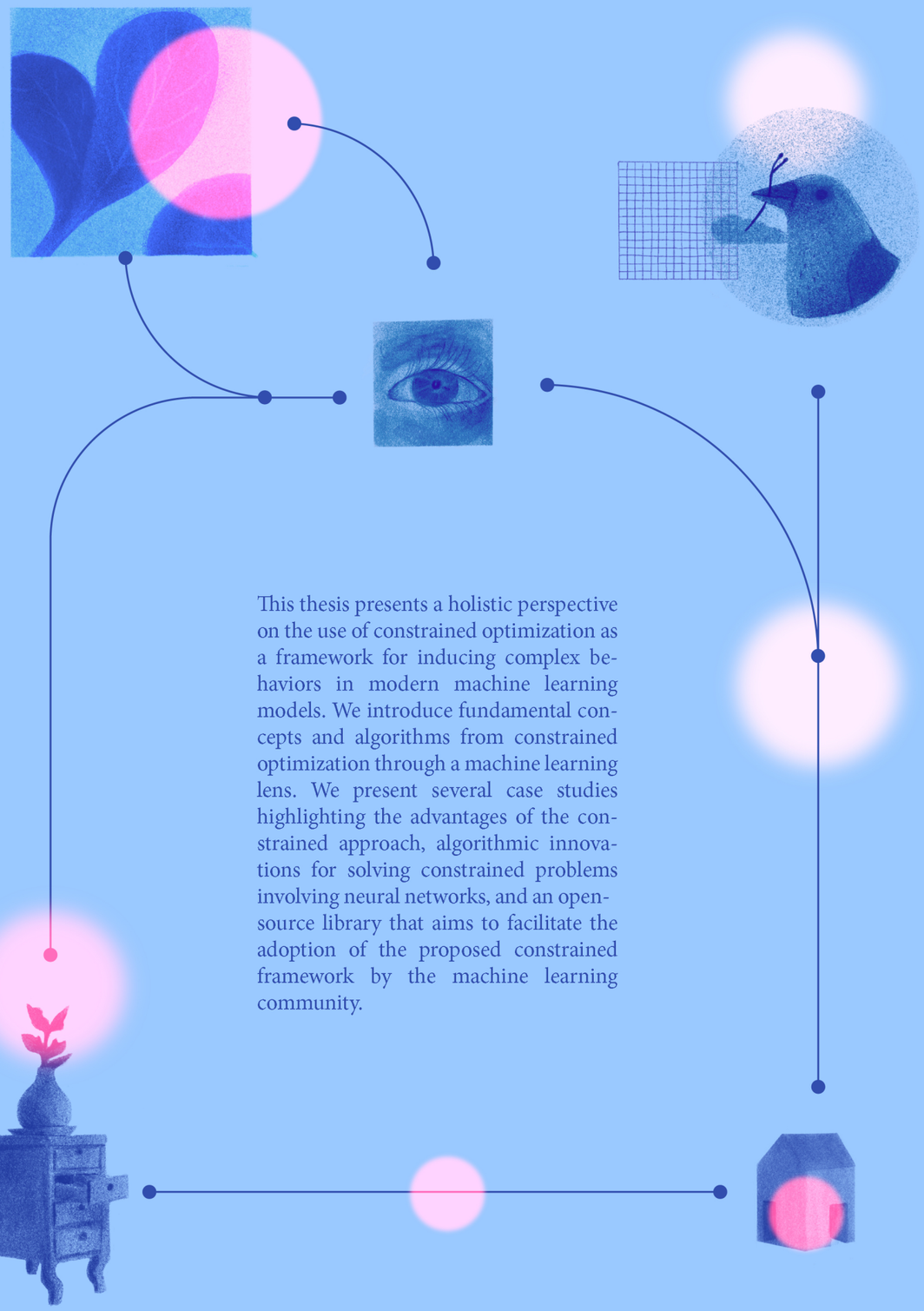


This thesis presents a holistic perspective on the use of constrained optimization as a framework for inducing complex behaviors in modern machine learning models. We introduce fundamental concepts and algorithms from constrained optimization through a machine learning lens. We present several case studies highlighting the advantages of the constrained approach, algorithmic innovations for solving constrained problems involving neural networks, and an open-source library that aims to facilitate the adoption of the proposed constrained framework by the machine learning community.

Constrained Optimization for Machine Learning • *Jose Gallego-Posada*

Jose Gallego-Posada

CONSTRAINED OPTIMIZATION *for* MACHINE LEARNING

Université de Montréal

CONSTRAINED OPTIMIZATION *for* MACHINE LEARNING

Algorithms and Applications

Jose Gallego-Posada

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée en vue de l'obtention du grade de
Philosophiæ Doctor en Informatique

Juin, 2024

Colophon

This document was typeset using LaTeX, based on the `classicthesis` template by André Miede, and stylistically inspired by Aaron Turon's thesis .

Original illustrations by Juan Jose Rodríguez Bianchi and Juan Esteban Tobón Alzate at *Chucha & Guagua*.

Constrained Optimization for Machine Learning: Algorithms and Applications

Université de Montréal

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Cette thèse intitulée

Constrained Optimization for Machine Learning:
Algorithms and Applications

présentée par

Jose Gallego-Posada

a été évaluée par un jury composé des personnes suivantes :

Ioannis Mitliagkas
Président-rapporteur

Simon Lacoste-Julien
Directeur de recherche

Pierre-Luc Bacon
Membre du jury

Jacob Abernethy
Examinateur externe

Résumé

Le déploiement généralisé de modèles d'apprentissage automatique de plus en plus performants a entraîné des pressions croissantes pour améliorer la robustesse, la sécurité et l'équité de ces modèles—souvent en raison de considérations réglementaires et éthiques. En outre, la mise en œuvre de solutions d'intelligence artificielle dans des applications réelles est limitée par leur incapacité actuelle à garantir la conformité aux normes industrielles et aux réglementations gouvernementales. Les pipelines standards pour le développement de modèles d'apprentissage automatique adoptent une mentalité de "construire maintenant, réparer plus tard", intégrant des mesures de sécurité a posteriori. Cette accumulation continue de dette technique entrave le progrès du domaine à long terme.

L'optimisation sous contraintes offre un cadre conceptuel accompagné d'outils algorithmiques permettant d'imposer de manière fiable des propriétés complexes sur des modèles d'apprentissage automatique. Cette thèse appelle à un changement de paradigme dans lequel les contraintes constituent une partie intégrante du processus de développement des modèles, visant à produire des modèles d'apprentissage automatique qui sont intrinsèquement *sécurisés par conception*.

Cette thèse offre une perspective holistique sur l'usage de l'optimisation sous contraintes dans les tâches d'apprentissage profond. Nous examinerons i) la nécessité de formulations contraintes, ii) les avantages offerts par le point de vue de l'optimisation sous contraintes et iii) les défis algorithmiques qui surgissent dans la résolution de ces problèmes. Nous présentons plusieurs études de cas illustrant l'application des techniques d'optimisation sous contraintes à des problèmes courants d'apprentissage automatique.

⁂

Dans la Contribution i, nous plaidons en faveur de l'utilisation des formulations sous contraintes en apprentissage automatique. Nous soutenons qu'il est préférable de gérer des régularisateurs interprétables via des contraintes explicites plutôt que par des pénalités additives, particulièrement lorsqu'il s'agit de modèles non convexes. Nous considérons l'entraînement de modèles creux avec une régularisation L_0 et démontrons que i) il est possible de trouver des solutions réalisables et performantes à des problèmes de grande envergure avec des contraintes non convexes ; et que ii) l'approche contrainte peut éviter les coûteux ajustements par essais et erreurs inhérents aux techniques basées sur les pénalités.

La Contribution ii approfondit la contribution précédente en imposant des contraintes explicites sur le taux de compression atteint par les Représentations Neuronales Implicites—une classe de modèles visant à entreposer efficacement des données (telles qu'une image) dans les paramètres d'un réseau neuronal. Dans ce travail, nous nous concentrons sur l'interaction entre la taille du modèle, sa capacité représentationnelle et le temps d'en-

traînement requis. Plutôt que de restreindre la taille du modèle à un budget fixe (qui se conforme au taux de compression requis), nous entraînons un modèle surparamétré et creux avec des contraintes de taux de compression. Cela nous permet d'exploiter la puissance de modèles plus grands pour obtenir de meilleures reconstructions, plus rapidement, sans avoir à nous engager à leur taux de compression indésirable.

La Contribution iii présente les avantages des formulations sous contraintes dans une application réaliste de la parcimonie des modèles avec des contraintes liées à l'équité non différentiables. Les performances des réseaux neuronaux élagués se dégradent de manière inégale entre les sous-groupes de données, nécessitant ainsi l'utilisation de techniques d'atténuation. Nous proposons une formulation qui impose des contraintes sur les changements de précision du modèle dans chaque sous-groupe, contrairement aux travaux antérieurs qui considèrent des contraintes basées sur des métriques de substitution (telles que la perte du sous-groupe). Nous abordons les défis de la non-différentiabilité et de la stochasticité posés par nos contraintes proposées, et démontrons que notre méthode s'adapte de manière fiable aux problèmes d'optimisation impliquant de grands modèles et des centaines de sous-groupes.

Dans la Contribution iv, nous nous concentrons sur la dynamique de l'optimisation lagrangienne basée sur le gradient, une technique populaire pour résoudre les problèmes sous contraintes non convexes en apprentissage profond. La nature adversariale du jeu min-max lagrangien le rend sujet à des comportements oscillatoires ou instables. En nous basant sur des idées tirées de la littérature sur les régulateur PID, nous proposons un algorithme pour modifier les multiplicateurs de Lagrange qui offre une dynamique d'entraînement robuste et stable. Cette contribution met en place les bases pour que les praticiens adoptent et mettent en œuvre des approches sous contraintes avec confiance dans diverses applications réelles.

Dans la Contribution v, nous fournissons un aperçu de `Cooper` : une bibliothèque pour l'optimisation sous contraintes basée sur le lagrangien dans PyTorch. Cette bibliothèque open-source implémente toutes les contributions principales présentées dans les chapitres précédents et s'intègre harmonieusement dans le cadre PyTorch. Nous avons développé Cooper dans le but de rendre les techniques d'optimisation sous contraintes facilement accessibles aux chercheurs et praticiens de l'apprentissage automatique.

Mots-clés : apprentissage automatique, optimisation, optimisation sous contraintes, optimisation lagrangienne, optimisation min-max, optimisation basée sur gradients, apprentissage profond, réseaux neuronaux, modèles creux, régulateur PID, équité, logiciel libre, PyTorch, `Cooper`.

Abstract

The widespread deployment of increasingly capable machine learning models has resulted in mounting pressures to enhance the robustness, safety and fairness of such models—often arising from regulatory and ethical considerations. Further, the implementation of artificial intelligence solutions in real-world applications is limited by their current inability to guarantee compliance with industry standards and governmental regulations. Current standard pipelines for developing machine learning models embrace a "build now, fix later" mentality, retrofitting safety measures as afterthoughts. This continuous incurrence of technical debt hinders the progress of the field in the long-term.

Constrained optimization offers a conceptual framework accompanied by algorithmic tools for reliably enforcing complex properties on machine learning models. This thesis calls for a paradigm shift in which constraints constitute an integral part of the model development process, aiming to produce machine learning models that are inherently *secure by design*.

This thesis provides a holistic perspective on the use of constrained optimization in deep learning tasks. We shall explore i) the need for constrained formulations, ii) the advantages afforded by the constrained optimization standpoint and iii) the algorithmic challenges arising in the solution of such problems. We present several case-studies illustrating the application of constrained optimization techniques to popular machine learning problems.

In Contribution i, we advocate for the use of constrained formulations in machine learning. We argue that it is preferable to handle interpretable regularizers via explicit constraints, rather than using additive penalties, specially when dealing with non-convex models. We consider the training of sparse models with L_0-regularization and demonstrate that i) it is possible to find feasible, well-performing solutions to large-scale problems with non-convex constraints; and that ii) the constrained approach can avoid the costly trial-and-error tuning inherent to penalty-based techniques.

Contribution ii expands on the previous contribution by imposing explicit constraints on the compression-rate achieved by Implicit Neural Representations—a class of models that aim to efficiently store data (such as an image) within a neural network's parameters. In this work we concentrate on the interplay between the model size, its representational capacity and the required training time. Rather than restricting the model size to a fixed budget (that complies with the required compression rate), we train an overparametrized, sparse model with compression-rate constraints. This allows us to exploit the power of larger models to achieve better reconstructions, faster; without having to commit to their undesirable compression rate.

Contribution iii showcases the advantages of constrained formulations in a realistic model sparsity application with non-differentiable fairness-related constraints. The performance of pruned neural networks degrades unevenly across data sub-groups, thus requiring the use of mitigation techniques. We propose a formulation that imposes constraints on changes in the model accuracy in each sub-group, in contrast to prior work which considers constraints based on surrogate metrics (such as the sub-group loss). We address the non-differentiability and stochasticity challenges posed by our proposed constraints, and demonstrate that our method scales reliably to optimization problems involving large models and hundreds of sub-groups.

In Contribution iv, we focus on the dynamics of gradient-based Lagrangian optimization, a popular technique for solving the non-convex constrained problems arising in deep learning. The adversarial nature of the min-max Lagrangian game makes it prone to oscillatory or unstable behaviors. Based on ideas from the PID control literature, we propose an algorithm for updating the Lagrange multipliers which yields robust, stable training dynamics. This contribution lays the groundwork for practitioners to adopt and implement constrained approaches confidently in diverse real-world applications.

In Contribution v, we provide an overview of `Cooper`: a library for Lagrangian-based constrained optimization in PyTorch. This open-source library implements all the core contributions presented in the preceding chapters and integrates seamlessly with the PyTorch framework. We developed Cooper with the goal of making constrained optimization techniques readily available to machine learning researchers and practitioners.

Keywords: machine learning, optimization, constrained optimization, Lagrangian optimization, min-max optimization, gradient-based optimization, deep learning, neural networks, sparse models, PID control, fairness, open-source software, PyTorch, `Cooper`.

Publications

This dissertation expands on the following publications:

Jose Gallego-Posada, Juan Ramirez, Akram Erraqabi, Yoshua Bengio, and Simon Lacoste-Julien: **Controlled Sparsity via Constrained Optimization or: How I Learned to Stop Tuning Penalties and Love Constraints**. In: *NeurIPS*. 2022. [Chapters 3 and 4].

Juan Ramirez and Jose Gallego-Posada: **L_0onie: Compressing COINs with L_0-Constraints**. In: *Sparsity in Neural Networks Workshop*. 2022. [Chapters 5 and 6].

Meraj Hashemizadeh, Juan Ramirez, Rohan Sukumaran, Golnoosh Farnadi, Simon Lacoste-Julien, and Jose Gallego-Posada: **Balancing Act: Constraining Disparate Impact in Sparse Models**. In: *ICLR*. 2024. [Chapters 7 and 8].

Motahareh Sohrabi, Juan Ramirez, Tianyue H. Zhang, Simon Lacoste-Julien, and Jose Gallego-Posada: **On PI controllers for updating Lagrange multipliers in constrained optimization**. In: *ICML*. 2024. [Chapters 9 and 10].

Jose Gallego-Posada, Juan Ramirez, Meraj Hashemizadeh, and Simon Lacoste-Julien: **Cooper: Constrained Optimization for Deep Learning**. *MLOSS (under submission)* `https://github.com/cooper-org/cooper`. 2024. [Chapters 11 and 12].

Excluded research

In order to keep this thesis topically-focused, the author has decided to exclude several publications and research outputs produced during the course of his doctoral studies:

Jose Gallego-Posada, Ankit Vani, Max Schwarzer, and Simon Lacoste-Julien: **GAIT: A Geometric Approach to Information Theory**. In: *AISTATS*. 2020.

Sourya Basu, Jose Gallego-Posada, Francesco Viganò, James Rowbottom, and Taco Cohen: **Equivariant Mesh Attention Networks**. In: *TMLR*, (2022).

Shin Koseki, Shazade Jameson, et al.: **AI & Cities: Risks, Applications and Governance**. Tech. rep. *Mila-UN Habitat*, 2022.

Hao-Jun M. Shi, Tsung-Hsien Lee, Shintaro Iwasaki, Jose Gallego-Posada, Zhijing Li, Kaushik Rangadurai, Dheevatsa Mudigere, and Michael Rabbat: **A Distributed Data-Parallel PyTorch Implementation of the Distributed Shampoo Optimizer for Training Neural Networks At-Scale**. In: *arXiv:2309.06497*, (2023).

Contents

List of Figures

List of Tables

Acronyms

ALM	Augmented Lagrangian Method
AI	Artificial Intelligence
APD-GDA	Alternating Primal-Dual Gradient Descent-Ascent
ADP-GDA	Alternating Dual-Primal Gradient Descent-Ascent
BPP	Bits-Per-Pixel
CEAG	Constrained Excess Accuracy Gaps
CMP	Constrained Minimization Problem
CNN	Convolutional Neural Network
EAG	Excess Accuracy Gap
GA	Gradient Ascent
GD	Gradient Descent
GMP	Gradual Magnitude Pruning
GPU	Graphics Processing Unit
IID	Independent and Identically Distributed
INR	Implicit Neural Representation
KKT	Karush-Kuhn-Tucker (Conditions)
MLP	Multi-Layer Perceptron
MAC	Multiply-Accumulate (Operation)
NFT	Naive Fine-Tuning
OG	Optimistic Gradient (Method)
OSS	Open-Source Software
PID	Proportional-Integral-Derivative (Controller)
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
S-GDA	Simultaneous Gradient Descent-Ascent

Notation

$\boldsymbol{x} \in \mathbb{R}^n$	An n-dimensional vector
$\boldsymbol{A} \in \mathbb{R}^{m \times n}$	An $m \times n$ matrix
$\boldsymbol{x}_i$	The i-th element of the vector $\boldsymbol{x}$
$\boldsymbol{x} \leq \boldsymbol{y}$	Element-wise inequality between vectors $\boldsymbol{x}$ and $\boldsymbol{y}$
$\boldsymbol{x} \odot \boldsymbol{y}$	The element-wise product of vectors $\boldsymbol{x}$ and $\boldsymbol{y}$
$\|\|\boldsymbol{x}\|\|_p$	The p-norm of a vector $\boldsymbol{x}$
$f : \mathbb{R}^n \to \mathbb{R}$	A scalar-valued function
$\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^p$	A vector-valued function
$\nabla f(\tilde{\boldsymbol{x}})$	The gradient of f at $\tilde{\boldsymbol{x}}$
$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}, \boldsymbol{y})$	The directional derivative of f in the direction of $\boldsymbol{x}$ at $\tilde{\boldsymbol{x}}$
$\mathcal{J}\boldsymbol{f}(\boldsymbol{x})$	The Jacobian matrix of $\boldsymbol{f}$ at $\tilde{\boldsymbol{x}}$
$\nabla_{\boldsymbol{x}\boldsymbol{x}} f(\tilde{\boldsymbol{x}})$	The Hessian matrix of f at $\tilde{\boldsymbol{x}}$
$\dot{\boldsymbol{x}}$	The time derivative of $\boldsymbol{x}$
$\ddot{\boldsymbol{x}}$	The second time derivative of $\boldsymbol{x}$
$\Pi_{\mathcal{X}}(\boldsymbol{x})$	The projection of $\boldsymbol{x}$ onto the set $\mathcal{X}$
$\mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$	The Lagrangian function with multipliers $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$
$\mathbb{E}_{\boldsymbol{x} \sim \mathbb{P}}$	The expectation with respect to the distribution $\mathbb{P}$
$[\boldsymbol{x}]_+$	The positive part of $\boldsymbol{x}$
$\#(\boldsymbol{x})$	The dimension a vector $\boldsymbol{x}$

Ithaka

As you set out for Ithaka
hope the voyage is a long one,
full of adventure, full of discovery.
Laistrygonians and Cyclops,
angry Poseidon—don't be afraid of them:
you'll never find things like that on your way
as long as you keep your thoughts raised high,
as long as a rare excitement
stirs your spirit and your body.
Laistrygonians and Cyclops,
wild Poseidon—you won't encounter them
unless you bring them along inside your soul,
unless your soul sets them up in front of you.

Hope the voyage is a long one.
May there be many a summer morning when,
with what pleasure, what joy,
you come into harbors seen for the first time;
may you stop at Phoenician trading stations
to buy fine things,
mother of pearl and coral, amber and ebony,
sensual perfume of every kind—
as many sensual perfumes as you can;
and may you visit many Egyptian cities
to gather stores of knowledge from their scholars.

Keep Ithaka always in your mind.
Arriving there is what you are destined for.
But do not hurry the journey at all.
Better if it lasts for years,
so you are old by the time you reach the island,
wealthy with all you have gained on the way,
not expecting Ithaka to make you rich.

Ithaka gave you the marvelous journey.
Without her you would not have set out.
She has nothing left to give you now.
And if you find her poor, Ithaka won't have fooled you.
Wise as you will have become, so full of experience,
you will have understood by then what these Ithakas mean.

— C. P. Cavafy

Acknowledgments

While this dissertation presents a compendium of the research that I have conducted during my past few years at Mila, it represents the culmination of a longer academic journey. This thesis is a tribute to the steadfast support and commitment of my grandparents towards my education. Thank you for helping me set sail on the marvelous journey.

I am grateful to Simon Lacoste-Julien for providing me with a once-in-a-lifetime experience of academic freedom in his lab. I shall remember my PhD as a "trial tenure" that allowed me to explore my research interests and develop my academic identity. I greatly respect your solid commitment to slow, responsible and rigorous science.

I have received invaluable guidance from many mentors on life and research, including Christos Louizos, Frans Oliehoek, Gilles Fayad, Golnoosh Farnadi, Guillaume Rabuseau, Ioannis Mitliagkas, Markus Nagel, Mart van Baalen, Mathias Reisser, Michael Shi, Mike Rabbat, Patrick Forre, Rahul Savani, Taco Cohen and Tijmen Blankevoort.

I want to give a special highlight to Ioannis Mitliagkas, whose ever-present smile and down-to-earth approach to the academic life have been a constant source of inspiration. Simon and Ioannis gave me ample opportunity to contribute back to the Mila community as a teaching assistant over many years, eventually leading to a teaching award at the University of Montréal. Thank you for your trust and support.

During my PhD I had the privilege of mentoring many talented students: Aton Kamanda, Daniel Otero, Helen Zhang, Isabel Urrego, Juan Ramirez, Lucas Maes, Meraj Hashemizadeh, Motahareh Sohrabi, and Rohan Sukumaran. Thank you for taking a chance on me. I have grown as a person and as a researcher thanks to you.

To my fellow Mila LabReps: Alekhya Dronavalli, Alex Hernandez-Garcia, Arna Ghosh, Arushi Jain, Dongyan Lin, Kshitij Gupta, Mandana Samiei, Manuela Girotti, Martin Weiss, Mashbayar Tugsbayar, Mélisande Teng, Niki Howe, Oumar Kaba, Raghav Gupta, Rim Assouel, Tegan Maharaj, and Victor Schmidt. I am confident that you will continue leading the field of machine learning with thoughtfulness and integrity.

I was fortunate to receive multiple sources of funding throughout my graduate studies. My research was supported by an Ivado PhD Excellence Scholarship and a Microsoft-Mila Doctoral Research Diversity Award. I completed my master's degree at the University of Amsterdam thanks to financial support from the Colfuturo foundation.

To Andrea Serdio, Aimee Quintana and Cait Mazurek for your friendship since my early days in Montréal. To Eliana Charlebois Gómez for being there in my hour of need. I have missed you all dearly!

To Alberto García, Alexandra Jaramillo, Carlos Cadavid, Carlos Mario Vélez, Clara Suaza, Cristina Moreno, Elva Sanchez, Francisco Correa, Francisco Zuluaga, Fredy Marín,

Guillermo Gil, Helmuth Trefftz, Jairo Villegas, Jorge Iván Castaño, María Eugenia Puerta, Myladis Cogollo, Olga Lucia Quintero, and Paula Escudero. You all have shown me what teaching looks like when it is driven by genuine care and dedication.

I am grateful for having met many other collaborators, co-authors and friends: Abdel Zayed, Akram Erraqabi, Alejandro Posada, Ankit Vani, António Góis, Aristide Baratin, Breandan Considine, Cristina Eickhoff de Oliveira, Damien Scieur, Dana Kianfar, Disha Shrivastava, Émélie Brunet, Emilio Botero, Francesco Viganò, Gabriel Huang, Gauthier Gidel, Hattie Zhou, Ignacio Hounie, James Rowbottom, Jonathan Lebensold, Joshua Iverson, Juan Elenter, Linda Peinthiere, Manfred Diaz, Manuela Girotti, Mansi Rankawat, Mattie Tesfaldet, Marwa El-Halabi, Max Schwarzer, Maxime Wabartha, Mohammad Pezeshki, Nadine Khairallah, Naz Sepahvand, Nicolas Louizou, Pablo Piantanida, Pedram Khorsandi, Pietro Astolfi, Quentin Bertrand, Ramon Emiliani, Remi LePriol, Reyhane Askari, Rozhin Nobahari, Samuel Lavoie, Sébastien Lachapelle, Shalaleh Rismani, Sharan Vaswani, Sourya Basu, and Vitoria Barin Pacela.

To my parents, my sister and the rest of my family: thank you for your patience, your love and your unwavering support. This thesis is as much yours as it is mine.

Julien—thank you for your loving, generous, and intellectually-stimulating company throughout the years. You truly are the wonder that is keeping the stars apart.

Part I

INTRODUCTION & BACKGROUND

Introduction 1

On September 16th, 1987, representatives of 46 countries* signed the *Montreal Protocol on Substances that Deplete the Ozone Layer* [Uni87]. This landmark† international treaty aimed to protect the Earth's ozone layer by phasing out the production and use of chlorofluorocarbons (CFCs), a family of ozone-depleting substances widely used as refrigerants and aerosol propellants.

Prior to the 1920s, refrigeration systems used toxic or flammable gases such as ammonia, sulfur dioxide, or methyl chloride. In 1928, aiming to address these safety concerns, Thomas Midgley Jr., an American chemical engineer at Kinetic Chemicals,‡ developed dichlorodifluoromethane (commonly known as R-12), the first CFC refrigerant.

R-12 not only solved the safety issues of previous refrigerants, being non-toxic and non-flammable—it also enjoyed excellent thermodynamic properties, including a low boiling point and a constant temperature across a wide range of pressures, which made it suitable for various cooling applications [MH30]. R-12's efficiency as a refrigerant contributed to its widespread adoption throughout the 20th century. Midgley§ was awarded the Priestley Medal in 1941 by the American Chemical Society.

As is often the case with influential technologies, the entire spectrum of consequences of CFCs was not fully understood at the time of their invention. By 1925, chemists had identified ozone, established its molecular structure, and recognized its role in the absorption of ultraviolet light in the stratosphere. However, it was not until 1930¶ that a *theory* by Chapman [Cha30] posited the existence of an atmospherical ozone layer.

The following decades witnessed, paradoxically, rapid expansion of the CFCs market alongside intense scientific research on the ozone layer. This included the work of Paul Crutzen, Mario Molina, and Sherwood Rowland during the 1970s, concerning the formation and decomposition of ozone in the stratosphere⊠. Research efforts culminated with the discovery of the Antarctic ozone hole in 1985 by British Antarctic Survey scientists [FGS85], and the subsequent realization that CFCs were responsible for the depletion of the ozone layer, ultimately leading to the signing of the Montreal Protocol.

The Montreal Protocol has resulted in a significant reduction in the production and use of ozone-depleting substances, and has contributed to the recovery of the ozone layer. The treaty's success has been attributed to the cooperation between governments, industry, and civil society in addressing a global environmental challenge.

R-12 has been replaced by R-134a, a hydrofluorocarbon (HFC)** with lower ozone depletion potential and improved cooling efficiency [CSP92]. Our search for more environmentally friendly refrigerants—a regulatory *constraint*—illustrates how responsible innovation can be a catalyst for developing all-around superior technologies.

* *The Montreal Protocol eventually became the first treaty to achieve universal ratification.*

† *Former UN Secretary-General Kofi Annan called it "perhaps the single most successful international agreement" [Uni12].*

‡ *A partnership founded by General Motors and chemical company DuPont.*

§ *Of prior controversy for his role in the development of leaded gasoline.*

¶ *After Midgley's discovery!*

⊠ *For which they were awarded the 1995 Nobel Prize in Chemistry.*

** *HFCs are potent greenhouse gases, and their use is being phased out under the Kigali Amendment to the Montreal Protocol.*

Artificial Intelligence (AI) holds great promise for protecting human rights, promoting social justice, and fostering sustainable economic prosperity. However, achieving these seemingly utopian goals requires concerted international efforts to regulate and incentivize the responsible development and deployment of AI technologies.

AI is already a disruptive innovation in many sectors, including healthcare, finance, transportation, and advertising. AI systems are being used to discover drug candidates, predict financial markets, optimize supply chains, and personalize advertising campaigns. The rapid adoption of AI technologies has been powered by unprecedented progress in computer vision, natural language processing, and reinforcement learning.

While AI is poised to revolutionize many fields, the implementation of AI solutions in real-world applications is limited by their current inability to guarantee compliance with governmental regulations, industry standards, or other safety guidelines. As a result, besides a widespread deployment of increasingly capable machine learning models, the last few years[††] have also been marked by mounting pressures to enhance the robustness, safety, and fairness of such models. These pressures often stem from regulatory [Cou24] or ethical [DAV18] considerations.

[††] *The International Organization for Standardization released the ISO/IEC42001 standard in December 2023, providing guidelines for the responsible development, deployment, and governance of AI technologies.*

Current standard pipelines for developing machine learning models embrace a *"build now, fix later"* mentality, retrofitting safety measures as afterthoughts. This thesis argues that this continuous incurrence of technical debt hinders long-term progress in the field.

A pertinent contemporary example is the release and subsequent rollback of Google's *AI Overviews* in May 2024 [Gra24]. AI Overviews are an enhancement to Google search, integrating AI-generated summaries into search results to provide more comprehensive and context-rich answers to user queries.

AI Overviews quickly faced backlash due to inaccurate answers. Infamously, when asked *"How many rocks should I eat[?]"*, the search engine (incorrectly) responded *"According to UC Berkeley geologists, you should eat at least one small rock per day"*[‡‡]. As a result of this incident, Google issued a statement [Rei24] acknowledging the limitations of AI Overviews and implementing *"more than a dozen technical improvements"* to address said safety and reliability concerns. *Build now, fix later.*

[‡‡] *The source for this answer was later identified to be a 2021 article by satirical publisher The Onion [Oni21].*

This thesis advocates for a paradigm shift in which application-specific *constraints* form an integral part of the model development process, aiming to produce machine learning models that are inherently *secure by design*.

The scientists who developed R-12 could not have foreseen some of its environmental consequences. However, once the issue of ozone depletion was identified, this environmental constraint was successfully integrated into the development process of new generations of refrigerants.

Similarly, while AI still has many *unknown unknowns*[§§], tangible fairness, privacy and safety risks have been identified [ST23]. However, in the absence of readily available techniques for incorporating constraints during model development, improvements on predictive performance are often prioritized over risk mitigation. The current (mis-) prioritization results in the "unsustainable" development of technologies, which, despite excellent performance, may never reach deployment due to regulatory or ethical concerns.

[§§] *Sparking heated debates on existential risk.*

This thesis proposes constrained optimization as a way forward: constrained optimization offers a solid conceptual framework, theoretical guarantees, and efficient algorithmic tools for reliably enforcing complex properties on machine learning models.

At a high level,[¶¶] the constrained approach modifies the training of the model by incorporating the constraints directly into the optimization problem. The optimization procedure then "dynamically re-weights" the constraints in a way that induces the model to satisfy them, while still attempting to be optimal in terms of the training objective.

[¶¶] *These ideas are presented formally in §2.2 and 2.3.*

This dissertation is by no means the first work to consider the use of constraints in machine learning. Support vector machines [CV95], constrained Markov decision processes [Alt99], fairness-aware learning [Dwo+12], and adversarial training [GSS15] are a few notable examples. The value of this thesis, however, lies in its holistic perspective on the use of constrained optimization in modern machine learning. In the following chapters, we shall expand on:

- the theoretical and algorithmic foundations of constrained optimization (§2),
- the advantages of constrained formulations in machine learning tasks, specially when dealing with non-convex problems (§4),
- the algorithmic challenges arising during the solution of constrained optimization problems (§4 and 10),
- empirical demonstrations of the success of constrained optimization in deep learning problems (§4, 6 and 8), and
- the development of open-source software to facilitate the application of constrained optimization techniques in machine learning (§12).

Overview of the contributions

Jose Gallego-Posada, Juan Ramirez, Akram Erraqabi, Yoshua Bengio, and Simon Lacoste-Julien: **Controlled Sparsity via Constrained Optimization or: How I Learned to Stop Tuning Penalties and Love Constraints**. In: *NeurIPS*. 2022. [Chapters 3 and 4]

Penalized methods are widely used for inducing desired properties in machine learning models [GBC16, §5.2.2], with recent works on training generative models [Hig+17], neural network sparsity [LWK18], self-supervised learning [BPL22] and fine-tuning of large language models [Raf+23].

For generalization-enhancing regularization, the penalty coefficient is typically tuned via cross-validation. On the other hand, when the penalties are used to induce other behaviors like sparsity, model alignment or preventing collapse of learned representations, the penalty coefficient is usually tuned via a trial-and-error* process.

An important shortcoming of penalized approaches in non-convex problems is that, in general, they are unable to span the entire set of Pareto-optimal solutions [BV04, §4.7.4].[†] In other words, there exist trade-offs between the objective and penalties that are not reachable by tuning the penalty coefficient. Therefore, protocols for training non-convex models that rely exclusively on penalized approaches are susceptible to ignoring important regions of the Pareto set.

[*] *Adjusting the value on the coefficient depending on whether the penalty was too strong or too weak.*

[†] *Parameter configurations for which neither the objective or the penalty can be improved without worsening the other.*

In this contribution, we take the work of Louizos et al. [LWK18] on training sparse neural networks via L_0 regularization as a case study to demonstrate the advantages that constrained formulations can bring compared to the popular penalty-based techniques. We formulate a constrained optimization problem in which the training loss of the model is minimized, subject to a constraint on the expected[‡] L_0-sparsity of the model.

[‡] *Louizos et al. [LWK18] propose a stochastic reparametrization of the neural network parameters to circumvent the non-differentiability of the L_0-"norm".*

In this work, we demonstrate i) that it is possible to successfully use constrained optimization techniques for achieving well-performing, feasible solutions to sparsity-constrained problems involving deep neural networks; and ii) that the controllability benefits of the constrained formulation enable to drastically mitigate the costly trial-and-error tuning of the penalty coefficient.

Additionally, we identify a challenge in the optimization dynamics of the multipliers. It is often the case that model initializations do not satisfy the constraints and require many parameter updates to reach feasibility. When solving the Lagrangian min-max problem with gradient-based updates, the historic constraint violations are accumulated in the values of the Lagrange multipliers. The long initial period of infeasibility can result in multipliers of large magnitude by the time the model is close to satisfying the constraints, thus pushing the model to *overshoot* towards the interior of the feasible set.

Constraint overshoot is undesirable since it can hinder the model performance. For example, in the case of sparsity constraints, the model performance is typically correlated with the number of active parameters. Converging to a solution that is much more sparse than prescribed by the constraint can lead to a significant drop in performance.

Optimistically, one could hope that "training for long enough" may resolve the constraint overshoot. However, in non-convex problems, the optimization can get stuck in feasible, yet poorly-performing local minima. Furthermore, "training for longer" may require re-engineering the training pipeline given the widespread use of budget-dependent training techniques, such as learning rate schedules.

To address the issue of constraint overshoot, we introduce the *dual restarts* technique: reset the value of a multiplier to zero whenever its corresponding constraint is (strictly) satisfied. While dual restarts are mainly heuristic in nature, they capture the intuition that the historic accumulated violation is not necessarily indicative of the current feasibility of the model. Dual restarts can be motivated in game-theoretic terms as a (partial) best-response [LS22, §2.2] of the dual player. More importantly, our experiments show that dual restarts are highly effective in eliminating overshoot in L_0-sparsity constraints, without leading to gross violation of the constraints.

Finally, as a result of a careful analysis of the learning dynamics, we propose two modifications to the training protocol used for training ResNet models with L_0-sparsity constraints: i) increasing the learning rate of the stochastic gates, and ii) removing the contribution of the weight decay term towards the gates. These two adjustments enable us to train L_0-sparse ResNets *without ruining the model performance*, a feat that had been elusive to prior works [GEH19].

Juan Ramirez and Jose Gallego-Posada: **L_0onie: Compressing COINs with L_0-Constraints**. In: *Sparsity in Neural Networks Workshop*. 2022. [Chapters 5 and 6].

Implicit Neural Representations (INRs) [Sit+20] are a family of data structures that rely on neural networks to represent complex objects, such as images or 3D shapes. For example, rather than storing an image as a grid of pixels, an INR consists of a neural network representing the *mapping* between pixel locations and RGB values.

Dupont et al. [Dup+21] introduced COIN, a data-compression technique based on INRs. Their encoding protocol works as follows: i) train an INR on the image to be compressed, ii) quantize the model parameters, and iii) store the quantized parameters as a code for the image. Reconstructing (part of) the image then amounts to querying the model at the desired pixel locations.

Note that the compression rate[§] of an INR is determined by the number of bits required to store the quantized model parameters. This leads to a natural trade-off between the compression rate and the quality of the reconstructions: the bigger the model, the better the reconstructions, but the worse the compression rate.

[§] *In the case of images, measured in "bits per pixel".*

How could we get the benefits of a large model, without the cost in compression rate? As a natural follow-up to the previous contribution, in L_0onie [¶] we propose to train over-parametrized, *sparse* models with compression-rate constraints. In other words, we take advantage of the inherent redundancies in large models, and sparsify them during training in order to achieve a pre-specified minimum compression rate.

[¶] *A nod to the colloquial name of the Canadian one-dollar coin.*

We highlight that the constraints used in this problem are more complex than those in Contribution i. Rather than constraining the expected L_0-sparsity of the model, we impose constraints on the number of bits required to store the model parameters under a *binarization* of the learned stochastic gates. The binarization ensures a fair assessment of the compression rate, but also introduces non-differentiability in the constraints. We employ the *proxy-constraint*[⊠] technique by Cotter et al. [Cot+19b] to address this issue.

[⊠] *Discussed further in §2.34.*

L_0onie enables the training of sparse INRs achieving better reconstructions, in less wall-clock time, while respecting the desired compression rate, and dispensing of the need for expensive architecture search.

Meraj Hashemizadeh, Juan Ramirez, Rohan Sukumaran, Golnoosh Farnadi, Simon Lacoste-Julien, and Jose Gallego-Posada: **Balancing Act: Constraining Disparate Impact in Sparse Models**. In: *ICLR*. 2024. [Chapters 7 and 8].

Pruning is a widely used approach to reduce the size of neural networks, making them more efficient to deploy on resource-constrained devices. Popular pruning techniques [ZG17] can achieve marginal loss of accuracy over the entire dataset at aggressive sparsity levels [Bla+20]. However, the degradation** in performance can vary significantly across data sub-groups [Hoo+19; Hoo+20; Pag20]. This fairness issue is known as the *disparate impact of pruning* [Tra+22].

** *Compared to the dense version of the model.*

Existing mitigation methods fall short in terms of their interpretability and their ability to scale to large numbers of sub-groups. For example, Tran et al. [Tra+22] constrain the model's loss to be equal across sub-groups. However, this approach does not account for the possibility of uneven performance in the original dense model. Furthermore, their proposed constraints capture issue of disparate impact *indirectly*, as the loss is merely

a surrogate for the model's accuracy. In contrast, the fairness-aware pruning approach of [LKJ22] decides which weights are removed based on importance scores computed for every parameter and every sub-group. Clearly, as the model size or the number of sub-groups increase, the computational cost of this technique becomes prohibitive.

In this contribution, we present a novel technique for mitigating pruning-induced disparity. We propose a constrained formulation of the problem by imposing (non-differentiable!) constraints that require the variation in the accuracy degradation across sub-groups to be low, without resorting to the use of surrogate metrics. Moreover, the proposed approach has a negligible computational overhead compared to naively fine-tuning the sparse model.

The use of the *proxy-constraint* technique [Cot+19b] to address the non-differentiability of the constraints[††] was an important source of progress compared to prior works. Undoubtedly, the *change in accuracy* on a given sub-group before and after pruning is a central metric for determining whether said sub-group was disproportionately affected by the pruning process. However, the differentiability challenges posed by the accuracy may have deterred prior works from considering accuracy-based constraints explicitly. In contrast, our problem formulation and proposed algorithmic approach directly aim to limit the variations on the accuracy degradation across sub-groups.

[††] *As in L*$_0$*onie.*

We emphasize that the use of accuracy-based constraints enhances the interpretability of the approach. For example, regulations or company policies may prescribe a maximum level of allowable disparity for a given application. Incorporating this problem specification is substantially simpler[‡‡] when the constraints are directly related to the model's disparity, rather than being based on a surrogate metric like the loss.

[‡‡] *We demonstrate this claim in our experiments by changing the maximum disparity level across datasets.*

Another key challenge in this work is the need to perform stochastic estimates of the constraint violations. Unlike the two initial contributions, in which the constraints could be evaluated in closed form, the constraints in this work involve computing expectations of the model's accuracy over sub-groups of data. Given the scale of the models and datasets we consider, the cost of computing the exact constraint violations would be prohibitive, thus leading us to resort to mini-batch estimates of the constraints. However, the inexact measurement of the constraints introduces noise in the update of the multipliers and model parameters. Inspired by the notion of *replay buffers* [Mni+13] used in reinforcement learning, we introduce a variance reduction technique that leverages model predictions on past mini-batches to estimate the constraints more reliably.

We empirically demonstrate the effectiveness of our approach for mitigating pruning-induced disparity across a variety of datasets, model architectures and sparsity levels. Notably, our technique scales seamlessly to tasks with hundreds of protected sub-groups.

Finally, while only our approach delivered reliable disparity mitigation on training data, our experimental results indicate that *all the methods we evaluated in this paper*[§§] *fail to mitigate pruning-induced disparities on unseen data*. Interestingly, our work was the first to document this generalization challenge. At the writing of this dissertation, achieving well-generalizing disparity mitigation remains an open problem in the field of fairness-aware pruning. We hope our empirical observations will motivate further research on this topic.

[§§] *Including ours!*

Motahareh Sohrabi, Juan Ramirez, Tianyue H. Zhang, Simon Lacoste-Julien, and Jose Gallego-Posada: **On PI controllers for updating Lagrange multipliers in constrained optimization**. In: *ICML*. 2024. [Chapters 9 and 10].

Having established the benefits of constrained approaches in several applications, we turn our attention to an important aspect of constrained optimization: the dynamics of the optimization process itself.

As we anticipated in our discussion of Contribution i, the use of gradient-based updates on the Lagrangian min-max problem can cause an excessive accumulation of the constraint violations in the Lagrange multipliers, leading to constraint overshoot. Other shortcomings of the gradient descent-ascent (GDA) scheme include instability, oscillations or slow convergence [PB87; Gid+19a; SAA20; Gal+22]. Alleviating the shortcomings of GDA on Lagrangian problems is an important step towards wider adoption of constrained optimization in deep learning.

While dual restarts [Gal+22] were successful in the L_0-sparsity task, they do not constitute a full solution to the deficient multiplier dynamics of GDA. For instance, dual restarts are not suitable for equality[¶¶] or stochastically-estimated constraints[***].

[¶¶] *Or inequality constraints satisfied with equality.*

[***] *Since estimation noise could mistakenly flag an infeasible costraint as being strictly satisfied.*

We identified that the central issue of GDA lay in the lack of adaptivity of the multiplier updates. In other words, GDA implements "memoryless" updates and, as a result, it discards historic information on the progress of the constraint violation. Intuitively, the Lagrange multipliers should react to *changes in the satisfaction of the constraint*, rather than just the immediately observed violation.

Thus, we decided to analyze whether generic adaptive or momentum-based methods for single-objective optimization (such as Polyak [Pol64], Nesterov [Nes83] or Adam [KB15]) would be sufficient to address these issues. Our exploration led to a negative answer in this respect, prompting us to look for alternative frameworks.

In this work, we study the dynamics of the Lagrange multipliers from a control-theoretic perspective. Following Stooke et al. [SAA20], we can think of the update rule of the multipliers as a *control algorithm* that aims to drive the model towards feasibility.

Proportional-Integral-Derivative (PID) controllers [ÅH95] are a popular class of control algorithms used in many applications, from temperature regulation in industrial processes to flight control systems in aerospace engineering. PID controllers regulate the value of the *control inputs* to an underlying dynamical system based on an *error signal* that quantifies the discrepancy between the current and desired states. As their name indicates, PID controllers use the value of the error signal, its integral, and its derivative to determine the control action.

In constrained optimization, the error signal is given by the violation of the constraints, while the control inputs to the system correspond to the Lagrange multipliers. The design principles behind PID controllers align precisely with our observation that the Lagrange multipliers should act based on both the value of the current violation, and how this compares to previous violations[†††].

[†††] *In dynamical systems lingo, the time derivative of the violations.*

We introduce a new update scheme for updating the Lagrange multipliers called $\boldsymbol{\nu}$PI, which constitutes a variant of a Proportional-Integral (PI) controller enriched with an

exponential moving average on the past constraint violations. Inspired by the works of Lessard et al. [LRP16] and Recht [Rec18], we also (constructively) prove that νPI constitutes a strict generalization of the Polyak and Nesterov methods.

We provide extensive insights on how νPI improves the dynamics of the Lagrange multipliers, and empirically demonstrate its effectiveness at stabilizing the training process in a variety of constrained optimization problems.

Jose Gallego-Posada, Juan Ramirez, Meraj Hashemizadeh, and Simon Lacoste-Julien: **Cooper: Constrained Optimization for Deep Learning**. *MLOSS (under submission)* `https://github.com/cooper-org/cooper`. 2024. [Chapters 11 and 12].

In the final contribution, we provide a brief overview of the `Cooper` library for solving constrained optimization problems involving deep learning models. We developed `Cooper` with the goal of making constrained optimization techniques readily available to machine learning researchers and practitioners.

`Cooper` implements (and unit-tests!) several first-order update schemes for Lagrangian-based constrained optimization, along with specialized features for tackling problems with large numbers of (possibly non-differentiable) constraints. We hope that `Cooper` can serve as a unifying[‡‡‡] framework to enhance the reproducibility and ease of comparison between research projects on constrained optimization for machine learning.

[‡‡‡] *Much like how PyTorch provides the community with standardized and tested implementations of neural network layers and optimizers.*

While `Cooper` can be used as a general-purpose library for non-convex constrained optimization,[§§§] it has a strong emphasis on deep learning. In particular, `Cooper` has been designed to provide native support for the framework of stochastic first-order optimization using mini-batch estimates that is prevalent in the training of deep learning models. `Cooper` relies on the PyTorch framework [Pas+19] for efficient tensor computation and automatic differentiation.

[§§§] *And we welcome users and contributors from fields adjacent to artificial intelligence!*

`Cooper` is an open-source project, available under an MIT license, and can be found at `https://github.com/cooper-org/cooper/`.

Excluded Research

In order to keep this thesis topically-focused, the author has decided to exclude several publications and research outputs produced during the course of his doctoral studies:

Jose Gallego-Posada, Ankit Vani, Max Schwarzer, and Simon Lacoste-Julien: **GAIT: A Geometric Approach to Information Theory**. In: *AISTATS*. 2020.

Sourya Basu, Jose Gallego-Posada, Francesco Viganò, James Rowbottom, and Taco Cohen: **Equivariant Mesh Attention Networks**. In: *TMLR*, (2022).

Shin Koseki, Shazade Jameson, et al.: **AI & Cities: Risks, Applications and Governance**. Tech. rep. *Mila-UN Habitat*, 2022.

Hao-Jun M. Shi, Tsung-Hsien Lee, Shintaro Iwasaki, Jose Gallego-Posada, Zhijing Li, Kaushik Rangadurai, Dheevatsa Mudigere, and Michael Rabbat: **A Distributed Data-Parallel PyTorch Implementation of the Distributed Shampoo Optimizer for Training Neural Networks At-Scale**. In: *arXiv:2309.06497*, (2023).

Background 2

In this chapter we provide a brief overview of the main concepts and techniques that are relevant to the work presented in this thesis. We start by introducing fundamental concepts from machine learning and popular techniques for deep learning optimization. We then present core ideas of constraint optimization, the theoretical foundations of the field, and the main algorithms used to solve constrained optimization problems.

2.1 Machine Learning

Machine learning is a scientific discipline that studies higher-level algorithms for automating the programming of lower-level algorithms, referred to as *models*.

The field of machine learning encompasses various *learning paradigms*, each tailored to distinct types of problems and data structures. *Supervised learning* involves training a model on a labeled dataset, where the desired output is known. Typical supervised learning tasks include classification, regression, ranking, object detection and image segmentation. In contrast, *self-supervised learning* leverages the data to generate labels, allowing models to learn representations and features without external labeling, commonly used in natural language processing and computer vision.* *Reinforcement learning*, on the other hand, trains agents through interactions with an environment, optimizing actions based on cumulative rewards, and is particularly suited for dynamic decision-making tasks such as game playing and robotic control.

* *Ultimately, self-supervised learning solves a (sequence of) supervised learning problems by procuring labels from the data itself.*

In this thesis we mostly focus on supervised learning problems. We refer the interested reader to Balestriero et al. [Bal+23] and Sutton and Barto [SB18] for overviews of self-supervised and reinforcement learning, respectively.

2.1.1 *Supervised learning*

In supervised learning, we are given a *training dataset* $\mathfrak{D} = \{(\boldsymbol{i}_n, \boldsymbol{o}_n)\}_{n=1}^{N}$ of labelled pairs, where $\boldsymbol{i}_n \in \mathcal{I}$ is an input vector and $\boldsymbol{o}_n \in \mathcal{O}$ is the corresponding output. The goal is to learn a function $h : \mathcal{I} \to \mathcal{O}$ that successfully approximates the relationship between inputs and outputs in the dataset. We call h the *hypothesis*, *model* or *predictor*.

Suppose the inputs $\boldsymbol{i}_n$ correspond to a 9x9 matrix representation of a Sudoku puzzle. Let us examine multiple instantiations of supervised learning tasks depending on the nature of the labels:

- **Classification**: $\boldsymbol{o}_n \in \{\texttt{True}, \texttt{False}\}$ a label determining whether the puzzle has been solved correctly.

- **Regression**: $\boldsymbol{o}_n \in \mathbb{R}_+$ indicates the average time humans took to solve the puzzle.
- **Structured prediction; multi-label classification**: $\boldsymbol{o}_n \in \mathcal{I}$ is a 9x9 matrix representing a solution to the puzzle.
- **Sequence prediction**: $\boldsymbol{o}_n$ is a sequence of cell-value pairs representing the steps needed to solve the puzzle.

Given a predictor, we can evaluate its performance by comparing its predictions to the true labels in the training data. We quantify the discrepancy between the predicted and true outputs via a *loss function* $\ell : \mathcal{O} \times \mathcal{O} \to \mathbb{R}$, which measures the error between the predicted and true outputs.[†]

[†] *The choice of a suitable loss function is problem-dependent and crucially influences the performance of the learned predictor.*

We typically do not have access to the entire distribution of the data $\mathbb{P}_{\text{data}}$,[‡] but only to the finite sample $\mathfrak{D}$. This prevents us from measuring (and thus optimizing!) the *true risk* of the predictor:

[‡] *For instance, there are* $\sim 10^{21}$ *valid Sudokus.*

$$R(h) = \mathbb{E}_{(\boldsymbol{i},\boldsymbol{o})\sim\mathbb{P}_{\text{data}}}[\ell(h(\boldsymbol{i}), \boldsymbol{o})]. \tag{2.1}$$

The *Empirical Risk Minimization* (ERM) principle [Vap91] proposes to replace the true risk functional in Eq. (2.1) by the *empirical risk* over the sample $\mathfrak{D}$ and treat this as an optimization objective:

$$\hat{R}_{\mathfrak{D}}(h) = \frac{1}{N}\sum_{n=1}^{N} \ell(h(\boldsymbol{i}_n), \boldsymbol{o}_n). \tag{2.2}$$

The goal of *learning* is to find a model that *generalizes* to unseen samples from the data distribution, i.e., a model with low true risk. In contrast, the goal of *training* is to find a model that minimizes the empirical risk over the available dataset.

This discrepancy between the learning and training goals is known as the *generalization gap*. A model that performs well on the training data but poorly on unseen data is said to *overfit* the training data. Many *regularization* techniques exist to mitigate overfitting, such as the use of (cross-)validation, early stopping, weight decay, and dropout. See Goodfellow et al. [GBC16, §5.2-5.3] for an overview of these techniques.

A notable challenge of the ERM protocol presented thus far is that the loss function ℓ may not be differentiable, making the minimization of the empirical risk intractable. For instance, we may wish to minimize the miss-classification rate, corresponding to the so-called *0-1 loss*. This loss function is non-differentiable and discontinuous. A common approach is to use a *surrogate loss function* $\tilde{\ell}$ that approximates the true loss function ℓ and has desirable optimization properties such as differentiability and convexity. Popular surrogate loss functions include the *cross-entropy loss* for classification problems and the *mean squared error* for regression problems.

2.1.2 *Regularization via additive penalties*

Better generalization is only one of the many different goals that can be pursued when applying regularization techniques. Other desired behaviors may include sparsity,[§] robustness, fairness, interpretability or numerical stability.

[§] *Be it for feature selection or compute efficiency*

A popular approach for enforcing these behaviors is to augment the empirical risk with an additive *regularization term* $\mathfrak{R}$, weighted by a *penalty parameter* λ:

$$\hat{R}_{\mathcal{D}}(h) + \lambda \mathfrak{R}(h). \tag{2.3}$$

This penalized approach is widespread in many deep learning applications including generative models [Hig+17], sparse models [LWK18], self-supervised learning [BPL22] and fine-tuning of large language models [Raf+23]. Often, the employed regularizers are motivated from statistical (e.g. a Bayesian prior) or optimization (e.g. improving convexity properties) considerations.

The penalty parameter λ controls the trade-off between the training objective and the regularization term, and typically must be tuned via cross-validation or other hyper-parameter optimization techniques.

2.1.3 *Parametric models and neural network architectures*

We concentrate on *parametric* models, where the predictor h is defined by a vector of parameters $\boldsymbol{x} \in \mathbb{R}^p$. In practice, training is thus "reduced" to the solution of an unconstrained, finite-dimensional[¶] minimization problem:

[¶] *Albeit often very large!*

$$\min_{\boldsymbol{x} \in \mathbb{R}^p} f(\boldsymbol{x}) \triangleq \frac{1}{N} \sum_{n=1}^{N} \tilde{\ell}(h_{\boldsymbol{x}}(\boldsymbol{i}_n), \boldsymbol{o}_n). \tag{2.4}$$

Deep learning is a prominent subfield of machine learning that favors *neural networks* as the parametric models of choice [GBC16]. A central tenet of deep learning is the use of large[▨] *deep* neural networks, which are compositions of functional modules called *layers* and non-linear activation functions.[**]

[▨] *I.e., with many parameters.*

[**] *The parameters of the network are often called weights and biases.*

Formally, a neural network is a function $h_{\boldsymbol{x}} : \mathcal{I} \to \mathcal{O}$ defined by the composition of a sequence of layers $\psi_1, \ldots, \psi_L$:

$$h_{\boldsymbol{x}} = \psi_L \circ \ldots \circ \psi_1. \tag{2.5}$$

There exist many options for the layers ψ_l, including *activation*, *fully-connected*, *convolutional*, *recurrent*, *batch normalization* and *attention* layers. Each of these layers introduces different *inductive biases*[††] that determine its performance on different learning tasks. A specific arrangement of layers is known as a *neural network architecture*. For further details on neural network architectures, see Goodfellow et al. [GBC16], Bishop and Bishop [BB23], and Prince [Pri23].

[††] *Modelling assumptions made about the data.*

2.1.4 *Neural network training*

During the past decade, software advances in automatic differentiation[‡‡] [Pas+19; Aba+15; Dee+20] and the development of specialized hardware for efficient matrix multiplications (such as GPUs, TPUs) have enabled the training of large deep networks on large-scale datasets.

[‡‡] *Removing the need for manually-derived gradients, while preserving numerical stability.*

Note that in the context of a large dataset, exactly evaluating the surrogate empirical risk in Eq. (2.4) can be computationally expensive. To mitigate this issue, we typically use *stochastic optimization* algorithms that approximate the gradient of the empirical risk using a small subset of the data, known as a *mini-batch*.

Furthermore, the large number of parameters in the network, coupled with the non-convexity of the problem have historically precluded the widespread use of second-order optimization methods such as Newton's method or the L-BFGS algorithm [LN89].

This socio-technical context motivated the surge in popularity of *stochastic gradient descent* (SGD) [RM51] and its variants, which are first-order, iterative optimization algorithms that are well-suited for the optimization of large-scale, non-convex problems.

A mini-batch SGD update on the parameters $\boldsymbol{x}$ of the network for Eq. (2.4) entails:

$$\begin{cases} \text{Sample a mini-batch } \{(\boldsymbol{i}_n, \boldsymbol{o}_n)\}_{n=1}^{B} \text{from } \mathfrak{D} \\ \text{Compute an unbiased gradient estimate: } \boldsymbol{g}_t \triangleq \frac{1}{B} \sum_{n=1}^{B} \nabla_{\boldsymbol{x}_t} \tilde{\ell}(h_{\boldsymbol{x}}(\boldsymbol{i}_n), \boldsymbol{o}_n) \\ \text{Update the parameters: } \boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \eta_{\text{primal}}\, \boldsymbol{g}_t, \end{cases} \tag{2.6}$$

where η_{primal} is a *hyper-parameter* of the optimization procedure known as the *learning rate* or *step-size*.

For convergence analyses of SGD in the convex and stochastic settings from a machine learning perspective, see Bubeck et al. [Bub+15].

Several variants of SGD have been proposed to improve convergence and robustness on the non-convex deep learning optimization landscapes. Well studied options are the *accelerated* and *adaptive* families of methods. Notable examples include Polyak [Pol64] and Nesterov [Nes83] momentum methods, as well as Adagrad [DHS11] and Adam [KB15].

2.2 Theory of Constrained Optimization

Many practical problems come with inherent constraints, such as budget limits, resource capacities, physical laws, or specific operational requirements. Constrained optimization allows for the incorporation of real-world limitations and requirements directly into the optimization process. By considering these constraints explicitly, constrained optimization ensures that the solutions are not only optimal in terms of the objective function but also feasible and applicable in real scenarios.

For instance, latency limitations might require a model to make predictions within a certain time frame. Since the latency of the model is controlled by the amount of computation required to make a prediction, which in turns depends on the number of active parameters in the model. Therefore, we can encode the latency requirements by constraining the number of active parameters in the model.

In this section, we introduce the basic language, concepts and mathematical results of non-linear constrained optimization. In Section 2.3 we present several algorithmic approaches for solving constrained optimization problems.

Throughout this section we assume that the objective and constraints are continuously differentiable functions from $\mathbb{R}^p$ to $\mathbb{R}$.

2.2.1 *Feasibility*

Problem-dependent requirements can be abstractly specified by restricting the optimization problem to a subset of the parameter space $\boldsymbol{\Omega} \subset \mathbb{R}^p$. $\boldsymbol{\Omega}$ is known as the *feasible set* and its elements are called *feasible vectors*.

Whenever $\boldsymbol{\Omega}$ is empty, we say that the problem is *infeasible*. Infeasibility may arise from different sources: the problem may be ill-posed, the constraints may be contradictory, or the feasible set may be too restrictive.

Unfortunately, establishing whether a problem is feasible may be as hard as solving the problem itself. Analyzing the semantics of the constraints at hand is often a productive approach to understanding the feasibility of the problem. For example, in the neural network sparsity problem considered in Chapter 4, the constraints demand that at most a certain number of parameters of the network are active. Given the parametrization chosen for that problem, it is evident that feasible solutions exist for any sparsity level.

However, in other cases, the constraints may be more complex and establishing feasibility is not possible from a simple inspection of the constraints. For instance, the disparate impact constraints presented in Chapter 8 request that the performance of the model be sufficiently high across different data sub-groups, within a prescribed tolerance. Here, the constraints depend not only on the model parameters, but also on the available training data. Setting appropriate tolerances to ensure feasibility of the problem is a design decision that needs to be informed by an understanding of the application's context. Regulations or company policies may dictate what level of disparate impact is acceptable for a given application.

2.2.2 *Feasibility and accountability*

We emphasize the fact that formulating the optimization problem requires an understanding of the application domain. For instance, the maximum allowable latency depends on how time-critical the predictions are.

Consider two vectors $\boldsymbol{x} \in \boldsymbol{\Omega}$ and $\boldsymbol{x}' \notin \boldsymbol{\Omega}$, such that $\boldsymbol{x}'$ achieves a better objective value than $\boldsymbol{x}$. Although in the unconstrained setting $\boldsymbol{x}'$ would be preferred, its infeasibility rules it out as a valid solution for the constrained problem. For example, returning to the latency-constrained problem, $\boldsymbol{x}'$ might correspond to a solution with better predictive performance, but whose latency makes it unsuitable for deployment.

While, strictly speaking, feasible vectors are the only valid solutions to the optimization problem, real applications often involve uncertainty on the admissible trade-offs: do users prefer a slightly slower model with better performance, or a faster model with slightly worse performance? Some results in constrained optimization, such as the sensitivity interpretation of the multipliers (Section 2.2.8), can help inform these decisions.

More importantly, we argue that constrained formulations provide enhanced accountability in the model development process. Consider a setting in which the constraints are prescribed by a regulatory body, such as the maximum allowable disparity across protected demographics in a credit scoring application. In the constrained approach, application requirements are embedded directly into the optimization pipeline. Therefore, rather than aiming for models that *happened* to satisfy the regulations, we look for models that are *designed* to respect them.

The accountability and transparency afforded by the constrained optimization framework are crucial in an era of (justified) increased scrutiny on the development of machine learning models for socially-impactful applications. Constrained optimization enables service providers to demonstrate regulatory compliance to stakeholders and enforcement agencies.

2.2.3 *Problem formulation*

In this section we concentrate on the *constrained* minimization problem

$$\text{find } \boldsymbol{x}^* \in \operatorname*{argmin}_{\boldsymbol{x}\in\boldsymbol{\Omega}} f(\boldsymbol{x}). \tag{CMP}$$

A point $\boldsymbol{x}^*$ is an *(constrained) global minimum of* f if $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in \boldsymbol{\Omega}$.

The *optimal value* of problem (CMP) is defined as the objective value attained at any constrained global minimum: $f^* \triangleq \inf_{\boldsymbol{x}\in\boldsymbol{\Omega}} f(\boldsymbol{x})$.

Often, achieving a global minimum is too ambitious and we must settle for a weaker notion of optimality. We a point $\boldsymbol{x}^*$ a *(constrained) local minimum of* f if $\boldsymbol{x}^*$ is feasible and there exists a neighborhood $\mathcal{N}$ of $\boldsymbol{x}^*$ such that $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathcal{N} \cap \boldsymbol{\Omega}$.

The feasible set $\boldsymbol{\Omega}$ is typically defined by a collection of *constraints* that must be respected for a candidate solution to be considered valid. Constraints can be classified into two categories: *equality constraints* and *inequality constraints*.

$$\boldsymbol{\Omega} = \left\{ \boldsymbol{x} \in \mathbb{R}^p \;\middle|\; \begin{array}{ll} g_i(\boldsymbol{x}) \leq 0, & i = 1, \dots, m \\ h_j(\boldsymbol{x}) = 0, & j = 1, \dots, n \end{array} \right\} \tag{2.7}$$

Defining $\boldsymbol{g}(\boldsymbol{x}) \triangleq (g_1(\boldsymbol{x}), \dots, g_m(\boldsymbol{x}))$ and $\boldsymbol{h}(\boldsymbol{x}) \triangleq (h_1(\boldsymbol{x}), \dots, h_n(\boldsymbol{x}))$, we can re-express the problem (CMP) as:

$$\min_{\boldsymbol{x}\in\mathbb{R}^p} f(\boldsymbol{x}) \;\text{ subject to }\; \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0} \;\text{ and }\; \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}. \tag{CMP}$$

Whenever $g_i(\boldsymbol{x}) > 0$, we say that the i-th inequality constraint is *violated* at $\boldsymbol{x}$. Similarly, if $h_j(\boldsymbol{x}) \neq 0$, we say the j-th equality constraint is *violated*. Otherwise, the constraint is said to be *satisfied*.

For a feasible point $\boldsymbol{x}$, the set of *active (inequality) constraints** is defined as

$$\mathcal{A}_\text{I}(\boldsymbol{x}) \triangleq \{ i \mid g_i(\boldsymbol{x}) = 0 \}. \tag{2.8}$$

* *The feasibility of* $\boldsymbol{x}$ *forces the equality constraints to be trivially active.*

The set of all active constraints at a feasible point $\boldsymbol{x}$ is defined as[†]

$$\mathcal{A}(\boldsymbol{x}) = \{j \in 1, \dots n\} \uplus \mathcal{A}_{\mathrm{I}}(\boldsymbol{x}). \tag{2.9}$$

[†] $\uplus$ *denotes the disjoint union of sets.*

If $\boldsymbol{x}$ is feasible and $i \notin \mathcal{A}_{\mathrm{I}}(\boldsymbol{x})$, the constraint $g_i(\boldsymbol{x}) < 0$ is said to be *inactive* or *strictly satisfied* at $\boldsymbol{x}$.

The set of *linearized feasible directions* at a feasible $\boldsymbol{x}$ indicates the set of perturbations that can be applied to $\boldsymbol{x}$ while locally preserving feasibility and is defined by

$$\mathcal{F}(\boldsymbol{x}) = \left\{ \boldsymbol{d} \in \mathbb{R}^p \;\middle|\; \begin{array}{ll} \boldsymbol{d}^\top \nabla g_i(\boldsymbol{x}) \leq 0, & i \in \mathcal{A}_{\mathrm{I}}(\boldsymbol{x}) \\ \boldsymbol{d}^\top \nabla h_j(\boldsymbol{x}) = 0, & j = 1, \dots, n \end{array} \right\}. \tag{2.10}$$

Note that if $\boldsymbol{x}^*$ is a local optimum of problem (CMP), then $\boldsymbol{x}^*$ is also a local optimum of a version of (CMP) where the inactive constraints at $\boldsymbol{x}^*$ are removed and the active inequality constraints are replaced by equalities. As Bertsekas [Ber16, p. 377] argues, at a local minimum "inactive constraints at $\boldsymbol{x}^*$ don't matter" and "active inequality constraints can be treated to a large extent as equalities".

Remark: These claims do not purport an equivalence between the problems with and without inactive constraints. First, removing a constraint may enlarge the feasible set and allow for new, better, local minima. Second, recall the discussion on the differences between optimization and learning presented in Section 2.1.1. The dynamics observed during training may be highly dependent on whether an inactive constraint is removed or not. During the course of optimization, the presence or absence of the constraint may lead to convergence to a *different* local minimum, which may have different objective value and generalization properties.

2.2.4 *Lagrangian duality*

For simplicity, throughout this subsection we assume problem (CMP) is feasible and has bounded optimal value.

The *Lagrangian function* of the problem (CMP) is defined by

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \triangleq f(\boldsymbol{x}) + \boldsymbol{\lambda}^\top \boldsymbol{g}(\boldsymbol{x}) + \boldsymbol{\mu}^\top \boldsymbol{h}(\boldsymbol{x}), \tag{2.11}$$

where $\boldsymbol{\lambda} \geq \mathbf{0} \in \mathbb{R}^m$and $\boldsymbol{\mu} \in \mathbb{R}^n$ are the *Lagrange multipliers* associated with the inequality and equality constraints, respectively.[‡]

[‡] *The domain restriction on $\boldsymbol{\lambda}$ ensures that we deal with $\boldsymbol{g}(\boldsymbol{x}) \leq 0$ constraints and not $\boldsymbol{g}(\boldsymbol{x}) \geq 0$.*

The structure of the Lagrangian function allows us to determine, through extremization of the multipliers, whether a given point $\boldsymbol{x}$ is feasible:

$$F(\boldsymbol{x}) \triangleq \sup_{\boldsymbol{\lambda} \geq \mathbf{0}, \boldsymbol{\mu}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \begin{cases} f(\boldsymbol{x}) & \text{if } \boldsymbol{x} \in \boldsymbol{\Omega}, \\ +\infty & \text{otherwise.} \end{cases} \tag{2.12}$$

Therefore, solving the constrained problem (CMP) is conceptually equivalent to minimizing the unconstrained function $F(\boldsymbol{x})$.[§] In other words,

[§] *Taking values in the extended reals.*

$$\min_{\boldsymbol{x}\in\boldsymbol{\Omega}} f(\boldsymbol{x}) \Leftrightarrow \min_{\boldsymbol{x}\in\mathbb{R}^p} F(\boldsymbol{x}) \Leftrightarrow \min_{\boldsymbol{x}\in\mathbb{R}^p} \max_{\boldsymbol{\lambda}\geq\mathbf{0},\boldsymbol{\mu}} \mathcal{L}(\boldsymbol{x},\boldsymbol{\lambda},\boldsymbol{\mu}). \tag{2.13}$$

Note that the Lagrangian, as a function, has the same structure as a penalized method: a linear combination of the objective functions and the constraints or penalties, weighted by appropriate coefficients.

However, there is a fundamental difference between the unconstrained minimization of a penalized objective, and the problem in Eq. (2.12). In the penalized approach, the penalty coefficients are *fixed hyper-parameters* of the problem. In contrast, in the Lagrangian approach, the multipliers are *optimization variables*, which *dynamically* reweigh the different constraints throughout the optimization process.

Let us denote by p^* the optimal value of problem (CMP):

$$p^* \triangleq \inf_{\boldsymbol{x}\in\mathbb{R}^p} \sup_{\boldsymbol{\lambda}\geq\mathbf{0},\boldsymbol{\mu}} \mathcal{L}(\boldsymbol{x},\boldsymbol{\lambda},\boldsymbol{\mu}). \tag{2.14}$$

[¶] *Since the multipliers $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ are the arguments of the dual function, they are also called the "dual variables".*

We define the *dual function* $q(\boldsymbol{\lambda},\boldsymbol{\mu})$[¶] as the unconstrained minimum of the Lagrangian:

$$q(\boldsymbol{\lambda},\boldsymbol{\mu}) \triangleq \inf_{\boldsymbol{x}\in\mathbb{R}^p} \mathcal{L}(\boldsymbol{x},\boldsymbol{\lambda},\boldsymbol{\mu}). \tag{2.15}$$

For any feasible $\boldsymbol{x}$, and $\boldsymbol{\lambda}\geq\mathbf{0}$, $\boldsymbol{\mu}$, we have the following lower-bound:

$$q(\boldsymbol{\lambda},\boldsymbol{\mu}) = \inf_{\bar{\boldsymbol{x}}} \mathfrak{L}(\bar{\boldsymbol{x}},\boldsymbol{\lambda},\boldsymbol{\mu}) \leq f(\boldsymbol{x}) + \underbrace{\boldsymbol{\lambda}^\top \boldsymbol{g}(\boldsymbol{x})}_{\text{non-positive}} + \cancel{\boldsymbol{\mu}^\top \boldsymbol{h}(\boldsymbol{x})} \leq f(\boldsymbol{x}). \tag{2.16}$$

Note that the dual function is concave in $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$, regardless of the convexity of the original problem (CMP). The concavity of the dual function naturally invites the following definition, which concerns the *best* lower-bound achievable by the dual function.

[⊠] *Problem (CMP) is called the primal problem, and $\boldsymbol{x}$ are known as the "primal variables".*

We define the *dual problem*[⊠] of (CMP) as:

$$\max_{\boldsymbol{\lambda},\boldsymbol{\mu}} q(\boldsymbol{\lambda},\boldsymbol{\mu}) \quad \text{subject to} \quad \boldsymbol{\lambda}\geq\mathbf{0}. \tag{CMP}$$

[**] *Which is always non-negative.*

Solutions to the dual problem are called *dual optimal* points. The difference between the optimal values of the primal and dual problems $p^* - d^*$ is known as the *duality gap*:[**]

Theorem 2.2.1 [Ber16, Thm. 6.1.3] ***(Weak Duality)***

$$\sup_{\boldsymbol{\lambda}\geq\mathbf{0},\boldsymbol{\mu}} q(\boldsymbol{\lambda},\boldsymbol{\mu}) = \sup_{\boldsymbol{\lambda}\geq\mathbf{0},\boldsymbol{\mu}} \inf_{\boldsymbol{x}\in\mathbb{R}^p} \mathfrak{L}(\boldsymbol{x},\boldsymbol{\lambda},\boldsymbol{\mu}) \leq \inf_{\boldsymbol{x}\in\mathbb{R}^p} \sup_{\boldsymbol{\lambda}\geq\mathbf{0},\boldsymbol{\mu}} \mathfrak{L}(\boldsymbol{x},\boldsymbol{\lambda},\boldsymbol{\mu}) = \inf_{\boldsymbol{x}\in\mathbb{R}^p} F(\boldsymbol{x}) \tag{2.17}$$

Concisely, $d^* \leq p^*$.

A *saddle-point*[††] of the Lagrangian is a tuple $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ such that:

$$\mathfrak{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}, \boldsymbol{\mu}) \leq \mathfrak{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \leq \mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \quad \forall \boldsymbol{x}, \boldsymbol{\lambda} \geq \mathbf{0}, \boldsymbol{\mu}. \tag{2.18}$$

[††] *This is equivalent to a pure Nash equilibrium of the corresponding zero-sum Lagrangian game.*

We define the *saddle point problem* as the join optimization:

$$\boldsymbol{x}^* \in \operatorname*{argmin}_{\boldsymbol{x}} \mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \quad \text{and} \quad \boldsymbol{\lambda}^*, \boldsymbol{\mu}^* \in \operatorname*{argmax}_{\boldsymbol{\lambda}, \boldsymbol{\mu}} \mathfrak{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}, \boldsymbol{\mu}). \tag{SPP}$$

The existence of a saddle-point implies zero duality gap. This condition is known as *strong duality*.

Theorem 2.2.2 [BV04, §5.4.2]

If $(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$ is a saddle-point of the Lagrangian, then $\boldsymbol{x}$ is primal optimal, and $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ are dual optimal, and strong duality holds. Conversely, if $\boldsymbol{x}^*$ is primal optimal and $\boldsymbol{\lambda}^*$ and $\boldsymbol{\mu}^*$ are dual optimal for a problem with no duality gap, then $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ is a saddle-point of the Lagrangian.

The previous theorem states that solving a constrained optimization problem for which strong duality holds is equivalent to finding a saddle-point. However, strong duality generally does not hold for non-convex problems, and thus saddle-points may not exist.[‡‡]

[‡‡] *Some of the constraint qualifications presented in Section 2.2.6 can be used to establish strong duality in convex problems.*

Fig. 2.1 illustrates a non-convex problem[§§] with a duality gap $p^* - d^* > 0$. The shaded region represents all the realizable trade-offs between the objective and the constraints: $\mathcal{S} = \{(g(\boldsymbol{x}), f(\boldsymbol{x})) | \boldsymbol{x} \in \mathbb{R}^p\}$. Note that this region includes both feasible (the left half-space) and infeasible configurations.

[§§] *With a single inequality constraint.*

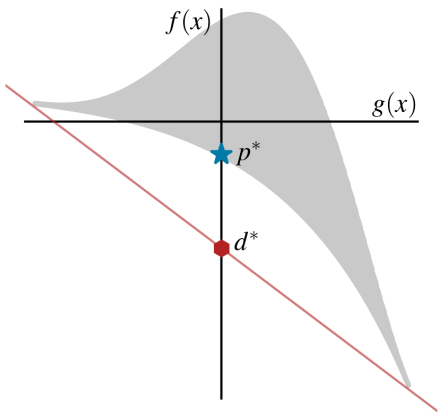


Figure 2.1: A non-convex constrained optimization problem with a positive duality gap.

For a general problem with equality and inequality constraints, evaluating the dual function is equivalent to solving a linear program over the set of realizable trade-offs:

$$q(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \inf_{\boldsymbol{x} \in \mathbb{R}^p} \mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \inf_{\boldsymbol{s} \in \mathcal{S}} [1, \boldsymbol{\lambda}, \boldsymbol{\mu}]^\top \boldsymbol{s} \tag{2.19}$$

Therefore, we can interpret the dual problem as searching for a *supporting hyperplane* that contains the set $\mathcal{S}$ in its positive half-space and has maximal intersection with the vertical axis. Said point of maximal intersection corresponds to the optimal dual value d^*, and the slope of the hyperplane corresponds to a dual optimal point $(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$.

We highlight that the inability to express p^* via one such hyperplane is directly connected to the inability of penalized methods to fully explore the Pareto frontier of the problem [BV04, §4.7.4].

For further details on the geometric interpretation of duality for Lagrangian problems, see Bertsekas [Ber16, §6.1].

2.2.5 *Karush-Kuhn-Tucker conditions*

The celebrated Karush-Kuhn-Tucker (KKT) conditions are a set of necessary conditions for optimality of a constrained optimization problem. The KKT conditions are a cornerstone of constrained optimization theory and constitute the foundation for many algorithmic approaches.

A feasible vector $\boldsymbol{x}$ is said to be *regular* if the gradients of all active constraints $\mathcal{A}(\boldsymbol{x})$ are linearly independent at $\boldsymbol{x}$.

Theorem 2.2.3 [Ber16, Prop. 4.3.1] ***(Karush-Kuhn-Tucker Necessary Conditions)***

Let $\boldsymbol{x}^*$ be a local minimum of problem (CMP) and assume $\boldsymbol{x}^*$ is regular. Then there exist unique Lagrange multipliers $\boldsymbol{\lambda}^*$ and $\boldsymbol{\mu}^*$ such that

$$\begin{aligned} \nabla_{\boldsymbol{x}} \mathfrak{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) &= \mathbf{0}, && \textit{(Stationarity)} \\ \boldsymbol{\lambda}^* &\geq \mathbf{0}, && \textit{(Dual feasibility)} \\ \boldsymbol{\lambda}_i &= 0 \quad \forall i \notin \mathcal{A}_I(\boldsymbol{x}^*). && \textit{(Complementary slackness)} \end{aligned} \tag{2.20}$$

Note that the stationarity condition states that, at a regular local minimum $\boldsymbol{x}^*$, the gradients of the objective and (active) constraints balance each other:

$$-\nabla f(\boldsymbol{x}^*) = \sum_{i \in \mathcal{A}_I(\boldsymbol{x}^*)} \boldsymbol{\lambda}_i^* \nabla g_i(\boldsymbol{x}^*) + \sum_{j=1}^{n} \boldsymbol{\mu}_j^* \nabla h_j(\boldsymbol{x}^*). \tag{2.21}$$

2.2.6 *Existence of Lagrange multipliers and constraint qualifications*

¶¶ *Regularity may be too strong an assumption for many applications.*

As we saw in Section 2.2.5, regularity¶¶ is sufficient to ensure the existence of (unique) Lagrange multipliers at a local minimum. Other assumptions of this kind, aimed at ensuring the existence of Lagrange multipliers, are known as *constraint qualifications*.

Different kinds of constraint qualifications arise from structure present in the problem's constraints. For example, the popular *Slater's condition* [Sla59] states:

Theorem 2.2.4 [Ber16, Prop. 4.3.9] ***(Slater's Condition for Convex Inequalities)***

Let $\boldsymbol{x}^*$ be local minimum of problem (CMP). Assume that the equality constraints are linear, the inequality constraints are convex, and that there exists a feasible point $\bar{\boldsymbol{x}}$ with

$$g_i(\bar{\boldsymbol{x}}) < 0 \quad \forall i \in \mathcal{A}_I(\boldsymbol{x}^*).$$

Then $\boldsymbol{x}^*$ satisfies the necessary conditions of Thm. 2.2.3.

The following theorem highlights the connection between the existence of Lagrange multipliers and the local geometry of the feasible set around a local minimum.

Theorem 2.2.5 [Ber16, Prop. 4.3.13] ***(Necessary and Sufficient Conditions for Existence of Lagrange Multipliers)***

Let $\boldsymbol{x}^*$ be a local minimum of problem (CMP). Then there exist Lagrange multipliers $\boldsymbol{\lambda}^*$ and $\boldsymbol{\mu}^*$ satisfying the KKT conditions (Eq. (2.20)) if and only if $\mathcal{F}(\boldsymbol{x}^*)$ does not contain any descent direction:

$$\nabla f(\boldsymbol{x}^*)^\top \boldsymbol{d} \geq 0 \quad \forall \boldsymbol{d} \in \mathcal{F}(\boldsymbol{x}^*).$$

In other words, Lagrange multipliers do *not* exist precisely when there is a feasibility-preserving direction (up to first order) along which the objective function decreases. This can only occur when the *true* set of feasible variations*** around $\boldsymbol{x}^*$ differs substantially from the linearized $\mathcal{F}(\boldsymbol{x}^*)$ [Ber16, p. 406].††† Constraint qualifications can be generally regarded as conditions that ensure alignment between these two sets.

Fig. 2.2 displays a hierarchy of classic constraint qualifications. For a modern, comprehensive review of constraint qualifications, we refer the reader to Giorgi [Gio18].

*** *This set called the tangent cone of $\boldsymbol{\Omega}$ at $\boldsymbol{x}^*$, denoted $\mathcal{T}_\Omega(\boldsymbol{x}^*)$; see Nocedal and Wright [NW06, Def. 12.2].*

††† *Points for which $\mathcal{T}_\Omega(\boldsymbol{x}^*)$ and $\mathcal{F}(\boldsymbol{x}^*)$ coincide are known as quasi-regular in Bertsekas [Ber16].*

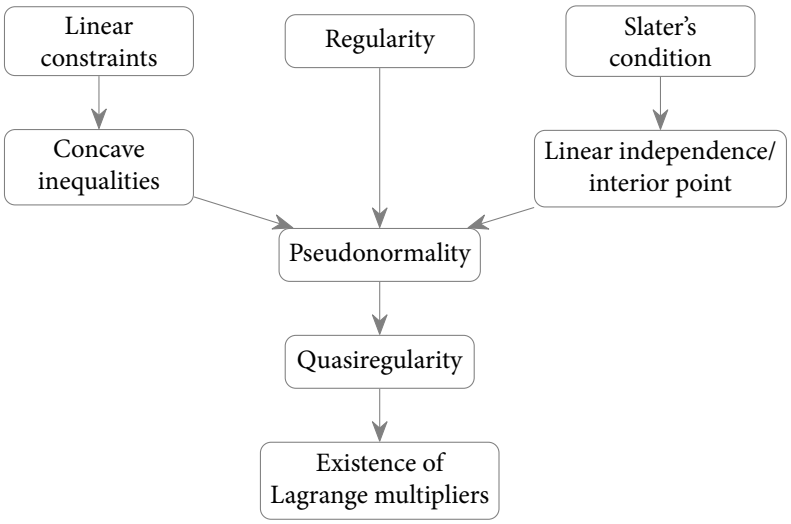


Figure 2.2: Hierarchy of constraint qualifications. Adapted from Bertsekas [Ber16, p. 407].

2.2.7 *Second-order conditions*

Given a feasible point $\boldsymbol{x}^*$ and Lagrange multipliers $\boldsymbol{\lambda}^*$ and $\boldsymbol{\mu}^*$ satisfying the KKT conditions in Eq. (2.20), we define the *critical cone* as

$$\mathcal{C}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \triangleq \left\{ \boldsymbol{d} \in \mathcal{F}(\boldsymbol{x}^*) \,\middle|\, \boldsymbol{d}^\top \nabla g_i(\boldsymbol{x}^*) = 0 \;\; \forall i \in \mathcal{A}_{\mathrm{I}}(\boldsymbol{x}^*) \text{ with } \boldsymbol{\lambda}_i^* > 0 \right\}. \quad (2.22)$$

Note that the critical cone is a subset of $\mathcal{F}(\boldsymbol{x}^*)$ and is constituted by directions that preserve feasibility *and* the set of active constraints (i.e., the constraints for which the Lagrange multipliers are positive).

From the stationarity KKT condition in Eq. (2.20), every $\boldsymbol{d} \in \mathcal{C}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ satisfies:

$$\boldsymbol{d}^\top \nabla \boldsymbol{f}(\boldsymbol{x}^*) = \sum_{i=1}^{m} \boldsymbol{\lambda}_i^* \boldsymbol{d}^\top \nabla g_i(\boldsymbol{x}^*) + \sum_{j=1}^{n} \boldsymbol{\mu}_j^* \boldsymbol{d}^\top \nabla h_j(\boldsymbol{x}^*) = 0. \quad (2.23)$$

Therefore, the critical cone $\mathcal{C}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ contains those feasibility-preserving directions for which first order information is insufficient to determine the change in the objective function.

In the remainder of this subsection, we assume that f, $\boldsymbol{g}$ and $\boldsymbol{h}$ are twice continuously differentiable.

Theorem 2.2.6 [NW06, Thm. 12.5] ***(Second-Order Necessary Conditions)***

Let $\boldsymbol{x}^*$ be a local minimum of problem (CMP) and assume $\boldsymbol{x}^*$ is regular. Let $\boldsymbol{\lambda}^*$ and $\boldsymbol{\mu}^*$ be the Lagrange multipliers that satisfy the KKT conditions Eq. (2.20). Then

$$\boldsymbol{d}^\top \nabla^2_{\boldsymbol{xx}} \mathfrak{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)\, \boldsymbol{d} \geq \boldsymbol{0}, \quad \text{for all } \boldsymbol{d} \in \mathcal{C}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*).$$

In other words, a valid KKT tuple $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ must yield a Hessian of the Lagrangian that is positive semi-definite over the critical cone. This clearly generalizes the second-order necessary conditions for unconstrained optimization [Ber16, Prop. 1.1.1] since the critical cone in the unconstrained case is the entire space $\mathbb{R}^p$.

Theorem 2.2.7 [NW06, Thm. 12.6] ***(Second-Order Sufficient Conditions)***

Consider a feasible point $\boldsymbol{x}^*$ and suppose there exist Lagrange multipliers $\boldsymbol{\lambda}^*$ and $\boldsymbol{\mu}^*$ satisfying the KKT conditions Eq. (2.20). Suppose also that

$$\boldsymbol{d}^\top \nabla^2_{\boldsymbol{xx}} \mathfrak{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)\, \boldsymbol{d} > \boldsymbol{0}, \quad \text{for all } \boldsymbol{d} \in \mathcal{C}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*),\ \boldsymbol{d} \neq \boldsymbol{0}.$$

Then $\boldsymbol{x}^*$ is a strict local minimum of problem (CMP).

2.2.8 *Interpretation of the Lagrange multipliers*

Given a valid KKT tuple $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$, it is possible to provide an intuitive interpretation of the Lagrange multipliers as indicating the sensitivity of the objective function to changes of the constraints.

Consider parametric generalization of the problem (CMP). Let $\boldsymbol{u} \in \mathbb{R}^m$ and $\boldsymbol{v} \in \mathbb{R}^n$:

$$\min_{\boldsymbol{x}\in\mathbb{R}^p} f(\boldsymbol{x}) \;\text{ subject to }\; \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{u} \;\text{ and }\; \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{v}. \tag{CMP$(\boldsymbol{u},\boldsymbol{v})$}$$

Depending on the signs of the components of $\boldsymbol{u}$ and $\boldsymbol{v}$, we are considering tightenings or relaxations of the constraints.

Theorem 2.2.8 [Ber16, Prop. 4.3.3] ***(Sensitivity)***

Let $\boldsymbol{x}^*$ be a local minimum of problem (CMP) and assume $\boldsymbol{x}^*$ is regular. Let $\boldsymbol{\lambda}^*$ and $\boldsymbol{\mu}^*$ be Lagrange multipliers satisfying the second-order sufficiency conditions in Thm. 2.2.7.

Then there exists an open sphere $\mathcal{S}$ centered at $(\mathbf{0},\mathbf{0})$ such that for any $(\boldsymbol{u},\boldsymbol{v}) \in \mathcal{S}$, there exist an $\boldsymbol{x}(\boldsymbol{u},\boldsymbol{v})$ that is a local minimum of problem (CMP$(\boldsymbol{u},\boldsymbol{v})$), and Lagrange multipliers $\boldsymbol{\lambda}(\boldsymbol{u},\boldsymbol{v})$ and $\boldsymbol{\mu}(\boldsymbol{u},\boldsymbol{v})$ that satisfy the KKT conditions for (CMP$(\boldsymbol{u},\boldsymbol{v})$).

The mapping $(\boldsymbol{u},\boldsymbol{v}) \mapsto (\boldsymbol{x}(\boldsymbol{u},\boldsymbol{v}), \boldsymbol{\lambda}(\boldsymbol{u},\boldsymbol{v}), \boldsymbol{\mu}(\boldsymbol{u},\boldsymbol{v}))$ is continuously differentiable on $\mathcal{S}$ and $\boldsymbol{x}(\mathbf{0},\mathbf{0}) = \boldsymbol{x}^*,\ \boldsymbol{\lambda}^*(\mathbf{0},\mathbf{0}) = \boldsymbol{\lambda}^*,\ \boldsymbol{\mu}^*(\mathbf{0},\mathbf{0}) = \boldsymbol{\mu}^*$.

Let $p(\boldsymbol{u},\boldsymbol{v})$ be the optimal value of problem (CMP$(\boldsymbol{u},\boldsymbol{v})$). Then

$$\nabla_{\boldsymbol{u}} p(\boldsymbol{u},\boldsymbol{v}) = -\boldsymbol{\lambda}(\boldsymbol{u},\boldsymbol{v}), \quad \nabla_{\boldsymbol{v}} p(\boldsymbol{u},\boldsymbol{v}) = -\boldsymbol{\mu}(\boldsymbol{u},\boldsymbol{v}).$$

Evidently, we have that $\nabla_{\boldsymbol{u}} p(\mathbf{0},\mathbf{0}) = -\boldsymbol{\lambda}^*$ and $\nabla_{\boldsymbol{v}} p(\mathbf{0},\mathbf{0}) = -\boldsymbol{\mu}^*$. Therefore, a small value[‡‡‡] of $\boldsymbol{\lambda}_i$ indicates that a small change in the constraint level for the i-th inequality constraint will not significantly affect the objective function.[§§§] Analogously, for example, a large positive values of the multiplier suggest a reduction (read improvement) of the objective function upon relaxing the constraint.

[‡‡‡] *Think of the case of an inactive constraint.*

[§§§] *Upon fully re-solving the problem with the updated constraint level.*

2.3 Algorithms for Constrained Optimization

In this section we present an overview of the several core algorithmic approaches for solving non-convex constrained optimization problems. Given the vast literature in the field of non-linear programming, with notable progress during the second half of the 20th century, our presentation is bound to be incomplete. We invite the interested reader to consult the works by Boyd and Vandenberghe [BV04], Nocedal and Wright [NW06], and Bertsekas [Ber16] for a comprehensive review of the field and pointers to other relevant literature.

We concentrate on Lagrangian-based methods since their reliance on well-developed technology for unconstrained minimization makes them most suitable to the type of optimization problems arising in deep learning applications. This is of particular importance given the highly specialized procedures used for training neural networks [Dah+23].

The differences in the generalization properties of multiple local minima require the use of optimization algorithms that integrate well with the learning dynamics of deep learning models. Informally, achieving feasibility too early in the optimization process may lead to convergence to a local minimum with poor generalization properties.

A number of techniques such as the Frank-Wolfe algorithm [FW56], and projection-based methods assume special structure of the objective or feasible set (like existence of an efficient proximal operator or a linear minimization oracle), making them less applicable to general non-convex optimization problems.

Notably, we do not touch upon interior point methods, given their heavy dependence on the use of of Newton's method for solving the resulting KKT systems. For a review on interior point methods, we refer the reader to Forsgren et al. [FGW02].

2.3.1 *Simultaneous Gradient Descent-Ascent*

As we discussed in Section 2.2.4, we cannot rely on the existence of saddle-points as strong duality typically does not hold for non-convex problems. In practice, we relax our notion of optimality to approximately-stationary points of the function $\Phi(\boldsymbol{x}) \triangleq \sup_{\boldsymbol{\lambda}\geq\mathbf{0},\boldsymbol{\mu}} \mathfrak{L}(\boldsymbol{x},\boldsymbol{\lambda},\boldsymbol{\mu})$. A point $\boldsymbol{x}$ is said to be an ϵ-*stationary point* of Φ if $\|\nabla\Phi(\boldsymbol{x})\| \leq \epsilon$.*

* *Φ is in general not differentiable, see Lin et al. [LJJ20] for more general stationarity definitions in that setting.*

Note that the search for approximately-stationary points of Φ breaks the symmetry of saddle point problem SPP, placing greater importance on the primal variables $\boldsymbol{x}$. This notion of optimality is appropriate for our purposes since we are usually more interested in finding a feasible solution with good optimality properties, rather than high-precision estimates of the multipliers.

This search for approximately-stationary points of Φ can be undertaken by first-order Lagrangian methods [AHU58]. The simplest Lagrangian method is the *(projected)*† *simultaneous gradient descent-ascent* scheme:

† *For preserving the non-negativity of the multipliers associated with the inequality constraints.*

$$\begin{cases} \boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \eta_{\text{primal}}\nabla_{\boldsymbol{x}}\mathfrak{L}(\boldsymbol{x}_t,\boldsymbol{\lambda}_t,\boldsymbol{\mu}_t) \\ \boldsymbol{\lambda}_{t+1} \leftarrow [\boldsymbol{\lambda}_t + \eta_{\text{dual}}\nabla_{\boldsymbol{\lambda}}\mathfrak{L}(\boldsymbol{x}_t,\boldsymbol{\lambda}_t,\boldsymbol{\mu}_t)]_+ \\ \boldsymbol{\mu}_{t+1} \leftarrow \boldsymbol{\mu}_t + \eta_{\text{dual}}\nabla_{\boldsymbol{\mu}}\mathfrak{L}(\boldsymbol{x}_t,\boldsymbol{\lambda}_t,\boldsymbol{\mu}_t). \end{cases} \tag{GDA}$$

Given the special structure of the Lagrangian function (Eq. (2.11)), we can simplify the GDA updates to:

$$\begin{cases} \boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \eta_{\text{primal}}\nabla_{\boldsymbol{x}}\mathfrak{L}(\boldsymbol{x}_t,\boldsymbol{\lambda}_t,\boldsymbol{\mu}_t) \\ \boldsymbol{\lambda}_{t+1} \leftarrow [\boldsymbol{\lambda}_t + \eta_{\text{dual}}\boldsymbol{g}(\boldsymbol{x}_t)]_+ \\ \boldsymbol{\mu}_{t+1} \leftarrow \boldsymbol{\mu}_t + \eta_{\text{dual}}\boldsymbol{h}(\boldsymbol{x}_t). \end{cases} \tag{2.24}$$

‡ *Such as* (GDA).

A tuple $(\boldsymbol{x}^*,\boldsymbol{\lambda}^*,\boldsymbol{\mu}^*)$ is said to be a *point of attraction* of an iterative scheme‡ if there exists an open set $S \subset \mathbb{R}^{p+m+n}$ such that when initializing at $(\boldsymbol{x}_0,\boldsymbol{\lambda}_0,\boldsymbol{\mu}_0) \in S$, then the sequence of iterates generated by said scheme belongs to S and converges to $(\boldsymbol{x}^*,\boldsymbol{\lambda}^*,\boldsymbol{\mu}^*)$.

The following result establishes the local convergence of the gradient descent-ascent scheme (GDA). Although the result is presented for a problem with only equality constraints, it also encompasses the case of inequality constraints.§

§ *One can use auxiliary variables to convert the inequality constraints into equalities and re-use the analysis.*

Theorem 2.3.1 [Ber16, Thm. 5.4.2] ***(Local Convergence of GDA)***

Assume that f and $\boldsymbol{h}$ are twice continuously differentiable and let $(\boldsymbol{x}^*, \boldsymbol{\mu}^*)$ be a local minimum-Lagrange multiplier pair. Assume also that $\boldsymbol{x}^*$ is regular and that the matrix $\nabla^2_{\boldsymbol{xx}}\mathfrak{L}(\boldsymbol{x}^*, \boldsymbol{\mu}^*)$ is positive definite. Then there exists $\bar{\alpha} > 0$ such that for all $\eta_{primal}, \eta_{dual} \in (0, \bar{\alpha}]$, the point $(\boldsymbol{x}^*, \boldsymbol{\mu}^*)$ is a point of attraction of the iteration (GDA), and if the generated sequence $\{(\boldsymbol{x}_k, \boldsymbol{\mu}_k)\}$ converges to $(\boldsymbol{x}^*, \boldsymbol{\mu}^*)$, then the rates of convergence of $||\boldsymbol{x}_k - \boldsymbol{x}^*||$ and $||\boldsymbol{\mu}_k - \boldsymbol{\mu}^*||$ are linear.

Lin et al. [LJJ20] provide more sophisticated results for the convergence of *stochastic* GDA on more general, nonconvex-concave min-max problems.

Having established the convergence of GDA under appropriate conditions, let us explore further properties of this iterative scheme.

2.3.2 *Dynamics of GDA*

Consider the update on the primal variables:

$$\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \eta_{\text{primal}} \nabla_{\boldsymbol{x}} \mathfrak{L}(\boldsymbol{x}_t, \boldsymbol{\lambda}_t, \boldsymbol{\mu}_t) \tag{2.25a}$$

$$= \boldsymbol{x}_t - \eta_{\text{primal}} \left[\nabla f(\boldsymbol{x}_t) + \sum_{i=1}^{m} \boldsymbol{\lambda}_t^{(i)} \nabla g_i(\boldsymbol{x}_t) + \sum_{j=1}^{n} \boldsymbol{\mu}_t^{(i)} \nabla h_i(\boldsymbol{x}_t) \right] . \tag{2.25b}$$

The primal update direction is a linear combination of the gradient of the objective function and the gradients of the constraints, weighted by the multipliers. Large values of the multipliers nudge the update $\nabla_{\boldsymbol{x}}\mathfrak{L}$ towards moving in a descent direction for the constraints, thus improving feasibility. Whenever the multipliers have small magnitude, the update is mostly determined by the gradient of the objective function.

The update for the inequality[¶] multipliers is given by:

$$\boldsymbol{\lambda}_{t+1} \leftarrow [\boldsymbol{\lambda}_t + \eta_{\text{dual}} \nabla_{\boldsymbol{\lambda}} \mathfrak{L}(\boldsymbol{x}_t, \boldsymbol{\lambda}_t, \boldsymbol{\mu}_t)]_+ = [\boldsymbol{\lambda}_t + \eta_{\text{dual}}\, \boldsymbol{g}(\boldsymbol{x}_t)]_+ . \tag{2.26}$$

[¶] *The analysis is analogous for equality multipliers, without the projection step.*

The update to the multiplier is determined by the *value* of the constraint at the current iterate $\boldsymbol{x}_t$. Whenever the i-th inequality constraint is strictly satisfied, $g_i(\boldsymbol{x}_t) < 0$, and the value of the corresponding multiplier is decreased.[⊠] In contrast, when the constraint is violated, $g_i(\boldsymbol{x}_t) > 0$ and the multiplier is increased.

[⊠] *Unless the previous multiplier value was zero.*

In other words, the value the multiplier corresponds to the accumulated constraint violation throughout the optimization process. Constraints that are consistently violated will have corresponding multipliers with large magnitude[**] and will exert a strong influence on the primal iterates to improve their feasibility.

[**] *Recall that equality constraints do not have a sign restriction in their multipliers.*

2.3.3 *Computational cost of GDA*

GDA is particularly suitable for deep learning problems. As a first order method, the computational cost of each iteration is dominated by the computation of objective func-

tion, the constraints and their corresponding gradients.

[††] *This is default in modern deep learning frameworks.*

In *reverse-mode automatic differentiation*,[††] a function is evaluated first and then its gradient is computed using the chain rule by following the computational graph in reverse order [GBC16, §6.5]. The evaluation of the function is known as the *forward pass*, while the computation of the gradient is called a *backward pass* through the computational graph.

The main ingredients required for the execution of a single iteration of GDA are:

- The gradient of the objective function $\nabla f(\boldsymbol{x}_t)$.
- The value and gradients of the constraints $\boldsymbol{g}(\boldsymbol{x}_t)$ and $\boldsymbol{h}(\boldsymbol{x}_t)$.
- The linear combination of objective and constraint gradients $\nabla_{\boldsymbol{x}} \mathfrak{L}(\boldsymbol{x}_t, \boldsymbol{\lambda}_t, \boldsymbol{\mu}_t)$.

In the presence of many constraints, storing individual gradients for each constraints would cause a significant memory overhead. Fortunately, this can be avoided by arranging the computation astutely:

1. Evaluate the objective function $f(\boldsymbol{x}_t)$ and the constraints $\boldsymbol{g}(\boldsymbol{x}_t)$ and $\boldsymbol{h}(\boldsymbol{x}_t)$.
2. Compute the Lagrangian $\mathfrak{L}(\boldsymbol{x}_t, \boldsymbol{\lambda}_t, \boldsymbol{\mu}_t)$.
3. Compute the gradient $\nabla_{\boldsymbol{x}} \mathfrak{L}(\boldsymbol{x}_t, \boldsymbol{\lambda}_t, \boldsymbol{\mu}_t)$ using automatic differentiation.
4. Update the primal variables to obtain $\boldsymbol{x}_{t+1}$.
5. Update the dual variables using the previously computed $\boldsymbol{g}(\boldsymbol{x}_t)$ and $\boldsymbol{h}(\boldsymbol{x}_t)$ to obtain $\boldsymbol{\lambda}_{t+1}$ and $\boldsymbol{\mu}_{t+1}$.

Given the objective and constraint values, computing the value of the Lagrangian can be done in $\mathcal{O}(m + n)$ time. Typically the number of constraints is many orders of magnitude smaller than the number of parameters in the model, making the cost so far dominated by the evaluation of the objective and constraint functions.

At this stage, for fixed $\boldsymbol{\lambda}_t$ and $\boldsymbol{\mu}_t$, the contribution of the constraints to the Lagrangian may be regarded as an additive *regularization* or *penalty* to objective [GBC16, §5.2.2]. Therefore, the cost of computing the gradient of the Lagrangian with respect to $\boldsymbol{x}$ and the subsequent primal update do not represent any additional computational burden compared to the additive regularization approach (see Section 2.1.2).

Finally, the update of the Lagrange multipliers also requires $\mathcal{O}(m + n)$ time, as it involves simple element-wise operations based on the measured constraint values. Again, this cost is negligible considering the typical scale of a deep learning model.

We highlight that it is common for the objective and constraints to share a large portion of the computational graph, which can translate into further reductions in computational cost. For example, constraints on the output of a neural network can be calculated as a byproduct of the computation of the training loss on a given mini-batch.

The presentation of the `Cooper` library in Chapter 12 expands on the practical aspects of implementing GDA-like schemes for deep learning problems.

2.3.4 *Alternating updates*

We have seen how the GDA approach can provably (locally) converge to KKT pairs. However, in practice, simultaneous updates may lead to oscillatory or unstable behavior. A simple modification to the GDA scheme is to *alternate* the updates of the primal and dual variables. Alternating schemes have been shown to improve the stability properties of the optimization process [Gid+19b].

The following recurrences formalize the *alternating primal-dual gradient descent-ascent* and *alternating dual-primal gradient descent-ascent* schemes. The core difference between these two variants is the order in which the primal and dual variables are updated.

$$\begin{cases} \boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \eta_{\text{primal}} \nabla_{\boldsymbol{x}} \mathfrak{L}(\boldsymbol{x}_t, \boldsymbol{\lambda}_t, \boldsymbol{\mu}_t) \\ \boldsymbol{\lambda}_{t+1} \leftarrow [\boldsymbol{\lambda}_t + \eta_{\text{dual}}\, \boldsymbol{g}(\boldsymbol{x}_{t+1})]_+ \\ \boldsymbol{\mu}_{t+1} \leftarrow \boldsymbol{\mu}_t + \eta_{\text{dual}} \boldsymbol{h}(\boldsymbol{x}_{t+1}) \end{cases} \tag{APD-GDA}$$

$$\begin{cases} \boldsymbol{\lambda}_{t+1} \leftarrow [\boldsymbol{\lambda}_t + \eta_{\text{dual}} \boldsymbol{g}(\boldsymbol{x}_t)]_+ \\ \boldsymbol{\mu}_{t+1} \leftarrow \boldsymbol{\mu}_t + \eta_{\text{dual}} \boldsymbol{h}(\boldsymbol{x}_t) \\ \boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \eta_{\text{primal}} \nabla_{\boldsymbol{x}} \mathfrak{L}(\boldsymbol{x}_t, \boldsymbol{\lambda}_{t+1}, \boldsymbol{\mu}_{t+1}) \end{cases} \tag{ADP-GDA}$$

Although these two variants are intimately connected,[‡‡] the ADP-GDA scheme can be more amenable to implementation in deep learning frameworks such as PyTorch. Strictly speaking, in the APD-GDA scheme we need to evaluate the loss and constraints at $\boldsymbol{x}_t$ to carry out the primal update. Then, we need to re-evaluate the constraints functions at the new iterate $\boldsymbol{x}_{t+1}$ to update the multipliers.

[‡‡] *Yet not equivalent algorithms!*

In the "full-batch" optimization setting, ingenious caching can re-use the computation performed during the evaluation of $\boldsymbol{g}(\boldsymbol{x}_{t+1})$ and $\boldsymbol{h}(\boldsymbol{x}_{t+1})$ for the subsequent primal update. However, in the stochastic setting, the validity of the cache-based approach is not immediately clear since the mini-batches of data might differ between the primal and dual updates.

In contrast, to carry out the APD-GDA updates, we require $\boldsymbol{g}(\boldsymbol{x}_t)$, $\boldsymbol{h}(\boldsymbol{x}_t)$ (for the dual updates), along with $f(\boldsymbol{x}_t)$ (for the primal update).[§§] Note that the primal iterate at which these functions are evaluated is always $\boldsymbol{x}_t$, and thus no re-evaluation or change of mini-batch is involved.

[§§] *And their automatically-differentiated gradients, of course.*

2.3.5 *Extragradient updates*

The *extragradient* method [Kor76] is another popular technique for solving saddle-point problems, with better stability and convergence properties than GDA. The ExtraGradient method visits an "extrapolated" (read simulated) point by performing a trial step in the direction of the current gradient. Then, the gradient at the extrapolated point is used to update the current (non-extrapolated) iterate.

Formally, the ExtraGradient method for solving finding saddle-points of the Lagrangian (SPP) is determined by the following recurrence relations:

$$\begin{cases} \hat{\boldsymbol{x}}_t & \leftarrow \boldsymbol{x}_t - \eta_{\text{primal}} \nabla_{\boldsymbol{x}} \mathfrak{L}(\boldsymbol{x}_t, \boldsymbol{\lambda}_t, \boldsymbol{\mu}_t) \\ \hat{\boldsymbol{\lambda}}_t & \leftarrow \boldsymbol{\lambda}_t + \eta_{\text{dual}} \left[\boldsymbol{g}(\hat{\boldsymbol{x}}_t)\right]_+ \\ \hat{\boldsymbol{\mu}}_t & \leftarrow \boldsymbol{\mu}_t + \eta_{\text{dual}} \nabla_{\boldsymbol{\mu}} \boldsymbol{h}(\hat{\boldsymbol{x}}_t) \end{cases} \tag{Extrapolation}$$

$$\begin{cases} \boldsymbol{x}_{t+1} & \leftarrow \boldsymbol{x}_t - \eta_{\text{primal}} \nabla_{\boldsymbol{x}} \mathfrak{L}(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{\lambda}}_t, \hat{\boldsymbol{\mu}}_t) \\ \boldsymbol{\lambda}_{t+1} & \leftarrow \boldsymbol{\lambda}_t + \eta_{\text{dual}} \left[\boldsymbol{g}(\hat{\boldsymbol{x}}_t)\right]_+ \\ \boldsymbol{\mu}_{t+1} & \leftarrow \boldsymbol{\mu}_t + \eta_{\text{dual}} \nabla_{\boldsymbol{\mu}} \boldsymbol{h}(\hat{\boldsymbol{x}}_t) \end{cases} \tag{Update}$$

Note that, as its name indicates, the ExtraGradient method requires the computation of the Lagrangian gradient and constraints at two different points for each iteration.

Theorem 2.3.2 [Kor76, Thm. 1] ***(Convergence of ExtraGradient)***

Consider the problem of finding a saddle-point of a function $\phi : S \times Q \to \mathbb{R}$, and assume that:

1. *The sets Q and S are closed and convex.*
2. *The function ϕ is convex-concave, differentiable, and its partial derivatives are L Lipschitz-continuous on $Q \times S$:*

$$||\nabla_{\boldsymbol{u}} \phi(\boldsymbol{u}, \boldsymbol{v}) - \nabla_{\boldsymbol{u}} \phi(\boldsymbol{u}', \boldsymbol{v}')|| \leq L \left|\left| [\boldsymbol{u}, \boldsymbol{v}]^\top - [\boldsymbol{u}', \boldsymbol{v}']^\top \right|\right|,$$
$$||\nabla_{\boldsymbol{v}} \phi(\boldsymbol{u}, \boldsymbol{v}) - \nabla_{\boldsymbol{v}} \phi(\boldsymbol{u}', \boldsymbol{v}')|| \leq L \left|\left| [\boldsymbol{u}, \boldsymbol{v}]^\top - [\boldsymbol{u}', \boldsymbol{v}']^\top \right|\right|.$$

3. *The set of saddle points of ϕ is non-empty.*

Then the sequence of iterates generated by the ExtraGradient method with step-size in $(0, \frac{1}{L})$, converges to a saddle-point of ϕ.

Gidel et al. [Gid+19b] extend on a method by Popov [Pop80] which re-uses the gradient evaluated at the previous extrapolation point to reduce the computational cost of the ExtraGradient method. Gidel et al. [Gid+19b] refer to this technique as *extrapolation from the past* and establish convergence results in the stochastic regime under assumptions of bounded gradient variance and compactness of the domain.

2.3.6 *Quadratic penalty method*

Penalty methods are a popular approach for solving constrained optimization problems. Penalty methods transform the constrained problem into the unconstrained minimization of an objective comprising the original objective function and additional terms encouraging[¶¶] the satisfaction of the constraints.

[¶¶] *For example, by being positive when the constraint is violated and zero otherwise.*

The *quadratic penalty method* [Cou43] transforms the constrained problem (CMP) into the unconstrained objective:

$$\mathfrak{L}_{\text{QP}}(\boldsymbol{x};\rho) \triangleq f(\boldsymbol{x}) + \frac{\rho}{2}||\boldsymbol{h}(\boldsymbol{x})||^2 + \frac{\rho}{2}||\left[\boldsymbol{g}(\boldsymbol{x})\right]_+||^2, \tag{2.28}$$

where $\rho > 0$ is known as the *penalty coefficient*.

Naturally, other types of penalty functions can be used. See Nocedal and Wright [NW06, §17.2] and Bertsekas [Ber16, §5.3] for further details.

The key idea behind the quadratic penalty method is that by increasing the penalty coefficient ρ, the minimizer of the penalized objective $\mathfrak{L}_{\text{QP}}(\cdot;\rho)$ will approach the solution of the original constrained problem (CMP).

Theorem 2.3.3 [NW06, Thm. 17.1] ***(Convergence of the Quadratic Penalty Method)***

Consider a positive sequence $\{\rho_t\} \uparrow \infty$, and let $\boldsymbol{x}_t$ be a global minimizer of $\mathfrak{L}_{QP}(\boldsymbol{x};\rho)$. Then every limit point of the sequence $\{\boldsymbol{x}_t\}$ is a global minimizer of the problem (CMP).

The assumption of global optimality in the Thm. 2.3.3 too strong in practice. Moreover, when allowing for inexact minimization, the sequence successive minimization of the quadratic penalty objective may converge to infeasible points, or to a KKT point which may not be a minimizer [NW06, Thm. 17.2].

Although the quadratic penalty method is extensively used in practice, an undesirable side-effect of the unbounded increase of the penalty coefficient ρ is the ill-conditioning of the optimization problem $\min \mathfrak{L}_{\text{QP}}(\cdot;\rho)$. The condition number of the Hessian matrix $\nabla^2_{\boldsymbol{xx}}\mathfrak{L}_{\text{QP}}(\cdot;\rho)$ grows with ρ, leading to numerical instability and round-off errors. Bertsekas [Ber16, p. 476] recommends the use of Newton-like methods and double-precision arithmetic. However, these techniques are not practical in the training of large-scale deep learning models.

2.3.7 *Augmented Lagrangian method*

In this section we discuss the *Augmented Lagrangian method*[***] (ALM) [Pow69; Hes69; HB70]. We refer the interested reader to [Ber82; BM14] for comprehensive treaties.

[***] *Also known as the "method of multipliers"*

The ALM alleviates the ill-conditioning challenges of the quadratic penalty method by maintaining explicit estimates of the multipliers, much like in the GDA scheme.

The *augmented Lagrangian function* for the problem (CMP) is defined[†††] as

$$\mathfrak{L}_{\text{A}}(\boldsymbol{x},\boldsymbol{\lambda},\boldsymbol{\mu};\rho) \triangleq f(\boldsymbol{x}) + \frac{\rho}{2}\left\|\boldsymbol{h}(\boldsymbol{x}) + \frac{\boldsymbol{\mu}}{\rho}\right\|^2 + \frac{\rho}{2}\left\|\left[\boldsymbol{g}(\boldsymbol{x}) + \frac{\boldsymbol{\lambda}}{\rho}\right]_+\right\|^2. \tag{2.29}$$

[†††] *For a derivation of the ALM as a proximal method see Nocedal and Wright [NW06, p. 523].*

The augmented Lagrangian function penalizes the violations modified by the "shifts" $\frac{\boldsymbol{\mu}}{\rho}$ and $\frac{\boldsymbol{\lambda}}{\rho}$ [BM14]. Note that when $\boldsymbol{\mu} = \mathbf{0}$ and $\boldsymbol{\lambda} = \mathbf{0}$, the augmented Lagrangian function reduces to the quadratic penalty objective.

We can express the augmented Lagrangian function equivalently as

$$\mathfrak{L}_{\mathrm{A}}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}; \rho) = f(\boldsymbol{x}) + \frac{1}{2\rho}\|\boldsymbol{\mu} + \rho \boldsymbol{h}(\boldsymbol{x})\|^2 + \frac{1}{2\rho}\| \left[\boldsymbol{\lambda} + \rho \boldsymbol{g}(\boldsymbol{x})\right]_+ \|^2. \tag{2.30}$$

‡‡‡ Along with its more realistic version by Bertsekas [Ber16, Prop. 5.2.3].

The following theorem‡‡‡ shows that given a local solution $\boldsymbol{x}^*$ and a sufficiently accurate estimate of the multipliers, the ALM problem identifies $\boldsymbol{x}^*$ as a strict local minimizer *without requiring unbounded increases in the penalty coefficient*.

Theorem 2.3.4 [NW06, Thm. 17.5]

Let $\boldsymbol{x}^*$ be a local minimum of problem (CMP). Assume $\boldsymbol{x}^*$ is regular and that the second-order sufficient conditions from Thm. 2.2.7 hold for $\boldsymbol{\lambda} = \boldsymbol{\lambda}^*$ and $\boldsymbol{\lambda} = \boldsymbol{\mu}^*$. Then there exists a threshold value $\bar{\rho} > 0$ such that for all $\rho > \bar{\rho}$, the point $\boldsymbol{x}^*$ is a strict local minimizer of $\mathfrak{L}_A(\boldsymbol{x}, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*; \rho)$.

Let us turn to the algorithmic components of the ALM. Suppose $\tilde{\boldsymbol{x}}$ is an approximate minimizer of $\mathfrak{L}_{\mathrm{A}}(\cdot, \boldsymbol{\lambda}, \boldsymbol{\mu}; \rho)$. Therefore, we have that

$$\begin{aligned}
\mathbf{0} &\approx \nabla_{\boldsymbol{x}} \mathfrak{L}_{\mathrm{A}}(\tilde{\boldsymbol{x}}, \boldsymbol{\lambda}, \boldsymbol{\mu}; \rho) && \text{(2.31a)}\\
&= \nabla_{\boldsymbol{x}} f(\tilde{\boldsymbol{x}}) + \sum_{i=1}^{m} \left[\boldsymbol{\lambda}_i + \rho g_j(\tilde{\boldsymbol{x}})\right]_+ \nabla_{\boldsymbol{x}} g_i(\tilde{\boldsymbol{x}}) + \sum_{j=1}^{n} (\boldsymbol{\mu}_j + \rho h_j(\tilde{\boldsymbol{x}})) \nabla_{\boldsymbol{x}} h_j(\tilde{\boldsymbol{x}}) && \text{(2.31b)}\\
&= \nabla_{\boldsymbol{x}} \mathfrak{L}(\tilde{\boldsymbol{x}}, \left[\boldsymbol{\lambda} + \rho \boldsymbol{g}(\tilde{\boldsymbol{x}})\right]_+, \boldsymbol{\mu} + \rho \boldsymbol{h}(\tilde{\boldsymbol{x}})). && \text{(2.31c)}
\end{aligned}$$

Note that the last equality corresponds to the stationarity KKT conditions (Eq. (2.20)), with a choice of multipliers $\boldsymbol{\lambda} \leftarrow \left[\boldsymbol{\lambda} + \rho \boldsymbol{g}(\tilde{\boldsymbol{x}})\right]_+$ and $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} + \rho \boldsymbol{h}(\tilde{\boldsymbol{x}})$. In other words, with said choice of dual variables, the first condition needed for optimality is satisfied *automatically*. This motivates the following algorithm:

Algorithm 1 Augmented Lagrangian Method

Input: Initial conditions $\boldsymbol{x}_0$, $\boldsymbol{\lambda}_0$, $\boldsymbol{\mu}_0$, ρ_0

1: **for** $t = 0, 1, 2, \ldots$
2: $\quad \boldsymbol{x}_{t+1} \leftarrow$ Approx. minimizer of $\mathfrak{L}_{\mathrm{A}}(\boldsymbol{x}, \boldsymbol{\lambda}_t, \boldsymbol{\mu}_t; \rho_t)$ (possibly warm-starting at $\boldsymbol{x}_t$)
3: $\quad \boldsymbol{\lambda}_{t+1} \leftarrow \left[\boldsymbol{\lambda}_t + \rho_t \boldsymbol{g}(\boldsymbol{x}_{t+1})\right]_+$
4: $\quad \boldsymbol{\mu}_{t+1} \leftarrow \boldsymbol{\mu}_t + \rho_t \boldsymbol{h}(\boldsymbol{x}_{t+1})$
5: $\quad$ Pick $\rho_{t+1} > \rho_t$ according to some heuristic or schedule
6: $\quad$ **if** a convergence test for problem (CMP) is satisfied:
7: $\quad\quad$ **return** $\boldsymbol{x}_{t+1}$, $\boldsymbol{\lambda}_{t+1}$, $\boldsymbol{\mu}_{t+1}$

Bertsekas [Ber16, §5.2] lists several heuristics for updating the penalty coefficient ρ.

In the language of Sections 2.3.4 and 2.3.5, Algo. 1 can be described as an alternating "primal-dual" scheme, where the primal objective is the augmented (and not just the regular) Lagrangian function.

Given our machine learning context, it is of particular interest to consider the case where the "approximate minimization" in step 2 of Algo. 1 is a single step of gradient descent. In this case, the primal parameters are updated using a value of the multipliers after a *simulated* update based on the constraint violations at the current iterate $\boldsymbol{x}_t$, reminiscent[§§§] of the ExtraGradient method. The update on the multipliers in step 3 is based on the new iterate $\boldsymbol{x}_{t+1}$ (and not an extrapolated estimate thereof).

[§§§] *But not equivalent!*

Formally, the ALM with primal-gradient updates consists of the following steps:

$$\begin{cases} \hat{\boldsymbol{\lambda}}_t & \leftarrow \boldsymbol{\lambda}_t + \eta_{\text{dual}} \left[\boldsymbol{g}(\boldsymbol{x}_t)\right]_+ \\ \hat{\boldsymbol{\mu}}_t & \leftarrow \boldsymbol{\mu}_t + \eta_{\text{dual}} \nabla_{\boldsymbol{\mu}} \boldsymbol{h}(\boldsymbol{x}_t) \\ \textit{No primal extrapolation.} \end{cases} \tag{2.32a}$$

$$\begin{cases} \boldsymbol{x}_{t+1} & \leftarrow \boldsymbol{x}_t - \eta_{\text{primal}} \nabla_{\boldsymbol{x}} \mathfrak{L}(\boldsymbol{x}_t, \hat{\boldsymbol{\lambda}}_t, \hat{\boldsymbol{\mu}}_t) \\ \boldsymbol{\lambda}_{t+1} & \leftarrow \boldsymbol{\lambda}_t + \eta_{\text{dual}} \left[\boldsymbol{g}(\boldsymbol{x}_{t+1})\right]_+ \\ \boldsymbol{\mu}_{t+1} & \leftarrow \boldsymbol{\mu}_t + \eta_{\text{dual}} \nabla_{\boldsymbol{\mu}} \boldsymbol{h}(\boldsymbol{x}_{t+1}) \end{cases} \tag{2.32b}$$

Note that as an alternating primal-dual method, the ALM described in Algo. 1 suffers the same computational shortcomings of the APD-GDA scheme in the stochastic setting.

2.3.8 *Non-differentiable constraints*

In this section, we briefly present the idea of *proxy-constraints* introduced by Cotter et al. [Cot+19b]. In many applications, requirements may come in the form of a non-differentiable constraint. For example, fairness constraints may demand that a model predicts at least a certain threshold of positive outcomes for a minority group [BHN23].

A naive solution to this issue would be to replace the non-differentiable constraint with a differentiable approximation or *surrogate*. The use of surrogate losses is widespread in the machine learning community, as discussed in Section 2.1.1.

However, a naively replacing the constraints by surrogates would lead to saddle-points that are feasible for the surrogate—but not for the original constraints. The key insight of Cotter et al. [Cot+19b] is to replace the constraints by surrogates *only when it is actually needed*.

Let us consider once more the GDA updates.[¶¶¶] Note that, while updating the primal variables requires the *gradient* of the constraints, the dual variables are updated based on the *value* of the constraints. Therefore, the Lagrange multipliers can be updated so as to satisfy the original constraints.

[¶¶¶] *This analysis extends to alternating schemes.*

$$\begin{cases} \boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \eta_{\text{primal}} \left[\nabla f(\boldsymbol{x}_t) + \sum_{i=1}^m \boldsymbol{\lambda}_t^{(i)} \nabla g_i(\boldsymbol{x}_t) + \sum_{j=1}^n \boldsymbol{\mu}_t^{(i)} \nabla h_i(\boldsymbol{x}_t)\right] \\ \boldsymbol{\lambda}_{t+1} \leftarrow \left[\boldsymbol{\lambda}_t + \eta_{\text{dual}} \boldsymbol{g}(\boldsymbol{x}_t)\right]_+ \\ \boldsymbol{\mu}_{t+1} \leftarrow \boldsymbol{\mu}_t + \eta_{\text{dual}} \boldsymbol{h}(\boldsymbol{x}_t). \end{cases} \tag{2.33}$$

Concretely, the *proxy-constraint* approach considers the *non-zero sum game*:

$$\begin{cases} \boldsymbol{x}^* & \in \operatorname*{argmin}_{\boldsymbol{x}} \mathfrak{L}_{\boldsymbol{x}}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \triangleq f(\boldsymbol{x}) + \boldsymbol{\lambda}^\top \tilde{\boldsymbol{g}}(\boldsymbol{x}) + \boldsymbol{\mu}^\top \tilde{\boldsymbol{h}}(\boldsymbol{x}), \\ \boldsymbol{\lambda}^*, \boldsymbol{\mu}^* & \in \operatorname*{argmax}_{\boldsymbol{\lambda}, \boldsymbol{\mu}} \mathfrak{L}_{\boldsymbol{\lambda}, \boldsymbol{\mu}}(\boldsymbol{x}^*, \boldsymbol{\lambda}, \boldsymbol{\mu}) \triangleq \boldsymbol{\lambda}^\top \boldsymbol{g}(\boldsymbol{x}^*) + \boldsymbol{\mu}^\top \boldsymbol{h}(\boldsymbol{x}^*), \end{cases} \tag{2.34}$$

where $\tilde{\boldsymbol{g}}$ and $\tilde{\boldsymbol{h}}$ are differentiable surrogates or *proxy-constraints* for the non-differentiable constraints $\boldsymbol{g}$ and $\boldsymbol{h}$.

Given the non-zero-sum nature of the game (2.34), it is not possible to find Nash equilibria (i.e., saddle-points) efficiently. Cotter et al. [Cot+19b] show that weaker solution concept (which can be found efficiently) is sufficient to guarantee near-optimality and near-feasibility in expectation.

In Chapters 6 and 8 we demonstrate the practical success of the proxy-constraint approach in the context of training neural networks with compression and fairness constraints. Proxy-constraints are natively supported in `Cooper` (see Chapter 12).

Part II

CONTRIBUTIONS

Prologue to the First Contribution

3

Article Details

Jose Gallego-Posada, Juan Ramirez, Akram Erraqabi, Yoshua Bengio and Simon Lacoste-Julien. **Controlled Sparsity via Constrained Optimization or: How I Learned to Stop Tuning Penalties and Love Constraints**. This paper was published at *NeurIPS*, 2022.*

* *An earlier version of this work was presented at the LatinX in AI workshop at NeurIPS 2021.*

Author Contributions

Jose Gallego-Posada proposed the original idea, contributed to the experiments, led the writing of the paper and acted as Juan Ramirez' internship advisor. Juan Ramirez conducted this research as an (undergrad!) intern at Mila and led the execution of the experiments. Akram Erraqabi contributed to the experiments, the writing of the paper and proposed to focus the narrative around the concept of controllability. Juan Ramirez and Jose Gallego-Posada were the main developers of the codebase. Juan Ramirez and Akram Erraqabi identified the issue and solution to the optimization challenges present in L_0-spare ResNet models†. Jose Gallego-Posada contributed the idea of dual restarts and its game-theoretic interpretation as a best-response scheme. Yoshua Bengio provided feedback on the final manuscript. Simon Lacoste-Julien provided supervision throughout the project.

† *I.e. the need for separate optimizers between gates and model parameters.*

Context

The idea for this paper was inspired by the excellent blogs by Degrave and Korshunova [DK21b; DK21a] on optimizing multi-objective problems using linear combinations of the objectives. We wanted to show the advantages afforded by constrained optimization over the ubiquitous penalization approach in a reasonably complex application for the machine-learning community. Additionally, we wanted to demonstrate that the constrained optimization approach was successful in practice and easily integrated with existing gradient-based training pipelines for deep-learning models.

Although the paper revolves heavily around the sparsity application, the insights we present are applicable to other machine learning problems where a "core" objective (such as the training loss) is subject to an interpretable "regularization" term (such as the sparsity constraints).

As a spin-off of the code developed for this paper, we have released the `Cooper` library, which provides several constrained optimization techniques in the PyTorch framework. We provide a brief overview of the library in Chapter 12.

4 Controlled Sparsity via Constrained Optimization or: How I Learned to Stop Tuning Penalties and Love Constraints

Abstract

The performance of trained neural networks is robust to harsh levels of pruning. Coupled with the ever-growing size of deep learning models, this observation has motivated extensive research on learning sparse models. In this work, we focus on the task of controlling the level of sparsity when performing sparse learning. Existing methods based on sparsity-inducing penalties involve expensive trial-and-error tuning of the penalty factor, thus lacking direct control of the resulting model sparsity. In response, we adopt a *constrained* formulation: using the gate mechanism proposed by Louizos et al. [LWK18], we formulate a constrained optimization problem where sparsification is guided by the training objective and the desired sparsity target in an end-to-end fashion. Experiments on CIFAR-{10, 100}, TinyImageNet, and ImageNet using WideResNet and ResNet{18, 50} models validate the effectiveness of our proposal and demonstrate that we can reliably achieve pre-determined sparsity targets without compromising on predictive performance.

4.1 Introduction

The great expressive power of neural networks as function approximators comes with an inherent need for regularization [BCV13]. Regularization aimed at controlling the generalization behavior of the network can be realized by a wide range of mechanisms: *explicitly*, through the use of additive penalties in training [VC74]; *implicitly*, via the choice of optimization algorithm used to train the network [NTS15; CLG01]; or even be in-grained in the *architecture* of the network through stochastic computation [Sri+14].

Commonly used neural networks result in *overparametrized* models, whose performance is robust to harsh levels of parameter pruning [HMD16; UMW17; FC19; GEH19]. Thus, regularization techniques aimed at learning sparse models can drastically reduce the computational cost associated with the learnt model by removing unnecessary parameters, and retain good performance in the learning task. Given the recent research trends which explore the capabilities of ever more ambitious large-scale models [Bro+20], developing techniques which provide reliable training of *sparsified* models becomes crucial for deploying them in massively-used systems, or on resource-constrained devices.

Pruning methods aim to reduce the storage and/or computational footprint of a model by discarding individual parameters [HMD16; MAV17] or groups thereof [Li+17; LAJ19; Nek+17], while inducing minimal distortion in the model's predictions. These methods can be further categorized based on whether the sparse model is obtained *while* or *after* training the model (also known as *in-training* and *post-training* sparsification).

Traditional post-training methods rely on heuristic rankings of the weights or filters to be pruned, often based on parameter magnitudes [LDS90; HMD16]. Despite their simplicity, these methods usually require retraining the weights to maintain high accuracy after pruning, and thus incur in additional computational overhead. On the other hand, in-training methods which *learn* a good sparsity pattern by augmenting the training loss with sparsity-inducing penalties [LWK18; LAJ19] do not perform fine-tuning, but face challenges regarding the tuning and interpretability of the penalty hyperparameter.

* *Our code is available at:* `https://github.com/gallego-posada/constrained_sparsity`

In this work*, we focus on the task of learning models with *controlled* levels of sparsity while performing in-training pruning. We tackle two central issues of the popular penalized method of Louizos et al. [LWK18]: ① tuning the L_0-penalty coefficient to achieve a desired sparsity level is non trivial and can involve computationally wasteful trial-and-error attempts; ② in the worst case the penalized method can outright fail at producing any sparsity, as documented by Gale et al. [GEH19].

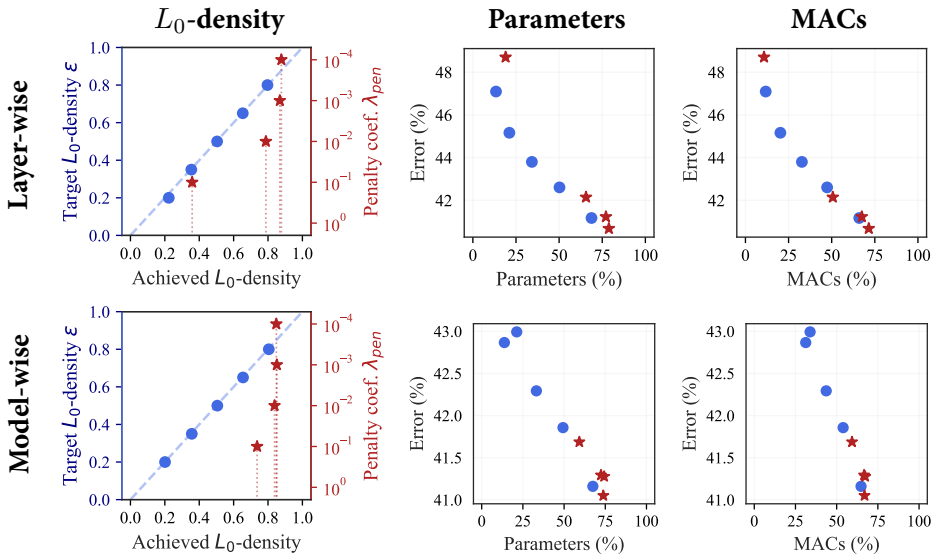


Figure 4.1: Training sparse ResNet18 models on TinyImageNet [LKJ17]. **The penalty-based method (red) shows a stagnating-then-overshooting behavior, making it difficult to tune. In contrast, our proposed constrained approach (blue) reliably achieves the desired target L_0-densities.** Density denotes the proportion of active gates in the model. The diagonal denotes the ideal case in which the achieved density exactly matches the target density used in the constrained setting. Parameters and MACs are computed for the corresponding test-time *purged* networks following the procedure described in Appx. A.4; the L_0-density (see Eq. (4.3)) is computed for the train-time model.

To address these limitations, we propose a constrained optimization approach in which ***arbitrary sparsity targets are expressed as constraints*** on the L_0-norm of the parameters.

Formally, we consider constraints of the type $\|\boldsymbol{\theta}_g\|_0 \leq K$, where $\boldsymbol{\theta}_g$ represents a group g of parameters of the network (e.g. individual layers, or the whole model), and resort to well established gradient-based methods for optimizing the Lagrangian associated with the constrained optimization problem.

Adopting this constrained formulation provides several advantages:

- Unlike the multiplicative factor λ of a penalty term, the constraint level ϵ_g has straight-forward and **interpretable semantics** associated with the *density* of a block of parameters $\boldsymbol{\theta}_g$, i.e. the percentage of active parameters.
- Requiring different density levels for different parameter groups (e.g. lower density for network modules with a larger computational or memory footprint), simply amounts to specifying several constraints with levels matching these desired densities, thus **avoiding the costly process of trial-and-error tuning**[†] and rebalancing various penalty factors.
- Much like the penalized approach in which additional regularizers can be "stacked" as other additive terms in the objective, new desired properties can be expressed in the constrained formulation in a **modular and extensible** fashion as additional constraints.
- In non-convex problems, the constrained formulation **can be strictly more powerful** than the penalized approach: there may be constraint levels that *cannot be achieved by any value of the penalty coefficient* [BV04, §4.7.4].

[†] *Appx. A.5 shows that the tuning challenges of penalized methods exist even for simple MLP tasks.*

The left column of Fig. 4.1 illustrates the interpretability and controllability advantages of the constrained approach when training a sparse ResNet18 model on TinyImageNet. We vary the constraint level (left axis) and the penalty coefficient (right axis) and compare the achieved parameter density at the end of training. Note how the penalized approach results in an very dense model ($> 80\%$) across several orders of magnitude of the penalty factor, and then suddenly drops to below 40% density. This behavior is in stark contrast with our proposed constrained approach, which *consistently achieves the desired target density*, across a wide range of values. See Section 4.5 for further discussion.

The purpose of our paper is to illustrate the feasibility and advantages of using constrained formulations in the study of sparse learning. We favor Lagrangian, gradient-based methods for tackling the constrained optimization problem due to their ease of use and scalability in the context of machine learning models. Exploring alternative constrained optimization techniques is an interesting direction for future studies, but lies beyond the scope of our work.

The main contributions of this work are:

- Building on the work of Louizos et al. [LWK18], we propose a constrained approach for learning models with controllable levels of sparsity (Section 4.3).
- We introduce a *dual restart* heuristic to avoid the excessive regularization caused by the accumulation of constraint violations in gradient-based Lagrangian optimization (Section 4.3).
- Previous studies [LWK18; GEH19] have been unsuccessful at training sparse ResNets [ZK16] based on L_0 regularization without significantly damaging performance. We propose two simple adjustments to the implementation of Louizos et al. [LWK18], allowing us to overcome these challenges (Appx. A.8).

- We provide empirical evidence that we can reliably achieve controllable sparsity across many different architectures and datasets. Moreover, the controlability and interpretability benefits of the constrained approach do not come at the expense of achieving competitive predictive performance (Section 4.5).

4.2 SPARSITY VIA L_0 PENALTIES

LOUIZOS et al. [LWK18] propose a framework for learning sparse models using the L_0-"norm" of the model parameters as an additive penalty to the usual training objective. The L_0-norm counts the *number* of non-zero entries in the parameter vector, and ignores the magnitude of said entries. Consider $h(x;\ \boldsymbol{\theta})$ be a predictor with parameters $\boldsymbol{\theta}$ and a supervised learning problem defined by a dataset of N i.i.d. pairs $\mathcal{D} = \{(x_i,\ y_i)\}_{i=1}^N$, a loss function ℓ and a regularization coefficient $\lambda_{\text{pen}} \geq 0$. LOUIZOS et al. [LWK18] formulate the L_0-regularized empirical risk objective:

$$\mathcal{R}(\boldsymbol{\theta}, \lambda_{\text{pen}}) = \mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}) + \lambda_{\text{pen}} \|\boldsymbol{\theta}\|_0 \tag{4.1a}$$

$$= \frac{1}{N} \left(\sum_{i=1}^{N} \ell\left(h(x_i;\ \boldsymbol{\theta}),\ y_i\right) \right) + \lambda_{\text{pen}} \sum_{j=1}^{|\boldsymbol{\theta}|} \mathbb{1}\{\theta_j \neq 0\} \tag{4.1b}$$

The non-differentiability of the L_0-norm makes it poorly suited for gradient-based optimization. The authors propose a reparametrization $\boldsymbol{\theta} = \tilde{\boldsymbol{\theta}} \odot \boldsymbol{z}$, where $\tilde{\boldsymbol{\theta}}$ are free (signed) parameter magnitudes, and $\boldsymbol{z}$ are independent stochastic *gates* indicating whether a parameter is active*. The authors model the gates using a modified version of the concrete distribution [MMT17; JGP17], with parameters denoted by $\boldsymbol{\phi}$.

* *The L_0-norm of $\boldsymbol{\theta}$ is determined by that of $\boldsymbol{z}$. This is because for commonly used weight initialization and optimization schemes, $\tilde{\boldsymbol{\theta}} \neq 0$ almost surely.*

This reparametrization allows for gradient-based optimization procedures, while retaining the possibility of achieving *exact* zeros in the parameters values. We provide a brief overview of the properties of the concrete distribution in Appx. A.1.

Moreover, this stochastic reparametrization induces a distribution over the network parameters $\boldsymbol{\theta}$. In consequence, the authors propose to re-define the training objective as the expectation (under the distribution of the gates) of the L_0-regularized empirical risk in Eq. (4.1):

$$\mathcal{R}(\tilde{\boldsymbol{\theta}}, \boldsymbol{\phi}, \lambda_{\text{pen}}) \triangleq \mathbb{E}_{\boldsymbol{z}\ |\ \boldsymbol{\phi}} \left[\mathcal{R}(\tilde{\boldsymbol{\theta}} \odot \boldsymbol{z}, \lambda_{\text{pen}}) \right] \tag{4.2a}$$

$$= \mathbb{E}_{\boldsymbol{z}\ |\ \boldsymbol{\phi}} \left[\mathcal{L}_{\mathcal{D}}(\tilde{\boldsymbol{\theta}} \odot \boldsymbol{z}) \right] + \lambda_{\text{pen}} \mathbb{E}_{\boldsymbol{z}\ |\ \boldsymbol{\phi}} \left[\|\boldsymbol{z}\|_0 \right] \tag{4.2b}$$

Test-time model. Since the stochastic reparametrization induces a distribution over models, LOUIZOS et al. [LWK18] propose a protocol to choose a sparse network at test time. We employ a slightly modified version of their strategy, based on the *medians* of the gates (see Appx. A.1.1).

Parameter grouping. Rather than considering a gate for each individual parameter (which would double the number of trainable parameters), several parameters may be gathered under a shared gate. We match the setup of LOUIZOS et al. [LWK18] who focus on *neuron sparsity*: using ① one gate per *input neuron* for fully connected layers; and

② one gate per *output feature map* for convolutional layers. This use of *structured sparsity* results in practical storage and computation improvements since entire parameter groups (e.g. slice of convolution kernels/activation) can be discarded.

Combining other norms. Louizos et al. [LWK18] show that their reparametrization can used in conjunction with other commonly used norms for regularization, such as the L_2-norm. One can express $\mathbb{E}_{\boldsymbol{z}|\boldsymbol{\phi}}\left[\|\hat{\boldsymbol{\theta}}\|_2^2\right] = \sum_{j=1}^{|\boldsymbol{\theta}|} \mathbb{P}[z_j \neq 0]\, \tilde{\theta}_j^2$, where $\hat{\boldsymbol{\theta}}$ is a *gate-rescaled* version of $\boldsymbol{\theta}$ in order to "avoid extra shrinkage for the gates". Further discussion on the challenges of combining weight-decay and their proposed reparametrization can be found in Appx. A.8 and A.9.

4.3 Sparsity via L_0 constraints

We favor formulating regularization goals as constraints, rather than as additive penalties with fixed scaling factors. We refer to these two approaches as *constrained* and *penalized*, respectively. Although a ubiquitous tool in machine learning, penalized formulations may come at the cost of hyper-parameter interpretability and are susceptible to intricate dynamics when incorporating multiple, potentially conflicting, sources of regularization.

4.3.1 *Constrained formulation*

In contrast to the penalized objective of Louizos et al. [LWK18] presented in Eq. (4.2), we propose to incorporate sparsity through constraints on the L_0-norm. We formulate an optimization problem that aims to minimize the model's expected empirical risk, subject to constraints on the expected L_0-norm of pre-determined parameter groups:

$$\min_{\tilde{\boldsymbol{\theta}},\boldsymbol{\phi}} \mathfrak{f}_{\text{obj}}(\tilde{\boldsymbol{\theta}}, \boldsymbol{\phi}) \triangleq \mathbb{E}_{\boldsymbol{z}|\boldsymbol{\phi}}\left[\mathcal{L}_{\mathcal{D}}(\tilde{\boldsymbol{\theta}} \odot \boldsymbol{z})\right] \tag{4.3a}$$

$$\text{s.t. } \mathfrak{g}_{\text{const}}(\boldsymbol{\phi}_g) \triangleq \frac{\overbrace{\mathbb{E}_{\boldsymbol{z}_g|\boldsymbol{\phi}_g}\left[\|\boldsymbol{z}_g\|_0\right]}^{L_0-\text{density}}}{\#(\tilde{\boldsymbol{\theta}}_g)} \leq \epsilon_g \;\; \text{for } g \in [1:G], \tag{4.3b}$$

where g denotes a subset of gates, $\#(\boldsymbol{x})$ counts the total number of entries in $\boldsymbol{x}$, and $\boldsymbol{x}_g$ denotes the entries of a vector $\boldsymbol{x}$ associated with the group g. See Appx. A.2 for details on parameter grouping.

Note how the $\#(\tilde{\boldsymbol{\theta}}_g)$ factor in the constraint levels allows us to **interpret** ϵ_g as the maximum *proportion* of gates that are allowed to be active within group g, in expectation. We refer to ϵ_g as the ***target density*** of group g. Lowering the target density demands a sparser model and thus a (not necessarily strictly) more challenging optimization problem in terms of the best *feasible* empirical risk. Moreover, for any choice of $\epsilon_g \geq 0$, the feasible set in Eq. (4.3) is always non-empty; while values of $\epsilon_g \geq 1$ result in vacuous constraints.

We highlight one important difference between the constrained and penalized formulations. The penalized approach is *jointly* optimizing the training loss $\mathfrak{f}_{\text{obj}}(\tilde{\boldsymbol{\theta}}, \boldsymbol{\phi})$ *and*

the expected L_0-norm $\mathfrak{g}_{\text{const}}(\phi)$, due to their additive combination (mediated by λ_{pen}). Meanwhile, the constrained method focuses on obtaining the best possible model within a prescribed density level ϵ: given two *feasible* solutions, the constrained formulation in Eq. (4.3) only discriminates based on the training loss. In other words, **we aim to *satisfy* the constraints, not to *optimize* them**.

4.3.2 *Solving the constrained optimization problem*

We start by considering the (nonconvex-concave) Lagrangian associated with the constrained formulation in Eq. (4.3), along with the corresponding min-max game:

$$\tilde{\boldsymbol{\theta}}^*, \boldsymbol{\phi}^*, \boldsymbol{\lambda}_{\text{co}}^* \triangleq \underset{\tilde{\boldsymbol{\theta}}, \boldsymbol{\phi}}{\text{argmin}} \underset{\boldsymbol{\lambda}_{\text{co}} \geq 0}{\text{argmax}} \; \mathfrak{L}(\tilde{\boldsymbol{\theta}}, \boldsymbol{\phi}, \boldsymbol{\lambda}_{\text{co}}) \tag{4.4a}$$

$$\triangleq \mathfrak{f}_{\text{obj}}(\tilde{\boldsymbol{\theta}}, \boldsymbol{\phi}) + \sum_{g=1}^{G} \lambda_{\text{co}}^g \left(\mathfrak{g}_{\text{const}}(\boldsymbol{\phi}_g) - \epsilon_g \right), \tag{4.4b}$$

where $\boldsymbol{\lambda}_{\text{co}} = [\lambda_{\text{co}}^g]_{g=1}^G$ are the *dual variables* corresponding to the (non-negative) Lagrange multipliers associated with each constraint.

A commonly used approach to optimize this Lagrangian is *simultaneous* gradient descent on $(\tilde{\boldsymbol{\theta}}, \boldsymbol{\phi})$ and projected (to $\mathbb{R}^+$) gradient ascent on $\boldsymbol{\lambda}_{\text{co}}$ [LJJ20]:

$$[\tilde{\boldsymbol{\theta}}^{t+1}, \boldsymbol{\phi}^{t+1}] \triangleq [\tilde{\boldsymbol{\theta}}^t, \boldsymbol{\phi}^t] - \eta_{\text{primal}} \nabla_{[\tilde{\boldsymbol{\theta}}, \boldsymbol{\phi}]} \mathfrak{L}(\tilde{\boldsymbol{\theta}}^t, \boldsymbol{\phi}^t, \boldsymbol{\lambda}_{\text{co}}^t) \tag{4.5a}$$

$$\hat{\boldsymbol{\lambda}}^{t+1} \triangleq \boldsymbol{\lambda}_{\text{co}}^t + \eta_{\text{dual}} \nabla_{\boldsymbol{\lambda}_{\text{co}}} \mathfrak{L}(\tilde{\boldsymbol{\theta}}^t, \boldsymbol{\phi}^t, \boldsymbol{\lambda}_{\text{co}}^t) \tag{4.5b}$$

$$= \boldsymbol{\lambda}_{\text{co}}^t + \eta_{\text{dual}} \left[\mathfrak{g}_{\text{const}}(\boldsymbol{\phi}_g^t) - \epsilon_g \right]_{g=1}^G$$

$$\boldsymbol{\lambda}_{\text{co}}^{t+1} \triangleq \max\left(0, \hat{\boldsymbol{\lambda}}^{t+1}\right) \tag{4.5c}$$

The gradient update for $\boldsymbol{\lambda}_{\text{co}}$ matches the value of the violation of each constraint. When a constraint is satisfied, the gradient for its corresponding Lagrange multiplier is non-positive, leading to a reduction in the value of the multiplier.

Negligible computational overhead. Just as the penalized formulation of Louizos et al. [LWK18], the update for $\tilde{\boldsymbol{\theta}}$ and $\boldsymbol{\phi}$ requires the gradient of the training loss and that of the expected L_0-norm. Hence, the cost of executing this update scheme is the same as the cost of a gradient descent update on the penalized formulation in Eq. (4.1), up to the negligible cost of updating the multipliers.

Choice of optimizers. We present simple gradient descent-ascent (GDA) updates in Eq. (4.5). However, our proposed framework is compatible with different choices for the primal and dual optimizers, including stochastic methods. Throughout our experiments, we opt for primal (model) optimizers which match standard choices for the different architectures. A choice of gradient ascent for the dual optimizer provided consistently robust optimization dynamics across all tasks. Detailed experimental configurations are provided in Appx. A.10. The evaluation and design of other optimizers, especially those for updating the Lagrange multipliers, is an interesting direction for future research.

Oscillations. The non-convexity of the optimization problem in Eq. (4.4) implies that a saddle point (pure strategy Nash Equilibrium) might not exist. In general, this can lead to oscillations and unstable optimization dynamics. Appx. A.6 provides pointers to more sophisticated constrained optimization algorithms which achieve better convergence guarantees on nonconvex-concave problems than GDA. Fortunately, throughout our experiments we observed oscillatory behavior that quickly settled around feasible solutions. Empirical evidence of this claim is presented in Section 4.5.4.

Extensibility. Our proposed constrained formulation is "modular" in the sense that it is easy to induced other properties in the model's behavior beside sparsity (e.g. fairness [Hoo+19; Cot+19b]) by prescribing them as *additional constraints*; much like extra additive terms in the penalized formulation. However, the improved interpretability and control afforded by the constrained approach removes the need to perform extensive tuning of the hyper-parameters to balance these potentially competing demands.

4.3.3 *Dual restarts*

A drawback of gradient-based updates for optimizing the Lagrangian in Eq. (4.4) is that the constraint violations accumulate in the value of the Lagrange multipliers throughout the optimization, and continue to affect the optimization dynamics, *even after a constraint has been satisfied*. This results in an excessive regularization effect, which forces the primal parameters towards the *interior* of the feasible set. This behavior can be detrimental if we are concerned about minimizing the objective function and *satisfying* (but not minimizing!) the constraints.

To address this, we propose a ***dual restart scheme*** in which the Lagrange multiplier λ_{co}^g associated with a constraint $\mathfrak{g}_{\text{const}}(\phi_g) \leq \epsilon_g$ is set to 0 whenever the constraint is satisfied; rather than waiting for the "negative" gradient updates ($\mathfrak{g}_{\text{const}}(\phi_g) - \epsilon_g < 0$ when feasible) to reduce its value. Formally,

$$\left[\boldsymbol{\lambda}_{\text{co}}^{t+1}\right]_g \triangleq \begin{cases} \max\left(0,\ \left[\boldsymbol{\lambda}_{\text{co}}^{t}\right]_g + \eta_{\text{dual}}\ \left(\mathfrak{g}_{\text{const}}(\boldsymbol{\phi}_g^t) - \epsilon_g\right)\right), & \text{if} \quad \mathfrak{g}_{\text{const}}(\boldsymbol{\phi}_g^t) > \epsilon_g \\ 0, \quad \text{otherwise} \end{cases} \tag{4.6}$$

Dual restarts remove the contribution of the expected L_0-norm to the Lagrangian for groups g whose constraints are satisfied, so that the optimization may focus on improving the predictive performance of the model. In fact, this dual restart strategy can be theoretically characterized as a *best response* (in the game-theoretic sense) by the dual player. The effect of dual restarts in the optimization dynamics is illustrated in Section 4.5.3 and Appx. A.7.

4.4 Related Work

Min-max optimization. Commonly used methods for solving constrained convex optimization problems [BV04; FW56; Jag13] make assumptions on the properties of the objective function, constraints or feasible set. In this work, we focus on applications involving neural networks, leading to the violation of such assumptions. We rely on a

GDA-like updates for optimizing the associated Lagrangian. However, our proposed formulation can be readily integrated with more sophisticated/theoretically supported algorithms for constrained optimization of non-convex-concave objectives, such as the extragradient method [Kor76]. Further discussion on guarantees and alternative algorithms for min-max optimization is provided in Appx. A.6.

Model sparsity. Learning sparse models is a rich research area in machine learning. There exist many different approaches for obtaining sparse models. Magnitude-based methods [TF95; HMD16] perform one or more rounds of pruning, by removing the parameters with the lowest magnitudes. Popular non-magnitude based techniques include [LDS90; GYC16; MAV17]. Structured pruning methods [Dai+18; Nek+17; LWK18; Li+17], remove entire neurons/channels rather than *individual* parameters. More recently, the Lottery Ticket Hypothesis [FC19] has sparked interest in techniques that provide the storage and computation benefits of sparse models *directly during training* [MW19; Evc+20]. However, finding "good" sparse sub-networks at initialization remains a central challenge for these techniques [Fra+20; Mal+20].

Controllable sparsity. Magnitude pruning [HMD16; Li+17] can achieve arbitrary levels of sparsity "by design" since it removes *exactly* the proportion of parameters with lowest magnitudes in order to match the desired density. However, the magnitude pruning method experiences certain shortcomings: ① retaining performance usually involves several round of fine-tuning*; ② it relies on the *assumption* that magnitude (of filters or activations) is a reasonable surrogate for parameter importance; and ③ it lacks the "extensibility" property of our constrained formulation: it is not immediately evident how to induce other desired properties in the model, besides sparsity.

* *This re-training overhead makes magnitude pruning less appealing compared with in-training alternatives, since magnitude pruning is typically performed given an already fully trained model.*

Note that several extensions of the basic magnitude pruning method have been proposed. Zhu and Gupta [ZG17] start from a partially or fully pre-trained model and consider a sparsification scheme in which the network density is gradually reduced, while fine-tuning the model to compensate for any potential loss in performance due to pruning. Wang et al. [Wan+21] start by *identifying* the parameters to be removed by applying magnitude pruning on a pre-trained model. However, rather than pruning the model immediately, the authors propose to fine-tune the model with an adaptive L_2-penalty. The weight of this penalty is increased over time for the previously identified parameters, leading their magnitudes to decrease during the fine-tuning process.

Sparsity via constrained optimization. Previous works have cast the task of learning sparse models as the solution of a constrained optimization problem. Carreira-Perpinan and Idelbayev [CI18] consider a reformulation of the constrained optimization problem using "auxiliary variables", and assume that the constraints enjoy an efficient proximal operator. Their empirical evaluation is limited to low-scale models and datasets.

Zhou et al. [Zho+21] adopt a constrained formulation similar to ours, although based on a different reparametrization of the gates. The authors tackle the constrained problem via projected gradient descent by cleverly exploiting the existence of an efficient projection of the gate parameters onto their feasible set. However, the applicability of their method is limited to constraints with an efficiently-computable projection operator.

Lemaire et al. [LAJ19] consider "budget-aware regularization" and tackle the constrained problem using a barrier method. Although originally inspired by a constrained approach, their resulting training objective corresponds to a penalized method with a penalty factor that requires tuning, in addition to the choice of barrier function.

Other constrained formulations in ML. Constrained formulations can be used to prescribe desired behaviors or properties in machine learning models. Nandwani et al. [Nan+19] study the problem of training deep models under constraints on the network's predicted labels, and approach the constrained problem in practice through a min-max Lagrangian formulation. Incorporating these constraints during training allows them to inject domain-specific knowledge into their models across several tasks in natural language processing.

Fioretto et al. [Fio+20] consider a wide range of applications spanning from optimal power flow in energy grids, to the training of fair classification models. Their work demonstrates how Lagrangian-based methods can be complementary to deep learning by effectively enforcing complex physical and engineering constraints.

Cotter et al. [Cot+19b] train models under constraints on the prediction rates of the model over different datasets. Note that the sparsity constraints we study in this paper depends only on properties of the *parameters* and not on the *predictions* of the model. We would like to highlight that the notion of *proxy constraints* introduced by Cotter et al. [Cot+19b] can enable training models based on constraints on their actual test time density, rather than the surrogate expected L_0-norm metric.

4.5 Experiments

The main goal of our work is to train models that attain good predictive performance, while having a fine-grained command on the sparsity of the resulting model. In this section we present a comparison with the work of Louizos et al. [LWK18]*; we explore the stability and controllability properties of our Lagrangian-based constrained approach, along with the effect of our proposed dual restarts heuristic. Finally, we present empirical evidence which demonstrates that our method successfully retains its interpretability and controllability advantages when applied to large-scale models and datasets.

** See a comparison to other sparsity methods therein, along with the survey of Gale et al. [GEH19].*

4.5.1 *Experimental setup*

Experiment configuration and hyperparameters. Details on our implementation, hyperparameter choices and information on the network architectures can be found in Appx. A.1, A.2, A.3, A.4 and A.10.

Model- and layer-wise settings. We present experiments using two kinds of constraints: one *global* constraint on the proportion of active gates throughout the entire model; or several *local* constraints prescribing a maximum density at each layer. Note that for models such as ResNet50, the layer-wise setting involves handling 48 constraints. The experiments below demonstrate that **our constrained approach can gracefully handle from a single constraint up to dozens of constraints** in a unified way and still achieve controllable sparsity for *each* of the layers/model. This level of control is an

intractable goal for penalized methods: as demonstrated in Appx. A.5, even trying to tame *one* constraint via a penalty factor can be prohibitively expensive.

L_0-regularization for residual models. ResNets have been a challenging setting for L_0-penalty based methods. GALE et al. [GEH19] trained WideResNets [ZK16] and ResNet50 [He+16] using the penalized L_0-regularization framework of LOUIZOS et al. [LWK18], and reported being unable to produce sparse ResNets without significantly degrading performance.

We propose two simple adjustments that enable us to successfully train WRNs and ResNets with controllable sparsity, while retaining competitive performance: ① increasing the learning rate of the stochastic gates; and ② removing the gradient contribution of the weight decay penalty towards the gates. Appx. A.8 and A.9 provide detailed analysis and empirical validation of these two modifications. We integrate these adjustments in all experiments involving residual models below.

Obtaining test-time models. Appx. A.4 describes our procedure to transform a model with stochastic gates into a deterministic, test-time model. The measurements of retained parameters and MACs (multiply-accumulate operations) percentages reported in the tables and figures below, are computed for the deterministic, purged, test-time models.

4.5.2 *Proof-of-concept experiments on MNIST*

We begin by comparing the behavior of our method with that of LOUIZOS et al. [LWK18] in the simple setting of training MLP and LeNet5 architectures on the MNIST dataset. The authors report the size of their pruned architectures found using the penalized formulation. In this section we aim to showcase the *controllability* advantages of our constrained approach. We manually computed the corresponding model-wise or layer-wise density levels achieved by the reported architectures of LOUIZOS et al. [LWK18] and used these values as the target density levels for our constrained formulation.

Table 4.1: Achieved density levels and performance for sparse MLP and LeNet5 models trained on MNIST for 200 epochs. Metrics aggregated over 5 runs. †Results presented by LOUIZOS et al. [LWK18] with N representing the training set size (see Appx. A.3).

Architecture	**Grouping**	**Method**	**Hyper-parameters**	**Pruned architecture**	**Val. Error** (%)	
					best	at 200 epochs (avg ± 95% CI)
MLP 784-300-100	Model	Pen.	†$\lambda_{pen} = 0.1/N$	219-214-100	1.4	–
		Const.	$\epsilon = 33\%$	198-233-100	1.36	1.77 ± 0.08
	Layer	Pen.	†$\lambda_{pen} = [0.1, 0.1, 0.1]/N$	266-88-33	1.8	–
		Const.	$\epsilon = [30\%, 30\%, 30\%]$	243-89-29	1.58	2.19 ± 0.12
LeNet5 20-50-800-500	Model	Pen.	†$\lambda_{pen} = 0.1/N$	20-25-45-462	0.9	–
		Const.	$\epsilon = 10\%$	20-21-34-407	0.56	1.01 ± 0.05
	Layer	Pen.	†$\lambda_{pen} = [10, 0.5, 0.1, 0.1]/N$	9-18-65-25	1.0	–
		Const.	$\epsilon = [50\%, 30\%, 70\%, 10\%]$	10-14-224-29	0.7	0.91 ± 0.05

Table 4.1 displays the results of our constrained method and the reported metrics for the penalized approach. Note that, as desired, the pruned models obtained using the constrained formulation resemble closely the "target architecture sizes" reported by Louizos et al. [LWK18]. Moreover, our method does not cause any loss in performance with respect to the penalized approach. This final observation will be confirmed for larger-scale tasks in later sections.

Note that the goal of this section is to demonstrate that our constrained approach can achieve arbitrary sparsity targets "in one shot" (i.e. without trial-and-error tuning) and without inducing any compromise in the predictive performance of the resulting models. Comprehensive experiments for MLP and LeNet5 models on MNIST across a wide range of sparsity levels for model- and layer-wise constraints are presented in Appx. A.11.1.

4.5.3 *Training dynamics and dual restarts*

We now discuss the effect of the dual restarts scheme introduced in Section 4.3.3 on the training dynamics of our constrained formulation. Fig. 4.2 illustrates the training of a convolutional network on MNIST under a 30% model-wise density constraint when using (blue) or not (orange) dual restarts.

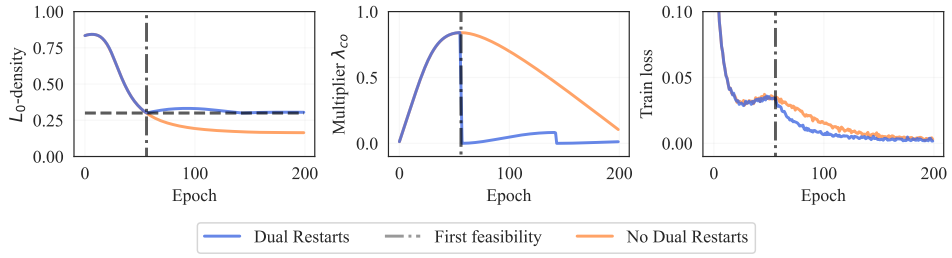


Figure 4.2: Effect of dual restarts for training a LeNet5 on MNIST with a model-wise target density $\epsilon_g = 30\%$ (horizontal dashed line). The accumulation of the constraint violations in the Lagrange multiplier leads to excessive sparsification when the model satisfies the constraint. Restarting the Lagrange multiplier allows the model to concentrate on improving the training loss.

We initialize the Lagrange multipliers to zero. Therefore, at the beginning of optimization there is no contribution from the L_0-norm in the Lagrangian (see Eq. (4.4)), and the optimization focuses on improving the training loss. As the optimization progresses, the constraint violations are accumulated in the value of the Lagrange multiplier. When the Lagrange multiplier is sufficiently large, the importance of satisfying the constraints outweighs that of optimizing the training loss.[†] In consequence, the model density decreases. As the model reaches the desired sparsity level, λ_{co} stops increasing.

Up until the time at which the model is first feasible, the multiplier value accumulates the constraint violations (scaled by the dual learning rate). Once the model is feasible, the constraint violation $\mathfrak{g}_{\text{const}}(\phi_g) - \epsilon < 0$ becomes negative, leading to a *decrease* in the Lagrange multiplier. However, at this stage, the Lagrange multiplier is large due to the accumulated constraint violations. This confers a higher relative importance to the gradient of the constraints over that of the training loss: the larger multiplier encourages to reduce the *constraints even if they are already being satisfied*.

[†] *Note that the Lagrange multiplier influences the update of the model parameters by dynamically adjusting the relative importance of the gradient of the training loss with respect to the gradient of the constraint. In the penalized method this relative importance is fixed.*

Our proposed dual restart heuristic reduces the Lagrange multiplier to zero whenever the constrained is satisfied, allowing the training to focus on minimizing the training loss faster. Although this heuristic may lead to slightly unfeasible solutions, as demonstrated throughout our experiments, our models remain consistently below (or close to) the required L_0-density levels.

4.5.4 *Stable constraint dynamics*

Despite the theoretical risk of oscillatory dynamics commonly associated with iterative constrained optimization methods, we consistently observed quickly stabilizing behavior in our experiments. Fig. 4.3 shows the density levels throughout training for a layer of a WideResNet-28-10 trained on CIFAR-10 (right), and the model-wise density of a ResNet18 trained on TinyImageNet (left).

The desired density levels are successfully achieved over a wide range of targets, and the constraint dynamics stabilize quickly. These dynamics were consistent across all our architectures and datasets.

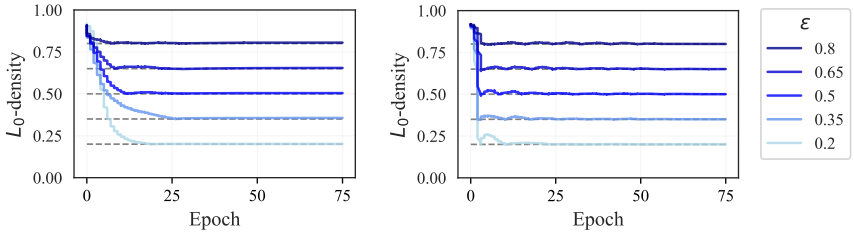


Figure 4.3: Density levels for a ResNet18 model (left; trained with a model-wise constraint) and the last sparsifiable layer of a WideResNet-28-10 model (right; trained with layer-wise constraints).

4.5.5 *Large-scale experiments*

We now demonstrate the scalability of our method to more challenging settings: we consider (Wide)ResNet models on the CIFAR-{10, 100}, TinyImageNet [LKJ17] and ImageNet [Den+09] datasets. Comprehensive experiment are provided in Appx. A.11.

CIFAR-{10, 100} and TinyImageNet. Figures 4.1 and 4.4 display the results for a ResNet18 model trained on Imagenet, and a WideResNet-28-10 trained on CIFAR-10, respectively. The left column shows the alignment between the achieved and desired densities (as expected proportion of *active gates* in the model). Our method (in blue) provides a robust control over the range of densities. In contrast, the penalized method (in red) exhibits an unreliable dependency between the penalty coefficient and the achieved density: when increasing the coefficient λ_{pen}, the achieved density seems to be insensitive to λ_{pen} for several orders of magnitude until it starts considerably changing. This brittle sensitivity profile limits the potential of the penalized method for achieving controlled sparsification.

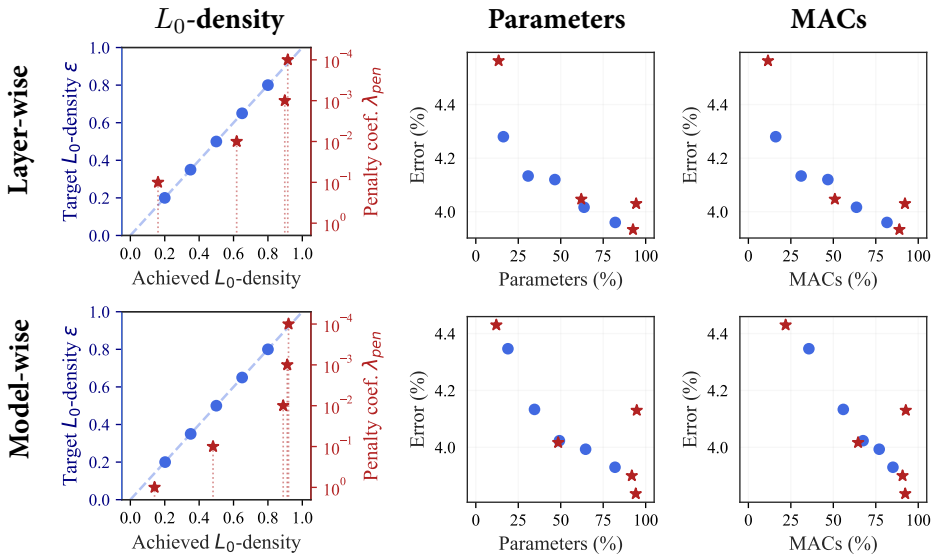


Figure 4.4: Training sparse WideResNet-28-10 models on CIFAR-10.

Columns two and three display the number of parameters and MACs (multiply-accumulate operations) of the resulting purged models, as a proportion of those of the fully-dense baseline model. Note that, while retaining a similar proportion of parameters, layer-wise constraints lead to a larger reduction in the number of MACs, compared to the model-wise case. This is because layer-wise constraints induce a strict, homogeneous sparsification of all the modules of the network; while the model-wise setting can allow for a more flexible allocation of the parameter budget across different layers.

ImageNet. We conducted experiments on ImageNet [Den+09] with a ResNet50 architecture. We compare with layer-wise structured magnitude pruning [Li+17][‡]. The results are presented in Table 4.2.

[‡] *For each layer, we remove the filters with the* $1 - \epsilon$ *lowest* L_1*-norms to achieved the desired* ϵ *density.*

Note that experiments with layer-wise constraints correspond to optimization problems with 48 constraints (one for each sparsifiable layer). We highlight the number of constraints since tuning such a large number of penalty coefficients is an intractable challenge when using the penalized method.

Just like the magnitude pruning method, our proposed approach successfully delivers the desired levels of sparsity in this challenging task. To the best of our knowledge, our work constitutes the first instance of successfully learning ResNet50 models using the L_0 reparametrization of Louizos et al. [LWK18] for structured sparsity while retaining high accuracy.

Our results clearly demonstrate that the constrained L_0 formulations can obtain large levels of structured parameter reduction while preserving performance. Table 4.2 shows a quick degradation in performance for the magnitude pruning method, and highlights the need for fine-tuning in heuristic-based pruning techniques.

Table 4.2: ResNet50 models on ImageNet with structured sparsity. "Fine-tuning" for zero epochs means *no* fine-tuning.

Target Density	Method	L_0-density (%)	Params (%)	MACs (%)	Best Val. Error (%) After fine-tuning for # epochs			
					0	1	10	20
–	Pre-trained Baseline	100	[25.5M]	[$4.12 \cdot 10^9$]	23.90	———	———	———
$\epsilon = 90\,\%$	Const. *Model-wise*	90.36	88.06	91.62	**24.68**	———	———	———
	Const. *Layer-wise*	90.58	87.07	85.97	24.97	———	———	———
	L1-MP *Layer-wise*	–	85.94	84.99	38.74	25.38	24.69	**24.68**
$\epsilon = 70\,\%$	Const. *Model-wise*	70.78	64.41	76.50	**25.53**	———	———	———
	Const. *Layer-wise*	70.36	61.91	58.59	26.98	———	———	———
	L1-MP *Layer-wise*	–	62.15	59.85	97.78	29.04	26.80	26.14
$\epsilon = 50\,\%$	Const. *Model-wise*	50.18	42.47	58.00	**27.51**	———	———	———
	Const. *Layer-wise*	50.70	43.15	38.25	27.89	———	———	———
	L1-MP *Layer-wise*	–	43.47	39.76	99.75	36.21	29.98	29.16
$\epsilon = 30\,\%$	Const. *Model-wise*	30.31	31.81	42.05	**29.65**	———	———	———
	Const. *Layer-wise*	31.44	30.16	23.74	31.71	———	———	———
	L1-MP *Layer-wise*	–	29.86	24.80	99.89	56.11	36.90	34.74

4.5.6 *Unstructured sparsity*

Appx. A.12 contains experiments with unstructured sparsity (i.e. one gate per parameter, rather than per neuron/activation map) for the MNIST and TinyImageNet datasets. These experiments show that the controllability advantages of our constrained formulation apply in the unstructured regime. Recall that Gale et al. [GEH19] report an apparent dichotomy between sparsity and performance when training (residual) models with unstructured sparsity using the L_0 reparametrization of Louizos et al. [LWK18]. Our experimental results demonstrate that it is in fact possible to achieve high levels of sparsity *and* predictive performance.

4.6 Conclusion

We resort to a constrained optimization approach as a tool to overcome the controllability shortcomings faced by penalty-based sparsity methods. Along with a reliable control of the model density, this technique provides a more interpretable hyper-parameter and removes the need for expensive iterative tuning. We adopt the L_0 reparametrization framework of Louizos et al. [LWK18] and integrate simple adjustments to remedy their challenges at training (Wide)ResNet models. Our proposed method succeeds at achieving the desired sparsity with no compromise on the model's performance for a broad range of architectures and datasets. These observations position the constrained approach as a solid, practical alternative to popular penalty-based methods in modern machine learning tasks.

Prologue to the Second Contribution

5

Article Details

Juan Ramirez and Jose Gallego-Posada. **L_0onie: Compressing COINs with L_0-constraints**. This paper was presented at the *Sparsity in Neural Networks Workshop*, 2022.

Author Contributions

Jose Gallego-Posada proposed the original idea and led the writing of the paper. Juan Ramirez and Jose Gallego-Posada contributed equally to the implementation and execution of the experiments.

Context

This paper extends on the ideas presented in Chapter 4 by applying the constrained optimization approach to a more complex task. In this case the sparsity of the model does not come from an L_0 constraint, but rather from a constraint on the bits-per-pixel (BPP) of the test-time model. Unlike the constraints considered in Chapter 4, the BPP constraint is non-differentiable and requires a (slightly) different algorithmic approach. Our experiments constitute a successful application of the proxy-constraint idea by Cotter et al. [Cot+19b].

Although we demonstrated the superior performance of our method compared to the basic COIN [Dup+21] approach and a magnitude-pruning baseline, our contribution did not tackle an important bottleneck in the COIN compression pipeline: the need for training a separate model for each datum. We believe this significant limitation has been nicely addressed by the work of Schwarz and Teh [ST22], in which the authors propose a meta-learning approach with sparse modulations [Per+17] learned per-datum.

6 L_0onie: Compressing coins with L_0-constraints

ABSTRACT

Advances in Implicit Neural Representations (INR) have motivated research on domain-agnostic compression techniques. These methods train a neural network to approximate an object, and then store the weights of the trained model. For example, given an image, a network is trained to learn the mapping from pixel locations to RGB values. In this paper, we propose L_0onie, a sparsity-constrained extension of the COIN compression method. Sparsity allows to leverage the faster learning of overparametrized networks, while retaining the desirable compression rate of smaller models. Moreover, our constrained formulation ensures that the final model respects a pre-determined compression rate, dispensing of the need for expensive architecture search.

6.1 INTRODUCTION

Implicit Neural Representations (INRs) train neural networks mapping coordinates (e.g. pixel locations) to features (e.g. RGB values) in order to approximate a given object. INRs have been applied to a wide range of data modalities including audio [Sit+20], images [Sta07], video [Li+21], 3D scenes [Mes+19] and temperature fields [DTD22]. INRs provide a new perspective on data compression: rather than dealing with the "raw" features of the object, train an INR to approximate the object and store the parameters of the learned model. COIN [Dup+21] pioneered this approach for image compression.

INR-based compression is a nascent technology. COIN exhibits sub-par performance and more intensive computational cost compared to domain-specific codecs. Moreover, it commonly requires architecture search to balance the relationship between the model capacity and reconstruction quality. Although these techniques are not yet competitive with established codecs for well-studied domains like audio or image compression, the greater promise of methods like COIN lies on their ability to provide compression standards that can be applicable to virtually any data modality.

Recent works [Dup+22; Lee+21; ST22] focus on improving COIN via a meta-learning approach in which a base network is (pre-)trained over a large collection of datapoints, so that instance-dependent INRs can be found quickly. The task of encoding of individual instances is casted as a search for *modulations* [Per+17] on the base architecture.

In this work we concentrate on the interplay between the model size, its representational capacity and the required training time. In L_0onie*, we combine "overparametrized"

* *Loonie is a colloquial name for the Canadian one dollar coin.*

COIN models with the sparse L_0-reparametrization of LOUIZOS et al. [LWK18]. This allows us to exploit the power of larger models to achieve better reconstructions, faster; without having to commit to their undesirable compression rate. We take advantage of the inherent redundancies in these larger models, and sparsify them during training in order to achieve a pre-specified compression rate. Our constrained formulation provides direct control over the resulting compression rate and removes the need for costly hyper-parameter tuning or architecture search. Fig. 6.1 provides an overview of the compression behavior of of L_0onie, COIN and JPEG at different BPP budgets.

Our code is available at: `https://github.com/juan43ramirez/l0onie`.

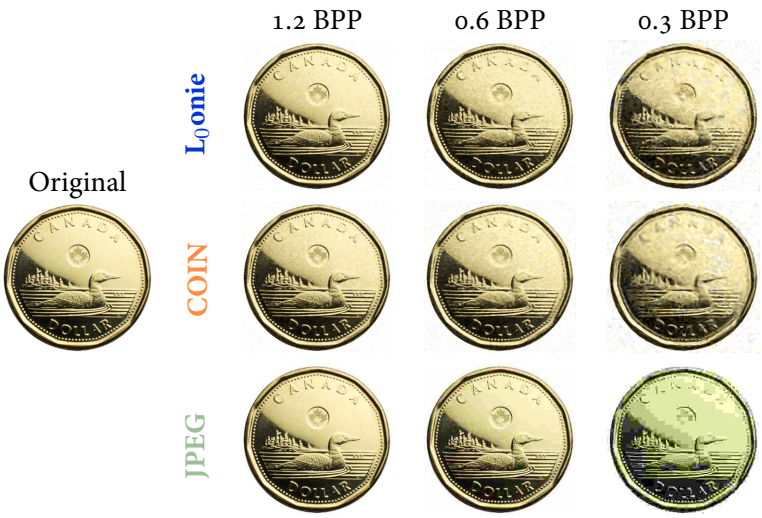


Figure 6.1: Qualitative comparison between L_0onie, COIN and JPEG on a picture of a loonie.

6.2 IMPLICIT NEURAL REPRESENTATIONS

** Note that a fixed choice of model architecture and activation functions might not be suitable for all data modalities.*

For concision, we present a description of INRs using terminology from image processing. However, we highlight that INRs are in principle agnostic* to the "data type" at hand. Consider the image $\mathbf{I}$ to be compressed as a collection of pixel coordinates $\boldsymbol{x}_p \in [-1, 1]^2$ and corresponding RGB pixel intensities $\boldsymbol{y}_p \in [0, 1]^3$, indexed over a discrete set of pixels $\mathcal{P}$.

Consider a family of neural networks $f_{\boldsymbol{\theta}} : \mathbb{R}^2 \to \mathbb{R}^3$ with parameters $\boldsymbol{\theta}$. An implicit neural representation (INR) of image $\mathbf{I}$ is a solution to the following supervised learning problem:

$$\min_{\boldsymbol{\theta}} \mathcal{L}_{\mathbf{I}}(\boldsymbol{\theta}) \triangleq \sum_{p \in \mathcal{P}} ||f_{\boldsymbol{\theta}}(\boldsymbol{x}_p) - \boldsymbol{y}_p||_2^2. \tag{6.1}$$

Upon solving the optimization problem, the original image can be (approximately) reconstructed by *evaluating* the function $f_{\boldsymbol{\theta}}$ at the pixel locations $\mathcal{P}$. The choice of a family of models can have a large influence in the reconstruction performance, and is an active area of research. Recent proposals combine fully-connected models with positional embeddings [Tan+20] or sine activation functions [Sit+20].

Dupont et al. [Dup+21] propose COIN as a creative approach for image compression: when compressing an image $\mathbf{I}$, rather than storing RGB values, store the *weights* of an INR of $\mathbf{I}$. We follow COIN's choice of MLPs with sinusoidal activation functions in this work.

The distortion-rate tradeoff in COIN involves balancing the higher expressive capacity of large models with the detrimental effect of more model parameters towards the compression rate. Achieving specific compression rates while retaining low distortion may involve time-consuming architecture search.

6.2.1 *Sparsifying INRs*

Given that compressing images using COIN already requires more time than standard methods like JPEG (by several orders of magnitude), dispensing of the need for architecture search is an important step towards a wider adoption of INR-based compression. Sparsity techniques are a natural approach to address this issue.

Magnitude pruning can be used to attain specific compression rates, but results in suboptimal performance even after fine-tuning (see Section 6.3). The assumptions made by heuristic-based sparsity methods popular in other machine learning tasks might not transfer well to the context of INRs.

For instance, magnitude pruning relies on the idea that low-norm parameters *should* have a small influence in the final prediction. This has been shown empirically for image classification tasks where the model predicts a label among a discrete number of classes [Li+17; GEH19]. We hypothesize that the sub-par performance of magnitude pruning in our experiments may be due to the use of sine activations in the model and the continuous nature of the targets.

Louizos et al. [LWK18] propose a framework for learning sparse models by means of a differentiable reparametrization of the model weights $\boldsymbol{\theta} = \tilde{\boldsymbol{\theta}} \odot \boldsymbol{z}$, where $\tilde{\boldsymbol{\theta}}$ are free (signed) parameter magnitudes, and $\boldsymbol{z}$ are stochastic gates indicating whether a parameter is active. The gates follow a *hard-concrete* distribution [LWK18] with parameters $\boldsymbol{\phi}$. The authors then augment the usual training objective with an additive penalty given by the expected L_0-norm of the gates $\boldsymbol{z}$, to encourage sparsity in the model.

We propose to combine the MLPs with sinusoidal activations of COIN with this L_0 reparametrization. This sparsity perspective opens the door for training "overparametrized" models which can achieve better performance, faster [ACH18]. Moreover, learning the sparsity pattern during training avoids having to commit to the undesirably low compression rate of the fully dense model.

The same reparametrization is used in the concurrent work of Schwarz and Teh [ST22], although stemming from a different motivation. While the authors provide valuable insights regarding the integration of sparsity within a meta-learning pipeline, they overlook the challenge of tuning the hyper-parameter λ_{pen}, which dictates the relative importance of the sparsity term. Adjusting λ_{pen} to achieve a *specific* compression rate can be as prohibitive as performing architecture search [BV04; Gal+22].

Note that the stochastic re-parametrization induces a distribution over the models. For practical reasons, it is convenient to have a single model at decoding time. We use the gate medians $\hat{\boldsymbol{z}}(\boldsymbol{\phi}) \in [0, 1]$ for constructing said model. Due to the "stretching" in the hard-concrete distribution, it is possible for the medians to be exactly 0 or 1 and not just fractional. See Appx. B.1 for more details.

6.2.2 L_0-*constrained formulation*

Gallego-Posada et al. [Gal+22] argue that constrained formulations can provide greater hyper-parameter interpretability and controllability when learning sparse neural networks, compared to the commonly used penalized approach. The authors consider constraints on the expected L_0-norm of the parameters using the reparametrization of Louizos et al. [LWK18]. We extend their constrained formulation by using *proxy-constraints* [Cot+19b] to directly control the compression rate of the resulting model (see Appx. B.2).

We consider the problem of finding a sparse INR for an image $\mathbf{I}$ given a constraint on the compression rate expressed in terms of a budget of $\tau_{\texttt{BPP}}$ bits-per-pixel. We observed degraded performance when training using Monte Carlo samples for the stochastic gates, as done in Louizos et al. [LWK18]. For this reason, we train a deterministic model using the gate medians. This setting coincides with the decoding time model described above.

Formally, we consider the following constrained optimization problem:

$$\min_{\tilde{\boldsymbol{\theta}}, \boldsymbol{\phi}} \mathcal{L}_{\mathbf{I}}(\tilde{\boldsymbol{\theta}} \odot \hat{\boldsymbol{z}}(\boldsymbol{\phi})) \text{ s.t. } \texttt{BPP}(\texttt{cast}(\tilde{\boldsymbol{\theta}} \odot \hat{\boldsymbol{z}}(\boldsymbol{\phi})) \leq \tau_{\texttt{BPP}}, \tag{6.2}$$

where $\texttt{BPP}(\texttt{cast}(\boldsymbol{u})) = \sum_i \mathbb{I}[\texttt{cast}(\boldsymbol{u}_i) \neq 0] \cdot \texttt{bits}(\texttt{cast}(\boldsymbol{u}_i))$, and $\texttt{cast}(\cdot)$ quantizes its input to a specified data type, such as `float16`.

In practice, we optimize the Lagrangian associated with the problem in Eq. (6.2). Let $\lambda_{\text{co}} \geq 0$ be the Lagrange multiplier corresponding to the constraint. The min-max Lagrangian problem is given by:

$$\tilde{\boldsymbol{\theta}}^*, \boldsymbol{\phi}^*, \lambda_{\text{co}}^* \triangleq \operatorname*{argmin}_{\tilde{\boldsymbol{\theta}}, \boldsymbol{\phi}} \operatorname*{argmax}_{\lambda_{\text{co}} \geq 0} \mathcal{L}_{\mathbf{I}}(\tilde{\boldsymbol{\theta}} \odot \hat{\boldsymbol{z}}(\boldsymbol{\phi})) + \lambda_{\text{co}} \left(\texttt{BPP}(\texttt{cast}(\tilde{\boldsymbol{\theta}} \odot \hat{\boldsymbol{z}}(\boldsymbol{\phi})) - \tau_{\texttt{BPP}} \right). \tag{6.3}$$

We apply *simultaneous* gradient descent on $(\tilde{\boldsymbol{\theta}}, \boldsymbol{\phi})$ and projected (to $\mathbb{R}^+$) gradient ascent on λ_{co}. Note that unlike the penalized formulation in which the multiplicative factor for the constraint is fixed, here it is dynamically adjusted throughout the optimization. As detailed in Appx. B.2, the medians face differentiability issues when they saturate at 0 or 1. Thus we employ the *expected* $\texttt{BPP}(\tilde{\boldsymbol{\theta}} \odot \bar{\boldsymbol{z}}(\boldsymbol{\phi}))$ as a proxy-constraint [Cot+19b] for computing the gradient for the update of the primal parameters.

Our constrained formulation grants a direct control over the compression rate of the final model. This provides an algorithmic approach to retain the best possible performance *and* achieve a specific target BPP, without having to perform searches over the hyper-parameter λ_{pen} or the model architecture.

6.3 Experiments

We perform experiments on the Kodak image dataset [Kod91] consisting of 24 images of size 768×512. We compare our approach with various image compression techniques: COIN [Dup+21], unstructured magnitude pruning with fine-tuning and JPEG [Wal92]. Details on the implementation and hyper-parameter configurations can be found in Appx. B.3. Comprehensive experiments and qualitative comparisons are presented in Appx. B.4.

We make the difficulty of the task equal across techniques. For example, for a BPP budget of 0.3, we train a COIN model of dimensions 2-10x[28]-3. Thus, COIN models are fully dense and have a fixed BPP throughout training. Since L_0onie and magnitude pruning remove some of the parameters, we initialize them from larger models and require that they deliver a final model with a BPP of 0.3. The experimental settings used for each budget are provided in Appx. B.3.

Benchmark performance. Fig. 6.2 displays the performance of all methods at various BPP budgets. Magnitude pruning consistently under-performs all other approaches, both immediately after pruning (see Fig. 6.3) and after fine-tuning. This is remarkable considering that we initialize magnitude pruning from a larger, fully trained COIN model. L_0onie slightly surpasses the performance obtained by COIN. On the other hand, although JPEG struggles at high compression rates it clearly dominates at larger BPPs; and its compression time is negligible in comparison to the other techniques.

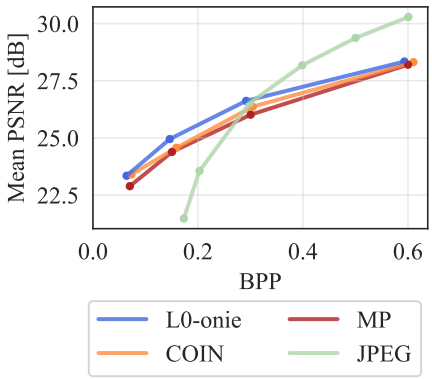


Figure 6.2: PSNR achieved by different techniques over the entire Kodak dataset.

Better performance, faster. Fig. 6.3 shows the PSNR of the compressed model as a function of the training *time*. As expected, the larger L_0onie model achieves better reconstructions much faster. However, recall that the original L_0onie model has a worse compression rate. To ensure a fair comparison, we refresh the PSNR meter for the L_0onie experiments as soon as they satisfy the BPP constraint – this is marked by a drop in the maximum PSNR metric.

Once the L_0onie models become feasible, they rapidly match and even surpass the performance of COIN. Despite each individual gradient step taking more time for the large L_0onie models, the L_0onie approach reaches a *higher PSNR, faster than COIN*. Note that for the same wall-time budget as COIN, L_0onie achieves comparable or higher PSNR, while respecting the BPP constraint.

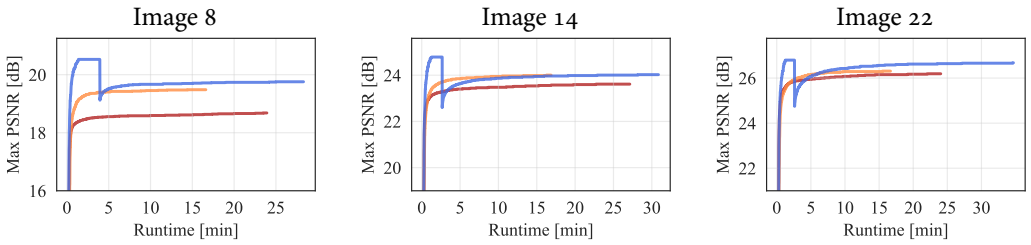


Figure 6.3: Performance comparison during runtime for L_0onie, COIN and magnitude pruning with a target of 0.3 BPP. All methods are trained for 50k steps. Whenever the L_0onie model achieves the desired BPP, we reset its best PSNR meter.

Training dynamics. Fig. 6.4 illustrate the training dynamics of all the methods. We include COIN and magnitude pruning experiments as baselines, whose BPP is fixed during training. These techniques exhibit a similar behavior, with fast initial growth, followed by a slow saturation period.

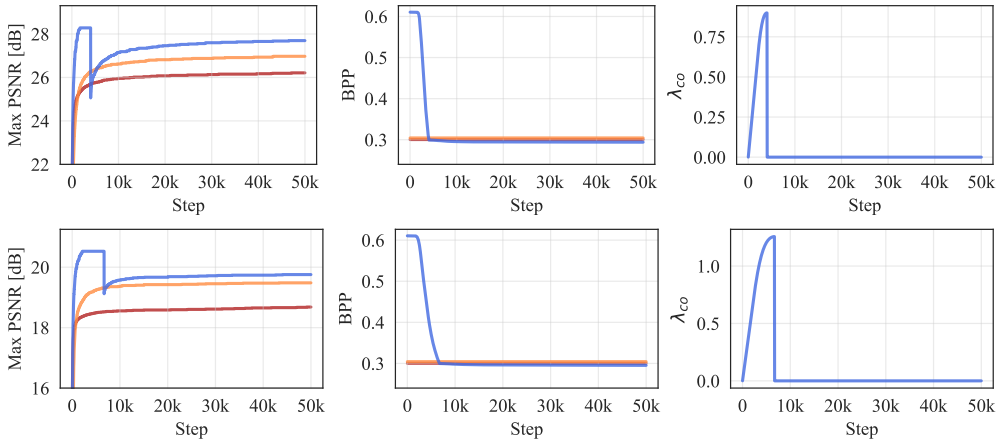


Figure 6.4: Training dynamics for L_0onie, COIN and magnitude pruning on images 7 (top) and 8 (bottom) of the Kodak dataset with a target of 0.3 BPP.

L_0onie is initialized from a model with an unfeasible BPP. This causes the Lagrange multiplier λ_{co} to grow at the beginning of training. In turn, the growing multiplier exerts a force that leads the optimization to trade-off reconstruction performance in order to reduce the model's BPP. Note that this feasibility is obtained relatively early during training. After the initial drop in PSNR, the now feasible model can focus on optimizing the reconstruction performance. We observe that once feasible, the PSNR for L_0onie model quickly recovers and surpasses that of COIN.

Note that after reaching feasibility, the BPP statistics barely change for the L_0onie model. Thus optimization is effectively taking place over a particular subnetwork. Pruning the network at this stage and "fusing" the gates and weights could allow for more efficient training. These gains could be particularly substantial when using structured sparsity, which was beyond the scope of our work.

6.4 Limitations and future work

Recent work has improved the efficiency of COIN by considering a meta-learning framework [Dup+22; Lee+21]. As demonstrated by Schwarz and Teh [ST22], sparsity and meta-learning are complementary avenues for improving INR-based compression. Despite the controllability advantages of our constrained approach, further research is required for understanding the optimization dynamics of INRs, both in the constrained and unconstrained settings.

For simplicity we concentrated on the unstructured sparsity case in this work. However, the compression gains resulting from structured sparsity (for example, grouping all weights associated with an input neuron under a shared gate) could be more notorious and more easily materialized. This is particularly important during training: once the INR becomes sparse, its unused parameters can be removed to enable faster training.

Finally, our sparsity-constrained formulation can support progressive sparsification of the model to reach different compression rates. Once a model has achieved a desired BPP, this very same model can be used in a straightforward manner as a starting point to obtain INRs with higher compression rates. In contrast, it is not immediately evident how to achieve this progressive sparsification using vanilla COIN models or penalized L_0 formulations, while reliably controlling the resulting compression rate.

6.5 Conclusion

We propose L_0onie, a sparse extension of COIN models trained using a constrained formulation. Our method allows to leverage the faster learning of overparametrized networks, while respecting a desired compression rate, without requiring costly hyperparameter tuning or architecture search.

Prologue to the Third Contribution

7

Article Details

Meraj Hashemizadeh*, Juan Ramirez*, Rohan Sukumaran, Golnoosh Farnadi, Simon Lacoste-Julien and Jose Gallego-Posada. **Balancing Act: Constraining Disparate Impact in Sparse Models**. This paper was published at *ICLR*, 2024.

* *Equal contribution.*

Author Contributions

Jose Gallego-Posada proposed the original idea. Meraj Hashemizadeh and Juan Ramirez led the code implementation. Meraj Hashemizadeh and Juan Ramirez spearheaded the execution of the experiments with significant contributions from Rohan Sukumaran. Jose Gallego-Posada and Juan Ramirez led the writing of the paper. Rohan Sukumaran contributed to the literature review and provided expertise in fairness-related topics. Jose Gallego-Posada provided hands-on guidance throughout the project. Golnoosh Farnadi and Simon Lacoste-Julien provided general supervision. Simon Lacoste-Julien guided Jose Gallego-Posada's advisory role.

Context

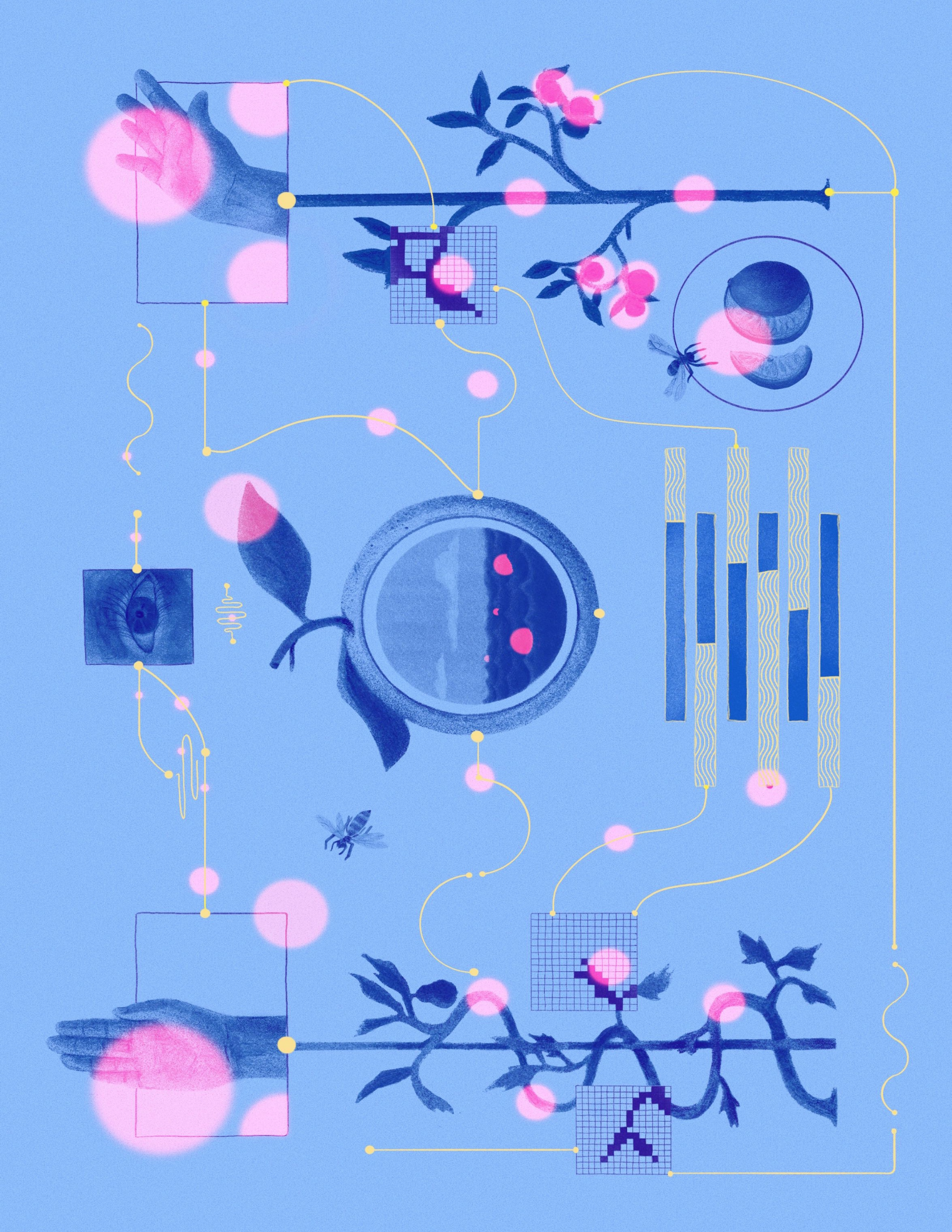
The idea for this work arose during the rebuttal phase for the paper presented in Chapter 4. We were familiar with the work of Hooker et al. [Hoo+19] documenting the disparate impact of pruning. We considered including an additional experiment for our submission to illustrate the *extensibility*† advantages of constrained optimization, combining sparsity and disparate impact constraints.

† *The ability to modularly "stack" desired behaviors as additional constraints, just like penalized formulations allow for multiple additive penalty terms*

At the time, Jose Gallego-Posada was an intern at Qualcomm Amsterdam supervised by Christos Louizos. Christos suggested developing this experiment more carefully rather than adding one more experiment to an already-substantial submission. This recommendation eventually led to a year-long investigation on the disparate impact of sparsified models. We faced many experimental challenges, as existing literature made claims explaining the disparate impact phenomenon based on factors like the size of a protected group, which did not necessarily generalize across tasks.

Opting for a slow-science approach was one of the factors enabling us to identify the lack of generalization present in existing techniques‡ for mitigating disparate impact in sparse models. We believe that documenting this scientific challenge is an important first step towards the development of robust mitigation methods with good generalization properties in practical model compression tasks.

‡ *Including our method.*



8 Balancing Act: Constraining Disparate Impact in Sparse Models

ABSTRACT

Model pruning is a popular approach to enable the deployment of large deep learning models on edge devices with restricted computational or storage capacities. Although sparse models achieve performance comparable to that of their dense counterparts at the level of the entire dataset, they exhibit high accuracy drops for some data sub-groups. Existing methods to mitigate this disparate impact induced by pruning (i) rely on surrogate metrics that address the problem indirectly and have limited interpretability; or (ii) scale poorly with the number of protected sub-groups in terms of computational cost. We propose a constrained optimization approach that *directly addresses the disparate impact of pruning*: our formulation bounds the accuracy change between the dense and sparse models, for each sub-group. This choice of constraints provides an interpretable success criterion to determine if a pruned model achieves acceptable disparity levels. Experimental results demonstrate that our technique scales reliably to problems involving large models and hundreds of protected sub-groups.

8.1 INTRODUCTION

Current deep learning practice displays a trend towards larger architectures [Bom+21], as exemplified by popular models such as GPT-4 [Ope23], Llama 2 [Tou+23] and DALL-E 2 [Ram+22]. Model compression techniques such as pruning [GEH19], knowledge distillation [HVD15], or quantization [Gho+21] are crucial towards enabling the deployment of large models across a wide range of platforms, including resource-constrained edge devices like smartphones.

Despite achieving comparable performance at an aggregate level over the entire dataset, pruned models often exhibit significant accuracy reduction for some data sub-groups [Hoo+19; Hoo+20; Pag20]. In particular, under-represented groups can suffer high performance degradation while the overall performance remains unaffected, thus exacerbating systemic biases in machine learning models. [Tra+22] refer to this phenomenon as the *disparate impact of pruning*.

Existing mitigation methods face challenges in terms of interpretability and scalability to a large number of sub-groups. [Tra+22] introduce constraints aiming to equalize the loss of the sparse model across sub-groups. However, their approach does not account for the unequal group-level performance of the dense model. Moreover, while the loss can be a useful surrogate for training, this method addresses the disparate impact

issue indirectly as it focuses on controlling the loss, rather than group-level changes in accuracy. Alternatively, [LKJ22] compute per-group importance scores for every model parameter to determine the weights to be pruned. This approach becomes prohibitively expensive when the model or the number of sub-groups is large.

In this work, we characterize the disparate impact of pruning in terms of the group-level accuracy gaps between the dense and sparse models. Additionally, we propose a problem *formulation that directly addresses the disparate impact of pruning* by imposing constraints on the per-group excess accuracy gaps (CEAG). A key advantage of our proposed formulation is that it *enjoys interpretable semantics*: feasible solutions of our optimization problem correspond to models with low pruning-induced disparity. Finally, our approach introduces a *negligible computational overhead* (Appx. C.5.1) compared to (disparity-agnostic) naive fine-tuning of the sparse model, making it applicable to problems with large numbers of groups, such as intersectional fairness tasks.

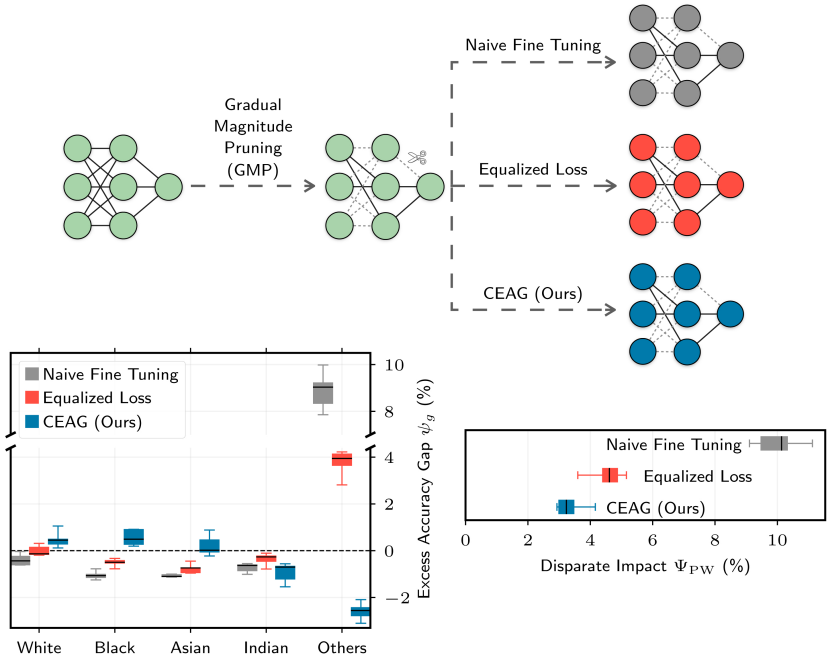


Figure 8.1: **Top:** A dense model is sparsified with GMP, and then subjected to either (i) naive fine-tuning (NFT, using ERM), (ii) equalized loss constraints [Tra+22, EL], or (iii) our approach (CEAG). **Bottom:** Positive (resp. negative) excess accuracy gaps (EAGs, §8.3.1) indicate groups whose performance degraded more (resp. less) than the model's overall accuracy change. Models with low disparate impact have EAGs that concentrate around zero. **CEAG consistenly yields models with lower disparity (Ψ_{PW}, §8.3.1) than NFT and EL.** For example, NFT yields a 10% hyper-degradation (EAG, ψ_g) on group *Others*. Results correspond to race prediction on UTKFace, with race as group attribute at 90% sparsity. Metrics on the training set and averaged over 5 seeds.

Fig. 8.1 illustrates the reliability of our approach at mitigating the disparate impact of pruning. We measure disparity in terms of excess accuracy gaps (EAGs, §8.3.1). Naive fine-tuning yields models that disproportionately affect group *Others*, and while the equalized loss formulation mitigates the issue, *our formulation consistently reduces the pruning-induced disparity*. See §8.5 for further discussion.

The main contributions of our work are as follows:

- We formulate a constrained optimization problem (CEAG, §8.3) that directly controls disparate impact by bounding group-level accuracy gaps between the dense and sparse models.
- We propose an algorithm for solving constrained optimization problems with *non-differentiable, stochastic constraints* (§8.4). We use proxy constraints [Cot+19b] to address non-differentiablity; and introduce replay buffers (§8.4.2) for handling noise in the estimation of constraints.
- Our replay buffers improve the training dynamics of the equalized loss formulation proposed by [Tra+22]. The improved dynamics lead to better models in terms of disparity.
- Our experiments demonstrate that we can reliably mitigate the disparate impact of pruning across multiple architectures, datasets, and sparsity levels (§8.5). These results carry over to tasks with intersectional groups, and up to hundreds of constraints.

Our code is available at `https://github.com/merajhashemi/balancing-act`.

Our experimental results indicate that *all methods considered in this paper (including ours) fail to mitigate pruning-induced disparities on unseen data*. To the best of our knowledge, we are the first to document this generalization challenge. Despite this, our proposed method constitutes a step in the right direction since our approach is *the only one* that reliably mitigates the disparate impact of pruning on the training set. We hope our empirical observations will motivate further research on improving the generalization properties of methods for mitigating the disparate impact of pruning.

8.2 Related Works

Disparate Impact of Pruning. [Hoo+19; Hoo+20] and [Pag20] document the disparate impact of pruning where some classes experience a more significant performance degradation compared to others. Existing methods to mitigate disparity involve fairness-aware pruning [LKJ22] or formulating constraints on a surrogate metric such as the loss [Tra+22].

[LKJ22] propose a pruning technique that removes weights based on a heuristic metric that relates parameters with their importance for predicting samples from each group. This approach scales poorly as it requires computing importance scores for each weight *and* group.

Tran et al. [Tra+22] apply constraints to match the sparse model's *loss* on each subgroup to the aggregate loss. These constraints are (i) agnostic to the performance of the *dense* model on each group and (ii) are based on the loss, which is a surrogate metric for assessing the accuracy-based disparate impact. Since the disparate impact of pruning is measured with respect to a reference model, the equalized loss formulation addresses the problem indirectly. Moreover, loss-based constraints lack the interpretability of the per-group accuracy changes between the sparse and dense models.

Fairness and Constraints. Independent of model pruning, fairness in machine learning models is a well studied problem [Dwo+12; DMB16; VR18; Meh+21; Zem+13; ZG22]. Enforcing fairness with constraints has mainly focused on imposing requirements such as demographic parity, equalized odds, equal opportunity [HPS16], accuracy parity [Aga+18; Ber+21], or combinations of these properties [Zaf+17; Low+21; Bak+20; Shu+22]. The *disparate impact of pruning* is a fairness notion in the context of sparsity that aims to match the performance of a sparse model to that of a reference dense model.

Constrained Optimization. Constrained formulations have gained popularity in different sub-fields of machine learning such as safe reinforcement learning [SAA20], active learning [ENR22] and sparsity [Gal+22]. These constrained formulations lead to stochastic min-max optimization problems, which can be challenging to optimize due to their non-convexity [LJJ20]. We make use of proxy constraints [Cot+19b] to solve problems with interpretable but non-differentiable constraints.

Variance Reduction. The stochasticity in gradient estimates introduces additional optimization challenges [Bez+23]. Variance reduction techniques [Gow+20] have been employed to improve convergence on stochastic optimization [DBL14], and in min-max games [Cha+19]. In this work, we leverage the idea of replay buffers [Mni+13] to reduce the noise in the estimation of stochastic constraints.

8.3 Addressing the Disparate Impact of Pruning via Accuracy Gaps

In this section, we propose using *accuracy gaps* (AGs) to quantify the disparate impact induced by model pruning. AGs are group-level measurements that quantify changes in accuracy between the dense and sparse models. As we will see, large discrepancies in AGs across groups correspond to scenarios where pruning-induced disparity is high. In §8.3.2, we propose a problem formulation that yields models with low disparity by explicitly constraining deviations in the group accuracy gaps.

8.3.1 *Accuracy gaps*

We consider a supervised learning problem on a dataset $\mathfrak{D} = \{(\boldsymbol{x}_i, y_i, g_i)\}_{i=1}^N$ of N i.i.d tuples, each comprising features $\boldsymbol{x} \in \mathcal{X}$, target class $y \in [K]$ and group membership $g \in \mathcal{G}$. The dataset can be partitioned into *sub-groups* $\mathfrak{D}_g \triangleq \{(\boldsymbol{x}_i, y_i, g_i) \in \mathfrak{D} \mid g_i = g\}$ for every $g \in \mathcal{G}$.

Let $h_{\boldsymbol{\theta}} : \mathcal{X} \to \mathbb{R}^K$ be a predictor with parameters $\boldsymbol{\theta} \in \Theta$. The accuracy of $h_{\boldsymbol{\theta}}$ on a sample set $\mathcal{D}$ is $A(\boldsymbol{\theta}|\mathcal{D}) \triangleq \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x},y,g)\in\mathcal{D}} \mathbb{1}\{\texttt{argmax}[h_{\boldsymbol{\theta}}(\boldsymbol{x})] = y\}$. In particular, $A(\boldsymbol{\theta}|\mathfrak{D})$ denotes the model accuracy on the entire dataset, while $A(\boldsymbol{\theta}|\mathfrak{D}_g)$ is the model accuracy on a specific sub-group g.

Given access to a dense pre-trained model, we are interested in the effect of pruning on the accuracy across sub-groups $\mathfrak{D}_g$. In realistic pruning applications the dense model may exhibit different accuracies across sub-groups, thus we do not aim to equalize the accuracy of the sparse model across groups. Therefore, **we argue that the accuracies after pruning should change (approximately) equally across sub-groups.**

Let $\boldsymbol{\theta}_d$ and $\boldsymbol{\theta}_s$ denote the parameters of the dense and sparse models, respectively. We define the *global accuracy gap* $\Delta(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d)$ and *group accuracy gaps* $\Delta_g(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d)$ as:

$$\Delta(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d) \triangleq A(\boldsymbol{\theta}_d|\mathfrak{D}) - A(\boldsymbol{\theta}_s|\mathfrak{D}), \tag{8.1}$$

$$\Delta_g(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d) \triangleq A(\boldsymbol{\theta}_d|\mathfrak{D}_g) - A(\boldsymbol{\theta}_s|\mathfrak{D}_g) \qquad \forall g \in \mathcal{G}. \tag{8.2}$$

A *positive gap* (resp. *negative*) corresponds to a *degradation* (resp. *improvement*) in the performance of the sparse model with respect to that of the dense model. This correspondence holds both at the global $\Delta(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d)$ and group levels $\Delta_g(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d)$.

Disparate Impact of Pruning. Following our discussion above, we say a sparse model $h_{\boldsymbol{\theta}_s}$ experiences low disparate impact (with respect to a dense model $h_{\boldsymbol{\theta}_d}$) if the changes in performance are similar across sub-groups. In other words,

$$\Delta_g(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d) \approx \Delta_{g'}(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d), \forall g, g' \in \mathcal{G}. \tag{8.3}$$

Due to the loss of model capacity caused by pruning, typically $\Delta(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d) > 0$. Thus, we consider $\Delta(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d)$ as the reference point for defining the group *excess accuracy gaps* (EAGs):

$$\psi_g(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d) \triangleq \Delta_g(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d) - \Delta(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d), \qquad \forall g \in \mathcal{G}. \tag{8.4}$$

If $\psi_g(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d) > 0$, then g is more negatively impacted by pruning than the overall dataset. Conversely, $\psi_{g'}(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d) < 0$ indicates that group g' was less affected relative to the overall model degradation.

Note that if $\psi_g = 0, \forall g \in \mathcal{G}$, then it follows that $\Delta_g(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d) = \Delta_{g'}(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d), \forall g, g' \in \mathcal{G}$, and there is no disparate impact. We quantify the disparate impact of pruning via:

$$\Psi_{\text{PairWise}}(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d) \triangleq \max_{g,g' \in \mathcal{G}} \psi_g(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d) - \psi_{g'}(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d) \tag{8.5a}$$

$$= \max_{g \in \mathcal{G}} \Delta_g(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d) - \min_{g' \in \mathcal{G}} \Delta_{g'}(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d). \tag{8.5b}$$

Note that $\Psi_{\text{PW}} \geq 0$ always. Moreover, $\Psi_{\text{PW}} = 0$ if and only if we are in an ideal setting where the accuracy gaps are *equal* across all groups. However, aiming to constraint Ψ_{PW} directly can be difficult in practice (see Appx. C.2.3). Instead, we consider constraints on each individual group EAG.

8.3.2 *Constrained Excess Accuracy Gaps formulation*

We propose to impose upper-bounds (with a tolerance level $\epsilon \geq 0$) on the values of $\psi_g(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d) \leq \epsilon$. Since $\epsilon \geq 0$, the constraints are effectively only enforced on $\psi_g(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d) > 0$, corresponding to groups experiencing hyper-degradation in performance (with respect to the average degradation)*. Imposing a lower bound on group EAGs ψ_g would allow for better control over the resulting disparate impact Ψ_{PW}. However, solving the problem with both of these bounds is challenging due to the small size of the feasible region relative to the estimation noise in the constraints. Appx. C.2.3 provides further discussion and motivation regarding the choice to constrain only positive ψ_g values.

* *Note that the set of hyper-degraded groups* $\{g \in \mathcal{G} \,|\, \psi_g(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d) > 0\}$ *depends directly on the parameters of the sparse model* $\boldsymbol{\theta}_s$ *and thus changes at every training step.*

This choice motivates an *operational definition of disparate impact* which focuses on the group with the highest EAG, given by $\max_g \psi_g$. Bounding this quantity can be achieved by imposing constraints on every EAG. This gives rise to the following optimization problem with per-group constraints:

$$\begin{aligned} (\textsf{CEAG}) \quad & \operatorname*{argmin}_{\boldsymbol{\theta}_s \in \Theta} L(\boldsymbol{\theta}_s | \mathfrak{D}) \\ & \text{s.t.} \quad \psi_g\left(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d\right) = \Delta_g\left(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d\right) - \Delta\left(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d\right) \leq \epsilon, \quad \forall g \in \mathcal{G} \end{aligned} \tag{8.6}$$

where $L(\boldsymbol{\theta}|\mathcal{D})$ is the loss of $h_{\boldsymbol{\theta}}$ on dataset $\mathcal{D}$, and the tolerance $\epsilon \geq 0$ is the maximum allowed EAG.

When $\Delta\left(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d\right) > 0$, the constraints require that the performance degradation for each group be at most the overall model degradation plus the tolerance. Conversely, if $\Delta\left(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d\right) < 0$, the constraints prescribe that all group accuracies must *increase* by at least the overall improvement, except for an ϵ.

8.3.3 *Discussion*

By formulating constraints on EAGs, CEAG directly addresses the disparate impact of pruning and has benefits in terms of interpretability, flexibility, and accountability. See Appx. C.2 for alternative constrained formulations for addressing the disparate impact of pruning.

Tackling disparate impact. Existing methods aim to mitigate disparate impact by enforcing properties on the sparse model while being agnostic to the performance of the dense model. Since EAGs relate the per-group performance of the dense and sparse models, we argue that our approach *actually* addresses pruning-induced disparity, rather than other fairness notions such as loss equalization as proposed by [Tra+22].

Interpretability. The choice of tolerance level ϵ directly translates to bounds on AGs. For example, setting $\epsilon = 1\%$ implies the worst affected class may not lose beyond 1% accuracy compared to the overall model change. In contrast, it is challenging to set interpretable tolerance levels for constraints based on losses.

Flexibility. CEAG allows for some slack in the disparity of the pruned model, as prescribed by the tolerance ϵ. This flexibility allows incorporating application-specific requirements into the learning procedure. For example, small tolerance values allow enforcing strict fairness regulations. Moreover, this flexibility may be necessary in practice since the reduced capacity of the sparse model can make it impossible to attain $\Delta_g\left(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d\right) = \Delta\left(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d\right)\ \forall g \in \mathcal{G}$.

Accountability. Being a constrained approach, establishing feasibility with respect to CEAG constitutes a clear success criterion to determine if a pruned model achieves acceptable disparity levels: a model is only admissible if it satisfies the constraints at a prescribed tolerance level.

8.4 Solving the constrained excess accuracy gaps problem

A popular approach to solve constrained optimization problems such as CEAGin Eq. (8.6) is to formulate its Lagrangian and optimize the resulting min-max problem:

$$\min_{\boldsymbol{\theta}_s \in \Theta} \max_{\boldsymbol{\lambda} \geq 0} \mathfrak{L}(\boldsymbol{\theta}_s, \boldsymbol{\lambda}) \triangleq L(\boldsymbol{\theta}_s | \mathfrak{D}) + \sum_{g \in \mathcal{G}} \lambda_g \left(\psi_g \left(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d \right) - \epsilon \right), \tag{8.7}$$

where $\lambda_g \geq 0$ is the Lagrange multiplier associated with the constraint for group g and $\boldsymbol{\lambda} = [\lambda_g]_{g \in \mathcal{G}}$. We refer to $\boldsymbol{\theta}_s$ as the *primal* parameters, and to $\boldsymbol{\lambda}$ as the *dual* parameters.

Optimizing deep neural networks can be challenging, and generally requires carefully crafted procedures and extensive hyper-parameter tuning [Cho+19]. We are interested in re-using standard techniques for optimizing $\boldsymbol{\theta}_s$. Therefore, we consider a generic optimization protocol on $\boldsymbol{\theta}_s$ and gradient ascent on $\boldsymbol{\lambda}$, instead of specialized optimization approaches for min-max games such as extragradient [Gid+19b; Kor76].

8.4.1 *Optimization with non-differentiable constraints*

A natural next step is to optimize Eq. (8.7) with gradient-based updates. Unfortunately, this is not possible as the ψ_g terms are not continuous (since they are accuracy gaps), and are non-differentiable with respect to $\boldsymbol{\theta}_s$. Therefore, we must resort to a surrogate $\tilde{\psi}_g$ for computing gradients with respect to $\boldsymbol{\theta}_s$. In contrast, Eq. (8.7) is differentiable with respect to $\boldsymbol{\lambda}$, with gradients corresponding to constraint violations. Thus, the dual variables can be updated using the non-differentiable constraints ψ_g. This update scheme is inspired by the proxy-constraint technique introduced by Cotter et al. [Cot+19b].

$$\boldsymbol{\theta}_s^*, \boldsymbol{\lambda}^* \in \begin{cases} \operatorname*{argmin}_{\boldsymbol{\theta}_s \in \Theta} \mathfrak{L}_\theta(\boldsymbol{\theta}_s, \boldsymbol{\lambda}) \triangleq L(\boldsymbol{\theta}_s | \mathfrak{D}) + \sum_{g \in \mathcal{G}} \lambda_g \tilde{\psi}_g \left(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d \right) \\ \operatorname*{argmax}_{\boldsymbol{\lambda} \geq 0} \mathfrak{L}_\lambda(\boldsymbol{\theta}_s, \boldsymbol{\lambda}) \triangleq \sum_{g \in \mathcal{G}} \lambda_g \big(\psi_g \left(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d \right) - \epsilon \big), \end{cases} \tag{8.8}$$

Specifically, we choose surrogates $\tilde{\psi}_g$ given by the *excess (negative) loss gaps*: $\tilde{\psi}_g \left(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d \right) \triangleq -\big(L(\boldsymbol{\theta}_d | \mathfrak{D}_g) - L(\boldsymbol{\theta}_s | \mathfrak{D}_g)\big) + \big(L(\boldsymbol{\theta}_d | \mathfrak{D}) - L(\boldsymbol{\theta}_s | \mathfrak{D})\big)$. Note that $\tilde{\psi}_g$ has the same structure as ψ_g, but replaces accuracy measurements with *negative* loss terms. This is a reasonable choice of surrogate function since *drops* in accuracy for the sparse model correspond to *increases* in loss.

Eq. (8.8) represents a two-player, non-zero-sum game. Rather than replacing the non-differentiable constraints with their surrogates everywhere, this approach only performs the replacement *when necessary*, i.e., for computing gradients for the primal parameters. Preserving the actual constraints on the dual objective $\mathfrak{L}_\lambda(\boldsymbol{\theta}_s, \boldsymbol{\lambda})$ is useful as it results in a problem closer to Eq. (8.7).

Equation (8.8) can be optimized via gradient descent on $\boldsymbol{\theta}_s$ (based on $\mathfrak{L}_\theta$) and gradient ascent on $\boldsymbol{\lambda}$ (based on $\mathfrak{L}_\lambda$). Alternating gradient descent-ascent (Alt-GDA) updates yield the following scheme:

$$\lambda_g^{(t+1)} = \left[\lambda_g^{(t)} + \eta_\lambda \left(\psi_g\left(\boldsymbol{\theta}_s^{(t)}, \boldsymbol{\theta}_d\right) - \epsilon\right)\right]_+ \tag{8.9}$$

$$\boldsymbol{\theta}_s^{(t+1)} = \boldsymbol{\theta}_s^{(t)} - \eta_\theta \left[\nabla_\theta L\left(\boldsymbol{\theta}_s^{(t)} | \mathfrak{D}\right) + \sum_{g \in \mathcal{G}} \lambda_g^{(t+1)} \nabla_\theta \tilde{\psi}_g\left(\boldsymbol{\theta}_s^{(t)}, \boldsymbol{\theta}_d\right)\right], \tag{8.10}$$

where η_θ and η_λ are step-sizes and $[\,\cdot\,]_+ = \max(\cdot, 0)$. We initialize the Lagrange multipliers to $\boldsymbol{\lambda}^{(0)} = \mathbf{0}$. Appx. C.1 contains more details on non-convex constrained optimization.

8.4.2 *Stochastic constraints and replay buffers*

In practice, the problem in Eq. (8.6) is solved by using mini-batch samples from the dataset to estimate the objective function, the constraints, and their gradients. This procedure can yield constraint estimates with high variance across mini-batches, especially for under-represented groups; or for all groups when the number of constraints is large. In extreme cases, a mini-batch may contain very few samples from a given sub-group, leading to multiplier updates based on very noisy estimates.

We overcome these issues by estimating constraints based on information across multiple mini-batches. For calculating AGs, (i) we compute the performance of the dense model *on the whole dataset* (once at the beginning of training), and (ii) we estimate the accuracy of the sparse model from per-sample accuracy measurements on the k most recent datapoints of each group. We refer to the data structure that stores historic accuracies as a *replay buffer* (RB), given the analogy to the technique used in reinforcement learning [Mni+13]. The choice of buffer size k introduces a trade-off between reducing the variance of the constraints, and biasing estimates towards old measurements.

These adjustments reduce variance in the estimation of the constraints, thus yielding stable updates for the multipliers. This allows us to solve Eq. (8.6) in settings with large numbers of constraints relative to the choice of batch size. We do not apply variance reduction on the model updates. For details on our implementation of replay buffers, see Appx. C.3. For experimental evidence on their benefits, see §8.5.3 and Appx. C.3.1.

8.4.3 *Algorithmic details*

Algo. 2 presents our approach for solving CEAG. Note that Algo. 2 is applicable to a broader class of constrained optimization problems with stochastic constraints, including the equalized loss formulation of [Tra+22] (see Appx. C.2.1 for details).

Computational Overhead. The constrained approach in Algo. 2 represents a negligible computational overhead compared to fine-tuning the sparse model with empirical risk minimization. An iteration of Alt-GDA (Eq. (8.9)) requires *one forward pass and one backward pass* through the model since the same iterate of $\boldsymbol{\theta}_s$ is used for both the primal and dual updates. This matches the cost of gradient descent for ERM, except for the minimal overhead associated with the evaluation of constraints after the forward pass. Note that, given our choice of surrogate, the gradient of the Lagrangian with respect to

Algorithm 2 Constrained Excess Accuracy Gap (CEAG)

Input: θ: Initial model parameters, η_θ: Primal step-size, η_λ: Dual step-size, k: Memory size for replay buffer, ϵ: Tolerance hyper-parameter, B: Batch size, T: Total number of iterations, A^g_{dense}: Accuracy of the dense model on each group g, A_{dense}: Aggregate accuracy of the dense model.

$\lambda_g \leftarrow 0, \quad \forall g \in \mathcal{G}$ {Initialize dual parameters}
$\texttt{buf}_g \leftarrow \texttt{queue}(k), \quad \forall g \in \mathcal{G}$ {Initialize replay buffer}
for $\texttt{iter} = 1, \ldots, T$ **do**
  $\mathbf{x}, \mathbf{y}, \mathbf{g} \leftarrow$ Sample $\{(x_i, y_i, g_i)\}_{i=1}^{B} \sim \mathfrak{D}$ {Sample batch from training set}
  $\texttt{idx}_g \leftarrow (\mathbf{g} == g), \quad \forall g \in \mathcal{G}$ {Calculate sub-group indices for batch}
  $\hat{\mathbf{y}} \leftarrow h_\theta(\mathbf{x})$ {Compute forward-pass}
  $\texttt{buf}_g \leftarrow \textsc{UpdateBuffer}(\texttt{buf}_g, \hat{\mathbf{y}}, \mathbf{y}, \texttt{idx}_g), \quad \forall g \in \mathcal{G}$ {Update replay buffer}
  $\psi_g \leftarrow \textsc{QueryBuffers}\left(\{\texttt{buf}_g\}_{g=1}^{\mathcal{G}}, k, \{A^g_{\text{dense}}\}_{g=1}^{\mathcal{G}}, A_{\text{dense}}\right)$ {Query buffers}
  $\tilde{\psi}_g \leftarrow \textsc{ComputeSurrogate}(\hat{\mathbf{y}}, \mathbf{y}, \texttt{idx}_g), \quad \forall g \in \mathcal{G}$ {Compute surrogates}
  $\lambda_g \leftarrow \max\{0, \lambda_g + \eta_\lambda(\psi_g - \epsilon)\}, \quad \forall g \in \mathcal{G}$ {Update dual params}
  $\texttt{grad}_\theta \leftarrow \nabla_\theta \left[L(\theta|(\mathbf{x}, \mathbf{y})) + \sum_{g \in \mathcal{G}} \lambda_g \tilde{\psi}_g\right]$ {Compute primal gradient}
  $\theta \leftarrow \textsc{PrimalOptimUpdate}(\eta_\theta, \texttt{grad}_\theta)$ {Update model params}
end for
return θ

$\boldsymbol{\theta}_s$ is a weighted average of the per-sample loss gradients, which autograd frameworks can compute as efficiently as $\nabla_\theta L(\boldsymbol{\theta}_s|\mathfrak{D})$. For empirical evidence supporting the claim that CEAG has negligible computational overhead compared to ERM, see Appx. C.5.1.

Memory Cost. The memory overhead of our approach is negligible in the context of training deep networks: storing the dual variables requires one float per constraint, and the replay buffers store only $|\mathcal{G}|$ booleans for each one of the k slots in the buffer memory.

8.5 Experiments

In this section, we present an empirical comparison between naive fine-tuning, equalized loss [Tra+22], and our proposed CEAG approach. The main goal of our experiments is to train sparse models with low pruning-induced disparity. While low disparity may introduce a trade-off with aggregate performance, we aim to achieve comparable overall accuracy to mitigation-agnostic methods. We explore the reliability and accountability of our approach, along with the effect of replay buffers on the constrained optimization problem. Our experiments demonstrate that our method successfully scales to problems with hundreds of groups.

8.5.1 *Experimental setup*

Tasks and architectures. We carry out experiments on the FairFace [KJ21] and UTK-Face [ZSQ17] datasets, following the works of [LKJ22] and [Tra+22]. Additionally, we perform experiments on CIFAR-100 [Kri09], a task with a large number of sub-groups. The choice of target and group attributes for each dataset is specified in Appx. C.4.1. Details on the architectures and pre-trained models are provided in Appx. C.4.3 and C.4.4

Baseline methods. We compare with three baseline mitigation methods (i) NFT: the last iterate when fine-tuning the sparse model via ERM, (ii) NFT+ES: the best iterate of NFT in terms of test accuracy (early stopping), and (iii) EL+RB: our re-implementation of the equalized loss formulation proposed by [Tra+22], enhanced with replay buffers (see Appx. C.2.1). The optimization hyper-parameters employed for each mitigation method (including CEAG) are described in Appx. C.4.6.

Model pruning. Previous work has shown that gradual magnitude pruning (GMP) [ZG17] achieves SOTA aggregate performance on unstructured sparsity tasks [Bla+20]. Because of this (and its simplicity), we employ unstructured GMP on all our tasks. GMP gradually prunes the model by removing parameters with the smallest magnitude once every epoch. The remaining weights are fine-tuned in between pruning episodes. We carry out GMP during the first 15 epochs. Appx. C.4.5 provides further details on our pruning protocol.

Choice of sparsity levels. For very high levels of unstructured sparsity (over 95%), [GEH19] observe that pruning has a devastating impact on the overall performance of ResNet-50 models [He+16]. In contrast, performance remains essentially unaffected for models with up to 85% sparsity. These observations may not carry over to other architectures such as MobileNets [San+18], or other ResNets. Nonetheless, our experiments stick to the [85%, 95%] range, except for FairFace experiments, where we consider 99% sparsity, akin to FairGrape [LKJ22].

Software. Our implementations use PyTorch 1.13.0 [Pas+19] and the Cooper library for constrained optimization [Gal+24].

Experimental uncertainty. All metrics reported in our tables and plots follow the pattern `avg` $\pm$ `std`. Unless mentioned otherwise, all our experimental metrics are aggregated across 5 seeds.

Appx. C.6 presents comprehensive experimental results for multiple tasks and sparsities.

8.5.2 *FairFace and UTKFace*

ResNet-34 Models on FairFace. Table 8.1 includes results for FairFace classification at 99% sparsity. We compare the behavior of NFT, NFT+ES, EL+RB, and CEAG. We quote the results reported for the FairGRAPE technique*, aggregated over 3 seeds.

** We do not re-run FairGRAPE owing to its high computational cost, see discussion in Appx. C.5.2*

We observe that CEAG attains a feasible model in training ($\max_g \psi_g \leq \epsilon$), as well as the smallest $\max_g \psi_g$ both in the training and test sets. This does not come at the cost of aggregate performance, as all methods achieve a comparable test accuracy of around 65%. We observe that FairGRAPE's $\max_g \psi_g$ and Ψ_{PW} are significantly higher than that of all other methods.

MobileNet-V2 Models on UTKFace. Fig. 8.2 illustrates results for UTKFace with race as group attribute. CEAG consistently attains feasible models in training, and the smallest values of $\max_g \psi_g$ in the test set. CEAG attains comparable performance to NFT and EL+RB in the test set.

Table 8.1: Race prediction task on FairFace with race as group attribute. **CEAG achieves a $\max_g \psi_g$ within the prescribed threshold.** Tol (ϵ) is the tolerance hyper-parameter of CEAG. We do not specify ϵ for other formulations as they do not admit a tolerance.

Sparsity	Method	Train				Test		
		Accuracy	Ψ_{PW}	$\max_g \psi_g$	Tol (ϵ)	Accuracy	Ψ_{PW}	$\max_g \psi_g$
99	NFT	76.1 $\pm$ 0.2	3.9 $\pm$ 0.9	2.3 $\pm$ 0.3	–	65.2 $\pm$ 0.4	4.2 $\pm$ 0.5	2.1 $\pm$ 0.5
	NFT + ES	74.0 $\pm$ 2.5	7.2 $\pm$ 3.3	4.0 $\pm$ 1.4	–	65.4 $\pm$ 0.4	6.3 $\pm$ 2.6	2.9 $\pm$ 1.3
	EL + RB	76.1 $\pm$ 0.1	8.8 $\pm$ 1.3	2.6 $\pm$ 0.2	–	65.1 $\pm$ 0.4	6.0 $\pm$ 1.5	2.4 $\pm$ 0.4
	FairGRAPE	–	–	–	–	65.1	15.9	10.7
	CEAG	76.2 $\pm$ 0.1	3.5 $\pm$ 0.6	1.8 $\pm$ 0.4	$\leq$ 2% ✓	65.2 $\pm$ 0.4	4.3 $\pm$ 0.8	2.0 $\pm$ 0.3

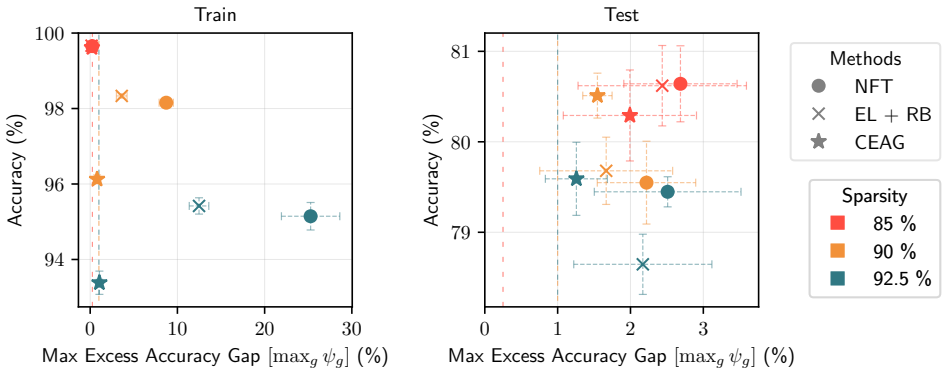


Figure 8.2: Trade-off between disparity and accuracy for UTKFace race prediction with race as group attribute. NFT and EL+RB yield models with high disparity. **In contrast, CEAG consistently produces models that mitigate the disparate impact of pruning.** CEAG's gains do not entail a degradation in overall test accuracy. Vertical dashed lines indicate the tolerance (ϵ) of our method, with colors indicating the sparsity level.

Table 8.2 presents results for UTKFace with intersectional groups (race $\cap$ gender). NFT and NFT+ES have very high disparity metrics. In contrast, CEAG attains a feasible $\max_g \psi_g$ and the smallest Ψ_{PW} in the training set, for all sparsities. Our approach has worse aggregate performance than NFT and EL+RB in the train set; however, the test accuracy of these three methods is comparable.

For NFT, both Fig. 8.2 and Table 8.2 show significantly higher disparity metrics in training when compared to in test. This is an indicator that the sparse model achieves good performance in training by overfitting to the majority groups and losing a lot of performance on the under-represented groups.

8.5.3 *Scaling to large numbers of groups*

CifarResNet-56 models on CIFAR-100. Table 8.3 contains results for CIFAR-100 classification at 92.5% sparsity. By having the groups correspond to class labels, constrained formulations for this experiment have 100 constraints. We include two additional experiments to illustrate the importance of replay buffers: equalized loss (EL), and CEAG (no RB), both *without replay buffers*.

Table 8.2: Race prediction task on the UTKFace dataset with the intersection of race and gender as group attribute. For instance, if a sample has race *Black* and gender *Female*, its group is *Black-Female*. **CEAG consistently achieves a** $\max_g \psi_g$ **within the threshold**.

Sparsity	Method	Train				Test		
		Accuracy	Ψ_{PW}	$\max_g \psi_g$	Tol (ϵ)	Accuracy	Ψ_{PW}	$\max_g \psi_g$
90	NFT	98.1 $\pm$ 0.1	11.5 $\pm$ 0.7	10.0 $\pm$ 0.7	–	79.6 $\pm$ 0.5	8.9 $\pm$ 2.3	3.1 $\pm$ 0.5
	NFT + ES	90.5 $\pm$ 4.7	49.8 $\pm$ 23.0	44.8 $\pm$ 20.8	–	81.0 $\pm$ 0.2	12.0 $\pm$ 5.3	6.9 $\pm$ 4.8
	EL + RB	98.3 $\pm$ 0.2	3.2 $\pm$ 0.6	2.4 $\pm$ 0.6	–	79.4 $\pm$ 0.5	11.4 $\pm$ 0.9	3.0 $\pm$ 1.1
	CEAG	96.2 $\pm$ 0.1	2.4 $\pm$ 0.6	1.0 $\pm$ 0.3	$\leq$ 3% ✓	80.2 $\pm$ 0.1	6.0 $\pm$ 2.5	2.3 $\pm$ 1.0
92.5	NFT	95.1 $\pm$ 0.2	34.2 $\pm$ 1.6	30.7 $\pm$ 1.5	–	79.2 $\pm$ 0.2	8.8 $\pm$ 3.2	3.6 $\pm$ 1.3
	NFT + ES	91.2 $\pm$ 2.7	53.3 $\pm$ 9.6	48.0 $\pm$ 8.3	–	80.4 $\pm$ 0.4	7.5 $\pm$ 3.4	5.4 $\pm$ 3.1
	EL + RB	95.4 $\pm$ 0.3	11.1 $\pm$ 1.5	8.6 $\pm$ 1.4	–	78.7 $\pm$ 0.3	16.3 $\pm$ 3.9	3.3 $\pm$ 0.6
	CEAG	93.4 $\pm$ 0.3	3.8 $\pm$ 0.4	2.3 $\pm$ 0.4	$\leq$ 3% ✓	79.5 $\pm$ 0.1	10.8 $\pm$ 2.2	3.3 $\pm$ 1.0

Disparity metrics for EL and CEAG are better when employing replay buffers, both on the train and test sets. This difference is more notable for EL. We also observe the RBs improve the training dynamics of the dual variables (Appx. C.3.1). CEAG obtains the best disparity on the train set. Nonetheless, all approaches have a significant generalization gap in terms of disparity measurements. We observe that the best accuracy and the smallest $\max_g \psi_g$ on the test set are obtained by EL+RB.

Table 8.3: CIFAR-100 classification with class labels as protected attribute at 92.5% sparsity. EL is the equalized loss formulation without replay buffers; CEAG (no RB) is similarly defined.

Sparsity	Method	Train				Test		
		Accuracy	Ψ_{PW}	$\max_g \psi_g$	Tol (ϵ)	Accuracy	Ψ_{PW}	$\max_g \psi_g$
92.5	NFT	99.8 $\pm$ 0.0	3.7 $\pm$ 0.9	3.0 $\pm$ 0.9	–	64.9 $\pm$ 0.4	26.2 $\pm$ 5.2	14.3 $\pm$ 3.4
	NFT + ES	99.3 $\pm$ 0.2	6.8 $\pm$ 1.9	5.8 $\pm$ 1.8	–	65.2 $\pm$ 0.4	27.4 $\pm$ 2.3	14.6 $\pm$ 2.0
	EL	98.5 $\pm$ 0.1	11.3 $\pm$ 0.9	9.8 $\pm$ 1.0	–	65.3 $\pm$ 0.5	25.8 $\pm$ 2.0	14.1 $\pm$ 1.3
	EL + RB	99.5 $\pm$ 0.0	6.7 $\pm$ 1.4	5.7 $\pm$ 1.5	–	65.3 $\pm$ 0.4	24.2 $\pm$ 2.9	13.3 $\pm$ 2.4
	CEAG (no RB)	99.6 $\pm$ 0.0	2.6 $\pm$ 0.3	1.7 $\pm$ 0.2	$\leq$ 2% ✓	65.0 $\pm$ 0.4	27.2 $\pm$ 2.6	14.9 $\pm$ 2.5
	CEAG	99.6 $\pm$ 0.0	2.4 $\pm$ 0.2	1.6 $\pm$ 0.1	$\leq$ 2% ✓	64.8 $\pm$ 0.3	25.0 $\pm$ 1.9	13.8 $\pm$ 1.2

8.6 DISCUSSION

It is important to develop techniques that reliably mitigate the disparate impact of pruning since deploying pruned models can have downstream consequences. We observe that NFT is unsucessful at doing this, and NFT+ES amplifies the disparity induced by pruning. In contrast, CEAG reduces disparity while achieving comparable aggregate performance to NFT. However, *we observe that all mitigation approaches may fail to mitigate disparate impact on unseen data*.

Mitigating the disparate impact of pruning. Unlike other mitigation methods, our approach consistently mitigates the disparate impact of pruning on the training set. We observe this across a wide range of tasks and architectures. In contrast, other mitigation approaches generally yield worse maximum degradation $\max_g \psi_g$. In particular, NFT+ES yields models with very high disparity.

Accuracy trade-off. CEAG may introduce a trade-off in terms of accuracy in order to satisfy the disparity requirements. On the train set, we observe a small degradation

in performance in comparison to NFT, typically of at most 2%; on the test set, CEAG's accuracy is comparable to that of NFT.

Reliability. Our approach reliably yields models within the requested disparity levels. Moreover, CEAG results in the smallest variance of the $\max_g \psi_g$ and Ψ_{PW} metrics across seeds.

Generalization. Although CEAG reliably satisfies the constraints on the train set, this may not transfer to the test set. We highlight that (i) these generalization issues are present for other mitigation methods, and (ii) our approach generally achieves better test disparity than the baselines. Improving the generalization of disparity mitigation methods is an important direction for future research.

8.7 Conclusion

In this paper, we explore mitigating the disparate impact of pruning. We formalize disparate impact in terms of accuracy gaps between the dense and sparse models, and propose a constrained optimization approach for mitigating it. Our formulation offers interpretable constraints and allows for algorithmic accountability. Although other methods can indirectly reduce disparity, our approach reliably addresses the disparate impact of pruning across a wide range of tasks, while attaining comparable aggregate performance. In particular, our method successfully scales to tasks with hundreds of sub-groups. Despite the fact that current mitigation methods exhibit generalization issues, our approach represents a solid step towards mitigating the disparate impact of pruning.

Ethics Statement

- **Facial recognition.** Our paper makes use of datasets that contain face images. We focus on these datasets as they illustrate the disparate impact of pruning, and for comparisons with previous work. We would like to highlight that although our method focuses on reducing the disparate impact across groups, we do not endorse the use of our algorithm in facial recognition systems.
- **Data annotation.** We use the UTKFace [ZSQ17] and FairFace [KJ21] datasets in this work. These datasets include annotations for sensitive demographic attributes such as race, gender, and age. However, it is essential to recognize that these annotations represent normative ways of perceiving gender, race, and age, and we do not endorse or promote these normative categorizations.
- **Ethical sourcing of data.** We don't endorse using datasets where the data may not have been ethically sourced or the workers/subjects involved in the data collection process are not fairly compensated.
- **Fairness notions.** We explore a specific notion of fairness in this paper: the disparate impact of pruning. Our framework can be extended to other fairness notions by incorporating additional constraints. However, certain notions of fairness are incompatible with each other, and a "fair" model in one definition could be "unfair" with respect to another [FSV21]. Therefore, our method should not be considered a solution to all notions of fairness.

- **Disparate impact of pruning.** In this paper, we propose a constrained optimization technique that mitigates the disparate impact of pruning and successfully solve the problem on the training data. Unfortunately, like all other surveyed techniques, we observe significant challenges at mitigating disparate impact on unseen data. We advise practitioners to consider the implications of these generalization challenges when deploying sparse models in real-world systems.
- **Deploying pruned models.** We hope our paper brings about an important discussion on the implications of deploying pruned deep learning models in edge devices. As shown in this work, despite the application of mitigation techniques, pruning can exacerbate systemic biases. In particular, given the generalization issues across mitigation methods, it could cause unintended consequences when used in commercial applications.

Reproducibility Statement

We provide our code*, including scripts to replicate the experiments in this paper. The pseudo-code of our algorithm is described in Algo. 2. Experimental details, as well as the hyper-parameters used in our experiments, are included in Appx. C.4. Our implementation uses the open-source libraries PyTorch [Pas+19] and Cooper [Gal+24].

* *Our code is available at* https://github.com/merajhashemi/balancing-act

Prologue to the Fourth Contribution

9

Article details

Motahareh Sohrabi*, Juan Ramirez*, Tianyue H. Zhang, Simon Lacoste-Julien and Jose Gallego-Posada. **On PI controllers for updating Lagrange multipliers in constrained optimization**. This paper was published at *ICML*, 2024.

* *Equal contribution.*

Author contributions

Jose Gallego-Posada proposed the original idea. The conceptualization of the project was done jointly by Jose Gallego-Posada, Motahareh Sohrabi and Juan Ramirez. Motahareh Sohrabi and Jose Gallego-Posada established the key theoretical results. Jose Gallego-Posada led the writing of the paper with significant contributions from Juan Ramirez and Motahareh Sohrabi. Motahareh Sohrabi led the SVM experiments and the fairness experiments (based on code written by Juan Ramirez). Juan Ramirez led the sparsity experiments. Tianyue H. Zhang created the graphics for interpreting the updates of $\boldsymbol{\nu}$PI, the two-dimensional experiments in the appendix, and several code contributions. Jose Gallego-Posada provided hands-on guidance throughout the project. Simon Lacoste-Julien provided general supervision and guided Jose Gallego-Posada's advisory role.

Context

Having established the advantages of constrained formulations for enforcing complex behaviors in machine learning models, the current paper focuses on improving the techniques used to solve the constrained problems themselves. After all, said advantages are moot† if solving the constrained optimization problems is impractical.

† *and the constrained viewpoint is unlikely to be widely adopted...*

This research was motivated by observations made in Chapter 4 on the (sometimes undesirable) dynamics of gradient-ascent for updating the Lagrange multipliers in constrained optimization problems. The dual restarts scheme presented in Section 4.3.3 is a partial remedy to these challenges and was successful in the L_0-sparsity task, but unfortunately does not generalize to equality constraints or stochastically-estimated constraints.

We identified that the issue lay in the lack of adaptivity in the multiplier updates. Thus, we decided to analyze whether generic adaptive single-objective optimization methods like Polyak, Nesterov or Adam would be sufficient to address these issues. Our exploration led to a negative answer, and decanted in favor of the PI approach. Note that the motivation for our work stems from a complementary perspective to the work of Stooke et al. [SAA20] as they focus exclusively on PID-based techniques. Notably, we were able to show that our proposed $\boldsymbol{\nu}$PI algorithm encompasses both the PID, Polyak and Nesterov methods as special cases.


(output)
(false)
(false)
(intput)
(false)
(false)
(false)
<leafhead/>
(else)
(false)
(yes)
(if)
(false)
(false)
(false)
(false)

10 ON PI CONTROLLERS FOR UPDATING LAGRANGE MULTIPLIERS IN CONSTRAINED OPTIMIZATION

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. [SAA20]. 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.

10.1 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 [Cot+19b; Zaf+19; Fio+20; Has+24], sparsity [Gal+22], active learning [ENR22], reinforcement learning [SAA20; FG21] and model quantization [HER23].

Algorithmic approaches based on the Lagrangian min-max representation of the original constrained optimization problem [BV04, §5] are commonly preferred in the context of neural networks since (i) they are amenable to inexact, gradient-based optimization [Ber16, §5.2], (ii) making it easy to incorporate constraints into existing pipelines for unconstrained optimization [Cot+19b; Gal+24], and (iii) they do not require special structure in the objective or constraint functions (such as convexity or efficient projection onto the feasible set [NW06]).

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 [BCN18], and the potential for instability and oscillations due to the adversarial min-max nature of the Lagrangian [SAA20].

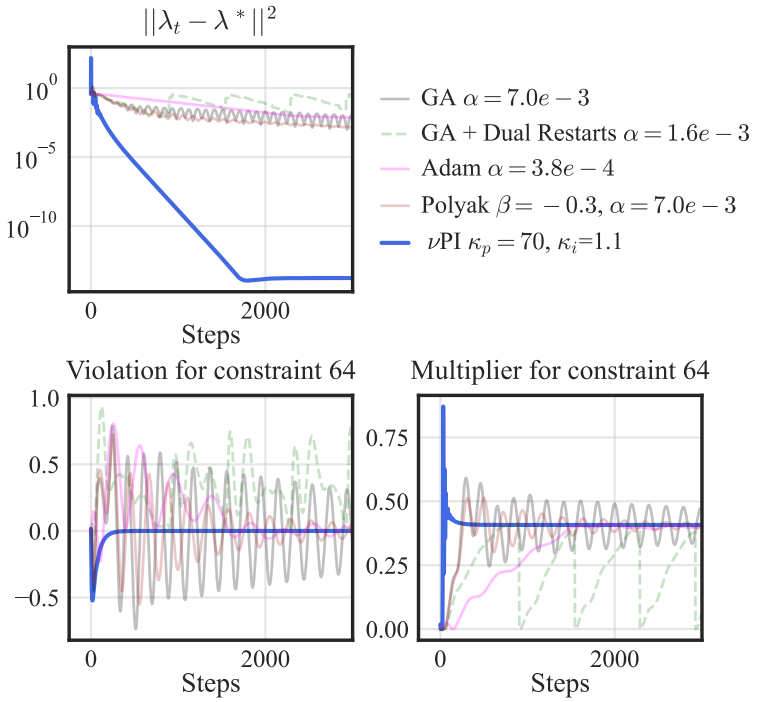


Figure 10.1: Dynamics for different dual optimizers on a hard-margin SVM problem (Eq. (10.12)). Amongst the tested methods, ***ν*PI is the only method to successfully converge to the optimal dual variables**. Each optimizer uses the best hyperparameters found after a grid-search aiming to minimize the 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.

Lagrangian problems are commonly optimized using some variant of gradient-descent ascent (GDA) [AHU58]. Despite local convergence results in idealized settings [LJJ20; Zha+22], the optimization dynamics of GDA typically exhibit instabilities, overshoot or oscillations [PB87; Gid+19a; SAA20; Gal+22].

Alleviating the shortcomings of GDA on Lagrangian problems is an important step towards wider adoption of constrained optimization in deep learning. Recently, STOOKE et al. [SAA20] proposed a solution based on a PID controller [ÅH95] for updating the Lagrange multipliers in safety-constrained reinforcement learning problems. Our manuscript expands on their work by providing an optimization-oriented analysis of *ν*PI (Algo. 3), a related PI controller that incorporates an exponential moving average on the error.

Fig. 10.1 illustrates how our proposed *ν*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 §10.5.1 for further details on this experiement.

Contributions: ① We introduce the *ν*PI algorithm (§10.4) and prove that *ν*PI generalizes popular momentum methods like POLYAK and NESTEROV (Thm. 10.4.1), as well

as traditional PI controllers. ② We provide conceptual insights explaining how $\boldsymbol{\nu}$PI improves the dynamics of the Lagrange multipliers: §10.4.3 presents a qualitative analysis of the updates executed by the $\boldsymbol{\nu}$PI algorithm in contrast to gradient ascent; in §10.4.4 we study the spectral properties of the continuous-time system. ③ In §10.4.5, 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: Our code is available at `https://github.com/motahareh-sohrabi/nuPI`.

Scope: Due to the highly specialized techniques used for training neural networks [Dah+23], 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 optimizer choices for the (primal) model parameters as used in the unconstrained setting.

10.2 Related works

Constrained optimization. We are interested in Lagrangian methods [AHU58] that allow tackling general (nonconvex) constrained optimization problems with differentiable objective and constraints. Classical constrained optimization [NW06; Ber16] techniques include projection methods [Ber76], barrier methods [Dik67], and methods of feasible directions [FW56; Zou60]. 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 [NW06] and the method of multipliers [Ber75], 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 [LJJ20], but may exhibit oscillations [PB87; Gid+19b]. Under stronger assumptions, extragradient [Kor76] and the optimistic gradient method [Pop80] converge at a nearly optimal rate [MOP20a]. These methods, as well as Polyak with negative momentum [Gid+19a] and PID controllers [SAA20], have been shown to dampen the oscillations of GDA. However, negative momentum may be suboptimal for strongly convex-strongly concave min-max problems [ZW21].

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. [An+18] 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

[HL17] and momentum [Rec18]. STOOKE et al. [SAA20] and CASTI et al. [Cas+23] 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. D.1, we elaborate on the distinctions between our work and existing research on PID controllers for optimization.

10.3 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:

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) \;\text{ subject to }\; \boldsymbol{g}(\boldsymbol{x}) \leq \mathbf{0} \;\text{ and }\; \boldsymbol{h}(\boldsymbol{x}) = \mathbf{0}. \tag{10.1}$$

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 §10.2.

The Lagrangian min-max problem associated with the constrained optimization problem in Eq. (10.1) is given by:

$$\min_{\boldsymbol{x}} \max_{\boldsymbol{\lambda} \geq \mathbf{0},\, \boldsymbol{\mu}} \mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \triangleq f(\boldsymbol{x}) + \boldsymbol{\lambda}^\top \boldsymbol{g}(\boldsymbol{x}) + \boldsymbol{\mu}^\top \boldsymbol{h}(\boldsymbol{x}), \tag{10.2}$$

where $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ are vectors of *Lagrange multipliers* associated with the inequality and equality constraints, respectively. Eq. (10.2) 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. (10.2) is simultaneous gradient descent-ascent (GDA) [AHU58]:

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

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:

$$\min_{\boldsymbol{x}} \max_{\boldsymbol{\theta} \in \mathbb{R}^m_+ \times \mathbb{R}^n} \mathfrak{L}(\boldsymbol{x}, \boldsymbol{\theta}) \triangleq f(\boldsymbol{x}) + \boldsymbol{\theta}^\top \boldsymbol{c}(\boldsymbol{x}) \tag{10.3}$$

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 $\mathcal{J}\boldsymbol{c}^*$ 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 $\boldsymbol{c}$.

* $\mathcal{J}\boldsymbol{f} \triangleq \left[\nabla f_1, \cdots, \nabla f_p\right] \in \mathbb{R}^{d \times p}$ *denotes the (transpose) Jacobian matrix of a function* $\boldsymbol{f} : \mathbb{R}^d \to \mathbb{R}^p$.

Alternating updates. Prior work has demonstrated the advantages of alternating updates in min-max optimization: Zhang et al. [Zha+22] 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. [Gid+19b] 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:

$$\begin{cases} \hat{\boldsymbol{\theta}}_{t+1} \leftarrow \boldsymbol{\theta}_t + \eta_{\text{dual}} \nabla_{\boldsymbol{\theta}} \mathfrak{L}(\boldsymbol{x}_t, \boldsymbol{\theta}_t) = \boldsymbol{\theta}_t + \eta_{\text{dual}} \boldsymbol{c}(\boldsymbol{x}_t) \\ \boldsymbol{\theta}_{t+1} \leftarrow \Pi_{\mathbb{R}^m_+ \times \mathbb{R}^n}(\hat{\boldsymbol{\theta}}_{t+1}) \\ \boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \eta_{\text{primal}} \nabla_{\boldsymbol{x}} \mathfrak{L}(\boldsymbol{x}_t, \boldsymbol{\theta}_{t+1}) \\ \qquad\quad = \boldsymbol{x}_t - \eta_{\text{primal}} \left(\nabla f(\boldsymbol{x}_t) + \mathcal{J}\boldsymbol{c}(\boldsymbol{x}_t)\, \boldsymbol{\theta}_t\right) \end{cases} \tag{10.4}$$

The alternating updates in Eq. (10.4), 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. (10.4) for solving the problem in Eq. (10.3) 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. (10.4) to achieve good performance, including any number of highly specialized procedures developed for training neural networks [Dah+23].

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.*

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. 10.1 and 10.2 for illustrations of these behaviors.

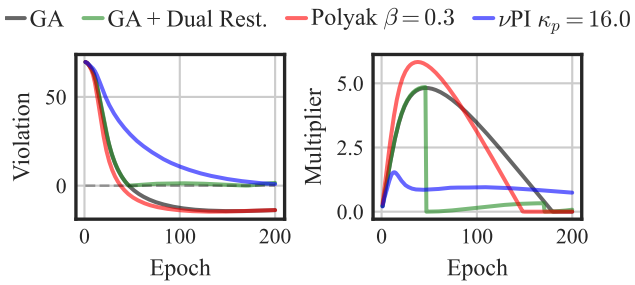


Figure 10.2: Constraint dynamics for GA, Polyak and νPI in a sparsity task (§10.5.3). Constrained optimal solutions for this problem lie at the boundary of the feasible set. The excessive growth in the value of the multiplier for GA causes the constraint to overshoot into the interior of the feasible 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 10.4 and 10.5 provide a *negative* answer to this question.

Dual restarts. Gallego-Posada et al. [Gal+22] 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. 10.1 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 §10.4, we show that νPI 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.

10.4 νPI control for constrained optimization

Following Stooke et al. [SAA20], we consider the learning of an optimal feasible model solving Eq. (10.1) 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 $\mathfrak{L}(\cdot, \boldsymbol{\theta}t)$. Fig. 10.3 illustrates the control pipeline we consider throughout this work.

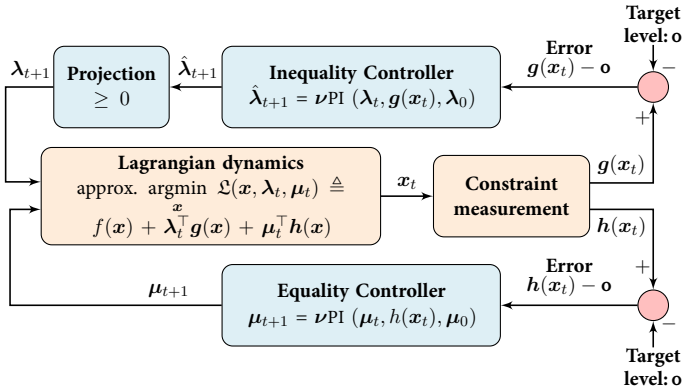


Figure 10.3: νPI control pipeline for updating the Lagrange multipliers in a constrained optimization problem. We consider the update on the primal variables as a black-box procedure that receives the multipliers $\boldsymbol{\lambda}_t$ and primal variables $\boldsymbol{x}_{t-1}$ as input, and returns 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. 10.3 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 §10.3).

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.

10.4.1 *νPI algorithm*

Our main algorithmic contribution is the multiplier update scheme presented in Algo. 3. This is a simple generalization of a PI controller (i.e. a PID controller [ÅH95] 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 3 $\boldsymbol{\nu}$PI update

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

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

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

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

The $\boldsymbol{\nu}$PI update can be equivalently expressed in terms of a recursive update (see Thm. D.2.1 in Appx. D.2) as:

$$\boldsymbol{\theta}_1 = \boldsymbol{\theta}_0 + \kappa_i\boldsymbol{e}_0 + \kappa_p\boldsymbol{\xi}_0 \tag{10.5}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \kappa_i\boldsymbol{e}_t + \kappa_p\left(\boldsymbol{\xi}_t - \boldsymbol{\xi}_{t-1}\right) \quad \text{for } t \geq 1. \tag{10.6}$$

10.4.2 *Connections to optimization methods*

When the error signal corresponds to the negative gradient of a cost function $\boldsymbol{e}_t = -\nabla f_t$, Algo. 3 has straightforward equivalences with common minimization methods. For example, $\boldsymbol{\nu}$PI ($\nu = 0, \kappa_p = 0, \kappa_i$) is equivalent to GD ($\alpha = \kappa_i$) [SAA20; LRP16; An+18]. When $\nu = 0$ and $\kappa_p = \kappa_i = \alpha$, $\boldsymbol{\nu}$PI recovers a single-player version of the Optimistic-Gradient (OG) method [Pop80], 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. [MOP20b]. Since we use $\boldsymbol{\nu}$PI for updating the multipliers, we phrase the updates in Algo. 3 based on a maximization convention.

Moreover, our proposed algorithm **$\boldsymbol{\nu}$PI generalizes popular momentum methods** such as Polyak—also known as HeavyBall—[Pol64] and Nesterov [Nes83]*. This con-

* *We consider a variant of the Nesterov method that uses a constant momentum coefficient.*

nection, stated formally in Thm. 10.4.1, will allow us to understand (Section 10.4.3) 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. [She+18] 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 4 UnifiedMomentum update [She+18]

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 10.4.1 [Proof in Appx. D.2.] *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:*

$$\nu \leftarrow \beta \qquad \boldsymbol{\xi}_0 \leftarrow (1-\beta)\boldsymbol{e}_0 \tag{10.7}$$

$$\kappa_i \leftarrow \frac{\alpha}{1-\beta} \qquad \kappa_p \leftarrow -\frac{\alpha\beta}{(1-\beta)^2}\left[1 - \gamma(1-\beta)\right]. \tag{10.8}$$

Table D.1 in Appx. D.2 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.

Fig. 10.4 emphasizes the greater generality of $\boldsymbol{\nu}$PI compared to Polyak and Nesterov, presented in Thm. 10.4.1.

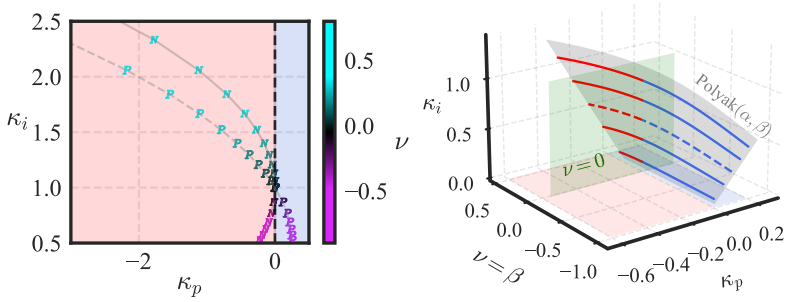


Figure 10.4: **Left:** Hyperparameter choices from Thm. 10.4.1 for which $\boldsymbol{\nu}$PI $(\nu, \kappa_p, \kappa_i)$ realizes Polyak $(\alpha = \frac{1}{2}, \beta)$ and Nesterov $(\alpha = \frac{1}{2}, \beta)$. **Right:** The right plot zooms on the range $-1 \leq \beta \leq 0.25$. **Polyak comprises 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.

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

$$\kappa_p^{\text{POLYAK}} = -\frac{\alpha\beta}{(1-\beta)^2}, \quad \kappa_p^{\text{NESTEROV}} = -\frac{\alpha\beta^2}{(1-\beta)^2} \leq 0. \tag{10.9}$$

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. [Gid+19a].

10.4.3 *Interpreting the updates of νPI*

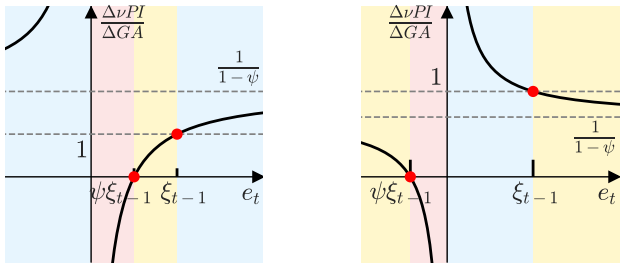
[†] *It is sufficient to consider a single scalar multiplier since the updates of both algorithms decouple across constraints/multipliers.*

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

$$\frac{\Delta\boldsymbol{\nu}\text{PI}}{\Delta\text{GA}} \triangleq \frac{\theta_{t+1}^{\nu\text{PI}} - \theta_t}{\theta_{t+1}^{\text{GA}} - \theta_t} = \frac{1}{1-\psi}\left[1 - \frac{\psi\xi_{t-1}}{e_t}\right], \tag{10.10}$$

where $\psi \triangleq \frac{\kappa_p(1-\nu)}{\kappa_i+\kappa_p(1-\nu)}$. Fig. 10.5 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[‡].

[‡] *The case of POLYAK with negative momentum resembles the left plot of Fig. 10.5. See Appx. D.3 for further details.*



$\nu\text{PI}(\kappa_i, \kappa_p > 0, \nu = 0)$ $\quad$ $\nu\text{PI}(\kappa_i, \kappa_p < 0, \nu)$=POLYAK$(\alpha, \beta > 0)$

Figure 10.5: 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 $\xi_{t-1} > 0$.

Consider the colored regions present in the *left* plot of Fig. 10.5:

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.

In stark contrast, Fig. 10.5 (right) shows a setting in which κ_i and κ_p have been chosen according to Thm. 10.4.1 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.

10.4.4 *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. D.4.1):

$$\begin{cases} \ddot{\boldsymbol{x}} = -\left(\nabla^2 f + \sum_{c'} \mu_{c'} \nabla^2 \boldsymbol{h}_{c'}\right) \dot{\boldsymbol{x}} - \mathcal{J}\boldsymbol{h}\dot{\boldsymbol{\mu}} & \text{(10.11a)} \\ \ddot{\boldsymbol{\mu}} = \kappa_i \mathcal{J}\boldsymbol{h}^\top \dot{\boldsymbol{x}} + \kappa_p \mathcal{J}\boldsymbol{h}^\top \ddot{\boldsymbol{x}} + \kappa_p \boldsymbol{\Xi}, & \text{(10.11b)} \end{cases}$$

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.4 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.

10.4.5 *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 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 $\boldsymbol{\nu}$PI 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 §10.5.2.

There is a predictable monotonic behavior of the damping of the system as the κ_p coefficient increases. This is illustrated in Fig. 10.9 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. 10.6. As a heuristic to tune $\boldsymbol{\nu}$PI, 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.

10.5 EXPERIMENTS

In this section, we present an empirical comparison between $\boldsymbol{\nu}$PI and a series of baseline optimization methods popular for minimization. We consider gradient ascent, gradient ascent with positive [Pol64; Nes83] and negative [Gid+19a] momentum, and Adam [KB15]. The goal of our experiments is to highlight the flexibility of $\boldsymbol{\nu}$PI and its ability to mitigate oscillations and overshoot when used to optimize Lagrange multipliers.

Our implementations use PyTorch [Pas+19] and the Cooper library for Lagrangian constrained optimization [Gal+24].

10.5.1 *Hard-margin SVMs*

We consider solving a *hard-margin* 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

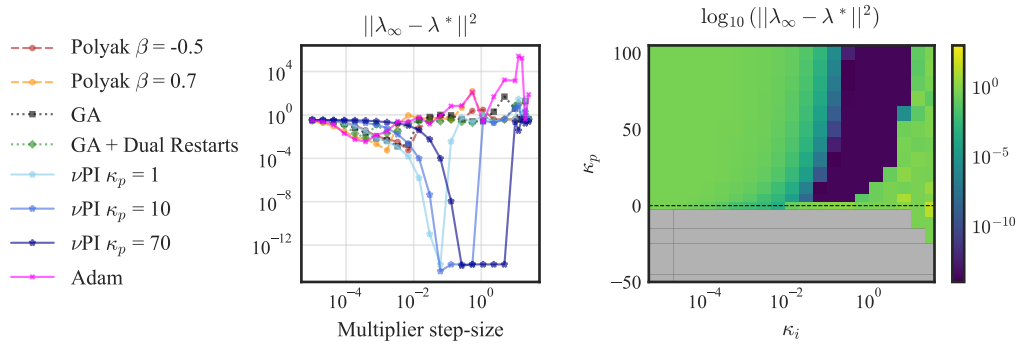
$$\min_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{w}\|^2 \quad \text{s.t.} \quad y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1 \text{ for } i \in [m], \tag{10.12}$$

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 [Fis88]. We apply alternating GDA updates on the Lagrangian associated with Eq. (10.12), with a fixed optimizer for the primal variables. For details on our SVM experiments, see Appx. D.6.1.

Multiplier dynamics. Fig. 10.1 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 $\boldsymbol{\nu}$PI algorithm. In contrast to all baselines, $\boldsymbol{\nu}$PI dampens the oscillations and converges to the optimal multiplier value.

Sensitivity analysis. Fig. 10.6 (left) illustrates the robustness of $\boldsymbol{\nu}$PI 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 in-



(a) Step-size sensitivity plot.

(b) κ_p and κ_i sensitivity plot.

Figure 10.6: **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_is.** 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^*.

crease 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). Morevoer, 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. 10.6 (right).

10.5.2 *Fairness*

We consider a classification task under *statistical parity* constraints, as described in Cotter et al. [Cot+19b]. This leads to the following constrained optimization problem:

$$\min_{\boldsymbol{w}} L(\boldsymbol{w}) \;\text{ s.t. }\; \mathrm{P}(\hat{y} = 1 \,|\, g) = \mathrm{P}(\hat{y} = 1), \quad \forall g \in G \tag{10.13}$$

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 [BK96]. 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. [Zaf+19] and Cotter et al. [Cot+19b]. However, in our setting, non-convexity precludes the use of specialized solvers (as done by Zafar et al. [Zaf+19]) 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 10.7 includes training curves for experiments with GA and νPI. We report two of the multipliers, the model accuracy, and the maximum constraint violation (in absolute value).

For this task, GA is a strong baseline as it successfully reduces the violation of the constraints. Both GA and νPI ($\nu = 0.99$) significantly improve compared to an *unconstrained* baseline which achieves a maximum violation of 20% (not shown in Fig. 10.7 for readability).

νPI ($\nu = 0$) 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, **νPI ($\nu = 0.99$) 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. 10.7, νPI yields multipliers that stabilize at their limiting values faster than those produced by GA.

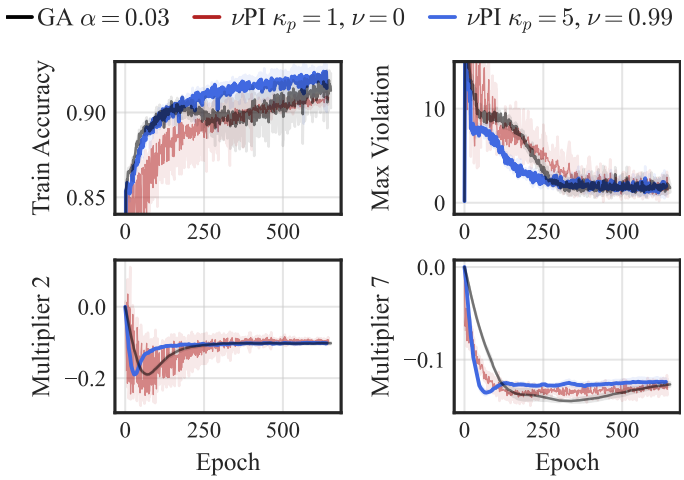


Figure 10.7: Dynamics of νPI compared to GA for the fairness task in Eq. (10.13). **νPI has faster multiplier convergence and achieves a better training accuracy than GA.** All dual optimizers use a step-size (κ_i for νPI) of 0.03. Results are aggregated across five seeds.

10.5.3 *Sparsity*

We consider the problem of learning models under inequality L_0-sparsity constraints [LWK18; Gal+22]. See Appx. D.6.2 for further background.

$$\min_{\boldsymbol{w},\boldsymbol{\phi}\in\mathbb{R}^d} \mathbb{E}_{\boldsymbol{z}|\boldsymbol{\phi}}\left[L(\boldsymbol{w}\odot\boldsymbol{z}\,|\,\mathcal{D})\right] \quad \text{s.t.} \quad \frac{\mathbb{E}_{\boldsymbol{z}|\boldsymbol{\phi}}[\|\boldsymbol{z}\|_0]}{\#(\boldsymbol{w})} \leq \epsilon \tag{10.14}$$

When using GA updates for the multipliers, Gallego-Posada et al. [Gal+22] 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 νPI on the sparsity-constrained task, and compare it with dual restarts [Gal+22, §10.3]. Our results show that νPI allows for fine-grained control over overshoot, thus enabling the sparse model to retain as much performance as possible.

Experiment configuration and hyperparameters. We classify CIFAR-10 [Kri09] images with ResNet-18 [He+16] models. To highlight the ease-of-use of νPI, our setup remains as close as possible to Gallego-Posada et al. [Gal+22]: 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 ϕ. 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. D.6.2. For comprehensive experimental results across multiple sparsity levels, and ablations on the use of momentum and Adam for updating the multipliers, see Appx. D.7.1.

Results. Fig. 10.8 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.

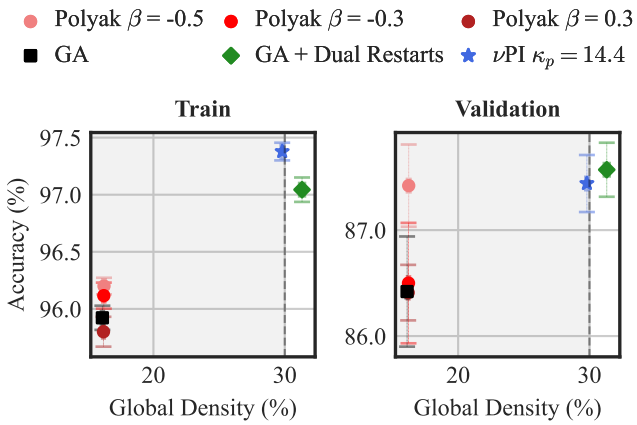


Figure 10.8: CIFAR10 trade-off plot for *global* sparsity under a 30% density target. **νPI successfully achieves the desired sparsity while achieving the highest train accuracy.** The shaded region is the feasible set. As higher density correlates to higher train accuracy, overshooting to a lower density is undesirable. All optimizers use the same step-size.

Fig. 10.9 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).

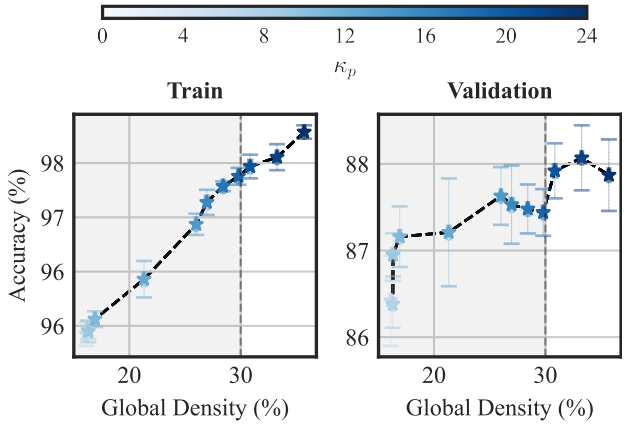


Figure 10.9: Ablation of κ_p values for νPI on CIFAR10. **An increasing κ_p leads to more damping and less overshoot.** Target density is 30%. The shaded region is the feasible set.

Table 10.1 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.

Table 10.1: 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

Multiplier dynamics. Figure 10.2 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 overshoot.

10.6 Conclusion

Previous work has highlighted that employing PID controllers on the multipliers of Lagrangian constrained optimization problems reduces oscillation and overshoot. In this paper, we consider $\boldsymbol{\nu}$PI, 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 insights apply to general min-max games where one of the players is linear. Investigating the behavior of $\boldsymbol{\nu}$PI 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.

Reproducibility Statement

We provide our code*, including scripts to replicate the experiments in this paper. Section 10.4.5 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. D.6. Our implementations use the open-source libraries PyTorch [Pas+19] and Cooper [Gal+24].

* *Available at:* https://github.com/motahareh-sohrabi/nuPI

Prologue to the Fifth Contribution

11

Article Details

Jose Gallego-Posada*, Juan Ramirez*, Meraj Hashemizadeh and Simon Lacoste-Julien. **Cooper: Constrained Optimization for Deep Learning**. This paper is under submission at *MLOSS@JMLR*, 2024.

* *Equal contribution.*

Author Contributions

Jose Gallego-Posada proposed the initial idea of developing a library for constrained optimization geared towards deep learning tasks. Jose Gallego-Posada and Juan Ramirez are the main developers for the code and documentation. Meraj Hashemizadeh contributed to the API design and the implementation of systematic tests. Jose Gallego-Posada and Juan Ramirez led the writing of the paper. Simon Lacoste-Julien provided funding and supervision during the execution of research projects that nurtured the development of the library.

Context

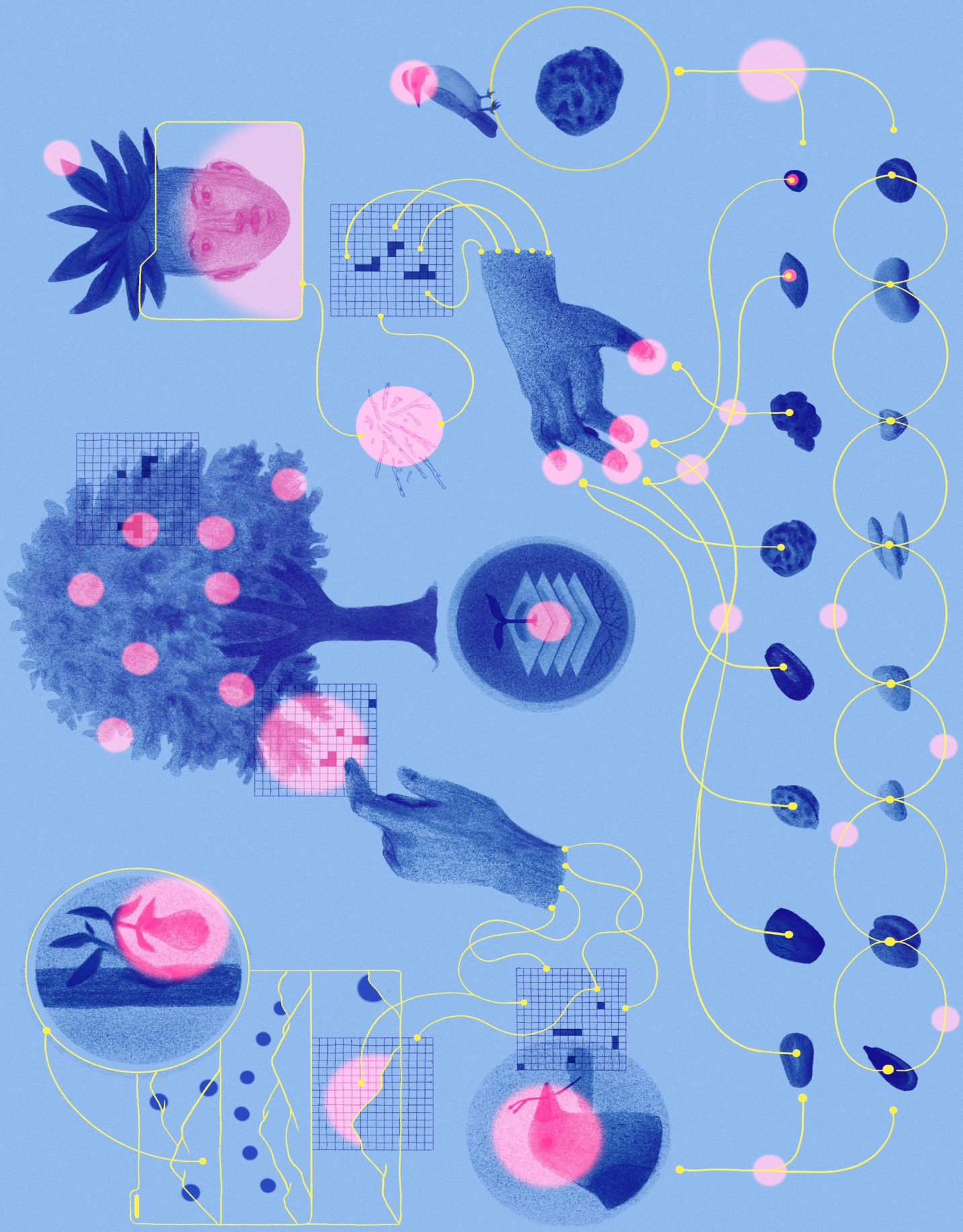
We decided to develop the open-source `Cooper` library as a complement to the paper presented in Chapter 4. While the main contribution of our NeurIPS2022 paper is to demonstrate the effectiveness of the constrained approach† for training sparse models, we were convinced that these benefits were not restricted to that specific task. The main goal of `Cooper` is to facilitate the adoption of constrained optimization techniques in the training of deep learning models, be it for research or applied projects.

† *Specially when compared to penalty-based methods.*

The experiments in the NeurIPS paper required the implementation of a pipeline for tackling constrained optimization problems in a way that was suitable for deep learning model training, and aligned with the `forward`, `backward`, and `step` pattern common within the PyTorch framework. Existing packages for constrained optimization were not applicable to the non-convex, high-dimensional, and non-smooth problems that arise in deep learning.

While some algorithms such as gradient descent-ascent are relatively straightforward, `Cooper` implements (and unit-tests!) more complex update schemes and includes several advanced features that enable efficiently tackling problems with large numbers of (potentially non-differentiable) constraints. `Cooper` can serve as a unifying‡ framework to enhance the reproducibility and ease of comparison between research projects on constrained optimization for machine learning.

‡ *Much like how PyTorch provides the community with standardized and tested implementations of neural network layers and optimizers.*



12 Cooper: Constrained Optimization for Deep Learning

Abstract

`Cooper` is an open-source package for solving constrained optimization problems involving deep learning models. `Cooper` implements several Lagrangian-based first-order update schemes, making it easy to combine constrained optimization algorithms with high-level features of PyTorch such as automatic differentiation, and specialized deep learning architectures and optimizers. Although `Cooper` is specifically designed for deep learning applications where gradients are estimated based on mini-batches, it is suitable for general non-convex continuous optimization. `Cooper`'s source code is available at https://github.com/cooper-org/cooper.

12.1 Introduction

The rapid advancement and widespread adoption of algorithmic decision systems, such as large-scale machine learning models, have generated significant interest from academic and industrial research organization in enhancing the robustness, safety, and fairness of these systems. These research efforts are typically driven by governmental regulations [Cou24] or ethical considerations [DAV18].

The ability to enforce complex behaviors in machine learning models is a central component for ensuring compliance with the mentioned regulatory and ethical guidelines. Constrained optimization offers a rigorous conceptual framework accompanied by algorithmic tools for reliably training machine learning models that satisfy the desired requirements. These requirements can often be formally encoded as numerical (equality or inequality) constraints accompanying the training objective of the model:

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) \quad \text{subject to} \quad \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0} \quad \text{and} \quad \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}. \tag{12.1}$$

For example, Dai et al. [Dai+24] successfully leverage a constrained optimization approach for striking a balance between the helpfulness, harmfulness and willingness-to-respond of large language models trained with reinforcement learning from human feedback [Chr+17; Ouy+22]. Other works have demonstrated the benefits of constrained optimization techniques in fairness [Cot+19b; Has+24], safe reinforcement learning [SAA20], active learning [ENR22] and model quantization [HER23]. Our previous work has highlighted the tunability advantages of constrained optimization over penalized formulations (where regularizers are incorporated as penalties in the objective function) for training sparse models [Gal+22]

This paper presents `Cooper`, a library for solving constrained optimization problems with PyTorch [Pas+19]. `Cooper` aims to facilitate the use of constrained optimization methods in machine learning research and applications. It implements several first-order update schemes for Lagrangian-based constrained optimization, along with specialized features for tackling problems with large numbers of (possibly non-differentiable) constraints. `Cooper` benefits from PyTorch for efficient tensor computation and automatic differentiation.

Key differentiators

`Cooper` is a general-purpose library for non-convex constrained optimization, with a strong emphasis on deep learning. In particular, `Cooper` has been designed with native support for the framework of stochastic first-order optimization using mini-batch estimates that is prevalent in the training of deep learning models.

`Cooper`'s Lagrangian-based approach makes it suitable for a wide range of applications. However, some optimization problems enjoy special structure and admit specialized optimization algorithms with enhanced convergence guarantees. We recommend the use of `Cooper` *unless* specialized algorithms are available for a given application.

Existing constrained optimization libraries

* *Not actively maintained at the time of writing.*

A notable precursor of `Cooper` is TensorFlow's TFCO* [Cot+]. We developed `Cooper` in response to the shift of the machine learning research community towards PyTorch. `Cooper` is heavily inspired by the design of TFCO.

Among the most popular alternatives for *convex* constrained optimization, we highlight the CVXPY library [DB16]. CVXPY provides a modeling language for disciplined convex programming in Python and automates the transformation of the problem into a canonical form, before executing open-source or commercial solvers. CVXPY is not focused on non-convex problems and thus not suitable for deep learning applications.

GeoTorch [Lez21] and CHOP [NP20] are alternatives for constrained optimization in PyTorch. JAXopt [Blo+22] is a JAX-based option. These libraries rely on the existence of efficient projection operators, linear minimization oracles, or specific manifold structure in the constraints—whereas `Cooper` is more generic and does not rely on these specialized structures.

Impact

`Cooper` has enabled several papers published at top machine learning conferences: Gallego-Posada et al. [Gal+22], Ramirez and Gallego-Posada [RG22], Zhu et al. [Zhu+23], Hashemizadeh et al. [Has+24], Sohrabi et al. [Soh+24], Lachapelle et al. [Lac+24], and Jang et al. [Jan+24].

12.2 ALGORITHMIC OVERVIEW

Problem formulation

Constrained optimization problems involving the outputs of deep neural networks are typically non-convex. A general approach to solving non-convex constrained problems

is finding a min-max point of the Lagrangian associated with the optimization problem:

$$\min_{\boldsymbol{x}} \max_{\boldsymbol{\lambda} \geq \mathbf{0}, \boldsymbol{\mu}} \mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \triangleq f(\boldsymbol{x}) + \boldsymbol{\lambda}^\top \boldsymbol{g}(\boldsymbol{x}) + \boldsymbol{\mu}^\top \boldsymbol{h}(\boldsymbol{x}), \tag{12.2}$$

where $\boldsymbol{\lambda} \geq \mathbf{0}$ and $\boldsymbol{\mu}$ are the Lagrange multipliers for the inequality and equality constraints, respectively. Solving the min-max problem in Eq. (12.2) is equivalent to the original problem in Eq. (12.1), *even if some of the functions are non-convex*. We refer the interested reader to the works by Platt and Barr [PB87], Boyd and Vandenberghe [BV04], Nocedal and Wright [NW06], and Bertsekas [Ber16] for comprehensive overviews on the theoretical and algorithmic aspects of constrained optimization.

Update schemes

`Cooper` implements several variants of (projected) gradient descent-ascent (GDA) to solve Eq. (12.2). The simplest approach is simultaneous GDA:

$$\boldsymbol{x}_{t+1} \leftarrow \texttt{PrimalOptimizerStep}\big(\boldsymbol{x}_t, \nabla_{\boldsymbol{x}} \mathfrak{L}(\boldsymbol{x}_t, \boldsymbol{\lambda}_t, \boldsymbol{\mu}_t)\big), \tag{12.3a}$$

$$\boldsymbol{\lambda}_{t+1} \leftarrow \big[\texttt{DualOptimizerStep}\big(\boldsymbol{\lambda}_t, \boldsymbol{g}(\boldsymbol{x}_t)\big)\big]_+, \tag{12.3b}$$

$$\boldsymbol{\mu}_{t+1} \leftarrow \texttt{DualOptimizerStep}\big(\boldsymbol{\mu}_t, \boldsymbol{h}(\boldsymbol{x}_t)\big), \tag{12.3c}$$

where $[\,\cdot\,]_+$ is an element-wise projection onto $\mathbb{R}_{\geq 0}$ to ensure the non-negativity of inequality multipliers. Note that the gradients of the Lagrangian with respect to $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ simplify to $\boldsymbol{g}(\boldsymbol{x}_t)$ and $\boldsymbol{h}(\boldsymbol{x}_t)$, respectively.

Convergence properties

Recent work demonstrates that GDA can work in practice for Lagrangian constrained optimization [Gal+22; Soh+24], although it may diverge for general min-max games [Gid+19b].

Optimizers

`Cooper` allows the use of generic PyTorch optimizers to perform the primal and dual updates in Eq. (12.3). This enables the use of pre-existing pipelines for unconstrained minimization when solving constrained optimization problems using `Cooper`.

Additional features

`Cooper` implements the Augmented Lagrangian [Ber16, §5.2.2] formulation. `Cooper` also implements the proxy-Lagrangian technique of Cotter et al. [Cot+19b], which allows for solving constrained optimization problem with *non-differentiable* constraints. Moreover, `Cooper` supports alternative schemes to simultaneous GDA such as *alternating* GDA and extragradient [Kor76; Gid+19b]. Finally, `Cooper` implements the νPI algorithm [Soh+24] for improving the multiplier dynamics.

12.3 USING `cooper`

Figure 12.1 presents `Cooper`'s main classes. The user implements a `ConstrainedMinimizationProblem` (`CMP`) holding `Constraint` objects, each in turn holding a corresponding `Multiplier`. The `CMP`'s `compute_cmp_state` method returns the objective value and constraints violations, stored in a `CMPState` dataclass. `CooperOptimizers` wrap the primal and dual optimizers and perform updates (such as simultaneous GDA). The `roll` method of `CooperOptimizers` is a convenience function to (i) perform a `zero_grad` on all optimizers, (ii) compute the Lagrangian, (iii) call its `backward` and (iv) perform the primal and dual optimizer steps.

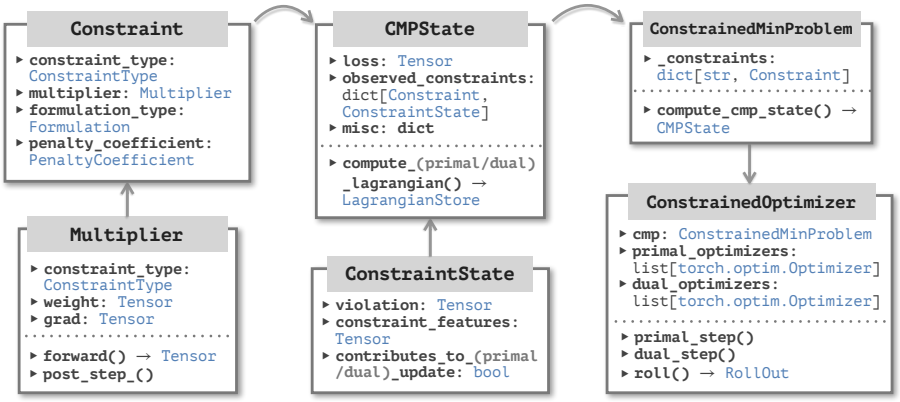


Figure 12.1: Dependency graph between the main classes in `Cooper`'s API.

Listing 12.1 presents a code example for solving a norm-constrained logistic regression problem with `Cooper`. This code illustrates the ease of integration of `Cooper` with a standard PyTorch training pipeline involving the use of a dataloader, GPU acceleration and the Adam optimizer [KB15] for the primal parameters.

12.4 SOFTWARE OVERVIEW

Installation

`Cooper` can be installed in Python 3.9-3.11 via `pip install cooper-optim`. It is supported on Linux, macOS and Windows. `Cooper` is compatible with PyTorch 1.13-2.3.

Collaboration and code quality

`Cooper` is hosted on GitHub under an MIT open-source license. We welcome external contributions that comply with `Cooper`'s contribution guide. We make extensive use of type-hints and apply automatic formatting using Black [Lan+18] and isort [Cro+13]. We ensure PEP8 compliance using Flake8 [Zia+11]. Continuous integration practices are in place to ensure that new contributions pass all tests and comply with the style guidelines before being merged.

All new contributions are expected to be tested following the contribution guidelines. For instance, every optimization scheme counts with both low-level tests ensuring that individual updates are performed correctly, and high-level tests on convex problems checking convergence to verified solutions*. The line coverage of our tests is above 90%.

* *We rely on CVXPY [DB16] to obtain solutions with optimality certificates.*

Listing 12.1: Example code for solving a norm-constrained logistic regression task using `Cooper`.

```python
import cooper
import torch

train_loader = ... # Create a PyTorch DataLoader
loss_fn = torch.nn.CrossEntropyLoss()

class NormConstrainedLogisticRegression(cooper.ConstrainedMinimizationProblem):
    def __init__(self, norm_threshold: float):
        self.norm_threshold = norm_threshold
        constraint_kwargs = {"constraint_type": cooper.ConstraintType.INEQUALITY}
        multiplier = cooper.multipliers.DenseMultiplier(device=DEVICE, **constraint_kwargs)
        # By default constraints are built using `formulation_type=cooper.LagrangianFormulation`
        self.norm_constraint = cooper.Constraint(multiplier=multiplier, **constraint_kwargs)

    def compute_cmp_state(self, model, inputs, targets) -> cooper.CMPState:
        logits = model.forward(inputs.view(inputs.shape[0], -1))
        loss = loss_fn(logits, targets)

        sq_l2_norm = model.weight.pow(2).sum() + model.bias.pow(2).sum()
        # Constraint violation uses the convention "g - \epsilon \leq 0"
        norm_constraint_state = cooper.ConstraintState(violation=sq_l2_norm - self.norm_threshold)

        # The `misc` field can be used to store any additional information
        misc = {"batch_accuracy": ...}

        # Declare observed constraints and their measurements
        observed_constraints = {"norm_constraint": norm_constraint_state}

        return cooper.CMPState(loss=loss, observed_constraints=observed_constraints, misc=misc)

cmp = NormConstrainedLogisticRegression(norm_threshold=1.0)

# Create a Logistic Regression model and primal and dual optimizers
model = torch.nn.Linear(in_features=IN_FEATURES, out_features=NUM_CLASSES, bias=True)
model = model.to(DEVICE)
primal_optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
# Must set `maximize=True` since the Lagrange multipliers solve a _maximization_ problem
dual_optimizer = torch.optim.SGD(cmp.dual_parameters(), lr=1e-2, maximize=True)

cooper_optimizer = cooper.optim.SimultaneousOptimizer(
    cmp=cmp, primal_optimizers=primal_optimizer, dual_optimizers=dual_optimizer
)

for epoch_num in range(NUM_EPOCHS):
    for inputs, targets in train_loader:
        inputs, targets = inputs.to(DEVICE), targets.to(DEVICE)

        # `roll` is a(n optional) convenience function that packages together the evaluation
        # of the loss, call for gradient computation, the primal and dual updates and zero_grad
        compute_cmp_state_kwargs = {"model": model, "inputs": inputs, "targets": targets}
        roll_out = cooper_optimizer.roll(compute_cmp_state_kwargs=compute_cmp_state_kwargs)
        # `roll_out` is a struct containing the loss, last CMPState, and the primal
        # and dual Lagrangian stores, useful for inspection and logging

torch.save(model.state_dict(), 'model.pt') # Regular  model checkpoint
torch.save(cmp.state_dict(), 'cmp.pt') # Checkpoint for multipliers and penalty coefficients
torch.save(cooper_optimizer.state_dict(), 'cooper_optimizer.pt') # PyTorch optimizer states
```

Documentation

[†] *Available at* https://cooper.readthedocs.io

`Cooper` provides extensive documentation[†] for all features. We include formal statements of the updates implemented by all optimizers along with bibliographical references to relevant sources. We provide quick-start guides aimed at i) users familiar with deep learning problems, and ii) to a broader audience of users interested in generic non-convex constrained optimization problems. Additionally, we have made available several well-documented tutorials illustrating the use of `Cooper`'s core features.

12.5 Conclusion

`Cooper` provides tools for solving constrained optimization problems in PyTorch. The library supports several Lagrangian-based first-order update schemes and has been successfully used in machine learning research projects. The structure of `Cooper` allows for easy implementation of new features such as alternative problem formulations, implicitly parameterized Lagrange multipliers, and additional `ConstrainedOptimizer` wrappers. Implementing a version of `Cooper` for JAX [Bra+18] constitutes promising future work.

Part III

CONCLUSIONS & PERSPECTIVES

Conclusions and Perspectives 13

In this thesis, we have presented an extensive study on the use of constrained optimization in machine learning. This section summarizes our main positions and contributions, and discusses limitations, along with directions for future research.

Why do we need constrained optimization in machine learning?

The machine learning community has made significant progress on building increasingly capable models during the last decade. As a result, research and regulatory interests have expanded towards ensuring that, besides achieving good performance, models are also safe, fair, and robust. However, in Chapter 1 we argued that the *"build now, fix later"* mentality that is currently widespread in the development of machine learning models is a hindrance to the-long term progress of the field.

The constrained approach allows incorporating application requirements directly into the optimization pipeline in a way that enhances experimental accountability: only solutions that satisfy the constraints are considered valid. Rather than aiming for models that *happened* to satisfy the constraints or regulations, we train models that are *designed* to respect them. The constrained optimization paradigm enables service providers to demonstrate regulatory compliance to stakeholders or enforcement agencies.

However, lacking readily available techniques for incorporating problem requirements during the model development process, the community has become overly reliant on penalized approaches.

How are constrained methods better than penalized approaches?

Penalized formulations have several shortcomings when handling non-convex problems with interpretable requirements.

Tuning and interpretability. Adjusting the penalty parameter to achieve pre-specified trade-off levels is a non-trivial task. As we saw in Chapter 4, it is difficult to quantify the influence of the penalty parameter in terms of the sparsity of the model at the end of training. Given a problem requirement with clear semantics such as the maximum allowable model sparsity, opting for a penalized approach leads to (unnecessary) sacrifices in hyper-parameter interpretability, and leads to expensive trial-and-error tuning.

In contrast, the problem specification (e.g. the desired sparsity) corresponds in a direct and straightforward manner to the constraint level in the constrained approach. Although the dual step-size is an additional hyper-parameter of gradient-based Lagrangian techniques, we highlight that it is typically more robust to tune than the penalty parameter itself. For example, in Fig. 4.1 we showed how the same value of the dual-step size was effective for different desired sparsity levels. Conversely, the penalty parameter requires challenging individual tuning for each target sparsity level.

Exploring trade-offs. In non-convex problems, the penalized approach is limited in its capacity to explore the entire Pareto frontier. As we discussed in §2.2.4, for problems with a strictly positive duality gap*, there are points on the Pareto front of the problem which cannot be reached by minimizing the penalized objective with any *fixed* choice of penalty parameter.

* *As is the case in most non-convex problems.*

In the constrained setting, the Lagrange multipliers weighing the constraints are not fixed, but rather are decision variables that adjust dynamically throughout the optimization process to encourage the satisfaction of the constraints. This fundamental difference allows the constrained approach to explore regions of the Pareto frontier that are not reachable via penalized techniques.

Computational overhead. As mentioned in Sections 2.3.3 and 2.3.4, appropriate implementations of Lagrangian methods† involve negligible computational overhead compared to the penalized approach.

† *Such as those in the* `Cooper` *library (Chapter 12)*

Considering the high cost and heavy engineering involved in the training of large-scale deep learning models, and in light of the discussed advantages of constrained methods, we argue that it is a sub-optimal experimental strategy for researchers and practitioners to focus *solely* on penalized approaches.

When are constrained methods **not** better than penalized approaches?

Our discussion of the advantages of constrained approaches relied heavily on the assumption that the problem requirements are interpretable. In other words, we assumed that the constraints have clear semantics, and that we have access‡ to the values of the constraint levels that are meaningful for the application.

‡ *Through domain knowledge or expert input.*

Consider the case where the constraint functions do not have interpretable semantics. For example, the L_2-norm is a popular penalty which has been show to correlate with the ability of deep learning models to generalize [Ney+17]. While the L_2-norm has a straightforward mathematical definition, it is not at all clear what a specific value of the L_2-norm of the model parameters *means* in terms of the behavior of the model.

As another example, Bardes et al. [BPL22] introduce VICReg, a method for preventing *collapse* in self-supervised learning. The authors propose a penalized objective that encourages the model to learn representations with sufficient per-dimension variance, and low covariance across dimensions. As in the previous setting, it is difficult to provide a clear interpretation to the *values* of the regularizers§ and how said values relate to the overall model performance.

§ *Beyond their mathematical definition.*

In situations like these, using a constrained formulation may not be appropriate. Our lack of understanding of the semantics of the constraints makes it is difficult to set relevant constraint levels. Should the L_2-norm of the model parameters be at most 1.7, 10^4, or $3.8 \cdot 10^6$? Another challenge in these settings is establishing whether the constrained problem to be formulated would be feasible. For example, in the case of VICReg, it is not clear whether the model class has enough capacity to learn representations with *both* a prescribed minimum threshold of per-dimension variance *and* a prescribed maximum of covariance level across dimensions.

Experimental design. A case of practical importance is when the penalty function is a *heuristic* surrogate h for a downstream relevant property H. Suppose that we would like to understand whether using h is a good regularizer for enforcing H. In this setting, *constrained optimization can be a useful tool for experimental design.*

Naively, we could select L penalty coefficients $\{\lambda_l\}$ and solve L penalized problems, leading to models $\{\boldsymbol{x}_l\}$ with objective-penalty trade-offs $\{(f(\boldsymbol{x}_l), h(\boldsymbol{x}_l))\}$. Finally, we can evaluate the behavior of the models at the downstream property, to obtain measurements $\{H(\boldsymbol{x}_l)\}$. A shortcoming of this scheme, inheriting from the tunability challenges of penalized approaches (see Chapter 4), is that we have no direct control over the achieved levels $\{h(\boldsymbol{x}_l)\}$.

In contrast, the constrained approach allows us to directly specify desired values of the heuristic $\{\hat{h}_l\}$ as constraint levels in L constrained problems, thus providing more fine-grained control over the experimental design.

A protocol similar to this was used by Lachapelle et al. [Lac+24] when studying the effect of mechanism sparsity for learning disentangled representations.

What are the main practical challenges when solving constrained optimization problems in deep learning?

Optimization dynamics. Throughout all our contributions, we payed special attention to the optimization dynamics of the Lagrangian problem. Given the heavily engineered protocols for training deep learning models, we opted for optimization schemes that were compatible with adaptive optimization techniques like Adam [KB15] or heuristic tuning methods like learning rate schedules. While more restrictive than generic min-max optimization techniques, this choice enabled us to successfully reuse the (frequently pre-existing) choice of primal optimizer from the unconstrained setting.

This restricted scope also allowed us to concentrate on developing algorithmic innovations for resolving some of the shortcomings of the plain gradient-ascent updates on the Lagrange multipliers.[¶] In particular, we showed how techniques like dual restarts (Chapter 4) and the $\boldsymbol{\nu}$PI algorithm (Chapter 10) improved the training dynamics of the multipliers, leading to more stable optimization and better-performing models.

[¶] *Such as their excessive growth, which caused constraint overshoot.*

Non-differentiable constraints. In Chapters 6 and 8 we successfully tackled problems which involved non-differentiable constraints on model efficiency and pruning-induced disparity, respectively. While in many cases the "true" constraints that are practically relevant for a problem may be non-differentiable, a differentiable surrogate[⊠] is typically available. For example, Chapter 6 we saw how a non-differentiable constraint on the number of active parameters in a sparse model (after binarization of the stochastic gates) admit a differentiable surrogate in terms of the *expected* number of active parameters. The proxy-constraint technique proposed by Cotter et al. [Cot+19b][**] provides a protocol for handling non-differentiable constraints by replacing them with a differentiable surrogate *only when algorithmically necessary*.

[⊠] *Whose gradient "points in the right direction" for improving the non-differentiable constraint.*

[**] *Discussed in Eq.* (2.34).

Generalization. An important distinction between constrained and unconstrained problems in deep learning is the appearance of a new axis of generalization. While in single-objective minimization problems we are only concerned with the ability of the

model to generalize to unseen data in terms of the loss, in constrained problems we must also consider the ability of the model to satisfy the constraints on unseen data. To mitigate the risk of constraint overfitting, Cotter et al. [Cot+19a] proposed a "two-dataset" approach in which the constraint satisfaction is evaluated on a separate validation set.

Feasibility. The gradient-based Lagrangian approaches we explore in this thesis are not *feasible methods*, i.e. the iterates visited during optimization are not guaranteed to be feasible. Thus, it is possible that the "solution" found at the end of training is not feasible.

A natural idea to handle this issue would be to incorporate a "feasibility restoration" phase into the execution of the algorithm [NW06, §15.4]. However, in machine learning, the use of a feasibility restoration phase could result in a catastrophic degradation of the model's ability to learn. For example, in a sparsity task with an aggressive sparsity level[††], triggering a restoration phase early in training could drastically reduce the model's capacity.

[††] *For example, leaving 1-5% active parameters.*

Since in the deep learning context, models are typically trained using a pre-specified number of epochs (rather than using a stopping condition based, for example, on the norm of the gradient of the objective), we believe techniques for annealing the constraint levels during training, which we refer to as *constraint schedulers*, are a promising direction for future research.

What are the implications of this thesis for researchers in other fields?

Although the research presented in this thesis was mostly developed in an academic machine learning environment, we believe the fundamental ideas are relevant to other contexts in which companies and organizations require compliance with constraints. As we argued in Chapter 1, satisfying domain-specific constraints can be a deciding factor for determining whether a machine learning model is suitable for deployment in a real-world application.

We hope this dissertation can serve as a catalyst for more interdisciplinary research between the machine learning community and other fields. While machine learning researchers and practitioners bring crucial expertise in developing machine learning models, experts in the domain of application are essential for defining the constraints that are relevant to the problem or industry at hand.

[‡‡] *Research question.*

RQ:[‡‡] How to deal with constraints that are difficult to quantify?

A key limitation of the constrained framework presented in this thesis is the implicit assumption that the relevant problem requirements can be quantified and represented numerically as constraint functions.

Already techniques like reinforcement learning from human feedback [Chr+17; Ouy+22] aim to "align" the behavior of large language models with human preferences, which are not easily quantifiable. Imminently, we are bound to face situations where the constraint functions, rather than being hard-coded by an experimenter, are *learned* from data.

RQ: Why do GDA-like schemes work in practical Lagrangian problems?

While theoretical results such as the work of Lin et al. [LJJ20] provide encouraging evidence for the convergence of GDA in idealized settings, the practical success of GDA for solving min-max Lagrangian problems remains a mystery. Why, despite all the potential instabilities warned by theory, do we observe good performance in practice from simple techniques like (alternating) GDA in non-convex deep learning tasks?

RQ: What is the role of overparametrization in constrained optimization?

The deep learning community has observed that vast overparametrization[§§] can lead to *better* generalization performance, at odds with the traditional bias-variance trade-off narrative [Bel+19]. How crucial is overparametrization for solving constrained optimization problems involving deep learning models?

[§§] *Models with many more parameters than required to fit the training data.*

Is there any connection between the success of gradient-based techniques on non-convex overparametrized problems [Cho+15] and the success of GDA mentioned above?

RQ: How can we improve the Lagrange multiplier dynamics further?

Techniques like dual restarts (Chapter 4) and νPI (Chapter 10) can induce better training dynamics in constrained optimization problems compared to plain gradient-ascent updates. However, we believe these methods are far from optimal and a truly *adaptive* method for updating the Lagrange multipliers is yet to be developed.

On the theoretical front, could improved convergence guarantees for νPI be established?

RQ: How can we make constrained techniques usable "during inference"?

In this thesis we focused on imposing constraints during the *training* phase of the model. However, in many real-world applications, it is desirable to enforce constraints during the *inference*[¶¶] phase as well. Unfortunately, gradient-based Lagrangian techniques require many updates to the Lagrange multipliers before finding good estimates that induce constraint satisfaction. What constrained optimization techniques are most suitable for these instances?

[¶¶] *I.e., evaluating the model on a given data-point.*

RQ: What is next for `Cooper`?

While `Cooper` is currently designed around PyTorch, other popular frameworks like Jax and PyTorch Lightning are suitable for integration with `Cooper`. Re-structuring the library in a framework-agnostic way would be of great benefit.[***]

[***] *If you are interested in becoming a contributor to `Cooper`, visit* `https://github.com/cooper-org/cooper/`.

Part IV

APPENDICES

Appendix to the First Contribution

A

A.1 Reparametrization of the Gates

Louizos et al. [LWK18] introduce the hard concrete distribution for modeling the stochastic gates $\boldsymbol{z}$. Consider a concrete random variable $s_j \sim q(\cdot \mid (\phi_j, \beta))$, given a fixed $0 < \beta < 1$. This variable is then stretched and clamped, resulting in a mixed distribution with point masses at 0 and 1, and a continuous density over $(0, 1)$.

Formally, given $U_j \sim \text{Unif}(0, 1)$ and hyper-parameters $\gamma < 0 < 1 < \zeta$,

$$s_j = \text{Sigmoid}\left(\frac{1}{\beta}\log\left(\frac{\phi_j\, U_j}{1 - U_j}\right)\right); \qquad \boldsymbol{z} = \text{clamp}_{[0,1]}(\mathbf{s}(\zeta - \gamma) + \gamma)) \tag{A.1}$$

The stochastic nature of $\boldsymbol{z}$ entails a model which is itself *stochastic*. Therefore, both its L_0-norm and predictions are random quantities. Nonetheless, the specific choice of re-parameterization in Eq. (A.1) allows for the training loss in Eq. (4.2) to be estimated using Monte Carlo samples.

Moreover, Louizos et al. [LWK18] show that the expected L_0-norm can be expressed in closed-form as:

$$\mathbb{E}_{\boldsymbol{z}|\boldsymbol{\phi}}\left[\, \|\boldsymbol{\theta}\|_0\right] = \sum_{j=1}^{|\boldsymbol{\theta}|} \mathbb{P}[z_j \neq 0] = \sum_{j=1}^{|\boldsymbol{\theta}|} \text{Sigmoid}\left(\log \phi_j - \beta \log \frac{-\gamma}{\zeta}\right) \tag{A.2}$$

The probability distribution of the gates has both learnable and fixed parameters. Table A.1 specifies the values of the fixed parameters employed throughout this work, following Louizos et al. [LWK18].

Table A.1: Fixed parameters of the hard concrete distribution.

Parameter	γ	ζ	β
Value	-0.1	1.1	2/3

A.1.1 *Choice of gates at test-time*

Recall that the stochastic reparametrization induces a distribution over models. We suggest a natural way of "freezing" the network gates so as to obtain a *deterministic* predictor when evaluating the model on unseen data: replace each stochastic gate by its median.

$$\hat{z}_j = \min\left(1,\ \max\left(0,\ \text{Sigmoid}\left(\frac{\log(\phi_j)}{\beta}\right)(\zeta - \gamma) + \gamma\right)\right) \tag{A.3}$$

Note that these medians may be *fractional*, i.e., $z_j \in [0, 1]$. Nonetheless, as shown in Appx. A.9, for trained sparse models we observe the medians to be highly concentrated at 0 and 1.

LOUIZOS et al. [LWK18, Eq. (13)] originally proposed a similar approach for obtaining a deterministic test-time model (without an explicit motivation for their choice). Their proposal differs from ours in that they do not perform a division by β, and thus their test-time gates are not the median (nor the mean) of the gate distributions.

In our preliminary experiments the division by β (under the settings of Table A.1) did not induce significant changes in behavior. However, we opt for Eq. (A.3) in our experiments based on its concise theoretical motivation.

A.1.2 *Initialization of the gates*

LOUIZOS et al. [LWK18] introduce a hyper-parameter $\rho_{\text{init}} \in (0, 1)$ which determines the initialization of the parameter ϕ_j of the hard concrete distribution of the gates (see Appx. A.1). Concretely, the gate parameters ϕ_j are initialized as:

$$\log \phi_j = \log\left(\frac{1-\rho_{\text{init}}}{\rho_{\text{init}}}\right) + \mathcal{N}(0, 10^{-2}) \tag{A.4}$$

Note that in practice, the optimization variable is $\log \phi_j$ (rather than ϕ_j) as this sidesteps having to preserve the non-negativity of ϕ_j.

The choice of ρ_{init} has an inverse relationship with the probability of a gate being active at initialization. For simplicitly, we ignore the small additive noise in the initialization of $\log \phi_j$, and let $\psi = (-\gamma/\zeta)^{\beta}$. Formally, the influence of the hyper-parameter ρ_{init} at initialization is given by:

$$\mathbb{P}[z_j \neq 0] = \text{Sigmoid}\left(\log\left(\frac{1-\rho_{\text{init}}}{\rho_{\text{init}}}\right) - \log\left(\frac{-\gamma}{\zeta}\right)^{\beta}\right) = \frac{1-\rho_{\text{init}}}{1-(1-\psi)\rho_{\text{init}}}. \tag{A.5}$$

A.2 SCHEMES FOR GROUPING PARAMETERS

We consider two schemes for grouping gates: a) with a single constraint/penalty on the proportion of active gates of the model, or b) with separate constraints/penalties for each layer. These groupings are referred to as *model-wise* and *layer-wise*.

For the penalized method, the corresponding optimization problems are given by:

Model-wise grouping

$$\min_{\tilde{\boldsymbol{\theta}}, \boldsymbol{\phi}} \quad \mathfrak{f}_{\text{obj}}(\tilde{\boldsymbol{\theta}}, \boldsymbol{\phi}) + \lambda_{\text{pen}}\, \mathfrak{g}_{\text{const}}(\boldsymbol{\phi})$$

Layer-wise grouping

$$\min_{\tilde{\boldsymbol{\theta}}, \boldsymbol{\phi}} \quad \mathfrak{f}_{\text{obj}}(\tilde{\boldsymbol{\theta}}, \boldsymbol{\phi}) + \sum_{g=1}^{\texttt{num\_layers}} \lambda_{\text{pen}}^{g}\, \mathfrak{g}_{\text{const}}(\boldsymbol{\phi}_g)$$

For the constrained method, the corresponding optimization problems are given by:

Model-wise grouping	**Layer-wise grouping**
$\min_{\tilde{\boldsymbol{\theta}},\boldsymbol{\phi}} \quad \mathfrak{f}_{\text{obj}}(\tilde{\boldsymbol{\theta}},\boldsymbol{\phi})$	$\min_{\tilde{\boldsymbol{\theta}},\boldsymbol{\phi}} \quad \mathfrak{f}_{\text{obj}}(\tilde{\boldsymbol{\theta}},\boldsymbol{\phi})$
s.t. $\quad \mathfrak{g}_{\text{const}}(\boldsymbol{\phi}) \leq \epsilon$	s.t. $\quad \mathfrak{g}_{\text{const}}(\boldsymbol{\phi}_g) \leq \epsilon_g \;\; \text{for } g \in [1 : \texttt{num\_layers}]$

For example, consider a $[d_{\text{in}}, d_{\text{hid}}, d_{\text{out}}]$ 1-hidden layer MLP with input-neuron sparsity on both of its two fully connected layers. For simplicity, we ignore the bias in the description below.

- **Grouping at the layer level** (akin to "λ sep." in [LWK18]), would yield $G = 2$ groups, with d_{in} gates in group $g = 1$. Each gate in group 1 is shared across d_{hid} parameters in $\tilde{\boldsymbol{\theta}}$, thus $\#(\tilde{\boldsymbol{\theta}}_1) = d_{\text{in}} \cdot d_{\text{hid}}$. Similarly for $g = 2$.
- **Grouping at the model level** (akin to "$\lambda \propto \frac{1}{N}$" in [LWK18]). corresponds to the case of $G = 1$ group comprising with $d_{\text{in}} + d_{\text{hid}}$ gates. Finally, $\#(\tilde{\boldsymbol{\theta}}_1) = d_{\text{in}} \cdot d_{\text{hid}} + d_{\text{hid}} \cdot d_{\text{out}}$ gives the total number of parameters in the network.

A similar analysis holds for the case of output feature-map sparsity used in convolutional layers.

A.3 Normalizing the L_0-norm

Louizos et al. [LWK18] normalize the expected L_0-norm of model parameters with respect to the *training set size* N, and not with respect to the *number of parameters*. This is done by selecting a $\lambda_{\text{pen}} = \lambda/N$. In contrast, as stated in Eq. (4.3), we favor normalizing by the total number of parameters in the model $\#(\tilde{\boldsymbol{\theta}})$. This yields an expected L_0-density consistently in the range $[0,\ 1]$ regardless of model architecture.

Optimization problems corresponding to each of these normalization schemes (grouping gates model-wise, for illustration) are given by:

Ours (Penalized)	**Louizos et al. [LWK18]**
$\min_{\tilde{\boldsymbol{\theta}},\boldsymbol{\phi}} \mathfrak{f}_{\text{obj}}(\tilde{\boldsymbol{\theta}},\boldsymbol{\phi}) + \lambda_{\text{pen}} \frac{\mathbb{E}_{\boldsymbol{z}\,\mid\,\boldsymbol{\phi}}[\|\boldsymbol{z}\|_0]}{\#(\tilde{\boldsymbol{\theta}})}$	$\min_{\tilde{\boldsymbol{\theta}},\boldsymbol{\phi}} \mathfrak{f}_{\text{obj}}(\tilde{\boldsymbol{\theta}},\boldsymbol{\phi}) + \lambda_{\text{pen}} \frac{\mathbb{E}_{\boldsymbol{z}\,\mid\,\boldsymbol{\phi}}[\|\boldsymbol{z}\|_0]}{N}$

Therefore, a $\lambda_{\text{pen}} = \lambda$ in the context of our work does *not* correspond to the same regularization factor as choosing $\lambda_{\text{pen}} = \lambda/N$ in Louizos et al. [LWK18]. For details on the number of parameters of each architecture and the associated number of training examples, see Appx. A.10.1. Note that in certain settings (e.g. CIFAR-10/100) the number of training points and number of parameters of the model can differ by several orders of magnitude.

A.4 Purging models

In this section, we describe how we transform a model with stochastic gates $\boldsymbol{z}$ and free parameters $\tilde{\boldsymbol{\theta}}$ into a deterministic test-time model. For conciseness, we present the procedure for convolutional layers. The case of fully connected layers is analogous: simply

consider the parameter groups to be "all those weights connecting an input neuron to any neuron in the next layer".

The procedure for removing filters from the i-th convolutional layer is as follows:

1. For each filter in the layer, compute the test-time value of its associated gate as described in Appx. A.1.1.
2. Gates with medians $\hat{z}_j = 0$ are considered inactive. The value of an active gate $\hat{z}_j > 0$ is "absorbed" multiplicatively by its associated weights.
3. Prune the filters associated with inactive gates and their corresponding activation maps. The kernel entries in the next convolutional layer associated with the pruned channels in layer i are also removed. If present, weights of the adjacent batch normalization layer are removed.
4. A new kernel matrix is created for both the i-th and $(i+1)$-th layer, and the remaining kernel weights are copied to the new model.

The pruning procedure described above guarantees equivalent outputs for the network before and after pruning under the assumption that the element-wise activation function h used in the network satisfies $h(0) = 0$, as is the case for ReLU activations.

Double sparsification. We highlight that the pruning of filters of the subsequent layer in step 3 happens *regardless of whether the following layer is sparsifiable* or not. This observation implies that if a model contains adjacent sparsifiable convolutional layers, the resulting number of active parameters in the second layer will be affected by its output sparsity rate, *as well as the output sparsity rate of the first layer*.

This "double sparsification" leads to a subtle issue when studying the relationship between the proportion of active gates in the model and the *effective* number of active parameters. For example, if 80% of the (output) gates of layer i are active, and 70% of the (output) gates of layer $i+1$ are active, the effective number of active parameters in layer $i+1$ will be $\sim 56\% = (0.8 \cdot 0.7)$ and not 70%!

Note that this procedure is identical to that of Li et al. [Li+17, §3.1]. However, our selection of filters to remove is based a sparsity pattern learned during training, rather than motivated by a heuristic ranking of the filter norms. We use a similar language and presentation to facilitate the comparison.

The double sparsification effect arises due to performing structured pruning in adjacent layers, while considering groupings (i.e. one gate per output channel) which disregard the sparsity from the previous and next layers. We discuss this issue to provide clarity when analyzing our results regarding controllability: the goal of our constrained formulation is to achieve (at most) a certain proportion of expected active *gates*. Our experiments demonstrate that we can indeed achieve said control (compare target density ϵ with L_0-density columns throughout all tables). Addressing the issue of "double sparsification" in structured pruning is beyond the scope of our work.

Parameters and MACs. To achieve fair comparisons between models trained using different techniques (e.g. penalized, constrained or magnitude pruning) we apply the same purging procedure to all models. For all of the experiments presented in the pa-

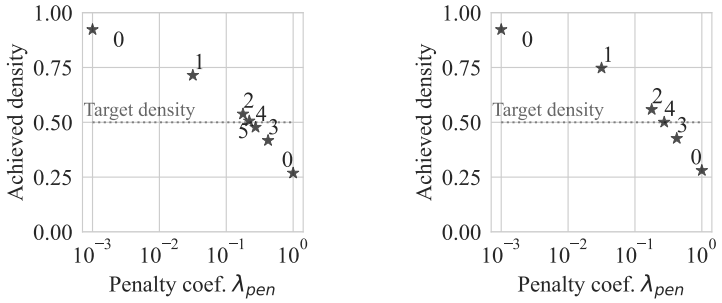
per, the reported parameter and MAC* counts are calculated based on the deterministic, purged models. The number of MACs corresponds to the number of multiply-accumulate operations involved in a *forward* pass through the network.

* *A MAC operation modifies an accumulator a as $a \leftarrow a + (b \times c)$.*

In the case of magnitude pruning, steps 1 is replaced with the ranking-and-thresholding operation of Li et al. [Li+17]. This results (conceptually) in binary 0-1 values for the gates associated with each of the filters, required in step 2.

A.5 Bisection Search

We execute a bisection search algorithm on the (logarithmic) value of λ_{pen}, aiming to achieve a model L_0-density of $50\%(\pm 1\%)$. Fig. A.1 shows the results for experiments with parameter groupings model- and layer-wise.



a) Layer-wise (b) Model-wise

Figure A.1: Iterations of bisection search on the logarithmic λ_{pen} space for achieving a model density of $50\%(\pm 1\%)$. Annotations represent iteration indices, with endpoints labelled as 0. We report the L_0-density of MLPs after 150 training epochs. On the left, parameters are grouped layer-wise and groups share a fixed λ_{pen}; on the right, parameters are grouped at the model level.

Although bisection search successfully finds a penalty value which achieves the desired density, it required the execution of at least 6 complete training cycles to be within 1% of the target. Performing such a high number of repeated experiments for tuning λ_{pen} can be in-admissibly costly in real applications. Moreover, we chose the endpoint values such that their resulting densities enclosed the target, reducing the difficulty of the search problem. While bisection search is by no means the optimal approach to adjust λ_{pen}, these experiments highlight the tunability challenges associated with penalized methods.

A.6 Constrained Optimization: Theory & Further Algorithms

Recall that a pure Nash equilibrium of the min-max game in Eq. (4.4) corresponds to a saddle point of the Lagrangian and determines an optimal, feasible solution [Neu28]. However, for non-convex problems such pure Nash equilibria might not exist, and thus simultaneous gradient descent-ascent (GDA) updates can *potentially* lead to oscillations in the parameters. [**cotter2019**; LJJ20].

There has been extensive research in the saddle-point optimization community studying in these type of problems. In particular, for convex-concave problems there are (non-)asymptotic convergence guarantees for averaged iterates from GDA with equal step-sizes [Kor76; CR97; Nem04]. Lin et al. [LJJ20] present a comprehensive bibliography of studies focusing on nonconvex-concave problems, as is the case for the Lagrangian in Eq. (4.4). The authors also present non-asymptotic complexity results showing that two-timescale GDA can find stationary points for nonconvex-concave minimax problems efficiently.

Cotter et al. [Cot+19b] propose an algorithm for non-convex constrained problems that returns an approximately optimal and feasible solution, consisting of a pair of mixed strategies (with support size of at most $m + 1$ for a problem with m constraints). Experimentally we observed convergent, non-oscillatory behavior when using simultaneous updates for solving the problem in Eq. (4.4) employing *pure strategies*, i.e., a single instance of primal and dual variables. This is discussed in detail in Section 4.5.4 and Appx. A.7.

Other approaches, such as extragradient [Kor76], provide better convergence guarantees for games like Eq. (4.4), compared to GDA. However, extragradient requires twice as many gradient computations per parameter update and the storage of an auxiliary copy of all trainable parameters. Nonetheless, extrapolation from the past [Gid+19b] enjoys similar convergence properties to extragradient without requiring a second gradient computation. Our preliminary experiments showed no significant difference in performance when using extragradient-based updates. These techniques can be useful for mitigating oscillatory behavior when applying Lagrangian-based optimization to other constrained problems.

A.7 Training dynamics and dual restarts

In this section we provide further details on the training dynamics of our gradient descent-ascent approach for solving the Lagrangian in Eq. (4.4). Fig. A.2 displays the training dynamics for a LeNet model on the MNIST dataset using model-wise density constraints of 30% and 70%, as well as whether or not using dual restarts. The experimental setup for this section matches that of Appx. A.10.3.

We employ the same learning rate for both cases. Since the left column corresponds to a constraint yielding a more sparse model, the initial constraint violations are larger. In consequence, the magnitude of the Lagrange multiplier, which accumulates the constraint violations, is also larger for the 30% case (compare the scale of the vertical axis in the plots of the second row).

The horizontal dashed line in the first row signals the desired density for each of the cases. Note that all models become feasible: blue and orange lines are at or below the density target.

However, *not* employing dual restarts leads to the model being "excessively" sparsified: the orange line overshoots past the desired density level. While in principle this behavior is not "wrong" since the constraint is satisfied, it can lead to slow learning and decreased performance. Note that our constrained formulation leads to a natural monotonicity

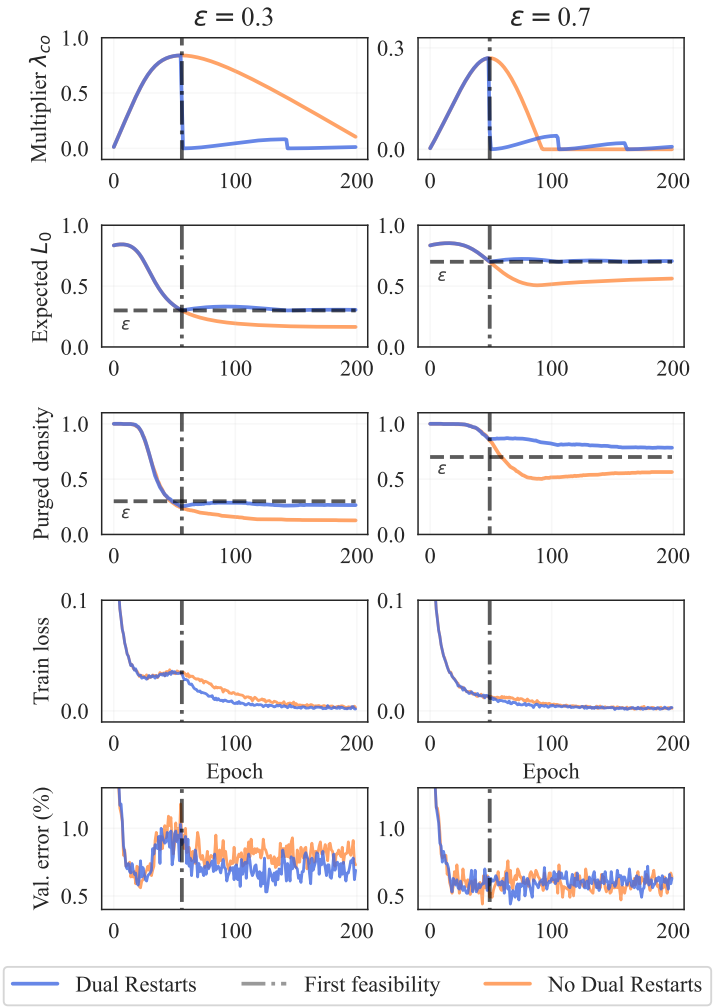


Figure A.2: Effect of the dual restarts scheme on the training dynamics for LeNet models on MNIST using model-wise constraints with target densities of 30% and 70%.

property in the constraints: if $\epsilon' < \epsilon$, the best performance achievable by a ϵ-dense model is greater than or equal to the best performance achievable by an ϵ'-dense model.

When dual restarts are applied, the contribution of the accumulated constraint violation to the Lagrangian is removed once the constraints are satisfied. Thus, the optimization is mainly guided towards minimizing the training loss (see plots in third row).

This ability to focus in reducing the loss usually come at the expense of increased density: note the slight "bounces" in model density. After reaching feasibility, models trained with dual restarts present small increases in density which are *quickly mitigated* by further growth of the multiplier. As demonstrated throughout our experiments, we can reliably achieve models that are feasible or within ($\sim 1\%$) of the desired target level.

A.7.1 *Dual restarts as best-responses*

Our proposed "dual restart" scheme is theoretically motivated as a choice of best-response from the dual player when the constraints are satisfied. Without loss of generality, we present the argument below in the case of a single inequality constraint. When there are multiple inequality constraints, the best response problem for the dual player decouples into *individual* problems for each of the Lagrange multipliers.

Given choices $[\theta,\ \phi]$ by the primal player, consider the optimization problem faced by the dual player:

$$\lambda_{\text{co}}^{\text{BR}}(\tilde{\theta}, \phi) = \underset{\lambda_{\text{co}} \geq 0}{\text{argmax}}\, \mathfrak{L}(\tilde{\theta}, \phi, \lambda_{\text{co}}) = \underset{\lambda_{\text{co}} \geq 0}{\text{argmax}} \quad \mathfrak{f}_{\text{obj}}(\tilde{\boldsymbol{\theta}}, \boldsymbol{\phi}) + \lambda_{\text{co}} \left(\mathfrak{g}_{\text{const}}(\phi) - \epsilon\right) \quad \text{(A.6)}$$

This is a linear optimization problem with a trivial solution: if the constraint is being satisfied ($\mathfrak{g}_{\text{const}}(\phi) - \epsilon < 0$), then $\lambda_{\text{co}}^{\text{BR}} = 0$. If the constraint is satisfied with equality, $\lambda_{\text{co}}^{\text{BR}} = \mathbb{R}^{+}$. Finally, if the constraint is violated ($\mathfrak{g}_{\text{const}}(\phi) - \epsilon > 0$), then $\lambda_{\text{co}}^{\text{BR}} = \infty$.

In summary, applying dual restarts corresponds to updating the value of the Lagrange multiplier following a best response for the dual player, regardless of the current value of the Lagrange multiplier! However, note that the same reasoning cannot be applied to the case of violated constraints: stability and overflow issues render a choice of ∞ for a Lagrange multiplier to be impractical for a numerical implementation.

Finally, we emphasize that while gradient ascent is an effective and simple tool for updating the Lagrange multipliers, further empirical and theoretical investigation on the influence of the dual update scheme could be beneficial.

A.8 LEARNING SPARSITY-CONTROLLED (WIDE)RESNETS

ResNets have been a challenging setting for L_0-penalty based methods. GALE et al. [GEH19] trained WideResNets (WRNs) [ZK16] and ResNet50 [He+16] using the penalized L_0-regularization framework of LOUIZOS et al. [LWK18], and reported being unable to produce sparse residual models without significantly compromising performance.* Our initial experiments on WRNs confirmed a similar behavior.

We detect two main modifications that enable us to learn WRNs and ResNets with controllable sparsity, while retaining good performance: ① increasing the learning rate of the stochastic gates, shown in Fig. A.3; and ② removing the gradient contribution of the weight decay penalty towards the gates, displayed in Fig. A.4.

For conciseness, we present these observations in the case of WideResNets. We adopt the two adjustments presented in this section for our experiments involving WideResNet-28-10, ResNet18 and ResNet50 models. These two simple modifications allowed us to achieve reliable controllability for (Wide)ResNets without performance degradation,

*GALE et al. [GEH19] state: *"Across hundreds of experiments, our [ResNet50] models were either able to achieve full test set performance with no sparsification, or sparsification with test set performance akin to random guessing."*; and *"Applying our weight-level L_0 regularization implementation to WRN produces a model with comparable training time sparsity, but with no sparsity in the test-time parameters. For models that achieve test-time sparsity, we observe significant accuracy degradation on CIFAR-10."*

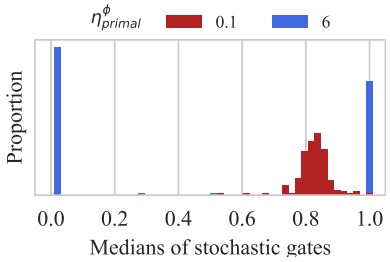


Figure A.3: Distribution of gate medians for the first layer of a WRN, at the end of (penalized) training using $\lambda_{\text{pen}} = 10^{-3}$.

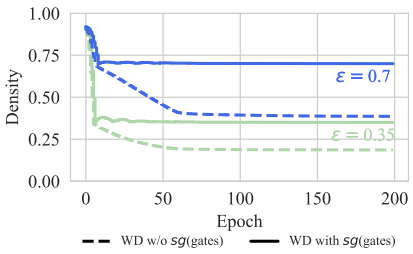


Figure A.4: Not detaching the gradient contribution of the weight decay (WD) penalty leads to excesive sparsification.

just as with the MLP and LeNet architectures.

Further exploration of these modifications, and their influence in the resulting sparsity of the model are provided in Appx. A.9.

A.8.1 *Achieving sparsity in (Wide)ResNets by tuning the learning rate of the gates.*

Replicating the CIFAR10 experiments of Louizos et al. [LWK18] on WRNs using their choice of regularization parameter and learning rate for the stochastic gates, results in a distribution of the stochastic gates which does not induce sparsity in the model. Fig. A.3 shows (in red) the distribution of the gate medians in the first layer of a WRN trained using a learning rate of $\eta^{\phi}_{\text{primal}} = 0.1$ for the gates parameters, as in Louizos et al. [LWK18]. We observed that this distribution of medians did not change significantly during training. Thus, since the model has a high initial density, the gate parameters at the end of training do not induce any sparsity.

To enable the gate parameters to effectively change during training, we decoupled the learning rate of the gates $\eta^{\phi}_{\text{primal}}$, from that of the model weights $\eta^{\tilde{\theta}}_{\text{primal}}$. Fig. A.3 illustrates how increasing $\eta^{\phi}_{\text{primal}}$ from 0.1 to 6 leads to a drastically different distribution for the gate medians. This simple change results in a distribution of medians which exhibits the desired concentration behavior: a non-negligible proportion of gates have a median of zero, and are therefore inactive (see Appx. A.1.1).

We adopt this learning rate adjustment in all our WRN experiments. Table A.10 presents the performance of WRNs trained using different values of $\eta^{\phi}_{\text{primal}}$, for both the constrained and penalized settings. Note how the models using a higher learning rate for the gates parameters successfully achieve sparsity *without any compromise in performance*.

A.8.2 *A loophole in weight-decay leads to excessive regularization.*

This section includes the adjustment to the learning rate of the gates presented above. We noticed a systematic over-sparsification behavior in WRNs when solving the constrained formulation: the constraint is satisfied, well beyond the prescribed density level.

This issue is illustrated in Fig. A.4. Dashed lines correspond to the weight decay from Louizos et al. [LWK18] and solid lines correspond to our method with the modified weight decay as in Eq. (A.7). Experimental details for CIFAR-10 experiments are provided in Appx. A.10.4.

We identified the cause of this phenomenon to be the L_2 weight decay term in the training objective of WRNs from Louizos et al. [LWK18] (see Section 4.2). This penalty term depends both on the probability of gates being active $\pi_j = \mathbb{P}[z_j \neq 0]$, as well as the norm of the signed magnitudes $\tilde{\theta}_j^2$. Reducing the value of this penalty could be achieved by turning off gates, such that the contribution of their associated magnitudes is ignored. This behavior is undesirable for the purpose of controllability.

We propose to restrict the effect of the weight decay penalty to $\tilde{\boldsymbol{\theta}}$, as a way to reduce the parameter magnitudes, and keep the gates deactivation under the *sole influence* of the constraint violation term. We achieve this by stopping (also known as detaching) the gradients from propagating through the gate-dependent terms in the L_2 norm:

$$\mathbb{E}_{\boldsymbol{z}|\boldsymbol{\phi}}\left[\|\hat{\boldsymbol{\theta}}\|_2^2\right] = \sum_{j=1}^{|\boldsymbol{\theta}|} \texttt{stop-grad}(\pi_j)\,\tilde{\theta}_j^2. \tag{A.7}$$

Fig. A.4 shows the effect of this simple adjustment when applying a model-wise constraint on a WRN for the CIFAR10 dataset. Note how, removing the influence of weight decay on the gate parameters allows us to reliably achieve the desired target density, without over-sparsifying the model.

A.9 Test-time gates: influence of learning rate and weight-decay

Recall that we make the gates "freeze" the gates at their medians to obtain a deterministic model to evaluate on unseen data (Appx. A.1.1). We analyze the behavior of gates at test-time by considering histograms of their medians across specific layers of models.* This section provides further empirical evidence to support the hypothesis presented in Appx. A.8 (see Appx. A.10.4 for experiment setup).

** Note the subtle detail that these histograms are based on a statistic (the median) of a probability distribution and do not represent distributions of gates by themselves.*

Fig. A.5 contains histograms for the first layer of an MLP and a LeNet trained to classify MNIST digits. These correspond to a fully connected and a convolutional layer, respectively. Sparsity requirements are specified via layer-wise constraints with a target density of 70% on both cases. Experimental details for these runs match those presented in Appx. A.10. Test-time gate medians are measured at the *end* of the training epochs shown in the panel titles.

At initialization, the distributions of different gates are highly similar and yield closely packed medians. These are mostly fractional in value, away from being 0 or 1. As training progresses, the medians drift apart. Histograms peak at 0 and 1 as of the 50th epoch. As expected, approximately $30\% = (100 - 70)\%$ of gates are inactive at the end of training.

Fig. A.6 contains histograms of the test-time gates associated with the first convolutional layer of various WRNs-28-10 trained on CIFAR10. The proportion of gates at

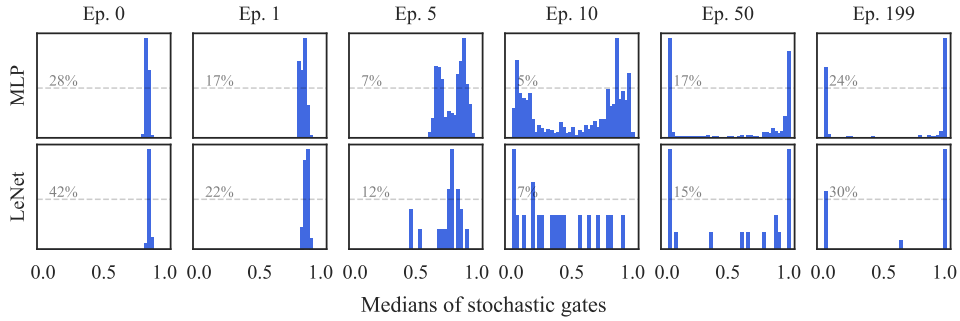


Figure A.5: Histograms of gate *medians* in first layer of MLP and LeNet models. Note the transition from highly concentrated around a fractional value of 0.8 at the first epochs, towards an approximately binary distribution peaking at 0 and 1 at the last epoch. These models were trained on MNIST with a layer-wise target density of 70%.

zero specifies the amount of inactive gates, while bars in $(0, 1]$ correspond to active gates. Since all histograms have different scales, we provide a reference line for each, corresponding to *half* the height of the tallest bar in said histogram. For example, if the dashed line is labeled as 15%, then the highest bar in that histogram corresponds to 30%.

We consider 6 configurations spanning: three learning rates for the gate parameters ϕ; and whether or not to remove the gradient contribution of the weight decay towards the gates (see Appx. A.8). All experiments use the same initialized model. Sparsity requirements are specified via layer-wise constraints with a target density of 70% on all cases. Measurements of the gate medians are made at the *end* of each of the presented epochs.

Medians do not drift apart noticeably when employing $\eta^{\phi}_{\text{primal}} = 0.1$. In addition, they maintain fractional values (i.e., remain away from 0 and 1). Given our protocol for choosing gates at test-time in Appx. A.1.1, this would lead to a "fully dense" test-time network, which violates the required sparsity constraints by a wide margin. Similar to Gale et al. [GEH19], we observed that when a WRN trained with $\eta^{\phi}_{\text{primal}} = 0.1$ achieved *any* significant degree of sparsity, it led to a performance akin to random guessing. Note that, unsurprisingly, the bulk of the medians gets closer to zero: this is a consequence of the model aiming to satisfy the constraint on the expectation of the L_0-norm.

The behavior when training with $\eta^{\phi}_{\text{primal}} = 1$ and $\eta^{\phi}_{\text{primal}} = 6$ stands in clear contrast to that of $\eta^{\phi}_{\text{primal}} = 0.1$. The former two resemble more closely the dynamics observed in Fig. A.5, where medians disperse and tend to accumulate and saturate at 0 or 1. Unsurprisingly, this accumulation happens more quickly for experiments with $\eta^{\phi}_{\text{primal}} = 6$. This setting also has the smallest proportion of fractional medians at the end of training. Note that in the case of $\eta^{\phi}_{\text{primal}} = 1$, longer training could result in a similar outlook to that of $\eta^{\phi}_{\text{primal}} = 6$. Choosing $\eta^{\phi}_{\text{primal}} > 1$ in our experiments led to better performance in terms of test-time error.

Finally, note how[†] removing the gradient contribution of the weight decay towards the gates (WD **with** $\mathtt{sg}(z)$) yields yields test-time models which have approximately 30%

[†] *Except for the undesirable setting* $\eta^{\phi}_{primal} = 0.1$, *as discussed in Appx. A.8.*

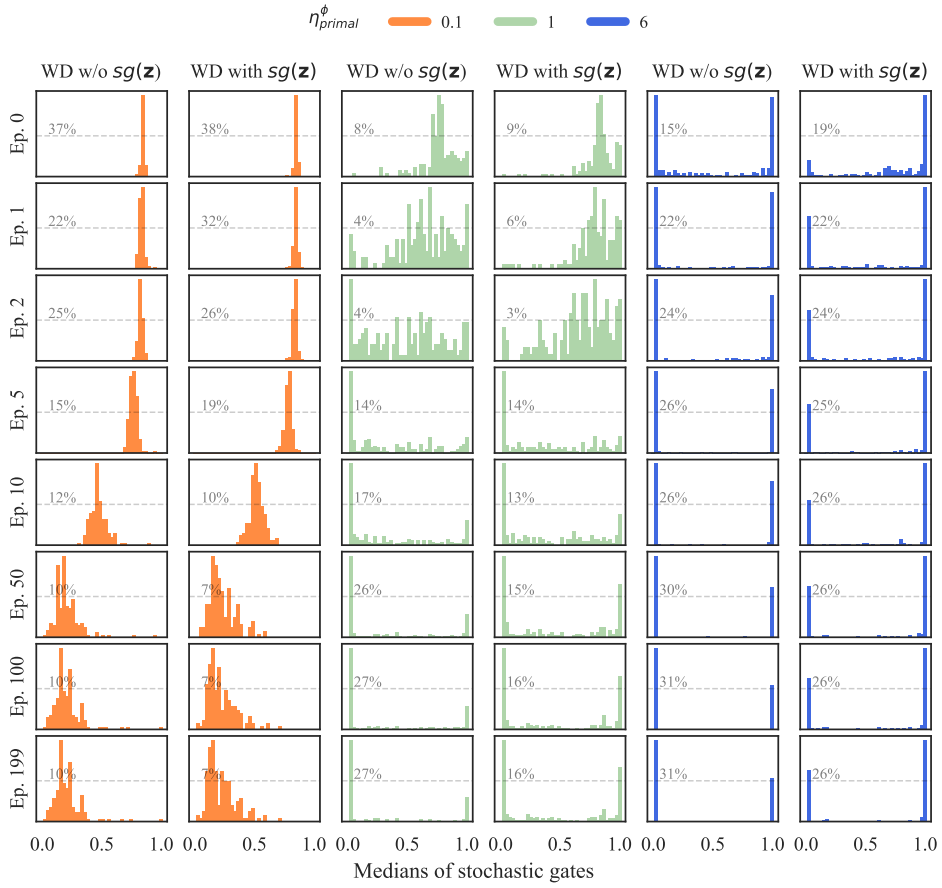


Figure A.6: Histograms of gate medians for the first convolutional layer of WRN models trained on CIFAR10 under 70% layer-wise density constraints. Configurations comprise three learning rates $\eta^{\phi}_{\text{primal}}$, and removing or not the gradient contribution of the weight decay towards the gate updates.

of their parameters *inactive*, as desired. Not removing this contribution (WD **without** $\texttt{sg}(z)$) results in over-sparsification: the distribution of gate medians is shifted towards zero for all learning rate choices. For example, consider the panels at the last epoch for $\eta^{\phi}_{\text{primal}} = 1$ and $\eta^{\phi}_{\text{primal}} = 6$. The experiments *with* detaching result in models whose sparsity is close to the desired sparsity level (peak at 0 close to 30%). In contrast, when not detaching, the rate of *inactive* gates is around 60%, twice as sparse as required.

A.10 Experimental Details

Our implementation is developed in Python 3.8, using Pytorch 1.11 [Pas+19] and the `Cooper` constrained optimization library [Gal+24]. We provide scripts to replicate the experiments in this paper at: `https://github.com/gallego-posada/constrained_sparsity`.

All our models use ReLU activations. Throughout our experiments ① we decouple the learning rates used for the weights and gates of the network, and ② when employing weight decay, we remove the gradient contribution of the penalty towards the gates, as

explained in Appx. A.8.

A.10.1 *Model statistics*

Table A.2 describes the different architectures used throughout our experiments in terms of their total number of trainable parameters and the computational cost involved in a *forward* calculation in MACs (multiply-accumulate operations). We also provide details on the input size and the number of training examples for their respective datasets.

Table A.2: Count of parameters and MACs for all architectures used in this paper, along with dimension and number of training examples.

Model Type	Parameters	MACs	Input size	Train set size	Dataset
MLP	266k	267k	(28, 28)	50k	MNIST
LeNet	431k	2,327M	(28, 28)	50k	MNIST
WideResNet-28-10	36.5M	5,959M	(3, 32, 32)	50k	CIFAR-10/100
ResNet18	11.3M	6,825M	(3, 64, 64)	100k	TinyImageNet
ResNet50	25.5M	4,120M	(3, 224, 224)	1.2M	ImageNet

A.10.2 *Dual optimizer*

Note that the constraint functions considered throughout this work involve expectations but can be computed in closed-form based on the parameters of the gates (Appx. A.1). Therefore, the computation of constraint violations is deterministic. We employ gradient *ascent* on the Lagrange multipliers. We initialize all Lagrange multipliers at zero. Details on the chosen dual learning rate, along with the use of dual restarts are provided for each experiment below.

A.10.3 *MNIST*

Following Louizos et al. [LWK18], our experiments on MNIST classification consider two different architectures: i) an MLP with 2 hidden layers with 300 and 100 units respectively, and ii) a LeNet-5 network, consisting on convolutional layers of 20 and 50 output channels, each succeeded by a max-pooling layer with stride 2; followed by two fully connected layers of 800 and 500 input dimensions. All fully connected layers in these models use input neuron sparsity, and all convolutional layers (for LeNet models) employ output feature map sparsity.

Table A.3: Default configurations for MLP and LeNet5 experiments on MNIST.

Approach	Weights		Gates		Lagrange Multipliers		
	Optim.	$\eta^{\tilde{\theta}}_{\text{primal}}$	Optim.	$\eta^{\phi}_{\text{primal}}$	Optim.	η_{dual}	Restarts
Constrained	Adam	$7 \cdot 10^{-4}$	Adam	$7 \cdot 10^{-4}$	Grad. Ascent	10^{-3}	Yes
Penalized					-	-	-

Table A.3 presents the hyper-parameters used for learning sparse MLPs and LeNets. Both cases employ the same configuration for the primal optimizer: Adam [**adam**] with $(\beta_1,\ \beta_2) = (0.9,\ 0.999)$, as provided by default in Pytorch, with a batch size of 128. These experiments do not use weight decay.

A.10.4 *CIFAR-10 and CIFAR-100*

We employ WideResNet-28-10 (WRN) models [ZK16] for the tasks of classifying CIFAR-10 and CIFAR-100 images. Akin to Louizos et al. [LWK18], the first convolutional layer in each residual block uses output feature map sparsity, whereas the following convolutional layer and the residual connection are kept to be fully dense. This model counts with 12 sparsifiable convolutional layers.

Table A.4 presents the hyper-parameters used for learning sparse WRNs. We use SGD with a momentum coefficient of 0.9 for the weights and gates. We use a batch size of 128 for 200 epochs. The primal learning rate is multiplied by 0.2 at 60, 120 and 160 epochs. This mimics the training procedure of Zagoruyko and Komodakis [ZK16] and Louizos et al. [LWK18]. These experiments use $\rho_{\text{init}} = 0.3$ (see Appx. A.1.2).

Table A.4: Default configurations for WideResNet-28-10 experiments on CIFAR-{10, 100}.

Approach	**Weights**		**Gates**		**Lagrange Multipliers**			**Weight decay**
	Optim.	$\eta^{\tilde{\boldsymbol{\theta}}}_{\text{primal}}$	Optim.	$\eta^{\boldsymbol{\phi}}_{\text{primal}}$	Optim.	η_{dual}	Restarts	**coefficient**
Constrained	SGDM	0.1	SGDM	6	Grad. Ascent	$7 \cdot 10^{-4}$	Yes	$5 \cdot 10^{-4}$
Penalized					-	-	-	

A.10.5 *TinyImageNet*

We employ ResNet18 models for the task of classifying TinyImageNet [LKJ17] images. The model's initial convolutional and final fully connected layers are kept fully dense. The residual connection of each `BasicBlock` in the model is kept fully dense, while all other convolutional layers employ output feature-map sparsity. This model thus counts with 16 sparsifiable convolutional layers.

Table A.5 presents the hyper-parameters used for learning sparse ResNet18s. We use SGD with a momentum coefficient of 0.9 for the weights and gates. We use a batch size of 100 for 120 epochs. The learning rate of the weights $\eta^{\tilde{\boldsymbol{\theta}}}_{\text{primal}}$ is multiplied by 0.1 at 30, 60 and 90 epochs. This mimics the training procedure of previous works [Kun+20]. This experiment uses $\rho_{\text{init}} = 0.3$ (see Appx. A.1.2).

A.10.6 *ImageNet*

We employ ResNet50 models for the task of classifying ImageNet [Den+09] images. The model's initial convolutional and final fully connected layers are kept fully dense. The residual connection of each `Bottleneck` block in the model is kept fully dense, while all other convolutional layers employ output feature-map sparsity. This model thus counts with 48 sparsifiable convolutional layers.

Table A.5: Default configurations for ResNet18 experiments on TinyImageNet.

Approach	Grouping	Weights		Gates		Lagrange Multipliers			Weight decay
		Optim.	$\eta^{\tilde{\theta}}_{\text{primal}}$	Optim.	$\eta^{\phi}_{\text{primal}}$	Optim.	η_{dual}	Restarts	coefficient
Constrained	Model	SGDM	0.1	SGDM	1	Grad. Ascent	$8 \cdot 10^{-4}$	Yes	$5 \cdot 10^{-4}$
	Layer						$1 \cdot 10^{-4}$		
Penalized	Model					-	-	-	
	Layer					-	-	-	

Due to the high computational cost of ImageNet experiments, and the tunability issues of the penalized method, we do not perform penalized experiments for this dataset.

Table A.6: Default configurations for ResNet50 experiments on ImageNet.

Approach	Grouping	Weights		Gates		Lagrange Multipliers			Weight decay
		Optim.	$\eta^{\tilde{\theta}}_{\text{primal}}$	Optim.	$\eta^{\phi}_{\text{primal}}$	Optim.	η_{dual}	Restarts	coefficient
Constrained	Model	SGDM	0.1	SGDM	1	Grad. Ascent	$3 \cdot 10^{-4}$	Yes	10^{-4}
	Layer						$3 \cdot 10^{-5}$		

Table A.6 presents the hyper-parameters used for learning sparse ResNet50s. SGD with a momentum coefficient of 0.9 is used for the weights and the gates. We use a batch size of 256 for 90 epochs. The learning rate of the weights $\eta^{\tilde{\theta}}_{\text{primal}}$ is multiplied by 0.1 at 30 and 60 epochs. This mimics the training procedure of previous works [SSM20].

Initialization of the gates

As discussed in Appx. A.1.2, the choice of hyperparameter ρ_{init} affects the initial sparsity of the network. For their experiments using WideResNet models on the CIFAR10/100 datasets, Louizos et al. [LWK18] chose $\rho_{\text{init}} = 0.3$. When executing our experiments with ResNet50 models on ImageNet, we noticed that this hyper-parameter of the L_0-reparametrization can have a significant impact in the behavior of the model throughout training.

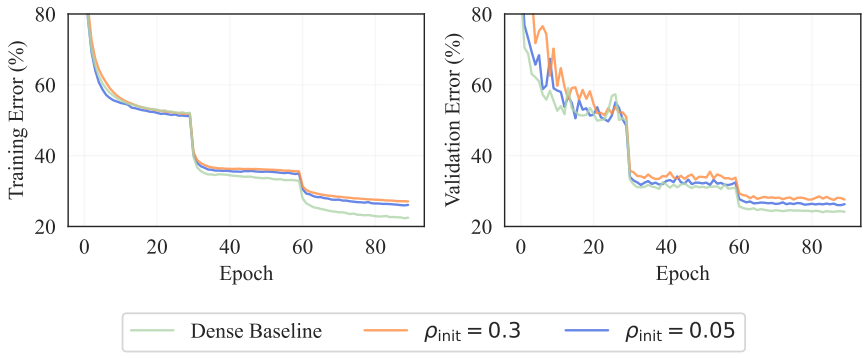


Figure A.7: Effect of the initialization hyper-parameter ρ_{init} for training ResNet50s on ImageNet.

Fig. A.7 shows the training and validation error of two different runs at a target density of 70% with $\rho_{\text{init}} = 0.3$ and $\rho_{\text{init}} = 0.05$. Although both runs achieve the desired sparsity target, the model initialized at $\rho_{\text{init}} = 0.05$ outperforms the model initialized with $\rho_{\text{init}} =$

0.3. This performance improvement persists throughout training, resulting in a final model with a better validation error.

Recall that we consider fixed values for the parameters β, γ and ζ associated with the concrete distribution, as detailed in Table A.1. With these values, Eq. (A.5) yields that the initial L_0-density of the model initialized with $\rho_{\text{init}} = 0.3$ is 92.03%, while for $\rho_{\text{init}} = 0.05$, the L_0-density of the initial model is 98.95%. This means that the case $\rho_{\text{init}} = 0.3$ starts from a (in expectation) $\sim 7\%$ sparser model; thus unnecessarily restring the model capacity at the beginning of training. This is consistent with the behavior displayed in Fig. A.7: the model initialized with $\rho_{\text{init}} = 0.05$ follows more closely the validation performance of a dense baseline (i.e. a standard ResNet50 without gates).

Considering this behavior, for all the ImageNet results reported below we use the lower value of $\rho_{\text{init}} = 0.05$. For other models and datasets, we employ the value used in Louizos et al. [LWK18] for ease of comparison.

A.10.7 *Magnitude pruning comparison on ImageNet*

We compare the performance of our in-training sparsity method with structured magnitude pruning. We start from a pre-trained ResNet50 model from Pytorch (`torchvision.models.resnet50`) and apply layer-wise pruning following the procedure of Li et al. [Li+17] to *the same layers* that were sparsifiable for our ImageNet models, described in Appx. A.10.6. After performing magnitude pruning, we fine-tune the models for 20 epochs on the ImageNet dataset, using SGD with momentum of 0.9 and a constant learning rate of 0.001. This matches the fine-tuning setting of Li et al. [Li+17].

A.10.8 *Sparsity collapse*

Some of the results for the penalized method presented in Appx. A.11 are labeled as "*Failed due to sparsity collapse*". This means that the penalty factor was too high and resulted in all the gates of a layer being turned off.

A.11 Comprehensive experimental results

In this section we provide complete results for all experiments, whose hyper-parameter configurations can be found in Appx. A.10. To make the navigation of these results easier, we repeat some of the tables and figures provided in the main paper. These repetitions are clearly marked in the caption of the corresponding resource.

A.11.1 *MNIST*

Table A.7: Achieved density levels and performance for sparse MLP and LeNet5 models trained on MNIST for 200 epochs. Metrics aggregated over 5 runs. †Results by Louizos et al. [LWK18] with N representing the training set size (see Appx. A.3). *This table is the same as Table 4.1. We repeat it here for the reader's convenience.*

Architecture	Grouping	Method	Hyper-parameters	Pruned architecture	Val. Error (%) best	Val. Error (%) at 200 epochs avg ± 95% CI
MLP 784-300-100	Model	Pen.	†$\lambda_{pen} = 0.1/N$	219-214-100	1.4	–
		Const.	$\epsilon = 33\%$	198-233-100	1.36	1.77 ± 0.08
	Layer	Pen.	†$\lambda_{pen} = [0.1, 0.1, 0.1]/N$	266-88-33	1.8	–
		Const.	$\epsilon = [30\%, 30\%, 30\%]$	243-89-29	1.58	2.19 ± 0.12
LeNet5 20-50-800-500	Model	Pen.	†$\lambda_{pen} = 0.1/N$	20-25-45-462	0.9	–
		Const.	$\epsilon = 10\%$	20-21-34-407	0.56	1.01 ± 0.05
	Layer	Pen.	†$\lambda_{pen} = [10, 0.5, 0.1, 0.1]/N$	9-18-65-25	1.0	–
		Const.	$\epsilon = [50\%, 30\%, 70\%, 10\%]$	10-14-224-29	0.7	0.91 ± 0.05

MLPs

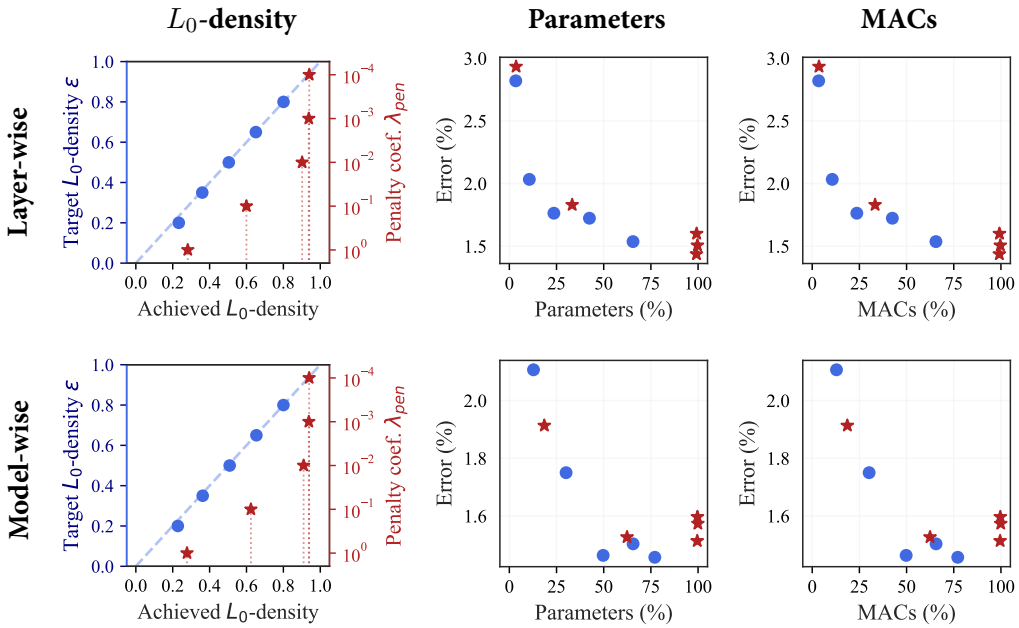


Figure A.8: Training sparse MLP models on MNIST.

Table A.8: Achieved density levels and performance for sparse MLP models trained on MNIST for 200 epochs. Metrics aggregated over 5 runs.

Method	Hyper-params	L_0-density (%)	Params (%)	MACs (%)	Val. Error (%)	
					best	at 200 epochs
Constrained $g \in [1:3]$ *Layer-wise*	$\epsilon_g = 20\%$	23.28 ±0.31	3.41 ±0.16	3.43 ±0.16	1.65	2.82 ±0.31
	$\epsilon_g = 35\%$	36.02 ±0.08	10.59 ±0.17	10.62 ±0.17	1.66	2.03 ±0.13
	$\epsilon_g = 50\%$	50.42 ±0.04	23.60 ±0.14	23.63 ±0.14	1.58	1.76 ±0.15
	$\epsilon_g = 65\%$	65.07 ±0.03	42.43 ±0.51	42.46 ±0.51	1.42	1.72 ±0.01
	$\epsilon_g = 80\%$	80.08 ±0.07	65.53 ±0.76	65.55 ±0.76	1.38	1.54 ±0.05
Penalized $g \in [1:3]$ *Layer-wise*	$\lambda^g_{pen} = 1$	28.16 ±0.97	3.58 ±0.51	3.60 ±0.51	2.62	2.93 ±0.09
	$\lambda^g_{pen} = 0.1$	59.97 ±0.09	33.29 ±0.06	33.32 ±0.06	1.37	1.83 ±0.07
	$\lambda^g_{pen} = 0.01$	90.18 ±0.22	99.15 ±0.94	99.15 ±0.94	1.29	1.44 ±0.13
	$\lambda^g_{pen} = 0.001$	93.77 ±0.10	99.23 ±0.22	99.23 ±0.22	1.33	1.60 ±0.10
	$\lambda^g_{pen} = 0.0001$	94.04 ±0.23	99.67 ±0.38	99.67 ±0.38	1.26	1.51 ±0.04
Constrained *Model-wise*	$\epsilon = 20\%$	22.77 ±0.17	12.80 ±0.41	12.87 ±0.41	1.40	2.11 ±0.05
	$\epsilon = 35\%$	36.25 ±0.06	30.07 ±0.39	30.16 ±0.39	1.37	1.75 ±0.11
	$\epsilon = 50\%$	50.89 ±0.12	49.71 ±0.27	49.78 ±0.27	1.20	1.46 ±0.16
	$\epsilon = 65\%$	65.37 ±0.01	65.57 ±0.22	65.62 ±0.22	1.27	1.50 ±0.03
	$\epsilon = 80\%$	80.02 ±0.05	77.08 ±0.86	77.11 ±0.86	1.16	1.46 ±0.16
Penalized *Model-wise*	$\lambda_{pen} = 1$	27.73 ±0.15	18.54 ±0.51	18.62 ±0.52	1.48	1.91 ±0.07
	$\lambda_{pen} = 0.1$	62.38 ±0.16	62.37 ±0.3	62.43 ±0.3	1.30	1.53 ±0.13
	$\lambda_{pen} = 0.01$	90.98 ±0.18	99.56 ±0.43	99.56 ±0.43	1.34	1.51 ±0.11
	$\lambda_{pen} = 0.001$	93.81 ±0.17	99.89 ±0.22	99.89 ±0.22	1.24	1.57 ±0.09
	$\lambda_{pen} = 0.0001$	93.99 ±0.06	99.67 ±0.38	99.67 ±0.38	1.32	1.60 ±0.17

LeNet5

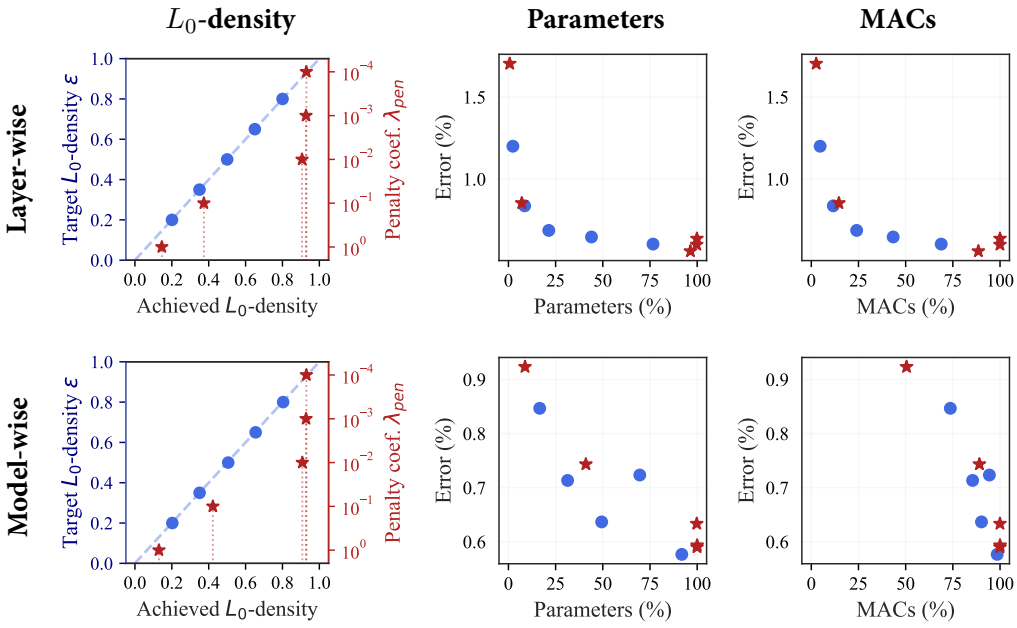


Figure A.9: Training sparse LeNet models on MNIST.

Table A.9: Achieved density levels and performance for sparse LeNet models trained on MNIST for 200 epochs. Metrics aggregated over 5 runs.

Method	Hyper-params	L_0-density (%)	Params (%)	MACs (%)	Val. Error (%)	
					best	at 200 epochs
Constrained $g \in [1:4]$ *Layer-wise*	$\epsilon_g = 20\%$	20.09 ±0.07	2.36 ±0.14	4.68 ±0.68	0.81	1.20 ±0.08
	$\epsilon_g = 35\%$	35.01 ±0.01	8.59 ±0.03	11.65 ±1.15	0.69	0.84 ±0.08
	$\epsilon_g = 50\%$	50.02 ±0.01	21.38 ±0.77	24.03 ±1.64	0.56	0.69 ±0.07
	$\epsilon_g = 65\%$	65.02 ±0.01	44.01 ±2.16	43.38 ±2.39	0.53	0.65 ±0.09
	$\epsilon_g = 80\%$	80.04 ±0.02	76.62 ±0.99	68.83 ±1.40	0.45	0.60 ±0.04
Penalized $g \in [1:4]$ *Layer-wise*	$\lambda^g_{pen} = 1$	14.58 ±0.44	0.67 ±0.13	2.60 ±0.17	0.46	1.7 ±0.14
	$\lambda^g_{pen} = 0.1$	37.38 ±0.49	7.06 ±0.36	14.62 ±1.68	0.54	0.85 ±0.02
	$\lambda^g_{pen} = 0.01$	90.63 ±0.20	96.43 ±0.83	88.52 ±7.17	0.40	0.56 ±0.03
	$\lambda^g_{pen} = 0.001$	92.77 ±0.06	99.75 ±0.12	99.95 ±0.02	0.47	0.60 ±0.05
	$\lambda^g_{pen} = 0.0001$	92.96 ±0.04	99.87 ±0.12	99.98 ±0.02	0.47	0.64 ±0.06
Constrained *Model-wise*	$\epsilon = 20\%$	20.27 ±0.30	16.58 ±0.73	73.67 ±0.14	0.54	0.85 ±0.09
	$\epsilon = 35\%$	35.06 ±0.09	31.36 ±0.27	85.47 ±0.85	0.55	0.71 ±0.03
	$\epsilon = 50\%$	50.58 ±0.09	49.41 ±0.29	90.18 ±0.94	0.44	0.64 ±0.03
	$\epsilon = 65\%$	65.47 ±0.05	69.60 ±1.06	94.37 ±0.20	0.44	0.72 ±0.03
	$\epsilon = 80\%$	80.36 ±0.08	91.86 ±0.79	98.49 ±0.15	0.46	0.58 ±0.03
Penalized *Model-wise*	$\lambda_{pen} = 1$	13.01 ±0.81	8.79 ±0.80	50.47 ±4.74	0.71	0.92 ±0.11
	$\lambda_{pen} = 0.1$	42.25 ±0.83	41.03 ±1.12	89.08 ±0.21	0.45	0.74 ±0.04
	$\lambda_{pen} = 0.01$	90.78 ±0.20	99.78 ±0.20	99.96 ±0.04	0.44	0.63 ±0.05
	$\lambda_{pen} = 0.001$	92.80 ±0.01	100 ±0.00	100 ±0.00	0.43	0.59 ±0.01
	$\lambda_{pen} = 0.0001$	92.97 ±0.03	99.94 ±0.12	99.99 ±0.02	0.45	0.59 ±0.07

A.11.2 *CIFAR-10*

Table A.10: Achieved density levels and performance for sparse WideResNets-28-10 models trained on CIFAR-10 for 200 epochs. Metrics aggregated over 5 runs. †Result reported by Louizos et al. [LWK18], with N denoting the training set size (see Appx. A.3).

Method	Hyper-params	$\eta^{\phi}_{\text{primal}}$	L_0-density (%)	Params (%)	MACs (%)	Val. Error (%)	
						best	at 200 epochs (avg ± 95% CI)
Penalized	†$\lambda_{pen} = 0.001/N$	0.1	–	–	–	3.83	–
	†$\lambda_{pen} = 0.002/N$	0.1	–	–	–	3.93	–
	$\lambda_{pen} = 0.001$	0.1	92.30 ±0.01	99.84 ±0.00	100 ±0.00	4.23	4.56 ±0.18
	$\lambda_{pen} = 0.001$	6	91.19 ±0.16	93.98 ±0.24	91.57 ±0.35	3.75	4.04 ±0.15
	$\lambda_{pen} = 0.002$	6	91.36 ±0.11	94.26 ±0.22	92.10 ±0.46	3.62	4.05 ±0.14
Constrained $g \in [1:12]$	$\epsilon_g = 100\%$	0.1	92.34 ±0.01	99.84 ±0.00	100 ±0.00	4.18	4.63 ±0.14
	$\epsilon_g = 100\%$	6	91.63 ±0.27	94.52 ±0.26	92.75 ±0.58	3.74	4.12 ±0.08
	$\epsilon_g = 70\%$	6	70.00 ±0.00	69.87 ±0.20	69.55 ±0.17	3.76	4.10 ±0.16

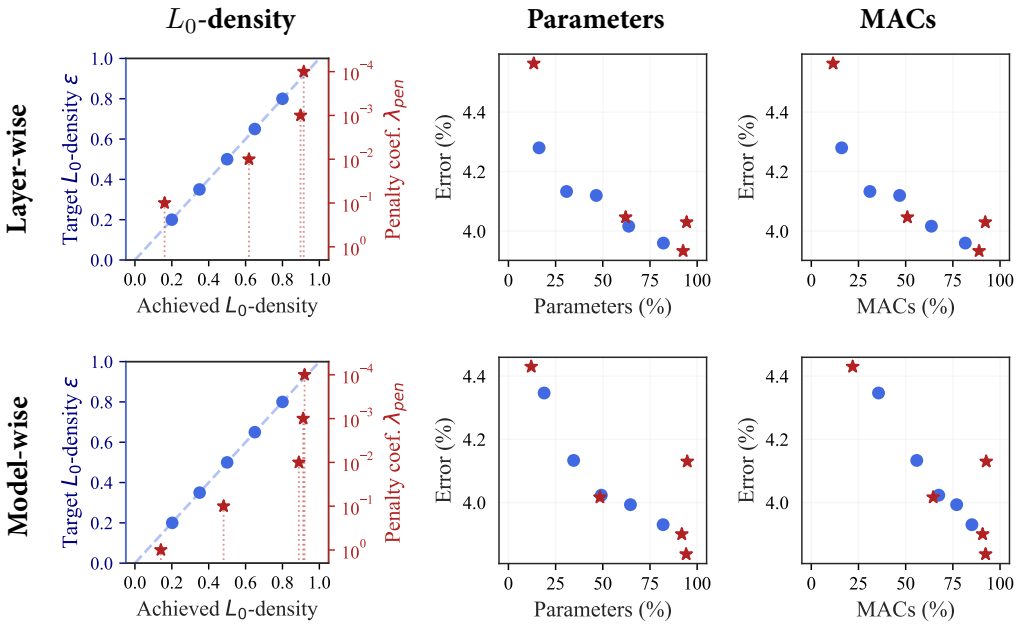


Figure A.10: Training sparse WideResNet-28-10 models on CIFAR-10. *This figure is the same as Fig. 4.4. We repeat it here for the reader's convenience.*

Table A.11: Achieved density levels and performance for sparse WideResNet-28-10 models trained on CIFAR-10 for 200 epochs. Metrics aggregated over 5 runs.

Method	**Hyper-params**	L_0**-density** (%)	**Params** (%)	**MACs** (%)	**Val. Error** (%)	
					best	at 200 epochs
Constrained $g \in [1:12]$ *Layer-wise*	$\epsilon_g = 20\%$	20.00 ±0.02	16.24 ±0.20	16.10 ±0.23	4.05	4.28 ±0.16
	$\epsilon_g = 35\%$	34.99 ±0.01	30.80 ±0.34	31.05 ±0.23	3.8	4.13 ±0.13
	$\epsilon_g = 50\%$	50.00 ±0.01	46.54 ±0.21	46.79 ±0.28	3.87	4.12 ±0.21
	$\epsilon_g = 65\%$	64.99 ±0.01	63.74 ±0.33	63.63 ±0.27	3.7	4.02 ±0.11
	$\epsilon_g = 80\%$	80.00 ±0.00	82.10 ±0.32	81.60 ±0.34	3.69	3.96 ±0.15
Penalized $g \in [1:12]$ *Layer-wise*	$\lambda^g_{pen} = 1$	——	*Failed due to sparsity collapse*		——	
	$\lambda^g_{pen} = 0.1$	16.09 ±0.14	13.43 ±0.11	11.54 ±0.29	4.23	4.56 ±0.09
	$\lambda^g_{pen} = 0.01$	61.83 ±0.86	62.10 ±0.97	50.91 ±0.62	3.83	4.05 ±0.02
	$\lambda^g_{pen} = 0.001$	89.81 ±0.06	92.55 ±0.13	88.91 ±0.15	3.7	3.93 ±0.15
	$\lambda^g_{pen} = 0.0001$	91.54 ±0.33	94.41 ±0.48	92.18 ±0.94	3.81	4.03 ±0.05
Constrained *Model-wise*	$\epsilon = 20\%$	20.22 ±0.01	18.91 ±0.03	35.62 ±0.84	3.95	4.35 ±0.25
	$\epsilon = 35\%$	35.07 ±0.03	34.55 ±0.08	55.87 ±1.23	3.76	4.13 ±0.27
	$\epsilon = 50\%$	50.00 ±0.01	49.21 ±0.33	67.53 ±0.08	3.79	4.02 ±0.06
	$\epsilon = 65\%$	65.01 ±0.01	64.65 ±0.34	77.02 ±0.43	3.80	3.99 ±0.14
	$\epsilon = 80\%$	80.01 ±0.00	81.96 ±0.12	85.11 ±0.58	3.83	3.93 ±0.09
Penalized *Model-wise*	$\lambda_{pen} = 1$	14.07 ±0.23	12.09 ±0.24	21.92 ±0.59	4.23	4.43 ±0.10
	$\lambda_{pen} = 0.1$	48.08 ±2.05	48.50 ±2.27	64.58 ±1.57	3.87	4.02 ±0.22
	$\lambda_{pen} = 0.01$	88.89 ±0.31	91.85 ±0.47	90.76 ±0.77	3.72	3.90 ±0.18
	$\lambda_{pen} = 0.001$	91.24 ±0.20	94.17 ±0.24	92.34 ±0.22	3.59	3.84 ±0.25
	$\lambda_{pen} = 0.0001$	91.98 ±0.42	94.75 ±0.31	92.68 ±0.36	3.91	4.13 ±0.09

A.11.3 *CIFAR-100*

Table A.12: Achieved density levels and performance for sparse WideResNets-28-10 models trained on CIFAR-100 for 200 epochs. Metrics aggregated over 5 runs. †Result reported by Louizos et al. [LWK18], with N denoting the training set size (see Appx. A.3).

Method	Hyper-params	$\eta^{\phi}_{\text{primal}}$	L_0-density (%)	Params (%)	MACs (%)	Val. Error (%)	
						best	at 200 epochs (avg ± 95% CI)
Penalized	†$\lambda_{pen} = 0.001/N$	0.1	–	–	–	18.75	–
	†$\lambda_{pen} = 0.002/N$	0.1	–	–	–	19.04	–
	$\lambda_{pen} = 0.001$	0.1	93.20 ±0.01	100.00 ±0.00	100.00 ±0.00	21.01	21.70 ±0.19
	$\lambda_{pen} = 0.001$	6	90.64 ±0.32	90.88 ±0.41	89.94 ±0.71	18.51	19.14 ±0.21
	$\lambda_{pen} = 0.002$	6	90.13 ±0.45	90.19 ±0.38	89.52 ±0.57	18.99	19.24 ±0.14
Constrained $g \in [1:12]$	$\epsilon_g = 100\%$	0.1	93.20 ±0.01	100.00 ±0.00	100.00 ±0.00	21.02	21.66 ±0.21
	$\epsilon_g = 100\%$	6	90.77 ±0.31	90.99 ±0.25	89.74 ±0.29	18.68	19.08 ±0.16
	$\epsilon_g = 70\%$	6	69.99 ±0.01	68.62 ±0.08	68.59 ±0.22	18.88	19.37 ±0.15

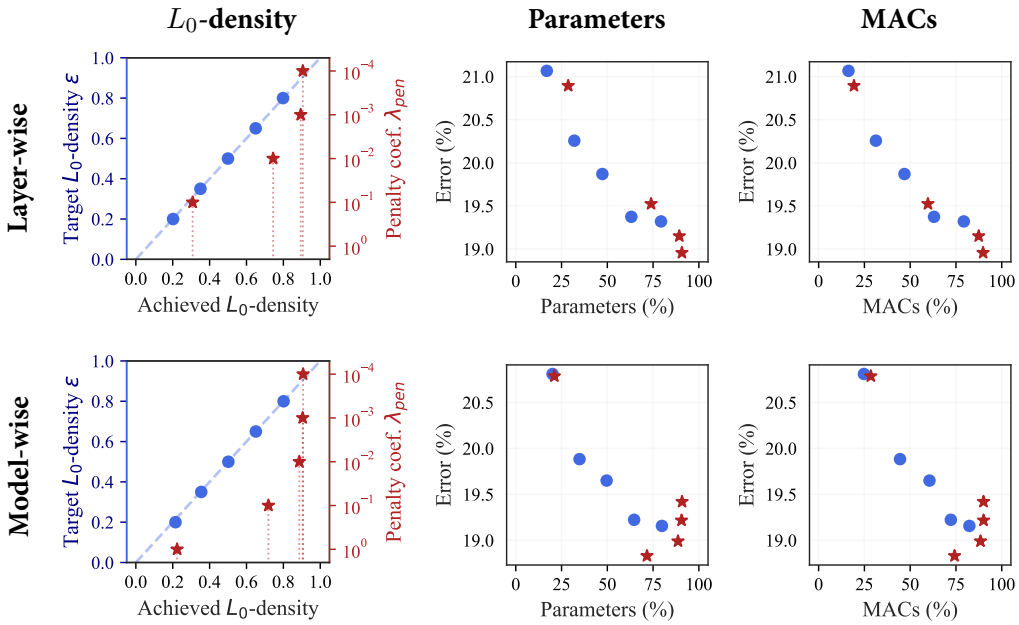


Figure A.11: Training sparse WideResNet-28-10 models on CIFAR-100.

Table A.13: Achieved density levels and performance for sparse WideResNet-28-10 models trained on CIFAR-100 for 200 epochs. Metrics aggregated over 5 runs.

Method	**Hyper-params**	L_0**-density** (%)	**Params** (%)	**MACs** (%)	**Val. Error** (%) best	**Val. Error** (%) at 200 epochs
Constrained $g \in [1:12]$ *Layer-wise*	$\epsilon_g = 20\%$	20.20 ±0.00	16.94 ±0.06	16.30 ±0.08	20.67	21.07 ±0.19
	$\epsilon_g = 35\%$	35.03 ±0.02	31.97 ±0.14	31.27 ±0.08	19.71	20.26 ±0.37
	$\epsilon_g = 50\%$	49.97 ±0.03	47.30 ±0.31	46.87 ±0.31	19.58	19.87 ±0.21
	$\epsilon_g = 65\%$	64.99 ±0.02	63.11 ±0.05	63.01 ±0.17	18.89	19.37 ±0.27
	$\epsilon_g = 80\%$	79.82 ±0.15	79.33 ±0.30	79.29 ±0.43	18.93	19.32 ±0.21
Penalized $g \in [1:12]$ *Layer-wise*	$\lambda^g_{pen} = 1$	——	*Failed due to sparsity collapse*		——	
	$\lambda^g_{pen} = 0.1$	30.80 ±0.54	28.63 ±0.52	19.30 ±0.20	20.40	20.89 ±0.29
	$\lambda^g_{pen} = 0.01$	74.46 ±0.41	73.90 ±0.39	59.69 ±0.36	19.03	19.52 ±0.15
	$\lambda^g_{pen} = 0.001$	89.38 ±0.12	89.31 ±0.17	87.42 ±0.40	18.72	19.15 ±0.33
	$\lambda^g_{pen} = 0.0001$	90.55 ±0.34	90.74 ±0.34	89.81 ±0.42	18.67	18.96 ±0.22
Constrained *Model-wise*	$\epsilon = 20\%$	21.50 ±0.19	20.10 ±0.29	24.72 ±0.31	20.42	20.81 ±0.07
	$\epsilon = 35\%$	35.46 ±0.01	34.82 ±0.11	44.46 ±0.76	19.29	19.88 ±0.16
	$\epsilon = 50\%$	50.16 ±0.01	49.63 ±0.09	60.54 ±0.88	19.25	19.65 ±0.17
	$\epsilon = 65\%$	65.08 ±0.04	64.64 ±0.14	72.20 ±0.60	18.86	19.22 ±0.25
	$\epsilon = 80\%$	80.12 ±0.03	79.82 ±0.04	82.33 ±0.46	18.91	19.16 ±0.25
Penalized *Model-wise*	$\lambda_{pen} = 1$	22.37 ±0.39	21.07 ±0.48	28.49 ±0.62	20.34	20.79 ±0.32
	$\lambda_{pen} = 0.1$	71.90 ±0.73	71.66 ±0.71	74.22 ±0.43	18.37	18.83 ±0.47
	$\lambda_{pen} = 0.01$	88.65 ±0.73	88.60 ±0.80	88.39 ±0.83	18.41	18.99 ±0.37
	$\lambda_{pen} = 0.001$	90.46 ±0.23	90.53 ±0.45	90.06 ±0.62	18.75	19.22 ±0.16
	$\lambda_{pen} = 0.0001$	90.70 ±0.33	90.83 ±0.35	90.03 ±0.32	19.02	19.42 ±0.10

A.11.4 *TinyImageNet*

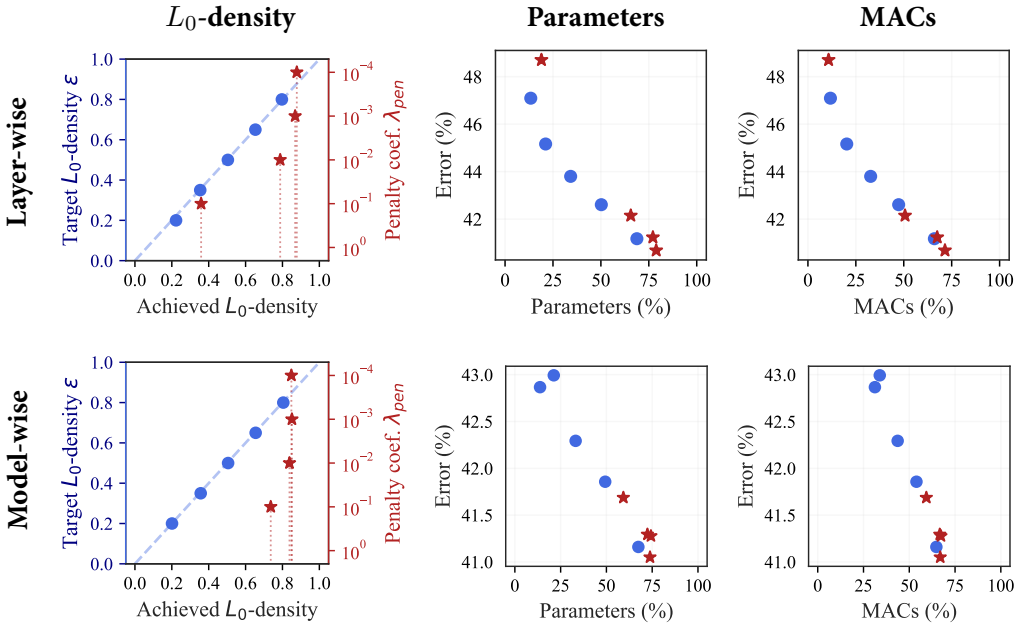


Figure A.12: Training sparse ResNet18 models on TinyImageNet. *This figure is the same as Fig. 4.1. We repeat it here for the reader's convenience.*

Table A.14: Achieved density levels and performance for sparse ResNet18 models trained on TinyImageNet for 120 epochs. Metrics aggregated over 3 runs.

Method	Hyper-params	L_0-density (%)	Params (%)	MACs (%)	Val. Error (%)	
					best	at 200 epochs
Constrained $g \in [1:16]$ *Layer-wise*	$\epsilon_g = 20\%$	22.99 ±0.53	13.40 ±0.54	11.75 ±0.45	46.12	47.09 ±0.15
	$\epsilon_g = 35\%$	35.41 ±0.07	21.16 ±0.03	20.25 ±0.16	43.87	45.16 ±0.31
	$\epsilon_g = 50\%$	50.40 ±0.04	34.22 ±0.28	32.73 ±0.49	42.52	43.80 ±0.44
	$\epsilon_g = 65\%$	65.32 ±0.42	50.22 ±0.54	47.34 ±0.66	41.80	42.61 ±0.10
	$\epsilon_g = 80\%$	79.68 ±0.88	68.78 ±1.70	65.98 ±1.53	40.55	41.17 ±0.61
Penalized $g \in [1:16]$ *Layer-wise*	$\lambda^g_{pen} = 1$	——	*Failed due to sparsity collapse*	——		
	$\lambda^g_{pen} = 0.1$	35.90 ±0.57	18.98 ±0.57	10.77 ±0.90	46.34	48.70 ±0.73
	$\lambda^g_{pen} = 0.01$	78.77 ±0.84	65.59 ±1.94	50.66 ±1.77	41.36	42.15 ±0.27
	$\lambda^g_{pen} = 0.001$	86.87 ±0.38	77.14 ±1.29	67.45 ±3.88	40.75	41.23 ±0.58
	$\lambda^g_{pen} = 0.0001$	87.77 ±0.42	78.85 ±0.91	71.49 ±2.36	40.32	40.68 ±0.04
Constrained *Model-wise*	$\epsilon = 20\%$	20.11 ±0.03	13.75 ±0.31	31.22 ±1.25	42.02	42.87 ±1.00
	$\epsilon = 35\%$	35.66 ±0.12	21.25 ±0.70	33.88 ±1.25	41.28	42.99 ±0.81
	$\epsilon = 50\%$	50.53 ±0.09	33.19 ±0.08	43.65 ±0.94	40.70	42.29 ±0.17
	$\epsilon = 65\%$	65.46 ±0.19	49.34 ±0.55	53.92 ±0.97	41.23	41.86 ±0.54
	$\epsilon = 80\%$	80.45 ±0.19	67.45 ±0.67	64.80 ±1.18	40.71	41.16 ±0.28
Penalized *Model-wise*	$\lambda_{pen} = 1$	——	*Failed due to sparsity collapse*	——		
	$\lambda_{pen} = 0.1$	73.64 ±0.72	59.27 ±0.76	59.33 ±2.19	40.84	41.69 ±0.16
	$\lambda_{pen} = 0.01$	83.80 ±0.08	72.41 ±0.67	66.64 ±1.26	40.33	41.29 ±0.50
	$\lambda_{pen} = 0.001$	85.12 ±0.89	74.31 ±1.72	67.30 ±1.62	40.71	41.28 ±0.56
	$\lambda_{pen} = 0.0001$	84.84 ±0.90	73.80 ±1.53	66.91 ±1.65	40.57	41.05 ±0.33

A.11.5 *ImageNet*

Here we provide further results for ResNet50 models on ImageNet, including experiments with model-wise constraints and the fine-tuned performance of the magnitude pruning method. We highlight that the controllability properties of our proposed constrained formulation extend to this large-scale setting (compare target density and L_0-density columns). The dense, pre-trained baseline, used as the starting point for magnitude pruning, corresponds to the `ResNet50_Weights.IMAGENET1K_V1` model made publicly available by Pytorch [Pas+19].

As expected, model-wise constraints allow for a more flexible allocation of the parameter budget throughout the network, thus leading to a better validation error. However, this flexibility can also result in models with larger memory and computational footprints. For example, with $\epsilon = 70\%$, the model learned with model-wise constraints has a very similar parameter count (64.41% vs 61.19%) but a significantly higher MAC count (76.50% vs 58.59%).

The results on magnitude pruning confirm the importance of the fine-tuning stage for this technique. The accuracy improves dramatically after a few epochs of retraining

Table A.15: Sparse ResNet50 models on ImageNet. "Fine-tuning" for zero epochs means *no* fine-tuning. *This table is the same as Table 4.2. We repeat it here for the reader's convenience.*

Target Density	**Method**	L_0**-density** (%)	**Params** (%)	**MACs** (%)	**Best Val. Error** (%) After fine-tuning for # epochs 0	1	10	20
–	Pre-trained Baseline	100	[25.5M]	$[4.12 \cdot 10^9]$	23.90	———	———	———
$\epsilon = 90\%$	Constrained *Model-wise*	90.36	88.06	91.62	**24.68**	———	———	———
	Constrained *Layer-wise*	90.58	87.07	85.97	24.97	———	———	———
	L1 - Mag. Prune *Layer-wise*	–	85.94	84.99	38.74	25.38	24.69	**24.68**
$\epsilon = 70\%$	Constrained *Model-wise*	70.78	64.41	76.50	**25.53**	———	———	———
	Constrained *Layer-wise*	70.36	61.91	58.59	26.98	———	———	———
	L1 - Mag. Prune *Layer-wise*	–	62.15	59.85	97.78	29.04	26.80	26.14
$\epsilon = 50\%$	Constrained *Model-wise*	50.18	42.47	58.00	**27.51**	———	———	———
	Constrained *Layer-wise*	50.70	43.15	38.25	27.89	———	———	———
	L1 - Mag. Prune *Layer-wise*	–	43.47	39.76	99.75	36.21	29.98	29.16
$\epsilon = 30\%$	Constrained *Model-wise*	30.31	31.81	42.05	**29.65**	———	———	———
	Constrained *Layer-wise*	31.44	30.16	23.74	31.71	———	———	———
	L1 - Mag. Prune *Layer-wise*	–	29.86	24.80	99.89	56.11	36.90	34.74

compared to the "just-pruned" model. Rather than a "pure" pruning technique, one can think of magnitude pruning as a method that, given a pretrained model, provides an *initialization* for a smaller model which needs to be trained (i.e. fine-tuned). At high density levels (e.g. 70-90%) the performance of the constrained L_0 formulation and fine-tuned magnitude pruning methods are located within a similar range. However, for harsher sparsity levels (30-50% density) the performance of models obtained using the constrained approach is significantly better than for magnitude pruning, even after fine-tuning.

A.12 Unstructured Sparsity

In this section we demonstrate that our constrained approach transfers successfully between the structured and unstructured sparsity regimes without major modifications. We carry out experiments using MLP and convolutional models on MNIST, and ResNet18

models on TinyImageNet.

Our unstructured experiments consider one gate *per model parameter*. This means that the number of gates in the unstructured setting is much larger than in the structured one, since it scales with the total number of model parameters and not with the number of units/output maps. The L_0 density of a layer with unstructured sparsity corresponds to the expected number of active gates within that layer.

We compare to a magnitude pruning baseline where a dense model is pre-trained, pruned in an unstructured way and fine-tuned. Since magnitude pruning is typically applied independently at each layer, we concentrate on experiments with layer-wise constraints.

A.12.1 *Experimental setup*

Throughout this section, the model architectures we use for experiments with unstructured sparsity match those of structured experiments detailed in Table A.2.

A.12.1.1 *MNIST*

Table A.16 presents the hyper-parameters used for training MLP and LeNet models on MNIST with unstructured sparsity. We train using a batch size of 128 and do not apply weight decay.

Table A.16: Configurations for MNIST experiments with unstructured sparsity.

Approach	Weights		Gates		Lagrange Multipliers		
	Optim.	$\eta^{\tilde{\boldsymbol{\theta}}}_{\text{primal}}$	Optim.	$\eta^{\boldsymbol{\phi}}_{\text{primal}}$	Optim.	η_{dual}	Restarts
Constrained	Adam	$7 \cdot 10^{-4}$	Adam	$1 \cdot 10^{-3}$	Grad. Ascent	10^{-3}	Yes
Magnitude Pruning	Adam	$7 \cdot 10^{-4}$	-	-	-	-	-

Models with L_0 gates. All the layers in these models have unstructured gates (one gate per weight entry and one gate per bias). The gate parameters are initialized using $\rho_{\text{init}} = 0.05$ (see Appx. A.1.2). We train these models for 200 epochs.

Magnitude pruning. For our magnitude pruning experiments we first trained a fully dense model for 200 epochs. We then apply unstructured pruning with the pre-determined target density, and retrain the resulting sparse model for another 200 epochs. We apply magnitude pruning to each of the layers in these models. Our magnitude pruning implementation keeps the biases fully dense.

A.12.1.2 *TinyImageNet*

Table A.17 presents the hyper-parameters used for learning sparse ResNet18 models on TinyImageNet. We use SGD with a momentum coefficient of 0.9 for the weights. The learning rate of the weights $\eta^{\tilde{\boldsymbol{\theta}}}_{\text{primal}}$ is multiplied by 0.1 at 30, 60 and 90 epochs. We use a batch size of 100.

Table A.17: Default configurations for TinyImageNet experiments with unstructured sparsity.

Approach	Weights		Gates		Lagrange Multipliers			Target	Weight
	Optim.	$\eta^{\tilde{\theta}}_{\text{primal}}$	Optim.	$\eta^{\phi}_{\text{primal}}$	Optim.	Restarts	η_{dual}	density	decay
Constrained	SGDM	0.1	Adam	$3 \cdot 10^{-2}$	Gradient Ascent	Yes	$9 \cdot 10^{-5}$	20%	$5 \cdot 10^{-4}$
							$2 \cdot 10^{-4}$	10%	
							$7 \cdot 10^{-4}$	5%	
							$2 \cdot 10^{-3}$	1%	
Magnitude Pruning	SGDM	$1 \cdot 10^{-4}$	-	-	-	-	-		$5 \cdot 10^{-4}$

Models with L_0 gates. The model's initial convolutional and final fully connected layers are kept fully dense. The residual connection of each `BasicBlock` in the model is kept fully dense, while all other convolutional layers use unstructured sparsity. This results in 16 sparsifiable convolutional layers. The gate parameters are initialized using $\rho_{\text{init}} = 0.05$ (see Appx. A.1.2) and optimized with Adam. We train these models for 120 epochs.

Magnitude pruning. For our magnitude pruning experiments we first trained a fully dense ResNet18 model for 120 epochs, using the weights learning rate schedule mentioned above. We then apply unstructured pruning with the pre-determined target density, and fine-tune the resulting sparse model for another 120 epochs using a fixed learning rate of $1 \cdot 10^{-4}$. We apply magnitude pruning to *the same layers* that were sparsifiable in models with L_0 gates. Our magnitude pruning implementation keeps the biases fully dense.

A.12.1.3 *Gates optimizer*

We originally tried the same optimization setup as with structured experiments (see Appx. A.10.5) for our unstructured experiments on TinyImageNet. Note that in the unstructured setting, there is a significantly larger number of gates whose parameters need to be optimized. Moreover, the influence of each individual gate on the sparsity of a layer is much smaller compared to the structured experiments. Thus the learning rates for the gates and the dual variables required to be tuned for these new unstructured tasks.

It was difficult to find a value of the gates learning rate that allowed the gates to move appropriately: (1) models trained with small learning rates would not achieve any sparsity (see Appx. A.1 for similar behavior in the structured sparsity regime); while (2) for larger gates learning rates we observed no decrease in model density for a long portion of training, followed by a sudden drop to the target density level. However, this sudden sparsification of the network caused a significant accuracy degradation. This behavior is consistent with the observations documented by Gale et al. [GEH19].

We hypothesize that training models with *unstructured* L_0 gates is highly susceptible to noise. In the unstructured case the information available for determining whether a gate should be active is mediated by its single associated parameter. Since we use mini-batch estimates of the model gradients, the training signal coming from this single parameter can be very noisy. In contrast, the structured setting has lower variance since each gate

aggregates information across a large group of parameters. Therefore, using an optimizer that is robust to this training noise is desirable.

We performed experiments with Adam as the optimizer for the model gates and this choice successfully delivered sparse models without breaking their predictive capacity.

Note that the experiments reported by Gale et al. [GEH19] did not use an adaptive optimizer for the gates. Exploring whether an adaptive optimizer like Adam would be sufficient to resolve the shortcomings of the L_0 reparametrization framework of Louizos et al. [LWK18] documented by Gale et al. [GEH19] is left as future work.

A.12.2 *Training dynamics*

In this section we explore the training dynamics of our proposed constrained formulation in the unstructured sparsity setting. We train a ResNet18 model on TinyImageNet with layer-wise constraints of 5% density. For other experimental settings see Appx. A.12.1.

Fig. A.13 shows the overall model density, the validation error, the density for a specific layer, and the Lagrange multiplier associated with the constraint for this layer. The behavior for the chosen layer is representative of that of other layers in the model. The density at the model and layer levels decreases stably to the desired target. Note that a high sparsity of 95% is achievable without causing irreparable damage to the model accuracy.

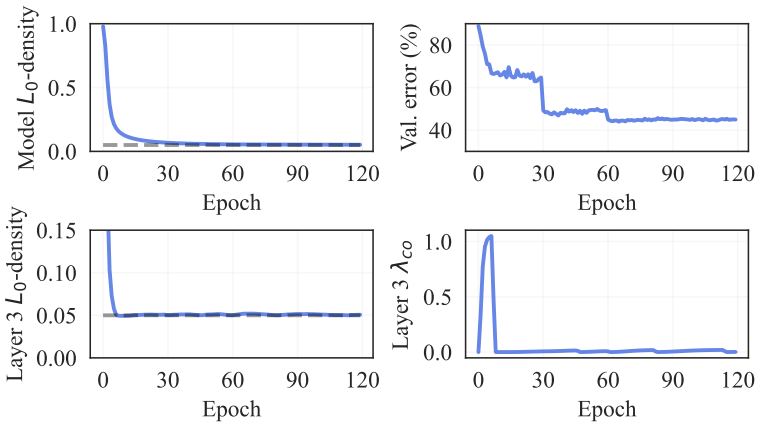


Figure A.13: Training dynamics for a ResNet18 model on TinyImageNet. This unstructured sparsity experiment uses layer-wise constraints with a target density of 5%. *Layer 3* corresponds to the first convolutional layer of the second `BasicBlock` of the first "layer/stage" of the ResNet18 model.

Overall, the training dynamics of our unstructured experiments are qualitatively equivalent to those of the structured experiments presented in Fig. A.2. This demonstrates that our constrained approach transfers successfully between structured and unstructured experiments.

A.12.3 *Performance comparison*

Tables A.18 and A.19 present the sparsity and performance statistics of MLP and LeNet models trained on MNIST at different target densities. We also include a (100%) dense model and layer-wise magnitude pruning experiments as baselines. For details on the experimental settings see Appx. A.12.1.

MNIST. Our proposed constrained approach consistently outperforms magnitude pruning even after fine tuning the magnitude pruning models for 200 epochs. Our approach reliably produces models with the desired density for experiments at 20%, 10%, and 5% density. The 1% density setting is challenging for both methods.

The constrained approach achieves high accuracy, although it incurs in a small violation of the sparsity constraint. Note that we employ the same dual learning rate across all densities. More extensive tuning of the dual learning rate can resolve this unfeasibility.

On the other hand, magnitude pruning experiments achieve the desired density at 1% (by design) but drastically fail in terms of performance. Note that the accuracy for magnitude pruning does not improve to an acceptable level even after fine-tuning for a large number of epochs. This behavior can be explained by the fact that our MNIST models have some layers with very few parameters. For example, at a 1% sparsity, the first convolutional layer of the model has only 5 active parameters.

The very low number of parameters make the high-sparsity pruned models difficult to fine-tune. This observation is consistent with the poor gradient dynamics reported by Evci et al. [Evc+22] when training highly sparse networks. We would like to highlight that our proposed method applies sparsity to the same layers as magnitude pruning, yet achieves high levels of unstructured sparsity per layer with minimal accuracy reduction. This shows that very sparse models with high accuracy do exist, however they seem to be out of reach when simply fine-tuning a magnitude-pruned model.

Table A.18: Performance of MLP models trained with unstructured sparsity on MNIST. Unstructured magnitude pruning is applied independently at each layer to retain the desired target density. "Fine-tuning" for zero epochs means *no* fine-tuning. Metrics averaged across 3 runs.

Target Density $g \in [1:3]$	**Method**	L_0**-density** (%)	**Best Val. Error** (%)			
			After fine-tuning for # epochs			
			0	50	100	200
–	Dense Baseline	100.00	1.72	———		
$\epsilon_g = 20\%$	Constrained	20.00	1.45	———		
	Magnitude Pruning	–	3.81	2.03	1.93	1.89
$\epsilon_g = 10\%$	Constrained	10.02	1.51	———		
	Magnitude Pruning	–	9.31	2.63	2.57	2.48
$\epsilon_g = 5\%$	Constrained	5.05	1.64	———		
	Magnitude Pruning	–	30.68	3.69	3.69	3.69
$\epsilon_g = 1\%$	Constrained	2.62	1.92	———		
	Magnitude Pruning	–	90.45	60.82	60.69	55.45

Table A.19: Performance of LeNet models trained with unstructured sparsity on MNIST. Unstructured magnitude pruning is applied independently at each layer to retain the desired target density. "Fine-tuning" for zero epochs means *no* fine-tuning. Metrics averaged across 3 runs.

Target Density $g \in [1:4]$	**Method**	L_0-**density** (%)	**Best Val. Error** (%)			
			After fine-tuning for # epochs			
			0	50	100	200
–	Dense Baseline	100.00	0.85	———		
$\epsilon_g = 20\%$	Constrained	19.99	0.73	———		
	Magnitude Pruning	–	2.28	0.98	0.92	0.92
$\epsilon_g = 10\%$	Constrained	10.00	0.78	———		
	Magnitude Pruning	–	5.23	1.38	1.38	1.32
$\epsilon_g = 5\%$	Constrained	5.01	0.89	———		
	Magnitude Pruning	–	12.53	2.39	2.39	2.39
$\epsilon_g = 1\%$	Constrained	1.55	1.26	———		
	Magnitude Pruning	–	88.76	88.76	88.76	88.76

TinyImageNet. Table A.20 displays the result for TinyImageNet experiments at 1%, 5%, 10% and 20% unstructured sparsity. We observe similar patterns as in the MNIST experiments: the constrained approach reliably achieves the desired sparsity targets and preserves reasonable performance.

Table A.20: Performance of ResNet18 models trained with unstructured sparsity on TinyImageNet. "Fine-tuning" for zero epochs means *no* fine-tuning.

Target Density $g \in [1:16]$	**Method**	L_0-**density** (%)	**Best Val. Error** (%)			
			After fine-tuning for # epochs			
			0	40	80	120
–	Dense Baseline	100.00	38.64	———		
$\epsilon_g = 20\%$	Constrained	20.26	42.06	———		
	Magnitude Pruning	–	43.45	39.81	39.35	39.27
$\epsilon_g = 10\%$	Constrained	10.57	42.54	———		
	Magnitude Pruning	–	54.06	42.19	41.45	41.25
$\epsilon_g = 5\%$	Constrained	5.20	43.98	———		
	Magnitude Pruning	–	75.44	46.00	44.21	43.75
$\epsilon_g = 1\%$	Constrained	1.87	47.24	———		
	Magnitude Pruning	–	99.21	81.67	72.95	69.00

In this task magnitude pruning outperforms the constrained approach, except for the case of 1% density. We hypothesize that the improvement of magnitude pruning at relatively larger density targets (10% and 20%) may be due to the significantly larger size of the ResNet18 model compared to those used for MNIST, and thus easier to fine-tune.

Finally, note that the performance of the constrained approach in the harsh 1% density setting is significantly better (albeit with a small violation of the constraint target) than that of magnitude pruning, even 120 epochs of fine-tuning for magnitude pruning.

Appendix to the Second Contribution

B

B.1 Stochastic Gates

Let $U_j \sim \text{Unif}(0,1)$ and $0 < \beta < 1$. A concrete random variable [MMT17; JGP17] $s_j \sim q(\cdot \mid (\phi_j, \beta))$ can be obtained by transforming the uniform random variable as:

$$s_j = \text{Sigmoid}\left(\frac{1}{\beta}\log\left(\frac{\phi_j\, U_j}{1-U_j}\right)\right). \tag{B.1}$$

Given hyper-parameters $\gamma < 0 < 1 < \zeta$, the hard concrete distribution [LWK18] corresponds to a stretching and clamping of a concrete random variable. The hard concrete distribution is a mixed distribution with point masses at 0 and 1, and a continuous density over $(0,1)$.

$$\boldsymbol{z} = \text{clamp}_{[0,1]}(\mathbf{s}(\zeta-\gamma)+\gamma)) \tag{B.2}$$

Table B.1 specifies the values of the fixed parameters associated with the gates employed throughout this work, following Louizos et al. [LWK18].

Table B.1: Fixed parameters of the hard concrete distribution.

Parameter	γ	ζ	β
Value	-0.1	1.1	2/3

We use $\boldsymbol{z}$ for modeling the stochastic gates of L_0onie models. The stochastic nature of hard concrete variables entails a model which is itself *stochastic*. For computing forwards and BPP measurements, we replace each gate by its median $\hat{\boldsymbol{z}}(\boldsymbol{\phi})$:

$$\hat{\boldsymbol{z}}(\boldsymbol{\phi}) = \min\left(1,\ \max\left(0,\ \text{Sigmoid}\left(\frac{\log(\boldsymbol{\phi})}{\beta}\right)(\zeta-\gamma)+\gamma\right)\right) \tag{B.3}$$

Gate medians are a deterministic function of the trainable parameter $\boldsymbol{\phi}$. Moreover, the stretching and clamping enables the medians to attain the values 0 or 1, thus producing a sparse network.

However, gate medians are poorly suited for setting up the BPP constraint as they do not allow for change once a gate is "fully turned off or on"; once the median is zero, the gradient with respect to ϕ_j is zero. Hence, we use expected BPP of the model as a

surrogate for computing gradients. Note that this quantity is differentiable with respect to ϕ_j.

$$\mathbb{E}_{\boldsymbol{z}|\boldsymbol{\phi}}\left[\,\texttt{BPP}(\tilde{\boldsymbol{\theta}} \odot \boldsymbol{z})\right] = \sum_i \texttt{bits}(\tilde{\boldsymbol{\theta}}_i \odot \boldsymbol{z}_j)\,\mathbb{P}[z_i \neq 0] \tag{B.4}$$

$$= \sum_i \texttt{bits}(\tilde{\boldsymbol{\theta}}_i)\,\text{Sigmoid}\left(\log(\phi_i) - \beta \log \frac{-\gamma}{\zeta}\right) \tag{B.5}$$

B.2 PROXY-CONSTRAINTS

Let us recall the min-max Lagrangian optimization problem from Eq. (6.3)

$$\tilde{\boldsymbol{\theta}}^*, \boldsymbol{\phi}^*, \lambda_{\text{co}}^* \triangleq \underset{\tilde{\boldsymbol{\theta}},\boldsymbol{\phi}}{\text{argmin}}\ \underset{\lambda_{\text{co}}\geq 0}{\text{argmax}}\ \mathcal{L}_{\text{I}}(\tilde{\boldsymbol{\theta}} \odot \hat{\boldsymbol{z}}(\boldsymbol{\phi})) + \lambda_{\text{co}}\left(\texttt{BPP}(\texttt{cast}(\tilde{\boldsymbol{\theta}} \odot \hat{\boldsymbol{z}}(\boldsymbol{\phi})) - \tau_{\texttt{BPP}}\right).$$

Applying simultaneous gradient descent-ascent on this problem corresponds to the update scheme:

$$[\tilde{\boldsymbol{\theta}}^{t+1}, \boldsymbol{\phi}^{t+1}] \triangleq [\tilde{\boldsymbol{\theta}}^{t}, \boldsymbol{\phi}^{t}] - \eta_{\text{primal}}\,\nabla_{[\tilde{\boldsymbol{\theta}},\boldsymbol{\phi}]}\left[\mathcal{L}_{\text{I}}(\tilde{\boldsymbol{\theta}} \odot \hat{\boldsymbol{z}}(\boldsymbol{\phi})) + \lambda_{\text{co}}\texttt{BPP}(\texttt{cast}(\tilde{\boldsymbol{\theta}} \odot \hat{\boldsymbol{z}}(\boldsymbol{\phi}))\right] \tag{B.6a}$$

$$\lambda_{\text{co}}^{t+1} \triangleq \max\left(0, \lambda_{\text{co}}^{t} + \eta_{\text{dual}}\,\left(\texttt{BPP}(\texttt{cast}(\tilde{\boldsymbol{\theta}} \odot \hat{\boldsymbol{z}}(\boldsymbol{\phi})) - \tau_{\texttt{BPP}}\right)\right) \tag{B.6b}$$

The gradient update for λ_{co} matches the value of constraint violation. Whenever the constraint is not satisfied, the Lagrange multiplier increases. We employ the dual restarts technique of GALLEGO-POSADA et al. [Gal+22] when the constraint is satisfied.

Note that primal update involves computing the gradient of the $\texttt{BPP}(\texttt{cast}(\tilde{\boldsymbol{\theta}} \odot \hat{\boldsymbol{z}}(\boldsymbol{\phi}))$ with respect to the parameters $\tilde{\boldsymbol{\theta}}$ and $\boldsymbol{\phi}$. As mentioned in Appx. B.1, this quantity is not differentiable when the gate medians attain the values 0 or 1. To overcome this issue we employ the *expected* BPP as a surrogate or *proxy-constraint* [Cot+19b, §4.2] when computing the gradient for the primal update. The original (non-proxy) constraint is used for updating the value of the Lagrange multiplier.

B.3 EXPERIMENTAL DETAILS

Our implementation is developed in Python 3.8, using Pytorch 1.11 [Pas+19] and the Cooper constrained optimization library [Gal+24]. We provide scripts to replicate our experiments at: `https://github.com/juan43ramirez/l0onie`.

B.3.1 *Codec baselines*

The baseline results for the JPEG [Wal92] standard were obtained using the CompressAI library [Bég+20].

B.3.2 *Model architectures*

We considered the same setup as Dupont et al. [Dup+21]: MLPs with 2 input dimensions corresponding to the position of input pixels normalized to lie in $[-1, 1]$ and 3 outputs corresponding to RGB values normalized to lie in $[0, 1]$. These models are trained at single (`float32`) precision.

All our models use Sine activations at every layer except for the output layer. These are setup with a fixed angular frequency of $\omega_0 = 30$.

When replicating the experiments of Dupont et al. [Dup+21], we consider specific architectures so as to yield bits-per-pixel of 0.07, 0.15, 0.3 and 0.6 at half (`float16`) precision. Table B.2 contains the hidden layers associated with each of these architectures.

Table B.2: Architectures used throughout this paper, along with their respective bits-per-pixel when used to compress an image of 768×512 pixels. BPPs are calculated for each architecture at single (`float32`) and half (`float16`) precision.

Hidden layers	**Width of Layers**	**BPP @ `float32`**	**BPP @ `float16`**
5	20	0.15	0.07
5	30	0.3	0.15
10	28	0.6	0.3
10	40	1.2	0.6
10	40	1.62	0.81

For L_0onie and magnitude pruning experiments, we consider *unstructured* sparsity, where each parameter can be pruned individually. We also consider *target BPPs* of compressed models at the same levels of 0.07, 0.15, 0.3 and 0.6 used to evaluate the COIN approach. Furthermore, the *initial* (dense) architectures which are sparsified by L_0onie and magnitude pruning to meet these target BPPs are also based on the architectures presented in Table B.2. We consider the next larger architecture from the one whose BPP matches the one targeted. This is illustrated in Table B.3.

Table B.3: Initial architectures considered when applying L_0onie and magnitude pruning to meet various target BPPs. Note that the initial architecture has a *larger* BPP than the desired target.

Target BPP (@ `float16`)	**Initial Architecture**		
	BPP (@ `float16`)	Hidden Layers	Width of Layers
0.07	0.15	5	30
0.15	0.3	10	28
0.3	0.6	10	40
0.6	0.81	13	40

B.3.3 *Magnitude pruning*

All magnitude pruning experiments follow the procedure presented in this section. First, we train a COIN model at a certain (larger) BPP. This serves as a baseline which can then be pruned to achieve a lower BPP. The selection of the initial architecture depends on the provided target BPP. The details about this choice are presented in Table B.3.

Thereafter, we loop over the layers of the baseline model, sorting their individual parameters based on magnitude. A suitable proportion of the parameters with the smaller magnitude is set to 0. We perform the sorting and pruning separately for weight matrices and biases.

We tried two variants for maginitude pruning: ① applying the *same* level of pruning for all the layers, or ② keeping the first and last layers fully dense. This last technique is commonly used in conjunction with magnitude pruning, and is particularly important in the setting of this work since the number of input dimensions is very low.

The results of method ① were significantly and consistently worse than those of method ②. The results reported in this paper correspond to method ②.

Immediately after pruning we evaluate the performance of the model in terms of PSNR. We identified a very significant degradation in performance and thus perform fine-tuning on the remaining parameters using the same number of iterations as the original training for the COIN baseline. The optimization hyper-parameters employed during this stage are presented in B.3.5.

B.3.4 *Parameter initialization*

We use the same method proposed by [LWK18] for initializing the learnable parameters ϕ. We leverage this scheme to ensure that the expected value of stochastic gates is centered around 0.5 at initialization.

The network's weights and biases are initialized in accordance to their use of sine activation functions, following the procedure described in [Sit+20]. This involves setting them based on a uniform distribution in $[-a, a]$. As mentioned previously, for L_0onie experiments, the gate distributions are initialized to be symmetric around 0.5, thus shrinking the effective value of parameters to lie in $[-a/2, a/2]$. We counter this by adjusting the initialization of weights and biases of L_0onie models to lie in $[-2a, 2a]$, thus having effective values of parameters in the desired range.

B.3.5 *Optimization hyper-parameters*

This section presents the optimization hyper-parameters used throughout our work. Table B.4 indicates those considered when training COIN models and for the fine-tuning of models after magnitude pruning. These are the same across different images and architectures. Both cases employ the same configuration for the optimizer: Adam [**adam**] with $(\beta_1,\ \beta_2) = (0.9,\ 0.999)$, as is default in Pytorch.

Table B.4: Optimization hyper-parameters for training COIN baselines and fine-tuning models which have been magnitude-pruned.

Approach	Train Precision	Decode Precision	Steps	Optim.	$\eta^{\tilde{\boldsymbol{\theta}}}_{\text{primal}}$
COIN MP fine-tuning	`float32`	`float16`	50.000	Adam	$2 \cdot 10^{-4}$

Note that the constraint functions considered for L_0onies involve expectations but can be computed in closed-form based on the parameters of the gates, as presented in Appx. B.1. Therefore, the computation of constraint violations is deterministic. We employ gradient *ascent* on the Lagrange multipliers. We initialize all Lagrange multipliers at zero. Details on the chosen dual learning rate, along with the use of dual restarts for L_0onie experiments are presented in Table B.5.

We use the same configurations across all Kodak images for a given target BPP. Since our goal is to "overfit" the model to the image, we do not use weight decay or other regularization techniques.

Table B.5: Optimization hyper-parameters for training L_0onie models.

Target	Initial	Weights		Gates		Lagrange Multipliers		
BPP	Architecture	Optim.	$\eta^{\tilde{\boldsymbol{\theta}}}_{\text{primal}}$	Optim.	$\eta^{\boldsymbol{\phi}}_{\text{primal}}$	Optim.	η_{dual}	Restarts
0.07	2 - 5×[30] - 3	Adam	10^{-3}	Adam	$7 \cdot 10^{-4}$	Grad. Ascent	$7 \cdot 10^{-3}$	Yes
0.15	2 - 10×[28] - 3						$3 \cdot 10^{-3}$	
0.3	2 - 10×[40] - 3						$1 \cdot 10^{-3}$	
0.6	2 - 13×[40] - 3						$8 \cdot 10^{-4}$	

B.4 Additional Results

We evaluate the performance of our approach when compressing each image in the Kodak dataset at 0.07, 0.15, 0.3 and 0.6 BPPs. In addition, we did the same for COIN and magnitude pruning plus fine-tuning. Fig. B.1 presents the best PSNRs we obtained.

Magnitude pruning always underperforms as opposed to L_0onie and COIN. This gap is more pronounced at 0.07 BPP, but is consistent across images. Moreover, the performance of L_0onie is generally competitive to that of COIN, whilst being slightly superior in the 0.15 and 0.3 BPP cases.

B.5 Qualitative Results

We provide qualitative comparisons for some images of the Kodak dataset of different levels of compression difficulty, according to the histograms in Fig. B.1. The image grids in Figs. B.2 to B.5 below are generated from reconstructions at various BPP budgets for L_0onie, COIN, magnitude pruning and JPEG. The experimental configuration for these experiments is provided in Appx. B.3.

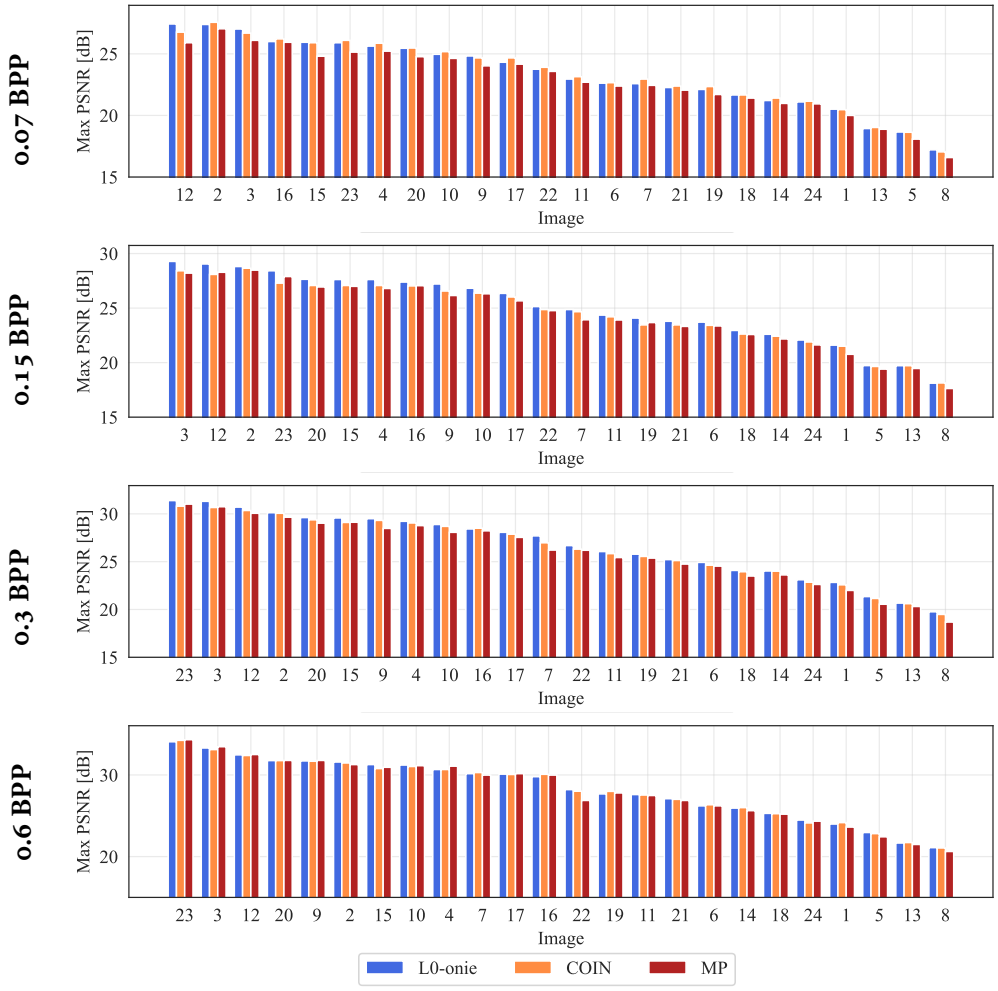


Figure B.1: Histogram of best PSNR for all images in the Kodak dataset for L_0onie, COIN and magnitude pruning at different compression rates. Images sorted in decreasing order of maximum PSNR for L_0onie at the end of trainig.

To reduce the file size of this manuscript, we only include *compressed* versions of the reconstruction grids. We invite the interested reader to consult the online version of this work* for the full, uncompressed set of images, including the original images and the reconstructions at different BPPs.

* *Available at* https://arxiv.org/abs/2207.04144

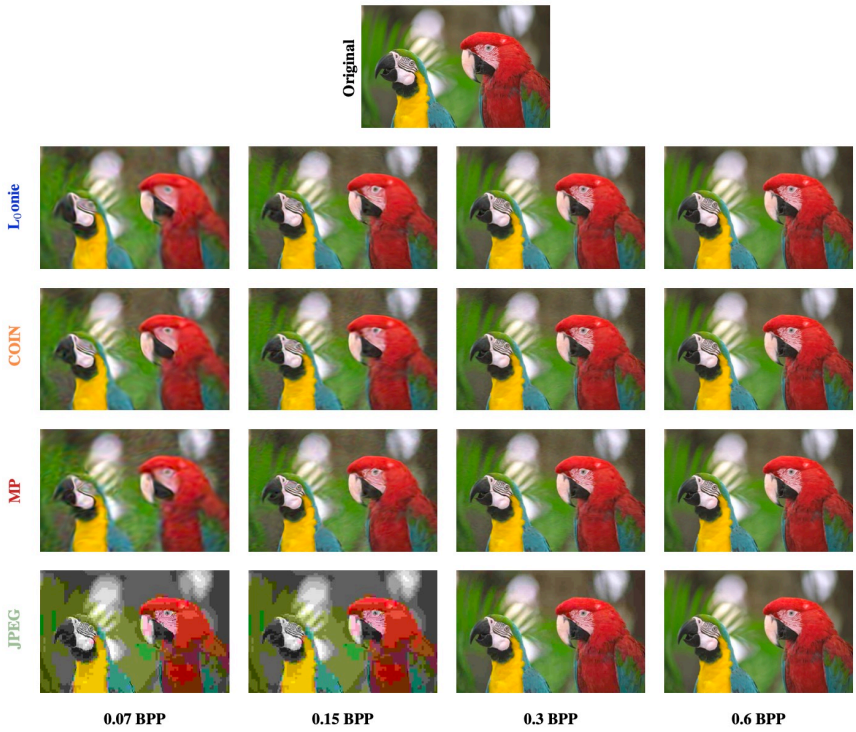


Figure B.2: Reconstruction of Kodak image 23 for all methods at different target BPP.

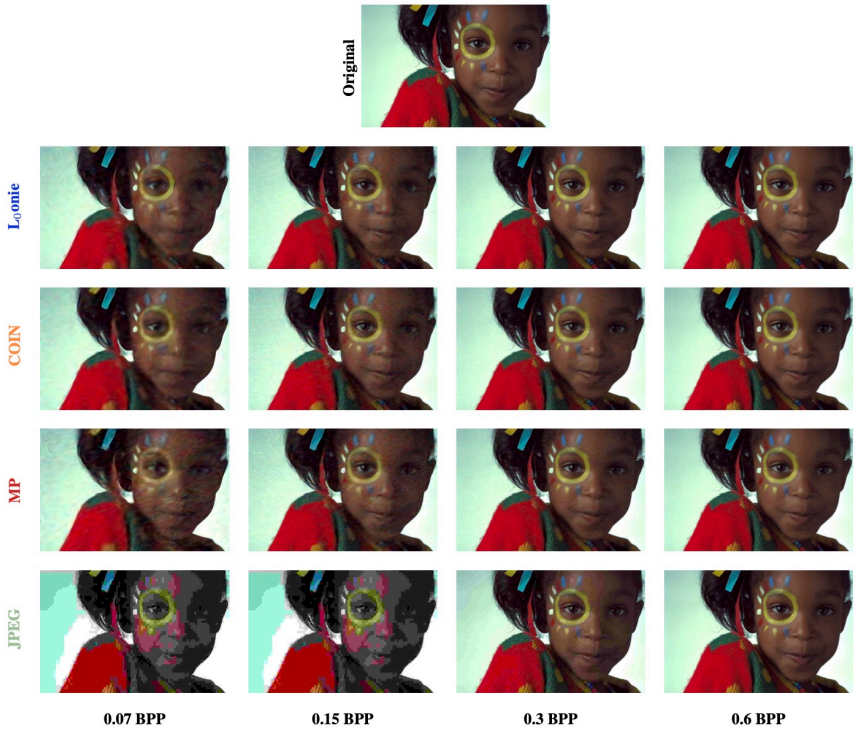


Figure B.3: Reconstruction of Kodak image 15 for all methods at different target BPP.

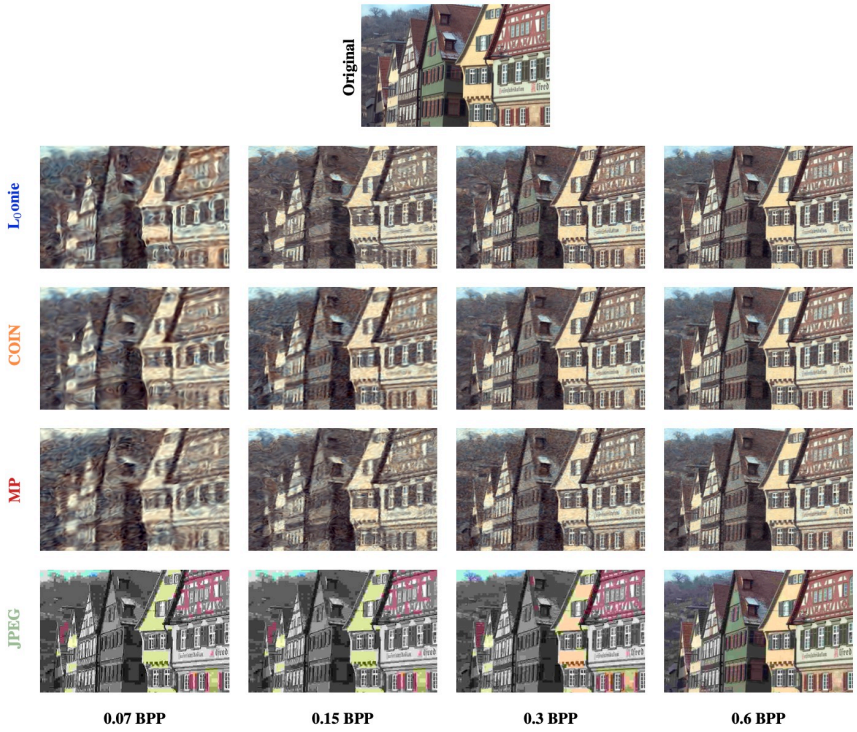


Figure B.4: Reconstruction of Kodak image 8 for all methods at different target BPP.

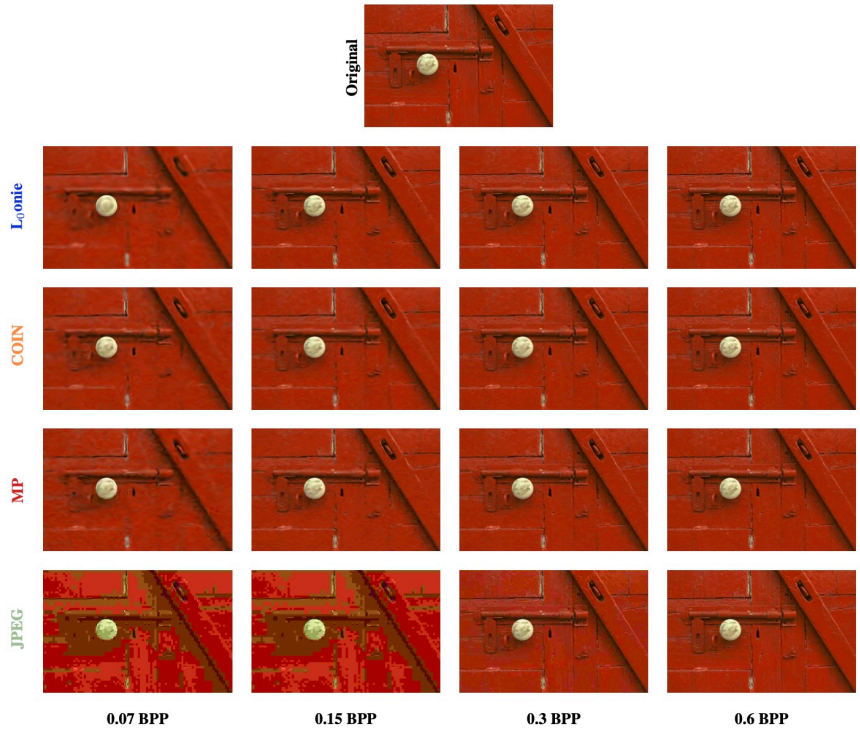


Figure B.5: Reconstruction of Kodak image 2 for all methods at different target BPP.

Appendix to the Third Contribution

C

C.1 Constrained and min-max optimization

Lagrangian-based constrained optimization has gained popularity in machine/deep learning owing to the fine-grained control it provides over specific properties of models [Cot+19b; SAA20; ENR22; Gal+22; HCR23]. In the context of our work, attaining a desired disparity level can be done directly by imposing the excess accuracy gap constraints presented in Eq. (8.6). In contrast, achieving bounded disparity by augmenting the training objective with *additive penalties* is challenging as it requires iteratively tuning a penalty coefficient per group [Gal+22].

The Lagrangian-based approach involves solving a non-convex-concave min-max optimization problem. In general, as long as the constraints are differentiable, the min-max problem can be optimized with gradient-based updates. Fortunately, [Cot+19b] show how even when the constraints are non-differentiable (but differentiable surrogates are available) *proxy constraints* can be used to find a semi-coarse correlated equilibrium of the min-max problem.

The solution to the min-max optimization problem (i.e. a saddle point) associated with the Lagrangian corresponds to a global constrained minimizer of the original constrained problem [Ber16]. However, a saddle point of the Lagrangian may not exist for non-convex problems [Cot+19b; Neu28].

In the context of machine learning, the success of adversarial formulations such as GANs [Goo+14] and adversarial training [Mad+18] has sparked interest in min-max optimization. [LJJ20] prove local linear convergence for simultaneous gradient descent-ascent in the non-convex-concave setting. Moreover, [Zha+22] prove local linear convergence of Alt-GDA in the strongly-convex-concave setting. They observe that the iteration complexity of Alt-GDA is optimal [MOP20a], thus matching that of extragradient [Kor76; Gid+19b]. These observations motivate our choice of Alt-GDA for optimizing Eq. (8.6).

Recent work has studied the statistical properties of constrained optimization problems [CR20; Cha+22]. This line of work has formulated PAC generalization bounds on feasibility and optimality, arguing that learning with constraints is *not* a more difficult problem than learning without constraints [CR20].

C.2 Alternative constrained formulations

This section elaborates on alternative constrained formulations for mitigating the disparate impact of pruning. Appx. C.2.1 presents the equalized loss formulation of [Tra+22],

Appx. C.2.2 describes a problem that constrains *excess loss gaps* of the sparse model, and Appx. C.2.3 formulates problems that (approximately) equalize the per-group excess accuracy gaps.

C.2.1 *Equalized Loss*

Equation (C.1) presents the equalized loss formulation for mitigating disparate impact [Tra+22]. This formulation matches the loss of each group with the overall loss.

$$\underset{\theta\in\Theta}{\operatorname{argmin}}\ L(\theta|\mathfrak{D}), \qquad \text{s.t.} \quad L(\theta|\mathfrak{D}_g) - L(\theta|\mathfrak{D}) = 0, \quad \forall g \in \mathcal{G} \tag{C.1}$$

[Tra+22] provide theoretical arguments to link disparate impact (in terms of group-level excess loss gaps) to the loss on each group. This justifies their choice of constraints.

Our implementation of this approach follows a pipeline akin to Algo. 2: we optimize it with alternating gradient descent-ascent and use group replay buffers to reduce variance in the estimation of the constraints for updating the dual variables. The storage cost associated with the buffer in this setting is higher than that for CEAG, since per-sample losses (floating point numbers) are stored instead of accuracies (booleans).

As shown in Appx. C.3.1, we notice smoother training dynamics for the multipliers when using the replay buffers. Table 8.3 shows how the equalized loss formulation benefits from them in terms of mitigating the disparate impact of pruning.

C.2.2 *Constrained Excess Loss Gaps*

An alternative to both CEAG and Eq. (C.1) is to constrain loss gaps between the dense and sparse models. This yields the following *constrained excess loss gaps* problem:

$$\begin{aligned} &\underset{\boldsymbol{\theta}_s\in\Theta}{\operatorname{argmin}}\ L(\boldsymbol{\theta}_s|\mathfrak{D}) \\ \text{s.t.} \quad &\tilde{\psi}_g = -\big(L(\boldsymbol{\theta}_d|\mathfrak{D}_g) - L(\boldsymbol{\theta}_s|\mathfrak{D}_g)\big) + \big(L(\boldsymbol{\theta}_d|\mathfrak{D}) - L(\boldsymbol{\theta}_s|\mathfrak{D})\big) \leq \epsilon, \quad \forall g \in \mathcal{G} \end{aligned} \tag{C.2}$$

This formulation addresses the disparate impact of pruning, although in terms of loss gaps instead of accuracy gaps. Selecting a tolerance ϵ for this formulation can be challenging as it requires specifying acceptable levels of excess loss gaps, which can vary significantly across tasks.

C.2.3 *Constrained Ψ_{PW}*

Constrained disparate impact. A natural formulation to consider involves constraining the disparate impact, as defined in Eq. (8.5a). The constrained optimization can be formulated as:

$$\underset{\boldsymbol{\theta}_s\in\Theta}{\operatorname{argmin}}\ L(\boldsymbol{\theta}_s|\mathfrak{D}), \qquad \text{s.t.} \quad \Psi_{\text{PW}} = \max_{g\in\mathcal{G}} \Delta_g\left(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d\right) - \min_{g\in\mathcal{G}} \Delta_g\left(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d\right) \leq \epsilon. \tag{C.3}$$

The constraint on Ψ_{PW} considers the difference between the most and least degraded groups. Therefore, when calculating the gradient of the Lagrangian, only the contribution from said extreme groups appears. This "lack of signal" may make optimization dynamics challenging, illustrated in Fig. C.1. The formulation successfully mitigates the disparate impact problem in the context of race prediction for the UTKFace dataset, which features 5 sub-groups. However, when confronted with the CIFAR-100 dataset, encompassing 100 sub-groups, gradient-based approaches to solve Eq. (C.3) are unable to identify a feasible solution. For both of these scenarios, we observed that the value of the (single) Lagrange multiplier grew continuously, without settling, confirming the previous intuition regarding poor optimization dynamics.

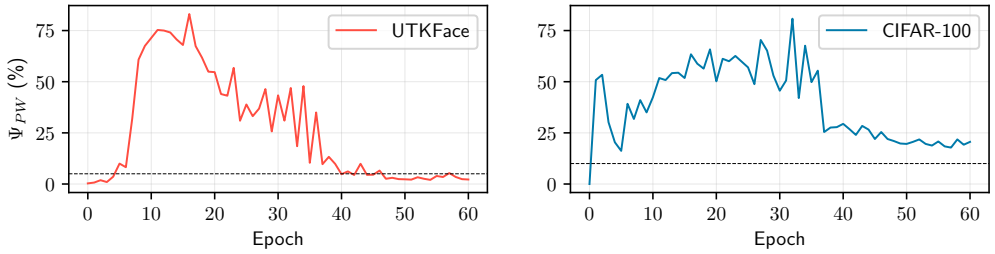


Figure C.1: Evolution of disparate impact of pruning (Ψ_{PW}) during training under Eq. (C.3). **Left:** UTKFace dataset at 92.5% sparsity. **Right:** CIFAR-100 dataset at 95% sparsity. The horizontal dashed lines indicate the tolerance (ϵ) of 5% and 10%, respectively.

A potential approach to alleviate this problem could be to introduce constraints on the pair-wise accuracy gaps:

$$\underset{\boldsymbol{\theta}_s \in \Theta}{\operatorname{argmin}} \; L(\boldsymbol{\theta}_s | \mathfrak{D}), \qquad \text{s.t.} -\epsilon \leq \Delta_g\left(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d\right) - \Delta_{g'}\left(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d\right) \leq \epsilon, \quad \forall g, g' \in \mathcal{G}. \tag{C.4}$$

However, this alternative formulation requires quadratically many constraints in the number of protected groups and does not scale to situations where the number of protected groups is large.

Equalized excess accuracy gaps. *Equalizing* the per-group excess accuracy gaps to zero gives rise to the following formulation:

$$\underset{\boldsymbol{\theta}_s \in \Theta}{\operatorname{argmin}} \; L(\boldsymbol{\theta}_s | \mathfrak{D}), \qquad \text{s.t.} \quad \psi_g = \Delta_g\left(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d\right) - \Delta\left(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d\right) = 0, \quad \forall g \in \mathcal{G}. \tag{C.5}$$

Compared to CEAG, this formulation (i) does not have an additional tolerance hyper-parameter ϵ, and (ii) prevents groups from having negative EAGs.

However, Eq. (C.5) can be challenging to solve because it may not have any feasible solutions; equalizing accuracy values may not be possible due to their discrete nature. Moreover, the lack of a tolerance hyper-parameter hurts flexibility as disparity requirements cannot be incorporated into the problem formulation.

Approximately equal excess accuracy gaps. A possible way to circumvent the limitations of Eq. (C.5) is to formulate the constrained problem:

$$\underset{\boldsymbol{\theta}_s \in \Theta}{\operatorname{argmin}}\ L(\boldsymbol{\theta}_s|\mathfrak{D}), \qquad \text{s.t.} \quad |\psi_g| = |\Delta_g\left(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d\right) - \Delta\left(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d\right)| \leq \epsilon, \quad \forall g \in \mathcal{G}. \tag{C.6}$$

Feasible solutions of Eq. (C.6) achieve $\Psi_{\text{PW}} \leq 2\epsilon$ by imposing both an upper and a lower bound on per-group EAGs. Compared to CEAG, this formulation prevents groups from experiencing a large improvement in performance compared to the global accuracy gap. Compared to Eq. (C.5), it allows for some tolerance at satisfying the equality. Naturally, for reasonable values of ϵ, Eq. (C.6) has a non-empty set of feasible solutions.

However, since the feasible set of Eq. (C.6) is small (as prescribed by ϵ), solving it is challenging, especially in the context of stochastic optimization. Mini-batch estimates of the constraints have a high chance of being infeasible due to the small feasible region and noise in the estimation. This leads to updates on the dual variables that are positive most of the time. In turn, this yields dual variables that perpetually increase and never stabilize.

Two-sided inequality. Alternatively, a two-sided inequality constrained optimization problem is:

$$\underset{\boldsymbol{\theta}_s \in \Theta}{\operatorname{argmin}}\ L(\boldsymbol{\theta}_s|\mathfrak{D}) \tag{C.7}$$

$$\text{s.t.} \quad \psi_g = \Delta_g\left(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d\right) - \Delta\left(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d\right) \leq \epsilon, \quad \forall g \in \mathcal{G} \tag{C.8}$$

$$-\psi_g = -\left(\Delta_g\left(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d\right) - \Delta\left(\boldsymbol{\theta}_s, \boldsymbol{\theta}_d\right)\right) \leq \epsilon, \quad \forall g \in \mathcal{G}. \tag{C.9}$$

This problem allows for *individual* dual variables to behave akin to those of CEAG. However, note how the two constraints for each EAG introduce conflicting terms to the gradient of $\boldsymbol{\theta}_s$: the model would aim to increase *or* decrease ψ_g depending on the current values of the dual variables.

Discussion. We focus on Eq. (8.6), and argue that constraining negative EAGs is not crucial for mitigating disparity. A side effect of this choice is allowing for sparse models whose group AGs are arbitrarily below the overall AG. In practice, this may lead to some groups improving their performance while the overall model accuracy decreases. We argue that this behavior is not problematic since it is only likely to manifest for underrepresented groups: groups with few samples can deviate in performance from other groups, without significantly influencing overall accuracy.

Eqs. (C.5) to (C.7) consider bounds on negative EAGs, but carrying out experiments on them is outside the scope of our work.

C.3 Replay Buffers

Algorithm 5 Update Buffer

Input: $\texttt{buf}_g$: Buffer for group g, $\hat{\mathbf{y}}$: A batch of model predictions, $\mathbf{y}$: The batch of true targets, $\texttt{idx}_g$: The sub-group indices of the batch.

function UpdateBuffer($\texttt{buf}_g, \hat{\mathbf{y}}, \mathbf{y}, \texttt{idx}_g$)
  $\texttt{SampleAcc} \leftarrow (\hat{\mathbf{y}} == \mathbf{y})[\texttt{idx}_g]$
  $\texttt{buf}_g \leftarrow \textsc{Push}\,(\texttt{buf}_g, \texttt{SampleAcc})$ {Drops old elements to respect capacity}
  return $\texttt{buf}_g$
end function

Algos. 5 and 6 contain functions for updating and querying the replay buffers, respectively. These are called by Algo. 2. Note that we wait until a buffer has been filled before considering its contents for computing the ψ_g terms. Before a buffer is filled, its corresponding ψ_g is 0.

For all groups, we consider the same buffer memory size k. Thus, the effective dataset used when computing EAGs of the sparse model is balanced across groups: it has k samples per group. However, when the original dataset is not balanced, this design implies that over-represented classes refresh their buffers faster as opposed to under-represented classes.

Algorithm 6 Query Buffers

Input: $\texttt{buf}_g, \forall g \in \mathcal{G}$: All replay buffers, k: Memory size for the replay buffers, A^g_{dense}: Accuracy of the dense model on each group g, A_{dense}: Aggregate accuracy of the dense model.

function QueryBuffers$\left(\{\texttt{buf}_g\}_{g=1}^{\mathcal{G}}, k, \{A^g_{\text{dense}}\}_{g=1}^{\mathcal{G}}, A_{\text{dense}}\right)$
  $\mathcal{I} \leftarrow \{\}$ {Indices of full buffers}
  for $g \in \mathcal{G}$ **do**
    if $\textsc{Len}(\texttt{buf}_g)$ **then**
      $\mathcal{I} \leftarrow \mathcal{I} \cup \{g\}$
      $\texttt{SampleAcc}_g \leftarrow \textsc{Query}\,(\texttt{buf}_g, k)$ {Query all elements of each buffer}
      $A^g_{\text{sparse}} \leftarrow \textsc{Average}\left(\texttt{SampleAcc}_g\right)$
    end if
  end for
  $A_{\text{sparse}} \leftarrow \textsc{Average}\left(\{A^g_{\text{sparse}}\}_{g\in\mathcal{I}}\right)$ {Compute aggregate from full buffers}
  for $g \in \mathcal{G}$ **do**
    if $g \in \mathcal{I}$ **then**
      $\psi_g \leftarrow (A^g_{\text{sparse}} - A^g_{\text{dense}}) - (A_{\text{sparse}} - A_{\text{dense}})$
    else
      $\psi_g \leftarrow 0$ {Ignore non-full buffers in ψ_g}
    end if
  end for
  return $\{\psi_g\}_{g=1}^{\mathcal{G}}$
end function

C.3.1 *Training Dynamics with Replay Buffers*

In Fig. C.2, we present the behavior of a select multiplier in a CIFAR-100 experiment with 90% sparsity. We depict two training stages: on the left, the multiplier consistently maintains a non-zero value, while on the right, it is frequently around zero. Multipliers are initialized at zero, and are expected to increase during the first stages of training when constraints may not be satisfied. Moreover, they are expected to eventually stabilize at a value (possibly zero) if their corresponding constraint is inactive at the solution.

On the left plot, we observe a smooth curve for the dual variable corresponding to the run with replay buffers. In contrast, the dual variable for the run without buffers is more noisy. On the right plot, the multiplier associated with the run without buffers becomes active more frequently than the multiplier of the run with buffers. Given the small magnitude of these multipliers (up to 0.003), the constraint may actually be feasible in this region and so the desirable behavior is to keep multipliers at 0.

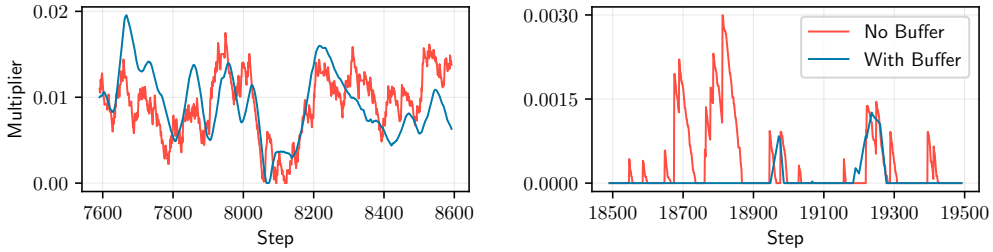


Figure C.2: Effects of replay buffers on the multiplier dynamics on CIFAR-100 under 90% sparsity. As expected, when using replay buffers, the multiplier exhibits notably smoother dynamics.

C.3.2 *Replay Buffer Size Ablation*

Table C.1: Effects of the memory size of replay buffers on a CIFAR-100 task at 95% sparsity. Not using a buffer yields poor results in terms of $\max_g \psi_g$. For experiments with buffers, different choices of memory sizes yield comparable results. We consider a tolerance of $\epsilon = 5\%$.

Buffer Size (k)	**Train**		**Test**	
	Accuracy	$\max_g \psi_g$	Accuracy	$\max_g \psi_g$
No Buffer	95.8 $\pm$ 0.15	5.8 $\pm$ 0.53	62.5 $\pm$ 0.41	17.1 $\pm$ 3.59
20	95.7 $\pm$ 0.10	5.4 $\pm$ 0.69	62.6 $\pm$ 0.30	16.0 $\pm$ 2.82
40	95.6 $\pm$ 0.12	5.7 $\pm$ 0.49	62.7 $\pm$ 0.28	14.8 $\pm$ 1.52
60	95.6 $\pm$ 0.16	5.5 $\pm$ 0.63	62.7 $\pm$ 0.26	14.5 $\pm$ 1.92
80	95.6 $\pm$ 0.17	5.5 $\pm$ 0.42	62.8 $\pm$ 0.44	16.4 $\pm$ 4.59

Table C.1 showcases the effect of the choice of buffer size in terms of accuracy and disparate impact. We observe that having a buffer is beneficial in terms of the train and

test $\max_g \psi_g$, while yielding models with similar accuracy to those obtained without replay buffers.

We observe that changing the buffer size has a small impact in terms of accuracy. In terms of $\max_g \psi_g$, the smallest (20) and largest (80) choices of buffer size result in more significant overfitting compared to 40 and 60. Moreover, the maximum EAG in test shows high variance in these cases. Table C.1 motivates our choice of $k = 40$ for most experiments.

C.4 Experimental details

Our implementations are in PyTorch 1.13.0 [Pas+19], with the Cooper library for Lagrangian constrained optimization [Gal+24].

Pipeline. As illustrated in Fig. 8.1, our pipeline consists of 3 stages: (i) obtaining a dense pre-trained model, (ii) pruning said model using gradual magnitude pruning, and (iii) fine-tuning the sparse model using either empirical risk minimization, the equalized loss formulation of [Tra+22], or our approach.

Dense models. Except for tasks involving the UTKFace dataset, we use publicly accessible pre-trained dense models. Appx. C.4.4 provides references to the pre-trained models we use throughout this work.

Pruning. We perform unstructured, layer-wise, gradual magnitude pruning [ZG17] with a cubic sparsity schedule (see Appx. C.4.5). We sparsify the weights of the model, but not the biases. We also do not sparsify the input and output layers of the model, as recommended by [GEH19]. See more details in Appx. C.4.3.

Tolerance level ϵ. We choose the tolerance level for each experiment by running NFT, measuring its corresponding $\max_g \psi_g$ and choosing a value of ϵ below this level. This protocol is ran independently for every task and every sparsity level. Finally, note that since EL imposes an equality constraint, there is no tolerance hyper-parameter to be chosen.

C.4.1 *Tasks and protected attributes*

We carry out experiments on the UTKFace [ZSQ17], FairFace [KJ21], and CIFAR-100 [Kri09] datasets; these respectively employ MobileNet-V2 [San+18], ResNet-34 [He+16], and CifarResNet-56 models [Che21], using different sparsity levels. These details are summarized in Table C.2.

We highlight the data transformations and the batch size we employ for each dataset in Table C.3.

Table C.2: Tasks considered throughout this work.

Dataset	Model	Predicted Attribute	Group Attribute	Sparsity
UTKFace	MobileNet-V2	Race	Race	85, 90, 92.5
		Gender	Race	85, 90, 92.5
		Race	Race ∩ Gender	85, 90, 92.5
FairFace	ResNet-34	Race	Race	99
		Gender	Race	99
		Race	Race ∩ Gender	99
CIFAR-100	CifarResNet-56	Class	Class	90, 92.5, 95

Table C.3: Transformations and batch sizes considered for each dataset.

Dataset	Transformations		Batch Size
	Train	Test	
UTKFace	`RandomHorizontalFlip(0.5)`	–	128
FairFace	`Resize(224,224)` `RandomHorizontalFlip(0.5)`	`Resize(224,224)`	256
CIFAR	–	–	128

C.4.2 *Mitigation schemes*

This section describes the approaches considered throughout this work for fine-tuning the sparse model, with or without a scheme to mitigate the disparate impact of pruning. We fine-tune sparse models on UTKFace and CIFAR for 45 epochs, and for 32 epochs on FairFace.

Naive Fine Tuning (NFT). The sparse model is fine-tuned on the training set using ERM.

Naive Fine Tuning with Early Stopping (NFT+ES). Obtained by selecting the best iterate of NFT in terms of *test* accuracy. We analyze this approach since early stopping is a popular technique in deep learning practice and, as evidenced by our experiments, often results in higher disparity (compared to the last iterate in NFT).

EL. Our implementation of the equalized loss method proposed by [Tra+22]. More details of this formulation can be found in Appx. C.2.1.

EL+RB. Enhanced version of EL employing replay buffers (§8.4.2) for updating the dual variables. The replay buffers store the per-sample losses observed at the mini-batch level across groups.

CEAG. Our constrained excess accuracy gap approach (see §8.4.3), which uses replay buffers by default.

C.4.3 *Model Architectures*

We employ MobileNet-V2 [San+18], ResNet-34, and CifarResNet-56 models [He+16]. ResNet-34 models are composed of bottleneck residual blocks [He+16], while CifarResNet-56 models use basic residual blocks [Che21].

Following [Evc+20], across all models, we do not sparsify the biases due to their low footprint towards the total number of parameters. We also do not sparsify the first and last layers of the model as recommended by [GEH19].

Table C.4 specifies the number of parameters of all considered architectures. We also provide the number of parameters remaining post-pruning across the considered sparsities for the reader's convenience.

Table C.4: Statistics on the total number of parameters and active parameters at different sparsity levels for our employed architectures. †Sparsifiable parameters indicate the number of parameters that *may* be removed during pruning (thus, excluding non-prunable parameters such as biases). ‡Parameter counts reported for MobileNet-V2 and ResNet-34 models are for race prediction tasks on UTKFace and FairFace, respectively.

Architecture	**Total Params**	**Sparsifiable Params**†	**Active parameters at sparsity:**				
			85%	90%	92.5%	95%	99%
MobileNet–V2‡	2,230,277	2,222,944	366,946	255,250	198,709	–	–
ResNet-34‡	21,288,263	21,275,136	–	–	–	–	241,751
CifarResNet-56	861,620	854,656	–	96,179	74,889	53,621	–

C.4.4 *Pre-Trained Models*

Reusing and fine-tuning of pre-trained deep learning models is a common practice. For example, a typical application pipeline might involve (i) obtaining a pre-trained model, (ii) fine-tuning it on an application-specific task, and (iii) pruning it before deployment.

Therefore, we concentrate on studying the behavior of mitigation techniques when applied to openly available pre-trained models.

- ResNet-34 models for FairFace use the weights provided by [KJ21].
- CifarResNet-56 models for CIFAR-100 use the weights provided by [Che21].
- We were unable to find publicly available pre-trained MobileNet-V2 models for the UTKFace dataset. Thus, we train these from scratch (see details below). *As part of our reproducibility efforts, we are making our pre-trained UTKFace MobileNet-V2 models openly available.*

For training UTKFace models, we use SGD with an initial learning rate of 0.01, decayed by a factor of 0.1 at training milestones of 60%, 80%, and 90% of total training epochs. We use a momentum coefficient of 0.9, and train for a total of 50 epochs. These hyper-parameters are used both for race and gender prediction tasks.

The group-wise performance for all the dense models is reported in Tables C.14, C.15, C.17, C.18, C.20, C.21, C.23, C.24, C.26, C.27, C.29 and C.30.

C.4.5 *Gradual Magnitude Pruning*

As mentioned in §8.5.1, our experiments perform Gradual Magnitude Pruning (GMP), where a fraction of the smallest weights (in magnitude) is pruned on every epoch. [ZG17] consider the following cubic schedule prescribing the proportion of parameters to prune at every epoch:

$$s_t = s_f + (s_i - s_f)\left(1 - \frac{t - t_0}{(T_{\text{end}} - t_0)\,\Delta t}\right)^3 \tag{C.10}$$

for $t \in \{t_0, t_0 + \Delta t, ..., t_0 + (T_{\text{end}} - t_0)\Delta t\}$, where t_0 is the initial training step, Δt is the pruning frequency (in epochs), T_{end} is final epoch of pruning, and s_i and s_f denote the initial and final sparsities, respectively.

Our experiments carry out GMP since epoch $t_0 = 0$, throughout $T_{\text{end}} = 15 - 1 = 14$ epochs, and perform pruning once every epoch ($\Delta t = 1$).

C.4.6 *Primal optimization hyper-parameters*

We make use of SGD with momentum as the primal optimizer for all of our experiments. In our initial ablation experiments on the choice of primal optimizer, we found that employing SGD with momentum outperformed or matched the performance of Adam [KB15].

For UTKFace and CIFAR-100 datasets, we employ a primal step size of $1 \cdot 10^{-2}$ along with a momentum of 0.9 (Polyak), and apply weight decay at the rate of $1 \cdot 10^{-4}$.

For FairFace, we employ Nesterov momentum with a step-size of $1 \cdot 10^{-3}$ and apply a weight decay of $1 \cdot 10^{-2}$. Specifically, for race prediction tasks, we utilize a momentum of 0.95, while for gender prediction tasks, a momentum of 0.99 is employed.

Additionally, we use PyTorch's `MultiStepLR` as the learning rate scheduler, with decay $\gamma = 0.1$ across all experiments. For UTKFace and CIFAR-100, we set scheduler milestones at 60%, 80% and 90% of the total training epochs (including the execution of GMP). For instance, for a task that has 60 epochs where it employs GMP on 15 epochs, the above milestones would activate at epoch 36, epoch 48 and epoch 54. For race prediction on FairFace we use a single milestone at 90%, while gender prediction on FairFace uses a constant learning rate of $1 \cdot 10^{-2}$.

C.4.7 *Dual optimization hyper-parameters*

We employ stochastic gradient *ascent* on the dual parameters (corresponding to the Lagrange multipliers) in all experiments. The choices of dual learning rate are presented in Tables C.5 to C.10.

We fix $k = 40$ as the memory size for the replay buffer. Preliminary ablations on the choice of $k \in [20, 80]$ showed low sensitivity to the specific value of this hyper-parameter (See Appx. C.3.2).

Note that the order of magnitude for the dual step-size choices is relatively consistent across datasets, tasks, sparsity levels and disparity tolerances. This highlights the ease of tuning exhibited by this hyper-parameter.

C.4.7.1 *UTKFace*

Table C.5: Tolerance and dual step-size for CEAG on UTKFace tasks.

Target Attribute	**Group Attribute**	**Sparsity**	**Dual Step-Size** (η_λ)	**Tolerance** ϵ **(%)**
Gender	Race	85	$2 \cdot 10^{-3}$	0.5
		90	$1 \cdot 10^{-4}$	0.5
		92.5	$3 \cdot 10^{-3}$	0.5
Race	Race	85	$1 \cdot 10^{-4}$	0.25
		90	$2 \cdot 10^{-3}$	1
		92.5	$2 \cdot 10^{-3}$	1
Race	Race ∩ Gender	85	$1 \cdot 10^{-5}$	0.5
		90	$1 \cdot 10^{-3}$	3
		92.5	$1 \cdot 10^{-3}$	3

Table C.6: Dual step-size for EL+RB on UTKFace tasks.

Target Attribute	**Group Attribute**	**Sparsity**	**Dual Step-Size** (η_λ)
Gender	Race	85	$1 \cdot 10^{-4}$
		90	$1 \cdot 10^{-5}$
		92.5	$1 \cdot 10^{-5}$
Race	Race	85	$1 \cdot 10^{-5}$
		90	$1 \cdot 10^{-5}$
		92.5	$1 \cdot 10^{-5}$
Race	Race ∩ Gender	85	$1 \cdot 10^{-5}$
		90	$1 \cdot 10^{-5}$
		92.5	$1 \cdot 10^{-5}$

C.4.7.2 *FairFace*

Table C.7: Tolerance and dual step-size for CEAG on FairFace tasks at 99% sparsity.

Target Attribute	**Group Attribute**	**Dual Step-Size** (η_λ)	**Tolerance** ϵ **(%)**
Gender	Race	$1 \cdot 10^{-5}$	1
Race	Race	$1 \cdot 10^{-4}$	2
Race	Race $\cap$ Gender	$1 \cdot 10^{-5}$	0.25

Table C.8: Dual step-size for EL+RB on FairFace tasks.

Target Attribute	**Group Attribute**	**Dual Step-Size** (η_λ)
Gender	Race	$1 \cdot 10^{-5}$
Race	Race	$1 \cdot 10^{-5}$
Race	Race $\cap$ Gender	$1 \cdot 10^{-5}$

C.4.7.3 *CIFAR*

Table C.9: Tolerance and dual step-size for CEAG on CIFAR tasks.

Target Attribute	**Group Attribute**	**Sparsity**	**Dual Step-Size** (η_λ)	**Tolerance** ϵ **(%)**
Class	Class	90	$2 \cdot 10^{-3}$	1
		92.5	$2 \cdot 10^{-3}$	2
		95	$1 \cdot 10^{-3}$	5

Table C.10: Dual step-size for EL+RB on CIFAR tasks.

Target Attribute	**Group Attribute**	**Sparsity**	**Dual Step-Size** (η_λ)
Class	Class	90	$1 \cdot 10^{-5}$
		92.5	$1 \cdot 10^{-5}$
		95	$1 \cdot 10^{-5}$

C.5 Additional Experiments

C.5.1 *Computational Overhead*

Table C.11 presents the wall-clock time of an experiment for different mitigation approaches on CIFAR-100 at 95% sparsity. Note that the reported time includes the 15 epochs of gradual magnitude pruning of the dense model, as well as the 45 epochs of fine-tuning.

Table C.11: Runtime of different mitigation approaches on CIFAR-100 at 95% sparsity. All runs are run on NVIDIA `A100-SXM4-80GB` GPUs. Runtimes are average across 5 runs for each mitigation method.

Method	Min		Median		Max	
	Wall-clock Time	Overhead wrt NFT	Wall-clock Time	Overhead wrt NFT	Wall-clock Time	Overhead wrt NFT
NFT	1h 0m 04s	1×	1h 3m 13s	1×	1h 12m 19s	1×
EL	1h 2m 37s	1.031×	1h 4m 34s	1.021×	1h 15m 50s	1.049×
EL+RB	1h 4m 15s	1.058×	1h 7m 10s	1.062×	1h 17m 35s	1.073×
CEAG (No RB)	1h 2m 08s	1.023×	1h 3m 08s	0.998×	1h 8m 35s	0.948×
CEAG	1h 1m 58s	1.020×	1h 4m 28s	1.020×	1h 6m 27s	0.919×

We observe a negligible increase in training time for constrained approaches that use replay buffers relative to NFT. For approaches that do not use replay buffers, the runtime is essentially the same as NFT. This overhead is especially insignificant considering that the CIFAR-100 problem involves 100 constraints.

C.5.2 *Comparison with FairGRAPE [LKJ22]*

Computational cost: As a benchmarking exercise, we re-ran the code provided by [LKJ22] for UTKFace Race prediction and Race as protected group*. The method took more than 90 hours of compute time on an NVIDIA `A100-SXM4-80GB` GPU. Given how prohibitively expensive this is, we refrained from running experiments with FairGRAPE. Furthermore, we expect the runtime to increase for tasks with larger numbers of protected groups.

* *The FairGRAPE implementation can be found here:* `https://github.com/Bernardo1998/FairGRAPE`

UTKFace: [LKJ22] apply their method on UTKFace, but remove race group *Others* from the dataset. The authors state that this was done as *Others* is an ambiguous class. Since we consider the complete dataset in our experiments, we can not compare directly to the numbers reported by [LKJ22].

Although we could apply CEAG to the UTKFace dataset without race group *Others*, we choose not to since we observe that this group is generally the most disproportionately affected by pruning. Table C.12 shows that NFT can achieve models with low disparity on UTKFace without *Others*, but presents significantly worse accuracy and higher disparity on experiments with *Others*.

Table C.12: NFT results on UTKFace race prediction with race as group attribute, with and without race group *Others*.

Setup	Train			Test		
	Accuracy	Ψ_{PW}	$\max_g \psi_g$	**Accuracy**	Ψ_{PW}	$\max_g \psi_g$
UTKFace (without *Others*)	99.5 $\pm$ 0.0	2.0 $\pm$ 0.16	0.5 $\pm$ 0.00	86.7 $\pm$ 0.55	10.9 $\pm$ 1.68	7.4 $\pm$ 0.14
UTKFace	98.2 $\pm$ 0.0	9.9 $\pm$ 0.82	8.7 $\pm$ 0.82	79.5 $\pm$ 0.46	6.1 $\pm$ 1.60	2.2 $\pm$ 0.68

C.5.3 *Sensitivity Analysis*

Table C.13 presents the sensitivity of our approach to the tolerance hyperparameter ϵ on a UTKFace race prediction task with race as group attribute.

Table C.13: Race prediction task for UTKFace with race as group attribute, at 92.5% sparsity. All experiments use a dual step size of $2 \cdot 10^{-3}$. Results are aggregated across 5 seeds.

Tolerance	**Accuracy**	$\max_g \psi_g$
1.5	93.6 $\pm$ 0.1	0.9 $\pm$ 0.61
1.0	93.4 $\pm$ 0.3	1.1 $\pm$ 0.36
0.5	93.2 $\pm$ 0.2	1.2 $\pm$ 0.44
0	93.2 $\pm$ 0.2	0.9 $\pm$ 0.24

We observe small improvements in performance for experiments with large tolerance values. For low tolerance values, we observe that feasibility is not attained. Moreover, the resulting $\max_g \psi_g$ values are similar across the considered tolerances. These observations are indicative of robustness to the choice of tolerance.

C.6 Comprehensive Experimental Results

This section contains the results corresponding to all experiments mentioned in Table C.2 across all datasets, sparsities, and tasks considered in our work. As mentioned earlier, all metrics reported in our tables and plots follow the pattern `avg` $\pm$ `std`. Unless mentioned otherwise, all our experimental metrics are aggregated across 5 seeds.

Some of the tables displayed below are extensions of tables presented in the main paper. Such tables have been clearly identified in the captions. For the tables reporting group accuracies, we report the numbers for both the dense model as well as all the sparse models.

C.6.1 *UTKFace*

We use MobileNet-V2 [San+18], similar to [LKJ22].

C.6.1.1 *Gender*

Table C.14: Groupwise train accuracy for gender prediction in UTKFace with race as protected attribute, across sparsities.

Sparsity	Method	White	Black	Asian	Indian	Others
85	Dense	100.0	99.9	99.9	99.8	99.3
	NFT	99.9 ± 0.02	99.8 ± 0.04	99.9 ± 0.06	99.8 ± 0.02	99.3 ± 0.1
	EL + RB	99.9 ± 0.01	99.8 ± 0.03	99.9 ± 0.03	99.8 ± 0.04	99.4 ± 0.13
	CEAG	99.9 ± 0.03	99.8 ± 0.06	99.9 ± 0.05	99.8 ± 0.03	99.3 ± 0.12
90	Dense	100.0	99.9	99.9	99.8	99.3
	NFT	99.8 ± 0.03	99.8 ± 0.04	99.7 ± 0.09	99.6 ± 0.1	98.9 ± 0.17
	EL + RB	99.8 ± 0.05	99.8 ± 0.05	99.7 ± 0.09	99.6 ± 0.08	99.0 ± 0.17
	CEAG	99.8 ± 0.02	99.8 ± 0.02	99.7 ± 0.12	99.7 ± 0.04	99.1 ± 0.1
92.5	Dense	100.0	99.9	99.9	99.8	99.3
	NFT	99.4 ± 0.07	99.5 ± 0.08	98.8 ± 0.22	99.1 ± 0.11	98.0 ± 0.23
	EL + RB	99.5 ± 0.05	99.6 ± 0.09	98.6 ± 0.23	99.1 ± 0.1	98.1 ± 0.45
	CEAG	99.1 ± 0.22	99.3 ± 0.28	99.6 ± 0.12	98.7 ± 0.11	98.5 ± 0.29

Table C.15: Groupwise test accuracy for gender prediction in UTKFace with race as protected attribute, across sparsities.

Sparsity	Method	White	Black	Asian	Indian	Others
85	Dense	94.2	95.0	89.5	93.2	89.5
	NFT	92.0 ± 0.42	94.0 ± 0.82	87.0 ± 0.62	92.5 ± 0.37	88.6 ± 0.83
	EL + RB	92.1 ± 0.45	94.2 ± 0.52	87.2 ± 0.72	92.1 ± 0.41	88.4 ± 0.4
	CEAG	91.9 ± 0.3	93.8 ± 0.41	86.7 ± 0.75	92.1 ± 0.55	88.1 ± 1.03
90	Dense	94.2	95.0	89.5	93.2	89.5
	NFT	91.1 ± 0.67	92.6 ± 0.57	85.9 ± 0.95	91.6 ± 0.67	87.1 ± 0.85
	EL + RB	91.0 ± 0.1	93.0 ± 0.38	86.4 ± 0.64	91.4 ± 0.45	87.6 ± 0.83
	CEAG	91.0 ± 0.56	93.2 ± 0.64	85.5 ± 0.73	91.8 ± 0.64	86.3 ± 1.03
92.5	Dense	94.2	95.0	89.5	93.2	89.5
	NFT	90.5 ± 0.67	92.6 ± 0.62	85.1 ± 0.83	91.0 ± 0.89	87.3 ± 1.05
	EL + RB	90.2 ± 0.54	93.2 ± 0.66	85.0 ± 1.57	90.6 ± 0.4	86.6 ± 1.35
	CEAG	90.6 ± 1.0	92.7 ± 0.2	85.8 ± 0.93	90.5 ± 0.76	87.1 ± 0.91

Table C.16: Gender prediction on UTKFace with race as group attribute, across sparsities. **CEAG consistently achieves a $\max_g \psi_g$ within the threshold, across sparsities.**

Sparsity	Method	Train				Test		
		Accuracy	Ψ_{PW}	$\max_g \psi_g$	Tol (ϵ)	Accuracy	Ψ_{PW}	$\max_g \psi_g$
85	NFT	99.8 $\pm$ 0.01	0.3 $\pm$ 0.15	0.0 $\pm$ 0.02	–	91.5 $\pm$ 0.25	2.2 $\pm$ 0.87	0.9 $\pm$ 0.43
	NFT + ES	98.0 $\pm$ 1.72	1.7 $\pm$ 1.19	1.3 $\pm$ 0.91	–	91.8 $\pm$ 0.28	2.1 $\pm$ 0.69	0.8 $\pm$ 0.23
	EL + RB	99.9 $\pm$ 0.02	0.3 $\pm$ 0.18	0.0 $\pm$ 0.01	–	91.5 $\pm$ 0.29	1.9 $\pm$ 0.71	0.9 $\pm$ 0.41
	CEAG	99.8 $\pm$ 0.01	0.3 $\pm$ 0.16	0.1 $\pm$ 0.05	$\leq$ 0.5% ✓	91.3 $\pm$ 0.3	2.1 $\pm$ 0.78	0.9 $\pm$ 0.51
90	NFT	99.7 $\pm$ 0.04	0.4 $\pm$ 0.26	0.3 $\pm$ 0.2	–	90.5 $\pm$ 0.2	2.7 $\pm$ 1.03	1.3 $\pm$ 0.65
	NFT + ES	97.4 $\pm$ 1.59	2.5 $\pm$ 1.42	1.8 $\pm$ 1.15	–	91.0 $\pm$ 0.15	1.8 $\pm$ 0.9	1.0 $\pm$ 0.65
	EL + RB	99.7 $\pm$ 0.05	0.3 $\pm$ 0.16	0.2 $\pm$ 0.15	–	90.6 $\pm$ 0.17	2.0 $\pm$ 0.52	0.8 $\pm$ 0.27
	CEAG	99.7 $\pm$ 0.03	0.2 $\pm$ 0.1	0.2 $\pm$ 0.05	$\leq$ 0.5% ✓	90.4 $\pm$ 0.37	3.0 $\pm$ 0.82	1.4 $\pm$ 0.51
92.5	NFT	99.2 $\pm$ 0.08	1.0 $\pm$ 0.17	0.7 $\pm$ 0.19	–	90.0 $\pm$ 0.44	3.1 $\pm$ 0.71	1.4 $\pm$ 0.57
	NFT + ES	96.2 $\pm$ 1.82	3.0 $\pm$ 0.98	2.2 $\pm$ 0.8	–	90.4 $\pm$ 0.37	2.8 $\pm$ 0.55	1.2 $\pm$ 0.52
	EL + RB	99.2 $\pm$ 0.07	1.3 $\pm$ 0.34	0.9 $\pm$ 0.3	–	89.8 $\pm$ 0.29	3.1 $\pm$ 1.64	1.5 $\pm$ 0.98
	CEAG	99.1 $\pm$ 0.14	0.8 $\pm$ 0.24	0.4 $\pm$ 0.09	$\leq$ 0.5% ✓	90.1 $\pm$ 0.52	2.2 $\pm$ 0.75	1.0 $\pm$ 0.2

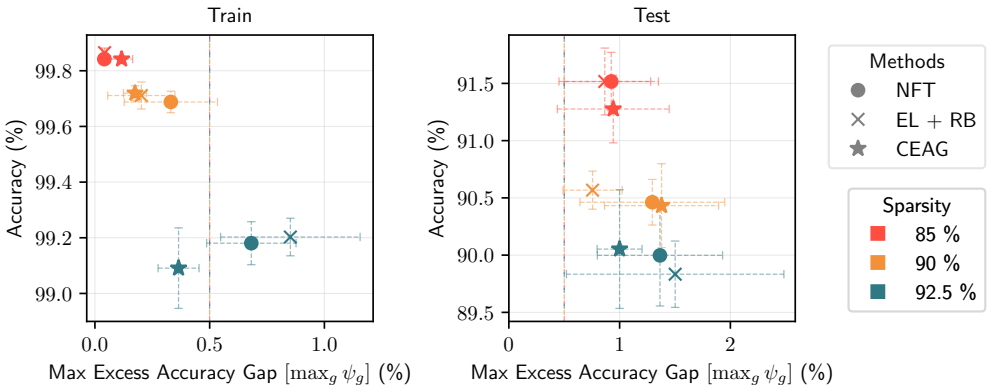


Figure C.3: UTKFace gender prediction with race as protected attribute.

C.6.1.2 *Race*

Table C.17: Groupwise train accuracy for race prediction in UTKFace with race as protected attribute, across sparsities.

Sparsity	Method	White	Black	Asian	Indian	Others
85	Dense	99.8	99.7	99.9	99.8	99.5
	NFT	99.7 ± 0.08	99.6 ± 0.11	99.8 ± 0.06	99.7 ± 0.06	99.0 ± 0.13
	EL + RB	99.6 ± 0.11	99.6 ± 0.18	99.8 ± 0.1	99.8 ± 0.14	99.5 ± 0.1
	CEAG	99.6 ± 0.13	99.5 ± 0.17	99.7 ± 0.09	99.7 ± 0.19	99.8 ± 0.09
90	Dense	99.8	99.7	99.9	99.8	99.5
	NFT	98.6 ± 0.26	99.2 ± 0.24	99.3 ± 0.04	98.9 ± 0.2	89.0 ± 0.79
	EL + RB	98.4 ± 0.19	98.8 ± 0.18	99.2 ± 0.22	98.7 ± 0.3	94.3 ± 0.61
	CEAG	95.7 ± 0.35	95.5 ± 0.35	96.0 ± 0.4	97.1 ± 0.52	98.4 ± 0.42
92.5	Dense	99.8	99.7	99.9	99.8	99.5
	NFT	96.8 ± 0.15	98.0 ± 0.23	98.2 ± 0.44	96.3 ± 0.47	69.4 ± 3.72
	EL + RB	95.9 ± 0.37	97.2 ± 0.23	97.6 ± 0.55	95.9 ± 0.56	82.5 ± 1.36
	CEAG	93.2 ± 0.28	92.9 ± 0.74	92.8 ± 0.99	93.9 ± 0.79	95.4 ± 0.35

Table C.18: Groupwise test accuracy for race prediction in UTKFace with race as protected attribute, across sparsities.

Sparsity	Method	White	Black	Asian	Indian	Others
85	Dense	90.6	87.9	88.5	80.7	29.2
	NFT	87.3 ± 0.24	84.4 ± 1.32	86.7 ± 0.82	74.9 ± 1.01	31.0 ± 1.55
	EL + RB	87.2 ± 0.48	84.1 ± 0.74	86.1 ± 1.33	75.2 ± 1.53	32.8 ± 1.47
	CEAG	86.5 ± 0.31	84.3 ± 1.31	85.7 ± 1.25	75.3 ± 1.26	32.1 ± 2.36
90	Dense	90.6	87.9	88.5	80.7	29.2
	NFT	86.5 ± 0.52	83.3 ± 1.28	84.2 ± 2.46	74.6 ± 0.44	28.8 ± 1.95
	EL + RB	86.2 ± 0.94	83.7 ± 0.91	83.6 ± 1.15	75.3 ± 1.08	31.4 ± 1.33
	CEAG	86.6 ± 0.83	84.2 ± 1.1	85.6 ± 1.12	77.7 ± 1.07	29.1 ± 2.73
92.5	Dense	90.6	87.9	88.5	80.7	29.2
	NFT	86.3 ± 0.78	83.8 ± 0.58	84.6 ± 0.8	73.8 ± 1.04	28.5 ± 2.52
	EL + RB	85.1 ± 1.0	82.8 ± 1.37	83.5 ± 1.49	73.4 ± 1.09	30.9 ± 2.49
	CEAG	85.5 ± 0.6	83.0 ± 0.65	84.6 ± 1.66	76.3 ± 0.47	31.8 ± 2.38

Table C.19: Race prediction on UTKFace with race as protected attribute, across sparsities. **CEAG almost always achieves a $\max_g \psi_g$ within the threshold**. It also has the minimum $\max_g \psi_g$ across sparsities.

Sparsity	Method	Train				Test		
		Accuracy	Ψ_{PW}	$\max_g \psi_g$	Tol (ϵ)	Accuracy	Ψ_{PW}	$\max_g \psi_g$
85	NFT	99.7 $\pm$ 0.02	0.3 $\pm$ 0.16	0.2 $\pm$ 0.13	–	80.6 $\pm$ 0.42	7.6 $\pm$ 1.8	2.7 $\pm$ 0.78
	NFT + ES	92.1 $\pm$ 4.2	35.1 $\pm$ 20.55	30.3 $\pm$ 17.82	–	81.1 $\pm$ 0.51	10.7 $\pm$ 2.01	5.2 $\pm$ 2.81
	EL + RB	99.7 $\pm$ 0.03	0.4 $\pm$ 0.09	0.2 $\pm$ 0.08	–	80.6 $\pm$ 0.44	9.2 $\pm$ 2.67	2.4 $\pm$ 1.16
	CEAG	99.6 $\pm$ 0.05	0.8 $\pm$ 0.14	0.2 $\pm$ 0.05	$\leq$ 0.25% ✓	80.3 $\pm$ 0.5	8.4 $\pm$ 2.95	2.0 $\pm$ 0.91
90	NFT	98.2 $\pm$ 0.08	9.9 $\pm$ 0.82	8.7 $\pm$ 0.82	–	79.5 $\pm$ 0.46	6.1 $\pm$ 1.6	2.2 $\pm$ 0.68
	NFT + ES	90.6 $\pm$ 4.72	45.5 $\pm$ 22.51	40.6 $\pm$ 20.13	–	81.0 $\pm$ 0.24	8.0 $\pm$ 4.73	5.3 $\pm$ 4.68
	EL + RB	98.3 $\pm$ 0.06	4.4 $\pm$ 0.58	3.6 $\pm$ 0.61	–	79.7 $\pm$ 0.37	8.0 $\pm$ 1.68	1.7 $\pm$ 0.91
	CEAG	96.1 $\pm$ 0.11	3.5 $\pm$ 0.51	0.8 $\pm$ 0.26	$\leq$ 1% ✓	80.5 $\pm$ 0.25	5.2 $\pm$ 1.86	1.5 $\pm$ 0.2
92.5	NFT	95.1 $\pm$ 0.36	28.2 $\pm$ 3.49	25.3 $\pm$ 3.35	–	79.4 $\pm$ 0.17	6.1 $\pm$ 3.0	2.5 $\pm$ 1.01
	NFT + ES	91.2 $\pm$ 4.29	43.5 $\pm$ 10.64	38.7 $\pm$ 8.87	–	80.2 $\pm$ 0.21	5.7 $\pm$ 2.79	3.5 $\pm$ 1.76
	EL + RB	95.4 $\pm$ 0.22	14.6 $\pm$ 1.31	12.5 $\pm$ 1.12	–	78.6 $\pm$ 0.33	9.0 $\pm$ 3.2	2.2 $\pm$ 0.95
	CEAG	93.4 $\pm$ 0.31	3.5 $\pm$ 0.86	1.1 $\pm$ 0.36	$\leq$ 1% ✗	79.6 $\pm$ 0.4	8.0 $\pm$ 2.59	1.3 $\pm$ 0.43

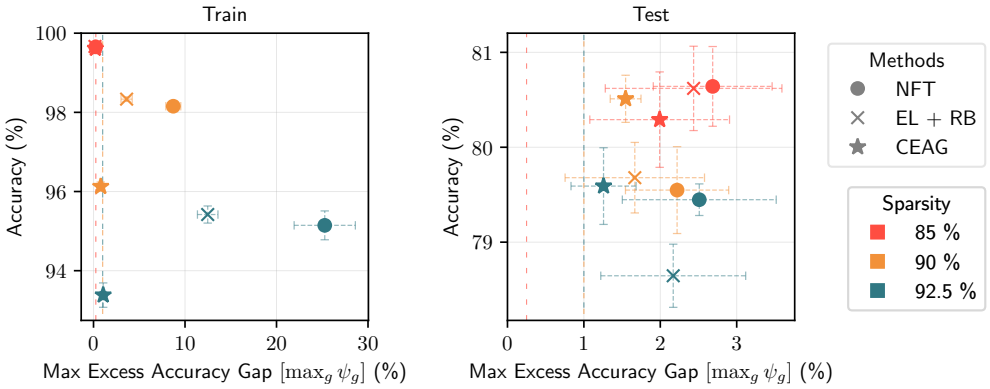


Figure C.4: UTKFace race prediction with race as protected attribute.

C.6.1.3 *Intersectional*

For the sake of brevity, we use acronyms to refer to the intersectional sub-groups. The acronyms are separated by a dash, the initial part refers to the race and the later part refers to the gender. For instance, **W-M** refers to White and Male. Other races are B-Black, A-Asian, I-Indian, and O-Others.

Table C.20: Groupwise train accuracy for race prediction in UTKFace with intersection of race and gender as protected attribute, across sparsities.

Sparsity	Method	W-M	W-F	B-M	B-F	A-M	A-F	I-M	I-F	O-M	O-F
85	Dense	99.7	99.9	99.6	99.8	99.8	99.9	99.7	99.9	99.3	99.6
	NFT	99.6 ± 0.09	99.7 ± 0.16	99.6 ± 0.18	99.7 ± 0.08	99.6 ± 0.09	99.9 ± 0.09	99.7 ± 0.1	99.8 ± 0.12	98.6 ± 0.37	99.2 ± 0.27
	EL + RB	99.5 ± 0.09	99.5 ± 0.14	99.6 ± 0.16	99.7 ± 0.07	99.7 ± 0.12	99.9 ± 0.06	99.7 ± 0.15	99.9 ± 0.15	99.6 ± 0.19	99.7 ± 0.13
	CEAG	99.6 ± 0.11	99.7 ± 0.1	99.6 ± 0.2	99.7 ± 0.09	99.7 ± 0.1	100.0 ± 0.04	99.7 ± 0.17	99.9 ± 0.07	99.4 ± 0.14	99.6 ± 0.17
90	Dense	99.7	99.9	99.6	99.8	99.8	99.9	99.7	99.9	99.3	99.6
	NFT	98.6 ± 0.2	98.4 ± 0.21	99.1 ± 0.27	99.1 ± 0.23	99.2 ± 0.24	99.5 ± 0.15	99.0 ± 0.33	98.8 ± 0.21	87.7 ± 1.07	89.9 ± 1.88
	EL + RB	98.3 ± 0.31	98.0 ± 0.29	98.7 ± 0.44	98.6 ± 0.28	98.9 ± 0.27	99.2 ± 0.15	98.3 ± 0.6	98.1 ± 0.45	97.3 ± 0.64	95.6 ± 0.59
	CEAG	96.4 ± 0.39	96.0 ± 0.33	96.1 ± 0.34	96.2 ± 0.27	95.3 ± 0.28	97.0 ± 0.78	95.7 ± 0.66	96.2 ± 0.57	96.0 ± 0.98	96.6 ± 1.29
92.5	Dense	99.7	99.9	99.6	99.8	99.8	99.9	99.7	99.9	99.3	99.6
	NFT	97.1 ± 0.61	96.7 ± 0.42	97.7 ± 0.46	98.0 ± 0.27	98.2 ± 0.33	98.8 ± 0.34	96.5 ± 0.54	96.3 ± 0.71	63.8 ± 1.49	70.9 ± 2.09
	EL + RB	95.9 ± 0.48	95.1 ± 0.43	96.8 ± 0.32	96.9 ± 0.23	97.1 ± 0.5	98.1 ± 0.58	94.8 ± 0.83	94.8 ± 0.51	88.7 ± 1.91	86.4 ± 1.64
	CEAG	94.2 ± 0.49	93.4 ± 0.16	93.6 ± 0.93	94.2 ± 0.58	91.9 ± 1.18	94.2 ± 0.55	91.6 ± 0.41	92.8 ± 1.31	92.0 ± 0.99	92.7 ± 1.89

Table C.21: Groupwise test accuracy for race prediction in UTKFace with intersection of race and gender as protected attribute, across sparsities.

Sparsity	Method	W-M	W-F	B-M	B-F	A-M	A-F	I-M	I-F	O-M	O-F
85	Dense	90.9	90.2	86.9	88.9	87.5	89.3	81.0	80.4	26.4	31.6
	NFT	87.3 ± 0.7	86.5 ± 0.79	82.6 ± 1.21	86.7 ± 1.66	84.2 ± 1.91	87.5 ± 0.6	74.3 ± 1.66	78.4 ± 2.26	29.3 ± 1.56	32.0 ± 3.4
	EL + RB	87.3 ± 0.9	86.1 ± 1.18	82.2 ± 1.73	85.6 ± 0.51	84.5 ± 2.53	86.7 ± 1.88	74.6 ± 1.57	78.1 ± 1.15	31.2 ± 2.36	32.1 ± 2.81
	CEAG	87.2 ± 0.65	85.8 ± 0.76	82.6 ± 1.27	86.0 ± 0.96	84.6 ± 0.52	87.7 ± 1.76	74.0 ± 1.18	77.2 ± 1.7	31.5 ± 1.83	32.3 ± 3.18
90	Dense	90.9	90.2	86.9	88.9	87.5	89.3	81.0	80.4	26.4	31.6
	NFT	86.8 ± 0.87	86.4 ± 0.97	81.9 ± 1.5	85.2 ± 1.1	82.2 ± 2.46	84.4 ± 1.99	74.2 ± 0.53	75.4 ± 1.09	26.8 ± 3.26	31.6 ± 1.74
	EL + RB	85.9 ± 1.17	85.6 ± 1.33	82.5 ± 0.58	83.8 ± 0.48	83.4 ± 1.95	85.0 ± 2.07	73.8 ± 1.29	76.1 ± 1.45	30.3 ± 1.11	32.3 ± 2.81
	CEAG	86.2 ± 1.35	87.2 ± 0.92	83.5 ± 1.57	85.6 ± 0.6	82.6 ± 1.52	86.5 ± 1.22	76.9 ± 1.26	76.6 ± 1.23	25.8 ± 2.63	29.8 ± 1.16
92.5	Dense	90.9	90.2	86.9	88.9	87.5	89.3	81.0	80.4	26.4	31.6
	NFT	86.5 ± 1.04	86.6 ± 0.92	81.9 ± 1.48	83.0 ± 1.28	82.3 ± 1.09	85.5 ± 1.61	72.8 ± 1.26	75.6 ± 2.2	26.8 ± 3.56	29.6 ± 1.36
	EL + RB	85.0 ± 0.83	84.9 ± 0.9	81.8 ± 0.92	82.8 ± 1.28	81.7 ± 0.97	85.6 ± 1.32	72.8 ± 0.81	74.0 ± 2.07	32.8 ± 6.05	33.9 ± 2.71
	CEAG	86.8 ± 0.64	85.3 ± 1.22	82.5 ± 1.47	84.8 ± 0.58	81.8 ± 1.98	86.2 ± 1.15	73.7 ± 1.73	76.0 ± 1.41	29.5 ± 1.4	30.7 ± 2.73

Table C.22: Race prediction task on the UTKFace dataset with the intersection of race and gender as group attribute, across sparsities. For instance, if a sample has race as *Black* and gender as *Female*, its group label is *Black-Female*. **CEAG consistently achieves a** $\max_g \psi_g$ **within the threshold, across sparsities.** This table is an extension of Table 8.2.

Sparsity	Method	Train				Test		
		Accuracy	Ψ_{PW}	$\max_g \psi_g$	Tol (ϵ)	Accuracy	Ψ_{PW}	$\max_g \psi_g$
85	NFT	99.6 $\pm$ 0.03	0.8 $\pm$ 0.26	0.5 $\pm$ 0.33	–	80.6 $\pm$ 0.44	10.3 $\pm$ 2.09	3.4 $\pm$ 1.29
	NFT + ES	91.8 $\pm$ 4.03	37.0 $\pm$ 21.45	31.8 $\pm$ 18.93	–	81.2 $\pm$ 0.4	13.0 $\pm$ 3.29	5.2 $\pm$ 2.03
	EL + RB	99.7 $\pm$ 0.02	0.8 $\pm$ 0.37	0.2 $\pm$ 0.08	–	80.4 $\pm$ 0.29	11.7 $\pm$ 3.4	3.4 $\pm$ 1.53
	CEAG	99.7 $\pm$ 0.03	0.5 $\pm$ 0.1	0.2 $\pm$ 0.04	$\leq$ 0.5% ✓	80.4 $\pm$ 0.27	12.2 $\pm$ 2.62	3.6 $\pm$ 1.14
90	NFT	98.1 $\pm$ 0.06	11.5 $\pm$ 0.72	10.0 $\pm$ 0.67	–	79.6 $\pm$ 0.49	8.9 $\pm$ 2.35	3.1 $\pm$ 0.46
	NFT + ES	90.5 $\pm$ 4.73	49.8 $\pm$ 23.02	44.8 $\pm$ 20.76	–	81.0 $\pm$ 0.24	12.0 $\pm$ 5.34	6.9 $\pm$ 4.79
	EL + RB	98.3 $\pm$ 0.19	3.2 $\pm$ 0.63	2.4 $\pm$ 0.61	–	79.4 $\pm$ 0.5	11.4 $\pm$ 0.91	3.0 $\pm$ 1.06
	CEAG	96.2 $\pm$ 0.1	2.4 $\pm$ 0.59	1.0 $\pm$ 0.27	$\leq$ 3% ✓	80.2 $\pm$ 0.13	6.0 $\pm$ 2.48	2.3 $\pm$ 1.03
92.5	NFT	95.1 $\pm$ 0.17	34.2 $\pm$ 1.64	30.7 $\pm$ 1.48	–	79.2 $\pm$ 0.16	8.8 $\pm$ 3.18	3.6 $\pm$ 1.3
	NFT + ES	91.2 $\pm$ 2.66	53.3 $\pm$ 9.55	48.0 $\pm$ 8.28	–	80.4 $\pm$ 0.35	7.5 $\pm$ 3.41	5.4 $\pm$ 3.13
	EL + RB	95.4 $\pm$ 0.27	11.1 $\pm$ 1.45	8.6 $\pm$ 1.42	–	78.7 $\pm$ 0.27	16.3 $\pm$ 3.92	3.3 $\pm$ 0.62
	CEAG	93.4 $\pm$ 0.31	3.8 $\pm$ 0.4	2.3 $\pm$ 0.41	$\leq$ 3% ✓	79.5 $\pm$ 0.14	10.8 $\pm$ 2.21	3.3 $\pm$ 1.02

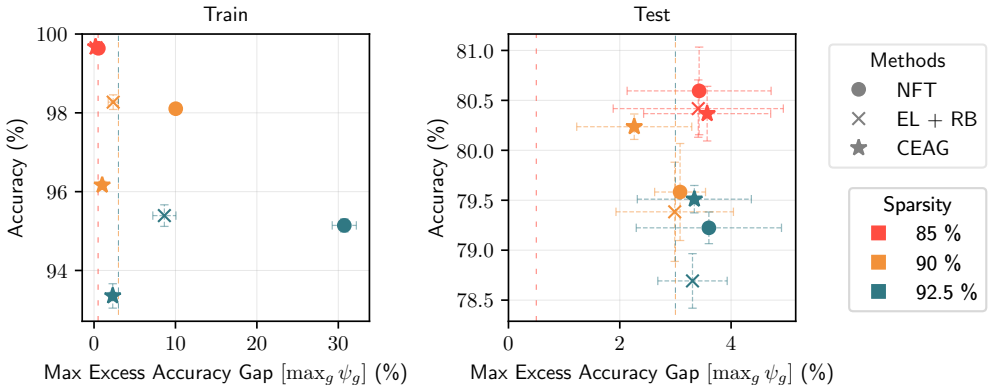


Figure C.5: UTKFace race prediction with race and gender (intersectional) as protected attributes.

C.6.2 *FairFace*

We make use of ResNet-34 models [He+16] on FairFace, similar to [LKJ22].

C.6.2.1 *Gender*

Table C.23: Groupwise train accuracy for gender prediction in FairFace with race as protected attribute.

Sparsity	Method	East Asian	Indian	Black	White	Middle Eastern	Latino Hispanic	S.E. Asian
99	Dense	97.2	97.1	94.9	97.4	98.0	97.2	96.8
	NFT	99.4 ± 0.21	99.4 ± 0.23	98.9 ± 0.12	99.3 ± 0.14	99.5 ± 0.15	99.3 ± 0.2	99.2 ± 0.31
	EL + RB	99.2 ± 0.21	99.4 ± 0.11	99.0 ± 0.24	99.4 ± 0.03	99.6 ± 0.06	99.4 ± 0.09	99.2 ± 0.13
	CEAG	99.3 ± 0.16	99.2 ± 0.21	98.9 ± 0.25	99.3 ± 0.12	99.5 ± 0.13	99.3 ± 0.15	99.1 ± 0.11

Table C.24: Groupwise test accuracy for gender prediction in FairFace with race as protected attribute.

Sparsity	Method	East Asian	Indian	Black	White	Middle Eastern	Latino Hispanic	S.E. Asian
99	Dense	95.2	95.6	90.0	94.2	96.6	95.2	94.4
	NFT	92.1 ± 0.54	93.4 ± 0.53	86.5 ± 0.78	91.6 ± 0.48	95.1 ± 0.56	93.4 ± 0.59	91.4 ± 0.59
	EL + RB	92.4 ± 0.41	92.9 ± 1.14	86.8 ± 0.68	91.8 ± 0.2	94.9 ± 0.39	93.6 ± 0.36	91.4 ± 0.33
	CEAG	92.8 ± 0.49	92.7 ± 0.32	86.2 ± 0.45	91.3 ± 0.66	94.6 ± 0.25	93.8 ± 0.25	91.2 ± 0.55

Table C.25: Gender prediction on FairFace with race as protected attribute. **CEAG achieves a $\max_g \psi_g$ within the threshold**.

Sparsity	Method	Train				Test		
		Accuracy	Ψ_{PW}	$\max_g \psi_g$	Tol (ϵ)	Accuracy	Ψ_{PW}	$\max_g \psi_g$
99	NFT	99.3 ± 0.17	2.6 ± 0.23	0.9 ± 0.13	–	91.8 ± 0.17	2.4 ± 0.81	1.1 ± 0.42
	NFT + ES	97.6 ± 1.29	1.5 ± 0.67	0.5 ± 0.19	–	92.2 ± 0.13	2.8 ± 0.45	1.5 ± 0.35
	EL + RB	99.3 ± 0.09	2.7 ± 0.22	0.8 ± 0.07	–	91.9 ± 0.21	2.1 ± 0.39	1.0 ± 0.34
	FairGrape	–	–	–	–	90.5	2.3	1.0
	CEAG	99.2 ± 0.14	2.7 ± 0.22	0.9 ± 0.07	≤ 1% ✓	91.7 ± 0.1	2.4 ± 0.48	1.2 ± 0.44

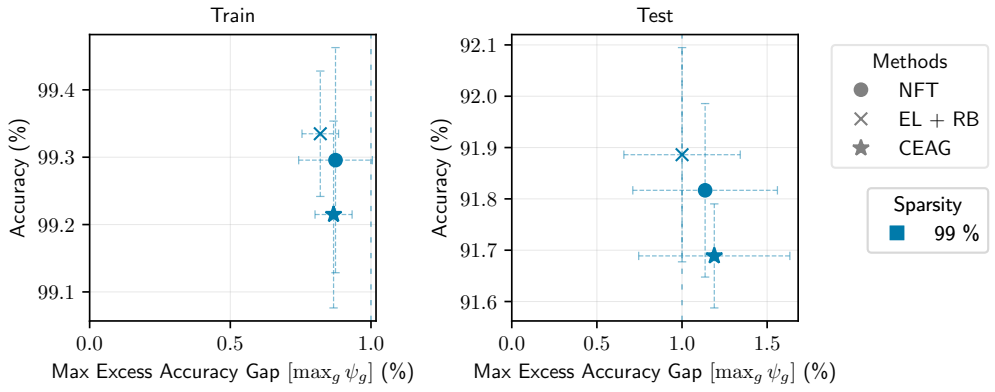


Figure C.6: FairFace gender prediction with race as protected attribute.

C.6.2.2 *Race*

Table C.26: Groupwise train accuracy for race prediction in FairFace with race as protected attribute.

Sparsity	Method	East Asian	Indian	Black	White	Middle Eastern	Latino Hispanic	S.E. Asian
99	Dense	84.8	81.9	90.9	84.7	76.8	68.0	74.1
	NFT	81.1 ± 0.88	78.2 ± 0.59	87.3 ± 0.43	80.9 ± 1.06	70.2 ± 0.73	63.2 ± 1.47	68.6 ± 1.18
	EL + RB	78.7 ± 0.58	77.0 ± 0.95	84.7 ± 0.93	78.8 ± 1.03	71.9 ± 0.39	69.7 ± 1.13	70.1 ± 1.13
	CEAG	80.7 ± 0.95	77.9 ± 0.61	87.2 ± 0.61	80.4 ± 1.14	73.7 ± 0.79	63.1 ± 1.46	68.5 ± 1.22

Table C.27: Groupwise test accuracy for race prediction in FairFace with race as metrics attribute.

Sparsity	Method	East Asian	Indian	Black	White	Middle Eastern	Latino Hispanic	S.E. Asian
99	Dense	78.3	72.2	86.2	77.2	63.9	58.1	64.3
	NFT	72.5 ± 0.98	65.8 ± 0.97	81.4 ± 0.41	70.1 ± 1.34	55.7 ± 0.94	50.9 ± 1.19	56.1 ± 1.06
	EL + RB	70.6 ± 1.28	65.0 ± 1.24	79.3 ± 0.57	68.3 ± 0.85	56.5 ± 0.54	54.8 ± 1.07	57.7 ± 1.86
	CEAG	72.5 ± 1.25	65.8 ± 1.04	81.5 ± 0.6	69.5 ± 1.32	57.5 ± 0.84	50.3 ± 1.22	56.4 ± 0.89

Table C.28: Race prediction task on FairFace with race as group attribute. Tol (ϵ) is the tolerance hyper-parameter of CEAG. We do not specify ϵ for other formulations as they do not admit a tolerance. **CEAG achieves a $\max_g \psi_g$ within the threshold.** This table is already presented in the main paper as Table 8.1, we include this for completeness.

Sparsity	Method	Train				Test		
		Accuracy	Ψ_{PW}	$\max_g \psi_g$	Tol (ϵ)	Accuracy	Ψ_{PW}	$\max_g \psi_g$
99	NFT	76.1 ± 0.19	3.9 ± 0.91	2.3 ± 0.26	–	65.2 ± 0.44	4.2 ± 0.51	2.1 ± 0.51
	NFT + ES	74.0 ± 2.55	7.2 ± 3.35	4.0 ± 1.38	–	65.4 ± 0.35	6.3 ± 2.59	2.9 ± 1.3
	EL + RB	76.1 ± 0.13	8.8 ± 1.26	2.6 ± 0.21	–	65.1 ± 0.44	6.0 ± 1.51	2.4 ± 0.36
	FairGrape	–	–	–	–	65.1	15.9	10.7
	CEAG	76.2 ± 0.13	3.5 ± 0.61	1.8 ± 0.42	$\leq$ 2% ✓	65.2 ± 0.37	4.3 ± 0.8	2.0 ± 0.32

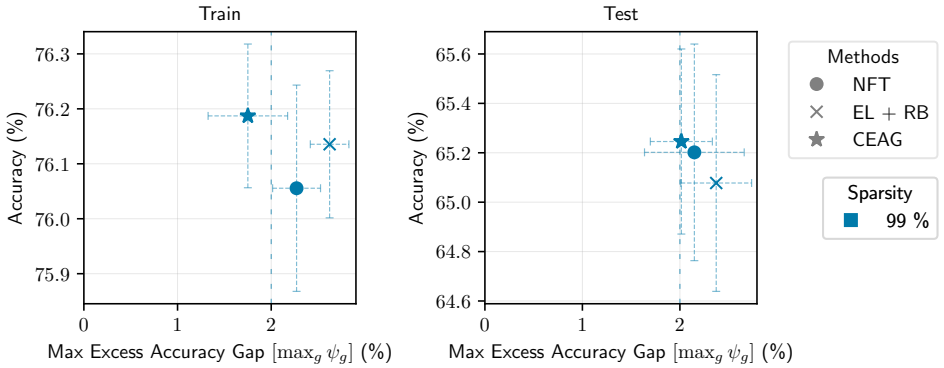


Figure C.7: FairFace race prediction with race as protected attribute.

C.6.2.3 *Intersectional*

For the sake of brevity, we use acronyms to refer to the intersectional sub-groups. The acronyms are separated by a dash, the initial part refers to the race and the later part refers to the gender. For instance, **EA-M** refers to East Asian and Male. Other races are I-Indian, B-Black, W-White, ME-Middle Eastern, LH-Latino Hispanic, and SA-South East Asian.

Table C.29: Groupwise train accuracy for race prediction in FairFace with intersection of race and gender as protected attribute.

Sparsity	Method	EA-M	EA-F	I-M	I-F	B-M	B-F	W-M	W-F	ME-M	ME-F	LH-M	LH-F	SA-M	SA-F
99	Dense	83.8	86.3	80.9	83.3	89.8	91.9	83.7	85.4	80.7	68.0	64.9	70.9	75.3	72.4
	NFT	78.6 ± 0.91	83.2 ± 0.81	76.8 ± 0.6	78.9 ± 0.48	86.5 ± 0.57	87.9 ± 0.43	79.3 ± 1.14	82.0 ± 1.08	74.0 ± 0.49	60.3 ± 1.02	59.3 ± 1.72	66.4 ± 1.31	69.8 ± 1.57	67.1 ± 1.16
	EL + RB	77.9 ± 0.77	82.5 ± 0.71	76.7 ± 0.72	78.9 ± 0.31	86.0 ± 0.58	87.3 ± 0.52	78.8 ± 1.33	81.5 ± 1.03	74.4 ± 0.56	62.6 ± 0.8	61.0 ± 2.1	67.4 ± 1.57	70.2 ± 1.17	67.3 ± 1.22
	CEAG	78.6 ± 0.72	83.1 ± 0.85	76.5 ± 0.77	79.0 ± 0.35	86.5 ± 0.58	87.9 ± 0.57	79.1 ± 1.15	81.7 ± 1.13	74.8 ± 0.77	63.1 ± 0.75	59.5 ± 1.59	66.0 ± 1.31	69.5 ± 1.48	66.8 ± 1.16

Table C.30: Groupwise test accuracy for race prediction in FairFace with intersection of race and gender as protected attribute.

Sparsity	Method	EA-M	EA-F	I-M	I-F	B-M	B-F	W-M	W-F	ME-M	ME-F	LH-M	LH-F	SA-M	SA-F
99	Dense	78.8	77.9	68.5	75.9	86.1	86.4	75.6	79.1	71.7	47.7	53.5	62.5	67.3	61.0
	NFT	70.6 ± 0.98	74.3 ± 1.63	61.4 ± 1.42	70.8 ± 0.9	83.1 ± 0.23	79.8 ± 0.53	69.3 ± 1.33	71.0 ± 1.78	62.4 ± 1.23	42.0 ± 2.72	46.1 ± 1.42	55.7 ± 1.28	57.6 ± 0.83	54.8 ± 1.0
	EL + RB	69.6 ± 0.95	73.9 ± 1.73	60.7 ± 1.62	70.4 ± 0.64	82.7 ± 1.02	79.2 ± 1.19	68.8 ± 1.3	70.3 ± 1.2	62.8 ± 0.85	44.2 ± 1.93	46.3 ± 1.69	56.1 ± 1.42	58.0 ± 0.72	55.2 ± 0.84
	CEAG	70.7 ± 1.15	74.3 ± 1.87	61.2 ± 1.48	70.9 ± 1.11	82.5 ± 0.24	80.1 ± 0.73	69.4 ± 1.22	70.5 ± 1.6	62.6 ± 1.05	43.3 ± 1.74	46.1 ± 1.18	55.1 ± 2.6	57.6 ± 0.82	55.1 ± 1.32

Table C.31: Race prediction on FairFace with intersection of gender and race as protected attribute. **CEAG achieves a** $\max_g \psi_g$ **within the threshold**.

Sparsity	Method	Train				Test		
		Accuracy	Ψ_{PW}	$\max_g \psi_g$	Tol (ϵ)	Accuracy	Ψ_{PW}	$\max_g \psi_g$
99	NFT	75.8 ± 0.17	5.2 ± 0.94	2.9 ± 0.42	–	65.3 ± 0.45	7.6 ± 0.66	3.4 ± 0.73
	NFT + ES	73.7 ± 2.33	8.9 ± 3.83	4.6 ± 2.14	–	65.4 ± 0.37	9.0 ± 2.45	4.3 ± 1.56
	EL + RB	75.8 ± 0.16	4.0 ± 1.72	2.3 ± 0.15	–	65.1 ± 0.4	7.6 ± 0.96	3.1 ± 0.36
	CEAG	75.8 ± 0.14	4.3 ± 0.42	2.3 ± 0.26	$\leq$ 2.5% ✓	65.3 ± 0.51	7.6 ± 1.23	3.5 ± 0.97

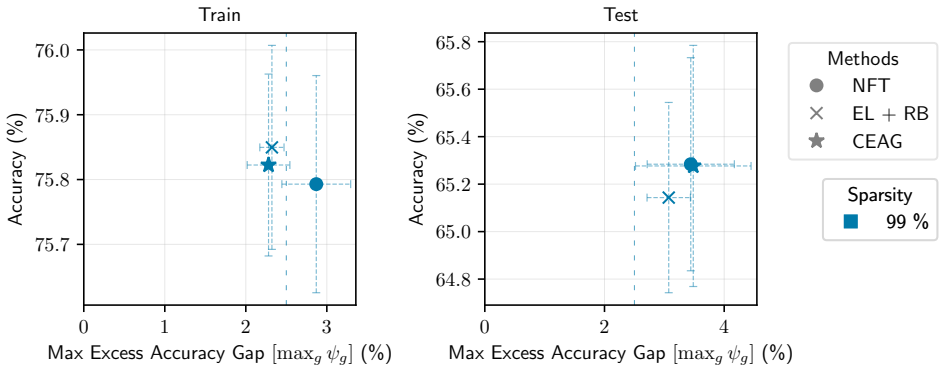


Figure C.8: FairFace race prediction with race and gender (intersection) as protected attribute.

C.6.3 *CIFAR-100*

We consider CifarResNet-56 [Che21] models for this task. We consider a scenario where both the target and group attributes correspond to the class labels. In the context of mitigating the disparate impact of pruning, we want to ensure that none of the classes degrade more than the average degradation with a tolerance of ϵ. The dense model performance is 72.61%.

Table C.32: CIFAR-100 classification with class labels as protected attribute. CEAG yields the best model in terms of disparate impact on the training set, and is competitive in terms of Ψ_{PW} and $\max_g \psi_g$ on the test set.

Sparsity	Method	Train				Test		
		Accuracy	Ψ_{PW}	$\max_g \psi_g$	Tol (ϵ)	Accuracy	Ψ_{PW}	$\max_g \psi_g$
90	NFT	99.9 ± 0.0	0.9 ± 0.18	0.4 ± 0.18	–	66.7 ± 0.35	25.0 ± 3.08	12.9 ± 2.92
	NFT + ES	99.9 ± 0.01	1.1 ± 0.3	0.6 ± 0.3	–	66.9 ± 0.27	24.6 ± 1.52	12.7 ± 1.72
	EL	99.8 ± 0.03	3.1 ± 0.61	2.4 ± 0.59	–	67.0 ± 0.32	24.4 ± 3.13	12.4 ± 1.72
	EL + RB	99.9 ± 0.01	1.3 ± 0.23	0.8 ± 0.22	–	67.0 ± 0.38	23.6 ± 1.52	12.2 ± 2.06
	CEAG (no RB)	100.0 ± 0.01	1.0 ± 0.09	0.4 ± 0.09	$\leq$ 1% ✓	66.4 ± 0.31	26.4 ± 3.21	13.8 ± 1.77
	CEAG	99.9 ± 0.01	1.0 ± 0.09	0.4 ± 0.08	$\leq$ 1% ✓	66.7 ± 0.34	23.4 ± 1.52	12.5 ± 1.4
92.5	NFT	99.8 ± 0.02	3.7 ± 0.86	3.0 ± 0.87	–	64.9 ± 0.39	26.2 ± 5.22	14.3 ± 3.39
	NFT + ES	99.3 ± 0.2	6.8 ± 1.9	5.8 ± 1.8	–	65.2 ± 0.42	27.4 ± 2.3	14.6 ± 1.97
	EL	98.5 ± 0.09	11.3 ± 0.9	9.8 ± 0.95	–	65.3 ± 0.51	25.8 ± 2.05	14.1 ± 1.29
	EL + RB	99.5 ± 0.01	6.7 ± 1.42	5.7 ± 1.48	–	65.3 ± 0.41	24.2 ± 2.86	13.3 ± 2.35
	CEAG (no RB)	99.6 ± 0.04	2.6 ± 0.3	1.7 ± 0.19	$\leq$ 2% ✓	65.0 ± 0.37	27.2 ± 2.59	14.9 ± 2.48
	CEAG	99.6 ± 0.04	2.4 ± 0.17	1.6 ± 0.15	$\leq$ 2% ✓	64.8 ± 0.3	25.0 ± 1.87	13.8 ± 1.16
95	NFT	96.2 ± 0.09	14.8 ± 1.16	11.1 ± 1.24	–	62.6 ± 0.29	28.4 ± 3.21	15.6 ± 2.57
	NFT + ES	93.9 ± 0.83	20.0 ± 1.52	14.6 ± 1.43	–	63.3 ± 0.21	30.6 ± 5.55	16.1 ± 3.22
	EL	87.9 ± 0.45	21.4 ± 1.36	14.5 ± 1.31	–	60.7 ± 0.48	32.8 ± 3.7	16.1 ± 4.57
	EL + RB	94.6 ± 0.12	18.4 ± 1.05	13.6 ± 1.07	–	62.8 ± 0.16	27.4 ± 1.34	14.8 ± 0.85
	CEAG (no RB)	95.8 ± 0.15	9.2 ± 0.52	5.8 ± 0.53	$\leq$ 5% ✗	62.5 ± 0.41	29.6 ± 4.34	17.1 ± 3.59
	CEAG	95.6 ± 0.12	9.1 ± 0.64	5.7 ± 0.49	$\leq$ 5% ✗	62.7 ± 0.28	27.6 ± 1.14	14.8 ± 1.52

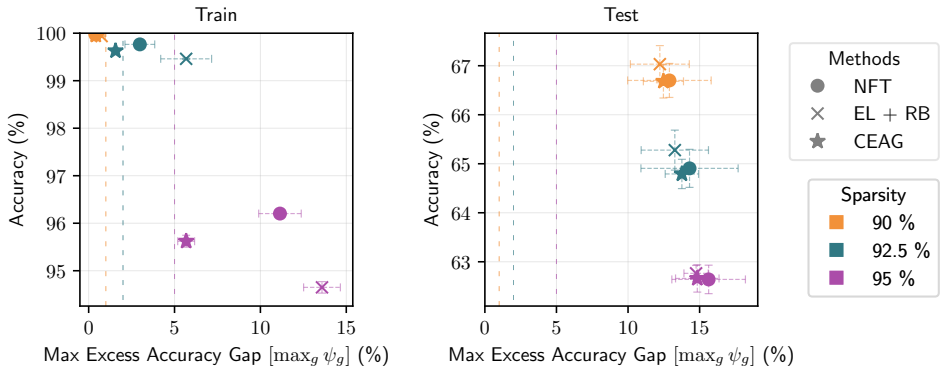


Figure C.9: CIFAR-100 classification with class labels as both target and protected attributes.

Appendix to the Fourth Contribution

D

D.1 Further discussion on prior works using PID controls in optimization

- In Stooke et al. [SAA20], 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. 10.4.1) shows that νPI provides a generalization of momentum-based optimization techniques. Note that the controller considered by Stooke et al. [SAA20] is unable to generalize momentum methods. Thanks to the unifying framework provided by Thm. 10.4.1, we provide insights to understand why momentum fails at Lagrangian optimization tasks (Fig. 10.5). Moreover, our experiments encompass SVMs, sparsity, and fairness tasks, and are not restricted to reinforcement learning.
- An et al. [An+18] 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. [Cas+23] 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 and Lessard [HL17] 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 PI algorithm and other momentum methods was inspired by this work.

D.2 CONNECTIONS BETWEEN νPI AND MOMENTUM METHODS

Table D.1: 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 D.2.1 *The* $\boldsymbol{\nu}$*PI* $(\nu, \kappa_p, \kappa_i; \boldsymbol{\xi}_0)$ *algorithm can be equivalently expressed as the recursion:*

$$\boldsymbol{\theta}_1 = \boldsymbol{\theta}_0 + \kappa_i \boldsymbol{e}_0 + \kappa_p \boldsymbol{\xi}_0 \tag{D.1a}$$
$$\boldsymbol{\xi}_t = \nu \boldsymbol{\xi}_{t-1} + (1-\nu)\boldsymbol{e}_t \tag{D.1b}$$
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \kappa_i \boldsymbol{e}_t + \kappa_p (1-\nu)\left(\boldsymbol{e}_t - \boldsymbol{\xi}_{t-1}\right) \text{ for } t \geq 1 \tag{D.1c}$$

Proof (Thm. D.2.1) *For* $t \geq 1$*, we have:*

$$\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t \overset{(\nu PI3)}{=} \kappa_p \boldsymbol{\xi}_t + \kappa_i \sum_{\tau=0}^{t} \boldsymbol{e}_\tau - \kappa_p \boldsymbol{\xi}_{t-1} - \kappa_i \sum_{\tau=0}^{t-1} \boldsymbol{e}_\tau \tag{D.2}$$
$$= \kappa_i \boldsymbol{e}_t + \kappa_p (\boldsymbol{\xi}_t - \boldsymbol{\xi}_{t-1}) \tag{D.3}$$
$$\overset{(\nu PI2)}{=} \kappa_i \boldsymbol{e}_t + \kappa_p (1-\nu)(\boldsymbol{e}_t - \boldsymbol{\xi}_{t-1}) \tag{D.4}$$

Lemma D.2.2 *The* UnifiedMomentum $(\alpha, \beta, \gamma; \boldsymbol{\phi}_0 = \mathbf{0})$ *algorithm can be expressed as the single-parameter recurrence:*

$$\boldsymbol{\theta}_1 = \boldsymbol{\theta}_0 + \alpha(1+\beta\gamma)\boldsymbol{e}_0 \tag{D.5a}$$
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \boldsymbol{e}_t + \beta(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}) + \alpha\beta\gamma\left(\boldsymbol{e}_t - \boldsymbol{e}_{t-1}\right) \text{ for } t \geq 1. \tag{D.5b}$$

Proof (Thm. D.2.2)

$$\boldsymbol{\phi}_1 \overset{(UM_2)}{=} \beta \cancel{\boldsymbol{\phi}_0} + \alpha \boldsymbol{e}_0 = \alpha \boldsymbol{e}_0 \tag{D.6a}$$
$$\boldsymbol{\theta}_1 \overset{(UM_3)}{=} \boldsymbol{\theta}_0 + \boldsymbol{\phi}_1 + \beta\gamma\left(\boldsymbol{\phi}_1 - \cancel{\boldsymbol{\phi}_0}\right) = \boldsymbol{\theta}_0 + \alpha(1+\beta\gamma)\boldsymbol{e}_0. \tag{D.6b}$$

$$\boldsymbol{\theta}_{t+1} \overset{(UM_3)}{=} \boldsymbol{\theta}_t + \boldsymbol{\phi}_{t+1} + \beta\gamma\left(\boldsymbol{\phi}_{t+1} - \boldsymbol{\phi}_t\right) \tag{D.7a}$$

$$\overset{(UM_2)}{=} \boldsymbol{\theta}_t + \beta\boldsymbol{\phi}_t + \alpha\boldsymbol{e}_t + \beta\gamma\left(\beta\boldsymbol{\phi}_t + \alpha\boldsymbol{e}_t - \beta\boldsymbol{\phi}_{t-1} - \alpha\boldsymbol{e}_{t-1}\right) \tag{D.7b}$$

$$= \boldsymbol{\theta}_t + \beta\boldsymbol{\phi}_t + \alpha\boldsymbol{e}_t + \beta\gamma\left(\beta(\boldsymbol{\phi}_t - \boldsymbol{\phi}_{t-1}) + \alpha(\boldsymbol{e}_t - \boldsymbol{e}_{t-1})\right) \tag{D.7c}$$

$$= \boldsymbol{\theta}_t + \alpha\boldsymbol{e}_t + \beta\left[\boldsymbol{\phi}_t + \gamma\beta(\boldsymbol{\phi}_t - \boldsymbol{\phi}_{t-1})\right] + \alpha\beta\gamma(\boldsymbol{e}_t - \boldsymbol{e}_{t-1}) \tag{D.7d}$$

$$\overset{(UM_3)}{=} \boldsymbol{\theta}_t + \alpha\boldsymbol{e}_t + \beta\left(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}\right) + \alpha\beta\gamma(\boldsymbol{e}_t - \boldsymbol{e}_{t-1}). \tag{D.7e}$$

Theorem D.2.3 *(Replica of* **Thm. 10.4.1***) Under the same initialization $\boldsymbol{\theta}_0$, we have that* 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:*

$$\nu = \beta \qquad \kappa_p = -\frac{\alpha\beta}{(1-\beta)^2}\left[1 - \gamma(1-\beta)\right] \qquad \kappa_i = \frac{\alpha}{1-\beta} \qquad \boldsymbol{\xi}_0 = (1-\beta)\boldsymbol{e}_0 \tag{D.8}$$

Proof (Thm. 10.4.1) *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. (D.5b). For $t \geq 2$ we have:*

$$\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t \overset{(D.5b)}{=} \alpha\boldsymbol{e}_t + \beta(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}) + \alpha\beta\gamma\left(\boldsymbol{e}_t - \boldsymbol{e}_{t-1}\right) \tag{D.9}$$

$$\kappa_i\boldsymbol{e}_t + \kappa_p\left(\boldsymbol{\xi}_t - \boldsymbol{\xi}_{t-1}\right) \overset{(D.1c)}{=} \alpha\boldsymbol{e}_t + \beta\left(\kappa_i\boldsymbol{e}_{t-1} + \kappa_p\left(\boldsymbol{\xi}_{t-1} - \boldsymbol{\xi}_{t-2}\right)\right) + \alpha\beta\gamma\left(\boldsymbol{e}_t - \boldsymbol{e}_{t-1}\right) \tag{D.10}$$

$$\boldsymbol{e}_t\left(\kappa_i - \alpha - \alpha\beta\gamma\right) + \boldsymbol{e}_{t-1}\left(-\beta\kappa_i + \alpha\beta\gamma\right) + \kappa_p\left[\boldsymbol{\xi}_t - \boldsymbol{\xi}_{t-1} - \beta\left(\boldsymbol{\xi}_{t-1} - \boldsymbol{\xi}_{t-2}\right)\right] = 0 \tag{D.11}$$

Several applications of ($\boldsymbol{\nu}$PI 2) give:

$$\boldsymbol{\xi}_t - \boldsymbol{\xi}_{t-1} - \beta\left(\boldsymbol{\xi}_{t-1} - \boldsymbol{\xi}_{t-2}\right) \tag{D.12}$$

$$= (1-\nu)\left[\boldsymbol{e}_t - \boldsymbol{\xi}_{t-1}\right] - \beta\boldsymbol{\xi}_{t-1} + \beta\boldsymbol{\xi}_{t-2} \tag{D.13}$$

$$= (1-\nu)\boldsymbol{e}_t - (1+\beta-\nu)\boldsymbol{\xi}_{t-1} + \beta\boldsymbol{\xi}_{t-2} \tag{D.14}$$

$$= (1-\nu)\boldsymbol{e}_t - (1+\beta-\nu)\left[\nu\boldsymbol{\xi}_{t-2} + (1-\nu)\boldsymbol{e}_{t-1}\right] + \beta\boldsymbol{\xi}_{t-2} \tag{D.15}$$

$$= (1-\nu)\boldsymbol{e}_t - (1-\nu)(1+\beta-\nu)\boldsymbol{e}_{t-1} + (1-\nu)(\beta-\nu)\boldsymbol{\xi}_{t-2} \tag{D.16}$$

Thus we can re-arrange to get:

$$\begin{bmatrix}\boldsymbol{e}_t & \boldsymbol{e}_{t-1} & \boldsymbol{\xi}_{t-2}\end{bmatrix}\begin{bmatrix}\kappa_i + (1-\nu)\kappa_p - \alpha(1+\beta\gamma) \\ -\beta\kappa_i - (1-\nu)(1+\beta-\nu)\kappa_p + \alpha\beta\gamma \\ (1-\nu)(\beta-\nu)\kappa_p\end{bmatrix} = 0 \tag{D.17}$$

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

$$\kappa_i + (1-\nu)\kappa_p = \alpha(1+\beta\gamma) \tag{D.18a}$$
$$\beta\kappa_i + (1-\nu)(1-\nu+\beta)\kappa_p = \alpha\beta\gamma \tag{D.18b}$$
$$(1-\nu)(\beta-\nu)\kappa_p = 0 \tag{D.18c}$$

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

$$\nu \leftarrow \beta \qquad \kappa_i \leftarrow \frac{\alpha}{1-\beta} \qquad \kappa_p \leftarrow -\frac{\alpha\beta}{(1-\beta)^2}\left[1-\gamma(1-\beta)\right] \tag{D.19}$$

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. (D.1a). Meanwhile, the first iterate of UnifiedMomentum *is given by:*

$$\boldsymbol{\theta}_1 \overset{(D.5a)}{=} \boldsymbol{\theta}_0 + \alpha(1+\beta\gamma)\boldsymbol{e}_0 \tag{D.20}$$
$$\overset{(D.19)}{=} \boldsymbol{\theta}_0 + (\kappa_i + (1-\beta)\kappa_p)\boldsymbol{e}_0 = \boldsymbol{\theta}_0 + \kappa_i\boldsymbol{e}_0 + (1-\beta)\kappa_p\boldsymbol{e}_0 \tag{D.21}$$

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.

D.3 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:

$$\theta_{t+1}^{\nu\text{PI}} = \theta_t + \kappa_i e_t + \kappa_p(1-\nu)(e_t - \xi_{t-1}) \tag{D.22}$$
$$\theta_{t+1}^{\text{GA}} = \theta_t + \kappa_i e_t \tag{D.23}$$

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:

$$\frac{\Delta\boldsymbol{\nu}\text{PI}}{\Delta\text{GA}} = \frac{\theta_{t+1}^{\nu\text{PI}} - \theta_t}{\theta_{t+1}^{\text{GA}} - \theta_t} = \frac{\kappa_i e_t + \kappa_p(1-\nu)(e_t - \xi_{t-1})}{\kappa_i e_t} \tag{D.24}$$
$$= 1 + \frac{\kappa_p(1-\nu)}{\kappa_i} - \frac{\kappa_p(1-\nu)}{\kappa_i}\frac{\xi_{t-1}}{e_t} \tag{D.25}$$
$$= \frac{1}{1-\psi}\left[1 - \frac{\psi\xi_{t-1}}{e_t}\right]. \tag{D.26}$$

PI (Fig. D.1, middle). We consider $\boldsymbol{\nu}$PI ($\nu = 0, \kappa_p = 1, \kappa_i = 1$) in, 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: $\frac{\Delta\boldsymbol{\nu}\text{PI}}{\Delta\text{GA}} \leq 0$. 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. D.1, left). We consider Polyak ($\beta = -0.4$) as a realization of $\boldsymbol{\nu}$PI, following Thm. 10.4.1. We observe similar behavior to that of $\boldsymbol{\nu}$PI ($\nu = 0, \kappa_p = 1, \kappa_i = 1$), in the middle figure of Fig. D.1. 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. 3.

Positive momentum (Fig. D.1, right). The right plot of Fig. D.1 considers Polyak ($\beta = 0.3$) as a realization of $\boldsymbol{\nu}$PI, following Thm. 10.4.1. 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 what GA would! 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 we 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.

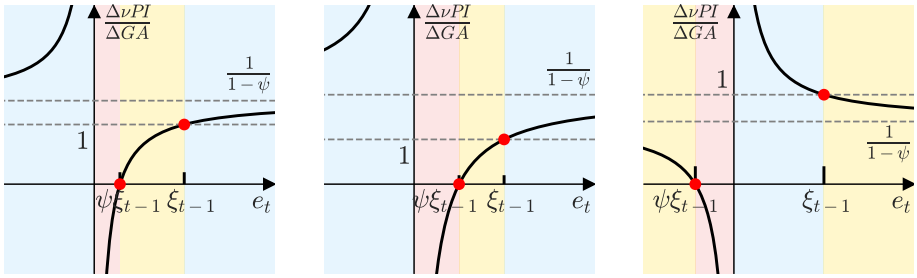


Figure D.1: Comparing the update of $\boldsymbol{\nu}$PI relative to GA, for different hyper-parameter configurations of $\boldsymbol{\nu}$PI. **Left:** $\boldsymbol{\nu}$PI is configured to recover POLYAK ($\beta = -0.4$). Updates exhibit dampening similar to that of $\boldsymbol{\nu}$PI ($\nu = 0, \kappa_p = 1, \kappa_i = 1$). **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 ($\beta = 0.4$). 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 $\boldsymbol{\nu}$PI algorithm are faster, slower, or in the opposite direction than those of GA, respectively. This plot illustrates the case $\xi_{t-1} > 0$. The middle and right figures presented here are the same as those in Fig. 10.5. We include them here for the reader's convenience.

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.

Ablation on the influence of κ_p. Appx. D.3 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 1. As κ_p increases in the middle and right plots of Appx. D.3, 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:icml2024: 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. 10.9.

D.4 ANALYSIS OF CONTINUOUS-TIME νPI DYNAMICS AS OSCILLATOR

In this section, we examine the spectral properties of the gradient-descent/$\boldsymbol{\nu}$PI flow dynamics presented in Algo. 7. We extend the analysis of STOOKE et al. [SAA20] (which only considers the dynamics of $\boldsymbol{x}$) to also consider $\boldsymbol{\lambda}$.

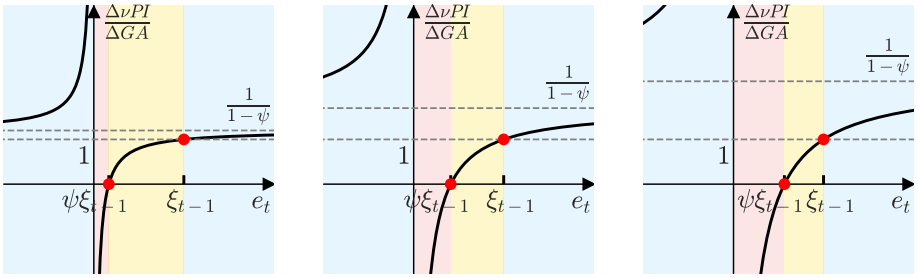


Figure D.2: Effect of κ_p in the update of $\boldsymbol{\nu}$PI relative to GA. When κ_p approaches 0, $\boldsymbol{\nu}$PI recovers GA (a constant function 1 for the relative update). **A larger κ_p leads to a wider "optimistic" region (in red) where $\boldsymbol{\nu}$PI decreases the multiplier to 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.

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. 7.

Algorithm 7 Continuous-time gradient descent/$\boldsymbol{\nu}$PI

Args: proportional (κ_p) and integral (κ_i) gains for $\boldsymbol{\nu}$PI flow

1: $\dot{\boldsymbol{x}} = -\nabla f(\boldsymbol{x}) - \mathcal{J}\boldsymbol{h}(\boldsymbol{x})\boldsymbol{\mu}$

2: $\dot{\boldsymbol{\mu}} = \kappa_i \boldsymbol{h}(\boldsymbol{x}) + \kappa_p \dot{\boldsymbol{h}}(\boldsymbol{x})$

Thm. D.4.1 characterizes the GD/νPI flow in Algo. 7 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.4.2 analyzes the resulting dynamical system for a quadratic program with linear equality constraints.

D.4.1 *Oscillator dynamics of GD/νPI flow*

Theorem D.4.1 *The dynamics of Algo. 7 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 $\dot{\boldsymbol{\mu}}(0) = \kappa_i \boldsymbol{h}(\boldsymbol{x}_0) + \mathcal{J}\boldsymbol{h}(\boldsymbol{x}_0)\dot{\boldsymbol{x}}(0)$:*

$$\begin{cases} \ddot{\boldsymbol{x}} = -\underbrace{\left(\nabla^2 f + \sum_{c'} \mu_{c'} \nabla^2 \boldsymbol{h}_{c'}\right)}_{\boldsymbol{\Phi}} \dot{\boldsymbol{x}} - \mathcal{J}\boldsymbol{h}\dot{\boldsymbol{\mu}} & \text{(D.27a)} \\ \ddot{\boldsymbol{\mu}} = \kappa_i \mathcal{J}\boldsymbol{h}^\top \dot{\boldsymbol{x}} + \kappa_p \mathcal{J}\boldsymbol{h}^\top \ddot{\boldsymbol{x}} + \kappa_p \boldsymbol{\Xi} & \text{(D.27b)} \end{cases}$$

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:

$$\begin{bmatrix} \mathbf{I}_{n\times n} & \mathbf{0}_{n\times c} \\ -\kappa_p \mathcal{J}\boldsymbol{h}^\top & \mathbf{I}_{c\times c} \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{x}} \\ \ddot{\boldsymbol{\mu}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{\Phi} & \mathcal{J}\boldsymbol{h} \\ -\kappa_i \mathcal{J}\boldsymbol{h}^\top & \mathbf{0}_{c\times c} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{x}} \\ \dot{\boldsymbol{\mu}} \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ -\beta \boldsymbol{\Xi} \end{bmatrix} = \mathbf{0}. \tag{D.28}$$

Or, equivalently:

$$\begin{bmatrix} \ddot{\boldsymbol{x}} \\ \ddot{\boldsymbol{\mu}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{\Phi} & \mathcal{J}\boldsymbol{h} \\ \mathcal{J}\boldsymbol{h}^\top (\kappa_p \boldsymbol{\Phi} - \kappa_i \mathbf{I}) & \kappa_p \mathcal{J}\boldsymbol{h}^\top \mathcal{J}\boldsymbol{h} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{x}} \\ \dot{\boldsymbol{\mu}} \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ -\beta \boldsymbol{\Xi} \end{bmatrix} = \mathbf{0}. \tag{D.29}$$

Proof (Thm. D.4.1) *We start by computing the time derivatives of the objective gradient and constraint Jacobian:*

$$\frac{d}{dt}\left[\nabla f\right] = \left[\sum_j \frac{\partial(\nabla f)}{\partial x_j}\frac{dx_j}{dt}\right]_i = \nabla^2 f \dot{\boldsymbol{x}} \tag{D.30}$$

$$\frac{d}{dt}\left[\mathcal{J}\boldsymbol{h}\right] = \begin{bmatrix} \nabla^2 \boldsymbol{h}_1 \dot{\boldsymbol{x}} & \nabla^2 \boldsymbol{h}_2 \dot{\boldsymbol{x}} & \dots & \nabla^2 \boldsymbol{h}_C \dot{\boldsymbol{x}} \end{bmatrix} \tag{D.31}$$

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

$$\ddot{\boldsymbol{x}} = \frac{d}{dt}\left[-\nabla f(\boldsymbol{x}) - \mathcal{J}\boldsymbol{h}(\boldsymbol{x})\boldsymbol{\mu}\right] = -\frac{d}{dt}\left[\nabla f\right] - \mathcal{J}\boldsymbol{h}\dot{\boldsymbol{\mu}} - \frac{d}{dt}\left[\mathcal{J}\boldsymbol{h}\right]\boldsymbol{\mu} \tag{D.32a}$$

$$= -\nabla^2 f \dot{\boldsymbol{x}} - \mathcal{J}\boldsymbol{h}\dot{\boldsymbol{\mu}} - \sum_{c'} \mu_{c'} \nabla^2 \boldsymbol{h}_{c'} \dot{\boldsymbol{x}} \tag{D.32b}$$

$$= -\underbrace{\left(\nabla^2 f + \sum_{c'} \mu_{c'} \nabla^2 \boldsymbol{h}_{c'}\right)}_{\boldsymbol{\Phi}} \dot{\boldsymbol{x}} - \mathcal{J}\boldsymbol{h}\dot{\boldsymbol{\mu}} \tag{D.32c}$$

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

$$\ddot{\boldsymbol{\mu}} = \frac{d}{dt}\left[\kappa_i \boldsymbol{h} + \kappa_p \mathcal{J}\boldsymbol{h}^\top \dot{\boldsymbol{x}}\right] = \alpha \dot{\boldsymbol{h}} + \kappa_p \boldsymbol{\Xi}\dot{\boldsymbol{x}} + \kappa_p \mathcal{J}\boldsymbol{h}^\top \ddot{\boldsymbol{x}}, \tag{D.33}$$

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

$$\boldsymbol{\Xi} \triangleq \frac{d}{dt}\left[\mathcal{J}\boldsymbol{h}^\top\right]\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\boldsymbol{x}}^\top \nabla^2 \boldsymbol{h}_1 \dot{\boldsymbol{x}} & \dot{\boldsymbol{x}}^\top \nabla^2 \boldsymbol{h}_2 \dot{\boldsymbol{x}} & \dots & \dot{\boldsymbol{x}} \nabla^2 \boldsymbol{h}_c \dot{\boldsymbol{x}} \end{bmatrix}^\top. \tag{D.34}$$

D.4.2 *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:

$$\min_{\boldsymbol{x}} \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{H}\boldsymbol{x} + \boldsymbol{c}^\top \boldsymbol{x} \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} = 0. \tag{D.35}$$

The Lagrangian min-max game associated with the problem in Eq. (D.35) is given by:

$$\mathfrak{L}(\boldsymbol{x}, \boldsymbol{\mu}) = \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{H}\boldsymbol{x} + \boldsymbol{c}^\top \boldsymbol{x} + \boldsymbol{\mu}^\top (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) = \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{H}\boldsymbol{x} + \boldsymbol{c}^\top \boldsymbol{x} + \boldsymbol{\mu}^\top \boldsymbol{A}\boldsymbol{x} - \boldsymbol{\mu}^\top \boldsymbol{b}. \quad \text{(D.36)}$$

The linearity of the constraints in Eq. (D.35) implies $\mathcal{J}\boldsymbol{h} = \boldsymbol{A}^\top$ and $\nabla^2 g_{c'} = \mathbf{0}$ for $c' \in [c]$, thus $\boldsymbol{\Phi} = \boldsymbol{H}$ and $\boldsymbol{\Xi} = \mathbf{0}$. Therefore, we obtain a homogeneous system of second-order differential equations with constant coefficients:

$$\begin{bmatrix} \ddot{\boldsymbol{x}} \\ \ddot{\boldsymbol{\mu}} \end{bmatrix} + \underbrace{\begin{bmatrix} \boldsymbol{H} & \boldsymbol{A}^\top \\ \boldsymbol{A}\left(\kappa_p \boldsymbol{H} - \kappa_i \mathbf{I}\right) & \kappa_p \boldsymbol{A}\boldsymbol{A}^\top \end{bmatrix}}_{\boldsymbol{U}} \begin{bmatrix} \dot{\boldsymbol{x}} \\ \dot{\boldsymbol{\mu}} \end{bmatrix} = \mathbf{0}. \quad \text{(D.37)}$$

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:

$$\dot{\boldsymbol{z}} = -\begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \boldsymbol{U} \end{bmatrix} \boldsymbol{z} = -\begin{bmatrix} \mathbf{0}_{(n+c)\times(n+c)} & \mathbf{I}_{(n+c)\times(n+c)} \\ \mathbf{0}_{(n+c)\times(n+c)} & \begin{bmatrix} \boldsymbol{H} & \boldsymbol{A}^\top \\ \boldsymbol{A}\left(\kappa_p \boldsymbol{H} - \kappa_i \mathbf{I}\right) & \kappa_p \boldsymbol{A}\boldsymbol{A}^\top \end{bmatrix} \end{bmatrix} \boldsymbol{z} \quad \text{(D.38)}$$

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. [Gid+19a, Eq. 18] in their study of negative momentum. In this case, the matrix $\boldsymbol{U}$ looks like:

$$-\boldsymbol{U} = -\begin{bmatrix} \mathbf{0} & \boldsymbol{A}^\top \\ -\kappa_i \boldsymbol{A} & \kappa_p \boldsymbol{A}\boldsymbol{A}^\top \end{bmatrix} \quad \text{(D.39)}$$

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 [Gid+19a].

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:

$$-\boldsymbol{U} = -\begin{bmatrix} h & a \\ a\left(\kappa_p h - \kappa_i\right) & \kappa_p a^2 \end{bmatrix} \quad \text{(D.40)}$$

$$\lambda = \frac{-\left(h + \kappa_p a^2\right) \pm \sqrt{\left(h + \kappa_p a^2\right)^2 - 4a^2\kappa_i}}{2} \quad \text{(D.41)}$$

As before, the eigenvalues of this matrix depend on the choice of κ_p. This is illustrated in Fig. D.3.

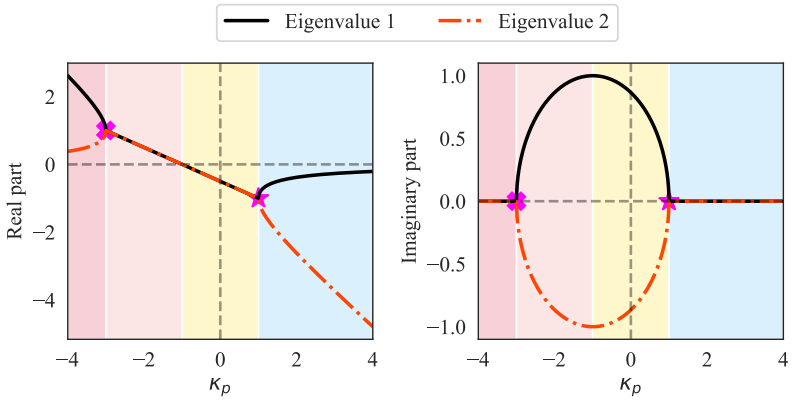


Figure D.3: Eigenvalues for Eq. (D.35) as a function of κ_p in the one-dimensional case. **A positive value of κ_p (denoted by $\star$) achieves *critical damping* (i.e. equal convergence rate for both dimensions).** This plot uses $h = 1$, $a = -1$ and $\kappa_i = 1$.

Note that when the discriminant of Eq. (D.41) 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. D.3. 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.

D.5 Illustrative 2D nonconvex problem

We demonstrate the behavior of $\boldsymbol{\nu}$PI on the two-dimensional, nonconvex, equality-constrained problem in Eq. (D.42). This problem was proposed by Boyd [Boy21]. 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.

$$\min_{\boldsymbol{x}=(x_1,x_2)} f(\boldsymbol{x}) \triangleq \left\| \begin{bmatrix} x_1 + e^{-x_2} \\ x_1^2 + 2x_2 + 1 \end{bmatrix} \right\|_2^2 \quad \text{s.t. } h(\boldsymbol{x}) \triangleq x_1 + x_1^3 + x_2 + x_2^2 = 2 \quad \text{(D.42)}$$

GA trajectories. In Fig. D.4, 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.

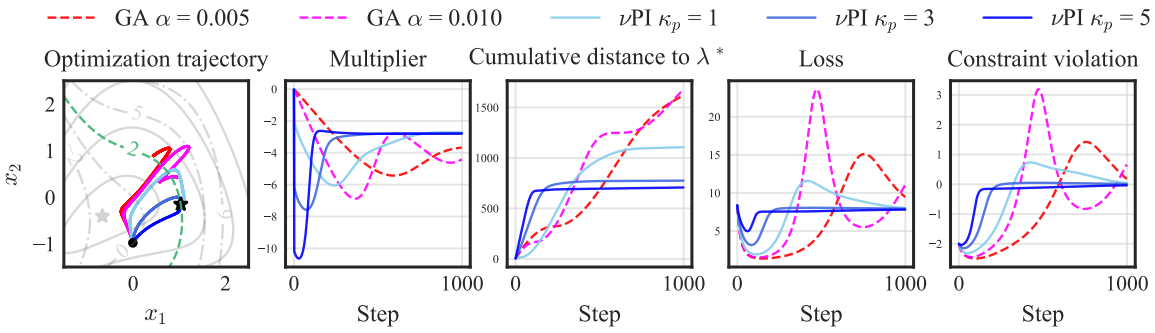


Figure D.4: Optimization trajectories for different algorithms on a 2D nonconvex equality-constrained minimization problem. $\boldsymbol{\nu}$PI runs use $\nu = 0$ and $\kappa_i = 0.01$. The light gray $\star$ marks the *unconstrained* optimum, while the black $\star$ marks the *constrained* optimum. Level sets correspond to the objective function (solid) and constraint (dashed).

$\boldsymbol{\nu}$PI trajectories. The three blue trajectories in Fig. D.4 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.

Ablation on ν. In Fig. D.5, 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

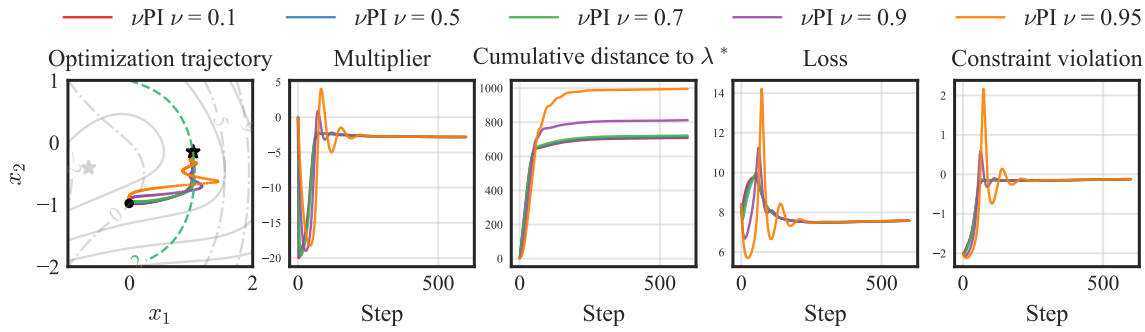


Figure D.5: Optimization trajectories for the νPI algorithm under different choices of ν. νPI runs use $\kappa_i = 0.01$ and $\kappa_p = 10$.

$\nu = 0.95$. 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.

D.6 EXPERIMENTAL DETAILS

Our implementation use PyTorch 2.0.0 [Pas+19] and the Cooper library for Lagrangian constrained optimization [Gal+24].

D.6.1 *SVM experiments*

In our experiment with the SVM task, we focus on two linearly separable classes from the Iris dataset [Fis88]. We select 100 instances from the Iris setosa and Iris versicolor species, which are two linearly separable classes in this dataset. 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^*. 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 step size 10^{-3} and momentum 0.9.

Different values of the parameter ν in νPI algorithm. We examine how changing the parameter ν in the Algo. 3 can affect the convergence of the SVM task with different choices of κ_i and κ_p. Fig. D.6 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. 10.4.1 indicates that the POLYAK and NESTEROV momentum algorithms can be instantiated using a specific

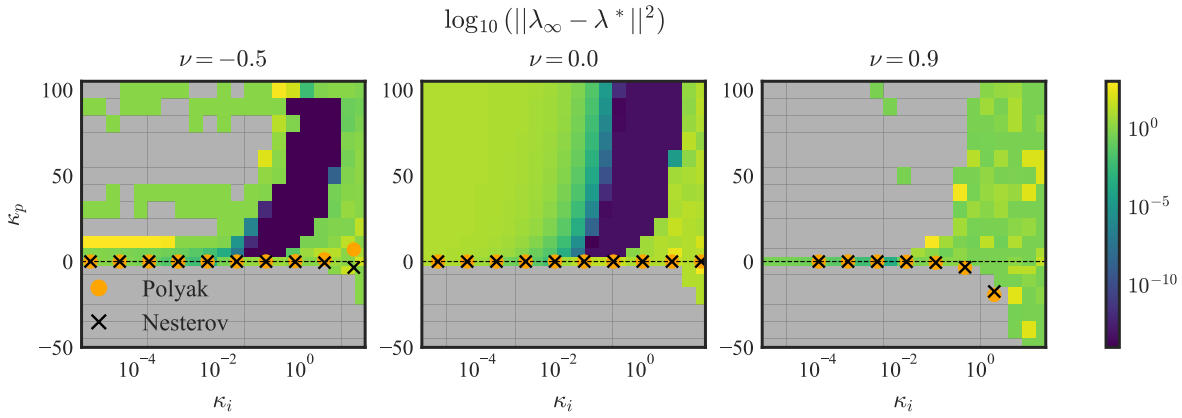


Figure D.6: Distance to optimal Lagrange multipliers for different selections of parameter ν in the νPI algorithm in the hard-margin SVM task. We also show where the equivalent κ_p and κ_i parameters for Nesterov $(\alpha, \beta = \nu)$ and Polyak $(\alpha, \beta = \nu)$ lie according to Thm. 10.4.1 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^*.

choice of κ_i and κ_p in the Algo. 3. In Fig. D.6 we show where Polyak (α, β) and Nesterov (α, β) lie for a choice of αs and with $\beta = \nu$. For each pair of (α_i, β) we calculate the value of κ_i and κ_p according to Thm. 10.4.1.

- When $\beta = \nu = 0$ there is no momentum. Thus all of the dots indicating momentum methods lie in the $\kappa_p = 0$ line which corresponds to gradient ascent.
- A positive $\beta = \nu = 0.9$ corresponds to the default choice of momentum in single-objective minimization. We see that there is no step-size value for which Polyak or Nesterov converge in this task. This observation supports the claims made by Gidel et al. [Gid+19a] on the ineffectiveness of positive momentum in games.
- With a negative value $\beta = \nu = -0.5$, there exist step-size choices leading to convergence. However, these regions do not overlap with the locations realizable via Polyak and Nesterov momentum.

  Another observation from this plot is that, with negative momentum, Polyak results in a positive κ_p while Nesterov cannot. This further supports the experimental results of [Gid+19a], where Polyak method 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.

D.6.2 *Sparsity experiments*

Background. Louizos et al. [LWK18] propose a re-parameterization of models that allows applying L_0-norm regularization on their weights. They propose the use of stochastic gates z that indicate whether each parameter is active or not, where 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. [Gal+22] formulate a constrained optimization problem that prescribes the desired sparsity of the model as a constraint.

$$\min_{\boldsymbol{w},\boldsymbol{\phi}\in\mathbb{R}^d} \mathbb{E}_{\boldsymbol{z}|\boldsymbol{\phi}}\left[L(\boldsymbol{w}\odot\boldsymbol{z}\,|\,\mathcal{D})\right] \quad \text{s.t.} \quad \frac{\mathbb{E}_{\boldsymbol{z}|\boldsymbol{\phi}}[\|\boldsymbol{z}\|_0]}{\#(\boldsymbol{w})} \leq \epsilon, \tag{D.43}$$

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. [LWK18] and GALLEGO-POSADA et al. [Gal+22].

Hard-concrete distribution. The L_0-norm re-parameterization proposed by LOUIZOS et al. [LWK18] 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 [MMT17]. Similar to LOUIZOS et al. [LWK18] and GALLEGO-POSADA et al. [Gal+22], 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+16] models with basic residual blocks for our sparsity experiments, which have a total of approximately 11.2 million parameters. Following LOUIZOS et al. [LWK18] and GALLEGO-POSADA et al. [Gal+22], 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. [Gal+22] consider up to 80% structured sparsity (20% density) for ResNet-18 models, GALE et al. [GEH19] 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 [LH17].

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. [Gal+22], we use Adam [KB15] 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. [Gal+22].

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. [Gal+22]). 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.

D.6.3 *Fairness experiments*

Dataset. In this experiment we consider the adult dataset [BK96], pre-processed following Zafar et al. [Zaf+19]. 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 [Zaf+19] shown in Eq. (10.13). This constraint is also known as statistical parity and demographic parity [**10.1145/3097983.3098095**; **10.1145/2090236.2090255**]. We consider two sensitive attributes in the adult dataset: sex, denoted as $A_1 = \{\textit{male}, \textit{female}\}$, and race, denoted as $A_2 = \{\textit{White}, \textit{Black}, \textit{Asian-Pac-Islander}, \textit{Amer-Indian-Eskimo}, \textit{Other}\}$. Eq. (10.13) 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. [Cot+19b]. 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 ($\alpha = 1e-2$).

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. 10.7.

D.7 Comprehensive results on the sparsity task

In this section we provide extensive experiment results for our sparsity experiments, complementing §10.5.3. 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.

D.7.1 *Global*

For global sparsity experiments (Figs. D.7 to D.10 and Tables D.2 to D.5), we observe a general trend for models that overshoot into becoming more sparse to achieve lower training performance. This insight is also generally true for validation accuracy, but not necessarily so. In particular, gradient ascent and momentum methods consistently exhibit overshoot, whereas νPI and gradient ascent with dual restarts do not overshoot and achieve good performance. Dual restarts generally produce (slightly) infeasible solutions. Note that at $\epsilon = 10\%$, negative momentum runs do not overshoot, but positive momentum runs do.

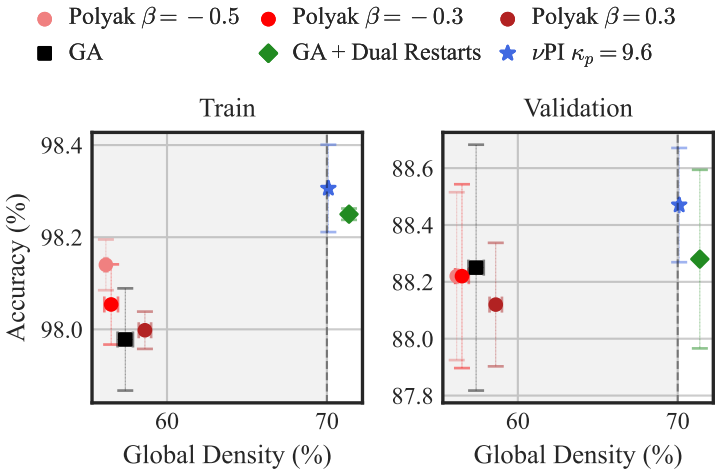


Figure D.7: CIFAR10 trade-off plot for *global* sparsity under a 70% density target. **νPI successfully achieves the desired sparsity while achieving the highest train accuracy.** The shaded region is the feasible set. As higher density correlates to higher train accuracy, overshooting to a lower density is undesirable. All optimizers use the same step-size. *This figure is the same as Fig. 10.8. We repeat it here for the reader's convenience.*

Table D.2: 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. D.7. 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

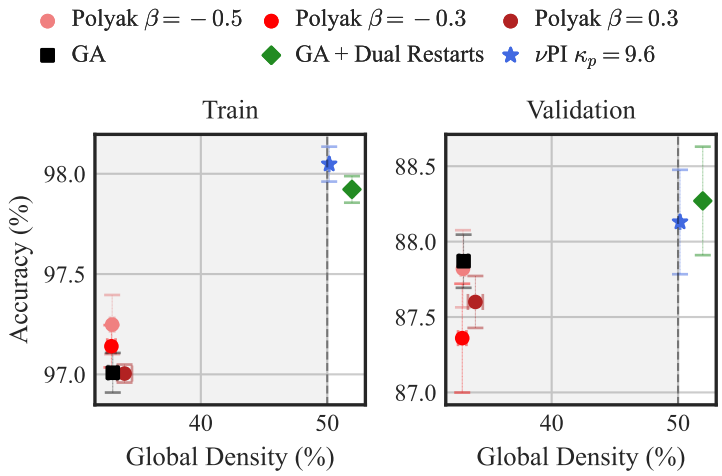


Figure D.8: CIFAR10 trade-off plot for *global* sparsity under a 50% density target.

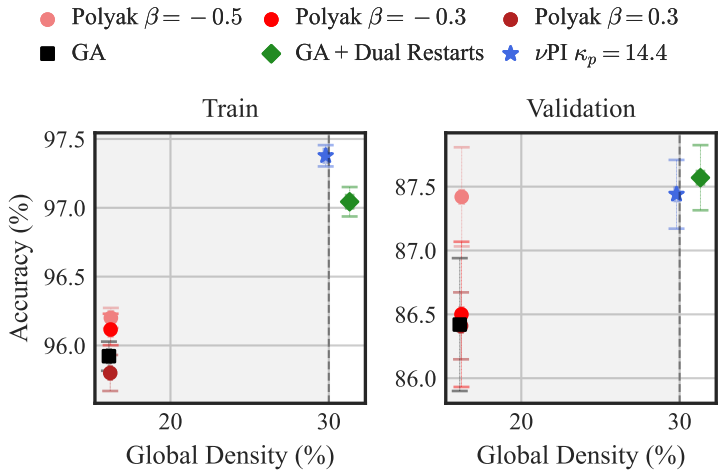


Figure D.9: CIFAR10 trade-off plot for *global* sparsity under a 30% density target.

Table D.3: CIFAR10 results for *global* sparsity under a 50% density target. The results in this table correspond to those in Fig. D.8.

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

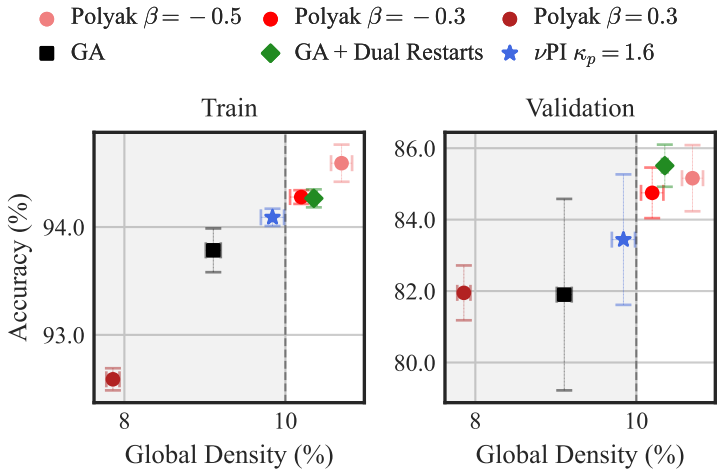


Figure D.10: CIFAR10 trade-off plot for *global* sparsity under a 10% density target.

Table D.4: CIFAR10 results for *global* sparsity under a 30% density target. The results in this table correspond to those in Fig. D.9.

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 D.5: CIFAR10 results for *global* sparsity under a 10% density target. The results in this table correspond to those in Fig. D.10.

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

D.7.2 *Layer-wise*

We perform layer-wise sparsity experiments with $\epsilon = 10\%, 30\%, 50\%, 70\%$ density targets (Table D.6). We observe a similar trend to that of global sparsity experiments: GA and momentum methods overshoot, while νPI and dual restarts reliably achieve feasible solutions, with small levels of overshoot. Moreover, we observe a small range in the value of the violations for νPI and dual restarts.

Table D.6: CIFAR10 results for *layer-wise* sparsity under 70-10% density targets. As higher density correlates to higher train accuracy, overshooting to a lower density is undesirable. All optimizers use the same step-size.

70% Density

This table is the same as Table 10.1, repeated for the reader's convenience.

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

50% Density

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

30% Density

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

10% Density

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

D.8 ADDITIONAL EXPERIMENTS

In this section, we include additional experimental results on the sparsity-constrained task. We analyze the dynamics of the multiplier throughout training, and conduct ablation studies on κ_p for $\boldsymbol{\nu}$PI, the momentum coefficients of POLYAK and NESTEROV, and the step-size of ADAM.

D.8.1 *Dynamics*

The dynamics shown in Fig. D.11 help understand the change of the constraint violation and multipliers throughout the optimization process, as opposed to only measuring the end-of-training values as in the previous section. We observe that GA, POLYAK, and ADAM quickly overshoot, but manage to regain some capacity as training progresses. This recovery is most notorious for ADAM, whose multiplier decreases quickly enough to allow for the sparsification to stop. Dual restarts reduce the value of the multiplier as soon as feasibility is achieved, thus preventing an incursion of the constraint into the feasible set. $\boldsymbol{\nu}$PI produces well-behaved multipliers whose constraints are reasonably damped.

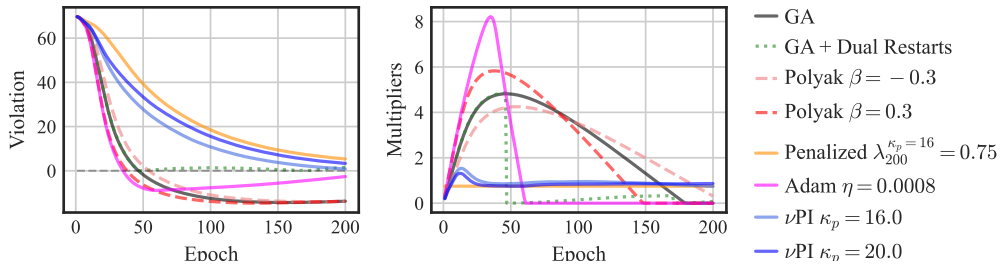


Figure D.11: Dynamics plot for global sparsity under a 30% density target.

Note that for this sparsity task, it is reasonable to expect that the constraint should be active at the constrained optimum, since more capacity in the model is likely to correlate with better performance on the training objective. However, note that $\boldsymbol{\nu}$PI is the only method that provides a non-zero estimate of the Lagrange multiplier. In the following experiment we showcase that this estimate yielded by $\boldsymbol{\nu}$PI is remarkably useful.

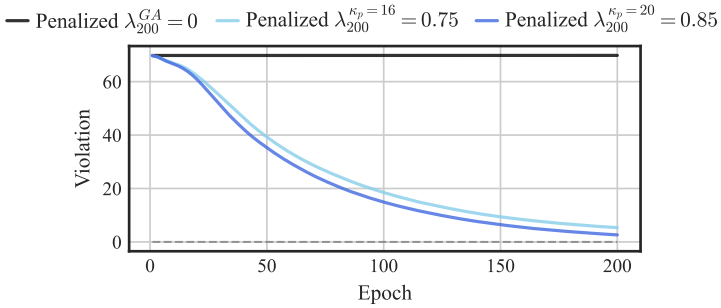


Figure D.12: Dynamics plot for global sparsity under a 30% density target.

Fig. D.12 considers an unconstrained L_0-regularization experiment. We use the final value of the νPI ($\kappa_p = 16$) and νPI ($\kappa_p = 20$) as the (fixed) penalty coefficient for 200 epochs in the penalized formulation of the problem, akin to Louizos et al. [LWK18]. We also include an experiment using the multiplier estimate (equal to zero) from GA.

Unsurprisingly, the run with the GA multiplier estimate remains at 100% density, since the penalty does not exert any influence during training. In contrast, the runs with the νPI multiplier estimates not only achieve some sparsity, but are also very close to the target density by the end of training. This is remarkable since the problem we are solving is nonconvex, and there might not even exist an optimal Lagrange multiplier value.

D.8.2 *Ablation on the value of κ_p*

In this section, we fix the dual step-size and ablate on the hyperparameter κ_p, at two sparsity levels. The results are presented in Fig. D.13 and Tables D.7 and D.8.

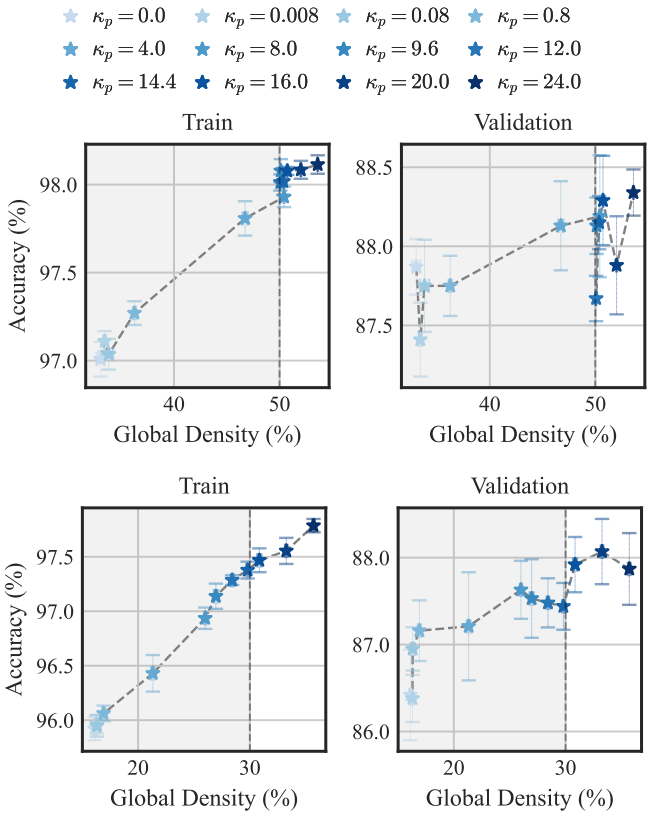


Figure D.13: Ablation on the density-accuracy trade-offs achievable by νPI under global density targets of 50% (top) and 30% (bottom).

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 D.7: 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 ± 0.10	87.9 ± 0.18	-17.0 ± 0.36	-33.9 ± 0.72
0.008	97.1 ± 0.06	87.4 ± 0.23	-16.6 ± 0.14	-33.2 ± 0.28
0.08	97.0 ± 0.09	87.7 ± 0.29	-16.2 ± 0.42	-32.4 ± 0.84
0.8	97.3 ± 0.07	87.7 ± 0.19	-13.8 ± 0.13	-27.5 ± 0.27
4	97.8 ± 0.10	88.1 ± 0.28	-3.3 ± 0.23	-6.5 ± 0.46
8	97.9 ± 0.06	88.2 ± 0.39	0.4 ± 0.03	0.9 ± 0.06
9.6	98.1 ± 0.07	88.1 ± 0.18	0.1 ± 0.02	0.2 ± 0.04
12	98.0 ± 0.05	87.7 ± 0.14	0.1 ± 0.01	0.2 ± 0.03
14.4	98.0 ± 0.08	88.2 ± 0.17	0.4 ± 0.02	0.7 ± 0.05
16	98.1 ± 0.02	88.3 ± 0.28	0.7 ± 0.02	1.5 ± 0.04
20	98.1 ± 0.05	87.9 ± 0.31	2.0 ± 0.03	4.0 ± 0.07
24	98.1 ± 0.05	88.3 ± 0.15	3.6 ± 0.03	7.2 ± 0.06

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

D.8.3 *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. D.14, and Table D.9.

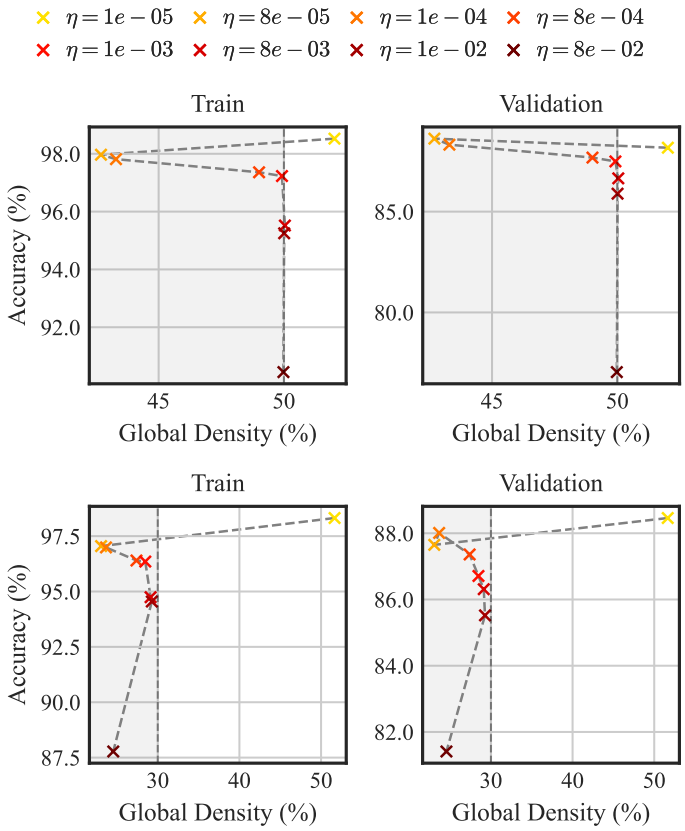


Figure D.14: Ablation on the density-accuracy trade-offs achievable by Adam under global density targets of 50% (top) and 30% (bottom).

Table D.9: Ablation on the step-size hyperparameter for Adam on a CIFAR10 task with a global density targets of $\epsilon = 50\%$ and $\epsilon = 30\%$.

50% Density			
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

30% Density			
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

We observe that the influence of Adam's learning on the constraint overshoot is not monotonic. When the step-size is too small, Adam does not satisfy the constraint. As the step-size increases, it begins to overshoot into the feasible region. There is a range of larger step-sizes that lie at the sweet spot of almost exact constraint satisfaction, right before the step-size is too large and then overshoot happens again.

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 $\beta_2 = 0.999$.

D.8.4 *Momentum*

We carried out similar ablations on the momentum coefficient of Polyak and Nesterov, using both positive and negative values. The results are shown in Fig. D.15, and Tables D.10 and D.11. We observe significant overshoot into the feasible region for all attempted values, compared to the desired target density of 30%.

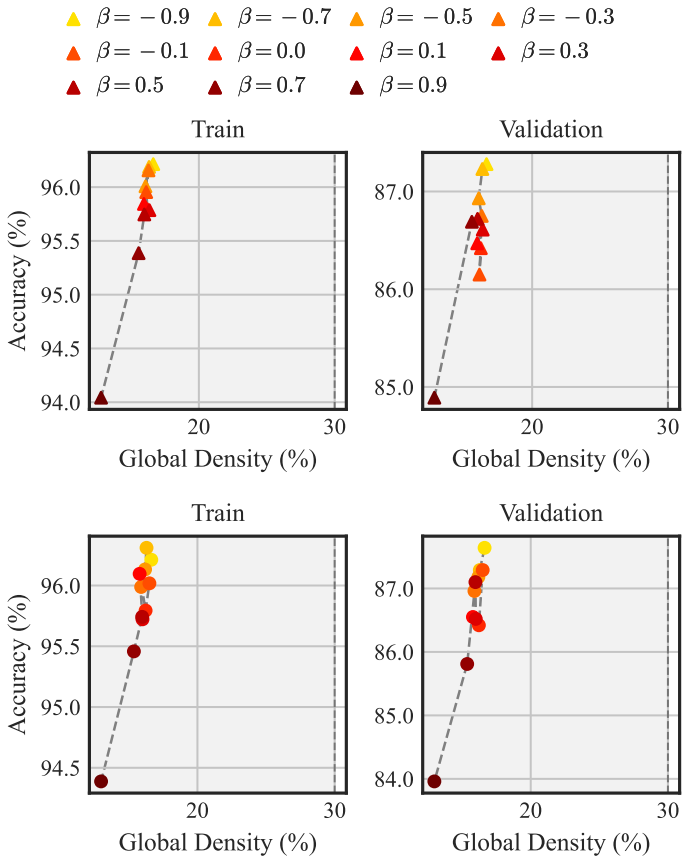


Figure D.15: Trade-off plot under a 30% global density target for Nesterov (top) and Polyak (bottom) momentum.

Table D.10: Ablation on the momentum hyperparameter for Nesterov on a CIFAR10 task with a global density target of $\epsilon = 30\%$.

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 D.11: Ablation on the momentum hyperparameter for Polyak on a CIFAR10 task with a global density target of $\epsilon = 30\%$.

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

Bibliography

[Aba+15] Martín Abadi et al.: **TensorFlow, Large-scale machine learning on heterogeneous systems**. Nov. 2015 (cit. on p. 13).

[Aga+18] Alekh Agarwal, Alina Beygelzimer, Miroslav Dudík, John Langford, and Hanna Wallach: **A Reductions Approach to Fair Classification**. In: *ICML*. 2018 (cit. on p. 66).

[Alt99] Eitan Altman: **Constrained Markov Decision Processes**. Routledge, 1999 (cit. on p. 5).

[An+18] Wangpeng An, Haoqian Wang, Qingyun Sun, Jun Xu, Qionghai Dai, and Lei Zhang: **A PID Controller Approach for Stochastic Optimization of Deep Networks**. In: *CVPR*. 2018 (cit. on pp. 81, 86, 177).

[ACH18] Sanjeev Arora, Nadav Cohen, and Elad Hazan: **On the Optimization of Deep Networks: Implicit Acceleration by Overparameterization**. In: *ICML*. 2018 (cit. on p. 55).

[AHU58] K.J. Arrow, L. Hurwicz, and H. Uzawa: **Studies in Linear and Nonlinear Programming**. Stanford University Press, 1958 (cit. on pp. 24, 80–82).

[ÅH95] K.J. Åström and T. Hägglund: **PID Controllers**. International Society for Measurement and Control, 1995 (cit. on pp. 9, 80, 86).

[Bak+20] Michiel Bakker, Humberto Riverón Valdés, Patrick D Tu, Krishna P Gummadi, Kush R Varshney, Adrian Weller, and Alex Sandy Pentland: **Fair Enough: Improving Fairness in Budget-Constrained Decision Making Using Confidence Thresholds**. In: *CEUR-WS*. 2020 (cit. on p. 66).

[Bal+23] Randall Balestriero et al.: **A Cookbook of Self-Supervised Learning**. In: *arXiv:2304.12210*, (2023) (cit. on p. 11).

[BPL22] Adrien Bardes, Jean Ponce, and Yann LeCun: **VICReg: Variance-Invariance-Covariance Regularization for Self-Supervised Learning**. In: *ICLR*. 2022 (cit. on pp. 5, 13, 108).

[BHN23] Solon Barocas, Moritz Hardt, and Arvind Narayanan: **Fairness and Machine Learning: Limitations and Opportunities**. MIT Press, 2023 (cit. on p. 31).

[Bas+22] Sourya Basu, Jose Gallego-Posada, Francesco Viganò, James Rowbottom, and Taco Cohen: **Equivariant Mesh Attention Networks**. In: *TMLR*, (2022) (cit. on pp. ix, 10).

[BK96] Barry Becker and Ronny Kohavi: **Adult**. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5XW20. 1996 (cit. on pp. 91, 191).

[Bég+20] Jean Bégaint, Fabien Racapé, Simon Feltman, and Akshay Pushparaja: **CompressAI: a PyTorch library and evaluation platform for end-to-end compression research**. In: *arXiv:2201.12904*, (2020) (cit. on p. 146).

[Bel+19] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal: **Reconciling modern machine-learning practice and the classical bias–variance trade-off**. In: *arXiv:1812.11118*, (2019) (cit. on p. 111).

[BCV13] Yoshua Bengio, Aaron Courville, and Pascal Vincent: **Representation Learning: A Review and New Perspectives**. In: *IEEE TPAMI*, **35**:8 (2013), pp. 1798–1828 (cit. on p. 37).

[Ber+21] Richard Berk, Hoda Heidari, Shahin Jabbari, Michael Kearns, and Aaron Roth: **Fairness in Criminal Justice Risk Assessments: The State of the Art**. In: *Sociological Methods & Research*, (2021) (cit. on p. 66).

[Ber75] D Bertsekas: **On the Method of Multipliers for Convex Programming**. In: *IEEE transactions on automatic control*, (1975) (cit. on p. 81).

[Ber16] D. Bertsekas: **Nonlinear Programming**. Athena Scientific, 2016 (cit. on pp. 17–18, 20–23, 25, 29–30, 79, 81, 101, 153).

[Ber76] Dimitri P Bertsekas: **On the Goldstein-Levitin-Polyak Gradient Projection Method**. In: *IEEE Transactions on automatic control*, (1976) (cit. on p. 81).

[Ber82] Dimitri P. Bertsekas: **Constrained Optimization and Lagrange Multiplier Methods**. New York: Academic Press, 1982 (cit. on p. 29).

[Bez+23] Aleksandr Beznosikov, Eduard Gorbunov, Hugo Berard, and Nicolas Loizou: **Stochastic Gradient Descent-Ascent: Unified Theory and New Efficient Methods**. In: *AISTATS*. 2023 (cit. on p. 66).

[BM14] Ernesto G. Birgin and José Mario Martínez: **Practical Augmented Lagrangian Methods for Constrained Optimization**. In: *Fundamentals of Algorithms*. 2014 (cit. on p. 29).

[BB23] Christopher Michael Bishop and Hugh Bishop: **Deep Learning - Foundations and Concepts**. Ed. by Springer. 1st ed. 2023 (cit. on p. 13).

[Bla+20] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag: **What is the State of Neural Network Pruning?** In: *MLSys*. 2020 (cit. on pp. 7, 72).

[Blo+22] Mathieu Blondel, Quentin Berthet, Marco Cuturi, Roy Frostig, Stephan Hoyer, Felipe Llinares-López, Fabian Pedregosa, and Jean-Philippe Vert: **Efficient and Modular Implicit Differentiation**. In: *NeurIPS*. 2022 (cit. on p. 100).

[Bom+21] Rishi Bommasani et al.: **On the Opportunities and Risks of Foundation Models**. In: *arXiv:2108.07258*, (2021) (cit. on p. 63).

[BCN18] Léon Bottou, Frank E. Curtis, and Jorge Nocedal: **Optimization Methods for Large-Scale Machine Learning**. In: *SIAM Review*, **60**:2 (2018), pp. 223–311 (cit. on p. 79).

[Boy21] Stephen Boyd: **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. 187).

[BV04] Stephen Boyd and Lieven Vandenberghe: **Convex Optimization**. Cambridge University Press, 2004 (cit. on pp. 5, 19–20, 23, 39, 43, 55, 79, 101).

[Bra+18] James Bradbury et al.: **JAX: composable transformations of Python+NumPy programs**. Version 0.3.13. 2018. URL: http://github.com/google/jax (cit. on p. 104).

[Bro+20] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al.: **Language Models are Few-Shot Learners**. In: *NeurIPS*. 2020 (cit. on p. 37).

[Bub+15] Sébastien Bubeck et al.: **Convex optimization: Algorithms and complexity**. In: *Foundations and Trends® in Machine Learning*, **8**:3-4 (2015), pp. 231–357 (cit. on p. 14).

[CI18] Miguel A. Carreira-Perpinan and Yerlan Idelbayev: **"Learning - Compression" Algorithms for Neural Net Pruning**. In: *CVPR*. 2018 (cit. on p. 44).

[CLG01] Rich Caruana, Steve Lawrence, and Lee Giles: **Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and Early Stopping**. In: *NeurIPS*. 2001 (cit. on p. 37).

[Cas+23] Umberto Casti, Nicola Bastianello, Ruggero Carli, and Sandro Zampieri: **A Control Theoretical Approach to Online Constrained Optimization**. In: *arXiv preprint arXiv:2309.15498*, (2023) (cit. on pp. 82, 177).

[CR20] Luiz Chamon and Alejandro Ribeiro: **Probably Approximately Correct Constrained Learning**. In: *NeurIPS*. 2020 (cit. on p. 153).

[Cha+22] Luiz FO Chamon, Santiago Paternain, Miguel Calvo-Fullana, and Alejandro Ribeiro: **Constrained Learning with Non-Convex Losses**. In: *IEEE Transactions on Information Theory*, (2022) (cit. on p. 153).

[Cha30] Sydney Chapman: **A Theory of Upper-Atmospheric Ozone**. In: *Memoirs of the Royal Meteorological Society*, **3**:26 (1930), pp. 103–125 (cit. on p. 3).

[Cha+19] Tatjana Chavdarova, Gauthier Gidel, François Fleuret, and Simon Lacoste-Julien: **Reducing Noise in GAN Training with Variance Reduced Extragradient**. In: *NeurIPS*. 2019 (cit. on p. 66).

[CR97] George HG Chen and R Tyrrell Rockafellar: **Convergence Rates in Forward–Backward Splitting**. In: *SIAM Journal on Optimization*, **7**:2 (1997), pp. 421–444 (cit. on p. 120).

[Che21] Yaofo Chen: **PyTorch CIFAR Models**. https://github.com/chenyaofo/pytorch-cifar-models. 2021 (cit. on pp. 159, 161, 176).

[Cho+19] Dami Choi, Christopher J. Shallue, Zachary Nado, Jaehoon Lee, Chris J. Maddison, and George E. Dahl: **On Empirical Comparisons of Optimizers for Deep Learning**. In: *arXiv:1910.05446*, (2019) (cit. on p. 69).

[Cho+15] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun: **The Loss Surfaces of Multilayer Networks**. In: *AISTATS*. 2015 (cit. on p. 111).

[Chr+17] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei: **Deep Reinforcement Learning from Human Preferences**. In: 2017 (cit. on pp. 99, 110).

[CV95] Corinna Cortes and Vladimir Vapnik: **Support-vector networks**. In: *Machine Learning*, **20**: (1995), pp. 273–297 (cit. on p. 5).

[Cot+] Andrew Cotter et al.: **TensorFlow Constrained Optimization (TFCO)**. https://github.com/google-research/tensorflow_constrained_optimization (cit. on p. 100).

[Cot+19a] Andrew Cotter, Maya Gupta, Heinrich Jiang, Nathan Srebro, Karthik Sridharan, Serena Wang, Blake Woodworth, and Seungil You: **Training Well-Generalizing Classifiers for Fairness Metrics and Other Data-Dependent Constraints**. In: *ICML*. 2019 (cit. on p. 110).

[Cot+19b] Andrew Cotter, Heinrich Jiang, Maya R Gupta, Serena Wang, Taman Narayan, Seungil You, and Karthik Sridharan: **Optimization with Non-Differentiable Constraints with Applications to Fairness, Recall, Churn, and Other Goals**. In: *JMLR*, (2019) (cit. on pp. 7–8, 31–32, 43, 45, 51, 56, 65–66, 69, 79, 91, 99, 101, 109, 120, 146, 153, 191).

[Cou24] Council of the European Union: **Regulation of the European Parliament and of the Council laying down harmonised rules on artificial intelligence and amending Regulations (EC) No 300/2008, (EU) No 167/2013, (EU) No 168/2013, (EU) 2018/858, (EU) 2018/1139 and (EU) 2019/2144 and Directives 2014/90/EU, (EU) 2016/797 and (EU) 2020/1828 (Artificial Intelligence Act)**. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=EP:P9_TA(2024)0138. 2024 (cit. on pp. 4, 99).

[Cou43] Richard Courant: **Variational methods for the solution of problems of equilibrium and vibrations**. In: *Bulletin of the American Mathematical Society*, **49**:1 (1943), pp. 1–23 (cit. on p. 29).

[Cro+13] Timothy Crosley et al.: **isort**. https://github.com/PyCQA/isort. 2013 (cit. on p. 102).

[CSP92] S. W. Crown, H. N. Shapiro, and M. B. Pate: **A Comparison Study of the Thermal Performance of R12 and R134a**. In: *International Refrigeration and Air Conditioning Conference*. 1992 (cit. on p. 3).

[Dah+23] George E. Dahl et al.: **Benchmarking Neural Network Training Algorithms**. In: *arXiv:2306.07179*, (2023) (cit. on pp. 23, 81, 83).

[Dai+18] Bin Dai, Chen Zhu, Baining Guo, and David Wipf: **Compressing Neural Networks using the Variational Information Bottleneck**. In: *ICML*. 2018 (cit. on p. 44).

[Dai+24] Josef Dai, Xuehai Pan, Ruiyang Sun, Jiaming Ji, Xinbo Xu, Mickel Liu, Yizhou Wang, and Yaodong Yang: **Safe RLHF: Safe Reinforcement Learning from Human Feedback**. In: *ICLR*. 2024 (cit. on p. 99).

[Dee+20] DeepMind et al.: **The DeepMind JAX Ecosystem**. http://github.com/google-deepmind. 2020 (cit. on p. 13).

[DBL14] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien: **SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives**. In: *NeurIPS*. 2014 (cit. on p. 66).

[DK21a] J. Degrave and I. Korshunova: **How we can make machine learning algorithms tunable**. Engraved, blog: www.engraved.blog/how-we-can-make-machine-learning-algorithms-tunable/. 2021 (cit. on p. 35).

[DK21b] J. Degrave and I. Korshunova: **Why machine learning algorithms are hard to tune and how to fix it**. Engraved, blog: www.engraved.blog/why-machine-learning-algorithms-are-hard-to-tune/. 2021 (cit. on p. 35).

[Den+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei: **ImageNet: A Large-Scale Hierarchical Image Database**. In: *CVPR*. 2009 (cit. on pp. 48–49, 128).

[DB16] Steven Diamond and Stephen Boyd: **CVXPY: A Python-Embedded Modeling Language for Convex Optimization**. In: *JMLR - MLOSS*, (2016) (cit. on pp. 100, 102).

[DMB16] William Dieterich, Christina Mendoza, and Tim Brennan: **COMPAS Risk Scales: Demonstrating Accuracy Equity and Predictive Parity**. In: *Northpointe Inc*, (2016) (cit. on p. 66).

[Dik67] I I Dikin: **Iterative Solution of Problems of Linear and Quadratic Programming**. In: *Doklady Akademii Nauk*. Russian Academy of Sciences. 1967 (cit. on p. 81).

[DAV18] Marc-Antoine Dilhac, Cristophe Abrassart, and Nathalie Voarino: **Montréal Declaration for a Responsible Development of Artificial Intelligence**. 2018 (cit. on pp. 4, 99).

[DHS11] John Duchi, Elad Hazan, and Yoram Singer: **Adaptive Subgradient Methods for Online Learning and Stochastic Optimization**. In: *Journal of Machine Learning Research*, **12**:61 (2011), pp. 2121–2159 (cit. on p. 14).

[Dup+21] Emilien Dupont, Adam Golinski, Milad Alizadeh, Yee Whye Teh, and Arnaud Doucet: **COIN: COmpression with Implicit Neural Representations**. In: *ICLR - Neural Compression Workshop*. 2021 (cit. on pp. 7, 51, 53, 55, 57, 147).

[Dup+22] Emilien Dupont, Hrushikesh Loya, Milad Alizadeh, Adam Goliński, Yee Whye Teh, and Arnaud Doucet: **COIN++: Data Agnostic Neural Compression**. In: *arXiv:2201.12904*, (2022) (cit. on pp. 53, 59).

[DTD22] Emilien Dupont, Yee Whye Teh, and Arnaud Doucet: **Generative Models as Distributions of Functions**. In: *AISTATS*. 2022 (cit. on p. 53).

[Dwo+12] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel: **Fairness Through Awareness**. In: *Innovations in Theoretical Computer Science*. 2012 (cit. on pp. 5, 66).

[ENR22] Juan Elenter, Navid NaderiAlizadeh, and Alejandro Ribeiro: **A Lagrangian Duality Approach to Active Learning**. In: *NeurIPS*. 2022 (cit. on pp. 66, 79, 99, 153).

[Evc+20] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsenl: **Rigging the Lottery: Making All Tickets Winners**. In: *ICML*. 2020 (cit. on pp. 44, 161).

[Evc+22] Utku Evci, Yani Ioannou, Cem Keskin, and Yann Dauphin: **Gradient Flow in Sparse Neural Networks and How Lottery Tickets Win**. In: *AAAI*. 2022 (cit. on p. 142).

[FG21] Amir-Massoud Farahmand and Mohammad Ghavamzadeh: **PID Accelerated Value Iteration Algorithm**. In: *ICML*. 2021 (cit. on p. 79).

[FGS85] Joseph C. Farman, Brian George Gardiner, and Jonathan D. Shanklin: **Large losses of total ozone in Antarctica reveal seasonal ClOx/NOx interaction**. In: *Nature*, **315**: (1985), pp. 207–210 (cit. on p. 3).

[Fio+20] Ferdinando Fioretto, Pascal Van Hentenryck, Terrence W. K. Mak, Cuong Tran, Federico Baldo, and Michele Lombardi: **Lagrangian Duality for Constrained Deep Learning**. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. 2020 (cit. on pp. 45, 79).

[Fis88] R. A. Fisher: **Iris Dataset**. UCI Machine Learning Repository. https://doi.org/10.24432/C56C76. 1988 (cit. on pp. 90, 188).

[FGW02] Anders Forsgren, Philip E. Gill, and Margaret H. Wright: **Interior Methods for Nonlinear Optimization**. In: *SIAM Review*, **44**:4 (2002), pp. 525–597 (cit. on p. 24).

[FW56] Marguerite Frank and Philip Wolfe: **An algorithm for quadratic programming**. In: *Naval Research Logistics Quarterly*, **3**:1-2 (1956), pp. 95–110 (cit. on pp. 24, 43, 81).

[FC19] Jonathan Frankle and Michael Carbin: **The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks**. In: *ICLR*. 2019 (cit. on pp. 37, 44).

[Fra+20] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin: **Linear Mode Connectivity and the Lottery Ticket Hypothesis**. In: *ICML*. 2020 (cit. on p. 44).

[FSV21] Sorelle A. Friedler, Carlos Scheidegger, and Suresh Venkatasubramanian: **The (Im)Possibility of Fairness: Different Value Systems Require Different Mechanisms for Fair Decision Making**. In: *Commun. ACM*, (2021) (cit. on p. 75).

[GEH19] Trevor Gale, Erich Elsen, and Sara Hooker: **The State of Sparsity in Deep Neural Networks**. In: *NeurIPS - Workshop in ODML-CDNNR*. 2019 (cit. on pp. 6, 37–39, 45–46, 50, 55, 63, 72, 122, 125, 140–141, 159, 161, 190).

[Gal+22] Jose Gallego-Posada, Juan Ramirez, Akram Erraqabi, Yoshua Bengio, and Simon Lacoste-Julien: **Controlled Sparsity via Constrained Optimization or: How I Learned to Stop Tuning Penalties and Love Constraints**. In: *NeurIPS*. 2022 (cit. on pp. ix, 5, 9, 55–56, 66, 79–80, 85, 92–93, 99–101, 146, 153, 190–191).

[Gal+24] Jose Gallego-Posada, Juan Ramirez, Meraj Hashemizadeh, and Simon Lacoste-Julien: **Cooper: Constrained Optimization for Deep Learning**. *MLOSS (under submission)* `https://github.com/cooper-org/cooper`. 2024 (cit. on pp. ix, 10, 72, 76, 79, 90, 95, 126, 146, 159, 188).

[Gal+20] Jose Gallego-Posada, Ankit Vani, Max Schwarzer, and Simon Lacoste-Julien: **GAIT: A Geometric Approach to Information Theory**. In: *AISTATS*. 2020 (cit. on pp. ix, 10).

[Gho+21] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael Mahoney, and Kurt Keutzer: **A Survey of Quantization Methods for Efficient Neural Network Inference**. In: *arXiv:2103.13630*, (2021) (cit. on p. 63).

[Gid+19a] Gauthier Gidel, Reyhane Askari, Mohammad Pezeshki, Remi LePriol, Gabriel Huang, Simon Lacoste-Julien, and Ioannis Mitliagkas: **Negative Momentum for Improved Game Dynamics**. In: *AISTATS*. 2019 (cit. on pp. 9, 80–81, 88, 90, 185, 189).

[Gid+19b] Gauthier Gidel, Hugo Berard, Gaëtan Vignoud, Pascal Vincent, and Simon Lacoste-Julien: **A Variational Inequality Perspective on Generative Adversarial Networks**. In: *ICLR*. 2019 (cit. on pp. 27–28, 69, 81, 83, 101, 120, 153).

[Gio18] Giorgio Giorgi: **A Guided Tour in Constraint Qualifications for Nonlinear Programming under Differentiability Assumptions**. DEM Working Papers Series 160. *University of Pavia, Department of Economics and Management*, June 2018 (cit. on p. 21).

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville: **Deep Learning**. MIT press Cambridge, 2016 (cit. on pp. 5, 12–13, 26).

[Goo+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio: **Generative Adversarial Networks**. In: *NeurIPS*. 2014 (cit. on p. 153).

[GSS15] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy: **Explaining and Harnessing Adversarial Examples**. In: *NeurIPS*. 2015 (cit. on p. 5).

[Gow+20] Robert Gower, Mark Schmidt, Francis Bach, and Peter Richtárik: **Variance-Reduced Methods for Machine Learning**. In: *Proceedings of the IEEE*, (2020) (cit. on p. 66).

[Gra24] Nico Grant: **Google Rolls Back A.I. Search Feature After Flubs and Flaws**. In: *The New York Times*, (June 1, 2024). URL: `https://www.nytimes.com/2024/06/01/technology/google-ai-overviews-rollback.html` (visited on 06/04/2024) (cit. on p. 4).

[GYC16] Yiwen Guo, Anbang Yao, and Yurong Chen: **Dynamic Network Surgery for Efficient DNNs**. In: *NeurIPS*. 2016 (cit. on p. 44).

[HB70] Pierre C. Haarhoff and J. D. Buys: **A New Method for the Optimization of a Nonlinear Function Subject to Nonlinear Constraints**. In: *The Computer Journal*, **13**:2 (1970), pp. 178–184 (cit. on p. 29).

[HMD16] Song Han, Huizi Mao, and William J Dally: **Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding**. In: *ICLR*. 2016 (cit. on pp. 37–38, 44).

[HPS16] Moritz Hardt, Eric Price, and Nati Srebro: **Equality of Opportunity in Supervised Learning**. In: *NeurIPS*. 2016 (cit. on p. 66).

[Has+24] Meraj Hashemizadeh, Juan Ramirez, Rohan Sukumaran, Golnoosh Farnadi, Simon Lacoste-Julien, and Jose Gallego-Posada: **Balancing Act: Constraining Disparate Impact in Sparse Models**. In: *ICLR*. 2024 (cit. on pp. ix, 7, 79, 99–100).

[He+16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun: **Deep Residual Learning for Image Recognition**. In: *CVPR*. 2016 (cit. on pp. 46, 72, 93, 122, 159, 161, 173, 190).

[Hes69] Magnus R Hestenes: **Multiplier and gradient methods**. In: *Journal of Optimization Theory and Applications*, **4**:5 (1969), pp. 303–320 (cit. on p. 29).

[Hig+17] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner: **beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework**. In: *ICLR*. 2017 (cit. on pp. 5, 13).

[HVD15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean: **Distilling the Knowledge in a Neural Network**. In: *Deep Learning and Representation Learning Workshop at NeurIPS*. 2015 (cit. on p. 63).

[Hoo+19] Sara Hooker, Aaron Courville, Gregory Clark, Yann Dauphin, and Andrea Frome: **What Do Compressed Deep Neural Networks Forget?** In: *arXiv:1911.05248*, (2019) (cit. on pp. 7, 43, 61, 63, 65).

[Hoo+20] Sara Hooker, Nyalleng Moorosi, Gregory Clark, Samy Bengio, and Emily Denton: **Characterising Bias in Compressed Models**. In: *arXiv: 2010.03058*, (2020) (cit. on pp. 7, 63, 65).

[HCR23] Ignacio Hounie, Luiz Chamon, and Alejandro Ribeiro: **Automatic Data Augmentation via Invariance-Constrained Learning**. In: *ICML*. 2023 (cit. on p. 153).

[HER23] Ignacio Hounie, Juan Elenter, and Alejandro Ribeiro: **Neural Networks with Quantization Constraints**. In: *ICASSP*. 2023 (cit. on pp. 79, 99).

[HL17] Bin Hu and Laurent Lessard: **Control Interpretations for First-Order Optimization Methods**. In: *American Control Conference*. 2017 (cit. on pp. 82, 177).

[Jag13] Martin Jaggi: **Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization**. In: *ICML*. 2013 (cit. on p. 43).

[Jan+24] Doseok Jang, Larry Yan, Lucas Spangher, and Costas J Spanos: **Active Reinforcement Learning for Robust Building Control**. In: *AAAI*. 2024 (cit. on p. 100).

[JGP17] Eric Jang, Shixiang Gu, and Ben Poole: **Categorical Reparameterization with Gumbel-Softmax**. In: *ICLR*. 2017 (cit. on pp. 40, 145).

[KJ21] Kimmo Kärkkäinen and Jungseock Joo: **FairFace: Face Attribute Dataset for Balanced Race, Gender, and Age**. In: *WACV*. 2021 (cit. on pp. 71, 75, 159, 161).

[KB15] Diederik Kingma and Jimmy Ba: **Adam: A Method for Stochastic Optimization**. In: *ICLR*. 2015 (cit. on pp. 9, 14, 90, 102, 109, 162, 190).

[Kod91] Kodak: **Kodak Dataset**. http://r0k.us/graphics/kodak/. 1991 (cit. on p. 57).

[Kor76] Galina M Korpelevich: **The Extragradient Method for Finding Saddle Points and Other Problems**. In: *Ekonomika i Matematicheskie Metody*, (1976) (cit. on pp. 27–28, 44, 69, 81, 101, 120, 153).

[Kos+22] Shin Koseki et al.: **AI & Cities: Risks, Applications and Governance**. Tech. rep. *Mila-UN Habitat*, 2022 (cit. on pp. ix, 10).

[Kri09] Alex Krizhevsky: **Learning Multiple Layers of Features from Tiny Images**. Tech. rep. Toronto, Ontario: *University of Toronto*, 2009 (cit. on pp. 71, 93, 159).

[Kun+20] Souvik Kundu, Mahdi Nazemi, Massoud Pedram, Keith M Chugg, and Peter A Beerel: **Pre-defined Sparsity for Low-Complexity Convolutional Neural Networks**. In: *IEEE Transactions on Computers*, **69**:7 (2020), pp. 1045–1058 (cit. on p. 128).

[Lac+24] Sébastien Lachapelle, Pau Rodríguez López, Yash Sharma, Katie Everett, Rémi Le Priol, Alexandre Lacoste, and Simon Lacoste-Julien: **Nonparametric Partial Disentanglement via Mechanism Sparsity: Sparse Actions, Interventions and Sparse Temporal Dependencies**. In: *arXiv:2401.04890*, (2024) (cit. on pp. 100, 109).

[Lan+18] Łukasz Langa et al.: **Black: The uncompromising Python code formatter**. https://github.com/psf/black. 2018 (cit. on p. 102).

[LDS90] Yann LeCun, John S Denker, and Sara A Solla: **Optimal Brain Damage**. In: *NeurIPS*. 1990 (cit. on pp. 38, 44).

[Lee+21] Jaeho Lee, Jihoon Tack, Namhoon Lee, and Jinwoo Shin: **Meta-Learning Sparse Implicit Neural Representations**. In: *NeurIPS*. 2021 (cit. on pp. 53, 59).

[LAJ19] Carl Lemaire, Andrew Achkar, and Pierre-Marc Jodoin: **Structured Pruning of Neural Networks With Budget-Aware Regularization**. In: *CVPR*. 2019 (cit. on pp. 38, 45).

[LRP16] Laurent Lessard, Benjamin Recht, and Andrew Packard: **Analysis and Design of Optimization Algorithms via Integral Quadratic Constraints**. In: *SIAM Journal on Optimization*, **26**:1 (2016), pp. 57–95 (cit. on pp. 10, 86).

[LS22] Kevin Leyton-Brown and Yoav Shoham: **Essentials of game theory: A concise multidisciplinary introduction**. Springer Nature, 2022 (cit. on p. 6).

[Lez21] Mario Lezcano-Casado: **GeoTorch**. https://github.com/lezcano/geotorch. 2021 (cit. on p. 100).

[LKJ17] Fei-Fei Li, Andrej Karpathy, and Justin Johnson: **Tiny ImageNet**. https://www.kaggle.com/c/tiny-imagenet. 2017 (cit. on pp. 38, 48, 128).

[Li+17] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf: **Pruning Filters for Efficient ConvNets**. In: *ICLR*. 2017 (cit. on pp. 38, 44, 49, 55, 118–119, 130).

[Li+21] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang: **Neural Scene Flow Fields for Space-Time View Synthesis of Dynamic Scenes**. In: *CVPR*. 2021 (cit. on p. 53).

[LJJ20] Tianyi Lin, Chi Jin, and Michael Jordan: **On Gradient Descent Ascent for Nonconvex-Concave Minimax Problems**. In: *ICML*. 2020 (cit. on pp. 24–25, 42, 66, 80–81, 111, 119–120, 153).

[LKJ22] Xiaofeng Lin, Seungbae Kim, and Jungseock Joo: **FairGRAPE: Fairness-Aware GRAdient Pruning mEthod for Face Attribute Classification**. In: *ECCV*. 2022 (cit. on pp. 8, 64–65, 71–72, 165–166, 173).

[LN89] Dong C Liu and Jorge Nocedal: **On the limited memory BFGS method for large scale optimization**. In: *Mathematical Programming*, **45**:1 (1989), pp. 503–528 (cit. on p. 14).

[LH17] Ilya Loshchilov and Frank Hutter: **SGDR: Stochastic Gradient Descent with Warm Restarts**. In: *ICLR*. 2017 (cit. on p. 190).

[LWK18] Christos Louizos, Max Welling, and Diederik P Kingma: **Learning Sparse Neural Networks through L_0 Regularization**. In: *ICLR*. 2018 (cit. on pp. 5–6, 13, 37–42, 44–47, 49–50, 54–56, 92, 115–117, 122–124, 127–131, 133, 135, 141, 145, 148, 189–190, 197).

[Low+21] Andrew Lowy, Sina Baharlouei, Rakesh Pavan, Meisam Razaviyayn, and Ahmad Beirami: **A Stochastic Optimization Framework for Fair Risk Minimization**. In: *TMLR*, (2021) (cit. on p. 66).

[MMT17] Chris J Maddison, Andriy Mnih, and Yee Whye Teh: **The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables**. In: *ICLR*. 2017 (cit. on pp. 40, 145, 190).

[Mad+18] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu: **Towards Deep Learning Models Resistant to Adversarial Attacks**. In: *ICLR*. 2018 (cit. on p. 153).

[Mal+20] Eran Malach, Gilad Yehudai, Shai Shalev-Schwartz, and Ohad Shamir: **Proving the Lottery Ticket Hypothesis: Pruning is All You Need**. In: *ICML*. 2020 (cit. on p. 44).

[Meh+21] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan: **A Survey on Bias and Fairness in Machine Learning**. In: *ACM Computing Surveys (CSUR)*, (2021) (cit. on p. 66).

[Mes+19] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger: **Occupancy Networks: Learning 3d Reconstruction in Function Space**. In: *CVPR*. 2019 (cit. on p. 53).

[MH30] Thomas Jr. Midgley and Albert L. Henne: **Organic Fluorides as Refrigerants1**. In: *Industrial & Engineering Chemistry*, **22**:5 (1930), pp. 542–545 (cit. on p. 3).

[Mni+13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller: **Playing Atari with Deep Reinforcement Learning**. In: *NeurIPS Deep Learning Workshop*. 2013 (cit. on pp. 8, 66, 70).

[MOP20a] Aryan Mokhtari, Asuman E Ozdaglar, and Sarath Pattathil: **Convergence Rate of O(1/k) for Optimistic Gradient and Extragradient Methods in Smooth Convex-Concave Saddle Point Problems**. In: *SIAM Journal on Optimization*, (2020) (cit. on pp. 81, 153).

[MOP20b] Aryan Mokhtari, Asuman E. Ozdaglar, and Sarath Pattathil: **A Unified Analysis of Extra-gradient and Optimistic Gradient Methods for Saddle Point Problems: Proximal Point Approach**. In: *AISTATS*. 2020 (cit. on p. 86).

[MAV17] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov: **Variational Dropout Sparsifies Deep Neural Networks**. In: *ICML*. 2017 (cit. on pp. 38, 44).

[MW19] Hesham Mostafa and Xin Wang: **Parameter Efficient Training of Deep Convolutional Neural Networks by Dynamic Sparse Reparameterization**. In: *ICML*. 2019 (cit. on p. 44).

[Nan+19] Yatin Nandwani, Abhishek Pathak, Mausam, and Parag Singla: **A Primal Dual Formulation For Deep Learning With Constraints**. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. 2019 (cit. on p. 45).

[NP20] Geoffrey Negiar and Fabian Pedregosa: **CHOP: continuous optimization built on Pytorch**. `https://github.com/openopt/chop`. 2020 (cit. on p. 100).

[Nek+17] Kirill Neklyudov, Dmitry Molchanov, Arsenii Ashukha, and Dmitry P Vetrov: **Structured Bayesian Pruning via Log-Normal Multiplicative Noise**. In: *NeurIPS*. 2017 (cit. on pp. 38, 44).

[Nem04] Arkadi Nemirovski: **Prox-Method with Rate of Convergence O(1/T) for Variational Inequalities with Lipschitz Continuous Monotone Operators and Smooth Convex-Concave Saddle Point Problems**. In: *SIAM Journal on Optimization*, **15**:1 (2004), pp. 229–251 (cit. on p. 120).

[Nes83] Yurii Evgen'evich Nesterov: **A method of solving a convex programming problem with convergence rate O**$(1/k^2)$. In: *Doklady Akademii Nauk*. Vol. 269. 3. Russian Academy of Sciences. 1983, pp. 543–547 (cit. on pp. 9, 14, 86, 90).

[Neu28] John von Neumann: **Zur Theorie der Gesellschaftsspiele**. In: *Mathematische Annalen*, **100**:1 (1928), pp. 295–320 (cit. on pp. 119, 153).

[Ney+17] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nathan Srebro: **Exploring Generalization in Deep Learning**. In: *NeurIPS*. 2017 (cit. on p. 108).

[NTS15] Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro: **In Search of the Real Inductive Bias: On the Role of Implicit Regularization in Deep Learning**. In: *ICLR - Workshop*. 2015 (cit. on p. 37).

[NW06] Jorge Nocedal and Stephen J. Wright: **Numerical Optimization**. Springer, 2006 (cit. on pp. 21–23, 29–30, 79, 81, 101, 110).

[Oni21] The Onion: **Geologists Recommend Eating At Least One Small Rock Per Day**. Apr. 13, 2021. URL: https://www.theonion.com/marvel-not-even-bothering-to-replace-green-screens-with-1850647137 (visited on 06/06/2024) (cit. on p. 4).

[Ope23] OpenAI: **GPT-4 Technical Report**. In: *arXiv:2303.08774*, (2023) (cit. on p. 63).

[Ouy+22] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al.: **Training language models to follow instructions with human feedback**. In: 2022 (cit. on pp. 99, 110).

[Pag20] Michela Paganini: **Prune Responsibly**. In: *arXiv:2009.09936*, (2020) (cit. on pp. 7, 63, 65).

[Pas+19] Adam Paszke et al.: **PyTorch: An Imperative Style, High-Performance Deep Learning Library**. In: *NeurIPS*. 2019 (cit. on pp. 10, 13, 72, 76, 90, 95, 100, 126, 137, 146, 159, 188).

[Per+17] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville: **FiLM: Visual Reasoning with a General Conditioning Layer**. In: *AAAI*. 2017 (cit. on pp. 51, 53).

[PB87] John Platt and Alan Barr: **Constrained Differential Optimization**. In: *NeurIPS*. 1987 (cit. on pp. 9, 80–81, 101).

[Pol64] Boris T Polyak: **Some methods of speeding up the convergence of iteration methods**. In: *USSR Computational Mathematics and Mathematical Physics*, **4**:5 (1964), pp. 1–17 (cit. on pp. 9, 14, 86, 90).

[Pop80] Leonid Denisovich Popov: **A modification of the Arrow-Hurwicz method for search of saddle points**. In: *Mathematical notes of the Academy of Sciences of the USSR*, **28**: (1980), pp. 845–848 (cit. on pp. 28, 81, 86).

[Pow69] M. J. D. Powell: **A method for nonlinear constraints in minimization problems**. In: *Optimization*. Ed. by R. Fletcher. New York: Academic Press, 1969, pp. 283–298 (cit. on p. 29).

[Pri23] Simon Prince: **Understanding Deep Learning**. MIT Press, 2023 (cit. on p. 13).

[Raf+23] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn: **Direct Preference Optimization: Your Language Model is Secretly a Reward Model**. In: *NeurIPS*. 2023 (cit. on pp. 5, 13).

[Ram+22] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen: **Hierarchical Text-Conditional Image Generation with CLIP Latents**. In: *CVPR*. 2022 (cit. on p. 63).

[RG22] Juan Ramirez and Jose Gallego-Posada: **L_0onie: Compressing COINs with L_0-Constraints**. In: *Sparsity in Neural Networks Workshop*. 2022 (cit. on pp. ix, 6, 100).

[Rec18] Ben Recht: **The Best Things in Life Are Model Free**. arg min, blog: https://archives.argmin.net/2018/04/19/pid/. 2018 (cit. on pp. 10, 82).

[Rei24] Liz Reid: **What happened with AI overviews and next steps**. Ed. by Google. 2024. URL: https://blog.google/products/search/ai-overviews-update-may-2024 (visited on 06/01/2024) (cit. on p. 4).

[RM51] Herbert Robbins and Sutton Monro: **A Stochastic Approximation Method**. In: *Annals of Mathematical Statistics*, **22**:3 (1951), pp. 400–407 (cit. on p. 14).

[San+18] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen: **MobileNetV2: Inverted Residuals and Linear Bottlenecks**. In: *CVPR*. 2018 (cit. on pp. 72, 159, 161, 166).

[SSM20] Pedro Savarese, Hugo Silva, and Michael Maire: **Winning the Lottery with Continuous Sparsification**. In: *NeurIPS*. 2020 (cit. on p. 129).

[ST22] Jonathan Schwarz and Yee Whye Teh: **Meta-Learning Sparse Compression Networks**. In: *arXiv:2205.08957*, (2022) (cit. on pp. 51, 53, 55, 59).

[She+18] Li Shen, Congliang Chen, Fangyu Zou, Zequn Jie, Ju Sun, and Wei Liu: **A Unified Analysis of AdaGrad with Weighted Aggregation and Momentum Acceleration**. In: *IEEE TNNLS*. 2018 (cit. on p. 87).

[Shi+23] Hao-Jun M. Shi, Tsung-Hsien Lee, Shintaro Iwasaki, Jose Gallego-Posada, Zhijing Li, Kaushik Rangadurai, Dheevatsa Mudigere, and Michael Rabbat: **A Distributed Data-Parallel PyTorch Implementation of the Distributed Shampoo Optimizer for Training Neural Networks At-Scale**. In: *arXiv:2309.06497*, (2023) (cit. on pp. ix, 10).

[Shu+22] Changjian Shui, Gezheng Xu, Qi Chen, Jiaqi Li, Charles X Ling, Tal Arbel, Boyu Wang, and Christian Gagné: **On Learning Fairness and Accuracy on Multiple Subgroups**. In: *NeurIPS*. 2022 (cit. on p. 66).

[Sit+20] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein: **Implicit Neural Representations with Periodic Activation Functions**. In: *NeurIPS*. 2020 (cit. on pp. 7, 53–54, 148).

[Sla59] Morton Slater: **Lagrange Multipliers Revisited**. Cowles Foundation Discussion Papers 80. 1959 (cit. on p. 20).

[Soh+24] Motahareh Sohrabi, Juan Ramirez, Tianyue H. Zhang, Simon Lacoste-Julien, and Jose Gallego-Posada: **On PI controllers for updating Lagrange multipliers in constrained optimization**. In: *ICML*. 2024 (cit. on pp. ix, 9, 100–101).

[Sri+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov: **Dropout: A Simple Way to Prevent Neural Networks from Overfitting**. In: *JMLR*. 2014 (cit. on p. 37).

[ST23] National Institute of Standards and Technology: **Artificial Intelligence Risk Management Framework 1.0**. 2023 (cit. on p. 4).

[Sta07] Kenneth O Stanley: **Compositional Pattern Producing Networks: A Novel Abstraction of Development**. In: *Genetic Programming and Evolvable Machines*, **8**:2 (2007), pp. 131–162 (cit. on p. 53).

[SAA20] Adam Stooke, Joshua Achiam, and Pieter Abbeel: **Responsive Safety in Reinforcement Learning by PID Lagrangian Methods**. In: *ICML*. 2020 (cit. on pp. 9, 66, 77, 79–82, 85–86, 99, 153, 177, 182).

[SB18] Richard S. Sutton and Andrew G. Barto: **Reinforcement Learning: An Introduction**. Second. The MIT Press, 2018 (cit. on p. 11).

[Tan+20] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng: **Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains**. In: *NeurIPS*. 2020 (cit. on p. 54).

[TF95] Georg Thimm and Emile Fiesler: **Evaluating Pruning Methods**. In: *ESANN*. 1995 (cit. on p. 44).

[Tou+23] Hugo Touvron et al.: **Llama 2: Open Foundation and Fine-Tuned Chat Models**. In: *arXiv:2307.09288*, (2023) (cit. on p. 63).

[Tra+22] Cuong Tran, Ferdinando Fioretto, Jung-Eun Kim, and Rakshit Naidu: **Pruning has a disparate impact on model accuracy**. In: *NeurIPS*. 2022 (cit. on pp. 7, 63–65, 68, 70–72, 153–154, 159–160).

[UMW17] Karen Ullrich, Edward Meeds, and Max Welling: **Soft Weight-Sharing for Neural Network Compression**. In: *ICLR*. 2017 (cit. on p. 37).

[Uni87] United Nations Environment Programme: **Montreal Protocol on Substances that Deplete the Ozone Layer**. 1987 (cit. on p. 3).

[Uni12] United Nations Environment Programme: **Key achievements of the Montreal Protocol to date**. 2012 (cit. on p. 3).

[Vap91] Vladimir Vapnik: **Principles of Risk Minimization for Learning Theory**. In: *NeurIPS*. 1991 (cit. on p. 12).

[VC74] Vladimir Vapnik and Alexey Chervonenkis: **Theory of Pattern Recognition**. 1974 (cit. on p. 37).

[VR18] Sahil Verma and Julia Rubin: **Fairness Definitions Explained**. In: *International Workshop on Software Fairness*. 2018 (cit. on p. 66).

[Wal92] Gregory K. Wallace: **The JPEG still picture compression standard**. In: *IEEE Transactions on Consumer Electronics*, **38**:1 (1992), pp. xviii–xxxiv (cit. on pp. 57, 146).

[Wan+21] Huan Wang, Can Qin, Yulun Zhang, and Yun Fu: **Neural Pruning via Growing Regularization**. In: *ICLR*. 2021 (cit. on p. 44).

[Zaf+19] Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez-Rodriguez, and Krishna P Gummadi: **Fairness Constraints: A Flexible Approach for Fair Classification**. In: *JMLR*, (2019) (cit. on pp. 79, 91, 191).

[Zaf+17] Muhammad Bilal Zafar, Isabel Valera, Manuel Rodriguez, Krishna Gummadi, and Adrian Weller: **From Parity to Preference-based Notions of Fairness in Classification**. In: *NeurIPS*. 2017 (cit. on p. 66).

[ZK16] Sergey Zagoruyko and Nikos Komodakis: **Wide Residual Networks**. In: *BMVC*. 2016 (cit. on pp. 39, 46, 122, 128).

[Zem+13] Rich Zemel, Yu Wu, Kevin Swersky, Toni Pitassi, and Cynthia Dwork: **Learning Fair Representations**. In: *ICML*. 2013 (cit. on p. 66).

[ZW21] Guodong Zhang and Yuanhao Wang: **On the Suboptimality of Negative Momentum for Minimax Optimization**. In: *AISTATS*. 2021 (cit. on p. 81).

[Zha+22] Guodong Zhang, Yuanhao Wang, Laurent Lessard, and Roger B Grosse: **Near-optimal Local Convergence of Alternating Gradient Descent-Ascent for Minimax Optimization**. In: *AISTATS*. 2022 (cit. on pp. 80, 83, 153).

[ZSQ17] Zhifei Zhang, Yang Song, and Hairong Qi: **Age Progression/Regression by Conditional Adversarial Autoencoder**. In: *CVPR*. 2017 (cit. on pp. 71, 75, 159).

[ZG22] Han Zhao and Geoffrey J Gordon: **Inherent Tradeoffs in Learning Fair Representations**. In: *JMLR*, (2022) (cit. on p. 66).

[Zho+21] Xiao Zhou, Weizhong Zhang, Hang Xu, and Tong Zhang: **Effective Sparsification of Neural Networks With Global Sparsity Constraint**. In: *CVPR*. 2021 (cit. on p. 44).

[Zhu+23] Beier Zhu, Kaihua Tang, Qianru Sun, and Hanwang Zhang: **Generalized Logit Adjustment: Calibrating Fine-tuned Models by Removing Label Bias in Foundation Models**. In: *NeurIPS*. 2023 (cit. on p. 100).

[ZG17] Michael Zhu and Suyog Gupta: **To Prune, or Not to Prune: Exploring the Efficacy of Pruning for Model Compression**. In: *arXiv:1710.01878*, (2017) (cit. on pp. 7, 44, 72, 159, 162).

[Zia+11] Tarek Ziadé et al.: **Flake8**. `https://github.com/PyCQA/flake8`. 2011 (cit. on p. 102).

[Zou60] Guus Zoutendijk: **Methods of Feasible Directions: A Study in Linear and Non-linear Programming**. Elsevier Publishing Company, 1960 (cit. on p. 81).