6
An Invitation to Imitation

Introduction

We take a detour now to study a conceptually simpler problem: that of *imitation learning*. Imitation learning is the study of algorithms that improve performance in making decisions by observing demonstrations from a teacher. Consider, for instance, Figure 6.0.1, which shows a human expert tele-operating a walking robot by commanding its footstep motions. Such motions and the decisions behind them are complex and difficult to encode in simple, manually programmed rules. While demonstrating a desired behavior may be easy, designing a system that behaves this way is often difficult, time consuming, and ultimately expensive. Machine learning promises to enable "programming by demonstration" for developing high-performance robotic systems.

Figure 6.0.1: Human expert demonstration to train a walking robot to cross very rough terrain. *Learning to Search (LEARCH)* [Zucker et al., 2011, Ratliff, 2009] attempts to make a footstep planner mimic the human pilot's choices. Imitation learning is the study of algorithms that improve decision making through data collected by observing an expert – often, but not always a person who can accomplish a task that is hard to hand-program.

Learning Behavior Without Generalization

Many of the references in imitation learning focus on learning fixed trajectories, or on controllers to achieve such trajectories in the presence of disturbances. (See a detailed discussion in [Argall et al., 2009, Osa et al., 2018].) Such work – including the foundational [Atkeson and Schaal, 1997] and the stunning helicopter acrobatics of [Coates et al., 2009] – vividly dramatizes the remarkable power of human demonstration. However, these approaches are limited in their ability to generalize to new circumstances. Our focus here is on strategies that can generalize to unfamiliar settings and base decisions on perceptual feedback. It is important to appreciate, however, that the boundary between trajectory learning approaches and general imitation learning is not clear. Atkeson [Atkeson and Morimoto, 2003], and others, notably [Safonova and Hodgins, 2007, Mülling et al., 2013], show that a library of trajectories can indeed be made to generalize very broadly through clever arbitration and blending.

Imitation ≠ Supervised Learning – The Distinctions

Unfortunately, many approaches that utilize the classical tools of supervised learning fail to meet the needs of imitation learning. We must address two critical departures from classical supervised learning to enable effective imitation learning.

Perhaps foremost, classical supervised machine learning exists in a vacuum. Predictions made by these algorithms are explicitly assumed to **have no effect** on the world in which they operate. We will consider the problems that result from ignoring the effect of actions that influence the world and highlight simple "reduction-based" approaches that mitigate these problems both in theory and in practice.

Second, robotic systems are typically built atop sophisticated planning algorithms that efficiently reason far into the future. Ignoring these planning algorithms in lieu of a reactive learning approach often leads to poor, myopic performance. While planners have demonstrated dramatic success in applications ranging from legged locomotion to outdoor unstructured navigation, such algorithms rely on fully specified cost functions that map sensor readings and environment models to a scalar cost. These cost functions are usually manually designed and hand programmed, which is difficult and time-consuming. Recently, a set of techniques for learning these functions from human demonstration by applying an Inverse Optimal Control (IOC) approach to find a cost function for which planned behavior mimics an expert's demonstration have been shown to be effective and efficient. These approaches shed new light on the intimate connections between probabilistic inference and optimal control. [1]

These two points are taken up in turn in the next two major sections.

6.1 *Cascading Errors and Imitation Learning*

Dean Pomerleau's work [2] on learning autonomous driving is the seminal work in the field of imitation learning. Moreover, it gets right to the heart of the differences between imitation learning and classical supervised learning. Figure 6.1.1 demonstrates the setup of Pomerleau's experiments on learning

[1] We prefer the older, more widely used, terminology *Inverse Optimal Control* as opposed to *Inverse Reinforcement Learning* (IRL) throughout. The central premise of research in inverse optimal control approaches to imitation learning is that the policy to be learned by demonstration can be thought of as a near-optimal policy for some plant with an unknown reward function. In Reinforcement Learning, by contrast, the plant itself is viewed as unknown. Thus we are typically solving the inverse problem of optimal control, but not of the inverse of reinforcement learning, rendering the phrasing IRL somewhat misleading. Moreover, it's valuable to connect to the original literature in control theory dating back to Kalman's [Kalman, 1964] foundational work.

[2] D. Pomerleau. ALVINN: An Autonomous Land Vehicle in a Neural Network. In *Advances in Neural Information Processing Systems (NIPS)*, 1989

to drive the NavLab vehicle by using a neural network to map camera images to steering angles. Pomerleau developed this procedure by driving the car and collecting pairs of coarse camera images and steering angles. He then trained a simple neural network in real time to take new images and predict the resulting steering angle. [3]

[3] The Pomerleau works truly hold up for today's reader both for their impact on autonomous vehicles and their deep insight into the key differences between supervised and imitation learning.

Figure 6.1.1: Pomerleau's Autonomous Land Vehicle in a Neural Network system at work driving the Carnegie Mellon NavLab vehicle. Used with permission.

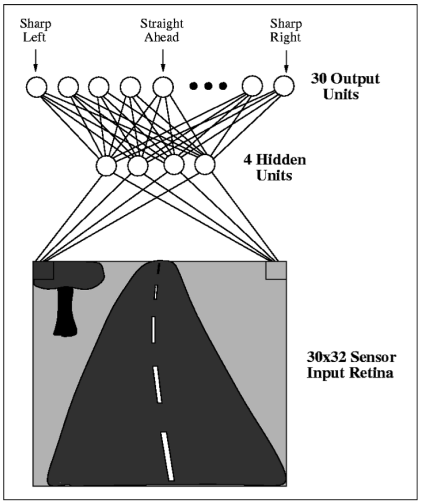

Figure 6.1.2: A schematic of Pomerleau's ALVINN driving system. The approach used a small neural network to map coarse camera images into a disretized set of steering angles. Image used with permission.

Consider a smaller, simplified version of the problem – learning to drive a car in a video game by performing a direct mapping from screen shots to steering angles. Figure 6.1.3 illustrates the classic supervised learning approach to learning such a mapping. [4]

Unfortunately, in this instance – as is quite common in practice – the approach fails disastrously and the learned controller quickly drives off the road. Let's consider what can go wrong. Of course, the learning problem

[4] Stephane Ross's results [Ross et al., 2011b, Ross, 2010a,b] applying such a procedure using linear regression on a simplified version of the screen image can be seen at Supervised Tux.

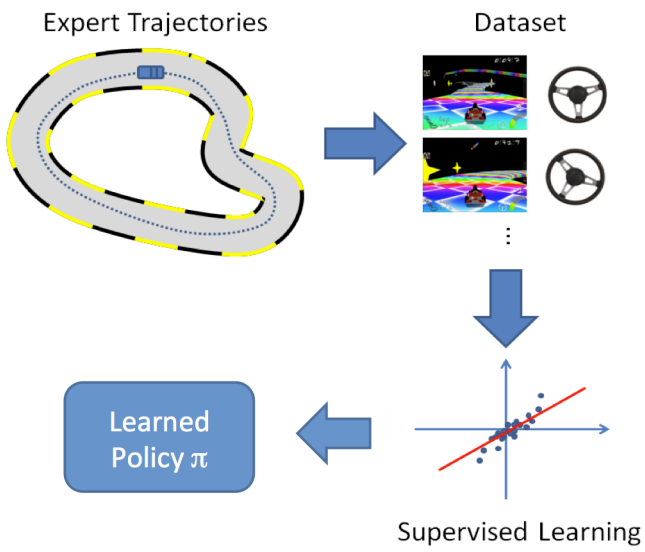

Figure 6.1.3: A sketch of the problem of learning to drive a video game simulation. A person drives the car around the course and collects data. That dataset consisting of images and associated steering angles is fed to a classic supervised learning algorithm, e.g., linear regression. The resulting policy π is used to drive the vehicle. Hilarity ensues.

may simply be too difficult. Perhaps we simply can't find a classifier or regressor that predicts the driver's steering decisions with small error. Perhaps a linear predictor is a bad choice for this problem; a richer hypothesis class might be more useful. That turns out not to be the case – a linear predictor is perfectly adequate for the task.

We could simply be *overfitting* – perhaps our training data set is too small to produce a good solution, which can lead to poor test performance. Avoiding overfitting has long been one of the central concerns in the study of learning theory[Shalev-Shwartz and Ben-David, 2014]. However, hold-out errors [5] are quite close to training errors in this example. Moreover, the learned policy[6] fails to perform well even with a very large set of training data.

What goes wrong? In a nutshell, learning errors *cascade* in imitation learning but are independent in supervised learning. Consider, for instance, a discrete version of the problem that only predicts "steer left" or "steer right". Inevitably, our learning algorithm will make some error – let's say with small probability ϵ for a good learner – and steer differently than a human driver would. At that point, the car will no longer be driving down the center of the road and the resulting images will look qualitatively different then the bulk of those used for training. Imitation learning has difficulty with this situation. The learner has never encountered these images before. Since learners can only attempt to do well in expectation over a distribution of familiar examples, an unusual image may incur further error, often with a higher probability.

As a result, the controller driving the simulation will steer the car close to the edge of the road – a very rare occurrence in training – and the resulting decision will likely be quite poor. Often, the learned controller will drive off the road, failing completely at the task. [7]

More formally, we can consider an imitation learning problem of T sequential decisions [Ross et al., 2011a]. If we learn a classifier making ϵ errors in predicting a driver's decisions in expectation over the distribution of examples induced by the teacher, we would hope to make $T\epsilon$ mistakes over the

[5] One can measure and control overfitting by considering the performance of a learned predictor on data that is "held-out": that is, data not available to the learning algorithm to train its predictor.

[6] We use *policy* here to refer to any learned predictor that maps features to actions. For discrete actions, this is simply a classifier. The terminology is common to optimal control and reinforcement learning, but is sometimes off-putting for roboticists and experts in supervised learning.

[7] Pomerleau's techniques for addressing these issues are particularly instructive. These include synthetic data generation, the use of online learning, and the emphasis on hard examples. This approach effectively manages *covariate shifts* similar to those caused when a learner influences its own test distribution. [Bagnell, 2005].

sequence of decisions. Unfortunately, an early error may compound into a long sequence of mistakes. As a result, the best we can hope for is $O(T^2\epsilon)$ mistakes[Ross and Bagnell, 2010]. [8] From a statistical point of view, our training and test data sets are not drawn from the same distribution and thus the supervised learning assumption of independent and identically distributed *(i.i.d.)* data is badly violated.

A natural suggestion for solving this problem is to collect data for all possible road conditions or over all images we may see. Unfortunately, it's difficult to obtain data for all possible inputs – the set of potential images is very large. Worse, no learner in our hypothesis class may be capable of handling all possible inputs. Assuming *realizability* – the "true" target function in our class– is generally far too strict, and algorithms that require this generally perform poorly. [Shalev-Shwartz and Ben-David, 2014] Instead, in machine learning we hope that there is a function in our hypothesis class that can work well *on average* over the actual distribution of training data that we encounter. [9]

[8] It's simplest to imagine a fixed time horizon. This fixed T can be replaced in analysis by notions of *mixing time*, *discount factor*, or a notion of how long any one mistake can propagate. It's therefore useful to consider T as representing an appropriate notion of the effective time horizon of the problem, not the actual number of decisions to be made.

[9] This point represents a general tension between the techniques of analysis in decision making and control – where one [Ljung, 1978] often requires a model or a controller to be *uniformly* good for all possible inputs, versus the paradigm of learning and statistics where it is recognized that this is not possible in high dimensional problems. In control, the focus is on ensuring good *expected* or average performance over the distribution of examples that actually occur. This mismatch lies at the heart of many of the difficulties of marrying learning and control. The interactive method discussed here – and no-regret learning in general – may serve as the bridge between these approaches.

A Simple Fix

If training data is plentiful and the time horizon is fixed and short, the compounding of errors is easily addressed. To proceed, we can train a policy for each of the T steps. The first policy is simply trained in normal supervised learning fashion by collecting data: the camera image and the person's steering angle at the initial decision. We train the next policy by executing the initially learned policy for the first time step, then turning over the wheel to the teacher. A new data set is collected for the second time step, consisting of the input images seen by the teacher at time 2, and the resulting steering decisions. A policy can then be learned for time step 2 via the usual machinery of supervised learning. We can easily repeat this process to train the k-th step in a time-varying policy by observing the teacher's decisions after running the first $k-1$ steps of the learned policy [Ross and Bagnell, 2010].

It follows that each policy learned is being tested in exactly the way it was trained. The policy encounters the same distribution of input examples– albeit not the same actual examples! If an earlier policy makes errors, later ones can learn to recover from them by mimicking the teacher's recovery strategy. This halts error compounding and achieves the error rate $T\epsilon$ that one would expect in standard supervised learning.

A practical solution: DAGGER

While the above approach cleanly addresses the problem of decisions affecting the input distribution in imitation learning, it is impractical for imitation learning problems like the video game driving problem. We simply can't afford to train a policy for every step in a long sequence of decisions like driving a vehicle. Moreover, this process **should** be unnecessary if the effective time horizon is shorter.

A solution to this problem relies on *interaction*: interleaving execution and learning. In particular, at each iteration of the algorithm, the current learned policy is executed. Throughout execution, the teacher "corrects" the solution – that is, provides a preferred steering angle that is recorded in a new data set but **not** executed. After sufficient data is collected, it is

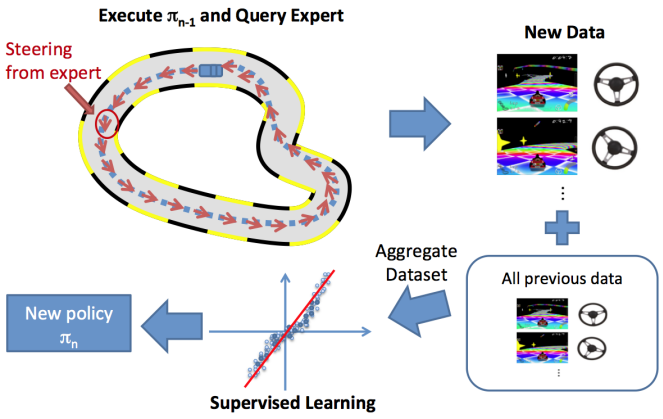

Figure 6.1.4: Illustration of the Dataset Aggregation (DAGGER) approach to imitation learning via repeated interaction. At each iteration of the algorithm, the current learned policy is executed. Throughout execution, the teacher "corrects" each step – that is, provides a preferred steering angle that is recorded in a new data set but **not** executed. Throughout these iterations, data is aggregated together to lead to the next policy. This provides much stronger guarantees than simple supervised learning.

aggregated together with all of the data that was previously collected. A supervised learning algorithm then generates a new policy by attempting to optimize performance on the aggregated data. This process of execution of the current policy, correction by the teacher, and data aggregation and training is repeated.

```
# Take an initial policy: π_0, Teacher: state -> action,
# Learner: [ (state,action) ] -> policy, GenSystemTrajectory : π -> [state]
def DAGGER(π_0, Teacher, GenSystemTrajectory, Learn):
    D = [], π = π_0
    for i in range(N): # run for N iterations
        D_i = [(state,Teacher(state)) for state in GenSystemTrajectory(π) ]
        D.append(D_i)
        π = Learn(D) #Optionally run any no-regret learner on the D_i
    return π
    # Preferred: instead return the stochastic policy that mixes uniformly between all the
    # policies learned or choose the best single policy on validation over the iterations
```

DAGGER Algorithm Pseudo-code

Intuitively, this approach creates policies that are capable of correcting their own mistakes. If the learner steers too close to the edge of the road, the policy will generate new training data that includes the teacher's preferred actions for handling such situations. The aggregation of data prevents it from forgetting previously-learned situations.

But what can one say formally about this approach? If our supervised learner is one of a large class of learners that have the *no-regret* property[Cesa-Bianchi et al., 1997], we can formalize the idea that learning a policy with low training error implies good performance at imitating the expert. Put differently, one of two things must happen: either the supervised learning problem will become too hard to solve (expected error greater then ϵ) or a policy that matches the teacher with only approximately $T\epsilon$ error over the full horizon will be learned throughout the iterations. [10]

[10] It need **not**, however, be the final policy learned. Instead, the claim is merely that one of the policies– or a uniform stochastic mixture of the entire set learned– must perform well. In practice, choosing the final learned policy is often simplest and sufficient.

Stability, Online Learning and "No-regret"

Case Study: DAGGER *in Anger*

When we apply this approach of teacher correction, aggregating data and iteratively learning policies to the car driving problem, the result is somewhat boring to watch. While simple supervised learning averages about 3-4 failures per lap, the interactive DAGGER learning approach **with the same number of examples from the teacher** very quickly reaches nearly 0 falls per

lap. **No** amount of training data enables the supervised learning approach to achieve that same performance– it always falls multiple times per lap.

It's more interesting to consider learning a complex, real-world reactive control task like flying through a cluttered domain – for example, between tree trunks underneath a forest canopy. [11] The problem follows the setup of Pomerleau's: compute features (optical flow, color histograms, simple texture features etc.), pool them over patches of the images, and provide the resulting large feature vector as an input to a regression algorithm. As output, the learner will predict the commanded lateral velocity of a human pilot and train the algorithm to reactively map these image features to controls.

[11] The "Forest of Endor" problem, to use Nick Roy's evocative phrase.

The result is a simple controller that navigates through dense forest at nearly the same effectiveness as a human pilot. [Ross et al., 2013a] [12]. Interestingly, failures largely come about due to the nature of a reactive controller and a small field of view. It's not unusual for the algorithm to dodge a tree, have that tree leave its field of view, then crash into the same tree sideways as it tries to avoid a new tree. Adding memory – whether through intelligently constructed features or through predictive state representations – represents the best hope for improving the learning of such control strategies.

[12] Videos of the approach can be found at LAIRLab BIRD Website [Ross et al., 2013b]

Recently other authors have demonstrated in success in applying DAGGER to a rich class of problems including playing a broad class of Atari 2600 games [Guo et al., 2014] and robot navigation [Kim et al., 2013].

Learning with a Goal Besides Imitation We focused entirely above on a loss function of **simple imitation**: our goal is to choose the same actions as the expert measured according to some loss function $l(y, \pi(x))$. But in many scenarios – for instance, driving – our real goal is actually substantially different. We may wish to minimize the probability of crashing, or maximize our success at manipulating an object, or achieve any other control objective that the teacher is presumably optimizing. The same style of approach is easily adopted – albeit with potentially substantially higher computational and sample complexity – for this setting by replacing the data about best action with an estimate of cost-to-go from the teacher. [Ross and Bagnell, 2014] Crudely speaking, this cost-to-go is an estimate of how hard it will be for the teacher to recover if the learner were to make a mistake. The key question of what to do when a teacher can't articulate their own cost function is taken up in the next section.

Summary

In an important sense, recent theory and algorithms for imitation learning formalize a simple lesson: one cannot learn to drive a car simply by watching someone else do it. Instead, feedback is essential – we must try to drive and receive instruction that corrects our mistakes.

Crucially, this general approach is largely agnostic to the underlying supervised learning approach. It is an interactive *reduction* to supervised learning methods. Formal results are only known for settings (like kernel machines, Gaussian processes, and linear predictors) where no-regret algorithms are known. But empirical evidence suggests that this approach is remarkably effective even when this condition doesn't formally hold, since many learning algorithms are actually both stable and good predictors.

Finally, it is important to note that all discussion here centered on learning

mappings directly from observations to controls without considering state-estimators (e.g. *filters*.) However, there is no reason one can not nor should not learn to *imitate in belief space*– that is learn mapping from the output of a filter (*e.g.* a best estimate of the underlying world state) to decisions. In practice, this is almost certainly necessary to achieve high performance; such approaches fall under the same general approach described here as we can consider the filter as simply a part of the environment and the filter output as a new, generalized observation.

6.2 *Decisions are Purposeful: Inverse Optimal Control*

Figure 6.2.1: An image of the DARPA UPI "Crusher" robot autonomously crossing rough off-road terrain. It is difficult to manually engineer the connection between perception and planning. Imitation learning techniques make it possible to automate this process. Further examples of the vehicle traversing rough terrain from temperate woodlands, to marshes, to dense vegetation, to mock-up urban environments all under autonomous control can be seen here and here.

Imitation learning is fundamentally different then classical supervised learning in another sense. For instance, consider the problem of navigating through very rough outdoor terrain – a major focus of robotics research for decades. Figure 6.2.1 shows Crusher, an autonomous robot that was developed as part of a DARPA fundamental research project into outdoor robotics. Crusher traversed thousands of kilometers of diverse, rough, terrain with minimal human intervention over years of field testing. In contrast to many other outdoor navigation efforts, it typically travelled from 0.5 to 10 kilometers between human provided waypoints. All decisions along the way were made based on information from its own perception system and (optionally) overhead maps (*e.g.* images collected from mapping companies like those used in Google Maps).

A reactive controller is unlikely to make any meaningful progress towards a goal in this domain; it is difficult to imagine training a simple supervised learning method to accomplish this complex task. The robot must instead execute a long, *coherent* sequence of decisions in order to achieve its goal. This requires a sense of planning – and of replanning as new perceptual information becomes available – to achieve good performance.

To adapt to imitation learning to this setting, it is valuable to consider the architectures that roboticists have created to achieve intelligent and deliberative navigation. Since the pioneering projects in off-road navigation [Hebert, 1997], effective robot navigation has relied on an optimal control or replanning architecture to structure decision making. This architecture has been replicated and refined throughout the field of robotics [Zucker et al.,

2011, Urmson et al., 2008, Wellington and Stentz, 2004, Leonard et al., 2008, Jackel et al., 2006, Bachrach et al., 2009] and is currently used in the most advanced autonomous navigation systems.

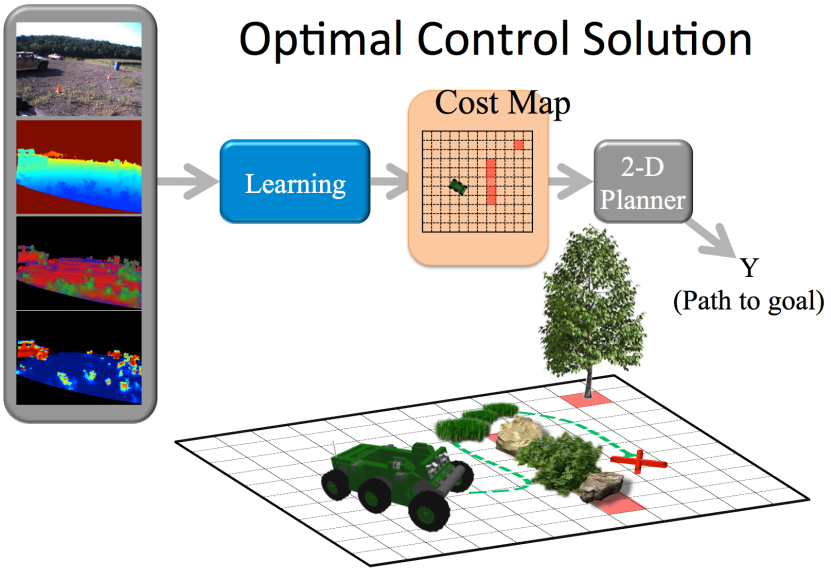

Figure 6.2.2: Components of a robot architecture: Sensors (LADAR, cameras) feed a perception system that computes a rich set of features (left side) developed in the computer vision and robotics fields. Depicted features include estimates of color and texture, estimated depth, and shape descriptors of a LADAR point cloud. Features that are not depicted here include estimates of terrain slope, semantic labels ("rock"), and other feature descriptors that can be assigned a location in a 2D grid map. These features are then massaged into an estimate of "traversability" – a scalar value that indicates how difficult it is for the robot to travel across the location on the map.

Figure 6.2.2 shows a diagram of such a robot architecture. Sensors (LADAR, cameras) feed a perception system that computes a rich set of features (left side) developed in the computer vision and robotics fields. Features that are shown in Figure 6.2.2 include color, texture, estimated depth, and shape descriptors of a LADAR point cloud. Features that aren't shown in the diagram include estimates of terrain slope, presence of semantic categories ("rock"), and many other feature descriptors that can be assigned a location in a 2D grid map. These features are then massaged into an estimate of "traversability" – a single scalar value that that indicates how difficult it is for the robot to travel across the location on the map. This value is included in a "cost map" for each state of the robot. The final decisions of the robot represent steps along a minimum cost plan from the robot's current location to a goal state. The robot executes a small part of the current plan at each time instant. As the robot moves, the perception system provides updates about the terrain it is crossing. The cost map is then updated with new traversibility values and a new plan is generated.

Real implementations, of course, have much richer spaces of states then simply a discretization of geometric locations of the robot center. Almost inevitably, they contain a hierarchy of planning layers that capture a state-space description of the robot at higher and higher fidelities as they consider shorter time-scales. [Zucker et al., 2011] The diagram in Figure 6.2.2 nevertheless captures the essential behavior of many such systems and is often exactly the behavior of the coarsest levels of such a hierarchy.

From the point of view of this architecture, only one role exists for imitation learning. Perception computes features that describe the environment;

the output control is always the prefix of the currently believed-to-be-optimal plan. The learning algorithm then must transform the perceptual description (a feature vector) of each state into a scalar cost value that the robot's planner uses to compute optimal trajectories.

Perhaps surprisingly, costing is one of the most difficult tasks in autonomous navigation. As documented in [Silver, 2010], this single piece of code required the largest number of changes and demanded the most engineering effort. The entire behavior of the robot depends on this module working correctly. Moreover, nearly all changes to the software end up requiring either validation or modification of the costing infrastructure. If a sensor changes or the perception system develops or refines features, the costing mechanism must be updated. If the planner changes – for instance by C-space expanding obstacles– the costing system must change. Tuning and validating such changes demands a tremendous amount of time and effort.

However, the robot can use imitation to learn this cost-function mapping. A teacher (that is, a human expert driver) drives the robot between waypoints through a representative stretch of complex terrain. We can then set up a problem of *Inverse Optimal Control*: that is, we attempt to find a cost function that maps perception features to a scalar cost signal so that the teacher's driving pattern appears to be optimal.

Nathan Ratliff formulated the problem of learning such a cost function as an application of *structured prediction* and demonstrated that very simple sub-gradient based algorithms are remarkably effective at solving it. [13]

Inverse Optimal Control (IOC) is a rich and fascinating subject that dates back to Kalman's work on the Linear-Quadratic-Regulator problem. Kalman [Kalman, 1964] asked (and answered) a natural question: given a linear controller or policy, is there a cost function that makes it optimal for a given Single-Input Single-Output plant?[14] Boyd [Boyd et al., 1994] provided a simple convex programming formulation for the multi-input, multi-output linear-quadratic problem.

Only recently, however, has Inverse Optimal Control become an engineering tool for designing intelligent systems. The recent work in the machine learning on this area [Ng and Russell, 2000, Abbeel and Ng, 2004, Ratliff et al., 2009c, Ziebart et al., 2008a, 2010] can be summarized as providing several advances over the early contributions:

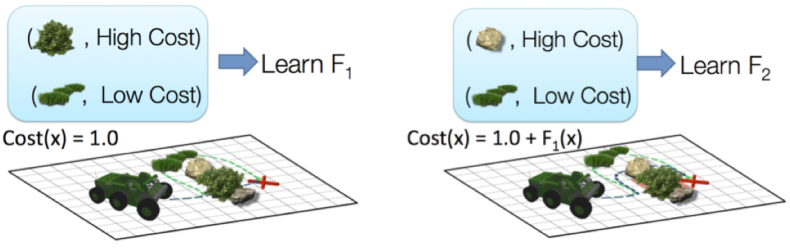

Figure 6.2.3: Iterations of the LEARCH algorithm. See the main text for a description of how this algorithm modifies its estimate of a cost function by mapping features of a state to a scalar traversability score.

(1) Enabling a cost function to be derived (at least in principle) for essentially arbitrary stochastic control problems using convex optimization techniques – any problem that can be formulated as a Markov Decision Problem.

(2) Requiring a weak notion of access to the purported optimal controller.

[13] In fact, surprisingly such sub-gradient methods are actually the best known algorithms for solving large support vector machine and more general structured margin problems in a follow-on paper. These techniques are now the de facto standard and have been implemented in a wide range of libraries [Agarwal et al., 2014].

[14] Amusingly, while Kalman's work was critical in advancing the use of state-space techniques for control, his solution to the IOC problem was rooted fundamentally in frequency domain techniques.

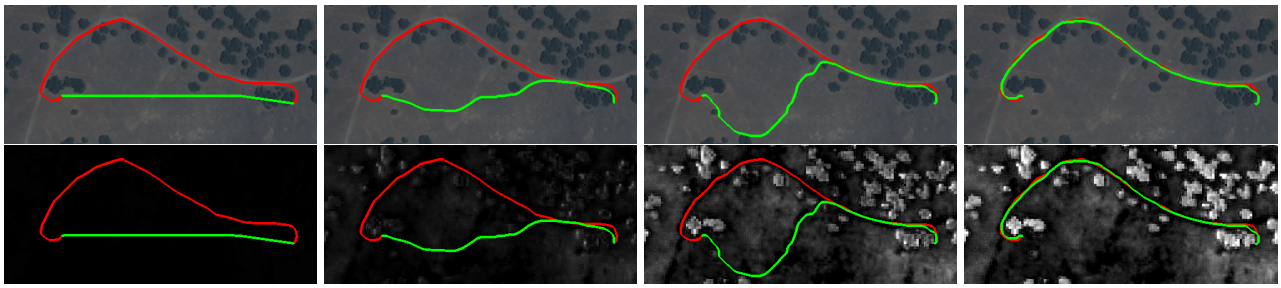

Figure 6.2.4: A demonstration of the Learning to Search (LEARCH) algorithm applied to provide automated interpretation in traversability cost (Bottom) of satellite imagery (Top) for use in outdoor navigation. Brighter pixels indicate a higher traversability cost on a logarithmic scale. From left to right illustrates progression of the algorithm, where we see the current optimal plan (green) progressively captures more of the demonstration (red) correctly.

No closed form description of the controller needs to exist, just access to example demonstrations.

(3) Statistical guarantees on the number of samples required to achieve good predictive performance and even stronger results in the online or no-regret setting that requires no probabilistic assumptions at all.

(4) Robustness to imperfect or near-optimal behavior and generalizations to probabilistically predict the behavior of such approximately optimal agents.

(5) Some algorithms further require **only** access to an oracle that can solve the optimal control problem with a proposed cost function a modest number of times to address the inverse problem.

The central premise of IOC techniques for imitation learning is that structuring a space of policies as approximately optimal solutions to a control problem is a representation that enables effective deliberative action. Moreover, IOC methods rely on the observation that cost functions generalize more broadly [Ng and Russell, 2000] then policies or value functions. Thus, one should seek to learn and then plan with cost functions when possible, and revert to directly learning values or policies only when it is too computationally difficult.

The Learning To Search (LEARCH) Algorithm. The key algorithmic ideas for modern IOC algorithms statistical guarantees can be understood in the framework of convex optimization of an objective function that stands as a surrogate for correctly predicting the plan or policy that the teacher will follow. As such, many of the original approaches were formulated in terms of large quadratic programs [Ratliff et al., 2006] or Linear-Matrix Inequalities [Boyd et al., 1994] and the resulting algorithms are somewhat opaque. However, more recent algorithms designed for solving large scale and non-linear problems are quite natural and might be guessed without even appreciating they are solving a well-defined optimization problem.

Consider, for instance, the *Learning to Search* (LEARCH) approach of [Ratliff et al., 2009c] in the context of rough-terrain outdoor navigation discussed above. We may step through the algorithm on a cartoon example to see why it might work. We first consider a path driven by teacher from a start point to a goal point, then imagine a simple planning problem on a discretized grid of states that the robot can occupy. Every iteration of the algorithm consists of the following: (a) computing the current best optimal plan/policy; (b) identifying where the plan and teacher disagree and creating a data set consisting of features and the direction in which we should modify

the costs; (c) using a supervised learning algorithm to turn that data set into a simple predictor of the direction to modify costs; and (d) computing a cost function as a (weighted) sum of the learned predictors.

```
# Take a sequence of MDPS and demonstrations [M_i,ξ_i)]_{i=1}^N where MDP M is a stochastic planning problems
      consisting of states, actions, and a transition function used for planning,
#  (optional) loss functions l_i: state, action->R that measures deviations from the demonstrated plan,
# feature function f: state, action -> R^d  that describes states in terms of features meaningful for
      cost

def LEARCH({(M_i,ξ_i)}_{i=1}^N, f, {l_i}_{i=1}^N = 0):
  s_0 = 0 #Initialize (log)-cost function, s_0 : R^d → R to zero
  for t in range(T): # run for T iterations
      D = [] # Initialize the data set to empty
      for i in range(N): # for each example in the data set
          c_i^l = e^{s_t(F_i)} - l_i^T # Compute costmap with optional loss augmentation
          μ_i^* = Plan(M_i,c_i) # find the resulting optimal plan μ_i^* = argmin_μ c_i^l μ, μ consistent with M_i
          # μ*'s counts the time spent in state/actions pairs under the plan—
          # for deterministic MDPS this is simply an indicator of whether the optimal plan
          # visits that edge in the planning graph
          μ_i = [ ξ_i.count((s,a)) for (s,a) in M_i ] #compute states-actions in demonstration
          # Generate positive and negative training examples:
      D_i = [ (f_i(s,a), sign(μ_i^{*sa} - μ_i^{sa}), |μ_i^{*sa} - μ_i^{sa}|) for (s,a) in M_i ]
      # if |μ_i^{*sa} - μ_i^{sa}| = 0 for a state-action we can simply not generate that point
      D.append(D_i)
      h_t = Learn(D) #Train a regressor (or classifier) h_t: R^d->R on the resulting weighted data set
      s_{t+1} = s_t + α_t h_t # Update the (log) hypothesis cost function
  return exp(s_T)
```

LEARCH Algorithm Pseudo-code

Theory and Guarantees. At its heart, the problem of correctly identifying a teacher's reward function is ill-posed. First, it is unreasonable to believe the teacher is truly an optimal controller for some simple Markov Decision Process that describes the world. Second, given a single behavior, there are generally **infinitely many** reward functions that lead to the same behavior and are thus unidentifiable. [Abbeel and Ng, 2004]

There are thus two commonly used notions of successful IOC used in machine learning. The first (originated by Abbeel and Ng [Abbeel and Ng, 2004]) considers a class of reward functions that are linear in a set of features that describe states. Our goal then is to ensure that whatever behavior is learned by imitation achieves the same reward as the teacher even when the reward function itself cannot be identified. The second (typified by *Maximum Margin Planning* [Ratliff et al., 2006, 2009c]) is agnostic to whether the teacher is actually an optimal controller or even cares about a reward function. Instead, it quantifies a notion of successful imitation – for instance, agreement with the trajectory taken by the teacher – and then attempts to optimize that notion of agreement with the teacher.

These notions are surprisingly closely tied. Methods like *Maximum Margin Planning* that ensure successful agnostic imitation also can provide guarantees with respect to the teacher's reward function (if it exists!).[Syed and Schapire, 2007] Conversely, while methods like the Maximum (Causal) Entropy approach of [Ziebart et al., 2008a], which we cover extensively in the next chapter, are also designed to achieve the same reward as a teacher, they can also be understood in a dual formulation as maximizing the likelihood of the teacher's plans under a robust statistical model of the agent's behavior. [Ziebart et al., 2010, 2013] Moreover, some methods, like that of [Ziebart et al., 2010], have yet another interpretation in terms of optimal control perturbed by certain shocks that are not visible to the learner. [Rust, 1994]

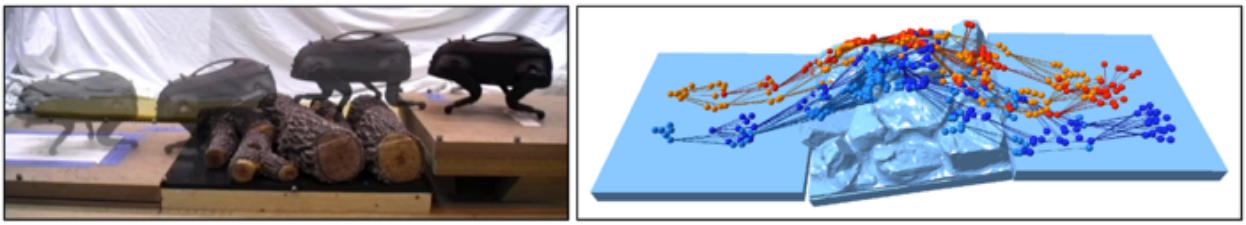

Figure 6.2.5: (Left) LittleDog platform crossing a terrain. (Right) Planning system that relies on a learning approach to cost function generation. Each color represents a different foot and arrows indicate the parent/child relationship between footsteps under consideration. [Zucker et al., 2011]

IOC in other Domains The notion of learning such deliberative strategies by tuning the cost function of a planner isn't unique to outdoor navigation– it arises anywhere long horizon plans are needed and relatively complicated features exist to describe the state space. [Ratliff et al., 2006, Zucker et al., 2011] developed a technique for learning costs (and a hierarchy of heuristics) by demonstration (see Figure 1) for a rough terrain legged locomotion planner. In essence, quasi-static locomotion is treated as discrete planning problem of carefully arranging footfalls. A complex cost function that takes into account the terrain at each individual foot as well as features of the entire robot pose that are correlated with good foot placements (for instance, the size of the polygon of support of the robot [Zucker et al., 2011]) was learned from expert demonstration. Multiple research groups have since embraced similar techniques [Kalakrishnan et al., 2011, Kolter et al., 2007].

Purposeful Prediction. Often, behavior demonstrated is only approximately optimal or may appear to have some non-determinism in its decisions. This can be understood in two ways: people are not in fact "optimal" in their decision-making for any reasonable definition of that word, and even more so, the world those people inhabit is not the simple Markov Decision Process we use as our model in Inverse Optimal Control techniques. [15] Recent IOC learning techniques manage such uncertainty and moreover can make probabilistic predictions of what people are likely to do even in such imperfect models.

The ability to imitate a person's imperfect but deliberative behavior implies the ability to predict it. In Figure 6.2.6 we see examples of Activity Forecasting: predicting people's likely trajectories in novel scenes via computer vision and inverse optimal control by learning what they are approximately optimizing in their decision making. [Kitani et al., 2012]

For instance, consider the problem of predicting the likely routes that a driver might take to travel from home to a store. We can consider a graph that describes the road network with nodes corresponding to road segments and edges between road segments that connect. Each road segment is annotated with a rich set of features x (dozens or hundreds) that describe it [Ziebart et al., 2008b] – such as expected travel times at the speed limit, the grade of the road, the toll cost of that segment, the number of lanes, whether a church is located along the road, or the presence of a guarded left turn.

The approach of [Ziebart et al., 2008a] efficiently learns a function $c(x)$ that linearly combines such features to best fit a distribution over trajectories ψ taken by the driver according to the maximum entropy model $p(\psi) \propto \exp(-V(\psi))$, where V is the total cost of the trajectory, $\sum_{x \in \psi} c(x)$.

[15] *I.e.,* the map is not the terrain.

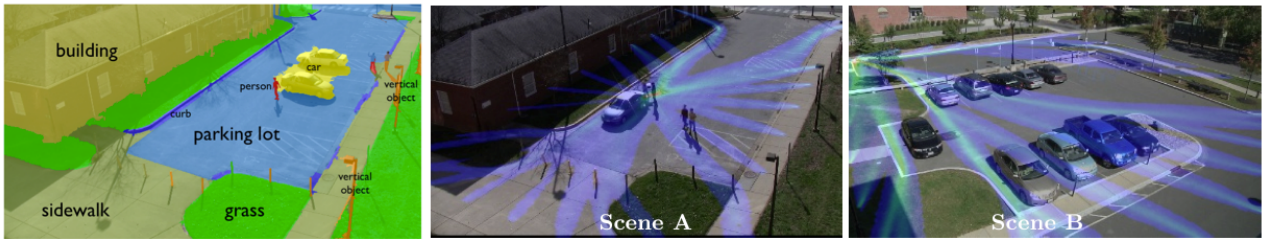

Figure 6.2.6: (Left) Automatic semantic classification of a scene via machine learning[Munoz et al., 2010, Munoz et al., 2013] (Right) Activity forecasting combines semantic perception techniques to identify the actors and object types in a scene with the probabilistic formulation of inverse optimal control to predict an actor's future destinations and likely trajectories based on partial trajectories. Each image depicts the predicted distribution of future states for a pedestrian. The absence of color implies very low probability, blue implies low probability, and yellow to red higher. Only a few potential goals are shown, and only with a single observation (predictions improve as more of the path is seen), to simplify the figure. [Kitani et al., 2012, Ziebart et al., 2013, 2008a]

When these models are combined with a prior distribution over potential destinations, they learn both a driver's implicit preferences (for example, going out of the way to avoid both unguarded left turns and expensive tolls) and provide a estimate of a drivers destination and likely future routes after beginning a trip. The use of the maximum entropy formulation ensures a strong guarantee on the predictions– no other approach to forecasting an agent's actions that uses the same information about features [Ziebart et al., 2013] can ensure smaller predictive loss.

This approach establishes the deep connection between probability theory, and particularly the Maximum Entropy Method, and inverse optimal control, where previously, these were understood as disparate techniques for modeling decision-making. [Ziebart et al., 2008a] This thread of work culminated in a new principle for the statistical prediction of interacting systems (e.g. a driver and the world, multiple agents playing a dynamic game) [Ziebart et al., 2010, 2013]. [16]

Similar techniques can be applied to predict where people are likely to walk in a complex visual scene. For instance, such methods could recognize cars and sidewalks in a scene and reason that a person will climb over a car if strictly necessary to reach a goal, but will preferentially take advantage of a sidewalk where available. [Kitani et al., 2012] Moreover, such techniques have been applied to aid robot navigation and predict pedestrian behavior. [Ziebart et al., 2009, Kretzschmar et al., 2014]

Work by [Baker et al., 2009] demonstrates people reason about others as deliberative agents as well. This inverse planning framework elegantly captures aspects of the human "Theory of Mind." Work in operations research and econometrics, particularly by Rust [Rust, 1992, 1994], derives predictive distributions by developing a framework for learning cost functions and predictive stochastic policies for agents acting according to a Markov Decision Process (MDP). Intriguingly, the same policy structure and dynamic programming algorithms derived from a maximum entropy formulation are developed from considering an economist with only partial access to the prediction problem and including "shocks" in a model of what would otherwise be optimal behavior. These close connections between operations research, control theory and machine learning deserve much deeper investigation.

6.3 *Structured Prediction as Imitation Learning*

At first blush, it seems counter-productive to phrase a supervised learning problem as one of imitation learning. Isn't the point of this article that imitation learning is a *harder* problem then that of supervised learning? However,

the relationship between the two is more subtle than this simple picture suggests. Within supervised learning, we often consider problems of *structured prediction* where the goal is to make a set of inter-related predictions – for instance, to semantically label all of the pixels within an image (e.g., Figure 6.2.6) or to turn a sentence into a parse tree. [Daumé III et al., 2009] suggests that a natural way to think about structured prediction is to consider it as predicting a sequence of decisions – *e.g.* what to label a particular pixel *given* current guesses of labels – and moreover that the expert we are imitating is simply the ground truth. [17]

From this viewpoint, structured prediction problems are merely degenerate versions of imitation learning problems, where the teacher can be specified algorithmically based on training data and the dynamics of the environment are particularly simple. When viewed through this lens, structured prediction problems suffer the same difficulties as problems of imitation learning. Predictions of some random variables (e.g., pixel classes) influence future predictions of other pixels and a naive training of such an architecture leads to disastrous compounding of errors.

For instance, consider the *inference machine* approach of [Munoz et al., 2010, Ross et al., 2011b]. The central idea is to consider labeling an image or point cloud sequentially in a pattern mimicking that of highly effective graphical model inference algorithms like *mean-field* or *belief-propagation*. We iteratively pass through each pixel and label it using a combination of (a) features that describe that particular visual element (e.g. texture, color) as well as (b) the currently predicted labels of visual elements that are nearby. The use of such nearby elements for predictions enables effective contextual reasoning. It's easier to distinguish a tree trunk from a telephone pole if we know that the material located above it is vegetation. Such contextual reasoning has traditionally been approached through the lens of probabilistic graphical models. We first learn a templated parameterized probabilistic model, then use approximate inference techniques to infer random variables in that model. The imitation learning approach makes the inference procedure itself the model. [18]

Such techniques— and more generally, applying methods like DAGGER to structured prediction– have been demonstrated to provide state-of-the-art predictive performance and speed of inference on a wide range of structured prediction tasks. These include examples from predicting semantic labels for images [Munoz et al., 2010], identifying human poses in images and video [Ramakrishna et al., 2014], summarizing documents with the SCP algorithm [Ross et al., 2013c], and a broad range of Natural Language Processing Tasks [Daumé III et al., 2009, He et al., 2012]. [19]

6.4 What's Next?

Only in the past decade has imitation learning come into its own as a problem distinct – and distinctly important – from the classical ones of reinforcement and supervised learning. The structure of the problem gives us far more purchase then the general reinforcement learning (RL) problem. But it also acknowledges that learning may actually affect the world and that the classic assumptions of supervised learning will lead to poor performance and compounding errors.

[17] Hal Daume at a NIPS workshop first clearly expressed to me the notion that we should often think of supervised learning problems as being imitation learning problems in disguise. This viewpoint has certainly been addressed by others – John Langford has particularly championed the notion that complex prediction problems should be thought of in terms of reductions to simpler problems.

[18] It is natural to view the inference machines in the language of deep modular neural networks [LeCun et al., 1998, Bengio, 2009] – an inference machine is a very deep set of repeated predictions about a particular visual element or other random variable. An alternative to the iterative training procedures espoused here includes a direct backpropagation of errors of final predictions made about such nodes. Interestingly, however, such results limit our prediction algorithms (no random forests!) and may not always be an optimal approach. Investigating when backpropagation can effectively tune the parameters of an inference machine remains an important subject for research.

Shih-En Wei, Varun Ramakrishna, Takeo Kanade, and Yaser Sheikh. Convolutional pose machines. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4724–4732, 2016

[19] Videos of such inference approaches approaches can be found at the Inference Machine Website.

Apprenticeship: From Imitation to Reinforcement

An important next step is moving from pure imitation to apprenticeship[20], which leverages user demonstration but optimizes performance on an alternate metric. Many examples in the literature consider where it can have significant benefits. For instance, [Nechyba and Bagnell, 1999] demonstrates a learned speed control for a simulated driving task that is improved by an RL gradient descent procedure that ensures good performance while attempting to stay as close as possible to demonstration. Similarly, the works of [Atkeson and Schaal, 1997], [Kober and Peters, 2010] and [Coates et al., 2009] use exactly same kind of benefits to achieve impressive performance. Such approaches are even more important when the learning **cannot** be interactive– for instance, when learning by watching a video.

[20] Borrowing this phrase from Pieter Abbeel, who uses it to refer to systems that combine imitation and reinforcement learning.

Interestingly, theoretical results suggest an enormous practical benefit for learning from an expert demonstrator – but perhaps not in the way typically considered. The theories of Policy Search by Dynamic Programming [**?**], Conservative Policy Iteration [Kakade and Langford, 2002], and No-Regret Policy Iteration [Ross and Bagnell, 2014] show that the key to making reinforcement learning easier is to identify the distribution of states that a good policy spends time in (the so-called *baseline measure* of [**?**]). Access to such a distribution makes the problem of a learning an optimal memory-less policy in a Partially Observed MDP a polynomial-time problem. It also effectively makes the sample complexity of learning into a policy with generative model access to a large MDP polynomial in the horizon of the problem.

Such results, however, show no significant benefit for observing what actions an expert demonstrator might choose – the benefit of this seems secondary to the benefit of knowing what states are important to focus on. Understanding practically and theoretically how we can get the best of imitation and reinforcement learning will be a major area of future research.

Extending Inverse Optimal Control for Imitation Learning.

Much recent work has focused on models for which the optimal control problem itself can only be approximately solved. [21]

Such methods and combinations of methods seem likely to dramatically increase the applicability of this rich class of predictive models and procedures for inferring reward functions.

[21] [Ziebart et al., 2012] and [Dragan and Srinivasa, 2012] and [Levine and Koltun, 2012] consider locally quadratic approximation of the maximum entropy inference problem. [Huang et al., 2015] has developed a variant of the maximum entropy IOC that relies on a combination of function approximation of the log-partition function and sampling to estimate the gradient. [Ratliff et al., 2009a] blends the advantages of IOC-based methods with methods that directly learn to predict actions.

Putting it together

Perhaps surprisingly, existing techniques rarely consider both aspects of imitation learning I have discussed in this paper: they tend to focus either on the problem of compounding errors or the need for learning deliberative strategies. As these problems are largely orthogonal, we expect future techniques for imitation learning will address both issues simultaneously.