Algorithms and Architectures for Improving Learning with Optimization Layers

Swaminathan Gurumurthy

CMU-RI-TR-25-90

September 2025

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Zico Kolter, *chair*
Zac Manchester, *chair*
Geoffrey Gordon
Max Simchowitz
Vladlen Koltun, *Apple*

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics.

Abstract

Many real-world problems, from robotic control to resource management, can be effectively formulated as optimization problems. Recent advancements have focused on incorporating these optimization problems as layers within deep learning pipelines, enabling the explicit inclusion of auxiliary constraints or cost functions, which is crucial for applications requiring enforcing physical laws, ensuring safety constraints, and optimizing complex objectives. However, these layers introduce several challenges, including inference inefficiencies, unstable training dynamics, modeling inaccuracies, and representational inefficiencies, which need to be addressed to fully harness their potential.

We systematically investigate these challenges and propose novel numerical methods and architectural solutions that mitigate them, making optimization layers more efficient and effective within deep learning pipelines. Our contributions include methods for enhancing computational efficiency by exploiting the iterative nature of optimization problems, tackling issues of gradient bias and variance in high dimensional problems by exploiting parallelism and network learnt priors about the system, improving sample efficiency in reinforcement learning using approximate simulators, and improving representational problems with using complicated constrained optimization layers by creating a tight feedback loop between the optimizer state and the network outputs. We demonstrate these contributions across different applications, ranging from input-optimization problems, 3D pose estimation and reconstruction, differentiable model predictive control and reinforcement learning problems. We also present a new approach for visual-inertial navigation in nanosatellites, highlighting the practical benefits of integrating optimization layers in challenging real-world scenarios.

Together, these contributions advance our understanding of the complexities and opportunities in integrating optimization layers within deep learning models, offering new frameworks and insights that improve efficiency, stability, and generalizability across a wide range of complex tasks.

Acknowledgments

As this PhD journey comes to a close, I find myself reflecting on the countless individuals who have shaped both this research and my life during these formative years. This thesis is not the product of a single mind, but rather the culmination of endless support, guidance, encouragement, and patience from an incredible community. It is impossible to fully express my gratitude, but I hope these words can serve as a small token of my deep appreciation for everyone who made this work possible. To all of you, thank you. This is what real privilege looks like.

Advisors

It is hard to overstate how fortunate I have been to have Prof. Zac Manchester and Prof. Zico Kolter as my advisors. People often say their advisors gave them freedom and support, but I was given an almost comical amount of both. They showed an incredible degree of patience as I pivoted from one wild idea to the next, offering guidance even when I was essentially just fumbling in the dark. Granted, much of this guidance came in the form of spirited arguments and lengthy debates, but it was through those discussions that I learned the most.

To Zico , thank you for teaching me the meaning of intellectual humility. You always engaged with ideas, no matter how half-baked, with genuine respect and curiosity. You taught me how to have the humility to question opinions and beliefs given new data, thing afresh about problems from the ground up and build new beliefs and opinions in the process. I hope to carry that same spirit of humble inquiry with me throughout my career.

To Zac, thank you for being a constant source of inspiration. In a field often swayed by external pressures and fleeting trends, you taught me how to stay relentlessly curious. At every point where I felt lost or distracted, you had an uncanny ability to remind me of the fundamental joy of discovery and push me towards trying to truely understand the underlying mechanisms involved instead of brute forcing a solution.

I genuinely don't know how to thank you both enough for having confidence in me, especially during the times I had none in myself.

Thesis Committee

I am immensely grateful to my committee members, Prof. Geoff Gordan, Dr. Vladlen Koltun and Prof. Max Simchowitz. When they agreed to join my committee, I had discussions with each of them. I didn't appreciate it at the time, but each of those conversations profoundly shaped the final year and a half of my research.

Thank you, Geoff, for pushing me to think deeply about how to define my identity as a researcher and helping me think deeply about various claims I make in the thesis from the variance issues we discussed in Chapter 4 to the convergence claims in the joint optimization problems. To Vladlen, thanks for helping me realize the unique freedom I get as a last year PhD student and making sure I make use of that freedom to the fullest extent possible. To Max, I appreciate you challenging my each of my research ideas and pushing me to tackle more ambitious directions in the final year of my PhD. Your collective wisdom was invaluable.

Collaborators

A PhD can be a lonely process, especially when you're working on projects that might seem arbitrary to the outside world. I was lucky enough to never truly feel that isolation, thanks to a group of wonderful collaborators who not only tolerated my randomness but embraced it. Thank you to Shaojie, Khai, Karnik, Bingqing, John, Arun, Gengshang, Kyle, Paulo, Shuo, Taylor, Kevin and Deva for being such amazing research partners.

To Shaojie, our work was the first project that got me started in this journey and I couldn't thank you enough for hand holding me throughout the project; I learned so much from you throughout. Khai, I will never forget our late-night brainstorming and pair coding sessions and my first actual hardware experiments of my PhD, that were equal parts productive and delirious. To Karnik and Bingqing, thanks a lot for helping me through a project in a domain I had no clue about and guiding me through the long detours I often took. Thank you all for the shared publications, the mutual support on deadlines, and for making research feel like a team sport.

The Lab and The Institute

My lab mates have been my brainstorming buddies, my debugging crew, my tour guides, and more importantly some of my best friends. Both

RexLab and LocusLab are special places where relationships are valued as much as research projects. Thank you to John, JJ, Khai, Jon, Juan, Sofia, Sam, Fausto, Mitch, Jacob, Arun, Ashley, Ibrahima, Kevin, Shuo, Brian, Aaron, Paulo, Chase, Pedro, Taylor, Simon, Anoushka, Guisy, Ben, Zixin, Suvansh, Chiyen, Benj and Alex from the RexLab and Ashwini, Yash, Sachin, Pratyush, Eric, Shaojie, Marc, Avi, Anna, Yiding, Sam, Priya, Vaishnav, Asher, Chun Kai, Mel, Leslie, Josh, Zhili, Christina, Zhengyang, Jeremy, Kelly, Kevin, Dylan, Andy, Mingjie, Runtian, Eungyeup and Victor from the LocusLab for making the labs feel like a second home. I will cherish our yapping sessions and continue to maintain our yapping leaderboard.

The broader Robotics Institute community has been incredibly supportive. The fact that so many of you showed up to my defense is either a testament to a wonderfully close-knit community or a sign that nobody else was doing work that day. I choose to believe the former.

Friends and Other Productivity Killers

If I had to blame a singular group of people for my lack of productivity, it would be my roommates and friends. I have actively tried to work, but they simply would not let me. Countless evenings were "wasted" playing board games, engaging in completely useless discussions, and debating the most pointless topics imaginable.

To my roommates over the years, Akshay, Gaurav, Yash, Karnik, Sriram, Santoshini, Anurag, Tarana and Saisri, thank you for making our home an escape. More seriously, you have truly been the family I found in Pittsburgh.

And to the rest of the crew, thank you for all the amazing conversations, the arguments, the socials, the trips, and the constant distractions. I could not have done any of what I did without all the wonderful time I spent with you all. To my friends in RI, Ceci, Gokul, Mono, Rishi, Suhail, Jay Patrikar, Shivam, Homanga, Uksang, Raghav, Vidhi, Tejus, Kheim, Mani, Rawal, Bhuvan Jambh, Jay Karhade, Nikhil, Roshan, Abhijat, Sudhu, Sarvesh, Ananya, Arka, Anish, Shagun, Sarthak, Shreya and so many more, and to my friends outside of RI, Pranav, Aravind, Simran, Neharika, Pragna, Deepanjali, Sagnik, Saumya, Maitreyi, Efe, Nari, Kanad, Aakash, Shantanu, Jon Francis and so many more, thank you for making life better and always being there. I would have loved to write the entire thesis just

about all the amazing things I've done with you all, but I've been told I need to stick to the projects I have worked.

Family

To counterbalance the friends determined to tank my productivity, my family was the force trying to keep me on track. They asked for progress reports more frequently than my advisors and were my most dedicated project managers. Every evening, I would get a call from them asking what I had finished that day. Thank you for the three separate reminders that my defense was at 2 PM and for your unwavering belief in me, even when you weren't entirely sure what it was I was doing all day.

To my parents, Rathna and Guru, my brother, Siddhu and my uncle, Sastha, your love and support have been my foundation. Thank you for everything.

Funding

We also thank the Bosch Center for Artificial Intelligence for supporting the work in this thesis throughout my PhD.

Contents

When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.

List of Figures

List of Tables

Chapter 1

Introduction

Incorporating task-specific priors into neural network training is often important for improving representational power, generalization, and interpretability, while providing greater control over the learning process. Optimization layers represent a particular class of such priors, enabling the explicit inclusion of auxiliary constraints or cost functions within the inference process. These layers are beneficial across various tasks, including enforcing physical laws in simulations and ensuring adherence to safety constraints in control systems.

However, the integration of optimization layers into deep networks is not without challenges. Unlike the conventional explicit layers—such as convolutions or activations—that process inputs in a feedforward manner, optimization layers are typically **implicit** and **iterative**. They require solving an optimization problem during inference, which introduces several complications that must be addressed to effectively harness their potential:

- **Inference and Representational Inefficiencies:** Using an explicit network often fails to fully leverage the implicit nature of optimization layers resulting in several representational inefficiencies. This inefficiency can manifest as suboptimal performance and increased computational requirements. For example, using a fixed parameter estimate throughout the optimization run is somewhat

suboptimal. Instead, we'd ideally like to adapt the parameter estimates to the optimizer's current state. This becomes crucial especially in termporal problems, such as robotic control or state estimation, where it's desirable for parameter estimates across time steps to be consistent with the optimizer state across those time-steps in order to obtain smooth predictions and computationally benefit from warm-starting. Moreover, when predicting constraint parameters in complicated optimization problems, it's often difficult to predict parameters that ensure constraint feasibility in one shot [**?**]. This can often create problems during training as well as inference.

- **Training dynamics challenges:** Optimization problems embedded as layers often result in non-convexities, discontinuities, or high gradient variance in the outer training problem. These characteristics can result in convergence to a local minima, impede stable training, or result in prohibitively slow convergence, ultimately limiting the effectiveness of end-to-end learning approaches. Addressing these issues will require specialized techniques for stabilizing training dynamics, such as advanced regularization methods, improved initialization strategies, or specialized optimization algorithms. Moreover, the interplay between the non-convex optimization landscape and the neural network parameters can lead to unpredictable behaviors during training, necessitating sophisticated techniques to understand and mitigate these issues. Strategies like using surrogate loss functions, adaptive learning rates, or even hybrid optimization schemes can be crucial for maintaining stability and ensuring consistent training progress.
- **Modeling Inaccuracies:** Simplifications or assumptions made in formulating the optimization problem can introduce **biases** into the learning and inference process. These inaccuracies, often stemming from discrepancies between the real-world and simplified models, can restrict the model's ability to achieve optimal performance and generalize effectively to unseen data. Such biases are particularly problematic in safety-critical applications, where performance guarantees are essential, and inaccuracies could lead to significant failures. Developing methods to understand and account for these modeling inaccuracies can be vital for ensuring that the integration of optimization layers results in reliable and high-performing models.

Addressing these challenges is crucial for unlocking the full potential of optimization layers in deep learning. It requires a careful examination of both the **numerical methods** used and the **architectural designs** of the neural networks. In this thesis, we systematically analyze these challenges across various scenarios, propose novel solutions to mitigate them, and demonstrate how these solutions can be applied to achieve improved performance, efficiency, and robustness in deep learning models. Our exploration covers a variety of domains, emphasizing the practical benefits of effectively integrating optimization layers to tackle complex, real-world problems.

1.1 Key Contributions

In this thesis, we present several key contributions that address the challenges associated with integrating optimization layers into deep learning models. Each chapter corresponds to a major research project, representing a distinct aspect of the contributions made:

Joint Inference and Input Optimization in Deep Equilibrium Networks: In **Chapter 3**, we consider input optimization problems in deep learning which involve optimizing over the *inputs* to a network to minimize or maximize some objective; examples include optimization over latent spaces in a generative model to match a target image, or adversarially perturbing an input to worsen classifier performance. Performing such optimization, however, is traditionally quite costly, as it involves a complete forward and backward pass through the network for each gradient step. However, we observe that typical input optimization methods do not leverage a key property of the problem, i.e, the fact that the network inference across successive input optimization iterations involve very similar computations. This results in **computational inefficiencies** in the inference and gradient computations. We address this inefficiency by formulating the optimization and the DEQ network inference problem as a unified fixed-point iteration. This allows us to 'reuse' the inference computation across iterations and achieve substantial speedups compared to traditional iterative methods.

Variance Reduction in Bundle Adjustment Layers for Visual Odometry:

In **Chapter 4**, we use differentiable bundle adjustment layers in visual odometry tasks as an example optimization layer to analyze the sources of gradient variance in optimization layers. We identify three sources of **gradient variance** in bundle adjustment layers and propose modifications to reduce this variance, leading to faster and more stable training. Experiments show a 2x to 2.5x speedup in training times and improved generalization across different training setups.

Value-Gradient Updates for Off-Policy Deep Iterative Learning Control: In **Chapter 5**, we use the reinforcement learning problem as an example to illustrate how we can leverage approximate models within end-to-end learning pipelines to achieve data-efficient learning while avoiding potential biases introduced by the modeling approximations. Specifically, we propose the Value Gradient Update taking inspiration from Iterative Learning Control to use approximate model jacobians while computing the gradients but using the real data for policy and value function evaluation. This method significantly enhances sample efficiency and improves zero-shot transfer performance even when just using the value gradient update in the simulator alone.

Leveraging GPU parallelism with Critic Gradient Actor-Critic Methods for scaling up On-Policy RL: Chapter 6 addresses the challenge of high variance in reinforcement learning updates, which can stem from the high dimensionality or dynamic nature of the underlying model. We present a Critic Gradient-based Actor-Critic (CGAC) method specifically designed to leverage the parallelism of modern GPU simulators. By utilizing critic gradients directly, our approach reduces variance in the policy updates, leading to more reliable and consistent learning while using the higher sampling rates to stabilize critic training. This approach improves training efficiency and performance in high-dimensional action space environments, achieving faster convergence and higher returns compared to state-of-the-art RL methods like PPO and SAC.

Visual Inertial Navigation for Satellites: In **Chapter 7**, we introduce VINSat, an end-to-end visual-inertial navigation system for orbit determination (OD) of nanosatellites. VINSat addresses the 'lost-in-space' problem by combining data from a low-cost RGB camera and an inertial measurement unit (IMU) to identify known

landmarks and determine the satellite's full state. Using a SOTA deep network based detection algorithm for landmark detection and a batch nonlinear least-squares state estimator, VINSat determines kilometer-level accurate satellite localization within a few hours, significantly outperforming traditional ground-based methods. We validate VINSat through simulations using real nadir-pointing imagery, showing that 85% of the simulated satellites are localized to within 5 km within 6 hours, demonstrating faster and more precise localization compared to ground radar.

Deep Equilibrium Model Predictive Control (DEQ-MPC): In **Chapter 8**, we consider the use of differentiable model predictive control problems (MPC) as a layer within a network and examine the representational and training stability issues observed with these layers. We observe that these issues arise from treating the optimization layer and the network as separate, independent entities, resulting in sub-optimal integration. We propose to co-develop the solver and architecture by formulating the forward pass as a *joint fixed-point problem* over the coupled network outputs and necessary conditions of the optimization problem. We solve this problem in an iterative manner where we alternate between network forward passes and optimization iterations. Through extensive ablations in various robotic control tasks, we demonstrate that our approach yields richer representations and more stable training, while naturally accommodating warm starts, a key requirement for MPC.

Together, these contributions advance the integration of optimization layers within deep learning, offering new frameworks, methods, and architectural insights that improve efficiency, stability, and the ability to generalize across complex tasks.

Chapter 2

Background

This chapter provides an in-depth overview of optimization layers and equilibrium layers, a class of implicit layers that are represented as fixed point operations. Most of the work in this thesis utilize these in some capacity.

2.1 Optimization Layers

Optimization layers represent a category of neural network layers that embed optimization problems directly into the inference problem. These layers can be either differentiable or non-differentiable (often referred to as black-box optimization layers). This allows the designer to incorporate cost functions and constraints as part of the inference problem itself. Optimization layers are utilized across a wide range of applications. In our work alone, we look at its applications in simulation , state estimation/Visual Odometry/SLAM , inverse problems and trajectory optimization.

Mathematical Definition: Optimization layers are typically framed as constrained optimization problems as follows:

$$x^*(z) := \underset{x}{\text{argmin}}\ f(x, z) \tag{2.1}$$

$$\text{subject to} \quad g(x, z) \leq 0, \tag{2.2}$$

$$h(x, z) = 0. \tag{2.3}$$

where

- $x \in \mathbb{R}^n$ is the optimization variable representing the parameters or decision variables in the layer,
- $z \in \mathbb{R}^m$ is the input to the optimization layer, often representing parameters or contextual data that affect the optimization,
- $f(x, z) : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ is the objective function, which defines the cost or loss to minimize,
- $g(x, z) \leq 0$ represents inequality constraints, where $g : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^p$,
- $h(x, z) = 0$ represents equality constraints, where $h : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^q$.

Further, in certain cases, this optimization problem is differentiable. We can then recover the gradients, $\frac{\partial x^*(z)}{\partial z}$, using a couple of different methods discussed below. Differentiable optimization layers facilitate end-to-end training by allowing regular backpropagation and fit naturally into deep learning pipelines.

2.1.1 Implicit differentiation

One such method [Amos and Kolter, 2017], relies on sensitivity analysis of the optimization problem being differentiated. Following the derivation in [Barratt, 2018, Kolter et al., 2020b] with minor modifications, the approach can be summarized with the following. First, we defined the following vector-valued function from the Karush-Kuhn-Tucker (KKT) conditions:

$$G(x, \lambda, \nu, z) = \begin{bmatrix} \nabla_x f(x, z) + \partial_x g(x, z)^T \lambda + \partial_x h(x, z)^T \nu \\ \lambda \circ g(x) \\ h(x) \end{bmatrix}. \tag{2.4}$$

Then, sensitivity analysis effectively leverages the fact that for any primal-dual optimal solution x^*, λ^* and ν^* (where the former is known as the primal variables and the later two the dual variables), the first order KKT conditions $G(x^*, \lambda^*, \nu^*, z) = 0$ for any choice of the parameter z (as long as the resulting problem remains convex of course). This implies the following condition:

$$D_z G(x^*, \lambda^*, \nu^*, z) = 0, \tag{2.5}$$

where D_z denotes the total derivative with respect to z. Now, by leveraging the implicit function theorem and a simple application of the chain rule, we can compute the partial derivative of interest:

$$\begin{aligned} &\partial_{x^*, \lambda^*, \nu^*} G(x^*, \lambda^*, \nu^*, z) \partial_z (x^*, \lambda^*, \nu^*) + \partial_z G(x^*, \lambda^*, \nu^*, z) = 0 \\ &\partial_z (x^*, \lambda^*, \nu^*) = -\partial_{x^*, \lambda^*, \nu^*} G(x^*, \lambda^*, \nu^*, z)^{-1} \partial_z G(x^*, \lambda^*, \nu^*, z). \end{aligned} \tag{2.6}$$

This gradient is exact when the optimization problem is convex. The resulting gradient computation can be easily added to any automatic differentiation library and used with regular backpropagation thereby naturally fitting into regular autodiff pipelines. However, when the problem is non-convex, the optimization problem does not have a unique fixed point. As a result, the gradient computed with IFT is non-unique and potentially inaccurate.

2.1.2 Rolling out solvers

Another approach to recovering the gradient of the optimization problem is to simply differentiate the actual optimization algorithm used to compute the (approximate) solution of the mathematical program. This is usually done using automatic differentiation. For example, given an initial guess x_{init} to the problem 2.3, many solution methods compute an optimal or suboptimal solution by relying on an iterative scheme of the form:

$$x_k \leftarrow x_{k-1} - \eta(x_{k-1}, f, g, h, z). \tag{2.7}$$

The key idea is that depending on the nature of the update step η, automatic differentiation can be used to recover the partials $\partial_{x_{k-1}} x_k$ and $\partial_z x_k$. Given those gradients for a single step, it is straightforward to recover the gradients for arbitrarily large number of update steps by applying the chain rule. This is especially useful when the optimization problem is non-convex as the gradient computed depends on the entire trajectory instead of the fixed point alone thereby giving a unique gradient.

2.1.3 Applications

2.2 Deep Equilibrium Layers

Deep equilibrium (DEQ) layers [Bai et al., 2019a] are a type of implicit layer that achieve their forward pass by finding a fixed-point solution rather than executing a pre-defined series of transformations. This approach allows the network to adaptively determine the computational depth required for a given input, leading to a more efficient and expressive model.

2.2.1 Forward Pass

Specifically, given an input $x \in \mathcal{X}$, computing the forward pass in a DEQ model involves finding a fixed point $z \in \mathcal{Z}$, such that

$$z^\star = d_\phi(z^\star, x), \tag{2.4}$$

where, $d_\phi : \mathcal{Z} \times \mathcal{X} \to \mathcal{Z}$ is some parameterized layer conditioned on input x, $\mathcal{Z}$ denotes the hidden state or outputs of the network which we are computing the fixed point on, $\mathcal{X}$ denotes the input space, and ϕ denotes the parameters of the layer. Computing this fixed point (under proper stability conditions) corresponds to the "infinite depth" limit of repeatedly applying the function $z^+ := d_\phi(z, x)$ starting at some arbitrary initial value of z (typically 0). The fixed point solution is considered the output of the layer. However, in practice DEQ models will typically compute this fixed point not simply by iterating the function d_ϕ, but by using a more accelerated root-finding or fixed-point approach such as Broyden's method [Broyden, 1965] or Anderson acceleration [Anderson, 1965, Walker and Ni, 2011]. Further, although little can be said about e.g., the existence or uniqueness of these fixed points *in general* (though there do exist restrictive settings where this is possible [Fung et al., 2021a, Revay et al., 2020, Winston and Kolter, 2020]), in practice a wide suite of techniques have been used to ensure that such fixed points exist, can be found using relatively few function evaluations, and are able to competitively model large-scale tasks [Bai et al., 2019b, 2020].

2.2.2 Backward Pass

Gradients are typically computed via implicit differentiation with respect to the fixed-point equation. This eliminates the need to unroll iterations from the forward pass, significantly reducing memory consumption while maintaining accurate gradient propagation. As described earlier, given some root finding problem $F(z, \theta) = 0$, the gradient of the solution with respect to the parameters θ can be computed as:

$$\frac{\partial z^*}{\partial \theta} = -\left(\frac{\partial F}{\partial z}\right)^{-1} \cdot \frac{\partial F}{\partial \theta}, \tag{2.5}$$

where $\frac{\partial F}{\partial z}$ is the jacobian matrix at the fixed point solution. This can again be

conveniently integrated into most automatic differentiation libraries. However, solving this linear system analytically is often infeasible for large problems. We typically instead use iterative linear system solvers such as GMRES, Broyden etc. to solve this linear system. But this is just as expensive as the forward pass itself. Recent work [Fung et al., 2021b, Geng et al., 2021] has shown that the approximations of the gradient by simply assuming an identity Jacobian or differentiating through the last few iterations of the fixed point iteration using vanilla backpropagation is equally/more effective while being computationally cheaper.

2.2.3 Trade-offs involved in using equilibrium layers

Equilibrium (DEQ) layers offer notable benefits in terms of flexibility and efficiency, but they also present specific challenges that require attention.

Benefits:

Dynamic Depth: DEQ layers introduce dynamic depth, allowing for an "infinite" number of layers without explicitly defining them. This enables computations to adapt to the complexity of each input, adjusting the depth accordingly.

Parameter Efficiency: DEQ models can achieve performance comparable to traditional deep networks while using fewer parameters, enhancing both training and inference efficiency.

Warm-Starting: DEQ layers can leverage solutions from previous time steps to initialize the fixed-point iterations, improving convergence speed, particularly for temporally correlated data (e.g., video sequences).

Challenges:

Training Stability and Sensitivity to Hyperparameters: DEQ models are sensitive to hyperparameter settings, and improper tuning can lead to divergence or very slow convergence during training. Various regularizers have been proposed to improve stability, although these add complexity to the training process.

Computational Overhead: Achieving convergence in the fixed-point iteration for both forward and backward passes can be computationally demanding, particularly in resource-constrained or real-time environments.

Chapter 3

Joint inference and input optimization with equilibrium models

In this chapter, we consider input optimization problems in deep learning which involve optimizing over the *inputs* to a network to minimize or maximize some objective; examples include optimization over latent spaces in a generative model to match a target image, or adversarially perturbing an input to worsen classifier performance. Performing such optimization, however, is traditionally quite costly, as it involves a complete forward and backward pass through the network for each gradient step. However, we observe that typical input optimization methods do not leverage a key property of the problem, i.e, the fact that the network inference across successive input optimization iterations involve very similar computations. This results in **computational inefficiencies** in the inference and gradient computations. We address this inefficiency by formulating the optimization and the DEQ network inference problem as a unified fixed-point iteration. This allows us to 'reuse' the inference computation across iterations and achieve substantial speedups compared to traditional iterative methods.

The contents of this chapter have been previously published at Neurips 2021 in [Gurumurthy

et al., 2021].

3.1 Introduction

Many settings in deep learning involve optimization over the inputs to a network to minimize some desired loss. For example, for a "generator" network $G : \mathcal{Z} \to \mathcal{X}$ that maps from latent space $\mathcal{Z}$ to an observed space $\mathcal{X}$, it may be desirable to find a latent vector $z \in \mathcal{Z}$ that most closely produces some target output $x \in \mathcal{X}$ by solving the optimization problem (e.g. [Bora et al., 2017, Chang et al., 2017])

$$\underset{z \in \mathcal{Z}}{\text{minimize}} \ \|x - G_\theta(z)\|_2^2. \tag{3.1}$$

As another example, constructing adversarial examples for classifiers [Goodfellow et al., 2014b, Madry et al., 2018] typically involves optimizating over a perturbation to a given input; i.e., given a classifier network $g : \mathcal{X} \to \mathcal{Y}$, task loss $\ell : \mathcal{Y} \to \mathbb{R}_+$, and a sample $x \in \mathcal{X}$, we want to solve

$$\underset{\|\delta\| \leq \epsilon}{\text{maximize}} \ \ell(g(x + \delta)). \tag{3.2}$$

More generally, a wide range of inverse problems [Bora et al., 2017] and other auxiliary tasks [Amos et al., 2018b, Finn et al., 2017] in deep learning can also be formulated in such a manner.

Orthogonal to this line of work, a recent trend has focused on the use of an *implicit layer* within deep networks to avoid traditional depth. For instance, Bai et al. [2019b] introduced deep equilibrium models (DEQs) which instead treat the network as repeated applications of a single layer and compute the output of the network as a solution to an equilibrium-finding problem instead of simply specifying a sequence of non-linear layer operations. Bai et al. [2019b] and subsequent work [Bai et al., 2020] have shown that DEQs can achieve results competitive with traditional deep networks for many realistic tasks.

In this work, we highlight the benefit of using these implicit models in the context

of input optimization routines. Specifically, because optimization over inputs itself is typically done via an iterative method (e.g., gradient descent), we can combine this optimization fixed-point iteration *with* the forward DEQ fixed point iteration all within a single "augmented" DEQ model that *simultaneously* performs forward model inference as well as optimization over the inputs. This enables the models to more quickly perform both the inference and optimization procedures, and the resulting speedups further allow us to *train* networks that use such "bi-level" fixed point passes. In addition, we also show a close connection between our proposed approach and the primal-dual methods for constrained optimization.

We illustrate our methods on 4 tasks that span across different domains and problems: 1) training DEQ-based generative models while optimizing over latent codes; 2) training models for inverse problems such as denoising and inpainting; 3) adversarial training of implicit models; and 4) gradient-based meta-learning. We show that in all cases, performing this simultaneous optimization and forward inference accelerates the process over a more naive inner/outer optimization approach. For instance, using the combined approach leads to a 3.5-9x speedup for generative DEQ networks, a 3x speedup in adverarial training of DEQ networks and a 2.5-3x speedup for gradient based meta-learning. In total, we believe this work points to a variety of new potential applications for optimization with implicit models.

3.2 Connections to related work

Implicit layers. Layers with implicitly defined depth have gained tremendous popularity in recent years[El Ghaoui et al., 2019, Gould et al., 2019, Kolter et al., 2020a]. Rather than a static computation graph, these layers define a condition on the output that the model must satisfy, which can represent "infinite" depth, be directly differentiated through via the implicit function theorem [Krantz and Parks, 2012], and are memory-efficient to train. Some recent examples of implicit layers include optimization layers [Amos and Kolter, 2017, Djolonga and Krause, 2017], deep equilibrium models[Bai et al., 2019b, 2020, Kawaguchi, 2021, Lu et al., 2021, Winston and Kolter, 2020], neural ordinary differential equations (ODEs) [Chen et al.,

2018, Dupont et al., 2019, Rubanova et al., 2019], logical structure learning [Wang et al., 2019], and continuous generative models [Grathwohl et al., 2019].

In particular, deep equilibrium models (DEQs) [Bai et al., 2019b] define the output of the model as the fixed point of repeated applications of a layer. They compute this using black-box root-finding methods[Bai et al., 2019b] or accelerated fixed-point iterations [Jeon et al., 2021] (e.g., Broyden's method [Broyden, 1965]). In this work, we propose an efficient approach to perform input optimization with the DEQ by *simultaneously* optimizing over the inputs and solving the forward fixed point of an equilibrium model as a joint, augmented system. As related work, Jeon et al. [2021] introduce fixed point iteration networks that generalize DEQs to repeated applications of gradient descent over variables. However, they don't address the specific formulation presented in this paper, which has a number of practical use cases (e.g., adversarial training). Lu et al. [2021] proposes an implicit version of normalizing flows by formulating a joint root-finding problem that defines an invertible function between the input x and output $z^\star$. Perhaps the most relevant approach to our work is Gilton et al. [2021], which specifically formulates inverse imaging problems as a DEQ model. In contrast, our approach focuses on solving input optimization problems where the network of interest is *already* a DEQ, and thus the combined optimization and forward inference task leads to a substantially different set of update equations and tradeoffs.

Input optimization in deep learning. Many problems in deep learning can be framed as optimizing over the inputs to minimize some objective . Some canonical examples of this include finding adversarial examples [Kolter and Wong, 2018, Madry et al., 2018], solving inverse problems [Bora et al., 2017, Chang et al., 2017, Ongie et al., 2020], learning generative models [Bojanowski et al., 2017, Zadeh et al., 2019], meta-learning [Finn et al., 2017, Gurumurthy et al., 2020, Rajeswaran et al., 2019, Zintgraf et al., 2019] etc. For most of these examples, input optimization is typically done using gradient descent on the input, i.e., we feed the input through the network and compute some loss, which we minimize by optimizing over the input with gradient descent. While some of these problems might not require differentiating through the entire optimization process, many do (introduced below), and can further slow down

training and impose massive memory requirements.

Input optimization has recently been applied to train generative models. Bojanowski et al. [2017], Zadeh et al. [2019] proposed to train generator networks by jointly optimizing the parameters and the latent variables corresponding to each example. Similarly, optimizing a latent variable to make the corresponding output match a target image is common in decoder-only models like GANs to get correspondences [Bora et al., 2017, Karras et al., 2020], and has been found useful to stabilize GAN training [Wu et al., 2020]. However, in all of these cases, the input is optimized for just a few (mostly 1) iterations. In this work, we present a generative model, where we optimize and find the *optimal* latent code for each image at each training step. Additionally, Bora et al. [2017], Chang et al. [2017] showed that we can take a pretrained generative model and use it as a prior to solve for the likely solutions to inverse problems by optimizing on the input space of the generative model (i.e., unsupervised inverse problem solving). Furthermore, Diamond et al. [2017], Gilton et al. [2020], Gregor and LeCun [2010] have shown that networks can also be trained to solve specific inverse problems by effectively unrolling the optimization procedure and iteratively updating the input. We demonstrate our approach in the unsupervised setting as in Bora et al. [2017], Chang et al. [2017], but also show flexible extension of our framework to train implicit models for supervised inverse problem solving.

Another crucial application of input optimization is to find adversarial examples [Goodfellow et al., 2014b, Szegedy et al., 2014]. This manifests as optimizing an objective that incentivices an incorrect prediction by the classifier, while constraining the input to be within a bounded region of the original input. Many attempts have been made on the defense side [Kannan et al., 2018, Papernot et al., 2015, Tao et al., 2018, Wong et al., 2020]. The most successful strategy thus far has been adversarial training with a projected gradient descent (PGD) adversary [Madry et al., 2018] which involves training the network on the adversarial examples computed using PGD *online during training*. We show that our joint optimization approach can be easily applied to this setting, allowing us to train implicit models to perform competitively with PGD in guaranteeing adversarial robustness, but at much faster speeds.

While the examples above were illustrated with non-convex networks, attempts

have also been made to design networks whose output is a convex function of the input [Amos et al., 2017]. This allows one to use more sophisticated optimization algorithms, but usually at a heavy cost of model capacity. They have been demonstrated to work in a variety of problems including multi-label prediction, image completion [Amos et al., 2017], learning stable dynamical systems [Kolter and Manek, 2019] and optimal transport mappings [Makkuva et al., 2020], MPC [Bünning et al., 2020], etc.

3.3 Joint inference and input optimization in DEQs

Here we present our main methodological contribution, which sets up an augmented DEQ that jointly performs inference and input optimization over an existing DEQ model. We first define the base DEQ model, and then illustrate a joint approach that simultaneously finds it's forward fixed point and optimizes over its inputs. We discuss several methodological details and extensions.

3.3.1 Preliminaries: DEQ-based models

To begin with, we recall the deep equilibrium model setting from Bai et al. [2019b], but with the notation slightly adapted to better align with its usage in this paper. Specifically, we consider an *input-injected* layer $f_\theta : \mathcal{Z} \times \mathcal{X} \to \mathcal{Z}$ where $\mathcal{Z}$ denotes the hidden state of the network, $\mathcal{X}$ denotes the input space, and θ denotes the parameters of the layer. Given an input $x \in \mathcal{X}$, computing the forward pass in a DEQ model involves finding a fixed point $z^\star(x) \in \mathcal{Z}$, such that

$$z_\theta^\star(x) = f_\theta(z_\theta^\star(x), x), \tag{3.3}$$

which (under proper stability conditions) corresponds to the "infinite depth" limit of repeatedly applying the f_θ function. We emphasize that under this setting, we can effectively think of $z_\theta^\star$ *itself* as the implicitly defined network (which thus is also parameterized by θ), and one can differentiate through this "network" via the implicit

function theorem [Binmore and Davies, 2001, Krantz and Parks, 2012] as described in Chaper 2.

3.3.2 Joint inference and input optimization

Now we consider the setting of performing *input optimization* for such a DEQ model. Specifically, consider the task of attempting to optimize the input $x \in \mathcal{X}$ to minimize some loss $\ell : \mathcal{Z} \times \mathcal{Y} \to \mathbb{R}_+$.

$$\underset{x \in \mathcal{X}}{\text{minimize}} \ \ell(z_\theta^\star(x), y) \tag{3.4}$$

where $y \in \mathcal{Y}$ represents the data point. To solve this, we typically perform such an optimization via e.g., gradient descent, which repeats the update

$$x^+ := x - \alpha \left(\frac{\partial \ell(z_\theta^\star(x), y)}{\partial x} \right)^\top \tag{3.5}$$

until convergence, where we use term $z^\star$ alone to denote the fixed output of the network $z_\theta^\star$ (i.e., just as a fixed output rather than a function). Using the chain rule and the implicit function theorem, we can further expand update (3.5) using the following analytical expression of the gradient:

$$\frac{\partial \ell(z_\theta^\star(x), y)}{\partial x} = \frac{\partial \ell(z^\star, y)}{\partial z^\star} \frac{\partial z_\theta^\star(x)}{\partial x} = \frac{\partial \ell(z^\star, y)}{\partial z^\star} \left(I - \frac{\partial f_\theta(z^\star, x)}{z^\star} \right)^{-\top} \frac{\partial f_\theta(z^\star, x)}{\partial x} \tag{3.6}$$

Thinking about $z_\theta^\star$ as an implicit function of x permits us to combine the DEQ fixed-point equation (on z) with this input optimization update (on x), thus performing a joint forward update:

$$\begin{bmatrix} z^+ \\ x^+ \end{bmatrix} := \begin{bmatrix} f_\theta(z, x) \\ x - \alpha \left(\frac{\partial f_\theta(z,x)}{\partial x} \right)^\top \left(I - \frac{\partial f_\theta(z,x)}{\partial z} \right)^{-\top} \left(\frac{\partial \ell(z,y)}{\partial z} \right)^\top \end{bmatrix} \tag{3.7}$$

It should be apparent that, if both iterates converge, then they have converged to a simultaneous fixed point $z^\star$ and an optimal $x^\star$ value for the optimization problem (3.4). However, simply performing this update can still be inefficient, because computing the inverse Jacobian in Eq. (3.6) is expensive and typically computed via an iterative

update – namely, we would first compute the variable $\mu = \left(I - \frac{\partial f_\theta(z,x)}{\partial z}\right)^{-\top}\left(\frac{\partial \ell(z,y)}{\partial z}\right)^\top$ via the following iteration (i.e., a Richardson iteration [Richardson, 1911]):

$$\mu^+ := \left(\frac{\partial f_\theta(z,x)}{\partial z}\right)^\top \mu + \left(\frac{\partial \ell(z,y)}{\partial z}\right)^\top. \tag{3.8}$$

Therefore, to efficiently solve the joint inference and input optimization problem, we propose combining *all three* iterative procedures into the update

$$\begin{bmatrix} z^+ \\ \mu^+ \\ x^+ \end{bmatrix} := \begin{bmatrix} f(z,x) \\ \left(\frac{\partial f_\theta(z,x)}{\partial z}\right)^\top \mu + \left(\frac{\partial \ell(z,y)}{\partial z}\right)^\top \\ x - \alpha\left(\frac{\partial f_\theta(z,x)}{\partial x}\right)^\top \mu \end{bmatrix} \tag{3.9}$$

Like Eq. (3.7), if this joint process converges to a fixed point, then it corresponds to a simultaneous optimum of both the inference and optimization processes. Such a formulation is especially appealing, as the iteration (3.9) is *itself* just an *augmented DEQ network* $v_\theta^\star(y)$ (i.e., with input injection y) whose forward pass optimizes on a joint inference-optimization space $v = (x, \mu, z)$. Moreover, we can use standard techniques to differentiate through *this* process, though there are also optimizations we can apply in several settings that we discuss below. This is in contrast to prior works where f_θ is an explicit deep neural network, where the model forward-pass and optimization processes are disentangled and have to be dealt with separately. We illustrated this in Figure 3.1 where the figure on the left shows the input optimization naively performed using gradient descent in DEQs v/s the figure on the right which shows the joint updates performed using the augmented DEQ network.

As a final note, we mention that, in practice, just as the gradient-descent update has a step size α, it is often beneficial to add similar "damping" step sizes to the other updates as well. This leads to the full iteration over the augmented DEQ

$$\begin{bmatrix} z^+ \\ \mu^+ \\ x^+ \end{bmatrix} := \begin{bmatrix} (1-\alpha_z)z + \alpha_z f(z,x) \\ (1-\alpha_\mu)\mu + \alpha_\mu\left(\left(\frac{\partial f_\theta(z,x)}{\partial z}\right)^\top \mu + \left(\frac{\partial \ell(z,y)}{\partial z}\right)^\top\right) \\ x - \alpha_x\left(\frac{\partial f_\theta(z,x)}{\partial x}\right)^\top \mu \end{bmatrix} \tag{3.10}$$

Finally, in order to speed up convergence, as is common in DEQ models, we apply a more involved fixed point solver, such as Anderson acceleration, on top of this naive iteration. We analyze the effect of these different root-finding approaches in subsection 3.4.5.

Notes on Convergence Our treatment of the above system as an augmented DEQ allows us to borrow results from Winston and Kolter [2020]Bai et al. [2021] to ensure convergence of the fixed point iteration. Specifically, if we assume the joint Jacobian of the fixed point iterations we describe are strongly monotone with smoothness parameter m and Lipschitz constant L, then by standard arguments (see e.g., Section 5.1 of [Ryu and Boyd, 2016]), the fixed point iteration with step size $\alpha < m/L^2$ will converge. Note that these are substantially weaker rates and constants than required for typical gradient descent or the minimization of locally convex function because the coupling between the three fixed point iterations introduce cross-terms in the joint Jacobian.

Due to these cross-terms, going from the strong monotonicity assumption on the joint fixed point iterations to specific assumptions on f_θ and ℓ is less straightforward. However, empirically, we observed that as long as the step sizes α's were kept reasonably small and the functions f_θ and ℓ were designed appropriately (e.g ℓ respecting notions of local convexity and f_θ with Jacobian eigenvalues less than 1, etc.) the fixed point iterations converged reliably.

3.3.3 Iterpretation as a primal-dual optimization

While the problem above was introduced as an input-optimization problem, its formulation as a joint optimization problem in the augmented DEQ system (3.9) can also be viewed as a constrained optimization problem where the DEQ fixed-point conditions are treated as constraints,

$$\underset{x,z}{\text{minimize}} \ \ell(z, y) \ \text{ subject to } \ z = f_\theta(z, x) \tag{3.11}$$

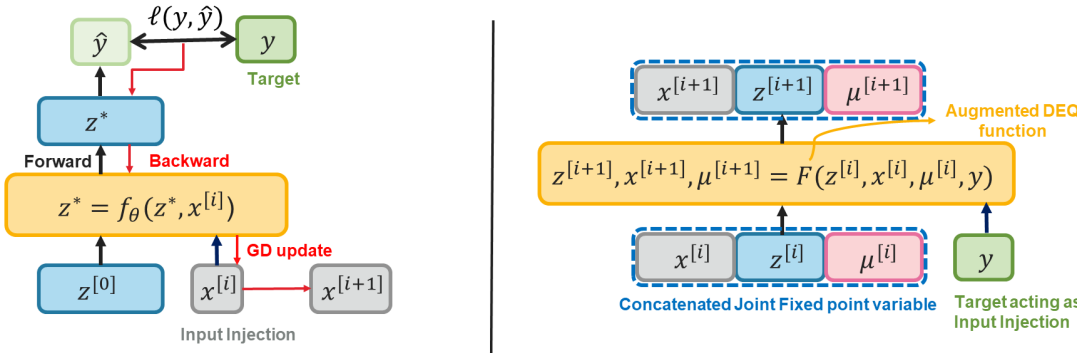


Figure 3.1: Left: Performing each gradient update on DEQ inputs requires a fixed point computation in the forward and backward pass. Right: Solving the 3 fixed points simultaneously as an "augmented" DEQ where the targets y act as input and the function F represents the joint fixed point updates in Eq. 3.10

.

which yield the Lagrangian

$$\underset{x,z}{\text{minimize}}\ \underset{\mu}{\text{maximize}}\ \ \mathcal{L}(x, z, \mu) \equiv \ell(z, y) + \mu^\top (f_\theta(z, x) - z) \tag{3.12}$$

and the corresponding KKT conditions

$$\begin{bmatrix} f_\theta(z,x) - z \\ \frac{\partial \mathcal{L}(x,z,\mu)}{\partial z} \\ \frac{\partial \mathcal{L}(x,z,\mu)}{\partial x} \end{bmatrix} = \begin{bmatrix} f_\theta(z,x) - z \\ \frac{\partial \ell(z,y)}{\partial z} + \mu^\top (\frac{\partial f_\theta(z,x)}{\partial z} - I) \\ \mu^\top \frac{\partial f_\theta(z,x)}{\partial x} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \in \mathbb{R}^{2n+d} \tag{3.13}$$

where μ are the dual variables corresponding to the equality constraints. Rearranging the terms in the KKT conditions of the above problem, introducing the step size parameters $\alpha's$ and treating it as fixed point iteration gives us the updates in Eq. (3.10). Indeed, performing such iterations is a variation of the classical primal-dual gradient method for solving equality-constrained optimization problems [Du and Hu, 2019, Elman and Golub, 1994, Hong et al., 2018].

3.3.4 Outer Optimization (Backward Pass)

A notable advantage of formulating the entire joint inference and input optimization problem as an augmented DEQ $v_\theta^\star(y)$ is that it allows us to abstract away the detailed function of v, and simply train parameters θ of this joint process as an outer optimization problem:

$$\underset{\theta}{\text{minimize}} \ \ell^{\text{outer}}(v_\theta^\star(y), y) \tag{3.14}$$

where $\ell^{\text{outer}} : \mathcal{X} \times \mathcal{Z} \times \mathcal{Z} \times \mathcal{Y} \to \mathbb{R}_+$

Given the solutions $x^\star, z^\star$, to the inner problem, computing updates to θ in the outer optimization problem is equivalent to the backward pass of the augmented DEQ and correspondingly is optimized using standard stochastic gradient optimizers like Adam [Kingma and Ba, 2015b]. Thus, as with any other DEQ model, we assume the inner problem was solved to a fixed point, and apply the implicit function theorem to compute the gradients w.r.t the augmented system (3.10). This gives us a constant memory backward pass which is invariant to the underlying optimizer used to solve the inner problem.

$$\frac{\partial \ell^{\text{outer}}(v_\theta^\star, y)}{\partial \theta} = \frac{\partial \ell^{\text{outer}}(y, v^\star)}{\partial v^\star}\frac{\partial v_\theta^\star}{\partial \theta} = -\frac{\partial \ell^{\text{outer}}(y, v^\star)}{\partial v^\star}\left(\frac{\partial K_\theta(v^\star)}{\partial v^\star}\right)^{-1}\frac{\partial K_\theta(v^\star)}{\partial \theta} \tag{3.15}$$

where $v = [x, z, \mu]^\top$ and $K_\theta(v) = 0$ represents the KKT conditions from (3.13). As with the original DEQ, instead of computing the above expression explicitly, we first solve the following linear system to compute u and then substitute it back in the equation above to obtain the full gradient,

$$u^\top = -\frac{\partial \ell^{\text{outer}}(y, v^\star)}{\partial v^\star}\left(\frac{\partial K_\theta(v)}{\partial v}\right)^{-1} \quad \Longleftrightarrow \quad \frac{\partial \ell^{\text{outer}}(y, v^\star)}{\partial v^\star}^\top + \left(\frac{\partial K_\theta(v)}{\partial v}\right)^\top u = 0 \tag{3.16}$$

Although we can train any joint DEQ in this manner, doing so in practice (e.g., via automatic differentiation), will require double backpropagation, because the definition of $K_\theta(v)$ above already includes vector-Jacobian products, and this expression will require differentiating again. However, in the case that ℓ^{outer} is the *same* as ℓ (or

in fact where it is the negation of ℓ), then there exists a substantial simplification of the outer optimization gradient. These cases are indeed quite common, as we may want e.g., to train the parameters of a generative model to minimize the same reconstruction error that we attempt to optimize via the latent variable; or in the case of adversarial examples, the inner adversarial optimization is precisely the negation of the outer objective.

In these cases, we have that

$$\ell^{\text{outer}}(y, v_\theta^\star(y)) = \ell(y, z_\theta^\star(y)) \tag{3.17}$$

so we have that

$$\frac{\partial \ell^{\text{outer}}(y, v_\theta^\star(y))}{\partial \theta} = \frac{\partial \ell(y, z^\star)}{\partial z^\star} \left(I - \frac{\partial f_\theta(z^\star, x^\star)}{\partial z^\star} \right)^{-1} \frac{\partial f_\theta(z^\star, x^\star)}{\partial \theta} = (\mu^\star)^\top \frac{\partial f_\theta(z^\star, x^\star)}{\partial \theta} \tag{3.18}$$

In other words, we can compute the exact needed derivatives with respect to θ by simply *re-using* the converged solution $v^\star$, without the need to double backpropagate through the KKT system. The same considerations, but just negative, apply to the case where $\ell^{\text{outer}} = -\ell$.

3.3.5 Regularization

As was observed in previous work [Bai et al., 2021, Winston and Kolter, 2020], the stability of convergence of a DEQ is directly related to the conditioning of the Jacobian matrix at the equilibrium point. To that end, we primarily adopt the regularization proposed by [Bai et al., 2021] which upper bounds the implicit model's stability by estimating their trace with the Hutchinson estimator: $\text{tr}(J_z) = \text{tr}\left(\frac{\partial f_\theta(z,x)}{\partial z}\right) = \mathbb{E}_{\epsilon \in \mathcal{N}(0,I)}[\epsilon^\top J_z^\top J_z \epsilon]$. However, our exact implementation is subtly different from the original proposal in that we regularize the Jacobian matrix at a *randomly chosen* iterate along the optimization trajectory $(x^{(k)}, z^{(k)})$ instead of just the last iterate (x^*, z^*). This modification is especially important as the optimization trajectories become long (which is the case for our problems; e.g., which could take >80 iterations), which essentially encourages the Jacobian to be not only stable at the end but also

during the root-solving process. Specifically, with this modification, the outer objective of JIIO becomes :

$$\underset{\theta}{\text{minimize}} \ \ell^{\text{outer}}(v_\theta^\star(y), y) + \lambda \mathbb{E}_{\epsilon \in \mathcal{N}(0,1)}[\epsilon^\top J_z^\top J_z \epsilon] \tag{3.19}$$

where λ is the regularization coefficient, and in practice we sample 1 or 2 ϵ's to produce a Monte-Carlo estimation of the expectation term.

3.4 Experiments

As our approach provides a generic framework for joint modeling of an implicit network's forward dynamics and the "inner" optimization over the input space, we demonstrate its effectiveness and generalizability on 4 different types of problems that are popular areas of research in machine learning: generative modeling [Goodfellow et al., 2014a, Kingma and Welling, 2014], inverse problems [Bora et al., 2017, Gilton et al., 2020, Gregor and LeCun, 2010], adversarial training [Madry et al., 2018, Wong et al., 2020] and gradient based meta-learning[Finn et al., 2017, Rajeswaran et al., 2019]. In all cases, we show that our joint inference and input optimization (JIIO) provides significant speedups over projected gradient descent applied to DEQ models and that the models trained using JIIO achieve results competitive with standard baselines. In all of our experiments, the design of our model layer f_θ follows from the prior work on multiscale deep equilibrium (MDEQ) models [Bai et al., 2020] that have been applied on large-scale computer vision tasks, and where we replace all occurrences of batch normalization [Ioffe and Szegedy, 2015] with group normalization [Wu and He, 2018] in order to ensure the inner optimization can be done independently for each instance in the batch.

We introduce below each problem instantiation, how they fit into our methodology described in Sec. 3.3.2, and the result of applying the JIIO framework compared to the alternative methods trained in similar settings. Overall, our results provide strong evidence of benefits of performing joint optimizations on implicit models, thus opening new opportunities for future research in this direction.

3.4.1 Generative Modeling

We study the application of JIIO to learning decoder-only generative models that compute the latent representations by directly minimizing the reconstruction loss [Bojanowski et al., 2017, Zadeh et al., 2019]; i.e., given a decoder network D, the latent representation x of a sample y (e.g., an image) is $x = \min_{x \in \mathcal{X}} \|D(x) - y\|_2^2$.[1] Moreover, instead of placing explicit regularizations on the latent space $\mathcal{X}$ (as in VAEs), we follow [Ghosh et al., 2020] to directly train the decoder for reconstruction (and then after training, we fit the resulting latents using a simple density model, post-hoc, for sampling). Formally, given sample data $y_1, \ldots, y_n$ (e.g., n images), the generative model we study takes the following form:

$$\begin{aligned} \underset{\theta}{\text{minimize}} \quad & \sum_{i=1}^{n} \|y_i - h_\theta(z_i^\star)\|^2 \\ \text{subject to} \quad & x_i^\star, z_i^\star = \underset{x, z: z = f_\theta(z, x)}{\text{argmin}} \quad \|y_i - h_\theta(z)\|^2, \; i = 1, \ldots, n \end{aligned} \tag{3.20}$$

where h_θ is a final output layer that transform the activations $z^\star$ to the target dimensionality. We train the MDEQ-based f_θ with the JIIO framework on standard 64×64 cropped images from CelebA dataset, which consists of 202,599 images. We use the standard train-val-test split as used in Liu et al. [2015] and train the model for 50k training steps. We use Fréchet inception distance (FID) [Heusel et al., 2017] to measure the quality of the sampling and test-time reconstruction of the implicit model trained with JIIO and compare with the other standard baselines such as VAEs [Kingma and Welling, 2014]. The results are shown in Table 3.1. JIIO-MDEQ refers to the MDEQ model trained using our setup with 40 JIIO iterations in the inner loop during training (and tested with 100 iterations). MDEQ-VAE refers to an equivalent MDEQ model but with an encoder and a decoder trained as a VAE. We observe that our model's generation quality is competitive with, or better than, each of these encoder-decoder based approaches. Moreover, with the joint optimization

[1] This notation differs from the "standard" notation of latent variable models (where the latent variable is typically denoted by z). However, because x, y, z all have standard meanings in setting above, we change from the common notation here to be more consistent with the remainder of this paper.

proposed, JIIO-MDEQ achieves the best reconstruction quality.

We additionally apply JIIO on pre-trained MDEQ-VAEs (i.e., train an MDEQ-based VAE as usual on optimizing ELBO [Kingma and Welling, 2014], and take the decoder out) for test-time image reconstruction. The result (shown in Table 3.1) suggests that the reconstructions obtained as a result are better even than the original MDEQ-VAE. In other words, JIIO can be used with general implicit-mode-based decoders at test time even if the decoder wasn't trained with JIIO.

Model	Generation	Reconstruction
VAE Kingma and Welling [2014]	48.12	39.12
RAE Ghosh et al. [2020]	**40.96**	36.01
MDEQ-VAE	57.15	45.81
MDEQ-VAE (w/ JIIO)	-	42.36
JIIO-MDEQ	46.82	**32.52**

Table 3.1: Comparison of FID scores attained by standard generative models with our method, which performs joint optimization. We use 40 solver iterations (for the augmented DEQ) to train the JIIO model reported in this table.

One of the key advantages presented by JIIO is the relative speed of optimization over simply running gradient descent (or its adaptive variants like Adam [Kingma and Ba, 2015a]). Table 3.2 shows our timing results for one optimization run on a single example for various models (averaged over 200 examples). We observe that performing 40 iterations of projected Adam takes more than 9× the time taken by 40 iterations of JIIO, which we used during training and more than 3.5× the time taken by 100 iterations of JIIO which we use for reconstructions at test time (e.g., to produce the results in Table 3.1, though both of them lead to similar levels of reconstruction loss).

Figure 3.2: Samples generated with JIIO on a small MDEQ network.

Fig 3.3 shows the reconstruction loss as it evolves across a single optimization run for an MDEQ model trained with JIIO. This again clearly shows that JIIO converges vastly faster (in terms of wall-clock time) than if we handle the inner optimization

separately as in prior works, demonstrating the advantage of joint optimization. However, it's interesting to note that JIIO optimization seems somewhat unstable (see Fig. 3.3) and fluctuates more as well. This seems to be an artifact of the specific acceleration scheme we use (see more details in 3.4.5).

3.4.2 Inverse Problems

We also extend the setup mentioned in section 3.4.1 directly to inverse problems. These problems, specifically, can be approached as either an unsupervised or a supervised learning problem, which we discuss separately in this section. To demonstrate how JIIO can be applied, we will be using image inpainting and image denoising as example inverse problem tasks, which was extensively studied in prior works like Chang et al. [2017], Gilton et al. [2021]. For the inpainting task, we randomly mask a 20x20 window from the image and train the model to adequately fill the missing pixels based on the surrounding image context. For the image denoising tasks, we add random gaussian noise $\varepsilon \sim \mathcal{N}(0, \sigma^2 I)$ with $\sigma = 0.2$ and $\sigma = 0.4$, respectively, to all pixels in the image, and train the model to recover the original image. We use the same datasets and train-test setups as in the generative modeling experiments in Sec. 3.4.1.

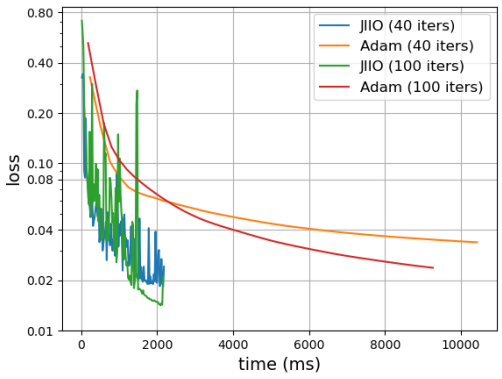


Figure 3.3: Cost changing with time for Adam v/s JIIO optimization. Tested on models trained with 40 and 100 JIIO iterations respectively

Model	time taken (ms)
PGD : 20 iters	4360
JIIO : 80 iters	1401

Table 3.2: Time taken to compute adversarial example of a MDEQ model on MNIST

Model	time taken (ms)
Adam : 40 iters	7659
JIIO : 40 iters	862
JIIO : 100 iters	2156

Table 3.3: Time taken to perform JIIO optimization v/s Adam in the generative modeling/inverse problem experiments

Unsupervised inverse problem solving

Bora et al. [2017], Chang et al. [2017] have showed that we can solve most inverse problems by taking a pre-trained generative model and using that as a prior to solve for the likely solutions to the inverse problems by optimizing on the input space of the generative model. Specifically, given a "generator" network $G : \mathcal{X} \to \mathcal{Y}$, mapping from the latent space $\mathcal{X}$ to an observed space $\mathcal{Y}$, that models the data generating distribution, they show that one can solve any inverse problem by optimizing the following objective:

$$\underset{x \in \mathcal{X}}{\text{minimize}} \ \|\hat{y} - AG(x)\|_2^2. \tag{3.21}$$

where $\hat{y} = Ay \in \mathcal{Y}$ represents the corrupted data point, $y \in \mathcal{Y}$ is the uncorrupted data and $A : \mathcal{Y} \to \mathcal{Y}$ denotes the measurement matrix that defines the specific type of inverse problem that we try to solve (e.g., for image inpainting, it would be a mask with the missing regions filled in with zeros. For deblurring, it would be a convolution with a gaussian blur operator etc.). They call it unsupervised inverse problem solving. Likewise, we can use the pre-trained generator from section 3.4.1 to solve most inverse problems by simply solving a slightly modified version of the inner problem in (3.28):

$$\underset{x, z: z = f_\theta(z, x)}{\text{minimize}} \quad \|\hat{y} - Ah_\theta(z)\|^2 \tag{3.22}$$

In table 3.4, the unsupervised results for VAE and MDEQ-VAE generators are obtained by optimizing (3.21) using Adam for 40 iterations, while for the JIIO trained models, we optimize (3.22) with 100 JIIO iterations. JIIO-MDEQ-40 and JIIO-MDEQ-100 refer to JIIO-MDEQ models trained with 40 and 100 inner-loop iterations respectively. The results in Table 3.4 show that on all 3 problems, JIIO trained generators produce results comparable to the VAE and MDEQ-VAE generators. Moreover, as shown in section 3.4.1, JIIO also converges much faster than Adam applied to a MDEQ-VAE generator.

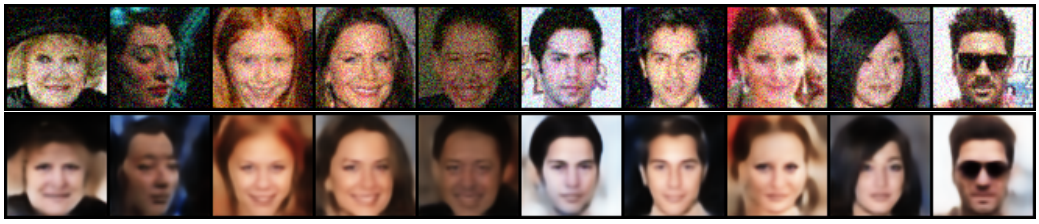


Figure 3.4: Unsupervised Image Denoising with additive noise sampled from $\mathcal{N}(0, 0.2)$: (top) Noisy image; (bottom) Recovered Image

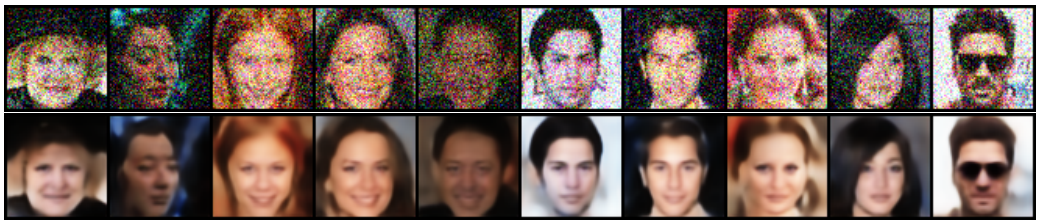


Figure 3.5: Unsupervised Image Denoising with additive noise sampled from $\mathcal{N}(0, 0.4)$: (top) Noisy image; (bottom) Recovered Image

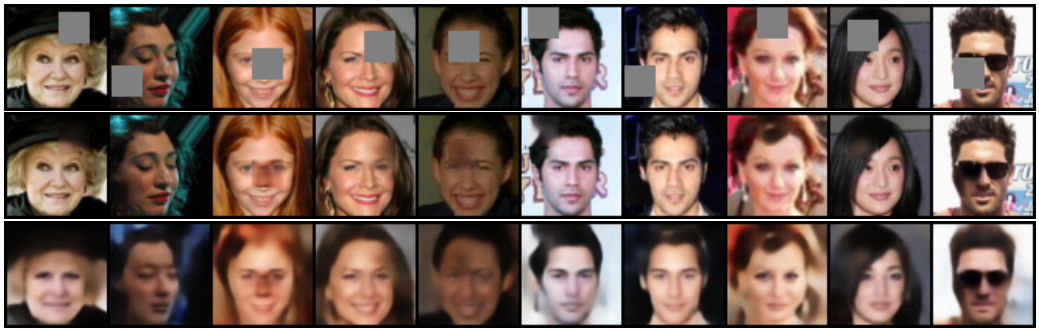


Figure 3.6: Unsupervised Image Inpaiting: (top) Incomplete image; (middle) Inpainted image; (bottom) Reconstructed Image

Table 3.4: Comparison of Median PSNR values for supervised and unsupervised inverse problem solving approaches. The top 3 rows show models that are trained for the specific inverse problem and the latter 5 show pre-trained generative models re-purposed for solving inverse problems.

Task	**Model**	**Inpainting**	**Denoising** ($\sigma = 0.2$)	**Denoising** ($\sigma = 0.4$)
Supervised	AE	17.9	18.72	18.32
	MDEQ-AE	17.06	18.58	18.49
	JIIO-MDEQ-100	16.90	18.22	17.89
Unsupervised	VAE (Adam) Bora et al. [2017]	15.34	15.31	15.24
	MDEQ-VAE (Adam)	16.62	16.96	16.87
	JIIO-MDEQ-40	15.88	17.08	16.03
	JIIO-MDEQ-100	15.87	17.86	17.55

Supervised Inverse problem solving

While the unsupervised inverse problem solving works reasonably well, we can also learn models to solve specific inverse problems to obtain better performance. Specifically, given uncorrupted data $y_1, \ldots, y_n$, and the measurement matrix A, we can train a network $G_\theta : \mathcal{Y} \to \mathcal{Y}$ mapping from the corrupted sample $\hat{y}_i = Ay_i$ to the uncorrupted sample y_i by minimizing:

$$\underset{\theta}{\text{minimize}} \quad \sum_{i=1}^{n} \|y_i - G_\theta(Ay_i)\|^2 \tag{3.23}$$

Now, instead of modeling G_θ as an explicit network, we could also model it as a solution to the inverse problem in (3.22) and the resulting parameters can be trained as follows:

$$\begin{aligned} &\underset{\theta}{\text{minimize}} \quad \sum_{i=1}^{n} \|y_i - h(z_i^\star)\|^2 \\ &\text{subject to} \quad x_i^\star, z_i^\star = \underset{x, z: z = f_\theta(z, x)}{\text{argmin}} \quad \|Ay_i - Ah(z)\|^2, \; i = 1, \ldots, n \end{aligned} \tag{3.24}$$

As shown in Table 3.4 this yields models competitive with their autoencoder based counterparts, while being better than all the unsupervised approaches. Each of the baseline models in the supervised section of Table 3.4 are trained by simply optimizing (3.23) with the corresponding model replacing G_θ. However, note that given the

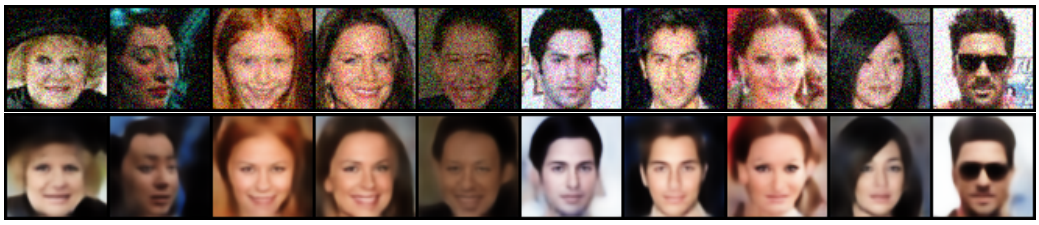


Figure 3.7: Supervised Image Denoising with additive noise sampled from $\mathcal{N}(0, 0.2)$: (top) Noisy image; (bottom) Recovered Image

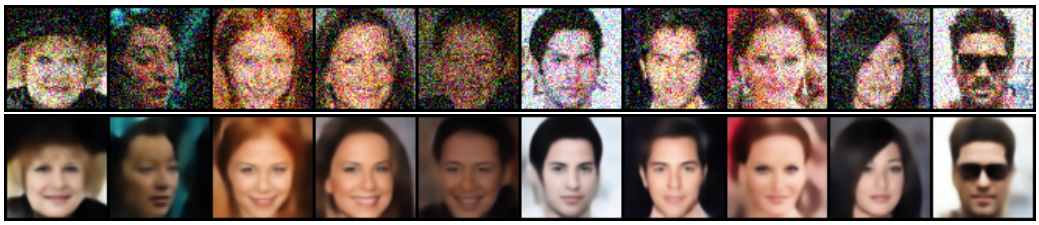


Figure 3.8: Supervised Image Denoising with additive noise sampled from $\mathcal{N}(0, 0.4)$: (top) Noisy image; (bottom) Recovered Image

models here are trained on specific inverse problems, one would have to train a new model for each new problem as opposed to the unsupervised approach.

Figures 3.7, 3.4, 3.8, 3.5, 3.9, 3.6 show examples from unsupervised and supervised trained JIIO-MDEQ models on the tasks. We see that the unsupervised models approach the performance of the supervised alternatives on most tasks, however, the supervised approaches do tend to perform significantly better on the inpainting task.

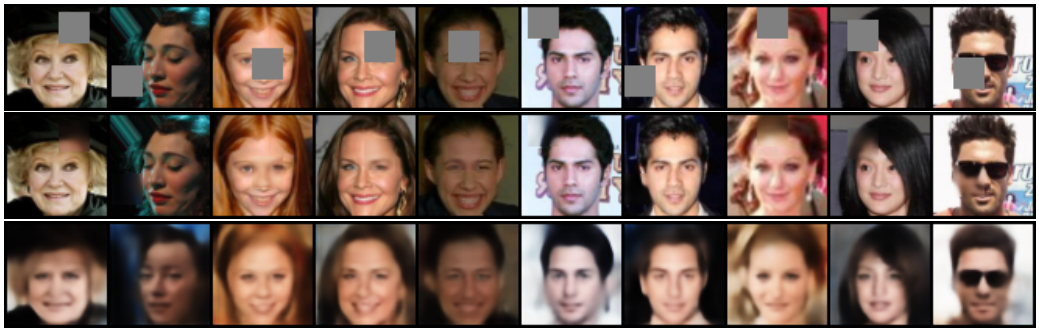


Figure 3.9: Supervised Image Inpaiting: (top) Incomplete image; (middle) Inpainted image; (bottom) Reconstructed Image

3.4.3 Adversarial Training

Although the previous two tasks were based on image generation, we note that our approach can be used more generally for input optimization in DEQs and illustrate that by applying it to ℓ_2 adversarial training on DEQ-based classification models. Specifically, given inputs x_i, y_i; $i = 1, \dots, n$, adversarial training seeks to optimize the objective

$$\underset{\theta}{\text{minimize}} \quad \sum_{i=1}^{n} \max_{\|\delta\|_2 \leq \epsilon} \ell(h_\theta(x_i + \delta_i), y_i) \tag{3.25}$$

We apply our setting to a DEQ-based classifier $h_\theta(x) = h_\theta(z^\star_\theta(x))$ where $z^\star = f_\theta(z^\star, x)$. In this setting, we embed the iterative optimization over δ (with projected gradient descent) into the augmented DEQ, and write the problem as

$$\begin{aligned} &\underset{\theta}{\text{minimize}} \quad \sum_{i=1}^{n} \ell(h_\theta(z_i^\star), y) \\ &\text{subject to} \quad z_i^\star, \delta_i^\star = \underset{z, \delta : z = f_\theta(z, x+\delta), \|\delta\|_2 \leq \epsilon}{\text{argmin}} -\ell(h_\theta(z), y), \ i = 1, \dots, n \end{aligned} \tag{3.26}$$

Specifically we train MDEQ models on CIFAR10 Krizhevsky et al. [2009] and MNIST LeCun et al. [1998] using adversarial training against L2 attacks with $\epsilon = 1$ for 20 epochs and 10 epochs respectively using the standard train-val-test splits. Table 3.5 shows the robust and clean accuracy of models trained using PGD adversarial training, JIIO adversarial training and vanilla training. We also show the robust and clean accuracy of models trained with PGD and JIIO with $\epsilon = 0.5$ for CIFAR10 in Table 3.6 given that $\epsilon = 0.5$ is common to use for CIFAR10. We find that models trained using JIIO have comparable robust and clean accuracy on both datasets. Furthermore, when tested on models trained without adversarial training, we observe that JIIO serves as a comparable attack method to PGD.

Table 3.4.2 shows the time taken to find the adversarial example for a single image of MNIST using 20 iterations of PGD and 80 iterations of JIIO. We again observe more than 3x speedups when using JIIO over using PGD while obtaining competitive robust accuracy. However, note that, unlike previous experiments in generative modeling/inverse problems, performing adversarial training with truncated JIIO

Table 3.5: Comparison of adversarial training approaches on L2 norm perturbations with $\epsilon = 1$. The rows represent the training procedure and the columns represent the testing procedure

Datasets	**Train (↓) Test (→)**	Clean	PGD	JIIO
MNIST	Clean	99.45 ± 0.03	80.1 ± 1.87	65.88 ± 4.72
	PGD	99.18 ± 0.03	96.53 ± 0.05	95.74 ± 0.04
	JIIO	99.32 ± 0.09	95.74 ± 0.22	96.63 ± 0.58
CIFAR	Clean	78.47 ± 0.94	2.38 ± 0.41	3.71 ± 4.01
	PGD	54.91 ± 1.01	37.4 ± 0.26	36.17 ± 0.55
	JIIO	55.54 ± 0.82	37.31 ± 0.67	37.77 ± 0.84

Table 3.6: Comparison of adversarial training approaches on L2 norm perturbations with $\epsilon = 0.5$. The rows represent the training procedure and the columns represent the testing procedure

Datasets	**Train (↓) Test (→)**	Clean	PGD	JIIO
CIFAR	Clean	78.47 ± 0.94	4.54 ± 1.33	4.85 ± 2.93
	PGD	68.48 ± 0.81	51.77 ± 0.75	50.14 ± 0.74
	JIIO	67.79 ± 2.33	51.39 ± 0.56	51.25 ± 0.66

iterations would lead to significant reduction in robust performance due to the adversarial setting.

3.4.4 Gradient based Meta-Learning

Gradient (or optimization) based meta-learning defines a bi-level optimization problem where the outer loop learns meta-parameters for a distribution of tasks while the inner loop learns task-specific parameters, typically using a small amount of data. This bi-level structure blends itself naturally into the types of input optimization problems we have been looking at. Specifically, taking few-shot learning as the use case, we are given a collection of tasks $\{\mathcal{T}_i\}_{i=1}^N$, each associated with a dataset $\mathcal{D}_i$, from which we can sample two disjoint sets: $\mathcal{D}_i^{tr} = \{(s_{i,k}^{tr}, y_{i,k}^{tr})\}_{k=1}^K$ and $\mathcal{D}_i^{te}\{(s_{i,k}^{te}, y_{i,k}^{te})\}_{k=1}^K$, where each (s, y) is a data-label pair. Gradient based meta-learning for few shot learning problem can be framed as a bi-level optimization problem with input optimization as

the inner loop:

$$\begin{aligned} &\underset{\theta}{\text{minimize}}\ \ell(x_i^\star, h_\theta(x_i^\star, s_{i,k}^{te}), y_{i,k}^{te}), \quad i = 1, \ldots, N \\ &\text{subject to}\ \ x_i^\star = \underset{x_i}{\text{argmin}}\ \ell(x_i, h_\theta(x_i, s_{i,k}^{tr}), y_{i,k}^{tr}), \quad k = 1, \ldots, K \end{aligned} \tag{3.27}$$

where $x_i^\star$ are the task specific parameters inferred during the inner-loop optimization and θ are the meta-parameters. Note that the task specific parameters can be treated as inputs, and thus the inner problem becomes an input optimization problem. In this case, with a DEQ network, the above problem can be modified slightly as:

$$\begin{aligned} &\underset{\theta}{\text{minimize}}\ \ell(x_i^\star, h_\theta({z_{i,k}}^{\star(te)}), y_{i,k}^{te}), \quad i = 1, \ldots, N \\ &\text{subject to}\ \ x_i^\star, {z_{i,k}^\star}^{(tr)}, {z_{i,k}^\star}^{(te)} = \underset{x_i, z_{i,k}^{(tr/te)} = f_\theta(z_{i,k}^{(tr/te)}, x_i)}{\text{argmin}} \ell(x_i, h_\theta(z_{i,k}^{tr}), y_{i,k}^{tr}), \quad k = 1, \ldots, K \end{aligned} \tag{3.28}$$

Clearly, this modified problem can now to solved using JIIO in the inner loop. Table 6 shows the accuracies obtained by a JIIO-trained DEQ model and various baseline meta-learning approaches like Implicit-MAML [Rajeswaran et al., 2019], MAML [Finn et al., 2017] and ReptileNichol and Schulman [2018] on the 5-way, 1-shot task on Omniglot Lake et al. [2015]. We observe that the JIIO-trained DEQ model achieves comparable accuracy to the baselines. The partially lower accuracy numbers of the DEQ model may be attributed to the fact that we do not differentiate through the fixed point variable $x_i^\star$ while optimizing the outer loop objective. We observed very poor conditioning in our experiments when trying to differentiate through $x_i^\star$ (thus requiring a large number of fixed-point updates for the backward pass and resulting in poor quality gradients) and instead hope to explore that further in future work. Table 7 shows the time taken by JIIO v/s Adam on the DEQ model to perform optimization in the inner loop. Again, we observe that JIIO takes more than 2.7x lesser time to converge, demonstrating the main advantage of using JIIO for input optimization problems with DEQs.

Algorithm/Model	5-way, 1-shot
MAML	98.7
Reptile	97.68
iMAML, GD	99.16
JIIO-DEQ	97.33

Table 3.7: Accuracy obtained on the 5-way, 1-shot task from the omniglot dataset

Model	time taken (ms)
PGD : 20 iters	28948
JIIO : 100 iters	11836

Table 3.8: Time taken by JIIO vs. Adam to perform inner-loop optimization in DEQ-based meta-learning tasks

3.4.5 Choice of Acceleration method

We perform ablation experiments on the choice of the acceleration methods for performing the joint optimization. Figure 6.3 shows various approaches that can be used to accelerate JIIO applied to a reconstruction task on a pre-trained MDEQ-VAE decoder. Broyden's method treats its solution as a solution to a root finding problem on the KKT conditions instead of as a minimization problem and hence, given the non-convex nature of the problem, could end up chasing arbitrary stationary points. The Anderson based approaches treat the problem as a minimization problem and hence are able to perform much better. Specifically, Type-I Anderson mixing is usually more unstable (a phenomenon that had been discussed in prior work like Fang and Saad [2009] and Zhang et al. [2020]), and yet manages to attain the lowest loss values in 100 JIIO iterations. Type-II Anderson, on the other hand, allows for much smoother optimization process although it plateaus at higher loss values. However, since speed was an important point of consideration for our experiments, we nevertheless went with Anderson Type-I over Type-II and picked the output of the optimizer using a heuristic that traded off between a small KKT residual and a small cost. Overall, we observed that the criterion for this iterate selection can be somewhat flexible but preferably kept consistent between training and testing runs when we perform JIIO.

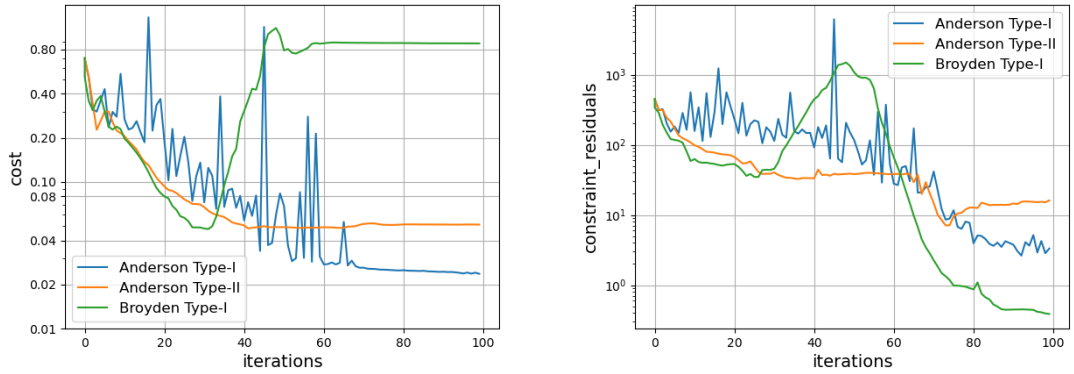


Figure 3.10: Comparing different acceleration techniques on a MDEQ-VAE decoder

3.4.6 Compute and Runtime

The generative model and inverse problems experiments were trained on 4 RTX-2080 Ti GPUs. The generative modeling experiments were run for 50k training steps. For the inverse problems experiments, we trained models with 100 JIIO iterations for 25k training steps, taking 4.5-5 days for each training run. Adversarial training experiments with JIIO trained models take 9-10 hours for CIFAR10 on 3 RTX-2080 Ti GPUs while taking 5-6 hours on 2 RTX-2080 Ti GPUs for MNIST experiments. The models trained with projected gradient descent take 20-21 hours to train on 3 RTX-2080 Ti GPUs for the CIFAR10 experiments while taking 13-14 hours to train on 2 RTX-2080 gpus for the MNIST experiments. For our meta-learning experiments, the models trained using JIIO used 4 RTX-2080 Ti GPUs for roughly 2 days.

3.4.7 Space complexity of the method

As pointed out earlier, JIIO has higher memory requirements than the corresponding ADAM version due to the higher memory sizes used in computing the fixed point. For example, in the generative modeling/inverse problems, optimization using JIIO requires 17.45 GB of GPU memory as opposed to vanilla Adam based optimization which simply costs 7.47 GB GPU memory for a batch of 48 images from celebA. However, for the adversarial training problems, the memory requirements were comparable - JIIO requires 2.19 GB memory as opposed to 2.15 for projected gradient

descent for a batch with 96 images from CIFAR10.

3.5 Concluding remarks

We present a novel optimization procedure for jointly optimizing over the input and the forward fixed point in DEQ models and show that, for the same class of models, it is $3 - 9\times$ faster than performing vanilla gradient descent or Adam on the inputs. We also apply this approach to 3 different settings to show it's effectiveness: training generative models, solving inverse problems, adversarial training of DEQs and gradient based meta-learning. In the process, we also introduce an entirely new type of decoder only generative model that performs competitively with it's autoencoder based counterparts.

Despite these features, we note that there is substantial room for future work in these directions. Notably, despite the fact that the augmented joint inference and input optimization DEQ can embed both processes in a "single" DEQ model, in practice these joint models take substantially more iterations to converge as well (often in the range of 50-100) than traditional DEQs (often in 10-20 iterations), and correspondingly often use larger memory caches within methods like Anderson acceleration. Thus, while we make the statement that these augmented DEQs are "just" another DEQ, this relative difficulty in finding a fixed point likely adds challenges to training the underlying models. Thus, despite ongoing work in improving the inference time of typical DEQ models, there is substantial room for improvement here in making these joint models truly efficient.

Chapter 4

Unbundling and Mitigating Gradient Variance in Differentiable Bundle Adjustment Layers

In this chapter, we use differentiable bundle adjustment layers in visual odometry tasks as an example optimization layer to analyze the sources of gradient variance in optimization layers. We identify three sources of **gradient variance** in bundle adjustment layers and propose modifications to reduce this variance, leading to faster and more stable training. Experiments show a 2x to 2.5x speedup in training times and improved generalization across different training setups.

The contents of this chapter have been previously published at CVPR 2024 in [Gurumurthy et al., 2024].

4.1 Introduction

Ego and exo pose estimation are essential for agents to safely interact with the physical world. These tasks have a long history of being tackled using geometry-based

optimizaton Engel et al. [2014, 2017], Lepetit et al. [2009], Mur-Artal et al. [2015], and in the last decade, using deep networks to directly map inputs to poses Park et al. [2019], Wang et al. [2017, 2021], Zhou et al. [2017]. However, both these classes of approaches have shown brittleness — not being robust to outliers in the data or having poor accuracy in unseen scenes.

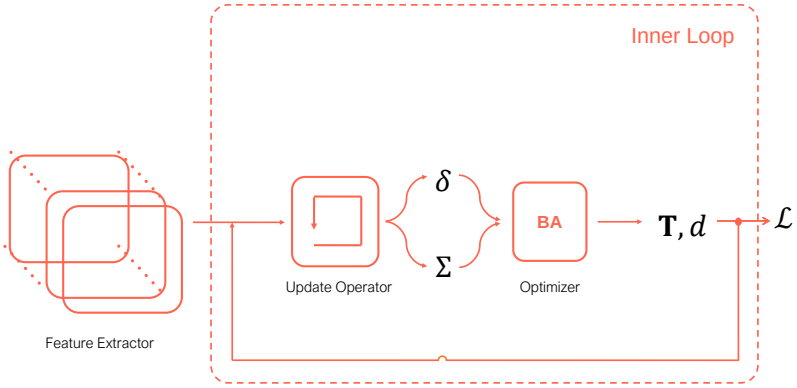


Figure 4.1: SOTA pose estimation methods Lipson et al. [2022], Teed and Deng [2021], Teed et al. [2023] tightly-couple learned front-ends with traditional BA optimizers. However, they can be slow to converge.

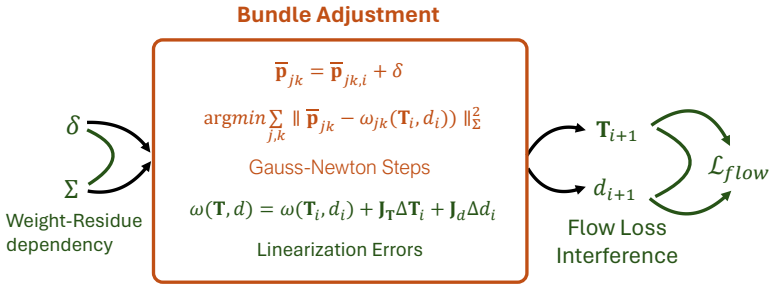


Figure 4.2: We identify three factors that lead to high variance in gradients during training with bundle adjustment layers. We then propose to mitigate them by using weights from the inner-loop optimization to weigh the correspondence outer objective resulting in a reduction in gradient variance and more stable training.

More recently, approaches that combine the best of both worlds in *learning to optimize* have demonstrated substantially better performance than previous methods Lipson

et al. [2022], Teed and Deng [2021], Teed et al. [2023]. These approaches combine a learned iterative update operator that mimics an optimization algorithm with implicit layers that enforce known geometric constraints on the outputs. This general architecture has appeared across many tasks even beyond pose estimation Adler and Öktem [2017, 2018], Bhardwaj et al. [2020], Fu and Fallon [2023], where in each case an accurate and robust task-specific optimization solver is learned. In Lipson et al. [2022], Teed and Deng [2021], Teed et al. [2023], for the task of pose estimation, a recurrent network that iteratively updates pose and depth is learned through a differentiable weighted bundle adjustment (BA) layer that constrains the updates. Feature correspondences are also iteratively refined together with the poses, thereby dynamically removing outliers and leading to better accuracy.

Although these methods achieve state-of-the-art (SOTA) results, they take exceedingly long to train. Teed and Deng [2021] mention that DROID-SLAM takes 1.5 weeks to train with 4x RTX 3090, while Teed et al. [2023] mention that DPVO takes 3.5 days to train on a RTX 3090. Likewise, in our experiments, training the object pose estimation method from Lipson et al. [2022] took 1 week with 2x RTX 6000 for the smallest dataset reported in their paper.

In this paper, we first investigate the reasons for the slow training convergence speeds of these methods, using *deep patch visual odometry* (DPVO) Teed et al. [2023] as an example problem setting for this analysis. We find that the bundle adjustment layer and the associated losses used in this setting lead to a high variance in the gradients. We identify three reasons contributing to the high variance. First, improper credit assignment arising from the specific choice of flow loss used which leads to interference between the gradients of outlier and inlier points. Second, improper credit assignment arising from the linearization issues in the bundle adjustment layer. And lastly, the dependence of the weight gradients on the residual of the BA objective resulting in the outliers dominating those gradients. We show how each of these problems lead to an increase in the gradient variance.

Next, we leverage the analysis to propose a surprisingly simple solution to reduce the variance in gradients by weighting the flow loss according to the 'importance' of the points for the problem, resulting in significant improvements in training speed and

stability while achieving better pose estimation accuracy. We also experiment with other variance reduction techniques and demonstrate the superior performance of our proposed solution. Using DPVO as an example, we demonstrate 2-2.5x speedups with these simple modifications. Furthermore, we show that the modifications also make the training less sensitive to specific training setups. As a result, we are able to train in a non-streaming setting, while reaching similar accuracies in the streaming setting, thereby leading to a further 1.2-1.5x speedup in training. Lastly, we apply the modifications to DROID-SLAM Teed and Deng [2021] with little hyperparameter tuning to show that the proposed modifications transfer to a completely new pipeline providing similar speedups and stable training. Furthermore, we show that our best models achieve about 50% improvement on the TartanAir validation set and a 24% improvement on the TartanAir test set. To summarize, the contributions of this paper are as follows:

- We identify three candidate reasons for high variance in the gradients when differentiating through the BA problems for Visual Odometry (VO) and SLAM and show how they are all affected by the presence of outliers.
- Using DPVO Teed et al. [2023] as an example VO pipeline, we propose a simple modification to the loss function that reduces the variance in the gradients by mitigating the effect of outliers on the objective.
- We show that the above modification results in significant speedups and improvements in accuracy of the model on the TartanAir Wang et al. [2020] validation and test splits used in the CVPR 2020 challenge. Further, we show that the modifications can be applied out-of-the-box to other settings/methods that use differentiable BA layers, such as DROID-SLAM and the non-streaming version of DPVO to obtain similar benefits.

4.2 Connections to related work

Pose Estimation using Deep Learning. A large body of works have tackled pose estimation, and we describe a few representative works that use deep learning here. For a broader overview, we refer the reader to Cadena et al. [2016], Chen

et al. [2020b], Fan et al. [2022]. Wang et al. [2017, 2021], Zhou et al. [2017] proposed deep networks to directly estimate ego pose between pairs of frames. Koestler et al. [2022], Min et al. [2020], Sarlin et al. [2020], Tateno et al. [2017] integrate learned representations (features or depth) into traditional ego-pose estimation pipelines. Czarnowski et al. [2020], Tang and Tan [2019], Yang et al. [2020] imposed geometric constraints on ego-pose network outputs via differentiable optimization layers. Similar approaches have been proposed for the task of multi-object pose estimation where 2D-3D correspondences are directly regressed Park et al. [2019], Rad and Lepetit [2017] and then passed through a differentiable PnP solver Chen et al. [2020a] for pose estimation. Overall, these works showed that deep learning could be applied to these tasks but fell short in accuracy and generalization.

Optimization-inspired iterative refinement methods have been applied to ego-pose Clark et al. [2018], Jatavallabhula et al. [2020], Ummenhofer et al. [2017], Zhou et al. [2018] and exo-pose estimation Iwase et al. [2021], Labbé et al. [2020] where the network iteratively refines its pose estimates as an update operator in order to satisfy geometric constraints. More recently, methods that iteratively refine poses and correspondences in a tightly-coupled manner have been proposed Lipson et al. [2022], Teed and Deng [2021], Teed et al. [2023]. In these works, a network predicts patch correspondences Teed et al. [2023] or dense flow Lipson et al. [2022], Teed and Deng [2021] which are then updated together with poses and depths in an alternating manner where one feeds into the other through differentiable geometric operations. In addition to correspondences, these methods also predict weights for the correspondences which have been shown to be important for pose estimation accuracy in many independent works Burnett et al. [2021], Kanazawa and Kanatani [2001], Muhle et al. [2023], Ranftl and Koltun [2018]. Overall, these iterative methods have achieved impressive performance in terms of accuracy and generalization, but they still need large GPU memories Teed et al. [2023] and their training times are prohibitively long which has limited their adoption for research.

Challenges with Implicit Optimization Layers. With the advent of implicit layers, it is possible to incorporate an optimization problem as a differentiable layer Agrawal et al. [2019], Amos and Kolter [2017], Pineda et al. [2022], which captures complex behaviours in a neural network. The BA layer Tang and Tan [2019] used in

this work is an instance of such layers. In the forward pass, an implicit optimization layer solves a regular optimization problem given the current estimate of problem parameters. In the backward pass, one differentiates through the KKT conditions of the optimization problem to update the problem parameters.

While these implicit optimization layers boast expressive representational power, there exist challenges with such layers. Firstly, these problems naturally take on a bilevel structure, where the inner optimization learns the problem parameters and the outer problem optimizes for the decision variables given the current estimation of problem parameters. As a result, these problems are inherently hard to solveAmos and Kolter [2017], Amos et al. [2018a], Howell et al. [2022a], as their easiest instantiation, e.g., linear programs for both inner and outer problems, can be non-convex Beck and Schmidt [2021]. While the convergence issues may be alleviated by techniques such as using good initialization Amos and Kolter [2017] or robust solvers, there does not exist a general solution to the authors' best knowledge. Secondly, a range of numerical issues can arise from implicit optimization layers. The gradients derived from KKT conditions are only valid at fixed points of the problem. In practice, the solver may need to run long enough to reach a fixed point or a fixed point may not exist at all Amos et al. [2018a], Donti et al. [2021]. The problem may be ill-conditioned due to reasons such as stiffness or discontinuities from physical systems Suh et al. [2022] or compounding of gradients in chaotic systems Metz et al. [2021]. A number of problem-specific solutions have been proposed Howell et al. [2022a,b], Suh et al. [2022] to these problems. For example, Antonova et al. [2023], Suh et al. [2022] use zeroth-order methods to deal with non-smoothness and non-convexity in the problem. Howell et al. [2022a,b] use interior point relaxations to smooth the discontinuities. Similarly, Bianchini et al. [2023], Donti et al. [2021] use penalty-based relaxations to handle the discontinuities. It's also common to regularize the inner problem during the backward pass to deal with ill-conditioning Amos et al. [2018a], Fung et al. [2022]. However, given the vastness of the problems, we are of the opinion that this is still a broadly under-studied area.

4.3 Background

In this section, we review the approach of DPVO Teed et al. [2023] for iterative ego-pose estimation, which serves as an example setting for all our analysis and experiments.

Feature Extraction. A scene, as observed from an input video, is represented as a set of camera poses $\mathbf{T}_j \in \mathbb{SE}(3)$ and square image patches $\mathbf{P}_k$. Patches are created by randomly sampling 2D locations in the image and extracting $p \times p$ feature maps centered at these coordinates $\mathbf{p}_k$. A bipartite patch-frame graph is constructed by placing an edge between every patch k and each frame j within distance r of the patch source frame. The reprojections of a patch in all of its connected frames form the trajectory of the patch.

Update Operator. The update operator iteratively updates the optical flow of each patch over its trajectory. The operator updates the embedding of each edge (k, j) of the patch graph via temporal convolutions and message passing. These updated embeddings are used by two MLPs to predict flow revisions $\delta_{jk} \in \mathbb{R}^2$ and confidence weights for each patch $\Sigma_{jk} \in \mathbb{R}^2$ between $[0, 1]$. The flow revisions are used to update the reprojected patch coordinates $\bar{\mathbf{p}}_{jk} := \bar{\mathbf{p}}_{jk} + \delta_{jk}$, which are passed to a differentiable BA layer along with their confidence weights Σ_{jk}.

Differentiable Bundle Adjustment. The bundle adjustment (BA) layer solves for the updated poses and depths that are geometrically consistent with the predicted flow revisions. The BA layer operates on a window of the patch graph to update the camera poses and patch depths, while keeping the revised patch coordinates $\bar{\mathbf{p}}_{jk}$ fixed. The BA objective is as follows:

$$\min_{\mathbf{T}_{ij}, d_k} \sum_{(k,j)} ||\bar{\mathbf{p}}_{jk} - \Pi(\mathbf{T}_{ij}, \Pi^{-1}(\mathbf{p}_k, d_k))||^2_{\Sigma_{jk}} \tag{4.1}$$

where Π denotes the projection operation, d_k denotes the depth of the k^{th} patch in the source frame i, and $\mathbf{T}_{ij}$ is the relative pose $\mathbf{T}_i \mathbf{T}_j^{-1}$. This objective is optimized using

two Gauss-Newton iterations. The optimized poses and depths are then passed back to the update operator to revise the patch coordinates, and so on in an alternating manner.

Training Loss. The network is supervised using a flow loss and pose loss computed on the intermediate outputs of the BA layer. The flow loss computes the distance between the ground truth patch coordinates and estimated patch coordinates over all the patches and frames:

$$\mathcal{L}_{\text{flow}} = \sum_{j,k} \|\mathbf{p}^*_{jk} - \hat{\mathbf{p}}_{jk}\|_2 \tag{4.2}$$

where $\hat{\mathbf{p}}_{jk} = \Pi(\mathbf{T}_{ij}, \Pi^{-1}(\mathbf{p}_k, d_k))$ and $\mathbf{p}^*_{jk}$ is the corresponding reprojection of patch k in frame j using the ground truth pose and depth. Note that this loss amounts to a difference in the patch coordinates and not in the flows as the source patch coordinates in each flow term cancel out.

The pose loss is the error between the ground truth poses $\mathbf{G}$ and estimated poses $\mathbf{T}$ for every pair of frames (i, j):

$$\mathcal{L}_{\text{pose}} = \sum_{(i,j)} \|\text{Log}_{\mathbb{SE}(3)}[(\mathbf{G}_i.\mathbf{G}_j^{-1})^{-1}.(\mathbf{T}_i.\mathbf{T}_j^{-1})]\|_2 \tag{4.3}$$

The total loss is a weighted combination of the flow loss and pose loss,

$$\mathcal{L} = 10\mathcal{L}_{\text{flow}} + 0.1\mathcal{L}_{\text{pose}} \tag{4.4}$$

The original DPVO model is trained on random sequences of 15 frames, where the first 8 frames are used together for initialization and the subsequent frames are added one at a time. Their model is trained for 240K iterations using 19GB of GPU memory which takes 3.5 days on an RTX 3090. A total of 18 iterations of the update operator is applied on each sequence, where the first 8 iterations are applied during initialization as a batch-optimization, and the subsequent iterations are for every new, added frame. In our paper, we refer to these update iterations as the 'inner-loop optimization', this mode of training as the 'streaming' setting, and training models

in our experiments to only batch-optimize the first 8 frames as the 'non-streaming' setting.

4.4 Factors affecting training convergence

In this section, we identify three possible causes for slow training convergence. We show how each of these result in noisier/higher variance gradients during training, and consequently result in instabilities and slowdowns.

4.4.1 Flow loss interference

The flow loss defined in Equation 4.2 operates on the reprojected patch coordinates $\hat{\mathbf{p}}_{jk}$ which are computed using the optimized poses $\mathbf{T}_i, \mathbf{T}_j$ and depth d_k outputs from the BA layer. Thus, the gradient of the loss with respect to d_k (and similarly for poses $\mathbf{T}_i, \mathbf{T}_j$) can be written as follows,

$$\nabla_{d_k} \mathcal{L}_{\text{flow}} \propto \sum_j \nabla_{d_k} \Pi(\mathbf{T}_{ij}, (\mathbf{p}_k, d_k)). \tag{4.5a}$$

$$\nabla_{T_i} \mathcal{L}_{\text{flow}} \propto \sum_{k,j} \nabla_{T_i} \Pi(\mathbf{T}_{ij}, (\mathbf{p}_k, d_k)). \tag{4.5b}$$

Thus, the gradients with respect to each reprojected patch $\hat{\mathbf{p}}_{jk}$ gets aggregated in the computation graph at the corresponding depth d_k (likewise for poses $\mathbf{T}_i, \mathbf{T}_j$) at the output of the BA layer. This becomes problematic when a significant fraction of the projections are noisy/outliers, as the noisy/outlier gradients would dominate the inlier gradients in the sum in Equation 4.5a, leading to more noise in the total gradient estimate.

Since these gradients are also backpropagated through the BA layer, it results in noisy gradient estimates for the network parameters as well. Specifically, in the BA layer, each $d_i/\mathbf{T}_i$ are again a function of all the predicted flows and weights associated with that point/frame. Thus, the same noisy gradient computed at $d_i/\mathbf{T}_i$, gets backpropagated to all the associated points. This leads to the gradient estimates

being noisy even at the 'good' predictions by the network.

4.4.2 Linearization errors in BA gradient

Given gradient estimates at the output poses and depth of the BA layer $\nabla_d \mathcal{L}, \nabla_{\mathbf{T}} \mathcal{L}$, the gradients with respect to its input flows and weights are computed as follows:

$$\begin{aligned} \nabla_\delta \mathcal{L} = &-(\nabla_{\mathbf{T}} \mathcal{L})^T (\mathbf{J}_{\mathbf{T}}^T \Sigma \mathbf{J}_{\mathbf{T}})^{-1} \mathbf{J}_{\mathbf{T}}^T \Sigma \\ &- (\nabla_d \mathcal{L})^T (\mathbf{J}_d^T \Sigma \mathbf{J}_d)^{-1} \mathbf{J}_d^T \Sigma \end{aligned} \tag{4.6a}$$

$$\begin{aligned} \nabla_\Sigma \mathcal{L} = &-(\nabla_{\mathbf{T}} \mathcal{L})^T (\mathbf{J}_{\mathbf{T}}^T \Sigma \mathbf{J}_{\mathbf{T}})^{-1} \mathbf{J}_{\mathbf{T}}^T \mathrm{diag}(\mathbf{r}) \\ &- (\nabla_d \mathcal{L})^T (\mathbf{J}_d^T \Sigma \mathbf{J}_d)^{-1} \mathbf{J}_d^T \mathrm{diag}(\mathbf{r}) \end{aligned} \tag{4.6b}$$

where, $\mathbf{r} = (\bar{\mathbf{p}}_{kj} - \hat{\mathbf{p}}_{kj})$ is the bundle adjustment residual, $\mathbf{J}_d$ and $\mathbf{J}_{\mathbf{T}}$ are the jacobians of the projection $\Pi(\mathbf{T}_{ij}, \Pi^{-1}(\mathbf{p}_k, d_k))$ with respect to depth d and pose $\mathbf{T}$ respectively. This expression can be derived by applying the implicit function theorem (Theorem 1B.1) Dontchev et al. [2009], on the BA problem.

Since the projection is non-linear containing multiple multiplicative operations, we observe that the Jacobians $\mathbf{J}_d$ and $\mathbf{J}_{\mathbf{T}}$ themselves are a function of d and $\mathbf{T}$. Thus, a high variance in the initialized d or $\mathbf{T}$ naturally lead to a high variance in the Jacobians, thereby leading to a high variance in the corresponding gradients ∇_Σ and ∇_δ, which are then backpropagated through the network. In our setup, d is initialized to random values and $\mathbf{T}$ is initialized to identity. Thus, the variance from linearization is primarily contributed by the linearization around the current d.

The use of a weighted objective in the BA problem partially mitigates this issue by masking out the gradients on the flows corresponding to the outlier points (which contribute the most to this high variance). However, the high variance remains problematic especially in the initial iterations of training (when the weight estimates themselves are not very accurate) and in the initial iterations of the inner-loop optimization when a large fraction of the depth and pose estimates are inaccurate.

4.4.3 Dependence of weight gradients on the BA residual

In the previous section, we discussed the effect of outliers on the BA linearization and consequently on the gradients. However, outliers in the BA problem contribute to an increase in gradient variance in a more straightforward way. Specifically, they have a direct effect on the gradient of the weights, as can be seen from the expression in Equation 4.6b. The expression shows the direct dependence of the weight gradients on the residual, $\mathbf{r} = (\bar{\mathbf{p}}_{jk} - \hat{\mathbf{p}}_{jk})$, of the BA problem. Thus, the presence of high residual points in the optimization problem result in high variance in the weight gradients.

In fact, the presence of outliers also biases the weight gradients towards highly positive values as the training objective tries to reduce the influence of the outliers. This consequently leads to a collapse in the weight distribution. However, we observe that a straightforward fix used by prior workTeed and Deng [2021], Teed et al. [2023], i.e, clipping the magnitude of gradient passing through the weights easily mitigates this bias. We discuss more details on this effect with a simple illustrative example in subsection 4.6.6.

To summarize, the above section highlights various aspects of the existing setup that contribute to noisy/high variance gradients. The noise and high variance in gradient estimates leads to ineffective parameter updates, thereby leading to training instabilities and slowdown. Furthermore, it's also important to note that the aforementioned effects exacerbate each other. For example, worse weight estimates result in bad BA outputs, which in turn contribute to worsening the flow loss interference and BA linearization errors, which further leads to noisier gradients thereby slowing down weight/flow updates, thus repeating the vicious cycle. By the same argument, mitigating either of these effects can also provide significant improvements on other problems!

4.5 A very simple solution: Weighted flow loss

We start with observing that all three problems mentioned in the previous section get exacerbated by the presence of outliers or computing gradients through outliers. So the natural question is if there exist obvious solutions to mask out the outliers in the outer training problem.

One of the tricks used by Teed and Deng [2021], Teed et al. [2023] already partially accounts for this in the pose loss, i.e, they do not include the pose loss for the first couple of inner-loop iterations, thereby mitigating some of the issues discussed in subsection 4.4.2. This simple modification in Teed and Deng [2021], Teed et al. [2023] seems to provide a significant boost in training speeds as we show in our ablation experiments in subsection 4.6.8.

Similar heuristics for the flow loss are harder to find as the depth/flow estimates of a significant fraction of points are bad even at the latter inner-loop iterations. Conventionally, SLAM and visual odometry problems define heuristic kernels on the flow residuals Barron [2019a], Chebrolu et al. [2021] depending on the expected distribution of residuals/errors to trade-off between robustness and accuracy. Unfortunately, coming up with a similar simple/consistent heuristic to define 'outliers' in the outer training problem is more challenging as the errors and distribution of errors vary across examples, training iterations and inner optimization iterations. This requires a heuristic that adapts to the specific example, training convergence, and inner-loop optimization iteration.

Conveniently, we find that the weights learnt by the inner update operator for the bundle adjustment problem satisfy all these properties as they adapt online with the changing distribution of errors/residuals. Moreover, empirically we observe that the network learns a reasonable weight distribution very early on in training, while adapting the weight distribution rapidly to any changes in flows. Thus, we observe that using these weights to weight the flow loss works surprisingly well. The resulting flow loss is as follows.

$$\mathcal{L}_{\text{flow}} = \sum_{j,k} \|\mathbf{p}^*_{jk} - \hat{\mathbf{p}}_{jk}\|_{\Sigma^{\perp}_{jk}} \tag{4.7}$$

where $\perp$ denotes the stop gradient operator to prevent the objective from directly driving the weights to zero (We provide more discussion on what factors prevent these weights from collapsing to zero in subsection 4.6.7). The main difference between this and Equation 4.2 is that each residual in this objective is weighted by the weights Σ_{jk} predicted by the network for the inner BA problem. Intuitively, this objective incentivizes the network to focus on the points which are important for the inner optimization problem at that optimizer step / training iteration for that particular example.

Although the modification seems trivial and obvious in hindsight, we observe that it is significantly more effective than various other (more complicated) variance reduction approaches we tried (studied in Appendix subsection 4.6.5). This apparent simplicity and effectiveness underscore the value of the proposed modifications!

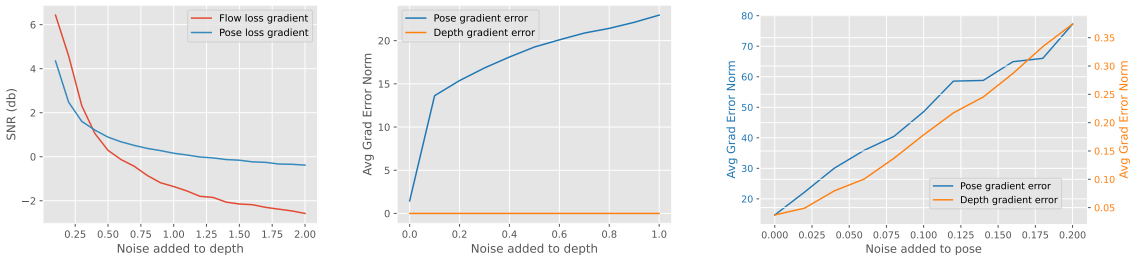
Balancing loss gradients. The introduction of the weighted flow loss changes the gradient contribution from the flow loss throughout training as the weight distribution changes. Thus, instead of using fixed coefficients to trade-off between pose and flow loss as in Equation 4.4, we periodically (every 50 training iterations) update the flow loss coefficient β to ensure the gradient contributions of the pose and flow loss remain roughly equal throughout training. Given the infrequency in these updates, they barely affect the training speed and hence are cheap to compute amortized over the entire training run.

$$\beta = \frac{\|\nabla_\theta \mathcal{L}_{\text{pose}}\|_2}{\|\nabla_\theta \mathcal{L}_{\text{flow}}\|_2} \tag{4.8}$$

$$\mathcal{L} = \mathcal{L}_{\text{pose}} + \beta \mathcal{L}_{\text{flow}} \tag{4.9}$$

4.6 Results and Analysis

We analyze the effect of the factors discussed in section 4.4 on the original DPVO model on the TartanAir Wang et al. [2020] dataset. We then analyze a version trained with our proposed weighted flow objective. We show that the weighted objective



(a) Impact of depth noise on linearization. (b) Impact of depth noise on flow loss gradients. (c) Impact of pose noise on flow loss gradients.

Figure 4.3: (a) We compute the signal-to-noise ratio (SNR) in the loss gradients as we artificially add depth noise while linearizing the BA problem for gradient computation. We observe that the SNR in the flow loss deteriorates rapidly indicating its sensitivity to linearization errors. (b) We artificially add noise to a subset of depths right before the flow loss computation. We show the average gradient errors on all the pose and 'clean' depth variables as a result of the added noise. We see a monotonic increase in gradient error in pose gradients as we increase the noise added showing the impact 'outliers' have on the gradients of even the 'inlier' variables. (c) Similar to (b), here we add noise to the the first frame's pose and show the gradient errors on the rest of the frames and depths.

helps increase the signal to noise ratio in the gradients throughout training and show the improvements in performance as a result. We also evaluate the pose estimation performance of this version on the TartanAir Wang et al. [2020], EuRoC Burri et al. [2016], and TUM-RGBD Sturm et al. [2012] benchmarks. We use the average absolute trajectory error (ATE) after Sim(3) alignment of the trajectories, as the evaluation metric for pose estimation.

4.6.1 Analyzing factors affecting gradient variance

To understand the impact of linearization on the gradient variance (subsection 4.4.2), we analyze the impact of adding noise to the depth used to compute the Jacobians in the BA problem. We leave the rest of the forward and backward pass unaltered and only add noise to the depth while computing the linearization for the backward pass in the BA problem. This helps us isolate the effects of linearization on the gradient computations. Specifically, Figure 4.3a shows the signal-to-noise ratio (SNR) of the flow and pose loss gradients with respect to δ with increasing levels of noise. The SNR

is computed assuming the no-depth-noise gradient as the true signal and treating any deviations from it as noise. This yields two interesting observations. First, the SNR deteriorates rapidly in the beginning indicating that the gradients are indeed sensitive to the noise in the iterates used for linearization. Second, the SNR in the flow loss gradients is high initially, but deteriorates rapidly compared to the pose loss gradients with increasing noise. This highlights the need to make flow loss robust to noisy points.

To analyze the effect of flow loss on the gradient noise (subsection 4.4.1), we introduce noise on a few depth points or a single frame pose right before computing the flow loss and study the effect of the noise on the gradients of all the other points/poses. Figure 4.3c shows a monotonic increase in gradient errors on the depths as well as all poses as we increase the noise added to the first pose. Likewise, Figure 4.3b shows the monotonic increase in gradient errors of all poses as we add increasing amounts of noise to all depths on the first frame. This shows how outliers with increasingly large errors can have an increasingly adverse effect on the gradients of the non-outlier points/frames as well. The gradient errors are computed as the average L2 norm of the deviation in gradient from the no-noise gradients.

We analyze the weight residual dependence (subsection 4.4.3) and the resulting variance / bias in Appendix subsection 4.6.6, as its connections to the use of weighted loss are less direct.

4.6.2 Effect of the weighted flow loss on training

To understand the effect of the weighted flow loss on the variance of the gradients, we study the SNR of the gradients on the flow network parameters. Figure 4.4 shows the SNR with the flow loss and the weighted flow loss at different points during training. The plots clearly demonstrate that the usage of weighted flow loss results in a boost in SNR throughout training. The boost is especially prominent in the initial stages of training, when the impact of outliers and noise in the pose/depth estimates are most significant. This clearly shows the promise of using the weighted flow loss instead of the regular flow loss for training.

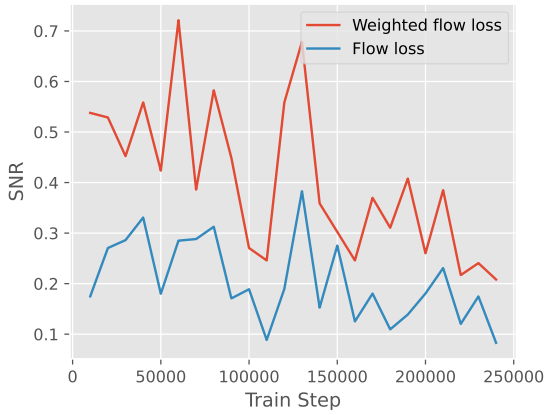


Figure 4.4: We compute the signal-to-noise ratio in the gradients of the flow loss and the weighted flow loss w.r.t flow network parameters at different training iterations of the base model. Specifically, we use the last linear layer's weights of the flow computation head of the network. We find that the weighted flow loss gradients have a higher SNR throughout the training. This is especially true in the initial iterations of training when the outlier count is very high.

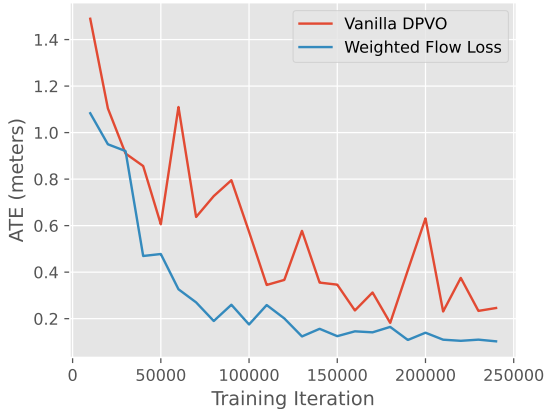


Figure 4.5: We observe that DPVO when trained with our weighted flow loss achieves much faster training, reaching $\sim$0.2 m accuracy in only 80K iterations, and is much more stable. We report the median ATE across three trials on the validation split of TartanAir.

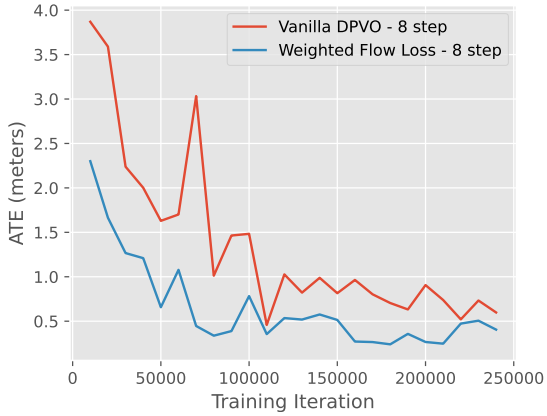


Figure 4.6: We retrain DPVO with and without the modified flow loss in the non-streaming batch setting and evaluate both models on validation sequences from TartanAir in the streaming setting. We observe that, beyond training faster and being more stable, the modified version generalizes better than the original model. This allows the model to be trained on shorter sequences without suffering high performance drops, thanks to the reduced gradient variance. We report the median ATE across three trials.

We retrain DPVO with our modified weighted flow loss on the TartanAir dataset and show its validation error performance across training iterations. We observe in Figure 4.5 that the average ATE of our method drops rapidly compared to the original. While our model takes only 80K iterations to reach an average error of $\sim$0.2 m, the original model reaches the same performance at 180K iterations. In fact, while that's the peak performance reached by the base model, our model continues to improve and reaches a final convergence error of $\sim$0.10 m, achieving half the base model's convergence error on the validation set. We also observe that, unlike the original model, the errors don't fluctuate rapidly over epochs and is more stable.

Further, the reduced variance in gradients allows us to train in other setups as well. For example, Figure 4.6 shows the ATE of our model against the base model trained in the non-streaming setting, i.e. using just 8 frame initialization sequences instead of 15 frame sequences. This allows the models to be trained faster (with per iteration cost of 0.6s vs 1.6s for the streaming version on an RTX A6000 GPU) and with lower GPU memory (7.2GB as opposed to 19.2GB GPU memory). Note that the evaluations are still done as earlier, i.e, by rolling out the model on the full validation sequences in the streaming setting. Yet, we observe that despite being

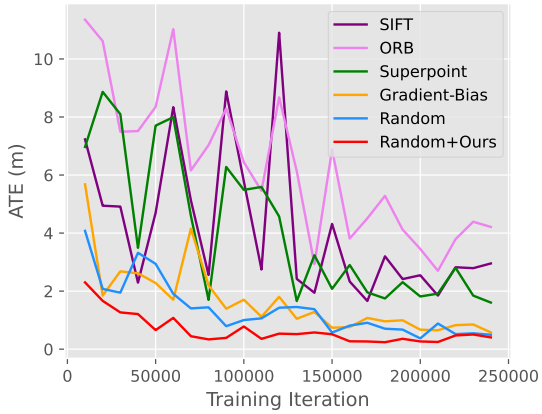


Figure 4.7: We retrain the original DPVO model with standard feature detection methods and observe that our method of random sampling with the modified flow objective has much improved training convergence. We report the median ATE across three trials on the validation split of TartanAir.

trained to only batch-optimize over 8 frames, it generalizes to the streaming setting with our modified models obtaining a peak performance of $\sim$0.2m pose errors in 180K iterations (i.e, 2.7$\times$ faster than the base streaming model). Furthermore, we also test our modifications on DROID-SLAM, a completely different pipeline that also uses Bundle Adjustment layers with little to no hyperparameter tuning. We present the results in Appendix subsection 4.6.4. Again, we observe that the modifications result in significant speedups and stability during training suggesting that the methods and analysis discussed in this paper applicable broadly to approaches using differentiable BA layers.

Finally, to evaluate the ability of our modified model to weight patches effectively, we compare against other standard methods for selecting patches. Specifically, we re-train the original DPVO model with patches selected using SIFT Lowe [2004], ORBRublee et al. [2011], SuperpointDeTone et al. [2018], and naive gradient based sampling instead of the default random sampling. As shown in Figure 4.7, we observe that random sampling along with the weights learned by our network is much more stable and accurate than other patch selection methods.

	ME 000	ME 001	ME 002	ME 003	ME 004	ME 005	ME 006	ME 007	MH 000	MH 001	MH 002	MH 003	MH 004	MH 005	MH 006	MH 007	Avg
ORB-SLAM3* Campos et al. [2021]	13.61	16.86	20.57	16.00	22.27	9.28	21.61	7.74	15.44	2.92	13.51	8.18	2.59	21.91	11.70	25.88	14.38
COLMAP* Schonberger and Frahm [2016]	15.20	5.58	10.86	3.93	2.62	14.78	7.00	18.47	12.26	13.45	13.45	20.95	24.97	16.79	7.01	7.97	12.50
DSO Engel et al. [2017]	9.65	3.84	12.20	8.17	9.27	2.94	8.15	5.43	9.92	0.35	7.96	3.46	-	12.58	8.42	7.50	7.32
DROID-SLAM* Teed and Deng [2021]	0.17	0.06	0.36	0.87	1.14	0.13	1.13	**0.06**	**0.08**	0.05	0.04	**0.02**	**0.01**	0.68	0.30	0.07	0.33
DROID-VO	0.22	0.15	0.24	1.27	1.04	0.14	1.32	0.77	0.32	0.13	0.08	0.09	1.52	0.69	0.39	0.97	0.58
DPVO	0.16	0.11	0.11	0.66	0.31	0.14	0.30	0.13	0.21	0.04	0.04	0.08	0.58	**0.17**	0.11	0.15	0.21
Ours	**0.08**	**0.05**	0.16	**0.30**	**0.27**	**0.08**	**0.20**	0.10	0.18	**0.03**	**0.03**	**0.02**	0.58	0.30	**0.08**	**0.05**	**0.16**

Table 4.1: ATE [m] results on the TartanAir Wang et al. [2020] test split compared to other SLAM methods. For our method and DPVO, we report the median of 5 runs. (*) indicates the method used global loop closure optimization.

	MH01	MH02	MH03	MH04	MH05	V101	V102	V103	V201	V202	V203	Avg
DPVO	0.087	0.055	**0.158**	**0.137**	0.114	0.050	0.140	**0.086**	0.057	**0.049**	0.211	**0.105**
Ours	**0.081**	0.067	0.171	0.179	0.115	**0.046**	0.160	0.097	0.056	0.059	0.252	0.117

Table 4.2: ATE [m] results on the EuRoC dataset Burri et al. [2016] compared to other visual odometry methods. For our method and DPVO, we report the median of 5 runs. The performance of our model is similar to DPVO.

	360	desk	desk2	floor	plant	room	rpy	teddy	xyz	Avg
DPVO	**0.135**	0.038	0.048	0.040	0.036	0.394	0.034	0.064	0.012	**0.089**
Ours	0.145	0.026	**0.044**	0.064	0.031	0.434	0.045	**0.046**	0.012	0.094

Table 4.3: ATE [m] results on the freiburg1 set of TUM-RGBD Sturm et al. [2012]. We evaluate *monocular* visual odometry, and is identical to the evaluation setting in DPVO Teed et al. [2023]. For all methods, we report the median of 5 runs. (x) indicates that the method failed to track. The performance of our model is similar to DPVO.

4.6.3 Test results for pose estimation

We report pose estimation results on the TartanAir Wang et al. [2020] test-split from the CVPR 2020 SLAM competition in Table 4.1, and compare to results from other baseline methods as reported in DPVO Teed et al. [2023]. Traditional optimization-based approaches such as ORB-SLAM3 Campos et al. [2021], COLMAP Schonberger and Frahm [2016], DSO Engel et al. [2017] fail to track accurately and have absolute trajectory errors (ATE) in the order of meters. Iterative learning-based DROID-SLAM Teed and Deng [2021] and its variant without global loop-closure correction (DROID-VO) show reasonable performance, but DPVO is able to show much better accuracy by only tracking a sparse number of patches instead of dense flow. Our modified version, is able to show even better accuracy with a 24% lower error on average. Morever, we observe that our model outperforms DPVO on all but two

sequences in the dataset. Using the same models trained on the TartanAir train set, we also report the results on the EuRoC Burri et al. [2016] and the TUM-RGBD Sturm et al. [2012] benchmark datasets in Table 4.2 and Table 4.3. Here, we obtain similar performance to DPVO. This suggests that, although the weighted flow loss helps improve the model accuracy on similar datasets, it doesn't resolve generalization issues related to domain shift from the TartanAir dataset to the real world.

4.6.4 Results on DROID-SLAM

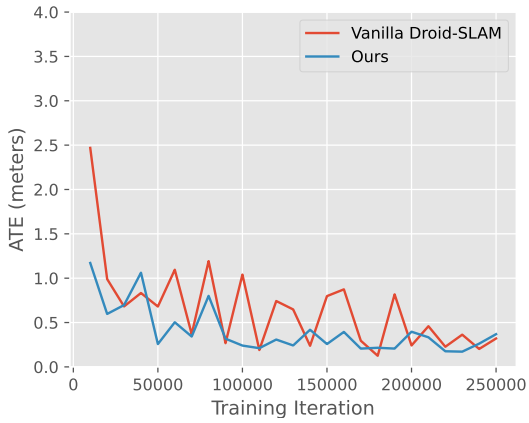


Figure 4.8: We plot the median validation ATE across three trials of DROID-SLAM and our modification at various iterations during training. We observe a clear speedup in training convergence and improved training stability suggesting that our method and analysis generalizes to other approaches using differentiable bundle adjustment layers.

We test our modifications on DROID-SLAM. Specifically, we simply use a weighted flow loss Equation 4.7 instead of their regular flow loss and tune the corresponding loss coefficient value. We keep the rest of the pipeline unchanged. Figure 4.8 shows the validation ATEs throughout training for the baseline model and our modification. We clearly observe that our modification leads to a significant speedup as well as stabilization during training thanks to the reduced variance in the updates. This suggests that the methods and analysis discussed in this paper generalize broadly to a wide variety of approaches using differentiable bundle adjustment layers.

4.6.5 Alternative variance reduction methods

Although we propose one specific method for variance reduction in the paper (i.e using the weighted flow loss), we tested various other approaches towards variance reduction achieving varying degrees of success. We discuss a few of them in this section and compare them against our proposed modification. Figure 4.9 shows the corresponding validation plots on the non-streaming 8-step version of the problem.

(a) *Removal of pose loss from initial inner-loop iterations*: As discussed in the previous section, removing the pose losses from the first few inner-loop iterations (Figure 4.12) is very effective in reducing the variance and indeed we incorporate this change during our training.

(b) *Pose and flow interpolation using ground truth*: Given that we have access to ground truth flow and poses during training, we could interpolate between the pose/flow estimated by the network and the ground truth in order to reduce the variance of the iterates. We vary the interpolation coefficient such that we move from the ground truth distribution at the beginning of training to the model distribution towards the end of training. While this indeed reduces the variance, the interpolation adds a bias due to the distribution shift over training resulting from changing the interpolation coefficients. The other issue here arises from having to also interpolate the hidden state in an ad-hoc manner to account for the interpolated poses/flows.

(c) *Grad Correction using ground truth flows*: The ground truths can alternatively also be used to correct the gradients. Specifically, we clip the gradients on the flows at the input of the BA layer that aren't aligned with the ground truth ($\nabla_{\bar{\mathbf{p}}_{jk}} \|\bar{\mathbf{p}}_{jk} - \mathbf{p}^*_{jk}\|$) to zero or reverse the sign. We observe that this and other similar methods for gradient correction bias the gradient, resulting in worse final performance, despite speeding up training in the initial iterations. One could potentially consider using this in the initial iterations of training to obtain the initial speedups and then switching to regular training.

(d) *(Heuristic) weights using ground truth*: We experimented with a number of heuristic weights that down-weight the outlier flows in the flow loss (Equation 4.7) based on their distance from the ground truth. We plot one such variant in Figure 4.9

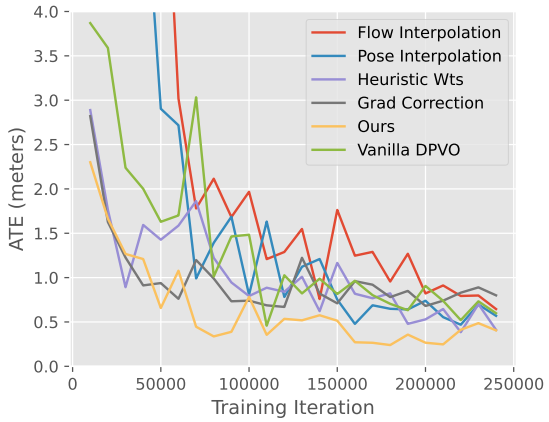


Figure 4.9: We plot the median validation ATE across three trials of alternate variance reduction strategies applied on the non-streaming 8-step version of DPVO.

that uses the following heuristic for weight computation in the flow loss:

$$\Sigma_{ijk} = \frac{1}{4(p^*_{ijk} - \hat{p}_{ijk})/m + 1)}; \quad m = \text{median}(\mathbf{p}^* - \hat{\mathbf{p}}) \tag{4.10}$$

We observe an improvement over the baseline, but couldn't find any heuristic that worked better than using the network-predicted weights (ours).

The experiments in this section show that although there exist various methods for variance reduction, finding the right combination that does not add unintended bias to the gradients is a challenging task. This underscores the value of our analysis and the significance of the performance boost provided by our method.

4.6.6 Analyzing the effect of weight residual dependence

To understand the impact of the gradients on the weights, let's consider a simplified version of the weighted least squares problem as follows:

$$f^* = \underset{f}{\text{argmin}} \frac{1}{2} \sum_i \Sigma_i (f - \hat{f}_i)^2 \tag{4.11}$$

Figure 4.10: We plot the mean weight values of our method and our method without the weight gradient clipping throughout training. We observe a clear reduction in the weight magnitudes in the variant without gradient clipping suggesting a clear positive bias in the weight gradients in this variant. These ablations are performed on the non-streaming 8-step version of the problem.

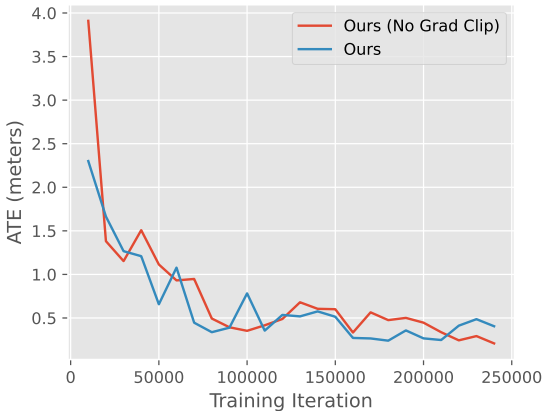


Figure 4.11: We plot the median validation ATE across three trials of our method and our method without the weight gradient clipping at various iterations during training. Both methods show similar performance suggesting that the architectures and inference method used in this paper is robust to the changes in weight distribution. These ablations are performed on the non-streaming 8-step version of the problem.

where Σ_i and f_i are the parameters of the optimization problem, with $\Sigma_i \in [0, 1]$ being the weights. We further assume that we know the ground truth, f_{gt}, desired as the output of the optimization problem. We thus define the outer minimization problem as a minimization over the following loss:

$$\min_{f_i, \Sigma_i \forall i} \mathcal{L} = \min_{f_i, \Sigma_i \forall i} |f_{\text{gt}} - f^*| \tag{4.12}$$

The corresponding gradients of the outer loss $\mathcal{L}$ w.r.t the weights Σ_i is given by :

$$\nabla_{\Sigma_i} \mathcal{L} = -\text{sign}(f^* - f_{\text{gt}}) * \left(\frac{1}{\sum_i \Sigma_i} \right) * (f^* - \hat{f}_i). \tag{4.13}$$

This gradient is positive when $(f^* - f_{\text{gt}})(f^* - \hat{f}_i) < 0$ given the weights are positive. In the presence of an outlier f_k with high weight Σ_k, the least squares solution f^* is biased towards f_k. Thus, the corresponding $(f^* - f_{\text{gt}})(f^* - \hat{f}_k) < 0$, thereby resulting in a highly positive gradient $\nabla_{\Sigma_k} \mathcal{L}$ given the high residual.

In our original stochastic parameterized problem, each $\Sigma_i = \Sigma_\theta(x)$ is parameterized by a network followed by a sigmoid. Thus, the outliers with small Σ_i don't contribute much to the gradient due to the sigmoid saturation. However, the outliers with large Σ_i contribute significantly to the gradient. Given the outliers with large Σ_i are inclined to get a highly positive gradient $\nabla_{\Sigma_k} \mathcal{L}$ as shown above, we observe an overall positive shift in the gradient values in the presence of outliers. If not handled carefully, this can potentially result in a collapse in the weight distribution to near zero values.

One of the architecture hacks used by Teed and Deng [2021], Teed et al. [2023], i.e, clipping the magnitude of gradient passing through the weights, incidentally already mitigates this issue to a large extent :

$$\nabla_{w_i} \mathcal{L} := \max(\min(\nabla_{\Sigma_i} \mathcal{L}, \gamma_{\max}), \gamma_{\min}) \tag{4.14}$$

where $\gamma_{\min}$ and $\gamma_{\max}$ are the minimum and maximum gradient value thresholds. Figure 4.10 shows the average weight magnitudes for the last optimization iterate on two training runs, with and without the gradient clipping throughout training.

We observe a clear reduction in the weight magnitudes as a result of removing the gradient clipping. This is especially pronounced in the initial iterations of training when the number of outliers are high.

Interestingly, we observe that this change in the weight distribution does not have much effect on the performance of the model itself as shown in Figure 4.11. The validation scores of the two models are similar throughout training. This phenomenon likely reflects the resilience inherent in the architecture and inference methodology used in this work.

However, we believe that the positive bias in weight gradients can result in training instabilities in other settings that involve differentiable weighted least squares problems if not handled carefully. We thus include it in our discussions in this paper for a complete treatment although they don't seem to have a direct negative effect in the specific settings we investigate in this paper.

4.6.7 Weight collapse

The analysis in the previous section raises an interesting question about our specific setting, given that any collapse in weight values will significantly affect our training setup as the weights appear explicitly in the loss. However, as pointed earlier, empirically we don't observe any weight collapse. This raises two important questions. Why do the weights not collapse? Does our setting have any natural regularization that prevents such a weight collapse? We attempt to answer these questions as follows.

- The positive bias in the weight gradients described in the previous section indeed exists resulting in the weight distribution being somewhat more conservative than necessary. However, this positive bias only results in this conservative offset instead of a total collapse in the weight values due to gradient clipping and the fraction of outliers being relatively minimal. Hence, this isn't problematic as a significant fraction of the weights still remain pretty high (Figure 4.10) thereby providing sufficient training signal.

- We detach the weight values from the computation graph used in the loss to enforce the stop grad operation (Section 5). Hence, the loss grads don't directly contribute towards reduction of weight values. In fact, we show the analytical gradient w.r.t. the weights when differentiating through a simplified least squares problem in Equation 4.12. An additional multiplication by the weight on these grads simply changes the individual grad magnitudes and doesn't change the direction. Furthermore, at a distributional level, we observe that the weight values between the two runs (weighted and unweighted) remain roughly similar throughout training.
- Due to the presence of both pose and flow loss, even if the weight values are reduced for some arbitrary reason, that would lead to a temporary reduction in the contribution of flow loss to the overall loss. This results in an increase in the contribution of pose loss gradients, acting as a buffer in case weights suddenly start collapsing (which we don't observe empirically). However, we acknowledge that the first two *inner-loop* iterations don't use pose loss and would not have this buffer. However, given that the procedure is iterative, the network would learn to correct for it from the third iteration onward.

4.6.8 Effect of adding pose loss to the first few inner loop iterates

As discussed in section 4.5, one of the tricks used by Teed and Deng [2021], Teed et al. [2023], already offer a partial solution to mask out the outliers in the loss, i.e., they simply do not add a pose loss on the first few inner loop iterates. Given the depth and pose estimates are especially bad in the first few inner loop iterations, this serves as an obvious solution to mitigate the gradient variance resulting from the linearization issues described in subsection 4.4.2. Figure 4.12 shows the effect of adding the pose loss on those initial iterates as well (in this case the first two iterates). We clearly observe a degradation in performance as a result due to the increased variance in gradients.

Figure 4.12: We plot the median validation ATE across three trials of our method and our method with pose loss added to all the iterates. We observe a clear deterioration in performance when adding the pose loss to the first two inner loop iterates as well. These ablations are performed on the non-streaming 8-step version of the problem.

4.7 Conclusions and Future work

In this paper, we analyze the high variance in gradients during the training of pose estimation pipelines that use differentiable bundle adjustment layers. We identify three plausible causes for the high variance and show how they lead to slower training and instability. We then propose a simple solution for these problems involving a weighted correspondence loss. We implement this on a SOTA VO pipeline and demonstrate improved training stability and a 2.5x training speedup. We also observe a 24% accuracy improvement on the TartanAir test split and similar accuracy as the vanilla model on other benchmarks. Unsurprisingly, the modifications don't automatically improve the model's ability to tackle distribution shift. We also observe that the depth accuracy for low-weight points, which might be important for dense SLAM approaches, deteriorates.

More broadly, this work highlights the peculiarities introduced by optimization layers in the training dynamics. For regular deep network layers, we control for the variance at initialization by appropriately initializing the parameters in the network. Then, during training we use appropriate normalization layers and take small steps on the parameters to ensure things remain 'nice'. However, with optimization layers, we

lack this freedom to arbitrarily change/normalize parameters to suit the variance given the parameters typically have a physical meaning. Moreover, using arbitrary variance reduction techniques with these layers can often result in biases creeping into the training run, thus requiring us to design priors and consider the bias-variance trade-off with more care.

Chapter 5

Value-Gradient Updates for Off-Policy Deep Iterative Learning Control

In this chapter, we use the reinforcement learning problem as an example to illustrate how we can leverage approximate models within end-to-end learning pipelines to achieve data-efficient learning while avoiding potential biases introduced by the modeling approximations. Specifically, we propose the Value Gradient Update taking inspiration from Iterative Learning Control to use approximate model jacobians while computing the gradients but using the real data for policy and value function evaluation. This method significantly enhances sample efficiency and improves zero-shot transfer performance even when just using the value gradient update in the simulator alone.

The contents of this chapter have been previously published at L4DC 2023 in [Gurumurthy et al., 2023a].

5.1 Introduction

Deep reinforcement learning has been applied successfully in domains ranging from games [Berner et al., 2019, Silver et al., 2017] to real-world robotics control tasks [Chen et al., 2022b, Hwangbo et al., 2019, Xie et al., 2018, Xu et al., 2019]. However, one of the primary issues hampering its widespread adoption in real-world domains is its sample inefficiency. Even relatively simple tasks sometimes require hundreds of thousands to millions of samples. One reason attributed to this sample inefficiency is the reliance on zeroth order gradient estimates to perform policy and value function updates. This often follows from the assumption that the environment being trained on is non-differentiable.

However, we often have access to an approximate simulator of the environment that can provide approximate jacobians and gradients of the dynamics and reward function respectively. In fact, another set of approaches to online adaptation, based on iterative learning control (ILC), have been shown to be very sample-efficient by utilizing these approximate jacobians to speed up policy optimization, albeit in very limited settings, involving repetitive tasks and systems with smooth dynamics [Mueller et al., 2012, Schoellig et al., 2012].

In this work, we aim to build on the generality of reinforcement learning based approaches while utilizing approximate jacobians from the simulator to speed up policy learning. Specifically, we propose a simple update to the gradient of the value function by utilizing the simulator gradients, which we call the value-gradient update. The update boils down to simply minimizing the residual of the gradient of the bellman error and can be added to any actor-critic reinforcement learning algorithm (even off-policy algorithms!). We show that the value gradient update is in fact equivalent to the multiplier (corresponding to the dynamics constraint) updates performed in optimal control approaches such as (i)LQR. We further show that, the update can be easily modified to accommodate approximate jacobians from a simulator by simply substituting the approximate jacobians evaluated along the rollouts (collected from the real environment) in the value gradient update, just like ILC.

This update has several advantages: 1) The ability to use approximate jacobians makes it useful even when we do not know the accurate model of the system; 2) By explicitly supervising the value function gradients, we obtain more reliable value function gradients thereby speeding up policy optimization; 3) It can be used to perform off-policy updates (especially important considering we want to use past experiences for better sample efficiency) on the value function and policy; and 4) It can handle non-smooth and discontinuous dynamics/losses by simply skipping or clipping the gradient updates on transitions with such discontinuities.

The contributions of this work are as follows:

- We introduce the value gradient update, a general update that can be used with any value function based RL algorithm given an (approximate) simulator. Unlike previous work, its compatibility with SOTA off-policy algorithms and its ability to accommodate approximate simulator jacobians makes it particularly appealing for applicability in real systems.
- We add the update to SAC, a popular off-policy RL algorithm, and demonstrate that adding the value gradient update leads to up to 2-3x sample efficiency (and sometimes better performance) across six tasks. In all the tasks, we assume access to a 'nominal' model of the system (often a simplified model with inaccurate parameters) and a more complicated, realistic model of the system as a proxy for the real world. Interestingly, we also observe that policies/value functions pretrained using the value-gradient update in the nominal model show better zero-shot performance and adapt faster in the real system compared to the RL baseline, indicating that using the simulator jacobians even for pre-training leads to more robust policies.

5.2 Connections to related work

Using differentiable simulators for policy optimization Trajectory Optimization [Bertsekas, 2012, Betts, 2001] approaches have traditionally used differentiable models for optimizing over a sequence of states and actions. However, most learning based approaches have relied on reinforcement learning [Bertsekas, 2019, Kober et al., 2013, Sutton and Barto, 2018] (i.e not assuming differentiability of the simulator) for

optimizing policies despite the physics being analytically differentiable. Multiple works recently [Metz et al., 2021, Suh et al., 2022] have pointed out issues with using first order estimates of the gradient for policy optimization. [Metz et al., 2021] shows that simulator gradients can often be chaotic and differentiating through multiple simulation steps can result in vanishing and exploding gradients. [Suh et al., 2022] further points out that first order estimates can also suffer from higher bias or variance in certain cases. [Mora et al., 2021] address these issues by computing optimal trajectories using trajectory optimization and then performing imitation learning. [Xu et al., 2022] address the exploding/vanishing gradient issue by only differentiating through a short horizon and using a value function instead at the tail to amortize the value and gradient estimates. [Parag et al., 2022] learns the value function using sobolev descent on optimal trajectories on smooth environments. However, they assume access to accurate gradients from the simulator and neither of these methods are suitable for learning in the real world using an approximate simulator as [Xu et al., 2022] is an on-policy algorithm and requires multiple parallel runs for each update. And [Mora et al., 2021, Parag et al., 2022] require optimal trajectories which we might not be able to easily obtain from the real system.

Using approximate gradients from a simulator We often have access to an approximate differentiable simulator even when trying to learn policies in the real world. One common approach to use this approximate model is to perform model predictive control where we replan the trajectories using the approximate model at each time step [Mayne, 2014]. This is also common practice with iterative learning control [Agarwal et al., 2021, Moore, 2012, Schöllig and D'Andrea, 2009, Vemula et al., 2022] based approaches where the jacobians from the simulator are evaluated along the sampled trajectory and used to find the updated actions for some repetitive task. ILC has also been extended to update the model [Abbeel et al., 2006, Jackson et al., 2022] where the model is then used to perform policy optimization or trajectory optimization assuming the learnt model is accurate. [Heess et al., 2015] further extend it by using the real samples and the approximate gradients from the learned model to compute policy gradients. However, they use importance sampled estimates of these policy gradients to compute off-policy updates which is hard to scale and unstable due to the higher variance[Glynn, 1994, Liu et al., 2020].

In this work, we take inspiration from these approaches and propose an update to the value function, using the approximate jacobians from a differentiable simulator, that can be used with any value function based RL algorithm. The suitability of this update for off-policy RL methods makes it particularly appealing compared to most prior work in this area.

5.3 Preliminaries and Background

5.3.1 Notation

We address policy and value function learning in continuous action spaces using approximate simulators with differentiable dynamics and reward functions. We will consider Markov decision processes (MDPs) with infinite horizon, but most of the proposed modifications can easily be extended to finite horizon as well. The MDP state transitions are modelled as a function $f : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ and the reward using a bounded reward function $r : \mathcal{S} \times \mathcal{A} \rightarrow [r_{min}, r_{max}]$. We also assume access to a simulator $g : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ which approximately models the transition dynamics f. The simulator g and the reward function r are assumed differentiable in most regions of the state action space.

5.3.2 Q-learning based Actor Critic Methods

Q-learning based actor critic methods such as DDPG [Lillicrap et al., 2015] and SAC [Haarnoja et al., 2018b] learn a Q-function by performing bellman backups over the transitions (s, a, s', r) sampled from the MDP as follows :

$$Q(s, a) := r(s, a) + \gamma Q(s', a'(s')), |_{s'=f(s,a)}, \tag{5.1}$$

where, $a'(s')$ can be obtained by using the current policy estimate $\pi_\theta(s')$ or by solving : $max_{a'} Q(s', a')$. For methods such as DDPG and SAC, we further learn a policy by optimizing

$$\max_\theta \mathcal{J}_\pi(\theta) = \max_\theta \mathbb{E}_{(s\sim\mathcal{D}, a\sim\pi_\theta(.|s))}[Q(s, a) + \alpha\mathcal{R}(\pi_\theta(a|s))], \tag{5.2}$$

Here, we actually care more about the gradient of the value function than the exact value function itself since the gradient of policy objective only consists of $\nabla_a Q(s,a)$:

$$\nabla_\theta \mathcal{J}_\pi(\theta) = \mathbb{E}_{s\sim\mathcal{D},\epsilon\sim\mathcal{N}}[\nabla_a Q(s,a)\nabla_\theta a_\theta(s,\epsilon) + \alpha\nabla_\theta \mathcal{R}(\pi_\theta(a_\theta(s,\epsilon)|s)], \tag{5.3}$$

where $\mathcal{R}(\pi_\theta(a|s))$ here is a regularizer and $a_\theta(s,\epsilon)$ is the policy output.

5.3.3 Iterative Learning Control

For repetitive tasks involving taking the same sequence of actions repeatedly, iterative learning control (ILC) offers a very sample efficient approach to adapt policies using real world rollouts. In this paper, we focus on the recent optimization based reformulations of the ILC problem [Agarwal et al., 2021, Schöllig and D'Andrea, 2009, Vemula et al., 2022] where we evaluate the control inputs by performing rollouts in the true system while computing updates to the controls using the approximate model. In an LQR like setup, this update can be computed by linearizing the dynamics and quadraticizing the cost around the observed trajectory and then computing the LQR updates:

$$\min_{\Delta a} \quad \sum_{t=0}^{N} J(\Delta s_t, \Delta a_t) \tag{5.4}$$

$$s.t \quad \Delta s_{t+1} = \hat{A}_t^g \Delta s_t + \hat{B}_t^g \Delta a_t \tag{5.5}$$

where $s_{0:N}$ is the trajectory observed when the controls $a_{0:N-1}$ are rolled out in the true system f, and $\hat{A}_t^g$, $\hat{B}_t^g$ are obtained by linearizing the approximate model g along the observed trajectory.

5.4 Method

In this section, we introduce the value gradient update. We show that it complements the TD update and can be introduced directly into existing reinforcement learning algorithms. We will then point out its relationship with the LQR backward pass and discuss the relationship with optimal control approaches more broadly. We will also

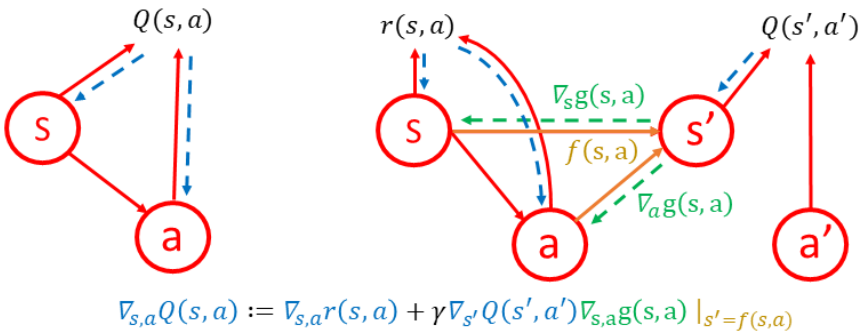


Figure 5.1: Diagram illustrating the value gradient update with an approximate simulator. The dotted lines indicate the gradient flow and the solid lines indicate the forward pass. The left diagram shows the value gradients directly computed from the Q function whereas the right diagram shows the value gradients computed using a single step rollout with the approximate jacobians $\nabla_{s,a} g(s,a)$ computed using the simulator. Note that the rollout is still computed in the true environment $f(s,a)$. The value gradient update uses the rolled out estimate to supervise the amortized gradients on the left.

discuss how to adapt it to settings when we have access to only approximate jacobians from an imperfect simulator. Finally, we add the value gradient update to SAC [Haarnoja et al., 2018b] as an example of an off-policy RL algorithm incorporating the update.

5.4.1 Value gradient update

The value gradient update can be derived as a straightforward extension of the TD update presented above by differentiating Eq. 6.3 with respect to the inputs :

$$\nabla_{s,a} Q(s,a) := \nabla_{s,a} r(s,a) + \gamma \nabla_{s,a} Q(s', a'(s')), |_{s'=f(s,a)} \tag{5.6}$$

where the R.H.S can be further expanded by applying chain rule,

$$\nabla_{s,a} Q(s,a) := \nabla_{s,a} r(s,a) + \gamma \Big(\nabla_{s'} Q(s', a'(s)) \ + \nabla_{a'} Q(s', a') \nabla_{s'} a'(s) \Big) \nabla_{s,a} f(s,a). \tag{5.7}$$

where, $a'(s')$ is the action taken at s' (which could be expressed as $\pi_\theta(s')$, or $\text{argmax}_u Q(s', u)$, or simply as a constant a'). Figure 5.1 shows the updates and

the corresponding computation graph diagramatically. Note that the R.H.S in Eq 5.7 can be obtained simply using matrix vector products and doesn't require explicit computation/storage of the jacobians. Thus, one could potentially compute the R.H.S using any auto-diff package by simply computing the target Q values (i.e., R.H.S in Eq. 6.3) and auto-diff it w.r.t the input (s,a) pair. This is useful especially for on-policy methods where each sample is used for a single or a few updates. Alternatively, for off-policy methods, where the samples are re-used multiple times for update computation, we can explicitly compute and store the jacobians in the buffer. The R.H.S can then be computed by simply plugging in the jacobians in Eq. 5.7 explicitly while computing the update.

As with most deep RL methods, instead of directly performing the updates in Eq. 5.6, we parameterize the Q function using a function approximator (say a neural network) and minimize the square of the residual in Eq. 5.6.

$$\min_{\phi} \mathbb{E}_{(s,a,s',r)\sim\mathcal{D}}\Big[\big(\nabla_{s,a}Q_{\phi}(s,a) - (\nabla_{s,a}r(s,a) + \gamma\nabla_{s,a}Q_{\bar{\phi}}(s', a'(s'))_{\perp}\big)^2\Big] \tag{5.8}$$

where, $\perp$ is the stop gradient operator denoted to mean that the terms within the corresponding parenthesis wouldn't affect the gradient of the loss. We call this the value gradient objective. In practice, we just minimize a weighted combination of the bellman residual and the above loss together. We can also derive a similar update rule for methods with state value function, $V(s)$, by simply computing the gradient of the corresponding bellman update and minimizing the gradient residual similar to Eq 5.8. In the case of actor critic methods, we can simultaneously update the policy using the current value function estimate. For example, with algorithms like DDPG, the policy is updated by taking a gradient step on the optimization problem : $\max_{\theta} Q(s, a_{\theta}(s))$.

5.4.2 Relationship with optimal control

The optimal control formulation for the above problem can be expressed as follows:

$$\begin{aligned} &\max_{s_t,a_t} \sum_{t=0}^{N} r(s_t, a_t) \\ &\text{s.t} \quad s_{t+1} = f(s_t, a_t) \end{aligned} \tag{5.9}$$

Solving the above problem using iLQR yields the following update to the co-states (multiplier variables corresponding to each dynamics constraint), λ_t, at each iteration

$$\lambda_t = (\nabla_{s_t} f(s_t, a_t))^T \lambda_{t+1} + \nabla_{s_t} r(s_t, a_t). \tag{5.10}$$

Interestingly, the co-states are also the gradients of the value function, $\nabla_{s_t} V_t(s_t)$ and the update we get in Eq. 5.10 are indeed the value gradient updates we proposed in section 5.4.1. This suggests that the update we proposed above actually just solves an amortized version of the iLQR problem where the value function and its gradients are instead represented using a function approximator (say a neural network).

5.4.3 Gradient update with approximate jacobians

The update introduced above assumes access to exact jacobians from the simulator. However, in a lot of scenarios, although we can collect experiences from the true environment (say the real world), we only have access to jacobians evaluated using an approximate simulator. However, taking inspiration from the ILC setup, we observe that we can simply use the approximate jacobians obtained from the simulator and substitute it in Eq 5.7 to obtain:

$$\nabla_{s,a} Q(s,a) := \nabla_{s,a} r(s,a) + \gamma \Big(\nabla_{s'} Q(s', a'(s)) \ + \nabla_{a'} Q(s', a') \nabla_{s'} a'(s) \Big) \nabla_{s,a} g(s,a). \tag{5.11}$$

Note that the sample (s, a, r, s') is still obtained from a rollout in the true environment. We simply evaluate the state and action jacobians $\nabla_{s,a} g(s,a)$ using the simulator g

at these samples and use it in Eq 5.11.

In practice, as mentioned in section 5.4.1, we represent the value functions and policy as neural nets and minimize a loss comprising a weighted combination of the residual of Eq. 5.11 and the bellman residual.

5.4.4 Practical Implementations

As we discussed in section 5.4.1, the value gradient update can be used with most existing reinforcement learning approaches based on bellman residual minimization. In this section, we describe one such algorithmic implementations we use in the paper, built on top of an off-policy reinforcement learning algorithm called soft actor-critic (SAC) [Haarnoja et al., 2018b].

We simply modify the critic objective in SAC to include the value gradient objective:

$$\mathcal{L}_q(\phi) = \mathbb{E}_{(s,a,s',r)\sim\mathcal{D}}\Big[\Big(Q_\phi(s,a) - \Big(r(s,a) + \gamma\big(Q_\phi(s', \pi_\theta(s')) - \alpha\log(\pi_\theta(a'|s'))\big)\Big)_\perp\Big)^2\Big] \tag{5.12}$$

$$\mathcal{L}_s(\phi) = \mathbb{E}_{(s,a,s')\sim\mathcal{D}}\Big[\Big(\nabla_s Q_\phi(s,a) - \Big(\nabla_s r(s,a) \ + \gamma\nabla_{s'}\big(Q_{\bar{\phi}}(s', a') - \alpha\log(\pi_\theta(a'|s'))\nabla_s g(s,a)\Big)_\perp\Big)^2\Big] \tag{5.13}$$

$$\mathcal{L}_a(\phi) = \mathbb{E}_{(s,a,s')\sim\mathcal{D}}\Big[\Big(\nabla_a Q_\phi(s,a) - \Big(\nabla_a r(s,a) + \gamma\nabla_{s'}\big(Q_{\bar{\phi}}(s', a') - \alpha\log(\pi_\theta(a'|s'))\nabla_a g(s,a)\Big)_\perp\Big)^2\Big] \tag{5.14}$$

$$\min_\phi \mathcal{L}_q(\phi) + \beta_s\mathcal{L}_s(\phi) + \beta_a\mathcal{L}_a(\phi) \tag{5.15}$$

where $\mathcal{L}_q(\phi)$ is the original bellman error, $\mathcal{L}_s(\phi)$ is the value gradient error for the state and $\mathcal{L}_a(\phi)$ is the value gradient error for the action, with β_s and β_a being the corresponding coefficients.

Other than the value gradient update, we keep the rest of the pipeline the same, i.e,

we use the same actor objective,

$$\max \mathcal{L}_\pi(\theta) = \mathbb{E}_{s_t \sim D}\left[\mathbb{E}_{a_t \sim \pi_\theta(.|s_t)}[\alpha \log(\pi_\theta(a_t|s_t)) - Q_\phi(s_t, a_t)]\right], \tag{5.16}$$

and perform alternating updates on the actor and the critic as in SAC.

5.5 Experiments

We present experimental evidence to show that adding the value gradient update leads to better sample efficiency both when training the policies from scratch in the true environment and while transferring policies trained in the approximate simulator to the true environment. For the experiments in this paper, we will refer to the approximate differentiable simulator as the *nominal* model and is often modelled with a simplified model with perturbed parameters. We refer to the true environment as the *real* model, which is assumed non-differentiable with the full model of the system. We aim to learn policies for the real model with as few samples from the real model as possible.

5.5.1 Tasks

Cartpole The task here is a simple swing-up task on a cartpole system. In the real model, we include viscous damping on all joints, coulomb friction at contact points (i.e the floor and cart), and a deadband in the control. None of these were modelled in the nominal model. Further, the mass of the cart and pole in the real system are 2x and 3x the masses in the nominal model respectively.

Acrobot The task here is the swing-up task with the acrobot model from [Gillen et al., 2020]. We use the same model for both the nominal and the real model except that the first link in the real model is 20% longer and has 20% higher mass.

Quadrotor We use the Quadrotor model from [Jackson et al., 2022]. The task here is to get the quadrotor from a randomly initialized point to the origin. The real model in this case additionally models the aerodynamic drag terms which are not modelled in the nominal model. The nominal model also has a parametric error of 10% on the system parameters (e.g. mass, rotor arm length, etc.).

Airplane We use the high fidelity airplane model constructed from wind-tunnel data [Manchester et al., 2017]. Similar to the quadrotor task, the task here is for the airplane to get to the origin from a distribution of randomly initialized initial positions. The real model uses the full model of the system whereas the nominal model uses a simple flat-plate wing model with linear lift and quadratic drag coefficient approximations.

HalfCheetah We use the halfcheetah model and reward functions from the Dflex simulator [Xu et al., 2022]. The nominal model here is the model from the original simulator in [Xu et al., 2022]. For the real model, we reduce damping and friction by 30% and increase the contact stiffness and contact damping coefficient by 5 times.

Hopper We use the hopper model and reward functions from the Dflex simulator [Xu et al., 2022]. The nominal model here is the model from the original simulator in [Xu et al., 2022]. For the real model, we reduce damping and friction by 20% and increase the contact stiffness and contact damping coefficient by 5 times.

5.5.2 Algorithm details

We use the SAC implementation from [Tandon] as the baseline RL algorithm with the policies and value functions represented as 2 hidden layer multi-layer perceptrons with relu and tanh nonlinearity respectively. We use the same architecture and hyperparameters for both the finetuning and training from scratch experiments. For the gradient terms, we observed that the value gradient losses lead to more stable updates when expressed as a hinge loss instead of quadratic loss. This is because

as pointed out in [Suh et al., 2022], the variance of the gradients increases with the magnitude and thus using a hinge loss reduces the sensitivity to high magnitude gradients thereby reducing the overall variance of the objective. Further, the value gradient loss coefficients β_s and β_a in Eq 5.4.4 are computed online so as to keep the ratio of gradient contributions of the different loss terms roughly constant.

$$\beta_s = \zeta_s \frac{\nabla_\phi \mathcal{L}_q}{\nabla_\phi \mathcal{L}_s}, \qquad \beta_a = \zeta_a \frac{\nabla_\phi \mathcal{L}_q}{\nabla_\phi \mathcal{L}_a}, \tag{5.17}$$

where ζ_s and ζ_a are constant hyperparameters. This heuristic has interesting connections with treating these coefficients as lagrange multipliers. We find that this also stabilizes training. These updates can however be expensive if performed at each iteration. Thus, we perform the updates every 20 iterations for most experiments except the HalfCheetah finetuning experiments (where we update them every 2 iterations).

Hyperparameters For most hyperparameters, we stick to the defaults used in [Tandon]. We use the entropy coefficient $\alpha = 0.2$ for all experiments, critic hidden size of 512 for all experiments other than cartpole and an actor hidden size of 256 for all experiments. We perform a single gradient update on the policy/value function using samples drawn from the buffer for each step taken in the environment. We use a constant value for the value gradient loss coefficients $\zeta_s = 0.5$ and $\zeta_a = 0.5$ for all experiments except the halfcheetah finetuning experiments where we use $\zeta_s = 0$ and $\zeta_a = 0.5$.

5.5.3 Training from scratch

In the first set of experiments, we compare the sample efficiency gains obtained from using the value gradient update when training from scratch in the real-model for all the tasks in 5.5.1. Specifically, we use a baseline SAC implementation as the RL algorithm and use the approximate dynamics jacobian and reward gradients obtained from the nominal model to compute the value gradient losses in Eq 5.4.4. Figure 6.1 shows the training plots comparing the two methods. SAC (Blue) in all the

plots represents the RL baseline and VG-SAC (Green) represents our modification which additionally uses the value gradient losses. We observe that across all the environments, using the value gradient objective leads to much faster convergence and in some environments (Airplane, Acrobot and Hopper) leads to superior performance.

5.5.4 Finetuning

In this section, we experiment with pre-training the policy and value function using the nominal model and then fine-tuning with the real model. Figure 5.3 shows the training plots for fine-tuning the pre-trained policies/value functions in the real model. In all the environments, we experiment with pre-training using SAC with and without the value gradient updates. We call them RLpretrain and VGpretrain respectively. Note that, while pre-training using the value gradient objective, we use the accurate gradients since we are pre-training on the nominal model. We pre-train the RL variants for longer to obtain maximum possible performance for both pre-trained policies. The pre-trained policies on Cartpole, Quadrotor, Hopper and Halfcheetah obtain similar returns of approximately 86, 650, 5300 and 4300 respectively in the nominal models for both the RL and the VG variants. However, the pre-trained RL variants in Acrobot and Airplane saturate at lower maximum returns of 84 and 296 as opposed to 109 and 304 for the value gradient variant on the nominal models.

We observe that for both sets of pre-trained networks, using the value gradient objective for fine-tuning leads to significant speedups for fine-tuning on Cartpole, Quadrotor, Airplane and Hopper tasks. However, on Acrobot, all variants struggle to improve during fine-tuning (probably due to exploration related issues). On Half-cheetah, the both fine-tuning methods perform similarly on the VG pretrained networks but VG fine-tuning outperforms the RL fine-tuning on the RL pretrained network.

Choice of pre-training algorithm Further, we observe that across all tasks, the networks pre-trained with the value gradient objective are easier to transfer, i.e, they often start with higher rewards and converge with fewer samples. Specifically,

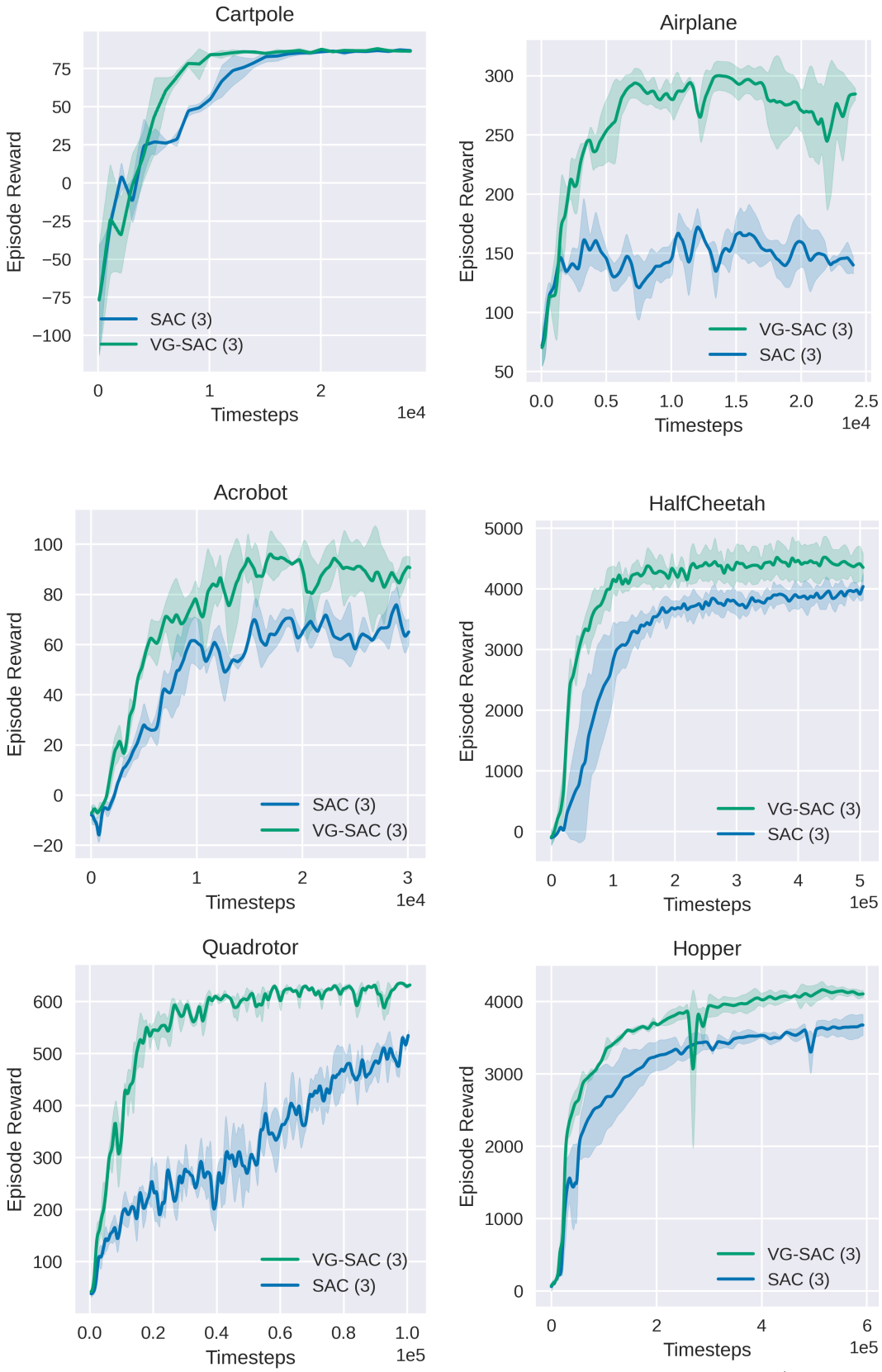


Figure 5.2: Comparison of methods using the value gradient objective (VG-SAC) v/s RL baselines (SAC) on training from scratch in the real model. SAC is a policy/value function trained from scratch in the real model using Soft actor-critic. VG-SAC is a policy/value function trained from scratch using the value gradient objective. All the plots have been evaluated using 3 random seeds.

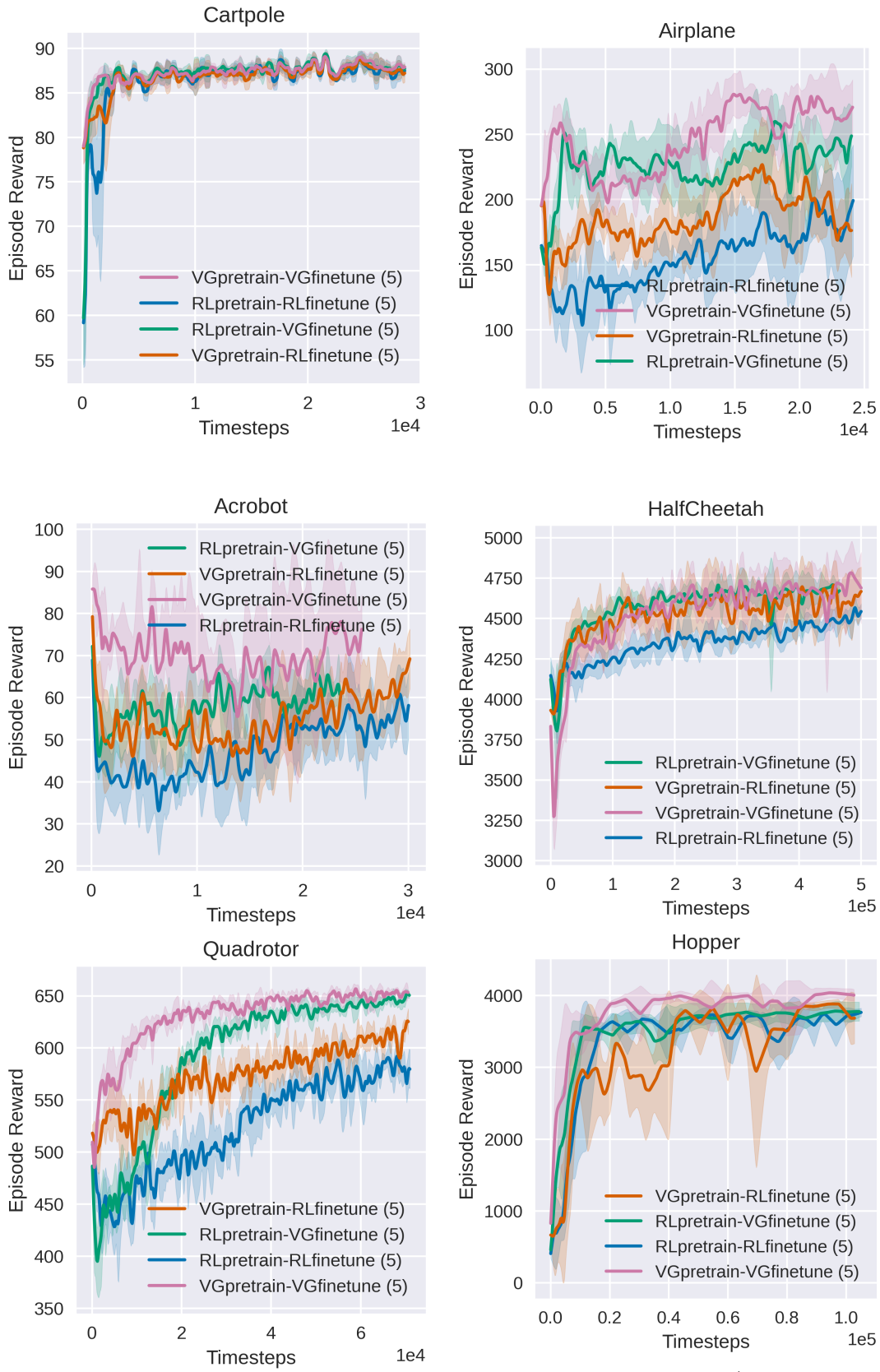


Figure 5.3: Comparison of methods using value gradient objective v/s the RL baselines (SAC) while transfering a pre-trained policy to a real model. RLpretrain indicates the networks were pretrained in the nominal model using SAC. VGpretrain indicates the networks were pretrained in the nominal model using the value gradient objective. Likewise, RLfinetune and VGfinetune indicate the networks were fine tuned on the real model using SAC and the value gradient objective respectively. All the plots have been evaluated using 5 random seeds.

comparing RLpretrain-RLfinetune vs VGpretrain-RLfinetune in Figure 5.3, we observe that although both variants are fine-tuned using the same algorithm, VGpretrain-RLfinetune consistently converges faster and starts off from a higher return than RLpretrain-RLfinetune across most environments (except hopper). The same is true when comparing RLpretrain-VGfinetune and VGpretrain-VGfinetune for most environments (except HalfCheetah where they are comparable). This indicates that using the value gradient objective even for pre-training can significantly boost transfer performance.

5.6 Discussions and Conclusion

We present an update to the gradient of the value function that can be easily incorporated in any actor critic RL algorithm. We show that the update can be computed easily using the dynamics jacobians and reward gradients obtained from a differentiable simulator even when the simulator model is just an approximation of the real model. We demonstrated the benefits of incorporating the value gradient update in a standard off-policy RL algorithm called soft actor-critic on various sim2sim tasks. We show that adding the value gradient update leads to much better sample efficiency both while training policies from scratch or fine-tuning a policy pre-trained in a perturbed environment. Moreover, we show that policies pre-trained using the value gradient update transfer faster in the new environments showing that it offers benefits for both pre-training and fine-tuning. In the future, we believe these methods need to be validated on hardware to demonstrate that the observed improvements are also observed when actually performing sim2real transfer.

Chapter 6

Leveraging parallelism with Critic Gradient Actor-Critic Methods for scaling up On-Policy RL

This chapter addresses the challenge of high variance in reinforcement learning updates, which can stem from the high dimensionality or dynamic nature of the underlying model. We present a Critic Gradient-based Actor-Critic (CGAC) method specifically designed to leverage the parallelism of modern GPU simulators. By utilizing critic gradients directly, our approach reduces variance in the policy updates, leading to more reliable and consistent learning while using the higher sampling rates to stabilize critic training. This approach improves training efficiency and performance in high-dimensional action space environments, achieving faster convergence and higher returns compared to state-of-the-art RL methods like PPO and SAC.

The contents of this chapter have been previously published at L4DC 2023 in [Gurumurthy et al., 2023b].

6.1 Introduction

Most existing on-policy algorithms (like PPO [Schulman et al., 2017], TRPO [Schulman et al., 2015a], A2C [Mnih et al., 2016] etc) which are widely used, use the likelihood ratio policy gradient to compute policy updates using a learnt value function. At first glance, given a learnt value function, this is an odd choice given that this only provides a directional derivative for the policy gradient along the sampled transition. As a result, this leads to higher variance estimates of the gradient. The other alternative, which is more common with off-policy algorithms such as DDPG [Silver et al., 2014] and SAC [Haarnoja et al., 2018a], is to directly differentiate through a learnt critic Q-function (represented as a function approximator such as a Neural Network). This provides the total derivative for the policy gradient at any sample, albeit, with the catch that the estimate might be more biased. However, with access to highly parallel simulators and a better control of the sampling distribution, it's worth considering if we could train Q-function approximations that aren't as susceptible to the bias and instability issues so that we can leverage the lower variance estimate of the policy gradients that the critic gradients could provide. Furthermore, a critic gradient based on-policy algorithm could help unify the on-policy and off-policy approaches and provide a broader framework that can arbitrarily interpolate between on-policy and off-policy updates.

Thus, in this paper, we propose a practical critic gradient based actor critic (CGAC) method, largely based on algorithms like soft actor-critic (SAC) [Haarnoja et al., 2018a] and deterministic policy gradients (DDPG), [Lillicrap et al., 2016] but for on-policy reinforcement learning. One of the primary enablers of this algorithm are the recently released highly parallel simulators [Freeman et al., 2021, Makoviychuk et al., 2021, Xu et al., 2021] which allow us to sample a large batch of environments simultaneously thereby providing a diverse set of samples to perform on-policy updates.

We observe that even when using a large batch of environments, training can still be unstable due to the interaction between the biased critic gradients and the rapidly changing policy distribution. We offer several remedies to fix these issues. First,

borrowing from the off-policy RL literature (TD3, SAC etc), we compute the Q-values as a min over two network outputs and use running averages of the Q-networks to compute target values. This helps mitigate the overestimation errors in the Q function. Second, we stabilize the policy sampling distribution by also introducing an averaged policy. The averaged policy is computed by maintaining a running average of the policy parameters and is used to perform rollouts in the environment and to compute the next action for the target value computation. Third, we provide an appropriate schedule for the entropy coefficients to balance the exploration-exploitation trade-off.

We list our key contributions here:

- We propose a practical on policy critic gradient based actor critic (CGAC) method building on previous work on DDPG and SAC.
- We demonstrate its effectiveness in several high dimensional environments such as Ant, Humanoid, SNU Humanoid and the AllegroHand with highly parallelizable simulators. We find that CGAC obtains higher returns compared to existing baselines especially on environments with high dimensional action spaces. Further, CGAC converges 4-5x faster than PPO on a muscle actuated humanoid environment (SNUHumanoid) with 152 dimensional action space.

6.2 Connections to related work

Actor critic methods for Reinforcement Learning Actor critic methods have been popular in RL for reducing the variance in the policy gradient updates originating in [Barto et al., 1983, Konda and Tsitsiklis, 1999, Witten, 1977]. Many approaches have been proposed since to further reduce the variance of these gradient estimates [Bhatnagar et al., 2007, Greensmith et al., 2004, Grondman et al., 2012, Gu et al., 2017a, Peters et al., 2008, Weaver and Tao, 2001]. One way to reduce variance, has been to use the analytical gradients from an action-value critic to estimate the policy gradients [Haarnoja et al., 2018a, Lillicrap et al., 2016, Silver et al., 2014]. This has been especially popular for off-policy learning as the alternative of using importance weighted likelihood ratio policy gradient estimates [Degris et al., 2012, Graves et al., 2021, Imani et al., 2018] have very high variance and unsuitable for large action spaces [Gu et al., 2016, 2017b]. However, most widely used on-policy methods such

as PPO [Schulman et al., 2017], TRPO [Schulman et al., 2015a], A2C[Mnih et al., 2016], still use likelihood ratio based policy gradient estimates. This is due to the biased gradient estimates from critic gradients [Gu et al., 2016, Silver et al., 2014]. Although there have been previous attempts at using critic gradient based policy gradient estimates for on-policy learning [Lillicrap et al., 2016, Silver et al., 2014], they have mostly served a didactic purpose without actually demonstrating practical merit. They point out the primary impediment as lack of diversity in samples. In this paper, we aim to demonstrate the efficacy of critic gradient based methods in large scale problems leveraging highly parallel simulators.

Reinforcement Learning in highly parallel environments Until recently most work in (deep) reinforcement learning was done with single or few parallel environment executions [Mnih et al., 2015, Schulman et al., 2017]. [Babaeizadeh et al., 2017, Clemente et al., 2017, Espeholt et al., 2018, Horgan et al., 2018, Mnih et al., 2016, Nair et al., 2015] moved towards using larger number of environments using distributed systems to simulate multiple parallel environments. However, even the largest of those [Clemente et al., 2017, Horgan et al., 2018] are still only able to scale to 256 environments on CPU clusters. Recent work [Andrychowicz et al., 2020, OpenAI, 2018] further scaled the computation to 1000s of CPU cores and use PPO [Schulman et al., 2017] as the underlying RL algorithm. More recently, we have seen development of a flurry of highly parallelizable GPU based simulators [Freeman et al., 2021, Makoviychuk et al., 2021, Xu et al., 2021] which are capable of simulating tens of 1000s of parallel simulations simultaneously on the GPU. However, most work [Allshire et al., 2021, Chen et al., 2022a, 2021a, Handa et al., 2022, Rudin et al., 2021, Xu et al., 2021] building on them have limited themselves to using standard reinforcement learning approaches such as PPO [Schulman et al., 2017], SAC [Haarnoja et al., 2018a] etc to learn control policies. In this work, we propose a practical critic gradient based on-policy RL algorithm that outperforms these standard approaches when used with highly parallel GPU-based simulators.

6.3 Preliminaries and Background

In this section, we will introduce the notation used in this paper and then discuss an on policy model free RL algorithm, Proximal Policy Optimization (PPO)[Schulman et al., 2017] and an off policy method called soft actor critic (SAC) [Haarnoja et al., 2018a].

6.3.1 Notation

We primarily focus on on-policy reinforcement learning in continuous action spaces. We will consider Markov decision processes (MDPs) with infinite horizon, but, as with most RL methods, the proposed modifications can easily be extended to finite horizon as well. The MDP state transitions are modelled as a function $f : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ and the reward using a bounded reward function $r : \mathcal{S} \times \mathcal{A} \to [r_{min}, r_{max}]$.

6.3.2 Critic gradient based off policy actor critic methods

Critic gradient based actor critic algorithms optimize the policy by directly maximizing the Q value estimates computed using the critic :

$$\max_{\theta} \mathcal{J}_{\pi}(\theta) = \max_{\theta} \mathbb{E}_{(s\sim\mathcal{D}, a\sim\pi_{\theta}(.|s))}[Q_{\phi}(s,a) + \alpha\mathcal{R}(\pi_{\theta}(a|s))], \tag{6.1}$$

where, Q_{ϕ} represents the Q function or the critic and $\mathcal{R}(\pi_{\theta}(a|s))$ here could be some regularizer (say the entropy of the policy as in SAC). We call these critic gradient based methods as these methods compute the policy gradient only using the gradient of the critic function:

$$\nabla_{\theta}\mathcal{J}_{\pi}(\theta) = \mathbb{E}_{s\sim\mathcal{D}, \epsilon\sim\mathcal{N}}[\nabla_{a}Q_{\phi}(s,a)\nabla_{\theta}a_{\theta}(s,\epsilon) + \alpha\nabla_{\theta}\mathcal{R}(\pi_{\theta}(a_{\theta}(s,\epsilon)|s))], \tag{6.2}$$

where $a_{\theta}(s,\epsilon)$ could be just the policy outputs (for deterministic policy) or could be a function applying the reparamaterization trick [Kingma and Welling, 2014] to compute the action using the policy output distribution (for stochastic policies) and

a randomly sampled noise ϵ from a fixed distribution $\mathcal{N}$.

Critic gradient based off-policy actor critic methods learn the above Q function, Q_ϕ by minimizing the bellman residual over samples drawn from a replay buffer $\mathcal{D}$ of stored samples. SAC and DDPG would be popular examples of such approaches. The bellman residual minimization is given as follows:

$$\min_{\phi} \mathbb{E}_{(s,a,s',r)\sim\mathcal{D},\epsilon\in\mathcal{N}}[(Q_\phi(s,a) - (R(s,a,s') + \gamma Q_{\bar{\phi}}(s', a_\theta(s',\epsilon))_\perp)^2] \tag{6.3}$$

where, $a_\theta(s',\epsilon)$ is obtained using the current policy estimate $\pi_\theta(.|s')$. For SAC, $R(s,a,s') = r + \alpha\mathcal{H}(\pi_\theta(.|s'))$ while DDPG just uses $R(s,a,s') = r$. $\perp$ is the stop gradient operator denoted to mean that the terms within the corresponding parenthesis wouldn't affect the gradient of the loss. Further, $\bar{\phi}$ are the parameters of the target Q function usually computed as a moving average of ϕ.

While these critic gradient based methods are very popular in off-policy RL, they are seldom used for on-policy RL. This is often attributed to critic gradient providing a biased estimate Silver et al. [2014] of the policy gradient which ends up being particularly problematic for the on-policy case, where the rapidly changing input distribution for the value function can further bias and destabilize training.

6.3.3 Advantage actor critic methods

Most popular on-policy RL approaches instead use the advantage to compute the likelihood ratio policy gradient. Examples include PPO [Schulman et al., 2017], TRPO [Schulman et al., 2015a], A2C [Mnih et al., 2016] etc. They typically learn a value function, V_ψ, by minimizing the error between $V_\psi(s)$ and a single/multi-step estimate of the discounted return, $R(s,a)$, on the current policy distribution ρ_π. Specifically, $R(s,a)$ is computed using :

$$R(s,a) = A_V^{(\lambda,\gamma,h)}(s,a) + V_\psi(s), \tag{6.4}$$

where $A^{(\lambda,\gamma,h)}(s,a)$ is the advantage estimate computed over a horizon h, with discount factor γ and a parameter λ. It can be computed using the generalized advantage

estimate [Schulman et al., 2015b]:

$$A_V^{(\lambda,\gamma,h)}(s_t, a_t) = \sum_{l=0}^{h} (\gamma\lambda)^l \delta_{t+l}^V, \quad \text{where} \quad \delta_{t+l}^V = r_{t+l} + \gamma V_\psi(s_{t+l+1}) - V_\psi(s_{t+l}). \tag{6.5}$$

Setting $\lambda = 1$ gives us the traditional advantage estimates used in A2C, TRPO etc. The advantage estimates are also used to compute the policy gradients as follows:

$$\nabla_\theta p_\theta(s, a) = \mathbb{E}_{(s,a)\sim\rho_\pi}[A_V^{(\lambda,\gamma,h)}(s, a)\nabla_\theta \log(\pi_\theta(a|s))]. \tag{6.6}$$

PPO and TRPO use slight variations of Eq 6.6 to additionally account for the mildly off-policy samples. We observe that the policy gradient estimate in Eq 6.6 computes a directional derivative along the sampled trajectory. As a result, advantage based estimate of the policy gradient ends up being higher variance than the critic gradient based estimate from Eq 6.1 [Gu et al., 2016, 2017b]. However, they are also less biased.

6.4 Method

We combine aspects of the above two approaches and propose a critic gradient based actor critic approach for on-policy RL. We expect the resulting algorithm to provide low variance policy gradients and hence converge faster especially on environments with high dimensional action spaces. We will first describe a naive implementation and then discuss the issues faced by this naive implementation. We will then discuss a few fixes that alleviate these problems.

6.4.1 Critic gradient based actor critic for on-policy RL

We parameterize the Q-function and policy using function approximators such as neural networks. The policy network outputs the mean and standard deviation to parameterize a gaussian distribution. As with other critic gradient based approaches,

we maximize the following objective to optimize the policy parameters

$$\max_{\theta} \mathcal{J}_{\pi}(\theta) = \max_{\theta} \mathbb{E}_{(s\sim\mathcal{D}, a\sim\pi_{\theta}(.|s))}[Q_{\phi}(s,a) - \alpha \log(\pi_{\theta}(a|s))], \tag{6.7}$$

where we regularize the entropy of the policy outputs to encourage appropriate exploration. We learn the Q function by supervising it using the single/multi-step discounted return estimate R,

$$\min_{\phi} \mathcal{J}_{\mathcal{Q}}(\phi) = \min_{\phi} \mathbb{E}_{(s\sim\mathcal{D}, a\sim\pi_{\theta}(.|s))}\left[||Q_{\phi}(s,a) - R(s,a)_{\perp}||^2\right], \tag{6.8}$$

$$\text{where,} \quad R(s,a) = A_{Q}^{(\lambda,\gamma,h)}(s,a) + Q_{\bar{\phi}}(s,a), \tag{6.9}$$

where $A_{Q}^{(\lambda,\gamma,h)}(s,a)$ is a slightly modified generalized advantage estimate [Schulman et al., 2015b] adapted for the Q-function :

$$A_{Q}^{(\lambda,\gamma,h)}(s_t,a_t) = \sum_{l=0}^{h} (\gamma\lambda)^l \delta_{t+l}^{Q}, \quad \text{where } \delta_{t+l}^{Q} = r_{t+l} + \gamma Q_{\bar{\phi}}(s_{t+l+1}, a_{t+l+1}) - Q_{\bar{\phi}}(s_{t+l}, a_{t+l}). \tag{6.10}$$

Note that we only use $A_{Q}^{(\lambda,\gamma,h)}(s_t,a_t)$ to compute the returns for value supervision and not to compute the policy gradient estimate.

Like the off-policy counterparts [Fujimoto et al., 2018, Haarnoja et al., 2018a], we make use of two Q-functions to mitigate the overestimation issues caused by the above updates. Specifically, we learn two Q-functions, Q_{ϕ_1} and Q_{ϕ_2}, independently to optimize $\mathcal{J}_{Q}(\phi_i)$. The Q-values in the value targets in Equation 6.9 and the policy objective in Equation 6.7 are then computed by taking the minimum of the two Q-functions.

$$Q(s,a) = \min(Q_{\phi_1}(s,a), Q_{\phi_2}(s,a)) \tag{6.11}$$

Further, we also maintain a moving average of both the Q-functions as is common with off-policy methods and use them while computing the value targets. This also helps mitigate the overestimation issues. Like the on-policy counterparts [Mnih et al., 2016, Schulman et al., 2015a, 2017], our method alternates between collecting experience from the parallel environments with the current policy and using these recent collected experiences to update the Q-functions and policy by optimizing

Equations 6.7 and 6.8.

This framework is particularly appealing given that off-policy samples can be easily incorporated in the updates by performing a DDPG/SAC type update step and can nicely complement the on-policy updates we perform. Although, we do not explore this in our paper, we believe this ability to conveniently combine off-policy and on-policy updates is one of the primary benefits of this method. Moreover, as discussed previously, we expect this critic gradient based approach for policy optimization to converge faster due to the lower variance of the estimated policy gradients.

However, this naive implementation can be unstable or saturate at lower returns as shown in the ablation experiments in 6.5.2. This is due to 1) the lack of diversity of the on-policy experiences used for the updates (leading to biased updates) and 2) the instabilities arising from the interaction between the biased critic gradients and the rapidly changing policy distribution. In the next section, we explore these in more detail and propose some candidate solutions to mitigate them.

6.4.2 Causes for instability and corresponding fixes

State Distribution Diversity One of the motivations for using a replay buffer and performing off-policy RL in the seminal DQN paper [Mnih et al., 2015] was to avoid the problem of correlated samples across timesteps biasing the gradients and value function. However, with the development of highly parallel simulators [Freeman et al., 2021, Makoviychuk et al., 2021, Xu et al., 2021], this problem can be significantly mitigated by sampling from a large batch of environments in parallel. With a decent choice of exploration strategy one can ensure that this sampling distribution is diverse enough to ensure a diverse set of states at each sampling step.

To encourage an appropriate amount of exploration throughout training we simply add a negative penalty on the entropy of the policy outputs as part of the policy objective as in SAC as shown in Eq 6.7. We use a slightly modified automatic entropy tuning procedure from SAC to tune the entropy coefficient α. We update α

to maximize

$$\max_{\alpha} \mathcal{J}_{\alpha}(\alpha) = \max_{\alpha} \left(\log \pi_{\theta}(a|s) + \beta n\right) \log \alpha \tag{6.12}$$

where n is the action space size and β is a hyperparameter. The value of β here chooses the target entropy for the policy that we wish to optimize towards using the entropy regularization. SAC sets this hyperparameter to a fixed value of $\beta = 1$. However, as the policy performance gets better, we typically wish to encourage exploitation. To that end, we propose to gradually decrease the target entropy for policy. Specifically, we use simple linear schedule for β based on the current reward values

$$\beta = \frac{\beta_{final} - \beta_{init}}{\textit{peak reward} - \textit{initial reward}} * (\textit{current reward} - \textit{initial reward}) + \beta_{init} \tag{6.13}$$

where β_{final} and β_{init} are the final and initial values of β we desire (with β_{final} ¿ β_{init}). *peak reward* and *initial reward* are rough estimates of peak and initial average reward we expect to get in the environment. This allows us to gradually decrease the target entropy for the policy as the policy performance improves during training encouraging further exploitation. Further, while solving Eq 6.12, we found it helpful to ensure that α monotonically decreases during training. Thus, we clamp the update on α:

$$\alpha = \min(\alpha_{prev}, \alpha_{prev} + \eta \nabla_{\alpha} \mathcal{J}_{\alpha}(\alpha_{prev})).$$

to ensure a monotonic decrease in it's value. Intuitively, this helps 1) prevent the policy from arbitrarily exploring regions of the state-action space outside the Q-function's support in the middle of training due to an increase in entropy penalty, 2) lower entropy penalty towards the end of training encourages exploitative behavior in the policy leading to lower variance updates.

Indeed, we observe in the ablation experiments in 6.5.2 that increasing the number parallel environment executions has a significant impact on the policy performance, both in terms of convergence speeds and the peak returns obtained. Furthermore, the α schedule leads to stable convergence and higher final returns. However, this doesn't completely resolve the issue.

Policy-Value Coupling We observe that, in some cases, a few bad gradient updates can lead to the policy performance suddenly deteriorating and collapsing. This results in a large fraction of parallel episodes abruptly coming to an end and the environments resetting to their initial states. As a consequence a large fraction of the parallel environments at that time step (and a few succeeding steps) collapse to very similar set of states. The sudden change in the input distribution and increase in correlated samples leads to a further deterioration on the value updates consequently going down a spiral of successively worse updates for both the policy and the value function. This issue is less pronounced in existing on-policy approaches such as PPO and TRPO as they are not fully on-policy and use trust region constraints to mitigate rapid changes in policy.

We propose a much simpler solution for this issue. We maintain a running average of the Q-functions to compute value targets, and a running average of the policy to perform rollouts in the environment and compute actions for the target value computation in Equation 6.8 and 6.9. Using these running averages of the policy and Q function for rollouts and target value computations ensures that a few bad gradient steps on the original policy and value function don't derail them. This leads to improved stability and prevents the policy distribution from changing too rapidly while keeping our method largely 'on-policy'.

6.5 Experiments

In this section, we present experimental evidence to demonstrate the effectiveness of CGAC especially in environments with large action spaces. We compare with proximal policy optimization (PPO) [Schulman et al., 2017] and soft actor critic (SAC) [Haarnoja et al., 2018a] based on episode returns obtained at convergence or end of training and the time taken to convergence.We also present ablation experiments to analyze and understand where the gains come from. Finally, we also present some initial promising experiments combining off-policy and on-policy updates in CGAC.

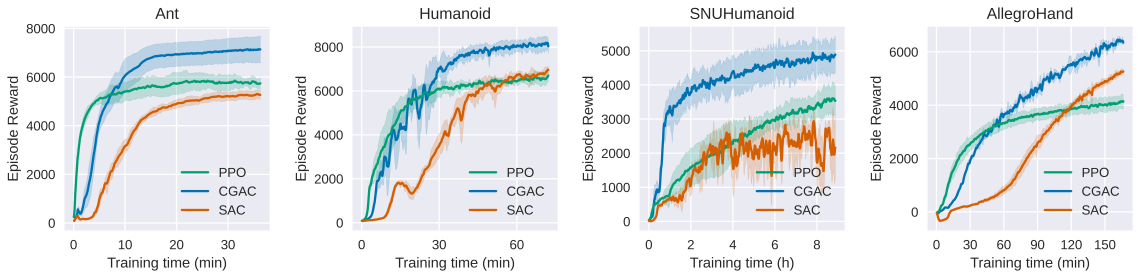


Figure 6.1: Comparison of on-policy critic gradient based actor critic (CGAC) with PPO and SAC on various high dimensional continuous control tasks.

Tasks We test the above methods on 4 tasks, namely: Ant, Humanoid, SNUHumanoid and AllegroHand. For the first 3, we use the Dflex simulator[Xu et al., 2021] and use IsaacGym[Makoviychuk et al., 2021] for AllegroHand. The environments have an action space of 8, 21, 152 and 16 dimensions respectively and observation space of 37, 76, 71 and 88 respectively.

Hyperparameters We use the hyperparameters provided in the official Isaacgym[Makoviychuk et al., 2021] and Dflex[Xu et al., 2021] repositories for PPO and SAC for all the environments while using the RL-games [Makoviichuk and Makoviychuk, 2022] implementation of the said algorithms. For CGAC, we use largely the same hyperparameters as the SAC implementation and list the ones that differ in Table 6.1. Although slight improvements in performance are possible in each environment with a different hyperparameter setting, we aim to keep the hyperparameters as similar as possible across environments to demonstrate the robustness to those hyperparameters.

Table 6.1: CGAC hyperparameters for each task

Hyperparameter	Ant	Humanoid	SNUHumanoid	AllegroHand
Policy Architecture	[256, 256]	[256, 256]	[256, 256]	[512, 256]
Critic Architecture	[1024, 1024, 256]	[1024, 1024, 256]	[512, 512, 256]	[1024, 512, 256]
Activation	ELU	ELU	ELU	ELU
Num Actors	4096	4096	4096	16384
Num Critic updates per step	2	2	6	2
(Start) Learning Rate	0.002	0.002	0.002	0.002
Critic avg coeff τ_Q	0.005	0.005	0.005	0.05
Policy avg coeff τ_π	0.001	0.001	0.001	1
final targ ent coeff β_{final}	3.5	7.5	3.5	5
init targ ent coeff β_{init}	0.2	0.2	0.2	0.2
GAE horizon length	32	32	8	8
Batch Size	4096	4096	4096	32768

6.5.1 Comparisons

Figure 6.1 shows the average episode return obtained by each method on the 4 continuous control tasks as training progresses. The averages are computed using 5 runs of each algorithm each with a different random seed. We observe that across all the tasks, CGAC outperforms PPO and SAC in terms of the peak episode returns. The difference is especially stark on SNUHumanoid which has a much higher dimensional action space of 152 dimensions. Further we observe that CGAC also converges faster than the baselines in environments with high dimensional action spaces. Again the difference is especially stark on SNUHumanoid, where CGAC converges within 1-2 hours whereas PPO doesn't converged even after 9 hours of training.

Further, we observe some peculiar convergence behaviors which contrasts PPO and CGAC. While CGAC is slower to converge in the initial stages of training, it reaches peak returns much faster than PPO, whereas PPO converges very quickly initially but converges very slowly towards the end. This is simply a consequence of the fact that the policy gradients estimated by CGAC are highly biased and of very poor quality in the initial stages of training when the Q-function is poorly trained but improve quickly in the later stages of training as the Q-function quality improves.

6.5.2 Ablations and analysis

In this section, we perform various ablations to understand the contribution of the various design decisions made in the algorithm.

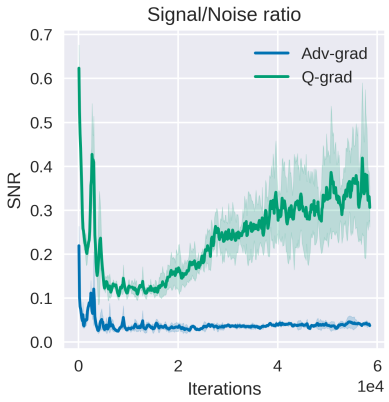


Figure 6.2: Signal to noise ratio of the policy gradients computed using the likelihood ratio estimate $A_V^{(\lambda,\gamma,h)}(s,a)\nabla_\theta log\pi_\theta(a|s)$ and the Q function gradients $\nabla_\theta Q_\phi(s,a_\theta)$ over the on-policy samples throughout training.

Variance analysis Figure 6.2 shows the signal to noise ratio of the policy gradients computed using the likelihood ratio estimate $A_V^{(\lambda,\gamma,h)}(s,a)\nabla_\theta log\pi_\theta(a|s)$ and the Q function gradients estimate $\nabla_\theta Q_\phi(s,a_\theta(s))$ over the on-policy samples throughout training for the Humanoid Environment. Note that for the

likelihood ratio estimate we train a separate value function V_ψ concurrently during training and use it to compute gae advantage estimates $A_V^{(\lambda,\gamma,h)}$. We observe that, as expected, the Q function gradients have a higher signal to noise ratio compared to the advantage based estimates. While it is not possible to conclude from this that the Q function gradients are 'better' as it's difficult to accurately measure the bias of those estimates, we could say that there's some reason to hope that if we mitigate the bias issues in the Q function gradient estimates, we could hope for a better performance. Our modifications to the vanilla algorithm aim to do exactly that by ensuring we use a diverse input distribution to train the networks and that the policy distribution changes smoothly.

Ablations We inspect the impact of various aspects of the algorithm in this section. Figure 6.3 shows the corresponding training plots for experiments on the Humanoid and Ant environments.

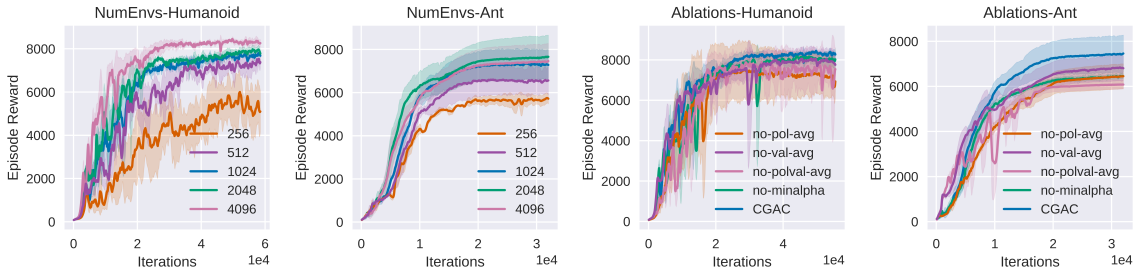
Figure 6.3a and 6.3b shows the training plots with different number of parallel environment runs. We find that the large number of parallel runs enabled by the highly parallel GPU based simulators are immensely useful as the performance improves significantly with the number of parallel runs both in terms of convergence speeds and the average returns at the end of training. This is one of the biggest contributing factors enabling the on-policy nature of the algorithm. To put it in context, before the availability of these GPU based simulators, DDPG/SAC based algorithms using the maximum number of parallel runs was [Horgan et al., 2018] with just 64 parallel runs for the continuous control experiments.

Further, we experiment with changing various aspects of the algorithm one at a time to illustrate their impact in Figures 6.3c and 6.3d. We observe that removing the value function averaging (no-val-avg) to compute the target values has a small effect on the training stability and performance as long as we perform policy averaging. This is an interesting contrast with off-policy methods, where the value function averaging plays a crucial role in combating the overestimation issues. However, we observe that removing policy averaging (no-pol-avg) alone has a significant effect on the stability and convergence speed of the method especially in the initial stages of

training. Further removing both value and policy averaging (no-polval-avg) leads to a further increase in instability and degradation of performance as it worsens the coupling between a update quality and the policy distribution. As a result, in the initial stages of training (when the Q-function quality is bad, resulting in bad quality updates), the algorithm becomes very unstable. The sudden drops in the episode rewards seen in the plots happen when a few bad updates deteriorate the policy resulting in a bunch of runs suddenly collapsing leading to a temporary collapse in the policy distribution.

Further, we also observe that the exploration parameter alpha often affects the exploration behavior of the agents. As shown in figure , ensuring a strictly monotonic decrease in the alpha values throughout training leads to better final performance as well as more stability during training.

We note that although these phenomenon might not be observed universally across all environments, we do observe that incorporating these changes more often than not do help to improve stability and peak performance and make the algorithm more robust to hyperparameter values. They become especially important as we go to environments with larger action spaces.



(a) (b) (c) (d)

Figure 6.3: (a) and (b) : number of parallel environment runs. (c) and (d) : Ablations with switching on/off various other aspects of the algorithm to understand their impact.

6.6 Discussions and Conclusion

We present a critic gradient based on-policy reinforcement learning algorithm that outperforms popular on-policy and off-policy reinforcement learning algorithms such

as PPO and SAC. We show that one of the primary factors enabling the superior performance is the availability of highly parallelizable GPU based simulators which allow us to run a large number of simulation runs in parallel. We also propose several other improvements to the naive version of the algorithm such as maintaining an averaged policy and value averaging for the policy rollouts and target value computations, and, a specific schedule on the entropy coefficient to manage the exploration-exploitation tradeoff. With these changes incorporated, we show that CGAC converges to much higher returns compared to PPO and SAC across all the environments. Further, CGAC also converges much faster than the baselines especially on environments with high dimensional action spaces.

In future work, we believe it's important to explore using off-policy experiences to complement the on-policy experiences for more stable implementations. Given the suitability of this algorithm to incorporate off-policy experiences, we believe that this would be a promising avenue of future research. Further, we also believe that the critic based policy gradient estimates are complementary to the advantage based estimates in algorithms like PPO. Thus, we believe exploring interpolations of these objectives such as in [Gu et al., 2016, 2017b] can also be important avenues of future work.

Chapter 7

VINSat: Solving the Lost-in-Space Problem with Visual-Inertial Navigation

In this chapter, we introduce VINSat, an end-to-end visual-inertial navigation system for orbit determination (OD) of nanosatellites. VINSat addresses the 'lost-in-space' problem by combining data from a low-cost RGB camera and an inertial measurement unit (IMU) to identify known landmarks and determine the satellite's full state. Using a SOTA deep network based detection algorithm for landmark detection and a batch nonlinear least-squares state estimator, VINSat determines kilometer-level accurate satellite localization within a few hours, significantly outperforming traditional ground-based methods. We validate VINSat through simulations using real nadir-pointing imagery, showing that 85% of the simulated satellites are localized to within 5 km within 6 hours, demonstrating faster and more precise localization compared to ground radar.

The contents of this chapter have been previously published at ICRA 2024 in [McCleary et al., 2024].

7.1 Introduction

Nanosatellites, including CubeSats, have been deployed in rapidly increasing numbers in recent years Kulu [2022]. They are compact, cost-effective satellites that have opened up new possibilities for space research, technology testing, and educational initiatives. Their modular design and affordability have democratized access to space, allowing a wider range of organizations and individuals to participate in space exploration. They have found applications in diverse areas such as agriculture You et al. [2017], land use classification, and disaster response Shao et al. [2020].

Current methods for orbit determination (OD) of nanosatellites, such as Global Positioning System (GPS) receivers, radio ranging, and ground-based radar are large, expensive, imprecise, or time consuming. These limitations are underscored by data from the European Space Agency (ESA) that show a considerable fraction of CubeSats remain unidentified more than 250 days after launch Letizia [2019].

Nanosatellites would greatly benefit from a solution to the "lost-in-space" problem, in which a spacecraft needs to determine its orbit relying solely on onboard computing and sensing with no prior state knowledge. This paper presents Visual-Inertial Navigation for Satellites (VINSat), a comprehensive solution to this problem that can determine a satellite's pose with kilometer-level accuracy within a few hours using only a low-cost camera, Inertial Measurement Unit (IMU), and on-board processing. For this paper, we will focus on a nadir-pointing satellite with an assumption that attitude is known and controlled.

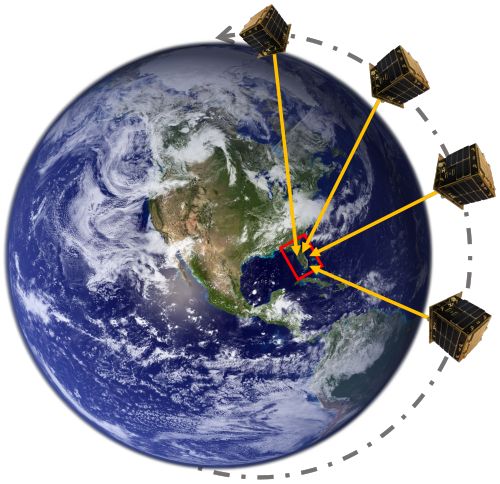


Figure 7.1: VINSat solves the "lost-in-space" problem by determining a satellite's orbital parameters. It combines low spatial resolution image sensing and machine-learning inference with batch least-squares estimation and an accurate dynamics model to achieve kilometer-level position accuracy.

VINSat is a fast, cost-effective method that is compatible with the size, weight, and

power limitations of nanosatellite hardware. VINSat addresses the challenge of OD by employing a computer vision model to identify landmarks on the Earth's surface and incorporating this information into a batch least-squares problem Robertson and Lee [1995], Simon [2006], aligning the detected landmarks with their known 3D positions while accounting for the orbital dynamics between consecutive time steps. Our main contributions include:

- A complete end-to-end visual-inertial navigation pipeline for OD onboard nadir-pointing small satellites using only low-cost camera sensors and onboard computation.
- A large dataset of real and synthetic Earth imagery collected from satellites for training vision-based navigation algorithms for space applications.
- A thorough evaluation based on Monte Carlo simulation of more than 500 low Earth orbit (LEO) deployments demonstrating that VINSat localizes 85% of satellites to within 5 km in just 6 hours; substantially faster and more precise than using ground-based radar.

This paper proceeds as follows: Section 7.2 discusses previous work on OD. Section 7.3 describes each subsystem making up the end-to-end system in detail. Section 7.4 then presents an evaluation framework and the results obtained with the system. Finally, Section 7.5 summarizes our conclusions and directions for future work.

7.2 Connections to related work

In this section, we provide a brief overview of the existing approaches to OD.

7.2.1 GPS Receivers

GPS receivers are the standard for precise OD Kang et al. [2006], Skinner et al. [2022], van den IJssel et al. [2015], but are ill-suited for nanosatellites, such as CubeSats or PocketQubes Denby et al. [2022], Radu et al. [2018], due to their size and cost.

A space-rated version of the NovAtel OEM719 GPS receiver noa is commonly used in CubeSats due to its relatively small size, low cost, and ability to operate at the extreme speeds (roughly 7.5 km/s) of a satellite in LEO. These receivers cost around $5000. Modules that use these receivers to perform OD on-orbit cost even more. One such system, gns, uses a NovAtel OEM719 for position data to perform OD and costs $10k-$19k. Standard GPS receivers for use on Earth are restricted to speeds less than 515 m/s and altitudes less than 18 km due to limits placed by the Coordinating Committee for Multilateral Export Controls (CoCom), its successor the Wassenaar Arrangement, and the Missile Technology Control Regime (MTCR) mtc.

7.2.2 Ground Radar

An alternative to GPS is radio ranging from ground stations or radar Caldwell [2021], Skinner et al. [2022], which can take weeks to months to determine a satellites orbit after launch, and may even fail to identify a satellite in a cluster Caldwell [2021]. Radio ranging from ground stations requires significant infrastructure, and positions from radar typically have errors of 20 km pla. Table 7.1 compares the existing commonly used methods to our approach.

Table 7.1: Comparison of Orbit Determination (OD) Methods

Property/Method	**GPS OD**	**Ground Radar**	**Visual OD**
Largest Dimension	96 mm gns	Off Satellite	< 50 mm
Mass	109 g gns	Off Satellite	< 15 g
Power	1-2 W noa	Off Satellite	< 5 W
Cost	$\sim$ \$10 k gns	\$0 – 10 k	$<$ \$100
OD Time	Secs Yang et al. [2016]	Wks-Mos ESA	Secs-Hrs
Precision	1.5 m Yang et al. [2016]	$\sim$ 10 km pla	$\sim$ 1 km

7.2.3 Visual Methods

Prior work on visual satellite navigation such as Straub and Christian [2015] and Shockley and Bettinger [2021], obtains coarse satellite position estimates using only visual

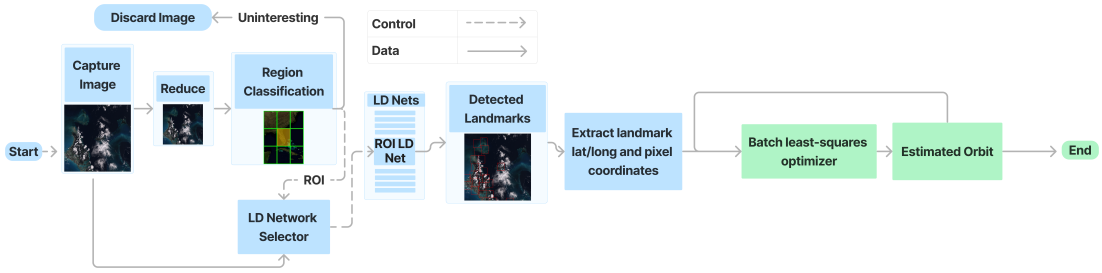


Figure 7.2: End-to-end block diagram of VINSat pipeline. The vision block is highlighted in blue and the state estimation block is highlighted in green. Captured images are first classified into regions of interest (ROIs). These ROIs are used to feed the captured image to the appropriate landmark detection (LD) networks. The identified landmarks, their positions in the image, and their associated confidences are passed to the batch least-squares optimizer. The batch least-squares optimizer uses the information to estimate the orbit of the satellite.

inputs from cameras and classical computer vision keypoint-matching techniques, such as SIFT Lowe [1999], FLANN Muja and Lowe [2014] and RANSAC Fischler and Bolles [1981]. These approaches have several limitations, including lack of robustness to clouds, the use of pre-extracted coastlines for localization, and localization errors of a few degrees of latitude and longitude.

7.3 System Design

This section describes the architecture of the VINSat system. We provide a high-level overview, followed by a detailed description of each subsystem.

7.3.1 System Overview

The VINSat system, as depicted in Fig. 7.2, has two main components: an image-processing subsystem responsible for extracting Earth landmarks from captured imagery and a batch least-squares optimization solver that calculates the satellite's orbit based on these landmarks.

Initially, images are captured by a camera at a resolution of 4608×2592 pixels. Identifying landmark correspondences in the captured images is challenging, particularly without prior knowledge of the satellite's pose, and is further compounded by the computational constraints imposed by a nanosatellite's limited power and processing capacity.

VINSat geolocates captured imagery by matching image features to landmarks on Earth. First, the system identifies the coarse geographic region over which the satellite is orbiting. Second, it matches ground landmarks with known locations to pixels in the image. VINSat performs region identification using a Region Classification (RC) Network. An RC network is a deep neural network that is trained to process each captured image at a downscaled resolution of 640×360 pixels to identify the region of Earth that the image depicts. Additionally, the RC network eliminates uninteresting images, such as those containing only clouds or ocean. These outputs are subsequently used to route the full-resolution images to specialized landmark detection (LD) deep neural networks, each trained for a specific region. The LD networks produce a mapping from pixel coordinates to landmarks with known location coordinates on Earth. These coordinates are the input to a batch optimization solver that produces an estimate of the satellite's orbit. Further details of each subsystem are elaborated upon in the sections that follow.

7.3.2 Region Classification

The RC network takes an image as input and produces the regions that the image most likely depicts as output. The purpose of the RC network is to narrow the scope of the later search for landmarks to a small set of regions on Earth, rather than all of Earth.

The RC network's output classes are region identifiers corresponding to regions defined in the Military Grid Reference System (MGRS), a NATO international standard for locating points on the Earth. The largest regions of the MGRS divide the Earth into a grid delineated by 22 North-South regions and 60 East-West regions. Each region is typically six degrees of longitude by eight degrees of latitude, with some variation

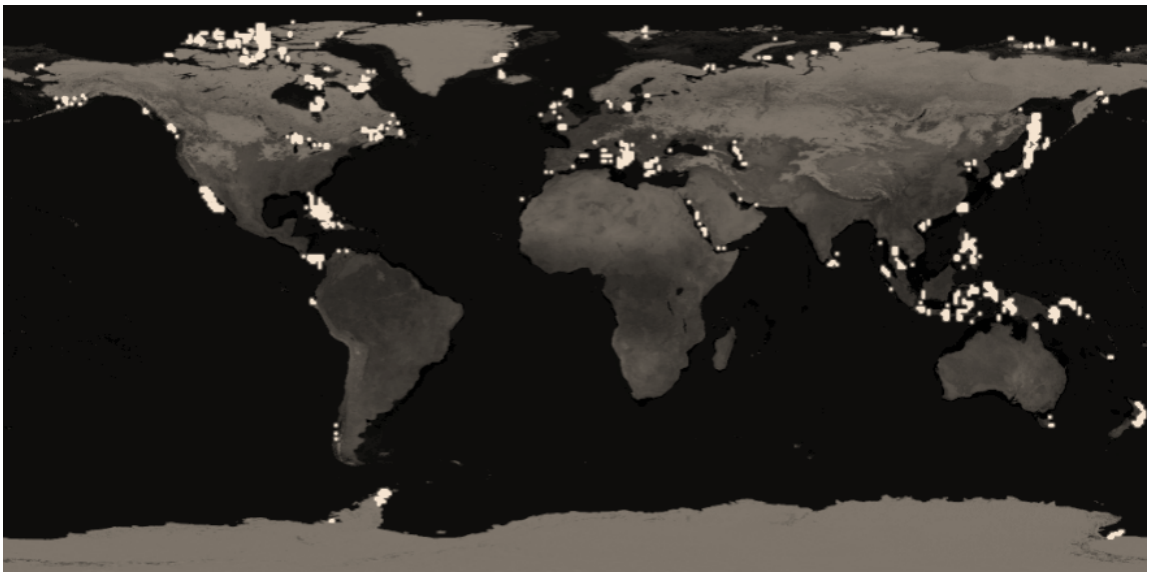


Figure 7.3: Saliency map from which regions of interest were selected. Brighter areas exhibit high saliency. These areas were used to select regions of interest for the region classification and landmark detection networks.

near the poles.

The RC network does not consider all regions in the MGRS, instead focusing on the most *salient* 16 regions. VINSat computes the saliency of a region by computing the cross-correlation of the NASA Blue Marble imagery data available for that region. Fig. 7.3 shows a map of the computed saliency of points on Earth. Based on a Monte Carlo simulation of 1000 typical random nadir-pointing LEO satellites, the average time for a CubeSat to pass over one of these regions is about 63 minutes.

The RC network is based on the EfficientNet-B0 architecture, a highly efficient neural-network model suitable for online inference on each captured image. To adapt it for our setting, we replace its output layer with a sigmoid activation function for region detection.

7.3.3 Landmark Detection

Each LD network takes an image as input and produces a set of landmarks contained within the image and their pixel locations as output. VINSat selects landmarks based

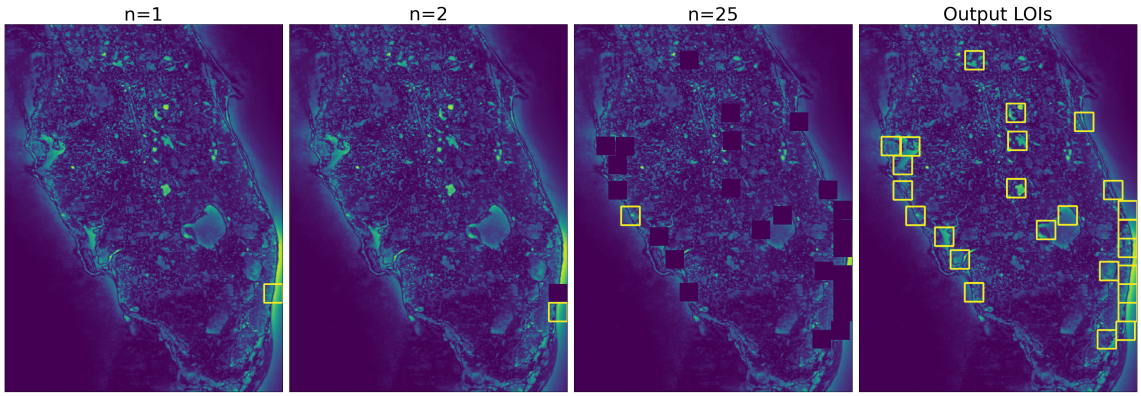


Figure 7.4: Landmarks from the saliency map of an individual region. Each yellow rectangle is a 25×25 km bounding box of a salient landmark. The saliency-based landmark-selection process determines the landmark detection network's classes for each region. 500 landmarks are selected for each region of interest.

on saliency of features within a region of interest (ROI). We use OpenCV's saliency API on NASA Blue Marble image data for each of our 16 regions of interest. We sequentially select the top 500 most salient boxes in each region of varying sizes between 5 km and 50 km. An example of this process is shown for boxes of 25×25 km in Fig. 7.4.

VINSat's LD network is based on the YOLOv8s object detection model, trained to produce bounding boxes around landmarks within an image. We convert landmarks and detection bounding boxes to points (which is needed for OD) by using their center point.

7.3.4 Orbit Determination

The primary objective of the OD pipeline is to use measurements from the image processing subsystem to generate accurate estimates of the satellite's state, which includes its position $\mathbf{r}$, and velocity $\mathbf{v}$. The estimation problem is formulated as a batch least-squares optimization problem Robertson and Lee [1995], Simon [2006], aiming to minimize the residuals arising from satellite dynamics across consecutive time steps and the discrepancies between projected and observed landmarks on the

camera frame.

The inputs to the system are the 3D locations of the detected landmarks in Earth-centered inertial (ECI) coordinates $\mathbf{p} = (x, y, z)$ and the pixel position of the corresponding detections in the image are $\mathbf{w} = (u, v)$.

The dynamics error term, ψ_t, is a summation of two key components: squared errors on the predicted and actual position and velocity. The predicted position ($\hat{\mathbf{r}}$) and velocity ($\hat{\mathbf{v}}$) are obtained by rolling out the orbital dynamics from the previous state.

$$\psi_t = \|\hat{\mathbf{r}}(\mathbf{r}_{t-1},\mathbf{v}_{t-1}) - \mathbf{r}_t\|^2_{Q_r} + \|\hat{\mathbf{v}}(\mathbf{r}_{t-1}, \mathbf{v}_{t-1}) - \mathbf{v}_t\|^2_{Q_v}, \tag{7.1}$$

where Q_r and Q_v are weight matrices corresponding.

The second major component in the batch optimization problem is the camera projection error term, ϕ_m. Given the satellite's estimated position $\mathbf{r}$ and assuming nadir orientation $\mathbf{q}_n$, along with the camera matrix K, we can project 3D landmarks $\mathbf{p}$ to their expected pixel positions $\hat{\mathbf{w}}$ in the camera frame. The error term ϕ_m quantifies the discrepancy between these predicted pixel positions and the actually observed positions $\mathbf{w}$:

$$\phi_m = \|\hat{\mathbf{w}}(\mathbf{r_m}, K, \mathbf{p_m}) - \mathbf{w}_m\|^2_R, \tag{7.2}$$

where R is a weight matrix. One challenge with this error term is its sensitivity to outliers, particularly when the landmark measurements are noisy. Thus, instead of using a constant R, we compute it as proposed in Barron [2019b] to obtain an adaptive kernel :

$$R_k = c^2 \left(\frac{(\epsilon^p_k/c)^2}{|\alpha - 2|} + 1 \right)^{1-\frac{\alpha}{2}}, \tag{7.3}$$

where R_k is the k-th diagonal term in R, ϵ^p_k is the corresponding residual term, and α and c are hyperparameters that control the shape and scale of the kernel, respectively. We set c to the median of the residuals to automatically adapt the scale of the kernel to the specific problem. We compute $\alpha = \max(2 - 2 * i/5, -1)$, as a function of the

Algorithm 1 Batch Least-Squares Optimization $LSQ(\mathbf{r_m}, K, \mathbf{p_m}, \mathbf{r}_{t-1}, \mathbf{v}_{t-1}, \Delta t, \mathbf{r_t}, \mathbf{v_t})$

for i *in* num_iters **do**
  $\hat{\mathbf{w}}_m, J_{g,m} = CameraProjection(\mathbf{r_m}, K, \mathbf{p_m})$
  $\hat{\mathbf{r}}_t, J_{d,t}, H_t = Dynamics(\mathbf{r}_{t-1}, \mathbf{v}_{t-1})$
  Compute ψ_t, ϕ_m using Eq 7.1, 7.2.
  $\hat{\mathbf{v}}_:, \hat{r}_: = LM\_Optimizer(J_{g,:}, J_{d,:}, \psi_:, \phi_:)$
end for
return $\hat{\mathbf{w}}, \hat{r}$

optimizer iteration count, i. $\alpha = 2$ corresponds to a least-squares kernel, and lower $\alpha's$ correspond to increasingly more robust kernels. Setting $\alpha = \max(2 - 2i/5, -1)$ allows us to compute an initial estimate using all points and then progressively make the estimates more robust to outliers.

Finally, we aggregate these costs across detections and time-steps and solve a joint optimization problem as follows:

$$\min_{(\mathbf{r},\mathbf{v})_{1:N}} \sum_m \phi_m + \lambda \sum_t \psi_t \tag{7.4}$$

We solve this optimization problem over all the poses jointly using a Levenberg-Marquardt (LM) method Levenberg [1944]. We provide an overview of the system for solving the batch least-squares problem in Algorithm 1.

7.4 Evaluation

The purpose of this evaluation is to demonstrate the ability of the VINSat system to sufficiently perform OD from typical CubeSat orbits in a reasonable time frame. This section details evaluations of the individual components of VINSat, as well as the end-to-end system. Our simulation results show that VINSat achieves kilometer-level OD in just a few orbits.

To evaluate VINSat, we require a large dataset of images captured from a low-cost and wide field of view camera on a satellite orbiting Earth, as well as images containing clouds, which are typically discarded by satellite imagery distributors. Unfortunately,

such a dataset is not currently available. Consequently, we divide our evaluation into three main components:

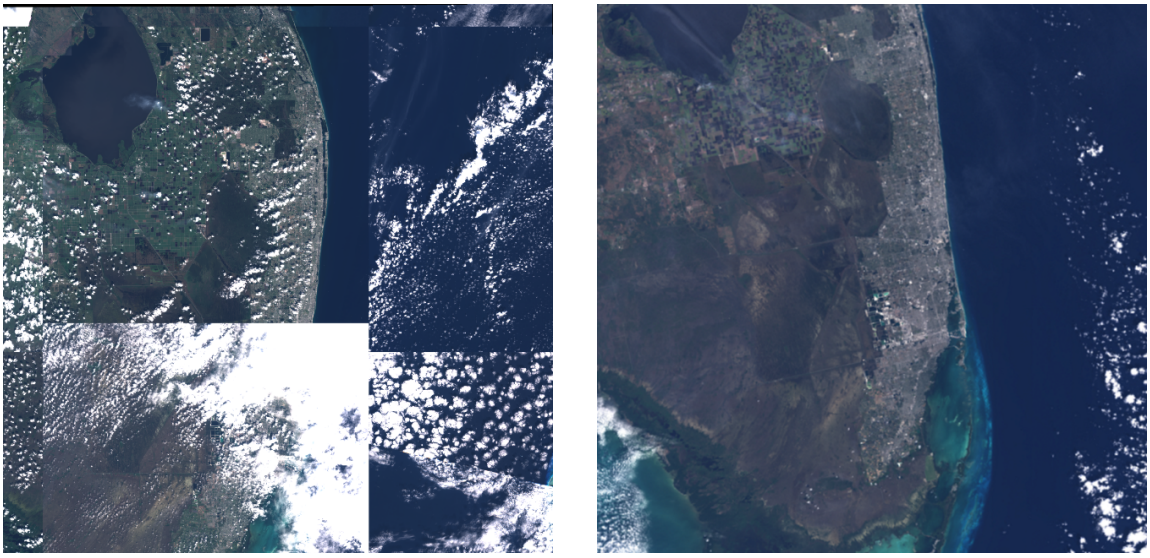
1. *Cross-Dataset Generalization*: We test generalization to different cameras and data sources by training on Sentinel 2 imagery and testing on Landsat imagery.
2. *Ablations and Sensitivity Analysis*: We perform ablation experiments to understand the importance of various design decisions on the batch optimizer.
3. *Simulation*: We simulate satellites in random polar and ISS-like orbits to evaluate the end-to-end pipeline.

7.4.1 Cross-Dataset Generalization

We test cross-dataset generalization by training on Sentinel 2 imagery and testing on Landsat imagery. We do this by first training on 2000 randomly mosaiced Sentinel-2 images for each of the 16 regions. The images are made from randomly selecting points in the region of interest, buffering around the point, mosaicing random Sentinel 2 images captured between 2020 and 2022, and exporting the image at a scale of 150 meters per pixel, similar to our reference camera. An example of one of these images is shown on the left of Fig. 7.5. We then validate the performance of each network on 500 Sentinel 2 images made the same way, but from data captured between 2018 and 2019. 250 raw Landsat 8 and 250 raw Landsat 9 scenes captured during 2023 are used for testing. An example Landsat 8 scene can be seen on the right of Fig. 7.5. All images were downloaded from Google Earth Engine Gorelick et al. [2017] in GeoTIFF format containing affine transformations of pixel points to ground points to maintain accurate ground truth when labeling landmarks.

For clarity the training, validation, and test datasets **for each region** are listed again below:

- *Training*: 2000 Sentinel 2 mosaics made from imagery captured between 2020 and 2022.
- *Validation*: 500 Sentinel 2 mosaics made from imagery captured between 2018 and 2020.



(a) Mosaiced imagery from Sentinel satellite (b) Imagery from Landsat satellite

Figure 7.5: Comparison of Sentinel and Landsat imagery. The many subtle differences between imagery sources, lighting conditions, and seasons lead to a challenging landmark detection problem.

- *Test*: 250 Landsat 8 and 250 Landsat 9 images captured during 2023.

Region Classification Network The RC network dataset is annotated by projecting landmarks onto the camera's image plane and identifying regions with visible landmarks. Each image is assigned a 16-value multi-hot label corresponding to the regions containing visible landmarks in the image. The RC network is validated by measuring the performance of the network on the validation dataset using network precision, recall, and F1 score.

The RC network achieves an overall accuracy of 99.2% on the validation set. The mean precision for all classes is 0.95, the mean recall for all classes is 0.97, and the mean F1 score for all classes is 0.96. These values demonstrate that the appropriate regions have high likelihood of being recognized in an image and passed to the corresponding landmark detection networks.

Landmark Detection Networks Each LD network dataset is annotated by projecting landmarks onto the camera's image plane and identifying the pixel locations of each landmark. We locate the center point and corners of each landmark in the image and draw a bounding box around it, ensuring that the center point of the label aligns with the center point of the landmark. Each LD network is evaluated by measuring the performance of the network on the test dataset by measuring mean pixel error of the center points of detected landmarks at varying confidence thresholds. Additionally, we measure the ratio of detections included at varying confidence thresholds.

We report results for the mean pixel error and ratio of points encompassed across all LD networks for confidence thresholds of 0.7, 0.8, and 0.9 in Table 7.2. From the table it is clear that a confidence threshold of 0.9 is too high as it includes a very small sample of the detections while not improving mean pixel error. We choose 0.8 as a confidence threshold as the majority of detections are included, but the mean error is improved over a threshold of 0.7.

These results suggest that both networks exhibit robust generalization to substantial variations in clouds, seasons, and image characteristics.

Table 7.2: Ld networks mean pixel error and ratio of included points at varying confidence thresholds.

	0.7	0.8	0.9
Mean Error (pixels)	1.79	1.66	1.87
Ratio of Included Points	0.86	0.65	0.03

7.4.2 Ablations and Sensitivity Analysis

We perform ablation experiments to understand the importance of various design decisions.

Effect of the orbital dynamics costs

We conducted tests to evaluate the influence of orbital dynamics terms on the performance of the batch optimizer using the end-to-end pipeline validation dataset and the detections from the image pipeline. We observe that, in the absence of the dynamics terms, the resulting trajectories have an RMS position error of 6.11 ± 3.32 km as opposed to 1.60 ± 0.87 km with the dynamics terms. Our observations highlight the critical role played by the dynamics terms.

Effect of outlier rejection

While our RC and LD networks excel at identifying landmarks on the Earth's surface, the presence of outliers in the detections significantly hampers OD. We use two primary methods for outlier rejection: confidence thresholding to remove detections below a specific confidence threshold and an adaptive cost kernel.

Table 7.3 presents the outcomes of confidence thresholding for ten randomly sampled LEO orbits, indicating that optimal results are achieved around a threshold of 0.8. This threshold strikes a balance between having enough data points for convergence while effectively controlling the impact of outliers. Higher confidence thresholds result in poorer performance due to data scarcity, whereas lower thresholds lead to an excess of outliers that disrupt convergence.

We also test our method without the adaptive kernel, replacing it with a simple L2 loss instead. This results in an RMS position error of 2.39 ± 1.44 km, which is much higher than our original results. We observe that the outlier rejection done by the adaptive kernel is very effective and contributes to significant performance improvements.

We observe that the performance of the batch least-squares optimizer improves monotonically with increasing density of detections at the same noise level. We test this by downsampling detections from a three hour orbit by varying it from 10% of detections per frame to 80% of the total detections per frame with the same noise level. An increase in the measurement density leads to a natural decrease in error. However,

Table 7.3: RMS OD error performance and confidence thresholds

RMS OD Error/Confidence	0.3	0.6	0.8	0.85
Average (km)	2.75	2.14	1.60	2.06
Median (km)	2.50	1.69	1.53	1.72
Std. Dev. (km)	1.71	1.18	0.87	1.30

as density continues to rise, the error asymptotically approaches zero, resulting in diminishing returns after 60% detections per frame. Table 7.4 shows the effect of detection density for 10 randomly sampled LEO orbits.

Table 7.4: RMS OD error performance and detection density

RMS OD Error/Detection Density	0.1	0.2	0.4	0.8
Average (km)	6.67	6.92	2.32	1.80
Median (km)	6.26	3.64	1.68	1.69
Std. Dev. (km)	4.93	8.31	1.28	0.65

Effect of detection signal noise

In Table 7.5, we illustrate the impact of pixel error on OD error. To simulate error originating from the detection network, we introduce white Gaussian noise to the camera pixel values of ten randomly sampled LEO orbits. The noise's standard deviation ranges from one to eight pixels. During these experiments, the OD error spans from 0.2 km to 1.35 km.

Table 7.5: RMS OD error performance for different pixel white noise standard deviation

RMS OD Error/Pixel noise σ	$1px$	$2px$	$4px$	$8px$
Average (km)	0.20	0.38	0.64	1.35
Median (km)	0.09	0.20	0.37	0.75
Std. Dev. (km)	0.17	0.32	0.49	1.11

7.4.3 Simulation

The simulation environment works by first generating a random near-polar or ISS-like orbit. This orbit is propagated at 1 Hz and the satellite pose is used at each timestep to determine the view of the Earth based on the reference camera intrinsics. Random Landsat 8 imagery is mosaiced to create an image in the view of the camera on the satellite. Images are captured at a rate of $\frac{1}{5}$ Hz. The image is then processed through the pipeline and detections are recorded. The detections are passed to the batch optimizer and the satellite's position is estimated.

We find that during simulation the LD networks sometimes struggle amidst significant cloud cover, but with the iterative elimination of "bad" classes (i.e. those with consistently high error) the LD networks consistently achieve detections with pixel error less than 10 pixels (1.5 km), often achieving low-single-digit pixel error. Occasional outliers with significant pixel error sometimes cause difficulty for the batch estimator. These outliers can be reduced by improving training of the LD networks and having more robust outlier detection.

An example view with detections and errors is shown in Fig. 7.6. Over 500 simulated random satellites were used to generate the cumulative distribution function-like plot shown in Fig. 7.7. The plot demonstrates that 85% of simulated nadir-pointing satellites reached a position error of less than 5 km in under four orbits. For satellites moving at 7.5 km/s, that corresponds to less than one second of position error.

7.5 Conclusions and Future Work

In this work, we introduce VINSat, an OD method that is, to the best our knowledge, the first fully autonomous, vision-based solution to the "lost-in-space" problem. Notably, our approach eliminates the need for expensive and bulky GPS receivers or time-consuming ground-based radar methods. We have also developed an evaluation pipeline and datasets using openly accessible tools, which we released alongside the paper. We hope that this spurs further research and development in this important

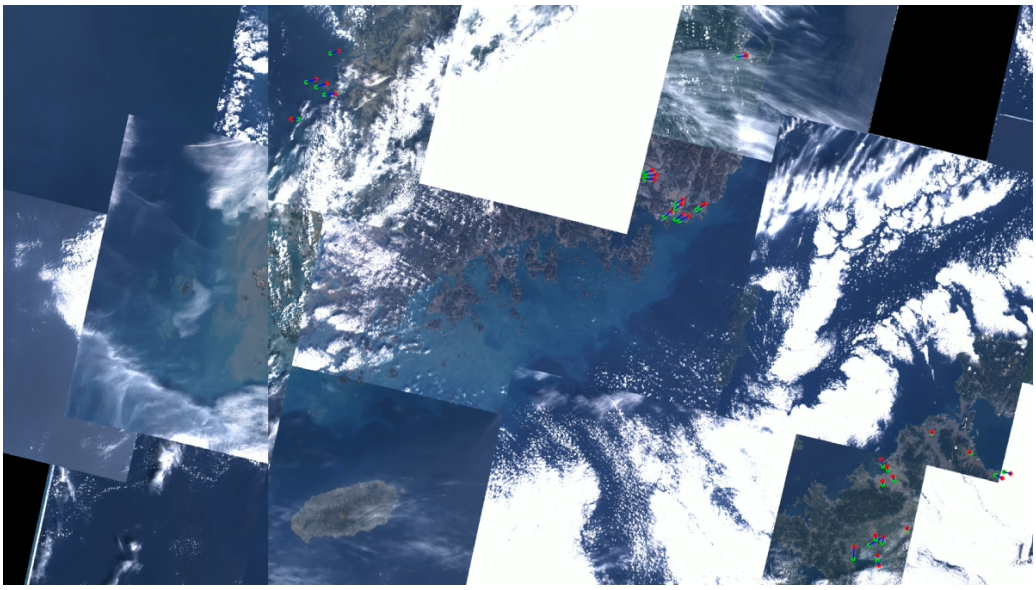


Figure 7.6: View of a simulation image. Predicted landmarks are green, ground truth landmarks are red, and a blue line connects each associated predicted landmark and ground truth landmark. The landmark detection networks are capable of detecting landmarks with minimal error in non-cloud obscured portions of images.

emerging field.

In fact, our labs have been building on this work to test the pipeline on hardware and finally on-orbit.

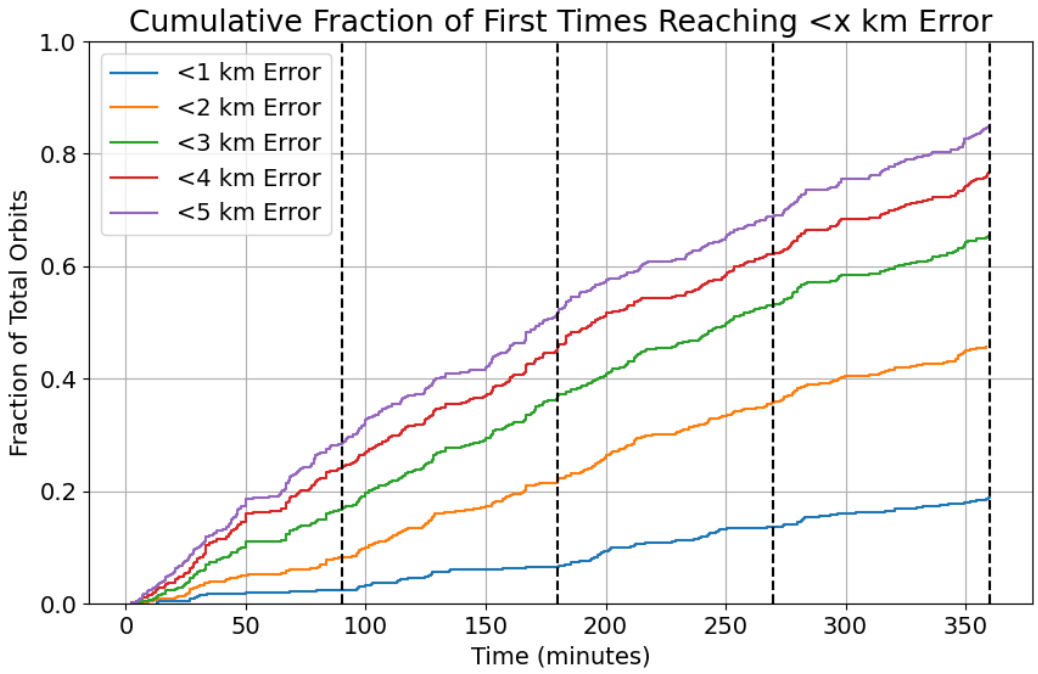


Figure 7.7: Cumulative distribution of satellite position error vs. time. The y-axis corresponds to the fraction of total simulated satellites and the x-axis corresponds to the time of the simulation in minutes. Each separate line corresponds to a localization threshold between one and five km. The time stops at six hours, equivalent to roughly four full orbits around the Earth. The dashed vertical lines represent 90 minute intervals, or around 1 orbit. After over 500 simulations, 85% of satellites were localized with less than 5 km error.

Chapter 8

Proposed Work : Deep Equilibrium Model Predictive Control

So far in this thesis we've looked at using least squares problems, non-linear least squares problems and then black box optimization problems such as reinforcement learning within the end-to-end learning pipelines. For this last chapter, we would like to consider differentiable constrained optimization layers. These layers offer an interesting set of representational and training stability challenges when plugged into deep networks naively. We will specifically consider robotics control applications to study these problems where we augment the network with a differentiable model predictive control (MPC) layer to account for systems constraints such as dynamics, control limits and collision avoidance. To address these issues, we propose a novel approach that co-develops the solver and architecture unifying the optimization solver and network inference problems. Through extensive ablations in various robotic control tasks, we demonstrate that our approach yields richer representations and more stable training, while naturally accommodating warm starts, a key requirement for MPC.

8.1 Introduction

Incorporating task-specific priors within the policy training pipeline is often beneficial for robotic control problems. These priors give the system designer additional control and flexibility while designing the system and play a vital role in enhancing safety, improving representation, and boosting generalization. Previous approaches to policy learning have explored various methods to embed such priors, including reward/loss shaping [Gupta et al., 2022], incorporating constrained optimization layers within the policy inference pipeline [Agrawal et al., 2020, Amos et al., 2018a], adding parallel/post-hoc safety checks/controllers [Ames et al., 2019], adversarial training [Schott et al., 2024], and domain randomization [Chen et al., 2021b].

Differentiable Model Predictive Control (MPC) layers [Amos et al., 2018a] have emerged as a promising approach [Diehl et al., 2023b, Shrestha et al., 2023, Xiao et al., 2022] to embed such priors. This method integrates MPC as a differentiable layer within neural network architectures, embedding the constraints and cost functions directly into the network architecture. Importantly, they allow us to preserve the interpretability and safety guarantees associated with traditional MPC while providing a general framework applicable to a diverse range of robotic control problems. Moreover, it allows for test-time modifications of the MPC problem and facilitates online adaptation – a critical feature in dynamic environments.

However, differentiable MPC layers treats the optimization solver as a black box, overlooking its unique characteristics. This simplification, while convenient, overlooks the unique characteristics of MPC solvers. Unlike typical NN layers, MPC solvers are implicit, iterative, and prone to issues like ill-conditioning and discontinuities, potentially destabilizing training. Their structure also enables efficient warm-starting, often underutilized in current frameworks.

To address these limitations, we propose Deep Equilibrium Model Predictive Control (DEQ-MPC), a novel approach that unifies the optimization solver and the neural network architecture. Instead of treating the optimization layer as just another layer within the network, we formulate a joint inference and optimization problem as shown in fig:model-fig, where we treat the network inference and the optimization problem

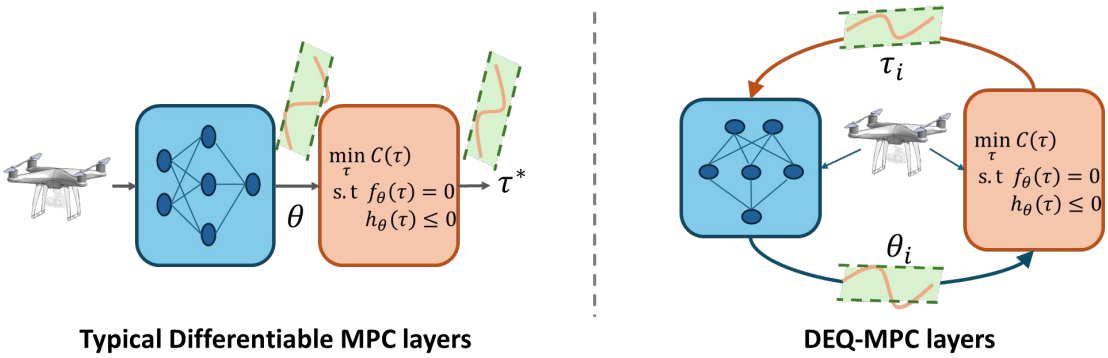


Figure 8.1: We propose DEQ-MPC layers (right) as a direct improvement over differentiable MPC layers (left). These layers offer increased representational power, smoother gradients, and are more amenable to warm-starting. DEQ-MPC layers formulate MPC problem input (θ) estimation and trajectory optimization (τ) as a joint fixed-point problem, solving them in an alternating iterative manner, instead of the single-shot sequential inference used in differentiable MPC setups. This approach allows the network to adapt the optimization input estimates, θ_i, based on the current optimizer state, τ_i, enabling a richer feedback process. The specific example in the figure shows a trajectory tracking example, where the robot observations (quadrotor) are fed to the system. The network predicts the waypoints θ_i (optimization inputs). The solver solves the tracking problem to spit out solved trajectories τ_i to track the waypoints θ_i.

as a unified system and jointly compute a fixed point over them. Thus, the network outputs can now depend on the solver iterates and vice-versa, thereby, allowing a tight coupling between the two. The fixed point is computed by alternating between the network forward pass (conditioned on the most recent optimizer iterate) and the optimization solver iterations (conditioned on the most recent network outputs) until the joint system reaches an equilibrium (hence the name DEQ-MPC, i.e, Deep Equilibrium Model Predictive Control).

This joint inference/optimization framework also allows us to explore several interesting aspects of the solver and architecture design. Specifically, for the optimization solver, we implement an augmented Lagrangian (AL) solver which works well with warm-starting and is robust at handling arbitrary non-linear constraints. This is important for the joint fixed point process as it allows us to change the optimization parameters (i.e, network outputs, θ_i) between successive optimization iterates. For the architecture, we experiment with parameterizing the network architecture itself

as a Deep Equilibrium model (DEQ), a type of implicit neural network that computes the outputs/latents as a fixed point of a non-linear transformation. It can be seen as an infinite depth network which applies the same layer an infinite number of times eventually reaching a fixed point in the outputs/latents. This iterative fixed point finding procedure blends nicely with the equilibrium/fixed point finding nature of the overall system. We observe nicer stability properties when using a DEQ as the network architecture when going to more complicated settings.

This unified approach results in several key benefits: First, it enables richer representations by allowing the network to adapt its features/outputs depending on the solver state. Second, it allows us to naturally compute smoother gradients during training, facilitating more stable and efficient learning. Third, it inherently accommodates warm-starting, leveraging the recurrent nature of MPC to improve computational efficiency and solution quality. DEQ-MPC thus offers a more robust and flexible framework for integrating optimization-based control with deep learning.

The primary contributions of this work are as follows: (1) We introduce DEQ-MPC, a novel framework to integrate MPC layers within deep networks. (2) Through extensive ablations, we show that this unified approach results in richer representations, improved gradient flow, and enhanced suitability for warm-starting, compared to standard differentiable MPC methods. (3) We propose a training setup specifically for streaming MPC applications that leverages warm-starting across time steps. (4) We provide empirical evidence demonstrating the advantages of DEQ-MPC on trajectory prediction and tracking problems across various continuous control tasks that require strict constraint satisfaction both on simulation and hardware. While the paper focuses on MPC to ground our methods in concrete problems, we believe that the insights and techniques would generalize to a much broader class of constrained optimization layers and applications.

8.2 Connections to related work

Differentiable optimization layers were introduced as a means to embed convex optimization problems [Agrawal et al., 2019, Amos and Kolter, 2017] as differentiable units within deep networks. Recent works have extended the range of optimization

and fixed-point problems that can be made differentiable [Gould et al., 2021, Jin et al., 2020, Landry et al., 2019, Pineda et al., 2022]. Since their introduction, they have been applied to a variety of robotics problems, such as state estimation [Yi et al., 2021], SLAM [Lipson et al., 2022, Teed et al., 2023], motion planning [Bhardwaj et al., 2020, Landry et al., 2019] and control [Agrawal et al., 2020, Amos et al., 2018a] for applications such as autonomous driving [Diehl et al., 2023a, Shrestha et al., 2023], navigation [Diehl et al., 2023b, Xiao et al., 2022], and manipulation [Landry et al., 2019]. We specifically look at differentiable MPC problems building off of work such as [Agrawal et al., 2020, Amos et al., 2018a, Howell et al., 2022b, Landry et al., 2019, Romero et al., 2023, Wan et al., 2024].

However, incorporating MPC (and optimization) layers within deep networks often comes with its own set of challenges. The bi-level problem can often be very non-convex resulting in the local gradient direction being mis-aligned with the desired global update direction [Amos et al., 2018a, Landry et al., 2019]. The gradient landscape often has discontinuities resulting in undesirable gradient artifacts [Antonova et al., 2023, Suh et al., 2022]. Furthermore, the problem structure can also result in very high variance in gradients [Gurumurthy et al., 2024]. It's often challenging to incorporate warm-starting techniques as the problem parameters change with each problem instance [Sambharya et al., 2024] resulting in long inference and solve times. The network predicted constraint parameters can often be infeasible [Donti et al., 2021], resulting in undefined problem solutions or gradients. Some modelling assumptions in the optimization layer are often not faithful to the real data causing model mismatch problems [Gurumurthy et al., 2023a]. Addressing these challenges is key for practical adoption of optimization layers.

Our approach formulates network inference and optimization as a joint equilibrium problem, addressing representation, gradient smoothness, and warm-starting. This relates to prior work using joint inference/optimization for inverse problems [Gurumurthy et al., 2021] and SLAM/pose estimation [Lipson et al., 2022, Teed and Deng, 2021, Teed et al., 2023], which focused on unconstrained non-linear least squares. We generalize this concept to constrained optimization, specifically MPC, and explicitly compare against standard differentiable optimization to highlight the benefits of the joint approach.

8.3 Background

8.3.1 Differentiable Model Predictive Control

Model Predictive Control (MPC) solves a finite-horizon optimal control problem at each time step :

$$\begin{aligned} \tau^*_{0:T} = \underset{\tau_{0:T}}{\text{argmin}} \quad & \sum_t C_{\theta,t}(\tau_t) \\ \text{subject to} \quad & x_0 = x_{\text{init}},\ x_{t+1} = f_\theta(\tau_t),\ h_\theta(\tau_t) \leq 0,\ t = 0, \ldots, T, \end{aligned} \tag{8.1}$$

where $\tau_t = (x_t, u_t)$, $C_{\theta,t}$ is the cost, f_θ the dynamics and h_θ ineq. constraints (e.g. safety constraints, joint limits, etc.). This problem is typically solved using non-linear optimization techniques.

Differentiable MPC computes gradients of the solution τ^* w.r.t. solver inputs θ using the implicit function theorem (IFT) on the Karush-Kuhn-Tucker (KKT) conditions of (8.1) [Amos et al., 2018a]. Let $z^* = (\tau^*, \lambda^*, \nu^*)$ be the primal-dual solution to the KKT conditions $F(z, \theta) = 0$ of (8.1). The corresponding gradient can be computed as:

$$\frac{\partial z^*}{\partial \theta} = -\left(\frac{\partial F}{\partial z}\right)^{-1} \cdot \frac{\partial F}{\partial \theta}, \tag{8.2}$$

8.3.2 Deep Equilibrium Models

Deep Equilibrium Models [Bai et al., 2019a] are a class of implicit deep learning models that compute the output as a solution to a fixed point problem. Specifically, given an input $x \in \mathcal{X}$, computing the forward pass in a DEQ model involves finding a fixed point $z \in \mathcal{Z}$, such that

$$z^\star = d_\phi(z^\star, x), \tag{8.3}$$

where, $d_\phi : \mathcal{Z} \times \mathcal{X} \to \mathcal{Z}$ is some parameterized layer conditioned on input x, $\mathcal{Z}$ denotes the hidden state or outputs of the network which we are computing the fixed point on, $\mathcal{X}$ denotes the input space, and ϕ denotes the parameters of the layer.

Computing this fixed point (under proper stability conditions) corresponds to the "infinite depth" limit of repeatedly applying the function $z^+ := d_\phi(z, x)$ starting at some arbitrary initial value of z (typically 0).

8.4 Method

8.4.1 Problem Grounding through Trajectory Prediction and Tracking

We begin by grounding our discussion in a simple trajectory prediction and tracking example. This setting will also serve as the default configuration for most of our subsequent experiments.

Given system dynamics f and a dataset of optimal trajectories across different initial and environmental conditions, we learn a policy using imitation learning problem while respecting several constraints. We model this policy as consisting of two components. The first is a neural network NN_ϕ that predicts the waypoints to be tracked and other environment parameters $\theta_{0:T}$ for the next T time steps given the current state x_{init} and some observations o:

$$\theta_{0:T} = \text{NN}_\phi(x_{\text{init}}, o). \tag{8.4}$$

The second is an MPC solver that solves the trajectory tracking problem to compute dynamically feasible trajectories $\tau_{0:T}$ that track the waypoints while satisfying the required constraints:

$$\begin{aligned} \tau^*_{0:T} = \underset{\tau_{0:T}}{\text{argmin}} \quad & \sum_t \|x_t - \theta_t\|^2_Q + \|u_t\|^2_R \\ \text{subject to} \quad & x_0 = x_{\text{init}},\ x_{t+1} = f_\theta(\tau_t),\ h_\theta(\tau_t) \leq 0,\ t = 0, \ldots, T. \end{aligned} \tag{8.5}$$

In a standard differentiable-MPC setup these two components are executed sequentially, one after the other as shown in Figure 8.1. The outputs of the system, $\tau^*_{0:T}$ are used to compute a loss, $\ell(\tau^*_{0:T})$, such as a supervised L1 loss with some expert trajectory demonstrations $\tau^{\text{exp}}_{0:T}$. The loss is then optimized using a stochastic gradient optimizer to learn the network parameters.

8.4.2 DEQ-MPC

The Inference Problem, Architecture and Solver

MPC solvers are implicit layers and hence iterative. Using a single θ estimate throughout the solver iterations is inefficient, especially for non-linear optimization problems. To address this, DEQ-MPC modifies the single-shot inference problem described in Equations 8.4 and 8.5 into a joint inference/optimization problem over the network outputs and the optimizer iterates. This approach, illustrated in Figure 8.1, allows us to condition the network outputs (solver inputs, θ) on the optimizer state τ and vice versa. This is expressed as a single constrained optimization problem:

$$\tau^*_{0:T}, \theta^* = \underset{\tau_{0:T},\theta}{\text{argmin}} \quad \sum_t C_{\theta,t}(\tau_t) \tag{8.6}$$

$$\text{subject to} \quad x_0 = x_{\text{init}},\ x_{t+1} = f_\theta,\ h_\theta(\tau_t) \leq 0, \tag{8.7}$$

$$\theta = \text{NN}_\phi(x_{\text{init}}, o, \tau_{0:T}),\ t = 0, \ldots, T, \tag{8.8}$$

where the last constraint expresses the neural network inference as an equality constraint. Typical non-linear optimization solvers [Biegler and Zavala, 2009, Gill et al., 2005] or bi-level optimization solvers [Huang et al., 2022, Le Cleac'h et al., 2024, Zheng et al., 2022] struggle with this constraint due to the nastiness of the resulting constraint Jacobians. We propose to solve this problem using the alternating direction method of multipliers (ADMM) algorithm [Boyd et al., 2011], alternating between (1) solving the MPC optimization problem (with fixed θ), Equations 8.6 and 8.7 using the augmented Lagrangian (AL) method and (2) the constraint projection step, Equation 8.8 (i.e, the neural net inference to compute θ with fixed τ). Specifically, we alternate between the following two operations for N iterations or until convergence,

$$\theta^i = \text{NN}_\phi(x_{\text{init}}, o, \tau^{i-1}), \tag{8.9}$$

$$\tau^i = \text{MPC-m}_{\theta^i}(x_{\text{init}}, \tau^{i-1}), \tag{8.10}$$

where MPC-m performs m solver iterations using the AL algorithm, with the most recent estimate θ^i from the network and warm-started using τ^{i-1} from the last MPC-m solve. The initial value τ^0 are initialized at x_{init} and zero controls across time steps.

We refer to each alternating step as a DEQ-MPC-iteration, with the super-script, i, denoting the iteration count. This is illustrated in Figure 8.1. This iterative inference/optimization approach enables the network to provide an initial coarse θ estimate and iteratively refine it based on the solver's progress.

Choice of N, m: Empirically, we find that updating the MPC inputs θ every two AL iterations ($m = 2$) is sufficient to obtain most of the gains. Furthermore, DEQ-MPC typically converges within $N = 6$ DEQ-MPC-iterations with $m = 2$ and thus we use these values for all our experiments. We discuss the considerations around the convergence of this alternating problem in section 8.5.4.

Network architecture. We explore two architectural choices for NN_ϕ with distinct trade-offs:

(1) *DEQ-MPC-NN:* NN_ϕ is a standard feedforward network. While this is simple and often effective, it has limitations. The iterative nature of the DEQ-MPC framework can lead to instabilities when using a standard feedforward architecture, particularly in complex settings. Moreover, this architecture is somewhat computationally inefficient, as it doesn't leverage the similarity of computations across successive iterations – each iteration starts anew without reusing previous computational results.

(2) *DEQ-MPC-DEQ:* NN_ϕ itself is a DEQ network [Bai et al., 2019a]. Specifically, the network inference step in Equation 8.9 is itself computed via an inner fixed-point solve: $z_i^\star = d_\phi(z_i^\star, x_{\text{init}}, o, \tau_{i-1})$. followed by $\theta_i = g_\phi(z_i^\star)$. Note that this fixed point solve is distinct from the equilibrium computations in the DEQ-MPC-iterations discussed earlier and is simply computing the network inference (i.e constraint projections) from Equation 8.9. Furthermore, we can also warm-start these inner fixed points across successive DEQ-MPC-iterations given they are likely to be similar, i.e, z_i can be conveniently initialized with z_{i-1} while computing the fixed points. This allows us to re-use the network computation from earlier iterations.

MPC-m solver. We implement an Augmented Lagrangian (AL) solver [Nocedal and Wright, 2006, Toussaint, 2014] for MPC-m, chosen for its robustness to non-linear constraints (handled as penalties) and suitability for warm-starting. The penalty-

based approach also allows us to use the unconverged iterations as smoothed/relaxed versions of the problem to handle discontinuities (more discussion in sec:lossgrad). Our solver implementation is friendly with both CPU and GPU.

Specifically, for the general MPC problem in (8.1), we form the following Lagrangian

$$\mathcal{L}(\tau, \lambda, \eta, \mu) = \sum_t C_{\theta,t}(\tau_t) + \lambda^T h_\theta(\tau) + \eta^T k_\theta(\tau_t, x_{t+1}) + \frac{\mu}{2}\|h_\theta(\tau_t)^+\|_2^2 + \frac{\mu}{2}\|k_\theta(\tau_t, x_{t+1})\|_2^2, \tag{8.11}$$

where $h_\theta(\tau_t) \leq 0$ are the inequality constraints and $k_\theta(\tau_t, x_{t+1}) = 0$ are all the equality constraints (including the dynamics and initial state constraints), λ and η are the corresponding Lagrange multipliers and $\mu > 0$ is the penalty parameter. $h_\theta(\tau_t)^+$ represents an element-wise clipping at zero $\max(0, h_\theta(\tau_t))$. The augmented Lagrangian method then proceeds by alternating between updating the primal variables (τ), dual variables (λ, η) and the penalty parameter (μ) as shown in algorithm 2.

Moreover, with MPC-m, we only perform m AL iterations per DEQ-MPC-iteration i, initializing all the AL variables (τ^i, λ^i, η^i, μ^i) at iteration i from the state at the end of iteration $i-1$.

Algorithm 2 Augmented Lagrangian Solver for MPC-m

Require: Initialize τ^0, λ^0, η^0, μ^0 (warm-started using previous DEQ-MPC-iteration, parameters), $\gamma > 1$

1: Set $j = 0$

2: **repeat**

3: **Primal update:** Solve the unconstrained minimization problem using the Gauss-Newton method

$$\tau^{j+1} = \arg\min_{\tau} \mathcal{L}(\tau, \lambda^j, \eta^j, \mu^j)$$

4: **Dual update:** Update the Lagrange multipliers

$$\lambda^{j+1} = \max(\lambda^j + \mu^j h_\theta(\tau^{j+1}), 0)$$

$$\eta^{j+1} = \eta^j + \mu^j k_\theta(\tau^{j+1})$$

5: **Penalty update:** Update the penalty parameter

$$\mu^{j+1} = \gamma \mu^j$$

6: $j = j + 1$

7: **until** Stopping criterion is met (or $j = m$ iterations)

8: **return** Final solution τ^m, λ^m, η^m, μ^m

Loss and Gradients

Augmented Lagrangian gradients. Previous work [Antonova et al., 2023, Suh et al., 2022] has shown that computing gradients through optimization problems can be problematic due to inherent discontinuities in the landscape and have proposed various relaxations to mitigate this problem. We take inspiration from these approaches and propose a relaxation for use with our solver.

We compute input gradients ∇_θ for the AL solver by applying IFT (Eq. 8.2) on the

Lagrangian :

$$\tau^* = -(\nabla^2_\tau \mathcal{L})^{-1} \nabla_{\tau\theta} \mathcal{L} = -(Q + \mu A^T A + \mu G^T G)^{-1} \nabla_{\tau\theta} \mathcal{L}. \tag{8.12}$$

where, A and G are the constraint Jacobians of the equality and inequality constraints respectively. At convergence, the value of μ is very high. This results in the components of the gradient in the column space of the linearized active constraints getting squished to zero. Thus, when the constraints are non-linear/discontinuous, and the optimizer converges to some arbitrary active sets, the gradients computed using Equation 8.12 are also arbitrary/meaningless. To address this, we compute losses on multiple intermediate iterates $\tau^i \forall i \in [1, N]$ from the DEQ-MPC iterations, not just the final one τ^N. Gradients computed through earlier iterates (with smaller effective μ) provide smoother signals, acting as a relaxation. Later iterates (with larger μ) provide more accurate gradients as the solver converges. This creates a natural curriculum during training. We discuss the details of gradient computation for the DEQ network in the appendix.

8.4.3 DEQ network gradients

Computing gradients through the fixed point iteration in a DEQ model typically requires using the implicit function theorem (8.2), which involves computing a linear system solve. However, recent work [Fung et al., 2021b, Geng et al., 2021] has shown that the approximations of the gradient by simply assuming an identity Jacobian or differentiating through the last few iterations of the fixed point iteration using vanilla backpropagation is equally/more effective while being computationally cheaper. We adopt this approach. Specifically, we run the function a couple more times after computing the fixed point, and simply backpropagate through those last couple of iterations to compute the parameter gradients.

Losses. We supervise the policy outputs with the expert trajectories for the following T steps. We use an L1 loss over the output states against the corresponding ground truths for supervision. As discussed before, we compute losses on multiple intermediate

iterates and backpropagate gradients through all of them. The resulting objective for a single instance is

$$\ell(x_{0:T}^{\text{exp}}, x_{0:T}^{1:I}) = \sum_{t=0:T} \sum_{j=1:I} \|x_t^{\text{exp}} - x_t^i\|_1, \tag{8.13}$$

where $x_{0:T}^{\text{exp}}$ are expert demonstrations and $x_{0:T}^{1:I}$ are the states output by the model across I iterations.

Warm-Starting and Streaming

Warm-starting. Warm-starting MPC by initializing the current solve with the previous time step's solution is vital for efficiency [Howell et al., 2019, Le Cleac'h et al., 2024, Nguyen et al., 2024]. Standard MPC warm-starting involves shifting the previous solution $\hat{\tau}_{t-1:T+t-1}$ to initialize the solve for $\tau_{t:T+t}$. The AL method accommodates this very elegantly. We initialize τ with the shifted estimate $\tau_{t:T+t} = [\hat{\tau}_{t:T+t-1}, \hat{\tau}_{T+t-1}]$, where $\hat{\tau}_{T+t-1}$ is assumed to be a reasonable estimate for τ_{T+t}, reset the dual variables λ and η to zeros and set the initial value of $\rho = \rho_{\max}/10^{N*m-i}$ where $(N*m-i)$ is the total number of AL iterations we expect to perform after warm-starting. Note that we warm-start the primal variables τ but not the dual variables. Dual variables are tricky to warm-start in the AL-MPC setup for two reasons : (1) the active sets often change across consecutive time-steps especially when the optimized state at t + 1 differs significantly from the next observation. This becomes prominent when deploying the policies in the real world due to the sim-2-real gap. (2) When warm-starting AL, the penalty ρ is often re-initialized to a smaller value which means that the relative scale of the multiplier updates $\lambda := \lambda + \rho * res$ are much smaller than the multiplier magnitudes making the initial updates ineffective. Thus, it's generally considered a safer practice to reset the multiplier values instead of warm-starting them.

In standard differentiable-MPC setups, the network infers the MPC solver inputs θ afresh at each successive time step. These estimates can often be arbitrarily far from the previous estimates, thus requiring a significant number of AL iterations post warm-start. On the other hand, in DEQ-MPC, the network is conditioned on the previous optimizer iterate. This allows us to train the network to predict consistent θ estimates across time-steps by training it specifically for the streaming setting as

described below.

Streaming training. We customize the training procedure to suit the warm-started streaming setup. Given a sampled ground truth trajectory $\tau^{\text{exp}}_{0:T+L}$, we break the inference problem into a two step process. First, we solve for $\tau_{0:T}$ given x^{exp}_0 as usual without any warm-starting. Then, we successively solve L problems for $\tau_{t:T+t}$ for $t = 1 \dots N$ with the iterates warm-started with solution from the previous solve, $\tau_{t-1:T+t-1}$. Then we simply compute losses on all the intermediate optimization iterates (from both steps) and supervise them using the corresponding ground truths as described in sec:lossgrad. For all of our experiments we use $L = 2$.

8.5 Experiments

We demonstrate the effectiveness of our proposed modifications across a variety of simulated robotic control tasks. Additionally, we present ablation studies to highlight the specific advantages of DEQ-MPC regarding representation, training stability, and warm-starting. Finally, we validate our approach with hardware experiments on a Crazyflie drone.

Setup. We use the trajectory prediction and tracking problem (Section 8.4.2) as our default experimental setting. For each task, we generate ground truth trajectories using expert policies; details on dataset generation and partitioning are in Appendix.Models are trained via supervised learning to predict the next T steps (T=5 unless specified) given the current state. We evaluate models using validation error (for general optimization layer effectiveness) and average returns over 200 receding-horizon rollouts (for MPC policy performance).

Variants/Baselines. Throughout the experiments, we compare our methods (DEQ-MPC-*) against their corresponding differentiable MPC counterparts (Diff-MPC-*):

- *DEQ-MPC-DEQ:* Our method using a DEQ network architecture.
- *DEQ-MPC-NN:* Our method using a standard feedforward network.

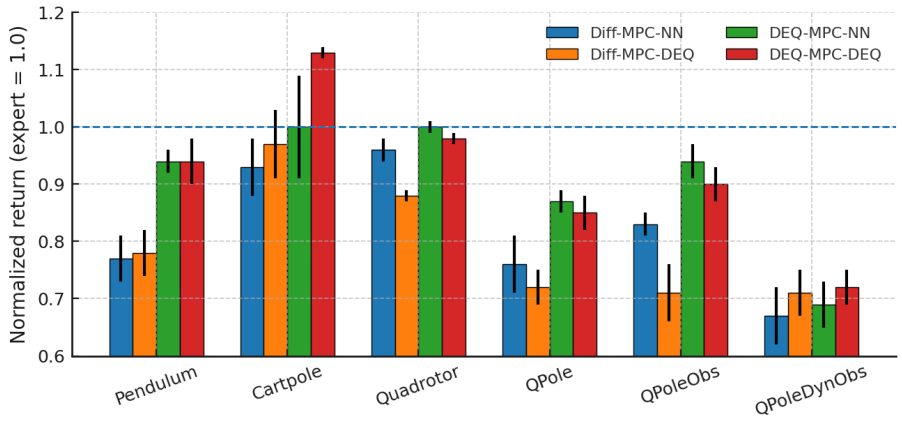


Figure 8.2: Performance comparison across various simulated environments, with values normalized against the expert return for each environment. Higher score is better.

- *Diff-MPC-NN:* Standard differentiable MPC with a feedforward network predicting θ in one shot, solved via MPC, with loss on the converged iterate via IFT.
- *Diff-MPC-DEQ:* Same as Diff-MPC-NN but using a DEQ network architecture.

Network Architecture. Given the sequential nature of the task, we use a temporal convolution-based architecture for both feedforward (NN) and DEQ networks. Details are in Appendix.

8.5.1 Comparison Results

We evaluate the methods on a series of underactuated continuous control tasks with constraints: *Pendulum*, *Cartpole*, *Quadrotor*, *QPole*, *QPoleObs*, and *QPoleDynObs*. QPole is a quadrotor with a pole hanging (task is to swing up the pole while reaching a goal). *QPoleObs* adds obstacles in the environment that need to be avoided (using obstacle avoidance constraints in MPC). *QPoleDynObs* makes these obstacles dynamic.

The policies are trained and executed in the streaming setting (Section 8.4.3) with a single DEQ-MPC-iteration (DEQ-MPC variants)/two AL iterations (Diff-MPC variants) with warm-starting across environments, except in the QPoleObs env,

where all methods needed two DEQ-MPC-iterations (DEQ-MPC)/four AL iterations (Diff-MPC). (Note that each DEQ-MPC iteration itself also does exactly two AL iterations with $m = 2$). Figure 8.2 shows the normalized returns obtained by each policy for each task averaged across policies trained with three dataset splits. The returns are normalized against the expert policy (1.00). We observe that the DEQ-MPC variants consistently perform better than the Diff-MPC counterparts across most environments. While DEQ-MPC-DEQ performs consistently well across all environments, we observed that DEQ-MPC-NN occasionally got unstable (e.g. resulting in its sub-par performance in the Cartpole balancing task).

8.5.2 Ablations

We explore three aspects: representation capabilities, training stability, and warm-startability, primarily using the QPole environment unless specified.

Representation Ablations

We demonstrate DEQ-MPC's enhanced representation capabilities. First, DEQ-MPC variants scale more effectively with dataset size and model capacity. Second, they show less performance degradation as constraint complexity increases. Additional representation ablations are in Appendi-

Generalization. Figure 8.3 shows validation error versus training set size. DEQ-MPC models show benefits even with smaller datasets and continue improving with more data, unlike Diff-MPC variants which saturate. This suggests better representation power. Networks trained without MPC layers (DEQ, NN) also saturate, indicating benefits arise from solver-network interplay.

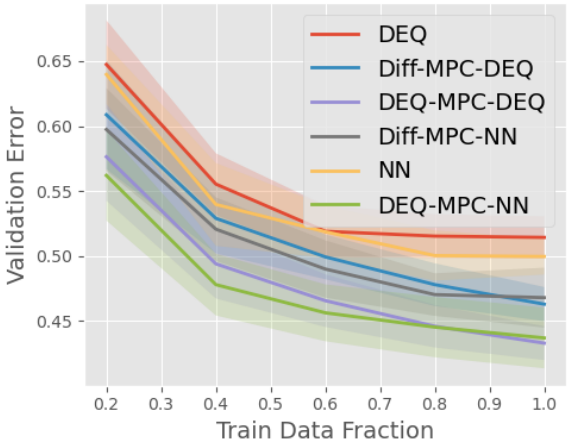


Figure 8.3: Generalization

Network capacity. Figure 8.4 shows validation error versus network hidden state size (128 to 1024). We observe that the DEQ-MPC variants benefit more from the higher network capacity than the Diff-MPC variants which saturate beyond hidden size of 512. This shows that the DEQ-MPC variants are better able to utilize additional model capacity and thus are more amenable to scaling.

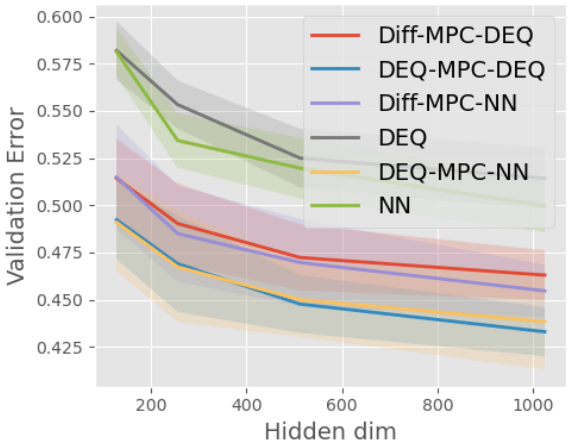


Figure 8.4: Network capacity

Constraint hardness. We add 40 obstacles to QPole with collision avoidance constraints ($\|x_d - x_o\|_2^2 \geq r^2$) added to MPC, where x_d = COM of drone and x_o = COM of obstacle. Figure 8.5 shows the returns obtained by different models as we vary the obstacle radius r from 0.20 to 0.50. DEQ-MPC's performance advantage persists as we add additional constraints and even increases as constraints become harder (larger r). (Note: These runs are without warm-starting to isolate representational effects).

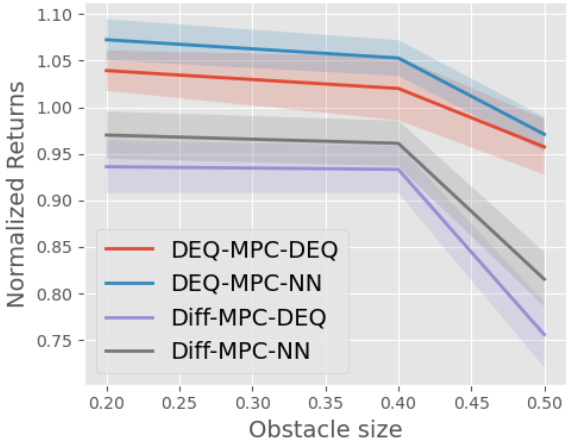


Figure 8.5: Constraints hardness

Training stability

Gradient niceness. Figure 8.6 presents the validation errors during training for DEQ-MPC-DEQ (where we compute losses across multiple intermediate AL iterates and backpropagate) and Diff-MPC-DEQ (where gradients are computed only with the final AL

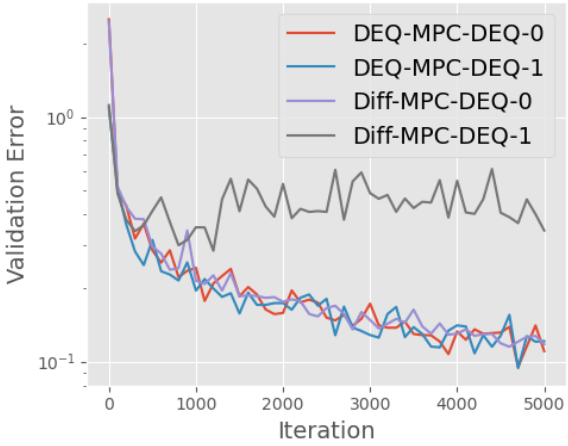


Figure 8.6: Gradient instability

iterate). Diff-MPC-DEQ shows significant instability when tight control limits are added (postfix 1), unlike DEQ-MPC-DEQ. Both are stable without these constraints (postfix 0). This highlights the benefit of using intermediate iterates for smoother gradients.

MPC input sensitivity ∇_θ. Figure 8.7 shows validation errors as we vary velocity coefficients in the MPC cost Q (lower values $\rightarrow$ worse conditioning $\rightarrow$ higher problem sensitivity). We observe that this leads to more training instability in models. The validation errors plotted represent the 'best' performance of the model throughout training (typically just before the training became unstable). DEQ-MPC-DEQ remains stable for the largest range of values. Even DEQ-MPC-NN, although best performing with well conditioned Q, quickly gets very unstable as conditioning worsens.

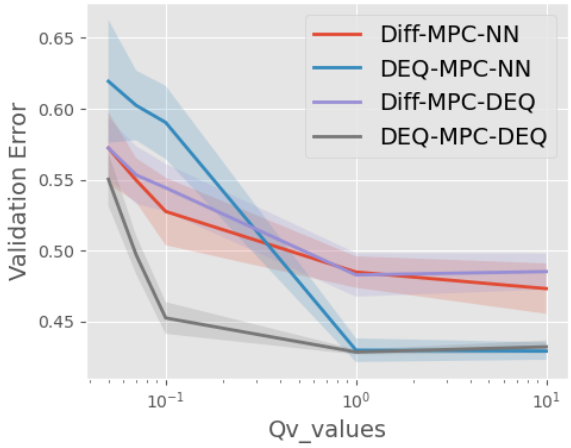


Figure 8.7: Cost parameter sensitivity

Warm-starting ablations

Figure 8.8 shows returns as we vary the number of DEQ-MPC/AL iterations allowed per step in the streaming/warm-started evaluation (Section 8.4.3, L=2). Note that, each DEQ-MPC iteration does exactly two AL iterations ($m = 2$). As the computational budget (iterations per step) decreases, the performance gap between DEQ-MPC and Diff-MPC widens significantly. DEQ-MPC's iterative nature makes it inherently

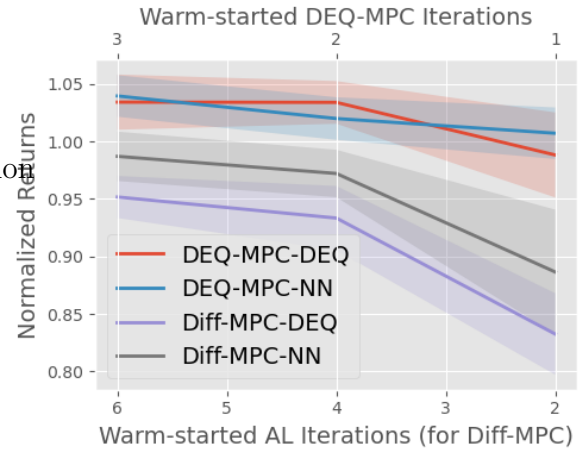


Figure 8.8: Warm-starting

better suited to leverage warm-starting effectively, given that the warm-starting required at each new time-step is very similar to the warm-starts done across DEQ-MPC iterations.

Representational hardness. We look at the effect of increasing horizon length as it serves as a good proxy for various metrics such as problem conditioning, dimensionality, practical utility etc. Specifically, Figure 8.9 shows the validation errors obtained by each model after training as we vary the horizon length from $T = 3$ to $T = 12$. We observe that the gap between the validation errors of the iterative models and the non-iterative ones is preserved even as we increase the size of the problem. Further, we observe that the representational benefits of the DEQ network in DEQ-MPC-DEQ starts becoming more obvious in the longer horizon problems as the difference in validation error between DEQ-MPC-DEQ and DEQ-MPC-NN increases. This illustrates the effectiveness of the infinite depth in DEQs helping with capturing the longer context.

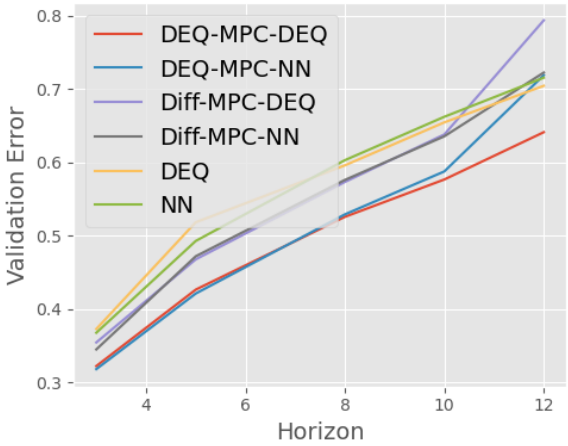


Figure 8.9: Time horizon ablations

Validation error with iteration count. Figure 8.10 shows the validation error across the Augmented Lagrangian iterations. As discussed earlier, the Diff-MPC variants here use the same predicted θ throughout iterations while the DEQ-MPC variants use ADMM and thus update the optimization inputs θ using the network inference every two AL iterations. Interestingly,

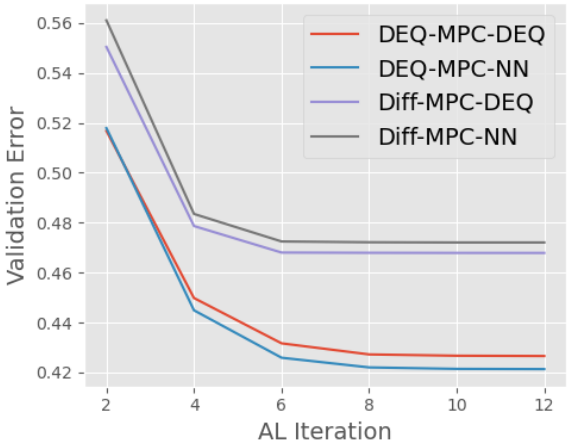
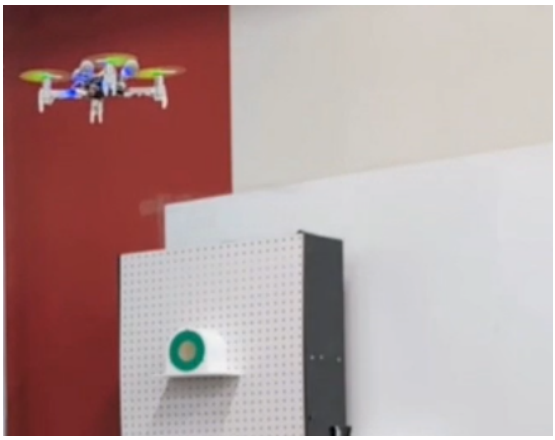


Figure 8.10: AL iteration ablations



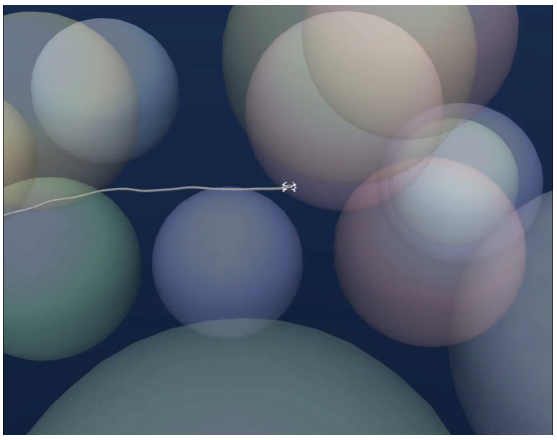


Figure 8.11: **Left :** The CrazyFlie flying with the DEQ-MPC-DEQ policy in the real-world assuming the virtual obstacles exist. **Right :** The CrazyFlie trajectory from the real world overlayed in the simulator with virtual obstacles.

the gap in validation error starts accruing from the early AL iterations itself. But gap gets pronounced after the fourth AL iteration as the Diff-MPC variants saturate while DEQ-MPC continues to improve thanks to the repeated updates to the problem inputs.

8.5.3 Hardware Experiments

Table 8.1: Hardware results

Method	Average Return	Failure Rate (%)
Diff-MPC-NN	0.82 (±0.15)	33.33
Diff-MPC-DEQ	0.86 (±0.12)	33.33
DEQ-MPC-NN	**0.93 (±0.03)**	33.33
DEQ-MPC-DEQ	**0.93 (±0.04)**	**0.0**

We further validate our approach on Crazyflie hardware. We deployed policies, learned through imitation of a pre-trained Reinforcement Learning (RL) expert, onto a Crazyflie 2.1 micro-quadrotor. The task involved navigating to the origin amidst 40 virtual static spherical obstacles of radius 10cm. Figure 8.11 shows the CrazyFly flying in the real-world assuming the virtual obstacles exist and the traversed trajectory visualized in the sim overlayed with the virtual obstacles. The policies were evaluated over 3 trials on the Crazyflie with different initializations.

Table 8.1 summarizes the hardware results, which corroborate simulation findings. The DEQ-MPC variants (DEQ-MPC-NN and DEQ-MPC-DEQ) achieved higher average returns compared to the Diff-MPC variants. Notably, DEQ-MPC-DEQ achieved a zero failure rate, successfully completing all trials without collisions or crashes. While DEQ-MPC-NN also showed strong performance, it experienced a collision on one of the runs. We observed that the OptiTrack Mocap measurements, while generally accurate, occasionally exhibited noise and transient fluctuations. Qualitatively, the DEQ-MPC based policies, particularly DEQ-MPC-DEQ, demonstrated greater robustness to these real-world imperfections. This robustness manifested in smoother flight paths and a better ability to maintain stability and track the desired trajectory despite state estimation noise, ultimately contributing to their superior performance in terms of both higher average returns and lower (or zero) failure rates. The Diff-MPC policies appeared more susceptible to these disturbances, leading to less reliable performance and a higher incidence of failures.

8.5.4 Notes on convergence and inference time

ADMM Convergence

Our treatment of the joint system as a DEQ allows us to borrow results from [Bai et al., 2021, Winston and Kolter, 2020] to ensure convergence of the fixed point iteration. Specifically, if we assume the joint Jacobian of the ADMM fixed point iteration is strongly monotone with smoothness parameter m and Lipschitz constant L, then by

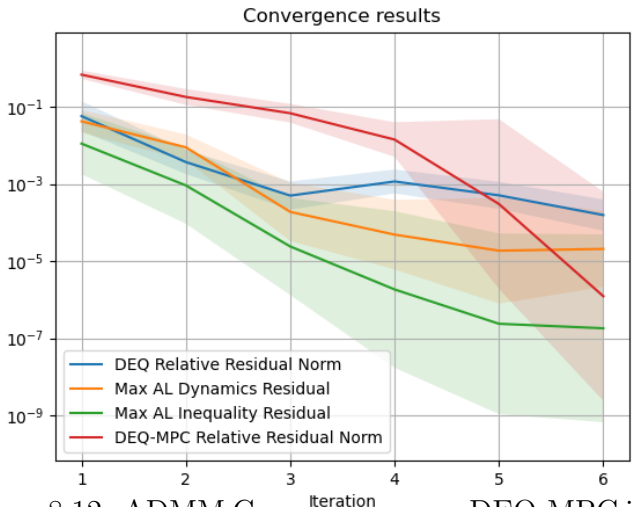


Figure 8.12: ADMM Convergence over DEQ-MPC iterations. All relevant constraint and iterative residual/relative residuals converge to $< 1e-3$ tolerance.

standard arguments (see e.g., Section 5.1 of [Ryu and Boyd, 2016]), the fixed point iteration with step size $\alpha < m/L^2$ will converge. However, going from the strong monotonicity assumption on the joint fixed point iterations to specific assumptions on the network or the optimization problem is less straightforward. But, in practice a wide suite of techniques have been used to ensure that such fixed points exist and can be found using relatively few fixed point iterations.

In fact, for all of our experiments, we that the problem converges within 6 ADMM iterations once trained. As an example, we show the convergence behavior of the joint optimization problem of the trained DEQ-MPC-DEQ model in the QuadPole-obstacle avoidance environment in Figure 8.12. We find that the maximum constraint residuals for both equality (dynamics) and inequality (obstacle avoidance and control limits) constraints, as well as the relative fixed-point residuals of the DEQ network and the DEQ-MPC iterates converge to $<$ 1e$-$3 tolerance within 6 DEQ-MPC iterations. We observe similar plots for the other systems we tested as well with the model often converging faster in the easier systems.

In fact, across all our experiments, we barely faced any issues with convergence of the optimization problem itself as long as we followed the best practices while designing the network architectures (used appropriate skip connections and normalization layers as is common in DEQ architectures [Bai et al., 2019a, 2021, Geng et al., 2021, Teed and Deng, 2021]) and the AL solver (using a merit function-based line search for monotonic convergence and appropriate solver initializations as described earlier). However, when dealing with more complicated (more ill-conditioned/non-convex) problems, it is conceivable that we might face convergence issues. In such cases, we might advise increasing the number of inner AL iterations as well as outer DEQ-MPC iterations to ensure convergence but we did not face those issues at least for the problems we tested.

Computational Cost & Inference Time We'd like to stress that the inference times during the MPC rollouts are very similar across our models and the baselines given that all of them perform one network inference and one inner optimization loop (2 AL iterations). For instance, the total inference times for our DEQ-MPC-DEQ (86.6ms) and the Diff-MPC-DEQ baseline (88.8ms) are comparable on our machine.

Similarly, the NN variants, DEQ-MPC-NN (71.2ms) and Diff-MPC-NN (61.6ms), also show similar timings. The solver clearly consumes the majority of the time (72ms-75ms for DEQ variants and 58-68 for NN variants). But the solver code is currently a PyTorch implementation and not optimized for speed. Production solver implementations can be made much faster but require significant systems effort which we leave for future work given that is not the focus of our work. However, even with our implementation, we're able to run the policies at > 10 Hz frequency as mentioned in our hardware experiments.

8.6 Discussion

Discussion. Our experimental results highlight several key advantages of DEQ-MPC over differentiable MPC layers. The performance gap between DEQ-MPC variants and Diff-MPC becomes increasingly apparent as task complexity increases, whether through harder constraints, longer planning horizons, or increased problem sensitivity. A particularly promising aspect of DEQ-MPC is its favorable scaling behavior. Unlike Diff-MPC variants which show signs of performance saturation, DEQ-MPC models continue to improve with increasing dataset size and network capacity. This suggests potential for exploiting scaling laws in robotics applications. Furthermore, DEQ-MPC's effectiveness in warm-starting scenarios, requiring fewer augmented Lagrangian iterations while maintaining performance, offers significant practical advantages for real-world deployment. This advantage was also evident in our hardware experiments, where DEQ-MPC methods demonstrated superior reliability. Interestingly, there exist trade-offs even between the DEQ-MPC variants. While DEQ-MPC-NN performs slightly better on average in simulation, DEQ-MPC-DEQ remains stable across a wider range of conditions compared to DEQ-MPC-NN.

Chapter 9

Conclusions and Future Directions

This thesis has systematically investigated the challenges and opportunities of integrating optimization layers into deep learning frameworks. As outlined in the introduction, these layers serve as powerful task-specific priors, yet their implicit and iterative nature introduces significant hurdles, namely: inference and representational inefficiencies, destabilizing training dynamics, and biases from modeling inaccuracies. Our work has demonstrated that by carefully co-designing the numerical methods and network architectures, these challenges can be overcome, unlocking substantial gains in performance, efficiency, and robustness.

The contributions of this thesis provide a multi-faceted approach to these problems.

To address representational and computational inefficiencies, we proposed unifying the network inference and the optimization problem into a joint fixed-point iteration. In Chapter 3 with input optimization for Deep Equilibrium (DEQ) models and Chapter 8 with DEQ-MPC, we showed this joint formulation leads to richer representations, naturally accommodates warm-starting, and yields significant computational speedups.

To mitigate training dynamics challenges, we focused on understanding and reducing gradient variance. In Chapter 4, we diagnosed the sources of variance in bundle adjustment layers for visual odometry and introduced targeted modifications that stabilized and accelerated training by up to 2.5x. In Chapter 6, our Critic Gradient-

based Actor-Critic (CGAC) method leveraged GPU parallelism to reduce variance in high-dimensional reinforcement learning problems, leading to faster and more stable convergence.

To counteract biases from modeling inaccuracies, we developed a method to leverage approximate models for policy learning. In Chapter 5, we propoosed the Value-Gradient Update, a general update rule that allows us to rely on real-world data for evaluation, while using the approximate model jacobians to guide the update direction, striking a balance that enhanced sample efficiency and zero-shot transfer performance.

Finally, in Chapter 7, the VINSat system served as a capstone application, demonstrating how these principles can be integrated into a complex, end-to-end system for a challenging real-world problem—satellite orbit determination—achieving state-of-the-art results.

Collectively, these contributions provide a principled framework for designing, training, and deploying hybrid models that effectively combine the expressive power of deep learning with the rigor of classical optimization.

9.1 Future Directions

The work presented in this thesis addresses foundational challenges in the use of optimization layers, but it also illuminates a rich landscape for future research. In the five years since this doctoral research began, the use of optimization layers has transitioned from a niche academic pursuit to a key component in a growing number of large-scale applications. The principles and methods developed here can be extended and applied to a wide array of established and emerging domains.

9.1.1 Broadening the Application Landscape

The techniques for stable and efficient integration of optimization layers are directly applicable to numerous complex fields:

3D Vision and Robotics: While this thesis focused on visual odometry, the broader field of 3D vision is ripe for these methods given the sensor models and geometric models are very accurate. Problems like 4D reconstruction of dynamic scenes, robust object pose estimation, and multi-modal sensor fusion (e.g., vision with tactile or sonar data) all rely on underlying geometric and physical principles. Using optimization layers to enforce these principles as inductive biases can dramatically improve the representational capacity and data efficiency of learned models, especially in data-poor domains like underwater or space robotics.

Motion Planning and Control: In safety-critical robotics applications, the ability to enforce hard constraints is paramount. Optimization layers provide a formal mechanism for ensuring safety and stability. We believe that the algorithms and architectures are now mature enough that future work could focus on scaling these methods to larger systems with higher degrees of freedom and partial observations, and expand the application domains further to problems such as in heavy machinery operation or robots in space and underwater.

Large-Scale Infrastructure and Scientific Simulation: Domains like electrical grid management and climate modeling are characterized by large-scale, safety-critical systems with strong underlying physical laws. Integrating optimization layers into deep learning models for forecasting and control in these areas could lead to more accurate, reliable, and physically consistent predictions, which is crucial for decision-making. Similarly, learning simulators for weather or fluid dynamics could benefit from physics-informed optimization layers to prevent divergence and ensure long-horizon stability.

Automated Machine Learning (AutoML): The process of training a large model is itself an optimization problem parameterized by a set of hyperparameters that often also need to be learnt or optimized for optimal training. These hyper-parameter optimization problems include learning optimal training schedules, dataset curricula, or even optimizer hyper-parameters like learning rates on the fly. These problems have gained increasing salience over the past few years as the training runs for large models have gotten increasingly expensive making the choice of hyperparameters very critical. This represents a powerful new direction for making the training of complex models more efficient and autonomous.

9.1.2 Emerging Research Frontiers

Beyond extending current applications, optimization layers are poised to be instrumental in tackling a new generation of challenges at the forefront of AI research:

Agentic Markets and Game-Theoretic Learning: With the rise of Large Language Models (LLMs) as autonomous agents, we are on the cusp of complex digital economies where these agents interact and compete. Training agents to behave optimally in such markets involves solving highly non-convex, partially observable, game-theoretic optimization problems. Naively differentiating through market dynamics to tune an LLM's policy is unlikely to succeed. The development of robust algorithms for this domain will require a deep integration of game theory, optimization, and policy learning, where the principles of stable gradient propagation and joint inference will be critical.

Next-Generation Meta-Learning: A fundamental limitation of current large models is their inability to learn new skills efficiently in a generalizable manner without forgetting other knowledge. Gradient-based meta-learning methods, which aim to "learn to learn," offer a path forward but have been hindered by computational and representational inefficiencies. By reformulating meta-learning as a joint optimization problem—akin to the input optimization based methods in this thesis—it may be

possible to develop new algorithms that are efficient enough to enable large models to continuously and rapidly adapt to new tasks and workflows.

9.1.3 Concluding Remarks

The integration of optimization layers with deep learning represents a pivotal shift from purely data-driven models to hybrid systems that synergize learning with domain knowledge. The work in this thesis has provided novel methods to make this integration more efficient, stable, and robust. As AI continues to be deployed in increasingly complex and safety-critical domains, this paradigm of principled, model-aware learning will become indispensable. The future of intelligent systems lies not in replacing classical methods with deep learning, but in building a deeper, more unified connection between them. The ultimate goal is to create models that are not only powerful predictors but are also robust, interpretable, and aligned with the physical and logical constraints of the world they aim to understand and shape.

Bibliography

Space environment statistics. URL https://sdup.esoc.esa.int/discosweb/statistics/. 7.1

GNSS Receiver Module (GPSRM 1) Kit. http://www.pumpkinspace.com/store/. 7.2.1, 7.1

MTCR Annex - MTCR. https://www.mtcr.info/en/mtcr-annex. 7.2.1

OEM719. URL https://novatel.com/products/receivers/gnss-gps-receiver-boards/oem719. 7.2.1, 7.1

Planet Labs public orbital ephemerides. https://ephemerides.planet-labs.com/. 7.2.2, 7.1

P. Abbeel, M. Quigley, and A. Ng. Using inaccurate models in reinforcement learning. *Proceedings of the 23rd international conference on Machine learning*, 2006. 5.2

J. Adler and O. Öktem. Solving ill-posed inverse problems using iterative deep neural networks. *Inverse Problems*, 2017. 4.1

J. Adler and O. Öktem. Learned primal-dual reconstruction. *IEEE transactions on medical imaging*, 2018. 4.1

N. Agarwal, E. Hazan, A. Majumdar, and K. Singh. A regret minimization approach to iterative learning control. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 100–109. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/agarwal21b.html. 5.2, 5.3.3

A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter. Differentiable convex optimization layers. *Advances in neural information processing systems*, 32, 2019. 4.2, 8.2

A. Agrawal, S. Barratt, S. Boyd, and B. Stellato. Learning convex optimization control policies. In *Learning for Dynamics and Control*, pages 361–373. PMLR, 2020. 8.1, 8.2

A. Allshire, M. Mittal, V. Lodaya, V. Makoviychuk, D. Makoviichuk, F. Widmaier,

M. Wüthrich, S. Bauer, A. Handa, and A. Garg. Transferring dexterous manipulation from gpu simulation to a remote real-world trifinger. *ArXiv*, abs/2108.09779, 2021. 6.2

A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada. Control barrier functions: Theory and applications. In *2019 18th European Control Conference (ECC)*, pages 3420–3431, 2019. doi: 10.23919/ECC.2019.8796030. 8.1

B. Amos and J. Z. Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning (ICML)*, pages 136–145, 2017. 2.1.1, 3.2, 4.2, 8.2

B. Amos, L. Xu, and J. Z. Kolter. Input convex neural networks. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 146–155. PMLR, 06–11 Aug 2017. URL http://proceedings.mlr.press/v70/amos17b.html. 3.2

B. Amos, I. Jimenez, J. Sacks, B. Boots, and J. Z. Kolter. Differentiable mpc for end-to-end planning and control. *Advances in neural information processing systems*, 31, 2018a. 4.2, 8.1, 8.2, 8.3.1

B. Amos, I. Jimenez, J. Sacks, B. Boots, and J. Z. Kolter. Differentiable mpc for end-to-end planning and control. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018b. URL https://proceedings.neurips.cc/paper/2018/file/ba6d843eb4251a4526ce65d1807a9309-Paper.pdf. 3.1

D. G. Anderson. Iterative procedures for nonlinear integral equations. *Journal of the ACM (JACM)*, 12(4):547–560, 1965. 2.2.1

O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020. 6.2

R. Antonova, J. Yang, K. M. Jatavallabhula, and J. Bohg. Rethinking optimization with differentiable simulation from a global perspective. In *Conference on Robot Learning*, pages 276–286. PMLR, 2023. 4.2, 8.2, 8.4.2

M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz. Reinforcement learning through asynchronous advantage actor-critic on a GPU. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=r1VGvBcxl. 6.2

S. Bai, J. Z. Kolter, and V. Koltun. Deep equilibrium models. *Advances in neural information processing systems*, 32, 2019a. 2.2, 8.3.2, 8.4.2, 8.5.4

S. Bai, J. Z. Kolter, and V. Koltun. Deep equilibrium models. *Advances in Neural Information Processing Systems*, 32:690–701, 2019b. 2.2.1, 3.1, 3.2, 3.3.1

S. Bai, V. Koltun, and J. Z. Kolter. Multiscale deep equilibrium models. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://github.com/locuslab/mdeq. 2.2.1, 3.1, 3.2, 3.4

S. Bai, V. Koltun, and Z. Kolter. Stabilizing equilibrium models by jacobian regularization. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 554–565. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/bai21b.html. 3.3.2, 3.3.5, 8.5.4

S. Barratt. On the differentiability of the solution to convex optimization problems. *arXiv preprint arXiv:1804.05098*, 2018. 2.1.1

J. T. Barron. A general and adaptive robust loss function. In *CVPR*, June 2019a. 4.5

J. T. Barron. A general and adaptive robust loss function. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4331–4339, 2019b. 7.3.4

A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13:834–846, 1983. 6.2

Y. Beck and M. Schmidt. A gentle and incomplete introduction to bilevel optimization. 2021. 4.2

C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019. 5.1

D. Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena scientific, 2012. 5.2

D. Bertsekas. *Reinforcement learning and optimal control*. Athena Scientific, 2019. 5.2

J. T. Betts. *Practical Methods for Optimal Control Using Nonlinear Programming*. Society for Industrial and Applied Mathematics, 2001. 5.2

M. Bhardwaj, B. Boots, and M. Mukadam. Differentiable gaussian process motion planning. In *2020 IEEE international conference on robotics and automation (ICRA)*, pages 10598–10604. IEEE, 2020. 4.1, 8.2

S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee. Incremental natural actor-critic algorithms. In *NIPS*, 2007. 6.2

B. Bianchini, M. Halm, and M. Posa. Simultaneous learning of contact and continuous dynamics. *arXiv preprint arXiv:2310.12054*, 2023. 4.2

L. T. Biegler and V. M. Zavala. Large-scale nonlinear programming using ipopt: An integrating framework for enterprise-wide dynamic optimization. *Computers & Chemical Engineering*, 33(3):575–582, 2009. 8.4.2

K. Binmore and J. Davies. *Calculus: concepts and methods*. Cambridge University Press, 2001. 3.3.1

P. Bojanowski, A. Joulin, D. Lopez-Paz, and A. Szlam. Optimizing the latent space of generative networks. *arXiv preprint arXiv:1707.05776*, 2017. 3.2, 3.4.1

A. Bora, A. Jalal, E. Price, and A. G. Dimakis. Compressed sensing using generative models. In *International Conference on Machine Learning (ICML)*, pages 537–546, 2017. 3.1, 3.1, 3.2, 3.4, 3.4.2, 3.4

S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, July 2011. ISSN 1935-8237, 1935-8245. doi: 10.1561/2200000016. URL https://www.nowpublishers.com/article/Details/MAL-016. Publisher: Now Publishers, Inc. 8.4.2

C. G. Broyden. A class of methods for solving nonlinear simultaneous equations. *Mathematics of Computation*, 1965. 2.2.1, 3.2

F. Bünning, A. Schalbetter, A. Aboudonia, M. H. de Badyn, P. Heer, and J. Lygeros. Input convex neural networks for building mpc. *ArXiv*, abs/2011.13227, 2020. 3.2

K. Burnett, D. J. Yoon, A. P. Schoellig, and T. D. Barfoot. Radar odometry combining probabilistic estimation and unsupervised feature learning. In *Robotics: Science and Systems*, 2021. 4.2

M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart. The euroc micro aerial vehicle datasets. *The International Journal of Robotics Research*, 2016. (document), 4.6, 4.2, 4.6.3

C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on robotics*, 2016. 4.2

S. Caldwell. 12.0 Identification and Tracking Systems. http://www.nasa.gov/smallsat-institute/sst-soa/identification-and-tracking-systems, Oct. 2021. 7.2.2

C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. Montiel, and J. D. Tardós. Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. *IEEE Transactions on Robotics*, 2021. **??**, 4.6.3

J. R. Chang, C.-L. Li, B. Póczos, B. Vijaya Kumar, and A. C. Sankaranarayanan. One network to solve them all — solving linear inverse problems using deep projection models. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 5889–5898, 2017. doi: 10.1109/ICCV.2017.627. 3.1, 3.2, 3.4.2, 3.4.2

N. Chebrolu, T. Läbe, O. Vysotska, J. Behley, and C. Stachniss. Adaptive robust kernels for non-linear least squares problems. *IEEE Robotics and Automation Letters*, 2021. 4.5

B. Chen, A. Parra, J. Cao, N. Li, and T.-J. Chin. End-to-end learnable geometric vision by backpropagating pnp optimization. In *CVPR*, 2020a. 4.2

C. Chen, B. Wang, C. X. Lu, N. Trigoni, and A. Markham. A survey on deep learning for localization and mapping: Towards the age of spatial machine intelligence, 2020b. 4.2

R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf. 3.2

S. Chen, B. Zhang, M. W. Mueller, A. Rai, and K. Sreenath. Learning torque control for quadrupedal locomotion. *ArXiv*, abs/2203.05194, 2022a. 6.2

T. Chen, J. Xu, and P. Agrawal. A system for general in-hand object re-orientation. In *5th Annual Conference on Robot Learning*, 2021a. URL https://openreview.net/forum?id=7uSBJDoP7tY. 6.2

T. Chen, J. Xu, and P. Agrawal. A system for general in-hand object re-orientation. In *Conference on Robot Learning*, pages 297–307. PMLR, 2022b. 5.1

X. Chen, J. Hu, C. Jin, L. Li, and L. Wang. Understanding domain randomization for sim-to-real transfer. *arXiv preprint arXiv:2110.03239*, 2021b. 8.1

R. Clark, M. Bloesch, J. Czarnowski, S. Leutenegger, and A. J. Davison. Learning to solve nonlinear least squares for monocular stereo. In *ECCV*, 2018. 4.2

A. V. Clemente, H. N. Castejón, and A. Chandra. Efficient parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1705.04862*, 2017. 6.2

J. Czarnowski, T. Laidlow, R. Clark, and A. J. Davison. Deepfactors: Real-time probabilistic dense monocular slam. *IEEE Robotics and Automation Letters*, 2020. 4.2

T. Degris, P. M. Pilarski, and R. S. Sutton. Model-free reinforcement learning with continuous action in practice. *2012 American Control Conference (ACC)*, pages 2177–2182, 2012. 6.2

B. Denby, E. Ruppel, V. Singh, S. Che, C. Taylor, F. Zaidi, S. Kumar, Z. Manchester, and B. Lucia. Tartan artibeus: A batteryless, computational satellite research platform. 2022. 7.2.1

D. DeTone, T. Malisiewicz, and A. Rabinovich. Superpoint: Self-supervised interest

point detection and description. In *CVPR Workshops*, 2018. 4.6.2

S. Diamond, V. Sitzmann, F. Heide, and G. Wetzstein. Unrolled optimization with deep priors. *arXiv preprint arXiv:1705.08041*, 2017. 3.2

C. Diehl, T. Klosek, M. Krueger, N. Murzyn, T. Osterburg, and T. Bertram. Energy-based potential games for joint motion forecasting and control. In *Conference on Robot Learning*, pages 3112–3141. PMLR, 2023a. 8.2

C. Diehl, T. Klosek, M. Krüger, N. Murzyn, and T. Bertram. On a connection between differential games, optimal control, and energy-based models for multi-agent interactions. *arXiv preprint arXiv:2308.16539*, 2023b. 8.1, 8.2

J. Djolonga and A. Krause. Differentiable learning of submodular models. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/192fc044e74dffea144f9ac5dc9f3395-Paper.pdf. 3.2

A. L. Dontchev, R. T. Rockafellar, and R. T. Rockafellar. *Implicit functions and solution mappings: A view from variational analysis*, volume 616. Springer, 2009. 4.4.2

P. L. Donti, D. Rolnick, and J. Z. Kolter. DC3: A learning method for optimization with hard constraints. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=V1ZHVxJ6dSS. 4.2, 8.2

S. S. Du and W. Hu. Linear convergence of the primal-dual gradient method for convex-concave saddle point problems without strong convexity. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 196–205. PMLR, 2019. 3.3.3

E. Dupont, A. Doucet, and Y. W. Teh. Augmented neural odes. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/21be9a4bd4f81549a9d1d241981cec3c-Paper.pdf. 3.2

L. El Ghaoui, F. Gu, B. Travacca, A. Askari, and A. Y. Tsai. Implicit deep learning. *arXiv preprint arXiv:1908.06315*, 2, 2019. 3.2

H. C. Elman and G. H. Golub. Inexact and preconditioned uzawa algorithms for saddle point problems. *SIAM Journal on Numerical Analysis*, 31(6):1645–1661, 1994. 3.3.3

J. Engel, T. Schöps, and D. Cremers. Lsd-slam: Large-scale direct monocular slam. In *ECCV*, 2014. 4.1

J. Engel, V. Koltun, and D. Cremers. Direct sparse odometry. *IEEE transactions on*

pattern analysis and machine intelligence, 2017. 4.1, ??, 4.6.3

L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pages 1407–1416. PMLR, 2018. 6.2

Z. Fan, Y. Zhu, Y. He, Q. Sun, H. Liu, and J. He. Deep learning on monocular object pose detection and tracking: A comprehensive overview. *ACM Computing Surveys*, 2022. 4.2

H.-r. Fang and Y. Saad. Two classes of multisecant methods for nonlinear acceleration. *Numerical Linear Algebra with Applications*, 16(3):197–221, 2009. 3.4.5

C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR, 06–11 Aug 2017. URL http://proceedings.mlr.press/v70/finn17a.html. 3.1, 3.2, 3.4, 3.4.4

M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. 7.2.3

C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem. Brax - a differentiable physics engine for large scale rigid body simulation, 2021. URL http://github.com/google/brax. 6.1, 6.2, 6.4.2

L. F. T. Fu and M. Fallon. Batch differentiable pose refinement for in-the-wild camera/lidar extrinsic calibration. In *Conference on Robot Learning*, 2023. 4.1

S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018. 6.4.1

S. W. Fung, H. Heaton, Q. Li, D. McKenzie, S. Osher, and W. Yin. Fixed point networks: Implicit depth models with Jacobian-free backprop. *arXiv:2103.12803*, 2021a. 2.2.1

S. W. Fung, H. Heaton, Q. Li, D. Mckenzie, S. J. Osher, and W. Yin. Jfb: Jacobian-free backpropagation for implicit networks. In *AAAI Conference on Artificial Intelligence*, 2021b. URL https://api.semanticscholar.org/CorpusID:238198721. 2.2.2, 8.4.3

S. W. Fung, H. Heaton, Q. Li, D. McKenzie, S. Osher, and W. Yin. Jfb: Jacobian-free backpropagation for implicit networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 6648–6656, 2022. 4.2

Z. Geng, X.-Y. Zhang, S. Bai, Y. Wang, and Z. Lin. On training implicit models.

Advances in Neural Information Processing Systems, 34:24247–24260, 2021. 2.2.2, 8.4.3, 8.5.4

P. Ghosh, M. S. Sajjadi, A. Vergari, M. Black, and B. Schölkopf. From variational to deterministic autoencoders. In *International Conference on Learning Representations*, 2020. 3.4.1, ??

P. E. Gill, W. Murray, and M. A. Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM review*, 47(1):99–131, 2005. 8.4.2

S. Gillen, M. Molnar, and K. Byl. Combining deep reinforcement learning and local control for the acrobot swing-up and balance task. *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 4129–4134, 2020. 5.5.1

D. Gilton, G. Ongie, and R. Willett. Neumann networks for linear inverse problems in imaging. *IEEE Transactions on Computational Imaging*, 6:328–343, 2020. doi: 10.1109/TCI.2019.2948732. 3.2, 3.4

D. Gilton, G. Ongie, and R. Willett. Deep equilibrium architectures for inverse problems in imaging. *arXiv preprint arXiv:2102.07944*, 2021. 3.2, 3.4.2

P. W. Glynn. Importance sampling for markov chains: Asymptotics for the variance. *Stochastic Models*, 10(4):701–717, 1994. 5.2

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014a. URL https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf. 3.4

I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014b. 3.1, 3.2

N. Gorelick, M. Hancher, M. Dixon, S. Ilyushchenko, D. Thau, and R. Moore. Google Earth Engine: Planetary-scale geospatial analysis for everyone. *Remote Sensing of Environment*, 2017. doi: 10.1016/j.rse.2017.06.031. 7.4.1

S. Gould, R. Hartley, and D. Campbell. Deep declarative networks: A new hope. *arXiv:1909.04866*, 2019. 3.2

S. Gould, R. Hartley, and D. Campbell. Deep declarative networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(8):3988–4004, 2021. 8.2

W. Grathwohl, R. T. Q. Chen, J. Bettencourt, and D. Duvenaud. Scalable reversible generative models with free-form continuous dynamics. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJxgknCcK7. 3.2

E. Graves, E. Imani, R. Kumaraswamy, and M. White. Off-policy actor-critic with

emphatic weightings. *ArXiv*, abs/2111.08172, 2021. 6.2

E. Greensmith, P. L. Bartlett, and J. Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. In *Journal of machine learning research*, 2004. 6.2

K. Gregor and Y. LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th international conference on international conference on machine learning*, pages 399–406, 2010. 3.2, 3.4

I. Grondman, L. Buşoniu, G. A. D. Lopes, and R. Babuka. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42:1291–1307, 2012. 6.2

S. Gu, T. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine. Q-prop: Sample-efficient policy gradient with an off-policy critic. *arXiv preprint arXiv:1611.02247*, 2016. 6.2, 6.3.3, 6.6

S. S. Gu, T. P. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine. Q-prop: Sample-efficient policy gradient with an off-policy critic. *ArXiv*, abs/1611.02247, 2017a. 6.2

S. S. Gu, T. P. Lillicrap, Z. Ghahramani, R. E. Turner, B. Scholkopf, and S. Levine. Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning. In *NIPS*, 2017b. 6.2, 6.3.3, 6.6

A. Gupta, A. Pacchiano, Y. Zhai, S. Kakade, and S. Levine. Unpacking reward shaping: Understanding the benefits of reward engineering on sample complexity. *Advances in Neural Information Processing Systems*, 35:15281–15295, 2022. 8.1

S. Gurumurthy, S. Kumar, and K. Sycara. Mame: Model-agnostic meta-exploration. In *Conference on Robot Learning*, pages 910–922. PMLR, 2020. 3.2

S. Gurumurthy, S. Bai, Z. Manchester, and J. Z. Kolter. Joint inference and input optimization in equilibrium networks. *Advances in Neural Information Processing Systems*, 34:16818–16832, 2021. 3, 8.2

S. Gurumurthy, J. Z. Kolter, and Z. Manchester. Deep off-policy iterative learning control. In *Learning for Dynamics and Control Conference*, pages 639–652. PMLR, 2023a. 5, 8.2

S. Gurumurthy, Z. Manchester, and J. Z. Kolter. Practical critic gradient based actor critic for on-policy reinforcement learning. In *5th Annual Learning for Dynamics & Control Conference*, 2023b. URL https://openreview.net/forum?id=ddl_4qQKFmY. 6

S. Gurumurthy, K. Ram, B. Chen, Z. Manchester, and Z. Kolter. From variance to veracity: Unbundling and mitigating gradient variance in differentiable bundle

adjustment layers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 27507–27516, 2024. 4, 8.2

T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, 2018a. 6.1, 6.2, 6.2, 6.3, 6.4.1, 6.5

T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018b. 5.3.2, 5.4, 5.4.4

A. Handa, A. Allshire, V. Makoviychuk, A. Petrenko, R. Singh, J. Liu, D. Makoviichuk, K. V. Wyk, A. Zhurkevich, B. Sundaralingam, Y. S. Narang, J.-F. Lafleche, D. Fox, and G. State. Dextreme: Transfer of agile in-hand manipulation from simulation to reality. *ArXiv*, abs/2210.13702, 2022. 6.2

N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa. Learning continuous control policies by stochastic value gradients. *Advances in neural information processing systems*, 28, 2015. 5.2

M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *arXiv preprint arXiv:1706.08500*, 2017. 3.4.1

M. Hong, M. Razaviyayn, and J. Lee. Gradient primal-dual algorithm converges to second-order stationary solution for nonconvex distributed optimization over networks. In *International Conference on Machine Learning*, pages 2009–2018. PMLR, 2018. 3.3.3

D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. van Hasselt, and D. Silver. Distributed prioritized experience replay. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=H1Dy---0Z. 6.2, 6.5.2

T. A. Howell, B. E. Jackson, and Z. Manchester. Altro: A fast solver for constrained trajectory optimization. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7674–7679. IEEE, 2019. 8.4.3

T. A. Howell, S. Le Cleac'h, J. Z. Kolter, M. Schwager, and Z. Manchester. Dojo: A differentiable simulator for robotics. *arXiv preprint arXiv:2203.00806*, 9, 2022a. 4.2

T. A. Howell, K. Tracy, S. Le Cleac'h, and Z. Manchester. Calipso: A differentiable solver for trajectory optimization with conic and complementarity constraints. In *The International Symposium of Robotics Research*, pages 504–521. Springer, 2022b. 4.2, 8.2

P. Huang, M. Xu, F. Fang, and D. Zhao. Robust reinforcement learning as a stackelberg game via adaptively-regularized adversarial training. *arXiv preprint*

arXiv:2202.09514, 2022. 8.4.2

J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019. 5.1

E. Imani, E. Graves, and M. White. An off-policy policy gradient theorem using emphatic weightings. In *NeurIPS*, 2018. 6.2

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, 2015. 3.4

S. Iwase, X. Liu, R. Khirodkar, R. Yokota, and K. M. Kitani. Repose: Fast 6d object pose refinement via deep texture rendering. In *ICCV*, 2021. 4.2

B. E. Jackson, J. H. Lee, K. Tracy, and Z. Manchester. Data-efficient model learning for control with jacobian-regularized dynamic mode decomposition. In *6th Annual Conference on Robot Learning*, 2022. URL https://openreview.net/forum?id=EDOG14V3WeH. 5.2, 5.5.1

K. M. Jatavallabhula, G. Iyer, and L. Paull. δ slam: Dense slam meets automatic differentiation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020. 4.2

Y. Jeon, M. Lee, and J. Y. Choi. Differentiable forward and backward fixed-point iteration layers. *IEEE Access*, 9:18383–18392, 2021. 3.2

W. Jin, Z. Wang, Z. Yang, and S. Mou. Pontryagin differentiable programming: An end-to-end learning and control framework. *Advances in Neural Information Processing Systems*, 33:7979–7992, 2020. 8.2

Y. Kanazawa and K. Kanatani. Do we really have to consider covariance matrices for image features? In *ICCV*, 2001. 4.2

Z. Kang, B. Tapley, S. Bettadpur, J. Ries, P. Nagel, and R. Pastor. Precise orbit determination for the grace mission using only gps data. *Journal of Geodesy*, 80: 322–331, 2006. 7.2.1

H. Kannan, A. Kurakin, and I. Goodfellow. Adversarial logit pairing. *ArXiv*, abs/1803.06373, 2018. 3.2

T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020. 3.2

K. Kawaguchi. On the theory of implicit deep learning: Global convergence with implicit layers. In *International Conference on Learning Representations (ICLR)*, 2021. 3.2

D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015a. 3.4.1

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015b. URL http://arxiv.org/abs/1412.6980. 3.3.4

D. P. Kingma and M. Welling. Auto-encoding variational bayes. *International Conference on Learning Representations*, 2014. 3.4, 3.4.1, ??, 6.3.2

J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013. 5.2

L. Koestler, N. Yang, N. Zeller, and D. Cremers. Tandem: Tracking and dense mapping in real-time using deep multi-view stereo. In *Conference on Robot Learning*, 2022. 4.2

J. Z. Kolter and G. Manek. Learning stable deep dynamics models. In *NeurIPS*, 2019. 3.2

J. Z. Kolter and E. Wong. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *ICML*, 2018. 3.2

J. Z. Kolter, D. Duvenaud, and M. Johnson. Deep implicit layers - neural odes, deep equilibirum models, and beyond. 2020a. URL http://implicit-layers-tutorial.org/. 3.2

M. Z. Kolter, D. Duvenaud, and M. Johnson. Deep implicit layers-neural odes, deep equilibirum models, and beyond, 2020. *NeurIPS Tutorial*, 2020b. 2.1.1

V. R. Konda and J. N. Tsitsiklis. Actor-critic algorithms. In *NIPS*, 1999. 6.2

S. G. Krantz and H. R. Parks. *The implicit function theorem: History, theory, and applications*. Springer, 2012. 3.2, 3.3.1

A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009. 3.4.3

E. Kulu. Nanosatellite launch forecasts-track record and latest prediction. 2022. 7.1

Y. Labbé, J. Carpentier, M. Aubry, and J. Sivic. Cosypose: Consistent multi-view multi-object 6d pose estimation. In *ECCV*, 2020. 4.2

B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015. 3.4.4

B. Landry, Z. Manchester, and M. Pavone. A differentiable augmented lagrangian method for bilevel nonlinear optimization. *arXiv preprint arXiv:1902.03319*, 2019. 8.2

S. Le Cleac'h, T. A. Howell, S. Yang, C.-Y. Lee, J. Zhang, A. Bishop, M. Schwager, and

Z. Manchester. Fast contact-implicit model predictive control. *IEEE Transactions on Robotics*, 2024. 8.4.2, 8.4.3

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 3.4.3

V. Lepetit, F. Moreno-Noguer, and P. Fua. Ep n p: An accurate o (n) solution to the p n p problem. *IJCV*, 2009. 4.1

F. Letizia. Results from esa's annual space environment report, july 2019, presented as a key-note address at the advanced maui optical and space surveillance technologies conference, held in wailea, maui, hawaii, september 2019. used by permission from esa, 2019. 7.1

K. Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, 2(2):164–168, 1944. 7.3.4

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015. 5.3.2

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. M. O. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2016. 6.1, 6.2

L. Lipson, Z. Teed, A. Goyal, and J. Deng. Coupled iterative refinement for 6d multi-object pose estimation. In *CVPR*, 2022. (document), 4.1, 4.1, 4.2, 8.2

Y. Liu, P.-L. Bacon, and E. Brunskill. Understanding the curse of horizon in off-policy evaluation via conditional importance sampling. In *International Conference on Machine Learning*, pages 6184–6193. PMLR, 2020. 5.2

Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015. 3.4.1

D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999. 7.2.3

D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 2004. 4.6.2

C. Lu, J. Chen, C. Li, Q. Wang, and J. Zhu. Implicit normalizing flows. In *ICLR*, 2021. 3.2

A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. *ArXiv*, abs/1706.06083, 2018. 3.1, 3.2, 3.4

A. Makkuva, A. Taghvaei, S. Oh, and J. Lee. Optimal transport mapping via input convex neural networks. In *ICML*, 2020. 3.2

D. Makoviichuk and V. Makoviychuk. rl-games: A high-performance framework for reinforcement learning. https://github.com/Denys88/rl_games, May 2022. 6.5

V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State. Isaac gym: High performance GPU based physics simulation for robot learning. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=fgFBtYgJQX_. 6.1, 6.2, 6.4.2, 6.5, 6.5

Z. R. Manchester, J. I. Lipton, R. J. Wood, and S. Kuindersma. A variable forward-sweep wing design for enhanced perching in micro aerial vehicles. In *55th AIAA Aerospace Sciences Meeting*, page 0011, 2017. 5.5.1

D. Q. Mayne. Model predictive control: Recent developments and future promise. *Automatica*, 50(12):2967–2986, 2014. ISSN 0005-1098. doi: https://doi.org/10.1016/j.automatica.2014.10.128. URL https://www.sciencedirect.com/science/article/pii/S0005109814005160. 5.2

K. McCleary, S. Gurumurthy, P. R. Fisch, S. Tayal, Z. Manchester, and B. Lucia. Vinsat: Solving the lost-in-space problem with visual-inertial navigation. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11774–11780. IEEE, 2024. 7

L. Metz, C. D. Freeman, S. S. Schoenholz, and T. Kachman. Gradients are not all you need. *arXiv preprint arXiv:2111.05803*, 2021. 4.2, 5.2

Z. Min, Y. Yang, and E. Dunn. Voldor: Visual odometry from log-logistic dense optical flow residuals. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020. 4.2

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015. 6.2, 6.4.2

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016. 6.1, 6.2, 6.2, 6.3.3, 6.4.1

K. L. Moore. Iterative learning control for deterministic systems. 2012. 5.2

M. A. Z. Mora, M. Peychev, S. Ha, M. Vechev, and S. Coros. Pods: Policy optimization via differentiable simulation. In *International Conference on Machine Learning*, pages 7805–7817. PMLR, 2021. 5.2

F. L. Mueller, A. P. Schoellig, and R. D'Andrea. Iterative learning of feed-forward corrections for high-performance tracking. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3276–3281. IEEE, 2012. 5.1

D. Muhle, L. Koestler, K. M. Jatavallabhula, and D. Cremers. Learning correspondence uncertainty via differentiable nonlinear least squares. In *CVPR*, 2023. 4.2

M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE transactions on pattern analysis and machine intelligence*, 36(11): 2227–2240, 2014. 7.2.3

R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 2015. 4.1

A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015. 6.2

K. Nguyen, S. Schoedel, A. Alavilli, B. Plancher, and Z. Manchester. Tinympc: Model-predictive control on resource-constrained microcontrollers. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–7. IEEE, 2024. 8.4.3

A. Nichol and J. Schulman. Reptile: a scalable metalearning algorithm. *arXiv: Learning*, 2018. 3.4.4

J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, 2e edition, 2006. 8.4.2

G. Ongie, A. Jalal, C. A. Metzler, R. G. Baraniuk, A. G. Dimakis, and R. Willett. Deep learning techniques for inverse problems in imaging. *IEEE Journal on Selected Areas in Information Theory*, 1(1):39–56, 2020. 3.2

OpenAI. Openai five. `https://blog.openai.com/openai-five/`, 2018. 6.2

N. Papernot, P. D. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a defense to adversarial perturbations against deep neural networks, 2016. In *2016 IEEE Symposium on Security and Privacy (SP)*, 2015. 3.2

A. Parag, S. Kleff, L. Saci, N. Mansard, and O. Stasse. Value learning from trajectory optimization and sobolev descent: A step toward reinforcement learning with superlinear convergence properties. In *International Conference on Robotics and Automation*, 2022. 5.2

K. Park, T. Patten, and M. Vincze. Pix2pose: Pixel-wise coordinate regression of objects for 6d pose estimation. In *ICCV*, 2019. 4.1, 4.2

J. Peters, S. Vijayakumar, and S. Schaal. Natural actor-critic. *Neurocomputing*, 71: 1180–1190, 2008. 6.2

L. Pineda, T. Fan, M. Monge, S. Venkataraman, P. Sodhi, R. T. Chen, J. Ortiz, D. DeTone, A. Wang, S. Anderson, et al. Theseus: A library for differentiable

nonlinear optimization. *Advances in Neural Information Processing Systems*, 35: 3801–3818, 2022. 4.2, 8.2

M. Rad and V. Lepetit. Bb8: A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth. In *ICCV*, 2017. 4.2

S. Radu, M. S. Uludag, S. Speretta, J. Bouwmeester, A. Menicucci, A. Cervone, A. Dunn, and T. Walkinshaw. PocketQube Standard. 2018. 7.2.1

A. Rajeswaran, C. Finn, S. M. Kakade, and S. Levine. Meta-learning with implicit gradients. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/072b030ba126b2f4b2374f342be9ed44-Paper.pdf. 3.2, 3.4, 3.4.4

R. Ranftl and V. Koltun. Deep fundamental matrix estimation. In *ECCV*, 2018. 4.2

M. Revay, R. Wang, and I. R. Manchester. Lipschitz bounded equilibrium networks. *arXiv:2010.01732*, 2020. 2.2.1

L. F. Richardson. Ix. the approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 210 (459-470):307–357, 1911. 3.3.2

D. G. Robertson and J. H. Lee. A least squares formulation for state estimation. *Journal of process control*, 5(4):291–299, 1995. 7.1, 7.3.4

A. Romero, Y. Song, and D. Scaramuzza. Actor-critic model predictive control. *arXiv preprint arXiv:2306.09852*, 2023. 8.2

Y. Rubanova, R. T. Chen, and D. Duvenaud. Latent ODEs for irregularly-sampled time series. *arXiv:1907.03907*, 2019. 3.2

E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *ICCV*, 2011. 4.6.2

N. Rudin, D. Hoeller, P. Reist, and M. Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. *ArXiv*, abs/2109.11978, 2021. 6.2

E. K. Ryu and S. Boyd. Primer on monotone operator methods. *Appl. Comput. Math*, 15(1):3–43, 2016. 3.3.2, 8.5.4

R. Sambharya, G. Hall, B. Amos, and B. Stellato. Learning to warm-start fixed-point optimization algorithms. *Journal of Machine Learning Research*, 25(166):1–46, 2024. 8.2

P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich. Superglue: Learning feature matching with graph neural networks. In *CVPR*, 2020. 4.2

A. P. Schoellig, F. L. Mueller, and R. D'andrea. Optimization-based iterative learning for precise quadrocopter trajectory tracking. *Autonomous Robots*, 33(1):103–127, 2012. 5.1

A. Schöllig and R. D'Andrea. Optimization-based iterative learning control for trajectory tracking. In *2009 European Control Conference (ECC)*, pages 1505–1510. IEEE, 2009. 5.2, 5.3.3

J. L. Schonberger and J.-M. Frahm. Structure-from-motion revisited. In *CVPR*, 2016. **??**, 4.6.3

L. Schott, J. Delas, H. Hajri, E. Gherbi, R. Yaich, N. Boulahia-Cuppens, F. Cuppens, and S. Lamprier. Robust deep reinforcement learning through adversarial attacks and training: A survey. *arXiv preprint arXiv:2403.00420*, 2024. 8.1

J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 Jul 2015a. PMLR. 6.1, 6.2, 6.3.3, 6.4.1

J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b. 6.3.3, 6.4.1

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017. 6.1, 6.2, 6.2, 6.3, 6.3.3, 6.4.1, 6.5

J. Shao, L. Tang, M. Liu, G. Shao, L. Sun, and Q. Qiu. BDD-Net: A General Protocol for Mapping Buildings Damaged by a Wide Range of Disasters Based on Satellite Imagery. *Remote Sensing*, 12(10):1670, Jan. 2020. ISSN 2072-4292. doi: 10.3390/rs12101670. 7.1

L. M. Shockley and R. A. Bettinger. Real-time aerospace vehicle position estimation using terrestrial illumination matching. In *2021 IEEE 8th International Workshop on Metrology for AeroSpace (MetroAeroSpace)*, pages 505–509, 2021. doi: 10.1109/MetroAeroSpace51421.2021.9511766. 7.2.3

J. Shrestha, S. Idoko, B. Sharma, and A. K. Singh. End-to-end learning of behavioural inputs for autonomous driving in dense traffic. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10020–10027. IEEE, 2023. 8.1, 8.2

D. Silver, G. Lever, N. M. O. Heess, T. Degris, D. Wierstra, and M. A. Riedmiller. Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, 2014. 6.1, 6.2, 6.3.2

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017. 5.1

D. Simon. *Optimal state estimation: Kalman, H infinity, and nonlinear approaches*. John Wiley & Sons, 2006. 7.1, 7.3.4

M. A. Skinner, M. Coletti, M. C. Voss, T. Svitek, J. C. Lee, K. Auman, H. Patel, and E. J. Moyer. Mitigating CubeSat confusion: Results of in-flight technical demonstrations of candidate tracking and identification technologies. *Journal of Space Safety Engineering*, 9(3):403–409, 2022. ISSN 2468-8967. doi: 10.1016/j.jsse.2022.05.008. 7.2.1, 7.2.2

M. Straub and J. A. Christian. Autonomous optical navigation for earth-observing satellites using coastline matching. In *AIAA Guidance, Navigation, and Control Conference*, page 1334, 2015. 7.2.3

J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, 2012. (document), 4.6, 4.3, 4.6.3

H. J. Suh, M. Simchowitz, K. Zhang, and R. Tedrake. Do differentiable simulators give better policy gradients? In *International Conference on Machine Learning*, pages 20668–20696. PMLR, 2022. 4.2, 5.2, 5.5.2, 8.2, 8.4.2

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018. 5.2

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. 2014. 3.2

P. Tandon. Pytorch implementation of soft actor critic. URL https://github.com/pranz24/pytorch-soft-actor-critic. 5.5.2, 5.5.2

C. Tang and P. Tan. BA-net: Dense bundle adjustment networks. In *ICLR*, 2019. URL https://openreview.net/forum?id=B1gabhRcYX. 4.2, 4.2

G. Tao, S. Ma, Y. Liu, and X. Zhang. Attacks meet interpretability: Attribute-steered detection of adversarial samples. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/b994697479c5716eda77e8e9713e5f0f-Paper.pdf. 3.2

K. Tateno, F. Tombari, I. Laina, and N. Navab. Cnn-slam: Real-time dense monocular slam with learned depth prediction. In *CVPR*, 2017. 4.2

Z. Teed and J. Deng. DROID-SLAM: Deep visual SLAM for monocular, stereo, and RGB-d cameras. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan,

editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=ZBfUo_dr4H. (document), 4.1, 4.1, 4.2, 4.4.3, 4.5, ??, 4.6.3, 4.6.6, 4.6.8, 8.2, 8.5.4

Z. Teed, L. Lipson, and J. Deng. Deep patch visual odometry. *Advances in Neural Information Processing Systems*, 2023. (document), 4.1, 4.1, 4.2, 4.3, 4.4.3, 4.5, 4.3, 4.6.3, 4.6.6, 4.6.8, 8.2

M. Toussaint. A novel augmented lagrangian approach for inequalities and convergent any-time non-central updates. *arXiv preprint arXiv:1412.4329*, 2014. 8.4.2

B. Ummenhofer, H. Zhou, J. Uhrig, N. Mayer, E. Ilg, A. Dosovitskiy, and T. Brox. Demon: Depth and motion network for learning monocular stereo. In *CVPR*, 2017. 4.2

J. van den IJssel, J. Encarnação, E. Doornbos, and P. Visser. Precise science orbits for the Swarm satellite constellation. *Advances in Space Research*, 56(6):1042–1055, 2015. ISSN 0273-1177. doi: 10.1016/j.asr.2015.06.002. 7.2.1

A. Vemula, W. Sun, M. Likhachev, and J. A. Bagnell. On the effectiveness of iterative learning control. In *Learning for Dynamics and Control Conference*, pages 47–58. PMLR, 2022. 5.2, 5.3.3

H. F. Walker and P. Ni. Anderson acceleration for fixed-point iterations. *SIAM Journal on Numerical Analysis*, 49(4):1715–1735, 2011. 2.2.1

W. Wan, Y. Wang, Z. M. Erickson, and D. Held. Difftop: Differentiable trajectory optimization for deep reinforcement and imitation learning. *ArXiv*, abs/2402.05421, 2024. URL https://api.semanticscholar.org/CorpusID:267548058. 8.2

P.-W. Wang, P. Donti, B. Wilder, and Z. Kolter. SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6545–6554. PMLR, 09–15 Jun 2019. URL http://proceedings.mlr.press/v97/wang19e.html. 3.2

S. Wang, R. Clark, H. Wen, and N. Trigoni. Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. In *IEEE Int. Conf. Robotics and Automation*, 2017. doi: 10.1109/ICRA.2017.7989236. URL https://doi.org/10.1109/ICRA.2017.7989236. 4.1, 4.2

W. Wang, D. Zhu, X. Wang, Y. Hu, Y. Qiu, C. Wang, Y. Hu, A. Kapoor, and S. Scherer. Tartanair: A dataset to push the limits of visual slam. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020. (document), 4.1, 4.6, 4.1, 4.6.3

W. Wang, Y. Hu, and S. Scherer. Tartanvo: A generalizable learning-based vo. In

Conference on Robot Learning, 2021. 4.1, 4.2

L. Weaver and N. Tao. The optimal reward baseline for gradient-based reinforcement learning. *ArXiv*, abs/1301.2315, 2001. 6.2

E. Winston and J. Z. Kolter. Monotone operator equilibrium networks. In *Neural Information Processing Systems*, 2020. 2.2.1, 3.2, 3.3.2, 3.3.5, 8.5.4

I. H. Witten. An adaptive optimal controller for discrete-time markov environments. *Inf. Control.*, 34:286–295, 1977. 6.2

E. Wong, L. Rice, and J. Z. Kolter. Fast is better than free: Revisiting adversarial training. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BJx040EFvH. 3.2, 3.4

Y. Wu and K. He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018. 3.4

Y. Wu, J. Donahue, D. Balduzzi, K. Simonyan, and T. Lillicrap. {LOGAN}: Latent optimisation for generative adversarial networks, 2020. URL https://openreview.net/forum?id=rJeU_1SFvr. 3.2

X. Xiao, T. Zhang, K. Choromanski, E. Lee, A. Francis, J. Varley, S. Tu, S. Singh, P. Xu, F. Xia, et al. Learning model predictive controllers with real-time attention for real-world navigation. *arXiv preprint arXiv:2209.10780*, 2022. 8.1, 8.2

Z. Xie, G. Berseth, P. Clary, J. Hurst, and M. van de Panne. Feedback control for cassie with deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1241–1246. IEEE, 2018. 5.1

J. Xu, T. Du, M. Foshey, B. Li, B. Zhu, A. Schulz, and W. Matusik. Learning to fly: computational controller design for hybrid uavs with reinforcement learning. *ACM Transactions on Graphics (TOG)*, 38(4):1–12, 2019. 5.1

J. Xu, V. Makoviychuk, Y. Narang, F. Ramos, W. Matusik, A. Garg, and M. Macklin. Accelerated policy learning with parallel differentiable simulation. In *International Conference on Learning Representations*, 2021. 6.1, 6.2, 6.4.2, 6.5, 6.5

J. Xu, M. Macklin, V. Makoviychuk, Y. Narang, A. Garg, F. Ramos, and W. Matusik. Accelerated policy learning with parallel differentiable simulation. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=ZSKRQMvttc. 5.2, 5.5.1, 5.5.1

N. Yang, L. v. Stumberg, R. Wang, and D. Cremers. D3vo: Deep depth, deep pose and deep uncertainty for monocular visual odometry. In *CVPR*, 2020. 4.2

Y. Yang, X. Yue, and A. G. Dempster. GPS-based onboard real-time orbit determination for leo satellites using consider Kalman filter. *IEEE Transactions on Aerospace and Electronic Systems*, 52(2):769–777, Apr. 2016. ISSN 1557-9603. doi:

10.1109/TAES.2015.140758. Conference Name: IEEE Transactions on Aerospace and Electronic Systems. 7.1

B. Yi, M. A. Lee, A. Kloss, R. Martín-Martín, and J. Bohg. Differentiable factor graph optimization for learning smoothers. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1339–1345. IEEE, 2021. 8.2

J. You, X. Li, M. Low, D. Lobell, and S. Ermon. Deep gaussian process for crop yield prediction based on remote sensing data. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1), Feb. 2017. doi: 10.1609/aaai.v31i1.11172. 7.1

A. Zadeh, Y.-C. Lim, P. P. Liang, and L.-P. Morency. Variational auto-decoder. *arXiv preprint arXiv:1903.00840*, 2019. 3.2, 3.4.1

J. Zhang, B. O'Donoghue, and S. Boyd. Globally convergent type-i anderson acceleration for nonsmooth fixed-point iterations. *SIAM Journal on Optimization*, 30(4):3170–3197, 2020. 3.4.5

L. Zheng, T. Fiez, Z. Alumbaugh, B. Chasnov, and L. J. Ratliff. Stackelberg actor-critic: Game-theoretic reinforcement learning algorithms. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pages 9217–9224, 2022. 8.4.2

H. Zhou, B. Ummenhofer, and T. Brox. Deeptam: Deep tracking and mapping. In *ECCV*, 2018. 4.2

T. Zhou, M. Brown, N. Snavely, and D. G. Lowe. Unsupervised learning of depth and ego-motion from video. In *CVPR*, 2017. 4.1, 4.2

L. Zintgraf, K. Shiarli, V. Kurin, K. Hofmann, and S. Whiteson. Fast context adaptation via meta-learning. In *International Conference on Machine Learning*, pages 7693–7702. PMLR, 2019. 3.2