Proceedings of Machine Learning Research vol 144:1–13, 2021

SEAGuL: Sample Efficient Adversarially Guided Learning of Value Functions

Benoit Landry BLANDRY@STANFORD.EDU
496 Lomita Mall, Stanford, CA, 94305

Hongkai Dai HONGKAI.DAI@TRI.GLOBAL
4440 El Camino Real, Los Altos, CA, 94022

Marco Pavone PAVONE@STANFORD.EDU
496 Lomita Mall, Stanford, CA, 94305

Abstract

Value functions are powerful abstractions broadly used across optimal control and robotics algorithms. Several lines of work have attempted to leverage trajectory optimization to learn value function approximations, usually by solving a large number of trajectory optimization problems as a means to generate training data. Even though these methods point to a promising direction, for sufficiently complex tasks, their sampling requirements can become computationally intractable. In this work, we leverage insights from adversarial learning in order to improve the sampling efficiency of a simple value function learning algorithm. We demonstrate how generating adversarial samples for this task presents a unique challenge due to the loss function that does not admit a closed form expression of the samples, but that instead requires the solution to a nonlinear optimization problem. Our key insight is that by leveraging duality theory from optimization, it is still possible to compute adversarial samples for this learning problem with virtually no computational overhead, including without having to keep track of shifting distributions of approximation errors or having to train generative models. We apply our method, named SEAGuL, to a canonical control task (balancing the acrobot) and a more challenging and highly dynamic nonlinear control task (the perching of a small glider). We demonstrate that compared to random sampling, with the same number of samples, training value function approximations using SEAGuL leads to improved generalization errors that also translate to control performance improvement.

Keywords: Adversarial learning, Value function approximation, Trajectory optimization, Model predictive control

1. Introduction

Value functions are powerful tools in machine learning and robotics as they encode the optimal cost of moving from each state to some desired goal and hence, can form the basis of many types of controllers. There has been extensive work in learning value function approximations using neural networks, including (Silver et al., 2016; Mnih et al., 2013; Deits et al., 2019). One approach to learning value functions is to first generate a set of samples, where each sample contains a pair of initial state and its associated optimal cost, and later train a neural network on the state-value pairs to produce an approximate value function. There exists several approaches to computing the optimal cost for a given state. When the system dynamics are unknown or stochastic, a typical approach consists of estimating it using Monte Carlo simulation starting from that state (Silver et al., 2016; Sutton and Barto, 2018). Alternatively, when the system dynamics are known and deterministic, the optimal cost can be computed (to a small integration error in the continuous dynamics case) using trajectory optimization, or equivalently optimal control (Betts, 2010; Kelly, 2017). In this work, we assume that the system dynamics are given and deterministic, and solve a trajectory optimization problem to compute each state-value pair. We then learn a value function from these pairs, allowing us to then synthesize a model predictive controller (MPC) from the learned value function.

One major difficulty of this approach is that generating samples of state-value pairs can be time consuming. For most systems, each sampled state-value pair involves solving an optimal control problem as a nonlinear optimization problem (NLP), which could take anywhere from seconds to hours for some complicated problems (Bertsekas, 1997; Schultz and Mombaur, 2009). We are hence motivated to decrease the total computation time, by reducing the number of sampled state-value pairs while retaining good accuracy of the learned value function approximation. To this end, we introduce SEAGuL, an algorithm that leverages ideas from adversarial training (Madry et al., 2017; Wong et al., 2020; Kurakin et al., 2016), and seeks adversarial samples at which the learned value function incurs large approximation error. Similar to many reinforcement learning algorithms, SEAGuL trains a value function approximation while also generating data, and uses feedback from the learning process to adapt its sampling. By prioritizing adversarial samples while generating training data, the training process penalizes more heavily where the neural network approximation error is large, and hence learns a more accurate approximation with fewer training samples.

However, there exists a major challenge to applying existing adversarial training approaches to our problem. Most commonly, adversarial training is used in classification problems, for which loss functions are closed form functions of the neural network input (for example, a negative log-likelihood loss, or an l_2 loss). A typical adversarial training algorithm computes the gradient of the loss function w.r.t the input sample, and then applies projected gradient ascent to move the input sample along the direction of that gradient and obtain a more adversarial sample (one with larger loss). In contrast, in our problem the loss function itself contains another optimization problem, namely the optimal control problem starting from the sampled state (in this optimization problem the sampled state is not a decision variable but a parameter); as a result, the loss function is *not* a closed form expression of the input sample, and computing the gradient of the loss function might appear to be difficult at first glance. Our key insight is that this gradient can be readily obtained from the result of the optimization itself simply by leveraging duality theory (Bertsekas, 1997). Our work therefore demonstrate that it is possible to apply adversarial training methodologies to learning value function approximations with negligible computational overhead. We list our contributions as follows,

1. we demonstrate how to easily compute adversarial samples for a value function learning algorithm by solving a nested optimization problem known as a bilevel optimization problem,
2. we demonstrate how those samples can improve the convergence of a simple supervised learning method for learning value function approximations,
3. we show how this improved convergence has meaningful impact on the quality of the resulting controllers, specifically in the context of model predictive control.

2. Related Work

Several approaches have been proposed that use trajectory optimization in order to learn value functions and policies. For example, (Zhong et al., 2013) proposed to learn a value function using the iterative LQG method and use the learned value function as terminal cost of an MPC problem in order to shorten the horizon of the controller. More recently, (Deits et al., 2019) proposed a similar approach but where the value function is trained using upper and lower bounds on the objective of a control problem solved as a mixed-integer convex program. Note that unlike our proposed method, neither of these directly leverage the optimization problem itself in order to guide the sampling, and they instead rely on a mix of random sampling and of sampling of states visited by the learned controller, similar to DAGGER (Ross et al., 2011). Moreover, our proposed method does not preclude the use of these existing sampling schemes, and could be combined with them in various ways.

There has also been substantial work done in leveraging adversarial learning ideas in order to speedup convergence of various learning algorithms. Some existing work in adversarial imitation learning looks at improving the sampling efficiency of these learning approaches, but generally rely on generative models to do so (Ho and Ermon, 2016; Torabi et al., 2019). Perhaps most similar to our approach is Prioritized Experience Replay (Schaul et al., 2015), where a distribution of the largest *temporal difference* (TD) error is estimated during training and used for sampling. However the problem formulation is slightly different in our case, where our training data comes from solving nonlinear trajectory optimization problems, and is therefore more costly (but more informative) on a per-sample basis. Adversarial learning has also been investigated in the context of improving robustness to parameter variation as in (Pattanaik et al., 2018; Mehta et al., 2019). These methods are similar to ours, but they once again do not leverage a trajectory optimization formulation of the underlying optimal control problem like our proposed approach does. Moreover, they generally introduce additional machinery such as estimates of shifting distributions or generative models, which our method does not.

Finally, our algorithm relies on solving a special class of optimization problems known as bilevel optimization problems. Previous work in machine learning has investigated the incorporation of convex optimization problem into the learning pipeline (Amos and Kolter, 2017; Amos et al., 2018). Specifically, the lower level problem of our bilevel programs are nonlinear optimization problems, an especially challenging type of problems that has been investigated in the context of robotics notably by (Landry et al., 2019) and (Chen and Posa, 2020), but with different solution methods.

3. Problem Formulation

Given a system with state $x \in \mathcal{X}$, control input $u \in \mathcal{U}$ and dynamics captured by a nonlinear differential equation, the optimal control problem defines an integrating stage cost function c which must be minimized while respecting both dynamics and problem-specific constraints on the state and input trajectories of the system. Those constraints generally include desired initial and final configurations as well as actuation limits, and any other problem specific constraints such as obstacle avoidance.

Direct methods are a class of solution methods that discretize finite horizon optimal control problems and casts them as nonlinear programs. Formally, the direct method trajectory optimization problem has the form

$$\begin{aligned} V(\mathrm{x}_{\text{init}}, \tau) = \underset{x_i, u_i, h_i; i=0 \dots m}{\text{minimize}} \quad & \sum_{i=0}^{m-1} c(x_i, u_i, h_i) + c_f(x_m, u_m, h_m) \\ \text{subject to} \quad & g(\mathbf{x}, \mathbf{u}, \mathbf{h}) \leq 0, \\ & h(\mathbf{x}, \mathbf{u}, \mathbf{h}) = 0, \\ & x_0 = x_{\text{init}}, \\ & \mathbf{e}^T \mathbf{h} = \tau \end{aligned} \tag{1}$$

where the decision variables are way-point states along the trajectory $\mathbf{x} := [x_0, x_1, \dots, x_m]$, way-point control inputs $\mathbf{u} := [u_0, u_1, \dots, u_m]$ and time between each way-point $\mathbf{h} := [h_0, h_1, \dots, h_m]$. c and c_f are the additive and terminal costs respectively. g and h are nonlinear inequality and equality constraints such as control input limits, obstacle avoidance or kinematic constraints, and $\mathbf{e} := [1, 1, \dots, 1]^T$. x_{init} denotes the given initial state. For all of our problems, we also constrain the "time-to-go" τ of our optimal control problems. Note that x_{init} and τ are **parameters** of the optimization problem, and manifests as the right-hand side of equality constraints, which plays an important role in duality theory. $V(\mathrm{x}_{\text{init}}, \tau)$ is the optimal cost-to-go, or *state value function*, starting from x_{init} with τ time remaining. For the remainder of the paper, unless specified, we will write $V(\mathrm{x}_{\text{init}}) := V(\mathrm{x}_{\text{init}}, \tau)$, since the treatment of τ is largely similar to the treatment of x_{init}. Note that computing the solution to $V(\mathrm{x}_{\text{init}}, \tau)$ also gives us the solutions to $V(\mathrm{x}_{\text{init}}, \tau_k)$ for $k = 0 \dots m-1$ with $\tau_k = \sum_{i=0}^{i=k} h_i$ through Bellman's principle of optimality (Kirk, 2012). We can therefore focus exclusively on generating state-value pairs for larger values of $\tau \in \mathcal{T}_0$ and use the state-value pairs produced (as by-products) for the smaller values of τ to train a time-varying value function approximation. The details of direct method trajectory optimization are beyond the scope of this document and we refer the reader to (Kelly, 2017) for a good introduction. However, we stress that nonlinear trajectory optimization has repeatedly been shown to be applicable to a *broad* range of important robotics problems, from high degrees of freedom humanoid (Dai et al., 2014) to control of reusable rockets (Ma et al., 2019).

In this work, we are interested in training a differentiable function approximator, most predominantly a deep neural network, such that it approximates $V(\mathrm{x}_{\text{init}})$. More formally, we are looking for $\tilde{V}_\theta(\mathrm{x}_{\text{init}})$, a differentiable function parametrized by θ, with the objective

$$\min_\theta \mathbb{E}_{\mathrm{x}_{\text{init}}} \sum_{m=0}^{M} (V(\mathrm{x}_{\text{init}}) - \tilde{V}_\theta(\mathrm{x}_{\text{init}}))^2. \tag{2}$$

Without loss of generality, we assume that the initial state x_{init} is subject to a uniform distribution within an admissible set $\mathcal{X}_0$. Even though our general approach has other interesting use cases, we are especially motivated by learning approximations (like trained deep neural networks) that are orders of magnitude faster to evaluate than solving the original nonlinear trajectory optimization problem (1), therefore enabling their use as part of high rate feedback controllers.

4. Limited lookahead MPC

There are numerous ways in which a learned value function $\tilde{V}_\theta(\mathrm{x}_{\text{init}})$ can be used to perform control efficiently. Here, we highlight an important one which we utilize for all of our experiments below:

limited lookahead model predictive control (Bertsekas, 1995; Zhong et al., 2013). This method relies on using the learned value function as a way to shorten the horizon of a model predictive control algorithm without reducing its efficacy. Namely, given the original problem (1), we pick $n \ll m$, and at every time step we solve the following optimization problem

$$\begin{array}{ll} \underset{x_i, u_i, h_i; i=1 \ldots n}{\text{minimize}} & \sum_{i=1}^{n} c(x_i, u_i, h_i) + \tilde{V}_\theta(x_n, \tau - \mathbf{e}^T \bar{\mathbf{h}}) \\ \text{subject to} & \bar{g}(\bar{\mathbf{x}}, \bar{\mathbf{u}}, \bar{\mathbf{h}}) \leq 0, \\ & \bar{h}(\bar{\mathbf{x}}, \bar{\mathbf{u}}, \bar{\mathbf{h}}) = 0, \\ & x_0 = x_{\text{current}}, \\ & \tau = \text{t}_{\text{togo}} \,. \end{array} \tag{3}$$

where $\bar{\mathbf{x}} := [x_0, x_1, \ldots, x_n]$, $\bar{\mathbf{u}} := [u_0, u_1, \ldots, u_n]$, $\bar{\tau} := [t_0, t_1, \ldots, t_n]$, and x_{current}, t_{togo} are the state and time-to-go at which the controller is being evaluated. This problem has much fewer decision variables than the original trajectory optimization problem (1), and hence can be solved online at a high rate. Our overall approach to using the learned value function is similar to (Hoeller et al., 2020).

We note that unlike many methods that attempt to speed-up model predictive controllers by synthesizing equivalent closed form policies (Kaufmann et al., 2020; Carius et al., 2020), limited lookahead MPC has the benefit that it preserves the full expressiveness of the original problem formulation over a short horizon. This has the advantage, among other things, of allowing the controller to include unforeseen constraints (e.g. additional obstacles to avoid) at run time.

Finally, note that the resulting optimization problem can be solved by a number of nonlinear optimization methods, including popular gradient-free ones like cross-entropy methods (Botev et al., 2013).

5. Adversarial Training

Even though (2) suggests that a simple supervised learning approach to this problem should work, and has in fact already been demonstrated with simple systems (Zhong et al., 2013), the cost of solving a single instance of a trajectory optimization problem to compute $V(x_{\text{init}})$ for that single sampled initial state x_{init}, mixed with the high number of samples required to produce a good approximations, can render the approach impractical in many cases. Specifically, our method addresses the two competing requirements of learning value functions through supervised learning, namely that (i) the approximation must be good over a large set of states and (ii) the generation of the training data must remain tractable.

We propose learning $\tilde{V}_\theta(\text{x}_{\text{init}})$ by training using a smaller number of *adversarial* samples. We define an adversarial sample x_{adv} as a sample that has a comparatively large difference between its true value and the value of the function approximation i.e. $|V(x_{\text{adv}}) - \tilde{V}_\theta(x_{\text{adv}})| \gg 0$. Our approach generates adversarial samples by solving

$$\begin{aligned} x_{\text{adv}} &= \arg\max_{\text{x}_{\text{init}}} \quad \frac{1}{2}(V(\text{x}_{\text{init}}) - \tilde{V}_\theta(\text{x}_{\text{init}}))^2, \\ &\quad\ \text{s.t.} \quad \text{x}_{\text{init}} \in \mathcal{X}_0, \\ &= \arg\max_{\text{x}_{\text{init}}} \quad \frac{1}{2}(\min_{\mathbf{x},\mathbf{u},\mathbf{h}} \sum_i c(x_i, u_i, h_i) + c_f(x_m, u_m, h_m) - \tilde{V}_\theta(\text{x}_{\text{init}}, \tau))^2, \\ &\quad\ \text{s.t.} \quad g(\mathbf{x}, \mathbf{u}, \mathbf{h}) \leq 0, \\ &\qquad\quad\ h(\mathbf{x}, \mathbf{u}, \mathbf{h}) = 0, \\ &\qquad\quad\ x_0 = \text{x}_{\text{init}}, \\ &\qquad\quad\ \mathbf{e}^T \mathbf{h} = \tau, \\ &\qquad\quad\ \text{x}_{\text{init}} \in \mathcal{X}_0, \tau \in \mathcal{T}_0, \end{aligned} \tag{4}$$

to suboptimality, where $\mathcal{X}_0$ is a set of admissible initial states.

Note that solving (4) is far from trivial, and this approach now requires us to solve this additional optimization problem on top of our original learning problem (2). However, this is greatly mitigated by the specifics of our algorithm. First, we only solve the adversarial problem (4) to suboptimality using a few iterations of a projected gradient ascent method (in practice one or two iterations often appear to suffice), i.e., in each iteration we compute the gradient of the cost function $\partial \left(\frac{1}{2}||V(\text{x}_{\text{init}}) - \tilde{V}_\theta(\text{x}_{\text{init}})||^2\right) / \partial\, \text{x}_{\text{init}}$ in (4), and move x_{init} along that gradient direction for a small step. Additionally, this method ensures that every individual **iterate** of each instance of the adversarial problem generates a valid sample pair $(x_{\text{adv}}, V(x_{\text{adv}}))$, which is computation that would have been required of any supervised training approach to (2). It is tempting to think that generating adversarial samples is a lot more expensive than randomly sampling the initial states in $\mathcal{X}_0$. We wish to clarify here that since obtaining the gradient of the cost in (4) is effortless (as will be shown shortly), the computational cost is the same between training with adversarial samples versus with random samples, as they both require solving the same number of trajectory optimization problems so as to generate the same number of state-value pairs.

In order to take a gradient step to obtain a new adversarial sample x_{init}, we need to compute the gradient of the loss function in (4), which eventually requires computing the gradient $\partial V(\text{x}_{\text{init}})/\partial\, \text{x}_{\text{init}}$. The challenge is that $V(\text{x}_{\text{init}})$ is *not* a closed-form function of x_{init}; instead it is the optimal cost of another minimization problem (1), where x_{init} is a parameter of the optimization problem. As a result, problem 4 as a whole is known as a *maxmin* problem or more generally as a *bilevel optimization problem* (Colson et al., 2007; Landry et al., 2019). Bilevel optimization problems are a class of problems where an optimization is embedded in another optimization, either as part of its constraints or objective. The embedded problem is usually referred to as the lower-level optimization, and the resulting nested optimization as the upper-level optimization. We refer the reader to (Colson et al., 2007; Bard, 2013) for a more complete treatment of bilevel optimization.

Our key insight is that according to the duality theory (Bertsekas, 1997), the gradient of the cost function in the bilevel optimization problem (4) can be readily obtained when the lower-level optimized trajectory is solved, by using the dual variable values associated with the constraint $x_0 = \text{x}_{\text{init}}$:

$$\frac{\partial V(\text{x}_{\text{init}})}{\partial\, \text{x}_{\text{init}}} = \lambda^* \tag{5}$$

where λ^* is the dual variable (or *shadow price*) associated with the constraint $x_0 = \mathrm{x}_{\text{init}}$. Most nonlinear optimization solvers (e.g., SNOPT Gill et al. (2005b), IPOPT Wächter and Biegler (2006)) report both the primal solution ($\mathbf{x}$ in (1)) and the dual solution when the solver finishes, hence by solving the nonlinear optimization problem in (1), we not only compute the value of $V(\mathrm{x}_{\text{init}})$, but also its gradient w.r.t the initial state, and therefore we can readily compute the gradient of the cost in (4) as

$$\left(V(\mathrm{x}_{\text{init}}) - \tilde{V}_\theta(\mathrm{x}_{\text{init}})\right)\left(\frac{\partial V}{\partial \mathrm{x}_{\text{init}}} - \frac{\partial \tilde{V}_\theta}{\partial \mathrm{x}_{\text{init}}}\right). \tag{6}$$

Note that since $\tilde{V}_\theta(\mathrm{x}_{\text{init}})$ is the output value of the neural network with x_{init} as the input, we can easily compute its gradient $\partial\tilde{V}_\theta/\partial\,\mathrm{x}_{\text{init}}$ using automatic differentiation (Paszke et al., 2017). The entire process of generating adversarial examples is summarized in algorithm 1.

Algorithm 1
Bilevel Adversarial Sample Generation

```
Result: ${x^1_{adv}, x^2_{adv}, \cdots}, {V^1, V^2, \cdots}$
$x^1_{adv} \sim$ UniformSample $\mathcal{X}_0$
$i \leftarrow 1$
while $i \le$ maxSteps do
    $V^i \leftarrow$ solve(1) using $x^i_{adv}$ as $\mathrm{x}_{init}$
    $\partial V^i/\partial \mathrm{x}_{init} \leftarrow \lambda^*$
    $x^{i+1}_{adv} \leftarrow \mathrm{project}_{\mathcal{X}_0}(x^i_{adv} + \alpha\partial V^i/\partial \mathrm{x}_{init})$
    if $||x^{i+1}_{adv} - x^i_{adv}||^2 \le \epsilon$ then
        return
    end
    $i \leftarrow i + 1$
end
```

Algorithm 2
SEAGuL: Sample-Efficient Adversarially Guided Value Learning

```
Result: $\theta$, such that $\tilde{V}_\theta(x) \sim V(x)$
buffer $\leftarrow \emptyset$
$i \leftarrow 0$
while $i \le$ maxIter do
    adv samples $\leftarrow$ algorithm 1
    buffer $\leftarrow$ buffer $\cup$ adv samples
    for $j$ in $1 \ldots$ numBatches do
        batch $\leftarrow$ sample(buffer)
        $\theta \leftarrow$ step($\theta$, batch)
    end
    $i \leftarrow i + 1$
end
```

Using the adversarial example generation described in Algorithm 1, we can now train our neural network approximation as follows. First, we generate a sample x^1_{adv} randomly, then take a number of projected gradient ascent steps. Every gradient ascent step generates a labeled sample x^i_{adv}, V^i, which is (on average) increasingly adversarial. The samples come from the solution to the inner problem of the bilevel nonlinear optimization. Next, we take the samples generated during the bilevel optimization and add them to a sample buffer. We then sample random batches (but usually start with the newly generated samples since they are most adversarial in the network's current state) from this buffer and perform a few iterations of gradient descent to optimize the parameters θ of the neural network. We repeat each step until convergence. The entire process is described in Algorithm 2.

6. Results

We apply our learning algorithm on two well-known robotic control problems: the acrobot and the perching glider. All the trajectory optimization problems are solved using the off-the-shelf

SQP-based nonlinear solver SNOPT (Gill et al., 2005a) through Drake interface Tedrake and the Drake Development Team (2019) and learning is performed with Pytorch (Paszke et al., 2017). For comparison, as a baseline, we train value function approximations using the exact same number of samples, but sampled uniformly over the state space of each system. The starting points for the adversarial sample problems are taken from a randomly selected subset of the samples used by the baseline in order to reduce the effect of chance in the comparison as much as possible. For both methods, the samples are generated and added to the respective datasets at fixed intervals between training steps. The important point to stress here is that both the baseline and the adversarially trained networks are trained by solving the exact same number of nonlinear optimization problems, which is by far the biggest computational bottleneck of the overall approaches.

6.1. Acrobot

We train a neural network to approximate the value function of an optimal control problem for an acrobot. This system consists of a double pendulum that is only actuated at its elbow. The samples are generated using a horizon of 50 time steps of 100ms each. We train a time varying value function using a feedforward neural network with 32 units and one hidden layer. The overall training process ends up generating 100 samples per network and 10,000 training steps. Figure (1) shows the mean squared error for both the baseline and the adversarially trained value functions over 100 validation samples taken from a grid in the state-space (these samples are *not* trained over). 10 samples are generated at every 500 training steps (so the number of training steps also corresponds to the number of samples generated at that point). The validation loss shows that adversarial training converges more quickly than the baseline. Note that both neural networks were initialized with identical parameters, which were initially trained for 100 steps over 10 random samples.

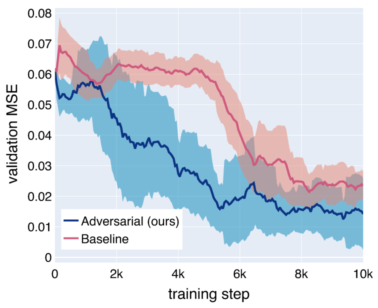

Figure 1: Validation losses during training of value function approximations for the acrobot.

Next, to validate that this improvement on the generalization of the value function leads to tangible improvement in control, we use the learned value functions as final costs for limited lookahead model predictive controllers with horizons of 5 time steps (compared to the 50 time steps used in the data generation). For each controller, we show the distance to the desired state (in which the acrobot is balancing itself pointing upwards) after simulating the system for 5 seconds. Figure (2) shows the performance of both the baseline as well as the adversarially trained approximation after 5,000 training steps (50 samples) and 10,000 training steps (100 samples). As expected, the improve convergence on our value function approximation leads to improved performance of the resulting model predictive controller.

6.2. Perching Glider

We train a neural network to approximate the value function of a more challenging task, which consists of trying to perch a small glider on a string by executing a high pitch maneuver as described in (Cory and Tedrake, 2008) and Figure (3). Specifically, we use a nonlinear trajectory optimization formulation of this task adapted from (Moore and Tedrake, 2012) with only minor modifications. The samples are generated using a horizon of 40 time steps of about 25ms each (variable time steps are permitted in the optimization). We train a time varying value function using the same network

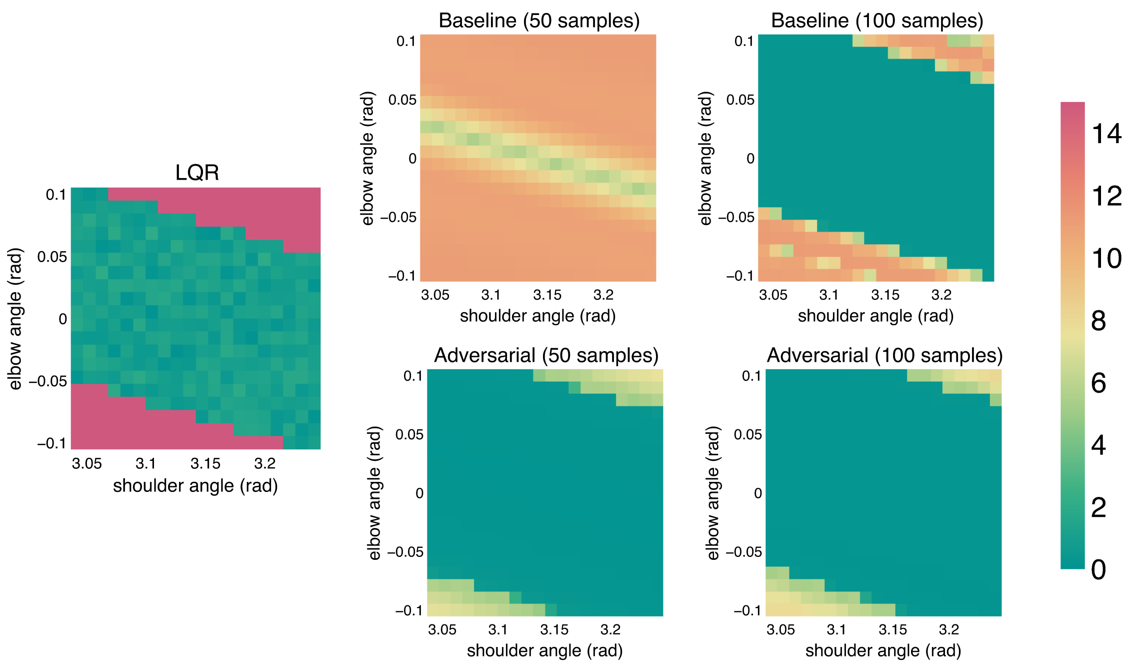

Figure 2: Control performance of the resulting model predictive controllers for the acrobot system for a grid of initial states. The color scale corresponds to the L_2 distance between the final state and the desired setpoint of the controller (with the acrobot balancing itself) after a 5 seconds of simulation. Thus, each point shows the final state error starting from the initial state at that point. Adversarially training the value function approximations leads to faster convergence to useful approximations. We include the performance of LQR on the same task for reference.

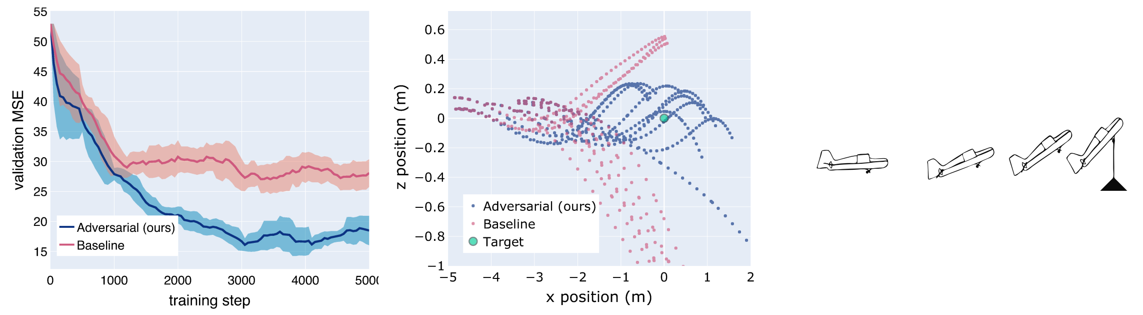

Figure 3: Left: Validation loss during adversarial training of the perching glider system. Center: Resulting trajectories of the perching glider using the model predictive controllers based on adversarially and randomly trained value function approximations, for various initial positions. Right: The perching glider task consists of attempting to land a small glider on a string located in front of it. The task is highly dynamic and requires aggressive control actions. Figure borrowed from (Cory and Tedrake, 2008)

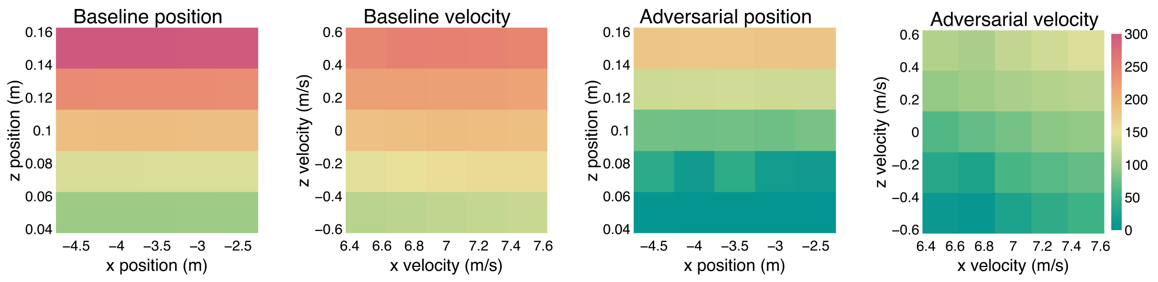

Figure 4: Control performance of the resulting model predictive controllers for the perching glider system for a grid of initial states. The color scale corresponds to the cost of the resulting trajectory for a 1 second simulation. Smaller cost means better trajectory. Each cell represents the trajectory cost starting from the initial state at that point. Again, adversarially training the value function approximations leads to faster convergence to useful approximations.

architecture and hyperparameters as the acrobot, except that since the state space of the glider is too large (7-dimensional) to properly cover with a grid, we build the validation set by combining a coarse grid of 216 samples and an additional 250 random samples. As with the acrobot, Figure 3 (left) shows that training over adversarial samples allows the neural network to converge to a low validation error with fewer training steps and therefore samples.

Figure 4 shows the resulting control performance for limited lookahead model predictive controllers (with about 250ms of lookahead horizon) using the randomly and adversarially trained value function approximations as terminal costs. The color scale corresponds to the cost of the resulting trajectories for each controller. Once again, for a given number of samples, adversarial sampling leads to better performance from the controller. Figure 3 (right) shows a few sampled rollouts from various initial positions.

7. Conclusion

In this work, we introduce the SEAGuL algorithm, and demonstrate how it is capable of easily generating adversarial samples in the context of learning a value function approximation, where the samples are generated by nonlinear trajectory optimization. We show how these samples can be used to more efficiently train value function approximations. Moreover, we demonstrate how these gains are also reflected in improvements of the resulting controllers. We stress that our method is both simple to implement and involves minimal computational overhead. We also note that there are many other ways in which generating similar adversarial samples could be useful, including by combining them with other adaptive sampling schemes such as generative models (the differentiability of our adversarial loss function makes this possible and straightforward). Furthermore, we note that our method can readily be extended to generating adversarial samples with respect to environmental *parameters*, therefore making our approach an attractive avenue to experiment with more principled ways to perform domain randomization, an exciting direction of future work.

References

B. Amos and J. Z. Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *Int. Conf. on Machine Learning*, 2017.

B. Amos, I. D. J. Rodriguez, J. Sacks, B. Boots, and J. Z. Kolter. Differentiable MPC for end-to-end planning and control. In *Conf. on Neural Information Processing Systems*, 2018.

Jonathan F Bard. *Practical bilevel optimization: algorithms and applications*, volume 30. Springer Science & Business Media, 2013.

D. Bertsekas. *Dynamic programming and optimal control*. Athena Scientific, 4 edition, 1995.

Dimitri P Bertsekas. Nonlinear programming. *Journal of the Operational Research Society*, 48(3): 334–334, 1997.

John T Betts. *Practical methods for optimal control and estimation using nonlinear programming*. SIAM, 2010.

Zdravko I Botev, Dirk P Kroese, Reuven Y Rubinstein, and Pierre L'Ecuyer. The cross-entropy method for optimization. In *Handbook of statistics*, volume 31, pages 35–59. Elsevier, 2013.

Jan Carius, Farbod Farshidian, and Marco Hutter. Mpc-net: A first principles guided policy search. *IEEE Robotics and Automation Letters*, 5(2):2897–2904, 2020.

Yu-Ming Chen and Michael Posa. Optimal reduced-order modeling of bipedal locomotion. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8753–8760. IEEE, 2020.

Benoît Colson, Patrice Marcotte, and Gilles Savard. An overview of bilevel optimization. *Annals of operations research*, 153(1):235–256, 2007.

Rick Cory and Russ Tedrake. Experiments in fixed-wing uav perching. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, page 7256, 2008.

Hongkai Dai, Andrés Valenzuela, and Russ Tedrake. Whole-body motion planning with centroidal dynamics and full kinematics. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 295–302. IEEE, 2014.

Robin Deits, Twan Koolen, and Russ Tedrake. Lvis: Learning from value function intervals for contact-aware robot controllers. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7762–7768. IEEE, 2019.

P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Review*, 47(1):99–131, 2005a.

Philip E Gill, Walter Murray, and Michael A Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM review*, 47(1):99–131, 2005b.

J. Ho and S. Ermon. Generative adversarial imitation learning. In *Conf. on Neural Information Processing Systems*, 2016.

David Hoeller, Farbod Farshidian, and Marco Hutter. Deep value model predictive control. In Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura, editors, *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pages 990–1004. PMLR, 30 Oct–01 Nov 2020. URL http://proceedings.mlr.press/v100/hoeller20a.html.

Elia Kaufmann, Antonio Loquercio, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Deep drone acrobatics. *arXiv preprint arXiv:2006.05768*, 2020.

Matthew Kelly. An introduction to trajectory optimization: How to do your own direct collocation. *SIAM Review*, 59(4):849–904, 2017.

Donald E Kirk. *Optimal control theory: an introduction*. Courier Corporation, 2012.

Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

B. Landry, Z. Manchester, and M. Pavone. A differentiable augmented lagrangian method for bilevel nonlinear optimization. In *Robotics: Science and Systems*, 2019.

Lin Ma, Kexin Wang, Zhijiang Shao, Zhengyu Song, and Lorenz T Biegler. Direct trajectory optimization framework for vertical takeoff and vertical landing reusable rockets: case study of two-stage rockets. *Engineering Optimization*, 51(4):627–645, 2019.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher J Pal, and Liam Paull. Active domain randomization. *arXiv preprint arXiv:1904.04762*, 2019.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Joseph Moore and Russ Tedrake. Control synthesis and verification for a perching uav using lqr-trees. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 3707–3714. IEEE, 2012.

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *Conf. on Neural Information Processing Systems - Autodiff Workshop*, 2017.

Anay Pattanaik, Zhenyi Tang, Shuijing Liu, Gautham Bommannan, and Girish Chowdhary. Robust deep reinforcement learning with adversarial attacks. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2040–2042. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Gerrit Schultz and Katja Mombaur. Modeling and optimal control of human-like running. *IEEE/ASME Transactions on mechatronics*, 15(5):783–792, 2009.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Russ Tedrake and the Drake Development Team. Drake: Model-based design and verification for robotics, 2019. URL https://drake.mit.edu.

Faraz Torabi, Sean Geiger, Garrett Warnell, and Peter Stone. Sample-efficient adversarial imitation learning from observation. *arXiv preprint arXiv:1906.07374*, 2019.

Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1): 25–57, 2006.

Eric Wong, Leslie Rice, and J Zico Kolter. Fast is better than free: Revisiting adversarial training. *arXiv preprint arXiv:2001.03994*, 2020.

Mingyuan Zhong, Mikala Johnson, Yuval Tassa, Tom Erez, and Emanuel Todorov. Value function approximation and model predictive control. In *2013 IEEE symposium on adaptive dynamic programming and reinforcement learning (ADPRL)*, pages 100–107. IEEE, 2013.