On PI Controllers for Updating Lagrange Multipliers in Constrained Optimization

Motahareh Sohrabi [*,†] **Juan Ramirez** [*,†] **Tianyue H. Zhang** [†] **Simon Lacoste-Julien** [†,‡] **Jose Gallego-Posada** [†]

Abstract

Constrained optimization offers a powerful framework to prescribe desired behaviors in neural network models. Typically, constrained problems are solved via their min-max Lagrangian formulations, which exhibit unstable oscillatory dynamics when optimized using gradient descent-ascent. The adoption of constrained optimization techniques in the machine learning community is currently limited by the lack of reliable, general-purpose update schemes for the Lagrange multipliers. This paper proposes the $\boldsymbol{\nu}$PI algorithm and contributes an optimization perspective on Lagrange multiplier updates based on PI controllers, extending the work of Stooke et al. (2020). We provide theoretical and empirical insights explaining the inability of momentum methods to address the shortcomings of gradient descent-ascent, and contrast this with the empirical success of our proposed $\boldsymbol{\nu}$PI controller. Moreover, we prove that $\boldsymbol{\nu}$PI generalizes popular momentum methods for single-objective minimization. Our experiments demonstrate that $\boldsymbol{\nu}$PI reliably stabilizes the multiplier dynamics and its hyperparameters enjoy robust and predictable behavior.

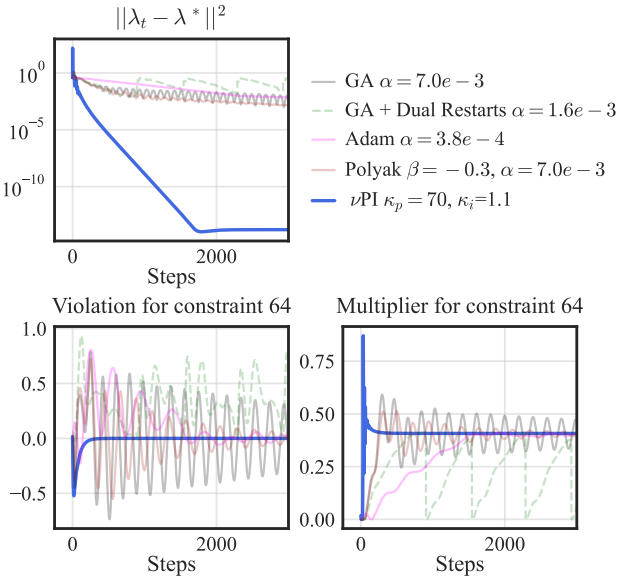


Figure 1: Dynamics for different dual optimizers on a hard-margin SVM problem (Eq. (12)). Amongst the tested methods, **$\boldsymbol{\nu}$PI is the only method to successfully converge to the optimal dual variables**. Each optimizer uses the best hyperparameters found after a grid-search aiming to minimize the distance to the optimal $\boldsymbol{\lambda}^*$ after 5.000 steps. For improved readability, the plot shows the first 3.000 steps. Constraint 64 corresponds to a support vector. All methods achieved perfect training accuracy.

1. Introduction

The need to enforce complex behaviors in neural network models has reinvigorated the interest of the machine learning community in constrained optimization techniques. Recent applications include fairness (Cotter et al., 2019; Zafar et al., 2019; Fioretto et al., 2020; Hashemizadeh et al., 2024), sparsity (Gallego-Posada et al., 2022), active learning (Elenter et al., 2022), reinforcement learning (Stooke et al., 2020; Farahmand & Ghavamzadeh, 2021) and model quantization (Hounie et al., 2023).

Algorithmic approaches based on the Lagrangian min-max representation of the original constrained optimization problem (Boyd & Vandenberghe, 2004, §5) are commonly preferred in the context of neural networks since (i) they are amenable to inexact, gradient-based optimization (Bertsekas, 2016, §5.2), (ii) making it easy to incorporate constraints into existing pipelines for unconstrained optimization (Cotter et al., 2019; Gallego-Posada & Ramirez, 2022), and (iii) they do not require special structure in the objective or constraint functions (such as convexity or efficient projection onto the feasible set (Nocedal & Wright, 2006)).

Despite their wider applicability, solving Lagrangian problems involving neural networks is challenging as it simultaneously entails the difficulties of nonconvex optimization on large-scale models (Bottou et al., 2018), and the potential for instability and oscillations due to the adversarial min-max nature of the Lagrangian (Stooke et al., 2020).

* Equal contribution. † Mila—Quebec AI Institute and DIRO, Université de Montréal. ‡ Canada CIFAR AI Chair. Correspondence to: Juan Ramirez <juan.ramirez@mila.quebec>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

Lagrangian problems are commonly optimized using some variant of gradient-descent ascent (GDA) (Arrow et al., 1958). Despite local convergence results in idealized settings (Lin et al., 2020; Zhang et al., 2022), the optimization dynamics of GDA typically exhibit instabilities, overshoot or oscillations (Platt & Barr, 1987; Gidel et al., 2019a; Stooke et al., 2020; Gallego-Posada et al., 2022).

Alleviating the shortcomings of GDA on Lagrangian problems is an important step towards widespread adoption constrained optimization in deep learning. Recently, Stooke et al. (2020) proposed a solution based on a PID controller (Åström & Hägglund, 1995) for updating the Lagrange multipliers in safety-constrained reinforcement learning problems. Our manuscript expands on their work by providing an optimization-oriented analysis of $\boldsymbol{\nu}$PI (Algo. 1), a related PI controller that incorporates an exponential moving average on the error signal.

Fig. 1 illustrates how our proposed $\boldsymbol{\nu}$PI controller successfully dampens the oscillations on a hard-margin SVM task, achieving fast convergence to the optimal Lagrange multipliers. In contrast, a wide range of popular methods for single-objective minimization exhibit unstable, oscillatory dynamics and fail to converge in this task. See §5.1 for further details on this experiement.

Contributions: ① We introduce the $\boldsymbol{\nu}$PI algorithm (§4) and prove that $\boldsymbol{\nu}$PI generalizes popular momentum methods like POLYAK and NESTEROV (Thm. 1), as well as traditional PI controllers. ② We provide conceptual insights explaining how $\boldsymbol{\nu}$PI improves the dynamics of the Lagrange multipliers: §4.3 presents a qualitative analysis of the updates executed by the $\boldsymbol{\nu}$PI algorithm in contrast to gradient ascent; in §4.4 we study the spectral properties of the continuous-time system. ③ In §4.5, we provide a heuristic to tune the new hyperparameter κ_p of the $\boldsymbol{\nu}$PI algorithm; we also demonstrate that it has a monotonic effect in the damping of oscillations. ④ Our experiments on hard-margin SVMs, sparsity tasks using ResNets, and algorithmic fairness demonstrate that $\boldsymbol{\nu}$PI leads to improved stability and convergence.

Code: `github.com/motahareh-sohrabi/nuPI`

Scope: Due to the highly specialized techniques used for training neural networks (Dahl et al., 2023), in this work we concentrate on iterative schemes that do *not* modify the optimization protocol used on the model parameters. In other words, we restrict our attention to update schemes on the Lagrange multipliers only, which allows us to reuse the same optimizer choices for the (primal) model parameters as used in the unconstrained setting.

2. Related Works

Constrained optimization. We are interested in Lagrangian methods (Arrow et al., 1958) that allow tackling general (nonconvex) constrained optimization problems with differentiable objective and constraints. Classical constrained optimization (Nocedal & Wright, 2006; Bertsekas, 2016) techniques include projection methods (Bertsekas, 1976), barrier methods (Dikin, 1967), and methods of feasible directions (Frank & Wolfe, 1956; Zoutendijk, 1960). These approaches usually make assumptions on the structure of the problem, such as convexity of the objective or constraints, the existence of an efficient projection operator onto the feasible set, or access to a linear minimization oracle. Such assumptions restrict their applicability to deep learning tasks. Other popular techniques such as penalty methods (Nocedal & Wright, 2006) and the method of multipliers (Bertsekas, 1975), apply to general nonconvex problems, but are outside the scope of this work.

Min-max optimization. The Lagrangian formulation of a nonconvex constrained optimization problem leads to a nonconvex concave min-max problem. Under idealized assumptions, gradient descent-ascent has local convergence guarantees for said problems (Lin et al., 2020), but may exhibit oscillations (Platt & Barr, 1987; Gidel et al., 2019b). Under stronger assumptions, extragradient (Korpelevich, 1976) and the optimistic gradient method (Popov, 1980) converge at a nearly optimal rate (Mokhtari et al., 2020a). These methods, as well as POLYAK with negative momentum (Gidel et al., 2019a) and PID controllers (Stooke et al., 2020), have been shown to dampen the oscillations of GDA. However, negative momentum may be suboptimal for strongly convex-strongly concave min-max problems (Zhang & Wang, 2021).

Our work focuses on the *dynamics* of Lagrangian games. We provide insights on why popular techniques for minimization may exacerbate oscillations and overshoot, and why PI controllers can be effective at damping oscillations. Our proposed method $\boldsymbol{\nu}$PI is a generalization of both (negative) momentum and the optimistic gradient method.

PID controllers and optimization. An et al. (2018) studied PID control for training machine learning models by considering the negative loss gradient as the error signal to the controller. PID controllers have been shown to generalize gradient descent (Hu & Lessard, 2017) and momentum (Recht, 2018). Stooke et al. (2020); Casti et al. (2023) have highlighted the effectiveness of controllers at optimizing constrained optimization tasks.

In this work, we propose a PI-like update rule for the dual variables in a Lagrangian min-max game. We prove our algorithm generalizes momentum methods and we provide conceptual insights to support the empirical effectiveness of PI controllers in reducing oscillations and overshoot in the constrained optimization dynamics. In Appx. A, we elaborate on the distinctions between our work and existing research on PID controllers for optimization.

3. Lagrangian Optimization

Consider a constrained optimization problem with m inequality and n equality constraints, represented by functions $\boldsymbol{g} : \mathcal{X} \to \mathbb{R}^m$ and $\boldsymbol{h} : \mathcal{X} \to \mathbb{R}^n$, respectively:

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) \;\text{ subject to }\; \boldsymbol{g}(\boldsymbol{x}) \leq \mathbf{0} \;\text{ and }\; \boldsymbol{h}(\boldsymbol{x}) = \mathbf{0}. \tag{1}$$

We do not make any assumptions on the functions f, $\boldsymbol{g}$, and $\boldsymbol{h}$ beyond almost-everywhere differentiability. We refer to the values of $\boldsymbol{g}$ and $\boldsymbol{h}$ as the *constraint violations*. In particular, we are interested in optimization problems where $\boldsymbol{x}$ corresponds to the parameters of a neural network, leading to objective and constraint functions that may be nonconvex. This typically precludes the use of "classical" constrained optimization methods, as those discussed in §2.

The Lagrangian min-max problem associated with the constrained optimization problem in Eq. (1) is given by:

$$\min_{\boldsymbol{x}} \max_{\boldsymbol{\lambda} \geq \mathbf{0},\, \boldsymbol{\mu}} \mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \triangleq f(\boldsymbol{x}) + \boldsymbol{\lambda}^\top \boldsymbol{g}(\boldsymbol{x}) + \boldsymbol{\mu}^\top \boldsymbol{h}(\boldsymbol{x}), \tag{2}$$

where $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ are vectors of *Lagrange multipliers* associated with the inequality and equality constraints, respectively. Eq. (2) constitutes a nonconvex-concave zero-sum game between $\boldsymbol{x}$ (known as the *primal* player) and $\{\boldsymbol{\lambda}, \boldsymbol{\mu}\}$ (known as the *dual* player). We are interested in algorithmic approaches that identify saddle points of the Lagrangian $\mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$ as these correspond to constrained optima.

In general, Lagrangian-based approaches do not constitute *feasible methods* (i.e. visiting only feasible iterates). We judge a method's success based on its asymptotic feasibility, or at the end of a pre-determined optimization budget.

Simultaneous updates. The simplest algorithm to solve the problem in Eq. (2) is simultaneous gradient descent-ascent (GDA) (Arrow et al., 1958):

$$\begin{aligned}
&\boldsymbol{\mu}_{t+1} \leftarrow \boldsymbol{\mu}_t + \eta_{\text{dual}}\, \nabla_{\boldsymbol{\mu}} \mathfrak{L}(\boldsymbol{x}_t, \boldsymbol{\lambda}_t, \boldsymbol{\mu}_t) = \boldsymbol{\mu}_t + \eta_{\text{dual}}\, \boldsymbol{h}(\boldsymbol{x}_t) \\
&\begin{cases}
\hat{\boldsymbol{\lambda}}_{t+1} \leftarrow \boldsymbol{\lambda}_t + \eta_{\text{dual}}\, \nabla_{\boldsymbol{\lambda}} \mathfrak{L}(\boldsymbol{x}_t, \boldsymbol{\lambda}_t, \boldsymbol{\mu}_t) = \boldsymbol{\lambda}_t + \eta_{\text{dual}}\, \boldsymbol{g}(\boldsymbol{x}_t) \\
\boldsymbol{\lambda}_{t+1} \leftarrow \Pi_{\mathbb{R}^m_+}(\hat{\boldsymbol{\lambda}}_{t+1}) = \max\left(0, \hat{\boldsymbol{\lambda}}_{t+1}\right)
\end{cases} \\
&\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \eta_{\text{primal}}\, \nabla_{\boldsymbol{x}} \mathfrak{L}(\boldsymbol{x}_t, \boldsymbol{\lambda}_t, \boldsymbol{\mu}_t),
\end{aligned}$$

where the middle two equations execute a projected gradient-ascent step enforcing the non-negativity of the multipliers $\boldsymbol{\lambda}$.

To simplify notation, we will group the dual variables as $\boldsymbol{\theta} = [\boldsymbol{\lambda}, \boldsymbol{\mu}]^\top$ and the constraints $\boldsymbol{c}(\boldsymbol{x}) = [\boldsymbol{g}(\boldsymbol{x}), \boldsymbol{h}(\boldsymbol{x})]^\top$ which yields the concise Lagrangian problem:

$$\min_{\boldsymbol{x}} \max_{\boldsymbol{\theta} \in \mathbb{R}^m_+ \times \mathbb{R}^n} \mathfrak{L}(\boldsymbol{x}, \boldsymbol{\theta}) \triangleq f(\boldsymbol{x}) + \boldsymbol{\theta}^\top \boldsymbol{c}(\boldsymbol{x}) \tag{3}$$

Note that the primal update direction $\nabla_{\boldsymbol{x}} \mathfrak{L}$ is a linear combination of the objective and constraint gradients—which can be efficiently computed using automatic differentiation, without storing ∇f and $\mathcal{J}\boldsymbol{c}$[1] separately. On the other hand, $\nabla_{\boldsymbol{\theta}} \mathfrak{L} = \boldsymbol{c}(\boldsymbol{x})$, and thus the GDA update on the multipliers corresponds to the *integration* (i.e. accumulation) of the constraint violations over time. We highlight that the cost of updating the Lagrange multipliers is typically negligible relative to the cost of computing f and $\boldsymbol{c}$.

Alternating updates. Prior work has demonstrated the advantages of alternating updates in min-max optimization: Zhang et al. (2022) established that alternating GDA achieves a near-optimal local convergence rate for strongly concave-strongly convex problems (strictly better than simultaneous GDA); Gidel et al. (2019b) showed that alternating GDA leads to bounded iterates on smooth bilinear games, as opposed to divergence for simultaneous updates. Besides the improved convergence and stability benefits, alternating updates are particularly suitable for Lagrangian games from a computational standpoint due to the linear structure of the Lagrangian with respect to the dual variables. Concretely, consider the *alternating* update scheme:

$$\begin{aligned}
&\begin{cases}
\hat{\boldsymbol{\theta}}_{t+1} \leftarrow \boldsymbol{\theta}_t + \eta_{\text{dual}}\, \nabla_{\boldsymbol{\theta}} \mathfrak{L}(\boldsymbol{x}_t, \boldsymbol{\theta}_t) = \boldsymbol{\theta}_t + \eta_{\text{dual}}\, \boldsymbol{c}(\boldsymbol{x}_t) \\
\boldsymbol{\theta}_{t+1} \leftarrow \Pi_{\mathbb{R}^m_+ \times \mathbb{R}^n}(\hat{\boldsymbol{\theta}}_{t+1})
\end{cases} \\
&\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \eta_{\text{primal}}\, \nabla_{\boldsymbol{x}} \mathfrak{L}(\boldsymbol{x}_t, \boldsymbol{\theta}_{t+1}) \\
&\qquad\quad = \boldsymbol{x}_t - \eta_{\text{primal}}\, (\nabla f(\boldsymbol{x}_t) + \mathcal{J}\boldsymbol{c}(\boldsymbol{x}_t)\, \boldsymbol{\theta}_t)
\end{aligned} \tag{4}$$

The alternating updates in Eq. (4), only require computing $f(\boldsymbol{x}_t)$ and $\boldsymbol{c}(\boldsymbol{x}_t)$ once, just as when performing simultaneous updates. In a general zero-sum game, where $\mathfrak{L}(\boldsymbol{x}_t, \boldsymbol{\theta}_t)$ does not decouple as in the Lagrangian case, the second part of the alternation might require re-evaluating $\mathfrak{L}(\boldsymbol{x}_t, \boldsymbol{\theta}_{t+1})$ entirely. However, note that thanks to the affine structure of $\mathfrak{L}$ with respect to $\boldsymbol{\theta}$, the update on $\boldsymbol{x}$ can be calculated efficiently without having to re-evaluate f or $\boldsymbol{c}$.

These theoretical and practical advantages motivate our decision to concentrate on alternating update schemes like Eq. (4) for solving the problem in Eq. (3) in what follows.

Practical remarks. In practice, updates on the primal variables require more sophisticated methods (with intricate hyperparameter tuning) than the plain gradient descent update presented in Eq. (4) to achieve good performance, including any number of highly specialized procedures developed for training neural networks (Dahl et al., 2023).

Moreover, for certain applications, a training pipeline designed to minimize a single, *unconstrained* objective might be in place. In these cases, it is desirable to develop update schemes for the Lagrange multipliers that allow for seamlessly incorporating constraints into the model development pipeline *without having to engineer from scratch a new recipe for training the model.*

[1] $\mathcal{J}\boldsymbol{f} \triangleq \begin{bmatrix} \nabla f_1 & \cdots & \nabla f_p \end{bmatrix} \in \mathbb{R}^{d \times p}$ denotes the (transpose) Jacobian matrix of a function $\boldsymbol{f} : \mathbb{R}^d \to \mathbb{R}^p$.

In this paper, we concentrate on different update schemes for the Lagrange multipliers and assume that a well-tuned optimizer for the model parameters is available.

Shortcomings of gradient ascent. As mentioned previously, gradient ascent (GA) on the Lagrange multipliers corresponds to accumulating the observed constraint violations over time. For simplicity, let us concentrate on a single inequality constraint $c(\boldsymbol{x})$. Whenever the constraint is being violated (resp. satisfied), the violation is positive $c(\boldsymbol{x}) > 0$ (resp. negative) and thus the value of the corresponding multiplier is increased (resp. decreased) by $\eta_{\text{dual}} c(\boldsymbol{x})$. Recall that the projection step ensures that the inequality multipliers remain non-negative.

Therefore, the value of the multiplier depends on the entire optimization trajectory through the value of the observed violations. In particular, after a long period of infeasibility, the value of the multiplier will be large, biasing the gradient $\nabla_{\boldsymbol{x}} \mathfrak{L}$ towards reducing the violation and thus improving the feasibility of the model.

An insufficient increase of the multiplier will cause the constraint to be *ignored*, while an excessively large value of the multiplier will lead the constraint to be enforced *beyond* the prescribed constraint level. The latter behavior can also occur if the multiplier fails to decrease sufficiently fast once the constraint is satisfied. Repeated cycles of insufficient or excessive change in the multiplier manifest in ignoring or overshooting, thus forming oscillations. See Figs. 1 and 2 for illustrations of these behaviors.

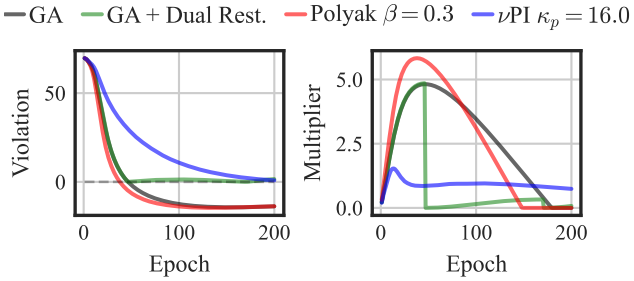


Figure 2: Constraint dynamics for GA, POLYAK and νPI in a sparsity task (§5.3). Constrained optimal solutions for this problem lie at the boundary of the feasible set. The excessive growth in the value of the multiplier for GA causes the constraint to overshoot into the interior of the feasible set. **The improved multiplier updates of the νPI algorithm remove the overshoot in the constraint and multiplier.**

In short, an ideal update rule for the multiplier would behave *adaptively*, based on the observed violations throughout the execution of the optimization. This begs the question of whether existing adaptive optimization such as POLYAK, NESTEROV and ADAM would reliably resolve these issues. Sections 4 and 5 provide a *negative* answer to this question.

Dual restarts. Gallego-Posada et al. (2022) proposed an approach to mitigate the overshoot in inequality constraints called *dual restarts*: once a constraint is *strictly* satisfied, its associated dual variable is reset to zero. This corresponds to a best response (in game-theoretic terms) of the dual player. Dual restarts prevent excessive enforcement of constraints, which can degrade the achieved objective function value.

However, dual restarts are not suitable for general constrained optimization problems since they rely on determining the satisfaction of the constraint *exactly*. Constraint estimates may (wrongly) indicate strict feasibility due to (i) stochasticity in their estimation, (ii) numerical precision errors making active constraints appear strictly feasible, or (iii) a "temporary" strict satisfaction of the constraint. Fig. 1 illustrates the undesirable dynamics caused by dual restarts when applied to an SVM task in which the support vectors correspond to strictly active inequality constraints.

In §4, we show that $\boldsymbol{\nu}$PI mitigates the overshoot of inequality constraints, with additional benefits: (i) controllable degree of overshoot (governed by the κ_p hyperparameter), (ii) compatibility with equality (and strictly feasible inequality) constraints, and (iii) damping of multiplier oscillations.

4. νPI Control for Constrained Optimization

Following Stooke et al. (2020), we consider the learning of an optimal feasible model solving Eq. (1) as a dynamical system. Thus, we can think of the update rule for the multipliers as a control algorithm that aims to *steer the system toward feasibility*. We emphasize that we are not trying to control general dynamical systems, but rather systems that arise from partial, inexact minimization (e.g. gradient-based updates) on a min-max Lagrangian game. In other words, we assume that $\boldsymbol{x}_{t-1} \mapsto \boldsymbol{x}_t$ is updated so as to minimize the current Lagrangian $\mathfrak{L}(\,\cdot\,, \boldsymbol{\theta}_t)$. Figure 3 illustrates the control pipeline we consider throughout this work.

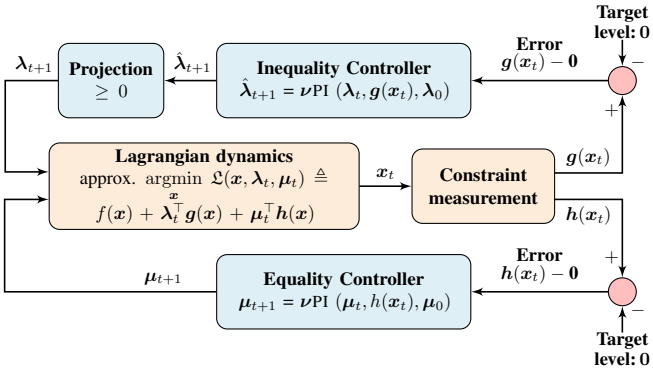


Figure 3: νPI control pipeline for updating the Lagrange multipliers in a constrained optimization problem. We consider the update on the primal variables as a black-box procedure that receives the multipliers $\boldsymbol{\lambda}_t$ and primal variables $\boldsymbol{x}_{t-1}$ as input, and returns an updated $\boldsymbol{x}_t$. The multiplier update is executed by the controller, using the constraint violations as the error signal.

An assumption entailed by the control perspective in Fig. 3 is that an increase (resp. decrease) in the control variable (the Lagrange multipliers $\boldsymbol{\theta}_t$) leads to a decrease (resp. increase) in the controlled quantity (the constraint violations $\boldsymbol{c}(\boldsymbol{x}_t)$). This assumption holds for constrained optimization problems since an increase in the multipliers leads the primal minimization of the Lagrangian to focus on reducing the value of the constraints (as mentioned during the discussion of gradient descent-ascent dynamics in §3).

Note that our black-box assumption on the nature of the primal update allows for an arbitrary choice of optimizer for minimizing $\mathfrak{L}(\,\cdot\,, \boldsymbol{\theta}_t)$. After obtaining an updated primal iterate $\boldsymbol{x}_t$, the new constraint violations $\boldsymbol{g}(\boldsymbol{x}_t)$ and $\boldsymbol{h}(\boldsymbol{x}_t)$ are measured and used as the error signals for the inequality- and equality-constraint controllers, yielding updated multipliers $\boldsymbol{\theta}_{t+1}$. The projection block ensures the non-negativity of the multipliers for inequality constraints.

4.1. νPI algorithm

Our main algorithmic contribution is the multiplier update scheme presented in Algo. 1. This is a simple generalization of a PI controller (i.e. a PID controller (Åström & Hägglund, 1995) with $\kappa_d = 0$) by including an exponential moving average (of the error signal) in the proportional term. Indeed, the traditional PI controller is recovered when $\nu = 0$.

Algorithm 1 $\boldsymbol{\nu}$PI update

Args: EMA coefficient ν, proportional (κ_p) and integral (κ_i) gains; initial conditions $\boldsymbol{\xi}_0$ and $\boldsymbol{\theta}_0$.

Measure current system error $\boldsymbol{e}_t$

$\boldsymbol{\xi}_t \leftarrow \nu\boldsymbol{\xi}_{t-1} + (1-\nu)\boldsymbol{e}_t$ ▷ for $t \geq 1$

$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_0 + \kappa_p\boldsymbol{\xi}_t + \kappa_i \sum_{\tau=0}^{t} \boldsymbol{e}_\tau$

The $\boldsymbol{\nu}$PI update can be equivalently expressed in terms of a recursive update (see Lemma 2 in Appx. B) as:

$$\boldsymbol{\theta}_1 = \boldsymbol{\theta}_0 + \kappa_i\boldsymbol{e}_0 + \kappa_p\boldsymbol{\xi}_0 \tag{5}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \kappa_i\boldsymbol{e}_t + \kappa_p\left(\boldsymbol{\xi}_t - \boldsymbol{\xi}_{t-1}\right) \quad \text{for } t \geq 1. \tag{6}$$

4.2. Connections to optimization methods

When the error signal corresponds to the negative gradient of a cost function $\boldsymbol{e}_t = -\nabla f_t$, Algo. 1 has straightforward equivalences with common minimization methods (see Appx. B). For example, $\boldsymbol{\nu}$PI $(\nu = 0, \kappa_p = 0, \kappa_i)$ is equivalent to GD $(\alpha = \kappa_i)$ (Stooke et al., 2020; Lessard et al., 2016; An et al., 2018). When $\nu = 0$ and $\kappa_p = \kappa_i = \alpha$, $\boldsymbol{\nu}$PI recovers a single-player version of the OPTIMISTICGRADIENT (OG) method (Popov, 1980), with step-size α. When $\nu = 0$, but κ_p and κ_i are allowed to differ, $\boldsymbol{\nu}$PI coincides with the generalized OG studied by Mokhtari et al. (2020b). Since we use $\boldsymbol{\nu}$PI for updating the multipliers, we phrase the updates in Algo. 1 based on a maximization convention.

Moreover, our proposed algorithm **$\boldsymbol{\nu}$PI generalizes popular momentum methods** such as POLYAK—also known as HEAVYBALL—(Polyak, 1964) and NESTEROV (Nesterov, 1983).[2] This connection, stated formally in Thm. 1, will allow us to understand (§4.3) why traditional momentum methods are *insufficient* to address the shortcomings of gradient ascent for Lagrangian optimization.

We take advantage of the UNIFIEDMOMENTUM (α, β, γ) framework introduced by Shen et al. (2018) to concisely develop a joint analysis of POLYAK$(\alpha, \beta) =$ UM$(\alpha, \beta, \gamma = 0)$ and NESTEROV$(\alpha, \beta) =$ UM$(\alpha, \beta, \gamma = 1)$.

Algorithm 2 UNIFIEDMOMENTUM update (Shen et al., 2018)

Args: step-size α, momentum coefficient β, interpolation factor $\gamma \in \left[0, \frac{1}{1-\beta}\right]$; initial conditions $\boldsymbol{\phi}_0 = \mathbf{0}$ and $\boldsymbol{\theta}_0$.

Measure current system error $\boldsymbol{e}_t$

$\boldsymbol{\phi}_{t+1} \leftarrow \beta\boldsymbol{\phi}_t + \alpha\boldsymbol{e}_t$

$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \boldsymbol{\phi}_{t+1} + \beta\gamma\left(\boldsymbol{\phi}_{t+1} - \boldsymbol{\phi}_t\right)$

Theorem 1. [Proof in Appx. B.] *Under the same initialization $\boldsymbol{\theta}_0$,* UNIFIEDMOMENTUM$(\alpha, \beta \neq 1, \gamma)$ *is a special case of the* $\boldsymbol{\nu}$PI *algorithm with the hyperparameter choices:*

$$\nu \leftarrow \beta \qquad \boldsymbol{\xi}_0 \leftarrow (1-\beta)\boldsymbol{e}_0 \tag{7}$$

$$\kappa_i \leftarrow \frac{\alpha}{1-\beta} \qquad \kappa_p \leftarrow -\frac{\alpha\beta}{(1-\beta)^2}\left[1 - \gamma(1-\beta)\right]. \tag{8}$$

Table 2 in Appx. B summarizes the connections we have established between $\boldsymbol{\nu}$PI and existing methods. We emphasize that the exponential moving average in $\boldsymbol{\nu}$PI is a crucial component to obtain the generalization of momentum methods.

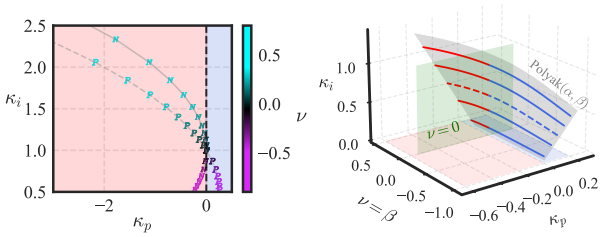


Figure 4: **Left:** Hyperparameter choices from Thm. 1 for which $\boldsymbol{\nu}$PI $(\nu, \kappa_p, \kappa_i)$ realizes POLYAK$(\alpha = \frac{1}{2}, \beta)$ and NESTEROV$(\alpha = \frac{1}{2}, \beta)$. **Right:** The right plot zooms on the range $-1 \leq \beta \leq 0.25$. **POLYAK comprises a limited surface in the $(\nu, \kappa_p, \kappa_i)$ space, leaving configurations outside this surface unexplored**. Note how positive (resp. negative) values of β result in negative (resp. positive) values of κ_p, colored in red (resp. blue). Colored paths correspond to different values of α. The dashed curves match between both plots.

Fig. 4 visually emphasizes the greater generality of $\boldsymbol{\nu}$PI compared to POLYAK and NESTEROV, presented in Thm. 1. Note

[2] We consider a variant of the Nesterov method that uses a constant momentum coefficient.

that the κ_p coefficient changes between POLYAK and NESTEROV, while the κ_i coefficient coincides. Formally,

$$\kappa_p^{\text{POLYAK}} = -\frac{\alpha\beta}{(1-\beta)^2}, \quad \kappa_p^{\text{NESTEROV}} = -\frac{\alpha\beta^2}{(1-\beta)^2} \leq 0. \quad (9)$$

Moreover, $\kappa_p^{\text{NESTEROV}}$ is *non-positive*, regardless of β. In contrast, a negative momentum value $\beta < 0$ induces a *positive* κ_p^{POLYAK}. This observation is in line with the benefits of using a negative POLYAK momentum coefficient (for both players) in adversarial games presented by Gidel et al. (2019a).

4.3. Interpreting the updates of νPI

Consider the execution of $\boldsymbol{\nu}$PI $(\nu, \kappa_p, \kappa_i)$ and GA$(\alpha = \kappa_i)$ at time t.[3] The relative size between these updates is:

$$\frac{\Delta\boldsymbol{\nu}\text{PI}}{\Delta\text{GA}} \triangleq \frac{\theta_{t+1}^{\nu\text{PI}} - \theta_t}{\theta_{t+1}^{\text{GA}} - \theta_t} = \frac{1}{1-\psi}\left[1 - \frac{\psi\xi_{t-1}}{e_t}\right], \quad (10)$$

where $\psi \triangleq \frac{\kappa_p(1-\nu)}{\kappa_i+\kappa_p(1-\nu)}$. Fig. 5 illustrates the behavior of the relative size of updates of $\boldsymbol{\nu}$PI compared to GA. The left plot displays $\boldsymbol{\nu}$PI with $\kappa_p > 0$ and $\nu = 0$. The right plot shows the $\boldsymbol{\nu}$PI-equivalent of POLYAK with *positive* momentum.[4].

Consider the colored regions present in the *left* plot of Fig. 5:

Mode A When $\xi_{t-1} < e_t$, the current violation is greater than the historical violation average (right region). $\boldsymbol{\nu}$PI algorithm increases the multiplier *faster* than GA. When $e_t < 0$ (left region), the primal iterate is feasible and the $\boldsymbol{\nu}$PI algorithm agrees with GA in decreasing the multiplier, but does so *much faster* (with a factor above $\frac{1}{1-\psi}$).

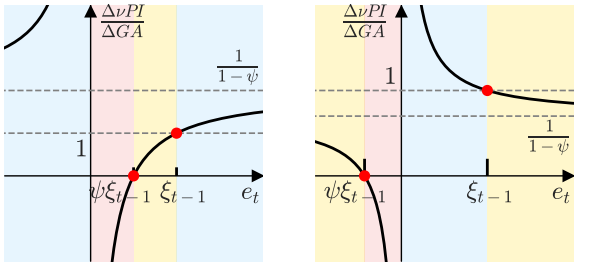
Mode B When $e_t \in [\psi\xi_{t-1}, \xi_{t-1}]$, the constraint violation has improved compared to the historical average but is still infeasible. In this case, $\boldsymbol{\nu}$PI increases the multiplier *more slowly* than GA, consistent with the perceived improvement in the violation.

Mode C When $e_t \in [0, \kappa\xi_{t-1}]$, the primal iterate is still infeasible. However, the $\boldsymbol{\nu}$PI algorithm determines that the constraint improvement is large enough to warrant a *decrease* in the multiplier. Note that in this case, GA would have continued increasing the multiplier.

In all of these cases, the $\boldsymbol{\nu}$PI optimizer can be seen as executing proactively by considering how the current constraint violation compares to the historical estimates. This proactive behavior allows the method to *increase the multiplier faster than* GA when the constraint satisfaction is degrading, *and reduce the multiplier faster than* GA whenever sufficient improvement has been made.

[3] It is sufficient to consider a single scalar multiplier since the updates of both algorithms decouple across constraints/multipliers.

[4] The case of POLYAK with negative momentum resembles the left plot of Fig. 5 See Appx. C for further details.



νPI$(\kappa_i, \kappa_p > 0, \nu = 0)$ νPI$(\kappa_i, \kappa_p < 0, \nu)$=POLYAK$(\alpha, \beta > 0)$

Figure 5: Comparing the update of $\boldsymbol{\nu}$PI relative to GA. **$\boldsymbol{\nu}$PI increases the multipliers faster than GA when the constraint violation is large, enhancing convergence speed; and proactively decreases them near the feasible set, preventing overshoot.** The blue, yellow, and red regions correspond to cases in which the updates performed by the $\boldsymbol{\nu}$PI algorithm are faster, slower, or in the opposite direction than those of GA, respectively. This plot illustrates the case $\xi_{t-1} > 0$.

In stark contrast, Fig. 5 (right) shows a setting in which κ_i and κ_p have been chosen according to Thm. 1 for $\beta = \nu = 0.3$, i.e. using *positive* Polyak momentum. In this case, the algorithm would produce *stronger increases* of the multiplier whenever feasibility is improved, while *weaker increases* are executed whenever feasibility worsens. This counter-intuitive behavior may be the cause of oscillations and overshoot underlying the failure of positive momentum methods in Lagrangian games.

4.4. Oscillator dynamics

The continuous-time dynamics of gradient-descent/νPI-ascent on an equality-constrained problem can be characterized by the second-order differential equations (see Thm. 4):

$$\begin{cases} \ddot{\boldsymbol{x}} = -\left(\nabla^2 f + \sum_{c'} \mu_{c'} \nabla^2 \boldsymbol{h}_{c'}\right)\dot{\boldsymbol{x}} - \mathcal{J}\boldsymbol{h}\dot{\boldsymbol{\mu}} & \text{(11a)} \\ \ddot{\boldsymbol{\mu}} = \kappa_i \mathcal{J}\boldsymbol{h}^\top \dot{\boldsymbol{x}} + \kappa_p \mathcal{J}\boldsymbol{h}^\top \ddot{\boldsymbol{x}} + \kappa_p \boldsymbol{\Xi}, & \text{(11b)} \end{cases}$$

where $\boldsymbol{\Xi} = \left[\dot{\boldsymbol{x}}^\top \nabla^2 h_1 \dot{\boldsymbol{x}}, \ldots, \dot{\boldsymbol{x}}^\top \nabla^2 h_c \dot{\boldsymbol{x}}\right]^\top \in \mathbb{R}^c$.

In Appx. D we present the spectral analysis for the Lagrangian system associated with an equality-constrained quadratic program. In particular, we demonstrate how the continous-time $\boldsymbol{\nu}$PI algorithm can modify the eigenvalues of the system and transition between divergent, oscillatory, *critically damped* and overdamped behaviors. We show how these regime changes are controlled by the κ_p hyperparameter. Moreover, critical damping may require a non-zero value of κ_p, and is thus not achievable by GA.

4.5. Practical remarks

In practice, we suggest the initial condition $\boldsymbol{\xi}_0 = \boldsymbol{e}_0$, as it ensures that the first step of $\boldsymbol{\nu}$PI matches that of gradient

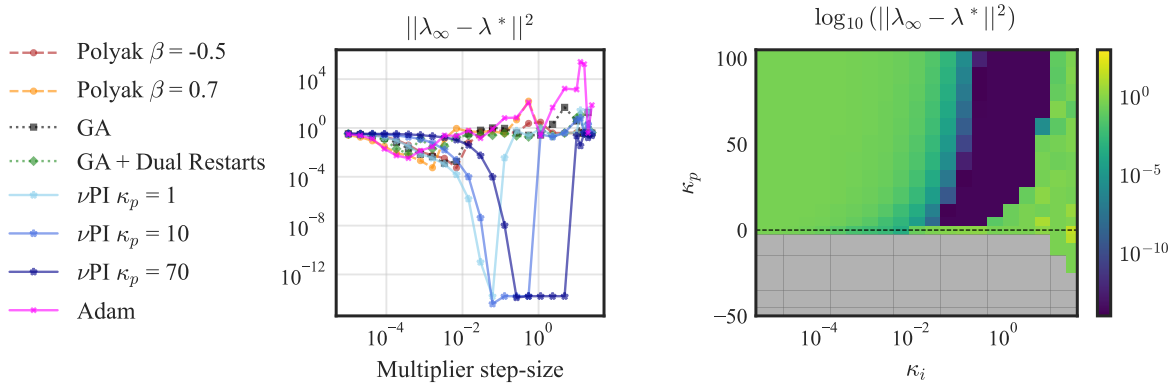


Figure 6: SVM experiment (§5.1). **Left:** Step-size sensitivity plot. **Right:** κ_p and κ_i sensitivity plot for νPI. **Without a κ_p of at least one, none of the methods converge to the optimal dual variable**. **Higher κ_p values allow for choosing higher and broader range of κ_i's.** The x-axis of the left plot represents κ_i for the νPI parameter and α, the step-size, for the other optimizers. In the right plot, the gray color shows the runs exceeding a distance of 10^3 to λ^*.

ascent. In cases where the constraints can be evaluated without noise, we suggest a default value of $\nu = 0$. This leaves only the additional hyperparameter κ_p to be tuned (besides the "step-size" κ_i). We highlight that the main benefits of the νPI algorithm remain even when $\nu = 0$. However, ν can be useful for filtering noise in the constraint measurement, as shown in our fairness experiments in §5.2.

There is a predictable monotonic behavior of the damping of the system as the κ_p coefficient increases. This is illustrated in Fig. 9 in §5.3 for a sparsity task. As a side effect, higher values of κ_p make the tuning of the κ_i coefficient easier, as seen in Fig. 6 in §5.1. As a heuristic to tune νPI, we suggest considering a large initial κ_p value (so that its influence on the optimization dynamics is significant), and then try a grid of κ_i values. A good starting place is a grid of κ_i values around a suitable step-size for gradient ascent.

5. Experiments

In this section, we present an empirical comparison between νPI and a series of baseline optimization methods popular for minimization. We consider gradient ascent, gradient ascent with positive (Polyak, 1964; Nesterov, 1983) and negative (Gidel et al., 2019a) momentum, and ADAM (Kingma & Ba, 2014). The goal of our experiments is to highlight the flexibility of νPI and its ability to mitigate oscillations and overshoot when used to optimize Lagrange multipliers.

Our implementations use PyTorch (Paszke et al., 2019) and the Cooper library for Lagrangian constrained optimization (Gallego-Posada & Ramirez, 2022).

5.1. Hard-margin SVMs

We consider solving a *hard-margin* linear SVM problem via its associated Lagrangian formulation. While specialized QP solvers exist to find solutions for this task, we consider the Lagrangian formulation in order to illustrate the dynamics of the multipliers in a simple machine learning task. These experiments show how standard methods for minimization produce oscillations on the multipliers, which have detrimental effects on convergence. Consider

$$\min_{\boldsymbol{w}} \|\boldsymbol{w}\|^2/2 \;\; \text{s.t.} \;\; y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1 \;\; \text{for } i \in [m], \quad (12)$$

where $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^m$ are labeled training datapoints, and $\boldsymbol{w}$ and b are the parameters of the SVM classifier.

We perform binary classification on two linearly separable classes from the Iris dataset (Fisher, 1988). We apply alternating GDA updates on the Lagrangian associated with Eq. (12), with a fixed optimizer for the primal variables. For details on our SVM experiments, see Appx. F.1.

Multiplier dynamics. Fig. 1 shows the oscillations on the multiplier in all of the baselines. In these tasks, all of the methods that do not diverge achieve perfect training and validation accuracy. However, among the methods we experiment with, the only method capable of achieving zero constraint violation is the νPI algorithm. In contrast to all baselines, νPI dampens the oscillations and converges to the optimal multiplier value.

Sensitivity analysis. Fig. 6 (left) illustrates the robustness of νPI to the choice of κ_i. The considered baselines fail to converge to the ground truth multiplier value, across a wide range of step-sizes. For these baselines, small step-size choices avoid divergence but do not lead to recovering the optimal Lagrange multipliers, while large step-sizes increase the oscillations. In contrast, introducing a κ_p term of more than 1 results in convergence for some step-sizes within the selected range (see the νPI curves). Moreover, increasing κ_p to a higher value broadens the range of step-sizes that lead to convergence, and enables the use of bigger step-size values that converge. This behavior can be observed more extensively in the heatmap of Fig. 6 (right).

5.2. Fairness

We consider a classification task under *statistical parity* constraints, as described in Cotter et al. (2019). This leads to the following constrained optimization problem:

$$\min_{\boldsymbol{w}} L(\boldsymbol{w}) \quad \text{s.t.} \quad \mathrm{P}(\hat{y}=1 \,|\, g) = \mathrm{P}(\hat{y}=1), \quad \forall g \in G \quad (13)$$

where $L(\boldsymbol{w})$ is the loss of model $\boldsymbol{w}$, $\hat{y}$ is the model prediction, and G represents the set of protected groups in the dataset. The constraints require the probability of positive prediction to be equal across all groups.

Model and data. We train binary classifiers on the Adult dataset (Becker & Kohavi, 1996). Groups correspond to the intersection of race (2 values) and gender (5 values), leading to 10 constraints. We use an MLP with two 100-dimensional hidden layers. Our experimental setup is similar to those of Zafar et al. (2019) and Cotter et al. (2019). However, in our setting, non-convexity precludes the use of specialized solvers (as done by Zafar et al. (2019)) and requires iterative optimization approaches.

Optimization configuration. We train the model using ADAM ($\alpha = 10^{-2}$) with a batch size of 512. To mitigate the noise in the estimation of the constraint satisfaction, we update the multipliers once every epoch, using the exact constraint measurement over the entire training set.

Results. Figure 7 includes training curves for experiments with GA and $\boldsymbol{\nu}$PI applied to the Lagrange multipliers. We report two of the multipliers, the model accuracy, and the maximum constraint violation (in absolute value).

For this task, gradient ascent is a strong baseline as it successfully reduces the violation of the constraints. Both GA and $\boldsymbol{\nu}$PI ($\nu = 0.99$) significantly improve compared to an *unconstrained* baseline which achieves a maximum violation of 20% (not shown in Fig. 7 for readability).

$\boldsymbol{\nu}$PI ($\nu = 0$) runs exhibit unstable multiplier dynamics as the noise of the constraints is amplified by the proportional term. During our experiments, we observed that when $\nu = 0$, larger κ_p values lead to noisier multipliers and unstable optimization. In contrast, **$\boldsymbol{\nu}$PI ($\nu = 0.99$) reduces the maximum violation faster and achieves better training accuracy (92.4% vs 89%).**

All experiments reach a final maximum violation of around 1.7%. We hypothesize that it is not possible to decrease this value further (while carrying out stochastic updates on the primal variables) since the constraint gradients may be misaligned across mini-batches.

Multiplier dynamics. As can be seen in the evolution of multipliers 2 and 7 shown in Fig. 7, $\boldsymbol{\nu}$PI yields multipliers that stabilize at their limiting values faster than those of GA.

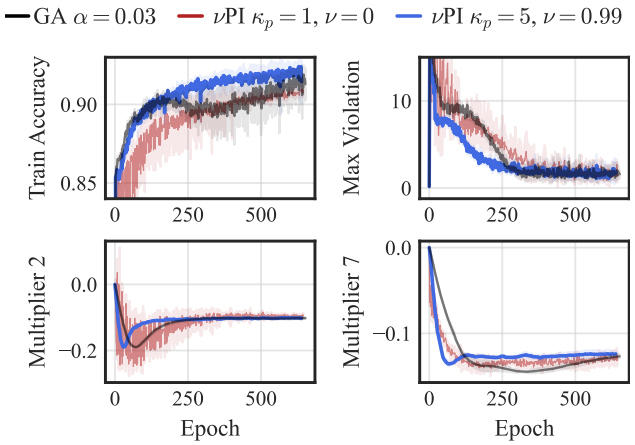


Figure 7: Dynamics of $\boldsymbol{\nu}$PI compared to GA for the fairness task in Eq. (13). **$\boldsymbol{\nu}$PI has faster convergence in the multiplier value and achieves a better training accuracy than GA.** All dual optimizers use a step-size (κ_i for $\boldsymbol{\nu}$PI) of 0.03. Results are aggregated across five seeds.

5.3. Sparsity

We consider the problem of learning models under inequality L_0-sparsity constraints (Louizos et al., 2018; Gallego-Posada et al., 2022). See Appx. F.2 for further background.

$$\min_{\boldsymbol{w},\boldsymbol{\phi}\in\mathbb{R}^d} \mathbb{E}_{\boldsymbol{z}|\boldsymbol{\phi}}\left[L(\boldsymbol{w}\odot\boldsymbol{z} \,|\, \mathcal{D})\right] \quad \text{s.t.} \quad \frac{\mathbb{E}_{\boldsymbol{z}|\boldsymbol{\phi}}[\|\boldsymbol{z}\|_0]}{\#(\boldsymbol{w})} \leq \epsilon \quad (14)$$

When using GA updates for the multipliers, Gallego-Posada et al. (2022) observe a tendency of the model to "overshoot" into the feasible region and become significantly less dense than the prescribed level. Since a reduction in model density corresponds to a reduction in capacity, this overshoot may have a detrimental effect on the performance of the model.

Our experiments explore the effect of $\boldsymbol{\nu}$PI on the sparsity-constrained task, and compare it with dual restarts (Gallego-Posada et al., 2022, §3). Our results show that $\boldsymbol{\nu}$PI allows for fine-grained control over overshoot, thus enabling the sparse model to retain as much performance as possible.

Experiment configuration and hyperparameters. We consider classifying CIFAR-10 (Krizhevsky, 2009) images with ResNet-18 (He et al., 2016) models. To highlight the ease-of-use of $\boldsymbol{\nu}$PI, our setup remains as close as possible to Gallego-Posada et al. (2022): we apply output channel sparsity on the first layer of each residual block in the model, and re-use the authors' choice of optimizer and step-size for $\boldsymbol{\phi}$. Our sparsity experiments consider $\nu = 0$.

Global and layer-wise settings. We present sparsity experiments with either ① one global constraint on the sparsity of the entire model, or ② multiple constraints, each prescribing a maximum density per layer.

The metrics reported in this section are aggregated across 5 seeds. Experimental details for this task can be found in Appx. F.2. For comprehensive experimental results across multiple sparsity levels, and ablations on the use of momentum and ADAM for updating the multipliers, see Appx. G.1.

Results. Fig. 8 shows how gradient ascent and positive and negative momentum values consistently yield runs that overshoot into becoming overly sparse. The extra reduction in capacity results in a loss in performance. In contrast, νPI consistently recovers feasible solutions, with minimal overshoot. While dual restarts do not incur in overshoot, they produce slightly infeasible solutions.

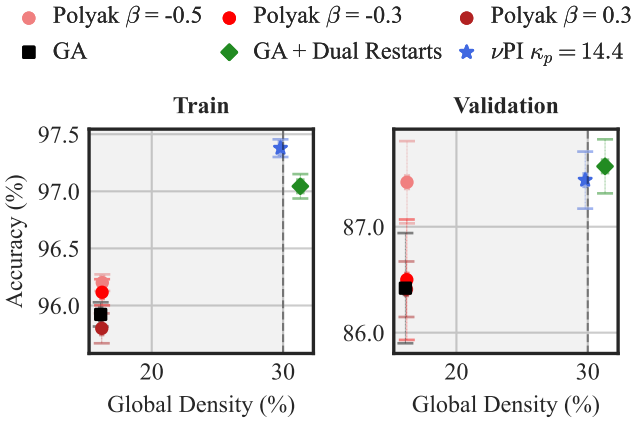


Figure 8: CIFAR10 trade-off plot for *global* sparsity under a 30% density target. **νPI successfully achieves the desired sparsity while achieving the highest train accuracy.** The shaded region is the feasible set. As higher density correlates to higher train accuracy, overshooting to a lower density is undesirable. All optimizers use the same step-size.

Fig. 9 consists on an ablation on the κ_p value. We observe that by increasing the hyper-parameter, overshoot is reduced, eventually turning into undershoot (which leads to infeasible solutions). Since the density of the model is monotonically tied to the choice of κ_p, tuning νPI for this task can be done via bisection search, without the need to consider a grid (which is usually required for tuning the step-size).

Table 1 shows sparsity experiments with layer-wise sparsity targets. Gradient ascent and momentum methods overshoot and the degree of overshoot differs significantly across layers. In contrast, GA with dual restarts and νPI mitigate overshoot and produce constraints spanning a narrow range of values. This highlights the robustness of νPI as the κ_p coefficient did not need to be tuned separately per constraint.

Multiplier dynamics. Figure 2 shows the training dynamics for a global sparsity constraint and its multiplier under a 30% density target. We observe that GA and POLYAK quickly lead to overshoot into the feasible set, but manage to regain some model capacity as training progresses. GA with dual restarts sets the value of the multiplier to zero as soon as feasibility is achieved, thus preventing an incursion of the constraint into the feasible set. νPI produces well-behaved multipliers and successfully avoids constraint overshoot.

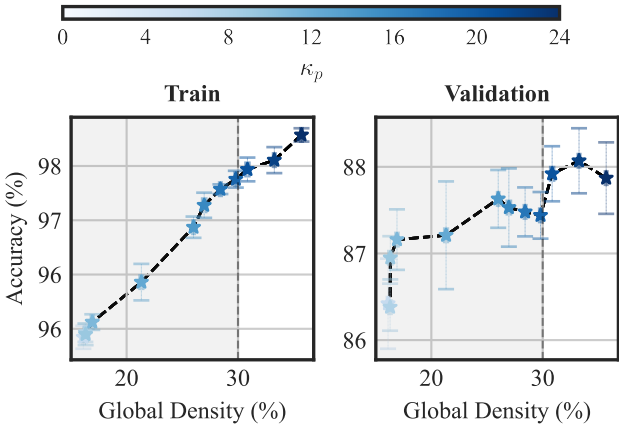


Figure 9: Ablation of κ_p values for νPI on CIFAR10. **An increasing κ_p leads to more damping and less overshoot.** Target density is 30%. The shaded region is the feasible set.

Table 1: CIFAR10 results for *layer-wise* sparsity under a 30% density target. GA and momentum methods overshoot to *different* values for each constraint. **νPI achieves the desired sparsity *on all layers* while achieving the highest train accuracy.** All dual optimizers use the same step-size.

Method	Accuracy		Violation		
	Train	Test	Min	Max	Range
Polyak $\beta = -0.5$	91.9	83.6	-26.5	-7.9	18.9
Polyak $\beta = -0.3$	92.1	83.4	-27.1	-6.7	20.6
Polyak $\beta = 0.3$	91.9	82.5	-26.3	-2.3	24.0
GA	92.0	84.1	-27.8	-5.2	22.0
GA + Dual Restarts	95.0	85.3	-0.0	1.2	1.2
Ours - νPI $\kappa_p = 8.0$	95.1	86.2	-1.7	0.1	1.8

6. Conclusion

Previous work has highlighted that employing PID controllers on the multipliers in Lagrangian constrained optimization problems reduces oscillation and overshoot. In this paper, we consider νPI, a variant of a PI controller that generalizes various popular methods for optimization. We complement previous work by providing insights justifying why PI controllers are desirable for Lagrangian optimization. Moreover, we highlight some intuitions as to why momentum methods fail in this context. While we focus our efforts on constrained optimization, our results indicate that νPI may improve the dynamics of linear players in general min-max games. Investigating the behavior of νPI on non-linear players is left as a direction of future work.

Impact Statement

Constrained optimization offers tools for reliably enforcing properties on machine learning models. It is, therefore, applicable for enhancing safety, robustness, and fairness in AI models. By integrating constraints into the model development process, rather than retrofitting safety measures as afterthoughts, we advocate for a paradigm shift towards building models that are inherently secure "by design." We intend our fairness experiments as a conceptual illustration of the potential for positive impact of constrained approaches in the development of machine learning models.

Our paper presents insights into the robustness of algorithms for constrained optimization, and highlights $\boldsymbol{\nu}$PI as a reliable tool for training models with constraints. Thus, our work lays the groundwork for practitioners to adopt and implement constrained approaches confidently in diverse real-world applications.

Acknowledgements

This research was partially supported by an IVADO PhD Excellence Scholarship, the Canada CIFAR AI Chair program (Mila), the NSERC Discovery Grant RGPIN2017-06936 ahd by Samsung Electronics Co., Ldt. Simon Lacoste-Julien is a CIFAR Associate Fellow in the Learning in Machines & Brains program.

This research was enabled in part by compute resources, software, and technical help provided by Mila.

We would like to thank Ioannis Mitliagkas for useful discussions during the development of this work.

Reproducibility Statement

We provide our code,[5] including scripts to replicate the experiments in this paper. §4.5 presents some considerations when using the $\boldsymbol{\nu}$PI algorithm in practice. Experimental details, as well as the hyper-parameters used in our experiments, are included in Appx. F. Our implementations use the open-source libraries PyTorch (Paszke et al., 2019) and Cooper (Gallego-Posada & Ramirez, 2022).

References

An, W., Wang, H., Sun, Q., Xu, J., Dai, Q., and Zhang, L. A PID Controller Approach for Stochastic Optimization of Deep Networks. In *CVPR*, 2018. (Cit. on p. 2, 5, 14)

Arrow, K., Hurwicz, L., and Uzawa, H. *Studies in Linear and Non-linear Programming*. Stanford University Press, 1958. (Cit. on p. 2, 3)

Åström, K. and Hägglund, T. *PID Controllers*. International Society for Measurement and Control, 1995. (Cit. on p. 2, 5)

Becker, B. and Kohavi, R. Adult. UCI Machine Learning Repository, 1996. DOI: https://doi.org/10.24432/C5XW20. (Cit. on p. 8, 24)

Bertsekas, D. On the Method of Multipliers for Convex Programming. *IEEE transactions on automatic control*, 1975. (Cit. on p. 2)

Bertsekas, D. *Nonlinear Programming*. Athena Scientific, 2016. (Cit. on p. 1, 2)

Bertsekas, D. P. On the Goldstein-Levitin-Polyak Gradient Projection Method. *IEEE Transactions on automatic control*, 1976. (Cit. on p. 2)

Bottou, L., Curtis, F. E., and Nocedal, J. Optimization Methods for Large-Scale Machine Learning. *SIAM Review*, 60 (2):223–311, 2018. (Cit. on p. 1)

Boyd, S. Stanford ENGR108: Introduction to applied linear algebra: 2020: Lecture 53-VMLS CSTRD nonlinear LS, 2021. URL https://youtu.be/SM_ZieyKicU?si=PWNMr7vxMQkhFBbf&t=815. (Cit. on p. 21)

Boyd, S. and Vandenberghe, L. *Convex Optimization*. Cambridge University Press, 2004. (Cit. on p. 1)

Casti, U., Bastianello, N., Carli, R., and Zampieri, S. A Control Theoretical Approach to Online Constrained Optimization. *arXiv preprint arXiv:2309.15498*, 2023. (Cit. on p. 2, 14)

Corbett-Davies, S., Pierson, E., Feller, A., Goel, S., and Huq, A. Algorithmic decision making and the cost of fairness. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017. (Cit. on p. 24)

Cotter, A., Jiang, H., Gupta, M. R., Wang, S., Narayan, T., You, S., and Sridharan, K. Optimization with Non-Differentiable Constraints with Applications to Fairness, Recall, Churn, and Other Goals. *JMLR*, 2019. (Cit. on p. 1, 8, 24)

Dahl, G. E., Schneider, F., Nado, Z., Agarwal, N., Sastry, C. S., Hennig, P., Medapati, S., Eschenhagen, R., Kasimbeg, P., Suo, D., Bae, J., Gilmer, J., Peirson, A. L., Khan, B., Anil, R., Rabbat, M., Krishnan, S., Snider, D., Amid, E., Chen, K., Maddison, C. J., Vasudev, R., Badura, M., Garg, A., and Mattson, P. Benchmarking Neural Network Training Algorithms. *arXiv:2306.07179*, 2023. (Cit. on p. 2, 3)

[5] https://github.com/motahareh-sohrabi/nuPI

Dikin, I. I. Iterative Solution of Problems of Linear and Quadratic Programming. In *Doklady Akademii Nauk*. Russian Academy of Sciences, 1967. (Cit. on p. 2)

Dwork, C., Hardt, M., Pitassi, T., Reingold, O., and Zemel, R. Fairness Through Awareness. In *Innovations in Theoretical Computer Science*, 2012. (Cit. on p. 24)

Elenter, J., NaderiAlizadeh, N., and Ribeiro, A. A Lagrangian Duality Approach to Active Learning. In *NeurIPS*, 2022. (Cit. on p. 1)

Farahmand, A.-M. and Ghavamzadeh, M. PID Accelerated Value Iteration Algorithm. In *ICML*, 2021. (Cit. on p. 1)

Fioretto, F., Van Hentenryck, P., Mak, T. W. K., Tran, C., Baldo, F., and Lombardi, M. Lagrangian Duality for Constrained Deep Learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2020. (Cit. on p. 1)

Fisher, R. A. Iris. UCI Machine Learning Repository, 1988. DOI: https://doi.org/10.24432/C56C76. (Cit. on p. 7, 22)

Frank, M. and Wolfe, P. An Algorithm for Quadratic Programming. *Naval Research Logistics Quarterly*, 1956. (Cit. on p. 2)

Gale, T., Elsen, E., and Hooker, S. The State of Sparsity in Deep Neural Networks. *arXiv:1902.09574*, 2019. (Cit. on p. 23)

Gallego-Posada, J. and Ramirez, J. Cooper: a toolkit for Lagrangian-based constrained optimization. https://github.com/cooper-org/cooper, 2022. (Cit. on p. 1, 7, 10, 22)

Gallego-Posada, J., Ramirez, J., Erraqabi, A., Bengio, Y., and Lacoste-Julien, S. Controlled Sparsity via Constrained Optimization or: *How I Learned to Stop Tuning Penalties and Love Constraints*. In *NeurIPS*, 2022. (Cit. on p. 1, 2, 4, 8, 23, 24)

Gidel, G., Askari, R., Pezeshki, M., LePriol, R., Huang, G., Lacoste-Julien, S., and Mitliagkas, I. Negative Momentum for Improved Game Dynamics. In *AISTATS*, 2019a. (Cit. on p. 2, 6, 7, 19, 20, 23)

Gidel, G., Berard, H., Vignoud, G., Vincent, P., and Lacoste-Julien, S. A Variational Inequality Perspective on Generative Adversarial Networks. In *ICLR*, 2019b. (Cit. on p. 2, 3)

Hashemizadeh, M., Ramirez, J., Sukumaran, R., Farnadi, G., Lacoste-Julien, S., and Gallego-Posada, J. Balancing Act: Constraining Disparate Impact in Sparse Models. In *ICLR*, 2024. (Cit. on p. 1)

He, K., Zhang, X., Ren, S., and Sun, J. Deep Residual Learning for Image Recognition. In *CVPR*, 2016. (Cit. on p. 8, 23)

Hounie, I., Elenter, J., and Ribeiro, A. Neural Networks with Quantization Constraints. In *ICASSP*, 2023. (Cit. on p. 1)

Hu, B. and Lessard, L. Control Interpretations for First-Order Optimization Methods. In *American Control Conference*, 2017. (Cit. on p. 2, 14)

Kingma, D. and Ba, J. Adam: A Method for Stochastic Optimization. In *ICLR*, 2014. (Cit. on p. 7, 23)

Korpelevich, G. M. The extragradient method for finding saddle points and other problems. *Matecon*, 1976. (Cit. on p. 2)

Krizhevsky, A. Learning Multiple Layers of Features from Tiny Images. Technical report, University of Toronto, Toronto, Ontario, 2009. (Cit. on p. 8)

Lessard, L., Recht, B., and Packard, A. Analysis and Design of Optimization Algorithms via Integral Quadratic Constraints. *SIAM Journal on Optimization*, 26(1):57–95, 2016. (Cit. on p. 5)

Lin, T., Jin, C., and Jordan, M. On Gradient Descent Ascent for Nonconvex-Concave Minimax Problems. In *ICML*, 2020. (Cit. on p. 2)

Loshchilov, I. and Hutter, F. SGDR: Stochastic Gradient Descent with Warm Restarts. In *ICLR*, 2017. (Cit. on p. 23)

Louizos, C., Welling, M., and Kingma, D. P. Learning Sparse Neural Networks through L_0 Regularization. In *ICLR*, 2018. (Cit. on p. 8, 23, 28)

Maddison, C. J., Mnih, A., and Teh, Y. W. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In *ICLR*, 2017. (Cit. on p. 23)

Mokhtari, A., Ozdaglar, A. E., and Pattathil, S. Convergence Rate of O(1/k) for Optimistic Gradient and Extragradient Methods in Smooth Convex-Concave Saddle Point Problems. *SIAM Journal on Optimization*, 2020a. (Cit. on p. 2)

Mokhtari, A., Ozdaglar, A. E., and Pattathil, S. A Unified Analysis of Extra-gradient and Optimistic Gradient Methods for Saddle Point Problems: Proximal Point Approach. In *AISTATS*, 2020b. (Cit. on p. 5)

Nesterov, Y. E. A method of solving a convex programming problem with convergence rate o\bigl(k^2\bigr). In *Doklady Akademii Nauk*, volume 269, pp. 543–547. Russian Academy of Sciences, 1983. (Cit. on p. 5, 7)

Nocedal, J. and Wright, S. J. *Numerical Optimization*. Springer, 2006. (Cit. on p. 1, 2)

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS*, 2019. (Cit. on p. 7, 10, 22)

Platt, J. C. and Barr, A. H. Constrained Differential Optimization. In *NeurIPS*, 1987. (Cit. on p. 2)

Polyak, B. T. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964. (Cit. on p. 5, 7)

Popov, L. D. A modification of the Arrow-Hurwicz method for search of saddle points. *Mathematical notes of the Academy of Sciences of the USSR*, 28:845–848, 1980. (Cit. on p. 2, 5)

Recht, B. The Best Things in Life Are Model Free. arg min, blog: https://archives.argmin.net/2018/04/19/pid/, 2018. (Cit. on p. 2)

Shen, L., Chen, C., Zou, F., Jie, Z., Sun, J., and Liu, W. A Unified Analysis of AdaGrad with Weighted Aggregation and Momentum Acceleration. In *IEEE TNNLS*, 2018. (Cit. on p. 5)

Stooke, A., Achiam, J., and Abbeel, P. Responsive Safety in Reinforcement Learning by PID Lagrangian Methods. In *ICML*, 2020. (Cit. on p. 1, 2, 4, 5, 14, 18)

Zafar, M. B., Valera, I., Gomez-Rodriguez, M., and Gummadi, K. P. Fairness Constraints: A Flexible Approach for Fair Classification. *JMLR*, 2019. (Cit. on p. 1, 8, 24)

Zhang, G. and Wang, Y. On the Suboptimality of Negative Momentum for Minimax Optimization. In *AISTATS*, 2021. (Cit. on p. 2)

Zhang, G., Wang, Y., Lessard, L., and Grosse, R. B. Near-optimal Local Convergence of Alternating Gradient Descent-Ascent for Minimax Optimization. In *AISTATS*, 2022. (Cit. on p. 2, 3)

Zoutendijk, G. *Methods of Feasible Directions: A Study in Linear and Non-linear Programming*. Elsevier Publishing Company, 1960. (Cit. on p. 2)

Appendix

Table of Contents

A. Further discussion on prior works using PID controls in optimization

- In Stooke et al. (2020), the authors focus almost exclusively on applying PID control to constrained reinforcement learning. The authors do not explore the optimization aspects of PID-based updates for Lagrange multipliers, which are the main focus of our work. Our key theoretical contribution (Thm. 1) shows that $\boldsymbol{\nu}$PI provides a generalization of momentum-based optimization techniques. Note that the controller considered by Stooke et al. (2020) is unable to generalize momentum methods. Thanks to the unifying framework provided by Thm. 1, we provide insights to understand why momentum fails at Lagrangian optimization tasks (Fig. 5). Moreover, our experiments encompass linear SVMs, sparsity, and fairness tasks, and are not restricted to reinforcement learning.

- An et al. (2018) propose directly updating the parameters of a neural network using a PID controller (for unconstrained minimization only). Their approach has not been widely adopted by the deep learning community, possibly due to the highly specialized training procedures that have been developed for training neural networks. Although connected due to their use of PID control, this paper is not directly relevant to our work as we limit our scope to not modifying the optimization protocol for the (primal) model parameters.

- The work of Casti et al. (2023) focuses on the theoretical aspects of using PID control for problems with linear constraints. Their analysis is not directly applicable to our setting since we are interested in general machine learning applications involving nonconvex constraints.

- Hu & Lessard (2017) present control interpretations of first-order optimization methods and show how worst-case convergence rates of optimization algorithms can be derived from a control theoretical perspective. The idea of examining a possible connection between our $\boldsymbol{\nu}$PI algorithm and other momentum methods was inspired by this work.

B. Connections between νPI and momentum methods

Table 2: Classical optimization methods as instances of $\boldsymbol{\nu}$PI $(\nu, \kappa_p, \kappa_i; \boldsymbol{\xi}_0)$.

Algorithm	$\boldsymbol{\xi}_0$	κ_p	κ_i	ν
UnifiedMomentum(α, β, γ)	$(1-\beta)\boldsymbol{e}_0$	$-\frac{\alpha\beta}{(1-\beta)^2}\left[1-\gamma(1-\beta)\right]$	$\frac{\alpha}{1-\beta}$	β
Polyak(α, β)	$(1-\beta)\boldsymbol{e}_0$	$-\frac{\alpha\beta}{(1-\beta)^2}$	$\frac{\alpha}{1-\beta}$	β
Nesterov(α, β)	$(1-\beta)\boldsymbol{e}_0$	$-\frac{\alpha\beta^2}{(1-\beta)^2}$	$\frac{\alpha}{1-\beta}$	β
PI	$\boldsymbol{e}_0$	κ_p	κ_i	0
OptimisticGradientAscent(α)	$\boldsymbol{e}_0$	α	α	0
$\boldsymbol{\nu}$PI$(\nu, \kappa_p, \kappa_i)$ in practice	$\mathbf{0}$	κ_i	κ_p	ν
GradientAscent(α)	–	0	α	0

Lemma 2. *The $\boldsymbol{\nu}$PI $(\nu, \kappa_p, \kappa_i; \boldsymbol{\xi}_0)$ algorithm can be equivalently expressed as the recursion:*

$$\boldsymbol{\theta}_1 = \boldsymbol{\theta}_0 + \kappa_i \boldsymbol{e}_0 + \kappa_p \boldsymbol{\xi}_0 \tag{15a}$$

$$\boldsymbol{\xi}_t = \nu \boldsymbol{\xi}_{t-1} + (1-\nu)\boldsymbol{e}_t \tag{15b}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \kappa_i \boldsymbol{e}_t + \kappa_p (1-\nu)\left(\boldsymbol{e}_t - \boldsymbol{\xi}_{t-1}\right) \text{ for } t \geq 1 \tag{15c}$$

Proof of Lemma 2. For $t \geq 1$, we have:

$$\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t \overset{(\nu\text{PI3})}{=} \kappa_p \boldsymbol{\xi}_t + \kappa_i \sum_{\tau=0}^{t} \boldsymbol{e}_\tau - \kappa_p \boldsymbol{\xi}_{t-1} - \kappa_i \sum_{\tau=0}^{t-1} \boldsymbol{e}_\tau = \kappa_i \boldsymbol{e}_t + \kappa_p (\boldsymbol{\xi}_t - \boldsymbol{\xi}_{t-1}) \overset{(\nu\text{PI2})}{=} \kappa_i \boldsymbol{e}_t + \kappa_p (1-\nu)(\boldsymbol{e}_t - \boldsymbol{\xi}_{t-1}) \tag{16}$$

□

Lemma 3. *The* UNIFIEDMOMENTUM$(\alpha, \beta, \gamma; \boldsymbol{\phi}_0 = \mathbf{0})$ *algorithm can be expressed as the single-parameter recurrence:*

$$\boldsymbol{\theta}_1 = \boldsymbol{\theta}_0 + \alpha(1+\beta\gamma)\boldsymbol{e}_0 \tag{17a}$$
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\boldsymbol{e}_t + \beta(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}) + \alpha\beta\gamma\left(\boldsymbol{e}_t - \boldsymbol{e}_{t-1}\right) \text{ for } t \geq 1. \tag{17b}$$

Proof of Lemma 3.

$$\boldsymbol{\phi}_1 \overset{(\mathrm{UM}_2)}{=} \beta\cancel{\boldsymbol{\phi}_0} + \alpha\boldsymbol{e}_0 = \alpha\boldsymbol{e}_0 \tag{18a}$$
$$\boldsymbol{\theta}_1 \overset{(\mathrm{UM}_3)}{=} \boldsymbol{\theta}_0 + \boldsymbol{\phi}_1 + \beta\gamma\left(\boldsymbol{\phi}_1 - \cancel{\boldsymbol{\phi}_0}\right) = \boldsymbol{\theta}_0 + \alpha(1+\beta\gamma)\boldsymbol{e}_0. \tag{18b}$$

$$\boldsymbol{\theta}_{t+1} \overset{(\mathrm{UM}_3)}{=} \boldsymbol{\theta}_t + \boldsymbol{\phi}_{t+1} + \beta\gamma\left(\boldsymbol{\phi}_{t+1} - \boldsymbol{\phi}_t\right) \tag{19a}$$
$$\overset{(\mathrm{UM}_2)}{=} \boldsymbol{\theta}_t + \beta\boldsymbol{\phi}_t + \alpha\boldsymbol{e}_t + \beta\gamma\left(\beta\boldsymbol{\phi}_t + \alpha\boldsymbol{e}_t - \beta\boldsymbol{\phi}_{t-1} - \alpha\boldsymbol{e}_{t-1}\right) \tag{19b}$$
$$= \boldsymbol{\theta}_t + \beta\boldsymbol{\phi}_t + \alpha\boldsymbol{e}_t + \beta\gamma\left(\beta(\boldsymbol{\phi}_t - \boldsymbol{\phi}_{t-1}) + \alpha(\boldsymbol{e}_t - \boldsymbol{e}_{t-1})\right) \tag{19c}$$
$$= \boldsymbol{\theta}_t + \alpha\boldsymbol{e}_t + \beta\left[\boldsymbol{\phi}_t + \gamma\beta(\boldsymbol{\phi}_t - \boldsymbol{\phi}_{t-1})\right] + \alpha\beta\gamma(\boldsymbol{e}_t - \boldsymbol{e}_{t-1}) \tag{19d}$$
$$\overset{(\mathrm{UM}_3)}{=} \boldsymbol{\theta}_t + \alpha\boldsymbol{e}_t + \beta\left(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}\right) + \alpha\beta\gamma(\boldsymbol{e}_t - \boldsymbol{e}_{t-1}). \tag{19e}$$

□

Theorem. 1 *Under the same initialization* $\boldsymbol{\theta}_0$, UNIFIEDMOMENTUM$(\alpha, \beta \neq 1, \gamma; \boldsymbol{\phi}_0 = \mathbf{0})$ *is a special case of* $\boldsymbol{\nu}$PI $(\nu, \kappa_p, \kappa_i; \boldsymbol{\xi}_0)$ *with the following hyperparameter choices:*

$$\nu = \beta \qquad \kappa_p = -\frac{\alpha\beta}{(1-\beta)^2}\left[1 - \gamma(1-\beta)\right] \qquad \kappa_i = \frac{\alpha}{1-\beta} \qquad \boldsymbol{\xi}_0 = (1-\beta)\boldsymbol{e}_0 \tag{20}$$

Proof of Thm. 1. We want to find values of ν, κ_p, κ_i and $\boldsymbol{\xi}_0$ such that the sequence of iterates produced by $\boldsymbol{\nu}$PI $(\nu, \kappa_p, \kappa_i; \boldsymbol{\xi}_0)$ satisfies Eq. (17b). For $t \geq 2$ we have:

$$\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t \overset{(17b)}{=} \alpha\boldsymbol{e}_t + \beta(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}) + \alpha\beta\gamma\left(\boldsymbol{e}_t - \boldsymbol{e}_{t-1}\right) \tag{21a}$$
$$\kappa_i\boldsymbol{e}_t + \kappa_p\left(\boldsymbol{\xi}_t - \boldsymbol{\xi}_{t-1}\right) \overset{(15c)}{=} \alpha\boldsymbol{e}_t + \beta\left(\kappa_i\boldsymbol{e}_{t-1} + \kappa_p\left(\boldsymbol{\xi}_{t-1} - \boldsymbol{\xi}_{t-2}\right)\right) + \alpha\beta\gamma\left(\boldsymbol{e}_t - \boldsymbol{e}_{t-1}\right) \tag{21b}$$

$$\boldsymbol{e}_t\left(\kappa_i - \alpha - \alpha\beta\gamma\right) + \boldsymbol{e}_{t-1}\left(-\beta\kappa_i + \alpha\beta\gamma\right) + \kappa_p\left[\boldsymbol{\xi}_t - \boldsymbol{\xi}_{t-1} - \beta\left(\boldsymbol{\xi}_{t-1} - \boldsymbol{\xi}_{t-2}\right)\right] = 0 \tag{22}$$

Several applications of the definition of $\boldsymbol{\xi}_t$ (line 2 in Algo. 1) give:

$$\boldsymbol{\xi}_t - \boldsymbol{\xi}_{t-1} - \beta\left(\boldsymbol{\xi}_{t-1} - \boldsymbol{\xi}_{t-2}\right) = (1-\nu)\left[\boldsymbol{e}_t - \boldsymbol{\xi}_{t-1}\right] - \beta\boldsymbol{\xi}_{t-1} + \beta\boldsymbol{\xi}_{t-2} \tag{23}$$
$$= (1-\nu)\boldsymbol{e}_t - (1+\beta-\nu)\boldsymbol{\xi}_{t-1} + \beta\boldsymbol{\xi}_{t-2} \tag{24}$$
$$= (1-\nu)\boldsymbol{e}_t - (1+\beta-\nu)\left[\nu\boldsymbol{\xi}_{t-2} + (1-\nu)\boldsymbol{e}_{t-1}\right] + \beta\boldsymbol{\xi}_{t-2} \tag{25}$$
$$= (1-\nu)\boldsymbol{e}_t - (1-\nu)(1+\beta-\nu)\boldsymbol{e}_{t-1} + (1-\nu)(\beta-\nu)\boldsymbol{\xi}_{t-2} \tag{26}$$

Thus we can re-arrange to get:

$$\begin{bmatrix}\boldsymbol{e}_t & \boldsymbol{e}_{t-1} & \boldsymbol{\xi}_{t-2}\end{bmatrix}\begin{bmatrix}\kappa_i + (1-\nu)\kappa_p - \alpha(1+\beta\gamma) \\ -\beta\kappa_i - (1-\nu)(1+\beta-\nu)\kappa_p + \alpha\beta\gamma \\ (1-\nu)(\beta-\nu)\kappa_p\end{bmatrix} = 0 \tag{27}$$

Therefore, from both algorithms coincide when the following system of equations is satisfied:

$$\kappa_i + (1-\nu)\kappa_p = \alpha(1+\beta\gamma) \tag{28a}$$
$$\beta\kappa_i + (1-\nu)(1-\nu+\beta)\kappa_p = \alpha\beta\gamma \tag{28b}$$
$$(1-\nu)(\beta-\nu)\kappa_p = 0 \tag{28c}$$

For $\beta \neq 1$, the solution to this system is given by:

$$\nu \leftarrow \beta \qquad \kappa_i \leftarrow \frac{\alpha}{1-\beta} \qquad \kappa_p \leftarrow -\frac{\alpha\beta}{(1-\beta)^2}\left[1-\gamma(1-\beta)\right] \tag{29}$$

Finally, we choose the initial condition $\boldsymbol{\xi}_0$ that ensures that the first two steps of the algorithms match (at $t=0$ and $t=1$). The first iterate of $\boldsymbol{\nu}$PI is given by $\boldsymbol{\theta}_1 = \boldsymbol{\theta}_0 + \kappa_i \boldsymbol{e}_0 + \kappa_p \boldsymbol{\xi}_0$ as per Eq. (15a). Meanwhile, the first iterate of UNIFIEDMOMENTUM is given by:

$$\boldsymbol{\theta}_1 \overset{(17a)}{=} \boldsymbol{\theta}_0 + \alpha(1+\beta\gamma)\boldsymbol{e}_0 \overset{(29)}{=} \boldsymbol{\theta}_0 + (\kappa_i + (1-\beta)\kappa_p)\boldsymbol{e}_0 = \boldsymbol{\theta}_0 + \kappa_i \boldsymbol{e}_0 + (1-\beta)\kappa_p \boldsymbol{e}_0 \tag{30}$$

Therefore, setting $\boldsymbol{\xi}_0 \leftarrow (1-\beta)\boldsymbol{e}_0$ makes both algorithms match in their first step at $t=0$.

The second iterate from UNIFIEDMOMENTUM is $\boldsymbol{\theta}_2 = \boldsymbol{\theta}_1 + \alpha\beta\left[1-\gamma(1-\beta)\right]\boldsymbol{e}_0 + \alpha\left[1-\gamma(1-\beta)\right]\boldsymbol{e}_1$. On the other hand, the second iterate of $\boldsymbol{\nu}$PI is $\boldsymbol{\theta}_2 = \boldsymbol{\theta}_1 + (\kappa_i + \kappa_p(1-\nu))\boldsymbol{e}_1 - \kappa_p(1-\beta)\boldsymbol{\xi}_0$. It is easy to see that, given the hyperparameter choices outlined above, both algorithms match at $t=1$.

An induction argument yields the equivalence between the algorithms. □

C. Interpreting the updates of νPI

Consider the execution of the algorithms $\boldsymbol{\nu}$PI $(\nu, \kappa_p, \kappa_i)$ and GA$(\alpha = \kappa_i)$ at time t, with updates given by:

$$\theta^{\nu\text{PI}}_{t+1} = \theta_t + \kappa_i e_t + \kappa_p(1-\nu)(e_t - \xi_{t-1}) \tag{31}$$

$$\theta^{\text{GA}}_{t+1} = \theta_t + \kappa_i e_t \tag{32}$$

Let $\psi = \frac{\kappa_p(1-\nu)}{\kappa_i + \kappa_p(1-\nu)}$. Note that whenever κ_p and κ_i are non-negative, $\psi \in [0,1]$. The ratio between these updates is:

$$\frac{\Delta\boldsymbol{\nu}\text{PI}}{\Delta\text{GA}} = \frac{\theta^{\nu\text{PI}}_{t+1} - \theta_t}{\theta^{\text{GA}}_{t+1} - \theta_t} = \frac{\kappa_i e_t + \kappa_p(1-\nu)(e_t - \xi_{t-1})}{\kappa_i e_t} = 1 + \frac{\kappa_p(1-\nu)}{\kappa_i} - \frac{\kappa_p(1-\nu)}{\kappa_i}\frac{\xi_{t-1}}{e_t} = \frac{1}{1-\psi}\left[1 - \frac{\psi\xi_{t-1}}{e_t}\right]. \tag{33}$$

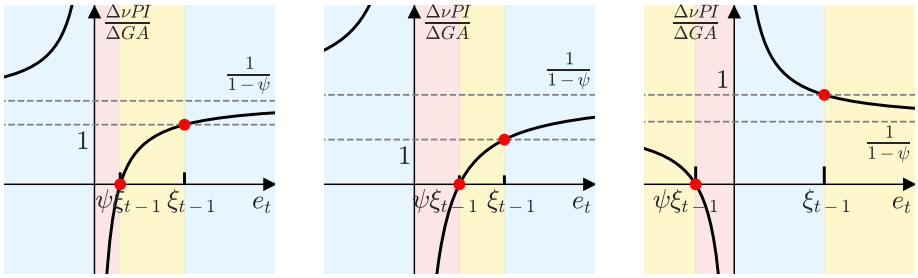


Figure 10: Comparing the update of $\boldsymbol{\nu}$PI relative to GA, for different hyper-parameter configurations of $\boldsymbol{\nu}$PI. **Left:** $\boldsymbol{\nu}$PI is configured to recover POLYAK$(\beta = -0.4)$. Updates exhibit dampening similar to that of $\boldsymbol{\nu}$PI $(\nu = 0, \kappa_p = 1, \kappa_i = 1)$. **Middle:** $\boldsymbol{\nu}$PI $(\nu = 0, \kappa_p = 1, \kappa_i = 1)$ corresponding to a PI controller. **$\boldsymbol{\nu}$PI increases the multipliers faster than GA when the constraint violation is large, enhancing convergence speed; and proactively decreases them near the feasible set, preventing overshoot. Right:** $\boldsymbol{\nu}$PI is configured to recover POLYAK$(\beta = 0.3)$. We observe an increased eagerness to increase the multipliers *as progress toward feasibility occurs*. This increases the chances of overshoot and subsequent oscillations. The blue, yellow, and red regions correspond to cases in which the updates performed by the νPI algorithm are faster, slower, or in the opposite direction than those of GA, respectively. This plot illustrates the case $\xi_{t-1} > 0$. The middle and right figures presented here are the same as those in Fig. 5. We include them here for the reader's convenience.

PI (Fig. 10, middle). We consider $\boldsymbol{\nu}$PI $(\nu = 0, \kappa_p = 1, \kappa_i = 1)$, which recovers PI$(\kappa_p = 1, \kappa_i = 1)$. The update of the PI optimizer relative to GA is as follows:

1. **Mode A** When either $e_t \geq \xi_{t-1}$ or $e_t \leq 0$, the relative update exceeds one and thus the PI controller update can be seen as *eager* compared to gradient ascent.
   (a) When $e_t \geq \xi_{t-1}$, the constraint has historically been infeasible and the current violation indicates an *increase in infeasibility*. In this case, PI not only increases the value of the multiplier but does so more strongly than GA. This proactive behavior serves to counteract the infeasibility increase.
   (b) When $e_t \leq 0$, the constraint has been satisfied despite historical infeasibility ($\xi_{t-1} > 0$). Here, the PI controller decreases the multiplier much more than GA. This serves to prevent overshoot into the feasible region.

2. **Mode B** In the range $0 < \psi\xi_{t-1} \leq e_t \leq \xi_{t-1}$, the constraint at step t (i) is not satisfied, (ii) it is smaller than the historical EMA of violations ξ_{t-1}, but not significantly (not beyond a factor of ψ). In this case, the PI controller proactively exerts *friction* by having a smaller update than GA. This reduces the risk of overshoot under the assumption that the primal variables continue to make progress toward feasibility.

3. **Mode C** In the *optimistic* phase, where $0 \leq e_t \leq \kappa\xi_{t-1}$, the PI controller's update goes in the opposite direction to that of GA: $\frac{\Delta\nu\text{PI}}{\Delta\text{GA}} \leq 0$. This corresponds to a scenario where the constraint made significant progress toward feasibility relative to the historic violation EMA. While GA would increase the multiplier in this case (since $g_t > 0$), PI *decreases* the value of the multiplier. This is useful to prevent overshoot since significant progress toward feasibility is an indicator that the multiplier is already exerting sufficient pressure for the constraints to be satisfied.

Negative momentum (Fig. 10, left). We consider POLYAK ($\beta = -0.4$) as a realization of $\boldsymbol{\nu}$PI, following Thm. 1. We observe similar behavior to that of $\boldsymbol{\nu}$PI ($\nu = 0, \kappa_p = 1, \kappa_i = 1$), in the middle figure of Fig. 10. Note that the current illustration assumes an equal value of the "optimizer state" ξ_{t-1} between the momentum and non-momentum cases. However, the value of ξ_t will be different depending on the momentum coefficient as $\beta = \nu$ also influences the update of ξ (see Algo. 1).

Positive momentum (Fig. 10, right). The right plot of Fig. 10 considers POLYAK ($\beta = 0.3$) as a realization of $\boldsymbol{\nu}$PI, following Thm. 1. We observe significantly different behavior compared to the left and middle plots.

1. **Mode A** When infeasibility is reduced, the algorithm is *eager* to increase the multiplier more than GA. This is a counter-intuitive operation of the algorithm considering that the current value of the multiplier can apply sufficient pressure to improve the constraint satisfaction. Increasing the multiplier further can lead to a higher risk of overshoot.

2. **Mode B** Consider the cases in which infeasibility increases ($e_t \geq \xi_{t-1}$), or the constraints suddenly become (sufficiently) strictly feasible $e_t \leq \psi\xi_{t-1} \leq 0$. These cases induce *frictioned* updates with the same sign as GA, but of smaller magnitude.

3. **Mode C** When the primal player is feasible, positive momentum would result in an *increase* of the multiplier; going against the update of GA, which would *decrease* the multiplier. In this context, increasing the multiplier is unreasonable since the current value of the multiplier is already sufficient to achieve feasibility.

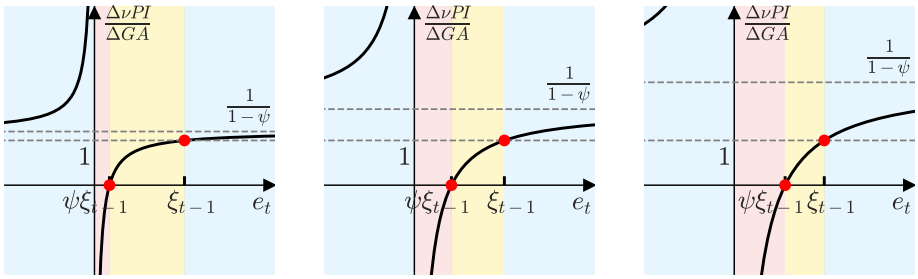


Figure 11: Effect of κ_p in the update of $\boldsymbol{\nu}$PI relative to GA. When κ_p approaches 0, $\boldsymbol{\nu}$PI recovers GA (a constant function $y = 1$ for the relative update). **A larger κ_p leads to a wider "optimistic" region (in red) where $\boldsymbol{\nu}$PI decreases the multiplier to prevent overshooting despite the constraint being violated.** We use $\kappa_i = 1$ and $\nu = 0$ and κ_p of 0.2, 0.7 and 1.3, respectively.

Ablation on the influence of κ_p. Figure 11 presents three configurations of κ_p for a $\boldsymbol{\nu}$PI ($\nu = 0, \kappa_p, \kappa_i = 1$) optimizer. We display κ_p at 0.2, 0.7 and 1.3, respectively. As $\kappa_p \to 0$, $\boldsymbol{\nu}$PI is equivalent to GA. This is confirmed by the relative updates between $\boldsymbol{\nu}$PI and GA converging to a constant function $y = 1$. As κ_p increases in the middle and right plots of Fig. 11, the asymptote at $1/(1-\psi)$ moves further away from 1, and the width of the "optimistic" region (Mode C) increases. In other words, as κ_p grows, the threshold for "sufficient improvement" is relaxed and the optimizer is more prone to decrease the multipliers upon improvements in constraint violation. This leads to a more "cautious" behavior from the algorithm: the multiplier is decreased earlier when the problem approaches the feasible region, which prevents overshooting but with potentially slower convergence. One can monotonically control for the convergence and overshoot behaviors by adjusting the κ_p value, see Fig. 9 in §5.3.

D. Analysis of continuous-time νPI dynamics as an oscillator

In this section, we examine the spectral properties of the gradient-descent/$\boldsymbol{\nu}$PI flow dynamics presented in Algo. 3. We extend the analysis of Stooke et al. (2020) (which only considers the dynamics of $\boldsymbol{x}$) to also consider $\boldsymbol{\lambda}$.

Consider a constrained optimization problem with equality constraints $\boldsymbol{h}$. The GD/νPI flow corresponds to a *continuous-time* dynamical system in which the primal player implements gradient descent on the Lagrangian, and the dual player implements νPI ascent. This is formalized in Algo. 3.

Algorithm 3 Continuous-time gradient descent/$\boldsymbol{\nu}$PI

Args: proportional (κ_p) and integral (κ_i) gains for $\boldsymbol{\nu}$PI flow
$\dot{\boldsymbol{x}} = -\nabla f(\boldsymbol{x}) - \mathcal{J}\boldsymbol{h}(\boldsymbol{x})\boldsymbol{\mu}$
$\dot{\boldsymbol{\mu}} = \kappa_i \boldsymbol{h}(\boldsymbol{x}) + \kappa_p \dot{\boldsymbol{h}}(\boldsymbol{x})$

Thm. 4 characterizes the GD/νPI flow in Algo. 3 in terms of a second-order dynamical system. Note that this relationship holds for any constrained problem where the objective and constraints have second derivatives. Appx. D.2 analyzes the resulting dynamical system for a quadratic program with linear equality constraints.

D.1. Oscillator dynamics of GD/νPI flow

Theorem 4. *The dynamics of Algo. 3 can be characterized by the following system of second-order differential equations, with initial conditions $\boldsymbol{x}(0) = \boldsymbol{x}_0$, $\boldsymbol{\mu}(0) = \boldsymbol{\mu}_0$, $\dot{\boldsymbol{x}}(0) = -\nabla f(\boldsymbol{x}_0) - \mathcal{J}\boldsymbol{h}(\boldsymbol{x}_0)\boldsymbol{\mu}_0$, and $\dot{\boldsymbol{\mu}}(0) = \kappa_i \boldsymbol{h}(\boldsymbol{x}_0) + \mathcal{J}\boldsymbol{h}(\boldsymbol{x}_0)\dot{\boldsymbol{x}}(0)$:*

$$\begin{cases} \ddot{\boldsymbol{x}} = -\underbrace{\left(\nabla^2 f + \sum_{c'} \mu_{c'} \nabla^2 \boldsymbol{h}_{c'}\right)}_{\boldsymbol{\Phi}} \dot{\boldsymbol{x}} - \mathcal{J}\boldsymbol{h}\dot{\boldsymbol{\mu}} & \text{(34a)} \\ \ddot{\boldsymbol{\mu}} = \kappa_i \mathcal{J}\boldsymbol{h}^\top \dot{\boldsymbol{x}} + \kappa_p \mathcal{J}\boldsymbol{h}^\top \ddot{\boldsymbol{x}} + \kappa_p \boldsymbol{\Xi} & \text{(34b)} \end{cases}$$

where $\boldsymbol{\Xi} = \left[\dot{\boldsymbol{x}}^\top \nabla^2 h_1 \dot{\boldsymbol{x}}, \ldots, \dot{\boldsymbol{x}}^\top \nabla^2 h_c \dot{\boldsymbol{x}}\right]^\top \in \mathbb{R}^c$.

This can be concisely represented in matrix form as:

$$\begin{bmatrix} \mathbf{I}_{n\times n} & \mathbf{0}_{n\times c} \\ -\kappa_p \mathcal{J}\boldsymbol{h}^\top & \mathbf{I}_{c\times c} \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{x}} \\ \ddot{\boldsymbol{\mu}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{\Phi} & \mathcal{J}\boldsymbol{h} \\ -\kappa_i \mathcal{J}\boldsymbol{h}^\top & \mathbf{0}_{c\times c} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{x}} \\ \dot{\boldsymbol{\mu}} \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ -\beta\boldsymbol{\Xi} \end{bmatrix} = \mathbf{0}. \tag{35}$$

Or, equivalently:

$$\begin{bmatrix} \ddot{\boldsymbol{x}} \\ \ddot{\boldsymbol{\mu}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{\Phi} & \mathcal{J}\boldsymbol{h} \\ \mathcal{J}\boldsymbol{h}^\top(\kappa_p\boldsymbol{\Phi} - \kappa_i \mathbf{I}) & \kappa_p \mathcal{J}\boldsymbol{h}^\top \mathcal{J}\boldsymbol{h} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{x}} \\ \dot{\boldsymbol{\mu}} \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ -\beta\boldsymbol{\Xi} \end{bmatrix} = \mathbf{0}. \tag{36}$$

Proof of Thm. 4. We start by computing the time derivatives of the objective gradient and constraint Jacobian:

$$\frac{d}{dt}[\nabla f] = \left[\sum_j \frac{\partial(\nabla f)}{\partial x_j}\frac{dx_j}{dt}\right]_i = \nabla^2 f \dot{\boldsymbol{x}} \qquad \frac{d}{dt}[\mathcal{J}\boldsymbol{h}] = \begin{bmatrix} \nabla^2 \boldsymbol{h}_1 \dot{\boldsymbol{x}} & \nabla^2 \boldsymbol{h}_2 \dot{\boldsymbol{x}} & \ldots & \nabla^2 \boldsymbol{h}_C \dot{\boldsymbol{x}} \end{bmatrix} \tag{37}$$

Therefore, the second order dynamics for $\boldsymbol{x}$ are given by:

$$\ddot{\boldsymbol{x}} = \frac{d}{dt}\left[-\nabla f(\boldsymbol{x}) - \mathcal{J}\boldsymbol{h}(\boldsymbol{x})\boldsymbol{\mu}\right] = -\frac{d}{dt}\left[\nabla f\right] - \mathcal{J}\boldsymbol{h}\dot{\boldsymbol{\mu}} - \frac{d}{dt}\left[\mathcal{J}\boldsymbol{h}\right]\boldsymbol{\mu} \tag{38a}$$

$$= -\nabla^2 f\dot{\boldsymbol{x}} - \mathcal{J}\boldsymbol{h}\dot{\boldsymbol{\mu}} - \sum_{c'}\mu_{c'}\nabla^2\boldsymbol{h}_{c'}\dot{\boldsymbol{x}} \tag{38b}$$

$$= -\underbrace{\left(\nabla^2 f + \sum_{c'}\mu_{c'}\nabla^2\boldsymbol{h}_{c'}\right)}_{\boldsymbol{\Phi}}\dot{\boldsymbol{x}} - \mathcal{J}\boldsymbol{h}\dot{\boldsymbol{\mu}} \tag{38c}$$

The second order dynamics for $\boldsymbol{\mu}$ are given by:

$$\ddot{\boldsymbol{\mu}} = \frac{d}{dt}\left[\kappa_i\boldsymbol{h} + \kappa_p\mathcal{J}\boldsymbol{h}^\top\dot{\boldsymbol{x}}\right] = \alpha\dot{\boldsymbol{h}} + \kappa_p\boldsymbol{\Xi}\dot{\boldsymbol{x}} + \kappa_p\mathcal{J}\boldsymbol{h}^\top\ddot{\boldsymbol{x}}, \tag{39}$$

where $\boldsymbol{\Xi}$ is defined as:

$$\boldsymbol{\Xi} \triangleq \frac{d}{dt}\left[\mathcal{J}\boldsymbol{h}^\top\right]\dot{\boldsymbol{x}} = \begin{bmatrix}\dot{\boldsymbol{x}}^\top\nabla^2\boldsymbol{h}_1\dot{\boldsymbol{x}} & \dot{\boldsymbol{x}}^\top\nabla^2\boldsymbol{h}_2\dot{\boldsymbol{x}} & \dots & \dot{\boldsymbol{x}}\nabla^2\boldsymbol{h}_c\dot{\boldsymbol{x}}\end{bmatrix}^\top. \tag{40}$$

□

D.2. Dynamics of GD/νPI flow for a constrained quadratic program

Let $\boldsymbol{H} \in \mathbb{R}^{n\times n}$ be positive semi-definite and consider the convex quadratic program with c linear constraints:

$$\min_{\boldsymbol{x}} \frac{1}{2}\boldsymbol{x}^\top\boldsymbol{H}\boldsymbol{x} + \boldsymbol{c}^\top\boldsymbol{x} \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} = 0. \tag{41}$$

The Lagrangian min-max game associated with the problem in Eq. (41) is given by:

$$\mathfrak{L}(\boldsymbol{x},\boldsymbol{\mu}) = \frac{1}{2}\boldsymbol{x}^\top\boldsymbol{H}\boldsymbol{x} + \boldsymbol{c}^\top\boldsymbol{x} + \boldsymbol{\mu}^\top(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) = \frac{1}{2}\boldsymbol{x}^\top\boldsymbol{H}\boldsymbol{x} + \boldsymbol{c}^\top\boldsymbol{x} + \boldsymbol{\mu}^\top\boldsymbol{A}\boldsymbol{x} - \boldsymbol{\mu}^\top\boldsymbol{b}. \tag{42}$$

The linearity of the constraints in Eq. (41) implies $\mathcal{J}\boldsymbol{h} = \boldsymbol{A}^\top$ and $\nabla^2 g_{c'} = \boldsymbol{0}$ for $c' \in [c]$, thus $\boldsymbol{\Phi} = \boldsymbol{H}$ and $\boldsymbol{\Xi} = \boldsymbol{0}$. Therefore, we obtain a homogeneous system of second-order differential equations with constant coefficients:

$$\begin{bmatrix}\ddot{\boldsymbol{x}}\\ \ddot{\boldsymbol{\mu}}\end{bmatrix} + \underbrace{\begin{bmatrix}\boldsymbol{H} & \boldsymbol{A}^\top\\ \boldsymbol{A}\left(\kappa_p\boldsymbol{H} - \kappa_i\mathbf{I}\right) & \kappa_p\boldsymbol{A}\boldsymbol{A}^\top\end{bmatrix}}_{U}\begin{bmatrix}\dot{\boldsymbol{x}}\\ \dot{\boldsymbol{\mu}}\end{bmatrix} = \boldsymbol{0}. \tag{43}$$

A simple state transformation $\boldsymbol{z} = [\boldsymbol{x},\boldsymbol{\mu},\dot{\boldsymbol{x}},\dot{\boldsymbol{\mu}}]^\top$ and $\dot{\boldsymbol{z}} = [\dot{\boldsymbol{x}},\dot{\boldsymbol{\mu}},\ddot{\boldsymbol{x}},\ddot{\boldsymbol{\mu}}]^\top$ yields:

$$\dot{\boldsymbol{z}} = -\begin{bmatrix}\boldsymbol{0} & \mathbf{I}\\ \boldsymbol{0} & \boldsymbol{U}\end{bmatrix}\boldsymbol{z} = -\begin{bmatrix}\boldsymbol{0}_{(n+c)\times(n+c)} & \mathbf{I}_{(n+c)\times(n+c)}\\ \boldsymbol{0}_{(n+c)\times(n+c)} & \begin{bmatrix}\boldsymbol{H} & \boldsymbol{A}^\top\\ \boldsymbol{A}\left(\kappa_p\boldsymbol{H} - \kappa_i\mathbf{I}\right) & \kappa_p\boldsymbol{A}\boldsymbol{A}^\top\end{bmatrix}\end{bmatrix}\boldsymbol{z} \tag{44}$$

Therefore, this $2(n+c)$-dimensional linear system has zero as an eigenvalue with algebraic multiplicity $n+c$, and the remaining eigenvalues correspond to the spectrum of $-\boldsymbol{U}$.

When the matrix $\boldsymbol{H}$ is zero, we recover the smooth bilinear games considered by Gidel et al. (2019a, Eq. 18) in their study of negative momentum. In this case, the matrix $\boldsymbol{U}$ looks like:

$$-\boldsymbol{U} = -\begin{bmatrix}\boldsymbol{0} & \boldsymbol{A}^\top\\ -\kappa_i\boldsymbol{A} & \kappa_p\boldsymbol{A}\boldsymbol{A}^\top\end{bmatrix} \tag{45}$$

It is easy to see that large enough values of κ_p cause the eigenvalues of the matrix to have negative real parts, and thus make the system converge. However, if $\kappa_p = 0$ (i.e. gradient descent-ascent), the eigenvalues of this matrix are either 0 or pure

imaginary. This fact is in line with existing results in the literature on the lack of convergence gradient descent-ascent on bilinear games (Gidel et al., 2019a).

Case of one-variable and one constraint. It is instructive to analyze the spectrum of $\boldsymbol{U}$ in the case of a problem with a one-dimensional primal variable and a single constraint (and thus one multiplier). In this case, $\boldsymbol{U}$ and its eigenvalues take the form:

$$-\boldsymbol{U} = -\begin{bmatrix} h & a \\ a\left(\kappa_p h - \kappa_i\right) & \kappa_p a^2 \end{bmatrix} \qquad \lambda = \frac{-\left(h + \kappa_p a^2\right) \pm \sqrt{\left(h + \kappa_p a^2\right)^2 - 4a^2\kappa_i}}{2} \tag{46}$$

As before, the eigenvalues of this matrix depend on the choice of κ_p. This is illustrated in Fig. 12.

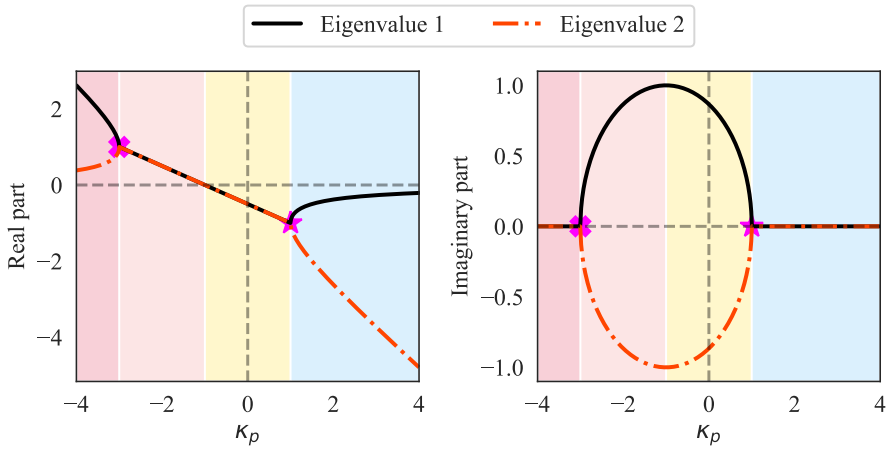


Figure 12: Eigenvalues for Eq. (41) as a function of κ_p in the one-dimensional case. **A positive value of κ_p (denoted by $\star$) achieves *critical damping* (i.e. equal convergence rate for both dimensions).** This plot uses $h = 1$, $a = -1$ and $\kappa_i = 1$.

Note that when the discriminant of Eq. (46) is zero, both eigenvalues match (and must thus be real). When this occurs and both eigenvalues are negative, the system converges and does so at the same rate in both dimensions. This is akin to the notion of *critical damping* from the control theory literature.

The values of κ_p that make the discriminant zero are $\kappa_p^* = \frac{-h \pm 2|a|\sqrt{\kappa_i}}{a^2}$, leading to the eigenvalues $\lambda(\kappa_p^*) = \frac{-\left(h + \kappa_p^* a^2\right)}{2} = \mp a\sqrt{\kappa_i}$. These values of κ_p^* are marked with $\star$ and $\times$ in Fig. 12. Note that out of the two values of κ_p producing matching eigenvalues, only the choice $\kappa_p^* > 0$ yields a convergent system.

More generally, depending on κ_p, the system exhibits different behaviors:

- **Divergence.** In the red regions, the system *diverges*; in light red region, this happens together with oscillations. Note how all the divergent configurations use a negative value of κ_p. The fuchsia cross ($\times$) denotes the value of κ_p for which both dimensions diverge at the same rate.

- **Underdamping.** In the yellow region, the system is *underdamped* and *converges with oscillations*. Interestingly, this system admits some negative values of κ_p (of sufficiently small magnitude) while remaining convergent.

- **Critical damping.** The fuchsia star ($\star$) shows the κ_p value that makes both dimensions of the system converge *at the same rate*. Note that this *critical damping* regime is achieved at a strictly positive value of κ_p, and thus is not achievable by gradient ascent.

- **Overdamping.** In the blue region, the system is *convergent without oscillations* but *overdamped* since the dimension corresponding to the black eigenvalue converges more slowly.

E. Illustrative 2D nonconvex problem

We demonstrate the behavior of $\boldsymbol{\nu}$PI on the two-dimensional, nonconvex, equality-constrained problem in Eq. (47). This problem was proposed by Boyd (2021). The setting is simple enough to allow for visualizing the optimization paths of each optimization variable and multiplier, while also being challenging due to nonconvexity.

$$\min_{\boldsymbol{x}=(x_1,x_2)} f(\boldsymbol{x}) \triangleq \left\| \begin{bmatrix} x_1 + e^{-x_2} \\ x_1^2 + 2x_2 + 1 \end{bmatrix} \right\|_2^2 \quad \text{subject to} \quad h(\boldsymbol{x}) \triangleq x_1 + x_1^3 + x_2 + x_2^2 = 2. \tag{47}$$

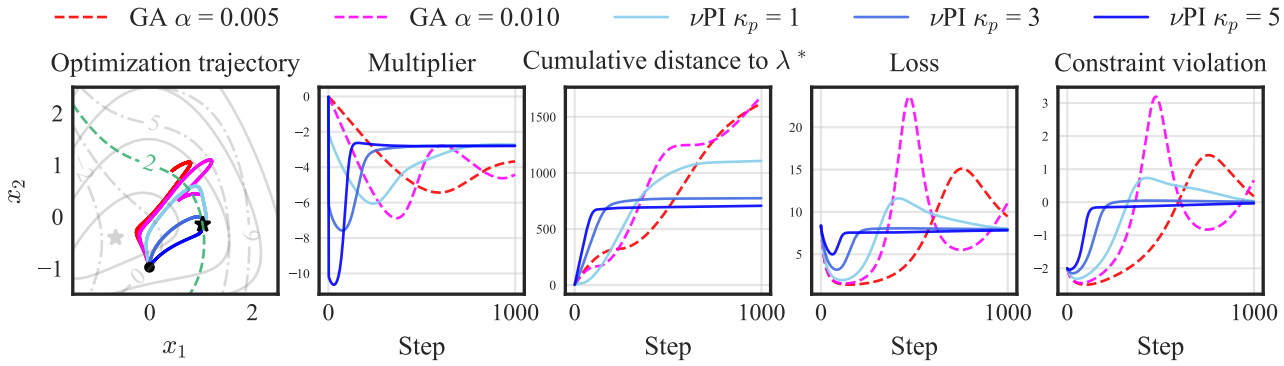


Figure 13: Optimization trajectories for different algorithms on a 2D nonconvex equality-constrained minimization problem. $\boldsymbol{\nu}$PI runs use $\nu = 0$ and $\kappa_i = 0.01$. The light gray $\star$ marks the *unconstrained* optimum, while the black $\star$ marks the *constrained* optimum. Level sets correspond to the objective function (solid) and constraint (dashed).

GA trajectories. In Fig. 13, GA trajectories are initially drawn toward the direction of the unconstrained optimum since multipliers grow slowly at first. As training progresses, the constraint plays a more significant role. With a step-size that is too small ($\alpha = 0.005$), the trajectory does not converge to the global optimum. In contrast, the system reaches the global constrained optimum point when employing a larger step-size ($\alpha = 0.01$). This is achieved while incurring in *oscillations*. The phase change from not converging with a small step-size, to converging with oscillations indicates that GA is not suitable for obtaining critical damping when solving the problem.

$\boldsymbol{\nu}$PI trajectories. The three blue trajectories in Fig. 13 show different behaviors of $\boldsymbol{\nu}$PI: underdamping (light blue, $\kappa_p = 1$), almost-critical damping ($\kappa_p = 3$) and overdamping (dark blue, $\kappa_p = 5$). Note the monotonic effect of κ_p on the damping of the system. $\boldsymbol{\nu}$PI provides the flexibility to obtain different levels of constraint overshoot, and can achieve feasibility and convergence at different speeds. This added flexibility leads to enhanced control over the dynamics of the system relative to GA, thus enabling applications of $\boldsymbol{\nu}$PI to safety-sensitive tasks.

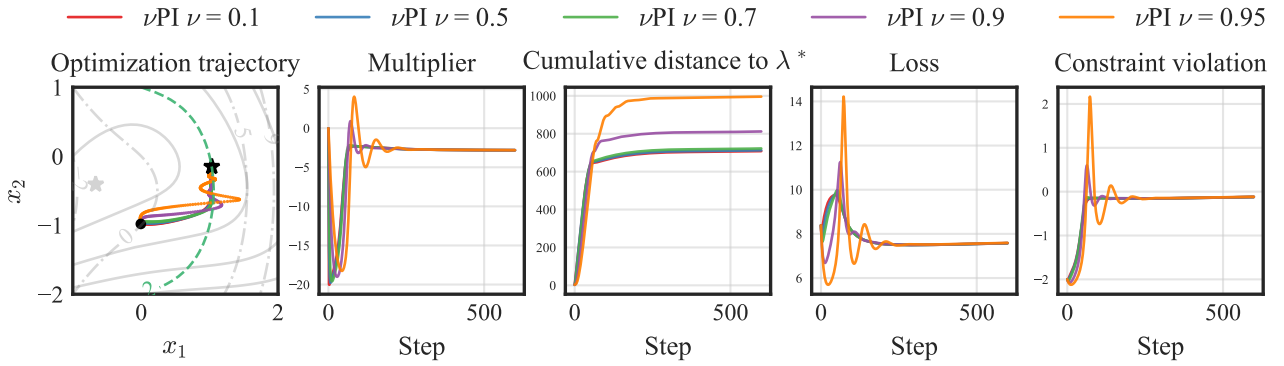


Figure 14: Optimization trajectories for the $\boldsymbol{\nu}$PI algorithm under different choices of ν. $\boldsymbol{\nu}$PI runs use $\kappa_i = 0.01$ and $\kappa_p = 10$.

Ablation on ν. In Fig. 14, we zoom in on the effect of ν for fixed choices of κ_p and κ_i. A ν closer to 0 tends towards PI, whereas a ν closer to one gives more importance to historical constraint violations. We observe that a larger ν behaves

qualitatively similar to positive momentum: the multiplier tends to increase faster if the constraint is not satisfied for a period of time. In this example, this leads to oscillations as shown for $\nu = 0.95$. Since this problem is deterministic, using a non-zero ν does not show any advantage. Our fairness experiments showcase an application where $\nu > 0$ is beneficial.

F. Experimental details

Our implementation use PyTorch 2.0.0 (Paszke et al., 2019) and the Cooper library for Lagrangian constrained optimization (Gallego-Posada & Ramirez, 2022). Our code is available at: `https://github.com/motahareh-sohrabi/nuPI`.

F.1. Linear SVM experiments

In our experiment with linear SVMs, we focus on two linearly separable classes from the Iris dataset (Fisher, 1988). We select 100 instances from the Iris setosa and Iris versicolor species, which are two linearly separable classes. Each data point in this dataset has four features. We selected 70% of data for training and the rest for validation. This gives the algorithm 70 Lagrange multipliers to learn.

We know that a unique λ^* exists in our experiments. The linearly independent constraint qualification (LICQ) holds for the selected data, so the Karush-Kuhn-Tucker (KKT) conditions imply the existence and uniqueness of λ^* at the constrained optimum x^*. All of the methods that do not diverge achieve perfect training and validation accuracy in this task.

Experiment configuration and hyperparameters. Throughout all of the experiments, we fixed the primal optimizer and only changed the dual optimizer. The primal optimizer is gradient descent with momentum, with a step size of 10^{-3} and momentum of 0.9.

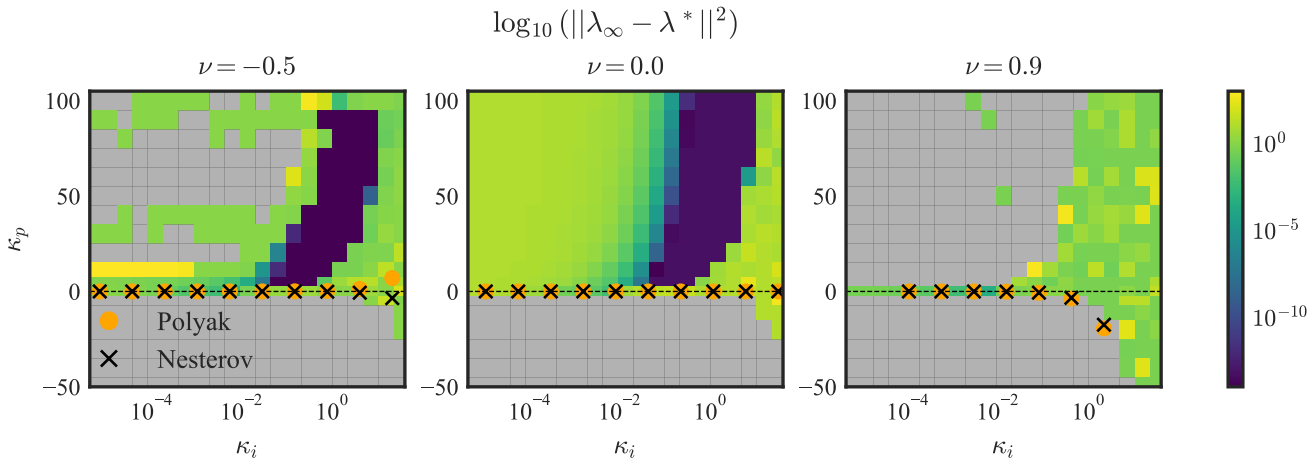


Figure 15: Distance to optimal Lagrange multipliers for different selections of parameter ν in νPI algorithm. We also show where the equivalent κ_p and κ_i parameters for NESTEROV$(\alpha, \beta = \nu)$ and POLYAK$(\alpha, \beta = \nu)$ lie according to Thm. 1 for different values of α. **The νPI algorithm with $\nu = 0$ can give the highest number of converging step-sizes. While negative $\nu = 0.5$ induces a range of converging step-sizes as well, there is no value of κ_i that the algorithm converges for $\nu = 0.9$.** The gray color shows the runs exceeding a distance of 10^3 to λ^*.

Different values of the parameter ν in νPI algorithm. We examine how changing the parameter ν in the Algo. 1 can affect the convergence of the SVM task with different choices of κ_i and κ_p. Fig. 15 shows how νPI behaves when choosing a negative, zero and positive value of ν. While $\nu = -0.5$ can lead to a converging algorithm for some step-sizes, $\nu = 0.0$ offers a wider range of converging step-sizes. There is no choice of step-size for which the νPI algorithm with a positive value of $\nu = 0.9$ converges to λ^*.

Relationship between momentum and νPI algorithms. Thm. 1 suggests that Polyak and Nesterov momentum algorithms can be instantiated using a specific choice of κ_i and κ_p in the Algo. 1. In Fig. 15 we show where POLYAK(α, β) and NESTEROV(α, β) lie for a choice of α's and with $\beta = \nu$. For each pair of (α_i, β), we calculate the value of κ_i and κ_p that recover POLYAK(α_i, β) and NESTEROV(α_i, β) according to Thm. 1.

- When $\beta = \nu = 0$ there is no momentum and both POLYAK and NESTEROV reduce to gradient ascent. Therefore, all of the dots indicating momentum methods lie in the $\kappa_p = 0$ line for the middle plot of Fig. 15.
- A positive $\beta = \nu = 0.9$ (Fig. 15, right) corresponds to a common momentum choice in minimization problems. We can see how there is no step-size value for which POLYAK or NESTEROV converge in this task. This observation supports the claim of Gidel et al. (2019a) on the ineffectiveness of positive momentum for convergence in games.
- A negative $\beta = \nu = -0.5$ leads to convergence for some choices of step-size. However, these do not correspond to what POLYAK and NESTEROV can achieve. This is consistent with our observation in Fig. 6 (left), indicating the necessity of adding a (positive) κ_p term to the optimizer in order to achieve convergence. This highlights the benefits of the increased generality of $\boldsymbol{\nu}$PI.

  Moreover, we observe that POLYAK with negative momentum achieves a positive κ_p while all momentum choices for NESTEROV lead to negative κ_p values. This further supports the experimental results of Gidel et al. (2019a), where POLYAK is used when they want to experiment with negative momentum. Our hypothesis is that negative momentum with Polyak is successful in games because it can induce a positive κ_p.

F.2. Sparsity experiments

Background. Louizos et al. (2018) propose a re-parameterization of models that allows applying L_0-norm regularization on their weights. They propose the use of stochastic gates $\boldsymbol{z}$ that indicate whether each parameter is active or not, where $\boldsymbol{z}$ follows a hard-concrete distribution parameterized by $\boldsymbol{\phi}$. Employing the re-parameterization trick allows the computation of gradients of the L_0-norm of the model with respect to $\boldsymbol{\phi}$. Gallego-Posada et al. (2022) formulate a constrained optimization problem that prescribes the desired sparsity of the model as a constraint.

$$\min_{\boldsymbol{w},\boldsymbol{\phi}\in\mathbb{R}^d} \mathbb{E}_{\boldsymbol{z}|\boldsymbol{\phi}}\left[L(\boldsymbol{w}\odot\boldsymbol{z}\,|\,\mathcal{D})\right] \quad \text{s.t.} \quad \frac{\mathbb{E}_{\boldsymbol{z}|\boldsymbol{\phi}}[\|\boldsymbol{z}\|_0]}{\#(\boldsymbol{w})} \leq \epsilon, \tag{48}$$

where $\boldsymbol{x}$ are the parameters of the model, L is an ERM objective, and $\mathcal{D}$ is a dataset. The constraint is normalized with the total number of parameters of the model $\#(\cdot)$, so that the constraint level $\epsilon \in [0, 1]$ corresponds to the maximum allowed *proportion* of active parameters. For details on the re-parameterization, and a closed form expression for $\mathbb{E}_{\boldsymbol{z}|\boldsymbol{\phi}}[\|\boldsymbol{z}\|_0]$, see Louizos et al. (2018); Gallego-Posada et al. (2022).

Hard-concrete distribution. The L_0-norm re-parameterization proposed by Louizos et al. (2018) considers a hard-concrete distribution for the stochastic gates of the model. The hard-concrete distribution is based on a stretched and clamped concrete distribution (Maddison et al., 2017). Similar to Louizos et al. (2018); Gallego-Posada et al. (2022), we choose a temperature of $2/3$ for the concrete distribution, and a stretching interval of $[-0.1, 1.1]$.

Architecture. We consider ResNet-18 (He et al., 2016) models with basic residual blocks for our sparsity experiments, which have a total of approximately 11.2 million parameters. Following Louizos et al. (2018) and Gallego-Posada et al. (2022), we employ output feature map sparsity on the first convolutional layer of each residual block, whereas the following convolutional layer and the residual connection are kept to be fully dense. The first convolutional layer of the model and the linear output layer are also kept fully dense. This model counts with 8 sparsifiable convolutional layers.

Choice of sparsity levels. Although Gallego-Posada et al. (2022) consider up to 80% structured sparsity (20% density) for ResNet-18 models, Gale et al. (2019) indicate that it is possible to train ResNet-50 models with structured sparsity of up to 95% (5% density or less), without incurring on a catastrophic loss on model accuracy. Therefore, we consider sparsity levels of between 30% and 90% (70% and 10% density, respectively).

Primal optimization. We consider an optimization pipeline for the model that incorporates standard techniques used to train L_0-sparse ResNet-18 models on CIFAR10. For the weights of the model, we use SGD with a momentum of 0.9, an initial learning rate of 0.01, and a cosine annealing learning rate scheduler (Loshchilov & Hutter, 2017).

We initialize the gates with a *droprate init* of 0.01, effectively yielding a fully dense model at initialization. Akin to Gallego-Posada et al. (2022), we use ADAM (Kingma & Ba, 2014) with a step-size of $8 \cdot 10^{-4}$ to optimize the $\boldsymbol{\phi}$ parameters of the stochastic gates. When applying L_2-norm regularization on the parameters, we detach the contribution of the gates as recommended by Gallego-Posada et al. (2022).

Dual optimization. For sparsity experiments, we consider $\nu = 0$. Since the constraint is deterministic given the state of the model (it does not need to be estimated from mini-batches), we consider the use of an EMA to not be crucial for

this task. Unless otherwise stated, we use a dual step-size of $8 \cdot 10^{-4}$ for all dual optimizer choices (as was provided by Gallego-Posada et al. (2022)). We decide against tuning the dual step-size separately for each optimizer to highlight the flexibility of $\boldsymbol{\nu}$PI: given a step-size that was tuned to yield good results for GA, $\boldsymbol{\nu}$PI may produce better-behaved dynamics.

All of our sparsity experiments use a batch size of 128 and are over 200 epochs.

F.3. Fairness experiments

Dataset. In this experiment we consider the adult dataset (Becker & Kohavi, 1996), pre-processed following Zafar et al. (2019). The raw data comprises eight categorical and six continuous attributes. After processing, the data is comprised of 50-dimensional sparse feature vectors. The train and test sets consist of 30,162 and 15,060 samples, respectively.

Background. We consider a fairness task under the disparate impact constraint (Zafar et al., 2019) shown in Eq. (13). This constraint is also known as statistical parity and demographic parity (Corbett-Davies et al., 2017; Dwork et al., 2012). We consider two sensitive attributes in the adult dataset: sex, denoted as $A_1 = \{\textit{male}, \textit{female}\}$, and race, denoted as $A_2 = \{\textit{White}, \textit{Black}, \textit{Asian-Pac-Islander}, \textit{Amer-Indian-Eskimo}, \textit{Other}\}$. Equation (13) entails the intersection of both attributes, leading to $|A_1| \times |A_2| = 10$ constraints.

Architecture and primal optimization. We train a 2-hidden-layer neural network with hidden sizes of (100, 100) similar to the experimental setup of Cotter et al. (2019). In order to choose the primal optimizer hyperparamters, we trained the unconstrained problem and chose the parameters of the run with the highest training accuracy. We fixed this primal optimizer across our constrained experiments to be ADAM ($\alpha = 10^{-2}$).

Dual optimization. We chose the best step-size for GA aimed at minimizing training accuracy, while ensuring that the maximum violation achieves the lowest possible value. This lead to a dual step-size of $\alpha = 0.03$. We then fixed this value as the κ_i parameter of $\boldsymbol{\nu}$PI and ran a grid search to find the best κ_p. The grid search for κ_p considered (logarithmically spaced) values in $[0.01, 100]$. The best results were found with $\kappa_p = 5$.

Due to the noise in the constraints, we also experimented with the effect of ν on the optimization dynamics. We tried ν values of 0.0, 0.5, 0.9, 0.95, and 0.99. We noticed that higher values of ν can improve the learning dynamics, with the best results achieved at 0.99. Setting $\nu = 0$ results in noisy Lagrange multipliers, which lead to unstable optimization. This is illustrated in Fig. 7.

G. Comprehensive results on the sparsity task

In this section we provide extensive experiment results for our sparsity experiments, complementing §5.3. We conducted experiments with global and layer-wise sparsity targets, at $\epsilon = 70\%, 50\%, 30\%, 10\%$ density levels. The shaded region of our plots corresponds to the feasible set. "Relative violations" are computed by dividing the absolute constraint violations by the target density.

G.1. Global

For global sparsity experiments (Figs. 16 to 19 and Tables 3 to 6), we observe a general trend for models that overshoot into becoming excessively sparse to achieve a lower training performance. This insight is also generally true for validation accuracy. In particular, gradient ascent and momentum methods consistently exhibit overshoot, whereas $\boldsymbol{\nu}$PI and gradient ascent with dual restarts do not overshoot and achieve good final performance. Dual restarts generally produce (slightly) infeasible solutions. Note that at $\epsilon = 10\%$, negative momentum runs do not overshoot, but positive momentum runs do.

G.2. Layer-wise

We perform layer-wise sparsity experiments with $\epsilon = 10\%, 30\%, 50\%, 70\%$ density targets (Tables 7 to 10). We observe a similar trend to global sparsity experiments: GA and momentum methods overshoot, while $\boldsymbol{\nu}$PI and GA with dual restarts reliably achieve feasible solutions, with small levels of overshoot. Moreover, we observe that the violations of $\boldsymbol{\nu}$PI and GA with dual restarts span a small range of values relative to other methods. This highlights the robustness of $\boldsymbol{\nu}$PI since the κ_p coefficient did not need to be tuned independently for each constraint.

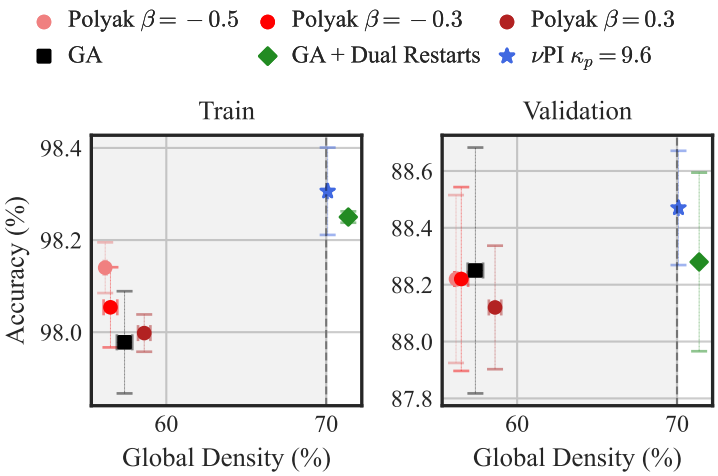


Figure 16: CIFAR10 trade-off plot for *global* sparsity under a 70% density target. **νPI successfully achieves the desired sparsity while achieving the highest train accuracy.** The shaded region is the feasible set. As higher density correlates to higher train accuracy, overshooting to a lower density is undesirable. All optimizers use the same step-size. *This figure is the same as Fig. 8. We repeat it here for the reader's convenience.*

Table 3: CIFAR10 results for *global* sparsity under a 70% density target. **νPI successfully achieves the desired sparsity while achieving the highest train accuracy.** The results in this table correspond to those in Fig. 16. As higher density correlates to higher train accuracy, overshooting to a lower density is undesirable. All optimizers use the same step-size.

Method	Train Acc.	Test Acc.	Violation	Relative Violation
Polyak $\beta = -0.5$	98.1 $\pm$ 0.06	88.2 $\pm$ 0.30	-13.8 $\pm$ 0.09	-19.7 $\pm$ 0.13
Polyak $\beta = -0.3$	98.1 $\pm$ 0.09	88.2 $\pm$ 0.32	-13.5 $\pm$ 0.43	-19.3 $\pm$ 0.61
Polyak $\beta = 0.3$	98.0 $\pm$ 0.04	88.1 $\pm$ 0.22	-11.4 $\pm$ 0.39	-16.3 $\pm$ 0.55
GA	98.0 $\pm$ 0.11	88.2 $\pm$ 0.43	-12.6 $\pm$ 0.48	-18.0 $\pm$ 0.69
GA + Dual Restarts	98.2 $\pm$ 0.01	88.3 $\pm$ 0.31	1.4 $\pm$ 0.07	2.0 $\pm$ 0.10
Ours - νPI $\kappa_p = 9.6$	98.3 $\pm$ 0.09	88.5 $\pm$ 0.20	0.1 $\pm$ 0.01	0.1 $\pm$ 0.02

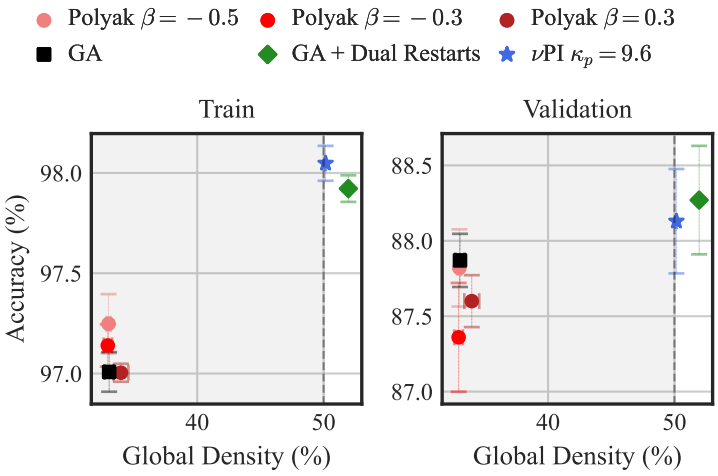


Figure 17: CIFAR10 trade-off plot for *global* sparsity under a 50% density target.

Table 4: CIFAR10 results for *global* sparsity under a 50% density target. The results in this table correspond to those in Fig. 17.

Method	Train Acc.	Test Acc.	Violation	Relative Violation
Polyak $\beta = -0.5$	97.2 $\pm$ 0.15	87.8 $\pm$ 0.26	-17.0 $\pm$ 0.17	-34.0 $\pm$ 0.34
Polyak $\beta = -0.3$	97.1 $\pm$ 0.11	87.4 $\pm$ 0.36	-17.1 $\pm$ 0.33	-34.2 $\pm$ 0.66
Polyak $\beta = 0.3$	97.0 $\pm$ 0.04	87.6 $\pm$ 0.17	-16.0 $\pm$ 0.59	-32.1 $\pm$ 1.18
GA	97.0 $\pm$ 0.10	87.9 $\pm$ 0.18	-17.0 $\pm$ 0.36	-33.9 $\pm$ 0.72
GA + Dual Restarts	97.9 $\pm$ 0.07	88.3 $\pm$ 0.36	2.0 $\pm$ 0.09	3.9 $\pm$ 0.18
Ours - νPI $\kappa_p = 9.6$	98.0 $\pm$ 0.09	88.1 $\pm$ 0.35	0.2 $\pm$ 0.03	0.3 $\pm$ 0.05

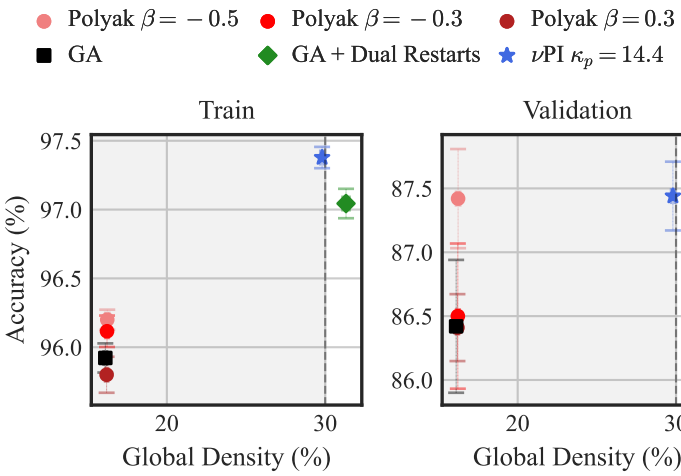


Figure 18: CIFAR10 trade-off plot for *global* sparsity under a 30% density target.

Table 5: CIFAR10 results for *global* sparsity under a 30% density target. The results in this table correspond to those in Fig. 18.

Method	Train Acc.	Test Acc.	Violation	Relative Violation
Polyak $\beta = -0.5$	96.2 $\pm$ 0.07	87.4 $\pm$ 0.39	-13.8 $\pm$ 0.13	-45.9 $\pm$ 0.43
Polyak $\beta = -0.3$	96.1 $\pm$ 0.11	86.5 $\pm$ 0.57	-13.8 $\pm$ 0.11	-45.9 $\pm$ 0.36
Polyak $\beta = 0.3$	95.8 $\pm$ 0.13	86.4 $\pm$ 0.26	-13.8 $\pm$ 0.09	-46.0 $\pm$ 0.31
GA	95.9 $\pm$ 0.11	86.4 $\pm$ 0.52	-13.9 $\pm$ 0.11	-46.3 $\pm$ 0.36
GA + Dual Restarts	97.0 $\pm$ 0.11	87.6 $\pm$ 0.26	1.3 $\pm$ 0.22	4.4 $\pm$ 0.73
Ours - νPI $\kappa_p = 14.4$	97.4 $\pm$ 0.08	87.4 $\pm$ 0.27	-0.2 $\pm$ 0.11	-0.7 $\pm$ 0.38

Table 6: CIFAR10 results for *global* sparsity under a 10% density target. The results in this table correspond to those in Fig. 19.

Method	Train Acc.	Test Acc.	Violation	Relative Violation
Polyak $\beta = -0.5$	94.6 $\pm$ 0.17	85.2 $\pm$ 0.93	0.7 $\pm$ 0.13	7.0 $\pm$ 1.31
Polyak $\beta = -0.3$	94.3 $\pm$ 0.06	84.7 $\pm$ 0.71	0.2 $\pm$ 0.14	2.0 $\pm$ 1.39
Polyak $\beta = 0.3$	92.6 $\pm$ 0.10	81.9 $\pm$ 0.77	-2.1 $\pm$ 0.08	-21.4 $\pm$ 0.82
GA	93.8 $\pm$ 0.20	81.9 $\pm$ 2.68	-0.9 $\pm$ 0.09	-9.0 $\pm$ 0.93
GA + Dual Restarts	94.3 $\pm$ 0.08	85.5 $\pm$ 0.59	0.4 $\pm$ 0.03	3.5 $\pm$ 0.35
Ours - νPI $\kappa_p = 1.6$	94.1 $\pm$ 0.08	83.4 $\pm$ 1.83	-0.2 $\pm$ 0.14	-1.6 $\pm$ 1.44

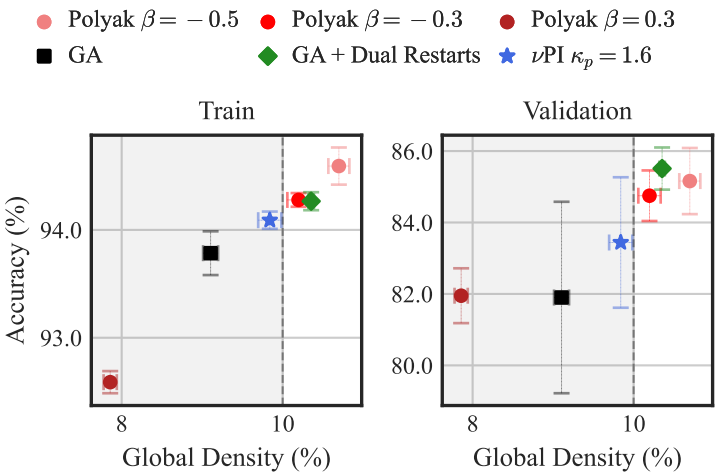


Figure 19: CIFAR10 trade-off plot for *global* sparsity under a 10% density target.

Table 7: CIFAR10 results for *layer-wise* sparsity under a 70% density target. GA and momentum methods overshoot to *different* values for each constraint. **νPI achieves the desired sparsity *on all layers* while achieving the highest train accuracy.** As higher density correlates to higher train accuracy, overshooting to a lower density is undesirable. All optimizers use the same step-size. *This table is the same as Table 1. We repeat it here for the reader's convenience.*

Method	Accuracy		Violation			Relative Violation		
	Train	Test	Min	Max	Range	Min	Max	Range
Polyak $\beta = -0.5$	91.9 $\pm$ 0.18	83.6 $\pm$ 1.40	-26.5 $\pm$ 0.81	-7.9 $\pm$ 0.86	18.9 $\pm$ 1.31	-37.8 $\pm$ 1.15	-11.4 $\pm$ 1.22	27.0 $\pm$ 1.87
Polyak $\beta = -0.3$	92.1 $\pm$ 0.07	83.4 $\pm$ 1.44	-27.1 $\pm$ 0.73	-6.7 $\pm$ 0.38	20.6 $\pm$ 0.92	-38.8 $\pm$ 1.05	-9.6 $\pm$ 0.55	29.4 $\pm$ 1.31
Polyak $\beta = 0.3$	91.9 $\pm$ 0.20	82.5 $\pm$ 1.50	-26.3 $\pm$ 0.82	-2.3 $\pm$ 0.69	24.0 $\pm$ 0.88	-37.5 $\pm$ 1.17	-3.2 $\pm$ 0.99	34.3 $\pm$ 1.26
GA	92.0 $\pm$ 0.08	84.1 $\pm$ 1.97	-27.8 $\pm$ 0.49	-5.2 $\pm$ 0.39	22.0 $\pm$ 0.56	-39.6 $\pm$ 0.70	-7.4 $\pm$ 0.55	31.4 $\pm$ 0.80
GA + Dual Restarts	95.0 $\pm$ 0.22	85.3 $\pm$ 0.61	-0.0 $\pm$ 0.00	1.2 $\pm$ 0.28	1.2 $\pm$ 0.28	-0.0 $\pm$ 0.00	1.8 $\pm$ 0.40	1.8 $\pm$ 0.40
Ours - νPI $\kappa_p = 8.0$	95.1 $\pm$ 0.06	86.2 $\pm$ 0.46	-1.7 $\pm$ 0.27	0.1 $\pm$ 0.04	1.8 $\pm$ 0.29	-2.4 $\pm$ 0.38	0.2 $\pm$ 0.06	2.5 $\pm$ 0.42

Table 8: CIFAR10 results for *layer-wise* sparsity under a 50% density target. As higher density correlates to higher train accuracy, overshooting to a lower density is undesirable. All optimizers use the same step-size.

Method	Accuracy		Violation			Relative Violation		
	Train	Test	Min	Max	Range	Min	Max	Range
Polyak $\beta = -0.5$	87.5 $\pm$ 0.17	80.2 $\pm$ 2.65	-32.6 $\pm$ 0.95	-15.9 $\pm$ 0.80	16.4 $\pm$ 1.31	-65.1 $\pm$ 1.89	-31.7 $\pm$ 1.59	32.9 $\pm$ 2.61
Polyak $\beta = -0.3$	87.7 $\pm$ 0.21	80.3 $\pm$ 2.13	-29.5 $\pm$ 0.69	-15.4 $\pm$ 0.81	13.6 $\pm$ 1.23	-59.1 $\pm$ 1.38	-30.8 $\pm$ 1.62	27.2 $\pm$ 2.46
Polyak $\beta = 0.3$	87.4 $\pm$ 0.21	79.7 $\pm$ 3.18	-30.9 $\pm$ 0.66	-14.1 $\pm$ 0.25	16.9 $\pm$ 0.70	-61.8 $\pm$ 1.33	-28.3 $\pm$ 0.49	33.9 $\pm$ 1.40
GA	87.6 $\pm$ 0.12	77.5 $\pm$ 3.14	-29.7 $\pm$ 1.02	-14.2 $\pm$ 0.60	14.7 $\pm$ 1.15	-59.4 $\pm$ 2.04	-28.3 $\pm$ 1.20	29.4 $\pm$ 2.30
GA + Dual Restarts	92.8 $\pm$ 0.07	83.5 $\pm$ 0.81	-0.0 $\pm$ 0.01	1.0 $\pm$ 0.46	1.0 $\pm$ 0.46	-0.0 $\pm$ 0.02	1.9 $\pm$ 0.92	1.9 $\pm$ 0.93
Ours - νPI $\kappa_p = 8.0$	93.2 $\pm$ 0.06	83.6 $\pm$ 0.87	-1.5 $\pm$ 0.13	0.1 $\pm$ 0.08	1.6 $\pm$ 0.19	-2.9 $\pm$ 0.26	0.2 $\pm$ 0.16	3.2 $\pm$ 0.37

H. Additional Experiments

In this section, we include additional experimental results on the sparsity-constrained task. We analyze the dynamics of the multiplier throughout training in Appx. H.1, and conduct ablation studies on κ_p for νPI, the momentum coefficients of POLYAK and NESTEROV, and the step-size of ADAM.

H.1. Dynamics

The dynamics shown in Fig. 20 illustrate the change of the constraint violation and multipliers throughout optimization. We observe that GA, POLYAK, and ADAM quickly lead to overshoot into the feasible region, leading to overly sparse models.

Table 9: CIFAR10 results for *layer-wise* sparsity under a 30% density target. As higher density correlates to higher train accuracy, overshooting to a lower density is undesirable. All optimizers use the same step-size.

Method	Accuracy		Violation			Relative Violation		
	Train	Test	Min	Max	Range	Min	Max	Range
Polyak $\beta = -0.5$	81.8 ± 0.19	63.5 ± 18.58	-25.2 ± 1.54	-17.0 ± 0.37	8.4 ± 1.73	-84.1 ± 5.12	-56.8 ± 1.23	28.0 ± 5.77
Polyak $\beta = -0.3$	82.1 ± 0.54	63.3 ± 8.65	-25.1 ± 1.15	-16.4 ± 0.38	8.7 ± 0.91	-83.5 ± 3.84	-54.7 ± 1.27	29.0 ± 3.04
Polyak $\beta = 0.3$	81.8 ± 0.32	72.7 ± 3.36	-25.1 ± 2.12	-17.5 ± 0.24	7.4 ± 2.12	-83.6 ± 7.07	-58.5 ± 0.79	24.8 ± 7.07
GA	81.8 ± 0.44	72.7 ± 4.40	-24.8 ± 1.11	-17.0 ± 0.60	8.5 ± 1.22	-82.5 ± 3.69	-56.7 ± 1.99	28.2 ± 4.07
GA + Dual Restarts	89.7 ± 0.23	82.9 ± 2.59	-0.0 ± 0.00	0.9 ± 0.33	0.9 ± 0.33	-0.0 ± 0.01	3.0 ± 1.10	3.0 ± 1.10
Ours - νPI $\kappa_p = 12.0$	89.8 ± 0.11	82.0 ± 2.45	-0.3 ± 0.13	0.3 ± 0.03	0.6 ± 0.12	-0.8 ± 0.42	1.0 ± 0.11	2.1 ± 0.39

Table 10: CIFAR10 results for *layer-wise* sparsity under a 10% density target. As higher density correlates to higher train accuracy, overshooting to a lower density is undesirable. All optimizers use the same step-size.

Method	Accuracy		Violation			Relative Violation		
	Train	Test	Min	Max	Range	Min	Max	Range
Polyak $\beta = -0.5$	71.3 ± 0.61	61.0 ± 9.50	-10.0 ± 0.14	-5.9 ± 0.47	4.0 ± 0.48	-100.0 ± 1.36	-58.7 ± 4.74	40.5 ± 4.83
Polyak $\beta = -0.3$	70.9 ± 0.60	49.5 ± 16.33	-10.0 ± 0.01	-5.9 ± 0.60	4.1 ± 0.60	-100.0 ± 0.11	-58.9 ± 5.97	41.1 ± 5.95
Polyak $\beta = 0.3$	69.2 ± 0.71	56.3 ± 15.05	-10.0 ± 0.02	-6.7 ± 0.06	3.3 ± 0.08	-100.0 ± 0.15	-67.3 ± 0.65	32.7 ± 0.79
GA	71.0 ± 0.32	49.6 ± 11.1	-10.0 ± 0.19	-6.1 ± 0.25	3.9 ± 0.42	-100.0 ± 1.91	-61.2 ± 2.54	38.8 ± 4.24
GA + Dual Restarts	83.1 ± 0.27	73.1 ± 4.87	-0.0 ± 0.00	1.6 ± 0.14	1.6 ± 0.14	-0.0 ± 0.02	16.1 ± 1.39	16.1 ± 1.40
Ours - νPI $\kappa_p = 12.0$	81.4 ± 0.39	42.8 ± 14.54	-1.9 ± 0.34	0.9 ± 0.46	3.2 ± 0.72	-19.1 ± 3.41	9.3 ± 4.62	31.7 ± 7.17

As training progresses however, these methods move closer to the boundary of the feasible region, reversing the initial overshoot. This recovery is most notorious for ADAM, whose multiplier decreases quickly after feasibility. GA with dual restarts sets the value of the multiplier to zero as soon as feasibility is achieved, thus preventing an incursion into the feasible set. $\boldsymbol{\nu}$PI produces well-behaved multipliers and successfully avoids constraint overshoot.

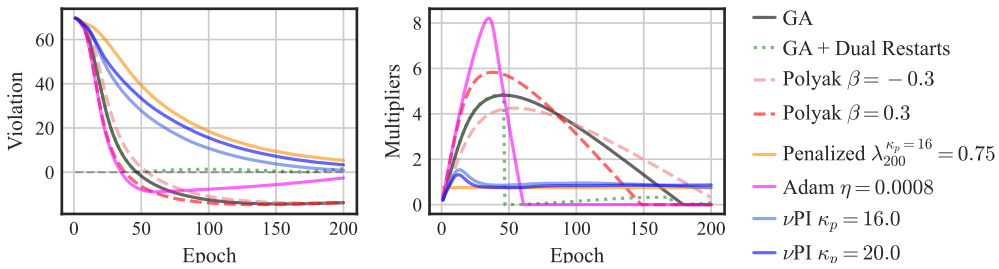


Figure 20: Dynamics plot for global sparsity under a 30% density target.

Note that for this sparsity task, it is reasonable to expect that the constraint is active at the constrained optimum since more model capacity correlates with better performance. However, note that $\boldsymbol{\nu}$PI is the only method that provides a non-zero estimate of the Lagrange multiplier. The usefulness of Lagrange multiplier estimates is highlighted in Fig. 21.

Figure 21 considers unconstrained L_0-regularization experiments. We use the final value of the multipliers corresponding to $\boldsymbol{\nu}$PI ($\kappa_p = 16$) and $\boldsymbol{\nu}$PI ($\kappa_p = 20$) runs as the (fixed) penalty coefficient for 200 epochs in the penalized formulation of the problem, akin to Louizos et al. (2018). We also include an experiment using the multiplier estimate from GA(equal to zero).

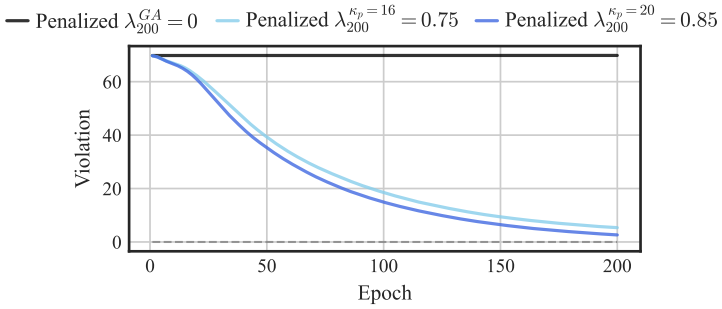


Figure 21: Dynamics plot for global sparsity under a 30% density target.

Unsurprisingly, the run with the multiplier of zero leads to 100% density, since the sparsity penalty does not exert any influence during training. In contrast, the runs with the νPI multiplier estimates not only lead to sparse models but are also very close to the desired model density by the end of training. This is remarkable since the problem we are solving is nonconvex, and optimal Lagrange multiplier values may not even exist.

H.2. Ablation on the value of κ_p

In this section, we fix κ_i for νPI and ablate on the hyperparameter κ_p for two sparsity levels. The results are presented in Fig. 22 and Tables 11 and 12.

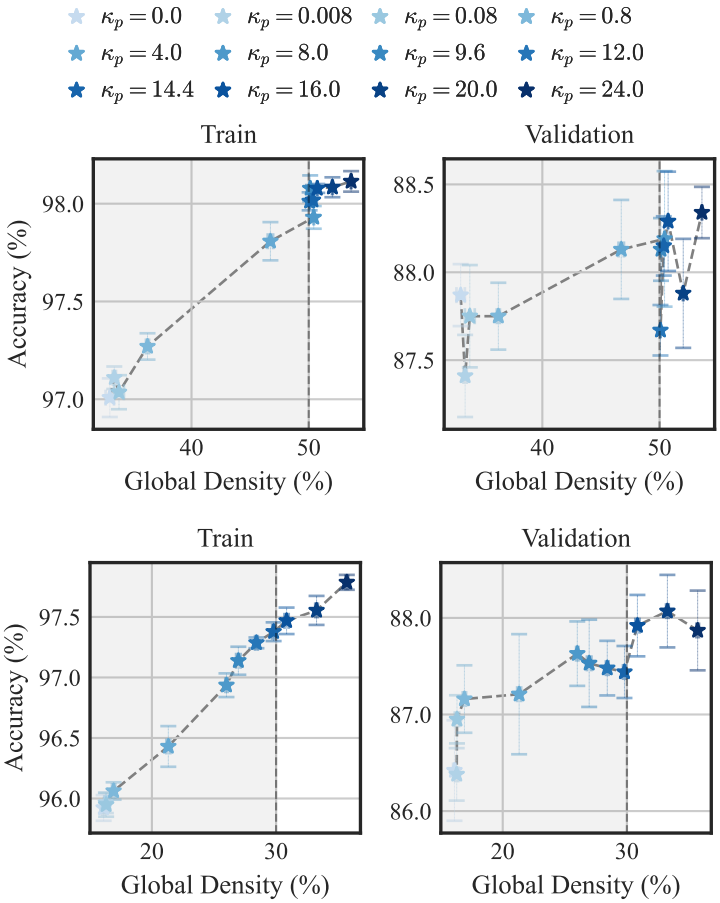


Figure 22: Ablation on trade-offs achievable by νPI under global density targets of 50% (top) and 30% (bottom).

We see that a larger κ_p leads to more damping and less overshoot. Note that there is a strong correlation between training accuracy and model density. Hence, it is important to be able to control overshoot in sparsity constraints and take advantage of the maximum allowed density for the sake of accuracy. There is a range of κ_p that can achieve such desired sparsity. The same trend roughly extends to validation accuracy (with some caveats due to generalization errors).

Table 11: Ablation on the κ_p hyperparameter for a CIFAR10 task with a global density target of $\epsilon = 50\%$. κ_p **monotonically controls the degree of damping and constraint overshoot.**

νPI κ_p	Train Acc.	Test Acc.	Violation	Relative Violation
0	97.0 $\pm$ 0.10	87.9 $\pm$ 0.18	-17.0 $\pm$ 0.36	-33.9 $\pm$ 0.72
0.008	97.1 $\pm$ 0.06	87.4 $\pm$ 0.23	-16.6 $\pm$ 0.14	-33.2 $\pm$ 0.28
0.08	97.0 $\pm$ 0.09	87.7 $\pm$ 0.29	-16.2 $\pm$ 0.42	-32.4 $\pm$ 0.84
0.8	97.3 $\pm$ 0.07	87.7 $\pm$ 0.19	-13.8 $\pm$ 0.13	-27.5 $\pm$ 0.27
4	97.8 $\pm$ 0.10	88.1 $\pm$ 0.28	-3.3 $\pm$ 0.23	-6.5 $\pm$ 0.46
8	97.9 $\pm$ 0.06	88.2 $\pm$ 0.39	0.4 $\pm$ 0.03	0.9 $\pm$ 0.06
9.6	98.1 $\pm$ 0.07	88.1 $\pm$ 0.18	0.1 $\pm$ 0.02	0.2 $\pm$ 0.04
12	98.0 $\pm$ 0.05	87.7 $\pm$ 0.14	0.1 $\pm$ 0.01	0.2 $\pm$ 0.03
14.4	98.0 $\pm$ 0.08	88.2 $\pm$ 0.17	0.4 $\pm$ 0.02	0.7 $\pm$ 0.05
16	98.1 $\pm$ 0.02	88.3 $\pm$ 0.28	0.7 $\pm$ 0.02	1.5 $\pm$ 0.04
20	98.1 $\pm$ 0.05	87.9 $\pm$ 0.31	2.0 $\pm$ 0.03	4.0 $\pm$ 0.07
24	98.1 $\pm$ 0.05	88.3 $\pm$ 0.15	3.6 $\pm$ 0.03	7.2 $\pm$ 0.06

Table 12: Ablation on the κ_p hyperparameter for a CIFAR10 task with a global density target of $\epsilon = 30\%$.

νPI κ_p	Train Acc.	Test Acc.	Violation	Relative Violation
0	95.9 $\pm$ 0.11	86.4 $\pm$ 0.52	-13.9 $\pm$ 0.11	-46.3 $\pm$ 0.36
0.008	96.0 $\pm$ 0.08	86.4 $\pm$ 0.27	-13.7 $\pm$ 0.13	-45.7 $\pm$ 0.43
0.08	95.9 $\pm$ 0.10	86.9 $\pm$ 0.25	-13.7 $\pm$ 0.18	-45.6 $\pm$ 0.59
0.8	96.1 $\pm$ 0.07	87.2 $\pm$ 0.35	-13.1 $\pm$ 0.13	-43.6 $\pm$ 0.45
4	96.4 $\pm$ 0.17	87.2 $\pm$ 0.62	-8.7 $\pm$ 0.16	-28.9 $\pm$ 0.54
8	96.9 $\pm$ 0.10	87.6 $\pm$ 0.33	-4.0 $\pm$ 0.10	-13.3 $\pm$ 0.32
9.6	97.1 $\pm$ 0.12	87.5 $\pm$ 0.45	-3.0 $\pm$ 0.18	-10.1 $\pm$ 0.60
12	97.3 $\pm$ 0.04	87.5 $\pm$ 0.28	-1.6 $\pm$ 0.11	-5.3 $\pm$ 0.37
14.4	97.4 $\pm$ 0.08	87.4 $\pm$ 0.27	-0.2 $\pm$ 0.11	-0.7 $\pm$ 0.38
16	97.5 $\pm$ 0.11	87.9 $\pm$ 0.32	0.8 $\pm$ 0.17	2.8 $\pm$ 0.57
20	97.6 $\pm$ 0.12	88.1 $\pm$ 0.38	3.3 $\pm$ 0.11	10.9 $\pm$ 0.36
24	97.8 $\pm$ 0.06	87.9 $\pm$ 0.41	5.7 $\pm$ 0.11	19.0 $\pm$ 0.36

H.3. ADAM

We also experimented with a range of learning choices for ADAM to explore their effect on constraint satisfaction and overshoot. The results are shown in Fig. 23, and Tables 13 and 14.

We observe that the influence of ADAM's learning on the constraint overshoot is not monotonic. When the step-size is small, ADAM runs do not satisfy the constraint at the end of training. As the step-size increases, satisfaction is achieved together with varying degrees of overshoot into the feasible region. A range of larger step-sizes that lie at the sweet spot of almost exact constraint satisfaction.

The sensitivity and non-monotonicity of the step-size make the tuning of the step-size hyperparameter for ADAM challenging. Note that we restricted our experiments to the default EMA coefficients for ADAM following PyTorch: $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

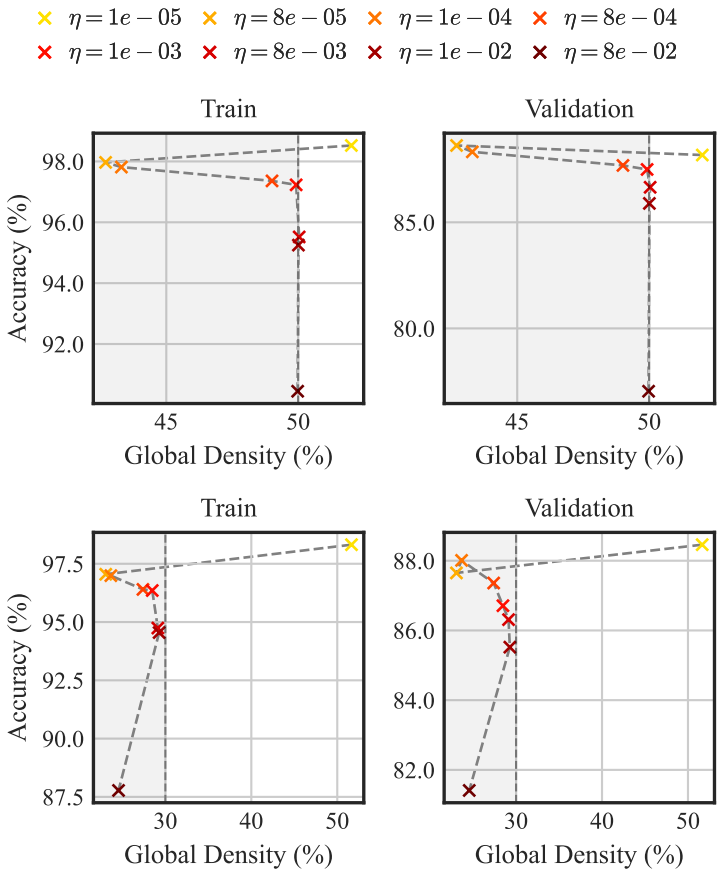


Figure 23: Ablation on the density-accuracy trade-offs achievable by ADAM under global density targets of 50% (top) and 30% (bottom).

Table 13: Ablation on the step-size hyperparameter for ADAM on a CIFAR10 task with a global density target of $\epsilon = 50\%$.

Adam η	Train Acc.	Test Acc.	Violation
$1 \cdot 10^{-5}$	98.52	88.17	2.02
$8 \cdot 10^{-5}$	97.97	88.62	-7.30
$1 \cdot 10^{-4}$	97.81	88.32	-6.70
$8 \cdot 10^{-4}$	97.36	87.68	-0.99
$1 \cdot 10^{-3}$	97.23	87.49	-0.08
$8 \cdot 10^{-3}$	95.52	86.65	0.04
$1 \cdot 10^{-2}$	95.25	85.89	0.01
$8 \cdot 10^{-2}$	90.45	77.04	-0.02

H.4. Momentum

We carried out similar ablations on the momentum coefficient of POLYAK and NESTEROVusing both positive and negative values. The results are shown in Fig. 24, and Tables 15 and 16. We observe significant overshoot into the feasible region for all attempted values, compared to the desired target density of 30%.

Table 14: Ablation on the step-size hyperparameter for ADAM on a CIFAR10 task with a global density target of $\epsilon = 30\%$.

Adam η	Train Acc.	Test Acc.	Violation
$1 \cdot 10^{-5}$	98.32	88.46	21.66
$8 \cdot 10^{-5}$	97.05	87.65	-6.94
$1 \cdot 10^{-4}$	96.99	88.01	-6.34
$8 \cdot 10^{-4}$	96.40	87.36	-2.61
$1 \cdot 10^{-3}$	96.35	86.71	-1.53
$8 \cdot 10^{-3}$	94.74	86.31	-0.88
$1 \cdot 10^{-2}$	94.55	85.52	-0.73
$8 \cdot 10^{-2}$	87.78	81.41	-5.45

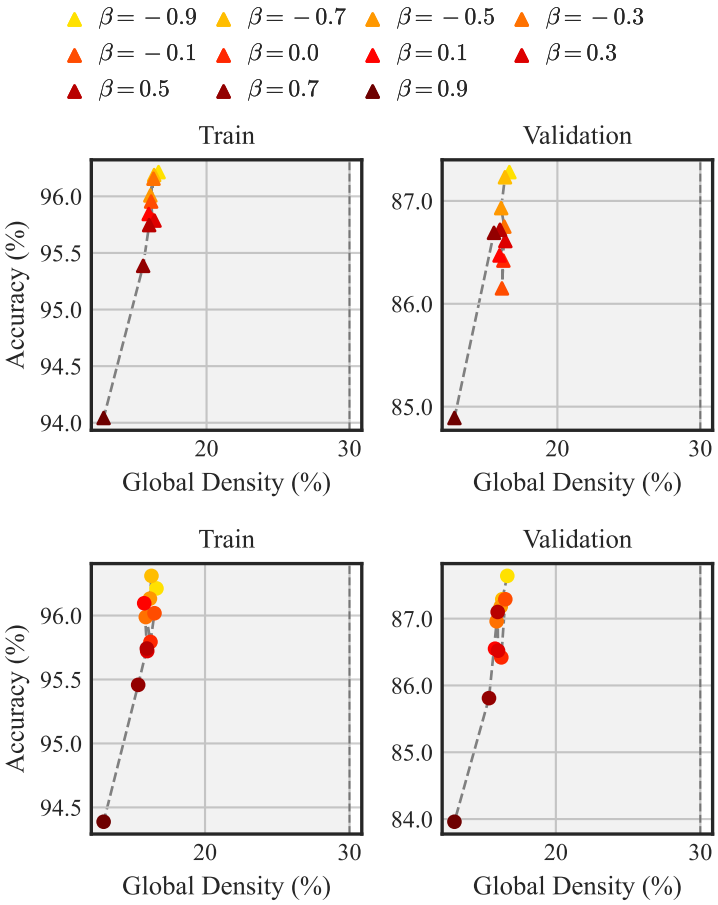


Figure 24: Trade-off plot under a 30% global density target for NESTEROV (top) and POLYAK (bottom) momentum.

Table 15: Ablation on the momentum hyperparameter for NESTEROV on a CIFAR10 task with a 30% global density target.

Nesterov β	Train Acc.	Test Acc.	Violation
-0.9	96.21	87.28	-13.36
-0.7	96.19	87.23	-13.67
-0.5	96.01	86.93	-13.93
-0.3	96.16	86.75	-13.71
-0.1	95.95	86.15	-13.88
0.0	95.79	86.42	-13.78
0.1	95.84	86.47	-14.05
0.3	95.79	86.61	-13.64
0.5	95.75	86.72	-14.02
0.7	95.39	86.69	-14.44
0.9	94.04	84.89	-17.20

Table 16: Ablation on the momentum hyperparameter for POLYAK on a CIFAR10 task with a 30% global density target.

Polyak β	Train Acc.	Test Acc.	Violation
-0.9	96.21	87.64	-13.36
-0.7	96.31	87.29	-13.71
-0.5	96.13	87.18	-13.82
-0.3	95.99	86.96	-14.10
-0.1	96.02	87.29	-13.50
0.0	95.79	86.42	-13.78
0.1	96.10	86.55	-14.21
0.3	95.72	86.52	-14.00
0.5	95.74	87.10	-14.03
0.7	95.46	85.81	-14.63
0.9	94.39	83.96	-17.02