Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence

Agnostic System Identification for Monte Carlo Planning

Erik Talvitie
Mathematics and Computer Science
Franklin & Marshall College
erik.talvitie@fandm.edu

Abstract

While model-based reinforcement learning is often studied under the assumption that a fully accurate model is contained within the model class, this is rarely true in practice. When the model class may be fundamentally limited, it can be difficult to obtain theoretical guarantees. Under some conditions the DAgger algorithm promises a policy nearly as good as the plan obtained from the most accurate model in the class, but only if the planning algorithm is near-optimal, which is also rarely the case in complex problems. This paper explores the interaction between DAgger and Monte Carlo planning, specifically showing that DAgger may perform poorly when coupled with a sub-optimal planner. A novel variation of DAgger specifically for use with Monte Carlo planning is derived and is shown to behave far better in some cases where DAgger fails.

1 Introduction

Model-based reinforcement learning (MBRL) approaches can typically be understood to consist of two main components: system identification (learning a model from observations) and planning (obtaining a policy from the learned model). While these two problems have been extensively studied separately, combining them raises additional challenges for both. This is especially true in the *agnostic* setting, where it is not assumed that a perfect model exists in the model class. When the model is assumed to be inaccurate, it can be difficult to draw firm connections between its properties and the quality of plans it generates. That said, it is critical to study MBRL in the agnostic setting – perfect models will rarely, if ever, be obtainable in complex, high-dimensional problems of genuine interest.

Ross and Bagnell (2012) point out that the common "textbook" framework for system identification (e.g. Ljung, 1999), in which a single batch of data is used to train a model which is then used to obtain a policy, can fail catastrophically when the plan enters states that the training set did not adequately cover. They introduce the DAgger (Data Aggregator) algorithm, which iteratively generates training sets containing states from a mixture of a given "exploration" distribution (supplied by an expert) and the distribution of states visited by the current policy. This ensures that the model will be accurate both in states that the expert would enter and in states that the generated plans would enter. Ross and Bagnell derive agnostic guarantees for DAgger that, roughly, promise that it will generate a policy that is nearly as good as the plan one would obtain from the most accurate model in the class.

There are, of course, many MBRL algorithms that could, in principle, be applied in the agnostic setting, though relatively few provide performance guarantees in that case. Schoknecht (2002), Parr et al. (2008), and Sutton et al. (2008) all point out the equivalence between the optimal value function in an approximate linear model and an approximate linear value function learned without a model. This observation sheds light on what solution will be yielded by linear models in the agnostic setting, but offers no performance guarantees regarding that solution. Sorg, Lewis, and Singh (2010) and Joseph et al. (2013) both address the agnostic setting by adapting model parameters to directly improve agent performance (rather than model accuracy) via policy gradient algorithms. As such they inherit the strengths and weaknesses of gradient approaches – guaranteed convergence, but to a locally optimal model parameterization.

In contrast to the above approaches, DAgger is able to provide general and strong agnostic performance guarantees. However, though the performance bound is agnostic to the model class, the tightness of the bound relies upon the assumption that the *planner* is near optimal. Furthermore, a near-optimal plan must be computed at every iteration of the algorithm. In sufficiently complex problems, obtaining even a single near-optimal plan is typically infeasible.

This paper considers the interaction between the DAgger algorithm and Monte Carlo planning, a popular family of methods that can offer good performance with computational complexity that is independent of the size of the state space. This paper will focus on the simple one-ply Monte Carlo planning algorithm, which has its roots in the work of Tesauro and Galperin (1996). At every step the one-ply Monte Carlo algorithm estimates the quality of each action by averaging the returns of several sample rollouts that begin with the action and follow some rollout policy π_r thereafter. We will see that, when coupled with this planner, the DAgger algorithm can fail dramatically. While the model is trained to be accurate in states the expert and the execu-

tion policy would encounter, it *is not* trained to be accurate in states *rollouts* would encounter. This can lead to persistent model inaccuracies that harm planning performance. To mitigate this effect this paper derives a novel variation of DAgger specifically for use with Monte Carlo planning. The analysis shows that, because it is aware of the planning algorithm's specific flaws, the DAgger-MC algorithm is better able to distinguish between performance loss that is genuinely irreducible and loss that can be reduced by improving the model. A simple empirical illustration (Section 5) demonstrates that these issues can arise in practice and that DAgger-MC can perform well in cases where DAgger fails.

Note that, though the focus here is on the one-ply Monte Carlo algorithm, these issues are also relevant to more sophisticated Monte Carlo tree search (MCTS) planners (Tavener et al. 2012), such as the popular UCT algorithm (Kocsis and Szepesvári 2006). While MCTS algorithms can explore shallow actions more systematically, they still rely upon fixed-policy rollouts for deep exploration and, for a practical number of rollouts, cannot typically promise ϵ-optimality. The possibility of extending these insights to MCTS planners is briefly discussed in Section 6.

2 Preliminaries

We will consider stochastic environments with discrete state, action, and time. For simplicity, assume the environment state is fully observable (Ross and Bagnell discuss how to extend the analysis to the partially observable case). Thus, the environment can be formalized as a Markov decision process (MDP). The initial state $s^0 \in \mathcal{S}$ is drawn from an initial distribution μ. At every step t the agent perceives the current state $s^t \in \mathcal{S}$ and selectes an action $a^t \in \mathcal{A}$. Then the next state s^{t+1} is selected with probability $\Pr(s^{t+1} \mid s^t, a^t)$ and the agent receives some reward $r^{t+1} \in [R_{min}, R_{max}]$. A *policy* π is a conditional probability distribution over actions, given state, and represents a way of behaving in the MDP. The *value* of a policy π at a given state s is defined as $V^\pi(s) = \mathrm{E}\left[\sum_{i=1}^{\infty} \gamma^{i-1} r^{t+i} \mid s^t = s\right]$, where $\gamma \in [0, 1)$ is the *discount factor*, which trades off the importance of short-term and long-term rewards. For a policy π, the *state-action value* $Q^\pi(s, a) = \mathrm{E}\left[\sum_{i=1}^{\infty} \gamma^{i-1} r^{t+i} \mid s^t = s, a^t = a\right]$ represents the value of taking action a in state s, and following policy π forever after. The goal of the agent will be to find a policy π that maximizes $\mathrm{E}_{s\sim\mu}[V^\pi(s)]$.

Let $P = \{P^a \mid a \in \mathcal{A}\}$ represent the true dynamics, where P^a is the $|\mathcal{S}| \times |\mathcal{S}|$ *transition matrix* representing the transition probabilities under action a. So, $P^a_{s,s'} = \Pr(s' \mid s, a)$. Then let P^π be the one-step transition matrix under policy π: $P^\pi_{s,s'} = \sum_a \pi(a \mid s) P^a_{s,s'}$. The agent does not have access to the true dynamics and will thus will learn an approximate *model* $\hat{P} = \{\hat{P}^a \mid a \in \mathcal{A}\}$, with $\hat{P}^\pi$ defined analogously. Let $\mathcal{M}$ be the *model class*, the space of models the agent's mode learning algorithm could possible produce. In the agnostic setting it is *not assumed* that $P \in \mathcal{M}$.

For simplicity, assume that the reward function $R : \mathcal{S} \times \mathcal{A} \to [R_{min}, R_{max}]$, where $R(s, a)$ is the expected reward obtained when taking action a in state s, is known. Let R^a be a vector of length $|\mathcal{S}|$ with entries $R^a_s = R(s, a)$. Let R^π be the expected reward under policy π: $R^\pi_s = \sum_a \pi(a \mid s) R^a$. By the familiar Bellman equation, $V^\pi = R^\pi + \gamma P^\pi V^\pi$. This implies that $V^\pi = (I - \gamma P^\pi)^{-1} R^\pi$.

From some distribution over states ω, imagine executing policy π with a $(1 - \gamma)$ probability of terminating at every step. Then $d^\pi_\omega = (1 - \gamma)\omega(I - \gamma P^\pi)^{-1}$ is the distribution over states at termination. Let D^π_ω be the corresponding state-action distribution starting from state distribution ω and following policy π: $D^\pi_\omega(s, a) = d^\pi_\omega(s)\pi(a \mid s)$. Note that $\mathrm{E}_{s\sim\mu}[V^\pi(s)] = \frac{1}{1-\gamma}\mathrm{E}_{(s,a)\sim D^\pi_\mu}[R(s, a)]$.

Algorithm 1 DAgger for Model-Based RL

Require: exploration distribution ν, number of iterations N, number of samples per iteration m, ONLINELEARNER, PLANNER

Get initial dataset $\mathcal{D}_1$
Get initial model $\hat{P}_1 \leftarrow$ ONLINELEARNER($\mathcal{D}_1$)
$\pi_1 \leftarrow$ PLANNER($\hat{P}_1$)
for $n \leftarrow 2 \dots N$ **do**
  Let $\mathcal{D}_n$ be an empty dataset
  for $k \leftarrow 1 \dots m$ **do**
    With probability $\frac{1}{2}$:
      Reset using μ
      Use π_{n-1} to obtain $(s, a) \sim D^{\pi_{n-1}}_\mu$
    Else:
      Reset to $(s, a) \sim \nu$
    Take action a to obtain $s' \sim P^a_{s,\cdot}$
    Add $\langle s, a, s' \rangle$ to $\mathcal{D}_n$
  end for
  Update: $\hat{P}_n \leftarrow$ ONLINELEARNER($\hat{P}_{n-1}, \mathcal{D}_n$)
  $\pi_n \leftarrow$ PLANNER($\hat{P}_n$)
end for
return the sequence $\pi_{1:N}$

2.1 The DAgger Algorithm

The DAgger algorithm (Ross and Bagnell 2012) is shown in Algorithm 1. Note that the algorithm requires that the world be able to reset to its initial state distribution μ; this is a reasonable assumption of, for instance, a training simulator. It also assumes the ability to reset to an "exploration distribution" ν over state-action pairs. This is to ensure that the model sees "good transitions" (transitions a good policy would encounter). Otherwise it may never be able to generate good plans. The basic idea of DAgger is to iteratively train the model on state-action pairs drawn from a mix of the exploration distribution ν and the distribution of states encountered by the policy generated at each iteration.

One way to generate ν is to reset to μ and execute an expert policy with $(1 - \gamma)$ termination probability. The "expert policy" could, in principle, be uninformed, for instance the uniform random policy. Though the random policy is guaranteed in the limit to visit all transitions infinitely often, it may be unlikely to visit important, hard-to-reach regions of the state space, and may thus overly concentrate training data on easy to reach states.

For any two policies π and π' and any state distribution ω, let the distribution $\zeta^{\pi,\pi'}_\omega = \frac{1}{2}D^\pi_\omega + \frac{1}{2}D^{\pi'}_\omega$. The analysis of

DAgger relies upon the following result.

Lemma 1. *Suppose the learned model $\hat{P}$ is approximately solved to obtain policy $\hat{\pi}$. Then for any policy π,*

$$\mathrm{E}_{s\sim\mu}[V^{\pi}(s) - V^{\hat{\pi}}] \leq \frac{\gamma(R_{max} - R_{min})}{(1-\gamma)^2}\mathrm{E}_{(s,a)\sim\zeta_{\mu}^{\pi,\hat{\pi}}}[\|\hat{P}^a_{s,\cdot} - \hat{P}^a_{s,\cdot}\|_1] + \epsilon_{oc}^{\pi,\hat{\pi}},$$

where $\epsilon_{oc}^{\pi,\hat{\pi}} = \mathrm{E}_{s\sim\mu}[\hat{V}^{\pi}(s) - \hat{V}^{\hat{\pi}}(s)]$.

The $\epsilon_{oc}^{\pi,\hat{\pi}}$ term can be related to "optimal control error" (hence the oc). For example, if the planner guarantees that $\hat{\pi}$ is ϵ-optimal, then $\epsilon_{oc}^{\pi,\hat{\pi}} \leq \epsilon$.

Now consider the sequence of policies $\pi_{1:N}$ generated by DAgger. Let μ be the initial state distribution. Let $\tilde{\pi} = \mathrm{argmax}_{\pi\in\pi_{1:N}} \mathrm{E}_{s\sim\mu}[V^{\pi}(s)]$ be the best policy in the sequence and let $\bar{\pi}$ be the uniform mixture over all policies in the sequence. For a policy π let $c_{\nu}^{\pi} = \sup_{s,a} \frac{D_{\mu}^{\pi}(s,a)}{\nu(s,a)}$ represent the mismatch between the discounted state-action distribution under π and the exploration distribution ν.

Let $\zeta_i = \frac{1}{2}D_{\mu}^{\pi_{i-1}} + \frac{1}{2}\nu$ be the distribution from which DAgger samples state-action pairs at interation i. For each iteration i define the loss function $L_i^{L1}(\hat{P}) = \mathrm{E}_{(s,a)\sim\zeta_i}[\|P^a_{s,\cdot} - \hat{P}^a_{s,\cdot}\|_1]$. Let $\bar{L}^{L1}_{prd} = \frac{1}{N}\sum_{i=1}^{N} L_i^{L1}(\hat{P}_i)$ be the average $L1$ prediction error of the generated models.

Let the overall training distribution be $\bar{\zeta} = \frac{1}{N}\sum_{i=1}^{N}\zeta_i$ and let $\bar{L}^{L1}_{mdl} = \inf_{P'\in\mathcal{M}} \mathrm{E}_{(s,a)\sim\bar{\zeta}}[\|P^a_{s,\cdot} - P'^a_{s,\cdot}\|_1]$ be the error of the best model in the class under the training distribution $\bar{\zeta}$. Then the average regret of the sequence of models $\hat{P}_{1:N}$ is $\bar{L}^{L1}_{rgt} = \bar{L}^{L1}_{prd} - \bar{L}^{L1}_{mdl}$. If a no-regret algorithm is used to learn the model, $\bar{L}^{L1}_{rgt} \to 0$ as $N \to \infty$.

Finally, for a reference policy π, let $\bar{\epsilon}_{oc}^{\pi} = \sum_{i=1}^{N} \epsilon_{oc}^{\pi,\pi_{i-1}}$ be the average optimal control error induced by the planner. Then Ross and Bagnell show the following.

Theorem 1. *In DAgger, the policies $\pi_{1:N}$ are such that for any policy π,*

$$\mathrm{E}_{s\sim\mu}[V^{\pi}(s) - V^{\bar{\pi}}(s)] \leq \mathrm{E}_{s\sim\mu}[V^{\pi}(s) - V^{\bar{\pi}}(s)] \leq \frac{\gamma(R_{max} - R_{min})}{(1-\gamma)^2} c_{\nu}^{\pi}(\bar{L}^{L1}_{mdl} + \bar{L}^{L1}_{rgt}) + \bar{\epsilon}_{oc}^{\pi}.$$

Informally, this theorem guarantees good performance in the limit if the model learner is no-regret, the exploration distribution resembles the state-action distribution of a good policy, there is a model in the class $\mathcal{M}$ with small prediction error, and the planner gives near optimal plans (for the given models). Ross and Bagnell also provide similar results using measures of prediction error such as log likelihood that are more practical than the L_1 error, which is typically not available during training. They also offer some insight into the behavior of the last policy in the sequence π_N. These results apply to the following analysis as well. The reader is referred to Ross and Bagnell (2012) for details.

The next section develops a bound on value error analogous to Lemma 1 that is specific to one-ply Monte Carlo planning, which will then inspire a modification to the DAgger algorithm. Lemma 1 does apply to this case, as it does to any planning method. However, because one-ply Monte Carlo planning does not guarantee near-optimal plans, the $\epsilon_{oc}^{\pi,\hat{\pi}}$ term could be very large. As a result, even if the expected prediction error becomes very small (or zero), there may still be significant value error. One way this manifests is that the prediction error term does not consider accuracy in states that might be encountered during rollouts, but not execution. Inaccuracies in such "off path" states can damage planning quality but may never be corrected by DAgger.

3 Bounding Error For MC Planning

Imagine a conceptual offline one-ply Monte Carlo planning algorithm that uses model rollouts to create an execution policy $\hat{\pi}$. That is, for each state s and action a, execute n rollouts of length l using the model $\hat{P}$ and the rollout policy π_r. Let $\bar{Q}(s,a)$ be the average discounted return of these rollouts. For sufficiently many rollouts of sufficient length, $\bar{Q}$ will closely approximate $\hat{Q}^{\pi_r}$, the state-action value function of π_r in the model. The execution policy will be greedy with respect to $\bar{Q}$: $\hat{\pi}(s) = \max_a \bar{Q}(s,a)$.

The more typical online one-ply Monte Carlo algorithm re-calculates $\bar{Q}(s,\cdot)$ each time s is encountered, giving a potentially different policy for s each time. For simplicity, we do not consider this effect. This conceptual version of the algorithm is equivalent to the on-line algorithm if the policy for each state is cached, rather than re-computed.

In order to analyze the one-ply Monte Carlo planning algorithm we need to operate on state-action value functions. To facilitate the analysis it will be convenient to image a new "pre-state" s_0 with $R(s_0,a) = 0$ for all actions a. Taking any action from s_0 yields the initial state distribution μ; $\Pr(s \mid s_0, a) = \mu(s)$ for every state s and action a. Let η be a state distribution with $\eta(s_0) = 1$. So for any policy π and any action distribution α,

$$\mathrm{E}_{s\sim\eta,a\sim\alpha}[Q^{\pi}(s,a)] = \gamma\mathrm{E}_{s\sim\mu}[V^{\pi}(s)].$$

Now, for a policy π let B^{π} be the *state-action transition matrix*. That is, for any two state-action pairs (s,a) and (s',a'), let $B^{\pi}_{(s,a),(s',a')} = \Pr(s' \mid s,a)\pi(a' \mid s')$. Then one can define the state-action Bellman operator for a policy π, $T^{\pi}Q(s,a) = R(s,a) + \gamma\sum_{s',a'} B^{\pi}_{(s,a),(s',a')}Q(s',a')$. The maximizing Bellman operator $TQ(s,a) = R(s,a) + \gamma\sum_{s'} P^a_{s,s'}\max_{a'} Q(s',a')$. Bellman operators can be similarly defined for state value functions as well. Then the following bound can be adapted from a result given by Munos (2007). The proof closely follows Munos' original argument, except with state-action value functions instead of state value functions, so it is not presented here.

Lemma 2. *For an arbitrary state-action value function Q, let $\hat{\pi}$ be a greedy policy w.r.t. Q. Then for any policy π,*

$$\mathrm{E}_{s\sim\mu}[V^{\pi}(s) - V^{\hat{\pi}}(s)] \leq \frac{2}{\gamma(1-\gamma)}\mathrm{E}_{(s,a)\sim\zeta_{\eta}^{\pi,\hat{\pi}}}[TQ(s,a) - Q(s,a)].$$

Now for any two policies π and π' define a mixed state-action distribution

$$\xi^{\pi,\pi'} = \frac{1}{2}D_{\mu}^{\pi'} + \frac{1}{4}D_{\mu}^{\pi} + \frac{1}{4}D_{\eta}^{\pi}B^{\pi'}.$$

The third term can be understood as the distribution over state-actions generated by following policy π, but choosing the last action using π'. Note that this term can be re-written entirely in terms of the original initial state distribution μ:

$$D_\eta^\pi B^{\pi'}(s,a) = (1-\gamma)\mu(s)\pi'(a \mid s) + \gamma D_\mu^\pi B^{\pi'}(s,a).$$

The following lemma is analogous to Lemma 1 but specific to one-ply Monte Carlo planning.

Lemma 3. *Let $\bar{Q}$ be the state-action value function obtained by one-ply Monte Carlo planning using $\hat{P}$. Let $\hat{\pi}$ be greedy with respect to $\bar{Q}$. Then for any policy π,*

$$\begin{aligned}
&\mathrm{E}_{s\sim\mu}[V^\pi(s) - V^{\hat{\pi}}(s)] \\
&\quad \le \frac{4}{1-\gamma}\mathrm{E}_{(s,a)\sim\xi^{\pi,\hat{\pi}}}[|\hat{Q}^{\pi_r}(s,a) - Q^{\pi_r}(s,a)|] \\
&\qquad + \frac{4}{1-\gamma}\|\bar{Q} - \hat{Q}^{\pi_r}\|_\infty + \frac{2}{1-\gamma}\|TV^{\pi_r} - V^{\pi_r}\|_\infty,
\end{aligned}$$

Before presenting the proof of this lemma, first consider the meaning of the three terms in the bound. The second term is the error incurred by the planning algorithm (due to estimating $\hat{V}^{\pi_r}$ with a finite number of finite horizon samples). With sufficiently many rollouts of sufficient length, this term can be arbitrarily small with high probability.

The third term is a property of the rollout policy and the true dynamics. It represents the irreducible error caused by the choice of a sub-optimal rollout policy.

The first term in the bound is the error in the value of the rollout policy induced by the approximate model. This can be bounded in terms of the prediction accuracy of the model. For a policy π and state-action distribution ϕ, let δ_ϕ^π be analogous to D_ω^π except with an initial state-action distribution instead of state distribution. Then the following is an adaptation of a result from Ross and Bagnell (2012) to state-action value functions. The proof is similar to Ross and Bagnell's argument, and is not presented here.

Lemma 4. *For any state-action distribution ϕ,*

$$\begin{aligned}
&\mathrm{E}_{(s,a)\sim\phi}[Q^\pi(s,a) - \hat{Q}^\pi(s,a)] \\
&\quad \le \frac{\gamma(R_{max} - R_{min})}{2(1-\gamma)^2}\mathrm{E}_{(s,a)\sim\delta_\phi^{\hat{\pi}}}[\|P_{s,\cdot}^a - \hat{P}_{s,\cdot}^a\|_1].
\end{aligned}$$

Combining this with Lemma 3 yields the main result of this section.

Lemma 5. *Let $\bar{Q}$ be the value function obtained by one-ply Monte Carlo planning using $\hat{P}$. Let $\hat{\pi}$ be greedy with respect to $\bar{Q}$. Then for any policy π,*

$$\begin{aligned}
&\mathrm{E}_{s\sim\mu}[V^\pi(s) - V^{\hat{\pi}}(s)] \\
&\quad \le \frac{2\gamma(R_{max} - R_{min})}{(1-\gamma)^3}\mathrm{E}_{(s,a)\sim\delta_{\xi^{\pi,\hat{\pi}}}^{\pi_r}}[\|P_{s,\cdot}^a - \hat{P}_{s,\cdot}^a\|_1] \\
&\qquad + \frac{4}{1-\gamma}\|\hat{Q} - \hat{Q}^{\pi_r}\|_\infty + \frac{2}{1-\gamma}\|TV^{\pi_r} - V^{\pi_r}\|_\infty.
\end{aligned}$$

3.1 Proof of Lemma 3

This section presents the proof of the key result above, Lemma 3. It may be useful to consult Table 1, which summarizes many of the symbols used in this argument.

π_r	The rollout policy used by one-ply MC.
Q^{π_r}	The true Q-function of π_r.
$\hat{Q}^{\pi_r}$	The Q-function of π_r in the model $\hat{P}$.
$\bar{Q}$	The Q-function obtained using one-ply MC.
$\hat{\pi}$	The execution policy (greedy w.r.t. $\bar{Q}$).
μ	The initial state distribution of the MDP.
η	The state distribution with all its mass on s_0.
B^π	The state-action transition matrix for policy π.
d_ω^π	The discounted state distribution generated by policy π starting in state distribution ω.
D_ω^π	The discounted state-action distribution generated by policy π starting in state distribution ω.
$\zeta_\omega^{\pi,\pi'}$	$\frac{1}{2}D_\omega^\pi + \frac{1}{2}D_\omega^{\pi'}$.
$\xi^{\pi,\pi'}$	$\frac{1}{2}D_\mu^{\pi'} + \frac{1}{4}D_\mu^\pi + \frac{1}{4}D_\eta^\pi B^{\pi'}$.

Table 1: Symbol Reference

Proof of Lemma 3. First note that

$$\begin{aligned}
&T\bar{Q} - \bar{Q} \\
&\quad = T^{\hat{\pi}}\bar{Q} - T^{\hat{\pi}}Q^{\pi_r} + Q^{\pi_r} - \bar{Q} + T^{\hat{\pi}}Q^{\pi_r} - Q^{\pi_r} \\
&\quad = \gamma B^{\hat{\pi}}(\bar{Q} - Q^{\pi_r}) + Q^{\pi_r} - \bar{Q} + T^{\hat{\pi}}Q^{\pi_r} - Q^{\pi_r} \\
&\quad \le \gamma B^{\hat{\pi}}|\bar{Q} - Q^{\pi_r}| + |Q^{\pi_r} - \bar{Q}| + |TQ^{\pi_r} - Q^{\pi_r}|.
\end{aligned}$$

Thus, combining with Lemma 2,

$$\begin{aligned}
&\mathrm{E}_{s\sim\mu}[V^\pi(s) - V^{\hat{\pi}}(s)] \\
&\quad \le \frac{2}{\gamma(1-\gamma)}\Big(\gamma\mathrm{E}_{(s,a)\sim\zeta_\eta^{\pi,\hat{\pi}}B^{\hat{\pi}}}[|\bar{Q}(s,a) - Q^{\pi_r}(s,a)|] \\
&\qquad + \mathrm{E}_{(s,a)\sim\zeta_\eta^{\pi,\hat{\pi}}}[|Q^{\pi_r}(s,a) - \bar{Q}(s,a)|] \\
&\qquad + \mathrm{E}_{(s,a)\sim\zeta_\eta^{\pi,\hat{\pi}}}[|TQ^{\pi_r}(s,a) - Q^{\pi_r}(s,a)|]\Big).
\end{aligned}$$

Consider the first term in this expression:

$$\begin{aligned}
&\gamma\mathrm{E}_{(s,a)\sim\zeta_\eta^{\pi,\hat{\pi}}B^{\hat{\pi}}}[|\bar{Q}(s,a) - Q^{\pi_r}(s,a)|] \\
&\qquad = \frac{\gamma}{2}\mathrm{E}_{(s,a)\sim D_\eta^\pi B^{\hat{\pi}}}[|\bar{Q}(s,a) - Q^{\pi_r}(s,a)|] \\
&\qquad\quad + \frac{\gamma}{2}\mathrm{E}_{(s,a)\sim D_\mu^{\hat{\pi}}}[|\bar{Q}(s,a) - Q^{\pi_r}(s,a)|].
\end{aligned}$$

Combine this with the second term to get

$$\begin{aligned}
&\frac{\gamma}{2}\mathrm{E}_{(s,a)\sim D_\eta^\pi B^{\hat{\pi}}}[|\bar{Q}(s,a) - Q^{\pi_r}(s,a)|] \\
&\quad + \frac{\gamma}{2}\mathrm{E}_{(s,a)\sim D_\mu^{\hat{\pi}}}[|\bar{Q}(s,a) - Q^{\pi_r}(s,a)|] \\
&\quad + \frac{1}{2}\mathrm{E}_{(s,a)\sim D_\eta^{\hat{\pi}}}[|Q^{\pi_r}(s,a) - \bar{Q}(s,a)|] \\
&\quad + \frac{1}{2}\mathrm{E}_{(s,a)\sim D_\eta^\pi}[|Q^{\pi_r}(s,a) - \bar{Q}(s,a)|].
\end{aligned}$$

Noting that $R(s_0,a) = 0$ for all a, this is equivalent to

$$\begin{aligned}
&\frac{\gamma}{2}\mathrm{E}_{(s,a)\sim D_\eta^\pi B^{\hat{\pi}}}[|\bar{Q}(s,a) - Q^{\pi_r}(s,a)|] \\
&\quad + \gamma\mathrm{E}_{(s,a)\sim D_\mu^{\hat{\pi}}}[|\bar{Q}(s,a) - Q^{\pi_r}(s,a)|] \\
&\quad + \frac{\gamma}{2}\mathrm{E}_{(s,a)\sim D_\mu^\pi}[|Q^{\pi_r}(s,a) - \bar{Q}(s,a)|] \\
&= 2\gamma\mathrm{E}_{(s,a)\sim\xi^{\pi,\hat{\pi}}}[|\bar{Q}(s,a) - Q^{\pi_r}(s,a)|].
\end{aligned}$$

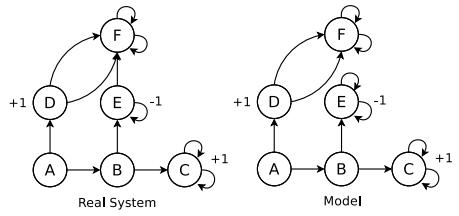

Figure 1: An example where the loss function suggested by Lemma 5 provides more guidance than Lemma 1.

Now note that $|\bar{Q} - Q^{\pi_r}| \leq |\bar{Q} - \hat{Q}^{\pi_r}| + |\hat{Q}^{\pi_r} - Q^{\pi_r}|$. So,

$$\begin{aligned}
&\mathrm{E}_{s\sim\mu}[V^{\pi}(s) - V^{\hat{\pi}}(s)] \\
&\quad\leq \frac{4}{1-\gamma}\mathrm{E}_{(s,a)\sim\xi^{\pi,\hat{\pi}}}[|\hat{Q}^{\pi_r}(s,a) - Q^{\pi_r}(s,a)|] \\
&\qquad + \frac{4}{1-\gamma}\mathrm{E}_{(s,a)\sim\xi^{\pi,\hat{\pi}}}[|\bar{Q}(s,a) - \hat{Q}^{\pi_r}(s,a)|] \\
&\qquad + \frac{2}{\gamma(1-\gamma)}\mathrm{E}_{(s,a)\sim\zeta_{\eta}^{\pi,\hat{\pi}}}[|TQ^{\pi_r}(s,a) - Q^{\pi_r}(s,a)|].
\end{aligned}$$

Using Jensen's inequality, one can see that

$$\begin{aligned}
&\mathrm{E}_{(s,a)\sim\zeta_{\eta}^{\pi,\hat{\pi}}}[|TQ^{\pi_r}(s,a) - Q^{\pi_r}(s,a)|] \\
&\quad= \gamma\mathrm{E}_{(s,a)\sim\zeta_{\eta}^{\pi,\hat{\pi}}}\left[\left|\mathrm{E}_{s'\sim P^a_{s,\cdot}}[TV^{\pi_r}(s') - V^{\pi_r}(s')]\right|\right] \\
&\quad\leq \gamma\mathrm{E}_{s\sim(\frac{1}{2}d_{\mu}^{\pi}+\frac{1}{2}d_{\mu}^{\hat{\pi}})}[|TV^{\pi_r}(s) - V^{\pi_r}(s)|] \\
&\quad\leq \gamma\|TV^{\pi_r} - V^{\pi_r}\|_{\infty},
\end{aligned}$$

The result is obtained by finally noting that

$$\mathrm{E}_{(s,a)\sim\xi^{\pi,\hat{\pi}}}[|\bar{Q}(s,a) - \hat{Q}^{\pi_r}(s,a)|] \leq \|\bar{Q} - \hat{Q}^{\pi_r}\|_{\infty}.$$

□

Before discussing the algorithmic implications of this result, the next section briefly discusses the relationship between Lemma 1 and Lemma 5.

3.2 Relationship Between the Bounds

Though the bound given in Lemma 1 is not strictly tighter or strictly looser than the one in Lemma 5, we can obtain some understanding of their relationship by comparing their first terms (the terms subject to optimization). For any policies π and π', let $c_{\pi_r}^{\pi',\pi} = \max_{\pi''\in\{\pi,\pi'\}} \sup_{(s,a)} \frac{\pi''(a|s)}{\pi_r(a)}$ bound the mismatch between the rollout policy and the two policies π and π'. Let $\mathcal{L}(\hat{P}) = \|P^a_{s,\cdot} - \hat{P}^a_{s,\cdot}\|_1$. Then it is straightforward to verify the following:

Proposition 6. *For any policies π and π'*

$$\mathrm{E}_{(s,a)\sim\zeta_{\mu}^{\pi,\pi'}}[\mathcal{L}(\hat{P})] \leq \frac{2c_{\pi_r}^{\pi',\pi}}{1-\gamma}\mathrm{E}_{(s,a)\sim\delta_{\xi^{\pi,\pi'}}^{\pi_r}}[\mathcal{L}(\hat{P})].$$

From this result we can see that if, by improving model accuracy, the first term of Lemma 5 approaches zero, then so does the first term of Lemma 1. But the converse is not necessarily true. At first glance this might seem to make Lemma 5 less informative than Lemma 1. However, in some such cases this is because Lemma 5 apportions error to its first term (subject to optimization) that Lemma 1 assigns to its second term (not subject to optimization).

Algorithm 2 DAgger-MC for one-ply Monte Carlo Planning

Require: exploration distribution ν, number of iterations N, number of samples per iteration m, rollout policy π_r, ONLINELEARNER, MC-PLANNER

Get initial dataset $\mathcal{D}_1$
Get initial model $\hat{P}_1 \leftarrow$ ONLINELEARNER($\mathcal{D}_1$)
$\pi_1 \leftarrow$ MC-PLANNER($\hat{P}_1$)
for $n \leftarrow 2 \dots N$ **do**
  for $k \leftarrow 1 \dots m$ **do**
    With probability $\frac{1}{2}$:
      Reset using μ.
      Use π_{n-1} to obtain $(z,b) \sim D_{\mu}^{\pi_{n-1}}$.
    Else with probability $\frac{1}{4}$:
      Reset to $(z,b) \sim \nu$.
    Else with probability $\frac{1}{4}(1-\gamma)$:
      Reset using μ to obtain z.
      Obtain $b \sim \pi_{n-1}(\cdot \mid z)$.
    Else:
      Reset to $(x,c) \sim \nu$.
      Take action c in x to obtain $z \sim P^c_{x,\cdot}$.
      Obtain $b \sim \pi_{n-1}(\cdot \mid z)$.
    Take action b in z to obtain $y \sim P^b_{z,\cdot}$.
    From y use π_r to obtain $(s,a) \sim \delta^{\pi_r}_{\xi^{\pi,\pi_{n-1}}}$.
    Take action a to obtain $s' \leftarrow P^a_{s,\cdot}$.
    Add $\langle s,a,s'\rangle$ to $\mathcal{D}$.
  end for
  Update: $\hat{P}_n \leftarrow$ ONLINELEARNER($\hat{P}_{n-1}, \mathcal{D}_n$)
  $\pi_n \leftarrow$ MC-PLANNER($\hat{P}_n$)
end for
return the sequence $\pi_{1:N}$

To see this, consider the system and model pictured in Figure 1. There are 6 states and two actions (Up and $Right$ indicated by the side of the states from which the arrows originate). The system starts in state A. Clearly the optimal policy π^* always chooses $Right$. Assume the MC planner uses uniform random π_r and, for simplicity, let $\hat{\pi}$ be greedy with respect to $\hat{Q}^{\pi_r}$. The rollout policy is equally likely to go Up or $Right$ from state B, so, in the flawed model, it severely undervalues state B. As such $\hat{\pi}$ will choose Up from state A. Since neither the expert policy π^* nor the initial execution policy $\hat{\pi}$ ever reach state E the model would not be corrected. Subsequent iterations would yield the same results, and the suboptimal behavior would persist forever.

In contrast, because the rollout policy does reach state E, the first term in Lemma 5 would be positive. Assuming the model class permits it, an algorithm minimizing this error would correct the model. With an accurate model the planner chooses $Right$ in state A for sufficiently large γ.

4 DAgger for Monte Carlo Planning

Motivated by Lemma 5, DAgger-MC is a variation on DAgger, specifically for use with the one-ply Monte Carlo algo-

rithm. As can be seen in Algorithm 2, the main differences appear on Lines 6-19 which generate the training examples.

The analysis of DAgger-MC closely follows the analysis of the original DAgger algorithm, with only a few changes. Let $\xi_i(s,a) = \frac{1}{2}D_\mu^{\pi_{i-1}}(s,a) + \frac{1}{4}\nu(s,a) + \frac{1}{4}((1-\gamma)\mu(s)\pi_{i-1}(a \mid s) + \gamma\nu B^{\pi_{i-1}}(s,a))$ be the distribution from which DAgger-MC samples state-action pairs to initiate rollouts in iteration i, named (z,b) in the pseudocode. Let $\delta_i = \delta_{\xi_i}^{\pi_r}$ be the distribution from which DAgger-MC samples state-action pairs for updating in iteration i. Let the overall training distribution be $\bar{\delta} = \frac{1}{N}\sum_{i=1}^{N}\delta_i$ and let $\bar{\Gamma}_{mdl}^{L1} = \inf_{P' \in \mathcal{M}} \mathrm{E}_{(s,a)\sim\bar{\delta}}[\|P_{s,\cdot}^a - P'^a_{s,\cdot}\|_1]$. For each iteration i define the loss function $\Gamma_i^{L1}(\hat{P}) = \mathrm{E}_{(s,a)\sim\delta_i}[\|P_{s,\cdot}^a - \hat{P}_{s,\cdot}^a\|_1]$ and let $\bar{\Gamma}_{prd}^{L1} = \frac{1}{N}\sum_{i=1}^{N}\Gamma_i^{L1}(\hat{P}_i)$. Then the average regret of the sequence of models $\hat{P}_{1:N}$ is $\bar{\Gamma}_{rgt}^{L1} = \bar{\Gamma}_{prd}^{L1} - \bar{\Gamma}_{mdl}^{L1}$. Finally, let $\bar{\epsilon}_{mc} = \frac{1}{N}\frac{4}{1-\gamma}\sum_{i=1}^{N}\|\hat{Q}_i - \hat{Q}_i^{\pi_r}\|_\infty + \frac{2}{1-\gamma}\|TV^{\pi_r} - V^{\pi_r}\|_\infty$.

The following is the main theorem, presented without proof because the argument closely follows that of Ross and Bagnell, except with Lemma 5 in place of Lemma 1. Let $\tilde{\pi}$ be the best policy in the sequence of policies, and let $\bar{\pi}$ be the uniform mixture over all of the policies.

Theorem 2. *In DAgger-MC, the policies $\pi_{1:N}$ are such that for any policy π,*

$$\mathrm{E}_{s\sim\mu}[V^\pi(s) - V^{\tilde{\pi}}(s)] \leq \mathrm{E}_{s\sim\mu}[V^\pi(s) - V^{\bar{\pi}}(s)] \leq \bar{\epsilon}_{mc} + \frac{4\gamma(R_{max} - R_{min})}{(1-\gamma)^3} c_\nu^\pi(\bar{\Gamma}_{mdl}^{L1} + \bar{\Gamma}_{rgt}^{L1}).$$

As discussed above, though this bound is often looser than the one given by Theorem 1, it acknowledges the importance of model accuracy in states that are reached by rollouts. As a result, DAgger-MC can perform much better than DAgger with a Monte Carlo planner. The next section provides a brief illustration of how these issues can arise in practice.

5 Experiments

Consider the simple game, Shooter, which is shown in Figure 3. The agent controls a ship at the bottom of the screen. The available actions are $noop$, $left$, $right$, and $shoot$. If the agent chooses to $shoot$ it receives a reward of -1 and, if there is not already a bullet directly above the ship, a bullet appears, traveling upwards. If a bullet hits one of the three targets, it explodes. Each target has a "sweet spot" in the middle. Hitting the sweet spot yields 20 reward. Hitting to either side yields 10 reward. The (known) reward function uses the explosion shape to determine how much reward is received. The optimal policy is simply to move $right$ across the screen, $shoot$ing a bullet at the middle of each target. Note that though the domain is simple, the state space is very large (due to the possible configurations of bullets on the screen) making near-optimal planning unlikely. On-line Monte Carlo planning is a popular approach in such a case.

The results of applying DAgger and DAgger-MC to this problem (using various expert policies to generate ν) are shown in Figure 2. In all cases the planning algorithm was one-ply Monte Carlo with 50 rollouts of length 15 of the uniform random policy. The model learner was a factored model where each factor was learned using Context Tree Switching (Veness et al. 2012), similar to the FAC-CTW algorithm (Veness et al. 2011). The model for each pixel used a 7×7 neighborhood around the pixel as input. Data was shared across all positions to encode some degree of positional invariance. The discount factor γ was set to 0.9. Each iteration generated a training batch of 500 samples. The discounted return obtained by the policy generated at each iteration in an episode of length 30 is reported, averaged over 200 trials. The shaded regions represent 95% confidence intervals for the mean performace. The benchmark lines marked "Perfect Model", and "Random" represent the average performance of one-ply Monte Carlo using a perfect model, and the uniform random policy, respectively.

In Figure 2a, the optimal policy was used as the expert. Note that in the DAgger analysis, using the optimal policy as the expert is the ideal scenario. However, in this case DAgger rarely even fires a bullet – it gets 0 reward in almost every iteration. Because the optimal policy only ever chooses $right$ or $shoot$ the initial model is never trained on examples of a bullet moving while the ship moves $left$. Rollouts are very likely to encounter such situations, and inspection of model rollouts revealed that the model's inaccuracy in those cases causes the planner to undervalue the $shoot$ action. As such, the policy obtained from planning never $shoot$s at all which prevents errors regarding bullet dynamics from being corrected. In contrast, DAgger-MC quickly learns to accurately predict bullet movement and performs well.

In Figure 2b, the uniform random policy was used as the expert. As might be expected, DAgger performs far better using this expert as it provides better coverage of states that will be encountered during rollouts. That said, using an uninformed expert has downsides as well. For instance, Theorem 1 will be loose due to an unacceptably large c_ν^π term. Also, the random policy is not very likely to hit the targets' sweet spots, so DAgger-MC takes substantially more data to learn about that aspect of the dynamics with this expert than with the optimal policy. Finally, because DAgger relies entirely on the exploration policy to provide coverage of rollout states it has a persistent disadvantage compared to DAgger-MC which systematically samples those states.

In Figure 2c, the expert was the one-ply Monte Carlo algorithm applied to a perfect model. This expert policy makes good decisions like the optimal policy, but is noisy, so DAgger does relatively well. But, again, because DAgger relies solely on the noise in the exploration policy to learn about "off-path" states, it persistently sees fewer such examples than DAgger-MC. As the model learns, it overgeneralizes in some "off-path" cases and is not corrected because such cases are rare in the training set. For instance, it learns that $shoot$ing causes a bullet to appear, but not that bullets can not appear directly next to each other. As a result, when two bullets appear next to each other in a rollout, this forms an unfamiliar context and the model subsequently behaves erratically in the rollout. These effects actually cause DAgger's performance to degrade slightly as it sees more data.

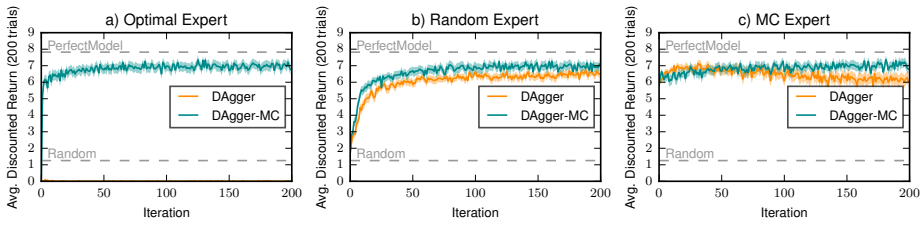

Figure 2: Results in the Shooter domain. See text for details.

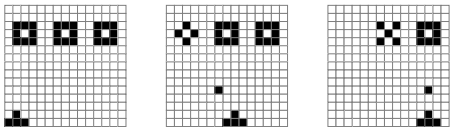

Figure 3: The Shooter game. Left: Initial screen. Middle: First target has just been hit in its sweet spot. Right: Second target has just been hit outside its sweet spot.

6 Discussion and Future Work

Though this paper has focused on the simple one-ply Monte Carlo algorithm, as the Introduction pointed out, the issue of model accuracy during rollouts is important for more sophisticated Monte Carlo tree search (MCTS) algorithms as well. Preliminary work suggests that a result in the spirit of Lemma 5 can likely be obtained for MCTS. However, unlike one-ply Monte Carlo, each MCTS rollout depends on the rollouts before it. Thus, in order to sample states from the rollout policy distribution, one must be able to simulate the entire MCTS process in the world/simulator during training. This is a fairly strong assumption, even when a simulator is available. Exploring these issues and, in particular, determining if similar guarantees can be obtained with weaker assumptions remain interesting directions for future research.

While Theorem 2 does provide guarantees in the agnostic setting, the bound does still rely on the existance of some model in the class with small one-step prediction error. As Talvitie (2014) recently pointed out, this may not be the case in many problems of interest, especially when using factored models. Further, when the model class is limited, rollouts may generate states that can never be reached in the real system. In that case no amount of training on examples from the real system can help. Talvitie introduced the heuristic Hallucinated Replay method, which incorporates erroneous states generated by sample rollouts into the training set, and found that it can mitigate the effects of compounding model errors during rollouts. Perhaps this insight can be combined with DAgger-MC to obtain tighter bounds and expand its applicability to a broader range of model classes.

7 Conclusions

The DAgger algorithm for model-based reinforcement learning in the agnostic setting was shown to fail when coupled with the one-ply Monte Carlo planning algorithm because it does not train the model on states that are reached during rollouts, but not execution. The DAgger-MC algorithm, a variation on DAgger for use with Monte Carlo planning, was derived based on a novel error bound, and was shown to perform well in some cases where DAgger fails.

References

Joseph, J.; Geramifard, A.; Roberts, J. W.; How, J. P.; and Roy, N. 2013. Reinforcement learning with misspecified model classes. In *2013 IEEE International Conference on Robotics and Automation (ICRA)*, 939–946.

Kocsis, L., and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *Proceedings of the 17th European Conference on Machine Learning (ECML)*, 282–293.

Ljung, L., ed. 1999. *System Identification: Theory for the User (2nd Ed.)*. Upper Saddle River, NJ: Prentice Hall.

Munos, R. 2007. Performance bounds in l_p-norm for approximate value iteration. *SIAM journal on control and optimization* 46(2):541–561.

Parr, R.; Li, L.; Taylor, G.; Painter-Wakefield, C.; and Littman, M. L. 2008. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, 752–759.

Ross, S., and Bagnell, D. 2012. Agnostic system identification for model-based reinforcement learning. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*, 1703–1710.

Schoknecht, R. 2002. Optimality of reinforcement learning algorithms with linear function approximation. In *Advances in Neural Information Processing Systems (NIPS)*, 1555–1562.

Sorg, J.; Lewis, R. L.; and Singh, S. 2010. Reward design via online gradient ascent. In *Advances in Neural Information Processing Systems (NIPS)*, 2190–2198.

Sutton, R. S.; Szepesvri, C.; Geramifard, A.; and Bowling, M. 2008. Dyna-style planning with linear function approximation and prioritized sweeping. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence (UAI)*, 528–536.

Talvitie, E. 2014. Model regularization for stable sample rollouts. In *Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence (UAI)*, 780–789.

Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Trans. on*.

Tesauro, G., and Galperin, G. R. 1996. On-line policy improvement using monte-carlo search. In *Advances in Neural Information Processing Systems (NIPS)*, 1068–1074.

Veness, J.; Ng, K. S.; Hutter, M.; Uther, W. T. B.; and Silver, D. 2011. A Monte-Carlo AIXI Approximation. *Journal of Artificial Intelligence Research* 40:95–142.

Veness, J.; Ng, K. S.; Hutter, M.; and Bowling, M. 2012. Context tree switching. In *Proceedings of the 2012 Data Compression Conference (DCC)*, 327–336. IEEE.