Constraint-aware Machine Learning

An end-to-end Integration of Constrained Optimization in ML

Ferdinando Fioretto

https://nandofioretto.com

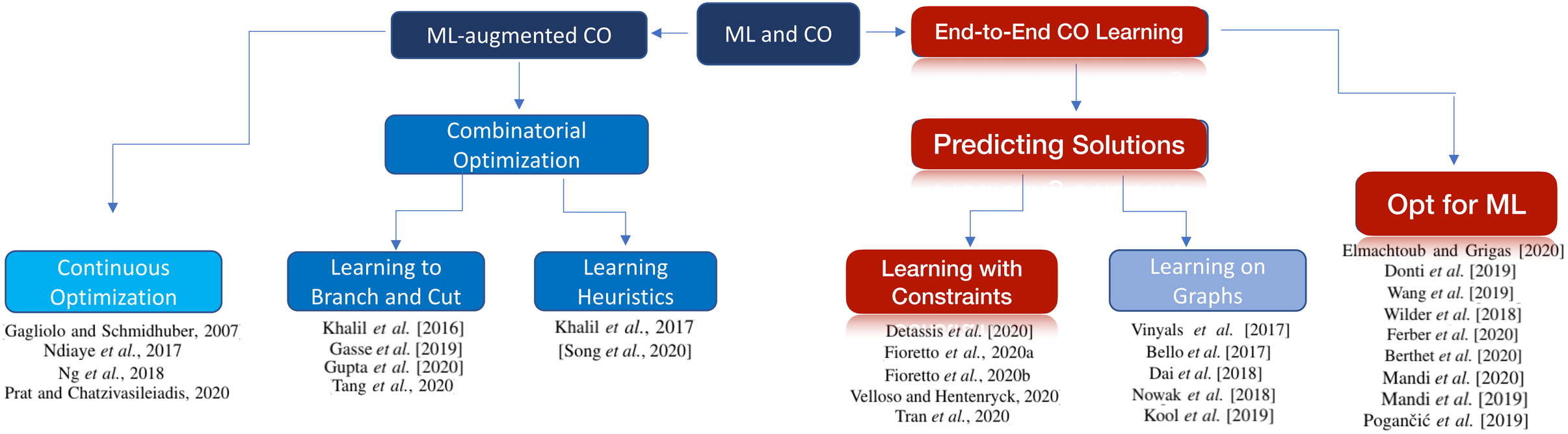
@nandofioretto

nandofioretto@gmail.com

ACP Summer School 2023

Machine Learning and Constrained Optimization

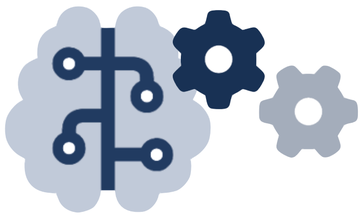
Main research branches



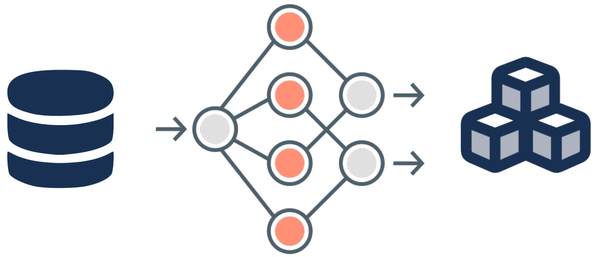
Kotary et al., IJCAI-21

ML for Optimization

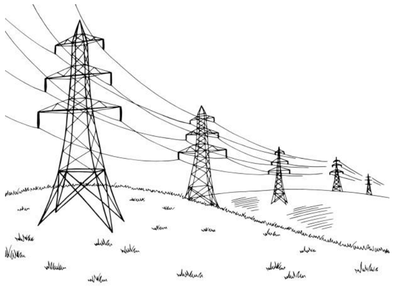
Constrained-aware ML



ML Proxy optimizers



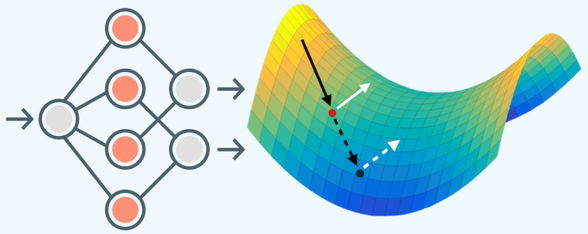
Power Systems



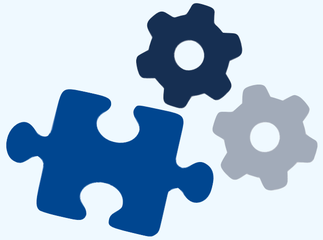
Model pruning

Optimization for ML

Differentiable optimization

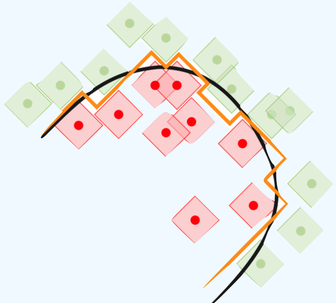


Optimization layers



Bias and Fairness

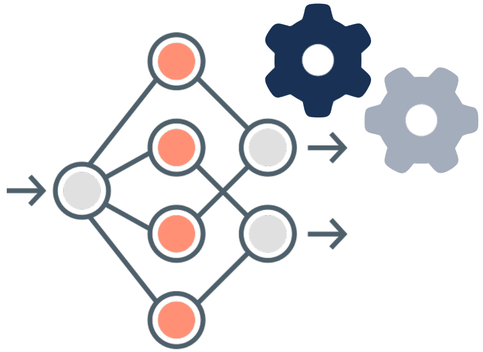
Model selection



Agenda

Intro and Motivations

ML for Optimization



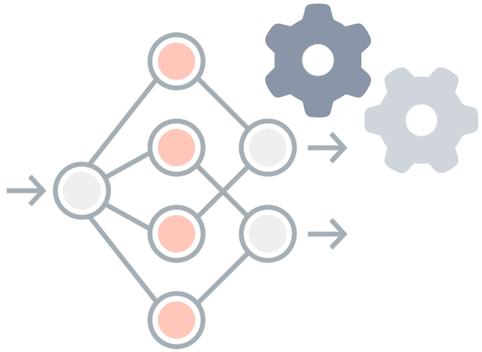
Optimization for ML

Open Questions

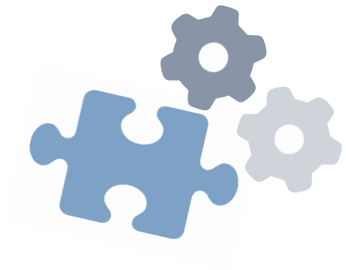
Agenda

Intro and Motivations

ML for Optimization



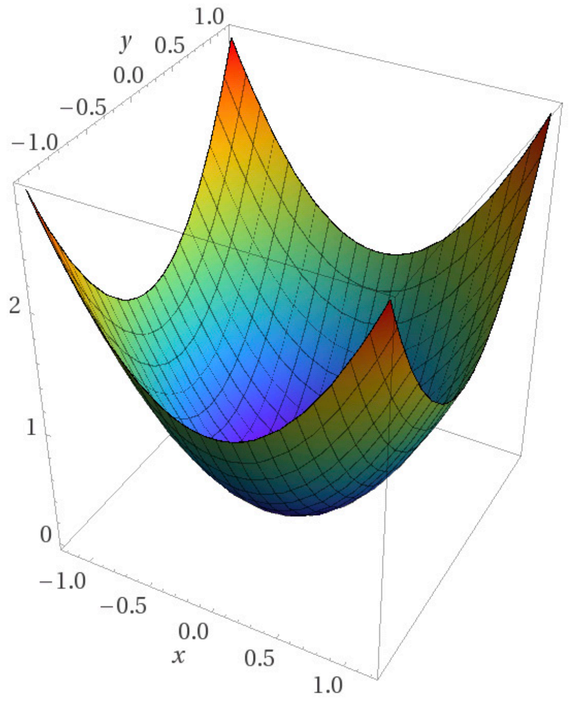
Optimization for ML



Open Questions

Constrained Optimization

Preliminaries



- Consider a problem of the form

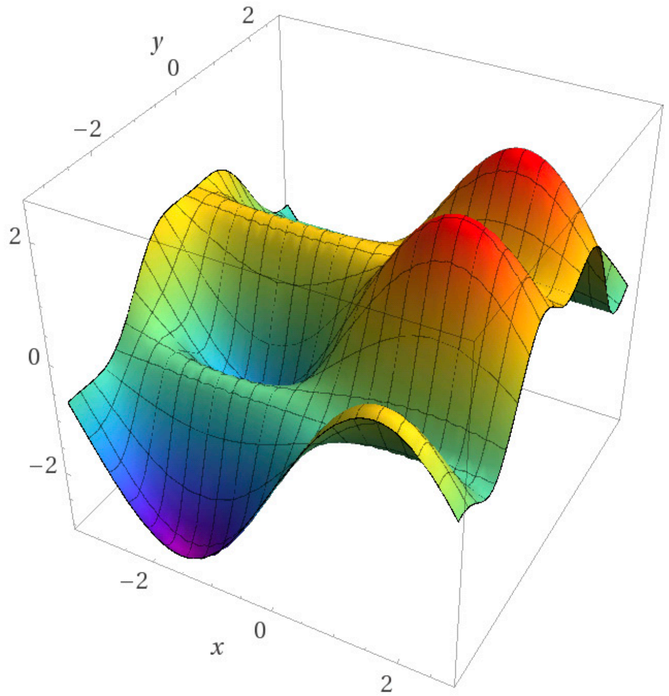
$$\mathcal{O} = \arg\min_{z} \;\; f(z)$$

$$\text{subject to} \;\; z \in \mathcal{C}$$

- with objective function $f : \mathbb{R}^n \to \mathbb{R}_+$ with decision variables $z \in \mathbb{R}^n$, subject to a set of constraints $\mathbf{C}$.
- The form of f and $\mathbf{C}$ characterizes the complexity of optimization problems.
- **Convex problems**: f convex function and $\mathbf{C}$ convex set — efficiently solvable & strong theoretical guarantees on the existence and uniqueness of solutions.
  - Linear problems (LP): $\mathbf{C} = \{z : Az \leq b\}$ convex set with $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and f affine function.
  - Quadratic programs (QP): $\mathbf{C}$ as above but with a quadratic objective.

Constrained Optimization

Preliminaries



- Consider a problem of the form

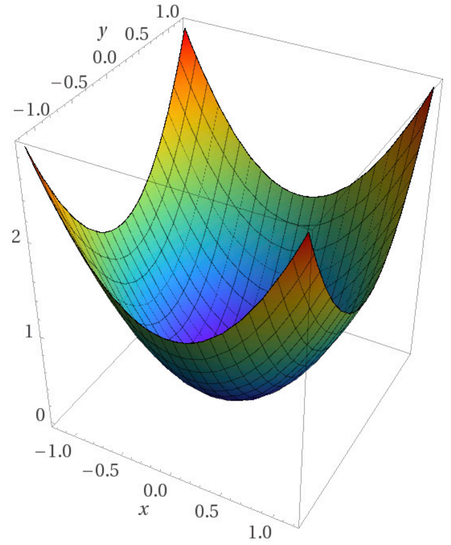
$$\mathcal{O} = \arg\min_{z} \; f(z)$$
$$\text{subject to} \;\; z \in \mathcal{C}$$

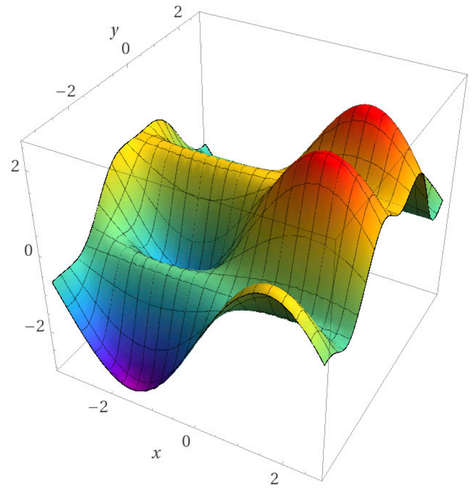
- with objective function $f : \mathbb{R}^n \to \mathbb{R}_+$ with decision variables $z \in \mathbb{R}^n$, subject to a set of constraints $\mathbf{C}$.
- The form of f and $\mathbf{C}$ characterizes the complexity of optimization problems.
- **Mixed integer program (MIP)**: A subset of the decision variables is required to take integer values.
  - Much more difficult problems than convex problems; The feasible set consists of distinct points in $\mathbb{R}^n$, not only non-convex but also disjoint. Generally NP-Hard.
- **Nonlinear programs (NLP)**: Some of the constraints or the objective function are nonlinear. Many NLPs are nonconvex and cannot be efficiently solved.

Constrained Optimization

Solving methods

- **Convex** programs enjoy well-developed theory for their resolution (including simplex, interior point, and augmented Lagrangian methods).
- **NLPs** (non-convex) include variant of methods used for convex programs (penalty-based methods, interior points,…)
- **MILPs** require different approaches (NP-Hard), e.g., branch and bound.
- **Constraint Programming** is an additional paradigm to solve MI(L)P by combining branch and bound with reasoning methods.



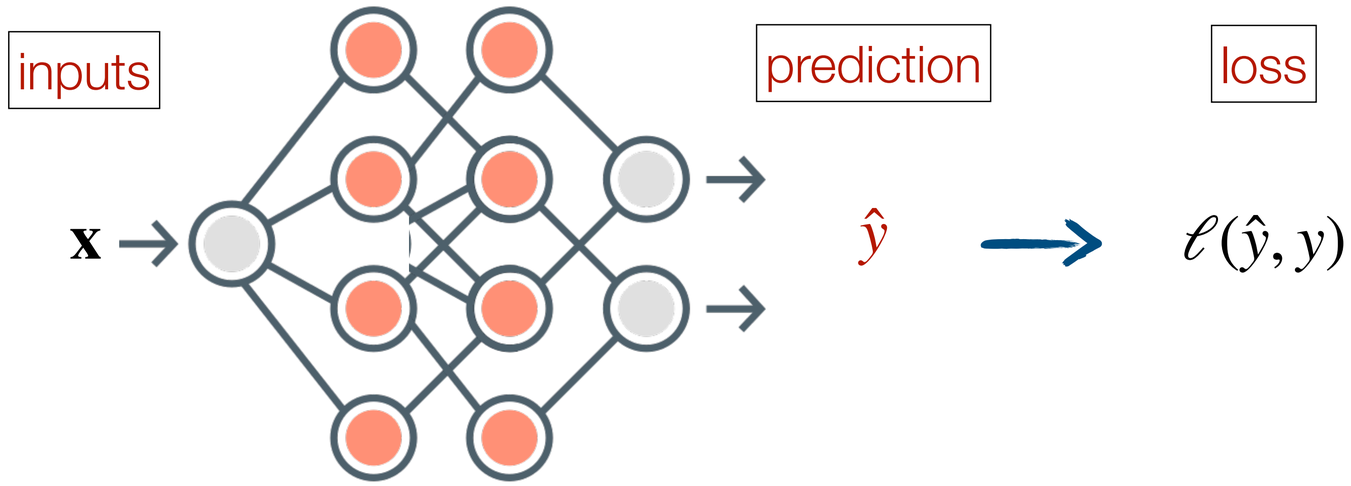


Boyd et al., 2004 Rossi et al., 2006 Powell, 1969

Supervised learning

Preliminaries

- Training dataset $D = (\mathbf{x}_i, y_i)_{i=1}^n$ with each $\mathbf{x} = (x_1, \ldots, x_p)$ defining feature vector and y its associated label.
- Learn a parametric function f_θ with $\theta \in \mathbb{R}^m$.



$\sigma(x) = h_1$

$\sigma(h_1) = h_2$

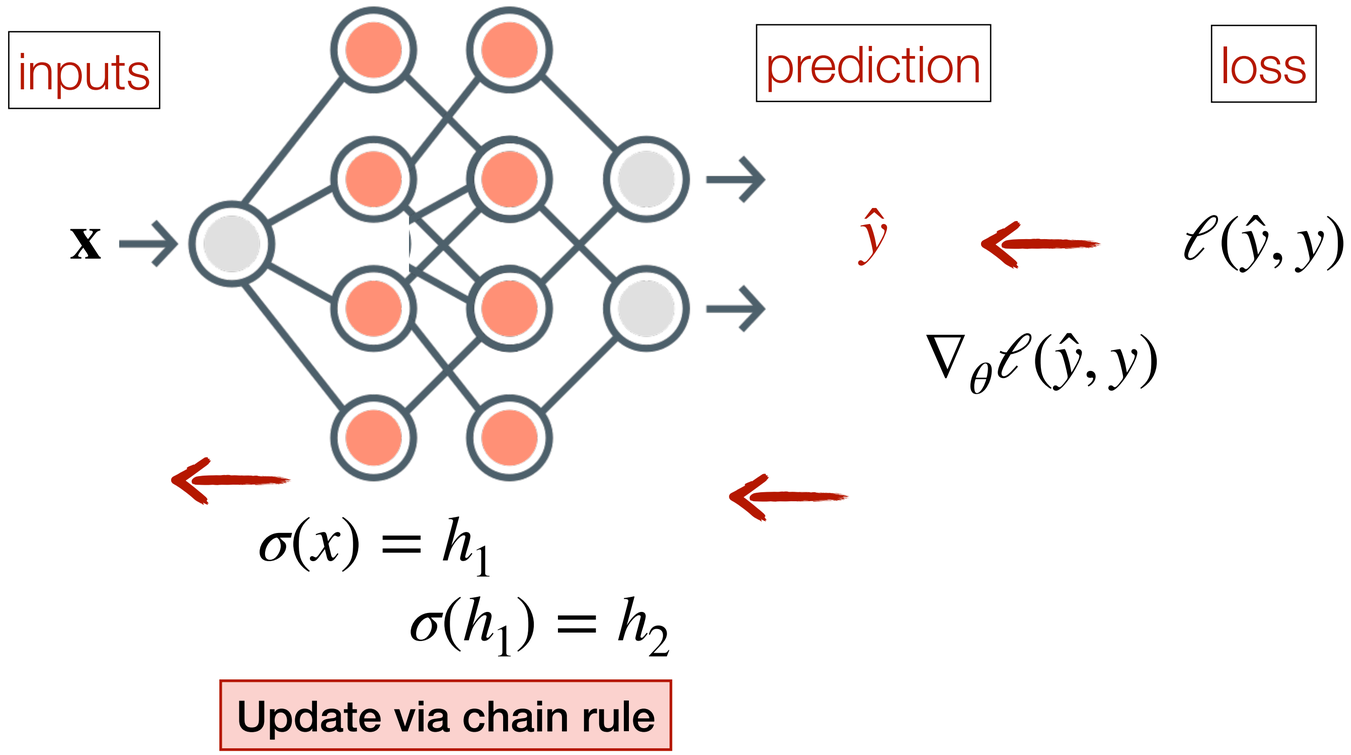
non-linear transforms

Supervised learning

Preliminaries

- Training dataset $D = (\mathbf{x}_i, y_i)_{i=1}^n$ with each $\mathbf{x} = (x_1, \ldots, x_p)$ defining feature vector and y its associated label.
- Learn a parametric function f_θ with $\theta \in \mathbb{R}^m$.
- Train by updating θ in the opposite direction of the gradients of the loss.
- Empirical Risk function:

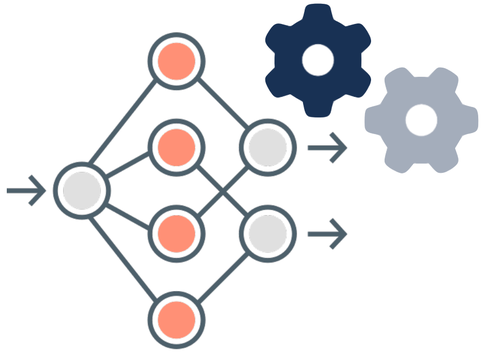
$$\min_\theta \frac{1}{n} \sum_i \ell(f_\theta(\boldsymbol{x}), y)$$



Agenda

Intro and Motivations

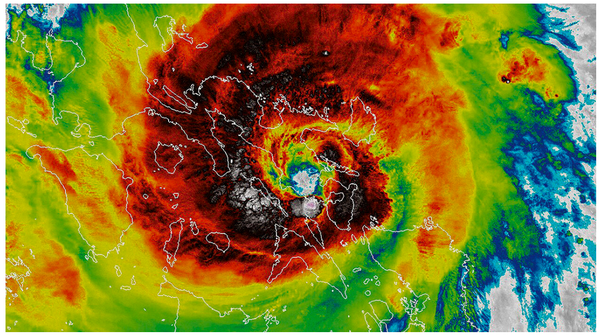
ML for Optimization

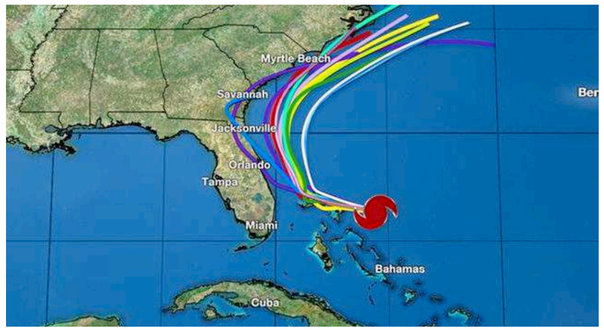


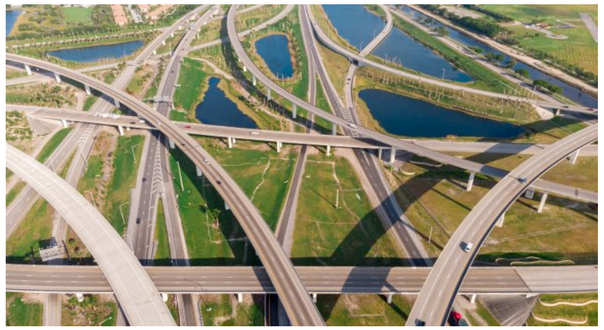
Optimization for ML

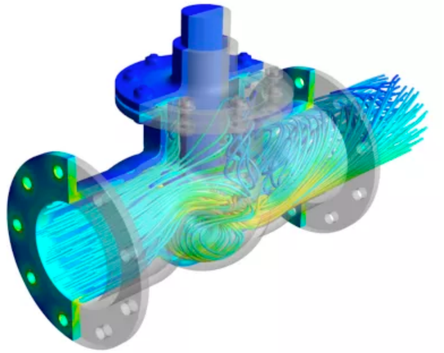
Open Questions

Optimization in Science and Engineering









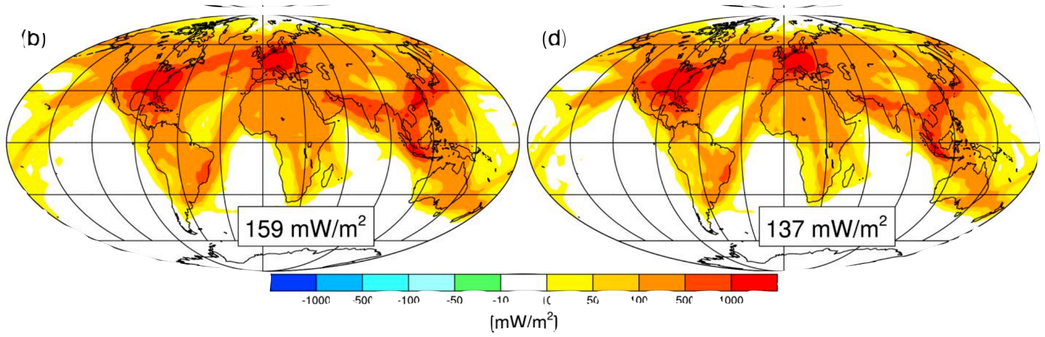
- Optimization has a profound impact on science, engineering, and society.
- The rapid modernization of many scientific and engineering domains is introducing significant challenges that complicate their operations.
- New solutions must be deployed often and be robust to different, uncertain, operating conditions.
- The result is a massive increase in computational resources.
- Force systems operators to produce suboptimal solutions, resulting in significant economical and societal costs.

Constrained optimization in Science and Engineering

The Electrical Power System

Impact

- Revenue: $400B (US, annual)
- Consumption: 4.18T KWh (US, annual)
- Climate impact: 1.87B metric tons (~40% of global) CO2 annual emissions



Optimization in Power Systems

- Benefits from fast and accurate approximations.
- Massive amounts of data are being generated.

Model 1 AC Optimal Power Flow (AC-OPF)

$$\mathcal{O}(\dot{\boldsymbol{p}}^d, \dot{\boldsymbol{q}}^d) = \mathbf{argmin}_{\boldsymbol{p}^g, \boldsymbol{v}} \quad \sum_{i \in \mathcal{N}} \text{cost}(p_i^g)$$

subject to:

$$\dot{v}_i^{\min} \leqslant v_i \leqslant \dot{v}_i^{\max} \qquad \forall i \in \mathcal{N}$$

$$-\dot{\theta}_{ij}^{\Delta} \leqslant \theta_i - \theta_j \leqslant \dot{\theta}_{ij}^{\Delta} \qquad \forall (ij) \in \mathcal{E}$$

$$\dot{p}_i^{g\,\min} \leqslant p_i^g \leqslant \dot{p}_i^{g\,\max} \qquad \forall i \in \mathcal{N}$$

$$\dot{q}_i^{g\,\min} \leqslant q_i^g \leqslant \dot{q}_i^{g\,\max} \qquad \forall i \in \mathcal{N}$$

$$(p_{ij}^f)^2 + (q_{ij}^f)^2 \leqslant \dot{S}_{ij}^{f\,\max} \qquad \forall (ij) \in \mathcal{E}$$

$$p_{ij}^f = \dot{g}_{ij} v_i^2 - v_i v_j (\dot{b}_{ij} \sin(\theta_i - \theta_j) + \dot{g}_{ij} \cos(\theta_i - \theta_j)) \qquad \forall (ij) \in \mathcal{E}$$

$$q_{ij}^f = -\dot{b}_{ij} v_i^2 - v_i v_j (\dot{g}_{ij} \sin(\theta_i - \theta_j) - \dot{b}_{ij} \cos(\theta_i - \theta_j)) \qquad \forall (ij) \in \mathcal{E}$$

$$p_i^g - \dot{p}_i^d = \sum_{(ij) \in \mathcal{E}} p_{ij}^f \qquad \forall i \in \mathcal{N}$$

$$q_i^g - \dot{q}_i^d = \sum_{(ij) \in \mathcal{E}} q_{ij}^f \qquad \forall i \in \mathcal{N}$$

output: $(\boldsymbol{p}^g, \boldsymbol{v})$ – The system operational parameters

Can we use ML to build better approximations?

Learning proxy optimizers

Learning to solve optimization problems

- We are interested in solving problems of the following form:

$$\mathcal{O}(d) = z^* = \arg\min_z \; f(z, d)$$
$$\text{subject to} \;\; g_i(z, d) \geq 0, \;\; \forall i \in [m]$$
$$h_i(z, d) = 0, \;\; \forall i \in [p]$$

- where inequality $g_i \leq 0$ and equality $h_i = 0$ constraints are arbitrary but continuous and differentiable.

Learning proxy optimizers

Key idea

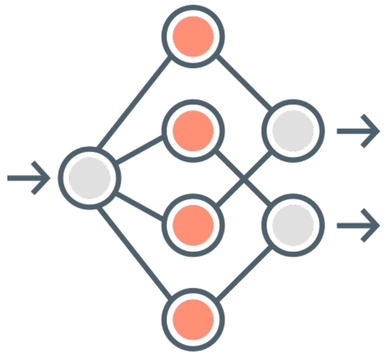
$$\mathcal{O}(d) : z^* = \arg\min_z \; f(z, d)$$

$$\text{s.t.} \;\; g(z, d) \leq 0,$$

$$h(z, d) = 0$$

Data

$$D = \{(d, \mathcal{O}(d))\}$$



$\tilde{z}^*$

Approximate Optimizer

$\tilde{\mathcal{O}}$

Learning proxy optimizers

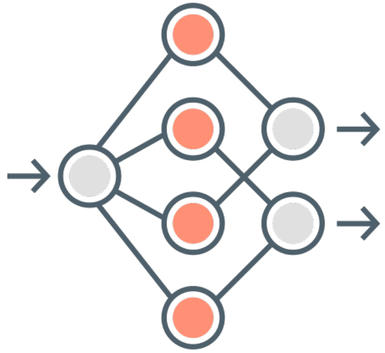
Key idea

$$\mathcal{O}(d) : z^* = \arg\min_z \; f(z, d)$$

$$\text{s.t.} \;\; g(z, d) \leq 0,$$
$$h(z, d) = 0$$

Data

$$D = \{(d, \mathcal{O}(d))\}$$



$\tilde{z}^*$

Approximate Optimizer

$$\tilde{\mathcal{O}} : f_{\theta^*}$$

$$\theta^* = \arg\min_\theta \; \sum_{i=1}^{n} \ell(f_\theta(d_i), z_i^*)$$

$$\text{s.t.} \;\; g(f_\theta(d_i), d_i) \leq 0$$
$$h(f_\theta(d_i), d_i) = 0$$

Learning proxy optimizers

Key idea

$$\mathcal{O}(d) : z^* = \arg\min_z \; f(z, d)$$

$$\text{s.t.} \quad g(z, d) \leq 0,$$
$$h(z, d) = 0$$

Data

$$D = \{(d, \mathcal{O}(d))\}$$

$$\theta^* = \arg\min_\theta \sum_{i=1}^{n} \ell(f_\theta(d_i), z_i^*)$$

$$\boxed{\begin{aligned} \text{s.t.} \quad & g(f_\theta(d_i), d_i) \leq 0 \\ & h(f_\theta(d_i), d_i) = 0 \end{aligned}}$$

challenge

$\tilde{z}^*$

Approximate Optimizer

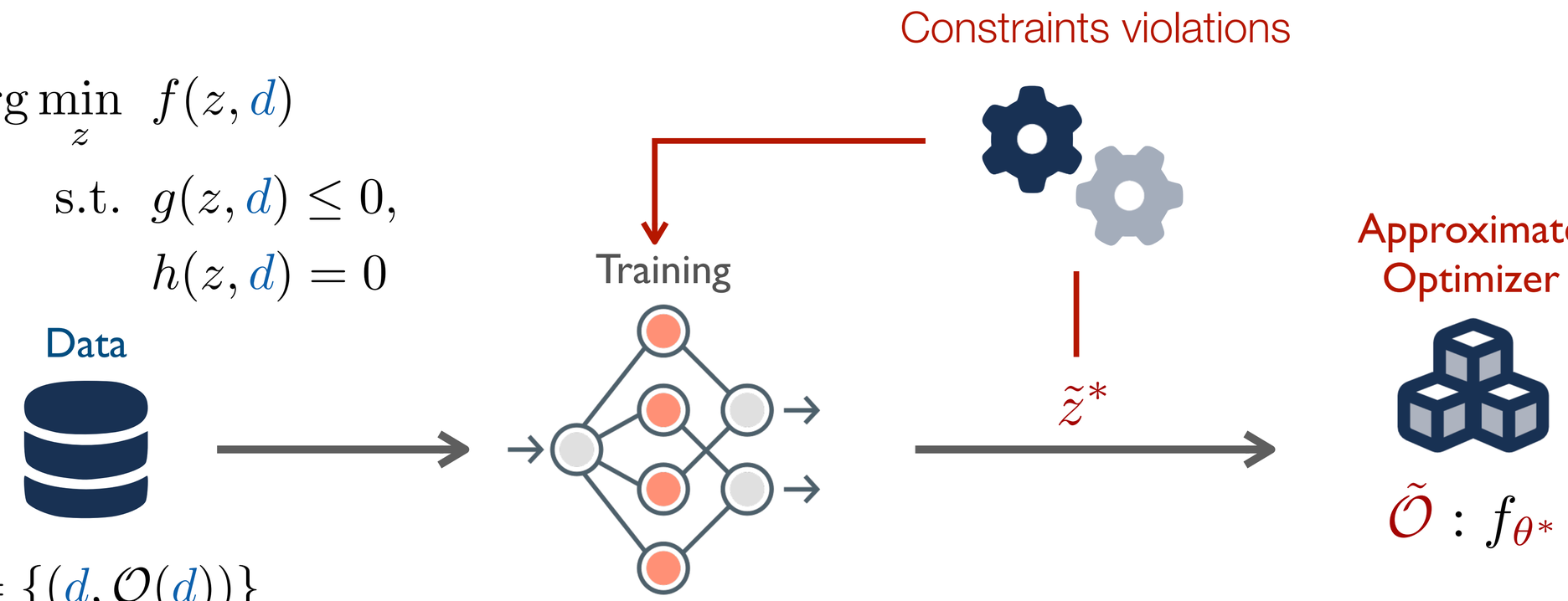
$$\tilde{\mathcal{O}} : f_{\theta^*}$$

Learning proxy optimizers

Key idea

$$\mathcal{O}(d) : z^* = \arg\min_z \; f(z, d)$$

$$\text{s.t.} \;\; g(z, d) \leq 0,$$
$$h(z, d) = 0$$



$$D = \{(d, \mathcal{O}(d))\}$$

$$\theta^* = \arg\min_\theta \; \sum_{i=1}^{n} \ell(f_\theta(d_i), z_i^*)$$

$$\boxed{\begin{aligned} \text{s.t.} \;\; & g(f_\theta(d_i), d_i) \leq 0 \\ & h(f_\theta(d_i), d_i) = 0 \end{aligned}}$$

challenge

Lagrangian Duality

A brief review

- Consider a problem of the type (only inequality constraints to ease presentation)

$$\mathcal{O} = \underset{z}{\text{argmin}} \;\; f(z) \;\; \text{subject to} \;\; g(z) \leq 0$$

Lagrangian Duality

A brief review

- Consider a problem of the type (only inequality constraints to ease presentation)

$$\mathcal{O} = \underset{z}{\operatorname{argmin}} \;\; f(z) \;\; \text{subject to} \;\; g(z) \leq 0$$

- The associated Lagrangian dual function is:

$$f_\lambda(z) = f(z) + \lambda g(z)$$

Violation-based Lagrangian function:

$$f_\lambda(z) = f(z) + \lambda \max(0, g(z))$$

The penalty terms are always non-negative: captures a quantification of the constraint violations

Lagrangian Duality

A brief review

- Consider a problem of the type (only inequality constraints to ease presentation)

$$\mathcal{O} = \underset{z}{\text{argmin}} \;\; f(z) \;\; \text{subject to} \;\; g(z) \leq 0$$

- The associated Lagrangian dual function is:

$$f_\lambda(z) = f(z) + \lambda g(z)$$

Violation-based Lagrangian function:

$$f_\lambda(z) = f(z) + \lambda \max(0, g(z))$$

The penalty terms are always non-negative: captures a quantification of the constraint violations

- The optimization problem becomes:

$$LR_\lambda = \underset{z}{\text{argmin}}\, f_\lambda(z)$$

Lagrangian Duality

A brief review

- Consider a problem of the type (only inequality constraints to ease presentation)

$$\mathcal{O} = \underset{z}{\text{argmin}} \;\; f(z) \;\; \text{subject to} \;\; g(z) \leq 0$$

- The associated Lagrangian dual function is:

$$f_\lambda(z) = f(z) + \lambda g(z)$$

Violation-based Lagrangian function:

$$f_\lambda(z) = f(z) + \lambda \max(0, g(z))$$

The penalty terms are always non-negative: captures a quantification of the constraint violations

- The optimization problem becomes:

$$LR_\lambda = \underset{z}{\text{argmin}}\, f_\lambda(z)$$

- The strongest Lagrangian relaxation of O is found by $LD = \underset{\lambda \geqslant 0}{\text{argmax}}\, f(LR_\lambda).$

Learning proxy optimizers

With Lagrangian duality

- Exploit the Lagrangian Dual approach in the learning task to approximate minimizer $\mathcal{O}$
- Given multipliers λ the Lagrangian loss function is defined as

$$\ell_\lambda(\cdot) = \ell\Big(f_\theta(d), z_i^*\Big) + \lambda\,\nu\Big(g(f_\theta(d), d) \leq 0\Big)$$

Learning proxy optimizers

With Lagrangian duality

- Exploit the Lagrangian Dual approach in the learning task to approximate minimizer $\mathcal{O}$
- Given multipliers λ the Lagrangian loss function is defined as

$$\ell_\lambda(\cdot) = \ell\Big(f_\theta(d), z_i^*\Big) + \lambda \boxed{\nu\Big(g(f_\theta(d), d) \leq 0\Big)} \longrightarrow \text{Constraint violation}$$

Fioretto et al.,AAAI:20, ECAI:20

Learning proxy optimizers

With Lagrangian duality

- Exploit the Lagrangian Dual approach in the learning task to approximate minimizer $\mathcal{O}$
- Given multipliers λ the Lagrangian loss function is defined as

$$\ell_\lambda(\cdot) = \ell\Big(f_\theta(d), z_i^*\Big) + \lambda \boxed{\nu\Big(g(f_\theta(d), d) \leq 0\Big)} \longrightarrow \text{Constraint violation}$$

- To produce an approximation $\tilde{O} = f_{\theta^*(\lambda)}$ of O for fixed multipliers λ, we solve

$$\theta^*(\lambda) = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{n} \ell_\lambda\Big(f_\theta(d_i), z_i^*\Big)$$

Fioretto et al.,AAAI:20, ECAI:20

Learning proxy optimizers

With Lagrangian duality

- Exploit the Lagrangian Dual approach in the learning task to approximate minimizer $\mathcal{O}$
- Given multipliers λ the Lagrangian loss function is defined as

$$\ell_\lambda(\cdot) = \ell\Big(f_\theta(d), z_i^*\Big) + \lambda \boxed{\nu\Big(g(f_\theta(d), d) \leq 0\Big)} \longrightarrow \text{Constraint violation}$$

- To produce an approximation $\tilde{O} = f_{\theta^*(\lambda)}$ of O for fixed multipliers λ, we solve

$$\theta^*(\lambda) = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{n} \ell_\lambda\Big(f_\theta(d_i), z_i^*\Big)$$

- To obtain $\tilde{O} = f_{\theta^*(\lambda^*)}$ (the strongest Lagrangian relaxation of O), the Lagrangian dual computes the optimal multipliers:

$$\lambda^* = \underset{\lambda}{\operatorname{argmax}} \min_{\theta} \sum_{i=1}^{n} \ell_\lambda\Big(f_\theta(d_i), z_i^*\Big)$$

Fioretto et al.,AAAI:20, ECAI:20

The Learning Step

- Use an iterative approach to find good values of primal and dual variables

$$\theta^{k+1} = \underset{\theta}{\operatorname{argmin}}\, \mathbb{E}_{(d,z^*)\sim D} \left[\ell(f_\theta^{\lambda^k}(d), z^*) \right]$$

Algorithm 1: LDF for Constrained Optimization Problems

input: $D = (d_l, z_l^*)_{l=1}^n$: Training data;
$\alpha, s = (s_0, s_1, \ldots)$: Optimizer and Lagrangian step sizes

$\lambda_i^0 \leftarrow 0 \;\; \forall i \in [m]$
for *epoch* $k = 0, 1, \ldots$ **do**
  foreach $(d_l, z_l^*) \in D$ **do**
    $\hat{z}_l \leftarrow f_\theta^{\lambda_k}(d_l)$
    $\theta(\lambda^k) \leftarrow \theta(\lambda^k) - \alpha \nabla \ell(\hat{z}_l, z_l^*)$
  $\lambda_i^{k+1} \leftarrow \lambda_i^k + \rho \sum_{l=1}^n \nu_i(g_i(\hat{z}_l)) \;\; \forall i \in [m]]$

The Learning Step

- Use an iterative approach to find good values of primal and dual variables

$$\theta^{k+1} = \underset{\theta}{\operatorname{argmin}}\, \mathbb{E}_{(d,z^*)\sim D} \left[\ell(f_\theta^{\lambda^k}(d), z^*) \right]$$

Algorithm 1: LDF for Constrained Optimization Problems

input: $D = (d_l, z_l^*)_{l=1}^n$: Training data;
$\alpha, s = (s_0, s_1, \ldots)$: Optimizer and Lagrangian step sizes

$\lambda_i^0 \leftarrow 0 \;\; \forall i \in [m]$
for *epoch* $k = 0, 1, \ldots$ **do**
  foreach $(d_l, z_l^*) \in D$ **do**
    $\hat{z}_l \leftarrow f_\theta^{\lambda_k}(d_l)$
    $\theta(\lambda^k) \leftarrow \theta(\lambda^k) - \alpha \nabla \ell(\hat{z}_l, z_l^*)$
  $\lambda_i^{k+1} \leftarrow \lambda_i^k + \rho \sum_{l=1}^n \nu_i(g_i(\hat{z}_l)) \;\; \forall i \in [m]]$

Fioretto et al.,AAAI:20, ECAI:20

The Learning Step

- Use an iterative approach to find good values of primal and dual variables

$$\theta^{k+1} = \underset{\theta}{\operatorname{argmin}}\, \mathbb{E}_{(d,z^*)\sim D} \left[\ell(f_\theta^{\lambda^k}(d), z^*)\right]$$

$$\lambda^{k+1} = \left(\lambda_c^k + \rho \nu_c(d, f_{\theta^{k+1}}^{\lambda^k}(d)) \mid c \in \mathcal{C}\right)$$

Constraint violation

e.g., max(0, g(x)) or |h(x)|

Algorithm 1: LDF for Constrained Optimization Problems

input: $D = (d_l, z_l^*)_{l=1}^n$: Training data;
$\alpha, s = (s_0, s_1, \ldots)$: Optimizer and Lagrangian step sizes

$\lambda_i^0 \leftarrow 0 \;\; \forall i \in [m]$
for *epoch* $k = 0, 1, \ldots$ **do**
  foreach $(d_l, z_l^*) \in D$ **do**
    $\hat{z}_l \leftarrow f_\theta^{\lambda_k}(d_l)$
    $\theta(\lambda^k) \leftarrow \theta(\lambda^k) - \alpha \nabla \ell(\hat{z}_l, z_l^*)$
  $\lambda_i^{k+1} \leftarrow \lambda_i^k + \rho \sum_{l=1}^n \nu_i(g_i(\hat{z}_l)) \;\; \forall i \in [m]]$

The Learning Step

- Use an iterative approach to find good values of primal and dual variables

$$\theta^{k+1} = \underset{\theta}{\operatorname{argmin}}\, \mathbb{E}_{(d,z^*)\sim D} \left[\ell(f_\theta^{\lambda^k}(d), z^*)\right]$$

$$\lambda^{k+1} = \left(\lambda_c^k + \rho \nu_c(d, f_{\theta^{k+1}}^{\lambda^k}(d)) \mid c \in \mathcal{C}\right)$$

Constraint violation

e.g., max(0, g(x)) or |h(x)|

Algorithm 1: LDF for Constrained Optimization Problems

input: $D = (d_l, z_l^*)_{l=1}^n$: Training data;
$\alpha, s = (s_0, s_1, \ldots)$: Optimizer and Lagrangian step sizes

$\lambda_i^0 \leftarrow 0 \;\; \forall i \in [m]$
for *epoch* $k = 0, 1, \ldots$ **do**
  foreach $(d_l, z_l^*) \in D$ **do**
    $\hat{z}_l \leftarrow f_\theta^{\lambda_k}(d_l)$
    $\theta(\lambda^k) \leftarrow \theta(\lambda^k) - \alpha \nabla \ell(\hat{z}_l, z_l^*)$
  $\lambda_i^{k+1} \leftarrow \lambda_i^k + \rho \sum_{l=1}^n \nu_i(g_i(\hat{z}_l)) \;\; \forall i \in [m]]$

- The "primal" step is approximated using a Stochastic Gradient method.

- This step does not recompute the primal variables from scratch! It uses a warm-start.

How does it works in Practice?

AC Optimal Power Flow Predictions

Solution Quality

Test Case		Dist. to load flow sol. (%) DC	$\mathcal{M}^-$	$\mathcal{M}$
30_ieee	p^g	2.6972	2.0793	0.0007
	v	1.2929	83.138	0.0037
118_ieee	p^g	0.2011	0.1071	0.0038
	v	1.9971	3.4391	0.0866
300_ieee	p^g	0.1336	0.0447	0.0084
	v	3.8526	31.698	0.1994
	p^g	0.7751	0.9843	0.0197
	v	2.4284	36.288	0.1995
		Total Average		

Objective cost distance and runtime

Test Case	Dist. to AC-OPF cost (%) DC	$\mathcal{M}^-$	$\mathcal{M}$	Runtime (sec.) AC	DC	$\mathcal{M}$
30_ieee	7.9894	2.9447	0.0180	0.1024	0.0148	$< 10^4$
118_ieee	4.7455	1.0973	0.5408	0.4207	0.0785	0.0001
300_ieee	4.7508	1.9543	0.3011	8.0645	0.2662	0.0001
	4.5733	2.3706	0.2124	1x	30.3x	$>10^4$x
	Total Average			Min Speedup		

M: constrained-aware ML

DC: linear approximation (as used in industry)

AC: full non-linear model

Fioretto et al.,AAAI:20,TPWRS:21

How does it works in Practice?

AC Optimal Power Flow Predictions

Solution Quality

Test Case		Dist. to load flow sol. (%) DC	$\mathcal{M}^-$	$\mathcal{M}$
30_ieee	p^g	2.6972	2.0793	0.0007
	v	1.2929	83.138	0.0037
118_ieee	p^g	0.2011	0.1071	0.0038
	v	1.9971	3.4391	0.0866
300_ieee	p^g	0.1336	0.0447	0.0084
	v	3.8526	31.698	0.1994
	p^g	0.7751	0.9843	0.0197
	v	2.4284	36.288	0.1995
		Total Average		

Objective cost distance and runtime

Test Case	Dist. to AC-OPF cost (%) DC	$\mathcal{M}^-$	$\mathcal{M}$	Runtime (sec.) AC	DC	$\mathcal{M}$
30_ieee	7.9894	2.9447	0.0180	0.1024	0.0148	$< 10^4$
118_ieee	4.7455	1.0973	0.5408	0.4207	0.0785	0.0001
300_ieee	4.7508	1.9543	0.3011	8.0645	0.2662	0.0001
	4.5733	2.3706	0.2124	1x	30.3x	$>10^4$x
	Total Average			Min Speedup		

M: constrained-aware ML

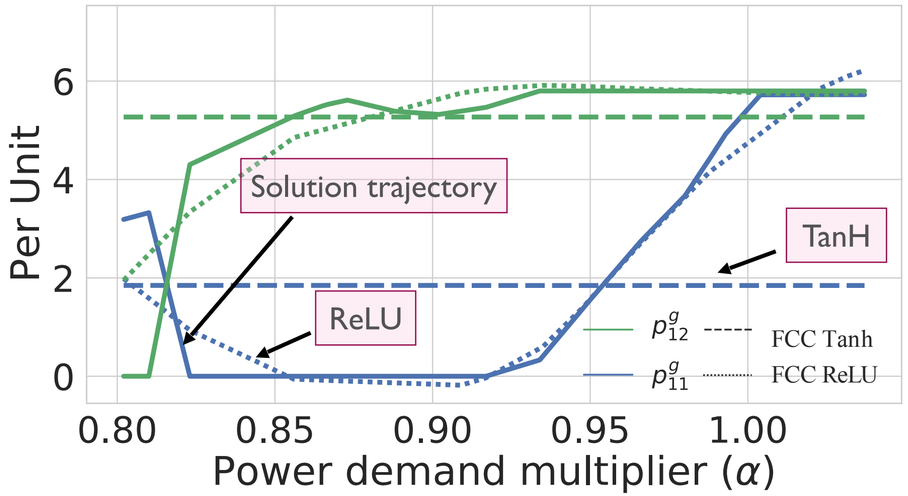
DC: linear approximation (as used in industry)

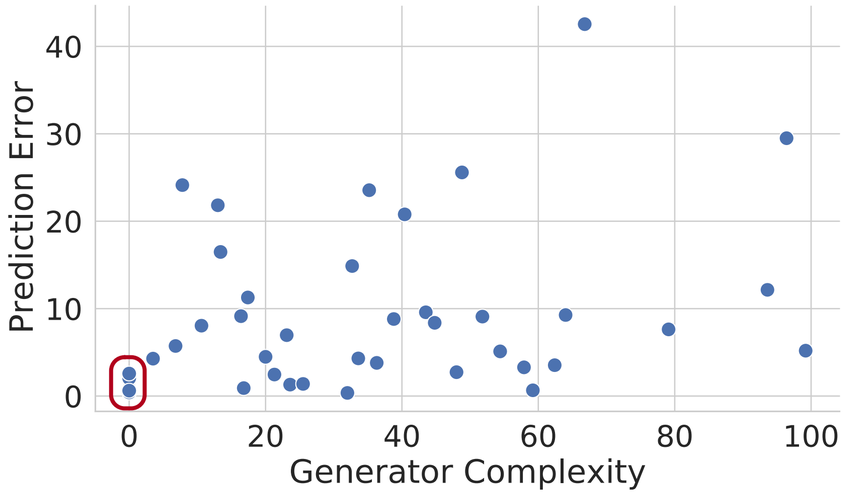
AC: full non-linear model

Summary: Constrained-aware ML can predict quantities several orders of magnitude more accurately and faster than the linear (DC) approximation (and a baseline learning model M^-) and reports significantly fewer constraint violations.

Why does it work?

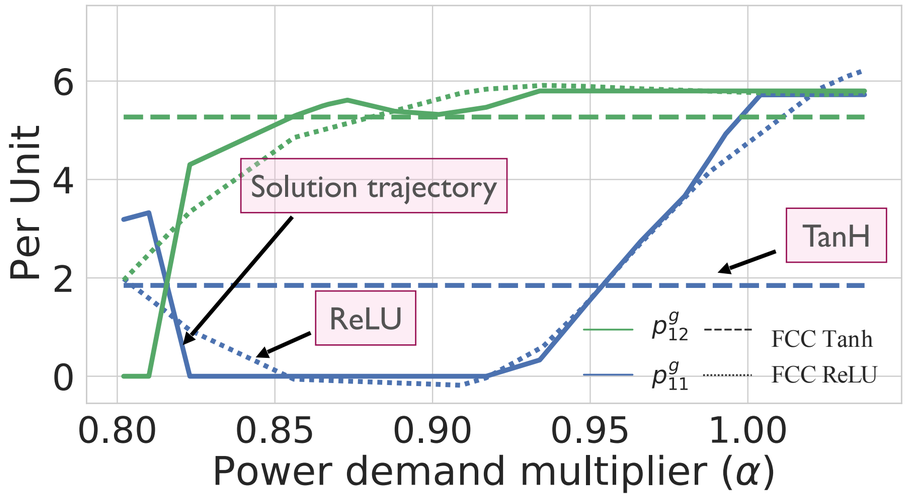
- Solution trajectories can be approximated by piecewise linear functions.
- ReLU neural networks have the ability to capture piecewise linear functions.



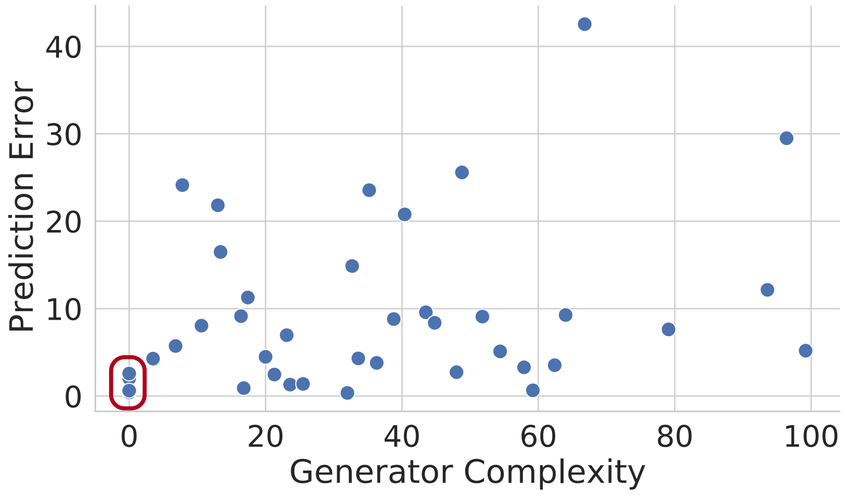


Why does it work?

- Solution trajectories can be approximated by piecewise linear functions.
- ReLU neural networks have the ability to capture piecewise linear functions.



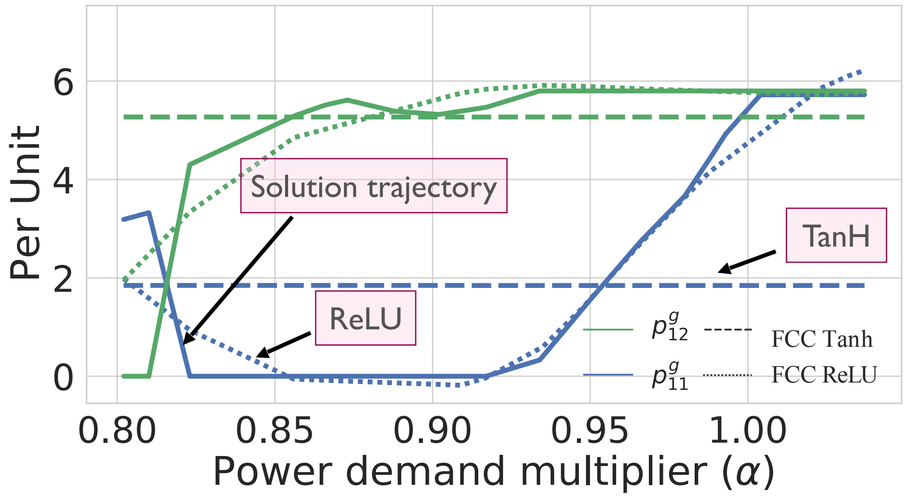
- **Theorem (model capacity).** *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a piecewise linear function with p pieces. If f is represented by a ReLU network with depth $k+1$, then it must have size at least $\frac{1}{2}kp^{\frac{1}{k}} - 1$.*
- **Corollary:** *Conversely, any piecewise linear function f that is represented by a ReLU network of depth $k+1$ and size at most s, can have at most $\left(\frac{2s}{k}\right)^k$ pieces.*



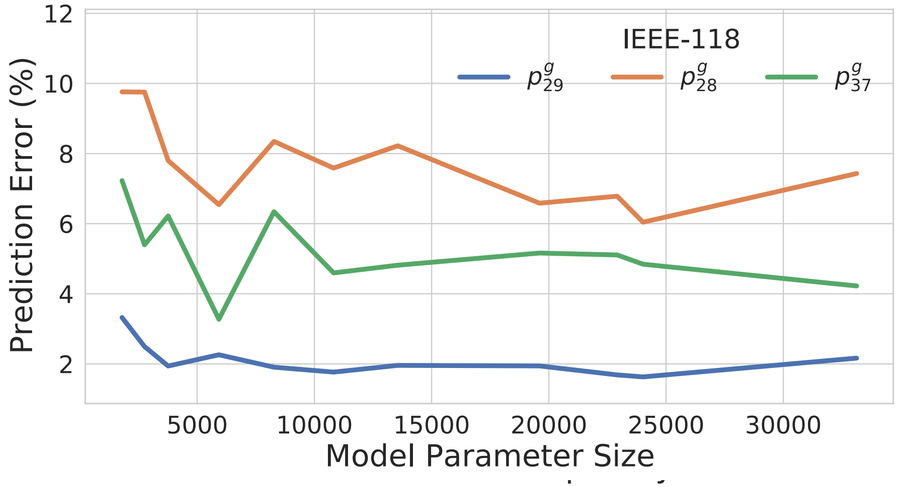
Fioretto et al.,NeurIPS:21

Why does it work?

- Solution trajectories can be approximated by piecewise linear functions.
- ReLU neural networks have the ability to capture piecewise linear functions.



- **Theorem (model capacity).** *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a piecewise linear function with p pieces. If f is represented by a ReLU network with depth $k+1$, then it must have size at least $\frac{1}{2}kp^{\frac{1}{k}} - 1$.*
- **Corollary:** *Conversely, any piecewise linear function f that is represented by a ReLU network of depth $k+1$ and size at most s, can have at most $\left(\frac{2s}{k}\right)^k$ pieces.*

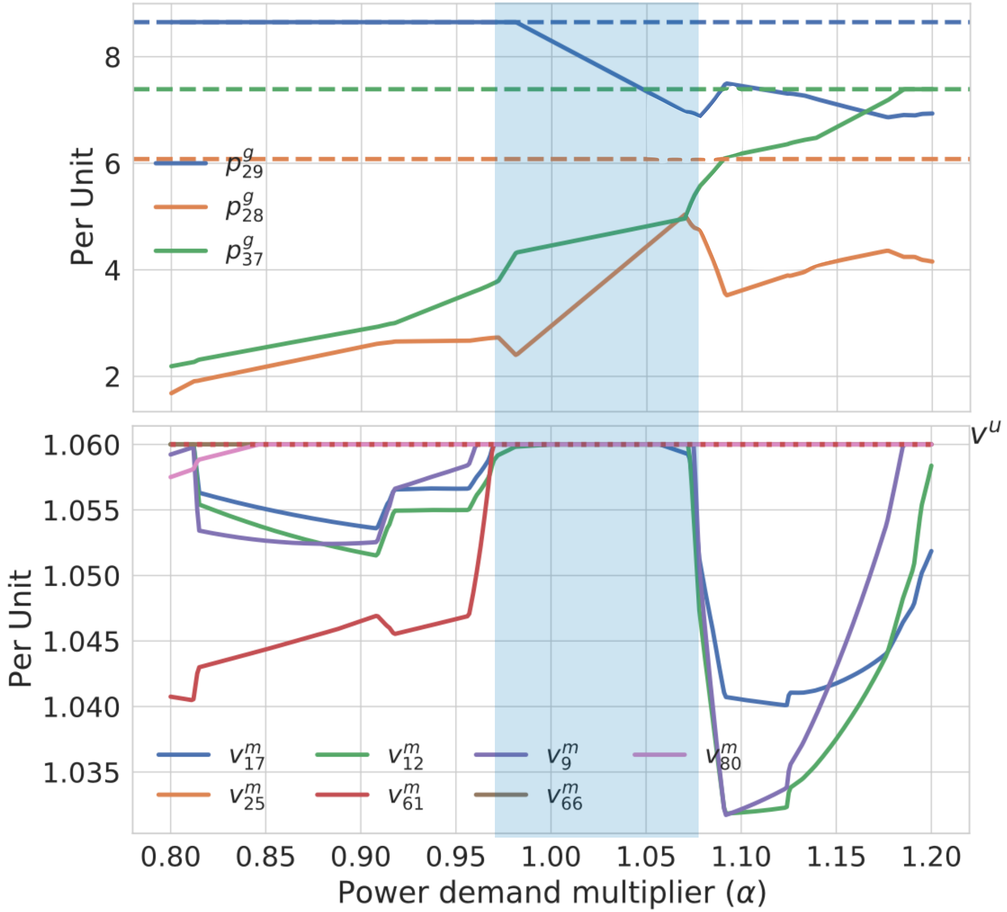


Fioretto et al.,NeurIPS:21

Why does it work?

The importance of modeling constraints

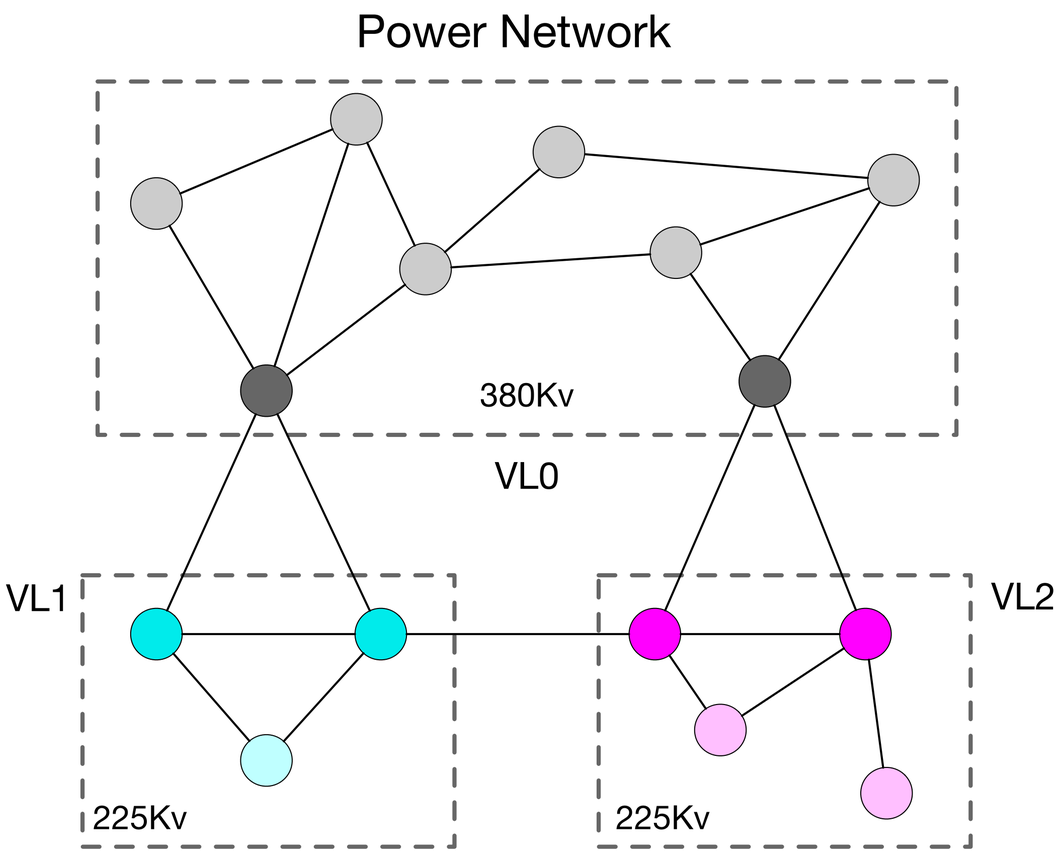
- High-volatile regions are often associated with “problematic” constraints.
- With classical training, the hidden representation of a DNN does not accurately learn the physical constraints regulating the OPF solutions.
- Introducing constraints using Lagrangian-based penalties is not a regularization term.
- It helps the model accurately learn different hidden features, i.e., to more accurately capture the dependencies across variables and their outputs.



Fioretto et al.,NeurIPS:21

Scaling to massive problem sizes

Exploiting the problem structure

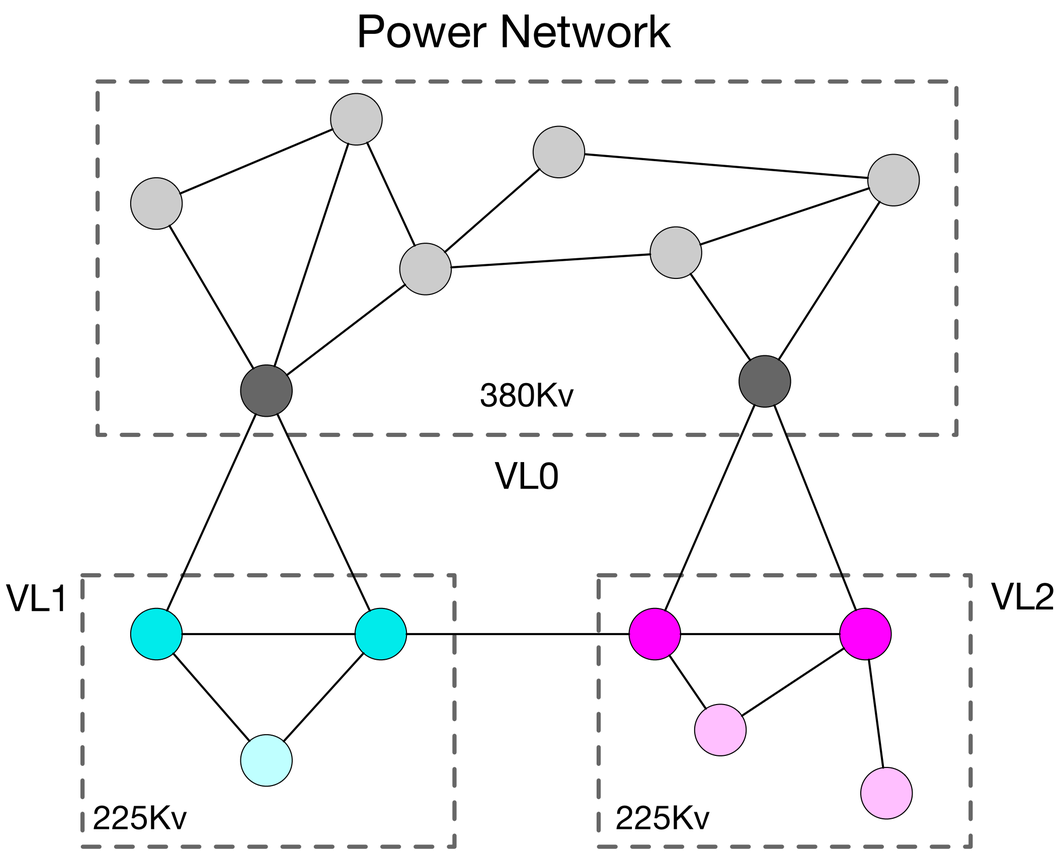


- The architectures adopted to construct proxy optimizers predict decision variables, thus they may have a large number of outputs and may suffer scalability issues.
- Many optimization problems exhibit a rich structure that is traditionally exploited.
- For example, power networks are organized in voltage levels, exposing subproblems that can be exploited.

Fioretto et al.,TPWPS:21

Scaling to massive problem sizes

Lagrangian decomposition



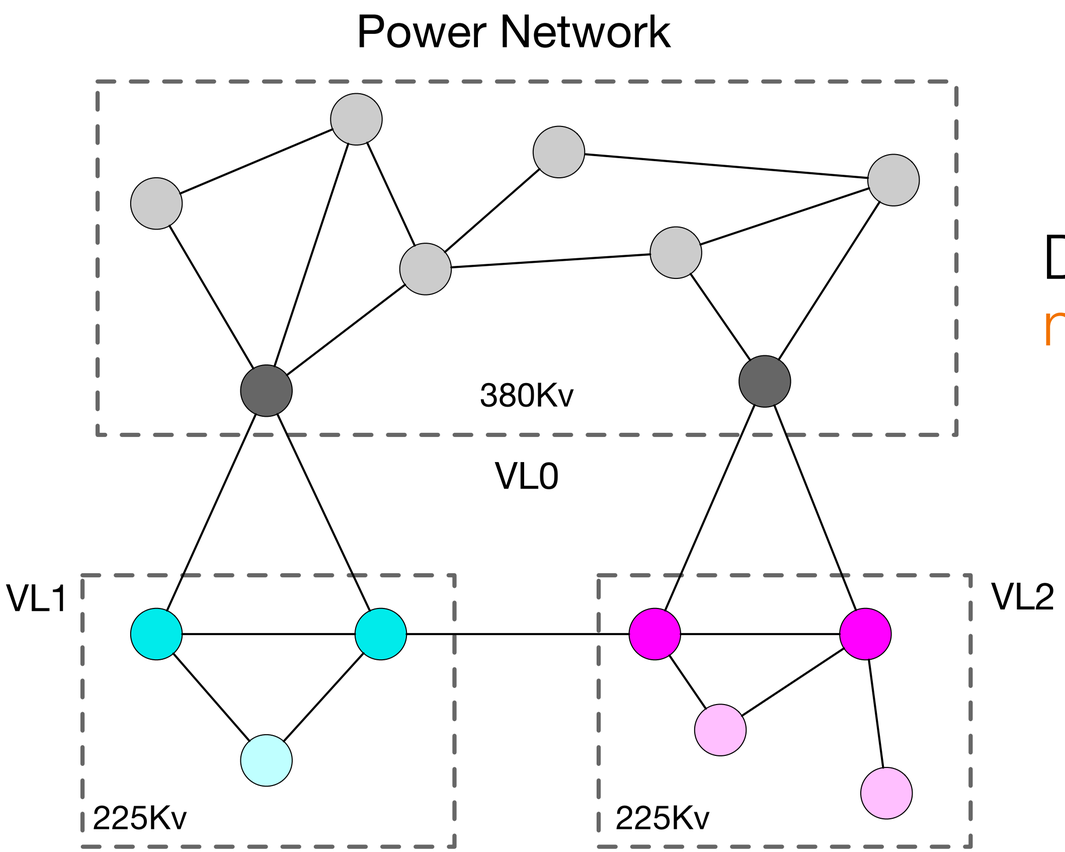
Consider an optimization problem

$$\underset{x}{\text{Minimize}} \quad f(x) \quad \text{subject to:} \quad h_1(x) \geq 0,\ h_2(x) \geq 0,\ x \geq 0$$

Fioretto et al.,TPWPS:21

Scaling to massive problem sizes

Lagrangian decomposition



Consider an optimization problem

$$\underset{x}{\text{Minimize}} \quad f(x) \quad \text{subject to:} \quad h_1(x) \geq 0,\ h_2(x) \geq 0,\ x \geq 0$$

Decoupling constraints through the introduction of new variables and performing a Lagrangian relaxation

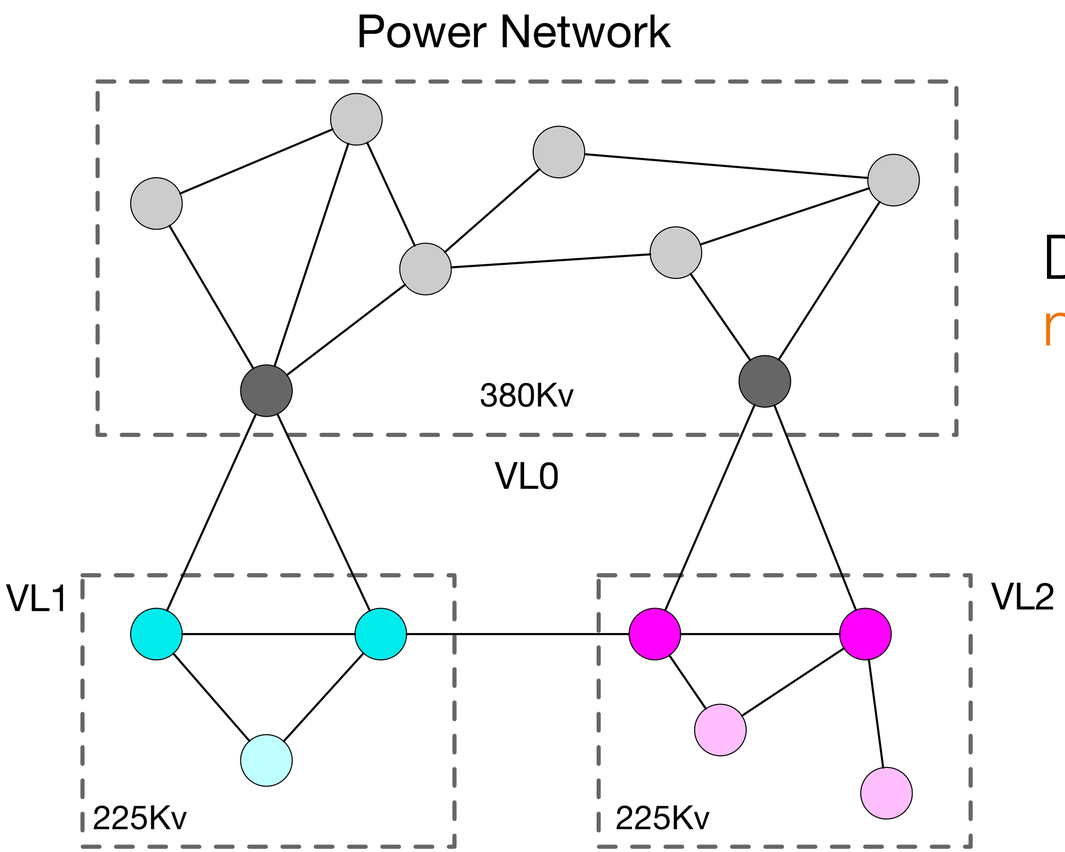
$$\underset{x}{\text{Minimize}} \quad f(x) + \lambda(x' - x)$$

$$\text{subject to:} \quad h_1(x) \geq 0,\ h_2(x') \geq 0,\ x, x' \geq 0$$

Fioretto et al.,TPWPS:21

Scaling to massive problem sizes

Lagrangian decomposition



Consider an optimization problem

$$\underset{x}{\text{Minimize}} \quad f(x) \quad \text{subject to:} \quad h_1(x) \geq 0,\ h_2(x) \geq 0,\ x \geq 0$$

Decoupling constraints through the introduction of new variables and performing a Lagrangian relaxation

$$\underset{x}{\text{Minimize}} \quad f(x) + \lambda(x' - x)$$

$$\text{subject to:} \quad h_1(x) \geq 0,\ h_2(x') \geq 0,\ x, x' \geq 0$$

Which leads to two subproblems,

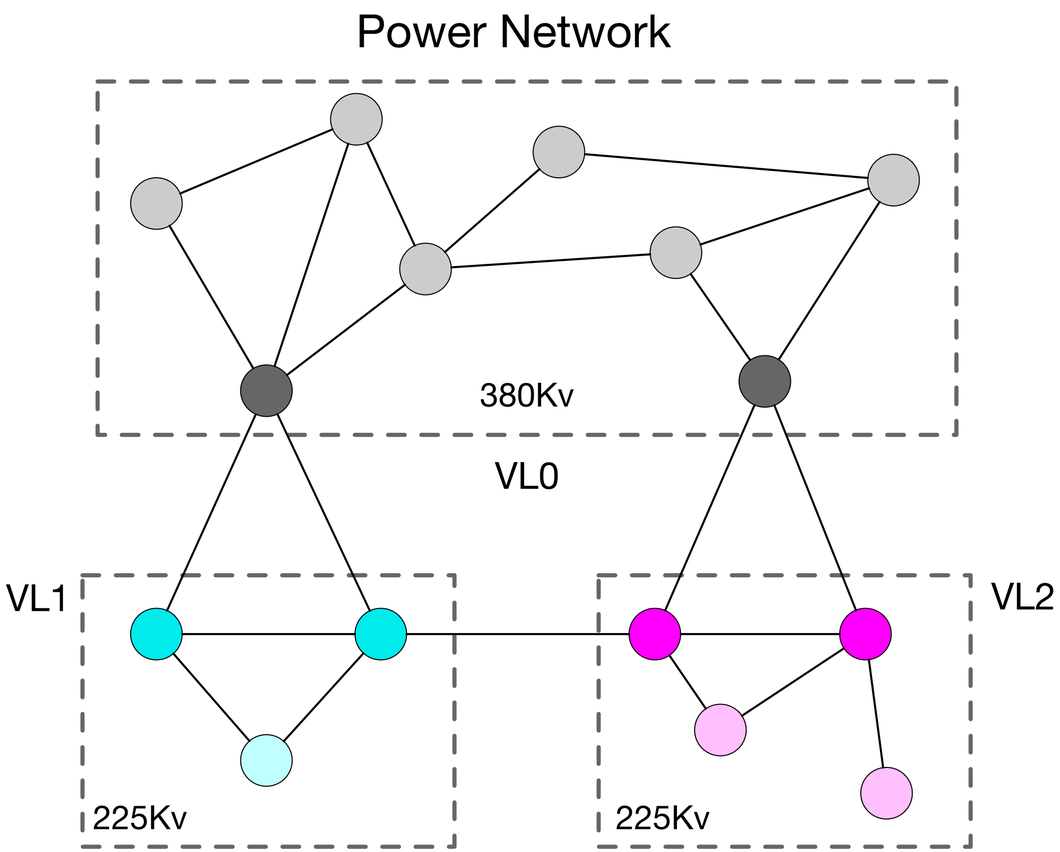
$$\underset{x}{\text{Minimize}} \ f(x) - \lambda x \ \text{ subject to: } h_1(x) \geq 0,\ x \geq 0$$

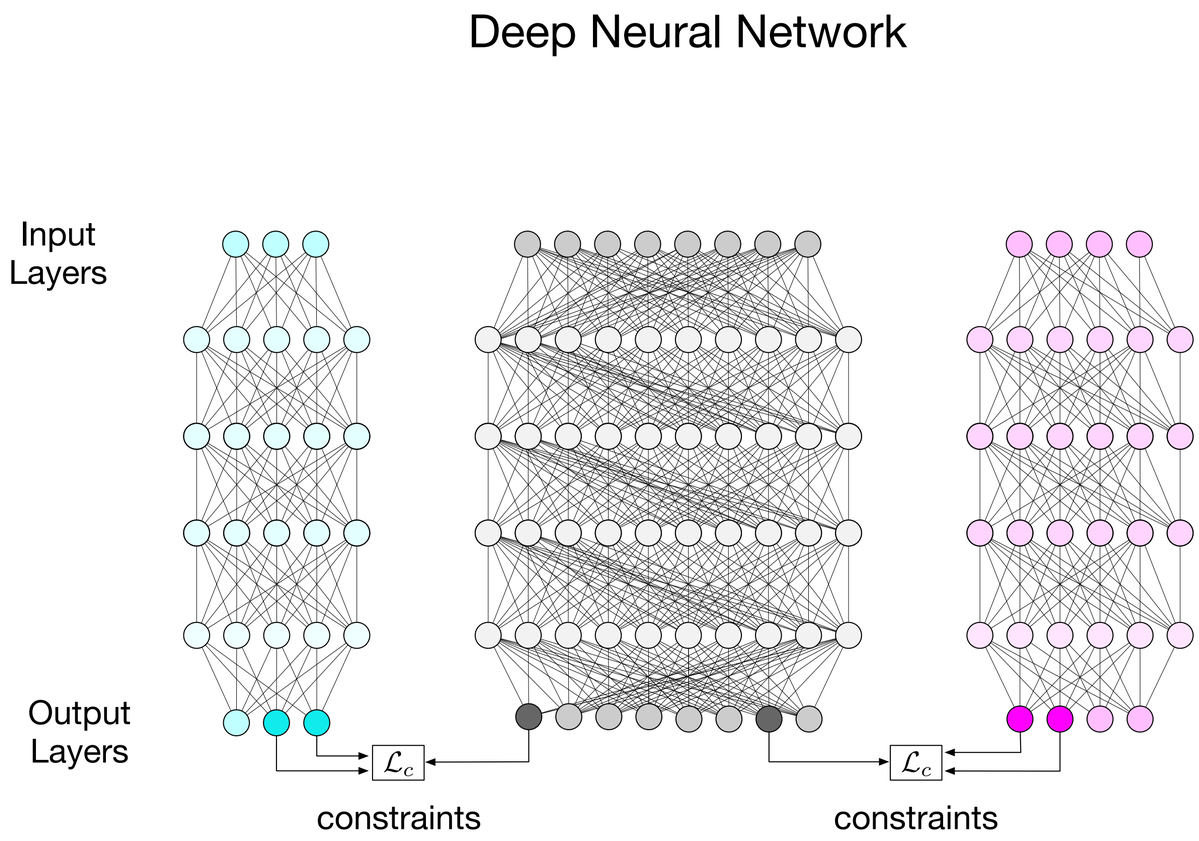
$$\underset{x'}{\text{Minimize}} \ \lambda x' \ \text{ subject to: } h_2(x') \geq 0,\ x' \geq 0.$$

Fioretto et al.,TPWPS:21

Scaling to massive problem sizes

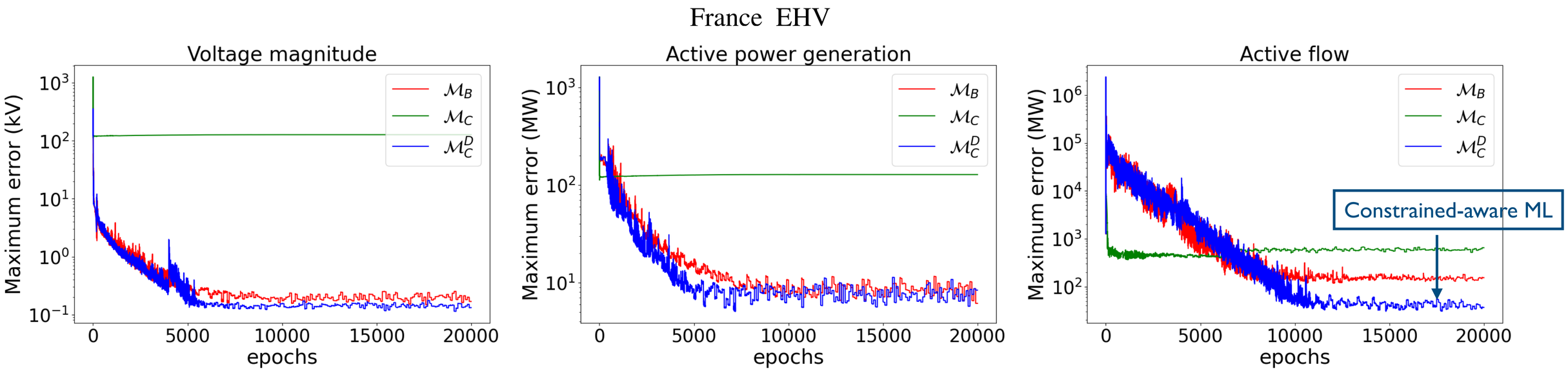
Exploiting the problem structure





How does it work?

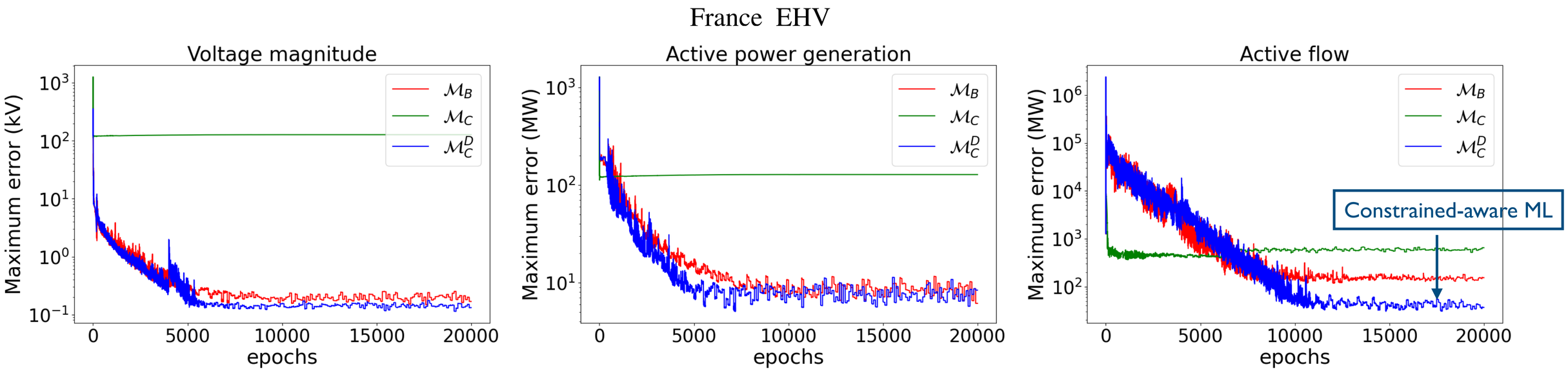
Optimizing country-wise systems



Test Case	$\hat{v}$	$\hat{\theta}$	$\hat{p}^g$	$\hat{q}^g$	Predict time (sec)	Train Mem. (GB)
89_pegase	0.025	0.005	0.585	0.425	0.0013	1.1
118_ieee	0.009	0.004	0.013	0.016	0.0013	1.1
300_ieee	0.006	0.020	0.183	0.068	0.0014	1.2
MSR	0.002	0.001	0.007	0.005	0.0016	1.5
France_EHV	0.012	0.008	0.032	0.104	0.0016	4.6
France_Lyon	0.025	0.005	0.098	0.093	0.0020	13.9

How does it work?

Optimizing country-wise systems

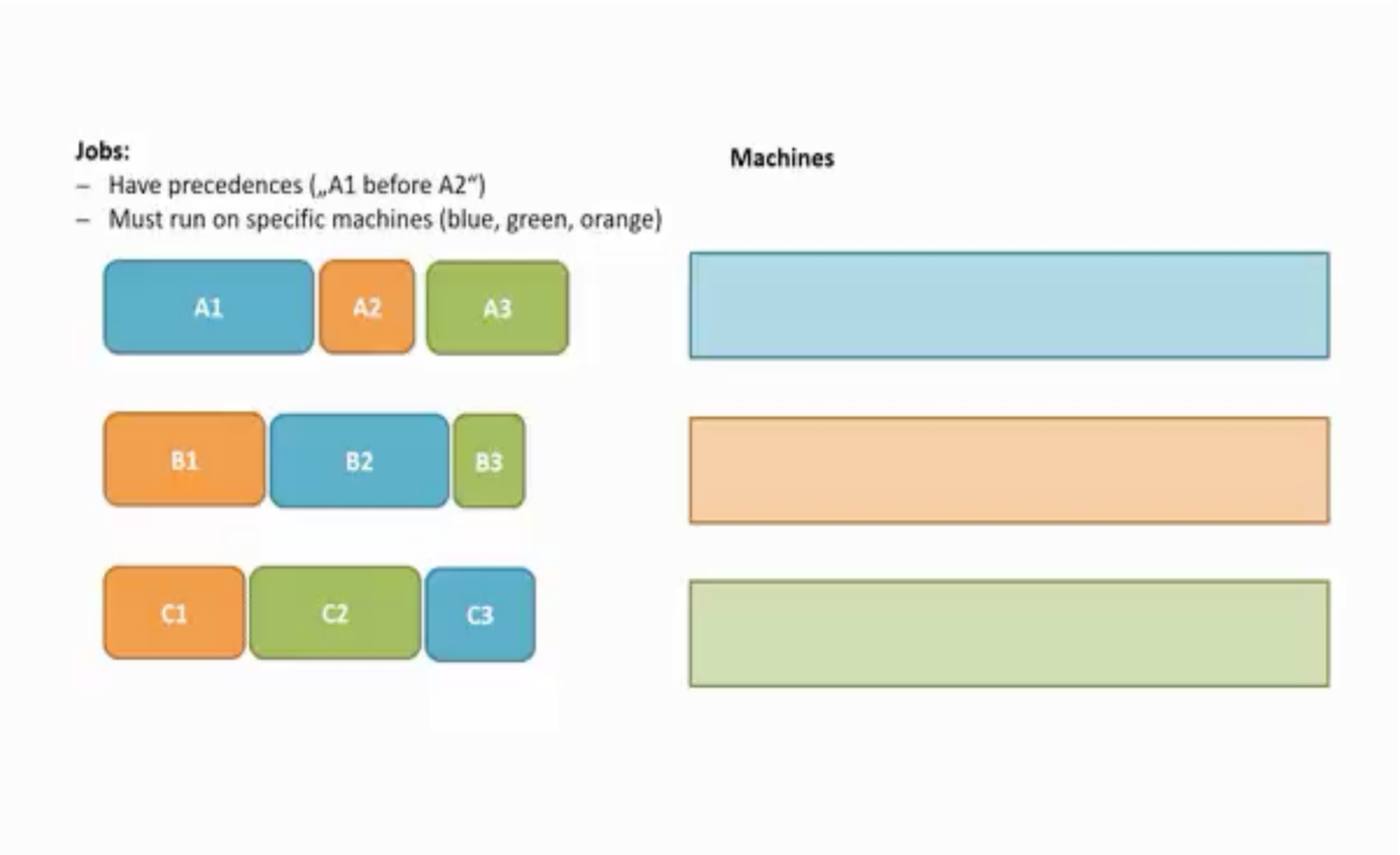


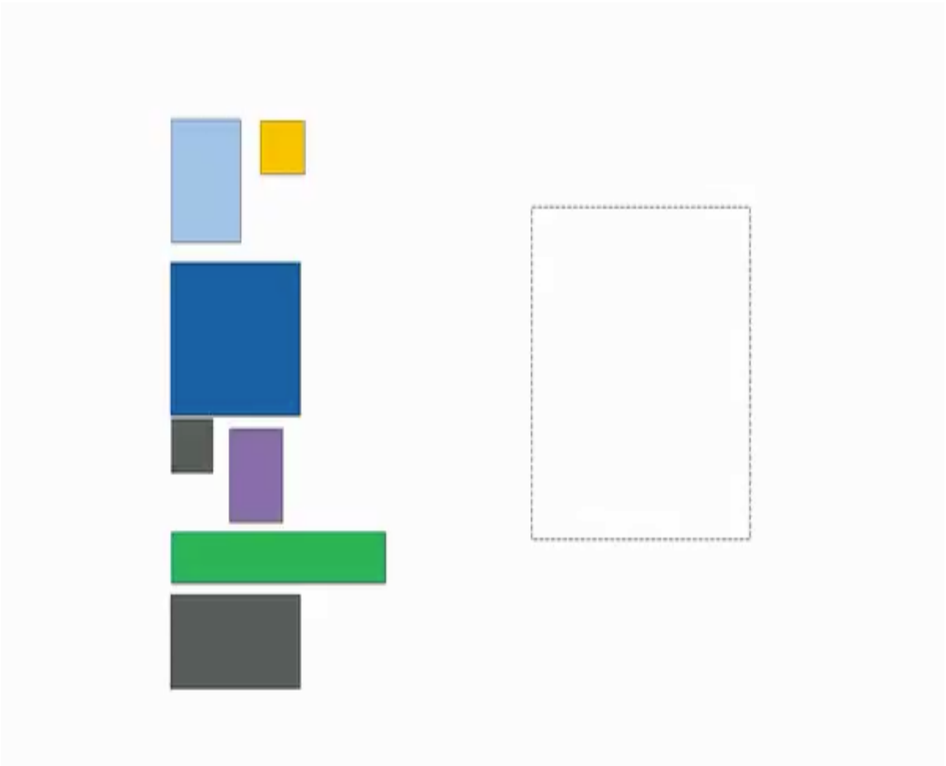
Test Case	$\hat{v}$	$\hat{\theta}$	$\hat{p}^g$	$\hat{q}^g$	Predict time (sec)	Train Mem. (GB)
89_pegase	0.025	0.005	0.585	0.425	0.0013	1.1
118_ieee	0.009	0.004	0.013	0.016	0.0013	1.1
300_ieee	0.006	0.020	0.183	0.068	0.0014	1.2
MSR	0.002	0.001	0.007	0.005	0.0016	1.5
France_EHV	0.012	0.008	0.032	0.104	0.0016	4.6
France_Lyon	0.025	0.005	0.098	0.093	0.0020	13.9

Summary: Constrained-aware ML can predict quantities with unprecedented accuracy and scale to country-wise (**>14,000 variables**) systems.

Discrete solutions and data challenges

Job Shop Scheduling Problems



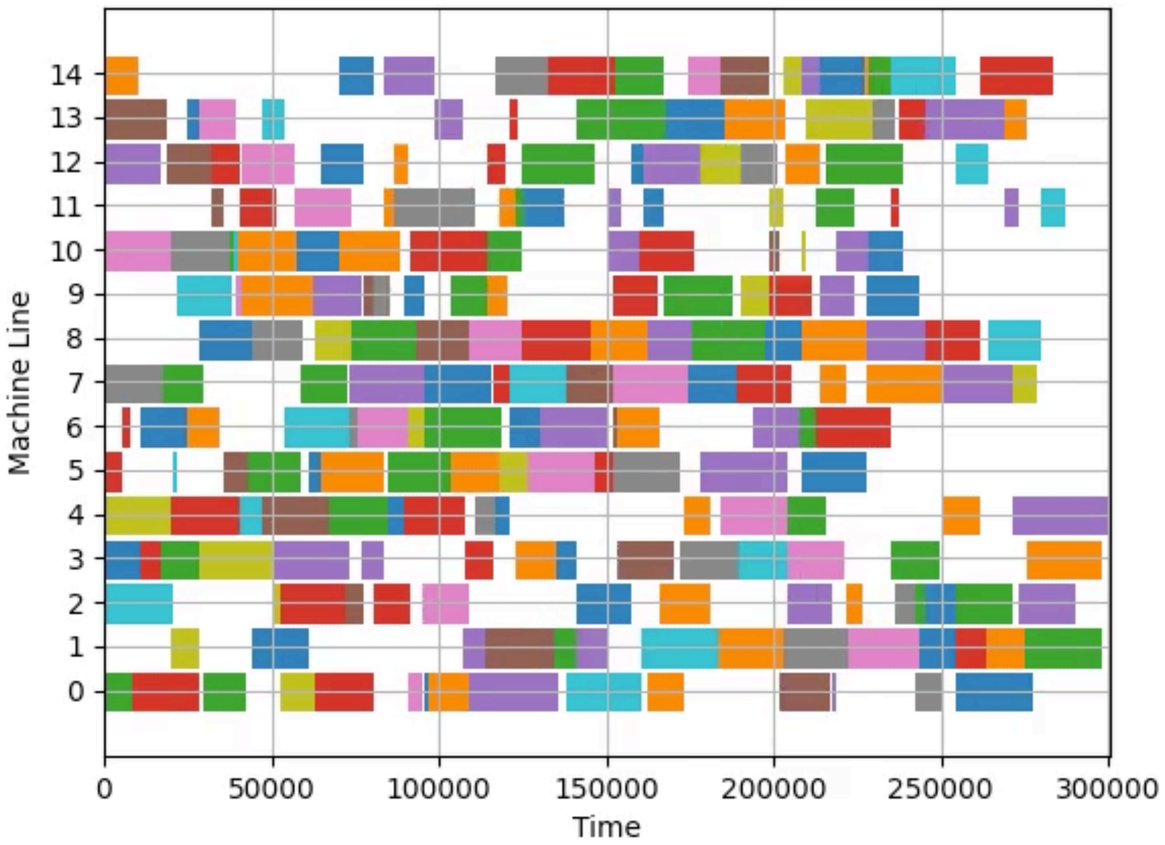


Schedule a set of jobs of varying processing times over a set of machines of varying processing power while minimizing the total length of the schedule.

Challenges in Learning Discrete Problems

Job shop scheduling

- Data generated by ”slowing down” machines, simulating some unexpected ill-functioning component in the scheduling pipeline.
- Slowing down even a single machine by 1 second produces large variabilities in the resulting schedules.
- Due to multiple symmetric solutions and/or when the outputs themselves are approximations.
- The resulting learning models produce inaccurate predictions.



La-39

Kotary et al.,AAAI:22

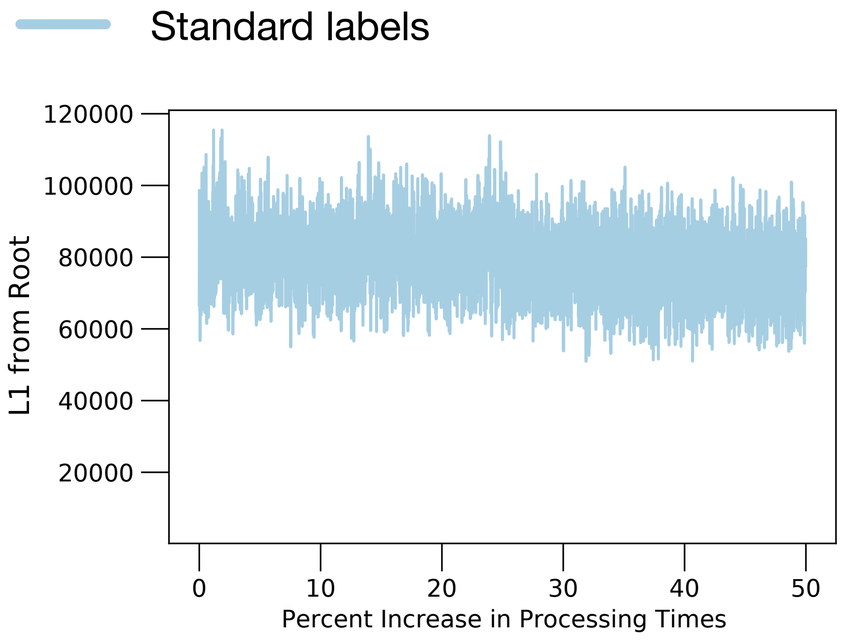
Optimal Data Generation

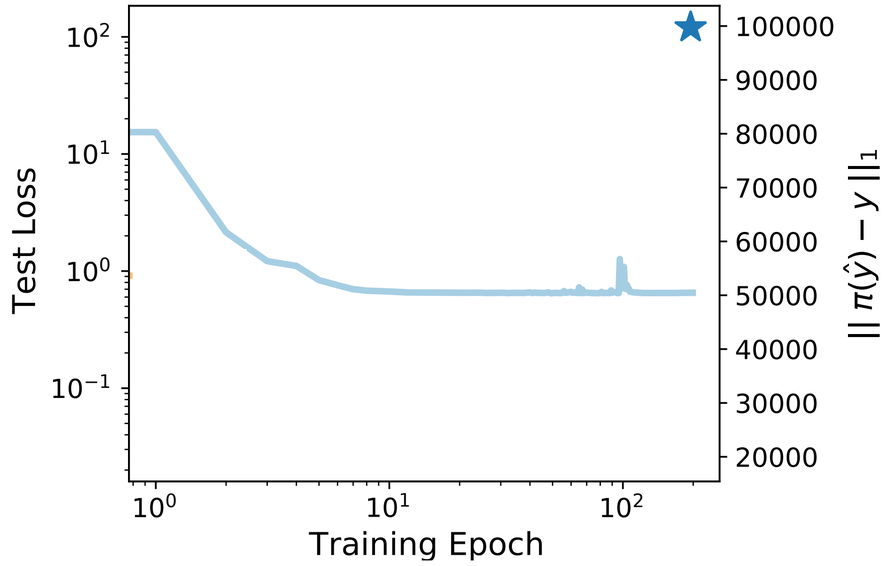
The solution trajectories may be significantly different depending on how the data is generated. Hence, the more volatile the trajectory, the harder it will be to learn.

Given a set of input data $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$, the goal is to construct the associated targets $y^{(i)}$ for each $i \in [N]$ that solve the following bi-level problem:

$$\min_{\theta, \boldsymbol{y}^{(i)}} \frac{1}{N} \sum_{i=1}^{N} \ell(f_\theta(\boldsymbol{x}^{(i)}), \boldsymbol{y}^{(i)})$$

$$\text{subject to}: \quad \boldsymbol{y}^{(i)} \in \operatorname*{argmin}_{\boldsymbol{y} \in \mathcal{C}_{\boldsymbol{x}^{(i)}}} f(\boldsymbol{y}, \boldsymbol{x}^{(i)}).$$





Kotary et al.,NeurIPS:21

Optimal Data Generation

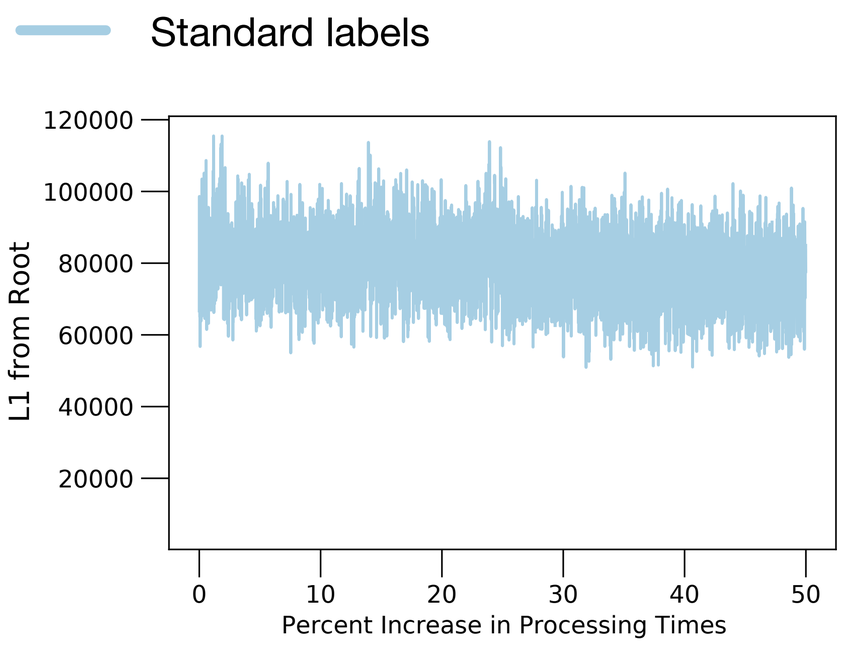
The solution trajectories may be significantly different depending on how the data is generated. Hence, the more volatile the trajectory, the harder it will be to learn.

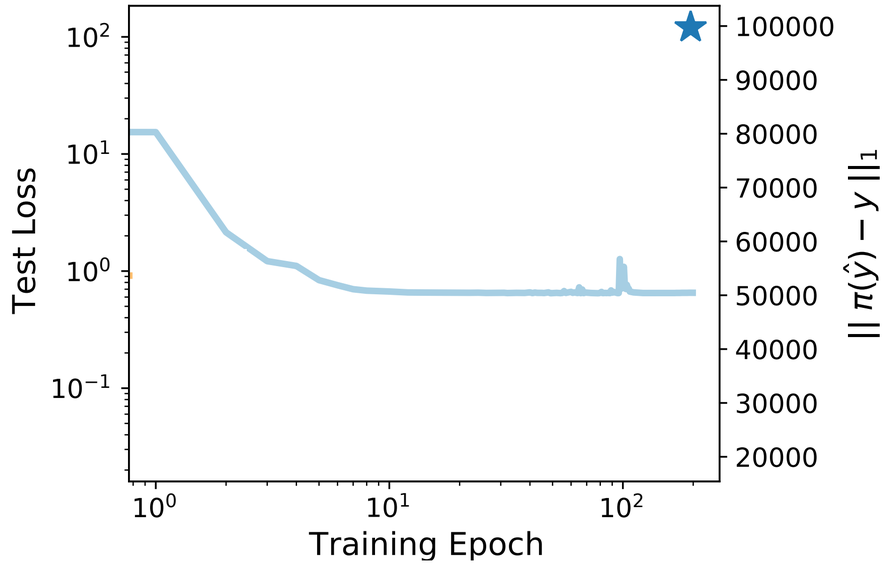
Given a set of input data $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$, the goal is to construct the associated targets $y^{(i)}$ for each $i \in [N]$ that solve the following bi-level problem:

$$\min_{\theta, \boldsymbol{y}^{(i)}} \frac{1}{N} \sum_{i=1}^{N} \ell(f_\theta(\boldsymbol{x}^{(i)}), \boldsymbol{y}^{(i)})$$

$$\text{subject to}: \quad \boldsymbol{y}^{(i)} \in \operatorname*{argmin}_{\boldsymbol{y} \in \mathcal{C}_{\boldsymbol{x}^{(i)}}} f(\boldsymbol{y}, \boldsymbol{x}^{(i)}).$$

Classical ERM





Kotary et al.,NeurIPS:21

Optimal Data Generation

The solution trajectories may be significantly different depending on how the data is generated. Hence, the more volatile the trajectory, the harder it will be to learn.

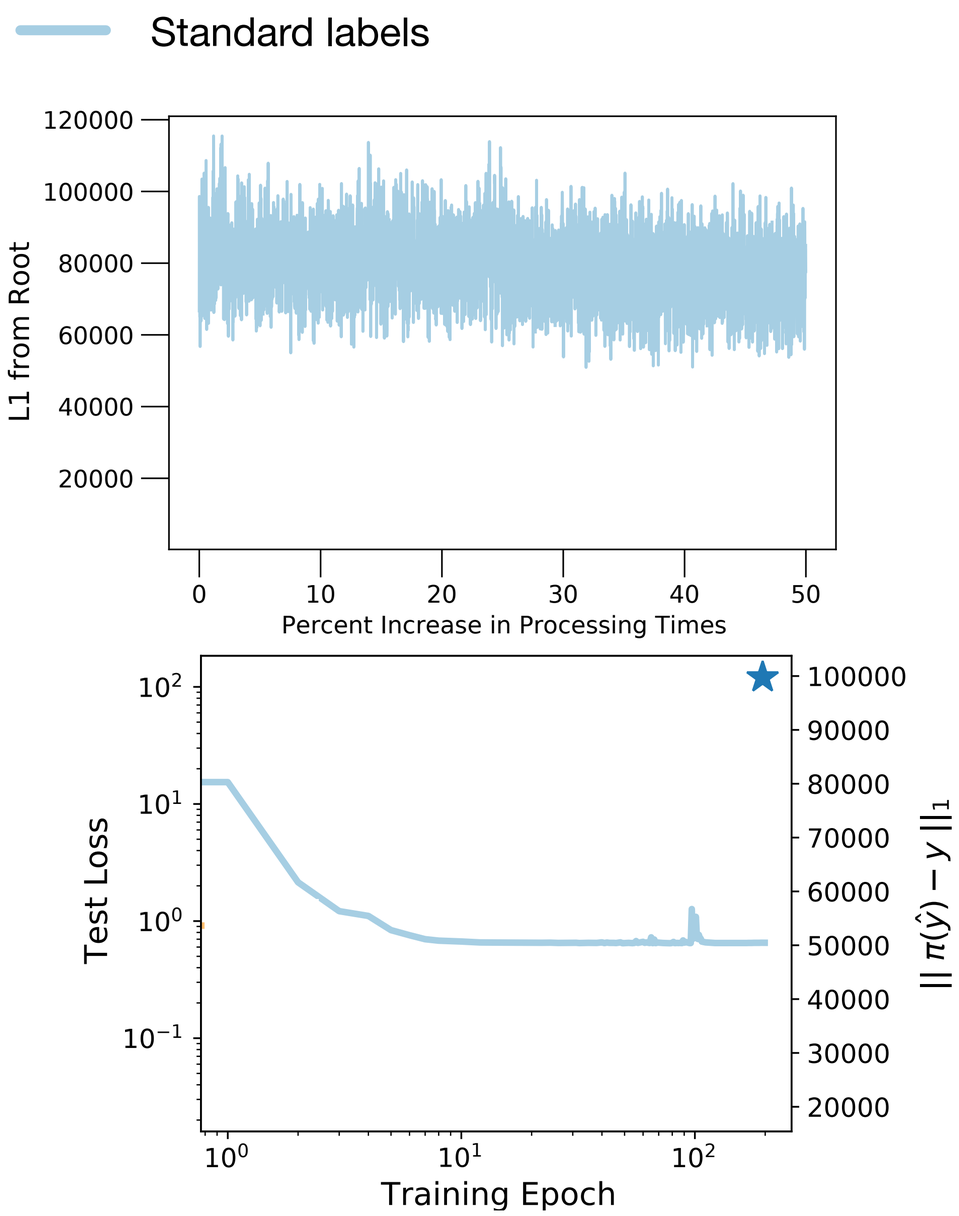
Given a set of input data $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$, the goal is to construct the associated targets $y^{(i)}$ for each $i \in [N]$ that solve the following bi-level problem:

$$\min_{\theta, \boldsymbol{y}^{(i)}} \frac{1}{N} \sum_{i=1}^{N} \ell(f_\theta(\boldsymbol{x}^{(i)}), \boldsymbol{y}^{(i)})$$

$$\text{subject to}: \ \boldsymbol{y}^{(i)} \in \operatorname*{argmin}_{\boldsymbol{y} \in \mathcal{C}_{\boldsymbol{x}^{(i)}}} f(\boldsymbol{y}, \boldsymbol{x}^{(i)}).$$

Classical ERM

Across all optimal solutions



Kotary et al.,NeurIPS:21

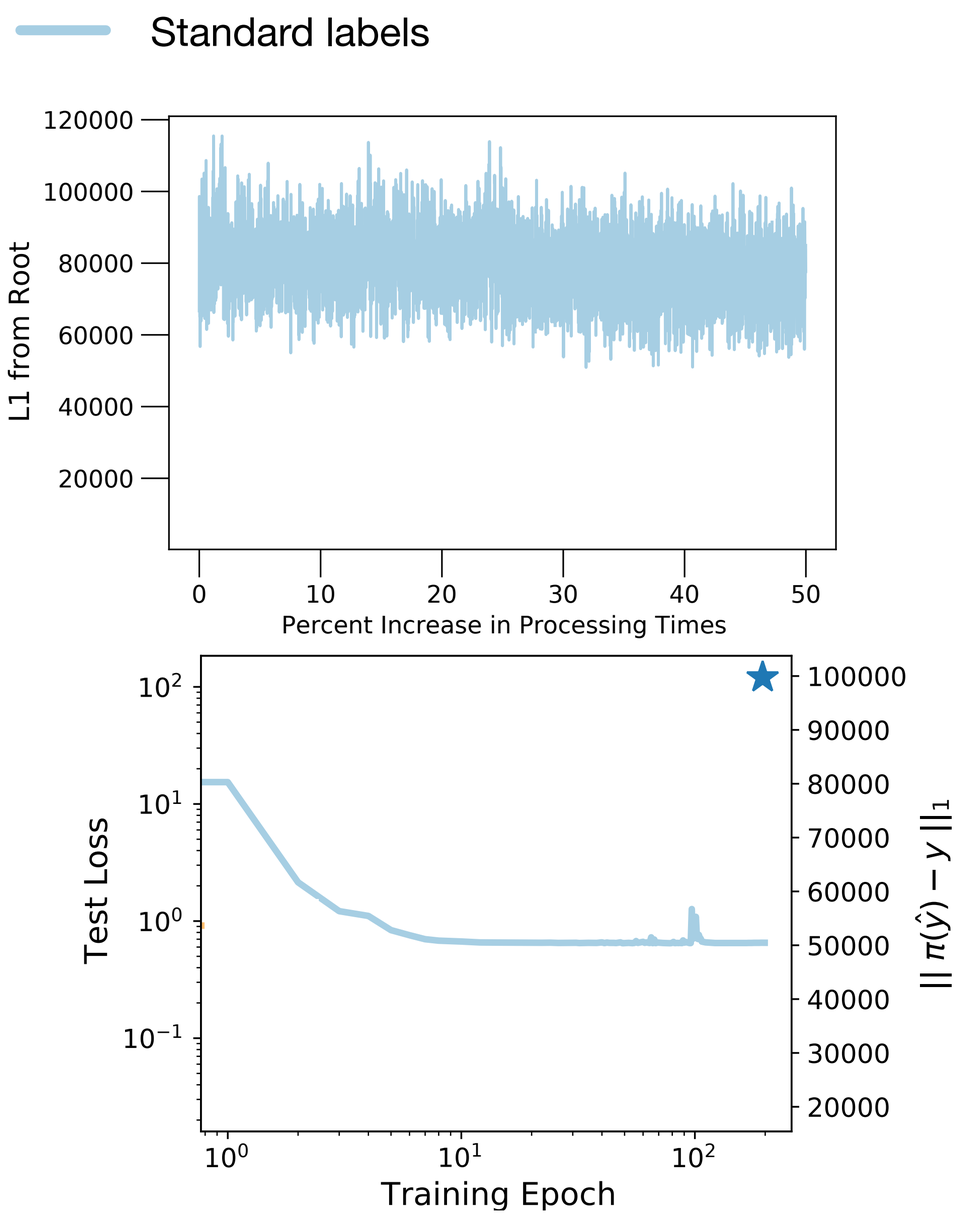
Optimal Data Generation

Theorem (model capacity). *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a piecewise linear function with p pieces. If f is represented by a ReLU network with depth $k+1$, then it must have size at least $\frac{1}{2}kp^{\frac{1}{k}} - 1$.*

Theorem (approximation). *If $f : [0,1]^n \to \mathbb{R}$ is L-Lipschitz continuous, then for every $\epsilon > 0$, there exists some single-layer neural network ρ of size N such that $\|f - \rho\|_\infty \leq \epsilon$ where $N = \binom{n+\frac{3L}{\epsilon}}{n}$.*

$$\text{ninimize } TV\left(\{\boldsymbol{y}^{(i)}\}_{i=1}^{N}\right) = \frac{1}{2}\sum_{i=1}^{N-1} \|\boldsymbol{y}^{(i+1)} - \boldsymbol{y}^{(i)}\|_p$$

$$\text{subject to : } \boldsymbol{y}^{(i)} = \underset{\boldsymbol{y} \in C_{\boldsymbol{x}^{(i)}}}{\operatorname{argmin}} f(\boldsymbol{y}, \boldsymbol{x}^{(i)}).$$



Kotary et al.,NeurIPS:21

Optimal Data Generation

Theorem (model capacity). *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a piecewise linear function with p pieces. If f is represented by a ReLU network with depth $k+1$, then it must have size at least $\frac{1}{2}kp^{\frac{1}{k}} - 1$.*

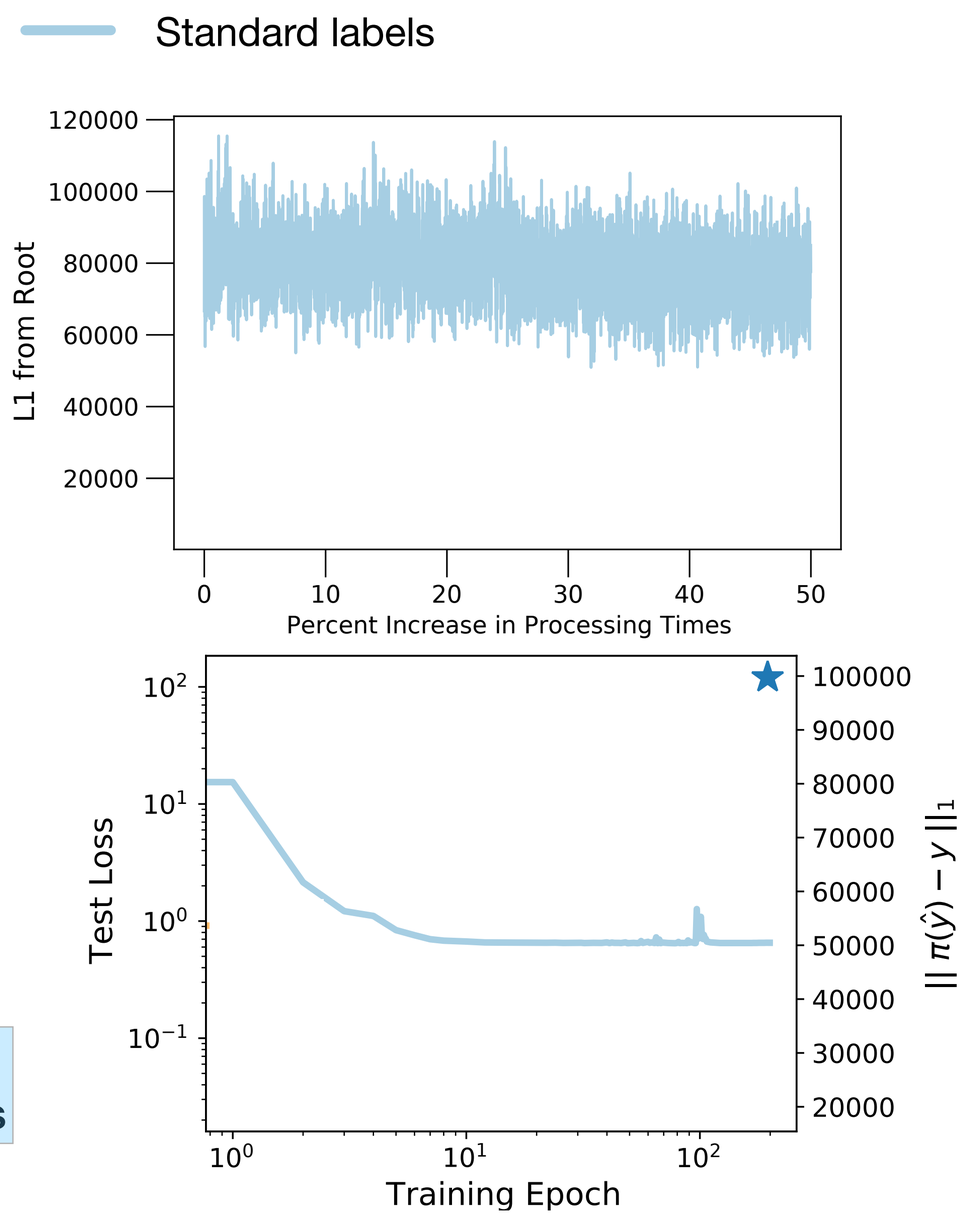
Theorem (approximation). *If $f : [0,1]^n \to \mathbb{R}$ is L-Lipschitz continuous, then for every $\epsilon > 0$, there exists some single-layer neural network ρ of size N such that $\|f - \rho\|_\infty \leq \epsilon$ where $N = \binom{n+\frac{3L}{\epsilon}}{n}$.*

$$\text{ninimize } TV\left(\{\boldsymbol{y}^{(i)}\}_{i=1}^{N}\right) = \frac{1}{2}\sum_{i=1}^{N-1} \|\boldsymbol{y}^{(i+1)} - \boldsymbol{y}^{(i)}\|_p$$

TV

$$\text{subject to : } \boxed{\boldsymbol{y}^{(i)}} = \underset{\boldsymbol{y} \in C_{\boldsymbol{x}^{(i)}}}{\operatorname{argmin}} f(\boldsymbol{y}, \boldsymbol{x}^{(i)}).$$

Across all optimal solutions



Kotary et al.,NeurIPS:21

Optimal Data Generation

Theorem (model capacity). *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a piecewise linear function with p pieces. If f is represented by a ReLU network with depth $k+1$, then it must have size at least $\frac{1}{2}kp^{\frac{1}{k}} - 1$.*

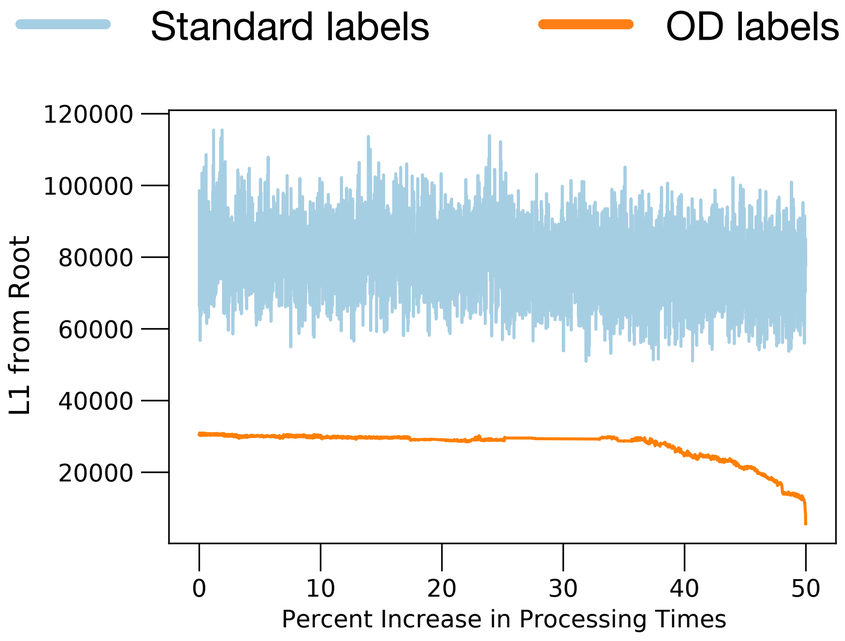
Theorem (approximation). *If $f : [0,1]^n \to \mathbb{R}$ is L-Lipschitz continuous, then for every $\epsilon > 0$, there exists some single-layer neural network ρ of size N such that $\|f - \rho\|_\infty \leq \epsilon$ where $N = \binom{n+\frac{3L}{\epsilon}}{n}$.*

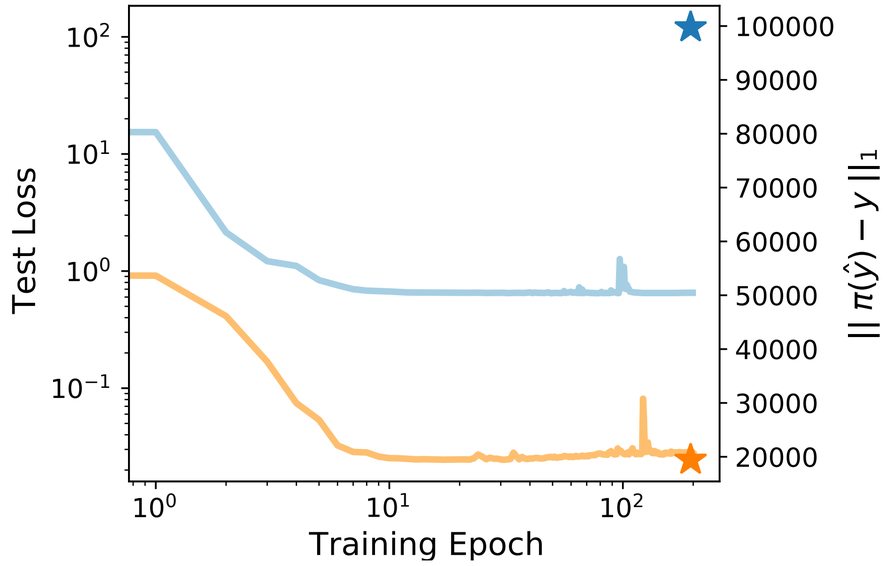
$$\text{ninimize } TV\left(\{\boldsymbol{y}^{(i)}\}_{i=1}^{N}\right) = \frac{1}{2}\sum_{i=1}^{N-1} \|\boldsymbol{y}^{(i+1)} - \boldsymbol{y}^{(i)}\|_p$$

TV

$$\text{subject to : } \boldsymbol{y}^{(i)} = \underset{\boldsymbol{y} \in C_{\boldsymbol{x}^{(i)}}}{\operatorname{argmin}} f(\boldsymbol{y}, \boldsymbol{x}^{(i)}).$$

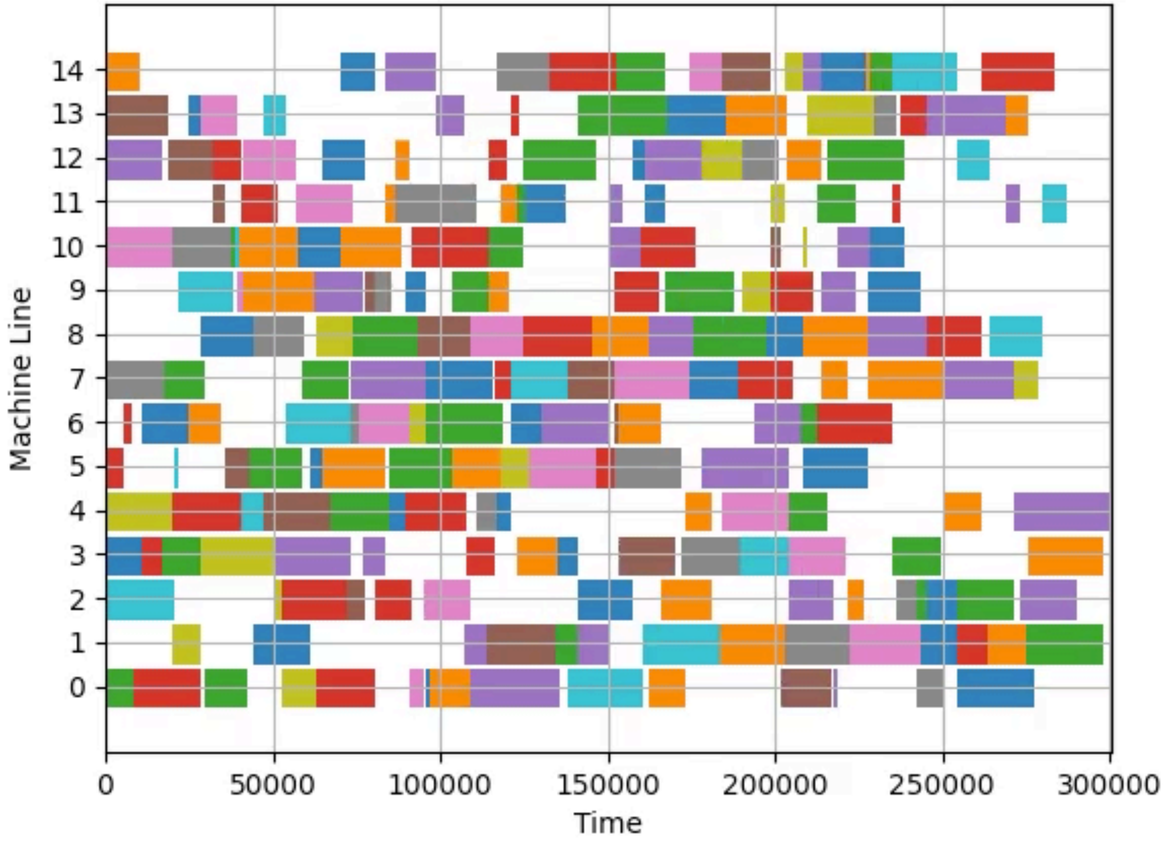
Across all optimal solutions



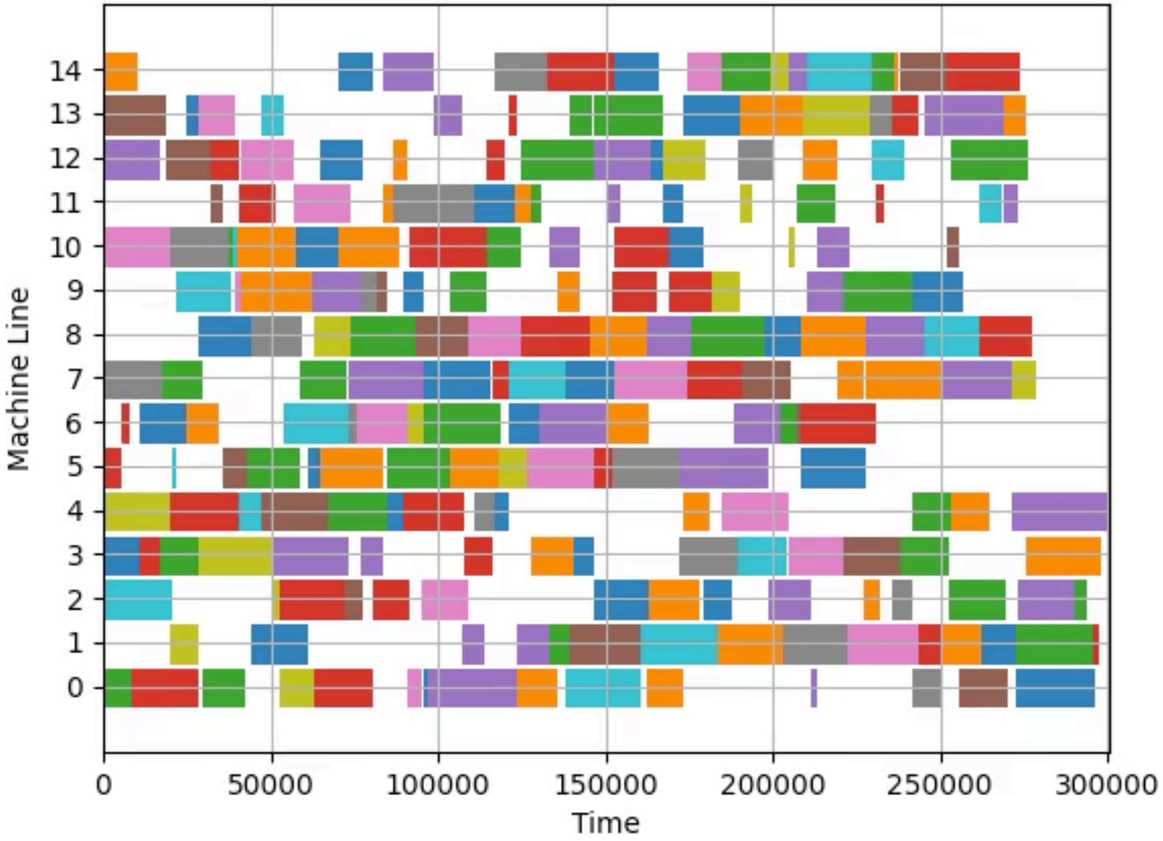


Kotary et al.,NeurIPS:21

Solution Volatility and Optimized Dataset



La-39 - standard labeled

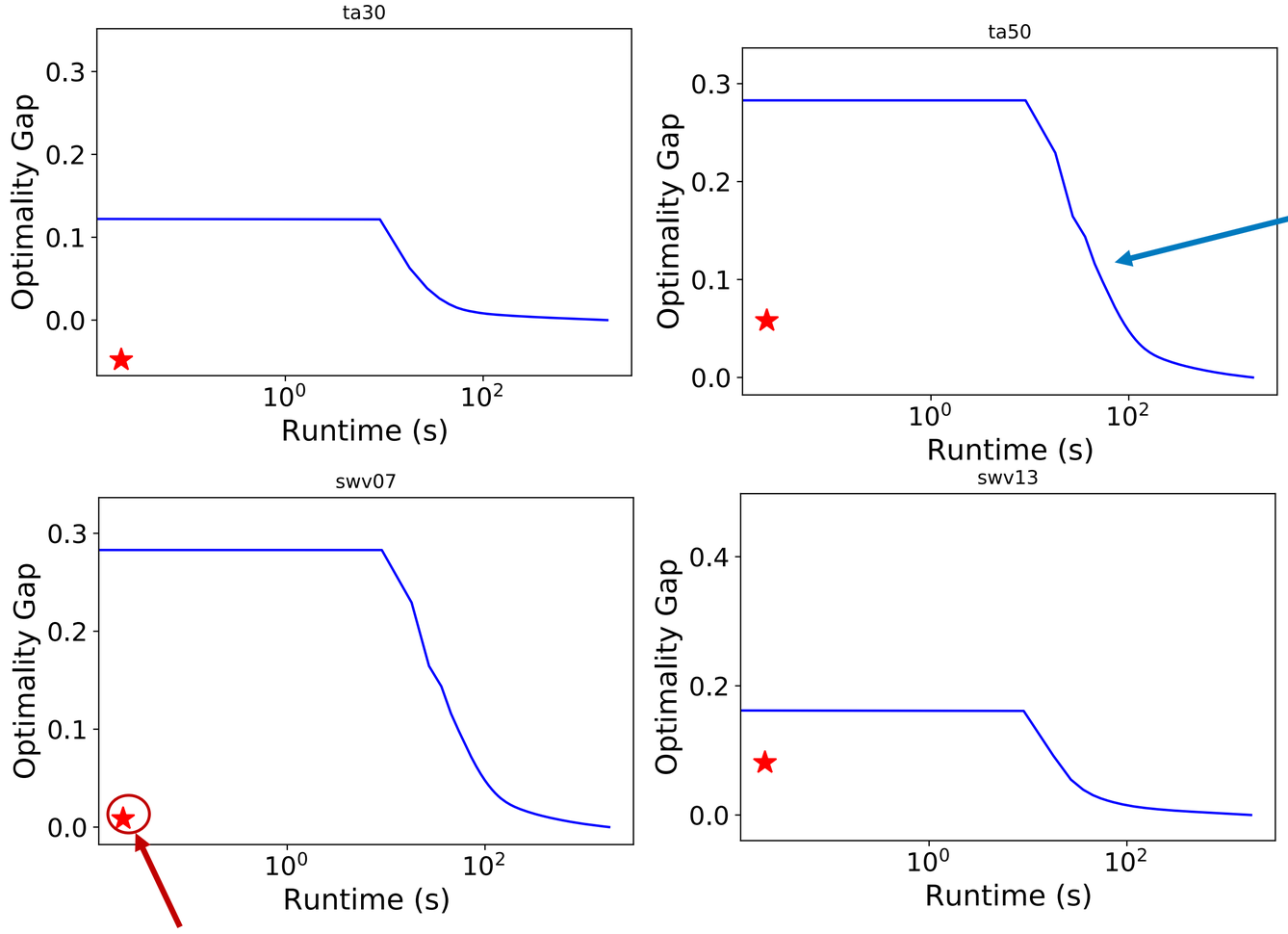


La-39 - optimized labeled

Kotary et al.,NeurIPS:21

How does it works in Practice?

Job shop scheduling problems



State-of-the-art commercial solver (IBM cp-optimizer)

Deep Learning based solution

Summary: In these settings, the ML-based models can predict feasible solutions often several order of magnitude faster than highly optimized combinatorial solvers.

Kotary et al.,AAAI:22

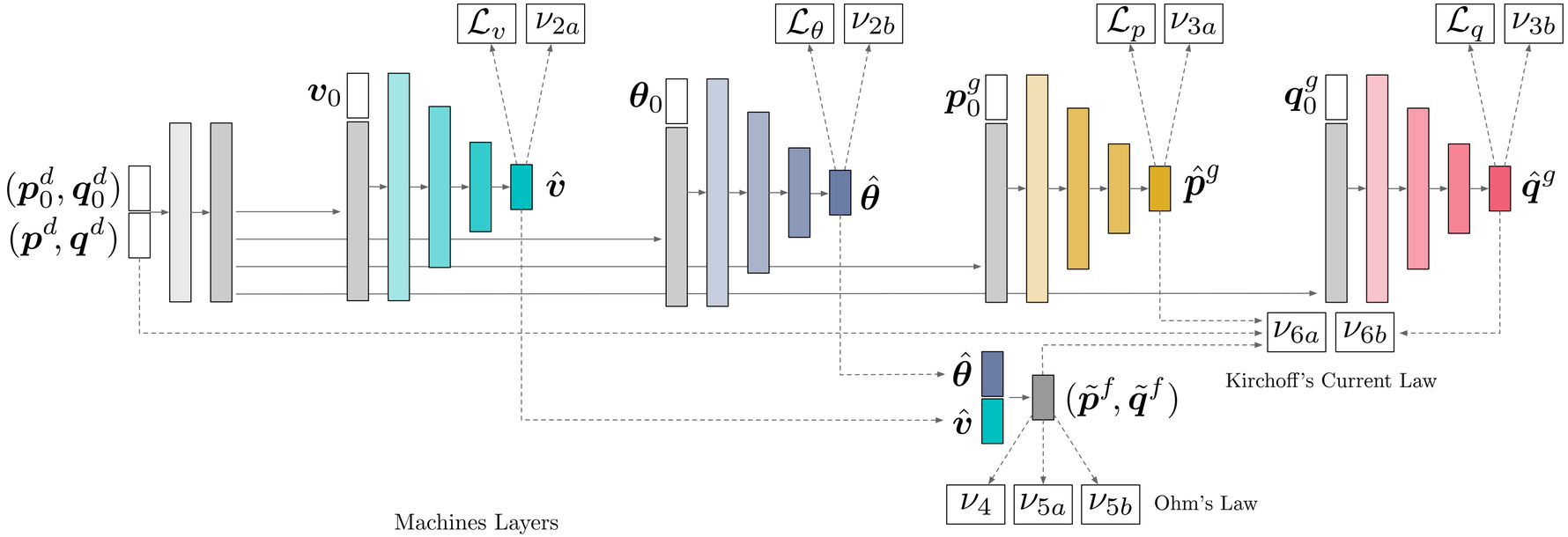
How does it works in Practice?

Job shop scheduling problems

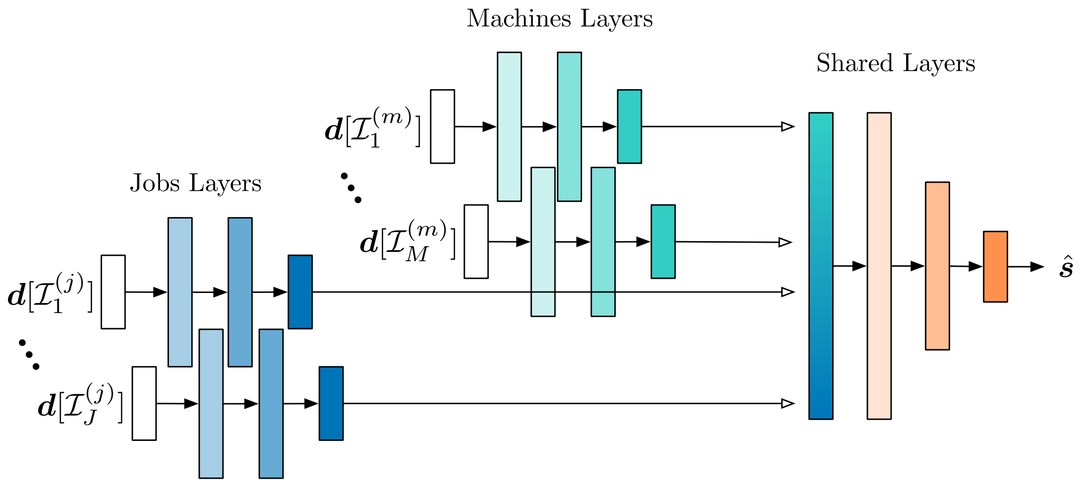
Instance	Size	Pred Err(×10) ↓		Con Viol($\times 10^2$) ↓		Opt. Gap Heuristics (%) ↓					Opt. Gap DNNs (%) ↓		Time SoTA Eq(s) ↑	
	$J \times M$	FC	JSP-DNN	FC	JSP-DNN	SPT	LWR	MWR	LOR	MOR	FC	JSP-DNN	FC	JSP-DNN
yn02	20×20	2.770	**0.138**	1.134	**0.122**	628	837	40	934	40	12.80 ± 5.4	**-0.045 ± 0.9**	10.20	**1800+**
ta25	20×20	1.607	**0.361**	0.631	**0.244**	593	877	59	787	46	13.61 ± 3.13	**-0.143 ± 0.8**	11.02	**1800+**
ta30	30×15	4.338	**1.196**	1.483	**0.357**	558	910	63	856	46	15.01 ± 2.63	**-0.48 ± 5.18**	9.06	**1800+**
ta40	30×20	7.880	**3.341**	1.863	**0.104**	492	794	57	836	25	23.11 ± 7.33	**3.19 ± 1.88**	8.40	**12.04**
ta50	50×10	4.580	**1.322**	1.223	**0.225**	789	789	53	1116	43	18.30 ± 5.22	**5.85± 2.72**	8.02	**90.30**
swv03	20×15	9.473	**2.683**	2.777	**0.850**	203	212	75	190	50	28.61 ± 14.27	**7.62 ± 2.51**	4.04	**36.36**
swv05	20×10	6.586	**2.950**	2.325	**0.626**	183	192	80	177	66	20.78 ± 10.54	**6.34 ± 1.82**	7.24	**18.18**
swv07	20×10	4.587	**0.681**	1.222	**0.223**	299	295	68	352	43	10.69 ± 6.83	**0.01± 4.75**	26.0	**254.5**
swv09	20×15	5.678	**3.462**	2.132	**0.211**	322	270	69	285	75	22.12 ± 8.52	**5.42 ± 1.21**	6.48	**28.32**
swv11	50×10	7.958	**3.244**	2.711	**0.282**	237	231	94	263	73	23.18 ± 2.27	**4.80 ± 4.47**	7.02	**92.00**
swv13	50×10	23.21	**3.557**	1.615	**0.323**	225	203	114	218	79	22.79 ± 16.21	**8.11 ± 4.20**	7.08	**24.08**

Kotary et al.,AAAI:22

Under the hood



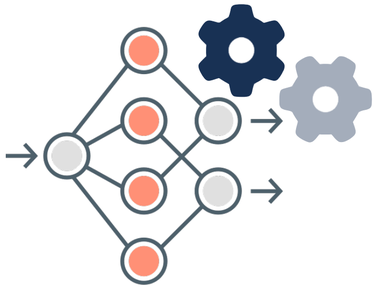
Constrained-aware network architecture for energy systems



Constrained-aware network architecture for scheduling problems

Summary: ML proxy optimizers

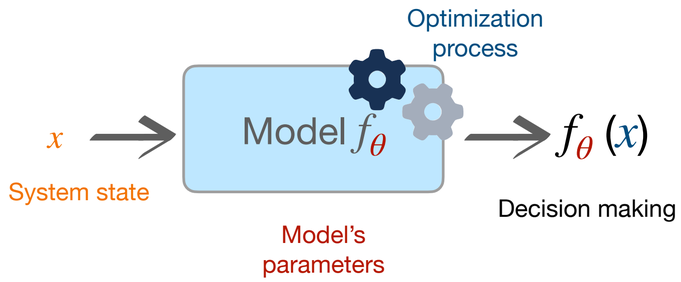
Enforcing physical principle and hard constraints



Motivation: Need for well-performing optimization surrogates that also obey the enforcement of physical principles and hard constraints.

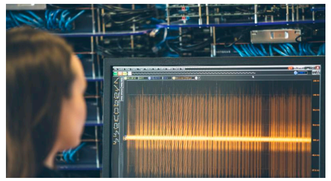
Insight: Combine optimization principles within the ML models training cycle.

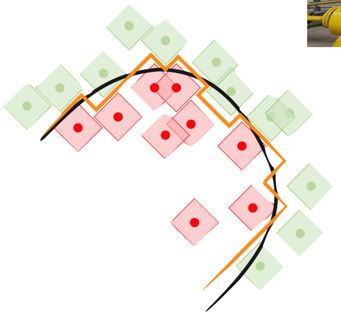
- Lagrangian-dual framework [ECML-19, AAAI-20]
- Lagrangian decomposition [TPWRS-21]



Settings:

- Optimal power flows in energy systems [AAAI-20, TPWRS-21, PMAS-22]
- Manufacture task scheduling [NeurIPS-21, AAAI-22]
- Transprecision computing [ECML-19]
- Enforcing fairness in ML models [ECAI-20, AAAI-21, IJCAI-22, AAAI-22]
- Enforcing fairness & robustness in ML models [CVPR-23*, ArXiv:22]
- Model pruning [NeurIPS-22]

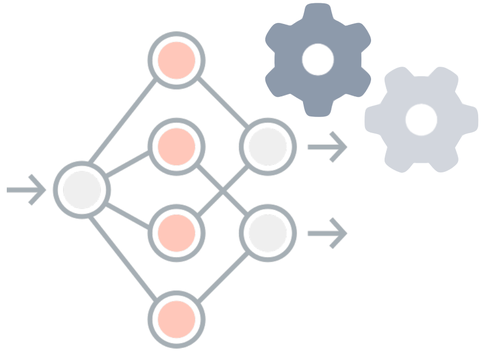




Agenda

Intro and Motivations

ML for Optimization



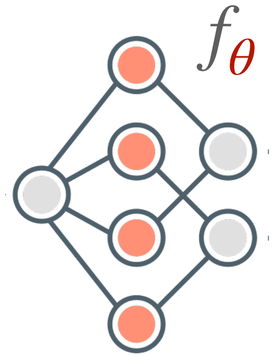
Optimization for ML

Open Questions

Optimization as Network Layers

Data

(x, y) $\rightarrow$

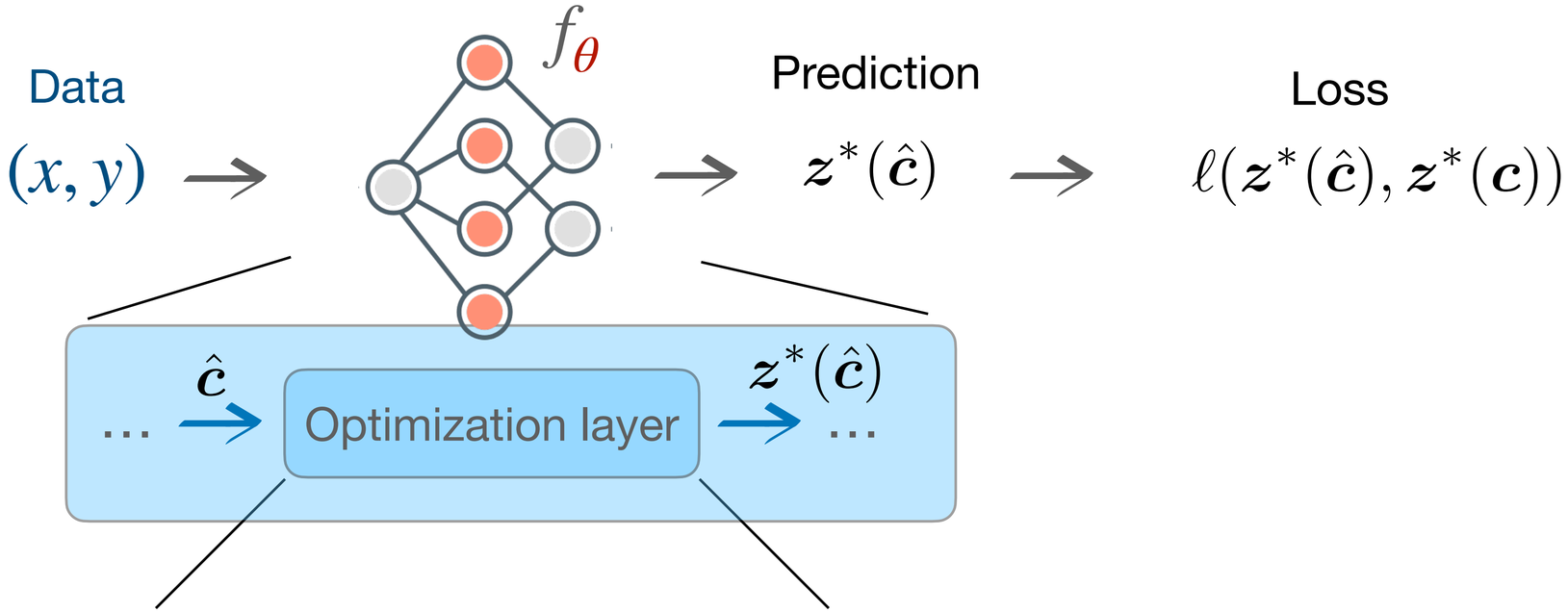


$\rightarrow$ Prediction

$\hat{\boldsymbol{c}}$ $\rightarrow$ Loss

$\ell(\hat{\boldsymbol{c}}, \boldsymbol{c})$

Optimization as Network Layers

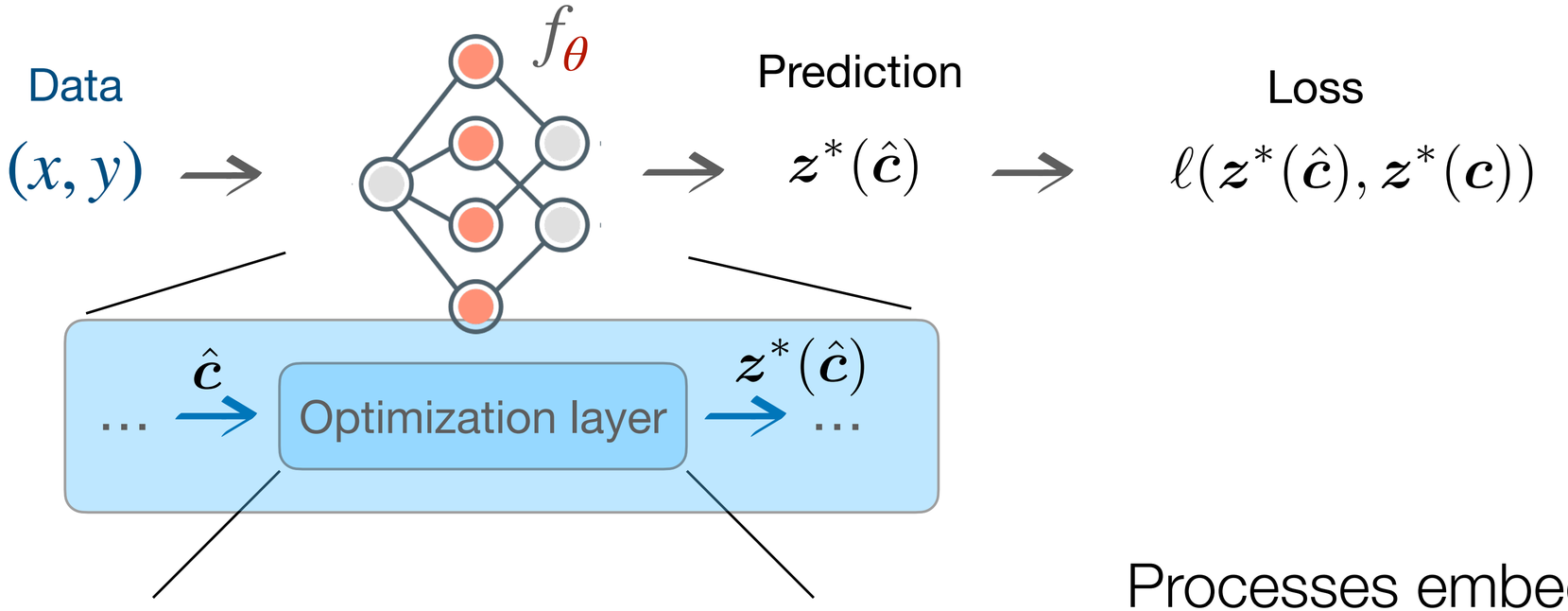


$$\boldsymbol{z}^*(\hat{\boldsymbol{c}}) = \underset{\boldsymbol{z}}{\operatorname{argmin}}\ f(\boldsymbol{z}, \hat{\boldsymbol{c}})$$

$$\text{subject to } g(\boldsymbol{z}, \hat{\boldsymbol{c}}) \leq 0$$

$$h(\boldsymbol{z}, \hat{\boldsymbol{c}}) = 0$$

Optimization as Network Layers

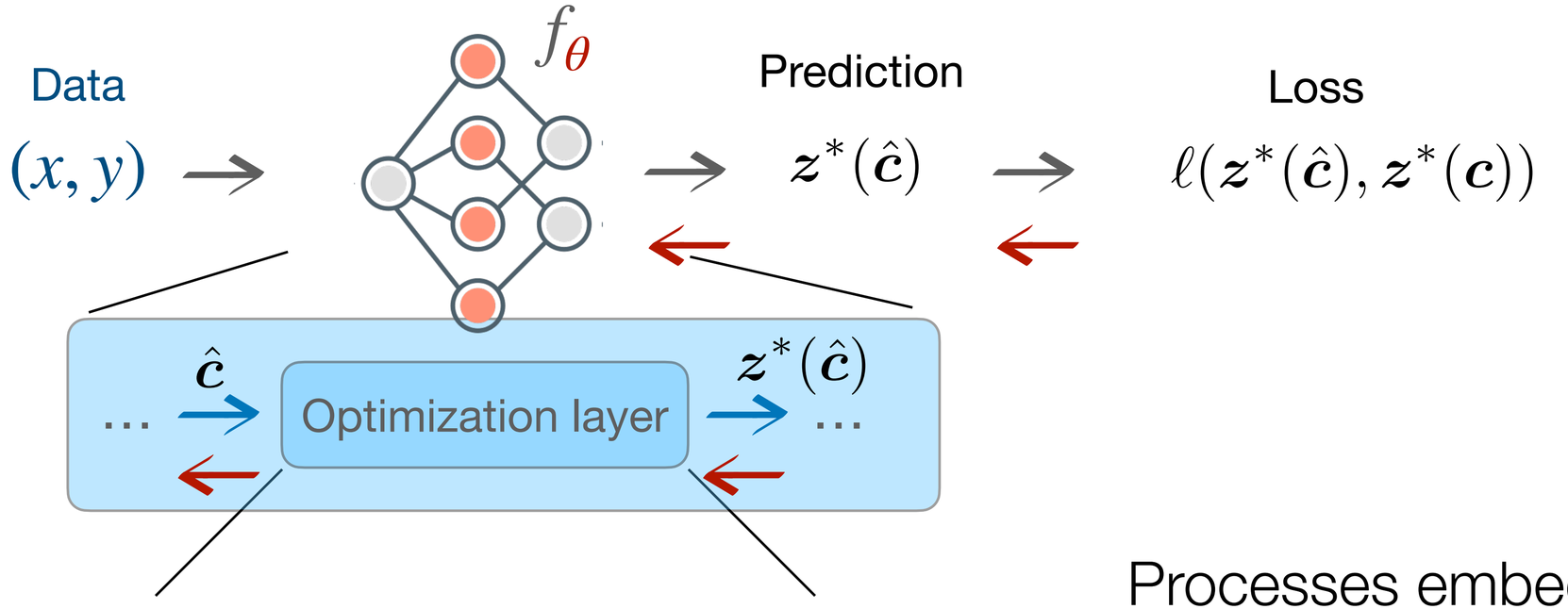


$$z^*(\hat{c}) = \underset{z}{\operatorname{argmin}}\ f(z, \hat{c})$$
$$\text{subject to } g(z, \hat{c}) \leq 0$$
$$h(z, \hat{c}) = 0$$

Processes embedding ($\hat{c}$) and outputs solutions $z^*(\hat{c})$ satisfying "arbitrary" structure.

Integrates domain knowledge and hard constraints in the learning pipeline (including at inference time).

Optimization as Network Layers



$$\boldsymbol{z}^*(\hat{\boldsymbol{c}}) = \underset{\boldsymbol{z}}{\operatorname{argmin}}\ f(\boldsymbol{z}, \hat{\boldsymbol{c}})$$

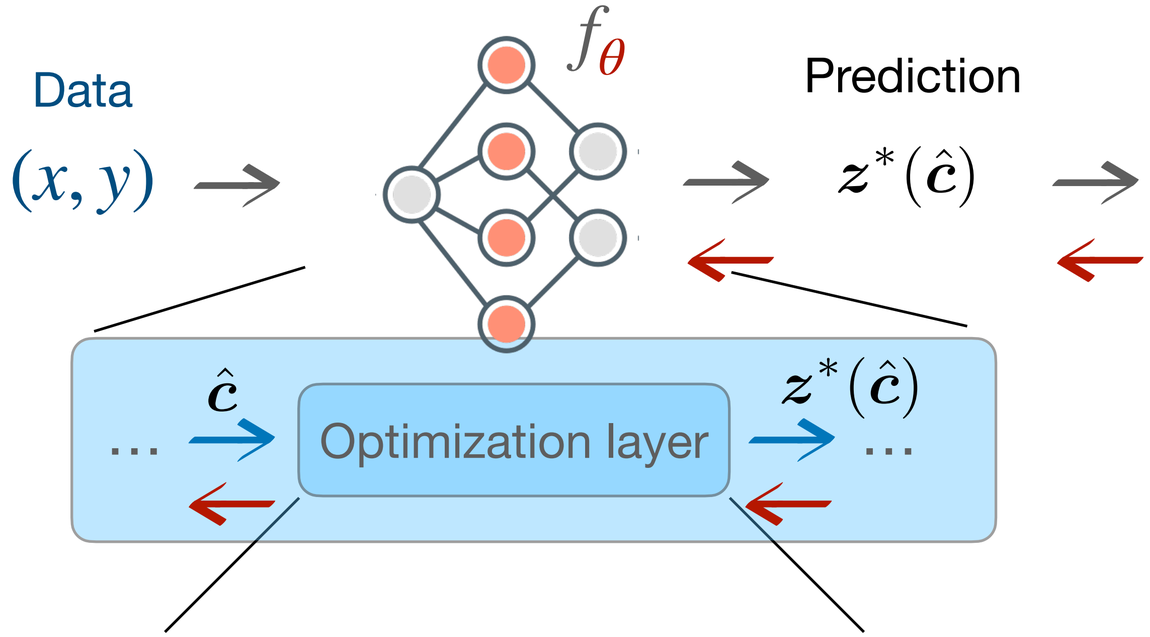
$$\text{subject to } g(\boldsymbol{z}, \hat{\boldsymbol{c}}) \leq 0$$

$$h(\boldsymbol{z}, \hat{\boldsymbol{c}}) = 0$$

Processes embedding ($\hat{c}$) and outputs solutions $z^*(\hat{c})$ satisfying “arbitrary" structure.

Integrates domain knowledge and hard constraints in the learning pipeline (including at inference time).

Optimization as Network Layers



$$\frac{\partial \ell}{\partial \theta} = \frac{\partial \hat{c}}{\partial \theta} \cdot \frac{\partial z^*(\hat{c})}{\partial \hat{c}} \cdot \frac{\partial \ell}{\partial z^*(\hat{c})}$$

$$z^*(\hat{c}) = \underset{z}{\operatorname{argmin}}\ f(z, \hat{c})$$

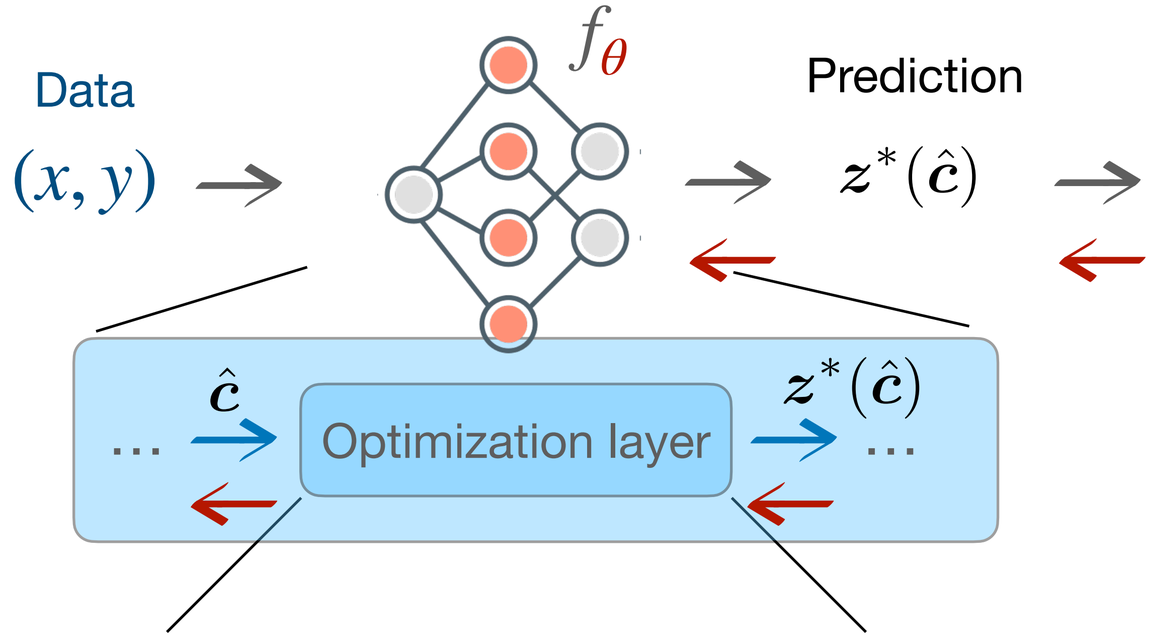
$$\text{subject to } g(z, \hat{c}) \leq 0$$

$$h(z, \hat{c}) = 0$$

Processes embedding ($\hat{c}$) and outputs solutions $z^*(\hat{c})$ satisfying "arbitrary" structure.

Integrates domain knowledge and hard constraints in the learning pipeline (including at inference time).

Optimization as Network Layers



Jacobian matrix of partial derivatives

$$\frac{\partial \ell}{\partial \theta} = \frac{\partial \hat{c}}{\partial \theta} \cdot \boxed{\frac{\partial z^*(\hat{c})}{\partial \hat{c}}} \cdot \frac{\partial \ell}{\partial z^*(\hat{c})}$$

$$z^*(\hat{c}) = \underset{z}{\operatorname{argmin}}\ f(z, \hat{c})$$
$$\text{subject to } g(z, \hat{c}) \leq 0$$
$$h(z, \hat{c}) = 0$$

Processes embedding ($\hat{c}$) and outputs solutions $z^*(\hat{c})$ satisfying "arbitrary" structure.

Integrates domain knowledge and hard constraints in the learning pipeline (including at inference time).

How do you differentiate through optimization programs?

Implicit differentiation

Convex quadratic programs

- QP layers can be learned by taking the gradients of some loss function w.r.t. the parameters.

- Differentiation can be obtained by differentiating the KKT conditions (sufficient and necessary conditions for optimality) at a solution to the problem.

QP Layer

Current layer

Previous layer

$$\boxed{z_{i+1}} = \underset{z}{\operatorname{argmin}} \ \frac{1}{2} z^T Q(\boxed{z_i}) z + q(\boxed{z_i})^T z$$

$$\text{subject to} \ \ A(\boxed{z_i}) z = b(\boxed{z_i})$$

$$G(\boxed{z_i}) z \leq h(\boxed{z_i})$$

Amos and Kolter ICML:2017

Implicit differentiation

Convex quadratic programs

- QP layers can be learned by taking the gradients of some loss function w.r.t. the parameters.

- Differentiation can be obtained by differentiating the KKT conditions (sufficient and necessary conditions for optimality) at a solution to the problem.

QP Layer

Current layer — Previous layer

$$\boxed{z_{i+1}} = \underset{z}{\operatorname{argmin}} \ \frac{1}{2} z^T Q(\boxed{z_i}) z + q(\boxed{z_i})^T z$$

$$\text{subject to} \ \ A(\boxed{z_i}) z = b(\boxed{z_i})$$

$$G(\boxed{z_i}) z \leq h(\boxed{z_i})$$

- The Lagrangian of the QP is $L(z, \boxed{\nu}, \boxed{\lambda}) = \frac{1}{2} z^T Q z + q^T z + \boxed{\nu}^T (Az - b) + \boxed{\lambda}^T (Gz - h)$

- The KKT conditions for stationarity, primal feasibility, and complementary slackness are:

optimal primal — optimal dual

$$Q\boxed{z^\star} + q + A^T \boxed{\nu^\star} + G^T \boxed{\lambda^\star} = 0$$

$$A\boxed{z^\star} - b = 0$$

$$D(\boxed{\lambda^\star})(G\boxed{z^\star} - h) = 0,$$

Diagonal matrix from vector

Taking the differentials $\longrightarrow$

$$\mathrm{d}Q\boxed{z^\star} + Q\mathrm{d}z + \mathrm{d}q + \mathrm{d}A^T \boxed{\nu^\star} +$$

$$A^T \mathrm{d}\nu + \mathrm{d}G^T \boxed{\lambda^\star} + G^T \mathrm{d}\lambda = 0$$

$$\mathrm{d}A\boxed{z^\star} + A\mathrm{d}z - \mathrm{d}b = 0$$

$$D(G\boxed{z^\star} - h)\mathrm{d}\lambda + D(\boxed{\lambda^\star})(\mathrm{d}G\boxed{z^\star} + G\mathrm{d}z - \mathrm{d}h) = 0$$

Amos and Kolter ICML:2017

Implicit differentiation

Convex quadratic programs

QP Layer

Current layer — Previous layer

$$\boxed{z_{i+1}} = \underset{z}{\operatorname{argmin}} \ \frac{1}{2} z^T Q(\boxed{z_i}) z + q(\boxed{z_i})^T z$$
$$\text{subject to} \ A(\boxed{z_i}) z = b(\boxed{z_i})$$
$$G(\boxed{z_i}) z \leq h(\boxed{z_i})$$

- with these equations, we can form the Jacobians of z^* w.r.t. any of the data parameters.
- For example, to compute $\frac{\mathrm{d}z^*}{\mathrm{d}b} \in \mathbb{R}^{n \times m}$ one would substitute $\mathrm{d}b = I$, solve the equation and retrieve the resulting value of $\mathrm{d}z$.

optimal primal — optimal dual

$$Q\boxed{z^\star} + q + A^T\boxed{\nu^\star} + G^T\boxed{\lambda^\star} = 0$$
$$A\boxed{z^\star} - b = 0$$
$$D(\boxed{\lambda^\star})(G\boxed{z^\star} - h) = 0,$$

Diagonal matrix from vector

Taking the differentials $\longrightarrow$

$$\mathrm{d}Q\boxed{z^\star} + Q\mathrm{d}z + \mathrm{d}q + \mathrm{d}A^T\boxed{\nu^\star} +$$
$$A^T\mathrm{d}\nu + \mathrm{d}G^T\boxed{\lambda^\star} + G^T\mathrm{d}\lambda = 0$$
$$\mathrm{d}A\boxed{z^\star} + A\mathrm{d}z - \mathrm{d}b = 0$$
$$D(G\boxed{z^\star} - h)\mathrm{d}\lambda + D(\boxed{\lambda^\star})(\mathrm{d}G\boxed{z^\star} + G\mathrm{d}z - \mathrm{d}h) = 0$$

Amos and Kolter ICML:2017

Implicit differentiation

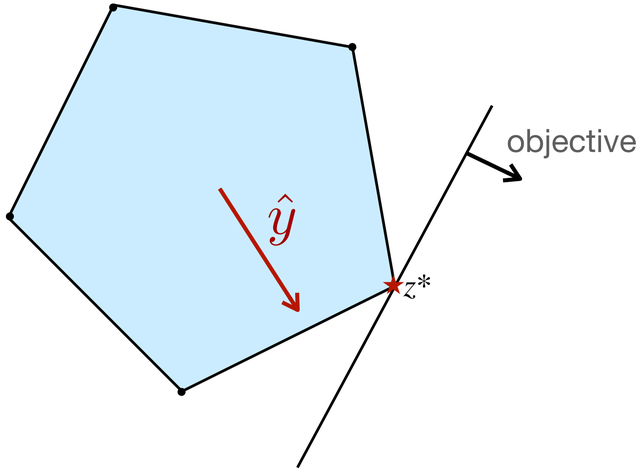
Linear Programs

- LP layers cannot be learned directly by differentiating through the KKT conditions.
- The optimal solution to an LP may not offer useful differentials w.r.t. predictions $\hat{y}$. This is because $\nabla_z^2 f(z, y) = y^t z$ is zero

LP Layer

Predictions from previous layer

$$z^* = \underset{z}{\operatorname{argmin}} \; \hat{y}^\top z$$
$$\text{subject to} \;\; Az \leq b$$



Wilder et al. AAAI 2019

Implicit differentiation

Linear Programs

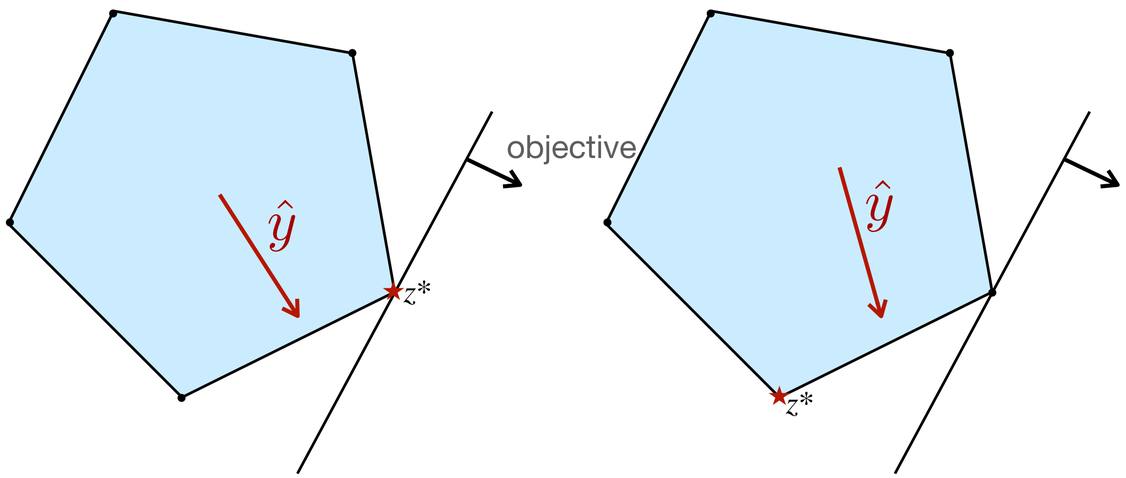
- LP layers cannot be learned directly by differentiating through the KKT conditions.
- The optimal solution to an LP may not offer useful differentials w.r.t. predictions $\hat{y}$. This is because $\nabla_z^2 f(z, y) = y^t z$ is zero
- Small changes in the model parameters $\hat{y}$ may cause discrete jumps to a new vertex.

LP Layer

Predictions from previous layer

$$z^* = \underset{z}{\operatorname{argmin}} \; \hat{y}^\top z$$

$$\text{subject to} \;\; Az \leq b$$



Wilder et al. AAAI 2019

Implicit differentiation

Linear Programs (Smooth surrogates)

LP Layer

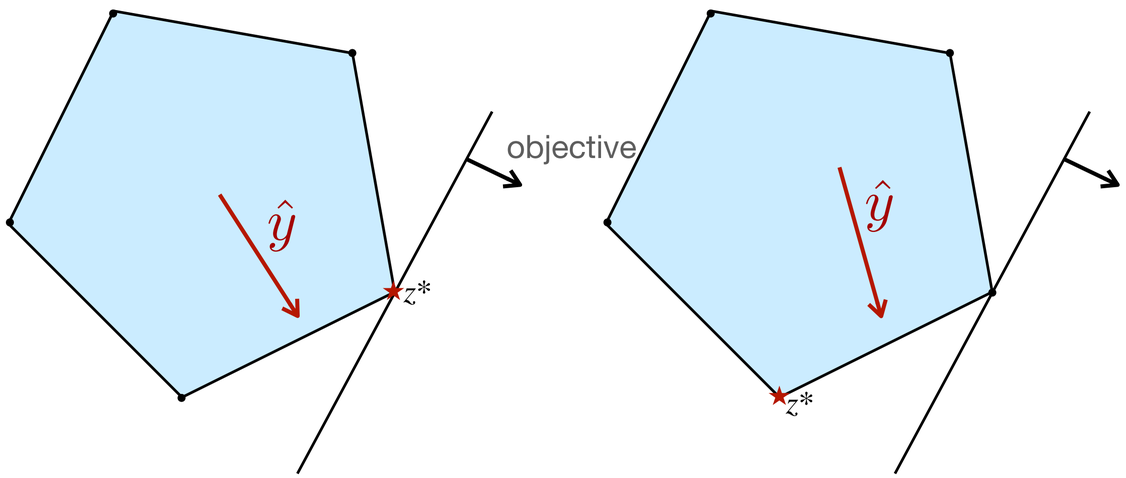
Predictions from previous layer

$$z^* = \underset{z}{\operatorname{argmin}} \ \hat{y}^\top z$$
$$\text{subject to} \ \ Az \leq b$$

- LP layers cannot be learned directly by differentiating through the KKT conditions.
- The optimal solution to an LP may not offer useful differentials w.r.t. predictions $\hat{y}$. This is because $\nabla_z^2 f(z, y) = y^t z$ is zero.
- Small changes in the model parameters $\hat{y}$ may cause discrete jumps to a new vertex.
- Solution: Add a regularizer to smooth the objective:

$$z^* = \underset{z}{\operatorname{argmin}} \ \hat{y}^\top z + \gamma \|z\|_2^2$$
$$\text{subject to} \ \ Az \leq b$$

- Hessian $\nabla_z^2 f(z, y) = 2\gamma I > 0$



Wilder et al. AAAI 2019

Implicit differentiation

Linear Programs (Perturbed optimizers)

LP Layer

Predictions from previous layer

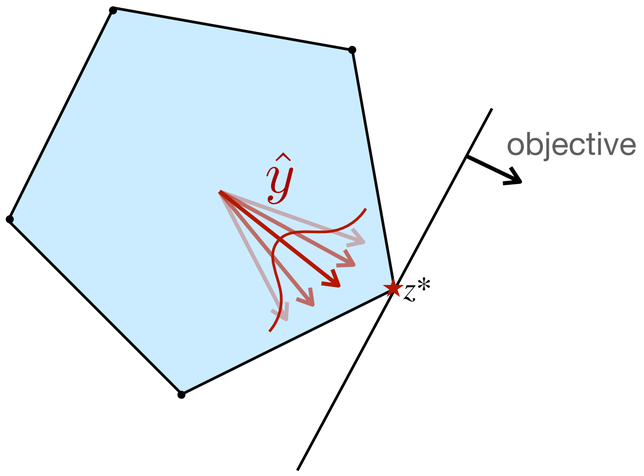
$$z^* = \underset{z}{\operatorname{argmin}} \; \hat{y}^\top z$$

$$\text{subject to} \;\; Az \leq b$$

- A different idea: Stochastically perturbing the coefficients $\hat{y}$ with noise from some distribution Z.

- This creates a distribution of *z* in the solution space for a given $\hat{y}$.

$$z^*(\hat{y}) = \mathbb{E}_Z \left[z * (\hat{y} + \epsilon Z) \right]$$



Berthet et al 2020

Implicit differentiation

Linear Programs (Perturbed optimizers)

LP Layer

Predictions from previous layer

$$z^* = \underset{z}{\operatorname{argmin}} \; \hat{y}^\top z$$

$$\text{subject to} \;\; Az \leq b$$

- A different idea: Stochastically perturbing the coefficients $\hat{y}$ with noise from some distribution Z.

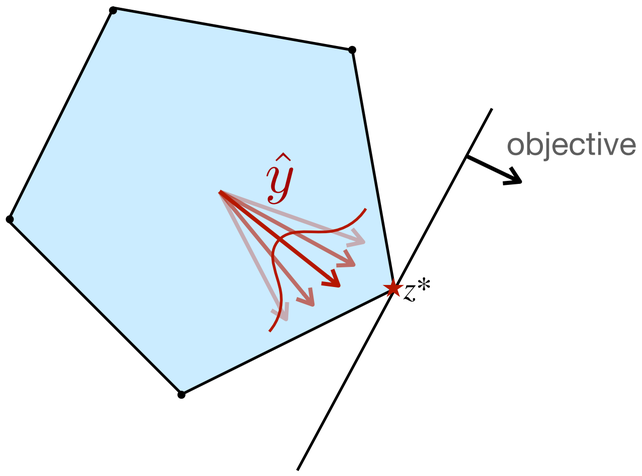
- This creates a distribution of *z* in the solution space for a given $\hat{y}$.

$$z^*(\hat{y}) = \mathbb{E}_Z \left[z * (\hat{y} + \epsilon Z) \right]$$

- The perturbed LP is now differentiable with derivative:

$$\frac{\partial z^*(\hat{y})}{\partial c} = \frac{1}{\epsilon} \mathbb{E}_Z \left[z * (\hat{y} + \epsilon Z) v'(Z)^\top \right]$$

parameter of the noise distribution



- (no close form) approximated via Monte-Carlo sampling.

Berthet et al 2020

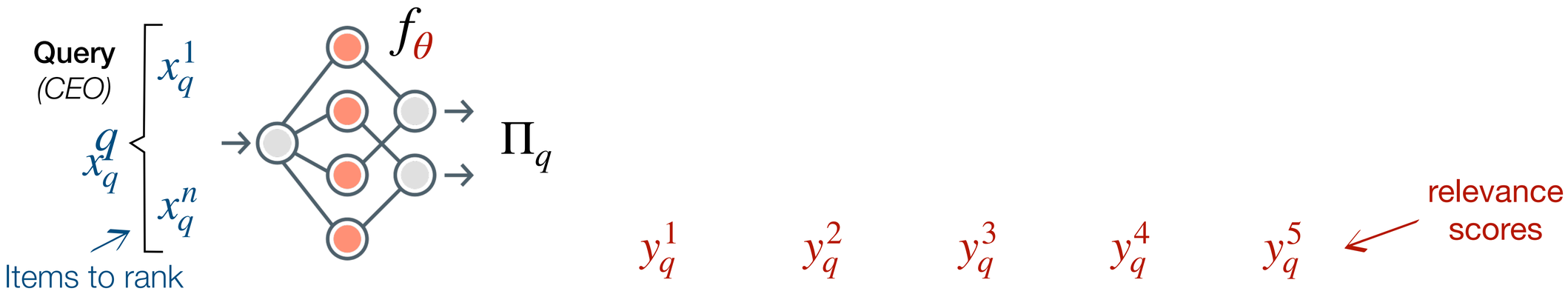
Some Applications

Enabling ML with the ability to satisfy constraints

Differentiable permutations

Certified fairness in learning to rank

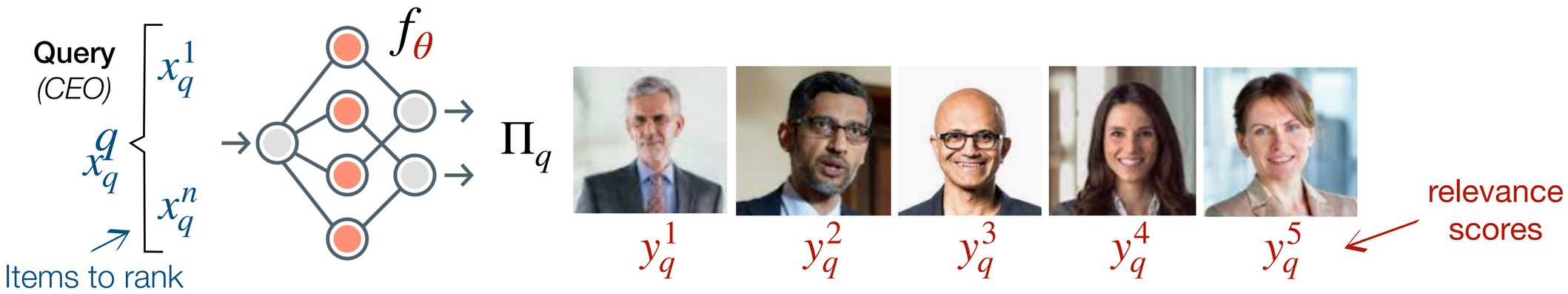
- Ranking systems are pervasive in online web searches, job searches, property renting, streaming content, etc.
- Learning to rank: construct mappings between lists of *n* items and permutations of the list [n] by learning a ranking policy Π from which rankings can be sampled.



Differentiable permutations

Certified fairness in learning to rank

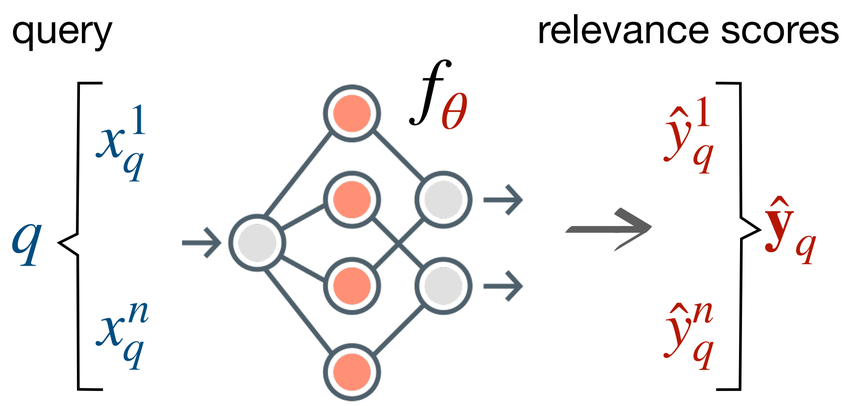
- Ranking systems are pervasive in online web searches, job searches, property renting, streaming content, etc.
- Learning to rank: construct mappings between lists of *n* items and permutations of the list [n] by learning a ranking policy Π from which rankings can be sampled.
- Algorithms used to learn rankings are typically oblivious to their potential disparate impacts on different groups of individuals.



Kotary et al WWW-22

Differentiable permutations

Learning fair rankings: challenges



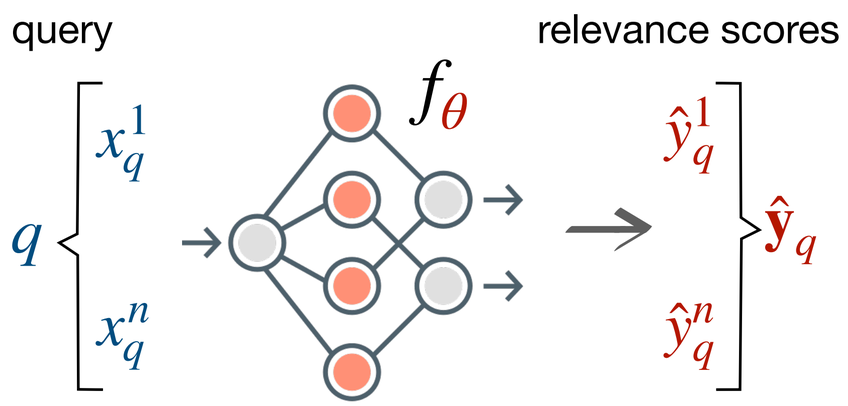
$$\theta^* = \underset{\theta}{\operatorname{argmax}} \ \frac{1}{N} \sum_{q=1}^{N} U(\ f_\theta(\boldsymbol{x}_q), \boldsymbol{y}_q)$$

$$\texttt{s.t.} \quad \left|\nu(\ f_\theta(\boldsymbol{x}_q), g)\right| \leq \delta \quad \forall q \in [N], g \in \mathcal{G}.$$

- **Fair learning to rank:** Learn rankings subject to fairness exposure constraints among protected groups.

Differentiable permutations

Learning fair rankings: challenges



$$\theta^* = \underset{\theta}{\operatorname{argmax}} \ \frac{1}{N} \sum_{q=1}^{N} U(\ f_\theta(\boldsymbol{x}_q), \boldsymbol{y}_q)$$

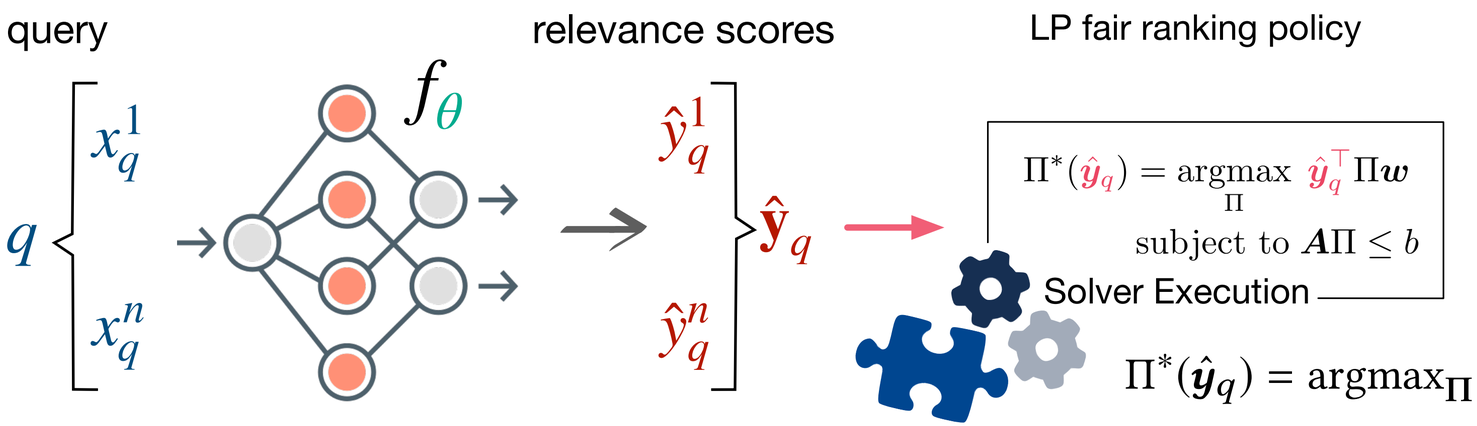
$$\texttt{s.t.} \quad \left|\nu(\ f_\theta(\boldsymbol{x}_q), g)\right| \leq \delta \quad \forall q \in [N], g \in \mathcal{G}.$$

- **Fair learning to rank:** Learn rankings subject to fairness exposure constraints among protected groups.
- Penalty-based methods are "satisfied" on average, thus disparities in favor of one group can cancel out those in favor of another group for different queries
- Cannot provide a tunable tradeoff between fairness and utility.

Kotary et al WWW-22

Differentiable permutations

Learning fair rankings: challenges



$$\Pi^*(\hat{\boldsymbol{y}}_q) = \text{argmax}_\Pi \ \hat{\boldsymbol{y}}_q^\top \Pi \, \boldsymbol{w}$$

$$\text{subject to:} \quad \sum_j \Pi_{ij} = 1 \quad \forall i \in [n]$$

$$\sum_i \Pi_{ij} = 1 \quad \forall j \in [n]$$

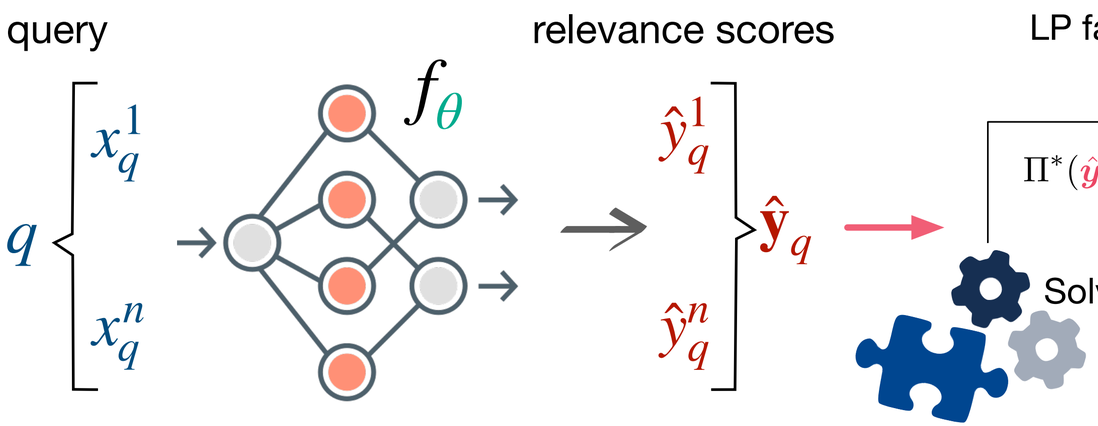
$$0 \leq \Pi_{ij} \leq 1 \quad \forall i, j \in [n]$$

$$|\nu(\Pi, g)| \leq \delta \quad \forall g \in \mathcal{G}$$

Kotary et al WWW-22

Differentiable permutations

Learning fair rankings: challenges



predictions

$$\Pi^*(\hat{\boldsymbol{y}}_q) = \operatorname{argmax}_\Pi\ \hat{\boldsymbol{y}}_q^\top \Pi\, \boldsymbol{w}$$

$$\text{subject to:}\quad \sum_j \Pi_{ij} = 1 \quad \forall i \in [n]$$

$$\sum_i \Pi_{ij} = 1 \quad \forall j \in [n]$$

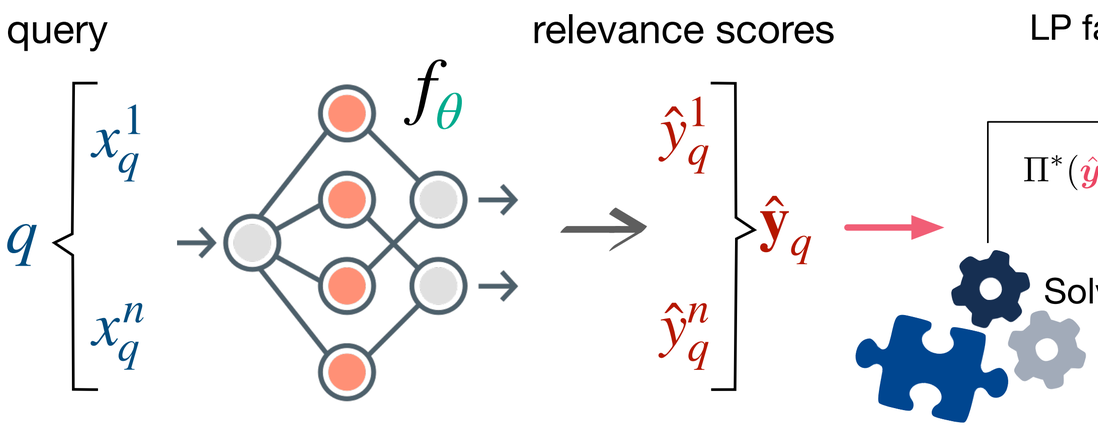
$$0 \leq \Pi_{ij} \leq 1 \quad \forall i, j \in [n]$$

$$|\nu(\Pi, g)| \leq \delta \quad \forall g \in \mathcal{G}$$

Kotary et al WWW-22

Differentiable permutations

Learning fair rankings: challenges



LP fair ranking policy

$$\Pi^*(\hat{\boldsymbol{y}}_q) = \underset{\Pi}{\operatorname{argmax}}\ \hat{\boldsymbol{y}}_q^\top \Pi \boldsymbol{w}$$
$$\text{subject to } \boldsymbol{A}\Pi \leq b$$

Solver Execution

predictions

permutation matrix

$$\Pi^*(\hat{\boldsymbol{y}}_q) = \operatorname{argmax}_\Pi\ \hat{\boldsymbol{y}}_q^\top \Pi \boldsymbol{w}$$

$$\text{subject to: } \sum_j \Pi_{ij} = 1 \quad \forall i \in [n]$$

doubly stochastic form

$$\sum_i \Pi_{ij} = 1 \quad \forall j \in [n]$$

$$0 \leq \Pi_{ij} \leq 1 \quad \forall i, j \in [n]$$

$$|\nu(\Pi, g)| \leq \delta \quad \forall g \in \mathcal{G}$$

A Birkhoff-von Neumann decomposition allows to represent ranking policies Π as a linear objective function.

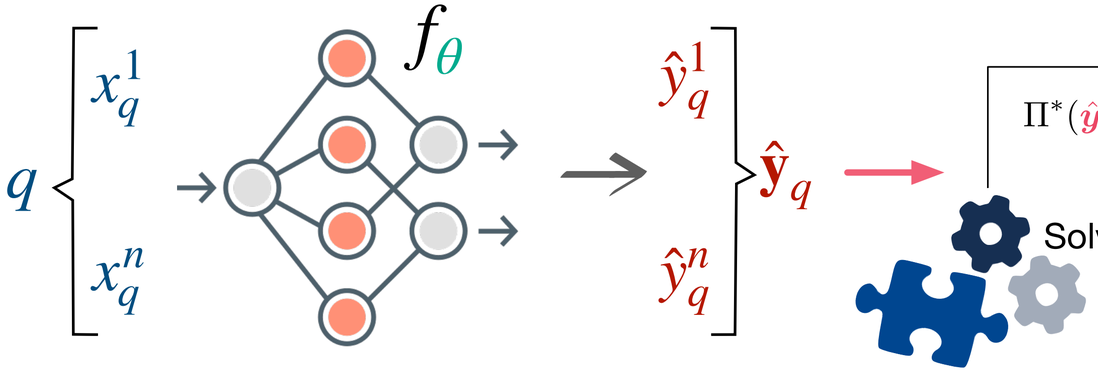
Differentiable permutations

Learning fair rankings: challenges

query

relevance scores

LP fair ranking policy



$$\Pi^*(\hat{\boldsymbol{y}}_q) = \underset{\Pi}{\operatorname{argmax}}\ \hat{\boldsymbol{y}}_q^\top \Pi \boldsymbol{w}$$
$$\text{subject to } \boldsymbol{A}\Pi \leq b$$

Solver Execution

predictions

permutation matrix

$$\Pi^*(\hat{\boldsymbol{y}}_q) = \operatorname{argmax}_\Pi\ \hat{\boldsymbol{y}}_q^\top \Pi \boldsymbol{w}$$

$$\text{subject to: } \sum_j \Pi_{ij} = 1 \quad \forall i \in [n]$$

doubly stochastic form

$$\sum_i \Pi_{ij} = 1 \quad \forall j \in [n]$$

$$0 \leq \Pi_{ij} \leq 1 \quad \forall i, j \in [n]$$

$$|\nu(\Pi, g)| \leq \delta \quad \forall g \in \mathcal{G}$$

Position bias disparity

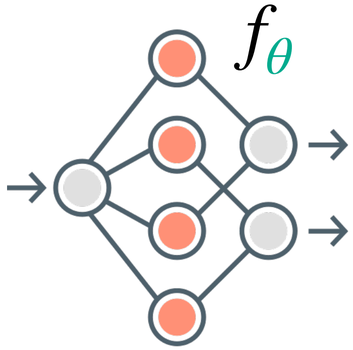
A Birkhoff-von Neumann decomposition allows to represent ranking policies Π as a linear objective function.

Differentiable permutations

Learning fair rankings: challenges

query

$$q \left\{ \begin{bmatrix} x_q^1 \\ \\ x_q^n \end{bmatrix} \right.$$



relevance scores

$$\left. \begin{bmatrix} \hat{y}_q^1 \\ \\ \hat{y}_q^n \end{bmatrix} \right\} \hat{\mathbf{y}}_q$$

LP fair ranking policy

$$\Pi^*(\hat{\boldsymbol{y}}_q) = \underset{\Pi}{\operatorname{argmax}} \ \hat{\boldsymbol{y}}_q^\top \Pi \boldsymbol{w}$$
$$\text{subject to } \boldsymbol{A}\Pi \leq b$$

Solver Execution

Regret loss

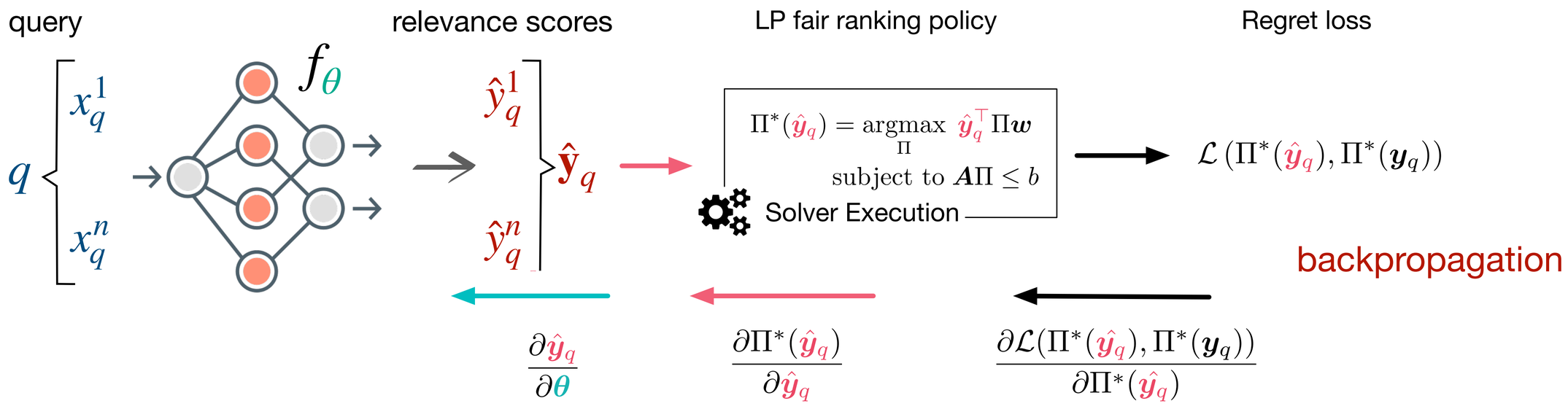
$$\mathcal{L}\left(\Pi^*(\hat{\boldsymbol{y}}_q), \Pi^*(\boldsymbol{y}_q)\right)$$

Regret between exact and learned policies

Kotary et al WWW-22

Differentiable permutations

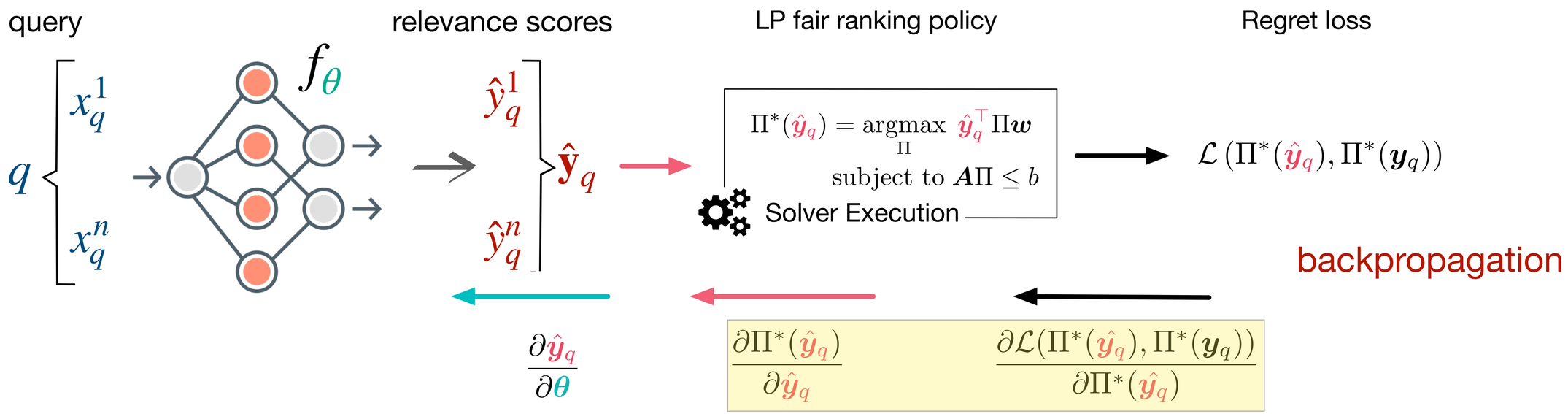
Learning fair rankings: challenges



Kotary et al WWW-22

Differentiable permutations

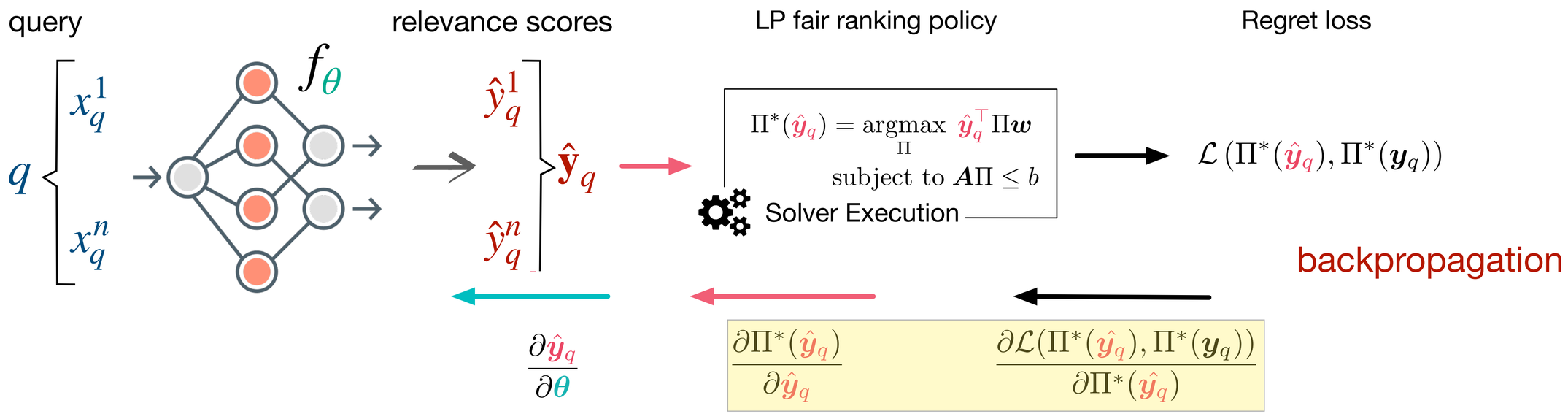
Learning fair rankings: challenges



- Discontinuous w.r.t. $\hat{y}$ for fixed y.

Differentiable permutations

Learning fair rankings: challenges



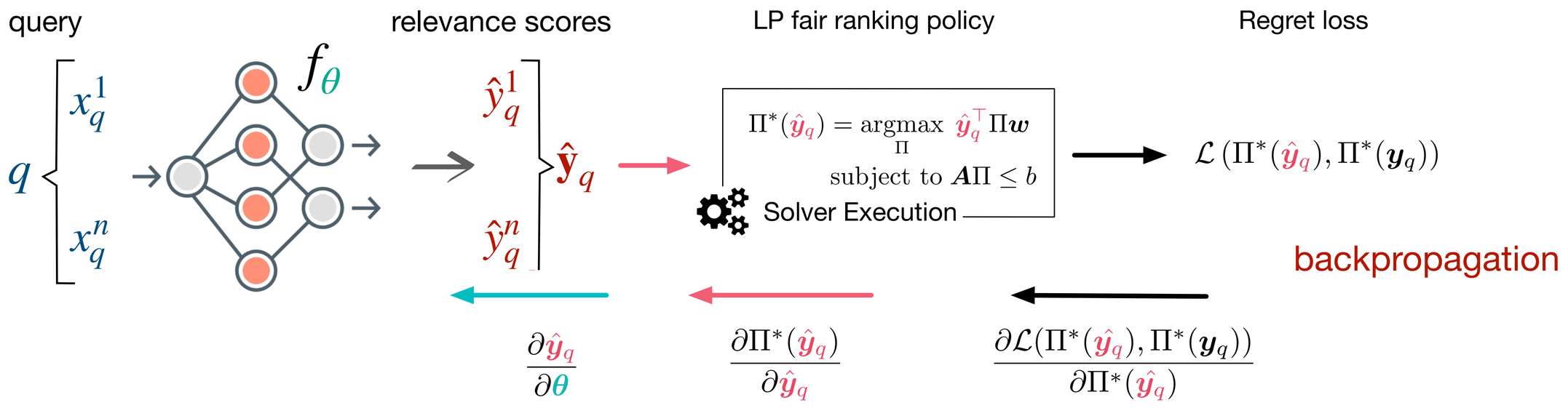
$$\approx \frac{\partial}{\partial \boldsymbol{y}} \mathcal{L}_{\text{SPO+}}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \Pi^*(2\hat{\boldsymbol{y}} - \boldsymbol{y}) - \Pi^*(\boldsymbol{y}).$$

- Discontinuous w.r.t. $\hat{y}$ for fixed y.
- Convex surrogate loss function which forms a convex upper-bounding function over $\mathcal{L}$.

Kotary et al WWW-22

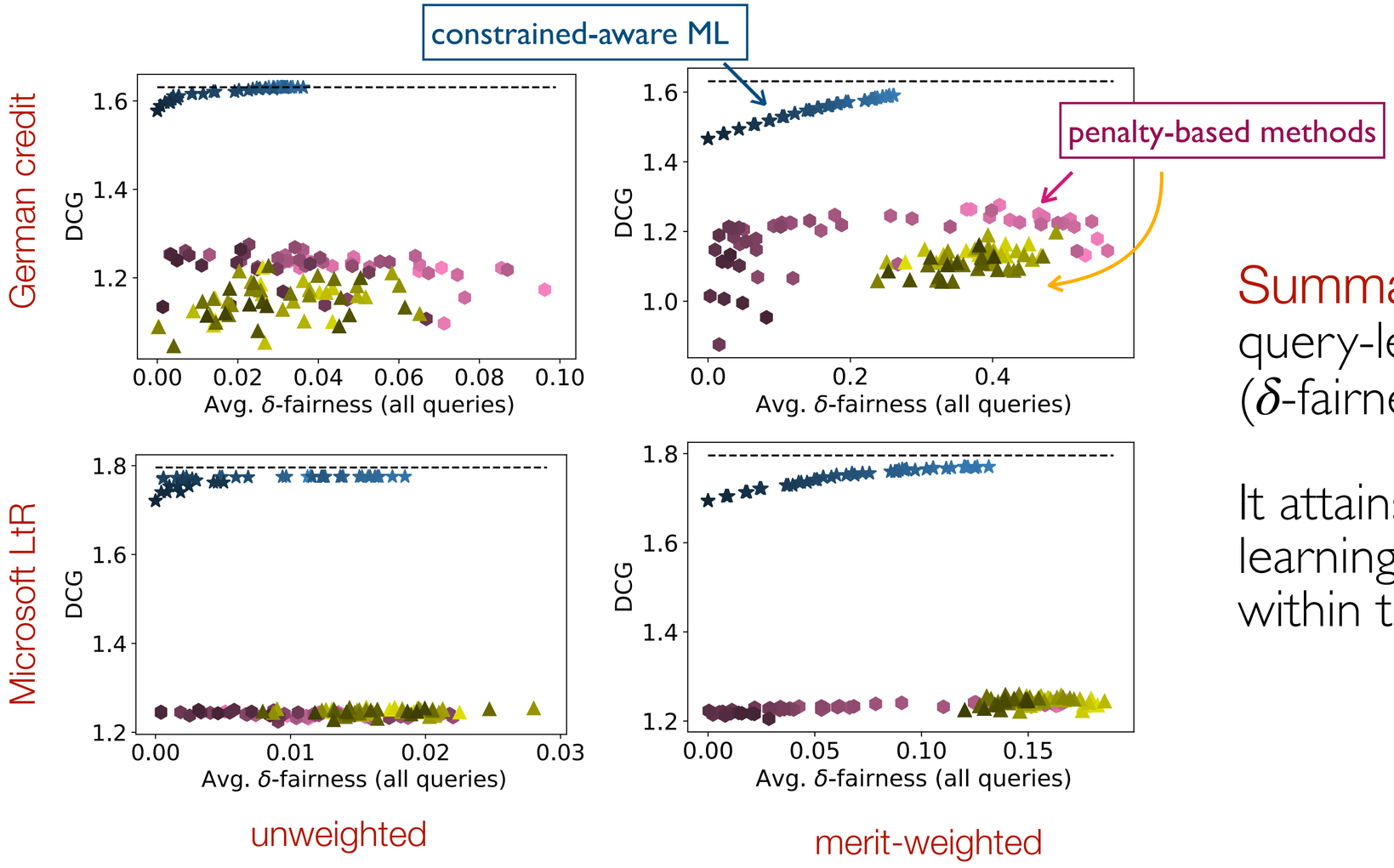
Differentiable permutations

Learning fair rankings: challenges



In contrast to SoTA fair-LtR methods, this end-to-end solution does not require sampling from ranking policy during training in order to evaluate the ranking utilities.

How does it work?

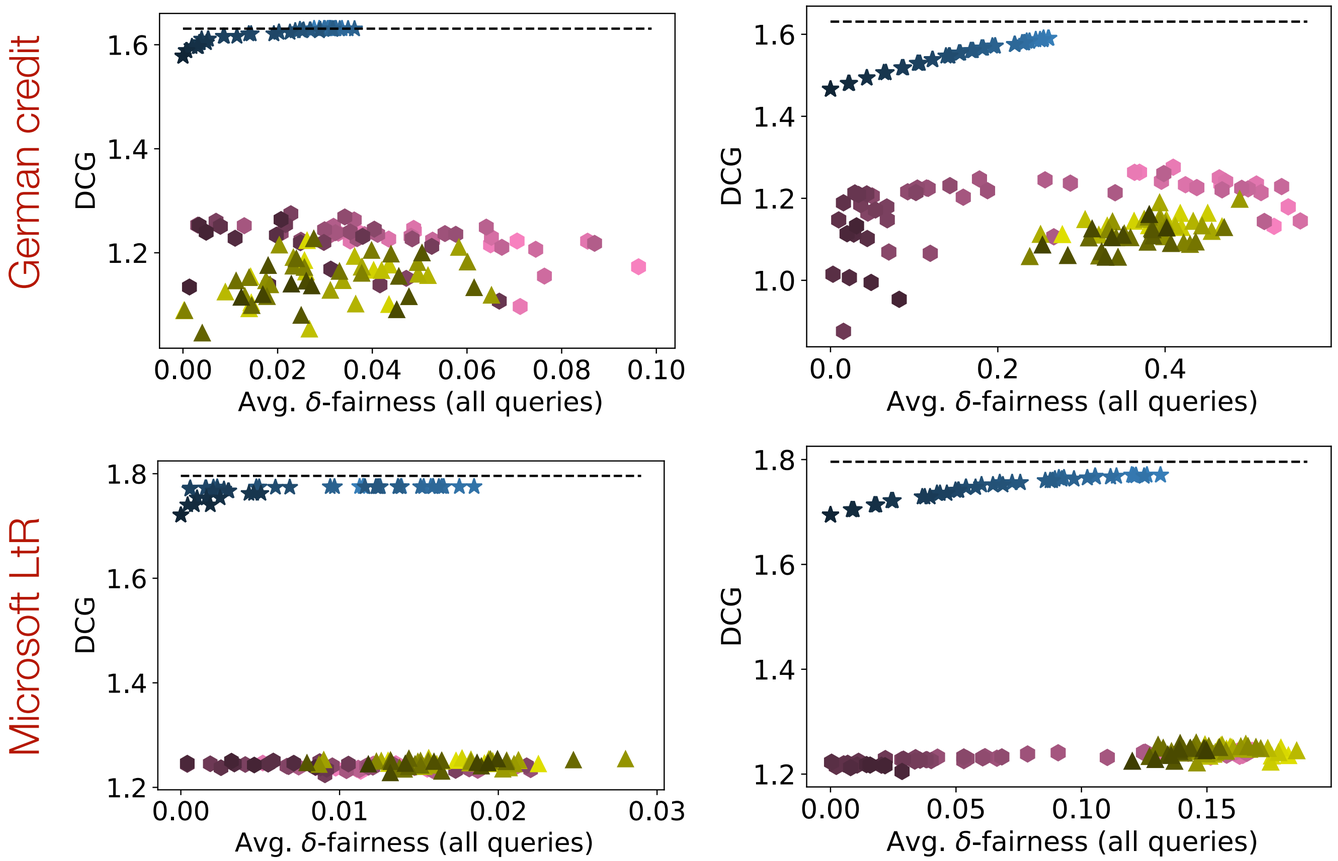


Summary: This integration provides query-level fairness guarantees! (δ-fairness).

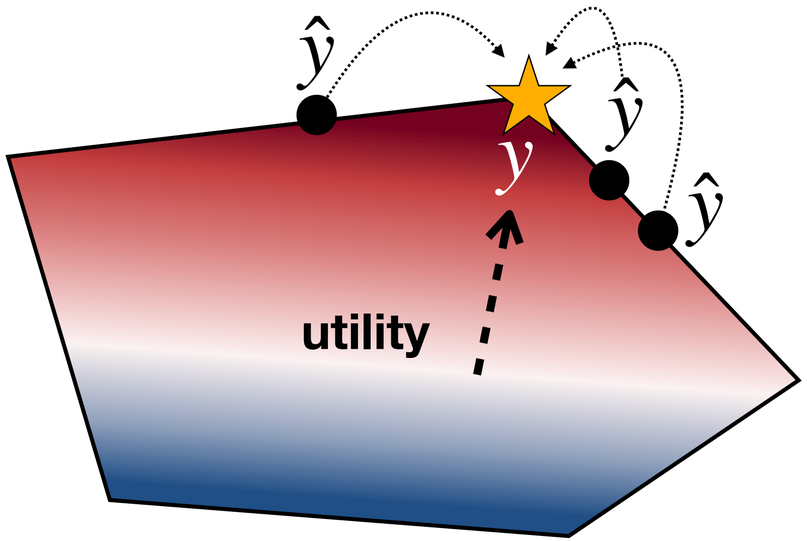
It attains much higher utilities as the learning task learns to maximize utility within the feasible subspace.

Why does it work?

Properties of end-to-end LPs



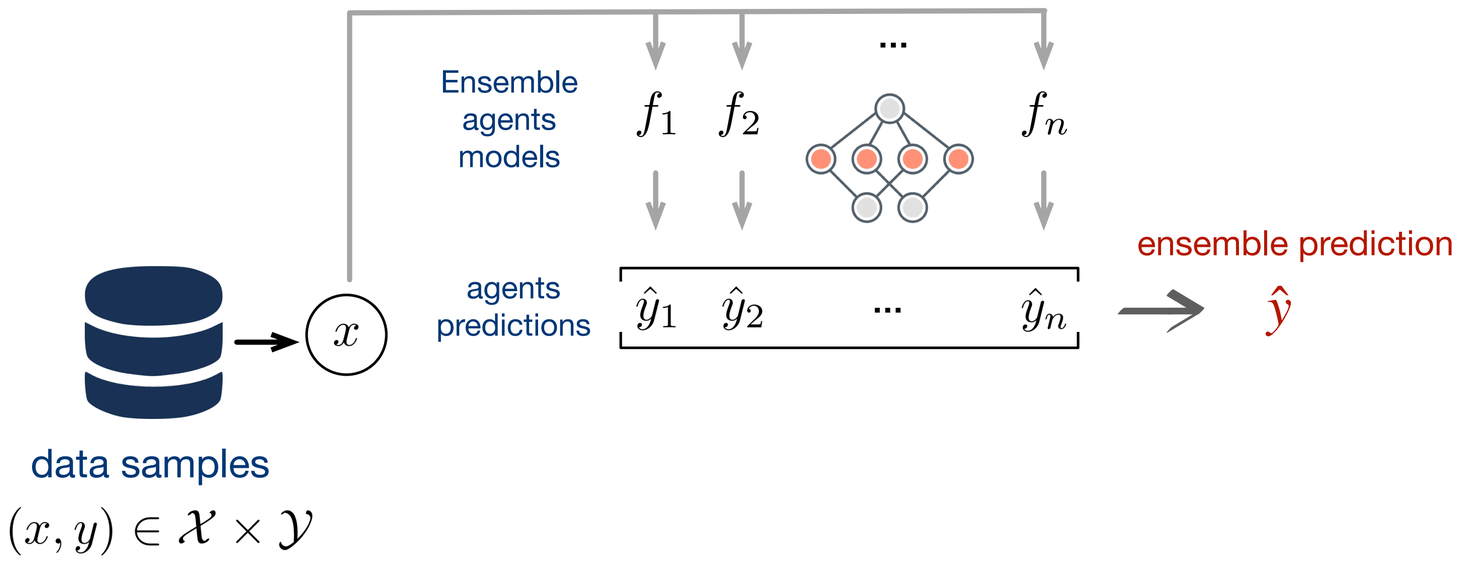
Theoretical insights: There is not a single point but a region of $\hat{y}$ which are regret-optimal under y.



Item scores need not be predicted precisely to find model parameters that maximize the empirical utility under fairness constraints.

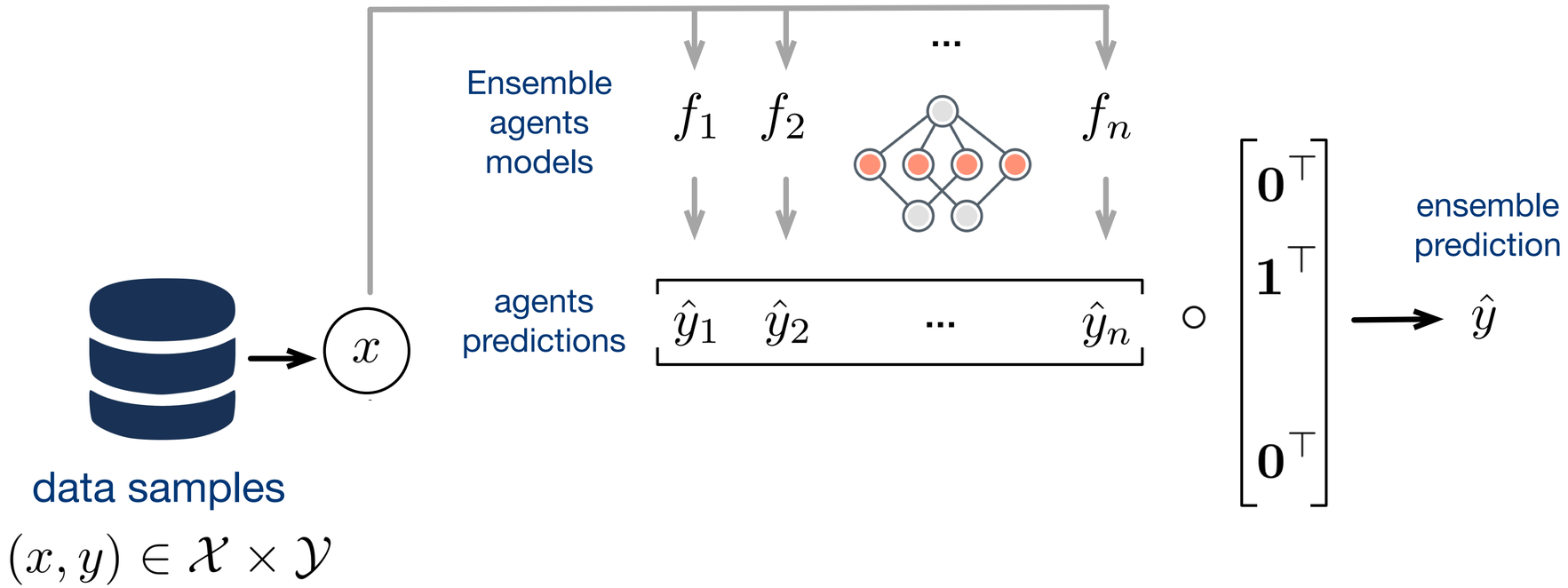
End-to-end combinatorial ensemble learning

- Ensemble learning aims at creating accurate and robust ML models by combining predictions from individual models.
- Key challenge: to find effective ways to combine the individual predictors for any particular input sample.



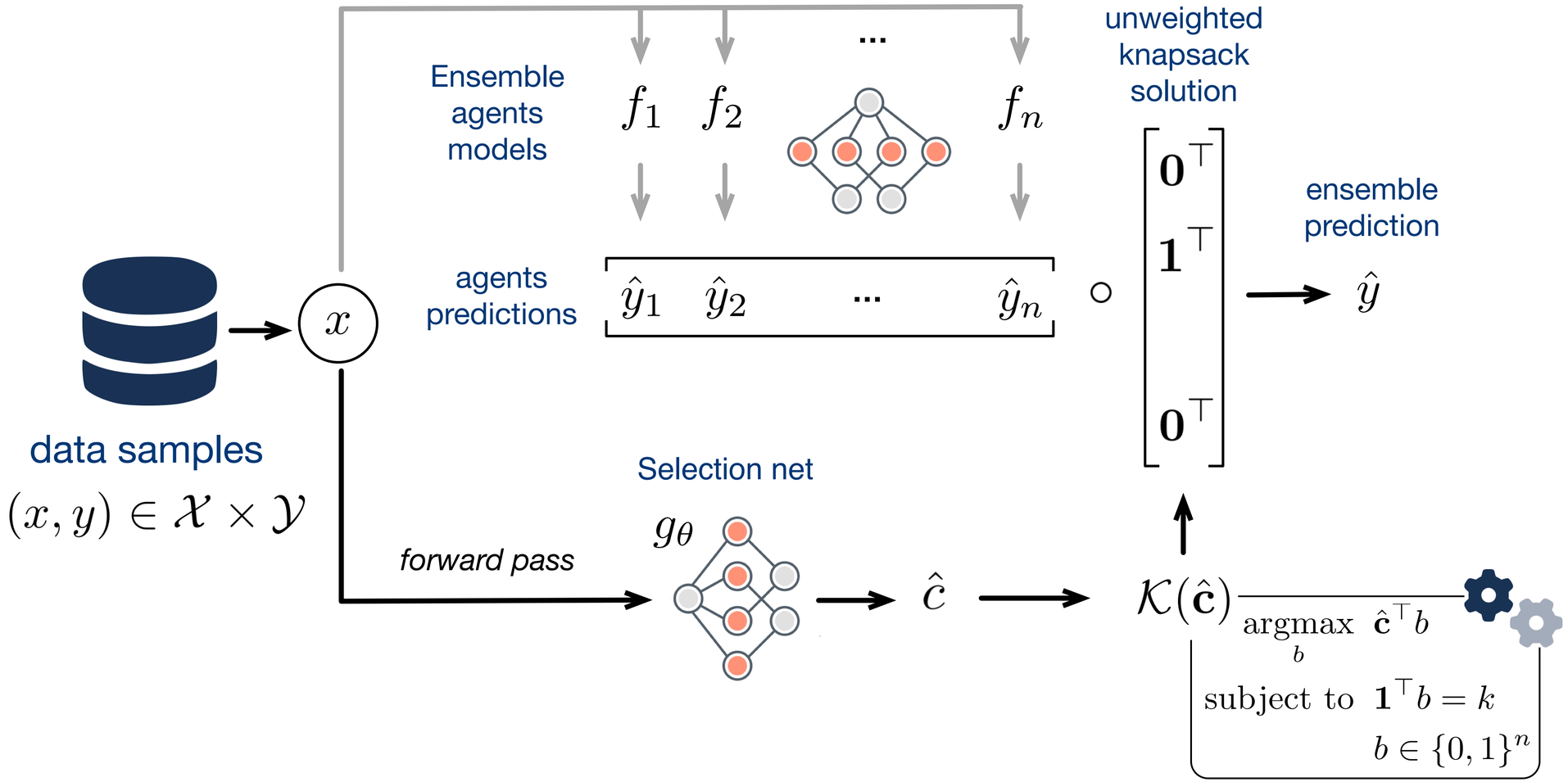
End-to-end combinatorial ensemble learning

Knapsack layers for specialized ensembles



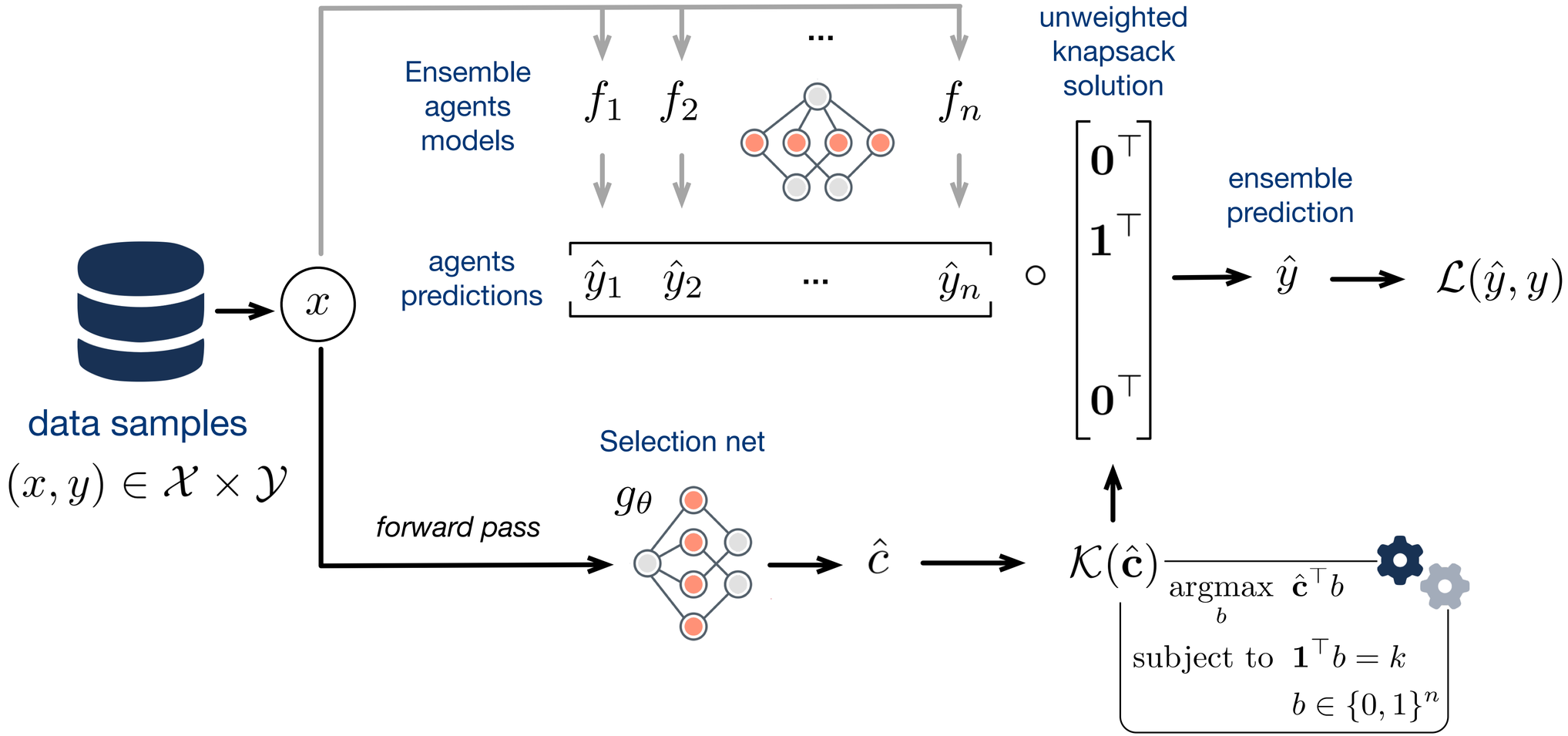
End-to-end combinatorial ensemble learning

Knapsack layers for specialized ensembles



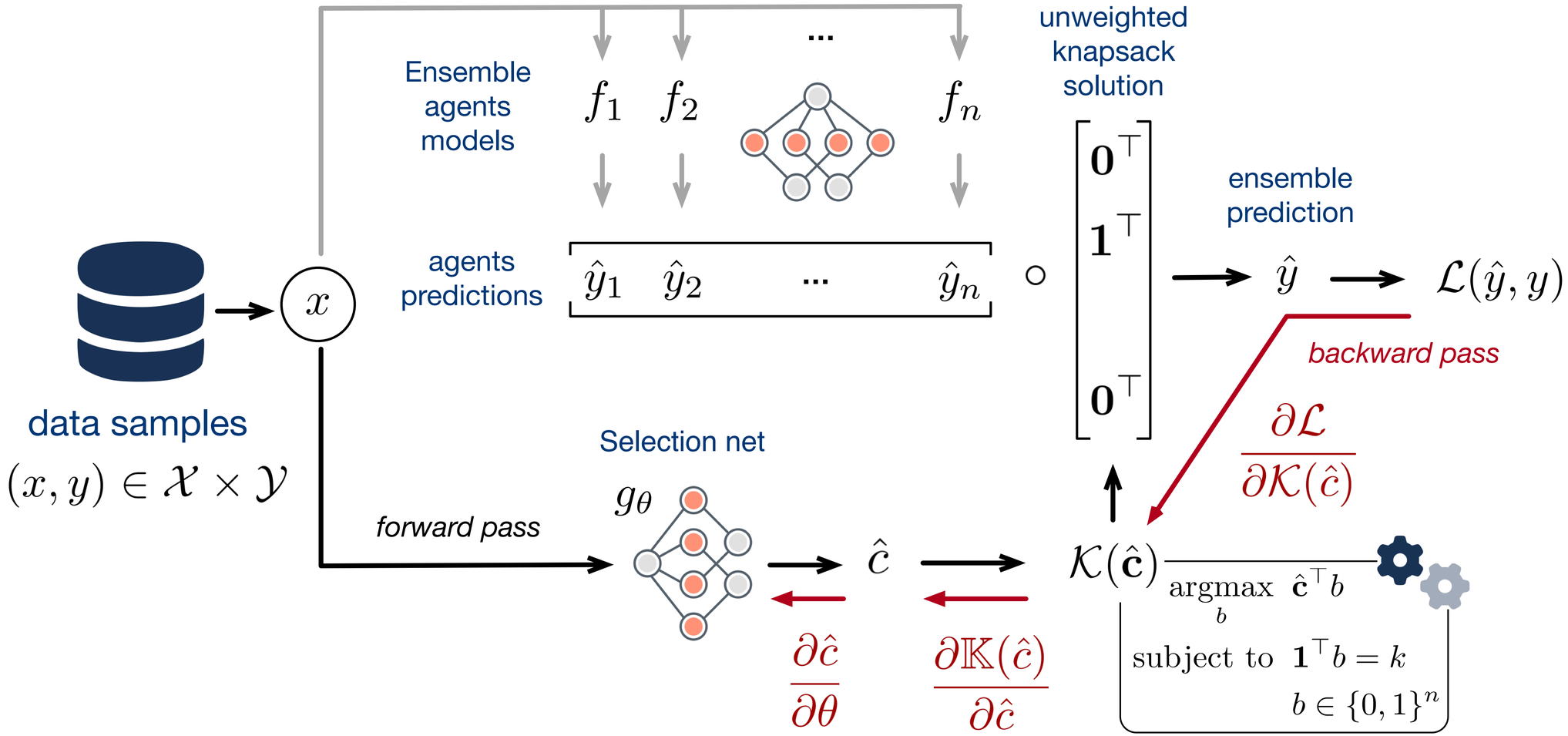
End-to-end combinatorial ensemble learning

Knapsack layers for specialized ensembles



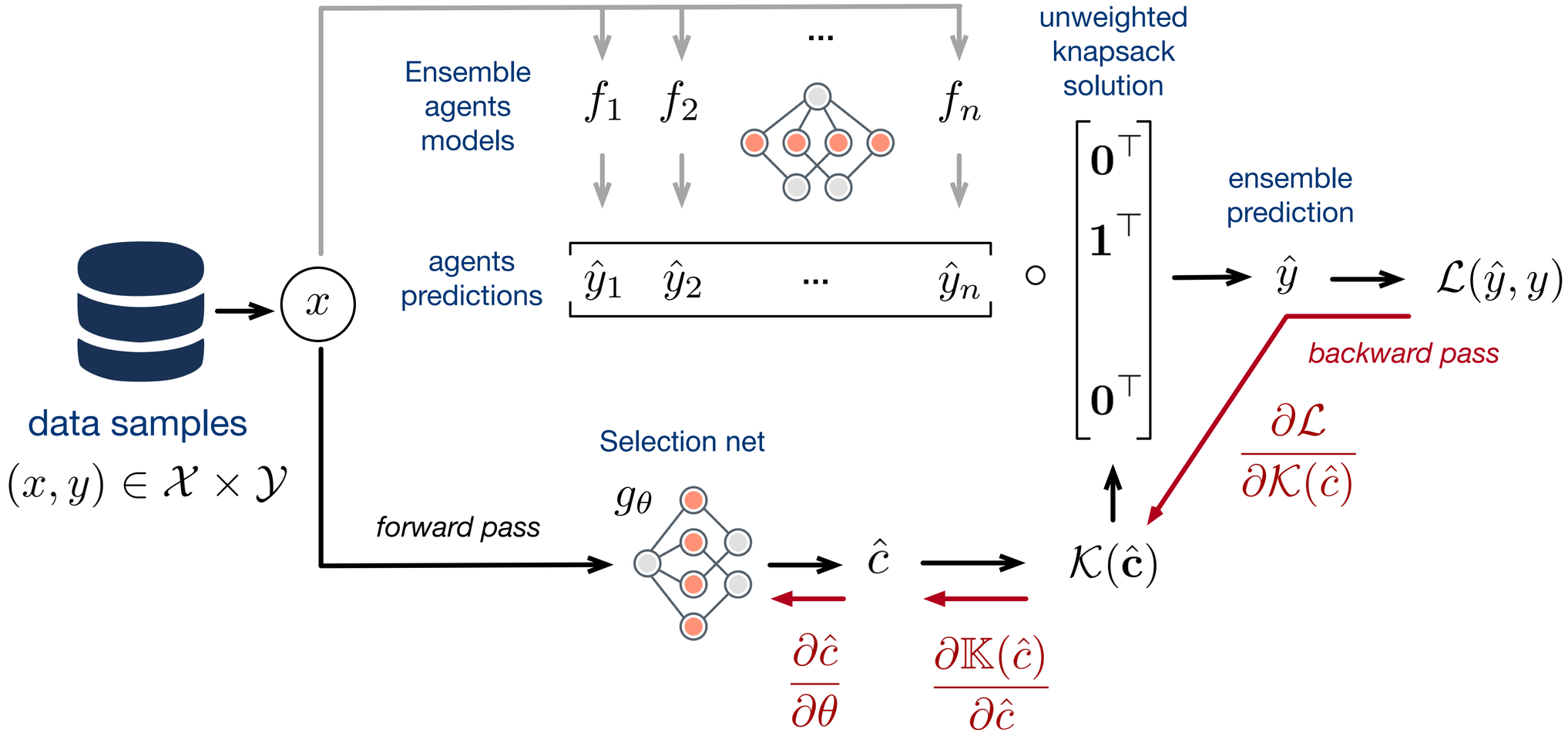
End-to-end combinatorial ensemble learning

Knapsack layers for specialized ensembles



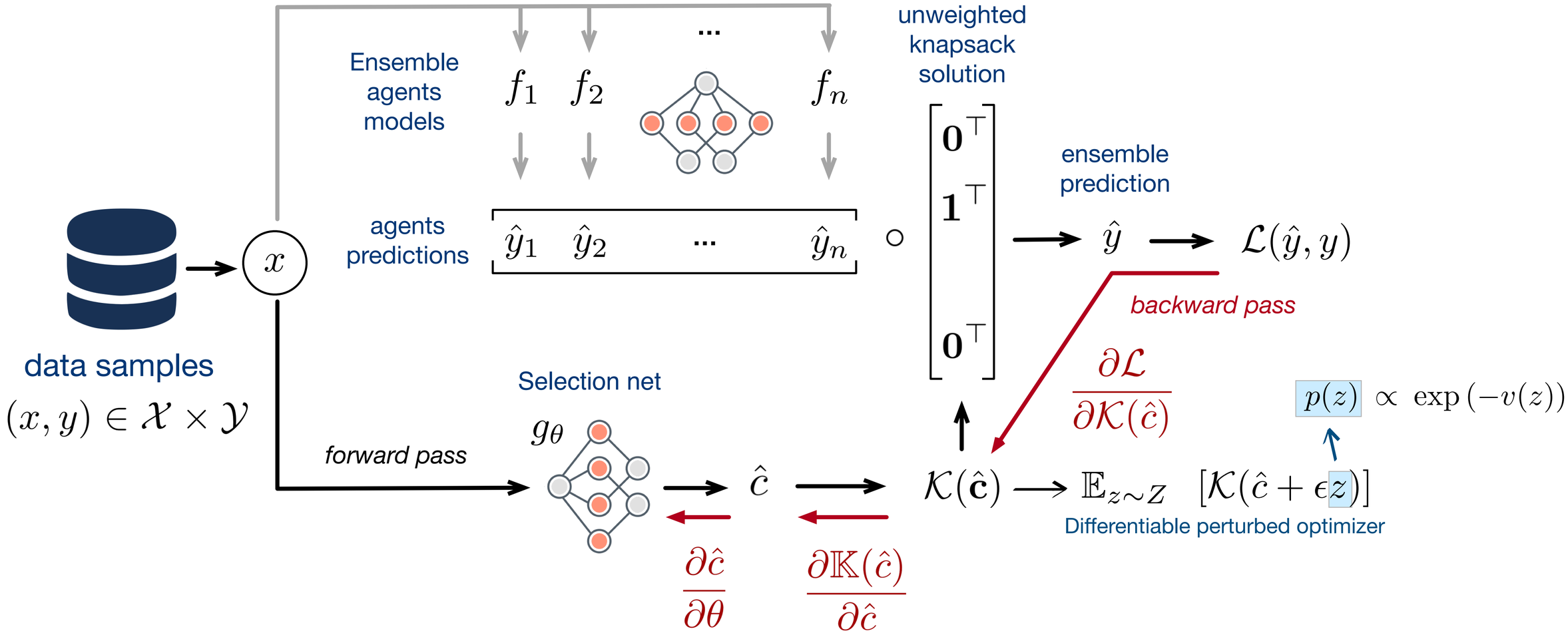
End-to-end combinatorial ensemble learning

Knapsack layers for specialized ensembles



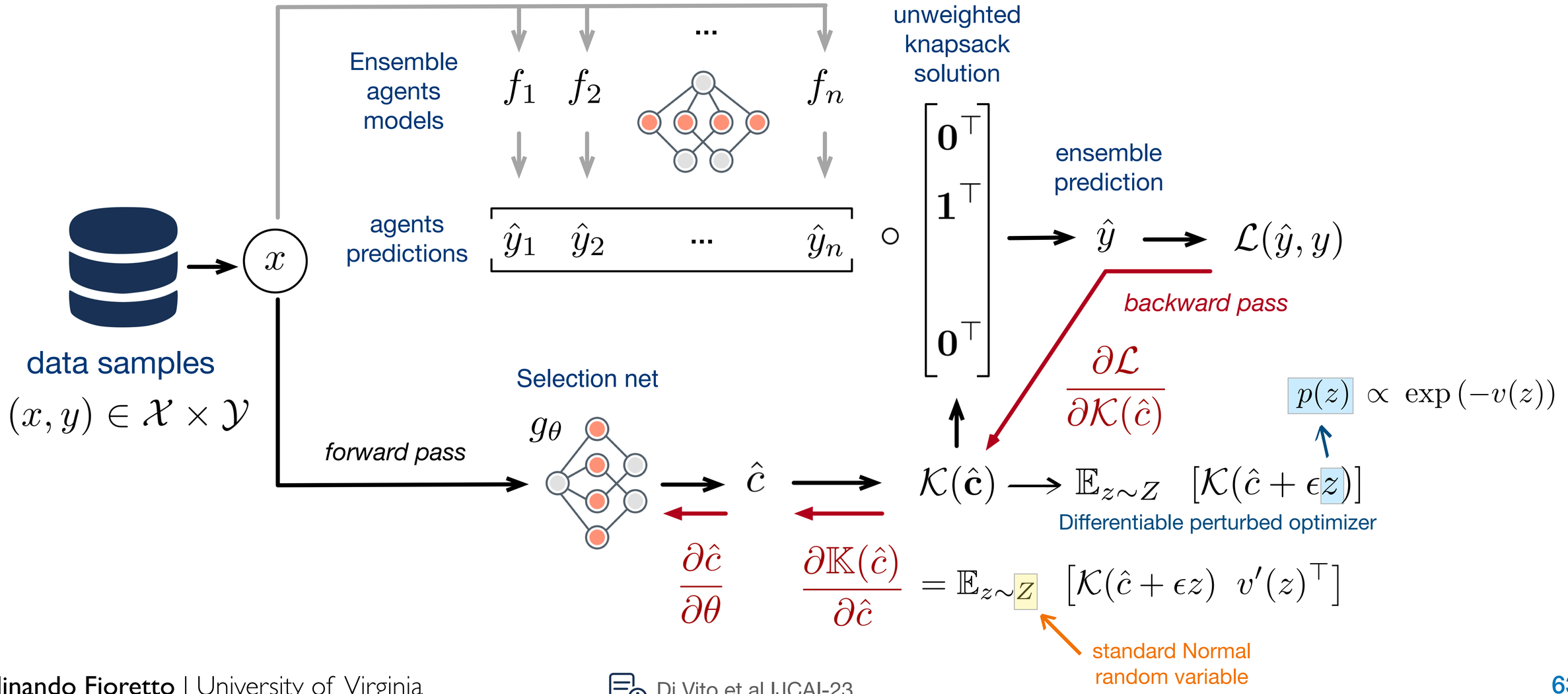
End-to-end combinatorial ensemble learning

Knapsack layers for specialized ensembles



End-to-end combinatorial ensemble learning

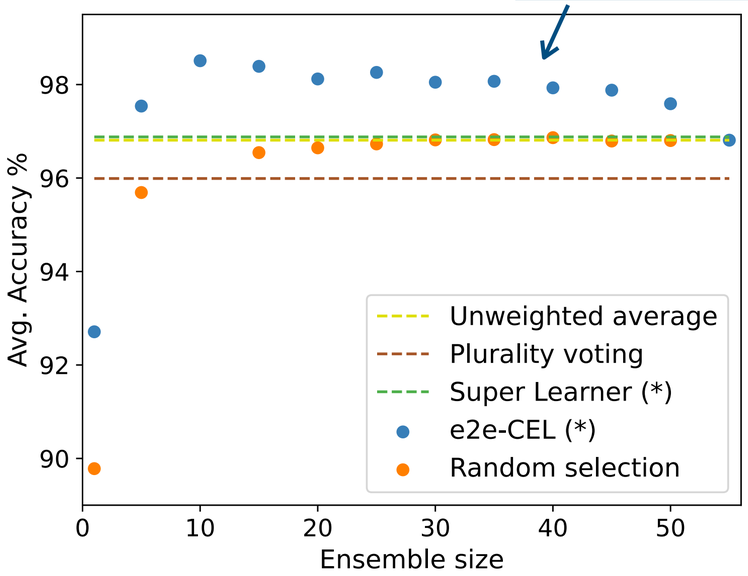
Knapsack layers for specialized ensembles

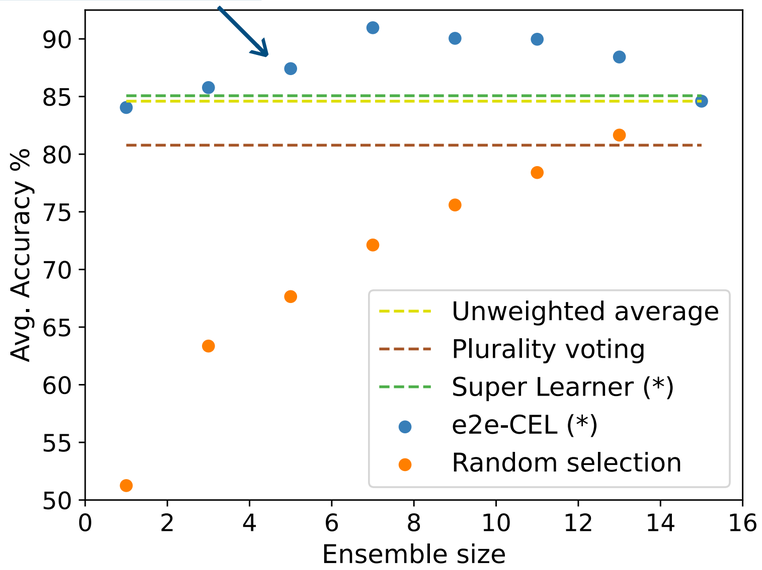


End-to-end combinatorial ensemble learning

Knapsack layers for specialized ensembles

constrained-aware ML



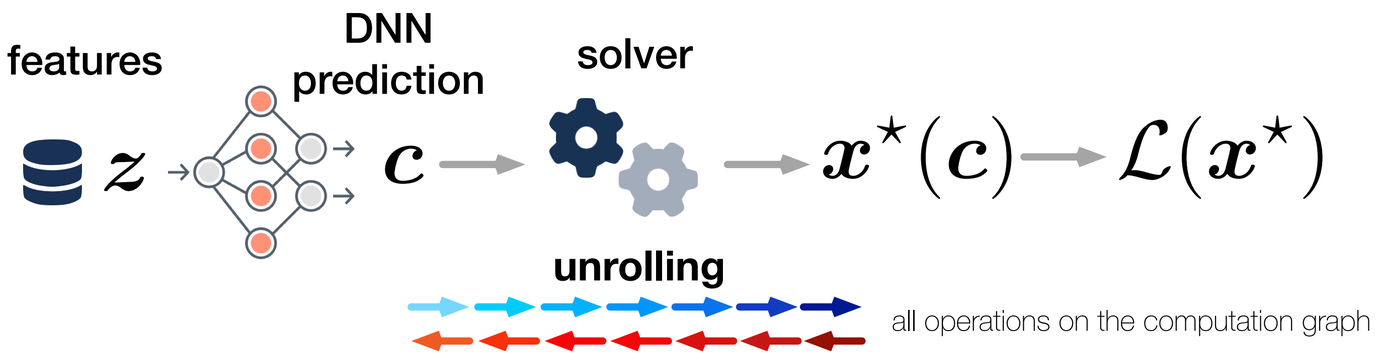


Summary: Selecting embedding through an end-to-end integration of a combinatorial optimization problem in the ML model brings large accuracy gains.

The learning task learns to maximize utility associated with each sample independently

	Accuracy (%)				
Dataset	**e2e-CEL**	**SL**	**UA**	**PV**	**RS**
MNIST	**98.55**	96.88	96.81	95.99	96.83
UTKFACE	**90.97**	85.07	84.60	80.78	84.60
FER2013	**66.31**	64.95	63.89	63.15	63.89
CIFAR10	**64.09**	60.13	60.59	60.35	60.59

Unrolling



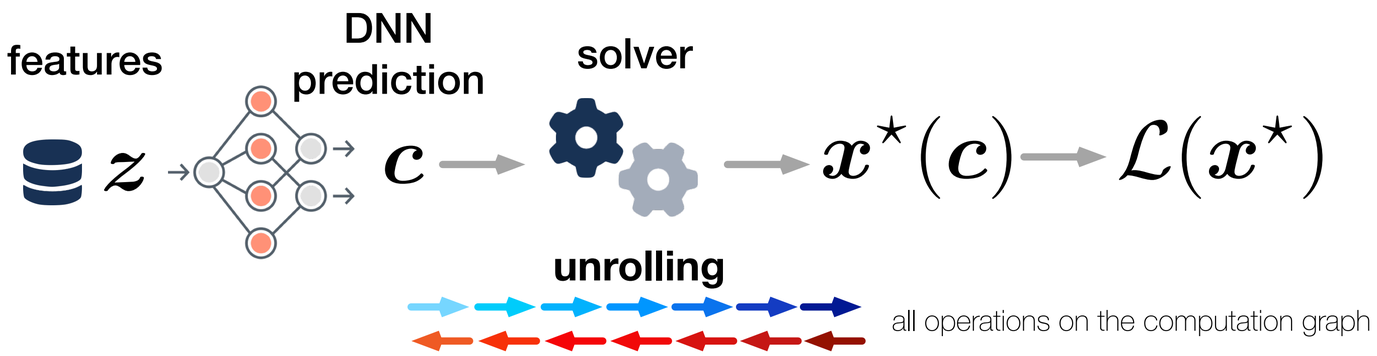
- Iterative optimizer which refines an initial starting point x_0 by applying an *update routine*

$$\mathbf{x}_{k+1}(\mathbf{c}) = \mathcal{U}(\mathbf{x}_k(\mathbf{c}),\ \mathbf{c})$$

which converges if $\mathbf{x}_k(\mathbf{c}) \to \mathbf{x}^\star(\mathbf{c})$ *as* $k \to \infty$

Kotary and Fioretto:IJCAI:23

Unrolling



- Iterative optimizer which refines an initial starting point x_0 by applying an *update routine*

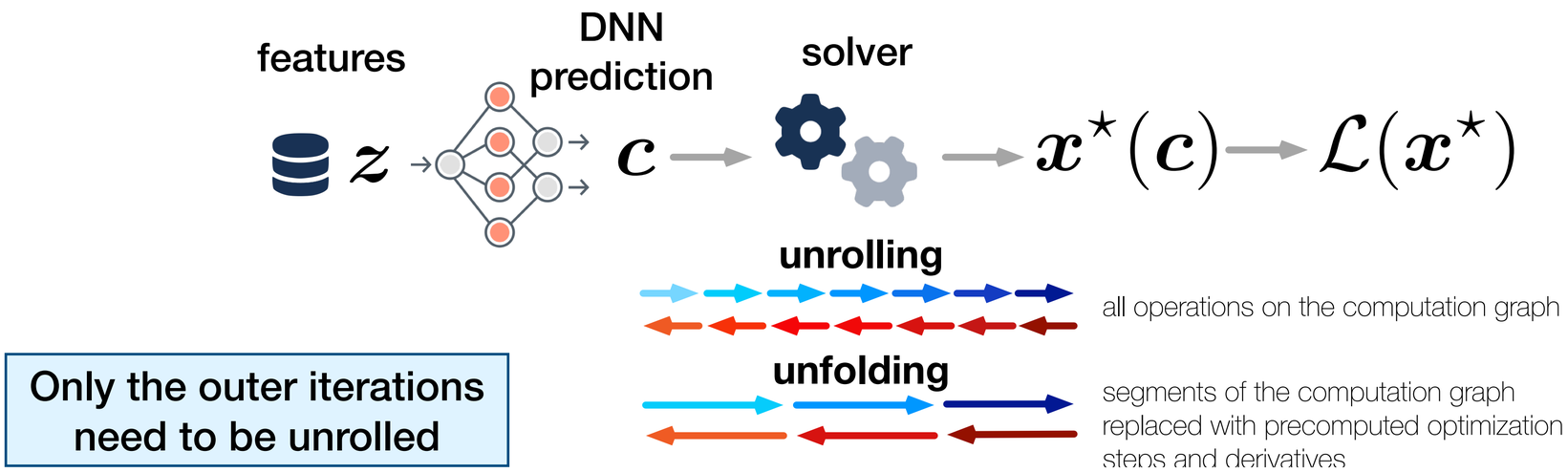
$$\mathbf{x}_{k+1}(\mathbf{c}) = \mathcal{U}(\mathbf{x}_k(\mathbf{c}),\ \mathbf{c})$$

which converges if $\mathbf{x}_k(\mathbf{c}) \to \mathbf{x}^\star(\mathbf{c})$ *as* $k \to \infty$

- *If the Jacobians of* $\mathcal{U}$ *can be computed analytically, we can avoid unrolling each eval. of* $\mathcal{U}$ *by modeling its backward pass with a Jacobian-vector product.*

Kotary and Fioretto:IJCAI:23

Unfolding



- Iterative optimizer which refines an initial starting point x_0 by applying an *update routine*

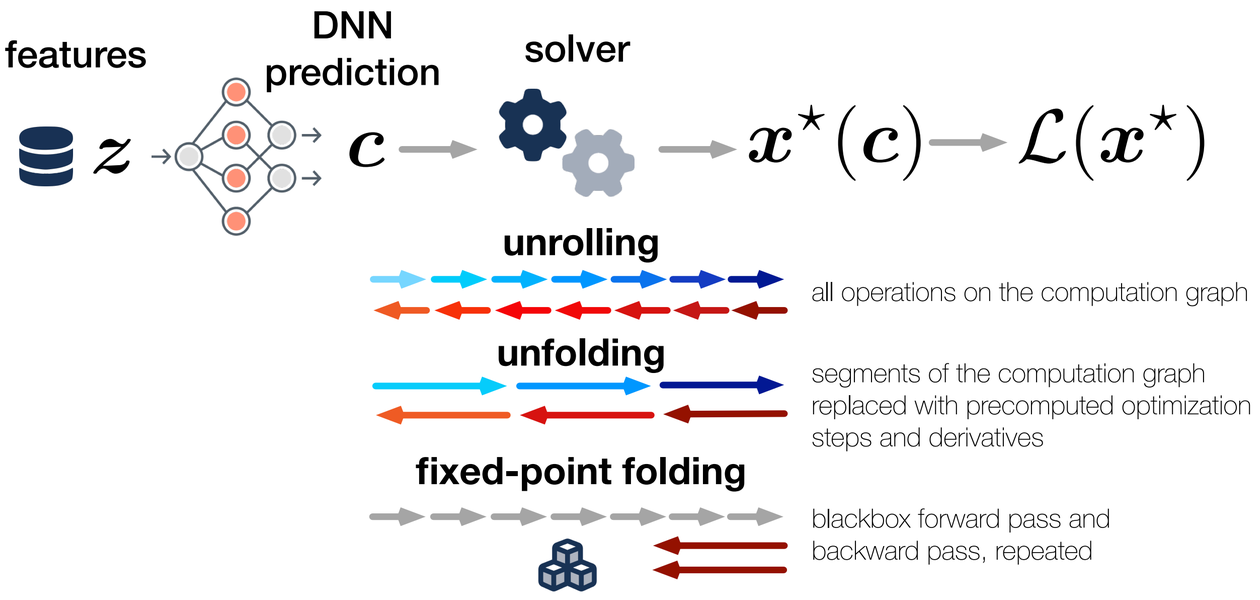
$$\mathbf{x}_{k+1}(\mathbf{c}) = \mathcal{U}(\mathbf{x}_k(\mathbf{c}),\ \mathbf{c})$$

which converges if $\mathbf{x}_k(\mathbf{c}) \to \mathbf{x}^\star(\mathbf{c})$ *as* $k \to \infty$

- *If the Jacobians of* $\mathcal{U}$ *can be computed analytically, we can avoid unrolling each eval. of* $\mathcal{U}$ *by modeling its backward pass with a Jacobian-vector product.*

Unfolding at a fixed point

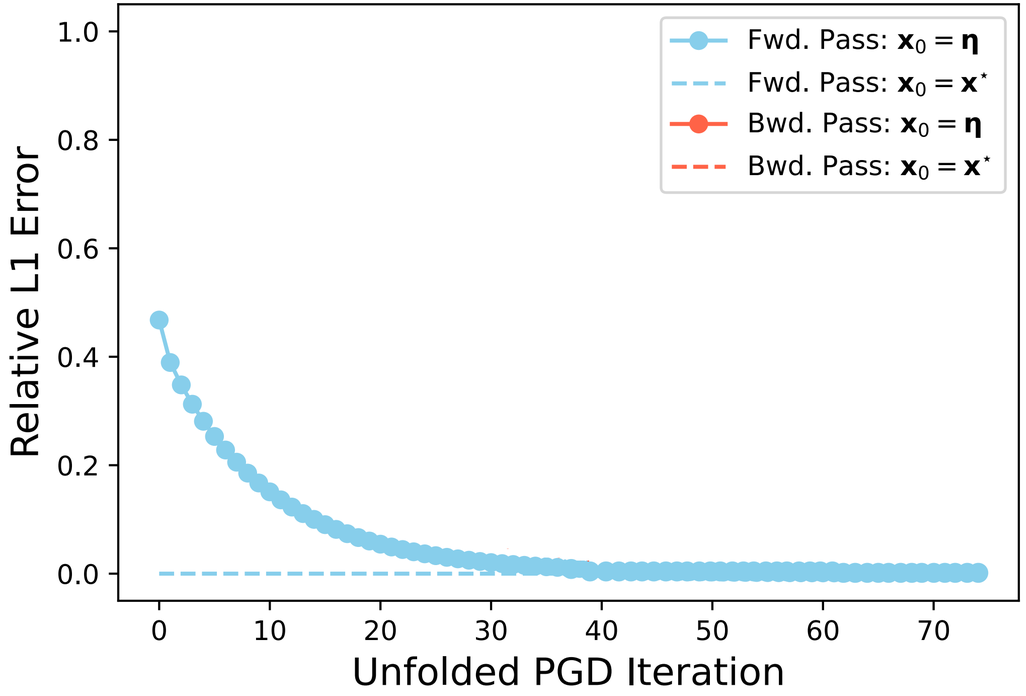
- We can unfold all the forward pass!



Kotary and Fioretto:IJCAI:23

Unfolding at a fixed point

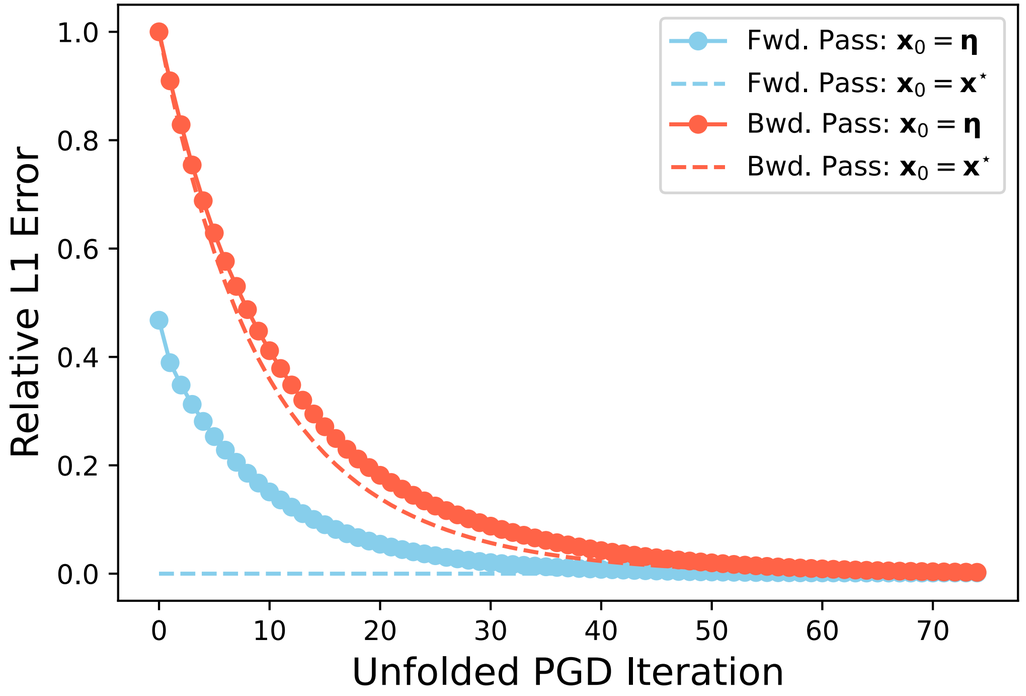
- We can unfold all the forward pass!
- Optimizing an iterative procedure, generally requires a starting point x_0.
- Convergence of the forward pass is guaranteed regardless of the nature of x_0. How about the backward pass?



Kotary and Fioretto:IJCAI:23

Unfolding at a fixed point

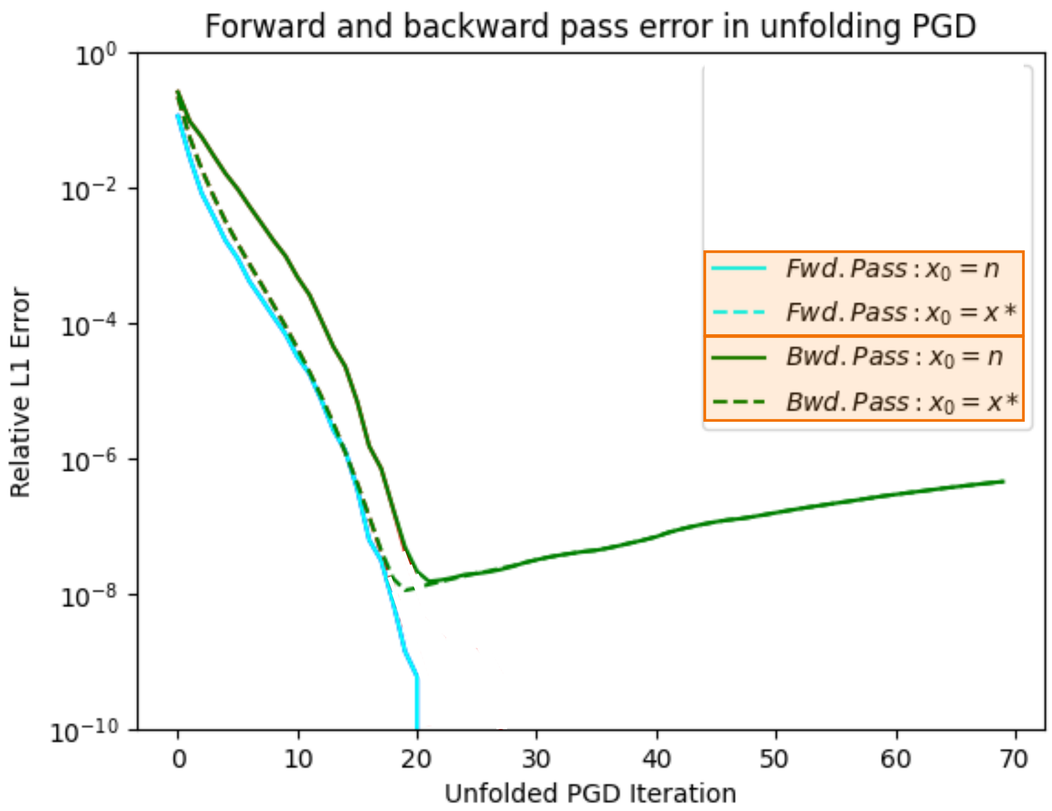
- We can unfold all the forward pass!
- Optimizing an iterative procedure, generally requires a starting point x_0.
- Convergence of the forward pass is guaranteed regardless of the nature of x_0. How about the backward pass?
- When an unfolded optimizer is iterated at a precomputed optimal solution $x_0 = x^\star$, the backward pass converges, but at its own rate.



Kotary and Fioretto:IJCAI:23

Unfolding at a fixed point

- We can unfold all the forward pass!
- Optimizing an iterative procedure, generally requires a starting point x_0.
- Convergence of the forward pass is guaranteed regardless of the nature of x_0. How about the backward pass?
- When an unfolded optimizer is iterated at a precomputed optimal solution $x_0 = x^\star$, the backward pass converges, but at its own rate.



Unfolding at a fixed point

- The backward pass of a fixed point optimizer does need to be computed to produce gradients for backdrop.
- However, since the iterates x_k are all fixed, so is their associated Jacobian $\boxed{\frac{\partial \mathcal{U}(\mathbf{x}_k)}{\partial \mathbf{x}_k}} = \frac{\partial \mathcal{U}(\mathbf{x}^\star)}{\partial \mathbf{x}^\star}$ used to model the backward pass of the update function $\mathcal{U}$.
- Therefore the calculation of this Jacobian needs to be computed only once!

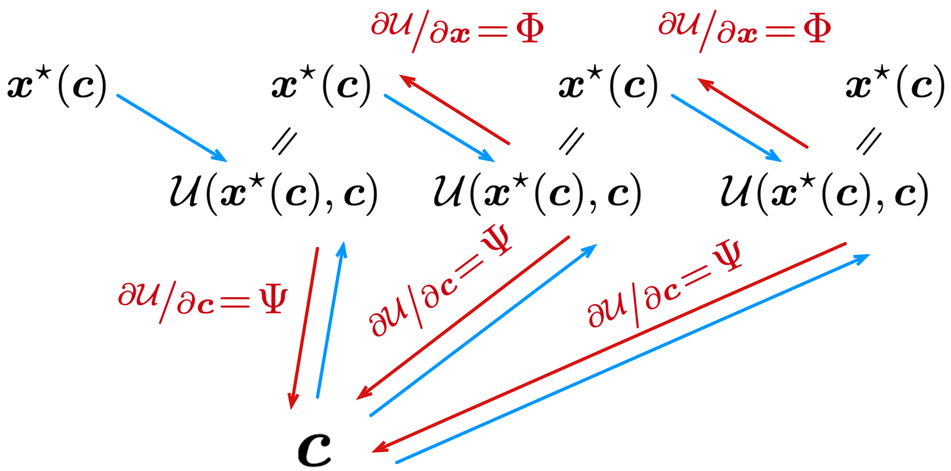
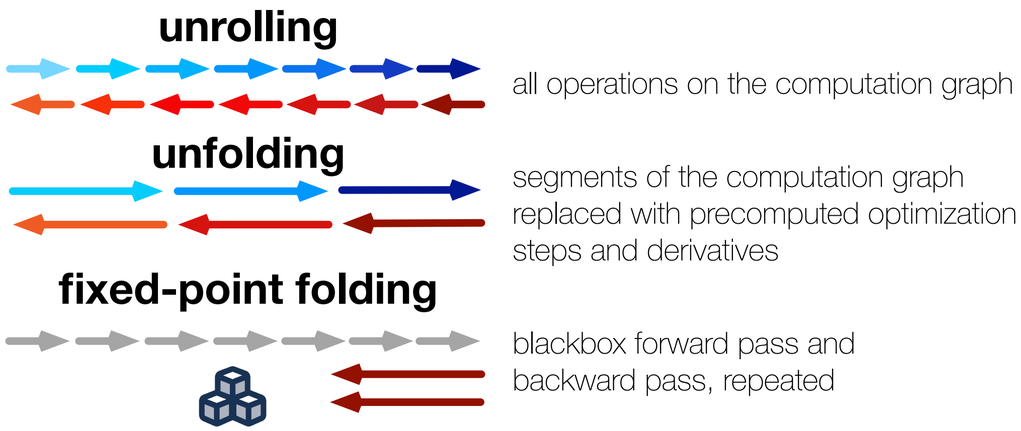


Figure 5: Computational graph for unfolding three iterations of (U) at a precomputed optimal solution $\mathbf{x}^\star$

- What remains is to iterate this backward pass until convergence to an accurate gradient of $x^*(c)$

Folded optimization

- Key Idea: Replace unfolding at a fixed-point $x^\star$ with the analytical solution of a linear system (see paper for more details).
- Unrolling is computationally equivalent to solving closed-form equations using a specific algorithm and does not require automatic differentiation.
- This results in a system for converting any unrolled implementation of an iterative optimizer into a folded optimizer that eliminates unrolling entirely.



Corollary 1. *Backpropagation of the fixed-point unfolding consists of the following rule:*

$$\mathbf{J}_0 = \mathbf{\Psi} \tag{18a}$$

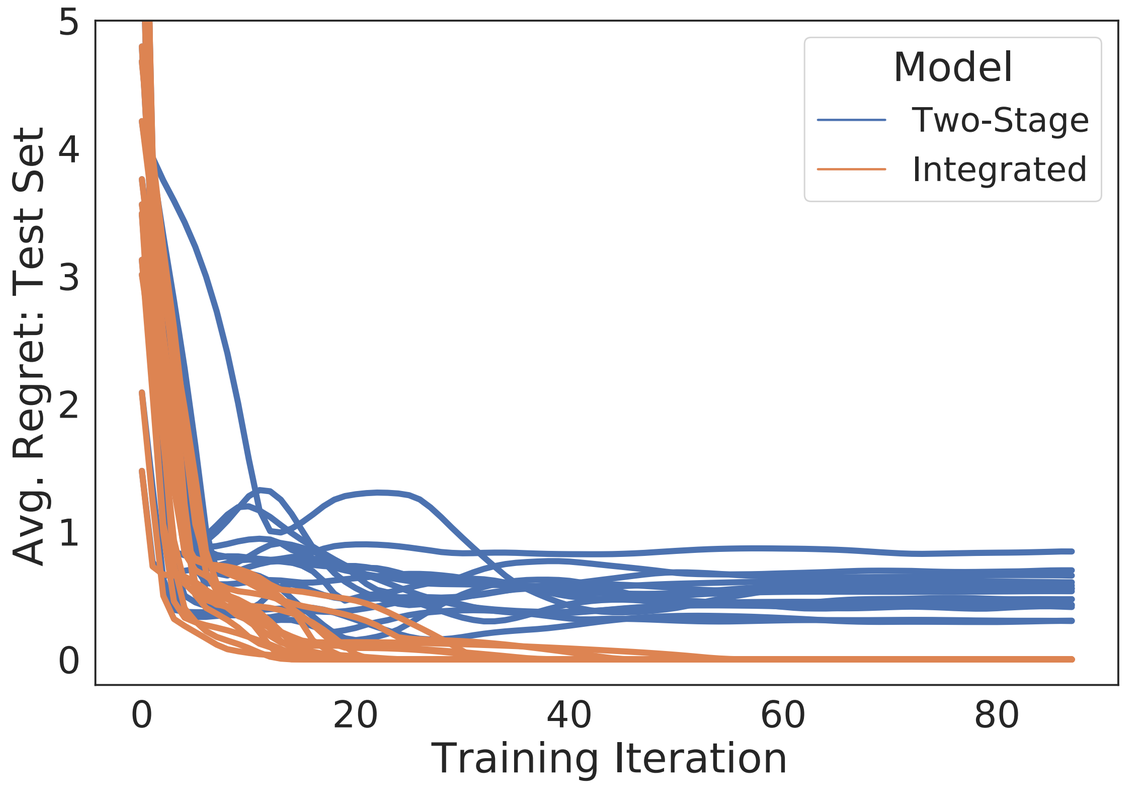
$$\mathbf{J}_{k+1} = \mathbf{\Phi}\mathbf{J}_k + \mathbf{\Psi}, \tag{18b}$$

where $\mathbf{J}_k := \frac{\partial \mathbf{x}_k}{\partial \mathbf{c}}(\mathbf{c})$.

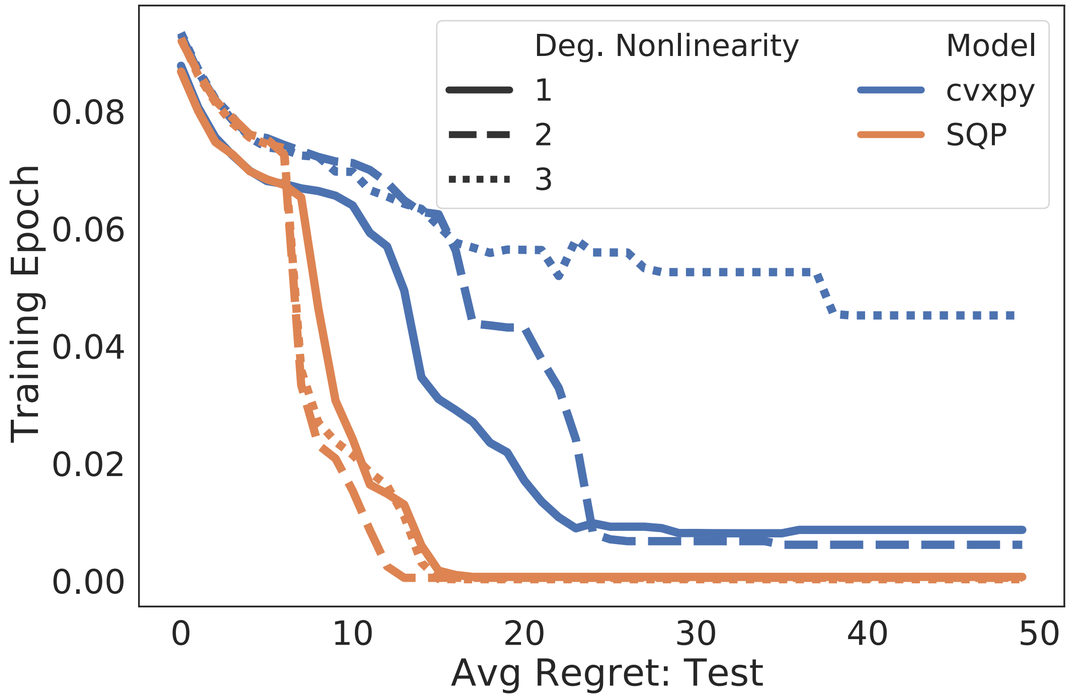
Folded optimization

Some results

Decision focused learning
with non-convex program



Portfolio optimization: Scalability



Up to 80% faster than cvxpy

Summary: Optimizers as Network Layers

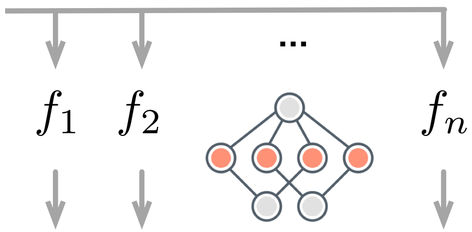
Enforcing structure in ML embeddings

Motivation: Need to enforce structure in Machine Learning embeddings.

Insight: Integrate optimizers within neural architectures as layers.

Settings:

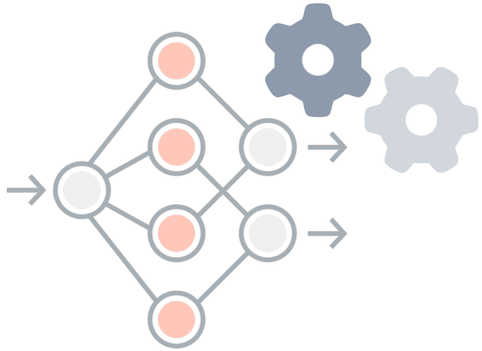
- Optimal power flows in energy systems [IJCAI-23]
- Fairness in ranking systems [WWW-22]
- Ensemble learning [AAMAS-23]
- Denoising [IJCAI-23]



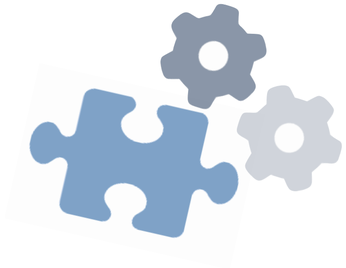
Agenda

Intro and Motivations

ML for Optimization



Optimization for ML



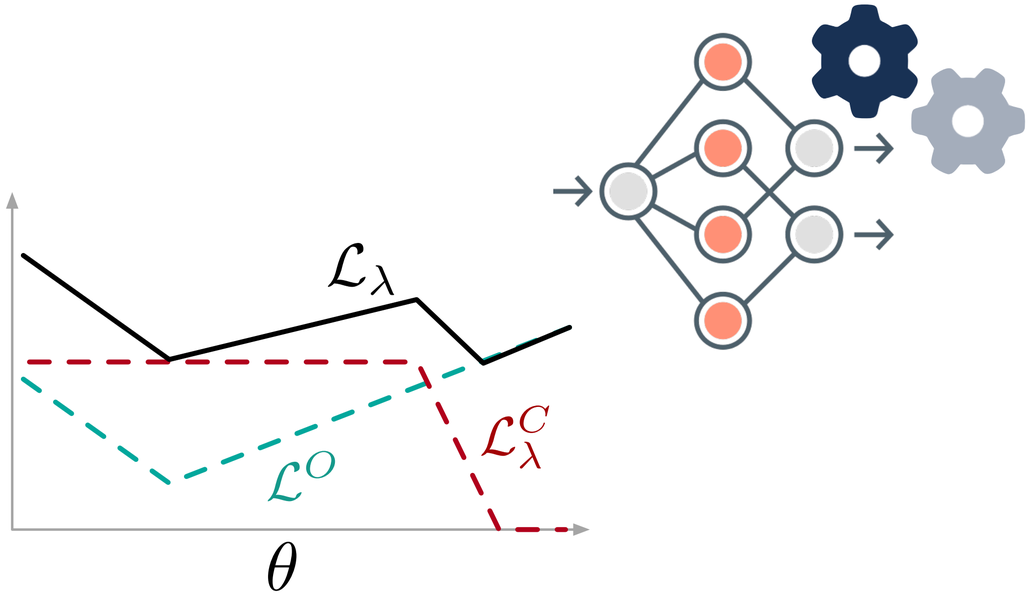
Open Questions

Knowledge transfer and generalization

Needs

Generalization: A critical shortcoming of current proxy optimizers is their inability to generalize beyond their data distribution.

- Can transfer learning on the dual space help to "warmstart" a different proxy optimizer?

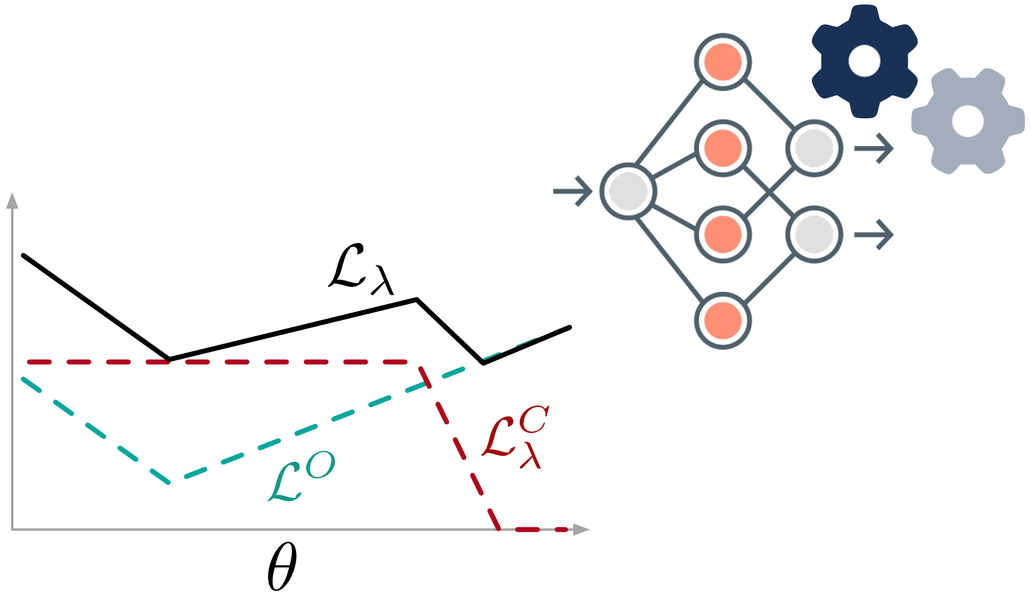


Knowledge transfer and generalization

Needs

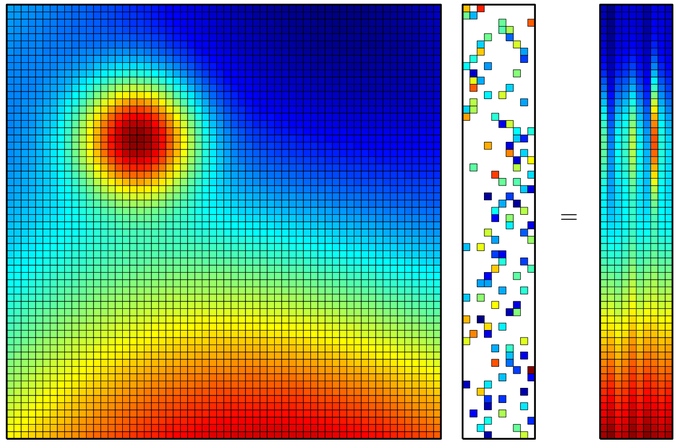
Generalization: A critical shortcoming of current proxy optimizers is their inability to generalize beyond their data distribution.

- Can transfer learning on the dual space help to "warmstart" a different proxy optimizer?



Data need: Proxy optimizers require training data which is expensive to attain.

- Can self-supervised approaches address this challenge?
- Can we exploit sketching and data summarization techniques?

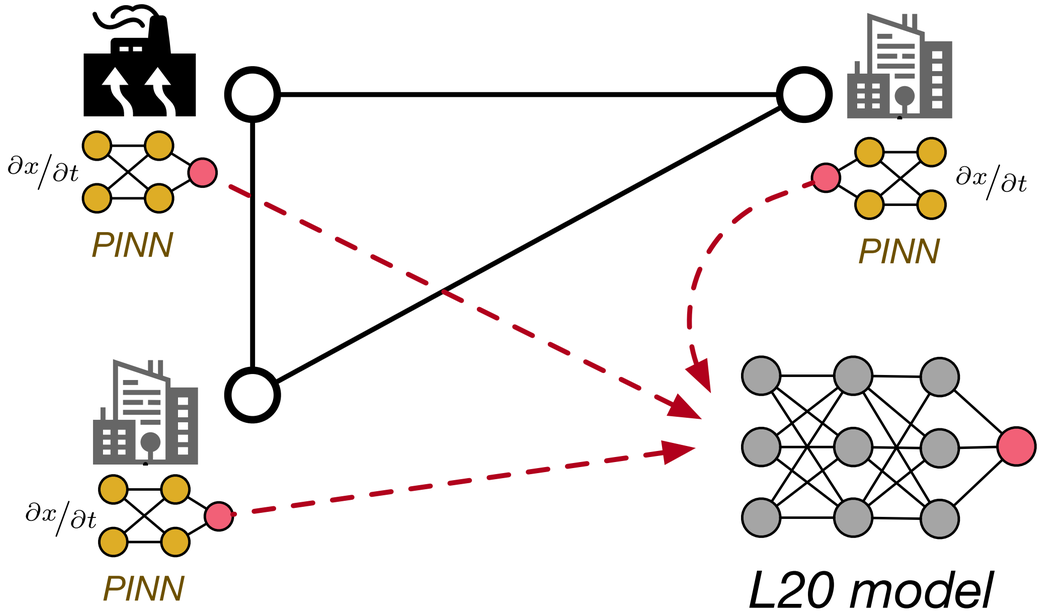


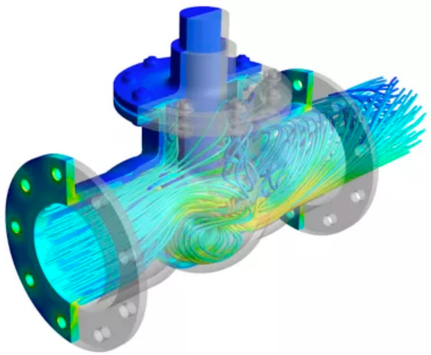
Scientific ML and digital twins

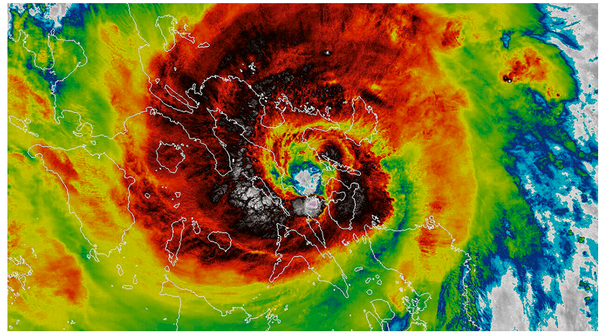
Needs

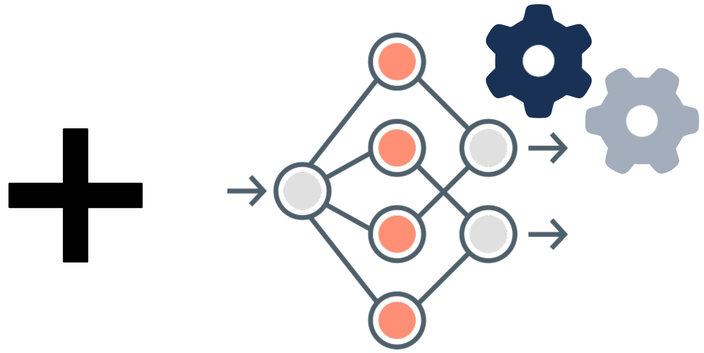
System dynamics: Many scientific and engineering systems are not static. The dynamic of these systems should be incorporated into the decision-making process.

- How to exploit proxy-optimizers and ML digital twins to produce solutions which are both efficient and can satisfy the system's dynamics?









Concluding remarks and discussion points

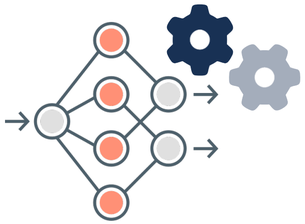
What are we missing?

- Despite the variety of approaches, the success of decision-focused learning has been demonstrated on a limited set of problems, mainly on LP formulations.
- Challenges posed by the parametrization of constraints stand in the way of broader applications.
- Issues associated with the runtime of solvers in-the-loop still make some potential applications impractical.
- Current surrogate optimization methods cannot reliably guarantee the problem constraints to be satisfied.
- Uniform benchmarks for systematic comparison are needed.

Conclusions

Constraint-aware Machine Learning

Motivated by the need for Scientific and Engineering applications requiring to solve many optimization problems at increasingly fast rates and large scales.



Proxy-optimizers as tools to obtain fast, approximate solutions to optimization problems.

Optimization-layers as effective tools to enhance the modeling ability and expressiveness of ML models.

The integration between constrained optimization and machine learning is an exciting and promising direction for the development of new, transformative, tools in constrained optimization and ML.

Constraint-aware Machine Learning

An end-to-end Integration of Constrained Optimization in ML

Thank you!

James Kotary

My Dinh

Cuong Tran

Vincenzo Di Vito

Jacob Christopher

Ferdinando Fioretto

https://nandofioretto.com

nandofioretto@gmail.com

@nandofioretto

References

Boyd, Stephen P., and Lieven Vandenberghe. “Convex optimization”. Cambridge university press, 2004.

Rossi, Francesca. "Principles of Constraint Programming” by Krzysztof R. Apt, Cambridge University Press, 2003.

MJD Powell "A method for non-linear constraints in minimization problems”. Optimization, 283-298

Wilder, Bryan, Bistra Dilkina, and Milind Tambe. "Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization." AAAI. 2019.

Amos, Brandon, and J. Zico Kolter. "Optnet: Differentiable optimization as a layer in neural networks." ICML, 2017.

Ferdinando Fioretto, Pascal Van Hentenryck, Terrence W.K. Mak, Cuong Tran, Federico Baldo, Michele Lombardi. "Lagrangian Duality for Constrained Deep Learning". ECML-PKDD, 2020.

Ferdinando Fioretto, Terrence W.K. Mak, Pascal Van Hentenryck. "Predicting AC Optimal Power Flows: Combining Deep Learning and Lagrangian Dual Methods". AAAI, 2020.

James Kotary, Ferdinando Fioretto, Pascal Van Hentenryck. "Learning Hard Optimization Problems: A Data Generation Perspective”. NeurIPS, 2021.

James Kotary, Ferdinando Fioretto, Pascal Van Hentenryck, Bryan Wilder."End-to-End Constrained Optimization Learning: A Survey”. IJCAI, 2021.

My H. Dinh, Ferdinando Fioretto, Mostafa Mohammadian, Kyri Baker. "Towards Understanding the Unreasonable Effectiveness of Learning AC-OPF Solutions”. CoRR abs/2111.11168 [cs.LG], 2021.

References

Ferdinando Fioretto. "Integrating Machine Learning and Optimization to Boost Decision Making". IJCAI, 2022.

James Kotary, Ferdinando Fioretto, Pascal Van Hentenryck, Ziwei Zhu. "End-to-end Learning for Fair Ranking Systems". WWW, 2022.

James Kotary, Ferdinando Fioretto, Pascal Van Hentenryck. "Fast Approximations for Job Shop Scheduling: A Lagrangian Dual Deep Learning Method". AAAI, 2022.

Mostafa Mohammadian, Kyri Baker, My H. Dinh, Ferdinando Fioretto. "Learning Solutions for Intertemporal Power Systems Optimization with Recurrent Neural Networks". Conference on Probabilistic Methods Applied to Power Systems (PMAPS), 2022.

Sawinder Kaur, Ferdinando Fioretto, Asif Salekin. "Deadwooding: Robust Global Pruning for Deep Neural Networks". CoRR abs/2202.05226 [cs.LG], 2022.

Mostafa Mohammadian, Kyri Baker, Ferdinando Fioretto. "Gradient-enhanced physics-informed neural networks for power systems operational support". Electric Power Systems Research, 2023.

James Kotary, My H. Dinh, Ferdinando Fioretto. "Folded Optimization for End-to-End Model-Based Learning". IJCAI, 2023.

James Kotary, Vincenzo Di Vito, Ferdinando Fioretto. "End-to-End Optimization and Learning for Multiagent Ensembles". IJCAI, 2023.

Terrence W.K. Mak, Ferdinando Fioretto, Pascal Van Hentenryck. "Load Encoding for Learning AC-OPF". IEEE PES General Meeting (PES), 2023.