Structured, Constrained and Creative Learning

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Marin Vlastelica
aus Split, Kroatien

Tübingen
2024

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	22.11.2024
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter:	Prof. Dr. Georg Martius
2. Berichterstatter:	Prof. Dr. Peter Dayan
3. Berichterstatter:	Prof. Dr. Mathias Niepert

To the sunshine of my life, my daughter Marina Valentina.

If I have seen further it is by standing on the shoulders of Giants.

Isaac Newton

Acknowledgments

For myself, and I believe many others, the PhD journey is truly transformative. I have been lucky to have been able to spend large amounts of time exploring exciting topics and just following my curiosity in a very engaging environment, the Max Planck Institute for Intelligent Systems. The atmosphere surrounding the institute has always been and will continue to be around doing impactful research in machine learning. This thesis would not have reached its current state without the influence of countless smart people at the institute and beyond. While I would like to acknowledge everyone who has had a profound impact on my journey, it is impossible to do so. As a result, some omissions are inevitable, but this does not diminish the importance of those who have been left out.

First and foremost, I thank my supervisor, Prof. Georg Martius, who is a shining example of staying curious and finding interesting things to do in any possible topic. His eagerness to learn about new areas and contribute impresses me to this day, which goes as well to the benefit of his students that are free to pursue their research passions. Additionally, the sheer goodwill and warm-heartedness that surrounds him should not be underestimated.

In addition to having a brilliant supervisor, I had the privilege of having a brilliant advisory committee. I am thankful to Prof. Peter Dayan and Michael Muehlebach for always giving constructive feedback that was on point, but also for the support beyond the research projects even outside the official meetings. Both consistently had an aura of support and mentorship around them that encouraged me to push forward.

Since I began my PhD endeavor, the Autonomous Learning group has evolved into a formidable research entity, to the point where our meeting room has become noticeably crowded. The free exchange of ideas within the group has been crucial for my personal development as a researcher and for the evolution of my own ideas. Ideas do not emerge in isolation, just as people are not shaped in solitude—we are all products of our experiences. I am grateful to the entire group for their support, engaging discussions, and enjoyable moments we shared. The PhD journey would have been significantly more challenging without these individuals around me. I would particularly like to extend my thanks to Michal Rolínek, with whom I had the privilege of working while he was still part of the group as a Postdoc.

The impact of his ideas and input has been substantial in my work and on me, but also just as a research role model, he is definitely one of the "coolest" mathematicians that I know. I say one of, because the list wouldn't be complete without Vít Musil, who, except for being a brilliant researcher, is also a decent drummer. We had really good times rocking out together at the institute to blow off some steam, these jam sessions have been weird, but an absolute joy – I hope we get to play together in a proper band one day.

I would like to express my gratitude to Sebastian Blaes for his friendship and understanding. Together, we have delved into deep topics, endured tough times, and collaborated on some exciting projects. Sebastian, you are the embodiment of stoicism and goodwill. I hope that some of your positive energy has transferred to me through our interactions. The list of the most capable PhD students I know would not be complete without mentioning Nico Guertler, who, it turns out, is also excellent with children. Thank you, Nico, for showing the robots to my daughter; she still fondly recalls her visit to Tübingen. Talking about sources of good mood, after Georg definitely comes Christian Gumbsch, with every presentation being more entertaining than the last one – that's a hard bar to beat. I would further like to thank Pavel Kolev for working with me on diverse offline imitation, your ideas have been cardinal in shaping the project, and you were always available for discussions. I thank Marco Bagatella, Núria Armengol Urpí and Anselm Paulus for giving this thesis a read and for their suggestions.

Not many PhD students had the opportunity to supervise interns that surpass expectations significantly. I am thankful to Chenhao Li and Jin Cheng for a seamless supervision experience. Chenhao and Jin are now successful PhD students in Zurich, and I believe they will become very impactful researchers. Both displayed a healthy dose of stubbornness and questioning that is a trait all successful scientists, it wasn't enough to just say something to Jin and Chenhao, one had to always prove that it makes sense, which made their internships a playful game of back and forth.

Beyond my research at the institute, I had the privilege of doing several research internships. For my time at Amazon, I thank Gyuri Szarvas and his team for hosting me and Patrick Ernst for valuable guidance on the research project. In this period I learned a lot about Natural Language Processing and the challenges that the models are facing – who would have known that shortly after our work on RL for dialogue systems large-scale fine-tuning with RL would become a thing.

I was also fortunate to have been hosted by the Structured Intelligence team at DeepMind, London. My time there was an amazing experience of constant learning and exchanging of ideas with other scientists. I want to thank my host, Kimberley Stachenfeld for supporting me throughout as well as Peter Battaglia for the constructive feedback and also Alvaro Sanchez-Gomez, Jessica Hamrick and Charlie Nash for useful input throughout the internship. Beyond my team I had the privilege of having coffee with many impactful scientists, I

am further impressed that researchers with such established careers would be ready to separate so much time to discuss with me. I had insightful conversations with more people than I can count, such as Nando de Freitas, Conor Durkan, Oriol Vinyals, George Papamakarios, Markus Wulfmeier, Tim Roecktaschel, Charles Blundell, Petar Veličković, Jovana Mitrović, Danilo Rezende, Martin Riedmiller, Brendan O'Donoghue. In particular I need to emphasize Danilo, the conversation that we had about generative models and variational inference is one of the deepest that I had on that topic since I entered the field of ML. I thank Jovana Mitrović, Matko Bošnjak, Nemanja Rakićević, Aleksandar Stanić, Joy Ortiz and Gianluca Scarpellini for their friendship, which really made my stay in London great joy, as well as all the interns from my cohort – we had an amazing time either jamming in the music room, playing foosball or having breakfast club discussions. Finally, I want to give special thanks to Prof. Arnaud Doucet, I really enjoyed working with you and you will remain a role model for me in the future, few people can match your intellectual vigour combined with kindness and willingness to assist others. This has really impressed me far and beyond. I miss London greatly, and I hope we will meet again soon.

I further thank Ali Thabet and his team at Meta GenAI in Zurich, Switzerland for providing me with an opportunity to work on large scale generative models, something that is not easily feasible in an academic setting. I further had the pleasure of fruitful discussions with many exceptional researchers at Meta such as Brandon Amos, Ricky Chen, Yaron Lipman and others.

Last, but not least, I want to give special thanks to my family for supporting my decisions in pursuing the PhD. Becoming a parent during my doctoral studies was difficult, striking the balance between career pursuit and allocating time for family is an impossible optimization problem, the time is never enough, but they were always in my thoughts. I want to thank my mother Asja, who heroically raised me as a single mother in a very challenging setting, she is always a reminder for me of courage, resilience and standing up for one's right. I thank my grandparents Mile and Miro for being there for me when my parents couldn't. I thank my partner Maren for her patience, understanding and for being a wonderful mother to our beloved daughter. The strenuous relationship with my father has marked a large chunk of my life. Regrettably, he passed away before seeing the culmination of my work as a PhD student or meeting his granddaughter. I would have liked for him to have experienced this, and I am sure he would have been proud in his own way. I thank him for teaching me how important the relationship between a father and his child is, I think about this constantly and I am a better parent for it. Finally, I thank my daughter Marina Valentina for blessing her mother and me with her presence in this world, every minute we spend together is a source of great joy and strength.

Abstract

The field of deep learning has undergone a substantial transformation in recent years, with improvements being made on the level of model architectures which through specific inductive biases thanks to their functional form give powerful generalization capabilities to neural networks. Due to the substantial amount of data and computing resources, they have received wide-spread adoption with impressive performance on a wide range of tasks such as language and image generation. Furthermore, deep learning models have shown the capability to even surpass human players in games that require higher-level reasoning capabilities. Despite this, there are many outstanding issues in important applications. Various problems require reasoning in spaces that are rich in structure and are highly constrained, a simple example is the planning problem which doesn't require exhaustive search of all possible sequences of visited states, but rather a search over valid paths, which form a possibly small subset. This indicates that the generalization capability of learning systems can be enhanced by incorporating structured prediction and constraints. Conversely, it is imperative for these systems to demonstrate flexibility and innovation, which is somewhat at odds with constraining their predictions. Creativity and adaptability are, indeed, quintessential elements of intelligence, they empower systems to formulate novel solutions and adapt efficiently to new or unforeseen circumstances.

In this work, we attempt to introduce structure and constraints such that intelligent systems can provide us with plausible suggestions in order to make critical decisions, while still remaining adaptable. In the first part we consider adding structure by means of combinatorial implicit layers, showing that this is a critical inductive bias for dealing with problems containing "hidden" combinatorial structure. To achieve this, it is necessary to employ a parameter update direction distinct from the traditional gradient descent direction, as the objective function becomes piecewise-constant given the finite nature of the feasible set. It turns out that there is an efficient way to compute an informative update direction, which is a derivative of a piecewise-affine interpolation of the original piecewise-constant function. This involves only one additional call to the combinatorial solver on the backward pass. We name this method *Blackbox Backpropagation* (BBBP), to highlight the fact that we don't make use of any information about the underlying combinatorial problem, or try to unroll it and differentiate. An alternative to this will be treating the implicit layer as a negative identity block on the backward pass, which is connected intimately to the initial piecewise-affine interpolation approach. As we shall see, many practical applications of interest contain combinatorial

structure, some of the settings that we shall explore are planning from images and optimizing rank-based metrics.

In the second part, we shift our attention to an important component of intelligent systems, sequential decision-making. We examine both online scenarios, where a policy is allowed to collect samples from the environment in order to improve, and the offline setting, where one fixed dataset is given and the task is to extract an optimal policy.

In the online setting, employing a model-based approach emerges as particularly advantageous for introducing safety constraints and risk-averseness through a trajectory optimization method that makes use of epistemic and aleatoric uncertainty separation. For this case, the policy is given implicitly and is solving an optimization problem to produce an action, the benefit of this being that we only need to learn model dynamics, which can be done from arbitrary data. By making use of epistemic bonus and a risk penalty, this method strikes a balance between exploration and exploitation, facilitating efficient learning of the model.

In contrast, the offline setting is interesting from the perspective that it resembles much of what we see today, large models making use of data collected from various sources. Up until now, the focus was on extracting policies from the data that do not hallucinate and generalize well out-of-distribution. However, the diversity of such policies has been neglected, and we argue that intelligent systems need to be creative problem solvers in order to be useful, since many problems have multiple solutions, some of which might be more or less robust. This is in particular true when demonstrations are coming from external sources, such as a human. We frame this problem as an imitation learning problem, where we have access to offline actions which are not coming from expert demonstrations, but rather from an arbitrary behavior policy. The expert is only allowed to demonstrate by showing us a sequences of states. This scenario necessitates a different strategy, as the absence of direct guidance from optimal actions complicates the learning process. Concretely, the formulation involves maximizing a diversity objective subject to f-divergence constraints on state occupancy distributions. By clever use of Fenchel duality, we solve a dual problem to obtain importance ratios that allow us to estimate expectations with respect to optimal policy state-action occupancies. This we validate in a sim-to-real quadruped benchmark.

Finally, we confirm that introducing these types of structure and constraints is necessary for data-efficient generalization and robust decision making.

Abstrakt

Das Gebiet des Deep Learning hat in den letzten Jahren einen erheblichen Wandel durchlaufen. Verbesserungen auf der Ebene der Modellarchitekturen haben durch ihre spezifischen induktiven Verzerrungen und dank ihrer funktionalen Form leistungsstarke Generalisierungsfähigkeiten für neuronale Netze ermöglicht. Aufgrund der beträchtlichen Menge an Daten und Rechenressourcen haben sie eine weite Verbreitung gefunden und zeigen beeindruckende Leistungen bei einer Vielzahl von Aufgaben wie Sprach- und Bildgenerierung. Darüber hinaus haben Deep-Learning-Modelle die Fähigkeit gezeigt, selbst in Spielen, die komplexes Denkvermögen erfordern, menschliche Spieler zu übertreffen. Trotzdem gibt es in wichtigen Anwendungen viele ungelöste Probleme. Verschiedene Probleme erfordern ein Denken in Räumen, die reich an Struktur sind und gleichzeitig stark eingeschränkt. Ein einfaches Beispiel ist das Planungsproblem, das keine erschöpfende Suche nach allen möglichen Abfolgen erreichbarer Zustände erfordert, sondern eine Suche nach gültigen Pfaden, die möglicherweise nur eine kleine Teilmenge bilden. Dies deutet darauf hin, dass die Generalisierungsfähigkeit von Lernsystemen durch die Einbeziehung von strukturierter Vorhersage und Einschränkungen verbessert werden kann. Im Gegensatz dazu ist es für diese Systeme unerlässlich, Flexibilität und Innovation zu zeigen, was im Widerspruch zu Einschränkungen ihrer Vorhersagen steht. Kreativität und Anpassungsfähigkeit sind in der Tat wesentliche Elemente der Intelligenz. Sie befähigen Systeme, neue Lösungen zu formulieren und sich effizient an neue oder unvorhergesehene Umstände anzupassen. In dieser Arbeit versuchen wir, Struktur und Einschränkungen einzuführen, damit intelligente Systeme uns plausible Vorschläge zur Entscheidungsfindung machen können, während sie dennoch anpassungsfähig bleiben. Im ersten Teil betrachten wir das Hinzufügen von Struktur durch kombinatorische implizite Schichten und zeigen, dass dies eine wichtige induktive Verzerrung für die Bewältigung von Problemen mit "verborgener" kombinatorischer Struktur ist. Um dies zu erreichen, ist es notwendig, eine Parameter-Aktualisierungsrichtung zu verwenden, die sich von der traditionellen Gradientenabstiegsrichtung unterscheidet, da die Zielfunktion aufgrund der endlichen Natur der zulässigen Menge stückweise konstant wird. Es stellt sich heraus, dass es einen effizienten Weg gibt, eine informative Aktualisierungsrichtung zu berechnen, die eine Ableitung einer stückweise affinen Interpolation der ursprünglichen stückweise konstanten Funktion ist. Dies erfordert nur einen zusätzlichen Aufruf des kombinatorischen Solvers im Backpropagation. Wir nennen diese Methode *Blackbox Backpropagation* (BBBP), um hervorzuheben, dass wir keine Informationen über das zugrunde liegende kombinatorische Problem verwenden oder versuchen, es aufzurollen und zu differenzieren.

Eine Alternative dazu besteht darin, die implizite Schicht im Rückwärtsgang als negativen Identitätsblock zu behandeln, was eng mit dem ursprünglichen Ansatz der stückweise afÏnen Interpolation verbunden ist. Wie wir sehen werden, enthalten viele praktische Anwendungen von Interesse kombinatorische Strukturen. Einige der Problemstellungen, die wir untersuchen werden, sind die Planung von Bildern und die Optimierung von Rang-basierten Metriken. Im zweiten Teil richten wir unser Augenmerk auf eine wichtige Komponente intelligenter Systeme, die sequentielle Entscheidungsfindung. Wir untersuchen sowohl Online-Szenarien, in denen eine Policy mit der Umgebung interagieren darf, um ihre Verhaltensweise zu verbessern, als auch das Offline-Setting, in dem ein fester Datensatz gegeben ist und die Aufgabe darin besteht, eine optimale Policy zu extrahieren. Im Online-Setting erweist sich ein modellbasierterAnsatz als besonders vorteilhaft, um Sicherheitsbeschränkungen und Risikoaversion durch ein Trajektorienoptimierungsverfahren einzuführen, das die Trennung von epistemischer und aleatorischer Unsicherheit nutzt. Für diesen Fall wird die Policy implizit gegeben und löst ein Optimierungsproblem, um eine Aktion zu erzeugen, wobei der Vorteil darin besteht, dass wir nur die Modelldynamik lernen müssen, was mit beliebigen Daten erfolgen kann. Durch die Nutzung von epistemischen Boni und einer Risikostrafe findet diese Methode ein Gleichgewicht zwischen Exploration und Ausbeutung und erleichtert so das effiziente Lernen des Modells.

Im Gegensatz dazu ist das Offline-Setting interessant aus der Perspektive, dass es dem ähnelt, was wir heute sehen, große Modelle, die Daten aus verschiedenen Quellen nutzen. Bis jetzt lag der Fokus darauf, Policies aus den Daten zu extrahieren, die nicht halluzinieren und gut außerhalb der Verteilung generalisieren. Jedoch wurde die Vielfalt solcher Policys vernachlässigt, und wir argumentieren, dass intelligente Systeme kreative Problemlöser sein müssen, um nützlich zu sein, da viele Probleme mehrere Lösungen haben, von denen einige mehr oder weniger robust sein könnten. Dies gilt insbesondere, wenn Demonstrationen von externen Quellen wie einem Menschen stammen. Wir bestimmen dieses Problem als die Nachahmung eines Lernproblems, bei dem wir Zugang zu Offline-Aktionen haben, die nicht von Expertendemonstrationen stammen, sondern von einer beliebigen Policy. Der Experte darf uns nur Sequenzen von Zuständen demonstrieren. Dies erfordert eine andere Strategie, da das Fehlen direkter Anweisungen von optimalen Aktionen den Lernprozess erschwert. Die Formulierung beinhaltet die Maximierung eines Diversitätsziels unter f-Divergenz Einschränkungen für Zustandsbelegungsverteilungen. Durch geschickte Nutzung der Fenchel-Dualität lösen wir ein duales Problem, um Importance Ratios zu erhalten, die es uns ermöglichen, Erwartungen in Bezug auf optimale Policy-Zustands-Aktionsbelegungen zu schätzen. Dies validieren wir in einem Sim-to-Real Quadruped-Benchmark.

Abschließend bestätigen wir, dass die Einführung dieser Arten von Struktur und Einschränkungen für dateneffiziente Generalisierung und robuste Entscheidungsfindung notwendig ist.

Contents

List of Figures

Glossary

Reinforcement Learning Notation

Q State-action value function.
V State value function.
$\mathcal{P}$ Transition density.
$\mathcal{A}$ Action space.
$\mathcal{J}(\pi)$ Epected return for policy π.
$\mathcal{M}$ Markov Decision Process.
$\mathcal{S}$ State space.
$\mathcal{T}$ Bellman operator.
π Policy.
$d^\pi(s)$ State occupancy distribution of π.
$d^\pi(s, a)$ State-action occupancy distribution of π.

Mathematical Notation

$\mathrm{D}_f(p \,\|\, q)$ f-divergence between distributions p and q.
$\Delta(\mathcal{X})$ Probability simplex.
$\mathbb{E}$ Expectation operator.
Π_n Set of all permutations of n elements.
Var Variance operator.
ℓ Objective function (loss).
$\langle x, y \rangle$ Dot product of vectors x and y.
$\mathbb{N}$ Set of all natural numbers.
$\mathbb{R}$ Set of all real numbers.
$\mathbf{H}$ Hessian matrix.
$\mathbf{I}$ Identity matrix.
$\mathbf{J}_f$ Jacobian matrix for function f.
$\mathbf{X}, \mathbf{Y}$ Matrices.
$\mathbf{X}^\top$ Transpose of matrix $\mathbf{X}$.
$\mathbf{X}^{-1}$ Inverse of matrix $\mathbf{X}$.
$\mathbf{X}^{\cdot,j}$ j-th column of matrix $\boldsymbol{X}$.
$\mathbf{X}^{i,\cdot}$ i-th row of matrix $\boldsymbol{X}$.
∇^2 Hessian operator.
∇ Gradient operator.
$\odot$ Element-wise (Hadammard) product.
$\partial_i f$ Partial derivative of f w.r.t. i-th argument.
π Permutation.
$\|x\|_p$ p-norm of vector x.
$\|x\|$ L_2 norm of vector x.
f^* Fenchel conjugate (Legendre transform) of function f .
$p(x)$ Probability distribution of random variable x.

Acronyms

A2C *Advantage Actor-Critic*. xxi, 34
A3C *Asynchronous Advantage Actor-Critic*. xxi
ADP *Approximate Dynamic Programming*. xxi
ANN *Artificial Neural Network*. xxi

BB *Black Box*. xxi
BBBP *Blackbox Backpropagation*. iv, vi, xxi, 55, 56, 59
BC *Behavior Cloning*. xxi, 86
BO *Bayesian Optimization*. xxi

CEM *Cross-Entropy Method*. xxi, 4, 7, 39, 40, 96, 103, 104, 138, 139, 146
CMA *Covariance Matrix Adaptation*. xxi
CMA-ES *Covariance Matrix Adaptation Evolution Strategy*. xxi
CMDP *Constrained Markov Decision Process*. xxi, 110, 133, 134
CNN *Convolutional Neural Network*. xxi, 1

DDM *Denoising Diffusion Model*. xxi, 6, 7, 137, 138, 141, 142
DDPG *Deep Deterministic Policy Gradient*. xxi
DEM *Deep Equilibrium Model*. xxi, 24
DICE *Distribution Correction Estimation*. xxi, 112
DOI *Diverse Offline Imitation*. xv, xvii, xxi, 110, 111, 119–121, 125, 131, 135, 148, 215, 223, 225–228, 230
DoMiNiC *Diversity Optimization under Multiple Near-optimal Constraints*. xv, xxi, 133–135
DOMiNO *Diversity Optimization Maintaining Near Optimality*. xxi, 133
DP *Dynamic Programming*. xxi, 34, 37, 38
DPO *Direct Preference Optimization*. xxi, 3
DQN *Deep Q-Network*. xxi
DrAC *Data-Regularized Actor Critic*. xiii, xxi, 86, 89–92
DRL *Deep Reinforcement Learning*. xxi

ES *Evolution Strategy*. xxi

GA *Genetic Algorithm*. xxi
GAIL *Generative Adversarial Imitation Learning*. xxi, 130
GAN *Generative Adversarial Network*. xxi
GNN *Graph Neural Network*. xxi, 1, 137, 146

Id *Identity*. xii, xxi, 55–57, 59
IP *Integer Program*. xxi, 16, 17

KKT *Karush-Kuhn-Tucker*. xxi, 21

LLM *Large Language Model*. xxi, 3, 33
LP *Linear Program*. xxi, 9
LQR *Linear Quadratic Regulator*. xxi, 37
LSTM *Long-short Term Memory*. xxi, 2

MAML *Model-Agnostic Meta-Learning*. xxi, 23
MBRL *Model-Based Reinforcement Learning*. xxi, 97
MC *Monte Carlo*. xxi, 40, 124
MCMC *Markov chain Monte Carlo*. xxi, 6
MCTS *Monte Carlo Tree Search*. xxi
MDP *Markov Decision Process*. xxi, 26, 80, 81, 111, 112, 133
MILP *Mixed Integer Linear Program*. xxi
MIP *Mixed Integer Program*. xxi, 16, 38
MLE *Maximumum Likelihood Estimation*. xxi, 39, 146

MPC *Model Predictive Control*. xxi, 4, 9, 36–39, 95–97
MPPI *Model Predictive Path Integral Control*. xxi, 4

NAP *Neuro Algorithmic Policy*. xiii, xvi, xxi, 84, 86–92, 199, 201–203

ODE *Ordinary Differential Equation*. xxi

PETS *Particle Ensemble Trajectory Sampling*. xiv, xxi, 96–98, 104, 105, 107, 146
PETSUS *Probabilistic Ensembles with Trajectory Sampling and Uncertainty Separation*. xiv, xxi, 96, 100, 102, 107
POMDP *Partially Observable Markov Decision Process*. xxi
PPO *Proximal Policy Optimization*. xiii, xxi, 3, 34, 89–91
PSO *Particle Swarm Optimization*. xxi

QP *Quadratic Program*. xxi, 38

RaMBO *Rank Metric Blackbox Optimization*. xxi, 66, 67, 70–77
RAZER *Risk-Averse Zero-Order Trajectory Optimization*. xiv, xxi, 96, 102, 104–107, 205, 207, 210
RL *Reinforcement Learning*. xxi, 3, 9, 26, 31, 37, 80, 109–112, 134
RLHF *Reinforcement Learning from Human Feedback*. xxi, 3

SAC *Soft Actor-Critic*. xxi, 34
SARSA *State Action Reward State Action*. xxi, 31
SDE *Stochastic Differential Equation*. xxi, 2
SDP *Semidefinite Program*. xxi
SGD *Stochastic Gradient Descent*. xxi, 11
SOCP *Second-Order Cone Program*. xxi
SOTA *State-of-the-Art*. xxi
SP *Shortest Path Problem*. xxi

TD3 *Twin Delayed Deep Deterministic Policy Gradient*. xxi
TDSP *Travelling Salesman Problem*. xxi, 80, 86, 90, 145
TRPO *Trust Region Policy Optimization*. xxi
TSP *Travelling Salesman Problem*. xxi

VDW *Van der Waalse*. xxi, 6, 135

WASABI *Wasserstein Adversarial Behavior Imitation*. xv, xxi, 129–131, 135

Simplicity is the ultimate sophistication.

Leonardo da Vinci

0 Introduction

Deep learning has seen great advances over the past decade, making significant improvements in multiple fields such as computer vision, natural language processing, and reinforcement learning. Learning a hierarchy of representations has proven to be a powerful tool for solving complex tasks as well as generalization. We have seen a range of impressive applications - protein folding [149], achieving super-human performance in games such as Go [298, 296, 297], StarCraft II [328], and Dota 2 [31], generating human-like natural language while also being able to perform simple reasoning tasks [149], realistic image [83] and video generation [20], learning control policies for robots from demonstrations [44, 43]. All of these successes seem to hinge on the powerful learning capabilities of neural networks in their various instantiations.

Architectural Developments

One axis of improvements that has been important to achieve these impressive feats is that of architectural modifications, which is closely connected to the concept of inductive bias. These improvements change the way the prediction is computed, effectively changing the hypothesis space of the model. Arguably, the most significant architectural modification that enables efficient training as well as achieves impressive results in terms of generalization nowadays is that of the attention mechanism [325], which has by now become ubiquitous in various transformer architectures and beyond [46, 55, 257]. Other examples include the introduction of *Graph Neural Networks* (GNNs) and their attention-augmented variations [24, 326]. These are in particular good in relational reasoning, but have also seen successes in algorithmic reasoning tasks [327]. An older example of an architecture modification that has had significant impact of the field is the *Convolutional Neural Network* (CNN) [178, 102], owing its motivation to visual invariance under translations. The U-Net architecture has been

a staple in the field of medical image segmentation [272], but also diffusion-based generative modelling [83]. The well-known *Long-short Term Memory* (LSTM) [140] recurrent architecture has long been the go-to approach for sequence modelling, where the key modification was the introduction of the gating mechanism and latent state to selectively remember or forget patterns. These modifications up till now are explicit, i.e. directly modifying the neural network with different differentiable transformations.

Architectural modifications need not be restricted such that we need to provide an explicit differentiable mapping to the output. Providing this mapping implicitly is one of the central topics of this thesis, as well as many concurrent works. Implicit layers have had a significant impact on the field, one recent famous example is generative modelling via solving an appropriate *Stochastic Differential Equation* (SDE) [306], here the output of the model approximates the so-called "score function" of noised marginal distributions at each time point of the stochastic process. Another well-known example is that of the Neural ODE [60], where the model predicts the infinitesimal change in the output over time, rather than directly predicting it. Further examples of implicit transformations include fixed-point layers [17], convex optimization layers [5]. A well-known example is that of meta-learning [97, 256], where the last "layer" is essentially an optimization procedure at training time. A challenge that is always being faced when defining implicit layers is *how to train them efficiently*, since gradient computation is either hard, the gradient is not informative or the layer is not differentiable.

When viewing the implicit layer as an optimization problem, one naive approach is to unroll the computation graph over optimization steps and propagate the gradient. This naturally does not scale, since it requires a considerable amount of memory and compute, on the order of the steps of the solver. If the optimization problem is continuous, one can make use of implicit differentiation [4, 256], in which case we avoid the unrolling of the optimization computation graph. This is however limited to the continuous setting, and there is a strong argument to be made for utilizing discrete solvers for combinatorial problems, which is the focus of Part I. An enduring belief is that reasoning is a combinatorial activity [24, 339, 92, 245], which is performed for humans in the abstract space of language. The brain itself is an analog signal processor which maps sensory input to abstract representations [30], on top of which we "reason". For humans, discreteness seems to be a necessary simplification in order to solve complex problems, how can we transfer this ability to reason in learned abstract discrete spaces to machines? Furthermore, many problems have specific solution sets that can reduce the search space greatly, leading to better generalization. Another motivation for looking at discrete solvers is of pure technical nature. If we want to introduce an implicit layer, it needs to be differentiable in order to train the model end-to-end. Since the gradient of such layers is necessarily 0 almost everywhere, owing to the piece-wise constant objective as a result of the discrete solution space of the solver, an alternative update rule needs to be introduced instead of the true gradient. Some approaches introduce different relaxations to

the solvers in order to make them differentiable [59, 367]. However, this is not desirable, since these relaxations can lead to suboptimal solutions [253]. To fill this gap, in Chapter 2 we introduce a method for computing an informative update to the parameters which is a gradient of a piece-wise affine interpolation of the original piece-wise constant function. This allows us to train hybrid architectures end-to-end, while treating the solver as a black box, the only requirement being that the solver's cost function is linear w.r.t. its input. By utilizing this framework we can successfully inject combinatorial inductive biases to any architecture, enabling structured constrained prediction in combinatorial spaces. We further validate this approach in relevant problem settings such as optimizing rank-based metrics (Chapter 3) and planning for sequential decision-making (Chapter 4).

Intelligent Systems in the Wild

Simple prediction however is not the only area that is of importance for intelligent systems. We are living in a time when these systems interact with the world and users, and are being used to solve difficult problems and drive decisions. Hence, it is important to consider their behaviors as part of dynamical systems – this is the focus of Part II. Here, safety plays a crucial role even beyond robotics applications. In the age of *Large Language Models* (LLMs), systems are regularly interacting with humans and solving tasks of increasing amount of complexity and importance. This has given rise to questions of safety, alignment and fairness of the solutions that are currently being deployed [295]. It is not a surprise that *Reinforcement Learning* (RL) has recently entered the world of large foundational models, through methods such as *Reinforcement Learning from Human Feedback* (RLHF) [52], *Direct Preference Optimization* (DPO) [252], PPO [331]. These approaches of fine-tuning and learning from demonstrations stem from fundamental ideas developed in the field, and can be viewed as offline model-free approaches to making decisions where extracting a value function which indicates how good an action is in the long-term if we follow a particular policy plays a crucial role. The policy, as per the policy gradient theorem [312] can increase the likelihood of an action proportionally to the value function. However, for the case of learning from demonstrations, the reward function still needs to be learned if there are no optimal actions specified [137]. Even if we are able to learn this reward function, we are still left with the issue of robustness [332, 62], the demonstration that we are observing may not be the most robust one in terms of state visitation from the point of view of the agent [188]. This is particularly damaging in the offline setting, where the environment cannot be queried in order to evaluate the learned policy [153, 183].

A policy in this case may be given explicitly as a mapping from states to actions, or implicitly, as a receding horizon planning method. For the latter case, the policy is formulated implicitly by solving the arg min problem of optimizing over action sequences [65, 346, 345].

There have been many approaches that have looked at utilizing learned models for planning from both the reinforcement learning [65, 247, 345] and optimal control community [217, 347], mostly involving solving an optimization problem at each step. There are many benefits in working with model-based implicit policies versus explicit policies. For one, humans tend to have a better intuition on how to incorporate prior knowledge into a model rather than into a policy or value function. Secondly, a model allows us to reason about the state visitation distribution by means of policy rollouts. Thirdly, models are in essence task-agnostic and not bound to a value function, which increases their usability across different tasks. Whereas in traditional approaches that fall under the category of *Model Predictive Control* (MPC), arduous system identification procedures need to be done to frame the optimal control problem [198], learning approaches hold the promise of substantially automatizing the system identification part. The resulting models however are complex and approximate, meaning that solving the optimal control problem to optimality is difficult. This would normally involve "convexifying" the original problem if possible, which necessarily involves computing second-order derivatives. These approaches tend to be too slow and unflexible for general-purpose problem solving. Hence, a lot of methods utilize zero order solvers such as *Cross-Entropy Method* (CEM) [65, 247] or *Model Predictive Path Integral Control* (MPPI) [345]. By employing a zero-order optimizer, we lift the restriction of having model architectures that fit nicely into classical optimization techniques, which is one of the main hindrances of MPC methods. For example, Chua et al. [65] introduce a method relying on an ensemble of Gaussian dynamics models for sampling the resulting trajectory given an action sequence. However, although utilizing a sampling method for solving the optimal control problem, the characteristics of the resulting trajectory distributions have been disregarded. Yet, a lot of information is contained within the induced trajectory distribution, such as uncertainty of the model about the dynamics but also the inherent stochasticity of the environment. The former we call "epistemic" uncertainty and the latter "aleatoric". In Chapter 5 we propose a method for separating the uncertainties and making use of them for guiding the data collection process for faster learning, risk-averse planning and introducing probabilistic safety constraints. In the planning formulation, where we are required to sample n trajectories and evaluate them by a cost function, these feats arise naturally from augmenting the cost function adequately with uncertainty information. With this we can effectively balance the exploitation-exploration tradeoff in the online setting. In order to have a probabilistic constraint, we need to formulate a probability of entering the violation set, which would require a tractable joint likelihood estimate of the trajectories. However, probabilistic reasoning with tractable joint likelihood of the trajectories comes at a price of constraining the model class with which the dynamics can be modelled [268, 86, 237, 118]. As we shall see, even in the case of tractable likelihood estimates, estimating probabilities of constraint violation is highly non-trivial, and necessitates integration over the violation set. By settling for being able to

just sample, we enable the usage of a much richer model class, increasing the flexibility of the method.

A sampling-based planning method with a rich model class gives us two important things. Firstly, we may incorporate safety constraints or uncertainty information in the cost and change the behavior at inference time. Secondly, we are able to sample from a complex distribution over trajectories given action sequences, which ensures diversity in behavior. We have however not explicitly optimized for diversity in this case. For model-free methods diversity needs to be handled more explicitly. Indeed, many works have introduced information-theoretic approaches for maximizing diversity in the online setting [94, 309] or distances in trajectory statistics [359]. As we have highlighted before, diversity is important for creative problem solving, however, the resulting diverse behavior still needs to be reasonable by some metric. Zahavy et al. [359] pinpoint this quality-diversity tradeoff problem, and proposed a constrained problem, with constraints on the value of the learned policies to ensure that each skill has a guaranteed performance level. This is achieved by continuous adjustment of Lagrange multipliers based on constraint violation. However, the online setting is rather forgiving, since all estimates can be computed with samples from the current policy. The question remains if the quality-diversity tradeoff can be controlled in the offline setting of learning from demonstrations, in the absence of a reward function and expert actions. In Chapter 6 we frame this problem as diversity maximization under f-divergence constraints between state occupancy distributions of the policies and the demonstration. For the diversity objective, we pick the lower bound on the mutual information between states and a latent z variable, which leads to the introduction of a skill discriminator. We show how this problem, by clever use of Fenchel duality [95, 270], can be mapped to a dual problem of learning a value function, which maps back to the primal solution of state-action occupancy ratios. This will allow us to estimate all the necessary objectives offline, resulting in an algorithm that similarly balances Lagrange multipliers controlled by violation levels.

Further Work and Connections

In Part III we discuss further work that is related to the topics of this thesis. Particularly, an alternative way of learning from demonstrations without access to expert actions is taking an adversarial approach which treats the policy as the generator and learns a discriminator to distinguish between policy samples and demonstration samples, which is the topic of Chapter 7. Usage of the Wasserstein GAN loss [14] will prove to be useful in order to extract fine-grained rewards. This can again be seen as a distribution matching problem between the demonstration state occupancy and the policy state occupancy. Surprisingly, with a few imperfect demonstrations which are given by moving the robot unactuated through the air, the policy is able to extract agile movements. When applying domain randomization with

appropriate regularizers to the reward, such as torque and rotation penalties, these policies transfer well to the real system. In the diversity maximization setting, regularizers can be treated each as a separate constraint, and the constraint level can be tuned adequately for specific problems. Taking the approach from Zahavy et al. [359] and extending it to multiple cosntraints for the sim-to-real setting is the topic of Chapter 8. The distinguishing factor of these approaches is that, in comparison to Chapter 6, we are allowed online samples in simulation. Moreover, Chapter 8 relies on the ℓ_2 distance between expected successor representations [75] and the *Van der Waalse* (VDW) force, enabling the algorithm further control of the diversity objective. In Chapter 9 we take a step back and consider the general problem of minimizing cost functions in diverse ways. One can frame this problem as sampling from the Boltzmann distribution $\pi(x)$ specified by an energy function, and we assume that this distribution has multiple modes. However, for complex energy functions this is highly non-trivial, and it is well known that simulation-based methods such as *Markov chain Monte Carlo* (MCMC) struggle with long mixing times [88, 185, 7, 73] in high dimensions without a good proposal distribution. Additionally, in the absence of samples it is difficult to fit $\pi(x)$, instead we choose to sample from $\tilde{p}(x) \propto p(x)\pi(x)$, where $p(x)$ is a *Denoising Diffusion Model* (DDM) fitted on iterations of optimization algorithm that minimize a family of cost functions. Assuming access to the cost function, the reverse process can be guided at inference time by taking the gradient of $\mathbb{E}[\mathbf{c}(x_0)|x_t]$ and modifying the score function, resulting in samples from $\tilde{p}(x)$ which is analogous to classifier-free guidance [139]. This can as well be seen as a way of constraining the search space of the optimization problem for a specific energy function to be $p(x)$, and we expect that from the extrapolation capabilities of the DDM we are able to obtain diverse solutions by capturing modes of $\pi(x)$.

Overview of this Thesis

In summary, in Part I of this thesis we tackle the problem of imposing structure and constraints to the forward pass of a neural network by embedding a combinatorial solver and proposing an update rule that circumvents the zero-gradient issue, followed by applications in computer vision and planning. In Part II we propose ways of imposing constraints on policies in sequential decision-making processes for the offline and online setting, while dealing with the exploration problem and quality-diversity tradeoff. Finally, we outline further approaches in Part III that are related to these topics and are tackling them from a different angle. Here we list the main contributions of this thesis with the associated chapters and the papers that they are based on:

- *"Differentiation of Blackbox Combinatorial Solvers"* from Vlastelica et al. [334] (Chapter 2), a method for obtaining an informative gradient for combinatorial solvers that

optimize a linear cost function.

- *"Optimizing Rank-based Metrics via Blackbox Differentiation"* from Rolínek et al. [271] (Chapter 3), an application of blackbox differentiation to the problem of optimizing rank-based metrics.
- *"Neuro-algorithmic Policies Enable Fast Combinatorial Generalization"* from Vlastelica, Rolínek, and Martius [335] (Chapter 4), an imitation learning method that learns a policy containing a time-dependent shortest path planner in the latent space.
- *"Risk-Averse Zero-Order Trajectory Optimization"* from Vlastelica et al. [330] (Chapter 5), a method for risk-averse constrained planning with CEM and ensemble models.
- *"Diverse Offline Imitation Learning"* from Vlastelica et al. [332] (Chapter 6), a primal-dual constrained formulation for learning from demonstrations with a mutual information diversity objective.

In addition to these contributions, there are several works that have been published in the context of this thesis, but are not included in the main body of the thesis and are deferred to Part III. These include **(i)** an adversarial online learning method for learning from demonstrations [188], **(ii)** an online algorithm for extracting diverse policies under multiple constraints [62], **(iii)** a method for amortized sampling from a target Boltzmann distribution based on DDMs [333].

Statement of Contributions

Table 1: Contribution percentages based on a characterization of what constitutes research work by some measure.

Paper	Scientific Ideas	Implementation	Data Generation	Writing
Marin Vlastelica*, Anselm Paulus*, Vít Musil, Georg Martius, and Michal Rolínek. "*Differentiation of Blackbox Combinatorial Solvers*". International Conference on Learning Representations (ICLR), 2020.	30%	50%	40%	25%
Michal Rolínek, Vít Musil, Anselm Paulus, ***Marin Vlastelica***, Claudio Michaelis, and Georg Martius. "*Optimizing Rank-Based Metrics With Blackbox Differentiation*". IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020.	10%	30%	-	20%
Marin Vlastelica, Michal Rolínek, and Georg Martius. "*Neuro-algorithmic Policies enable Fast Combinatorial Generalization*". International Conference on Machine Learning (ICML), 2021.	80%	80%	100%	90%
Marin Vlastelica*, Sebastian Blaes*, Cristina Pinneri, and Georg Martius. "*Risk-Averse Zero-Order Trajectory Optimization*". Conference on Robot Learning (CoRL), 2021.	60%	40%	40%	60%
Marin Vlastelica, Jin Cheng, Georg Martius and Pavel Kolev. "*Offline Diversity Maximization under Imitation Constraints*". Reinforcement Learning Conference (RLC), 2024.	50%	90%	20%	70%
Subham Sekhar Sahoo*, Anselm Paulus*, ***Marin Vlastelica***, Vít Musil, Volodymyr Kuleshov, and Georg Martius. "*Backpropagation through Combinatorial Algorithms: Identity with Projection Works*". International Conference on Learning Representations (ICLR), 2022.	10%	20%	-	25%
Chenhao Li, ***Marin Vlastelica***, Sebastian Blaes, Jonas Frey, Felix Grimminger, and Georg Martius. "*Learning Agile Skills via Adversarial Imitation of Rough Partial Demonstrations*". Conference on Robot Learning (CoRL), 2023.	40%	5%	0%	40%
Jin Cheng, ***Marin Vlastelica***, Chenhao Li, Pavel Kolev and Georg Martius. "*Learning Diverse Skills for Local Navigation under Multi-constraint Optimality*". IEEE International Conference on Robotics and Automation (ICRA), 2023.	20%	5%	0%	25%
Marin Vlastelica, Tatiana López-Guevara, Kelsey Allen, Peter Battaglia, Arnaud Doucet, and Kimberley Stachenfeld. "*Diffusion Generative Inverse Design*". SPIGM Workshop @ International Conference on Machine Learning (ICML), 2023.	80%	100%	100%	80%

We feel free because we lack the very language to articulate our unfreedom.

Slavoj Žižek

1 Background

In this chapter we lay the foundations necessary for understanding the rest of this thesis by taking a journey through various optimization problems and techniques for their solution. We start off by the general formulation of a constrained optimization problem (Section 1.2) followed by the practice of optimizing deep neural networks in Section 1.2.1. After the basic concepts, we tackle the concept of duality (Section 1.2.2) and its appearance in *Linear Programs* (LPs). We then transition into the special case of combinatorial problems where the optimization domain is a discrete set (Section 1.2.4) and the nature of the loss landscape when we optimize a combinatorial problem, which is again a special case of the broader problem of differentiating through arg min (Section 1.3). For the last part, we define the field of RL (Section 1.4), which consists of advanced optimization techniques for dealing with sequential decision making problems, primarily dealing with the non-existence of a direct gradient of the objective. Interestingly, the duality concepts from Section 1.2.2 also appear in RL, leading to interesting primal-dual problem formulations that are relevant for this thesis. Finally, we connect the field of MPC with RL, and introduce a zero-order optimization method that we will make use of at a later point (Section 1.5.3).

1.1 A Word on Notation

In this thesis, in order to avoid cluttering the notation, we will sometimes omit the domain of a function, and assume that it is the whole space $\mathbb{R}^n$, we will oftentimes drop the domain of optimization if it is $\mathbb{R}^n$ or the probability simplex, which should be clear from the context. If a function is used in a context where the domain is not clear, we will explicitly state it, if it is used without arguments, then the function is treated an operator that acts on the whole space. Occasionally we will use the notation ∇f to denote the gradient of a function f with respect to its arguments, when there is no ambiguity. The notation $\nabla_x f$ will be used when we

want to explicitly state the gradient with respect to x. We adhere to the standard convention that all standalone vectors are considered to be column vectors. All functions are assumed to be from the family of continuous differentiable functions, unless stated otherwise. For the objective function of an optimization problem, we will use the notation ℓ, for the cost function we will use $\mathbf{c}$, and for the reward function we will use R. With $\langle x, y \rangle$ we denote the dot-product between two vectors, x and y, on the other hand, we write just xy for their element-wise product, which is short-form for the Hadammard product $x \odot y$. The same applies for function vectors, in particular, given probability densities $p(x)$ and $q(y|x)$ we have that their joint distribution can be written simply as $p \odot q$ in functional vector form. Furthermore, $\mathbb{E}_p[f(x)]$ may be written as $\langle p, f \rangle$. We extend these notions from vectors to matrices, for given matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, we will overload the vector notation $\langle \boldsymbol{A}, \boldsymbol{B} \rangle$ so that it denotes their "dot-product", moreover, $\boldsymbol{A} \odot \boldsymbol{B}$ denotes their element-wise Hadammard product, and $\boldsymbol{A} \odot v$ a Hadammard product between a matrix and a vector that broadcasts across columns for v and rows for $v^\top$ (this operation is also commutative).

1.2 Optimization

The field that subsumes most of the methods used in this thesis is optimization. In the most general form, we are interested in finding the minimum of an objective function ℓ with respect to some parameters θ, perhaps subject to a set of constraints.

Problem 1.2.1 (Constrained Optimization Problem)**.** A constrained optimization problem is defined as

$$\begin{aligned} \min_{x \in \mathbb{F}} \quad & \ell(x) \\ \text{s.t.} \quad & g_i(x) \leq 0, \quad i \in \mathcal{I}, \\ & h_j(x) = 0, \quad j \in \mathcal{E}, \end{aligned} \tag{1.1}$$

where $\mathcal{I}$ and $\mathcal{E}$ are the index sets of inequality and equality constraints, ℓ is the objective function and $\mathbb{F}$ is the optimization domain. With h_i and g_i we denote the respective equality and inequality constraint functions.

The feasible set of the optimization problem is defined as the set of all x that satisfy the constraints and lie in the optimization domain. Oftentimes, we drop the optimization domain from the notation, since it is assumed to be the whole space $\mathbb{R}^n$, and any further constraints are defined by the inequality and equality constraints. In this thesis we will encounter optimization problems where the optimization domain is a discrete set, or the probability simplex. In most of deep learning, the feasible set is $\mathbb{R}^n$, and the objective function is a continuous differentiable function $\ell : \mathbb{R}^n \to \mathbb{R}$ that works well for gradient-based optimization methods.

In its most basic form gradient descent is the update rule

$$\theta_{t+1} = \theta_t - \alpha \nabla \ell(\theta_t), \tag{1.2}$$

where α is the learning rate and θ_t function parameters. For training purposes in deep learning, we make use of minibatch *Stochastic Gradient Descent* (SGD), where we sample a subset of the data and compute the gradient on this subset. Denoting with x_i the data points and y_i regression targets, the minibatch SGD update rule is

$$\theta_{t+1} = \theta_t - \alpha \nabla \frac{1}{B} \sum_{i=1}^{B} \ell(\theta_t, x_i, y_i), \tag{1.3}$$

where B is the batch size. This can be interpreted as a Monte Carlo estimate of the expected loss gradient, where the expectation is taken over the data distribution $p(x, y)$. In practice however, more sophisticated optimization methods are used, such as Adam [158], RMSProp or AdaDelta [360] which make use of momentum and variance terms for adaptive learning rate scaling. Minibatch SGD circumvents the problem of computing the average on the whole dataset, which is expensive both in terms of computation and memory, since gradients need to be either stored or recomputed.

1.2.1 Optimization of Neural Networks

All of the methods used in this thesis are based on neural networks, which are a class of models that are inspired by the human brain. Except this rather vague analogy, it is better to view them as function approximators which are composed of a series of transformations. In the early days of deep learning, the transformations used were mostly affine followed by nonlinear activation functions. Nowadays, the architectures of deep neural networks transcend any such definition, with the introduction of convolutional layers, recurrent layers, attention mechanisms, residual connections and many more modifications. However, the basic principle of computing the gradient of the loss with respect to the parameters of the network remains the same, and it relies on the well-known chain rule of differentiation. Here we state the multi-variate version of the chain rule, which is the basis of backpropagation, for completeness.

Definition 1.2.1 (Chain Rule of Differentiation)**.** Let $f : \mathbb{R}^n \to \mathbb{R}^m$ and $g : \mathbb{R}^m \to \mathbb{R}^p$ be two functions, then the chain rule states that the Jacobian of the composition is given by

$$\nabla(g \circ f)(x) = \mathbf{J}_g(f(x))\mathbf{J}_f(x), \tag{1.4}$$

where $\mathbf{J}_f(\cdot)$ and $\mathbf{J}_g(\cdot)$ are the Jacobian matrices of f and g, respectively.

This naturally extends to the case where we have a composition of several functions, as is the case in neural networks. We may note one important thing, in order to compute the gradient with respect to x, we need to have the Jacobian of the function at that point. For the case where $p = 1$, the Jacobian $\mathbf{J}_g(f(x))$ is simply the gradient $\nabla g(f(x))^\top$, which effectively means that obtaining the gradient of the composition is equivalent to taking vector-Jacobian products, starting from last function transformation (output layer) and propagating the gradients backwards. Automatic differentiation libraries such as PyTorch [13] and JAX [42] allow us to compute these gradients efficiently by keeping track of the computation graph and applying the chain rule in reverse order. (i.e. starting from the loss and propagating the gradients backwards), also known as reverse-mode automatic differentiation. The algorithm that does this is known as backpropagation, and it is the cornerstone of training deep neural networks. The efficiency of backpropagation comes from the fact that the Jacobian computation is a result of elementary operations that re-use the function values computed during the forward pass. A good reference for understanding backpropagation in more detail is Goodfellow et al. [112]. An alternative would be forward-mode automatic differentiation that starts building up the Jacobian from the input, which would be more efficient for functions with large output dimensionality and a small input dimensionality (not the case in deep learning). The most intuitive way to understand the computation tradeoff between forward and reverse-mode automatic differentiation is to consider that the computation of matrix products vary in complexity based on the dimensions of the matrices. Consider matrices $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $\boldsymbol{B} \in \mathbb{R}^{n\times p}$ and $\boldsymbol{C} \in \mathbb{R}^{p\times 1}$, the complexity of computing the product $\boldsymbol{ABC}$ from right-to-left is $O(mn + np)$, on the other hand, computing the product from left-to-right is $O(mnp + mp)$, which is less efficient if $m \gg n, m \gg p$.

We can also notice that based on the chain rule, a Jacobian consisting of zeros in the chain results in a zero-gradient. Such is the case if there is a transformation that is piecewise-constant. One simple example of this are ReLU activations $f(x) = \max(0, x)$, which are zero for all negative inputs. Let us define a single layer of a neural network with $\boldsymbol{W}x + b$, $\boldsymbol{W}$ being the weight matrix and b the bias term, the i-th ReLU activation is therefore

$$a^i(x, \boldsymbol{W}, b) = \max(0, \boldsymbol{W}^{i,\cdot}x + b^i).$$

The expression for the i-th row of $\mathbf{J}_a(\boldsymbol{W}; x, b)$, which corresponds to the gradient w.r.t. the i-th row of $\boldsymbol{W}$ is

$$\mathbf{J}_a^{i,\cdot}(\boldsymbol{W}; x, b) = \mathrm{ReLU}(\boldsymbol{W}^{i,\cdot}x + b^i)x.$$

Once a ReLU unit is zero, the gradient w.r.t. the respective weight vector is zero, and a section of the network is effectively turned off, i.e. cannot be updated anymore. The issue that we will address in Chapter 2 is in this way similar, since using a combinatorial solver as network layer effectively results in a zero gradient of all of the parameters that precede it.

1.2.2 Duality in Constrained Problems

The concept of duality is a fundamental concept in constrained optimization which oftentimes enables us to transform a seemingly hard problem into a more manageable one. We start by observing that the general form from Problem 1.2.1 can be relaxed, by defining the Lagrangian function

Definition 1.2.2 (Lagrangian)**.** The Lagrangian function for the optimization problem Problem 1.2.1 is defined as

$$\mathcal{L}(x, \lambda, \nu) = \ell(x) + \sum_{i \in \mathcal{I}} \lambda_i g_i(x) + \sum_{j \in \mathcal{E}} \nu_j h_j(x), \tag{1.5}$$

where $\lambda \geq 0$ and ν are the Lagrange multipliers for the inequality and equality constraints, respectively.

A simple way to imitate the hard constraints is the maximization of the Lagrangian over the Lagrange multipliers, i.e. we can reformulate Problem 1.2.1 as

$$\inf_{x} \sup_{\lambda, \nu} \mathcal{L}(x, \lambda, \nu). \tag{1.6}$$

This is also known as the primal problem. The interpretation here is that for a particular x, we are maximizing the Lagrangian with respect to the Lagrange multipliers, meaning that if we have that a certain constraint is violated, then the inner maximization will result in an infinite value. Assume for example that $g_i(x) > 0$ (constraint is violated), then we simply set $\lambda_i = \infty$ in order to achieve infinite cost. This primal problem is, however, a hard problem to solve. We may instead consider the *unconstrained* minimization over x, which yields the Lagrange dual function.

Definition 1.2.3 (Lagrange Dual Function)**.** The Lagrange dual function is defined as

$$q(\lambda, \nu) = \inf_{x} \mathcal{L}(x, \lambda, \nu). \tag{1.7}$$

It is easy to check that Definition 1.2.3 is a concave function, since it is the pointwise infimum of affine functions. Furthermore, we have that the dual function is a lower bound on the optimal value of the primal problem since, by definition, at a feasible point $\tilde{x}$ the inequality constraints become ≤ 0. On the other hand, the Lagrangian for an arbitrary feasible point is the upper bound of the dual function, while lower bounding the original loss ℓ.

$$q(\lambda, \mu) \leq \mathcal{L}(\tilde{x}, \lambda, \mu) \leq \ell(\tilde{x}). \tag{1.8}$$

Now, we are equipped to define the Lagrange dual problem.

Definition 1.2.4 (Lagrange Dual Problem)**.** The Lagrange dual problem is defined as

$$\begin{aligned} &\sup_{\lambda,\nu} q(\lambda,\nu) = \sup_{\lambda,\nu} \inf_{x} \mathcal{L}(x,\lambda,\nu) \\ &\text{s.t.} \quad \lambda \geq 0. \end{aligned} \tag{1.9}$$

This dual problem is a convex optimization problem, even if the original primal problem is not convex, since The sup is taken over a concave objective and we have a convex inequality constraint. Indeed, eq. (1.9) and eq. (1.6) do not necessarily yield the same optimal value, for smooth $\mathcal{L}$, we have that the dual problem is a lower bound on the primal, also known as the weak duality theorem.

Theorem 1.2.1 (Weak Duality)**.** *Let x^* be a feasible solution to the primal problem and (λ^*, μ^*) be a feasible solution to the dual problem, then*

$$\ell(x^*) \geq q(\lambda^*, \mu^*). \tag{1.10}$$

The difference $\ell(x^*) - q(\lambda^*, \mu^*)$ is called the duality gap.

A stronger result is the strong duality theorem, which states that under certain conditions, such as convexity of the primal problem, and the existence of a feasible solution, the duality gap is zero.

Theorem 1.2.2 (Strong Duality)**.** *If the primal problem is convex and has a feasible solution, and the Lagrange dual problem has a feasible solution, then*

$$\ell(x^*) = q(\lambda^*, \mu^*). \tag{1.11}$$

Conjugate Functions

A useful tool when dealing with the dual problem and the Lagrange dual function is the Fenchel conjugate.

Definition 1.2.5. Let $f : \mathbb{R}^n \to \mathbb{R}$, then the Fenchel conjugate of f is defined as

$$f^*(y) = \sup_{x \in \mathbb{R}^n} \langle x, y \rangle - f(x) \tag{1.12}$$

An immidiate consequence of the Fenchel conjugate is that $f(x) + f^*(y) \geq \langle x, y \rangle$. The Fenchel conjugate is a useful tool in deriving the dual problem of a convex optimization problem, as it allows us to express the dual problem in terms of the conjugate of the objective function. In the special case if f is a convex function, then $f^{**} = f$, moreover we have

that $f(x) + f^*(y) = \langle x, y \rangle$. The graphical intuition of the conjugate function is that for a particular normal vector y of the hyperplane passing through the origin, it gives us the maximum distance of that hyperplane to the function f, this is achieved for a point x that lies on the hyperplane $\langle y, x \rangle = f^*(y)$, which is tangent to f. In the function f is not convex, the conjugate function is still convex, since it is a pointwise supremum of affine functions.

To better understand the conjugate function, it's useful to consider the epigraph of a function f.

Definition 1.2.6. The epigraph of a function $f : \mathbb{R}^n \to \mathbb{R}$ is defined as

$$\text{epi}(f) = \{(x, t) \mid f(x) \leq t\}. \tag{1.13}$$

It is easy to verify that the epigraph is a convex set if f is a convex function. We also have the useful property that $\text{epi}(f^{**})$ is the best convex approximation of $\text{epi}(f)$. If $f^{**} = f$ holds, then we also have the useful property

$$\nabla_x f^*(y) = \arg\max_x \langle x, y \rangle - f(x). \tag{1.14}$$

Furtheremore, for a minimizer y^* of f^*, we have that $x^* = \nabla f^*(y^*)$ is a minimizer of f.

1.2.3 Linear Programming

An important class of optimization problems that are the basis of understanding more complex optimization problems such as combinatorial problems are linear programs. There is a rich theory of linear programming, and it is a well studied field in optimization [41, 72, 34, 200].

Problem 1.2.2 (Linear Program)**.** A linear program is an optimization problem of the form

$$\begin{aligned} \min_x \quad & \langle w, x \rangle \\ \text{s.t.} \quad & \boldsymbol{A}x - b \geq 0 \end{aligned} \tag{1.15}$$

where w is the cost vector, $\boldsymbol{A}$ is the matrix of inequality constraints, b is the vector of inequality constraint values, and $x \in \mathbb{R}^d$ is the vector of optimization variables.

Problem 1.2.2 is a special case of Problem 1.2.1, where the objective function and the constraints are linear. The feasible set for Problem 1.2.2 is a polyhedron $P = \{x \mid \boldsymbol{A} - b \geq 0\}$, which is the intersection of half-spaces defined by the inequality constraints. The solution to a linear program is always at the vertex of the polyhedron, which is a consequence of the linearity of the objective function and the constraints. An equivalent problem to Problem 1.2.2

is the dual problem which is also a linear program, obtained by maximizing the Lagrangian of the primal problem over the Lagrange multipliers. For linear program, we have that strong duality holds, meaning that the optimal values of the primal and dual problem are equal and there is no duality gap. Based on strong duality, can derive the dual linear program (Problem 1.2.3).

Problem 1.2.3 (Dual Linear Program)**.** The dual linear program to Problem 1.2.2 is

$$\begin{aligned} \max_{\lambda} \quad & \langle b, \lambda \rangle \\ \text{s.t.} \quad & \boldsymbol{A}^\top \lambda = w, \\ & \lambda \geq 0. \end{aligned} \tag{1.16}$$

General methods for solving linear programs are the simplex method and the interior-point methods [226]. As we shall see in Section 1.2.4, there are great benefits in viewing combinatorial optimization problems through the lense of polyhedra in case of linear objective functions and arbitrary constraints over a finite optimization domain.

1.2.4 Combinatorial Optimization

The field of combinatorial optimization can very well be seen as a special case of optimization, where the feasible set is discrete. A natural framework for viewing combinatorial optimization problems is through *Integer Programs* (IPs).

Problem 1.2.4 (Mixed Integer Program)**.** A *Mixed Integer Program* (MIP) is an optimization problem of the form

$$\begin{aligned} \min_{x} \quad & \ell(x) \\ \text{s.t.} \quad & g_i(x) \leq 0, && i \in \mathcal{I}, \\ & h_j(x) = 0, && j \in \mathcal{E}, \\ & x_k \in \mathbb{Z}, && k \in I, \\ & x_l \in \mathbb{R}, && l \in J, \end{aligned} \tag{1.17}$$

where ℓ is the objective function, g_i and h_j are the inequality and equality constraints, I and J are the index sets of integer and real variables, respectively.

If we remove the real variables from the MIP, we obtain a pure IP.

Definition 1.2.7 (Integer Program)**.** An IP is a problem of the form

$$\begin{aligned} \min_{x} \quad & \ell(x) \\ \text{s.t.} \quad & g_i(x) \leq 0, \qquad i \in \mathcal{I}, \\ & h_j(x) = 0, \qquad j \in \mathcal{E}, \\ & x \in \mathbb{Z}^d, \end{aligned} \tag{1.18}$$

where ℓ is the objective function, g_i and h_j are the inequality and equality constraints.

In this thesis we won't deal with general IPs. The form of the combinatorial problem that is of particular interest in this thesis is for the case where the objective is linear as a function of a cost vector $w \in \mathbb{R}^d$, and the feasible set is an arbitrary discrete set Y, i.e.

$$\min_{y \in Y} \langle w, y \rangle. \tag{1.19}$$

The cost $\mathbf{c}(w, y) = \langle w, y \rangle$ is continuously differentiable with respect to w, moreover we have that at the solution point $\nabla_w \mathbf{c}(w, y^*) = y^*$. For convenience, we defined this arg min problem as the mapping $y : \mathbb{R}^n \to Y$. It is clear that any function of $y(w)$ is going to be piecewise constant with respect to w, since the solution is discrete, hence the true gradient is zero everywhere except on the jumps between solution surfaces.

As $w \to 0$, the solution $y(w)$ becomes less robust to perturbations of w. Indeed, within an ε-ball around $w = 0$, we can obtain any solution $y \in Y$. This requires special care, since any type of update rule for w can result in a drastic change in the solution y, causing instabilities in the optimization process.

For our purposes, we will be interested in the behavior of the mapping $y(w)$ as w changes. In order to gain intuition on this, it is useful to look at the combinatorial problem geometrically from a maximization perspective, which is equivalent to doing a substitution $w = -w'$, where w' is a utility(reward) vector. Suppose that the set $Y = \{y_i\}_{i=0}^n$ consists of n feasible solutions, now we ask ourselves the question of when does a specific y_1 maximize the utility $\mathrm{r}(w, y) = -\mathbf{c}(w, y) = \langle w', y \rangle$. The obvious answer is that y_i maximizes the utility if $\langle w', y_i \rangle \geq \langle w', y_j \rangle$ for all $j \neq i$. We may arrange this statement into a more compact matrix form for the case of $i = 1$,

$$\begin{bmatrix} \langle w', y_1 \rangle \\ \langle w', y_1 \rangle \\ \vdots \\ \langle w', y_1 \rangle \end{bmatrix} \geq \begin{bmatrix} \langle w', y_2 \rangle \\ \langle w', y_3 \rangle \\ \vdots \\ \langle w', y_n \rangle \end{bmatrix}. \tag{1.20}$$

Rearranging the terms, we obtain,

$$\begin{bmatrix} \langle w', y_1 - y_2 \rangle \\ \langle w', y_1 - y_3 \rangle \\ \vdots \\ \langle w', y_1 - y_n \rangle \end{bmatrix} \geq \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{1.21}$$

Each row of the matrix on the LHS, is a hyperplane with normal vector $y_1 - y_k$, passing through the origin. This system of linear inequalities define a specific polyhedron for each y_i, which we may call the solution polyhedron for y_1. In general, we specify a polyhedron P as the intersection of half-spaces, $P = \{x \mid \boldsymbol{A}x \leq b\}$, where $\boldsymbol{A}$ is a matrix of normal vectors of the half-spaces. A closed polyhedron is also called a polytope, and it is the convex hull of a finite number of points. From this perspective, it is not surprising that polytopes and polyhedra appear ubiquitously in combinatorial optimization problems, many solutions to such problems are in essence reductions to linear programming problems [226]. A useful analogy is that of countries on a map, where the borders are the hyperplanes defined by eq. (1.21), the polyhedra are the countries, and the points are the country capitals (the actual solutions y_i). A 2D example is provided in Figure 1.1, where the shaded regions with respective colors are the "countries" and the points are the "capitals".

Let us ruminate on the implications of the polyhedra defined by eq. (1.21) regarding any updates of the utility vector w'. Suppose that for a particular w', the solution $y(w')$ is y_1. In order to change the solution to y_k, we need to move w' to the polyhedron belonging to y_k. Moreover, the displacement of w' needs to be proportional to $\|w'\|$, since the distance to the closest polyhedron increases as $\|w'\|$ increases. In Chapter 2 we will see that we can formulate this displacement as a difference of solutions to the original problem and a problem with a perturbed cost vector. Perturbing the cost vector by some function $f(y)$ is the equivalent of shifting the bounds of the polyhedra, with the regions that relatively have more cost starting to shrink, allowing another solution to become optimal (the one that relatively incurs the least additional cost $f(y)$). This intuition is aligned with the method that we will propose in Chapter 2 for end-to-end training of neural networks with combinatorial layers.

Examples of Combinatorial Optimization Problems

For illustration purposes, we will now present some examples of combinatorial optimization problems that satisfy the form of eq. (1.19).

Example 1.2.1 (Shortest Path Problem)**.** We can embed the shortest path problem in the form of eq. (1.19) by defining the cost vector $w \in \mathbb{R}^n$ as the weights of n edges or n vertices of a graph, and the set $Y = \{0, 1\}^n$ acts as an indicator vector of the selected edges or vertices.

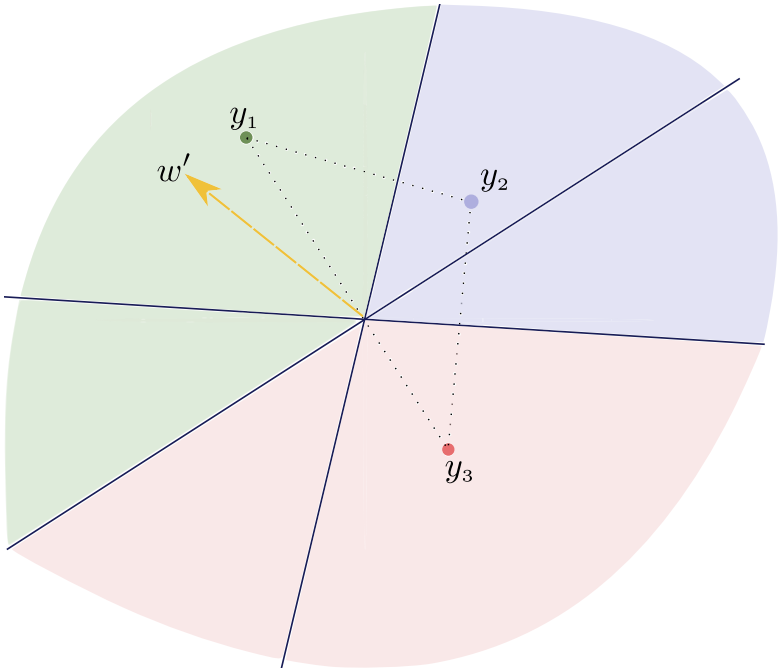


Figure 1.1: The points are discrete solutions y_i embedded in $\mathbb{R}^2$, the shaded regions are the polyhedra defined by the inequalities eq. (1.21). Once the utility vector w' is chosen, the solution $y(w')$ is the point that belongs to the shaded region where w' is located.

Moreover, the members of the set Y satisfy the constraints of the shortest path problem, i.e. the selected edges or vertices form a path from the source to the target node.

Example 1.2.2 (Travelling Salesman Problem)**.** Similarly to Example 1.2.1, we have $Y = \{0, 1\}^n$, where n is the number of edges or vertices in the graph. While the cost vector w is the weights of the edges or vertices, the constraints are that the selected edges or vertices form a cycle that visits each vertex exactly once, as per definition of the classical TSP problem.

Example 1.2.3 (Ranking Problem)**.** The ranking problem is defined as finding the permutation of a set of items that sorts them based on their cost in ascending order. The cost vector w is the weights of the items, and the set $Y = \Pi_n$ is the set of all permutations of the items.

It is not immediately obvious that Example 1.2.3 is an arg min problem which satisfies the form eq. (1.19), i.e. that the cost $\mathbf{c}(w, y)$ is linear. However, as we shall see, by a clever application of the permutation inequality we may formulate Example 1.2.3 into eq. (1.19). We will see several applications of this in Chapter 3 for optimizing rank-based metrics in computer vision problems.

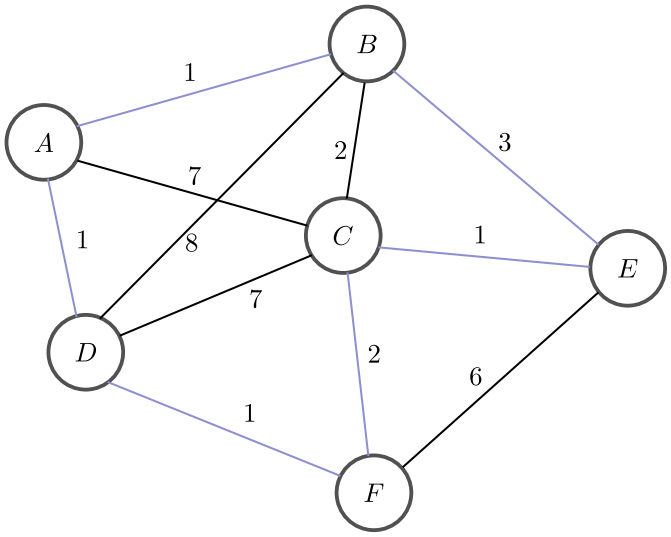


Figure 1.2: An example of a TSP problem, where the points are the cities and the lines are the connections between the cities. The goal is to find the shortest path that visits each city exactly once (highlighted in blue).

1.2.5 Convex Optimization

An important class of optimization problems are convex optimization problems due to their simplicity. Let us begin with an understanding of what it means for a function to be convex.

Proposition 1.2.3 (Line Segment Criterion)**.** *A function $f : \mathbb{R}^n \to \mathbb{R}$, $f \in C^1$ is convex if for all $x, y \in \mathbb{R}^n$ and $\lambda \in [0, 1]$ we have*

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y). \tag{1.22}$$

I.e. the line segment between any two points on the graph of the function lies above the graph. Further we have the criterion on the tangent lines of the function.

Proposition 1.2.4 (Tangent Line Criterion)**.** *A function $f : \mathbb{R}^n \to \mathbb{R}$, $f \in C^1$ is convex if for all $x, y \in \mathbb{R}^n$ and $\lambda \in [0, 1]$ we have*

$$f(y) \geq f(x) + \nabla f(x)^\top (y - x). \tag{1.23}$$

If the function is twice differentiable, then we have the second-order condition for convexity, which states that the Hessian of the function is positive semi-definite.

Proposition 1.2.5 (Hessian Criterion)**.** *$f : \mathbb{R}^n \to \mathbb{R}$, $f \in C^2$ is convex if for all $x \in \mathbb{R}^n$ we have*

$$\nabla^2 f(x) \succeq 0. \tag{1.24}$$

More general notions exist for the case where the function is not differentiable but only sub-differentiable. A convex problem is defined as an optimization problem where the objective function and the constraints are convex. For convex problems, the local minimum is also a global minimum, and strong duality (Theorem 1.2.2) holds under mild conditions, meaning that the optimal values of the primal and dual problem are equal. For the constrained case, we have necessary and sufficient conditions for optimality, the *Karush-Kuhn-Tucker* (KKT) conditions [230, 41].

For convex functions f, we also have that the conjugate function f^* is convex, and the conjugate of the conjugate is the original function, $f^{**} = f$.

Examples of Conjugate Functions

Here we list some examples of conjugate functions that are frequently used, amongst ones that are important for the work in this thesis. Their proofs can be found in standard convex analysis textbooks, such as Boyd and Vandenberghe [41].

Proposition 1.2.6. *The conjugate of the indicator function of a convex set C is the support function of the set C.*

Proposition 1.2.7. *The conjugate of the affine function $f(x) = \langle a, x \rangle + b$ is*

$$f^*(y) = \begin{cases} -b & \text{if } y = a \\ \infty & \text{otherwise.} \end{cases} \tag{1.25}$$

Proposition 1.2.8. *The conjugate of $f(x) = \|x\|_1$ is*

$$f^*(y) = \begin{cases} 0 & \|y\|_\infty \leq 1 \\ \infty & \text{otherwise.} \end{cases} \tag{1.26}$$

In fact, we can write this equivalently for any p-norm.

Proposition 1.2.9. *The conjugate of the $\|x\|_p$ norm is*

$$f^*(y) = \begin{cases} 0 & \|y\|_{p^*} \leq 1 \\ \infty & \text{otherwise,} \end{cases} \tag{1.27}$$

where $\|\cdot\|_{p^*}$ is the dual norm of $\|\cdot\|_p$.

Conjugate Functions of f-divergences

In machine learning, we care about similarity measures between probability distributions, a useful class of such measures are the $f-$divergences.

Definition 1.2.8. Let f be a convex function, then the f-divergence between two probability distributions of joint support p and q is defined as

$$\mathrm{D}_f(p||q) = \mathop{\mathbb{E}}_{q}\left[f\left(\frac{p(s)}{q(s)}\right)\right]. \tag{1.28}$$

It is not a surprise that f-divergence regularized objective are ubiquitous in machine learning, since they provide a way to introduce soft constraints into the optimization problem, for which the optimum can be found in closed form. The tool for achieving this is the conjugate function of the f-divergence.

Proposition 1.2.10 (f-Divergence Conjugate [221])**.** *The conjugate of the f-divergence $\mathrm{D}_f(x||q)$ at a function $y(s) = \frac{p(s)}{q(s)}$ is given by*

$$\mathrm{D}_{*,f}(y) = \mathop{\mathbb{E}}_{q}\left[f^*(y(s))\right]. \tag{1.29}$$

Furthermore, the minimizer satisfies

$$p^*(s) = q(s)\nabla f^*(y(s)). \tag{1.30}$$

An important special case of the f-divergence is the KL-divergence, which is the f-divergence with $f(x) = x\log x$. Notably, the reverse KL divergence has $f(x) = -\log(x)$.

Definition 1.2.9. The KL-divergence between two probability distributions p and q is defined as

$$\mathrm{D}_{\mathrm{KL}}(p\,\|\,q) = \mathop{\mathbb{E}}_{p}\left[\log\left(\frac{p(s)}{q(s)}\right)\right]. \tag{1.31}$$

A convenient fact regarding the KL-divergence is that its conjugate is obtainable in closed form, which does not in general hold for all f-divergences.

Proposition 1.2.11 (KL-Divergence Conjugate [221])**.** *Let $y(s) = \frac{p(s)}{q(s)}$. The conjugate of the* KL*-divergence is*

$$\mathrm{D}_{*,\mathrm{KL}}(y) = \log \mathop{\mathbb{E}}_{q}\left[e^{y(s)}\right]. \tag{1.32}$$

We will extensively make use of this when obtaining an offline skill discovery algorithm in Chapter 6. This closed form expression for the conjugate is only obtainable for the KL divergence from the class of f-divergences, which is also why we base our algorithm around this divergence in Chapter 6. For an illuminating perspective on the f-divergences and their use in reinforcement learning, we refer the reader to Belousov and Peters [29] and Nachum and Dai [221].

It is also worth considering the setting where we have the regularized optimization problem,

Problem 1.2.5 (KL-regularized Minimization)**.**

$$\inf_{p \in \Delta(\mathcal{S})} -\langle p, f \rangle + \beta \mathrm{D}_{\mathrm{KL}}(p \,\|\, q), \tag{1.33}$$

where we are minimizing an objective that is an expectation of $f(s)$, regularized by the KL divergence. The optimal solution to this problem has a closed form [244, 227, 109, 167].

Proposition 1.2.12 (Optimal Solution to KL-regularized Minimization)**.** *For q and p with equal support, Problem 1.2.5 has the optimal solution*

$$p^*(x) = \frac{1}{Z} q(x) \exp \frac{f(x)}{\beta}, \tag{1.34}$$

where Z is the normalizing constant.

1.3 Gradients through Argmin

Embedding optimization problems in neural networks has become a popular approach in recent years. Several works such as Burges [47], Finn, Abbeel, and Levine [97], Rajeswaran et al. [256], Bai, Kolter, and Koltun [17], and Chen et al. [60] have utilized different techniques. As an example, *Model-Agnostic Meta-Learning* (MAML) [97] is a a meta-learning algorithm that learns an initialization of a predictor such that it can quickly adapt to new tasks with a few gradient steps, which is an approximate arg min operation. The most naive approach to learn the parameters of the meta-learner is to unroll the optimization process and compute the gradients through the arg min operation. This is first of all very costly in terms of computation, secondly it is not always possible to compute the gradients through the arg min operation without very prohibitive assumptions. Other approaches such as Bai, Kolter, and Koltun [17] rely on the implicit function theorem, with which they circumvent the need to compute the gradients through the unrolled or relaxed optimization problem.

Theorem 1.3.1: Implicit Function Theorem

Let $f\colon \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^m$ be a continuously differentiable function, and let $f(x_0, y_0) = 0$. If the Jacobian of f with respect to y is invertible at (x_0, y_0), then there exists a neighborhood of x_0 and a function $g : \mathbb{R}^n \to \mathbb{R}^m$ such that $f(x, g(x)) = 0$ for all x in the neighborhood.

1.3.1 Differentiating a Fixed-Point Solver

We will now proceed with an important example, for the case of an arg min operation formulated as a fixed-point iteration procedure, which is the main idea behind *Deep Equilibrium Models* (DEMs) [17]. The fix point iteration is defined as

$$x_{t+1} = f(w, x_t), \tag{1.35}$$

where f is a function that depends on the parameters w and the previous iterate x_t. The goal is to learn the parameters w such that the fixed-point iteration converges to the desired solution. The fix point of the function f is defined as the point where $x^* = f(w, x^*)$, so we can write this as

$$\begin{aligned} x^* &= \arg\min_x \ell(w, x) \\ \ell(w, x) &= \|x - f(w, x)\|^2. \end{aligned} \tag{1.36}$$

Assuming that the fix point exists and is unique, we may apply the implicit function theorem to obtain the gradient of the fix point with respect to w, that is, by Theorem 1.3.1, there exists a differentiable mapping $x^* : \mathbb{R}^n \to \mathbb{R}^n$ such that $x^* = x^*(w)$. First of all, consider that

$$g(x^*(w), w) = f(x^*(w), w) - x^* = 0. \tag{1.37}$$

Taking the derivative of this equation on both sides with respect to w we obtain

$$\partial_0 f(x^*(w), w)\partial_w x^*(w) + \partial_1 f(x^*(w), w) - \partial_w x^*(w) = 0. \tag{1.38}$$

By rearranging the terms, we obtain the gradient of the fix point with respect to w as

$$\partial_w x^*(w) = (\mathbf{I} - \partial_0 f(x^*(w), w))^{-1} \partial_1 f(x^*(w), w). \tag{1.39}$$

In reverse-mode automatic differentiation, we are interested in computing vector-Jacobian products, multiplying by the vector v on both sides and taking the transpose we obtain

$$\partial_w x^*(w)^\top v = \partial_1 f(x^*(w), w)^\top (\mathbf{I} - \partial_0 f(x^*(w), w))^{-\top} v. \tag{1.40}$$

What can cause problems in this expression is the inversion of $(\mathbf{I} - \partial_0 f(x^*(w), w))$. However, if we define

$$g \coloneqq (\mathbf{I} - \partial_0 f(x^*(w), w))^{-\top} v, \tag{1.41}$$

we find that g is the solution to an affine fixed point iteration

$$g = \partial_0 f(x^*(w), w)^\top g + v. \tag{1.42}$$

In this way, the inversion of $(\mathbf{I} - \partial_0 f(x^*(w), w))$ never needs to occur, rather we can rely on solving eq. (1.42). In general, Theorem 1.3.1 is a powerful tool for obtaining gradients through arg min operations, however, it is useless for the case where we have a combinatorial solver as a layer in the network, since the resulting objective is piecewise constant, and the Jacobian is zero. Nevertheless, it is interesting that in Chapter 2, solving a modified optimization problem on the backward pass is as well a key step in obtaining an informative update direction for the case of combinatorial layers.

1.3.2 Dealing with Piecewise-Constant Objectives

As we have argued before, for problems of the form eq. (1.19), the objective is piecewise constant with respect to the parameters w. One way of dealing with this is to use zero-order methods for optimization of w, however, due to the curse of dimensionality [36, 202, 112], this is not feasible in practice. If we use the solver as a layer in the network, the parameters w are the result of a projection $\phi(x)$, where ϕ is a neural network. Neural networks are parametrized by billions of parameters, which are impossible to sample and evaluate many times, rendering zero-order methods infeasible. Another approach might be to use finite differences to estimate the gradient $\nabla_w \ell(y(w))$. For demonstration purposes, assume that we have $Y = \{0, 1\}^d$, hence $w \in \mathbb{R}^d$. We may sample a random vector $\varepsilon \in \mathbb{R}^n$ from a noise distribution, and compute the finite difference estimate as

$$\nabla_w \ell(y(w)) \approx \left[\frac{\ell(y(w + \varepsilon_1)) - \ell(y(w))}{\|\varepsilon_1\|}, \dots, \frac{\ell(y(w + \varepsilon_d)) - \ell(y(w))}{\|\varepsilon_d\|} \right]. \tag{1.43}$$

Here, ε_i is the ε vector with exactly one non-zero element at the i-th position. This requires us to evaluate the objective function $d + 1$ times, which is computationally expensive since it involves $d+1$ invocations of the combinatorial solver. Moreover, the finite difference estimate is noisy, and the choice of ε is crucial for the quality of the estimate, in order to obtain a steepest descent direction. Without any priors built into the choice of ε, we would need to sample many different noise vectors, i.e. the computation cost in terms of number of solver invocations is $(d+1) \times n$, where n is the number of perturbation samples. Since many solvers that we want to use are solving problems that are $\mathcal{NP}$-hard, this is not feasible in practice.

As we will see in Chapter 2, it is enough to have 2 evaluations of the combinatorial solver in order to obtain a descent direction that is informative enough for optimization, moreover, no sampling is necessary.

1.4 Reinforcement Learning

The field of RL is mainly concerned with the problem of sequential decision making, however it has its roots in neuroscience and the dopaminergic system of the brain [289, 231], traced back to the early findings of Pavlov [243]. Given an agent modelled through a policy π that interacts with the world, the goal is to find a policy that maximizes the cumulative return. In order to model this interaction, the concept of a *Markov Decision Process* (MDP) is central.

Definition 1.4.1: Markov Decision Process (MDP)

A Markov Decision Process (MDP) is a tuple $\mathcal{M} = (\mathcal{S}, \rho_0, \mathcal{A}, \mathcal{P}, R, \gamma)$ where,

- $\mathcal{S}$ is the set of states,
- $\mathcal{A}$ is the set of actions,
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability function,
- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function,
- $\gamma \in [0, 1]$ is the discount factor.
- $\rho_0 : \mathcal{S} \to [0, 1]$ is the initial state distribution.

Section 1.4 defines a fully observable MDP, meaning that the states $s \in \mathcal{S}$ are fully observable to the agent after executing an action $a \in \mathcal{A}$ in the environment. Here, the rule of choosing the action given state s is defined by a policy mapping $\pi : \mathcal{S} \to \mathcal{A}$. In general, we are interested in the performance of this policy given a specific reward function R and transition function P, which we write as the expected discounted return

$$\mathcal{J}(\pi) = (1 - \gamma)\, \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)\right]. \tag{1.44}$$

Here, T is the time horizon of the episode, which can be finite or infinite. We are seeking the optimal policy, i.e.

$$\pi^* = \arg\max_{\pi} \mathcal{J}(\pi). \tag{1.45}$$

The expectation eq. (1.44) is a non-trivial objective to optimize, as it involves the interaction of the agent with the environment, which is often unknown.

Equation (1.44) allows us to define the return of the policy when starting from a state s, which we have in the form of a value function

Definition 1.4.1 (State Value Function (V))**.** The value function of a policy π is defined as

$$V^{\pi}(s) \coloneqq \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t) \mid s_0 = s, a_t \sim \pi(\cdot|s_t), s_{t+1} \sim \mathcal{P}(\cdot|s_t, a_t)\right]. \tag{1.46}$$

As per this definition, we immediately see the connection between $V^{\pi}(s)$ and $J(\pi)$,

$$\mathcal{J}(\pi) = (1-\gamma) \mathop{\mathbb{E}}_{s_0 \sim \rho_0(s)} \left[V^{\pi}(s_0)\right] \tag{1.47}$$

It is easily verifiable that the value function satisfies the convenient recursive relation

$$V^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) V^{\pi}(s')\right]. \tag{1.48}$$

This is known as the Bellman equation, and it is a fundamental equation in reinforcement learning. Similarly, we have the state-action value function, also known as the Q-function.

Definition 1.4.2 (State-Action Value Finction (Q))**.** The Q-function of a policy π is defined as

$$Q^{\pi}(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) V^{\pi}(s'). \tag{1.49}$$

Similarly to Definitions 1.4.1 and 1.4.2, the optimal state value function and the optimal state-action value function can be defined as

Definition 1.4.3 (Optimal Value Function)**.** The optimal value function is defined as

$$V^*(s) = \max_{\pi} V^{\pi}(s). \tag{1.50}$$

Definition 1.4.4 (Optimal Q-function)**.** The optimal Q-function is defined as

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a). \tag{1.51}$$

We may restate the optimization problem in eq. (1.45) in terms of the value function, as the optimal policy is the one that maximizes the value function,

$$\pi^* = \arg\max_{\pi} V^{\pi}(s) \quad \forall s \in \mathcal{S}. \tag{1.52}$$

Or, more succinctly, the optimal policy is the one that maximizes the optimal Q-function

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s, a) \quad \forall s \in \mathcal{S}, a \in \mathcal{A}. \tag{1.53}$$

For our purposes, it will be also useful to define the notion of the Bellman operator.

Definition 1.4.5 (Bellman Operator)**.** The Bellman operator $\mathcal{T}^{\pi} : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$ is defined as

$$\mathcal{T}^{\pi} V = R^{\pi} + \gamma \mathcal{P}^{\pi}_{ss'} V, \tag{1.54}$$

where R^{π} is the reward vector, and $\mathcal{P}^{\pi}_{ss'}$ is the transition matrix induced under the policy π.

One can show that the Bellman operator is a contraction mapping [27], and hence has a unique fixed point, which is the value function of the policy π. Similarly, the optimal value function is the fixed point of the Bellman operator induced by the optimal policy.

Given a policy π, we may define the state occupancy measure, which is the probability of being in a state s under the policy π.

Definition 1.4.6 (State Occupancy Measure)**.** The state occupancy measure of a policy π is defined as

$$d^{\pi}(s) \coloneqq (1-\gamma) \sum_{t=0}^{\infty} \gamma^t \Pr\Big[s_t = s \,|\, s_0 \sim \rho_0, a_t \sim \pi(\cdot|s_t), s_{t+1} \sim \mathcal{P}(\cdot|s_t, a_t)\Big], \tag{1.55}$$

which is useful to express the discounted return in a more compact form, namely $\mathcal{J}(\pi) = \sum_s d^{\pi}(s) R(s) = \mathbb{E}_{d^{\pi}(s)}[R(s)]$, in case of the reward function being action-dependent as well, we may define the state-action occupancy measure as $d^{\pi}(s, a) = d^{\pi}(s)\pi(a|s)$ and equivalently define the discounted return as $\mathcal{J}(\pi) = \sum_{s \in \mathcal{S}} d^{\pi}(s, a) R(s, a)$. In function space, this is a simple dot-product $\mathcal{J}(\pi) = \langle d^{\pi}, R \rangle$. Note that $d^{\pi}(s)$ is a proper probability distribution, as it sums to one over all states and actions. It may be understood as how often does the agent visit a particular state-action pair, discounted by the γ factor. This viewpoint will prove to be useful in Chapter 6, for deriving an offline algorithm for discovering diverse policies. Most importantly, the state occupancy measure satisfies the so-called Bellman flow constraint.

Definition 1.4.7 (Bellman Flow Constraint)**.** A proper state occupancy measure d^π satisfies the Bellman flow constraint if

$$d^\pi(s) = (1-\gamma)\rho_0(s) + \gamma \sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}} d^\pi(s', a')\mathcal{P}(s|s', a'). \tag{1.56}$$

Definition 1.4.7 states how much probability mass is flowing into a state s from other states, and how much is coming from the initial state distribution ρ_0. Moreover, $d^\pi(s)$ is the single unique solution of eq. (1.56). If we define the transpose (adjoint) policy transition operator $\mathcal{P}^\pi_*$ as

$$\mathcal{P}^\pi_* d(s) = \sum_{s' \in \mathcal{S}} \sum_{a' \in \mathcal{A}} d^\pi(s', a')\mathcal{P}(s|s', a'), \tag{1.57}$$

equivalently, when acting on state-action occupancies, we have

$$\mathcal{P}^\pi_* d(s, a) = \pi(a|s) \sum_{s' \in \mathcal{S}} \sum_{a' \in \mathcal{A}} d^\pi(s', a')\mathcal{P}(s|s', a'). \tag{1.58}$$

This gives us the notion of how much density "flows" into the current state from the future when applying policy π. We may re-write the Bellman flow constraint in operator form as

$$d^\pi = (1-\gamma)\rho_0 + \gamma \mathcal{P}^\pi_* d^\pi. \tag{1.59}$$

1.4.1 Extracting an Optimal Policy

Since we know that the value function is the fixed point of the Bellman operator, we may use this to derive iterative algorithms for finding the value function of a specific policy π, which we call policy evaluation. Here we describe some tractable algorithms for finite-dimensions $\mathcal{S}$ and $\mathcal{A}$, also known as tabular algorithms, some of which rely on the fact that we can replace the expectations with sums, *exactly*. All the following algorithms can be seen as an application of eq. (1.54).

Definition 1.4.8 (Tabular Policy Evaluation)**.** Given a policy π, we may evaluate its value by iterating with the Bellman operator, from an initial guess V^π_0,

$$V^\pi_{i+1}(s) = \mathop{\mathbb{E}}_{\substack{\pi(a \mid s) \\ \mathcal{P}(s' \mid s,a)}} \Big[R(s, a) + \gamma V^\pi_i(s') \Big], \forall s \in \mathcal{S}. \tag{1.60}$$

A key point in eq. (1.60) is that we can compute the expectation tractably. In case this is not possible, one can rely on the error incurred in the value estimate when taking action a, this is also called Temporal Difference (TD) learning.

Definition 1.4.9 (TD(0) Learning)**.** Given a policy π, we observe the transition (s, a, s') and apply the update rule

$$V_{i+1}^{\pi}(s) = V_i^{\pi}(s) + \alpha\Big[R(s,a) + \gamma V_i^{\pi}(s') - V_i^{\pi}(s)\Big], \tag{1.61}$$

where α is the learning rate.

The term $R(s,a) + \gamma V_i^{\pi}(s') - V_i^{\pi}(s)$ is also known as the Temporal Difference error which closely resembles the activity in dopaminergic neurons, that fire in response to unexpected rewards [289]. The question is however, if we don't have Q^* but rather Q^{π} (equivalently for V^* and V^{π}) from an arbitrary policy, can we obtain a better policy than π? The answer is yes, as shown by Theorem 1.4.1.

Theorem 1.4.1: Policy Improvement Theorem

Let π be a policy and V^{π} be its corresponding state-value function. Define a new policy π' such that, for every state $s \in \mathcal{S}$,

$$\pi'(s) = \arg\max_a \mathop{\mathbb{E}}_{s' \sim \mathcal{P}(s'|s,a)} \Big[R(s,a) + \gamma V^{\pi}(s')\Big].$$

Then the policy π' is at least as good as policy π, i.e.,

$$V^{\pi'}(s) \geq V^{\pi}(s) \quad \text{for all} \quad s \in \mathcal{S}.$$

What Theorem 1.4.1 tells us is that, by maximizing the state-action value function $Q^{\pi}(s,a)$ with policy π', we obtain an improved or at least as good of a policy as π. Leveraging this, we may introduce an improvement step in conjunction to eq. (1.60), which is the basis of most reinforcement learning algorithms [312]. If we don't have an explicit policy, but we define it implicitly through a maximization of Q with an initial guess of the value function at a random V_0, we arrive to the value iteration algorithm [27, 293].

Definition 1.4.10 (Value Iteration Update)**.** With initial guess V_0, value iteration performs the update

$$V_{i+1}(s) = \max_a \mathop{\mathbb{E}}_{\mathcal{P}(s' \,|\, s,a)} \Big[R(s,a) + \gamma V_i(s')\Big], \forall s \in \mathcal{S}. \tag{1.62}$$

In the limit of infinite updates, this converges to the optimal value function V^*.

Naturally, value iteration is only possible if we have access to R and $\mathcal{P}$. Here the policy is not given explicitly, but can be extracted by maximizing $Q^*(s,a)$, which is obtainable from

$V^*(s)$ and R. If we lift the requirement of knowing $\mathcal{P}$ to just receiving samples from the environment, and modify the hard update by a learning rate, we would arrive to the well-known Q-learning algorithm [341].

Definition 1.4.11 (Q-Learning)**.** Following an ε-greedy policy and starting from an initial guess Q_0, we perform the update

$$Q_{k+1}(s,a) = Q_k(s,a) + \alpha\Big[R(s,a) + \gamma \max_{a'} Q_k(s',a') - Q_k(s,a)\Big]. \tag{1.63}$$

An ε-greedy policy either takes a random action with probability ε or maximizes the Q value (acts greedily). This update rule is off-policy, since the action taken at s' is not ε-greedy, but rather a maximizer of Q_k at s'. A surprising fact is that Q_k converges to Q^* with probability 1, although relying on the bootstrap update. We state the theorem here for completeness, while a proof can be found in any RL textbook or the original work from Watkins and Dayan [341].

Theorem 1.4.1 (Convergence of Q-Learning [341])**.** *The update rule from eq.* (1.63) *converges to Q^* with probability 1, if the following conditions on the learning rate hold.*

$$\sum_{t=0}^{\infty} \alpha_t(s,a) = \infty, \qquad \sum_{t=0}^{\infty} \alpha_t^2(s,a), \qquad \forall s \in \mathcal{S}, a \in \mathcal{A}$$

To obtain an on-policy update from eq. (1.63), one must simply remove the maximization of Q_k at s' and act according to the ε-greedy policy. This would lead us to an update rule such as *State Action Reward State Action* (SARSA) [312].

At their core, most deep reinforcement learning algorithms are particular instantiations of the above-mentioned update rules, coupled with approximation.

1.4.2 Duality in Reinforcement Learning

The notion of duality, as we have introduced it in Section 1.2.2 appears in RL as well, There are two equivalent formulations of $\mathcal{J}(\pi)$,

$$\mathcal{J}(\pi) = (1-\gamma) \mathop{\mathbb{E}}_{\substack{s_0 \sim \rho(s) \\ a_0 \sim \pi(a|s_0)}} [Q(s_0,a_0)] = \mathop{\mathbb{E}}_{s,a \sim d^\pi(s,a)} [R(s,a)]. \tag{1.64}$$

One is making use of the initial distribution ρ_0 and Q, the other is a reward $R(s,a)$ and occupancy $d^\pi(s,a)$.

$\mathcal{J}(\pi)$ can be obtained by solving the optimization problem, also known as the Q-LP [221],

$$\mathcal{J}(\pi) = \min_{Q}(1-\gamma) \mathop{\mathbb{E}}_{\substack{s_0\sim\rho_0(s)\\ a_0\sim\pi(a|s_0)}} [Q(s_0,a_0)] \tag{1.65}$$

$$\text{s.t.} \quad Q(s,a) \geq R(s,a) + \gamma\mathcal{P}^\pi Q(s,a), \tag{1.66}$$

$$\forall s \in \mathcal{S}, a \in \mathcal{A}. \tag{1.67}$$

The optimal $Q^* = Q^\pi$, the Q function of policy π. $\mathcal{P}^\pi$ is the transition operator under the policy π which has an appropriate adjoint(transpose) operator $\mathcal{P}^\pi_*$. The dual of this LP on the other hand has the form

$$\mathcal{J}(\pi) = \max_{d\geq 0} \sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}} d(s,a)R(s,a) \tag{1.68}$$

$$\text{s.t.} \quad d(s,a) = (1-\gamma)\rho_0(s)\pi(a\,|\,s) + \gamma\mathcal{P}^\pi_* d(s,a) \tag{1.69}$$

$$\forall s \in \mathcal{S}, a \in \mathcal{A}. \tag{1.70}$$

Here, the optimal d^* is the state-action occupancy d^π.

A particular duality relationship that we are interested in is the case where an f-divergence appears in the objective, consider the policy evaluation problem

$$\max_d -D_f(d\,\|\,d^t) \quad \text{s.t.} \quad d = (1-\gamma)\rho_0 + \gamma\mathcal{P}^\pi_* d, \tag{1.71}$$

where the constraint enforces that the distribution d is a valid occupancy distribution under the policy π in $\Delta(\mathcal{S}\times\mathcal{A})$ and d^t is some target occupancy distribution. Applying Fenchel duality yields the unconstrained dual problem [221],

$$\min_Q (1-\gamma)\langle\rho_0\odot\pi, Q\rangle + \gamma\langle d^t, f_*(\gamma\mathcal{P}^\pi Q - Q)\rangle, \tag{1.72}$$

where f_* is the conjugate of the f-function in the f-divergence. Moreover, we may obtain the importance ratios of the state-action occupancy measure that minimizes the divergence as

$$d^*(s,a) = d^t(s,a) f'_*(\gamma\mathcal{P}^\pi Q(s,a) - Q(s,a)). \tag{1.73}$$

As we shall see, in Chapter 6, this dual problem is a key component in deriving an offline algorithm for skill discovery. It allows us to make use of Definition 1.4.12 in order to correct for the difference in the state-action occupancy measure of the policy π and the target distribution.

The previous problem does not take into consideration the reward function, but just tries to match the state-action occupancy measures. Instead, consider the following primal problem,

$$\max_d \langle d, R\rangle - D_f(d\,\|\,d^t) \quad \text{s.t.} \quad d = (1-\gamma)\rho_0 + \gamma\mathcal{P}^\pi_* d. \tag{1.74}$$

Unsurprisingly, this type of problem is ubiquitous throughout the RL field, but also beyond it has gained traction, for example in use-cases of RLHF [64] and DPO [252] in LLM fine-tuning.

Applying Fenchel duality and following the results in f-divergence conjugates from Section 1.2.5, we obtain the dual problem

$$\min_Q (1-\gamma)\langle \rho_0 \odot \pi, Q\rangle + \gamma \langle d^t, f_*(R + \gamma \mathcal{P}^\pi Q - Q)\rangle. \tag{1.75}$$

Specifically, for the case of the often used KL divergence, the unconstrained problem becomes

$$\min_Q (1-\gamma)\langle \rho_0 \odot \pi, Q\rangle + \gamma \log\langle d^t, \exp(R + \gamma \mathcal{P}^\pi Q - Q)\rangle. \tag{1.76}$$

Here the exp is an element-wise exponential and $\odot$ denotes the column-wise Hadammard product, i.e. $\rho_0 \odot \pi$ is a matrix of $|\mathcal{S}| \times |\mathcal{A}|$ elements. While the problem from eq. (1.68) is overconstrained, the following dual of the V-LP problem is not, and its solution yields an optimal policy occupancy for reward $R(s,a)$.

$$\mathcal{J}(\pi) = \max_{d \geq 0} \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} d(s,a) R(s,a) \tag{1.77}$$

$$\text{s.t.} \quad \sum_a d(s,a) = (1-\gamma)\rho_0(s_0) + \gamma \mathcal{T}^* d(s) \tag{1.78}$$

$$\forall s \in \mathcal{S}. \tag{1.79}$$

Note that the dependency on a policy in the constraint is removed, and we only consider state-action occupancies. As it turns out, regularizing eq. (1.77) with an f-divergence $D_f(d||d^t)$ yields an unconstrained dual learning problem

$$\min_V (1-\gamma)\langle \rho_0, V\rangle + \gamma \log\langle d^t, \exp(R + \gamma \mathcal{T} V - V)\rangle, \tag{1.80}$$

which we can map back to the primal occupancy solution d by

$$d(s,a) = d^t(s,a)\text{softmax}_{d^t}(R(s,a) + \gamma \mathcal{T} V(s) - V(s)). \tag{1.81}$$

We will make use of this formulation when deriving a constrained offline skill discovery algorithm in Chapter 6, where the d^t will be an expert occupancy distribution.

1.4.3 Policy Gradient Methods

In complex environments, tabular methods on which we enumerate all possible states, actions and their products in order to find π^* are infeasible. Hence, we resolve to approximate solution methods, which involve some form of gradient optimization. Policy gradient methods are a class of algorithms that directly optimize parameters θ of the policy π in order to find π^*. However, estimating the gradient of the objective function in eq. (1.44) is not trivial, since the direct gradient with respect to θ is not available. Fortunately, there is a way to estimate the gradient of $\mathcal{J}(\pi)$, without having direct access to R and $\mathcal{P}$, this is what the Policy Gradient Theorem [312] provides us with.

Theorem 1.4.2: Policy Gradient Theorem

Let π_θ be a differentiable policy, d^π the state-action occupancy measure and Q^π the state-action value function induced by π. The gradient of the expected return is given by

$$\begin{aligned}\nabla_\theta \mathcal{J}(\pi_\theta) &= \nabla_\theta \sum_{s\in\mathcal{S}} d^\pi(s) \sum_{a\in\mathcal{A}} Q^\pi(s,a)\pi_\theta(a \mid s) \\ &\propto \sum_{s\in\mathcal{S}} d^\pi(s) \sum_{a\in\mathcal{A}} Q^\pi(s,a)\nabla_\theta \pi_\theta(a \mid s).\end{aligned} \tag{1.82}$$

Intuitively, the policy gradient theorem tells us that, in order to improve the policy π, we should move the parameters θ in the direction of the action likelihood gradient weighted by the state-action value function, i.e. the better the action is, the more we should move θ in that direction. This still however requires us to have access to the state-action value function, which may be hard to estimate in practice. Theorem 1.4.2 is the basis of many so-called actor-critic algorithms, such as PPO [287], *Advantage Actor-Critic* (A2C) [213] and *Soft Actor-Critic* (SAC) [126], which involve both learning the policy and the value function simultaneously. Actor-critic methods can be seen as an instance of approximate *Dynamic Programming* (DP), where the we have an approximate policy evaluation step, and an approximate policy improvement step.

1.4.4 Off-Policy Methods

In order to obtain good estimates of the gradient in Theorem 1.4.2, we need to sample from the state-action occupancy measure d^π. This is obtainable normally by rolling out the policy in the environment, if we have access to it. In fact, $Q(s,a)$ may be estimated simply by running many rollouts of the policy and doing a Monte Carlo estimate of the expected discounted return by taking the average. If we base our estimates on the same policy that we

are trying to optimize, we are said to be *on-policy*. However, this is prohibitively expensive in many cases, as we may not have access to the environment, or the environment may be too costly to run (such as a robot in the real world). Therefore, it is of great interest to learn from data generated from different policies, also known as *off-policy* learning. A prime example of off-policiness is the usage of experience replay in order to leverage past experiences in the learning process [3], which is by now a key component of most online reinforcement learning algorithms. The motivation for utilizing experience replay already stems from neuroscience, where studies have shown that the brain replays experiences during awake resting and sleep [16, 234], and numerous "versions" have been suggested in the literature for creating additional off-policy data [12, 199]. Moreover, further evidence has suggested that all experience is not the same and that the brain prioritizes certain experiences over others [300], similarly, multiple forms of distribution correction have been introduced by re-weighting the data [282]. As is visible from Theorem 1.4.2, we take the expectation of the weighted gradient over $d^\pi(s)$, which is the state-action occupancy measure of the policy π. If we have a behavioral policy π_b that generates the data, we may use importance sampling [25, 210] to correct for the difference in the state-action occupancy measure.

Definition 1.4.12 (Importance Sampling)**.** Let $f(x)$ be a function, and $p(x)$ and $q(x)$ be two distributions, then the expectation of $f(x)$ under $p(x)$ can be estimated by sampling from $q(x)$. The estimate is given by

$$\mathop{\mathbb{E}}_{p(x)}[f(x)] = \mathop{\mathbb{E}}_{q(x)}\left[\frac{p(x)}{q(x)} f(x)\right]. \tag{1.83}$$

It takes a simple re-weighting of the samples to correct for the difference in the state-action occupancy measure, meaning that we can take expectations with respect to d^π even if we only have samples from d^{π_b}. However, this requires us to be able to evaluate the importance ratio $\frac{d^\pi(s)}{d^{\pi_b}(s)}$. Chapter 6 shows an application of this technique in the context of offline skill discovery, for the case where we have access to only to an un-normalized likelihood, a self-normalized version may be utilized.

Definition 1.4.13 (Self-Normalized Importance Sampling)**.** Suppose that for samples x_i from $q(x)$, we have access to the un-normalized likelihood $\tilde{p}(x_i)$. Let $W_i = \frac{\tilde{p}(x_i)}{q(x_i)}$ be the importance ratio, then the self-normalized importance sampling estimate of the expectation of $f(x)$ under $p(x)$ is given by

$$\mathop{\mathbb{E}}_{p(x)}[f(x)] = \sum_i \left[\frac{W_i}{\sum_{i=1}^n W_i} f(x)\right]. \tag{1.84}$$

1.4.5 Exploration and Exploitation Tradeoff

Since in most cases the transition density $\mathcal{P}$ is unknown, the agent needs to obtain samples in order to obtain good estimates of the policy gradient and the value function. In the most basic problem setting, the agent starts with no knowledge of the environment, and gradually collects samples in order to gain more insight. This gives rise to the exploration-exploitation tradeoff, where the agent needs to balance between exploiting the current knowledge and exploring new regions of the state-action space. As a result, simple optimization of eq. (1.44) may lead to suboptimal policies. Many methods have introduced ways to encourage exploration, mostly in the form of an additional reward term such as the entropy of the policy [126], use of intrinsic rewards [242] or epistemic uncertainty [330]. In Chapter 5 we investigate the usage of epistemic uncertainty in a zero-order trajectory optimization setting that increases the sample efficiency of learning the dynamics model.

1.5 Model Predictive Control

MPC is a another class of algorithms that are used to solve the problem of optimal sequential decision making, mostly relying on the availability of a model of the system. With the assumption of having a good model, MPC methods have shown to be very effective in practice, and are still widely used. Although continuous formulations exist [261], we will focus on the discrete-time case, which is more relevant for the work in this thesis. Suppose we have a model of the system dynamics $f(s, a)$, we may write the non-stochastic discrete-time dynamics as

$$s_{t+1} = f(s_t, a_t). \tag{1.85}$$

We are searching for a sequence of actions $\{a_i\}_{i=0}^{H}$ for timesteps t to $t + H$ that minimize the accumulated cost over the horizon H. This problem can be formulated as a minimization problem subject to dynamics constraints and initial condition for initial state $s_0 = \bar{s}$.

$$\begin{aligned} \min_{a_0 \dots a_H} \quad & \sum_{i=0}^{H} \mathbf{c}(s_i, a_i) + \mathbf{c}_{\text{term}}(s_{H+1}) \\ s.t. \quad & s_{i+1} = f(s_i, a_i), \quad i = 0, \dots, H. \\ & s_0 = \bar{s}. \end{aligned} \tag{1.86}$$

the cost function $\mathbf{c}$ is a function of the state and action, and $\mathbf{c}_{\text{term}}$ is the terminal cost function. Note that we are optimizing over both sequences of actions and states, however, the sequence of states needs to be consistent with the dynamics of the system. If we take the planning horizon H to be infinite, we obtain the infinite-horizon optimal control problem, and for particularly simple systems, this can be solved in closed form. An example of this is

the *Linear Quadratic Regulator* (LQR), where we have linear dynamics of states and actions, and quadratic cost functions. For LQR the solution is a linear feedback controller (policy), which is a function of the state [189]. We may of course re-write eq. (1.86) in terms of a reward function R and optimal value V^* as

$$\begin{aligned} \max_{a_0 \dots a_H} \quad & \left[\sum_{i=0}^{H} R(s_i, a_i) + V^*(s_{H+1}) \right] \\ s.t. \quad & s_{i+1} = f(s_i, a_i), \quad i = 0, \dots, H \\ & s_0 = \bar{s} \end{aligned} \tag{1.87}$$

This problem is somewhat equivalent to the problem

$$\begin{aligned} \max_{\pi} \quad & V^{\pi}(s_0) \\ s.t. \quad & s_{i+1} = f(s_i, a_i), \quad i = 0, \dots, H \\ & s_0 = s_t. \end{aligned} \tag{1.88}$$

where the policy π is a function of the state, and the value function V^{π} is the expected return of the policy as per Definition 1.4.1. Whereas MPC approaches account for the dynamics function f in the constraints of the optimization problem, RL approaches either learn the transition kernel (model-based) or learn the policy directly (model-free) by interacting with the environment, with the goal of estimating the value function V^{π} and the policy π. In fact, one well-known approach to solving the MPC problem is DP [33], and RL can be seen as an instance of approximate DP where the value function is approximated by samples from the environment [32].

1.5.1 Exact Dynamic Programming

Exact DP methods are approaches to solving the MPC problem by solving the Bellman equation (1.48). In this regard, there is little difference to the RL problem, as the goal is to estimate the value function V^{π}, or, equivalently in the control literature, the value function would be called the *cost-to-go*. The fundamental ideas underlying DP and eq. (1.48) stem from the principle of optimality. Although in the control literature one adopts the cost terminology, it is analogous to the reward in RL.

Definition 1.5.1 (Principle of Optimality)**.** Let $\{a_0^* \dots a_{N-1}^*\}$ be the optimal sequence of actions with the corresponding state sequence $\{s_0^* \dots s_N^*\}$. Then, the subsequence $\{a_k^* \dots a_{N-1}^*\}$ is also optimal for the subproblem starting from state s_k^*,

$$J^*(s_k^*) = \min_{a_k \dots a_{N-1}} \sum_{i=k}^{N-1} \mathbf{c}(s_i, a_i) + \mathbf{c}_{\text{term}}(s_N). \tag{1.89}$$

The sub-problem in Definition 1.5.1 is also referred to as the *tail subproblem*. This means that in order to solve a problem of length N, we may solve the tail subproblems in a backwards manner, starting from the last state. This is only possible if the dynamics are known, and the problem is deterministic. We would need to construct a sequence of optimal cost-to-go functions $J_k^*(s)$, starting from the last step N and working backwards to 0. In the control literature, $J_k^*(s)$ is also known as the optimal *cost-to-go* at state s for time k. Immediately it is clear that except needing a good dynamics model, this approach is only feasible for small problems, where a good approximation of $J_k^*(s)$ can be obtained in a reasonable amount of time. Whereas exact DP methods require solving each subproblem to optimality, which requires exact estimate of $J_k^*(s)$, approximate DP methods only require an approximate estimate of $J_k^*(s)$ [32].

1.5.2 Difficulties in Naive Optimization

One main difficulty that plagues MPC is system identification, which is the process of obtaining a model of the system dynamics (transition kernel) f. In many cases, even if a good model can be obtained for the system, this leads to a difficult non-convex optimization problem which is hard to solve efficiently, since most methods rely on first and second-order approximations resulting in a series of *Quadratic Programs* (QPs) [189]. Since the MPC effectively provides us with an implicit policy that at each environment step needs to solve the optimization problem to provide an action, this is computationally very restrictive. However, MPC has successfully been used in a teacher-student setup, where a closed-loop policy is distills the MPC policy, resulting in more computational efficiency [265]. Furthermore, the dynamics in these cases may not be differentiable, meaning that even solving the problem locally is difficult. In the case of rigid body dynamics, the dynamics are non-differentiable due to the presence of contact forces at the contact points. Modeling these problems results in tedious MIPs that are difficult to solve fast to optimality [276, 292]. For many applications, it is critical that the systems are controlled in real-time, allowing little room for latency such as in autonomous driving or autonomous robots.

1.5.3 Cross-Entropy Method

Previously we have considered optimization approaches that make use of first order information. Oftentimes, gradients are expensive to compute, are not available because the model is a black-box function or, as we have seen, not informative since the function is piecewise constant. In these cases one can take the approach of sampling, i.e. zero-order optimization, in which we sample from a proposal distribution $p(x;\theta)$ and evaluate each sample with the cost we are optimizing. The CEM [274] is an example of such an approach. The idea is to maintain a probability distribution $p(x;\theta)$ over the space of candidate solutions, and iteratively refine this distribution in order to pinpoint the optimal solution. The refinement step involves taking statistics of "elite" solutions, i.e. the top k samples according to some cost ℓ, and maximizing the likelihood of these samples under the distribution p. If the distribution p is simple enough, we can compute the *Maximumum Likelihood Estimation* (MLE) parameters θ in closed form. Assuming a multivariate normal distribution with mean μ and covariance Σ, the refinement step reduces to computing the sample mean and covariance of the elite samples which is simply given by

$$\begin{aligned} \mu_{\text{new}} &= \frac{1}{k}\sum_{i=1}^{k} x_i, \\ \Sigma_{\text{new}} &= \frac{1}{k-1}\sum_{i=1}^{k}(x_i - \mu_{\text{new}})(x_i - \mu_{\text{new}})^\top. \end{aligned} \tag{1.90}$$

We give a simple description of CEM in Algorithm 1, where we run the algorithm for T iterations or the convergence criterion is met. The convenient thing about the algorithm is that we don't need any additional information about the cost function, we just need to be able to invoke it on a sample x_i in order to sort out the "elite" samples. Going back to our discussion about MPC approaches and convex optimization, we may replace expensive computations of Jacobian matrices and Hessian approximations with simple sampling, since CEM is a general optimizer.

The form that will be particularly of interest for us, and is the standard formulation for MPC, is when we want to solve an arg min problem over a sequence of actions $\{a_i\}_{i=0}^{H}$ for a specific planning horizon H. If we have access to the reward function R and transition density $\mathcal{P}$, without ever needing the explicit gradient of the objective function, which is the approximation to the expected return. However, this comes at a cost of needing to call $\mathcal{P}$ in an autoregressive fashion to sample trajectories and evaluate them with R. Such ideas have already been introduced in data-driven algorithms, where an approximate model of $\mathcal{P}$, f_θ, is learned based on data obtained from the environment and subsequently used to estimate

Algorithm 1 Cross-Entropy Method (CEM)

Initialize population size N, elite fraction ρ, and iteration limit T
Initialize parameter distribution θ (e.g., mean μ and covariance Σ)
for $t = 1$ to T **do**
    Sample N candidate solutions $\{x_1, x_2, \ldots, x_N\}$ from distribution $p(x; \theta)$
    Evaluate each candidate x_i using the objective function $\ell(x_i)$
    Sort candidates by $\ell(x_i)$ in descending order
    Select the top $\rho \times N$ candidates as elites
    Update distribution parameters θ
    if convergence criteria met **then**
        break
return the best solution found

the expected return of a sequence of actions [65, 247, 330] in order to obtain elite particles for refinement of p. In comparison to model-free RL methods, the sources of approximation here are the approximate model and the truncated *Monte Carlo* (MC) estimate of the expected return. Chapter 5 deals with introducing a risk-sensitive version of CEM with constraints, in order to continuously improve the model of the system dynamics by guiding the exploration towards regions of the state-action space that are not well understood, while still being risk-averse at evaluation time.

I

Introducing Combinatorial Structure

"You are very harsh."
"I have seen the world."
Voltaire, Candide

2 Differentiation of Blackbox Combinatorial Solvers

This chapter is based on the work "Differentiation of Blackbox Combinatorial Solvers" [334]. We introduce an update rule for differentiating through blackbox solvers that optimize linear cost functions, with validating experiments on shortest path, min-cost perfect matching and traveling salesman problems. The core idea behind the update rule is to construct a piecewise linear interpolation of the objective and taking its gradient. The linearity of the surrogate objective is controlled by a hyperparameter λ.

2.1 Motivation

In this chapter we introduce a technique for incorporating specific classes of combinatorial solvers into differentiable architectures by deducing an "informative" update direction of the parameters, which is *different than the actual gradient*. The toolbox of popular methods in computer science currently sees a split into two major components. On the one hand, there are classical algorithmic techniques from discrete optimization – graph algorithms, SAT-solvers, integer programming solvers – often with heavily optimized implementations and theoretical guarantees on runtime and performance. On the other hand, there is the realm of deep learning allowing data-driven feature extraction as well as the flexible design of end-to-end architectures. The fusion of deep learning with combinatorial optimization is desirable both for foundational reasons – extending the reach of deep learning to data with large combinatorial complexity – and in practical applications. These often occur for example in computer vision problems that require solving a combinatorial sub-task on top

of features extracted from raw input such as establishing global consistency in multi-object tracking from a sequence of frames.

The fundamental problem with constructing hybrid architectures is differentiability of the combinatorial components. State-of-the-art approaches pursue the following paradigm: introduce suitable approximations or modifications of the objective function or of a baseline algorithm that eventually yield a differentiable computation. The resulting algorithms are often sub-optimal in terms of runtime, performance and optimality guarantees when compared to their *unmodified* counterparts. While the sources of sub-optimality vary from example to example, there is a common theme: any differentiable algorithm in particular outputs continuous values and as such it solves a *relaxation* of the original problem. It is well-known in combinatorial optimization theory that even strong and practical convex relaxations induce lower bounds on the approximation ratio for large classes of problems [253, 317] which makes them inherently sub-optimal. This inability to incorporate the best implementations of the best algorithms is lacking.

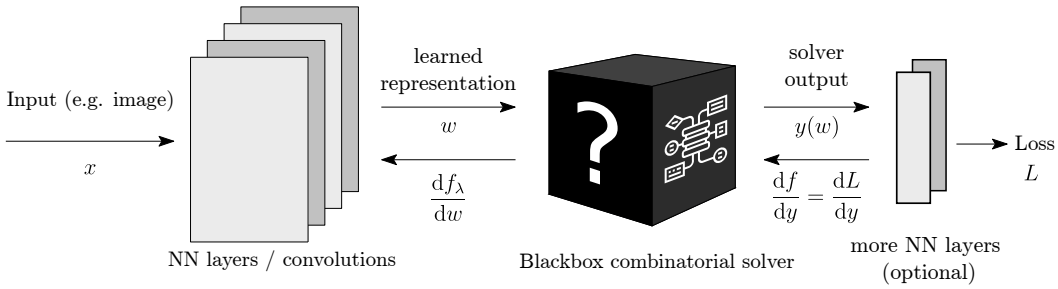


Figure 2.1: Architecture design enabled by Theorem Theorem 2.4.1. Blackbox combinatorial solver embedded into a neural network.

We propose a method that, at the cost of one hyperparameter, implements a backward pass for a *blackbox implementation* of a combinatorial algorithm or a solver that optimizes a linear objective function. This effectively turns the algorithm or solver into a composable building block of neural network architectures, as illustrated in figure 2.1. Suitable problems with linear objective include classical problems such as SHORTEST-PATH, TRAVELING-SALESMAN (TSP), MIN-COST-PERFECT-MATCHING, various cut problems as well as entire frameworks such as integer programs (IP), Markov random fields (MRF) and conditional random fields (CRF).

The main technical challenge boils down to providing an informative gradient of a piecewise constant function. To that end, we are able to heavily leverage the minimization structure of the underlying combinatorial problem and efficiently compute a gradient of a continuous interpolation. While the roots of the method lie in loss-augmented inference, the employed mathematical technique for continuous interpolation is novel. The computational

cost of the introduced *backward pass matches the cost of the forward pass*. In particular, it also amounts to one call to the solver.

In experiments, we train architectures that contain *unmodified implementations* of the following efficient combinatorial algorithms: general-purpose mixed-integer programming solver Gurobi [124], state-of-the-art C implementation of MIN-COST-PERFECT-MATCHING algorithm – Blossom V [165] and Dijkstra's algorithm [85] for SHORTEST-PATH. We demonstrate that the resulting architectures train without sophisticated tweaks and are able to solve tasks that are beyond the capabilities of conventional neural networks.

2.2 Related Work

Multiple lines of work lie at the intersection of combinatorial algorithms and deep learning. We primarily distinguish them by their motivation.

Motivated by applied problems. Even though computer vision has seen a substantial shift from combinatorial methods to deep learning, some problems still have a strong combinatorial aspect and require hybrid approaches. Examples include multi-object tracking [288], semantic segmentation [58], multi-person pose estimation [248, 304], stereo matching [161] and person re-identification [355]. The combinatorial algorithms in question are typically Markov random fields (MRF) [59], conditional random fields (CRF) [206], graph matching [355] or integer programming [288]. In recent years, a plethora of hybrid end-to-end architectures have been proposed. The techniques used for constructing the backward pass range from employing various relaxations and approximations of the combinatorial problem [59, 367] over differentiating a fixed number of iterations of an iterative solver [241, 321, 196] all the way to relying on the structured SVM framework [323, 59].

Motivated by "bridging the gap". Building links between combinatorics and deep learning can also be viewed as a foundational problem; for example, [24] advocate that "combinatorial generalization must be a top priority for AI". One such line of work focuses on designing architectures with algorithmic structural prior – for example by mimicking the layout of a Turing machine [310, 329, 119, 120] or by promoting behavior that resembles message-passing algorithms as it is the case in Graph Neural Networks and related architectures [280, 190, 24]. Another approach is to provide neural network building blocks that are specialized to solve some types of combinatorial problems such as satisfiability (SAT) instances [339], mixed integer programs [96], sparse inference [229], or submodular maximization [322]. A related mindset of learning inputs to an optimization problem gave rise to the "predict-and-optimize" framework and its variants [90, 79, 204]. Some works have directly addressed the question of learning combinatorial optimization algorithms such as

the TRAVELING-SALESMAN-PROBLEM in [28] or its vehicle routing variants [225]. A recent approach also learns combinatorial algorithms via a clustering proxy [344].

There are also efforts to bridge the gap in the opposite direction; to use deep learning methods to improve state-of-the-art combinatorial solvers, typically by learning (otherwise handcrafted) heuristics. Some works have again targeted the TRAVELING-SALESMAN-PROBLEM [166, 82, 28] as well as other $\mathcal{NP}$-Hard problems [191]. Also, more general solvers received some attention; this includes SAT-solvers [290, 291], integer programming solvers (often with learning branch-and-bound rules) [152, 18, 105] and SMT-solvers (satisfiability modulo theories)[19].

2.3 What is a Solver?

We consider solvers of the form $\arg\min_{y \in Y} \mathbf{c}(w, y)$ as per eq. (1.19), where $\mathbf{c}(w, y)$ is some cost function of w and y. Note that, in the general case, the form of this cost function can be arbitrarily complex and there might be no efficient algorithm for computing the optimal solution $y(w)$. Now, we restrict ourselves to the case where the cost function $\mathbf{c}(w, y)$ is linear, i.e.

$$\mathbf{c}(w, y) = \langle w, \phi(y) \rangle \quad \text{for } w \in W \text{ and } y \in Y \tag{2.1}$$

in which $\phi \colon Y \to \mathbb{R}^N$ is an injective representation of $y \in Y$ in $\mathbb{R}^N$. For brevity, we omit the mapping ϕ and instead treat elements of Y as discrete points in $\mathbb{R}^N$. This definition of a solver with linear cost might seem trivial, however note that it has *no assumptions on the set of constraints or on the structure of the output space Y.*

2.4 Solver Differentiation

The task to solve during back-propagation is the following. We receive the gradient $\mathrm{d}L/\mathrm{d}y$ of the global loss L with respect to solver output y at a given point $\hat{y} = y(\hat{w})$. We are expected to return $\mathrm{d}L/\mathrm{d}w$, the gradient of the loss with respect to solver input w at a point $\hat{w}$.

Since Y is finite, there are only finitely many values of $y(w)$. In other words, this function of w is **piecewise constant** and the gradient is identically zero or does not exist (at points of jumps). This should not come as a surprise; if one does a small perturbation to edge weights of a graph, one *usually* does not change the optimal TSP tour and *on rare occasions* alters it drastically. This has an important consequence:

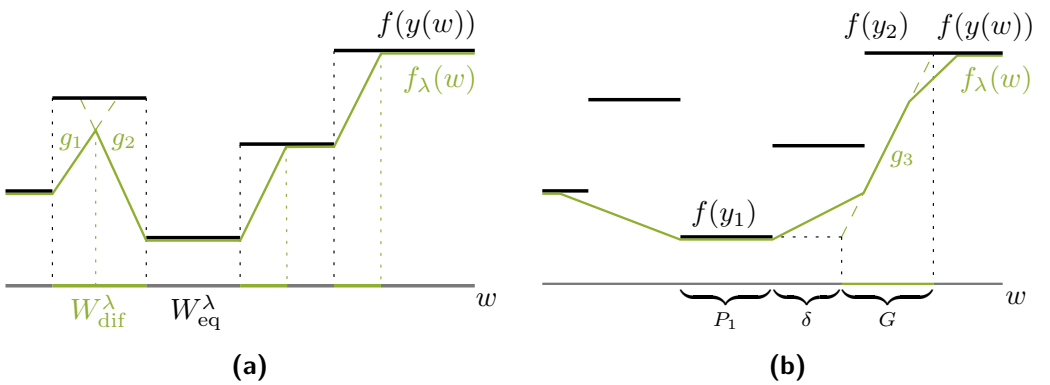


Figure 2.2: Continuous interpolation of a piecewise constant function. (a) f_λ for a small value of λ; the set W^λ_{eq} is still substantial and only two interpolators g_1 and g_2 are incomplete. Also, all interpolators are 0-interpolators. (b) f_λ for a high value of λ; most interpolators are incomplete and we also encounter a δ-interpolator g_3 (between y_1 and y_2) which attains the value $f(y_1)$ δ-away from the set P_1. Despite losing some local structure for high λ, the gradient of f_λ is still informative.[i]

Remark 2.4.1

The fundamental problem with differentiating through combinatorial solvers is not the lack of differentiability; the gradient exists *almost everywhere*. However, this gradient is a constant zero and as such is unhelpful for optimization.

Let us consider the linearization f of L at the point $\hat{y}$. Then for

$$f(y) = L(\hat{y}) + \frac{\mathrm{d}L}{\mathrm{d}y}(\hat{y}) \cdot (y - \hat{y}) \quad \text{we have} \quad \frac{\partial f\big(y(w)\big)}{\partial w} = \frac{\partial L}{\partial w} \tag{2.2}$$

and therefore it suffices to focus on differentiating the piecewise constant function $f\big(y(w)\big)$.

However, the function $f\big(y(w)\big)$ is a result of a minimization process and it is known that for smooth spaces Y there are techniques for such "differentiation through argmin" [283, 278, 98, 87, 11, 10]. It turns out to be possible to build – with different mathematical tools – a viable discrete analogy. In particular, we can efficiently **construct a function** $f_\lambda(w)$, **a continuous interpolation** of $f\big(y(w)\big)$, whose gradient we return (see Figure 2.2). The hyper-parameter $\lambda > 0$ controls the trade-off between "informativeness of the gradient" and "faithfulness to the original function".

Before we give the exact definition of the function f_λ, we formulate several requirements on it. This will help us understand why $f_\lambda(w)$ is a reasonable replacement for $f\big(y(w)\big)$ and, most importantly, why its gradient captures changes in the values of f.

[i] Figure attributed to Vít Musil.

***Property* 1.** For each $\lambda > 0$, f_λ is continuous and piecewise affine.

The second property describes the trade-off induced by changing the value of λ. For $\lambda > 0$, we define sets W^λ_{eq} and W^λ_{dif} as the sets where $f\big(y(w)\big)$ and $f_\lambda(w)$ coincide and where they differ, i.e.

$$W^\lambda_{\text{eq}} = \big\{w \in W : f_\lambda(w) = f\big(y(w)\big)\big\} \quad \text{and} \quad W^\lambda_{\text{dif}} = W \setminus W^\lambda_{\text{eq}}.$$

***Property* 2.** The sets W^λ_{dif} are monotone in λ and they vanish as $\lambda \to 0^+$, i.e.

$$W^{\lambda_1}_{\text{dif}} \subseteq W^{\lambda_2}_{\text{dif}} \quad \text{for } 0 < \lambda_1 \leq \lambda_2 \quad \text{and} \quad W^\lambda_{\text{dif}} \to \emptyset \quad \text{as } \lambda \to 0^+.$$

In other words, Property 2 tells us that λ controls the size of the set where f_λ deviates from f and where f_λ has meaningful gradient. This behaviour of f_λ can be seen in figure 2.2.

In the third and final property, we want to capture the interpolation behavior of f_λ. For that purpose, we define a *δ-interpolator* of f. We say that g, defined on a set $G \subset W$, is a δ-interpolator of f between y_1 and $y_2 \in Y$, if

- g is non-constant affine function;
- the image $g(G)$ is an interval with endpoints $f(y_1)$ and $f(y_2)$;
- g attains the boundary values $f(y_1)$ and $f(y_2)$ at most δ-far away from where $f(y(w))$ does. In particular, there is a point $w_k \in G$ for which $g(w_k) = f(y_k)$ and $\text{dist}(w_k, P_k) \leq \delta$, where $P_k = \{w \in W : y(w) = y_k\}$, for $k = 1, 2$.

In the special case of a 0-interpolator g, the graph of g connects (in a topological sense) two components of the graph of $f\big(y(w)\big)$. In the general case, δ measures *displacement* of the interpolator (see also figure 2.2 for some examples). This displacement on the one hand loosens the connection to $f\big(y(w)\big)$ but on the other hand allows for less local interpolation which might be desirable.

***Property* 3.** The function f_λ consists of finitely many (possibly incomplete) δ-interpolators of f on W^λ_{dif} where $\delta \leq C\lambda$ for some fixed C. Equivalently, the *displacement* is linearly controlled by λ.

In Vlastelica et al. [334] we show that f_λ can be defined by considering the value $f(y)$ and the solution to a perturbed optimization problem

$$y_\lambda(w) = \arg\min_{y \in Y} \{\mathbf{c}(w, y) + \lambda f(y)\}. \tag{2.3}$$

Subsequently, the main result is Theorem 2.4.1 which states the definition of f_λ and its properties.

Theorem 2.4.1: Definition of f_λ

Let $\lambda > 0$. The function f_λ defined by

$$f_\lambda(w) = f\big(y_\lambda(w)\big) - \frac{1}{\lambda}\Big[\mathbf{c}\big(w, y(w)\big) - \mathbf{c}\big(w, y_\lambda(w)\big)\Big] \tag{2.4}$$

satisfies Properties 1 to 3 defined above.

Theorem 2.4.1 is a general result which holds for any solver optimizing a linear dot-product cost, a detailed proof can be found in Supplementary A and Vlastelica et al. [334]. Since, the resulting f_λ is continuous and differentiable, we may take its gradient, which results in the update rule

$$\nabla f_\lambda(w) = -\frac{1}{\lambda}\Big[\frac{\mathrm{d}\mathbf{c}}{\mathrm{d}w}\big(w, y(w)\big) - \frac{\mathrm{d}\mathbf{c}}{\mathrm{d}w}\big(w, y_\lambda(w)\big)\Big] = -\frac{1}{\lambda}\big[y(w) - y_\lambda(w)\big]. \tag{2.5}$$

What are the implications regarding practical usage of this update rule? First, note that w may be output of a neural network and $y(w)$ is the output of the solver – meaning that we may use the solver as a layer in a neural network. The only thing that needs to be modified is the backward pass, i.e. replacing the 0-gradient resulting from the piecewise-constant function $L(y)$ in the chain rule. $L(y)$ itself may contain non-trivial differentiable transformations (e.g. subsquent neural network layers) which are involved in computing $\nabla L(y)$ via chain rule.

Algorithm 2 Forward and Backward Pass

function ForwardPass($\hat{w}$)
  $\hat{y} :=$ **Solver**($\hat{w}$) // $\hat{y} = y(\hat{w})$
  save $\hat{w}$ and $\hat{y}$ for backward pass
  return $\hat{y}$

function BackwardPass($\frac{\mathrm{d}L}{\mathrm{d}y}(\hat{y})$, λ)
  load $\hat{w}$ and $\hat{y}$ from forward pass
  $w' := \hat{w} + \lambda \cdot \frac{\mathrm{d}L}{\mathrm{d}y}(\hat{y})$
    // Calculate perturbed weights
  $y_\lambda :=$ **Solver**(w')
  return $\nabla_w f_\lambda(\hat{w}) := -\frac{1}{\lambda}\big[\hat{y} - y_\lambda\big]$
    // Gradient of f_λ at $\hat{w}$

The question remains how can we efficiently compute $y_\lambda(w)$ in eq. (2.3)? The following Proposition gives the answer.

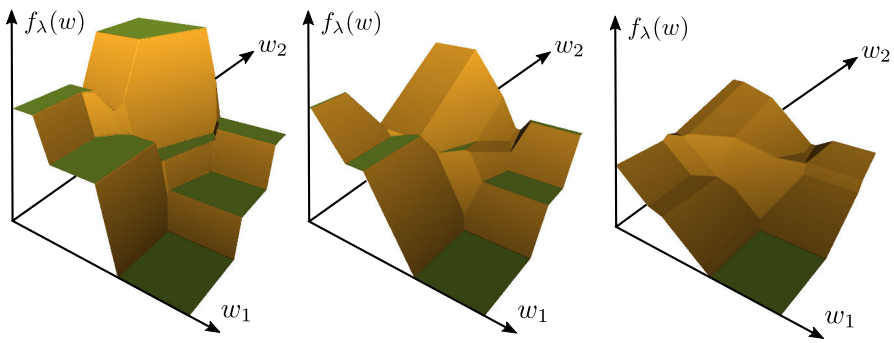


Figure 2.3: Example f_λ for $w \in \mathbb{R}^2$ and $\lambda = 3, 10, 20$ (left to right). As λ changes, the interpolation f_λ is less faithful to the piecewise constant $f(y(w))$ but provides reasonable gradient on a larger set.[ii]

Proposition 2.4.1

Let $\hat{w} \in W$ be fixed. If we set $w' = \hat{w} + \lambda \frac{\mathrm{d}L}{\mathrm{d}y}(\hat{y})$, we can compute y_λ as

$$y_\lambda(\hat{w}) = \arg\min_{y \in Y} \mathbf{c}(w', y).$$

Combinining Theorem 2.4.1 and Proposition 2.4.1 we obtain the algorithm for computing the gradient of $L(y)$ with respect to w. The resulting algorithm that can be implemented in an autofiff library (such as PyTorch) is shown in Algorithm 3.

Applying what we have developed so far, observe the transformation of the piecewise-constant landscape with increasing λ in Figure 2.3.

2.5 Experiments

In this section, we *experimentally validate a proof of concept*: that architectures containing exact blackbox solvers (with backward pass provided by Algorithm 2) can be trained by standard methods.

Table 2.1: Experiments Overview.

Graph Problem	Solver	Solver instance size	Input format
Shortest path	Dijkstra	up to 900 vertices	(image) up to 240×240
Min Cost PM	Blossom V	up to 1104 edges	(image) up to 528×528
Traveling Salesman	Gurobi	up to 780 edges	up to 40 images (20×40)

[ii] Figure attributed to Anselm Paulus

To that end, we solve three synthetic tasks as listed in table 2.1. These tasks are designed to mimic practical examples from Section 5.2 and solving them anticipates a two-stage process: 1) extract suitable features from raw input, 2) solve a combinatorial problem over the features. The dimensionalities of input and of intermediate representations also aim to mirror practical problems and are chosen to be prohibitively large for zero-order gradient estimation methods.

We include the performance of ResNet18 [132] as a sanity check to demonstrate that the constructed datasets are too complex for standard architectures.

Remark 2.5.1

The included solvers have very efficient implementations and do not severely impact runtime. All models train in under two hours on a single machine with 1 GPU and no more than 24 utilized CPU cores. Only for the large TSP problems the solver's runtime dominates.

2.5.1 Warcraft Shortest Path

Problem input and output. The training dataset for problem SP(k) consists of 10000 examples of randomly generated images of terrain maps from the Warcraft II tileset [125]. The maps have an underlying grid of dimension $k \times k$ where each vertex represents a terrain with a fixed cost that is unknown to the network. The shortest (minimum cost) path between top left and bottom right vertices is encoded as an indicator matrix and serves as a label (see also Figure 2.4). We consider datasets SP(k) for $k \in \{12, 18, 24, 30\}$. More experimental details are provided in Supplementary A.4.

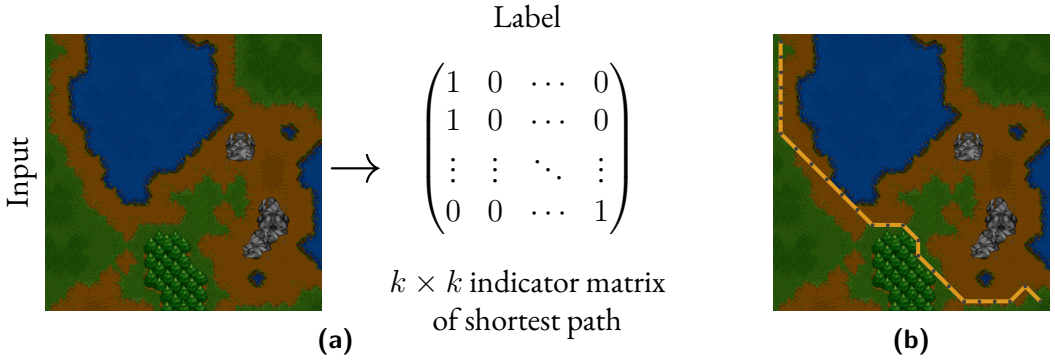


Figure 2.4: The SP(k) dataset. (a) Each input is a $k \times k$ grid of tiles corresponding to a Warcraft II terrain map, the respective label is a the matrix indicating the shortest path from top left to bottom right. (b) is a different map with correctly predicted shortest path.

Architecture. An image of the terrain map is presented to a convolutional neural network which outputs a $k \times k$ grid of vertex costs. These costs are then the input to the Dijkstra

algorithm to compute the predicted shortest path for the respective map. The loss used for computing the gradient update is the Hamming distance between the true shortest path and the predicted shortest path.

Table 2.2: Results for Warcraft shortest path. Reported is the accuracy, i.e. percentage of paths with the optimal costs. Standard deviations are over five restarts.

	Embedding Dijkstra		ResNet18	
k	Train %	Test %	Train %	Test %
12	99.7 ± 0.0	96.0 ± 0.3	100.0 ± 0.0	23.0 ± 0.3
18	98.9 ± 0.2	94.4 ± 0.2	99.9 ± 0.0	0.7 ± 0.3
24	97.8 ± 0.2	94.4 ± 0.6	100.0 ± 0.0	0.0 ± 0.0
30	97.4 ± 0.1	94.0 ± 0.3	95.6 ± 0.5	0.0 ± 0.0

Results. Our method learns to predict the shortest paths with high accuracy and generalization capability, whereas the ResNet18 baseline unsurprisingly fails to generalize already for small grid sizes of $k = 12$. Since the shortest paths in the maps are often nonunique (i.e. there are multiple shortest paths with the same cost), we report the percentage of shortest path predictions that have optimal cost. The results are summarized in table 2.2.

2.5.2 Globe Traveling Salesman Problem

Problem input and output. The training dataset for problem TSP(k) consists of 10000 examples where the input for each example is a k-element subset of fixed 100 country flags and the label is the shortest traveling salesman tour through the capitals of the corresponding countries. The optimal tour is represented by its adjacency matrix (see also figure 2.5). We consider datasets TSP(k) for $k \in \{5, 10, 20, 40\}$.

Architecture. Each of the k flags is presented to a convolutional network that produces k three-dimensional vectors. These vectors are projected onto the unit sphere in $\mathbb{R}^3$; a representation of the globe. The TSP solver receives a matrix of pairwise distances of the k computed locations. The loss of the network is the Hamming distance between the true and the predicted TSP adjacency matrix. The architecture is expected to learn the correct representations of the flags (i.e. locations of the respective countries' capitals on Earth, up to rotations of the sphere). The employed Gurobi solver optimizes a mixed-integer programming formulation of TSP using the cutting plane method [205] for lazy sub-tour elimination.

[iii] Figure attributed to Michal Rolínek.

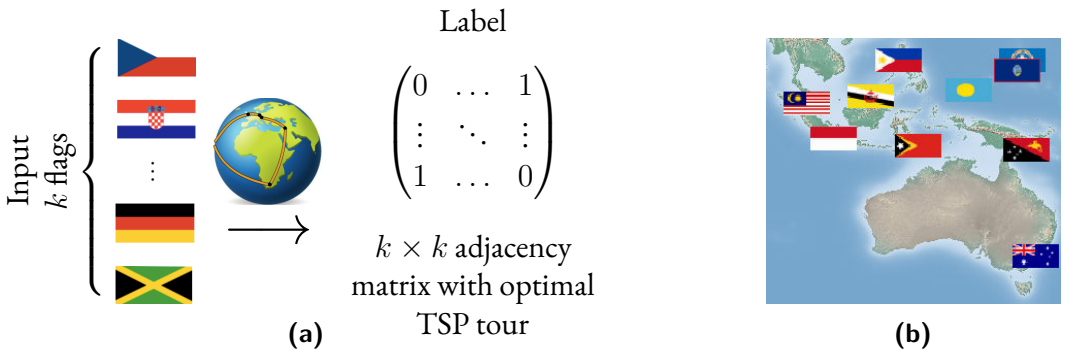


Figure 2.5: The TSP(k) problem. (a) illustrates the dataset. Each input is a sequence of k flags and the corresponding label is the adjacency matrix of the optimal TSP tour around the corresponding capitals. (b) displays the learned locations of 10 country capitals in southeast Asia and Australia, accurately recovering their true position.[iii]

Table 2.3: Results for Globe TSP. Reported is the full tour accuracy. Standard deviations are over five restarts.

	Embedding TSP Solver		ResNet18	
k	Train %	Test %	Train %	Test %
5	99.8 ± 0.0	99.2 ± 0.1	100.0 ± 0.0	1.9 ± 0.6
10	99.8 ± 0.1	98.7 ± 0.2	99.0 ± 0.1	0.0 ± 0.0
20	99.1 ± 0.1	98.4 ± 0.4	98.8 ± 0.3	0.0 ± 0.0
40	97.4 ± 0.2	96.7 ± 0.4	96.9 ± 0.3	0.0 ± 0.0

Results. This architecture not only learns to extract the correct TSP tours but also learns the correct representations. Quantitative evidence is presented in table 2.3, where we see that the learned locations generalize well and lead to correct TSP tours also on the test set and also on somewhat large instances (note that there are $39! \approx 10^{46}$ admissible TSP tours for $k = 40$). The baseline architecture only memorizes the training set. Additionally, we can extract the suggested locations of world capitals and compare them with reality. To that end, we present Figure 2.5b, where the learned locations of 10 capitals in Southeast Asia are displayed.

2.5.3 MNIST Min-cost Perfect Matching

Problem input and output. The training dataset for problem PM(k) consists of 10000 examples where the input to each example is a set of k^2 digits drawn from the MNIST dataset arranged in a $k \times k$ grid. For computing the label, we consider the underlying $k \times k$ grid graph (without diagonal edges) and solve a MIN-COST-PERFECT-MATCHING problem, where edge weights are given simply by reading the two vertex digits as a two-digit number (we read

downwards for vertical edges and from left to right for horizontal edges). The optimal perfect matching (i.e. the label) is encoded by an indicator vector for the subset of the selected edges, see example in figure 2.6.

Architecture. The grid image is the input of a convolutional neural network which outputs a grid of vertex weights. These weights are transformed into edge weights as described above and given to the solver. The loss function is Hamming distance between solver output and the true label.

Table 2.4: Results for MNIST Min-cost perfect matching. Reported is the accuracy of predicting an optimal matching. Standard deviations are over five restarts.

	Embedding Blossom V		ResNet18	
k	Train %	Test %	Train %	Test %
4	99.97 ± 0.01	98.32 ± 0.24	100.0 ± 0.0	92.5 ± 0.3
8	99.95 ± 0.04	99.92 ± 0.01	100.0 ± 0.0	8.3 ± 0.8
16	99.02 ± 0.84	99.06 ± 0.57	100.0 ± 0.0	0.0 ± 0.0
24	95.63 ± 5.49	92.06 ± 7.97	96.1 ± 0.5	0.0 ± 0.0

Results. The architecture containing the solver is capable of good generalizations suggesting that the correct representation is learned. The performance is good even on larger instances and despite the presence of noise in supervision – often there are many optimal matchings. In contrast, the ResNet18 baseline only achieves reasonable performance for the simplest case PM(4). The results are summarized in table 2.4.

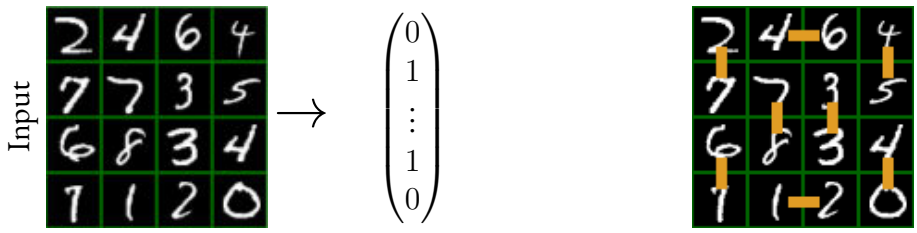


Figure 2.6: Visualization of the PM dataset. (a) shows the case of PM(4). Each input is a 4×4 grid of MNIST digits and the corresponding label is the indicator vector for the edges in the min-cost perfect matching. (b) shows the correct min-cost perfect matching output from the network. The cost of the matching is 348 ($46 + 12$ horizontally and $27 + 45 + 40 + 67 + 78 + 33$ vertically).

2.6 Negative Identity on the Backward Pass

Till now we have dealt with the zero-gradient issue by introducing a principled approach that returns the gradient of the piecewise linear interpolation, requiring one more extra call to the solver on the backward pass. This method, which we named *Blackbox Backpropagation* (BBBP), computes the gradient w.r.t. w as

$$\Delta^{\mathrm{BB}} w = \frac{1}{\lambda}\big(y_\lambda(w) - y(w)\big), \tag{2.6}$$

where $y_\lambda(w)$ is the solution of a perturbed problem.

An alternative approach to avoid the additional call to the solver is to treat the solver as a negative identity block on the backward pass, we shall call this method Id [277].

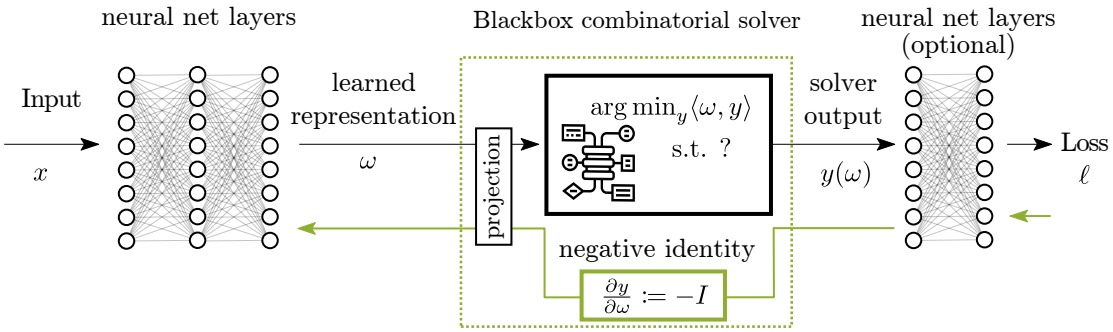


Figure 2.7: Hybrid architecture with blackbox combinatorial solver and Id module (green dotted line) with the projection of a cost w and negative identity on the backward pass.

Consider the case where we have a loss function that provides for each x, y pair a target y^*, this can be for example the simple ℓ_2 loss $\frac{1}{2}\|y^* - y(w)\|^2$ with Y being a subset of $\{0, 1\}^n$. For this case, we have the gradient $\mathrm{d}\ell/\mathrm{d}y = y^* - y(w)$. Treating the solver as the negative identity transformation leads to the gradient with respect to w

$$\begin{aligned} &\tfrac{\mathrm{d}}{\mathrm{d}w}\langle w, y^* - y(w)\rangle && (2.7)\\ &= y^* - y(w) && (2.8)\\ &= -\eta\tfrac{\mathrm{d}\ell}{\mathrm{d}y}. && (2.9)\end{aligned}$$

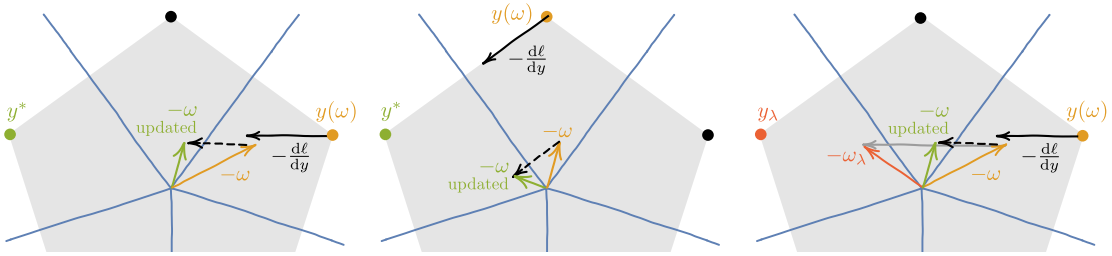
Let us use $\Delta^{\mathrm{I}} w$ as shorthand for the Id update. Because of linearity of the cost $\langle w, y\rangle$ this gradient is guaranteed to increase the cost of $y(w)$ relative to y^*, if $y^* \neq y(w)$.

There is a simple connection of the identity update rule and BBBP. Observe that in eq. (2.6) we have $y_\lambda(w)$ as the "target" with an additional scaling term $\frac{1}{\lambda}$. The target of

BBBP is the solution of the problem minimizing the cost $\mathbf{c}(y, w) + \lambda f(y)$. Furthermore, for sufficiently large λ, $y_\lambda = y^*$, therefore,

$$\Delta^{\mathrm{BB}} w = \frac{1}{\lambda}\Big(y\big(w + \lambda \tfrac{\mathrm{d}\ell}{\mathrm{d}y}\big) - y(w)\Big) = \frac{1}{\lambda}\big(y^* - y(w)\big) = -\frac{\eta}{\lambda}\frac{\mathrm{d}\ell}{\mathrm{d}y} = \frac{\eta}{\lambda}\Delta^{\mathrm{I}} w. \tag{2.10}$$

We therefore conclude for this case that the Id and BBBP updates are equivalent up to the scaling factor $\frac{1}{\lambda}$. Figure 2.8 illustrates this situation, in Figure 2.8a we have the identity update pointing to y^*, for in the next step (Figure 2.8b) we arrive to the desired solution. In Figure 2.8c showcases the situation when $y_\lambda = y^*$ for some λ.



(a) Id update to neighboring solution

(b) Id update to final solution

(c) BB update (identical for some λ)

Figure 2.8: Intuitive illustration of the Id gradient and its equivalence to Blackbox Backpropagation (BB) when $-\mathrm{d}\ell/\mathrm{d}y$ points directly to a target y^*. The cost and solution spaces are overlayed; the cost space partitions resulting in the same solution are drawn in blue. Note that the drawn updates to w are only of illustrative nature, as the updates are typically applied to the weights of a neural network backbone.[iv]

2.7 Solver Invariants [v]

In practice, Id can lead to problematic cost updates when the optimization problem is invariant to a certain transformation of the cost vector w. For instance, adding the same constant to every component of w will not affect its rank or top-k indices. Formally, this means that there exists a mapping $P\colon \mathbb{R}^n \to \mathbb{R}^n$ of the cost vector w that does not change the optimal solver solution, i.e.

$$\arg\min_{y\in Y}\langle w, y\rangle = \arg\min_{y\in Y}\langle P(w), y\rangle \quad \text{for every } w \in W. \tag{2.11}$$

[iv] Figure attributed to Anselm Paulus

[v] Formal treatment attributed to Vít Musil

Linear Transforms. Let us demonstrate why invariants can be problematic in a simplified case assuming that P is linear. Consider an incoming gradient $\mathrm{d}\ell/\mathrm{d}y$, for which the Id method suggests the cost update $\Delta^{\mathrm{I}}w = -\mathrm{d}\ell/\mathrm{d}y$. We can uniquely decompose $\Delta^{\mathrm{I}}w$ into $\Delta^{\mathrm{I}}w = \Delta^{\mathrm{I}}w_1 + \Delta^{\mathrm{I}}w_0$ where $\Delta^{\mathrm{I}}w_1 = P(\Delta^{\mathrm{I}}w) \in \operatorname{Im} P$ and $\Delta^{\mathrm{I}}w_0 = (I - P)\Delta^{\mathrm{I}}w \in \ker P$. Now observe that only the parallel update $\Delta^{\mathrm{I}}w_1$ affects the updated optimization problem, as

$$\begin{aligned}\operatorname*{arg\,min}_{y\in Y}\langle w - \alpha\Delta^{\mathrm{I}}w, y\rangle &= \operatorname*{arg\,min}_{y\in Y}\langle P(w - \alpha\Delta^{\mathrm{I}}w), y\rangle \\ &= \operatorname*{arg\,min}_{y\in Y}\langle P(w - \alpha\Delta^{\mathrm{I}}w_1), y\rangle = \operatorname*{arg\,min}_{y\in Y}\langle w - \alpha\Delta^{\mathrm{I}}w_1, y\rangle\end{aligned} \tag{2.12}$$

for every $w \in W$ and for any step size $\alpha > 0$. In the second equality, we used

$$P(w - \alpha\Delta^{\mathrm{I}}w) = P(w - \alpha\Delta^{\mathrm{I}}w_1) - \alpha P(\Delta^{\mathrm{I}}w_0) = P(w - \alpha\Delta^{\mathrm{I}}w_1), \tag{2.13}$$

exploiting linearity and idempotency of P.

In the case when a user has no control about the incoming gradient $\Delta^{\mathrm{I}}w = -\mathrm{d}\ell/\mathrm{d}y$, the update $\Delta^{\mathrm{I}}w_0$ in $\ker P$ can be much larger in magnitude than $\Delta^{\mathrm{I}}w_1$ in $\operatorname{Im} P$. In theory, if updates were applied directly in cost space, this would not be very problematic, as updates in $\ker P$ do not affect the optimization problem. However, in practice, the gradient is further backpropagated to update the weights in the components before the solver. Spurious irrelevant components in the gradient can therefore easily overshadow the relevant part of the update, which is especially problematic as the gradient is computed from stochastic minibatch samples.

Therefore, it is desirable to discard the irrelevant part of the update. Consequently, for a given incoming gradient $\mathrm{d}\ell/\mathrm{d}y$, we remove the irrelevant part ($\ker P$) and return only its projected part

$$\Delta^{\mathrm{I}}w_1 = -P\tfrac{\mathrm{d}\ell}{\mathrm{d}y}. \tag{2.14}$$

Nonlinear Transforms. For general P, we use the chain rule to differentiate the composed function $y \circ P$ and set the solver's Jacobian to identity. Therefore, for a given w and an incoming gradient $\mathrm{d}\ell/\mathrm{d}y$, we return

$$-P'(w)\tfrac{\mathrm{d}\ell}{\mathrm{d}y}. \tag{2.15}$$

If P is linear, then $P'(w) = P$ and hence update (2.15) is consistent with linear case (2.14). Intuitively, if we replace $P(w - \alpha\Delta^{\mathrm{I}}w)$ in the above-mentioned considerations by its affine approximation $P(w) - \alpha P'(w)\Delta^{\mathrm{I}}w$, the term $P'(w)$ plays locally the role of the linear projection.

Another view on invariant transforms is the following. Consider our combinatorial layer as a composition of two sublayers. First, given w, we simply perform the map $P(w)$ and then pass it to the arg min solver. Clearly, on the forward pass the transform P does not affect the solution $y(w)$. However, the derivative of the combinatorial layer is the composition of the derivatives of its sublayers, i.e. $P'(w)$ for the transform and negative identity for the arg min. Consequently, we get the very same gradient (2.15) on the backward pass. In conclusion, enforcing guarantees on the forward pass by a mapping P is in this sense dual to projecting gradients onto Im P'.

Examples. In our experiments, we will encounter two types of invariant mappings. The first one is the standard projection onto a hyperplane. It is always applicable when all the solutions in Y are contained in a hyperplane, i.e. there exists a unit vector $a \in \mathbb{R}^n$ and a scalar $b \in \mathbb{R}$ such that $\langle a, y\rangle = b$ for all $y \in Y$. Consider the projection of w onto the subspace orthogonal to a given by $P_{\text{plane}}(w|a) = w - \langle a, w\rangle a$. This results in

$$\arg\min_{y\in Y}\langle P_{\text{plane}}(w|a), y\rangle = \arg\min_{y\in Y}\langle w, y\rangle - \langle a, w\rangle b = \arg\min_{y\in Y}\langle w, y\rangle \quad \text{for every } w \in W, \tag{2.16}$$

thereby fulfilling assumption (2.11). This projection is relevant for the ranking and top-k experiment, in which the solutions live on a hyperplane with the normal vector $a = \mathbf{1}/\sqrt{n}$. Therefore, the projection

$$P_{\text{mean}}(w) = P_{\text{plane}}(w|\mathbf{1}/\sqrt{n}) = w - \langle \mathbf{1}, w\rangle \mathbf{1}/n \tag{2.17}$$

is applied on the forward pass which simply amounts to subtracting the mean from the cost vector.

The other invariant mapping arises from the stability of the arg min solution to the magnitude of the cost vector. Due to this invariance, the projection onto the unit sphere $P_{\text{norm}}(w) = w/\|w\|$ also fulfills assumption (2.11). As the invariance to the cost magnitude is independent of the solutions Y, normalization P_{norm} is always applicable and we, therefore, test it in every experiment. Observe that

$$P'_{\text{norm}}(w) = \Big(\frac{I}{\|w\|} - \frac{w \otimes w}{\|w\|^3}\Big) \tag{2.18}$$

and the first order approximation of P_{norm} corresponds to the projection onto the tangent hyperplane given by $a = w$ and $b = 1$. When both P_{norm} and P_{mean} are applicable, we speak about standardization $P_{\text{std}} = P_{\text{norm}} \circ P_{\text{mean}}$.

2.8 Discussion

In this chapter we provided a unified mathematically sound algorithm to embed specific types of combinatorial algorithms into neural networks. We have propose two variants, BBBP that requires one additional call to the solver on the backward pass, and Id which requires no additional call and acts as if the solver were a negative identity transformation. This allows us to train hybrid architectures that contain exact solvers for combinatorial problems end-to-end with gradient descent. However, up until now we have only observed toy experiments as a proof of concept that the combinatorial layer convincingly increases generalization capabilities in comparison to standard architectures. In Chapters 3 and 4 we will see applications of these ideas in a relevant computer vision setting and imitation learning task, respectively.

The only thing worse than being blind is having sight but no vision.

Helen Keller

3 Optimizing Rank-based Metrics

This chapter is based on the work "Optimizing Rank-based Metrics via Blackbox Differentiation" [271]. The fundamental ideas from Chapter 2 are applied in the context of rank-based metrics. Some examples that we consider here are the Average Precision and Recall metrics on an object detection and image retrieval task.

3.1 Motivation

In Chapter 2 we have developed the basis for utilizing combinatorial solvers as layers in neural networks. In this chapter, we consider the application of these ides to rank-based metric optimization. Rank-based metrics are frequently used to evaluate performance on a wide variety of computer vision tasks. For example, in the case of image retrieval, these metrics are required since, at test-time, the models produce a ranking of images based on their relevance to a query. Rank-based metrics are also popular in classification tasks with unbalanced class distributions or multiple classes per image. One prominent example is object detection, where an average over multiple rank-based metrics is used for final evaluation. The most common metrics are recall [106], Average Precision (*AP*) [357], Normalized Discounted Cumulative Gain (*NDCG*) [53], and the Spearman Coefficient [68].

Directly optimizing for the rank-based metrics is tempting, but also notoriously difficult due to the *non-differentiable* (piecewise constant) and *non-decomposable* nature of such metrics. A trivial solution is to use one of several popular surrogate functions such as 0-1 loss [193], the area under the ROC curve [23] or cross entropy. Many studies from the last two decades have addressed direct optimization with approaches ranging from histogram binning approximations [48, 266, 133], finite difference estimation [136], loss-augmented inference

[357], gradient approximation [305] all the way to using a large LSTM to fit the ranking operation [91].

Despite the clear progress in direct optimization [216, 48, 57], these methods are notably omitted in the most publicly used implementation hubs for object detection [56, 349, 207, 144], and image retrieval [273]. The reasons include poor scaling with sequence lengths, lack of publicly available implementations that are efficient on modern hardware, and fragility of the optimization itself.

It turns out that the ranking function itself can be seen as an instance of eq. (1.19), i.e. the ranking operation minimizes a linear combinatorial objective. Conveniently, this allows us to apply the theory developed in Chapter 2 to rank-based metric optimization, yielding substantial improvements on terms of visual benchmarks.

Having a conceptually pure solution for the differentiation, we can then focus on another key aspect: sound loss design. To avoid ad-hoc modifications, we take a deeper look at the caveats of direct optimization for rank-based metrics. We offer multiple approaches for addressing these caveats, most notably we introduce *margin-based versions* of rank-based losses and mathematically derive a *recall-based loss function* that provides dense supervision.

Experimental evaluation is carried out on image retrieval tasks where we optimize the recall-based loss and on object detection where we directly optimize mean Average Precision. On the retrieval experiments, we achieve performance that is on-par with state-of-the-art while using a simpler setup. On the detection tasks, we show consistent improvement over highly-optimized implementations that use the cross-entropy loss, while our loss is used in an out-of-the-box fashion. We als released the code used for our experiments[i].

3.2 Related Work

Optimizing for rank-based metrics As rank-based evaluation metrics are now central to multiple research areas, their direct optimization has become of great interest to the community. Traditional approaches typically rely on different flavors of loss-augmented inference [216, 357, 215, 208], or gradient approximation [305, 136]. These approaches often require solving a combinatorial problem as a subroutine where the nature of the problem is dependent on the particular rank-based metric. Consequently, efficient algorithms for these subproblems were proposed [216, 305, 357].

More recently, differentiable histogram-binning approximations [48, 133, 134, 266] have gained popularity as they offer a more flexible framework. Completely different techniques including learning a distribution over rankings [315], using a policy-gradient update rule

[i] https://github.com/martius-lab/blackbox-backprop.

[258], learning the sorting operation entirely with a deep LSTM [91] or perceptron-like error-driven updates have also been applied [57].

Metric learning There is a great body of work on metric learning for retrieval tasks, where defining a suitable loss function plays an essential role. Bellet, Habrard, and Sebban [26] and Kulis et al. [170] provide a broader survey of metric learning techniques and applications. Approaches with local losses range from employing pair losses [45, 163], triplet losses [286, 141, 305] to quadruplet losses [177]. While the majority of these works focus on local, decomposable losses as above, multiple lines of work exist for directly optimizing global rank-based losses [91, 315, 266]. The importance of good batch sampling strategies is also well-known, and is the subject of multiple studies [233, 286, 106, 348], while others focus on generating novel training examples [365, 305, 219].

Object detection Modern object detectors use a combination of different losses during training [108, 264, 195, 262, 131, 192]. While the biggest performance gains have originated from improved architectures [264, 131, 263, 107] and feature extractors [132, 368], some works focused on formulating better loss functions [192, 267, 110]. Since its introduction in the Pascal VOC object detection challenge [93] *mean Average Precision* (*mAP*) has become the main evaluation metric for detection benchmarks. Using the metric as a replacement for other less suitable objective functions has thus been studied in several works [305, 136, 258, 57].

3.3 Rank-based Metrics

For a positive integer n, we denote by Π_n the set of all permutations of $\{1, \ldots, n\}$. The rank of vector $y = [y_1, \ldots, y_n] \in \mathbb{R}^n$, denoted by $\mathbf{rk}(y)$, is a permutation $\pi \in \Pi_n$ satisfying

$$y_{\pi^{-1}(1)} \geq y_{\pi^{-1}(2)} \geq \cdots \geq y_{\pi^{-1}(n)}, \tag{3.1}$$

i.e. sorting y. Note, that rank is not defined uniquely for those vectors for which any two components coincide. In the formal presentation, we reduce our attention to *proper rankings* in which ties do not occur.

The rank **rk** of the i-th element is one plus the number of members in the sequence exceeding its value, i.e.

$$\mathbf{rk}(y)_i = 1 + |\{j : y_j > y_i\}|. \tag{3.2}$$

Average Precision

For a fixed query, let $y \in \mathbb{R}^n$ be a vector of relevance scores of n examples. We denote by $y^* \in \{0,1\}^n$ the vector of their ground truth labels (relevant/irrelevant) and by

$$\text{rel}(y^*) = \{i : y_i^* = 1\} \tag{3.3}$$

the set of indices of the relevant examples. Then Average Precision is given by

$$AP(y, y^*) = \frac{1}{|\,\text{rel}(y^*)|} \sum_{i \in \text{rel}(y^*)} \text{Prec}(i), \tag{3.4}$$

where precision at i is defined as

$$\text{Prec}(i) = \frac{|\{j \in \text{rel}(y^*) : y_j \geq y_i\}|}{\mathbf{rk}(y)_i} \tag{3.5}$$

and describes the ratio of relevant examples among the i highest-scoring examples.

In classification tasks, the dataset typically consists of annotated images. This we formalize as pairs (x_i, y_i^*) where x_i is an input image and y_i^* is a binary class vector, where, for every i, each $(y_i^*)_c \in \{0,1\}$ denotes whether an image x_i belongs to the class $c \in \mathcal{C}$. Then, for each example x_i the model provides a vector of suggested class-relevance scores $y_i = \phi(x_i, \theta)$, where θ are the parameters of the model.

To evaluate mean Average Precision (*mAP*), we consider for each class $c \in \mathcal{C}$ the vector of scores $y(c) = [(y_i)_c]_i$ and labels $y^*(c) = [(y_i^*)_c]_i$. We then take the mean of Average Precisions over all the classes

$$mAP = \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} AP\big(y(c), y^*(c)\big). \tag{3.6}$$

Note that $mAP \in [0,1]$ and that the highest score 1 corresponds to perfect score prediction in which all relevant examples precede all irrelevant examples.

Recall

Recall is a metric that is often used for information retrieval. Let again $y \in \mathbb{R}^n$ and $y^* \in \{0,1\}^n$ be the scores and the ground-truth labels for a given query over a dataset. For a positive integer K, we set

$$r@K(y, y^*) = \begin{cases} 1 & \text{if } \exists i \in \text{rel}(y^*) \text{ with } \mathbf{rk}(y)_i \leq K \\ 0 & \text{otherwise,} \end{cases} \tag{3.7}$$

where $\mathrm{rel}(y^*)$ is given in eq. (3.3).

In a setup where each element x_i of the dataset $\mathcal{D}$ is a possible query, we define the ground truth matrix as follows. We set $y_i^*(j) = 1$ if x_j belongs to the same class as the query x_i, and zero otherwise. The scores suggested by the model are again denoted by $y_i = [\phi(x_i, x_j, \theta) \colon j \in \mathcal{D}]$.

In order to evaluate the model over the whole dataset $\mathcal{D}$, we average $r@K$ over all the queries x_i, namely

$$R@K = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} r@K\big(y_i, y_i^*\big). \tag{3.8}$$

Again, $R@K \in [0, 1]$ for every K. The highest score 1 means that a relevant example is always found among the top K predictions.

3.4 Method [ii]

Here we present a method for direct optimization of rank-based metrics resulting from the theory developed in Chapter 2.

3.4.1 Blackbox Differentiation for Ranking

In order to apply blackbox differentiation method for ranking, we need to find a suitable combinatorial objective. Let $y \in \mathbb{R}^n$ be a vector of n real numbers (the scores) and let $\mathbf{rk} \in \Pi_n$ be their ranks. The connection between blackbox solver and ranking is captured in the following proposition.

Proposition 3.4.1: Ranking Problem

In the notation set by Eqs. (3.1) and (3.2), we have

$$\mathbf{rk}(y) = \operatorname*{arg\,min}_{\pi \in \Pi_n} \langle y, \pi \rangle. \tag{3.9}$$

In other words, the mapping $y \to \mathbf{rk}(y)$ is a minimizer of a linear combinatorial objective just as Chapter 2 requires.

The proof of Proposition 3.4.1 rests upon a classical rearrangement inequality [129, Theorem 368]. The following theorem is its weaker formulation that is sufficient for our purpose.

[ii] Fromal treatment attributed to Michal Rolínek and Vít Musil.

Theorem 3.4.1: Rearrangement Inequality

For every positive integer n, every choice of real numbers $y_1 \geq \cdots \geq y_n$ and every permutation $\pi \in \Pi_n$ it is true that

$$y_1 \cdot 1 + \cdots + y_n \cdot n \leq y_1\pi(1) + \cdots + y_n\pi(n).$$

Moreover, if $y_1, \ldots, y_n$ are distinct, equality occurs precisely for the identity permutation π.

Proof of Proposition 3.4.1. Let π be the permutation that minimizes (3.9). This means that the value of the sum

$$y_1\pi(1) + \cdots + y_n\pi(n) \tag{3.10}$$

is the lowest possible. Using the inverse permutation π^{-1} (3.10) rewrites as

$$y_{\pi^{-1}(1)} \cdot 1 + \cdots + y_{\pi^{-1}(n)} \cdot n \tag{3.11}$$

and therefore, being minimal in (3.11) makes (3.1) hold due to Theorem 3.4.1. This shows that $\pi = \mathbf{rk}(y)$. ■

The resulting gradient computation is provided in Algorithm 3 and only takes a few lines of code. We call the method *Ranking Metric Blackbox Optimization* (*Rank Metric Blackbox Optimization* (RaMBO)).

Note again the presence of a blackbox ranking operation. In practical implementation, we can delegate this to a built-in function of the employed framework (e.g. TORCH.ARGSORT). Consequently, we inherit the $O(n \log n)$ computational complexity as well as a fast vectorized implementation on a GPU. To our knowledge, the resulting algorithm is the first to have both truly sub-quadratic complexity (for both forward and backward pass) *and* to operate with a general ranking function as can be seen in table 3.1 (not however that [216] have a lower complexity as they specialize on AP and not general ranking).

3.4.2 Caveats for Sound Loss Design

Is resolving the non-differentiability all that is needed for direct optimization? Unfortunately not. To obtain well-behaved loss functions, some delicate considerations need to be made. Below we list a few problems (P1)–(P3) that arise from direct optimization without further adjustments.

Method	forward + backward	general ranking
RaMBO	$O(n \log n)$	✓
Mohapatra et al. [216]	$O(n \log p)$	x
Chen et al. [57]	$O(np)$	✓
Yue et al. [357]	$O(n^2)$	x
FastAP [48]	$O\big((n+p)L\big)$	✓
SoDeep [91]	$O\big((n+p)h^2\big)$	✓

Table 3.1: Computational complexity of different approaches for differentiable ranking. The numbers of the negative and of the positive examples are denoted by n and p, respectively. For SoDeep, h denotes the LSTM's hidden state size ($h \approx n$) and for FastAP L denotes the number of bins. RaMBO is the first method to directly differentiate general ranking with a truly sub-quadratic complexity.

Algorithm 3 RaMBO: Blackbox differentiation for ranking

define Ranker as blackbox operation computing ranks
function ForwardPass(y)
  rk$(y) :=$ **Ranker**(y)
  save y and **rk**(y) for backward pass
  return rk(y)
function BackwardPass($\frac{\mathrm{d}L}{\mathrm{d}\,\mathbf{rk}}$)
  load y and **rk**(y) from forward pass
  load hyperparameter λ
  $y_\lambda := y + \lambda \cdot \frac{\mathrm{d}L}{\mathrm{d}\,\mathbf{rk}}$
  rk$(y_\lambda) :=$ **Ranker**(y_λ)
  return $-\frac{1}{\lambda}\big[\mathbf{rk}(y) - \mathbf{rk}(y_\lambda)\big]$

(P1) Evaluation of rank-based metrics is typically carried out over the whole test set while direct optimization methods rely on mini-batch approximations. This, however, **does not yield an unbiased gradient estimate**. Particularly small mini-batch sizes result in optimizing a very poor approximation of *mAP*, see Figure 3.1.

(P2) Rank-based metrics are **brittle when many ties happen in the ranking**. As an example, note that any rank-based metric attains *all its values* in the neighborhood of a dataset-wide tie. Additionally, once a positive example is rated higher than all negative examples even by the slightest difference, the metric gives no incentive for increasing the difference. This induces a high sensitivity to potential shifts in the statistics when switching to the test set. The need to pay special attention to ties was also noted in [48, 133].

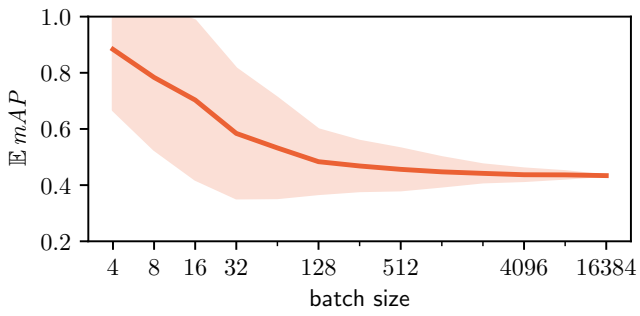


Figure 3.1: Mini-batch estimation of *mean Average Precision*. The expected mAP (i.e. the optimized loss) is an overly optimistic estimator of the true mAP over the dataset; particularly for small batch sizes. The mean and standard deviations over sampled mini-batch estimates are displayed.

(P3) Some metrics give only **sparse supervision**. For example, the value of $r@K$ only improves if the *highest-ranked* positive example moves up the ranking, while the other positives have no incentive to do so. Similarly, Average Precision does not give the incentive to decrease the possibly high scores of negative examples, unless also some positive examples are present in the mini-batch. Since positive examples are typically rare, this can be problematic.

3.4.3 Score Memory

In order to mitigate the negative impact of small batch sizes on approximating the dataset-wide loss (P1) we introduce a simple running memory. It stores the scores for elements of the last τ previous batches, thereby reducing the bias of the estimate. All entries are concatenated for loss evaluation, but the gradients only flow through the current batch. This is a simpler variant of "batch-extension" mechanisms introduced in [48, 266]. Since only the scores are stored, and not network parameters or the computational graph, this procedure has a minimal GPU memory footprint.

3.4.4 Score Margin

Our remedy for brittleness around ties (P2) is inspired by the triplet loss [286]; we introduce a shift in the scores during training in order to induce a *margin*. In particular, we add a negative shift to the positively labeled scores and positive shift the negatively labeled scores as illustrated in Figure 3.2. This also implicitly removes the destabilizing scale-invariance. Using notation as before, we modify the scores as

$$\overleftrightarrow{y}_i = \begin{cases} y_i + \frac{\alpha}{2} & \text{if } y_i^* = 0 \\ y_i - \frac{\alpha}{2} & \text{if } y_i^* = 1 \end{cases} \tag{3.12}$$

Figure 3.2: Naive rank-based losses can collapse during optimization. Shifting the scores during training induces a margin and a suitable scale for the scores. Red lines indicate negative scores and green positive scores.

where α is the prescribed margin. In the implementation, we replace the ranking operation with $\mathbf{rk}_\alpha$ given by

$$\mathbf{rk}_\alpha(y) = \mathbf{rk}\left(\overleftrightarrow{y}\right). \tag{3.13}$$

3.4.5 Recall Loss Design

Let y be scores and y^* the truth labels, as usual. As noted in (P3) the value of $r@K$ only depends on the highest scoring relevant element. We overcome the sparsity of the supervision by introducing a refined metric

$$\widetilde{r}@K(y, y^*) = \frac{|\{i \in \mathrm{rel}(y^*) : r_i < K\}|}{|\,\mathrm{rel}(y^*)|}, \tag{3.14}$$

where $\mathrm{rel}(y^*)$ denotes the set of relevant elements (3.3) and r_i stands for the number of irrelevant elements outrunning the i-th element. Formally,

$$r_i = \mathbf{rk}_\alpha(y)_i - \mathbf{rk}_\alpha(y^+)_i \quad \text{for } i \in \mathrm{rel}(y^*), \tag{3.15}$$

in which $\mathbf{rk}_\alpha(y^+)_i$ denotes the rank of the i-th element only within the relevant ones. Note that $\widetilde{r}@K$ depends on all the relevant elements as intended. We then define the loss at K as

$$L@K(y, y^*) = 1 - \widetilde{r}@K(y, y^*). \tag{3.16}$$

Next, we choose a weighting $w_K \geq 0$ of these losses

$$L_{\mathrm{rec}}(y, y^*) = \sum_{K=1}^{\infty} w_K \, L@K(y, y^*), \tag{3.17}$$

over values of K.

Proposition Theorem B.2.2 (see the Supplementary material) computes a closed form of eq. (3.17) for a given sequence of weights w_K. Here, we exhibit closed-form solutions for two

natural decreasing sequences of weights:

$$L_{\text{rec}}(y, y^*) = \begin{cases} \mathop{\mathbb{E}}_{i \in \text{rel}(y^*)} \ell(r_i) & \text{if } w_K \approx \frac{1}{K} \\ \mathop{\mathbb{E}}_{i \in \text{rel}(y^*)} \ell\big(\ell(r_i)\big) & \text{if } w_K \approx \frac{1}{K \log K}, \end{cases} \tag{3.18}$$

where $\ell(k) = \log(1 + k)$.

This also gives a theoretical explanation why some previous works [57, 136] found it "beneficial" to optimize the logarithm of a ranking metric, rather than the metric itself. In our case, the log arises from the most natural weight decay $1/K$.

3.4.6 Average Precision Loss Design

Having differentiable ranking, the generic *AP* does not require any further modifications. Indeed, for any relevant element index $i \in \text{rel}(y^*)$, its precision obeys

$$\text{Prec}(i) = \frac{\mathbf{rk}_\alpha(y^+)_i}{\mathbf{rk}_\alpha(y)_i} \tag{3.19}$$

where $\mathbf{rk}(y^+)_i$ is the rank of the i-th element within all the relevant ones. The *AP* loss then reads

$$L_{AP}(y, y^*) = 1 - \mathop{\mathbb{E}}_{i \in \text{rel}(y^*)} \text{Prec}(i). \tag{3.20}$$

For calculating the mean Average Precision loss L_{mAP}, we simply take the mean over the classes $\mathcal{C}$.

To alleviate the sparsity of supervision caused by rare positive examples (P3), we also consider the *AP* loss across all the classes. More specifically, we treat the matrices $y(c)$ and $y^*(c)$ as concatenated vectors $\bar{y}$ and $\bar{y}^*$, respectively, and set

$$L_{AP\mathcal{C}} = L_{AP}(\bar{y}, \bar{y}^*). \tag{3.21}$$

This practice is consistent with Chen et al. [57].

3.5 Experiments

We evaluate the performance of RaMBO on object detection and several image retrieval benchmarks. The experiments demonstrate that our method for differentiating through *mAP* and recall is generally on-par with the state-of-the-art results and yields in some cases better performance. We will release code upon publication. Throughout the experimental section, the numbers we report for RaMBO are averaged over three restarts.

Figure 3.3: *Stanford Online Products* image retrieval examples.

3.5.1 Image Retrieval

To evaluate the proposed Recall Loss eq. (3.18) derived from RaMBO we run experiments for image retrieval on the CUB-200-2011 [342], Stanford Online Products [302], and In-shop Clothes [197] benchmarks. We compare against a variety of methods from recent years, multiple of which achieve state-of-the-art performance. The best-performing methods are ABE-8 [157], FastAP [48], and Proxy NCA [219].

Architecture For all experiments, we follow the most standard setup. We use a pre-trained ResNet50 [132] in which we replace the final softmax layer with a fully connected embedding layer which produces a 512-dimensional vector for each batch element. We normalize each vector so that it represents a point on the unit sphere. The cosine similarities of all the distinct pairs of elements in the batch are then computed and the ground truth similarities are set to 1 for those elements belonging to the same class and 0 otherwise. The obvious similarity of each element with itself is disregarded. We compute the L_{rec} loss for each batch element with respect to all other batch elements using the similarities and average it to compute the final loss. Note that our method does not employ any sampling strategy for *mining* suitable pairs/triplets from those present in a batch. It does, however, share a *batch preparation* strategy with [48] on two of the datasets.

Parameters We use Adam optimizer [158] with an amplified learning rate for the embedding layer. We consistently set the batch size to 128 so that each experiment runs on a GPU with 16GB memory. Full details regarding training schedules and exact values of hyperparameters for the different datasets are in the Supplementary material.

Datasets For data preparation, we resize images to 256×256 and randomly crop and flip them to 224×224 during training, using a single center crop on evaluation.

We use the *Stanford Online Products* dataset consisting of $120,053$ images with $22,634$ classes crawled from Ebay. The classes are grouped into 12 superclasses (e.g. cup, bicycle) which are used for mini-batch preparation following the procedure proposed in [48]. We follow the evaluation protocol proposed in [302], using $59,551$ images corresponding to $11,318$ classes for training and $60,502$ images corresponding to $11,316$ classes for testing.

The *In-shop Clothes* dataset consists of $54,642$ images with $11,735$ classes. The classes are grouped into 23 superclasses (e.g. MEN/Denim, WOMEN/Dresses), which we use for mini-batch preparation as before. We follow previous work by using $25,882$ images corresponding to $3,997$ classes for training and $14,218 + 12,612$ images corresponding to $3,985$ classes each for testing (split into a query + gallery set respectively). Given an image from the query set, we retrieve corresponding images from the gallery set.

The *CUB-200-2011* dataset consists of $11,788$ images of 200 bird categories. Again we follow the evaluation protocol proposed in [302], using the first 100 classes consisting of $5,864$ images for training and the remaining 100 classes with $5,924$ images for testing.

Results For all retrieval results in the tables we add the embedding dimension as a superscript and the backbone architecture as a subscript. The letters R, G, V represent ResNet [135], GoogLeNet [313], and VGG-16 [299], respectively. We report results for both RaMBO $^{512}_{R50}$ log and RaMBO $^{512}_{R50}$ log log, the main difference being if the logarithm is applied once or twice to the rank in Eq. (3.18).

On Stanford Online Products we report $R@K$ for $K \in \{1, 10, 100, 1000\}$ in table 3.2. The fact that the dataset contains the highest number of classes seems to favor RaMBO, as it outperforms all other methods. Some example retrievals are presented in Figure 3.3.

On CUB-200-2011 we report $R@K$ for $K \in \{1, 2, 4, 8\}$ in table 3.3. For fairness, we include the performance of Proxy NCA with a ResNet50 [132] backbone even though the results are only reported in an online implementation [273]. With this implementation Proxy NCA and RaMBO are the best-performing methods.

On In-shop Clothes we report $R@K$ for value of $K \in \{1, 10, 20, 30, 50\}$ in table 3.4. The best-performing method is probably FastAP, even though the situation regarding reproducibility is puzzling[iii]. RaMBO matches the performance of ABE-8 [157], a complex ensemble method.

We followed the reporting strategy of [157] by evaluating on the test set in regular training

$R@K$	1	10	100	1000
Contrastive$_G^{512}$ [233]	42.0	58.2	73.8	89.1
Triplet$_G^{512}$ [233]	42.1	63.5	82.5	94.8
LiftedStruct$_G^{512}$ [233]	62.1	79.8	91.3	97.4
Binomial Deviance$_G^{512}$ [324]	65.5	82.3	92.3	97.6
Histogram Loss$_G^{512}$ [324]	63.9	81.7	92.2	97.7
N-Pair-Loss$_G^{512}$ [301]	67.7	83.8	93.0	97.8
Clustering$_G^{64}$ [232]	67.0	83.7	93.2	-
HDC$_G^{384}$ [356]	69.5	84.4	92.8	97.7
Angular Loss$_G^{512}$ [337]	70.9	85.0	93.5	98.0
Margin$_{R50}^{128}$ [348]	72.7	86.2	93.8	98.0
Proxy NCA$_G^{64}$ [219]	73.7	-	-	-
A-BIER$_G^{512}$ [235]	74.2	86.9	94.0	97.8
HTL$_G^{128}$ [106]	74.8	88.3	94.8	98.4
ABE-8$_G^{512}$ [157]	76.3	88.4	94.8	98.2
FastAP$_{R50}^{512}$ [48]	76.4	89.1	95.4	98.5
RaMBO$_{R50}^{512}$ log	77.8	90.1	95.9	**98.7**
RaMBO$_{R50}^{512}$ log log	**78.6**	**90.5**	**96.0**	**98.7**

Table 3.2: Comparison with the state-of-the-art on the Stanford Online Products [233]. On this dataset, with the highest number of classes in the test set, RaMBO gives better performance than other state-of-the-art methods.

R@K	1	2	4	8
Contrastive$^{512}_{G}$ [233]	26.4	37.7	49.8	62.3
Triplet$^{512}_{G}$ [233]	36.1	48.6	59.3	70.0
LiftedStruct$^{512}_{G}$ [233]	47.2	58.9	70.2	80.2
Binomial Deviance$^{512}_{G}$ [324]	52.8	64.4	74.7	83.9
Histogram Loss$^{512}_{G}$ [324]	50.3	61.9	72.6	82.4
N-Pair-Loss$^{64}_{G}$ [301]	51.0	63.3	74.3	83.2
Clustering$^{64}_{G}$ [232]	48.2	61.4	71.8	81.9
Proxy NCA$^{512}_{G}$ [219]	49.2	61.9	67.9	72.4
Smart Mining$^{64}_{G}$ [130]	49.8	62.3	74.1	83.3
Margin$^{128}_{G}$ [348]	63.8	74.4	83.1	90.0
HDC$^{384}_{G}$ [356]	53.6	65.7	77.0	85.6
Angular Loss$^{512}_{G}$ [337]	54.7	66.3	76.0	83.9
HTL$^{128}_{G}$ [106]	57.1	68.8	78.7	86.5
A-BIER$^{512}_{G}$ [235]	57.5	68.7	78.3	86.2
ABE-8$^{512}_{G}$ [157]	60.6	71.5	80.5	87.7
Proxy NCA$^{512}_{R50}$ [273]	**64.0**	**75.4**	**84.2**	90.5
RaMBO$^{512}_{R50}$ log	63.5	74.8	84.1	90.4
RaMBO$^{512}_{R50}$ log log	**64.0**	75.3	84.1	**90.6**

Table 3.3: Comparison with the state-of-the-art on the CUB-200-2011 [342] dataset. Our method RaMBO is on-par with an (unofficial) ResNet50 implementation of Proxy NCA.

R@K	1	10	20	30	50
FashionNet$_V$ [197]	53.0	73.0	76.0	77.0	80.0
HDC$_G^{384}$ [356]	62.1	84.9	89.0	91.2	93.1
DREML$_{R18}^{48}$ [353]	78.4	93.7	95.8	96.7	-
HTL$_G^{128}$ [106]	80.9	94.3	95.8	97.2	97.8
A-BIER$_G^{512}$ [235]	83.1	95.1	96.9	97.5	98.0
ABE-8$_G^{512}$ [157]	87.3	96.7	97.9	98.2	98.7
FastAP-Matlab$_{R50}^{512}$ [48]	**90.9**	**97.7**	**98.5**	**98.8**	**99.1**
FastAP-Python$_{R50}^{512}$ [49] [iii]	83.8?	95.5?	96.9?	97.5?	98.2?
RaMBO$_{R50}^{512}$ log	88.1	97.0	97.9	98.4	98.8
RaMBO$_{R50}^{512}$ log log	86.3	96.2	97.4	97.9	98.5

Table 3.4: Comparison with the state-of-the-art methods on the In-shop Clothes [197] dataset. RaMBO is on par with an ensemble-method ABE-8. Leading performance is achieved with a Matlab implementation of FastAP.

intervals and reporting performance at a time-point that maximizes *R@1*.

3.5.2 Object Detection

We follow a common protocol for testing new components by using Faster R-CNN [264], the most commonly used model in object detection, with standard hyperparameters for all our experiment. We compare against baselines from the highly optimized mmdetection toolbox [56] and only exchange the cross-entropy loss of the classifier with a weighted combination of L_{mAP} and L_{APC}.

Datasets and evaluation All experiments are performed on the widely used Pascal VOC dataset [93]. We train our models on the Pascal VOC 07 and VOC 12 `trainval` sets and test them on the VOC 07 `test` set. Performance is measured in AP^{50} which is AP computed for bounding boxes with at least 50% intersection-over-union overlap with any of the ground truth bounding boxes.

Parameters The model was trained for 12 epochs on a single GPU with a batch-size of 8. The initial learning rate 0.1 is reduced by a factor of 10 after 9 epochs. For the L_{AP} loss, we use $\tau = 7$, $\alpha = 0.15$, and $\lambda = 0.5$. The losses L_{mAP} and L_{APC} are weighted in the $2:1$ ratio.

[iii] FastAP public code [49] offers Matlab and PyTorch implementations. Confusingly, the two implementations give very different results. We contacted the authors but neither we nor they were able to identify the source of this discrepancy in two seemingly identical implementations. We report both numbers.

Method	Backbone	Training	CE	RaMBO
Faster R-CNN	ResNet50	07	74.2	**75.7**
Faster R-CNN	ResNet50	07+12	80.4	**81.4**
Faster R-CNN	ResNet101	07+12	82.4	**82.9**
Faster R-CNN	X101 32×4d	07+12	83.2	**83.6**

Table 3.5: Object detection performance on the Pascal VOC 07 `test` set measured in AP^{50}. Backbone X stands for ResNeXt and CE for cross entropy loss.

Length	100k	1M	10M	100M
CPU	33 ms	331 ms	3.86 s	36.4 s
GPU	1.3 ms	7 ms	61 ms	0.62 s

Table 3.6: Processing time of Average Precision (using plain PYTORCH implementation) depending on sequence length for forward/backward computation on a single Tesla V100 GPU and 1 Xeon Gold CPU core at 2.2GHz.

Results We evaluate Faster R-CNN trained on VOC 07 and VOC 07+12 with three different backbones (ResNet50, ResNet101, and ResNeXt101 32x4d [132, 350]). Training with our AP loss gives a consistent improvement (see table 3.5) and pushes the standard Faster R-CNN very close to state-of-the-art values ($\approx$ 84.1) achieved by significantly more complex architectures [364, 156].

3.5.3 Speed

Since RaMBO can be implemented using sorting functions it is very fast to compute (see table 3.6) and can be used on very long sequences. Computing AP loss for sequences with 320k elements as in the object detection experiments takes less than 5 ms for the forward/backward pass. This is $< 0.5\%$ of the overall computation time on a batch.

R@1	CUB200	In-shop	Online Prod.
Full RaMBO	64.0	88.1	78.6
No batch memory	62.5	87.0	72.4
No margin	63.2	x	x

Table 3.7: Ablation experiments for margin(Section 3.4.4) and batch memory (Section 3.4.3) in retrieval on the CUB200, In-shop and Stanford Online Products datasets.

Method	RaMBO	λ	margin	AP^{50}
Faster R-CNN				74.2
Faster R-CNN	✓	0.5		74.6
Faster R-CNN	✓	0.1	✓	75.2
Faster R-CNN	✓	0.5	✓	**75.7**
Faster R-CNN	✓	2.5	✓	74.3

Table 3.8: Ablation for RaMBO on the object detection task.

3.5.4 Ablation Studies

We verify the validity of our loss design in multiple ablation studies. table 3.7 shows the relevance of margin and batch memory for the retrieval task. In fact, some of the runs without a margin diverged. The importance of margin is also shown for the mAP loss in table 3.8. Moreover, we can see that the hyperparameter λ of the scheme [334] does not need precise tuning. Values of λ that are within a factor 5 of the selected $\lambda = 0.5$ still outperform the baseline.

3.6 Discussion

In this chapter we proposed a method for rank-based metric optimization, which we name RaMBO, that is unique conceptual purity in directly optimizing for the desired metric while being simple, flexible, and computationally efficient. Driven only by basic loss-design principles and without serious engineering efforts, it can compete with state-of-the-art methods on image retrieval and consistently improve near-state-of-the-art object detectors. Exciting opportunities for future work lie in utilizing the ability to efficiently optimize ranking-metrics of sequences with millions of elements, to which the method should scale seamlessly, since it relies on efficient algorithms for the ranking operation.

I've been imitated so well I've heard people copy my mistakes.

Jimi Hendrix

4

Neuro-Algorithmic Policies Enable Fast Combinatorial Generalization

This chapter is based on the work "Neuro-algorithmic Policies Enable Fast Combinatorial Generalization" [335]. An interesting application area of blackbox differentiation is in the context of sequential decisions and control. In this chapter, by using theory from Chapter 2, we embed a time-dependent shortest path solver into a policy network. When we apply this in an imitation learning setting, we see significant improvements in generalization in dynamic environments.

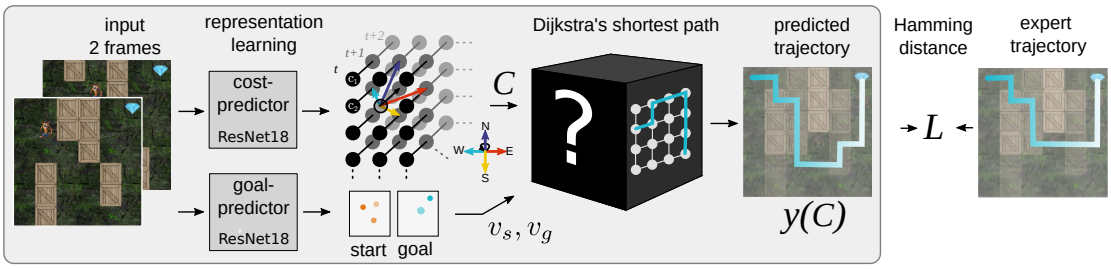


Figure 4.1: Architecture of the neuro-algorithmic policy. Two subsequent frames are processed by two simplified ResNet18s: the cost-predictor outputs a tensor (width $\times$ height $\times$ time) of vertex costs $C^t(v)$ and the goal-predictor outputs heatmaps for start and goal. The time-dependent shortest path solver finds the shortest path to the goal. Hamming distance between the proposed and expert trajectory is used as loss for training.

4.1 Motivation

In Chapter 3 we have focused on rank-based metric optimization in computer vision problems, building heavily upon the findings in Chapter 2. Now we turn our attention to another

problem that fits naturally into the realm of combinatorial problems of interest. One of the central topics in machine learning research is learning control policies for autonomous agents. Many different problem settings exist within this area. On one end of the spectrum are imitation learning approaches, where prior expert data is available and the problem becomes a supervised learning problem. On the other end lie approaches that require interaction with the environment to obtain data for policy extraction, posing the problem of exploration. Most RL algorithms fall into the latter category. In this chapter, we concern ourselves primarily with the setting where limited expert data is available, and a policy needs to be extracted by imitation learning. Independently of how a policy is extracted, a central question of interest is: how well will it generalize to variations in the environment and the task? Recent studies have shown that standard deep RL methods require exhaustive amounts of exposure to environmental variability before starting to generalize [67].

There exist several approaches addressing the problem of generalization in control. One option is to employ model-based approaches that learn a transition model from data and use planning algorithms at runtime or to improve value-learning. This has been argued to be the best strategy in the presence of an accurate model and sufficient computation time [74]. Furthermore, one can use the transition model alongside a reward model to generate offline data to improve value function learning [311, 148]. However, learning a precise transition model is often harder than learning a policy. The transition model often has a much larger dimensionality than the policy since it needs to model aspects of the environmental dynamics that are perhaps irrelevant for the task. This is particularly true for learning in problems with high-dimensional inputs, such as raw images. In order to alleviate this problem, learning specialized or partial models has shown to be a viable alternative, e.g. in MuZero [284].

By making use of the toolset that we have developed in Chapter 2, we construct a hybrid control policy architecture, which we call neuro-algorithmic policy. We will utilize a time-dependent shortest path (*Travelling Salesman Problem* (TDSP)) solver acting on a temporally evolving graph generated by a deep network from the inputs. By learning the time-evolving costs of the graph, a specific model of the system is built that is sufficient for planning. This choice is akin to goal-conditioned MDPs since the shortest path algorithm expects a goal to be reached. We demonstrate the effectiveness of this approach in an offline imitation learning setting where a few expert trajectories are provided. Due to the combinatorial generalization capabilities of planners, our learned policy is able to generalize to new variations in the environment out of the box and needs orders of magnitude fewer samples than naive learners. Using neuro-algorithmic architectures facilitates generalization by shifting the combinatorial aspect of the problem to efficient algorithms, while using neural networks to extract a good representation for the problem at hand. They have the potential to endow artificial agents with the main component of intelligence, the ability to reason.

4.2 Markov Decision Processes and Shortest Paths

Following from Section 1.4, we consider a special case of the MDP that allows us to formulate policies that solve a time-dependent shortest path problem on a latent graph representation. In order to achieve this, some conditions on the MDP need to be satisfied.

First, we consider discrete MDPs with deterministic transitions. In addition, we follow a goal-conditioned setting [281]. This is used in sequential decision making problems where a specific terminal state has to be reached.

Definition 4.2.1 (Goal-Conditioned MDP)**.** A goal-conditioned Markov Decision Process (gcMDP) $\mathcal{M}$ is defined by the tuple $(\mathcal{S}, \mathcal{A}, p, g, r)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $p(s' \mid a, s)$ the probability of making the transition $s \in \mathcal{S} \to s' \in \mathcal{S}$ when taking the action $a \in \mathcal{A}$, $g \in \mathcal{S}$ is the goal, $r(s, a, s', g)$ the reward obtained when transitioning from state s to s' while taking action a and aiming for goal g.

In episodic reinforcement learning, the objective is to find a policy that maximizes the return $G = \sum_{t=0}^{T} r_t$ of such a process. In gcMDPs, the reward is such that the maximal return can be achieved by reaching the goal state g, which is also the terminal state.

Definition 4.2.2 (Deterministic Discrete Goal-Conditioned MDP)**.** A discrete and deterministic goal-conditioned Markov Decision Process (ddgcMDP) $\ddot{\mathcal{M}}$ is a gcMDP with discrete and finite state space $\mathcal{S}$ and action space $\mathcal{A}$ with deterministic transitions, i.e. the probability mass $p(s' \mid a, s)$ is concentrated at a single point.

Let us consider the following graph representation (G, v_s, v_g): a weighted graph $G = (V, E, C)$ together with start vertex v_s and goal vertex v_g, where V is the vertex set, $E \subset (V \times V)$ is the edge set, and $C \in \mathbb{R}_+^{|E|}$ is the cost matrix with positive entries. We write $C(e)$ for the cost of edge e. In addition, we consider an inverse model $\psi \colon E \to \mathcal{A}$, associating an edge to an action.

A direct translation of a ddgcMDP to a graph is given by a bijective correspondence $\phi \colon \mathcal{S} \to V$ between states and vertices, for each possible transition (s, a, s') there is an edge $e = (\phi(s), \phi(s'))$ with $\psi(e) = a$, and the goal vertex is $v_g = \phi(g)$. The cost matrix C takes the role of the reward function with $C(\phi(s), \phi(s')) = c_{\text{max}} - r(s, a, s', g)$ where c_{max} is an upper bound on the reward ensuring positive cost values. To deal with variable or infinite episode lengths, g can be seen as an absorbing state where $r(g, \cdot, g, g) = c_{\text{max}}$, incurring a cost of 0 at the goal. Due to the deterministic nature of the ddgcMDP, the optimal policy yields a path with maximal return (sum of rewards) from start s_0 to goal g, which now coincides with the shortest path according to C by definition.

4.2.1 Factorized MDPs and Time-dependent Shortest Path

In many interesting environments, the state space is exponentially large, e.g. due to independent dynamics of entities. As an illustrative example, consider the game environment shown in Figure 4.1 and Figure 4.2 – an agent with moving obstacles. For such environments, the corresponding graph would become intractably large. We now define conditions under which the size of the graph can be drastically reduced, without losing the property that its shortest path solves the MDP. To this end, we assume the state space can be factorized as $\mathcal{S} = \mathcal{S}_A \times \mathcal{S}_E$, where $\mathcal{S}_A$ is affected only by the actions and $\mathcal{S}_E$ by the environment dynamics independent of the actions. We write the decomposition of each state as $s = (s^a, s^e)$. The mapping from state space to graph vertices $\phi \colon \mathcal{S} \to V$ is bijective only w.r.t. $\mathcal{S}_A$ and ignores $\mathcal{S}_E$, i.e. $\forall s^a \in \mathcal{S}_A \; \exists\, v \in V \colon \phi(s^a, s^e) = v \,, \forall\, s^e \in \mathcal{S}_E$. For brevity, we write $\phi(s^a)$. For instance, using $\mathcal{S}_A$ as the agent's positions on a discrete grid results in just as many vertices as there are positions of the agent.

Next, we show how a solution to this factorized ddgcMDP can be found by solving a time-dependent shortest path (TDSP) problem on the reduced graph. The cost matrix C is now a time-dependent quantity, i.e. $C \in \mathbb{R}_+^{H \times |E|}$ that assigns every edge e a positive cost value for each time $t \in \{1, 2, \ldots, H\}$ of the planning horizon/episode. To ease the notation, we write $C^t(e)$ for the cost of edge e at time t. The TDSP problem is defined as reaching the goal vertex v_g within at most H steps when starting at time step 1 in the start vertex v_s.

Two situations need to be modeled by the cost:[i] a) the environment dynamics can make an edge e become unavailable (for instance, an obstacle is moving in the way), then the cost should be infinite: $C^t(e) = \infty$, and b) the environment dynamics changes the reward, thus for all other edges we have

$$C^t(\phi(s^a), \phi(s'^a)) = c_{\max} - r((s^a, s_t^e), a, (s'^a, s_t^e)) \tag{4.1}$$

where $a = \psi(\phi(s^a), \phi(s'^a))$. Again, with this construction, the time-dependent shortest path solution coincides with the optimal policy of the specific ddgcMDP. We can also deal with stochastic environment dynamics as long as the action's effect on the state stays deterministic. This changes the reward term in eq. (4.1) to an expectation over environment dynamics,

$$\mathop{\mathbb{E}}_{s_t^e}[r((s^a, s_t^e), a, (s'^a, s_t^e))].$$

In our architecture, described below, the policy generates a latent graph at every observation/state and solves a ddgcMDP to optimality at every time step following a model predictive control approach using receding horizon planning.

[i] Note that learning the *exact* costs is not necessary, since we are only interested in optimal trajectories.

Algorithm 4 Forward and backward pass for the shortest-path algorithm

function FORWARDPASS(C, v_s, v_g)
  $y := \textbf{TDSP}(C, v_s, v_s)$ *// Run Dijkstra's algo.*
  save y, C, v_s, v_e *// Needed for backward pass*
  return y

function BACKWARDPASS($\nabla L(Y), \lambda$)
  load Y, C, v_s, v_e
  $C_\lambda := C + \lambda \nabla L(Y)$ *// Calculate modified costs*
  $y_\lambda := \textbf{TDSP}(C_\lambda, v_s, v_g)$ *// Run Dijkstra's algo.*
  return $\frac{1}{\lambda}\big(y_\lambda - y\big)$

4.3 Time Dependent Shortest Path Algorithm

In Chapter 2 we have seen how can we create differentiable layers from combinatorial solvers with dot-product cost formulations. As we shall see now, this framework is also applicable in the time-dependent shortest path case which we consider here. We aim to predict the cost matrix C via a deep neural network and train the entire system end-to-end via gradient descent on expert trajectories, as illustrated in Figure 4.1. We will employ an efficient implementation of Dijkstra's algorithm for computing the shortest path.

Time-dependent shortest path with vertex costs. In our neuro-algorithmic policy, we use the TIME-DEPENDENT-SHORTEST-PATH (TDSP) formulation based on vertex-costs instead of edge-costs, due to the reduction in cost matrix size by a factor of $|\mathcal{A}|$. The TDSP solver has as input the graph $G(V, E, C)$ with time-dependent cost matrix $C \in \mathbb{R}_+^{H\times|V|}$ and a pair of vertices $v_s, v_g \in V$ (start and goal). We write $C^t(v)$ for the cost of vertex v at time t.

This version of the shortest path problem can be solved by executing the Dijkstra's shortest path algorithm[ii] [85] on an augmented graph. In particular, we set

$$V^* = \{(v,t) \colon v \in V, t \in [1,H]\}$$
$$E^* = \{((v_1,t),(v_2,t+1)) \colon (v_1,v_2) \in E^{\circlearrowleft}, t \in [1,H-1]\},$$

where the cost of vertex $(v_i, t) \in V^*$ is simply $C^t(i)$ and $E^{\circlearrowleft}$ is the original edge set E appended with all self-loops. This allows to "wait" at a fixed vertex v from timestep t to timestep $t+1$. In this graph, the task is to reach the vertex (v_g, H) from $(v_s, 1)$ with the minimum traversal cost.

[ii] Even though the classical formulation of Dijkstra's algorithm is edge-based, all of its properties hold true also in this vertex-based formulation.

For this differentiation scheme to hold, we need to satisfy the conditions cost linearity condition (see Chapter 2), while we allow for arbitrary constraints. To that end, for a given graph $G = (V, E, C)$ as described above, we define $Y \in \{0, 1\}^{H \times |V|}$ an indicator matrix of visited vertices. In particular, $y_i^t = 1$ if and only if vertex v_i is visited at time point t. The set of such indicator matrices that correspond to valid paths in the graph (V^*, E^*) from start to goal will be denoted as $Y := \text{Adm}(G, v_s, v_g)$. The time-dependent shortest path optimization problem can be then rewritten as

$$\text{TDSP}(C, v_s, v_g) = \arg\min_{y \in Y} \sum_{(i,t)} y_i^t C_i^t, \tag{4.2}$$

where $Y = \text{Adm}(G, v_s, v_g)$. This is an inner-product objective and thus the theory from Chapter 2 applies and allows us to learn the cost-matrix generating network with gradient descent.

4.4 Neuro-Algorithmic Policy Framework

Here we introduce the NAPframework, which is an end-to-end trainable deep policy architecture embedding an algorithmic component using the afore-mentioned techniques. Following the definitions from Section 1.4, we concern ourselves with learning the mapping $\varphi_\theta : \mathcal{S} \mapsto \mathbb{R}^{|V| \times T} \times V \times V$, i.e. mapping from MDP states to cost matrices and respective start and end vertices for the TDSP problem[iii] of planning horizon H. This enables us to construct the policy

$$\pi_\theta := \psi \circ \text{TDSP} \circ \varphi_\theta. \tag{4.3}$$

The mapping φ_θ can be decomposed into φ_θ^c (*cost-predictor*), φ_θ^s (*start-predictor*), φ_θ^g (*goal-predictor*), i.e. mappings from state to costs, start vertex and goal vertex. In practice, instead of learning φ_θ^s and φ_θ^g directly, we learn the conditional probability densities $p_\theta^s(v|s)$ and $p_\theta^g(v|s)$.

In this work, we examine the application of neuro-algorithmic policies to the imitation learning setting, where we have access to trajectories τ sampled from the expert policy distribution $\eta(\pi^*)$. Given a fixed planning horizon H, the objective that we consider consists of three main parts, the latent cost term, start vertex term and goal vertex term. The latent cost objective is defined as

$$\ell^C(\theta, H) = \mathop{\mathbb{E}}_{\tau_{\text{t:t+H}} \sim \eta(\pi^*)} \left[\text{d}^{\text{H}}(\text{TDSP}(\varphi_\theta(\tau_t)), \phi(\tau)) \right], \tag{4.4}$$

[iii] We hold the graph structure fixed, namely the set of edges E and learn the costs C. Therefore we replace G with costs C in the TDSP solver to simplify the notation.

where $\mathrm{d}^{\mathrm{H}}(\cdot,\cdot)$ denotes the Hamming distance between predicted and expert paths in the latent graph, and $\phi' \colon \mathcal{S} \mapsto V$ the mapping of the expert-visited states to latent graph vertices. The second part of the objective is a standard cross-entropy term for the start and goal vertices that allows us to train a *start-* and *goal-predictor*:

$$\ell^P(\theta, H) = \mathop{\mathbb{E}}_{\tau_{\mathrm{t:t+H}} \sim \eta(\pi^*)} \big[-\log p^s_\theta(\phi(\tau_t)|\tau_t) - \log p^g_\theta(\phi'(\tau_{t+H})|\tau_t) \big]. \tag{4.5}$$

We assume access to ϕ' at training time in order to map the expert to the latent graph for calculation of J^C and J^P. Finally, we optimize for the sum of J^C and J^P. The cost matrix C is given to the solver along with the start vertex v_s and end vertex v_e to compute the time-dependent shortest path Y. The *cost-predictor* is trained using the Hamming distance between the predicted plan Y and the expert plan Y^* that we use for supervision.

Given a planning horizon T and parameters θ, we optimize for the objective

$$\begin{aligned} \ell(\theta, T) = \mathbb{E}_{\tau_{t:t+T} \sim \eta^{\pi^*}} \big[& \\ & + \mathrm{Hamm}(\mathrm{TDSP}(\varphi_\theta(\tau_t)), \phi'(\tau)) \\ & - \log p^s_\theta(\phi'(\tau_t)) - \log p^e_\theta(\phi'(\tau_{t+T})) \big] \end{aligned} \tag{4.6}$$

We utilize a concrete architecture consisting of two main components: a backbone ResNet18 architecture (without the final fully connected layers (a detailed description is available in Supplementary C.3 in the supplementary) and the shortest path solver, see Figure 4.1. At each time step, the policy receives two images concatenated channel-wise from which it predicts the cost tensor C for the planning horizon H with the *cost-predictor*, the start vertex v_s and goal vertex v_g with the *goal-predictor*, explained below.

The policy is used in a model-predictive control fashion, i.e. at execution time, we predict the plan Y for horizon H at each time step and execute the first action from the plan.

4.4.1 Global and Local Goal Prediction

In order to apply the solver to the learned latent graph representation, we need to map the current state of the environment to appropriate start and goal vertices (v_s, v_g). To this end, we employ a second ResNet18 similar to the *cost-predictor* that approximates $p^s(v|s)$ and $p^g(v|s)$, i.e. the start and goal conditional densities.

At training time, given the expert trajectories, we have access to the mapping ϕ' that maps the expert trajectory to the latent graph. In the *global* setting, the last position of the expert is the goal v_g, corresponding to, for instance, the jewel in CRASH JEWEL HUNT which is also the terminal state, see Figure 4.2.

In the *local* setting, we expect the end vertex to be an intermediate goal (for instance "collect an orb"), which effectively allows for high-level planning strategies, while the low-level

planning is delegated to the discrete solver. In this case, the positively labeled supervision at time t are all locations of the (expert) agent between step $t + H$ and $t + 2H$.

The local setting allows to limit the complexity of our method, which grows with the planning horizon. This is also a trade-off between the combinatorial complexity solved by the TDSP solver and the goal predictor. Ideally, the planning horizon H used for the cost-prediction is long enough to capture the combinatorial intricacies of the problem at hand, such as creating detours towards the goal in the case of future dangerous states, or avoiding dead-ends in a maze.

This formulation makes our architecture akin to hierarchical methods similar to Blaes et al. [37] and Nachum et al. [223], and allows for solving tasks that are not typical goal-reaching problems, such as the CHASER environment.

4.5 Experiments

To validate our hypothesis that embedding planners into neural network architectures leads to better generalization in control problems, we consider several procedurally generated environments (from the ProcGen suite [67] and CRASH JEWEL HUNT) with considerable variation between *levels*.

We compare with the following baselines: a standard *Behavior Cloning* (BC) baseline using a ResNet18 architecture trained with a cross-entropy classification loss on the same dataset as our method; the PPO algorithm as implemented in Cobbe et al. [67] and DrAC [255]. A comparison to DrAC is especially interesting, since the method claims to improve generalization by applying optimized data augmentations. As there are multiple variations of data augmentation suggested by Raileanu et al. [255], we run all of them and select the best result as DrAC* for performance comparison to our method.[iv] We also ablate the *start-* and *goal-predictor* and replace with the ground truth vertices, this serves as an upper baseline for NAP which we denote with NAP*. More details on the training procedure and the hyperparameters can be found in Supplementary C.4.

For the experimental validation, we aim to answer the following questions: **(i)** Can NAP be trained to perform well in procedurally generated environments? **(ii)** Can NAP generalize in a low data regime, i.e. after seeing only few different levels? **(iii)** Can we also solve non-goal-reaching environments?

[iv] We defer a more detailed description of DrAC along with performance plots to Supplementary C.6

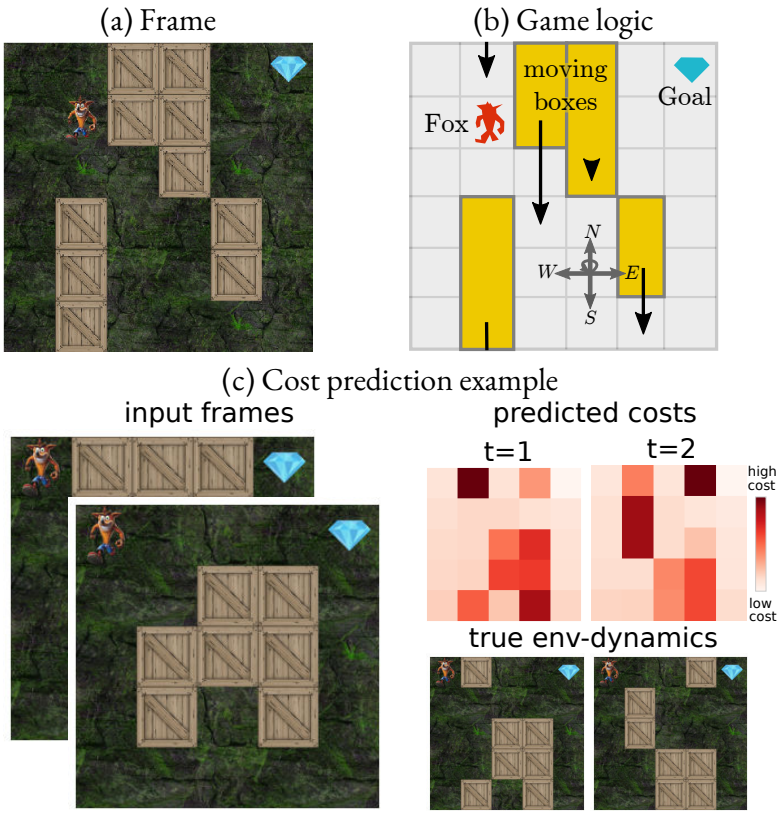


Figure 4.2: The CRASH JEWEL HUNT environment. The goal for the fox, see (a), is to obtain the jewel in the right most column, while avoiding the moving wooden boxes (arrows in (b)). When the agent collides with a wooden box it instantly fails to solve the task. We observe that the predictions of the costs in (c) are highly interpretable, corresponding to (future) movements of the boxes.

4.5.1 CRASH JEWEL HUNT

For proof of concept, we first consider an environment we constructed to test NAP, called CRASH JEWEL HUNT which can be seen in Figure 4.2. The environment corresponds to a grid-world of dimensions *height* $\times$ *width* where the goal is to move the agent (Fox) from an arbitrary start position in the left-most column to the goal position (jewel) arbitrarily positioned in the right-most column. Between the agent and the goal are obstacles, wooden boxes that move downwards (with cyclic boundary conditions) with velocities that vary across levels but not within a level, see Figure 4.2 (right). At each time step, the agent can choose to move horizontally or vertically in the grid by one cell or take no action.

To make the task challenging, we sample distinct environment configurations for the training set and the test set, respectively. More concretely, we vary the velocities, sizes and initial

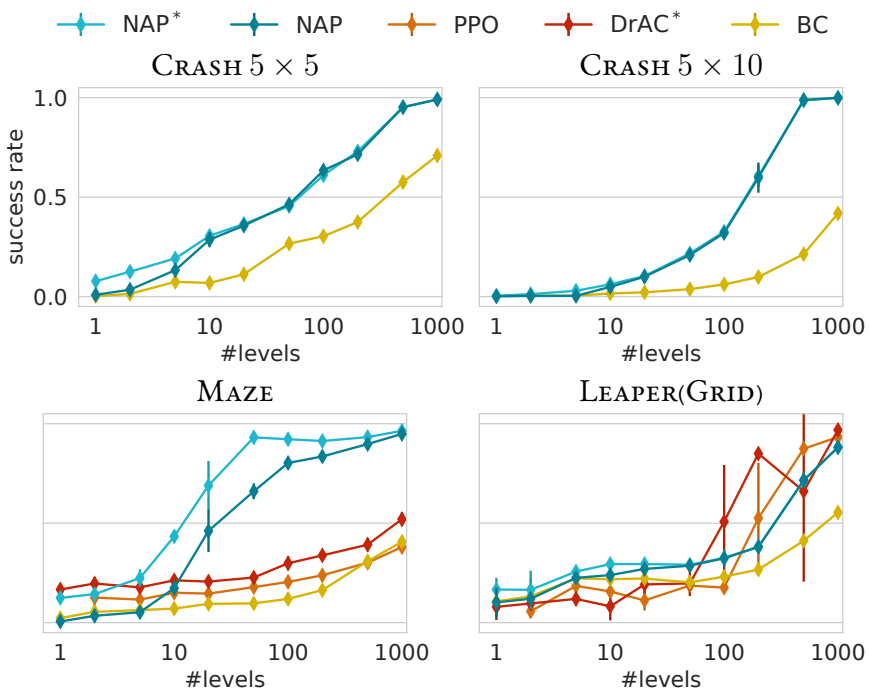


Figure 4.3: Performance on unseen test levels for a different number of training levels. We observe that NAP already shows signs of generalization after only being trained on 100 levels.

positions of the boxes as well as the start and goal positions.

4.5.2 ProcGen Benchmark

In addition to the jewel hunt environment, we evaluate our method on the hard version Maze, Leaper and Chaser environments from the ProcGen suite [67]. We have chosen these environments because their structure adheres to our assumptions. For Leaper, we modified the environment such that grid-world dynamics applies and denote it as Leaper(Grid).

The Maze and the Leaper(Grid) tasks have a static goal whose position only varies across levels, whereas the Chaser requires collection of all orbs without contacting the spiders, so the local goals need to be inferred on the fly. The Chaser environment is also *particularly challenging* as even the expert episodes require on average 150 steps, most of which carry the risk of dying. For this reason, we used three human expert trajectories per level.

4.5.3 Results

We train our method (NAP) and the imitation learning baseline until saturation on a training set, resulting in virtually 100% success rate when evaluating on train configurations in the environment. For the PPO baseline we use the code from [67] and provide also two subsequent frames and 200M time steps for training. For the DrAC baselines we use the code from Raileanu et al. [255]. For our method, we also report performance of a version with access to the true start and end-point prediction (NAP*), with the exception of the CHASER where true goals are not well-defined.

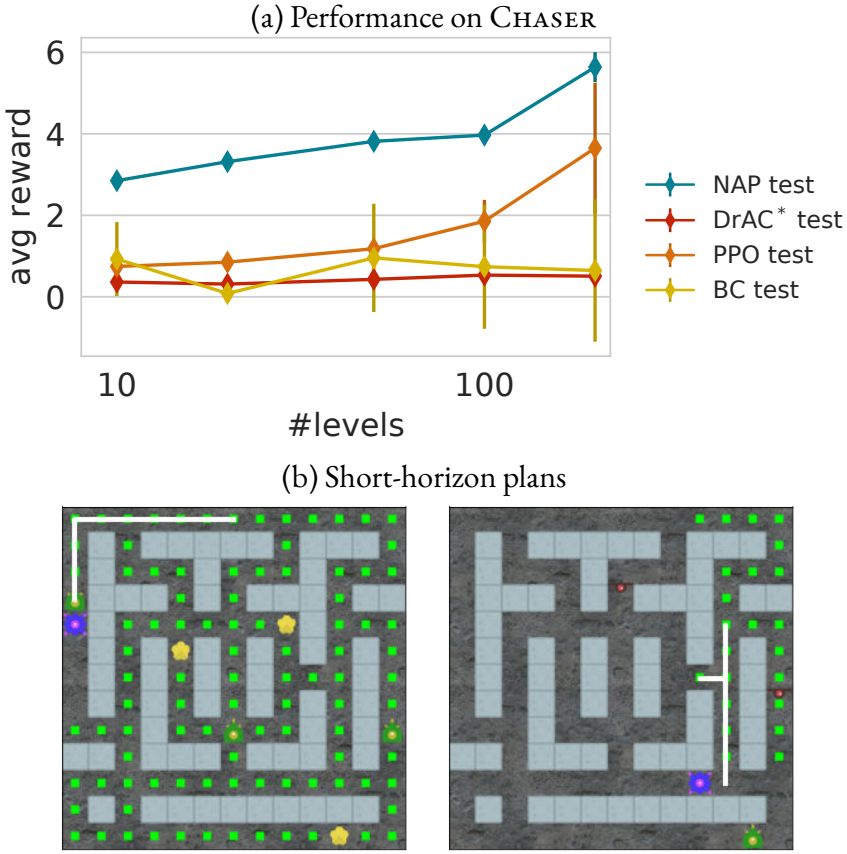


Figure 4.4: Performance of NAP against PPO on the CHASER environment (a) trained on 10, 20, 50, 100 and 200 levels. In (b) we show the short-horizon plans (white) of the agent (blue) at step 5 and 110 in the environment.

In Figure 4.3, we show the performance of the methods when exposed to a different number of levels at training time. As reported in Cobbe et al. [67], the baselines have a large

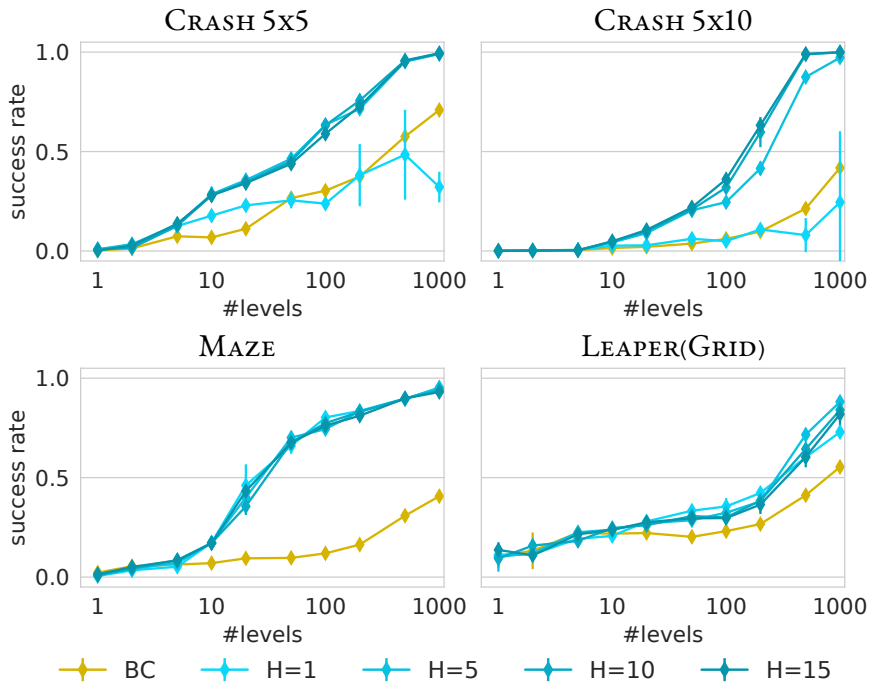


Figure 4.5: Test success rate of our method with different horizon lengths. The solver assumes that the last horizon step costs remain to infinity. In this sense, the horizon of length 1 corresponds to a static solver.

generalization gap and poor performance when $< 10\,000$ levels are seen.

We find that NAP shows strong generalization performance already for < 500 levels. In some environments, such as Maze we obtain near 80% success rate already with just 100 levels which is never reached by PPO or DrAC even after seeing a 1000 levels. Our NAP trained on 1000 levels reaches the same performance as PPO trained on 200 000 levels of the Maze environment. For Crash Jewel Hunt 5×5, already with 30 trajectories a third of the 1000 test-levels can be solved, while the behavior cloning baseline manages less than 50 out of the 1000.

In Chaser we learn a subgoal predictor from expert trajectories that predicts a subgoal for the TDSP algorithm at each time step, since there is no clear goal state specified by the environment. The performance plots in Figure 4.4 show that NAP outperforms the baselines. Additionally, in Figure 4.4(b) we observe that using NAP the agent is able to perform detours to collect orbs, indicating that the learned costs in the latent planning graph reflect the actual task.

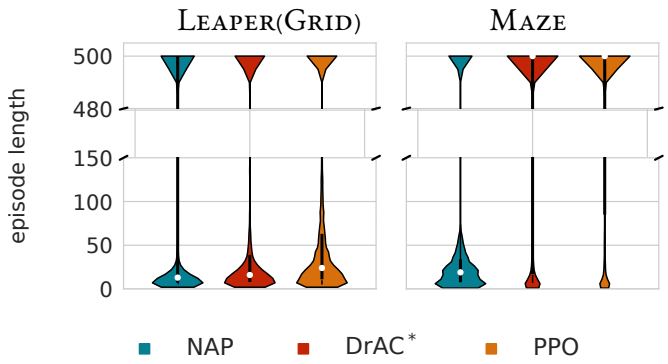


Figure 4.6: Distributions of test level episode lengths after training on 500 levels for the LEAPER(GRID) and MAZE environments over 3 seeds. We observe that NAP tends to take shorter paths than DrAC* and PPO. We also observe that the median episode length for the baselines in the MAZE task is located in the upper part, which corresponds to unsolved levels.

4.5.4 Sensitivity to the Planning Horizon

We provide a sensitivity analysis of the performance with different planning horizons. Our results indicate that longer horizons benefit environments with increased dynamical interactions. As apparent from Figure 4.5, our method outperforms the behavior cloning baseline in all of the environments, the gap between the methods being correlated with the complexity of the environment. It can also be seen that making the planning horizon smaller in environments with dynamics hurts performance.

On the other hand, for environments with no dynamics, such as the maze environment, there is no benefit in using a longer planning horizon, as expected. Nevertheless, there is still strong performance gain in generalization when using NAP as opposed to vanilla imitation learning from expert trajectories thanks to the introduced combinatorial inductive bias.

4.5.5 Path Optimality

We have observed that introducing combinatorial structural biases with a method such as NAP can benefit generalization greatly. Another interesting question is how optimal are the paths taken by NAP. Even though NAP has an advantage in having access to expert optimal or near-optimal trajectories, this does not necessarily translate to being able to act optimally in unseen situations.

For this analysis we have plotted the episode length distribution across runs for NAP, DrAC* and PPO, which can be seen in Figure 4.6. This shows us that NAP generalizes with more optimal (shorter) paths in comparison to DrAC* and PPO, even when compared only to levels solved by all methods. Even in the LEAPER(GRID) environment, where DrAC* out-

performs NAP by measure of success rate, NAP still outperforms DrAC* in terms of path lengths.

4.6 Discussion

We have shown that hybrid neuro-algorithmic policies consisting of deep feature extraction and a shortest path solver – made differentiable by the techniques that we developed in Chapter 2 – *enable learning policies that generalize to unseen environment settings in the low-data regime*. Hybrid architectures are a stepping stone towards the usage of better inductive biases that enable stronger generalization. In NAP, the inductive bias that we impose is the topology of the latent planning graph in conjunction with a planning algorithm. Introducing the shortest-path solver as a module shifts the combinatorial complexity of the planning problem to efficient algorithmic implementations while facilitating the learning of interpretable representations for planning in the latent space.

Although there is a clear benefit in using NAP, the method comes with certain caveats. We assume that the topological structure (i.e. that there is an underlying grid structure with a set of 5 actions) of the latent planning graph is known a priori. Furthermore, we assume that the structure of the latent graph is fixed and not dynamically changing over time, i.e. that each available action at a vertex corresponds to the same edge. Any results allowing to abandon some of these assumptions will vastly increase the applicability of this method.

II

Decision Making under Constraints

In order to seek truth, it is necessary once in the course of our life to doubt, as far as possible, of all things.

René Descartes

5 Risk-Averse Zero-Order Trajectory Optimization

This chapter is based on the work "Risk-averse Zero-Order Trajctory Optimization" [330]. We propose a model-based method for epistemic and aleatoric uncertainty separation which can be utilized for risk-averse planning and exploration while making use of probabilistic safety constraints.

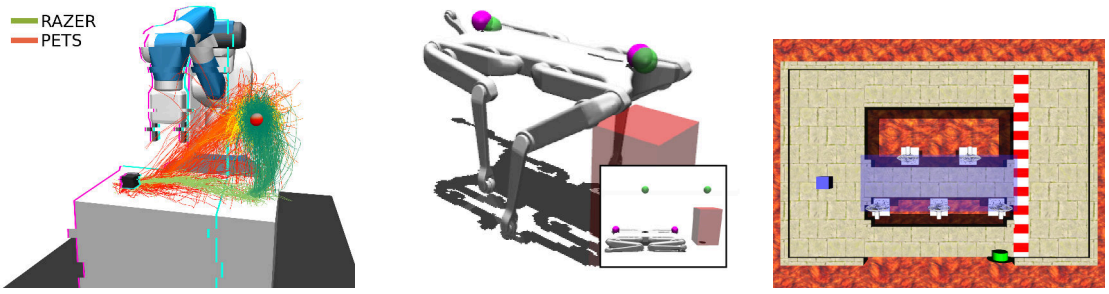
5.1 Motivation

In this chapter we turn our attention to the problem of planning and making decisions in a dynamical system. Data-driven approaches to sequential decision-making are becoming increasingly popular [354, 145, 249, 285]. They hold the promise of reducing the number of prior assumptions about the system that are imposed by traditional approaches that are based on nominal models.

Such approaches come in several different flavors [162]. Model-free approaches attempt to extract closed-loop control policies directly from data, while model-based approaches rely on a learned model of the dynamics to either generate novel data to extract a policy or to be used in a model-predictive control fashion (MPC). This work belongs to the latter line of work.

Model-based methods have several advantages over pure model-free approaches. Firstly, humans tend to have a better intuition on how to incorporate prior knowledge into a model rather than into a policy or value function. Secondly, most model-free policies are bounded to a specific task, while models are task-agnostic and can be applied for optimizing arbitrary cost functions, given sufficient exploration.

Nevertheless, learning models for control comes with certain caveats. Traditional MPC methods require the model and cost function to permit a closed-form solution which restricts the function class prohibitively. Alternatively, gradient-based iterative optimization can be employed, which allows for a larger class of functions but typically fails to yield satisfactory solutions for complicated function approximators such as deep neural network models. In addition, calculating first-order or even second-order information for trajectory optimization tends to be computationally costly, which makes it hard to meet the time constraints of real-world settings. This motivates the usage of zero-order methods, i.e gradient-free or sample-based, such as the CEM that do not rely on gradient information but are efficiently parallelizable.



(a) Noisy-FetchPickAndPlace **(b)** Solo8-LeanOverObject **(c)** BridgeMaze

Figure 5.1: Environments considered for uncertainty-aware planning. Code and videos are available at https://martius-lab.github.io/RAZER/.

Many methods relying on a learned model and zero-order trajectory optimizers have been proposed [65, 338, 345], but all share the same problem: compounding of errors through auto-regressive model prediction. This naturally brings us to the question of how can we effectively manage model errors and uncertainty to be more data-efficient and safe. Arguably, this is one of the main obstacles to applying data-driven model-based methods to the real world, e.g. to robotics settings.

In this work, we introduce a risk-averse zero-order trajectory optimization method (RAZER) for managing errors and uncertainty in zero-order MPC and test it on challenging scenarios (Figure 5.1). We argue that it is essential to differentiate between the two types of uncertainty in the model-predictive setting: the aleatoric uncertainty arising from inherent noise in the system and epistemic uncertainty arising from the lack of knowledge [142, 159]. We measure these uncertainties by making use of probabilistic ensembles with trajectory sampling similar to PETS [65]. Our contributions can be summarized as follows: (i) method for separation of uncertainties in probabilistic ensembles (PETSUS); (ii) efficient use of aleatoric and epistemic uncertainty in model-based zero-order trajectory optimizers; (iii) a simple but practical approach to probabilistic safety constraints in zero-order MPC.

5.2 Related Work

Uncertainty Estimation In the typical *Model-Based Reinforcement Learning* (MBRL) setting, the true transition dynamics function is modeled through an approximator. Impressive results have been achieved by both parametric models [184, 101, 103, 127], such as neural networks, and nonparametric models [164, 228, 117, 77], such as Gaussian Processes (GP). The latter inspired seminal work on the incorporation of the dynamics model's uncertainty for long-term planning [77, 150]. However, their usability is limited to low-data, low-dimensional regimes with smooth dynamics [259, 260], which is not ideal for robotics applications. Alternative parametric approaches include ensembling of deep neural networks, used both in the MBRL community [65, 174], and outside [236, 175]. In particular, ensembles of *probabilistic* neural networks established state-of-the-art results [65], but focus mainly on estimating the expected cost and disregard the underlying uncertainties. In comparison, we propose a treatment of the resulting uncertainties of the ensemble model.

Zero-order MPC The learned model can be used for policy search like in PILCO [78, 77, 150, 69] or for online model-predictive control (MPC) [217, 347, 65]. In this work, we do planning in an MPC fashion and employ a zero-order method as a trajectory optimizer, since they have shown to be less likely to get stuck in local minima and make an explicit treatment of the uncertainty in the cost possible. Specifically, we consider a sample-efficient implementation of the Cross-Entropy method [274, 40] introduced in [247].

Safe MPC Separating the sources of uncertainty is of particular importance for applications directly affecting humans' safety, as self-driving cars, elderly care systems, or in general any application that involves a physical interaction between agents and humans. Disentangling epistemic from aleatoric uncertainty allows for separate optimization of the two, as they represent semantically different objectives as per definition. Extensive research on uncertainty decomposition has been done in the Bayesian setting and the context of safe policy search [211, 104, 81, 80], MPC planning [15, 182, 1], and distributional RL [66, 362]. On the other hand, a state-of-the-art baseline for ensemble learning like PETS [65], despite estimating uncertainty, only optimizes for the *expected* cost during action evaluation. Our work aims at filling this gap by explicitly integrating the propagated uncertainty information in the zero-order MPC planner.

5.3 Method

Our approach concerns itself with the efficient usage of uncertainties in zero-order trajectory optimization and is therefore generally applicable to such optimizers. We are interested in modeling noisy system dynamics $s_{t+1} = f(s_t, a_t, w(s_t, a_t))$ where f is a nonlinear function,

s_t the observation vector, a_t applied control input and $w(s_t, a_t)$ a noise term sampled from an arbitrary distribution.

Consequently, in the absence of prior knowledge about the function f, the system needs to be modeled by a complex function approximator such as a neural network. Furthermore, we are interested in managing uncertainties based on our fitted model, which is erroneous. To this end, we use stochastic ensembles of size K, where the output of each model $\boldsymbol{\vartheta}^k(s_t, a_t)$ are parameters of a normal distribution depending on input observation s_t and control a_t. As a by-product, our auto-regressive model prediction based on sequence of control inputs $\boldsymbol{a}$ becomes a predictive distribution over trajectories $\tau = (s_0, a_0, s_1, a_1 \dots)$; $\psi^\tau(s_t, \boldsymbol{a}) \coloneqq p(\tau | s_t, \boldsymbol{a}; \theta)$ where θ denotes the parameters of the ensemble. For convenience, from this point onward we will differentiate between multiple usages of ψ^τ. We denote with $\psi^s_{\Delta t}$ the distribution $p(x_{t+\Delta t} | s_t, \boldsymbol{a}_{t:t+\Delta t-1}; \theta)$ over states at time step $t + \Delta t$, $\psi^{\boldsymbol{\vartheta}}_{\Delta t}$ the distribution over the Gaussian parameter outputs $p(\boldsymbol{\vartheta}_{t+\Delta t} | s_t, \boldsymbol{a}_{t:t+\Delta t-1}; \theta)$ at time step $t + \Delta t$ of the planner.

5.3.1 Planning and Control

To validate our hypothesis that accounting for uncertainty in the environment and model prediction is essential to develop risk-averse policies, we use the Cross-Entropy Method (CEM) with improvements suggested in Pinneri et al. [247]. Accordingly, at each time step t we sample a finite number of control sequences $\boldsymbol{a}$ for a finite horizon H from an isotropic Gaussian prior distribution which we evaluate from the state s_t using an auto-regressive forward-model and the cost function. The sampling distribution is refitted in multiple rounds based on the best performing trajectories. After this optimization step, the first action of the mean of the fitted Gaussian distribution is executed. Since this approach utilizes a predictive model for a finite horizon at each time step, it naturally falls into the category of MPC methods.

Although we use CEM, our approach of managing uncertainty can generically be applied to other zero-order trajectory optimizers such as MPPI [347] by a modification of the trajectory cost function.

5.3.2 The Problem of Uncertainty Estimation

Since we have a stochastic model of the dynamics, at the model prediction time step t we observe a distribution over potential outcomes. Indeed, since our model outputs are parameters of a Gaussian distribution, with auto-regressive predictions we end up with a distribution over possible Gaussians for a certain time step t.

Given a sampled action sequence $\boldsymbol{a}$ and the initial state s_t we observe a distribution over trajectories ψ_τ. To efficiently sample from the trajectory distribution ψ_τ we use the technique introduced by Chua et al. [65] (PETS) which involves prediction particles that are sampled

from the probabilistic models and randomly mixed between ensemble members at each prediction step. In this way, the sampled trajectories are used to perform a Monte Carlo estimate of the expected trajectory cost $\mathbb{E}_{\tau\sim\psi^\tau}[c(\tau)]$. However, this does not take the properties of ψ^τ into account, which might be a high-entropy distribution and may lead to very risky and unsafe behavior. In this work, we alleviate this by looking at the properties of ψ^τ, i.e. different kinds of uncertainties arising from the predictive distribution.

5.3.3 Learned Dynamics Model

We learn a dynamics model f_θ that approximates the true system dynamics

$$s_{t+1} = f(s_t, a_t, w(s_t, a_t)).$$

As a model class, we use an ensemble of neural networks with stochastic outputs as in Chua et al. [65]. Each model k, parametrizes a multivariate Gaussian distribution with diagonal covariance, $f_\theta^k(s_t, a_t) = \mathcal{N}(s_{t+1}; s_t + \mu_\theta^k(s_t, a_t), \Sigma_\theta^k(s_t, a_t))$ where $\mu_\theta^k(\cdot,\cdot)$ and $\Sigma_\theta^k(\cdot,\cdot)$ are model functions outputting the respective parameters.

Iteratively, while interacting with the environment, we collect a dataset of transitions $\mathcal{D}$ and train each model k in the ensemble by the following negative log-likelihood loss on the Gaussian outputs:

$$\ell(\theta, k) = \mathop{\mathbb{E}}_{s_t,a_t,s_{t+1}\sim\mathcal{D}}\Big[-\log\mathcal{N}(s_{t+1}; s_t + \mu_\theta^k(s_t, a_t), \Sigma_\theta^k(s_t, a_t))\Big] \tag{5.1}$$

In addition, we use several regularization terms to make the model training more stable. We provide more details on this in Supplementary D.1.1.

5.3.4 Separation of Uncertainties

In the realm of parametric estimators, two uncertainties are of particular interest. *Aleatoric* uncertainty is the kind that is irreducible and results from inherent noise of the system, e.g. sensor noises in robots. On the other hand, we have *epistemic* uncertainty resulting from lack of data or knowledge which is reducible. This begs the question, how can we separate these uncertainties given an auto-regressive dynamics model f_θ? The way that we efficiently sample from ψ^τ is by mixing sampled prediction particles. This process is illustrated by the red lines in Fig. 5.2.

Simple model prediction disagreement is not a good measure for *aleatoric* uncertainty since it can be entangled with epistemic uncertainty. Given our how we model the system dynamics, we measure *aleatoric* uncertainty as entropy of the predicted normal distributions across ensemble members. More concretely, given a sampled particle state $\tilde{s}_t$, we define the

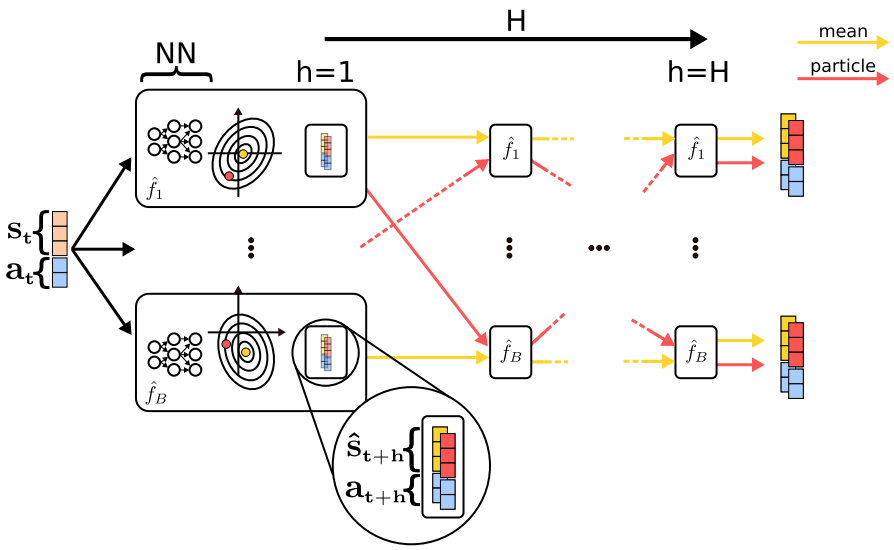


Figure 5.2: Probabilistic Ensembles with Trajectory Sampling and Uncertainty Separation (PETSUS)

estimated aleatoric uncertainty for an ensemble member associated to particle b at time step t as:

$$\mathfrak{A}_b(x|\tilde{s}_t, a_t) = \mathcal{H}_{x\sim\psi^s_{\Delta t,b}}(x) \tag{5.2}$$

Where $\psi^s_{\Delta t,b}$ is the output distribution of ensemble model based on inputs $\tilde{s}_t, a_t$. Since in the end we are interested in the aleatoric uncertainty incurred from applying the action sequence $\boldsymbol{a}$ from initial state s_t, the quantity of interest for us is the expected aleatoric uncertainty for time slice t:

$$\mathfrak{A}(s|a_t) = \mathop{\mathbb{E}}_{\tilde{s}_b\sim\psi^s_{\Delta t}}\Big[\mathfrak{A}_b(s|\tilde{s}_b, a_t)\Big] \tag{5.3}$$

Intuitively, because we only have access to the ensemble for sampling, we take a time-slice in the sampled trajectories from ψ^τ and compute the output entropies. Moreover, since we assume a Gaussian 1-step predictive distribution this is an expectation over differential Gaussian entropy. An alternative way of computation which we also explore is calculating the expected particle variance for time slice $t+1$ of the prediction horizon

$$\mathrm{Var}^{\mathfrak{A}}_{t+1} = \frac{1}{B}\sum_{b=1}^{B}\Sigma^k_\theta(\tilde{s}_{t,b}, a_t) \tag{5.4}$$

Note that $\Sigma^k_\theta(\tilde{s}_{t,b}, a_t)$ outputs the covariance of the prediction at $t+1$.

For estimating the *epistemic* uncertainty, one would be tempted to look at the disagreement between ensemble models in parameter space $\mathrm{Var}[\theta]$, but this is not completely satisfying,

since neural networks tend to be over-parametrized and variance within the ensemble still may exist albeit the optimum has been reached by all ensemble models.

Given the assumption of local Gaussianity, the true epistemic uncertainty for this case is the predictive entropy over the Gaussian parameters $\boldsymbol{\vartheta}$ at time step $t + h$.

$$\mathfrak{E}(s_t, \boldsymbol{a}_{t:t+\Delta t-1}) = \mathcal{H}_{\psi^{\boldsymbol{\vartheta}}_{\Delta t}}(\boldsymbol{\vartheta} \mid s_t, \boldsymbol{a}_{t:t+\Delta t-1}) \tag{5.5}$$

It is easy to verify that this quantity is 0 given perfect predictions of the model. Note that, because of auto-regressive predictions of a nonlinear model, this is a very difficult object to handle. Nevertheless, since our predictive distribution $p(s|s_t, a_t; \boldsymbol{\vartheta})$ is parametrized by model output, we may utilize disagreement in $\boldsymbol{\vartheta}_t$ to approximate $\mathfrak{E}$. To get correct estimations, we need to propagate mean predictions $\bar{s}$ in addition to the particles as illustrated as the yellow lines in Fig. Figure 5.2. We quantify epistemic uncertainty as ensemble disagreement at time step t:

$$\mathrm{Var}^{\mathfrak{E}}(s_{t+1}) = \mathrm{Var}^{e}[\mu^k_\theta(\bar{s}_t, a_t)] + \mathrm{Var}^{e}[\Sigma^k_\theta(\bar{s}_t, a_t)] \tag{5.6}$$

where Var^e is the empirical variance over the $k = 1 \dots K$ ensembles.

5.3.5 Probabilistic Safety Constraints

When applying data-driven control algorithms to real systems, safety is of utmost importance. In the realm of zero-order optimization, safety constraints can be easily introduced by putting an infinite cost on constraint-violating trajectories. Nevertheless, we are dealing with erroneous stochastic nonlinear models which lead to nontrivial predictive distributions of future states, based on the control sequence $\boldsymbol{a}$. For this reason, we want to control the risk of violating the safety constraints that we, as practitioners, are willing to tolerate. If we denote the observation space as $\mathcal{S}$, given a violation set $\mathbb{C} \subset \mathcal{S}$, we define the probability of the control sequence $\boldsymbol{a}$ to enter the violation set at time $t+\Delta t$ as $p(s \in \mathbb{C} \mid s_t, \boldsymbol{a}) = \int_{s \in \mathbb{C}} \psi^s_{\Delta t}(s \mid s_t, \boldsymbol{a})$. In practice, it is hard to compute this integral efficiently, since our distribution $\psi^s_{\Delta t}$ is nontrivial as a result of nonlinear propagation of uncertainty. Furthermore, the violation set $\mathbb{C}$ might not have the structure necessary to allow an efficient solution to the integral, in which case one needs to resort to Monte Carlo estimation.

To simplify computation and gain speed, we consider box violation sets resulting in each dimension of x being constrained to be outside of $[a, b] \in \{a, b \,|\, a, b \in \mathbb{R}^2,\ a < b\}$. By performing moment matching by a Gaussian in each time-slice $\psi^s_{\Delta t}$, the probability of ending up in state s at time step $t + \Delta t$ is given by integrating $\mathcal{N}(x; \mu_{t+\Delta t}, \Sigma_{t+\Delta t})$, where μ and Σ are estimated by Monte Carlo sampling. If we further assume a diagonal covariance Σ, this integral can be deconstructed into d univariate Gaussian integrals, which can be computed

fast and in closed form. Hence, the probability of a constraint violation happening at time step t is defined by:

$$p(s \in \mathbb{C} \mid s_t, \boldsymbol{a}) = \prod_{i=0}^{d} \int_{s \in \mathbb{C}} \mathcal{N}(s^i; \mu^i_{t+\Delta t}, \sigma^i_{t+\Delta t}) \tag{5.7}$$

5.3.6 Implementing Risk-Averse ZERo-Order Trajectory Optimization (RAZER)

We assume the task definition is provided by the cost $c(s_t, \boldsymbol{a})$. For trajectory optimization, we start from a state x_t and predict with an action sequence $\boldsymbol{a}$ the future development of the trajectory τ. Along this trajectory, we want to compute a single cost term which is conveniently defined as the expected cost of all particles $\tilde{s}$ summed over the planning horizon H:

$$c(s_t, \boldsymbol{a}) = \sum_{\Delta t=1}^{H} \frac{1}{B} \sum_{b=1}^{B} c(\tilde{s}^b_{t+\Delta t}, a_{t+\Delta t}). \tag{5.8}$$

The optimizer, in our case CEM, will optimize the action sequence $\boldsymbol{a}$ to minimize the cost in a probabilistic sense, i.e. $p(\boldsymbol{a} \mid s) \propto \exp(-\beta\, c(s, \boldsymbol{a}))$ where β reflects the strength of the optimizer (the higher the more likely it finds the global optimum). To make the planner uncertainty-aware, we need to make sure it avoids unpredictable parts of the state space by making them less likely. Using the aleatoric uncertainty provided by PETSUS eq. (5.4), we define the aleatoric penalty as

$$c_{\mathfrak{A}}(s_t, \boldsymbol{a}) = w_{\mathfrak{A}} \cdot \sum_{\Delta t=1}^{H} \sqrt{\mathrm{Var}^{\mathfrak{A}}_{t+\Delta t}}, \tag{5.9}$$

where $w_{\mathfrak{A}} > 0$ is a weighting constant. The larger the aleatoric uncertainty, the higher the cost.

To guide the exploration to states where the model has epistemic uncertainty eq. (5.6) (due to lack of data), we use an epistemic bonus:

$$c_{\mathfrak{E}}(s_t, \boldsymbol{a}) = -w_{\mathfrak{E}} \cdot \sum_{\Delta t=1}^{H} \sqrt{\mathrm{Var}^{\mathfrak{E}}_{t+\Delta t}}, \tag{5.10}$$

where $w_{\mathfrak{E}} > 0$ is a weighting constant. To be able to operate on a real system, the most important part is to adhere to safety constraints. As formulated in eq. (5.7), the predicted

safety violations need to be uncertainty aware, independent of the source of uncertainty. We integrate this into the planning method by adding:

$$c_{\mathfrak{S}}(s_t, \boldsymbol{a}) = w_{\mathfrak{S}} \cdot \sum_{\Delta t=1}^{H} [\![p(\hat{s}_{t+\Delta t} \in \mathbb{C}) > \delta]\!] \tag{5.11}$$

where $[\![\cdot]\!]$ is the Iverson bracket. and $w_{\mathfrak{S}}$ is either a large penalty c_{max} or 0 to disable safety. An alternative for implementing safety constraints into CEM is by changing the ranking function [343]. The overall algorithm used in a model-predictive control fashion is outlined in Supplementary D.2.

5.4 Experiments

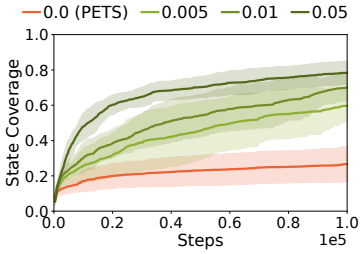
We study our uncertainty-aware planner in 4 continuous state and action space environments and compare to naively optimizing the particle-based estimate of the expected cost similarly to Chua et al. [65]. We start by giving a description of the environments.

BridgeMaze This toy environment (see Figure 5.1c) was specifically designed to study the different aspects of uncertainty independently. The agent (blue cube) starts on the left platform and has to reach the goal platform on the right. To reach the goal platform, the agent has to move over one of three bridges without falling into the lava. The upper bridge is safeguarded by walls; hence, it is the safest path to the goal but also the longest. The lower bridge has no walls and therefore is more dangerous for an unskilled agent to cross but the path is shorter. The middle bridge is the shortest path to the goal. However, randomly appearing strong winds perpendicular to the bridge might cause the agent to fall off the bridge with some probability, making this bridge dangerous.

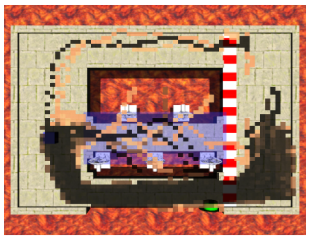
Noisy-HalfCheetah This environment is based on *HalfCheetah-v3* from the OpenAI Gym toolkit. We introduce aleatoric uncertainty to the system by adding Gaussian noise $\xi \sim \mathcal{N}(\mu, \sigma^2)$ to the actions when the forward velocity is above 6. The action noise translates into a non-Gaussian and potentially very complicated state space noise distribution that makes the control problem very challenging.

Noisy-FetchPickAndPlace Based on the *FetchPickAndPlace-v1* gym environment. Additive action noise is applied to the gripper so that its grip on the box might become tighter or looser. The noise is applied for x-positions < 0.8 which is illustrated in Figure 5.1a by a blue line causing the agent to drop the box with high probability if it tries to lift the box too early.

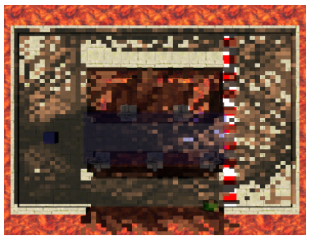
Solo8-LeanOverObject In this robotic environment, the task of a quadrupedal robot [122] is to stand up and lean forward to reach a target position (purple markers need to reach green dots in Figure 5.1b) without hitting an object visualized by the red cube



(a) State space exploration over time depending on epistemic bonus ($w_{\mathfrak{E}}$).



(b) State space coverage with $w_{\mathfrak{E}} = 0$.



(c) State space coverage with $w_{\mathfrak{E}} = 0.05$.

Figure 5.3: Active learning setting: The epistemic bonus allows RAZER to seek states for which no or only little training data exists (a,c). Means and standard deviations for (a) were computed over 5 runs. PETS overfits to a particular solution (b). In (b) and (c), the brightness of the dots is proportional to the time when they were first encountered.

representing the unsafe zone. The robot starts in a laying position as shown in the inset of Figure 5.1b. As in the *Noisy-HalfCheetah* environment, Gaussian action noise is applied to mimic real-world perturbances.

5.4.1 Algorithmic Choices and Training Details

For model-predictive planning we use the CEM implementation from Pinneri et al. [247]. Further details about hyperparameters can be found in Supplementary D.1.2. For planning, we use the same architecture for the ensemble of probabilistic models, both in RAZER and in PETS. The only difference is that in RAZER we also forward propagate the mean state predictions in addition to the sampled state predictions. Further details can be found in Supplementary D.1.1.

5.4.2 Active Learning for Model Improvement

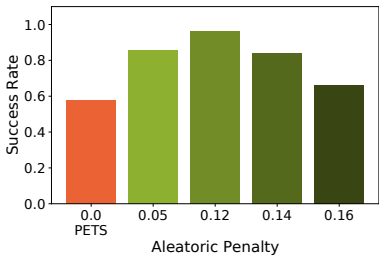
If model uncertainties are used for risk-averse planning, they are only meaningful if the model has the right training data. Only from good data can the parameters of the approximate noise model be learned correctly. In case of too little data, the agent might avoid parts of the state space due to an overestimation of the model uncertainties. On the other hand, the agent might enter unsafe regions for which the uncertainties are underestimated. By adding the epistemic bonus to our domain-specific cost, the planner can actively seek states with high epistemic uncertainty, i.e. for which no or only little training data exists.

Figure 5.4a shows this active data gathering process for the *BridgeMaze* environment. PETS finds one particular solution to the problem of reaching the goal platform. It chooses

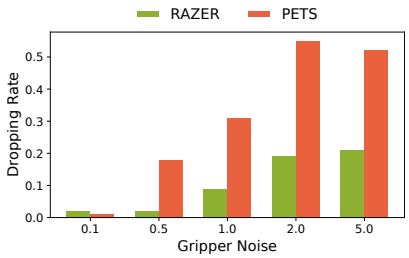
the path over the safer, lower bridge rather than the dangerous middle path and the longer path via the upper bridge (Figure 5.3b). Once, one solution is found, the model overfits to it without exploring any other parts of the state space. This is also reflected in the plateauing of the red curve in Figure 5.3a.

In comparison, RAZER actively explores larger and larger parts of the state space with an increasing weight of the epistemic bonus (Figure 5.3a). RAZER not only finds the easy solution found by PETS but also extensively explores other parts of the state space (Figure 5.3c). To not get stuck at the middle bridge during exploration due to the inherent noise, it is important to separate between epistemic and aleatoric uncertainties. Only the former should be used for exploration. With enough data, our model can correctly capture the uncertainties of these states resulting in the epistemic uncertainty approaching zero.

5.4.3 Risk-Averse Planning



(a) *BridgeMaze* success depending on $w_{\mathfrak{A}}$ for 50 runs.



(b) Dropping rate in *Noisy-FetchPickAndPlace* for 100 runs.

Figure 5.4: Risk-averse planning in the face of aleatoric uncertainty yields higher success rates in noisy environments. For (b) we use ground truth models and a fixed aleatoric penalty weight $w_{\mathfrak{A}}$.

Once a good model is learned, it can be used for safe planning. What differentiates RAZER from PETS is that it makes explicit use of uncertainty estimates while in the latter uncertainties only enter planning by taking the mean over the particle costs and not differentiating between different sources of uncertainty.

BridgeMaze Figure 5.4a shows the success rate of PETS and RAZER in the *BridgeMaze*. In both cases, we use the same model that was trained from data collected during a training run with $w_{\mathfrak{E}} = 0.05$. Hence, the model saw enough training data from all parts of the state space. The noise in the environment is tuned such that there is a chance to cross the bridge without falling. While in Figure 5.3b PETS avoided this path because of an overestimation of the state's value due to a lack of training data and sometimes sees a chance to cross the bridge. However, these attempts are very likely to fail because of stronger winds that occur randomly, resulting in a success rate of only 58%. RAZER does not rely on sampling for the aleatoric

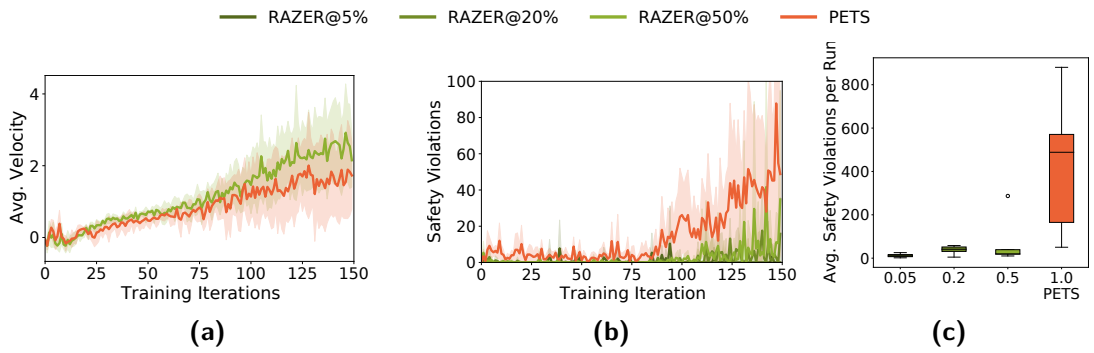


Figure 5.5: *Noisy-HalfCheetah* environment (task lengths 300 steps) with learned models from scratch. At 150 iterations we have seen only 45k points. **(a)** Performance under noisy actions with aleatoric penalty (10 runs). By applying the aleatoric penalty, RAZER can navigate the uncertainties better – leading to higher returns faster. **(b)** Safety violations above a certain body height (simulating a low ceiling) for different values of δ. In **(c)** the number of violations is averaged over the last 50 iterations (summed over 10 rollouts).

part and can thus avoid risk. With a higher penalty constant the success rate increases up to 96% but only as long as the agent is willing to take a risk at all. For large values of $w_{\mathfrak{A}}$ the agent becomes so conservative that it only moves slowly (decreasing reward in Figure 5.4a).

Noisy-HalfCheetah How does RAZER perform on the *Noisy-HalfCheetah* environment when models are learned from scratch? Without aleatoric penalty, the planner is optimistic. Risky situations are only detected if a failing particle is sampled. Thus, the noise is mostly neglected and the robot increases its velocity, gets destabilized, and ends up slower than with the aleatoric penalty (Figure 5.5a).

FetchPickAndPlace In this environment, a 7-DoF robot arm should bring the box to a target position – starting and target positions are at the opposite sides of the table. The shortest path is to lift the box and move in a straight line to the target. However, with noise applied to the gripper action, there is a certain probability to drop the box along the way. When penalizing aleatoric uncertainty, this is avoided and also fewer trajectory samples are "wasted" in high-entropic regions, as presented in Figure 5.1a. Figure 5.4b shows the number of times the box is dropped on the table depending on the aleatoric penalty. RAZER adopts a cautious behavior, preferring to slide the box on the table and lifting it only in the area without action noise, maintaining a dropping rate lower than 20%, even when considerable noise is applied.

5.4.4 Planning with External Safety Constraints

Noisy-HalfCheetah: We consider a safety constraint on the height of the body above ground simulating a narrow passage. Figure 5.5b shows the number of safety violations.

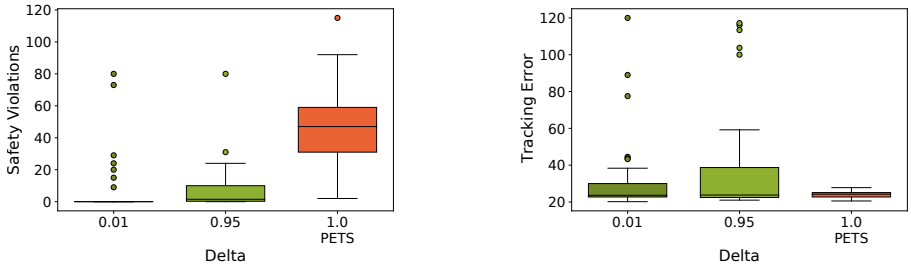


Figure 5.6: Safe planning vs. task-oriented planning in the *Solo8-LeanOverObject* environment with noisy actions. Left: number of safety violations for different values of δ (eq. (5.11)). Right: enforcing safety constraints causes slight reduction in tracking accuracy due to the fixed planning budget and the competing objectives of task and safety costs.

Note that PETS has the same penalty cost for hard violations.

Solo8-LeanOverObject: In this experiment, the robot has to move to two target points with its front and rear of the trunk while avoiding entering a specified rectangular area (fragile object). The front feet are fixed. To track the points, the robot has to lean forward, such that it can lose balance due to noisy actions. In contrast to PETS, RAZER successfully manages to satisfy the safety constraints almost always as shown in Figure 5.6. However, satisfying the safety constraint comes with the cost of reduced tracking accuracy.

5.5 Discussion

In this chapter, we have provided a methodology to separate uncertainties in stochastic ensemble models (PETSUS) which can be used as a tool to build risk-averse model-based planners that are also data-efficient and enforce safety through probabilistic safety constraints (RAZER). This type of risk-averseness can be achieved by a simple modification of the cost function in form of uncertainty penalties in zero-order trajectory optimizers.

Furthermore, the separation of uncertainties allows us to do proper exploration via epistemic bonus which benefits generalization of the model. As future work, it would be of interest to see this approach applied to a proper transfer learning setting from simulations to real systems, where risk-averseness combined with exploratory behavior is crucial for efficient learning and safe operation.

The most important questions of life are indeed, for the most part, really only problems of probability.

Pierre-Simon Laplace

6 Diverse Offline Imitation Learning

This chapter is based on the work "Offline Diversity Maximization under Imitation Constraints" [332]. We introduce an offline algorithm for extracting diverse skills from demonstrations. The core motivation of the algorithm is the Fenchel-dual imitation learning formulation, which connects the dual problem of value learning to the problem of maximizing over occupancy distributions.

6.1 Motivation

In Chapter 5, we had a model-based planning problem and we needed to deal with separation of uncertainties to do efficient data collection, and risk-averse planning. In this chapter, we consider the case of offline learning under constraints, where the policy is not allowed to collect any additional data.

Recent advancements in RL have included substantial progress in unsupervised skill discovery, aiming to empower autonomous agents with the capability to acquire a diverse set of skills directly from their environment, without relying on predefined human-engineered rewards or demonstrations. These methods have the potential to revolutionize the way RL agents learn to solve complex tasks. The growing interest in unsupervised skill discovery has led to various approaches, typically rooted in information-theoretic concepts, including empowerment [160, 214, 94], information bottleneck [318, 115, 155] and information gain [143, 309, 240]. Despite these advancements, there remains a significant challenge. Current methods demand substantial online interaction with the environment, making exploration in high-dimensional state-action spaces inefficient. Although Zahavy et al. [359] introduced constraints to enhance skill performance and narrow the exploration space by incentivize diverse skills to meet a certain utility measure, their approach does not eliminate the need for

considerable online interaction with the environment. Meanwhile, there have been significant recent advances in large-scale data collection [269, 336, 43] and in the development of scalable and sample-efficient offline RL algorithms that leverage diverse behaviors of pre-collected experience. However, these approaches struggle with well-known challenges, including off-policy evaluation and the out-of-distribution problem, which have been studied extensively in previous work [186, 250].

In this work, we address the aforementioned challenges by introducing a novel problem formulation and complementing it with the first principled *offline* RL algorithm for unsupervised skill discovery that, in addition to maximizing diversity, ensures that each learned skill imitates state-only expert demonstrations to a certain degree. More specifically, we consider a problem formulation with two datasets: a large one with diverse state-action demonstrations and another much smaller one with state-only expert demonstrations. This setting is particularly valuable in robotics scenarios where expert demonstrations are limited and the domain of the expert may be different from that of the agent, such as in human demonstrations. Another potential application is to enhance the realism of computer games by creating an immersive experience of interacting with non-player characters, each behaving in a slightly different style, while all partially imitating the behavior of a human expert.

We formulate the problem as a *Constrained Markov Decision Process* (CMDP) [9, 314] that seeks to maximize diversity through a mutual information objective, subject to Kullback-Leibler (KL) divergence state occupancy constraints ensuring that each skill imitates state expert demonstrations to a certain degree. The resulting CMDP has convex objective and constraints, making the optimization problem intractable. We adopt a tractable relaxation approach consisting of an alternating scheme that maximizes a variational lower bound on mutual information, and to handle the constraints it applies Lagrange relaxation. Our method, DOI, overcomes the off-policy evaluation by leveraging the Fenchel-Rockafellar duality in RL [221, 154, 201] to connect a dual optimal value solution (computed using offline samples) with primal optimal state-action occupancy ratios. These ratios serve as importance weights for offline training of a skill-conditioned policy, skill-discriminator, KL-divergence estimators, and Lagrange multipliers. We demonstrate the effectiveness of our method on the standard offline benchmark D4RL [100] and on a custom offline dataset collected from a 12-DoF quadruped robot Solo12 [187]. In addition, we show that DOI trained on simulation data transfers well to a real robot system.

6.2 Related Work

In the context of skill discovery Achiam et al. [2] and Campos et al. [51] showed that methods like DIAYN [94] can struggle to learn large numbers of skills and have a poor coverage of the state space. Strouse et al. [309] observed that when a novel state is visited, the discriminator

lacks sufficient training data to accurately classify skills, which results in a low intrinsic reward for exploration. They address this by introducing an information gain objective (involving an ensemble of discriminators) as a bonus term. Kim, Park, and Kim [155] gave a skill discovery approach based on an information bottleneck that leads to disentangled and interpretable skill representations. Park et al. [238, 239] proposed a Lipschitz-constrained skill discovery method based on a distance-maximizing and controllability-aware distance function to overcome the bias toward static skills and to allow the agent to learn complex and far-reaching behaviors. Sharma et al. [294] developed a method that simultaneously discovers predictable skills and learns their dynamics. In a follow-up work, Park and Levine [240] addresses the problem of errors in predictive models by learning a transformed MDP, whose action space contains only easy to model and predictable actions. These works provide RL algorithms for unsupervised skill discovery that require *online* interaction with the environment and do not impose utility measures on the learned skills. In contrast, DOI gives a principled *offline* algorithm for maximizing diversity under imitation constraints.

A large body of research has focused on successor features [75, 22], a powerful technique in RL for transfer of knowledge across tasks by capturing environmental dynamics, particularly promising for skill discovery when coupled with variational intrinsic motivation [121, 21, 128] to enhance feature controllability, generalization, and task inference. In contrast to our work, these approaches do not impose performance constraints on the learned skills. Zahavy et al. [359] cast the task of learning diverse skills, each achieving a near-optimal performance with respect to a given reward, into a constrained MDP setting with a physics-inspired diversity objective based on a minimum ℓ_2 distance between the successor features of different skills. However, this approach requires significant *online* interaction with the environment to learn the skills.

Numerous practical algorithms for offline RL have been proposed [186, 250], including methods based on advantage-weighted behavioral cloning [224, 340], conservative strategies to stay close to the original data distribution [172, 61] and using only on-data samples [168, 351]. While these methods excel at learning a policy that maximizes a fixed reward, they are not directly applicable in our setting, which has a non-stationary reward that depends on: i) the log-likelihood of a skill discriminator, and ii) Lagrange multipliers. In addition, these techniques cannot be used to i) train a skill discriminator and ii) estimate a KL divergence offline.

Naive importance sampling approaches for off-policy estimation are known to suffer from unbounded variance in the infinite horizon setting, a problem known in the literature as "the curse of horizon". Liu et al. [194] and Mousavi et al. [218] addressed this challenge by providing theoretical foundations and a principled off-policy algorithm, using a backward Bellman operator, that avoids exploding variance by applying importance sampling to state-visitation distributions, and by providing practical solutions in Reproducing Kernel Hilbert Spaces.

An alternative research direction in off-policy estimation, referred to as *Distribution Correction Estimation* (DICE), has introduced innovative techniques, with Nachum et al. [220] mitigating variance with importance sampling, Nachum et al. [222] enabling policy gradient from off-policy data without importance weighting, Kim et al. [154] stabilizing offline imitation learning with imperfect demonstrations, Zhang, Liu, and Whiteson [363] improving density ratio estimation, Dai et al. [71] providing high-confidence off-policy evaluation. Subsequently, Xu et al. [352] applied this approach to offline RL and demonstrated its effectiveness in continuous control tasks. Our work uses a DICE-based off-policy approach similar to OptiDICE [179, 180] for estimating importance ratios, while considering a constrained formulation with a mutual information objective and KL-divergence imitation constraints. Lee et al. [180] studies a cost-conservative constrained setting, with an OptiDICE-based off-policy evaluation.

6.3 Preliminaries

Here we utilize the framework of MDPs (see Section 1.4). In the skill discovery setting, $z \sim p(Z)$ denotes a fixed latent skill on which we condition a policy $\pi_z : S \times Z \mapsto \Delta(\mathcal{A})$. We will treat $p(Z)$ as a categorical distribution over a discrete set Z of $|Z|$ many distinct indicator vectors in $\mathbb{R}^{|Z|}$. The skill-conditioned policy π_z induces a state occupancy denoted by $d_z(s) := d^{\pi_z}(s)$, and when it is clear from the context we will refer to $d_z(s)$ as a "skill".

We consider an offline setting with access to the following datasets: i) $\mathcal{D}_E$ sampled from an expert state occupancy $d_E(S)$; and ii) $\mathcal{D}_O$ sampled from a state-action occupancy $d_O(S, A)$ generated by a mixture of behaviors. Our analysis makes use of the following coverage assumption on state occupancies.

Assumption 6.3.1 (Expert coverage)**.** *We assume that* $d_E(s) > 0$ *implies* $d_O(s) > 0$.

6.4 Method

Given an expert and a coverage dataset as above, we aim to solve *offline* the constrained optimization problem

$$\max_{\{d_z(S)\}_{z \in Z}} \quad \mathcal{I}(S; Z) \tag{6.1}$$

$$\text{subject to} \quad \mathrm{D}_{\mathrm{KL}}\left(d_z(S) || d_E(S)\right) \leq \varepsilon \quad \forall z, \tag{6.2}$$

where $\mathcal{I}(S; Z)$ denotes the mutual information between states and skills. Henceforth, we shall make use of color coding to highlight the diversity signal in blue and the imitation signal

in orange. The preceding problem formulation and our algorithmic framework can be easily extended to capture: i) objectives in (6.1) that combine conditional mutual information (c.f. DADS in [294]) and information gain (c.f. DISDAIN in [309]); and ii) general f-divergence constraints in (6.2), see Nachum and Dai [221] and Ma et al. [201]. We leave the study of these variants for future work.

Since maximizing the mutual information is generally intractable, in line with previous work [94] we assume that the latent skills are sampled uniformly at random, i.e., $p(z) = \frac{1}{|Z|}$, and as a trackable surrogate we consider instead the following variational lower bound

$$\mathcal{I}(S; Z) \geq \mathbb{E}_{p(z), d_z(s)} [\log q(z|s)] + \mathcal{H}(p(z)) = \sum_z \mathbb{E}_{d_z(s)} \left[\frac{\log (|Z| q(z|s))}{|Z|} \right]. \quad (6.3)$$

Here with $q(z|s)$ we denote a skill-discriminator tasked with distinguishing between latent skills.

Ma et al. [201] proposed an offline algorithm (SMODICE) that on input an expert dataset $\mathcal{D}_E \sim d_E(S)$ and a coverage dataset $\mathcal{D}_O \sim d_O(S, A)$ such that $\mathcal{D}_E \subset \text{States}[\mathcal{D}_O]$, trains a policy $\pi_{\widetilde{E}}$ which optimizes the problem

$$\min_\pi \mathrm{D}_{\mathrm{KL}} \left(d^\pi(S) || d_E(S) \right), \quad (6.4)$$

and also outputs ratios $\eta_{\widetilde{E}}(s, a) = d_{\pi_{\widetilde{E}}}(s, a) / d_O(s, a)$ for every state-action pair $(s, a) \in \mathcal{D}_O$.

An important observation is that the state constraints (6.2) can be reduced to state-action constraints, by training an expert policy $\pi_{\widetilde{E}}$, which optimizes eq. (6.4). More specifically, for each latent skill z we replace the state constraint (6.2) with the following state-action constraint

$$\mathrm{D}_{\mathrm{KL}} \left(d_z(S, A) || d_{\widetilde{E}}(S, A) \right) \leq \varepsilon, \quad (6.5)$$

where $d_{\widetilde{E}}(s, a)$ denotes the state-action occupancy $d_{\pi_{\widetilde{E}}}(s, a)$ induced by the expert policy $\pi_{\widetilde{E}}$.

We focus on a reduction of CMDPs to MDPs using gradient-based techniques, known as Lagrangian methods [39, 35, 316]. In contrast to prior work on CMDP, which has focused primarily on linear objectives and constraints, we consider the nonlinear setting with convex objectives and constraints. More specifically, we seek to maximize the right-hand side of eq. (6.3) subject to eq. (6.5). Solving this problem is equivalent to

$$\max_{\substack{d_z(s,a) \\ q(z|s)}} \min_{\lambda \geq 0} \sum_z \mathbb{E}_{d_z(s)} \left[\frac{\log (|Z| q(z|s))}{|Z|} \right] + \sum_z \lambda_z \left[\varepsilon - \mathrm{D}_{\mathrm{KL}} \left(d_z(S, A) || d_{\widetilde{E}}(S, A) \right) \right], \quad (6.6)$$

where with λ_z we denote the Lagrange multiplier corresponding to latent skill z.

6.4.1 Approximation Scheme

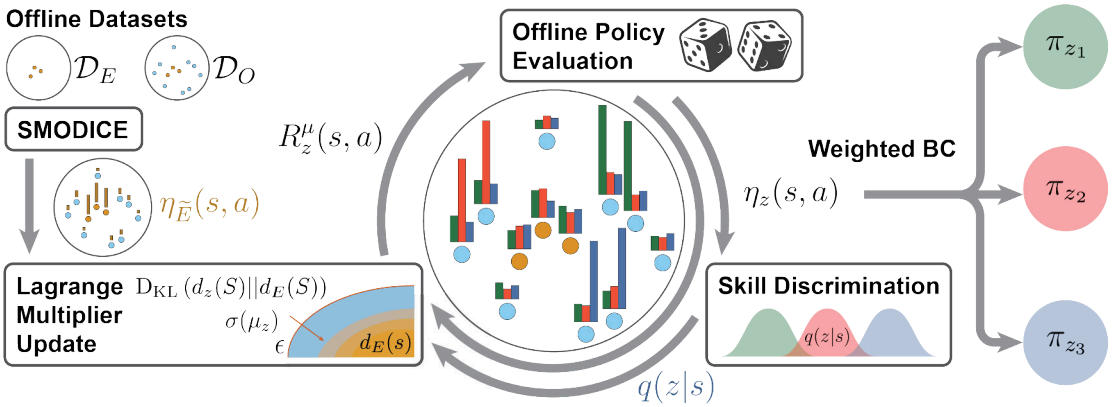


Figure 6.1: Illustration of Algorithm 5. We compute expert importance ratios $\eta_{\widetilde{E}}(s,a)$ by running SMODICE on the offline datasets $\mathcal{D}_E$ and $\mathcal{D}_O$. These expert ratios are then used in the alternating scheme described in Section 6.4.1 to obtain the importance ratios $\eta_z(s,a)$ (with support in $\mathcal{D}_O$) for each skill z. Specifically, the skill-ratios $\eta_z(s,a)$ are computed by a DICE-like offline policy evaluation algorithm on input a reward $R_z^\mu(s,a)$ that balances skill diversity (skill-discriminator $q(z|s)$) and expert imitation (importance ratios $\eta_{\widetilde{E}}(s,a)$).

We use a popular heuristic, known in the literature as *alternating optimization*, to approximately compute a local optimum of Problem (6.6). More precisely, the method alternates between optimizing each model while holding all others fixed, and iteratively refines the solution until convergence is reached or a stopping criterion is met. Furthermore, as we can guarantee in practice that the Lagrange multipliers λ are always positive, we consider Problem (6.6) with $\lambda > 0$, that is

$$\max_{\substack{d_z(s,a)\\ q(z|s)}} \min_{\lambda>0} \sum_z \lambda_z \Big\{ \varepsilon + \mathbb{E}_{d_z(s,a)} \left[R_z^\lambda(s,a) \right] - \mathrm{D}_{\mathrm{KL}} \left(d_z(S,A) || d_O(S,A) \right) \Big\}, \tag{6.7}$$

where

$$R_z^\lambda(s,a) := \underbrace{\frac{1}{\lambda_z}}_{\text{Constraint Violation}} \underbrace{\frac{\log\left(q(z|s)|Z|\right)}{|Z|}}_{\text{Skill Diversity}} + \underbrace{\log \eta_{\widetilde{E}}(s,a)}_{\text{Expert Imitation}}. \tag{6.8}$$

The reward in (6.8) is derived in Supplementary E.3 and relies on the equality (see Supplementary E.4.3)

$$\mathrm{D}_{\mathrm{KL}}\big(d_z(S,A)||d_{\widetilde{E}}(S,A)\big) = \mathrm{D}_{\mathrm{KL}}\big(d_z(S,A)||d_O(S,A)\big) - \mathbb{E}_{d_z(s,a)}\big[\log \frac{d_{\widetilde{E}}(s,a)}{d_O(s,a)}\big], \tag{6.9}$$

and the definition of $\eta_{\widetilde{E}}(s,a) = d_{\widetilde{E}}(s,a)/d_O(s,a)$.

Intuitively, the reward $R_z^\lambda(s,a)$ balances between diversity and KL-closeness to the expert state-action occupancy. The Lagrange multiplier λ_z scales down the log-likelihood of the skill-discriminator $q(z|s)$, effectively reducing the diversity signal, when the state-action occupancy $d_z(S,A)$ violates the KL-divergence constraint (6.5), and vice versa. Each term in the reward (6.8) involves a separate optimization procedure, which will be described in the next section.

6.4.2 Approximation Phases

Using the alternating optimization scheme, Algorithm 5 decomposes into the following three optimization phases. In PHASE 1, we train a value function $V_z^\star$, ratios $\eta_z(s,a)$ and a skill-conditioned policy π_z. In PHASE 2, we train a skill-discriminator $q(z|s)$. Then in PHASE 3, we compute a KL constraint estimator ϕ_z and update accordingly the Lagrange multipliers λ_z. In addition, we perform a preprocessing phase to compute the expert ratios $\eta_{\widetilde{E}}(s,a)$ by invoking the SMODICE algorithm.

Phase 1

With fixed skill-discriminator $q(z|s)$ and Lagrange multipliers $\lambda > 0$, Problem (6.7) becomes

$$\max_{\{d_z(s,a)\}_{z\in Z}} \sum_z \lambda_z \Big\{ \mathbb{E}_{d_z(s,a)} \left[R_z^\lambda(s,a) \right] - \mathrm{D}_{\mathrm{KL}} \left(d_z(S,A) || d_O(S,A) \right) \Big\}, \tag{6.10}$$

or equivalently for every skill z:

$$\begin{aligned} \max_{d_z(s,a)\geq 0} \quad & \mathbb{E}_{d_z(s,a)} \left[R_z^\lambda(s,a) \right] - \mathrm{D}_{\mathrm{KL}} \left(d_z(S,A) || d_O(S,A) \right) \\ \text{subject to} \quad & \textstyle\sum_a d_z(s,a) = (1-\gamma)\rho_0(s) + \gamma \mathcal{T} d(s) \qquad \forall s, \end{aligned} \tag{6.11}$$

where we denote with $\mathcal{T}$ the transition operator: $\mathcal{T}d(s') = \sum_{s,a} \mathcal{P}(s'|s,a)d(s,a)$.

Assumption 6.4.1 (Strict Feasibility)**.** *We assume there exists a solution such that the constraints Section 6.4.2 are satisfied and $d(s,a) > 0$ for all states-action pairs $(s,a) \in \mathcal{S} \times \mathcal{A}$.*

Using Lagrange duality, Theorem 6.4.1 (which implies strong duality) and the Fenchel conjugate (see Supplementary E.2), [221] and [201] showed that Problem 6.11 shares the same optimal value as the following optimization problem

$$V^\star = \arg\min_{V(s)} (1-\gamma)\mathbb{E}_{s\sim\rho_0} \left[V(s) \right] + \log \mathbb{E}_{d_O(s,a)} \exp \left\{ R_z^\lambda(s,a) + \gamma \mathcal{T} V(s,a) - V(s) \right\}, \tag{6.12}$$

where $\mathcal{T}V(s,a) := \mathbb{E}_{\mathcal{P}(s'|s,a)}V(s')$. Moreover, the primal optimal solution is given by

$$\eta_z(s,a) := \frac{d_z^\star(s,a)}{d_O(s,a)} = \text{softmax}\left(R_z^\lambda(s,a) + \gamma\mathcal{T}V_z^\star(s,a) - V_z^\star(s)\right). \tag{6.13}$$

These ratios $\eta_z(s,a)$ are then used to design an offline importance-weighted sampling procedure that, for an arbitrary function f, satisfies

$$\mathbb{E}_{p(z)}\mathbb{E}_{d_z^\star(s,a)}\left[f(s,a,z)\right] = \mathbb{E}_{p(z)}\mathbb{E}_{d_O(s,a)}\left[\eta_z(s,a)f(s,a,z)\right]. \tag{6.14}$$

Afterwards, the optimal skill-conditioned policy $\pi_z^\star$ is trained offline using a weighted behavioral cloning, which is obtained by setting $f(s,a,z) = \log(\pi_z(a|s))$ and maximizing the RHS of eq. (6.14) over all skill-conditioned policies π_z. In practice, gradient descent is used for optimization.

Phase 2

We now give an offline procedure for training a skill-discriminator $q(z|s)$, which takes as input ratios $\eta_z(s,a)$ of a skill-conditioned policy $\pi_z^\star$. The proof is presented in Supplementary E.4.2.

Lemma 6.4.2. *Given ratios $\eta_z(s,a)$, using eq. (6.14) applied with $f(s,a,z) = \log(q(z|s))$, we can compute offline an optimal skill-discriminator $q^\star(z|s)$. In particular, we optimize by gradient descent the following optimization problem*

$$\max_{q(z|s)} \mathbb{E}_{p(z)}\mathbb{E}_{d_O(s,a)}\left[\eta_z(s,a)\log\left(q(z|s)\right)\right].$$

The skill-conditioned policy $\pi_z^\star$ (Phase 1) and the skill-discriminator $q^\star$ (Phase 2), allow us to maximize *offline* the variational lower bound in eq. (6.3) and thus skill diversity. It remains to estimate possible constraint violations in eq. (6.5) and to update the Lagrange multipliers accordingly.

Phase 3

With fixed skill-discriminator $q^\star(z|s)$ and skill-conditioned policy $\pi_z^\star(s)$, Problem (6.7) reduces to $\min_{\lambda>0}\sum_z \lambda_z\left[\varepsilon - \mathrm{D}_{\mathrm{KL}}\left(d_z^\star(S,A)||d_{\widetilde{E}}(S,A)\right)\right]$. We will optimize the Lagrange multipliers by gradient descent. To this end, we now give an offline estimator of the KL-divergence term. The proof is presented in Supplementary E.4.3.

Lemma 6.4.3. *Given skill-conditioned policy ratios $\eta_z(s,a)$ and expert ratios $\eta_{\widetilde{E}}(s,a)$, using eq. (6.14) applied with $f(s,a,z) = \log(\eta_z(s,a)/\eta_{\widetilde{E}}(s,a))$, we can compute offline an estimator of $\mathrm{D}_{\mathrm{KL}}\left(d_z^\star(S,A)||d_{\widetilde{E}}(S,A)\right)$ which is given by*

$$\phi_z := \mathbb{E}_{d_O(s,a)}\left[\eta_z(s,a)\log(\eta_z(s,a)/\eta_{\widetilde{E}}(s,a))\right].$$

We note that the ratios $\eta_z(s,a)$ and $\eta_{\widetilde{E}}(s,a)$ are computed only on state-action pairs within the offline dataset $\mathcal{D}_O$. Furthermore, in practice, we ensure that these ratios are strictly positive, so that the KL estimator ϕ_z is well defined and bounded.

6.5 Algorithm

Our optimization method consists of three phases, each of which optimizes a specific model and fixes the remaining ones. It is important to emphasize that in contrast to prior work, our problem formulation considers an optimization problem with constraints. Furthermore, the reward function in eq. (6.8) is non-stationary, since it depends on the bounded Lagrange multipliers that balance diversity ($\log q(z|s)$) and expert imitation ($\log \eta_{\widetilde{E}}(s,a)$). This has significant algorithmic implications, as it requires solving a sequence of standard RL problems, each of which admits offline policy evaluation.

To smooth the transition of the reward signal between successive iterations, we enforce a slow change of the Lagrange multipliers. More specifically, we use the technique of bounded Lagrange multipliers [308, 359], which applies a Sigmoid transformation $\lambda = \sigma(\mu)$ component-wise to unbounded variables $\mu \in \mathbb{R}^{|Z|}$, so that the effective reward is a convex combination of a diversity term and an expert imitation term. In practice, this transformation ensures that $\lambda > 0$. Hence, the reward for each latent skill z becomes

$$R_z^{\mu}(s,a) := (1-\sigma(\mu_z))\,\frac{\log\left(q^{\star}(z|s)|Z|\right)}{|Z|} + \sigma(\mu_z)\log\eta_{\widetilde{E}}(s,a). \tag{6.15}$$

We now present the resulting multi-phase optimization procedure in Algorithm 5. For a practical implementation, we leverage the power of neural networks and deep learning techniques for accurate function approximation. More specifically, we train an expert policy $\pi_{\widetilde{E}}$, a skill-conditioned policy $\{\pi_z\}_{z\in Z}$ and a value function $\{V_z\}_{z\in Z}$. While practically convenient, this means that each phase of Algorithm 5 is only approximately solved. In practice, we do not solve the optimization problem to optimality in each phase, but rather perform a few gradient descent steps.

We have found that fitting the skill-discriminator $q(z|s)$ is prone to collapse to the uniform distribution. To alleviate this issue, in addition to the variational lower bound objective (6.3), we add the DISDAIN information gain term, proposed in [309]. This bonus term is an entropy-based disagreement penalty that estimates the epistemic uncertainty of the skill-discriminator, and is implemented in practice by an ensemble of randomly initialized skill-discriminators. Due to the high initial disagreement on unvisited states, this intrinsic reward provides a strong exploration signal and leads to the discovery of more diverse behaviors. Intuitively, for states with small epistemic uncertainty, the skill-discriminator (averaged over

Algorithm 5 Diverse Offline Imitation (DOI)

Input: a state-only expert dataset $\mathcal{D}_E \sim d_E(S)$ and a state-action offline dataset $\mathcal{D}_O \sim d_O(S, A)$
Pre-compute a state-discriminator $c^\star : \mathcal{S} \to (0, 1)$ via optimizing the following objective with the gradient penalty in [123] $\min_c \mathbb{E}_{d_E(s)}[\log c(s)] + \mathbb{E}_{d_O(s)}[\log(1 - c(s))]$
Apply **Phase 1** with reward $R(s, a) = \log \frac{c^\star(s)}{1-c^\star(s)}$ to compute ratios $\eta_{\widetilde{E}}(s, a) := \frac{d_{\widetilde{E}}(s,a)}{d_O(s,a)}$ for all $s, a \in \mathcal{D}_O$

Repeat until convergence:
Phase 1. (Fixed Lagrange multipliers $\sigma(\mu)$ and skill-discriminator values $q^\star(z|s)$)
For each latent skill z:
compute a value function $V_z^\star$ optimizing eq. (6.12) with reward $R_z^\mu(s, a)$ in eq. (6.15)
compute ratios $\eta_z(s, a) := \frac{d_z^\star(s,a)}{d_O(s,a)} = \text{softmax}\left(R_z^\mu(s, a) + \gamma \mathcal{T} V_z^\star(s, a) - V_z^\star(s)\right)$ for all $s, a \in \mathcal{D}$
train a skill-conditioned policy $\pi_z^\star = \arg\max_{\pi_z} \mathbb{E}_{d_O(s,a)}\left[\eta_z(s, a) \log \pi_z(a|s)\right]$

Phase 2. (Fixed ratios $\eta_z(s, a)$ and bounded Lagrange multipliers $\sigma(\mu)$)
Train a skill-discriminator $q^\star = \arg\max_{q(\cdot|s)} \mathbb{E}_{p(z)}\mathbb{E}_{d_O(s,a)}\left[\eta_z(s, a) \log q(z|s)\right]$

Phase 3. (Fixed ratios $\eta_{\widetilde{E}}(s, a)$ and $\eta_z(s, a)$)
Compute for each latent skill z an estimator $\phi_z := \mathbb{E}_{d_O(s,a)}\left[\eta_z(s, a) \log(\eta_z(s, a)/\eta_{\widetilde{E}}(s, a))\right]$
Optimize the loss $\min_\mu \sum_z \sigma(\mu_z)(\varepsilon - \phi_z)$

the ensemble members) should reliably discriminate between latent skills, thus making the intrinsic reward of the skill-discriminator's log-likelihood more accurate.

6.6 Experiments

For evaluation of our method we consider 12 degree-of-freedom quadruped robot Solo12 [122], on a simple locomotion task in both the *simulation* and the *real* system. We complement this with an obstacle navigation task, in simulation, and demonstrate that some of the learned diverse skills robustly reach a target position while the expert fails. Furthermore, we provide evaluation on the Ant, Walker2D, HalfCheetah and Hopper environments from the standard D4RL benchmark [100].

6.6.1 Locomotion

Data collection. For the Solo12 evaluation, we collected domain-randomized offline and expert data from simulation in the Isaac Gym [203], using pretrained policy checkpoints obtained by training the robot to track a certain speed of the base with the on-policy diversity

maximization algorithm DOMiNiC [62]. We defer the data collection procedure to the Supplementary E.7. The *expert dataset* was collected by using the best deterministic policy from the last checkpoint of the training procedure, which was trained to track forward velocity only without diversity objective. In contrast, the *offline dataset* was acquired by employing stochastic policies gathered from various checkpoints throughout the training of the expert, featuring multiple latent skills. More than half of the *offline dataset* was collected by a random Gaussian policy. In line with previous approaches by Kim et al. [154] and Ma et al. [201], our practical implementation aims to fulfill the expert coverage Theorem 6.3.1. To achieve this, we create the coverage dataset $\mathcal{D}_O$ by adding a small number of expert trajectories to the offline dataset, resulting in an *unlabeled* expert fraction of 1/160 in $\mathcal{D}_O$. We discard expert actions from the expert dataset to ensure that our algorithm does not have labeled access to them. The resulting *expert dataset* $\mathcal{D}_E$ is used to learn a state classifier $c(s)$. Then the SMODICE is executed to compute the importance ratios $\eta_{\widetilde{E}}(s,a)$, see Section 6.4. We trained the policy for 350 steps, where each step involves the stages described in Section 6.5. In each stage, we execute 200 epochs of batched training over the data.

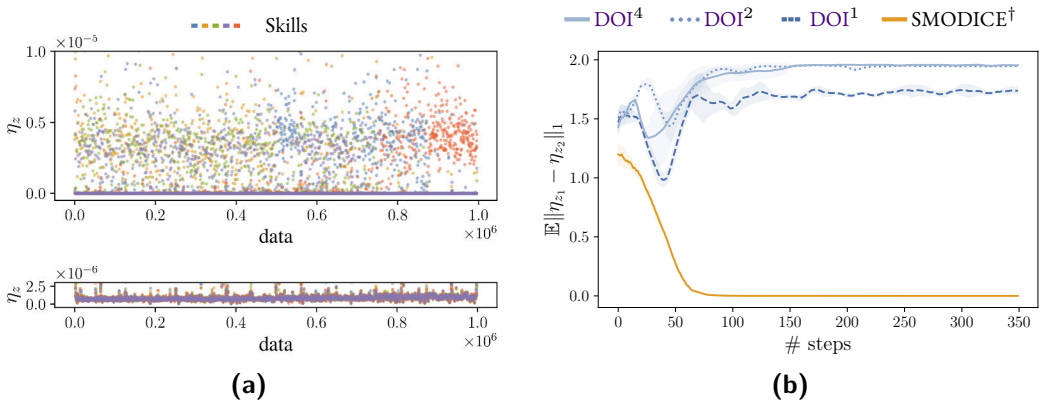


Figure 6.2: Data points separation by importance ratios $\eta_z(s,a)$, given different levels of ε in Solo12. (a) Distribution of importance ratios $\eta_z(s,a)$ over the offline dataset $\mathcal{D}_O$ for distinct skills with DOI^4 ($\varepsilon = 4$) (upper) and a skill-conditioned variant of SMODICE (lower). (b) Average ℓ_1 distance of ratios η_z belonging to distinct skills, depending on ε. The higher the value of ε, the greater the ℓ_1 distance.

Here with DOI$^\varepsilon$ we denote an execution of Algorithm 5 with constraint threshold set to ε. We proceed by analyzing the learned DOI skills in three evaluation settings: i) over the fixed offline datasets; ii) a Monte Carlo on-policy evaluation in the simulator; and iii) the resulting clustering structure involving the offline and expert datasets, as well as the DOI skills and the SMODICE expert evaluated in the simulation.

Importance ratios distance. In Figure 6.2, we measure the state-action occupancy $d_z(s, a)$ for each latent skill z through the proxy of importance ratios $\eta_z(s, a)$,[i] for different values of ε. As expected, a higher value of ε increases diversity, resulting in different importance ratios per skill for individual data points. This difference is then aggregated by computing an expected ℓ_1 distance between importance ratios of distinct skills, i.e., $\mathbb{E} \|\eta_{z_i} - \eta_{z_j}\|_1$, and is reported in Figure 6.2. We note that the looser the constraint (lighter color), the easier it is to diversify in the sense of η_z. Figure 6.2b shows the average ℓ_1 distance between skill importance vectors η_z over the dataset for $\varepsilon \in \{0.0, 1.0, 2.0, 4.0\}$ (lighter color indicates higher ε). Moreover, the tighter the constraint (smaller ε), the smaller the difference between the distinct skill importance ratios.

To analyze the influence of the diversity objective on the learned skills, we consider as a baseline a skill-conditioned variant of [201], denoted SMODICE†, which does not have access to the skill discriminator $q(z|s)$. This is equivalent to DOI with fixed $\sigma(\mu_z) = 1$ in the reward eq. (6.15). We defer further experiments with fixed Lagrange multipliers to Supplementary E.14. In Figure 6.2a, we observe diversification across the dataset assignment to skills when using DOI, whereas training an ensemble of skills with only expert imitation reward (i.e., $\sigma(\mu_z) = 1$) collapses to nearly the same importance ratios per skill per data point.

Successor features distance. We have further evaluated diversity on the Monte Carlo estimates of the expected successor features over the initial state, based on 30 policy rollouts per skill. The γ-discounted successor features (SFs) for state s are defined as $\psi_z(s) = \mathbb{E}_{d_z(s)}[\phi(s)]$, where $d_z(s)$ is the γ-discounted state occupancy for a skill policy π_z. With slight abuse of notation, we define $\psi_z = \mathbb{E}_{\rho_0(s)}[\psi_z(s)]$, the expected SFs over the initial state distribution. As a diversity metric, we take the expected ℓ_2 distance between the SFs of distinct skills, i.e., $\mathbb{E} \|\psi_{z_1} - \psi_{z_2}\|_2$. The results are presented in Figure 6.3 and are consistent with the proxy diversity metric. In particular, there is a correspondence between the offline data separation induced by the importance ratios η_z (see Figure 6.2a), and a higher distance between the expected SFs ψ_z (see Figure 6.3a). In terms of performance, DOI achieves a forward velocity comparable to the expert (see Figure 6.3a) while learning diverse skills with respect to base height h (see Figure 6.3b). We also observed that the multipliers $\sigma(\mu_z)$ are non-zero for all skills, indicating that the constraint is active. In addition, they stabilize at reasonable levels as training progresses, which we show in Supplementary E.9 for both the Solo12 and Ant.

DOI skills form well-separated clusters. Here we conduct a controlled experiment with full trajectory information, which remains hidden to the DOI algorithm. In Figure 6.4, the Successor Features of each trajectory in the expert dataset are transformed by UMAP [209] algorithm into 2D space. This transformation is then used to map the SFs of each trajectory

[i] For the computation of the skill-ratios $\eta_z(s, a)$, we choose a projection Π of the expert state (see Supplementary E.11) that yields 3-dimensional planar and angular velocities of the robot's base in the base frame.

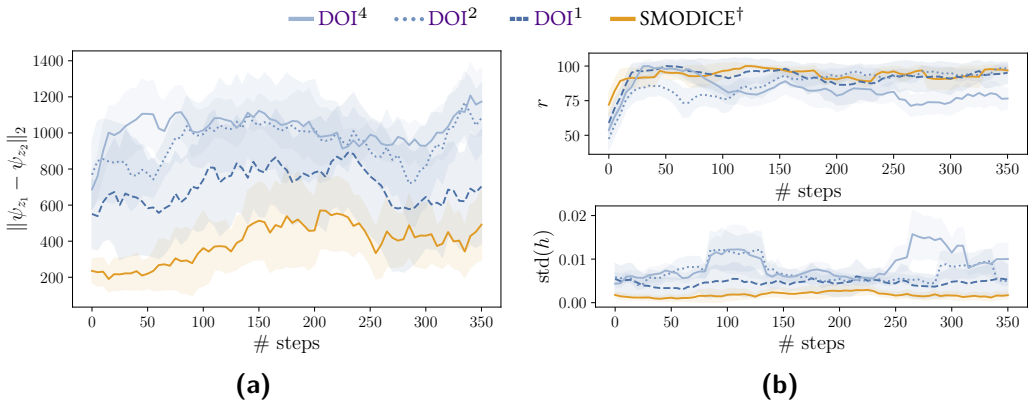


Figure 6.3: (a) Average ℓ_2 distance between Monte Carlo estimates of successor features ψ_z of distinct skills; (b) return r as % of expert return and standard deviation of base height $\text{std}_z(h)$. Both depend on ε for the SOLO12.

into 2D space for: i) the offline dataset, ii) the SMODICE expert evaluated in simulation, and iii) the learned DOI skills (red, green, blue, purple, cyan) also evaluated in simulation. The diversity of learned DOI skills is reflected in a well-separated cluster structure.

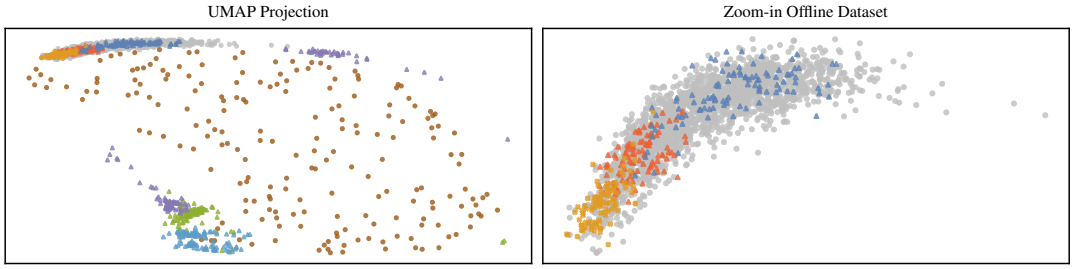


Figure 6.4: Successor Features projection onto 2D space using the UMAP algorithm.[ii]

6.6.2 Robust Obstacle Navigation

Data collection. Similarly to the locomotion task in Section 6.6.1, both *expert dataset* and *offline dataset* were generated from pretrained policy checkpoints from training a robot to navigate in the terrain of obstacles with fixed time limit using the DOMiNiC [62] algorithm. Unlike the previous task, the *expert dataset* was collected using the best deterministic skill-conditioned policy from the last checkpoint of the training procedure, which exhibits diverse strategies to navigate the obstacle terrain, including bypassing it from both sides or climbing over it. The *offline dataset* was acquired through rolling out stochastic policies gathered from

[ii] Figure attributed to Pavel Kolev

multiple checkpoints with multiple skills. Both *expert dataset* and *offline dataset* were collected in a terrain of a single obstacle of a fixed height of 0.2 meters. For details on collecting the dataset for the obstacle navigation task, we refer interested readers to the Supplementary E.7.

Multi-modal expert limitations. Deriving a single policy by SMODICE from expert demonstrations, even in the setting when the dataset was collected from diverse expert strategies, may lack robustness to distribution shifts. This observation emphasizes the need for diverse policy extraction. To illustrate this with a concrete example, consider a scenario where a Solo12 robot navigates around a single box obstacle to reach a target position behind it. The target position can be reached either from the sides (left or right) of the box or by climbing over it (the less robust path). In our experiments, the expert dataset $\mathcal{D}_E$ contains all of the above strategies. As shown in Figure 6.5, for boxes with a height of at least 0.3 meters, the SMODICE expert consistently positions itself in front of the box and thus fails to robustly reach the target position.

Extracting robust policies. In Figure 6.5, we analyze return distributions and sampled trajectories for box heights of $\{0.3, 0.6\}$ meters. The SMODICE expert predominantly fails to reach the target position, due to a bias towards climbing over the box. In contrast, a DOI skill consistently chooses the left side, which leads robustly to the target position and achieves a superior return distribution. However, it is important to note that not all learned DOI skills are robust. Hence, a subsequent selection process is required. Further details about all learned DOI skills, their return distributions and sampled trajectories, different box heights, and additional experimental information are presented in Supplementary E.13.

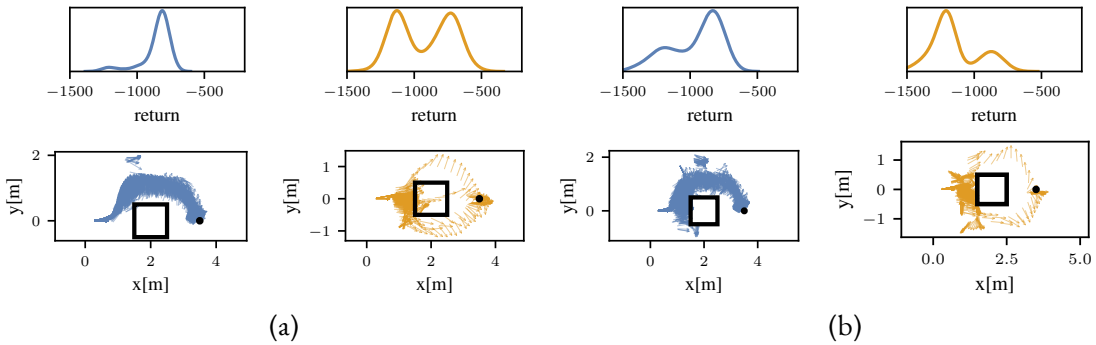


Figure 6.5: Return distributions and sampled trajectories of SMODICE and a DOI skill for terrains with box height (a) 0.3 and (b) 0.6. The heights of the boxes are out-of-distribution for the SMODICE, which tends to get stuck in front of the box due to a bias towards climbing over it. In contrast, the robust DOI skill takes a detour to the left side of the box.

6.6.3 Standard D4RL Environments

We consider the case where we have offline data generated from a random policy mixed with a small amount of expert trajectories.[iii] Figure 6.6 shows the results for both the expected ℓ_2 distance between the SFs or the importance ratios η_z of distinct skills. We normalize the state feature $\phi(s)$ when comparing SFs ψ_z across environments in Figure 6.6a. In most cases, we report a trade-off between the average skill return and the imitation level ε. The larger the imitation slack ε, the more diverse the skills become, but at the cost of lowering the average return, and vice versa. Nevertheless, in Figure 6.6a we show that ε retains some controllability over diversity. The WALKER2D is particularly sensitive to relaxation of the occupancy constraint with respect to performance. We hypothesize that this is due to the fact that the space of policies that achieve a stable gait is very restrictive, resulting in a significant loss of task return for even a small amount of skill diversity. In contrast, the ANT exhibits high stability, with several skills achieving close to expert performance in terms of r. These results are also consistent with SMODICE expert policies used for computing $\eta_{\tilde{E}}(s, a)$ (see Supplementary E.8).

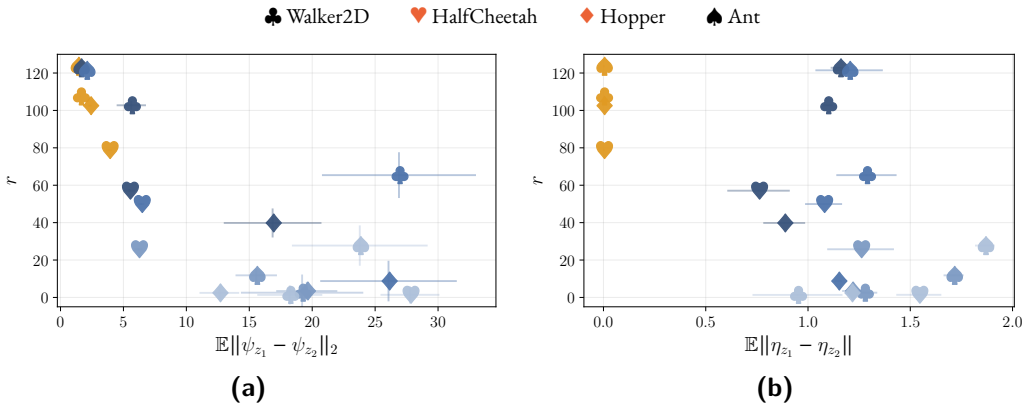


Figure 6.6: Results on D4RL environments with offline data collected from a random policy for $\varepsilon = 0.0$, 0.5, 1.0, 2.0, 4.0. In figure (a) we observe the tradeoff between average skill return and average successor features distance over skills. In figure (b), we report the tradeoff w.r.t. average ℓ_1 distance of importance ratios η_z.

6.7 Removing the Discriminator

The inconvenience of Section 6.4 is that we need to learn the discriminator $q(z \mid s)$ to estimate the lower bound on the mutual information. Indeed, we may as well consider an alternative

[iii] The same setting was considered by Ma et al. [201].

formulation that circumvents the need of a discriminator. Consider the general problem

$$\max_{d_1,\dots,d_n} \text{Diversity}(d_1,\dots,d_n) \quad \text{s.t.} \quad \mathrm{D}_{\mathrm{KL}}\left(d_i(S) \,\|\, d_E(S)\right) \leq \varepsilon \quad \forall i \in \{1,\dots,n\}, \tag{6.16}$$

where we want to optimize $d_1,\dots,d_n$ occupancy measures subject to n KL divergence constraints. From the perspeective of eq. (6.16), we know that from Section 1.4.2 that we can transform KL-regularized expected return objectives into the simpler problem of value function learning. The requirement to do so is to be able to formulate an adequate reward function for the problem. For the case of the Diversity$(d_1,\dots,d_n)$, one can model it as the maximization of the minimum squared ℓ_2 distance between feature expectations of different skills, namely

$$\max_{d_1^\pi,\dots,d_n^\pi} \frac{1}{2}\sum_{z=1}^{n} \min_{k\neq z} \left\|\psi^z - \psi^k\right\|_2^2. \tag{6.17}$$

Here, the index z indexes a particular policy with the expected feature vector $\psi^z = \langle d_z, \phi\rangle$. We take as an example of two occupancy distributions $d_1(s)$ and $d_2(s)$ for which

$$\text{Diversity}(d_1, d_2) = \left\|\psi^1 - \psi^2\right\|_2^2.$$

If we fix $d_2(s)$ and take the gradient $w.r.t.$ d_1 of the objective in eq. (6.17), we arrive to

$$\nabla \frac{1}{2}\left\|\psi^1 - \psi^2\right\|_2^2 = \langle \phi, \psi^1 - \psi^2\rangle. \tag{6.18}$$

Since the objective is convex, maximizing eq. (6.18) maximizes it. Generalizing to an arbitrary number n of occupancy distributions we have for the occupancy distribution d_z, the closest occupancy distribution in terms of the expected feature vector, which we index with j^*. Now we have the same expression, however, with the closest reference occupancy chosen from a pool of occupancy distributions

$$\nabla \frac{1}{2}\sum_{z=1}^{n} \min_{k\neq z} \left\|\psi^z - \psi^k\right\|_2^2 = \phi(\psi^1 - \psi^{j^*}), \tag{6.19}$$

which yields the state reward

$$\beta_z(s) = \langle \phi, \psi^1 - \psi^{j^*}\rangle. \tag{6.20}$$

Now, we may phrase our divergence constrained optimization problem as

$$\max_{d_z(S)} \mathop{\mathbb{E}}_{d_z(S)}[\beta_z(S)], \quad \text{s.t.} \quad \mathrm{D}_{\mathrm{KL}}(d_z(s) \,\|\, d_E(S)) \leq \varepsilon. \tag{6.21}$$

The expected feature components of the reward in eq. (6.20) are readily obtainable from the offline data with the importance ratios $\eta_z(s,a)$ via an importance-weighted MC estimator.

6.8 Discussion

In this chapter, we proposed DOI, a principled offline RL algorithm for unsupervised skill discovery that, in addition to maximizing diversity, ensures that each learned skill imitates state-only expert demonstrations to a certain degree. Our main analytical contribution is to connect Fenchel duality, reinforcement learning, and unsupervised skill discovery to maximize a mutual information objective subject to KL-divergence state occupancy constraints. We have shown that DOI can diversify offline policies for a 12-DoF quadruped robot (in simulation and in reality) and for several environments from the standard D4RL benchmark in terms of both ℓ_2 distance of expected successor features and ℓ_1 distance of importance ratios, which is visible from the data separation induced by $\eta_z(s, a)$ among skills. The importance ratio distance, computed offline, is a robust indicator of diversity, which aligns with the online Monte Carlo diversity metric of expected successor features. The resulting skill diversity naturally entails a trade-off in task performance. We can control the amount of diversity via an imitation level ε, which ensures that distinct skills remain close to the expert in terms of state-action occupancy, which also indirectly controls task performance loss. A promising direction for future research is to impose constraints on the value function of each skill to ensure near-optimal task performance, similarly as is done in the online setting in Chapter 8.

III

Further Works

7

Wasserstein Adversarial Behavior Imitation

This chapter is based on the work "Learning Agile Skills via Adversarial Imitation of Rough Partial Demonstrations" [188]. We introduce a Wasserstein adversarial apprach to imitate from partial demonstrations, resulting in robust policies that transfer to a real robot. The key to the method is treating the policy as a generator, while the discriminator term enters the reward of the policy.

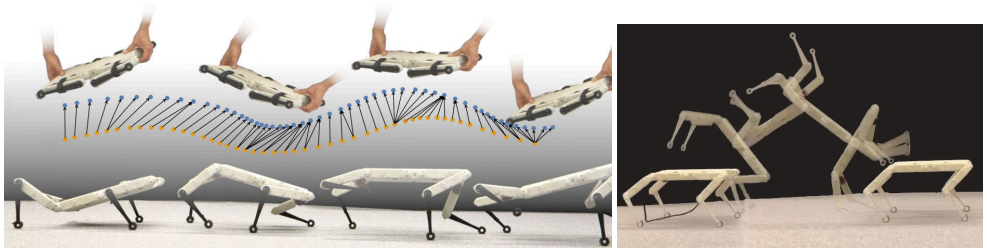


Figure 7.1: Our method (WASABI) achieves agile physical behaviors from rough (hand-held) and partial (robot base) motions. The illustrated performance measure is the Dynamic Time Warping distance of the base trajectories (left). A learned backflip policy is deployed on Solo8 (right). [i]

[i] Figure attributed to Sebastian Blaes.

7.1 Summary

Obtaining dynamic skills for autonomous machines has been a cardinal challenge in robotics. In the field of legged systems, many attempts have been made to attain diverse skills using conventional inverse kinematics techniques [254, 84]. In recent years, learning-based quadrupedal locomotion has been achieved by reinforcement learning (RL) approaches to address more complex environments and improve performance [146, 181, 171, 212]. However, the demand for acquiring more highly dynamic motions has brought new challenges to robot learning. A primary shortage of motivating desired behaviors by reward engineering is the arduous reward-shaping process involved. It can sometimes become extremely demanding in developing highly dynamic skills such as jumping and backflipping, where various terms of motivation and regularization require elaborated refinement. To the end of imitating rich behaviors, it is useful to take a distribution matching viewpoint. This we have already seen in the more restricted offline setting in Algorithm 5, where we made clever use of Fenchel duality in order to diversely imitate expert demonstrations. An alternative is to make use of adversarial methods, such as *Generative Adversarial Imitation Learning* (GAIL) [137] which involve learning a discriminator which distinguishes between samples from the current policy and a demonstration. In this sense, the policy becomes the "generator" object which the discriminator needs to distinguish. A benefit of adversarial methods is their flexibility in the choice of which distribution we want to match. For example, if there are no actions provided by the expert to imitate, we may match state occupancy distributions, or marginals thereof.

In this work, we propose WASABI, which makes use of Wasserstein adversarial imitation learning of partial demonstrations, circumventing the arduous reward design task, the proposed Wasserstein loss being

$$\arg\min_{D} - \mathop{\mathbb{E}}_{d^{\mathcal{M}}} \left[D(o, o') \right] + \mathop{\mathbb{E}}_{d^{\pi}} \left[D(\Phi(s), \Phi(s')) \right], \tag{7.1}$$

where $\Phi(\cdot)$ is a projection operator of the true state s and next state s', highlighting the fact that the demonstration is available only partially. Under conditions of Lipschitz continuity, the Wasserstein loss is an efficient approximation to the earth mover's distance which effectively measures the distance between two probability distributions [275]. Equation (7.1) indicates that the discriminator D acts as a "score" of how close the policies state distribution is to the demonstrator $\mathcal{M}$. Now, the discriminator may be utilized in the reward function of the agent. The reward term in WASABI involves various regularizing factors, to enable better sim-to-real transfer.

[ii] Figure attributed to Chenhao Li.

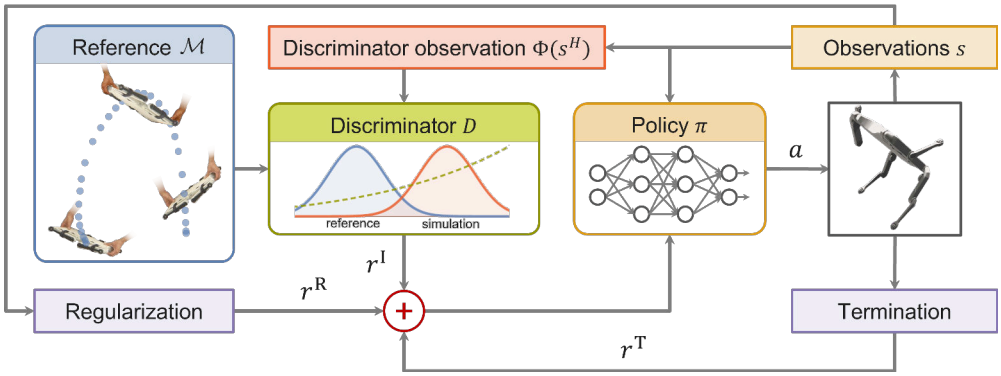


Figure 7.2: System overview. Given a reference dataset defining the desired base motion, the system trains a discriminator that learns an imitation reward for the policy training. This imitation reward is then combined with a regularization reward and termination penalty to train a policy that enables the robot to replicate the demonstrated motion while maintaining feasible and stable joint actuation.[ii]

7.2 Relation to this Thesis

The idea behind WASABI is closely connected to DOI (see Chapter 6), while WASABI attempts to do state-occupancy matching between the policy and demonstration by an adversarial approach from which the reward is obtained from the discriminator, DOI achieves this by the f-divergence constraint. Most importantly, WASABI has an easier task in a sense, since it makes use of an *online* RL algorithm (PPO) in order to extract the policy, meaning that it is allowed to utilize the simulator in order to obtain many environment interaction. In contrast, DOI is completely *offline*. Furthermore, WASABI does not handle the problem of skill extraction, which is the main motivation behind Chapter 6.

8

Learning Diverse Skills for Local Navigation under Multi-constraint Optimality

This chapter is based on the paper "Learning Diverse Skills for Local Navigation under Multi-constraint Optimality" [62]. We introduce a method for extracting diverse skills based on online estimation of successor features under multiple constraints and a physiscs-inspired force for separating the skills. This is framed as a constrained MDP problem.

8.1 Summary

In this work we introduce an *online* method for extracting skills that solve a particular task in diverse ways. However, optimizing diversity is often at odds with task performance, therefore in order to ensure that the task is solved to a satisfying degree, we necessarily need to introduce constraints. For doing so, we make use of the constrained MDP framework, similaraly to *Diversity Optimization Maintaining Near Optimality* (DOMiNO) [359], with one key difference – we account for multiple value constraints at the same time. This is motivated by the fact that, particularly in the robotics settng, we have situations where multiple regularizing factors enter the reward(cost) to ensure smooth natural motions. In this work, we account for these regularizing factors as well as the task reward as value constraints. We call this DoMiNiC, an adaptive extension to the DOMiNO [359].

Zahavy et al. [359] studied the CMDP formulation, which seeks to compute a set of policies $\Pi^n = \{\pi^z\}_{z=1}^n$ that satisfy

$$\max_{\Pi^n} \text{Diversity}(\Pi^n) \text{ s.t. } \langle d_\pi, r_e \rangle \geq \alpha v_e^*, \quad \forall \pi \in \Pi^n, \tag{8.1}$$

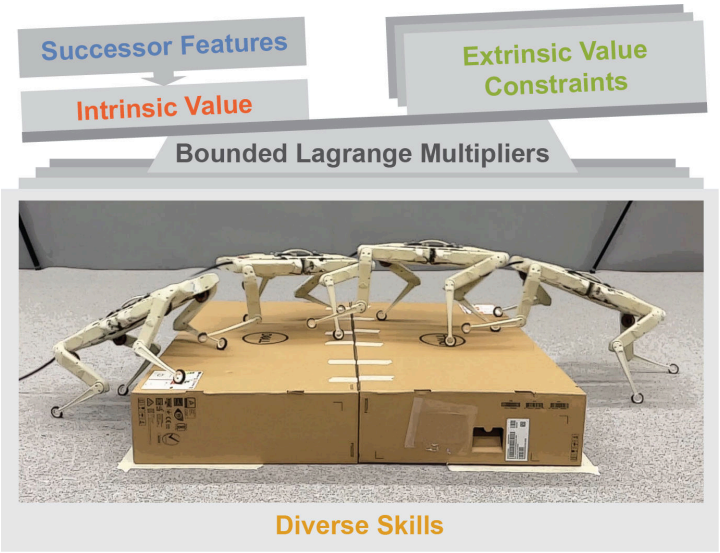


Figure 8.1: Our framework (DoMiNiC) uses a gradient-based Lagrange method to maximize diversity within a specified set of constraints. The learned skills exhibit diverse behaviors in real-world robotic systems.

where r_e and v_e^* correspond to the extrinsic reward and optimal extrinsic value. Intuitively, it computes a set of diverse policies while maintaining a certain level of extrinsic optimality specified by the optimality ratio $\alpha \in [0, 1]$.

As shown in previous work [358], convex diversity objectives can be optimized by solving a sequence of standard RL problems, each with an intrinsic reward equal to the gradient of the objective evaluated at a state-action occupancy d_π of the current iteration:

$$r_i^z = \nabla_{d_\pi^z} \text{Diversity}(d_\pi^1, ..., d_\pi^n), \quad \forall z. \tag{8.2}$$

The CMDP in eq. (8.1) can be formulated into an RL problem by utilizing the Lagrange multiplier $\lambda \geq 0$ to balance the extrinsic and intrinsic reward [39]

$$r^z = r_e + \lambda^z r_i^z, \quad \forall z, \tag{8.3}$$

The diversity objective that we seek to optimize operates over the distance in expected skill successor representations ψ^z.

$$\max_{d_\pi^1,...,d_\pi^n} 0.5 \sum_{z=1}^{n} \min_{k \neq z} \left\| \psi^z - \psi^k \right\|_2^2. \tag{8.4}$$

More specifically, given a feature mapping $\phi : \mathcal{S} \to \mathbb{R}^n$, the feature expectations are defined by $\psi^z = \mathbb{E}_{d_\pi^z(s)}[\phi(s)]$.

Similarly to Zahavy et al. [359], we introduce the VDW objective

$$\max_{d_\pi^1,\ldots,d_\pi^n} 0.5 \sum_{z=1}^{n} \ell_z^2 - 0.2(\ell_z^5/\ell_0^3), \tag{8.5}$$

where $\ell_z = \min_{k \neq z} \left\| \psi^z - \psi^k \right\|_2$, which allows the level of diversity to be controlled by ℓ_0.

Our algorithm relies on the following two properties: i) feature expectations satisfy

$$\psi^z = \mathop{\mathbb{E}}_{\rho(s_0)}[\psi^z(s_0)], \tag{8.6}$$

where $\rho(s_0)$ is the initial state distribution; and ii) SFs $\psi^z(s)$ can be trained by a learning process similar to training a value function, using Temporal Difference (TD) updates [312], which minimizes the loss

$$\mathcal{L}_\psi = \sum_{z=1}^{n} \mathop{\mathbb{E}}_{\pi^z} \left\| \phi(s) + \gamma \psi^z(s') - \psi^z(s) \right\|_2^2 . \tag{8.7}$$

At each time step, the intrinsic reward r_i is computed from the learned SFs $\psi(s)$ either by the repulsive force in eq. (8.4),

$$r_i^z(s) = \langle \phi(s), \psi^z - \psi^{z^\star} \rangle, \tag{8.8}$$

or by the VDW force in eq. (8.5),

$$r_i^z(s) = (1 - (\ell_z/\ell_0)^3) \langle \phi(s), \psi^z - \psi^{z^\star} \rangle. \tag{8.9}$$

Figure 8.2 showcases the results on the real robot, where different skills are obtained to traverse the obstacles.

8.2 Relation to this Thesis

DoMiNiC is a novel method that provides ways to optimize diversity while ensuring that multiple constraints are satisfied. There are two key differences to DOI from Chapter 6. Firstly, DoMiNiC is, similarly to WASABI(see Chapter 7) an online algorithm – it requires extensive amount of interactions with simulation to extract the skills. On the other hand, unlike DOI and WASABI, the algorithm doesn't require an additional demonstration dataset, however, it does require well-defined reward functions for which it can balance the value constraints.

[i] Figure attributed to Jin Cheng.

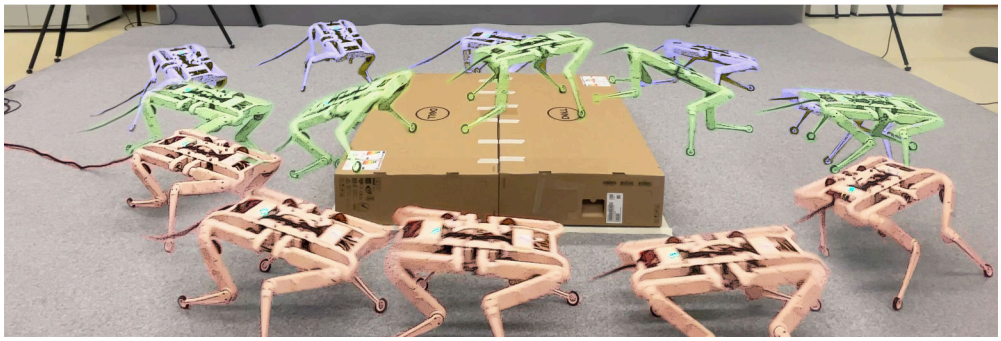


Figure 8.2: Obstacle experiment on hardware, we observe that the extracted skills explore different options in solving the obstacle. We have skills that go over the obstacle, to the right or to the left in different styles. The green skill is the one closest to the expert, which never takes detours around the box.[i]

9 Diffusion Generative Inverse Design

This chapter is based on the work "Diffusion Generative Inverse Design" [333]. Motivated by solving design problems which are mostly described by a cost function, we formulate the problem as sampling from a Boltzmann distribution and propose efficient sampling schemes based on a guided diffusion prior distribution by the gradient of the energy.

9.1 Summary

Substantial improvements to our way of life hinge on devising solutions to engineering challenges, an area in which Machine Learning (ML) advances is poised to provide positive real-world impact. Many such problems can be formulated as designing an object that gives rise to some desirable physical dynamics (e.g. designing an aerodynamic car or a watertight vessel). Here we are using ML to accelerate this design process by learning both a forward model of the dynamics and a distribution over the design space.

Prior approaches to ML-accelerated design have used neural networks as a differentiable forward model for optimization [54, 63, 111]. We build on work in which the forward model takes the specific form of a GNN trained to simulate fluid dynamics [8]. Since the learned model is differentiable, design optimization can be accomplished with gradient-based approaches (although these struggle with zero or noisy gradients and local minima) or sampling-based approaches (although these fare poorly in high-dimensional design spaces). Both often require multiple expensive calls to the forward model. However, generative models can be used to propose plausible designs, thereby reducing the number of required calls [99, 366, 173].

In this work, we use DDMs to optimize designs by sampling from a target distribution informed by a learned data-driven prior. DDMs have achieved extraordinary results in im-

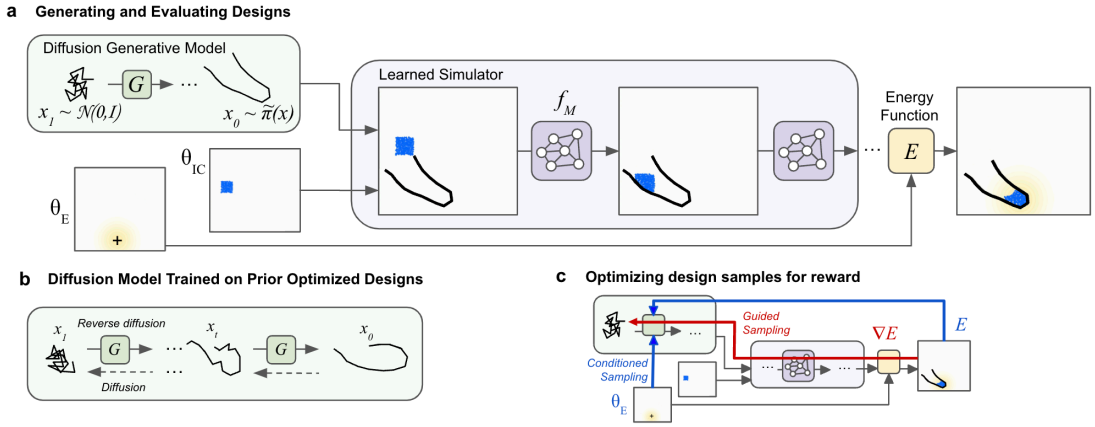


Figure 9.1: (a) Given initial conditions governed by θ_{IC}, energy function parameters θ_E, and learned GNN dynamics model f_M, design samples x from the diffusion model are assigned a cost $E(x)$. (b) Schematic of the DDM training (c) Gradients ∇E and conditioning set (θ_E and E) inform energy and conditional guidance, resp.

age generation [303, 306, 151, 138], and has since been used to learn efficient planners in sequential decision making and reinforcement learning [147, 6], sampling on manifolds [76] or constrained optimization formulations [116]. Our primary contribution is to consider DDM in the setting of physical problem solving. We find that such models combined with continuous sampling procedures enable to solve design problems orders of magnitude faster than off-the-shelf optimizers such as CEM and Adam. This can be further improved by utilizing a particle sampling scheme to update the base distribution of the diffusion model which by cheap evaluations (few ODE steps) with a learned model leads to better designs in comparison to vanilla sampling procedures. We validate our findings on multiple experiments in a particle fluid design environment.

9.2 Method

Given some task specification $\boldsymbol{c}$, we have a target distribution of designs $\pi(x)$ which we want to optimize w.r.t. x. To simplify notation, we do not emphasize the dependence of π on $\boldsymbol{c}$. This distribution is a difficult object to handle, since a highly non-convex cost landscape might hinder efficient optimization. We can capture prior knowledge over 'sensible' designs in form of a prior distribution $p(x)$ learned from existing data. Given a prior, we may sample from the distribution

$$\tilde{\pi}(x) \propto p(x)\pi(x), \tag{9.1}$$

which in this work is achieved by using a diffusion method with guided sampling. The designs will subsequently be evaluated by a learned forward model comprised of a pre-trained GNN simulator and a reward function [8, 246, 279].

Let $E : \mathbb{X} \mapsto \mathbb{R}$ be the cost (or "energy") of a design $x \in \mathbb{X}$ for a specific task $\boldsymbol{c}$ under the learned simulator (dependence of E on $\boldsymbol{c}$ is omitted for simplicity). The target distribution of designs $\pi(x)$ is defined by the Boltzmann distribution

$$\pi(x) := \frac{1}{Z} \exp\left(-\frac{E(x)}{\tau}\right), \tag{9.2}$$

where Z denotes the unknown normalizing constant and τ a temperature parameter. As $\tau \to 0$, this distribution concentrates on its modes, that is on the set of the optimal designs for the cost $E^{\boldsymbol{c}}(x)$. Direct methods to sample from $\pi(x)$ rely on expensive Markov chain Monte Carlo techniques or variational methods minimizing a reverse KL criterion.

We will rely on a data-driven prior learned by the diffusion model from previous optimization attempts. We collect optimization trajectories of designs for different task parametrizations $\boldsymbol{c}$ using Adam [158] or CEM [274] to optimize x. Multiple entire optimization trajectories of designs are included in the training set for the generative model, providing a mix of design quality. These optimization trajectories are initialized to flat tool(s) below the fluid, which can be more easily shaped into successful tools than a randomly initialized one. Later, when we compare the performance of the DDM to Adam and CEM, we will be using randomly initialized tools for Adam and CEM, which is substantially more challenging.

9.2.1 Diffusion Generative Models

We use DDMs to fit $p(x)$ [138, 306]. The core idea is to initialize using training data $x_0 \sim p$, captured by a diffusion process $(x_t)_{t\in[0,1]}$ defined by

$$\mathrm{d}x_t = -\beta_t x_t \mathrm{d}t + \sqrt{2\beta_t}\mathrm{d}w_t, \tag{9.3}$$

where $(w_t)_{t\in[0,1]}$ denotes the Wiener process. We denote by $p_t(x)$ the distribution of x_t under (9.3). For β_t large enough, $p_1(x) \approx \mathcal{N}(x; 0, \mathbf{I})$. The time-reversal of (9.3) satisfies

$$\mathrm{d}x_t = -\beta_t[x_t + 2\nabla_x \log p_t(x_t)]\mathrm{d}t + \sqrt{2\beta_t}\mathrm{d}w_t^-, \tag{9.4}$$

where $(w_t^-)_{t\in[0,1]}$ is a Wiener process when time flows backwards from $t = 1$ to $t = 0$, and $\mathrm{d}t$ is an infinitesimal negative timestep. By initializing (9.4) using $x_1 \sim p_1$, we obtain $x_0 \sim p$. In practice, the generative model is obtained by sampling an approximation of (9.4), replacing $p_1(x)$ by $\mathcal{N}(x; 0, I)$ and the intractable score $\nabla_x \log p_t(x)$ by $s_\theta(x, t)$. The score estimate $s_\theta(x, t)$ is learned by denoising score matching, i.e. we use the fact that $\nabla_x \log p_t(x) =$

$\int \nabla_x \log p(x_t|x_0)p(x_0|x_t)\mathrm{d}x_0$ where $p(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\alpha_t}x_0, \sqrt{1-\alpha_t}\mathbf{I})$ is the transition density of (9.3), α_t being a function of $(\beta_s)_{s\in[0,t]}$ [306]. It follows straightforwardly that the score satisfies $\nabla_x \log p_t(x) = -\mathbb{E}[\varepsilon|x_t = x]/\sqrt{1-\alpha_t}$ for $x_t = \sqrt{\alpha_t}x_0 + \sqrt{1-\alpha_t}\varepsilon$. We then learn the score by minimizing

$$\mathcal{L}(\theta) = \mathop{\mathbb{E}}_{x_0\sim p, t\sim\mathcal{U}(0,1), \varepsilon\sim\mathcal{N}(0,I)} \|\varepsilon_\theta(x_t, t) - \varepsilon\|^2, \tag{9.5}$$

where $\varepsilon_\theta(x,t)$ is a denoiser estimating $\mathbb{E}[\varepsilon|x_t = x]$. The score function $s_\theta(x,t) \approx \nabla_x \log p_t(x)$ is obtained using

$$s_\theta(x,t) = -\frac{\varepsilon_\theta(x,t)}{\sqrt{1-\alpha_t}}. \tag{9.6}$$

Going forward, ∇ refers to ∇_x unless otherwise stated. We can also sample from $p(x)$ using an ordinary differential equation (ODE) developed in [306].

Let us define $\bar{x}_t = x_t/\sqrt{\alpha_t}$ and $\sigma_t = \sqrt{1-\alpha_t}/\sqrt{\alpha_t}$. Then by initializing $x_1 \sim \mathcal{N}(0, I)$, equivalently $\bar{x}_1 \sim \mathcal{N}(0, \alpha_1^{-1}I)$ and solving backward in time

$$\mathrm{d}\bar{x}_t = \varepsilon_\theta^{(t)}\left(\frac{\bar{x}_t}{\sqrt{\sigma_t^2+1}}\right)\mathrm{d}\sigma_t, \tag{9.7}$$

then $x_0 = \sqrt{\alpha_t}\,\bar{x}_0$ is an approximate sample from $p(x)$.

9.2.2 Approximately Sampling from Target Distribution

We want to sample $\tilde{\pi}(x)$ defined in (9.1) where $p(x)$ can be sampled from using the diffusion model. We describe two possible sampling procedures with different advantages for downstream optimization.

Energy guidance. Observe that

$$\tilde{\pi}_t(x_t) = \int \tilde{\pi}(x_0)p(x_t|x_0)\mathrm{d}x_0,$$

and the gradient satisfies

$$\nabla \log \tilde{\pi}_t(x_t) = \nabla \log p_t(x_t) + \nabla \log \pi_t(x_t),$$

where $\pi_t(x_t) = \int \pi(x_0)p(x_0|x_t)\mathrm{d}x_0$. We approximate this term by making the approximation

$$\begin{aligned} \hat{x}(x_t,t) &= \underbrace{\left(\frac{x_t - \sqrt{1-\alpha_t}\varepsilon_\theta(x_t,t)}{\sqrt{\alpha_t}}\right)}_{\text{“ estimated } x_0\text{”}}, \\ \pi_t(x_t) &\approx \pi(\hat{x}_t(x_t,t)). \end{aligned} \tag{9.8}$$

Now, by (9.6), and the identity $\nabla \log \pi(x) = -\tau^{-1}\nabla E(x)$, we may change the reverse sampling procedure by a modified denoising vector

$$\tilde{\varepsilon}_\theta(x_t, t) = \varepsilon_\theta(x_t, t) + \lambda\tau^{-1}\sqrt{1-\alpha_t}\nabla E(\hat{x}(x_t, t)), \tag{9.9}$$

with λ being an hyperparameter.

Conditional guidance. Similarly to *classifier-free* guidance [139], we explore conditioning on cost (energy) $\boldsymbol{e}$ and task $\boldsymbol{c}$. A modified denoising vector in the reverse process follows as a combination between the denoising vector of a conditional denoiser ε_ϕ and unconditional denoiser ε_θ

$$\tilde{\varepsilon}(x_t, \boldsymbol{c}, \boldsymbol{e}, t) = (1+\lambda)\varepsilon_\phi(x_t, \boldsymbol{c}, \boldsymbol{e}, t) - \lambda\varepsilon_\theta(x_t, t), \tag{9.10}$$

where ε_ϕ is learned by conditioning on $\boldsymbol{c}$ and cost $\boldsymbol{e}$ from optimization trajectories. In our experiments we shall choose $\boldsymbol{c}$ to contain the design cost percentile and target goal destination θ_E for fluid particles (Figure 9.1c).

9.2.3 A Modified Base Distribution through Particle Sampling

Our generating process initializes samples at time $t = 1$ from $\mathcal{N}(x; 0, \mathbf{I}) \approx p_1(x)$. The reverse process with modifications from Section 9.2.2 provides approximate samples from $\tilde{\pi}(x)$ at $t = 0$. However, as we are approximately solving the ODE of an approximate denoising model with an approximate cost function, this affects the quality of samples with respect to E[i]. Moreover, "bad" samples from $\mathcal{N}(x; 0, \mathbf{I})$ are hard to correct by guided sampling.

9.3 Relation to this Thesis

This chapter presents a compelling application of advanced machine learning techniques to solve complex design optimization problems efficiently. It relates closely to the central themes of this thesis, which are learning under constraints and promoting diversity in learning models.

9.3.1 Learning under Constraints

The use of DDMs in the design process inherently addresses the theme of learning under constraints. The optimization process in inverse design is constrained by the physical and practical limitations of the designs themselves, as well as by the need for efficient computation.

[i] Ideally at test time we would evaluate the samples with the ground-truth dynamics model, but we have used the approximate GNN model due to time constraints on the project.

Algorithm 6 Particle optimization of base distribution.

input energy function E, diffusion generative model p_θ, temperature τ, noise scale σ, rounds K.
$\mathbb{S}_1^0 = \{x_1^i\}_{i=1}^N$ for $x_1^i \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, I)$
$\mathbb{S}_0 = \emptyset, \mathbb{S}_1 = \emptyset$ # $t = 0$ and $t = 1$ sample sets
for $k \in \{0 \dots K\}$ **do**
  Compute $\mathbb{S}_0^k = \{x_0^i\}_{i=1}^N$ from $\mathbb{S}_1^k$ by solving reverse ODE in Eq. (9.7).
  $\mathbb{S}_0 = \mathbb{S}_0 \cup \mathbb{S}_0^k, \mathbb{S}_1 = \mathbb{S}_1 \cup \mathbb{S}_1^k \sum_0^{|\mathbb{S}_T^k|} \exp(\frac{c(x_T)}{\tau})$
  Compute normalized importance weights
  $\mathbb{W} = \left\{ w \,|\, w \propto \exp\left(-\frac{E(x_0)}{\tau}\right),\ x_0 \in \mathbb{S}_0 \right\}$
  Set $\bar{\mathbb{S}}_1^{k+1} = \{\bar{x}_1^i\}_{i=1}^{|\mathbb{S}_1|}$ for $\bar{x}_1^i \overset{\text{i.i.d.}}{\sim} \sum_{i=1}^{|\mathbb{S}_1|} w^i \delta_{x_1^i}(x)$
  Set $\mathbb{S}_1^{k+1} = \{\tilde{x}_1^i\}_{i=1}^{|\mathbb{S}_1|}$ for $\tilde{x}_1^i \sim \mathcal{N}(x; \bar{x}_1^i, \sigma^2 I)$
return $\arg\min_{x \in \mathbb{S}_0} E(x)$

DDMs, by learning a distribution over the design space informed by prior data, effectively navigate these constraints. They optimize design parameters within the bounds of feasible and practical solutions, ensuring that the generated designs are not only innovative but also applicable and realizable under real-world conditions. Moreover, the introduction of methods like energy guidance and conditional guidance in the sampling process allows for the incorporation of specific constraints directly into the learning model. These methods adjust the generative process based on predefined energy functions or conditions related to the task, ensuring that the solutions adhere to necessary specifications and constraints.

9.3.2 Facilitating Diversity

Diversity in learning is crucial for developing robust and generalizable models. The chapter's exploration of DDMs contributes to this by utilizing a generative approach that samples from a diverse set of potential designs. This is particularly evident in the particle sampling scheme, which refines the base distribution from which designs are sampled. By re-evaluating and resampling designs based on their energy, the model promotes a diverse set of solutions, rather than converging prematurely on a limited number of designs.

This approach not only enhances the diversity of the generated designs but also aligns with the thesis's emphasis on diversity in learning paradigms. By learning from a broad set of data and continuously updating the sampling process based on performance metrics, DDMs embody the principles of diversity and adaptability that are central to this thesis.

IV

Epilogue

The only true wisdom is knowing that you know nothing.

Socrates

10 Conclusion

10.1 Combinatorial Inductive Biases

This thesis has dealt with two main challenges. The first one is introducing structure in the prediction of neural networks by means of combinatorial solvers as implicit layers. We have tackled the challenge of optimizing such architectures by a clever way of computing the gradient of a piecewise affine interpolation of the original piecewise constant objective with the cost of one additional call to the solver on a modified combinatorial problem on the backward pass. Most importantly, the proposed method, while principled, is completely ambivalent towards the solver being embedded. The only requirement is the linearity of the cost. Furthermore, the additional call to the solver can be avoided completely if the solver is treated as negative identity in the computational graph when computing the gradient, which works just as well in many cases. We have shown that modifying standard neural network architectures with combinatorial layers leads to substantial improvements in generalization capabilities of the models in real-world tasks, such as object detection and retrieval by formulating the ranking problem as a combinatorial arg min problem (Chapter 3) or receding horizon planning where the underlying solver is Dijkstra's algorithm applied on a TDSP (Chapter 4). With this, we confirm the hypothesis that combinatorial inductive biases are an essential ingredient for models in problem settings that contain "hidden" combinatorial structure.

Still, some open questions remain. Throughout this thesis, we assumed that we have an oracle that tells us which combinatorial solver we should choose for a particular problem. Once we know the problem, we can learn the underlying cost function. If we want to have intelligent systems that are general problem solvers, this is a very limiting assumption. A promising future direction is to lift this restriction, which would lead to learning the feasible set Y, or the constraint set. It is not clear if this problem might be too hard to provide any kind of boost in generalization, the search over all possible solvers for the problem might be

as hard as solving the problem without having access to the solver. Furthermore, we have investigated combinatorial solver layers in architectures that cannot deal with arbitrary input sizes, meaning that the maximal combinatorial instance size that we can deal with at inference time is fixed and cannot be changed. An interesting question is if we can achieve the same generalization properties with architectures that can deal with arbitrary input sizes, such as transformers or GNNs, and does learning from smaller combinatorial instances transfer to larger ones?

10.2 Model-based Risk Averseness

While in Chapter 2 we deal with constraints that are imposed on the model rather explicitly, when making decisions it is essential to consider the model's uncertainty about its predictions coupled together with any explicit constraints. This is in particular important in cases where we don't necessarily care about the expected return, but rather about different quantiles of the return distribution. In Chapter 5 we have seen the benefits of incorporating uncertainty in zero-order trajectory optimizers such as CEM. To this end, the separation of uncertainties into epistemic and aleatoric has proven to be crucial, which we achieve by an ensemble-based method. By choosing an ensemble of models and a PETS sampling scheme, we avoid the difficult problem of propagating uncertainty through autoregressive predictions, which would require further restrictions on the model architecture or computationally expensive approximations. In addition, we have shown that we can incorporate probabilistic safety constraints based on an MLE fit for a local approximation with a Gaussian across different samples from the trajectory distribution.

The control of the policy's risk-averseness is achieved by simple incorporation of the aleatoric, epistemic and safety terms into the cost of the trajectory optimizer. Usage of the epistemic uncertainty is two-fold, firstly we use it in form of an epistemic bonus for obtaining better data for fitting the model, making the learning much more efficient. Secondly, at inference time, all uncertainty is treated as a cost that is scaled by a hyperparameter. This is a very simple way of incorporating risk-averseness into the planning method, an interesting avenue of future research is to connect the uncertainty with the cost function of the task being solved.

The proposed method is very general and can be applied to any zero-order model-based trajectory optimizer. In our empirical studies we have shown for several control tasks that the proposed method increases robustness of the resulting policy in environments where there is stochasticity in the dynamics by incorporating the uncertainty penalty at inference time. Furthermore, the espitemic reward for exploration has shown to greatly improve the model's learning efficiency in environments that are hard to explore. The tasks that we considered were all modelled with the Mujoco physics engine, and all the data was coming from simula-

tion. However, the perhaps more interesting setting is when we transfer from simulation to the real system. There are several challenges that need to be addressed, such as the domain gap between the simulation and the real system, or the sampling speed of the policy. One can expect that this is a realistic setting where the proposed method can really make a difference, since we are bound to encounter dynamics that are not perfectly modelled by the simulator.

By not imposing architectural constraints on the model, we have full flexibility for fitting the system dynamics. However, this comes with the sacrifice of not being able to obtain exact likelihood estimates of the resulting trajectory, since we cannot reliably propagate uncertainty through predictions of an arbitrary neural network. In case we would utilize generative models for which exact likelihood computation is possible, such as normalizing flows, this still doesn't provide us with an efficient way of integrating over the violation set, which is essential for computing the probability of constraint violation. Furthermore, with such models, we are still left with the problem of separating the epistemic and aleatoric uncertainties. An interesting direction of future research is to investigate how to incorporate more powerful generative models which can give us an exact likelihood estimate or an efficient way of its approximation, while still being able to separate the uncertainties.

10.3 Diversity Subject to Quality Constraints

Aligned with the desiderata that intelligent agents should find novel ways of solving problems which we frame as a quality-diversity tradeoff [38, 50], we have investigated in Chapter 6 another setting where constraints are of interest, the offline imitation learning setting where we want to extract diverse behaviors from a dataset of expert demonstrations without sacrificing quality, which we measure by closeness to the expert state occupancy distribution. To the extent of our knowledge, this is the first such algorithm that balances diversity and quality in the offline imitation learning setting. The constraint that we consider is provided in form of an f-divergence between the state-occupancy distributions of the reference expert policy and the learned policy. The diversity signal on the other hand is the mutual information $\mathcal{I}(Z; S)$, which indicates how dependent is the state occupancy distribution on the latent skill variable z. By using a variational lower bound on the mutual information, we obtain a tractable objective that involves a skill discriminator. With clever use of the Fenchel dual RL formulation [221], we obtain a value learning problem from which the optimal value function can be mapped back to the primal occupancy distribution solution. This enables the usage of an importance-weighted objective for learning the optimal discriminator and optimal policy. However, in the approximation setting, we cannot solve this problem in one go, since when the occupancy distribution of the policy shifts, so does the reward function with which we obtain the importance weights.

The corresponding reward function in the value learning problem consists of two inter-

pretable parts that are balanced by Lagrange multipliers, the diversity part which indicates how certain is the discriminator of the proposed latent skill variable and the quality part which indicates how close is the state occupancy distribution to the expert's. We have evaluated the proposed method, which we name DOI, on several quadruped locomotion tasks in both simulation and the real system, and have shown that it is able to extract diverse behaviors. Moreover, the resulting behaviors are different from the expert's behaviors, meaning that the method generalizes.

However, DOI is not without its limitations. First and foremost, it involves an alternating optimization procedure that involves alternating between value learning, reward adjustment and Lagrange multiplier updates. This procedure needs to be carefully tuned in order to achieve desired results. In future work, it would be interesting to investigate how we can reduce the complexity. Furthermore, it is questionable if the mutual information is the best diversity signal – there might be simpler diversity objectives that lead to more stable training.

Going beyond the mutual information lower bound, we show how an alternative diversity objective can be formulated without the mutual information, which would alleviate the problem of learning a discriminator at the cost of choosing an adequate feature space for the diversity signal. This would rely on the maximum L_2 distance of the expected successor feature representations, which are computable by importance-weighted expectation estimates via obtained importance ratios from the value learning problem. The corresponding reward function for this is the gradient of the diversity objective with respect to the occupancy distribution. We leave empirical evaluation of this method for future work.

Learning by imitation is going to be an important part of the future of AI and leveraging offline data that is massively being collected has shown to yield impressive results. However, most of the methods that are currently used in the field don't consider the absence of actions in the demonstrations. This is a very important aspect of the problem, furthermore, alleviating the requirement that the agent needs to match the expert's state occupancy exactly is beneficial, since the agent can find novel ways of solving the task that are more suitable for its embodiment and are more robust. It would be interesting to expand the proposed method to the large-scale setting where demonstrations are collected from multiple sources and more complex state spaces such as images in videos or other modalities, which would be a significant step towards more adaptive AI.

10.4 Outlook

As an overarching theme, this thesis has explored the beneficial role of constraints in enhancing the learning process of intelligent systems and ensuring their reliability during inference. There are well-founded arguments for the necessity of combinatorial generalization in artificial general intelligence [24]. To address this, we demonstrated that incorporating combina-

torial solvers to constrain the model's predictions serves as a powerful inductive bias, leading to improved generalization properties, with guarantees on the structure of the output. This is especially the case when we can pinpoint the type of underlying combinatorial problem that needs to be solved. In the context of dynamical systems requiring sequential decision-making, we presented methods to enhance robustness in both online and offline settings. In the online setting, we incorporated model uncertainty within a zero-order trajectory optimizer to make more robust decisions and enable efficient exploration. This is made possible by the separation of uncertainties which ensures that the agent's focus remains on "obtainable" knowledge. As has been argued by several works, the trait of creativity is a fine balance between quality and diversity [38, 50]. Operating under this desiderata, in the offline setting, we leveraged f-divergence constraints to balance quality and diversity, extracting behaviors that differ from the expert's while still achieving high task performance. Notably, in these cases, constraints were applied not to the model's architecture, but to its behavior either at inference time or during the learning process. The range of problems where constrained predictions or behaviors are important is vast, and their effect on learning is substantial. Beyond the work presented in this thesis, the recent wide adoption of large-scale generative models raises important questions about safety, reliability and creativity in AI. A delicate balance needs to be struck between generating something that is known versus something that is novel and useful. Ideas presented in this thesis provide a foundation for future research in these topics and application areas - I hope that they will inspire further work in this direction, with the goal of creating systems that leverage constraints to boost generalization and facilitate controlled exploration of the unknown .

List of Publications

Marin Vlastelica, Tatiana López-Guevara, Kelsey Allen, Peter Battaglia, Arnaud Doucet, and Kimberley Stachenfeld. "*Diffusion Generative Inverse Design*". SPIGM Workshop @ International Conference on Machine Learning (ICML), 2023.

Núria Armengol Urpí, Marco Bagatella, ***Marin Vlastelica***, and Georg Martius. "*Causal Action Influence Aware Counterfactual Data Augmentation*". International Conference on Machine Learning (ICML), 2024.

Marin Vlastelica, Jin Cheng, Georg Martius and Pavel Kolev. "*Offline Diversity Maximization under Imitation Constraints*". Reinforcement Learning Conference (RLC), 2024.

Cian Eastwood, Shashank Singh, Andrei L Nicolicioiu, ***Marin Vlastelica***, Julius von Kügelgen, and Bernhard Schölkopf. "*Spuriosity Didn't Kill the Classifier: Using Invariant Predictions to Harness Spurious Features*". Advances in Neural Information Processing Systems (NeurIPS), 2024.

Jin Cheng, ***Marin Vlastelica***, Chenhao Li, Pavel Kolev and Georg Martius. "*Learning Diverse Skills for Local Navigation under Multi-constraint Optimality*". IEEE International Conference on Robotics and Automation (ICRA), 2023.

Chenhao Li, ***Marin Vlastelica***, Sebastian Blaes, Jonas Frey, Felix Grimminger, and Georg Martius. "*Learning Agile Skills via Adversarial Imitation of Rough Partial Demonstrations*". Conference on Robot Learning (CoRL), 2023.

Marin Vlastelica, Patrick Ernst, and Gyuri Szarvas. "*Taming Continuous Posteriors for Latent Variational Dialogue Policies*". AAAI Conference on Artificial Intelligence (AAAI), 2023.

Subham Sekhar Sahoo*, Anselm Paulus*, ***Marin Vlastelica***, Vít Musil, Volodymyr Kuleshov, and Georg Martius. "*Backpropagation through Combinatorial Algorithms: Identity with Projection Works*". International Conference on Learning Representations (ICLR), 2022.

Marin Vlastelica*, Sebastian Blaes*, Cristina Pinneri, and Georg Martius. "*Risk-Averse Zero-Order Trajectory Optimization*". Conference on Robot Learning (CoRL), 2021.

Marin Vlastelica, Michal Rolínek, and Georg Martius. "*Neuro-algorithmic Policies enable Fast Combinatorial Generalization*". International Conference on Machine Learning (ICML), 2021.

Michal Rolínek, Vít Musil, Anselm Paulus, ***Marin Vlastelica***, Claudio Michaelis, and Georg Martius. "*Optimizing Rank-Based Metrics With Blackbox Differentiation*". IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020.

Marin Vlastelica*, Anselm Paulus*, Vít Musil, Georg Martius, and Michal Rolínek. "*Differentiation of Blackbox Combinatorial Solvers*". International Conference on Learning Representations (ICLR), 2020.

Sebastian Blaes, ***Marin Vlastelica***, Jiajie Zhu, and Georg Martius. "*Control What You Can: Intrinsically Motivated Task-Planning Agent*". Advances in Neural Information Processing Systems (NeurIPS), 2019.

Bibliography

[1] Ian Abraham, Ankur Handa, Nathan Ratliff, Kendall Lowrey, Todd D. Murphey, and Dieter Fox. "Model-Based Generalization Under Parameter Uncertainty Using Path Integral Control". In: *IEEE Robotics and Automation Letters* 2 (2020), (cit. on p. 97).

[2] Joshua Achiam, Harrison Edwards, Dario Amodei, and Pieter Abbeel. "Variational Option Discovery Algorithms". In: *CoRR* (2018) (cit. on p. 110).

[3] Sander Adam, Lucian Buşoniu, and Robert Babuska. "Experience Replay for Real-Time Reinforcement Learning Control". In: *IEEE Transactions on Systems, Man, and Cybernetics* 2 (2012), (cit. on p. 35).

[4] Akshay Agrawal, Brandon Amos, Shane T. Barratt, Stephen P. Boyd, Steven Diamond, and J. Zico Kolter. "Differentiable Convex Optimization Layers". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2019, (cit. on p. 2).

[5] Akshay Agrawal, Shane Barratt, Stephen Boyd, Enzo Busseti, and Walaa M Moursi. "Differentiating through a cone program". In: *Journal of Applied Numerical Optimization* 2 (2019), (cit. on p. 2).

[6] Anurag Ajay, Yilun Du, Abhi Gupta, Joshua B Tenenbaum, Tommi S Jaakkola, and Pulkit Agrawal. "Is Conditional Generative Modeling all you need for Decision-Making?" In: *NeurIPS 2022 Foundation Models for Decision Making Workshop*. 2022 (cit. on p. 138).

[7] David Aldous and Persi Diaconis. "Shuffling Cards and Stopping Times". In: *The American Mathematical Monthly* () (cit. on p. 6).

[8] Kelsey R. Allen, Tatiana Lopez-Guevara, Kimberly L. Stachenfeld, Alvaro Sanchez-Gonzalez, Peter W. Battaglia, Jessica B. Hamrick, and Tobias Pfaff. "Physical Design using Differentiable Learned Simulators". In: *CoRR* (2022) (cit. on pp. 137, 139).

[9] Eitan Altman. "Constrained Markov decision processes". 1999 (cit. on p. 110).

[10] Brandon Amos and J. Zico Kolter. "OptNet: Differentiable Optimization as a Layer in Neural Networks". In: *arXiv*. ICML (2017), (cit. on p. 47).

[11] Brandon Amos, Lei Xu, and J Zico Kolter. "Input convex neural networks". In: *International Conference on Machine Learning*. ICML. JMLR. 2017, (cit. on p. 47).

[12] Marcin Andrychowicz, Dwight Crow, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Joshua Tobin, P. Abbeel, and Wojciech Zaremba. "Hindsight Experience Replay". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2017, (cit. on p. 35).

[13] Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, CK Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Michael Suo, Phil Tillet, Eikan Wang, Xiaodong Wang, William Wen, Shunting Zhang, Xu Zhao, Keren Zhou, Richard Zou, Ajit Mathews, Gregory Chanan, Peng Wu, and Soumith Chintala. "PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation". In: *ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 2024 (cit. on p. 12).

[14] Martín Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein Generative Adversarial Networks". In: *International Conference on Learning Representations*. ICML (2017), (cit. on p. 5).

[15] Ermano Arruda, Michael J. Mathew, Marek Kopicki, M. Mistry, M. Azad, and J. Wyatt. "Uncertainty averse pushing with model predictive path integral control". In: *IEEE International Conference on Humanoid Robotics* (2017), (cit. on p. 97).

[16] Laura A. Atherton, David Dupret, and Jack R. Mellor. "Memory trace replay: the shaping of memory consolidation by neuromodulation". In: *Trends in Neurosciences* 9 (2015), (cit. on p. 35).

[17] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. "Deep Equilibrium Models". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2019 (cit. on pp. 2, 23, 24).

[18] Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. "Learning to Branch". In: *International Conference on Machine Learning*. ICML. 2018, (cit. on p. 46).

[19] Mislav Balunovic, Pavol Bielik, and Martin Vechev. "Learning to Solve SMT Formulas". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2018, (cit. on p. 46).

[20] Omer Bar-Tal, Hila Chefer, Omer Tov, Charles Herrmann, Roni Paiss, Shiran Zada, Ariel Ephrat, Junhwa Hur, Yuanzhen Li, Tomer Michaeli, Oliver Wang, Deqing Sun, Tali Dekel, and Inbar Mosseri. "Lumiere: A Space-Time Diffusion Model for Video Generation". 2024 (cit. on p. 1).

[21] Andre Barreto, Diana Borsa, John Quan, Tom Schaul, David Silver, Matteo Hessel, Daniel Mankowitz, Augustin Zidek, and Remi Munos. "Transfer in deep reinforcement learning using successor features and generalised policy improvement". In: *International Conference on Machine Learning*. ICML. PMLR. 2018, (cit. on p. 111).

[22] André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado Van Hasselt, and David Silver. "Successor features for transfer in reinforcement learning". In: *arXiv preprint arXiv:1606.05312* (2016) (cit. on pp. 111, 224).

[23] Brian T. Bartell, Garrison W. Cottrell, and Richard K. Belew. "Automatic Combination of Multiple Ranked Retrieval Systems". In: *ACM Conference on Research and Development in Information Retrieval*. SIGIR'94. 1994, (cit. on p. 61).

[24] Peter Battaglia, Jessica Blake Chandler Hamrick, Victor Bapst, Alvaro Sanchez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andy Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Jayne Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. "Relational inductive biases, deep learning, and graph networks". In: *arXiv* (2018) (cit. on pp. 1, 2, 45, 148).

[25] "Bayesian Estimates of Equation System Parameters: An Application of Integration by Monte Carlo". In: *Econometrica: Journal of the Econometric Society* (1978), (cit. on p. 35).

[26] Aurélien Bellet, Amaury Habrard, and Marc Sebban. "A survey on metric learning for feature vectors and structured data". In: *arXiv preprint arXiv:1306.6709* (2013) (cit. on p. 63).

[27] Richard Bellman. "A Markovian Decision Process". In: *Journal of Mathematics and Mechanics* (1957), (cit. on pp. 28, 30).

[28] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. "Neural Combinatorial Optimization with Reinforcement Learning". In: *International Conference on Learning Representations*. ICLR. 2017 (cit. on p. 46).

[29] Boris Belousov and Jan Peters. "f-Divergence constrained policy improvement". 2017 (cit. on p. 23).

[30] Silvia Bernardi, Marcus K Benna, Mattia Rigotti, Jérôme Munuera, Stefano Fusi, and C Daniel Salzman. "The Geometry of Abstraction in the Hippocampus and Prefrontal Cortex". In: *Cell* (2020) (cit. on p. 2).

[31] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. "Dota 2 with large scale deep reinforcement learning". In: *arXiv preprint arXiv:1912.06680* (2019) (cit. on p. 1).

[32] Dimitri Bertsekas. "A course in reinforcement learning". 2023 (cit. on pp. 37, 38).

[33] Dimitri P Bertsekas, Dimitri P Bertsekas, Dimitri P Bertsekas, and Dimitri P Bertsekas. "Dynamic programming and optimal control". 1995 (cit. on p. 37).

[34] Dimitris Bertsimas and John N Tsitsiklis. "Introduction to Linear Optimization". 1997 (cit. on p. 15).

[35] Shalabh Bhatnagar and K Lakshmanan. "An online actor–critic algorithm with function approximation for constrained markov decision processes". In: *Journal of Optimization Theory and Applications* 3 (2012), (cit. on p. 113).

[36] Christopher M. Bishop. "Pattern Recognition and Machine Learning". 2006 (cit. on p. 25).

[37] Sebastian Blaes, Marin Vlastelica, Jiajie Zhu, and Georg Martius. "Control What You Can: Intrinsically Motivated Task-Planning Agent". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2019, (cit. on p. 86).

[38] Margaret A Boden. "The Creative Mind: Myths and Mechanisms". 2004 (cit. on pp. 147, 149).

[39] Vivek S Borkar. "An actor-critic algorithm for constrained Markov decision processes". In: *Systems & control letters* 3 (2005), (cit. on pp. 113, 134).

[40] Zdravko Botev, Dirk Kroese, Reuven Rubinstein, and Pierre L'Ecuyer. "Chapter 3. The Cross-Entropy Method for Optimization". In: 2013, (cit. on p. 97).

[41] Stephen P Boyd and Lieven Vandenberghe. "Convex optimization". 2004 (cit. on pp. 15, 21, 216).

[42] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. "JAX: composable transformations of Python+NumPy programs". Version 0.3.13. 2018 (cit. on p. 12).

[43] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil J. Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael S. Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. "RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control". In: (2023) (cit. on pp. 1, 110).

[44] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alexander Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil Joshi, Ryan C. Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael S. Ryoo, Grecia Salazar, Pannag R. Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Anand Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan Ho Vuong, F. Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. "RT-1: Robotics Transformer for Real-World Control at Scale". In: *Robotics, Science and Systems*. 2022 (cit. on p. 1).

[45] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. "Signature verification using a "siamese" time delay neural network". In: *Advances in Neural Information Processing Systems*. NeurIPS. 1994, (cit. on p. 63).

[46] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. "Language Models are Few-shot Learners". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2020, (cit. on p. 1).

[47] Christopher JC Burges. "From RankNet to LambdaRank to LambdaMART: An overview". In: *Learning* 23-581 (2010), (cit. on p. 23).

[48] Fatih Cakir, Kun He, Xide Xia, Brian Kulis, and Stan Sclaroff. "Deep Metric Learning to Rank". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019 (cit. on pp. 61, 62, 67, 68, 71–73, 75, 194–196).

[49] Fatih Cakir, Kun He, Xide Xia, Brian Kulis, and Stan Sclaroff. "Deep Metric Learning to Rank". Commit: 7ca48aa. 2019 (cit. on pp. 75, 191).

[50] Francisco Câmara Pereira. "Creativity and Artificial Intelligence: A Conceptual Blending Approach". 2007 (cit. on pp. 147, 149).

[51] Victor Campos, Alexander Trott, Caiming Xiong, Richard Socher, Xavier Giró-i-Nieto, and Jordi Torres. "Explore, Discover and Learn: Unsupervised Discovery of State-Covering Skills". In: *International Conference on Machine Learning*. ICML. 2020, (cit. on p. 110).

[52] Stephen Casper, Xander Davies, Claudia Shi, Thomas Krendl Gilbert, J'er'emy Scheurer, Javier Rando, Rachel Freedman, Tomasz Korbak, David Lindner, Pedro Freire, Tony Wang, Samuel Marks, Charbel-Raphaël Ségerie, Micah Carroll, Andi Peng, Phillip J. K. Christoffersen, Mehul Damani, Stewart Slocum, Usman Anwar, Anand Siththaranjan, Max Nadeau, Eric J. Michaud, Jacob Pfau, Dmitrii Krasheninnikov, Xin Chen, Lauro Langosco di Langosco, Peter Hase, Erdem Biyik, Anca D. Dragan, David Krueger, Dorsa Sadigh, and Dylan Hadfield-Menell. "Open Problems and Fundamental Limitations of Reinforcement Learning from Human Feedback". In: *Transactions on Machine Learning Research* (2023) (cit. on p. 3).

[53] Soumen Chakrabarti, Rajiv Khanna, Uma Sawant, and Chiru Bhattacharyya. "Structured learning for non-smooth ranking losses". In: *KDD*. 2008, (cit. on p. 61).

[54] Adithya Challapalli, Dhrumil Patel, and Gouqiang Li. "Inverse machine learning framework for optimizing lightweight metamaterials". In: *Materials & Design* (2021), (cit. on p. 137).

[55] Chun-Teh Chen and Grace X. Gu. "Generative Deep Neural Networks for Inverse Materials Design Using Backpropagation and Active Learning". In: *Advanced Science* 5 (2020), (cit. on p. 1).

[56] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. "MMDetection: Open MMLab Detection Toolbox and Benchmark". In: *arXiv preprint arXiv:1906.07155* (2019). Commit: 9d767a03c0ee60081fd8a2d2a200e530bebef8eb (cit. on pp. 62, 75).

[57] Kean Chen, Jianguo Li, Weiyao Lin, John See, Ji Wang, Lingyu Duan, Zhibo Chen, Changwei He, and Junni Zou. "Towards Accurate One-Stage Object Detection with AP-Loss". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. CVPR'19. 2019, (cit. on pp. 62, 63, 67, 70).

[58] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs". In: 2018, (cit. on p. 45).

[59] Liang-Chieh Chen, Alexander G. Schwing, Alan L. Yuille, and Raquel Urtasun. "Learning Deep Structured Models". In: *International Conference on Machine Learning*. ICML. 2015, (cit. on pp. 3, 45).

[60] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. "Neural ordinary differential equations". In: NeurIPS (2018), (cit. on pp. 2, 23).

[61] Ching-An Cheng, Tengyang Xie, Nan Jiang, and Alekh Agarwal. "Adversarially Trained Actor Critic for Offline Reinforcement Learning". In: *International Conference on Machine Learning*. ICML. 2022, (cit. on p. 111).

[62] Jin Cheng, Marin Vlastelica, Pavel Kolev, Chenhao Li, and Georg Martius. "Learning Diverse Skills for Local Navigation under Multi-constraint Optimality". In: *IEEE International Conference on Robotics and Automation*. ICRA. 2023 (cit. on pp. 3, 7, 119, 121, 133, 223).

[63] Thomas Christensen, Charlotte Loh, Stjepan Picek, Domagoj Jakobović, Li Jing, Sophie Fisher, Vladimir Ceperic, John D. Joannopoulos, and Marin Soljačić. "Predictive and generative machine learning models for photonic crystals". In: *Nanophotonics* 13 (2020), (cit. on p. 137).

[64] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. "Deep reinforcement learning from human preferences". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2017, (cit. on p. 33).

[65] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2018, (cit. on pp. 3, 4, 40, 96–99, 103, 205, 212).

[66] W. Clements, Benoit-Marie Robaglia, B. V. Delft, Reda Bahi Slaoui, and S'ebastien Toth. "Estimating Risk and Uncertainty in Deep Reinforcement Learning". In: *International Conference on Machine Learning Workshop on Uncertainty & Robustness in Deep Learning* (2020) (cit. on p. 97).

[67] Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. "Leveraging procedural generation to benchmark reinforcement learning". In: *International Conference on Machine Learning*. ICML. 2020, (cit. on pp. 80, 86, 88, 89, 198, 200, 201).

[68] R. Cohendet, Clair-Hélène Demarty, Ngoc Duong, Mats Sjöberg, Bogdan Ionescu, and Thanh-Toan Do. "MediaEval 2018: Predicting Media Memorability". In: *arXiv:1807.01052* (2018) (cit. on p. 61).

[69] Sebastian Curi, Felix Berkenkamp, and Andreas Krause. "Efficient Model-Based Reinforcement Learning through Optimistic Policy Search and Planning". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2020, (cit. on p. 97).

[70] Bo Dai, Niao He, Yunpeng Pan, Byron Boots, and Le Song. "Learning from conditional distributions via dual embeddings". In: *Artificial Intelligence and Statistics*. PMLR. 2017, (cit. on p. 216).

[71] Bo Dai, Ofir Nachum, Yinlam Chow, Lihong Li, Csaba Szepesvári, and Dale Schuurmans. "Coindice: Off-policy confidence interval estimation". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2020, (cit. on p. 112).

[72] George B Dantzig. "Linear Programming and Extensions". In: 2016 (cit. on p. 15).

[73] Aldous David and J Fill. "Reversible Markov chains and Random Walks on Graphs". In: *University of California, Berkeley* (1995) (cit. on p. 6).

[74] Nathaniel D Daw, Yael Niv, and Peter Dayan. "Uncertainty-based competition between prefrontal and dorsolateral striatal systems for behavioral control". In: *Nature Neuroscience* 12 (2005), (cit. on p. 80).

[75] Peter Dayan. "Improving Generalization for Temporal Difference Learning: The Successor Representation". In: *Neural Computation* 4 (1993), (cit. on pp. 6, 111).

[76] Valentin De Bortoli, Emile Mathieu, Michael Hutchinson, James Thornton, Yee Whye Teh, and Arnaud Doucet. "Riemannian Score-Based Generative Modelling". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2022 (cit. on p. 138).

[77] Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. "Gaussian processes for data-efficient learning in robotics and control". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2 (2013), (cit. on p. 97).

[78] Marc Peter Deisenroth and Carl Edward Rasmussen. "PILCO: A Model-Based and Data-Efficient Approach to Policy Search." In: *International Conference on Machine Learning*. ICML. 2011, (cit. on p. 97).

[79] Emir Demirovic, Peter J. Stuckey, James Bailey, Jeffrey Chan, Christopher Leckie, Kotagiri Ramamohanarao, and Tias Guns. "Predict+Optimise with Ranking Objectives: Exhaustively Learning Linear Functions". In: *International Joint Conferences on Artificial Intelligence Organization*. IJCAI. 2019, (cit. on p. 45).

[80] Stefan Depeweg, Jose-Miguel Hernandez-Lobato, Finale Doshi-Velez, and Steffen Udluft. "Decomposition of uncertainty in Bayesian deep learning for efficient and risk-sensitive learning". In: *International Conference on Machine Learning*. ICML. PMLR. 2018, (cit. on p. 97).

[81] Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft. "Learning and Policy Search in Stochastic Dynamical Systems with Bayesian Neural Networks". In: *International Conference on Learning Representations*. ICLR. 2017 (cit. on p. 97).

[82] Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau. "Learning Heuristics for the TSP by Policy Gradient". In: *Proc. of Intl. Conf. on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. 2018, (cit. on p. 46).

[83] Prafulla Dhariwal and Alexander Nichol. "Diffusion Models Beat GANs on Image Synthesis". In: NeurIPS (2021), (cit. on pp. 1, 2).

[84] Jared Di Carlo, Patrick M Wensing, Benjamin Katz, Gerardo Bledt, and Sangbae Kim. "Dynamic Locomotion in the MIT Cheetah 3 through Convex Model-Predictive Control". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IROS. IEEE. 2018, (cit. on p. 130).

[85] E. W. Dijkstra. "A Note on Two Problems in Connexion with Graphs". In: *Numer. Math.* 1 (1959), (cit. on pp. 45, 83).

[86] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. "Density Estimation using Real NVP". In: *International Conference on Learning Representations*. ICLR. 2017 (cit. on p. 4).

[87] Justin Domke. "Generic methods for optimization-based modeling". In: *Artificial Intelligence and Statistics*. 2012, (cit. on p. 47).

[88] Arnaud Doucet, Nando De Freitas, Neil James Gordon, et al. "Sequential Monte Carlo methods in practice". 2001 (cit. on p. 6).

[89] Jack Edmonds. "Paths, Trees, and Flowers". In: *Canadian Journal of Mathematics* (1965), (cit. on p. 188).

[90] Adam N. Elmachtoub and Paul Grigas. "Smart "Predict, then Optimize"". In: *ArXiv* 1 (2017). in press, arXiv preprint 1710.08005, (cit. on p. 45).

[91] Martin Engilberge, Louis Chevallier, Patrick Pérez, and Matthieu Cord. "SoDeep: a Sorting Deep net to learn ranking loss surrogates". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. CVPR'19. 2019, (cit. on pp. 62, 63, 67, 194, 196).

[92] Richard Evans and Edward Grefenstette. "Learning Explanatory Rules from Noisy Data". In: *Journal of Artificial Intelligence Research*. JAIR (2018) (cit. on p. 2).

[93] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman. "The Pascal Visual Object Classes (VOC) Challenge". In: *International Journal of Computer Vision* 2 (2010), (cit. on pp. 63, 75).

[94] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. "Diversity is All You Need: Learning Skills without a Reward Function". In: *International Conference on Learning Representations*. ICLR. 2019 (cit. on pp. 5, 109, 110, 113).

[95] Werner Fenchel. "Convex Cones, Sets, and Functions". 1949 (cit. on p. 5).

[96] Aaron Ferber, Bryan Wilder, Bistra Dilkina, and Milind Tambe. "MIPaaL: Mixed Integer Program as a Layer". In: *CoRR*. AAAI (2019), (cit. on p. 45).

[97] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *International Conference on Machine Learning*. ICML. 2017, (cit. on pp. 2, 23).

[98] Chuan-sheng Foo, Chuong B Do, and Andrew Y Ng. "Efficient multiple hyperparameter learning for log-linear models". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2008, (cit. on p. 47).

[99] Antonio Elia Forte, Paul Z. Hanakata, Lishuai Jin, Emilia Zari, Ahmad Zareei, Matheus C. Fernandes, Laura Sumner, Jonathan T. Alvarez, and Katia Bertoldi. "Inverse Design of Inflatable Soft Membranes Through Machine Learning". In: *Advanced Functional Materials* 16 (2022) (cit. on p. 137).

[100] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. "D4RL: Datasets for deep data-driven reinforcement learning". In: *arXiv preprint arXiv:2004.07219* (2020) (cit. on pp. 110, 118).

[101] Justin Fu, Sergey Levine, and Pieter Abbeel. "One-shot learning of manipulation skills with online dynamics adaptation and neural network priors". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IROS. IEEE. 2016, (cit. on p. 97).

[102] Kunihiko Fukushima. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological Cybernetics* 4 (1980), (cit. on p. 1).

[103] Yarin Gal, Rowan McAllister, and Carl Edward Rasmussen. "Improving PILCO with Bayesian neural network dynamics models". In: *International Conference on Machine Learning*. ICML. 2016, (cit. on p. 97).

[104] Javier Garcıa and Fernando Fernández. "A comprehensive survey on safe reinforcement learning". In: *Journal of Machine Learning Research* 1 (2015), (cit. on p. 97).

[105] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. "Exact Combinatorial Optimization with Graph Convolutional Neural Networks". In: NeurIPS (2019), (cit. on p. 46).

[106] Weifeng Ge. "Deep metric learning with hierarchical triplet loss". In: *European Conference on Computer Vision*. ECCV'18. 2018, (cit. on pp. 61, 63, 73–75).

[107] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. "Nas-fpn: Learning scalable feature pyramid architecture for object detection". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. CVPR'19. 2019, (cit. on p. 63).

[108] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. CVPR'14. 2014, (cit. on p. 63).

[109] Dongyoung Go, Tomasz Korbak, Germán Kruszewski, Jos Rozen, Nahyeon Ryu, and Marc Dymetman. "Aligning Language Models with Preferences through f-Divergence Minimization". In: NeurIPS (2023) (cit. on p. 23).

[110] Eran Goldman, Roei Herzig, Aviv Eisenschtat, Jacob Goldberger, and Tal Hassner. "Precise Detection in Densely Packed Scenes". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. CVPR'19. 2019, (cit. on p. 63).

[111] Rafael Gómez-Bombarelli, Jennifer N. Wei, David Duvenaud, JoséMiguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. "Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules". In: *ACS Central Science* 2 (2018), (cit. on p. 137).

[112] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. "Deep learning". 2016 (cit. on pp. 12, 25).

[113] "Google's OR-Tools". Google. 2019. URL: https://developers.google.com/optimization/ (cit. on p. 190).

[114] John C. Gower and Garmt B. Dijksterhuis. "Procrustes problems". Oxford Statistical Science Series. 2004 (cit. on p. 189).

[115] Anirudh Goyal, Riashat Islam, Daniel Strouse, Zafarali Ahmed, Hugo Larochelle, Matthew M. Botvinick, Yoshua Bengio, and Sergey Levine. "InfoBot: Transfer and Exploration via the Information Bottleneck". In: *International Conference on Learning Representations*. ICLR. 2019 (cit. on p. 109).

[116] Alexandros Graikos, Nikolay Malkin, Nebojsa Jojic, and Dimitris Samaras. "Diffusion Models as Plug-and-Play Priors". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2022 (cit. on p. 138).

[117] Alexandra Grancharova, Juš Kocijan, and Tor A Johansen. "Explicit stochastic predictive control of combustion plants based on Gaussian process models". In: *Automatica* 6 (2008), (cit. on p. 97).

[118] Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. "FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models". In: *International Conference on Learning Representations*. ICLR. 2019 (cit. on p. 4).

[119] Alex Graves, Greg Wayne, and Ivo Danihelka. "Neural Turing Machines". In: *arXiv* (2014) (cit. on p. 45).

[120] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. "Hybrid computing using a neural network with dynamic external memory". In: *Nature* 7626 (2016), (cit. on p. 45).

[121] Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. "Variational Intrinsic Control". In: *International Conference on Learning Representations*. ICLR. 2017 (cit. on p. 111).

[122] F. Grimminger, A. Meduri, M. Khadiv, J. Viereck, M. Wüthrich, M. Naveau, V. Berenz, S. Heim, F. Widmaier, T. Flayols, J. Fiene, A. Badri-Spröwitz, and L. Righetti. "An Open Torque-Controlled Modular Robot Architecture for Legged Locomotion Research". In: *IEEE Robotics and Automation Letters* 2 (2020), (cit. on pp. 103, 118, 215).

[123] Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. "Improved Training of Wasserstein GANs". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2017, (cit. on p. 118).

[124] LLC Gurobi Optimization. "Gurobi Optimizer Reference Manual". 2019 (cit. on p. 45).

[125] Jean Guyomarch. "Warcraft II open-source map editor". 2017 (cit. on pp. 51, 187).

[126] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International Conference on Machine Learning*. ICML. 2018 (cit. on pp. 34, 36).

[127] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. "Learning latent dynamics for planning from pixels". In: *International Conference on Machine Learning*. ICML. PMLR. 2019, (cit. on p. 97).

[128] Steven Hansen, Will Dabney, André Barreto, David Warde-Farley, Tom Van de Wiele, and Volodymyr Mnih. "Fast Task Inference with Variational Intrinsic Successor Features". In: *International Conference on Learning Representations*. ICLR. 2020 (cit. on p. 111).

[129] Godfrey Harold Hardy, John Edensor Littlewood, and George Pólya. "Inequalities". 1952, (cit. on p. 65).

[130] Ben Harwood, BG Kumar, Gustavo Carneiro, Ian Reid, Tom Drummond, et al. "Smart mining for deep metric learning". In: *IEEE International Conference on Computer Vision*. ICCV. 2017, (cit. on p. 74).

[131] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. "Mask R-CNN". In: *IEEE International Conference on Computer Vision*. ICCV. 2017, (cit. on p. 63).

[132] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2016, (cit. on pp. 51, 63, 71, 72, 76).

[133] Kun He, Fatih Cakir, Sarah Adel Bargal, and Stan Sclaroff. "Hashing as tie-aware learning to rank". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. CVPR'18. 2018, (cit. on pp. 61, 62, 67).

[134] Kun He, Yan Lu, and Stan Sclaroff. "Local descriptors optimized for average precision". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. CVPR'18. 2018, (cit. on p. 62).

[135] Wenhao He, Xu-Yao Zhang, Fei Yin, and Cheng-Lin Liu. "Deep direct regression for multi-oriented scene text detection". In: *IEEE International Conference on Computer Vision*. ICCV. 2017, (cit. on p. 72).

[136] Paul Henderson and Vittorio Ferrari. "End-to-end training of object class detectors for mean average precision". In: *Asian Conference on Computer Vision*. Springer. 2016, (cit. on pp. 61–63, 70).

[137] Jonathan Ho and Stefano Ermon. "Generative Adversarial Imitation Learning". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2016, (cit. on pp. 3, 130).

[138] Jonathan Ho, Ajay Jain, and Pieter Abbeel. "Denoising Diffusion Probabilistic Models". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2020, (cit. on pp. 138, 139).

[139] Jonathan Ho and Tim Salimans. "Classifier-free diffusion guidance". In: *NeurIPS Workshop on Deep Generative Models and Downstream Applications*. 2021 (cit. on pp. 6, 141).

[140] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural Computation* 8 (1997), (cit. on p. 2).

[141] Elad Hoffer and Nir Ailon. "Deep metric learning using triplet network". In: *International Workshop on Similarity-Based Pattern Recognition*. Springer. 2015, (cit. on p. 63).

[142] Stephen C. Hora. "Aleatory and epistemic uncertainty in probability elicitation with an example from hazardous waste management". In: *Reliability Engineering & System Safety*. Treatment of Aleatory and Epistemic Uncertainty 2 (1996), (cit. on p. 96).

[143] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. "Vime: Variational information maximizing exploration". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2016, (cit. on p. 109).

[144] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. "Tensorflow Object Detection API". Commit: 0ba83cf. 2017 (cit. on p. 62).

[145] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. "Imitation learning: A survey of learning methods". In: *ACM Computing Surveys (CSUR)* 2 (2017), (cit. on p. 95).

[146] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. "Learning agile and dynamic motor skills for legged robots". In: *Science Robotics* 26 (2019), (cit. on p. 130).

[147] Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. "Planning with Diffusion for Flexible Behavior Synthesis". In: *International Conference on Machine Learning*. ICML. 2022, (cit. on p. 138).

[148] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. "When to trust your model: Model-based policy optimization". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2019, (cit. on p. 80).

[149] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* 7873 (2021), (cit. on p. 1).

[150] Sanket Kamthe and Marc Deisenroth. "Data-efficient reinforcement learning with probabilistic model predictive control". In: *AAAI Conference on Artificial Intelligence*. AAAI. PMLR. 2018, (cit. on p. 97).

[151] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. "Elucidating the design space of diffusion-based generative models". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2022 (cit. on p. 138).

[152] Elias B. Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. "Learning to Branch in Mixed Integer Programming". In: *AAAI Conference on Artificial Intelligence*. AAAI. 2016, (cit. on p. 46).

[153] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. "Morel: Model-based Offline Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. NeurIPS (2020) (cit. on p. 3).

[154] Geon-Hyeong Kim, Seokin Seo, Jongmin Lee, Wonseok Jeon, HyeongJoo Hwang, Hongseok Yang, and Kee-Eung Kim. "DemoDICE: Offline Imitation Learning with Supplementary Imperfect Demonstrations". In: *International Conference on Learning Representations*. ICLR. 2022 (cit. on pp. 110, 112, 119).

[155] Jaekyeom Kim, Seohong Park, and Gunhee Kim. "Unsupervised Skill Discovery with Bottleneck Option Learning". In: *International Conference on Machine Learning*. ICML. 2021, (cit. on pp. 109, 111).

[156] Seung-Wook Kim, Hyong-Keun Kook, Jee-Young Sun, Mun-Cheon Kang, and Sung-Jea Ko. "Parallel feature pyramid network for object detection". In: *European Conference on Computer Vision*. ECCV'18. 2018, (cit. on p. 76).

[157] Wonsik Kim, Bhavya Goyal, Kunal Chawla, Jungmin Lee, and Keunjoo Kwon. "Attention-based ensemble for deep metric learning". In: *European Conference on Computer Vision*. ECCV'18. 2018, (cit. on pp. 71–75).

[158] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations*. ICLR. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. 2015 (cit. on pp. 11, 72, 139, 187).

[159] Armen Der Kiureghian and Ove Ditlevsen. "Aleatory or epistemic? Does it matter?" In: *Structural Safety*. Risk Acceptance and Risk Communication 2 (2009), (cit. on p. 96).

[160] A.S. Klyubin, D. Polani, and C.L. Nehaniv. "Empowerment: a universal agent-centric measure of control". In: *IEEE Congress on Evolutionary Computation*. 2005, (cit. on p. 109).

[161] Patrick Knöbelreiter, Christian Reinbacher, Alexander Shekhovtsov, and Thomas Pock. "End-To-End Training of Hybrid CNN-CRF Models for Stereo". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2017 (cit. on p. 45).

[162] Jens Kober, J Andrew Bagnell, and Jan Peters. "Reinforcement learning in robotics: A survey". In: *The International Journal of Robotics Research* 11 (2013), (cit. on p. 95).

[163] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. "Siamese neural networks for one-shot image recognition". In: *ICML deep learning workshop*. 2015 (cit. on p. 63).

[164] Juš Kocijan, Roderick Murray-Smith, Carl Edward Rasmussen, and Agathe Girard. "Gaussian process model based predictive control". In: *IEEE American Control Conference*. IEEE. 2004, (cit. on p. 97).

[165] Vladimir Kolmogorov. "Blossom V: a new implementation of a minimum cost perfect matching algorithm". In: *Mathematical Programming Computation* 1 (2009), (cit. on pp. 45, 188).

[166] Wouter Kool, Herke van Hoof, and Max Welling. "Attention, Learn to Solve Routing Problems!" In: *International Conference on Learning Representations*. ICLR. 2019 (cit. on p. 46).

[167] Tomasz Korbak, Hady Elsahar, Germán Kruszewski, and Marc Dymetman. "On Reinforcement Learning and Distribution Matching for Fine-tuning Language Models with No Catastrophic Forgetting". In: *Advances in Neural Information Processing Systems*. NeurIPS (2022), (cit. on p. 23).

[168] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. "Offline Reinforcement Learning with Implicit Q-Learning". In: *International Conference on Learning Representations*. ICLR. 2022 (cit. on p. 111).

[169] Ilya Kostrikov, Denis Yarats, and Rob Fergus. "Image Augmentation Is All You Need: Regularizing Deep Reinforcement Learning from Pixels". In: *arXiv (Cornell University)*. ICLR (2020) (cit. on p. 203).

[170] Brian Kulis et al. "Metric learning: A survey". In: *Foundations and Trends in Machine Learning* 4 (2013), (cit. on p. 63).

[171] Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. "RMA: Rapid Motor Adaptation for Legged Robots". In: *Robotics: Science and Systems*. 2021 (cit. on p. 130).

[172] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. "Conservative q-learning for offline reinforcement learning". In: NeurIPS (2020), (cit. on p. 111).

[173] Siddhant Kumar, Stephanie Tan, Li Zheng, and Dennis M. Kochmann. "Inverse-designed spinodoid metamaterials". In: *npj Computational Materials* 1 (2020), (cit. on p. 137).

[174] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. "Model-Ensemble Trust-Region Policy Optimization". In: *International Conference on Learning Representations*. ICLR. 2018 (cit. on p. 97).

[175] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. "Simple and Scalable Predictive Uncertainty Estimation Using Deep Ensembles". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2017, (cit. on p. 97).

[176] Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. "Reinforcement learning with augmented data". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2020 (cit. on p. 203).

[177] Marc T Law, Nicolas Thome, and Matthieu Cord. "Quadruplet-wise image similarity learning". In: *IEEE International Conference on Computer Vision*. ICCV. 2013, (cit. on p. 63).

[178] Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. "Handwritten Digit Recognition with a Back-Propagation Network". In: *Advances in Neural Information Processing Systems*. NeurIPS. 1989, (cit. on p. 1).

[179] Jongmin Lee, Wonseok Jeon, Byungjun Lee, Joelle Pineau, and Kee-Eung Kim. "Optidice: Offline policy optimization via stationary distribution correction estimation". In: *International Conference on Machine Learning*. ICML. PMLR. 2021, (cit. on p. 112).

[180] Jongmin Lee, Cosmin Paduraru, Daniel J Mankowitz, Nicolas Heess, Doina Precup, Kee-Eung Kim, and Arthur Guez. "COptiDICE: Offline Constrained Reinforcement Learning via Stationary Distribution Correction Estimation". In: *International Conference on Learning Representations*. ICLR. 2022 (cit. on p. 112).

[181] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. "Learning quadrupedal locomotion over challenging terrain". In: *Science robotics* 47 (2020), (cit. on p. 130).

[182] Keuntaek Lee, Gabriel Nakajima An, Viacheslav Zakharov, and Evangelos A. Theodorou. "Perceptual Attention-based Predictive Control". In: *Conference on Robot Learning*. Proceedings of Machine Learning Research. 2020, (cit. on p. 97).

[183] Seunghyun Lee, Younggyo Seo, Kimin Lee, Pieter Abbeel, and Jinwoo Shin. "Offline-to-Online Reinforcement Learning via Balanced Replay and Pessimistic Q-Ensemble". In: *Conference on Robot Learning*. CoRL. 2022 (cit. on p. 3).

[184] Ian Lenz, Ross A Knepper, and Ashutosh Saxena. "DeepMPC: Learning deep latent features for model predictive control." In: *Robotics: Science and Systems*. Rome, Italy. 2015 (cit. on p. 97).

[185] David A Levin and Yuval Peres. "Markov Chains and Mixing Times". 2017 (cit. on p. 6).

[186] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. "Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems". In: *CoRR* (2020) (cit. on pp. 110, 111).

[187] Pierre-Alexandre Léziart, Thomas Flayols, Felix Grimminger, Nicolas Mansard, and Philippe Souères. "Implementation of a reactive walking controller for the new open-hardware quadruped solo-12". In: *IEEE International Conference on Robotics and Automation*. ICRA. IEEE. 2021, (cit. on p. 110).

[188] Chenhao Li, Marin Vlastelica, Sebastian Blaes, Jonas Frey, Felix Grimminger, and Georg Martius. "Learning Agile Skills via Adversarial Imitation of Rough Partial Demonstrations". In: *Conference on Robot Learning*. Proceedings of Machine Learning Research. 2023, (cit. on pp. 3, 7, 129).

[189] Weiwei Li and Emanuel Todorov. "Iterative linear quadratic regulator design for nonlinear biological movement systems". In: *International Conference on Informatics in Control, Automation and Robotics*. SciTePress. 2004, (cit. on pp. 37, 38).

[190] Yujia Li, Richard Zemel, Marc Brockschmidt, and Daniel Tarlow. "Gated Graph Sequence Neural Networks". In: *International Conference on Learning Representations*. ICLR. 2016 (cit. on p. 45).

[191] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. "Combinatorial Optimization with Graph Convolutional Networks and Guided Tree Search". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2018, (cit. on p. 46).

[192] Tsung-Yi Lin, Priyal Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. "Focal Loss for Dense Object Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. ICCV (2018) (cit. on p. 63).

[193] Yi Lin, Yoonkyung Lee, and Grace Wahba. "Support Vector Machines for Classification in Nonstandard Situations". In: *Machine Learning* 1 (2002), (cit. on p. 61).

[194] Qiang Liu, Lihong Li, Ziyang Tang, and Dengyong Zhou. "Breaking the Curse of Horizon: Infinite-Horizon Off-Policy Estimation". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2018, (cit. on p. 111).

[195] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. "Ssd: Single shot multibox detector". In: *European Conference on Computer Vision*. Springer. 2016, (cit. on p. 63).

[196] Ziwei Liu, Xiaoxiao Li, Ping Luo, Chen-Change Loy, and Xiaoou Tang. "Semantic Image Segmentation via Deep Parsing Network". In: *IEEE International Conference on Computer Vision*. ICCV. 2015, (cit. on p. 45).

[197] Ziwei Liu, Ping Luo, Shi Qiu, Xiaogang Wang, and Xiaoou Tang. "DeepFashion: Powering Robust Clothes Recognition and Retrieval with Rich Annotations". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2016, (cit. on pp. 71, 75).

[198] Lennart Ljung. "System Identification: Theory for the User". 1999 (cit. on p. 4).

[199] Cong Lu, Philip J. Ball, and Jack Parker-Holder. "Synthetic Experience Replay". In: NeurIPS (2023) (cit. on p. 35).

[200] David G Luenberger, Yinyu Ye, et al. "Linear and Nonlinear Programming". 1984 (cit. on p. 15).

[201] Yecheng Jason Ma, Andrew Shen, Dinesh Jayaraman, and Osbert Bastani. "Versatile Offline Imitation from Observations and Examples via Regularized State-Occupancy Matching". In: *International Conference on Machine Learning*. ICML. 2022, (cit. on pp. 110, 113, 115, 119, 120, 123, 222, 223, 225).

[202] David John Cameron MacKay. "Information Theory, Inference, and Learning Algorithms". 2004, (cit. on p. 25).

[203] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. "Isaac gym: High performance gpu-based physics simulation for robot learning". In: NeurIPS (2021) (cit. on pp. 118, 223).

[204] Jaynta Mandi, Emir Demirovic, Peter J. Stuckey, and Tias Guns. "Smart Predict-and-Optimize for Hard Combinatorial Optimization Problems". In: *CoRR* 02 (2019), (cit. on p. 45).

[205] Hugues Marchand, Alexander Martin, Robert Weismantel, and Laurence Wolsey. "Cutting Planes in Integer and Mixed Integer Programming". In: *Discrete Applied Mathematics* 1-3 (2002), (cit. on p. 52).

[206] Dmitrii Marin, Meng Tang, Ismail Ben Ayed, and Yuri Boykov. "Beyond Gradient Descent for Regularized Segmentation Losses". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019 (cit. on p. 45).

[207] Francisco Massa and Ross Girshick. "maskrcnn-benchmark: Fast, modular reference implementation of Instance Segmentation and Object Detection algorithms in PyTorch". Commit: f027259. 2018 (cit. on p. 62).

[208] Brian McFee and Gert R Lanckriet. "Metric Learning to Rank". In: *International Conference on Machine Learning*. ICML. 2010, (cit. on p. 62).

[209] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Großberger. "UMAP: Uniform Manifold Approximation and Projection". In: *Journal of Open Source Software* 29 (2018), (cit. on p. 120).

[210] "Methods of reducing sample size in Monte Carlo computations". In: *Journal of the Operations Research Society of America* 5 (1953), (cit. on p. 35).

[211] Oliver Mihatsch and Ralph Neuneier. "Risk-sensitive reinforcement learning". In: *Machine learning* 2 (2002), (cit. on p. 97).

[212] Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. "Learning robust perceptive locomotion for quadrupedal robots in the wild". In: *Science Robotics* 62 (2022), (cit. on p. 130).

[213] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. "Asynchronous methods for deep reinforcement learning". In: *International Conference on Machine Learning*. ICML. PMLR. 2016, (cit. on p. 34).

[214] Shakir Mohamed and Danilo Jimenez Rezende. "Variational Information Maximisation for Intrinsically Motivated Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2015, (cit. on p. 109).

[215] Pritish Mohapatra, CV Jawahar, and M Pawan Kumar. "Efficient Optimization for Average Precision SVM". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2014, (cit. on p. 62).

[216] Pritish Mohapatra, Michal Rolínek, C.V. Jawahar, Vladimir Kolmogorov, and M. Pawan Kumar. "Efficient Optimization for Rank-Based Loss Functions". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. CVPR'18. 2018, (cit. on pp. 62, 66, 67).

[217] Manfred Morari and Jay H Lee. "Model predictive control: past, present and future". In: *Computers & Chemical Engineering* 4-5 (1999), (cit. on pp. 4, 97).

[218] Ali Mousavi, Lihong Li, Qiang Liu, and Denny Zhou. "Black-box Off-policy Estimation for Infinite-Horizon Reinforcement Learning". In: *International Conference on Learning Representations*. ICLR. 2020 (cit. on p. 111).

[219] Yair Movshovitz-Attias, Alexander Toshev, Thomas K Leung, Sergey Ioffe, and Saurabh Singh. "No fuss distance metric learning using proxies". In: *IEEE International Conference on Computer Vision*. ICCV. 2017, (cit. on pp. 63, 71, 73, 74).

[220] Ofir Nachum, Yinlam Chow, Bo Dai, and Lihong Li. "Dualdice: Behavior-agnostic estimation of discounted stationary distribution corrections". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2019, (cit. on p. 112).

[221] Ofir Nachum and Bo Dai. "Reinforcement learning via fenchel-rockafellar duality". 2020 (cit. on pp. 22, 23, 32, 110, 113, 115, 147, 223).

[222] Ofir Nachum, Bo Dai, Ilya Kostrikov, Yinlam Chow, Lihong Li, and Dale Schuurmans. "AlgaeDICE: Policy Gradient from Arbitrary Experience". 2019 (cit. on p. 112).

[223] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. "Data-efficient hierarchical reinforcement learning". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2018, (cit. on p. 86).

[224] Ashvin Nair, Murtaza Dalal, Abhishek Gupta, and Sergey Levine. "AWAC: Accelerating Online Reinforcement Learning with Offline Datasets". In: *CoRR* (2020) (cit. on p. 111).

[225] MohammadReza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takac. "Reinforcement Learning for Solving the Vehicle Routing Problem". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2018, (cit. on p. 46).

[226] George L. Nemhauser and Laurence A. Wolsey. "Integer and Combinatorial Optimization". In: *Wiley interscience series in discrete mathematics and optimization*. 1988, (cit. on pp. 16, 18).

[227] Gerhard Neumann and Jan Peters. "Fitted Q-iteration by Advantage Weighted Regression". In: *Advances in Neural Information Processing Systems*. NeurIPS (2008) (cit. on p. 23).

[228] Duy Nguyen-Tuong, Jan Peters, and Matthias Seeger. "Local gaussian process regression for real time online model learning and control". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2008, (cit. on p. 97).

[229] Vlad Niculae, Andre Martins, Mathieu Blondel, and Claire Cardie. "SparseMAP: Differentiable Sparse Structured Inference". In: *International Conference on Machine Learning*. ICML. 2018, (cit. on p. 45).

[230] Jorge Nocedal and Stephen J. Wright. "Numerical Optimization". 2018, (cit. on p. 21).

[231] John P O'Doherty, Peter Dayan, Karl Friston, Hugo Critchley, and Raymond J Dolan. "Temporal Difference Models and Reward-Related Learning in the Human Brain". In: *Neuron* (2003) (cit. on p. 26).

[232] Hyun Oh Song, Stefanie Jegelka, Vivek Rathod, and Kevin Murphy. "Deep metric learning via facility location". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. CVPR'17. 2017, (cit. on pp. 73, 74).

[233] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. "Deep metric learning via lifted structured feature embedding". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. CVPR'16. 2016, (cit. on pp. 63, 73, 74).

[234] H. Freyja Ólafsdóttir, Caswell Barry, Aman B. Saleem, Demis Hassabis, and Hugo J. Spiers. "Hippocampal place cells construct reward related sequences through unexplored space". In: *eLife* (2015) (cit. on p. 35).

[235] Michael Opitz, Georg Waltner, Horst Possegger, and Horst Bischof. "Deep metric learning with BIER: Boosting independent embeddings robustly". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2018), (cit. on pp. 73–75).

[236] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. "Deep Exploration via Bootstrapped DQN". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2016, (cit. on p. 97).

[237] George Papamakarios, Theo Pavlakou, and Iain Murray. "Masked Autoregressive Flow for Density Estimation". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2017, (cit. on p. 4).

[238] Seohong Park, Jongwook Choi, Jaekyeom Kim, Honglak Lee, and Gunhee Kim. "Lipschitz-constrained Unsupervised Skill Discovery". In: *International Conference on Learning Representations*. ICLR. 2022 (cit. on p. 111).

[239] Seohong Park, Kimin Lee, Youngwoon Lee, and Pieter Abbeel. "Controllability-Aware Unsupervised Skill Discovery". In: *CoRR*. ICML (2023), (cit. on p. 111).

[240] Seohong Park and Sergey Levine. "Predictable MDP Abstraction for Unsupervised Model-Based RL". In: *International Conference on Machine Learning*. ICML. 2023, (cit. on pp. 109, 111).

[241] Despoina Paschalidou, Ali Osman Ulusoy, Carolin Schmitt, Luc Gool, and Andreas Geiger. "RayNet: Learning Volumetric 3D Reconstruction with Ray Potentials". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, (cit. on p. 45).

[242] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. "Curiosity-Driven Exploration by Self-Supervised Prediction". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2017, (cit. on p. 36).

[243] Ivan Petrovich Pavlov. "Conditioned Reflexes: An Investigation of the Physiological Activity of the Cerebral Cortex". 1927 (cit. on p. 26).

[244] Jan Peters and Stefan Schaal. "Reinforcement Learning by Reward-Weighted Regression for Operational Space Control". In: *International Conference on Machine Learning*. ICML. 2007 (cit. on p. 23).

[245] Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. "Elements of Causal Inference". 2017 (cit. on p. 2).

[246] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter Battaglia. "Learning Mesh-Based Simulation with Graph Networks". In: *International Conference on Learning Representations*. ICLR. 2021 (cit. on p. 139).

[247] Cristina Pinneri, Shambhuraj Sawant, Sebastian Blaes, Jan Achterhold, Joerg Stueckler, Michal Rolínek, and Georg Martius. "Sample-efficient Cross-Entropy Method for Real-time Planning". In: *Conference on Robot Learning*. 2020 (cit. on pp. 4, 40, 97, 98, 104, 206, 210).

[248] Leonid Pishchulin, Eldar Insafutdinov, Siyu Tang, Björn Andres, Mykhaylo Andriluka, Peter Gehler, and Bernt Schiele. "DeepCut: Joint Subset Partition and Labeling for Multi Person Pose Estimation". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2016, (cit. on p. 45).

[249] Athanasios S Polydoros and Lazaros Nalpantidis. "Survey of model-based reinforcement learning: Applications on robotics". In: *Journal of Intelligent & Robotic Systems* 2 (2017), (cit. on p. 95).

[250] Rafael Figueiredo Prudencio, Marcos R. O. A. Maximo, and Esther Luna Colombini. "A Survey on Offline Reinforcement Learning: Taxonomy, Review, and Open Problems". In: *CoRR* (2022) (cit. on pp. 110, 111).

[251] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". 2015 (cit. on p. 189).

[252] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. "Direct Preference Optimization: Your Language Model is Secretly a Reward Model". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2023 (cit. on pp. 3, 33).

[253] Prasad Raghavendra. "Optimal Algorithms and Inapproximability Results for Every CSP?" In: *ACM Symposium on Theory of Computing*. STOC '08. 2008, (cit. on pp. 3, 44).

[254] Marc Raibert, Kevin Blankespoor, Gabriel Nelson, and Rob Playter. "Bigdog, the rough-terrain quadruped robot". In: *IFAC Proceedings Volumes* 2 (2008), (cit. on p. 130).

[255] Roberta Raileanu, Max Goldstein, Denis Yarats, Ilya Kostrikov, and Rob Fergus. "Automatic Data Augmentation for Generalization in Reinforcement Learning". In: NeurIPS (2021), (cit. on pp. 86, 89, 200–203).

[256] Aravind Rajeswaran, Chelsea Finn, Sham M Kakade, and Sergey Levine. "Meta-learning with implicit gradients". In: NeurIPS (2019), (cit. on pp. 2, 23).

[257] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. "Zero-shot Text-to-Image Generation". In: *International Conference on Machine Learning*. ICML. 2021 (cit. on p. 1).

[258] Yongming Rao, Dahua Lin, Jiwen Lu, and Jie Zhou. "Learning Globally Optimized Object Detector via Policy Gradient". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. CVPR'18. 2018, (cit. on p. 63).

[259] Carl Rasmussen and Malte Kuss. "Gaussian Processes in Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2003, (cit. on p. 97).

[260] CE. Rasmussen and CKI. Williams. "Gaussian Processes for Machine Learning". Adaptive Computation and Machine Learning. 2006, (cit. on p. 97).

[261] James Blake Rawlings, David Q Mayne, Moritz Diehl, et al. "Model predictive control: theory, computation, and design". 2017 (cit. on p. 36).

[262] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. "You Only Look Once: Unified, Real-Time Object Detection". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. CVPR'16. 2016 (cit. on p. 63).

[263] Joseph Redmon and Ali Farhadi. "YOLOv3: An Incremental Improvement". 2018 (cit. on p. 63).

[264] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster r-cnn: Towards real-time object detection with region proposal networks". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2015, (cit. on pp. 63, 75).

[265] Alexander Reske, Jan Carius, Yuntao Ma, Farbod Farshidian, and Marco Hutter. "Imitation learning from mpc for quadrupedal multi-gait control". In: *International Conference on Robotics and Automation*. ICRA. IEEE. 2021, (cit. on p. 38).

[266] Jerome Revaud, Jon Almazan, Rafael Sampaio de Rezende, and Cesar Roberto de Souza. "Learning with Average Precision: Training Image Retrieval with a Listwise Loss". In: *IEEE International Conference on Computer Vision*. ICCV. 2019, (cit. on pp. 61–63, 68).

[267] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. "Generalized intersection over union: A metric and a loss for bounding box regression". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. CVPR'19. 2019, (cit. on p. 63).

[268] Danilo Rezende and Shakir Mohamed. "Variational Inference with Normalizing Flows". In: *International Conference on Machine Learning*. ICML. PMLR. 2015 (cit. on p. 4).

[269] "RoboHive – A Unified Framework for Robot Learning". 2020. URL: https://sites.google.com/view/robohive (cit. on p. 110).

[270] R. Tyrrell Rockafellar. "Fenchel Duality Theorem and Applications". In: *Mathematica Scandinavica* (1966), (cit. on p. 5).

[271] Michal Rolínek, Vít Musil, Anselm Paulus, Marin Vlastelica, Claudio Michaelis, and Georg Martius. "Optimizing Rank-Based Metrics With Blackbox Differentiation". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, (cit. on pp. 7, 61).

[272] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*. Springer. 2015, (cit. on p. 2).

[273] Karsten Roth and Biagio Brattoli. "Easily Extendable Basic Deep Metric Learning Pipeline". Commit: 59d48f9. 2019 (cit. on pp. 62, 72, 74).

[274] Reuven Rubinstein. "The cross-entropy method for combinatorial and continuous optimization". In: *Methodology and Computing in Applied Probability* (1999), (cit. on pp. 39, 97, 139).

[275] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. "A metric for distributions with applications to image databases". In: *IEEE International Conference on Computer Vision*. ICCV. IEEE. 1998, (cit. on p. 130).

[276] Sebastian Sager. "Numerical Methods for Mixed-Integer Optimal Control Problems". 2005 (cit. on p. 38).

[277] Subham Sekhar Sahoo, Anselm Paulus, Marin Vlastelica, Vít Musil, Volodymyr Kuleshov, and Georg Martius. "Backpropagation through Combinatorial Algorithms: Identity with Projection Works". In: *International Conference on Learning Representations*. ICLR. 2022 (cit. on p. 55).

[278] Kegan GG Samuel and Marshall F Tappen. "Learning optimized map estimates in continuously-valued mrf models". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2009, (cit. on p. 47).

[279] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. "Learning to simulate complex physics with graph networks". In: *International Conference on Machine Learning*. ICML. 2020, (cit. on p. 139).

[280] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. "The Graph Neural Network Model". In: *Trans. Neur. Netw.* 1 (2009), (cit. on p. 45).

[281] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. "Universal Value Function Approximators". In: *International Conference on Machine Learning*. ICML. 2015, (cit. on p. 81).

[282] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. "Prioritized Experience Replay". 2015 (cit. on p. 35).

[283] Uwe Schmidt and Stefan Roth. "Shrinkage fields for effective image restoration". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2014, (cit. on p. 47).

[284] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. "Mastering Atari, Go, chess and shogi by planning with a learned model". In: *Nature* 7839 (7839 2020), (cit. on p. 80).

[285] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. "Mastering atari, go, chess and shogi by planning with a learned model". In: *Nature* 7839 (2020), (cit. on p. 95).

[286] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "Facenet: A unified embedding for face recognition and clustering". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. CVPR'15. 2015, (cit. on pp. 63, 68).

[287] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. "Trust region policy optimization". In: *International Conference on Machine Learning*. ICML. 2015, (cit. on p. 34).

[288] Samuel Schulter, Paul Vernaza, Wongun Choi, and Manmohan Krishna Chandraker. "Deep Network Flow for Multi-object Tracking". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2017, (cit. on p. 45).

[289] Wolfram Schultz, Peter Dayan, and P Read Montague. "A Neural Substrate of Prediction and Reward". In: *Science* 5306 (1997), (cit. on pp. 26, 30).

[290] Daniel Selsam and Nikolaj Bjørner. "Guiding High-Performance SAT Solvers with Unsat-Core Predictions". In: *Theory and Applications of Satisfiability Testing*. 2019, (cit. on p. 46).

[291] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. "Learning a SAT Solver from Single-Bit Supervision". In: *International Conference on Learning Representations*. ICLR. 2019 (cit. on p. 46).

[292] M Shahid Shaikh and Peter E Caines. "On the Hybrid Optimal Control Problem: Theory and Algorithms". In: *IEEE Transactions on Automatic Control* (2007) (cit. on p. 38).

[293] Lloyd S Shapley. "Stochastic Games". In: *Proceedings of the National Academy of Sciences* () (cit. on p. 30).

[294] Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. "Dynamics-Aware Unsupervised Discovery of Skills". In: *International Conference on Learning Representations*. ICLR. 2020 (cit. on pp. 111, 113).

[295] Tianhao Shen, Renren Jin, Yufei Huang, Chuang Liu, Weilong Dong, Zishan Guo, Xinwei Wu, Yan Liu, and Deyi Xiong. "Large Language Model Alignment: A Survey". 2023 (cit. on p. 3).

[296] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Artfhur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 6419 (2018), (cit. on p. 1).

[297] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm". In: *arXiv* (2017) (cit. on p. 1).

[298] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. "Mastering the game of go without human knowledge". In: *Nature* 7676 (2017), (cit. on p. 1).

[299] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *International Conference on Learning Representations*. ICLR. 2015 (cit. on p. 72).

[300] Annabelle C. Singer and Loren M. Frank. "Rewarded Outcomes Enhance Reactivation of Experience in the Hippocampus". In: *Neuron* 6 (2009), (cit. on p. 35).

[301] Kihyuk Sohn. "Improved deep metric learning with multi-class n-pair loss objective". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2016, (cit. on pp. 73, 74).

[302] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. "Deep Metric Learning via Lifted Structured Feature Embedding". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2016 (cit. on pp. 71, 72).

[303] Jiaming Song, Chenlin Meng, and Stefano Ermon. "Denoising Diffusion Implicit Models". In: *International Conference on Learning Representations*. ICLR. 2020 (cit. on p. 138).

[304] Jie Song, Bjoern Andres, Michael Black, Otmar Hilliges, and Siyu Tang. "End-to-end Learning for Graph Decomposition". In: *arXiv* (2018), (cit. on p. 45).

[305] Yang Song, Alexander Schwing, Richard, and Raquel Urtasun. "Training Deep Neural Networks via Direct Loss Minimization". In: *International Conference on Machine Learning*. ICML. 2016, (cit. on pp. 62, 63).

[306] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. "Score-based generative modeling through stochastic differential equations". In: *International Conference on Learning Representations*. ICLR. 2020 (cit. on pp. 2, 138–140).

[307] Aravind Srinivas, Michael Laskin, and Pieter Abbeel. "Curl: Contrastive unsupervised representations for reinforcement learning". In: *arXiv preprint arXiv:2004.04136* (2020) (cit. on p. 203).

[308] Adam Stooke, Joshua Achiam, and Pieter Abbeel. "Responsive Safety in Reinforcement Learning by PID Lagrangian Methods". In: *International Conference on Machine Learning*. ICML. 2020, (cit. on p. 117).

[309] DJ Strouse, Kate Baumli, David Warde-Farley, Volodymyr Mnih, and Steven Stenberg Hansen. "Learning more skills through optimistic exploration". In: *International Conference on Learning Representations*. ICLR. 2022 (cit. on pp. 5, 109, 110, 113, 117).

[310] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. "End-To-End Memory Networks". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2015, (cit. on p. 45).

[311] Richard S Sutton. "Dyna, an integrated architecture for learning, planning, and reacting". In: *ACM Sigart Bulletin* 4 (1991), (cit. on p. 80).

[312] Richard S Sutton and Andrew G Barto. "Reinforcement learning: An introduction". Second. 2018, (cit. on pp. 3, 30, 31, 34, 135).

[313] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. CVPR'15. 2015 (cit. on p. 72).

[314] Csaba Szepesvári. "Constrained MDPs and the reward hypothesis". 2020 (cit. on p. 110).

[315] Michael Taylor, John Guiver, Stephen Robertson, and Tom Minka. "Softrank: optimizing non-smooth rank metrics". In: *International Conference on Web Search and Data Mining*. ACM. 2008, (cit. on pp. 62, 63).

[316] Chen Tessler, Daniel J. Mankowitz, and Shie Mannor. "Reward Constrained Policy Optimization". In: *International Conference on Learning Representations*. ICLR. 2019 (cit. on p. 113).

[317] Johan Thapper and Stanislav Živný. "The Limits of SDP Relaxations for General-valued CSPs". In: *ACM/IEEE Symposium on Logic in Computer Science*. LICS '17. 2017, (cit. on p. 44).

[318] Naftali Tishby, Fernando C. Pereira, and William Bialek. "The information bottleneck method". In: *Allerton Conference on Communication, Control and Computing*. 1999, (cit. on p. 109).

[319] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. "Domain randomization for transferring deep neural networks from simulation to the real world". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IROS. IEEE. 2017, (cit. on p. 203).

[320] Emanuel Todorov, Tom Erez, and Yuval Tassa. "Mujoco: A Physics Engine for Model-Based Control". In: *International Conference on Intelligent Robots and Systems*. IROS. IEEE. 2012, (cit. on p. 210).

[321] Jonathan J Tompson, Arjun Jain, Yann LeCun, and Christoph Bregler. "Joint Training of a Convolutional Network and a Graphical Model for Human Pose Estimation". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2014, (cit. on p. 45).

[322] Sebastian Tschiatschek, Aytunc Sahin, and Andreas Krause. "Differentiable Submodular Maximization". In: *International Joint Conferences on Artificial Intelligence Organization*. IJCAI. 2018, (cit. on p. 45).

[323] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. "Large Margin Methods for Structured and Interdependent Output Variables". In: *Journal of Machine Learning Research* 50 (2005), (cit. on p. 45).

[324] Evgeniya Ustinova and Victor Lempitsky. "Learning deep embeddings with histogram loss". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2016, (cit. on pp. 73, 74).

[325] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need". In: NeurIPS (2017), (cit. on p. 1).

[326] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. "Graph Attention Networks". In: *International Conference on Learning Representations*. ICLR. 2018 (cit. on p. 1).

[327] Petar Velickovic, Rex Ying, Matilde Padovano, Raia Hadsell, and Charles Blundell. "Neural Execution of Graph Algorithms". In: *International Conference on Learning Representations*. ICLR. 2020 (cit. on p. 1).

[328] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michael Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature* 7782 (2019), (cit. on p. 1).

[329] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. "Pointer Networks". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2015, (cit. on p. 45).

[330] Marin Vlastelica, Sebastian Blaes, Cristina Pinneri, and Georg Martius. "Risk-Averse Zero-Order Trajectory Optimization". In: *Conference on Robot Learning*. 2021, (cit. on pp. 7, 36, 40, 95).

[331] Marin Vlastelica, Patrick Ernst, and Gyuri Szarvas. "Taming Continuous Posteriors for Latent Variational Dialogue Policies". In: *AAAI Conference on Artificial Intelligence*. AAAI. 2023 (cit. on p. 3).

[332] Marin Vlastelica, Pavel Kolev, Jin Cheng, and Georg Martius. "Offline Diversity Maximization under Imitation Constraints". In: *Reinforcement Learning Conference*. 2024 (cit. on pp. 3, 7, 109).

[333] Marin Vlastelica, Tatiana López-Guevara, Kelsey Allen, Peter Battaglia, Arnaud Doucet, and Kimberley Stachenfeld. "Diffusion Generative Inverse Design". In: *International Conference on Machine Learning, Structured Probabilistic Inference & Generative Modeling*. ICML SPGiM. 2023 (cit. on pp. 7, 137).

[334] Marin Vlastelica, Anselm Paulus, Vít Musil, Georg Martius, and Michal Rolínek. "Differentiation of Blackbox Combinatorial Solvers". In: *International Conference on Learning Representations*. ICLR. 2020 (cit. on pp. 6, 43, 48, 49, 77, 194–196).

[335] Marin Vlastelica, Michal Rolínek, and Georg Martius. "Neuro-algorithmic Policies enable Fast Combinatorial Generalization". In: *International Conference on Machine Learning*. ICML. PMLR. 2021, (cit. on pp. 7, 79).

[336] Homer Walke, Kevin Black, Abraham Lee, Moo Jin Kim, Max Du, Chongyi Zheng, Tony Zhao, Philippe Hansen-Estruch, Quan Vuong, Andre He, Vivek Myers, Kuan Fang, Chelsea Finn, and Sergey Levine. "BridgeData V2: A Dataset for Robot Learning at Scale". In: *Conference on Robot Learning*. 2023, (cit. on p. 110).

[337] Jian Wang, Feng Zhou, Shilei Wen, Xiao Liu, and Yuanqing Lin. "Deep metric learning with angular loss". In: *IEEE International Conference on Computer Vision*. ICCV. 2017, (cit. on pp. 73, 74).

[338] Tingwu Wang and Jimmy Ba. "Exploring Model-based Planning with Policy Networks". In: *International Conference on Learning Representations*. ICLR. 2020 (cit. on p. 96).

[339] Po-Wei Wang, Priya L. Donti, Bryan Wilder, and Zico Kolter. "SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver". In: *arXiv (Cornell University)*. ICML (2019), (cit. on pp. 2, 45).

[340] Ziyu Wang, Alexander Novikov, Konrad Zolna, Josh S Merel, Jost Tobias Springenberg, Scott E Reed, Bobak Shahriari, Noah Siegel, Caglar Gulcehre, Nicolas Heess, et al. "Critic regularized regression". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2020, (cit. on p. 111).

[341] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* (1992), (cit. on p. 31).

[342] Peter Welinder, Steve Branson, Takeshi Mita, Catherine Wah, Florian Schroff, Serge Belongie, and Pietro Perona. "Caltech-UCSD Birds 200". Tech. rep. California Institute of Technology, 2010 (cit. on pp. 71, 74).

[343] Min Wen and Ufuk Topcu. "Constrained Cross-Entropy Method for Safe Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2018, (cit. on p. 103).

[344] Bryan Wilder, Eric Ewing, Bistra Dilkina, and Milind Tambe. "End to end learning and optimization on graphs". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2019, (cit. on p. 46).

[345] Grady Williams, Andrew Aldrich, and Evangelos Theodorou. "Model Predictive Path Integral Control using Covariance Variable Importance Sampling". 2015 (cit. on pp. 3, 4, 96).

[346] Grady Williams, Andrew Aldrich, and Evangelos A Theodorou. "Model Predictive Path Integral Control: From Theory to Parallel Computation". In: *Journal of Guidance, Control, and Dynamics* (2017) (cit. on p. 3).

[347] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M Rehg, Byron Boots, and Evangelos A Theodorou. "Information theoretic MPC for model-based reinforcement learning". In: *IEEE International Conference on Robotics and Automation*. ICRA. IEEE. 2017, (cit. on pp. 4, 97, 98).

[348] Chao-Yuan Wu, R Manmatha, Alexander J Smola, and Philipp Krahenbuhl. "Sampling matters in deep embedding learning". In: *IEEE International Conference on Computer Vision*. ICCV. 2017, (cit. on pp. 63, 73, 74).

[349] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. "Detectron2". Commit: dd5926a. 2019 (cit. on p. 62).

[350] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. "Aggregated residual transformations for deep neural networks". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. CVPR'17. 2017, (cit. on p. 76).

[351] Haoran Xu, Li Jiang, Jianxiong Li, Zhuoran Yang, Zhaoran Wang, Wai Kin Victor Chan, and Xianyuan Zhan. "Offline RL with No OOD Actions: In-Sample Learning via Implicit Value Regularization". In: *International Conference on Learning Representations*. ICLR. 2023 (cit. on p. 111).

[352] Haoran Xu, Xianyuan Zhan, Jianxiong Li, and Honglei Yin. "Offline Reinforcement Learning with Soft Behavior Regularization". In: *CoRR* (2021) (cit. on p. 112).

[353] Hong Xuan, Richard Souvenir, and Robert Pless. "Deep randomized ensembles for metric learning". In: *European Conference on Computer Vision*. ECCV'18. 2018, (cit. on p. 75).

[354] Yuxiang Yang, Ken Caluwaerts, Atil Iscen, Tingnan Zhang, Jie Tan, and Vikas Sindhwani. "Data Efficient Reinforcement Learning for Legged Robots". In: *Conference on Robot Learning* (2019) (cit. on p. 95).

[355] Mang Ye, Andy J. Ma, Liang Zheng, Jiawei Li, and Pong C. Yuen. "Dynamic Label Graph Matching for Unsupervised Video Re-Identification". In: *IEEE International Conference on Computer Vision*. ICCV. (Cit. on p. 45).

[356] Yuhui Yuan, Kuiyuan Yang, and Chao Zhang. "Hard-aware deeply cascaded embedding". In: *IEEE International Conference on Computer Vision*. ICCV. 2017, (cit. on pp. 73–75).

[357] Yisong Yue, Thomas Finley, Filip Radlinski, and Thorsten Joachims. "A support vector method for optimizing average precision". In: *ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM. 2007, (cit. on pp. 61, 62, 67).

[358] Tom Zahavy, Brendan O'Donoghue, Guillaume Desjardins, and Satinder Singh. "Reward is enough for convex MDPs". In: *Advances in Neural Information Processing Systems*. NeurIPS. 2021, (cit. on p. 134).

[359] Tom Zahavy, Yannick Schroecker, Feryal M. P. Behbahani, Kate Baumli, Sebastian Flennerhag, Shaobo Hou, and Satinder Singh. "Discovering Policies with DOMiNO: Diversity Optimization Maintaining Near Optimality". In: *CoRR* (2022) (cit. on pp. 5, 6, 109, 111, 117, 133, 135).

[360] Matthew D. Zeiler. "ADADELTA: An Adaptive Learning Rate Method". In: *ArXiv* (2012) (cit. on p. 11).

[361] Albert Zhan, Philip Zhao, Lerrel Pinto, Pieter Abbeel, and Michael Laskin. "A Framework for Efficient Robotic Manipulation". In: *arXiv preprint arXiv:2012.07975* (2020) (cit. on p. 203).

[362] Jianyi Zhang and P. Weng. "Safe Distributional Reinforcement Learning". In: *International Conference on Distributional Artificial Intelligence*. 2021, (cit. on p. 97).

[363] Shangtong Zhang, Bo Liu, and Shimon Whiteson. "Gradientdice: Rethinking generalized offline estimation of stationary values". In: *International Conference on Machine Learning*. ICML. PMLR. 2020, (cit. on p. 112).

[364] Shifeng Zhang, Longyin Wen, Xiao Bian, Zhen Lei, and Stan Z Li. "Single-shot refinement neural network for object detection". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. CVPR'18. 2018, (cit. on p. 76).

[365] Yiru Zhao, Zhongming Jin, Guo-jun Qi, Hongtao Lu, and Xian-sheng Hua. "An adversarial approach to hard triplet generation". In: *European Conference on Computer Vision*. ECCV'18. 2018, (cit. on p. 63).

[366] Li Zheng, Siddhant Kumar, and Dennis M. Kochmann. "Data-driven topology optimization of spinodoid metamaterials with seamlessly tunable anisotropy". In: *Computer Methods in Applied Mechanics and Engineering* (2020), (cit. on p. 137).

[367] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip H. S. Torr. "Conditional Random Fields As Recurrent Neural Networks". In: *IEEE International Conference on Computer Vision*. ICCV. 2015, (cit. on pp. 3, 45).

[368] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. "Learning transferable architectures for scalable image recognition". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. CVPR'18. 2018 (cit. on p. 63).

V

Appendix

A
Appendix to Chapter 2

A.1 Guidelines for Setting the Values of λ

In practice, λ has to be chosen appropriately, but we found its exact choice uncritical (no precise tuning was required). Nevertheless, note that λ should cause a noticeable disruption in the optimization problem from equation (2.3), otherwise it is too likely that $y(w) = y_\lambda(w)$ resulting in a zero gradient. In other words, λ should roughly be of the magnitude that brings the two terms in the definition of w' in Prop. 2.4.1 to the same order:

$$\lambda \approx \frac{\langle w \rangle}{\left\langle \frac{\partial L}{\partial y} \right\rangle}$$

where $\langle \cdot \rangle$ stands for the average. This again justifies that λ is a **true hyperparameter** and that there is no reason to expect values around $\lambda \to 0^+$.

A.2 Proofs[i]

Proof of Proposition 2.4.1. Let us write $L = L(\hat{y})$ and $\nabla L = \frac{\mathrm{d}L}{\mathrm{d}y}(\hat{y})$, for brevity. Thanks to the linearity of $\mathbf{c}$ and the definition of f, we have

$$\mathbf{c}(\hat{w}, y) + \lambda f(y) = \hat{w}y + \lambda\big(L + \nabla L(y - \hat{y})\big) = (\hat{w} + \lambda\nabla L)y + \lambda L - \lambda\nabla L\hat{y} = \mathbf{c}(w', y) + \mathbf{c}_0,$$

where $\mathbf{c}_0 = \lambda L - \lambda\nabla L\hat{y}$ and $w' = \hat{w} + \lambda\nabla L$ as desired. The conclusion about the points of minima then follows. ■

[i] Proofs mostly attributed to Vít Musil and Michal Rolínek.

Before we prove Theorem 2.4.1, we make some preliminary observations. To start with, due to the definition of the solver, we have the fundamental inequality

$$\mathbf{c}(w, y) \geq \mathbf{c}\big(w, y(w)\big) \quad \text{for every } w \in W \text{ and } y \in Y. \tag{A.1}$$

Observation 1. *The function $w \mapsto \mathbf{c}\big(w, y(w)\big)$ is continuous and piecewise linear.*

Proof. Since $\mathbf{c}$'s are linear and distinct, $\mathbf{c}\big(w, y(w)\big)$, as their pointwise minimum, has the desired properties. ■

Analogous fundamental inequality

$$\mathbf{c}(w, y) + \lambda f(y) \geq \mathbf{c}\big(w, y_\lambda(w)\big) + \lambda f\big(y_\lambda(w)\big) \quad \text{for every } w \in W \text{ and } y \in Y \tag{A.2}$$

follows from the definition of the solution to the optimization problem (2.3).

A counterpart of Observation 1 reads as follows.

Observation 2. *The function $w \mapsto \mathbf{c}\big(w, y_\lambda(w)\big) + \lambda f\big(y_\lambda(w)\big)$ is continuous and piecewise affine.*

Proof. The function under inspection is a pointwise minimum of distinct affine functions $w \mapsto \mathbf{c}(w, y) + \lambda f(y)$ as y ranges Y. ■

As a consequence of above-mentioned fundamental inequalities, we obtain the following two-sided estimates on f_λ.

Observation 3. *The following inequalities hold for $w \in W$*

$$f\big(y_\lambda(w)\big) \leq f_\lambda(w) \leq f\big(y(w)\big).$$

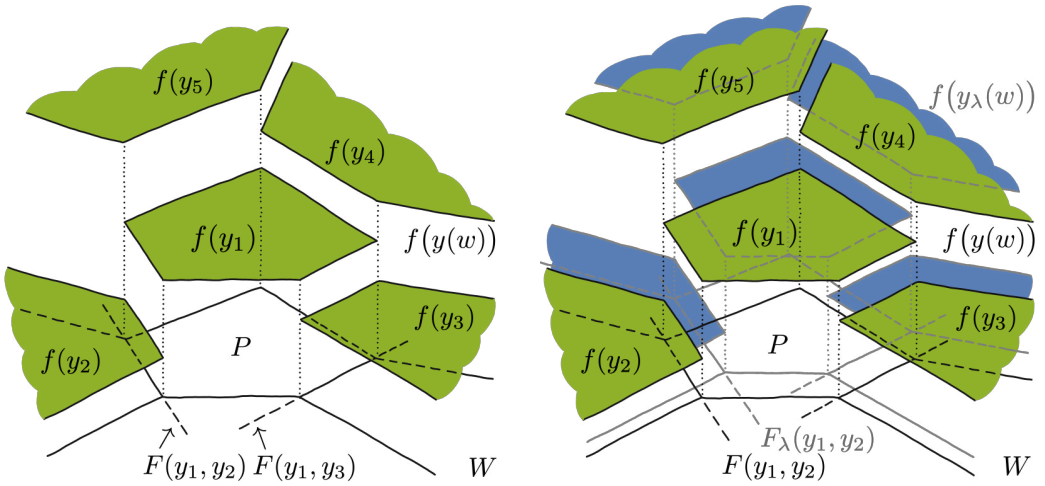
Proof. Inequality (A.1) implies that $\mathbf{c}\big(w, y(w)\big) - \mathbf{c}\big(w, y_\lambda(w)\big) \leq 0$ and the first inequality then follows simply from the definition of f_λ. As for the second one, it suffices to apply (A.2) to $y = y(w)$. ■

Now, let us introduce few notions that will be useful later in the proofs. For a fixed λ, W partitions into maximal connected sets P on which $y_\lambda(w)$ is constant (see figure A.1). We denote this collection of sets by $\mathcal{W}_\lambda$ and set $\mathcal{W} = \mathcal{W}_0$.

For $\lambda \in \mathbb{R}$ and $y_1 \neq y_2 \in Y$, we denote

$$F_\lambda(y_1, y_2) = \big\{w \in W : c(w, y_1) + \lambda f(y_1) = c(w, y_2) + \lambda f(y_2)\big\}.$$

We write $F(y_1, y_2) = F_0(y_1, y_2)$, for brevity. For technical reasons, we also allow negative values of λ here.



(a) The situation for $\lambda = 0$. We can see the polytope P on which $y(w)$ attains $y_1 \in Y$. The boundary of P is composed of segments of lines $F(y_1, y_k)$ for $k = 2, \ldots, 5$.

(b) The same situation is captured for some relatively small $\lambda > 0$. Each line $F_\lambda(y_1, y_k)$ is parallel to its corresponding $F(y_1, y_k)$ and encompasses a convex polytope in $\mathcal{W}_\lambda$.

Figure A.1: The family $\mathcal{W}_\lambda$ of all maximal connected sets P on which y_λ is constant.

Note, that if $W = \mathbb{R}^N$, then F_λ is a hyperplane since **c**'s are linear. In general, W may just be a proper subset of $\mathbb{R}^N$ and, in that case, F_λ is just the restriction of a hyperplane onto W. Consequently, it may happen that $F_\lambda(y_1, y_2)$ will be empty for some pair of y_1, y_2 and some $\lambda \in \mathbb{R}$. To emphasize this fact, we say "hyperplane in W". Analogous considerations should be taken into account for all other linear objects. The note "in W" stands for the intersection of these linear object with the set W.

Observation 4. *Let $P \in \mathcal{W}_\lambda$ and let $y_\lambda(w) = y$ for $w \in P$. Then P is a convex polytope in W, where the facets consist of parts of finitely many hyperplanes $F_\lambda(y, y_k)$ in W for some $\{y_k\} \subset Y$.*

Proof. Assume that $W = \mathbb{R}^N$. The values of y_λ may only change on hyperplanes of the form $F_\lambda(y, y')$ for some $y' \in Y$. Then P is an intersection of corresponding half-spaces and therefore P is a convex polytope. If W is a proper subset of $\mathbb{R}^N$ the claim follows by intersecting all the objects with W. ■

Observation 5. *Let $y_1, y_2 \in Y$ be distinct. If nonempty, the hyperplanes $F(y_1, y_2)$ and $F_\lambda(y_1, y_2)$ are parallel and their distance is equal to $|\lambda| K(y_1, y_2)$, where*

$$K(y_1, y_2) = \frac{|f(y_1) - f(y_2)|}{\|y_1 - y_2\|}.$$

Proof. If we define a function $c(w) = \mathbf{c}(w, y_1) - \mathbf{c}(w, y_2) = w(y_1 - y_2)$ and a constant $C = f(y_2) - f(y_1)$, then our objects rewrite to

$$F(y_1, y_2) = \{w \in W : c(w) = 0\} \quad \text{and} \quad F_\lambda(y_1, y_2) = \{w \in W : c(w) = \lambda C\}.$$

Since c is linear, these sets are parallel and $F(y_1, y_2)$ intersects the origin. Thus, the required distance is the distance of the hyperplane $F_\lambda(y_1, y_2)$ from the origin, which equals to $|\lambda C|/\|y_1 - y_2\|$. ■

As the set Y is finite, there is a uniform upper bound K on all values of $K(y_1, y_2)$. Namely

$$K = \max_{\substack{y_1, y_2 \in Y \\ y_1 \neq y_2}} K(y_1, y_2). \tag{A.3}$$

A.3 Proof of Theorem 2.4.1

Proof of Property A1. Now, Property A1 follows, since

$$f_\lambda(w) = \frac{1}{\lambda}\Big[\mathbf{c}\big(w, y_\lambda(w)\big) + \lambda f\big(y_\lambda(w)\big)\Big] - \frac{1}{\lambda}\mathbf{c}\big(w, y(w)\big)$$

and f_λ is a difference of continuous and piecewise affine functions. ■

Proof of Property A2. Let $0 < \lambda_1 \leq \lambda_2$ be given. We show that $W^{\lambda_2}_{\text{eq}} \subseteq W^{\lambda_1}_{\text{eq}}$ which is the same as showing $W^{\lambda_1}_{\text{dif}} \subseteq W^{\lambda_2}_{\text{dif}}$. Assume that $w \in W^{\lambda_2}_{\text{eq}}$, that is, by the definition of $W^{\lambda_2}_{\text{eq}}$ and f_λ,

$$\mathbf{c}\big(w, y(w)\big) + \lambda_2 f\big(y(w)\big) = \mathbf{c}(w, y_2) + \lambda_2 f(y_2), \tag{A.4}$$

in which we denoted $y_2 = y_{\lambda_2}(w)$. Our goal is to show that

$$\mathbf{c}\big(w, y(w)\big) + \lambda_1 f\big(y(w)\big) = \mathbf{c}(w, y_1) + \lambda_1 f(y_1), \tag{A.5}$$

where $y_1 = y_{\lambda_1}(w)$ as this equality then guarantees that $w \in W^{\lambda_1}_{\text{eq}}$. Observe that (A.2) applied to $\lambda = \lambda_1$ and $y = y(w)$, yields the inequality "$\geq$" in (A.5).

Let us show the reversed inequality. By Observation 3 applied to $\lambda = \lambda_1$, we have

$$f\big(y(w)\big) \geq f(y_1). \tag{A.6}$$

We now use (A.2) with $\lambda = \lambda_2$ and $y = y_1$, followed by equality (A.4) to obtain

$$\begin{aligned}
\mathbf{c}(w, y_1) + \lambda_1 f(y_1) &= \mathbf{c}(w, y_1) + \lambda_2 f(y_1) + (\lambda_1 - \lambda_2) f(y_1) \\
&\geq \mathbf{c}(w, y_2) + \lambda_2 f(y_2) + (\lambda_1 - \lambda_2) f(y_1) \\
&= \mathbf{c}\big(w, y(w)\big) + \lambda_2 f\big(y(w)\big) + (\lambda_1 - \lambda_2) f(y_1) \\
&= \mathbf{c}\big(w, y(w)\big) + \lambda_1 f\big(y(w)\big) + (\lambda_2 - \lambda_1)\big[f\big(y(w)\big) - f(y_1)\big] \\
&\geq \mathbf{c}\big(w, y(w)\big) + \lambda_1 f\big(y(w)\big)
\end{aligned}$$

where the last inequality holds due to (A.6).

Next, we have to show that $W^{\lambda}_{\text{dif}} \to \emptyset$ as $\lambda \to 0^+$, i.e. that for almost every $w \in W$, there is a $\lambda > 0$ such that $w \notin W^{\lambda}_{\text{dif}}$. To this end, let $w \in W$ be given. We can assume that $y(w)$ is a unique solution of solver (1.19), since two solutions, say y_1 and y_2, coincide only on the hyperplane $F(y_1, y_2)$ in W, which is of measure zero. Thus, since Y is finite, the constant

$$c = \min_{\substack{y \in Y \\ y \neq y(w)}} \big\{ \mathbf{c}(w, y) - \mathbf{c}\big(w, y(w)\big) \big\}$$

is positive. Denote

$$d = \max_{y \in Y} \big\{ f\big(y(w)\big) - f(y) \big\}. \tag{A.7}$$

If $d > 0$, set $\lambda < c/d$. Then, for every $y \in Y$ such that $f\big(y(w)\big) > f(y)$, we have

$$\lambda < \frac{\mathbf{c}(w, y) - \mathbf{c}\big(w, y(w)\big)}{f\big(y(w)\big) - f(y)}$$

which rewrites

$$\mathbf{c}\big(w, y(w)\big) + \lambda f\big(y(w)\big) < \mathbf{c}(w, y) + \lambda f(y). \tag{A.8}$$

For the remaining y's, (A.8) holds trivially for every $\lambda > 0$. Therefore, $y(w)$ is a solution of the minimization problem (2.3), whence $y_\lambda(w) = y(w)$. This shows that $w \in W^{\lambda}_{\text{eq}}$ as we wished. If $d = 0$, then $f\big(y(w)\big) \leq f(y)$ for every $y \in Y$ and (A.8) follows again. ■

Proof of Property A3. Let $y_1 \neq y_2 \in Y$ be given. We show that on the component of the set

$$\{ w \in W : y(w) = y_1 \text{ and } y_\lambda(w) = y_2 \} \tag{A.9}$$

the function f_λ agrees with a δ-interpolator, where $\delta \leq C\lambda$ and $C > 0$ is an absolute constant. The claim follows as there are only finitely many sets and their components of the form (A.9) in W^{λ}_{dif}.

Let us set

$$h(w) = \mathbf{c}(w, y_1) - \mathbf{c}(w, y_2) \quad \text{for } w \in W$$

and

$$g(w) = f(y_2) - \frac{1}{\lambda} h(w).$$

The condition on $\mathbf{c}$ tells us that h is a non-constant affine function. It follows by the definition of $F(y_1, y_2)$ and $F_\lambda(y_1, y_2)$ that

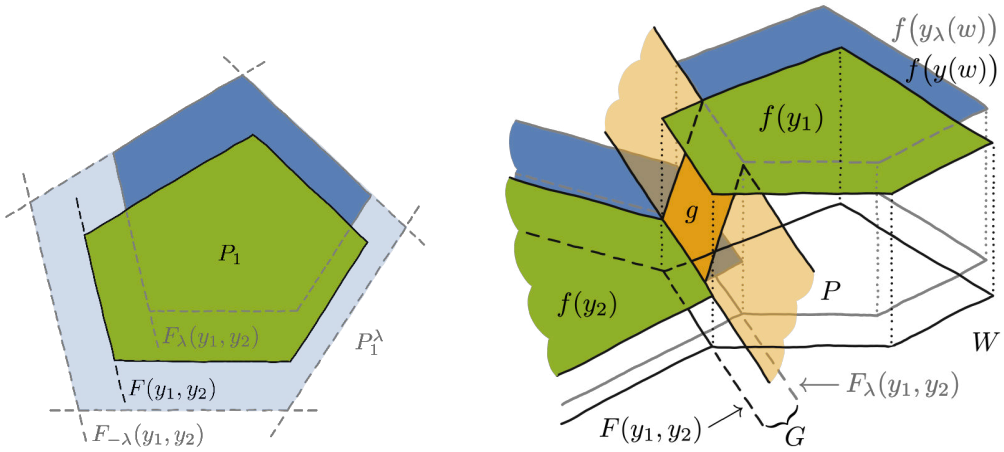
$$h(w) = 0 \quad \text{if and only if} \quad w \in F(y_1, y_2) \tag{A.10}$$

and

$$h(w) = \lambda\big(f(y_2) - f(y_1)\big) \quad \text{if and only if} \quad w \in F_\lambda(y_1, y_2). \tag{A.11}$$

By Observation 5, the sets F and F_λ are parallel hyperplanes. Denote by G the nonempty intersection of their corresponding half-spaces in W. We show that g is a δ-interpolator of f on G between y_1 and y_2, with δ being linearly controlled by λ.

We have already observed that g is the affine function ranging from $f(y_1)$ – on the set $F_\lambda(y_1, y_2)$ – to $f(y_2)$ – on the set $F(y_1, y_2)$. It remains to show that g attains both the values $f(y_1)$ and $f(y_2)$ at most δ-far from the sets P_1 and P_2, respectively, where $P_k \in \mathcal{W}$ denotes a component of the set $\{w \in W : y(w) = y_k\}$, $k = 1, 2$.



(a) The facets of P_1 consist of parts of hyperplanes $F(y_1, z_k)$ in W. Each facet $F(y_1, z_k)$ has its corresponding shifts F_λ and $F_{-\lambda}$, from which only one intersects P. The polytope P_1^λ is then bounded by those outer shifts.

(b) The interpolator g attains the value $f(y_1)$ on a part of $F_\lambda(y_1, y_2)$ – a border of the domain G. The value $f(y_2)$ is attained on a part of $F(y_1, y_2)$ – the second border of the strip G.

Figure A.2: The polytopes P_1 and P_1^λ and the interpolator g.

Consider y_1 first. By Observation 4, there are $z_1, \dots, z_\ell \in Y$, such that facets of P_1 are parts of hyperplanes $F(y_1, z_1), \dots, F(y_1, z_\ell)$ in W. Each of them separates W into two half-spaces, say W_k^+ and W_k^-, where W_k^- is the half-space which contains P_1 and W_k^+ is the other one. Let us denote

$$c_k(w) = \mathbf{c}(w, y_1) - \mathbf{c}(w, z_k) \quad \text{for } w \in W \text{ and } k = 1, \dots, \ell.$$

Every c_k is a non-zero linear function which is negative on W_k^- and positive on W_k^+. By the definition of y_1, we have

$$\mathbf{c}(w, y_1) + \lambda f(y_1) \le \mathbf{c}(w, z_k) + \lambda f(z_k) \quad \text{for } w \in P_1 \text{ and for } k = 1, \dots, \ell,$$

that is

$$c_k(w) \leq \lambda\big(f(z_k) - f(y_1)\big) \quad \text{for } w \in P_1 \text{ and for } k = 1, \ldots, \ell.$$

Now, denote

$$W_k^\lambda = \big\{w \in W : c_k(w) \leq \lambda\big|f(z_k) - f(y_1)\big|\big\} \quad \text{for } k = 1, \ldots, \ell.$$

Each W_k^λ is a half-space in W containing W_k^- and hence P_1. Let us set $P_1^\lambda = \bigcap_{k=1}^{\ell} W_k^\lambda$. Clearly, $P_1 \subseteq P_1^\lambda$ (see figure A.2). By Observation 5, the distance of the hyperplane $\big\{w \in W : c_k(w) = \lambda\big|f(z_k) - f(y_1)\big|\big\}$ from P_1 is at most λK, where K is given by (A.3). Therefore, since all the facets of P_1^λ are at most λK far from P_1, there is a constant C such that each point of P_1^λ is at most $C\lambda$ far from P_1.

Finally, choose any $w_1 \in P_1^\lambda \cap F_\lambda(y_1, y_2)$. By (A.11), we have $g(w_1) = f(y_1)$, and by the definition of P_1^λ, w_1 is no farther than $C\lambda$ away from P_1.

Now, let us treat y_2 and define the set P_2^λ analogous to P_1^λ, where each occurrence of y_1 is replaced by y_2. Any $w_2 \in P_2^\lambda \cap F(y_1, y_2)$ has desired properties. Indeed, (A.10) ensures that $g(w_2) = f(y_2)$ and w_2 is at most $C\lambda$ far away from P_2. ■

A.4 Details of Experiments

A.4.1 Warcraft Shortest Path

The maps for the dataset have been generated with a custom random generation process by using 142 tiles from the Warcraft II tileset [125]. The costs for the different terrain types range from 0.8–9.2. Some example maps of size 18×18 are presented in figure A.3a together with a histogram of the shortest path lengths. We used the first five layers of ResNet18 followed by a max-pooling operation to extract the latent costs for the vertices.

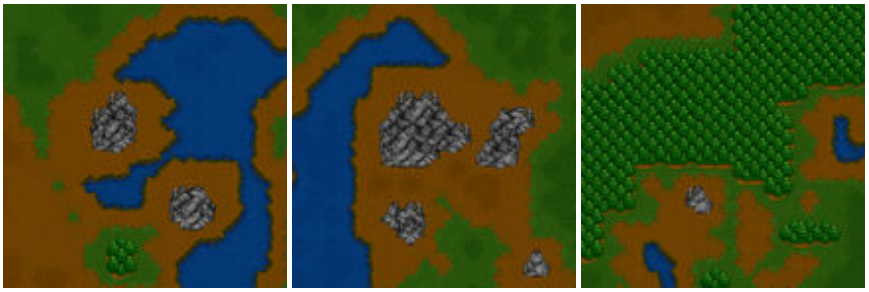
Optimization was carried out via Adam optimizer [158] with scheduled learning rate drops dividing the learning rate by 10 at epochs 30 and 40. Hyperparameters and model details are listed in table A.1

k	Optimizer(LR)	Architecture	Epochs	Batch Size	λ
12, 18, 24, 30	Adam(5×10^{-4})	subset of ResNet18	50	70	20

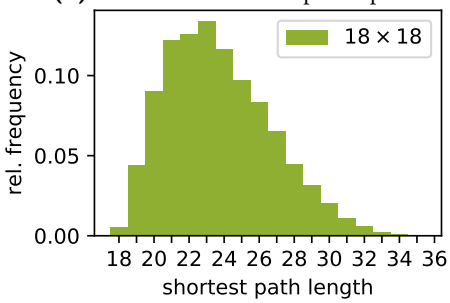
Table A.1: Experimental setup for Warcraft Shortest Path.

MNIST Min-cost Perfect Matching

The dataset consists of randomly generated grids of MNIST digits that are sampled from a subset of 1000 digits of the full MNIST dataset. We trained a fully convolutional neural



(a) Three random example maps.



(b) the shortest path distribution in the training set. All possible path lengths (18-35) occur.

Figure A.3: Warcraft SP(18) dataset.

network with two convolutional layers followed by a max-pooling operation that outputs a $k \times k$ grid of vertex costs for each example. The vertex costs are transformed into the edge costs via the known cost function and the edge costs are then the inputs to the Blossom V solver [89] as implemented in [165].

Regarding the optimization procedure, we employed the Adam optimizer along with scheduled learning rate drops dividing the learning rate by 10 at epochs 10 and 20, respectively. Other training details are in table A.2. Lower batch sizes were used to reduce GPU memory requirements.

k	**Optimizer(LR)**	**Architecture** [channels, kernel size, stride]	**Epochs**	**Batch Size**	λ
4, 8	Adam(10^{-3})	[[20, 5, 1], [20, 5, 1]]	30	70	10
16	Adam(10^{-3})	[[50, 5, 1], [50, 5, 1]]	30	40	10
24	Adam(10^{-3})	[[50, 5, 1], [50, 5, 1]]	30	30	10

Table A.2: Experimental setup for MNIST Min-cost Perfect Matching.

A.4.2 Globe Traveling Salesman Problem

For the Globe Traveling Salesman Problem we used a convolutional neural network architecture of three convolutional layers and two fully connected layers. The last layer outputs a vector of dimension $3k$ containing the k 3-dimensional representations of the respective countries' capital cities. These representations are projected onto the unit sphere and the matrix of pairwise distances is fed to the TSP solver.

The high combinatorial complexity of TSP has negative effects on the loss landscape and results in many local minima and high sensitivity to random restarts. For reducing sensitivity to restarts, we set Adam parameters to $\beta_1 = 0.5$ (as it is done for example in GAN training [251]) and $\varepsilon = 10^{-3}$.

The local minima correspond to solving planar TSP as opposed to spherical TSP. For example, if all cities are positioned to almost identical locations, the network can still make progress but it will never have the incentive to spread the cities apart in order to reach the global minimum. To mitigate that, we introduce a repellent force between epochs 15 and 30. In particular, we set

$$L_{\text{rep}} = \mathop{\mathbb{E}}_{i \neq j} e^{-\|x_i - x_j\|}$$

where $x_i \in \mathbb{R}^3$ for $i = 1, \ldots, k$ are the positions of the k cities on the unit sphere. The regularization constants C_k were chosen as $2.0, 3.0, 6.0$, and 20.0 for $k \in \{5, 10, 20, 40\}$.

For fine-tuning we also introduce scheduled learning rate drops where we divide the learning rate by 10 at epochs 80 and 90.

k	Optimizer(LR)	Architecture [channels, kernel size, stride], linear layer size	Epochs	Batch Size	λ
5, 10, 20	Adam(10^{-4})	[[20, 4, 2], [50, 4, 2], 500]	100	50	20
40	Adam(5×10^{-5})	[[20, 4, 2], [50, 4, 2], 500]	100	50	20

Table A.3: Experimental setup for the Globe Traveling Salesman Problem.

In figure 2.5b, we compare the true city locations with the ones learned by the hybrid architecture. Due to symmetries of the sphere, the architecture can embed the cities in any rotated or flipped fashion. We resolve this by computing "the most favorable" isometric transformation of the suggested locations. In particular, we solve the orthogonal Procrustes problem [114]

$$R^* = \mathop{\arg\min}_{R : R^T R = I} \|RX - Y\|^2$$

where X are the suggested locations, Y the true locations, and R^* the optimal transformation to apply. We report the resulting offsets in kilometers in table A.4.

k	5	10	20	40
Location offset (km)	69 ± 11	19 ± 5	11 ± 5	58 ± 7

Table A.4: Average errors of city placement on the Earth.

A.5 Traveling Salesman with an Approximate Solver

Since approximate solvers often appear in practice where the combinatorial instances are too large to be solved exactly in reasonable time, we test our method also in this setup. In particular, we use the approximate solver (OR-Tools [113]) for the Globe TSP. We draw two conclusions from the numbers presented below in table A.5.

(i) The choice of the solver matters. Even if OR-Tools is fed with the ground truth representations (i.e. true locations) it does not achieve perfect results on the test set (see the right column). We expect, that also in practical applications, running a suboptimal solver (e.g. a differentiable relaxation) substantially reduces the maximum attainable performance.

(ii) The suboptimality of the solver didn't harm the feature extraction – the point of our method. Indeed, the learned locations yield performance that is close to the upper limit of what the solver allows (compare the middle and the right column).

	Embedding	OR-tools	OR-tools on GT locations
k	Train %	Test %	Test %
5	99.8 ± 0.0	99.3 ± 0.1	100.0
10	84.3 ± 0.2	84.4 ± 0.2	88.6
20	49.2 ± 0.2	48.6 ± 0.8	54.4
40	14.6 ± 0.1	15.1 ± 0.3	15.2

Table A.5: Perfect path accuracy for Globe TSP using the approximate solver OR-Tools [113]. The maximal achievable performance is in the right column, where the solver uses the ground truth city locations.

B

Appendix to Chapter 3

B.1 Parameters of Retrieval Experiments

In all experiments we used the ADAM optimizer with a weight decay value of 4×10^{-4} and batch size 128. All experiments ran at most 80 epochs with a learning rate drop by 70% after 35 epochs and a batch memory of length 3. We used higher learning rates for the embedding layer as specified by defaults in Cakir et al. [49].

We used a super-label batch preparation strategy in which we sample a consecutive batches for the same super-label pair, as specified by Cakir et al. [49]. For the In-shop Clothes dataset we used 4 batches per pair of super-labels and 8 samples per class within a batch. In the Online Products dataset we used 10 batches per pair of super-labels along with 4 samples per class within a batch. For CUB200, there are no super-labels and we just sample 4 examples per classes within a batch. These values again follow Cakir et al. [49]. The remaining settings are in table B.1.

	Online Products	In-shop	CUB200
lr	3×10^{-6}	10^{-5}	5×10^{-6}
margin	0.02	0.05	0.02
λ	4	0.2	0.2

Table B.1: Hyperparameter values for retrieval experiments.

B.2 Proofs

Lemma B.2.1. *Let $\{w_k\}$ be a sequence of nonnegative weights and let $r_1, \dots, r_n$ be positive integers. Then*

$$\sum_{k=1}^{\infty} w_k |\{i : r_i \geq k\}| = \sum_{i=1}^{n} W(r_i), \tag{B.1}$$

where

$$W(k) = \sum_{i=1}^{k} w_i \quad \text{for } k \in \mathbb{N}. \tag{B.2}$$

Note that the sum on the left hand-side of (B.1) is finite.

Proposition B.2.2. *Let w_K be nonnegative weights for $K \in \mathbb{N}$ and assume that L_{rec} is given by*

$$L_{rec}(y, y^*) = \sum_{K=1}^{\infty} w_K \, L@K(y, y^*). \tag{B.3}$$

Then

$$L_{rec}(y, y^*) = \frac{1}{|\operatorname{rel}(y^*)|} \sum_{i \in \operatorname{rel}(y^*)} W(r_i), \tag{B.4}$$

where W is as in (B.2).

Proof. Taking the complement of the set $\operatorname{rel}(y^*)$ in the definition of $L@K$, we get

$$L@K(y, y^*) = \frac{|\{i \in \operatorname{rel}(y^*) : r_i \geq K\}|}{|\operatorname{rel}(y^*)|}, \tag{B.5}$$

whence eq. (B.3) reads as

$$L_{\text{rec}}(y, y^*) = \frac{1}{|\operatorname{rel}(y^*)|} \sum_{k=1}^{\infty} w_K |\{i : r_i \geq K\}|.$$

Equation eq. (B.4) then follows by Lemma Theorem B.2.1. ∎

proof of Lemma Theorem B.2.1. Observe that $w_k = W(k) - W(k-1)$ and $W(0) = 0$. Then

$$\begin{aligned}
\sum_{i=1}^{n} W(r_i) &= \sum_{k=1}^{\infty} W(k)|\{i : r_i = k\}| \\
&= \sum_{k=1}^{\infty} W(k)\big|\{i : r_i \geq k\} \setminus \{i : r_i \geq k+1\}\big| \\
&= \sum_{k=1}^{\infty} W(k)|\{i : r_i \geq k\}| \\
&\quad - \sum_{k=1}^{\infty} W(k-1)|\{i : r_i \geq k\}| \\
&= \sum_{k=1}^{\infty} \big(W(k) - W(k-1)\big)|\{i : r_i \geq k\}| \\
&= \sum_{k=1}^{\infty} w_k|\{i : r_i \geq k\}|
\end{aligned}$$

and eq. (B.1) follows. ■

Proof of eq. (3.18). Let us set $w_k = \log(1+1/k)$ for $k \in \mathbb{N}$. Then from Taylor's expansion of log we have the desired $w_k \approx \frac{1}{k}$ and

$$\begin{aligned}
W(k) &= \sum_{i=1}^{k} \log\left(1 + \frac{1}{i}\right) \\
&= \log\left(\prod_{i=1}^{k} \frac{1+i}{i}\right) = \log(1+k).
\end{aligned}$$

If we set

$$w_k = \log\left(1 + \frac{\log\left(1 + \frac{1}{k}\right)}{1 + \log k}\right), \quad \text{for } k \in \mathbb{N}$$

then, using Taylor's expansions again,

$$w_k \approx \frac{\log\left(1 + \frac{1}{k}\right)}{1 + \log k} \approx \frac{1}{k \log k}$$

and

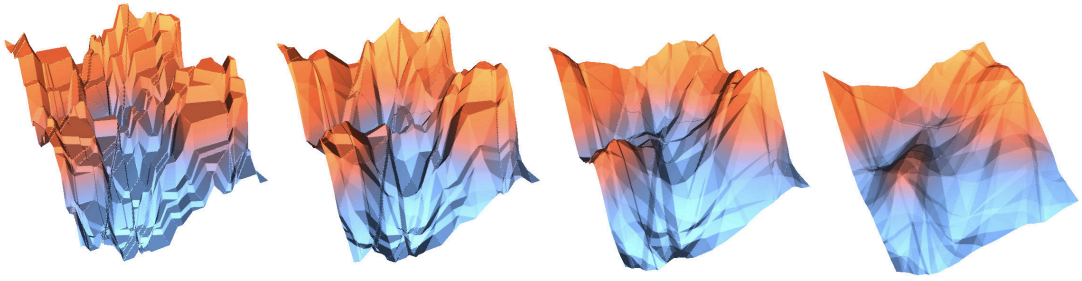
$$\begin{aligned} W(k) &= \sum_{i=1}^{k} \log\left(1 + \frac{\log\left(1+\frac{1}{k}\right)}{1+\log k}\right) \\ &= \log\left(\prod_{i=1}^{k} \frac{1+\log(1+i)}{1+\log i}\right) \\ &= \log\big(1+\log(1+k)\big). \end{aligned}$$

The conclusion then follows by Proposition B.2.2. ■

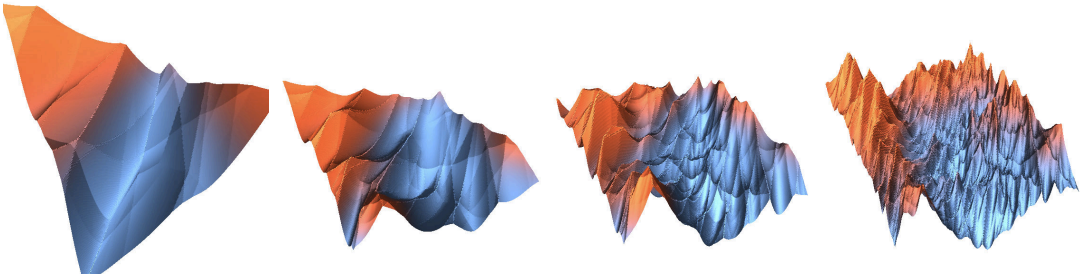
B.3 Ranking Surrogates Visualization

For the interested reader, we additionally present visualizations of smoothing effects introduced by different approaches for direct optimization of rank-based metrics. We display the behaviour of our approach using blackbox differentiation [334], of FastAP [48], and of SoDeep [91].

In the following, we fix a 20-dimensional score vector $w \in \mathbb{R}^{20}$ and a loss function L which is a (random but fixed) linear combination of the ranks of w. We plot a (random but fixed) two-dimensional section of $\mathbb{R}^{20}$ of the loss landscape $L(w)$. In Figure B.2a we see the true piecewise constant function. In Figure B.2b, Figure B.2c and Figure B.2d the ranking is replaced by interpolated ranking [334], FastAP soft-binning ranking [48] and by pretrained SoDeep LSTM [91], respectively. In Figure B.1a and Figure B.1b the evolution of the loss landscape with respect to parameters is displayed for the blackbox ranking and FastAP.

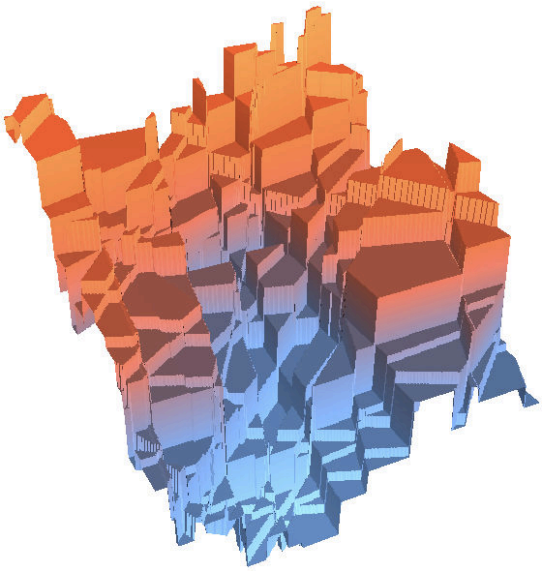


(a) Ranking interpolation by [334] for $\lambda = 0.2, 0.5, 1.0, 2.0$.

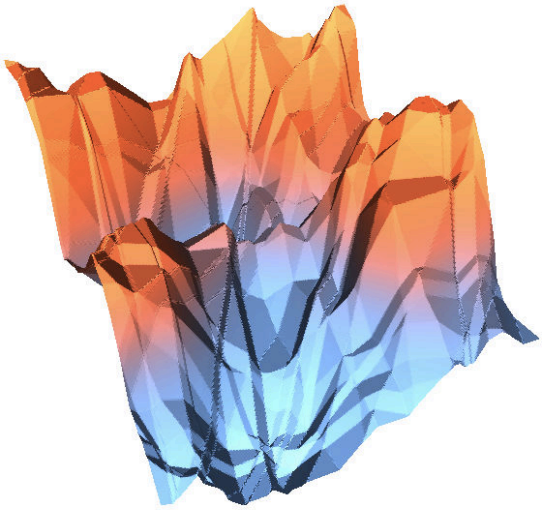


(b) FastAp [48] with bin counts $5, 10, 20, 40$.

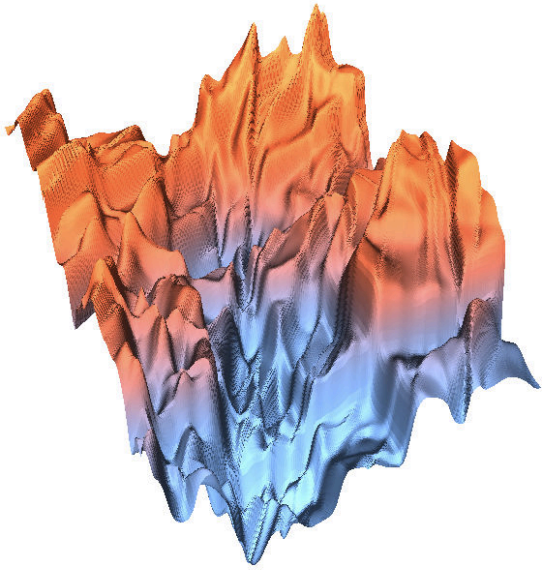
Figure B.1: Evolution of the ranking-surrogate landscapes with respect to their parameters.



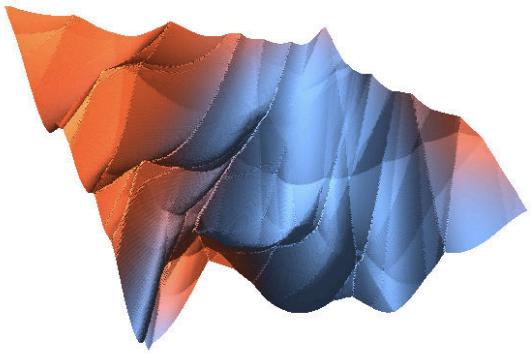
(a) Original piecewise constant landscape



(b) Piecewise linear interpolation scheme of [334] with $\lambda = 0.5$



(c) SoDeep LSTM-based ranking surrogate [91]



(d) FastAP [48] soft-binning with 10 bins.

Figure B.2: Visual comparison of various differentiable proxies for piecewise constant function.

C

Appendix to Chapter 4

C.1 Data Generation

To do imitation learning, we require expert data. For Crash Jewel Hunt (Crash 5×5 and 5×10), Leaper(Grid) and Maze we can determine the exact ground truth costs leading to optimal behavior. As an example, Crash 5×5 contains moving boxes that when encountered lead to instant death, meaning infinite costs and otherwise the fixed cost of moving around in the environment.

Since the environments become deterministic for a fixed random seed, we first unrolled their dynamics for each level. After obtaining the underlying grid structure and entities, we labeled them with costs and constructed a graph that reflects the grid structure. An expert trajectory is constructed by applying Dijkstra's algorithm on this graph and the human-labeled costs and then executing in simulation.

For the Crash Jewel Hunt experiments, we randomly sampled 2000 solvable levels by varying number of boxes per column, their speed, the agent start position and the jewel position. The training levels were taken from the first half and the second half of levels was used for testing. For the ProcGen environments Leaper(Grid) and Maze we have taken the levels determined by seeds 0-1000.

For Chaser, we applied a similar procedure but additionally, we recorded two sets of human trajectories, as we observed benefits in performance by incorporating more different expert trajectories for the same level. Since both the search procedure and human labeling are time consuming for this environment, we collected fewer expert trajectories for the Chaser than for the other environments, 3×100, two-thirds of which are from human players.

Level seeds 1000000-1001000 were taken for testing in the ProcGen experiments.

	Crash 5×5	Crash 5×10	Leaper(Grid)	Maze
learning rate	10^{-3}	10^{-3}	10^{-3}	10^{-3}
α	0.2	0.2	0.15	0.15
λ	20	20	20	20
resnet layers	4	4	4	4
kernel size	4	4	6	6
batch size	32	32	16	16

Table C.1: Training hyperparameters, where $alpha$ denotes the margin that was used on the vertex costs and λ the interpolation parameter for blackbox differentiation of Dijkstra's algorithm. We vary the kernel size of the initial convolution for ResNet18.

C.2 Environments

Our method is applicable in discrete environments, therefore we evaluated on environments from the ProcGen benchmark and the Crash Jewel Hunt environment.

We created the Crash Jewel Hunt environment to evaluate our method, where the goal is for the fox (Crash) to reach the jewel. We found this environment convenient since we can influence the combinatorial difficulty directly, which is not true for the ProcGen benchmark where we are limited to the random seeds used in the OpenAI implementation. The sources of variation in the Crash Jewel Hunt are the box velocities, initial positions, sizes, as well as the agent initial position and the jewel position.

We modified the Leaper environment to make grid steps for our method to be applicable. This involved making the logs on the river move in discrete steps as well as the agent. Moreover, in our version, the agent is not transported by the logs as they move, but has to move actively with them. For an additional description of the ProcGen environmnets, we refer the reader to Cobbe et al. [67].

C.3 Network Architecture and Input

For all of our experiments, we use the PyTorch implementation of the ResNet18 architecture as the base of our model. All approaches receive two stacked frames of the two previous time steps as input to make dynamics prediction possible. For the PPO baseline, we did not observe any benefit in adding the stacked frames as input and we used stable-baselines implementation from OpenAI to train it on the ProcGen environments.

In the case of the behavior cloning baseline, the problem is a multi-class classification prob-

lem with the output being a multinomial distribution over actions.

For the variant NAP*, we train a cost prediction network on top of which we run Dijkstra's algorithm on the output costs of the planning graph. This requires modifications to the original ResNet18 architecture. We remove the linear readout of the original ResNet18 architecture and replace it with a convolutional layer of filter size 1 and adaptive max pooling layer to obtain the desired dimensions of the underlying latent planning graph. More concretely, the output x of the last ResNet18 block is followed by the following operation (as output by PyTorch) to obtain the graph costs:

```
Sequential(
    Conv2d(256, 2, kernel_size=(1, 1), stride=(1, 1))
    Abs()
    AdaptiveMaxPool2d(output_size=(grid_height, grid_width))
)
```

Where grid_{height,width} denotes the height and width of the planning grid. For the full variant of NAP with goal prediction and agent position prediction we have a separate position classifier that has the same base architecture as the cost prediction network with 2 additional linear readouts for the likelihoods of the latent graph vertices, more concretely (as output by PyTorch):

```
Sequential(
    Conv2d(256, 2, kernel_size=(1, 1), stride=(1, 1))
    Abs()
    AdaptiveMaxPool2d(output_size=(grid_height, grid_width))
    Flatten()
    Linear(grid_height × grid_width, grid_height × grid_width)
)
```

For training the position classifier, we use a standard cross-entropy loss on the likelihoods. For NAP with position classification, we use the ground-truth expert start and goal positions to calculate the Hamming loss of the predicted path by the solver. At evaluation time, NAP uses the position classifier to determine the start and end vertices in the latent planning graph.

C.4 Training Procedure

For Crash 5×5, Crash 5×10, Leaper(Grid) and Maze we train the models on the same #levels, namely 1, 2, 5, 10, 20, 50, 100, 200, 500 and 1000. We evaluate on unseen 1000 levels in order to show that NAP exhibits superior generalization. The levels are generated as per

description in section C.1. Each dataset is normalize to be zero mean and unit variancesize. For each dataset size (#levels) we run experiments with 3 random restarts (seeds for network initialization). For all experiments, we make use of the ADAM optimizer.

We determine the number of epochs for training depending on each dataset size as $\min(150000/\#\text{levels}, 15000)$ to have roughly the same number of gradient updates in each experiment. We take the minimum over the 2 values because for smaller number of levels a large number iterations is not necessary to achieve good performance, but for a larger number of levels it is necessary. If we observe no error on the training set, we stop the training.

For the CHASER, the training conditions were analogous to the other environments, only of slightly smaller scale due to its higher complexity. Models were trained on 10, 20, 50, 100 and 200 levels and evaluated on 200 unseen levels.

	CHASER
learning rate	$1e^{-3}$
α	0.2
λ	40
resnet layers	3
kernel size	4
batch size	16

Table C.2: Training hyperparameters for the CHASER experiment, where α denotes the margin that was used on the vertex costs and λ the interpolation parameter for blackbox differentiation od Dijkstra.

C.4.1 PPO Training Procedure

The training of the PPO baseline is exactly the same as described in Cobbe et al. [67] using the official code from https://github.com/openai/train-procgen, see table C.3 for the used parameters. The network architecture is the IMPALA-CNN. The algorithm is trained on the specified number of levels for 200 million environments interactions gathered from 256 (instead of 64 as in Cobbe et al. [67]) to compensate for not having access to 4 parallel workers. We report numbers for 3 independent restarts.

C.4.2 DrAC Training Procedure

For the DrAC algorithm, we run all versions introduced by Raileanu et al. [255] (Meta-DrAC, RL2-DrAC, UCB-DrAC and DrAC-Crop) and choose the best one in the main

learning rate	$5e^{-4}$
α	0.2
discount γ	0.999
entropy coefficient	0.01
steps per update	2^{16}

Table C.3: PPO hyperparameters, as used in Cobbe et al. [67].

plots, denoted as DrAC*. We used the original hyperparameters from Raileanu et al. [255] and the implementation from https://github.com/rraileanu/auto-drac. As with the other experiments, we report numbers from 3 different random seeds.

C.5 On Comparing Imitation Learning to Reinforcement Learning

We compare our method to Data Augmented Actor Critic, PPO and a behavior cloning baseline. Arguably, since NAP is used in an imitation learning setting, it reaps benefits from having access to expert trajectories. Nevertheless, it is not straight forward that embedding a solver in a neural architecture leads to better generalization in comparison to reinforcement learning methods.

Behavior cloning with a standard neural architecture has access to the same amount of data, whereas reinforcement learning agents have access to orders of magnitude more data for inference ($2 \cdot 10^8$ transitions in comparison to $\sim 10^5$ max). This would lead us to believe that reinforcement learning agents are able to generalize well because of the sheer amount of data that they have at their disposal, but we show that nevertheless it is possible to extract meaningful policies even with a small number of training levels seen with expert trajectories, that are more optimal.

In addition, NAP is a general architecture paradigm that may be composed with various different objective functions, including a reinforcement learning formulation. It would be interesting to see how NAP behaves when used in such a formulation and if this would lead to even better generalization properties with more data. We provide training performance curves in Fig. C.1 and density plots for different numbers of training levels in Fig. C.2.

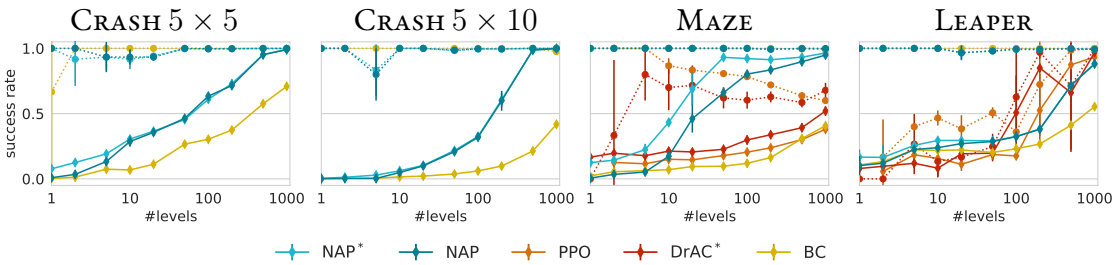


Figure C.1: The dotted lines denote the training performance of the methods. We observe that the behavior cloning baseline and NAP have fitted the training set almost perfectly.

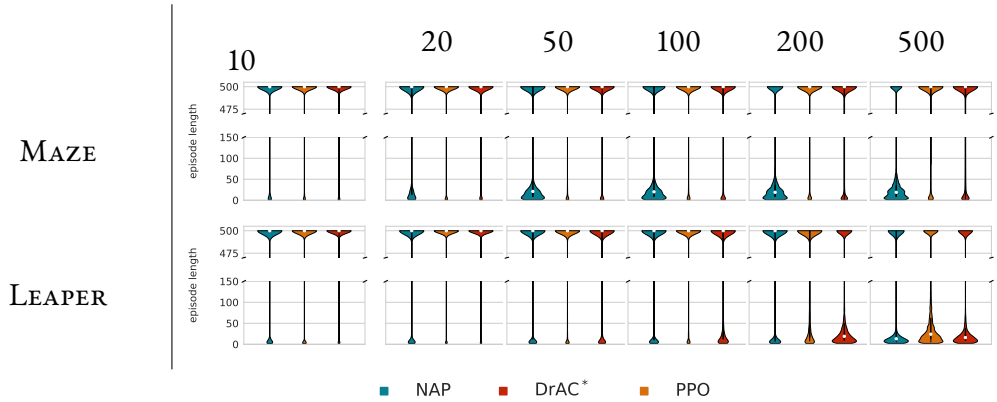


Figure C.2: Density plots of performance on the test set (1000 unseen levels) after different number of training levels for the MAZE and LEAPER environments, the white point denotes the median performance on the test set.

C.6 Data Regularized Actor-Critic

The DrAC algorithm [255] attacks the problem of generalization in reinforcement learning from a different angle, namely applying (in some versions optimized) data augmentations to the PPO algorithm. The main insight is that naively applied data augmentations result in faulty policy gradient estimates because the resulting policy after applying the augmentation is too far from the behavior policy.

To alleviate this, a policy regularization term G_π and value function regularization term G_V are added to the PPO objective:

$$\ell_{\text{DrAC}} = \ell_{\text{PPO}} - \alpha_r(G_\pi + G_V). \tag{C.1}$$

Furthermore, various augmentations and augmentation selection schemes were proposed. We ran all of the proposed selectin schemes on our environments, Meta-DrAC, RL2-DrAC, UCB-DrAC and DrAC. Meta-DrAC meta-learns the weights of a convolutional neural net-

work used for data augmentation. RL2-DrAC meta-learns a policy that selects an augmentation from a pre-defined set of augmentations. UCB-DrAC is a bandit formulation of the augmentation selection problem with application of a upper confidence bound algorithm for selection strategy. DrAC denotes the version with the crop augmentation, which has been shown to work well with more than half of the environments in the ProcGen benchmark. For more details, we refer the reader to Raileanu et al. [255].

DrAC's approach to improving generalization is orthogonal to NAP and the approaches may be composed in order to achieve even better generalization capabilities.

C.7 Additional Related Work

Generalization in reinforcement learning In addition to the work of Raileanu et al. [255], there is a plethora of approaches that attempt to improve generalization in reinforcement learning by considering various data augmentation techniques while mainly drawing motivation from supervised learning approaches [169, 176]. Other approaches combine unsupervised learning with data augmentation [307, 361].

Notably, the problem of sim-to-real transfer can be seen as a problem of generalization to different system dynamics. Domain randomization [319], i.e. augmenting system dynamics in a structured way, has emerged as one of the main techniques for tackling this problem.

C.8 Cost Margin Ablation

Here we show the results for the Maze environment with 200, 500 and 1000 train levels evaluated on 1000 unseen test levels, with and without margin. The results indicate that using the margin on the costs improves generalization.

	200	500	1000
w	0.839 ± 0.014	0.895 ± 0.007	0.948 ± 0.006
w/o	0.834 ± 0.015	0.883 ± 0.008	0.855 ± 0.241

Table C.4: Results for cost margin ablation in the Maze environment.

D Appendix to Chapter 5

In this supplementary we provide additional details for our method. We also provide videos that showcase risk-averse behavior of RAZER at https://sites.google.com/view/razer-traj-opt.

Our research suffered from pandemic impacts on lab access which is detailed in **??**.

D.1 Implementation Details

D.1.1 Model Learning

Parameters used for model learning in the *BridgeMaze* experiments.

We bound the predicted log variance by applying (as in [65, A.1])

```
logvar = max_logvar - softplus(max_logvar - logvar)

logvar = min_logvar + softplus(logvar - min_logvar)
```

to the output of the network that predicts the log variance, `logvar`. In principle, we could differentiate through this bound to automatically adjust the bounds `max_logvar` and `min_logvar`. However, we decided to not make these parameters learnable.

Parameters used for model learning in the *Noisy-HalfCheetah* environment (only differences to *BridgeMaze* environment).

For training the predictive model, we alternate between two phases: data collection and model fitting. In the *BridgeMaze* environment, we collect 5 rollouts of length 80 steps and append them to the previous rollouts. Afterwards, we fit the model for 25 epochs. For *Noisy-HalfCheetah*, we collect 1 rollout and fit for 50 epochs. For Noisy-FetchPickAndPlace and Solo8-LeanOverObject we replace the $\hat{f}$ in Figure 5.2 with independent instances of noisy ground truth simulators.

Table D.1: Model parameters for the *BridgeMaze* Environment.

Ensemble parameters

Name	Value
num_layers	6
size	400
activation	silu
ensemble_size (n)	5
output_activation	None
l1_reg	0
weight_initializer	truncated_normal
bias_initializer	0
use_spectral_normalization	False

Stochastic NN parameters

Name	Value
var_clipping_low	-10.0
var_clipping_high	4
state_dependent_var	True
regularize_automatic_var_scaling	False

Remaining parameters

Name	Value
lr	0.002
grad_norm	2.0
batch_size	512
weight_decay	$1e^{-5}$
use_input_normalization	True
use_output_normalization	False
epochs	25
predict_deltas	True
train_epochs_only_with_latest_data	False
iterations	0
optimizer	Adam
propagation_method	TS1
sampling_method	sample

D.1.2 Controller Parameters

Parameters used in the CEM controller. For an explanation of the different parameters, we refer the reader to Pinneri et al. [247].

D.1.3 Timings

While our code is not tuned for speed specifically, in table table D.6 we provide some timings for a single step in the environment (hyper-parameters are set as specified in Supplementary D.1.1 and Supplementary D.1.2, with num_simulated_trajectories $=$ 128 and op_iterations $=$ 3).

D.1.4 Uncertainty Separation

In our method, we separate the epistemic uncertainty, denoted as $\mathfrak{E}$ and aleatoric uncertainty, denoted as $\mathfrak{A}$, the details of which are explained in Section 5.3 with the resulting costs that arise. Since we are using a variant of the CEM algorithm that needs to sort the sampled action

Table D.2: Model parameters (only differences wrt table D.1 are shown) for *Noisy-HalfCheetah* environment.

Ensemble parameters

Name	Value
num_layers	4
size	200

Stochastic NN parameters

Name	Value
var_clipping_low	-6.0
state_dependent_var	True

Remaining parameters

Name	Value
lr	0.0002
grad_norm	None
batch_size	256
weight_decay	$3e^{-5}$
epochs	50

sequences $\boldsymbol{s}$ according to their cost, the cost of an action sequence is a single floating point number.

The stochastic NN ensemble that we are using samples trajectories from the predictive distribution ψ_τ for each action sequence $\boldsymbol{s}$. In addition, our variant (PETSUS), also propagates the mean prediction $\bar{s}_t$ for each ensemble member for an action sequence $\boldsymbol{s}$. The autoregressive prediction follows a recursive relation:

$$[\bar{s}_{t+1}, \Sigma_{t+1}] = \boldsymbol{\vartheta}(\bar{s}_t, a_t)$$

We make use of this in order to estimate the epistemic uncertainty $\mathfrak{E}$. At each time point of the predicted sequence of observations, we take the empirical variance of the outputted Gaussian parameters $\boldsymbol{\vartheta}(\bar{s}_t, a_t)$, predicted from the previous mean prediction $\bar{s}_t$ and control a_t, across the ensembles for that time slice in the predicted trajectories. This is then summed up across horizon H to obtain the epistemic bonus for action sequence $\boldsymbol{s}$.

Figure D.1 shows that scaling $w_{\mathfrak{E}}$ results in better state-coverage. This is of particular interest if we want to learn models that are able to generalize to different task settings, e.g. when changing the cost function. While the naive PETS algorithm overfits the model to the task at hand, RAZER learns a truly task-agnostic model and is able to reap the benefits of model-based approaches to control.

For the aleatoric penalty we rely on the actual predictions of the covariance $\Sigma(s_t, a_t)$ and average them across the time slice, following with the sum across horizon H. Alternatively to this, we also use the entropy of the Gaussian as the $\mathfrak{A}$ uncertainty measurement. In Supplementary D.1.5 we argue how these terms are interchangeable.

Note that, for the safety term ideally we want to use the full distribution ψ_τ and separation in aleatoric and epistemic uncertainty is neither required nor desirable.

Table D.3: Controller parameters, BridgeMaze environment.

Action sampler parameters

Name	Value
alpha	0.1
colored_noise	true
elite_size	10
execute_best_elite	true
finetune_first_action	false
fraction_elites_reused	0.3
init_std	0.5
keep_previous_elites	true
noise_beta	2.0
opt_iterations	3
relative_init	true
shift_elites_over_time	true
use_mean_actions	true

Remaining parameters

Name	Value
cost_along_trajectory	sum
delta	0.0
factor_decrease_num	1
horizon	30
num_simulated_trajectories	128

Table D.4: Controller parameters, Noisy-HalfCheetah environment (only difference wrt table D.3 are shown).

Action sampler parameters

Name	Value
noise_beta	0.25
opt_iterations	4

Remaining parameters

Name	Value
num_simulated_trajectories	120

D.1.5 Entropy vs. Variance as Uncertainty Measurement

We use entropy of Gaussian and variance interchangebly as uncertainty estimates. Indeed, since the Gaussian distribution is the maximum entropy distribution for certain variance σ^2, the entropy scales linearly with $\log \sigma^2$. We have found that utilizing the variance directly causes RAZER to be much more risk-averse, which can be explained by the variance not being suppressed by the log term in the entropy. Moreover, using the variance directly is much more interpretable and easier to tune because it's of the same scale as the observation space.

Table D.5: Controller parameters, Solo8-LeanOverObject environment (only difference wrt table D.3 are shown).

Action sampler parameters

Name	Value
init_std	0.3
noise_beta	3.0

Table D.6: Timings per one environment step in ms. We measured the timings on a system with 1 GeForce GTX 1050 Ti, an Intel Core i7-6800K and 31GB of memory.

Environment	Timing [ms]
BridgeMaze	0.25
Noisy-HalfCheetah	0.14

D.1.6 Observation Space vs. Cost Space Uncertainty

A natural question to ask when attempting to make efficient use of uncertainties in MPC is where to measure these uncertainties. As an alternative to observation space uncertainties, one could measure uncertainty in cost space. Here we argue why this is not a reasonable thing to do for each of the individual cost terms.

Epistemic Bonus Since we operate under the desiderata that the benefit of model-based methods is in task-agnosticism, we shouldn't measure epistemic uncertainty in the cost space, since this would decouple the task definition through the cost from the observation space and would lead to learning models that are not task-agnostic.

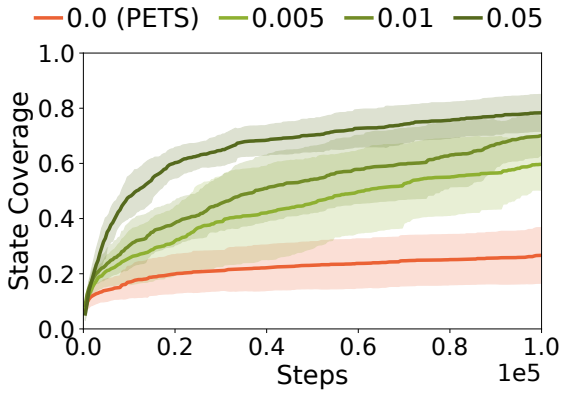


Figure D.1: Exploration over time.

Aleatoric Penalty This is perhaps the most questionable case for using observation space uncertainty instead of cost space uncertainty. Nevertheless, we assume that high-aleatoric uncertainty translates to control difficulty, and we want to avoid parts of the observation space that are difficult to control. Moreover, the uncertainty measurements become completely invalidated in the case of a task switch, which plays against the task-agnosticism desiderata.

Safety Penalty Safety is something that is enforced by infusing the algorithm with prior knowledge through a set of constraints which mostly manifest themselves as subsets of the observation space $\mathcal{X}$ or action space $\mathcal{U}$.

D.2 Algorithm

In Algorithm 7 we provide an overview of the CEM algorithm that we utilize for implementing RAZER. Concretely, we use an improved sample efficient version of CEM as proposed by Pinneri et al. [247] that involves shift-initialization of the distribution mean, sampling time-correlated noise and further improvements.

Algorithm 7 RAZER: Risk-aware and safe CEM-MPC

Parameters:
  N: number of samples; B: Number of particles, H: planning horizon; $w_{\mathfrak{A}}$, $w_{\mathfrak{E}}$, $w_{\mathfrak{S}}$ CEM-iterations
for t = 1 **to** T loop over episode length **do**
  for i = 1 **to** CEM-iterations **do**
    $(\text{samples}_p)_{p=1}^{P} \leftarrow N$ samples from CEM(μ_t^i, Σ_t^i), with P particles per sample
    $c, c_{\mathfrak{A}}, c_{\mathfrak{E}}, c_{\mathfrak{S}} \leftarrow$ compute cost functions over particles
    $c_{\text{tot}} = c + c_{\mathfrak{A}} + c_{\mathfrak{E}} + c_{\mathfrak{S}}$ *// compute total cost*
    elite-set$_t \leftarrow$ best K samples according to total cost
    $\mu_t^{i+1}, \Sigma_t^{i+1} \leftarrow$ fit Gaussian distribution to elite-set$_t$
  execute first action of best elite sequence
  shift-initialize μ_{t+1}^1

D.3 Environments

All environments are based on the MuJoCo physics engine [320]. The **Noisy-Halfcheetah** and **Noisy-FetchPickAndPlace** environments are based on *HalfCheetah-v3* and *FetchPickAndPlace-v1*, respectively.

BridgeMaze We designed the *BridgeMaze* environment to show the different aspects of uncertainty, namely the epistemic and aleatoric uncertainty, in isolation. The agent is a simple cube with only a free joint attached to it. The state-space $s = [x_0, x_1, x_2, a, b, c, d, v_{x_0}, v_{x_1}, v_{x_2}]$ is 10-dimensional, consisting of 3 positional (x_0 to x_2), 4 rotational (a to d) and 3 velocity-based (v_{x_0} to v_{x_2}), agent-centric coordinates. The action-space $u = [\tau_{x_0}, \tau_{x_1}]$ is 2-dimensional. The torque τ applied to the agent in x_0- and x_1-direction.

The task in the environment is to reach a goal platform at $x_0^\star \geq 12$ by crossing one of three bridges that go over deadly lava.

The domain reward is defined as

$$r_t(s_t, a_t, s_{t+1}) = \begin{cases} |(s_0)_t - s_0^\star| - |(s_0)_{t+1} - s_0^\star| & \text{, if } (s_1)_{t+1} \geq -1.5 \\ 0 & \text{, if } (s_0)_{t+1} \geq x_0^\star \text{ and } (s_1)_{t+1} \geq -1.5 \\ -1 & \text{, otherwise} \end{cases} \tag{D.1}$$

where $x^\star$ is the goal state. We define the cost for planning as $c_t(s_t, a_t, s_{t+1}) = -r_t(s_t, a_t, s_{t+1})$.

We designed the environments such that the agent is able to accelerate fast and also comes to a full stop relatively fast if no torque is applied. This makes the control problem and the task of learning the model relatively easy.

Noise is added in form of an external force in s_1-direction injected through the `xfrc_applied` attribute of the model. The sign of the force, as well as the force amplitude, sampled from $f_{\text{ext}} \in \mathcal{U}(0, f_{\text{ext}}^{\max})$, are randomly changing every 5 simulation steps. The external force is added only if $-8 \leq s_0 \leq 8$ and $-3.6 \leq x_1 \leq 3.6$. Otherwise the external force is zero.

Noisy-HalfCheetah We utilize a modified HalfCheetah environment where we apply a normally distributed noise term $\xi \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ to the simulator state in the case when the velocity of the cheetah is greater than 6. More concretely, let s_t denote the simulator state at time step t, then the modified state is calculated as follows:

$$s_t' = s_t + \xi_t \tag{D.2}$$

In our case, Σ is a diagonal covariance matrix with the diagonal terms equal to 0.2. In addition, for the safety experiments with the Noisy-HalfCheetah we create a virtual ceiling at height $h = 0.3$. In the case that the body height crosses this threshold, the agent incurs a large penalty. When the safety-constraint is violated, we don't end the episode.

Noisy-FetchPickAndPlace We modified the *FetchPickAndPlace-v1* environment to show the effect of the aleatoric penalty on the CEM action plan. Given the difficulty of the task, we

performed the experiments without the learned model, using instead an ensemble of noisy ground truth dynamics. In this way, we could more easily understand the role of the aleatoric uncertainty during planning.

The noise term $\xi \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ is applied to the action controlling the gripper state: a positive additive noise forces the robot to open the grip with a force proportional to the noise magnitude. This noise is applied to all the ground truth models of the ensemble, and to the environment as well.

In particular, the box position is centered at y-coordinate -1.5 while the target is at $y = 2.0$. The gripper state is noisy until $y = 1.67$, right before the target.

The dropping rate is computed considering the height variation of the box (z-coordinate). If the downward velocity is greater than a fixed threshold, the box is considered dropped. The threshold velocity also includes cases in which the box is dropped and possibly re-grabbed, as this is still part of the risky behaviors we want to avoid. The plotted dropping is the minimum over different aleatoric penalties.

Solo8-LeanOverObject The state space of the this environment is 47-dimensional. It contains the absolute position, rotation, velocity and angular velocity of the robot as well as the positions and velocities of all the joints. In addition, the state contains the positions of the end-effectors and of the sites at the front and back of the robot. The actions space is 8-dimensional and controls the relative position of the joints. We fixed the two front legs of the robot with a soft-constraint to the ground to prevent the robot from uncontrollable jumping. We apply Gaussian noise to the action with a mean of 0 and a diagonal covariance matrix with the diagonal elements all being 0.3. The noise is uniformly applied over the entire state-action-space.

The experiments for the *Solo8-LeanOverObject* environment use the ground truth model during planning. The same noise were applied in the 'mental' as well as the 'real' environment.

D.3.1 Computing State-Space Coverage

For computing the state coverage in Figure 5.3a we divided the continuous state-space in 50 equally spaced bins in the range $-20 \leq s_0 \leq 20$ and $-10 \leq s_1 \leq 15$. The state space-coverage is the fractions between states visited at least once and the total number of states.

D.4 Application to Transfer Learning

In this work we have demonstrated that an approach such as PETS[65] to data-driven MPC that relies on zero-order trajectory optimization of the expected cost is not enough to manage uncertain environments and safety constraints. These problems need to be addressed

when dealing with sim-to-real. The separation of uncertainties allows us to effectively manage epistemic uncertainty in the real system, which is important for improving the model once distribution shift to the real system happens. This can be done in a way of combining the epistemic bonus and probabilistic safety constraints, such that the policy explores parts of the state space where there is knowledge to be obtained while avoiding high-cost regions as a consequence of the incurred safety and aleatoric penalties.

In comparison to standard approaches for sim-to-real which involve domain randomization at training time, this approach incurs lower computational overhead and relies on learning on the real system.

E
Appendix to Chapter 6

E.1 Reproducibility

For implementation of DOI we have used the PyTorch autograd framework. For the Solo12 training we made use of Isaac Gym for data collection and evaluation of the learned skill policies. For the D4RL experiments we evaluated the policies using the Mujoco v2.1 rigid body simulator. The training of the skill policies with evaluation and pre-training of the SMODICE expert ratios takes about 4 hours on an NVIDIA GeForce RTX 4080 graphics card with a batch size of 512. We plan on opensourcing the code and the Solo12 data post conference acceptance. The Solo12 robot has been developed as part of the Open Dynamic Robot Initiative [122], and a full assembly kit is available at a cheap price in order to reproduce the real system experiments from Supplementary E.10.

E.2 Fenchel Conjugate

The Fenchel conjugate $f_\star$ of a function $f : \Omega \to \mathbb{R}$ is given by

$$f_\star(y) = \sup_{x \in \Omega} \langle x, y \rangle - f(x), \tag{E.1}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product defined on a space Ω. For any proper, convex and lower semi-continuous function f the following duality statement holds $f_{\star\star} = f$, that is

$$f(x) = \sup_{y \in \Omega_\star} \langle x, y \rangle - f_\star(y), \tag{E.2}$$

where $\Omega_\star$ denotes the domain of $f_\star$. For any probability distributions $p, q \in \Delta(S)$ with $p(s) > 0$ implying $q(s) > 0$, we define for convex continuous functions f the family of

f-divergences

$$\mathrm{D}_f(p||q) = \mathop{\mathbb{E}}_{q}\left[f\left(\frac{p(x)}{q(x)}\right)\right]. \tag{E.3}$$

The Fenchel conjugate of an f divergence $\mathrm{D}_f(p||q)$ at a function $y(s) = p(s)/q(s)$ is, under certain conditions[i], given by

$$\mathrm{D}_{\star,f}(y) = \mathop{\mathbb{E}}_{q(s)}\left[f_\star(y(s))\right]. \tag{E.4}$$

Furthermore, its maximizer satisfies

$$p^\star(s) = q(s) f'_\star(y(s)). \tag{E.5}$$

In the important special case where $f(x) = x\log(x)$, we obtain the well-known Kullback-Leibler (KL) divergence

$$\mathrm{D}_{\mathrm{KL}}(p||q) = \sum_s p(s)\log\frac{p(s)}{q(s)}. \tag{E.6}$$

The Fenchel conjugate $\mathrm{D}_{\star,\mathrm{KL}}$ of the KL-divergence at a function $y(s) = p(s)/q(s)$ has a closed-form [41, Example 3.25]

$$\mathrm{D}_{\star,\mathrm{KL}}(y) = \log \mathbb{E}_{q(s)}[\exp y(s)], \tag{E.7}$$

and its maximizer $p^\star$ satisfies

$$p^\star(s) = q(s)\mathrm{softmax}_q(y(s)), \quad \text{where} \quad \mathrm{softmax}_q(y(s)) = \frac{\exp y(s)}{\mathbb{E}_{q(s')}[\exp y(s'))]} \tag{E.8}$$

E.3 Lagrange Relaxation

The Lagrange relaxation is given by

$$\max_{d_z(s,a),q(z|s)} \min_{\lambda>0} \sum_z \mathbb{E}_{d_z(s)}\left[\frac{\log\left(|Z|q(z|s)\right)}{|Z|}\right] + \sum_z \lambda_z\left[\varepsilon - \mathrm{D}_{\mathrm{KL}}\left(d_z(S,A)||d_{\widetilde{E}}(S,A)\right)\right].$$

By combining Theorem E.4.4 and the definition of $\eta_{\widetilde{E}}(s,a) = d_{\widetilde{E}}(s,a)/d_O(s,a)$, we have

$$\mathrm{D}_{\mathrm{KL}}\left(d_z(S,A)||d_{\widetilde{E}}(S,A)\right) = \mathrm{D}_{\mathrm{KL}}\left(d_z(S,A)||d_O(S,A)\right) - \mathbb{E}_{d_z(s,a)}\left[\log\eta_{\widetilde{E}}(s,a)\right]$$

[i] f needs to satisfy certain regularity conditions [70]

and thus

$$\max_{d_z(s,a),q(z|s)} \min_{\lambda>0} \sum_z \lambda_z \left[\varepsilon + \mathbb{E}_{d_z(s,a)}\left[R_z^\lambda(s,a)\right] - \mathrm{D}_{\mathrm{KL}}\left(d_z(S,A)||d_O(S,A)\right)\right], \quad \text{(E.9)}$$

where the reward is given by

$$R_z^\lambda(s,a) := \frac{\log\left(|Z|q(z|s)\right)}{\lambda_z|Z|} + \log \eta_{\widetilde{E}}(s,a).$$

E.4 Algorithmic Phases

E.4.1 Value Function Training

With fixed skill-discriminator $q(z|s)$ and Lagrange multipliers $\lambda > 0$, the Problem E.9 becomes:

$$\max_{\{d_z(s,a)\}_{z\in Z}} \sum_z \lambda_z \left\{\mathbb{E}_{d_z(s,a)}\left[R_z^\lambda(s,a)\right] - \mathrm{D}_{\mathrm{KL}}\left(d_z(s,a)||d_O(s,a)\right)\right\}$$

or equivalently for every skill z:

$$\begin{aligned} \max_{d_z(s,a)\geq 0} \quad & \mathbb{E}_{d_z(s,a)}\left[R_z^\lambda(s,a)\right] - \mathrm{D}_{\mathrm{KL}}\left(d_z(S,A)||d_O(S,A)\right) \\ \text{s.t.} \quad & \textstyle\sum_a d_z(s,a) = (1-\gamma)\rho_0(s) + \gamma \mathcal{T} d(s) \quad \forall s. \end{aligned} \quad \text{(E.10)}$$

We note that the preceding problem formulation involves state-action occupancy.

$$\begin{aligned} V^\star = \quad & \arg\min_{V(s)} (1-\gamma)\mathbb{E}_{s\sim\rho_0}\left[V(s)\right] \\ & + \log \mathbb{E}_{d_O(s,a)} \exp\left\{R_z^\lambda(s,a) + \gamma \mathcal{T} V(s,a) - V(s)\right\}, \end{aligned} \quad \text{(E.11)}$$

where

$$\mathcal{T}V(s,a) = \mathbb{E}_{\mathcal{P}(s'|s,a)} V(s').$$

Moreover, the optimal primal solution reads

$$\frac{d_z^\star(s,a)}{d_O(s,a)} = \mathrm{softmax}_{d_O(s,a)}\left(R_z^\lambda(s,a) + \gamma \mathcal{T} V_z^\star(s,a) - V_z^\star(s)\right). \quad \text{(E.12)}$$

E.4.2 Skill Discriminator Training

With fixed skill-conditioned policy $\pi_z^\star$ and Lagrange multipliers $\lambda > 0$, the Problem E.9 becomes

$$\max_{q(z|s)} \sum_z \left\{ \mathbb{E}_{d_z(s,a)} \left[R_z^\lambda(s,a) \right] - \mathrm{D}_{\mathrm{KL}} \left(d_z(S,A) || d_O(S,A) \right) \right\}$$

and reduces to

$$\max_{q(z|s)} \mathbb{E}_{p(z)} \mathbb{E}_{d_z(s,a)} \log q(z|s).$$

Lemma E.4.1. *Given ratios $\eta_z(s,a)$, using weighted-importance sampling, we can train offline an optimal skill-discriminator $q(z|s)$. In particular, we optimize by gradient descent the following optimization problem*

$$\max_{q(z|s)} \mathbb{E}_{p(z)} \mathbb{E}_{d_O(s,a)} \left[\eta_z(s,a) \log q(z|s) \right].$$

Proof. The statement follows by combining Theorem E.4.2 and Theorem E.5.1. ■

Lemma E.4.2 (Discriminator Gradient)**.** *It holds that*

$$\nabla_\phi \mathbb{E}_{p(s)} \left[\mathrm{D}_{\mathrm{KL}} \left(p(Z|s) || q_\phi(Z|s) \right) \right] = -\mathbb{E}_{p(z)} \mathbb{E}_{p(s|z)} \left[\nabla_\phi \log q_\phi(z|s) \right].$$

Proof. Observe that

$$\begin{aligned} \nabla_\phi \mathrm{D}_{\mathrm{KL}} \left(p(Z|s) || q(Z|s) \right) &= \nabla_\phi \mathbb{E}_{p(z|s)} \log \frac{p(z|s)}{q_\phi(z|s)} \\ &= -\mathbb{E}_{p(z|s)} \nabla_\phi \log q_\phi(z|s), \end{aligned}$$

where the second equality follows by

$$\nabla_\phi \log \frac{p(z|s)}{q_\phi(z|s)} = -\frac{q_\phi(z|s)}{p(z|s)} p(z|s) \frac{\nabla_\phi q_\phi(z|s)}{[q_\phi(z|s)]^2} = -\frac{\nabla_\phi q_\phi(z|s)}{q_\phi(z|s)} = -\nabla_\phi \log q_\phi(z|s).$$

■

E.4.3 KL-divergence Constraint Violation

Lemma E.4.3 (State-Action KL Estimator)**.** *Suppose we are given offline datasets $\mathcal{D}_O(S,A) \sim d_O$, $\mathcal{D}_E(S) \sim d_E$ and optimal ratios $\eta_z(s,a) = \frac{d_z(s,a)}{d_O(s,a)}$ and $\eta_{\tilde{E}}(s,a) =$*

$\frac{d_{\widetilde{E}}(s,a)}{d_O(s,a)}$ *for all* $(s,a) \in \mathcal{D}_O$*, where the state-action occupancy* $d_{\widetilde{E}}$ *is induced by a policy* $\pi_{\widetilde{E}}$ *agreeing on the state occupancy of an expert* π_E*, i.e.*

$$\pi_{\widetilde{E}} \in \arg\min_{\pi} \mathrm{D}_{\mathrm{KL}}\left(d_\pi(S)||d_E(S)\right).$$

Then, we can compute offline an estimator of $\mathrm{D}_{\mathrm{KL}}\left(d_z(S,A)||d_{\widetilde{E}}(S,A)\right)$ *which is given by*

$$\phi_z = \mathbb{E}_{d_O(s,a)}\left[\eta_z(s,a)\log\frac{\eta_z(s,a)}{\eta_{\widetilde{E}}(s,a)}\right].$$

Proof. By Theorem E.4.4 we have

$$\mathrm{D}_{\mathrm{KL}}\left(d_z(S,A)||d_{\widetilde{E}}(S,A)\right) = \mathrm{D}_{\mathrm{KL}}\left(d_z(S,A)||d_O(S,A)\right) - \mathbb{E}_{d_z(s,a)}\left[\log\frac{d_{\widetilde{E}}(s,a)}{d_O(s,a)}\right].$$

For the first term, we have

$$\begin{aligned}\mathrm{D}_{\mathrm{KL}}\left(d_z(S,A)||d_O(S,A)\right) &= \mathbb{E}_{d_z(s,a)}\log\frac{d_z(s,a)}{d_O(s,a)}\\ &= \mathbb{E}_{d_O(s,a)}\left[\eta_z(s,a)\log\eta_z(s,a)\right].\end{aligned}$$

The second term reduces to

$$\mathbb{E}_{d_z(s,a)}\left[\log\frac{d_{\widetilde{E}}(s,a)}{d_O(s,a)}\right] = \mathbb{E}_{d_O(s,a)}\left[\eta_z(s,a)\log\eta_{\widetilde{E}}(s,a)\right].$$

■

Lemma E.4.4 (Structural)**.** *Suppose* $0 < \eta_z(s,a), \eta_{\widetilde{E}}(s,a) < \infty$ *for all* $(s,a) \in \mathcal{D}_O$*. Then, we have*

$$\mathrm{D}_{\mathrm{KL}}\left(d_z(S,A)||d_{\widetilde{E}}(S,A)\right) = \mathrm{D}_{\mathrm{KL}}\left(d_z(S,A)||d_O(S,A)\right) - \mathbb{E}_{d_z(s,a)}\left[\log\frac{d_{\widetilde{E}}(s,a)}{d_O(s,a)}\right].$$

Proof. By definition of KL-divergence, we have

$$\begin{aligned}\mathrm{D}_{\mathrm{KL}}\left(d_z(S,A)||d_{\widetilde{E}}(S,A)\right) &= \mathbb{E}_{d_z(s,a)}\left[\log\left(\frac{d_z(s,a)}{d_O(s,a)}\cdot\frac{d_O(s,a)}{d_{\widetilde{E}}(s,a)}\right)\right]\\ &= \mathrm{D}_{\mathrm{KL}}\left(d_z(S,A)||d_O(S,A)\right) - \mathbb{E}_{d_Z(s,a)}\left[\log\frac{d_{\widetilde{E}}(s,a)}{d_O(s,a)}\right].\end{aligned}$$

■

E.5 Importance Sampling

Lemma E.5.1 (Importance Sampling). *Given ratios $\eta_z(s,a)$, it holds for any function $f(s,a)$ that*

$$\mathbb{E}_{d_z^\star(s,a)}\left[f(s,a)\right] = \mathbb{E}_{d_O(s,a)}\left[\eta_z(s,a) f(s,a)\right].$$

In particular, for any function $g(s)$ we have

$$\mathbb{E}_{d_z^\star(s)}\left[g(s)\right] = \mathbb{E}_{d_O(s,a)}\left[\eta_z(s,a) g(s)\right].$$

Proof. The first conclusion follows by definition of $\eta_z(s,a) = d_z(s,a)/d_O(s,a)$, whereas the second uses

$$\mathbb{E}_{d_z^\star(s)}\left[g(s)\right] = \mathbb{E}_{d_z^\star(s,a)\pi_z^\star(a|s)}\left[g(s)\right] = \mathbb{E}_{d_z^\star(s,a)}\left[g(s)\right] = \mathbb{E}_{d_O(s,a)}\left[\eta_z(s,a) g(s)\right].$$

■

E.5.1 Empirical Estimators

Recall that the primal optimal solution satisfies

$$\eta_z(s,a) := \frac{d_z^\star(s,a)}{d_O(s,a)} = \text{softmax}_{d_O(s,a)}\left(R_z^\lambda(s,a) + \gamma \mathcal{T} V_z^\star(s,a) - V_z^\star(s)\right),$$

where

$$\text{softmax}_{p(x)}(g(x)) = \frac{\exp\{g(x)\}}{\mathbb{E}_{p(x')}[\exp\{g(x')\}]}. \tag{E.13}$$

In the rest of this section, we denote the above TD-error term by

$$\delta_z(s,a) = R_z^\mu(s,a) + \gamma \mathcal{T} V_z^\star(s,a) - V_z^\star(s).$$

By assumption, the offline dataset $\mathcal{D}_O$ is sampled u.a.r. from a state-action occupancy distribution $d_O(s,a)$. Let $\{w_z(s,a)\}_{(s,a)\in\mathcal{D}_O}$ be a discrete probability distribution, computed by a softmax, over the offline dataset $\mathcal{D}_O$, namely

$$w_z(s,a) = \text{softmax}_{\mathcal{D}_O}(\delta_z(s,a)) = \frac{\exp\{\delta_z(s,a)\}}{\sum_{(s',a')\in\mathcal{D}_O}\exp\{\delta_z(s',a')\}}.$$

We are now ready to state the main result of this section.

Lemma E.5.2 (KL-divergence Estimator)**.** *The following expression*

$$\sum_{(s,a)\in\mathcal{D}_O} w_z(s,a)\big[\log w_z(s,a) - \log w_{\widetilde{E}}(s,a)\big]$$

is an empirical estimator of the KL-divergence $\mathrm{D}_{\mathrm{KL}}\left(d_z(S,A)||d_{\widetilde{E}}(S,A)\right)$.

Proof. We estimate the expectation $\mathbb{E}_{d_O(s,a)}\exp\{\delta(s,a)\}$ using an empirical estimate $\frac{1}{|\mathcal{D}_O|}\sum_{(s,a)\in\mathcal{D}_O}\exp\{\delta_z(s,a)\}$ over the offline-dataset $\mathcal{D}_O$. By definition of $\text{softmax}_{d_O(s,a)}$, see eq. (E.13), the following expression

$$\widetilde{\eta}_z(s,a) = \frac{\exp\{\delta_z(s,a)\}}{\frac{1}{|\mathcal{D}_O|}\sum_{(s',a')\in\mathcal{D}_O}\exp\{\delta_z(s',a')\}} = |\mathcal{D}_O| w_z(s,a)$$

is an empirical estimator of the importance weight $\eta_z(s,a)$. Similarly, $\widetilde{\eta}_{\widetilde{E}}(s,a) = |\mathcal{D}_O| w_{\widetilde{E}}(s,a)$ is an estimator of $\eta_{\widetilde{E}}(s,a)$. Then, the statement follows by combining Lemma Theorem E.4.3, the definition of importance ratios $\eta_z(s,a) = d_z(s,a)/d_O(s,a)$, $\eta_{\widetilde{E}}(s,a) = d_{\widetilde{E}}(s,a)/d_O(s,a)$ and

$$\begin{aligned}
D_{\mathrm{KL}}\left(d_z(S,A)||d_{\widetilde{E}}(S,A)\right) &= \mathbb{E}_{d_O(s,a)}\left[\eta_z(s,a)\log\frac{\eta_z(s,a)}{\eta_{\widetilde{E}}(s,a)}\right] \\
&\approx \frac{1}{|\mathcal{D}_O|}\sum_{(s,a)\in\mathcal{D}_O}\widetilde{\eta}_z(s,a)\log\frac{\widetilde{\eta}_z(s,a)}{\widetilde{\eta}_{\widetilde{E}}(s,a)} \\
&= \sum_{(s,a)\in\mathcal{D}_O} w_z(s,a)\log\left(\frac{w_z(s,a)}{w_{\widetilde{E}}(s,a)}\right).
\end{aligned}$$

■

Lemma E.5.3 (Off-Policy Expectation Estimator)**.** *For any function* $f(s,a)$ *the following expression*

$$\sum_{(s,a)\in\mathcal{D}_O} w_z(s,a) f(s,a)$$

is an empirical estimator of the expectation $\mathbb{E}_{d_z^\star(s,a)}\left[f(s,a)\right]$.

Proof. By combining Theorem E.5.1 and similar arguments as in the proof of Theorem E.5.2, we have

$$\begin{aligned}
\mathbb{E}_{d_z^\star(s,a)}\left[f(s,a)\right] &= \mathbb{E}_{d_O(s,a)}\left[\eta_z(s,a)f(s,a)\right] \\
&\approx \frac{1}{|\mathcal{D}_O|}\sum_{(s,a)\in\mathcal{D}_O}\widetilde{\eta}_z(s,a)f(s,a) \\
&= \sum_{(s,a)\in\mathcal{D}_O} w_z(s,a)f(s,a).
\end{aligned}$$

■

E.6 Unconstrained Formulation

SMODICE [201] minimizes a KL-divergence between the policy state occupancy and the expert state occupancy, expressed as

$$\min_{d(S)} \mathrm{D}_{\mathrm{KL}}\left(d(S)||d_E(S)\right). \tag{E.14}$$

A naive approach to extend the above problem formulation to the unsupervised skill discovery setting, is to consider an additional diversity term in the objective. In particular, adding a scaled mutual information term $\mathcal{I}(S;Z)$ and maximizing over a set of skill-conditioned state occupancies $\{d_z(S)\}_{z\in Z}$, namely

$$\max_{\{d_z(S)\}_{z\in Z}} \alpha\mathcal{I}(S;Z) - \sum_{z\in Z} \mathrm{D}_{\mathrm{KL}}\left(d_z(S)||d_E(S)\right). \tag{E.15}$$

Here, the level of diversity is controlled by a hyperparameter α. However, α is arbitrary, and no constraint on closeness to the expert state occupancy is enforced. We proceed by using the variational lower bound in eq. (6.3) and assuming a categorical uniform distribution $p(z)$ over the set of latent skills Z, which consists of $|Z|$ distinct indicator vectors in $\mathbb{R}^{|Z|}$. This reduce the optimization problem to

$$\max_{d_z(s),q(z|s)} \sum_{z\in Z} \left\{ \alpha\mathbb{E}_{d_z(s)}\left[\frac{\log\left(q(z|s)|Z|\right)}{|Z|}\right] - \mathrm{D}_{\mathrm{KL}}\left(d_z(S)||d_E(S)\right)\right\}. \tag{E.16}$$

Theorem E.6.1. *[201] Suppose Theorem 6.3.1 holds. Then, we have*

$$\mathrm{D}_{\mathrm{KL}}\left(d_z(S)\|d_E(S)\right) \leq \mathop{\mathbb{E}}_{d_z(s)}\left[\log\frac{d_O(s)}{d_E(s)}\right] + \mathrm{D}_{\mathrm{KL}}(d_z(S,A)\|d_O(S,A)).$$

By Theorem E.6.1 and linearity of the objective, Problem (E.16) reduces to optimizing separately for each latent skill z the following optimization problem

$$\max_{d_z(s),q(z|s)} \mathbb{E}_{d_z(s)}\left[R_z^\alpha(s,a)\right] - \mathrm{D}_{\mathrm{KL}}(d_z(S,A)\|d_O(S,A)), \tag{E.17}$$

where $R_z^\alpha(s,a)$ is defined as

$$R_z^\alpha(s,a) := \underbrace{\log\frac{d_E(s)}{d_O(s)}}_{\text{Expert Imitation}} + \underbrace{\alpha\frac{\log\left(q(z|s)|Z|\right)}{|Z|}}_{\text{Skill Diversity}}. \tag{E.18}$$

The ratios $\frac{d_E(s)}{d_O(s)}$ can be computed by training a discriminator $c(s)$ tasked to distinguish between samples from $d_E(s)$ and $d_O(s)$. More specifically, since the optimal Bayes discriminator satisfies $c^\star(s) = d_E(s)/(d_E(s)+d_O(s))$, in practice we can use an estimator $c(s)/(1-c(s)) \approx \frac{d_E(s)}{d_O(s)}$.

Similar to the DOI, we can apply the alternating optimization scheme, here with two phases:(i) fixed skill-discriminator (similarly to Section 6.4.2); and (ii) fixed importance ratios and policy $\pi_z^\star$, where we train the skill-discriminator $q(z|s)$ (see Supplementary E.4.2). For the first phase, we use the importance ratios $\eta_z(s,a)$ computed by optimizing the dual-value problem and then applying softmax to the corresponding TD error terms (see eq. (6.13) and Nachum and Dai [221] and Ma et al. [201]).

E.7 Solo-12 Dataset Collection

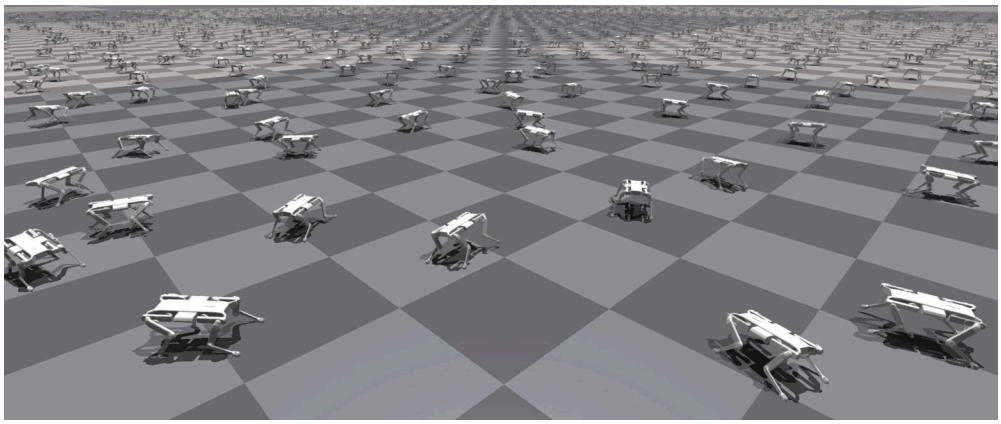


Figure E.1: Solo-12 datasets are collected with 4000 environments in parallel using IsaacGym.

As shown in Figure E.1, both *expert dataset* and *offline dataset* are collected in parallelized GPU-based environments in Isaac Gym [203]. The policies from both locomotion task and obstacle navigation tasks with SOLO12 are trained using the DOMiNiC [62] algorithm to exhibit diverse behaviors while maintaining a certain level of task completion. For details on the algorithm used to train the data collection policies, we refer interested readers to [62].

Locomotion task. The collecting policies are trained to track randomly sampled velocity commands on the flat ground. The state space consists of the linear and angular base velocity vectors, projected gravity vector, joint position, and velocity and commanded velocity. The actions contain the joint target angles, which will be taken by a PD controller to generate applied torque for each motor. During collecting, the policies are fed with a fixed

forward velocity command of 1 m/s, and zeros for side velocity and yaw rate. As mentioned in Section 5.4, the policy used for collecting the *expert dataset* is the last and best checkpoint (iteration 2000) and trained without diversity objective, which exhibits a stable mid-height trotting gait pattern. The policies for collecting the *offline dataset* are different stochastic checkpoints throughout the training of the skill-conditioned policy. The intrinsic reward is designed to maximize the ℓ_2 distance of the successor features [22] between distinct skills, where in this setting the feature space includes: the base height velocity, base roll and pitch velocities, and feet height velocities. The *offline dataset* is composed of 1/2 data from checkpoint 0, 1/4 data from checkpoint 50, 1/8 data from checkpoint 100, 1/16 data from checkpoint 500, 1/32 data from checkpoint 1500 and 1/32 data from checkpoint 2000. For each policy checkpoint, we collect data from the 5 corresponding skills, including the target skill. It is worth noting that more than half of the data from the *offline dataset* comes from the nearly random policies from the start of the training (checkpoints 0 and 50). Both datasets contain 4000 trajectories with an episode length of 250 steps, or 1 million transitions each.

Obstacle navigation task. The policies are trained to track the target position in a terrain of random obstacles of various heights of $\{0.0, 0.05, \ldots, 0.25\}$ meters within a fixed time horizon. The state space of the agent contains the linear and angular base velocity vectors, projected gravity vector, joint position and velocity, a surrounding height map of the robot and time information, while the actions remain the same as the locomotion task. During data collection, the policies are tasked with tracking the target 3.0 meter away in the front direction while confronting a 1.0×1.0 meter square obstacle of 0.2 meter height. The intrinsic reward for training the policy is designed to diversify the base velocity direction such that distinct skills exhibit diverse strategies. For the *expert dataset*, the used policy is the last and best checkpoint (iteration 2000) trained with diversity objective. The *expert dataset* is multi-modal in nature, as the dataset contains diverse strategies for navigating in front of the obstacle, either avoiding it from both sides or climbing it. On the other hand, the policies for collecting the *offline dataset* are the skill-conditioned checkpoints from iterations $\{0, 50, 100, 150, 200, 250, 500, 1000, 1500, 2000\}$. Both datasets contain 2000 trajectories with an episode length of 500 steps, or 1 million transitions each.

Sim-to-Real transfer. In addition, we use domain randomization during training and data collection, in order to tackle the sim-to-real transfer and to simulate more diverse environment interaction. Specifically, we randomize the friction coefficient between $[0.5, 1.5]$, additional base mass between $[-0.5, 0.5]$ kg, and simulate the observation noise and an actuator lag of 15 ms.

E.8 SMODICE Expert Return

In table E.1 we show the performance of the evaluated policies trained by SMODICE[201] on the WALKER2D and HALFCHEETAH. The results are consistent with the performance that we obtain with DOI in Figure 6.6. We also note here the importance of having expert state coverage in the offline data that is reflected in the performance of the policies.

Environment	dataset	N	r
halfcheetah	medium-expert	25	81.25
		50	80.47
		200	73.56
	medium-replay	25	29.28
		50	36.73
		200	60.67
	random	25	10.89
		50	27.71
		200	78.94
walker2d	medium-expert	25	3.98
		50	19.22
		200	4.10
	medium-replay	25	15.09
		50	3.60
		200	0.95
	random	25	52.62
		50	103.52
		200	108.20

Table E.1: Expected return for SMODICE-learned expert policies in the WALKER2D and ANT environments for N expert trajectories mixed-in.

E.9 Lagrange Multiplier Stability

In Figure E.2 we observe the behavior of the Lagrange multipliers for different levels of ε for a specific skill z in the SOLO12 experiment. In case of $\varepsilon \in \{1.0, 2.0\}$, the multipliers fluctuate around a specific level that strikes the balance between diversity and expert imitation. This can also be validated when observing the violation level in Figure E.2b of the constraint given estimator ϕ_z, which is for $\varepsilon \in \{1.0, 2.0\}$ around 0. On the other hand, if we introduce

a strong constraint on the KL-divergence ($\varepsilon = 0.0$), which is constantly violated, hence $\sigma(\mu_z) = 1$. Similarly, if the constraint is too weak, only diversity is optimized, in which case there is a significant degradation in performance (see figure Figure 6.3).

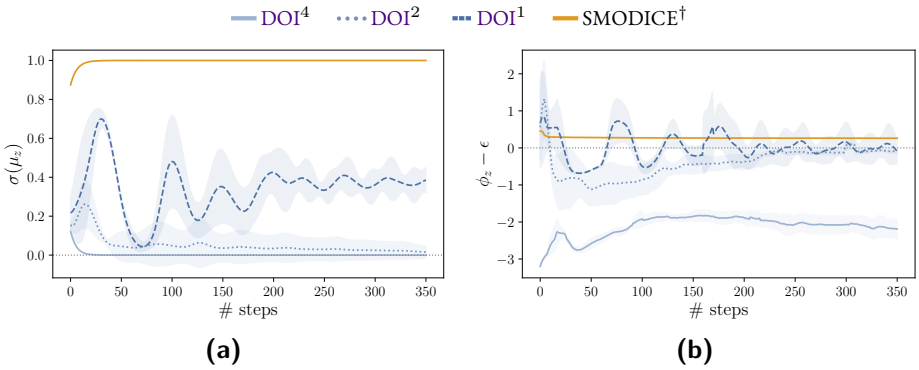


Figure E.2: Behavior of Lagrange multipliers. (a) Evolution of $\sigma(\lambda_z)$ for one skill ($z = 1$ chosen arbitrarily), (b) violation of the constraint for different ε. Negative $\phi_z - \varepsilon$ indicates no violation. Means and standard deviation across restarts.

In Figure E.3 we show the bounded lagrange multiplier values for three skills and the resulting violations for different ε levels for the ANT experiment. Again, the multiplier values fluctuate around appropriate levels ensuring the the violation of the constraint remains close to 0.

E.10 Real Robot Deployment

For the locomotion task, we successfully deployed policies exhibiting diverse skills extracted from the *offline dataset* while being able to track a certain velocity similar to the expert on real hardware. Our skill-conditioned policy exhibits different walking behaviors with diverse base motions. Snapshots of these diverse behaviors can be seen in Figure E.4.

E.11 Observation Projection

Imitation learning is of particular interest when the agent's and the target expert policy's state spaces do not necessarily match, but overlap in certain parts, as is often the case when learning from demonstrations. Our framework naturally accounts for this. If we consider $\mathcal{S}'$ to be the state space of the expert and $\mathcal{S}$ the state space of the agent, we assume that there exists a simple projection mapping $\Pi : \mathcal{S}' \mapsto \mathcal{O}$, where $\mathcal{O} := \{o : o \subset s, s \in \mathcal{S}\}$ is the power set of observations, allowing us to potentially imitate beyond expert policies with the same state

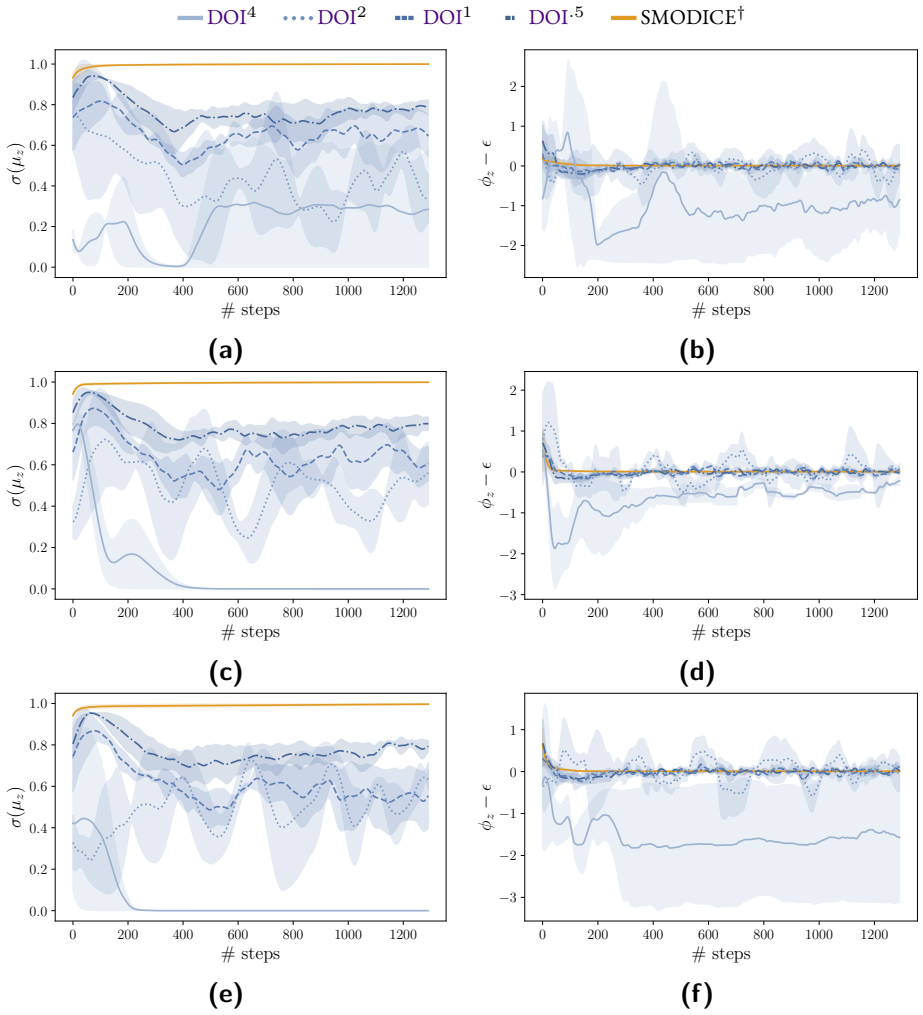
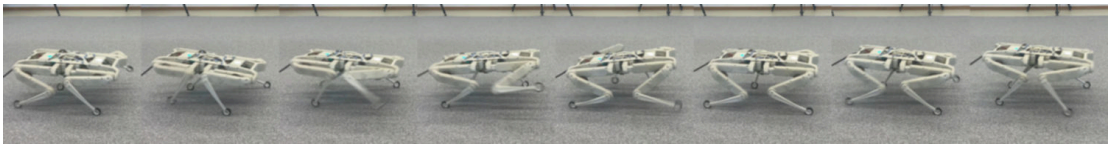


Figure E.3: Behavior of Lagrange multipliers. (a) Evolution of $\sigma(\lambda_z)$ for one skill ($z = 1$ chosen arbitrarily), (b) violation of the constraint for different ε. Negative $\phi_z - \varepsilon$ indicates no violation. Means and standard deviation across restarts.



(a) Trot locomotion with wave trunk motion and low base height.

(b) Trot locomotion with middle base height.

(c) Trot locomotion with high base height.

Figure E.4: Snapshots of the trained policy exhibiting distinct skills on hardware. From above to bottom, the policy has low, middle and high base positions while moving forward.

space as the agent. Note that the agent still observes its full state s, however the projected state $\Pi(s)$ is observed by the expert classifier and skill discriminator. The projection Π can be selected to specify which parts of the state we want to diversify and constrain in terms of occupancy, depending on the task at hand.

E.12 Limitations

The DOI method also comes with certain caveats. Maximizing the mutual information, as a diversity objective, poses a hard optimization problem due to its convexity. Thus, designing alternative diversity objectives can be beneficial. Furthermore, closeness in state-action occupancy can be quite restrictive in terms of availability of diverse behaviors that satisfy the constraint. Replacing this with constraints on the return of the policy would allow more freedom to optimize diversity in cases where the optimal policy may be multimodal. The above challenges are promising directions for future work.

E.13 Robust Obstacle Navigation

When the expert data is multi-modal, some modes might be more robust to distribution shift than others. However, using a uni-modal algorithm such as SMODICE, which tries to match the expert's state occupancy distribution, may not result in a robust policy. In contrast, each learned DOI skill recovers a particular mode, and as shown in this experiment, at least one DOI skill is robust against a distribution shift.

We consider the task of navigating across a box obstacle to a target position behind it, for the Solo12 robot. For training the DOI skills, we choose the feature vector $\phi(s)$ with linear and angular velocity as input to the skill-discriminator $q(z|\phi(s))$. The agent used to collect the *expert dataset* can go over or around the box obstacle from the left or right side to reach the target position in the traversable obstacle terrain. The box has a height of 0.2 meters and a square size of 1.0×1.0 meters. As a result, the collected expert data is multi-modal and consists of trajectories over and from the sides of the box obstacle to the target position.

It is important to emphasize that the less direct route to the target position (left or right side of the box) is always the more robust choice, since the agent runs into the risk of slipping or falling while climbing the box. We evaluate the learned DOI skills and SMODICE expert on 6 different heights: $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$ meters. The $\{0.3, 0.4, 0.5, 0.6\}$ meters boxes are out-of-distribution and increasingly difficult to traverse from above the box. In Figure E.5, we observe the trajectory distributions of the DOI skills and SMODICE expert collected in simulation. The arrows indicate the yaw angle of the robot at the trajectory points.

As we can see from the return distributions in Figure E.7, the performance of the SMODICE expert is strongly affected by the height of the box, as it is biased towards climbing over the box (this also depends on the initial state of the agent), which becomes increasingly difficult and may not be feasible. This can be observed from the trajectory distribution shown in the right-most column of Figure E.5; the trajectories of the SMODICE expert become increasingly concentrated in front of the box as its height increases. On the other hand, the three DOI skills (learned with a fixed Lagrange multiplier $\sigma(\mu) = 0.5$) recover diverse behaviors and robustly reach the goal. Here it is DOI-Skill 3, which is the most robust in reaching the target position and gives the highest return (see Figure E.5 and Figure E.7).

In Figure E.5, each row corresponds to a box with a fixed height $H \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$ meters. Each of the first three columns is associated with a fixed DOI skill (red, green, and blue) and the last column represents the SMODICE expert. Each cell shows every 10th step of 60 randomly initialized trajectories, all computed in simulation. This experiment demonstrates that although SMODICE expert is multimodal, it gets stuck in front of the box and fails to robustly reach the target position already at a box height of 0.4 meters. In contrast, the DOI-Skill 3 robustly reaches the target position by bypassing the box from

the left side. The fraction of randomly initialized trajectories stuck in front of the box is significantly smaller for the DOI-Skill 3 than the SMODICE expert. This is reflected in the return distribution shown in Figure E.7, which has the same row and column structure as Figure E.5.

E.14 Additional Experiments

Instead of learning the Lagrange multipliers λ_z via KL estimators ϕ_z, we can also fix λ_z at a certain level, making it a hyperparameter. In our setting, this also works well, and we demonstrate a tradeoff between diversity and task reward optimization, see Figures E.8 and E.9. However, in this case we lose the possibility to enforce a certain constraint on the KL-divergence between the skill state-action occupancy and expert state-action occupancy.

We further provide results of applying DOI to different levels of expert trajectory mix-in to the *medium-replay* and *random* datasets of WALKER2D and HALFCHEETAH in tables E.2 and E.3.

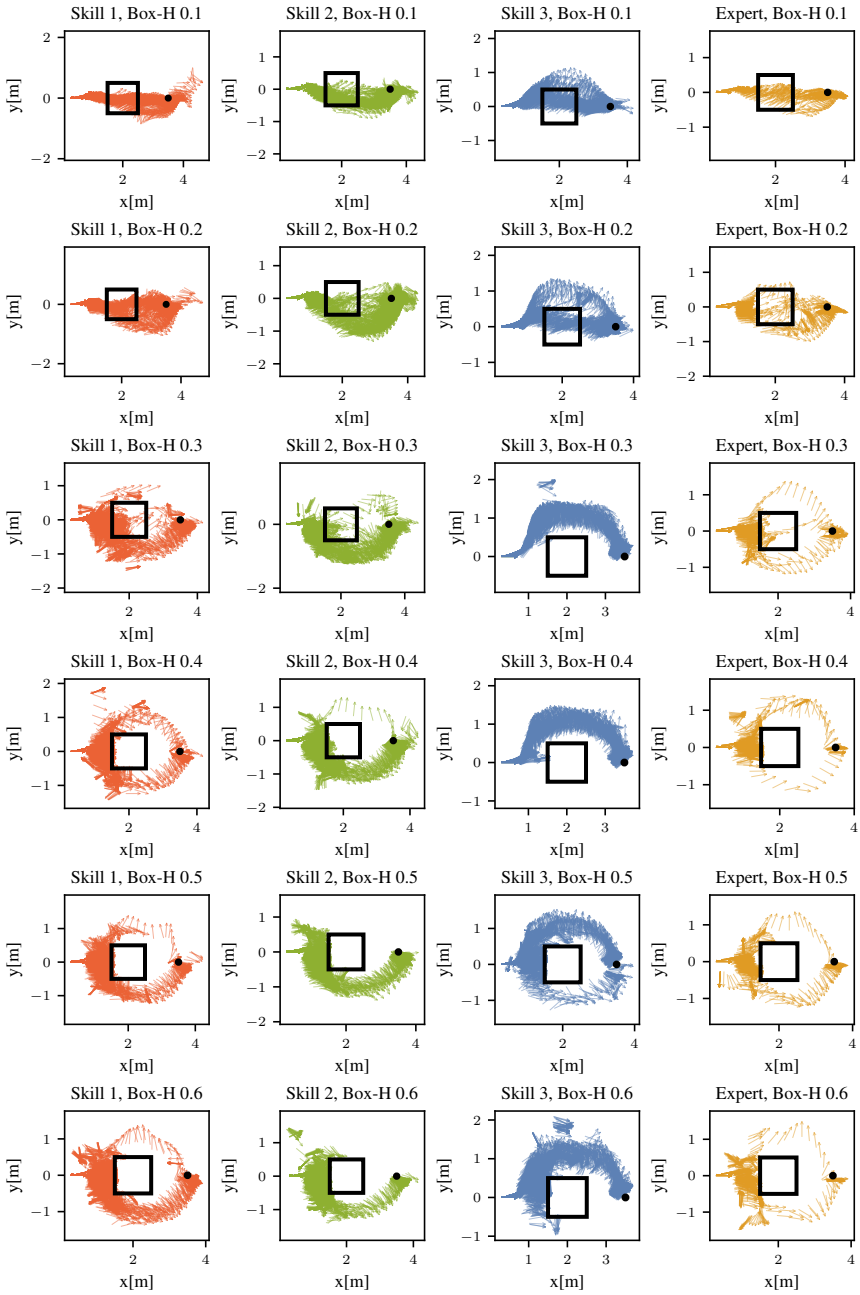


Figure E.5: A performance benchmark of the DOI skills and the SMODICE expert on an obstacle navigation task, where the Solo12 is initialized in front of a box and tries to reach a target position behind the box. The task consists of six levels of increasing difficulty depending on the height of the box.

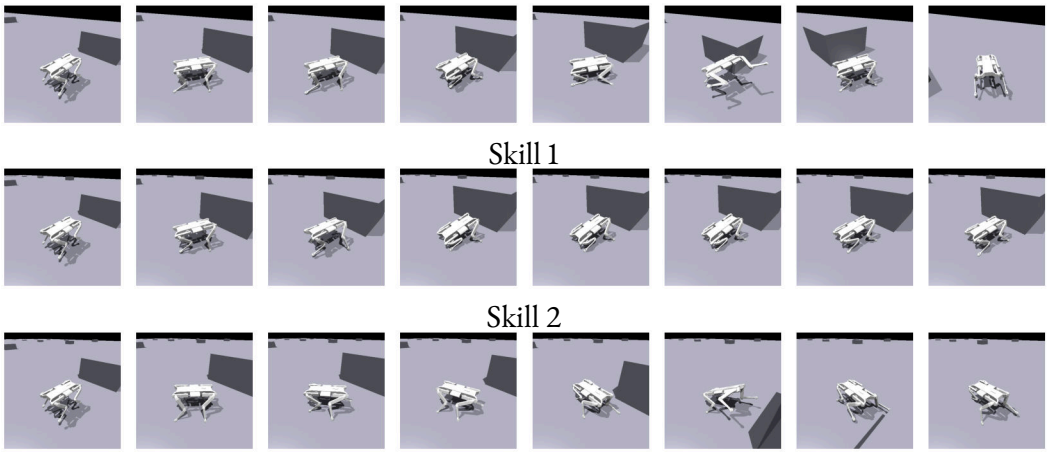


Figure E.6: Frames from rollout videos of the learned DOI skills for the highest box task, skills 1 and 3 go from the side of the boxes to the goal, and skill 2 reimains in front of the box since it mostly tries to climb it.

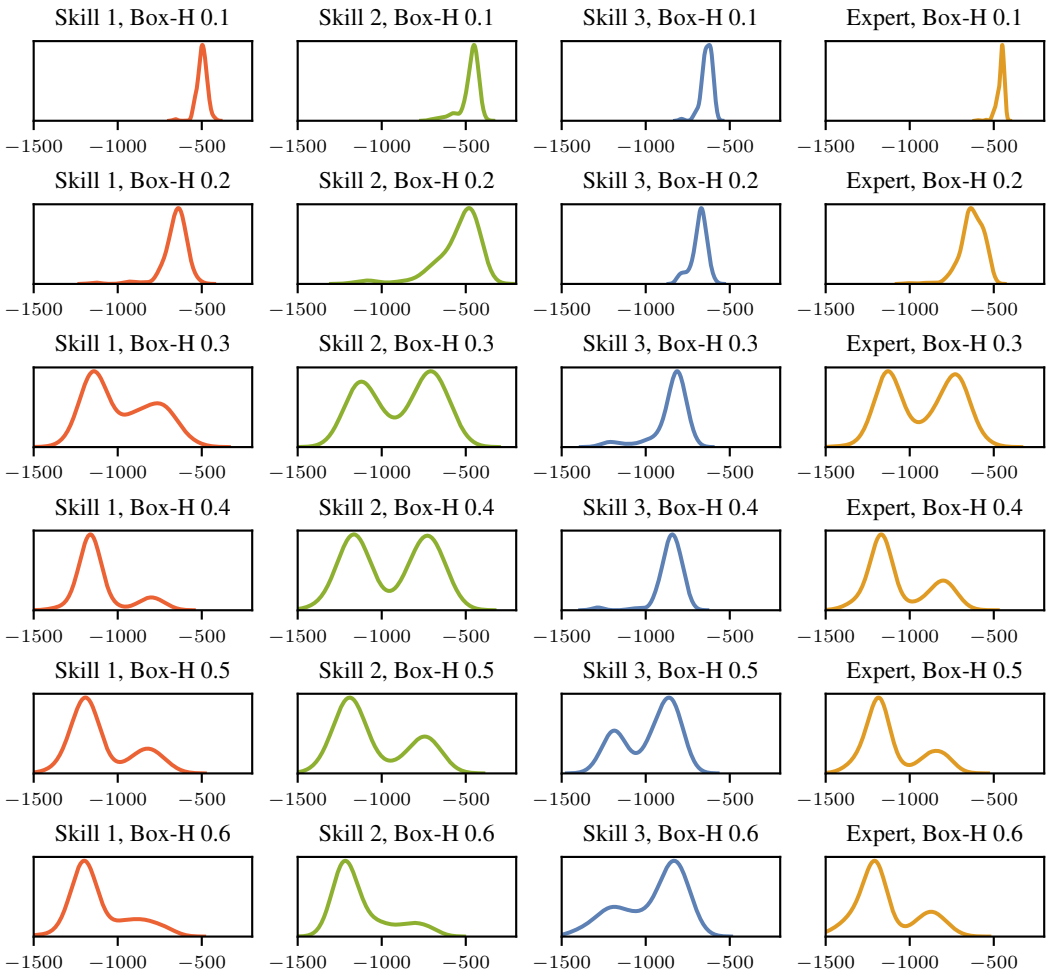
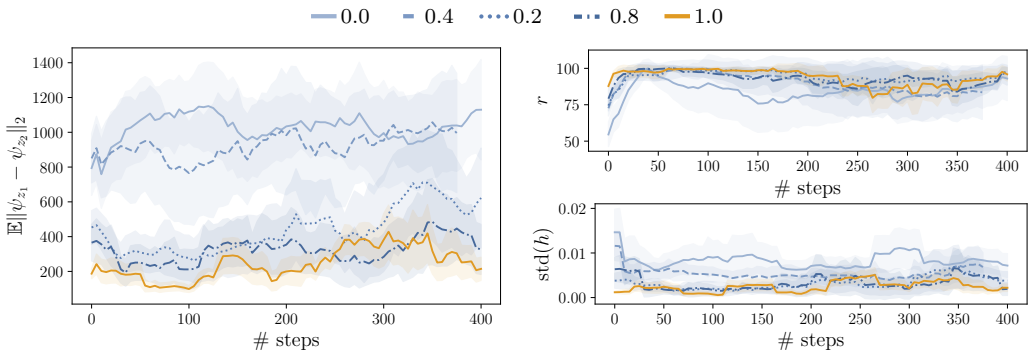
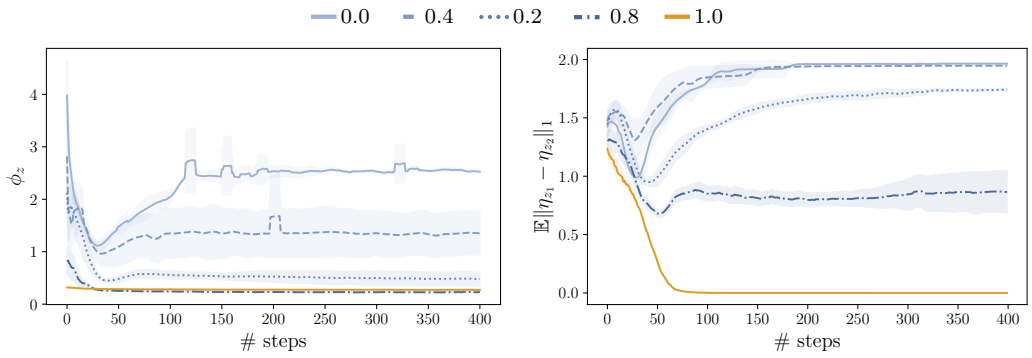


Figure E.7: Return distributions for DOI skills and SMODICE, we see in particular that the SMODICE policy return distribution is greatly affected by increasing the height of the box.



(a) **(b)**

Figure E.8: (a) Average ℓ_2 distance between Monte Carlo estimated successor representations ψ_z of distinct skills, (b) return r as % of expert return and standard deviation of base height $\text{std}_z(h)$, depending on a fixed $\sigma(\lambda_z)$ (see legend).



(a) **(b)**

Figure E.9: Divergence estimate and η_z distance for the case of fixed $\sigma(\lambda_z)$. (a) Value of divergence estimator ϕ_z for a specific skill over the course of training ($z = 1$ chosen arbitrarily), (b) average ℓ_1 distance of η_z's of skills. Means and standard deviation across restarts.

dataset	# expert mixin	ε	$\mathbb{E}\|\eta_{z_1} - \eta_{z_2}\|$	r	$\mathbb{E}\|\psi_{z_1} - \psi_{z_2}\|$
medium-replay	25	0.0	0.00 ± 0.00	46.00 ± 1.46	6.16 ± 0.30
		0.5	0.21 ± 0.08	0.33 ± 0.48	3.54 ± 2.14
		1.0	1.40 ± 0.05	2.33 ± 0.51	6.09 ± 2.40
		2.0	1.30 ± 0.03	0.64 ± 0.11	7.67 ± 4.27
		4.0	1.54 ± 0.08	2.30 ± 1.64	19.26 ± 2.29
	50	0.0	0.00 ± 0.00	54.29 ± 2.13	5.53 ± 0.14
		0.5	0.82 ± 0.28	31.31 ± 7.03	14.13 ± 1.86
		1.0	1.21 ± 0.15	4.33 ± 0.75	0.42 ± 0.05
		2.0	1.37 ± 0.03	1.61 ± 0.41	13.85 ± 2.50
		4.0	1.48 ± 0.12	1.11 ± 0.36	22.02 ± 1.33
	200	0.0	0.00 ± 0.00	98.33 ± 0.44	2.67 ± 0.26
		0.5	0.45 ± 0.11	74.59 ± 8.96	6.22 ± 1.17
		1.0	1.20 ± 0.09	2.52 ± 1.50	12.97 ± 4.33
		2.0	1.30 ± 0.03	2.07 ± 0.65	3.23 ± 2.02
		4.0	1.59 ± 0.06	1.43 ± 0.64	19.48 ± 1.43
random	25	0.0	0.00 ± 0.00	36.49 ± 11.54	15.70 ± 0.48
		0.5	0.93 ± 0.02	20.48 ± 7.90	16.81 ± 3.14
		1.0	1.30 ± 0.12	3.72 ± 1.38	8.16 ± 5.43
		2.0	1.45 ± 0.09	1.22 ± 0.32	20.47 ± 3.08
		4.0	1.27 ± 0.05	0.60 ± 0.26	20.60 ± 4.17
	50	0.0	0.00 ± 0.00	103.16 ± 0.69	3.32 ± 0.07
		0.5	1.03 ± 0.13	33.60 ± 6.64	18.27 ± 2.50
		1.0	1.37 ± 0.09	5.05 ± 2.66	20.16 ± 3.05
		2.0	1.46 ± 0.06	0.77 ± 0.29	10.46 ± 3.77
		4.0	1.23 ± 0.09	0.26 ± 0.11	14.33 ± 1.97
	200	0.0	0.00 ± 0.00	107.43 ± 0.26	1.84 ± 0.08
		0.5	1.29 ± 0.07	103.29 ± 1.38	6.75 ± 0.77
		1.0	1.26 ± 0.22	2.43 ± 0.30	7.30 ± 4.86
		2.0	1.46 ± 0.10	0.47 ± 0.15	15.39 ± 1.56
		4.0	1.29 ± 0.01	1.91 ± 0.57	19.66 ± 3.36

Table E.2: WALKER2D metrics across random and medium-replay variants with varying number of mixed-in trajectories of the expert to satisfy the coverage assumption.

dataset	# expert mixin	ε	$\mathbb{E}\|\eta_{z_1} - \eta_{z_2}\|$	r	$\mathbb{E}\|\psi_{z_1} - \psi_{z_2}\|$
medium-replay	25	0.0	0.00 ± 0.00	37.64 ± 0.30	3.22 ± 0.06
		0.5	0.83 ± 0.12	36.95 ± 0.63	3.02 ± 0.10
		1.0	1.36 ± 0.09	24.30 ± 6.28	13.34 ± 4.84
		2.0	1.44 ± 0.06	6.73 ± 3.65	22.09 ± 8.15
		4.0	1.27 ± 0.09	2.68 ± 0.72	21.68 ± 1.87
	50	0.0	0.01 ± 0.01	45.40 ± 0.22	3.26 ± 0.27
		0.5	1.14 ± 0.02	42.89 ± 0.19	2.94 ± 0.12
		1.0	1.41 ± 0.12	37.28 ± 2.41	6.18 ± 1.21
		2.0	1.32 ± 0.11	8.60 ± 4.66	13.66 ± 1.97
		4.0	1.24 ± 0.16	1.72 ± 0.18	28.74 ± 7.84
	200	0.0	0.00 ± 0.00	73.60 ± 0.39	3.65 ± 0.09
		0.5	1.16 ± 0.08	69.91 ± 1.14	3.67 ± 0.10
		1.0	1.28 ± 0.13	23.74 ± 12.94	13.47 ± 1.73
		2.0	1.49 ± 0.10	15.52 ± 4.29	32.03 ± 0.56
		4.0	1.42 ± 0.07	2.16 ± 0.04	11.92 ± 2.28
random	25	0.0	0.00 ± 0.00	2.80 ± 0.36	5.55 ± 1.18
		0.5	1.12 ± 0.04	3.03 ± 0.28	4.30 ± 0.85
		1.0	1.14 ± 0.12	2.24 ± 0.09	10.45 ± 3.30
		2.0	1.24 ± 0.08	1.73 ± 0.33	25.01 ± 8.78
		4.0	1.44 ± 0.03	1.60 ± 0.30	35.08 ± 8.27
	50	0.0	0.00 ± 0.00	31.89 ± 1.14	9.97 ± 0.58
		0.5	1.14 ± 0.11	10.29 ± 3.13	17.90 ± 6.01
		1.0	1.42 ± 0.15	6.45 ± 2.95	23.30 ± 0.96
		2.0	1.41 ± 0.08	2.73 ± 0.43	23.91 ± 6.98
		4.0	1.68 ± 0.06	1.44 ± 0.27	35.07 ± 8.08
	200	0.0	0.00 ± 0.00	68.35 ± 1.25	5.20 ± 0.31
		0.5	1.30 ± 0.08	50.85 ± 17.30	9.80 ± 3.68
		1.0	1.21 ± 0.12	15.06 ± 5.58	29.57 ± 4.26
		2.0	1.03 ± 0.10	2.10 ± 1.99	10.84 ± 7.57
		4.0	1.20 ± 0.20	2.16 ± 0.05	16.90 ± 5.95

Table E.3: HalfCheetah metrics across random and medium-replay variants with varying number of mixed-in trajectories of the expert to satisfy the coverage assumption.