# 7
# *Moment Matching, GANs, and all that*

## *Introduction*

In this chapter we review Inverse Optimal Control from the *Maximum Entropy* perspective and connect these to the general goal of learning probability distributions from examples. This chapter establishes the key role of both moment matching/integral probability metrics and a game theoretic view of learning behavior. This viewpoint allows us to connect IOC and the Maximum Entropy Principle more broadly to a family of generative models known as *Generative Adversarial Networks*. Efficiency is achieved for continuous control problems via a Laplace approximation and techniques are studied to learn costs and find anomalies with this approximation.

This chapter extends the previous one by considering methods for learning cost functions from human demonstration that highlight the intimate connections between probabilistic inference and optimal control. Moreover, the approach to Inverse Optimal Control embraced in this chapter establishes a connection to recent development in *Generative Adversarial Networks*, optimal transport, and the general approach of *moment matching*.

## *7.1 Decisions are Purposeful: Inverse Optimal Control*

## *7.2 IOC as Moment Matching*

Let us consider the problem of matching such expert driver behavior. What would success mean here? A reasonable requirement might be that:

$$p(\xi|\Gamma) \approx \tilde{p}(\xi|\Gamma)$$

where we use $\xi$ to represent a trajectory the expert might take, $\Gamma$ to represent the general context of the planning problem including environment and sensor data at a particular time-step, we use $p$ to represent our model distribution and we use $\tilde{p}$ to represent the empirical distribution of examples.

A weaker condition that we'll work with here is that,

$$E_{p(\tilde{\Gamma})}[p(\xi|\Gamma)] \approx E_{p(\tilde{\Gamma})}[\tilde{p}(\xi|\Gamma)]$$

Throughout the remainder of this work, we'll largely suppress the dependence on the context $\Gamma$. Near the end, we'll consider stronger notions than the average case performance.

I'll ignore here the philosophical quandaries around the demonstrations coming from a probability distribution. They don't. It's simpler to imagine they do for the purposes of this work and defer a generalization to regret and other non-probabilistic notions of performance to another time.

How, then, should we make it so that $p(\xi) \approx q(\xi)$ (where in general $q$ is a distribution, often the empirical one $\tilde{p}$) in a manner that is empirically measurable, and where we only have sample access to $q$ (and might prefer for engineering reasons to only require sample access to $p$)? Let us consider first a finite set of *cost functions* $\mathcal{F}$ that measure, according to some notion, how good a trajectory is. We might then require that the demonstrations and the learned model achieve the same costs for each of the cost functions: [1]

$$\forall f \in \mathcal{F}, E_p[f] = E_q[f] \tag{7.2.1}$$

[1] We might prefer, instead that the learned model has a **lower** cost, $E_{p(\tilde{\Gamma})}[p(\xi|\Gamma)] \leq E_{p(\tilde{\Gamma})}[\tilde{p}(\xi|\Gamma)]$. This is also easy to implement, (and covered in generality in ) but if $\mathcal{F}$ is closed under negation ($f \in \mathcal{F} \Rightarrow -f \in \mathcal{F}$) then meeting the inequality immediately implies we meet the equality.

K. Waugh, B. D. Ziebart, and J. A. Bagnell. Computational rationalization: The inverse equilibrium problem. In *Proceedings of the International Conference on Machine Learning*, June 2011

## *Moment matching*

This constraint, Equation 7.2.1, is known as a moment matching constraint. We note the important property that moment matching implies that for the much broader class of cost functions composed of linear combinations of $\mathcal{F}$, $\mathcal{F}_{\text{lin}} = \{\sum_i \lambda_i f_i | f \in \mathcal{F}\}$, we also have

$$\forall f \in \mathcal{F}_{\text{lin}}, E_p[f] = E_q[f] \tag{7.2.2}$$

That is, moment matching under a set of cost functions, also implies matching all linear combination of such moments.

**Prove it in the margin.**

**Classes of moments.** The classical moments are the monomials, e.g. $x_1^2 x_3^4 x_5$. Classic results, perhaps unsurprisingly, indicate matching all monomial moments implies convergence in distribution. Matching all bounded functions implies that the total variation distance between distributions is 0. [2]. A more classic set of functions relevant to robot motion planning including quadratic hinges on an individual variable [3] $\max^2(0, x_i)$, or generalized to $\max^2(0, g(x, \Gamma))$ for a class of functions $g$.

[2] Bharath K. Sriperumbudur, Arthur Gretton, Kenji Fukumizu, Gert R. G. Lanckriet, and Bernhard Schölkopf. A note on integral probability metrics and $\phi$-divergences. 2009. URL http://arxiv.org/abs/0901.2698

[3] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *ICRA*, 2009b

## *From moments to metric*

We can allow slack in the notion of moment matching and provide a distance metric (divergence) between probability distributions. We denote by

$$D_{\mathcal{F}}(p, q) = \max_{f \in F} E_p[f] - E_q[f] \tag{7.2.3}$$

the *Moment Matching Metric* also known as the *Integral Probability Metric* associated with $\mathcal{F}$. That is, we measure the difference between distributions by considering the worst-case gap between them in the class of cost functions $\mathcal{F}$.

We consider here the case that the class $\mathcal{F}$ is symmetric and I blithely ignore the difference between a supremum and a maximum.

If the function class is a Reproducing Kernel Hilbert Space of fixed norm, this metric is known as Maximum Mean Discrepancy [4], while if the class of functions is all 1-Lipshitz continuous ones, this metric is equivalent to the *earth mover distance*.

[4] Bharath K. Sriperumbudur, Arthur Gretton, Kenji Fukumizu, Gert R. G. Lanckriet, and Bernhard Schölkopf. A note on integral probability metrics and $\phi$-divergences. 2009. URL http://arxiv.org/abs/0901.2698

## *Measuring by samples*

This definition of divergence between distributions is useful both because it captures the important distinctions in a concrete notion of cost, but also because expectations can be measured by only observing samples. If we have, for instance, a single $f$ to consider in our cost function space, the strong law implies immediately $\frac{1}{N}\sum_i f(\xi_i) \to E_p[f]$ for paths drawn from $p(\xi)$,

and we can bound the error in this estimate with high probability, typically as a rate of $\epsilon = O(\frac{1}{\sqrt{N}})$. Intuitively this follows from the observation that the variance of a sum of *i.i.d.* random variables is linear in the number of samples and hence that the standard deviation of an average of random variables must decrease as $O(\frac{1}{\sqrt{N}})$.

For broad classes of cost functions we are basically solving a classical supervised learning problem: finding the function $f$ that maximally distinguishes between $p$ and $q$. If we take a gradient of equation 7.2.3 with respect to the function $f$ and then take a random instance, we find an (unbiased) estimate of the gradient is:

$$f(\xi_p)k(\xi_p, \cdot) - f(\xi_q)k(\xi_q, \cdot)$$

in an RKHS— replace k with a delta function or an indicator function as appropriate for the space), where $f$ is the element of $\mathcal{F}$ that is the current linearization point.

## *Entropy Regularization*

In general, we can only approximately estimate moments from samples. For a finite number of moments, there will be many distributions that are consistent with the moments known. A classic approach to breaking this ambiguity is the maximum entropy method that prescribes assigning the highest entropy distribution consistent with the known moment constraints. That are many justifications of this principle (see [5]) that are more-or-less compelling depending on applications. Such regularization has proven critical in inverse optimal control applications. The probabilistic viewpoint leads to a well-defined answer to modeling imperfect "optimal" behavior while managing the ill-posedness of equally good solutions.

The result sets up an optimization problem:

$$\max_p H[p(\xi)] \tag{7.2.4}$$

$$s.t. \forall f \in \mathcal{F}, \tag{7.2.5}$$

$$E_p[f(\xi)] = E_q[f(\xi)] \tag{7.2.6}$$

where $q$ is typically the empirical distribution over observed paths, $\tilde{p}$.

Is it easy to relax the equality above with slack via the moment matching metric,

$$\max_{f \in F} E_p[f] - E_q[f] \leq \epsilon$$

We defer that now, except to note that slack in the primal, MaxEnt problem, corresponds to regularization in the dual parameters – a beautiful result due to Dudik et al. [2004].

The resulting Lagrangian optimization problem is a game between two players. A generator $p(\xi)$ that computes the best distribution, and a cost function (weighting) $\lambda$ attempts to discriminate between the two distributions. Some techniques, like GANs, attempt to solve the problem by a saddle point finding approach. This is a potentially powerful approach for IOC [6]. However, the industry standard, if you'll oblige, is to solve for the optimal generator $p$ in closed form. Given a function class $F_{\text{lin}}$ that is closed under linearity, we can conclude

$$p(\xi) = \frac{exp(-\text{cost}(\xi))}{Z} \tag{7.2.7}$$

[5] E. T. Jaynes. *Probability theory: The logic of science*. Cambridge University Press, 2003; and Peter D. Grunwald and A. Philip Dawid. Game theory, maximum entropy, minimum discrepancy and robust bayesian decision theory, 2004. URL http://arxiv.org/abs/math/0410076 Interestingly, such entropic regularization occurs in purely computational settings as well from AdaBoost to Sinkhorn iterations where it break ambiguities and leads to fast, numerically stable algorithms.

It's interesting to note that the duality viewpoint suggests that we should often actually measure a divergence not by its maximum over $\mathcal{F}$, but by the $L_2$ norm of the violations. I'm unaware of that being explored in the probabilistic literature, but it is much more natural notion of approximate moment matching than IPMs for many applications.

[6] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. 2016. URL http://arxiv.org/abs/1606.03476

where in the linear case for a set of features of a path $\phi(\xi)$, we have cost $= \lambda^T \phi(\xi)$.

Given the form, the goal now is only to compute the cost function, or equivalently it's parameters ($\lambda$).

**Derive this in the margin using Lagrange multipliers.**

## *Markov structure*

We being by considering a problem with structure defined by cost $f(\xi) = \sum_t cost(x_t, u_t)$ and markov transition dynamics $x_{t+1} = h(x_t, u_t)$. For the linear case we write such costs as $\lambda^T \phi(x, u)$, for a set of feature functions $\phi$. This structure makes it (exponentially in $T$) more efficient to find solutions and corresponds to the classical structure of optimal control.

We note in all cases there is nothing essentially different about making each weight or cost a function of $t$ and that this could be useful for receding horizon control where our uncertainty about the future grows rapidly. [7]

If we want to compute derivatives with respect to $\lambda$, we should find the optimal generator $p$ and then eliminate to form a dual optimization depending only on the costs (i.e. only on the variable $\lambda$ rather than $p$). A quick computation of the derivatives with respect to $\lambda$ after eliminating $p$ gives us:

$$\sum_t E_p[\phi(x_t, u_t)] - E_q[\phi(x_t, u_t)]$$

where $E_q[\phi(x_t, u_t)]$ is typically a constant estimated by observed data (i.e. $q = \tilde{p}$). Optimization then boils down to computing expectations under the model efficiently $E_p[\phi(x_t, u_t)]$. We can do this via a dynamic programming algorithm (effectively inference in a random field) which is precisely equivalent to classical value iteration with the min replaced by $\log \sum \exp$, aka *softmin*. The forward pass can then be computed analytically or via samples by noting

$$p(u_t|x_t) \propto \exp(-Q_t(x_t, u_t))$$

where $Q_t(x_t, u_t) = \text{cost}(x_t, u_t) + V_{t+1}(x_{t+1})$ and $V_{t+1}(x_{t+1}) = \text{softmin}_u Q_{t+1}(s_{t+1}, u)$. This can then be pushed through forward dynamics. In general, we can view the backwards pass as transforming an undirected graphical model into a directed one, and then employing an ancestral sampling procedure (or exact forward integration) to compute expectations.

This recursive definition enables an optimal policy computation for a tabular model and suggests methods to enable approximation in the non-tabular case.

## *Scaling to continuous trajectory generation*

We're typically interested in high dimensional continuous control problems. Some powerful tools exist here for inference including Monte-Carlo methods and Vernaza's value-function symmetry method[8]. The simplest version however is the Linear-Quadratic / Gaussian model. This was first developed by [9] (following Ratliff's thesis version of IOC for LQR). We can follow the development presented in Ziebart, or take an alternate approach of approximation. In particular, we can consider the Laplace approximation where we find the most probable trajectory $\xi^*$ and a quadratic expansion about it to compute approximate partition functions, entropy, variances, samples and

The general case of stochastic dynamics is handled in Ziebart et al. [2013], and requires more careful reasoning about causal entropy, but the resulting algorithms are largely identical.

[7] Having variable costs as a function of time (at least without exponential damping) can, however, lead to inconsistent decision making. For instance, an agent might always choose to defer expensive decisions to the future, where costs are lower, as during training, "the future never arrives". This is closely related to the problems of off-policy imitation described in the previous chapter.

[8] P. Vernaza and D. D. Lee. Efficient dynamic programming for high-dimensional, optimal motion planning by spectral learning of approximate value function symmetries. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011

[9] B. D. Ziebart, J. Andrew Bagnell, and A. K. Dey. Modeling interaction via the principle of maximum causal entropy. In *Proceedings of the 27th International Conference on Machine Learning*, 2010

expectations. Under linear dynamics and a quadratic cost function, the true distribution on both actions $p(u_t|x_t)$ and on states are Gaussian and thus the approximation is actually exact.

In the absence of these, the technique is an approximation, albeit a powerful one. The key idea is to leverage a family of techniques based on *Differential Dynamic Programming* [10] to compute efficiently the best (and thus most probable) action as a policy $u_t|x_t$ and the curvature in the action-value function as a function of $u$, denoted $Q_{uu}$. This quadratic approximation in value (cost-to-go) provides a Gaussian approximation of action-selection which enables both efficient inference and easing understanding.

Note crucially that such methods as DDP (and iLQR and variants) provide a complete feedback policy and an estimate of the cost-to-go that depends on the state we arrive in. Equivalently, in probabilistic terms, they provide a sequence of ancestral conditional distributions rather than merely marginals. [11]

We begin below by describing a Laplace approximation based sampler. It assumes a differential dynamic programming procedure has already been called that provides gain matrixes and control biases, as well as curvatures. The notation here matches that used by Tassa et al. [2012].

```
# Take an initial state x_0, a model of forward dynamics f: state, action -> state,
# a sequence [] of Quu curvature approximations (conditional partition functions), and
# gain matrices/vectors K,k computed via a Differential Dynamic Programming
# method, and a maximum time T
def Sample(x_0, f, Quu, K, k, T):
    x = [x_0] # state sequence
    u = [] # action sequence
    for t in range(T):
        π = Normal(k[t] + K[t] x[t], Quu(t)^-1)
        u.append(Sample(π))
        x.append(f(x[t],u[t]))
    return (x,u)
```

LQR forward sampler

We note however, that in the spirit of receding horizon, model-predictive control, we can actually do better. Once we have begun sampling, we can re-linearize and resolve the optimal control problem that results from arriving at state $x_1$. This suggests an $O(T^2)$ procedure which samples ancestrally and recomputes the optimal gain matrices and curvatures as it advances. We outline this in pseudo-code below.

```
# Take an initial state x_0, a model of forward dynamics f: state, action -> state,
# a cost function cost: state, action -> R,  a maximum time T,
# and a procedure DDPSolve: f, cost, state, integer --> a tuple of
# ([gain Matrix K],[gain vector k],[quadratic approximation of Q_uu] ) , each a time varying
# sequence computed via a Differential Dynamic Programming method.
# It returns a single sample trajectory of states and controls.
def Sample(x_0, f, cost, T, DDPSolve):
    x = [x_0] # state sequence
    u = [] # action sequence
    for t in range(T):
        (k,K,Q)= DDPSolve( f, cost, x[-1], T) # compute optimal gain matrices
        π = Normal(k[0] + K[0] x[0], Quu[0]^-1)
        u.append(Sample(π))
        x.append(f(x[0],u[0]))
    return (x,u)
```

Re-linearized MPC LQR forward sampler

In the above procedure, care must be taken with passing around a context $\Gamma$: the context should remain unchanged during the sampling and all time-varying costs must be indexed according to the correct time, as the *DDPSolve()* routine is unaware that the time-steps are changing in the outer loop of the algorithm.

[10] D. H. Jacobson and D. Q. Mayne. *Differential Dynamic Programming*. Elsevier, 1970; C. G. Atkeson. Using local trajectory optimizers to speed up global optimization in dynamic programming. In *Advances in Neural Information Processing Systems (NIPS)*, 1994; and Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4906–4913. IEEE, 2012

[11] This point turns out to be an important difficulty in creating constraints in LQR or in probabilistic inference algorithms— in either case many methods lose they key advantage of a feedback policy and regress to only optimization. LQR is not only optimization– it's a full policy. This point is addressed further below.

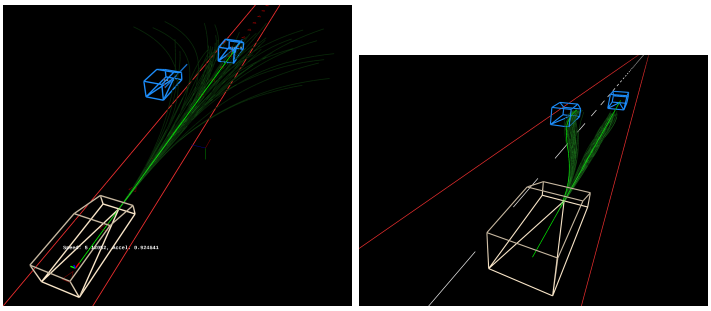

Figure 7.2.1: Both images depict sample trajectories drawn from a Gaussian approximation, computed via Differential Dynamic Programming, of the maximum entropy distribution for a an autonomous vehicle. The left image shows the naive Gaussian sampling procedure. Note the expansion in time at approximately $O(\sqrt{T})$ of the lateral position of the vehicle. The red boundaries shown, which are expensive to violate, do not impact the mode trajectory, and therefore they do not affect the Gaussian sampling approximation. In the right figure, we explore the use of a sampler that iteratively re-linearizes and resolves after each step in ancestral sampling from $x_0$. By contrast, we see two basins (corresponding to lanes), and we note the lateral position of the trajectories is strongly influenced by boundaries.

### *Learning cost functions from samples*

We have already seen how a straightforward gradient can be computed from the difference of $E_p[\phi]$ and its empirical version $E_{\bar{p}}[\phi]$. Let's consider expanding to a space of functions and learning using samples. We begin by replacing $\lambda^T \phi(x, u)$ by $\text{cost}(x, u)$ from a linear space of cost functions (that is, closed under linear combination).

There are three general strategies for learning such cost functions, and perhaps surprisingly, they are all actually closely linked. The first two can be understood as generically gradient descent in a space of functions (1) Boosting and (2) Kernel Gradient Descent, while the final one (3) parametric gradient descent in a function class, is the older and at times most computationally efficient approach. [12]

Taking the MaxEnt family $p(\xi) \propto \exp -cost(\xi)$ and plugging it into the maximum entropy problem Equation 7.2.4 yields a maximum likelihood problem over the space of functions. [13] An unbiased estimate of that gradient can be computed in approach (3) as

$$\nabla_\theta \text{cost}(\xi_i) - \nabla_\theta \text{cost}(\xi_{\text{sample}}) \tag{7.2.8}$$

where $\xi_i$ is a demonstration sample and $\xi_{\text{sample}}$ is drawn from the model $p(\xi)$. Note the cost function being additive over time, this turns into a batch of updates, one for each time-step. The functional versions of this procedure simply generate datapoints indexed by $x, u$ and a positive or negative regression target. This is demonstrated below for a particular variant of boosting.

The result is at heart a two-sample test and update procedure for learning cost functions that is essentially equivalent to the Learning to Search procedure outlined. We use here samples of model behavior rather than simply the most probable/optimal trajectory as in the original algorithm description. The approach is also closely connected to *Generative Adversarial Models* (GANs). The key difference is that GANs, unlike the exponential family or IOC methods, don't solve for the optimal generator in closed form, but instead update an approximate generator. As a result, inference requires only forward execution of the generator model and may be better behaved for singular/constrained-to-a-manifold distributions. The cost is that inference doesn't enable building in engineered constraints or cost-function insight.

*Solving the game* A critical note is that we are indeed solving a game between generator/policy and discriminator/cost function so we must carefully control step size in learning a cost function. Thus it is particularly

Kernel gradient descent is nearly the same as boosting with a particular class of weak learners, while in a particularly important, infinite-width limit, parametric gradient descent on a deep, non-convex function class behaves precisely as a kernel gradient descent including convergence to a global optimum and appropriate regularization guarantees.

Arthur Jacot-Guillarmod, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 8580–8589. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/8076-neural-tangent-kernel-convergence-and-general.pdf

[13] Check it!

important in the family of optimization algorithms above to take constrained steps, or regularize to prior solutions. The optimization objective for a fixed inference policy is linear, and hence will run away to infinity in the cost function update. This is the classic problem of a learning in a game– we have a stable strategy applied for the outer player while the inner player (policy generation) is best response. Boosting style methods do this automatically via their additive model and step size control.

### *Structured Cost Families*

It's quite natural to write down parametric families of cost functions like $\phi(x,u) = \max^2(0, x_i - g_\theta(x_0, \Gamma))$ (for a context variable $\Gamma$) where we are attempting to constrain the form of the objective function to be nicely behaved (single global optimum, strong curvature, multiple derivatives everywhere) in a variable of interested. This encourages inference to remain efficient and enables building in engineering insight in cost-function design.

## *7.3 LEARCH generalized.*

In the previous chapter, we discussed a *functional gradient* approach to solving the Structured Maximum Margin formulation of Inverse Optimal Control. That general non-parametric learning strategy is equally applicable to the MaxEnt framework, as is a parametric "deep variant" enabled through backpropagation.

We may step through the LEARCH-MaxEnt 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 a sample plan/policy from our approximate MaxEnt distribution; (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) recomputing a cost function as a (weighted) sum of the learned predictors.

```
# Take a sequence of trajectory optimization problems and demonstrations [(M_i, ξ_i)]_{i=1}^N where MDP M is a
      planning problem consisting of states, actions, and a transition function used for planning,
# feature function f: state, action -> R^d that describes states in terms of features meaningful for
      cost
# α, a step-size (which can be generalized to a shrinking sequence)
# Relies on procedures to initialize the cost function,
# and to build an optimal maxent policy, and
# to sample that policy

def MaxEntLEARCH({(M_i, ξ_i)}_{i=1}^N, f, α):
  s_0 = init() #Initialize cost function, s_0 : R^d → R
  for t in range(T): # run for T iterations
       D = [] # Initialize the training data set to empty
       for i in range(N): # for each example in the data set
            c_i = s_t(f) # Compute cost function for this problem
            π_i^* = Optimize(M_i, c_i) # find (approximately) the resulting MaxEnt policy π_i^*(x)
            #      that leads to p(ξ) ∝ exp Σ_{(x,u)∈ξ} c_i(x,u)
            μ^* = Sample(M_i, π_i^*)
            # μ^*'s contains the state action pairs from a random trajectory created by the Sampler.
            # More sophisticated samplers might interleave sampling and "optimization".
            # Generate positive and negative training examples:
       D_i^pos = [ (f_i(s,a), 1) for (s,a) in μ^* ]
       D_i^neg = [ (f_i(s,a), -1) for (s,a) in ξ_i ]
       # if the same state occurs in both samples, we can remove it
       D.append(D_i^pos)
       D.append(D_i^neg)
```

```
        h_t = Learn(D) # Train a regressor (or classifier) h_t: R^d->R on the resulting data set
        # The Data Aggregation is not required but does at times lead to more stable performance
        s_{t+1} = s_t + α_t h_t # Update the hypothesis cost function
    return s_T
```

MaxEnt LEARCH Algorithm Pseudo-code

As with the previous LEARCH algorithm, we initialize the algorithm by guessing at a cost function: for instance, by assuming a constant cost everywhere. Instead of a planner, we run the sampler that generates trajectories. We can identify where the sample path agrees and disagrees with a demonstration by a teacher of the correct path. Again, we create a data-point that contains the features that describe the state and assign it a target value to increase or lower the cost depending on whether the sample or the teacher's path traverses that location.

The same procedure is run for locations of disagreement across multiple trajectories (that is multiple planning problems). The resulting data set is then handed to a supervised learning algorithm (linear regression, Support Vector machines, a neural network) that produces a new predictor which maps features to a scalar cost.

As with any boosting style algorithm, the proposed cost function is simply the old cost function added to the new predictor, and we continue to update it by adding in new components.

### *MaxEnt Relation to Maximum Margin Planning*

If we consider the limiting case of $\text{cost}(\xi)_{\text{temp}} = \frac{\text{cost}(\xi)}{T}$ as $T \to 0$, and use the gradient/boosting rules above we recover the max-margin approach to cost function generation. This approximation is less robust (although can prove very useful!) as it tends to lead to cost function collapses. Intuitively, this occurs as we see demonstrations that are highly sub-optimal and we can only generate optimal samples. These will tend to be lower in every element of the feature vector $\phi(x, u)$ and hence the gradient will continue to shrink the cost. No cost shrinkage, however, leads to higher entropy behavior, and thus the costs can collapse. As such, for sub-optimal demonstrations, Maximum Entropy should be preferred whenever it is appropriate. The LQR/Laplace approximation dramatically increases the places where that is possible, as sampling from the model is no more expensive than optimization.

## *7.4 Anomaly Detection*

We can identify important actions (or trajectories) by computing

$$\log p(\xi_{\text{example}}) = -\text{cost}(\xi_{\text{example}}) - \log \sum_{\xi} \exp\left(-\text{cost}(\xi)\right)$$

where the second term comes from the normalizing constant $Z$ and is the softmin of the path costs. A rough and ready approximation is to simply compute

$$\log p(\xi_{\text{example}}) \approx -\text{cost}(\xi_{\text{example}}) + \min_{\xi} \text{cost}(\xi)$$

which requires only access to an optimal planner rather than sampling or computing the complete partition function. Demonstrations that have highly negative log-probabilities should be considered as outliers— the model poorly captures the behavior.

*Refined estimates via Laplace Approximation* We can, of course, also use the Laplace (Gaussian) approximation to estimate how many standard deviations a particular control variable is from from the expected (mode/mean) control. This also makes it easy to make more refined analysis of outliers and detect plans that are, say, $k$-standard deviations away from the mean in one control or state variable axis. For instance, if we have a robotic system with a clear longitudinal control mode, we can identify points that are $3\sigma$ away in negative longitudinal control. Such anomalies (hard deceleration) can be important to identify.

We can also plot (or statistically test) the Gaussian approximation of any state variable and test how far wrong our current model is on any individual state variable.

*Gradients estimates.* Given we know that $\phi(x_{\text{example}}, u_{\text{example}}) - \phi(x_{\text{sample}}, u_{\text{sample}})$ is an unbiased estimate of the gradient, we know it should on average be 0 if our model is well fit. Plots of the elements of these gradients provide clear signal of model under-fitting or failure to set costs appropriately.

*Forecasting.* 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.

## *7.5 Test-time Costs and Constraints*

An important power of the approach of model-based IOC, as opposed to naive policy learning techniques, is the freedom to add new constraints and cost functions not present in the training data or that are critical to enforce at test time. Said differently, when generating trajectories for use in anger, we may wish to further shape these beyond what is represented directly in the data.

The planning-based approach allows a powerful combination of engineering design and machine learning integrated through test-time optimization. The price we pay, of course, is in needing to solve a potentially complex optimization problem at test time, and the unfortunate fact that efficient 2nd order dynamic programming solutions aren't available in industry-standard *differentiable programming frameworks* like Tensorflow or Torch.

We also note that it is often important that test time inference actually be the most probable trajectory, or at least the temperature lowered dramatically in sampling. This is imperative for a few reasons, notably that: a) we usually want the best solution at test time, not a sample one and b) likelihood and

entropy based models are strongly incentivized to "cover" and explain the data available to them rather than simply provide the optimal generation. [14]

[14] Massimo Caccia, Lucas Caccia, William Fedus, Hugo Larochelle, Joelle Pineau, and Laurent Charlin. Language gans falling short. *CoRR*, abs/1811.02549, 2018

### *Policies, Probabilities, and Constraints*

An important part of both the Bellman Equation (in its many instantiations including LQR) and the MaxEnt Inverse Optimal Control formalism is that it computes policies rather than trajectories. A notable difficulty arises in combining either technique with hard constraints. We can use primal-dual (lagrangian) methods for identifying the most probable trajectory, but both feedback policies and conditional distributions break down with these techniques. It's unclear if it's possible to achieve the best combination of primal-dual and policy/probabilistic approaches. This failure tends to privilege reparameterization based techniques as both feedback and uncertainty can still be sensible. These tend to require more careful design than primal-dual methods, at least for strictly enforcing constraints.

## *Acknowledgements*

We thank colleagues Brian Ziebart, Arun Venkatraman, and Wen Sun for tremendous insight in this area and many productive conversations. The images of sampling and the implementation used for differential dynamic programming are due to Arun Venkatraman.