Inverse Optimal Control:
What do we optimize?

Nathan Ratliff

May 27, 2014

Abstract

How do we design the behavior of an autonomous system? Optimal control says, give us a cost function and we'll give you an optimal policy. Can't get any better than that, right? So we go to work designing a cost function. But for all our efforts to stay principled, we've all been there, in the lab, late at night, tuning parameters, weighing cost components, designing cost features, and tweaking, tweaking, tweaking to get the darned thing to do what we want. Optimal control methods are optimizers, but in reality, we don't really know what we should be optimizing. This document addresses the question of how we can, instead, leverage our strong intuition of how we'd like the robot to behave to *learn* a cost function that embodies demonstrations of that behavior. So defines Inverse Optimal Control (IOC). Rather than computing an optimal policy given a cost function, IOC uncovers the cost function that best explains demonstrated optimal behavior. This problem is strongly related to a broad class of very large scale multiclass classification problems (Structured Prediction) in machine learning, and we start by deriving and discussing that connection in detail. We then derive a simple gradient-based method for learning that that solves IOC. The algorithm consists of an outer loop optimization that repeatedly solves the optimal control problem under successively improving hypotheses and subsequently corrects errors the perceived behavior. The algorithm effectively automates the manual guess and check process that plagues the design of optimal controllers.

1 Cost functions are arbitrary

Optimal control is built on an abstraction of the form

$$\begin{aligned}\min_{\boldsymbol{u}_{1:T},\boldsymbol{s}_{1:T+1}} & \sum_{t=1}^{T} c_t(\boldsymbol{s}_t,\boldsymbol{u}_t)+\psi(\boldsymbol{s}_{T+1}) \\ \text{s.t.}\quad & \boldsymbol{s}_{t+1}=\boldsymbol{f}(\boldsymbol{s}_t,\boldsymbol{u}_t).\end{aligned} \tag{1}$$

But what does this actually mean in practice? Defining optimal control in terms of cost functions c_t posits that we know explicitly how to rank states and actions relative to one another. But how bad is it really if a robot gets close to a wall and applies a particular torque to one of its joints? And more importantly, is that 10 units of cost worse than being close to a bush? We don't really know. All we can really say is that we have a collection of cost measurement components, such as the proximity of various points on the robot to the closest obstacle, penalty terms on motion derivatives and torques, stability measures, measures of contact utility, and often many many more. How we trade those terms off is entirely unclear from the outset. All we can do in practice is try them out. Guess, check, and tune until the system does what we want.

This cost tuning is one of those dirty secrets that makes implementing real-world robotic systems so hard. We never know if we fundamentally need more expressive terms or if we just haven't found the right combination of the ones we already have to achieve the desired behavior. But, on the other hand, we usually do actually know what we want, and, importantly, we can even demonstrate that desired behavior. Ideally, we'd like to invert the optimal control problem: from those demonstrations of desired system behavior (which are relatively easy to come by), can we automatically find the cost function under which the Optimal Control abstraction of Equation 1 implement that behavior?

That question is the central question of Inverse Optimal Control IOC Kalman (1964); Ratliff (2009). This document studies the relationship between this IOC problem and the more general problem of multiclass classification in machine learning that addresses the more abstract problem of generalizing the information encoded in data. We'll see how extremely large scale multiclass classification renders a framework under which we can formalize and solve IOC problems robustly in a way that intelligently automates the guess and check cycle that's typically forced on the engineer.

2 Linear combinations of features and general applicability

Before formalizing the discussion above, lets consider for a moment at what actually goes into a cost function. We abstractly denote state-action costs as $c_t(\boldsymbol{s}_t, \boldsymbol{u}_t)$, but hidden within that cost function is a notion of generality. We want our cost functions to work anywhere, not just along the test corridor of our lab. That means the cost components need to encode information about the geometry of the environment around it. We mentioned above one such cost function that allows us to do this: proximity to obstacles. A cost component that measures proximity to obstacles works anywhere. Place the robot in a forest, we can measure proximity to obstacles; place the robot on a sidewalk, we can still measure proximity to obstacles. This number, the measured proximity to an obstacle, generalizes. It's not tied to any single location—it works everywhere.

2.1 Abstracting optimal control in terms of features

From here on out, we'll assume our cost components generalize. That means each cost component is not just a function of the state and action, but also a function of the environment or context. Abstractly, for this problem of optimal control, we can denote the context as $\boldsymbol{\gamma}$. Denoting each of the k context dependent intermediate cost components as $c_t^{(i)}(\boldsymbol{\gamma}, \boldsymbol{s}_t, \boldsymbol{u}_t)$ and each of the l context dependent terminal cost components as $\psi^{(j)}(\boldsymbol{\gamma}, \boldsymbol{s}_{T+1})$, our optimal control problem becomes

$$\begin{aligned} \min_{\boldsymbol{u}_{1:T}, \boldsymbol{s}_{1:T+1}} & \sum_{t=1}^{T} \left(\sum_{i=1}^{k} \alpha_i c_t^{(i)}(\boldsymbol{s}_t, \boldsymbol{u}_t) \right) + \sum_{j=1}^{l} \beta_j \psi^{(j)}(\boldsymbol{s}_{T+1}) \\ \text{s.t.} \;\; & \boldsymbol{s}_{t+1} = \boldsymbol{f}(\boldsymbol{s}_t, \boldsymbol{u}_t). \end{aligned} \tag{2}$$

To emphasize the structure even further, we can denote the trajectory as $\xi = \{(\boldsymbol{s}_1, \boldsymbol{u}_1), \ldots, (\boldsymbol{s}_T, \boldsymbol{u}_T), \boldsymbol{s}_{T+1}\}$ and write the optimization problem as

$$\min_{\xi \in \Xi(\boldsymbol{\gamma})} \underbrace{\sum_{i=1}^{k} \alpha_i \left(\sum_{t=1}^{T} c_t^{(i)}(\boldsymbol{\gamma}, \boldsymbol{s}_t, \boldsymbol{u}_t) \right)}_{\boldsymbol{\alpha}^T \boldsymbol{c}(\boldsymbol{\gamma}, \xi)} + \underbrace{\sum_{j=1}^{l} \beta_j \boldsymbol{\psi}^{(j)}(\boldsymbol{\gamma}, \boldsymbol{s}_{T+1})}_{\boldsymbol{\beta}^T \boldsymbol{\psi}(\boldsymbol{\gamma}, \xi)}, \tag{3}$$

where $\Xi(\boldsymbol{\gamma})$ denotes the set of all feasible trajectories under the dynamical constraints $\boldsymbol{s}_{t+1} = \boldsymbol{f}(\boldsymbol{s}_t, \boldsymbol{u}_t)$ for this context. As noted in the above equation we can abstractly describe this optimal control problem as an optimization over a linear combination of context-dependent *cumulative* features $[\boldsymbol{c}(\boldsymbol{\gamma}, \xi); \boldsymbol{\psi}(\boldsymbol{\gamma}, \xi)]$ that we calculate across a given trajectory ξ. Intuitively, these cumulative features represent what the robot tends to see across the trajectory. Note that the character or behavior of the trajectory is in a sense encoded in these cumulative features. Section 4.1 details these points more extensively.

For the purposes of this document we can define Inverse Optimal Control concretely as: given data of demonstrated behavior $\mathcal{D} = \{(\boldsymbol{\gamma}_i, \xi_i)\}_{i=1}^{N}$, where each demonstration ξ_i is created relative to context $\boldsymbol{\gamma}_i$, find parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ under which each trajectory ξ_i looks optimal with respect to the resulting problem given by Equation 1. Technically, we want to understand the properties of the resulting optimal control system, such as how well it performs in contexts that we haven't yet seen with respect to what we would have demonstrated in that context, but those questions dig more strongly into machine learning than we'd like to do in this document. We can say, however, that the framework developed below proceeds by first reducing the problem to a very large-scale form of multiclass classification, called Structured Prediction Lafferty et al. (2001); Tsochantaridis et al. (2004); Taskar et al. (2003, 2005). This form of machine learning is widely studied, and the algorithms we apply have strong associated out-of-sample generalization guarantees that will, indeed, answer some of these theoretical questions. That's one advantage to connecting the problem of Inverse Optimal Control to such a heavily studied field. Machine learning addresses interesting questions very relevant to designing optimal controllers from demonstration.

2.2 The final abstraction: optimal control as a multiclass classifier

There's just one final step to draw the connection between optimal control and multiclass classification, and it's mainly just a change of notation. Instead of using the control specific notation above, lets instead represent the context as a generic $\boldsymbol{x}$, the trajectory and set of trajectories as a $\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})$, and the cost function that we're optimizing as $\boldsymbol{w}^T \boldsymbol{F}(\boldsymbol{x}, \boldsymbol{y})$. The relationship between this new notation and the optimal control notation is

$$\boldsymbol{w} = \begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{bmatrix} \quad \text{and} \quad \boldsymbol{F}(\boldsymbol{x}, \boldsymbol{y}) = \begin{bmatrix} \boldsymbol{c}(\gamma, \xi) \\ \boldsymbol{\psi}(\gamma, \xi) \end{bmatrix} \tag{4}$$

$$\text{with} \quad \boldsymbol{w}^T \boldsymbol{F}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{t=1}^{T} \left(\sum_{i=1}^{k} \alpha_i c_t^{(i)}(\boldsymbol{s}_t, \boldsymbol{u}_t) \right) + \sum_{j=1}^{l} \beta_j \psi^{(j)}(\boldsymbol{s}_{T+1}).$$

Now the optimal control problem is

$$\boldsymbol{y}^*(\boldsymbol{x}) = \operatorname*{argmin}_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})} \boldsymbol{w}^T \boldsymbol{F}(\boldsymbol{x}, \boldsymbol{y}). \tag{5}$$

There's no difference between this expression and a general multiclass classifier. $\mathcal{Y}(\boldsymbol{x})$ is a set of classes, $\boldsymbol{x}$ is an input, and the function $\boldsymbol{w}^T \boldsymbol{F}(\boldsymbol{x}, \boldsymbol{y})$ parametrizes the score we place on how appropriate class $\boldsymbol{y}$ is for input $\boldsymbol{x}$. The classifier, itself, simply returns best scoring (minimum cost) class. Multiclass classification Bishop (2007) is a sub-field of machine learning that studies *learning* classification functions of this form from labeled data $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$. The learning problem, specifically, is to find a weight vector $\boldsymbol{w}$ that "best" fits the data for some notion of "best". For the control problem above, this problem translates to finding parameter vectors $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ under which the resulting optimal controller best mimics the demonstrated trajectories. Again, we've been vague about what we mean by "best", but that we'll define more concretely below through an incremental exploration of what we'd like in a good solution.

There's one caveat, though. We've described the problem of Inverse Optimal Control as a multiclass classification problem, but the number of classes $|\mathcal{Y}(\boldsymbol{x})|$ in the optimal control problem is huge. Each possible trajectory in a given context is a different class, and to "classify" a context we need to find the optimal trajectory using techniques from optimal control to optimize the objective in Equation 5 which is an abstraction of the problem in Equation 1. That means there are an infinite number of classes. Even if we have a tiny optimal control problem where we can discretize the state space, the number of trajectories will still be exponential in the horizon T.

Typically, in multiclass classification we consider just a handful of classes, maybe two (true or false, does $\boldsymbol{x}$ have a property or does it not?) or maybe on the order of 20 or 30 if we're classifying handwritten digits or letters, but as we'll see below the particular generalized framework of multiclass classification we consider here works well even for exponential or infinite class spaces. All we'll require is that the argmin optimization be computationally tractable.[1] The framework we describe subsequently to solve these very large-scale classification problems is called Maximum Margin Structured Classification (MMSC) Taskar et al. (2005); Ratliff (2009), a generalization of the Support Vector Machine (SVM).

3 Learning huge classifiers

Suppose we have a data set of examples $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$. How can we characterize good classifier behavior with respect to this data? The function $\boldsymbol{w}^T\boldsymbol{F}(\boldsymbol{x}, \boldsymbol{y})$ scores the cost of associating label $\boldsymbol{y}$ with input $\boldsymbol{x}$. For starters, then, since the classifier is going to report as its classification the lowest cost class $\boldsymbol{y}^*(\boldsymbol{x}) = \text{argmin}_{\boldsymbol{y}\in\mathcal{Y}(\boldsymbol{x})}\, \boldsymbol{w}^T\boldsymbol{F}(\boldsymbol{x}, \boldsymbol{y})$ for each example $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ we'd like the labeled class $\boldsymbol{y}_i$ to be assigned less cost than any other possible label $\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})$. As an equation, this means we want to find $\boldsymbol{w}$ such that

$$\boldsymbol{w}^T\boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}_i) \le \boldsymbol{w}^T\boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}) \quad \text{for each } i = 1, \ldots, N. \tag{6}$$

But, in ultimately, we really want more than that. Typically, we have some notion of how bad it is for the classifier to choose a given class $\boldsymbol{y}$ when it should have chosen the example $\boldsymbol{y}_i$. Abstractly, we can write that notion of loss in erroneously choosing $\boldsymbol{y}$ over $\boldsymbol{y}_i$ as a function $\mathcal{L}(\boldsymbol{y}_i, \boldsymbol{y}) = \mathcal{L}_i(\boldsymbol{y})$ (where the latter is just shorthand notation for the former). The main properties we require of this *loss function* is that it's everywhere positive and that the true label $\boldsymbol{y} = \boldsymbol{y}_i$ gets zero loss $\mathcal{L}(\boldsymbol{y}_i, \boldsymbol{y}_i) = 0$.

For instance, more concretely, for the optimal control problem (writing it in terms of trajectories ξ_i and ξ now), the loss function $\mathcal{L}(\xi_i, \xi)$ might measure the difference between where we should have been $\boldsymbol{s}_t^{(i)}$ at time t and where the trajectory in question actually places us $\boldsymbol{s}_t$:

$$\mathcal{L}(\xi_i, \xi) = \sum_{t=1}^{T+1} \|\boldsymbol{s}_t^{(i)} - \boldsymbol{s}_t\|^2. \tag{7}$$

3.1 Loss-augmentation

Given such a *loss function* $\mathcal{L}_i(\boldsymbol{y})$, we would really like to encode that loss information in the costs as well. If the loss function indicates that a given class $\boldsymbol{y}$ is really bad ($\mathcal{L}_i(\boldsymbol{y})$ is large), then intuitively, it's not enough that $\boldsymbol{w}^T\boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}) - \boldsymbol{w}^T\boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}_i) = \epsilon > 0$ where ϵ is some tiny positive value. We

[1] Technically, we actually need to be able to find the global minimum of the classification optimization for the theory to hold, but in practice algorithms that that find good local minima, such as nonlinear optimal control solvers, work well, too.

want the difference between the costs to be large. This class is *really bad*—we want it to cost a whole lot more than then example.

On the other hand, if $\mathcal{L}_i(\boldsymbol{y})$ is relatively small for a given $\boldsymbol{y}$, then we can say that $\boldsymbol{y}$'s almost correct, so it doesn't really matter as much even if we incorrectly classify $\boldsymbol{x}_i$ as $\boldsymbol{y}$ rather than $\boldsymbol{y}_i$. We'd like to be correct, but we don't need the gap $\boldsymbol{w}^T\boldsymbol{F}(\boldsymbol{x}_i,\boldsymbol{y}) - \boldsymbol{w}^T\boldsymbol{F}(\boldsymbol{x}_i,\boldsymbol{y}_i)$ to be very large. This notion of cost gap size is related to what we call the *margin*. We want the margin between a possible label $\boldsymbol{y}$ and the true label $\boldsymbol{y}_i$ to be large when the loss $\mathcal{L}_i(\boldsymbol{y})$ is large, but we're okay with a smaller margin when that loss $\mathcal{L}_i(\boldsymbol{y})$ is small. In equations, this margin requirement turns the inequalities of Equation 6 into modified set of inequalities of the form

$$\boldsymbol{w}^T\boldsymbol{F}(\boldsymbol{x}_i,\boldsymbol{y}_i) \le \boldsymbol{w}^T\boldsymbol{F}(\boldsymbol{x}_i,\boldsymbol{y}) - \mathcal{L}_i(\boldsymbol{y}) \quad \text{for each } i = 1,\ldots,N. \tag{8}$$

Rather than simply desiring that the cost of $\boldsymbol{y}_i$ be smaller than the cost of all other classes $\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})$, this equation says that it specifically has to be smaller by an amount $\mathcal{L}_i(\boldsymbol{y})$. When $\boldsymbol{y}$ is really bad, we want that the gap be huge. When $\boldsymbol{y}$ is pretty close to correct, it's okay if that gap is smaller.

This subtlety, called Loss Augmentation Taskar et al. (2005), gives the learning problem a *structured margin* that adapts to the class. When there are an exponential, or even infinite, number of classes this structured margin can be really important. Theoretically, this structured margin is crucial to the development of strong generalization guarantees Ratliff et al. (2007); Ratliff (2009).

3.2 Defining the optimization

Notice, though, that there's an ambiguity in the constraints of Equation 8. If $\boldsymbol{w}^T\boldsymbol{F}(\boldsymbol{x}_i,\boldsymbol{y}) - \boldsymbol{w}^T\boldsymbol{F}(\boldsymbol{x}_i,\boldsymbol{y}_i) = \epsilon$, some really tiny positive number, we can scale that ϵ up to be arbitrarily large just by multiplying $\boldsymbol{w}$ by some constant. So one solution to satisfy the constraints is to just scale up $\boldsymbol{w}$ until all positive gaps exceed the loss values $\mathcal{L}_i(\boldsymbol{y})$. Clearly, that's not the solution we're looking for because scaling $\boldsymbol{w}$ doesn't actually change the classifier $\boldsymbol{y}^*(\boldsymbol{x}) = \operatorname{argmin}_{\boldsymbol{y}\in\mathcal{Y}(\boldsymbol{x})} \boldsymbol{w}^T\boldsymbol{F}(\boldsymbol{x},\boldsymbol{y})$ (it just scales the objective that the classifier's optimizing, which doesn't affect the minimizer). The constraints by themselves aren't enough to fully specify the learning problem.

What we'd really like is to find the smallest weight vector $\boldsymbol{w}$ such that those constraints in Equation 8 hold. In other words, we'd like to minimize $\|\boldsymbol{w}\|^2$ subject to the constraints given in Equation 8. In practice, there can be noise in our data, and our features $\boldsymbol{F}(\boldsymbol{x},\boldsymbol{y})$ may not encode enough information to truly represent the perfect function, so it may not actually be possible to find a single weight vector $\boldsymbol{w}$ that satisfies all those constraints. Instead, we allow there to be constraint violations for a price by adding non-negative slack variables ζ_i that relax the constraints slightly. These considerations all lead to an optimization problem of the following form

$$\begin{aligned} &\min_{\boldsymbol{w}} \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{N} \zeta_i \\ &\text{s.t.} \quad \boldsymbol{w}^T \boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}_i) \le \boldsymbol{w}^T \boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}) - \mathcal{L}_i(\boldsymbol{y}) + \zeta_i \\ &\qquad \zeta_i \ge 0 \qquad \text{for each } i = 1, \ldots, N, \end{aligned} \tag{9}$$

where $\lambda > 0$ is a constant trade-off parameter. Notice that this problem has a convex objective and a bunch of linear equality and inequality constraints. It's, therefore, a very large convex optimization problem (indeed, a very large QP). We can't solve it because it's way to large right now, but at least we know it's convex. We'll need to further exploit structure in the problem to see how we can make it tractable.

3.3 Making the optimization tractable

The optimization problem described by Equation 9 now correctly defines what we want in a good classifier that well-represents our data, but it's still a very large optimization problem. For every example i, there's one constraint for each class. As discussed above, there could be an infinite number of classes, or at least an exponential number; that's far too many constraints to handle in practice.

3.3.1 A tractable number of constraints

A simple observation, though, reduces this exponential number of linear constraints down to just a handful of nonlinear constraints. There's actually a lot of redundancy in the constraints for a given example i. Suppose a particular $\boldsymbol{y}^* \in \mathcal{Y}(\boldsymbol{x}_i)$ minimizes the right hand side of the cost constraints in Equation 9 in the sense $\boldsymbol{y}^* = \operatorname{argmin}_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x}_i)} \boldsymbol{w}^T \boldsymbol{F}(\boldsymbol{x}, \boldsymbol{y}) - \mathcal{L}_i(\boldsymbol{y})$. Then if the single equation

$$\boldsymbol{w}^T \boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}_i) \le \boldsymbol{w}^T \boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}^*) - \mathcal{L}_i(\boldsymbol{y}^*) \tag{10}$$

holds for that label $\boldsymbol{y}^*$, it must also hold for all of those cost constraints in example i. Thus, it's perfectly valid to replace the exponential number of constraints for example i in Equation 9 with the single equation that specifies that the constraint must hold for the optimizer $\boldsymbol{y}^*$. That transforms the problem into the following:

$$\begin{aligned} &\min_{\boldsymbol{w}} \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{N} \zeta_i \\ &\text{s.t.} \quad \boldsymbol{w}^T \boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}_i) \le \min_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x}_i)} \left\{ \boldsymbol{w}^T \boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}) - \mathcal{L}_i(\boldsymbol{y}) \right\} + \zeta_i \\ &\qquad \zeta_i \ge 0 \qquad \text{for each } i = 1, \ldots, N. \end{aligned} \tag{11}$$

This version replaces the exponential number of linear constraints with a more tractable collection of N nonlinear constraints. For those familiar with convex optimization, you may notice that this optimization problem is still convex Boyd & Vandenberghe (2004). For those less familiar, the following manipulation will make that property more clear.

3.3.2 Placing the constraints into the objective

If we rearrange the constraints to place ζ_i alone on one side

$$\zeta_i \geq \boldsymbol{w}^T \boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}_i) - \min_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x}_i)} \left\{ \boldsymbol{w}^T \boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}) - \mathcal{L}_i(\boldsymbol{y}) \right\}, \tag{12}$$

we can leverage a nice property of this constraint. Consider the value of $\boldsymbol{w}^T \boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}) - \mathcal{L}_i(\boldsymbol{y})$ at $\boldsymbol{y} = \boldsymbol{y}_i$. The loss function is zero there $\mathcal{L}_i(\boldsymbol{y}_i) = 0$, so the expression just reduces to $\boldsymbol{w}^T \boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}_i)$. So it must be that

$$\min_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x}_i)} \left\{ \boldsymbol{w}^T \boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}) - \mathcal{L}_i(\boldsymbol{y}) \right\} \leq \boldsymbol{w}^T \boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}_i), \tag{13}$$

since the left hand side of that expression is minimizing over a set of values which includes the right hand side. Rearranging Equation 13 and combining it with Equation 12 show that

$$\zeta_i \geq \boldsymbol{w}^T \boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}_i) - \min_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x}_i)} \left\{ \boldsymbol{w}^T \boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}) - \mathcal{L}_i(\boldsymbol{y}) \right\} \geq 0. \tag{14}$$

In other words, the constraint $\zeta_i \geq 0$ is actually redundant. But more, since the objective is minimizing $\sum_i \zeta_i$, there's no value (with respect to this objective) in ζ_i being greater than the difference

$$r_i(\boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}_i) - \min_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x}_i)} \left\{ \boldsymbol{w}^T \boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}) - \mathcal{L}_i(\boldsymbol{y}) \right\}. \tag{15}$$

At the minimizer ζ_i will be exactly that difference. This means it's perfectly valid for us to entirely replace ζ_i in the objective with these difference values $r_i(\boldsymbol{w})$. Doing so forms the following objective

$$\begin{aligned} f(\boldsymbol{w}) &= \sum_{i=1}^{N} r_i(\boldsymbol{w}) + \frac{\lambda}{2}\|\boldsymbol{w}\|^2 \qquad (16) \\ &= \sum_{i=1}^{N} \left(\boldsymbol{w}^T \boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}_i) - \min_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x}_i)} \left\{ \boldsymbol{w}^T \boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}) - \mathcal{L}_i(\boldsymbol{y}) \right\} \right) + \frac{\lambda}{2}\|\boldsymbol{w}\|^2. \end{aligned}$$

This form of the problem is known as a *regularized risk function* Schölkopf & Smola. (2002); Bishop (2007), and as we'll show below, it's now in a form that we can easily optimize using any number of gradient-based methods. The sum of terms on the left is known as the *risk* function, which measures simply how well the hypothesis $\boldsymbol{w}$ fits the data. The last term is known as the *regularizer*, and its purpose is to regulate the size of the hypothesis $\boldsymbol{w}$ to ensure it doesn't grow too big as we attempt to fit it to the data.

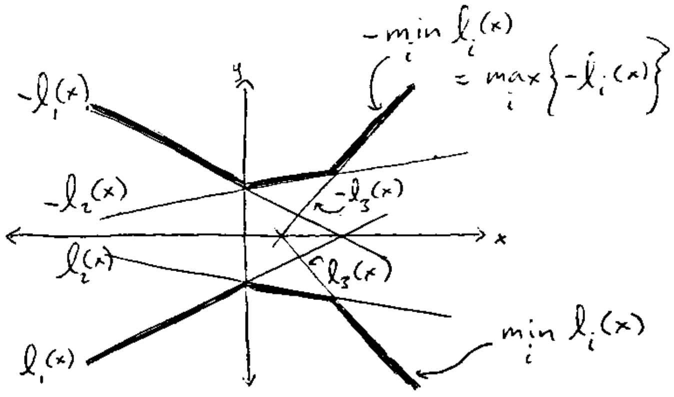

Figure 1: The negative of a min over linear functions $-\min_i l_i(\boldsymbol{x}_i)$ equals a max over the negatives of those linear functions $\max_i\{-l_i(\boldsymbol{x}_i)\}$. Moreover, since both a linear function and its negative are convex, this negative min over linear functions is convex.

3.4 Optimizing the regularized risk function

Our risk term in Equation 16 contains a min operator, which, on the surface seems hard to deal with. We'll show here that it's not.

3.4.1 Calculating the gradient

First, note that the difficult term is the negative of a min over linear functions. That's equivalent to a max over the negatives of those linear functions (see Figure 1). So to understand the behavior of this term, we really just need to understand the behavior of a term of the form $l(\boldsymbol{w}) = \max_i l_i(\boldsymbol{w})$, where each constituent function $l_i(\boldsymbol{w})$ is differentiable.

Suppose for a given $\boldsymbol{w}$ there's only a single maximizer $l_{i^*}(\boldsymbol{w})$ (i.e. $l_{i^*}(\boldsymbol{w})$ is larger than any other $l_j(\boldsymbol{w})$ for this $\boldsymbol{w}$). Then this function $l_{i^*}(\boldsymbol{w})$ forms the surface of $l(\boldsymbol{w})$ at this point $\boldsymbol{w}$. Clearly, then, the gradient of the function is just the gradient of $l_{i^*}(\boldsymbol{w})$ here. More concisely,

$$\nabla l(\boldsymbol{w}) = \nabla l_{i^*}(\boldsymbol{w}) \quad \text{where} \quad i^* = \operatorname*{argmax}_i l_i(\boldsymbol{w}). \tag{17}$$

In other words, to find the gradient, we just need to optimize the over the terms and then take the gradient there.

There's a slight subtlety here in that there are points, called *kinks*, in the function where the optimizer $\text{argmax}_i\, l_i(\boldsymbol{w})$ may not be uniquely defined, i.e. there may be a set $\mathcal{I}$ such that for each $i^* \in \mathcal{I}$, $l_{i^*}(\boldsymbol{w}) \geq l_i(\boldsymbol{w})$ for all i (which means the values $l_{i^*}(\boldsymbol{w})$ are all equal for $i^* \in \mathcal{I}$). It turns out, in those cases, the choice of which particular optimal $i^* \in \mathcal{I}$ you use doesn't really matter. The different gradients $\nabla l_{i^*}(\boldsymbol{w})$ for each $i^* \in \mathcal{I}$ are all called *sub*gradients (and, in fact, any convex combination of those subgradients are also subgradients); the field of convex optimization formalizes the use of subgradients for optimization Boyd & Vandenberghe (2004); Shor (1985). Of particular interest to us are *subgradient methods*, which, for our purposes, we can think of as effectively gradient descent methods where we use a fixed step size sequence rather than a line search at each iteration. We'll see below how these gradient based methods apply to our optimization problem.

The bottom line is that we can find the gradient (really subgradient) of the term $\min_{\boldsymbol{y}\in\mathcal{Y}(\boldsymbol{x}_i)} \boldsymbol{w}^T\boldsymbol{F}(\boldsymbol{x}_i,\boldsymbol{y}) - \mathcal{L}_i(\boldsymbol{y})$ by first optimizing to find $\boldsymbol{y}_A^*(\boldsymbol{x}_i) = \operatorname{argmin}_{\boldsymbol{y}\in\mathcal{Y}(\boldsymbol{x}_i)} \boldsymbol{w}^T\boldsymbol{F}(\boldsymbol{x}_i,\boldsymbol{y}) - \mathcal{L}_i(\boldsymbol{y})$ and then taking the gradient of $\boldsymbol{w}^T\boldsymbol{F}(\boldsymbol{x}_i, y_A^*(\boldsymbol{x}_i)) - \mathcal{L}_i(\boldsymbol{y}_A^*(\boldsymbol{x}_i))$. In other words, the gradient of this term is

$$\nabla_{\boldsymbol{w}} \min_{\boldsymbol{y}\in\mathcal{Y}(\boldsymbol{x}_i)} \Big\{ \boldsymbol{w}^T\boldsymbol{F}(\boldsymbol{x}_i,\boldsymbol{y}) - \mathcal{L}_i(\boldsymbol{y}) \Big\} = \boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}_A^*(\boldsymbol{x}_i)). \tag{18}$$

We denote this optimizer $\boldsymbol{y}_A^*$ using a subscript A because this optimization is almost the same optimization as the original classification problem $\boldsymbol{y}^* = \operatorname{argmin} \boldsymbol{w}^T\boldsymbol{F}(\boldsymbol{x},\boldsymbol{y})$, except it's loss-augmented. The A denotes the loss-augmentation. Below, we'll see how this loss-augmentation affects the problem in the case of optimal control; typically, when the structure of the loss-function $\mathcal{L}(\boldsymbol{y}_i,\boldsymbol{y})$ decomposes naturally across the structure of the problem, the resulting loss-augmented problem has the same form as the original problem, just with slightly modified costs. That means that optimizing this augmented problem is no harder than optimizing the original classification problem. See Section 4.2 for a specific example of how loss-augmentation affects the optimal control problem.

Finally, the gradients of the remaining terms of the regularized risk objective in Equation 16 are easy to calculate since $\boldsymbol{w}^T\boldsymbol{F}(\boldsymbol{x}_i,\boldsymbol{y}_i)$ is linear and $\frac{\lambda}{2}\|\boldsymbol{w}\|^2$ is quadratic:

$$\begin{aligned}
&\nabla \boldsymbol{w}^T\boldsymbol{F}(\boldsymbol{x}_i,\boldsymbol{y}_i) = \boldsymbol{F}(\boldsymbol{x}_i,\boldsymbol{y}_i)\\
&\nabla \frac{\lambda}{2}\|\boldsymbol{w}\|^2 = \lambda\boldsymbol{w}.
\end{aligned}$$

Now we're fully equipped to calculate the gradient of the regularized risk function in Equation 16:

$$\begin{aligned}
\nabla f(\boldsymbol{w}) &= \nabla\left[\sum_{i=1}^N \left(\boldsymbol{w}^T\boldsymbol{F}(\boldsymbol{x}_i,\boldsymbol{y}_i) - \min_{\boldsymbol{y}\in\mathcal{Y}(\boldsymbol{x}_i)} \big\{\boldsymbol{w}^T\boldsymbol{F}(\boldsymbol{x}_i,\boldsymbol{y}) - \mathcal{L}_i(\boldsymbol{y})\big\}\right) + \frac{\lambda}{2}\|\boldsymbol{w}\|^2\right]\\
&= \sum_{i=1}^N \Big(\boldsymbol{F}(\boldsymbol{x}_i,\boldsymbol{y}_i) - \boldsymbol{F}(\boldsymbol{x}_i,\boldsymbol{y}_A^*(\boldsymbol{x}_i)) \Big) + \lambda\boldsymbol{w}, \qquad (19)\\
&\text{where } \ \boldsymbol{y}_A^*(\boldsymbol{x}_i) = \operatorname*{argmin}_{\boldsymbol{y}\in\mathcal{Y}(\boldsymbol{x}_i)} \boldsymbol{w}^T\boldsymbol{F}(\boldsymbol{x}_i,\boldsymbol{y}).
\end{aligned}$$

As a final note, the above analysis also indicates that this objective is convex. The regularizer is just a quadratic function, the term $\boldsymbol{w}^T\boldsymbol{F}(\boldsymbol{x}_i,\boldsymbol{y}_i)$ is linear, and the remaining term is a negative min over linear functions, which we saw above is just a max over the negatives of those linear functions, which is again convex. Thus, this formulation of the learning problem fully specifies the solution. There are no local optima; starting from any $\boldsymbol{w}_0$, we can simply following the gradients downhill until the procedure converges and that's the unique globally optimal hypothesis $\boldsymbol{w}^*$.

3.4.2 Gradient-based optimization

Now that we have a gradient, since we've placed all the constraints up into the objective, all we need to do now is run gradient descent to optimize the unconstrained objective. Algorithm 1 lists this gradient descent algorithm with one modification. Since this objective function is a sum of terms, it's often a better use of gradient information to iterate over the terms and for *each* term i take a step in the direction of the negative gradient of just that term. For *incremental* subgradient method (also known as a stochastic gradient method), we can fold $1/N$th of the regularizer into each term, which leads to a regularization update of the form

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta_t \frac{\lambda}{N}\boldsymbol{w} = (1 - \frac{\eta_t \lambda}{N})\boldsymbol{w}, \tag{20}$$

where η_t is the step size. There are theoretical constraints on the step size η_t and the regularization parameter λ (see Boyd & Vandenberghe (2004)) that we won't discuss much here besides to say that η_t is often a decreasing sequence that diverges to infinity when summed up such as $\eta_t = \frac{\eta}{t+c}$ for some constant c or $\eta_t = \frac{\eta}{\sqrt{t+c}}$. The divergence ensures that we don't stop prematurely just because we're taking steps that are too small (e.g. $\eta_t = \eta^t$ is a geometric series that converges to $\frac{1}{1-\eta} - 1 = \frac{\eta}{1-\eta}$, so if the norm of each gradient is bounded above by some value b, the hypothesis won't be able to travel more than $b(\frac{\eta}{1-\eta})$ through the hypothesis space). Additionally, the product $\lambda\eta_t$ is typically smaller than 1. In combination, that means the rule in Equation 20 simply suggests that after each risk term gradient step, we should shrink the weights by a factor $1 - \eta_t\lambda/N$.

Notice that the main update of this algorithm

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta_t \Big(\boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}_A^*) - \boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}_i) \Big) \tag{21}$$

has a very intuitive interpretation. Consider the behavior of the algorithm mid training process. Before the algorithm has converged, the optimal class that the optimizer returns is $\boldsymbol{y}_A^*$, but really $\boldsymbol{y}_i$ should be the optimum. As a step toward fixing that discrepancy, this learning algorithm increases the cost of features characteristic of $\boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}_A^*)$ by adding it to the weight vector, and decreases the cost of features characteristic of $\boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}_i)$ by subtracting it from the weight vector. After this learning step, (the wrong) class $\boldsymbol{y}_A^*$ will now cost a little more, and the (correct) class $\boldsymbol{y}_i$ will cost a little less. The algorithm has made a small incremental step in the right direction. As the optimization continues,

Algorithm 1 Subgradient Method for Structured Classification

```
**Input:** Data $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$, step-size sequence $\{\eta_t\}_{i=1}^\infty$, initial hypothesis $\boldsymbol{w}_0$.
Initialize $\boldsymbol{w} \leftarrow \boldsymbol{w}_0$
while not converged do
  for $i = 1, \ldots, N$ do
    Solve $\boldsymbol{y}_A^* = \text{argmin}_{y \in \mathcal{Y}(\boldsymbol{x}_i)} \{\boldsymbol{w}^T \boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}) - \mathcal{L}_i(\boldsymbol{y})\}$.
    Update $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta_t \left( \boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}_A^*) - \boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}_i) \right)$.
    Shrink $\boldsymbol{w} \leftarrow \left(1 - \frac{\lambda \eta_t}{N}\right) \boldsymbol{w}$.
  end for
end while
```

the learning procedure tries to find a hypothesis $\boldsymbol{w}$ under which, to the best of its ability, each $\boldsymbol{y}_i$ looks as close to optimal as possible. Mathematically, we can say the algorithm attempts to find a (reasonably sized) $\boldsymbol{w}$ for which

$$\sum_{i=1}^N \mathcal{L}\left(\boldsymbol{y}_i,\ \boldsymbol{y}^*(\boldsymbol{x}_i; \boldsymbol{w}) \right) \tag{22}$$

$$\text{with} \quad \boldsymbol{y}^*(\boldsymbol{x}_i; \boldsymbol{w}) = \operatorname*{argmin}_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x}_i)} \boldsymbol{w}^T \boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}) \quad \text{for each} \quad i$$

is small. But since optimizing this total loss directly would be hard (full of local minima, in particular), we instead choose to optimize the convex regularized risk function in Equation 16, which, Algorithm 1 shows, is actually relatively easy.

4 Algorithm intuition for optimal control

Much of the above discussion has been very abstract. We started with the broad and poorly specified problem of trying to tune the cost function of an optimal control problem, translated that in to the more specific and well-formed problem of tuning the weights of constituent cost function terms, and then entirely shifted notation to demonstrate that that problem is really a very large scale classification algorithm that can formalized leveraging techniques from Machine Learning. We ultimately derived a simple gradient-based algorithm for this general form of large scale classification (Maximum Margin Structured Classification), but how does that algorithm translate back to the optimal control problem. This section now reverses the process of abstraction and translates our final learning algorithm back to the notation of optimal control so we can see how the training steps, and in particular the loss-augmented inner loop optimization, play out to solve the Inverse Optimal Control problem.

The application of structured classification to Optimal Control results in a very general class of algorithms for modeling and solving Inverse Optimal Control. In the context of behavior learning, one may view it as a form of Imitation

Learning: a trainer demonstrates the desired behavior, and this algorithm learns an optimal controller to predict that behavior in new contexts.

4.1 The feature function: cumulative feature vectors

Thinking back to Section 2.1, the primary contribution to the feature function $\boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}_i)$ is the vector of cost terms $\boldsymbol{c}(\boldsymbol{\gamma}, \xi)$, which we can write more explicitly as

$$\boldsymbol{c}(\boldsymbol{\gamma}, \xi) = T \begin{bmatrix} \frac{1}{T}\sum_{t=1}^{T} c_t^{(1)}(\boldsymbol{\gamma}, \boldsymbol{s}_t, \boldsymbol{u}_t) \\ \frac{1}{T}\sum_{t=1}^{T} c_t^{(2)}(\boldsymbol{\gamma}, \boldsymbol{s}_t, \boldsymbol{u}_t) \\ \vdots \\ \frac{1}{T}\sum_{t=1}^{T} c_t^{(k)}(\boldsymbol{\gamma}, \boldsymbol{s}_t, \boldsymbol{u}_t) \end{bmatrix}. \tag{23}$$

In other words, it's (proportional to) a vector consisting of the average values of each constituent cost term $c_t^{(i)}(\boldsymbol{\gamma}, \boldsymbol{s}_t, \boldsymbol{u}_t)$ across the trajectory in question $\xi = \{(\boldsymbol{s}_1, \boldsymbol{u}_1), \ldots, (\boldsymbol{s}_T, \boldsymbol{u}_T), \boldsymbol{s}_{T+1}\}$. This vector represents what cost-features trajectory ξ *sees* on average as it traverses its route. Each different trajectory sees on average a different distribution of features; this vector $\boldsymbol{c}(\boldsymbol{\gamma}, \xi)$ characterizes what each trajectory sees on average allowing us to quantify the differences between trajectories.

Suppose, for instance, cost feature $c_t^{(1)}$ measures the intensity of the color yellow directly below the robot, feature $c_t^{(2)}$ measures the intensity of the color green below the robot, and suppose the example ξ_i demonstrates that the robot should follow the yellow line painted atop a green surface. Then on average along the example trajectory $c_t^{(1)}$, which measures the intensity of yellow, will be high along the demonstration ξ_i, and $c_t^{(2)}$, which measures the intensity of green, on average will be low. However, if we haven't yet tuned the cost function properly, the actual (loss-augmented) optimal trajectory returned by the optimal control algorithm may be some trajectory ξ_A^* that doesn't yet follow the line properly. Along that (erroneous) trajectory, the robot won't be seeing the right distribution of features. On average the value of the yellow feature $c_t^{(1)}$ will be much lower than it should (it's not following the line well), and the value of the green feature $c_t^{(2)}$ will be much larger than it should (it's traveling too much along raw green patches). These discrepancies are encoded in the difference between $\boldsymbol{c}(\boldsymbol{\gamma}, \xi^*)$ and $\boldsymbol{c}(\boldsymbol{\gamma}, \xi_i)$, which in part makes up the difference vector $\boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}_A^*) - \boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}_i)$.

Additionally, during the update, the algorithm adds a multiple of $\boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}_A^*)$ to the weight vector and subtracts a multiple of $\boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{y}_i)$. In terms of the yellow and green features of the optimal control problem, this update would increase the weight on the green feature (it's seen much more in ξ_A^* than it is in the demonstration ξ_i), and decrease the weight on the yellow feature (it's seen much more in the demonstration ξ_i than it is in ξ_A^*). This update makes green features now more costly and yellow features less costly, so the next time we run the optimal control algorithm, the system will tend to favor yellow features over green features more than it used to.

In general, if the current loss-augmented optimal trajectory ξ_A^* sees more of any particular feature than the example ξ_i does, the algorithm will increase the cost of that because the behavior of the example indicates that that feature should be higher cost than it currently is; and if ξ_A^* tends to see less of a particular feature than ξ_i, the algorithm decreases the cost of that feature because the behavior of example indicates the feature is less costly than it currently is.

Another quick and intuitive example is a tree feature. Suppose we have a feature that indicates whether the robot is currently plowing through a tree. Clearly, the demonstration is going to avoid trees, so on average it'll see none of that feature at all. If our current hypothesis accidentally has the robot plowing through a tree, the difference between the average value of that feature for ξ_A^* and ξ_i will be largely positive, so the algorithm will greatly increase the cost of trees. The next time around, hopefully, the optimal controller will be less likely to plow through trees because we've increased their cost.

4.2 The loss-augmented optimal control problem

We've been thus far vague about how you might actually go about optimizing the loss-augmented function. Here we'll show that if the loss function structurally decomposes in the same way as the cost function, we can usually use the same optimizer that we used on the original problem to optimize this loss-augmented problem.

This notion of structural decomposition is vague right now, but lets take a look specifically at the optimal control problem. Equation 7 gives one example of a loss function applicable to the optimal control problem. We repeat it here for convenience:

$$\mathcal{L}(\xi_i, \xi) = \sum_{t=1}^{T+1} \|\boldsymbol{s}_t^{(i)} - \boldsymbol{s}_t\|^2. \tag{24}$$

Notice that it decomposes as a sum over time, just as the optimal control cost function does. Above, in Section 2.1, we incrementally converted our notation from the optimal control problem to a more abstract representation that emphasized its relationship to large-scale multiclass classification, ultimately leading to $\boldsymbol{y}^*(\boldsymbol{x}; \boldsymbol{w}) = \operatorname{argmin}_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})} \boldsymbol{w}^T \boldsymbol{F}(\boldsymbol{x}, \boldsymbol{y})$ as a generic representation of optimal control focusing on the relationship between the parameter weights $\boldsymbol{w}$ and the features $\boldsymbol{F}(\boldsymbol{x}, \boldsymbol{y})$. Ultimately, though, in the inner loop of the learning algorithm, we really want to solve the loss-augmented variant of this problem

$$\boldsymbol{y}_A^*(\boldsymbol{x}; \boldsymbol{w}) = \operatorname*{argmin}_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})} \left\{ \boldsymbol{w}^T \boldsymbol{F}(\boldsymbol{x}, \boldsymbol{y}) - \mathcal{L}(\boldsymbol{y}_i, \boldsymbol{y}) \right\}. \tag{25}$$

The following block of equations converts this abstraction back to control-specific notation to see how the augmentation affects the optimal control problem. In this manipulation, we remove the example index i since it notationally clashes with the sum over cost features and instead denote states along the demonstrated trajectory by $\widetilde{\boldsymbol{s}}_t$:

$$
\begin{aligned}
&\min_{\boldsymbol{y}\in\mathcal{Y}(\boldsymbol{x})} \left\{\boldsymbol{w}^T\boldsymbol{F}(\boldsymbol{x},\boldsymbol{y}) - \mathcal{L}(\boldsymbol{y}_i,\boldsymbol{y})\right\} \\
&\quad = \min_{\xi\in\Xi(\boldsymbol{\gamma})} \left[\sum_{i=1}^{k}\alpha_i\left(\sum_{t=1}^{T} c_t^{(i)}(\boldsymbol{\gamma},\boldsymbol{s}_t,\boldsymbol{u}_t)\right) + \sum_{j=1}^{l}\beta_j\boldsymbol{\psi}^{(j)}(\boldsymbol{\gamma},\boldsymbol{s}_{T+1})\right] - \sum_{t=1}^{T+1}\|\widetilde{\boldsymbol{s}}_t - \boldsymbol{s}_t\|^2 \\
&\quad = \min_{\xi\in\Xi(\boldsymbol{\gamma})} \left[\sum_{t=1}^{T}\left(\sum_{i=1}^{k}\alpha_i c_t^{(i)}(\boldsymbol{\gamma},\boldsymbol{s}_t,\boldsymbol{u}_t)\right) + \sum_{j=1}^{l}\beta_j\boldsymbol{\psi}^{(j)}(\boldsymbol{\gamma},\boldsymbol{s}_{T+1})\right] - \left(\sum_{t=1}^{T}\|\widetilde{\boldsymbol{s}}_t - \boldsymbol{s}_t\|^2\right) - \|\widetilde{\boldsymbol{s}}_{T+1} - \boldsymbol{s}_{T+1}\|^2 \\
&\quad = \min_{\xi\in\Xi(\boldsymbol{\gamma})} \sum_{t=1}^{T}\left(\sum_{i=1}^{k}\alpha_i c_t^{(i)}(\boldsymbol{\gamma},\boldsymbol{s}_t,\boldsymbol{u}_t) - \|\widetilde{\boldsymbol{s}}_t - \boldsymbol{s}_t\|^2\right) + \left(\sum_{j=1}^{l}\beta_j\boldsymbol{\psi}^{(j)}(\boldsymbol{\gamma},\boldsymbol{s}_{T+1}) - \|\widetilde{\boldsymbol{s}}_{T+1} - \boldsymbol{s}_{T+1}\|^2\right).
\end{aligned}
$$

In other words, this loss-augmented optimization is equivalent to

$$
\begin{aligned}
&\min_{\boldsymbol{u}_{1:T},\boldsymbol{s}_{1:T+1}} \sum_{t=1}^{T}\widetilde{c}_t(\boldsymbol{s}_t,\boldsymbol{u}_t) + \widetilde{\psi}(\boldsymbol{s}_{T+1}) \\
&\text{s.t.}\quad \boldsymbol{s}_{t+1} = \boldsymbol{f}(\boldsymbol{s}_t,\boldsymbol{u}_t),
\end{aligned}
\tag{26}
$$

where

$$
\widetilde{c}_t(\boldsymbol{s}_t,\boldsymbol{u}_t) = \underbrace{\sum_{i=1}^{k}\alpha_i c_t^{(i)}(\boldsymbol{\gamma},\boldsymbol{s}_t,\boldsymbol{u}_t)}_{c_t(\boldsymbol{s}_t,\boldsymbol{u}_t)} - \|\widetilde{\boldsymbol{s}}_t - \boldsymbol{s}_t\|^2
$$

and

$$
\widetilde{\psi}(\boldsymbol{s}_{T+1}) = \underbrace{\sum_{j=1}^{l}\beta_j\boldsymbol{\psi}^{(j)}(\boldsymbol{\gamma},\boldsymbol{s}_{T+1})}_{\psi(\boldsymbol{s}_{T+1})} - \|\widetilde{\boldsymbol{s}}_{T+1} - \boldsymbol{s}_{T+1}\|^2.
$$

Because the loss-function decomposes as a sum over time in the same way the original cost function did, the loss-augmented optimal control problem takes exactly the same form as the original optimal control problem. The only difference is that each cost term has an added (negative) term $-\|\widetilde{\boldsymbol{s}}_t - \boldsymbol{s}_t\|^2$. These new terms actually decrease the cost of states designated as "bad" by the loss function. That sounds backward, but it means that during training it's extra easy for the optimal policy to make mistakes, so the outer learning loop needs to try that much harder to find a policy under this handicap. When we remove the loss-augmentation once the learning algorithm has converged, the optimal control algorithm will perform extra well now on the examples. Moreover, theoretically, this loss-augmentation introduces a construct that makes it possible to prove strong generalization guarantees for this approach Ratliff et al.

(2006, 2007). Both theoretically, and practically, using this loss-augmentation handicap during training improves the final performance of the learned optimal controller, especially in contexts we didn't see in the training data.

5 Some closing thoughts

This document shows just a small sample of Inverse Optimal Control, and in particular it focuses on one specific formalization of Inverse Optimal Control as Maximum Margin Structured Classification Ratliff et al. (2006, 2009, 2007). That view clarifies the relationship between Inverse Optimal Control and Machine Learning, and demonstrates that this form of behavior learning, which is a form of Imitation Learning, is a convex problem fundamentally distinguishing it from more difficult forms of behavior learning such as Reinforcement Learning, but there are a number of alternative Inverse Optimal Control formulations and algorithms in the literature (also known as Inverse Reinforcement Learning). Ratliff (2009) reviews some of the history of the problem, and some interesting papers for further reading include the work of Ng & Russell (2000); Abbeel & Ng (2004); Ratliff et al. (2009); Silver et al. (2010); Ziebart et al. (2008); Dvijotham & Todorov (2010),

One commonality among a number of these algorithms is that the learning procedure ends up forming an outer loop optimization around an inner loop optimal control solver or planner. For each hypothesized cost function, we run the inner loop optimal control algorithm and rate the resulting behavior relative to what we would have liked the behavior to be. Given that information, we update the policy, and then iterate. Often it takes a number of iterations to converge, but we the engineers, fortunately, aren't doing this by hand—it's all automated.

References

Abbeel, P. and Ng, A. Y. Apprenticeship learning via inverse reinforcement learning. In *In ICML 04: Proceedings of the twenty-first international conference on Machine learning*, 2004.

Bishop, Christopher M. *Pattern Recognition and Machine Learning.* Springer, 2007.

Boyd, Stephen and Vandenberghe, Lieven. *Convex Optimization.* Cambridge University Press, 2004.

Dvijotham, K. and Todorov, E. Inverse optimal control with linearly-solvable mdps. In *In International Conference on Machine Learning*, 2010.

Kalman, R. When is a linear control system optimal? *Transactions ASME, Journal Basic Engineering.*, 86:51–60, 1964.

Lafferty, J., McCallum, A., and Pereira, F. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*, pp. 282–289, 2001.

Ng, A. Y. and Russell, S. Algorithms for inverse reinforcement learning. In *In Proc. 17th International Conf. on Machine Learning*, 2000.

Ratliff, Nathan. *Learning to Search: Structured Prediction Techniques for Imitation Learning*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, May 2009.

Ratliff, Nathan, Bagnell, J. Andrew (Drew), and Zinkevich, Martin. Maximum margin planning. In *International Conference on Machine Learning*, July 2006.

Ratliff, Nathan, Bagnell, J. Andrew (Drew), and Zinkevich, Martin. (online) subgradient methods for structured prediction. In *Eleventh International Conference on Artificial Intelligence and Statistics (AIStats)*, March 2007.

Ratliff, Nathan, Silver, David, and Bagnell, J. Andrew (Drew). Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, 27(1): 25–53, July 2009.

Schölkopf, Bernhard and Smola., Alex J. *Learning with Kernels*. MIT Press, 2002.

Shor, N. Z. *Minimization Methods for Non-differentiable Functions*. Springer-Verlag, 1985.

Silver, David, Bagnell, J. Andrew (Drew), and Stentz, Anthony (Tony). Learning from demonstration for autonomous navigation in complex unstructured terrain. *International Journal of Robotics Research*, 29(12):1565 – 1592, October 2010.

Taskar, B., Guestrin, C., and Koller, D. Max-margin markov networks. In *Neural Information Processing Systems Conference (NIPS03)*, Vancouver, Canada, December 2003.

Taskar, B., Chatalbashev, V., Koller, D., and Guestrin, C. Learning structured prediction models: A large margin approach. In *Twenty Second International Conference on Machine Learning (ICML05)*, Bonn, Germany, August 2005.

Tsochantaridis, I., Hofmann, T., Joachims, T., and Altun, Y. Support vector machine learning for interdependent and structured output spaces. In *International Conference on Machine Learning (ICML)*, pp. 104–112, 2004.

Ziebart, Brian D., Maas, Andrew, Bagnell, J. Andrew (Drew), and Dey, Anind. Maximum entropy inverse reinforcement learning. In *Proceeding of AAAI 2008*, July 2008.