Regularized Composite ReLU-ReHU Loss Minimization with Linear Computation and Linear Convergence

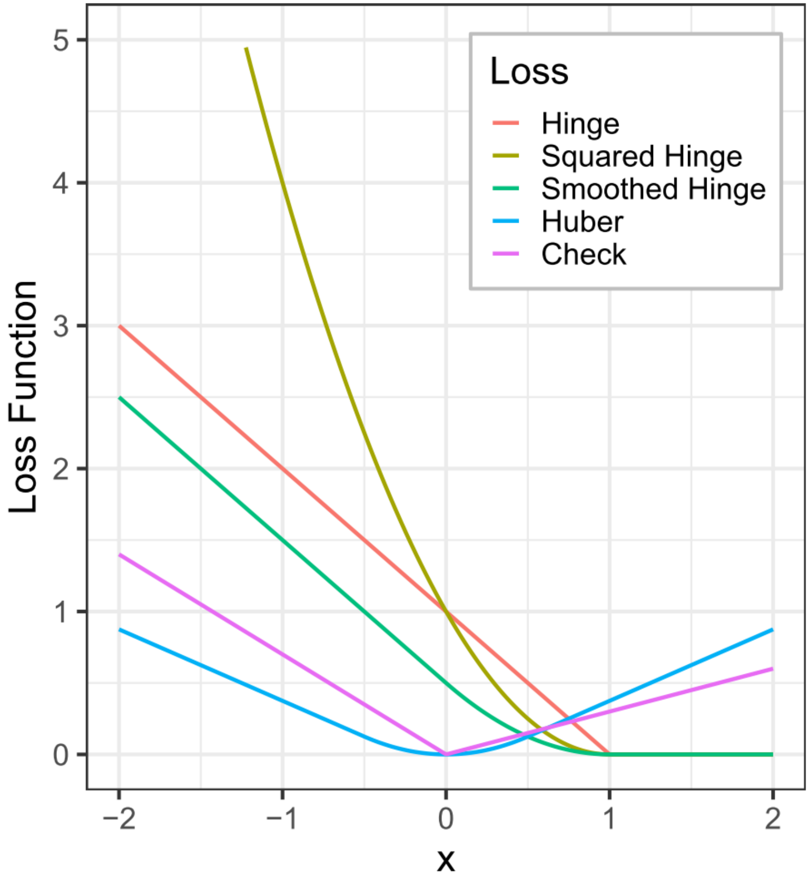
Ben Dai [1], **Yixuan Qiu** [2]

Equal Contribution

[1] Chinese University of Hong Kong (CUHK), [2] Shanghai University of Finance and Economics

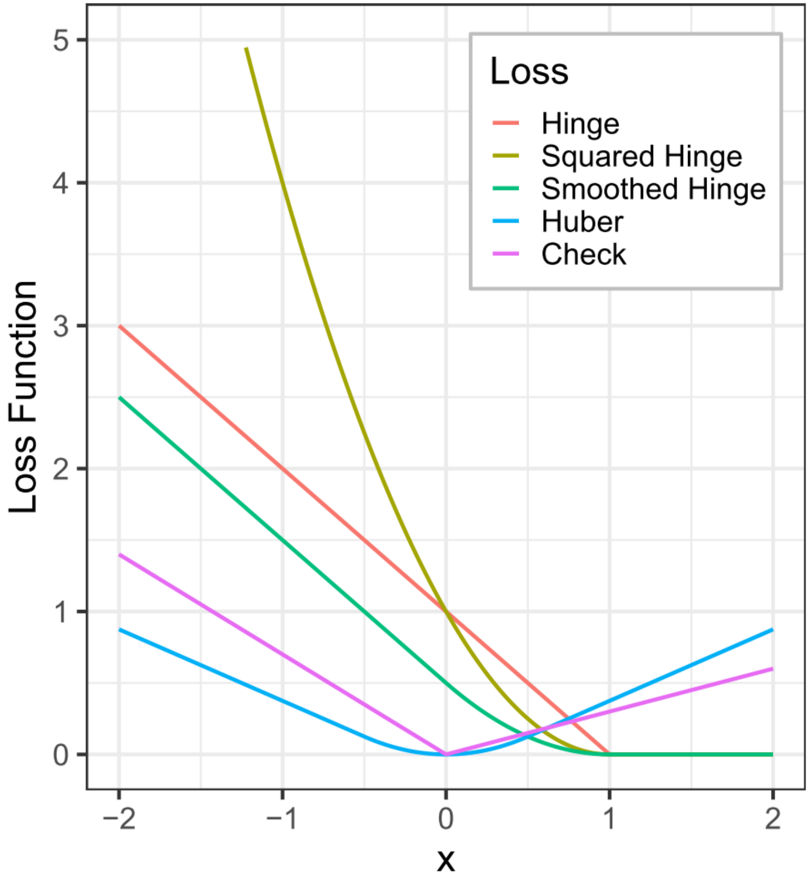
Motivation

- Empirical risk minimization (ERM) is a fundamental framework in machine learning
- Many different loss functions
- Efficient solvers exist for specific problems
- E.g., Liblinear for hinge loss SVM



Motivation

- Can we develop optimization algorithms for general ERM loss functions?
- Can we achieve provable fast convergence rates?
- Can we transfer the empirical success of Liblinear to general ERM problems?



Model

In this paper, we consider a general regularized ERM based on a **convex PLQ loss** with linear constraints:

$$\min_{\beta \in \mathbb{R}^d} \sum_{i=1}^{n} L_i(\mathbf{x}_i^\intercal \beta) + \frac{1}{2}\|\beta\|_2^2, \qquad \text{s.t. } \mathbf{A}\beta + \mathbf{b} \geq \mathbf{0},$$

- $L_i(\cdot) \geq 0$ is the proposed **composite ReLU-ReHU loss**.
- $\mathbf{x}_i \in \mathbb{R}^d$ is the feature vector for the i-th observation.
- $\mathbf{A} \in \mathbb{R}^{K \times d}$ and $\mathbf{b} \in \mathbb{R}^K$ are **linear inequality constraints** for β.
- We focus on working with a **large-scale** dataset, where the dimension of the coefficient vector and the total number of constraints are comparatively much smaller than the sample sizes, that is, $d \ll n$ and $K \ll n$.

Composite ReLU-ReHU Loss

Definition 1 (*Dai and Qiu. 2023*). A function $L(z)$ is composite ReLU-ReHU, if there exist $\mathbf{u}, \mathbf{v} \in \mathbb{R}^L$ and $\boldsymbol{\tau}, \mathbf{s}, \mathbf{t} \in \mathbb{R}^H$ such that

$$L(z) = \sum_{l=1}^{L} \mathrm{ReLU}(u_l z + v_l) + \sum_{h=1}^{H} \mathrm{ReHU}_{\tau_h}(s_h z + t_h)$$

where $\mathrm{ReLU}(z) = \max\{z, 0\}$, and $\mathrm{ReHU}_{\tau_h}(z)$ is defined below.

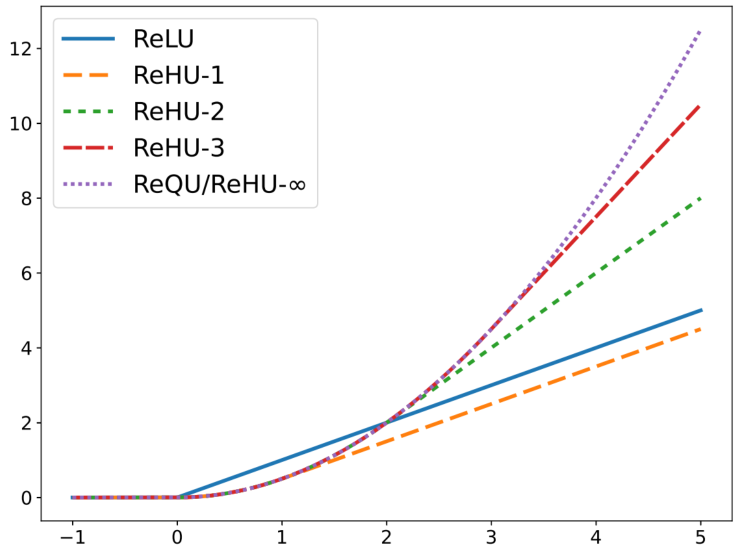
Composite ReLU-ReHU Loss

Definition 1 (*Dai and Qiu. 2023*). A function $L(z)$ is composite ReLU-ReHU, if there exist $\mathbf{u}, \mathbf{v} \in \mathbb{R}^L$ and $\tau, \mathbf{s}, \mathbf{t} \in \mathbb{R}^H$ such that

$$L(z) = \sum_{l=1}^{L} \mathrm{ReLU}(u_l z + v_l) + \sum_{h=1}^{H} \mathrm{ReHU}_{\tau_h}(s_h z + t_h)$$

where $\mathrm{ReLU}(z) = \max\{z, 0\}$, and $\mathrm{ReHU}_{\tau_h}(z)$ is defined below.

$$\mathrm{ReHU}_\tau(z) = \begin{cases} 0, & z \le 0 \\ z^2/2, & 0 < z \le \tau \\ \tau(z - \tau/2), & z > \tau \end{cases}$$



Composite ReLU-ReHU Loss

Theorem 1 (*Dai and Qiu. 2023*). A loss function $L : \mathbb{R} \to \mathbb{R}_{\geq 0}$ is **convex PLQ** *if and only if* it is **composite ReLU-ReHU**.

Table 2: Some widely used composite ReLU-ReHU losses as in (3). Here SVM is weighted SVMs based on the hinge loss [7], sSVM is smoothed SVMs based on the smoothed hinge loss [33], SVM^2 is weighted squared SVMs based on the squared hinge loss [7], LAD is the least absolute deviation regression, SVR is support vector regression with the ε-insensitive loss [44], and QR is quantile regression with the check loss [26].

Problem	Loss ($L_i(z_i)$)	Composite ReLU-ReHU Parameters
SVM	$c_i(1 - y_i z_i)_+$	$u_{1i} = -c_i y_i$, $v_{1i} = c_i$
sSVM	$c_i \text{ReHU}_1(-(y_i z_i - 1))$	$s_{1i} = -\sqrt{c_i} y_i, t_{1i} = \sqrt{c_i}$, $\tau = \sqrt{c_i}$
SVM^2	$c_i((1 - y_i z_i)_+)^2$	$s_{1i} = -\sqrt{2c_i} y_i$, $t_{1i} = \sqrt{2c_i}$, $\tau = \infty$
LAD	$c_i \lvert y_i - z_i \rvert$	$u_{1i} = c_i$, $v_{1i} = -c_i y_i$, $u_{2i} = -c_i$, $v_{2i} = c_i y_i$
SVR	$c_i(\lvert y_i - z_i \rvert - \varepsilon)_+$	$u_{1i} = c_i, v_{1i} = -(y_i + \varepsilon), u_{2i} = -c_i$, $v_{2i} = y_i - \varepsilon$
QR	$c_i \rho_\kappa(y_i - z_i)$	$u_{1i} = -c_i \kappa$, $v_{1i} = \kappa c_i y_i$, $u_{2i} = c_i(1 - \kappa)$, $v_{2i} = -c_i(1 - \kappa) y_i$

Composite ReLU-ReHU Loss

Table 2: Some widely used composite ReLU-ReHU losses as in (3). Here SVM is weighted SVMs based on the hinge loss [7], sSVM is smoothed SVMs based on the smoothed hinge loss [33], SVM^2 is weighted squared SVMs based on the squared hinge loss [7], LAD is the least absolute deviation regression, SVR is support vector regression with the ε-insensitive loss [44], and QR is quantile regression with the check loss [26].

PROBLEM	LOSS ($L_i(z_i)$)	COMPOSITE RELU-REHU PARAMETERS
SVM	$c_i(1 - y_i z_i)_+$	$u_{1i} = -c_i y_i,\ v_{1i} = c_i$
sSVM	$c_i \text{ReHU}_1(-(y_i z_i - 1))$	$s_{1i} = -\sqrt{c_i} y_i, t_{1i} = \sqrt{c_i},\ \tau = \sqrt{c_i}$
SVM^2	$c_i((1 - y_i z_i)_+)^2$	$s_{1i} = -\sqrt{2c_i} y_i,\ t_{1i} = \sqrt{2c_i},\ \tau = \infty$
LAD	$c_i \lvert y_i - z_i \rvert$	$u_{1i} = c_i,\ v_{1i} = -c_i y_i,\ u_{2i} = -c_i,\ v_{2i} = c_i y_i$
SVR	$c_i(\lvert y_i - z_i \rvert - \varepsilon)_+$	$u_{1i} = c_i, v_{1i} = -(y_i + \varepsilon), u_{2i} = -c_i,\ v_{2i} = y_i - \varepsilon$
QR	$c_i \rho_\kappa(y_i - z_i)$	$u_{1i} = -c_i \kappa,\ v_{1i} = \kappa c_i y_i,\ u_{2i} = c_i(1 - \kappa), v_{2i} = -c_i(1 - \kappa) y_i$

ReHLine applies to **any** convex piecewise linear-quadratic loss function (potential for non-smoothness included), including the hinge loss, the check loss, the Huber loss, etc.

Main Results

Table 1: Overview of existing algorithms for solving (1). Column COMPLEXITY (PER ITERATION) shows the computational complexity of the algorithm per iteration. Here, we focus only on the order of n since $d \ll n$ is assumed in our setting. Column #ITERATIONS shows the number of iterations needed to achieve an accuracy of ε to the minimizer.

ALGORITHM	COMPLEXITY (PER ITERATION)	#ITERATIONS	COMPLEXITY (TOTAL)
P-GD	$\mathcal{O}(n)$	$\mathcal{O}(\varepsilon^{-1})$ [6]	$\mathcal{O}(n\varepsilon^{-1})$
CD	$\mathcal{O}(n^2)$	$\mathcal{O}(\log(\varepsilon^{-1}))$ [31]	$\mathcal{O}(n^2 \log(\varepsilon^{-1}))$
IPM	$\mathcal{O}(n^2)$	$\mathcal{O}(\log(\varepsilon^{-1}))$ [18]	$\mathcal{O}(n^2 \log(\varepsilon^{-1}))$
ADMM	$\mathcal{O}(n^2)$	$o(\varepsilon^{-1})$ [9, 20]	$o(n^2\varepsilon^{-1})$
SDCA	$\mathcal{O}(n)$	$\mathcal{O}(\varepsilon^{-1})$ [39]	$\mathcal{O}(n\varepsilon^{-1})$
ReHLine (ours)	$\mathcal{O}(n)$	$\mathcal{O}(\log(\varepsilon^{-1}))$	$\mathcal{O}(n \log(\varepsilon^{-1}))$

ReHLine has a provable linear convergence rate. The per-iteration computational complexity is linear in the sample size.

ReHLine

- Inspired by **Coordinate Descent** (CD) and **Liblinear**

Theorem 2. *The Lagrangian dual problem of (6) is:*

$$(\widehat{\boldsymbol{\xi}}, \widehat{\boldsymbol{\Lambda}}, \widehat{\boldsymbol{\Gamma}}) = \underset{\boldsymbol{\xi}, \boldsymbol{\Lambda}, \boldsymbol{\Gamma}}{\operatorname{argmin}} \ \mathcal{L}(\boldsymbol{\xi}, \boldsymbol{\Lambda}, \boldsymbol{\Gamma}) \quad \text{s.t.} \ \boldsymbol{\xi} \geq \mathbf{0}, \quad \mathbf{E} \geq \boldsymbol{\Lambda} \geq \mathbf{0}, \quad \boldsymbol{\tau} \geq \boldsymbol{\Gamma} \geq \mathbf{0}, \tag{7}$$

$$\begin{aligned}\mathcal{L}(\boldsymbol{\xi}, \boldsymbol{\Lambda}, \boldsymbol{\Gamma}) = & \frac{1}{2}\boldsymbol{\xi}^\intercal \mathbf{A}\mathbf{A}^\intercal \boldsymbol{\xi} + \frac{1}{2} vec(\boldsymbol{\Lambda})^\intercal \bar{\mathbf{U}}_{(3)}^\intercal \bar{\mathbf{U}}_{(3)} vec(\boldsymbol{\Lambda}) + \frac{1}{2} vec(\boldsymbol{\Gamma})^\intercal (\bar{\mathbf{S}}_{(3)}^\intercal \bar{\mathbf{S}}_{(3)} + \mathbf{I}) vec(\boldsymbol{\Gamma}) \\ & - \boldsymbol{\xi}^\intercal \mathbf{A} \bar{\mathbf{U}}_{(3)} vec(\boldsymbol{\Lambda}) - \boldsymbol{\xi}^\intercal \mathbf{A} \bar{\mathbf{S}}_{(3)} vec(\boldsymbol{\Gamma}) + vec(\boldsymbol{\Lambda})^\intercal \bar{\mathbf{U}}_{(3)}^\intercal \bar{\mathbf{S}}_{(3)} vec(\boldsymbol{\Gamma}) \\ & + \boldsymbol{\xi}^\intercal \mathbf{b} - \operatorname{Tr}(\boldsymbol{\Lambda}\mathbf{V}^\intercal) - \operatorname{Tr}(\boldsymbol{\Gamma}\mathbf{T}^\intercal),\end{aligned} \tag{8}$$

where $\boldsymbol{\xi} \in \mathbb{R}^K$, $\boldsymbol{\Lambda} = (\lambda_{li}) \in \mathbb{R}^{L \times n}$, *and* $\boldsymbol{\Gamma} = (\gamma_{hi}) \in \mathbb{R}^{H \times n}$ *are dual variables,* $\bar{\mathbf{U}}_{(3)} \in \mathbb{R}^{d \times nL}$ *and* $\bar{\mathbf{S}}_{(3)} \in \mathbb{R}^{d \times nH}$ *are the mode-3 unfolding of the tensors* $\bar{\mathbf{U}} = (u_{lij}) \in \mathbb{R}^{L \times n \times d}$ *and* $\bar{\mathbf{S}} = (s_{hij}) \in \mathbb{R}^{H \times n \times d}$*, respectively,* $u_{lij} = u_{li} x_{ij}$, $s_{hij} = s_{hi} x_{ij}$, $\mathbf{I}$ *is the identity matrix, and all inequalities are elementwise.*

Moreover, the optimal point $\widehat{\boldsymbol{\beta}}$ *of (6) can be recovered as:*

$$\widehat{\boldsymbol{\beta}} = \sum_{k=1}^{K} \widehat{\xi}_k \mathbf{a}_k - \sum_{i=1}^{n} \mathbf{x}_i \left(\sum_{l=1}^{L} \widehat{\lambda}_{li} u_{li} + \sum_{h=1}^{H} \widehat{\gamma}_{hi} s_{hi} \right) = \mathbf{A}^\intercal \widehat{\boldsymbol{\xi}} - \bar{\mathbf{U}}_{(3)} vec(\widehat{\boldsymbol{\Lambda}}) - \bar{\mathbf{S}}_{(3)} vec(\widehat{\boldsymbol{\Gamma}}). \tag{9}$$

The **linear** relationship between primal and dual variables greatly simplifies the computation of CD.

Canonical CD updates. As a first step, we consider the canonical CD update rule that directly optimizes the dual problem (7) with respect to a single variable. For brevity, in this section we only illustrate the result for λ_{li} variables, and the full details are given in Appendix B.

By excluding the terms unrelated to λ_{li}, we have $\lambda_{li}^{\text{new}} = \operatorname{argmin}_{0 \le \lambda \le 1} \mathcal{L}_{li}(\lambda)$, where

$$\mathcal{L}_{li}(\lambda) = \frac{1}{2} u_{li}^2 (\mathbf{x}_i^\intercal \mathbf{x}_i) \lambda^2 + \sum_{(l',i') \neq (l,i)} \lambda_{l'i'} u_{l'i'} u_{li} (\mathbf{x}_{i'}^\intercal \mathbf{x}_i) \lambda - \sum_{k=1}^{K} \xi_k u_{li} (\mathbf{a}_k^\intercal \mathbf{x}_i) \lambda + \sum_{h',i'} u_{li} \gamma_{h'i'} s_{h'i'} \mathbf{x}_i^\intercal \mathbf{x}_{i'} \lambda - v_{li} \lambda.$$

Therefore, by simple calculations we obtain

$$\lambda_{li}^{\text{new}} = \mathcal{P}_{[0,1]} \left(\frac{u_{li} \mathbf{x}_i^\intercal \Big(\sum_{k=1}^{K} \xi_k \mathbf{a}_k - \sum_{(l',i') \neq (l,i)} \lambda_{l'i'} u_{l'i'} \mathbf{x}_{i'} - \sum_{h',i'} \gamma_{h'i'} s_{h'i'} \mathbf{x}_{i'} \Big) + v_{li}}{u_{li}^2 \|\mathbf{x}_i\|_2^2} \right), \tag{10}$$

where $\mathcal{P}_{[a,b]}(x) = \max(a, \min(b, x))$ means projecting a real number x to the interval $[a, b]$.

Clearly, assuming the values $\mathbf{x}_i^\intercal \mathbf{a}_k$ and $\|\mathbf{x}_i\|_2^2$ are cached, updating one λ_{li} value requires $\mathcal{O}(K + nd + nL + nH)$ of computation, and updating the whole $\boldsymbol{\Lambda}$ matrix requires $\mathcal{O}\big(nL(K + nd + nL + nH)\big)$. Adding all variables together, the canonical CD update rule for one full cycle has a computational complexity of $\mathcal{O}\big((K + nd + nL + nH)(K + nL + nH)\big)$.

ReHLine

Canonical CD updates. As a first step, we consider the canonical CD update rule that directly optimizes the dual problem (7) with respect to a single variable. For brevity, in this section we only illustrate the result for λ_{li} variables, and the full details are given in Appendix B.

By excluding the terms unrelated to λ_{li}, we have $\lambda_{li}^{\text{new}} = \operatorname{argmin}_{0 \le \lambda \le 1} \mathcal{L}_{li}(\lambda)$, where

$$\mathcal{L}_{li}(\lambda) = \frac{1}{2} u_{li}^2 (\mathbf{x}_i^\intercal \mathbf{x}_i) \lambda^2 + \sum_{(l',i') \neq (l,i)} \lambda_{l'i'} u_{l'i'} u_{li} (\mathbf{x}_{i'}^\intercal \mathbf{x}_i) \lambda - \sum_{k=1}^{K} \xi_k u_{li} (\mathbf{a}_k^\intercal \mathbf{x}_i) \lambda + \sum_{h',i'} u_{li} \gamma_{h'i'} s_{h'i'} \mathbf{x}_i^\intercal \mathbf{x}_{i'} \lambda - v_{li} \lambda.$$

Therefore, by simple calculations we obtain

$$\lambda_{li}^{\text{new}} = \mathcal{P}_{[0,1]} \left(\frac{u_{li} \mathbf{x}_i^\intercal \left(\sum_{k=1}^{K} \xi_k \mathbf{a}_k - \sum_{(l',i') \neq (l,i)} \lambda_{l'i'} u_{l'i'} \mathbf{x}_{i'} - \sum_{h',i'} \gamma_{h'i'} s_{h'i'} \mathbf{x}_{i'} \right) + v_{li}}{u_{li}^2 \|\mathbf{x}_i\|_2^2} \right), \tag{10}$$

where $\mathcal{P}_{[a,b]}(x) = \max(a, \min(b, x))$ means projecting a real number x to the interval $[a, b]$.

Clearly, assuming the values $\mathbf{x}_i^\intercal \mathbf{a}_k$ and $\|\mathbf{x}_i\|_2^2$ are cached, updating one λ_{li} value requires $\mathcal{O}(K + nd + nL + nH)$ of computation, and updating the whole $\mathbf{\Lambda}$ matrix requires $\mathcal{O}\big(nL(K + nd + nL + nH)\big)$. Adding all variables together, the canonical CD update rule for one full cycle has a computational complexity of $\mathcal{O}\big((K + nd + nL + nH)(K + nL + nH)\big)$.

$$\widehat{\boldsymbol{\beta}} = \sum_{k=1}^{K} \widehat{\xi}_k \mathbf{a}_k - \sum_{i=1}^{n} \mathbf{x}_i \left(\sum_{l=1}^{L} \widehat{\lambda}_{li} u_{li} + \sum_{h=1}^{H} \widehat{\gamma}_{hi} s_{hi} \right) = \mathbf{A}^\intercal \widehat{\boldsymbol{\xi}} - \bar{\mathbf{U}}_{(3)} vec(\widehat{\mathbf{\Lambda}}) - \bar{\mathbf{S}}_{(3)} vec(\widehat{\mathbf{\Gamma}}). \tag{9}$$

ReHLine

Canonical CD updates. As a first step, we consider the canonical CD update rule that directly optimizes the dual problem (7) with respect to a single variable. For brevity, in this section we only illustrate the result for λ_{li} variables, and the full details are given in Appendix B.

By excluding the terms unrelated to λ_{li}, we have $\lambda_{li}^{\text{new}} = \text{argmin}_{0 \leq \lambda \leq 1} \mathcal{L}_{li}(\lambda)$, where

$$\mathcal{L}_{li}(\lambda) = \frac{1}{2} u_{li}^2 (\mathbf{x}_i^\intercal \mathbf{x}_i) \lambda^2 + \sum_{(l',i') \neq (l,i)} \lambda_{l'i'} u_{l'i'} u_{li} (\mathbf{x}_{i'}^\intercal \mathbf{x}_i) \lambda -$$

$$+ \sum_{h',i'} u_{li} \gamma_{h'i'} s_{h'i'} \mathbf{x}_i^\intercal \mathbf{x}_{i'} \lambda - v_{li} \lambda.$$

Therefore, by simple calculations we obtain

$$\lambda_{li}^{\text{new}} = \mathcal{P}_{[0,1]} \left(\frac{u_{li} \mathbf{x}_i^\intercal \left(\sum_{k=1}^K \xi_k \mathbf{a}_k - \sum_{(l',i') \neq (l,i)} \lambda_{l'i'} u_{l'i'} \mathbf{x}_{i'} - \sum \right.}{u_{li}^2 \|\mathbf{x}_i\|_2^2} \right.$$

where $\mathcal{P}_{[a,b]}(x) = \max(a, \min(b, x))$ means projecting a real number

Clearly, assuming the values $\mathbf{x}_i^\intercal \mathbf{a}_k$ and $\|\mathbf{x}_i\|_2^2$ are cached, updating one $nL + nH)$ of computation, and updating the whole $\mathbf{\Lambda}$ matrix requires (Adding all variables together, the canonical CD update rule for one f complexity of $\mathcal{O}((K + nd + nL + nH)(K + nL + nH))$

ReHLine updates. The proposed ReHLine algorithm, on the other hand, significantly reduces the computational complexity of canonical CD by updating $\boldsymbol{\beta}$ according to the KKT condition (9) after each update of a dual variable. To see this, let $\boldsymbol{\mu} := (\boldsymbol{\xi}, \mathbf{\Lambda}, \mathbf{\Gamma})$ denote all the dual variables, and define

$$\boldsymbol{\beta}(\boldsymbol{\mu}) = \sum_{k=1}^K \xi_k \mathbf{a}_k - \sum_{i=1}^n \mathbf{x}_i \left(\sum_{l=1}^L \lambda_{li} u_{li} + \sum_{h=1}^H \gamma_{hi} s_{hi} \right).$$

Then it can be proved that $(\nabla_\lambda \mathcal{L}_{li})(\lambda_{li}) = -(u_{li} \mathbf{x}_i^\intercal \boldsymbol{\beta}(\boldsymbol{\mu}) + v_{li})$. Therefore, when $\boldsymbol{\mu}$ is fixed at $\boldsymbol{\mu}^{\text{old}} = (\boldsymbol{\xi}^{\text{old}}, \mathbf{\Lambda}^{\text{old}}, \mathbf{\Gamma}^{\text{old}})$ and let $\boldsymbol{\beta}^{\text{old}} = \boldsymbol{\beta}(\boldsymbol{\mu}^{\text{old}})$, (10) can be rewritten as

$$\lambda_{li}^{\text{new}} = \mathcal{P}_{[0,1]} \left(\lambda_{li}^{\text{old}} - \frac{(\nabla_{\lambda_{li}} \mathcal{L})(\lambda^{\text{old}})}{u_{li}^2 \|\mathbf{x}_i\|_2^2} \right) = \mathcal{P}_{[0,1]} \left(\lambda_{li}^{\text{old}} + \frac{u_{li} \mathbf{x}_i^\intercal \boldsymbol{\beta}^{\text{old}} + v_{li}}{u_{li}^2 \|\mathbf{x}_i\|_2^2} \right).$$

Accordingly, the primal variable $\boldsymbol{\beta}$ is updated as

$$\boldsymbol{\beta}^{\text{new}} = \boldsymbol{\beta}^{\text{old}} - (\lambda_{li}^{\text{new}} - \lambda_{li}^{\text{old}}) u_{li} \mathbf{x}_i,$$

which can then be used for the next dual variable update. Simple calculations show that this scheme only costs $\mathcal{O}(d)$ of computation for one λ_{li} variable.

$$\widehat{\boldsymbol{\beta}} = \sum_{k=1}^K \widehat{\xi}_k \mathbf{a}_k - \sum_{i=1}^n \mathbf{x}_i \left(\sum_{l=1}^L \widehat{\lambda}_{li} u_{li} + \sum_{h=1}^H \widehat{\gamma}_{hi} s_{hi} \right) = \mathbf{A}^\intercal \widehat{\boldsymbol{\xi}} - \bar{\mathbf{U}}_{(3)} vec(\widehat{\mathbf{\Lambda}}) - \bar{\mathbf{S}}_{(3)} vec(\widehat{\mathbf{\Gamma}}). \quad (9)$$

Experiments

Software. **generic**/ **specialized** software

- **cvx/cvxpy**
- **mosek** (IPM)
- **ecos** (IPM)
- **scs** (ADMM)
- **dccp** (DCP)
- **liblinear** -> SVM
- **hqreg** -> Huber
- **lightning** -> sSVM

Table 5: The averaged running times (± standard deviation) of SOTA solvers on machine learning tasks. "✗" indicates cases where the solver produced an invalid solution or exceeded the allotted time limit. Speed-up refers to the speed-up in the averaged running time (on the largest dataset) achieved by ReHLine, where "∞" indicates that the solver fails to solve the problem.

TASK	DATASET	ECOS	MOSEK	SCS	DCCP	REHLINE
FairSVM	SPF (×1e-4)	✗	✗	✗	✗	4.25(±0.5)
	Philippine (×1e-2)	1550(±0.6)	87.4(±0.2)	130(±42)	1137(±9.2)	1.03(±0.2)
	Sylva-prior (×1e-2)	✗	✗	✗	✗	0.47(±0.1)
	Creditcard (×1e-1)	175(±0.2)	64.2(±0.1)	161(±405)	✗	0.64(±0.2)
	Fail/Succeed	2/2	2/2	2/2	3/1	0/4
	Speed-up (on Creditcard)	273x	100x	252x	∞	–

TASK	DATASET	ECOS	MOSEK	SCS	REHLINE
ElasticQR	LD (×1e-4)	✗	106(±7)	34.9(±25.0)	2.60(±0.30)
	Kin8nm (×1e-3)	✗	92.0(±1.0)	63.1(±58.5)	4.12(±0.95)
	House-8L (×1e-3)	887(±161)	277(±34)	✗	7.21(±1.99)
	Topo-2-1 (×1e-2)	4752(±2015)	✗	✗	3.04(±0.49)
	BT (×1e-0)	7079(±2517)	✗	✗	2.49(±0.56)
	Fail/Succeed	3/2	2/3	3/2	0/5
	Speed-up (on BT)	2843x	∞	∞	–

TASK	DATASET	ECOS	MOSEK	SCS	HQREG	REHLINE
RidgeHuber	Liver-disorders (×1e-4)	✗	✗	✗	4.90(±0.00)	1.40(±0.20)
	Kin8nm (×1e-3)	✗	✗	✗	1.58(±0.21)	2.04(±0.30)
	House-8L (×1e-3)	✗	925(±2)	✗	2.42(±0.34)	0.80(±0.21)
	Topo-2-1 (×1e-2)	2620(±1040)	267(±1)	213(±2)	3.53(±0.67)	1.78(±0.32)
	BT (×1e-1)	✗	2384(±433)	✗	12.5(±1.8)	5.28(±1.31)
	Fail/Succeed	4/1	2/3	4/1	0/5	0/5
	Speed-up (on BT)	∞	452x	∞	2.37x	–

TASK	DATASET	ECOS	MOSEK	SCS	LIBLINEAR	REHLINE
SVM	SPF (×1e-4)	✗	372(±1)	237(±27)	12.7(±0.1)	3.90(±0.10)
	Philippine (×1e-2)	1653(±41)	86.5(±0.2)	153(±146)	1.80(±0.02)	0.82(±0.02)
	Sylva-prior (×1e-3)	✗	731(±2)	843(±1006)	16.0(±0.6)	4.08(±0.84)
	Creditcard (×1e-2)	2111(±804)	✗	1731(±4510)	23.1(±2.5)	5.08(±1.45)
	Fail/Succeed	2/2	1/3	0/4	0/4	0/4
	Speed-up (on Creditcard)	415x	∞	340x	4.5x	–

TASK	DATASET	SAGA	SAG	SDCA	SVRG	REHLINE
sSVM	SPF (×1e-4)	39.9(±4.6)	28.3(±5.0)	15.0(±2.4)	41.4(±3.9)	4.80(±1.20)
	Philippine (×1e-2)	24.3(±27.8)	5.53(±9.8)	1.47(±0.19)	15.8(±6.8)	0.89(±0.10)
	Sylva-prior (×1e-2)	3.37(±9.81)	3.00(±0.56)	1.57(±0.23)	3.40(±0.84)	0.86(±0.14)
	Creditcard (×1e-2)	10.4(±1.4)	15.0(±2.0)	14.0(±1.9)	11.2(±1.4)	6.36(±1.92)
	Fail/Succeed	0/4	0/4	0/4	0/4	0/4
	Speed-up (on Creditcard)	1.6x	2.3x	2.2x	1.7x	–

Thank you!