
Learning to Search:

Structured Prediction Techniques for Imitation Learning

NATHAN D. RATLIFF

CMU-RI-TR-09-19

THE ROBOTICS INSTITUTE
CARNEGIE MELLON UNIVERSITY
PITTSBURGH, PENNSYLVANIA 15213

May 2009

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

Thesis committee:

J. Andrew Bagnell, Chair
Geoffrey Gordon
Siddhartha Srinivasa
James Kuffner
Andrew Ng, Stanford University

© Nathan Ratliff MMIX

Abstract

Modern robots successfully manipulate objects, navigate rugged terrain, drive in urban settings, and play world-class chess. Unfortunately, programming these robots is challenging, time-consuming and expensive; the parameters governing their behavior are often unintuitive, even when the desired behavior is clear and easily demonstrated. Inspired by successful end-to-end learning systems such as neural network controlled driving platforms (Pomerleau, 1989), learning-based “programming by demonstration” has gained currency as a method to achieve intelligent robot behavior. Unfortunately, with highly structured algorithms at their core, modern robotic systems are hard to train using classical learning techniques. Rather than redefining robot architectures to accommodate existing learning algorithms, this thesis develops learning techniques that leverage the performance of modern robotic components.

We begin with a discussion of a novel imitation learning framework we call Maximum Margin Planning which automates finding a cost function for optimal planning and control algorithms such as A*. In the linear setting, this framework has firm theoretical backing in the form of strong generalization and regret bounds. Further, we have developed practical nonlinear generalizations that are effective and efficient for real-world problems. This framework reduces imitation learning to a modern form of machine learning known as Maximum Margin Structured Classification (Taskar *et al.*, 2005); these algorithms, therefore, apply both specifically to training existing state-of-the-art planners as well as broadly to solving a range of structured prediction problems of importance in learning and robotics.

In difficult high-dimensional planning domains, such as those found in many manipulation problems, high-performance planning technology remains a topic of much research. We close with some recent work which moves toward simultaneously advancing this technology while retaining the learnability developed above.

Throughout the thesis, we demonstrate our algorithms on a range of applications including overhead navigation, quadrupedal locomotion, heuristic learning, manipulation planning, grasp prediction, driver prediction, pedestrian prediction, optical character recognition, and LADAR classification.

Acknowledgements

I owe a deep debt of gratitude to my family and the many friends who have given me the opportunity to write this thesis. I am particularly grateful to my wife Ellie for keeping me sane when the work got intense, and to my parents for pointing me in the right direction early on. Most importantly, this thesis would not have been possible without the invaluable counsel of my advisor Drew, who remains a constant source of exciting and creative ideas. Thank you.

1	Introduction	6
1.1	A categorization of imitation learning techniques	7
1.2	Inverse optimal control	8
1.3	Reader’s guide to this thesis	10
1.4	Approaches	11
1.5	Taxonomy of MMP algorithms	12
1.6	Contributions	12
2	A Practical Overview of Imitation learning for Robotics	15
2.1	An implementational introduction to LEarning to seaRCH	16
2.2	Loss-augmentation	19
3	Subgradient Convex Optimization	23
3.1	Subgradients and strong convexity	26
3.1.1	Subgradient definition and properties	26
3.1.2	Strong convexity	27
3.1.3	Bounding the effective optimization radius	28
3.2	The online setting	29
3.2.1	Online regret	30
3.2.2	The unregularized case	32
3.2.3	Constant regularization	33
3.2.4	Attenuated regularization	35
3.3	From regret bounds to generalization bounds	38
3.4	The batch setting	42
3.4.1	Reductions to online learning	44
3.4.2	Batch convergence bounds	44
3.4.3	Traditional Analysis	46
4	Functional Gradient Optimization	49
4.1	Gradient descent through Euclidean function spaces	51
4.1.1	Euclidean functional gradient projections	52

4.1.2	Euclidean functional gradients as data sets	55
4.1.3	A generalized class of objective functions	56
4.1.4	Comparing functional gradients techniques	56
4.2	Generalizing exponentiated gradient descent to function spaces	61
4.2.1	Exponentiated functional gradient descent	61
4.2.2	Theoretical results	63
4.3	Functional bundle methods	65
4.3.1	L_2 functional regularization	66
4.3.2	The functional bundle	67
4.3.3	Optimizing the functional bundle	67
4.3.4	Experimental results	69
5	Maximum Margin Planning	71
5.1	Preliminaries	72
5.2	Reducing imitation learning to maximum margin structured classification	74
5.3	Optimizing the maximum margin planning objective	77
5.3.1	Computing the subgradient for linear MMP	77
5.3.2	An approximate projection algorithm for cost positivity constraints	79
5.4	Learning linear quadratic regulators	81
5.5	A compact quadratic programming formulation	82
5.6	Experimental validation	83
6	LEARCH: Learning to Search	87
6.1	The MMP functional	88
6.2	General setting	89
6.3	Intuition	90
6.4	A log-linear variant	91
6.4.1	Deriving the log-linear variant	91
6.4.2	Log-linear LEARCH vs linear MMP	93
6.5	Case study: Multiclass classification	94
6.5.1	Footstep prediction	94
6.5.2	Grasp prediction	98
6.6	MMPBOOST : A stage-wise variant	100
6.6.1	Overhead navigation	102
6.6.2	Training a fast planner to mimic a slower one	103
7	Maximum Margin Structured Classification	107
7.1	Maximum margin structured classification	110
7.1.1	Batch learning	111
7.1.2	Online learning	112
7.1.3	Subgradient computation	113
7.2	Theoretical results	114
7.2.1	Convergence bounds of batch learning	115
7.2.2	Sublinear regret of online learners	115
7.2.3	Generalization bounds	116

7.3	Robustness to approximate settings	117
7.3.1	Using approximate inference	117
7.3.2	Optimizing with approximate subgradients	118
7.4	Experimental results	119
7.4.1	Optical character recognition	119
7.4.2	LADAR scan classification	120
8	Maximum Margin Structured Regression	123
8.1	Motivation	123
8.2	Defining MMSR	124
8.3	Linear derivation and optimization	125
8.4	Computing functional gradients of MMSR	129
8.5	An application to value function approximation	130
9	Inverse Optimal Heuristic Control	132
9.1	Introduction	133
9.2	Inverse optimal heuristic control	134
9.2.1	Gibbs models for imitation learning	134
9.2.2	Combining inverse optimal control and behavioral cloning	135
9.2.3	Gradient-based optimization	136
9.3	On the efficient optimization of inverse optimal heuristic control	137
9.4	Convex approximations	141
9.4.1	The perceptron algorithm	142
9.4.2	Expert augmentation	142
9.4.3	Soft-backup modification	142
9.5	Experimental results	144
9.5.1	An illustrative example	144
9.5.2	Turn prediction for taxi drivers	145
9.5.3	Pedestrian prediction	147
9.6	Conclusions	148
10	Covariant Hamiltonian Optimization for Motion Planning	149
10.1	Introduction	150
10.2	The CHOMP Algorithm	151
10.2.1	Covariant gradient descent	152
10.2.2	Understanding the update rule	155
10.2.3	From gradient descent to Monte Carlo sampling	157
10.2.4	Obstacles and distance fields	161
10.2.5	Defining an obstacle potential	163
10.2.6	Functions vs functionals	165
10.2.7	Smooth projection for joint limits	166
10.3	Experiments on a robotic arm	167
10.3.1	Collision heuristic	168
10.3.2	Planning performance results	169
10.3.3	An empirical analysis of optimization initialization	170
10.4	Conclusions	172

11 Future directions	174
11.1 The role of reinforcement learning	174
11.2 Functional bundles for structured prediction	175
11.3 Theoretical understanding of functional gradient algorithms	176

Chapter 1

Introduction

Evidence in support of sophisticated planning techniques continues to build in robotics. The robotics literature contains increasingly sophisticated algorithms for efficient and intelligent long-range reasoning. Today's robots can navigate rugged terrain, drive in urban settings, manipulate household objects, and play world-class chess. Researchers often attribute these success stories to modern state-of-the-art planners that reason about long-term consequences of actions.

Then why aren't intelligent robots ubiquitous? The truth is it takes an expert to apply modern planning technology to new domains. The space of planning algorithms defined by most modern planning systems is effectively infinite-dimensional; it includes both planners that make good decisions as well as planners that make very bad decisions. Roboticists must navigate this space to find a single planner that performs well across a range of problems. While the desired behavior is often clear, manipulating a planner's parameters to implement that behavior can be an expensive process of trial and error.

Researchers frequently look to machine learning for fast and efficient tools to aid in developing behavior. Unfortunately, it is not clear how to train planners using traditional learning machines. This chapter outlines the subcategories of imitation learning, and describes in detail one class of imitation learning particularly applicable to training optimal control and planning algorithms by demonstration. In Section 1.3, we outline the organization of this thesis as a guide to readers of

varying interests. This thesis references a wide range of algorithms by name; Section 1.5 summarizes the relationships between these names for reference. Finally, a list of contributions offered by this thesis is presented in Section 1.6.

1.1 A categorization of imitation learning techniques

The robotics literature focuses on two primary categories of imitation learning techniques: *task-specific imitation learning* and *generalizable imitation learning*.

Task-specific imitation learning trains an agent to perform a single task well in a given domain. The literature in this area has seen exciting successes ranging from humanoid robots that learn to juggle (Schaal & Atkeson, 1993) to autonomous helicopters that learn acrobatic maneuvers and perform airshows (Abbeel *et al.*, 2007; Coates, Abbeel, & Ng, 2008). However, task-specific imitation learning focuses on only a single task; algorithms here often design controllers to reliably replay demonstrated trajectories. This thesis, on the other hand, concentrates on the second area of imitation learning which we term *generalizable imitation learning*. In contrast to task-specific imitation learning, this generalizable imitation learning focuses on generalizing demonstrations to new domains unseen during training.

We can further classify generalizable imitation learning into two subcategories. The first, which we term *behavioral cloning* (BC), is a straightforward reduction to supervised learning (Bain & Sammut, 1995). BC techniques train classifiers to map observations to actions. The ALVINN system (Pomerleau, 1989), for instance, achieved early success in a behavioral cloning form of generalizable imitation learning by training neural networks to drive an autonomous car across the country.

Although behavioral cloning applies to a range of hard problems, reducing imitation learning to reactive action classification relinquishes control over reasoning about action consequences. All information relevant to the current decision must be encoded in the collection of features representing the observation. Feature extraction, therefore, becomes almost as hard as designing the behavior itself. The second form of generalizable imitation learning is known as *inverse optimal control* (IOC). This form offers an alternative approach by training optimal control algorithms to

reason over sequences of decisions in a way that generalizes demonstrations (Boyd *et al.*, 1994). Reasoning over action sequences is crucial both theoretically and empirically. IOC models perform strongly on problems ranging from driver behavior modeling and route prediction (Abbeel & Ng, 2004; Ziebart *et al.*, 2008a) to legged locomotion and autonomous navigation (Ratliff *et al.*, 2006; Ratliff, Bagnell, & Zinkevich, 2006; Silver, Bagnell, & Stentz, 2008; Kolter, Abbeel, & Ng, 2008). This thesis develops its ideas around this second form of imitation learning.

1.2 Inverse optimal control

Inverse optimal control was first proposed—and solved for one-dimensional inputs in linear systems with quadratic costs—by Kalman (1964); solutions of increasing generality were developed over the following years (Anderson & Moore, 1990), culminating in the work of Boyd *et al.* (1994), who generalized IOC to the linear control setting. Under the name *inverse reinforcement learning* (IRL), Ng & Russell (2000) brought renewed interest at the turn of the century by exploring the problem in terms of discrete Markov decision processes (MDPs). The goal, by their definition, was to learn a cost function for the MDP under which the demonstrated policy is optimal. The authors note, however, that their definition is ill-posed; many cost functions may display this property.¹ Early algorithms proposed by the authors work around these issues using specialized heuristics.

In 2004, Abbeel & Ng (2004) observed that IRL can be reformulated in terms of the *cumulative feature counts* observed by a policy² when the costs are linear functions of the features. By linearity, when the cumulative feature counts of two policies match, the cumulative costs match as well. This observation became the formal objective of a new algorithm for IRL now known as *apprenticeship learning*.

While this reformulation improves on the original definition, there may still exist many policies that collect the same feature counts. Additionally, by formulating their algorithm around linearity, the authors preclude many straightforward extensions to nonlinear hypotheses. Perhaps more importantly, this formulation offers no connection between the demonstrated behavior and the

¹For instance, every policy is optimal when the cost function is zero everywhere.

²Abbeel & Ng (2004) define the cumulative feature count as the expected number of times a feature is encountered while running the policy.

recovered behavior; matching cost functions does not necessarily imply successful imitation.

Our work offers the first well-formed general solution to inverse optimal control for MDPs (Ratliff, Bagnell, & Zinkevich, 2006) by reducing IOC to a new form of structured prediction in machine learning known as *maximum margin structured classification* (MMSC) (Taskar, Guestrin, & Koller, 2003). We call our framework *maximum margin planning* (MMP). Our formulation derives a strictly convex regularized risk function to govern the learning process. This objective function upper bounds a well-defined notion of loss between policies, and strict convexity guarantees that a single, globally optimal cost function resides at the minimum. In (Ratliff, Bagnell, & Zinkevich, 2006), we proposed a class of optimization procedures that efficiently implement learning and lead to the first online regret and batch generalization results for IOC.

While the original linear implementations of our framework hold feature matching interpretations similar to IRL, by explicitly defining learning in terms of optimization, MMP opens a door to important nonlinear generalizations (Ratliff *et al.*, 2006; Ratliff, Silver, & Bagnell, 2009) that demonstrate strong practical performance on real-world problems. These algorithms were the first general IOC solutions to be successfully used on real-world robotic platforms (Ratliff *et al.*, 2006; Silver, Bagnell, & Stentz, 2008). They have been applied to a wide variety of problems including footstep prediction, grasp prediction, heuristic learning, and overhead navigation (Ratliff, Srinivasa, & Bagnell, 2007; Ratliff *et al.*, 2006; Silver, Bagnell, & Stentz, 2008). Moreover, MMP problems are large even relative to other structured prediction problems; the optimization procedures that implement MMP offer maximum margin structured classification a new class of rapidly converging and memory efficient optimization techniques with concomitant generalization and regret bounds. These subgradient and functional gradient approaches to structured learning demonstrate state-of-the-art performance on a diverse set of problems ranging from optical character recognition to LADAR point cloud classification (Ratliff, Bagnell, & Zinkevich, 2007a; Munoz, Vandapel, & Hebert, 2008; Munoz *et al.*, 2009).

More recently, newer formulations of IOC have been developed as interest in this body of work continues to grow. Returning to the feature matching formulation, Ziebart *et al.* (2008a) derive a maximum entropy formalization of the problem (*MaxEnt* IOC) that retains strict convexity in

its governing objective function. Specifically, the authors define a stochastic policy in terms of the distribution it generates over trajectories through the MDP. Given demonstrated trajectories, the authors maximize the entropy of this distribution subject to the constraint that the expected cumulative feature counts of the distribution match those of the expert’s policy. Both this technique and IRL address the same class of policies, but MaxEnt IRL places a principled ordering across equivalence classes of policies with matching feature counts. The authors’ introductory and application papers (Ziebart *et al.*, 2008a,c) demonstrate this algorithm on driver route prediction problems.

In the latter half of this thesis, we outline efforts to combine behavioral cloning with inverse optimal control through a new model we call *inverse optimal heuristic control* (IOHC). These techniques were strongly influenced by new research into Gibbs models for IOC. In both (Neu & Szepesvari, 2007) and (Ramachandran & Amir, 2007), the authors utilize the Gibbs model as a smooth approximation to the hypothesized policy in order to utilize a variety of loss functions under IOC. IOHC, in conjunction with recent work in *covariant Hamiltonian optimization for motion planning* (CHOMP) for framing high-dimensional motion planning as optimization (Ratliff *et al.*, 2009b), studies the use of IOC techniques in imitation learning settings where optimal control may be intractable.

1.3 Reader’s guide to this thesis

This thesis opens by introducing inverse optimal control in this chapter and continues with an intuitive discussion of one of our IOC algorithms in Chapter 2 that emphasizes the simplicity of its implementation. The algorithm is a particular implementation of our maximum margin planning (MMP) framework using the functional gradient techniques. MMP and the class of its gradient-based implementations known as LEArning to seaRCH (LEARCH) are developed, respectively, in Chapters 5 and 6. Chapters 3 and 4 present the formal analysis of these optimization procedures. These two theoretical chapters are highly technical; the reader may wish to skim these chapters or skip directly to Chapter 5 on first reading.

The MMP framework defines a reduction from IOC to a form of structured prediction known

as maximum margin structured classification (MMSC). Chapter 7 analyzes the linear subclass algorithms forming LEARCH within the general context of MMSC. A new form of structured prediction generalizing traditional ϵ -insensitive support vector regression techniques is then derived and analyzed in Chapter 8.

Later, Chapters 9 and 10 move beyond the class of problems addressed by MMP to focus on high-dimensional problems where optimal control is impractical or intractable. Chapter 9 presents a class of algorithms called *inverse optimal heuristic control* (IOHC) designed to solve problems where dynamics such as velocities and accelerations along the trajectory may significantly affect the policy, and Chapter 10 derives a novel high-dimensional motion planning algorithm called *covariant Hamiltonian optimization for motion planning* (CHOMP) that addresses high-dimensional configuration spaces.

We close the thesis with Chapter 11 where we discuss open problems in IOC and future directions for our research.

1.4 Approaches

Throughout this thesis, we focus on the idea of reducing inverse optimal control to maximum margin structured classification and solving the resulting optimization problem using generalized gradient-based optimization techniques. Our algorithms train optimal controllers to mimic the behavior demonstrated in a collection of examples presented as decision sequences. These learning routines include traditional parametric gradient descent procedures and contemporary functional gradient variants that enjoy fast convergence, small memory requirements, and strong theoretical guarantees.

Two chapters in this thesis deviate from these general rules. In Chapter 9, we define a class of imitation learning algorithms that combines concepts in IOC and BC. This idea derives a hybrid model that opens structured prediction to more general sources of information. Additionally, Chapter 10 derives and empirically analyzes a novel motion planning algorithm designed to address high-dimensional manipulation problems where the (approximately) optimal inference required by MMSC is not possible. This algorithm is designed to maneuver motion planning toward a setting

that more naturally fits within the IOC learning framework we outline in this thesis.

1.5 Taxonomy of MMP algorithms

The term *maximum margin planning* (MMP) is used throughout the thesis to reference the reduction of IOC to *maximum margin structured classification* (MMSC). In particular, we refer to the regularized risk function (the generalized hinge loss) that governs learning under this reduction as the *MMP objective*. We may additionally refer to this setting as the *MMP framework*.

The term *LEARNING to seaRCH* (LEARCH) refers to a collection of generalized gradient-based optimization algorithms that implement learning under MMP. In particular, all optimization algorithms within LEARCH apply to the primal form of the MMP objective. This distinguishes LEARCH from alternative optimization algorithms in the literature which optimize within the dual space of the problem (e.g. see (Taskar, Guestrin, & Koller, 2003; Bartlett *et al.*, 2004; Taskar *et al.*, 2005; Taskar, Lacoste-Julien, & Jordan, 2006)).

We further categorize LEARCH into a number of subclasses which we reference throughout the thesis. *Linear LEARCH* is the class of primal optimization routines designed around the subgradient method used in the original implementation of MMP (Ratliff, Bagnell, & Zinkevich, 2006); these algorithms apply specifically to linear hypotheses. The term LEARCH without qualification typically refers to the functional gradient generalizations of the algorithms in linear LEARCH. We refer to our novel class of exponentiated functional gradient optimization procedures for MMP as *exponentiated LEARCH*. Using linear regression to implement the functional gradient approximation step in exponentiated LEARCH leads to a novel closed form subgradient method that performs updates in log-space. We call this algorithm *log-linear LEARCH*.

1.6 Contributions

This section lists the contributions of the work presented in this thesis.

1. **Maximum margin planning (MMP)**. Originally published in (Ratliff, Bagnell, & Zinkevich, 2006), this framework reduces a large class of inverse optimal control problems to a form

of structured prediction known as maximum margin structured classification. MMP can be viewed as an objective function governing learning. Chapter 5 derives this framework in full.

2. **LEarning to seaRCH (LEARCH).** This collection of algorithms is a class of generalized gradient based methods used to implement learning under the MMP reduction. An overview of this class of algorithms is given in (Ratliff, Silver, & Bagnell, 2009); the original linear variants of LEARCH were described and analyzed in (Ratliff, Bagnell, & Zinkevich, 2006). The first nonlinear variant was presented in (Ratliff *et al.*, 2006) and applications of a novel exponentiated functional gradient algorithm within this class were first published in (Ratliff, Srinivasa, & Bagnell, 2007). Chapters 4 and 6 detail this work.
3. **Gradient-based approaches to maximum margin structured classification.** Our gradient-based optimization routines that form LEARCH apply broadly to the encompassing class of MMSC problems. We demonstrate faster convergence and better generalization across a collection of standard structured prediction problems (Ratliff, Bagnell, & Zinkevich, 2007a). These results are presented in Chapter 7.
4. **Functional gradient optimization procedures.** We introduce two novel functional gradient optimization procedures in this thesis. The first of these procedures is the exponentiated functional gradient descent algorithm which has been used extensively across numerous imitation learning applications as part of LEARCH. The second procedure is a generalization of bundle methods (Smola, Vishwanathan, & Le., 2008) which we call functional bundle methods. These functional bundle methods promise fast optimization with compact representation. These approaches are derived and discussed in Chapter 4.
5. **Theoretical results for structured prediction.** In (Ratliff, Bagnell, & Zinkevich, 2006), we introduced a collection of theoretical results for a linear LEARCH implementation of MMP, including proofs of fast convergence in the batch setting, regret bounds for online learning, and batch generalization bounds. We generalized these results in (Ratliff, Bagnell, & Zinkevich, 2007a) and provided additional results indicating robustness to approximate inference. This theoretical analysis spans Chapters 3, 7, and 8.

6. **Maximum margin structured regression.** In (Ratliff, Bagnell, & Zinkevich, 2007a), we additionally introduced and analyzed a novel structured prediction framework that generalizes ϵ -insensitive support vector regression techniques (Smola & Schölkopf, 2003) to the structured setting. Chapter 8 presents this material and further generalizes it to the functional setting.
7. **Inverse optimal heuristic control.** We introduced and analyzed a framework for combining behavioral cloning and inverse optimal control techniques in (Ratliff *et al.*, 2009a). This setting is nonconvex, but we prove and empirically demonstrate that it is quantifiably *almost* convex. We detail this work in Chapter 9.
8. **Covariant Hamiltonian optimization for motion planning.** In (Ratliff *et al.*, 2009b), we introduce a novel motion planning algorithm that reduces high-dimensional motion planning to optimization. This algorithm relaxes the collision-free precondition assumed by most trajectory optimizers, and for many problems it removes the need for a separate randomized planning algorithm. By reducing motion planning to optimization, we facilitate the application of tools from MMP and LEARCH to this class of higher-dimensional problems. This material is presented in Chapter 10.

Chapter 2

A Practical Overview of Imitation learning for Robotics

Programming modern robots is hard. Roboticists often understand intuitively how the robot should behave, but uncovering a set of parameters for a modern system that embodies that behavior can be time consuming and expensive. When programming a robot’s behavior, researchers often adopt an informal process of repeated guess-and-check. For a skilled practitioner, this process borders on algorithmic. Imitation learning studies the algorithmic formalization for programming behavior by demonstration. Since many robot control systems are defined in terms of optimization (such as those designed around optimal planners), imitation learning can be modeled as the process of finding optimization criteria that make the expert look optimal. This intuition is formalized by maximum margin planning (MMP). At a high level, MMP may be viewed as an objective function measuring the suboptimality of the expert’s policy. Optimizing this objective, therefore, attempt to find an optimization criterion under which the policy looks optimal.

MMP arises through a reduction of imitation learning to a form of machine learning called structured prediction. Structured prediction studies problems in which making multiple predictions simultaneously can improve accuracy. The term “structure” refers to the relationships among the predictions that make this improvement possible. For instance, attempting to predict independently whether individual states occur along an optimal path in a graph has little hope of success without accounting for the connectivity of the graph. Robotics researchers naturally exploit this



Figure 2.1: Imitation learning applies to a wide variety of robotic platforms. This figure shows a few of the robots on which the imitation learning algorithms discussed here have been implemented. From left to right, we have (1) an autonomous ground vehicle build by the National Robotics Engineering Center (NREC) known as Crusher, (2) Boston Dynamics’s LittleDog quadrupedal robot, and (3) Barrett Technologies’s WAM arm, wrist, and 10-DOF hand.

connectivity when developing efficient planning algorithms. By reducing imitation learning to structured prediction, we leverage this body of work within learning to capture the problem’s structure and improve prediction. Algorithms that solve MMP imitate by learning to predict the entire sequence of actions that an expert would take toward a goal.

2.1 An implementational introduction to LEarning to searCH

The core algorithm forming the basic approach to imitation learning we advocate in this thesis is sufficiently intuitive that in this section we describe it first as it might be practically implemented. This algorithm is part of the LEarning to searCH class of algorithms we present in Chapter 6.

Let $\mathcal{D} = \{(\mathcal{M}_i, \xi_i)\}_{i=1}^N$ denote a set of examples, each consisting of an MDP $\mathcal{M}_i$ (excluding a specific reward function) and an example trajectory ξ_i between start and goal points. Figure 2.2 visually depicts the type of training data we consider here for imitation learning problems. Often, we can think of an example as a path between a pair of end points. Each MDP is imbued with a feature function that maps each state-action pair (s, a) in the MDP to a feature vector $f_i^{sa} \in \mathbb{R}^d$. This feature vector represents a set of d sensor readings (or quantities derived from sensor readings) that distinguish one state from the next.

For clarity, we consider here only the deterministic case in which the MDP can be viewed as a directed graph (states connected by actions). Planning between a pair of points in the graph

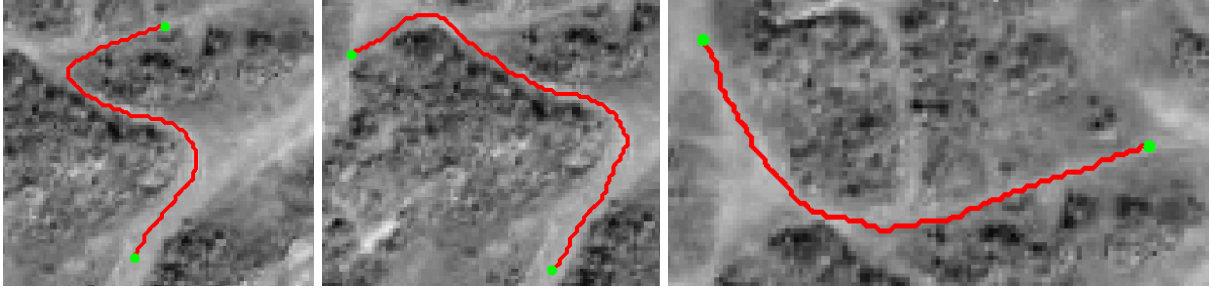


Figure 2.2: This figure demonstrates the flavor of the training data considered here in the context of imitation learning. In this case, a human expert specified by hand examples of the path (red) that a mobile robot should take between pairs of end points (green) through overhead satellite images. These example paths demonstrate the form of training data used for the outdoor navigational planning setting discussed in this chapter.

can be implemented efficiently using combinatorial planning algorithms such as Dijkstra or A*. In this setting, it is common to consider costs rather than rewards. Intuitively, a cost can be viewed simply as a negative reward; the planner must minimize the cumulative cost of the plan rather than maximize the cumulative reward. For the moment, we ignore positivity constraints on the cost function required by many combinatorial planners such as A*, but we will address these constraints formally later on. The formal derivation of MMP is presented in terms of a more general class of policies (see Chapter 5), and Chapter 9 defines an alternative model designed to explicitly account for stochasticity.

Intuitively, LEARCH iteratively refines the cost function $c : \mathbb{R}^d \rightarrow \mathbb{R}$ in order to make the example trajectories appear optimal. Since there is a feature vector associated with each state-action pair, a cost map (i.e. a mapping from state-action pairs to costs) can be generated for each $\mathcal{M}_i$ by evaluating the cost function at each state-action feature vector f_i^{sa} . Given the cost map, any black box deterministic planning algorithm can be run to determine the optimal path. Since the example path ξ_i is a valid path, the minimum cost path returned by the planning algorithm will usually have lower cost. In essence, the goal of the learning algorithm is to find a cost function for which the example path is the minimum cost path. The gap between the cost of the example path and the cost of the minimum cost path, therefore, acts as a quantitative measure of suboptimality.

During each iteration, LEARCH suggests local corrections to the cost function to progress toward minimizing this gap. In particular, the algorithm suggests that the cost function be increased

in regions of the feature space encountered along the planned path, and decreased in regions of the feature space encountered along the example path.¹

Specifically, for each example i , the algorithm considers two paths: the example path ξ_i , and the planned path $\xi_i^* = \arg \min_{\xi \in \Xi_i} \sum_{(sa) \in \xi} c(f_i^{sa})$. In order to decrease the difference between the example path's cost and the planned path's cost, the algorithm needs to modify the cost function so that the cost of the planned path increases and the cost of the example path decrease. For each path, the path cost is simply a sum of state-action costs encountered along the way, which are each generated by evaluating the cost function at the feature vector f_i^{sa} associated with that state-action pair. The algorithm can, therefore, raise or lower the cost of this path incrementally simply by increasing or decreasing the cost function at the feature vectors encountered along the path.

Many planning algorithms, such as A*, require strictly positive costs in order to ensure the existence of an admissible heuristic. We can accommodate these positivity constraints by making our modifications to the log of the cost function and exponentiating the resulting log-costs before planning. Intuitively, since the exponential enforces positivity, decreasing the log-cost function in a particular region simply pushes it closer toward zero.

We can write this algorithm succinctly as depicted in Algorithm 1. We will see in Chapter 6 that this rather intuitive algorithm implements an exponentiated variant of functional gradient descent. Figure 2.3 depicts an iteration of this algorithm pictorially. The final step in which we raise or lower the cost function (or the log-cost function) in specific regions of the feature space is intentionally left vague at this point. The easiest way to implement this step is to find a regression function that is positive in regions of the feature space where we want the cost function to increase, and negative in regions where we want the function to decrease. We can find such a function by specifying for each feature vector f_i^{sa} under consideration a label of either $+1$ or -1 , indicating whether we want the function to be raised or lowered in that region. Given this data set, we can use any of a number of out-of-the-box regression algorithms to learn a function with the desired property.

¹Below we show that the accumulation of these corrections minimizes an objective function that measures the error on our current hypothesis.

Algorithm 1 LEARCH intuition

```
1: procedure LEARCH( training data  $\{(\mathcal{M}_i, \xi_i)\}_{i=1}^N$ , feature function  $f_i$  )
2:   while not converged do
3:     for each example  $i$  do
4:       Evaluate the cost function at each state-action feature vector  $f_i^{sa}$  for MDP  $\mathcal{M}_i$  to
         create the cost map  $c_i^{sa} = c(f_i^{sa})$ .
5:       Plan through the cost map  $c_i^{sa}$  to find the minimum cost path  $\xi_i^* =$ 
          $\arg \min_{\xi \in \Xi_i} \sum_{(s,a) \in \xi} c_i^{sa}$ .
6:       Increase the (log-)cost function at the points in the feature space encountered along
         the minimum cost path  $\{f_i^{sa} \mid \forall (s,a) \in \xi_i^*\}$ , and decrease the (log-)cost function
         at points in the feature space encountered along the example path  $\{f_i^{sa} \mid \forall (s,a) \in$ 
          $\xi_i\}$ .
7:     end for
8:   end while
9: end procedure
```

2.2 Loss-augmentation

Most students can attest that learning on difficult problems makes simple problems easier. In this section, we use this intuition to devise a simple modification to the algorithm discussed in Section 2.1 that greatly improves generalization both in theory and in practice. Chapter 5 builds a formal interpretation of the resulting algorithm in terms of margin-maximization. Indeed, if we break from the traditional view, all margin-based learning techniques, such as the support vector machine, have a similar interpretation.

Surprisingly, a simple modification to the cost map inserted immediately before the planning step is sufficient to inject a notion of margin into the algorithm. Intuitively, this cost map augmentation makes it more difficult for the planning algorithm to return the example path by making alternative paths look more desirable. Applying this handicap during training forces the algorithm to continue updating the cost function until the demonstrated path ξ_i appears significantly more desirable than alternative paths. Specifically, we lower the cost of undesirable state-action pairs to make them more likely to be chosen by the planner during training. With this augmentation, even if the example path is currently the minimum cost path through the actual cost map, it may not be the minimum cost path through the augmented cost map.

In order to solidify this concept of undesirable state-action pair, we define what we call a *loss*

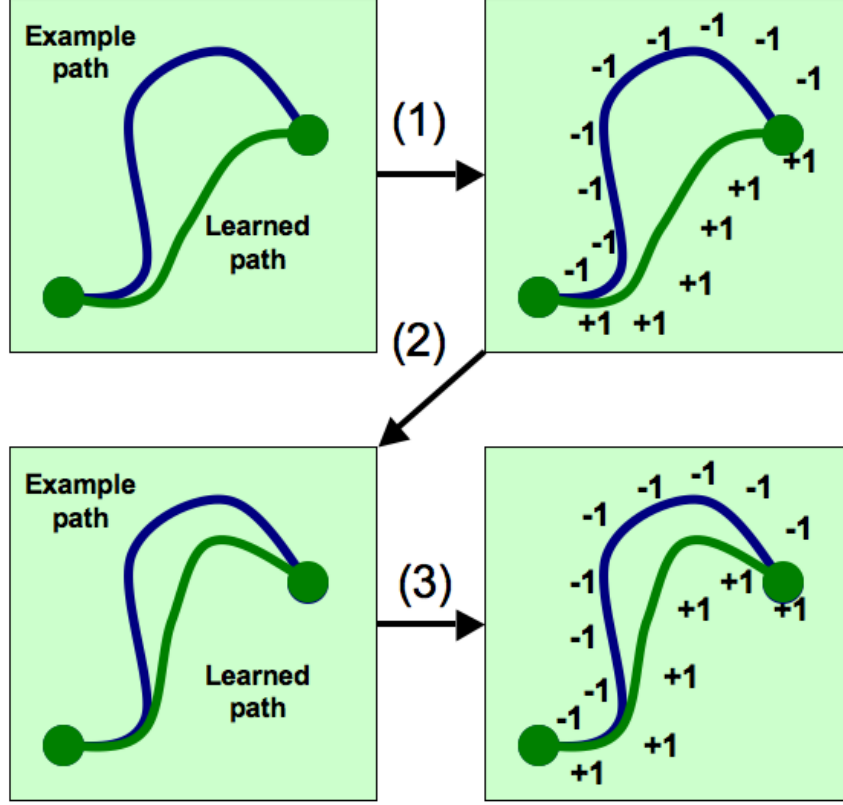


Figure 2.3: This figure visualizes an iteration of the algorithm discussed in section 2.1. Arrow (1) depicts the process of determining at which points in the feature space the function should be increased or decreased. Points encountered along the example path are labeled as -1 to indicate that their costs should be lowered, and points along the planned path are labeled as $+1$ to indicate that their costs should be raised. Along arrow (2) we generalize these suggestions to the entire feature space in order to implement the cost function modification. This incremental modification slightly improves the planning performance. We iterate this process, as depicted by arrow (3), until convergence.

field. Each pair consisting of an MDP $\mathcal{M}_i$ and an example trajectory through that MDP ξ_i has an associated loss field which maps each state-action pair of $\mathcal{M}_i$ to a nonnegative value. This value quantifies how bad it is for an agent to end up traversing a particular state-action pair when it should be following the example path. The simplest example of a loss field is the hamming field which places a loss of 0 over state-action pairs found along the example path and a loss of 1 over all other pairs. In our experiments, we typically use a generalization of this hamming loss that increases more gradually from a loss of 0 along the example path to a loss of 1 away from the example path. This induces a quantitative notion of “almost correct” which is useful when there

Algorithm 2 Loss-augmented LEARCH intuition

```
1: procedure LOSS-AUGLEARCH( training data  $\{(\mathcal{M}_i, \xi_i)\}_{i=1}^N$ , loss function  $l_i$ , feature function  
    $f_i$  )  
2:   while not converged do  
3:     for each example  $i$  do  
4:       Evaluate the cost function at each state-action feature vector  $f_i^{sa}$  for MDP  $\mathcal{M}_i$  to  
         create the cost map  $c_i^{sa} = c(f_i^{sa})$ .  
5:       Subtract the loss field from the cost map to create the loss-augmented cost map  
          $\tilde{c}_i^{sa} = c_i^{sa} - l_i^{sa}$ .  
6:       Plan through the loss-augmented cost map  $\tilde{c}_i^{sa}$  to find the loss-augmented path  $\xi_i^* =$   
          $\arg \min_{\xi \in \Xi_i} \sum_{(s,a) \in \xi} \tilde{c}_i^{sa}$ .  
7:       Increase the (log-)cost function at the points in the feature space encountered along  
         the loss-augmented path  $\{f_i^{sa} \mid \forall (s,a) \in \xi_i^*\}$ , and decrease the (log-)cost function  
         at points in the feature space encountered along the example path  $\{f_i^{sa} \mid \forall (s,a) \in$   
          $\xi_i\}$ .  
8:     end for  
9:   end while  
10: end procedure
```

is noise in the training trajectories. In what follows, for a given state-action pair (s, a) , we denote the state-action element of the loss field as l_i^{sa} , and the state-action element of the cost map as $c_i^{sa} = c(f_i^{sa})$.

The cost map modification step is called the *loss-augmentation*. During this step, the algorithm subtracts this loss field from the cost map element-wise. This subtraction amounts to defining the *loss-augmented* cost as $\tilde{c}_i^{sa} = c_i^{sa} - l_i^{sa}$. For state-action pairs that lie along the example path ξ_i , the loss is zero, and the cost function therefore remains untouched. As we venture away from the example path, the state-action loss values become increasingly large, and the augmentation step begins to lower the cost values substantially.

Intuitively, while the original algorithm discussed in Section 2.1 cares only that the example path be the minimum cost path through the final cost map, the loss-augmentation step forces the algorithm to continue making updates until the cost of the example path is smaller than that of any other path by a margin that scales with the loss of that path. If the loss of a path is low, then the path is similar to the example path and the algorithm allows the costs to be similar. However, if the loss is large, the two paths differ substantially, and the loss-augmented algorithm tries to find

a cost function for which the example path looks significantly more desirable than the alternative. In full, the algorithm becomes that given in Algorithm 2.

The algorithm discussed here is a novel *exponentiated* variant on functional gradient descent applied to the convex objective functional governing MMP. The next two Chapters develop the optimization tools underlying this algorithm. Chapter 6 then formally derives and presents the algorithm (which we list in Algorithm 11) after a discussion of the MMP framework in Chapter 5.

Chapter 3

Subgradient Convex Optimization

In this chapter and the next, we discuss some tools that have proven critical in developing the inverse optimal control techniques discussed in this thesis. We focus on two methods for convex optimization: finite-dimensional subgradient-based optimization and their nonparametric generalizations for functional optimization. The former class of techniques provides a strong theoretical basis for our study of learning algorithms in the linear setting, while the latter generalizes many of these techniques to nonlinear settings. In this Chapter, we first review batch and online subgradient methods for convex optimization and present some well known regret bounds which are important to the development of our theory of inverse optimal control.

Traditionally, optimization has been at the core of machine learning. Learning techniques originally developed without reference to explicit optimization are often later found to optimize, at least approximately, an understood objective. Such discoveries, e.g. boosting as functional gradient optimization, often shed light on the subject and make them more widely applicable. High-expectations regarding the performance of artificial neural networks in the '80s following the discovery of the back-propagation algorithm for computing gradients, punctuated by the quick success of support vector machines in the mid '90s and the subsequent difficulties with nonconvex optimization, drew the machine learning community away from biologically inspired models and toward stronger mathematical frameworks built around the formal tools of convex programming.

Throughout the history of machine learning, gradient descent has been a staple within the optimization toolbox. The backpropagation algorithm is at its core an efficient dynamic programming algorithm for computing the gradient of an objective function that capitalizes on the highly structured form of the network. However, as a result of the early success of support vector machines, the learning community has more recently focused its attention on a collection of sophisticated tools for solving constrained convex programming problems, particularly quadratic programming problems. Relations among supervised data are represented as constraints to which all valid hypotheses must (approximately) adhere, and learning techniques then optimize a relatively simple objective subject to these constraints. Extensive research has developed a number of tools for rapid convergence to the global minima of these constrained problems using dual optimization, but these tools, such as interior point methods, often have memory requirements that may grow cubically in the number of constraints (Boyd & Vandenberghe, 2004). In machine learning, these requirements are of particular concern since the number of constraints is often linear in the number of data points. Large-scale problems, therefore, are often prohibitive without special consideration. In spite of these practical problems, dual optimization has long been considered the method of choice for convex optimization in machine learning.

However, in the 60's, independent of machine learning applications, N. Z. Shor developed a generalization of the gradient known as the subgradient. This discovery led to a collection of primal gradient-based optimization techniques for convex nondifferentiable objective functions (Shor, 1985) that operated in the primal space rather than in the dual. These techniques require very little memory, and enjoy provable sublinear and linear convergence guarantees.

In 2003, M. Zinkevich found these simple subgradient-based optimization techniques to have strong theoretical properties in an online optimization setting that generalizes a breadth of prior work in minimizing errors online (Cesa-Bianchi, Long, & Warmuth, 1994; Kivinen & Warmuth, 1997; Gordon, 1999; Herbster & Warmuth, 2001; Kivinen & Warmuth, 2001) and expert problems (Freund & Schapire, 1999; Littlestone & Warmuth, 1989). This online optimization setting has close ties to online learning, which has proven to be an increasingly competitive and natural alternative to the batch learning setting. The online optimization framework can be modeled as a simple game

played between the optimizer and the environment. At the beginning of each round, the optimizer presents a hypothesis and the environment responds with an objective function used to score this hypothesis. At the end of each round, the optimizer has an opportunity to modify its hypothesis for the next round based on the collection of objective functions seen so far. The performance of the optimizer is scored throughout the game using a quantitative notion of regret over not having played the single best hypothesis in retrospect for each round. In his paper, Zinkevich showed that the regret of a simple optimizer who greedily follows the negative subgradient of the current objective function at each time step grows only sublinearly in time. This result sparked strong interest in the study of gradient-based solutions to online optimization and learning problem.

Evidence is building in the machine learning literature demonstrating that subgradient-based optimization in the primal is an important and applicable learning technique for a wide range of problems; these algorithms have particularly nice properties for the difficult memory intensive problems we study in this thesis. Most traditional treatments of subgradient methods develop theory first for what we would call the “batch” setting, where the goal is to optimize a single fixed objective function well. We find it more natural to develop a collection of basic tools in the online setting first. Many of the batch results are the straightforward to derive as simple corollaries; indeed, the subgradient-based batch optimization techniques we consider here are all special cases of the online setting. Following this theme further using the work of (Cesa-Bianchi, Conconi, & Gentile, 2004a), we can also bound the generalization performance of a hypothesis found by an online algorithm, effectively converting our regret bounds into generalization bounds. We will see in Chapter 7 that these bounds improve on the current state-of-the-art in the generalization performance of maximum margin structured classification.

Both the online and batch subgradient optimization procedures reside at the core of the learning procedures discussed in this thesis. Below we present a collection of subgradient-based primal convex optimization tools and analyze their performance in both online and batch settings. Chapter 4 explores their generalization to infinite-dimensional function spaces.

3.1 Subgradients and strong convexity

A *subgradient* generalizes the notion of derivative and gradient to functions that are convex but not necessarily differentiable. In this section, we define the subgradient and review some properties often used in computing subgradients for functions in machine learning. We additionally review the notion of strong-convexity, a property that often arises frequently in regularized risk functions. This property is known to improve convergence in both online and batch optimization settings for gradient-based algorithms.

3.1.1 Subgradient definition and properties

Formally, a subgradient of a convex function h at a point $w \in \mathcal{W}$ is any vector g which can be used to form an affine function that lower bounds h everywhere in $\mathcal{W}$ and equals h at w . Mathematically, we can write this condition as

$$\forall w' \in \mathcal{W}, h(w') \geq h(w) + g^T(w' - w). \quad (3.1)$$

The expression on the right side of the inequality is the affine function. When $w' = w$, the rightmost term vanishes and the affine function equals $h(w)$. This inequality requires that the affine function lower bound h across the entire domain $\mathcal{W}$. In general, there could be a continuum of vectors g , denoted $\partial h(w)$, for which this condition holds. However, at points of differentiability, the gradient is the single unique subgradient.

We list here four common properties of subgradients which we use throughout this thesis.

1. Subgradient operators are linear. Formally, for any convex functions $f, g : \mathbb{R}^d \rightarrow \mathbb{R}$ and constants $\alpha, \beta \in \mathbb{R}$, if $\partial f(x)$ and $\partial g(x)$ are the subgradient sets at x for f and g , respectively, then the subgradient of $h = f + g$ can be written as follows:

$$\partial h(x) = \{y = y_1 + y_2 \mid y_1 \in \partial f(x), y_2 \in \partial g(x)\}. \quad (3.2)$$

2. The gradient is the unique subgradient of a differentiable function.

3. Denoting $y^* = \operatorname{argmax}_y f(x, y)$ for convex functions $f(., y)$ differentiable in their first argument, $\nabla_x f(x, y^*)$ is a subgradient of the piecewise differentiable convex function $\tilde{f}(x) = \max_y f(x, y)$.
4. The chain rule may be applied in a way analogous to the strictly differentiable case.

Because of the similarity between properties of subgradients and properties of traditional gradients, for convenience we often denote the subgradient of a function using same well known notation $\nabla f(x)$.

3.1.2 Strong convexity

In many of the theorems proven below, we require a stronger lower bound on the function than the one provided solely by the subgradient in Equation 3.1. We attain such a bound using a concept called *strong convexity*. Intuitively, strong convexity indicates that the function grows faster than quadratically everywhere in the domain.

Definition 3.1.1: *H-strong convexity* A convex function $c : \mathcal{W} \rightarrow \mathbb{R}$ is said to be *H-strongly convex* if there is an $H > 0$ such that for all $w, w' \in \mathcal{W}$,

$$c(w') \geq c(w) + g^T(w' - w) + \frac{H}{2} \|w' - w\|^2, \quad (3.3)$$

where g is any subgradient at w .

We first note that the second-order Taylor expansion is exact for the convex function $\frac{\lambda}{2} \|w\|^2$, so the strong convexity bound holds trivially with $H = \lambda$ (the right-hand-side of the bound in Equation 3.3 is the second-order Taylor expansion for twice differentiable functions with isotropic Hessian). This function is, therefore, λ -strongly convex. Next, we show that the sum of an H_1 -strongly convex function and an H_2 -strongly convex function is $(H_1 + H_2)$ -strongly convex. In particular, this means any convex regularized risk function with regularizer $\frac{\lambda}{2} \|w\|^2$ is at least λ -strongly convex.

Theorem 3.1.2: Let $h_1 : \mathcal{W} \rightarrow \mathbb{R}$ be an H_1 -strongly convex functions, and let $h_2 : \mathcal{W} \rightarrow \mathbb{R}$ be an H_2 -strongly convex function, where $H_1, H_2 \geq 0$. (We allow either or both of H_1 and H_2 to

potentially equal zero.) Then $h = h_1 + h_2$ is $(H_1 + H_2)$ -strongly convex.

Proof. By definition, for all $w, w' \in \mathcal{W}$, $h_1(w') \geq h_1(w) + g_1^T(w' - w) + \frac{H_1}{2}\|w' - w\|^2$, and $h_2(w') \geq h_2(w) + g_2^T(w' - w) + \frac{H_2}{2}\|w' - w\|^2$, where g_1 and g_2 are subgradients of h_1 and h_2 , respectively, at w . Adding these two inequalities gives

$$h(w') = h_1(w') + h_2(w') \geq h_1(w) + h_2(w) + (g_1 + g_2)^T(w' - w) + \frac{H_1 + H_2}{2}\|w' - w\|^2 \quad (3.4)$$

$$= h(w) + g^T(w' - w) + \frac{H}{2}\|w' - w\|^2, \quad (3.5)$$

where g is a subgradient of h at w and $H = H_1 + H_2$. Therefore, h is $(H_1 + H_2)$ -strongly convex. $\square$

3.1.3 Bounding the effective optimization radius

We now present a simple theorem which will prove useful for a number of the settings studied in this chapter. In machine learning, we're often interesting in optimizing strongly-convex regularized risk functions that take the form $c(w) = r(w) + \frac{\lambda}{2}\|w\|^2$. We can show that the norm of the optimizer of such a function is bounded by $\|w^*\| \leq \frac{G}{\lambda}$, where G bounds the gradient of the risk term $r(w)$. Interestingly, this property implies that the gradient of the regularized risk function is everywhere bounded by $\|\nabla c(w)\| \leq 2G$ when we constrain $\|w\| \leq \frac{G}{\lambda}$. This property simplifies many of the analytical expressions derived below.

Theorem 3.1.3: *Let $c(w) = r(w) + \frac{\lambda}{2}\|w\|^2$ where $r(w)$ is an arbitrary convex function with subgradient bounded by G . Then $\|w^*\| \leq \frac{G}{\lambda}$ where $w^* = \arg \min_{w \in \mathcal{W}} c(w)$. Moreover, this bound is tight.*

Proof. We first show that the gradient of this function always has positive inner product with w when $\|w\| > \frac{1}{\lambda}$. (Taking a step in the direction of the negative gradient will therefore bring the point closer to the ball of radius $\frac{1}{\lambda}$).

The inner product between gradient of $c(w)$ and w takes the form

$$\nabla c(w)^T w = (g + \lambda w)^T w = w^T g + \lambda \|w\|^2, \quad (3.6)$$

Algorithm 3 Online subgradient method update

```
1: procedure ONLINESUBGRADIENTUPDATE(  $c_t(w) = r_t(w) + \frac{\lambda_t}{2}\|w\|^2, w_t, \alpha_t, G$ )
2:   choose  $g_t \in \partial c(w_t)$ 
3:   set  $w_{t+1} = \mathcal{P}_{\mathcal{W}_t}[w_t - \alpha_t g_t]$ 
4:   return  $w_{t+1}$ 
5: end procedure
```

where $g = \nabla r(w)$. This inner product is minimized when g directly opposes w . Since $\|g\| \leq G$, the gradient minimizing this inner product must be $g = -G \frac{w}{\|w\|}$. The inner product at that point becomes

$$\min_{\|g\| \leq G} c(w)^T w = -G w^T \frac{w}{\|w\|} + \lambda \|w\|^2 = -G\|w\| + \lambda \|w\|^2. \quad (3.7)$$

This expression is a positive definite quadratic in $\|w\|$ with minimum at $\frac{G}{2\lambda}$ and zeros at 0 and $\|w\| = \frac{G}{\lambda}$. Therefore, the inner product is positive for all $\|w\| \geq \frac{G}{\lambda}$.

We now show that the bound is tight by example. Let $c_q(w) = w^T x + \frac{\lambda}{2}\|w\|^2$ with $\|x\| = G$. The minimizer of this quadratic can be found by setting its gradient to zero.

$$\nabla c_q(w^*) = x + \lambda w^* = 0 \quad (3.8)$$

$$\Rightarrow w^* = -\frac{1}{\lambda}x. \quad (3.9)$$

The norm of w^* is $\|w^*\| = \frac{G}{\lambda}$ since $\|x\| = G$. Therefore, w^* lies on the ball of radius $\frac{G}{\lambda}$. $\square$

3.2 The online setting

In the online setting for convex optimization, the optimizer is presented with a sequence of objective functions that each score hypotheses. The game is to play a hypothesis at the beginning of each round before seeing the objective. Once the objective has been presented, the optimizer is scored based on accrued objective value.

Formally, in the online prediction setting for a sequence of regularized risk functions, our online update is given explicitly in Algorithm 3. We denote our space of hypotheses as $\mathcal{W} \subset \mathbb{R}^d$. The online optimization algorithm chooses a sequence of iterates $\{w_t\}_{t=1}^T$ in response to objective functions $\{c_t(\cdot)\}_{t=1}^T$. We present this update in its most general form, in which we allow the sequence of

regularizers to decrease systematically over time. At each iteration, the algorithm takes a step in the direction of the negative subgradient and, if necessary, projects back onto a feasible set which at a minimum is a ball of radius $\frac{G}{\lambda_t}$. This projection need only be approximate as defined by the following *approximate projection* property

$$\forall w' \in \mathcal{W}, \quad \|\mathcal{P}_{\mathcal{W}_t}[w] - w'\| \leq \|w - w'\|. \quad (3.10)$$

3.2.1 Online regret

We consider two setting for the analysis of online algorithms in this thesis. The first setting is concerned with online optimization precisely as defined in (Zinkevich, 2003), where the goal is to perform well with respect to the sequence of objective functions encountered. The second, however, is concerned with online learning and prediction. In the online prediction setting, we consider sequences of regularized risk objective functions of the form

$$c_t(w) = r_t(w) + \frac{\lambda_t}{2} \|w\|^2, \quad (3.11)$$

where $r_t(w)$ is a risk function upper bounding the true loss $l_t(w) \leq r_t(w)$. (E.g. for online support vector learning, $r_t(w)$ is the hinge loss which upper bounds the true zero-one loss.) In this case, the goal is not necessarily to perform well on the sequence of objective functions, but on the sequence of loss functions $l_t(w)$ they upper bound. As we will see, analyzing this setting becomes slightly more tricky, particularly because we want to compare our performance on $l_t(w)$ to the optimal upper bound on the sequence of unregularized risk terms $r_t(w)$. We will discuss this setting in detail below. Importantly, the online algorithm is the same between both settings; only the analysis of the algorithm differs.

Optimization regret

At a high level, we measure the optimization success of an online algorithm using a quantitative notion of *optimization regret*. Specifically, we define this form of regret as

$$\text{regret}_o(T) = \sum_{t=1}^T c_t(w_t) - \min_{w \in \mathcal{W}} c_t(w). \quad (3.12)$$

Intuitively, $\text{regret}_o(T)$ measures how much better the optimizer would have performed if it had played the single best hypothesis in retrospect at each round rather the sequence it chose.¹

If the regret increases linearly with time, it grows by a constant amount each iteration and the average regret therefore does not approach zero over time. In particular, were we to apply such an algorithm to a sequence of objective functions sampled i.i.d. from a fixed distribution, we would not be able to prove that the algorithm minimizes the expected objective over time. Therefore, at a minimum, we strive to prove sublinear regret bounds for candidate online algorithms.

Typically, the baseline rate of regret of an online optimization algorithm increases as $O(\sqrt{T})$, although in some cases we can do better than that. There are a number of algorithms that optimize very well when the objective sequence is strongly convex as we will show below. These algorithms can achieve bounds on the optimization regret of the order $\text{regret}_o(T) \leq O(\log T)$.

Prediction regret

On the other hand, in online learning we care more about minimizing the loss $l_t(w)$ than optimizing the upper bound on the value formed by our objective. In this case, we can analyze regret expressions of the following form:

$$\text{regret}_p(T) = \sum_{t=1}^T l_t(w_t) - \min_{w \in \mathcal{W}} \sum_{t=1}^T r_t(w). \quad (3.13)$$

¹There exist generalizations of this notion of regret that, for instance, rate the sequence of hypotheses relative to all slowly varying sequences (Zinkevich, 2003), but the measure presented here is convenient for understanding the performance of online algorithms relative their batch counterparts.

Regret bounds

We have the tools to derive bounds on both the optimization and prediction regret in a number of settings starting with a simple setting in which the regularization is zero $\lambda_t = 0$ for all objective functions in the sequence. It is often beneficial to include an explicit regularization term in our objective sequence in order to regulate the hypothesis during learning. We consider two additional settings where we have explicit regularization and discuss the relative tradeoffs of each.

In the theorems below, we often require bounds on the gradient of the objective and on the gradient of the risk function. We denote these bounds as G_o and G , respectively, and define them as (for all t) $\sup_{w \in \mathcal{W}} \|\nabla c_t(w)\| \leq G_o$ and $\sup_{w \in \mathcal{W}} \|\nabla r_t(w)\| \leq G$. For the unregularized setting, we constrain the hypotheses to reside in a ball around the origin of a specified radius $R > 0$. Since the size of the optimal weight vector $\|w^*\|$ is inversely proportional to the size of the margin achieved during optimization for margin-based learning machines, whenever possible we write our prediction bounds in terms of $\|w^*\|$ rather than immediately upper bounding that term.

3.2.2 The unregularized case

Zinkevich (2003) shows that the simplest gradient-based online optimization algorithm which repeatedly applies the update given in Algorithm 3 using a step size sequence $\{c/\sqrt{t}\}_{t=1}^\infty$, has sublinear optimization regret. This theorem holds for any sequence of convex objective functions, but we present the result here in terms of a sequence of convex risk functions for convenience.

Theorem 3.2.1: Sublinear regret of the online subgradient method *Let $\{r_t(\cdot)\}_{t=1}^T$ be an arbitrary sequence of convex risk functions, and denote the diameter of the space as $D \leq 2R$. Then the online subgradient method using step size sequence $\alpha_t = \frac{D}{G\sqrt{2t}}$ returns a sequence of iterates $\{w_t\}_{t=1}^T$ with regret*

$$\sum_{t=1}^T r_t(w_t) - \min_{w \in \mathcal{W}} \sum_{t=1}^T r_t(w) \leq \sqrt{2}GD \left(\sqrt{T} - \frac{1}{4} \right). \quad (3.14)$$

Proof. Using the arguments of Zinkevich in (Zinkevich, 2003), but using a scaled step size of $\alpha_t = \frac{\eta}{\sqrt{t}}$

gives a regret bound with the following form

$$\sum_{t=1}^T c_t(w_t) - \min_{w \in \mathcal{W}} \sum_{t=1}^T c_t(w) \leq \frac{D^2 \sqrt{T}}{2\eta} + \eta \left(\sqrt{T} - \frac{1}{2} \right) G^2. \quad (3.15)$$

We now optimize the portion of this bound multiplying $\sqrt{T}$ by taking its derivative with respect to η and setting it to zero. Doing so gives the following expression:

$$-\frac{D^2}{\eta^{*2}} + G^2 = 0 \quad \Rightarrow \quad \eta^* = \frac{\sqrt{2} D}{2 G}. \quad (3.16)$$

Plugging this result back into our Equation 3.15 gives

$$\frac{D^2 \sqrt{T}}{2\eta^*} + \eta^* \left(\sqrt{T} - \frac{1}{2} \right) G^2 = \frac{\sqrt{2}GD}{2} \sqrt{T} + \frac{GD}{\sqrt{2}} \left(\sqrt{T} - \frac{1}{2} \right) \quad (3.17)$$

$$= \sqrt{2}GD\sqrt{T} - \frac{GD\sqrt{2}}{4} \quad (3.18)$$

$$= \sqrt{2}GD \left(\sqrt{T} - \frac{1}{4} \right). \quad (3.19)$$

□

This theorem presents the optimization regret bound. However, since the sequence of objective functions are unregularized, we can easily extract the corresponding prediction regret bound since each objective is a risk function $r_t(w)$ which upper bounds the corresponding loss $l_t(w)$. In this case, the bounding term for the prediction regret is equivalent to the bounding term of Theorem 3.2.1.

3.2.3 Constant regularization

For many learning problem, explicit regularization may be preferred over projecting onto a constrained feasible set. This subsection explores the affect on online learning and prediction when a constant regularization term is added to the risk. In these results, we make use of Theorem 3.1.3 to define a feasible set in terms the regularization constant $R \leq \frac{G}{\lambda_t}$, where G upper bounds the risk gradient as defined above.

In this setting, we can capitalize on the strong convexity of the regularizer. Theorem 3.2.1 demonstrates that the online subgradient method achieves $O(\sqrt{T})$ regret in the general case, where

the objective can be any convex function, but by gaining strong convexity, we now achieve a stronger bound for online optimization. In particular, if each objective in our sequence is λ -strongly convex, choosing a more aggressive step size $\alpha_t = \frac{1}{\lambda t}$ enables the proof of the following bound, originally presented in (Hazan, Agarwal, & Kale, 2006) (adapted to our notion for the special case of regularized risk functions):

Theorem 3.2.2: *Let $\{c_t(\cdot)\}_{t=1}^\infty$ be a sequence of convex objective functions with $c_t(w) = r_t(w) + \frac{\lambda}{2}\|w\|^2$. Then the online subgradient method using step size sequence $\alpha_t = \frac{1}{\lambda t}$ returns a sequence of iterates $\{w_t\}_{t=1}^T$ with the property*

$$\sum_{t=1}^T c_t(w_t) - \min_{w \in \mathcal{W}} \sum_{t=1}^T c_t(w) \leq \frac{2G^2}{\lambda}(1 + \log T). \quad (3.20)$$

Proof. From the discussion in Section 3.1.2, our objective sequence is λ -strongly convex. The general bound on the optimization regret of the online subgradient algorithm from (Hazan, Agarwal, & Kale, 2006) is

$$\sum_{t=1}^T c_t(w_t) - \min_{w \in \mathcal{W}} \sum_{t=1}^T c_t(w) \leq \frac{G_o^2}{2\lambda}(1 + \log T), \quad (3.21)$$

where G_o upper bounds the gradient of the full objective. Using the argument of 3.1.3, we know $G_o \leq 2G$. Plugging this gradient bound into Equation 3.21 gives the desired result. $\square$

Since each risk term is adorned with a regularizer deriving a prediction bound is less straightforward. In particular, the constant regularization term introduces a systematic bias to learning which prevents the learner from competing well with the sequence of risk functions (alone) on the long run. However, if we know the game horizon T (i.e. the number of online iterations) in advance, we can choose the regularization constant to be sufficiently small in order to achieve the best performance possible within those T iterations. The following theorem summarizes this result.

Theorem 3.2.3: Prediction regret for constant regularization. *Let $c_t(w) = r_t(w) + \frac{\lambda}{2}\|w\|^2$ be a sequence of regularized risk functions where each risk term $r_t(w)$ upper bounds the true prediction error $l_t(w)$, and let $\{w_t\}_{t=1}^T$ be a sequence of iterates produced by the online subgradient method on $\{c_t(w)\}_{t=1}^T$. The prediction regret of the online subgradient method with a step size*

sequence $\alpha_t = \frac{1}{\lambda t}$ and regularization constant $\lambda = \frac{2G}{\|w^*\|} \sqrt{\frac{1+\log T}{T}}$ is

$$\sum_{t=1}^T l_t(w_t) \leq \sum_{t=1}^T r_t(w^*) + 2G\|w^*\| \sqrt{T(1+\log T)}. \quad (3.22)$$

where $w^* = \arg \min_{w \in \mathcal{W}} \sum_{t=1}^T r_t(w)$.

Proof. By Theorem 3.2.2 we have

$$\sum_{t=1}^T l_t(w_t) \leq \sum_{t=1}^T r_t(w_t) + \frac{\lambda}{2} \|w_t\|^2 \quad (3.23)$$

$$\leq \min_{w \in \mathcal{W}} \sum_{t=1}^T \left(r_t(w) + \frac{\lambda}{2} \|w\|^2 \right) + \frac{2G^2}{\lambda} (1 + \log T) \quad (3.24)$$

$$\leq \sum_{t=1}^T r_t(w^*) + \frac{\lambda T}{2} \|w^*\|^2 + \frac{2G^2}{\lambda} (1 + \log T). \quad (3.25)$$

Choosing $\lambda = \frac{2G}{\|w^*\|} \sqrt{\frac{1+\log T}{T}}$, then gives the desired result. (This value for λ optimizes the bound. It can be derived by taking the derivative of the bound with respect to λ and setting it to zero.) $\square$

3.2.4 Attenuated regularization

As observed in the above discussion, adding a constant regularization term introduces a systematic bias thereby preventing us from attaining zero average regret across an infinite horizon. Indeed, Equation 3.25 shows that in the online setting, without knowledge of the game horizon T , the constant regularization introduces a regret term that grows linearly with time.

We can gain some intuition for this problem by considering the batch setting. In the batch setting, the regularized risk objective function takes the form $c(w) = \sum_{i=1}^N r_i(w) + \frac{\lambda}{2} \|w\|^2 \propto \frac{1}{N} \sum_{i=1}^N r_i(w) + \frac{\lambda_N}{2} \|w\|^2$, where $\lambda_N = \frac{\lambda}{N}$. Here, the regularization term protects against overfitting when the number of examples is small, but as the number of examples increase, the regularization term is less important and therefore attenuates in strength relative to the risk term.

We, therefore, explore an online setting in which we allow the degree of strong convexity of the function sequence, i.e. the size of the regularization, to decrease systematically over time. The following theorem bounds the optimization regret of this setting. Again, we present the theorem

in terms of regularized risk functions of the form shown in Equation 3.11, but the generalization to arbitrary strongly-convex objective sequences is straightforward.

Theorem 3.2.4: Optimization regret of attenuated strong convexity. *Let $\{c_t(\cdot)\}_{t=1}^T$ be a sequence of strongly convex functions with attenuated regularization of the form $c_t(x) = r_t(x) + \frac{\lambda}{2\sqrt{t}}\|x\|^2$. The online subgradient method under step size sequence $\alpha_t = \frac{1}{\lambda\sqrt{t}}$ with time varying radius constraint $R_t = \frac{\sqrt{t}}{\lambda}$ has regret bounded by*

$$\sum_{t=1}^T c_t(w_t) - \min_{w \in \mathcal{W}} \sum_{t=1}^T c_t(w) \leq \frac{4G^2}{\lambda} \left(\sqrt{T} - \frac{1}{2} \right), \quad (3.26)$$

Proof. We can expand a single step as $\|w_{t+1} - w^*\|^2 \leq \|w_t - \alpha_t g_t - w^*\|^2 = \|w_t - w^*\|^2 - 2\alpha_t g_t^T(w_t - w^*) + \alpha_t^2 \|g_t\|^2$. Since each objective $c_t(\cdot)$ is $\frac{\lambda}{\sqrt{t}}$ -strongly convex, we have the following bound:

$$c_t(w^*) \geq c_t(w_t) + g_t^T(w^* - w_t) + \frac{H_t}{2} \|w^* - w_t\|^2. \quad (3.27)$$

Combining this bound with the above expansion and summing across all time, we get

$$\sum_{t=1}^T (c_t(w_t) - c_t(w^*)) \leq \frac{1}{2} \sum_{t=1}^T \left[\left(\frac{1}{\alpha_t} - H_t \right) \|w_t - w^*\|^2 - \frac{1}{\alpha_t} \|w_{t+1} - w^*\|^2 \right] + \frac{1}{2} \sum_{t=1}^T \alpha_t \|g_t\|^2 \quad (3.28)$$

$$= \frac{1}{2} \sum_{t=1}^T \alpha_t \|g_t\|^2 + \frac{1}{2} \left(\frac{1}{\alpha_1} - H_1 \right) \|w_1 - w^*\|^2 + \frac{1}{2} \sum_{t=1}^T \left(\frac{1}{\alpha_t} - \frac{1}{\alpha_{t+1}} - H_t \right) \|w_t - w^*\|^2. \quad (3.29)$$

Examining these terms, we find $\frac{1}{\alpha_1} - H_1 = H - H = 0$ and

$$\frac{1}{\alpha_t} - \frac{1}{\alpha_{t+1}} - H_t = H \left(\sqrt{t} - \sqrt{t+1} - \frac{1}{\sqrt{t}} \right) \quad (3.30)$$

$$\leq H \left(\frac{1}{2\sqrt{t-1}} - \frac{1}{\sqrt{t}} \right). \quad (3.31)$$

The final expression follows from the mean value theorem since the first derivative of $f(x) = \sqrt{x}$ is monotonically decreasing. Expanding that final expression, we find $\frac{1}{2\sqrt{t-1}} - \frac{1}{\sqrt{t}} \leq \frac{\sqrt{t} - 2\sqrt{t-1}}{2\sqrt{t(t-1)}}$. For $t > 1$ the

denominator is positive, and the numerator can be bounded as

$$\sqrt{t} - 2\sqrt{t-1} = (\sqrt{t} - \sqrt{t-1}) - \sqrt{t-1} \quad (3.32)$$

$$\leq \frac{1}{2\sqrt{t-1}} - \sqrt{t-1} = \frac{1 - 2(t-1)}{2\sqrt{t-1}} \quad (3.33)$$

$$= \frac{3-2t}{2\sqrt{t-1}} < 0, \quad (3.34)$$

where the final inequality holds for $t \geq 2$. Therefore,

$$\sum_{t=1}^T (c_t(w_t) - c_t(w^*)) \leq \frac{1}{2} \sum_{t=1}^T \alpha_t \|g_t\|^2. \quad (3.35)$$

Now, given a bound on the gradient $\|g_t\| \leq G + \frac{\lambda}{\sqrt{T}} R_t \leq 2G$, and using $\sum_{t=1}^T \frac{1}{\sqrt{x}} \leq 1 + \int_1^T \frac{1}{\sqrt{x}} dx = 2\sqrt{T} - 1$, we arrive at Equation 3.26. $\square$

This attenuated regularization framework makes it possible utilize regularization during learning while still retaining the ability to bound the prediction regret of an online learner without advanced knowledge of the horizon T . Moreover, this bound improves on that presented in Theorem 3.2.3 by removing extraneous $\log T$ terms. The following theorem presents this result for a general class of convex regularized risk functions.

Theorem 3.2.5: Prediction regret for attenuated regularization. *Let $c_t(w) = r_t(w) + \frac{\lambda}{2\sqrt{t}} \|w\|^2$ be a sequence of regularized risk functions where each risk term $r_t(w)$ upper bounds the true prediction error $l_t(w)$, and let $\{w_t\}_{t=1}^T$ be a sequence of iterates produced by the online subgradient method on $\{c_t(w)\}_{t=1}^T$. Choosing $\lambda = \frac{2G}{\|w^*\|}$, we get the following regret bound:*

$$\sum_{t=1}^T l_t(w_t) - \min_{w \in \mathcal{W}} \sum_{t=1}^T r_t(w) \leq 4G\|w^*\| \left(\sqrt{T} - \frac{1}{2} \right), \quad (3.36)$$

where $w^* = \arg \min_{w \in \mathcal{W}} \sum_{t=1}^T r_t(w)$.

Proof. From Theorem 3.2.4, we have

$$\sum_{t=1}^T l_t(w_t) \leq \sum_{t=1}^T \left(r_t(w_t) + \frac{\lambda}{2\sqrt{t}} \|w_t\|^2 \right) \leq \min_{w \in \mathcal{W}} \left\{ \sum_{t=1}^T r_t(w) + \frac{\lambda}{2\sqrt{t}} \|w\|^2 \right\} + \frac{4G^2}{\lambda} \left(\sqrt{T} - \frac{1}{2} \right) \quad (3.37)$$

$$\leq \sum_{t=1}^T r_t(w^*) + \frac{\lambda}{2} \|w^*\|^2 \sum_{t=1}^T \frac{1}{\sqrt{t}} + \frac{4G^2}{\lambda} \left(\sqrt{T} - \frac{1}{2} \right) \quad (3.38)$$

$$\leq \sum_{t=1}^T r_t(w^*) + \lambda \|w^*\|^2 \left(\sqrt{T} - \frac{1}{2} \right) + \frac{4G^2}{\lambda} \left(\sqrt{T} - \frac{1}{2} \right) \quad (3.39)$$

$$= \sum_{t=1}^T r_t(w^*) + \left(\lambda \|w^*\|^2 + \frac{4G^2}{\lambda} \right) \left(\sqrt{T} - \frac{1}{2} \right). \quad (3.40)$$

We can optimize the factor multiplying $\left(\sqrt{T} - \frac{1}{2} \right)$ by taking the derivative with respect to λ and setting it to zero. Doing so gives the optimal value $\lambda^* = \frac{2G}{\|w^*\|}$, which explains our choice of λ . Plugging this value back into the factor gives

$$\lambda \|w^*\|^2 + \frac{4G^2}{\lambda} = 4G \|w^*\|. \quad (3.41)$$

Using this factor in the above regret bound gives Equation 3.26. $\square$

3.3 From regret bounds to generalization bounds

Statistical generalization from batch data drawn i.i.d. from a fixed distribution to new data drawn from that same distribution has been a core focus of machine learning since the early '80s with the development of the Probably Approximately Correct (Valiant, 1984) learning theory. The statistics and machine learning communities have worked over the past two decades to develop a collection of tools to facilitate the proof of generalization bounds. Since the discovery of the VC-dimension (Vapnik, 1995), bounds on covering numbers have been found to provide strong bounds for a range of learning algorithms. In particular, they have been used extensively in proving bounds for margin-based learning machines (Zhang, 2002). In (Taskar, Guestrin, & Koller, 2003), these arguments are carried over to prove the first generalization bounds for the maximum margin structured classification setting.

However, covering number arguments are often lengthy and complicated. More recently, re-

searchers have been studying the strong connection between the online and batch settings in theoretical machine learning, and interesting connections between regret bounds derived for online algorithms and batch generalization have been discovered. In 2001, Cesa-Bianchi and his coauthors developed these ideas into a framework for converting regret bounds into strong generalization bounds for online algorithms using the theory of Martingales (Dietterich, Becker, & Ghahramani, 2001; Cesa-bianchi, Conconi, & Gentile, 2004b). The resulting bounds are straightforward and often state-of-the-art. We demonstrate later in this thesis that these arguments can be used to both simplify and improve the generalization bounds for maximum margin structured classification.

We reproduce and discuss the implications of one of the results (Theorem 2) from (Dietterich, Becker, & Ghahramani, 2001) which will prove important later in this thesis. (We modify the notation slightly for clarity.) In the most general setting, we are concerned with training learners to map between a pair of arbitrary sets $\mathcal{X}$ and $\mathcal{Y}$. Distributions over these spaces are represented implicitly through the random variables X and Y . This mapping uses an abstract decision space $\mathcal{D}$ as a conduit, and the loss of a hypothesis $h : \mathcal{X} \rightarrow \mathcal{D}$ on an example (x, y) is measured using a nonnegative bounded loss function $l : \mathcal{D} \times \mathcal{Y} \rightarrow [0, L]$. For our purposes, this loss function must be convex in its first argument. For instance, the binary hinge loss for the linear two-class support vector machine is defined over a decision space of real valued scores $s \in \mathbb{R}$ via $l(s, y) = \frac{1}{2} \max\{0, 1 - ys\}$, where $y \in \{-1, 1\}$. The class of hypotheses $\mathcal{H}$ in this case is parameterized by a finite-dimensional weight vector w defining a linear function mapping the input space $\mathcal{X}$ of vectors to a real value $s = h(x) = w^T x$. By constraining w to a ball of radius 1, the loss function becomes bounded by 1 when each input is at most unit norm $\|x\| \leq 1$.

These definitions provide us the nomenclature to present the following theorem:

Theorem 3.3.1: *Let $\{(x_t, y_t)\}_{t=1}^T$ be a random sample of T examples from a distribution represented by the random variable $Z = (X, Y)$, and let $\{h_t\}_{t=1}^T$ be the sequence of hypotheses induced by an online learning algorithm. Then with probability greater than $1 - \delta$,*

$$E(l(\bar{h}(X), Y)) < \frac{1}{T} \sum_{t=1}^T l(h_t(x_t), y_t) + L \sqrt{\frac{2}{T} \log \frac{1}{\delta}}, \quad (3.42)$$

where $\bar{h} = \frac{1}{T} \sum_{t=1}^T h_{t-1}$ is the average hypothesis, and $\delta \in [0, 1]$.

Using this theorem, given any average regret bound of the form

$$\frac{1}{T} \sum_{t=1}^T l(h_t(x_t), y_t) \leq \frac{1}{T} \min_{h \in \mathcal{H}} \sum_{t=1}^T l(h(x_t), y_t) + \phi(T), \quad (3.43)$$

we can induce a generalization bound. The probability guarantee of the above theorem remains if we replace the right-hand-side of the inequality with an upper bound. Since Equation 3.43 upper bounds the empirical risk of an i.i.d. data set $\{(x_t, y_t)\}_{t=1}^T$, the bound induces the following probability guarantee: With probability $1 - \delta$,

$$E(l(\bar{h}(X), Y)) < \frac{1}{T} \min_{h \in \mathcal{H}} \sum_{t=1}^T l(h(x_t), y_t) + \phi(T) + L \sqrt{\frac{2}{T} \log \frac{1}{\delta}}. \quad (3.44)$$

This bound relates empirical performance of the batch learning problem to the generalization performance of the average hypothesis found by the online learner.

Theorem 3.3.2: Generalization of the online subgradient method. *Let $\{c_t\}_{t=1}^T$ be a sequence of convex risk functions of the form $c_t(w) = l_w(x_t, y_t)$ for some loss function $l_w(\cdot, \cdot)$ parameterized by w , with $\{(x_t, y_t)\}_{t=1}^T$ sampled i.i.d. from a fixed distribution. If $\{w_t\}_{t=1}^T$ is a sequence of iterates produced by the online subgradient method on these objective functions, then the expected risk of $\bar{w} = \frac{1}{T} \sum_{t=1}^T w_t$ is bounded by*

$$E(l_{T+1}(\bar{w})) \leq \min_{w \in \mathcal{W}} \frac{1}{T} \sum_{t=1}^T r_t(w) + \left(GD + L \sqrt{\log \frac{1}{\delta}} \right) \sqrt{\frac{2}{T}}, \quad (3.45)$$

where G is a bound on the gradient of the risk functions, and D is the diameter of the hypothesis space $\mathcal{W}$.

Proof. By Theorem 3.2.1, the average regret of the online subgradient method is bounded by

$$\frac{1}{T} \sum_{t=1}^T r_t(w_t) - \min_{w \in \mathcal{W}} \frac{1}{T} \sum_{t=1}^T r_t(w) \leq \frac{1}{T} \sqrt{2} GD \sqrt{T} = GD \sqrt{\frac{2}{T}}. \quad (3.46)$$

Combining this bound with the generic generalization bound given in Theorem 3.3.1 in the way outlined

above, gives the following expression:

$$E(l_{T+1}(\bar{w})) \leq \frac{1}{T} \sum_{t=1}^T r_t(w_t) + L\sqrt{\frac{2}{T} \log \frac{1}{\delta}} \quad (3.47)$$

$$\leq \min_{w \in \mathcal{W}} \frac{1}{T} \sum_{t=1}^T r_t(w) + GD\sqrt{\frac{2}{T}} + L\sqrt{\frac{2}{T} \log \frac{1}{\delta}} \quad (3.48)$$

$$= \min_{w \in \mathcal{W}} \frac{1}{T} \sum_{t=1}^T r_t(w) + \left(GD + L\sqrt{\log \frac{1}{\delta}} \right) \sqrt{\frac{2}{T}}. \quad (3.49)$$

□

Since we often know the number of examples in advance in the batch setting, the constant regularization online setting is a natural choice. The next theorem applies these generalization ideas to the constant regularization setting.

Theorem 3.3.3: Generalization of online learners with constant regularization. *Let $\{l_t(w)\}_{t=1}^T$ be a sequence of loss functions with each of the form $l_t(w) = l_w(x_t, y_t)$ for some loss function $l_w(\cdot, \cdot)$ parameterized by w , with $\{(x_t, y_t)\}_{t=1}^T$ sampled i.i.d. from a fixed distribution. If $\{w_t\}_{t=1}^T$ is a sequence of iterates produced by the online subgradient method applied to objective functions $c_t(w) = r_t(w) + \frac{\lambda}{2}\|w\|^2$ with $l_t(w) \leq r_t(w)$, then the expected risk of $\bar{w} = \frac{1}{T} \sum_{t=1}^T w_t$ is bounded by*

$$E(l_{T+1}(\bar{w})) \leq \min_{w \in \mathcal{W}} \frac{1}{T} \sum_{t=1}^T r_t(w) + \left(G\|w^*\| \sqrt{2(1 + \log T)} + L\sqrt{\log \frac{1}{\delta}} \right) \sqrt{\frac{2}{T}}. \quad (3.50)$$

Proof. The proof of this theorem parallels the proof in Theorem 3.3.2, but using the regret bound of Theorem 3.2.3. □

Finally, we develop a cleaner and tighter generalization bound using the result from Theorem 3.2.5. Interestingly, in this case, since we shrink the regularization constant over time, the size of the competing class of hypotheses grows with the number of examples.

Theorem 3.3.4: Generalization of online learners with attenuated regularization. *Let $\{l_t\}_{t=1}^T$ be a sequence of risk functions with each $l_t(w)$ of the form $l_t(w) = l_w(x_t, y_t)$ for some loss function $l_w(\cdot, \cdot)$ parameterized by w , with $\{(x_t, y_t)\}_{t=1}^T$ sampled i.i.d. from a fixed distribution. If $\{w_t\}_{t=1}^T$ is a sequence of iterates produced by the online subgradient method applied to objective*

functions $c_t(w) = l_t(w) + \frac{\lambda}{2\sqrt{t}}\|w\|^2$, then the expected risk of $\bar{w} = \frac{1}{T} \sum_{t=1}^T w_t$ is bounded by

$$E(l_{T+1}(\bar{w})) \leq \min_{w \in \mathcal{W}_T} \frac{1}{T} \sum_{t=1}^T r_t(w) + \left(2\sqrt{2}G\|w^*\| + L\sqrt{\log \frac{1}{\delta}} \right) \sqrt{\frac{2}{T}}, \quad (3.51)$$

where $\mathcal{W}_T$ is a growing space of hypotheses bounded by radius $\frac{G\sqrt{T}}{\lambda}$.

Proof. We note that the effective radius each iteration is $R_t = \frac{G\sqrt{T}}{\lambda}$ based on the arguments of Theorem 3.1.3 using the regularizer $\lambda_t = \frac{\lambda}{\sqrt{t}}$. From here, proof of this theorem parallels the proof in Theorem 3.3.2, although using the regret bound of Theorem 3.2.5. $\square$

3.4 The batch setting

So far we have explored various online settings for both optimization and prediction, and we have demonstrated that they lead to strong generalization bounds when the stream of data is i.i.d. This section furthers the connection between the online and batch settings by showing that many traditional batch subgradient-based optimization techniques can be viewed as online algorithms for a particular sequence of objectives. This connection enables the reduction of batch optimization to online optimization thereby greatly simplifying their analysis.

In the batch optimizations setting, the optimizer is presented with a single objective function and the goal is to simply optimize it. There are two ways we can measure success. The first measure rates the convergence of the algorithm to the global optimum or optimal set. We analyze the subgradient method under this traditional setting in Section 3.4.3. This metric is useful if the goal is to find the global optimum quickly, but in machine learning, we often have a subtly different goal. The objective function in machine learning is typically a measure of how well the hypothesis fits the data. In this case, we primarily care about simply having a small objective value relative to the optimal value. Therefore, we strive for bounds how fast the algorithm converges minimizes the objective value. We study this setting in Section 3.4.2 using tools we developed in our analysis of online setting.

The batch optimization setting we consider here is a setting common to machine learning known as *regularized risk minimization* (Rifkin & Poggio, 2003). The objective function may be viewed as

Algorithm 4 The subgradient method

```
1: procedure SUBGRADIENT(  $c(w) = \frac{1}{N} \sum_{i=1}^N r_i(w) + \frac{\lambda}{2} \|w\|^2$ ,  $w_0$ ,  $\{\alpha_t\}_{t=1}^T$  )
2:   for  $t = 1, \dots, T$  do
3:     choose  $g_t \in \partial c(w_t)$ 
4:     set  $w_t = \mathcal{P}_{\mathcal{W}} [w_{t-1} - \alpha_t g_t]$ 
5:   end for
6:   return  $w_T$ 
7: end procedure
```

an average over the sequence of regularized objectives studied in Section 3.2.3. We restate it here for convenience.

$$c(w) = \frac{1}{N} \sum_{i=1}^N r_i(w) + \frac{\lambda}{2} \|w\|^2. \quad (3.52)$$

Objective function of this form are λ -strongly convex; by averaging the risk terms, we can guarantee that the gradient of that term is bounded by G as long as each risk term is individually bounded by G . Theorem 3.1.3, then tells us that our minimizer will be within a radius of $\frac{G}{\lambda}$ of the origin. This fact will be useful later on. Many of the results can be written more generally than stated, but we tune the notion to this special case for clarity and to fit better with the rest of this thesis.

We consider two algorithms for optimization. The first, shown in Algorithm 4, is called the *subgradient method* (Shor, 1985). At a high level, it amounts simply taking a small step in the direction of the objective's negative subgradient at each iteration, after which we project back onto the ball of radius $\frac{G}{\lambda}$ if needed. The second algorithm is called the *incremental* subgradient method (Nedic & Bertsekas, 2000). We list it explicitly in Algorithm 5; it is similar to the subgradient method, except rather than summing the gradient contributions from each risk term before taking a step, this algorithm exploits the structure of the objective by taking a small step for each gradient contribution in turn before computing the next. We will see that this subtle difference gives approximately a factor N speedup in convergence.

Algorithm 5 The incremental subgradient method

```
1: procedure INCSUBGRAD(  $c(w) = \frac{1}{N} \sum_{i=1}^N r_i(w) + \frac{\lambda}{2} \|w\|^2$ ,  $w_0$ ,  $\{\alpha_t\}_{t=1}^T$  )
2:   for  $t = 1, \dots, T$  do
3:     set  $w_t^0 = w_{t-1}$ 
4:     for  $i = 1, \dots, N$  do
5:       let  $g_t^i = \nabla r_i(w_t) + \frac{\lambda}{N} w_t$ 
6:       set  $w_t^i = \mathcal{P}_{\mathcal{W}} [w_t^{i-1} - \alpha_t g_t^i]$ 
7:     end for
8:     set  $w_t = w_t^N$ 
9:   end for
10:  return  $w_T$ 
11: end procedure
```

3.4.1 Reductions to online learning

Both algorithms listed here, the subgradient method shown in Algorithm 4 and the incremental subgradient method shown in Algorithm 5, can be viewed as the online subgradient method applied to a sequence of objectives with a particular form. We explicitly list the connections here. These connections will allow us to use the analysis from the online subgradient method to derive batch optimization convergence bounds.

1. **Subgradient method reduction.** Form a sequence of objectives whose elements are all the same objective: $\{c_t(w)\}_{t=1}^T$ with $c_t(w) = c(w) = \frac{1}{N} \sum_{i=1}^N r_i(w) + \frac{\lambda}{2} \|w\|^2$ for all t . The online subgradient method (Algorithm 3) applied to this objective sequence is equivalent to the subgradient method (Algorithm 4).
2. **Incremental subgradient method reduction.** Form a sequence of objectives that cycles T times through the sequence of objectives constituting the batch objective: $\{\{c_t^i(w)\}\}_{t=1}^T$ where $c_t^i(w) = r_i(w) + \frac{\lambda}{2N} \|w\|^2$. Then the online subgradient method (Algorithm 3) applied to this objective sequence is equivalent to the subgradient method (Algorithm 4).

3.4.2 Batch convergence bounds

We are now in a position to bound the number of iterations to achieve ϵ -convergence, which we define as the number of iterations required to achieve an objective value within ϵ of the minimum

objective value. We begin with a convergence bound for the subgradient method.

Theorem 3.4.1: Convergence analysis of the subgradient method. *Consider regularized risk functions of the form $c(w) = \frac{1}{N} \sum_{i=1}^N r_i(w) + \frac{\lambda}{2} \|w\|^2$. Let $\{w_t\}_{t=1}^T$ be the sequence of iterates returned by running the subgradient method shown in Algorithm 4 for T iterations using step size sequence $\{\frac{1}{\lambda t}\}_{t=1}^T$. Then we get the following ϵ -convergence guarantee:*

$$T = \tilde{O}\left(\frac{2G^2}{\epsilon\lambda}\right) \quad (3.53)$$

Proof. The subgradient method is equivalent to the online subgradient method when the latter is shown a sequence of objective function that are all the same. Therefore, we achieve the following online optimization regret bound:

$$\sum_{t=1}^T c(w_t) \leq \sum_{t=1}^T c(w^*) + \frac{2G^2}{\lambda} (1 + \log T). \quad (3.54)$$

Dividing this objective through by T and noting that $c(w_T^*) \leq \frac{1}{T} \sum_{t=1}^T c(w_t)$, where $w_T^* = \arg \min_{t=1, \dots, T} c(w_t)$, brings us to the following bound:

$$c(w_T^*) \leq c(w^*) + \frac{2G^2}{\lambda} \left(\frac{1 + \log T}{T} \right). \quad (3.55)$$

where $w^* = \arg \min_{w \in \mathcal{W}} c(w)$. Solving in terms of T for $\frac{2G^2}{\lambda} \left(\frac{1 + \log T}{T} \right) = 0$ (dropping the log terms), gives the desired result. $\square$

Theorem 3.4.2: Convergence analysis of the incremental subgradient method. *Consider regularized risk functions of the form $c(w) = \frac{1}{N} \sum_{i=1}^N r_i(w) + \frac{\lambda}{2} \|w\|^2$. Let $\{\{w_t^i\}_{i=1}^N\}_{t=1}^T$ be the sequence of iterates returned by running the incremental subgradient method shown in Algorithm 5 for T iterations using step size sequence $\{\frac{1}{\lambda t}\}_{t=1}^T$. Then we get the following ϵ -convergence guarantee:*

$$T = \tilde{O}\left(\frac{1}{N} \frac{2G^2}{\epsilon\lambda}\right). \quad (3.56)$$

Proof. The incremental subgradient method is equivalent to the online subgradient method when the latter is shown a sequence of objective functions that cycle T times through the constituent objectives $c_i(w) = r_i(w) + \frac{\lambda}{2N}\|w\|^2$. Therefore, we achieve the following online optimization regret bound:

$$\sum_{t=1}^T \sum_{i=1}^N \left(\frac{1}{N} r_i(w_t^i) + \frac{\lambda}{2N} \|w_t^i\|^2 \right) \leq \sum_{t=1}^T \sum_{i=1}^N \left(\frac{1}{N} r_i(w^*) + \frac{\lambda}{2N} \|w^*\|^2 \right) + \frac{1}{2} \sum_{t=1}^T \sum_{i=1}^N \alpha_t \|g_t^i\|^2, \quad (3.57)$$

where $w^* = \arg \min_{w \in \mathcal{W}} \frac{1}{N} \sum_{i=1}^N r_i(w) + \frac{\lambda}{2} \|w\|^2$. Each gradient term is bounded as $\|g_t^i\| \leq \frac{G}{N} + \frac{\lambda}{N} \|w_t\| \leq \frac{2G}{N}$, where the second inequality comes because we know that $\|w_t\| \leq \frac{G}{\lambda}$ from Theorem 3.1.3. This observation allows us to bound the last term of Inequality 3.57 as

$$\frac{1}{2} \sum_{t=1}^T \sum_{i=1}^N \alpha_t \|g_t^i\|^2 \leq \sum_{t=1}^T N \alpha_t \left(\frac{2G}{N} \right)^2 = \frac{2G^2}{N} \sum_{t=1}^T \frac{1}{\lambda t} \quad (3.58)$$

$$\leq \frac{2G^2}{N} (1 + \log T). \quad (3.59)$$

Now define $w_T^* = \arg \min_{t=1, \dots, T} \frac{1}{N} \sum_{i=1}^N r_i(w_t) + \frac{\lambda}{2} \|w_t\|^2$ and $w^* = \arg \min_{w \in \mathcal{W}} \frac{1}{N} \sum_{i=1}^N r_i(w) + \frac{\lambda}{2} \|w\|^2$. We can bound the first term in Inequality 3.57 below by

$$T \sum_{i=1}^N \left(\frac{1}{N} r_i(w_T^*) + \frac{\lambda}{2N} \|w_T^*\|^2 \right) \leq \sum_{t=1}^T \sum_{i=1}^N \left(\frac{1}{N} r_i(w_t^i) + \frac{\lambda}{2N} \|w_t^i\|^2 \right). \quad (3.60)$$

Plugging these bound into the above regret expression and dividing through by T gives

$$c(w_T^*) \leq c(w^*) + \frac{2G^2}{\lambda N} \left(\frac{1 + \log T}{T} \right), \quad (3.61)$$

Solving in terms of T for $\frac{2G^2}{\lambda N} \left(\frac{1 + \log T}{T} \right) = 0$ (dropping the log terms), gives the desired result. $\square$

The results presented in Theorems 3.4.1 and 3.4.2 are very similar, except they differ by a factor of N . These results suggest that for regularized risk functions of the form given in Equation 3.52, the incremental subgradient method should converge in about a factor of N fewer iterations than the general subgradient method.

3.4.3 Traditional Analysis

We now take a more traditional approach to convergence analysis and bound the rate of convergence of the subgradient method to the optimum through the hypothesis space $\mathcal{W}$. When the objective

function is convex and differentiable (and the second derivatives are everywhere bounded above), it is well known that the conventional gradient descent algorithms with line search converges at a linear rate to the global minimum. Specifically, we say the algorithm converges linearly if it produces a sequence of iterates $\{w_t\}_{t=0}^T$ with the property that there exists some positive constant $c \in [0, 1)$ for which

$$\forall t, \quad \|w_{t+1} - w^*\| \leq c \|w_t - w^*\| \leq c^{t+1} \|w_0 - w^*\|, \quad (3.62)$$

where w^* is the global minimizer. In other words, the error between the current iterate and the global minimizer reduces exponentially fast.

A natural question is whether this same property holds true of nondifferentiable objective functions under the subgradient method. There is one primary difference between the differentiable and nondifferentiable cases that affects the analyzing and implementing the subgradient method. The subgradient method cannot utilize a line search subroutine because the line search may only drive the hypothesis to a kink (point of nondifferentiability) in the function from which it cannot easily escape (Shor, 1985). Because of this phenomenon, one must choose in advance a fixed step size sequence. The subgradient method is therefore not strictly a descent method; the objective value under the subgradient method may increase slightly at times. However, under mild conditions on the form of the step size sequence, the algorithm is guaranteed to converge to the global minimum.

Traditional analysis shows that the subgradient method converges linearly to a region of the global minimum under a small constant step size. To guarantee global convergence to the minimum, the algorithm must use a diminishing step size sequence such as $\{r/\sqrt{t}\}_{t=1}^\infty$ or $\{r/t\}_{t=1}^\infty$. Unfortunately, the bounds on the convergence rate for these cases are only sublinear convergence, but the approximate optimization offered by the constant step size sequence is sufficient for many applications. In particular, while learning algorithms often reduce learning to optimization, the true goal is typically to generalize from the given data. Approximate optimization therefore, may at times be preferred as it can help prevent problems with overfitting which often plague over-optimization.

For completeness, we prove here that the subgradient algorithm converges linearly to a region of the global minimum under a fixed constant step size. This proof closely follows the proof of an

analogous result originally presented in (Nedic & Bertsekas, 2000). Following their presentation, to prove linear convergence, we require that the objective be strongly convex.

The next theorem shows that the subgradient method applied to strongly convex functions converges at a linear rate to a tight region around the global minimum when a sufficiently small constant step size is used.

Theorem 3.4.3: Linear convergence of the subgradient method. *Consider a regularized risk objective function of the form $c(w) = \frac{1}{N} \sum_{i=1}^N r_t(w) + \frac{\lambda}{2} \|w\|^2$. Under a constant step size $0 < \alpha \leq \frac{1}{\lambda}$, the subgradient algorithm will converge at a linear rate to a region around the minimum of radius $2G\sqrt{\frac{\alpha}{\lambda}}$.*

Proof. Expanding the distance from w_{t+1} to the global minimum w^* , we get

$$\|w_{t+1} - w^*\|^2 = \|\mathcal{P}_{\mathcal{W}}[w_t - \alpha g_t] - w^*\|^2 \leq \|w_t - \alpha g_t - w^*\|^2 \quad (3.63)$$

$$= \|w_t - w^*\|^2 - 2\alpha g_t^T(w_t - w^*) + \alpha^2 \|g_t\|^2. \quad (3.64)$$

Since $c(w)$ is λ -strongly convex, we can use the strong-convexity and subgradient bounds, along with a bound on the gradient $\nabla \left(\frac{1}{N} \sum_{i=1}^N r_t(w) + \frac{\lambda}{2} \|w\|^2 \right) \leq \tilde{G}$, to rewrite the final expression as

$$\|w_{t+1} - w^*\|^2 \leq (1 - \alpha\lambda) \|w_t - w^*\|^2 + \alpha^2 \tilde{G}^2 \quad (3.65)$$

$$= (1 - \alpha\lambda)^{t+1} \|w_0 - w^*\|^2 + \alpha^2 \tilde{G}^2 \sum_{\tau=0}^t (1 - \alpha\lambda)^\tau \quad (3.66)$$

$$\leq \mu^{t+1} \|w_0 - w^*\|^2 + \frac{\alpha \tilde{G}^2}{\lambda}, \quad (3.67)$$

where $\mu = 1 - \alpha\lambda$. This inequality demonstrates linear convergence to a region. Taking the square root of both sides of this inequality and taking the limit as t approaches infinity gives the radius of this region:

$$\lim_{t \rightarrow \infty} \|w_t - w^*\| = \lim_{t \rightarrow \infty} \sqrt{\mu^{t+1} \|w_0 - w^*\|^2 + \frac{\alpha \tilde{G}^2}{\lambda}} = \tilde{G} \sqrt{\frac{\alpha}{\lambda}}, \quad (3.68)$$

where $\mu^{t+1} \|w_0 - w^*\|^2$ vanishes because $0 < \alpha \leq \frac{1}{\lambda}$ implies $0 \leq \mu < 1$. Finally, plugging in $\tilde{G} = 2G$ found by using the relation $\|w\| \leq \frac{G}{\lambda}$ using Theorem 3.1.3, gives the desired result. $\square$

Chapter 4

Functional Gradient Optimization

Much of learning theory, particularly for online learning, focuses on the special case of linear hypotheses. The literature supports this setting with substantial theory; linear algorithms perform very well relative to the best available hypothesis. Unfortunately, extracting features that form a sufficiently expressive hypothesis space for a particular domain remains difficult.

The machine learning community, therefore, spends substantial effort to develop nonlinear frameworks that reduce demands on carefully engineered feature extraction. Traditionally, three competing mainstream classes of nonlinear supervised learning techniques have taken precedent over the past two decades. These approaches roughly segment into direct neural network training, learning in reproducing kernel Hilbert spaces (RKHS), and boosting.

Artificial neural networks have a long history of application, particularly in engineering fields, as a result of their highly expressive multi-layered network architecture. Unfortunately, these networks are prone to overfitting; the breadth of the hypothesis space and the nonconvexity of the optimization problem that governs learning are problematic in practice. Recent research into unsupervised initialization techniques for deep networks have shown promise (Hinton, Osindero, & Teh, 2006), but this research is still very new; many open question regarding its practicality and reliability remain.

These problems with nonconvexity, punctuated by the quick success of support vector machines

in the mid '90s, led to a wave of interest in learning within reproducing kernel Hilbert spaces. Extensive research has built strong theory in support of their empirical success, and these algorithms currently enjoy ubiquitous application spanning many fields. Unfortunately, large-scale learning remains difficult. In practice, the size of the hypothesis often grows linearly with the number of training examples as suggested by the representer theorem (Schölkopf *et al.*, 2000). Structured prediction problems emphasize this problem; the kernel representation can grow linearly with the number of parts (e.g. the number of nodes and edges in an associative Markov random field or the number of state-action pairs in a Markov decision process) forming the structured domain.

Since the discovery of AdaBoost in 1996 (Freund & Schapire, 1995), boosting techniques have grown steadily in popularity. AdaBoost was originally developed using *probably approximately correct* (PAC) arguments demonstrating that an ensemble of weak learners can form a *strong* learner that outperforms each constituent learner individually. However, at the turn of the century, Mason *et al.* (1999) and Friedman (1999a) independently proposed a new interpretation of boosting as functional gradient descent, which generalized it to a large class of loss functions found in the machine learning literature. Because of this innovative interpretation, boosting, today, holds a reputation as a practical widespread solution to nonlinear learning.

Later work found that gradient boosting solutions approximately build sparse solutions in the space of ensembles (Rosset, Zhu, & Hastie, 2004). This observation supports gradient boosting as our method of choice for developing nonlinear algorithms for inverse optimal control. Below we review a generalized interpretation of gradient boosting in Section 4.1.1 before developing two novel nonlinear algorithms that utilize functional gradients. Section 4.2 derives the first of these algorithms, which we call *exponentiated functional gradient descent*. This algorithm operates in an exponentiated hypothesis space that better represents our prior beliefs on functions. Much of the theory behind gradient boosting suggests the use of small constant sized-steps each iteration. The resulting hypotheses, therefore, perform well in practice, but tend to be highly redundant. We subsequently derive our second algorithm, called *the functional bundle method*, in Section 4.3 to address redundancy in representation. Without relinquishing generalization performance, this algorithm demonstrates fast convergence which, in practice, translates to compact representations

and few objective function evaluations.

4.1 Gradient descent through Euclidean function spaces

There are a number of formalizations of Boosting as gradient descent through a function space. Friedman (1999a) takes a statistical approach and shows that a least-squares regression provides a consistent estimate of the true functional gradient of a statistical risk function when only a sample of points from the data distribution is available. Alternatively, Mason *et al.* (1999) address the empirical risk minimization problem directly. At each iteration, their algorithm evaluates a functional gradient and finds a most correlating function to use as a search direction. This approach effectively separates the statistical learning problem from its reduction to optimization. The authors, however, derive the algorithm specifically for using a set of classifiers as candidate search directions (rather than considering a more general, potentially continuous, space of functions), and they build their arguments around a notion of inner product that measures correlation only in terms of the functions evaluated at the given data points.¹

This section unifies both of these ideas by deriving a form of projected Euclidean functional gradient descent. This algorithm approaches the problems from the perspective of Mason *et al.* (1999) and operates directly on the empirical risk function, but it explicitly evaluates a Euclidean functional gradient at each iteration and projects it onto an arbitrary space of candidate functions. Our analysis suggests implementing the projection operation as a least-squares minimization over the data similar to the procedure proposed by Friedman (1999a). We additionally show that this projection reduces to the correlation criterion used by Mason *et al.* (1999) when the space of candidate search directions is restricted to a space of classifiers.

In (Ratliff *et al.*, 2006), we provide an alternative derivation of the Euclidean functional gradient ideas we present here. That derivation generalizes the derivation of Mason *et al.* (1999) by more directly utilizing the notion of a Euclidean functional gradient as a linear combination of generalized functions (Hassani, 1998), but it remains tailored specifically to projecting onto a set of classifiers

¹Their inner product is technically degenerate in that it forms a norm which evaluates to zero on a collection of functions that are not necessarily the zero function.

who take values in $\{-1, 1\}$.

4.1.1 Euclidean functional gradient projections

Gradients of functionals defined over Euclidean function spaces (i.e. $\mathcal{L}^2$ spaces of square-integrable functions) are often defined in terms of generalized functions (Hassani, 1998) such as the Dirac delta function. The theory of generalized functions has been rigorously established through multiple methods (distribution theory, generalized functions, non-standard analysis), and they are frequently used in theoretical physics, electrical engineering, and related fields. However, the full machinery of convex optimization for functionals defined over these spaces remains an area of research.

On the other hand, the theory of functional gradients in an RKHS is rigorously defined and used throughout the machine learning literature (Schölkopf & Smola, 2002; Kivinen, Smola, & Williamson, 2002; Bagnell & Schneider, 2003b; Ratliff & Bagnell, 2007). We derive our operations in this section by noting that the dual to a Euclidean space of functions (i.e. the space of Euclidean functional gradients) is the limit point of the dual spaces of a continuum of reproducing kernel Hilbert spaces. Let $g_\sigma(x, x')$ be a normalized Gaussian of the form $g_\sigma(x, x') = \frac{1}{Z_\sigma} e^{-\frac{\|x-x'\|^2}{2\sigma^2}}$, where the constant Z_σ is normalizer ensuring that $g_\sigma(x, x')$ integrates to 1, and let $p(x)$ be a probability density function with support over the domain $\mathcal{X}$. Then the space of Euclidean functional gradients is the limit (as σ approaches 0) of the RKHS² formed by the kernel³ $k_\sigma(x, x') = \frac{g_\sigma(x, x')}{p(x)p(x')}$ (Schölkopf & Smola, 2002). This class of kernels, in the limit as $\sigma \rightarrow 0$, approaches the delta function for the weighted L_2 inner product defined as $\langle \phi, \psi \rangle = \int_{\mathcal{X}} \phi(x)\psi(x)p(x)dx$ for functions $\phi, \psi \in \mathcal{L}^2$ (Hassani, 1998). This section makes extensive use of RKHS functional gradients to aid in deriving the least-squares technique for projecting a Euclidean functional gradient onto a space of functions; we do not review RKHS functional gradients in detail here, but an introduction to these concepts can be found in (Bagnell, 2004) and (Ratliff & Bagnell, 2007).

²All Hilbert spaces are reflexive, which means that the dual space is isometric to space itself (Hassani, 1998). From here on out, we therefore refer to both the RKHS and its dual space as the RKHS.

³This kernel is positive definite because $g_\sigma(x, x')$ is a scaled radial basis function which is known to be positive definite, and all Gram matrices of this kernel take the form $\mathbf{P}^{-\frac{1}{2}} \mathbf{A} \mathbf{P}^{-\frac{1}{2}}$, where $\mathbf{A}$ is the positive definite Gram matrix of $g_\sigma(x, x')$ and $\mathbf{P}$ is the positive definite diagonal matrix containing evaluations of $p(x)$.

In an RKHS formed by the kernel $k_\sigma(x, x')$, functional gradients take the general form

$$\nabla_k \mathcal{F}[f] = \sum_{i=1}^N \nabla_k l(f(x_i)) = \sum_{i=1}^N l'_i(f(x_i)) k_\sigma(x_i, \cdot) \quad (4.1)$$

$$= \sum_{i=1}^N l'_i(f(x_i)) \frac{g_\sigma(x_i, \cdot)}{\sqrt{p(x_i)p(\cdot)}} \quad (4.2)$$

(Schölkopf & Smola, 2002; Ratliff & Bagnell, 2007). In this case, we take $p(x)$ to be the data distribution from which the points $\{x_i\}_{i=1}^N$ were sampled. We can represent the Euclidean functional gradient in terms of generalized functions by evaluating its limit as the RKHS approaches the dual to the Euclidean space of functions. In the limit, we find

$$\lim_{\sigma \rightarrow 0} \nabla_k \mathcal{F}[f](x) = \sum_{i=1}^N l'_i(f(x_i)) \frac{1}{\sqrt{p(x_i)p(x)}} \delta(x - x_i) = \sum_{i=1}^N l'_i(f(x_i)) \frac{1}{p(x_i)} \delta(x - x_i), \quad (4.3)$$

where we leverage properties of the Delta function in making the final transformation.

If we denote, for notational convenience, $\eta_i = l'_i(f(x_i))$, we can write the functional least-squares projection of the RKHS functional gradient onto a given space of functions $\mathcal{H}$, performed with respect to the weighted L_2 inner product, as

$$\begin{aligned} h^* &= \arg \min_{h \in \mathcal{H}} \int_{\mathcal{X}} \left(h(x) - \sum_{i=1}^N \eta_i \frac{g_\sigma(x_i, x)}{\sqrt{p(x_i)p(x)}} \right)^2 p(x) dx \\ &= \arg \min_{h \in \mathcal{H}} \int_{\mathcal{X}} h(x)^2 p(x) dx - 2 \int_{\mathcal{X}} h(x) \sum_{i=1}^N \eta_i \frac{g_\sigma(x_i, x)}{\sqrt{p(x_i)p(x)}} p(x) dx + \int_{\mathcal{X}} \left(\sum_{i=1}^N \eta_i \frac{g_\sigma(x_i, x)}{\sqrt{p(x_i)p(x)}} \right)^2 p(x) dx \\ &= \arg \min_{h \in \mathcal{H}} \left[\int_{\mathcal{X}} h(x)^2 p(x) dx - 2 \sum_{i=1}^N \frac{\eta_i}{\sqrt{p(x_i)}} \int_{\mathcal{X}} h(x) g_\sigma(x_i, x) \sqrt{p(x)} dx \right]. \end{aligned}$$

We can drop the last (squared) term after the second step because it is independent of h . Since we have samples $\{x_i\}_{i=1}^N$ from the data distribution $p(x)$, we can approximate the first terms as

$$\int_{\mathcal{X}} h(x)^2 p(x) dx \approx \frac{1}{N} \sum_{i=1}^N h(x_i)^2. \quad (4.4)$$

Additionally, in the limit, as σ approaches 0 (where the RKHS approaches the Euclidean function

space), we have the relation

$$\int_{\mathcal{X}} h(x) g_{\sigma}(x_i, x) \sqrt{p(x)} dx \rightarrow h(x_i) \sqrt{p(x_i)}. \quad (4.5)$$

Therefore, we get

$$h^* \approx \arg \min_{h \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^N h(x_i)^2 - \frac{2}{N} \sum_{i=1}^N \tilde{\eta}_i h(x_i) \quad (4.6)$$

$$= \arg \min_{h \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^N (h(x_i) - \tilde{\eta}_i)^2, \quad (4.7)$$

where $\tilde{\eta}_i = N\eta_i$ (i.e. $\tilde{\eta}_i$ is proportional to η_i). These observations suggest that (up to a constant scaling) we can implement the orthogonal (least-squares) functional L_2 projection of a Euclidean functional gradient onto a predefined space of functions by solving the least-squares problem defined over the data set $\{(x_i, \tilde{\eta}_i)\}_{i=1}^N$. In practice, we typically replace $\tilde{\eta}_i$ by η_i since many nonlinear least-squares function approximators are agnostic to constant scalings of the data.

Typically, we view $\mathcal{H}$ as a space of candidate search directions through the function space. When this space is a continuous function space, we can approximately project the Euclidean functional gradient onto $\mathcal{H}$ using least-squares. Alternatively, when a $\mathcal{H}$ is a space of classifiers c defined by $c : \mathcal{X} \rightarrow \{-1, 1\}$, the least-squares objective has the following interpretation:

$$\frac{1}{N} \sum_{i=1}^N (c(x_i) - \eta_i)^2 = \frac{1}{N} \left(\sum_{i=1}^N c(x_i)^2 - 2\eta_i c(x_i) + \eta_i^2 \right) \quad (4.8)$$

$$\propto -\frac{1}{N} \sum_{i=1}^N \eta_i c(x_i), \quad (4.9)$$

since $c(x_i)^2 = 1$ and since the term $\frac{1}{N} \sum_{i=1}^N \eta_i^2$ is independent of $c(\cdot)$. This final expression is the inner product definition used by Mason *et al.* (1999). Therefore, the idea of finding the most correlating function from a given class of candidate search directions is a special case of our projection framework. In the general setting, however, we require the least-squares interpretation since the inner product criterion, by itself, may not have a finite maximizer.

Algorithm 6 Projected Euclidean functional gradient descent (intuition)

```
1: procedure PROJFUNGRAD( objective functional  $F[\cdot]$ , initial function  $h_0$ , step size sequence  
    $\{\gamma_t\}_{t=1}^T$  )  
2:   for  $t = 1, \dots, T$  do  
3:     Evaluate the functional gradient  $g_t$  of the objective functional at the current hypothesis  
        $f_t = -\sum_{\tau=0}^{t-1} \gamma_\tau h_\tau$ .  
4:     Project the functional gradient onto the space of candidate search directions using least-  
       squares to find  $h_t = h^* \in \mathcal{H}$ .  
5:     Take a step in the negative of that direction forming  $f_{t+1} = f_t - \gamma_t h_t = -\sum_{\tau=0}^t \gamma_\tau h_\tau$ .  
6:   end for  
7: end procedure
```

Algorithm 6 provides an intuitive presentation of the projected Euclidean functional gradient descent algorithm.

4.1.2 Euclidean functional gradients as data sets

The arguments above suggest that we can view a Euclidean functional gradient as a regression data set. This data set suggests where and how strongly the function should be increased or decreased at a collection of points in the feature space. For instance, let $\mathcal{D} = \{x_i, y_i\}_{i=1}^N$ be such a data set.⁴ Each y_i is a real number suggesting how the function should be modified at x_i . If y_i is strongly positive, then this data point suggests a strong increase in the function at x_i . Alternatively, if y_i is strongly negative, then the function should be strong decreased at that point. Moreover, if y_i is zero, or close to zero, the data point indicates that the function should not be modified at all. Black box regression algorithms generalize these suggested updates to the rest of the domain $\mathcal{X}$ and return to us a function that implements the suggested modifications when added to the current hypothesis. Figure 4.1 demonstrates pictorially the intuitive affect of a projected Euclidean functional gradient update. On the left is a plot of the original hypothesized function superimposed with the Euclidean functional gradient data set of suggested modifications. The rightmost plot depicts the new hypothesis that results from the update.

⁴In projected Euclidean functional gradient descent, we take a step in the negative direction of the projected functional gradient at each iteration. y_i should, therefore, be interpreted as the negative of the η_i used in Section 4.1.1.

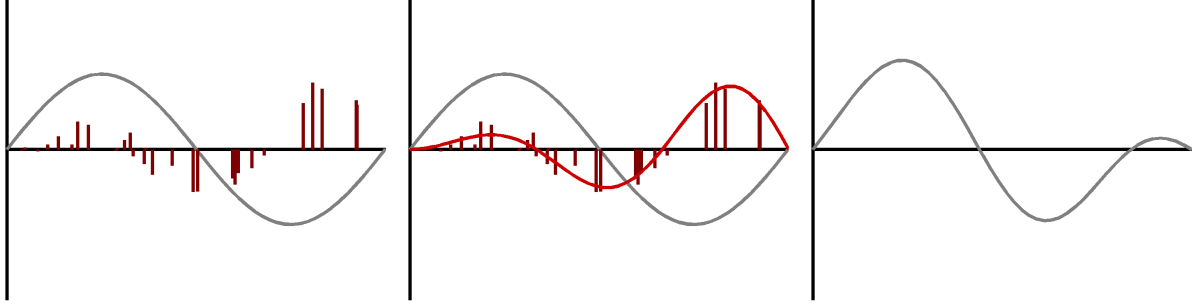


Figure 4.1: This figure shows a pictorial representation of the action of a functional gradient data set. The left plot shows the original function in gray along with a functional gradient data set indicated where and to what extent the function should be modified. The second plot includes a nonlinear regressor that generalizes from that data set to the rest of the domain. Finally, the right-most image shows the result of taking a functional gradient step: the nonlinear regressor in the second plot is simply added to the original function effectively implementing the discrete set of suggested modifications.

4.1.3 A generalized class of objective functions

In later chapters, particularly in Chapter 6, we use the functional gradient techniques developed here in a more general setting where the distribution of domain points $\{x_i\}$ seen by each risk term may change every iteration. (Concretely, for IOC applications, the behavior of the planner changes every iteration as we converge toward the desired behavior. This causes the distribution over feature vectors encountered along the optimal planned path to also change at every iteration.) In the above derivation of the least-squares functional gradient projection, the changing distribution of domain points means that the specific choice of $p(x)$ changes with every iteration. Intuitively, we project the functional gradient onto the space of functions using a weighted L_2 inner product that reflects the distribution of domain points seen during the current iteration.

4.1.4 Comparing functional gradients techniques

Above we derived our projected Euclidean functional gradient algorithm in terms of a limit of functional gradients evaluated in an RKHS as that RKHS approaches the space of Euclidean functional gradients. However, there is a long precedence for performing functional gradient descent directly within an RKHS; evidence supporting it as an efficient and simple optimization technique continues to grow (Schölkopf & Smola, 2002; Kivinen, Smola, & Williamson, 2002; Bagnell & Schneider,

2003b; Ratliff & Bagnell, 2007). This section compares the performance of the projected Euclidean functional gradient descent algorithm to the performance of functional gradient descent through an RKHS. We show empirically that projected Euclidean directions can be more effective search directions than RKHS functional gradients. We additionally offer intuition and some simple analysis supporting why this observation is true.

The projected Euclidean functional gradient is, in a sense, agnostic to the choice of parameterization, a property reminiscent of covariant gradient techniques for optimization (Amari & Nagaoka, 2000; ichi Amari, 1998; Kakade, 2002; Bagnell & Schneider, 2003a) studied in machine learning. Rather than using the chain rule to push the gradient through the function parameterization, the projected Euclidean functional gradient defers the choice of parameters to the function approximator implementing the projection. Therefore, whether they be constructed as kernel machines, neural networks, or decision trees, the resulting search direction is similar regardless of its parameterization. Intuitively, these search directions generalize errors signals extracted directly from the governing loss function.

To explore this property empirically, we implement both the project Euclidean functional gradient descent algorithm as well as the kernel functional gradient descent algorithm for optimizing a *kernel logistic regression* (KLR) objective functional for binary classification on the USPS data set. The kernel functional gradient is common in this setting since the objective is defined over an RKHS. Throughout this presentation, we denote the kernel functional gradient of a functional $\mathcal{F}[\cdot]$ as $\nabla_k \mathcal{F}[\cdot]$ and the Euclidean functional gradient of that functional as $\nabla_f \mathcal{F}[\cdot]$.

In this setting, we are given classification data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ where the classes can take the binary values $y_i \in \{-1, 1\}$. KLR models the probability of a class as $p(y = y_i | x_i) = \frac{e^{-y_i f(x_i)}}{1 + e^{-y_i f(x_i)}}$, where f is in the RKHS formed by kernel $k(\cdot, \cdot)$ which we denote $\mathcal{H}_k$. The objective function, derived as the negative log-likelihood of the data using a Gaussian process prior, is given by

$$\mathcal{F}_{\text{klr}}[f] = \sum_{i=1}^N \left[y_i f(x_i) + \log \left(1 + e^{-y_i f(x_i)} \right) \right] + \frac{\lambda}{2} \|f\|_k^2. \quad (4.10)$$

Here, we define the RKHS norm in terms of the RKHS inner product $\|f\|_k^2 = \langle f, f \rangle_k$ (Schölkopf

& Smola, 2002). The kernel functional gradient is straightforward to compute using the formulas in (Ratliff & Bagnell, 2007):

$$\nabla_k \mathcal{F}_{\text{klr}}[f] = \sum_{i=1}^N y_i \left(1 - \frac{e^{-y_i f(x_i)}}{1 + e^{-y_i f(x_i)}} \right) k(x_i, \cdot) + \lambda f \quad (4.11)$$

$$= \sum_{i=1}^N y_i p(y \neq y_i | x_i) + \lambda f. \quad (4.12)$$

Functional gradient descent through the RKHS (starting at the zero function) finds a hypothesis of the form $f = \sum_{i=1}^N \alpha_i k(x_i, \cdot)$ (i.e. a linear combination of $\{k(x_i, \cdot)\}_{i=1}^N$). Equation 4.11, therefore, reduces to $\nabla_k \mathcal{F}_{\text{klr}}[f] = \sum_i b_i k(x_i, \cdot)$, where $b_i = y_i p(y \neq y_i | x_i) + \alpha_i$.

Since the regularization term is defined as an RKHS norm, it is not immediately clear how to derive the Euclidean functional gradient of this objective. However, by invoking the representer theorem (Schölkopf *et al.*, 2000), we can parameterize the function as a linear combination of kernels centered at the data points $f(\cdot) = \sum_{j=1}^N \alpha_j k(x_j, \cdot)$ and rewrite the regularizer as

$$\|f\|_k^2 = \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j k(x_i, x_j) = \sum_{i=1}^N \alpha_i f(x_i) \quad (4.13)$$

Deriving the Euclidean functional gradient of the regularizer in light of Equation 4.13 is straightforward: $\nabla_f \|f\|_k^2 = \sum_{i=1}^N \alpha_i \delta_{x_i}$. The full Euclidean functional gradient of Equation 4.10 is therefore

$$\nabla_f \mathcal{F}_{\text{klr}}[f] = \sum_{i=1}^N (y_i p(y \neq y_i | x_i) + \alpha_i) \delta_{x_i} \quad (4.14)$$

$$= b_i \delta_{x_i}. \quad (4.15)$$

In these experiments, we implement the Euclidean functional gradient projection step using *regularized least squares* (RLS) with data set $\{x_i, b_i\}_{i=1}^N$ Schölkopf & Smola (2002). Let $\mathbf{K}$ denote the kernel matrix formed by evaluating the kernel at all pairs of points x_i . RLS applied to this

Algorithm 7 Kernel functional gradient descent for KLR

```
1: procedure KERNELKLR(  $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N, \lambda > 0, T$  )
2:   set  $\alpha = 0$ 
3:   for  $t = 1, \dots, T$  do
4:     for  $i = 1, \dots, N$  do
5:       compute  $p_i = p(y \neq y_i | x_i, \alpha)$ 
6:       set direction components  $b_i = y_i p_i + \lambda \alpha_i$ 
7:     end for
8:     perform line search in direction  $-\mathbf{b}$  to update  $\alpha$ 
9:   end for
10: end procedure
```

Algorithm 8 Projected Euclidean functional gradient descent for KLR

```
1: procedure EUCLIDEANKLR(  $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N, \lambda > 0, \gamma > 0, T$  )
2:   set  $\alpha = 0$ 
3:   cache matrix inverse  $\mathbf{B} = (\mathbf{K} + \gamma \mathbf{I})^{-1}$ 
4:   for  $t = 1, \dots, T$  do
5:     for  $i = 1, \dots, N$  do
6:       compute  $p_i = p(y \neq y_i | x_i, \alpha)$ 
7:       set direction components  $b_i = y_i p_i + \lambda \alpha_i$ 
8:     end for
9:     transform direction  $\tilde{\mathbf{b}} = \mathbf{B}\mathbf{b}$ 
10:    perform line search in direction  $-\tilde{\mathbf{b}}$  to update  $\alpha$ 
11:  end for
12: end procedure
```

data set has the following closed form solution:

$$\tilde{\mathbf{b}} = (\mathbf{K} + \gamma \mathbf{I})^{-1} \mathbf{b}, \quad (4.16)$$

where $\gamma > 0$ is the regularization constant of the RLS approximator and $\mathbf{I}$ is an $N \times N$ identity matrix. Algorithms 7 and 8 depict functional gradient descent through the RKHS and the projected Euclidean functional gradient descent algorithms, respectively, applied to the KLR objective.

Importantly, by using RLS to implement the projection step, the representer theorem says that the search direction approximating the Euclidean functional gradient resides in the same subspace as the kernel functional gradient. This observation allows us to directly compare the quality of these search directions.

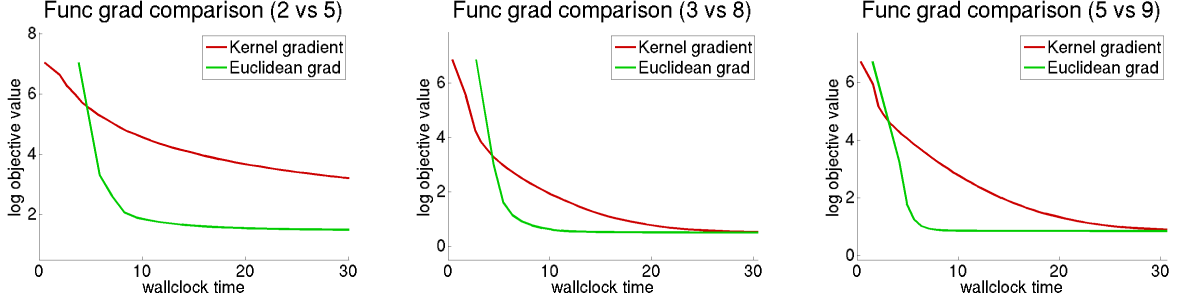


Figure 4.2: This figure compares the projected Euclidean functional gradient descent algorithm (green) discussed in this chapter (see Section 4.1.1) to functional gradient descent through an RKHS (red) on binary classification problems using kernel logistic regression (KLR). The Euclidean functional gradient projection step was implemented using regularized least squares (RLS). Although the inner product in this space is defined by the kernel (favoring the kernel functional gradient) the projected Euclidean functional gradients prove to be far better search directions. The x-axis of these plots gives the computation time. See Section 4.1.4 for additional details.

Figure 4.2 compares the performance of these algorithms across three binary classification problems taken from the USPS data set (Schölkopf & Smola, 2002): 2 vs 5 (1657 training points and 464 test points), 3 vs 8 (1376 training points and 345 test points), and 5 vs 9 (1194 training points and 366 test points). Each plot depicts the objective progression of the projected Euclidean functional gradient descent algorithm in green and the objective progression of functional gradient descent through the RKHS in red. We measure progress in terms of the computation time required to achieve a particular objective value. For each trial, we standardized the features, used radial basis function kernels with standard deviation 15, and chose values $\gamma = 5$ and $\lambda = .001$. We implemented both gradient descent variants using a line search to find the optimal step size.

The projected Euclidean functional gradient descent algorithm drastically outperforms functional gradient descent through the RKHS on these problems in spite of the added computational costs of the initial inversion of the kernel matrix and the matrix-vector multiplication at each iteration. Classification accuracy on the hold out sets for these problems was .985, .991, and .992, respectively, for Euclidean functional gradient descent. In most cases, the accuracy of functional gradient descent through the RKHS was the same by the time the optimization converged, although, in the case of the 2 vs 5 problem, it was slightly lower, achieving an accuracy of only .981.

Theorem 4.1.1 demonstrates that these algorithms are equivalent to preconditioned parametric gradient descent with particular choices of preconditioners.

Theorem 4.1.1: *Algorithm 7 is equivalent to preconditioned parametric gradient descent with preconditioner $\mathbf{K}$, and Algorithm 8 is equivalent to preconditioned parametric gradient descent with preconditioner $\mathbf{K}(\mathbf{K} + \gamma\mathbf{I})$.*

Proof. We first note that the parameters $\mathbf{b}$ of the kernel functional gradient are related to the parametric gradient $\mathbf{g}$ by $\mathbf{g} = \mathbf{K}\mathbf{b}$ (see (Ratliff & Bagnell, 2007) for an explanation of this property). Therefore, updating the parameters α using $\mathbf{b}$, as is done in the kernel functional gradient descent algorithm, is equivalent to preconditioning by $\mathbf{K}$ (in which case each parametric gradient is transformed by $\mathbf{K}^{-1}$). Algorithm 8 additionally transforms the gradient by $(\mathbf{K} + \gamma\mathbf{I})^{-1}$ at each iteration implying that that algorithm utilizes a preconditioner of the form $\mathbf{K}(\mathbf{K} + \gamma\mathbf{I})$. $\square$

Note that the Hessian of KLR at $f = 0$ is $\frac{1}{4}\mathbf{K}(\mathbf{K} + \tilde{\gamma}\mathbf{I})$ with $\tilde{\gamma} = 4\lambda$. The first step of the projected Euclidean functional gradient descent algorithm is, therefore, approximately a Newton step. This observation explains the steep downward drop of the algorithm during the initial iterations relative to functional gradient descent through the RKHS which we see in Figure 4.2.

4.2 Generalizing exponentiated gradient descent to function spaces

In some cases, it is useful to operate solely within a space of positive hypothesis functions. In this section, we generalize the finite-dimensional exponentiated gradient descent algorithm (Kivinen & Warmuth, 1997) to function spaces using the projected Euclidean functional gradient tools we developed above. We prove that the resulting update in function space has positive inner product with the negative functional gradient. Section 4.2 derives this algorithms, which we call *exponentiated* functional gradient descent, by analogy to the parametric variant, and Chapter 6 applies it in the context of the MMP framework we develop in this thesis.

4.2.1 Exponentiated functional gradient descent

We can characterize the traditional Euclidean gradient descent update rule as a minimization problem. At our current hypothesis w_t , we create a linear approximation to the function (in the case of convex functions, this approximation is also a lower bound), and we minimize the

approximation while regularizing back toward w_t . Mathematically, we write this as

$$w_{t+1} = \arg \min_{w \in \mathcal{W}} f(w_t) + g_t^T(w - w_t) + \frac{\lambda_t}{2} \|w - w_t\|^2, \quad (4.17)$$

where $g_t = \nabla f(w_t)$ is the gradient (or subgradient) at w_t . Analytically solving for the minimizer by setting the gradient of the expression to zero derives the Euclidean gradient descent rule: $w_{t+1} = w_t - \alpha_t g_t$ with step size $\alpha_t = 1/\lambda_t$. Thus, the gradient descent rule naturally encourages solutions that have a small norm in the sense of $\|\cdot\|_2$ (see (Zinkevich, 2003)). A similar procedure derives the update rule for exponentiated gradient descent as well. Replacing the Euclidean regularization term in Equation 4.17 with an unnormalized KL-divergence regularization of the form $\text{uKL}(w, w_t) = \sum_j w^j \log \frac{w^j}{w_t^j} - \sum_j w^j + \sum_j w_t^j$ and analytically solving for the minimizer results in the exponentiated gradient update rule (Kivinen & Warmuth, 1997):

$$w_{t+1} = w_t e^{-\alpha_t g_t} = e^{-\sum_{\tau=0}^t \alpha_\tau g_\tau} \quad (4.18)$$

$$= e^{u_{t+1}} \quad (4.19)$$

where $\{\alpha_t\}_{t=1}^\infty$ is a sequence of step sizes (sometimes called learning rates). For simplicity, we assume in what follows that w_0 is the vector of all ones, i.e. $w_0 = e^z$ where z is the zero vector. This assumption is not necessary, but it lightens the notation and clarifies the argument.

In the final expression of Equation 4.18, we denote $u_t = -\sum_{\tau} \alpha_\tau g_\tau$ in order to point out the relationship to the Euclidean gradient descent algorithm. The quantity u_t is the hypothesis that would result from an objective function whose gradients were g_t at each x_t . The only difference between the gradient descent algorithm and the exponentiated gradient descent algorithm is that in the gradient descent algorithm we simply evaluate the objective and its gradient using u_t , while in the exponentiated gradient algorithm, since the objective is a function of vectors from only the positive orthant, we first exponentiate this vector $w_t = e^{u_t}$ before evaluating the objective and its gradient. Essentially, the exponentiated gradient update rule is equivalent to the gradient descent update rule, except we exponentiate the result before using it.

In addition to the immediate benefit of only positive solutions, the key benefit enjoyed by

the exponentiated gradient algorithm is a robustness to large numbers of potentially irrelevant features. In particular, powerful results (Kivinen & Warmuth, 1997; Cesa-Bianchi & Lugosi, 2006) demonstrate that the exponentiated gradient algorithm is closely related to the growing body of work in the signal processing community on sparsity and $\|\cdot\|_1$ regularized regression. (Tropp, 2004; Donoho & Elad, 2003) Exponentiated gradient achieves this by rapidly increasing the weight on a few important predictors while quickly decreasing the weights on a bulk of irrelevant features. The unnormalized KL prior from which it is derived encourages solutions with a few large values and a larger number of smaller values. In the functional setting, the weights are analogous to the hypothesized function evaluated at particular locations in feature space. We believe this form of regularization— or prior, taking a Bayesian view— is natural for many planning problems in robotics where there is a very large dynamic range in the kind of costs to be expected.

This work generalizes the connection between these two optimization algorithms to the functional setting. A formal derivation of the algorithm is quite technical, but the resulting algorithm is straightforward and analogous to the finite-dimensional setting. We derive this algorithm informally in what follows.

In projected Euclidean functional gradient descent, we evaluate the functional gradient of the objective function, projected it onto a predefined set of hypothesis functions (search directions in function space), and take a step in the negative of that direction. In the exponentiated functional gradient descent algorithm we will essentially perform the same update (take a step in the negative direction of the projected functional gradient), but before evaluating the objective function or the functional gradient we will exponentiate the current hypothesis to ensure that it resides in a space of positive hypotheses.

Explicitly, the exponentiated functional gradient descent algorithm dictates the procedure presented in Algorithm 9.

4.2.2 Theoretical results

We show here that when the functional gradient can be represented explicitly, the exponentiated functional gradient descent algorithm produces a modification to the hypothesis that has a positive

Algorithm 9 Exponentiated functional gradient descent (intuition)

```

1: procedure EXPFUNGRAD( objective functional  $F[\cdot]$ , initial function  $h_0$ , step size sequence
    $\{\gamma_t\}_{t=1}^T$  )
2:   for  $t = 1, \dots, T$  do
3:     Evaluate the functional gradient  $g_t$  of the objective functional at the current hypothesis
        $f_t = e^{-\sum_{\tau=0}^{t-1} \gamma_\tau h_\tau}$ .
4:     Project the functional gradient onto the space of candidate search directions using least-
       squares to find  $h_t = h^* \in \mathcal{H}$ .
5:     Take a step in the negative of that direction forming  $f_{t+1} = f_t e^{-\gamma_t h_t} = e^{-\sum_{\tau=0}^t \gamma_\tau h_\tau}$ .
6:   end for
7: end procedure

```

inner product with negative functional gradient. Therefore, on each iteration, there always exists a finite step length interval for which the algorithm necessarily decreases the desired objective while preserving the natural sparsity and dynamic range of exponentiated function values.

Let $g_t(x)$ be the functional gradient of a functional at hypothesis $f_t(x)$. Under the exponentiated functional gradient algorithm, $f_t(x) = e^{h_t(x)}$ for some log-hypothesis $h_t(x)$. Thus, we need only consider positive hypotheses: $f(x) > 0$ for all x . Our update is of the form $f_{t+1}(x) = f_t(x)e^{-\lambda_t g(x)}$, where $\lambda_t > 0$ is a positive step size. Therefore, we can write our update offset vector as

$$\begin{aligned}
v_t(x) &= f_{t+1}(x) - f_t(x) = f(x)e^{-\lambda g(x)} - f(x) \\
&= f_t(x) \left(e^{-\lambda g_t(x)} - 1 \right).
\end{aligned}$$

We suppress the dependence on t in what follows for convenience.

Theorem 4.2.1: *The update direction $v(x)$ has positive inner product with the negative gradient. Specifically,*

$$- \int_{\mathcal{X}} g(x)v(x)dx = \int_{\mathcal{X}} g(x)f(x) \left(1 - e^{-\lambda g(x)} \right) dx > 0.$$

Proof. We first note that $\phi(u) = \frac{1}{u} (e^u - 1)$ is continuous and everywhere positive since u and $e^u - 1$

always have the same sign.⁵ We can rewrite our expression as

$$\begin{aligned}\int_{\mathcal{X}} g(x)f(x) \left(1 - e^{-\lambda g(x)}\right) dx &= \int_{\mathcal{X}} g(x)f(x) \left(\lambda g(x)\phi(-\lambda g(x))\right) dx \\ &= \lambda \int_{\mathcal{X}} g(x)^2 f(x)\phi(-\lambda g(x))dx.\end{aligned}$$

The integrand is everywhere nonnegative, and when our functional gradient is not the zero function, there exist measurable regions over which the integrand is strictly positive. Therefore, $-\int_{\mathcal{X}} g(x)v(x)dx > 0$. $\square$

4.3 Functional bundle methods

The gradient descent and functional gradient descent algorithms we have discussed so far in this chapter and in the previous chapter show strong performance across a number of convex machine learning formulations. They are particularly alluring for structured prediction problems due to their low memory requirements (Ratliff, Bagnell, & Zinkevich, 2007b), and recent theoretical work has shown that they converge fast across a wide range of problems in terms of both optimization and generalization (Ratliff, Bagnell, & Zinkevich, 2007b; Shalev-Shwartz, Singer, & Srebro, 2007; Shalev-Shwartz & Srebro, 2008). Additionally, functional gradient descent algorithms have seen success in a number of real-world problems (Ratliff *et al.*, 2006; Mason *et al.*, 1999).

Unfortunately, these functional gradient boosting algorithms are often inefficient in terms of their representation: the algorithm adds a new nonlinear base learner to its hypothesis at each iteration, regardless of whether that new base learner already correlates strongly with previous learners. Recent work in bundle methods for machine learning (Smola, Vishwanathan, & Le., 2008) has shown bundle optimization to be very efficient in terms of their representation, particularly for SVM learning problems (Joachims, 2006). In this section, we expand on the idea of representational efficiency by generalizing bundle methods to function spaces using the projected Euclidean functional gradient techniques derived in this chapter.

⁵Technically $\phi(u)$ has a singularity at $u = 0$. However, since the left and right limits at that point both equal 1, without loss of generality we can define $\phi(0) = 1$ to attain a continuous function.

4.3.1 L_2 functional regularization

Before we derive the functional bundle method, we introduce a notion of L_2 -regularization for the objective functional. This regularization term essentially constrains the size of the function values at each of the data points. We show that one straightforward way we can optimize the resulting objective is to simply apply the projected Euclidean functional gradient descent algorithm we derived above. However, by introducing this regularization term, we are additionally able to derive a functional bundle method that is can utilize generic Quadratic Program (QP) solvers in the inner loop. QP technology is very fast for reasonably small problems; the QPs that arise under the functional bundle method operate in the dual space and are of dimension T , where T is the number of iterations. These properties mean that we both have a clear cut termination criterion that we can measure in terms of the primal-dual gap, and we can leverage fast commercial QP-solvers in the inner loop for strong bundle optimization and rapid convergence.

Consider the following regression setting. We are provided a data set $\{(x_i, y_i)\}_{i=1}^N$ with $x_i \in \mathcal{X}$ and $y_i \in \mathbb{R}$. We derive the functional bundle method here assuming this straightforward regression setting specifically in terms of the squared-error loss, although generalizations to other convex loss functions are straightforward.

We add an L_2 -functional regularizer of the form $\sum_{i=1}^N f(x_i)^2$ with regularization constant $\lambda > 0$ to our loss term. Thus, we arrive at the following objective functional:

$$\mathcal{F}[f] = \frac{1}{2} \sum_{i=1}^N (y_i - f(x_i))^2 + \frac{\lambda}{2} \sum_{i=1}^N f(x_i)^2. \quad (4.20)$$

Intuitively, this regularization term penalizes the function at the data points.

Optimizing this objective using the functional gradient descent techniques discussed above is straightforward; we can easily derive the functional gradient of this objective as

$$\nabla_f \mathcal{F}[f] = - \sum_{i=1}^N \left[(y_i - f(x_i)) - \lambda f(x_i) \right] \delta_{x_i}. \quad (4.21)$$

The addition of the regularization term dictates that we decrease the i^{th} example's functional

gradient label proportionally to the size of the function at x_i . This modification ensures that the function values never grow unbounded, independent of the choice of loss function. We compare the functional bundle method to this projected Euclidean functional gradient descent formulation below in Section 4.3.4.

4.3.2 The functional bundle

At every iteration of the functional bundle method, we evaluate a new functional gradient $g_t = \sum_{i=1}^N \alpha_t^i \delta_{x_i}$ of the risk function $R[f] = \sum_{i=1}^N (y_i - f(x_i))^2$ at the current hypothesis f_t . The linear hyperplane in function space $\mathcal{F}[f_t] + \langle g_t, f - f_t \rangle = \mathcal{F}[f_t] + \sum_{i=1}^N \alpha_t^i (f(x_i) - f_t(x_i))$ lower bounds the functional $\mathcal{F}[f]$ everywhere and equals it at f_t . This property, analogous to the finite-dimensional subgradient case, allows us to write a lower bound to the objective as

$$\mathcal{B}_T[f] = \max_{t=1:T} \left\{ r[f_t] + \sum_{i=1}^N \alpha_t^i (f(x_i) - f_t(x_i)) \right\} + \frac{\lambda}{2} \sum_{i=1}^N f(x_i)^2, \quad (4.22)$$

where we denote $t = 1, \dots, T$ as $t = 1 : T$ for convenience. We call this convex piecewise quadratic approximation the L_2 functional bundle.

4.3.3 Optimizing the functional bundle

The functional bundle shown in Equation 4.22 depends on the hypothesis function f only through its evaluation at data points. We can gain insight into the form of the final functional bundle method by writing out an N -dimensional bundle optimization in terms of only the function values at the data points, as though we had complete freedom to arbitrarily choose those values. This optimization will specify precisely what we would like the function values to be at the data points; although we will not directly use this optimization problem during training, analyzing the form of the solution will provide insight into deriving a parameterization for the functional bundle optimization.

The functional bundle optimization problem is

$$\underset{f \in \mathcal{H}}{\text{minimize}} \quad \max_t \left\{ r[f_t] + \sum_{i=1}^N \alpha_t^i (f(x_i) - f_t(x_i)) \right\} + \frac{\lambda}{2} \sum_{i=1}^N f(x_i)^2. \quad (4.23)$$

If we define a vector $\mathbf{f}$ and a vector $\mathbf{a}_t$ as the vectors of function values and functional gradient coefficients (over all data points) as

$$\mathbf{f} = \begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_N) \end{pmatrix} \quad \text{and} \quad \mathbf{a}_t = \begin{pmatrix} \alpha_t^1 \\ \alpha_t^2 \\ \vdots \\ \alpha_t^N \end{pmatrix}, \quad (4.24)$$

then we can rewrite the optimization problem in Equation 4.23 as

$$\underset{\mathbf{f} \in \mathbb{R}^N}{\text{minimize}} \quad \max_t \left\{ c_t + \mathbf{a}_t^T (\mathbf{f} - \mathbf{f}_t) \right\} + \frac{\lambda}{2} \mathbf{f}^T \mathbf{f}. \quad (4.25)$$

If we were to solve this problem in the dual space, then from the T -dimensional dual solution β^* we can retrieve a primal solution through the dual connection

$$\mathbf{f}^* = -\frac{1}{\lambda} \sum_{t=1}^T \beta_t^* \mathbf{a}_t. \quad (4.26)$$

In other words, the optimal set of function values at the data points is simply a linear combination of the functional gradient coefficients. Since we have already trained a function approximator for each functional gradient data set $\{(x_i, \alpha_t^i)\}_{i=1}^N$, this observation suggested that in optimizing the functional bundle, we should parameterize the hypothesis function as a linear combination of the function approximators that have already been trained. (Alternatively, we could train a new function approximator based on the data set $\{x_i, \sum_{t=1}^T \beta_t^* \alpha_t^i\}$ to replace all other function approximators. This procedure will constrain the size of the hypothesis representation, although the computational and representational tradeoffs between these two techniques have not been explored in detail.)

We, therefore, parameterize the functional bundle optimization given in Equation 4.22 in terms of a linear combination of trained function approximators $f(x) = \sum_{t=1}^T \gamma_t \tilde{g}_t(x)$, where we denote the trained function approximators as $\tilde{g}_t(x)$ with $\tilde{g}_t(x_i) \approx \alpha_t^i$. Writing out the dual of this finite-

Table 4.1: Functional bundle prediction accuracies

	3 vs 8	4 vs 5	2 vs 5	3 vs 7	4 vs 8	5 vs 9	0 vs 1	1 vs 7	2 vs 7	3 vs 6
bundle	0.971	0.990	0.983	0.986	0.991	0.987	1.000	0.992	0.981	0.996
ss 0.005	0.971	0.990	0.981	0.982	0.989	0.986	1.000	0.990	0.979	0.996
ss 0.011	0.969	0.991	0.983	0.984	0.990	0.986	0.999	0.990	0.979	0.995
ss 0.022	0.970	0.989	0.979	0.983	0.991	0.986	0.999	0.989	0.977	0.995
ss 0.047	0.963	0.987	0.975	0.979	0.989	0.982	0.994	0.988	0.974	0.993
ss 0.100	0.959	0.988	0.973	0.981	0.987	0.979	0.995	0.991	0.976	0.990

dimensional problem in terms of dual variables β gives

$$\underset{\beta \in \mathbb{R}^T}{\text{maximize}} \quad -\frac{1}{2\lambda} \beta^T \mathbf{C} \beta + \beta^T \mathbf{b} \quad (4.27)$$

$$s.t. \quad \beta \geq 0 \quad \text{and} \quad \|\beta\|_1 = 1 \quad (4.28)$$

with $\mathbf{C} = \mathbf{A}^T (\mathbf{G} \mathbf{G}^T)^{-1} \mathbf{A}$ where

$$\mathbf{G} = \begin{pmatrix} \tilde{g}_1(x_1) & \tilde{g}_1(x_2) & \cdots & \tilde{g}_1(x_N) \\ \tilde{g}_2(x_1) & \tilde{g}_2(x_2) & \cdots & \tilde{g}_2(x_N) \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{g}_T(x_1) & \tilde{g}_T(x_2) & \cdots & \tilde{g}_T(x_N) \end{pmatrix}, \quad (4.29)$$

and $\mathbf{A}$ is a matrix formed by the column vectors $\mathbf{a}_t$. Again invoking the primal dual connection, we see that our final best-fit hypothesis is then $f^*(x) = \sum_{t=1}^T \gamma_t \tilde{g}_t(x)$, where the coefficients are given by $\gamma_t^* = \mathbf{C}^{-1} \mathbf{A} \beta$.

4.3.4 Experimental results

We implemented the L_2 functional bundle method on a collection of binary classification problems using the MNIST data set (LeCun *et al.*, 1998),⁶ and compared its performance to the projected Euclidean functional gradient descent approach outlined in Section 4.3.1. This implementation used the squared-error loss function presented in Section 4.3.1 to implement a form of regularized least-squares classification (Rifkin & Poggio, 2003).

⁶The data set may be obtained at <http://yann.lecun.com/exdb/mnist/>

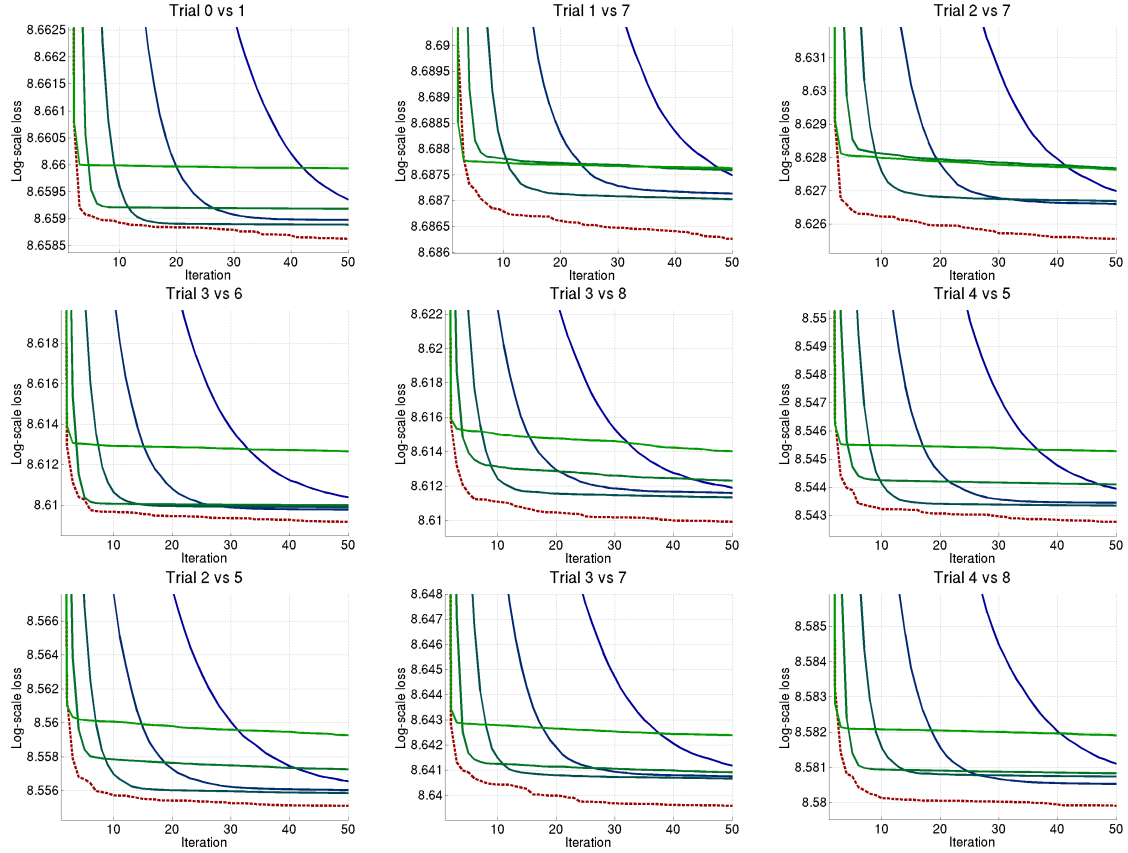


Figure 4.3: These plots compare the optimization performance of the functional bundle method (red dotted) to the performance of the functional gradient descent method (blue/green shades solid) on the same problem. The text in Section 4.3.4 provides a detailed explanation of these plots.

Figure 4.3 plots (in log-scale) the objective progressions across a collection of these results. The functional bundle method progression is shown in red (dotted), and a series of projected Euclidean functional gradient descent progressions using constant step sizes ranging from .005 to .1 on a log-scale are shown in solid blue and green shades (gradating from pure blue at .005 to pure green at .1). In this case, faster optimization implies a smaller representation. Table 4.1 lists the classification accuracies on hold out data of the final classifiers for each binary prediction problem. This table demonstrates that improved optimization implies improved performance at test time for these problems. In all cases, we used very simple neural network function approximators consisting of a single hidden node trained for 25 iterations. For all binary classification problems, the training and test sets contained approximately 12,000 and 1,550 examples, respectively.

Chapter 5

Maximum Margin Planning

This chapter introduces the maximum margin planning (MMP) framework for solving imitation learning via inverse optimal control. This framework reduces IOC to a contemporary form of machine learning known as maximum margin structured classification (Taskar, Guestrin, & Koller, 2003; Taskar, Lacoste-Julien, & Jordan, 2006), and accordingly forms the first well-defined general solution to the inverse optimal control problem outlined in Chapter 1. This chapter defines the linear theory of MMP originally introduced in (Ratliff, Bagnell, & Zinkevich, 2006).

Our linear theory of MMP defines a strictly convex objective function to govern learning allowing us to leverage the subgradient optimization tools developed in Chapter 3. The convergence, online regret, and batch generalization results presented in that chapter, therefore, carry over to linear MMP setting. We discuss these theoretical results in association with the encompassing class of maximum margin structured classification problems in Chapter 7.

We begin by reviewing some notation in Section 5.1 before presenting our core result, the reduction of inverse optimal control to maximum margin structured classification, in Section 5.2. Section 5.3 presents a simple implementation of MMP using the subgradient method reviewed in Chapter 3. Finally, Section 5.6 provides some experimental validation of this framework on real-world overhead navigation problems.

Chapter 6 derives nonlinear implementations of the maximum margin planning framework using

the techniques outlined in Chapter 4.

5.1 Preliminaries

We model an environment as a Markov Decision Process (MDP). Throughout this document, we denote the set of states by $\mathcal{S}$, the set of possible actions by $\mathcal{A}$, and the combined set of state-action pairs by $\mathcal{M} = \mathcal{S} \times \mathcal{A}$. Each MDP has a transition function, denoted $\mathcal{T}_{sa}^{s'}$, which defines the probability of transitioning to state s' when taking action a from state s . The set of transition probabilities defines the dynamics of the MDP.

Following (Ratliff, Bagnell, & Zinkevich, 2006), we denote policies using the dual flow. Intuitively, a policy, when run either infinitely or to a predefined horizon, visits each state-action pair an expected number of times. We denote the vector of these state-action frequency counts by $\mu \in \mathbb{R}_+^{|\mathcal{S}||\mathcal{A}|}$. The elements of these vectors adhere to a particular set of flow constraints (see (Gordon, 1999) for details). The constraints solidify our intuition of flow by specifying that the expected flow into a given state equals the expected flow out of that state, except at the start state which acts as a source, and at the goal state (should one exist) which acts as a sink. This notation is simply a matter of convenience for describing the algorithm; there is a one-to-one correspondence between the set of stationary Markovian policies and the set of feasible flow vectors (Puterman, 1994). The constraints can, therefore, be satisfied simply by invoking a generic MDP solver (i.e. a planning algorithm). We denote the set of all feasible flow vectors for a given MDP as $\mathcal{G}$.

At a high level, we define the imitation learning problem as the task of training a system to generalize from demonstrated behavior. Each training example i consists of an MDP (states, actions, and dynamics) and an expert policy. Each state-action pair has an associated fully observed feature vector $f^{sa} \in \mathbb{R}^d$ that concisely describes distinguishing qualities of that pair. These feature vectors are collected in a feature matrix which we denote $F \in \mathbb{R}^{d \times |\mathcal{S}||\mathcal{A}|}$. We are also given the set of possible flow vectors $\mathcal{G}_i$ (i.e. the set of all policies), and denote the expert policy by $\mu_i \in \mathcal{G}_i$. Finally, each example has an associated loss function $\mathcal{L}_i(\mu)$ which quantifies how bad a given policy μ is with respect to the desired policy μ_i . We require that the loss function decompose over state-action pairs so that the loss of a policy can be defined as $\mathcal{L}_i(\mu) = l_i^T \mu$, where $l_i \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$. As in Section

2.2, we call this vector the *loss field* and refer to each element l_i^{sa} of the loss field as a loss element. Each loss element intuitively describes the loss that a policy accrues by traversing that particular state-action pair. Using this notation, we can write the data set as $\mathcal{D} = \{(\mathcal{M}_i, F_i, \mathcal{G}_i, \mu_i, l_i)\}_{i=1}^N$.

The feature vectors in this problem definition transfer information from one MDP to another. By defining a policy implicitly in terms of the optimal controller for a cost function over a set of features, the policy can be applied to generalize to new MDPs outside the training set.

Of particular interest to us is the special case for which the dynamics of the MDP are deterministic and the problem is goal oriented. In this case, we can reduce the set of flow vectors $\mathcal{G}_i$ to only those that denote deterministic acyclic paths from the start state to the goal state. Each $\mu \in \mathcal{G}_i$ then becomes simply an indicator vector denoting whether or not the policy traverses a particular state-action pair along its path to the goal. Many combinatorial planning algorithms, such as A*, return only policies from this reduced set.

We assume that we have access to planner. Given a cost vector $c \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$ that assigns a cost c^{sa} to each state action pair, the MDP solver returns an optimal policy. Formally, the planner solves the following inference problem:

$$\mu^* = \arg \min_{\mu \in \mathcal{G}} c^T \mu, \quad (5.1)$$

where $c^T \mu$ is the cumulative cost of policy $\mu \in \mathcal{G}$.

It is sometime useful to overload the notation $\mathcal{M}_i$ to denote the set of all state-action pairs in the i^{th} MDP. Additionally, we often denote the (s, a) th element of a vector $v \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{A}|}$ (defined over all state action pairs) as v^{sa} .¹

¹When the path is deterministic, each element of μ is either 1 if the path traverses that particular state-action pair, or 0 if it does not. The path cost expression in this case reduces to simply the sum of the state-action costs over only the state-action pairs found in the path. This observation often eases implementation.

5.2 Reducing imitation learning to maximum margin structured classification

We discussed informally in Section 2.2 that the goal of imitation learning can be formalized as the problem of learning a cost function for which the example policy has lower expected cost than each alternative policy by a margin that scales with the loss of that policy. Intuitively, if a particular policy is very similar to the example policy as quantified by the loss function (i.e. the policy has low loss), then the margin for that policy is very small and the algorithm requires that the cost of the example policy be only slightly smaller. On the other hand, if the policy is very different from the example policy (i.e. the policy has high loss), then the margin for that policy will be large and the algorithm will want the cost of that policy to greatly exceed that of the example policy. Essentially, the margin adapts to each policy based on how bad that policy is relative to the example policy.

We can formalize this intuition using a form of machine learning known as maximum margin structured classification (MMSC) (Taskar, Lacoste-Julien, & Jordan, 2006; Ratliff, Bagnell, & Zinkevich, 2007a). Taking the cost function to be linear in the features, we can write the cost of a policy $\mu \in \mathcal{G}_i$ under MDP i as $c(\mu) = w^T F_i \mu$ for any real valued weight vector $w \in \mathbb{R}^d$. Using this notation, MMSC formalizes the above intuition through a set of constraints enforcing that for each MDP $\mathcal{M}_i$ and for each policy $\mu \in \mathcal{G}_i$,

$$w^T F_i \mu_i \leq w^T F_i \mu - l_i^T \mu. \quad (5.2)$$

These constraints, known in the structured prediction (MMSC) literature as structured margin constraints, explicitly state that the cost of the example policy $w^T F_i \mu_i$ should be lower than the cost of the alternative policy $w^T F_i \mu$ by an amount (i.e. a margin) that scales with the loss $l_i^T \mu$. If the loss term $l_i^T \mu$ is small, then we require the example policy μ_i to have cost only slightly less than μ . Alternatively, if the loss $l_i^T \mu$ is large, then the constraints require that the example policy's cost should be much smaller than that of μ .

At face value, Equation 5.2 specifies an exponential number of constraints since the number of

policies $|\mathcal{G}|$ is exponential in the number of state-action pairs $|\mathcal{S}||\mathcal{A}|$. However, following the logic originally introduced in (Taskar, Guestrin, & Koller, 2003; Taskar *et al.*, 2005) we note that, for a given example i , the left-hand-side of Equation 5.2 is constant across all policies $\mu \in \mathcal{G}_i$. Therefore, if the constraint holds for the single policy that minimizes the right-hand-side expression then it holds for all policies. In other words, we need only worry about the constraint corresponding to the particular minimizing policy

$$\mu_i^* = \arg \min_{\mu \in \mathcal{G}_i} \{w^T F_i \mu - l_i^T \mu\} = \arg \min_{\mu \in \mathcal{G}_i} \{(w^T F_i - l_i^T) \mu\}. \quad (5.3)$$

If we were to remove the loss function term in Equation 5.3, then the resulting expression would represent the traditional planning problem (see Equation 5.1). With the presence of the additional loss term, it becomes what we call a *loss-augmented* planning problem. As in Section 2.2, we refer to the vector $w^T F_i - l_i^T$ as the loss-augmented cost map. From this expression we can see that the loss-augmented planning problem can be solved simply by sending the loss-augmented cost map to the planner as described in 2.2.

This manipulation allows us to rewrite the constraints in Equation 5.2 in a more compact form:

$$\forall i, \quad w^T F_i \mu_i \leq \min_{\mu \in \mathcal{G}_i} \{w^T F_i \mu - l_i^T \mu\}. \quad (5.4)$$

While these new constraints are no longer linear, they remain convex². Importantly, this transformation will allow us to derive a convex objective function for the imitation learning problem that can be efficiently optimized using the subgradient method.

These constraints, themselves, are not sufficient to characterize the desired solution. If the example policy's cost is even only a small $\epsilon > 0$ less than the cost of another policy μ , then a simple scaling of the vector w (i.e. a scaling of the cost function) can make the cost gap between the two policies arbitrarily large. With no additional constraints on the size of the weight vector w , this observation trivializes the structured margin criterion. Consequently, in order to make the margin term meaningful, we want to find the smallest weight vector w for which the constraints in

²The term on the right-hand-side of the inequality is a min over affine functions and is, therefore, concave.

Equation 5.4 are satisfied. Moreover, since there may not be a weight vector that uniformly satisfies all of the constraints, much less a small one that exactly satisfies the constraint, we introduce a set of slack variables, one for each example $\{\zeta_i\}_{i=1}^N$, that allow constraint violations for a penalty.

These additional criteria suggest the following constrained convex optimization problem:³

$$\begin{aligned} \min_{w \in \mathcal{W}, \zeta_i \in \mathbb{R}_+} \quad & \frac{1}{N} \sum_{i=1}^N \zeta_i + \frac{\lambda}{2} \|w\|^2 \\ \forall i, \quad & w^T F_i \mu_i \leq \min_{\mu \in \mathcal{G}_i} \{w^T F_i \mu - l_i^T \mu\} + \zeta_i \end{aligned} \tag{5.5}$$

where $\lambda \geq 0$ is a constant that trades off the penalty on constraint violations with the desire for small weight vectors. This optimization problem tries to find a simple (small) hypothesis w for which there are few constraint violations.

Technically, convex programming problems of this sort require nonnegativity constraints on the slack variables. However, in our case, the example policy is an element of the collection of all policies $\mu_i \in \mathcal{G}_i$, so the difference between the left and right sides of the cost constraints can never be less than zero ($w^T F_i \mu_i - \min_{\mu \in \mathcal{G}_i} \{w^T F_i \mu - l_i^T \mu\} \geq 0$). The slack variables will, therefore, always be nonnegative independent of explicit nonnegativity constraints.

Since the slack variables are in the objective function, the minimization drives the slack variables to be as small as possible. In particular, at the minimizer the slack variables will always exactly equal the constraint violation. The following equality condition, therefore, holds at the minimizer

$$\xi_i = w^T F_i \mu_i - \min_{\mu \in \mathcal{G}_i} \{w^T F_i \mu - l_i^T \mu\}. \tag{5.6}$$

This observation allows us to move the constraints directly into the objective function by replacing the slack variables with the expression given in Equation 5.6. Doing so leads us to the following

³This optimization problem is a generalization of the traditional support vector machine (SVM). If we restrict $\mathcal{G}_i$ (which is formally interpreted as the (exponentially large) set of classes in a structured prediction problem (Taskar, Lacoste-Julien, & Jordan, 2006)), to contain only two elements and choose the loss function to be the hamming loss, then the convex program reduces to the one typically seen in the SVM literature.

objective function which we term the Maximum Margin Planning (MMP) objective:

$$R(w) = \frac{1}{N} \sum_{i=1}^N \left(w^T F_i \mu_i - \min_{\mu \in \mathcal{G}_i} \{w^T F_i \mu - l_i^T \mu\} \right) + \frac{\lambda}{2} \|w\|^2. \quad (5.7)$$

This convex⁴ objective function takes the form of a regularized risk function (Rifkin & Poggio, 2003); its two terms trade off data fit with hypothesis complexity.

We emphasize that this objective function forms an upper bound on our structured loss function $\mathcal{L}(\mu_i, \mu) = l_i^T \mu$ in a way that generalizes the upper bound formed on the zero-one loss by the support vector machine's binary hinge-loss. This structured loss function measures the difference in behavior between the learner and demonstrating expert. Optimizing Equation 5.7, therefore, minimizes an upper bound on the the desired non-convex loss.

5.3 Optimizing the maximum margin planning objective

The maximum margin planning objective function given in Equation 5.7 can be optimized in a number of ways. In this section, we discuss the application of the subgradient method (see Chapter 3) to this problem. In the context of maximum margin planning, this algorithm manifests as a simple and intuitive iterative procedure that trains a planner through repeated execution.

5.3.1 Computing the subgradient for linear MMP

The terms $w^T F_i \mu_i$ and $\frac{\lambda}{2} \|w\|^2$ are differentiable and, therefore, their unique subgradients are the gradients $F_i \mu$ and w , respectively. The term $-\min_{\mu \in \mathcal{G}_i} \{w^T F_i \mu - l_i^T \mu\}$ is only slightly more complicated. The surface of this convex function is formed as a max over a set of affine functions.⁵ The subgradient at w is, therefore, the gradient of the affine function forming the surface at that point. We can find this surface affine function simply by solving the loss-augmented inference problem given in Equation 5.3. Using that notation, the affine function forming the surface at w

⁴Again, as was the case in the constraints given in Equation 5.4, the term $\min_{\mu \in \mathcal{G}_i} \{w^T F_i \mu - l_i^T \mu\}$ is a min over affine functions which is known to be concave; its negative is therefore convex.

⁵For clarity, we use the transformation $-\min_i h_i(w) = \max_i \{-h_i(w)\}$ to simplify the argument.

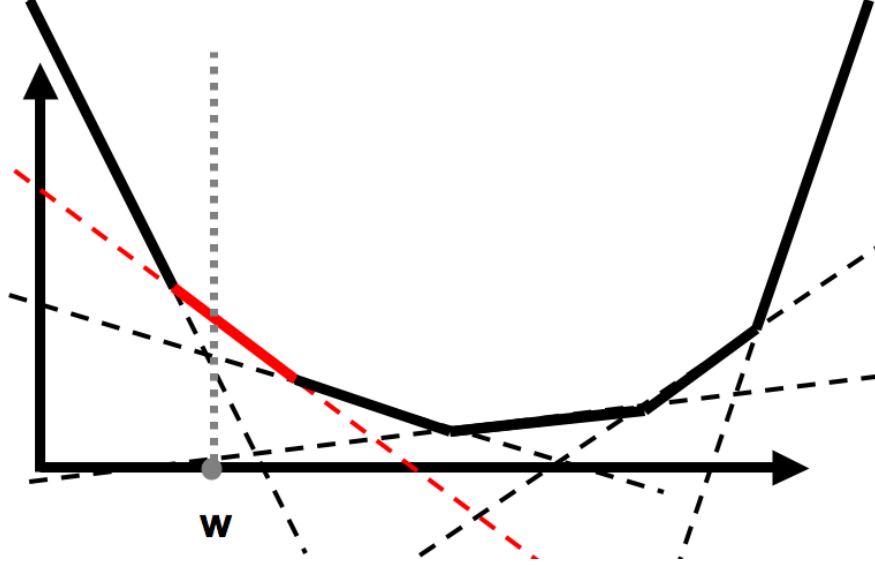


Figure 5.1: This figure visualizes the subgradient of a max over affine functions. Each affine function is denoted as a dashed line, and the surface of the resulting convex function is delineated in bold. The subgradient at a point $w \in \mathcal{W}$ is simply the subgradient of the affine function that forms the surface at that point, shown here in red.

is $(w^T F_i - l_i^t) \mu_i^*$, and we know the subgradient of that function to be $F_i \mu_i^*$.⁶ Figure 5.1 visualizes the process of evaluating the subgradient of a max over affine functions.

Using the above results, we can write the subgradient of the maximum margin planning objective given in Equation 5.7 as

$$\begin{aligned} \nabla R(w) &= \frac{1}{N} \sum_{i=1}^N F_i (\mu_i - \mu_i^*) + \lambda w \\ &= \frac{1}{N} \sum_{i=1}^N F_i \Delta \mu_i^* + \lambda w, \end{aligned} \tag{5.8}$$

where μ_i^* is the solution to the loss-augmented inference problem from Equation 5.3 for example i . We denote $\Delta \mu_i^* = \mu_i - \mu_i^*$ to emphasize that this component of the gradient is constructed by transforming the difference in frequency counts between the example policy μ_i and the loss-augmented policy μ_i^* into the space of features using the matrix F_i . This term singles out the

⁶There is a slight subtlety at points of nondifferentiability. At such points, two or more affine functions intersect and the loss-augmented inference problem has more than one solution (i.e. there are multiple optimal policies through the loss-augmented cost map). Since the subgradient algorithm requires only that one of the possible subgradients be followed at each time step, at these points we can choose any optimizer to construct the subgradient.

feature vectors at states for which the frequency counts differ substantially. If the example policy visits a particular state more frequently than the loss-augmented policy, the subgradient update rule will suggest a modification that will decrease the cost of that state. On the other hand, if the example policy visits a state less frequently, the update will want to increase the cost of that state.

The maximum margin planning algorithm, therefore, iterates the following update rule until convergence

$$w_{t+1} = \mathcal{P}_{\mathcal{W}} [w_t - \alpha_t (F_i(\mu_i - \mu_i^*) + \lambda w)]. \quad (5.9)$$

When both the environment and the policies are deterministic, the vector matrix product $F_i\mu$ can be implemented efficiently using sparse multiplication. For instance, many of our experiments use the A* algorithm to find an optimal deterministic policy (i.e. a path) through the environment. We compute the product $F_i\mu$ simply by accumulating the feature vectors encountered while traversing the path represented by μ .

5.3.2 An approximate projection algorithm for cost positivity constraints

Define $f_j^{(i)}$ to be the feature (column) vector over the i th cell in the j th map⁷, and define the j th map's feature matrix as $F_j = (f_j^{(1)}, f_j^{(2)}, \dots, f_j^{(M_j)})$, where M_j is the number of cells in the j th map. Defined $F = (F_1, F_2, \dots, F_N)$ to be the matrix formed by stringing all map's feature matrices together.

The math works as demonstrated in Figure 5.2. We have a constraint for each feature f_i vector in F :⁸

$$w^T f_i \geq c_i \quad (5.10)$$

for some $c_i \geq 0$. The first step is to find the most violated constraint, i.e. the i that maximizes $c_i - w^T f_i$. For this f_i , the constraint forms a surface $\mathcal{S}_{f_i} = \{w \in \mathcal{W} \mid w^T f_i = c_i\}$, onto which we want to project the vector w . Note that the surface is orthogonal to f_i , making the math relatively

⁷I make the perhaps overly conservative assumption that all features are non-negative to ensure convergence of the algorithm.

⁸I've re-indexed the column vectors of F for convenience. Column i of F is f_i .

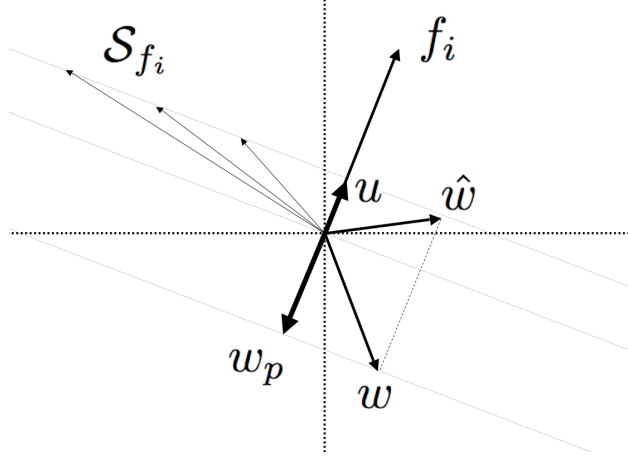


Figure 5.2: f_i is the feature vector in question and our (violating) weight vector is w . We want to project w onto the constraint surface $\mathcal{S}_{f_i}$ to get $\hat{w}$, which we can do by subtracting w_p and adding u .

easy. If we define $u \in \mathcal{S}_{f_i}$ to be the unique element of $\mathcal{S}_{f_i}$ aligned with f_i , and w_p to be the projection of w onto f_i , then our update should be $\hat{w} = w - w_p + u$ (see Figure 5.2). Now we just have to calculate these.

For u we set $u = \lambda f_i$ and solve for the λ that touches the surface: $u^T f_i = \lambda f_i^T f_i = c_i$. This gives $\lambda = \frac{c_i}{f_i^T f_i}$ and $u = c_i \frac{f_i}{\|f_i\|^2}$. For the projection, we get

$$w_p = w^T \left(\frac{f_i}{\|f_i\|} \right) \frac{f_i}{\|f_i\|} \quad (5.11)$$

$$= w^T f_i \frac{f_i}{\|f_i\|^2}. \quad (5.12)$$

Thus, our update becomes

$$\hat{w} = w - w_p + u \quad (5.13)$$

$$= w - w^T f_i \frac{f_i}{\|f_i\|^2} + c_i \frac{f_i}{\|f_i\|^2} \quad (5.14)$$

$$= w - (w^T f_i - c_i) \frac{f_i}{\|f_i\|^2}. \quad (5.15)$$

This procedure is presented in full in Algorithm 10. The following theorem proves that the algorithm implements an approximate projection operator as defined in Equation 3.10.

Algorithm 10 Approximate projection for cost positivity constraints

```
1: procedure APPROXPROJECT( feature matrix  $F$ , vector of minimum costs  $c$  )
2:   while not converged do
3:     Compute  $\tilde{c} = F^T w$  and  $V = c - \tilde{c}$ 
4:     If  $V$  is component-wise non-positive then exit
5:     Find  $i = \arg \max_i v_i$ , where  $v_i$  is the  $i$ th element of  $V$ 
6:     Project:  $w \leftarrow w - (\tilde{c}_i - c_i) \frac{f_i}{\|f_i\|^2}$ 
7:   end while
8: end procedure
```

Theorem 5.3.1: *Algorithm 10 implements an approximate projection operator.*

Proof (sketch). Projection onto a plane brings the projected point closer to every point on the opposite side of the plane that it originally was. Since the algorithm iteratively projects onto planes and since the feasible set is a union over the half spaces defined by these planes, every projection step brings the point closer to every feasible point. $\square$

5.4 Learning linear quadratic regulators

We note briefly here that the MMP framework does not require the MDP to be discrete. One common continuous state (discrete time) decision process used commonly in practice is the *linear quadratic regulator* (LQR) used in linear systems theory (Boyd *et al.*, 1994). In this setting, the reward function takes the form $r(x; Q) = x^T Q x$ where $x \in \mathbb{R}^n$ is the continuous state of the system and Q is a positive definite matrix parameterizing the rewards. This reward function is linear as a function of Q ; the gradient takes a simple form $\nabla_Q x^T Q x = x x^T$. Deriving the learning updates is, therefore, straightforward. After each update, we can easily project the resulting matrix onto the space of positive (nonnegative) definite matrices using the diagonalization $Q = U \Sigma U^T$. In this case, $\mathcal{P}_{\mathbb{R}_+^{n \times n}}[Q] = U \Sigma_+ U^T$, where Σ_+ denotes the diagonal matrix resulting from thresholding all negative Eigenvalues to zero. Alternatively, we can implement a matrix exponentiated gradient procedure using the techniques of (Tsuda, Rätsch, & Warmuth, 2005). Such an algorithm amounts to performing the updates in log-space to find a matrix $\tilde{Q}$ which we use in practice to evaluate rewards via matrix exponentiation $r(x; \tilde{Q}) = x^T \exp \left\{ \tilde{Q} \right\} x$. (Operationally, matrix exponentiation of symmetric matrices is implemented by exponentiating the Eigenvalues of the matrix.)

5.5 A compact quadratic programming formulation

Consider again the convex programming problem presented in Equation 5.5, which we restate here in terms of rewards (negative costs) for convenience:

$$\min_{w, \zeta_i} \frac{\lambda}{2} \|w\|^2 + \frac{1}{N} \sum_{i=1}^N \zeta_i \quad (5.16)$$

$$s.t. \quad \forall i \quad w^T F_i \mu_i + \zeta_i \geq \max_{\mu \in \mathcal{G}_i} w^T F_i \mu + l_i^T \mu \quad (5.17)$$

Each Bellman-flow vector $\mu \in \mathcal{G}_i$ satisfies a set of Bellman-flow constraints which specify that the flow into a state must equal the flow out of the state (modulo the start and goal states):

$$\sum_{x,a} \mu^{x,a} p_i(x'|x, a) + s_i^{x'} = \sum_a \mu^{x',a}$$

The nonlinear, convex constraints in Equation 5.17 can be transformed into a compact set of linear constraints (Taskar *et al.*, 2005; Taskar, Guestrin, & Koller, 2003) by computing the dual of the right hand side of each yielding:

$$\forall i \quad w^T F_i \mu_i + \zeta_i \geq \min_{v \in V_i} s_i^T v \quad (5.18)$$

where $v \in V_i$ are the value-functions that satisfy the Bellman primal constraints:

$$\forall x, a \quad v^x \geq (w^T F_i + l_i)^{x,a} + \sum_{x'} p_i(x'|x, a) v^{x'} \quad (5.19)$$

By combining the constraints together we can write one compact quadratic program:

$$\min_{w, \zeta_i, v_i} \frac{\lambda}{2} \|w\|^2 + \frac{1}{N} \sum_{i=1}^N \zeta_i \quad (5.20)$$

$$s.t. \quad \forall i \quad w^T F_i \mu_i + \zeta_i \geq s_i^T v_i \quad (5.21)$$

$$\forall i, x, a \quad v_i^x \geq (w^T F_i + l_i)^{x,a} + \sum_{x'} p_i(x'|x, a) v_i^{x'} \quad (5.22)$$

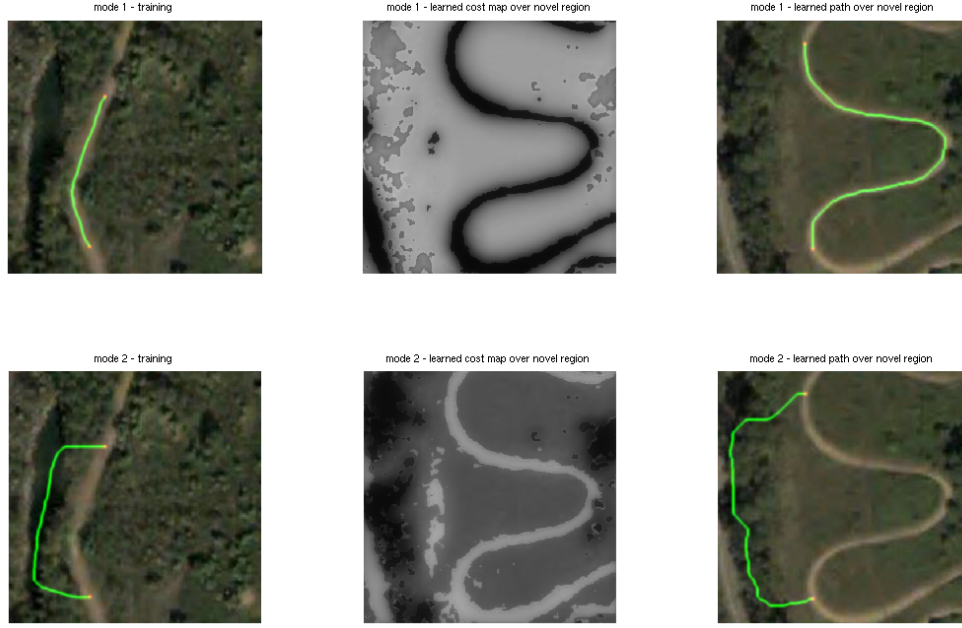


Figure 5.3: Demonstration of learning to plan based on satellite color imagery. For a particular training/holdout region pair, the top row of images depicts training the learner to follow the road while the bottom row depicts training the learner to “hide” in the trees. From left to right, the columns show the single training example presented to the learner, the learned cost map over the holdout region, and the corresponding behavior learned for that region. Cost values scale with intensity in these images.

This result demonstrates that we can represent MMP as a compact quadratic program (QP). This results provides a representation

For small problems, we can therefore exploit commercial off-the-shelf quadratic programming software for training. Unfortunately, in practice, since the number of constraints scales linearly with the number of state-action pairs, the QP is typically too large for this approach to be practical. The LEARCH algorithms discussed in this thesis between this chapter and in Chapter 6 are, therefore, crucial for the efficient implementation of MMP.

5.6 Experimental validation

To validating these concepts we focused on the practical problem of path planning using the subgradient-based learning algorithm discussed in Section 5.3. In this setting, the MDP can be viewed as a two-dimensional map discretized uniformly into an array of cells. Each cell represents

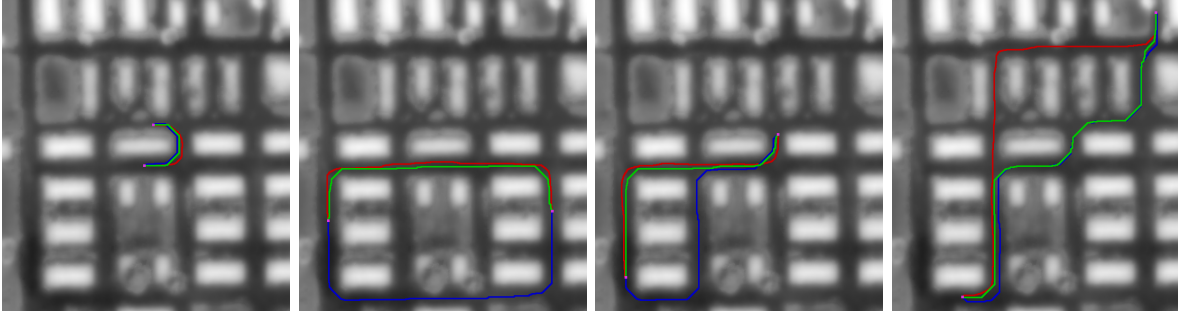


Figure 5.4: See Figure 5.5 for data-set. Data shown are MMP learned cost maps (dark low cost) with a teacher supplied path (red), loss-augmented path (blue), and final learned path (green). These are learned results on a holdout set.

a particular location in the world and typical actions include moving from a given cell to one of the eight neighboring cells. In all experiments, we used A^* as our specialized planning algorithm and chose reasonable regularization constants by hand.

We first exhibit the versatility of our algorithm in learning distinct concepts within a single domain. Differing example trajectories, demonstrated in one region of a map, lead to a significantly different behavior in a separate holdout region after learning. Figure 5.3 shows qualitatively the results of this experiment. The behavior presented in the top row represents a road following concept, while that portrayed in the bottom row embodies a “stealthy” behavior. By column, from left to right, the images depict the training example presented to the algorithm, the learned cost map on a holdout region after training, and the resulting behavior produced by A^* over this region.⁹

For our second experiment, the data derived entirely from laser range readings (ladar) over the region of interest collected during an overhead helicopter sweep.¹⁰ A visualization of the raw data is depicted in Figure 5.5. Figure 5.4 shows typical results from a holdout region. The learned behavior (green) often matches well the desired behavior (red). Even when the learner failed to match the desired trajectory exactly, the learned behavior adheres to the primary rules set forth

⁹The features used in this experiment were derived entirely from a single overhead satellite image. We discretized the image into five distinct color classes and added smoothed versions of the resulting features to propagate proximity information.

¹⁰Raw features were computed from mean and standard deviations from each of elevation, signal reflectance, hue, saturation, and local ladar shape information (Vandapel *et al.*, 2004). Again, we added smoothed versions of the raw features to utilize proximity information.

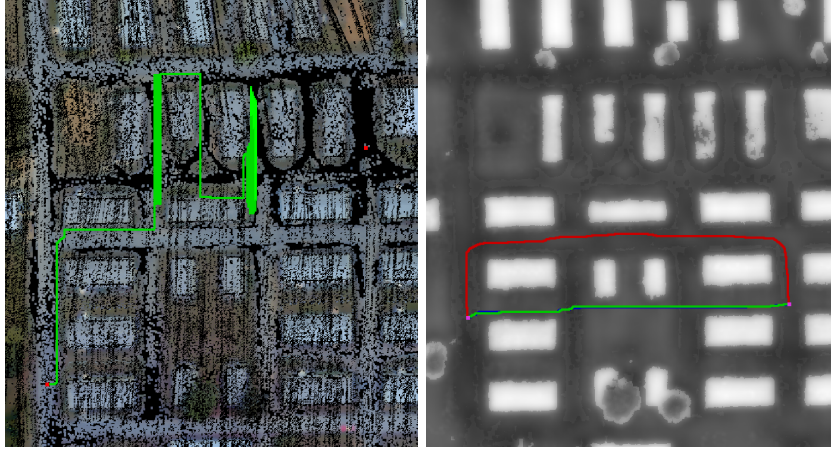


Figure 5.5: Left: the result of a next-action classifier applied super-imposed on a visualization of the second data-set. Right: a cost map learned by manual training of a regression. The learned paths (green) in both cases are poor approximations of the training examples (not shown on left, red on right).

implicitly by the examples. Namely, the learner finds an efficient path that avoids buildings (white) and grassy areas (gray) in lieu of roads.

Notice that the loss-augmented path (blue) in this figure performs generally worse than the final learned trajectory. This is because loss-augmentation makes areas of high loss more desirable than they would be in the final learned map. Intuitively, if the learner is able to perform well with respect to the loss-augmented cost map, then it should perform even better without the loss-augmentation; that is, the concept is learned with *margin*. For comparison, we attempted to learn similar behavior using two alternative approaches to MMP. First, we tried the reactive approach of directly learning from examples a mapping that takes state features to next actions as in (LeCun *et al.*, 2006).¹¹ Unfortunately, the resulting paths were rather poor matches to the training data. See Figure 5.5 for a typical example of a path learned by the classifier.

A somewhat more successful attempt was to try to learn costs directly by a hand labeling of regions. This provides dramatically more explicit information to the learner than MMP requires: a trainer provided examples regions of low, medium, and high costs, based upon (1) expert knowledge of the planner, (2) iterated training and observation, and (3) the trainer had prior knowledge of the

¹¹We used the same training data, training Regularized Least Squares classifiers (Rifkin & Poggio, 2003) to predict which nearby state to transition to. It proved difficult to engineer good features here; our best results come from using the same local state features as MMP augmented with distance and orientation to the goal. The learner typically achieved between 0.7-0.85 prediction accuracy.

cost maps found under MMP batch learning on this data set.¹² Although cost maps given this extra information looked qualitatively correct, Figure 5.5 demonstrates that the planning performance was significantly inferior.

¹²The low cost examples came from the example paths and the medium/high cost examples were supplied separately. Low cost and high cost examples were chosen as minimum and maximum values for A^* , respectively. Multiple medium cost levels were tried.

Chapter 6

LEARCH: Learning to Search

Chapter 5 introduced the MMP framework for solving inverse optimal control flavored imitation learning problems, and introduces a linear theory for implementing the framework. In this chapter, we extend this theory to nonlinear settings, making it easier to apply to a wide range of real-world applications. While the linear theory is more readily understood, feature extraction for these linear models can be difficult. The algorithms presented here implement MMP using the functional gradient techniques outlined in Chapter 4. This class of functional gradient algorithms is known collectively as LEArning to seaRCH (LEARCH).

Section 6.1 generalizes the linear MMP framework to the nonlinear setting by rewriting the MMP objective as a functional defined over a general function space. We then present the functional gradient of this functional in Section 6.2 and discuss some intuition behind the procedure defined by applying exponentiated functional gradient algorithm to this problem in Section 6.3. Section 6.4 then derives a novel log-linear variant of LEARCH that outperforms the original linear MMP algorithm while retaining its representational efficiency. We additionally discuss issues pertaining to representational efficiency in Section 6.6, where we present a stagewise variant of LEARCH known as MMPBOOST .

6.1 The MMP functional

In the functional setting, the MMP objective takes on essentially the same form as Equation 5.7, but with each policy cost term $w^T F_i \mu$ replaced by the more general term $\sum_{(s,a) \in \mathcal{M}_i} c(f_i^{sa}) \mu^{sa}$:

$$R[c] = \frac{1}{N} \sum_{i=1}^N \left(\sum_{(s,a) \in \mathcal{M}_i} c(f_i^{sa}) \mu_i^{sa} - \min_{\mu \in \mathcal{G}_i} \left\{ \sum_{(s,a) \in \mathcal{M}_i} (c(f_i^{sa}) - l_i^{sa}) \mu^{sa} \right\} \right). \quad (6.1)$$

As before, this functional sums over all examples the difference between the cumulative cost of the i^{th} example policy $\sum_{(s,a) \in \mathcal{M}_i} c(f_i^{sa}) \mu_i^{sa}$ and the cumulative cost of the (loss-augmented) minimum cost policy $\min_{\mu \in \mathcal{G}_i} \left\{ \sum_{(s,a) \in \mathcal{M}_i} (c(f_i^{sa}) - l_i^{sa}) \mu^{sa} \right\}$. Since the example policy is a valid policy, the minimum cost policy will always be smaller. Each example's objective term (the i^{th} term) is, therefore, always nonnegative. It represents the degree to which the example policy is suboptimal under the hypothesized cost function.

While the linear setting typically includes an explicit L_2 regularization term, we remove the regularization term in this expression to simplify the functional gradient computation. Boosting-type functional gradient descent procedures often admit regularization path arguments of the type discussed in (Rosset, Zhu, & Hastie, 2004). These arguments state that the number of boosting steps executed quantitatively determines the effective size or complexity of the model class being consideration. Early stopping, therefore, plays a similar role to regularization.¹ Alternatively, an L_2 functional regularization term, such as the one discussed in Section 4.3.1, can be added to Equation 6.1 to implement explicit regularization.

This section discusses an algorithm derived as an application of exponentiated functional gradient descent to optimizing this functional. This algorithm is a more formal version of the algorithm discussed at an intuitive implementational level in Chapter 2.

The contributions of exponentiated functional gradient descent to the hypothesis space are two fold. First, the use of functional gradients admits nonlinear hypotheses which one may interpret as a form of nonlinear feature selection. Second, the exponentiation of cost space gives the learned cost

¹Additionally, boosting relies implicitly on the class of weak learning algorithms to induce generalization and limit hypothesis complexity. Strongly regularized weak learners induce slower complexity growth.

Algorithm 11 Exponentiated functional gradient descent for maximum margin planning

```

1: procedure LEARCH( training data  $\{(\mathcal{M}_i, \xi_i)\}_{i=1}^N$ , loss function  $l_i$ , feature function  $f_i$  )
2:   Initialize log-costmap to zero:  $s_0 : \mathbb{R}^d \rightarrow \mathbb{R}, s_0 = 0$ 
3:   for  $t = 0, \dots, T - 1$  do
4:     Initialize the data set to empty:  $\mathcal{D} = \emptyset$ 
5:     for  $i = 1, \dots, N$  do
6:       Compute the loss-augmented costmap  $c_i^l = e^{s_t(F_i)} - l_i^T$  and find the minimum cost
         loss-augmented path through it  $\mu_i^* = \arg \min_{\mu \in \mathcal{G}_i} c_i^l \mu$ 
7:       Generate positive and negative examples:  $\forall (s, a) \in \mathcal{M}_i, \mathcal{D} =$ 
          $\{\mathcal{D}, (f_i^{sa}, 1, \mu_i^{*sa}), (f_i^{sa}, -1, \mu_i^{sa})\}$ 
8:     end for
9:     Train a regressor or classifier on the collected data set  $\mathcal{D}$  to get  $h_t$ 
10:    Update the log-hypothesis  $s_{t+1} = s_t + \alpha_t h_t$ 
11:  end for
12:  return Final costmap  $e^{s_T}$ 
13: end procedure

```

map a stronger dynamic range giving the algorithm access to a larger class of policies. Section 6.4 derives a log-linear variant of LEARCH which demonstrates that this latter point alone substantially improves performance, even each functional gradient is approximated using a linear function.

6.2 General setting

Using the tools described in Chapter 4, we can derive the L_2 functional gradient of the maximum margin planning objective functional (see Equation 6.1) as

$$\nabla_f R[c] = \frac{1}{N} \sum_{i=1}^N \left(\sum_{(s,a) \in \mathcal{M}_i} \mu_i^{sa} \delta_{f_i^{sa}} - \sum_{(s,a) \in \mathcal{M}_i} \mu_i^{*sa} \delta_{f_i^{sa}} \right). \quad (6.2)$$

In this expression, we denote $\mu_i^* = \arg \min_{\mu \in \mathcal{G}_i} \{\sum_{(s,a) \in \mathcal{M}_i} (c(f_i^{sa}) - l_i^{sa}) \mu^{sa}\}$; we call this quantity the optimal loss-augmented policy.

The functional gradient has the same form as that considered in Section 4.1.1: it is a weighted sum of delta (impulse) functions $\sum_j \gamma_j \delta_{x_j}$. In this case, magnitude of a given weight is determined by the frequency count at that state-action pair, and the sign of the weight is determined by whether it comes from the loss-augmented policy or the example policy.

6.3 Intuition

Algorithm 11 details the LEARCH algorithm. This listing demonstrates explicitly how to implement the operation of finding a direction function that correlates well with the functional gradient. Intuitively, the functional gradient can be viewed as a weighted classification or regression data set, where weights come from the magnitude of the delta function coefficients in the gradient term, and the label comes from the sign of these coefficients.

At each iteration, the exponentiated functional gradient algorithm starts by finding a direction function, defined over the feature space, that correlates well with the negative functional gradient. Intuitively, this means that the function is positive in regions of the feature space where there are many positive delta functions (in the negative gradient) and negative in regions where there are many negative delta functions. It then adds this direction function to the log of the previously hypothesized cost function with a small scalar step size α_t . (The step size may decrease toward zero over time as discussed in Section 5.3.) Adding the direction function to the cost function effectively increases and decrease the hypothesis as dictated by the impulse signals found in the negative functional gradient. Finally, the algorithm exponentiates the modified log-hypothesis to arrive at a valid positive cost function.

Intuitively, the negative functional gradient places negative impulses at feature vectors found along state-action pairs seen while executing the example policy so that the cost function is decreased in those regions. Conversely, it places positive impulses at feature vectors found along state-action pairs encountered while executing the loss-augmented policy so that the cost function is increased in those regions. In both cases, the magnitude of each impulse is proportional to the frequency with which the relevant policy traverses the state-action pair. If the distribution of feature vectors seen by both the example policy and the loss-augmented policy coincide, then the positive and negative impulses cancel resulting in no net suggested update. However, if the distributions diverge, then the algorithm will decrease the cost function in regions of the feature space where the example policy dominates and increase the cost function in regions where the loss-augmented policy (erroneously) dominates.

We have already seen this algorithm in Section 2.2, where we motivated it from a practical

standpoint for the specific case of deterministic planning. In some problems, we do not require the cost function to be positive everywhere. For those cases, we may simply apply the more traditional non-exponentiated variant (i.e. gradient boosting (Mason *et al.*, 1999)). Section 6.5 describes experiments using both exponentiated and non-exponentiated variants of the algorithm on two imitation learning problems from the field of robotics.

6.4 A log-linear variant

The mathematical form of the cost function learned under the LEARCH framework is dictated by the choice of the regressor or classifier used to implement the functional gradient projection step. In this section, we look at the simplest case of applying linear regression to approximate the functional gradient—this often represents a simple, efficient, and effective starting point even when additional non-linear functional gradient approximations are to be applied.

Since a linear combination of linear functions is also a linear function, the final cost function has an efficient log-linear representation

$$f_k(x) = e^{\sum_{t=1}^k \alpha_t h_t(x)} = e^{\sum_{t=1}^k \alpha_t u_t^T x} = e^{w_k^T x},$$

where $w_k = \sum_t \alpha_t u_t$. Moreover, exponentiating the linear function creates a hypothesis space of cost functions with substantially higher dynamic ranges for a given set of features than our original linear alternative which we presented in Section 5.3. We find that this log-linear variant demonstrates empirically superior performance.

6.4.1 Deriving the log-linear variant

We derive this variant of LEARCH simply by choosing a set of linear functions $h(x) = w^T x$ as the direction set. The following theorem presents the resulting update rule.

Theorem 6.4.1: *Let w_t be the hypothesized weight vector at the t^{th} time step. Then the update rule with step size η_t under least-squares projection (see Section 4.1.1 of Chapter 4) takes the form*

$w_{t+1} = w_t - \eta_t C_t^{-1} g_t$ where g_t is the parametric Euclidean gradient given in Equation 5.8 and

$$C_t = \sum_{i=1}^N F_i \text{diag}(\mu_i + \mu_i^*) F_i^T.$$

Specifically, our hypothesis at time T takes the form $c_{\mathcal{M}_i}^T(\mu) = e^{-(\sum_{t=1}^T \eta_t C_t^{-1} g_t)^T F_i \mu}$.

Proof. We prove the theorem for the general objective discussed in Section 4.1.1. Applying this result to Equation 6.1 completes the proof. Given a linear hypothesis space, the least-squares functional gradient projection operator induces the following quadratic objective function:

$$\begin{aligned} \left\langle h_w, \sum_{j=1}^k \alpha_j x_j \right\rangle - \frac{1}{2} \sum_{j=1}^k |\alpha_j| h_w(x_j)^2 &= \sum_{j=1}^k \alpha_j w^T x_j - \frac{1}{2} \sum_{j=1}^k |\alpha_j| (w^T x_j)^2 \\ &= w^T \sum_{j=1}^k \alpha_j x_j - \frac{1}{2} \sum_{j=1}^k |\alpha_j| w^T (x_j x_j^T) w = w^T \sum_{j=1}^k \alpha_j x_j - \frac{1}{2} w^T \left(\sum_{j=1}^k |\alpha_j| x_j x_j^T \right) w. \end{aligned}$$

Since this expression is quadratic, we can solve for the optimal update direction by setting its gradient to zero:

$$\begin{aligned} \nabla \left(w^T \sum_{j=1}^k \alpha_j x_j - \frac{1}{2} w^T \left(\sum_{j=1}^k |\alpha_j| x_j x_j^T \right) w \right) &= \sum_{j=1}^k \alpha_j x_j - \left(\sum_{j=1}^k |\alpha_j| x_j x_j^T \right) w = 0 \\ \Rightarrow w &= C^{-1} \sum_{j=1}^k \alpha_j x_j. \end{aligned}$$

where $C = \sum_{j=1}^k |\alpha_j| x_j x_j^T$. Since each α_j is implicitly a function of w , C is also a function of w , and we can therefore view C as an adaptive whitening matrix. $\square$

One may view this modified search direction as the parametric gradient taken under the Riemannian metric C_t . Under the MMP functional, this Riemannian metric adapts to the current combined distribution of feature vectors included by the example and loss-augmented policies.

This algorithm addresses the feature scaling issues discussed in (Neu & Szepesvari, 2007). Specifically, linear MMP is sensitive to the relative scaling of its features, a problem common to margin-based learning formulations.² Using linear regression to implement the modified correla-

²Intuitively, since we typically require the weight vector to lie within a Euclidean ball, a feature whose range is 10 times larger than another will likely dominate the hypothesis since a tiny weight on that feature can produce the same degree of cost variation as a large weight on the other feature.

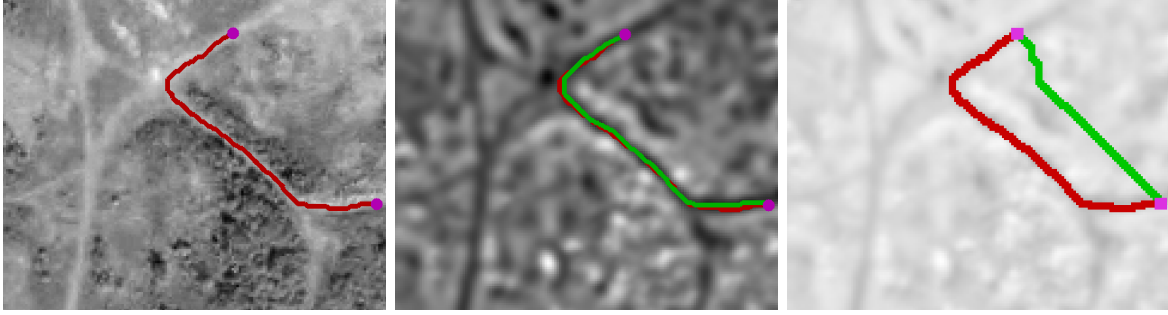


Figure 6.1: The LEARCH framework suggests a log-linear algorithm which can be used as an alternative to linear maximum margin planning (MMP). The cost functions in the log-linear variant’s hypothesis space generally achieve higher dynamic ranges for a given feature set and, therefore, tend to show empirically superior performance. This figure compares the two algorithms on a simple application using the holdout region shown in the leftmost panel. The rightmost panel shows the planning performance on the best linear combination of features achieved by linear MMP, and the center panel shows best exponentiated linear combination of features found by the log-linear LEARCH algorithm. The log-linear algorithm generalizes the expert’s behavior well and clearly outperforms linear MMP on this problem.

tion criterion within the log-linear LEARCH variant effectively removes this dependence on feature scaling.

6.4.2 Log-linear LEARCH vs linear MMP

Figure 6.1 depicts a straightforward example of where the log-linear LEARCH algorithm is able to substantially outperform linear MMP. The leftmost panel shows an overhead satellite image depicting a test region, held out from the training set, which we use to evaluate both algorithms. The feature set for this problem consisted solely of Gaussian smoothings of the original grayscale overhead satellite images. We purposefully chose these features to be simple to emphasize the performance differences. The linear MMP algorithm failed to generalize the expert’s behavior to the test region (rightmost panel). The best linear combination of features found by linear MMP defined a cost function with very small dynamic range and implemented a naïve minimum distance policy. However, when allowed to exponentiated the linear combination of features, within 6 iterations, the log-linear LEARCH algorithm (center panel) successfully converged to an expressive cost function that generalized the behavior well.

Additionally, Figure 6.2 depicts the difference in validation performance between log-linear LEARCH and linear MMP on a more realistic problem using a stronger feature set, including color

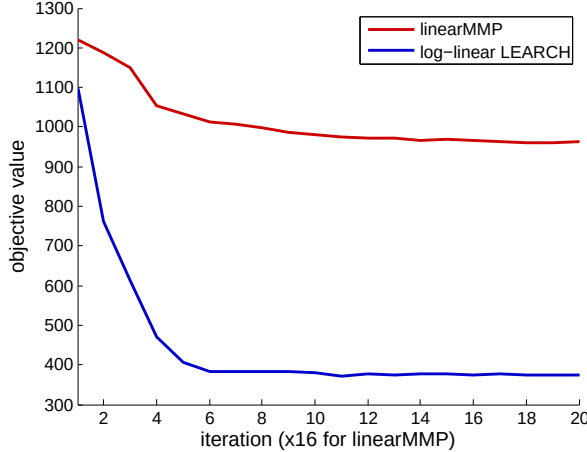


Figure 6.2: Objective values obtained for the problem depicted in Figure 6.1 under log-linear LEARCH (blue) and linear MMP (red), the latter optimized using the subgradient method with approximate projection (see Section 3.1). The linear MMP plot is scaled to fit on the graph, although it represents 300 iterations of the algorithm (16 iterations per point). The log-linear LEARCH algorithm converged to a substantially better objective value within 20 iterations.

class, multi-spectral, and texture features. For this experiment, we optimized the linear MMP objective using functional gradient boosting of linear hypotheses and satisfied cost-positivity constraints by truncating the costs to a minimum value. Log-linear LEARCH significantly outperforms linear MMP on this problem because of the increased dynamic range in its hypothesis space.

6.5 Case study: Multiclass classification

In this section, we demonstrate LEARCH on two multiclass classification problems: footstep prediction and grasp prediction. These experiments used single-hidden-layer neural networks to implement the functional gradient approximation step in line 9 of algorithm 11.³

6.5.1 Footstep prediction

Recent work has demonstrated that decomposing legged locomotion into separate footstep planners and execution controllers is an effective strategy for many problems (Chestnutt *et al.*, 2003, 2005). The footstep planner finds a sequence of feasible footstep across the terrain, and the execution

³In practice, we typically use ensembles of small neural networks to reduce variance. We trained the ensembles simply by training the network k times to get $\mathcal{S} = \{f_i(x)\}_{i=1}^k$ and averaging the results: $f(x) = \frac{1}{k} \sum_{i=1}^k f_i(x)$.

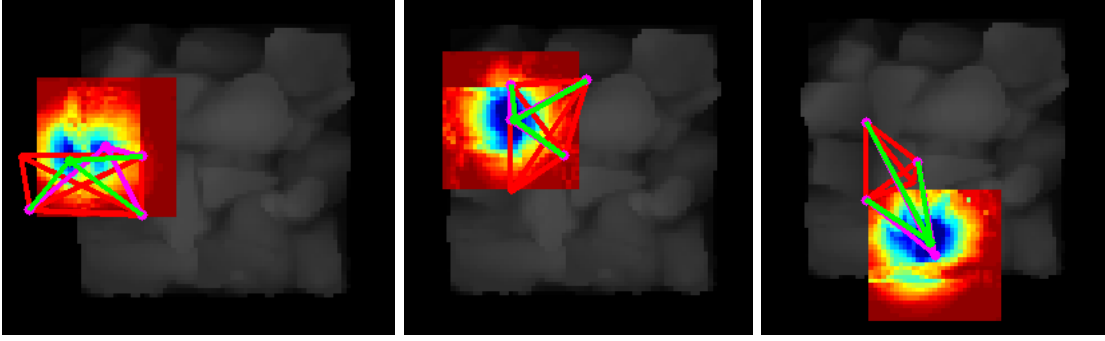


Figure 6.3: Validation results: Green indicates predicted next footstep, purple indicates desired next footstep, and red indicates current stance. The associated cost function is provided gradating from red at high cost regions to dark blue at low cost regions. The influence of the terrain features on the cost function is most apparent in the left-most prediction whose hypotheses straddle the border between the terrain and flat ground.

controller finds a trajectory through the full body configuration space of the robot that successfully places the feet at those locations. The feasibility of the suggested footstep locations is crucial to the overall success of system. In this experiment, we define and train a greedy footstep planning algorithm for a quadrupedal robot using the functional imitation learning techniques discussed in Section 6.2.

Our greedy footstep prediction algorithm chooses the minimum cost next footstep location, for a specific foot, given the current four-foot configuration of the robot and the patch of terrain residing directly below the hypothesized footstep location. The cost is defined to be a function of two types of features: action features and terrain features. A similar experimental setup is discussed in (Ratliff, Srinivasa, & Bagnell, 2007); we build on those results here by including stronger terrain features designed from nonparametric models of the terrain in these experiments.

Action features encode information about the kinematic demands of the action on the robot and the stability of the stance that results. These features include quantities describing how far the robot must stretch to make the footstep and the size of the support triangle that would result after taking the step (see Figure 6.5). Specifically, we compute: the distance and square distance from the hypothesized footstep location to each of the remaining three supporting feet and the original swing foot location, the exponentiated negative radius of the inscribed circle for the support triangle resulting from the footplacement, and an indicator of whether or not the foot is a front foot.

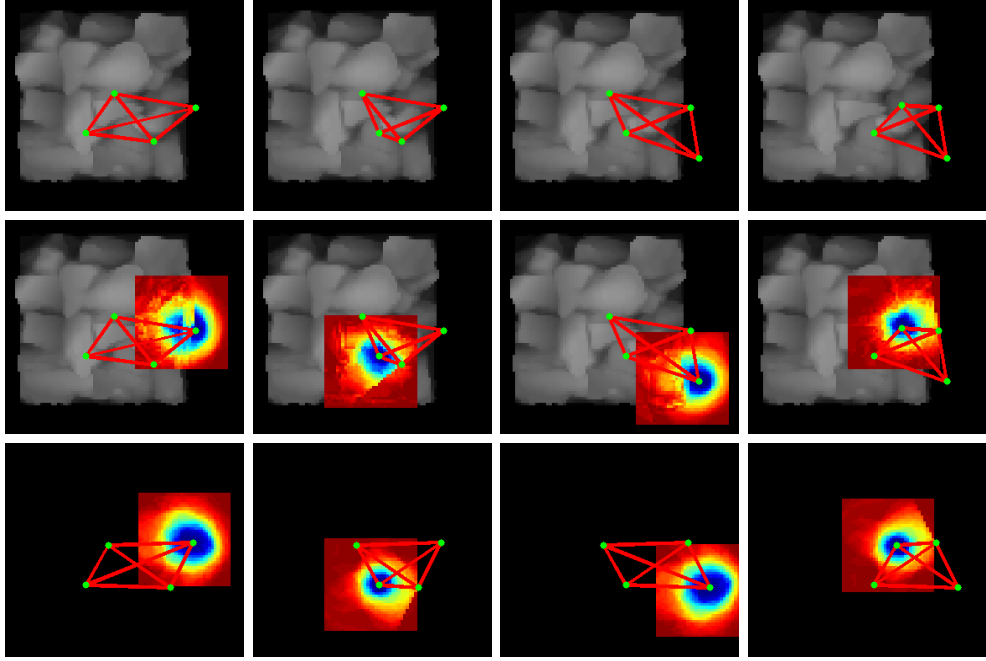


Figure 6.4: The first two rows show predicted footstep sequences across rough terrain both with and without the corresponding score function. The bottom row demonstrates a predicted sequence for walking across flat ground. Generalization of quadruped footstep placement. The four foot stance was initialized to a configuration off the left edge of the terrain facing from left to right. The images shown demonstrate a sequence of footsteps predicted by the learned greedy planner using a fixed foot ordering. Each prediction starts from result of the previous. The first row shows the footstep predictions alone; the second row overlays the corresponding cost region (the prediction is the minimizer of this cost region). The final row shows footstep predictions made over flat ground along with the corresponding cost region showing explicitly the kinematic feasibility costs that the robot has learned.

On the other hand, the terrain features encode the local shape of the terrain residing directly below the hypothesized next footstep location. In these experiments, we used two types of terrain features. The first set was the responses of a series of Gaussian convolutions to the height map. These features present averages of the terrain heights in a local region at varying resolutions. We derived the second set from the parameters of two locally quadratic regression approximations to the terrain built at two different resolutions. The latter set of features have proven useful on the current robotic platform.

We collected examples of good footstep behavior by teleoperating the quadruped robot shown in Figure 2.1 across the terrain shown in background of the overhead images in Figure 6.4. We trained our footstep predictor with these examples using LEARCH, shown in Algorithm 11, using a

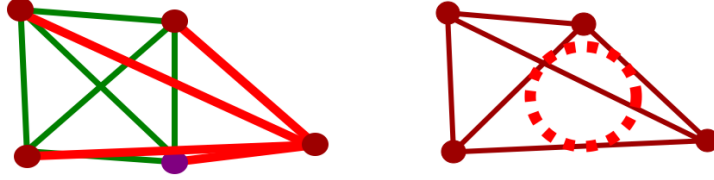


Figure 6.5: This figure shows some of the action features used for quadrupedal footstep prediction. In the left image, green lines delineate the initial four foot configuration; the purple dot signifies which foot is currently active. The bright red lines connecting each foot in the initial configuration to the hypothesized next foot location represent the “stretch” features. The rightmost figure shows the maximum radius inscribed circle of the support triangle that would result from taking the hypothesized step. We used this radius measure the stability of the support triangle.

direction set of small sigmoidal neural networks each with a single hidden layer of 15 nodes. For this experiment, we implemented optimal cost footstep prediction under the cost model described above using a brute force enumeration of a set of 961 feasible next footstep locations from a square region ahead of the foot in question.⁴

The loss function used for this problems was the squared Euclidean distance between the desired footstep location ν_i and the hypothesized footstep location ν : $\mathcal{L}(\nu, \nu_i) = \frac{1}{2}\|\nu - \nu_i\|^2/\sigma^2$. The increased dynamic range of the exponentiated variant of LEARCH allowed us to successfully utilize this relatively simple loss function as hypothesized in (Ratliff, Srinivasa, & Bagnell, 2007). The experiment depicted in that paper required that the loss function to range only between 0 and 1 in order to successfully generalize under the non-exponentiated variant.

Figure 6.3 depicts generalization results on a validation set. For each image, the current four-foot configuration is depicted in red, and we compare the desired footstep (green) to the predicted footstep (purple).

We additionally used our trained one-step-lookahead footstep predictor to predict a sequence of footsteps to traverse both rugged and flat terrain. These results are depicted in Figure 6.4. The top-most row shows four consecutive footsteps predicted across a rocky terrain, and the middle row renders the corresponding learned cost function. Our system successfully mimicked the expert’s preference for stable cracks in the terrain that were found to induce more robust footholds. The final

⁴We computed the offset defining what we mean by “ahead of” relative to the current four foot location so as to be rotationally invariant.

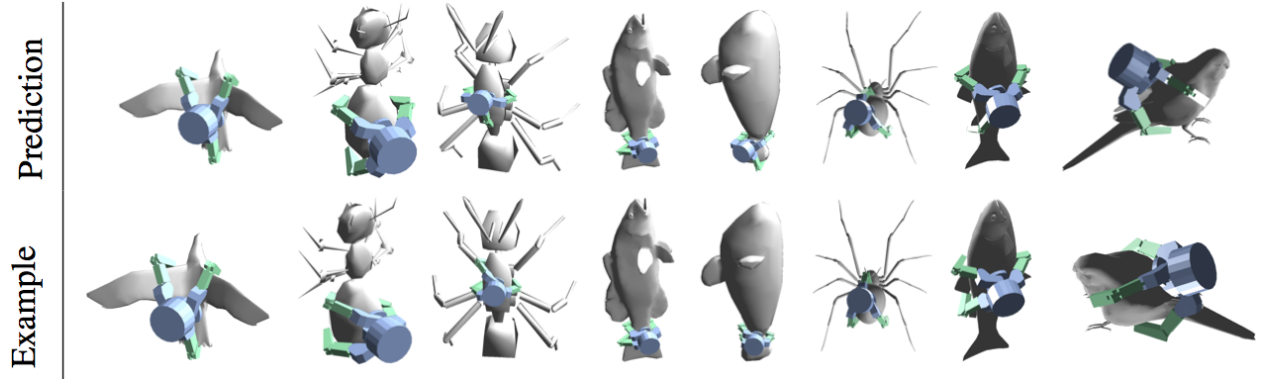


Figure 6.6: Grasp prediction results on ten holdout examples. The training set consisted of 23 examples in total; we generated each test result by holding out the example in question and training on the remaining 22.



Figure 6.7: The first three images from the left demonstrate grasp generalization from multiple approach direction on a single object. The final two images show from two perspectives a unique grasp that arises because of the simple feature set. See the text for details.

four images demonstrate the effect of action features alone on footstep prediction by running the predictor over flat ground. Our algorithm successfully learned the kinematic constraints represented in the data.

6.5.2 Grasp prediction

This section describes an application of LEARCH, shown in Algorithm 11, to grasp prediction. This implementation of LEARCH used a direction set consisting of neural networks with three sigmoidal hidden units and one linear output. The goal in this problem is to learn to predict grasp configuration for grasping objects with a Barrett hand from a given approach direction. The Barrett hand, shown in Figure 6.6, has ten degrees of freedom, six specifying the rotation and translation of the hand, and four specifying the configuration of the hand (all three fingers curl in independently, and two of the fingers can rotate around the palm in unison), although we restrict the translation and rotation of the hand to the provided approach direction. We do not attempt to learn the approach direction since it often depends on a number of manipulation criteria independent of the

grasping problem itself.

To produce grasp configurations, we use a control strategy similar to that used in the GraspIt! system (Miller *et al.*, 2003), although in this case, we constrain the wrist axis to align with a single approach direction (the palm always faces the object). A preshape for the hand is formed as a function of two parameters: the *roll* and the *finger spread*. The roll is the rotation angle of the hand around the axis of approach, and the finger spread of the angle between the hand’s two movable fingers. Given a hand preshape, our controller moves the hand forward it collides with the object. From there, it backs away a prespecified distance known as the *standoff* before closing its fingers around the object. In essence, we form a mapping between a three-dimensional space of parameters (the roll, finger spread, and standoff) to the space of grasp configurations. In practice, we discretize this space into a total of 2,496 cells.

In this experiment, we restrict our feature set to be simple quantities that describe only the local shape of the object immediately below the fingertip and the palm in order to demonstrate the generalization ability of our algorithms. These features summarize the set of point responses detected from rays shooting toward the object from the hand’s fingertips and palm. Specifically, we measure the exponentiated negative distance to collision for each ray. Since we compute many ray responses from each source point, the resulting feature vectors are very high-dimensional. We, therefore, use principal component analysis to project the vectors onto the fifteen orthogonal directions with highest variance computed across the training set.

We applied the exponentiated LEARCH algorithm to generalize the grasping behavior exemplified by a set of training examples demonstrated in simulation by a human expert. The second row of Figure 6.6 depicts select grasp demonstrations for a variety of objects taken from Princeton Shape Database⁵.

The loss function we used for this experiment measured the physical discrepancy between the final configurations produced by the simple controller. This loss is implemented as the minimum distance matching between points in the fingertips of the example configuration and corresponding points in the predicted configuration. Specifically, let p_1 , p_2 , and p_3 be points in the three fingertips

⁵<http://shape.cs.princeton.edu/benchmark/>

of the example configuration y and p'_1 , p'_2 , and p'_3 be corresponding points in the fingertips of the predicted configuration y' . Let Π be the set of all permutations of the set of indices $\mathcal{S} = \{1, \dots, 3\}$, and denote a particular permutation as a mapping $\pi : \mathcal{S} \rightarrow \mathcal{S}$. We define the loss function as

$$\mathcal{L}(y, y') = \min_{\pi \in \Pi} \sum_{i=1}^3 |p_i - p_{\pi(i)}|. \quad (6.3)$$

This gives low loss to configurations that are similar despite having vastly differing grasp parameters due to symmetries in the hand, while still giving high loss to configurations that are physically different. Importantly, since the cost function is defined as a function of local shape descriptive features, configurations with different grasp parameters but low loss under this loss function will tend to have similar features and therefore similar costs. This property allows functionally similar grasps to be assigned similar costs during learning without being artificially penalized for being different from the desired grasp parameters in terms of Euclidean distance through the parameter space.

Figure 6.6 shows our test predictions (from a holdout set) side-by-side with the grasp the human would have predicted for the problem. The algorithm typically generalizes well. It often learns concepts reminiscent of form closure for these unseen objects.

6.6 MmpBoost : A stage-wise variant

The nonlinear stepwise LEARCH algorithms discussed above are effective implementations of the MMP framework. However, The algorithm learns a new nonlinear function approximator at each iteration. Over time, these function approximators accumulate making each function evaluation increasingly costly. In many real-world applications, such as overhead navigation over long distances in which new terrain information is continually rolling in, this cost may be undesirable. We therefore seek an algorithm that may better utilize each learning stage.

We introduce here a stagewise variant of LEARCH called MMPBOOST that addresses this problem. Rather than taking a single step in the direction of the negative functional gradient, this variant uses each new function approximator as a new feature and learns the best weights for the

Algorithm 12 Stagewise functional gradient optimization for maximum margin planning

```

1: procedure MMPBOOST( training data  $\{(\mathcal{M}_i, \xi_i)\}_{i=1}^N$ , loss function  $l_i$ , base feature matrices
    $F_i^b$  )
2:   Initialize learned feature matrices  $F_i^l$  to empty
3:   for  $t = 1, \dots, T$  do
4:     Initialize the data set to empty:  $\mathcal{D} = \emptyset$ 
5:     Construct current feature matrices  $F_i$  by concatenating  $F_i^b$  and  $F_i^l$ 
6:     Run MMP to find the best linear model  $w_t$  for the current feature set
7:     for  $i = 1, \dots, N$  do
8:       Compute the loss-augmented costmap  $c_i^l = w_t^T F_i - l_i^T$  and find the minimum cost
         loss-augmented path through it  $\mu_i^* = \arg \min_{\mu \in \mathcal{G}_i} c_i^l \mu$ 
9:       Generate positive and negative examples under this map:  $\forall (s, a) \in \mathcal{M}_i, \mathcal{D} =$ 
          $\{\mathcal{D}, (f_i^{sa}, 1, \mu_i^{*sa}), (f_i^{sa}, -1, \mu_i^{sa})\}$ 
10:    end for
11:    Train a regressor or classifier on the collected data set  $\mathcal{D}$  to get  $h_t$ 
12:    Evaluate function approximator on base features to find new feature  $s_i = h_t(F_i^b)$ 
13:    Add  $s_i$  as a new row of  $F_i^l$ 
14:  end for
15:  return Final model  $(w_t, \{F_i\}_{i=1}^N, \{h_t\}_{t=1}^T)$ 
16: end procedure

```

new set of features at each stage using linear MMP. Alternatively, one may view this procedure as a functional direction set method: at each stage, the algorithm adds a new search direction to the direction set and then finds the optimal linear combination of these directions for the problem.

While this algorithm is resigned to work within a less dynamic (non-exponentiated) hypothesis space, the method has two primary advantages. First, as discussed above, it spends more time processing each feature set to reduce the number of duplicate search directions required during learning. This effort can result in somewhat more efficient hypothesis representation thereby reducing computational cost at test time. Second, by running linear MMP in the inner loop, MMPBOOST is able to satisfy a wider range of constraints than variants of LEARCH based on exponentiated functional gradient.

Algorithm 12 presents this algorithm in detail. Below, we describe two applications utilizing MMPBOOST .

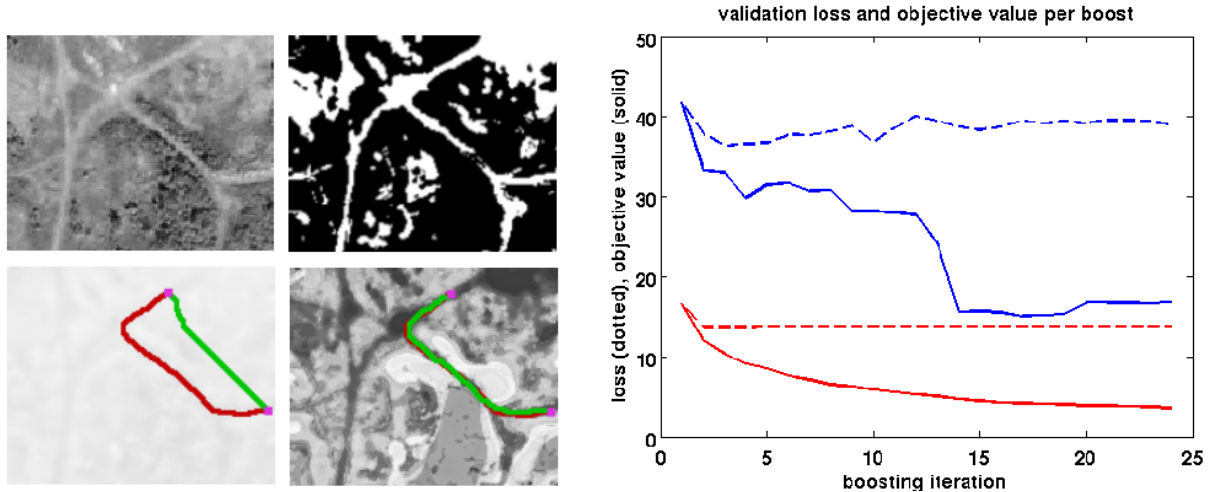


Figure 6.8: The four subimages to the left show (clockwise from upper left) a grayscale image used as base features for a hold out region, the first boosted feature learned by boosted MMP for this region, the results of boosted MMP on an example over this region (example red, learned path green), and the best linear fit of this limited feature set. The plot on the right compares boosting objective function value (red) and loss on a hold out set (blue) per boosting iteration between linear MMP (dashed) and boosted MMP (solid).

6.6.1 Overhead navigation

We first consider a problem of learning to imitate example paths drawn by humans on publicly available overhead imagery. In this experiment, a teacher demonstrates optimal paths between a set of start and goal points on the image, and we compare the performance of MMPBOOST to that of a linear MMP algorithm in learning to imitate the behavior. The base features for this experiment consisted of the raw grayscale image, 5 Gaussian convolutions of it with standard deviations 1, 3, 5, 7, and 9, and a constant feature. Cost maps were created as a linear combination of these features in the case of MMP, and as a nonlinear function of these features in the case of MMPBOOST. The planner being trained was an 8-connected implementation of A*.

The results of these experiments are shown in Figure 6.8. The upper right panel on the left side of that Figure shows the grayscale overhead image of the holdout region used for testing. The training region was similar in nature, but taken over a different location. The features are particularly difficult for MMP since the space of cost maps it considers for this problem consists of only linear combinations of the same image at different resolutions. e.g. imagine taking various blurred versions of an image and trying to combine them to make any reasonable cost map. The lower left panel on the left side of Figure 6.8 shows that the best cost map MMP was able to

find within this space was largely just a map with uniformly high cost everywhere. The learned cost map was largely uninformative causing the planner to choose the straight-line path between endpoints.

The lower right panel on the left side of Figure 6.8 shows the result of MMPBOOST on this problem on a holdout image of an area similar to that on which we trained. In this instance, we used regression trees with 10 terminal nodes as our dictionary $\mathcal{H}$, and trained them on the base features to match the functional gradient as described in Sections 4.1.1 and 6.2. Since MMPBOOST searches through a space of nonlinear cost functions, it is able to perform significantly better than the linear MMP. Interestingly, the first feature it learned to explain the supervised behavior was to a large extent a road detection classifier. The right panel of Figure 6.8 compares plots of the objective value (red) and the loss on the holdout set (blue) per iteration between the linear MMP (dashed) and MMPBOOST (solid).

The first feature shown in figure 6.8 is interesting in that it largely represents the result of a path detector. The boosting algorithm chooses positive examples along the example path, and negative examples along the loss-augmented path, which are largely disjoint from the example paths. Surprisingly, MMPBOOST also outperformed linear MMP applied to additional features that were hand-engineered for this imagery. In principle, given example plans, MMPBOOST can act as a sophisticated image processing technique to transform any overhead (e.g. satellite) image directly to a cost map with no human intervention and feature engineering.

6.6.2 Training a fast planner to mimic a slower one

Legged robots have unique capabilities not found in many mobile robots. In particular, they can step over or onto obstacles in their environment, allowing them to traverse complicated terrain. Algorithms have been developed which plan for foot placement in these environments, and have been successfully used on several biped robots (Chestnutt *et al.*, 2005). In these cases, the planner evaluates various steps the robot can execute, to find a sequence of steps that is safe and is within the robot’s capabilities. Another approach to legged robot navigation uses local techniques to reactively adjust foot placement while following a predefined path (Yagi & Lumelsky, 1999). This

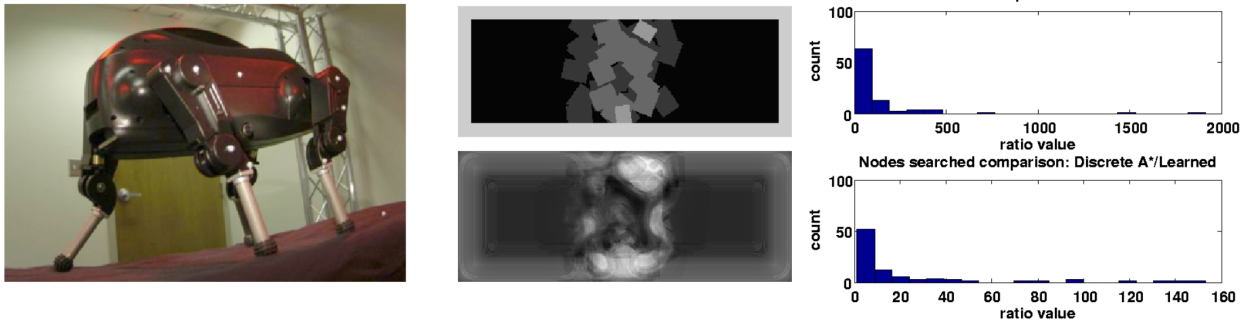


Figure 6.9: Left is an image of the robot used for the quadruped experiments. The center pair of images shows a typical height map (top), and the corresponding learned cost map (bottom) from a holdout set of the biped planning experiments. Notice how platform-like regions are given low costs toward the center but higher costs toward the edges, and the learned features interact to lower cost chutes to direct the planner through complicated regions. Right are two histograms showing the ratio distribution of the speed of both the admissible Euclidean (top) and the engineered heuristic (bottom) over an uninflated MMPBOOST heuristic on a holdout set of 90 examples from the biped experiment. In both cases, the MMPBOOST heuristic was uniformly better in terms of speed.

approach can fall into local minima or become stuck if the predefined path does not have valid footholds along its entire length.

Footstep planners have been shown to produce very good footstep sequences allowing legged robots to efficiently traverse a wide variety of terrain. This approach uses much of the robot’s unique abilities, but is more computationally expensive than traditional mobile robot planners. Footstep planning occurs in a high-dimensional state space and therefore is often too computationally burdensome to be used for real-time replanning, limiting its scope of application to largely static environments. For most applications, the footstep planner implicitly solves a low dimensional navigational problem simultaneously with the footstep placement problem. Using MMPBOOST , we use body trajectories produced by the footstep planner to learn the nuances of this navigational problem in the form of a 2.5-dimensional navigational planner that can reproduce these trajectories. We are training a simple, navigational planner to effectively reproduce the body trajectories that typically result from a sophisticated footstep planner. We could use the resulting navigation planner in combination with a reactive solution (as in (Yagi & Lumelsky, 1999)). Instead, we pursue a hybrid approach of using the resulting simple planner as a heuristic to guide the footstep planner.

Using a 2-dimensional robot planner as a heuristic has been shown previously (Chestnutt *et al.*, 2005) to dramatically improve planning performance, but the planner must be manually tuned to

	cost diff		speedup		cost diff		speedup	
	mean	std	mean	std	mean	std	mean	std
	biped admissible				biped inflated			
MMPBOOST vs Euclidean	0.91	10.08	123.39	270.97	9.82	11.78	10.55	17.51
MMPBOOST vs Engineered	-0.69	6.7	20.31	33.11	2.55	6.82	11.26	32.07
	biped best-first				quadruped inflated			
MMPBOOST vs Euclidean	-609.66	5315.03	272.99	1601.62	3.69	7.39	2.19	2.24
MMPBOOST vs Engineered	3.42	37.97	6.4	17.85	-4.34	8.93	3.51	4.11

Figure 6.10: Statistics comparing the MMPBOOST heuristic to both a Euclidean and discrete navigational heuristic. See the text for descriptions of the values.

provide costs that serve as reasonable approximations of the true cost. To combat these computational problems we focus on the heuristic, which largely defines the behavior of the A* planner. Poorly informed admissible heuristics can cause the planner to erroneously attempt numerous dead ends before happening upon the optimal solution. On the other hand, well informed inadmissible heuristics can pull the planner quickly toward a solution whose cost, though suboptimal, is very close to the minimum. This lower-dimensional planner is then used in the heuristic to efficiently and intelligently guide the footstep planner toward the goal, effectively displacing a large portion of the computational burden.

We demonstrate our results in both simulations and real-world experiments. Our procedure is to run a footstep planner over a series of randomly drawn two-dimensional terrain height maps that describe the world the robot is to traverse. The footstep planner produces trajectories of the robot from start to goal over the terrain map. We then apply MMPBOOST again using regression trees with 10 terminal nodes as the base classifier to learn cost features and weights that turn height maps into cost functions so that a 2-dimensional planner over the cost map mimics the body trajectory. We apply the planner to two robots: first the HRP-2 biped robot and second the LittleDog⁶ quadruped robot. The quadruped tests were demonstrated on the robot.⁷

Figure 6.10 shows the resulting computational speedups (and the performance gains) of planning with the learned MMPBOOST heuristic over two previously implemented heuristics: a simple Euclidean heuristic that estimates the cost-to-go as the straight-line distance from the current state to the goal; and an alternative 2-dimensional navigational planner whose cost map was hand engi-

⁶Boston Dynamics designed the robot and provided the motion capture system used in the tests.

⁷A video demonstrating the robot walking across a terrain board is provided with this paper.

neered. We tested three different versions of the planning configuration: (1) no inflation, in which the heuristic is expected to give its best approximation of the exact cost so that the heuristics are close to admissible (Euclidean is the only one who is truly admissible); (2) inflated, in which the heuristics are inflated by approximately 2.5 (this is the setting commonly used in practice for these planners); and (3) Best-first search, in which search nodes are expanded solely based on their heuristic value. The cost diff column relates on average the extent to which the cost of planning under the MMPBOOST heuristic is above or below the opposing heuristic. Loosely speaking this indicates how many more footsteps are taken under the MMPBOOST heuristic, i.e. negative values support MMPBOOST. The speedup column relates the average ratio of total nodes searched between the heuristics. In this case, large values are better, indicating the factor by which MMPBOOST outperforms its competition.

The most direct measure of heuristic performance arguably comes from the best-first search results. In this case, both the biped and quadruped planner using the learned heuristic significantly outperform their counterparts under a Euclidean heuristic.⁸ While Euclidean often gets stuck for long periods of time in local minima, both the learned heuristic and to a lesser extent the engineered heuristic are able to navigate efficiently around these pitfalls. We note that A* biped performance gains were considerably higher: we believe this is because orientation plays a large role in planning for the quadruped.

⁸The best-first quadruped planner under the MMPBOOST heuristic is on average approximately 1100 times faster than under the Euclidean heuristic in terms of the number of nodes searched.

Chapter 7

Maximum Margin Structured Classification

Up to this point, we have discussed our algorithms in the context of inverse optimal control based imitation learning. However, as we stated up front, the MMP framework was derived as a reduction from inverse optimal control to a form of supervised machine learning known as maximum margin structured classification (MMSC). In this chapter, we explore this connection further and discuss how the subgradient and functional gradient algorithms for optimization presented in Chapters 3 and 4 apply to this more general setting. Moreover, we specialize the theoretical batch convergence/generalization and online regret bounds derived in Chapter 3 to MMSC. These theoretical results carry over to MMP as well since it is a specific form of MMSC.

Developing subgradient and functional gradient algorithms for MMP proved crucial for practical and efficient implementation of learning. Alternative past techniques for optimization in MMSC were either slow to converge or too memory intensive to be practical in this setting.

Historically, starting with the support vector machine (SVM), learning methods formalized as convex programming, especially quadratic programming (QP), have been optimized by exploiting the theory of convex duality (Boyd & Vandenberghe, 2004) under the suggestion that optimizing the dual of the convex program can be more efficient than directly optimizing the primal. Many algorithms, such as interior point methods are indeed quick to apply, but they can scale cubically in the number of constraints. As we have seen in MMP, the number of constraints in the QP

formulation can be impractically large, making such techniques infeasible for large scale structured prediction problems (see Chapter 5).

Because of their success for SVMs and other smaller-scale kernel machines, dual optimization remains prominent in a wide range of learning techniques. Accordingly, much of the early research into MMSC focused on methods for optimizing in the dual space starting with the original learning procedure proposed in (Taskar, Guestrin, & Koller, 2003), which was a variant on a popular early SVM training algorithm known as sequential minimal optimization (SMO).

Unfortunately, no formal guarantees could be proven for the SMO variant, which lead to a barrage of research into making these techniques more efficient between 2003 and 2006. The first solid convergence result was proven for an algorithm that performed exponentiated gradient in the dual space (Bartlett *et al.*, 2004). Analysis of this algorithm proved a sublinear rate of convergence and demonstrated its empirical improvement over the SMO algorithm. Later, another algorithm was developed that leveraged a classical saddle-point optimization routine known as the extragradient method (Taskar, Lacoste-Julien, & Jordan, 2006). Analysis of this algorithm showed that it converged at a linear rate to the optimum. Focus throughout this period remained in the dual space, and the application of these algorithms to larger structured prediction problems, such as learning to plan, remained impractical.

Our work demonstrates that computing subgradients of the primal objective is straightforward and cheap, both in terms of computation and memory, when efficient inference algorithms exist (Ratliff, Bagnell, & Zinkevich, 2006, 2007a). The application of the subgradient method in the primal is, therefore, simultaneously faster and more widely applicable to a range of structured prediction problems than existing alternatives. Since 2005, there has been a separate line of work on cutting plane techniques and bundle methods for maximum margin structured classification¹ that also operate in the primal space (Tsochantaridis *et al.*, 2005). These convex optimization techniques work well in practice and, indeed, we have generalized bundle methods to function

¹This work derived what the authors call *structural SVMs* independently of the MMSC formalism. They describe both a margin-scaling variant and a slack-scaling variant. Although, the latter can be difficult to apply in practice, the former is equivalent to what Taskar et al. call MMSC. We choose to use the term MMSC in relation to our work because it was under this name that the first generalization analysis was presented for this form of structured prediction (Taskar, Guestrin, & Koller, 2003).

spaces (see Chapter 4) in order to leverage their representational efficiency.

There is currently a debate raging in the machine learning community regarding the relative merits of bundle methods in machine learning (Smola, Vishwanathan, & Le., 2008; Joachims, 2006) and the subgradient method (Shalev-Shwartz, Singer, & Srebro, 2007; Bottou & Bousquet, 2008; Shalev-Shwartz & Srebro, 2008). In both cases, essentially the same convergence guarantees are available in terms of batch optimization. However, subgradient methods scale well to very large data sets and, indeed, continuous streams of data in online settings. In these online settings, we can develop strong theory using regret analysis techniques that lends itself well to other areas of machine learning theory including batch generalization and convergence (see Chapter 3). This chapter adds to the arguments in favor of subgradient methods by using them to analyze online regret, batch generalization, and convergence of maximum margin structured classification techniques.

The application of the subgradient method to MMP was, therefore, crucial to the success of this framework for inverse optimal control. Our analysis shows that the algorithm achieves linear convergence, sublinear regret, and strong generalization guarantees. Moreover, its memory requirements are determined primarily by the requirements of the inference algorithm. In many cases there exist efficient specialized inference algorithms that the subgradient method exploits. The implementation of this learning algorithm is simple and has intuitive appeal since an integral part of the computation comes from running the inference algorithm being trained in the inner loop.

This property distinguishes our algorithm from other dual-optimization procedures for MMSC. Typically, algorithms that optimize in the dual exploit structure by formulating the inference algorithm as a linear program (LP). However, this transformation means that the inference algorithm used during training may be different from the algorithm used at test time. This discrepancy is particularly unsettling when inference can be implemented only approximately. While the size of the approximation errors may be bounded, the type of errors produced during training may differ from those used at test time.

Additionally, this work connects two distinct threads of research in structured prediction. We show that the gradient descent approach to learning graph transformer backpropagation networks

pioneered in (LeCun *et al.*, 1998) may be straightforwardly extended to solve the novel, margin-scaling structured classification approach developed by (Taskar, Lacoste-Julien, & Jordan, 2006).² This yields perhaps the simplest, most computationally efficient algorithms for solving structured maximum margin problems. The application of the subgradient method to the structured margin loss functions brings benefits concomitant with convexity: efficient global optimization, small online regret, and new bounds on generalization error for these algorithms.

Further, we study the robustness of these algorithms to approximate settings, namely, when inference is only approximate or subgradients cannot be computed exactly. Finally, we consider application of our techniques to two previously studied classification problems.

7.1 Maximum margin structured classification

We begin by generalizing the construction of MMP to the general class of MMSC problems. In this setting, we attempt to predict a structured object $y \in \mathcal{Y}(x)$ (e.g. a parse tree, label sequence, robot trajectory) from a given input $x \in \mathcal{X}$. For our purposes we assume that the inference problem can be described in terms of a computationally tractable max over a score function $s_x : \mathcal{Y}(x) \rightarrow \mathbb{R}$ such that $y^* = \arg \max_{y \in \mathcal{Y}(x)} s_x(y)$ and take as our hypothesis class functions of the linear form $h(x; w) = \arg \max_{y \in \mathcal{Y}(x)} w^T f(x, y)$, with $w \in \mathcal{W}$ for some convex set $\mathcal{W}$.

We focus on the widespread case of MMSC in which inference can be written in a succinct form $\mu^* = \arg \max_{\mu \in \mathcal{G}_x} w^T F_x \mu$, where $F_x \in \mathbb{R}^{d \times B_x}$, is an appropriately defined feature matrix with bounded feature values $(F_x)_{ij} \in [0, 1]$. Here d denoting the dimension of the feature space and B_x denoting the number of bits being predicted for structured input element x . We additionally have each element μ_j bounded by $\mu_j \in [0, 1]$. For instance, in the case of MMP with deterministic planning, F_x is the feature matrix with d defined as a dimension of the feature space, B defined as the number of state-action pairs in the MDP, and each μ_j is an indicator variable specifying whether the path passes through a given state-action pair. Similar definitions hold across a range of MMSC problems (Taskar *et al.*, 2005; Anguelov *et al.*, 2005; Taskar, Lacoste-Julien, & Jordan,

²Recent other work has attempted to make similar connections including suggesting related loss functions (LeCun *et al.*, 2007) that are *not* equivalent to the structured maximum margin criteria. Section 7.4 suggests these methods have poorer performance both empirically and theoretically.

2006). The inference procedure, under this parameterization, may be written

$$h(x; w) = \arg \max_{\mu \in \mathcal{G}(x)} w^T F_x \mu. \quad (7.1)$$

When a data element (x_i, y_i) is available, we often abbreviate $\mathcal{G}_{x_i} = \mathcal{G}_i$ and $F_{x_i} = F_i$. Let $\mathcal{L}(y_i, y) = \mathcal{L}_i(y)$ be a loss function measuring the discrepancy between the true label y_i and an alternative label y . One natural choice of this loss function for sequence labeling is the generalized hamming loss that measures the number of labels that disagree between two label sequences. In the case of MMP, we often use a smoothed version of this generalized hamming loss that measures how far each state along the proposed path is from the desired path. As in the presentation of MMP, we consider only the class of loss functions of the following linear form $\mathcal{L}_i(y) = l_i^T \mu$, where l_i is some loss vector and μ is the vector representing y in the above representation. Moreover, the loss must be nonnegative for all y and zero at y_i .

7.1.1 Batch learning

In the batch setting, the learner is given a preselected set of data $\mathcal{D} = \{(x_i, y_i, \mathcal{L}_i(y))\}_{i=1}^N = \{(F_i, \mu_i, l_i)\}_{i=1}^n$ from which it must generalize. Informally, the learner must find a small hypothesis that scores the desired label better than any other label by a margin that scales with the loss of that label. If it succeeds, then at least on the training examples, the inference algorithm will output the desired label. Additionally, since the learner made the effort to find a hypothesis that achieves a loss-scaled margin over all other labels, and since the hypothesis is reasonably small, it is unlikely that the learner overfit. This intuition indicates why we believe such an algorithm will generalize well.

In the following exposition, we use the matrix notation introduced above exclusively. Formally, this margin criterion gives us the following constraint:³

$$\forall i, \mu \in \mathcal{G}_i, \quad w^T F_i(\mu_i) \geq w^T F_i(\mu) + l_i^T \mu \quad (7.2)$$

³(Tsochantaridis *et al.*, 2005) describes an alternative formulation for MMSC based around scaling the slack variables by the loss rather than scaling the margin. Subgradient methods are applicable to this formulation as well, though we do not formally discuss this case.

Maximizing the left hand side over all $y \in \mathcal{Y}_i$, and adding slack variables, we can express this mathematically as following compact convex program:

$$\begin{aligned} \min_{w, \zeta_i} \quad & \frac{\lambda}{2} \|w\|^2 + \frac{1}{N} \sum_i \frac{\zeta_i}{B_i} \\ \text{s.t.} \quad & \forall i \quad w^T F_i \mu_i + \zeta_i \geq \max_{y \in \mathcal{Y}_i} (w^T F_i \mu + l_i^T \mu) \end{aligned} \quad (7.3)$$

where $\lambda \geq 0$ is a hyperparameter that trades off constraint violations for margin maximization (i.e. fit for simplicity). In many structured prediction problems, the size of the examples (more precisely, B_i) may differ significantly. We normalize the slacks by $\frac{1}{B_i}$ in order to ensure that each example receives equal weight in the objective function. (Intuitively, in the case of MMP, demonstrating a short range maneuver, e.g. avoiding a rock, is often a specific attempt by the trainer to introduce an important concept. Normalizing in this way prevents large examples from dominating these smaller examples simply because they are large.)

We note that the constraints of this convex program are tight (equality holds at the optimum) so we can place them directly into the objective. Doing so we arrive at the following regularized risk function:⁴

$$\begin{aligned} c(w) &= \frac{1}{N} \sum_{i=1}^N r_i(w) + \frac{\lambda}{2} \|w\|^2 \\ \text{where } r_i(w) &= \frac{1}{B_i} \left(\max_{\mu \in \mathcal{G}_i} (w^T F_i \mu + l_i^T \mu) - w^T F_i \mu_i \right) \end{aligned} \quad (7.4)$$

7.1.2 Online learning

We consider three online settings for MMSC. In the first setting, we consider unregularized loss functions, while in the second setting, we consider objectives augmented with a decreasing sequence of regularizers.

⁴More generally, we can scale the risk by a data dependent constant and raise it to a power $q \geq 1$ as is done in (Ratliff, Bagnell, & Zinkevich, 2006). The resulting objective is still convex and a chain rule for subgradients allows for the calculation of its subgradient. The primary components of this theory are captured most simply with $q = 1$, however, so we have opted to leave it out.

Algorithm 13 MMSC subgradient calculation

```
1: procedure SUBGRADMMSC(  $(x_i, y_i), \mathcal{L}_i(y), f_i : \mathcal{X} \rightarrow \mathbb{R}^d, w \in \mathcal{W}$  )  
2:    $y^* = \arg \max_{y \in \mathcal{Y}} w^T f_i(y) + \mathcal{L}_i(y)$   
3:    $g \leftarrow g + f_i(y^*) - f_i(y_i)$   
4:   return  $\frac{1}{B_i} g$   
5: end procedure
```

1. **Unregularized:** In the first setting, which we consider the *classical* setting, the online learner receives a sequence of unregularized risk functions $\{r_t(\cdot)\}_{t=1}^T$. At each round, the learner chooses a hypothesis from a convex set $w \in \mathcal{W}$.
2. **Constant regularization:** In the second setting, the learner receives a sequence of objective functions $\{c_t(\cdot)\}_{t=1}^T$ with constant regularization of the form $c_t(w) = r_t(w) + \frac{\lambda}{2} \|w\|^2$.
3. **Attenuated regularization:** In the third setting, the learner receives a sequence of objective functions $\{c_t(\cdot)\}_{t=1}^T$ with decreasing regularization of the form $c_t(w) = r_t(w) + \frac{\lambda}{2\sqrt{t}} \|w\|^2$.

In all cases, we measure regret in terms of the online prediction loss $\sum_{t=1}^T l_t(w_t) \leq \sum_{t=1}^T r_t(w_t)$ and compare against the best risk value $\min_{w \in \mathcal{W}} \sum_{t=1}^T r_t(w)$ without reference to the regularization. Regret and generalization theorems for these settings are presented below in Section 7.2.

7.1.3 Subgradient computation

Algorithm 13 demonstrates how to calculate the exact subgradient for a single term of the MMSC regularized risk function. Following the negative of this subgradient has intuitive appeal: the algorithm decreases the score if it is too high and increases the score if it is too low. The theoretical analysis and experimental results that follow show that even this simple, intuitively appealing, algorithm perform well for structured learning.

In both the online and the batch settings, we apply the subgradient methods discussed in Chapter 3. In the batch setting, at each iteration we accumulate the subgradients across all examples (and the regularizer) and take a single step in the resulting direction. We analyze the algorithm in terms of its rate of convergence. On the other hand, in the online settings we take a step in the direction of only the current objective function's subgradient at the end of each round.

In the classical setting this subgradient is simply the subgradient of the risk term, while in the attenuated regret setting it additionally includes a regularization contribution. For these settings, we bound the online prediction regret defined in Chapter 3.

We additionally bound the generalization performance of the hypothesis returned by the online learner in both online settings when applied to a batch learning problem.

7.2 Theoretical results

Framing these structured learning problems as convex regularized risk functions and optimizing them via variants of the subgradient method allows for straightforward analysis of the optimization and learning convergence in the batch, online, and approximate settings. Here we consider the case in which we can compute the subgradients exactly. Approximate settings are analyzed in Section 7.3.

Under these definitions, we can easily bound the size of the subgradient for MMSC as presented in the following lemma.

Lemma 7.2.1: Subgradient bound for MMSC. *Assume that the l_2 -norm of the feature vectors forming the columns of F_i are bounded by 1. Then the l_2 -norm of any MMSC risk function subgradient is bounded by $\|\nabla r_i(w)\| \leq 1$, where $r_i(w)$ is defined as in Equation 7.4.⁵*

Proof. The risk gradient takes the form $\nabla r_i(w) = \frac{1}{B_i} F(\mu_i^* - \mu_i) = \frac{1}{B_i} \sum_{b=1}^{B_i} \alpha_i^b f_i^b$ where $\alpha_i^b = \mu_i^{*b} - \mu_i^b$ has absolute value at most 1 and f_i^b is the b^{th} column of F_i with l_2 -norm bounded by 1. By the triangle inequality,

$$\|\nabla r_i(w)\| \leq \frac{1}{B_i} \sum_{b=1}^{B_i} |\alpha_i^b| \|f_i^b\| \leq \frac{1}{B_i} \sum_{b=1}^{B_i} 1 = 1. \quad (7.5)$$

□

⁵Alternatively, we can assume that each entry in the feature matrix F_i is bounded in absolute value by 1 (i.e. that the l_1 -norm of each column vector is bounded by d). In that case, the risk gradient becomes bounded by d , the dimension of the feature space. Under this setting the subgradient method will typically have a linear dependence on the number of features. However, we can instead explore one of the feature selection gradient-based optimization variants such as the parametric exponentiated gradient descent algorithm which has only a logarithmic dependence on the number of features Cesa-Bianchi & Lugosi (2006).

7.2.1 Convergence bounds of batch learning

We first explore the convergence properties of the subgradient and incremental subgradient methods for MMSC. This first theorem bounds the number of iterations required to converge to a solution with ϵ error.

Theorem 7.2.2: Convergence bounds of subgradient MMSC. *The subgradient method applied with step size sequence $\{\frac{1}{\lambda t}\}_{t=1}^T$ to the MMSC objective function presented in Equation 7.4 converges to an ϵ -accuracy in $\tilde{O}\left(\frac{2}{\epsilon\lambda}\right)$ iterations. Moreover, the incremental subgradient method will converge in $\tilde{O}\left(\frac{1}{N}\frac{2}{\epsilon\lambda}\right)$ iterations.*

Proof. These results follow immediately from Theorems 3.4.1 and 3.4.2 with the identity $G = 1$ from Lemma 7.2.1. $\square$

In terms of convergence rate of the iterate to the global optimum $w^* \in \mathcal{W}$, we can say the following about the subgradient method:

Theorem 7.2.3: Linear convergence rate of subgradient MMSC. *The subgradient method applied with constant step size $\alpha \leq \frac{1}{\lambda}$ to the MMSC objective function presented in Equation 7.4 converges linearly to a region around the minimum of size $2\sqrt{\frac{\alpha}{\lambda}}$.*

Proof. This result follows immediately from Theorem 3.4.3 and the gradient bound presented in Lemma 7.2.1. $\square$

This theorem says that, with a sufficiently small step size, the subgradient method converges at a linear rate to a small region around the optimum.

7.2.2 Sublinear regret of online learners

The next theorem analyzes the regret of each of the online settings outlined in Section 7.1.2. In each case, we achieve a sublinear regret.

Theorem 7.2.4: Regret bounds for online subgradient MMSC. *Let $\lambda > 0$ be a regularization constant and denote $w^* = \arg \min_{w \in \mathcal{W}} \sum_{t=1}^T r_t(w)$.*

1. **Unregularized.** Let our sequence of objectives be $\{r_t(w)\}_{t=1}^T$. Then we achieve a regret bound of the form $\sum_{t=1}^T l_t(w) \leq \sum_{t=1}^T r_t(w^*) + \frac{2\sqrt{2}}{\lambda} \left(\sqrt{T} - \frac{1}{4} \right)$.
2. **Constant regularization.** Let our sequence of objectives be $\{r_t(w) + \frac{\lambda}{2}\|w\|^2\}_{t=1}^T$. Then we achieve a regret bound of the form $\sum_{t=1}^T l_t(w) \leq \sum_{t=1}^T r_t(w^*) + 2\|w^*\|\sqrt{T(1 + \log T)}$.
3. **Attenuated regularization.** Let our sequence of objectives be $\left\{r_t(w) + \frac{\lambda}{2\sqrt{t}}\|w\|^2\right\}_{t=1}^T$. Then we achieve a regret bound of the form $\sum_{t=1}^T l_t(w) \leq \sum_{t=1}^T r_t(w^*) + 4\|w^*\| \left(\sqrt{T} - \frac{1}{2} \right)$.

Proof. These results follow from Theorems 3.2.1, 3.2.3, and 3.2.5 using $G = 1$ from Lemma 7.2.1. In the unregularized bound, we constrain the size of the space to be $\frac{1}{\lambda}$ to match radius of convergence of a regularized problem with regularization constant λ per Theorem 3.1.3. $\square$

7.2.3 Generalization bounds

Our online algorithm also inherits interesting generalization guarantees when applied in the batch setting. In the next theorem, we utilize the analysis of Section 3.3 to derive generalization bounds using the regret bounds from Theorem 7.2.4.

Theorem 7.2.5: Generalization bounds for online subgradient MMSC. Let $\lambda > 0$ be a regularization constant and denote $w^* = \arg \min_{w \in \mathcal{W}} \sum_{t=1}^T r_t(w)$.

1. **Unregularized.** Let our sequence of objectives be $\{r_t(w)\}_{t=1}^T$. Then we achieve a generalization bound of the form $E(l_{T+1}(\bar{w})) \leq \min_{w \in \mathcal{W}} \frac{1}{T} \sum_{t=1}^T r_t(w) + \frac{1}{\lambda} \left(1 + \frac{3}{2} \sqrt{\log \frac{1}{\delta}} \right) \sqrt{\frac{2}{T}}$.
2. **Constant regularization.** Let our sequence of objectives be $\{r_t(w) + \frac{\lambda}{2}\|w\|^2\}_{t=1}^T$. Then we achieve a generalization bound of the form $E(l_{T+1}(\bar{w})) \leq \min_{w \in \mathcal{W}} \frac{1}{T} \sum_{t=1}^T r_t(w) + \left(\|w^*\| \sqrt{2(1 + \log T)} + \frac{3}{2\lambda} \sqrt{\log \frac{1}{\delta}} \right) \sqrt{\frac{2}{T}}$.
3. **Attenuated regularization.** Let our sequence of objectives be $\left\{r_t(w) + \frac{\lambda}{2\sqrt{t}}\|w\|^2\right\}_{t=1}^T$. Then we achieve a generalization bound of the form $E(l_{T+1}(\bar{w})) \leq \min_{w \in \mathcal{W}_T} \frac{1}{T} \sum_{t=1}^T r_t(w) + \left(2\sqrt{2}\|w^*\| + \frac{3}{2\lambda} \sqrt{\log \frac{1}{\delta}} \right) \sqrt{\frac{2}{T}}$.

Proof. These results follow from Theorems 3.3.2, 3.3.3, and 3.3.4 using $G = 1$ from Lemma 7.2.1. Given that $\|w^*\| \leq \frac{1}{\lambda}$ per Theorem 3.1.3, the risk term is bounded by $\frac{1}{\lambda}$ and the regularization term is

bounded by $\frac{1}{2\lambda}$. Each objective is, therefore, bounded by $L = \frac{1}{\lambda} + \frac{1}{2\lambda} = \frac{3}{2\lambda}$. In the unregularized bound, we additionally constrain the size of the space to be $\frac{1}{\lambda}$ to match the bound on $\|w^*\|$ of a regularized problem with regularization constant λ . $\square$

These generalization bounds are similar in form to previous generalization bounds given using covering number techniques (Taskar, Guestrin, & Koller, 2003). Importantly, though, this approach removes entirely the dependency on the number of bits B being predicted in structured learning. Most existing techniques introduce a $\log B$ factor for the number of predicted bits.

7.3 Robustness to approximate settings

This section derives two robustness results. In the first subsection, we consider the case in which inference is only approximate, and in the second subsection we analyze the case in which we can only compute approximate subgradients of the structured margin objective. Unfortunately, we find that the approximate subgradient resulting from approximate inference is not that which is needed in the latter theoretical analysis, but nevertheless these results illustrate a general robustness in our algorithm.

7.3.1 Using approximate inference

Following (Shmoys & Swamy, 2004), we define a γ -subgradient similar to the way an exact subgradient is defined via Equation 3.1, but we replace the inequality with $\forall w' \in \mathcal{W}, h(w') \geq h(w) + g^T(w' - w) - \gamma h(w)$. In other words, we allow the lower bound to be violated slightly by an amount that scales with the approximation constant γ and objective value $h(w)$ at the point in question.

Additionally, we define an approximate inference operator η -max as follows:

Definition 7.3.1: η -max. *We call an algorithm an η -approximate max operator, denoted $\max^\eta$, if for any collection $\{s_y \mid y \in \mathcal{Y}\}$, we are guaranteed $\max_{y \in \mathcal{Y}}^\eta s_y \geq \eta \max_{y \in \mathcal{Y}} s_y$. η is known as the competitive ratio of the approximate max.*

It is well known that if each s_y is a convex function over $\mathcal{W}$, then $h(w) = \max_{y \in \mathcal{Y}} s_y(w)$ is a convex

function and $\nabla s_{y^*}(w)$ is a subgradient of that function for any $y^* = \arg \max_{y \in \mathcal{Y}} s_y(w)$. We prove here a generalized theorem of this sort in terms of an approximate max operator.

Theorem 7.3.2: η -max gives $(1 - \eta)$ -subgradient. Define $h = \max_{y \in \mathcal{Y}} s_y(w)$ and let $g = \nabla s_{y_\eta^*}(w)$ where $y_\eta^* = \arg \max_{y \in \mathcal{Y}}^\eta w_y(w)$. Then g is a $(1 - \eta)$ -subgradient per Definition 7.3.1.

Proof. Since g is a subgradient of the score function $s_{y_\eta^*}(w)$, we have $g^T(w' - w) \leq s_{y_\eta^*}(w') - s_{y_\eta^*}(w) \leq h(w') - \eta h(w)$, where the final inequality comes from the optimality of h and the definition of η -max. Rearranging, we get $h(w') - h(w) \geq g^T(w' - w) - (1 - \eta)h(w)$. $\square$

7.3.2 Optimizing with approximate subgradients

In this section, we can bound the regret of following approximate subgradients rather than exact subgradients within the online setting defined in Section 7.1.2. Borrowing notation from that Section and following arguments similar to those in Theorem 7.2.4, we can derive the following

$$\begin{aligned} \sum_{t=1}^T \mathcal{L}_t(y_t^*) &\leq \sum_{t=1}^T c_t(w_t) \\ &\leq \sum_{t=1}^T r_t(w^*) + \|w^*\| \sqrt{T(1 + \ln T)} + \gamma \sum_{t=1}^T c_t(w_t) \end{aligned}$$

Another, potentially more insightful, way to write this is in terms of the average regret. In this case, if we denote $S(T) = \|w^*\| \sqrt{T(1 + \ln T)}$ (note that this is a sublinear function), we find

$$\frac{1}{T} \sum_{t=1}^T \mathcal{L}_t(y_t^*) \leq \frac{1}{1 - \gamma} \left(\frac{1}{T} \sum_{t=1}^T r_t(w^*) + \frac{S(T)}{T} \right) \quad (7.6)$$

$$\xrightarrow{T \rightarrow \infty} \frac{1}{1 - \gamma} R, \quad (7.7)$$

where $R = \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=1}^T r_t(w^*)$ is the asymptotic optimal average risk. Equation 7.7 says that in the limit, we have paid on average only a factor $\frac{1}{1 - \gamma}$ more regret each time step than if we had been able to compute and follow exact subgradients.

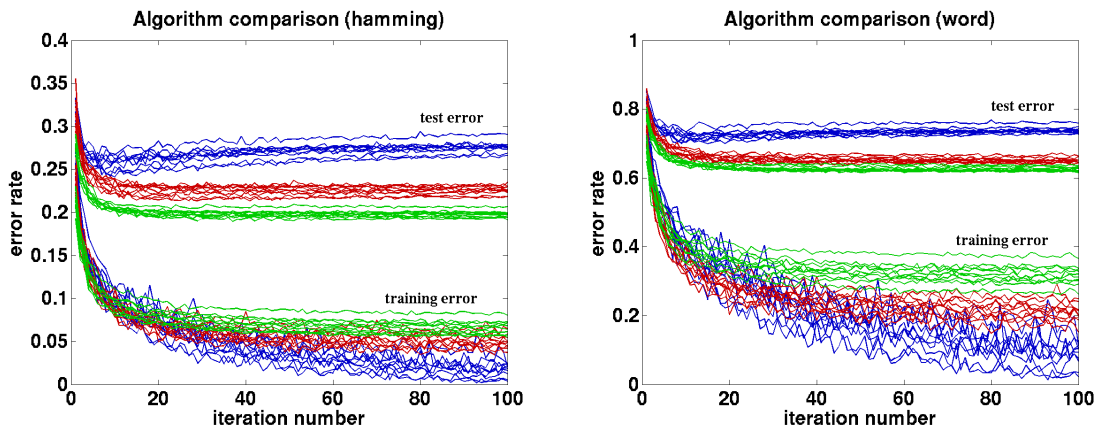


Figure 7.1: These plots show a comparison between the structured margin (green), perceptron (blue), and unstructured margin (red) algorithms using 10 fold cross-validation iterations of 600 training examples and 5500 test examples. The figure on the left displays error in terms of hamming loss, and the figure on the right displays word classification error. Upper lines of a given color represent test error and lower lines represent training error. See text for details.

7.4 Experimental results

We present experimental results on two previously studied structured classification problems: optical character recognition (Taskar, Guestrin, & Koller, 2003), and LADAR classification (Angelov *et al.*, 2005).

7.4.1 Optical character recognition

We implemented the incremental subgradient method⁶ for the sequence labeling problem originally explored by (Taskar, Guestrin, & Koller, 2003) who used the Structured SMO algorithm.⁷ Running our algorithm with 600 training examples and 5500 test examples using 10 fold cross validation, as was done in (Taskar, Guestrin, & Koller, 2003), we attained an average prediction error of 0.20 using a linear kernel. This result is statistically equivalent to the previously published result; however, the entire 10 fold cross validation run completed within 17 seconds. Furthermore, when running the experiment using the entire data set partitioned into 10 folds of 5500 training and 600 test examples each, we achieved a significantly lower average error of 0.13, again using the linear kernel.

⁶Similar to the online method, this method updates the weights with each term's subgradient contribution rather than combining them into a single step.

⁷This data can be found at <http://www.cs.berkeley.edu/~taskar/ocr/>

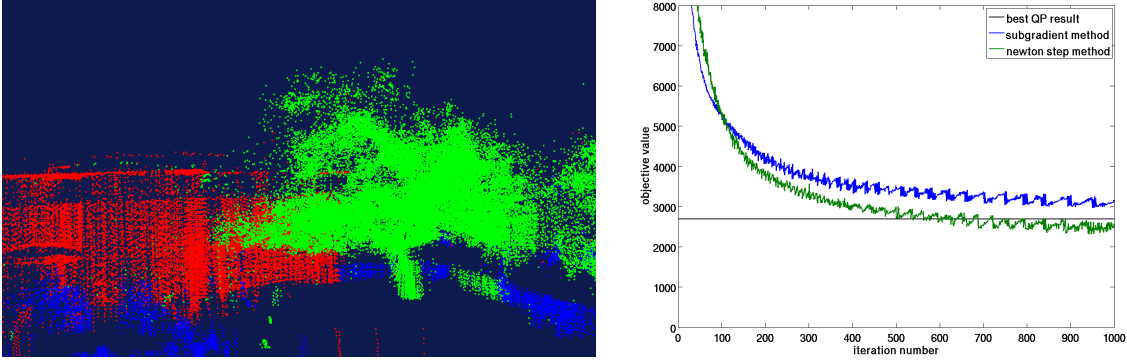


Figure 7.2: Left: Pictorial representation of LADAR classification results on a test region. Classes are denoted as red: building, green: tree, and blue: shrubbery. Right: LADAR scan classification results. Subgradient method (blue) converges off the edge of the graph, but within the same amount of time as it took to obtain the best QP result. The Newton Step method converges significantly faster. See Section 7.4.2 for details.

We additionally compared our algorithm to two previously proposed algorithms: the perceptron algorithm, and the unstructured margin (LeCun *et al.*, 2007).⁸ We ran each algorithm using 10 fold cross validation with the partitioning of 600 training examples and 5500 test examples. Figure 7.1 plots both the training error (lower lines) and the test error (upper lines) for each in terms of both hamming loss (left) and word classification (right). The structured margin algorithm (our algorithm), displayed in green, generalizes noticeably better than the other two algorithms. The perceptron algorithm (blue) overfits very quickly on this problem, and the unstructured margin algorithm (red) falls somewhat between the other two in terms of performance. In all cases, we used a step size rule of $\alpha_t = \frac{1}{2\sqrt{t}}$ and set the regularization constant to $\lambda = \frac{1}{200N}$ where N is the number of training examples.

7.4.2 LADAR scan classification

We next consider application of subgradient techniques to a problem of classifying LADAR point clouds captured by a mobile robot. Full details of the training data can be found in (Anguelov *et al.*, 2005). Briefly, a maximum margin structured classification problem is set up to classify each point in a point cloud of laser range data into one of four classes: ground, shrubbery, trees, and building. One-vs-all classification of ground based on a height threshold was reportedly simple,

⁸The perceptron risk is given by $r_i(w) = \max_{y \in \mathcal{Y}_i} w^T f_i(y) - w^T f_i(y_i)$; The unstructured margin risk is given by $r_i(w) = \max\{0, 1 + \max_{y \in \mathcal{Y}_i \setminus y_i} w^T f_i(y) - w^T f_i(y_i)\}$.

effectively reducing the problem to a three class classification problem (per LADAR point).

To capture spatial correlation between classification labels of the LADAR points, an associative conditional Markov random field (AMN) between nearby points was constructed throughout the point cloud. Labels for the point clouds were determined by the joint maximum probability labeling of the nodes in the Markov network. (Anguelov *et al.*, 2005) built the maximum margin structured classification problem as a quadratic program (QP) and solved it using CPLEX, a well known commercial solver. Node potentials were log-linear in 90 features each derived from the original LADAR data (e.g. spin images features, distance from ground) and edge potentials were constant for each class. See (Anguelov *et al.*, 2005) for more additional information on the features.

Limited by CPLEX’s fairly intensive memory requirements, the training set consisted of only approximately 30 thousand of the original 20 million points in the data set. We note that the subgradient methods we here have only linear memory requirements in the number of training points.

Moreover, the quadratic programming problem used for training was derived as a relaxation to the intractable integer programming problem, but the alpha-beta swap/expansion algorithm (Szeliski *et al.*, 2006) was employed for approximate inference at test time. While both of these algorithms admit a constant factor approximation, they qualitatively differ in practice. The subgradient method has the additional appeal of relying solely on the alpha-beta swap/expansion algorithm (Szeliski *et al.*, 2006), iteratively optimizing it to perform well.

We ran the subgradient method and a modified approximate Newton step method⁹ to optimize this problem, the results of which are shown in Figure 7.2. We preprocessed the node features using a whitening operation to remove linear dependencies and poor conditioning of the features. Whitening intuitively amounts to scaling the principle directions of variance of the feature vectors inversely proportional to the standard deviation along those directions.

The black horizontal line across Figure 7.2 denotes the minimum objective value attained by CPLEX on this problem, and the blue and green plots, respectively, show the objective values per

⁹This more complex variant works better in practice for certain problems and is an extension of Newton type methods to nondifferentiable problems where the Hessian might not exist. See (Hazan, Agarwal, & Kale, 2006) for details and analysis of this method. Briefly, under the Newton step method, the update rule becomes $w_{t+1} \leftarrow w_t - \alpha_t(H_t + \epsilon I)^{-1}g_t$, where g_t is the subgradient at time t and H_t is updated as $H_{t+1} \leftarrow \frac{t}{t+1}H_t + \frac{1}{t+1}g_t g_t^T$.

iteration of the subgradient method and the Newton step method. The Newton step objective progression drops below the smallest CPLEX value within 550 iteration, which is equivalent to approximately 15 minutes of CPU time. This computation time is primarily dominated by executing of the alpha-beta expansion algorithm (Szeliski *et al.*, 2006). While the first-order subgradient method lags behind the Newton step counterpart, it is important to note that it also does well, surpassing the CPLEX result by iteration 1950. This amounts to approximately 65 minutes of computation time, the same amount of time as was reported in (Anguelov *et al.*, 2005) for CPLEX training. Importantly, however, both of these subgradient-based algorithms scale to data set sizes significantly greater than those reported here, which neared the upper bound of what CPLEX could originally handle. Indeed, they are limited solely by the computational performance of the inference algorithm.

Recent work by Munoz et al. has extended this application to more sophisticated AMN models capable of distinguishing linear segments such as wire from other classes including vegetation and facade (Munoz, Vandapel, & Hebert, 2008). This ground-breaking work, in conjunction with their later work exploring the importance of the exponentiated functional gradient algorithm for training models with higher-order cliques (Munoz *et al.*, 2009), demonstrate that the algorithms of this thesis are the state-of-the-art in LADAR classification.

Chapter 8

Maximum Margin Structured Regression

Value functions are unique in that the values are represented using a structured definition that integrates information across the entire space. Value function approximation techniques, however, often attempt to represent each value as a function of a single feature vector without exploiting the structure of their computation. In this chapter, we develop a novel form of structured prediction called *maximum margin structured regression* (MMSR) which generalizes traditional ϵ -insensitive support vector regression techniques (Smola & Schölkopf, 2003). We focus here on value function approximation, but our technique is a general approach to regression that exploits structure in the computation of regressed values.

8.1 Motivation

In Section 6.6.2, we presented an application of maximum margin planning to heuristic learning and demonstrated the approach on footstep planning. Splitting heuristic learning into two steps (where we first learn to predict the correct path, and then scale the cost of the predicted paths to match the desired cost-to-go values) allowed us to directly utilize maximum margin planning algorithms, but it also imposed restrictions on the learning approach. The primary component of learning optimized the wrong risk function. The algorithm we present here integrates these two steps into a

single framework which we call maximum margin structured regression. Rather first than training the planner to match the behavior demonstrated by the examples and then scaling the result, we directly train the planner to output plans whose costs match the desired values. Importantly, the framework retains its convexity and lends itself to the same optimization tools used under maximum margin planning. In this section, we discuss both linear and nonlinear formulation of the problem.

8.2 Defining MMSR

The data set for this problem is similar to the data set used by maximum margin planning, but now each example is augmented with a single scalar value: $\mathcal{D} = \{(\mathcal{M}_i, \mu_i, v_i)\}_{i=1}^N$. Our goal is to find a planner that returns plans with cost-to-go values that match the desired values v_i . While the problem as stated is inherently nonconvex,¹ we attain convexity by utilizing the additional information provided in the example trajectories.

The objective function that governs this algorithm is

$$r(w) = \frac{1}{N} \sum_{i=1}^N \left(\max\{\epsilon, v_i - \min_{\mu \in \mathcal{G}_i} w^T F_i \mu\} + \max\{\epsilon, w^T F_i \mu_i - v_i\} \right) + \frac{\lambda}{2} \|w\|^2 \quad (8.1)$$

$$= \frac{1}{N} \sum_{i=1}^N \left(h_i^l(w) + h_i^u(w) \right) + \frac{\lambda}{2} \|w\|^2. \quad (8.2)$$

In the final expression, we denote the terms measuring the lower bound error as $h_i^l(w) = \max\{\epsilon, v_i - \min_{\mu \in \mathcal{G}_i} w^T F_i \mu\}$ and the terms measuring the upper bound error as $h_i^u(w) = \max\{\epsilon, w^T F_i \mu_i - v_i\}$ to emphasize their roles during learning.

This chapter presents the maximum margin structured regression framework in detail, and reviews an application of this new form of structured prediction to value function approximation that utilizes a minimum cost planner in the inner loop.

¹For instance, a least-squares algorithm under this hypothesis class would optimize the following nonconvex objective: $r(w) = \frac{1}{N} \sum_{i=1}^N (v_i - \min_{\mu \in \mathcal{G}_i} w^T F_i \mu)^2$.

Algorithm 14 MMSR subgradient calculation

```
1: procedure SUBGRADMMSR(  $(x_i, y_i, v_i), f_i : \mathcal{X} \rightarrow \mathbb{R}^d, w \in \mathcal{W}$  )
2:    $y^* = \arg \max_{y \in \mathcal{Y}} w^T f_i(y)$ 
3:   if  $w^T f_i(y^*) > v_i$  then
4:      $g \leftarrow g - f(y^*)$ 
5:   end if
6:   if  $w^T f_i(y_i) < v_i$  then
7:      $g \leftarrow g + f_i(y_i)$ 
8:   end if
9:   return  $g$ 
10: end procedure
```

8.3 Linear derivation and optimization

This section derives a linear form of the maximum margin structured regression algorithm in full. Intuitively, the algorithm proceeds by iteratively planning under the currently hypothesized cost map and (1) increasing the cost of the planned path if that cost is currently lower than the desired cost v_i , and (2) decreasing the cost of the example path if the cost of that path is larger than the desired cost.² This algorithm attempts to push together the cost of the planned path and the cost of the example path, so that they meet at the desired value v_i . Since the example cost upper bounds the (minimizing) planned cost, the learned predictor tends to err on the side of underestimation. When used for heuristic learning, this property promotes admissibility.

We define the value of an action a taken from state s through MDP $\mathcal{M}_i$ as

$$v(s, a) = w^T F_i x^{sa} + \min_{\mu \in \mathcal{G}_i} w^T F_i \mu. \quad (8.3)$$

Intuitively, it is the cost of taking that action from state s and following the optimal policy from then on out.

In the typical value function approximation setting, we are given a data set containing examples of the values that correspond to a particular set of state-action pairs. In this setting, however, we will find that we can form a convex objective function by including additional information about

²Optionally, we can substitute the loss-augmented path for the vanilla planned path in the first step of the algorithm.

the particular policy that formed those values. Therefore, we assume we are provided a data set $\mathcal{D} = \{(\mathcal{M}_i, \mu_i, v_i)\}$. This data set is essentially the same as that seen under the maximum margin planning setting, but for each example, we are provided a target value v_i which specifies the exact value we would like the policy to return for that example.

We can write down a set of constraints on the solution that force the inference algorithm to return a solution of cost v_i for each example i :

$$\forall i, \min_{\mu \in \mathcal{G}_i} w^T F_i \mu \geq v_i - \epsilon \quad (8.4)$$

$$\min_{\mu \in \mathcal{G}_i} w^T F_i \mu \leq v_i + \epsilon. \quad (8.5)$$

In these expressions, $\epsilon > 0$ defines the insensitivity margin. These constraints enforce only that the value returned by the inference algorithm be within ϵ of the desired value v_i .

The first set of constraints given in Equation 8.4 are convex but the second set in Equation 8.5 are not. However, since we are provided with the example policy in the data set, we can replace the minimum cost inference term in these nonconvex constraints with a term representing the value of the example policy.

$$\forall i, \min_{\mu \in \mathcal{G}_i} w^T F_i \mu \geq v_i - \epsilon \quad (8.6)$$

$$w^T F_i \mu_i \leq v_i + \epsilon. \quad (8.7)$$

These modified constraints are now convex, and when they are satisfied, the original set of constraints are also satisfied since $\min_{\mu \in \mathcal{G}_i} w^T F_i \mu \leq w^T F_i \mu_i \leq v_i$.

We again add slacks to allow constraint violations for a penalty, and attempt to maximize the

margin by minimizing the norm on the weight vector

$$\min_{w \in \mathcal{W}} \frac{1}{N} \sum_{i=1}^N (\zeta_i^l + \zeta_i^u) + \frac{\lambda}{2} \|w\|^2 \quad (8.8)$$

$$\forall i \quad \min_{\mu \in \mathcal{G}_i} w^T F_i \mu \geq v_i - \epsilon - \zeta_i^l \quad (8.9)$$

$$w^T F_i \mu_i \leq v_i + \epsilon + \zeta_i^u \quad (8.10)$$

$$\zeta_i^l \geq 0, \quad \zeta_i^u \geq 0. \quad (8.11)$$

In this program we have two sets of slack variables, one for each set of constraints. We denote the slack variables on the lower bound constraints by ζ_i^l , and we denote the slack variables on the upper bound constraints by ζ_i^u .

A change of variable $\tilde{\zeta}_i^l = \zeta_i^l + \epsilon$ and $\tilde{\zeta}_i^u = \zeta_i^u + \epsilon$ eases notation:

$$\min_{w \in \mathcal{W}} \frac{1}{N} \sum_{i=1}^N (\tilde{\zeta}_i^l + \tilde{\zeta}_i^u) + \frac{\lambda}{2} \|w\|^2 \quad (8.12)$$

$$\forall i \quad \min_{\mu \in \mathcal{G}_i} w^T F_i \mu \geq v_i - \tilde{\zeta}_i^l \quad (8.13)$$

$$w^T F_i \mu_i \leq v_i + \tilde{\zeta}_i^u \quad (8.14)$$

$$\tilde{\zeta}_i^l \geq \epsilon, \quad \tilde{\zeta}_i^u \geq \epsilon. \quad (8.15)$$

Placing these constraints into the objective function is more difficult here than it was for maximum margin planning since the constraints are not tight. We first rewrite the constraints in a tight form by solving for slack variables in the value constraints and absorbing the nonnegativity constraints on the slack variables by taking a max as follows:

$$\forall i \quad \tilde{\zeta}_i^l \geq \max\{\epsilon, v_i - \min_{\mu \in \mathcal{G}_i} w^T F_i \mu\} \quad (8.16)$$

$$\tilde{\zeta}_i^u \geq \max\{\epsilon, w^T F_i \mu_i - v_i\}. \quad (8.17)$$

Since these constraints are tight, we can again place them up into the objective function to

produce the maximum margin structured prediction regularized risk function:

$$r(w) = \frac{1}{N} \sum_{i=1}^N \left(\max\{\epsilon, v_i - \min_{\mu \in \mathcal{G}_i} w^T F_i \mu\} + \max\{\epsilon, w^T F_i \mu_i - v_i\} \right) + \frac{\lambda}{2} \|w\|^2 \quad (8.18)$$

$$= \frac{1}{N} \sum_{i=1}^N \left(h_i^l(w) + h_i^u(w) \right) + \frac{\lambda}{2} \|w\|^2, \quad (8.19)$$

where we denote the lower bound terms by $h_i^l(w) = \max\{\epsilon, v_i - \min_{\mu \in \mathcal{G}_i} w^T F_i \mu\}$ and the upper bound terms by $h_i^u(w) = \max\{\epsilon, w^T F_i \mu_i - v_i\}$ for convenience.

This is a convex, though nondifferentiable, objective function. The simplest algorithm for optimizing this objective function is the subgradient method (Ratliff, Bagnell, & Zinkevich, 2007a) which can be computed using the tools discussed in Section 5.3.

The subgradient contributions from the lower bound terms are given by:

$$\nabla h_i^l(w) = \begin{cases} -F_i \mu^* & \text{if } \min_{\mu \in \mathcal{G}_i} w^T F_i \mu < v_i - \epsilon \\ 0 & \text{otherwise,} \end{cases} \quad (8.20)$$

where $\mu_i^* = \arg \min_{\mu \in \mathcal{G}_i} w^T F_i \mu$. Similarly, the subgradients contributions from the upper bound terms are given by:

$$\nabla h_i^u(w) = \begin{cases} F_i \mu_i & \text{if } w^T F_i \mu_i > v_i + \epsilon \\ 0 & \text{otherwise.} \end{cases} \quad (8.21)$$

Taking a step in the direction of the expected feature counts of a given policy increases the weights of features that are seen frequently by the policy, thereby increasing the cost-to-go of the policy. Similarly, taking a step in the direction of the negative feature counts decreases the cost-to-go of that policy. Intuitively, since the subgradient algorithm follows the negative subgradient at each iteration, the subgradient contributions from the lower bound terms $h_i^l(w)$ attempt to increase the cost-to-go of the optimal policy μ_i^* if the cost-to-go of that policy is not lower bounded by $v_i - \epsilon$. Similarly, the subgradient contributions from the upper bound terms $h_i^u(w)$ attempt to lower the cost-to-go of the example policy if the cost-to-go of that policy is not currently upper bounded by

$v_i + \epsilon$.

In summary, if the cost of the minimum cost path drops more than ϵ below the desired cost, then the algorithm tries to increase the cost of that path. Alternatively, if the cost of the example path, which upper bounds that of the minimum cost path, raises more than ϵ above the desired cost, the algorithm tries to decrease the cost of that example path. The algorithm attains a zero subgradient contribution from a given MDP only if both the minimum cost path and the cost of the example path are within ϵ of v_i . For this condition to hold, the cost of the two paths must also be within 2ϵ of each other. Thus, as an implicit subgoal, the algorithm also tries to bring the minimum cost path and the example path together in cost.

8.4 Computing functional gradients of MMSR

As was done for maximum margin planning, we can apply the functional exponentiated gradient descent algorithm to a functional form the the maximum margin structured regression (MMSR) objective function given in Equation 8.18. As before, this optimization technique simultaneously generalizes the MMSR framework to learning nonlinear cost functions, while automatically satisfying implicit state-action positivity constraints.

Using the notation defined in Section 6.1, the functional form of the MMSR objective can be written

$$r[w] = \frac{1}{N} \sum_{i=1}^N \left(\max \left\{ \epsilon, v_i - \min_{\mu \in \mathcal{G}_i} \sum_{(s,a) \in \mathcal{M}_i} c(f_i^{sa}) \mu^{sa} \right\} + \max \left\{ \epsilon, \sum_{(s,a) \in \mathcal{M}_i} c(f_i^{sa}) \mu_i^{sa} - v_i \right\} \right) \quad (8.22)$$

$$= \frac{1}{N} \sum_{i=1}^N \left(h_i^l[c] + h_i^u[c] \right), \quad (8.23)$$

where we define

$$h_i^l[c] = \max \left\{ \epsilon, v_i - \min_{\mu \in \mathcal{G}_i} \sum_{(s,a) \in \mathcal{M}_i} c(f_i^{sa}) \mu^{sa} \right\}, \quad (8.24)$$

and

$$h_i^u[c] = \max \left\{ \epsilon, \sum_{(s,a) \in \mathcal{M}_i} c(f_i^{sa}) \mu_i^{sa} - v_i \right\}. \quad (8.25)$$

The functional gradient of this objective is again a linear combination of Dirac delta functions. The contribution from the lower bound terms is given by

$$\nabla_f h_i^l[c] = \begin{cases} -\sum_{(s,a) \in \mathcal{M}_i} \mu_i^{*sa} \delta_{f_i^{sa}} & \text{if } \min_{\mu \in \mathcal{G}_i} \sum_{(s,a) \in \mathcal{M}_i} c(f_i^{sa}) \mu_i^{sa} < v_i - \epsilon \\ 0 & \text{otherwise,} \end{cases} \quad (8.26)$$

where $\mu_i^* = \arg \min_{\mu \in \mathcal{G}_i} \sum_{(s,a) \in \mathcal{M}_i} c(f_i^{sa}) \mu_i^{sa}$. Similarly, the subgradients contributions from the upper bound terms are given by:

$$\nabla h_i^u(w) = \begin{cases} \sum_{(s,a) \in \mathcal{M}_i} \mu_i^{*sa} \delta_{f_i^{sa}} & \text{if } \sum_{(s,a) \in \mathcal{M}_i} c(f_i^{sa}) \mu_i^{sa} > v_i + \epsilon \\ 0 & \text{otherwise.} \end{cases} \quad (8.27)$$

Given these functional gradient computations, the algorithm proceeds as described in Chapter 4.

8.5 An application to value function approximation

We applied the maximum margin structured regression algorithm to the heuristic learning problem described in Section 6.6. Rather than directly running a nonlinear variant, we demonstrated the superiority of this integrated framework by running the linear algorithm using the final set of boosted features learned under MMPBOOST for this problem.

Figure 8.1 shows the performance improvement offered by MMSR. The plot shows the root-mean-square (RMS) error in training prediction of the original two-stage learning approach in blue, which we refer to as *regressed MMP*. The loss per iteration of the regressed MMP algorithm decreases initially, but soon asymptotes. The basic MMSR algorithm we described in this section, shown in red, demonstrates substantially superior performance.

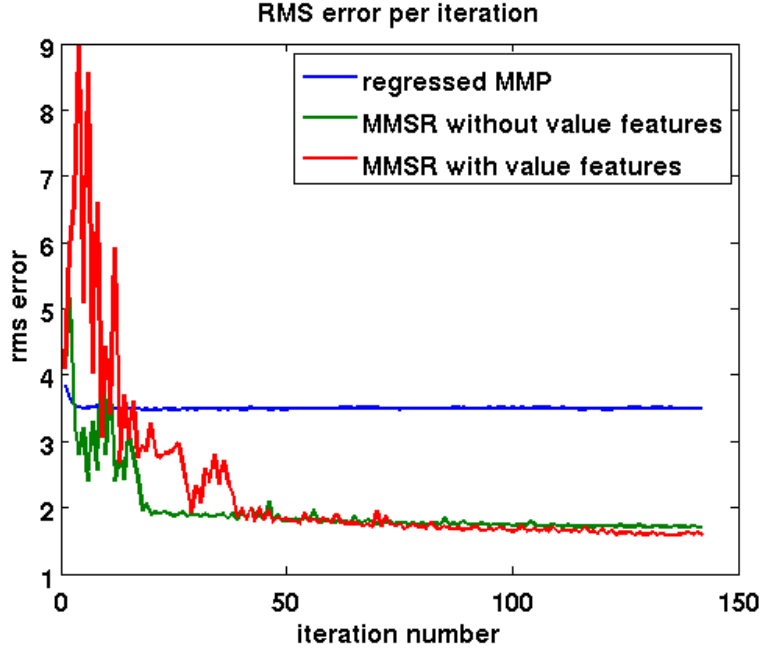


Figure 8.1: MMSR Value function approximation results. See text for details.

An additional benefit to using the MMSR algorithm for value function approximation is the option of utilizing value features in addition to cost features.³ The required augmentation to the objective function is discussed in (Ratliff, Bagnell, & Zinkevich, 2007a), but intuitively, we choose our value function approximator to be the sum of the planned cost-to-go and a linear combination of value features. This augmentation essentially amounts to combining the MMSR risk with an ϵ -insensitive SVM regression risk. The figure shows the RMS error progression of this variant in red. The added dimensions cause slightly slower convergence in the beginning, but it begins to overtake the basic variant around iteration 50.

³Value features include quantities such as the Euclidean distance between start and goal points, or the cost-to-go estimates returned by a fixed set of expert planners. These quantities encode information about the problem which can directly aid in approximating the desired value.

To this point in the thesis, we have discussed primarily inverse optimal control algorithm under the framework of MMP. These algorithms generalize well in both theory and practice, but they are limited in a fundamental sense: they require optimal control. Unfortunately, the class of problems in robotics where optimal control is tractable is relatively small. Successful application of optimal control techniques in practice often requires either strong assumptions to simplify the problem, or additional structure in the system to support it.

In many mobile robot systems, controllers must reason about the robot’s dynamics in order to perform well. In these cases, fully modeling the system as an MDP can be difficult because optimal control quickly becomes intractable as the dimension of the system increases and real-time constraints restrict the computation time of the MDP solver. MDP solvers are therefore restricted to two or at most three dimensions in order to perform efficiently. For many systems, the optimal control strategy cannot be used directly to suggest actions for the robot because the MDP does not account for the robot’s dynamics.

Instead, roboticists often design a separate set of local actions that better account for the higher-dimensional nature of the robot’s true state. The MDP is then used to estimate each action’s long-term consequences by simulating the action forward and evaluating the cost-to-go from a lower dimensional representation of the resulting state. Importantly, this action score often

additionally includes contributions encoding dynamical considerations and higher-resolution local features such as image features or LADAR responses extracted from this set of higher dimensional actions. This approach combines two sources of information. The first source directly estimates the value of taking the action, while the second searches forward through a lower-dimensional MDP to compute and estimate for the cost-to-go.

In this chapter, we build on this intuition to extend IOC to higher dimensional problems. We introduce a new model called *inverse optimal heuristic control* (IOHC) with this two part structure to effectively and efficiently model the probability of an action given an observation as a combination of an long-term IOC style cost and a higher-dimensional BC style cost. In this sense, IOHC joins the two previously distinct generalizable imitation learning categories of inverse optimal control and behavioral cloning.

We analyze the training characteristics of this model and demonstrate its state-of-the-art performance on two stochastic imitation learning problems.

9.1 Introduction

We frame the training of our combined model as an optimization problem. Although the resulting objective function is non-convex, we present a collection of convex approximations that may be optimized as surrogates. Further, we demonstrate both empirically and theoretically that the objective function is nearly convex. Optimizing it directly leads to improved performance across a range of imitation learning problems. Section 9.5 begins by illustrating the theoretical properties of our algorithm on a simple problem. We then demonstrate the algorithm and compare its performance to previous approaches on a taxi route prediction problem (Section 9.5.2) and on a pedestrian prediction problem (Section 9.5.3) using real-world data sets.

Prior work has previously examined combining aspects of behavioral cloning and inverse optimal control (under the names *direct* and *indirect* approaches) (Neu & Szepesvari, 2007), however, the authors focus on only the relationships between the loss-functions typically considered under the two approaches. The techniques described in that paper remain limited by the low-dimensionality restrictions of inverse optimal control. Formally, similar to previous work (Neu & Szepesvari, 2007;

Ramachandran & Amir, 2007; Ziebart *et al.*, 2008a), our technique fits a Gibbs/Maximum Entropy model over actions based on features in the environment. In this paper, however, we take a direct approach to training and propose to use our Gibbs-based model to learn a stochastic policy that predicts the probability that the expert takes each action given an observation.

9.2 Inverse optimal heuristic control

In this section, we examine a relationship between behavioral cloning and inverse optimal control that becomes clear through the use of Gibbs distributions. After discussing this relationship in Section 9.2.1, we propose a novel Gibbs model in Section 9.2.2 that combines the strengths of these individual models. Section 9.2.3 presents an efficient gradient-based learning algorithm for fitting this model to training data.

9.2.1 Gibbs models for imitation learning

Recent research in inverse optimal control has introduced a Gibbs model of action selection in which the probability of taking an action is inversely proportional to that action’s exponentiated Q-value in the MDP (Neu & Szepesvari, 2007; Ramachandran & Amir, 2007). Denoting the immediate cost of taking an action a from state s as $c(s, a)$, and the cost-to-go from a state s' as $J(s')$, we can write $Q^*(s, a) = c(s, a) + J(\mathcal{T}_s^a)$, where $\mathcal{T}_s^a$ denotes the deterministic transition function.¹ The Gibbs model is therefore

$$p(a|s) = \frac{e^{-c(s,a)-J(\mathcal{T}_s^a)}}{\sum_{a' \in \mathcal{A}_s} e^{-c(s,a')-J(\mathcal{T}_s^{a'})}}, \quad (9.1)$$

where the function $Q^*(s, a) = c(s, a) + J(\mathcal{T}_s^a)$ is known as the energy function of the Gibbs model.

The form of this inverse optimal control model is strikingly similar to a multi-logistic regression classification model (Nigam, Lafferty, & McCallum, 1999). We arrive at a straight-forward behavioral cloning model based on multi-logistic regression by simply replacing the energy function with a linear combination of features: $E(s, a) = w^T f(s, a)$, where $f(s, a)$ denotes a function mapping

¹In this paper, we restrict ourselves to deterministic MDPs for modeling the lower-dimensional problem.

each state-action pair to a representative vector of features.

9.2.2 Combining inverse optimal control and behavioral cloning

The observations presented in Section 9.2.1 suggest that a natural way to combine these two models is to design an energy function that utilizes the strengths of both paradigms. While inverse optimal control has demonstrated better generalization than behavioral cloning in real-world settings (Ratliff, Bagnell, & Zinkevich, 2006), the technique’s applicability is limited by its reliance on an MDP solver. On the other hand, the range of application for behavioral cloning has historically surpassed the modeling capacity of MDPs.

To prevent needlessly restricting our model, we consider a general class of policies in which the agent simply maps observations to actions $p(a|o) \propto e^{-\tilde{E}(o,a)}$, where $a \in \mathcal{A}$ is an arbitrary action and $o \in \mathcal{O}$ is a given observation.

In many cases, there exists a problem specific MDP $\mathcal{M}$ that can model a subproblem of the decision process. Let $\mathcal{S}_{\mathcal{M}}$ and $\mathcal{A}_{\mathcal{M}}$ be the state and action spaces of our lower-dimensional MDP $\mathcal{M}$. We require a mapping $\phi(o, a)$ from an observation-action pair to a sequence of state-action pairs that represents the behavior exhibited by the action through the lower-dimensional MDP. Specifically, we denote $\phi(o, a) = \{(s_1, a_1), (s_2, a_2), \dots\}$ and use $\mathcal{T}_o^a$ to indicate the state resulting from following the action-trajectory $\phi(o, a)$. Throughout the paper, we denote the cost of a trajectory ξ through this MDP as $C(\xi)$, although for convenience we often abbreviate $C(\phi(o, a))$ as $C(o, a)$.

Our combined Gibbs model, in this setting, uses the following energy function

$$\tilde{E}(o, a) = E(o, a) + Q_{\mathcal{M}}^*(o, a),$$

where $Q_{\mathcal{M}}^*(o, a) = C(o, a) + J_{\mathcal{M}}(\mathcal{T}_o^a)$ denotes the cumulative cost-to-go of taking the short action-trajectory $\phi(o, a)$ and then following the minimum cost path from the resulting state to the goal. Intuitively, the learning procedure chooses between BC and IOC paradigms under this model, or finds a combination of the two that better represents the data.

In what follows, we denote a trajectory as a sequence of observation-action pairs $\xi = \{(o_t, a_t)\}_{t=1}^{T_\xi}$, the set of all such trajectories starting from a given observation o as Ξ_o , and the set of actions that

can be taken given a particular observation o as $\mathcal{A}_o$. Sections 9.3 and 9.4 present some theoretical results for the Gibbs IOC model. In those sections, we refer to only a single MDP, and we can talk about states in place of observations. When applicable, we replace the observation o with a state s in this notation.

Choosing linear parameterizations of both terms of the energy function we can write our model as

$$p(a|s) = \frac{e^{-w_v^T f_v(o,a) - w_c^T F_{\mathcal{M}}^*(o,a)}}{\sum_{a' \in \mathcal{A}} e^{-w_v^T f_v(o,a') - w_c^T F_{\mathcal{M}}^*(o,a')}}, \quad (9.2)$$

where we define $F_{\mathcal{M}}(\xi) = \sum_t f_c(s_t, a_t)$ as the sum of the feature vectors encountered along trajectory ξ , and denote

$$F_{\mathcal{M}}^*(o, a) = F_{\mathcal{M}}(\phi(o, a)) + \sum_{(s_t, a_t) \in \xi^*} f_c(s_t, a_t),$$

where ξ^* is the optimal trajectory starting from $\mathcal{T}_o^a$ and $f_c(s, a)$ denotes a feature vector associated with state-action pair (s, a) . Below we use w to denote the combined set of parameters (w_v, w_c) . This notation utilizes the common observation that for linear costs, the cumulative cost-to-go of a trajectory through an MDP is a linear combination of the cumulative feature vector (Ng & Russell, 2000). Choosing $w_v = 0$ results in a generalized Gibbs model for IOC in which each action may be a small trajectory through the MDP.

We often call $f_v(s, a)$ and $f_c(s, a)$ *value features* and *cost features* respectively because of their traditional uses for value function approximation and cost parameterization in the separate behavioral cloning and inverse optimal control models.

9.2.3 Gradient-based optimization

Following the tradition of multi-logistic regression for behavioral cloning, given a trajectory $\xi = \{(o_t, a_t)\}_{t=1}^T$, we treat each observation-action pair as an independent training example and optimize the negative log-likelihood of the data. Given a set of trajectories $\mathcal{D} = \{\xi_i\}_{i=1}^N$, the exponential

form of our Gibbs distribution allows us to write our objective $l(\mathcal{D}; w_v, w_c) = -\log \prod_{i=1}^N p(\xi_i)$ as

$$\begin{aligned} l(\mathcal{D}; w_v, w_c) &= \sum_{i=1}^N \sum_{t=1}^{T_i-1} w_v^T f_v(o_t, a_t) + w_c^T F_{\mathcal{M}}^*(o_t, a_t) \\ &\quad + \log \sum_{a \in \mathcal{A}} e^{-w_v^T f_v(o_t, a) - w_c^T F_{\mathcal{M}}^*(o_t, a)} + \frac{\lambda}{2} \|w\|^2, \end{aligned} \quad (9.3)$$

where we denote $T_i = T_{\xi_i}$ for convenience; $\lambda \geq 0$ is a regularization constant. In our gradient expressions and discussion below, we suppress the regularization term for notational convenience.

This objective function is piecewise-differentiable; at points of differentiability the following formula gives its gradient in terms of both w_v and w_c :

$$\begin{aligned} \nabla_{w_v} l(\mathcal{D}) &= \sum_{i=1}^N \sum_{t=1}^{T_i-1} f_v(o_t, a_t) - E_{p_w(a|o_t)}[f_v(o_t, a)] \\ \nabla_{w_c} l(\mathcal{D}) &= \sum_{i=1}^N \sum_{t=1}^{T_i-1} F_{\mathcal{M}}^*(o_t, a_t) - E_{p_w(a|o_t)}[F_{\mathcal{M}}^*(o_t, a)], \end{aligned}$$

where we use $p_w(a|o) \propto \exp\{-w_v^T f_v(o, a) - w_c^T F_{\mathcal{M}}^*(s, a)\}$ to denote policy under our Gibbs model parameterized by the combined vector of parameters $w = (w_v; w_c)$. At points of nondifferentiability, there are multiple optimal paths through the MDP; choosing any one of them in the above formula results in a valid subgradient. Algorithm 15 presents a simple optimization routine based on exponentiated gradient descent for optimizing this objective.

9.3 On the efficient optimization of inverse optimal heuristic control

Our experiments indicate that the simple gradient-based procedure of Section 9.2.3 is robust to variations in starting point, a property often reserved for convex functions. This section demonstrates that, in many ways, our objective closely resembles a convex function. We first note that for any fixed w_c the objective as a function of w_v is convex. Moreover, we demonstrate below that for $w_v = 0$, the resulting (generalized) IOC Gibbs model is almost-convex in a rigorous sense. In

Algorithm 15 Optimization of the negative log-likelihood via exponentiated gradient descent

```

1: procedure OPTIMIZE(  $\mathcal{D} = \{(\xi_i, \mathcal{M}_i)\}_{i=1}^N$  )
2:   Initialize log-parameters  $w_v^l \leftarrow 0$  and  $w_c^l \leftarrow 0$ 
3:   for  $k = 0, \dots, K$  do
4:     Initialize  $g_v = 0$  and  $g_c = 0$ 
5:     for  $i = 1, \dots, N$  do
6:       Set  $w_v = e^{w_v^l}$  and  $w_c = e^{w_c^l}$ 
7:       Construct cost map  $c_i^{sa} = w^T f_c(s, a)$  for our low-dimensional MDP  $\mathcal{M}_i$ 
8:       Compute cumulative feature vectors  $F_{\mathcal{M}}^*(o, a)$  through MDP  $\mathcal{M}_i$  for each potential
          action from each observation found along the trajectory
9:       for  $t = 1, \dots, T_i$  do
10:         $g_v^t \leftarrow f_v(o_t, a_t) - E_{p_w(a|o_t)}[f_v(o_t, a)]$ 
11:         $g_c^t \leftarrow F_{\mathcal{M}_i}(o_t, a_t) - E_{p_w(a|o_t)}[F_{\mathcal{M}_i}^*(o_t, a)]$ 
12:      end for
13:    end for
14:    Update  $w_v^l \leftarrow w_v^l - \alpha_k \sum_{t=1}^{T_i} g_v^t$  and
           $w_c^l \leftarrow w_c^l - \alpha_k \sum_{t=1}^{T_i} g_c^t$ 
15:  end for
16:  return Final values  $w_v = e^{w_v^l}$  and  $w_c = e^{w_c^l}$ 
17: end procedure

```

combination, these results suggest that an effective strategy for optimizing our model is to:

1. Set $w_v = 0$ and optimize the almost-convex IOC Gibbs model.
2. Fix w_c and optimize the resulting convex problem in w_v .
3. Further optimize w_c and w_v jointly.

Optionally, in (2), one may use the fixed Q-values as features thereby guaranteeing that the resulting model can only improve over the Gibbs model found in step (1). The final joint optimization phase can then only additionally improve over the model found in (2).

In what follows, we derive our almost-convexity results strictly in terms of the Gibbs model for IOC. Similar arguments can be used to prove analogous results for our generalized IOC Gibbs model as well.

Definition 9.3.1: Almost-convexity. *A function $f(x)$ is almost-convex if there exists a constant $c \in \mathbb{R}$ and a convex function $h(x)$ such that $h(x) \leq f(x) \leq h(x) + c$ for all x in the*

domain.

The notion of almost-convexity formalizes the intuition that the objective function may exhibit the general shape of a convex function, while not necessarily being precisely convex. In what we show below, the negative log-likelihood of the Gibbs model for IOC is largely dominated by a commonly seen function in the machine learning literature known as the perceptron objective.² The nonconvexities of the negative log-likelihood arise from a collection of bounded discrepancy terms that measure the difference between the hard-min and the soft-min functions.

The negative log-likelihood of an example trajectory ξ , under this model is

$$\begin{aligned} -\log p(\xi) &= -\sum_{t=1}^{T-1} \log \frac{e^{-Q^*(s_t, a_t)}}{\sum_{a \in \mathcal{A}_{s_t}} e^{-Q^*(s_t, a)}} \\ &= \sum_{t=1}^{T-1} \left(Q^*(s_t, a_t) + \log \sum_{a \in \mathcal{A}_{s_t}} e^{-Q^*(s_t, a)} \right). \end{aligned} \quad (9.4)$$

By expanding $Q^*(s, a) = c(s, a) + J(\mathcal{T}_s^a)$, and then pushing the sum through, we can write this as

$$-\log p(\xi) = C(\xi) + \sum_{t=1}^{T-1} \left(J(\mathcal{T}_{s_t}^{a_t}) + \log \sum_{a \in \mathcal{A}_{s_t}} e^{-J(s_t, a)} \right),$$

where we denote the cumulative cost of a path as $C(\xi) = \sum_{t=1}^{T-1} c(s_t, a_t)$.

By noting that $\mathcal{T}_{s_t}^{a_t} = s_{t+1}$ (taking action a_t from state s_t gets us to the next state along the trajectory in a deterministic MDP), we rewrite the second term as

$$\sum_{t=1}^{T-1} J(\mathcal{T}_{s_t}^{a_t}) = \sum_{t=2}^T J(s_t) = -J(s_1) + \sum_{t=1}^{T-1} J(s_t).$$

For the final simplification, we added and subtracted $J(s_1) = \min_{\xi \in \Xi_{s_1}} C(\xi)$ and use the fact that the cumulative cost-to-go of the goal state s_T is zero (i.e., $J(s_T) = 0$). We additionally note that

²We call this function the perceptron objective because the perceptron algorithm that has been used in the past for various structured prediction problems (Collins & Roark, 2004) is a particular subgradient algorithm for optimizing the function. We note, however, that technically, as an objective, this function is degenerate in the sense that it is successfully optimized by the zero function. Many of the properties of the perceptron algorithm cited in the literature are specific to that particular algorithm, and not general properties of this objective.

$J(s) = \min_{a \in \mathcal{A}_s} c(s, a) + J(\mathcal{T}_s^a)$. Our negative log-likelihood expression therefore simplifies to

$$\begin{aligned} -\log p(\xi) &= C(\xi) - \min_{\xi \in \Xi_{s_1}} C(\xi) \\ &\quad + \sum_{t=1}^{T-1} \left(\min_{a \in \mathcal{A}_{s_t}} Q^*(s_t, a) + \log \sum_{a \in \mathcal{A}_{s_t}} e^{-Q^*(s_t, a)} \right). \end{aligned}$$

Finally, if we denote the soft-min function by $\min_{a \in \mathcal{A}_s}^s Q^*(s, a) = -\log \sum_{a \in \mathcal{A}_s} e^{-Q^*(s, a)}$, we can write the negative log-likelihood of a trajectory as

$$\begin{aligned} -\log p(\xi) &= C(\xi) - \min_{\xi \in \Xi_{s_1}} C(\xi) \\ &\quad + \sum_{t=1}^{T-1} \min_{a \in \mathcal{A}_{s_t}}^{\Delta} Q^*(s_t, a). \end{aligned} \tag{9.5}$$

where we use the notation $\min_i^{\Delta} c_i = \min_i c_i - \min_i^s c_i$ to denote the discrepancy between the hard- and soft-min operators over a set of values $\{c_i\}$.

When the cost function is defined as a linear combination of features, the cost function itself is linear, and the Q^* function, as a min over linear functions, is therefore concave (making $-Q^*$ convex). Thus, the first two terms are both convex. In particular, they form the convex structured perceptron objective. Intuitively, these terms contrast the cumulative cost-to-go of the given trajectory with the minimum hypothesized cost-to-go over all trajectories.

The non-convexity of the negative log-likelihood objective arises from the final set of terms in Equation 9.5. Each of these terms simply denotes the difference between the soft-min function and the hard-min function. We can bound the absolute difference between the hard- and soft-min by $\log n$, where n is the number of elements over which the min operates. In our case, this means that each hard/soft-min discrepancy term can contribute no more to the objective than the constant value $\log |\mathcal{A}|$. In particular, we can state the following

Theorem 9.3.2: Gibbs IOC is almost-convex. *Denote the negative log-likelihood of ξ by $f(w)$ and let $h(w) = C(\xi) - \min_{\xi \in \Xi_{s_1}} C(\xi)$. If $n \geq |\mathcal{A}_s|$ for all s and $T = |\xi|$ is the length of the trajectory, then $h(w) \leq f(w) \leq h(w) + |\xi| \log n$.*

Proof. The soft-min is everywhere strictly less than the hard-min. Each discrepancy term $\min_{a \in \mathcal{A}_{s_t}}^\Delta Q^*(s_t, a)$ is therefore positive. Thus, $h(w) \leq f(w)$. Additionally, we know that $\min_i c_i - \log n \leq -\log \sum_i e^{-c_i}$ for any collection $\{c_i\}_{i=1}^n$. For a discrepancy term, this bound gives

$$\begin{aligned} \min_i^\Delta c_i &= \min_i c_i + \log \sum_i e^{-c_i} \\ &\leq \min_i c_i - (\min_i c_i - \log n) = \log n. \end{aligned}$$

Applying this upper bound to our objective gives the desired result. $\square$

Section 9.5.1 presents some simple experiments illustrating these results.

9.4 Convex approximations

The previous section demonstrates that the generalized Gibbs model for inverse optimal control is almost-convex and suggests that directly optimizing the objective following algorithm 15 will work well on a wide range of problems. While our experimental results support this analysis, without a proof of (approximate) global convergence, we cannot guarantee these observations to hold across all problems. This section, therefore, presents three convex approximations to the negative log-likelihood that can be efficiently optimized to attain a good starting point for algorithm 9.2.3 should arbitrary initialization fail.

We present the first two results for the traditional IOC Gibbs model, although analogous results hold for our generalized Gibbs model as well. However, to emphasize its application to our full combined model, the discussion in Section 9.4.3 of what we call the soft-backup approximation is presented in full generality. In particular, steps (1) and (2) of the optimization strategy proposed in Section 9.3 may be replaced by optimizing this convex approximation.

9.4.1 The perceptron algorithm

Given the discussion of almost-convexity in Section 9.3, the simplest convex approximation is the perceptron objective

$$h(w) = \sum_{i=1}^N C(\xi_i) - \min_{\xi \in \Xi_i} C(\xi). \quad (9.6)$$

See (Collins & Roark, 2004) for details regarding the perceptron algorithm. Since each discrepancy term is positive, this approximation is a convex lower bound.

9.4.2 Expert augmentation

Our second convex approximation can be derived by augmenting how we compute the action probabilities of the Gibbs model. Equation 9.1 dictates that the probability of taking action a from state s_t is inversely proportional to $E(s_t, a) = Q^*(s_t, a)$ through our MDP. We will modify this energy function only for the value at the action a_t chosen by the expert from that state. Specifically, we will prescribe $E(s_t, a_t) = C(\xi_t) - \sum_{\tau=t}^T c(s_\tau, a_\tau)$; the energy of the expert’s action now becomes the cumulative cost-to-go of the trajectory taken by the expert.

The negative log-likelihood of the resulting policy is convex. Moreover, for any parameter setting w , since $Q^*(s_t, a_t) \leq Q(\xi_t)$ and the energies of each alternative action remain unchanged, the probability of taking the expert’s action cannot be smaller under the actual policy than it was under this modified policy. This observation shows that our modified negative log-likelihood forms a convex upper bound to the desired negative log-likelihood objective.

9.4.3 Soft-backup modification

Arguably, the most accurate convex approximation we have developed comes from deriving a simple soft-backup dynamic programming algorithm to modify the energy values for the expert’s action used by the Gibbs policy. Applying this procedure backward along the example trajectory $\xi = \{(o_t, a_t)\}_{t=1}^T$ starting from the o_T and proceeding toward o_1 modifies the policy model in such a way that the contrasting hard/soft-min terms of Equation 9.5 cancel.

Algorithm 16 Soft-backup procedure for convex approximation

```

1: procedure SOFTBACKUP(  $\xi, c : \mathcal{S} \rightarrow \mathcal{A}$  )
2:   Compute cost-to-go values  $J_{\mathcal{M}}(s)$  for  $s$  in  $\mathcal{M}$  that can be reached by at most one action
   from any  $o_t \in \xi$ .
3:   Initialize  $\tilde{J}_{\mathcal{M}}(s_T) = 0$ 
4:   for  $t = T - 1, \dots, 1$  do
5:     Set  $\alpha_t = \sum_{a \in \mathcal{A} \setminus a_t} e^{-E(o_t, a) - C(o_t, a) - J_{\mathcal{M}}(\mathcal{T}_{s_t}^a)}$ 
6:     Set  $\tilde{\alpha}_t = e^{-E(o_t, a_t) - C(o_t, a_t) - \tilde{J}_{\mathcal{M}}(o_{t+1})}$ 
7:     Update  $\tilde{J}_{\mathcal{M}}(o_t) = -\log(\alpha_t + \tilde{\alpha}_t)$ 
8:   end for
9:   return Updated  $J$ -values.
10: end procedure

```

Specifically, as detailed in Algorithm 16 the soft-backup algorithm proceeds recursively from observation o_T replacing each hard-min J-value $J_{\mathcal{M}}^*(o_t) = \min_{a \in \mathcal{A}} C(o_t, a) + J_{\mathcal{M}}(\mathcal{T}_{o_t}^a)$ with the associated soft-min. We define

$$\tilde{J}_{\mathcal{M}}(o_t) = -\log \sum_{a \in \mathcal{A}} e^{-C(o_t, a) - \tilde{J}_{\mathcal{M}}(\mathcal{T}_{o_t}^a)}, \quad (9.7)$$

with $\tilde{J}(\mathcal{T}_{o_t}^a) = J(\mathcal{T}_{o_t}^a)$ when $a \neq a_t$. Our new policy along the example trajectory, therefore, becomes

$$\tilde{p}(a|o_t) = \frac{e^{-E(o_t, a) - C(o_t, a) - \tilde{J}_{\mathcal{M}}(\mathcal{T}_{o_t}^a)}}{\sum_{a' \in \mathcal{A}_s} e^{-E(o_t, a') - C(o_t, a') - \tilde{J}_{\mathcal{M}}(\mathcal{T}_{o_t}^{a'})}} \quad (9.8)$$

Theorem 9.4.1: *The negative log-likelihood of the modified policy of Equation 9.8 is convex and takes the form*

$$\tilde{l}(w; \xi) = \sum_{(o_t, a_t) \in \xi} (E(o_t, a_t) + C(o_t, a_t)) - \tilde{J}_{\mathcal{M}}(o_1).$$

Proof (sketch). The true objective function is given in Equation 9.3. Under the modified policy model, each energy value of the Gibbs model becomes $E(o, a) + C(o, a) + \tilde{J}_{\mathcal{M}}(\mathcal{T}_o^a)$. In particular, for each t along the trajectory, the modified J-value in that expression cancels with the log-partition term corresponding to observation-action pair (o_{t+1}, a_{t+1}) . Therefore, summing across all time steps leaves only the energy and cost segment terms $\sum_t E(o_t, a_t) + C(o_t, a_t)$ and the first observation's modified J-value $\tilde{J}(o_1)$. This

argument demonstrates the form of the approximation. Additionally, since each backup operation is a soft-min performed over concave functions, each term $\tilde{J}(o_t)$ is also concave. The term $-\tilde{J}(o_1)$ is therefore convex, which proves the convexity of our approximation. $\square$

Setting the value features to zero $w_v = 0$, we can additionally show that this final approximation $\tilde{l}(w)$ is also bounded by the perceptron $h(w)$ in the same way as our negative log-likelihood objective $l(w)$. Moreover, $l(w) \leq \tilde{l}(w)$ everywhere. Combining these bounds, we can state that for a given example trajectory ξ and for all w ,

$$h(w) \leq l(w) \leq \tilde{l}(w) \leq h(w) + |\xi| \log |A|. \quad (9.9)$$

This observation suggests that the soft-backup convex approximation is potentially the tightest of the approximations presented here. Our experiments in Section 9.5.1 support this claim.

9.5 Experimental results

In this section, we first demonstrate our algorithm on a simple two-dimensional navigational problem to illustrate the overall convex behavior of optimizing the negative log-likelihood, and compare that to the performance of the convex approximations discussed in section 9.4. We then present two real-world experiments, and compare the performance of the combined model (with value features) to that of the Gibbs model alone (without value features).

9.5.1 An illustrative example

In this experiment, we implemented Algorithm 15 on the simple navigational problem depicted in the leftmost panel of Figure 9.1 and compared its performance to that of each convex approximation presented in Section 9.4 using only the traditional IOC Gibbs model. We manually generated 10 training examples chosen specifically to demonstrate stochasticity in the behavior. Our feature set consisted of 14 randomly positioned two-dimensional radial basis features along with a constant feature. We set our regularization parameter to zero for this problem.

As we predicted in Section 9.4, the backup approximation performs the best on this problem.

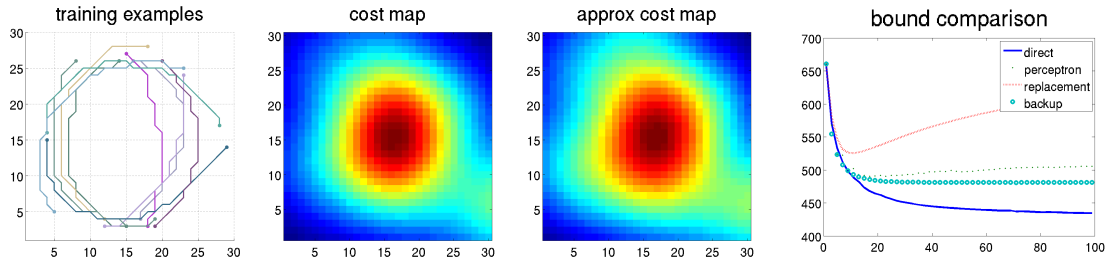


Figure 9.1: This figure shows the examples (left), the cost map learned by directly optimizing the negative log-likelihood (middle-left), the cost map learned by optimizing the soft-backup approximation (middle-right), and a plot comparing the performance of each convex approximation in terms of its ability to optimize the negative log-likelihood (right). The training examples were chosen specifically to exhibit stochasticity. See Section 9.5.1 for details.

Although it converges to a suboptimal solution, the negative log-likelihood levels off and does not significantly increase from the minimum attained value in contrast to the behavior seen in the perceptron and replacement approximations. The center two panels show the cost functions learned for this problem using Algorithm 15 (center-left) and by optimizing the soft-backup approximation (center-right).

9.5.2 Turn prediction for taxi drivers

We now apply our approach to modeling the route planning decisions of drivers so that we can predict, for instance, whether a driver will make a turn at the next stoplight. The imitation learning approach to this problem Ziebart *et al.* (2008a), which learns a cost function based on road network features, has been shown to outperform direct action modeling approaches Ziebart *et al.* (2008b), which estimate the action probabilities according to previous observation proportions Simmons *et al.* (2006); Krumm (2008). However, computational efficiency in the imitation learning approach comes at a cost: the state space of the corresponding Markov Decision Process must be kept small. Table 9.1 shows how the number of states in the Markov decision process grows when paths of length K decisions are represented by the state. Previously, the model was restricted to using only the driver’s last decision as the state of the Markov Decision Process to provide efficient inference.

As a consequence, given the driver’s intended destination, decisions within the imitation learning model are assumed to be independent of the previous driving decisions that led the driver to his

Table 9.1: State space size for larger previous segment history.

History (K)	States
1	315704
2	901046
3	2496122
4	7733620
5	23281701
$\vdots$	$\vdots$
k	$\approx 3^k * 10^5$

current intersection. We incorporate the following action value features that are functions of the driver’s previous decisions to relax this independence assumption:

- Has the driver previously driven on this road segment on this trip?
- Does this road segment lead to an intersection previously encountered on this trip?

We compare our approach with a combination of these action value features and additional cost-to-go features that are road network characteristics (e.g., length, speed limit, road category, number of lanes) with a model based on only cost-to-go features. Additionally, the *Full* model contains additional unique costs for every road segment in the network. We use the problem of predicting a taxi driver’s next turn given final destination on a withheld dataset (over 55,000 turns, 20% of the complete dataset) to evaluate the benefits of our approach. We include baselines from previously applied approaches to this problem Ziebart *et al.* (2008b) and compare against the Gibbs model without value features, and our new inverse optimal heuristic control (IOHC) approach, which includes value features.

Table 9.2 shows the accuracy of each model’s most likely prediction and the average log-probability of the driver’s decisions within each model. We note for both sets of cost-to-go features a roughly 18% reduction in the turn prediction error over the Gibbs models when incorporating action value features (IOHC), which is statistically significant ($p < 0.01$). We also find corresponding improvements in our log-probability metric. We additionally note better improvement in both metrics over the best previously applied approaches for this task.

Table 9.2: Turn prediction evaluation for various models.

Model	Accuracy	Log Probability
Random Guess	46.4%	-0.781
Markov Model	86.2%	-0.319
MaxEnt IOC (Full)	91.0%	-0.240
Gibbs (Basic)	88.8%	-0.319
IOHC (Basic)	90.8%	-0.246
Gibbs (Full)	89.9%	-0.294
IOHC (Full)	91.9%	-0.226



Figure 9.2: The three images shown here depict the office setting in which the pedestrian tracking data was collected.

9.5.3 Pedestrian prediction

Predicting pedestrian motion is important for many applications, including robotics, home automation, and driver warning systems, in order to safely interact in potentially crowded real-world environments. Under the assumption that people move purposefully, attempting to achieve some goal, we can model a person’s movements using an MDP and train a motion model using imitation learning.

In this experiment, we demonstrate that trajectories sampled from a distribution trained using momentum-based value features better match human trajectories than trajectories sampled from a model without value features. The resulting distribution over states is therefore a superior estimate of future behavior of the person being tracked.

Tracks of pedestrians were collected in an office environment using a laser-based tracker (see Figure 9.2). The outline of the room and the objects in the room were also recorded. The laser map was discretized into 15cm by 15cm cells and convolved with a collection of simple Gaussian smoothing filters. These filtered values and one feature representing the presence of an object make

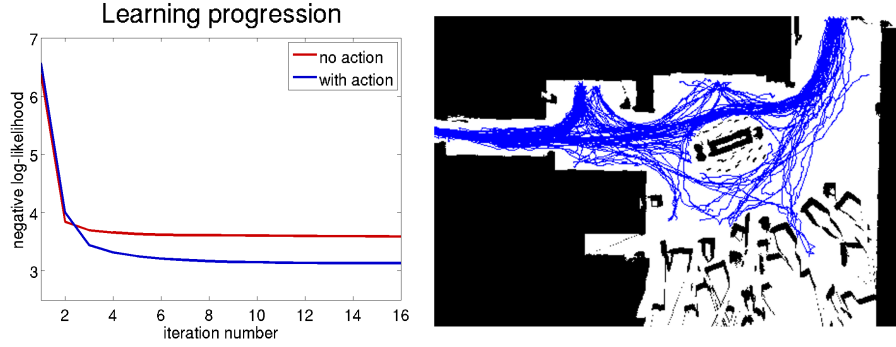


Figure 9.3: This figure compares the negative log-likelihood progressions between a traditional Gibbs model (without value features) and an IOHC model (with value features) on a validation set. Access to features encoding dynamic characteristics of the actions substantially improves modeling performance. The full collection of pedestrian trajectories is shown on the right from an overhead perspective.

up the state-based feature set. Additionally, we include a set of action value features consisting of a history of angles between the current and previous displacements. These action features incorporate a smoothness objective that would otherwise require a higher-dimensional state space to incorporate.

We constructed a Gibbs model (without value features) and an IOHC model (with value features) using a set of 20 trajectories and we tested the model on a set of 20 distinct validation trajectories. Figure 9.3 compares the negative log-likelihood progression of both models on the test set during learning. The algorithm is able to exploit features that encode dynamical aspects of each action to find a superior model of pedestrian motion.

9.6 Conclusions

We have presented an imitation learning model that combines the efficiency and generality of behavioral cloning strategies with the long-horizon prediction performance of inverse optimal control. Our experiments have demonstrated empirically the benefits of this approach on real-world problems. In future work, we plan to explore applications of the pedestrian prediction model to developing effective robot-pedestrian interaction behaviors. In particular, since stochastic sampling may be implemented using efficient Monte Carlo techniques, the computational complexity of predictions can be controlled to satisfy real-time constraints.

Chapter 10

Covariant Hamiltonian Optimization for Motion Planning

In the previous chapter, we studied how to effectively apply IOC techniques while accounting for detailed dynamics of the vehicle. Our model combined aspects of behavioral cloning with the long-range reasoning capabilities of the IOC framework. In this chapter, we study a second class of techniques for solving high-dimensional imitation learning problems. In this case, we study the problem of high-dimensional manipulation where planning is intractable and common solutions often require variations on approximate probabilistic planning algorithms.

Instead of augmenting our model with additional structured, as we did in the previous chapter under IOHC, in this chapter we modify the planner itself. We demonstrate that efficient obstacle representations can provide important gradient information from the environment, enabling the development of full motion planning techniques designed around covariant forms of trajectory optimization. This work reduces high-dimensional motion planning to optimization, opening this class of high-dimensional planning to the IOC techniques developed in this thesis. Learning, in this context, again becomes an iterative procedure designed to mold the cost function to make the expert look optimal.

10.1 Introduction

In recent years, sampling-based planning algorithms have met with widespread success due to their ability to rapidly discover the connectivity of high-dimensional configuration spaces. Planners such as Probabilistic Road Map (PRM) and Rapidly-exploring Random Tree (RRT) algorithms, along with their descendents, are now used in a multitude of robotic applications (Kavraki *et al.*, 1996; Kuffner & LaValle, 2000). Both algorithms are typically deployed as part of a two-phase process: first find a feasible path, and then optimize it to remove redundant or jerky motion.

Perhaps the most prevalent method of path optimization is the so-called “shortcut” heuristic, which picks pairs of configurations along the path and invokes a local planner to attempt to replace the intervening sub-path with a shorter one (Kavraki & Latombe, 1998; Chen & Hwang, 1998). “Partial shortcuts” as well as medial axis retraction have also proven effective (Geraerts & Overmars, 2006). Another approach used in elastic bands or elastic strips planning involves modeling paths as mass-spring systems: a path is assigned an internal energy related to its length or smoothness, along with an external energy generated by obstacles or task-based potentials. Gradient based methods are used to find a minimum-energy path (Quinlan & Khatib, 1993; Brock & Khatib, 2002).

In this chapter, we present *covariant Hamiltonian optimization for motion planning* (CHOMP), a novel method for generating and optimizing trajectories for robotic systems. The approach shares much in common with elastic bands planning; however, unlike many previous path optimization techniques, we drop the requirement that the input path be collision free. As a result, CHOMP can often transform a naïve initial guess into a trajectory suitable for execution on a robotic platform without invoking a separate motion planner. A covariant gradient update rule ensures that CHOMP converges rapidly to a locally optimal trajectory.

In many respects, CHOMP is related to optimal control of robotic systems. Instead of merely finding feasible paths, our goal is to directly construct trajectories which optimize over a variety of dynamic and task-based criteria. Few current approaches to these forms of optimal control are equipped to handle obstacle avoidance, though. Of those that do, many approaches require some description of configuration space obstacles, which can be prohibitive to create for high-dimensional manipulators (Shiller & Dubowsky, 1991). Many optimal controllers which do handle



Figure 10.1: Experimental robotic platforms: Boston Dynamics’s LittleDog (left), and Barrett Technology’s WAM arm (right).

obstacles are framed in terms of mixed integer programming, which is known to be an NP-hard problem (Schouwenaars *et al.*, 2001; Earl *et al.*, 2005; Ma *et al.*, 2006; Vitus *et al.*, 2008). Approximately optimal algorithms exist, but so far, they only consider very simple obstacle representations (Sundar *et al.*, 1997).

In the rest of this chapter, we give a detailed derivation of the CHOMP algorithm, show experimental results on a 6-DOF robot arm, and outline future directions of work. CHOMP has additionally been successfully applied as a core component of a quadrupedal locomotion system (see (Ratliff *et al.*, 2009b) for details).

10.2 The CHOMP Algorithm

In this section, we present CHOMP, a new trajectory optimization procedure based on covariant gradient descent. An important theme throughout this exposition is the proper use of geometrical relations, particularly as they apply to inner products. This is an important idea in differential geometry (do Carmo, 1976). Our technique utilizes more natural notions of geometry in three ways. First, we measure the size of a trajectory perturbation in terms of how much it affects the trajectory’s dynamics (such as total velocity or total acceleration). This measure is independent of the particular parameterization chosen to represent the trajectory. Second, measurements of obstacle costs should be taken in the workspace so as to correctly account for the geometrical

relationship between the robot and the surrounding environment. And finally, the same geometrical considerations used to update a trajectory should be used when correcting any joint limit violations that may occur. Sections 10.2.1, 10.2.5, and 10.2.7 detail each of these points in turn.

10.2.1 Covariant gradient descent

Formally, our goal is to find a smooth, collision-free, trajectory through the configuration space $\mathbb{R}^m$ between two prespecified end points $q_{\text{init}}, q_{\text{goal}} \in \mathbb{R}^m$. In practice, we discretize our trajectory into a set of n way-points $q_1, \dots, q_n$ (excluding the end points) and measure dynamics using finite differencing. We focus presently on finite-dimensional optimization, although we will return to the continuous trajectory setting in Section 10.2.5. Section 10.2.6, discusses the relationship between these settings.

We model the cost of a trajectory using two terms: an obstacle term f_{obs} , which measures the cost of being near obstacles; and a prior term f_{prior} , which measures dynamics across the trajectory. We generally assume that f_{prior} is independent of the environment. Our objective can, therefore, be written

$$\mathcal{U}(\xi) = f_{\text{prior}}(\xi) + f_{\text{obs}}(\xi).$$

More precisely, the prior term is a sum of squared derivatives. Given suitable finite differencing matrices K_d for $d = 1, \dots, D$, we can represent f_{prior} as a sum of terms

$$f_{\text{prior}}(\xi) = \frac{1}{2} \sum_{d=1}^D w_d \|K_d \xi + e_d\|^2, \quad (10.1)$$

where e_d are constant vectors that encapsulate the contributions from the fixed end points. For instance, the first term ($d = 1$) represents the total squared velocity along the trajectory. In this

case, we can write K_1 and e_1 as

$$K_1 = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ -1 & 1 & 0 & \dots & 0 & 0 \\ 0 & -1 & 1 & \dots & 0 & 0 \\ & \vdots & & \ddots & & \vdots \\ 0 & 0 & 0 & \dots & -1 & 1 \\ 0 & 0 & 0 & \dots & 0 & -1 \end{bmatrix} \otimes I_{m \times m} \quad \text{and} \quad e_1 = \begin{bmatrix} -q_0 \\ 0 \\ \vdots \\ 0, q_{n+1} \end{bmatrix}, \quad (10.2)$$

where $\otimes$ denotes the Kronecker (tensor) product. We note that f_{prior} has a simple quadratic form:

$$f_{prior}(\xi) = \frac{1}{2} \xi^T A \xi + \xi^T b + c$$

for suitable matrix, vector, and scalar constants A, b, c . When constructed as defined above, A will always be symmetric positive definite for all d .

Our technique aims to improve the trajectory at each iteration by minimizing a local approximation of the function that suggests only smooth perturbations to the trajectory, where Equation 10.1 defines our measure of smoothness. At iteration k , within a region of our current hypothesis ξ_k , we can approximate our objective using a first-order Taylor expansion:

$$\mathcal{U}(\xi) \approx \mathcal{U}(\xi_k) + g_k^T (\xi - \xi_k), \quad (10.3)$$

where $g_k = \nabla \mathcal{U}(\xi_k)$. Using this expansion, our update can be written formally as

$$\xi_{k+1} = \arg \min_{\xi} \left\{ \mathcal{U}(\xi_k) + g_k^T (\xi - \xi_k) + \frac{\lambda}{2} \|\xi - \xi_k\|_M^2 \right\}, \quad (10.4)$$

where the notation $\|\delta\|_M^2 = \delta^T M \delta$ denotes the norm of the displacement $\delta = \xi - \xi_k$ taken with respect to the Riemannian metric M . Setting the gradient of the right hand side of Equation 10.4

to zero and solving for the minimizer results in the following more succinct update rule:

$$\xi_{k+1} = \xi_k - \frac{1}{\lambda} M^{-1} g_k$$

It is well known in optimization theory that solving a regularized problem of the form given in Equation 10.4 is equivalent to minimizing the linear approximation in Equation 10.3 within a ball around ξ_k whose radius is related to the regularization constant λ (Boyd & Vandenberghe, 2004). In our case, under the metric A the norm of non-smooth trajectories is large; such trajectories are not likely to be contained within the ball. Our update rule, therefore, ensures that the trajectory remains smooth after each trajectory update.

Two components dominate the computational complexity of evaluating the covariant gradient at each iteration. The first component is the evaluation of the objective function and the computation of its Euclidean gradient. These steps include the evaluation of the obstacle potential which we show in Sections 10.2.4 and 10.2.5 can be implemented in time linear in the number of trajectory way points, as well as the evaluation of the prior potential which again can be implemented in linear time since it requires only finite-differencing operations. The second component, however, is the transformation of this Euclidean gradient by the inverse metric A^{-1} . Naïvely, this operation requires a one-time matrix inverse preprocessing step with scales as $O(n^3)$, where n is the number of way points, and a matrix multiplication at each iteration which scales as $O(n^2)$. However, we can exploit the band-diagonal structure typically found in the metric to reduce the computation time to linear in n using back-substitution techniques to solve the linear system at each iteration without precomputing the matrix inverse (Wainwright, 2002).¹ Overall, CHOMP can be implemented to require only $O(n)$ computation every iteration.

¹In practice, n is typically small enough that computing the matrix inverse and performing the matrix-vector product do not dominate the per-iteration computation; our experiments, therefore, simply use the naïve implementation.

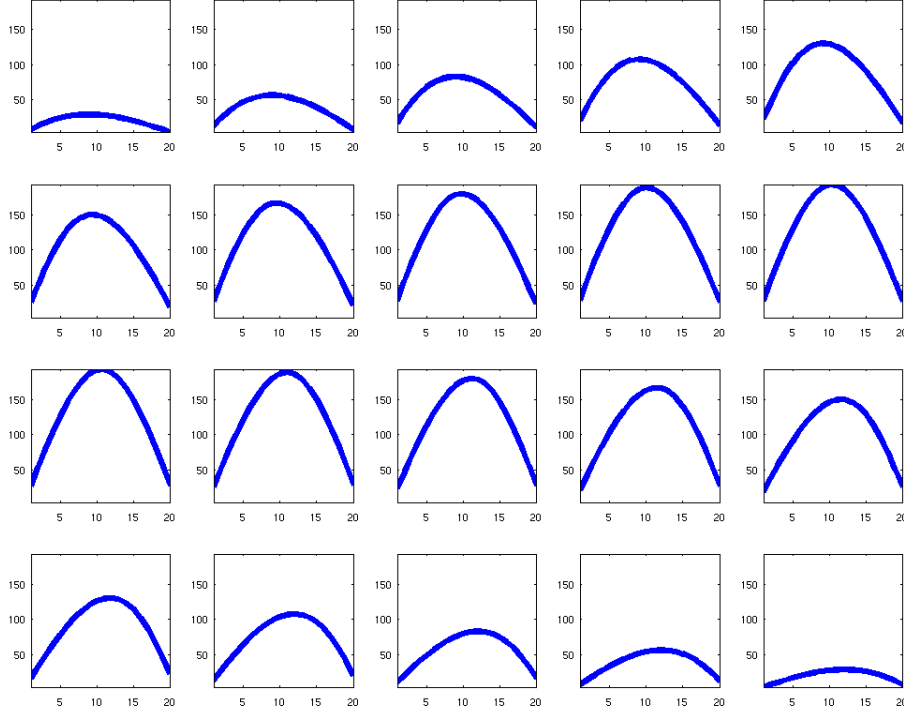


Figure 10.2: This figure shows the rows (equivalently, the columns) of the symmetric matrix A^{-1} with $d = 1$ in Equation 10.1. The i th subplot (counted from left to right, top to bottom) shows the components of the vector forming the i th row of A^{-1} as a function of the vector's index. As discussed in the text, i row of the matrix has zero acceleration everywhere, except at index i , where the acceleration is exactly 1. Transforming the Euclidean gradient g_k by this matrix effectively spreads the gradient influence across the trajectory.

10.2.2 Understanding the update rule

This update rule is a special case of a more general rule known as covariant gradient descent (Bagnell & Schneider, 2003a; Zlochin & Baram, 2001), in which the matrix A need not be constant.² In our case, it is useful to interpret the action of the inverse operator A^{-1} as spreading the gradient across the entire trajectory so that updating by the resulting covariant gradient decreases the cost while retaining trajectory smoothness. As an example, we take $d = 1$ and note that A is a finite differencing operator for approximating accelerations. Since $AA^{-1} = I$, we see that the i th row (equivalently, column, by symmetry) of A^{-1} has zero acceleration everywhere, except at the i th

²In the most general setting, the matrix A may vary smoothly as a function of the trajectory ξ .

entry. The transformed gradient $A^{-1}g_k$ can, therefore, be viewed as a vector of projections of g_k onto the set of smooth basis vectors forming A^{-1} . Figure 10.2 shows each row of A^{-1} as a function of the element index.

By measuring the size of trajectory perturbations in terms of their effect on the trajectory’s dynamics, we remove any dependence on the particular choice of trajectory representation. CHOMP is therefore *covariant*. This normative approach makes it easy to derive the CHOMP update rule: we can understand Equation 10.4 as the Lagrangian form of an optimization problem (Amari & Nagaoka, 2000) that attempts to maximize the decrease in our objective function subject to making only a small change in the relevant dynamics of the trajectory, rather than simply making a small change in the parameters that define the trajectory for a given representation.

We gain additional insight into the computational benefits of the covariant gradient based update by considering the analysis tools developed in the online learning/optimization literature, especially (Zinkevich, 2003; Hazan, Agarwal, & Kale, 2006). Analyzing the behavior of the CHOMP update rule in the general case is very difficult to characterize. However, by considering in a region around a local optima sufficiently small that f_{obs} is convex we can gain insight into the performance of both standard gradient methods (including those considered by, e.g. (Quinlan & Khatib, 1993)) and the CHOMP rule.

We first note that under these conditions, the overall CHOMP objective function is *strongly convex* (see Chapter 3)—that is, it can be lower-bounded over the entire region by a quadratic with curvature A . The authors of (Hazan, Agarwal, & Kale, 2006) show how gradient-style updates can be understood as sequentially minimizing a local quadratic approximation to the objective function. Gradient descent minimizes an uninformed, isotropic quadratic approximation while more sophisticated methods, like Newton steps, compute tighter lower bounds using a Hessian. In the case of CHOMP, the Hessian need not exist as our objective function may not even be differentiable, however we may still form a quadratic lower bound using A . This bound is much tighter than the isotropic alternative and leads to a correspondingly faster minimization of our objective—in particular, in accordance with the intuition of adjusting large parts of the trajectory due to the impact at a single way point we would generally expect it to be $O(n)$ times faster to

converge than a standard Euclidean gradient based method that initially adjusts only a single way point due an obstacle.

Importantly, we note that we are not simulating a mass-spring system as in (Quinlan & Khatib, 1993). We instead formulate the problem as covariant optimization in which we optimize directly within the space of trajectories; we posit that trajectories have natural notions of size and inner product as measured by their dynamics. In (Quinlan, 1994), a similar optimization setting is discussed, although, more traditional Euclidean gradients are derived. We demonstrate below that optimizing with respect to our smoothness norm substantially improves convergence.

Beyond deterministic gradient descent, we are additionally interested in utilizing covariant gradients to efficiently sample from a distribution defined by the cost function. To this end, we develop our algorithm around the *Hamiltonian Monte Carlo* (HMC) (Neal, 1993; Zlochin & Baram, 2001) sampling procedure. This Monte Carlo sampling technique utilizes gradient information and energy conservation concepts to efficiently navigate equiprobability curves of an augmented state-space. It can essentially be viewed as a well formulated method of integrating gradient information into Monte Carlo sampling; importantly, the samples are guaranteed to converge to a stationary distribution inversely proportional to the exponentiated objective function. HMC moves CHOMP in a direction toward designing a complete motion planning algorithm built solely on ideas from trajectory optimization. The next section describes HMC in detail.

10.2.3 From gradient descent to Monte Carlo sampling

Optimization and sampling can be tightly linked by considering the objective function $E(x)$ as the energy function of a Gibbs distribution of the form

$$p(x) \propto e^{-E(x)}. \quad (10.5)$$

While optimization routines are agnostic to the scale of the objective, distributions of this form read the scale as an indication of how quickly the probability should decrease away from the optimum; the larger the scale, the more tightly a sampling algorithm will center around a local minimizer (see (Neal, 1993) for details on this connection). This section reviews the *Hamiltonian*

Monte Carlo (HMC) sampling algorithm, which we use within CHOMP to turn the gradient descent procedure discussed above into an algorithm for sampling from a distribution over trajectories which places high probability in regions close to local minima of our objective function. Section 10.2.3 demonstrates how this class of sampling procedures allows us to use covariant gradient information to greatly improve the sampler’s performance.

Hamiltonian Monte Carlo

The most commonly used technique for sampling from general distributions of this form is Monte Carlo sampling. At a high level, these procedures randomly walk through the energy landscape $E(x)$, spending proportionally more time in low-energy regions than in high energy regions. Unfortunately, this random walk behavior makes naïve Monte Carlo procedures impractical for many time-sensitive applications such as motion planning. The Hamiltonian Monte Carlo algorithm, which we discuss here, removes this random walk behavior by utilizing conservation of energy concepts from physics to efficiently move between distant regions of the sampling space.

The algorithm proceeds by first augmenting the space with what are known as momentum variables u . Rather than sampling directly from $p(x)$, the algorithm instead generates samples from the augmented joint distribution

$$p(x, u) \propto e^{-E(x)-K(u)} = e^{-H(x, u)}. \quad (10.6)$$

In physics, $E(x)$ and $K(u)$ are understood as the potential energy and kinetic energy, respectively, and $H(x, u)$ is known as the Hamiltonian of the dynamical system. The following system of first-order differential equations can be simulated numerically using simple integration procedures to determine the motion of a particle that starts with an initial position x and momentum (velocity) u :

$$\begin{cases} \frac{dx_i}{dt} &= u_i \\ \frac{du_i}{dt} &= -\frac{\partial E}{\partial x_i} \end{cases} \quad (10.7)$$

where x_i and u_i are the i th components of x and u , respectively. This system simply states the well-known physical principles that a particle's change in position is given by its momentum, and the change in momentum is governed by the force from the potential field (which is known to be the negative gradient of that potential field). Importantly, an analysis of this system shows that all integral curves conserve total energy. Specifically, this means that if the pair $(x(t), u(t))$ is a solution to the system, the value of the Hamiltonian $H(x(t), u(t))$ (i.e. the total energy of the system) is constant, independent of t .

With regard to our joint distribution in Equation 10.6, this observation implies that the probabilities $p(x, u) \propto \exp\{-H(x(t), u(t))\}$ at any times t along this solution are all equal. In other words, simulating the Hamiltonian dynamics of the system traces out an equipotential curve of the distribution, allowing a sampler to easily travel between distinct regions of the space without a high risk of rejection. The Hamiltonian Monte Carlo algorithm builds on this intuition by using random walks only to wander between distinct equipotential curves. After each random transition between these curves, the sampler is allowed to randomly move anywhere along the equipotential curve; rejection occurs only if the numerical precision of the dynamical simulation is not sufficiently accurate.

Specifically, the kinetic energy is usually taken to be a simple isotropic quadratic function of the form $K(u) = \frac{1}{2}\|u\|^2$. The algorithm proceeds by iteratively sampling a momentum from the marginal $p(u) \propto \exp\{-K(u)\} = \exp\{-\frac{1}{2}\|u\|^2\}$, which, given our choice of kinetic energy function, is simply an isotropic Gaussian distribution. Taking that sample as the initial momentum for the Hamiltonian simulation, the procedure is then able to sample efficiently from $p(x|u)$. In combination, this algorithm produces samples from $p(u)p(x|u) = p(x, u)$ which provides a sample from the desired marginal $p(x)$ simply by removing the momentum components.

Often, the Hamiltonian dynamics are simulated using the following second-order integration

technique, known as the *leapfrog* method (Neal, 1993):

$$\begin{cases} u_{t+\frac{\epsilon}{2}} &= u_t - \frac{\epsilon}{2} \nabla E(x_t) \\ x_{t+\epsilon} &= x_t + \epsilon u_{t+\frac{\epsilon}{2}} \\ u_{t+\epsilon} &= u_{t+\frac{\epsilon}{2}} - \frac{\epsilon}{2} \nabla E(x_{t+\epsilon}) \end{cases} . \quad (10.8)$$

While it is common to write these equations as presented, it should be noted that when chaining multiple leapfrog steps together, the last half-step momentum update of the current iteration and the first half-step update of the next iteration can be combined into a single full-step update to avoid extraneous function and gradient evaluations.

There are typically two formal additions to this simple procedure that arise from the theory of Monte Carlo sampling. These additions ensure that the final samples are distributed according to $p(x)$ even when integration errors accumulate during the dynamical simulation.

First, the algorithm simulates the dynamics specifically for a random number of iterations forward in time with probability 1/2 and backward in time with probability 1/2 in order to maintain the time-reversibility property required by Monte Carlo procedures. This property states that the sampling procedure at each iteration must be able to get back to where it came from (along an equipotential curve) with the same probability as the probability of it arriving at that location from the starting configuration. The leapfrog method, itself, is time-reversible³, making it particularly convenient for this application.

Second, since each dynamical simulation is performed using a numerical integration procedure, the total energy of the final momentum and position variables may differ slightly from the initial energy. To compensate for this integration error, the formal Hamiltonian Monte Carlo algorithm prescribes a Monte Carlo rejection step at that point: if the new total energy is smaller than the original energy, then retain the point, otherwise, retain it with probability proportional to its likelihood ratio $p(x_T, u_T)/p(x_0, u_0)$. As has been observed in a number of other applications (Neal, 1993; Zlochins & Baram, 2001), in practice this step can be skipped without significantly affecting

³Specifically, if the leapfrog method is simulated forward with step size ϵ for n iterations, and then backward from the resulting point for n iterations with step size $-\epsilon$, it will end up precisely at the initial starting configuration (up to finite precision floating-point errors).

the empirical distribution of the resulting samples (or the application to which they pertain).

In practice, we implement a simulated annealing variant of HMC (Neal, 1993) which allows the procedure to converge specifically to a local minimum of the objective. One may view this simulated annealing variant as a principled way to merge the random restart and subsequent optimization stages of nonconvex optimization strategies typically left distinct in many optimizers.

HMC with constant metric covariant gradients

The Hamiltonian Monte Carlo algorithm is usually described in terms of Euclidean inner products. In our framework, we utilize alternate inner products that implement well informed priors over the space of trajectories as discussed in Section 10.2.1. In (Zlochín & Baram, 2001), the authors discuss a generalization of the Hamiltonian Monte Carlo algorithm to the case in which the inner product is defined by a general Riemannian metric that may vary from point to point in the space. In our case, the metric is constant, thus simplifying the algorithm slightly.

In particular, given a constant metric A over the space of trajectories, such as the metric described in Section 10.2.1, we must modify the algorithm in two places where the inner product is relevant: (1) in the gradient computation, for the same reasons as discussed above (the covariant gradient g_c is related to the Euclidean gradient g_e through the relation $g_c = A^{-1}g_e$); and (2) in the definition of kinetic energy. For this latter modification, we again choose the kinetic energy to be one-half the square norm of the momentum variable, but this time we define the norm in terms of the given inner product: $K(u) = \frac{1}{2}\langle u, u \rangle_A = \frac{1}{2}u' Au$.

10.2.4 Obstacles and distance fields

Let $\mathcal{B}$ denote the set of points comprising the robot body. When the robot is in configuration q , the workspace location of the element $u \in \mathcal{B}$ is given by the forward kinematics function

$$x(q, u) : \mathbb{R}^m \times \mathcal{B} \mapsto \mathbb{R}^3$$

A trajectory for the robot is then collision-free if for every configuration q along the trajectory and for all $u \in \mathcal{B}$, the distance from $x(q, u)$ to the nearest obstacle is more than $\varepsilon \geq 0$.

If obstacles are static and the description of $\mathcal{B}$ is geometrically simple, it becomes advantageous to simply precompute a Distance Field (DF) denoted $d(x)$, which stores the distance from a point $x \in \mathbb{R}^3$ to the boundary of the nearest obstacle. Values of $d(x)$ are zero inside obstacles and positive outside. Section 10.3 discusses a heuristic for dealing with obstacles in collision that works well for our manipulation experiments where obstacles may be thin relative to the robot. Alternative, we can compute a Signed Distance Field (SDF) that places negative distance inside obstacles, thereby allowing the obstacle to provide a valid gradient signal inside obstacles to push the robot free from collision. This latter heuristic works well when obstacles are large relative to the robot’s structure.

Computing $d(x)$ on a uniform grid is straightforward. We form the distance field by computing a Euclidean Distance Transform (EDT) over a boolean-valued voxel representation of the environment. For signed distance fields we additionally compute the logical compliment to that obstacle map and return a map whose voxels contain the difference between these two DFs. Computing the EDT is surprisingly efficient: for a lattice of K samples, computation takes time $O(K)$ (Felzenszwalb & Huttenlocher, 2004).

When applying CHOMP, we typically use a simplified geometric description of our robots, approximating the robot as a “skeleton” of spheres and capsules, or line-segment swept spheres. For a sphere of radius r with center x , the distance from any point in the sphere to the nearest obstacle is no less than $d(x) - r$. An analogous lower bound holds for capsules.

There are a few key advantages of using the signed distance field to check for collisions. Collision checking is very fast, taking time proportional to the number of voxels occupied by the robot’s “skeleton”. Since the signed distance field is stored over the workspace, computing its gradient via finite differencing is a trivial operation. Finally, because we have distance information everywhere, not just outside of obstacles, we can generate a valid gradient even when the robot is in collision – a particularly difficult feat for other representations and distance query methods.

Now we can define the workspace potential function $c(x)$, which penalizes points of the robot for being near obstacles. The simplest such function might be

$$c(x) = \max(\varepsilon - d(x), 0).$$

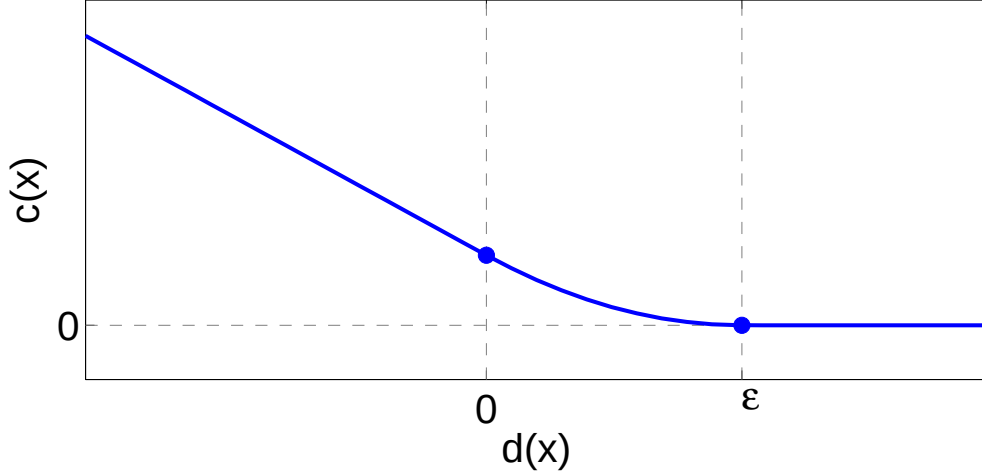


Figure 10.3: Potential function for obstacle avoidance

A smoother version, shown in figure 10.3, is given by

$$c(x) = \begin{cases} -d(x) + \frac{1}{2} \varepsilon, & \text{if } d(x) < 0 \\ \frac{1}{2\varepsilon} (d(x) - \varepsilon)^2, & \text{if } 0 \leq d(x) \leq \varepsilon \\ 0, & \text{otherwise} \end{cases}$$

10.2.5 Defining an obstacle potential

We will switch for a moment to discussing optimization of a continuous trajectory $q(t)$ by defining our obstacle potential as a functional over q . We can also derive the objective in a finite-dimensional setting by a priori choosing a trajectory discretization, but the properties of the objective function present themselves more clearly in the functional setting (see Section 10.2.6).

To begin, we define a workspace potential $c : \mathbb{R}^3 \rightarrow \mathbb{R}$ that quantifies the cost of a body element $u \in \mathcal{B}$ of the robot residing at a particular point x in the workspace.

Intuitively, we would like to integrate these cost values across the entire robot. A straightforward integration across time, however, is undesirable since moving more quickly through regions of high cost will be penalized less. Instead, we choose to integrate the cost elements with respect to an arc-length parameterization. Such an objective will have no motivation to alter the velocity profile along the trajectory since such operations do not change the trajectory's length. We will see that

this intuition manifests in the functional gradient as a projection of the workspace gradients onto the two-dimensional plane orthogonal to the direction of motion of a body element $u \in \mathcal{B}$ through the workspace.

We therefore write our obstacle objective as

$$f_{obs}[q] = \int_0^1 \int_{\mathcal{B}} c\left(x(q(t), u)\right) \left\| \frac{d}{dt} x(q(t), u) \right\| du dt$$

Since f_{obs} depends only on workspace positions and velocities (and no higher order derivatives), we can derive the functional gradient as $\bar{\nabla} f_{obs} = \frac{\partial v}{\partial q} - \frac{d}{dt} \frac{\partial v}{\partial \dot{q}}$, where v denotes everything inside the time integral (Courant & Hilbert, 1953; Quinlan, 1994). Applying this formula to f_{obs} , we get

$$\bar{\nabla} f_{obs} = \int_{\mathcal{B}} J^T \|x'\| \left[(I - \hat{x}' \hat{x}'^T) \nabla c - c\kappa \right] du \quad (10.9)$$

where κ is the curvature vector (do Carmo, 1976) defined as

$$\kappa = \frac{1}{\|x'\|^2} (I - \hat{x}' \hat{x}'^T) x''$$

and J is the kinematic Jacobian $\frac{\partial}{\partial q} x(q, u)$. To simplify the notation we have suppressed the dependence of J , x , and c on integration variables t and u . We additionally denote time derivatives of $x(q(t), u)$ using the traditional prime notation, and we denote normalized vectors by $\hat{x}$.

This objective function is similar to the objective discussed in Section 3.12 of (Quinlan, 1994). However, there is an important difference that substantially improves performance in practice. Rather than integrating with respect to arc-length through configuration space, we integrate with respect to arc-length in the workspace. This simple modification represents a fundamental change: instead of assuming the geometry in the configuration space is Euclidean, we compute geometrical quantities directly in the workspace where Euclidean assumptions are more natural.

Intuitively, we can more clearly see the distinction by examining the functional gradients of the two formulations. Operationally, the functional gradient defined in (Quinlan, 1994) can be

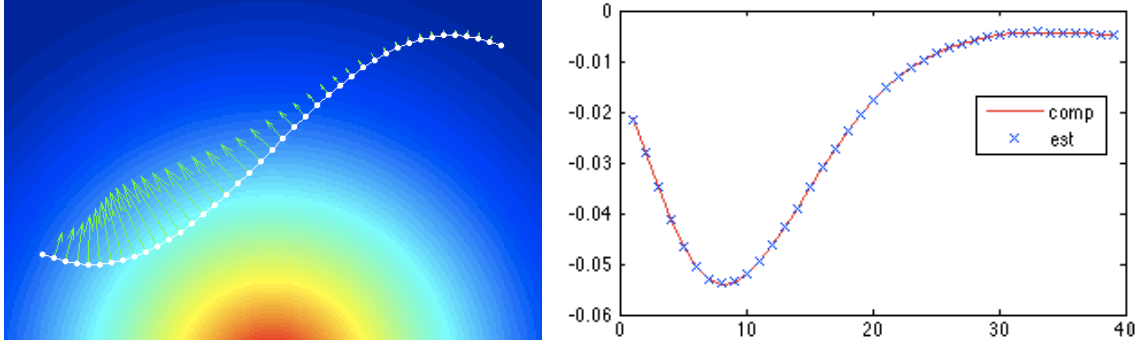


Figure 10.4: Left: A simple two-dimensional trajectory traveling through an obstacle potential (with large potentials in red gradating to small potentials in blue). The gradient at each configuration of the discretization depicted as a green arrow. Right: A plot of both the continuous functional gradient given in red and the corresponding Euclidean gradient component values of the discretization at each way point in blue.

computed in two steps. First, the configuration space gradient contributions that result from transforming each body element’s workspace gradient through the corresponding Jacobian are integrated across all body elements. Second, that single summarizing vector is projected orthogonally to the trajectory’s direction of motion in the configuration space. Alternatively, our objective performs this projection directly in the workspace before the transformation and integration steps. This difference ensures that orthogonality is measured with respect to the workspace geometry.

In practice, to implement these updates on a discrete trajectory ξ we approximate time derivatives using finite differences wherever they appear in the objective and its functional gradient. (The Jacobian J , of course, can be computed using the straightforward Jacobian of the robot.)

10.2.6 Functions vs functionals

Although Section 10.2.1 presents our algorithm in terms of a specific discretization, writing the objective in terms of functionals over continuous trajectories often emphasizes its properties. Section 10.2.5 exemplifies this observation. As Figure 10.4 demonstrates, the finite-dimensional Euclidean

gradient of a discretized version of the functional

$$f_{obs}(\xi) = \sum_{t=1}^n \sum_{u=1}^U \frac{1}{2} \left(c(x_u(q_{t+1})) + c(x_u(q_t)) \right) \cdot \|x_u(q_{t+1}) - x_u(q_t)\|$$

converges rapidly to the functional gradient as the resolution of the discretization increases. (In this expression, we denote the forward kinematics mapping of configuration q to body element u using $x_u(q)$.) However, the gradient of any finite-dimensional discretization of the f_{obs} takes on a substantially different form; the projection properties that are clearly identified in the functional gradient (Equation 10.9) are no longer obvious.

We note that the prior term can be written as a functional as well:

$$f_{prior}[\xi] = \sum_{d=1}^D \int_0^1 \|q'(t)\|^2 dt,$$

with functional gradient

$$\bar{\nabla} f_{prior}[\xi] = \sum_{d=1}^D (-1)^d q^{(2d)}$$

In this case, a discretization of the functional gradient $g = (\bar{\nabla} f_{prior}[\xi](\epsilon), \dots, \bar{\nabla} f_{prior}[\xi](\epsilon)(1 - \epsilon))^T$ exactly equals the gradient of the discretized prior when central differences are used to approximate the derivatives.

10.2.7 Smooth projection for joint limits

Joint limits are traditionally handled by either adding a new potential to the objective function which penalizes the trajectory for approaching the limits, or by performing a simple projection back onto the set of feasible joint values when a violation of the limits is detected. In our experiments, we follow the latter approach. However, rather than simply resetting the violating joints back to

Algorithm 17 Approximate projection for cost positivity constraints

```
1: procedure APPROXPROJECT( feature matrix  $F$ , vector of minimum costs  $c$  )
2:   while violations remain do
3:     Compute the update vector  $v$  used for  $L_1$  projection
4:     Transform the vector via our Riemannian metric  $\tilde{v} = A^{-1}v$ 
5:     Scale the resulting vector by  $\alpha$  such that  $\tilde{\xi} = \xi + \alpha\tilde{v}$  entirely removes the largest joint
       limit violation
6:   end while
7: end procedure
```

their limit values, which can be thought of as a L_1 projection on to the set of feasible values, we implement an approximate projection technique that projects with respect to the norm defined by the matrix A in Section 10.2.1.

At each iteration, we first find the vector of updates v that would implement the L_1 projection when added to the trajectory. However, before adding it, we transform that vector by the inverse of our metric A^{-1} . As discussed in Section 10.2.1, this transformation effectively smooths the vector across the entire trajectory so that the resulting update vector has little effect on the trajectory's dynamics. As a result, when we add a scaled version of that vector to our trajectory ξ , we can simultaneously remove the violations while retaining smoothness.

Our projection algorithm is listed formally in Algorithm 10.2.7. As indicated, we may need to iterate this procedure since the smoothing operation degrades a portion of the original projection signal. However, in our experiments, joint limit violations were often corrected within a single iteration of this procedure. Figure 10.5 plots the final joint angle curves over time from the final optimized trajectory on a robotic arm (see Section 10.3). The fourth subplot typifies the behavior of this procedure. While L_1 projection often produces trajectories that threshold at the joint limit, projection with respect to the acceleration norm produces a smooth joint angle trace which only briefly brushes the joint limit as a tangent.

10.3 Experiments on a robotic arm

This section presents experimental results for our implementation of CHOMP on Barrett Technology's WAM arm shown in Figure 10.1. We demonstrate the efficacy of our technique on a set of

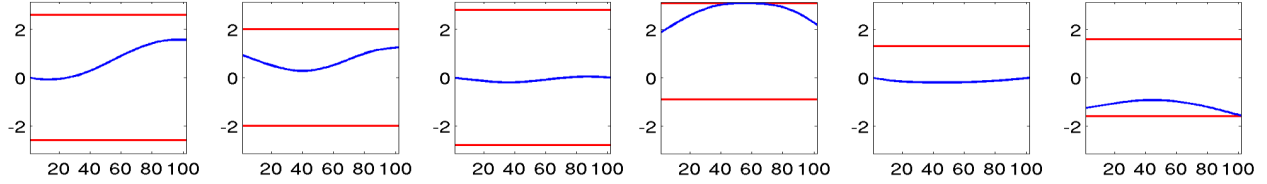


Figure 10.5: Left: This figure shows the joint angle traces that result from running CHOMP on the robot arm described in Section 10.3 using the smooth projection procedure discussed in Section 10.2.7. Each subplot shows a different joint’s trace across the trajectory in blue with upper and lower joint limits denoted in red. The fourth subplot typifies the behavior of projection procedure. The trajectory retains its smoothness while staying within the joint limit.

tasks representative of the type of tasks that may be commonly encountered in a home manipulation setting. The arm has seven degrees of freedom, although, we planned using only the first six in these experiments.⁴ Footage of the real-world implementation can be seen in the accompanying video.

10.3.1 Collision heuristic

In the home setting, obstacles are often thin (e.g. they may be pieces of furniture such as tables or doors). Section 10.2.4 discusses a heuristic based on the signed distance field under which the obstacles themselves specify how the robot should best remove itself from collision. This heuristic works well when the obstacle is large relative to the robot, but it can provide invalid information for smaller obstacles. An initial straight-line trajectory through the configuration space often contains configurations that pass entirely through an obstacle. In that case, the naïve workspace potential tends to simultaneously push the robot out of collision on one side and pull the robot further through the obstacle on the other side.

We avoid this behavior by adding an indicator function to the objective that makes all workspace terms that appear after the first collision along the arm vanish (as ordered via distance to the base). This indicator factor can be written mathematically as $I(\min_{j \leq i} d(x_j(q)))$, although it is implemented simply by ignoring all terms after the first collision while iterating from the base of the body out toward the end effector for a given time step along the trajectory.

Intuitively, this heuristic suggests simply that the workspace gradients encountered after then

⁴The last degree of freedom simply rotates the hand in place.

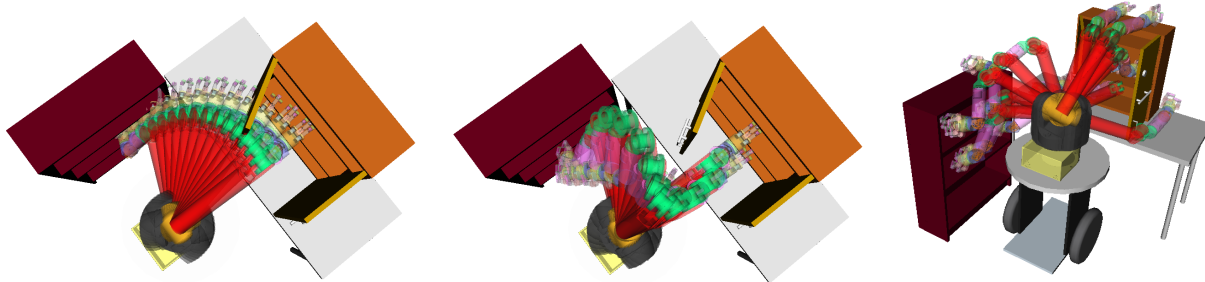


Figure 10.6: Left: the initial straight-line trajectory through configuration space. Middle: the final trajectory post optimization. Right: the 15 end point configurations used to create the 105 planning problems discussed in Section 10.3.

first collision of a given configuration are invalid and should therefore be ignored. Since we know the base of the robotic arm is always collision free, we are assured of a region along the arm prior to the first collision that can work to pull the rest of the arm out of collision. In our experiments, this heuristic works well to pull the trajectory free of obstacles commonly encountered in the home environment.

10.3.2 Planning performance results

We designed this experiment to evaluate the efficacy of CHOMP and its probabilistic variants as a replacement for planning on a variety of everyday household manipulation problems. We chose 15 different configurations in a given scene representing various tasks such as picking up an object from the table, placing an object on a shelf, or pulling an item from a cupboard. Using these start/goal points we generated 105 planning problem consisting of planning between all pairs of end configurations. Figure 10.6 shows the 15 end configurations (right) and compares the initial trajectory (left) to the final smoothed trajectory (middle) for one of these problems.

For this implementation, we modeled each link of the robot arm as a straight line, which we subsequently discretized into 10 evenly spaced points to numerically approximate the integrals over u in f_{obs} . Our voxel space used a discretization of $50 \times 50 \times 50$, and we used Matlab’s `bwdist` to compute the distance field. Under this resolution, the average distance field computation time was about .8 seconds. We ran both a straightforward covariant gradient descent variant of CHOMP in addition to a more sophisticated stochastic variant based on covariant Hamiltonian

Monte Carlo (HMC) sampling. See Section 10.2.2 for details. Under covariant gradient descent, we successfully solved 85 of the 105 problems. For each of these instances, we ran our optimizer for 400 iterations (approximately 12 seconds), although the core of the optimization typically completed within the first 100 iterations (approximately 3 seconds) during successful runs. However, adding stochasticity significantly improved the success rate. We implemented a restart procedure which reset the algorithm to its initial trajectory after 200 iterations if a collision free trajectory had not been found. Using this procedure our optimizer successfully found smooth collision free trajectories for all 105 of the problems. There were only five instances in which the procedure needed to restart more than twice. In the vast majority of the cases, it needed at most one restart.

We note that in our experiments, setting $A = I$ and performing Euclidean gradient descent performed extremely poorly. Euclidean gradient descent was unable to successfully pull the trajectory free from the obstacles.

10.3.3 An empirical analysis of optimization initialization

In the experiments discussed above, CHOMP was initialized to a simple straight-line trajectory through the configuration space. However, there is a long history of segmenting high-dimensional motion planning into two parts: an initialization stage, during which a randomized planner computes a feasible collision free trajectory; and an optimization stage where a trajectory optimizer smooths the resulting trajectory and optimizes it for dynamics. In practice, although the initial high-dimensional randomized planning stage is considered more difficult, the latter trajectory optimization stage often takes as much or even more computation time to converge on a good solution. The final optimized solution often shows little resemblance to the initial trajectory returned by the RRT.

This section empirically analyzes the difference between initializing CHOMP from a naïve straight-line trajectory and initializing CHOMP from a feasible trajectory found by an RRT. Interestingly, we find that in a large number of cases, the naïve initialization strategy outperforms optimization from an RRT solution both in terms of convergence rate and final trajectory cost. Since feasibility is not longer a precondition of optimization, it can actually be detrimental to

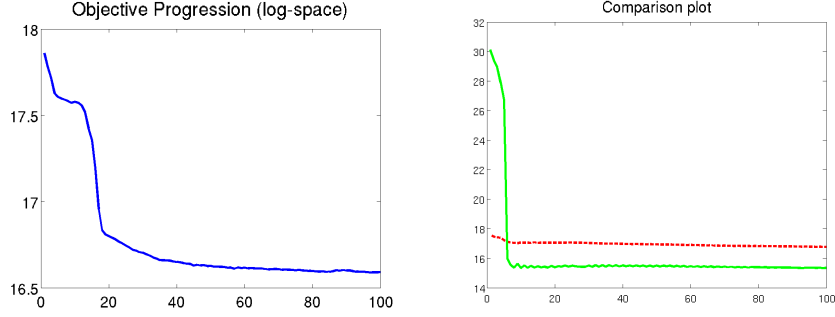


Figure 10.7: Left: the objective value per iteration of the first 100 iterations of CHOMP. Right: a comparison between the progression of objective values produced when starting CHOMP from a straight-line trajectory (green), and when starting CHOMP from the solution found by a bi-directional RRT. Without explicitly optimizing trajectory dynamics, the RRT returns a poor initial trajectory which causes CHOMP to quickly fall into a suboptimal local minimum.

spend the effort finding an initial feasible solution for initialization using randomized planning techniques.

We first analyze the performance of a straightforward covariant gradient descent variant of CHOMP by running it on the collection of problems introduced in Section 10.3.2 both initialized from a straight-line trajectory through configuration space and initialized from the feasible solution returned by a bi-direction RRT (we shorten this feasible path during a postprocessing phase using the traditional randomized shortcut heuristic (Kuffner & LaValle, 2000)). When CHOMP successfully finds a collision free trajectory, straight-line initialization typically outperforms the RRT initialization. On average, the log-objective value achieved when starting from a straight-line trajectory was approximately .5 units smaller than the value achieved when starting from the RRT solution on a scale that typically ranged from 17 to 24. This difference amounts to approximately 3% of the entire log-objective range spanned during optimization. Figure 10.7 depicts an example of the objective progressions induced by each of these initialization strategies.

Next, we ran a similar experiment, but using the HMC variant of CHOMP and varying the number of shortcut heuristic iterations used to postprocess the RRT path. The number of shortcut iterations ranged from 0 (i.e. no postprocessing of the RRT trajectory) to 20. Figure 10.8 shows the results of this experiment. Each plot depicts the difference per iteration between the objective progression under straight-line initialization and the objective progression under RRT initialization

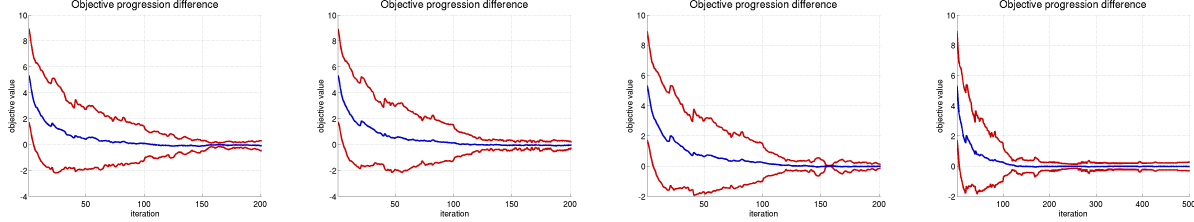


Figure 10.8: These plots show the difference in optimization performance between the straight-line initialization of CHOMP and an RRT initialization. Positive values indicate that RRT initialization has lower cost on average during those iterations, while negative values indicate that the straight-line initialization has lower cost. The average objective progression difference is shown in blue, and the upper and lower standard deviation bars are shown in red. From left to right, the plots depict the performance difference under 0, 3, 10, and 20 shortcut heuristic postprocessing iterations of the RRT solution.

averaged across all 105 planning problems. The mean progression difference is shown in blue and the upper and lower standard deviation bars are shown in red. From left to right, the plots compare the performance under 0, 3, 10, and 20 shortcut iterations, respectively.

During the initial stages of optimization, the RRT solution has lower cost simply because it is collision-free. The difference plot is, therefore, always initially positive through these iterations. However, the lower standard deviation bar quickly drops slightly below zero in all cases, indicating that for a large portion of the problems the straight-line initialization rapidly starts outperforming all other initialization strategies. Toward the end of the optimization the standard deviation bars tighten around the mean. Close observation of these plots shows that this average performance difference typically drops below zero by the time CHOMP converges, again supporting the observation that RRT initialization may bias the optimization toward local minima of slightly higher cost. As the number of postprocessing iterations of the shortcut heuristic increases, the optimization performance under the RRT initialization improves, but similar trends relative to the straight-line initialization strategy remain.

10.4 Conclusions

This work presents a powerful new trajectory optimization procedure that solves a much wider range of problems than previous optimizers by utilizing gradient information from the environment. The key concepts that contribute to the success of CHOMP all stem from utilizing superior notions

of geometry. Our experiments show that this algorithm substantially outperforms alternatives and improves performance on real world robotic systems. This work steps toward the fusion of randomized planning with trajectory optimization.

There are a few of important issues we have not yet addressed. First, in choosing *a priori* a discretization of a particular length, we are effectively constraining the optimizer to consider only trajectories of a predefined duration. A more general tool should dynamically add and remove samples during optimization. We believe the discretization-free functional representation discussed in Section 10.2.6 will provide a theoretically sound avenue through which we can accommodate trajectories of differing time lengths.

The Hamiltonian Monte Carlo variant performs well in our experiments and significantly improves the success rate across planning problems. However, further study is required to fully understand how it compares to competing state-of-the-art probabilistically complete planning procedures on problems spanning a wider range of difficulties. However, we note that each covariant gradient computation can easily leverage parallelization, both in the computation of the Euclidean gradient and in the solution of the linear system which transforms that gradient into a covariant gradient. Additionally, during the execution of HMC, each random perturbation stage effectively represents a branching point where parallelization can again be exploited to follow multiple branches simultaneously. Parallelization in these contexts provides an interesting avenue of future research.

Finally, this chapter demonstrates that CHOMP can be used to reduce motion planning to optimization. We are interested in applying the IOC learning techniques discussed throughout this thesis to the broad area of high-dimensional motion planning. Applying learning in this setting is not straightforward since good examples for imitation are difficult to generate. We, therefore, plan to explore techniques for connecting task-specific imitation learning with generalizable imitation learning by using the former to refine course human-generated examples for a specific domain and the latter to then generalize the refined examples to new domains.

The work described in this thesis is only a small step toward robots that learn efficiently and continuously. This chapter briefly outlines some interesting open problems that we are eager to pursue in future work.

11.1 The role of reinforcement learning

Reinforcement learning is a very hard problem; the objective function being optimized by a policy search algorithm is highly nonconvex and adorned with large suboptimal plateaus where the optimizer can easily become caught. Arbitrary random initialization of these algorithms finds poor performance in practical applications. Successful application of policy search techniques require expert initialization to good seed policies that already perform reasonably well. One obvious role of imitation learning in this setting is in programming an initial policy for subsequent optimization using reinforcement learning techniques that exploit environmental signals. Thus, imitation learning can be used to improve the performance of reinforcement learning techniques.

Interestingly, the relationship also points in the opposite direction: reinforcement learning can be used to improve the performance of imitation learning. There exists a large class of problems, including manipulation and legged locomotion, where it may be difficult to demonstrate appro-

priate example behavior. Effectively teleoperating a robot with many degrees of freedom can be unintuitive; in some cases, we may be able to backdrive the robot, but the resulting performance may still be insufficient for demonstration. However, since reinforcement learning algorithms are very good at improving policies and finding superior locally optimal solutions in a given domain, the achievable demonstrations may provide enough information to act as initial seeds to a domain specific reinforcement learning technique that can fine-tune the demonstration into a trajectory exemplifying the desired behavior. In this case, reinforcement learning can be used to generate good domain specific examples from which our imitation learning algorithms can generalize.

In essence, there exists a strong connection between imitation learning and reinforcement learning that is not currently being exploited: imitation learning can be used to improve reinforcement learning, and reinforcement learning can be used to improve imitation learning. We are eager to explore this connection to develop an integrated algorithm that can continually augment the policy using local environmental signals in the absence of expert demonstration, and also leverage global signals from example trajectories in order to more significantly modify the policy when such information is available.

11.2 Functional bundles for structured prediction

In Section 4.3, we demonstrated that functional bundle methods optimize very well in practice. In this case, superior optimization translates both into better generalization performance as well as into a more efficient function representation. Both of these considerations are particularly critical for large-scale structured prediction problems. For these problems, evaluating the objective may be very slow as it requires running a structured inference algorithm, and in practice slow hypothesis evaluation can subvert real-time performance. Experimental efforts with functional bundle methods in structured prediction settings are currently under way.

In Section 4.3.3, we used the solution to the *function value* mathematical program shown in Equation 4.3.3 to motivate parameterizing the functional bundle as a linear combination of existing function approximators. However, we additionally want to explore the alternative bundle optimization technique suggested in that section where the optimal function values attained directly

by the function value optimization are used as labels to train a single function approximator as the new hypothesis. These optimal function values are the values we would like our hypothesis to attain at the existing data points; training a function approximator using these labels generalizes this information to the entire domain. This function value bundle optimization remains a small problem in the dual space. With no extra work (except perhaps the added computation of training a more complex function approximator at each iteration), this variant can train a function approximator with a constant sized representation. For some problems in structured prediction, in particular, this feature may prove to be important.

11.3 Theoretical understanding of functional gradient algorithms

Throughout this thesis, we have shown that our novel functional gradient algorithms perform well in practice. Indeed, they demonstrate much better performance on real-world problems than their linear counterparts because their flexible hypothesis space aids in avoiding complicated feature engineering. Unfortunately, we know very little about the theoretical properties of these algorithms.

We have shown, in the case of exponentiated functional gradient descent, that the algorithm will make progress on each iteration as long as the functional gradients are exact. In practice, however, we rarely have access to the exact functional gradient; all boosting-type functional gradient descent variants require function approximators to best represent the abstract functional gradient. In future work, we want to explore connections between the generalization or regret performance of the function approximator and the performance of the optimization procedure.

- Abbeel, P., and Ng, A. Y. 2004. Apprenticeship learning via inverse reinforcement learning. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*.
- Abbeel, P.; Coates, A.; Quigley, M.; and Ng, A. Y. 2007. An application of reinforcement learning to aerobatic helicopter flight. In *Neural Information Processing Systems 19*.
- Amari, S., and Nagaoka, H. 2000. *Methods of Information Geometry*. Oxford University Press.
- Anderson, B. D. O., and Moore, J. B. 1990. *Optimal control : linear quadratic methods*. Prentice Hall, Englewood Cliffs, N.J. :.
- Anguelov, D.; Taskar, B.; Chatalbashev, V.; Koller, D.; Gupta, D.; Heitz, G.; and Ng, A. 2005. Discriminative learning of markov random fields for segmentation of 3d scan data. In *Conference on Computer Vision and Pattern Recognition*.
- Bagnell, J., and Schneider, J. 2003a. Covariant policy search. In *International Joint Conference on Artificial Intelligence*.
- Bagnell, J. A. D., and Schneider, J. 2003b. Policy search in reproducing kernel hilbert space. Technical Report CMU-RI-TR-03-45, Robotics Institute, Pittsburgh, PA.
- Bagnell, J. A. D. 2004. *Learning Decisions: Robustness, Uncertainty, and Approximation*. Ph.D. Dissertation, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.
- Bain, M., and Sammut, C. 1995. A framework for behavioral cloning. In *Machine Intelligence Agents*. Oxford University Press.

- Bartlett, P.; Collins, M.; Taskar, B.; and McAllester, D. 2004. Exponentiated gradient algorithms for large-margin structured classification. In *Advances in Neural Information Processing Systems (NIPS04)*.
- Bottou, L., and Bousquet, O. 2008. The tradeoffs of large scale learning. In Platt, J.; Koller, D.; Singer, Y.; and Roweis, S., eds., *Advances in Neural Information Processing Systems*, volume 20, 161–168.
- Boyd, S., and Vandenberghe, L. 2004. *Convex Optimization*. Cambridge University Press.
- Boyd, S.; Ghaoui, L. E.; Feron, E.; and Balakrishnan, V. 1994. *Linear Matrix Inequalities in System and Control Theory*. Society for Industrial and Applied Mathematics (SIAM).
- Brock, O., and Khatib, O. 2002. Elastic Strips: A Framework for Motion Generation in Human Environments. *The International Journal of Robotics Research* 21(12):1031.
- Cesa-Bianchi, N., and Lugosi, G. 2006. *Prediction, Learning, and Games*. New York, NY, USA: Cambridge University Press.
- Cesa-Bianchi, N.; Conconi, A.; and Gentile, C. 2004a. On the generalization ability of on-line learning algorithms. In *IEEE Trans. on Information Theory*, volume 50-9, 2050–2057. Preliminary version in Proc. of the 14th conference on Neural Information processing Systems (NIPS 2001).
- Cesa-bianchi, N.; Conconi, A.; and Gentile, C. 2004b. On the generalization ability of on-line learning algorithms. *IEEE Transactions on Information Theory* 50:2050–2057.
- Cesa-Bianchi, N.; Long, P.; and Warmuth, M. K. 1994. Worst-case quadratic bounds for on-line prediction of linear functions by gradient descent. *IEEE Transactions on Neural Networks* 7:604–619.
- Chen, P., and Hwang, Y. 1998. SANDROS: a dynamic graph search algorithm for motion planning. *Robotics and Automation, IEEE Transactions on* 14(3):390–403.
- Chestnutt, J.; Kuffner, J.; Nishiwaki, K.; and Kagami, S. 2003. Planning biped navigation strategies in complex environments. In *Proceedings of the IEEE-RAS International Conference on Humanoid Robots*.
- Chestnutt, J.; Lau, M.; Cheng, G.; Kuffner, J.; Hodgins, J.; and Kanade, T. 2005. Footstep planning for the Honda ASIMO humanoid. In *Proceedings of the IEEE International Conference on Robotics and Automation*.
- Coates, A.; Abbeel, P.; and Ng, A. Y. 2008. Learning for control from multiple demonstrations. In *Proceedings of ICML*.

- Collins, M., and Roark, B. 2004. Incremental parsing with the perceptron algorithm. In *Proc. ACL*, 111–118.
- Courant, R., and Hilbert, D. 1953. *Methods of Mathematical Physics*. Interscience, 1953. Repulished by Wiley in 1989.
- Dietterich, T. G.; Becker, S.; and Ghahramani, Z., eds. 2001. *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*. MIT Press.
- do Carmo, M. P. 1976. *Differential geometry of curves and surfaces*. Prentice-Hall.
- Donoho, D. L., and Elad, M. 2003. Maximal sparsity representation via l_1 minimization. In *the Proc. Nat. Aca. Sci.* 100, 2197–2202.
- Earl, M.; Technol, R.; Syst, B.; and Burlington, M. 2005. Iterative MILP methods for vehicle-control problems. *IEEE Transactions on Robotics* 21(6):1158–1167.
- Felzenszwalb, P., and Huttenlocher, D. 2004. Distance Transforms of Sampled Functions. Technical Report TR2004-1963, Cornell University.
- Freund, Y., and Schapire, R. E. 1995. A decision-theoretic generalization of on-line learning and an application to boosting. In *EuroCOLT '95: Proceedings of the Second European Conference on Computational Learning Theory*, 23–37. London, UK: Springer-Verlag.
- Freund, Y., and Schapire, R. 1999. Adaptive game playing using multiplicative weights. *Games and Economic Behavior* 79–103.
- Friedman, J. H. 1999a. Greedy function approximation: A gradient boosting machine. In *Annals of Statistics*, volume 29(5).
- Geraerts, R., and Overmars, M. 2006. Creating High-quality Roadmaps for Motion Planning in Virtual Environments. *IEEE/RSJ International Conference on Intelligent Robots and Systems* 4355–4361.
- Gordon, G. 1999. *Approximate Solutions to Markov Decision Processes*. Ph.D. Dissertation, Robotics Institute, Carnegie Mellon University.
- Hassani, S. 1998. *Mathematical Physics*. Springer.
- Hazan, E.; Agarwal, A.; and Kale, S. 2006. Logarithmic regret algorithms for online convex optimization. In *In COLT*, 499–513.

- Herbster, M., and Warmuth, M. K. 2001. Tracking the best linear predictor. *Journal of Machine Learning Research* 1:281–309.
- Hinton, G. E.; Osindero, S.; and Teh, Y. 2006. A fast learning algorithm for deep belief nets. In *Neural Computation*, volume 18, 1527–1554.
- ichi Amari, S. 1998. Natural gradient works efficiently in learning. *Neural Computation* 10(2):251–276.
- Joachims, T. 2006. Training linear svms in linear time. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD)*.
- Kakade, S. 2002. A natural policy gradient. In Dietterich, T. G.; Becker, S.; and Ghahramani, Z., eds., *Advances in Neural Information Processing Systems 14*. Cambridge, MA: MIT Press.
- Kalman, R. 1964. When is a linear control system optimal? *Trans. ASME, J. Basic Engrg.* 86:51–60.
- Kavraki, L., and Latombe, J. 1998. Probabilistic roadmaps for robot path planning. *Practical Motion Planning in Robotics: Current Approaches and Future Directions* 53.
- Kavraki, L.; Švestka, P.; Latombe, J. C.; and Overmars, M. H. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration space. *IEEE Trans. on Robotics and Automation* 12(4):566–580.
- Kivinen, J., and Warmuth, M. K. 1997. Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation* 132.
- Kivinen, J., and Warmuth, M. 2001. Relative loss bounds for multidimensional regression problems. *Machine Learning Journal* 45:301–329.
- Kivinen, J.; Smola, A. J.; and Williamson, R. C. 2002. Online learning with kernels. In Dietterich, T. G.; Becker, S.; and Ghahramani, Z., eds., *Advances in Neural Information Processing Systems 14*. Cambridge, MA: MIT Press.
- Kolter, J. Z.; Abbeel, P.; and Ng, A. Y. 2008. Hierarchical apprenticeship learning with application to quadruped locomotion. In *Neural Information Processing Systems 20*.
- Krumm, J. 2008. A markov model for driver route prediction. *Society of Automotive Engineers (SAE) World Congress*.

- Kuffner, J., and LaValle, S. 2000. RRT-Connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation*, 995–1001.
- LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86(11), 2278–2324.
- LeCun, Y.; Muller, U.; Ben, J.; Cosatto, E.; and Flepp, B. 2006. Off-road obstacle avoidance through end-to-end learning. In *Advances in Neural Information Processing Systems 18*. MIT Press.
- LeCun, Y.; Chopra, S.; Hadsell, R.; Huang, F.-J.; and Ranzato, M.-A. 2007. A tutorial on energy-based learning. In *Predicting Structured Outputs*. The MIT Press.
- Littlestone, N., and Warmuth, M. K. 1989. The weighted majority algorithm. In *IEEE Symposium on Foundations of Computer Science*.
- Ma, C.; Miller, R.; Syst, N.; and El Segundo, C. 2006. MILP optimal path planning for real-time applications. In *American Control Conference, 2006*, 6.
- Mason, L.; J.Baxter; Bartlett, P.; and Frean, M. 1999. Functional gradient techniques for combining hypotheses. In *Advances in Large Margin Classifiers*. MIT Press.
- Miller, A. T.; Knoop, S.; Allen, P. K.; and Christensen, H. I. 2003. Automatic grasp planning using shape primitives. In *Proceedings of the IEEE International Conference on Robotics and Automation*.
- Munoz, D.; Bagnell, J. A. D.; Vandapel, N.; and Hebert, M. 2009. Contextual classification with functional max-margin markov networks. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Munoz, D.; Vandapel, N.; and Hebert, M. 2008. Directional associative markov network for 3-d point cloud classification. In *Fourth International Symposium on 3D Data Processing, Visualization and Transmission*.
- Neal, R. M. 1993. Probabilistic Inference Using Markov Chain Monte Carlo Methods. Technical Report CRG-TR-93-1, University of Toronto, Dept. of Computer Science.
- Nedic, A., and Bertsekas, D. 2000. Convergence rate of incremental subgradient algorithms. *Stochastic Optimization: Algorithms and Applications*.
- Neu, G., and Szepesvari, C. 2007. Apprenticeship learning using inverse reinforcement learning and gradient methods. In *Proc. UAI*, 295–302.

- Ng, A. Y., and Russell, S. 2000. Algorithms for inverse reinforcement learning. In *Proc. 17th International Conf. on Machine Learning*.
- Nigam, K.; Lafferty, J.; and McCallum, A. 1999. Using maximum entropy for text classification. In *IJCAI-99 Workshop on Machine Learning for Information Filtering*.
- Pomerleau, D. 1989. ALVINN: An autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems 1*.
- Puterman, M. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley.
- Quinlan, S., and Khatib, O. 1993. Elastic bands: connecting path planning and control. In *IEEE International Conference on Robotics and Automation*, 802–807.
- Quinlan, S. 1994. *The Real-Time Modification of Collision-Free Paths*. Ph.D. Dissertation, Stanford University.
- Ramachandran, D., and Amir, E. 2007. Bayesian inverse reinforcement learning. In *Proc. IJCAI*, 2586–2591.
- Ratliff, N., and Bagnell, J. A. D. 2007. Kernel conjugate gradient for fast kernel machines. In *International Joint Conference on Artificial Intelligence*, volume 20.
- Ratliff, N.; Bagnell, J. A.; and Zinkevich, M. 2006. Maximum margin planning. In *Twenty Second International Conference on Machine Learning (ICML06)*.
- Ratliff, N.; Bagnell, J. A.; and Zinkevich, M. 2007a. (Online) subgradient methods for structured prediction. In *Artificial Intelligence and Statistics*.
- Ratliff, N.; Bagnell, J. A.; and Zinkevich, M. 2007b. (online) subgradient methods for structured prediction. In *Proc. AISTATS*.
- Ratliff, N.; Bradley, D.; Bagnell, J. A.; and Chestnutt, J. 2006. Boosting structured prediction for imitation learning. In *NIPS*.
- Ratliff, N.; Ziebart, B.; Peterson, K.; Bagnell, J. A. D.; Hebert, M.; Dey, A.; and Srinivasa, S. 2009a. Inverse optimal heuristic control for imitation learning. In *Twelfth International Conference on Artificial Intelligence and Statistics (AISTats)*.
- Ratliff, N.; Zucker, M.; Bagnell, J. A. D.; and Srinivasa, S. 2009b. Chomp: Gradient optimization techniques for efficient motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*.

- Ratliff, N.; Silver, D.; and Bagnell, J. A. 2009. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots, Special Issue on Robot Learning*.
- Ratliff, N.; Srinivasa, S.; and Bagnell, J. A. 2007. Imitation learning for locomotion and manipulation. In *IEEE-RAS International Conference on Humanoid Robots*.
- Rifkin, Y., and Poggio. 2003. Regularized least squares classification. In Suykens, e. a., ed., *Advances in Learning Theory: Methods, Models and Applications*, volume 190. IOS Press.
- Rosset, S.; Zhu, J.; and Hastie, T. 2004. Boosting as a regularized path to a maximum margin classifier. *J. Mach. Learn. Res.* 5:941–973.
- Schaal, S., and Atkeson, C. 1993. Open loop stable control strategies for robot juggling. In *Proceedings of the 93 IEEE Int. Conf. on Robotics and Automation*.
- Schölkopf, B., and Smola, A. 2002. *Learning with Kernels: Support vector machines, regularization, optimization, and beyond*. MIT Press.
- Schölkopf, B.; Herbrich, R.; Smola, A. J.; and Williamson, R. C. 2000. A generalized representer theorem: Nc-tr-00-081. Technical report, NeuroCOLT Technical Report.
- Schouwenaars, T.; De Moor, B.; Feron, E.; and How, J. 2001. Mixed integer programming for multi-vehicle path planning. In *European Control Conference*, 2603–2608.
- Shalev-Shwartz, S., and Srebro, N. 2008. Svm optimization: Inverse dependence on training set size. In *Proceedings of ICML*.
- Shalev-Shwartz, S.; Singer, Y.; and Srebro, N. 2007. Pegasos: Primal estimated sub-gradient solver for svm. In *Proceedings of ICML*.
- Shiller, Z., and Dubowsky, S. 1991. On computing the global time-optimal motions of robotic manipulators in the presence of obstacles. *IEEE Transactions on Robotics and Automation* 7(6):785–797.
- Shmoys, D., and Swamy, C. 2004. An approximation scheme for stochastic linear programming and its application to stochastic integer programs. In *FOCS*.
- Shor, N. Z. 1985. *Minimization Methods for Non-Differentiable Functions*. Springer-Verlag.
- Silver, D.; Bagnell, J. A.; and Stentz, A. 2008. High performance outdoor navigation from overhead data using imitation learning. In *Proceedings of Robotics Science and Systems*.

- Simmons, R.; Browning, B.; Zhang, Y.; and Sadekar, V. 2006. Learning to predict driver route and destination intent. *Proc. Intelligent Transportation Systems Conference* 127–132.
- Smola, A. J., and Schölkopf, B. 2003. A tutorial on support vector regression. Technical report, Statistics and Computing.
- Smola, A.; Vishwanathan, S.; and Le., Q. 2008. Bundle methods for machine learning. In *NIPS 20*.
- Sundar, S.; Shiller, Z.; Inc, A.; and Santa Clara, C. 1997. Optimal obstacle avoidance based on the Hamilton-Jacobi-Bellman equation. *Robotics and Automation, IEEE Transactions on* 13(2):305–310.
- Szeliski, R.; Zabih, R.; Scharstein, D.; Veksler, O.; Kolmogorov, V.; Agarwala, A.; Tappen, M.; and Rother, C. 2006. A comparative study of energy minimization methods for markov random fields. In *European Conference on Computer Vision, II*: 16–29.
- Taskar, B.; Chatalbashev, V.; Guestrin, C.; and Koller, D. 2005. Learning structured prediction models: A large margin approach. In *Twenty Second International Conference on Machine Learning (ICML05)*.
- Taskar, B.; Guestrin, C.; and Koller, D. 2003. Max margin markov networks. In *Advances in Neural Information Processing Systems (NIPS-14)*.
- Taskar, B.; Lacoste-Julien, S.; and Jordan, M. 2006. Structured prediction via the extragradient method. In *Advances in Neural Information Processing Systems 18*. MIT Press.
- Tropp, J. A. 2004. Greed is good: Algorithmic results for sparse approximation. *IEEE Trans. Inform. Theory* 50:2231–2242.
- Tsochantaridis, I.; Joachims, T.; Hofmann, T.; and Altun, Y. 2005. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research* 1453–1484.
- Tsuda, K.; Rätsch, G.; and Warmuth, M. K. 2005. Matrix exponentiated gradient updates for on-line learning and bregman projection. *Journal of Machine Learning Research* 6:995–1018.
- Valiant, L. G. 1984. A theory of the learnable. In *C. ACM*, volume 27, 1134–1142.
- Vandapel, N.; Huber, D.; Kapuria, A.; and Hebert, M. 2004. Natural terrain classification using 3-d lidar data. In *IEEE International Conference on Robotics and Automation*.
- Vapnik, V. 1995. *The Nature of Statistical Learning Theory*. Springer-Verlag, NY, USA.

- Vitus, M.; Pradeep, V.; Hoffmann, G.; Waslander, S.; and Tomlin, C. 2008. Tunnel-milp: Path planning with sequential convex polytopes. In *AIAA Guidance, Navigation, and Control Conference*.
- Wainwright, M. J. 2002. *Stochastic Processes on Graphs: Geometric and Variational Approaches*. Ph.D. Dissertation, Massachusetts Institute of Technology.
- Yagi, M., and Lumelsky, V. 1999. Biped robot locomotion in scenes with unknown obstacles. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 375–380.
- Zhang, T. 2002. Covering number bounds of certain regularized linear function classes. *Journal of Machine Learning Research* 2:527–550.
- Ziebart, B.; Bagnell, J. A.; Maas, A.; and Dey, A. 2008a. Maximum entropy inverse reinforcement learning. In *Twenty-third AAAI Conference*.
- Ziebart, B.; Maas, A.; Dey, A.; and Bagnell, J. A. 2008b. Navigate like a cabbie: Probabilistic reasoning from observed context-aware behavior. In *Proc. UbiComp*, 322–331.
- Ziebart, B.; Maas, A.; Dey, A.; and Bagnell, J. D. 2008c. Navigate like a cabbie: Probabilistic reasoning from observed context-aware behavior. In *UBICOMP: Ubiquitous Computation*.
- Zinkevich, M. 2003. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the Twentieth International Conference on Machine Learning*.
- Zlochin, M., and Baram, Y. 2001. Manifold stochastic dynamics for bayesian learning. *Neural Comput.* 13(11):2549–2572.