Work-in-progress: Corrected Self Imitation Learning via Demonstrations

Yunshu Du*
Washington State University
yunshu.du@wsu.edu

Garrett Warnell
Army Research Laboratory
garrett.a.warnell.civ@mail.mil

Assefaw Gebremedhin
Washington State University
assefaw.gebremedhin@wsu.edu

Peter Stone
The University of Texas at Austin
pstone@cs.utexas.edu

Matthew E. Taylor
Washington State University
matthew.e.taylor@wsu.edu

ABSTRACT

While reinforcement learning (RL) agents have the remarkable ability to learn by interacting with their environments, this process is often slow and data inefficient. Because environment interaction is typically expensive, many approaches have been studied to speed up RL. One popular method for doing so is to leverage human knowledge via imitation learning (IL), in which a demonstrator provides an example of the desired behavior, and the agent seeks to imitate. In this in-progress work, we propose a new way of integrating IL and deep RL, which we call *corrected self imitation learning*, where an agent provided with demonstration can learn faster compared to an agent with no demonstration. Our method does not increase the number of environmental interactions compared to a baseline RL method, and works well even in the case when the demonstrator is not an expert. We evaluate our method in the Atari game of Ms. Pac-Man and achieve promising results indicating our method has the potential to speed up deep RL algorithms.

KEYWORDS

Deep Reinforcement Learning; Leaning from Humans; Behavior Cloning

1 INTRODUCTION

Deep Reinforcement Learning (deep RL) has become an increasingly popular general machine learning technique due to its ability to achieve record-setting performance in multiple domains [10, 12, 14, 15, 22, 23]. However, one of the main drawbacks of deep RL is its sample complexity as it learns through trial-and-error, i.e., it commonly requires millions of steps taken in the environment before the agent can begin to act reasonably. While acceptable in a simulator, this exploration becomes unaffordable in a real-world environment (e.g., robotics [13]) since taking so many steps is often expensive in terms of risk, energy, or time.

Because of this issue, many approaches have been proposed to improve the data efficiency of deep RL algorithms. One class of such methods involves leveraging available *human* knowledge to bootstrap learning [2]; behavior cloning (BC) is a simple and efficient example of such techniques [3, 18–21]. Generally, BC assumes there exists an expert (usually a human) who can demonstrate the desired behavior, which is recorded and stored in the form of state-action pairs. An agent then attempts to learn the expert's policy using supervised learning over those pairs—a process that requires computation but no additional environment interaction. While BC models are fast to train and avoid costly environment interaction, they are known to perform poorly on out-of-sample data [19]. In contrast, even given their poor sample complexity, RL algorithms *can* overcome this drawback by interacting with the environment.

In this paper, we propose a new method for integrating BC and RL that introduces one way to get the best of both: an RL algorithm is used to learn the task, but a BC model is incorporated during training to speed up learning by providing valuable experiential data *where it can*. Different from some existing BC approaches where a static set of demonstrations is used (e.g., [7, 9, 17]), our method allows the BC model to generate new trajectories from states that have been visited by—but resulted in poor performance from—the current RL policy. If a trajectory generated by the BC model turns out better than what the RL agent originally experienced, this additional data is stored for the RL agent to leverage later. Our procedure lets the agent essentially perform counterfactual reasoning: *how would things have gone if I had followed the BC policy instead of my policy?* Further, while in this paper we emphasis the use of a BC model to generate the additional trajectories because it is easy to obtain, this policy could actually come from *any* source, including a hand-coded agent, a rule-based agent, or another agent trained using RL (as we will show later).

One limitation of the proposed technique is that it requires the ability for the agent to be able to teleport back to previously-visited states and roll forward in time from there. Thus, our method is only suitable for tasks where a simulator is available—either the task itself is a simulation like a video game or we can build a simulator of the real world—and that reset to arbitrary states in the simulator is achievable. Despite this constraint, we carefully account for *all* environment interactions—including steps taken by the BC policy to generate trajectories—and show that our method reduces the overall sample complexity of learning compared to methods that do not leverage a BC policy in this way.

We show experimentally in the game of Ms. Pac-Man that our method outperforms the baseline method in which a BC model is not used, even when the BC model is far from expert. We also conduct additional ablation studies to understand how each component functions in our method. While this work is still in-progress, our method has significant potential given the exciting preliminary results—it suggests a new way of incorporating human knowledge into RL algorithms.

*Corresponding author

2 BACKGROUND

Our algorithm builds upon several existing methods, which we briefly review in this section.

2.1 Asynchronous Advantage Actor-Critic

We consider a reinforcement learning (RL) problem that is modeled using a Markov Decision Process (MDP), represented by a 5-tuple $\langle S, A, P, R, \gamma \rangle$. A *state* s_t represents the environment at time t. An agent learns what *action* $a_t \in \mathcal{A}(s)$ to take in s_t by interacting with the environment. A *reward* $r_{t+1} \in \mathcal{R} \subset \mathbb{R}$ is given based on the action executed and the next state reached, s_{t+1}. The goal is to maximize the expected cumulative return $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$, where $\gamma \in [0, 1]$ is a discount factor that determines the relative importance of future and immediate rewards [24].

Policy-based methods such as the asynchronous advantage actor-critic (A3C) algorithm [14] combine a deep neural network with the actor-critic framework. In this work, we leverage the A3C framework to learn both a *policy function* $\pi(a_t|s_t;\theta)$ (parameterized as θ) and a *value function* $V(s_t;\theta_v)$ (parameterized as θ_v). The policy function is the *actor* that takes action while the value function is the *critic* that evaluates the quality of the action against some baseline (e.g., state value). A3C directly minimizes the policy loss

$$L^{a3c}_{policy} = \nabla_\theta log(\pi(a_t|s_t;\theta))\big(Q^{(n)}(s_t, a_t;\theta,\theta_v) - V(s_t;\theta_v)\big) - \beta^{a3c}\mathcal{H}\nabla_\theta\big(\pi(s_t;\theta)\big)$$

where $Q^{(n)}(s_t, a_t;\theta,\theta_v) = \sum_{k=0}^{n-1}\gamma^k r_{t+k} + \gamma^n V(s_{t+n};\theta_v)$ is the n-step bootstrapped value that is bounded by a hyperparameter t_{max} ($n \leq t_{max}$). $\mathcal{H}$ is an entropy regularizer for policy π (weighted by β^{a3c}) which helps to prevent premature convergence to sub-optimal policies. The value loss is

$$L^{a3c}_{value} = \nabla_{\theta_v}\Big(\big(Q^{(n)}(s_t, a_t;\theta,\theta_v) - V(s_t;\theta_v)\big)^2\Big)$$

The A3C loss given by Mnih et al. [14] is then

$$L^{a3c} = L^{a3c}_{policy} + \alpha L^{a3c}_{value} \tag{1}$$

where α is a weight for the value loss. A3C runs k actor-critic workers in parallel and each with their own copies of the environment and parameters. In this work, we use the feedforward version of A3C for all experiments since it runs faster than the recurrent version while the performance is still comparable Mnih et al. [14]. The architecture consists of three convolutional layers, one fully connected layer, followed by two branches of a fully connected layer: a policy function output layer and a value function output layer. Details on the network architecture can be found in [14].

2.2 Transformed Bellman Operator for A3C

The A3C algorithm uses reward clipping to help stabilize learning. However, Hester et al. [9] found that clipping rewards to $[+1, -1]$ results in the agent being unable to distinguish between small and large rewards, thus hurting the performance in the long-term. Pohlen et al. [17] proposed the *transformed Bellman (TB) operator* to overcome this problem in the deep Q-network (DQN) algorithm [15]. The authors consider reducing the scale of the action-value function while keeping the relative differences between rewards which enables DQN to use raw rewards instead of clipping. Pohlen et al. [17] apply a transform function $h : z \mapsto sign(z)\left(\sqrt{|z|+1} - 1\right) + \varepsilon z$ to reduce the scale of $Q^{(n)}(s_t, a_t;\theta,\theta_v)$ as

$$Q^{(n)}_{TB}(s_t, a_t;\theta,\theta_v) = \sum_{k=0}^{n-1} h\left(\gamma^k r_{t+k} + \gamma^n h^{-1}\left(V\left(s_{t+n};\theta_v\right)\right)\right) \tag{2}$$

Pohlen et al. [17] prove that the TB operator reduces the variance of the optimization goal while still enabling learning an optimal policy. Given this benefit, de la Cruz Jr et al. [8] applied the TB operator to A3C, denoted as *A3CTB*, and found that *A3CTB* outperforms A3C.

2.3 Self Imitation Learning for A3CTB

Oh et al. [16] propose the *self imitation learning* (SIL) algorithm with the intuition that the agent can exploit its own past *good* experiences to drive deeper exploration and improve performance. Built upon the actor-critic framework [14], SIL adds a prioritized experience replay buffer $\mathcal{D} = (S, A, G)$ to store the agent's past experiences. In addition to the A3C update in equation (1), at each step t, SIL samples a minibatch from $\mathcal{D}$ for M time steps and optimizes the following actor-critic loss:

$$L^{sil}_{policy} = -log(\pi(a_t|s_t;\theta))\big(G_t - V(s_t;\theta_v)\big)_+$$
$$L^{sil}_{value} = \frac{1}{2}||\big(G_t - V(s_t;\theta_v)\big)_+||^2$$

where $G_t = \sum_{k=0}^{\infty}\gamma^k r_{t+k+1}$ is the discounted cumulative return, V is the state value, and $(\cdot)_+ = max(\cdot, 0)$ meaning that only experiences with positive advantage can contribute to the policy update. The experience buffer is prioritized by $\big(G_t - V(s_t;\theta_v)\big)_+$ to increase the chance that a *good* experience is sampled. The SIL loss is then

$$L^{sil} = L^{sil}_{policy} + \beta^{sil} L^{sil}_{value} \tag{3}$$

where β^{sil} weigh the value loss. de la Cruz Jr et al. [8] leverage this framework to incorporate SIL in A3CTB, denoted as *A3CTBSIL*, which outperforms both the A3C and A3CTB algorithms. Therefore, in this work we use A3CTBSIL as the baseline.

2.4 Behavior Cloning for A3C

Behavior cloning (BC) is an imitation learning method that reduces an RL task into a supervised learning task [19, 20]. In a standard BC algorithm, a set of demonstration data is first collected; a classifier can then be trained to learn a mapping between the state and the action. Pre-training a BC model for A3C, however, requires a few more steps than just using supervised learning. A3C has two output layers and the policy output is what we usually train a supervised classifier for. But the value output layer is usually initialized randomly without being pre-trained. de la Cruz Jr et al. [8] observe this inconsistency and propose to leverage demonstration data to also pre-train the value output layer. In particular, since the demonstration data contains the true return G, we can obtain a value loss that is almost identical to A3C's value loss L^{a3c}_{value}: instead of using the n-step bootstrap value $Q^{(n)}$ to compute the advantage, the true return G is used.

Inspired by the supervised autoencoder (SAE) framework [11], de la Cruz Jr et al. [8] also blend in an unsupervised loss for pre-training. In SAE, an image reconstruction loss is incorporated with the supervised loss to help extract better feature representations and

achieve better performance. de la Cruz Jr et al. [8] show that a BC model pre-trained jointly with supervised, value, and unsupervised losses can lead to better performance after fine-tuning with RL compared to pre-training with supervised loss only. In this work, we leverage this method to jointly pre-train a BC model and propose a new way of combining BC with A3CTBSIL.

3 CORRECTED SELF IMITATION LEARNING

As reviewed in Section 2.3, the key component of the SIL framework is to use a replay buffer to store past trajectories; the agent can then repeatedly learn from its *good* states (i.e., states with positive advantage value). The motivation for our method is the observation that the *bad* states (i.e., states with negative advantage value) can potentially also be informative. Ignoring these states results in SIL not taking full advantage of exploiting its past experiences.

Given this motivation, we propose in this paper *corrected self imitation learning (CSIL)*, an algorithm that allows an SIL agent to leverage other policies to "correct" the bad experiences by finding possible better trajectories so that information in these states can also be exploited. We leverage *human demonstration data* to generate such a supplemental policy. In particular, we collect a small set of demonstrations from a human and train a behavior cloning (BC) model using the data. This BC model is then used as a "corrector" for an RL agent while it learns. Note that the corrector could actually come from *any* source—here we chose to leverage demonstration data because it is easy to obtain, and also to illustrate that our method works well even when the source is a non-expert. In Section 4 we include an experiment that considers an agent trained using RL as the source to show the scenario when the corrector is an expert, which provides an empirical upper-bound on the performance of CSIL.

From a high level perspective, we expect the corrector to help learning by allowing the agent to witness and learn from alternate and advantageous behaviors. Our method builds upon the *A3C with Transformed Bellman operator and Self Imitation Learning (A3CTBSIL)* algorithm by de la Cruz Jr et al. [8]. We chose this algorithm as our baseline because it provides improved performance over the original A3C and SIL algorithms [14, 16], especially in our evaluation domain, the Atari game of Ms. Pac-Man.[1]

Figure 1 shows the proposed architecture for CSIL. Blue components represent the A3C algorithm in which k parallel workers interact with their own copies of the environment to update the global policy π. Orange components represent the SIL algorithm in which a SIL worker and a prioritized replay buffer $\mathcal{D}$ is added to A3C. Buffer $\mathcal{D}$ stores all experiences from the A3C workers. The SIL worker runs in parallel with the A3C workers but does not interact with the environment; it only takes samples from buffer $\mathcal{D}$ and updates π using samples that have positive advantage values.

We make the following additions to A3C and SIL. First, instead of ignoring the bad states that were not used for an SIL update, we store them in a priority queue. This queue is prioritized by advantage, but in a reverse order to that of buffer $\mathcal{D}$. In contrast to buffer $\mathcal{D}$ where the most positive states are sampled first, the

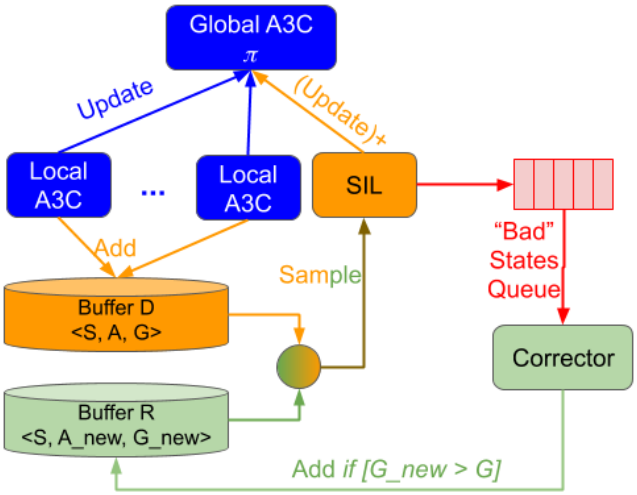

Figure 1: CSIL architecture. The original A3C algorithm is shown in blue: k parallel workers interact with their own copies of the environment and update the global policy π. The SIL algorithm is shown in orange: one SIL worker runs in parallel with A3C and samples from experience buffer $\mathcal{D}$ and update π using only *good* states. We propose to add a corrector worker, shown in green, to generate new experiences from a given *bad* state. The corrector is a behavior cloning model pre-trained using demonstration data as described in Section 2.4. The replay buffer $\mathcal{R}$ stores corrector-generated experiences only if they are better than before.

queue populates the most negative states first. That is, we want to correct states where the current policy performed the worst. Second, we add a corrector worker in parallel with the A3C and SIL workers. Our corrector is a pre-trained BC model that can be queried for actions when provided with states (i.e., a policy). The corrector worker also has access to the environment and takes as input the states from the queue. For each state in the queue, the corrector resets the environment to that state and thereafter uses its policy to perform a rollout until the end of the episode. Third, we use a prioritized buffer $\mathcal{R}$ to store the trajectories generated by the corrector only if the return of the new trajectory G_{new} is better than the previous return G (sampled from buffer $\mathcal{D}$) for a state. $\mathcal{R}$ is prioritized the same way as in $\mathcal{D}$, i.e., positive advantage first. Finally, the SIL worker samples equally from both buffers $\mathcal{D}$ and $\mathcal{R}$ instead of only from $\mathcal{D}$, and performs its updates using *good* experiences only (i.e., experiences with positive advantage values). We summarize the correction procedure of CSIL in Algorithm 1.[2]

CSIL leverages both human demonstration and RL, and it enjoys two main benefits. First, as our experimental results show, CSIL works well with a policy that is not an expert at the task. Second (and more importantly), even though the corrector requires additional interaction with the environment, we show that the speedup in learning it provides actually reduces the overall number of environment interactions required for learning. It seems that the high

[1]Note that A3CTBSIL is the baseline method in de la Cruz Jr et al. [8], which does not leverage demonstrations; this is our baseline here. They also explored combining demonstration pre-training with A3CTBSIL, which we leave for future work.

[2]We omit presenting the A3C and SIL worker procedures due to space limit; the procedures are the same to those in Mnih et al. [14] and Oh et al. [16] respectively.

Algorithm 1 Corrected Self Imitation Learning: Corrector Worker

//Assume a behavior cloning model $f(\cdot;\theta_{BC})$ is available and can interact with the environment
//Shared replay buffer $\mathcal{R}$
//Initialize $A_{new} \leftarrow \emptyset$, $R \leftarrow \emptyset$, $S \leftarrow \emptyset$
//Given a sample $\{s, a, G\}$ from the bad state queue, reset the environment to s
while not terminal **do**
    Query BC for an action: $a \leftarrow f(s;\theta_{BC})$
    Execute a, obtain reward r and next state s'
    //Store the experience
    $A_{new} \leftarrow A_{new} \cup a, R \leftarrow R \cup r, S \leftarrow S \cup s'$
    Go to next state $s \leftarrow s'$
end while
//Compute new return
$G_{\text{new}} = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \forall r \in R$
if $G_{\text{new}} > G$ **then**
    Add to buffer $\mathcal{R} \leftarrow \mathcal{R} \cup \{S, A_{new}, G_{\text{new}}\}$
end if

quality of the corrector's additional experience compensates for the additional *quantity* of policy updates it requires. Our method increases the quality of corrector-generated experience because we only use it when $G_{\text{new}} > G$. We empirically justify the importance of this hand-coded rule via an ablation study in Section 5. Our method reduces the quantity of policy updates (i.e., number of gradient updates) because it replaces one of the A3C workers with the corrector. Since policy updates are only performed by A3C workers, our method performs strictly fewer policy updates than would be performed by A3CTBSIL over a similar time period[3] (see Section 4 for detailed parameter choices). Therefore, our method trades off quantity for quality, achieving better results without increasing the number of environmental steps while still reducing the number of policy updates.

4 EXPERIMENTS

We empirically evaluate CSIL in the Atari game of Ms. Pac-Man [4] by comparing the following techniques:

(1) *A3CTBSIL*: the baseline method from de la Cruz Jr et al. [8]. This method does not leverage any demonstrations; the agent is trained from scratch and its architecture consists only of the blue and the orange components in Figure 1.
(2) *CSIL-BC*: CSIL in which a behavior cloning (BC) model is used as the corrector worker. The BC model only imitates the demonstration data and its performance in Ms. Pac-Man is non-expert.
(3) *CSIL-TA*: CSIL in which a trained agent (TA) is used as the corrector worker. The trained agent performs well in Ms. Pac-Man and we consider it as an expert agent.

Note that our method does not require the corrector to be an expert to improve performance—we include here the CSIL-TA setting to provide an empirical upper-bound on performance of our method when an expert policy is available. We describe how we obtained the BC model and the trained agent later in this section. For all experiments, we use the same network architecture as in the feed-forward version of A3C [14] (as reviewed in Section 2.1). Game images are gray-scaled and resized to 88×88 with 4 images stacked as the input. We train for 50 million environmental steps and for every 1 million steps we perform a test of 125,000 steps and report the testing scores.

[3]This difference can be considerable giving that training takes millions of steps.

Table 1: Demonstration size and quality, collected from de la Cruz et al. [7].

Game	Worst score	Best score	# of states	# of episodes
Ms. Pac-Man	4020	18241	14504	8

The baseline A3CTBSIL is trained with 17 parallel workers; 16 A3C workers and 1 SIL worker. The RMSProp optimizer is used with learning rate = 0.0007. We use $t_{max} = 20$ for n-step bootstrap ($n \leq t_{max}$). The SIL worker performs $M = 4$ SIL policy updates per step t with minibatch size 32 (i.e., 32×4=128 total samples per step). The size of the buffer $\mathcal{D}$ is 10^5, and $\beta^{sil} = 0.5$.

For CSIL-BC, the BC model is obtained by pre-training with demonstration data collected by de la Cruz et al. [7].[4] The data statistics are shown in Table 1. As described in Section 2.4, we follow de la Cruz Jr et al. [8] and jointly pre-train with supervised, value, and unsupervised autoencoder losses for 50,000 steps and minibatch size 32. Adam optimizer is used with learning rate = 0.0005. For CSIL-TA, the trained agent is obtained by first transferring the weights of the pre-trained BC model to the agent (instead of initializing it randomly), and then continuing to train it using the A3CTBSIL algorithm (with the same parameters as in the baseline). Both CSIL settings are trained with 17 parallel workers: 15 A3C workers, 1 SIL worker, and 1 corrector worker—we keep the total number of workers in A3CTBSIL and CSIL the same to ensure a fair performance comparison. The SIL worker in CSIL also uses minibatch size of 32 as in A3CTBSIL, where 16 samples are taken from buffer $\mathcal{D}$ and 16 are from $\mathcal{R}$.

Figure 2 shows the results of CSIL compared with A3CTBSIL (averaged over 3 trials). As expected, CSIL-TA's performance is significantly higher than A3CTBSIL's when using a more expert agent as the corrector. We however note that CSIL-TA does not outperform the trained agent (whose performance is shown as the purple dashed line[5]). Our method underperforms the trained agent could be because CSIL does not follow the corrector *everywhere*, which would be the right thing to do if the corrector were optimal. Instead, CSIL allows the learning agent to explore on its own, which is necessary when the corrector is sub-optimal.

The more interesting result is the performance of CSIL-BC, which demonstrates that our method works well even when using a corrector that is far from expert. In particular, the black dashed line shows the average performance level of the non-expert BC model, which is roughly 2,000 points (estimated by executing the BC model in Ms. Pac-Man for 125,000 steps). By incorporating the corrector, the CSIL agent can learn to outperform BC quickly and achieve better results compared to the baseline. In addition, it is worth noting

[4]The dataset is publicly available at github.com/gabrieledcjr/atari_human_demo
[5]Note here the expert score shown is the *best* testing score of the agent. That is, take the best checkpoint of the trained agent, execute it in the environment for 125,000 steps, and record the highest score it achieved.

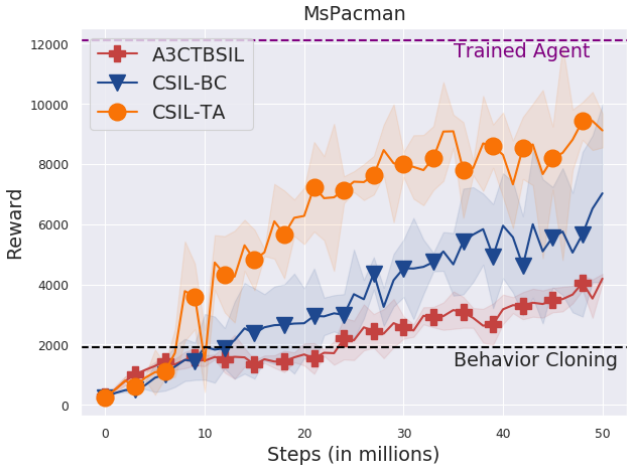

Figure 2: Ms. Pac-Man performance. The x-axis is the total number of environmental steps. For A3CTBSIL, steps are counted in 16 A3C workers. For CSIL, steps are counted in 15 A3C workers plus 1 corrector worker. The y-axis is the average testing score over three trials; shaded regions are the standard deviation. The dashed black line shows the average score of the BC corrector, estimated by executing the BC model in the game for 125,000 steps. The dashed purple line shows the *best* testing score the trained agent achieved.

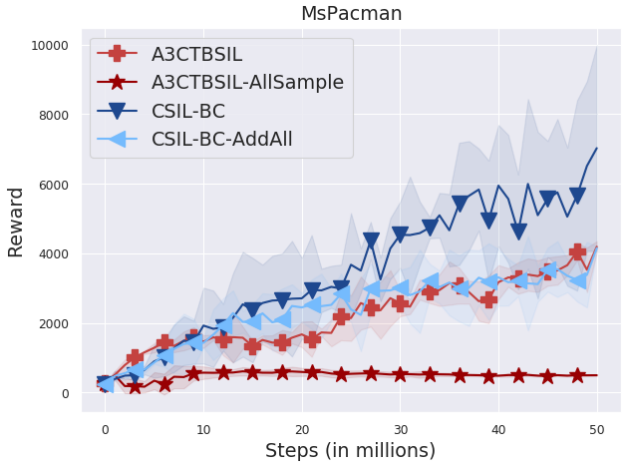

Figure 3: Ablation study. The x-axis is the total number of environmental steps. For A3CTBSIL-based method, steps are counted for 16 A3C workers. For CSIL, steps are counted for 15 A3C workers plus 1 corrector worker. The y-axis is the average testing score over three trials; shaded regions are the standard deviation.

that our method achieved better results with fewer policy updates (i.e., CSIL uses one fewer A3C worker). This observation shows that data generated by the corrector can be efficiently leveraged by the policy and thus compensate for the reduced quantity of updates. We highlight this observation in particular since it shows significant potential for learning from imperfect demonstrations.

5 ABLATION STUDY

We have shown that our method can effectively leverage knowledge from different sources (a BC model and a trained agent) and achieve better results in the Atari game of Ms. Pac-Man compared to the baseline. In this section, we perform two ablation studies to further validate our design choices.

First, the primary motivation of our method is that we hypothesize that the original SIL algorithm does not take full advantage of past experiences and that bad experiences should also be used *after correction*. Therefore, we empirically justify the need for correction by conducting an experiment in the original SIL framework in which *all* samples (even bad ones) are used for policy updates instead of only the good ones.[6] We denote this experiment as *A3CTBSIL-AllSample*.

Second, we show that it is important to store the corrector-generated experiences only if those experiences are better, i.e., if the return G_{new} computed from the corrector experience is higher than the return G that the agent previously obtained (shown as the green arrow in Figure 1). Behavior cloning suffers from distribution shift and performs poorly on out-of-sample data [19]. This phenomenon is particularly true in our CSIL-BC setting where the BC model is trained with a small set of demonstration data—states encountered by the demonstrator are far from covering all possible states in Ms. Pac-Man. However in practice, it is possible that the BC model is knowledgeable in some states where the agent has not yet learned. Intuitively, our method seeks this type of situation and instead of letting the agent imitate the demonstrator all the time, the demonstrator should only provide help *where it can*. To validate this hypothesis, we conduct an experiment in which the corrector adds *all* trajectories to buffer $\mathcal{R}$ to simulate the situation when the imperfect BC model is trying to help everywhere—this leads to decreased performance. We denote this experiment as *CSIL-BC-AddAll*.

The results shown in Figure 3 verify that both of our hypotheses are true, i.e., our design choices are well-founded. In A3CTBSIL-AllSample, the agent barely learns anything despite that there is only one SIL worker updating the global policy among 16 A3C workers. Directly leveraging *bad* experiences without correction badly hurts performance, which verifies the assumption of the original SIL framework that only good experiences should be imitated [16]. The CSIL-BC-AddAll result shows a slight improvement over the baseline A3CTBSIL during early stages of training but then plateaus lower than CSIL-BC, indicating that the non-expert BC model is still helpful but only in a limited state space. Therefore, it is important to identify where it can help by filtering out samples that are out-of-distribution, only exploiting places where the BC model can do better than what the agent had experienced.

6 RELATED WORK

Our work is related to imitation learning (IL) via behavior cloning (BC) [3, 18–20]. One of the most popular BC algorithms, DAgger,

[6] This setting was not explored in the original SIL paper. Oh et al. [16] showed the agent should imitate past good experiences but did not study the effect of imitating bad experiences.

assumes access to an expert demonstrator throughout training. The imitator queries the expert for new data at every iteration and retrains a BC model using aggregated data [20]. Our method is similar to DAgger in that we also consider using a behavior cloning model for generating new data (which we call the corrector), but is different in two respects. First, we don't require the corrector to be an expert (as shown in Figure 2). And second, DAgger-like IL frameworks can't learn to outperform the demonstrator, while in our method, the corrector is combined with RL to allow the agent to also learn from environment interactions, thus enabling it to surpass the (non-expert) demonstrator's performance.

More recent research combines imitation learning in RL to assist exploration in complex environments. The most straightforward approach is to pre-train an initial policy using demonstration data and then fine-tune with RL algorithms [2, 7, 8, 21, 22]. One can also add an imitation loss (usually a classification loss) to the RL loss and minimize both during training. The deep Q-learning from demonstrations (DQfD) algorithm falls into this category [9]. In DQfD, a prioritized replay buffer is initialized with expert demonstration data and new trajectories generated by the agent during training is added to the buffer without overwriting the demonstration data. The agent jointly minimizes an RL temporal difference loss with a large margin supervised loss using a batch of samples from the buffer. Similarly in our method, we use a separate buffer to store experiences generated by a demonstration model. The difference is that we train on a single RL loss instead of a joint loss. We also sample from the demonstrator's buffer explicitly and leverage its knowledge where helpful, while DQfD mixes all experiences together and samples based on TD error priority. We point out that, although we have not yet performed a complete set of comparisons to DQfD, the final score of our method at around 6,000 is already higher than the score reported for DQfD for Ms. Pac-Man, which is 4,695.7. Ape-X DQfD [17] improves upon DQfD in which the transformed Bellman (TB) operator was proposed. The baseline A3CTBSIL we used in this work also leverages the TB operator.

Another paradigm for leveraging human knowledge for RL is through advice [5]. The assumption is that an RL agent has access to an existing "advisor" policy; this policy can either come from a human or another agent trained from a different source. During training, the learner can query the advisor for explicit action advice, which helps improve the sample efficiency of the learner [1, 25, 26]. One question in this type of approach is that the advised action might not always be helpful. The confidence-based human agent transfer (CHAT) algorithm [27] copes with this problem. Instead of following the advise all the time, CHAT executes the suggested action only when the suggestion's confidence is higher than a threshold. The mechanism of our corrector worker is similar in that we store its trajectory only when it obtains a higher return than before. Another problem when leveraging advice is that advisor might not always be available, or the communication could be costly, requiring the agent to minimize the amount of advice requested. Da Silva et al. [6] proposed uncertainty-aware advice that is suitable when the advice budget is limited. Instead of querying at every step, the learner asks for advice only when it is uncertain about what to do; the uncertainty is measured by the variance of the learner's Q-values. Our method is different in that we don't limit the number of times the corrector can perform a rollout—it produces as many more new samples as it can within training time.

7 DISCUSSION AND FUTURE WORK

We proposed *corrected self imitation learning (CSIL)* in this paper. First, we identified one disadvantage in the SIL framework in that it does not take full advantage of exploiting its past experiences; only *good* states are used while *bad* states are ignored. We then added a pre-trained behavior cloning model, which we called a *corrector* worker, to the SIL framework to explore possible better trajectories for each *bad* state. Information in previously *bad* states can then be fully leveraged after the correction. Meanwhile, one limitation of our method is that it is only suitable in situations in which teleporting the agent to any previously seen state is possible, and in which the agent can take steps forward from there.

While we have shown preliminary success, more study needs to be done to further improve our method. First, we have only evaluated our method in the game of Ms. Pac-Man with three seeds. An immediate next step is to perform experiments in more games to test the robustness of our method. We will also try to understand the underlying behaviour of the corrector. For example, we chose to store only experiences generated by the corrector that are better than the initial trajectories, given the intuition that a behavior cloning model can only perform well for in-sample states. As the agent learns a better policy over-time and explores a larger state space, the possibility of the BC model seeing an out-of-sample state increases. Continuing to perform the rollout would reduce the efficiency of the corrector since it becomes less likely to generate better samples. An interesting direction is to quantitatively measure how helpful the corrector is during the course of training, stopping the corrector when it is no longer helpful, and allowing the agent to learn on is own; this is similar to the DAgger algorithm where the agent samples less from the expert policy over time. We argue that the sampling schedule should not be purely based on time steps, but should also consider the helpfulness of the demonstrator model.

We have not yet compared our method with other RL algorithms that leverage demonstrations. CSIL should be compared with existing methods to understand when it works better/worse than the other methods. For example, an alternative and straightforward appraoch is to directly transfer the weights of the pre-trained BC model to an agent and thereafter fine-tune using RL [7, 8], as we have done to obtain the trained agent in the CSIL-TA setting. While CSIL-TA did not reach the same level of performance as the trained agent (as discussed briefly in Section 4), it will be interesting to explore if combining our method with weight transfer could further improve performance. That is, we are curious whether keeping the BC model in a separate worker even after the weight transfer can still help the agent.

Lastly, we plan to extend our method into incorporating multiple policies. Currently, our architecture adds one corrector worker along with other workers, and the corrector can leverage an existing policy (that comes from a BC model or a trained agent). It is therefor natural to consider adding multiple corrector workers, each with its own policy that contains knowledge on different state spaces. The agent can then compare and sample from the best policy.

ACKNOWLEDGMENTS

We thank Gabriel V. de la Cruz Jr. for helpful discussions; his open-source code at https://github.com/gabrieledcjr/DeepRL is used for training the corrector models in this work. This research used resources of Kamiak, Washington State University's high performance computing cluster. This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (CPS-1739964, IIS-1724157, NRI-1925082), ONR (N00014-18-2243), FLI (RFP2-000), ARL, DARPA, Lockheed Martin, GM, and Bosch. Peter Stone serves as the Executive Director of Sony AI America and receives financial compensation for this work. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research.

REFERENCES

[1] Ofra Amir, Ece Kamar, Andrey Kolobov, and Barbara J. Grosz. 2016. Interactive Teaching Strategies for Agent Training. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI'16)*. AAAI Press, 804–811.

[2] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. 2009. A survey of robot learning from demonstration. *Robotics and autonomous systems* 57, 5 (2009), 469–483.

[3] Michael Bain and Claude Sammut. 1999. A Framework for Behavioural Cloning. In *Machine Intelligence 15, Intelligent Agents [St. Catherine's College, Oxford, July 1995]*. Oxford University, GBR, 103–129.

[4] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47 (2013), 253–279.

[5] Felipe Leno Da Silva and Anna Helena Reali Costa. 2019. A survey on transfer learning for multiagent reinforcement learning systems. *Journal of Artificial Intelligence Research* 64 (2019), 645–703.

[6] Felipe Leno Da Silva, Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. 2020. Uncertainty-Aware Action Advising for Deep Reinforcement Learning Agents. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*.

[7] Gabriel V de la Cruz, Yunshu Du, and Matthew E Taylor. 2019. Pre-training with non-expert human demonstration for deep reinforcement learning. *The Knowledge Engineering Review* 34 (2019).

[8] Gabriel V de la Cruz Jr, Yunshu Du, and Matthew E Taylor. 2019. Jointly Pre-training with Supervised, Autoencoder, and Value Losses for Deep Reinforcement Learning. *arXiv preprint arXiv:1904.02206* (2019).

[9] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. 2018. Deep Q-learning from demonstrations. In *AAAI*.

[10] Niels Justesen, Philip Bontrager, Julian Togelius, and Sebastian Risi. 2019. Deep learning for video game playing. *IEEE Transactions on Games* (2019).

[11] Lei Le, Andrew Patterson, and Martha White. 2018. Supervised autoencoders: Improving generalization performance with unsupervised regularizers. In *Advances in Neural Information Processing Systems*. 107–117.

[12] Yuxi Li. 2018. Deep reinforcement learning. *arXiv preprint arXiv:1810.06339* (2018).

[13] Long-Ji Lin. 1992. *Reinforcement learning for robots using neural networks*. Ph.D. Dissertation.

[14] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *ICML*. 1928–1937.

[15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[16] Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. 2018. Self-imitation learning. In *ICML*.

[17] Tobias Pohlen, Bilal Piot, Todd Hester, Mohammad Gheshlaghi Azar, Dan Horgan, David Budden, Gabriel Barth-Maron, Hado van Hasselt, John Quan, Mel Večerík, et al. 2018. Observe and look further: Achieving consistent performance on Atari. *arXiv preprint arXiv:1805.11593* (2018).

[18] Dean A Pomerleau. 1989. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*. 305–313.

[19] Stéphane Ross and Drew Bagnell. 2010. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 661–668.

[20] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. 2011. 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*. 627–635.

[21] Stefan Schaal. 1997. Learning from demonstration. In *Advances in neural information processing systems*. 1040–1046.

[22] 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. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.

[23] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362, 6419 (2018), 1140–1144.

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

[25] Matthew E. Taylor, Nicholas Carboni, Anestis Fachantidis, Ioannis Vlahavas, and Lisa Torrey. 2014. Reinforcement Learning Agents Providing Advice in Complex Video Games. *Connect. Sci* 26, 1 (Jan. 2014), 45–63.

[26] Lisa Torrey and Matthew Taylor. 2013. Teaching on a Budget: Agents Advising Agents in Reinforcement Learning. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS '13)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 1053–1060.

[27] Zhaodong Wang and Matthew E. Taylor. 2017. Improving Reinforcement Learning with Confidence-Based Demonstrations. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. 3027–3033. https://doi.org/10.24963/ijcai.2017/422