Efficient Reductions for Imitation Learning

Stéphane Ross
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA, USA

J. Andrew Bagnell
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA, USA

Abstract

Imitation Learning, while applied successfully on many large real-world problems, is typically addressed as a standard supervised learning problem, where it is assumed the training and testing data are i.i.d.. This is not true in imitation learning as the learned policy influences the future test inputs (states) upon which it will be tested. We show that this leads to compounding errors and a regret bound that grows quadratically in the time horizon of the task. We propose two alternative algorithms for imitation learning where training occurs over several episodes of interaction. These two approaches share in common that the learner's policy is slowly modified from executing the expert's policy to the learned policy. We show that this leads to stronger performance guarantees and demonstrate the improved performance on two challenging problems: training a learner to play 1) a 3D racing game (Super Tux Kart) and 2) Mario Bros.; given input images from the games and corresponding actions taken by a human expert and near-optimal planner respectively.

1 INTRODUCTION

Imitation Learning, where decision-making behavior is programmed by demonstration, has led to state-of-the-art performance in a variety of applications, including, e.g., outdoor mobile robot navigation (Silver 2008), legged locomotion (Ratliff 2006), advanced manipulation (Schaal 1999), and electronics games. A common approach to imitation learning is to train a classifier or regressor to replicate an expert's policy given training data of the encountered observations and actions performed by the expert. Given access to a planner, current state-of-the-art techniques based on Inverse Optimal Control (IOC) (Abbeel 2004, Ratliff 2006) achieves this indirectly by learning the cost function the expert is optimizing from the observed behavior, and the planner is used by the learner to minimize the long-term costs. These techniques can also be thought as training a classifier (the planner), which is parametrized by the cost function. This often has the advantage that learning the cost function generalizes better over the state space or across similar tasks. A broad spectrum of learning techniques have been applied to imitation learning (Argall 2009; Chernova 2009), however these applications all violate the crucial assumption made by statistical learning approaches that a learner's prediction does not influence the distribution of examples upon which it will be tested.

Ignoring the effects of the learner on the underlying state distribution leads to serious practical difficulties (Thrun 1995, Pomerleau 1989). Consider the widely-known imitation learning success of ALVINN (Pomerleau 1989), a neural network designed to drive at speed in on-road environments by mimicking a human driver. Pomerleau one of the first to elucidate this problem, notes that, "when driving for itself, the network may occasionally stray from the center of road and so must be prepared to recover by steering the vehicle back to the center of the road." Unfortunately, demonstration of such "recovery behavior" is rare for good human drivers and thus is poorly represented in training data. The result is that naive application of machine learning techniques leads to a compounding of errors and unacceptable driving performance. Formally, as shown in Section 2.1, we find that a supervised learner that makes mistakes with some small probability ϵ achieves a total cost that grows *quadratically* $O(\epsilon T^2)$ in the task horizon T rather than as $O(\epsilon T)$ as we would expect for a typical supervised learning task.

Ideally, we would optimize the total T-step cost of the learned policy under the state distribution it induces. This is a "chicken-or-the-egg" problem, however, as without knowing the policy in advance it is not possible to generate samples from the induced distribution of states. When optimizing the learned policy then, we do not know how the

Appearing in Proceedings of the 13^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2010, Chia Laguna Resort, Sardinia, Italy. Volume 9 of JMLR: W&CP 9.

state distribution will be affected, so that the learners are nearly always too optimistic about their own performance. This problem is commonly encountered by policy iteration approaches to reinforcement learning. One approach that has proven useful is to change the current policy slowly when doing the improvement step, so that the new policy's state distribution is close to the old policy (Kakade 2002). In this work, we demonstrate that such an approach leads to better performance in imitation learning. Intuitively, the idea is to start from a policy that always queries and executes the expert's (that the learner is attempting to mimic) action, and slowly replace it with the learned policy. This allows the learner to retrain under the new state distribution as the policy changes. Such approaches require a slightly more interactive setting than traditional imitation learning, where the learner is allowed to interact with the system and can query the expert at any given state. We will assume such a setting throughout the paper. Such interactivity is possible in many real-world imitation learning problem, e.g. any teleoperated robot can fall under this setting.

After introducing notation, we begin by analyzing the traditional supervised approach to imitation learning. Our analyses throughout are reduction-based (Beygelzimer 2005) where we relate the performance of the harder imitation learning problem to a series of derived, simpler classification or optimization tasks. We present a simple, sequential algorithm that trains a non-stationary policy by iterating over time steps and demonstrate that this method achieves better performance bounds. While providing motivation and intuition for more practical algorithms, this approach is naturally limited to finite task horizons. We then draw upon connections to the Conservative Policy Iteration (CPI) (Kakade 2002) (reinforcement learning) and SEARN (Daume 2009) (structured prediction) approaches that extend to arbitrary/infinite horizons as they generate a stationary policy. While these methods are impractical for our imitation learning domain, we present the *Stochastic Mixing Iterative Learning* (SMILe) algorithm, a simple, iterative approach which provides the benefits of SEARN with substantially simpler implementation and less demanding interaction with an expert. We demonstrate experimentally the improved performance of SMILe on two challenging tasks: 1) learning how to steer a car in a 3D racing game and 2) learning how to play Mario Bros.; given only input images and corresponding actions taken by a human expert and near-optimal planner respectively.

2 PRELIMINARIES

We begin by introducing notation familiar from the sequential decision-making literature (Putterman 1994). Let π^* be the expert's policy we wish to mimic, assumed to be deterministic and π_s be the distribution over actions of a policy π in state s. We let T refer to the task horizon. Define $C(s,a)$ in $[0,1]$ to be the immediate cost of doing action a in state s for the task we are interested in. Correspondingly, $C_\pi(s) = \mathbb{E}_{a\sim\pi_s}(C(s,a))$ is the expected immediate cost of performing policy π in state s. For imitation learning we are most concerned with $e(s,a) = I(a \neq \pi^*(s))$, the 0-1 loss of executing action a in state s, compared to the expert's policy π^*. We allow $e_\pi(s) = \mathbb{E}_{a\sim\pi_s}(e(s,a))$ to denote the expected 0-1 loss of policy π in state s. As the relative weighting of state distributions are crucial we denote by d^i_π the state distribution at time step i if we followed policy π from the initial time step. The distribution $d_\pi = \frac{1}{T}\sum_{i=1}^{T} d^i_\pi$ encodes the state visitation frequency over T time steps if we followed policy π. We denote by $J(\pi) = T\mathbb{E}_{s\sim d_\pi}(C_\pi(s))$ the expected T-step cost (under C) of executing policy π. Finally, we will be interested in bounding the regret (in T step cost) of a policy π with respect to the best policy in a particular policy class Π, which is defined as $\mathcal{R}_\Pi(\pi) = J(\pi) - \min_{\pi'\in\Pi} J(\pi')$. For most of our discussion, we will assume that $\pi^* \in \Pi$ and that π^* is a good policy (under C), i.e. its regret $\mathcal{R}_\Pi(\pi^*)$ is $O(1)$, which is negligible for large T. However all theorems will be stated such as to not rely on these assumptions.

2.1 THE SUPERVISED LEARNING APPROACH TO IMITATION

The traditional approach to imitation learning trains a classifier that learns to replicate the expert's policy under the state distribution induced by the expert. Formally, the traditional approach minimizes 0-1 loss under distribution d_{π^*}: $\hat{\pi} = \operatorname{argmin}_{\pi\in\Pi} \mathbb{E}_{s\sim d_{\pi^*}}(e_\pi(s))$. Now assume that the resulting classifier (policy) $\hat{\pi}$ makes a mistake with probability ϵ under d_{π^*}, i.e. $\mathbb{E}_{s\sim d_{\pi^*}}(e_{\hat{\pi}}(s)) = \epsilon$. Then we have the following guarantee:

Theorem 2.1. *Let $\hat{\pi}$ be such that $\mathbb{E}_{s\sim d_{\pi^*}}[e_{\hat{\pi}}(s)] \leq \epsilon$. Then $J(\hat{\pi}) \leq J(\pi^*) + T^2\epsilon$. (Proof in Supplementary Material)*

Intuitively, the reason this grows quadratically in T is because as soon as $\hat{\pi}$ makes a mistake, it could end up in new states that were not visited by π^*, and always incur maximal cost of 1 at each step from then on. Most importantly, this bound is tight; it is easy to construct a dynamic system, cost function C and policy $\hat{\pi}$ with ϵ 0-1 loss on d_{π^*} that incurs expected T-step cost of $(1-\epsilon T)J(\pi^*) + T^2\epsilon$ (e.g. example in Supplementary Material or example given by Kääriäinen (2006)). It is natural to ask if any approach could guarantee bounds closer to that of the classical supervised learning setting, i.e. regret linear in the number of examples to classify. The following section shows that, indeed, we can achieve near-linear regret on large subclasses of problems.

3 FORWARD TRAINING ALGORITHM

The traditional approach fails to give good performance bounds due to the discrepancy between the testing and

training distribution when $\hat{\pi} \neq \pi^*$ and because the learner does not learn how to recover from mistakes it makes. The intuition behind the next approach, called forward training algorithm, is that both of these problems can be solved if we allow the training to occur over several iterations, where at each iteration we train one policy for one particular time step. If we do this training sequentially, starting from the first time step to the last, then at the i^{th} iteration, we can sample states from the actual testing distribution at time step i by executing the learned policies for each previous step, and then ask the expert what to do at the i^{th} time step to train the next policy. Furthermore, if the learner makes mistakes, the expert demonstrates how to recover at future steps. The algorithm terminates once it has learned a policy for all T steps. This is similar to the Sequential Stacking algorithm (Cohen 2005) for sequence classification.

Formally, let π_i^n denote the policy executed at time step i after the n^{th} iteration of the algorithm and π^n the non-stationary policy defined by π_i^n for $i = 1, \ldots, T$. Initially, we set $\pi_1^0, \pi_2^0, \ldots, \pi_T^0$ to query the expert and do the same action. At iteration i, the algorithm trains the policy π_i^i on state distribution $d^i_{\pi^{i-1}}$, and all other policies remain unchanged (i.e. $\pi_j^i = \pi_j^{i-1} \forall j \neq i$). After T iterations, π^T does not query the expert anymore and we are done.

We can bound the expected total cost of the final policy π^T as follows. Let $J^\pi(\pi', t)$ denote the expected T-step cost of executing π' at time t and policy π at all other time steps, and denote the policy disadvantage of π_i^i with respect to π^{i-1} as $\mathbb{A}(\pi^{i-1}, \pi_i^i) = J^{\pi^{i-1}}(\pi_i^i, i) - J(\pi^{i-1})$. The policy disadvantage represents how much increase in T-step cost we incur by changing the current policy at a single step.

Theorem 3.1. *$J(\pi^n) = J(\pi^*) + n\bar{\mathbb{A}}$, where $\bar{\mathbb{A}} = \frac{1}{n}\sum_{i=1}^n \mathbb{A}(\pi^{i-1}, \pi_i^i)$. (Proof in Supplementary Material)*

This bound suggests that we should choose π_i^i to minimize $\mathbb{A}(\pi^{i-1}, \pi_i^i)$, or equivalently $J^{\pi^{i-1}}(\pi_i^i, i)$, i.e. minimize the cost-to-go from step i, assuming we follow π^* afterwards. Minimizing cost-to-go is impractical in imitation learning, however. To train a classifier, this requires the ability to try several actions from the same state and see what cost-to-go is incurred after for each of these actions. This can only be achieved if we have the ability to restart the system in any particular state, which is usually not the case in practice. Learning a value-estimate (i.e. regression) also typically requires far more samples and is less robust if we are only interested in determining the best action. Besides sample complexity, the interaction required with an expert in this case is quite unnatural as well– we have to try an action, use the expert in the loop and then try alternate actions "reseting" to the original state for multiple runs.

Finally, in imitation learning the true cost function C we would ideally like to minimize for mimicry is often unclear; instead we typically use agreement with the expert to bound the loss with respect to an arbitrary cost. That

Initialize $\pi_1^0, \ldots, \pi_T^0$ to query and execute π^*.
for $i = 1$ **to** T **do**
  Sample T-step trajectories by following π^{i-1}.
  Get dataset $\mathcal{D} = \{(s_i, \pi^*(s_i))\}$ of states, actions taken by expert at step i.
  Train classifier $\pi_i^i = \text{argmin}_{\pi \in \Pi} \mathbb{E}_{s \sim \mathcal{D}}(e_\pi(s))$.
  $\pi_j^i = \pi_j^{i-1}$ for all $j \neq i$
end for
Return $\pi_1^T, \ldots, \pi_T^T$

Algorithm 3.1: Forward Training Algorithm.

is, if π_i^i mimics exactly $\pi_i^{i-1} = \pi^*$, then the policy disadvantage $\mathbb{A}(\pi^{i-1}, \pi_i^i) = 0$. Hence, small classification error mimicking the policy we are replacing at each iteration should guarantee good performance under all cost functions. Thus, our forward algorithm (detailed in Algorithm 3.1) chooses π_i^i to minimize the 0-1 loss with respect to π^* under $d^i_{\pi^{i-1}}$. From the reduction point of view, any standard classification algorithm, IOC technique or other imitation learning method can be used to train this classifier. Now let $J^\pi(\pi', t, s)$ represent the expected T-step cost of π conditioned on being at state s at time t and executing π' instead of π at time t and suppose that $\sup_{\pi \in \Pi, s | d^i_{\pi^{i-1}}(s) > 0} [J^{\pi^{i-1}}(\pi, i, s) - J^{\pi^{i-1}}(\pi^*, i, s)] \leq u_i$. Intuitively u_i represents the maximal increase in expected cost-to-go from any probable state at time i, when changing only the policy at time i of π^{i-1}. Then if π_i^i is such that $\mathbb{E}_{s \sim d^i_{\pi^{i-1}}}(e_{\pi_i^i}(s)) = \epsilon_i$, the policy disadvantage is bounded by $\mathbb{A}(\pi^{i-1}, \pi_i^i) \leq \epsilon_i u_i$. This implies that $J(\pi^T) \leq J(\pi^*) + T\overline{\epsilon u}$, where $\overline{\epsilon u} = \frac{1}{T}\sum_{i=1}^T \epsilon_i u_i$.

In the worst case, $\overline{\epsilon u} \leq T\bar{\epsilon}$, where $\bar{\epsilon} = \frac{1}{T}\sum_{i=1}^T \epsilon_i$, as each of the u_i could be $T - i + 1$. This is the same guarantees as the traditional approach. However, it is often the case that changing only one action in the current policy will not increase the cost by much more than some small constant k *on average*. Especially, if π^* has some stability properties, i.e. it can rapidly bring itself back into its typical state distribution upon random disturbances then, we can expect the cost to increase only for a few steps in the worst case. In such cases, the forward algorithm has a regret of $O(T\bar{\epsilon})$.

In particular, if the cost function is truly to minimize imitation loss ($e(s, a)$), then $u_i = 1$ because π^* achieves cost-to-go of 0, no matter the current state. This implies $J(\pi^T) \leq J(\pi^*) + T\bar{\epsilon}$. In general, if we have a system where $\sup_{t \leq T, \pi_i^i \in \Pi \forall i \leq t, s | d_{\pi^{t-1}}(s) > 0} [J^{\pi^{t-1}}(\pi_t^t, t, s) - J^{\pi^{t-1}}(\pi^*, t, s)] \leq k$ (i.e. there exist no $(t-1)$-step trajectory from which the expected cost-to-go of the expert can be made much worse by changing only the action at time t), then we will have that $u_i \leq k$ for all i, so that $J(\pi^T) \leq J(\pi^*) + kT\bar{\epsilon}$. For instance, navigation problems where the expert can reach a goal state within k steps (in expectation) starting in any state would fall in that cate-

gory. In those favorable cases, the traditional approach can be shown to still have quadratic regret in T.

The drawback of this approach is that it is impractical when T is large and does not extend to infinite horizon tasks as we are training a non-stationary policy. The next algorithm we present trains a stationary policy, enabling extension to infinite horizon settings while also guaranteeing similar performance guarantees to the forward algorithm.

4 STOCHASTIC MIXING ITERATIVE LEARNING ALGORITHM

The forward algorithm can guarantee smaller regret because it is changing the policy slowly. This can also be achieved by training a stationary stochastic policy over several iteration, where at iteration n, the current learner's policy π^n is stochastically mixed with a new policy $\hat{\pi}^{n+1}$ to construct the next policy π^{n+1}, i.e. $\pi^{n+1} = (1-\alpha)\pi^n + \alpha\hat{\pi}^{n+1}$. Here α is a small probability of executing the new policy $\hat{\pi}^{n+1}$ and with large probability $1-\alpha$ we still do the old policy π^n. If α is small enough, then we can expect the new policy to only be executed at most once over T steps with high probability, which ensures good performance as in the forward algorithm. Initially, the learner starts with the policy $\pi^0 = \pi^*$, which always queries and executes the expert's action. After n iterations, the probability of querying the expert at any step is $(1-\alpha)^n$, which goes to 0 as $n \to \infty$. We can terminate after any iteration N, by removing the probability of querying the expert and re-normalizing to return the unsupervised policy $\tilde{\pi}^N = \frac{1}{1-(1-\alpha)^N}[\pi^N - (1-\alpha)^N\pi^0]$ that never queries the expert. Our analysis below indicates how to train the policies $\hat{\pi}^n$, and choose the parameter α and number of iteration N so as to ensure good performance of $\tilde{\pi}^N$.

We begin with a stochastic mixing algorithm based on CPI/SEARN. However, these are impractical if applied directly to imitation learning as they require optimization of the cost-to-go at each iteration for reasons discussed in Section 3. Furthermore, the known performance guarantees provided by Daume (2009) are no better, and in fact worse by a logarithmic factor, than the performance of the traditional approach[1]. Here, we provide an alternative analysis of stochastic mixing methods defined in terms of the policy disadvantage, related to the policy advantage in Kakade (2002). Further, we bound the policy disadvantage by minimizing the immediate 0-1 classification loss to obtain a practical approach for our imitation learning setting. Additionally our analysis also provides a tighter bound on the performance of previous algorithms and further insight as to why they outperform naive supervised learning for structured prediction (Daume 2009).

Let's denote $J_k^\pi(\pi', t_1, \ldots, t_k)$ the expected T-step cost of executing π' at steps $\{t_1, \ldots, t_k\}$, $\bar{J}_k^\pi(\pi') = \frac{1}{\binom{T}{k}} \sum_{t_1=1}^{T-k+1} \cdots \sum_{t_k=t_{k-1}+1}^{T} J_k^\pi(\pi', t_1, \ldots, t_k)$ the expected T-step cost of executing π' k times and policy π at all other steps, and by $\mathbb{A}_k(\pi, \pi') = \bar{J}_k^\pi(\pi') - J(\pi)$ the k^{th}-order policy disadvantage of π' with respect to π. Then:

Lemma 4.1. *If $\alpha \leq \frac{1}{T}$, then for any $k \in \{1, 2, \ldots, T-1\}$, $J(\pi^n) \leq J(\pi^0) + n\sum_{i=1}^{k} \alpha^i \binom{T}{i}(1-\alpha)^{T-i}\bar{\mathbb{A}}_i + n\alpha^{k+1}T\binom{T}{k+1}$, where $\bar{\mathbb{A}}_i = \frac{1}{n}\sum_{j=1}^{n} \mathbb{A}_i(\pi^{j-1}, \hat{\pi}^j)$. (Proof in Supplementary Material)*

At each time step π^n chooses the expert (π^0) with probability $p_n = (1-\alpha)^n$. We would like to have strong performance guarantees for the performance of the unsupervised policy $\tilde{\pi}^n$, which never queries the expert. The performance of $\tilde{\pi}^n$ can be bounded as follows:

Lemma 4.2. *$J(\tilde{\pi}^n) \leq J(\pi^n) + p_n T^2$. (Proof in Supplementary Material)*

This lemma implies that if $n \geq \frac{2}{\alpha}\ln T$, then $p_n \leq \frac{1}{T^2}$, such that $J(\tilde{\pi}^n) \leq J(\pi^n) + 1$, where the additional cost of 1 becomes negligible for large T.

The SEARN algorithm effectively seeks to minimize directly the previous bound for $k = 1$ by choosing $\hat{\pi}^n$ to minimize $\mathbb{A}_1(\pi^{n-1}, \hat{\pi}^n)$ [2], using $N = \frac{2}{\alpha}\ln T$ and $\alpha = T^{-3}$. With those parameters, SEARN guarantees $J(\tilde{\pi}^N) \leq J(\pi^*) + O(T\ln T\bar{\mathbb{A}}_1 + \ln T)$. To minimize the cost-to-go, SEARN solves a cost-sensitive classification problem where for each state sample in the dataset, the cost-to-go under the current policy must be estimated for each action. Such estimates place the expert back in an identical state for each action and require T-step roll-outs. Such a reset is impractical in the physical world. Even when possible, it is difficult for a person to perform a task (e.g. driving) with multiple reset. Additionally, T-step roll-outs require tedious interaction: for A actions, k trajectories per estimate, and m sampled states per iteration, it can be shown that SEARN needs $O(mAkT^4\log T)$ queries to the expert to complete all $O(T^3\log T)$ iterations. Also when the cost function we would like to minimize is unknown, then again it is not possible to obtain the cost-to-go estimates. For these reasons, minimizing cost-to-go as SEARN is impractical in our setting.

Instead, we choose $\hat{\pi}^n$ to mimic π^{n-1}, as if $\hat{\pi}^n = \pi^{n-1}$ exactly, then as for all k, we get $\mathbb{A}_k(\pi^{n-1}, \hat{\pi}^n) = 0$. Since only unknown component of π^{n-1} is the expert's policy π^*, we can focus on only learning π^* (under the state distribution induced by π^{n-1}). That is, if $\hat{\pi}^{*n} = \text{argmin}_{\pi\in\Pi}\mathbb{E}_{s\sim d_{\pi^{n-1}}}(e_\pi(s))$, then the policy

[1] While they appear to scale linearly with T, we note that in SEARN, for loss functions that are sums of losses on each prediction, the definition of ϵ must scale with the number of predictions as the cost-to-go will so scale. Thus it is effectively $O(T^2\log T)$ in the horizon length.

[2] SEARN minimizes $\bar{J}_1^{\pi^{n-1}}(\hat{\pi}^n)$, which is equivalent to minimizing $\mathbb{A}_1(\pi^{n-1}, \hat{\pi}^n)$.

Initialize $\pi^0 \leftarrow \pi^*$ to query and execute expert.
for $i = 1$ **to** N **do**
  Execute π^{i-1} to get $\mathcal{D} = \{(s, \pi^*(s))\}$.
  Train classifier $\hat{\pi}^{*i} = \text{argmin}_{\pi \in \Pi} \mathbb{E}_{s \sim \mathcal{D}}(e_\pi(s))$.
  $\pi^i = (1-\alpha)^i \pi^* + \alpha \sum_{j=1}^{i} (1-\alpha)^{j-1} \hat{\pi}^{*j}$.
end for
Remove expert queries: $\tilde{\pi}^N = \frac{\pi^N - (1-\alpha)^N \pi^*}{1-(1-\alpha)^N}$
Return $\tilde{\pi}^N$

Algorithm 4.1: The SMILe Algorithm.

$\hat{\pi}^n = p_{n-1}\hat{\pi}^{*n} + (1-p_{n-1})\tilde{\pi}^{n-1}$ approximates π^{n-1} using the new estimate $\hat{\pi}^{*n}$ of π^*, and the previously trained policies $\tilde{\pi}^{n-1}$. Using the same update rule as before ($\pi^n = (1-\alpha)\pi^{n-1} + \alpha\hat{\pi}^n$), we obtain that $\pi^n = p_n\pi^* + \alpha\sum_{i=1}^{n} p_{i-1}\hat{\pi}^{*i}$. Hence we will use directly this update rule instead as we don't have to construct the $\hat{\pi}^n$. Finally, to ensure we get training data $(s, \pi^*(s))$ for every state we encounter while following π^n, we always query the expert's action, but π^n only executes it with probability $(1-\alpha)^n$. This now completes the description of our algorithm, called SMILe, for Stochastic Mixing Iterative Learning, which is detailed in Algorithm 4.1. Again, from the reduction perspective, SMILe can use any classification algorithm, IOC technique or other imitation learning technique to train the classifier at each iteration. Note that for SMILe the weights of each learned policy remain constant over the iterations and decays exponentially as n gets larger. This contrasts with SEARN where old policies have much smaller weights than the newest policies. This makes intuitive sense as we might expect that for SMILe the old policies are better at mimicing π^* on the states encountered most often when π^n does a good job at mimicing π^*.

Now let $\epsilon_i = \mathbb{E}_{s \sim d_{\pi^{i-1}}}(e(s, \hat{\pi}^{*i}))$, $\tilde{\epsilon} = \frac{\alpha}{(1-(1-\alpha)^n)} \sum_{i=1}^{n} (1-\alpha)^{i-1}\epsilon_i$ and $\tilde{\mathbb{A}}_1 = \frac{\alpha}{(1-(1-\alpha)^n)} \sum_{i=1}^{n} (1-\alpha)^{i-1}\mathbb{A}_1(\pi^{i-1}, \hat{\pi}^{*i}|\pi^*)$, for $\mathbb{A}_1(\pi^{i-1}, \hat{\pi}^{*i}|\pi^*) = \bar{J}_1^{\pi^{i-1}}(\hat{\pi}^{*i}) - \bar{J}_1^{\pi^{i-1}}(\pi^*)$ then for SMILe we have the following guarantee:

Theorem 4.1. *For $\alpha = \frac{\sqrt{3}}{T^2\sqrt{\log T}}$, and $N = 2T^2(\ln T)^{3/2}$, then $J(\tilde{\pi}^N) \leq J(\pi^*) + O(T(\tilde{\mathbb{A}}_1 + \tilde{\epsilon}) + 1)$. (Proof in Supplementary Material)*

Since in most problems, $\tilde{\mathbb{A}}_1$ will be at least in the order of $\tilde{\epsilon}$, then this is effectively $O(T\tilde{\mathbb{A}}_1 + 1)$. Hence SMILe needs $O(T^2(\ln T)^{3/2})$ iterations to guarantee regret of $O(T\tilde{\mathbb{A}}_1 + 1)$. This is an improvement over SEARN which requires $O(T^3 \ln T)$ iterations.

Just as with the forward algorithm, in the general case, $O(T\tilde{\mathbb{A}}_1)$ may be as large as $O(T^2\tilde{\epsilon})$ as an early error may simply be unrecoverable. Again, in parallel with the forward algorithm, in many instances– particularly those relevant to imitation learning– we can show that this policy disadvantage is upper bounded because of favorable

$\mathcal{D}_n \leftarrow \emptyset$
while $|\mathcal{D}_n| \leq m$ **do**
  Sample $r \sim \text{Uniform}(0, 1)$
  if $r \leq (1-\alpha)^T$ **then**
    Pick trajectory (T samples) from $\mathcal{D}_{n-1}$ (without replacement) and add it to $\mathcal{D}_n$
  else
    Sample t uniformly in $\{1, 2, \ldots, T\}$
    Add new trajectory (T samples) to $\mathcal{D}_n$ by executing π^n at every step $i \neq t$, and $\hat{\pi}^n$ at step t.
  end if
end while
Return $\mathcal{D}_n$

Algorithm 4.2: First algorithm to construct dataset for state distribution d_{π^n} at iteration $n+1$ efficiently ($n \geq 1$).

mixing properties of the dynamic system or recovery behavior of the policy to be mimicked. In particular, it is straightforward to construct classes where $\tilde{\mathbb{A}}_1$ is bounded by $O(\log T\bar{\epsilon})$, while standard supervised training is still only able to achieve quadratic regret (see example in Supplementary Material). Further, we find in practice that the policy disadvantage is typically bounded on average due to the "recoverability" of the problem, leading to significantly better performance of the SMILe approach.

Sample Complexity. The sample complexity of the SMILe algorithm might seem prohibitive as it requires $O(T^2(\log T)^{3/2})$ iterations. If it takes m examples to train a classifier then, we might fear needing more than $O(T^2 m)$ samples to guarantee our error bound. Fortunately, this is not the case as one can reuse a significant part of the samples collected at previous iterations as shown below.

Intuitively, the requirement to slowly change the policy implies that some of the samples that we draw for a given algorithm iteration are exactly that which the previous policies would have generated– no actions have actually changed on these. Concretely, as $\pi^n = (1-\alpha)\pi^{n-1} + \alpha\hat{\pi}^n$, with probability $(1-\alpha)^T$, π^n always executes π^{n-1} over T steps. Thus a first simple approach, detailed in Algorithm 4.2, is to reuse a trajectory from the previous dataset for π^{n-1} with probability $(1-\alpha)^T$. With this approach, we expect to collect less than $\alpha T m$ new samples at each iteration. Summing over $N = \frac{2}{\alpha}\ln T$ iterations, we need $O(T \ln T m)$ samples (in expectation) to complete all iterations, rather than $O(T^2 \ln T m)$. This first approach ignores that $\hat{\pi}^n$ is similar to π^{n-1}, so that even more samples can be reused. This is shown in the following lemma. Let $D(\pi) = d_\pi$, $D_k^\pi(\pi')$ the T-step state frequencies if π' is executed k times and π at all other timesteps, and $D_{k,j}^\pi(\pi', \pi'')$ the T-step state frequencies if π' is executed k times, π'' j times and π at all other timesteps. Then $D(\pi^n)$ can be related to the distributions $D_k^{\pi^*}(\tilde{\pi}^{n-1})$ (which are sampled during execution of π^{n-1}):

$\mathcal{D}_n^k \leftarrow \emptyset$ for $k \in \{0, 1, \ldots, T\}$
while $|\bigcup_{k=0}^{T} \mathcal{D}_n^k| \leq m$ **do**
  Sample $k \sim \text{Binomial}(T, 1-(1-\alpha)^n)$
  Sample $r \sim \text{Uniform}(0, 1)$
  if $k = 0$ or $r \leq (\frac{1-(1-\alpha)^{n-1}}{1-(1-\alpha)^n})^k$ **then**
    Pick trajectory (T samples) from $\mathcal{D}_{n-1}^k$ (without replacement) and add it to $\mathcal{D}_n^k$
  else
    Sample t_1 uniformly in $\{1, \ldots, T\}$
    for $i = 2$ to k **do**
      Sample t_i uniformly $\{1, \ldots, T\} \backslash \{t_1, \ldots, t_{i-1}\}$
    end for
    Add new trajectory (T samples) to $\mathcal{D}_n^k$ by executing π^* at every step $t \notin \{t_1, \ldots, t_k\}$, $\hat{\pi}^{*n}$ at step t_1, and $\tilde{\pi}^n$ at all steps t_i for $i \geq 2$.
  end if
end while
Return $\{\mathcal{D}_n^0, \mathcal{D}_n^1, \ldots, \mathcal{D}_n^T\}$

Algorithm 4.3: Second algorithm to construct dataset for state distribution d_{π^n} at iteration $n+1$ efficiently ($n \geq 1$).

Lemma 4.3. $D(\pi^n) = \sum_{k=0}^{T} c_{n,k}[\beta_n^k D_k^{\pi^*}(\tilde{\pi}^{n-1}) + (1-\beta_n^k) D_{1,k-1}^{\pi^*}(\hat{\pi}^{*n}, \tilde{\pi}^n)]$ *for* $\beta_n = \frac{1-p_{n-1}}{1-p_n}$ *and* $c_{n,k} = (1-p_n)^k p_n^{T-k}\binom{T}{k}$. *(Proof in Supplementary Material)*

This equality suggests a sampling procedure that leads to a sample complexity of $O(Tm)$ for SMILe, which is detailed in Algorithm 4.3. The drawback is that the sampling procedure is more involved and requires additional bookkeeping, as we need to maintain seperate datasets $\mathcal{D}_n^k$ for $k \in \{0, 1, \ldots, T\}$ at each iteration ($\mathcal{D}_n^k$ contains trajectories where π^* executed $T-k$ times; their union is the dataset for training π^{*n+1}). With this procedure, the expected number of new samples that we need to collect at each iteration after the first one is less then $\alpha(1-\alpha)^{n-1}Tm$. Thus the total expected number of samples we need to complete all iteration is less than $(T+1)m$. Hence, the sample complexity of SMILe is in the same order as the forward algorithm, which only required T iterations. This is also a major improvement over the sample complexity of SEARN, which requires a factor $O(T^3 \log T)$ more queries to the expert.

Practical Choice of Learning Rate. In practice, we will see that it is often unnecessary to choose such a small α and iterate until we reach $p_n = \frac{1}{T^2}$ to achieve good performance. Further, as SMILe does not need to learn a policy for each time step, it is more practical than the forward algorithm when T is large, as we may terminate after some small number of iterations $N < T$.

5 EXPERIMENTAL RESULTS

To verify the efficacy of the SMILe approach, we applied it on two challenging imitation learning problems: 1) the Super Tux Kart open source 3D racing game (Figure 1) and 2) the Mario Bros. game (Figure 3).

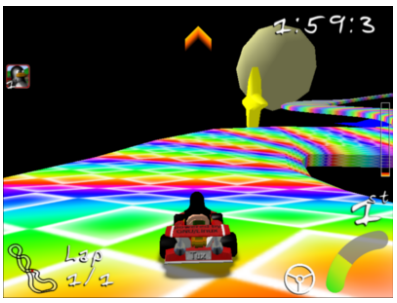

Figure 1: Image from Super Tux Kart's Star Track.

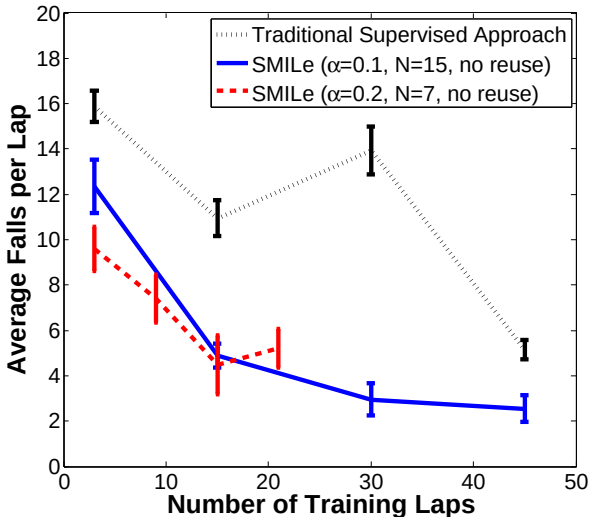

Figure 2: Performance of SMILe and the traditional approach in Super Tux Kart as a function of training laps.

SUPER TUX KART. In Super Tux Kart, our goal is to train the computer to steer the car properly at fixed speed on a particular race track given screen captured images from the game and corresponding joystick commands from a human player. Here, since planning from the image is non-trivial, the base classifier used was a Neural Network trying to map directly images to actions, rather than using more sophisticated IOC techniques. [3]

We compare the performance of SMILe (without sample reuse) to the traditional approach on a track called "Star

[3]Implementation details: To make the problem tractable, screen images are resized to 24×18 pixels, and we transform the RGB color of these pixels to LAB color, yielding 1296 features. The joystick commands are integers ranging from -32768 to 32767 (negative is left, positive is right, magnitude controls degree of turn). We choose to discretize this range into 15 values (15 class). To learn the controller, we use single layer neural networks with 1296 input nodes, 32 hidden layer nodes and 15 output nodes and train it with backward propagation (Mitchell 1997), where each neuron corresponds to a sigmoid function and use ten-fold cross-validation to avoid overfitting.

Track". As this track floats in space, a car can fall off the track at any point. For this task, we measure performance by the average number of times the vehicle falls off the track per lap. When using SMILe, each iteration consists of three laps of data with screen capture and joystick commands recorded at 15 Hz ($\sim$1000 datapoints/lap). As T is not a fixed horizon here, we made a practical choice for α and N, comparing $\alpha = 0.1$ and $\alpha = 0.2$, and choosing to stop once the probability of choosing the expert reaches roughly $\frac{1}{5}$, yielding $N = 15$ and $N = 7$ respectively. When comparing the result of our approach after n iterations, we give equivalent training data to the classification approach, i.e. data collected over $3n$ laps at 15 Hz, where the human is always playing, using the same parameters for the neural network. A video that enables more qualitative comparison is available on YouTube (Ross 2009a). Figure 2 shows the performance (with 95% confidence intervals) of the controller trained with SMILe and the traditional approach, as a function of the total number of training laps. We observe that the traditional approach requires 45 laps of data before begining to yield reasonable performance. SMILe with $\alpha = 0.1$ achieves similar performance after only 5 iterations (15 laps) and gets much better performance after 15 iterations (45 laps). However, SMILe with $\alpha = 0.2$ doesn't achieve as good performance at the end of its 7 iterations. This suggests that mixing more slowly is desirable and leads to better performance. Hence, we recommend choosing the smallest α one can afford, considering the higher number of iterations required.

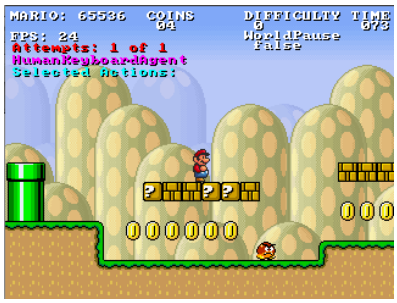

Figure 3: Captured image from Mario Bros.

MARIO BROS. For Mario Bros., we used the open source simulator from the recent Mario Bros. AI competition (Togelius 2009). In this game, Mario must move across a level without falling into gaps and being hit by enemies. Levels are randomly generated and vary in difficulty (more difficult gaps and types of enemies). Our goal is to train the computer to play this game by providing features extracted from screen captured images and corresponding actions taken by a near-optimal planner having full access to the game's internal state. Here the actions are represented by 4 binary variables, corresponding to whether the Left, Right, Jump and Speed buttons are pressed. The player can choose any combination of these buttons at any step. Again here, since planning from the image features is non-

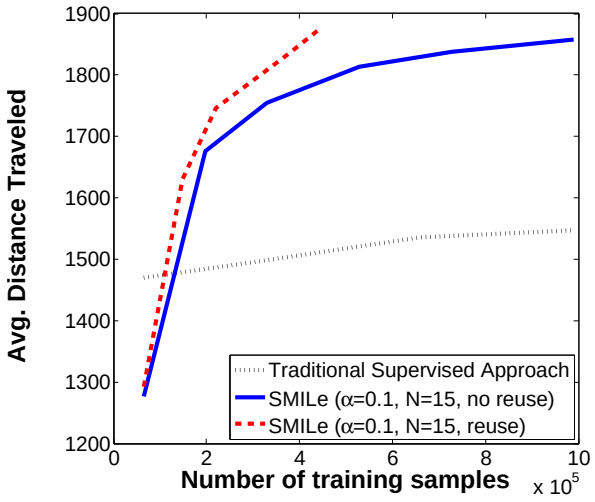

Figure 4: Performance of SMILe and the traditional approach in Mario Bros. as a function of training data.

trivial, the base classifier used was a Neural Network trying to map directly image features to actions, rather than using more sophisticated IOC techniques. [4]

We compare the performance of SMILe (with and without sample reuse) to the traditional approach on randomly generated levels ranging in difficulty from 0 to 10 (where difficulty is chosen uniformly randomly). Difficulty 0 is easy even for beginners, however difficulty 10 is challenging even for very experienced human players. We measure performance by the average distance per level traveled by Mario before falling into a gap, being killed by an enemy or running out of time. As the average total distance of a level is around 4300 units, this can vary in roughly [0, 4300].

[4]Implementation details: Each image is divided into 22x22 cells of equal size, and 14 binary features is extracted for each cell: whether it contains ground, pass-through ground, pipe, cannon, enemy, winged enemy, spiky enemy, flower enemy, bullet, shell, fireball, destructible block, item block and mushroom/flower. The game runs at 24 frames/second but new actions are only issued at every 4 frames (keeping actions constant over 4 frames). This is done to make the planner run in near real-time. As a single image does not allow the learner to detect the direction enemies are moving, we give as input the history of features for the last 4 images where decisions were made. In addition, the history of the last chosen actions and 4 other observable binary variables (mario on ground, mario can jump, fire mario, large mario) over the last 6 steps where new actions were chosen are also added to the input features. This yields 27152 input binary features, which are very sparse. We trained single hidden layer neural networks with 32 hidden nodes, using an available library optimized for sparse features (Karampatziakis 2008). The planner used as expert has access to the internal state and can simulate exactly the future using the game's simulator. It plans 24 frames ahead, considering sequences of 3 actions (each action held constant over 8 frames), and chooses the one that leads to the best state in 1 second according to an heuristic function. It replans at every 4 frames. This achieves beyond human-level performance, as it can go through every level at any difficulty over 95 to 99% of the times.

When using SMILe, each iteration consists 6000 training examples (state,action pair) for each difficulty in $\{0, 1, \ldots, 10\}$, and we use $\alpha = 0.1$ and $N = 15$. When reusing samples, we choose T as $\frac{1}{\alpha} = 10$ and use the approach in Algorithm 4.3. The traditional approach is also given equal training data for each difficulty and we look at its performance after different total amount of data. Figure 4 compares each approach[5]. Again, SMILe performs better even without reusing samples. When samples are reused, performance is not affected as it compares to the performance without reuse for the same number of iterations. However as a function of samples collected, the approach is even more efficient and gives better results. A video that enables more qualitative comparison is available on YouTube (Ross 2009b). What we observe is that the traditional approach is only good on the easiest difficulty levels, falling into gaps, hitting enemies or getting stuck often at the harder difficulties, whereas SMILe also performs well at easy difficulties and performs better at the harder difficulties. SMILe doesn't get stuck like the traditional approach but still falls quite often into gaps.

6 CONCLUSION

We have presented an analysis of two different approaches to imitation learning that have better performance guarantees than the traditional approach. We showed that SMILe works better in practice than the traditional approach on two challenging tasks. Our theoretical analysis provides further novel insights into why other stochastic mixing algorithms like SEARN works well in practice. As we never made any assumption that the states were markovian, our analysis also directly extends to partially observable systems where the "state" used for predicting the expert's action might be only the last observation (if the expert's policy is reactive) or a function of the history of action and observation.[6]. While we have experimented using SMILe with standard classification algorithms, it would be interesting to use more sophisticated IOC techniques as the base classifier with SMILe when possible. We believe such combination should offer the best performance. SMILe can also be adapted for general structured prediction and would provide similar guarantees to SEARN, but with the advantage of only having to solve simpler classification problems, without requiring rollouts of full cost-to-go.

[5]The 95% confidence intervals are fairly tight, roughly ± 20, so they are not shown in the plot.

[6]The only assumptions we need for our analysis to hold is that the "state" must be a sufficient statistic of the history for predicting the next expert's action and the immediate costs, in the sense that they are conditionnally independent of any information in the history given the current "state".

Acknowledgements

This work is supported by the ONR MURI grant N00014-09-1-1052, Reasoning in Reduced Information Spaces, the National Science Foundation through the Quality of Life Technology Center ERC and by the National Sciences and Engineering Research Council of Canada (NSERC).

References

P. Abbeel and A. Y. Ng (2004). Apprenticeship learning via inverse reinforcement learning. In *ICML*.

B. D. Argall, S. Chernova, M. Veloso and B. Browning (2009). A Survey of Robot Learning from Demonstration. In *Robotics and Autonomous Systems*.

A. Beygelzimer, V. Dani, T. Hayes, J. Langford and B. Zadrozny (2005). Error limiting reductions between classification tasks. In *ICML*.

S. Chernova and M. Veloso (2009). Interactive Policy Learning through Confidence-Based Autonomy. In *JAIR*.

W. W. Cohen and V. R. Carvalho (2005). Stacked sequential learning. In *IJCAI*.

H. Daume, J. Langford and D. Marcu (2009). Search-based structured prediction. *Machine Learning Journal*.

S. Kakade and J. Langford (2002). Approximately Optimal Approximate Reinforcement Learning. In *ICML*.

N. Karampatziakis (2008). Sparse NN library. `http://www.cs.cornell.edu/~nk/sparsenn/`.

M. Kääriäinen (2006). Lower bounds for reductions. Atomic Learning workshop.

T. Mitchell (1997). Machine Learning. McGraw-Hill.

D. Pomerleau (1989). ALVINN: An Autonomous Land Vehicle in a Neural Network. In *NIPS*.

M. Puterman (1994). Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley.

N. Ratliff, D. Bradley, J. A. Bagnell and J. Chestnutt (2006). Boosting structured prediction for imitation learning. In *NIPS*.

S. Ross (2009). Comparison of Supervised Learning and SMILe for Imitation Learning in Super Tux Kart. `http://www.youtube.com/watch?v=ywH9Z2NivjY`.

S. Ross (2009). Comparison of Supervised Learning and SMILe for Imitation Learning in Mario Bros. `http://www.youtube.com/watch?v=ldl7TQJxE5U`.

S. Schaal (1999). Is imitation learning the route to humanoid robots? In *Trends in Cognitive Sciences*.

D. Silver, J. A. Bagnell and A. Stentz (2008). High Performance Outdoor Navigation from Overhead Data using Imitation Learning. In *Proceedings of Robotics Science and Systems* (RSS).

J. Togelius and S. Karakovskiy (2009). Mario AI Competition. `http://julian.togelius.com/mariocompetition2009`.

S. Thrun (1995). Learning To Play the Game of Chess. In *NIPS*.