩院
ALIBABA DAMO ACADEMY



- ◆ Our aim is to make neural network outputs **satisfy general linear constraints** while **maximizing utilization of GPU parallelism**
- ◆ By adding slack variables and linearly transforming the variables, we first **convert the linear constraint into a standard form**

$$A_1x \leq b_1, A_2x \geq b_2, A_3x = b_3, l' \leq x \leq u'$$



$$Ax = b, \mathbf{0} \leq x \leq u$$

2 Methodology



- ◆ We use dot product to measure similarity between c and x , and reformulate projection problem as **a logistic entropy-regularized linear programming problem** to make the problem differentiable
- ◆ We show that the problem can be transformed into **an unconstrained convex optimization problem with Lipschitz continuous gradient**.

$$\begin{array}{ccc}
 A_1 x \leq b_1, A_2 x \geq b_2, A_3 x = b_3, l' \leq x \leq u' & \min -c^T x + \frac{1}{\theta} \mathbf{1}^T \left(\frac{x}{u} \circ \log \frac{x}{u} + \left(\mathbf{1} - \frac{x}{u} \right) \circ \log \left(\mathbf{1} - \frac{x}{u} \right) \right) & \\
 & \text{s.t. } Ax = b, \mathbf{0} \leq x \leq u & \\
 \downarrow & \nearrow & \downarrow \\
 Ax = b, \mathbf{0} \leq x \leq u & & \min -\frac{1}{\theta} \mathbf{1}^T \log \sigma \left(\theta u \circ (-c - A^T y) \right) - b^T y
 \end{array}$$

2 Methodology



- ◆ The subsequent question is how to use **GPU for efficient solution** in the calculation.
- ◆ Note that the problem can be transformed into **an unconstrained convex optimization problem with Lipschitz continuous gradient**
- ◆ In forward pass, we design a **batch matrix-factorization-free** algorithm that can efficiently utilize the GPU based on the **adaptive primal-dual accelerated gradient descent method (APDAGD)**.

Algorithm 1: Solving the entropy-regularized linear programming problem in GLinSAT

Input: $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$, $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{u} \in \mathbb{R}_+^n$, inverse temperature $\theta > 0$, tolerance $\varepsilon > 0$, initial estimate of Lipschitz constant $L^{(0)}$, initial estimate of dual variables $\boldsymbol{\eta}^{(0)}$, numerical precision $\delta > 0$

Set $k = 0$, $M^{(0)} = L^{(0)}$, $\boldsymbol{\eta}^{(0)} = \boldsymbol{\zeta}^{(0)} = \mathbf{y}^{(0)}$,
 $\mathbf{x}^{(0)} = \mathbf{u} \circ \sigma(-\theta \mathbf{u} \circ (-\mathbf{c} - \mathbf{A}^T \mathbf{y}^{(0)}))$, $\beta^{(0)} = \alpha^{(0)} = 0$, $f = \text{False}$;

while $\|\mathbf{Ax}^{(k)} - \mathbf{b}\|_2 > \varepsilon$ **do**

- Set $\alpha^{(k+1)} = (1 + \sqrt{1 + 4M^{(k)}\beta^{(k)}}) / (2M^{(k)})$;
- Set $\beta^{(k+1)} = \beta^{(k)} + \alpha^{(k+1)}$;
- Set $\tau^{(k+1)} = \alpha^{(k+1)} / \beta^{(k+1)}$;
- Set $\boldsymbol{\lambda}^{(k+1)} = \boldsymbol{\eta}^{(k)} + \tau^{(k+1)} (\boldsymbol{\zeta}^{(k)} - \boldsymbol{\eta}^{(k)})$;
- Set $\mathbf{x}(\boldsymbol{\lambda}^{(k+1)}) = \mathbf{u} \circ \sigma(-\theta \mathbf{u} \circ (-\mathbf{c} - \mathbf{A}^T \boldsymbol{\lambda}^{(k+1)}))$;
- Set $\boldsymbol{\zeta}^{(k+1)} = \boldsymbol{\zeta}^{(k)} - \alpha^{(k+1)} (\mathbf{Ax}(\boldsymbol{\lambda}^{(k+1)}) - \mathbf{b})$;
- Set $\boldsymbol{\eta}^{(k+1)} = \boldsymbol{\eta}^{(k)} + \tau^{(k+1)} (\boldsymbol{\zeta}^{(k+1)} - \boldsymbol{\eta}^{(k)})$;
- if** $(-g(\boldsymbol{\eta}^{(k+1)})) - (-g(\boldsymbol{\lambda}^{(k+1)})) - \delta \leq -\|\mathbf{Ax}(\boldsymbol{\lambda}^{(k+1)}) - \mathbf{b}\|_2^2 / (2M^{(k)})$ **then**
 - if** $f = \text{True}$ **then**
 - Set $M^{(k+1)} = M^{(k)} / 2$;
 - else**
 - Set $M^{(k+1)} = M^{(k)}$;
 - end**
 - Set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \tau^{(k+1)} (\mathbf{x}(\boldsymbol{\lambda}^{(k+1)}) - \mathbf{x}^{(k)})$, $f = \text{True}$;
 - Set $k = k + 1$;
- else**
 - Set $M^{(k)} = 2M^{(k)}$, $f = \text{False}$;
- end**

end

Output: Optimal primal variables $\mathbf{x}^{(k)}$, Optimal dual variables $\boldsymbol{\eta}^{(k)}$

2 Methodology



- ◆ The subsequent question is how to use **GPU for efficient solution** in the calculation.
- ◆ Note that the problem can be transformed into **an unconstrained convex optimization problem with Lipschitz continuous gradient**
- ◆ In backward pass, we not only designed a direct derivation method using **Pytorch's automatic differentiation**, but also designed a method using **implicit differentiation based on KKT conditions**.

$$h(\mathbf{y}) = A \left(\mathbf{u} \circ \sigma \left(-\theta \mathbf{u} \circ \left(-\mathbf{c} - A^T \mathbf{y} \right) \right) \right) - \mathbf{b} = \mathbf{0}$$

$$\frac{\partial l}{\partial \mathbf{c}} = \frac{\partial l}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial \mathbf{c}} - \left(\frac{\partial l}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial \mathbf{y}} + \frac{\partial l}{\partial \mathbf{y}} \right) \left(\frac{\partial \mathbf{h}}{\partial \mathbf{y}} \right)^{-1} \frac{\partial \mathbf{h}}{\partial \mathbf{c}}$$

Content

01

Background

02

Methodology

03

Experiments

3 Experiments



清华大学
Tsinghua University

達摩院
ALIBABA DAMO ACADEMY



- In large-scale unit commitment, we predict the optimal unit status and make them satisfy the key rigid constraints: the minimum uptime/downtime constraints.
- Matrix with more than 1,000,000 rows and 2,000,000 columns in one batch
- Compared with the existing linear satisfiability layer OptNet and CvxpyLayers, **our method achieves 10 times or even 100 times acceleration**
- We can find the satisfiability layer can significantly **improve the feasibility of neural network prediction while ensuring a certain degree of optimality.**

Comparison of computation time of linear satisfiability layers in unit commitment

Method	Time used in forward pass/s	Time used in backward pass/s
CvxpyLayers	2771	684.0
OptNet	257.4	23.60
GLinSAT	26.78	1.636

Feasibility ratio and average gap after fixing unit status via neural networks

Parameter $1/\theta$	Feasibility Ratio	Average Optimality Gap
0.01	86.23%	0.1119%
0.001	98.17%	0.1109%
0.0001	100%	0.1114%

3 Experiments



清华大学
Tsinghua University

達摩院
ALIBABA DAMO ACADEMY



- Since GLinSAT can be applied to general linear constraints, we also apply GLinSAT to the **Traveling Salesman Problem (TSP)**, **partial graph matching** to illustrate the effectiveness of GLinSAT.
- Compared with existing satisfiability layers, **our method requires less memory and can achieve significant acceleration**, improving training efficiency and making the training process which was previously impractical now viable and efficient.

Comparison of computation time of linear satisfiability layers in TSP

Method	TSP-StartEnd		TSP-Priority	
	Batch Memory/MB	Batch Time/s	Batch Memory/MB	Batch Time/s
CvxpyLayers	---	130.49	---	136.44
OptNet	19310	19.849	19338	21.396
LinSAT	74289.2	3.25	74452.9	3.246
GLinSAT	66.58	0.449	66.58	0.495

Note: The GPU memory used by CvxpyLayers is not counted since CvxpyLayers use the CPU parallel mechanism

Comparison of computation time of linear satisfiability layers in partial graph matching

Method	Batch Memory/MB	Batch Time/s
CvxpyLayers	---	80.35
OptNet	993.5	8.002
LinSAT	3076.3	8.456
GLinSAT	862.1	5.428

Note: The GPU memory used by CvxpyLayers is not counted since CvxpyLayers use the CPU parallel mechanism

Thanks!

Paper



Code

