Perspectives on Policy Learning

Samuel K. Ainsworth

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2022

Reading Committee:
Siddhartha S. Srinivasa, Chair
Kevin Jamieson
Byron Boots

Program Authorized to Offer Degree:
Paul G. Allen School of Computer Science and Engineering

University of Washington

Abstract

Perspectives on Policy Learning

Samuel K. Ainsworth

Chair of the Supervisory Committee:
Professor Siddhartha S. Srinivasa
Paul G. Allen School of Computer Science & Engineering

Sequential decision making, especially in the face of uncertainty, is a central challenge in our quest to build increasingly safe, capable, and (seemingly-)intelligent autonomous systems. Whereas supervised learning is concerned with selecting optimal actions in independent interactions with an environment, policy learning studies action selection in sequential, dependent interactions in an environment. Policy learning constitutes an elemental component of many techniques across reinforcement learning, optimal control, and robotics.

In this dissertation, a variety of perspectives on policy learning are presented, each taking a slightly different lens to the problem. We analyze theoretical and practical speedups to policy learning, and explore subtle yet critical ways in which it differs from supervised learning.

We proceed in three parts,

1. In Chapter 2 we study which interactions with the environment are most beneficial, sparked by the intuition that many environments include "dead end" areas of state space. We extend this framework to additionally study the tradeoffs associated with safety interventions in real-world deployments of policy learning techniques. We analyze the regret behavior of emergency stopping and present empirical results in discrete and continuous settings demonstrating that our reset mechanism can provide order-of-magnitude speedups on top of existing reinforcement learning methods.

2. In Chapter 3 we study the estimation of policy gradients for continuous-time systems with known dynamics. By reframing policy learning in continuous-time, we show that it is possible to construct a more efficient and accurate gradient estimator. With the explicit goal of estimating continuous-time gradients, we are able to discretize adaptively and construct a more efficient policy gradient estimator which we call the Continuous-Time Policy Gradient (CTPG). We show that replacing conventional policy gradients with more efficient CTPG estimates results in faster and more robust learning in a variety of control tasks and simulators.

3. Intrigued by the failures of policy gradient methods in certain settings, we study how policy learning differs from supervised learning in Chapter 4. In particular we explore the unreasonable effectiveness of stochastic gradient descent operating in supervised learning loss landscapes. We hypothesize that this ease is due to such loss landscapes effectively having only a single basin, modulo symmetries in the model parameterization. We propose three novel canonicalization algorithms to reconcile the symmetries between model weights, and justify the single-basin claim with experiments across a variety of model architectures and datasets including the first demonstration – to the best of our knowledge – of perfect linear mode connectivity between two completely independently trained large ResNet models.

TABLE OF CONTENTS

Page

List of Figures iii

Chapter 1: Introduction 1
1.1 Organization 3
1.2 Publications 4

Chapter 2: Policy Learning with Intervention 5
2.1 Introduction 5
2.2 Related Work 6
2.3 Problem setup 8
2.4 Incorporating e-stop interventions 9
2.5 Learning from observation 13
2.6 Empirical study 14
2.7 Types of support sets and their tradeoffs 19
2.8 Discussion 20

Chapter 3: Policy Learning with Differentiable Simulation 22
3.1 Introduction 22
3.2 Learning with Policy Gradients 23
3.3 Numerical Policy Gradient Estimators 26
3.4 Experiments 31
3.5 Related Work 35
3.6 Discussion 36

Chapter 4: Differences between Policy Learning and Supervised Learning 38
4.1 Introduction 38
4.2 Background 40

4.3 Permutation selection methods . . . 42
4.4 Interlude: a counterexample . . . 47
4.5 Experiments . . . 49
4.6 Related work . . . 54
4.7 Discussion . . . 56

Appendix A: Policy Learning with Intervention . . . 80
A.1 Proof of Eq. (2.5) . . . 80
A.2 Proof of Theorem 2.4.1 . . . 80
A.3 Proof of Theorem 2.4.2 . . . 81
A.4 Proof of Corollary 2.4.2.1 . . . 82
A.5 Proof of Theorem 2.5.1 . . . 83
A.6 Imperfect e-stops in terms of $\hat{\rho}_{\pi_e}(s)$ . . . 83
A.7 Experimental details . . . 84

Appendix B: Policy Learning with Differentiable Simulation . . . 88
B.1 Experimental models . . . 88
B.2 Neural ODEs and their linear stability properties . . . 91
B.3 Fusing Operations in the CTPG Algorithm . . . 91

Appendix C: Differences between Policy Learning and Supervised Learning . . . 93
C.1 Auxiliary plots . . . 93
C.2 Failed idea: A method for steepest descent . . . 93
C.3 Proof of Lemma 1 . . . 94

LIST OF FIGURES

Figure Number		Page
2.1	An illustration of e-stop interventions	7
2.2	E-stop results on discrete state/action environments	15
2.3	E-stop performance as a function of the quality of the expert policy	17
2.4	E-stop results on continuous MuJoCo benchmark environments	19
3.1	Comparing policy gradient estimators	25
3.2	The policy gradient compute/accuracy trade-off	28
3.3	The instability of Neural ODE policy gradients	30
3.4	Training a differential drive control policy	32
3.5	CTPG with gradients from an external differentiable physics simulator	33
3.6	CTPG with finite difference gradients from an external non-differentiable physics simulator	34
3.7	CTPG on a quadrotor control problem	34
4.1	A counterexample to the linear mode connectivity/permutation conjecture	48
4.2	Linear mode connectivity is possible after permuting hidden units	49
4.3	Linear mode connectivity does not work at initialization	51
4.4	Wider models exhibit better linear mode connectivity	53
4.5	Models trained on disjoint datasets can be merged with positive-sum results	53
A.1	Illustration of the OpenAI Gym FrozenLake-v0 environment	85
C.1	Linear mode connectivity counterexample problem data	93

ACKNOWLEDGMENTS

Thank you to my many friends and accomplices who have been a source of support and joy throughout this wretched experiment. In no particular order, thank you to Ofir Press, Jennifer Brennan, John Thickstun, Krishna Pillutla, Rahul "Raulito" Kidambi, Jonathan Hayase, Ian Covert, Darcy "Motions and Lemonade" Covert, Melanie Sclar, Manaswi Saha, Ivan Evtimov, Vivek Ramanujan, Sarah Pratt, Mitchell Wortsman, Matthew "Fremont" Wallingford, Aditya Kusupati, Mohammadreza "GQ" Salehi, Gabriel Ilharco Magalhães, Matthew Barnes, Jesse "Dodgecoin" Dodge, Amanda "Swolemanda" Baughan, Edward Misback, Brian Hou, Nicasia Beebe-Wang, Vinayak "Vinny Magalhães" Goyal, Crystal Acevedo, Shreyas Jaganmohan, Frank Goodman, Lancelot Wathieu, Andrew Mullen, Gian Marco Visani, Artidoro Pagnoni, Jared Roesch, Eunice Jun, Marisa Kirisame, Nelson Liu, Inna Lin, Tim Dettmers, and certainly many others. Getting to know each of you has been the greatest outcome from my time here.

Thank you to Scott Shiebler and members of the Men's Group for keeping me mentally and emotionally afloat for the past few years. I would not be the same person today without your support.

Thank you to Elise Dorough for getting me out of more quagmires than I would like to admit. Your care for and support of the graduate student population keeps the entire department afloat.

Thank you to Erik Sudderth for introducing me to the world of machine learning research and taking a chance on a much younger, more naïve version of myself. Your patience working with junior researchers is something I hope to emulate one day.

Thank you to Stuart Geman for instilling in me a love of probability, graphical models,

and applied mathematics. I maintain fond memories of my time in your office hours hearing stories of (others') election gambling, tire slashing, and stock trading escapades.

Thank you to Mr. Jason Hughes for teaching me so much more than Tae Kwon Do. The many lessons learned at Live Oak Martial Arts carry through to this day. Your impact on my life cannot be overstated.

Finally, thank you to my parents – Scott and Susan – for bringing me into existence and to my brother Benjamin for his fellow, continued existence.

DEDICATION

To the victims of traffic violence.

Forty-two thousand nine hundred fifteen lives lost in the United States in the year 2021 alone.

That's 9/11, every single month.

Chapter 1

INTRODUCTION

This thesis explores the science of sequential decision making, and in particular policy learning. An intuition for the problem may be gleaned by analogy to video games. Imagine playing a new video game for the first time. Upon starting, you are presented with information: your player health, items at your disposal, current power-ups, and so on. Other forms of *state* may be contextual, eg. you are presented with your location and orientation within some three-dimensional world. Video games prey on our human social, contextual, and experiential expectations of object affordances and physics. In other words, you expect boxes to contain other items, doors to open, and so forth. A computer attempting to "learn" to play the same game is not privileged with the same prior knowledge as you or I. But we will eschew the issue of perception in this thesis and set human and machine on equal footing by assuming access to some machine-usable representation of state.

Next, you hold a controller with buttons, sticks, and other gizmos allowing you to take *actions* to affect your player state. It is obvious to you which actions are at your disposal – there are only so many buttons on the controller. However what effect these actions may have is unclear. Depending on which state you happen to be in, pressing a certain button may move your player forward, may pick up an item, or may simply do nothing at all. Furthermore, the outcome of taking an action may itself be random.

Finally, you are presented with a score. Your goal is to maximize this number. By acting in the game environment you progress your player state and accumulate *reward*, changing your score. Rewards may be positive, zero, or negative, depending on the circumstances of your current state and chosen action. Because of the sequential nature of the game you may prefer to forgo immediate rewards in the interest of future rewards, or vice versa.

In your quest to play this game you must approach two tasks simultaneously: (1) learn the

dynamics of the game, ie. what happens when you take each action in each state, and (2) select a *policy* of play in the game, ie. what action you select in each state. This latter task is known as *policy learning* and is the primary problem of interest in this thesis. When the game dynamics are unknown and must be learned in concert with a policy, we term the problem *reinforcement learning*. On the other hand, if the dynamics are fully known ahead of time – say in a game with known rules like solitaire, or an open source video game – we term the problem *planning*.

Formally, we encode this setup in the language of Markov decision processes (MDPs).

Definition 1.0.1. A *finite-horizon Markov decision process (MDP)* is a tuple, $\langle \mathcal{S}, \mathcal{A}, P, R, H, \rho_0 \rangle$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition dynamics and $\Delta(\cdot)$ is a probability distribution over some space, $R : \mathcal{S}^2 \times \mathcal{A} \to [0, 1]$ is the reward function, $\rho_0 \in \Delta(\mathcal{S})$ is the distribution of the initial state s_0, and we "play the game" H steps.

Hopefully the connection to our video game analogy is now clear: MDPs offer an abstraction to formally study the task of sequential decision making. Questions like "Should I forego immediate reward in the expectation of future reward in this state?" and "Should I prefer action A or B in this state?" now have concrete mathematical translations. Within this abstraction, we formalize *policies* as functions – possibly stochastic – from states, $\mathcal{S}$, to actions, $\mathcal{A}$.

Definition 1.0.2. A *policy* is a function, $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, mapping states to distributions over actions.

In some cases, we may rely on deterministic policies, those where states are mapped to a single action with probability one, such as when operating in continuous-time in Chapter 3. In other cases, freeing ourselves to express some "uncertainty" in actions yields increased flexibility, and generally results in easier optimization.

We measure the fitness of a policy by its expected future rewards throughout a finite episode,

$$J(\pi) = \mathbb{E}\left[\sum_{t=0}^{H-1} R(s_t, a_t, s_{t+1}) \mid \pi, s_0\right]$$

In other words, we measure the expected reward accrued based on all possible trajectories and their probabilities under the policy, π, and transition dynamics, $P(s, a)$. That said, extensions to the time-discounted, infinite-horizon setting are also possible for all the results presented herein.

This thesis studies ways in which policy learning can be done more efficiently by leveraging additional information, or unique problem structure. Finally, we discuss the hardness of policy learning in relation to supervised deep learning.

1.1 Organization

In Chapter 2, we investigate policy learning in the presence of supervisor feedback. In the video game analogy, suppose you play with your friend who has sunk an ungodly amount of time into the game to reach expert performance. Your friend could periodically prevent you from executing behaviors that are either too dangerous or reward-infertile, based on their expertise. Although annoying, we show in Chapter 2 that this form of expert feedback can be extremely fruitful, yielding significantly faster policy learning. Furthermore, we analyze and bound the regret incurred by this supervision. In practical terms, Chapter 2 characterizes the trade-off between imitation and exploration, with applications to robotics, reinforcement learning, and autonomous system safety.

In Chapter 3, we switch to the continuous regime and demonstrate that, in compatible environments, policy learning in continuous time can be done an order of magnitude faster than prior methods. Leveraging differentiable physics simulators, we are able to leverage methods from the numerical integrator literature for vastly improved performance.

A recurring motif in this thesis will be that policy learning is hard. It is harder than supervised learning. It is harder than trajectory optimization. We explore what makes policy learning so challenging in Chapter 4 via comparison to supervised deep learning. In particular, we argue that supervised deep learning is practically straightforward thanks to its loss landscapes containing only a single basin. We demonstrate this phenomenon via the development of three novel algorithms for model averaging that account for permutation symmetries among the model weights. Using these model averaging algorithms we are able to demonstrate that in many cases independently

trained models all exist in the same basin, modulo permutation symmetries.

1.2 Publications

For the committee's convenience the publications contributing to this thesis are as follows:

- Samuel K. Ainsworth, Nicholas J. Foti, Adrian K. C. Lee, and Emily B. Fox. oi-vae: Output interpretable vaes for nonlinear group factor analysis. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 119–128. PMLR, 2018b. URL `http://proceedings.mlr.press/v80/ainsworth18a.html`

- Samuel K. Ainsworth, Nicholas J. Foti, and Emily B. Fox. Disentangled VAE Representations for Multi-Aspect and Missing Data. *CoRR*, abs/1806.09060, 2018a

- Laurel J. Orr, Samuel K. Ainsworth, Kevin G. Jamieson, Walter Cai, Magdalena Balazinska, and Dan Suciu. Mosaic: A Sample-Based Database System for Open World Query Processing. In *10th Conference on Innovative Data Systems Research, CIDR*, 2020

- Samuel K. Ainsworth, Matt Barnes, and Siddhartha S. Srinivasa. Mo' States Mo' Problems: Emergency Stop Mechanisms from Observation. In *Conference on Neural Information Processing Systems, NeurIPS*, 2019

- Samuel K. Ainsworth, Kendall Lowrey, John Thickstun, Zaïd Harchaoui, and Siddhartha S. Srinivasa. Faster policy learning with continuous-time gradients. In Ali Jadbabaie, John Lygeros, George J. Pappas, Pablo A. Parrilo, Benjamin Recht, Claire J. Tomlin, and Melanie N. Zeilinger, editors, *Proceedings of the 3rd Annual Conference on Learning for Dynamics and Control, L4DC 2021, 7-8 June 2021, Virtual Event, Switzerland*, volume 144 of *Proceedings of Machine Learning Research*, pages 1054–1067. PMLR, 2021. URL `http://proceedings.mlr.press/v144/ainsworth21a.html`

Chapter 2

POLICY LEARNING WITH INTERVENTION

2.1 Introduction

We consider the problem of determining when along a training roll-out feedback from the environment is no longer beneficial, and an intervention such as resetting the agent to the initial state distribution is warranted. We show that such interventions can naturally trade off a small sub-optimality gap for a dramatic decrease in sample complexity. In particular, we focus on the reinforcement learning setting in which the agent has access to a reward signal in addition to either (a) an expert supervisor triggering the e-stop mechanism in real-time or (b) expert state-only demonstrations used to "learn" an automatic e-stop trigger. Both settings fall into the same framework.

Evidence already suggests that using simple, manually-designed heuristic resets can dramatically improve training time. For example, the classic pole-balancing problem originally introduced in Widrow and Smith [1964] prematurely terminates an episode and resets to an initial distribution whenever the pole exceeds some fixed angle off-vertical. More subtly, these manually designed reset rules are hard-coded into many popular OpenAI gym environments Brockman et al. [2016].

Some recent approaches have demonstrated empirical success learning when to intervene, either in the form of resetting, collecting expert feedback, or falling back to a safe policy Eysenbach et al. [2018], Laskey et al. [2016], Richter and Roy [2017], Kahn et al. [2017]. We specifically study reset mechanisms which are more natural for human operators to provide – in the form of large red buttons, for example – and thus perhaps less noisy than action or value feedback Bagnell [2015]. Further, we show how to build automatic reset mechanisms from state-only observations which are often widely available, e.g. in the form of videos Torabi et al. [2018].

The key idea of our method is to build a *support set* related to the expert's state-visitation

probabilities, and to terminate the episode with a large penalty when the agent leaves this set, visualized in Fig. 2.1. This support set defines a modified MDP and can either be constructed implicitly via an expert supervisor triggering e-stops in real-time or constructed a priori based on observation-only roll-outs from an expert policy. As we will show, using a support set explicitly restricts exploration to a smaller state space while maintaining guarantees on the learner's performance. We emphasize that our technique for incorporating observations applies to *any* reinforcement learning algorithm in either continuous or discrete domains.

The contributions and organization of the remainder of the chapter is as follows.

- We provide a general framework for incorporating arbitrary emergency stop (e-stop) interventions from a supervisor into any reinforcement learning algorithm using the notion of support sets in Section 2.4.

- We present methods and analysis for building support sets from observations in Section 2.5, allowing for the creation of automatic e-stop devices.

- In Section 2.6 we empirically demonstrate on benchmark discrete and continuous domains that our reset mechanism allows us to naturally trade off a small asymptotic sub-optimality gap for significantly improved convergence rates with any reinforcement learning method.

- Finally, in Section 2.7, we generalize the concept of support sets to a spectrum of set types and discuss their respective tradeoffs.

2.2 Related Work

The problem of learning when to intervene has been studied in several contexts and generally falls under the framework of safe reinforcement learning Garcıa and Fernández [2015] or reducing expert feedback Laskey et al. [2016]. Richter and Roy [2017] use an auto-encoder as an anomaly detector to determine when a high dimensional state is anomalous, and revert to a safe policy. Laskey et al. [2016] use a one-class SVM as an anomaly detector, but instead for the purposes of

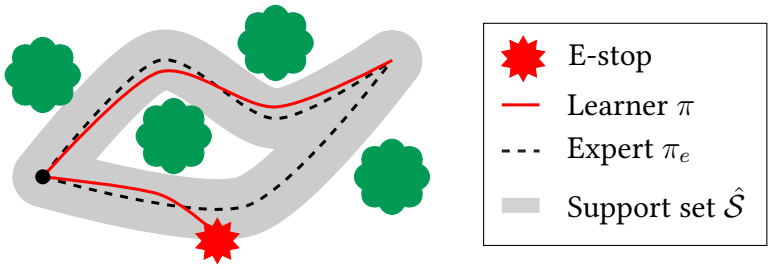

Figure 2.1: A robot is tasked with reaching a goal in a cluttered environment. Our method allows incorporating e-stop interventions into any reinforcement learning algorithm. The grey support set may either be implicit (from a supervisor) or, if available, explicitly constructed from demonstrations.

reducing the amount of imitation learning feedback during DAgger training Ross et al. [2011]. Garcia and Fernández [2012] perturb a baseline policy and request action feedback if the current state exceeds a minimum distance from any demonstration. Geramifard et al. [2011] assume access to a function which indicates whether a state is safe, and determines the risk of the current state by Monte Carlo roll-outs. Similarly, "shielding" Alshiekh et al. [2018] uses a manually specified safety constraint and a coarse, conservative abstraction of the dynamics to prevent an agent from violating the safety constraint. Eysenbach et al. [2018] learn a second "soft reset" policy (in addition to the standard "hard" reset) which prevents the agent from entering nearly non-reversible states and returns the agent to an initial state. Hard resets are required whenever the soft reset policy fails to terminate in a manually defined set of safe states $\mathcal{S}_{\text{reset}}$. Our method can be seen as learning $\mathcal{S}_{\text{reset}}$ from observation. Their method trades off hard resets for soft resets, whereas ours learns when to perform the hard resets.

The general problem of Learning from Demonstration (LfD) has been studied in a variety of contexts. In inverse reinforcement learning, Abbeel and Ng [2004] assume access to state-only trajectory demonstrations and attempt to learn an unknown reward function. In imitation learning, Ross et al. [2011] study the distribution mismatch problem of behavior cloning and propose DAgger, which collects action feedback at states visited by the current policy. Gail

addresses the problem of imitating a set of fixed trajectories by minimizing the Jensen-Shannon divergence between the policies' average-state-action distributions Ho and Ermon [2016]. This reduces to optimizing a GAN-style minimax objective with a reinforcement learning update (the generator) and a divergence estimator (the discriminator).

The setting most similar to ours is Reinforcement Learning with Expert Demonstrations (RLED), where we observe both the expert's states and actions in addition to a reward function. Abbeel and Ng [2006] use state-action trajectory demonstrations to initialize a model-based RL algorithm, which eliminates the need for explicit exploration and can avoid visiting all of the state-action space. Smart and Kaelbling [2000] bootstrap Q-values from expert state-action demonstrations. Maire and Bulitko [2005] initialize any value-function based RL algorithm by using the shortest observed path from each state to the goal and generalize these results to unvisited states via a graph Laplacian. Nagabandi et al. [2018] learn a model from state-action demonstrations and use model-predictive control to initialize a model-free RL agent via behavior cloning. It is possible to extend our method to RLED by constructing a support superset based on state-action pairs, as described in Section 2.7. Thus, our method and many RLED methods are complimentary. For example, DQfD Hester et al. [2018] would allow pre-training the policy from the state-action demonstrations, whereas ours reduces exploration during the on-policy learning phase.

Most existing techniques for bootstrapping RL are not applicable to our setting because they require either (a) state-action observations, (b) online expert feedback, (c) solving a reinforcement learning problem in the original state space, incurring the same complexity as simply solving the original RL problem, or (d) provide no guarantees, even for the tabular setting. Further, since our method is equivalent to a one-time modification of the underlying MDP, it can be used to improve any existing reinforcement learning algorithm and may be combined with other bootstrapping methods.

2.3 Problem setup

Let $M = \langle \mathcal{S}, \mathcal{A}, P, R, H, \rho_0 \rangle$ be a finite horizon, episodic Markov decision process (MDP) defined by the tuple M, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition

probability distribution and $\Delta(\cdot)$ is a probability distribution over some space, $R : \mathcal{S}^2 \times \mathcal{A} \to [0, 1]$ is the reward function, H is the time horizon and $\rho_0 \in \Delta(\mathcal{S})$ is the distribution of the initial state s_0. Let $\pi \in \Pi : \mathbb{N}_{1:H} \times \mathcal{S} \to \Delta(\mathcal{A})$ be our learner's policy and π_e be the potentially sub-optimal expert policy (for now assume the realizability setting, $\pi_e \in \Pi$).

The state distribution of policy π at time t is defined recursively as

$$\rho_\pi^{t+1}(s) = \sum_{s_t, a_t} \rho_\pi^t(s_t)\pi(a_t|s_t, t)P(s_t, a_t, s) \qquad \rho_\pi^0(s) \equiv \rho_0(s). \tag{2.1}$$

The expected sum of rewards over a single episode is defined as

$$J(\pi) = \mathbb{E}_{s\sim\rho_\pi, a\sim\pi(s), s'\sim P(s,a)} R(s, a, s') \tag{2.2}$$

where ρ_π denotes the average state distribution, $\rho_\pi = \frac{1}{H}\sum_{t=0}^{H-1} \rho_\pi^t$.

Our objective is to learn a policy π which minimizes the notion of external regret over K episodes. Let $T = KH$ denote the total number of time steps elapsed, $(r_1, \ldots, r_T)$ be the sequence of rewards generated by running algorithm $\mathfrak{A}$ in M and $R_T = \sum_{t=1}^{T} r_t$ be the cumulative reward. Then the T-step expected regret of $\mathfrak{A}$ in M compared to the expert is defined as

$$\text{Regret}_M^{\mathfrak{A}}(T) := \mathbb{E}_M^{\pi_e}[R_T] - \mathbb{E}_M^{\mathfrak{A}}[R_T]. \tag{2.3}$$

Typical regret bounds in the discrete setting are some polynomial of the relevant quantities $|\mathcal{S}|$, $|\mathcal{A}|$, T, and H. We assume we are given such an RL algorithm. Later, we assume access to either a supervisor who can provide e-stop feedback or a set of demonstration roll-outs $D = \left\{\tau^{(1)}, \ldots, \tau^{(n)}\right\}$ of an expert policy π_e in M, and show how these can affect the regret bounds. In particular, we are interested in using D to decrease the effective size of the state space $\mathcal{S}$, thereby reducing the amount of required exploration when learning in M.

2.4 Incorporating e-stop interventions

In the simplest setting, we have access to an external supervisor who provides minimal online feedback in the form of an e-stop device triggered whenever the agent visits states outside of some to-be-determined set $\hat{\mathcal{S}} \subseteq \mathcal{S}$. For example, if a mobile robot is navigating across a cluttered

room as in Fig. 2.1, a reasonable policy will rarely collide into objects or navigate into other rooms which are unrelated to the current task, and the supervisor may trigger the e-stop device if the robot exhibits either of those behaviors. The analysis for other support types (e.g. state-action, time-dependent, visitation count) are similar, and their trade-offs are discussed in Section 2.7.

2.4.1 The sample complexity and asymptotic sub-optimality trade-off

We argue that for many practical MDPs, ρ_{π_e} is near-zero in much of the state space, and constructing an appropriate $\hat{\mathcal{S}}$ enables efficiently trading off asymptotic sub-optimality for potentially significantly improved convergence rate. Given some reinforcement learning algorithm $\mathfrak{A}$, we proceed by running $\mathfrak{A}$ on a newly constructed "e-stop" MDP $\widehat{M} = (\hat{\mathcal{S}}, \mathcal{A}, P_{\hat{\mathcal{S}}}, R_{\hat{\mathcal{S}}}, H, \rho_0)$. Intuitively, whenever the current policy leaves $\hat{\mathcal{S}}$, the e-stop prematurely terminates the current episode with no further reward (the maximum penalty). These new transition and reward functions are defined as

$$P_{\hat{\mathcal{S}}}(s_t, a_t, s_{t+1}) = \begin{cases} P(s_t, a_t, s_{t+1}), & \text{if } s' \in \hat{\mathcal{S}} \\ \sum_{s' \notin \hat{\mathcal{S}}} P(s_t, a_t, s'), & s_{t+1} = s_{\text{term}} \\ 0, & \text{else} \end{cases} \tag{2.4}$$

$$R_{\hat{\mathcal{S}}}(s_t, a_t, s_{t+1}) = \begin{cases} R(s_t, a_t, s_{t+1}), & \text{if } s_{t+1} \in \hat{\mathcal{S}} \\ 0, & \text{else} \end{cases}$$

where s_{term} is an absorbing state with no reward. A similar idea was discussed for the imitation learning problem in Ross et al. [2013].

The key trade-off we attempt to balance is between the *asymptotic sub-optimality* and *reinforcement learning regret*

$$\text{Regret}(T) \leq \underbrace{\lceil * \rceil \tfrac{T}{H} \left[J(\pi_e) - J(\hat{\pi}^*) \right]}_{\text{Asymptotic sub-optimality}} + \underbrace{\mathbb{E}^{\hat{\pi}^*}_{\widehat{M}} \left[R_T \right] - \mathbb{E}^{\mathfrak{A}}_{\widehat{M}} \left[R_T \right]}_{\text{Learning regret}} \tag{2.5}$$

where π_e and $\hat{\pi}^*$ are the expert policy and optimal policies in $\widehat{M}$, respectively (proof in Appendix A.1). The first term is due to the approximation error introduced when constructing $\widehat{M}$,

and depends on our choice of $\hat{\mathcal{S}}$. The second term is the familiar reinforcement learning regret, e.g. Azar et al. [2017] recently proved an upper regret bound of $\sqrt{H|\mathcal{S}||\mathcal{A}|T} + H^2|\mathcal{S}|^2|\mathcal{A}|$. We refer the reader to Kakade et al. [2018] for an overview of state-of-the-art regret bounds in episodic MDPs.

Our focus is primarily on the first term, which in turn decreases the learning regret of the second term via $|\hat{\mathcal{S}}|$ (typically quadratically). This forms the basis for the key performance trade-off. We introduce bounds for the first term in various conditions, which inform our proposed methods. By intelligently modifying M through e-stop interventions, we can decrease the required exploration and allow for the early termination of uninformative, low-reward trajectories. Note that the reinforcement learning complexity of $\mathfrak{A}$ is now independent of $\mathcal{S}$, and instead dependent on $\hat{\mathcal{S}}$ according to the same polynomial factors. Depending on the MDP and expert policy, this set may be significantly smaller than the full set. In return, we pay a worst case asymptotic sub-optimality penalty.

2.4.2 Perfect e-stops

To begin, consider an idealized setting, $\hat{\mathcal{S}} = \{s|h(s) > 0\}$ (or some superset thereof) where $h(s)$ is the probability π_e visits state s at any point during an episode. Then the modified MDP $\widehat{M}$ has an optimal policy which achieves at least the same reward as π_e on the true MDP M.

Theorem 2.4.1. *Suppose $\widehat{M}$ is an e-stop variant of M such that $\hat{\mathcal{S}} = \{s|h(s) > 0\}$ where $h(s)$ denotes the probability of hitting state s in a roll-out of π_e. Let $\hat{\pi}^* = \arg\max_{\pi \in \Pi} J_{\widehat{M}}(\pi)$ be the optimal policy in $\widehat{M}$. Then $J(\hat{\pi}^*) \geq J(\pi_e)$.*

In other words, and not surprisingly, if the expert policy never visits a state, then we pay no penalty for removing it. (Note that we could have equivalently selected $\hat{\mathcal{S}} = \{s|\rho_{\pi_e}(s) > 0\}$ since $\rho_{\pi_e}(s) > 0$ if and only $h(s) > 0$.)

Algorithm 1 Resetting based on demonstrator trajectories

procedure LEARNEDESTOP($M, \mathfrak{A}, \pi_e, n, \xi$)
  Rollout multiple trajectories from π_e: $D \leftarrow [s_1^{(1)}, \ldots, s_H^{(1)}], \ldots, [s_1^{(n)}, \ldots, s_H^{(n)}]$
  Estimate the hitting probabilities: $\hat{h}(s) = \frac{1}{n} \sum_i \mathbb{I}\{s \in \tau^{(i)}\}$. (Or $\hat{\rho}$ in continuous settings.)
  Construct the smallest $\hat{\mathcal{S}}$ allowed by the $\sum_{s \in \mathcal{S} \setminus \hat{\mathcal{S}}} \hat{h}(s) \leq \xi$ constraint. (Or $\hat{\rho}(\mathcal{S} \setminus \hat{\mathcal{S}}) \leq \xi$.)
  Add e-stops, resulting in a modified MDP, $\widehat{M}$, where $P_{\hat{\mathcal{S}}}(s, a, s'), R_{\hat{\mathcal{S}}}(s, a, s') \leftarrow Eq.$ (2.4)
return $\mathfrak{A}(\widehat{M})$

2.4.3 Imperfect e-stops

In a more realistic setting, consider what happens when we "remove" (i.e. $s \notin \hat{\mathcal{S}}$) states as e-stops that have low but non-zero probability of visitation under π_e. This can happen by "accident" if the supervisor interventions are noisy or we incorrectly estimate the visitation probability to be zero. Alternatively, this can be done intentionally to trade off asymptotic performance for better sample complexity, in which case we remove states with known low but non-zero visitation probability.

Theorem 2.4.2. *Consider $\widehat{M}$, an e-stop variation on MDP M with state spaces $\hat{\mathcal{S}}$ and $\mathcal{S}$, respectively. Given an expert policy, π_e, let $h(s)$ denote the probability of visiting state s at least once in an episode roll-out of policy π_e in M. Then*

$$J(\pi_e) - J(\hat{\pi}^*) \leq H \sum_{s \in \mathcal{S} \setminus \hat{\mathcal{S}}} h(s) \tag{2.6}$$

where $\hat{\pi}^*$ is the optimal policy in $\widehat{M}$. Naturally if we satisfy some "allowance," ξ, such that $\sum_{s \in \mathcal{S} \setminus \hat{\mathcal{S}}} h(s) \leq \xi$ then $J(\pi_e) - J(\hat{\pi}^*) \leq \xi H$.

Corollary 2.4.2.1. *Recall that $\rho_{\pi_e}(s)$ denotes the average state distribution following actions from π_e, $\rho_{\pi_e}(s) = \frac{1}{H} \sum_{t=0}^{H-1} \rho_{\pi_e}^t(s)$. Then*

$$J(\pi_e) - J(\hat{\pi}^*) \leq \rho_{\pi_e}(\mathcal{S} \setminus \hat{\mathcal{S}}) H^2 \tag{2.7}$$

In other words, removing states with non-zero hitting probability introduces error into the policy $\hat{\pi}^*$ according to the visitation probabilities h.

Remark. The primary slack in these bounds is due to upper bounding the expected cumulative reward for a given state trajectory by H. Although this bound is necessary in the worst case, it's worth noting that performance is much stronger in practice. In non-adverserial settings the expected cumulative reward of a state sequence, τ, is correlated with the visitation probabilities of the states along its path: very low reward trajectories tend to have low visitation probabilities, assuming sensible expert policies. We opted against making any assumptions about the correlation between $h(s)$ and the value function, $V(s)$, so this remains an interesting option for future work.

2.5 Learning from observation

In the previous section, we considered how to incorporate general e-stop interventions – which could take the form of an expert supervisor or some other learned e-stop device. Here, we propose and analyze a method for building such a learned e-stop trigger using state observations from an expert demonstrator. This is especially relevant for domains where action observations are unavailable (e.g. videos).

Consider the setting where we observe n roll-outs $\tau^{(1)}, \ldots, \tau^{(n)}$ of a demonstrator policy π_e in M. We can estimate the hitting probability $h(s)$ empirically as $\hat{h}(s) = \frac{1}{n} \sum_i \mathbb{I}\{s \in \tau^{(i)}\}$. Next, Theorem 2.4.2 suggests constructing $\hat{\mathcal{S}}$ by removing states from $\mathcal{S}$ with the lowest $\hat{h}(s)$ values as long as is allowed by the $\sum_{s \in \mathcal{S} \setminus \hat{\mathcal{S}}} \hat{h}(s) \leq \xi$ constraint. In other words, we should attempt to remove as many states as possible while considering our "budget" ξ. The algorithm is summarized in Algorithm 1. In practice, implementing Algorithm 1 is actually even simpler: pick $\hat{\mathcal{S}}$, take any off-the-shelf implementation and simply end training roll-outs whenever the state leaves $\hat{\mathcal{S}}$.

Theorem 2.5.1. *The e-stop MDP $\widehat{M}$ with states $\hat{\mathcal{S}}$ in Algorithm 1 has asymptotic sub-optimality*

$$J(\pi_e) - J(\hat{\pi}^*) \leq (\xi + \epsilon)H \tag{2.8}$$

with probability at least $1 - |\mathcal{S}|e^{-2\epsilon^2 n/|\mathcal{S}|^2}$, for any $\epsilon > 0$. Here ξ denotes our approximate state removal "allowance", where we satisfy $\sum_{s \in \mathcal{S} \setminus \hat{\mathcal{S}}} \hat{h}(s) \leq \xi$ in our construction of $\widehat{M}$ as in Theorem 2.4.2.

As expected, there exists a tradeoff between the number of trajectories collected, n, the state removal allowance, ξ, and the asymptotic sub-optimality gap, $J(\pi_e) - J(\hat{\pi}^*)$. In practice we find performance to be fairly robust to n, as well as the quality of the expert policy. See Section 2.6.1 for experimental results measuring the impact of each of these variables.

Note that although this analysis only applies to the discrete setting, the same method can be extended to the continuous case by estimating and thresholding on $\rho_{\pi_e}(s)$ in place of $h(s)$, as implied by Corollary 2.4.2.1. In Section 2.6.2 we provide empirical results in continuous domains. We also present a bound similar to Theorem 2.5.1 based on $\hat{\rho}_{\pi_e}(s)$ instead of $\hat{h}(s)$ in the appendix (Theorem A.6.1), although unfortunately it retains a dependence on $|\mathcal{S}|$.

2.6 Empirical study

2.6.1 Discrete environments

We evaluate LearnedEStop on a modified FrozenLake-v0 environment from the OpenAI gym. This environment is highly stochastic: for example, taking a left action can move the character either up, left, or down each with probability $1/3$. To illustrate our ability to evade states that are low-value but non-terminating, we additionally allow the agent to "escape" the holes in the map and follow the usual dynamics with probability 0.01. As in the original problem, the goal state is terminal and the agent receives a reward of 1 upon reaching the goal and 0 elsewhere. To encourage the agent to reach the goal quickly, we use a discount factor of $\gamma = 0.99$.

Across all of our experiments, we observe that algorithms modified with our e-stop mechanism are far more sample efficient thanks to our ability to abandon episodes that do not match the behavior of the expert. We witnessed these benefits across both planning and reinforcement learning algorithms, and with both tabular and policy gradient-based techniques.

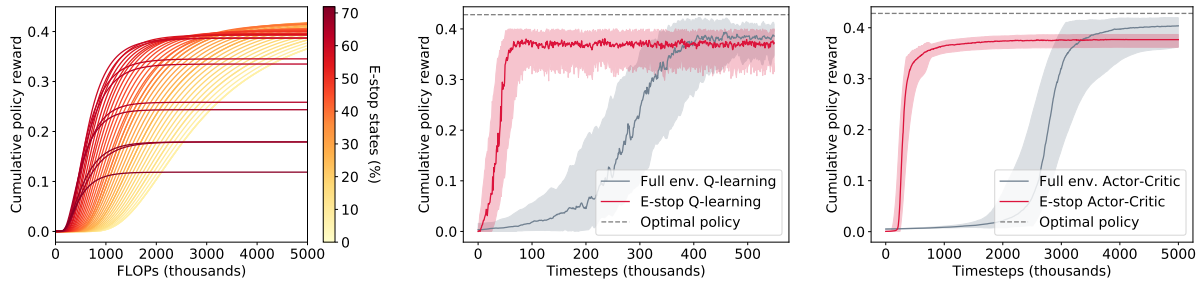

Figure 2.2: *Left:* Value iteration results with varying portions of the state space replaced with e-stops. Color denotes the portion of states that have been replaced. Note that significant performance improvements may be realized before the optimal policy reward is meaningfully affected. *Middle:* Q-learning results with and without the e-stop mechanism. *Right:* Actor-critic results with and without the e-stop mechanism. Both plots show results across 100 trials. We observe that e-stopping produces drastic improvements in sample efficiency while introducing only a small sub-optimality gap.

Although replacing states with e-stops introduces a small sub-optimality gap, practical users need not despair: any policy trained in a constrained e-stop environment is portable to the full environment. Therefore using e-stops to warm-start learning on the full environment may provide a "best of both worlds" scenario. Annealing this process could also have a comparable effect.

Value iteration. To elucidate the relationship between the size of the support set, $\hat{S}$, and the sub-optimality gap, $J(\pi_e) - J(\hat{\pi}^*)$, we run value iteration on e-stop environments with progressively more e-stop states. First, the optimal policy with respect to the full environment is computed and treated as the expert policy, π_e. Next, we calculate $\rho_{\pi_e}(s)$ for all states. By progressively thresholding on $\rho_{\pi_e}(s)$ we produce sparser and sparser e-stop variants of the original environment. The results of value iteration run on each of these variants is shown in Fig. 2.2 (left). Lines are colored according to the portion of states removed from the original environment, darker lines indicating more aggressive pruning. As expected we observe a tradeoff: decreasing the size of $\hat{S}$ introduces sub-optimality but speeds up convergence. Once pruning becomes too

aggressive we see that it begins to remove states crucial to reaching the goal and $J(\pi_e) - J(\hat{\pi}^*)$ is more severely impacted as a result.

RL results. To evaluate the potential of e-stops for accelerating reinforcement learning methods we ran LearnedEStop from Algorithm 1 with the optimal policy as π_e. Half of the states with the lowest hitting probabilities were replaced with e-stops. Finally, we ran classic RL algorithms on the resulting e-stop MDP. Fig. 2.2 (middle) presents our Q-learning results, demonstrating removing half of the states has a minor effect on asymptotic performance but dramatically improves the convergence rate. We also found the e-stop technique to be an effective means of accelerating policy gradient methods. Fig. 2.2 (right) presents results using one-step actor-critic with a tabular, value function critic Sutton and Barto [2018]. In both cases, we witnessed drastic speedups with the use of e-stops relative to running on the full environment.

Expert sub-optimality. The bounds presented Section 2.4.3 are all in terms of $J(\pi_e) - J(\hat{\pi}^*)$, prompting the question: To what extent is e-stop performance dependent on the quality of the expert, $J(\pi_e)$? Is it possible to exceed the performance of π_e as in Theorem 2.4.1, even with "imperfect" e-stops? To address these questions we artificially created sub-optimal policies by adding noise to the optimal policy's Q-function. Next, we used these sub-optimal policies to construct e-stop MDPs, and calculated $J(\hat{\pi}^*)$. As shown in Fig. 2.3 (left), e-stop performance is quite robust to the expert quality. Ultimately we only need to take care of capturing a "good enough" set of states in order for the e-stop policy to succeed.

Estimation error. The sole source of error in Algorithm 1 comes from the estimation of hitting probabilities via a finite set of n expert roll-outs. Theorem 2.5.1 suggests that the probability of failure in empirically building an e-stop MDP decays exponentially in terms of the number of roll-outs, n. We test the relationship between n and $J(\hat{\pi}^*)$ experimentally in Fig. 2.3 (right) and find that in this particular case it's possible to construct very good e-stop MDPs with as few as 10 expert roll-outs.

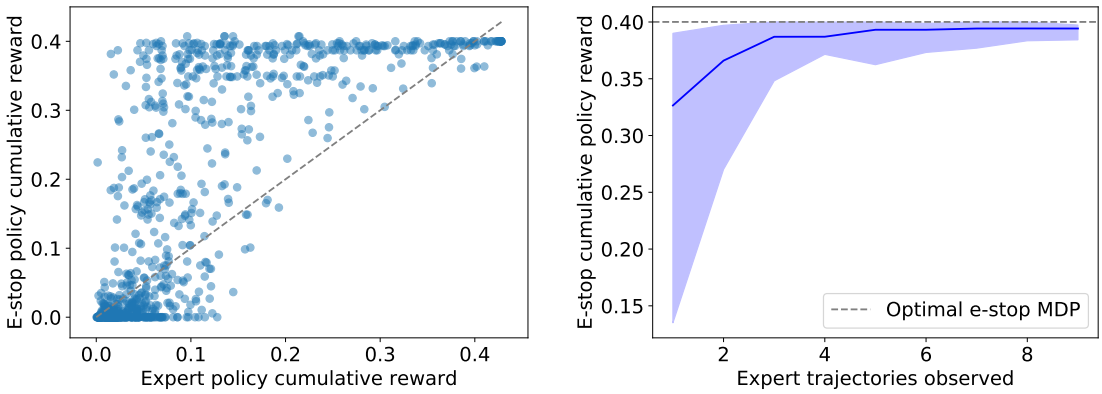

Figure 2.3: *Left:* E-stop results based on sub-optimal expert policies. *Right:* The number of expert trajectories used to construct $\hat{\mathcal{S}}$ vs the final performance in the e-stop environment. E-stop results seem to be quite robust to poor experts and limited demonstrations.

2.6.2 Continuous environments

To experimentally evaluate the power of e-stops in continuous domains we took two classic continuous control problems: inverted pendulum control and the HalfCheetah-v3 environment from the OpenAI gym Brockman et al. [2016], and evaluated the performance of a deep reinforcement learning algorithm in the original environments as well as in modified versions of the environments with e-stops.

Although e-stops are more amenable to analysis in discrete MDPs, nothing fundamentally limits them from being applied to continuous environments in a principled fashion. The notion of state-hitting probabilities, $h(s)$, is meaningless in continuous spaces but the stationary (infinite-horizon), or average state (finite-horizon) distribution, $\rho_{\pi_e}(s)$ is well-defined and many techniques exist for density estimation in continuous spaces. Applying these techniques along with Corollary 2.4.2.1 provides fairly solid theoretical grounds for using e-stops in continuous problems.

For the sake of simplicity we implemented e-stops as min/max bounds on state values. For each environment we trained a number of policies in the full environments, measured their performance, and calculated e-stop min/max bounds based on roll-outs of the resulting best policy. We found

that even an approach as simple as this can be surprisingly effective in terms of improving sample complexity and stabilizing the learning process.

Inverted pendulum. In this environment the agent is tasked with balancing an inverted pendulum from a random starting semi-upright position and velocity. The agent can apply rotational torque to the pendulum to control its movement. We found that without any intervention the agent would initially just spin the pendulum as quickly as possible and would only eventually learn to actually balance the pendulum appropriately. However, the e-stop version of the problem was not tempted with this strange behavior since the agent would quickly learn to keep the rotational velocity to reasonable levels and therefore converged far faster and more reliably as shown in Fig. 2.4 (left).

Half cheetah. Fig. 2.4 (right) shows results on the HalfCheetah-v3 environment. In this environment, the agent is tasked with controling a 2-dimensional cheetah model to run as quickly as possible. Again, we see that the e-stop agents converged much more quickly and reliably to solutions that were meaningfully superior to DDPG policies trained on the full environment. We found that many policies without e-stop interventions ended up trapped in local minima, e.g. flipping the cheetah over and scooting instead of running. Because e-stops were able to eliminate these states altogether, policies trained in the e-stop regime consistently outperformed policies trained in the standard environment.

Broadly, we found training with e-stops to be far faster and more robust than without. In our experiments, we considered support sets $\hat{\mathcal{S}}$ to be axis-aligned boxes in the state space. It stands to reason that further gains could be squeezed out of this framework by estimating $\rho_{\pi_e}(s)$ more prudently and triggering e-stops whenever our estimation, $\hat{\rho}_{\pi_e}(s)$, falls below some threshold. In general, results will certainly be dependent on the structure of the support set used and the parameterization of state space, but our results suggest that there is promise in the tasteful application of e-stops to continuous RL problems.

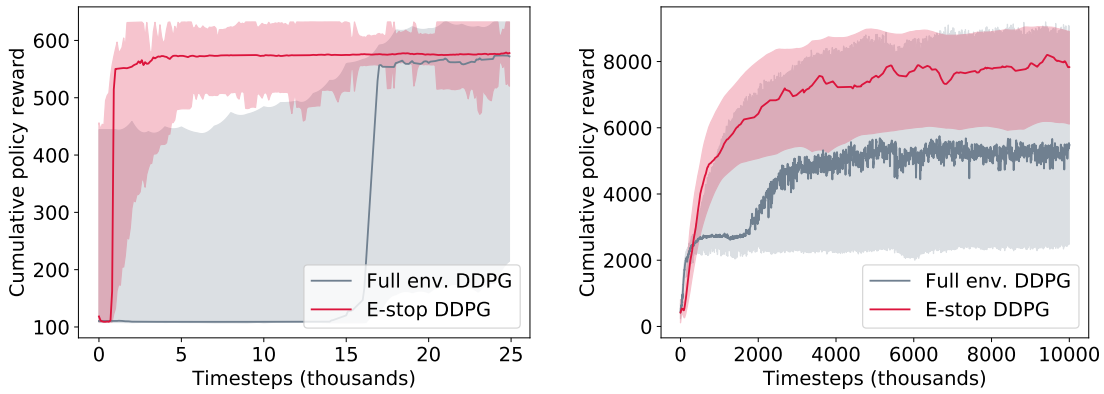

Figure 2.4: *Left:* DDPG results on the pendulum environment. *Right:* Results on the HalfCheetah-v3 environment from the OpenAI gym. All experiments were repeated with 48 different random seeds. Note that in both cases e-stop agents converged much more quickly and with lower variance than their full environment counterparts.

2.7 Types of support sets and their tradeoffs

In the previous sections, we proposed reset mechanisms based on a continuous or discrete state support set $\hat{\mathcal{S}}$. In this section, we describe alternative support set constructions and their respective trade-offs.

At the most basic level, consider the sequence of sets $\mathcal{S}^1_{\pi_e}, \ldots, \mathcal{S}^H_{\pi_e}$, defined by $\mathcal{S}^t_{\pi_e} = \{s | \rho^t_{\pi_e} > \epsilon\}$ for some small ϵ. Note that the single set $\hat{\mathcal{S}}$ we considered in Section 2.4.2 is the union of these sets when $\epsilon = 0$. The advantage of using a sequence of time-dependent support sets is that $\mathcal{S}_{\pi_e}$ may significantly over-support the expert's state distribution at any time and not reset when it is desirable to do so, i.e. $s_t \in \mathcal{S}_{\pi_e}$ but $s_t \notin \mathcal{S}^t_{\pi_e}$ for some $t > 0$. The downside of using time-dependent sets is that it increases the memory complexity from $\mathcal{O}(|\mathcal{S}|)$ to $\mathcal{O}(|\mathcal{S}|H)$. Further, if the state distributions $\rho^1_{\pi_e}, \ldots, \rho^H_{\pi_e}$ are similar, then using their union effectively increases the number of demonstrations by a factor of H.

To illustrate a practical scenario where it is advantageous to use time-dependent sets, we

revisit the example in Fig. 2.1, where an agent navigates from a start state s_0. $\mathcal{S}_{\pi_e}$ does not prevent π from remaining at s_0 for the duration of the episode, as π_e is initialized at this state and thus $s_0 \in \mathcal{S}_{\pi_e}$. Clearly, this type of behavior is undesirable, as it does not move the agent towards its goal. However, the time-dependent sets would trigger an intervention after only a couple time steps, since $s_0 \in \mathcal{S}^0_{\pi_e}$ but $s_0 \in \mathcal{S}^t_{\pi_e}$ for some $t > 0$.

Finally, we propose an alternative support set based on visitation counts, which balances the trade-offs of the two previous constructions $\mathcal{S}_{\pi_e}$ and $\{\mathcal{S}^1_{\pi_e}, \ldots, \mathcal{S}^H_{\pi_e}\}$. Let $s_f \in \mathbb{N}_0^{|\mathcal{S}|}$ be an auxiliary state, which denotes the number of visits to each state. Let $f(s) = \sum_{t=1}^{H} \mathbb{I}\{s \in \mathcal{S}^t_{\pi_e}\}$ be the visitation count to state s by the demonstrator. The modified MDP in this setting is defined by

$$P_{\hat{\mathcal{S}}}(s, s_f, a, s', s'_f) = \begin{cases} P(s, a, s'), & \text{if } s_f \leq f(s), s'_f = s_f + e_s \\ 1, & \text{if } s_f > f(s), s' = s_{\text{term}}, s'_f = s_f + e_s \\ 0 & \text{else} \end{cases} \tag{2.9}$$

$$R_{\hat{\mathcal{S}}}(s, s_f, a, s') = \begin{cases} R(s, a, s'), & \text{if } s_f \leq f(s) \\ 0, & \text{else} \end{cases}$$

where e_s is the one-hot vector for state s. In other words, we terminate the episode with no further reward whenever the agent visits a state more than the demonstrator. The mechanism in Eq. (2.9) has memory requirements independent of H yet fixes some of the over-support issues in $\mathcal{S}_{\pi_e}$. The optimal policy in this MDP achieves at least as much cumulative reward as π_e (by extending Theorem 2.4.1) and can be extend to the imperfect e-stop setting in Section 2.4.3.

We leave exploration of these and other potential e-stop constructions to future work.

2.8 Discussion

We introduced a general framework for incorporating e-stop interventions into any reinforcement learning algorithm, and proposed a method for learning such e-stop triggers from state-only observations. Our key insight is that only a small support set of states may be necessary to operate effectively towards some goal, and we contribute a set of bounds that relate the performance of an

agent trained in this smaller support set to the performance of the expert policy. Tuning the size of the support set allows us to efficiently trade off an asymptotic sub-optimality gap for significantly lower sample complexity.

Empirical results on discrete and continuous environments demonstrate significantly faster convergence on a variety of problems and only a small asymptotic sub-optimality gap, if any at all. We argue this trade-off is beneficial in problems where environment interactions are expensive, and we are less concerned with achieving no-regret guarantees as we are with small, finite sample performance. Further, such a trade-off may be beneficial during initial experimentation and for bootstrapping policies in larger state spaces. For example, we are particularly excited about graduated learning processes that could increase the size of the support set over time.

In larger, high dimensional state spaces, it would be interesting and relatively straightforward to apply anomaly detectors such as one-class SVMs Laskey et al. [2016] or auto-encoders Richter and Roy [2017] within our framework to implicitly construct the support set. Our bounds capture the reduction in exploration due to reducing the state space size, which could be further tightened by incorporating our a priori knowledge of s_{term} and the ability to terminate trajectories early.

Chapter 3

POLICY LEARNING WITH DIFFERENTIABLE SIMULATION

3.1 Introduction

Many robotic control problems can be reduced to finding a desirable trajectory through a physical system governed by piece-wise smooth dynamics, e.g. control of ground vehicles, robot arms, quadrotors, and so forth. When the dynamics of a system are known we can efficiently optimize a trajectory via first-order methods, constructed using explicit gradients of the dynamics. We can use these gradients to solve open-loop trajectory optimization problems [Carius et al., 2019, Orsolino et al., 2018, Carius et al., 2018] or to train a closed-loop feedback controller, i.e. a policy [Hirose et al., 2019, Faulkner et al., 2018, Kim et al., 2018]. In this chapter we focus on the latter problem of policy learning, although our insights can be applied equally well to trajectory optimization.

Computing exact policy gradients for most continuous-time dynamical systems proves intractable. Policy gradient algorithms rely on discrete numerical methods to estimate a robot's trajectory under a policy, and the sensitivity of this trajectory to the robot's actions dictated by the policy. The standard numerical estimate of policy gradients is back-propagation through time (BPTT); this algorithm abandons study of the continuous problem and immediately discretizes time, computing exact policy gradients with respect to discretized dynamics. The discretization step-size controls the accuracy of BPTT, resulting in a tradeoff between the accuracy of the gradient estimates and the computational cost of computing an estimate.

In this chapter we defer discretization and begin by characterizing the continuous-time policy gradient. Rather than computing an exact gradient of a discretized system, we instead compute an approximate gradient of the continuous system. When we discretize to compute this approximation, we do so with the explicit goal of constructing an efficient estimate of the continuous-time policy gradient. This approach takes inspiration from the numerical ODE literature, as well as the neural

ODE [Chen et al., 2018]. We are motivated by the following questions:

Q1: Can we compute more accurate estimates of the policy gradient? BPTT introduces numerical error by discretizing the robot trajectory under a policy. This error injects variance into the learning process, which could slow down learning. By computing a more accurate estimate of the policy gradient, can we optimize a policy in fewer iterations?

Q2: Can we compute policy gradient estimates more efficiently? Suppose we want a target level of accuracy for our policy gradient estimates. We can achieve the target accuracy by controlling the discretization step-size for BPTT. By using better numerical algorithms, can we achieve the same accuracy with a smaller computational budget?

We give affirmative answers to both **Q1** and **Q2**. We propose a new policy gradient estimator (Section 3.3) which we call the Continuous-Time Policy Gradient (CTPG). This estimator strictly improves upon the BPTT estimator, in the sense that the tradeoff curve between gradient accuracy and computational budget for CTPG Pareto-dominates the tradeoff curve for BPTT. In the spirit of algorithms like SGD, we ask: is it possible to sacrifice accuracy of the gradient estimates in order to optimize more efficiently? We find that a certain amount of numerical accuracy is required of our policy gradient estimator in order for optimization to succeed: unlike SGD, poor numerical estimates of policy gradients are not unbiased. However, in a variety of experimental settings (Section 3.4) we demonstrate that replacing expensive BPTT estimates with relatively inexpensive CTPG estimates of comparable accuracy results in faster overall learning.

3.2 Learning with Policy Gradients

Our goal is to learn how to act within a deterministic physical system while minimizing a cost functional. Given a model of the dynamics $f : \mathbb{R}^d \times \mathbb{R}^k \to \mathbb{R}^d$ and time-varying control inputs (actions) $u : [0, \infty) \to \mathbb{R}^k$, the state of the system $x : [0, \infty) \to \mathbb{R}^d$ is governed by a first-order ordinary differential equation

$$\frac{dx(t)}{dt} = f(x(t), u(t)). \tag{3.1}$$

Guided by a local cost (reward function) $w : \mathbb{R}^d \times \mathbb{R}^k \to \mathbb{R}$ of taking action u in state x, and a terminal cost $J : \mathbb{R}^d \to \mathbb{R}$, we seek controls u that minimize the global cost over a trajectory of

length T:

$$\begin{aligned} \underset{u(t)}{\text{minimize}} \quad & \int_0^T w(x(t), u(t))\, dt + J(x(T)) \\ \text{subject to} \quad & \frac{dx(t)}{dt} = f(x(t), u(t)). \end{aligned} \tag{3.2}$$

We are interested in learning feedback controllers in the form of policies $u(t) = \pi_\theta(x(t))$ with parameters θ. For any fixed initial conditions $x(0) = x_0$, the optimization Eq. equation 3.2 is a trajectory optimization problem with parameters θ. For any fixed policy $\pi_\theta(x(t))$, computation of the trajectory $x(\cdot)$ is an initial value problem that is completely determined by the starting state x_0. The global loss of following the policy π_θ along a trajectory $x(\cdot)$ is given by

$$\mathcal{L}(x(0), \theta) \triangleq \int_0^T w(x(t), \pi_\theta(x(t)))\, dt + J(x(T)). \tag{3.3}$$

We seek to learn a policy that generalizes over a distribution of initial states $x_0 \sim \rho_0$, motivating us to minimize the expected loss $\mathcal{L}(\theta) \triangleq \mathbb{E}_{x_0 \sim \rho_0} \mathcal{L}(x_0, \theta)$.

Adjoint sensitivity analysis [Pontryagin et al., 1964] characterizes $\frac{\partial \mathcal{L}(x_0, \theta)}{\partial \theta}$ in a form that is amenable to computational approximation. Consider the adjoint process $\alpha(t) \triangleq \frac{\partial \mathcal{L}(x_0, \theta)}{\partial x(t)}$; the evolution of this adjoint process for a trajectory x can be described by the dynamics

$$\frac{d\alpha(t)}{dt} = -\alpha(t)^\top \frac{\partial f}{\partial x(t)} - \frac{dw(x(t), \pi_\theta(x(t)))}{dx(t)}. \tag{3.4}$$

These dynamics can be compared to Chen et al. [2018], with the notable distinction that the loss in Eq. equation 3.3 is path-dependent. This causes the instantaneous loss $w(x(t), u(t))$ to accumulate continuously in the adjoint process $\alpha(t)$. The adjoint process itself is constrained by the final value $\alpha(T) = \frac{\partial J}{\partial x(T)}$. Sensitivity of the loss to the parameters θ is given by

$$\frac{\partial \mathcal{L}(x_0, \theta)}{\partial \theta} = \int_0^T \alpha(t)^\top \frac{\partial f}{\partial u} \frac{\partial u}{\partial \theta} + \frac{\partial w}{\partial u} \frac{\partial \pi_\theta(x(t))}{\partial \theta}\, dt. \tag{3.5}$$

The integral in Eq. equation 3.5 is amenable to estimation using various numerical techniques, each of which is susceptible to different forms of numerical error; a stylized illustration of the behaviour of various gradient estimators is presented in Figure 3.1.

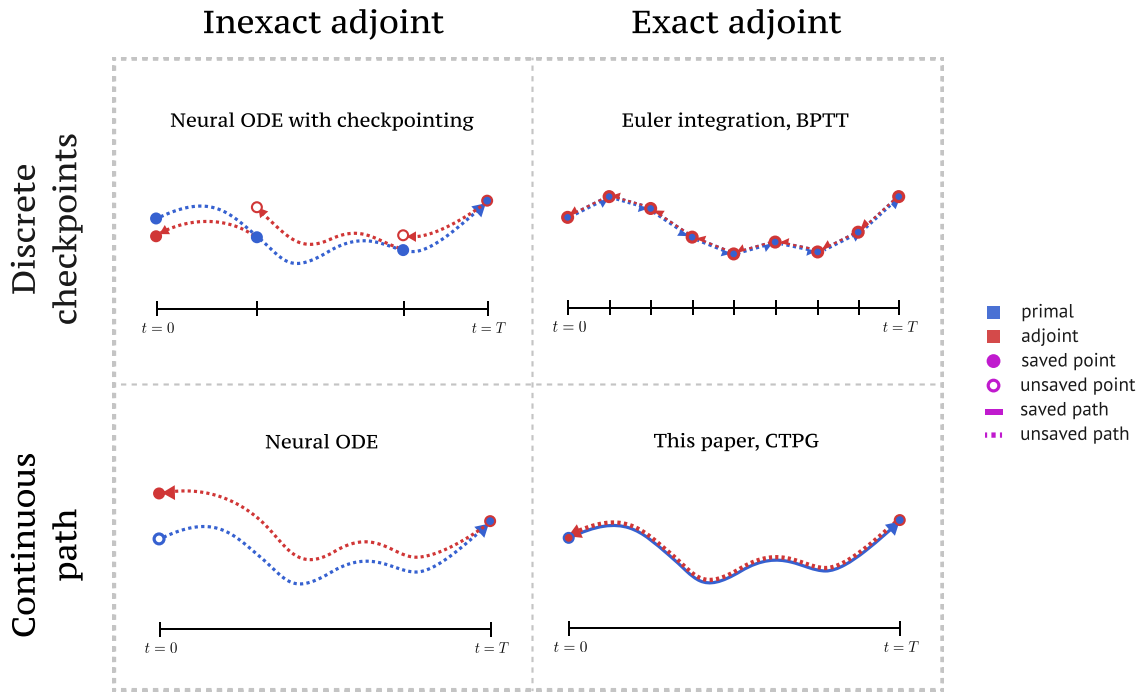

Figure 3.1: Comparing policy gradient estimators. Neural ODE (Sec. 3.3.3) approximates continuous trajectories, but accumulated error in its adjoint estimates. Neural ODE with checkpointing mitigates this error, but still accumulates error between checkpoints. Euler integration with BPTT (Sec. 3.3.2) stores a discretized trajectory, which prevents error accumulation, at the cost of a naive discretization. CTPG (Sec. 3.3.1) discretizes adaptively, and stores a spline approximation of the trajectory to avoid error accumulation in the adjoint.

3.3 Numerical Policy Gradient Estimators

Policy gradients can be calculated via the integral described in Eq. equation 3.5. This integral in turn depends upon the trajectory $x(t)$ and the adjoint process $\alpha(t)$. To estimate a policy gradient, we require solutions to the following three sub-problems:

1. The trajectory $x(t)$, governed by the dynamics in Eq. equation 3.1.

2. The adjoint process $\alpha(t)$, governed by the dynamics in Eq. equation 3.4.

3. The policy gradient accrued along $x(t)$ and $\alpha(t)$, described by the integral in Eq. equation 3.5.

The accumulated θ gradients, Eq. equation 3.5, can be described as the value at $t = 0$ of the process $g(t)$, with final value $g(T) = 0$, defined by the dynamics $\frac{dg(t)}{dt} = -\alpha(t)^\top \frac{\partial f}{\partial u}\frac{\partial u}{\partial \theta} - \frac{\partial w}{\partial u}\frac{\partial \pi_\theta(x(t))}{\partial \theta}$. Collectively, the trajectory process $x(t)$, the corresponding adjoint process $\alpha(t)$, and the accumulated loss $g(t)$ comprise the following two-point boundary value problem (BVP):

Initial Value	Dynamics	Final Value	
$x(0) = x_0,$	$\frac{dx(t)}{dt} = f(x(t), u(t)),$		(3.6)
	$\frac{d\alpha(t)}{dt} = -\alpha(t)^\top \frac{\partial f}{\partial x} - \frac{dw}{dx(t)},$	$\alpha(T) = \frac{dJ}{dx(T)},$	(3.7)
	$\frac{dg(t)}{dt} = -\alpha(t)^\top \frac{\partial f}{\partial u}\frac{\partial \pi_\theta}{\partial \theta} - \frac{\partial w}{\partial u}\frac{\partial \pi_\theta}{\partial \theta},$	$g(T) = 0.$	(3.8)

To understand how various subsystems of the policy gradient estimator interact, we highlight quantities computed by the physics simulator in red and quantities computed by automatic (or symbolic) differentiation in blue.

The most common approach to estimating $g(0) = \frac{\partial \mathcal{L}(x_0,\theta)}{\partial \theta}$ in the controls community is backpropagation through time (BPTT). The idea is to discretize the dynamics $x(t)$ with a constant time step h, after which sensitivities to the policy parameters θ are computed by the standard backpropagation algorithm. In Section 3.3.2 we describe how this algorithm can be interpreted as an approximate solution to the continuous boundary-value problem, using Euler integration to

estimate the dynamics $x(t)$, $\alpha(t)$, and $g(t)$. The numerical ODE community widely eschews Euler integration in favor of more sophisticated solvers. A natural question arises: can we improve upon naive Euler integration to construct better gradient estimates?

In Section 3.3.1 we introduce a new, general class of gradient estimators for this boundary value problem that we call the Continuous-Time Policy Gradient. In Section 3.3.2 we show how BPTT can be viewed as an instantiation of Continuous-Time Policy Gradients, using Euler integration as a numerical solver. The CTPG generalization allows us to replace Euler integration with a more sophisticated solver; in our experiments we opted for Runge-Kutta (RK4) and adaptive Adams-Moulton methods but these solvers can be seamlessly replaced by whatever solver is most suitable for a particular problem. In Section 3.3.3 we discuss the Neural ODE estimator and its inadequacy for use with stabilizing control problems. And in Appendix B.3 we discuss alternate methods for solving the BVP that may be of interest in specialized settings.

3.3.1 Continuous-Time Policy Gradients

We propose to solve the boundary value problem described by Eqs. equation 3.6, equation 3.7, and equation 3.8 using a forward-backward meta-algorithm, which is analogous to (but distinct from) the forward and backward passes of an automatic differentiation system (and more generally, the forward-backward approach to dynamic programming). First, we use a numerical solver to estimate the solution to the initial value problem given by Eq. equation 3.6 (the forward pass). Crucially we store the estimated trajectory for later use, incurring a storage cost off $O(s)$ where s is the number of steps visited by the numerical solver. Second, we use a numerical solver to estimate the solution to initial value problem given by Eq. equation 3.7 (the backward pass) using the cached trajectory computed in the forward pass. As we compute the backward pass we accumulate an estimate of the integral in Eq. equation 3.8, the total sensitivity of the loss to θ along the estimated trajectory. Details are presented in Algorithm 2.

Figure 3.2 illustrates performance of Continuous-Time Policy Gradients (using the RK4 solver) and several other algorithms. The dynamics are given by the linear-quadratic regulator (LQR) and the policy is fixed to be the optimal LQR policy. By adjusting the error tolerance of the RK4 solver,

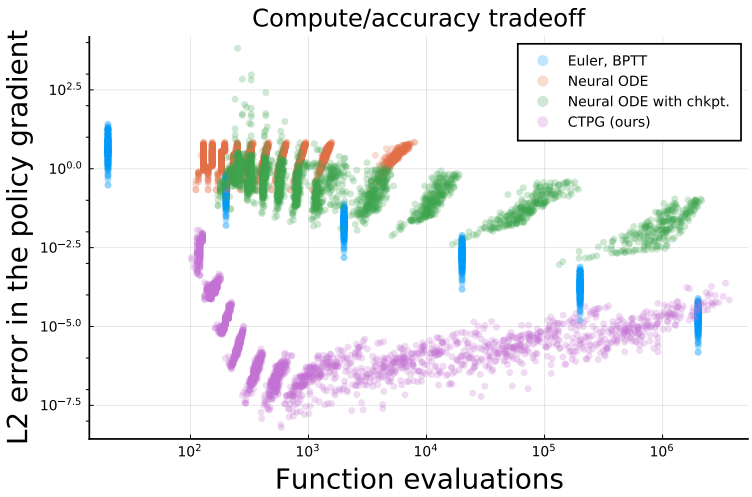

Figure 3.2: Approaching the Pareto frontier: Euler integration and other methods are Pareto dominated by CTPG. Lower and to the left is better.

we construct a trade-off curve between the accuracy of the CTPG estimates and the computational cost of computing an estimate. The CTPG estimator Pareto-dominates the other estimators. In this sense, we argue that the other estimators are inadmissible: e.g. for any BPTT estimator, there is a CTPG estimator that is either (1) more accurate at the same level of performance or (2) more efficient at the same level of accuracy.

The effectiveness of CTPG is attributable to the efficiency of the RK4 solver, which can estimate the trajectory and adjoint paths while visiting many fewer states than a comparably accurate Euler solver (BPTT). This also means that CTPG is more memory efficient than BPTT: both algorithms require storage that grows linearly in the number of states visited in the forward pass. The (possibly surprising) under-performance of the Neural ODE estimator is explained in Section 3.3.3.

3.3.2 Backpropagation Through Time

Backpropagation through time (BPTT) is simply the application of reverse-mode automatic differentiation (AD) to computations involving a "time" axis. Like CTPG, AD proceeds with a forward and backward pass. In the forward pass, the trajectory $x(t)$ is approximated using discrete, fixed-

Algorithm 2 Continuous-Time Policy Gradients

Given: Differentiable physics simulator $f(x, u)$, cost/reward function $w(x, u)$, and a numerical ODE solver Solve[initial_conditions, dynamics]

Input: Policy $\pi_\theta(x)$, initial state x_0

Result: An approximation of $\frac{\partial \mathcal{L}(x_0,\theta)}{\partial \theta}$

Forward pass: (compute and store an approximation of the trajectory $x : [0, T] \to \mathbb{R}^d$)

$$\tilde{x}(\cdot) \leftarrow \text{Solve}\left[x(0) = x_0, \frac{dx(t)}{dt} = f(x, \pi_\theta(x))\right].$$

Backward pass: (compute an approximation of the Pontryagin adjoints $\alpha : [T, 0] \to \mathbb{R}^d$)

$$\tilde{\alpha}(\cdot) \leftarrow \text{Solve}\left[\alpha(T) = \frac{dJ}{d\tilde{x}(T)}, \frac{d\alpha(t)}{dt} = -\alpha(t)^\top \frac{\partial f}{\partial \tilde{x}(t)} - \frac{dw}{d\tilde{x}(t)}\right].$$

Return:

$$\int_0^T \tilde{\alpha}(t)^\top \frac{\partial f}{\partial u}\frac{\partial u}{\partial \theta} + \frac{\partial w}{\partial u}\frac{\partial \pi_\theta}{\partial \theta}\, dt \text{ as an approximation of } \frac{\partial \mathcal{L}(x_0, \theta)}{\partial \theta}.$$

length linearized steps (see Figure 3.1); this is equivalent to using the Euler numerical integration method to solve the initial value problem Eq. equation 3.6. States computed in the forward pass are stored, incurring an $O(1/h)$ memory cost, and avoiding the need to recompute these values in the backward pass. In the backward pass of AD computes adjoints of the forward computation graph, using the values $x(t)$ stored during the forward pass. This is equivalent to using Euler integration to solve the final value problems described by Eqs. equation 3.7 and equation 3.8.

Backpropagation is an exact algorithm, in the sense that it computes exact gradients of the loss for the discretized dynamics. But these dynamics are only an approximation of the continuous trajectory $x(\cdot)$, and therefore the gradients only approximate the desired value $g(0)$. Computing an accurate estimate of the continuous trajectory using Euler integration requires an excessively small step size h, and a correspondingly large memory allocation. Furthermore, the resulting computation graph is very deep, which could make learning difficult [Quaglino et al., 2020].

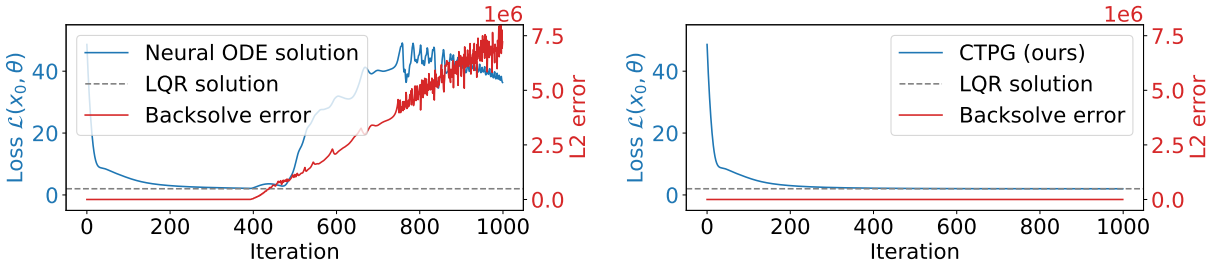

Figure 3.3: Left: The Neural ODE estimator used to learn an LQR policy. The trajectory loss $\mathcal{L}(x_0, \theta)$ is shown in blue, and the discrepancy $||x(0) - \tilde{x}(0)||^2$ is shown in red where $\tilde{x}(t)$ is the "recovered" path taken by the adjoint process. The discrepancy between the forward and reverse paths is negligible until the control learns to stabilize the system, at which point it becomes impossible to recover $x(0)$ from $x(T)$. Right: The same problem, but with CTPG. Unlike the Neural ODE, CTPG records a lightweight spline estimate of $x(t)$.

3.3.3 The Neural ODE Algorithm

To efficiently compute the backward pass, we rely on a stored spline of states s visited from the forward pass, leading to an algorithm with $O(s)$ space complexity. For many physical systems, the dynamics $f(x, u)$ are "invertible"; this invites us to consider algorithms with constant space complexity, recomputing the state variables $x(t)$ in the backward pass following the reverse dynamics $-f(x, u)$ [Chen et al., 2018]. Constant-memory algorithms are intriguing, but we observe that their results are numerically unstable for stabilizing control problems. We visualize this instability for an LQR system in Figure 3.3.

In principle, a trajectory $x(t)$ is uniquely defined by either its initial value $x(0)$ thanks to the Picard-Lindelöf theorem. But for control problems that seek to stabilize the system towards some fixed goal-point, it is numerically unstable to reconstruct the trajectory from its final state; moreover, this instability becomes more pronounced as the policy learns to more effectively achieve its goal, and computing the inverse trajectory becomes increasingly chaotic.

To better formalize this notion, we appeal to linear stability theory. For a system $\frac{dx}{dt} = \phi(x)$,

linear stability analysis suggests inspecting the eigenspectrum of $\frac{d\phi}{dx}$: if $\frac{d\phi}{dx}$ has all eigenvalues with negative real part the system is said to be linearly stable, and if there exists an eigenvalue with positive real part it is said to be linearly unstable. Stabilizing control problems by their nature demand solutions such that the system has negative eigenvalues $-\lambda \ll 0$, as small as possible subject to control costs. This becomes catastrophic in the reverse-direction when we are faced with a linearly unstable system with eigenvalues $\lambda \gg 0$! Furthermore, as we discuss in Appendix B.2, the neural ODE backpropagation dynamics are in fact linearly unstable everywhere.

3.4 Experiments

To better understand CTPG's behavior, we compare CTPG with BPTT for a variety of control problems. We restrict our experiments to settings in which first-order policy gradients optimization can effectively discover the optimal policy; extension of these local search algorithms to global policy optimization is beyond the scope of this work. We begin with a relatively straightforward control problem in Sec. 3.4.1 for which we wrote a simple differentiable simulator. We then evaluate CTPG for a task defined in an existing differentiable physics engine from Hu et al. [2020] in Sec. 3.4.2. In Sec. 3.4.3 we evaluate CTPG for a task defined in the MuJoCo physics simulator, using finite difference approximations to the gradients of the physical system. We conclude in Sec. 3.4.4 with a challenging quadrotor control problem running in a purpose-built differentiable flight simulator. Code to reproduce our experiments is available at `https://github.com/samuela/ctpg`.

3.4.1 Differential Drive Robot

The differential drive is a "Roomba-like" robot with two wheels that are individually actuated to rotate or advance the robot. We use the dynamics defined in LaValle [2006], but control the torque applied to each wheel rather than directly controlling their angular velocities. We initialized the robot to random positions and rotations throughout the plane and learn a policy that drives it to the origin. See Figure 3.4 for a visualization of results. Both CTPG and BPTT are able to learn working policies, but CTPG is able to do so more efficiently. Videos of the training process are

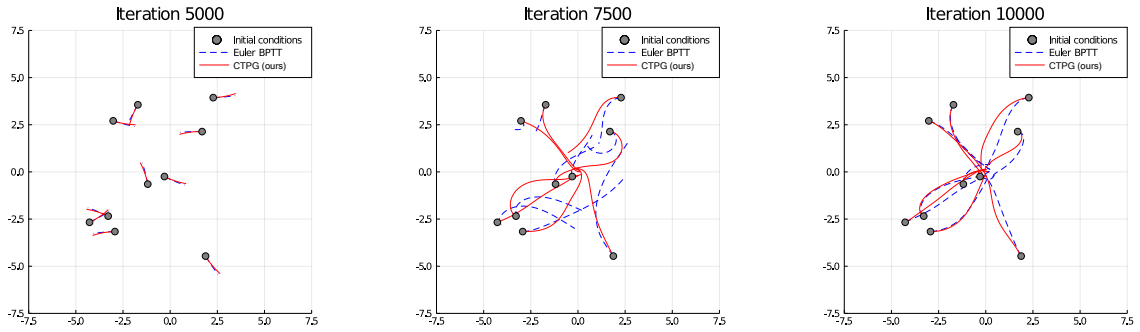

Figure 3.4: Stages of training policies to control a differential drive robot to stabilize towards the origin. Each curve denotes the trajectory of the robot in the (x, y) plane under the respective policies. A full video is available in the supplementary material.

available in the supplementary material.

3.4.2 DiffTaichi Electric Field Control

To test CTPG's ability to work with third-party differentiable physics simulators, we integrated and compared against the DiffTaichi (meta) physics engine [Hu et al., 2020, 2019]. DiffTaichi (now part of Taichi) is a differentiable, domain-specific language for writing physics simulators. We tested against their electric field control experiment in which eight fixed electrodes placed in a 2-D square modulate their charge driving a red ball with a static charge around the square. The red ball experiences electrostatic forces given by Coulomb's law, $\mathbf{F} = k_e \frac{q_1 q_2}{|\mathbf{r}|^2} \hat{\mathbf{r}}$ [Zangwill, 2013].

We evaluate performance both in terms of wall-clock time and in terms of the number of oracle function calls made to the simulator, namely the combined evaluations of f, $\partial f/\partial x$, and $\partial f/\partial u$. Calls to the simulator constitute the vast majority of the run time, so we find this to be the most accurate hardware-independent measure of performance. We present the results in Fig. 3.5. We find that CTPG reliably outperforms BPTT both in terms of wall-clock time and number of function evaluations, even with DiffTaichi's exact BPTT configuration.

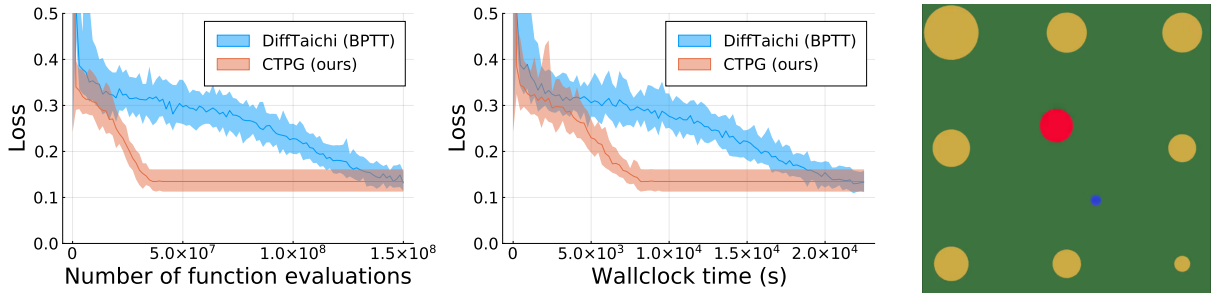

Figure 3.5: The DiffTaichi electric experiment involves modulating the charges of the 8 yellow electrodes to push the red charge to follow the blue dot (right). We found that CTPG performance was dominant in terms of the number of function evaluations (left) as well as wall-clock time (center) across 32 random trials.

3.4.3 MuJoCo Cartpole

MuJoCo [Todorov et al., 2012] is a widely used physics simulator in both the robotics and reinforcement learning communities. In contrast to DiffTaichi (Section 3.4.2) MuJoCo models are not automatically differentiable, but rather they are designed to be finite differenced. We demonstrate that BPTT and CTPG can be applied to such black-box physics engines. These finite difference derivatives are used in Eqs. 3.7 and 3.8 alongside the automatically differentiated policy derivatives. We test on the Cartpole swing-up task specified by the dm_control suite [Tunyasuvunakool et al., 2020], a classic non-linear control problem. This model was re-implemented in the Lyceum framework [Summers et al., 2020] with the Euler integration timestep used for BPTT kept the same as the model specification (0.01 seconds). Total dynamics evaluations included those used for finite differencing.

Results are presented in Fig. 3.6. We find that CTPG needs significantly fewer dynamics evaluations than BPTT per learning instance. While both techniques eventually reach a similar level of performance for this task, CTPG demonstrates substantial efficiency gains.

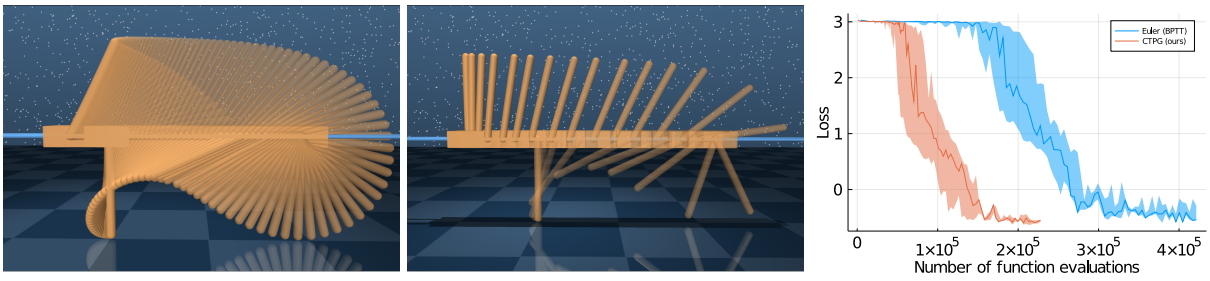

Figure 3.6: Visualization of all states visited by a policy trained with BPTT (left) and CTPG (middle). Fewer states are visited by CTPG during training leading to more efficient learning (right). Error bars on the training plot show the 0.25-0.75 quantile range of 5 random seeds.

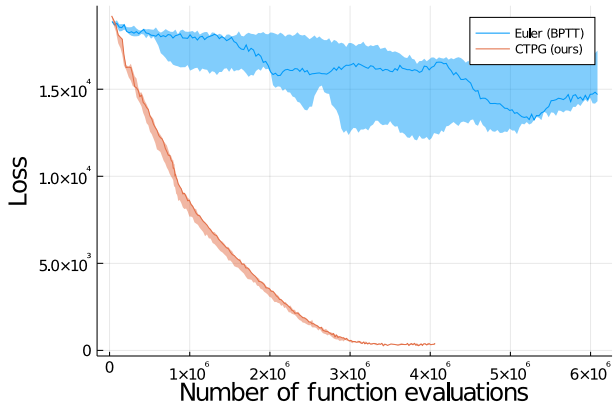

Figure 3.7: CTPG learns policies to stabilize the quadrotor towards the origin, while Euler integration and BPTT–with the same number of function evaluations–can not.

3.4.4 Quadrotor Control

Quadrotors are notoriously difficult to control, especially given the strict real-time compute requirements of a flying aircraft, and tradeoffs between compute power, weight, and power draw. As such, they make for a strong use case of the policy learning tools that CTPG brings to bear. We test CTPG's ability to learn a small, efficient neural network controller that flies a quadrotor.

We implemented a differentiable quadrotor simulator based on the dynamics from Sabatino [2015], and train policies to stabilize a quadrotor at the origin from a wide distribution of initial positions, orientations, and velocities. Our experimental setup is similar to that of Tang et al. [2018]. Results are presented in Fig. 3.7. As a flying robot, the quadrotor dynamics are inherently unstable. As a result, we found that Euler integration is often not able to integrate trajectories to sufficient accuracy for policy learning using a similar sample budget to CTPG.

3.5 Related Work

Neural ODEs. Chen et al. [2018] introduced neural ordinary differential equations, based on the adjoint sensitivity analysis method of Pontryagin et al. [1964] and using backwards dynamics to reconstruct $x(t)$ as described in Eq. 3.6. Recent work by Du et al. [2020] and Quaglino et al. [2020] studied using Neural ODEs for system identification as opposed to policy learning. Gholaminejad et al. [2019] explored a gradient checkpointing scheme for Neural ODEs.

Differentiable physics. Differentiable physics engines have recently attracted attention as a richer alternative to conventional "black-box" simulators. de Avila Belbute-Peres et al. [2018] introduced a differentiable rigid-body physics simulator based on analytic derivatives through physics defined via a linear complementarity problem. Qiao et al. [2020] expanded on the types of physical interactions that could be efficiently simulated and differentiated. Li et al. [2020, 2019] and Battaglia et al. [2016] proposed developing differentiable physics engines by learning them from data. Schenck and Fox [2017, 2018] built a differentiable simulator for liquids with applications to robotics. Hu et al. [2019, 2020] created a differentiable domain-specific language for the creation of physics simulators, enabling backpropagation through a variety of different

physical phenomena.

Trajectory optimization and control theory. Pontryagin et al. [1964] introduced necessary conditions for optimality of a continuous trajectory, drawing from the calculus of variations. An alternative perspective, the Hamilton-Jacobi-Bellman equation, provides necessary and sufficient conditions for optimality of policies [Speyer and Jacobson, 2010]. Prior work describes the analytic adjoints for backpropagation through time [Tedrake, Robinson and Failside, 1987, Werbos, 1988, Mozer, 1989]. Unlike the majority of work in trajectory optimization and control, we use Pontryagin's maximum principle to learn global feedback policies instead of fitting single trajectories. Concurrently Jin et al. [2020] explored differentiating through Pontryagin's maximum principle. However, they still considered standard time discretizations, eg. BPTT. Other works have investigated similar forward-backward schemes for solving the adjoint equations [McAsey et al., 2012].

Numerical methods for ODEs. The study of numerical methods for solving ODEs has a long and rich history, dating back to Euler's method [Hairer et al., 1993]. In this work we leveraged higher-order explicit solvers with adaptive step sizes, especially Runge-Kutta [Wang et al., 2021, Tsitouras, 2011] and Adams-Moulton methods [Shampine and Watts, 1979, LeVeque, 2007]. For a thorough treatment of ODE solvers we refer the reader to Hairer et al. [1993].

3.6 Discussion

In this chapter, we introduced Continuous-Time Policy Gradients (CTPG), a new class of policy gradient estimator for continuous-time systems. By leveraging advanced integrators developed by the numerical ODE community, CTPG enjoys superior performance to the standard back-propagation through time (BPTT) estimator. Although we studied policy gradients in this work, it's worth noting that CTPG is general enough to operate as a "layer" within any larger differentiable model. This work could be extended to other related scenarios: systems with non-deterministic dynamics (using stochastic ODE solvers) infinite-time planning problems (using steady state ODE solvers) and sensor/actuation latency (using delay differential equations solvers). Adjoints for these alternatives have already been implemented in Rackauckas and Nie [2017]. Another

compelling direction for future work would be the incorporation of value function learning via extension to the Hamilton-Jacobi-Bellman equation. We are interested in the following questions: (1) When and why do policy gradient optimizations fail? (2) How can we properly incorporate policy gradient estimates into a global policy search?

Chapter 4

DIFFERENCES BETWEEN POLICY LEARNING AND SUPERVISED LEARNING

4.1 Introduction

We investigate the surprising effectiveness of stochastic gradient descent (SGD) algorithms on the massive non-convex optimizations problems of deep learning. In particular, we are motivated by a few phenomena:

1. Why does SGD thrive in the optimization of massive non-convex deep learning loss landscapes, despite being noticeably less robust in other non-convex optimization settings like policy learning [Ainsworth et al., 2021], trajectory optimization [Kelly, 2017], and recommender systems [Kang et al., 2016]?

2. Where are all the local minima? When linearly interpolating between initialization and final trained weights, why does the loss smoothly, monotonically decrease [Goodfellow and Vinyals, 2015, Frankle, 2020, Lucas et al., 2021, Vlaar and Frankle, 2021]?

3. How is it that two independently trained models with different random initializations and data batch orders inevitably achieve nearly identical performance? Furthermore, why do their training loss curves look identical?

We especially argue that this final phenomenon points to the existence of some yet uncharacterized invariance(s) in the training dynamics, such that independent training runs exhibit identical characteristics. Brea et al. [2019] noted the permutation symmetries of hidden units in neural networks. Briefly: one can swap any two units of a hidden layer in a network and – assuming weights are adjusted accordingly – the functionality of the network remains unchanged.

Concurrent to our work, Entezari et al. [2021] conjectured that these permutation symmetries may allow us to linearly connect points in weight space.

Conjecture 1 (Permutation invariance, informal [Entezari et al., 2021])**.** Most SGD solutions belong to a set whose elements can be permuted in such a way that there is no barrier on the linear interpolation between any two permuted elements.

We refer to such solutions as being *linearly mode connected* (LMC) [Frankle et al., 2020]. If true, Conjecture 1 has the potential to materially expand our understanding of how SGD works in the context of deep learning, and offers a credible explanation for these three phenomenon in particular.

Contributions. In this paper, we attempt to uncover what invariances may be responsible for these three phenomena and the unreasonable effectiveness of SGD in deep learning. We make the following contributions:

1. **Matching methods.** We propose three novel algorithms to align the weights of two independently trained models, grounded in concepts and techniques from combinatorial optimization. Where appropriate, we prove hardness results for these problems and propose approximation algorithms.

2. **Relationship to SGD.** We demonstrate by means of counterexample that linear mode connectivity is an emergent property of SGD training, as opposed to model architectures. We connect this result to prior work on the implicit biases of SGD.

3. **Experiments, including zero-barrier LMC for realistic ResNets.** Empirically, we explore the existence of linear mode connectivity modulo permutation symmetries in experiments across MLPs, CNNs, and ResNets trained on MNIST, CIFAR-10, and CIFAR-100. We contribute the first ever demonstration of zero-barrier LMC between two completely independently trained large ResNet models. We explore the relationship between LMC and model width, as well as training time. In addition, we exhibit our methods' ability to

ARCHITECTURE	NUM. PERMUTATION SYMMETRIES
MLP (3 layers, 512 width)	$10 \wedge 3498$
VGG16	$10 \wedge 35160$
ResNet50	$10 \wedge 55109$
Atoms in the observable universe	$10 \wedge 82$

Table 4.1: Permutation symmetries of deep learning models vs. an upper estimate on the number of atoms in the known, observable universe. Deep learning loss landscapes contain incomprehensible amounts of geometric repetition.

combine models trained on independent datasets into a merged model that outperforms both input models.

4.2 Background

We consider models following an L-layer multi-layer perceptron (MLP) architecture [Bishop, 2007],

$$f(\boldsymbol{x}; \Theta) = \boldsymbol{z}_{L+1}, \quad \boldsymbol{z}_{\ell+1} = \sigma(\boldsymbol{W}_\ell \boldsymbol{z}_\ell + \boldsymbol{b}_\ell), \quad \boldsymbol{z}_1 = \boldsymbol{x}$$

where σ denotes an element-wise nonlinear activation function. Note that our methods are not inherently limited to MLP architectures, but we will proceed with MLPs for their ease of presentation. Furthermore, we assume the existence of a loss function, $\mathcal{L}(\Theta)$ measuring the suitability of a particular choice of weights Θ towards some goal, say fitting to a training dataset.

Central to our investigation is the phenomenon of *permutation symmetries* of weight space. For example given Θ, we can take any intermediate layer, ℓ, of the model and apply some permutation, denoted by permutation matrix $\boldsymbol{P}$, to its output

$$\boldsymbol{z}_{\ell+1} = \boldsymbol{P}^\top \boldsymbol{P} \boldsymbol{z}_{\ell+1} = \boldsymbol{P}^\top \boldsymbol{P} \sigma(\boldsymbol{W}_\ell \boldsymbol{z}_\ell + \boldsymbol{b}_\ell) = \boldsymbol{P}^\top \sigma(\boldsymbol{P}\boldsymbol{W}_\ell \boldsymbol{z}_\ell + \boldsymbol{P}\boldsymbol{b}_\ell)$$

which is kosher so long as $\boldsymbol{P} \in S_d$ is a permutation matrix and σ operates element-wise. (We denote the set of all $d \times d$ permutation matrices, or symmetric group, as S_d, perhaps to the chagrin of pure mathematicians.) It follows that as long as we reorder the weights of layer $\ell + 1$ according to $\boldsymbol{P}^\top$, we will have a functionally equivalent model. To be precise, if we define Θ' to be identical to Θ with the exception of

$$\boldsymbol{W}'_\ell = \boldsymbol{P}\boldsymbol{W}_\ell, \quad \boldsymbol{b}'_\ell = \boldsymbol{P}\boldsymbol{b}_\ell, \quad \boldsymbol{W}'_{\ell+1} = \boldsymbol{W}_{\ell+1}\boldsymbol{P}^\top$$

then the two models are functionally equivalent: $f(\boldsymbol{x}; \Theta) = f(\boldsymbol{x}, \Theta')$ for all inputs $\boldsymbol{x}$. This implies that for any trained weights Θ, there is not just one such weight assignment but an entire equivalence class of weight assignments, and your convergence to any one specific element of this equivalence class – as opposed to any of the others – is determined only by random seed and the whims of God. We denote a functionality-preserving permutation of weights as $\pi(\Theta)$.

Consider the task of reconciling the weights of two, independently trained models – model A and model B with weights Θ_A and Θ_B, respectively – such that we can linearly interpolate between them. We assume that models A and B were trained with equivalent architectures but different random initializations, data orders, and potentially different hyperparameters or datasets as well. Our central question is: Given Θ_A and Θ_B, can we identify some π such that when linearly interpolating between Θ_A and $\pi(\Theta_B)$, all intermediate models enjoy performance similar to Θ_A and Θ_B?

We base any claims of loss landscape (quasi-)convexity on the usual definition of multi-dimensional convexity in terms of one-dimensional (quasi-)convexity,

Definition 4.2.1 (Convexity)**.** A function $f : \mathbb{R}^d \to \mathbb{R}$ is (quasi-)convex if every one-dimensional slice is (quasi-)convex, ie. for all $x, y \in \mathbb{R}^d$, the function $g(\lambda) = f((1-\lambda)x + \lambda y)$ is (quasi-)convex in λ.

Thanks to Definition 4.2.1, it suffices to show that arbitrary one-dimensional slices of a function are convex in order to reason about the convexity of complex, high-dimensional functions. In practice, we rarely observe perfect convexity, but instead hope to approximate it as closely as

possible. Following Draxler et al. [2018], Entezari et al. [2021], Frankle et al. [2020] and others, we measure approximations to convexity via "barriers".

Definition 4.2.2 (Loss barrier [Frankle et al., 2020])**.** Given two points Θ_A, Θ_B such that $\mathcal{L}(\Theta_A) \approx \mathcal{L}(\Theta_B)$, the *loss barrier* is defined as $\max_{\lambda \in [0,1]} \mathcal{L}((1-\lambda)\Theta_A + \lambda\Theta_B) - \frac{1}{2}(\mathcal{L}(\Theta_A) + \mathcal{L}(\Theta_B))$.

Note that loss barriers are non-negative, with zero indicating an interpolation of flat or negative curvature.

4.3 Permutation selection methods

We introduce three methods of matching units between model A and model B. In addition, we present an appealing but failed method in Appendix C.2.

4.3.1 Matching activations

Following the classic Hebbian mantra "[neural network units] that fire together, wire together" [Hebb, 2005] we propose associating units across two models by performing regression between their activations. Provided activations for each model, we wish to associate each unit in A to a unit in B. It stands to reason that there may exist a linear relationship between the activations of the two models. In particular, we are interested in a bijection mapping each unit in A to a *unique* unit in B and vice versa. We fit this into the regression framework by constraining ordinary least squares (OLS) regression to solutions in the set of permutation matrices, S_d. For activations of the ℓ'th layer, let $\boldsymbol{Z}^{(A)}, \boldsymbol{Z}^{(B)} \in \mathbb{R}^{d \times n}$ denote the d-dimensional activations for all n training data points, in models A and B respectively. Then,

$$\boldsymbol{P}_\ell = \underset{\boldsymbol{P} \in S_d}{\arg\min} \sum_{i=1}^{n} ||\boldsymbol{Z}^{(A)}_{:,i} - \boldsymbol{P}\boldsymbol{Z}^{(B)}_{:,i}||^2 \tag{4.1}$$

$$= \underset{\boldsymbol{P} \in S_d}{\arg\min} \; ||\boldsymbol{Z}^{(A)} - \boldsymbol{P}\boldsymbol{Z}^{(B)}||_F^2 \tag{4.2}$$

$$= \underset{\boldsymbol{P} \in S_d}{\arg\max} \; \langle \boldsymbol{P}, \boldsymbol{Z}^{(A)}(\boldsymbol{Z}^{(B)})^\top \rangle_F \tag{4.3}$$

where $\langle \boldsymbol{A}, \boldsymbol{B} \rangle_F = \sum_{i,j} A_{i,j} B_{i,j}$ denotes the Frobenius inner product between real-valued matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ and $||\boldsymbol{A}||_F^2 = \langle \boldsymbol{A}, \boldsymbol{A} \rangle_F$ is the corresponding norm. Conveniently, Eq. (4.3) constitutes a "linear assignment problem" problem (LAP) [Bertsekas, 1998] for which efficient, practical algorithms are known. Having solved this assignment problem on each layer, we can then permute the weights of model B to match model A as closely as possible:

$$\boldsymbol{W}'_\ell = \boldsymbol{P}_\ell \boldsymbol{W}_\ell^{(B)} \boldsymbol{P}_{\ell-1}^\top, \quad \boldsymbol{b}'_\ell = \boldsymbol{P}_\ell \boldsymbol{b}_\ell^{(B)}$$

for each layer ℓ, producing new weights, Θ', such that the intermediate activations align as closely possible with those of Θ_A.

Matching activations between models is compelling as it captures the intuitive notion that two models must learn similar features in order to accomplish the same task [Li et al., 2016]. Furthermore, this entire process is relatively computationally lightweight: the $\boldsymbol{Z}^{(A)}$ and $\boldsymbol{Z}^{(B)}$ matrices can be computed in a single pass over the training dataset, and in practice, a full run through the training dataset may not even be necessary. Decent correlation estimates are likely possible with only a subset of the data. Solving Eq. (4.3) is possible thanks to well-established, polynomial-time algorithms for solving the linear assignment problem [Kuhn, 2010, Jonker and Volgenant, 1987, Crouse, 2016]. Also, conveniently, the activation matching at each layer is independent of the matching at every other layer resulting in a separable and straightforward optimization problem, an advantage that will not be enjoyed by our other two methods.

Dispensing with regression, one could similarly associate units by matching against a matrix of cross-correlation coefficients in place of $\boldsymbol{Z}^{(A)}(\boldsymbol{Z}^{(B)})^\top$. We observed correlation matching to work equally well, but found matching via OLS regression to be more principled and easier to implement.

4.3.2 Matching weights

Instead of associating intermediate units by their activations under some data distribution, we could alternatively inspect the weights of the model itself. Consider for a moment, the first layer weights, $\boldsymbol{W}_1$. Each row of $\boldsymbol{W}_1$ corresponds to a single feature of the model. If two such rows were

equal, they would compute exactly the same feature (ignoring bias terms for the time being). And if $[\boldsymbol{W}_1^{(A)}]_{i,:} \approx [\boldsymbol{W}_1^{(B)}]_{j,:}$, it stands to reason that units i and j should be associated. Extending this idea to every layer, we are inspired to pursue the optimization,

$$\arg\min_{\pi} \; ||\text{vec}(\Theta_A) - \text{vec}(\pi(\Theta_B))||^2 \; = \; \arg\max_{\pi} \; \text{vec}(\Theta_A) \cdot \text{vec}(\pi(\Theta_B))$$

We can re-express this in terms of the full weights,

$$\arg\max_{\pi=\{\boldsymbol{P}_i\}} \langle \boldsymbol{W}_1^{(A)}, \boldsymbol{P}_1 \boldsymbol{W}_1^{(B)} \rangle_F + \langle \boldsymbol{W}_2^{(A)}, \boldsymbol{P}_2 \boldsymbol{W}_2^{(B)} \boldsymbol{P}_1^\top \rangle_F + \cdots + \langle \boldsymbol{W}_L^{(A)}, \boldsymbol{W}_L^{(B)} \boldsymbol{P}_{L-1}^\top \rangle_F \tag{4.4}$$

resulting in another matching problem. We term this formulation the "bilinear assignment problem" (BLAP). Sadly, this matching problem is larger and thornier than the classic linear assignment matching problem, presented in Eq. (4.3). Unlike the linear assignment problem of Eq. (4.3), we are interested in permuting *both* the rows and columns of $\boldsymbol{W}_\ell^{(B)}$ to match $\boldsymbol{W}_\ell^{(A)}$, and this turns out to be crucially different from only permuting rows or only permuting columns. We formalize this difficulty as follows,

Lemma 1. *The bilinear assignment problem is NP-hard for $L > 2$.*

which we prove in Appendix C.3. Lemma 1 stands in stark contrast to classical LAP for which practical, polynomial-time algorithms are known.

Undeterred, we propose a greedy approximation algorithm for BLAP anyhow. Looking at a single $\boldsymbol{P}_\ell$ while holding the others fixed, note that the optimization problem can be reduced to a classic linear assignment problem,

$$\begin{aligned}
&\arg\max_{\boldsymbol{P}_\ell} \langle \boldsymbol{W}_\ell^{(A)}, \boldsymbol{P}_\ell \boldsymbol{W}_\ell^{(B)} \boldsymbol{P}_{\ell-1}^\top \rangle_F + \langle \boldsymbol{W}_{\ell+1}^{(A)}, \boldsymbol{P}_{\ell+1} \boldsymbol{W}_{\ell+1}^{(B)} \boldsymbol{P}_\ell{}^\top \rangle_F \\
&\qquad = \arg\max_{\boldsymbol{P}_\ell} \langle \boldsymbol{P}_\ell, \boldsymbol{W}_\ell^{(A)} \boldsymbol{P}_{\ell-1} (\boldsymbol{W}_\ell^{(B)})^\top \rangle_F + \langle \boldsymbol{P}_\ell, (\boldsymbol{W}_{\ell+1}^{(A)})^\top \boldsymbol{P}_{\ell+1} \boldsymbol{W}_{\ell+1}^{(B)} \rangle_F \\
&\qquad = \arg\max_{\boldsymbol{P}_\ell} \langle \boldsymbol{P}_\ell, \boldsymbol{W}_\ell^{(A)} \boldsymbol{P}_{\ell-1} (\boldsymbol{W}_\ell^{(B)})^\top + (\boldsymbol{W}_{\ell+1}^{(A)})^\top \boldsymbol{P}_{\ell+1} \boldsymbol{W}_{\ell+1}^{(B)} \rangle_F
\end{aligned}$$

This leads us to a convenient greedy algorithm: go through each of the layers, and greedily select the best $\boldsymbol{P}_\ell$ at that layer. Repeat until convergence. We present this in Algorithm 3.

Algorithm 3 Greedy-BLAP

Given: Model weights $\Theta_A = \left\{\boldsymbol{W}_1^{(A)}, \ldots, \boldsymbol{W}_L^{(A)}\right\}$ and $\Theta_B = \left\{\boldsymbol{W}_1^{(B)}, \ldots, \boldsymbol{W}_L^{(B)}\right\}$

Result: A permutation $\pi = \{\boldsymbol{P}_1, \ldots, \boldsymbol{P}_{L-1}\}$ of Θ_B such that $\text{vec}(\Theta_A) \cdot \text{vec}(\pi(\Theta_B))$ is approximately maximized.

$\boldsymbol{P}_1 \leftarrow \boldsymbol{I}, \ldots, \boldsymbol{P}_{L-1} \leftarrow \boldsymbol{I}$

repeat

  for $\ell \leftarrow \text{RandomPermutation}(1, \ldots, L-1)$ **do**

    $\boldsymbol{P}_\ell \leftarrow \text{SolveLAP}\left(\boldsymbol{W}_\ell^{(A)} \boldsymbol{P}_{\ell-1} (\boldsymbol{W}_\ell^{(B)})^\top + (\boldsymbol{W}_{\ell+1}^{(A)})^\top \boldsymbol{P}_{\ell+1} \boldsymbol{W}_{\ell+1}^{(B)}\right)$

  end for

until convergence

Although we present Algorithm 3 in terms of an MLP without bias terms, in practice our implementation can handle the weights of models of arbitrary architecture, including bias terms, residual connections, convolutional layers, and so forth. In experiments we observed this algorithm to be fast in terms of both wall-clock time and iterations necessary for convergence. We did not observe any examples of it failing to converge. We cowardly defer a formal analysis of Algorithm 3 to future work.

Unlike the activation matching method presented in Section 4.3.1, weight matching is "blind" in the sense that it ignores the data distribution. In theory, ignoring the input data distribution and therefore the loss landscape handicaps weight matching, but allows it to be much faster, encouraging its potential application in fields such as fine-tuning [Devlin et al., 2019, Wortsman et al., 2022b,a], federated learning [McMahan et al., 2017, Konečný et al., 2016a,b], and model patching [Matena and Raffel, 2021, Sung et al., 2021, Raffel, 2021]. In practice we found weight matching to be surprisingly competitive with data-aware methods. We present a detailed comparison of this trade-off and the models' results in Section 4.5.

4.3.3 *Learning permutations with a straight-through estimator*

Inspired by the success of straight-through estimators (STEs) in other discrete optimization problems [Bengio et al., 2013, Kusupati et al., 2021, Rastegari et al., 2016, Courbariaux and Bengio, 2016], in this section we attempt to "learn" the ideal permutation of weights $\pi(\Theta_B)$. More specifically, our goal is to optimize

$$\min_{\tilde{\Theta}_B} \mathcal{L}\left(\frac{1}{2}\left(\Theta_A + \text{proj}\left(\tilde{\Theta}_B\right)\right)\right), \qquad \text{proj}(\Theta) \triangleq \underset{\pi(\Theta_B)}{\arg\max}\ \text{vec}(\Theta) \cdot \text{vec}(\pi(\Theta_B)) \tag{4.5}$$

where $\tilde{\Theta}_B$ denotes an approximation of $\pi(\Theta_B)$, allowing us to implicitly optimize π. Tragically Eq. (4.5) involves non-differentiable projection operations, proj, complicating the optimization. We overcome this via a "straight-through" estimator: we parameterize the problem in terms of a set of weights $\tilde{\Theta}_B \approx \pi(\Theta_B)$. In the forward pass, we project $\tilde{\Theta}_B$ to the closest realizable $\pi(\Theta_B)$. Then, in the backwards pass we switch back to the unrestricted weights $\tilde{\Theta}_B$. In this way, we are guaranteed to stay true to the projection constraints in evaluating the loss, but are still able to get usable gradients at our current parameters, $\tilde{\Theta}_B$.[1]

Conveniently, we can re-purpose Algorithm 3 to solve proj $\left(\tilde{\Theta}_B\right)$. Furthermore, we found initializing $\tilde{\Theta}_B = \Theta_A$ performed better than random initialization. This is to be expected immediately at initialization since the initial matching will be equivalent to the weight matching method of Section 4.3.1, but it is not immediately clear why these solutions continue to outperform a random initialization asymptotically.

Unlike the aforementioned methods, Algorithm 4 makes an attempt to explicitly "learn" the best permutation π using a conventional training loop. By initializing to the weight matching solution of Section 4.3.2 and leveraging the data distribution as in Section 4.3.1, it seeks to offer a best-of-both-worlds solution. Sadly, this comes at a very steep computational cost relative to the other two methods.

---

[1] Note again that projecting according to inner product distance is equivalent to L2 distance when parameterizing the estimator based on the B endpoint. We also experimented with learning the midpoint directly, $\tilde{\Theta} \approx \frac{1}{2}(\Theta_A + \pi(\Theta_B))$, in which case the L2 and inner product projections diverge. In testing all possible variations, we found that optimizing the B endpoint had a slight advantage, but all possible variations performed admirably.

Algorithm 4 Straight-through estimator training

Given: Model weights Θ_A, Θ_B, and a learning rate η.

Result: A permutation π of Θ_B such that $\mathcal{L}(\frac{1}{2}(\Theta_A + \pi(\Theta_B)))$ is approximately minimized.

Initialize: $\tilde{\Theta}_B \leftarrow \Theta_A$

repeat

$\pi(\Theta_B) \leftarrow \text{proj}(\tilde{\Theta}_B)$ using Algorithm 3.

Evaluate the loss of the midpoint, $\mathcal{L}(\frac{1}{2}(\Theta_A + \pi(\Theta_B)))$.

Evaluate the gradient, $\nabla\mathcal{L}$, using $\tilde{\Theta}_B$ in place of $\pi(\Theta_B)$ in the backwards pass.

Update parameters, $\tilde{\Theta}_B \leftarrow \tilde{\Theta}_B - \eta\nabla\mathcal{L}$.

until convergence

4.4 Interlude: a counterexample

Before arguing for the presence of linear mode connectivity, we take a step back to consider its limitations. In this section we present a counterexample proving that there exist models that do not enjoy linear mode connectivity under any permutation of weights. It follows that – to the extent we are able to achieve linear mode connectivity in practice – it is an artifact of our optimization methods, not our architectures.

Consider a simple 2-dimensional classification task: Our data points are drawn $\boldsymbol{x} \sim \text{Uniform}([-1, 1]^2)$ and $y = \mathbf{1}_{x_1>0 \text{ and } x_2>0}$. In other words, points in the positive quadrant are labeled $y = 1$, while everything else is labeled $y = 0$. We provide a visualization of a sample of such data in Figure C.1.

We utilize an MLP architecture consisting of two hidden layers, with two units each, and ReLU nonlinearities. Consider two weight assignments that both achieve a perfect fit to the data:

$$f_A(\boldsymbol{x}) = \begin{bmatrix} -1 & -1 \end{bmatrix} \sigma\left(\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \sigma\left(\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right) + \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right) \tag{4.6}$$

$$f_B(\boldsymbol{x}) = \begin{bmatrix} -1 & -1 \end{bmatrix} \sigma\left(\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \sigma\left(\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}\right) + \begin{bmatrix} 0 \\ 1 \end{bmatrix}\right) \tag{4.7}$$

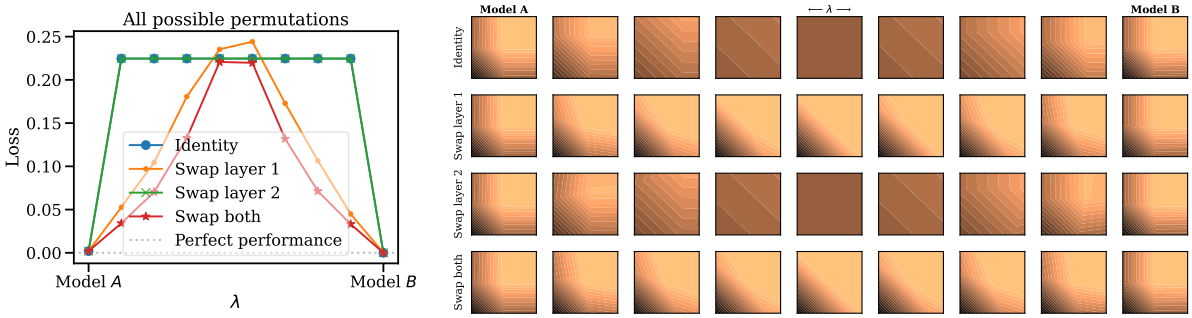

Figure 4.1: A counterexample: There exist models such that no possible permutation of weights allows for linear mode connectivity. *Left:* performance of all possible linear interpolations between the two models. *Right:* A visualization of the prediction functions $f(\boldsymbol{x})$ through each of the linear sweeps. Each row corresponds to one of the four possible permutations and each column corresponds to a value of λ, the linear interpolant. The existence of such cases suggests that linear mode connectivity is an artifact of SGD.

We predict a positive label when $f(\boldsymbol{x}) \geq 0$ and a negative label otherwise. Intuitively, these networks are organized such that each layer makes a classification whether $\boldsymbol{x}_1 > 0$ or $\boldsymbol{x}_2 > 0$. In model A, the first layer tests whether $\boldsymbol{x}_2 > 0$ and the second layer tests whether $\boldsymbol{x}_1 > 0$, whereas in model B the order is reversed. With a bit of algebra, it is possible to see that both f_A and f_B achieve perfect performance.

Since these networks have small width, we can simply inspect all four possible permutations of the intermediate units. We visualize this in Figure 4.1. Critically, no permutation results in linear mode connectivity.

We present this counterexample in part to establish some basic intuition for Conjecture 1, but more importantly to highlight that any success interpolating between permuted networks is due to inherent bias in the optimization algorithms used and not the network architectures themselves. We have SGD to thank any time we find two independent models that admit some form of linear mode connectivity. We also take this opportunity to point out that there are invariances beyond

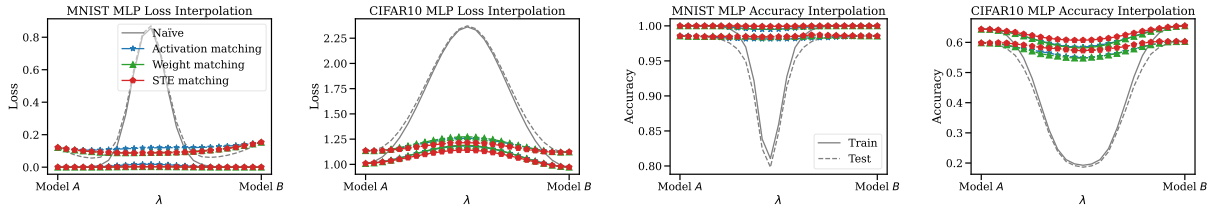

Figure 4.2: **Linear mode connectivity is possible after permuting.** Loss and accuracy interpolating between MLP models trained on MNIST and CIFAR-10. In all cases we are able to significantly improve over naïve interpolation. Straight-through estimator matching performs best but is very computationally expensive. Weight matching and activation matching perform similarly, even though weight matching is orders of magnitude faster and does not rely on the input data distribution.

permutation symmetries: It is also possible to move features between layers, re-scale layers, and so forth. Prior works have previously pointed out the feature/layer-association [Nguyen et al., 2021] and layer re-scaling invariances [Ainsworth et al., 2018b], albeit in different contexts.

4.5 Experiments

Our base experimental methodology will be to separately train two models, A and B, starting from different random initializations and with different random batch orders, resulting in trained weights Θ_A and Θ_B respectively. We then evaluate slices through the loss landscape, $\mathcal{L}((1-\lambda)\Theta_A + \lambda\pi(\Theta_B))$ for $\lambda \in [0,1]$, where π is selected according to the methods presented in Section 4.3.[2] Ideally, we seek a completely flat or even convex one-dimensional slice. As discussed in Section 4.2, the ability to exhibit this behavior for arbitrary Θ_A, Θ_B empirically suggests that the loss landscape contains only a single basin, modulo permutation symmetries.

[2]We also experimented with spherical linear interpolation ("slerp") and found that it performed slightly better than linear interpolation but that the change was not interesting or significant enough to warrant diverging from the pre-existing literature.

We remark that failure to find a π such that linear mode connectivity holds cannot rule out the existence of a satisfactory permutation. Given the astronomical number of permutation symmetries, Conjecture 1 is essentially impossible to disprove for any realistically wide model architecture.

4.5.1 Loss landscapes before and after matching

We present results for MLPs trained on MNIST [LeCun et al., 1998] and CIFAR-10 [Krizhevsky, 2009] in Figure 4.2. Naïve interpolation ($\pi(\Theta) = \Theta$)) results in substantial performance degradation. On the other hand, the methods introduced in Section 4.3 are able to achieve much better barriers. We achieve zero-barrier linear mode connectivity on MNIST with all three methods, although activation matching performs just slightly less favorably than weight matching and straight-through estimator (STE) matching. We especially note that the test loss landscape is made convex after applying our weight matching and STE permutations! In other words, our interpolation actually yields a merged model that outperforms both model A and B. We expand on this phenomenon in Section 4.5.4.

On CIFAR-10 we fall short of zero-barrier connections between MLP solutions, although we do see a ten-fold decrease in barrier relative to naïve interpolation. On CIFAR-10 the benefits of STE relative to weight matching become more clear. We see that STE matching identifies a noticeably better permutation than the other two methods. CIFAR-10 is not out-of-reach, however. As we demonstrate in Section 4.5.3, we are able to achieve zero-barrier LMC on CIFAR-10 with large ResNet models. Therefore, we hypothesize that the presence of linear mode connectivity is dependent on the capacity of the model being sufficient to capture the input data distribution.

STE matching, the most expensive method, produces the best solutions as expected. Somewhat surprising however is that the gap between STE and the other two methods is not all that large. In particular, it is remarkable how much can be done without looking at the input data at all! We found that weight matching offered a very compelling balance between computational cost and performance: it runs in mere seconds (on modern hardware at the time of writing), and produces high quality solutions. On the other hand, activation matching appears to be of little utility as it is

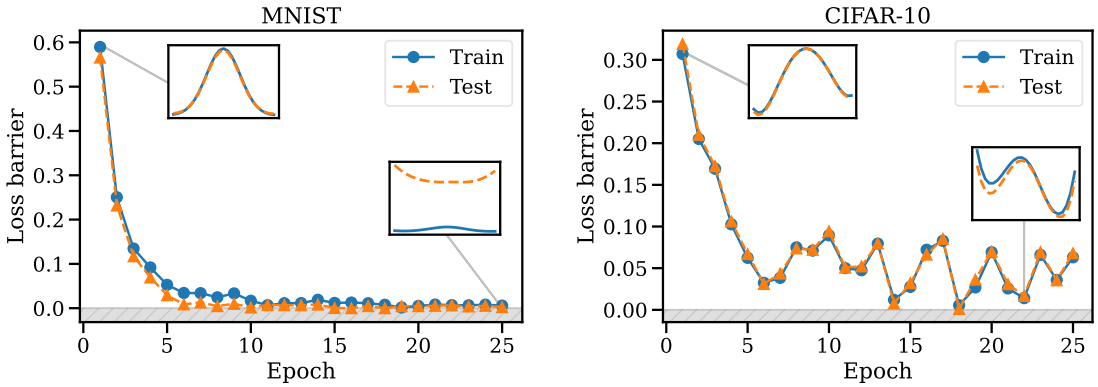

Figure 4.3: **Linear mode connectivity does not work at initialization.** Loss barriers as a function of training time for MLPs trained on MNIST (left) and CIFAR-10 (right). Loss interpolation plots inlaid to highlight results in initial and later epochs. LMC manifests gradually as models are trained. The single basin theory does not hold for models at initialization, suggesting distinct loss landscape geometries at the beginning and end phases of training. (The y-axis scales differ in each of the inlaid plots.)

more expensive to run and did not produce higher quality solutions in our experiments.

4.5.2 Onset of mode connectivity

Given the results of Section 4.5.1 it may be tempting to conclude that the entirety of weight space contains only a single basin modulo permutation symmetries. However we found that linear mode connectivity is an emergent property of SGD training, and we were unable to uncover it early in training. We explore the emergence of linear mode connectivity in Figure 4.3. The emergence of LMC partway through training fits with previous works suggesting that training consists of an initial "burn-in" phase followed by a much longer "tuning" phase [Frankle et al., 2020].

Note also that the final inlaid interpolation plot in Figure 4.3(right) demonstrates an important shortcoming of the loss barrier metric: the interpolation includes points with lower loss than either

of the two models, however the loss barrier is still positive due to non-negativity as mentioned in Section 4.2.

4.5.3 Effect of model width

Conventional wisdom suggests that wider architectures are easier to optimize [Jacot et al., 2018, Lee et al., 2019]. In this section, we investigate whether wider architectures are also easier to linearly mode connect. We train VGG-16 [Simonyan and Zisserman, 2015] and ResNet20 [He et al., 2016] architectures of varying width on the CIFAR-10 dataset. Results are presented in Figure 4.4. Sadly, $8\times$ width VGG-16 training was unattainable as it exhausted GPU memory on available hardware at the time of writing.

As expected, there is a clear relationship between model width and linear mode connectivity, as measured by the loss barrier between solutions. Although $1\times$-sized models did not seem to exhibit linear mode connectivity, we found that larger width models decreased loss barriers all the way to zero. In Figure 4.4(right) we show the first ever demonstration – to the best of the authors' knowledge – of zero-barrier linear mode connectivity between two large ResNet models trained on a non-trivial dataset.

It is essential to point out that relatively skinny models do not seem to obey linear mode connectivity, yet still exhibit similarities in training dynamics. This suggests that either our permutation selection methods are failing to find satisfactory permutations on skinnier models or that some form of invariance other than LMC must be at play in the skinny model regime.

4.5.4 Model patching and split data training

Inspired by work on fine-tuning [Wortsman et al., 2022a], model patching [Raffel, 2021], and federated learning [McMahan et al., 2017, Konečný et al., 2016a,b], we study whether it is possible to combine the weights of two models trained on disjoint datasets in a positive-sum way. Consider for example an organization with multiple (possibly biased!) datasets separated for regulatory (eg. GDPR) or privacy (eg. on-device data) considerations. Models can be trained on each dataset

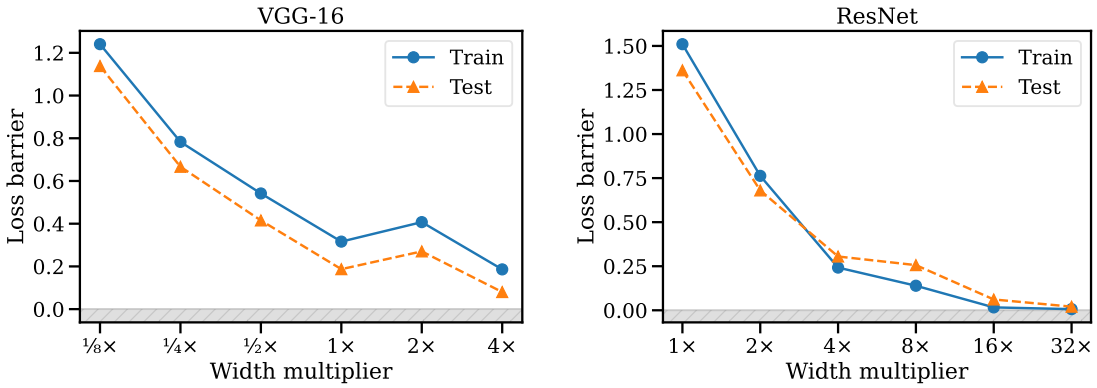

Figure 4.4: **Wider models exhibit better linear mode connectivity.** Training convolutional and ResNet architectures on CIFAR-10, we ablate their width and visualize their loss barriers after weight matching. Notably, we achieve zero-barrier LMC between ResNet models, the first such demonstration to the best of the authors' knowledge.

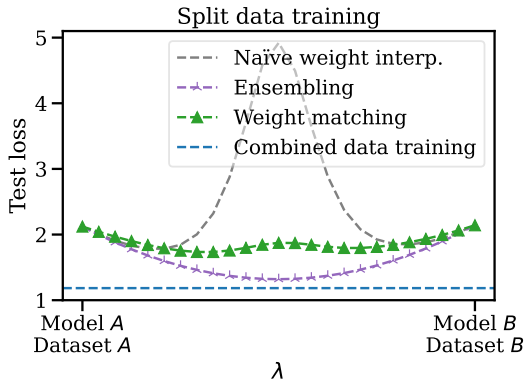

Figure 4.5: **Models trained on disjoint datasets can be merged with positive-sum results.** Two ResNet models trained on disjoint, biased subsets of CIFAR-100 can be merged in weights space, thanks to weight matching.

individually but training in aggregate is not feasible. Can we combine separately trained models in such a way that the merged model outperforms input models?

To address this question we split the CIFAR-100 dataset [Krizhevsky, 2009] into two disjoint subsets: dataset A, containing 20% examples labelled 0-49 and 80% labelled 50-99, and dataset B vice-versa. No image exists in both dataset A and B. ResNet20 models A and B were trained on their corresponding datasets. We show the result of merging the two models with weight matching in Figure 4.5. Naïve weight interpolation, ensembling of the model logits, and full-data training are also benchmarked for comparison.

As expected, merging separately-trained models with weight matching was not able to match the performance of omnipotent model trained on the full dataset, or an ensemble of the two models with twice the number of effective weights. On the other hand, we did manage to merge the two models in weight space achieving an interpolated model that outperforms both input models in terms of test log-likelihood. It also vastly outperformed naïve interpolation, the status quo for model combination in federated learning. Incorporating Algorithm 3 into a federated learning framework could be an exciting avenue for future work.

4.6 Related work

(Linear) mode connectivity Garipov et al. [2018] introduced the concept of mode connectivity: the idea that SGD solutions in the loss landscape are connected by constant-loss curves in weight space. Further explorations were undertaken in Freeman and Bruna [2017], Draxler et al. [2018] among others. Frankle et al. [2020] demonstrated a connection between *linear* mode connectivity and the lottery ticket hypothesis. Brea et al. [2019] noted the existence of permutation symmetries and implicated them as a source of saddle points in the loss landscape. Concurrent to our work, Entezari et al. [2021] conjectured that SGD solutions could be linear mode connected modulo permutation symmetries.

Loss landscapes and training dynamics Li et al. [2016], Yosinski et al. [2014] investigated whether independently trained networks learn similar features, and to what extent they transfer. Jiang et al. [2021] argued that independently trained networks meaningfully differ in the features

they learn in certain scenarios. Zhang et al. [2019] studied the relative importance of layers and the features they learn. Li et al. [2018a] showed that residual connections result in smoother loss landscapes. On the theoretical front, Kawaguchi [2016] proved that deep linear networks contain no local minima. In other words, every minima is a global minima. Boursier et al. [2022] characterized the training dynamics of one-hidden layer ReLU networks on orthogonal data, including a proof that they converge to zero loss.

Federated learning and model merging McMahan et al. [2017], Konečný et al. [2016a,b] introduced the concept of "federated learning": learning split across across multiple devices and data. Wang et al. [2020] proposed an exciting federated learning method in which model averaging is done after permuting units. Unlike this work, however, they proposed merging smaller "child" models into a larger "main" model, and doing so with a more specific, layer-wise matching algorithm that does not support residual connections or normalization layers. Raffel [2021], Matena and Raffel [2021], Sung et al. [2021] introduced the study of "model patching", the idea that models should be easy to modify and submit changes to. Singh and Jaggi [2020] proposed merging models soft-aligning associations weights, inspired by optimal transport. Wortsman et al. [2022a] demonstrated state-of-the-art ImageNet performance by averaging weights of models fine-tuned starting from some initial trained state.

Differentiating through permutations Akin to differentiable permutation learning, a number of prior works have studied differentiable sorting [Grover et al., 2019, Prillo and Eisenschlos, 2020, Cuturi et al., 2019, Petersen et al., 2022, 2021, Mena et al., 2018]. Blondel et al. [2020] studied differentiable sorting and ranking with asymptotics that correspond to their non-differentiable versions. Fogel et al. [2015] explored recovering the linear orderings of items based on pairwise information, another form of permutation optimization. Bengio et al. [2013] introduced the straight-through estimator for differentiating through discrete projections that we utilize in Section 4.3.3.

4.7 Discussion

We explore the role of permutation symmetries in the linearly mode connectivity of SGD solutions. In pursuit of this, we introduce three novel algorithms, of varying complexity and computational cost, to canonicalize independent neural network weights such that the loss landscape between them is made as flat as possible. Despite presenting successes across multiple architectures and datasets, linear mode connectivity between thin models seems unlikely even though thin models also exhibit phenomenon suggesting some underlying invariances in their training dynamics. Therefore we conclude that the permutation symmetry hypothesis is a necessary piece, though not a complete picture, of whatever fundamental invariances are at play in real-world neural network training dynamics. In particular, we hypothesize that SGD is biased towards solutions such that there is a linear relationship between layer-wise activations across models. In the infinite width limit it just so happens that there exist satisfactory linear relationships that are also a permutations.

An expanded theory and empirical exploration of other invariances – such as cross-layer scaling or general linear relationships between activations – would be an exciting avenue for future work. Ultimately, it is our hope that a lucid understanding of loss landscape geometry will not only advance the theory of deep learning but will also unlock the development of better optimization, federated learning, and ensembling techniques.

BIBLIOGRAPHY

Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. *Proceedings of the 21st International Conference on Machine Learning*, 2004.

Pieter Abbeel and Andrew Y. Ng. Exploration and apprenticeship learning in reinforcement learning. In *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*, volume 119 of *ACM International Conference Proceeding Series*, pages 1–8. ACM, 2006. doi: 10.1145/1102351.1102352.

Samuel K. Ainsworth, Nicholas J. Foti, and Emily B. Fox. Disentangled VAE Representations for Multi-Aspect and Missing Data. *CoRR*, abs/1806.09060, 2018a.

Samuel K. Ainsworth, Nicholas J. Foti, Adrian K. C. Lee, and Emily B. Fox. oi-vae: Output interpretable vaes for nonlinear group factor analysis. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 119–128. PMLR, 2018b. URL `http://proceedings.mlr.press/v80/ainsworth18a.html`.

Samuel K. Ainsworth, Matt Barnes, and Siddhartha S. Srinivasa. Mo' States Mo' Problems: Emergency Stop Mechanisms from Observation. In *Conference on Neural Information Processing Systems, NeurIPS*, 2019.

Samuel K. Ainsworth, Kendall Lowrey, John Thickstun, Zaïd Harchaoui, and Siddhartha S. Srinivasa. Faster policy learning with continuous-time gradients. In Ali Jadbabaie, John Lygeros, George J. Pappas, Pablo A. Parrilo, Benjamin Recht, Claire J. Tomlin, and Melanie N. Zeilinger, editors, *Proceedings of the 3rd Annual Conference on Learning for Dynamics and Control, L4DC*

2021, 7-8 June 2021, Virtual Event, Switzerland, volume 144 of *Proceedings of Machine Learning Research*, pages 1054–1067. PMLR, 2021. URL http://proceedings.mlr.press/v144/ainsworth21a.html.

Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Marcin Andrychowicz, Anton Raichuk, Piotr Stanczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What matters in on-policy reinforcement learning? A large-scale empirical study. *CoRR*, abs/2006.05990, 2020.

Kendall Atkinson. *An Introduction to Numerical Analysis, 2nd Edition*. Wiley, 1989.

Jean Yves Audibert, Rémi Munos, and Csaba Szepesvári. Exploration-exploitation tradeoff using variance estimates in multi-armed bandits. *Theoretical Computer Science*, 410(19):1876–1902, 2009. ISSN 03043975. doi: 10.1016/j.tcs.2009.01.016.

Mohammad Gheshlaghi Azar, Ian Osband, and Rémi Munos. Minimax regret bounds for reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

J Andrew Bagnell. An invitation to imitation. Technical report, Carnegie Mellon University, 2015.

Akshay Balsubramani and Aaditya Ramdas. Sequential nonparametric testing with the law of the iterated logarithm. *arXiv preprint arXiv:1506.03486*, 2015.

Peter W. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4502–4510, 2016.

Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013. URL http://arxiv.org/abs/1308.3432.

Jean Bertoin. Random covering of an interval and a variation of Kingman's coalescent. *Random Structures & Algorithms*, 25(3):277–292, 2004. ISSN 1042-9832.

D.P. Bertsekas. *Network Optimization: Continuous and Discrete Methods*. Athena scientific optimization and computation series. Athena Scientific, 1998. ISBN 9788865290279. URL https://books.google.com/books?id=afYYAQAAIAAJ.

Christopher M. Bishop. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007. ISBN 9780387310732. URL https://www.worldcat.org/oclc/71008143.

Mathieu Blondel, Olivier Teboul, Quentin Berthet, and Josip Djolonga. Fast differentiable sorting and ranking. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 950–959. PMLR, 2020. URL http://proceedings.mlr.press/v119/blondel20a.html.

Etienne Boursier, Loucas Pillaud-Vivien, and Nicolas Flammarion. Gradient flow dynamics of shallow relu networks for square loss and orthogonal inputs. *CoRR*, abs/2206.00939, 2022. doi: 10.48550/arXiv.2206.00939. URL https://doi.org/10.48550/arXiv.2206.00939.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Johanni Brea, Berfin Simsek, Bernd Illing, and Wulfram Gerstner. Weight-space symmetry in deep networks gives rise to permutation saddles, connected by equal-loss valleys across the loss landscape. *CoRR*, abs/1907.02911, 2019. URL http://arxiv.org/abs/1907.02911.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016.

John C. Butcher. *Numerical methods for ordinary differential equations*. Wiley, 2003.

Jan Carius, René Ranftl, Vladlen Koltun, and Marco Hutter. Trajectory optimization with implicit hard contacts. *IEEE Robotics Autom. Lett.*, 3(4):3316–3323, 2018.

Jan Carius, René Ranftl, Vladlen Koltun, and Marco Hutter. Trajectory optimization for legged robots with slipping motions. *IEEE Robotics Autom. Lett.*, 4(3):3013–3020, 2019.

E. Cela. *The Quadratic Assignment Problem: Theory and Algorithms*. Combinatorial Optimization. Springer US, 2013. ISBN 9781475727876. URL https://books.google.com/books?id=20QGCAAAQBAJ.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 6572–6583, 2018.

Matthieu Courbariaux and Yoshua Bengio. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830, 2016. URL http://arxiv.org/abs/1602.02830.

David Frederic Crouse. On implementing 2d rectangular assignment algorithms. *IEEE Trans. Aerosp. Electron. Syst.*, 52(4):1679–1696, 2016. doi: 10.1109/TAES.2016.140952. URL https://doi.org/10.1109/TAES.2016.140952.

Marco Cuturi, Olivier Teboul, and Jean-Philippe Vert. Differentiable ranks and sorting using optimal transport. *CoRR*, abs/1905.11885, 2019. URL http://arxiv.org/abs/1905.11885.

Filipe de Avila Belbute-Peres, Kevin A. Smith, Kelsey R. Allen, Josh Tenenbaum, and J. Zico Kolter. End-to-end differentiable physics for learning and control. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 7178–7189, 2018.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019. doi: 10.18653/v1/n19-1423. URL https://doi.org/10.18653/v1/n19-1423.

Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred A. Hamprecht. Essentially no barriers in neural network energy landscape. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1308–1317. PMLR, 2018. URL http://proceedings.mlr.press/v80/draxler18a.html.

Jianzhun Du, Joseph Futoma, and Finale Doshi-Velez. Model-based reinforcement learning for semi-markov decision processes with neural odes. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Rahim Entezari, Hanie Sedghi, Olga Saukh, and Behnam Neyshabur. The role of permutation invariance in linear mode connectivity of neural networks. *CoRR*, abs/2110.06296, 2021. URL https://arxiv.org/abs/2110.06296.

Leonhard Euler. *Institutionum calculi integralis*. 1768.

Benjamin Eysenbach, Shixiang Gu, Julian Ibarz, and Sergey Levine. Leave no trace: learning to reset for safe and autonomous reinforcement learning, 2018.

Jiameng Fan and Wenchao Li. Safety-Guided Deep Reinforcement Learning via Online Gaussian Process Estimation. *arXiv preprint arXiv:1903.02526*, 2019.

Taylor Kessler Faulkner, Elaine Schaertl Short, and Andrea Lockerd Thomaz. Policy shaping with supervisory attention driven exploration. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain, October 1-5, 2018*, pages 842–847. IEEE, 2018. doi: 10.1109/IROS.2018.8594312.

Leopold Flatto and Alan G Konheim. The random division of an interval and the random covering of a circle. *Siam Review*, 4(3):211–222, 1962. ISSN 0036-1445.

Fajwel Fogel, Rodolphe Jenatton, Francis R. Bach, and Alexandre d'Aspremont. Convex relaxations for permutation problems. *SIAM J. Matrix Anal. Appl.*, 36(4):1465–1488, 2015. doi: 10.1137/130947362. URL https://doi.org/10.1137/130947362.

Jonathan Frankle. Revisiting "qualitatively characterizing neural network optimization problems". *CoRR*, abs/2012.06898, 2020. URL https://arxiv.org/abs/2012.06898.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 3259–3269. PMLR, 2020. URL http://proceedings.mlr.press/v119/frankle20a.html.

C. Daniel Freeman and Joan Bruna. Topology and geometry of half-rectified network optimization. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=Bk0FWVcgx.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. *arXiv preprint arXiv:1812.02900*, 2018.

Javier Garcia and Fernando Fernández. Safe exploration of state and action spaces in reinforcement learning. *Journal of Artificial Intelligence Research*, 45:515–564, 2012.

Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P. Vetrov, and Andrew Gordon Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 8803–8812, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/be3087e74e9100d4bc4c6268cdbe8456-Abstract.html.

Alborz Geramifard, Joshua Redding, Nicholas Roy, and Jonathan P How. UAV cooperative control with stochastic risk models. In *Proceedings of the 2011 American Control Conference*, pages 3393–3398. IEEE, 2011.

Shlomo Geva and Joaquin Sitte. The cart-pole experiment as a benchmark for trainable controllers. Technical report, AUS, 1992.

Amir Gholaminejad, Kurt Keutzer, and George Biros. ANODE: unconditionally accurate memory-efficient gradients for neural odes. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 730–736. ijcai.org, 2019. doi: 10.24963/ijcai.2019/103.

Michel Goemans and Jan Vondrák. Stochastic covering and adaptivity. In *Latin American symposium on theoretical informatics*, pages 532–543. Springer, 2006.

Aidan N. Gomez, Mengye Ren, Raquel Urtasun, and Roger B. Grosse. The reversible residual network: Backpropagation without storing activations. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 2214–2224, 2017.

Ian J. Goodfellow and Oriol Vinyals. Qualitatively characterizing neural network optimization problems. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6544.

Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. FFJORD: free-form continuous dynamics for scalable reversible generative models. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 399–406. Omnipress, 2010.

Andreas Griewank and Andrea Walther. *Evaluating derivatives - principles and techniques of algorithmic differentiation, Second Edition*. SIAM, 2008. ISBN 978-0-89871-659-7. doi: 10.1137/1.9780898717761.

Aditya Grover, Eric Wang, Aaron Zweig, and Stefano Ermon. Stochastic optimization of sorting networks via continuous relaxations. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=H1eSS3CcKX.

Ernst Hairer, Syvert P. Nørsett, and Gerhard Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. 1993. doi: 10.1007/978-3-540-78862-1.

Matthew Hatem and Wheeler Ruml. Simpler bounded suboptimal search. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.90. URL https://doi.org/10.1109/CVPR.2016.90.

Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, and Ian Osband. Deep Q-learning from demonstrations. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Noriaki Hirose, Fei Xia, Roberto Martín-Martín, Amir Sadeghian, and Silvio Savarese. Deep visual mpc-policy learning for navigation. *IEEE Robotics Autom. Lett.*, 4(4):3184–3191, 2019. doi: 10.1109/LRA.2019.2925731.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, 2016.

Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Trans. Graph.*, 38(6):201:1–201:16, 2019. doi: 10.1145/3355089.3356506.

Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

Arthur Jacot, Clément Hongler, and Franck Gabriel. Neural tangent kernel: Convergence and generalization in neural networks. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 8580–8589, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/5a4be1fa34e62bb8a6ec6b91d2462f5a-Abstract.html.

Svante Janson. Random coverings in several dimensions. *Acta Mathematica*, 156(1):83–118, 1986. ISSN 0001-5962.

Yiding Jiang, Vaishnavh Nagarajan, Christina Baek, and J. Zico Kolter. Assessing generalization of SGD via disagreement. *CoRR*, abs/2106.13799, 2021. URL https://arxiv.org/abs/2106.13799.

Wanxin Jin, Zhaoran Wang, Zhuoran Yang, and Shaoshuai Mou. Pontryagin differentiable programming: An end-to-end learning and control framework. 2020.

Roy Jonker and A. Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38(4):325–340, 1987. doi: 10.1007/BF02278710. URL https://doi.org/10.1007/BF02278710.

Gregory Kahn, Adam Villaflor, Vitchyr Pong, Pieter Abbeel, and Sergey Levine. Uncertainty-aware reinforcement learning for collision avoidance. *arXiv: preprint*, 2017.

Sham Kakade, Michael J Kearns, and John Langford. Exploration in metric state spaces. In *Proceedings of the 20th International Conference on Machine Learning*, 2003.

Sham Kakade, Mengdi Wang, and Lin F Yang. Variance reduction methods for sublinear reinforcement learning. *arXiv:1802.09184*, 2018.

Rudolf Emil Kálmán. The theory of optimal control and the calculus of variations. 1960.

Zhao Kang, Chong Peng, and Qiang Cheng. Top-n recommender system via matrix completion. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 179–185. AAAI Press, 2016. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/11824.

Kenji Kawaguchi. Deep learning without poor local minima. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 586–594, 2016. URL https://proceedings.neurips.cc/paper/2016/hash/f2fc990265c712c49d51a18a32b39f0c-Abstract.html.

Matthew Kelly. An introduction to trajectory optimization: How to do your own direct collocation. *SIAM Rev.*, 59(4):849–904, 2017. doi: 10.1137/16M1062569. URL https://doi.org/10.1137/16M1062569.

Taewan Kim, Chungkeun Lee, Hoseong Seo, Seungwon Choi, Wonchul Kim, and H. Jin Kim. Vision-based target tracking for a skid-steer vehicle using guided policy search with field-of-view constraint. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain, October 1-5, 2018*, pages 2418–2425. IEEE, 2018. doi: 10.1109/IROS.2018.8593843.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

Jakub Konečný, H. Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *CoRR*, abs/1610.02527, 2016a. URL http://arxiv.org/abs/1610.02527.

Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and

Dave Bacon. Federated learning: Strategies for improving communication efficiency. *CoRR*, abs/1610.05492, 2016b. URL http://arxiv.org/abs/1610.05492.

Tjalling C. Koopmans and Martin Beckmann. Assignment problems and the location of economic activities. *Econometrica*, 25(1):53–76, 1957. ISSN 00129682, 14680262. URL http://www.jstor.org/stable/1907742.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

Harold W. Kuhn. The hungarian method for the assignment problem. In Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, pages 29–47. Springer, 2010. doi: 10.1007/978-3-540-68279-0_2. URL https://doi.org/10.1007/978-3-540-68279-0_2.

Aditya Kusupati, Matthew Wallingford, Vivek Ramanujan, Raghav Somani, Jae Sung Park, Krishna Pillutla, Prateek Jain, Sham M. Kakade, and Ali Farhadi. LLC: accurate, multi-purpose learnt low-dimensional binary codes. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 23900–23913, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/c88d8d0a6097754525e02c2246d8d27f-Abstract.html.

W. Kutta. Beitrag zur näherungsweisen Integration totaler Differentialgleichungen. *Zeit. Math. Phys.*, 46:435–53, 1901.

Michael Laskey, Sam Staszak, Wesley Yu-Shu Hsieh, Jeffrey Mahler, Florian T Pokorny, Anca D Dragan, and Ken Goldberg. SHIV: Reducing supervisor burden in DAgger using support vectors for efficient learning from demonstrations in high dimensional state spaces. In *International Conference on Robotics and Automation*, 2016.

Michael Laskey, Caleb Chuck, Jonathan Lee, Jeffrey Mahler, Sanjay Krishnan, Kevin Jamieson, Anca Dragan, and Ken Goldberg. Comparing human-centric and robot-centric sampling for robot deep learning from demonstrations. In *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017*, pages 358–365. IEEE, 2017. doi: 10.1109/ICRA.2017.7989046. URL https://doi.org/10.1109/ICRA.2017.7989046.

Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791. URL https://doi.org/10.1109/5.726791.

Jaehoon Lee, Lechao Xiao, Samuel S. Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8570–8581, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/0d1a9651497a38d8b1c3871c84528bd4-Abstract.html.

Randall J. LeVeque. *Finite difference methods for ordinary and partial differential equations - steady-state and time-dependent problems*. SIAM, 2007. ISBN 978-0-89871-629-0. doi: 10.1137/1.9780898717839.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 6391–

6401, 2018a. URL https://proceedings.neurips.cc/paper/2018/hash/a41b3bb3e6b050b6c9067c67f663b915-Abstract.html.

Liam Li. Massively Parallel Hyperparameter Optimization, 2018. URL https://blog.ml.cmu.edu/2018/12/12/massively-parallel-hyperparameter-optimization/.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1), 2018b. ISSN 1532-4435.

Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John E. Hopcroft. Convergent learning: Do different neural networks learn the same representations? In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1511.07543.

Yunzhu Li, Jiajun Wu, Jun-Yan Zhu, Joshua B. Tenenbaum, Antonio Torralba, and Russ Tedrake. Propagation networks for model-based control under partial observation. In *International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019*, pages 1205–1211. IEEE, 2019. doi: 10.1109/ICRA.2019.8793509.

Yunzhu Li, Hao He, Jiajun Wu, Dina Katabi, and Antonio Torralba. Learning compositional koopman operators for model-based control. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1509.02971.

James Lucas, Juhan Bae, Michael R. Zhang, Stanislav Fort, Richard S. Zemel, and Roger B. Grosse. On monotonic linear interpolation of neural network parameters. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 7168–7179. PMLR, 2021. URL http://proceedings.mlr.press/v139/lucas21a.html.

Frederic Maire and Vadim Bulitko. Apprenticeship learning for initial value functions in reinforcement learning. In *IJCAI Workshop on Planning and Learning in A Priori Unknown or Dynamic Domains*, 2005.

Michael Matena and Colin Raffel. Merging models with fisher-weighted averaging. *CoRR*, abs/2111.09832, 2021. URL https://arxiv.org/abs/2111.09832.

Andreas Maurer and Massimiliano Pontil. Empirical bernstein bounds and sample variance penalization. In *COLT 2009 - The 22nd Conference on Learning Theory*, jul 2009. URL http://arxiv.org/abs/0907.3740.

Michael McAsey, Libin Mou, and Weimin Han. Convergence of the forward-backward sweep method in optimal control. *Comput. Optim. Appl.*, 53(1):207–226, 2012. doi: 10.1007/s10589-011-9454-7.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In Aarti Singh and Xiaojin (Jerry) Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 2017. URL http://proceedings.mlr.press/v54/mcmahan17a.html.

Gonzalo E. Mena, David Belanger, Scott W. Linderman, and Jasper Snoek. Learning latent permutations with gumbel-sinkhorn networks. In *6th International Conference on Learning Representa-*

tions, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. OpenReview.net, 2018. URL https://openreview.net/forum?id=Byt3oJ-0W.

Sobhan Miryoosefi, Kianté Brantley, Hal Daumé III, Miroslav Dudik, and Robert Schapire. Reinforcement Learning with Convex Constraints. *arXiv preprint arXiv:1906.09323*, 2019.

Forest Ray Moulton. *New methods in exterior ballistics*. 1926.

Michael C. Mozer. A focused backpropagation algorithm for temporal pattern recognition. *Complex Syst.*, 3(4), 1989.

Tong Mu, Karan Goel, and Emma Brunskill. PLOTS: Procedure Learning from Observations using Subtask Structure. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1007–1015. International Foundation for Autonomous Agents and Multiagent Systems, 2019. ISBN 1450363091.

Rémi Munos. Policy gradient in continuous time. *J. Mach. Learn. Res.*, 7:771–791, 2006.

Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022. URL probml.ai.

Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *International Conference on Robotics and Automation*, 2018.

Thao Nguyen, Maithra Raghu, and Simon Kornblith. Do wide and deep networks learn the same things? uncovering how neural network representations vary with width and depth. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=KJNcAkY8tY4.

Laurel J. Orr, Samuel K. Ainsworth, Kevin G. Jamieson, Walter Cai, Magdalena Balazinska, and

Dan Suciu. Mosaic: A Sample-Based Database System for Open World Query Processing. In *10th Conference on Innovative Data Systems Research, CIDR*, 2020.

Romeo Orsolino, Michele Focchi, Carlos Mastalli, Hongkai Dai, Darwin G. Caldwell, and Claudio Semini. Application of wrench-based feasibility analysis to the online trajectory optimization of legged robots. *IEEE Robotics Autom. Lett.*, 3(4):3363–3370, 2018.

Felix Petersen, Christian Borgelt, Hilde Kuehne, and Oliver Deussen. Differentiable sorting networks for scalable sorting and ranking supervision. *CoRR*, abs/2105.04019, 2021. URL `https://arxiv.org/abs/2105.04019`.

Felix Petersen, Christian Borgelt, Hilde Kuehne, and Oliver Deussen. Monotonic differentiable sorting networks. *CoRR*, abs/2203.09630, 2022. doi: 10.48550/arXiv.2203.09630. URL `https://doi.org/10.48550/arXiv.2203.09630`.

L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko. *The mathematical theory of optimal processes*. The Macmillan Co., New York, 1964. Translated by D. E. Brown, A Pergamon Press Book.

Sebastian Prillo and Julian Eisenschlos. Softsort: A continuous relaxation for the argsort operator. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 7793–7802. PMLR, 2020. URL `http://proceedings.mlr.press/v119/prillo20a.html`.

Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C. Lin. Scalable differentiable physics for learning and control. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 7847–7856. PMLR, 2020.

Alessio Quaglino, Marco Gallieri, Jonathan Masci, and Jan Koutník. SNODE: spectral discretization of neural odes for system identification. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

Maxim Rabinovich, Aaditya Ramdas, Michael I Jordan, and Martin J Wainwright. Optimal Rates and Tradeoffs in Multiple Testing. *arXiv preprint arXiv:1705.05391*, 2017.

Christopher Rackauckas and Qing Nie. Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in julia. *Journal of Open Research Software*, 5(1), 2017.

Christopher Rackauckas, Yingbo Ma, Vaibhav Dixit, Xingjian Guo, Mike Innes, Jarrett Revels, Joakim Nyberg, and Vijay Ivaturi. A comparison of automatic differentiation and continuous sensitivity analysis for derivatives of differential equation solutions. *CoRR*, abs/1812.01892, 2018.

Colin Raffel. A call to build models like we build open-source software. `https://colinraffel.com/blog/a-call-to-build-models-like-we-build-open-source-software.html`, 2021. Accessed: 2022-06-17.

Aaditya Ramdas, Rina Foygel Barber, Martin J Wainwright, and Michael I Jordan. A unified treatment of multiple testing with prior knowledge using the p-filter. *arXiv preprint arXiv:1703.06222*, 2017.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, volume 9908 of *Lecture Notes in Computer Science*, pages 525–542. Springer, 2016. doi: 10.1007/978-3-319-46493-0_32. URL `https://doi.org/10.1007/978-3-319-46493-0_32`.

Charles Richter and Nicholas Roy. Safe visual navigation via deep learning and novelty detection. In *Robotics: Science and Systems*, 2017.

A. J. Robinson and Frank Fallside. The utility driven dynamic error propagation network. Technical

Report CUED/F-INFENG/TR.1, Engineering Department, Cambridge University, Cambridge, UK, 1987.

Anthony J. Robinson and F. Failside. Static and dynamic error propagation networks with application to speech coding. In Dana Z. Anderson, editor, *Neural Information Processing Systems, Denver, Colorado, USA, 1987*, pages 632–641. American Institue of Physics, 1987.

Stephane Ross, Geoffrey J Gordon, and J Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. *14th International Conference on Artificial Intelligence and Statistics*, 2011.

Stéphane Ross, Narek Melik-Barkhudarov, Kumar Shaurya Shankar, Andreas Wendel, Debadeepta Dey, J Andrew Bagnell, and Martial Hebert. Learning monocular reactive UAV control in cluttered natural environments. In *IEEE International Conference on Robotics and Automation*, 2013.

Carl Runge. Über die numerische auflösung von differentialgleichungen. *Mathematische Annalen*, 46(2):167–178, 1895.

F. Sabatino. Quadrotor control: modeling, nonlinear control design, and simulation. 2015.

Sartaj Sahni and Teofilo F. Gonzalez. P-complete approximation problems. *J. ACM*, 23(3):555–565, 1976. doi: 10.1145/321958.321975. URL `https://doi.org/10.1145/321958.321975`.

Joao Salvado, Robert Krug, Masoumeh Mansouri, and Federico Pecora. Motion planning and goal assignment for robot fleets using trajectory optimization. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain, October 1-5, 2018*, pages 7939–7946. IEEE, 2018.

Connor Schenck and Dieter Fox. Reasoning about liquids via closed-loop simulation. In Nancy M. Amato, Siddhartha S. Srinivasa, Nora Ayanian, and Scott Kuindersma, editors, *Robotics: Science*

and Systems XIII, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, July 12-16, 2017, 2017.

Connor Schenck and Dieter Fox. Spnets: Differentiable fluid dynamics for deep neural networks. In *2nd Annual Conference on Robot Learning, CoRL 2018, Zürich, Switzerland, 29-31 October 2018, Proceedings*, volume 87 of *Proceedings of Machine Learning Research*, pages 317–335. PMLR, 2018.

L.F. Shampine and H.A. Watts. *DEPAC: Design of a User Oriented Package of ODE Solvers*. Sandia report. Sandia National Laboratories. UC 32, 1979.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1409.1556`.

Sidak Pal Singh and Martin Jaggi. Model fusion via optimal transport. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL `https://proceedings.neurips.cc/paper/2020/hash/fb2697869f56484404c8ceee2985b01d-Abstract.html`.

William D Smart and Leslie Pack Kaelbling. Practical reinforcement learning in continuous spaces. In *Proceedings of the 17th International Conference on Machine Learning*, 2000.

Jason L. Speyer and David H. Jacobson. *Primer on Optimal Control Theory*. SIAM, 2010. ISBN 978-0-89871-694-8. doi: 10.1137/1.9780898718560.

Colin Summers, Kendall Lowrey, Aravind Rajeswaran, Siddhartha S. Srinivasa, and Emanuel Todorov. Lyceum: An efficient and scalable ecosystem for robot learning. In Alexandre M. Bayen, Ali Jadbabaie, George J. Pappas, Pablo A. Parrilo, Benjamin Recht, Claire J. Tomlin, and Melanie N. Zeilinger, editors, *Proceedings of the 2nd Annual Conference on Learning for*

Dynamics and Control, L4DC 2020, Online Event, Berkeley, CA, USA, 11-12 June 2020, volume 120 of *Proceedings of Machine Learning Research*, pages 793–803. PMLR, 2020.

Yi-Lin Sung, Varun Nair, and Colin Raffel. Training neural networks with fixed sparse masks. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 24193–24205, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/cb2653f548f8709598e8b5156738cc51-Abstract.html.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2018.

Umar Syed and Robert E Schapire. A game-theoretic approach to apprenticeship learning. In *Advances in neural information processing systems*, pages 1449–1456, 2008.

Gao Tang, Weidong Sun, and Kris Hauser. Learning trajectories for real- time optimal control of quadrotors. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain, October 1-5, 2018*, pages 3620–3625. IEEE, 2018. doi: 10.1109/IROS.2018.8593536.

Russ Tedrake. *Underactuated Robotics: Algorithms for Walking, Running, Swimming, Flying, and Manipulation*. URL http://underactuated.mit.edu/.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*, pages 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

Faraz Torabi, Garrett Warnell, and Peter Stone. Generative adversarial imitation from observation. *arXiv:1807.06158*, 2018.

Charalampos Tsitouras. Runge-kutta pairs of order 5(4) satisfying only the first column simplifying assumption. *Comput. Math. Appl.*, 62(2):770–775, 2011.

Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy P. Lillicrap, Nicolas Heess, and Yuval Tassa. dm_control: Software and tasks for continuous control. *Softw. Impacts*, 6:100022, 2020. doi: 10.1016/j.simpa.2020.100022.

Julian Viereck, Jules Kozolinsky, Alexander Herzog, and Ludovic Righetti. Learning a structured neural network policy for a hopping task. *IEEE Robotics Autom. Lett.*, 3(4):4092–4099, 2018. doi: 10.1109/LRA.2018.2861466.

Tiffany Vlaar and Jonathan Frankle. What can linear interpolation of neural network loss landscapes tell us? *CoRR*, abs/2106.16004, 2021. URL `https://arxiv.org/abs/2106.16004`.

Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris S. Papailiopoulos, and Yasaman Khazaeni. Federated learning with matched averaging. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=BkluqlSFDS`.

Peng Wang, Yanzhao Cao, Xiaoying Han, and Peter Kloeden. Mean-square convergence of numerical methods for random ordinary differential equations. *Numer. Algorithms*, 87(1): 299–333, 2021. doi: 10.1007/s11075-020-00967-w.

Paul J. Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339–356, 1988. doi: 10.1016/0893-6080(88)90007-X.

Bernard Widrow and Fred W Smith. Pattern-recognizing control systems. *Computer and Information Sciences*, pages 288–317, 1964.

Mitchell Wortsman, Gabriel Ilharco, Samir Yitzhak Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. *Proceedings of the 39th International Conference on Machine Learning, ICML 2022*, 2022a.

Mitchell Wortsman, Gabriel Ilharco, Jong Wook Kim, Mike Li, Simon Kornblith, Rebecca Roelofs, Raphael Gontijo Lopes, Hannaneh Hajishirzi, Ali Farhadi, Hongseok Namkoong, et al. Robust fine-tuning of zero-shot models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7959–7971, 2022b.

Fanny Yang, Aaditya Ramdas, Kevin G Jamieson, and Martin J Wainwright. A framework for Multi-A (rmed)/B (andit) testing with online FDR control. In *Advances in Neural Information Processing Systems*, pages 5957–5966, 2017.

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3320–3328, 2014. URL https://proceedings.neurips.cc/paper/2014/hash/375c71349b295fbe2dcdca9206f20a06-Abstract.html.

Andrew Zangwill. *Modern electrodynamics*. Cambridge University Press, Cambridge, 2013.

Chiyuan Zhang, Samy Bengio, and Yoram Singer. Are all layers created equal? *CoRR*, abs/1902.01996, 2019. URL http://arxiv.org/abs/1902.01996.

Appendix A

POLICY LEARNING WITH INTERVENTION

A.1 Proof of Eq. (2.5)

Lemma. *Let $M = \langle \mathcal{S}, \mathcal{A}, P, R, H, \rho_0 \rangle$ be a finite horizon, episodic Markov decision process, and $\widehat{M} = (\hat{\mathcal{S}}, \mathcal{A}, P_{\hat{\mathcal{S}}}, R_{\hat{\mathcal{S}}}, H, \rho_0)$ be a corresponding e-stop version of M. Given a reinforcement learning algorithm $\mathfrak{A}$, the regret in M after running $\mathfrak{A}$ for T timesteps in $\widehat{M}$ is bounded by*

$$\text{Regret}(T) \leq \lceil * \rceil \tfrac{T}{H} \left[J(\pi^*) - J(\hat{\pi}^*) \right] + \mathbb{E}^{\hat{\pi}^*}_{\widehat{M}} \left[R_T \right] - \mathbb{E}^{\mathfrak{A}}_{\widehat{M}} \left[R_T \right] \tag{A.1}$$

where π^* and $\hat{\pi}^*$ are the optimal policies in M and $\widehat{M}$, respectively

Proof. Let $\hat{\pi}^* = \arg\max_{\pi \in \Pi} J_{\widehat{M}}(\pi)$. Then beginning with the external regret definition,

$$\begin{aligned}
\text{Regret}^{\mathfrak{A}}_{M}(T) &= \mathbb{E}^{\pi_e}_{M} \left[R_T \right] - \mathbb{E}^{\mathfrak{A}}_{M} \left[R_T \right] && \text{(A.2)} \\
&= \left[\mathbb{E}^{\pi_e}_{M} \left[R_T \right] - \mathbb{E}^{\hat{\pi}^*}_{\widehat{M}} \left[R_T \right] \right] + \left[\mathbb{E}^{\hat{\pi}^*}_{\widehat{M}} \left[R_T \right] - \mathbb{E}^{\mathfrak{A}}_{\widehat{M}} \left[R_T \right] \right] + \left[\mathbb{E}^{\mathfrak{A}}_{\widehat{M}} \left[R_T \right] - \mathbb{E}^{\mathfrak{A}}_{M} \left[R_T \right] \right] && \text{(A.3)} \\
&\leq \left[\mathbb{E}^{\pi_e}_{M} \left[R_T \right] - \mathbb{E}^{\hat{\pi}^*}_{\widehat{M}} \left[R_T \right] \right] + \left[\mathbb{E}^{\hat{\pi}^*}_{\widehat{M}} \left[R_T \right] - \mathbb{E}^{\mathfrak{A}}_{\widehat{M}} \left[R_T \right] \right] && \text{(A.4)} \\
&\leq \left[\mathbb{E}^{\pi_e}_{M} \left[R_T \right] - \mathbb{E}^{\hat{\pi}^*}_{M} \left[R_T \right] \right] + \left[\mathbb{E}^{\hat{\pi}^*}_{\widehat{M}} \left[R_T \right] - \mathbb{E}^{\mathfrak{A}}_{\widehat{M}} \left[R_T \right] \right] && \text{(A.5)} \\
&\leq \lceil * \rceil \frac{T}{H} \left[J(\pi_e) - J(\hat{\pi}^*) \right] + \left[\mathbb{E}^{\hat{\pi}^*}_{\widehat{M}} \left[R_T \right] - \mathbb{E}^{\mathfrak{A}}_{\widehat{M}} \left[R_T \right] \right] && \text{(A.6)}
\end{aligned}$$

Eq. (A.4) follows by the definition of $\widehat{M}$. □

A.2 Proof of Theorem 2.4.1

Theorem. *Suppose $\widehat{M}$ is an e-stop variant of M such that $\hat{\mathcal{S}} = \{s | h(s) > 0\}$ where $h(s)$ denotes the probability of hitting state s in a roll-out of π_e. Let $\hat{\pi}^* = \arg\max_{\pi \in \Pi} J_{\widehat{M}}(\pi)$ be the optimal policy in $\widehat{M}$. Then $J(\hat{\pi}^*) \geq J(\pi_e)$.*

Proof. Let $J_{\widehat{M}}(\pi)$ denote the value of executing policy π in $M_{\hat{\mathcal{S}}}$. By the definition of $M_{\hat{\mathcal{S}}}$,

$$J(\pi) \geq J_{\widehat{M}}(\pi) \quad \forall \pi \tag{A.7}$$

$$J_{\widehat{M}}(\pi_e) = J(\pi_e) \tag{A.8}$$

because π_e never leaves $\mathcal{S}_{\pi_e}$ (and thus never leaves $\hat{\mathcal{S}}$). Finally, by the definition of $\hat{\pi}^*$ and the realizability of π_e,

$$J_{\widehat{M}}(\hat{\pi}^*) \geq J_{\widehat{M}}(\pi_e) \tag{A.9}$$

Combining Eqs. (A.7) to (A.9) implies $J(\hat{\pi}^*) > J(\pi_e)$. □

A.3 Proof of Theorem 2.4.2

Theorem. *Consider $\widehat{M}$, an e-stop variation on MDP M with state spaces $\hat{\mathcal{S}}$ and $\mathcal{S}$, respectively. Given an expert policy, π_e, let $h(s)$ denote the probability of visiting state s at least once in an episode roll-out of policy π_e in M. Then*

$$J(\pi_e) - J(\hat{\pi}^*) \leq H \sum_{s \in \mathcal{S} \setminus \hat{\mathcal{S}}} h(s) \tag{A.10}$$

where $\hat{\pi}^*$ is the optimal policy in $\widehat{M}$. Naturally if we satisfy some "allowance," ξ, such that $\sum_{s \in \mathcal{S} \setminus \hat{\mathcal{S}}} h(s) \leq \xi$ then $J(\pi_e) - J(\hat{\pi}^*) \leq \xi H$.

Proof. We proceed by analyzing the probabilities and expected rewards of entire trajectories $\tau = (\tau_1, \ldots, \tau_H)$, in M and $\widehat{M}$. Let

$$\mu(\tau) = \sum_{t=1}^{H-1} \mathbb{E}\left[R(\tau_t, A_t, \tau_{t+1}) | \tau, \pi_e\right] \tag{A.11}$$

be the expected reward of a trajectory τ and let $p_M(\tau)$ denote the probability of trajectory τ when following policy π_e in MDP M. Note that

$$h(s) = \sum_{\tau} p_S(\tau) \mathbb{I}\{s \in \tau\} \tag{A.12}$$

Now,

$$J(\pi_e) - J(\hat{\pi}^*) \leq J_M(\pi_e) - J_{\widehat{M}}(\hat{\pi}^*) \tag{A.13}$$

$$\leq J_M(\pi_e) - J_{\widehat{M}}(\pi_e) \tag{A.14}$$

$$= \sum_\tau p_S(\tau)\mu(\tau) - \sum_\tau p_{\hat{S}}(\tau)\mu(\tau) \tag{A.15}$$

$$\leq \sum_\tau p_S(\tau)\mu(\tau)\mathbb{I}\{\tau \text{ leaves } \hat{S}\} \tag{A.16}$$

$$\leq H \sum_\tau p_S(\tau)\mathbb{I}\{\tau \text{ leaves } \hat{S}\} \tag{A.17}$$

$$\leq H \sum_\tau p_S(\tau) \sum_{s \in S \setminus \hat{S}} \mathbb{I}\{s \in \tau\} \tag{A.18}$$

$$= H \sum_{s \in S \setminus \hat{S}} \sum_\tau p_S(\tau)\mathbb{I}\{s \in \tau\} \tag{A.19}$$

$$= H \sum_{s \in S \setminus \hat{S}} h(s) \tag{A.20}$$

as desired. □

A.4 Proof of Corollary 2.4.2.1

Corollary. *Recall that $\rho_{\pi_e}(s)$ denotes the average state distribution following actions from π_e, $\rho_{\pi_e}(s) = \frac{1}{H}\sum_{t=0}^{H-1} \rho^t_{\pi_e}(s)$. Then*

$$J(\pi_e) - J(\hat{\pi}^*) \leq \rho_{\pi_e}(\mathcal{S} \setminus \hat{\mathcal{S}})H^2 \tag{A.21}$$

Proof. Note that

$$h(s) = \mathbb{P}\left(\bigcup_{t=0}^{H-1}(s_t = s)\right) \leq \sum_{t=0}^{H-1} \rho^t_{\pi_e}(s) = H\rho_{\pi_e}(s) \tag{A.22}$$

where the inequality follows from a union bound over time steps. Then

$$J(\pi_e) - J(\hat{\pi}^*) \leq \rho_{\pi_e}(S \setminus \hat{S})H^2 \tag{A.23}$$

as a consequence of Theorem 2.4.2. □

A.5 Proof of Theorem 2.5.1

Theorem. *The e-stop MDP $\widehat{M}$ with states $\hat{\mathcal{S}}$ in Algorithm 1 has asymptotic sub-optimality*

$$J(\pi_e) - J(\hat{\pi}^*) \leq (\xi + \epsilon)H \tag{A.24}$$

with probability at least $1 - |\mathcal{S}|e^{-2\epsilon^2 n/|\mathcal{S}|^2}$, for any $\epsilon > 0$. Here ξ denotes our approximate state removal "allowance", where we satisfy $\sum_{s\in\mathcal{S}\setminus\hat{\mathcal{S}}} \hat{h}(s) \leq \xi$ in our construction of $\widehat{M}$ as in Theorem 2.4.2.

Proof. With Hoeffding's inequality and a union bound,

$$\mathbb{P}(\forall s, \hat{h}(s) > h(s) - \epsilon/|\mathcal{S}|) = 1 - \mathbb{P}(\exists s, \hat{h}(s) \leq h(s) - \epsilon/|\mathcal{S}|) \tag{A.25}$$

$$\geq 1 - |\mathcal{S}|e^{-2\epsilon^2 n/|\mathcal{S}|^2} \tag{A.26}$$

Note that the $\hat{h}(s)$ values are not independent yet the union bound still allows us to bound the probability that any of them deviate meaningfully from $h(s)$. Now if $\hat{h}(s) > h(s) - \epsilon/|\mathcal{S}|$ for all s, it follows that

$$\xi \geq \sum_{s\in\mathcal{S}\setminus\hat{\mathcal{S}}} h(s) - \frac{\epsilon}{|\mathcal{S}|}(|\mathcal{S}| - |\hat{\mathcal{S}}|) \geq \sum_{s\in\mathcal{S}\setminus\hat{\mathcal{S}}} h(s) - \epsilon \tag{A.27}$$

and so $\sum_{s\in\mathcal{S}\setminus\hat{\mathcal{S}}} h(s) \leq \xi + \epsilon$. By Theorem 2.4.2 we have that

$$J(\pi_e) - J(\hat{\pi}^*) \leq (\xi + \epsilon)H \tag{A.28}$$

completing the proof. □

A.6 Imperfect e-stops in terms of $\hat{\rho}_{\pi_e}(s)$

While Theorem 2.5.1 provides an analysis for an approximate e-stopping algorithm, its reliance on hitting probabilities does not extend nicely to continuous domains. Here we present a result analogous to Theorem 2.5.1, but using $\hat{\rho}_{\pi_e}(s)$ in place of $\hat{h}(s)$. Unfortunately, we are not able to escape a dependence on $|\mathcal{S}|$ with this approach however. Furthermore, we require that π_e always runs to episode completion without hitting any terminal states, ie. the length of all π_e roll-outs is H.

Definition A.6.1. Let $\varrho(s)$ be a random variable denoting the average number of times π_e visits state s,

$$\varrho(s) \triangleq \frac{|\{t \in 1, \ldots, H | \tau_t = s\}|}{H}. \tag{A.29}$$

Note that with n roll-outs, our approximate average state distribution is the same as the average of the ϱ's:

$$\hat{\rho}_{\pi_e}(s) \triangleq \frac{1}{nH} \sum_{i=1}^{n} \sum_{t=1}^{H} \mathbb{I}\{\tau_t^{(i)} = s\} = \frac{1}{n} \sum_{i=1}^{n} \varrho^{(i)}(s).$$

Theorem A.6.1. *The e-stop MDP $\widehat{M}$ with states $\hat{\mathcal{S}}$ resulting from running the $\hat{\rho}_{\pi_e}$ version of Algorithm 1 with n expert roll-outs has asymptotic sub-optimality*

$$J(\pi_e) - J(\hat{\pi}^*) \leq \left(\xi + \sqrt{\frac{2\log(2|\mathcal{S}|/\delta)}{n}} \sum_{s \in \mathcal{S}} \sqrt{V_n(\varrho^{(1:n)}(s))} + \frac{7|\mathcal{S}|\log(2|\mathcal{S}|/\delta)}{3(n-1)} \right) H^2 \tag{A.30}$$

with probability at least $1 - \delta$. Here ξ denotes our approximate state removal "allowance", where we satisfy $\hat{\rho}_{\pi_e}(\mathcal{S} \setminus \hat{\mathcal{S}}) \leq \xi$ in our construction of $\widehat{M}$, and V_n denotes the sample variance.

Proof. We follow the same structure as in the proof of Theorem 2.5.1, but use an empirical Bernstein bound in place of Hoeffding's inequality. We know from Theorem 4 of Maurer and Pontil [2009] that, for each s,

$$\hat{\rho}_{\pi_e}(s) \leq \rho_{\pi_e}(s) - \left(\sqrt{\frac{2V_n(\varrho^{(1:n)}(s))\log(2|\mathcal{S}|/\delta)}{n}} + \frac{7\log(2|\mathcal{S}|/\delta)}{3(n-1)} \right) \tag{A.31}$$

with probability no more than $\delta/|\mathcal{S}|$. It follows that it will hold for *every* $s \in \mathcal{S}$ with probability at least $1 - \delta$. In that case we underestimated the true ρ-mass of any subset of the state space $\mathcal{S}$ by at most

$$\sqrt{\frac{2\log(2|\mathcal{S}|/\delta)}{n}} \sum_{s \in \mathcal{S}} \sqrt{V_n(\varrho^{(1:n)}(s))} + \frac{7|\mathcal{S}|\log(2|\mathcal{S}|/\delta)}{3(n-1)} \tag{A.32}$$

and so by Theorem 2.4.2 we have the desired result. □

A.7 Experimental details

All experiments were implemented with Numpy and JAX [Bradbury et al., 2018]. Experiments were run on AWS.

Our code and results are available on GitHub at `https://github.com/samuela/e-stops`.

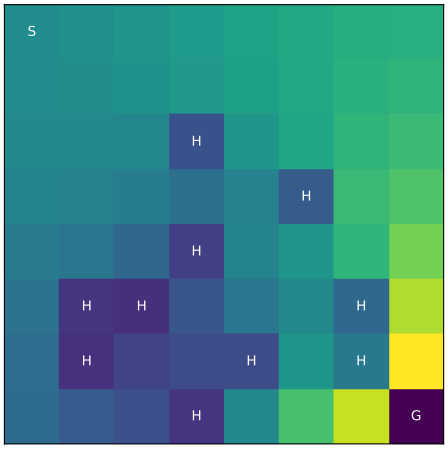

Figure A.1: Our FrozenLake-v0 environment. The agent starts in the upper left square and attempts to reach the goal in the lower right square. Tiles marked with "H" are holes in the lake which the agent can fall in and recover with only probability 0.01. The optimal state-value function is overlaid.

A.7.1 Value iteration

We ran value iteration on the full environment to convergence (tolerance $1e-6$) to establish the optimal policy. We calculated the state hitting probabilities of this policy exactly through an interpretation of expert policy roll-outs as absorbing Markov chains. These hitting probabilities were then ranked and states were removed in order of their rank until there was no longer a feasible path to the goal ($J(\pi) = 0$). The number of floating point operations (FLOPs) used was calculated based on $4\ |\mathcal{S}|^2|A|$ FLOPs per value iteration update:

1. For each state s and each action a, calculating the expected value of the next state. ($|\mathcal{S}||A|$ dot products of $|\mathcal{S}|$-vectors.)

2. Multiplying those values by γ.

3. Adding in the expected rewards for every state-action-state transition.

4. Calculating the maximum for each state s and each action a over $|\mathcal{S}|$ possible next state outcomes.

A.7.2 Policy gradient methods

We ran value iteration on the full environment to convergence (tolerance $1e-6$) to establish the optimal policy. We estimated the state hitting probabilities of this policy with 1,000 roll-outs in the environment. Based on this estimate of $\rho_{\pi_e}(s)$ we replaced the least-visited 50% of states with e-stops.

We ran both Q-learning and Actor-Critic across 96 trials (random seeds 0-95) and plot the median performance per states seen. Error bars denote one standard deviation around the mean and are clipped to the maximum/minimum values. We ran iterative policy evaluation to convergence on the current policy every 10 episodes in order to calculate the cumulative policy reward as plotted.

In order to accommodate the fact that two trials may not have x-coordinates that align (episodes may not be the same length), we linearly interpolated values and plot every 1,000 states seen.

A.7.3 DDPG

Continuous results were trained with DDPG with $\gamma = 0.99, \tau = 0.0001$, Adam with learning rate 0.001, batch size 128, and action noise that was normally distributed with mean zero and standard deviation 0.1. The replay buffer had length $2^{20} = 1,048,576$. The actor network had structure

- Dense(64)
- ReLU
- Dense(64)
- ReLU

- Dense(action_shape)
- Tanh

and the critic network had structure

- Dense(64)
- ReLU
- Dense(64)
- ReLU
- Dense(64)
- ReLU
- Dense(1)

We periodically paused training to run policy evaluation on the current policy (without any action noise).

Plotting and error bars are the same as in the deterministic experiments.

Appendix B

POLICY LEARNING WITH DIFFERENTIABLE SIMULATION

B.1 Experimental models

For complete experimental details, please see our code at `https://github.com/samuela/ctpg`.

B.1.1 Linear quadratic regulator

As usual,

$$w(x, u) = x^\top Q x + u^\top R u, \tag{B.1}$$

$$f(x, u) = Ax + Bu. \tag{B.2}$$

For the example in Fig. 3.3, we used $A = 0$, $B = Q = R = I$, $x_0 = [1, 1]^\top$ and $T = 25$. The control policy was given by a single hidden layer neural network with 32 hidden units and tanh activations. Training was run for 1,000 iterations.

B.1.2 Differential drive robot

Following `http://planning.cs.uiuc.edu/node659.html`, the physical dynamics for a differential drive robot with left wheel speed ω_ℓ, right wheel speed ω_r, position x, y, heading θ, and wheelbase L are given by,

$$\frac{dx}{dt} = \frac{\omega_\ell + \omega_r}{2} \cos\theta, \quad \frac{dy}{dt} = \frac{\omega_\ell + \omega_r}{2} \sin\theta, \quad \frac{d\theta}{dt} = \frac{\omega_r - \omega_\ell}{L}. \tag{B.3}$$

Control is applied such that

$$\frac{d\omega_\ell}{dt} = u_l, \quad \frac{d\omega_r}{dt} = u_r. \tag{B.4}$$

The instantaneous cost is defined as

$$w(x, y, \omega_\ell, \omega_r, u_l, u_r) = x^2 + y^2 + \frac{1}{10}(\omega_\ell^2 + \omega_r^2 + u_l^2 + u_r^2). \tag{B.5}$$

For the control policy we used a two hidden layer neural network with 64 units in each hidden layer and tanh activations. We additionally augment the input to the network with a few useful manual features including $\cos\theta$ and $\sin\theta$.

B.1.3 DiffTaichi Electric

We integrated with the DiffTaichi example code. This integration required refactoring the example code to return forces. For an apples-to-apples timing comparison we also wrote a wrapper BPTT implementation following exactly the same implementation as the reference code. We evaluated BPTT with the same dt, and other hyperparameters, as was specified in the DiffTaichi code. Experiments were run with 32 random trials, and plots show the 5%-95% percentile error bars. Following Hu et al. [2020], we used a single hidden layer neural network policy with 64 hidden units and tanh activations.

B.1.4 MuJoCo Cartpole

The specifications of the MuJoCo Cartpole experiment are as follows. MuJoCo derivatives were calculated through forward finite differencing (chosen over central for speed) with an epsilon of $1e-6$. Derivatives with respect to positions, velocities, and controls were all calculated in this manner. For reference, an epsilon of $1e-4$ to $1e-8$ all perform approximately the same. The policy was a two layer network with 32 hidden units and `tanh` activations. The last layer's initialization weights were scaled, in this case by 0.1, as is typical in reinforcement learning Andrychowicz et al. [2020]. The policy function maps observations of the system to controls.

During optimization we used the ADAM optimizer with a step size of 0.001. The number of function evaluations we count includes those of the finite difference calculations. More specifically, every time the MuJoCo forward dynamics function `mj_forward` was called was included. We use a mini-batch of 2 rollouts, with each starting state sampled according to the DMcontrol

specifications; the sequence of starting states is set to be the same between BPTT and CTPG as well as the policy initializations. Both methods were run for a fixed 100 iterations, with the losses and number of function evaluations counted across 5 random seeds.

B.1.5 Quadrotor

The quadrotor policy was a two layer network with 16 hidden units and `tanh` activations. The policy was trained with the ADAM optimizer with a step size of 0.01, with the gradient values clipped to ± 1.0. Training was performed with 5 random seeds with a minibatch size of 32 per optimization step. The dynamics of the quadrotor experiment are given by Sabatino [2015].

B.2 Neural ODEs and their linear stability properties

Consider the LQR control problem with dynamics $\frac{dx(t)}{dt} = Ax(t) + Bu(t)$. Under a linear feedback policy $u(t) = -Kx(t)$, the system reduces to a first-order linear ODE: $\frac{dx(t)}{dt} = (A - BK)x(t)$. With any luck we will identify K that *stabilizes* the system so that $x(t)$ converges exponentially to 0 everywhere. Consider running such a system forward to some time T, and then attempting to run it backward along the same path. Because all paths will converge to the origin, when starting at the origin and trying to run backwards it becomes numerically infeasible to recover the correct trajectory to $x(0)$.

Claim. For any non-constant dynamics f, the dynamics of the Neural ODE backpropagation process will have eigenvalues $\pm\lambda_1, \ldots, \pm\lambda_d$ split among the $x(t)$ and $\alpha(t)$ process, along with zero eigenvalues for $g(t)$. Therefore the Neural ODE backpropagation process is linearly unstable for all t.

Let $\psi(t) = [x(t), \alpha(t), g(t)]$ denote the combined Neural ODE backpropagation process. The Neural ODE stipulates that in the reverse time solve we begin with $\psi(T)$ and follow the dynamics $-\dot{\psi}(t)$ as given in Eqs. 3.6, 3.7, and 3.8. Denoting the eigenvalues of $\frac{\partial f}{\partial x}$ as $\lambda_1, \ldots, \lambda_d$, we have

$$-\frac{\partial \dot{\psi}(t)}{\partial \psi(t)} = \begin{bmatrix} -\frac{\partial f}{\partial x} & 0 & 0 \\ \alpha(t)^\top \frac{\partial^2 f}{\partial x^2} & \left(\frac{\partial f}{\partial x}\right)^\top & 0 \\ 0 & \left(\frac{\partial f}{\partial \theta}\right)^\top & 0 \end{bmatrix} \tag{B.6}$$

which has eigenvalues $\pm\lambda_1, \ldots, \pm\lambda_d$, and 0 with multiplicity n_θ. Note that a system is considered stable when $\text{Re}(\lambda_i) < 0$ for all such eigenvalues λ_i. However because all eigenvalues come in positive and negative pairs, we are doomed to experience eigenvalues with positive real parts, indicating eigen-directions of exponential blowup.

B.3 Fusing Operations in the CTPG Algorithm

One opportunity for computational efficiency that we take advantage of in our experiments is that $\alpha(t)$ and $g(t)$ evolve in lockstep. This means that we do not need to store the estimates $\tilde{\alpha}(t)$

computed in Algorithm 2, but instead accumulate $g(t)$ concurrently as we solve for $\tilde{\alpha}(t)$. The most direct way to do this, while still treating the solver as a black box, is to simply concatenate or "fuse" the adjoint and gradient dynamics given by Equations equation 3.7 and equation 3.8 and perform a single backward solve to simultaneously compute the adjoints and accumulate the gradient. The automatic differentiation implementation of BPTT performs this fusion by default.

While the dynamics described by Eqs. equation 3.6 and equation 3.7 both evolve in $\mathbb{R}^d$, the dynamics equation 3.8 evolve in the parameter space of the parameter space of the policy π_θ, which can be large. Concatenating the dynamics on Eqs. equation 3.7 and equation 3.8 could cause problems for numerical ODE solvers that are designed for low dimensional (e.g. physical) spaces. Nevertheless, we find that fusing these two computations works well for the policies and physics considered in this work.

We could take this fusion perspective further and ask whether we can directly solve the full BVP, rather than sequentially computing the forward and backward passes. This is possible, and could be approached using a collocation algorithm. We favor the CTPG because, if our policy optimization is part of some larger robotic system that is sensitive to downstream consequences of arriving at a final state $x(T)$, then we need an estimate $\tilde{x}(T)$ in order to define the boundary condition $\frac{dJ}{dx(T)}$. This means that CTPG could be used not just as a policy gradient estimator, but also as a layer in a larger end-to-end problem with a continuous-time submodule.

Appendix C

DIFFERENCES BETWEEN POLICY LEARNING AND SUPERVISED LEARNING

C.1 Auxiliary plots

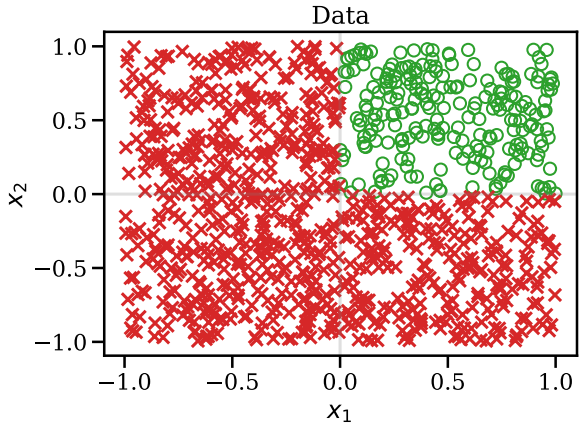

Figure C.1: The counterexample classification problem data.

C.2 Failed idea: A method for steepest descent

Imagine standing in weight space at Θ_A and trying to decide in which immediate direction to move in order to approach a Θ_B-equivalent point. There are many, many possible permutations of Θ_B – call them $\pi^{(1)}(\Theta_B), \pi^{(2)}(\Theta_B), \ldots$ – to aim for in the distance. Assuming that the loss landscape is in fact (quasi-)convex modulo these permutation symmetries, a natural choice would be to pick the $\pi^{(i)}(\Theta_B)$ that corresponds to the direction of steepest descent starting from Θ_A

since we expect $\pi^{(i)}(\Theta_B)$ to lie in the same basin as Θ_A. In other words,

$$\min_{\pi} \left. \frac{d\,\mathcal{L}(\Theta_A + \lambda(\pi(\Theta_B) - \Theta_A))}{d\,\lambda} \right|_{\lambda=0} = \min_{\pi} \nabla\mathcal{L}(\Theta_A)^\top(\pi(\Theta_B) - \Theta_A) \tag{C.1}$$

$$= -\nabla\mathcal{L}(\Theta_A)^\top\Theta_A + \min_{\pi} \nabla\mathcal{L}(\Theta_A)^\top\pi(\Theta_B) \tag{C.2}$$

Now, we are tenuously in a favorable situation: $\nabla\mathcal{L}(\Theta_A)$ is straightforward to compute, and picking the best π reduces to a matching problem. In particular it is a matching problem of the same form as in Section 4.3.2. However, there is a fast, exact solution for the single intermediate layer case ($L = 2$).

In practice, we found that this method can certainly find directions of steepest descent, but that they are paired with high barriers in between the initial dip and $\pi(\Theta_B)$.

C.3 Proof of Lemma 1

To lighten notation we use $\langle \cdot, \cdot \rangle = \langle \cdot, \cdot \rangle_F$ in this section.

Lemma. *Given* $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{d\times d}$*,*

$$\min_{\boldsymbol{P},\boldsymbol{Q}\ \textit{perm. matrices}} \langle \boldsymbol{P}\boldsymbol{A}\boldsymbol{Q}^\top, \boldsymbol{B} \rangle$$

is strongly NP-hard.

Proof. We proceed by reduction from the quadratic assignment problem (QAP) [Koopmans and Beckmann, 1957, Cela, 2013]. Consider a QAP,

$$\min_{\boldsymbol{P}\ \text{perm. matrix}} \langle \boldsymbol{P}\boldsymbol{C}\boldsymbol{P}^\top, \boldsymbol{D} \rangle$$

for $\boldsymbol{C}, \boldsymbol{D} \in \mathbb{R}^{d\times d}$.

Now, pick $\boldsymbol{A} = \boldsymbol{C} + \lambda \boldsymbol{I}$, $\boldsymbol{B} = \boldsymbol{D} - \lambda \boldsymbol{I}$. The we have,

$$\min_{\boldsymbol{P},\boldsymbol{Q}} \langle \boldsymbol{P}(\boldsymbol{C} + \lambda \boldsymbol{I})\boldsymbol{Q}^\top, \boldsymbol{D} - \lambda \boldsymbol{I} \rangle = \langle \boldsymbol{P}\boldsymbol{C}\boldsymbol{Q}^\top + \lambda \boldsymbol{P}\boldsymbol{Q}^\top, \boldsymbol{D} - \lambda \boldsymbol{I} \rangle \tag{C.3}$$

$$= \langle \boldsymbol{P}\boldsymbol{C}\boldsymbol{Q}^\top, \boldsymbol{D} \rangle - \lambda \langle \boldsymbol{P}\boldsymbol{C}\boldsymbol{Q}^\top, \boldsymbol{I} \rangle + \lambda \langle \boldsymbol{P}\boldsymbol{Q}^\top, \boldsymbol{D} \rangle - \lambda^2 \langle \boldsymbol{P}\boldsymbol{Q}^\top, \boldsymbol{I} \rangle \tag{C.4}$$

$$= \langle \boldsymbol{P}\boldsymbol{C}\boldsymbol{Q}^\top, \boldsymbol{D} \rangle - \lambda \langle \boldsymbol{P}^\top \boldsymbol{Q}, \boldsymbol{C} \rangle + \lambda \langle \boldsymbol{P}\boldsymbol{Q}^\top, \boldsymbol{D} \rangle - \lambda^2 \operatorname{tr}(\boldsymbol{P}\boldsymbol{Q}^\top) \tag{C.5}$$

For sufficiently large λ, the $\operatorname{tr}(\boldsymbol{P}\boldsymbol{Q}^\top)$ term will dominate. Letting $\alpha = \max\left(\max_{i,j} |C_{i,j}|, \max_{i,j} |D_{i,j}|\right)$, we can bound the other terms,

$$-d^2\alpha^2 \leq \quad \langle \boldsymbol{P}\boldsymbol{C}\boldsymbol{Q}^\top, \boldsymbol{D} \rangle \leq d^2\alpha^2 \tag{C.6}$$

$$-\lambda d\alpha \leq -\lambda \langle \boldsymbol{P}^\top \boldsymbol{Q}, \boldsymbol{C} \rangle \quad \leq \lambda d\alpha \tag{C.7}$$

$$-\lambda d\alpha \leq \quad \lambda \langle \boldsymbol{P}\boldsymbol{Q}^\top, \boldsymbol{D} \rangle \quad \leq \lambda d\alpha \tag{C.8}$$

Now there are two classes of solutions: those where $\boldsymbol{P} = \boldsymbol{Q}$ and those where $\boldsymbol{P} \neq \boldsymbol{Q}$. We seek to make the best (lowest) possible $\boldsymbol{P} \neq \boldsymbol{Q}$ solution to have worse (higher) objective value than the worst (highest) $\boldsymbol{P} = \boldsymbol{Q}$ solution. When $\boldsymbol{P} = \boldsymbol{Q}$, the highest possible objective value is

$$d^2\alpha^2 + \lambda d\alpha + \lambda d\alpha - \lambda^2 d$$

and similarly, the lowest possible objective value when $\boldsymbol{P} \neq \boldsymbol{Q}$ is

$$-d^2\alpha^2 - \lambda d\alpha - \lambda d\alpha - \lambda^2 d + \lambda^2$$

where the final term is due to the fact that at least one entry of $\boldsymbol{P}\boldsymbol{Q}^\top$ must be 0. With some algebra, it can be seen that $\lambda > 5d\alpha$ is sufficient to guarantee that all $\boldsymbol{P} = \boldsymbol{Q}$ solutions are superior to all $\boldsymbol{P} \neq \boldsymbol{Q}$ solutions.

Now when $\boldsymbol{P} = \boldsymbol{Q}$, all frivolous terms reduce to constants and we are left with the QAP objective:

$$\min_{\boldsymbol{P}} \langle \boldsymbol{P}\boldsymbol{C}\boldsymbol{P}^\top, \boldsymbol{D} \rangle - \lambda \langle \boldsymbol{P}^\top \boldsymbol{P}, \boldsymbol{C} \rangle + \lambda \langle \boldsymbol{P}\boldsymbol{P}^\top, \boldsymbol{D} \rangle - \lambda^2 \operatorname{tr}(\boldsymbol{P}\boldsymbol{P}^\top)$$

$$= -\lambda \operatorname{tr}(\boldsymbol{C}) + \lambda \operatorname{tr}(\boldsymbol{D}) - \lambda^2 d + \min_{\boldsymbol{P}} \langle \boldsymbol{P}\boldsymbol{C}\boldsymbol{P}^\top, \boldsymbol{D} \rangle$$

completing the reduction. QAP is known to be strongly NP-hard [Sahni and Gonzalez, 1976], thus completing the proof. □